

# Configuration Manual

MSc Research Project  
MSc. in Data Analytics

Karan Mahendra Singh  
Student ID: 18192726

School of Computing  
National College of Ireland

Supervisor: Dr. Rashmi Gupta

National College of Ireland  
Project Submission Sheet  
School of Computing



<b>Student Name:</b>	Karan Mahendra Singh
<b>Student ID:</b>	18192726
<b>Programme:</b>	MSc. in Data Analytics
<b>Year:</b>	2020
<b>Module:</b>	Research Project
<b>Supervisor:</b>	Dr. Rashmi Gupta
<b>Submission Due Date:</b>	17/12/2020
<b>Project Title:</b>	Deep Learning for Detection of Tumours from CT Images of Multiple Organs
<b>Word Count:</b>	XXX
<b>Page Count:</b>	19

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.

<b>Signature:</b>	
<b>Date:</b>	16th December 2020

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:**

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

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

<b>Office Use Only</b>	
Signature:	
Date:	
Penalty Applied (if applicable):	

# Configuration Manual

Karan Mahendra Singh  
18192726

## 1 System Configuration

This section provides the expected system configuration to run the code smoothly. The project code was developed with Google Colab. The same code may still be run on a base machine with the following hardware and software configurations which will ensure seamless execution.

### 1.1 Hardware Requirements

The system on which the code was developed had the hardware specifications shown in Figure 1.

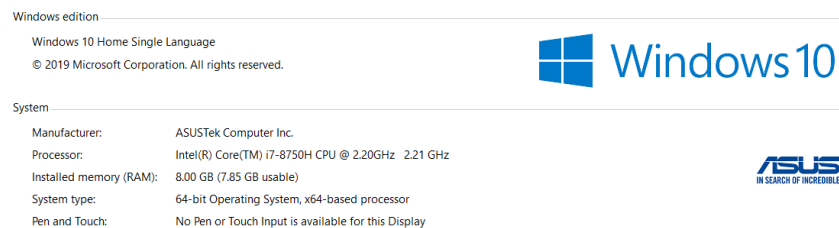


Figure 1: Hardware Configuration

### 1.2 Software Requirements

The following software configurations were used to develop the code.

#### 1.2.1 Google Colab

The project was developed using Google Colab. All required data was stored in Google drive, which was mounted on Colab as shown in Figure 2. The user is redirected to a login page by clicking on the link that appears after executing the cell shown in Figure 2. Thereafter, the user is expected to paste the authorization code shown to him after successfully logging in to the Google Drive account.

Furthermore, this code uses a hardware accelerator available on Google Colab. The runtime type is changed to use GPU Hardware accelerator for gaining computation power, Figure 3. TPU is another accelerator provided by Colab.

```
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

Figure 2: Mounting of Google Drive

## Notebook settings

Hardware accelerator  
GPU ▼ ?

To get the most out of Colab, avoid using a GPU unless you need one. [Learn more](#)

☐ Omit code cell output when saving this notebook

CANCEL SAVE

Figure 3: GPU Hardware accelerator with Colab

### 1.2.2 Anaconda and Jupyter Notebook

Anaconda is an open-source platform for data science practitioners. It is suitable for use of students and professionals. It provides IDEs for Python and R. Anaconda is required to use Jupyter Notebook. A suitable edition may be downloaded and installed easily from the website<sup>1</sup>. After installation, Anaconda Navigator shows a list of IDEs available to work with. Jupyter may be opened by clicking the Launch button next to it, shown in Figure 4. This project may be easily run with the Individual Edition of Anaconda, a Jupyter Notebook and Python 3.

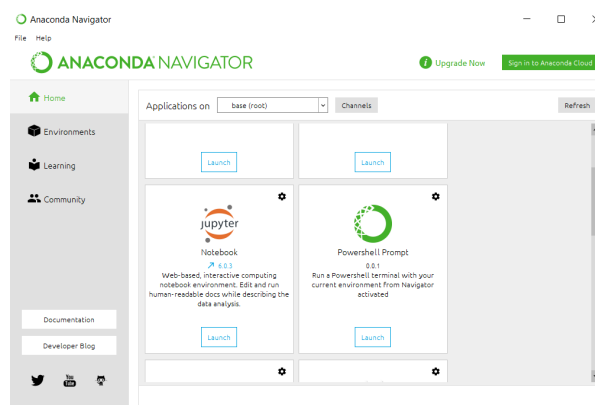


Figure 4: Anaconda Navigator

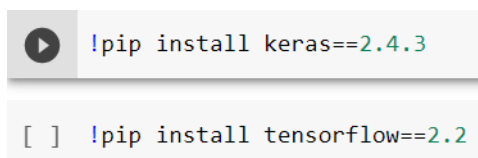
<sup>1</sup><https://www.anaconda.com/products/individual>

### 1.2.3 Other softwares

Jupyter Notebook requires a web browser to deploy itself. Google Chrome was used for this. Further, all documentation of the project was done with TeXstudio.

### 1.2.4 Environment setup

The code requires certain versions of keras and tensorflow to be installed. This was done with code shown in Figure 5:



```
!pip install keras==2.4.3

[ ] !pip install tensorflow==2.2
```

Figure 5: Keras and tensorflow versions

Further, the libraries shown in Figure 6 were imported for loading data, both image and csv, its representation, plotting, visualization and segmentation:



