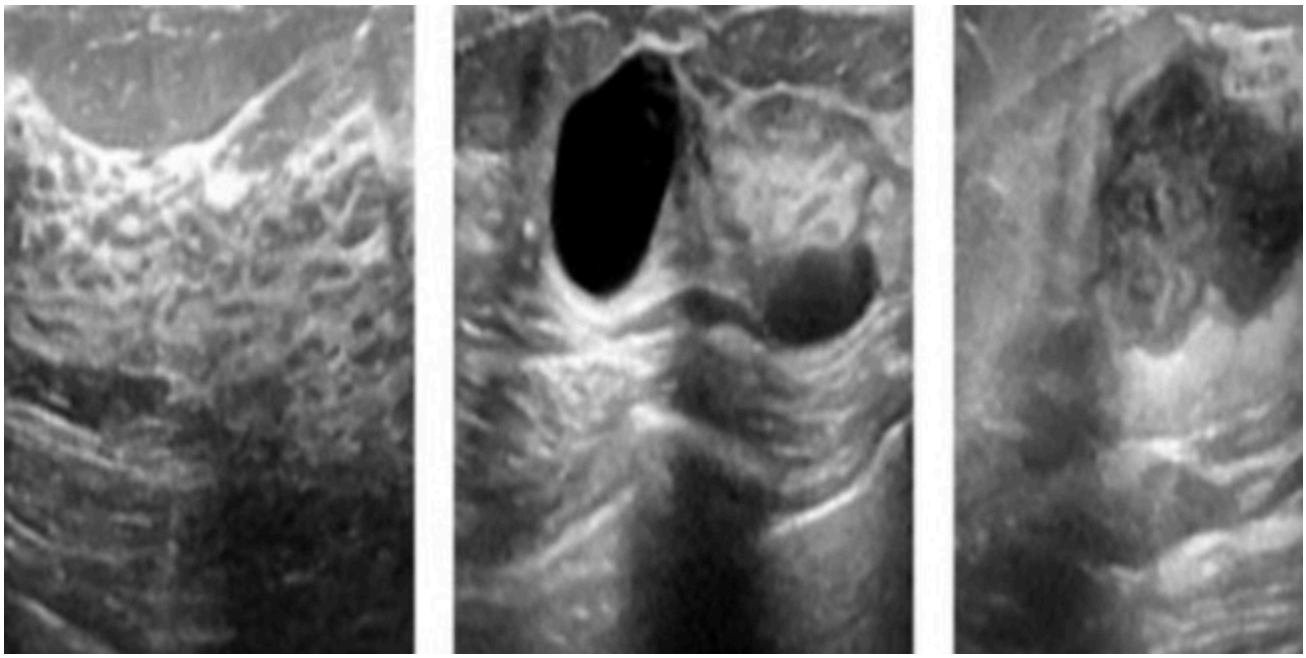
Hello! Welcome to my notebook💗💗💗



📌Importing all needed Librarires

In [23]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import cv2
from tqdm import tqdm
import random
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
from tensorflow.math import confusion_matrix
import seaborn as sns
tf.random.set_seed(3)
```

🔴*Data about Breast Ultrasound Images Dataset*

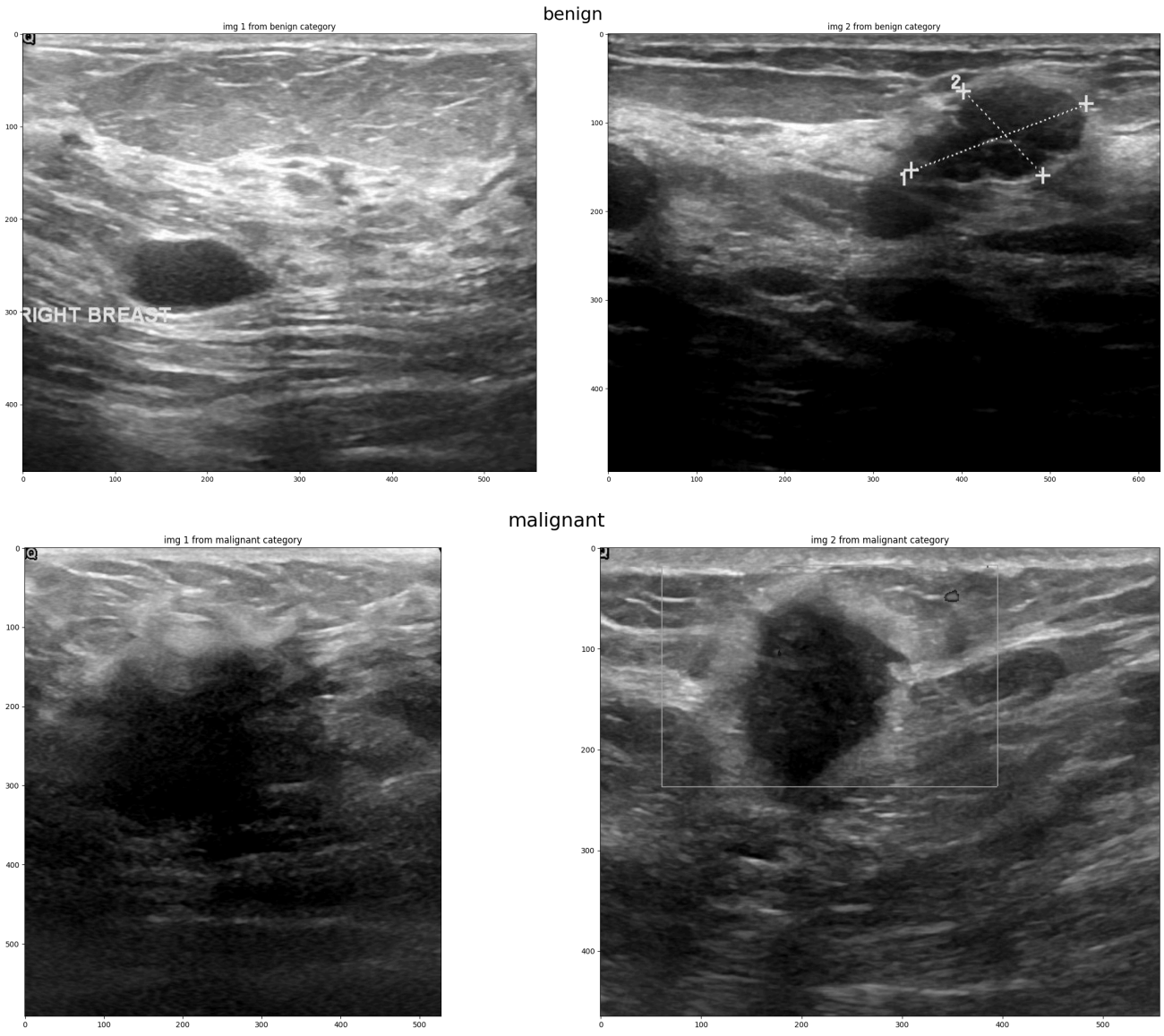
🔴*and contains 3 files (Classes) [benign,malignant,normal]*

📌 Display 2 images from each class

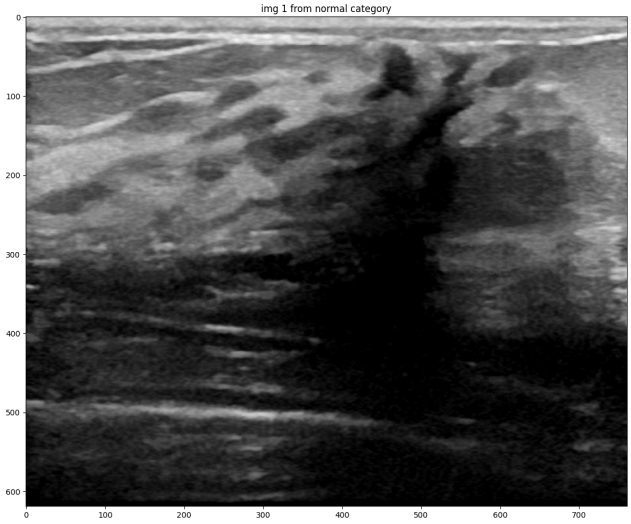
