SENIOR PROJECT SM2-2022

Convolutional Neural Networks for Medical Image Segmentation

Design Specification

Submitted to

School of Information, Computer and Communication Technology
Sirindhorn International Institute of Technology
Thammasat University

November 2022

by

Purana Junhaman 6222770909
Naravit Saenguthaijamorn 6222770123
Weerapat Tosup 6222770875
Bhramn Saidoung 622277107
Advisor: Dr. Stanislav S. Makhanov

Abstract

Imaging techniques are utilized to capture human body aberrations. The collected images must be interpreted in order to diagnose, prognostic, and plan treatment for the anomalies. Medical image interpretation is often performed by trained medical personnel. However, the scarcity of human experts, as well as their exhaustion and rough estimation techniques, limit the effectiveness of image understanding conducted by experienced medical professionals. Convolutional neural networks (CNNs) are powerful picture understanding techniques. In many picture interpretation challenges, they outperformed human specialists. The purpose of this article is to provide a complete assessment of CNN applications in medical image interpretation. The overarching goal is to encourage medical image understanding experts to use CNNs extensively in their research and diagnosis.A brief overview of CNNs has been provided. CNN and its multiple award-winning frameworks have been discussed. The key medical image understanding tasks have been introduced, namely picture classification, segmentation, localization, and detection. CNN applications in medical image interpretation of brain, breast, lung, and other organ illnesses have been critically and completely reviewed. A critical evaluation of some of the issues is also provided.

Acknowledgements

This study was supported by Dr. Stanislav S. Makhanov.

Contents

List of Figures

Chapter 1

Project Concept

1.1 Summary

Effectively identifying medical images is critical for clinical care and treatment. Although classic machine learning approaches, such as support vector methods (SVMs), have been routinely employed in medical image classification, they have significant drawbacks: performance is far from practical, and development has been slower in recent years compared to newer methods. Deep neural networks (DNN), particularly convolutional neural networks (CNNs), have been widely employed in recognizing picture classification tasks and have obtained significantly improved performance since 2012. For the time being, this research proposes to use CNN to classify 300 malignant growth photos based on their shapes.

1.2 Motivation

In the field of medical image analysis, the ultimate goal is to help improving the doctor to diagnose the cancer. Deep learning can assist in a variety of broad ways to accomplish this goal, including:

- Reducing the workload on medical professionals and the healthcare system by speeding up and streamlining the examination of medical images by partially or completely automating processes including diagnosis, outcome prediction, image quantification, and image reconstruction.

- Developing technology which enables completely novel clinical workflows which are not possible without AI support.

Therefore, in order to start harnessing the massive potential of deep learning for healthcare, and to actually use it to improve real patient outcomes, I aim to do the project that helps to bridge this gap between deep learning and clinical practice.

1.3 Users and Benefits

This deep learning is intended to validate ultra-sound images in order to determine whether they are normal, benign, or malignant. These would benefit those who work in medical fields as well as inpatients who are awaiting a cancer diagnosis.

- Inpatients that undertake the process of diagnosis the cancer. The benefits to inpatients include: the ability to determine and evaluated cancer for their ultra-sound scan images with greater precision and in less time.
- Doctors' workload should be reduced. The ability of deep learning to determine whether or not it is cancer with precision would reduce the number of causes that doctors would need to run through the test and medical process.

1.4 Typical Usage

To overcome devastating news, early diagnosis with the support of artificial intelligence methods is critical. Doctors could use a computer-aided system introduced for detecting lung cancer in a dataset collected from various sources by using a convolutional neural network technique for assisting with the diagnosis of the patient's cases: normal, benign, or malignant, through this project. The optimum model has a high accuracy of more than 90 percent. Other performance indicators, such as sensitivity and specificity, have high values. CNNs are used to detect and categorize lung cancer CT scans of patients in order to analyze data using a grid pattern like images. Doctors and medical personnel should be aware that a Convolutional Neural Network is a Deep Learning algorithm that can take in an image as input and assign potentially learnable weights and biases to various objects within the image, allowing it to distinguish one from the other. The CNN plays a vital role in compressing images to a format that is easier to interpret while retaining properties that are necessary for accurate prediction.

1.5 Main Challenges

The model design phase of this project presents the biggest challenge. A model that we intend to develop will be based on the CNN algorithm, which is used to classify images and train the model to determine whether or not a given image is a cancer cell. Since we had to create the model from scratch, we began by teaching it basic shapes and labeling one particular shape as cancer so that it could recognize that shape. After many trials and errors, we were able to develop a model with test data sets that had an accuracy of about 90 percent. Using the TensorFlow software and Conv2D as a hidden layer, we build a model with three hidden layers. The next progress will be to test it on the real ultra-sound images.

Chapter 2

Requirements Specification

2.1 System Description

A medical images classification system that identify different images from incisional biopsy. Depending on shape, each biopsy can be classify as normal, benign, or malignan. The CNN-based techniques, such as UNet, are utilized to train 300 ultrasound images. The ideal system would also high average accuracy with a precise sensitivity and a validity specifcity.

2.1.1 Perspective

The model including the input layer, hidden layer and output layer as shown in Figure 2.1. U-Net was created for the purpose of processing biomedical images. Because there isn't a lot of data available, this field relies heavily on data augmentation. As the desired outputs should be localized, each pixel will have a class label and a corresponding bounding box as shown in figure 2.2. Because there will be overlapping boxes, the network must run on each bounding box, making it redundant and computationally intensive. To localize larger boxes, more max-pooling layers must be used, but this is ineffective if the box is small. As a result, there is a tradeoff.

