2.4. Dataset

As a dataset for image segmentation, i used Knee MRI Quantitative Meniscal, Cartilage, and Bone Measurements titled in OAI dataset from NIHM DATA Archive [D1] which is features 3D MRI imaging of sagittal view of knee from different patients . Dataset was obtained from many hospital. In the training ,I used 3D MR imaging of sagittal view of 100 patient. One image has 160 slices.After converting 3D to 2D, I have 16.000 samples.

I chose MR images which have Flair sequence. The main reason is that the femoral cartilage looks more prominent than the other sequences, so i can train our model more easily and successfully.

To obtain masks of knee femoral cartilage which are segmented from knee bone, I used pyKNEEr library in Python.

2.5.Programming Steps in Deep Learning

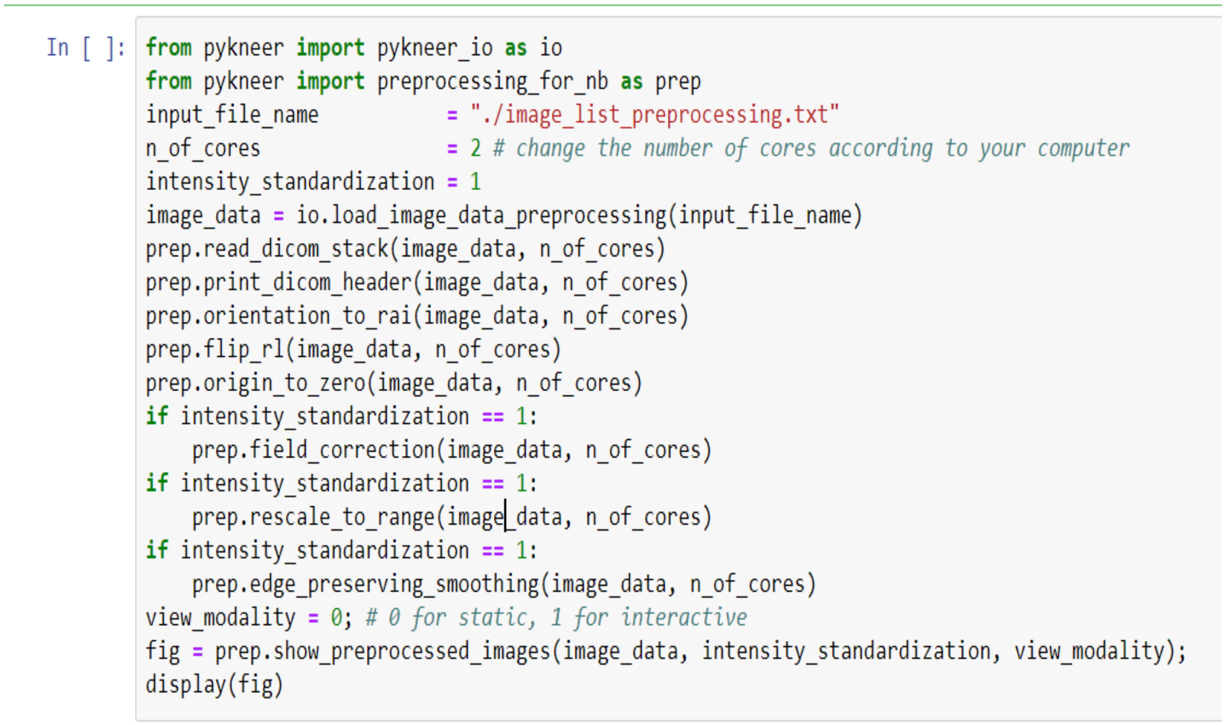
2.5.1.Creating Masks of Knee Femoral Cartilages on MR images

There are two parts like Preprocessing and segmentation MR images.

i. Preprocessing Images

Preprocessing of MR images is fundamental to standardize spatial information and intensity of images. And there is need to preprocessing process before starting segmentation process in pyKNEEr library.(I did not have to apply preprocessing on MR images again.)

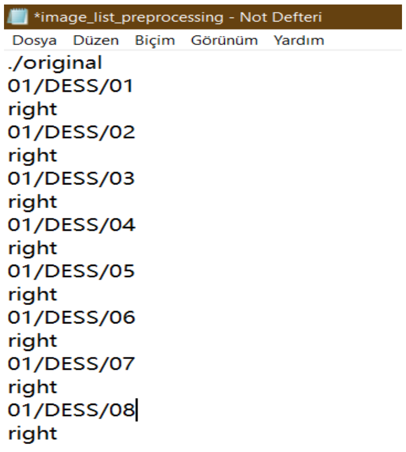
Below codes is for preprocessing images:



```
In [ ]: from pykneer import pykneer_io as io
        from pykneer import preprocessing_for_nb as prep
        input_file_name           = "./image_list_preprocessing.txt"
        n_of_cores                = 2 # change the number of cores according to your computer
        intensity_standardization = 1
        image_data = io.load_image_data_preprocessing(input_file_name)
        prep.read_dicom_stack(image_data, n_of_cores)
        prep.print_dicom_header(image_data, n_of_cores)
        prep.orientation_to_rai(image_data, n_of_cores)
        prep.flip_rl(image_data, n_of_cores)
        prep.origin_to_zero(image_data, n_of_cores)
        if intensity_standardization == 1:
            prep.field_correction(image_data, n_of_cores)
        if intensity_standardization == 1:
            prep.rescale_to_range(image_data, n_of_cores)
        if intensity_standardization == 1:
            prep.edge_preserving_smoothing(image_data, n_of_cores)
        view_modality = 0; # 0 for static, 1 for interactive
        fig = prep.show_preprocessed_images(image_data, intensity_standardization, view_modality);
        display(fig)
```

Figure 2.2. Codes of Preprocessing Images

Addresses and name of MR Images are written in txt file like below picture. In each preprocessing process, my txt file was including eight images name to apply process on eight MR images to be more quickly.


*image_list_preprocessing - Not Defteri
Dosya Düzen Biçim Görünüm Yardım

```
./original
01/DESS/01
right
01/DESS/02
right
01/DESS/03
right
01/DESS/04
right
01/DESS/05
right
01/DESS/06
right
01/DESS/07
right
01/DESS/08
right
```

Figure 2.3. Notepad file including names and address of images to preprocess

This program is used correct magnetic field inhomogeneities, rescale intensities, edge preserving smoothing which are preprocessing techniques to obtain better resolution of images.

After preprocessing process ,result was like:

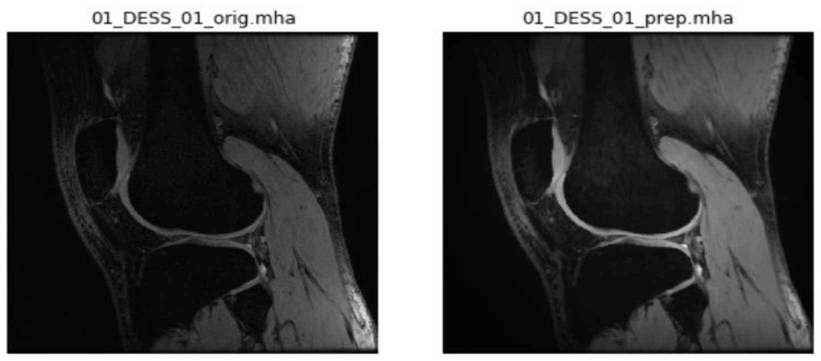


Figure 2.4. 01_DESS_01_orig.mha file is orginal MR image, 01_DESS_01_prep.mha is preprocessed MR image.

i. Segmentation of Knee Femoral Cartilages

Final part is femoral cartilage segmentation of preprocessed MR images.

Segmentation is performed using an atlas-based algorithm. Given a segmented image (reference image), registration to segment each image (moving image) of the dataset used.

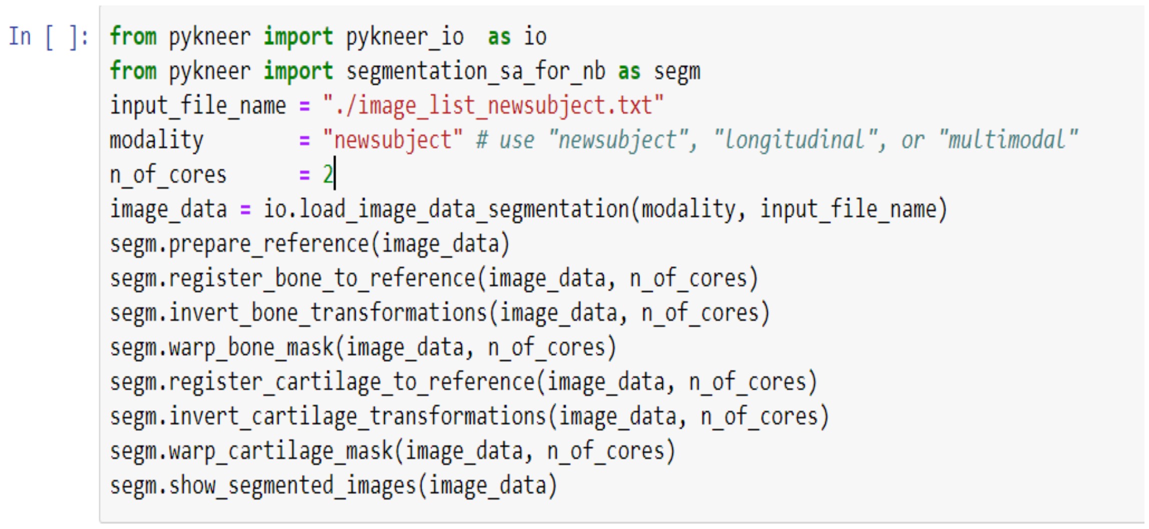
The segmentation consists of two parts:

- Segment bone (used to initialize femoral cartilage segmentation)
- Segment cartilage

Each part is composed by 3 steps:

I. Register image to reference. The moving image is registered to the reference image
II. Invert transformation. Transformations are inverted
III. Warp reference mask to moving image. Inverted transformations are applied to the mask of the reference image to obtain the mask of the moving image

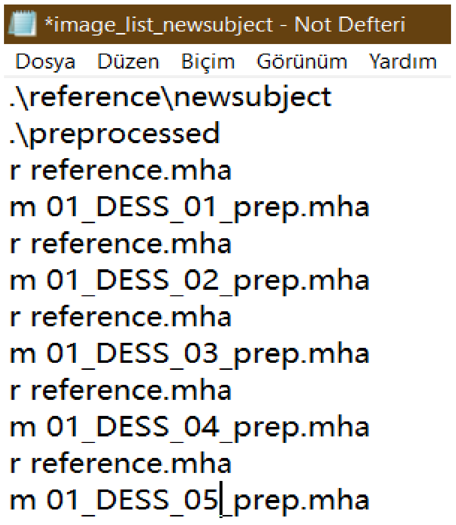
Segmentation codes are like below



```
In [ ]: from pykneer import pykneer_io  as io
        from pykneer import segmentation_sa_for_nb as segm
        input_file_name = "./image_list_newsubject.txt"
        modality        = "newsubject" # use "newsubject", "longitudinal", or "multimodal"
        n_of_cores      = 2
        image_data = io.load_image_data_segmentation(modality, input_file_name)
        segm.prepare_reference(image_data)
        segm.register_bone_to_reference(image_data, n_of_cores)
        segm.invert_bone_transformations(image_data, n_of_cores)
        segm.warp_bone_mask(image_data, n_of_cores)
        segm.register_cartilage_to_reference(image_data, n_of_cores)
        segm.invert_cartilage_transformations(image_data, n_of_cores)
        segm.warp_cartilage_mask(image_data, n_of_cores)
        segm.show_segmented_images(image_data)
```

Figure 2.5. Codes of Segmentation of Knee Femoral Cartilages

Addresses and name of preprocessed MR Images are written in notepad file like below picture. In each preprocessing process, my notepad file was including name of five images to apply process on five MR images to be more quickly.