🔴*we notice that images with different sizes*

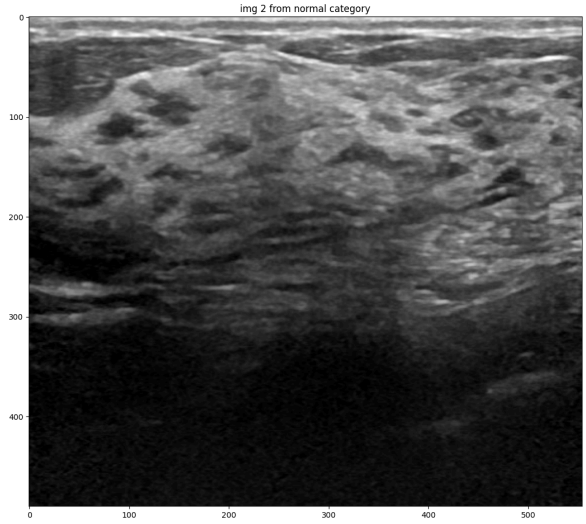
In [24]:
```
folder_name = '/kaggle/input/breast-ultrasound-images-dataset/Dataset_BUSI_with_GT'
files_names = ['benign', 'malignant', 'normal']
for file in files_names:
    path = os.path.join(folder_name, file)
    x = 0
    fig, axes = plt.subplots(1,2, figsize=(25, 10))
    for img in os.listdir(path):
        img_array = cv2.imread(os.path.join(path, img), cv2.IMREAD_GRAYSCALE)
        axes[x].imshow(img_array, cmap='gray')
        axes[x].set_title(f"img {x+1} from {file} category")
        x += 1
        if x == 2:
            break