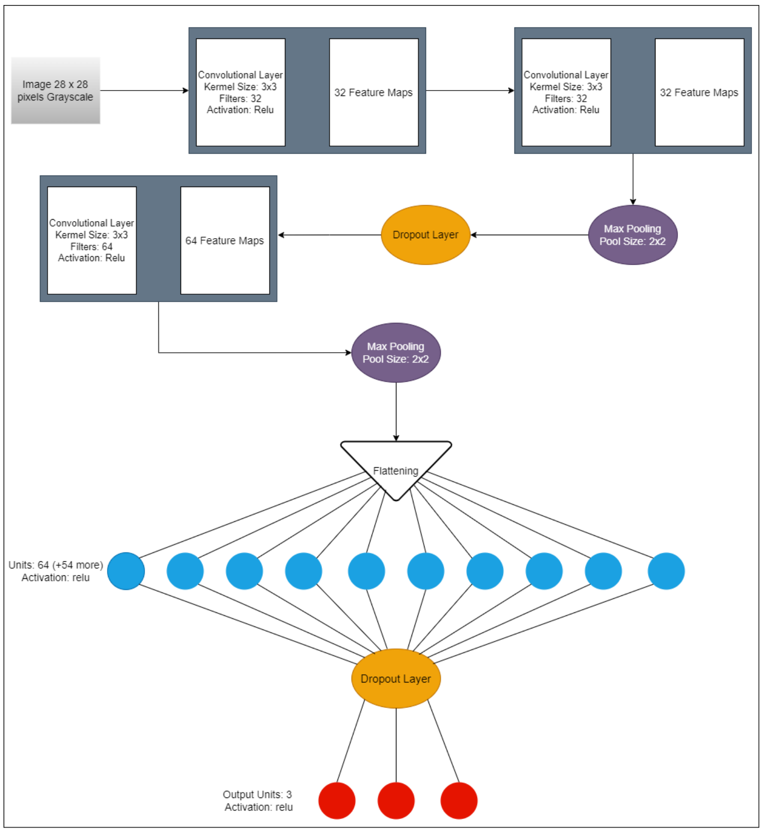


Image 28 x 28 pixels Grayscale
Convolutional Layer Kermel Size: 3x3 Filters: 32 Activation: Relu
32 Feature Maps
Convolutional Layer Kermel Size: 3x3 Filters: 32 Activation: Relu
32 Feature Maps
Max Pooling Pool Size: 2x2
Dropout Layer
Convolutional Layer Kermel Size: 3x3 Filters: 64 Activation: Relu
64 Feature Maps
Max Pooling Pool Size: 2x2
Flattening
Units: 64 (+54 more) Activation: relu
Dropout Layer
Output Units: 3 Activation: relu

Figure 2.1: System Perspective

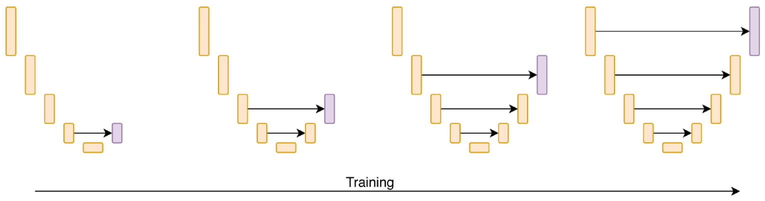


Figure 2.2: System Perspective (CNN model)

2.1.2 Functions

The functions of the CNN model are shown in Figure 2.3.

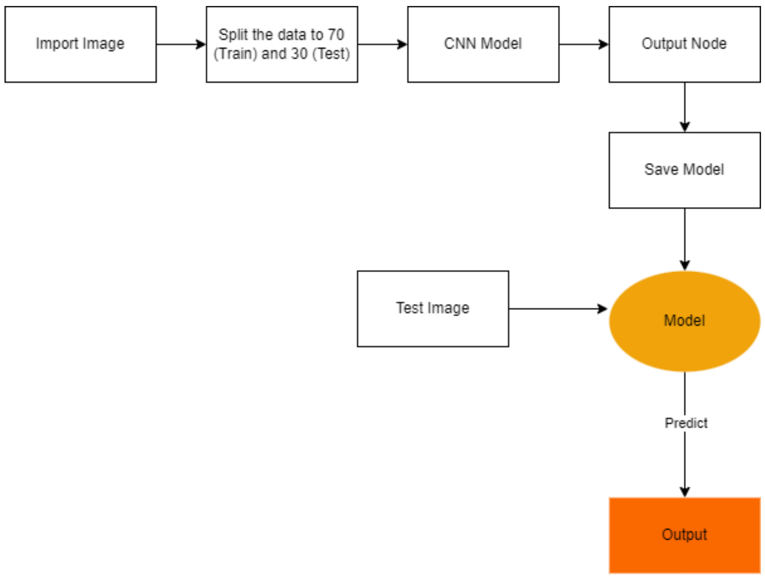


Figure 2.3: System functions (U-Net model)

2.2 Requirements

2.2.1 Model

M1 the model must be able to generate the model from the training data set.

M2 the model must be able to calculate the accuracy of the output.

M3 the model should be able to identify the input image whether the model can inform if the image is labeled as cancer or not with the previous generated model from train data set.

2.2.2 Database

D1 The database should not be allowed to interact or be directly interacted with the user in any way.

D2 The database must be able to store the output data received from the compiling model swiftly and accurately.

D3 The database must be able to separate and store each type of biopsy accordingly.

D4 The database must be able to be accessed through verified source(s)

Chapter 3

Design Specification

The interpretation of medical images such as CT and MRI requires considerable knowledge and skills because organ and lesion segmentation must always be performed layer by layer. Manual segmentation imposes a significant workload on doctors, which can introduce bias if it involves their subjective opinions. To analyze complex images, doctors must usually make a joint diagnosis, which takes time. Furthermore, automatic segmentation is our objective.

