National College of Ireland

Configuration Manual

Liver Tumor Segmentation

Using Deep Learning

ALBIN BABY

Student ID: x23173742

School of Computing

National College of Ireland

Supervisor: Mr. Hicham Rifai

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ALBIN BABY

Student ID: x23173742

Programme: M.sc Data Analytics **Year:** 2024

Module: Research In computing

Lecturer: Mr. Hicham Rifa

Submission Due Date: 12-12-2024

Project Title: Liver Tumour Segmentation Using Deep Learning

Word Count: 2422 **Page Count:** 15

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ALBIN BABY

Date: 12-12-2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

ALBIN BABY

Student ID: x23173742

1 Introduction

The presented configuration manual showcases the demonstration of the hardware and software used with detailed implementation and application for the **Liver Tumor Segmentation with UNET with Transfer Learning techniques – RESNET**.

2 System Configuration

2.1 Hardware Configuration

The hardware configuration manual is illustrated by the Table 1 Hardware Config with a 12-core system and 12 GB RAM.

Table 1 Hardware Config

Component	*Specification*
Processor	Intel Core i7-12700K (12 Cores, 20 Threads, 3.6 GHz Base, 5.0 GHz Turbo)
Graphics Card	NVIDIA RTX 3080 Ti (12 GB GDDR6X)
RAM	32 GB DDR5 (3200 MHz)
Storage	1 TB NVMe SSD, 2 TB HDD
Operating System	Windows 11 Pro
Motherboard	ASUS ROG Strix Z690-E Gaming
Power Supply Unit	850W Gold Certified PSU
Cooling System	Liquid Cooler

2.2 Software Configuration

The languages employed in this research have been well chosen to provide an enhanced development and assessment of the model. Based on the above discussion, this analysis has been performed using Jupyter Notebook software. The main programming language has been Python, as it has a lot of support from available libraries for machine learning and deep learning. Several core libraries including TensorFlow, Keras, and PyTorch have been used for performing the actual implementation of the U-Net, ResNet and V-Net models. Based on these frameworks, it has been easy to build, train, and assess their models due to the many features put in place. In order to manipulate data manipulation, particularly data cleaning and preparation, tools like NumPy, Pandas, and OpenCV have been used to process medical images. Based on this, libraries such as Matplotlib and Seaborn have been applied for visualizing the results of the model performance and the segmentation. These libraries and tools facilitated the processing, optimization and analysis of liver tumor segmentation tasks undertaken as part of this project.

The software version is further illustrated in the Table 2 Software Configuration

Table 2 Software Configuration

LIBRARY	VERSION
PYTHON	3.9.7 or higher
NUMPY	1.23.5
PANDAS	1.4.4
MATPLOTLIB	3.5.3
PLOTLY	5.14.0
OPENCV (CV2)	4.8.0
TENSORFLOW	2.14.0
SCIKIT-LEARN	1.3.1
PILLOW (PIL)	9.3.0
IMAGEIO	2.31.1
IMGAUG	0.4.0
MATPLOTLIB-PYPLOT	3.5.3
NIBABEL	5.1.0
TQDM	4.66.0
PYDICOM	2.4.1

3 Project Development

The methods and approach used in this study have been aligned with applying and assessing best practice deep learning algorithms for liver tumour segmentation. Three of the most successful architectures, namely U-Net, ResNet and V-Net have been selected for this work due to their proven performance on the task of medical image segmentation. Based on this edge and efficiency in terms of spatial features, a pre-trained encoder-decoder-based U-Net has been applied whereas deep residual connections of ResNet enhanced the computational power for handling high-dimensioned and complex data (Bezabih et al. 2024). This V-Net has been included because it uses 3D convolutional architecture which is effective for volumetric medical images. These models have been built, trained and also validated employing the Kaggle dataset to gauge their efficiency relative to conventional segmentation metrics like the DSC and the IoU.

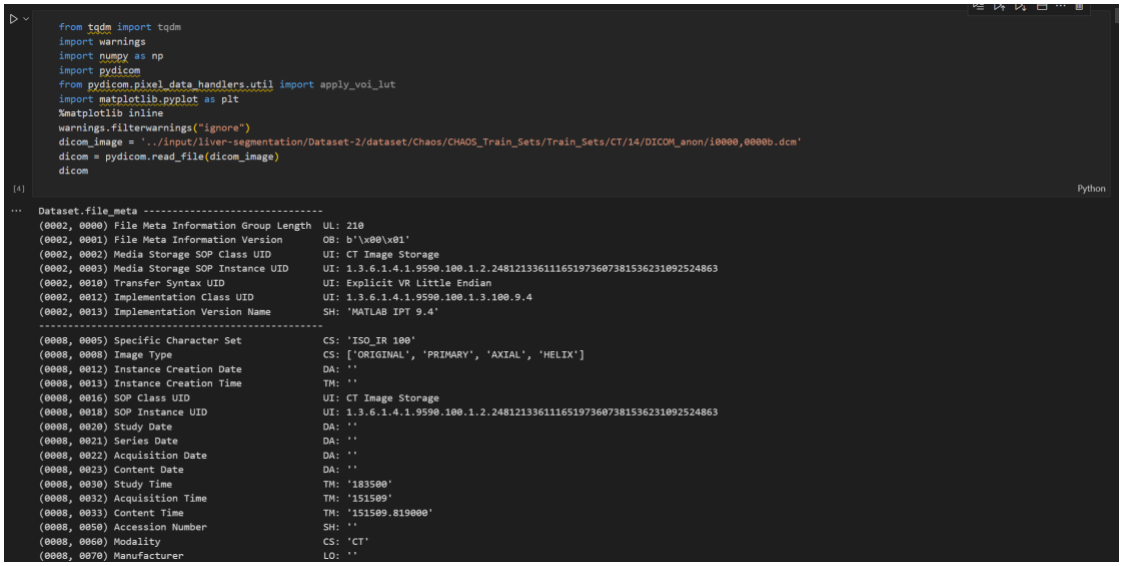
The code is written in Jupyter and Python primarily as discussed in the previous section. The project development section defines the steps of processing.