    plt.suptitle(file, fontsize=26)
    plt.tight_layout()
    plt.show()
```



normal



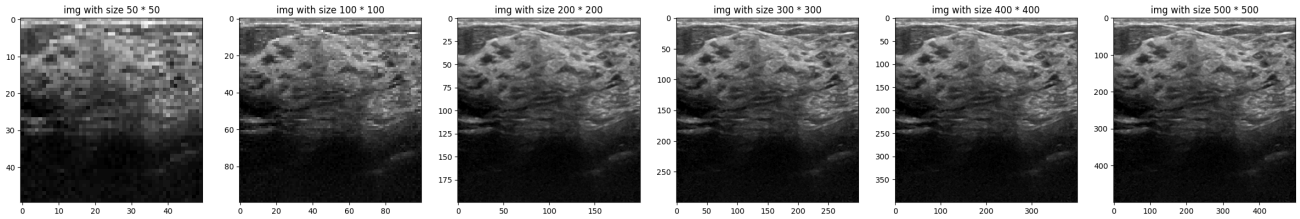


📌choose the best size for the images

In [25]:
```
img_sz = [50, 100, 200, 300, 400, 500]
plt.figure(figsize=(30, 10))

for i, sz in enumerate(img_sz):
    new_array = cv2.resize(img_array, (sz, sz))
    plt.subplot(1, len(img_sz), i+1)
    plt.imshow(new_array, cmap='gray')
    plt.title(f"img with size {sz} * {sz}")

plt.show()
```



In [26]:
```
#size (300*300) is okay
img_sz=300
```

📌Loading Data

In [27]:
```
training_data = []

def create_training_data():
    for file in files_names:
        path = os.path.join(folder_name, file)
        class_num = files_names.index(file)
        print(file,class_num)

        for img in tqdm(os.listdir(path)):
            img_array = cv2.imread(os.path.join(path, img), cv2.IMREAD_GRAYSCALE)
            new_array = cv2.resize(img_array, (img_sz, img_sz))  # Include resizing
            training_data.append([new_array, class_num])

create_training_data()
```

```
benign 0

100%|██████████| 891/891 [00:06<00:00, 144.69it/s]

malignant 1

100%|██████████| 421/421 [00:02<00:00, 148.77it/s]

normal 2

100%|██████████| 266/266 [00:02<00:00, 112.93it/s]
```

📌📌Display classes for the first 5 images

In [28]:
```
for i in range(5):
    print("Class number for image", i+1, ":", training_data[i][1])
```

```
Class number for image 1 : 0
Class number for image 2 : 0
Class number for image 3 : 0
Class number for image 4 : 0
Class number for image 5 : 0
```

📌📌Display classes for the last 5 images

In [29]:
```
for i in range(-1, -6, -1):
    print("Class number for image", len(training_data) + i + 1, ":", training_data[i]
```