*image_list_newsubject - Not Defteri
Dosya Düzen Biçim Görünüm Yardım

```
.\reference\newsubject
.\preprocessed
r reference.mha
m 01_DESS_01_prep.mha
r reference.mha
m 01_DESS_02_prep.mha
r reference.mha
m 01_DESS_03_prep.mha
r reference.mha
m 01_DESS_04_prep.mha
r reference.mha
m 01_DESS_05_prep.mha
```

Figure 2.6. Notepad file including names and address of images to segment

After segmentation part, Result was like:

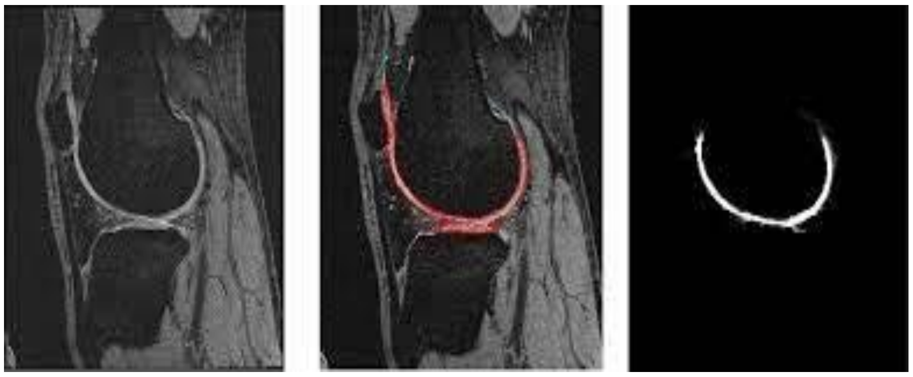


Figure 2.7. First images is preprocessed image. Second is image with segmented images. Third is just segmented images which is named to Mask.

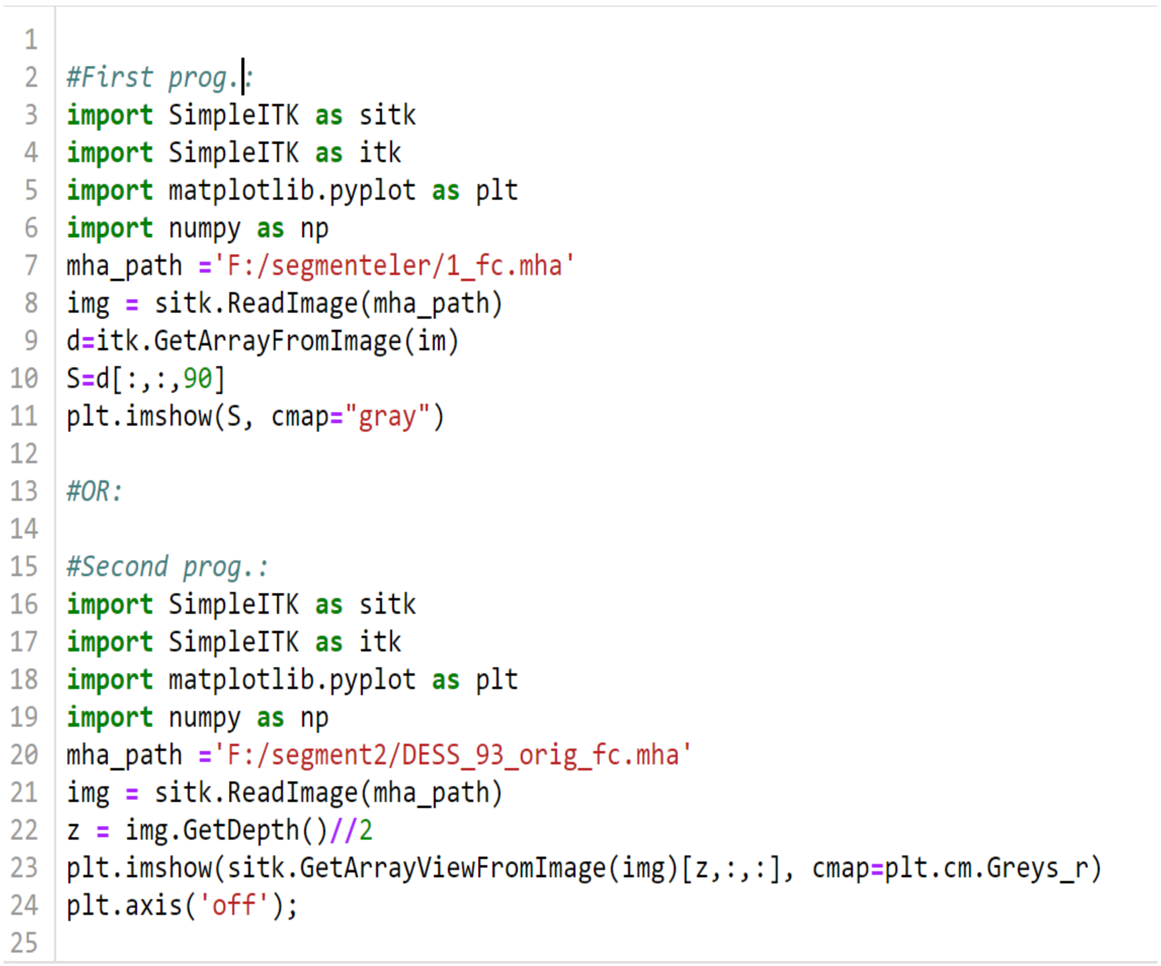
Finally , I had two hundered preprocessed and mask 3D images. One images have 160 channel size of slice. One slice has 384width size x 384 height size of pixels.

2.5.2.Checking That All Images are deserved or not

All images is 3D not 2D. So, I had to show images by slice to slice.

Steps of Codes

i. Read image
ii. Every pixel was connverted to numbers which creates array. Change to array from image format.(Because computer just recognizes numbers between 0 to 255.)
iii. Choose one slice from z-axis.
iv. Show them



```

#First prog.:
import SimpleITK as sitk
import SimpleITK as itk
import matplotlib.pyplot as plt
import numpy as np
mha_path ='F:/segmenteler/1_fc.mha'
img = sitk.ReadImage(mha_path)
d=itk.GetArrayFromImage(im)
S=d[:,:,90]
plt.imshow(S, cmap="gray")

#OR:

#Second prog.:
import SimpleITK as sitk
import SimpleITK as itk
import matplotlib.pyplot as plt
import numpy as np
mha_path ='F:/segment2/DESS_93_orig_fc.mha'
img = sitk.ReadImage(mha_path)
z = img.GetDepth()//2
plt.imshow(sitk.GetArrayViewFromImage(img)[z,:,:], cmap=plt.cm.Greys_r)
plt.axis('off');

```

Figure 2.8 Codes of Checking That All Images

Generally, result was like:

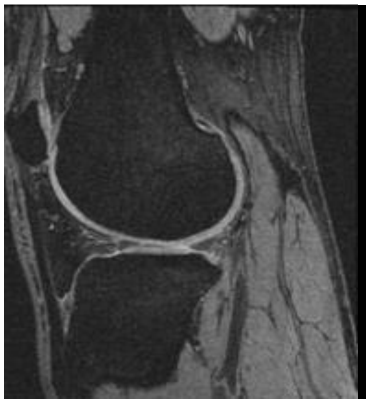


Figure 2.9. Preprocessed MR image

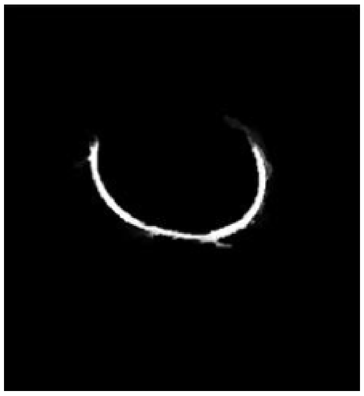


Figure 2.10. Segmented cartilage image(Mask)

But, some slice of preprocessed MR images have no slice of knee cartilage.This mean is that some of mask are all black images.

2.5.3.Preparing Images to Read

1- Converting 3D Images to 2D Images

In machine learning, Python uses image data in the form of a NumPy array, i.e., [Height, Width, Channel] format. To enhance the performance of the predictive model, we must know how to load and manipulate images. In Python, we can perform one task in different ways. Another way, Computer just recognizes numbers between 0 to 255. Every pixel should be connverted to numbers which creates array. Every number In array represent each pixel of images.

Steps of Codes

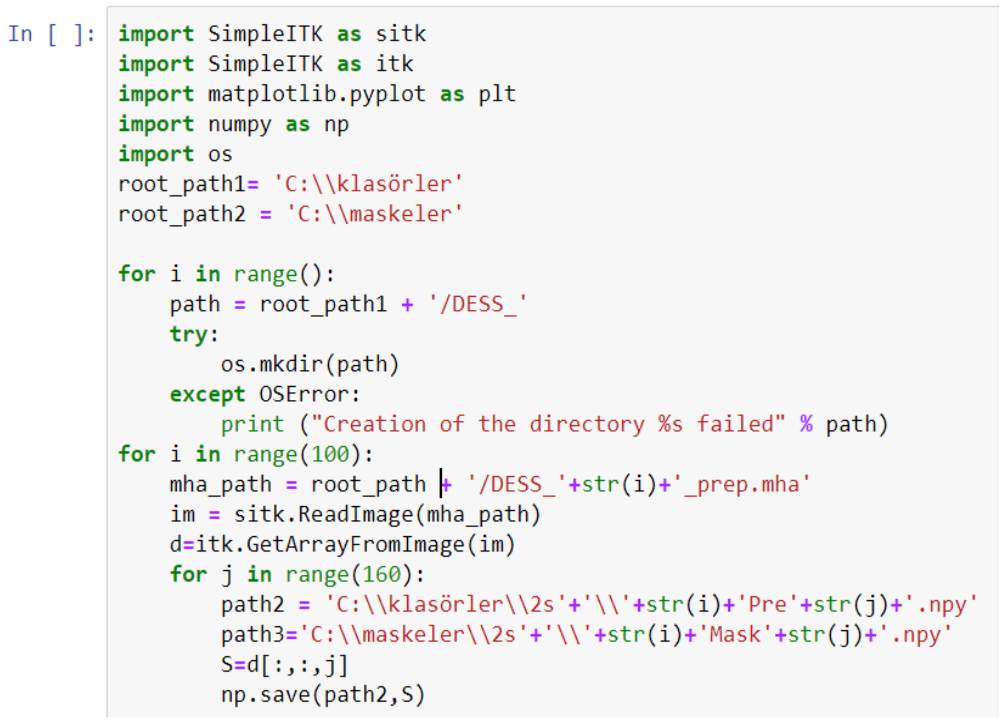
1-Read one 3D image in for loop.

2-Convert 3D images to 2D array.

3-Save 2D arrays to files.

Rooth_path1 is way of preprocessed images to be read, Rooth_path2 is way of mask to be read.

