Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

# School of Computer Science and Engineering

## J Component report

**Programme : MTech Integrated (MIA)**

**Course Title : Deep Learning**

**Course Code : CSE4037**

**Slot : G1**

**Title: Liver Tumor Segmentation**


**Team Members:**

**Nithya Sharma | 19MIA1028**

**K Niharika Samyuktha| 19MIA1083**

**Faculty: Dr. R. Rajalakshmi**


**Date: 29-04-2021**

# CSE4037 - Deep Learning

## J Component Report

**A project report titled**

# Liver Tumor Segmentation

*By*


19MIA1028 - Nithya Sharma

19MIA1083 - K Niharika Samyuktha

Computer Science and Engineering with Specialization in Business Analytics


*Submitted to*

## Dr. R. Rajalakshmi


**School of Computer Science and Engineering**

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)


*April 2022*

## DECLARATION BY THE CANDIDATE

I hereby declare that the report titled "**Liver Tumor Segmentation**" submitted by me to VIT Chennai is a record of bona-fide work undertaken by me under the supervision of **Dr. R. Rajalakshmi, Associate Professor, SCOPE, Vellore Institute of Technology, Chennai.**

Signature of the Candidate

# ACKNOWLEDGEMENT

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Dr. R. Rajalakshmi,** School of Computer Science and Engineering for her consistent encouragement and valuable guidance offered to us throughout the course of the project work.

We are extremely grateful to **Dr. R. Ganesan, Dean,** School of Computer Science and Engineering (SCOPE), Vellore Institute of Technology, Chennai, for extending the facilities of the School towards our project and for his unstinting support.

We express our thanks to our **Head of the Department** for his support throughout the course of this project.

We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the courses.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

# BONAFIDE CERTIFICATE

Certified that this project report entitled "**Liver Tumor Segmentation"** is a bona-fide work of **Nithya Sharma (19MIA1028) and K Niharika Samyuktha (19MIA1083)** carried out the "Liver Tumor Segmentation" - Project work under my supervision and guidance for CSE4037 - Deep Learning

**Dr. R. Rajalakshmi**

SCOPE

## TABLE OF CONTENTS


| Ch. No | Chapter | Page Number |
|---|---|---|
| 1 | Introduction | 8 |
| 2 | Literature Survey | 9 |
| 3 | Proposed Methodology | 15 |
| 4 | Results and Discussion | 17 |
| 5 | Appendix | 19 |
| 6 | Conclusion | 33 |
| 7 | Reference | 34 |

# ABSTRACT

Automatic liver tumor segmentation would have a big impact on liver therapy planning procedures and follow-up assessment, thanks to standardization and incorporation of full volumetric information. We develop a fully automatic method for liver tumor segmentation in CT images based on a 2D fully convolutional neural network with an object-based postprocessing step. We describe our experiments on the LiTS challenge training data set and evaluate segmentation and detection performance. Liver tumor segmentation from CT image is very important to analyze the liver function, pathological and anatomical study of the liver. It is also important for the diagnosis of disease.

# INTRODUCTION

Cancer is the second chief cause of death globally. As per the statistics from World Health Organization (WHO), it was accountable for 8.8 million fatalities in 2015 out of which 788,000 deaths were caused by liver cancer. Liver is largest gland and important metabolic organ of human body. Functions of liver are digestion, metabolism and detoxification. Liver cancers are categorized in two parts such as primary and secondary liver cancer depending upon cause of cancer. Primary liver cancer is cancer that instigates in the tissue of the liver. Primary liver cancer has two types such as Hepatocellular carcinoma and Hemangioma. Hepatocellular carcinoma (HCC), the development of cancer cells in the tissues of the liver, is the most frequent kind of liver cancer. A liver Hemangioma is made up of a tangle of blood vessels.Secondary metastatic liver cancer takes place due to spread of cancer from other body part . An abnormality in the liver causes the change in the liver texture and shape. Exaction and accurate segmentation of liver, its vessels and tumors is required in the disease diagnosis. However due to the intensity of homogeneity inside liver , shape of liver, low contrast, presence of adjacent abdominal organs it becomes challenging task of accurate liver segmentation. Liver diseases can be diagnosed through various medical imaging schemes such as computed tomography (CT), ultrasound (US), Magnetic Resonance Imaging (MRI) etc.

# LITERATURE SURVEY

| S. No | TITLE | JOURNAL/ YEAR OF PUBLICATI ON | DATASET USED | ALGORITHM USED | INTERPRETATION OF RESULTS |
|---|---|---|---|---|---|
| 1) | Liver Cancer Detection Using Hybridized Fully Convolutional Neural Network Based on Deep Learning Framework | 2018 | LiTS Dataset | Neural network | The neural network is capable of detecting bigger lesions (the longest axial diameter ≥10mm) more reliably than smaller ones (<10mm). We presume, based on the performed comparison of LiTS annotations with those done by an experienced MTRA, that this can be attributed to a bigger inter-observer variability with respect to detection of smaller lesions. |

| | | | | | |
|---|---|---|---|---|---|
| 2) | Liver Cancer Detection Using Hybridized Fully Convolutional Neural Network Based on Deep Learning Framework | 2021 | Scopus Database | Hybridized Fully Convolutional Neural Network (HFCNN) | A deep end-to-end learning approach to help discrimination in abdominal CT images of the liver between liver metastases of colorectal cancer and benign cysts has been analyzed. |