```
Class number for image 1578 : 2
Class number for image 1577 : 2
Class number for image 1576 : 2
Class number for image 1575 : 2
Class number for image 1574 : 2
```

🔴***Now Data has benign images then malignant then normal***

📌 Shuffling the Data

📍📍Suffling and display the first 20 classes after suffling

In [30]:
```
random.shuffle(training_data)

for i in range(30):
    print(f"Sample {i+1}:")
    print("Class number:", training_data[i][1],"\n")
```

```
Sample 1:
Class number: 2

Sample 2:
Class number: 0

Sample 3:
Class number: 0

Sample 4:
Class number: 0

Sample 5:
Class number: 1

Sample 6:
Class number: 2

Sample 7:
Class number: 0

Sample 8:
Class number: 1

Sample 9:
Class number: 2

Sample 10:
Class number: 1

Sample 11:
Class number: 0

Sample 12:
Class number: 0

Sample 13:
Class number: 0

Sample 14:
Class number: 0

Sample 15:
Class number: 1

Sample 16:
Class number: 1

Sample 17:
Class number: 1

Sample 18:
Class number: 0

Sample 19:
Class number: 0

Sample 20:
Class number: 0

Sample 21:
Class number: 0

Sample 22:
Class number: 2
```

```
Sample 23:
Class number: 2

Sample 24:
Class number: 2

Sample 25:
Class number: 0

Sample 26:
Class number: 2

Sample 27:
Class number: 1

Sample 28:
Class number: 0

Sample 29:
Class number: 2

Sample 30:
Class number: 1
```

📌Feature Selection

In [31]:
```
X=[]
y=[]

for feature,label in training_data:
    X.append(feature)
    y.append(label)


X=np.array(X)
y=np.array(y)
```

📌📌Convert from list to numpy array

In [32]:
```
X=np.array(X)
y=np.array(y)
print(X.shape)
print(y.shape)
```

```
(1578, 300, 300)
(1578,)
```

🔴(300,300) is the shape of images in X

📌Spliting the Data ¶

In [33]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(1262, 300, 300)
(1262,)
(316, 300, 300)
(316,)
```

📌📌 Unique values in y (Classes)

In [34]:
```
print(np.unique(y_train))

print(np.unique(y_test))
```

```
[0 1 2]
[0 1 2]
```

📌 Scaling the values of X (images' Values)

In [35]:
```
X_train = X_train/255
X_test = X_test/255
```

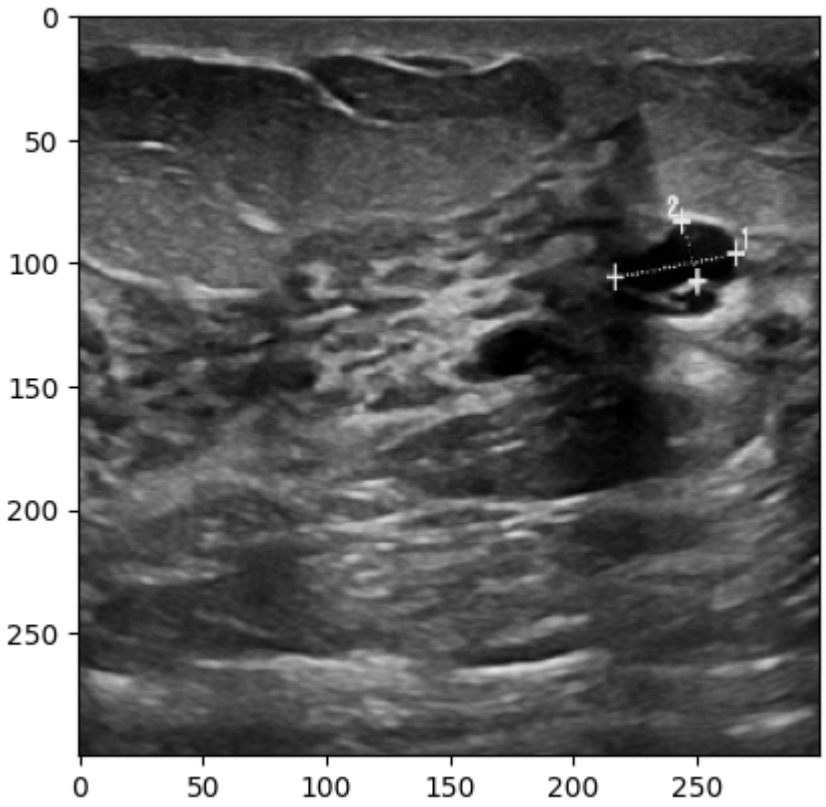
🔴 *Scaling the values to the range [0, 1]*

