Deep RNAs applied to Rats tibia

Vitor Souza Premoli Pinto de Oliveira*[1]
Eduardo Destefani Stefanato [1]
Prof. Dr. Christiano Jorge Gomes Pinheiro[1]
Prof. Dr. Anderson Alvarenga de Moura Meneses[2]

[1]Federal University of Espírito Santo
[2]Federal University of Oeste do Pará

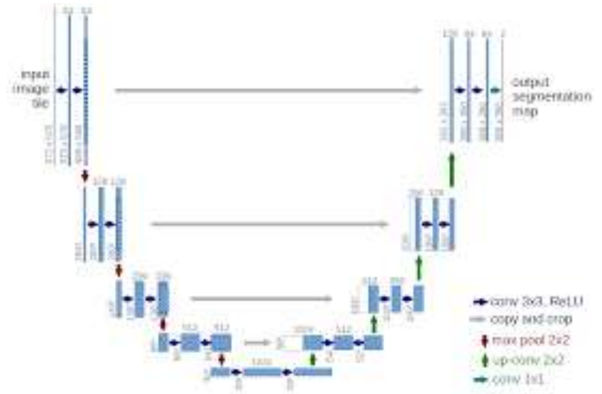
Objectives

1. Introduction;
   I. Rats tibia segmentation;

2. Theoretical Background;
   I. Data acquisition: SR-uCT;
   II. U-net: Convolutional Networks;

3. Methodology;
   I. Manual segmentation;
   II. Choosing Architecture;
   III. Hyperparameters;
   IV. Data augmentation and Cross-validation;

4. Conclusion;
   I. Results and discussion;
   II. Future objectives.

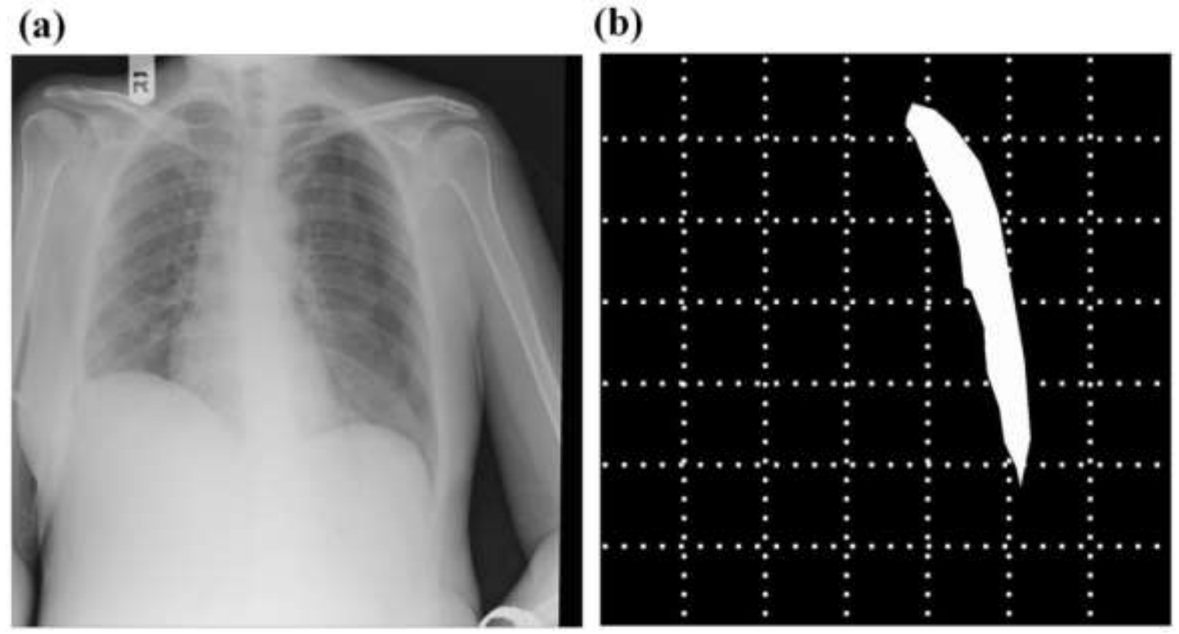


DEEP RNAS APPLIED TO RATS TIBIA

Introduction

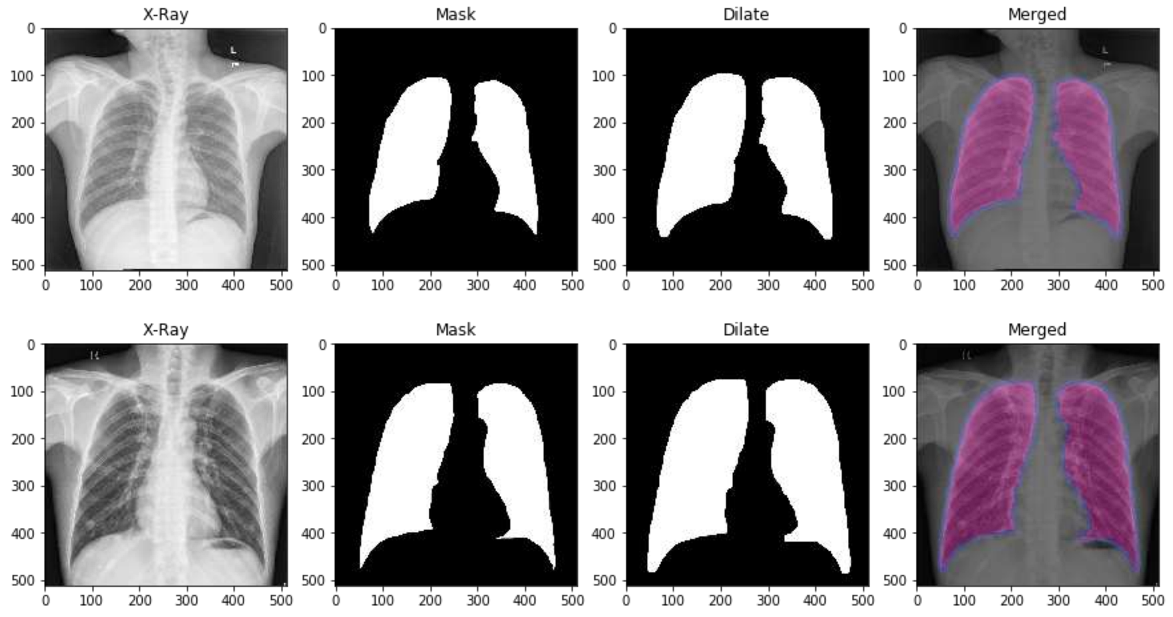
Biomedical images

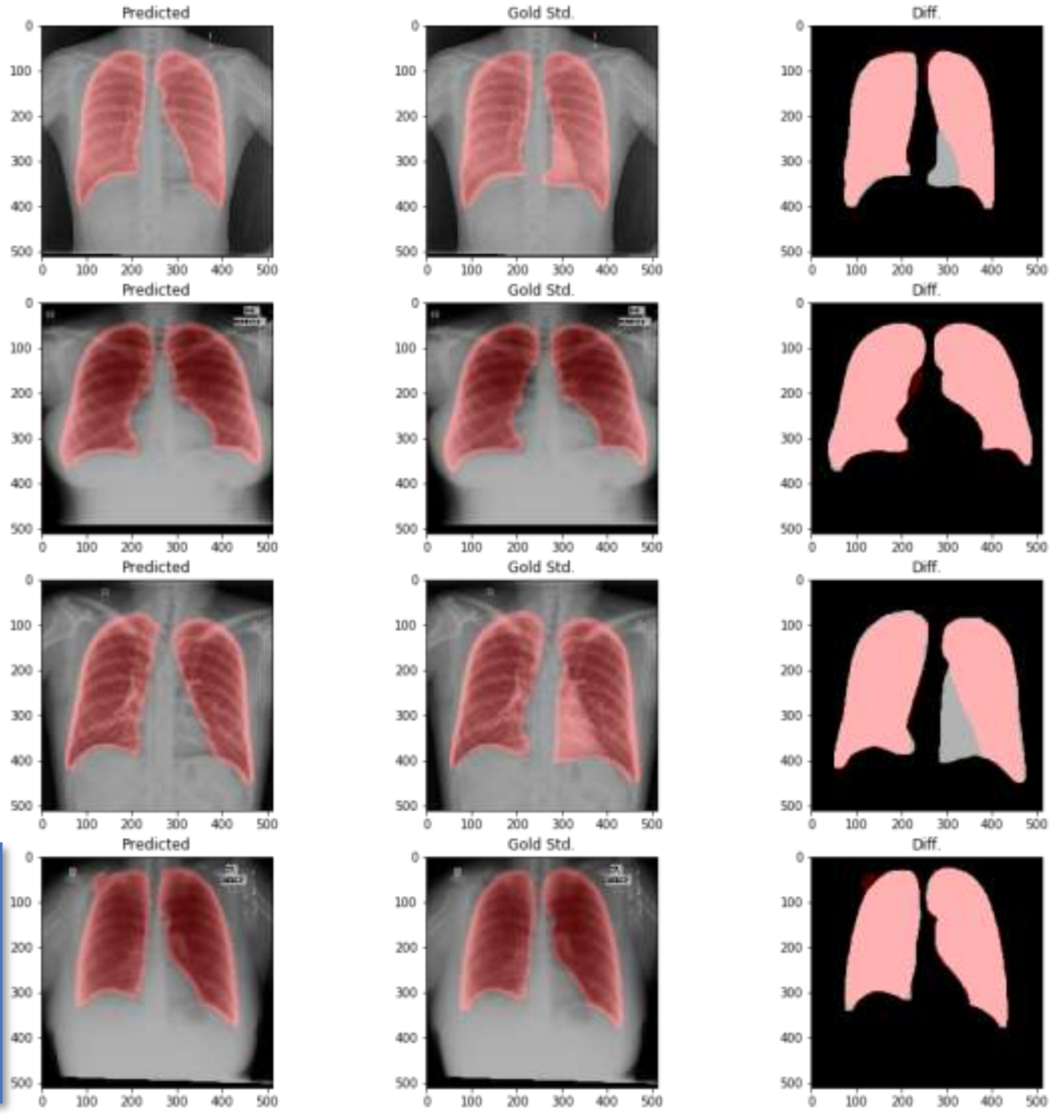
Several researches using image segmentation have been carried out in several areas. Among them, we can mention the medical area, where segmentations can be applied to delimit in the image regions such as tumors, cells, glands, organs, cellular tissues, among others. Several segmentation methods can be developed specifically to delimit each of these elements.



Example of chest X-ray image processing for deep-learning methods. (**a**) Original chest X-ray image. (**b**) Image obtained after marking the location of pneumothorax in white and the remaining areas in black (CHO, 2021) .

Biomedical images



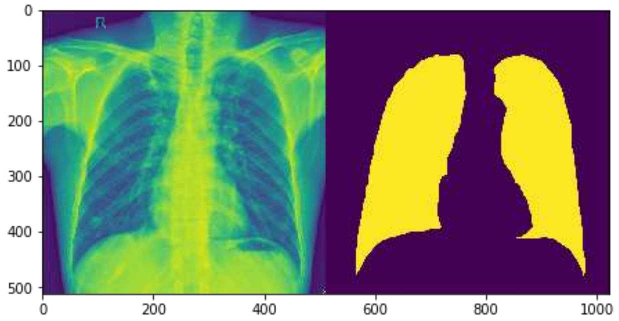
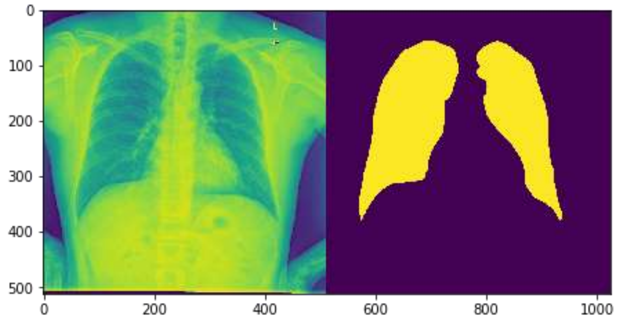
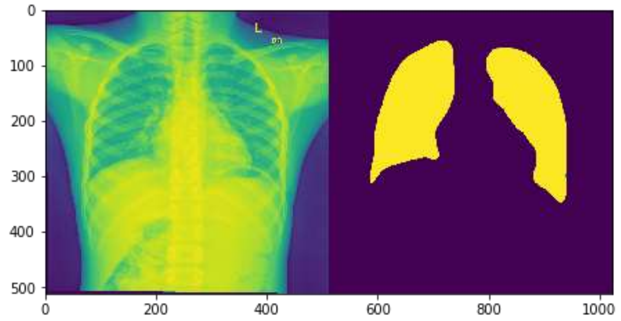


U-Net lung segmentation (Montgomery + Shenzhen)

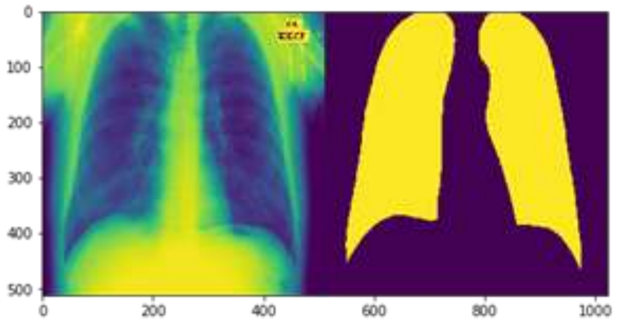
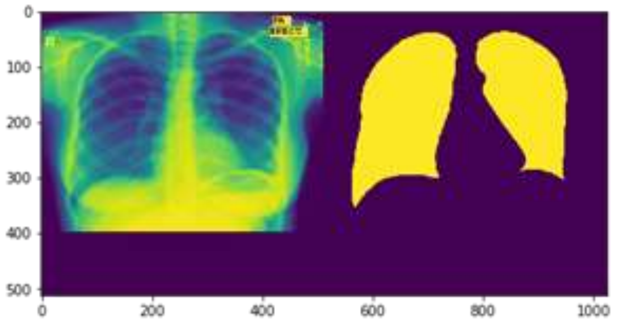
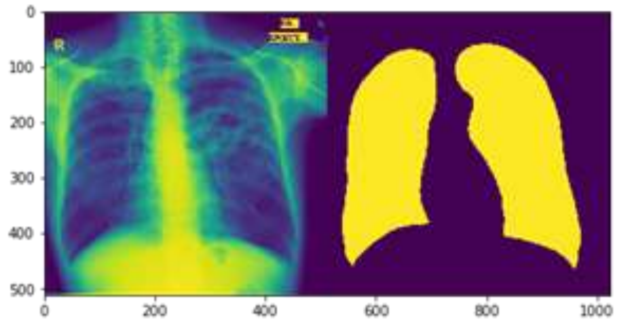
https://www.kaggle.com/eduardomineo/u-net-lung-segmentation-montgomery-shenzhen/notebook

Biomedical images

Training set

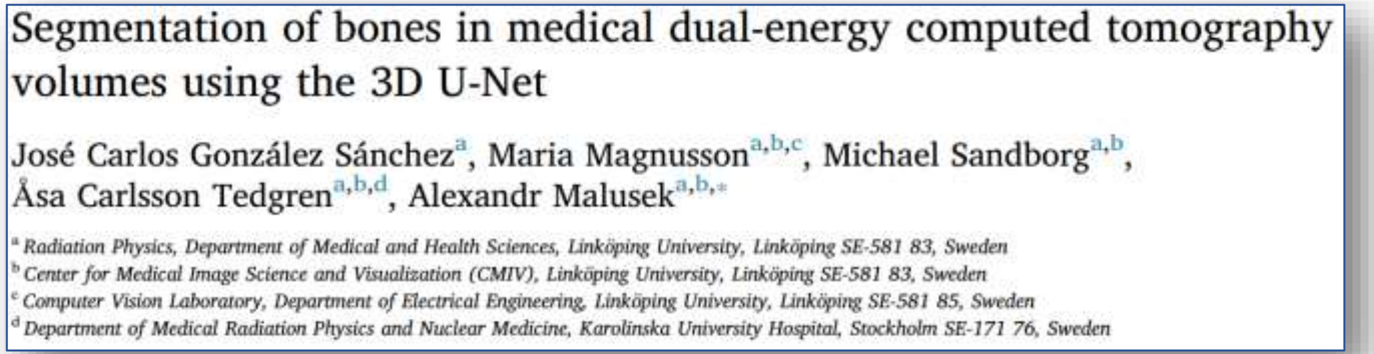
Testing set

Lung segmentation from Chest X-Ray dataset

https://www.kaggle.com/nikhilpandey360/lung-segmentation-from-chest-x-ray-dataset/notebook

Biomedical images

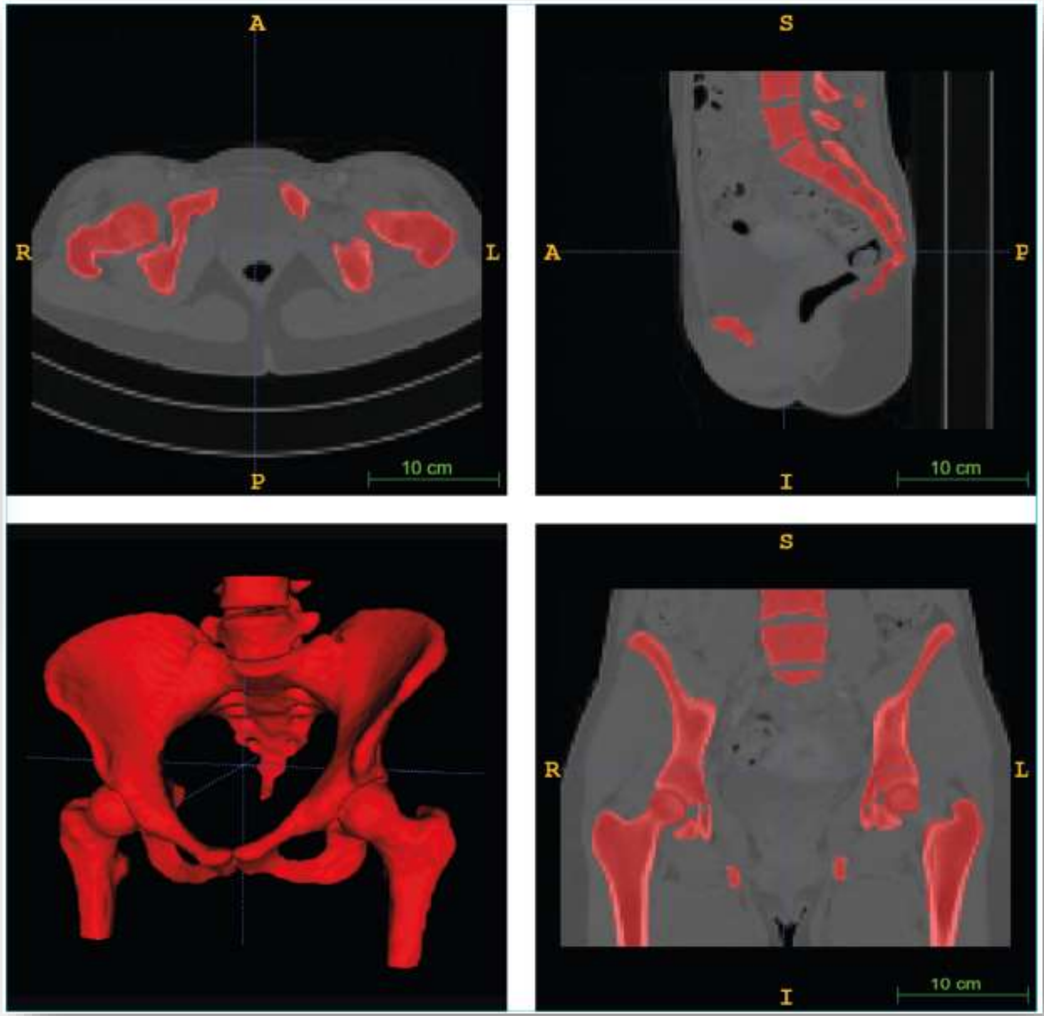


Segmentation of bones in medical dual-energy computed tomography volumes using the 3D U-Net