3.1 Loading Data

Loading data defines the process and path of the data processing pipeline. It is divided into two sections notably 3.1.1 or Loading Single Image and 3.1.2 or Loaded Dataset

3.1.1 Loading Single Image

The Loading of a single image is a test precheck for loading all the images, thus its very useful to detect any path or OS related issues.
In this step we load the libraries and open the dicom image with the help of pydicom package.
Figure 1 Loading Dicom File presented shows a piece of Python code and the result that is given. The code runs some essential libraries required for image processing the code also displays how to read a DICOM image file. The result provides the type of file, the date of creation and other specifications of the specific file, for instance DICOM. This means that the code is likely to be used in a medical context involving images, maybe in processing or analysing the images.

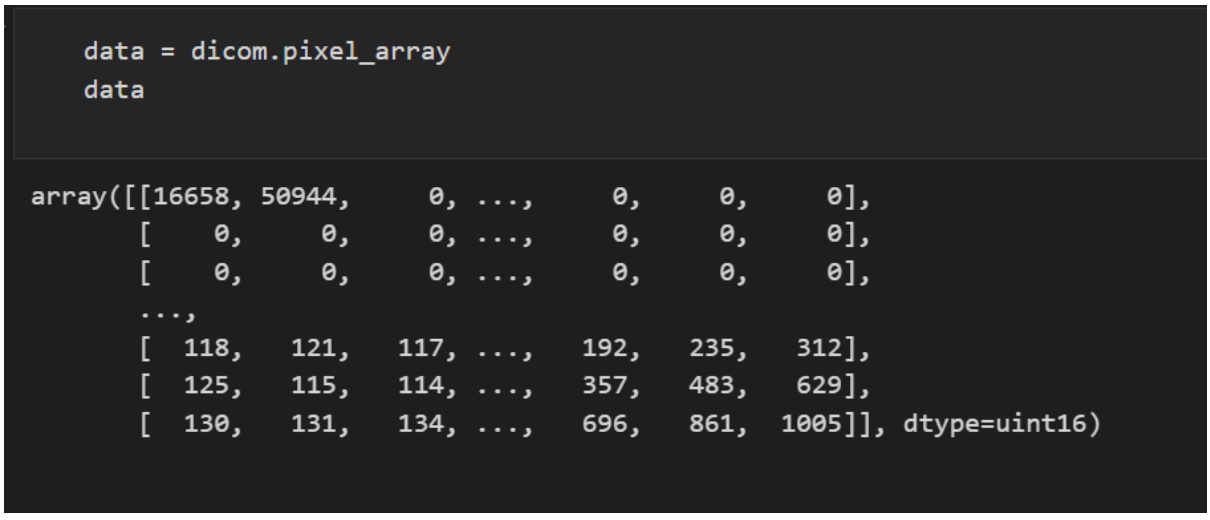


```
from tqdm import tqdm
import warnings
import numpy as np
import pydicom
from pydicom.pixel_data_handlers.util import apply_voi_lut
import matplotlib.pyplot as plt
%matplotlib inline
warnings.filterwarnings("ignore")
dicom_image = '../input/liver-segmentation/Dataset-2/dataset/Chaos/CHAOS_Train_Sets/Train_Sets/CT/14/DICOM_anon/i0000,0000b.dcm'
dicom = pydicom.read_file(dicom_image)
dicom
```

```
Dataset.file_meta -------------------------------
(0002, 0000) File Meta Information Group Length  UL: 210
(0002, 0001) File Meta Information Version       OB: b'\x00\x01'
(0002, 0002) Media Storage SOP Class UID         UI: CT Image Storage
(0002, 0003) Media Storage SOP Instance UID      UI: 1.3.6.1.4.1.9590.100.1.2.248121336111651973607381536231092524863
(0002, 0010) Transfer Syntax UID                 UI: Explicit VR Little Endian
(0002, 0012) Implementation Class UID            UI: 1.3.6.1.4.1.9590.100.1.3.100.9.4
(0002, 0013) Implementation Version Name         SH: 'MATLAB IPT 9.4'
-------------------------------------------------
(0008, 0005) Specific Character Set              CS: 'ISO_IR 100'
(0008, 0008) Image Type                          CS: ['ORIGINAL', 'PRIMARY', 'AXIAL', 'HELIX']
(0008, 0012) Instance Creation Date              DA: ''
(0008, 0013) Instance Creation Time              TM: ''
(0008, 0016) SOP Class UID                       UI: CT Image Storage
(0008, 0018) SOP Instance UID                    UI: 1.3.6.1.4.1.9590.100.1.2.248121336111651973607381536231092524863
(0008, 0020) Study Date                          DA: ''
(0008, 0021) Series Date                         DA: ''
(0008, 0022) Acquisition Date                    DA: ''
(0008, 0023) Content Date                        DA: ''
(0008, 0030) Study Time                          TM: '183500'
(0008, 0032) Acquisition Time                    TM: '151509'
(0008, 0033) Content Time                        TM: '151509.819000'
(0008, 0050) Accession Number                    SH: ''
(0008, 0060) Modality                            CS: 'CT'
(0008, 0070) Manufacturer                        LO: ''
```