📌📌printing and display the first image after Scalling

In [36]:
```
print(X_train[0])
```

```
[[0.74901961 0.78431373 0.78823529 ... 0.74901961 0.74117647 0.72941176]
 [0.34901961 0.31764706 0.28235294 ... 0.34509804 0.34509804 0.35294118]
 [0.27058824 0.27058824 0.28627451 ... 0.2745098  0.2627451  0.23529412]
 ...
 [0.16078431 0.14117647 0.1254902  ... 0.1254902  0.1372549  0.15686275]
 [0.14117647 0.12941176 0.11764706 ... 0.14117647 0.17254902 0.18431373]
 [0.13333333 0.12941176 0.1372549  ... 0.16470588 0.2        0.19607843]]
```

In [55]:
```
plt.imshow(X_train[0:],cmap='gray')
plt.show()
```



📌 Building the Neural Network (NN)

In [38]:
```
model = keras.Sequential([
                          keras.layers.Flatten(input_shape=(img_sz,img_sz)),
                          keras.layers.Dense(256, activation='relu'),
                          keras.layers.Dense(128, activation='relu'),
                          keras.layers.Dense(64, activation='relu'),
                          keras.layers.Dense(32, activation='relu'),
                          keras.layers.Dense(16, activation='relu'),
                          keras.layers.Dense(8, activation='relu'),
                          keras.layers.Dense(3, activation='sigmoid')
])
```

```
/opt/conda/lib/python3.10/site-packages/keras/src/layers/reshaping/flatten.py:37: Us
erWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using
Sequential models, prefer using an `Input(shape)` object as the first layer in the m
odel instead.
  super().__init__(**kwargs)
```

📌 Compiling the Neural Network

In [39]:
```
model.compile(optimizer='adam',
              loss = 'sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

📌 Training the Neural Network

In [56]:
```
history=model.fit(X_train, y_train, epochs=30,validation_split=0.1)
```