```
%matplotlib inline
from glob import glob
import os, pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from skimage.io import imread
import seaborn as sns
from matplotlib.collections import PatchCollection
from matplotlib.patches import Rectangle
```

Figure 6: Libraries

## 2 Data Preparation

### 2.1 About the data

The dataset used in this study was taken from Kaggle<sup>2</sup> and is called DeepLesion. It was developed and compiled by NIH<sup>3</sup> and was very large in size. It was therefore downloaded from Kaggle as it is a subset in a much smaller size and stored on Google Drive for accessing through Colab, as described earlier.

The dataset consists of over 1300 CT (computed tomography) scan images of lesions from eight body organs such as lungs, liver, kidneys, pelvis, soft tissue, bone and abdomen. This provides a good mix of CT scans for the aim of developing a unified segmentation algorithm for detecting tumor from various body parts.

---

<sup>2</sup><https://www.kaggle.com/kmader/nih-deeplesion-subset/data>

<sup>3</sup><https://www.nih.gov/news-events/news-releases/nih-clinical-center-releases-dataset-32000-ct-images>

Further, it contains a csv file which holds filenames of images, the type of lesion, patient index and so on. Most importantly, it contains a column called "Bounding-boxes" which gives segmentation coordinates of the lesion in that image file, Figure 7.

	File_name	Patient_index	Study_index	Series_ID	Key_slice_index	Measurement_coordinates	Bounding_boxes
0	000001_01_01_109.png	1	1	1	109	233.537, 95.0204, 234.057, 106.977, 231.169, 1...	226.169, 90.0204, 241.252, 111.977
1	000001_02_01_014.png	1	2	1	14	224.826, 289.296, 224.016, 305.294, 222.396, 2...	217.396, 284.296, 233.978, 310.294
2	000001_02_01_017.png	1	2	1	17	272.323, 320.763, 246.522, 263.371, 234.412, 3...	229.412, 258.371, 285.221, 325.763
3	000001_03_01_088.png	1	3	1	88	257.759, 157.618, 260.018, 133.524, 251.735, 1...	246.735, 128.524, 270.288, 162.618
4	000001_04_01_017.png	1	4	1	17	304.019, 230.585, 292.217, 211.789, 304.456, 2...	287.217, 206.789, 309.456, 235.585

Figure 7: csv file loaded as a data frame

## 2.2 Analysis of the csv file

Histogram of number of patients according to gender are shown in Figure 8.

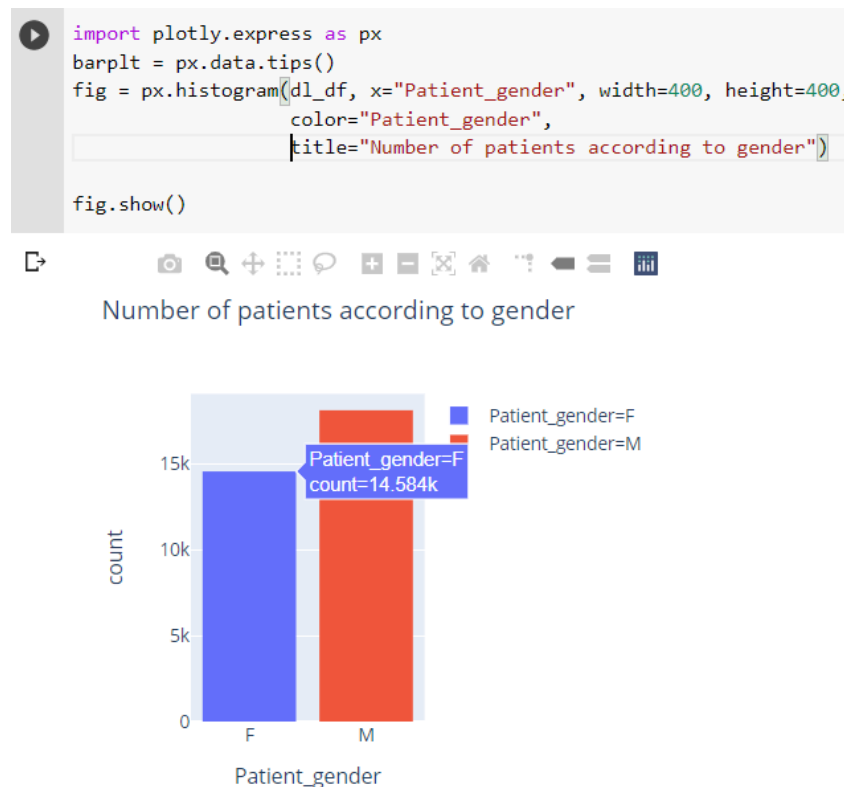


Figure 8: Number of Patients according to gender

Looking at mostly which age group and gender of people have the most lesions, Figure 9.



Figure 9: Patients according to age and gender

Trying to see if patients of specific age and gender have larger sizes of lesions, Figure 10.

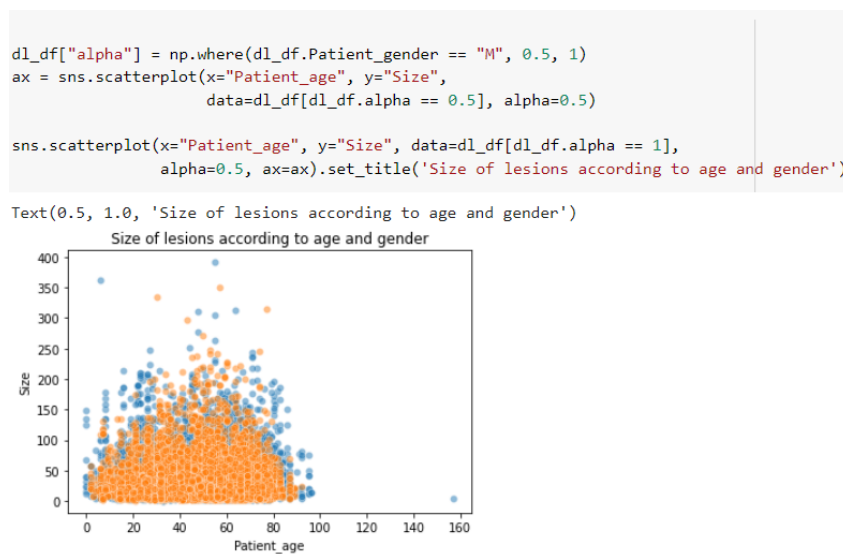


Figure 10: Size of lesions according to age and gender

Finding out which type of lesions most patients have, Figure 11.

Further breaking down the number of patients with types of lesions according to gender, Figure 12.



Where types of lesions are as follows:

1: bone, 2: abdomen, 3: mediastinum, 4: liver, 5: lung,  
6: kidney, 7: soft tissue, and 8: pelvis

Figure 11: Number of patients according to type of lesion

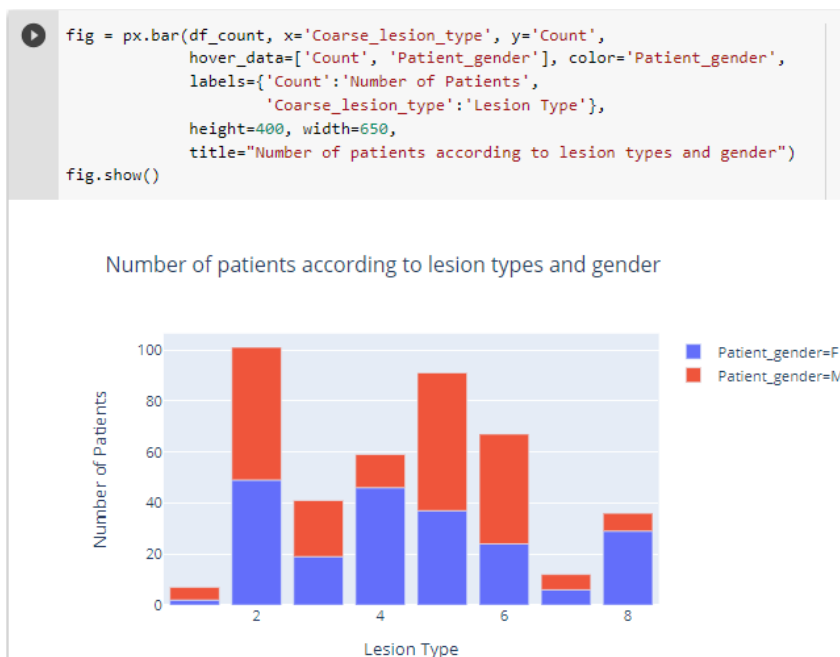


Figure 12: Number of patients according to lesion types and gender

## 2.3 Obtaining segmentations

### 2.3.1 Extracting bounding boxes

The data set contains RECIST annotations for lesions in the csv file under "Bounding\_boxes" column. These were contained in a new column and made fit to be forming



segmentations, Figure 13.