Figure 1 Loading Dicom File

The dicom image is a multitude Json based key value pair data field as the field consists of multiple notation units it becomes difficult to capture the actual imagery in question. Thus, its important to glance through the dicom key vector as shown by the Figure 1 Loading Dicom File. After analyzing we find out that the pixel array key filed hold the data input, thus we extract the image information from the pixel_array unit as shown in the Figure 2 Loading image pixel array.



```
data = dicom.pixel_array
data
```

```
array([[16658, 50944,     0, ...,     0,     0,     0],
       [    0,     0,     0, ...,     0,     0,     0],
       [    0,     0,     0, ...,     0,     0,     0],
       ...,
       [  118,   121,   117, ...,   192,   235,   312],
       [  125,   115,   114, ...,   357,   483,   629],
       [  130,   131,   134, ...,   696,   861,  1005]], dtype=uint16)
```

Figure 2 Loading image pixel array

Figure 3 Loaded Image showcases the loaded data point in after the data being captured from the Figure 1 Loading Dicom File and Figure 2 Loading image pixel array

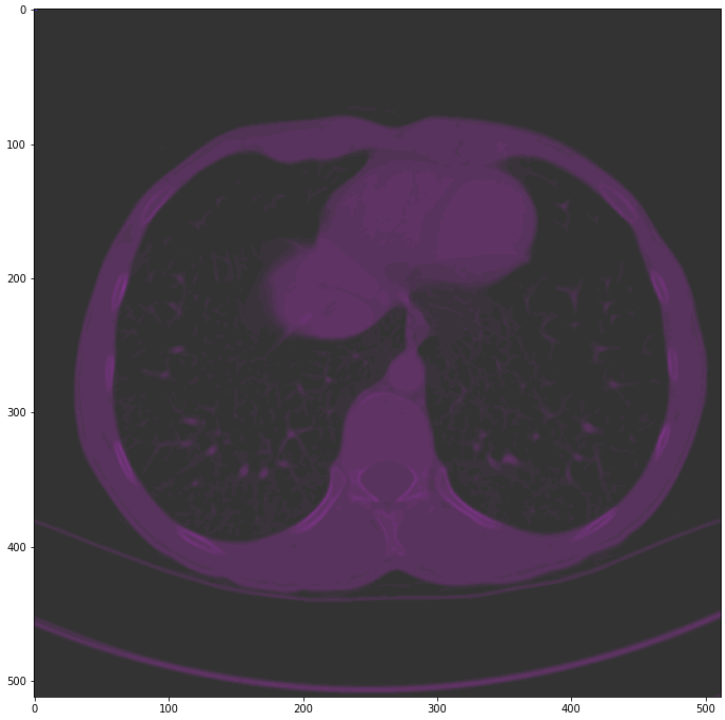


Figure 3 Loaded Image

3.1.2 Loaded Dataset

We orchestrate the whole path simulation from the train directory within the dicom images and store them as a dataset with Image and Mask attribute. This helps in maintaining the files read and unread during actual loading of data.

```
DIR = '../input/liver-segmentation/Dataset-2/dataset/Chaos/CHAOS_Train_Sets/Train_Sets/CT/14/'
df = pd.DataFrame()
for i in os.listdir(DIR+'DICOM_anon/'):
    num = i[3:5]
    df=  df.append({
        'Image':DIR+'DICOM_anon/'+i,
        'Mask':DIR+'Ground/liver_GT_0'+num+'.png'
    },ignore_index=True)
df.head()
```

	Image	Mask
0	../input/liver-segmentation/Dataset-2/dataset/...	../input/liver-segmentation/Dataset-2/dataset/...
1	../input/liver-segmentation/Dataset-2/dataset/...	../input/liver-segmentation/Dataset-2/dataset/...
2	../input/liver-segmentation/Dataset-2/dataset/...	../input/liver-segmentation/Dataset-2/dataset/...
3	../input/liver-segmentation/Dataset-2/dataset/...	../input/liver-segmentation/Dataset-2/dataset/...
4	../input/liver-segmentation/Dataset-2/dataset/...	../input/liver-segmentation/Dataset-2/dataset/...

Figure 4 Loaded Dataset Image and Mask

Figure 4 Loaded Dataset Image and Mask shown is of a bit of a Python programming code followed by its result. The keyword arguments specify a directory path and then create a new Data Frame then for loop through the images in the directory appends the path to the image itself as well as the mask. The output gives the first few rows of the Data Frame to observe the image and the path of mask for each of the entries. This can be used for organizing and storing medical images for analysis or for any other purposes as the data is prerequisite for such a job.