Path2 is way of preprocessed images to be saved like 2D array file.Path3 is way of mask to saved like 2D array file .



```
In [ ]: import SimpleITK as sitk
        import SimpleITK as itk
        import matplotlib.pyplot as plt
        import numpy as np
        import os
        root_path1= 'C:\\klasörler'
        root_path2 = 'C:\\maskeler'

        for i in range():
            path = root_path1 + '/DESS_'
            try:
                os.mkdir(path)
            except OSError:
                print ("Creation of the directory %s failed" % path)
        for i in range(100):
            mha_path = root_path + '/DESS_'+str(i)+'_prep.mha'
            im = sitk.ReadImage(mha_path)
            d=itk.GetArrayFromImage(im)
            for j in range(160):
                path2 = 'C:\\klasörler\\2s'+'\\'+str(i)+'Pre'+str(j)+'.npy'
                path3='C:\\maskeler\\2s'+'\\'+str(i)+'Mask'+str(j)+'.npy'
                S=d[:,:,j]
                np.save(path2,S)
```

Figure 2.11. Codes of Converting 3D Images to 2D Images

1- Normalization of Images

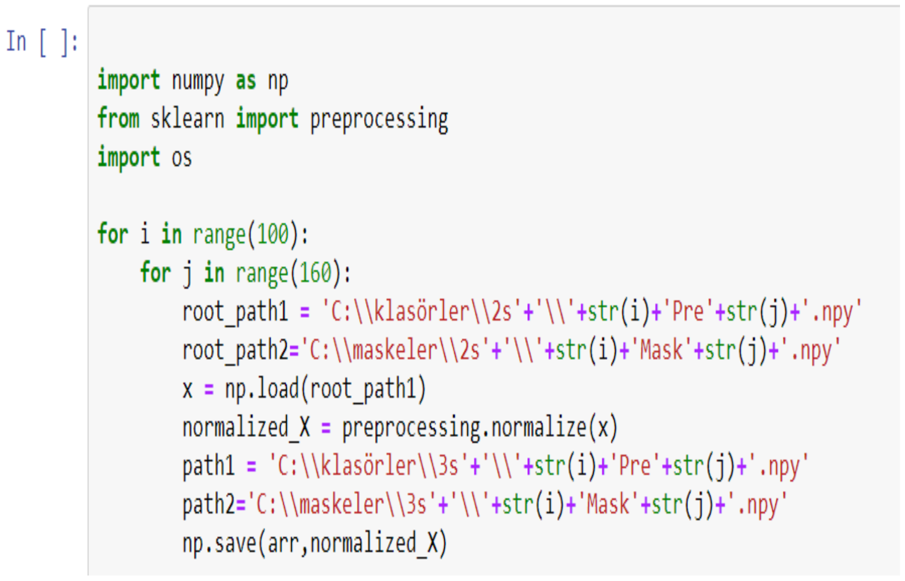
Normalization is a technique often applied as part of data preparation for machine learning. The goal of normalization is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values. For machine learning, every dataset does not require normalization. It is required only when features have different ranges.

Steps of Codes

1-Load array files in for loop.

2-Normalize arrays.

3-Save 2D normalized arrays to files.



```
In [ ]:
import numpy as np
from sklearn import preprocessing
import os

for i in range(100):
    for j in range(160):
        root_path1 = 'C:\\klasörler\\2s'+'\\'+str(i)+'Pre'+str(j)+'.npy'
        root_path2='C:\\maskeler\\2s'+'\\'+str(i)+'Mask'+str(j)+'.npy'
        x = np.load(root_path1)
        normalized_X = preprocessing.normalize(x)
        path1 = 'C:\\klasörler\\3s'+'\\'+str(i)+'Pre'+str(j)+'.npy'
        path2='C:\\maskeler\\3s'+'\\'+str(i)+'Mask'+str(j)+'.npy'
        np.save(arr,normalized_X)
```

Figure 2.12 Codes of Normalization of Images

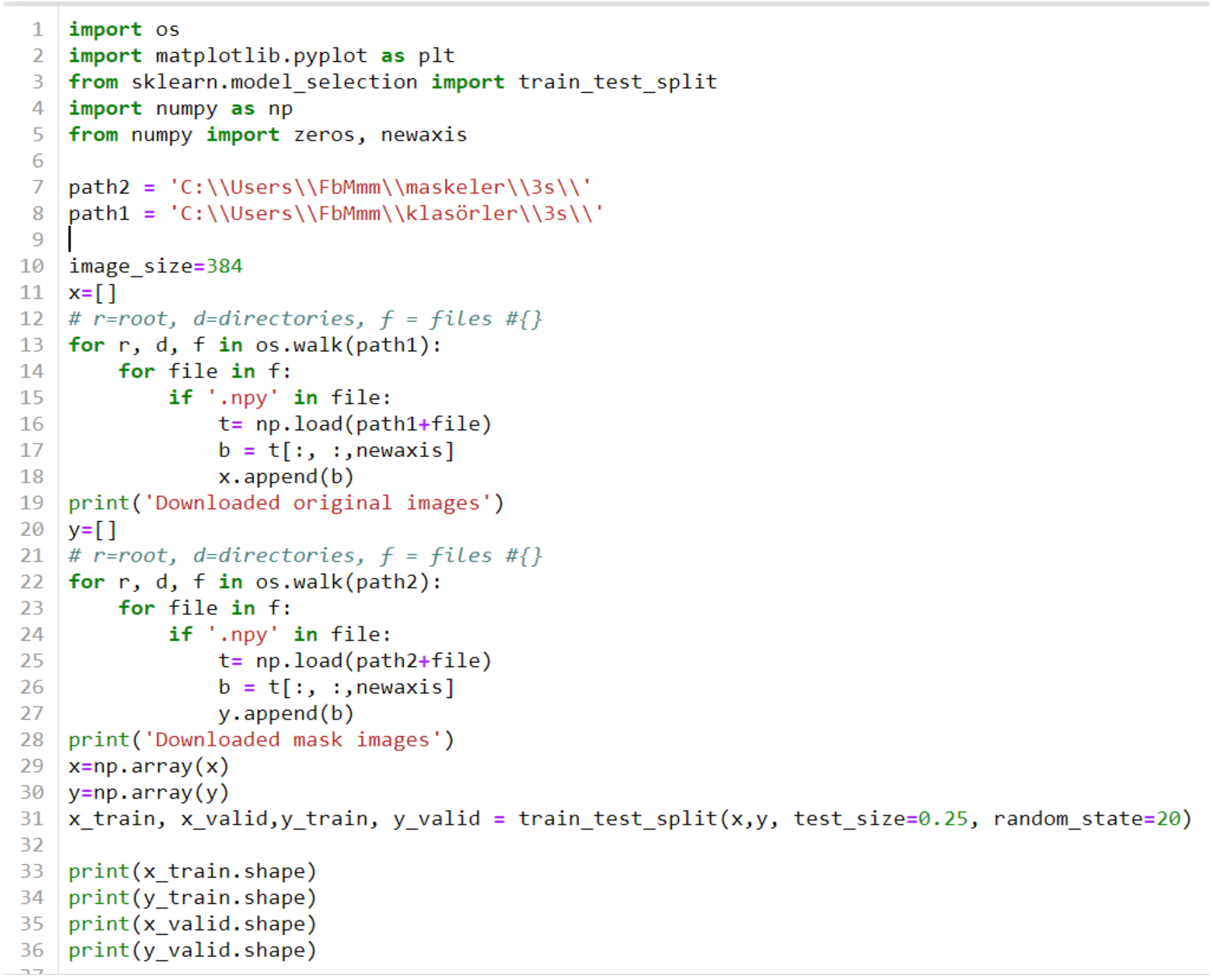
2.5.4.Loading Prepared All Datas And Split Them Into Train And Validation

Steps of Codes

1- Load mask and preprocessed data.
2- Reshape all data and add newaxis in z-axis for input of architecture.
3- Append array in a variable.

4- Split them into train and validation. Rate of train size is 75%, rate of validation size is 25%.

5- Check expected shape of all array.



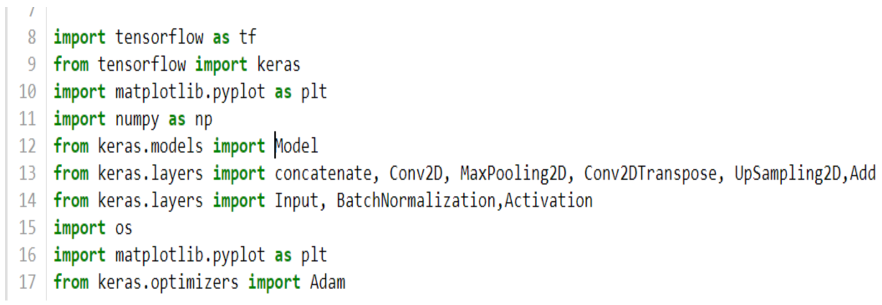
```python
import os
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import numpy as np
from numpy import zeros, newaxis

path2 = 'C:\\Users\\FbMmm\\maskeler\\3s\\'
path1 = 'C:\\Users\\FbMmm\\klasörler\\3s\\'

image_size=384
x=[]
# r=root, d=directories, f = files #{}
for r, d, f in os.walk(path1):
    for file in f:
        if '.npy' in file:
            t= np.load(path1+file)
            b = t[:, :,newaxis]
            x.append(b)
print('Downloaded original images')
y=[]
# r=root, d=directories, f = files #{}
for r, d, f in os.walk(path2):
    for file in f:
        if '.npy' in file:
            t= np.load(path2+file)
            b = t[:, :,newaxis]
            y.append(b)
print('Downloaded mask images')
x=np.array(x)
y=np.array(y)
x_train, x_valid,y_train, y_valid = train_test_split(x,y, test_size=0.25, random_state=20)

print(x_train.shape)
print(y_train.shape)
print(x_valid.shape)
print(y_valid.shape)
```

Figure 2.13 Codes of Loading Prepared All Datas And Split Them Into Train And Validation

2.5.5.Architectures

I used two similar architecture which are U-net and ResU-net.

1- Define common libraries

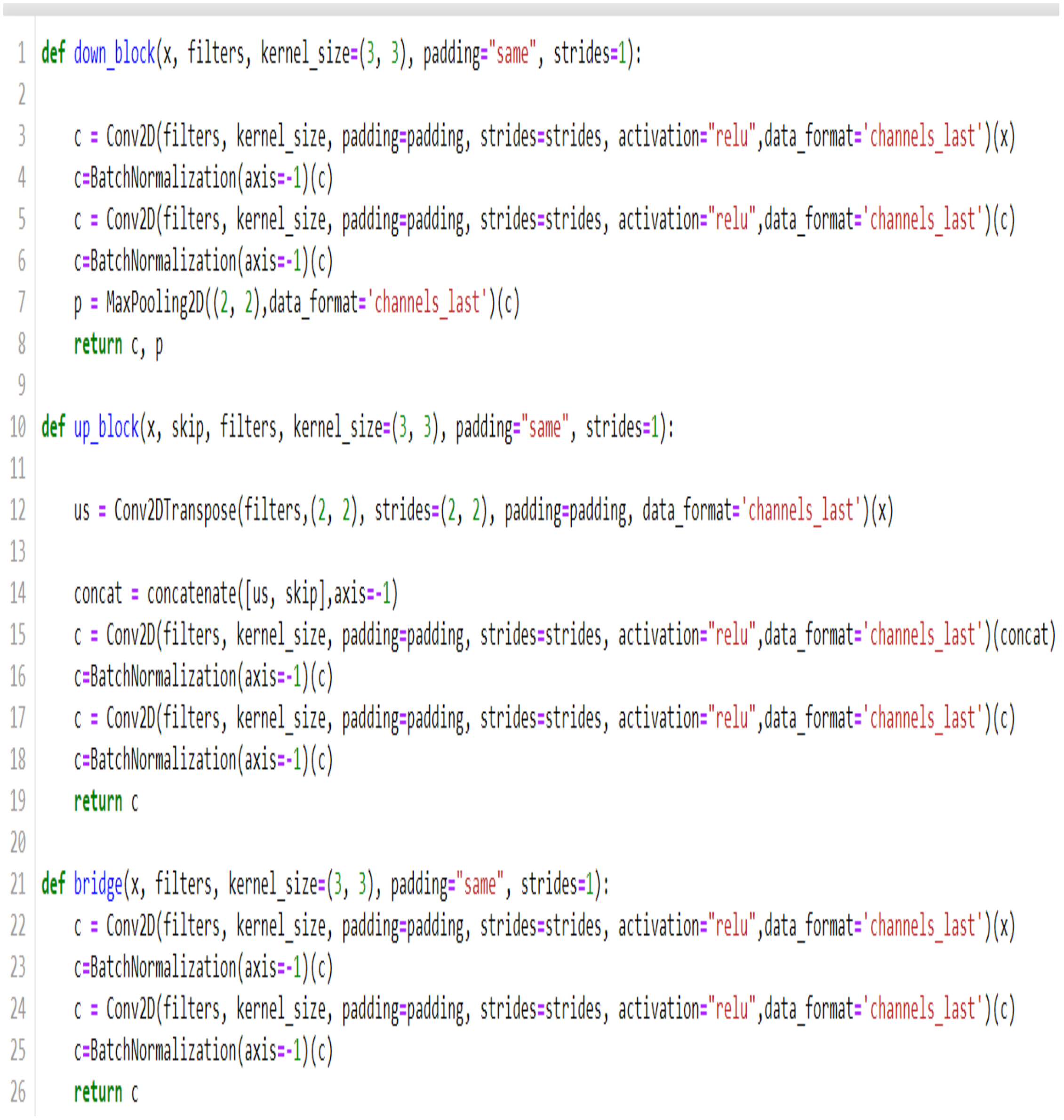


```python
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
import numpy as np
from keras.models import Model
from keras.layers import concatenate, Conv2D, MaxPooling2D, Conv2DTranspose, UpSampling2D,Add
from keras.layers import Input, BatchNormalization,Activation
import os
import matplotlib.pyplot as plt
from keras.optimizers import Adam
```