```

▶ dl_df['exists'] = dl_df['location'].map(os.path.exists)
dl_df = dl_df[dl_df['exists']].drop('exists', 1)
# extracting the bounding boxes from csv for creating masks
dl_df['bbox'] = dl_df['Bounding_boxes'].map(lambda x: np.reshape([float(y) for y in x.split(',')], (-1, 4)))
print('Found', dl_df.shape[0], 'patients with images')

```

Found 1350 patients with images

Figure 13: Bounding Boxes of lesions

After extracting the bounding boxes, masks may be formed as shown in Figure 14.

```

▶ img_read = lambda x: imread(x).astype(np.float32)-32768

_, test_row = next(dl_df.sample(1, random_state=0).iterrows())
fig, ax1 = plt.subplots(1, 1, figsize = (10, 10))
c_img = img_read(test_row['location'])
ax1.imshow(c_img, vmin = -1200, vmax = 600, cmap = 'gray')
ax1.add_collection(PatchCollection(masks(test_row), alpha = 0.25, facecolor = 'yellow'))
ax1.set_title('Gender:{Patient_gender}, Age:{Patient_age}'.format(**test_row))

```

Text(0.5, 1.0, 'Gender:F, Age:34.0')

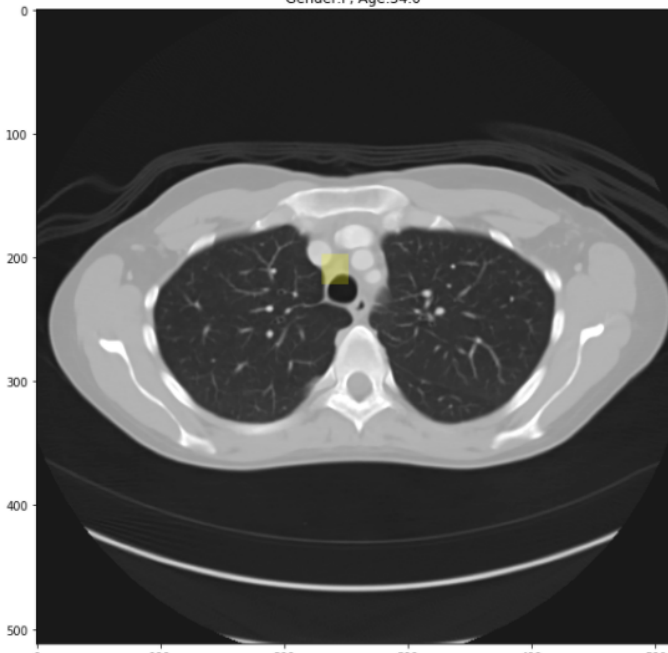


Figure 14: Segmented lesions in lungs

### 2.3.2 Forming segmentations

Segmentations may be formed using the method shown in Figure 15.

Segmentations so formed are displayed in Figure 16.

## 2.4 Exporting to HDF5

All components such as image, lesion segmentations and image paths which are essentially just filenames are stored in the DL.info.csv file, are gathered to form a HDF5 file so that everything may be obtained in one place. HDF5<sup>4</sup> enables easy storage and manipulation

<sup>4</sup><https://docs.h5py.org/en/stable/>