José Carlos González Sánchez[a], Maria Magnusson[a,b,c], Michael Sandborg[a,b], Åsa Carlsson Tedgren[a,b,d], Alexandr Malusek[a,b,*]

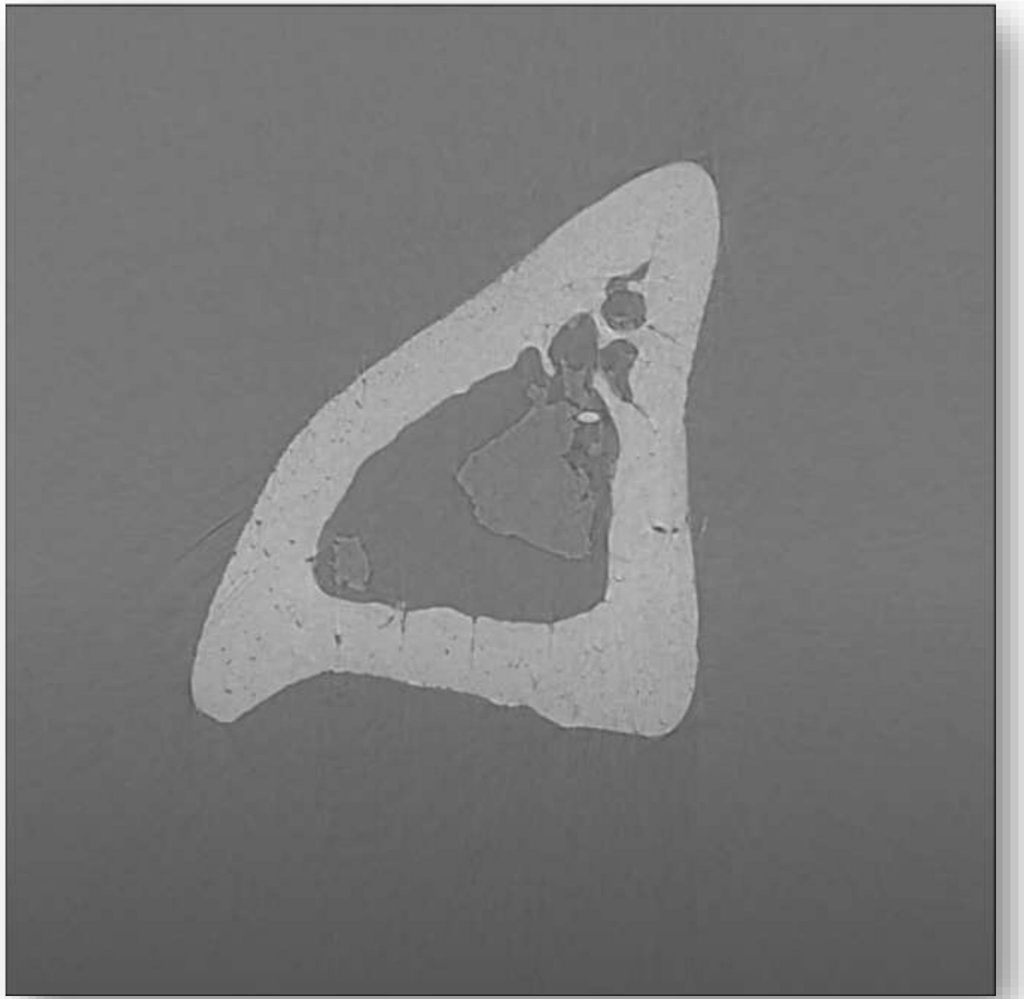
[a] Radiation Physics, Department of Medical and Health Sciences, Linköping University, Linköping SE-581 83, Sweden
[b] Center for Medical Image Science and Visualization (CMIV), Linköping University, Linköping SE-581 83, Sweden
[c] Computer Vision Laboratory, Department of Electrical Engineering, Linköping University, Linköping SE-581 85, Sweden
[d] Department of Medical Radiation Physics and Nuclear Medicine, Karolinska University Hospital, Stockholm SE-171 76, Sweden

(SÁNCHEZ, 2020)

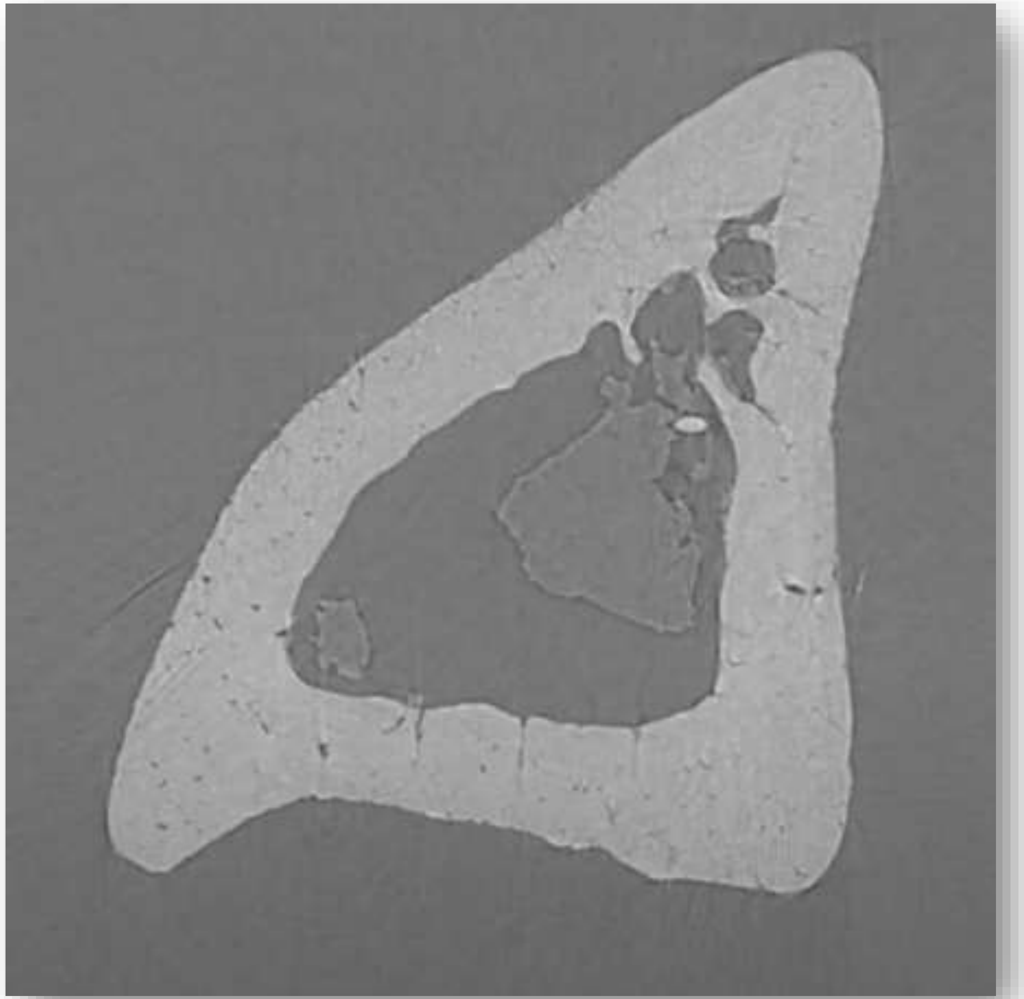


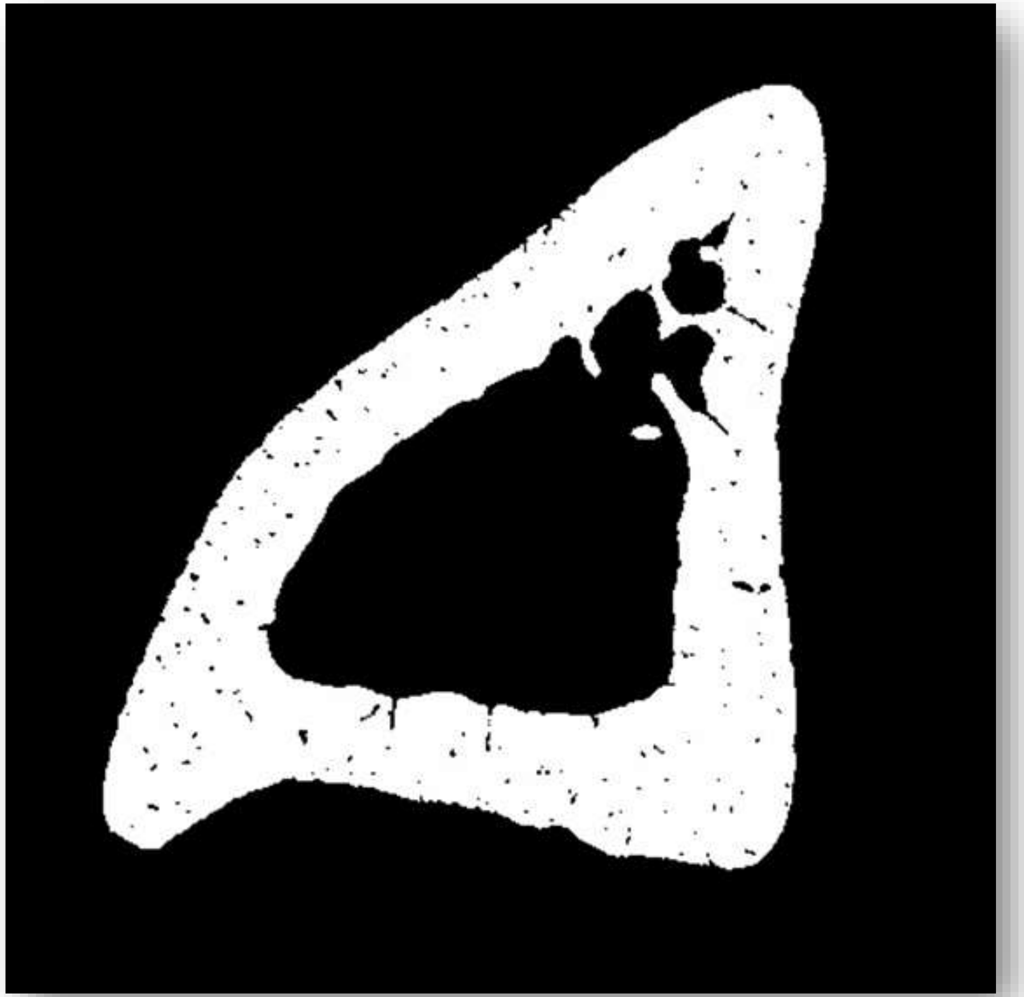
Rats tibia segmentation

Manual segmentation methods can be very tedious, time-consuming and subject to inter and intraindividual variability (TINGELHOF, 2008).



Rats tibia segmentation



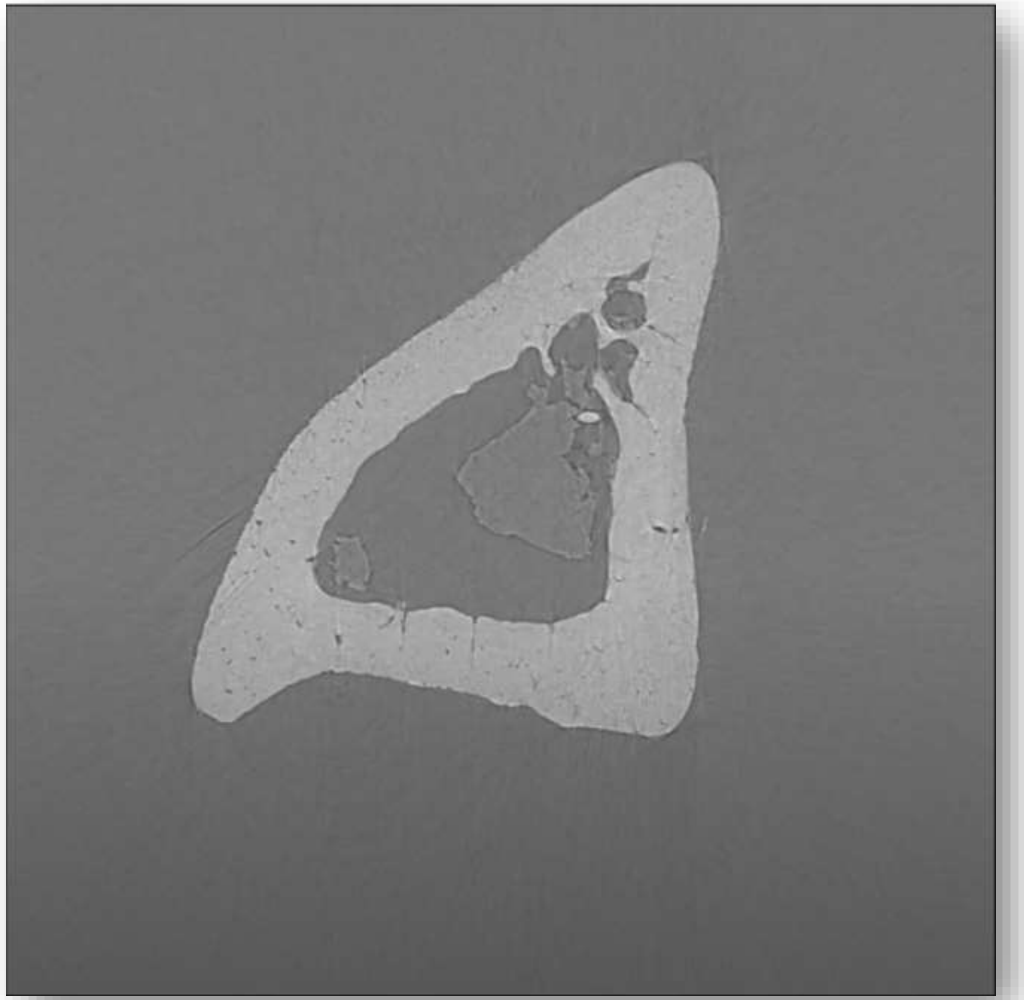


Rats tibia segmentation

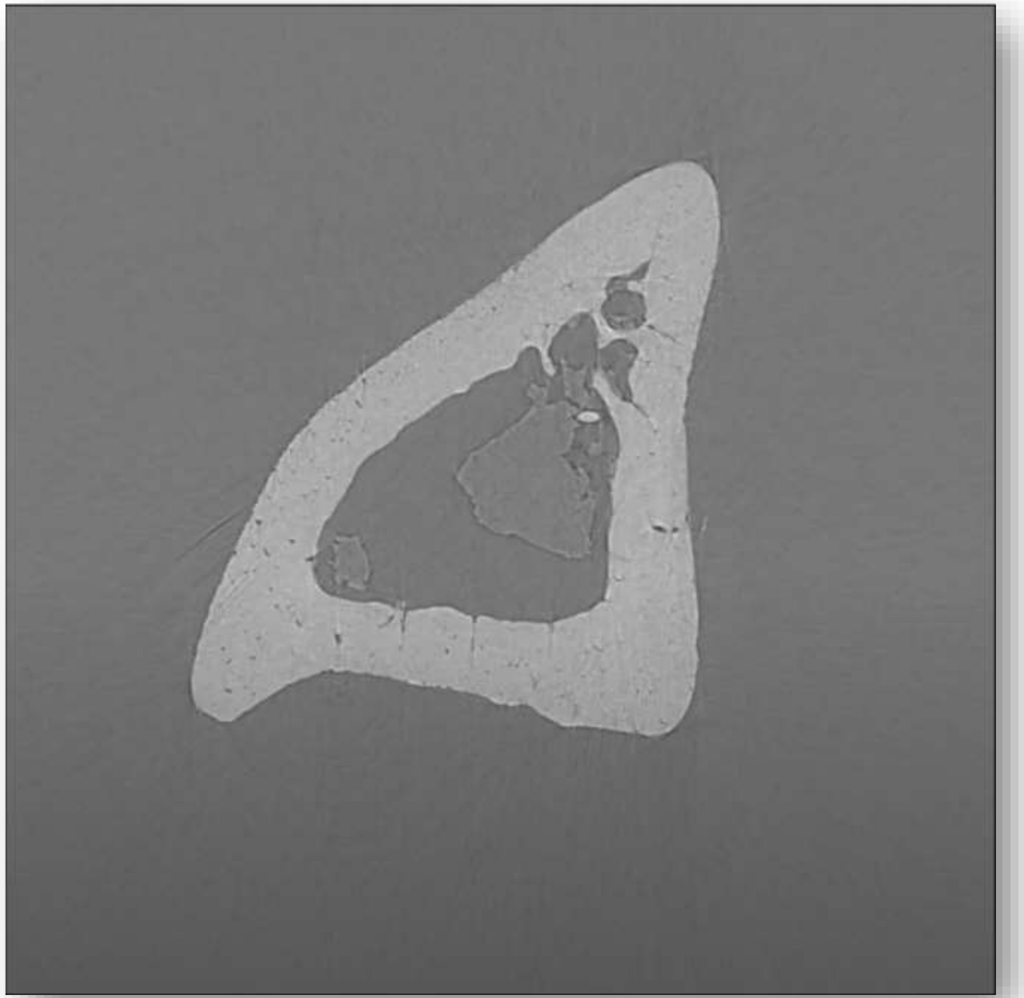
Manual segmentation methods can be very tedious, time-consuming and subject to inter and intraindividual variability (TINGELHOF, 2008).

Automatic segmentation methods can overcome these problems, nevertheless their performance is in many cases adversely affected by low tissue contrast and image artifacts. Traditional segmentation methods are often based on conventional computer vision and machine learning approaches(GONZALEZ & WOODS, 2018)(SÁNCHEZ, 2020).

Recently, methods based on deep learning have demonstrated a potential to notably outperform the traditional methods in medical image analysis (SÁNCHEZ, 2020).



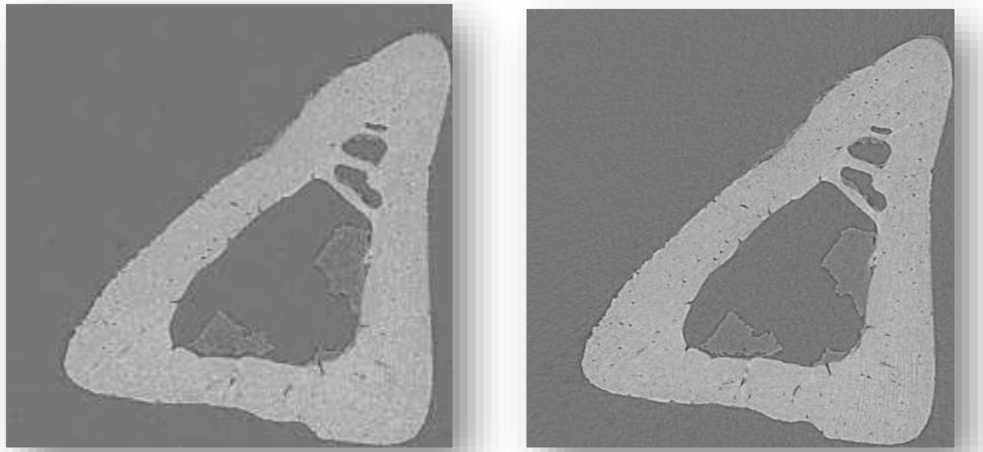
is it possible to segment the rat tibia image by neural networks?

