

# UNet Model with Wavelet Pooling, measures - [accuracy, precision, recall] and compare with Max Pooling

- Under the Guidance of Jyoti Bharti Ma'am

## Imports

```
In [ ]: import os
import random

import numpy as np
import cv2
import tensorflow as tf
import pywt
from tensorflow import keras
from sklearn.metrics import roc_curve

import matplotlib.pyplot as plt

## Seeding
seed = 2019
random.seed = seed
np.random.seed = seed
tf.seed = seed
```

## Load Dataset

```
In [ ]: image_size = 128
# train_path = "dataset/stage1_train/"
train_path = "train"
epochs = 1000
batch_size = 8

train_ids = next(os.walk(train_path))[1]
val_data_size = 20

valid_ids = train_ids[:val_data_size]
# print(valid_ids)
train_ids = train_ids
```

## Custom Pooling Layer

```
In [ ]: import tensorflow as tf
from tensorflow.keras.layers import Layer

class CustomPoolingLayer(Layer):
    def __init__(self, call_function, compute_output_shape=None, **kwargs)
```

```

super(CustomPoolingLayer, self).__init__(**kwargs)
self._call_function = call_function
self._compute_output_shape_function = compute_output_shape_function

def call(self, inputs, training=None):
    return self._call_function(inputs)

def compute_output_shape(self, input_shape):
    if self._compute_output_shape_function:
        return self._compute_output_shape_function(input_shape)
    else:
        # If compute_output_shape_function is not provided, infer output shape
        output_shape = self._call_function(tf.zeros(input_shape))
        return output_shape.shape

def get_config(self):
    config = super(CustomPoolingLayer, self).get_config()
    return config

@classmethod
def from_config(cls, config):
    return cls(**config)

```

## Wavelet Pooling

```

In [ ]: import cv2
import pywt
import numpy as np
def wavelet_pooling(inputs):
    image = inputs
    # image_size = 128
    # image_path = "new.png"

    # image = cv2.imread(image_path, 1)
    # image = cv2.resize(image, (image_size, image_size))
    # print(image.shape)

    LL_channels = []
    for channel in range(image.shape[2]):
        coeffs = pywt.dwt2(image[:, :, channel], 'haar')
        LL, _, _, _ = coeffs
        LL_channels.append(LL)

    # Stack LL coefficients from each channel to reconstruct the LL coefficient image
    LL_image = np.stack(LL_channels, axis=-1)

    return LL_image
    # image = cv2.resize(image, (64, 64))
    # image = np.concatenate([image, image[:, :, 1:2]], axis=-1)

    # cv2.imshow("Image", image)

    # print(image.shape)

```

```
In [ ]: WaveletPooling2D = CustomPoolingLayer(call_function=wavelet_pooling)
```

## 1.2 Advantages

### Advantages:

1. The UNet combines the location information from the downsampling path to finally obtain a general information combining localisation and context, which is necessary to predict a good segmentation map.
2. No Dense layer is used, so image sizes can be used.

```
In [ ]: debug = True
WaveletPooling2D = CustomPoolingLayer(call_function=wavelet_pooling)
if(debug): WaveletPooling2D = keras.layers.MaxPool2D
```