```

def form_segments(in_img, in_row):
    yy, xx = np.meshgrid(range(in_img.shape[0]),
                           range(in_img.shape[1]),
                           indexing='ij')
    out_seg = np.zeros_like(in_img)
    for (start_x, start_y, end_x, end_y) in in_row['bbox']:
        c_seg = (xx<end_x) & (xx>start_x) & (yy<end_y) & (yy>start_y)
        out_seg+=c_seg
    return np.clip(out_seg, 0, 1).astype(np.float32)

```

Figure 15: Forming segmentations

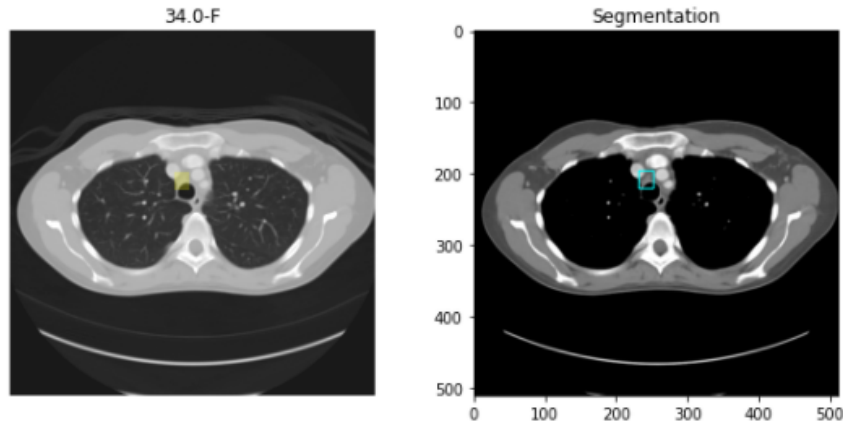


Figure 16: Segmented lesions

of data in the numerical format. They may be used as NumPy arrays. Images are stored as float64 and masks as boolean. After downloading the necessary libraries, Figure 17, the file is opened and split into train and test sets, Figure 18.

```

%matplotlib inline
from glob import glob
import os
import pandas as pd
import matplotlib.pyplot as plt
from skimage.io import imread
import zipfile as zf
import numpy as np
import h5py
from keras.utils.io_utils import HDF5Matrix

```

Figure 17: Libraries

### 3 Models

The main aim of this study is to find a unified approach to segment lesions from a variety of organs. This way, the same model could be applied to a maximum possible organs for detecting tumors. This study applies the basic U-Net model against the proposed model. Our model uses a modified version of U-Net to compare the results. Furthermore, this

```

h5_path = os.path.join(base_h5_dir, 'deepleesion.h5')
with h5py.File(h5_path, 'r') as h:
    print(list(h.keys()))
    for k in ['image', 'mask']:
        print(k, h[k].shape, h[k].dtype)
    base_shape = h['image'].shape

['file_name', 'image', 'mask']
image (1350, 512, 512, 1) float64
mask (1350, 512, 512, 1) bool

ct_window_func = lambda x: np.clip((x+175.0)/275, -1, 1)
get_xyf = lambda s, e: (HDF5Matrix(h5_path, 'image', start=s, end=e, normalizer=ct_window_func),
                        HDF5Matrix(h5_path, 'mask', start=s, end=e),
                        HDF5Matrix(h5_path, 'file_name', start=s, end=e)
                        )

train_split = 0.7
cut_val = int(base_shape[0]*train_split)
train_x, train_y, train_paths = get_xyf(0, cut_val)
test_x, test_y, test_paths = get_xyf(cut_val, None)

```

Figure 18: HDF5 and test-train split

study applies Watershed and color based segmentation methods for lesion segmentation. Finally, all results are presented.

### 3.1 Basic UNet model

The code in Figure 19 is used to define the model.

```

from keras.layers import Input, Activation, Conv2D, MaxPool2D, UpSampling2D,
Dropout, concatenate, BatchNormalization, Cropping2D, ZeroPadding2D,
SpatialDropout2D
from keras.layers import Conv2DTranspose, Dropout, GaussianNoise
from tensorflow.keras.models import Model
from keras import backend as K

def up_scale(in_layer):
    filt_count = in_layer.shape[-1]
    return Conv2DTranspose(filt_count//2+2, kernel_size = (2,2),
                           strides = (2,2), padding = 'same')(in_layer)

def up_scale_fancy(in_layer):
    return UpSampling2D(size=(2,2))(in_layer)