Finally we load all the images and extract the tumor and liver masks form each of it, this is a time consuming and computationally heavy step thus after the step is performed we visualize the images to confirm the quality of the samples loaded in there raw format. The Figure 5 Loading Liver and Tumor demonstrates the input CT scan, tumor mask and predicted overlap for the same sample.

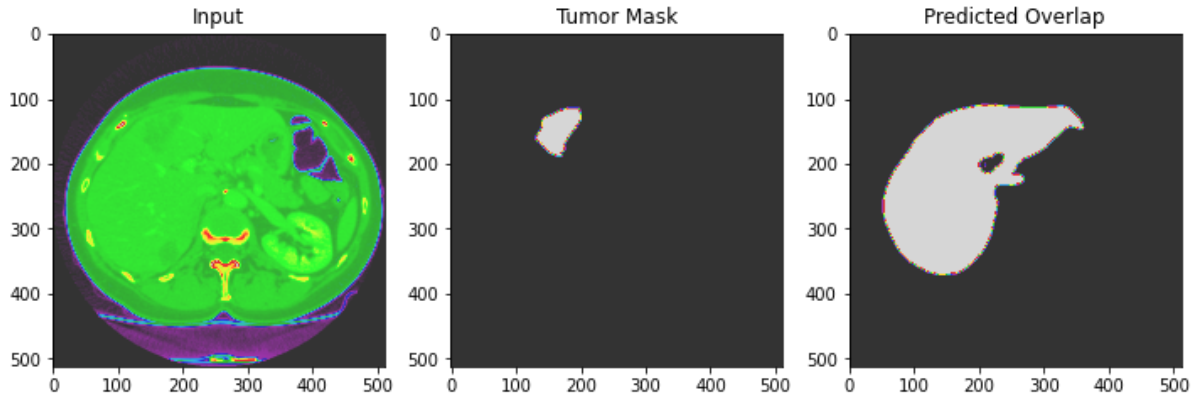


Figure 5 Loading Liver and Tumor

3.2 Normalisation of Data

The normalisation framework enables the picture dataset, to be reduced between pixels of 0 and 1. This further highlights the need to minimise overfitting as well as the decrease of data channels to reduce processing. The liver CT-Scan however is an artificially made picture but owing to the existence of human mistake, mechanical noise and instrumental difference among multiple universities and hospitals involved the dataset is catered with the presence of noise uniformly throughout numerous samples. To prevent this the study proposes the use of bilateral filtering to avoid noise inside the pictures.

Figure 6 Data loading and Transformation allows the loading of dicom image, normalises it and reshapes as per the effective shape and size.

```
X = []
Y = []
for i in df.index:
    dicom_image = df.iloc[i,0]
    dicom = pydicom.read_file(dicom_image)
    data = dicom.pixel_array
    X.append(data)
    img = Image.open(df.iloc[i,1])
    numpydata = asarray(img)
    numpydata = numpydata.astype('int')
    Y.append(numpydata)
X = np.array(X)
Y =np.array(Y)
X.shape, Y.shape
```

```
((95, 512, 512), (95, 512, 512))
```

```
print(X.shape , Y.shape)
```

```
(95, 512, 512) (95, 512, 512)
```

```
X = X.reshape(X.shape[0],X.shape[1],X.shape[2],1)
Y = Y.reshape(Y.shape[0],Y.shape[1],Y.shape[2],1)
X = np.float32(X)
Y = np.float32(Y)
```

Figure 6 Data loading and Transformation

3.3 Split train and test

The code illustrated in Figure 7 Train and Test Split is a methodical way to separating a dataset into training, validation, and testing subsets. Initially, the dataset `X` and its related labels `Y_dash` are separated into training and testing sets, with 20% of the data kept for testing, ensuring that the model is tested on a different set of unseen data. To further optimise the training process, the training set is further separated into training and validation subsets, assigning 15% of the training data for validation. This guarantees that the model may be tuned on the training set while its performance is evaluated on a distinct subset before final testing. The usage of a fixed `random_state=42` assures repeatability of the splits. This strategy enables balanced and successful model building by preserving adequate data distribution across all subsets.

```
from sklearn.model_selection import train_test_split
X = np.float32(X)
X = np.resize(X,(X.shape[0],dim,dim,1))
Y_dash = np.float32(Y)
Y_dash = np.resize(Y_dash,(Y_dash.shape[0],dim,dim,1))
X_train, X_test, y_train, y_test = train_test_split(X, Y_dash, test_size=0.20, random_state=42)
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.15, random_state=42)
X_train.shape,X_valid.shape, X_test.shape, y_train.shape, y_valid.shape, y_test.shape
```

```
((64, 512, 512, 1),
 (12, 512, 512, 1),
 (19, 512, 512, 1),
 (64, 512, 512, 1),
 (12, 512, 512, 1),
 (19, 512, 512, 1))
```