U-Net is a convolutional neural network intended for biomedical image segmentation that consists of an encoder, a bottleneck module, and a decoder. Because of its U-shaped structure combined with context information, fast training speed, and small amount of data used, the widely used U-Net meets the requirements of medical image segmentation. Figure below (figure 3.11) depicts the structure of U-Net.

Because of the large measurement size of medical images, U-Net could be used for segmentation. When large medical images are segmented and must be cut into small pieces, they cannot be uploaded to the network. Because of the network structure of U-Net, overlapping-tilling strategies are appropriate for cutting small pieces. As a result, it could take images of any size as input.

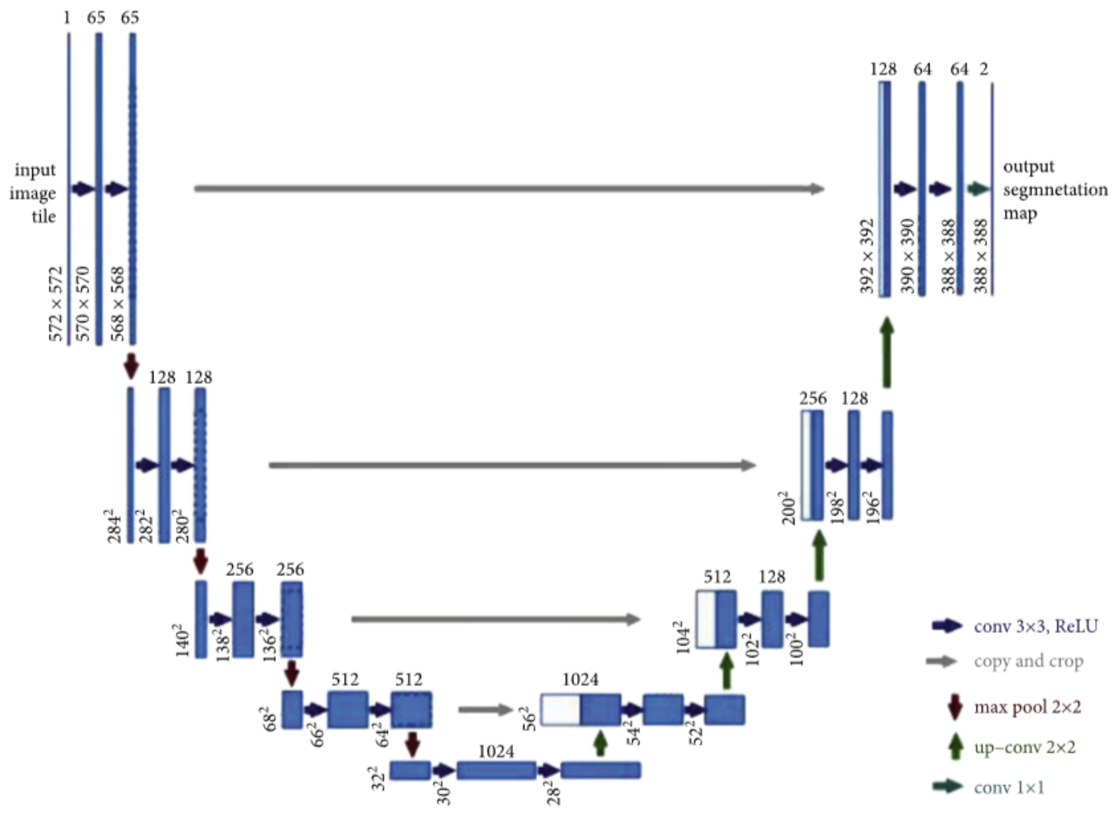


Figure 3.1: Illustration of U-Net convolution network structure

3.1 System Architecture

As you can see from figure 3.11, the left side of the U-shape is the encoding stage, also called contraction path with each layer consisting of two 3*3 convolutions with LeakyReLU activation and a 2*2 maximum pooling layer. The right side of the U-shape, also called expansion part, consists of the decoding stage and the upsampling process that is realized via 2*2 deconvolution to reduce the quantity of input channels by half

It consists of a contracting path and an expansive path. The contracting path follows the typical architecture of a convolutional network. It consists of the repeated application of two 3x3 convolutions (unpadded convolutions), each followed by a rectified linear unit (LeakyReLU) and a 2x2 max pooling operation with stride 2 for downsampling. At each downsampling step we double the number of feature channels. Every step in the expansive path consists of an upsampling of the feature map followed by a 2x2 convolution (“up-convolution”) that halves the number of feature channels, a concatenation with the correspondingly cropped feature map from the contracting path, and two 3x3 convolutions, each followed by a LeakyReLU. The cropping is necessary due to the loss of border pixels in every convolution.A 1x1 convolution is used at the final layer to map each 64-component feature vector to the desired number of classes. The network has a total of 23 convolutional layers.

figure (figure 3.11) below shows the encoder network called a contraction path and the decoder network called a expansion path.