| | | | | | |
|---|---|---|---|---|---|
| 3) | Classification and mutation prediction based on histopathology H&E images in liver cancer using deep learning | 2020 | Genomic Data Commons portal (GDC-portal, https://portal.gdc.cancer.gov/) | Neural Network (Inception V3) | A total of 387 WSIs of HCC with corresponding gene mutation information were available. Besides, 67 WSIs of HCC with histopathological grade and related gene mutation information and 34 WSIs of normal liver tissue were selected from Sir Run-Run Shaw Hospital (SRRSH). After each WSI was cropped into small "Tiles", there are 119,596 "Tiles"(HCC vs. normal liver tissue, 87,422 vs. 32,174), 84,149 "Tiles"with histopathological grade (well vs. moderate vs. poor, 14,713 vs. 41,370 vs. 28,066) and 86,323 "Tiles" with corresponding gene mutation information. |

| | | | | | |
|---|---|---|---|---|---|
| 4) | Automatic liver tumor segmentation in CT with fully convolutional neural networks and object-based postprocessing | 2018 | LiTS Dataset | 2D fully convolutional neural network | Achieved segmentation quality for detected tumors comparable to a human expert and is able to detect 77% of potentially measurable tumor lesions in the LiTS reference according to the RECIST 1.1 guidelines. They have observed that the neural network is capable of detecting bigger lesions (the longest axial diameter ≥10 mm) more reliably than smaller ones (<10 mm).<br>Segmentation quality for detected tumors (mean Dice 0.69 vs. 0.72), but is inferior in the detection performance (recall 63% vs. 92%). |

| | | | | | |
|---|---|---|---|---|---|
| 5) | Automatic liver tumor segmentation in follow-up CT studies using Convolutional Neural Networks | 2015 | Private dataset from a hospital. | Convolutional Neural Network (CNN) | Method uses a cascade of registration methods to define a well-fitted tumor ROI on the follow-up scan based on the baseline delineation. A Convolutional Neural Network is trained on all baseline liver masking to class. The CNN is used as a voxel classifier to produce the follow-up tumor segmentation. The segmentation leaks in the resulting tumor segmentation are then removed to produce the final result. |

| | | | | | |
|---|---|---|---|---|---|
| 6) | Liver Cancer Analysis using Machine Learning Techniques | 2017 | Not mentioned | Maching Learning Techniques | The liver tumor is difficult to detect from the CT or MRI images because of two reasons: one is the difference in the liver and non-liver pixel intensities in CT images and another one is detection of liver from overlapped organs. Hence segmentation helps doctors to provide effective treatment by knowing nature of the tumor. |

# PROPOSED METHODOLOGY

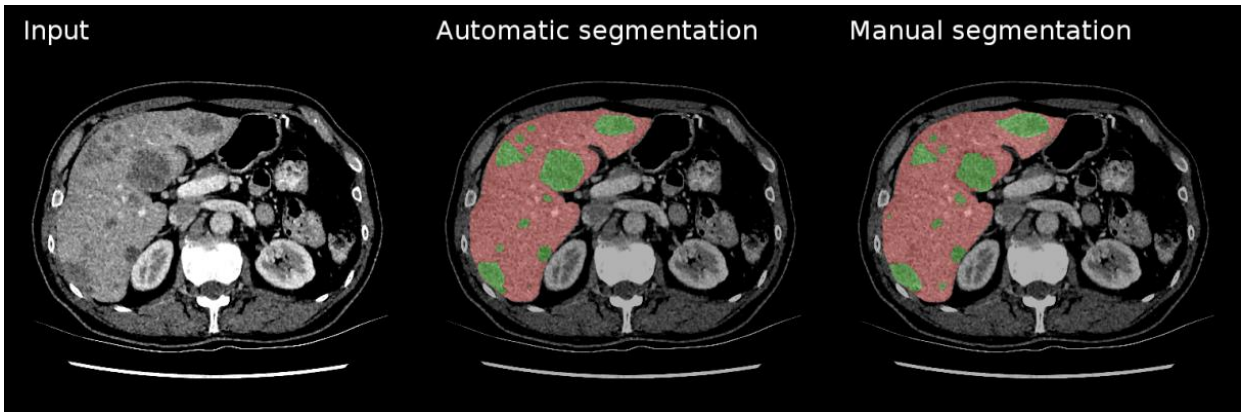
**DATASET:**

Liver cancer is the fifth most commonly occurring cancer in men and the ninth most commonly occurring cancer in women. There were over 840,000 new cases in 2018.

The liver is a common site of primary or secondary tumor development. Due to their heterogeneous and diffusive shape, automatic segmentation of tumor lesions is very challenging.

In light of that, we encourage the development of automatic segmentation algorithms to segment liver lesions in contrast-enhanced abdominal CT scans. The data and segmentations are provided by various clinical sites around the world.