input_layer = Input(shape=train_x.shape[1:])
sp_layer = GaussianNoise(0.1)(input_layer)
bn_layer = BatchNormalization()(sp_layer)

```

Figure 19: Defining the model

Some layers in the model are shown in Figure 20.

### 3.2 Proposed Model

The proposed model is a modified UNet model built from scratch and trained entirely on the dataset used in the research. Layers of this model are shown in Figure 21.

Layers can be plotted to show a neater diagram illustrating how the system will work internally, Figure 22.

Model: "model"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 512, 512, 1)]	0	
gaussian_noise (GaussianNoise)	(None, 512, 512, 1)	0	input_1[0][0]
batch_normalization (BatchNormaliza	(None, 512, 512, 1)	4	gaussian_noise[0][0]
conv2d (Conv2D)	(None, 512, 512, 8)	208	batch_normalization[0][0]
max_pooling2d (MaxPooling2D)	(None, 256, 256, 8)	0	conv2d[0][0]
conv2d_1 (Conv2D)	(None, 256, 256, 16)	1168	max_pooling2d[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 128, 128, 16)	0	conv2d_1[0][0]
conv2d_2 (Conv2D)	(None, 128, 128, 32)	4640	max_pooling2d_1[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 64, 64, 32)	0	conv2d_2[0][0]
conv2d_3 (Conv2D)	(None, 64, 64, 32)	9248	max_pooling2d_2[0][0]
spatial_dropout2d (SpatialDropo	(None, 64, 64, 32)	0	conv2d_3[0][0]
conv2d_4 (Conv2D)	(None, 64, 64, 16)	4624	spatial_dropout2d[0][0]
conv2d_5 (Conv2D)	(None, 64, 64, 16)	4624	spatial_dropout2d[0][0]
conv2d_6 (Conv2D)	(None, 64, 64, 16)	4624	spatial_dropout2d[0][0]
conv2d_7 (Conv2D)	(None, 64, 64, 16)	4624	spatial_dropout2d[0][0]
conv2d_8 (Conv2D)	(None, 64, 64, 16)	4624	spatial_dropout2d[0][0]
conv2d_9 (Conv2D)	(None, 64, 64, 16)	4624	spatial_dropout2d[0][0]

Figure 20: Layers in UNet

Model: "model"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 512, 512, 1)]	0	
gaussian_noise (GaussianNoise)	(None, 512, 512, 1)	0	input_1[0][0]
batch_normalization (BatchNormaliza	(None, 512, 512, 1)	4	gaussian_noise[0][0]
conv2d (Conv2D)	(None, 512, 512, 8)	208	batch_normalization[0][0]
max_pooling2d (MaxPooling2D)	(None, 256, 256, 8)	0	conv2d[0][0]
conv2d_1 (Conv2D)	(None, 256, 256, 16)	1168	max_pooling2d[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 128, 128, 16)	0	conv2d_1[0][0]
conv2d_2 (Conv2D)	(None, 128, 128, 32)	4640	max_pooling2d_1[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 64, 64, 32)	0	conv2d_2[0][0]
conv2d_3 (Conv2D)	(None, 64, 64, 32)	9248	max_pooling2d_2[0][0]
spatial_dropout2d (SpatialDropo	(None, 64, 64, 32)	0	conv2d_3[0][0]
conv2d_4 (Conv2D)	(None, 64, 64, 16)	4624	spatial_dropout2d[0][0]
conv2d_5 (Conv2D)	(None, 64, 64, 16)	4624	spatial_dropout2d[0][0]
concatenate (Concatenate)	(None, 64, 64, 64)	0	spatial_dropout2d[0][0] conv2d_4[0][0] conv2d_5[0][0]
conv2d_transpose (Conv2DTranspo	(None, 128, 128, 34)	8738	concatenate[0][0]
concatenate_1 (Concatenate)	(None, 128, 128, 66)	0	conv2d_transpose[0][0] conv2d_2[0][0]
spatial_dropout2d_1 (SpatialDro	(None, 128, 128, 66)	0	concatenate_1[0][0]
conv2d_6 (Conv2D)	(None, 128, 128, 128)	33920	spatial_dropout2d_1[0][0]
batch_normalization_1 (BatchNor	(None, 128, 128, 128)	512	conv2d_6[0][0]

Figure 21: Layers in the proposed model

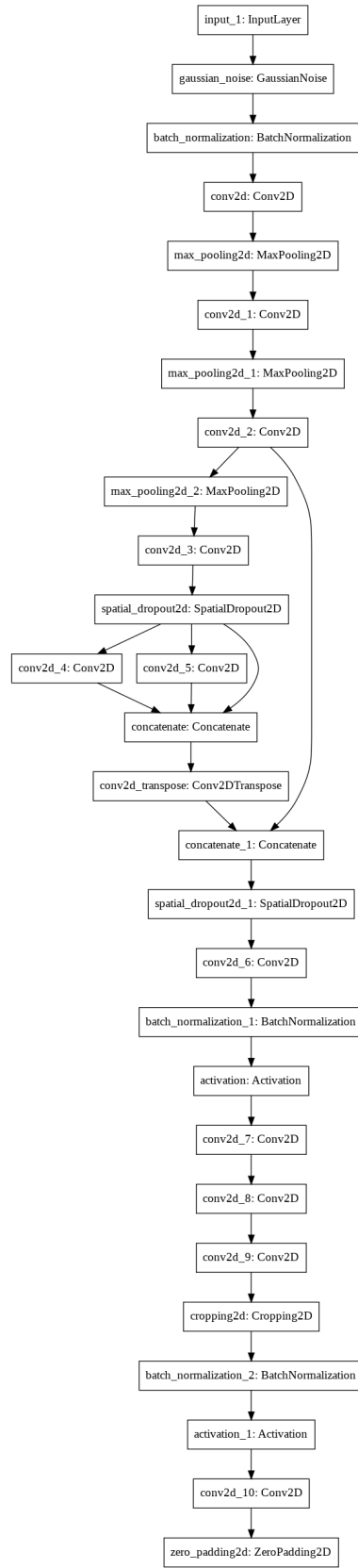


Figure 22: Architecture of the proposed model

### 3.3 Watershed Algorithm

Some libraries required for using Watershed algorithm for segmentation are shown in Figure 23.