## Data Generator

```
In [ ]: class DataBatchGenerator(keras.utils.Sequence):
    def __init__(self, ids, path, batch_size=8, image_size=128):
        # print(self.ids)
        self.ids = ids
        self.path = path
        self.batch_size = batch_size
        self.image_size = image_size
        self.on_epoch_end()

    def __load__(self, id_name):
        # Read and preprocess the image
        image_path = os.path.join(self.path, id_name, "images", id_name) + ".png"
        mask_path = os.path.join(self.path, id_name, "masks/")
        all_masks = os.listdir(mask_path)

        image = cv2.imread(image_path, 1)
        image = cv2.resize(image, (self.image_size, self.image_size))

        mask = np.zeros((self.image_size, self.image_size, 1))

        for name in all_masks:
            _mask_path = mask_path + name
            _mask_image = cv2.imread(_mask_path, -1)
            _mask_image = cv2.resize(_mask_image, (self.image_size, self.image_size))
            _mask_image = np.expand_dims(_mask_image, axis=-1)
            mask = np.maximum(mask, _mask_image)

        # Apply wavelet transformation (Haar wavelet)
        coeffs = pywt.dwt2(image, 'haar')
        LL, (LH, HL, HH) = coeffs
        image = LL

        image = np.reshape(image, (image.shape[0], image.shape[1], -1))
```

```

        image = cv2.resize(image, (128, 128))
        image = np.concatenate([image, image[:, :, 1:2]], axis=-1)
        image = image / 255.0
        mask = mask / 255.0

    return image, mask

def __getitem__(self, index):
    # Load data for a batch
    if (index + 1) * self.batch_size > len(self.ids):
        self.batch_size = len(self.ids) - index * self.batch_size

    files_batch = self.ids[index * self.batch_size : (index + 1) * self.batch_size]

    image = []
    mask = []

    for id_name in files_batch:
        _img, _mask = self.__load__(id_name)
        image.append(_img)
        mask.append(_mask)

    image = np.array(image)
    mask = np.array(mask)

    return image, mask

def on_epoch_end(self):
    pass

def __len__(self):
    # print(self.ids)
    return int(np.ceil(len(self.ids) / float(self.batch_size)))

train_gen = DataBatchGenerator(train_ids, train_path, image_size=image_size, batch_size=batch_size)
valid_gen = DataBatchGenerator(valid_ids, train_path, image_size=image_size, batch_size=batch_size)

```

In [ ]: # Data batch generator  
train\_gen = DataBatchGenerator(train\_ids, train\_path, image\_size=image\_size, batch\_size=batch\_size)  
print(train\_gen.\_\_getitem\_\_(4)[0].shape)  
# valid\_gen = DataBatchGenerator(valid\_ids, train\_path, image\_size=image\_size, batch\_size=batch\_size)

(8, 128, 128, 3)

In [ ]: gen = DataBatchGenerator(valid\_ids, train\_path, batch\_size=batch\_size, image\_size=image\_size)  
print(len(valid\_ids))  
x, y = gen.\_\_getitem\_\_(0)  
print(x.shape, y.shape)

20

(8, 128, 128, 3) (8, 128, 128, 1)

```
In [ ]: r = random.randint(0, len(x)-1)

# Generate a random index
r = random.randint(0, len(x)-1)

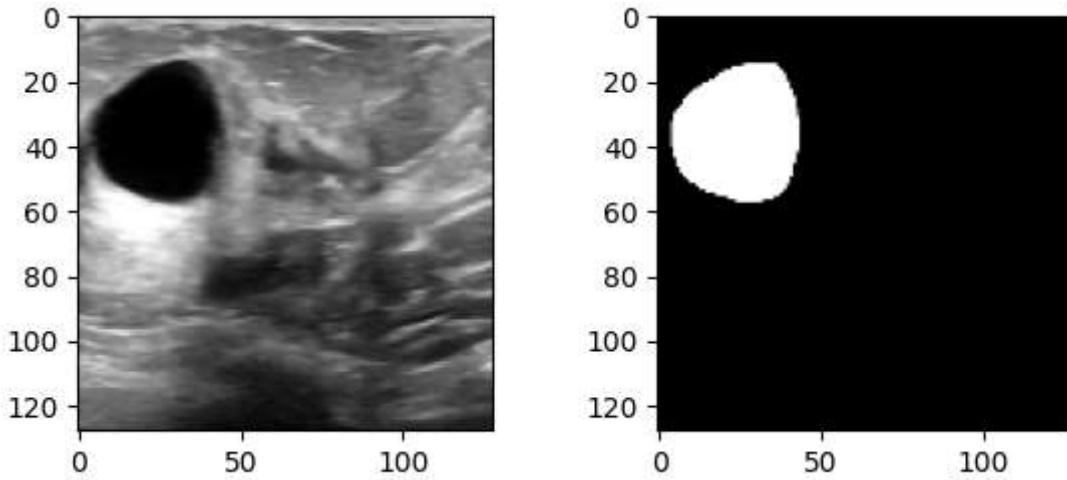
# Create a figure and subplots
fig = plt.figure()
fig.subplots_adjust(hspace=0.4, wspace=0.4)

# Plot the original image
ax = fig.add_subplot(1, 2, 1)
# Since the image has been transformed by wavelet, we may need to reshape it
# For example, if LL is the transformed image, we can plot it like this:
# ax.imshow(x[r], cmap='gray') # Assuming the transformed image is grayscale
# Assuming the transformed image has a shape of (128, 64, 3) after wavelet transform
ax.imshow(x[r][:, :, 0], cmap='gray') # Display the first channel of the RGB image

# If LL is a multi-channel image, we need to select one channel for display

# Plot the corresponding mask
ax = fig.add_subplot(1, 2, 2)
ax.imshow(np.reshape(y[r], (image_size, image_size)), cmap="gray") # Assuming y co
```