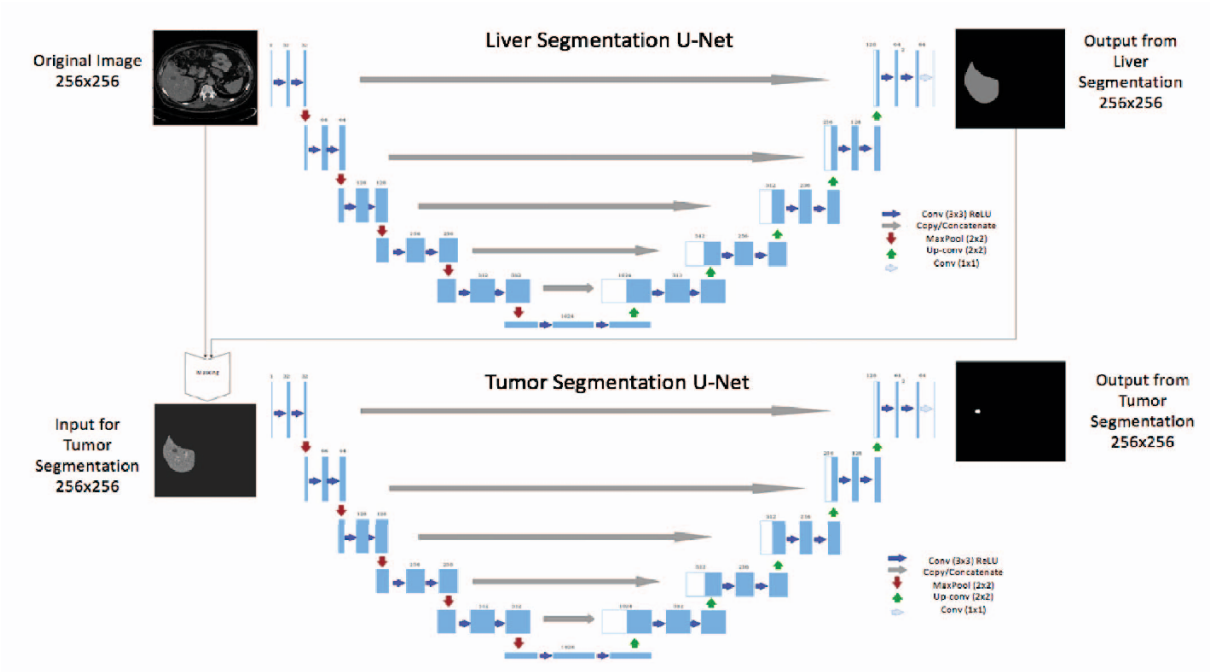
This dataset was extracted from LiTS – Liver Tumor Segmentation Challenge (LiTS17) organised in conjunction with ISBI 2017 and MICCAI 2017.



Automatic Liver and Lesion Segmentation in CT Using Cascaded Fully Convolutional NeuralNetworks and 3D Conditional Random Fields.

We trained CNNs using image patches centered at each pixel. These patches were divided into tumor and normal liver tissue. A given patch is labeled as positive sample if it contains at least 50% or more of liver tumor pixels, otherwise it is labeled as negative sample.


MODEL USED: U-Net



# RESULTS AND DISCUSSION

Batch Normalization:

Batch Normalization is a process to make neural networks faster and more stable through adding extra layers in a deep neural network. The new layer performs the standardizing and normalizing operations on the input of a layer coming from a previous layer.

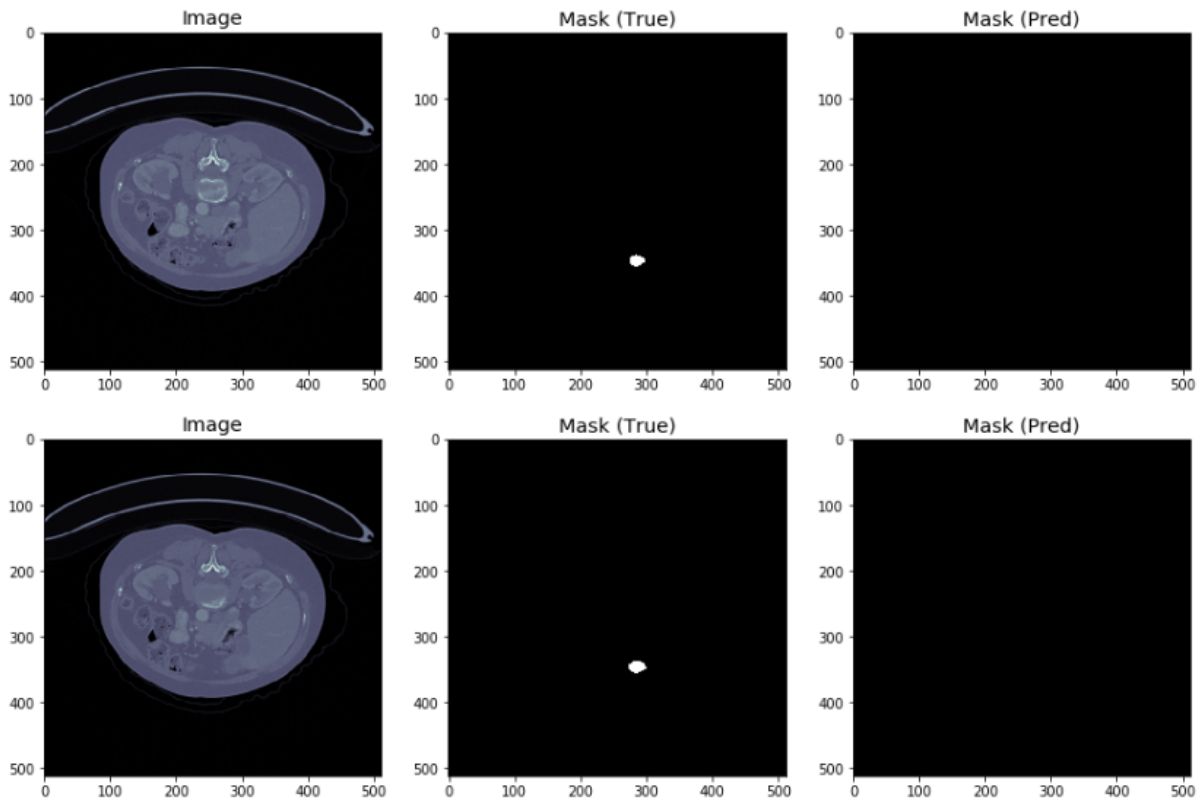
Conv2D:

Keras Conv2D is a 2D Convolution Layer, this layer creates a convolution kernel that is wind with layers input which helps produce a tensor of outputs.

```
model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input (InputLayer)             [(None, 64, 64, 1)]  0           []

 conv1_1 (Conv2D)               (None, 64, 64, 32)   320         ['input[0][0]']

 conv1_1_bn (BatchNormalization  (None, 64, 64, 32)  128         ['conv1_1[0][0]']
 )

 conv1_2 (Conv2D)               (None, 64, 64, 32)   9248        ['conv1_1_bn[0][0]']

 conv1_2_bn (BatchNormalization  (None, 64, 64, 32)  128         ['conv1_2[0][0]']
 )

 pool1 (MaxPooling2D)           (None, 32, 32, 32)   0           ['conv1_2_bn[0][0]']

 conv2_1 (Conv2D)               (None, 32, 32, 64)   18496       ['pool1[0][0]']

 conv2_1_bn (BatchNormalization  (None, 32, 32, 64)  256         ['conv2_1[0][0]']
 )

 conv2_2 (Conv2D)               (None, 32, 32, 64)   36928       ['conv2_1_bn[0][0]']

 conv2_2_bn (BatchNormalization  (None, 32, 32, 64)  256         ['conv2_2[0][0]']
 )

 pool2 (MaxPooling2D)           (None, 16, 16, 64)   0           ['conv2_2_bn[0][0]']

 conv3_1 (Conv2D)               (None, 16, 16, 128)  73856       ['pool2[0][0]']

 conv3_1_bn (BatchNormalization  (None, 16, 16, 128)  512        ['conv3_1[0][0]']
 )

 conv3_2 (Conv2D)               (None, 16, 16, 128)  147584      ['conv3_1_bn[0][0]']

 conv3_2_bn (BatchNormalization  (None, 16, 16, 128)  512        ['conv3_2[0][0]']
 )
```

```
concat8 (Concatenate)          (None, 32, 32, 128)  0          ['conv2_2_bn[0][0]',
                                                                 'dropout_6[0][0]']

conv8_1 (Conv2D)               (None, 32, 32, 64)   73792      ['concat8[0][0]']

conv8_1_bn (BatchNormalization  (None, 32, 32, 64)  256        ['conv8_1[0][0]']
)

conv8_2 (Conv2D)               (None, 32, 32, 64)   36928      ['conv8_1_bn[0][0]']

conv8_2_bn (BatchNormalization  (None, 32, 32, 64)  256        ['conv8_2[0][0]']
)

conv2d_transpose_3 (Conv2DTran  (None, 64, 64, 32)  8224       ['conv8_2_bn[0][0]']
spose)

dropout_7 (Dropout)            (None, 64, 64, 32)   0          ['conv2d_transpose_3[0][0]']

concat9 (Concatenate)          (None, 64, 64, 64)   0          ['conv1_2_bn[0][0]',
                                                                 'dropout_7[0][0]']

conv9_1 (Conv2D)               (None, 64, 64, 32)   18464      ['concat9[0][0]']

conv9_1_bn (BatchNormalization  (None, 64, 64, 32)  128        ['conv9_1[0][0]']
)

conv9_2 (Conv2D)               (None, 64, 64, 32)   9248       ['conv9_1_bn[0][0]']

conv9_2_bn (BatchNormalization  (None, 64, 64, 32)  128        ['conv9_2[0][0]']
)

dropout_8 (Dropout)            (None, 64, 64, 32)   0          ['conv9_2_bn[0][0]']

conv10 (Conv2D)                (None, 64, 64, 1)    33         ['dropout_8[0][0]']

==================================================================================================
Total params: 7,771,297
Trainable params: 7,765,409
Non-trainable params: 5,888
__________________________________________________________________________________________________
```