```
# import libraries
import numpy as np
import cv2, matplotlib.pyplot as plt
%matplotlib inline
```

Figure 23: Libraries for Watershed Algorithm

The data used for this algorithm may be generated with the code in Figure 24 . For generating the exact images used in the study, it is necessary to specify the random\_state = 4, 55, 74, and 3 for obtaining images of lungs, pelvis, liver and abdomen, respectively.

To be used for generating images for Water Segmentation model

```
[ ] img_read = lambda x: imread(x).astype(np.float32)-32768

_, test_row = next(dl_df.sample(1, random_state=86).iterrows())
fig, ax1 = plt.subplots(1, 1, figsize = (10, 10))
c_img = img_read(test_row['location'])
ax1.imshow(c_img, vmin = -1200, vmax = 600, cmap = 'gray')

# This line may be excluded for generating image without bounding boxes of lesions
ax1.add_collection(PatchCollection(masks(test_row), alpha = 0.25, facecolor = 'yellow'))

ax1.set_title('{File_name} Gender:{Patient_gender}, Age:{Patient_age}, File:{File_name}'.format(**test_row))
print('{File_name}'.format(**test_row))
fig.savefig('/content/drive/My Drive/unsegmented_img.png', dpi = 300)
```

Figure 24: Data for Watershed Algorithm

The image is imported in BGR by default thus, it was converted to RGB, Figure 25. This step is done for both Watershed as well as other OpenCV method.

```
# read image and show
img = cv2.imread('drive/My Drive/masks/001276_01_02_151.png')
# convert to RGB mode
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

plt.axis('off')
plt.imshow(img)
```

Figure 25: Converting BGR to RGB

The image was binarized to convert to grayscale with Otsu's binarization, Figure 26.

Noise in the image was removed and sure background and foreground of the image was realized, Figure 27.

### 3.4 OpenCV method

This method performs segmentation based on color therefore, it is essential to define the minimum and maximum values of the color to be segmented in RGB format, Figure 28.

Performing Otsu's binarization

```
[ ] # performing otsu's binarization
# converting to gray scale
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
print("Threshold limit: " + str(ret))

plt.axis('off')
plt.imshow(thresh, cmap = 'gray')
```

Figure 26: Otsu's binarization

```
# noise removal
kernel = np.ones((3, 3), np.uint8)
opening = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel, iterations = 2)

# sure background area
bkgr = cv2.dilate(opening, kernel, iterations = 3)

# sure foreground area
dist_transform = cv2.distanceTransform(opening,cv2.DIST_L2,5)
ret, frgr = cv2.threshold(dist_transform,0.7*dist_transform.max(),255,0)

# Finding unknown region
frgr = np.uint8(frgr)
unknown = cv2.subtract(bkgr,frgr)
```

Figure 27: Removing noise

```
blue_min = np.array([151, 151, 74], np.uint8) #Providing a wide range of yellow color
blue_max = np.array([255, 255, 74], np.uint8)
threshold_yellow_img = cv2.inRange(img_hsv, blue_min, blue_max)

# show threshold bits
threshold_yellow_img = cv2.cvtColor(threshold_yellow_img, cv2.COLOR_GRAY2RGB)
plt.subplot(121)
plt.imshow(threshold_yellow_img)
plt.title('Segmented Image')
masked_img = cv2.imread('drive/My Drive/masks/001276_01_02_151.png')
plt.subplot(122)
plt.imshow(masked_img)
plt.title('Expected Segmentation')
```

Figure 28: Color segmentation with OpenCV

## 4 Evaluation

### 4.0.1 Basic UNet

The metrics for basic UNet changed while training as shown in Figure 29.

Segmentations formed are shown in Figure 30.

Evaluation results are shown in Figure 31.

### 4.0.2 Proposed Model

Loss and metrics by using the least dilation rate (2 and 4) as per Figure 21 are shown in Figure 32. Segmentations are in Figure 33.

Further, using `brightness_range = [0.7, 1.0]` for augmentation of training images, `SpatialDropout2D(0.15)` showed loss as Figure 34.

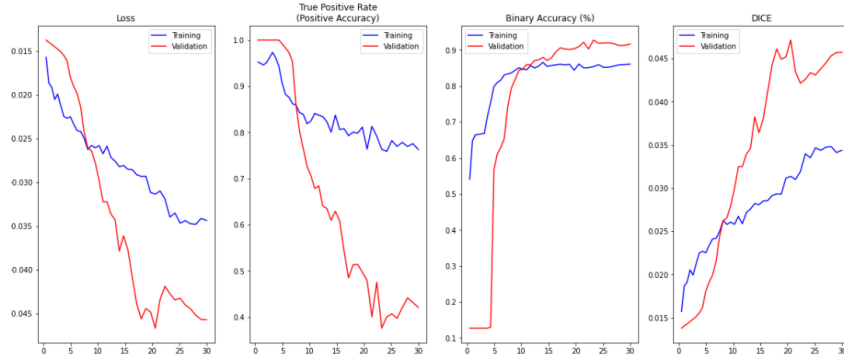


Figure 29: Loss for Basic UNet

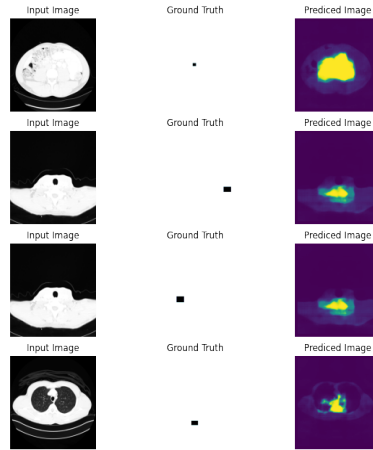


Figure 30: Segmentations with Basic UNet

```
print("test loss:", results[0])
print("test dice:", results[1])
print("test binary accuracy:", results[2])
print("test true positive rate:", results[3])
```