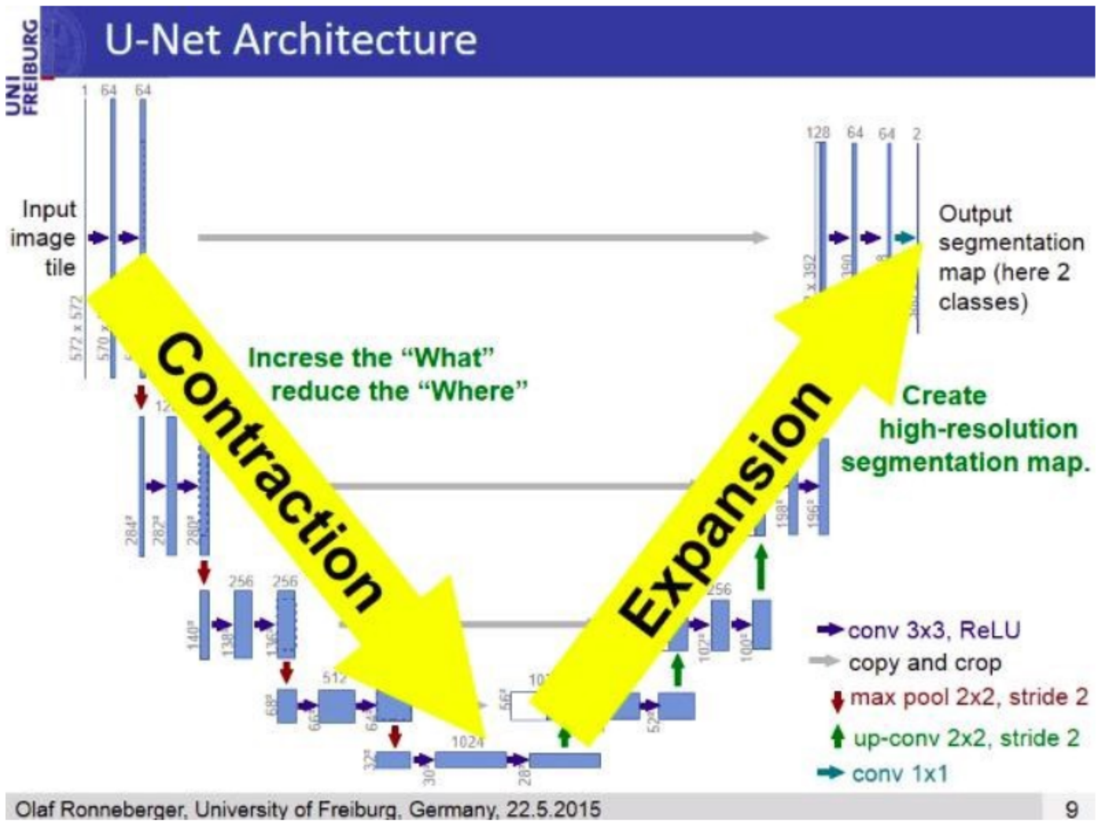

U-Net Architecture
UNI FREIBURG
Input image tile
Output segmentation map (here 2 classes)
Contraction
Expansion
Increse the "What" reduce the "Where"
Create high-resolution segmentation map.
conv 3x3, ReLU
copy and crop
max pool 2x2, stride 2
up-conv 2x2, stride 2
conv 1x1
Olaf Ronneberger, University of Freiburg, Germany, 22.5.2015
9

Figure 3.2: Illustration of U-Net convolution network structure

The contraction and expansion layers differ in that the contraction network's pooling operations are replaced by up sampling operations in the expansion network. Furthermore, the expansion network has a greater number of filters, resulting in high resolution layers. Because there are so few images in biomedical imaging, the network should be highly invariant. That is, it should concentrate on what the features are rather than where they are. U-Net is able to accomplish this due to its u-shape, which allows features to be merged together to produce high resolution results.

Starting with the first step, the 572x572x1 input image is passed through two 3x3 convolutional layers (padding='valid' and strides=1) and LeakyReLU activation, increasing the number of channels. After that, the image is down sampled in a 2x2 max-pooling layer. This process is then repeated several times until the image is 28x28x1024. Instead of being down sampled, it is now passed through a 2x2 deconvolutional layer (or transpose convolution) with strides=2. Then, as in the contraction path, it is concatenated with a cropped version of the previous feature map and sent through two 3x3 convolutional layers. The preceding process is repeated (as shown in the figure) until we obtain an image of size

388x388x64, after which a 1x1 convolution layer is applied to produce a 388x388x2 output due to the experiment's two channels.

The dimension's third column represents the number of channels, which is equivalent to the number of classes present. As we can see, this architecture is easily extensible to multiple classes.

3.2 Detailed Design

3.2.1 U-Net Model

Figure 3.3 shows the network architecture. It comprises of a contracting path (on the left) and an expanded path (right side). The contracting path adheres to the conventional convolutional network architecture. It entails applying two 3x3 convolutions (unpadded convolutions) repeatedly, each time being followed by a rectified linear unit (ReLU) and a 2x2 max pooling operation with stride 2 for downsampling. We increase the number of feature channels by two at each step in the downsampling process. Every step in the expansive path entails upsampling the feature map, followed by a 2x2 convolution ("up-convolution") that reduces the number of feature channels, a concatenation with the correspondingly cropped feature map from the contracting path, and two 3x3 convolutions, each followed by a ReLU. The loss of border pixels in each convolution necessitates cropping. A 1x1 convolution is applied as the last layer to convert each 64-component feature vector to the desired number of classes. There are 23 convolutional layers in the network as a whole.