# APPENDIX

## CODE:

```
from keras.layers import Input, Conv2D, MaxPooling2D, concatenate,
Conv2DTranspose, Dropout
from keras.models import Model
from tensorflow.keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from keras import regularizers
from tensorflow.keras.layers import BatchNormalization as bn
from keras.callbacks import ModelCheckpoint, TensorBoard
from keras.models import model_from_json
import keras.backend as K
import tensorflow as tf
import nibabel as nib
from glob import glob
import matplotlib.pyplot as plt
import numpy as np
from random import shuffle

cd /content/drive/My Drive/DL/Dataset

img_path = glob("volume-*.nii")
mask_path = glob("segmentation-*.nii")

print("Number of images :", len(img_path))


print("Image(volume) path example :", img_path[:3])
print("Mask path example :", mask_path[:3])

print("Number of slices : {0} / Size of each slice :
{1}".format(img_ex.shape[2], img_ex.shape[:2]))


fig, ([ax1, ax2], [ax3, ax4], [ax5, ax6]) = plt.subplots(3, 2, figsize =
(15, 15))
ax1.imshow(img_ex[:, :, img_ex.shape[2] // 2], cmap = 'bone')
ax1.set_title("Image", fontsize = 'x-large')
ax1.grid(False)
ax1.set_xticks([])
ax1.set_yticks([])
```

```
ax2.imshow(mask_ex[:, :, mask_ex.shape[2] // 2], cmap = 'bone')
ax2.set_title("Mask", fontsize = 'x-large')
ax2.grid(False)
ax2.set_xticks([])
ax2.set_yticks([])

ax3.imshow(img_ex[:, :, img_ex.shape[2] // 2 + 36], cmap = 'bone')
ax3.set_title("Image", fontsize = 'x-large')
ax3.grid(False)
ax3.set_xticks([])
ax3.set_yticks([])

ax4.imshow(mask_ex[:, :, mask_ex.shape[2] // 2 + 36], cmap = 'bone')
ax4.set_title("Mask", fontsize = 'x-large')
ax4.grid(False)
ax4.set_xticks([])
ax4.set_yticks([])

ax5.imshow(img_ex[:, :, img_ex.shape[2] // 2 - 100], cmap = 'bone')
ax5.set_title("Image", fontsize = 'x-large')
ax5.grid(False)
ax5.set_xticks([])
ax5.set_yticks([])

ax6.imshow(mask_ex[:, :, mask_ex.shape[2] // 2 - 100], cmap = 'bone')
ax6.set_title("Mask", fontsize = 'x-large')
ax6.grid(False)
ax6.set_xticks([])
ax6.set_yticks([])
class_percentage = []

for i in range(mask_ex.shape[2]):
  _, count = np.unique(mask_ex[: ,:, i], return_counts = True)

  if len(count) == 2:
    per = count[1] / np.sum(count) * 100
  else:
    per = 0

  class_percentage.append(per)
plt.plot(class_percentage)
plt.xlabel("Index", fontsize = "x-large")
plt.ylabel("Class percentage (%)", fontsize = "x-large")
max_idx = class_percentage.index(max(class_percentage))
```

```
print("The index of maximum class percentage : {0}
[{1}%]".format(max_idx, format(class_percentage[max_idx], '.2f')))
max_idx=120

fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (10, 10))
ax1.imshow(img_ex[:, :, max_idx], cmap = 'bone')
ax1.set_title("Image", fontsize = 'x-large')
ax1.grid(False)
ax1.set_xticks([])
ax1.set_yticks([])

mask_max = mask_ex[:, :, max_idx]
mask_max[mask_max == 1] = 0
ax2.imshow(mask_max, cmap = 'bone')
ax2.set_title("Mask", fontsize = 'x-large')
ax2.grid(False)
ax2.set_xticks([])
ax2.set_yticks([])

max_percentage = []

for path in mask_path:

  mask_ex = nib.load(path).get_fdata()
  mask_ex[mask_ex == 1] = 0

  class_percentage = []

  for i in range(mask_ex.shape[2]):
    _, count = np.unique(mask_ex[: ,:, i], return_counts = True)

    if len(count) == 2:
      per = count[1] / np.sum(count) * 100
    else:
      per = 0

    class_percentage.append(per)

  max_percentage.append(max(class_percentage))

class_percentage = []

plt.scatter(list(range(len(max_percentage))), max_percentage)
plt.xlabel("Image ", fontsize = "x-large")
```

```
plt.ylabel("Max Class percentage (%)", fontsize = "x-large")

print("\'Maximum\' Class percentage of all images :",
format(max(max_percentage), '.3f'), "%")
print("\'Minimum\' Class percentage of all images :",
format(min(max_percentage), '.3f'), "%")
```