Figure 7 Train and Test Split

4 Model Application

The Figure 8 Model training and validation cycle for 100 epochs shows Python code to call the different libraries required for image segmentation. It establishes hyperparameters of an image, including dimensions, size of a batch and the rate of learning. The ResNet34 is used as a backbone for the U-Net model, and the loss function considered is Dice loss and Focal loss. The model with Adam optimizer and the calculated loss function and metrics for training and testing.

```
from sklearn.model_selection import train_test_split
X = np.float32(X)
X = np.resize(X,(X.shape[0],dim,dim,1))
Y_dash = np.float32(Y)
Y_dash = np.resize(Y_dash,(Y_dash.shape[0],dim,dim,1))
X_train, X_test, y_train, y_test = train_test_split(X, Y_dash, test_size=0.20, random_state=42)
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.15, random_state=42)
X_train.shape,X_valid.shape, X_test.shape, y_train.shape, y_valid.shape, y_test.shape
```

```
((64, 512, 512, 1),
 (12, 512, 512, 1),
 (19, 512, 512, 1),
 (64, 512, 512, 1),
 (12, 512, 512, 1),
 (19, 512, 512, 1))
```

```
X.shape
```

```
(95, 512, 512, 1)
```

```
history =  model.fit(
   x=X_train,
   y=y_train,
   batch_size=16,
   epochs=100,
    validation_data=(X_valid, y_valid)
)
```

```
Epoch 1/100
4/4 [==============================] - 34s 2s/step - loss: 0.7924 - iou_score: 0.0728 - f1-score: 0.1337 - val_loss: 1.8345 -
val_iou_score: 0.0903 - val_f1-score: 0.1656
Epoch 2/100
4/4 [==============================] - 3s 891ms/step - loss: 0.7233 - iou_score: 0.1230 - f1-score: 0.2188 - val_loss: 1.8226
- val_iou_score: 0.0906 - val_f1-score: 0.1661
Epoch 3/100
4/4 [==============================] - 3s 893ms/step - loss: 0.6998 - iou_score: 0.1582 - f1-score: 0.2719 - val_loss: 1.8071
- val_iou_score: 0.0917 - val_f1-score: 0.1680
Epoch 4/100
4/4 [==============================] - 4s 895ms/step - loss: 0.6809 - iou_score: 0.2062 - f1-score: 0.3419 - val_loss: 1.7533
```

Figure 8 Model training and validation cycle for 100 epochs

Figure 9 Training and validation plots for the UNET TLT RESNET showcases the python matplotlib plot for the training and validation plot across 100 epochs.

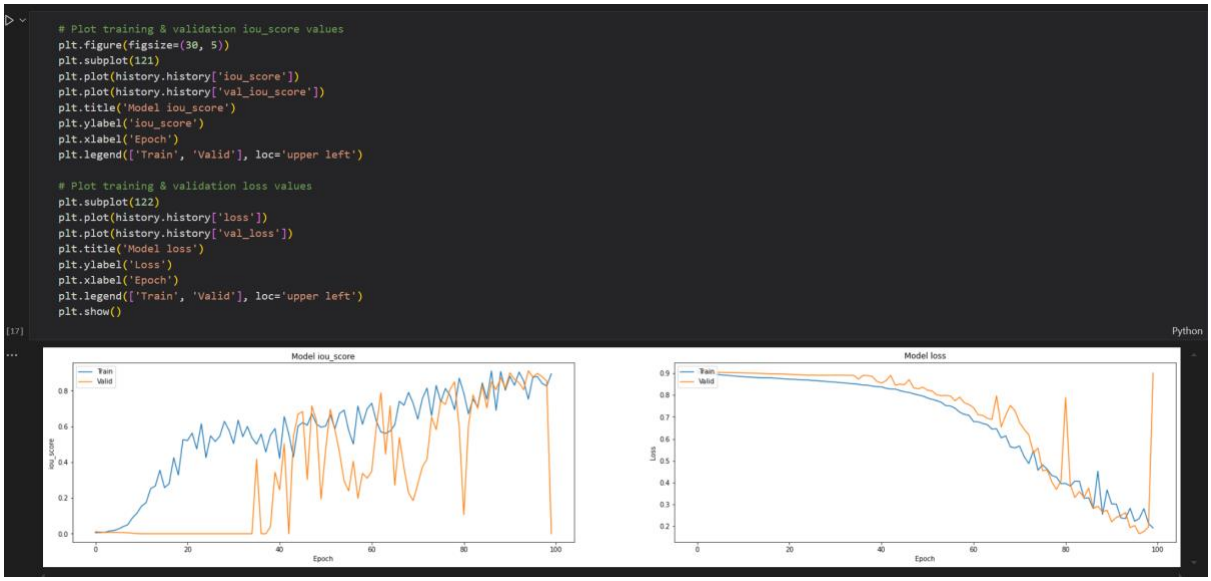


Figure 9 Training and validation plots for the UNET TLT RESNET

Figure 10 Performance Metrics for UNET TLT RESNET showcases the results for the training, testing and validation plots in a tabular plot across loss, accuracy, binary accuracy, mAP, false negatives, false positives, true negatives, true positives, auc, Specificity and Sensitivity.

```
df_res = pd.DataFrame()

df_res =df_res.append(Metrics(X_train,y_train),
                      ignore_index=True)
df_res =df_res.append(Metrics(X_test, y_test),
                      ignore_index=True)
df_res =df_res.append(Metrics(X_valid, y_valid),
                      ignore_index=True)

df_res.index = ['Train','Test','Valid']

df_res.to_csv('Output_'+BACKBONE+'.csv')
df_res
```

	loss	accuracy	binary accuracy	mAP	false negatives	false positives	true negatives	true positives	auc	Specificty	Sensitivity
Train	0.900839	0.995474	0.995474	0.497737	103221.0	0.0	22703308.0	0.0	0.496463	1.0	0.0
Test	0.900920	0.994827	0.994827	0.497413	35258.0	0.0	6780486.0	0.0	0.496565	1.0	0.0
Valid	0.900971	0.994312	0.994312	0.497156	23858.0	0.0	4170446.0	0.0	0.497159	1.0	0.0