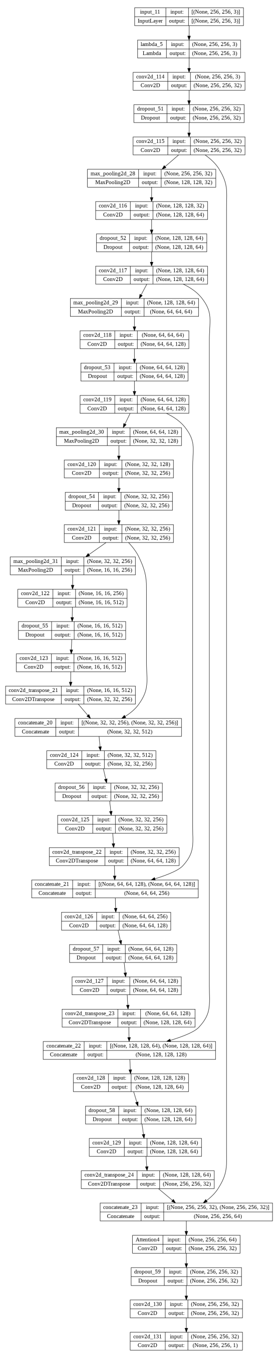

input_11 | input: | [(None, 256, 256, 3)]
InputLayer | output: | [(None, 256, 256, 3)]
lambda_5 | input: | (None, 256, 256, 3)
Lambda | output: | (None, 256, 256, 3)
conv2d_114 | input: | (None, 256, 256, 3)
Conv2D | output: | (None, 256, 256, 32)
dropout_51 | input: | (None, 256, 256, 32)
Dropout | output: | (None, 256, 256, 32)
conv2d_115 | input: | (None, 256, 256, 32)
Conv2D | output: | (None, 256, 256, 32)
max_pooling2d_28 | input: | (None, 256, 256, 32)
MaxPooling2D | output: | (None, 128, 128, 32)
conv2d_116 | input: | (None, 128, 128, 32)
Conv2D | output: | (None, 128, 128, 64)
dropout_52 | input: | (None, 128, 128, 64)
Dropout | output: | (None, 128, 128, 64)
conv2d_117 | input: | (None, 128, 128, 64)
Conv2D | output: | (None, 128, 128, 64)
max_pooling2d_29 | input: | (None, 128, 128, 64)
MaxPooling2D | output: | (None, 64, 64, 64)
conv2d_118 | input: | (None, 64, 64, 64)
Conv2D | output: | (None, 64, 64, 128)
dropout_53 | input: | (None, 64, 64, 128)
Dropout | output: | (None, 64, 64, 128)
conv2d_119 | input: | (None, 64, 64, 128)
Conv2D | output: | (None, 64, 64, 128)
max_pooling2d_30 | input: | (None, 64, 64, 128)
MaxPooling2D | output: | (None, 32, 32, 128)
conv2d_120 | input: | (None, 32, 32, 128)
Conv2D | output: | (None, 32, 32, 256)
dropout_54 | input: | (None, 32, 32, 256)
Dropout | output: | (None, 32, 32, 256)
conv2d_121 | input: | (None, 32, 32, 256)
Conv2D | output: | (None, 32, 32, 256)
max_pooling2d_31 | input: | (None, 32, 32, 256)
MaxPooling2D | output: | (None, 16, 16, 256)
conv2d_122 | input: | (None, 16, 16, 256)
Conv2D | output: | (None, 16, 16, 512)
dropout_55 | input: | (None, 16, 16, 512)
Dropout | output: | (None, 16, 16, 512)
conv2d_123 | input: | (None, 16, 16, 512)
Conv2D | output: | (None, 16, 16, 512)
conv2d_transpose_21 | input: | (None, 16, 16, 512)
Conv2DTranspose | output: | (None, 32, 32, 256)
concatenate_20 | input: | [(None, 32, 32, 256), (None, 32, 32, 256)]
Concatenate | output: | (None, 32, 32, 512)
conv2d_124 | input: | (None, 32, 32, 512)
Conv2D | output: | (None, 32, 32, 256)
dropout_56 | input: | (None, 32, 32, 256)
Dropout | output: | (None, 32, 32, 256)
conv2d_125 | input: | (None, 32, 32, 256)
Conv2D | output: | (None, 32, 32, 256)
conv2d_transpose_22 | input: | (None, 32, 32, 256)
Conv2DTranspose | output: | (None, 64, 64, 128)
concatenate_21 | input: | [(None, 64, 64, 128), (None, 64, 64, 128)]
Concatenate | output: | (None, 64, 64, 256)
conv2d_126 | input: | (None, 64, 64, 256)
Conv2D | output: | (None, 64, 64, 128)
dropout_57 | input: | (None, 64, 64, 128)
Dropout | output: | (None, 64, 64, 128)
conv2d_127 | input: | (None, 64, 64, 128)
Conv2D | output: | (None, 64, 64, 128)
conv2d_transpose_23 | input: | (None, 64, 64, 128)
Conv2DTranspose | output: | (None, 128, 128, 64)
concatenate_22 | input: | [(None, 128, 128, 64), (None, 128, 128, 64)]
Concatenate | output: | (None, 128, 128, 128)
conv2d_128 | input: | (None, 128, 128, 128)
Conv2D | output: | (None, 128, 128, 64)
dropout_58 | input: | (None, 128, 128, 64)
Dropout | output: | (None, 128, 128, 64)
conv2d_129 | input: | (None, 128, 128, 64)
Conv2D | output: | (None, 128, 128, 64)
conv2d_transpose_24 | input: | (None, 128, 128, 64)
Conv2DTranspose | output: | (None, 256, 256, 32)
concatenate_23 | input: | [(None, 256, 256, 32), (None, 256, 256, 32)]
Concatenate | output: | (None, 256, 256, 64)
Attention4 | input: | (None, 256, 256, 64)
Conv2D | output: | (None, 256, 256, 32)
dropout_59 | input: | (None, 256, 256, 32)
Dropout | output: | (None, 256, 256, 32)
conv2d_130 | input: | (None, 256, 256, 32)
Conv2D | output: | (None, 256, 256, 32)
conv2d_131 | input: | (None, 256, 256, 32)
Conv2D | output: | (None, 256, 256, 1)

Figure 3.3: Overview Structure of CNN Model

3.2.2 CNN model

The CNN model consists of an input layer, four hidden layers, and an output layer. The model is summarised as follows:

- Input layer: The size of the partial images determines how big the filters are to start, and the step length determines how many pixels we proceed after each computation. Which turn out to be 256*256 pixels will be input into the model and through the BatchNormalization, Dropout (0.2) and LeakyReLU as activation function with the MaxPooling2D to preserves small features in a few pixels that are crucial for the task solution.
- Hidden layer: Consists of 4 comparable layers of repeating functions, including MaxPooling2D, Conv2D, BatchNormalization, Drop out, and LeakyReLU (activation function). Each available filter function lowered the image's dimensionality and filtered the image to preserve the image's quality so that the study's findings could be seen.
- Output layer: This is where the result is begin produced for future use.