Out[ ]: <matplotlib.image.AxesImage at 0x2483227e660>



## Different Convolutional Blocks

```
In [ ]: def contraction_path(x, filters, kernel_size=(3, 3), padding="same", strides=1):
    c = keras.layers.Conv2D(filters, kernel_size, padding=padding, strides=strides,
    c = keras.layers.Conv2D(filters, kernel_size, padding=padding, strides=strides,
    p = WaveletPooling2D((2, 2), (2, 2))(c)
    return c, p

def expansive_path(x, skip, filters, kernel_size=(3, 3), padding="same", strides=1):
    us = keras.layers.UpSampling2D((2, 2))(x)
    concat = keras.layers.concatenate([us, skip])
    c = keras.layers.Conv2D(filters, kernel_size, padding=padding, strides=strides,
    c = keras.layers.Conv2D(filters, kernel_size, padding=padding, strides=strides,
```

```

    return c

def bottleneck(x, filters, kernel_size=(3, 3), padding="same", strides=1):
    c = keras.layers.Conv2D(filters, kernel_size, padding=padding, strides=strides,
    c = keras.layers.Conv2D(filters, kernel_size, padding=padding, strides=strides,
    return c

```

```

In [ ]: def contraction_path(x, filters, kernel_size=(3, 3), padding="same", strides=1):
    c = keras.layers.Conv2D(filters, kernel_size, padding=padding, strides=strides,
    c = keras.layers.Conv2D(filters, kernel_size, padding=padding, strides=strides,
    p = keras.layers.MaxPool2D((2, 2), (2, 2))(c)
    return c, p

def expansive_path(x, skip, filters, kernel_size=(3, 3), padding="same", strides=1):
    us = keras.layers.UpSampling2D((2, 2))(x)
    concat = keras.layers.concatenate([us, skip])
    c = keras.layers.Conv2D(filters, kernel_size, padding=padding, strides=strides,
    c = keras.layers.Conv2D(filters, kernel_size, padding=padding, strides=strides,
    return c

def bottleneck(x, filters, kernel_size=(3, 3), padding="same", strides=1):
    c = keras.layers.Conv2D(filters, kernel_size, padding=padding, strides=strides,
    c = keras.layers.Conv2D(filters, kernel_size, padding=padding, strides=strides,
    return c

```

## UNet Model

```

In [ ]: def UNet():
    f = [16, 32, 64, 128, 256]
    inputs = keras.layers.Input((image_size, image_size, 3))

    p0 = inputs
    c1, p1 = contraction_path(p0, f[0]) #128 -> 64
    c2, p2 = contraction_path(p1, f[1]) #64 -> 32
    c3, p3 = contraction_path(p2, f[2]) #32 -> 16
    c4, p4 = contraction_path(p3, f[3]) #16->8

    bn = bottleneck(p4, f[4])

    u1 = expansive_path(bn, c4, f[3]) #8 -> 16
    u2 = expansive_path(u1, c3, f[2]) #16 -> 32
    u3 = expansive_path(u2, c2, f[1]) #32 -> 64
    u4 = expansive_path(u3, c1, f[0]) #64 -> 128

    outputs = keras.layers.Conv2D(1, (1, 1), padding="same", activation="sigmoid")(
    model = keras.models.Model(inputs, outputs)
    return model