Figure 10 Performance Metrics for UNET TLT RESNET

The respective Figure 9 Liver Segmentation and Predicted Overlap with UNET TLT RESNET shows part Python code and the result of this code fragment. The code reads necessary libraries, changes the data set format, divides the data set into a training set, validation set, and a test set and trains a model. The output provided utilities the dimension of the split datasets and also the training features such as loss of IoU score and F1-score for training and validation dataset. The purpose of such code is perhaps

in the case of segmentation where one aims at categorizing all the pixels in an image given some specific label.

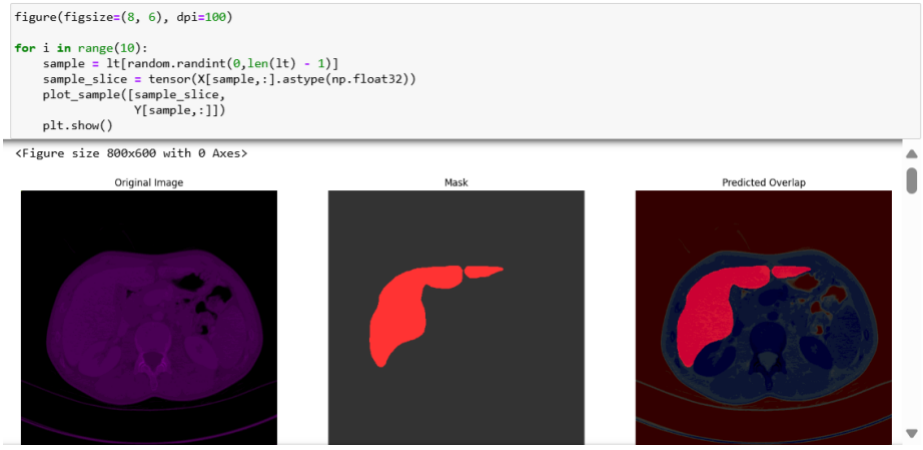


Figure 11 Liver Segmentation and Predicted Overlap with UNET TLT RESNET

The following Figure 10 Liver Segmentation, Predicted Overlap and Bounding Box with UNET TLT RESNET represented that fragment of the Python code on the left and the result of its execution on the right. The loop goes through the input dataset, it selects several samples at random then plots the illustration along with the mask and the predicted overlap. The output displays a few pictures of the original images, and their ground truth masks and the model predictions superimposed on the original images. They also facilitate an understanding of how well the model is doing and, conversely, where it might be having some issues.