```
test loss: -0.05511686950922012
test dice: 0.055907074362039566
test binary accuracy: 0.9261903166770935
test true positive rate: 0.40928414463996887
```

Figure 31: Evaluation of Basic UNet

Furthermore, two other combinations of dilation rate were tested. Dilation rate of (2,4 and 6) yielded as Figure 35. Segmentations are in Figure 36.

Lastly, combination of dilation rate (2, 4, 6, 8, 12 and 18) performed as Figure 37. Segmentations are in Figure 38.

Evaluation results are shown in Figure 39.



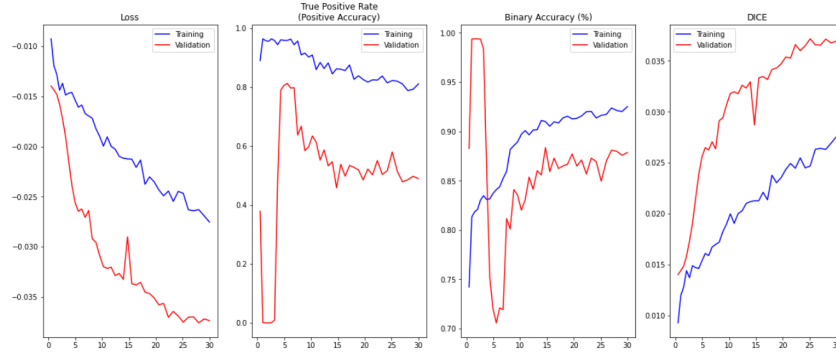


Figure 32: Dilation rates 2 and 4

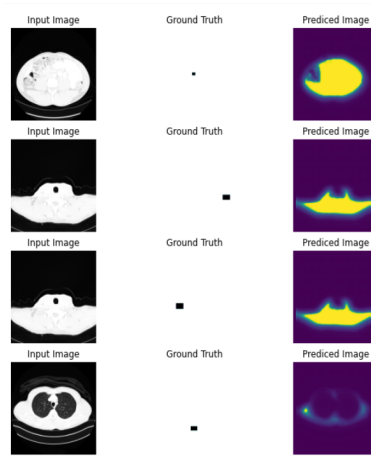


Figure 33: Segmentations with dilation rates 2 and 4

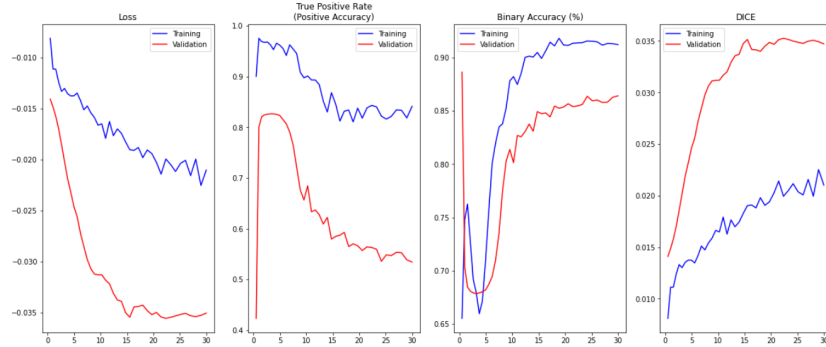


Figure 34: Changed Augmentation and SpatialDropout

#### 4.0.3 Watershed Algorithm

The segmentations formed by Watershed are marked by green lines in Figure 40.

#### 4.0.4 OpenCV

OpenCV formed segmented lesions as in Figure 41.

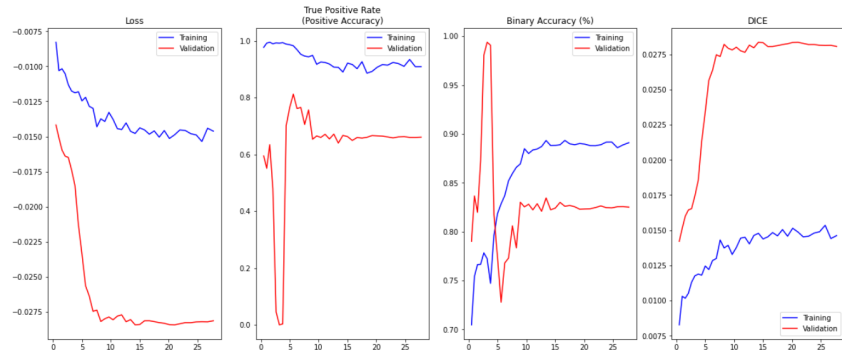


Figure 35: Dilation rates 2,4 and 6

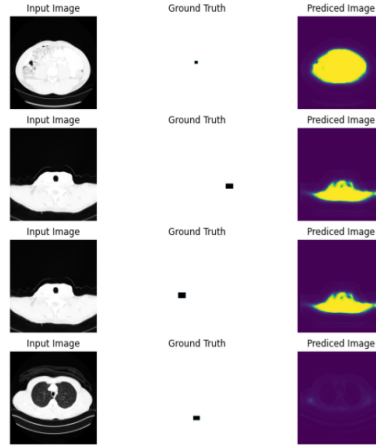


Figure 36: Segmentations with dilation rates 2,4 and 6

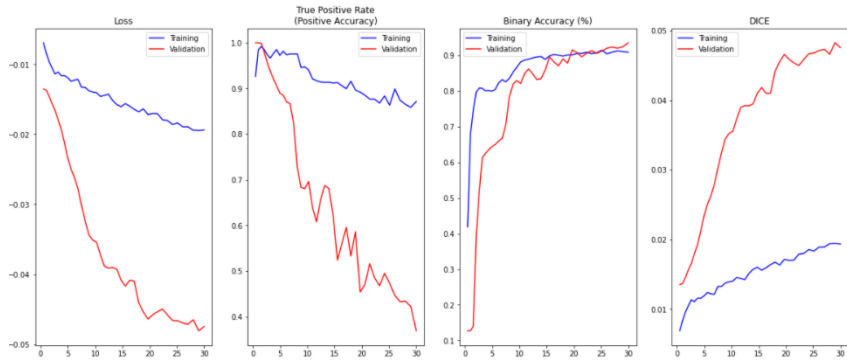


Figure 37: Dilation rates 2, 4, 6, 8, 12 and 18

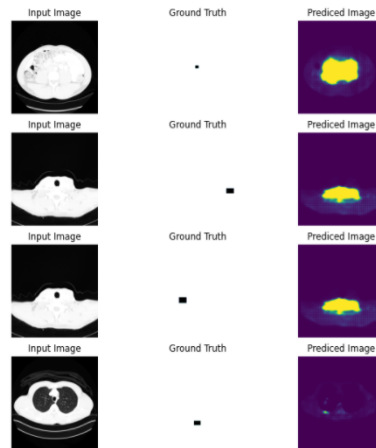


Figure 38: Segmentations with dilation rates 2, 4, 6, 8, 12 and 18