```

```

In [ ]: model = UNet()
model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["acc"])
model.summary()

```

Model: "functional\_9"

Layer (type)	Output Shape	Param #	Connected to
input_layer_4 (InputLayer)	(None, 128, 128, 3)	0	-
conv2d_76 (Conv2D)	(None, 128, 128, 16)	448	input_layer_4[0]...
conv2d_77 (Conv2D)	(None, 128, 128, 16)	2,320	conv2d_76[0][0]
max_pooling2d_16 (MaxPooling2D)	(None, 64, 64, 16)	0	conv2d_77[0][0]
conv2d_78 (Conv2D)	(None, 64, 64, 32)	4,640	max_pooling2d_16...
conv2d_79 (Conv2D)	(None, 64, 64, 32)	9,248	conv2d_78[0][0]
max_pooling2d_17 (MaxPooling2D)	(None, 32, 32, 32)	0	conv2d_79[0][0]
conv2d_80 (Conv2D)	(None, 32, 32, 64)	18,496	max_pooling2d_17...
conv2d_81 (Conv2D)	(None, 32, 32, 64)	36,928	conv2d_80[0][0]
max_pooling2d_18 (MaxPooling2D)	(None, 16, 16, 64)	0	conv2d_81[0][0]
conv2d_82 (Conv2D)	(None, 16, 16, 128)	73,856	max_pooling2d_18...
conv2d_83 (Conv2D)	(None, 16, 16, 128)	147,584	conv2d_82[0][0]
max_pooling2d_19 (MaxPooling2D)	(None, 8, 8, 128)	0	conv2d_83[0][0]
conv2d_84 (Conv2D)	(None, 8, 8, 256)	295,168	max_pooling2d_19...
conv2d_85 (Conv2D)	(None, 8, 8, 256)	590,080	conv2d_84[0][0]
up_sampling2d_16 (UpSampling2D)	(None, 16, 16, 256)	0	conv2d_85[0][0]
concatenate_16 (Concatenate)	(None, 16, 16, 384)	0	up_sampling2d_16... conv2d_83[0][0]
conv2d_86 (Conv2D)	(None, 16, 16, 128)	442,496	concatenate_16[0]...
conv2d_87 (Conv2D)	(None, 16, 16,	147,584	conv2d_86[0][0]

	128)		
up_sampling2d_17 (UpSampling2D)	(None, 32, 32, 128)	0	conv2d_87[0][0]
concatenate_17 (Concatenate)	(None, 32, 32, 192)	0	up_sampling2d_17... conv2d_81[0][0]
conv2d_88 (Conv2D)	(None, 32, 32, 64)	110,656	concatenate_17[0...]
conv2d_89 (Conv2D)	(None, 32, 32, 64)	36,928	conv2d_88[0][0]
up_sampling2d_18 (UpSampling2D)	(None, 64, 64, 64)	0	conv2d_89[0][0]
concatenate_18 (Concatenate)	(None, 64, 64, 96)	0	up_sampling2d_18... conv2d_79[0][0]
conv2d_90 (Conv2D)	(None, 64, 64, 32)	27,680	concatenate_18[0...]
conv2d_91 (Conv2D)	(None, 64, 64, 32)	9,248	conv2d_90[0][0]
up_sampling2d_19 (UpSampling2D)	(None, 128, 128, 32)	0	conv2d_91[0][0]
concatenate_19 (Concatenate)	(None, 128, 128, 48)	0	up_sampling2d_19... conv2d_77[0][0]
conv2d_92 (Conv2D)	(None, 128, 128, 16)	6,928	concatenate_19[0...]
conv2d_93 (Conv2D)	(None, 128, 128, 16)	2,320	conv2d_92[0][0]
conv2d_94 (Conv2D)	(None, 128, 128, 1)	17	conv2d_93[0][0]

Total params: 1,962,625 (7.49 MB)

Trainable params: 1,962,625 (7.49 MB)