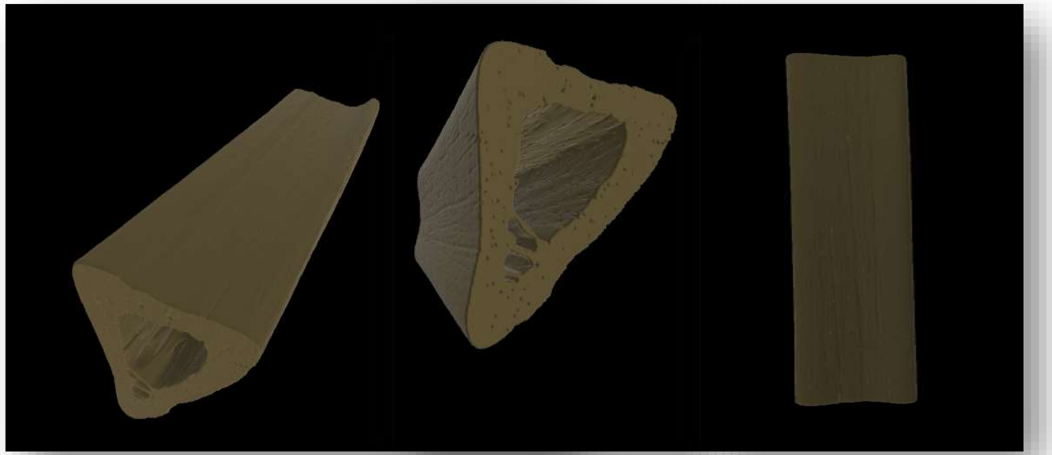


DEEP RNAS APPLIED TO RATS TIBIA

Theoretical Background

Data acquisition: SR-uCT

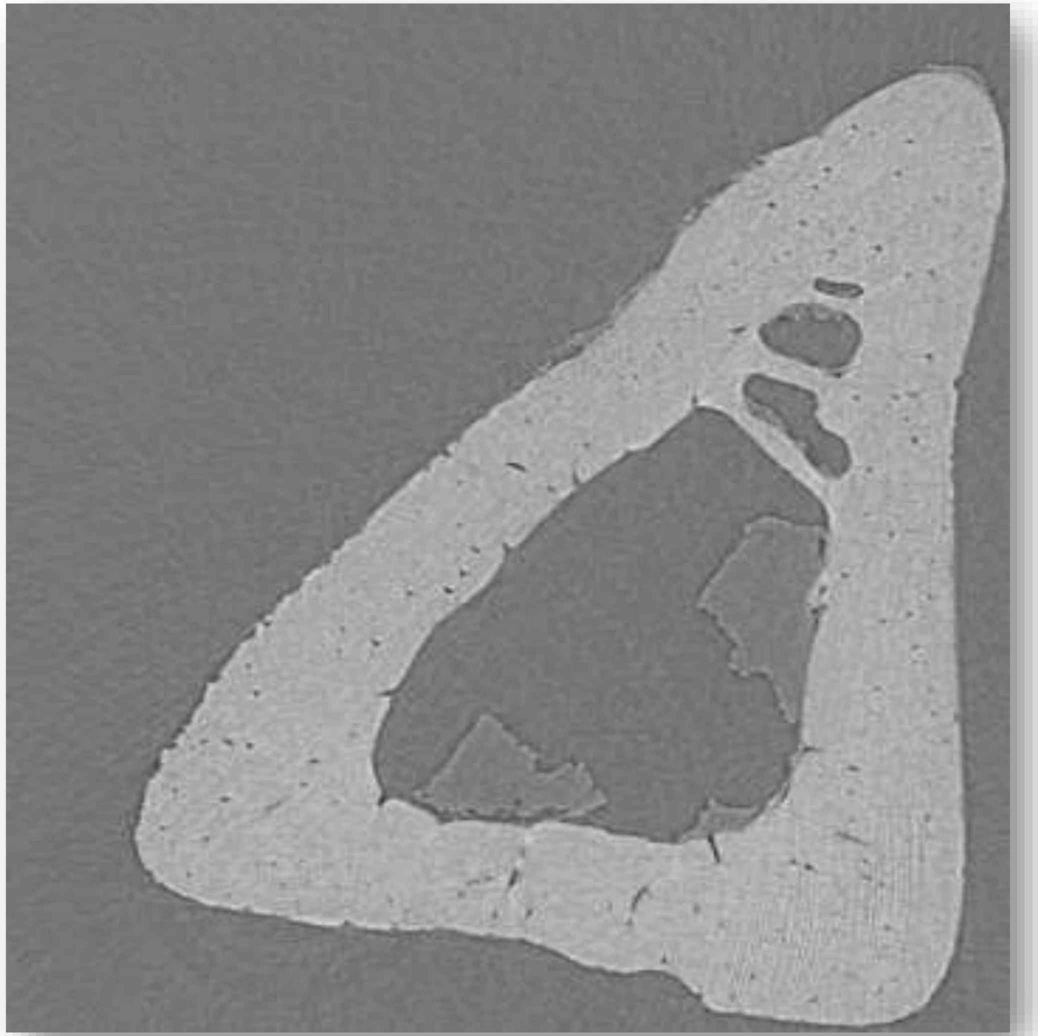




The reconstructed tibial slices. Above, the slices and below they are used to create the 3D volume

- Synchrotron radiation is produced by ring accelerators in which a pulsed stream of high-energy electrons circulates at nearly the speed of light;
- The main characteristic of these synchrotron radiation sources is the large and continuous energy spectrum that provides a high photon flux over an energy range of up to 50 keV or greater;
- The beam has a high natural collimation and a high degree of coherence in space and time. A non-destructive technique that allows visualization of the internal structure of objects;
- According to Abrami (2005), these characteristics in combination with the sophisticated optics differentiate these sources from standard clinical and research tools.

Data acquisition: SR-uCT



Data acquisition: SR-uCT

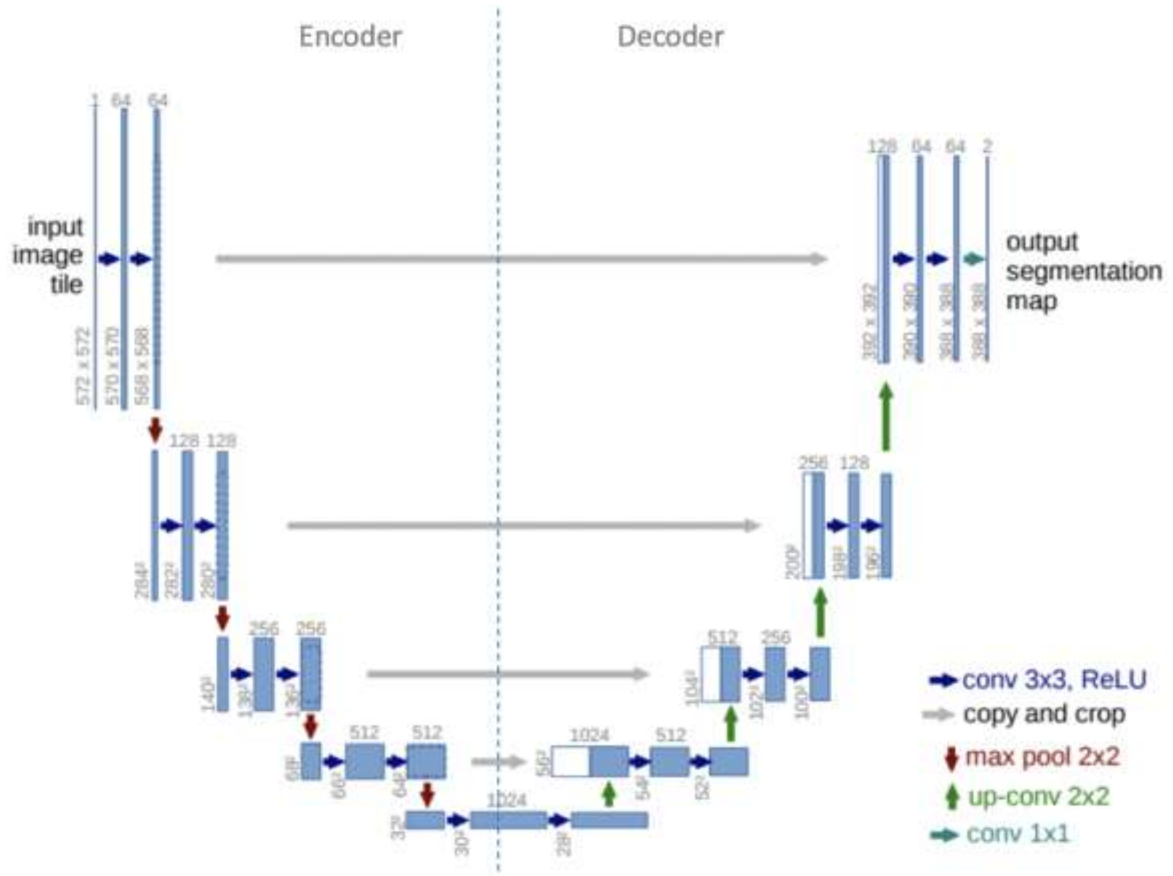
Elettra Sincrotrone Trieste

- According to Meneses (2018), these SR sources provide higher intensity and spatial coherence and therefore higher spatial resolution. It is through these facilities allowed for higher quality images compared to conventional x-ray imaging;

- As per to Lewis (2004) and Attwood (2007), the technique provides better image enhancement and detail and often at lower doses when compared to conventional X-ray techniques.

U-net: Convolutional Networks

Better segmentation results were achieved when the encoder and decoder parts were organized in levels and connected with skip connections. This design known as the U-Net was first implemented by Ronneberger et al (2015), in 2D and later extended to 3D by Çiçek (2016).

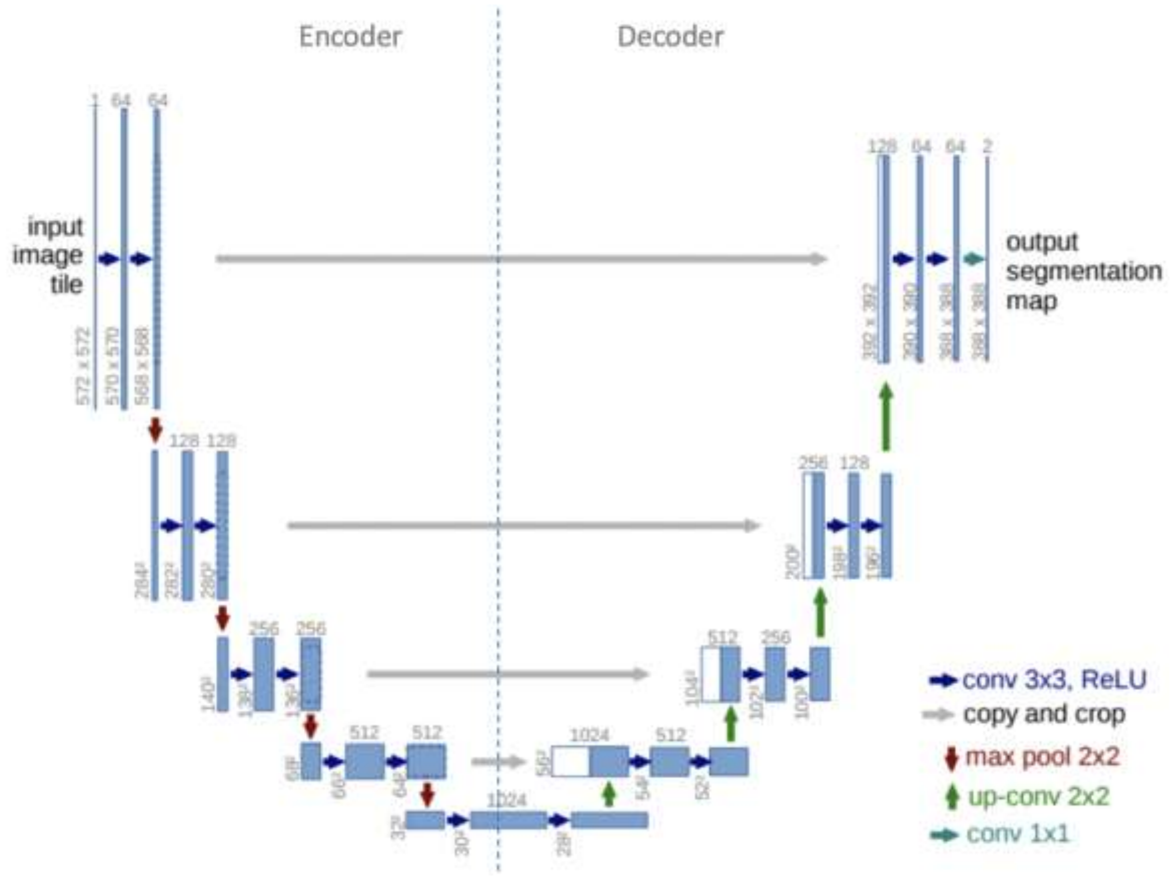


U-net architecture. Blue boxes represent multi-channel feature maps, while while boxes represent copied feature maps. The arrows of different colors represent different operations

U-net: Convolutional Networks

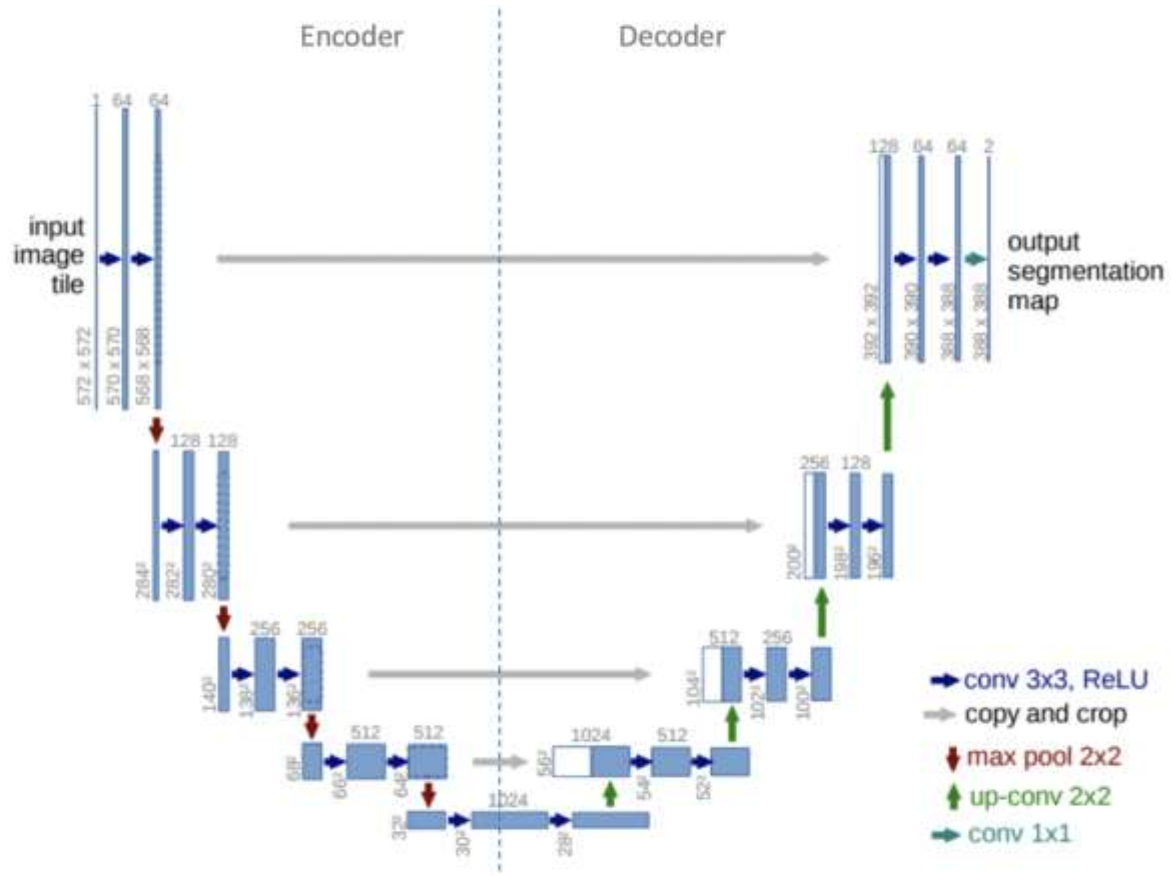
UNet, evolved from the traditional Convolutional Neural Network (CNN), was first designed and applied in 2015 to process biomedical images (ZHANG, 2022).

As a general convolutional neural network focuses its task on image classification, where input is an image and output is one label, but in biomedical cases, it requires us not only to distinguish whether there is a disease, but also to localise the area of abnormality.



U-net architecture. Blue boxes represent multi-channel feature maps, while while boxes represent copied feature maps. The arrows of different colors represent different operations

U-net: Convolutional Networks



U-net architecture. Blue boxes represent multi-channel feature maps, while while boxes represent copied feature maps. The arrows of different colors represent different operations

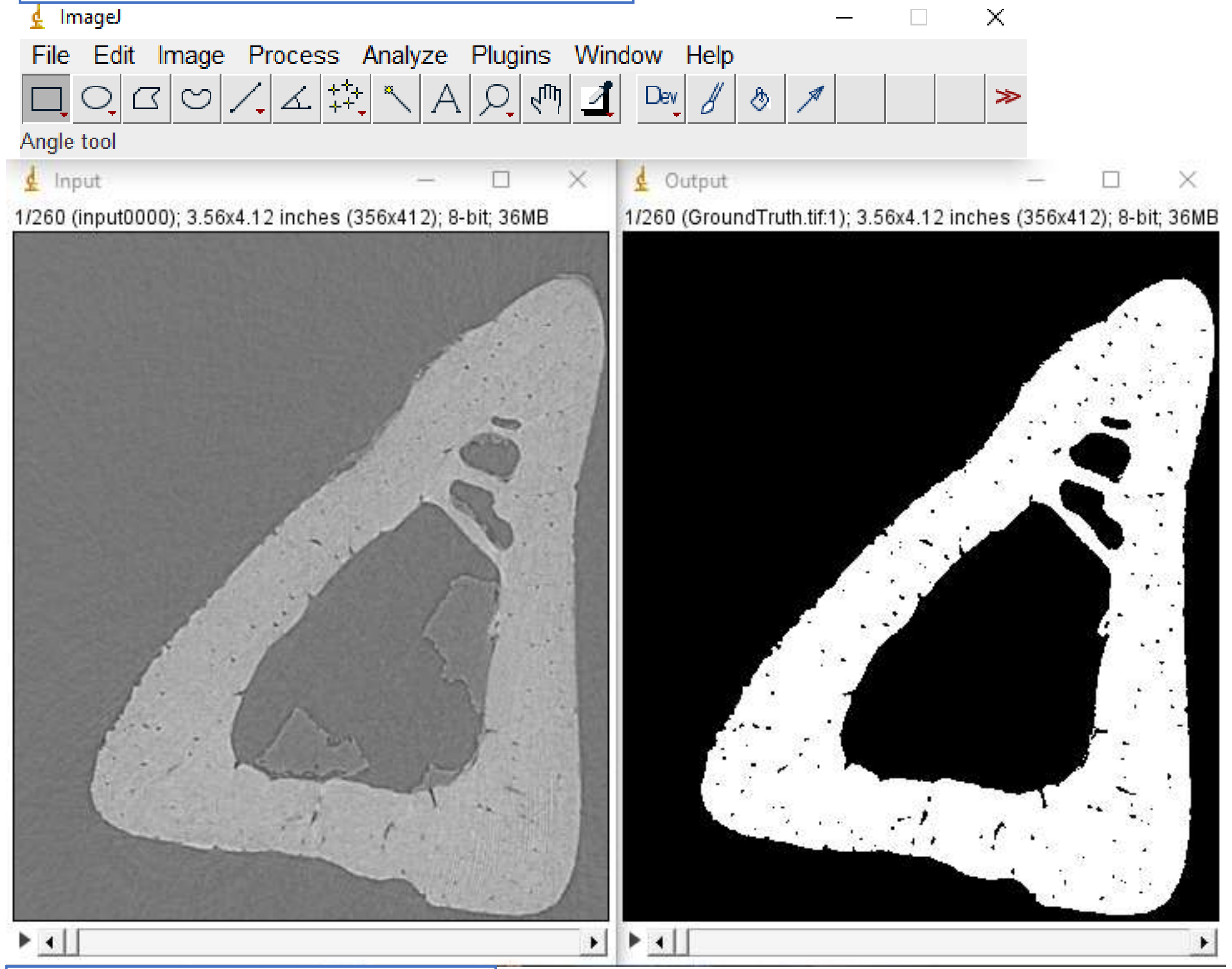
- The network architecture was developed for biomedical images, and consequently has a higher performance for these types of data (RONNEBERGER ET AL, 2015), (MONTE, 2021), (ÇIÇEK , 2016);
- Due to the limited availability of medical images, this neural network was created with the purpose of obtaining efficient results using a small amount of data (MONTE, 2021);
- A significant amount of recent work are using this architecture. (SÁNCHEZ, 2020), (MAIER, 2019) (BREININGE, 2018), (CHEN, 2018) as representative examples;
- Many examples using U-net and easy availability of the code either through Kaggle or Github.