```
Epoch 1/30
36/36 ━━━━━━━━━━━━━━━━━━━━ 12s 329ms/step - accuracy: 0.7489 - loss: 0.5640 - val_ac
curacy: 0.6850 - val_loss: 0.7725
Epoch 2/30
36/36 ━━━━━━━━━━━━━━━━━━━━ 12s 337ms/step - accuracy: 0.7463 - loss: 0.5407 - val_ac
curacy: 0.6457 - val_loss: 0.7986
Epoch 3/30
36/36 ━━━━━━━━━━━━━━━━━━━━ 11s 315ms/step - accuracy: 0.7360 - loss: 0.5351 - val_ac
curacy: 0.6457 - val_loss: 0.7728
Epoch 4/30
36/36 ━━━━━━━━━━━━━━━━━━━━ 12s 324ms/step - accuracy: 0.7787 - loss: 0.4816 - val_ac
curacy: 0.7087 - val_loss: 0.8002
Epoch 5/30
36/36 ━━━━━━━━━━━━━━━━━━━━ 13s 366ms/step - accuracy: 0.8014 - loss: 0.4738 - val_ac
curacy: 0.6693 - val_loss: 0.8852
Epoch 6/30
36/36 ━━━━━━━━━━━━━━━━━━━━ 19s 334ms/step - accuracy: 0.7881 - loss: 0.4609 - val_ac
curacy: 0.6457 - val_loss: 0.8946
Epoch 7/30
36/36 ━━━━━━━━━━━━━━━━━━━━ 19s 304ms/step - accuracy: 0.8041 - loss: 0.4270 - val_ac
curacy: 0.6693 - val_loss: 1.0482
Epoch 8/30
36/36 ━━━━━━━━━━━━━━━━━━━━ 21s 324ms/step - accuracy: 0.8249 - loss: 0.4220 - val_ac
curacy: 0.6772 - val_loss: 0.9925
Epoch 9/30
36/36 ━━━━━━━━━━━━━━━━━━━━ 12s 319ms/step - accuracy: 0.8109 - loss: 0.4384 - val_ac
curacy: 0.6535 - val_loss: 1.2328
Epoch 10/30
36/36 ━━━━━━━━━━━━━━━━━━━━ 12s 326ms/step - accuracy: 0.8233 - loss: 0.4216 - val_ac
curacy: 0.6772 - val_loss: 1.1012
Epoch 11/30
36/36 ━━━━━━━━━━━━━━━━━━━━ 20s 312ms/step - accuracy: 0.8191 - loss: 0.4280 - val_ac
curacy: 0.6614 - val_loss: 1.0729
Epoch 12/30
36/36 ━━━━━━━━━━━━━━━━━━━━ 21s 327ms/step - accuracy: 0.8602 - loss: 0.3559 - val_ac
curacy: 0.6850 - val_loss: 1.0110
Epoch 13/30
36/36 ━━━━━━━━━━━━━━━━━━━━ 20s 316ms/step - accuracy: 0.8261 - loss: 0.4291 - val_ac
curacy: 0.6693 - val_loss: 0.9142
Epoch 14/30
36/36 ━━━━━━━━━━━━━━━━━━━━ 12s 321ms/step - accuracy: 0.8521 - loss: 0.3745 - val_ac
curacy: 0.7559 - val_loss: 0.8368
Epoch 15/30
36/36 ━━━━━━━━━━━━━━━━━━━━ 11s 311ms/step - accuracy: 0.8687 - loss: 0.3105 - val_ac
curacy: 0.6693 - val_loss: 0.9030
Epoch 16/30
36/36 ━━━━━━━━━━━━━━━━━━━━ 11s 316ms/step - accuracy: 0.8644 - loss: 0.3565 - val_ac
curacy: 0.7402 - val_loss: 0.9700
Epoch 17/30
36/36 ━━━━━━━━━━━━━━━━━━━━ 20s 302ms/step - accuracy: 0.8917 - loss: 0.2763 - val_ac
curacy: 0.6693 - val_loss: 1.2158
Epoch 18/30
36/36 ━━━━━━━━━━━━━━━━━━━━ 12s 321ms/step - accuracy: 0.8469 - loss: 0.3847 - val_ac
curacy: 0.6929 - val_loss: 1.5180
Epoch 19/30
36/36 ━━━━━━━━━━━━━━━━━━━━ 20s 315ms/step - accuracy: 0.8529 - loss: 0.3850 - val_ac
curacy: 0.7008 - val_loss: 1.0553
Epoch 20/30
36/36 ━━━━━━━━━━━━━━━━━━━━ 21s 322ms/step - accuracy: 0.8635 - loss: 0.3584 - val_ac
curacy: 0.6929 - val_loss: 1.0173
Epoch 21/30
36/36 ━━━━━━━━━━━━━━━━━━━━ 11s 306ms/step - accuracy: 0.9034 - loss: 0.2548 - val_ac
curacy: 0.6693 - val_loss: 1.5579
Epoch 22/30
36/36 ━━━━━━━━━━━━━━━━━━━━ 11s 300ms/step - accuracy: 0.8550 - loss: 0.4031 - val_ac
```