Figure 2.14 Codes of Common Libraries

1-U-net

I. Describe U-net Different Convolutional Blocks



```python
def down_block(x, filters, kernel_size=(3, 3), padding="same", strides=1):

    c = Conv2D(filters, kernel_size, padding=padding, strides=strides, activation="relu",data_format='channels_last')(x)
    c=BatchNormalization(axis=-1)(c)
    c = Conv2D(filters, kernel_size, padding=padding, strides=strides, activation="relu",data_format='channels_last')(c)
    c=BatchNormalization(axis=-1)(c)
    p = MaxPooling2D((2, 2),data_format='channels_last')(c)
    return c, p

def up_block(x, skip, filters, kernel_size=(3, 3), padding="same", strides=1):

    us = Conv2DTranspose(filters,(2, 2), strides=(2, 2), padding=padding, data_format='channels_last')(x)

    concat = concatenate([us, skip],axis=-1)
    c = Conv2D(filters, kernel_size, padding=padding, strides=strides, activation="relu",data_format='channels_last')(concat)
    c=BatchNormalization(axis=-1)(c)
    c = Conv2D(filters, kernel_size, padding=padding, strides=strides, activation="relu",data_format='channels_last')(c)
    c=BatchNormalization(axis=-1)(c)
    return c

def bridge(x, filters, kernel_size=(3, 3), padding="same", strides=1):
    c = Conv2D(filters, kernel_size, padding=padding, strides=strides, activation="relu",data_format='channels_last')(x)
    c=BatchNormalization(axis=-1)(c)
    c = Conv2D(filters, kernel_size, padding=padding, strides=strides, activation="relu",data_format='channels_last')(c)
    c=BatchNormalization(axis=-1)(c)
    return c
```

Figure 2.15 Codes of U-net Different Convolutional Blocks

II. Built U-net Model

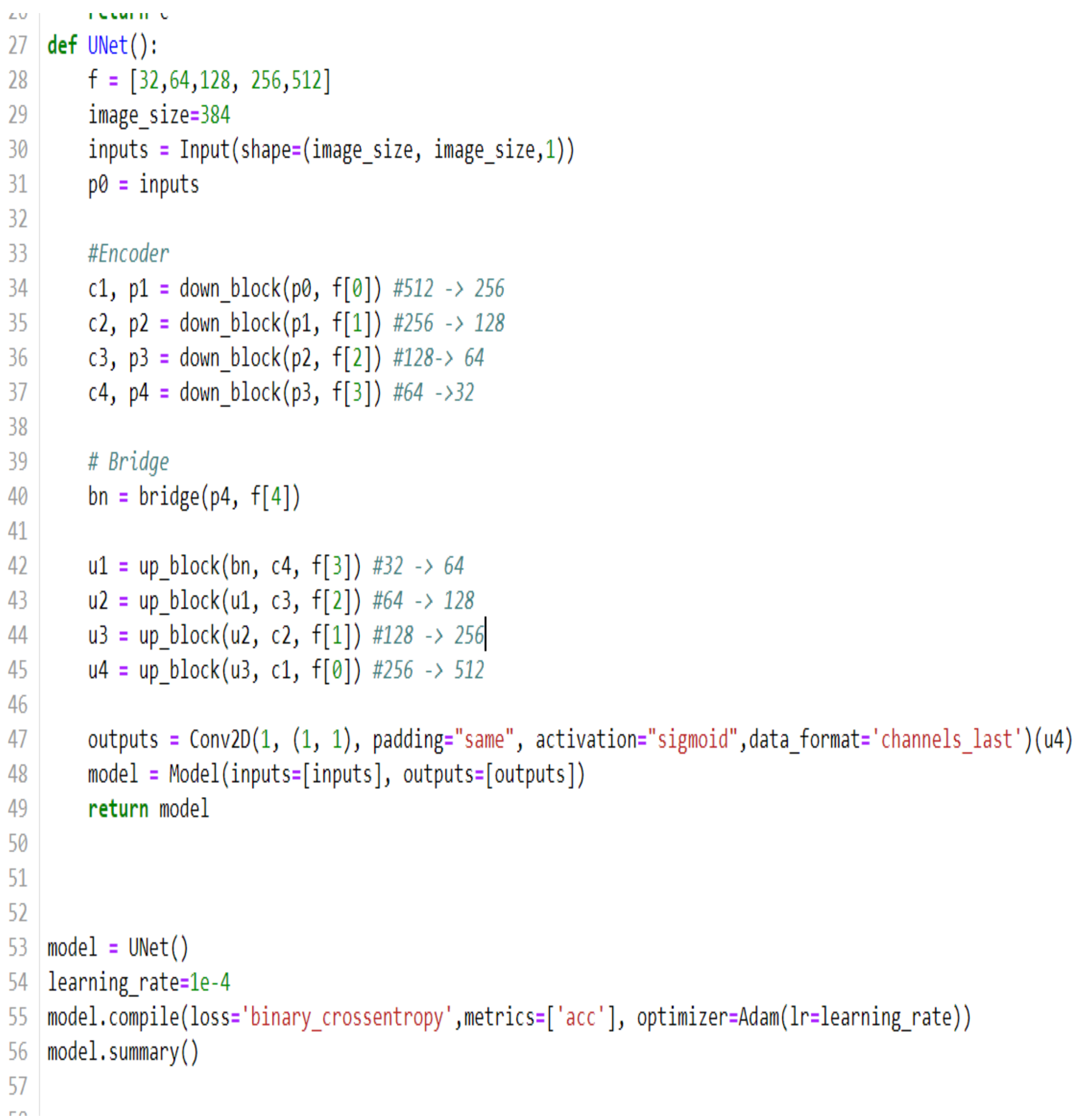
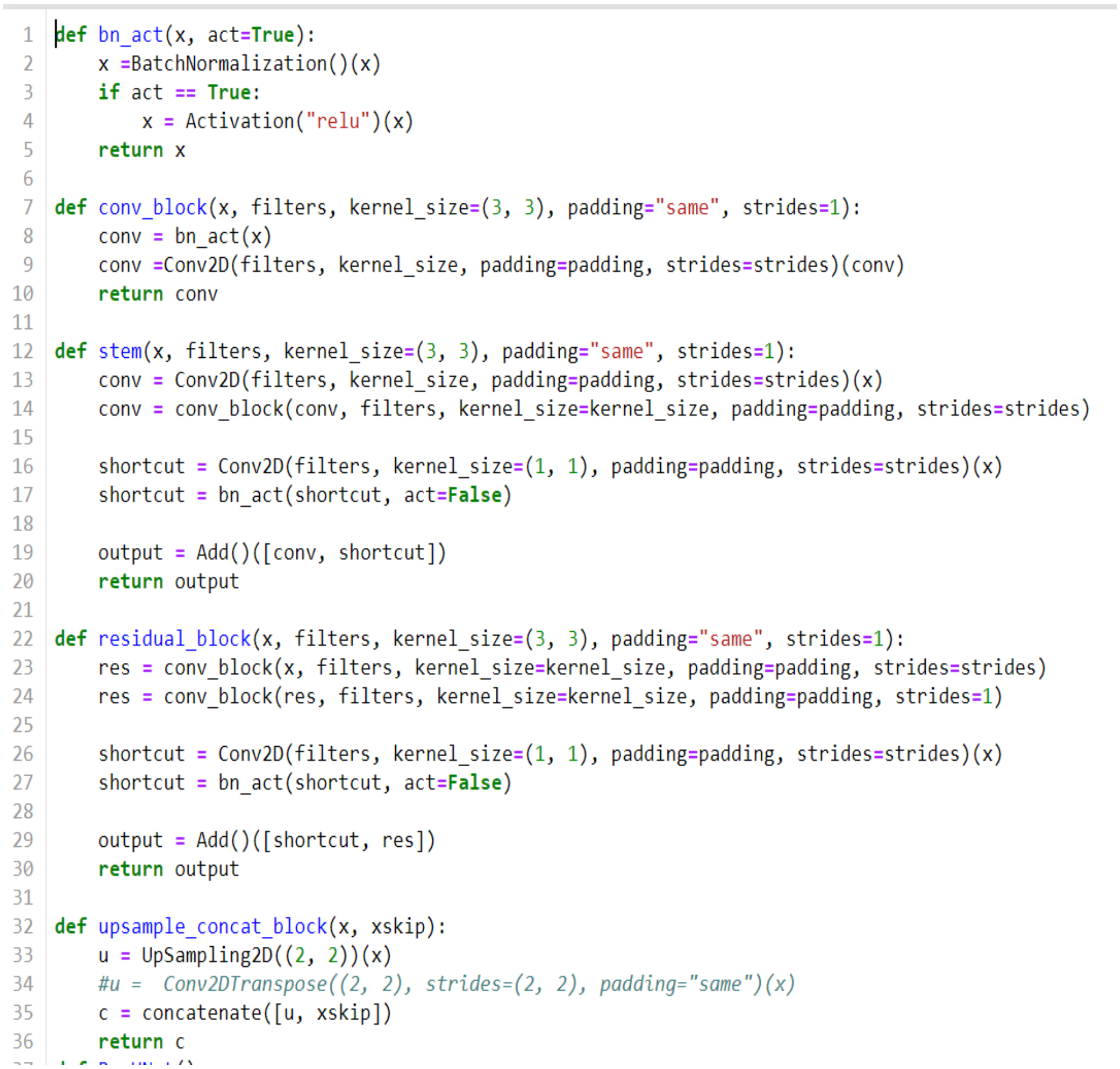

```
    return c
def UNet():
    f = [32,64,128, 256,512]
    image_size=384
    inputs = Input(shape=(image_size, image_size,1))
    p0 = inputs

    #Encoder
    c1, p1 = down_block(p0, f[0]) #512 -> 256
    c2, p2 = down_block(p1, f[1]) #256 -> 128
    c3, p3 = down_block(p2, f[2]) #128-> 64
    c4, p4 = down_block(p3, f[3]) #64 ->32

    # Bridge
    bn = bridge(p4, f[4])

    u1 = up_block(bn, c4, f[3]) #32 -> 64
    u2 = up_block(u1, c3, f[2]) #64 -> 128
    u3 = up_block(u2, c2, f[1]) #128 -> 256
    u4 = up_block(u3, c1, f[0]) #256 -> 512

    outputs = Conv2D(1, (1, 1), padding="same", activation="sigmoid",data_format='channels_last')(u4)
    model = Model(inputs=[inputs], outputs=[outputs])
    return model



model = UNet()
learning_rate=1e-4
model.compile(loss='binary_crossentropy',metrics=['acc'], optimizer=Adam(lr=learning_rate))
model.summary()
```