DEEP RNAS APPLIED TO RATS TIBIA

Methodology

Manual segmentation

Training and validation sets



1/260 (input0000); 3.56x4.12 inches (356x412); 8-bit; 36MB

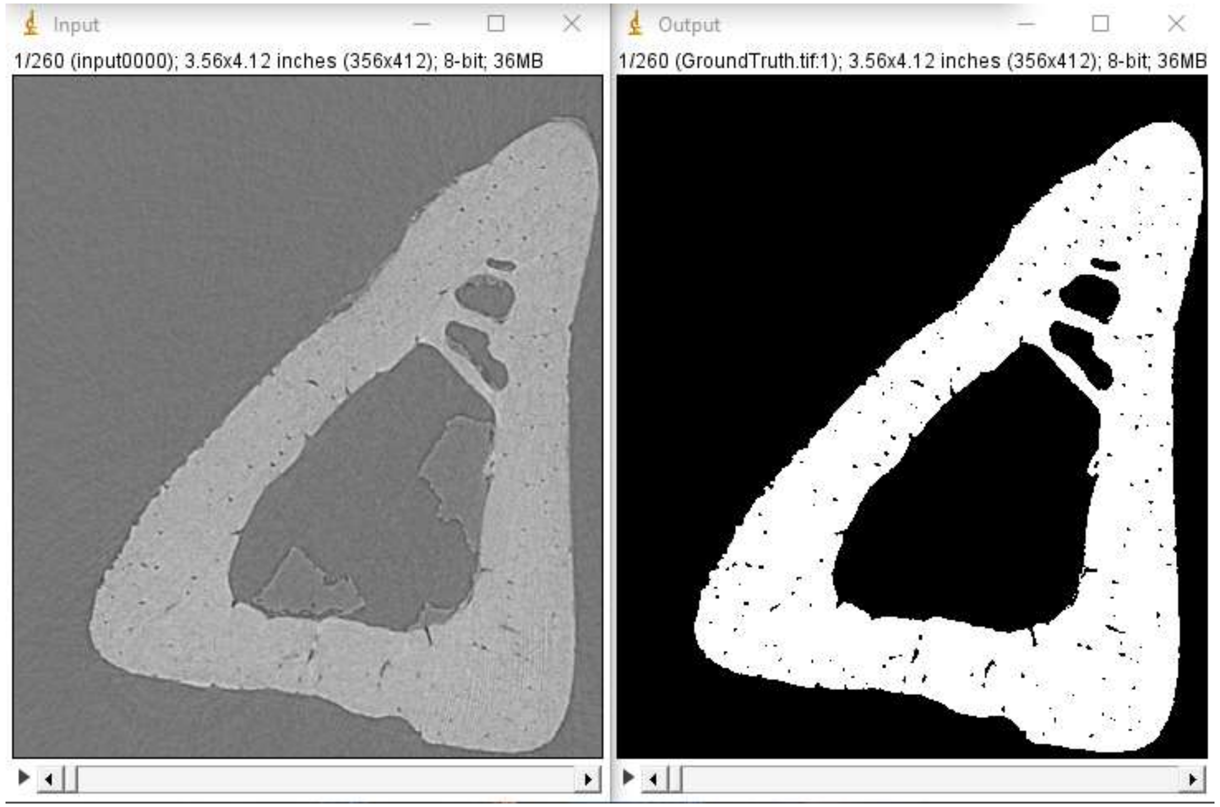
1/260 (GroundTruth.tif:1); 3.56x4.12 inches (356x412); 8-bit; 36MB

Rat_24.tif – 260 slices

ImageJ
Image Processing and Analysis in Java

Manual segmentation

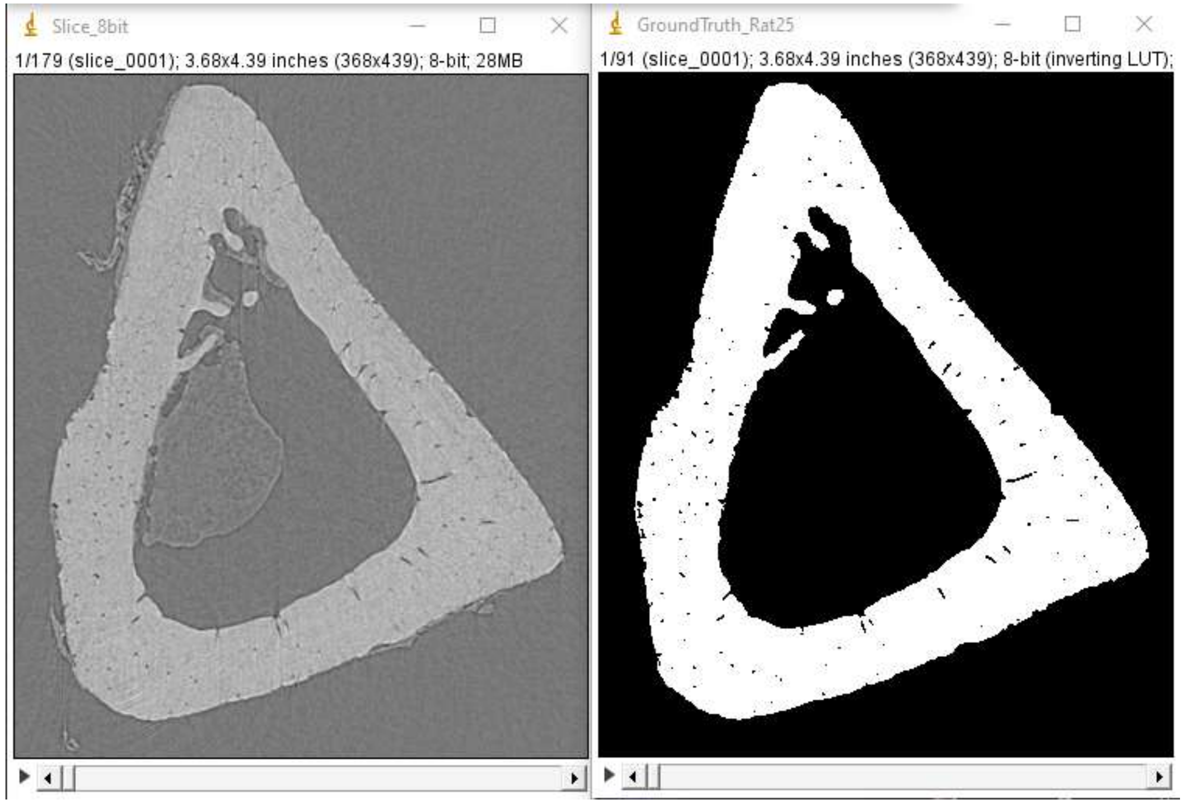
Training and validation sets


Input
1/260 (input0000); 3.56x4.12 inches (356x412); 8-bit; 36MB

Output
1/260 (GroundTruth.tif:1); 3.56x4.12 inches (356x412); 8-bit; 36MB

Rat_24.tif – 260 slices

Test set

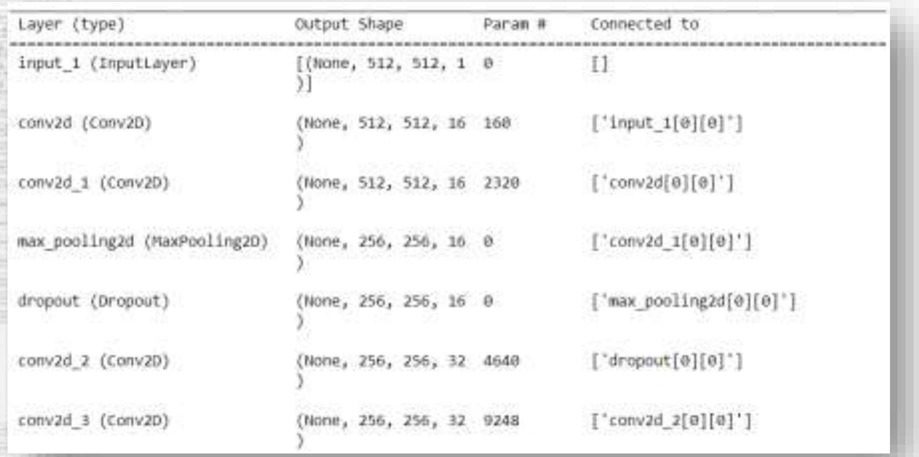

Slice_8bit
1/179 (slice_0001); 3.68x4.39 inches (368x439); 8-bit; 28MB

GroundTruth_Rat25
1/91 (slice_0001); 3.68x4.39 inches (368x439); 8-bit (inverting LUT);

Rat_25.tif – 179 slices

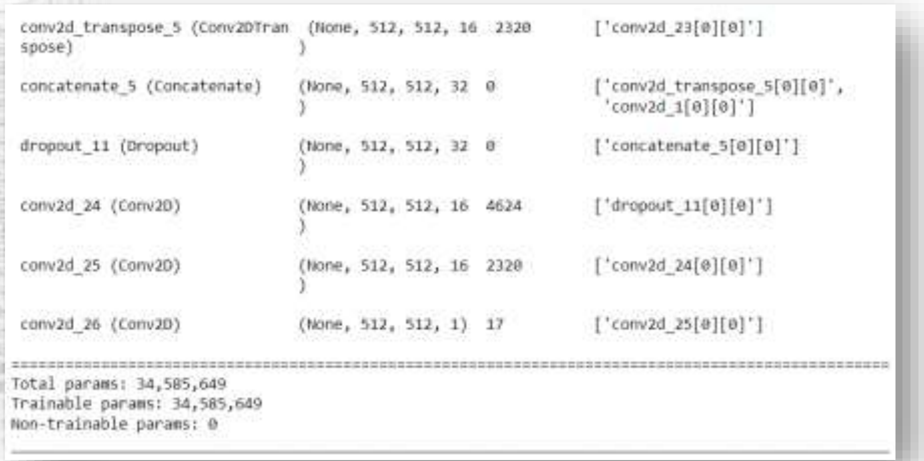
Choosing Architecture

Unet_Model_V2 – 12 layers



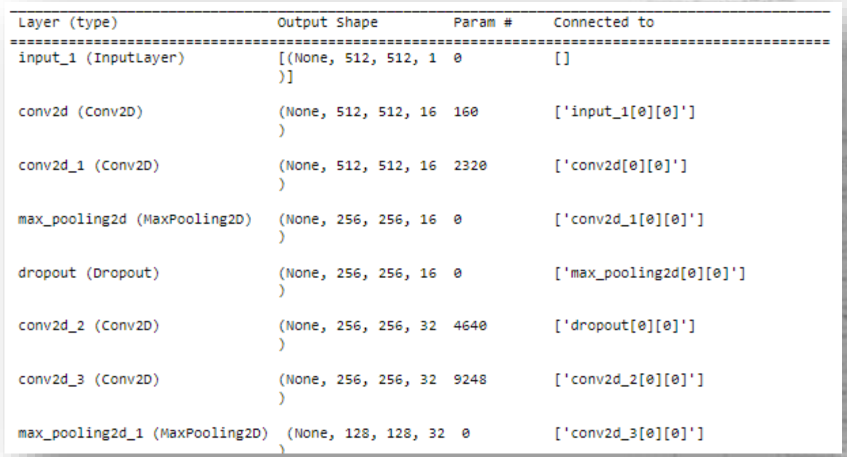
```
Layer (type)                 Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)         [(None, 512, 512, 1  0           []
                             )]

conv2d (Conv2D)              (None, 512, 512, 16  160         ['input_1[0][0]']
                             )

conv2d_1 (Conv2D)            (None, 512, 512, 16  2320        ['conv2d[0][0]']
                             )

max_pooling2d (MaxPooling2D) (None, 256, 256, 16  0           ['conv2d_1[0][0]']
                             )

dropout (Dropout)            (None, 256, 256, 16  0           ['max_pooling2d[0][0]']
                             )

conv2d_2 (Conv2D)            (None, 256, 256, 32  4640        ['dropout[0][0]']
                             )

conv2d_3 (Conv2D)            (None, 256, 256, 32  9248        ['conv2d_2[0][0]']
                             )
```

⋮ ⋮ ⋮



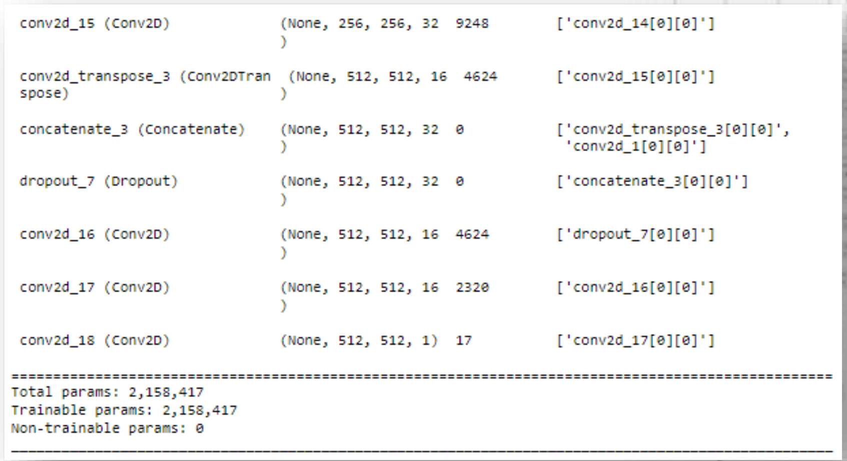
```
conv2d_transpose_5 (Conv2DTran  (None, 512, 512, 16  2320     ['conv2d_23[0][0]']
spose)                          )

concatenate_5 (Concatenate)     (None, 512, 512, 32  0        ['conv2d_transpose_5[0][0]',
                                )                              'conv2d_1[0][0]']

dropout_11 (Dropout)            (None, 512, 512, 32  0        ['concatenate_5[0][0]']
                                )

conv2d_24 (Conv2D)              (None, 512, 512, 16  4624     ['dropout_11[0][0]']
                                )

conv2d_25 (Conv2D)              (None, 512, 512, 16  2320     ['conv2d_24[0][0]']
                                )

conv2d_26 (Conv2D)              (None, 512, 512, 1)  17       ['conv2d_25[0][0]']

==================================================================================================
Total params: 34,585,649
Trainable params: 34,585,649
Non-trainable params: 0
```

Unet_Model_V1 – 8 layers



```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 512, 512, 1  0           []
                                )]

conv2d (Conv2D)                 (None, 512, 512, 16  160         ['input_1[0][0]']
                                )

conv2d_1 (Conv2D)               (None, 512, 512, 16  2320        ['conv2d[0][0]']
                                )

max_pooling2d (MaxPooling2D)    (None, 256, 256, 16  0           ['conv2d_1[0][0]']
                                )

dropout (Dropout)               (None, 256, 256, 16  0           ['max_pooling2d[0][0]']
                                )

conv2d_2 (Conv2D)               (None, 256, 256, 32  4640        ['dropout[0][0]']
                                )

conv2d_3 (Conv2D)               (None, 256, 256, 32  9248        ['conv2d_2[0][0]']
                                )

max_pooling2d_1 (MaxPooling2D)  (None, 128, 128, 32  0           ['conv2d_3[0][0]']
                                )
```

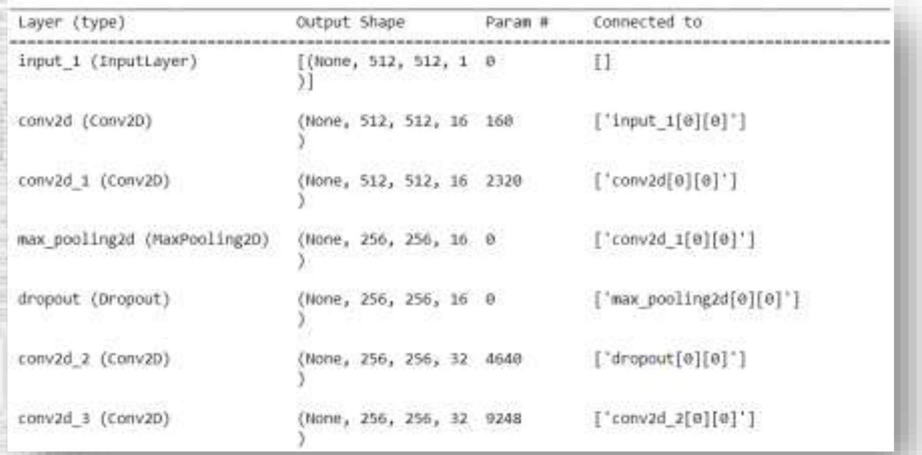
⋮ ⋮ ⋮



```
conv2d_15 (Conv2D)              (None, 256, 256, 32  9248        ['conv2d_14[0][0]']
                                )

conv2d_transpose_3 (Conv2DTran  (None, 512, 512, 16  4624        ['conv2d_15[0][0]']
spose)                          )

concatenate_3 (Concatenate)     (None, 512, 512, 32  0           ['conv2d_transpose_3[0][0]',
                                )                                 'conv2d_1[0][0]']

dropout_7 (Dropout)             (None, 512, 512, 32  0           ['concatenate_3[0][0]']
                                )

conv2d_16 (Conv2D)              (None, 512, 512, 16  4624        ['dropout_7[0][0]']
                                )

conv2d_17 (Conv2D)              (None, 512, 512, 16  2320        ['conv2d_16[0][0]']
                                )

conv2d_18 (Conv2D)              (None, 512, 512, 1)  17          ['conv2d_17[0][0]']

==================================================================================================
Total params: 2,158,417
Trainable params: 2,158,417
Non-trainable params: 0
__________________________________________________________________________________________________
```

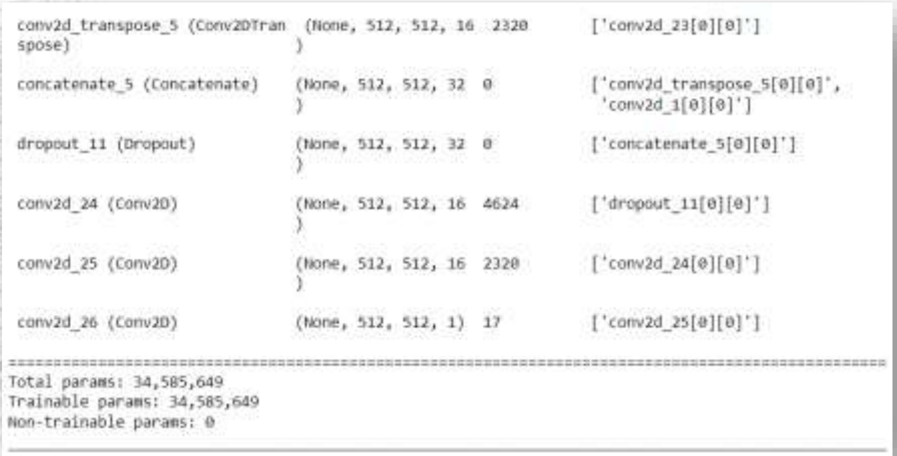
Choosing Architecture

Unet_Model_V2 – 12 layers

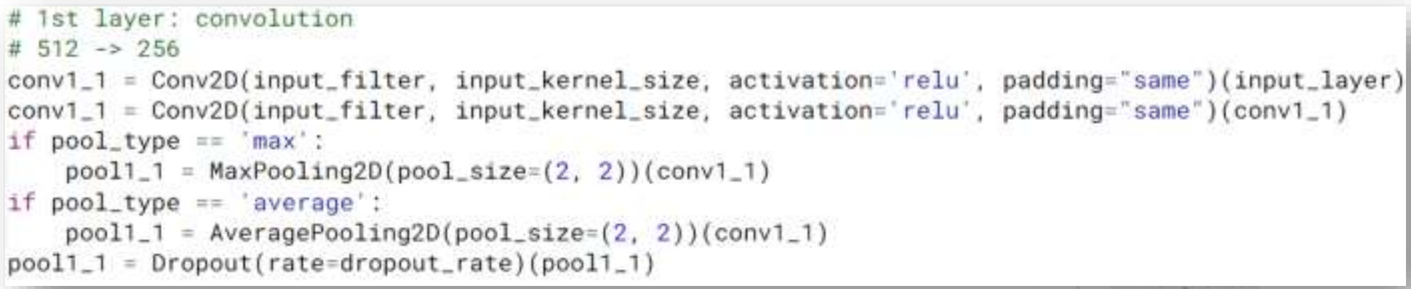


```
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 512, 512, 1  0           []
                                )]

conv2d (Conv2D)                 (None, 512, 512, 16  160         ['input_1[0][0]']
                                )

conv2d_1 (Conv2D)               (None, 512, 512, 16  2320        ['conv2d[0][0]']
                                )

max_pooling2d (MaxPooling2D)    (None, 256, 256, 16  0           ['conv2d_1[0][0]']
                                )

dropout (Dropout)               (None, 256, 256, 16  0           ['max_pooling2d[0][0]']
                                )

conv2d_2 (Conv2D)               (None, 256, 256, 32  4640        ['dropout[0][0]']
                                )

conv2d_3 (Conv2D)               (None, 256, 256, 32  9248        ['conv2d_2[0][0]']
                                )
```