```
curacy: 0.7402 - val_loss: 1.1816
Epoch 23/30
36/36 ━━━━━━━━━━━━━━━━━━━━ 21s 317ms/step - accuracy: 0.8831 - loss: 0.2907 - val_ac
curacy: 0.7087 - val_loss: 1.4306
Epoch 24/30
36/36 ━━━━━━━━━━━━━━━━━━━━ 21s 323ms/step - accuracy: 0.9198 - loss: 0.2302 - val_ac
curacy: 0.7244 - val_loss: 1.3726
Epoch 25/30
36/36 ━━━━━━━━━━━━━━━━━━━━ 20s 312ms/step - accuracy: 0.9052 - loss: 0.2373 - val_ac
curacy: 0.7008 - val_loss: 1.3171
Epoch 26/30
36/36 ━━━━━━━━━━━━━━━━━━━━ 12s 323ms/step - accuracy: 0.9075 - loss: 0.2369 - val_ac
curacy: 0.7559 - val_loss: 1.2762
Epoch 27/30
36/36 ━━━━━━━━━━━━━━━━━━━━ 11s 313ms/step - accuracy: 0.8939 - loss: 0.2539 - val_ac
curacy: 0.7717 - val_loss: 1.3931
Epoch 28/30
36/36 ━━━━━━━━━━━━━━━━━━━━ 21s 317ms/step - accuracy: 0.9349 - loss: 0.1553 - val_ac
curacy: 0.7323 - val_loss: 1.4709
Epoch 29/30
36/36 ━━━━━━━━━━━━━━━━━━━━ 20s 302ms/step - accuracy: 0.9372 - loss: 0.1792 - val_ac
curacy: 0.6614 - val_loss: 1.6293
Epoch 30/30
36/36 ━━━━━━━━━━━━━━━━━━━━ 21s 305ms/step - accuracy: 0.8719 - loss: 0.3496 - val_ac
curacy: 0.7323 - val_loss: 1.5908
```

📌 Model Summary

In [58]:
```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 90000)	0
dense (Dense)	(None, 256)	23,040,256
dense_1 (Dense)	(None, 128)	32,896
dense_2 (Dense)	(None, 64)	8,256
dense_3 (Dense)	(None, 32)	2,080
dense_4 (Dense)	(None, 16)	528
dense_5 (Dense)	(None, 8)	136
dense_6 (Dense)	(None, 3)	27

Total params: 69,252,539 (264.18 MB)

Trainable params: 23,084,179 (88.06 MB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 46,168,360 (176.12 MB)

📌 Evaluation the Model

In [59]:
```
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Accuarcy of the model is : {accuracy*100:.2f} %")
```

```
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - accuracy: 0.8093 - loss: 1.2151
Accuarcy of the model is : 77.85 %
```

📍📍Y prediction

In [60]:
```
y_pred=model.predict(X_test)
print(y_pred[0])
```

```
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
[9.997858e-01 1.000000e+00 9.142023e-25]
```

🔴*[first val is probability of class 0 "benign",second val is probability of class 1 "malignant",third val is probability of class 2"normal"]*

📍📍converting the prediction probabilities to class label

In [61]:
```
y_pred = [np.argmax(i) for i in y_pred]
print(y_pred)
```

```
[1, 2, 0, 1, 0, 0, 0, 0, 0, 2, 0, 2, 0, 0, 2, 0, 0, 0, 1, 0, 1, 0, 2, 0, 0, 2, 0, 0,
2, 2, 0, 2, 0, 0, 0, 0, 0, 0, 1, 0, 2, 0, 0, 0, 0, 0, 1, 2, 0, 0, 0, 0, 2, 0, 0, 0,
1, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0,
2, 0, 0, 1, 1, 0, 0, 1, 2, 0, 0, 0, 0, 0, 2, 1, 0, 0, 0, 2, 2, 0, 1, 0, 0, 2, 1, 2,
0, 0, 0, 2, 0, 0, 2, 2, 1, 0, 0, 1, 0, 2, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 2, 0, 2, 0,
0, 0, 0, 0, 1, 0, 0, 2, 0, 1, 1, 0, 0, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 2, 2, 0, 0, 1, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 2,
2, 2, 1, 0, 0, 0, 0, 0, 0, 1, 0, 2, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 2, 1, 0, 0, 2, 2,
0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 2, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
0, 2, 0, 0, 0, 0, 0, 0, 2, 2, 0, 1, 1, 0, 0, 0, 0, 0, 2, 0, 2, 1, 1, 0, 0, 0, 0, 0,
1, 2, 0, 0, 0, 0, 0, 2, 0, 1, 0, 2, 0, 0, 2, 0, 0, 1, 0, 2, 0, 0, 2, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 1, 0]
```