Figure 2.16 Codes of Encoder ,Decoder and Bridge of U-net

2- ResU-net

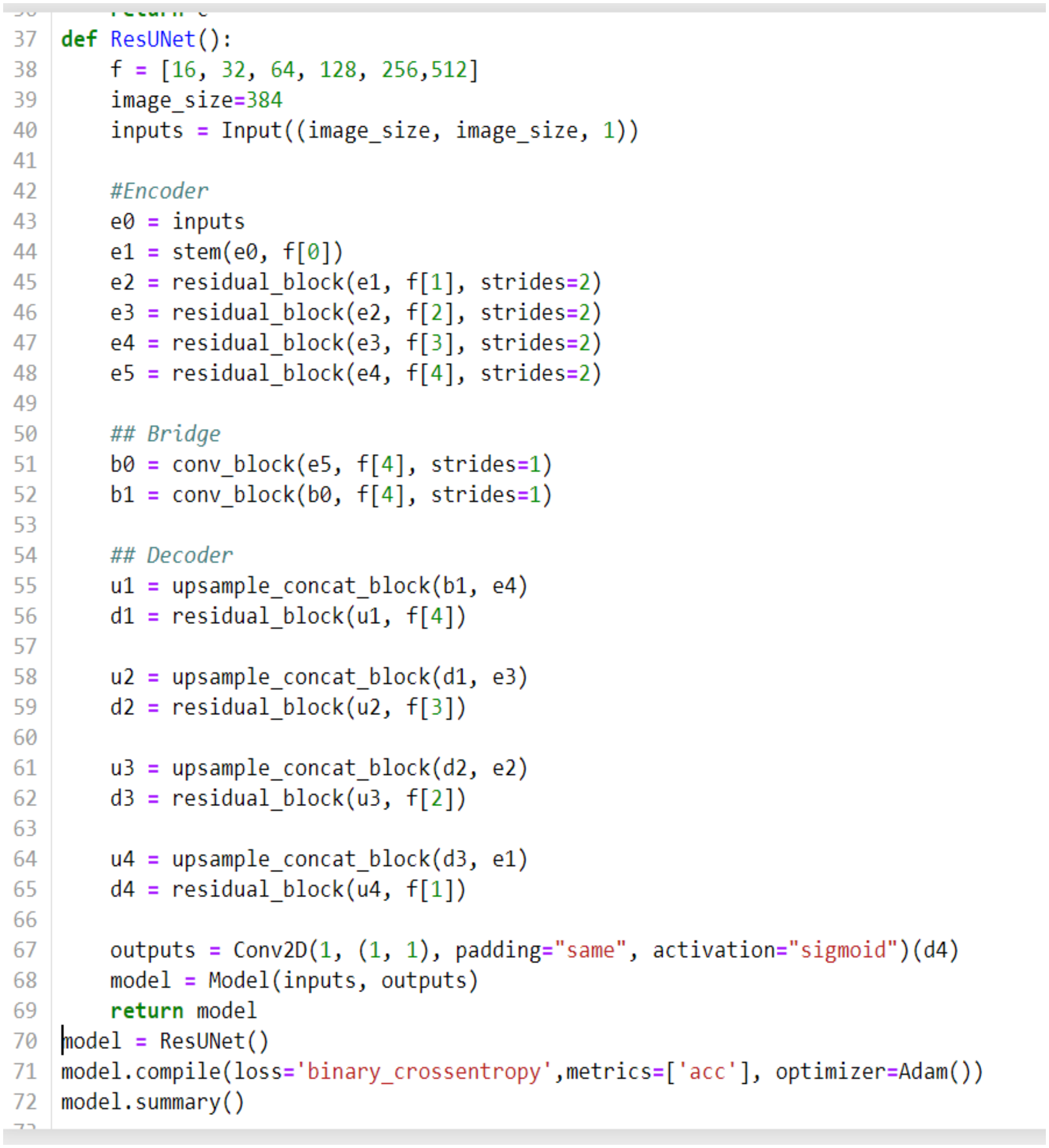
I. Describe ResU-net Different Blocks and Functions



```python
def bn_act(x, act=True):
    x =BatchNormalization()(x)
    if act == True:
        x = Activation("relu")(x)
    return x

def conv_block(x, filters, kernel_size=(3, 3), padding="same", strides=1):
    conv = bn_act(x)
    conv =Conv2D(filters, kernel_size, padding=padding, strides=strides)(conv)
    return conv

def stem(x, filters, kernel_size=(3, 3), padding="same", strides=1):
    conv = Conv2D(filters, kernel_size, padding=padding, strides=strides)(x)
    conv = conv_block(conv, filters, kernel_size=kernel_size, padding=padding, strides=strides)

    shortcut = Conv2D(filters, kernel_size=(1, 1), padding=padding, strides=strides)(x)
    shortcut = bn_act(shortcut, act=False)

    output = Add()([conv, shortcut])
    return output

def residual_block(x, filters, kernel_size=(3, 3), padding="same", strides=1):
    res = conv_block(x, filters, kernel_size=kernel_size, padding=padding, strides=strides)
    res = conv_block(res, filters, kernel_size=kernel_size, padding=padding, strides=1)

    shortcut = Conv2D(filters, kernel_size=(1, 1), padding=padding, strides=strides)(x)
    shortcut = bn_act(shortcut, act=False)

    output = Add()([shortcut, res])
    return output

def upsample_concat_block(x, xskip):
    u = UpSampling2D((2, 2))(x)
    #u =  Conv2DTranspose((2, 2), strides=(2, 2), padding="same")(x)
    c = concatenate([u, xskip])
    return c
```

Figure 2.17 Codes of ResU-net Different Convolutional Blocks and Functions

II. Built Res U-net Model

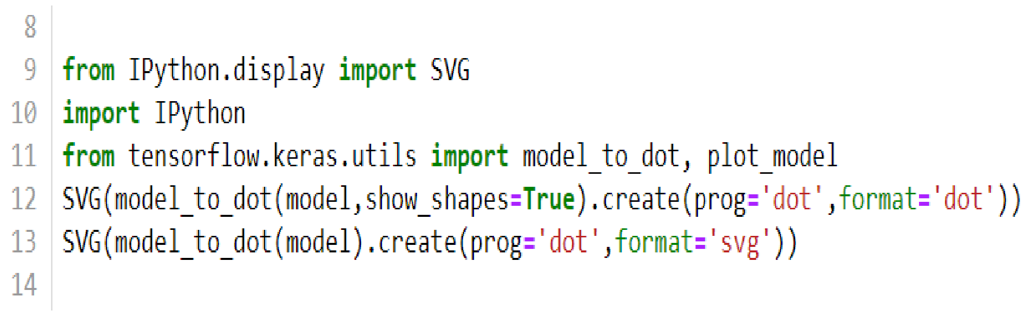


```
def ResUNet():
    f = [16, 32, 64, 128, 256,512]
    image_size=384
    inputs = Input((image_size, image_size, 1))

    #Encoder
    e0 = inputs
    e1 = stem(e0, f[0])
    e2 = residual_block(e1, f[1], strides=2)
    e3 = residual_block(e2, f[2], strides=2)
    e4 = residual_block(e3, f[3], strides=2)
    e5 = residual_block(e4, f[4], strides=2)

    ## Bridge
    b0 = conv_block(e5, f[4], strides=1)
    b1 = conv_block(b0, f[4], strides=1)

    ## Decoder
    u1 = upsample_concat_block(b1, e4)
    d1 = residual_block(u1, f[4])

    u2 = upsample_concat_block(d1, e3)
    d2 = residual_block(u2, f[3])

    u3 = upsample_concat_block(d2, e2)
    d3 = residual_block(u3, f[2])

    u4 = upsample_concat_block(d3, e1)
    d4 = residual_block(u4, f[1])

    outputs = Conv2D(1, (1, 1), padding="same", activation="sigmoid")(d4)
    model = Model(inputs, outputs)
    return model
model = ResUNet()
model.compile(loss='binary_crossentropy',metrics=['acc'], optimizer=Adam())
model.summary()
```

Figure 2.18 Codes of Encoder ,Decoder and Bridge of ResU-net

3- Display Diagram of Models

Codes are like below

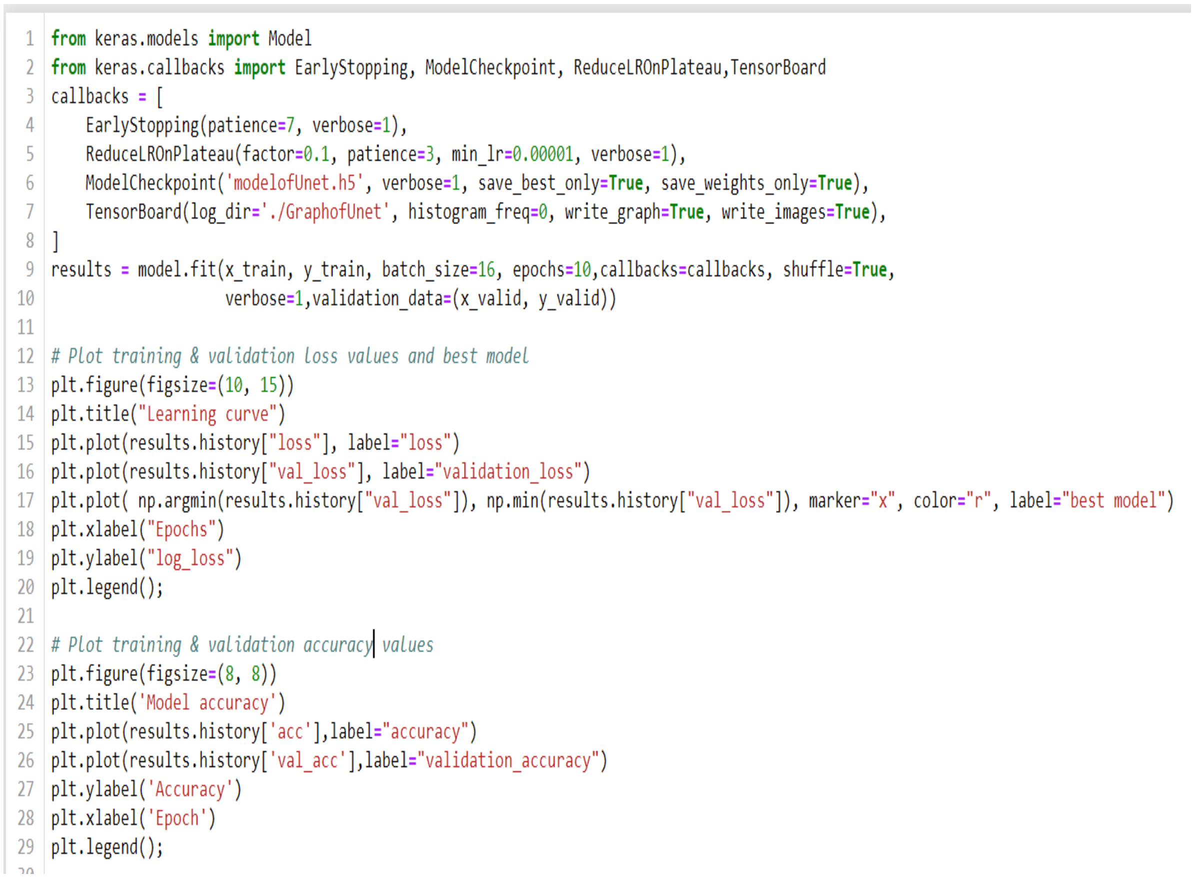


```python

from IPython.display import SVG
import IPython
from tensorflow.keras.utils import model_to_dot, plot_model
SVG(model_to_dot(model,show_shapes=True).create(prog='dot',format='dot'))
SVG(model_to_dot(model).create(prog='dot',format='svg'))

```

Figure 2.19 Codes of Display Diagram of Models

2.5.6.Fitting Models and Plotting Results

a. Describe callbacks for using fitting models.
b. Use model.fit function by taking as basis to validation dataset to start training
c. Plot validation&training loss values with best model and validation&training accuracy values.



```python
from keras.models import Model
from keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau,TensorBoard
callbacks = [
    EarlyStopping(patience=7, verbose=1),
    ReduceLROnPlateau(factor=0.1, patience=3, min_lr=0.00001, verbose=1),
    ModelCheckpoint('modelofUnet.h5', verbose=1, save_best_only=True, save_weights_only=True),
    TensorBoard(log_dir='./GraphofUnet', histogram_freq=0, write_graph=True, write_images=True),
]
results = model.fit(x_train, y_train, batch_size=16, epochs=10,callbacks=callbacks, shuffle=True,
                    verbose=1,validation_data=(x_valid, y_valid))

# Plot training & validation loss values and best model
plt.figure(figsize=(10, 15))
plt.title("Learning curve")
plt.plot(results.history["loss"], label="loss")
plt.plot(results.history["val_loss"], label="validation_loss")
plt.plot( np.argmin(results.history["val_loss"]), np.min(results.history["val_loss"]), marker="x", color="r", label="best model")
plt.xlabel("Epochs")
plt.ylabel("log_loss")
plt.legend();

# Plot training & validation accuracy values
plt.figure(figsize=(8, 8))
plt.title('Model accuracy')
plt.plot(results.history['acc'],label="accuracy")
plt.plot(results.history['val_acc'],label="validation_accuracy")
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend();
```