### Patch Sampling

```
img_ex = nib.load(img_path[10]).get_fdata()
mask_ex = nib.load(mask_path[10]).get_fdata()

mask_ex[mask_ex == 1] = 0



class_percentage = []

for i in range(mask_ex.shape[2]):
  _, count = np.unique(mask_ex[: ,:, i], return_counts = True)

  if len(count) == 2:
    per = count[1] / np.sum(count) * 100
  else:
    per = 0

  class_percentage.append(per)

max_idx = class_percentage.index(max(class_percentage))

max_idx

patch_ratio = [0, 64, 128, 192, 256, 320, 384, 448, 512]

positive_patch = []
positive_mask = []

for x_bin in range(2, len(patch_ratio)):
  for y_bin in range(2, len(patch_ratio)):
    mask_patch = mask_ex[patch_ratio[x_bin-2] : patch_ratio[x_bin],
patch_ratio[y_bin - 2] : patch_ratio[y_bin], 390]


    if 2 in np.unique(mask_patch):
```

```
      img_patch = img_ex[patch_ratio[x_bin-2] : patch_ratio[x_bin],
patch_ratio[y_bin - 2] : patch_ratio[y_bin], 390]

      positive_patch.append(img_patch)
      positive_mask.append(mask_patch)

len(positive_mask)

fig, ([ax1, ax2], [ax3, ax4], [ax5, ax6], [ax7, ax8], [ax9, ax10]) =
plt.subplots(5, 2, figsize = (20, 20))



ax1.imshow(positive_patch[0], cmap = 'bone')
ax1.grid(False)
ax2.imshow(positive_mask[0], cmap = 'bone')
ax2.grid(False)

ax3.imshow(positive_patch[2], cmap = 'bone')
ax3.grid(False)
ax4.imshow(positive_mask[2], cmap = 'bone')
ax4.grid(False)

ax5.imshow(positive_patch[3], cmap = 'bone')
ax5.grid(False)
ax6.imshow(positive_mask[3], cmap = 'bone')
ax6.grid(False)

ax7.imshow(positive_patch[4], cmap = 'bone')
ax7.grid(False)
ax8.imshow(positive_mask[4], cmap = 'bone')
ax8.grid(False)

ax9.imshow(positive_patch[5], cmap = 'bone')
ax9.grid(False)
ax10.imshow(positive_mask[5], cmap = 'bone')
ax10.grid(False)


patch_ratio = [0, 64, 128, 192, 256, 320, 384, 448, 512]

def patch_sampling(img, mask, patch_ratio, threshold):
```

```
  temp_mask = mask

  temp_mask[temp_mask == 1] = 0
  temp_mask[temp_mask == 2] = 1

  positive_patch = []
  positive_mask = []


  for i in range(temp_mask.shape[2]):
    mask_slice = temp_mask[:, :, i]

    if len(np.unique(mask_slice)) != 0:

      for x_bin in range(2, len(patch_ratio)):
        for y_bin in range(2, len(patch_ratio)):
          mask_patch = mask_slice[patch_ratio[x_bin-2] :
patch_ratio[x_bin], patch_ratio[y_bin - 2] : patch_ratio[y_bin]]
          _, count = np.unique(mask_patch, return_counts = True)


          if len(count) == 2:
            mask_percentage = count[1] / sum(count) * 100


            if mask_percentage>0:
              img_patch = img[patch_ratio[x_bin-2] : patch_ratio[x_bin],
patch_ratio[y_bin - 2] : patch_ratio[y_bin], i]
              positive_patch.append(img_patch)
              positive_mask.append(mask_patch)



  return positive_patch, positive_mask


img_ex = nib.load(img_path[5]).get_fdata()
mask_ex = nib.load(mask_path[5]).get_fdata()

patch, mask = patch_sampling(img_ex, mask_ex, patch_ratio, 0)

len(patch)
```

```
len(mask)


per = []

for i, slice in enumerate(mask):
  _, count = np.unique(slice, return_counts = True)

  temp = count[1] / sum(count) * 100
  per.append(temp)

plt.plot(sorted(per))
plt.imshow(mask[5], cmap = 'bone')
plt.grid(False)

def patch_to_slice(patch, patch_ratio, input_shape, conf_threshold):

  slice = np.zeros((512, 512, 1))
  row_idx = 0
  col_idx = 0

  for i in range(len(patch)):

    slice[patch_ratio[row_idx]:patch_ratio[row_idx + 2],
patch_ratio[col_idx]:patch_ratio[col_idx + 2]][patch[i] > conf_threshold]
= 1

    col_idx += 1

    if i != 0 and (i+1) % 15 == 0:
      row_idx += 1
      col_idx = 0

  return slice
input_shape = [64, 64, 1]
dropout_rate = 0.3
l2_lambda = 0.0002

def u_net(input_shape, dropout_rate, l2_lambda):

  # Encoder
  input = Input(shape = input_shape, name = "input")
  conv1_1 = Conv2D(32, (3, 3), padding = "same", activation='relu',
kernel_regularizer=regularizers.l2(l2_lambda), name = "conv1_1")(input)
```

```
  conv1_1 = bn(name = "conv1_1_bn")(conv1_1)
  conv1_2 = Conv2D(32, (3, 3), padding = "same", activation='relu',
kernel_regularizer=regularizers.l2(l2_lambda), name = "conv1_2")(conv1_1)
  conv1_2 = bn(name = "conv1_2_bn")(conv1_2)
  pool1 = MaxPooling2D(name = "pool1")(conv1_2)
  drop1 = Dropout(dropout_rate)(pool1)

  conv2_1 = Conv2D(64, (3, 3), padding = "same", activation='relu',
kernel_regularizer=regularizers.l2(l2_lambda), name = "conv2_1")(pool1)
  conv2_1 = bn(name = "conv2_1_bn")(conv2_1)
  conv2_2 = Conv2D(64, (3, 3), padding = "same", activation='relu',
kernel_regularizer=regularizers.l2(l2_lambda), name = "conv2_2")(conv2_1)
  conv2_2 = bn(name = "conv2_2_bn")(conv2_2)
  pool2 = MaxPooling2D(name = "pool2")(conv2_2)
  drop2 = Dropout(dropout_rate)(pool2)

  conv3_1 = Conv2D(128, (3, 3), padding = "same", activation='relu',
kernel_regularizer=regularizers.l2(l2_lambda), name = "conv3_1")(pool2)
  conv3_1 = bn(name = "conv3_1_bn")(conv3_1)
  conv3_2 = Conv2D(128, (3, 3), padding = "same", activation='relu',
kernel_regularizer=regularizers.l2(l2_lambda), name = "conv3_2")(conv3_1)
  conv3_2 = bn(name = "conv3_2_bn")(conv3_2)
  pool3 = MaxPooling2D(name = "pool3")(conv3_2)
  drop3 = Dropout(dropout_rate)(pool3)

  conv4_1 = Conv2D(256, (3, 3), padding = "same", activation='relu',
kernel_regularizer=regularizers.l2(l2_lambda), name = "conv4_1")(pool3)
  conv4_1 = bn(name = "conv4_1_bn")(conv4_1)
  conv4_2 = Conv2D(256, (3, 3), padding = "same", activation='relu',
kernel_regularizer=regularizers.l2(l2_lambda), name = "conv4_2")(conv4_1)
  conv4_2 = bn(name = "conv4_2_bn")(conv4_2)
  pool4 = MaxPooling2D(name = "pool4")(conv4_2)
  drop4 = Dropout(dropout_rate)(pool4)

  conv5_1 = Conv2D(512, (3, 3), padding = "same", activation='relu',
kernel_regularizer=regularizers.l2(l2_lambda), name = "conv5_1")(pool4)
  conv5_1 = bn(name = "conv5_1_bn")(conv5_1)
  conv5_2 = Conv2D(512, (3, 3), padding = "same", activation='relu',
kernel_regularizer=regularizers.l2(l2_lambda), name = "conv5_2")(conv5_1)
  conv5_2 = bn(name = "conv5_2_bn")(conv5_2)

  # Decoder
  upconv6 = Conv2DTranspose(256,(2, 2), strides=(2, 2),
padding='same')(conv5_2)
```

```
  upconv6 = Dropout(dropout_rate)(upconv6)
  concat6 = concatenate([conv4_2, upconv6], name = "concat6")
  conv6_1 = Conv2D(256, (3, 3), padding = "same",
kernel_regularizer=regularizers.l2(l2_lambda), name = "conv6_1")(concat6)
  conv6_1 = bn(name = "conv6_1_bn")(conv6_1)
  conv6_2 = Conv2D(256, (3, 3), padding = "same",
kernel_regularizer=regularizers.l2(l2_lambda), name = "conv6_2")(conv6_1)
  conv6_2 = bn(name = "conv6_2_bn")(conv6_2)

  upconv7 = Conv2DTranspose(128,(2, 2), strides=(2, 2),
padding='same')(conv6_2)
  upconv7 = Dropout(dropout_rate)(upconv7)
  concat7 = concatenate([conv3_2, upconv7], name = "concat7")
  conv7_1 = Conv2D(128, (3, 3), padding = "same",
kernel_regularizer=regularizers.l2(l2_lambda), name = "conv7_1")(concat7)
  conv7_1 = bn(name = "conv7_1_bn")(conv7_1)
  conv7_2 = Conv2D(128, (3, 3), padding = "same",
kernel_regularizer=regularizers.l2(l2_lambda), name = "conv7_2")(conv7_1)
  conv7_2 = bn(name = "conv7_2_bn")(conv7_2)

  upconv8 = Conv2DTranspose(64,(2, 2), strides=(2, 2),
padding='same')(conv7_2)
  upconv8 = Dropout(dropout_rate)(upconv8)
  concat8 = concatenate([conv2_2, upconv8], name = "concat8")
  conv8_1 = Conv2D(64, (3, 3), padding = "same",
kernel_regularizer=regularizers.l2(l2_lambda), name = "conv8_1")(concat8)
  conv8_1 = bn(name = "conv8_1_bn")(conv8_1)
  conv8_2 = Conv2D(64, (3, 3), padding = "same",
kernel_regularizer=regularizers.l2(l2_lambda), name = "conv8_2")(conv8_1)
  conv8_2 = bn(name = "conv8_2_bn")(conv8_2)

  upconv9 = Conv2DTranspose(32,(2, 2), strides=(2, 2),
padding='same')(conv8_2)
  upconv9 = Dropout(dropout_rate)(upconv9)
  concat9 = concatenate([conv1_2, upconv9], name = "concat9")
  conv9_1 = Conv2D(32, (3, 3), padding = "same",
kernel_regularizer=regularizers.l2(l2_lambda), name = "conv9_1")(concat9)
  conv9_1 = bn(name = "conv9_1_bn")(conv9_1)
  conv9_2 = Conv2D(32, (3, 3), padding = "same",
kernel_regularizer=regularizers.l2(l2_lambda), name = "conv9_2")(conv9_1)
  conv9_2 = bn(name = "conv9_2_bn")(conv9_2)
  dropout = Dropout(dropout_rate)(conv9_2)

  conv10 = Conv2D(1, (1, 1), padding = "same", activation = 'sigmoid',
```

```
name = "conv10")(dropout)


  model = Model(input, conv10)

  return model

model = u_net(input_shape, dropout_rate, l2_lambda)

model.summary()

tf.keras.utils.plot_model(
    model,
    to_file='model.png',
    show_shapes=False,
    show_layer_names=True,
    rankdir='TB',
    expand_nested=False,
    dpi=96
)


print(len(img_path))
for i in range(20):
  print(img_path[i])


def slice_to_patch(slice, patch_ratio):

  slice[slice == 1] = 0
  slice[slice == 2] = 1

  patch_list = []

  for x_bin in range(2, len(patch_ratio)):
    for y_bin in range(2, len(patch_ratio)):
      patch = slice[patch_ratio[x_bin-2] : patch_ratio[x_bin],
patch_ratio[y_bin - 2] : patch_ratio[y_bin]]
      patch = patch.reshape(patch.shape + (1,))
      patch_list.append(patch)

  return np.array(patch_list)
```

```
def patch_to_slice(patch, patch_ratio, input_shape, conf_threshold):

  slice = np.zeros((512, 512, 1))
  row_idx = 0
  col_idx = 0

  for i in range(len(patch)):

    slice[patch_ratio[row_idx]:patch_ratio[row_idx + 2],
patch_ratio[col_idx]:patch_ratio[col_idx + 2]][patch[i] > conf_threshold]
= 1

    col_idx += 1

    if i != 0 and (i+1) % 15 == 0:
      row_idx += 1
      col_idx = 0

  return slice


def weighted_binary_crossentropy(y_true, y_pred):
    y_pred = tf.clip_by_value(y_pred, 10e-8, 1.-10e-8)
    loss = - (y_true * K.log(y_pred) * 0.90 + (1 - y_true) * K.log(1 -
y_pred) * 0.10)

    return K.mean(loss)


smooth = 1.
def dice_coef(y_true, y_pred):
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = K.sum(y_true_f * y_pred_f)
    return (2. * intersection + smooth) / (K.sum(y_true_f) +
K.sum(y_pred_f) + smooth)
```