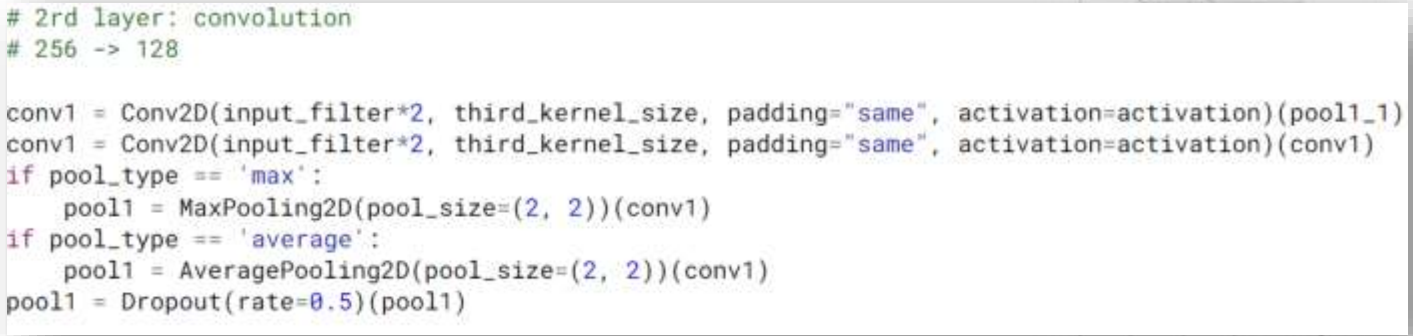
⋮ ⋮ ⋮



```
conv2d_transpose_5 (Conv2DTran  (None, 512, 512, 16  2320        ['conv2d_23[0][0]']
spose)                          )

concatenate_5 (Concatenate)     (None, 512, 512, 32  0           ['conv2d_transpose_5[0][0]',
                                )                                 'conv2d_1[0][0]']

dropout_11 (Dropout)            (None, 512, 512, 32  0           ['concatenate_5[0][0]']
                                )

conv2d_24 (Conv2D)              (None, 512, 512, 16  4624        ['dropout_11[0][0]']
                                )

conv2d_25 (Conv2D)              (None, 512, 512, 16  2320        ['conv2d_24[0][0]']
                                )

conv2d_26 (Conv2D)              (None, 512, 512, 1)  17          ['conv2d_25[0][0]']

==================================================================================================
Total params: 34,585,649
Trainable params: 34,585,649
Non-trainable params: 0
```

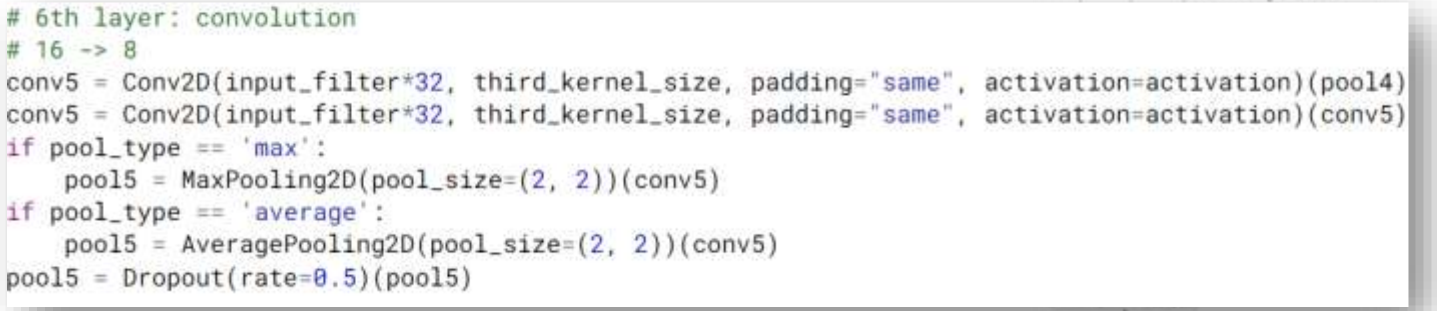


```
# 1st layer: convolution
# 512 -> 256
conv1_1 = Conv2D(input_filter, input_kernel_size, activation='relu', padding="same")(input_layer)
conv1_1 = Conv2D(input_filter, input_kernel_size, activation='relu', padding="same")(conv1_1)
if pool_type == 'max':
    pool1_1 = MaxPooling2D(pool_size=(2, 2))(conv1_1)
if pool_type == 'average':
    pool1_1 = AveragePooling2D(pool_size=(2, 2))(conv1_1)
pool1_1 = Dropout(rate=dropout_rate)(pool1_1)
```



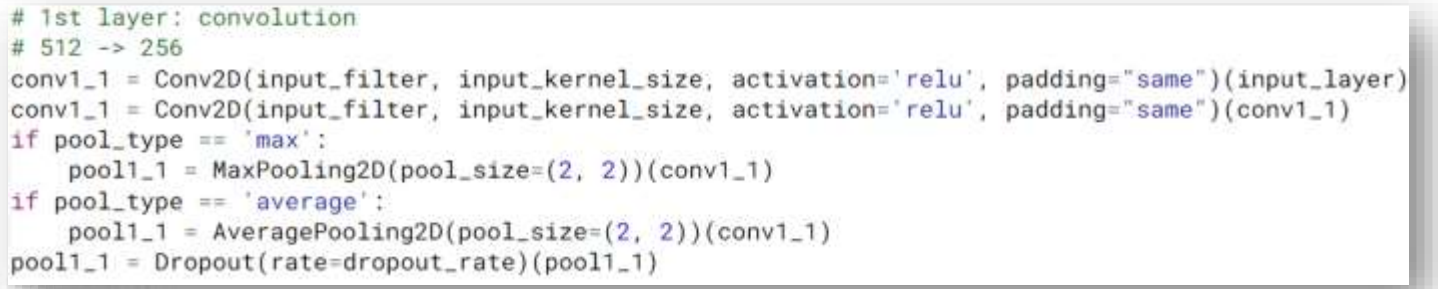
```
# 2rd layer: convolution
# 256 -> 128

conv1 = Conv2D(input_filter*2, third_kernel_size, padding="same", activation=activation)(pool1_1)
conv1 = Conv2D(input_filter*2, third_kernel_size, padding="same", activation=activation)(conv1)
if pool_type == 'max':
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
if pool_type == 'average':
    pool1 = AveragePooling2D(pool_size=(2, 2))(conv1)
pool1 = Dropout(rate=0.5)(pool1)
```

⋮ ⋮ ⋮

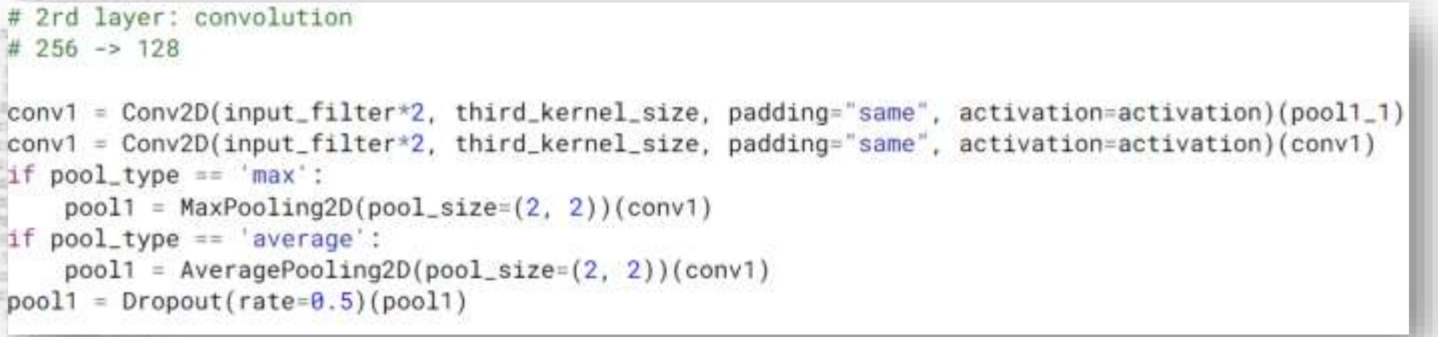


```
# 6th layer: convolution
# 16 -> 8
conv5 = Conv2D(input_filter*32, third_kernel_size, padding="same", activation=activation)(pool4)
conv5 = Conv2D(input_filter*32, third_kernel_size, padding="same", activation=activation)(conv5)
if pool_type == 'max':
    pool5 = MaxPooling2D(pool_size=(2, 2))(conv5)
if pool_type == 'average':
    pool5 = AveragePooling2D(pool_size=(2, 2))(conv5)
pool5 = Dropout(rate=0.5)(pool5)
```

Choosing Architecture

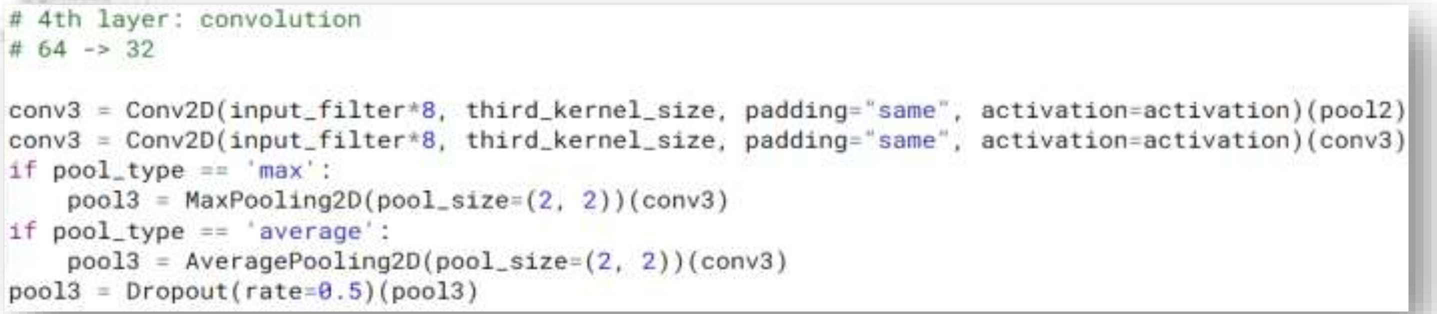


```
# 1st layer: convolution
# 512 -> 256
conv1_1 = Conv2D(input_filter, input_kernel_size, activation='relu', padding="same")(input_layer)
conv1_1 = Conv2D(input_filter, input_kernel_size, activation='relu', padding="same")(conv1_1)
if pool_type == 'max':
    pool1_1 = MaxPooling2D(pool_size=(2, 2))(conv1_1)
if pool_type == 'average':
    pool1_1 = AveragePooling2D(pool_size=(2, 2))(conv1_1)
pool1_1 = Dropout(rate=dropout_rate)(pool1_1)
```



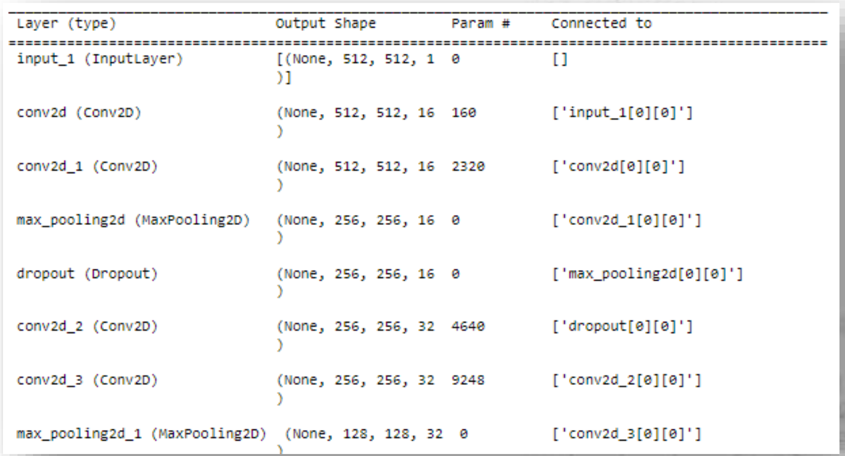
```
# 2rd layer: convolution
# 256 -> 128

conv1 = Conv2D(input_filter*2, third_kernel_size, padding="same", activation=activation)(pool1_1)
conv1 = Conv2D(input_filter*2, third_kernel_size, padding="same", activation=activation)(conv1)
if pool_type == 'max':
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
if pool_type == 'average':
    pool1 = AveragePooling2D(pool_size=(2, 2))(conv1)
pool1 = Dropout(rate=0.5)(pool1)
```

⋮ ⋮ ⋮



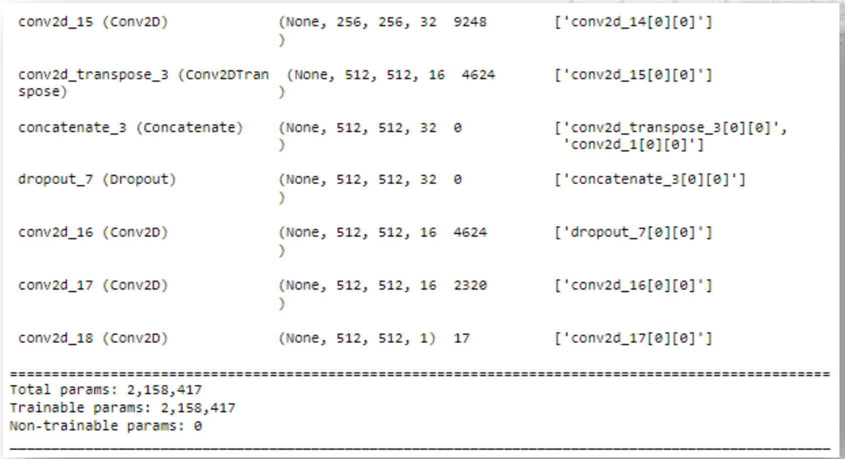
```
# 4th layer: convolution
# 64 -> 32

conv3 = Conv2D(input_filter*8, third_kernel_size, padding="same", activation=activation)(pool2)
conv3 = Conv2D(input_filter*8, third_kernel_size, padding="same", activation=activation)(conv3)
if pool_type == 'max':
    pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)
if pool_type == 'average':
    pool3 = AveragePooling2D(pool_size=(2, 2))(conv3)
pool3 = Dropout(rate=0.5)(pool3)
```

Unet_Model_V1 – 8 layers



```
____________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
============================================================================================
 input_1 (InputLayer)           [(None, 512, 512, 1  0           []
                                )]

 conv2d (Conv2D)                (None, 512, 512, 16  160         ['input_1[0][0]']
                                )

 conv2d_1 (Conv2D)              (None, 512, 512, 16  2320        ['conv2d[0][0]']
                                )

 max_pooling2d (MaxPooling2D)   (None, 256, 256, 16  0           ['conv2d_1[0][0]']
                                )

 dropout (Dropout)              (None, 256, 256, 16  0           ['max_pooling2d[0][0]']
                                )

 conv2d_2 (Conv2D)              (None, 256, 256, 32  4640        ['dropout[0][0]']
                                )

 conv2d_3 (Conv2D)              (None, 256, 256, 32  9248        ['conv2d_2[0][0]']
                                )

 max_pooling2d_1 (MaxPooling2D)  (None, 128, 128, 32  0          ['conv2d_3[0][0]']
                                )
```

⋮ ⋮ ⋮



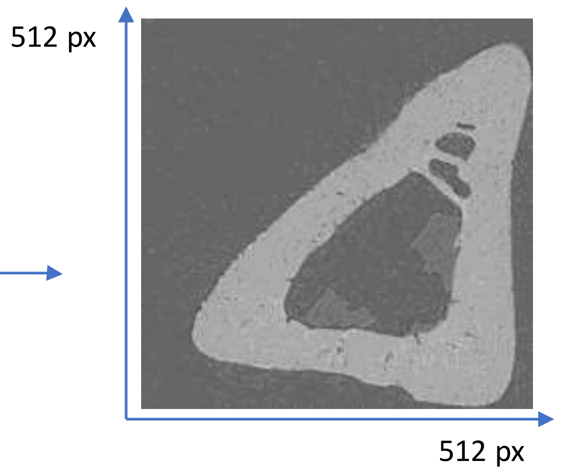
```
 conv2d_15 (Conv2D)             (None, 256, 256, 32  9248        ['conv2d_14[0][0]']
                                )

 conv2d_transpose_3 (Conv2DTran  (None, 512, 512, 16  4624       ['conv2d_15[0][0]']
 spose)                         )

 concatenate_3 (Concatenate)    (None, 512, 512, 32  0           ['conv2d_transpose_3[0][0]',
                                )                                 'conv2d_1[0][0]']

 dropout_7 (Dropout)            (None, 512, 512, 32  0           ['concatenate_3[0][0]']
                                )

 conv2d_16 (Conv2D)             (None, 512, 512, 16  4624        ['dropout_7[0][0]']
                                )

 conv2d_17 (Conv2D)             (None, 512, 512, 16  2320        ['conv2d_16[0][0]']
                                )

 conv2d_18 (Conv2D)             (None, 512, 512, 1)  17          ['conv2d_17[0][0]']

============================================================================================
Total params: 2,158,417
Trainable params: 2,158,417
Non-trainable params: 0
____________________________________________________________________________________________
```

Hyperparameters

```
def U_net(input_layer,
          input_filter = 16,
          input_kernel_size = (3,3),
          second_kernel_size = (3,3),
          third_kernel_size = (3,3),
          pool_type ='max',
          activation = 'tanh',
          optimizer = 'SGD',
          dropout_rate = 0.25):
```

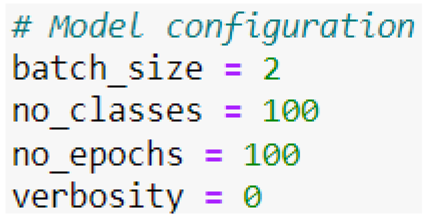
```
# 1st layer: convolution
# 512 -> 256
conv1_1 = Conv2D(input_filter, input_kernel_size, activation='relu', padding="same")(input_layer)
conv1_1 = Conv2D(input_filter, input_kernel_size, activation='relu', padding="same")(conv1_1)
if pool_type == 'max':
    pool1_1 = MaxPooling2D(pool_size=(2, 2))(conv1_1)
if pool_type == 'average':
    pool1_1 = AveragePooling2D(pool_size=(2, 2))(conv1_1)
pool1_1 = Dropout(rate=dropout_rate)(pool1_1)
```

```
#output
output_layer = Conv2D(1, (1,1), padding="same", activation="sigmoid")(uconv1_1)

return output_layer
```

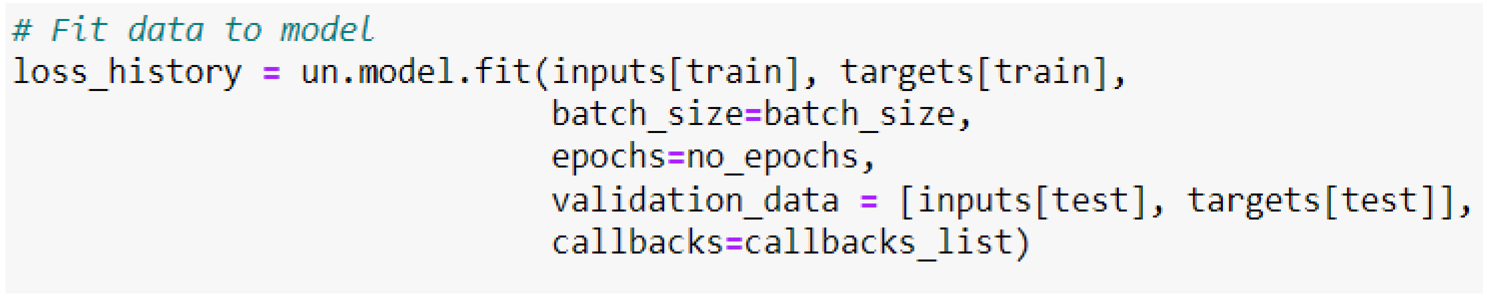


```
# Compile the model
un.model.compile(optimizer= 'Adam',
                 loss= un.dice_coef_loss,
                 metrics = [un.dice_coef])
```