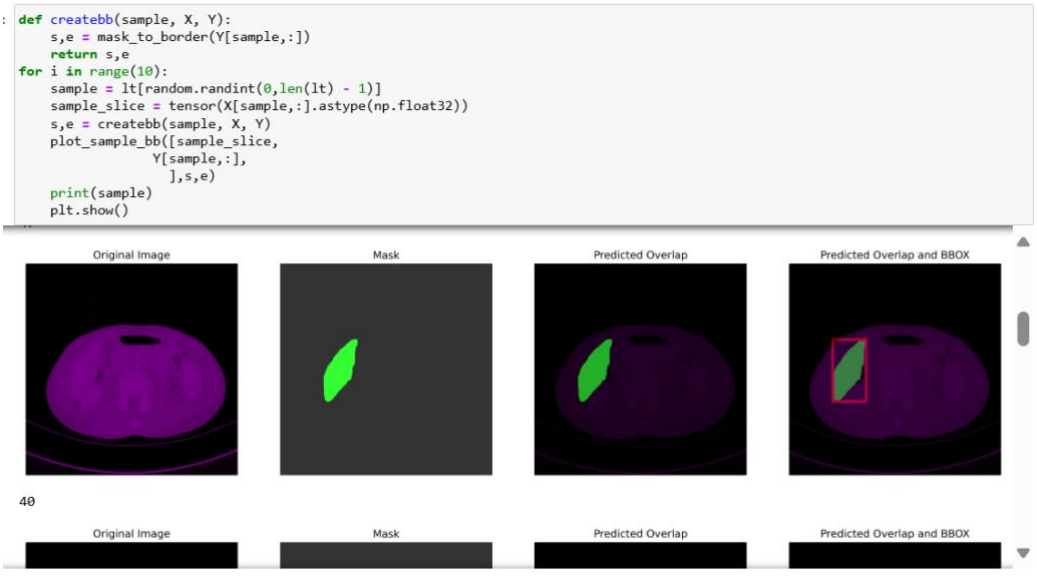


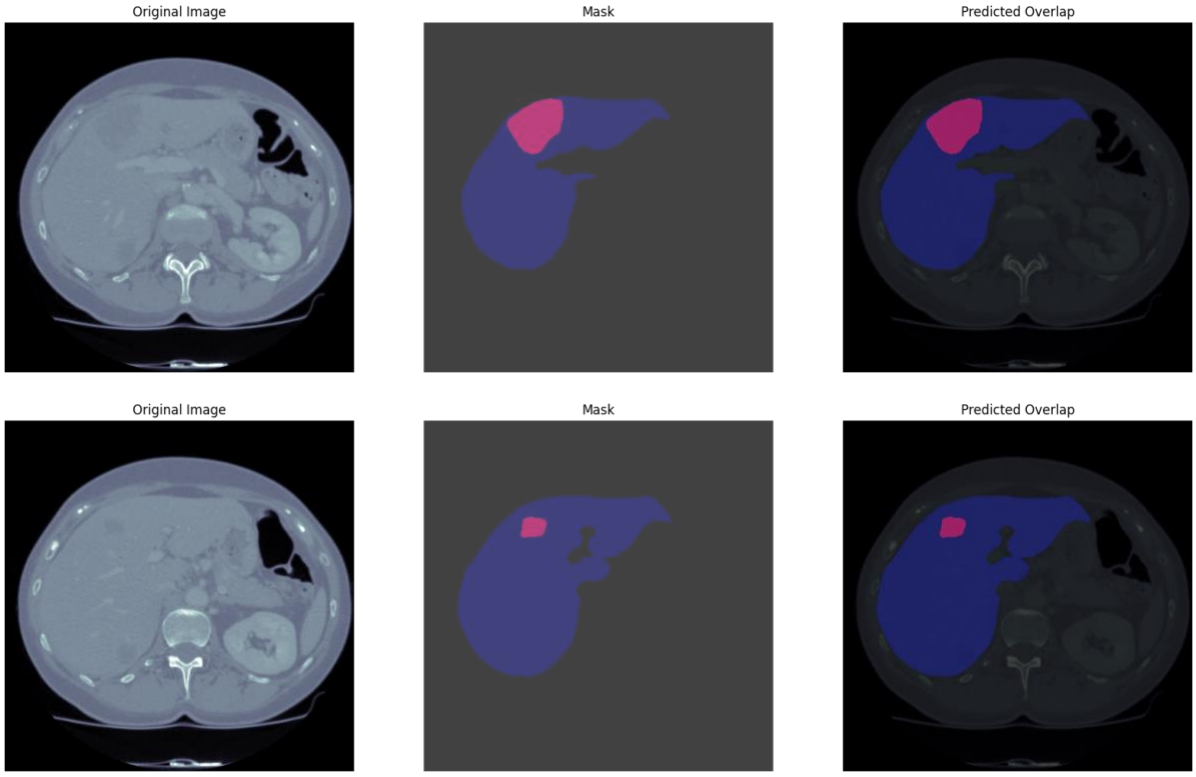
Figure 12 Liver Segmentation, Predicted Overlap and Bounding Box with UNET TLT RESNET

The Figure 10 Liver Segmentation, Predicted Overlap and Bounding Box with UNET TLT RESNET shows Python code and its output: It draws a figure, loops through a set, randomly samples items, and

plots them with their masks, predicted overlap and bounding boxes. The output presents a number of samples of the original images, the ground truth masks, the model predictions on top of the original images with the bounding boxes drawn around the predicted areas and their corresponding ground truth masks. This visualization we can evaluate the model according to not only the segmentation part but also the localization of the box part.

5 Final Results

Figure 13 Results of Segmentation with UNET TLT RESNET showcases the final results for the model allows the segmentation of tumor and liver prediction and overlap. UNET framework, which provides increased visibility of segmented sections inside the picture. By using region suggestions, the model finds and focuses on particular regions of interest, allowing exact segmentation. The combination of UNET for segmentation, TLT for feature refinement, and RESNET as a backbone boosts the network's capacity to capture detailed features and boundaries inside the segmented units. This technique guarantees improved precision in identifying areas of interest while preserving resilience across varied datasets.