Figure 3.4 shows the CNN architecture.

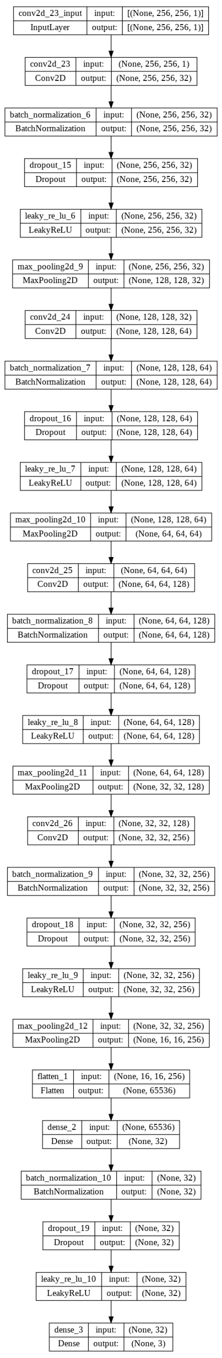

conv2d_23_input | input: | [(None, 256, 256, 1)]
InputLayer | output: | [(None, 256, 256, 1)]
conv2d_23 | input: | (None, 256, 256, 1)
Conv2D | output: | (None, 256, 256, 32)
batch_normalization_6 | input: | (None, 256, 256, 32)
BatchNormalization | output: | (None, 256, 256, 32)
dropout_15 | input: | (None, 256, 256, 32)
Dropout | output: | (None, 256, 256, 32)
leaky_re_lu_6 | input: | (None, 256, 256, 32)
LeakyReLU | output: | (None, 256, 256, 32)
max_pooling2d_9 | input: | (None, 256, 256, 32)
MaxPooling2D | output: | (None, 128, 128, 32)
conv2d_24 | input: | (None, 128, 128, 32)
Conv2D | output: | (None, 128, 128, 64)
batch_normalization_7 | input: | (None, 128, 128, 64)
BatchNormalization | output: | (None, 128, 128, 64)
dropout_16 | input: | (None, 128, 128, 64)
Dropout | output: | (None, 128, 128, 64)
leaky_re_lu_7 | input: | (None, 128, 128, 64)
LeakyReLU | output: | (None, 128, 128, 64)
max_pooling2d_10 | input: | (None, 128, 128, 64)
MaxPooling2D | output: | (None, 64, 64, 64)
conv2d_25 | input: | (None, 64, 64, 64)
Conv2D | output: | (None, 64, 64, 128)
batch_normalization_8 | input: | (None, 64, 64, 128)
BatchNormalization | output: | (None, 64, 64, 128)
dropout_17 | input: | (None, 64, 64, 128)
Dropout | output: | (None, 64, 64, 128)
leaky_re_lu_8 | input: | (None, 64, 64, 128)
LeakyReLU | output: | (None, 64, 64, 128)
max_pooling2d_11 | input: | (None, 64, 64, 128)
MaxPooling2D | output: | (None, 32, 32, 128)
conv2d_26 | input: | (None, 32, 32, 128)
Conv2D | output: | (None, 32, 32, 256)
batch_normalization_9 | input: | (None, 32, 32, 256)
BatchNormalization | output: | (None, 32, 32, 256)
dropout_18 | input: | (None, 32, 32, 256)
Dropout | output: | (None, 32, 32, 256)
leaky_re_lu_9 | input: | (None, 32, 32, 256)
LeakyReLU | output: | (None, 32, 32, 256)
max_pooling2d_12 | input: | (None, 32, 32, 256)
MaxPooling2D | output: | (None, 16, 16, 256)
flatten_1 | input: | (None, 16, 16, 256)
Flatten | output: | (None, 65536)
dense_2 | input: | (None, 65536)
Dense | output: | (None, 32)
batch_normalization_10 | input: | (None, 32)
BatchNormalization | output: | (None, 32)
dropout_19 | input: | (None, 32)
Dropout | output: | (None, 32)
leaky_re_lu_10 | input: | (None, 32)
LeakyReLU | output: | (None, 32)
dense_3 | input: | (None, 32)
Dense | output: | (None, 3)

Figure 3.4: Overview Structure of CNN Model

3.3 Feature List

List and status of project's features:

1. Model Design: Develop a CNN algorithm to diagnose and predict whether it is normal, malignant and benign.

   Feature status: [X]Complete [ ]Partial [ ]Initial

2. Model Training: Fit the training data into the developed CNN algorithm.

   Feature status: [X]Complete [ ]Partial [ ]Initial

3. Model Validation: Evaluate the accuracy of the CNN algorithm with testing data.

   Feature status: [X]Complete [ ]Partial [ ]Initial

4. Display Prediction Result: Present the predictive result on the program.

   Feature status: [X]Complete [ ]Partial [ ]Initial

3.4 Test Results

We have trained the data to categorize the most accurate cancer diagnosis for the test result. To more clearly characterize the output before training in Model2, we quantify the data as loss, accuracy, and IOU in Model 1. For Model 1, we obtained the following values for the loss, accuracy, and IOU: 0.1092, 0.9593, and 0.4627, respectively. The orange line represents the validation value, and the blue represents the training value. Figure 3.5