Hyperparameters



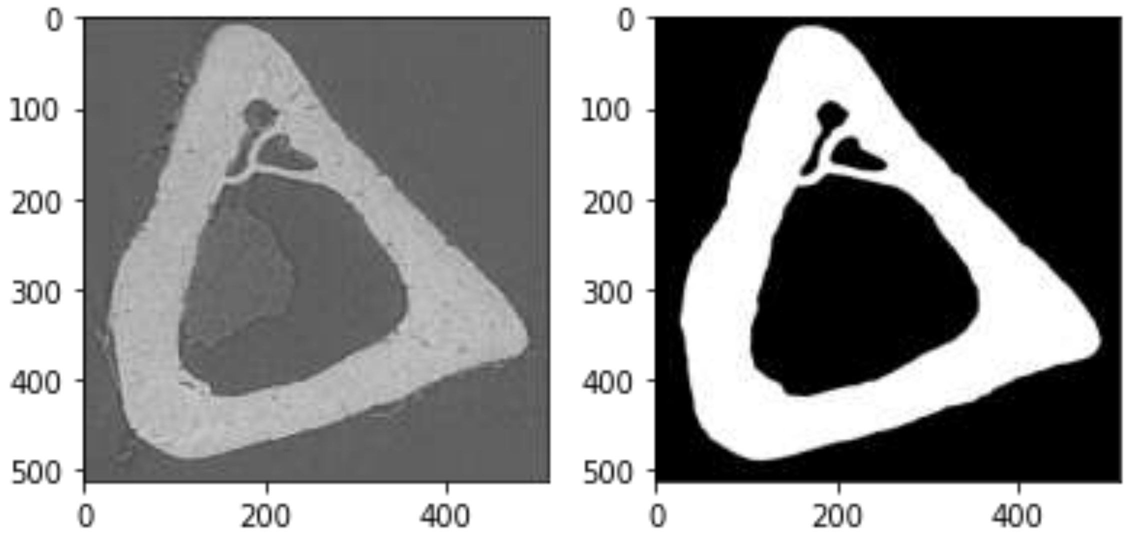
```
# Model configuration
batch_size = 2
no_classes = 100
no_epochs = 100
verbosity = 0
```



```
# Fit data to model
loss_history = un.model.fit(inputs[train], targets[train],
                            batch_size=batch_size,
                            epochs=no_epochs,
                            validation_data = [inputs[test], targets[test]],
                            callbacks=callbacks_list)
```

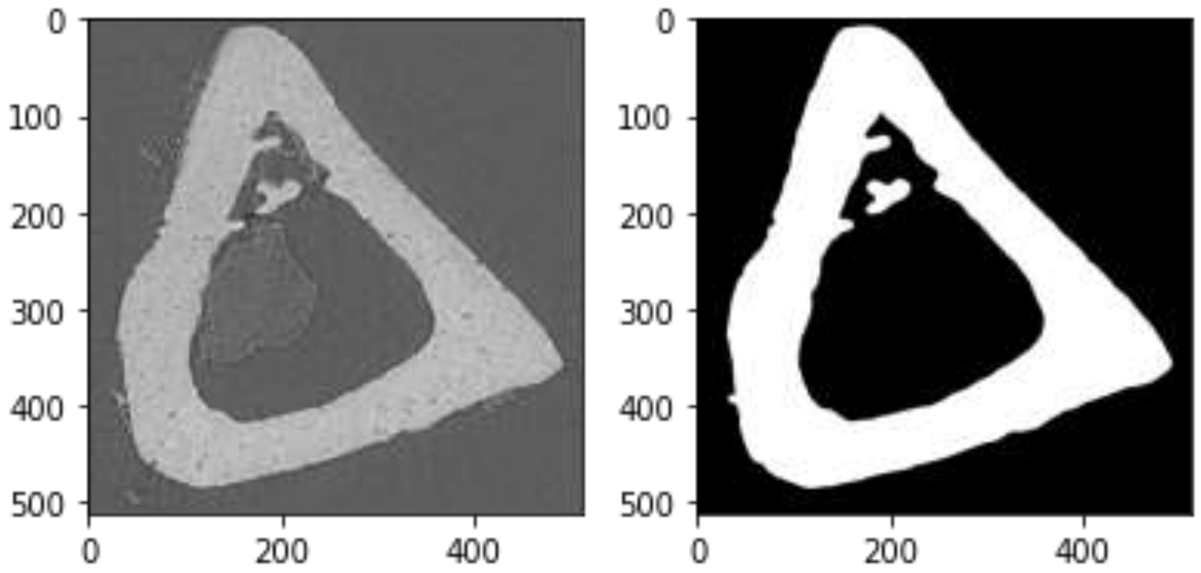
Hyperparameters

batch_size = 5, epochs = 100



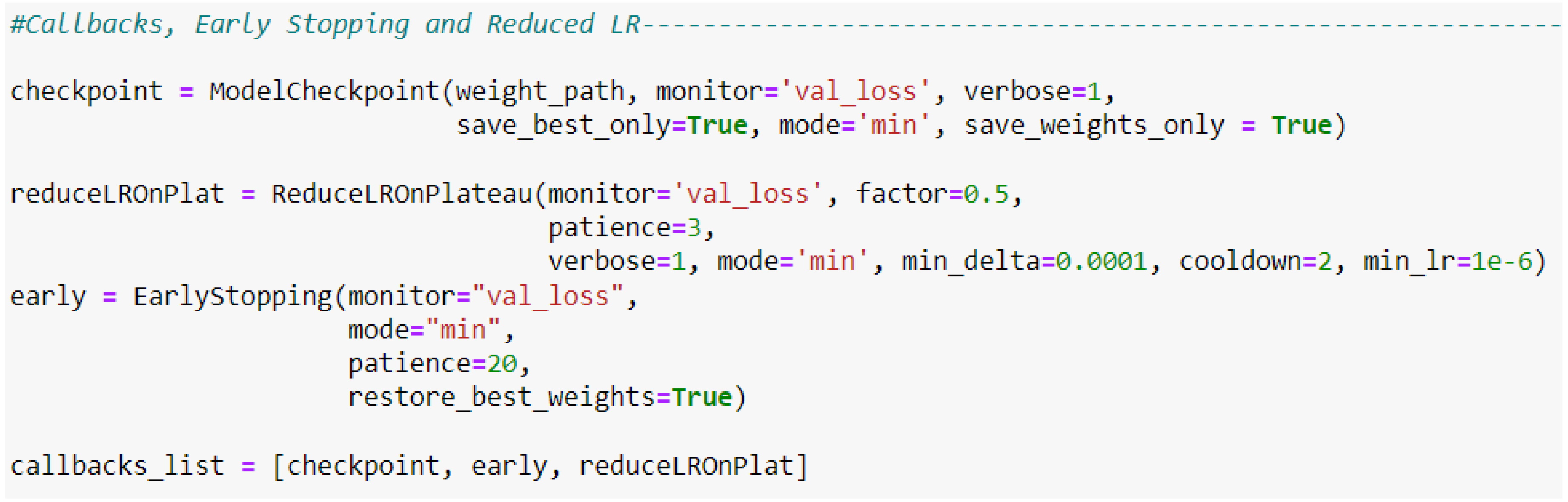
Dice coefficient : (57 +/-25)%

batch_size = 1, epochs = 100



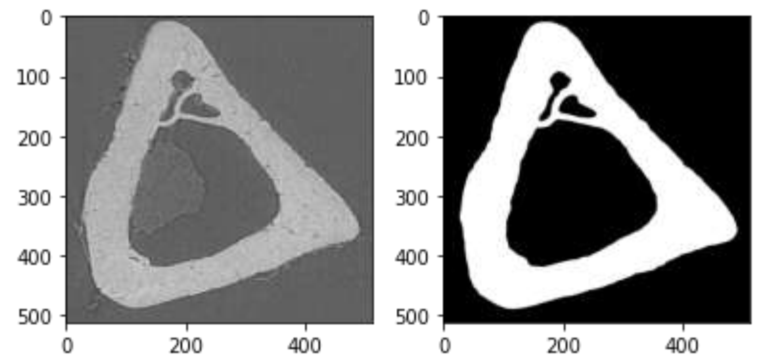
Dice coefficient : (76 +/- 7)%

Hyperparameters

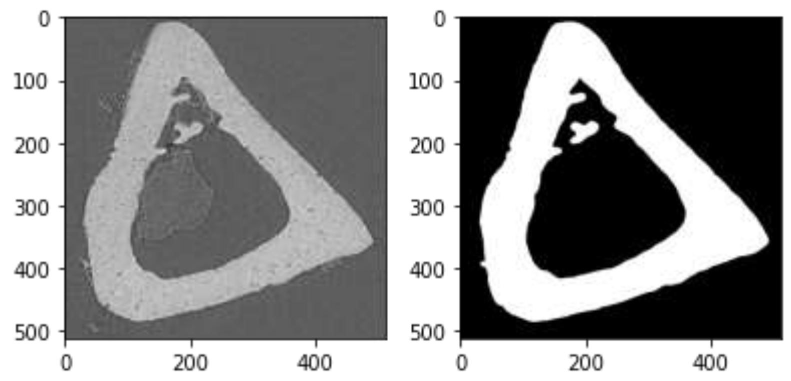


```
#Callbacks, Early Stopping and Reduced LR------------------------------------------------------------

checkpoint = ModelCheckpoint(weight_path, monitor='val_loss', verbose=1,
                             save_best_only=True, mode='min', save_weights_only = True)

reduceLROnPlat = ReduceLROnPlateau(monitor='val_loss', factor=0.5,
                                   patience=3,
                                   verbose=1, mode='min', min_delta=0.0001, cooldown=2, min_lr=1e-6)
early = EarlyStopping(monitor="val_loss",
                      mode="min",
                      patience=20,
                      restore_best_weights=True)

callbacks_list = [checkpoint, early, reduceLROnPlat]
```

batch_size = 5, epochs = 100



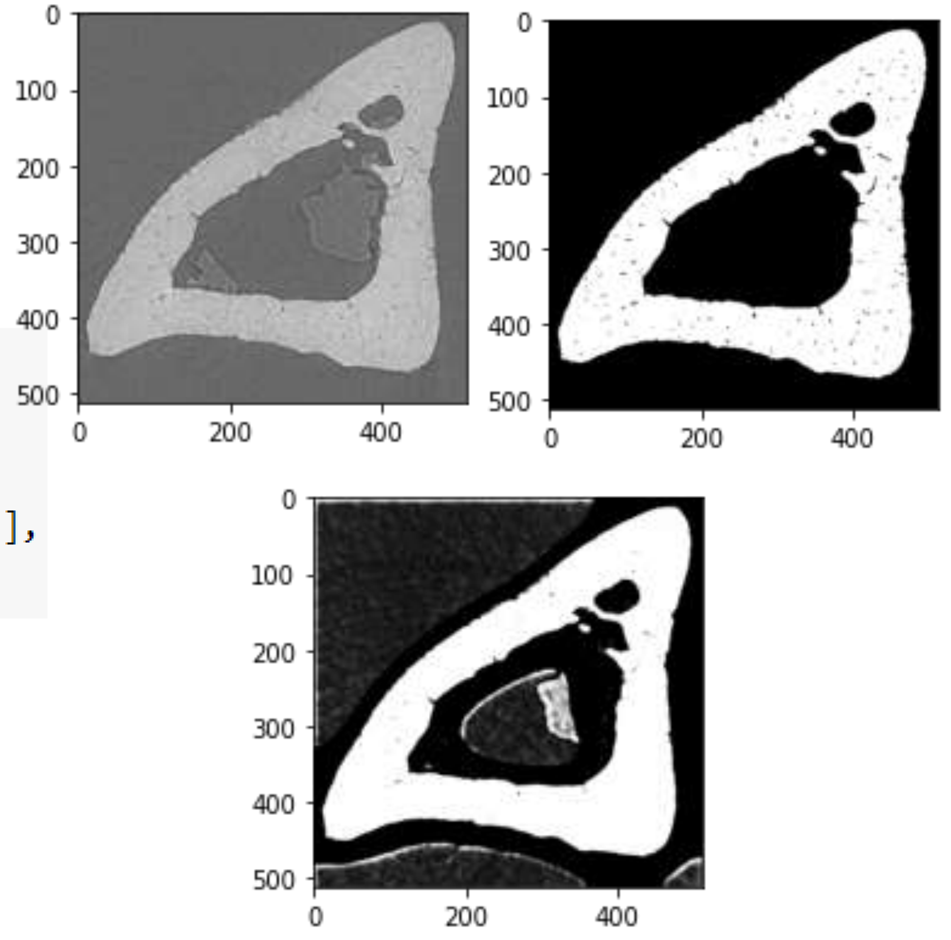
batch_size = 1, epochs = 100



Hyperparameters

```
# Model configuration
batch_size = 2
no_classes = 100
no_epochs = 100
verbosity = 0
```

```
# Fit data to model
loss_history = un.model.fit(inputs[train], targets[train],
                            batch_size=batch_size,
                            epochs=no_epochs,
                            validation_data = [inputs[test], targets[test]],
                            callbacks=callbacks_list)
```

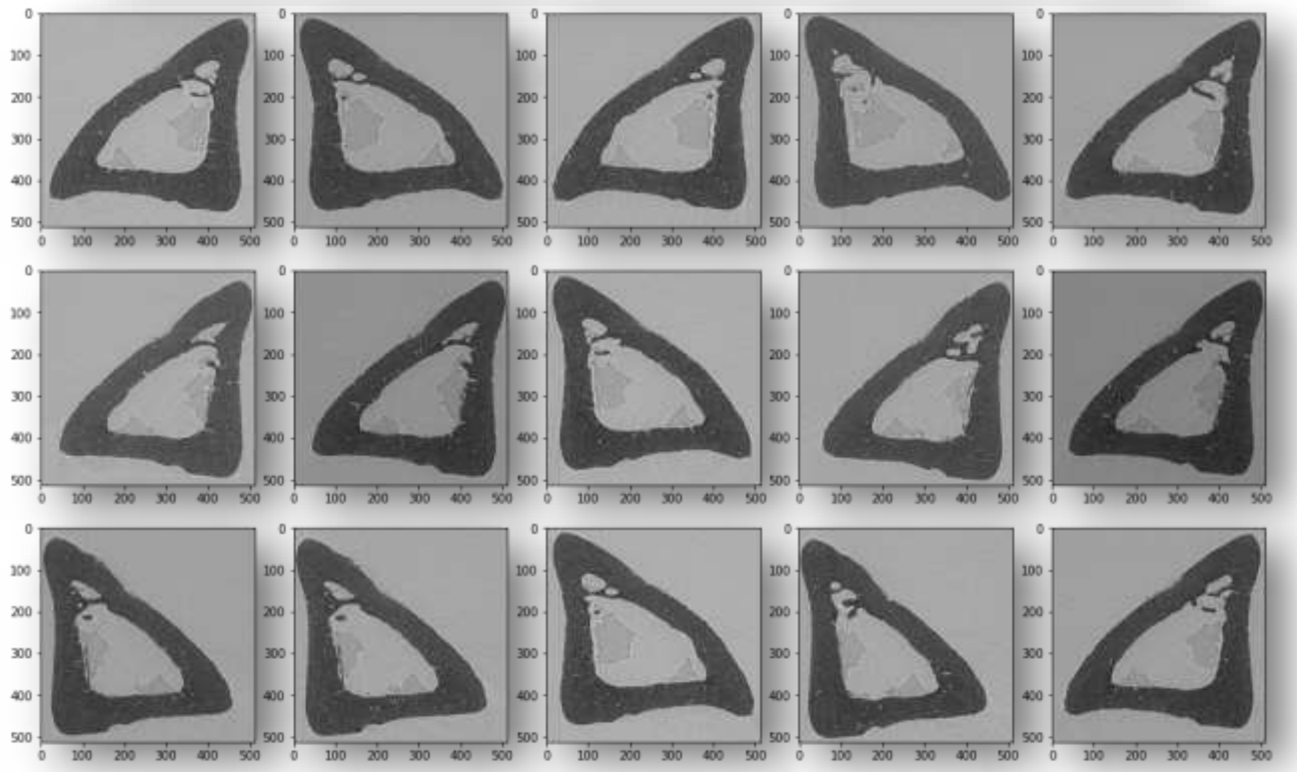


batch_size = 10, epochs = 100

- It has been empirically observed that smaller batch sizes not only have faster training dynamics, but also generalization to the test dataset versus larger batch sizes (deep learning book, 2022).

Data augmentation and Cross-validation

```python
import numpy as np
input = np.append(X, [np.fliplr(x) for x in X], axis=0)
output = np.append(y, [np.fliplr(x) for x in y], axis=0)
```



```python
# Define the K-fold Cross Validator
kfold = KFold(n_splits=num_folds,
              shuffle=True,
              random_state = 18)

# K-fold Cross Validation model evaluation
fold_no = 10
for train, test in kfold.split(inputs, targets):
```