📌📌y_test vs y_pred

In [62]:
```
comparison_df = pd.DataFrame({ 'Actual': y_test,'Predicted': y_pred})

print(comparison_df[1:21])
```

```
    Actual  Predicted
1        2          2
2        0          0
3        1          1
4        0          0
5        0          0
6        0          0
7        0          0
8        2          0
9        2          2
10       0          0
11       2          2
12       1          0
13       1          0
14       2          2
15       0          0
16       0          0
17       0          0
18       1          1
19       1          0
20       1          1
```

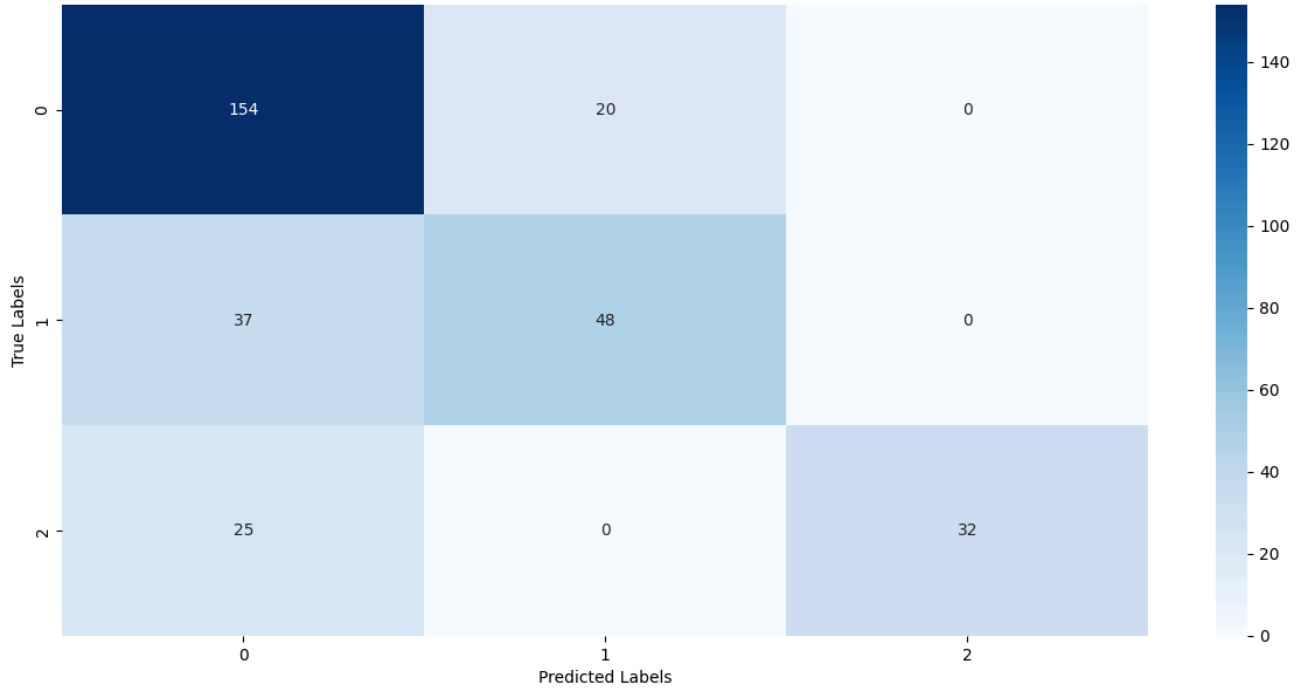
📌📌Confusion Matrix

In [63]:
```
conf_mat = confusion_matrix(y_test, y_pred)
print(conf_mat)
```

```
tf.Tensor(
[[160   7   7]
 [ 43  41   1]
 [ 12   0  45]], shape=(3, 3), dtype=int32)
```

In [47]:
```
plt.figure(figsize=(15,7))
sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues')
plt.ylabel('True Labels')
plt.xlabel('Predicted Labels')
```

Out[47]: Text(0.5, 47.7222222222222, 'Predicted Labels')



📌Classification Report

In [64]:
```
from sklearn.metrics import classification_report

print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.74      0.92      0.82       174
           1       0.85      0.48      0.62        85
           2       0.85      0.79      0.82        57

    accuracy                           0.78       316
   macro avg       0.82      0.73      0.75       316
weighted avg       0.79      0.78      0.77       316
```

Wish U luck ❤️

Ahmed Sheta

In [ ]:
```
p
```