## Training

```
total_data = 0
total_patch = []
total_mask = []
```

```
for i in range(len(img_path) - 105):

  img_3D = nib.load(img_path[i]).get_data()
  mask_3D = nib.load(mask_path[i]).get_data()

  pos_patch, pos_mask, neg_patch, neg_mask = patch_sampling(img_3D,
mask_3D, patch_ratio, 3, 3.0)
  total_patch += (pos_patch + neg_patch)
  total_mask += (pos_mask + neg_mask)

  print("======= Step [{0} / {1}] : # of patches = {2} | # of total
training images = {3} =======".
        format(format(i+1, '>2'), len(img_path), format(len(pos_patch) +
len(neg_patch), '>5'), format(len(total_patch), '>5')))

mask_3D.shape

total_patch = np.array(total_patch).reshape((len(total_patch), 64, 64,
1))
total_mask = np.array(total_mask).reshape((len(total_mask), 64, 64, 1))

np.save("total_patch.npy", total_patch)
np.save("total_mask.npy", total_mask)

total_mask.shape



adam = Adam(lr = 0.0001)

model.compile(optimizer = adam, loss = weighted_binary_crossentropy,
metrics = [dice_coef])

model.fit(total_patch, total_mask, batch_size = 512, epochs = 10)

#model = model.get_layer("model_7")

model_json = model.to_json()
with open("model_json_final.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model.save_weights("model_weights_final.h5")
```

```
print("Saved model to disk")



model_json = model.to_json()
with open("model_json.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model.save_weights("model_weights.h5")
print("Saved model to disk")
```