Non-trainable params: 0 (0.00 B)

```
In [ ]: from keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D, Concatenate
from keras.models import Model

def build_model():
    # Define input layer
    inputs = Input(shape=(128, 128, 3), name='input_layer')

    # Define convolutional layers
    conv1 = Conv2D(16, (3, 3), activation='relu', padding='same', name='conv2d_76')
```

```

conv2 = Conv2D(16, (3, 3), activation='relu', padding='same', name='conv2d_77')
pool1 = MaxPooling2D((2, 2), name='max_pooling2d_16')(conv2)

conv3 = Conv2D(32, (3, 3), activation='relu', padding='same', name='conv2d_78')
conv4 = Conv2D(32, (3, 3), activation='relu', padding='same', name='conv2d_79')
pool2 = MaxPooling2D((2, 2), name='max_pooling2d_17')(conv4)

conv5 = Conv2D(64, (3, 3), activation='relu', padding='same', name='conv2d_80')
conv6 = Conv2D(64, (3, 3), activation='relu', padding='same', name='conv2d_81')
pool3 = MaxPooling2D((2, 2), name='max_pooling2d_18')(conv6)

conv7 = Conv2D(128, (3, 3), activation='relu', padding='same', name='conv2d_82')
conv8 = Conv2D(128, (3, 3), activation='relu', padding='same', name='conv2d_83')
pool4 = MaxPooling2D((2, 2), name='max_pooling2d_19')(conv8)

conv9 = Conv2D(256, (3, 3), activation='relu', padding='same', name='conv2d_84')
conv10 = Conv2D(256, (3, 3), activation='relu', padding='same', name='conv2d_85'

# Define upsampling layers
up1 = UpSampling2D((2, 2), name='up_sampling2d_16')(conv10)
concat1 = Concatenate(axis=-1, name='concatenate_16')([up1, conv8])
conv11 = Conv2D(128, (3, 3), activation='relu', padding='same', name='conv2d_86')
conv12 = Conv2D(128, (3, 3), activation='relu', padding='same', name='conv2d_87'

up2 = UpSampling2D((2, 2), name='up_sampling2d_17')(conv12)
concat2 = Concatenate(axis=-1, name='concatenate_17')([up2, conv6])
conv13 = Conv2D(64, (3, 3), activation='relu', padding='same', name='conv2d_88')
conv14 = Conv2D(64, (3, 3), activation='relu', padding='same', name='conv2d_89'

up3 = UpSampling2D((2, 2), name='up_sampling2d_18')(conv14)
concat3 = Concatenate(axis=-1, name='concatenate_18')([up3, conv4])
conv15 = Conv2D(32, (3, 3), activation='relu', padding='same', name='conv2d_90')
conv16 = Conv2D(32, (3, 3), activation='relu', padding='same', name='conv2d_91'

up4 = UpSampling2D((2, 2), name='up_sampling2d_19')(conv16)
concat4 = Concatenate(axis=-1, name='concatenate_19')([up4, conv2])
conv17 = Conv2D(16, (3, 3), activation='relu', padding='same', name='conv2d_92')
conv18 = Conv2D(16, (3, 3), activation='relu', padding='same', name='conv2d_93'

output = Conv2D(1, (1, 1), activation='sigmoid', name='conv2d_94')(conv18)

# Create model
model = Model(inputs=inputs, outputs=output)

return model

# Build the model
model = build_model()

model.load_weights("unet_1k_epochs_MIOS.weights.h5")

model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["acc"])

# Print model summary
model.summary()