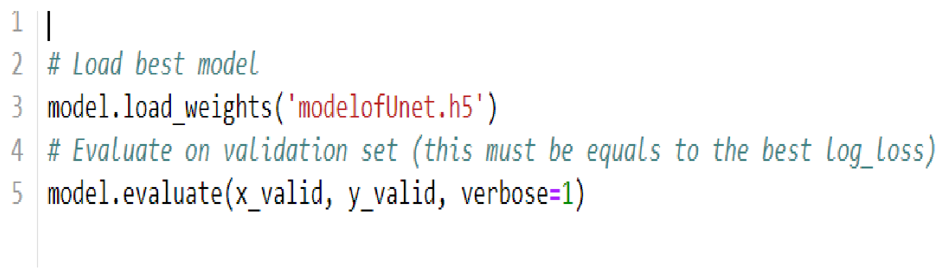
Figure 2.20 Codes of Fitting Models and Plotting Results

To plotting result , There is another way ise use of Tensorboard.On Spyder screen,

2.5.7.Evaluating Model

I. Load best model to start prediction.

II. Evaluate model on validation dataset.



```python

# Load best model
model.load_weights('modelofUnet.h5')
# Evaluate on validation set (this must be equals to the best log_loss)
model.evaluate(x_valid, y_valid, verbose=1)
```

Figure 2.21 Codes of Evaluating Models

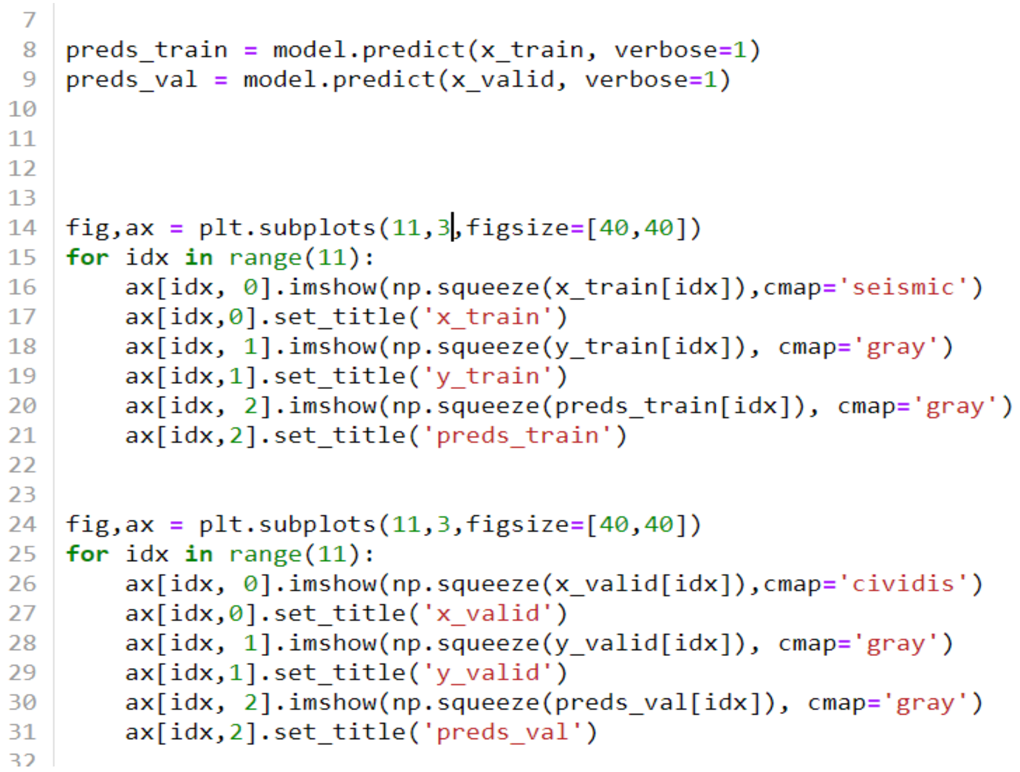
2.5.8.Prediction on Model and Displaying Results

Codes are below

I. Predict on training dataset.

II. Predict on validation dataset.

III. Show results.



```python

preds_train = model.predict(x_train, verbose=1)
preds_val = model.predict(x_valid, verbose=1)




fig,ax = plt.subplots(11,3,figsize=[40,40])
for idx in range(11):
    ax[idx, 0].imshow(np.squeeze(x_train[idx]),cmap='seismic')
    ax[idx,0].set_title('x_train')
    ax[idx, 1].imshow(np.squeeze(y_train[idx]), cmap='gray')
    ax[idx,1].set_title('y_train')
    ax[idx, 2].imshow(np.squeeze(preds_train[idx]), cmap='gray')
    ax[idx,2].set_title('preds_train')


fig,ax = plt.subplots(11,3,figsize=[40,40])
for idx in range(11):
    ax[idx, 0].imshow(np.squeeze(x_valid[idx]),cmap='cividis')
    ax[idx,0].set_title('x_valid')
    ax[idx, 1].imshow(np.squeeze(y_valid[idx]), cmap='gray')
    ax[idx,1].set_title('y_valid')
    ax[idx, 2].imshow(np.squeeze(preds_val[idx]), cmap='gray')
    ax[idx,2].set_title('preds_val')

```

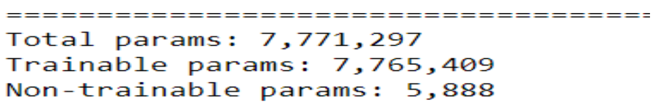
Figure 2.22 Codes of Prediction on Model and Display Results

3.RESULTS

3.1. U-net

Compile on Model

After compilation of model, Number of our trainable parameters is 7.765.409. Non-trainable parameters which are in hidden layer cannot be trained.



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

Figure 3.1 Non-Trainable and Trainable Parameters in U-net Model

Fit on Model

After fitting of model, graph of model loss model and accuracy were below. Addionally, It was shown where was best model.

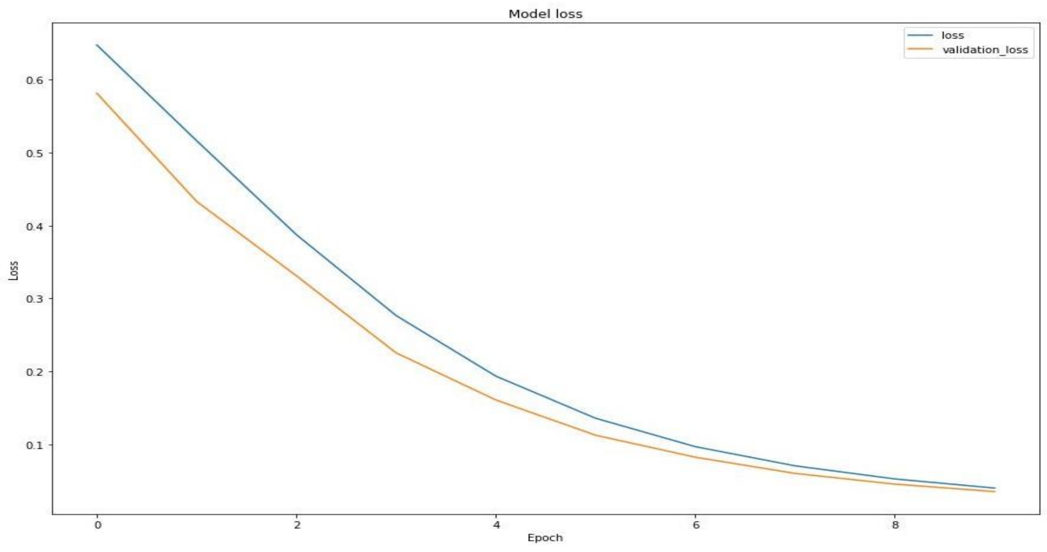


Table 3.1. Model loss in U-net model

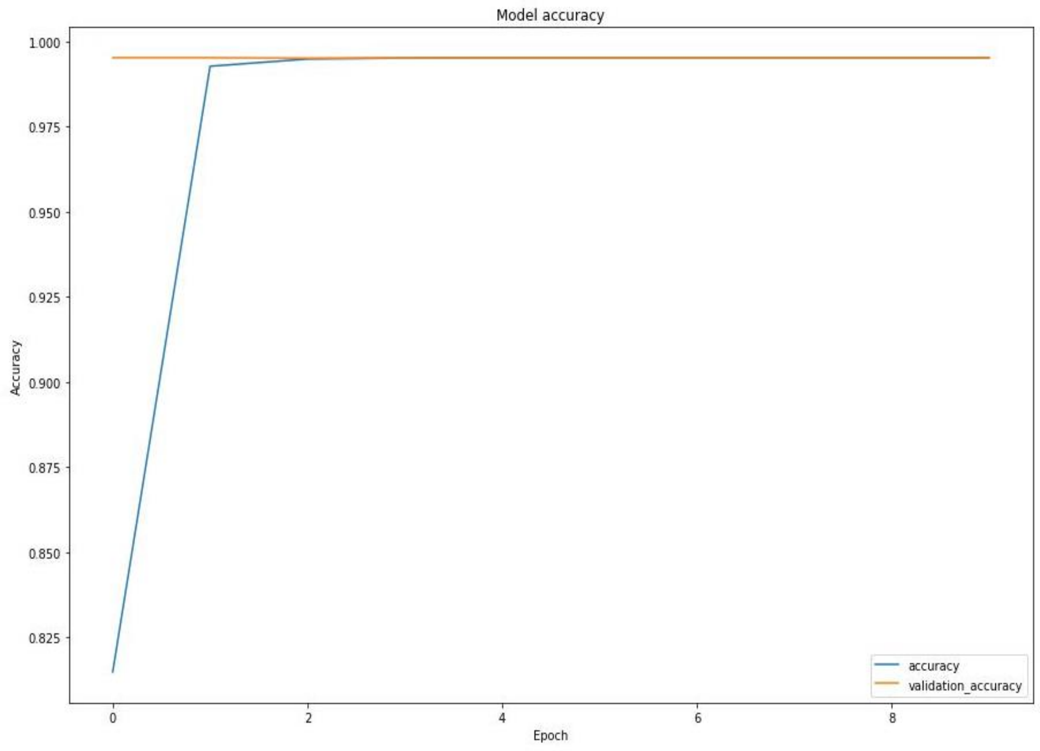


Table 3.2. Model accuracy in U-net model

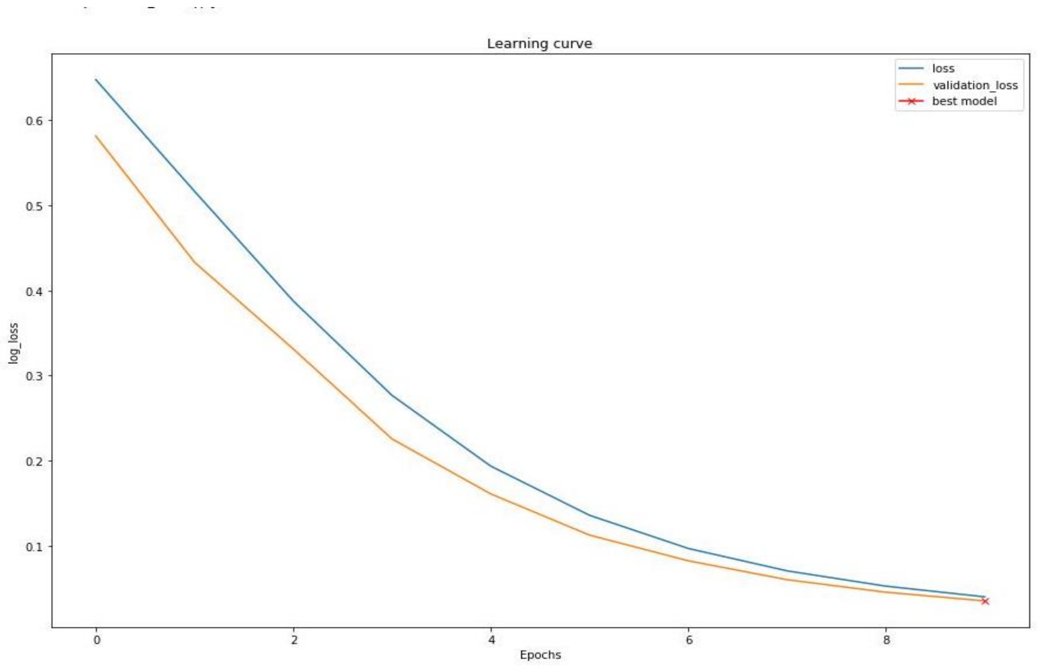
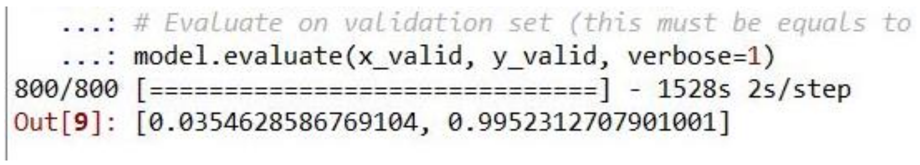