```
print("test loss:", results[0])
print("test dice:", results[1])
print("test binary accuracy:", results[2])
print("test true positive rate:", results[3])
```

```
test loss: -0.030674900859594345
test dice: 0.03085225820541382
test binary accuracy: 0.7954323887825012
test true positive rate: 0.7507731318473816
```

Figure 39: Evaluation of proposed model

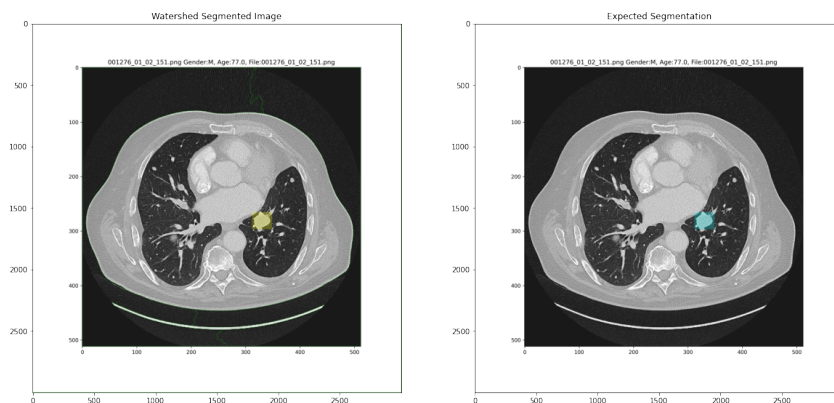


Figure 40: Segmentations with Watershed

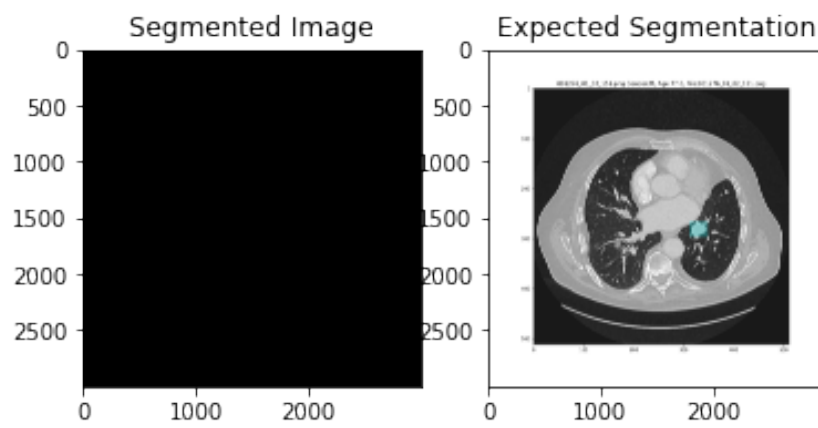


Figure 41: Segmentations with OpenCV

## References

- Colab.research.google.com*. 2020. *Google Colaboratory*. (n.d.).  
**URL:** <https://colab.research.google.com/notebooks/intro.ipynb>
- Docs.opencv.org*. 2020. *Opencv: Changing Colorspaces*. (n.d.).  
**URL:** [https://docs.opencv.org/master/df/d9d/tutorial\\_py\\_colorspaces.html](https://docs.opencv.org/master/df/d9d/tutorial_py_colorspaces.html)
- GitHub*. 2020. *Krshrimali/Image-Segmentation-Using-Opencv-And-CNN* (n.d.).  
**URL:** <https://github.com/krshrimali/Image-Segmentation-using-OpenCV-and-CNN/blob/master/ImageSegmentationLung.ipynb>
- Kaggle.com*. 2020. *Deeplesion Overview* (n.d.).  
**URL:** <https://www.kaggle.com/kmader/deeplesion-overview>
- Piratefsh.github.io*. 2020. *Image Processing 101* (n.d.).  
**URL:** <https://piratefsh.github.io/image-processing-101/>
- TensorFlow*. 2020. *Module: Tf.Keras.Layers — Tensorflow Core V2.3.1*. (n.d.).  
**URL:** [https://www.tensorflow.org/api\\_docs/python/tf/keras/layers/](https://www.tensorflow.org/api_docs/python/tf/keras/layers/)
- Yan, K., Wang, X., Lu, L. and Summers, R. M. (2018). Deeplesion: automated mining of large-scale lesion annotations and universal lesion detection with deep learning, *Journal of Medical Imaging* **5**(3): 036501.
- Zhang, T., Gao, H., Xing, Y., Chen, Z. and Zhang, L. (n.d.). Dualres-unet: Limited angle artifact reduction for computed tomography, *2019 IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC)*, IEEE, pp. 1–3.