⋮ ⋮ ⋮

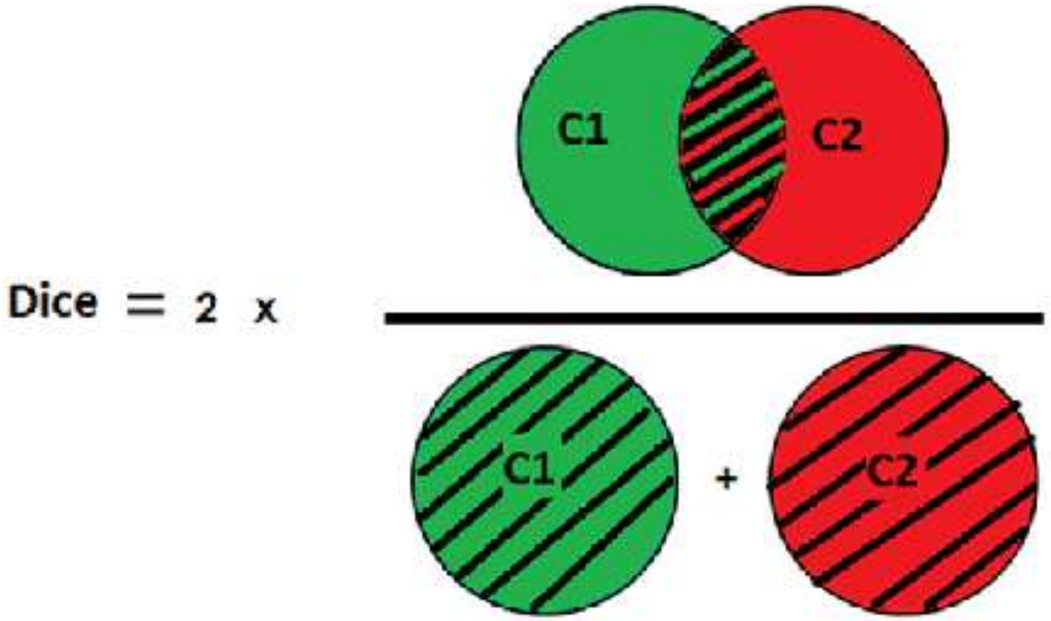
Dice coefficiente

$$D = \frac{2\sum_i^N p_i g_i}{\sum_i^N p_i^2 + \sum_i^N g_i^2}$$

Data augmentation and Cross-validation

Dice coefficiente

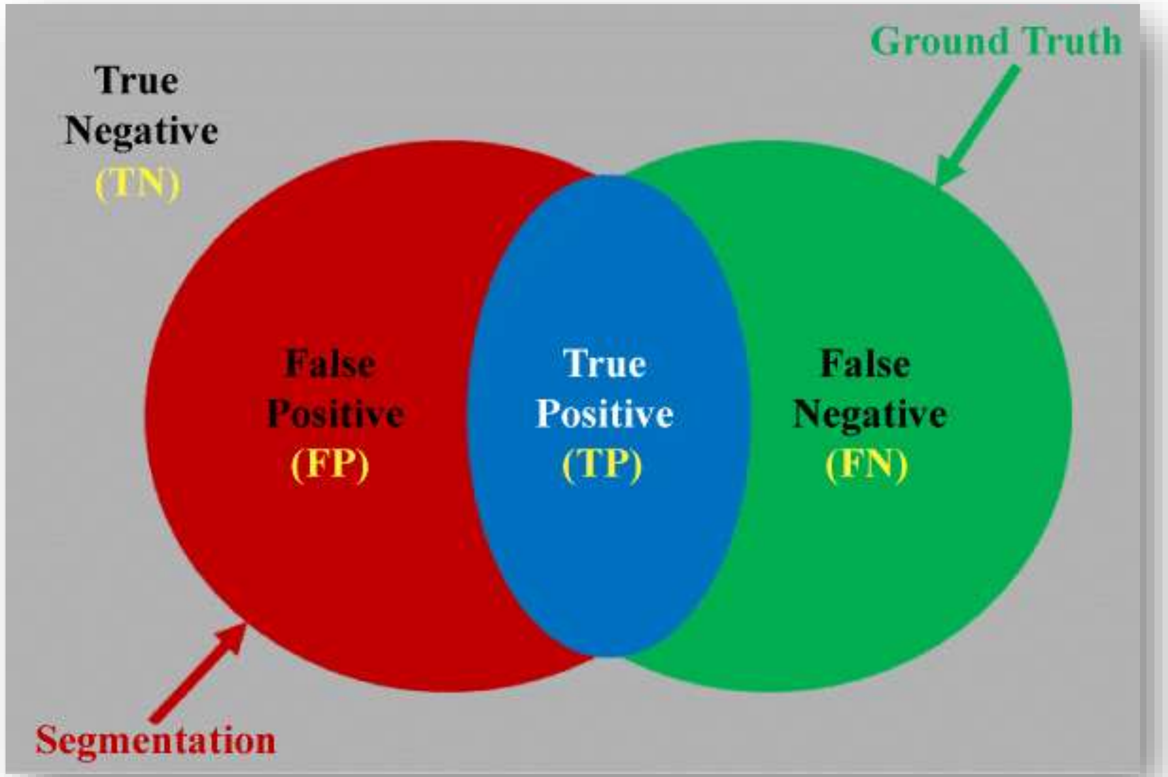
$$D = \frac{2\sum_i^N p_i g_i}{\sum_i^N p_i^2 + \sum_i^N g_i^2}$$



Data augmentation and Cross-validation

Dice coefficiente

$$D = \frac{2\sum_i^N p_i g_i}{\sum_i^N p_i^2 + \sum_i^N g_i^2}$$

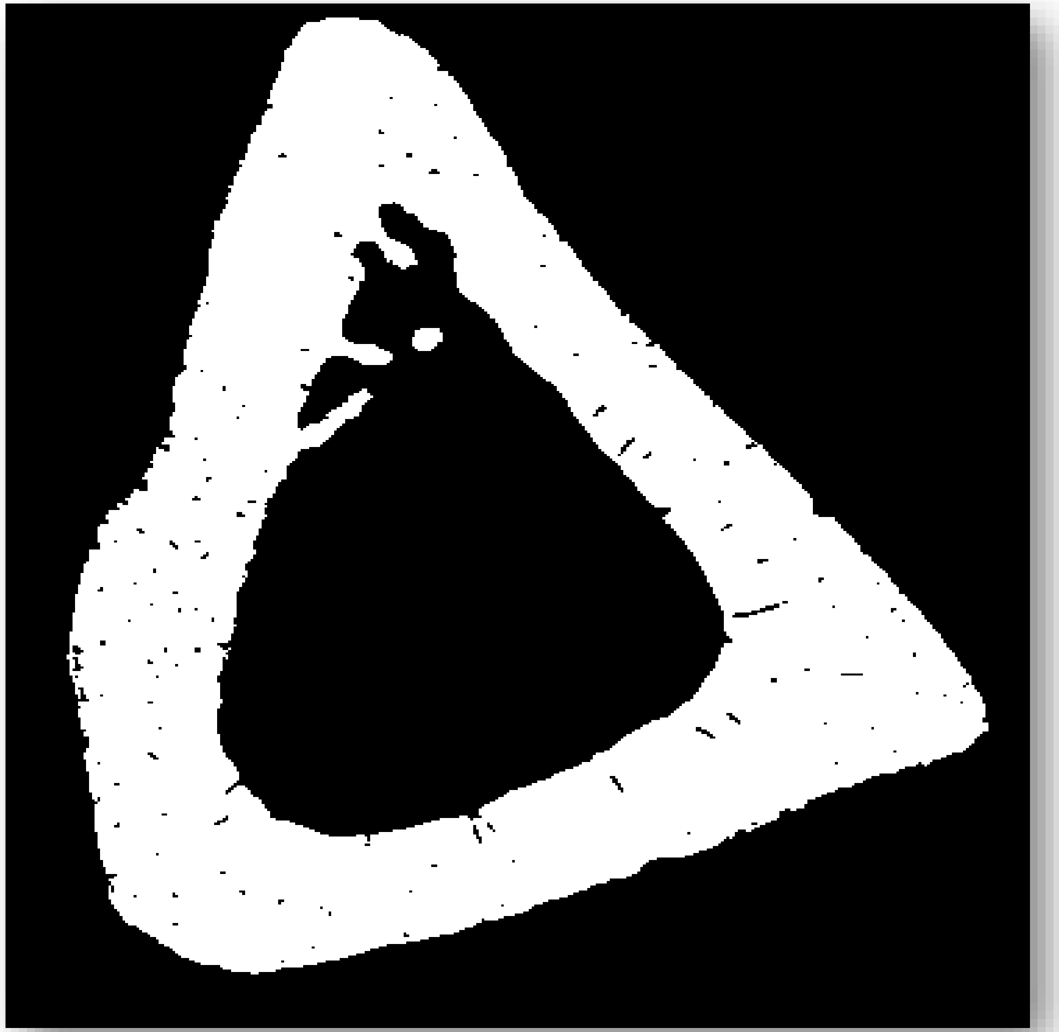


Conclusion

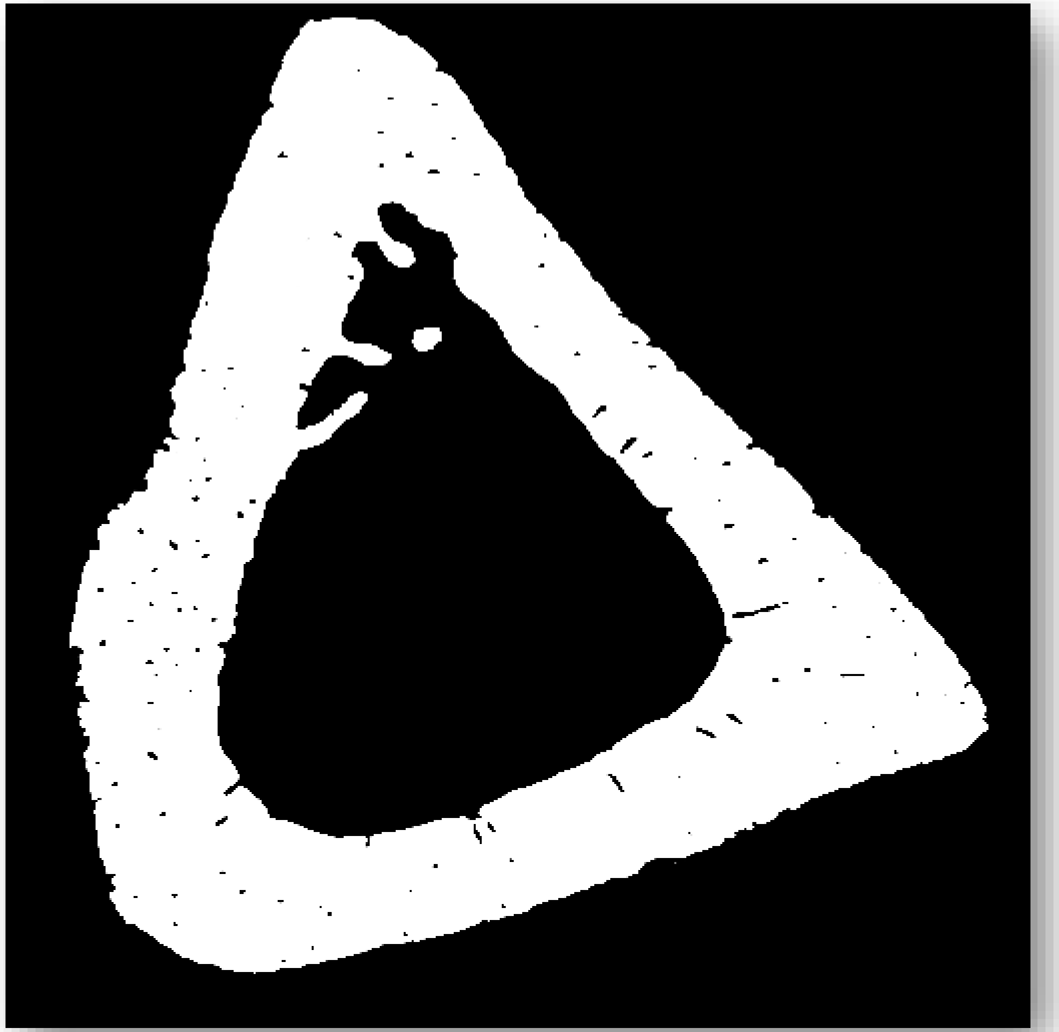
is it possible to segment the rat tibia image by neural networks? ✓

Results and discussion

Rat 25 - GroundTruth

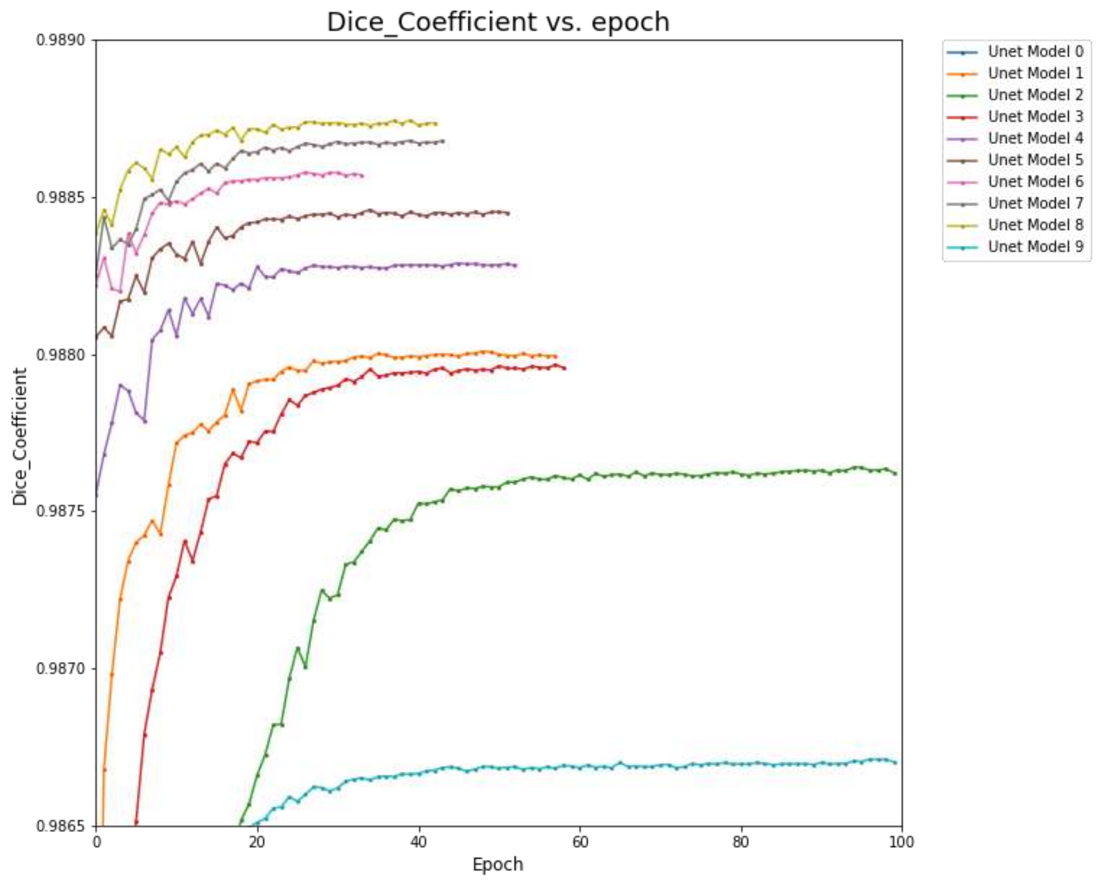


Rat 25 - Predicted

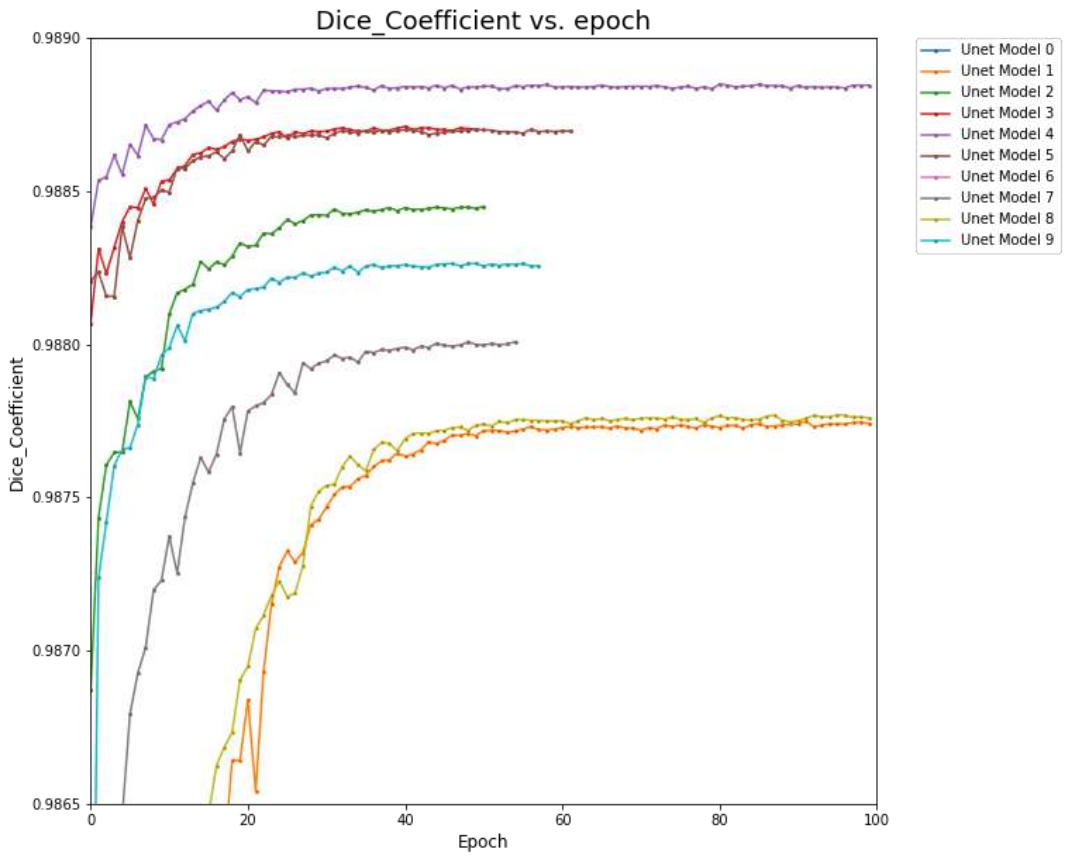


Results and discussion

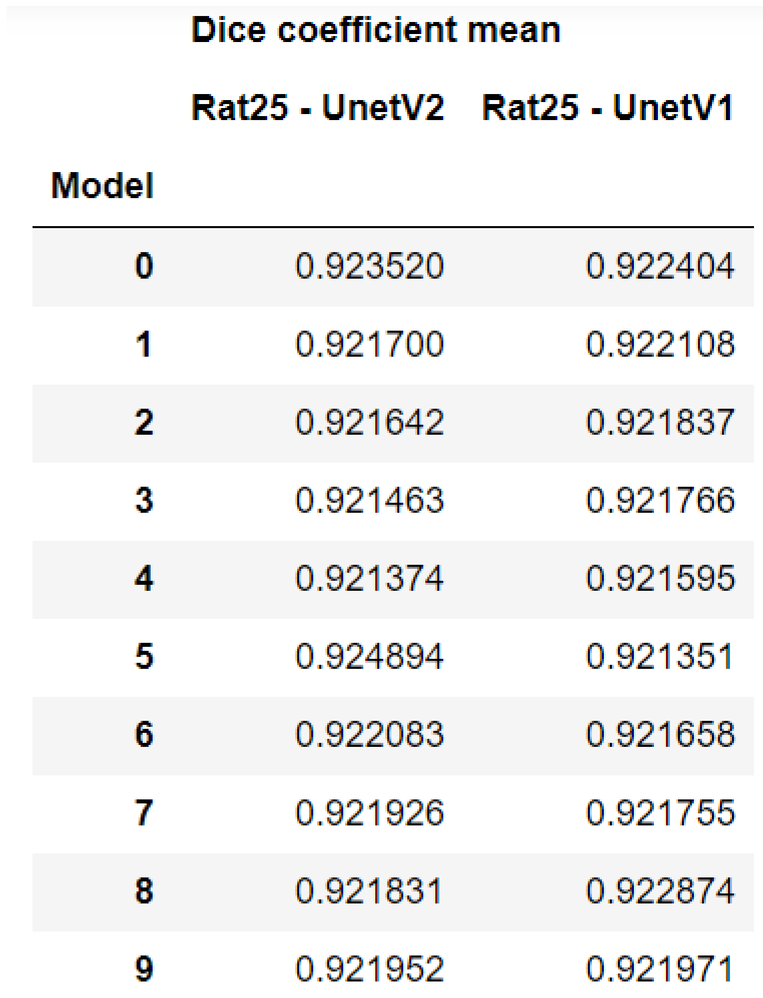
Unet – Model V1



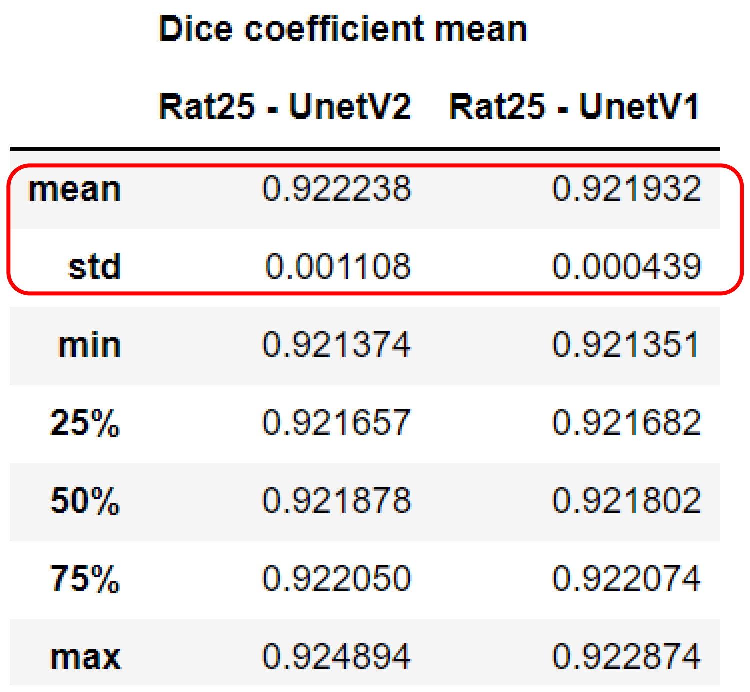
Unet – Model V2



Results and discussion

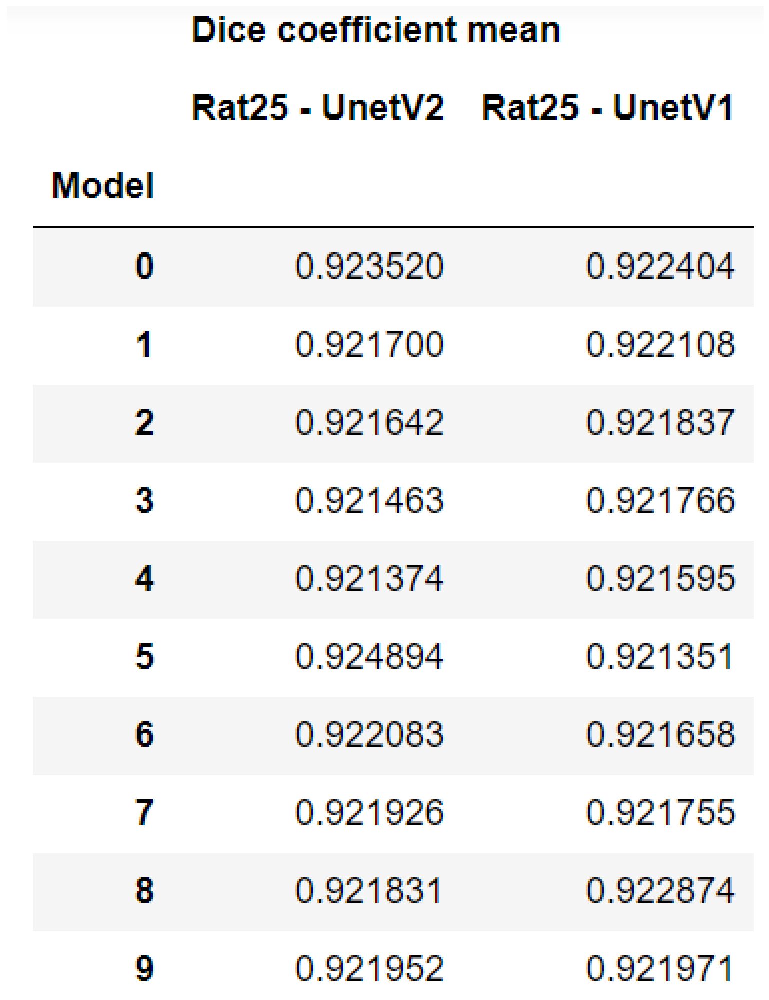


	Dice coefficient mean	
	Rat25 - UnetV2	Rat25 - UnetV1
Model		
0	0.923520	0.922404
1	0.921700	0.922108
2	0.921642	0.921837
3	0.921463	0.921766
4	0.921374	0.921595
5	0.924894	0.921351
6	0.922083	0.921658
7	0.921926	0.921755
8	0.921831	0.922874
9	0.921952	0.921971