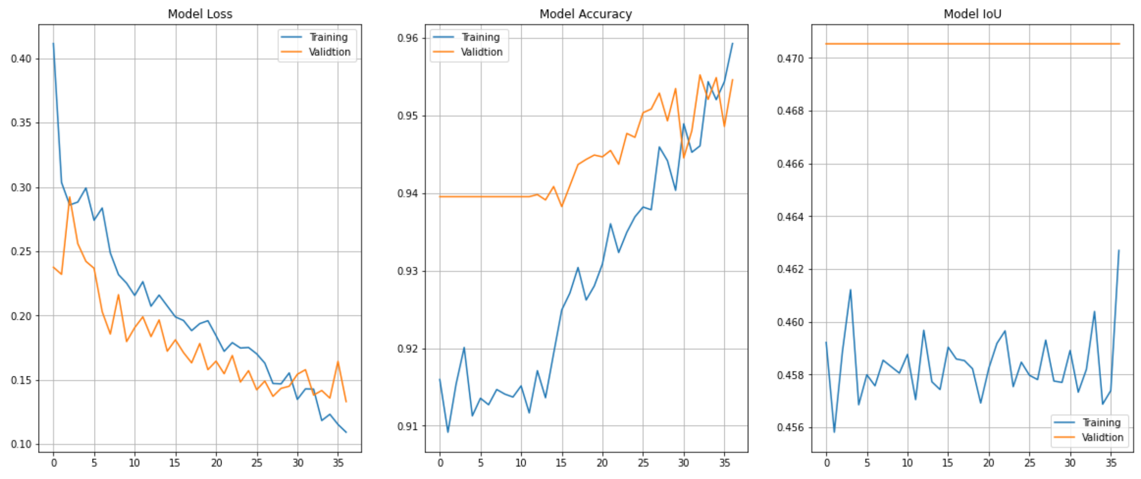


Figure 3.5: The graph of Model 1 result

The output has a more distinct shape called as Processed Mask once we trained Model 1. Figure3.6

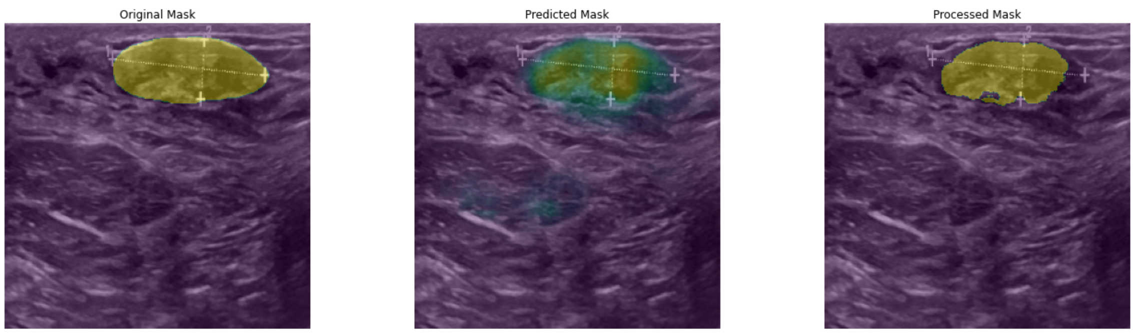


Figure 3.6: The result of Model 1

Then, we train Model 2 using the Model 1 results that contain obvious images. We obtained the loss value = 0.2915 and accuracy value = 0.9840 after training the data for 400 Epochs.

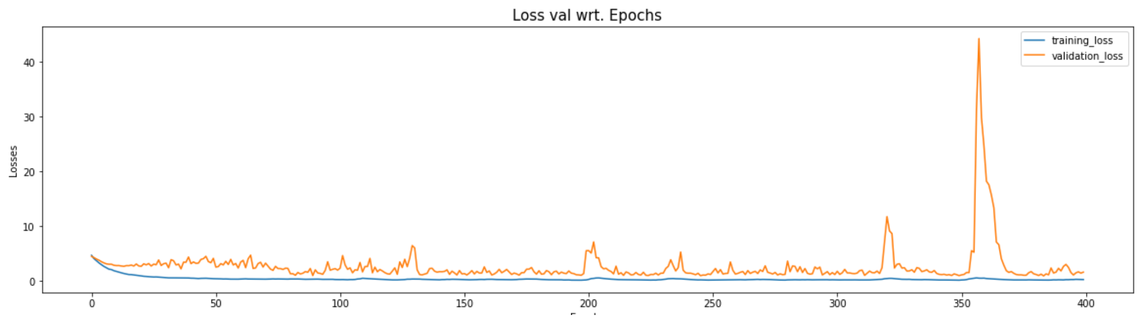


Figure 3.7: The graph of Model 2 result

For Figure3.7 , The graph demonstrates that the model 2 result has marginally increased and reached its peak at 350. As a result, with more epochs, the training data become over-fit. This is the result of model 2 after trained the data in Figure 3.8