**Load**

```
json_file = open('model_json.json', 'r')

loaded_model_json = json_file.read()

json_file.close()

loaded_model = model_from_json(loaded_model_json)

# load weights into new model

loaded_model.load_weights("model_weights.h5")

print("Loaded model from disk")

loaded_model.summary()

img_ex = nib.load(img_path[25]).get_data()
mask_ex = nib.load(mask_path[25]).get_data()


mask_ex[mask_ex == 1] = 0

for i in range(mask_ex.shape[2]):
    _, count = np.unique(mask_ex[:, :, i], return_counts=True)

    if len(count) > 1 and count[1] > 300:

        patch_ex = slice_to_patch(img_ex[:, :, i], patch_ratio)
        prediction = loaded_model.predict(patch_ex)
        prediction_mask = patch_to_slice(prediction, patch_ratio,
input_shape, conf_threshold = 0.97)

        fig, (ax1,ax2,ax3) = plt.subplots(1, 3, figsize = ((15, 15)))
```

```
        ax1.imshow(np.rot90(img_ex[:, :, i], 3), cmap = 'bone')
        ax1.set_title("Image", fontsize = "x-large")
        ax1.grid(False)
        ax2.imshow(np.rot90(mask_ex[:, :, i], 3), cmap = 'bone')
        ax2.set_title("Mask (True)", fontsize = "x-large")
        ax2.grid(False)
        ax3.imshow(np.rot90(prediction_mask.reshape((512, 512)), 3), cmap
= 'bone')
        ax3.set_title("Mask (Pred)", fontsize = "x-large")
        ax3.grid(False)
        plt.show()
```

# CONCLUSION

This paper discusses the Liver Tumor Segmentation from CT-scan slices.

U-Net a deep learning architecture was used. We observed that the neural network is capable of detecting bigger lesions (the longest axial diameter ≥10 mm) more reliably than smaller ones (<10 mm). We see the method described in this paper as promising, but it is clear that more work needs to be done to match the human detection performance.

# REFERENCES


- https://www.sciencedirect.com/topics/computer-science/tumor-segmentation
- Ayalew, Y. A., Fante, K. A., & Mohammed, M. A. (2021, March 1). *Modified U-net for liver cancer segmentation from computed tomography images with a new class balancing method - BMC Biomedical Engineering*. BioMed Central. Retrieved May 10, 2022, from https://bmcbiomedeng.biomedcentral.com/articles/10.1186/s42490-021-00050-y
- Chlebus, G., Schenk, A., Moltz, J. H., van Ginneken, B., Hahn, H. K., & Meine, H. (2018, October 19). *Automatic liver tumor segmentation in CT with fully convolutional neural networks and object-based postprocessing*. Nature News. Retrieved May 10, 2022, from https://www.nature.com/articles/s41598-018-33860-7
- Almotairi, S., Kareem, G., Aouf, M., Almutairi, B., & Salem, M. A.-M. (2020, March 10). *Liver tumor segmentation in CT scans using modified segnet*. MDPI. Retrieved May 10, 2022, from https://www.mdpi.com/1424-8220/20/5/1516
- Christ, P. F., Ettlinger, F., Grün, F., Elshaera, M. E. A., Lipkova, J., Schlecht, S., Ahmaddy, F., Tatavarty, S., Bickel, M., Bilic, P., Rempfler, M., Hofmann, F., Anastasi, M. D., Ahmadi, S.-A., Kaissis, G., Holch, J., Sommer, W., Braren, R., Heinemann, V., & Menze, B. (2017, February 23). *Automatic liver and tumor segmentation of CT and MRI volumes using cascaded fully convolutional neural networks*. arXiv.org. Retrieved May 10, 2022, from https://arxiv.org/abs/1702.05970
- Rela, M., Rao, S. N., & Reddy, P. R. (n.d.). *Liver tumor segmentation using Superpixel based fast fuzzy C means clustering*. International Journal of Advanced Computer Science and Applications (IJACSA). Retrieved May 10, 2022, from https://thesai.org/Publications/ViewPaper?Volume=11&Issue=11&Code=IJACSA&SerialNo=49