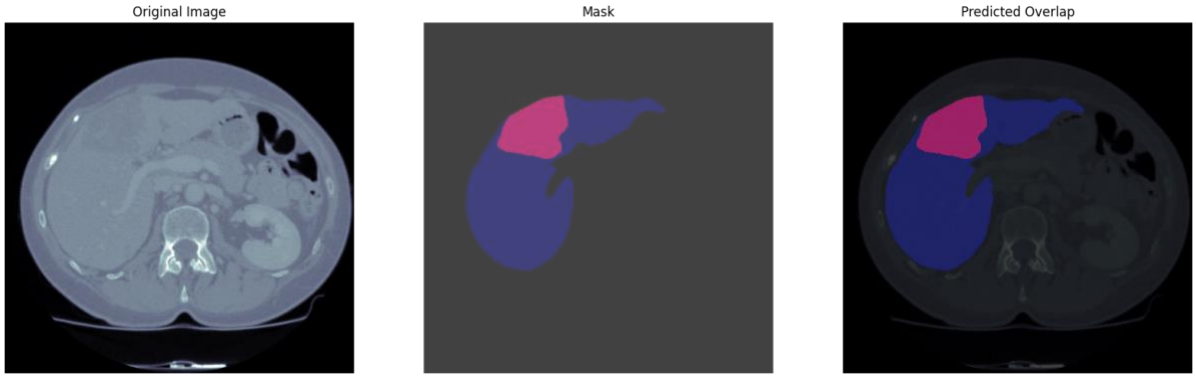


Figure 13 Results of Segmentation with UNET TLT RESNET

Figure 14 Region Proposal based UNET TLT RESNET allows the visualisation of for the region proposition of the segmented unit. UNET framework, which facilitates enhanced visualization of segmented regions within the image. By leveraging region proposals, the model identifies and focuses on specific areas of interest, enabling precise segmentation. The combination of UNET for segmentation, TLT for feature refinement, and RESNET as a backbone enhances the network's ability to capture intricate details and boundaries within the segmented units. This approach ensures higher accuracy in delineating regions of interest while maintaining robustness across diverse datasets.

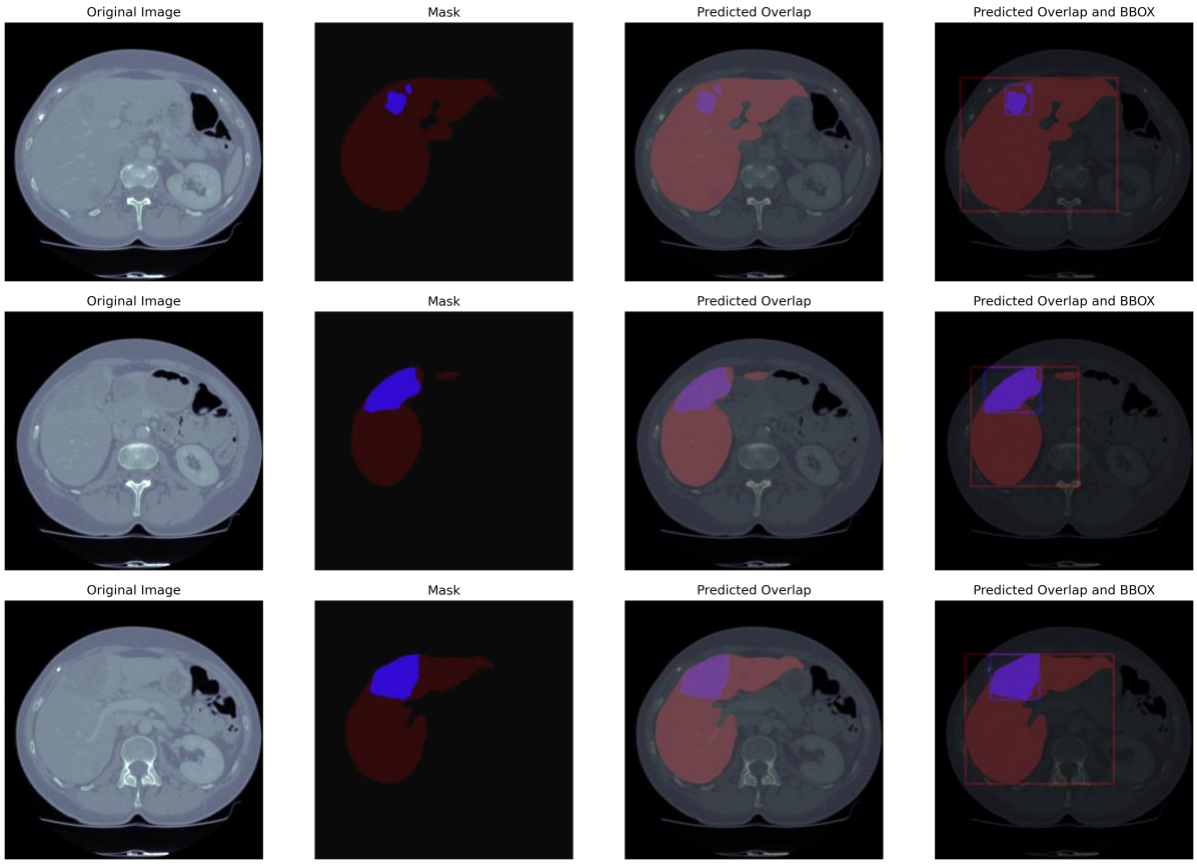


Figure 14 Region Proposal based UNET TLT RESNET

References

[1]. Gayoso, A., Lopez, R., Xing, G., Boyeau, P., Valiollah Pour Amiri, V., Hong, J., Wu, K., Jayasuriya, M., Mehlman, E., Langevin, M. and Liu, Y., 2022. A Python library for probabilistic analysis of single-cell omics data. Nature biotechnology, 40(2), pp.163-166.

[2]. Gunawan, T.S., Abdullah, N.A.J., Kartiwi, M. and Ihsanto, E., 2020, October. Social network analysis using python data mining. In 2020 8th international conference on cyber and IT service management (CITSM) (pp. 1-6). IEEE.

[3]. Bezabih, T.D., Glaety, M.G., Wako, D.A. and Worku, S.G., 2024. Geospatial Data Analysis: A Comprehensive Overview of Python Libraries and Implications. Ethics, Machine Learning, and Python in Geospatial Analysis, pp.72-93.