Table 3.3 Best Model in U-net Model

Evaluate on Model

After evaluation of model, The best validation loss and accuracy were shown below.



```
   ...: # Evaluate on validation set (this must be equals to
   ...: model.evaluate(x_valid, y_valid, verbose=1)
800/800 [==============================] - 1528s 2s/step
Out[9]: [0.0354628586769104, 0.9952312707901001]
```

Figure 3.2 The best loss and accuracy in U-net model

Predict on Train Dataset

After prediction on model, to the right way, orginal images, guide mask and prediction mask.To down, there were three different prediction on three images belonging to train dataset .

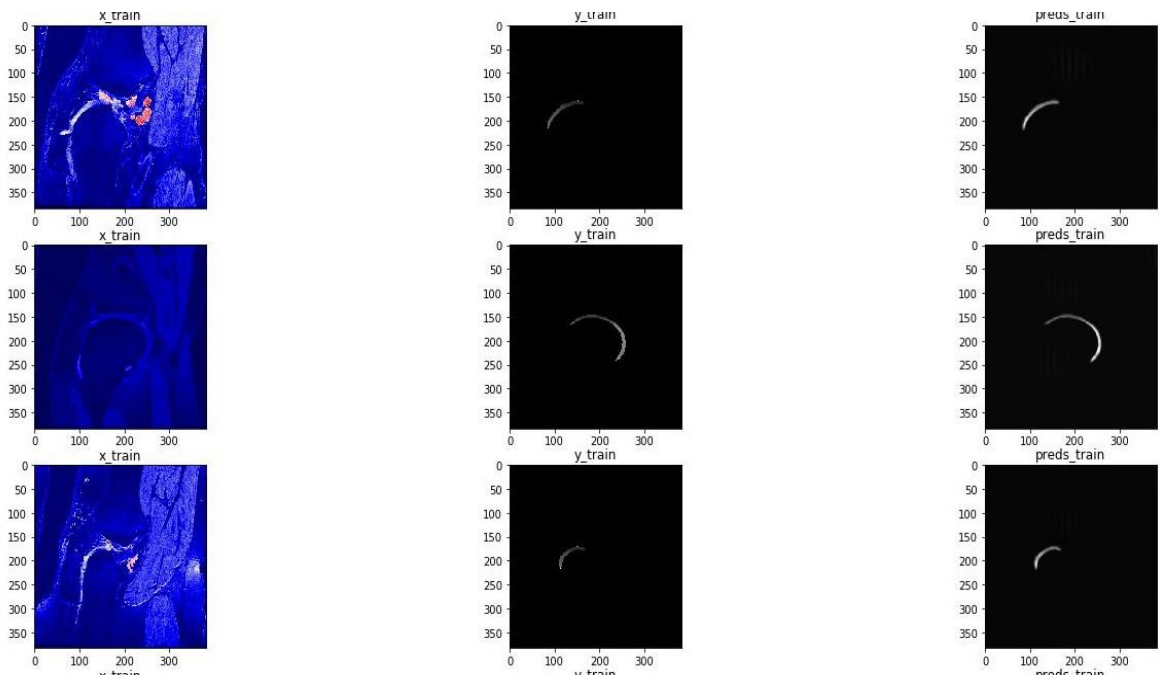


Figure 3.3 Pictures of train images(left) its train masks(middle) and predictions(right) mask on train dataset in U-net model

Prediction on Validation Dataset

After prediction on model, to the right way, orginal images, guide mask and prediction mask.To down, there were three different prediction on three images belonging to validation dataset .

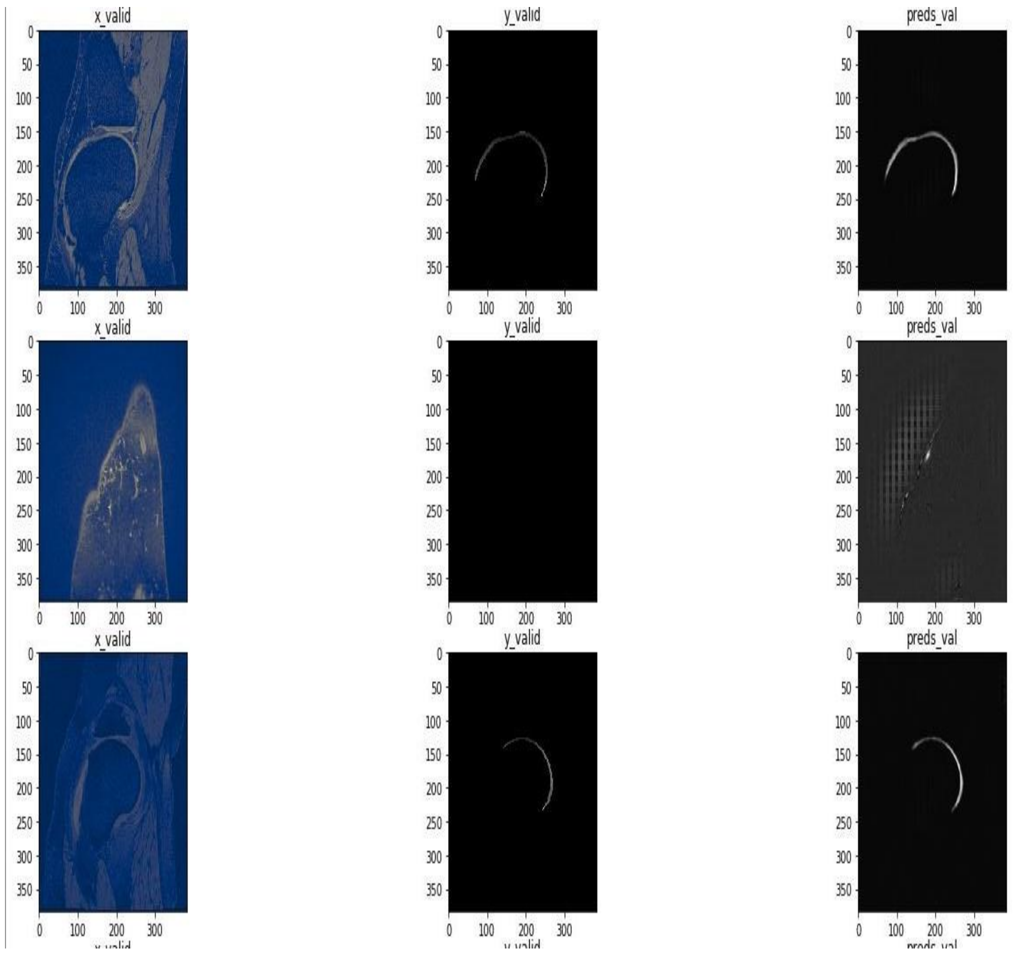
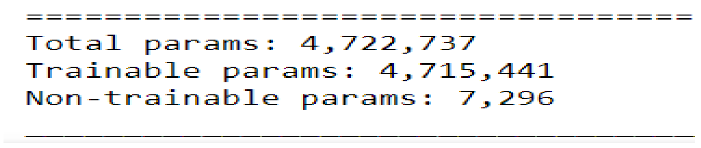


Figure 3.4 Pictures of train images(left) its train masks(middle) and predictions(right) mask on validation dataset in U-net model

3.2. ResU-net

Compile on Model

After compilation of model, Number of our trainable parameters is 4.715.441. Non-trainable parameters which are in hidden layer cannot be trained are 7.296.



```
====================================
Total params: 4,722,737
Trainable params: 4,715,441
Non-trainable params: 7,296
____________________________________
```

Figure 3.5 Non-Trainable and Trainable Parameters in ResU-net Model

Fit on Model

After fitting of model, graph of model loss model and accuracy were below. Addionally, It was shown where was best model.