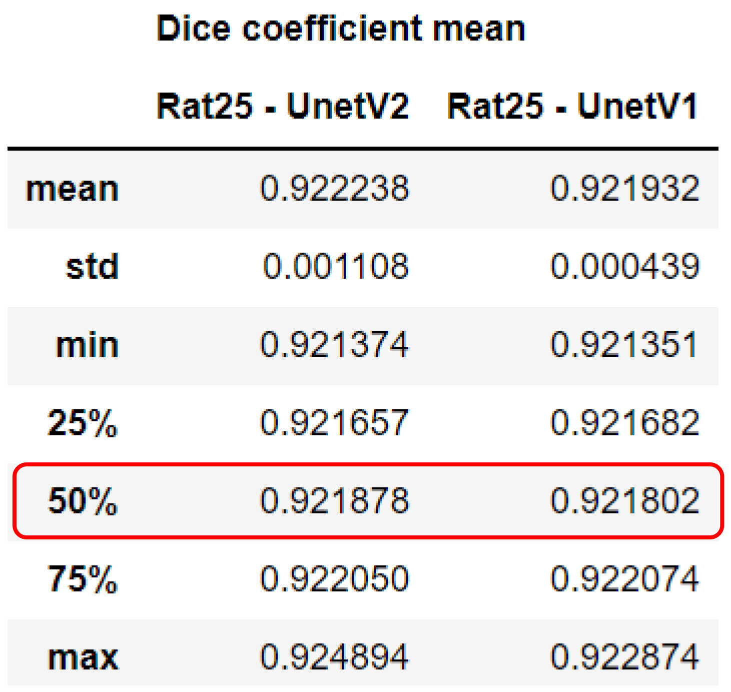


	Dice coefficient mean	
	Rat25 - UnetV2	Rat25 - UnetV1
mean	0.922238	0.921932
std	0.001108	0.000439
min	0.921374	0.921351
25%	0.921657	0.921682
50%	0.921878	0.921802
75%	0.922050	0.922074
max	0.924894	0.922874

Results and discussion



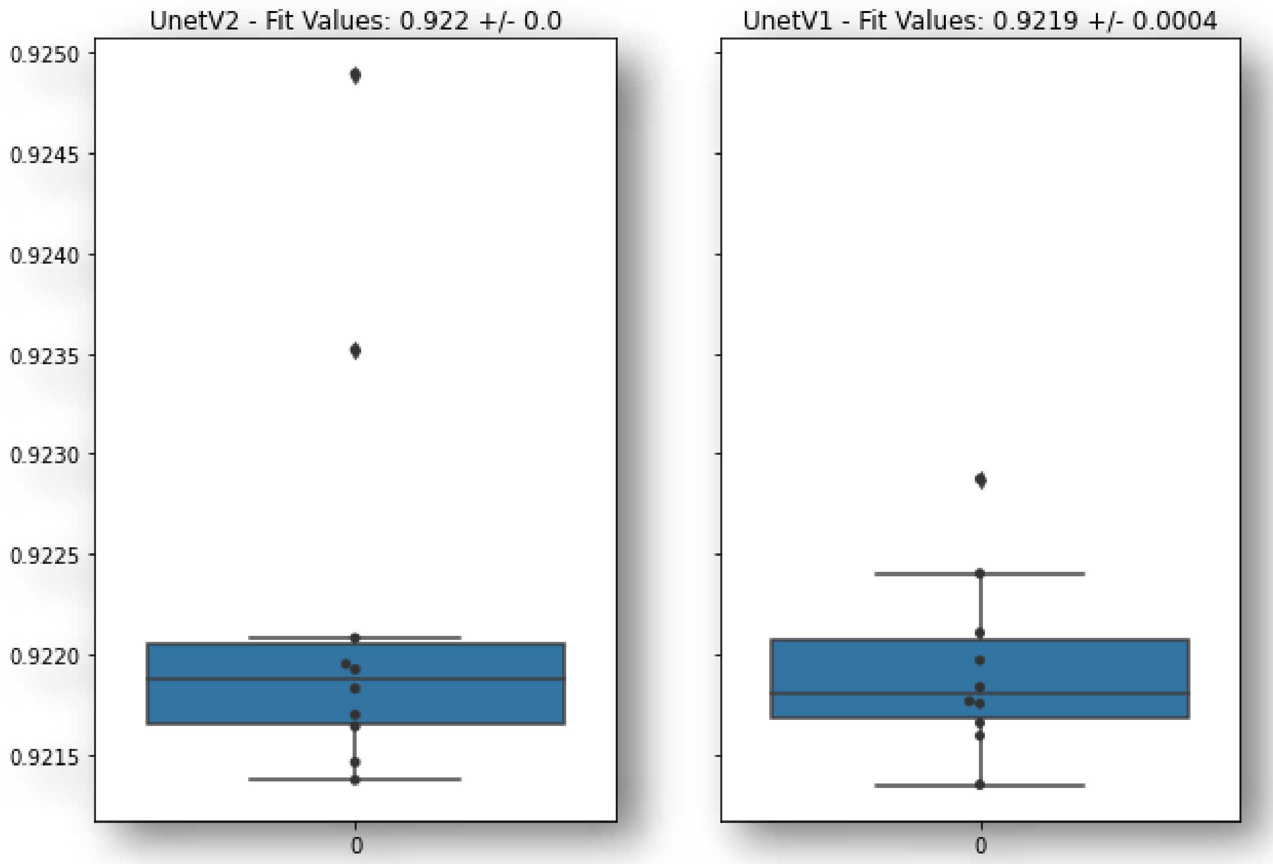
	Dice coefficient mean	
	Rat25 - UnetV2	**Rat25 - UnetV1**
Model		
0	0.923520	0.922404
1	0.921700	0.922108
2	0.921642	0.921837
3	0.921463	0.921766
4	0.921374	0.921595
5	0.924894	0.921351
6	0.922083	0.921658
7	0.921926	0.921755
8	0.921831	0.922874
9	0.921952	0.921971



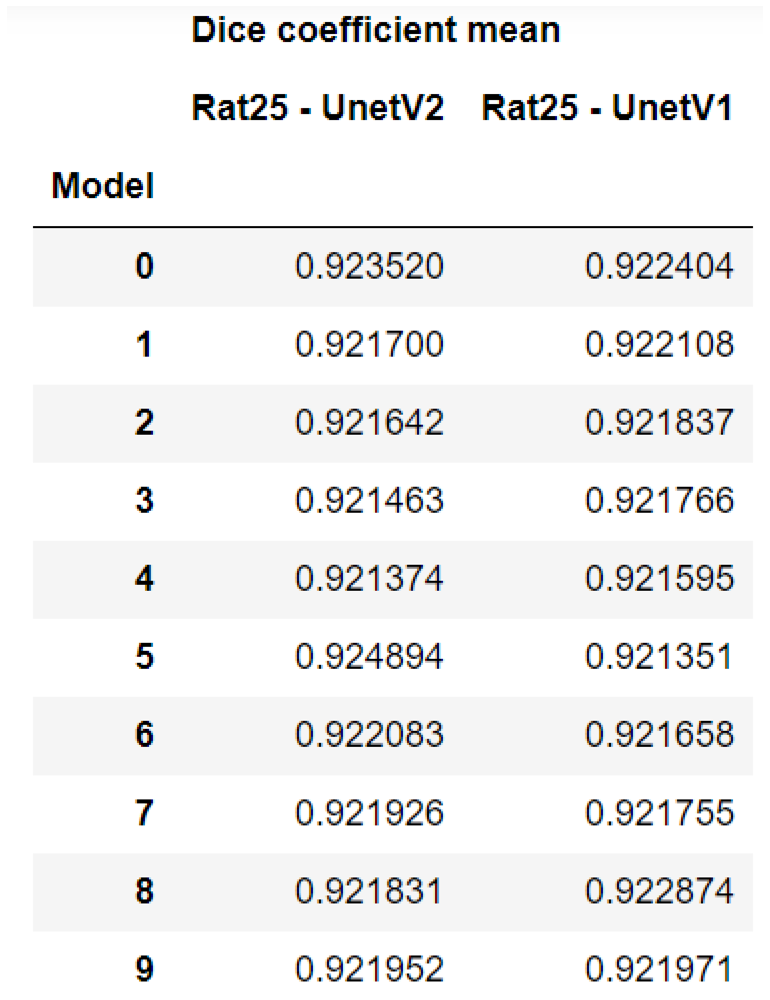
	Dice coefficient mean	
	Rat25 - UnetV2	**Rat25 - UnetV1**
mean	0.922238	0.921932
std	0.001108	0.000439
min	0.921374	0.921351
25%	0.921657	0.921682
50%	0.921878	0.921802
75%	0.922050	0.922074
max	0.924894	0.922874

Results and discussion

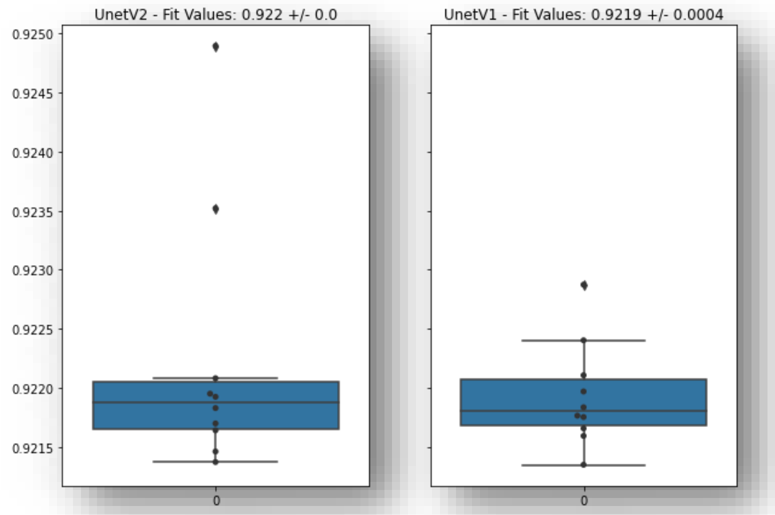
	Dice coefficient mean	
	Rat25 - UnetV2	Rat25 - UnetV1
Model		
0	0.923520	0.922404
1	0.921700	0.922108
2	0.921642	0.921837
3	0.921463	0.921766
4	0.921374	0.921595
5	0.924894	0.921351
6	0.922083	0.921658
7	0.921926	0.921755
8	0.921831	0.922874
9	0.921952	0.921971

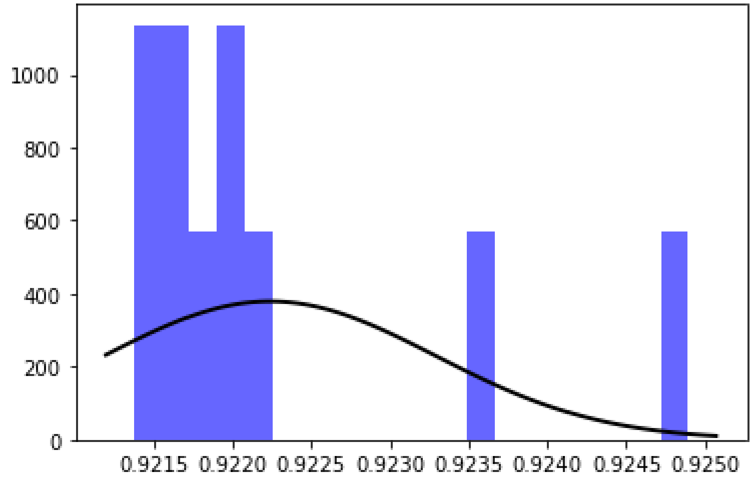


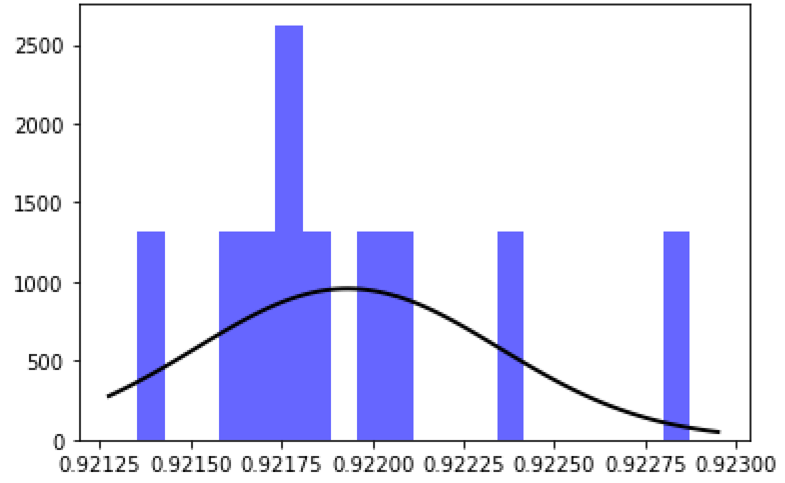
Results and discussion



	Dice coefficient mean	
	Rat25 - UnetV2	Rat25 - UnetV1
Model		
0	0.923520	0.922404
1	0.921700	0.922108
2	0.921642	0.921837
3	0.921463	0.921766
4	0.921374	0.921595
5	0.924894	0.921351
6	0.922083	0.921658
7	0.921926	0.921755
8	0.921831	0.922874
9	0.921952	0.921971







Results and discussion

Wilcoxon Test

Perform the Mann-Whitney U rank test on two independent samples.

The Mann-Whitney U test is a nonparametric test of the null hypothesis that the distribution underlying sample x is the same as the distribution underlying sample y. It is often used as a test of of difference in location between distributions.

Conditions:

- If P-value < 0.05: Reject NULL hypothesis - Significant differences exist between groups;
- If P-value > 0.05: Accept NULL hypothesis - No significant difference between groups.

Wilcoxon Test

Perform the Mann-Whitney U rank test on two independent samples.

The Mann-Whitney U test is a nonparametric test of the null hypothesis that the distribution underlying sample x is the same as the distribution underlying sample y. It is often used as a test of of difference in location between distributions.

Conditions:

- If P-value < 0.05: Reject NULL hypothesis - Significant differences exist between groups;
- If P-value > 0.05: Accept NULL hypothesis - No significant difference between groups.

Dice Coefficient:
H-statistic: 49.0
P-Value: 0.9698499769931556

Futures Objectives

Deep RNAs applied to Rats tibia

Vitor Souza Premoli Pinto de Oliveira*[1]
Eduardo Destefani Stefanato [1]
Prof. Dr. Christiano Jorge Gomes Pinheiro[1]
Prof. Dr. Anderson Alvarenga de Moura Meneses[2]

[1]Federal University of Espírito Santo
[2]Federal University of Oeste do Pará

REFERENCES

ABRAMI, A., et al, “Medical Applications of Synchrotron Radiation at the SYRMEP Beamline of ELETTRA”. **Nuc. inst. Met. A** 548, 221-227, 2005.

CHO, Yongil et al. Detection of the location of pneumothorax in chest X-rays using small artificial neural networks and a simple training process. **Scientific reports**, v. 11, n. 1, p. 1-8, 2021.

Tingelhoff K, Eichhorn KWG, Wagner I, Kunkel ME, Moral AI, Rilk ME, et al. Analysis of manual segmentation in paranasal CT images. Eur Arch Otorhinolaryngol 2008;265(9):1061–70. https://doi.org/10.1007/s00405-008-0594-z.

Gonzalez RC, Woods RE. Digital Image Processing. 4th ed. New York, NY: Pearson; 2018.

SÁNCHEZ, José Carlos González et al. Segmentation of bones in medical dual-energy computed tomography volumes using the 3D U-Net. Physica Medica, v. 69, p. 241-247, 2020.

ZHANG, Jeremy. **UNet — Line by Line Explanation:** Example UNet Implementation. Available in: https://towardsdatascience.com/unet-line-by-line-explanation-9b191c76baf5 . Access in: 08 FEB. 2022

DROZDZAL, Michal et al. The importance of skip connections in biomedical image segmentation. In: **Deep learning and data labeling for medical applications**. Springer, Cham, 2016. p. 179-187.

MENESES, Anderson Alvarenga de Moura; et al. **Graph cuts and neural networks for segmentation and porosity quantification in Synchrotron Radiation X-ray μCT of an igneous rock sample**. Applied Radiation and Isotopes, v. 133, p. 121-132, 2018.

ATTWOOD, D. **Soft X-Rays and Extreme Ultraviolet Radiation:** Principles an Applications. Cambridge University Press, 2007.

LEWIS, R. A. **Medical Phase Contrast X-Ray Imaging:** Current Status and Future Prospects. Physics Medicine and Biology 49:3573-3583, 2014.

MONTE, Leonardo de A. et al. Semantic Segmentation for People Detection on Beach Images. In: **Anais do XVIII Encontro Nacional de Inteligência Artificial e Computacional**. SBC, 2021. p. 691-702.

BREININGER, Katharina et al. Intraoperative stent segmentation in X-ray fluoroscopy for endovascular aortic repair. **International journal of computer assisted radiology and surgery**, v. 13, n. 8, p. 1221-1231, 2018.

CHEN, Shuqing et al. Automatic multi-organ segmentation in dual energy CT using 3D fully convolutional network. 2018.

MAIER, Andreas et al. A gentle introduction to deep learning in medical image processing. **Zeitschrift für Medizinische Physik**, v. 29, n. 2, p. 86-101, 2019.

Data Science Academy. **Deep Learning Book**, 2022. Disponível em: <https://www.deeplearningbook.com.br/>. Acesso em: 22 May. 2022.