```

Model: "functional\_13"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 128, 128, 3)	0	-
conv2d_76 (Conv2D)	(None, 128, 128, 16)	448	input_layer[0][0]
conv2d_77 (Conv2D)	(None, 128, 128, 16)	2,320	conv2d_76[0][0]
max_pooling2d_16 (MaxPooling2D)	(None, 64, 64, 16)	0	conv2d_77[0][0]
conv2d_78 (Conv2D)	(None, 64, 64, 32)	4,640	max_pooling2d_16...
conv2d_79 (Conv2D)	(None, 64, 64, 32)	9,248	conv2d_78[0][0]
max_pooling2d_17 (MaxPooling2D)	(None, 32, 32, 32)	0	conv2d_79[0][0]
conv2d_80 (Conv2D)	(None, 32, 32, 64)	18,496	max_pooling2d_17...
conv2d_81 (Conv2D)	(None, 32, 32, 64)	36,928	conv2d_80[0][0]
max_pooling2d_18 (MaxPooling2D)	(None, 16, 16, 64)	0	conv2d_81[0][0]
conv2d_82 (Conv2D)	(None, 16, 16, 128)	73,856	max_pooling2d_18...
conv2d_83 (Conv2D)	(None, 16, 16, 128)	147,584	conv2d_82[0][0]
max_pooling2d_19 (MaxPooling2D)	(None, 8, 8, 128)	0	conv2d_83[0][0]
conv2d_84 (Conv2D)	(None, 8, 8, 256)	295,168	max_pooling2d_19...
conv2d_85 (Conv2D)	(None, 8, 8, 256)	590,080	conv2d_84[0][0]
up_sampling2d_16 (UpSampling2D)	(None, 16, 16, 256)	0	conv2d_85[0][0]
concatenate_16 (Concatenate)	(None, 16, 16, 384)	0	up_sampling2d_16... conv2d_83[0][0]
conv2d_86 (Conv2D)	(None, 16, 16, 128)	442,496	concatenate_16[0...]
conv2d_87 (Conv2D)	(None, 16, 16,	147,584	conv2d_86[0][0]

	128)		
up_sampling2d_17 (UpSampling2D)	(None, 32, 32, 128)	0	conv2d_87[0][0]
concatenate_17 (Concatenate)	(None, 32, 32, 192)	0	up_sampling2d_17... conv2d_81[0][0]
conv2d_88 (Conv2D)	(None, 32, 32, 64)	110,656	concatenate_17[0...]
conv2d_89 (Conv2D)	(None, 32, 32, 64)	36,928	conv2d_88[0][0]
up_sampling2d_18 (UpSampling2D)	(None, 64, 64, 64)	0	conv2d_89[0][0]
concatenate_18 (Concatenate)	(None, 64, 64, 96)	0	up_sampling2d_18... conv2d_79[0][0]
conv2d_90 (Conv2D)	(None, 64, 64, 32)	27,680	concatenate_18[0...]
conv2d_91 (Conv2D)	(None, 64, 64, 32)	9,248	conv2d_90[0][0]
up_sampling2d_19 (UpSampling2D)	(None, 128, 128, 32)	0	conv2d_91[0][0]
concatenate_19 (Concatenate)	(None, 128, 128, 48)	0	up_sampling2d_19... conv2d_77[0][0]
conv2d_92 (Conv2D)	(None, 128, 128, 16)	6,928	concatenate_19[0...]
conv2d_93 (Conv2D)	(None, 128, 128, 16)	2,320	conv2d_92[0][0]
conv2d_94 (Conv2D)	(None, 128, 128, 1)	17	conv2d_93[0][0]

Total params: 1,962,625 (7.49 MB)

Trainable params: 1,962,625 (7.49 MB)

Non-trainable params: 0 (0.00 B)

## Training the model

```
In [ ]: train_gen = DataBatchGenerator(train_ids, train_path, image_size=image_size, batch_
valid_gen = DataBatchGenerator(valid_ids, train_path, image_size=image_size, batch_
train_steps = len(train_ids)//batch_size
valid_steps = len(valid_ids)//batch_size
```

```
model.fit(train_gen, validation_data=valid_gen, steps_per_epoch=int(train_steps), epochs=epochs)
```