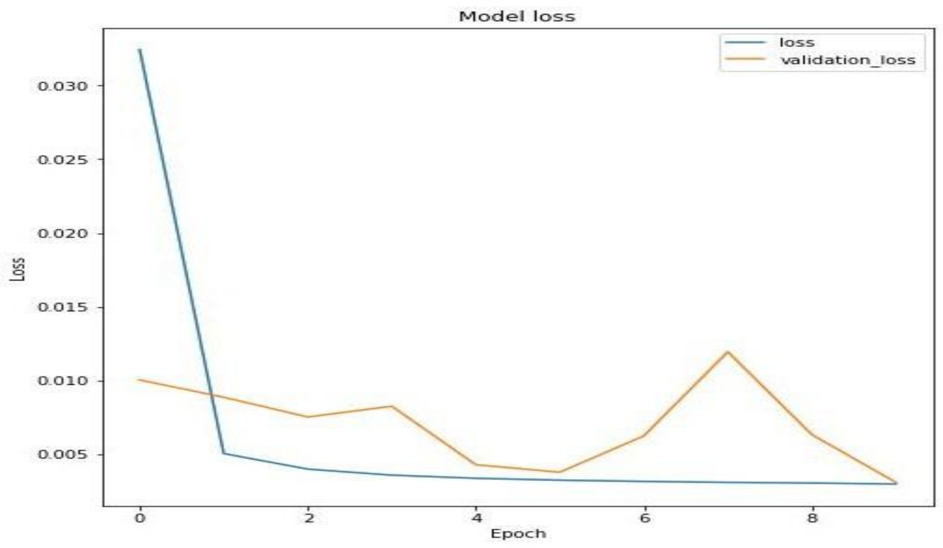


Table 3.4. Model loss in ResU-net Model

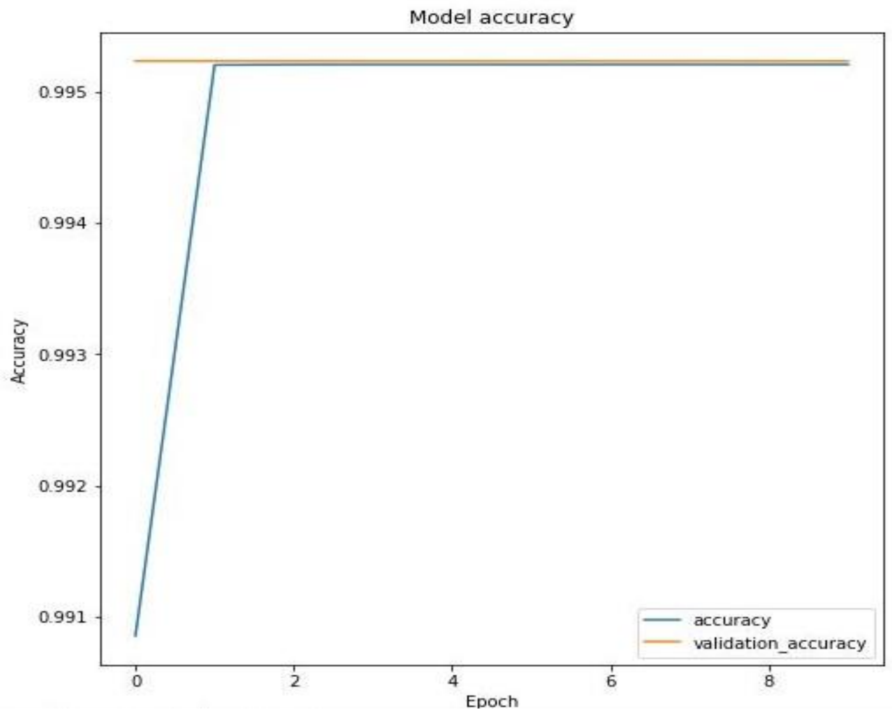


Table 3.5. Model accuracy in in ResU-net Model

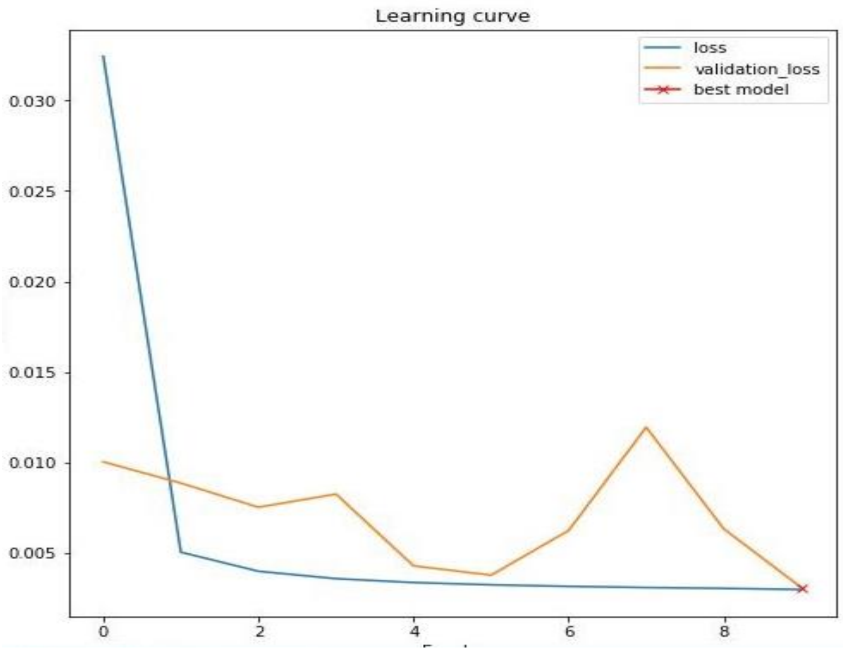
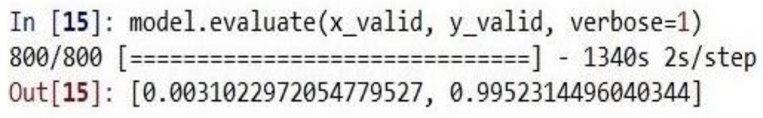


Table 3.6 Best Model in in ResU-net Model

Evaluate on Model

After evaluation of model, The best validation loss and accuracy were shown below.



```
In [15]: model.evaluate(x_valid, y_valid, verbose=1)
800/800 [==============================] - 1340s 2s/step
Out[15]: [0.0031022972054779527, 0.9952314496040344]
```

Figure 3.6 The best loss and accuracy values in in ResU-net Model

Predict on Train Dataset

After prediction on model, to the right way, orginal images, guide mask and prediction mask.To down, there were three different prediction on three images belonging to train dataset .

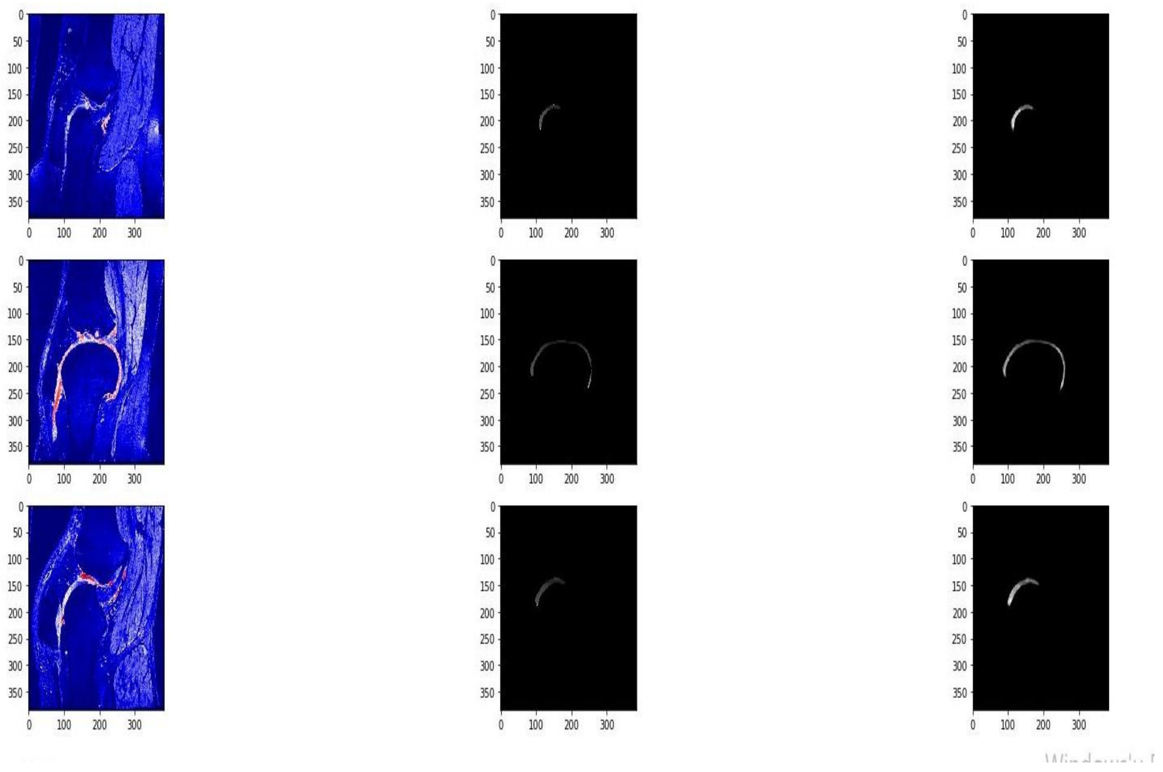


Figure 3.7 Pictures of train images(left) its train masks(middle) and predictions(right) mask on train dataset in ResU-net model

Prediction on Validation Dataset

After prediction on model, to the right way, orginal images, guide mask and prediction mask.To down, there were three different prediction on three images belonging to validation dataset .

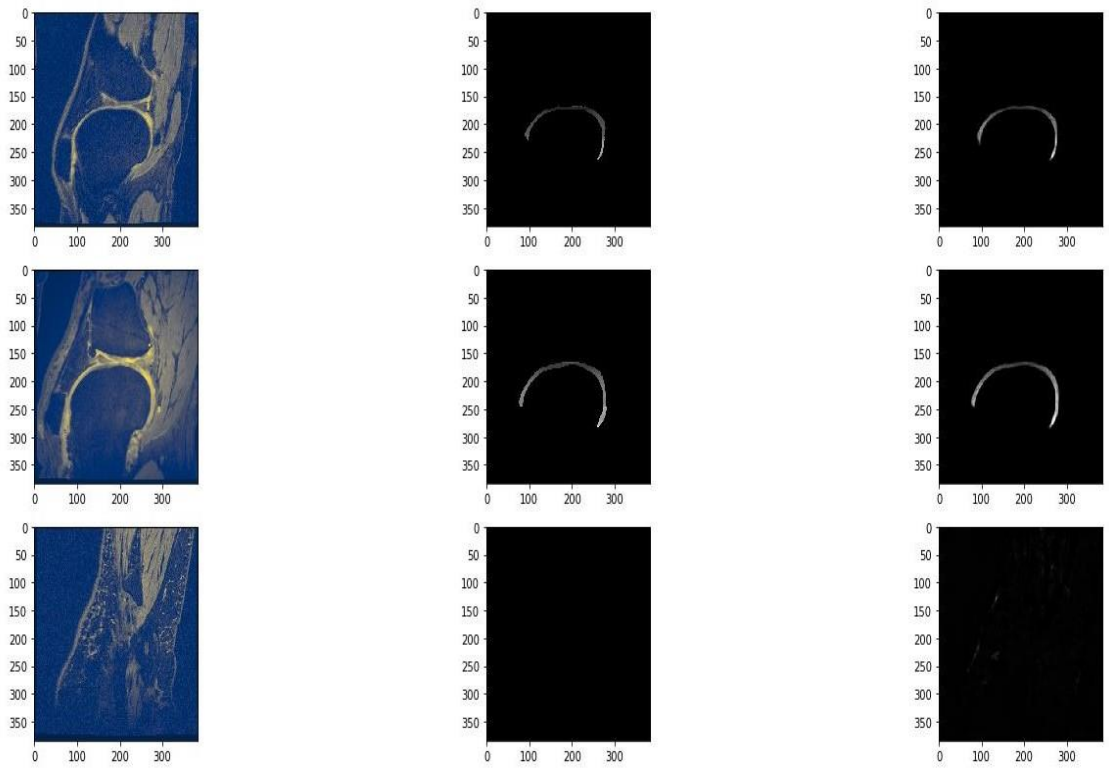


Figure 3.8 Pictures of train images(left) its train masks(middle) and predictions(right) mask on train dataset in ResU-net model

4. DISCUSSION

When to discuss result , We should taking a basis questions which are :

- How are close loss and validation each other?,
- How close are accuracy and validation accuracy each other?,
- How close are loss and validation to 0?,
- How close are accuracy and validation accuracy to 1?
- Is there enhancement after all epochs?

1-In U-net model:

After one epoch, loss=0.678 ,acc=0.711 ,val loss=0.75, val acc=9.09098e$^{-0.4}$.

In best learner epoch, loss=0.40,acc=0.995,,val loss=0.35, val acc=0.995.

At the start;

- Diffrence of loss and validation loss is 0.08.
- But values of them are very high to train.
- Difference of accuracy and validation accuracy is 8.29. This value is very high.
- So, when to look all values at the beginning, U-net model did not made good start.

At the its best learning;

- Diffrence of loss and validation loss is 0.05.
- Difference was reduced by 0.03 points after epochs.

- Their values are acceptable.
- Difference of accuracy and validation accuracy is 0. Their values are so close 100%.
- Between of first epoch and best epoch , loss was reduced by 0.27 and validation loss is reduced by 0.4. There is a very good development

2-In Res U-net model:

After one epoch, loss=0.40, ,acc=0. 990 ,val loss=0.1, val acc=0.90

In best learner epoch, loss=0.0032,acc=0.995,,val loss=0.0031, val acc=0.995.

At the start;

- Diffrence of loss and validation loss is 0.3.
- Values of them is very good according to beginning.
- But difference value of them are very high to train.
- Difference of accuracy and validation accuracy is 0.05, This value is very available.
- So, when to look all values at the beginning, Res U-net made good start.

At its best learning epoch;

- Diffrence of loss and validation loss is 0.0001.
- Difference was reduced by 0.2999 points after epochs. Their values are so acceptable.

- Difference of accuracy and validation accuracy is 0. Their values are so close 100%.
- Between of first epoch and best epoch , loss was reduced by 0.3968 and validation loss is reduced by 0.0969. There is a very good development.

When to compare U-net and Res U-net:

Apparently, the prediction of both is very good and Like they made same prediction analysis.Another way, Res U-net has the best model because it has lowest loss , validation loss and difference of them.The Winner is Res U-net mathematically.