## Testing the model

```
In [ ]: from sklearn.metrics import confusion_matrix, accuracy_score, roc_curve, auc

valid_gen = DataBatchGenerator(valid_ids, train_path, image_size=image_size, batch_size=batch_size)
x_batch, y_batch = valid_gen.__getitem__(0)

# print(x_batch.shape)

batch_accuracy = 0
results_batch = model.predict(x_batch)
results_batch = results_batch > 0.5
batch_accuracy=np.mean(results_batch == y_batch)

predictions = []
true_labels = []

for out, val in zip(results_batch, y_batch):

    out = np.squeeze(out)
    val = np.squeeze(val)

    val = val > 0.5

    for i in range(128):
        for j in range(128):
            predictions.append(out[i][j])
            true_labels.append(val[i][j])

    break

predictions = np.array(predictions)
true_labels = np.array(true_labels)

# print(predictions)
# print(true_labels)

conf_matrix = confusion_matrix(true_labels.flatten(), predictions.flatten())
accuracy = accuracy_score(true_labels.flatten(), predictions.flatten())
fpr, tpr, thresholds = roc_curve(true_labels.flatten(), predictions.flatten())
roc_auc = auc(fpr, tpr)

print("Confusion Matrix:")
print(conf_matrix)
print("\nAccuracy:", accuracy)
print("ROC AUC:", roc_auc)

TP = conf_matrix[1, 1]
FP = conf_matrix[0, 1]
```

```

FN = conf_matrix[1, 0]

# Compute Precision and Recall
precision = TP / (TP + FP + FN - FP)
recall = TP / (TP + FN)

print("Precision:", precision)
print("Recall:", recall)

# print(results_batch.shape)
# print(batch_accuracy*100)

```

1/1 ━━━━━━ 0s 52ms/step

Confusion Matrix:

```

[[16249    74]
 [    2    59]]

```

Accuracy: 0.995361328125

ROC AUC: 0.9813398171944847

Precision: 0.9672131147540983

Recall: 0.9672131147540983

```

In [ ]: model.save_weights("UNet.weights.h5")

x, y = valid_gen.__getitem__(0)
result = model.predict(x)

result = result > 0.5

```

1/1 ━━━━━━ 0s 63ms/step

```

In [ ]: fig = plt.figure()
fig.subplots_adjust(hspace=0.4, wspace=0.4)

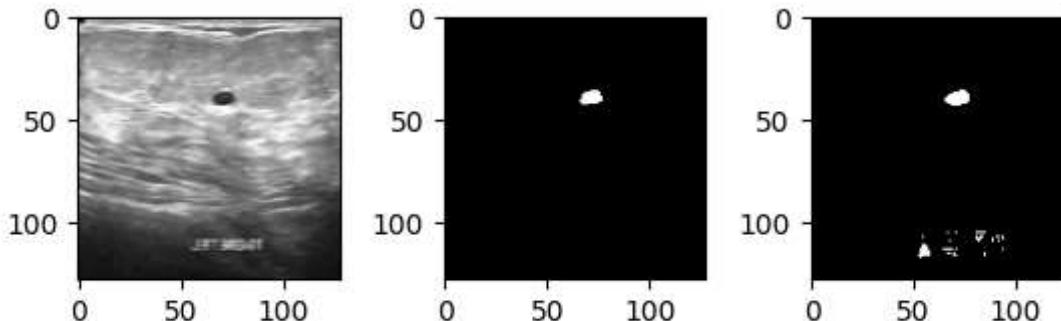
ax = fig.add_subplot(1, 3, 1)
ax.imshow(np.reshape(x[0][:, :, :1], (image_size, image_size)), cmap="gray")

ax = fig.add_subplot(1, 3, 2)
ax.imshow(np.reshape(y[0]*255, (image_size, image_size)), cmap="gray")

ax = fig.add_subplot(1, 3, 3)
ax.imshow(np.reshape(result[0]*255, (image_size, image_size)), cmap="gray")

```

Out[ ]: <matplotlib.image.AxesImage at 0x248303b0140>



```
In [ ]: fig = plt.figure()
fig.subplots_adjust(hspace=0.4, wspace=0.4)

ax = fig.add_subplot(1, 3, 1)
ax.imshow(np.reshape(x[1][:, :, :1], (image_size, image_size)), cmap="gray")

ax = fig.add_subplot(1, 3, 2)
ax.imshow(np.reshape(y[1]*255, (image_size, image_size)), cmap="gray")

ax = fig.add_subplot(1, 3, 3)
ax.imshow(np.reshape(result[1]*255, (image_size, image_size)), cmap="gray")
```

Out[ ]: <matplotlib.image.AxesImage at 0x24830530e00>