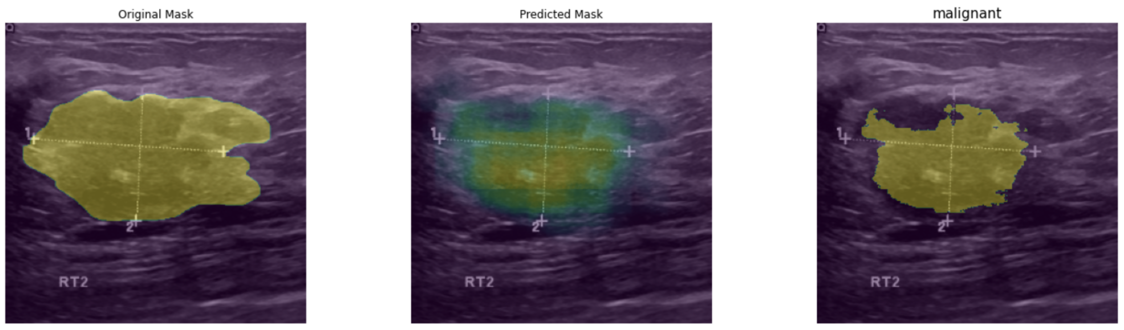
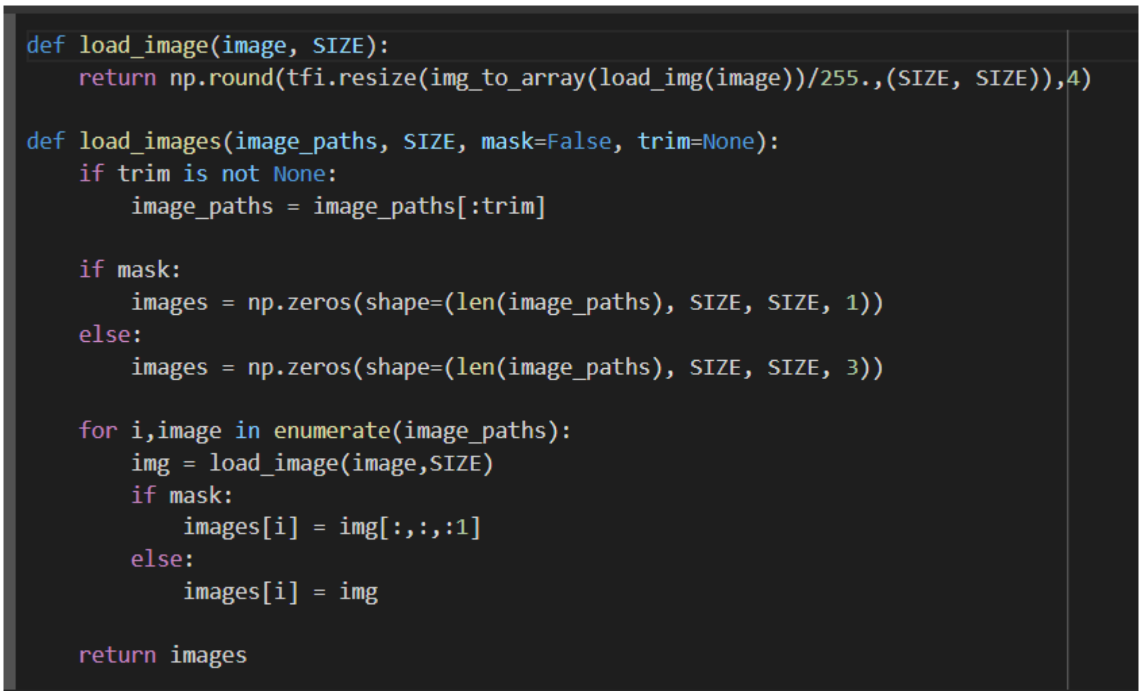


Figure 3.8: The result of Model 2

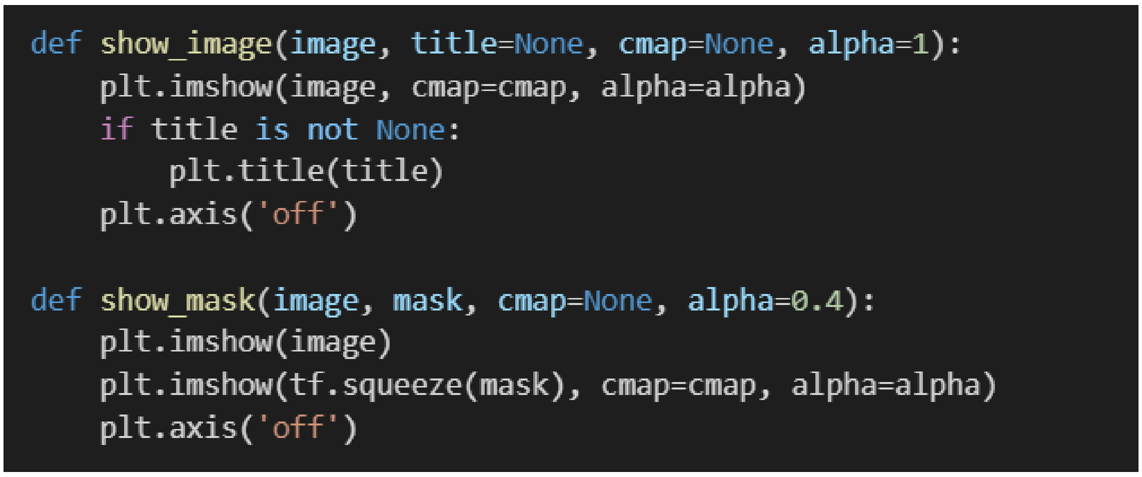
3.5 Demonstration of Mockup system

The image will be load by function show in figure 3.9 and can be display by the figure 3.10 functions.



```python
def load_image(image, SIZE):
    return np.round(tfi.resize(img_to_array(load_img(image))/255.,(SIZE, SIZE)),4)

def load_images(image_paths, SIZE, mask=False, trim=None):
    if trim is not None:
        image_paths = image_paths[:trim]

    if mask:
        images = np.zeros(shape=(len(image_paths), SIZE, SIZE, 1))
    else:
        images = np.zeros(shape=(len(image_paths), SIZE, SIZE, 3))

    for i,image in enumerate(image_paths):
        img = load_image(image,SIZE)
        if mask:
            images[i] = img[:,:,:1]
        else:
            images[i] = img

    return images
```

Figure 3.9: Load image functions



```python
def show_image(image, title=None, cmap=None, alpha=1):
    plt.imshow(image, cmap=cmap, alpha=alpha)
    if title is not None:
        plt.title(title)
    plt.axis('off')

def show_mask(image, mask, cmap=None, alpha=0.4):
    plt.imshow(image)
    plt.imshow(tf.squeeze(mask), cmap=cmap, alpha=alpha)
    plt.axis('off')
```

Figure 3.10: Display image functions

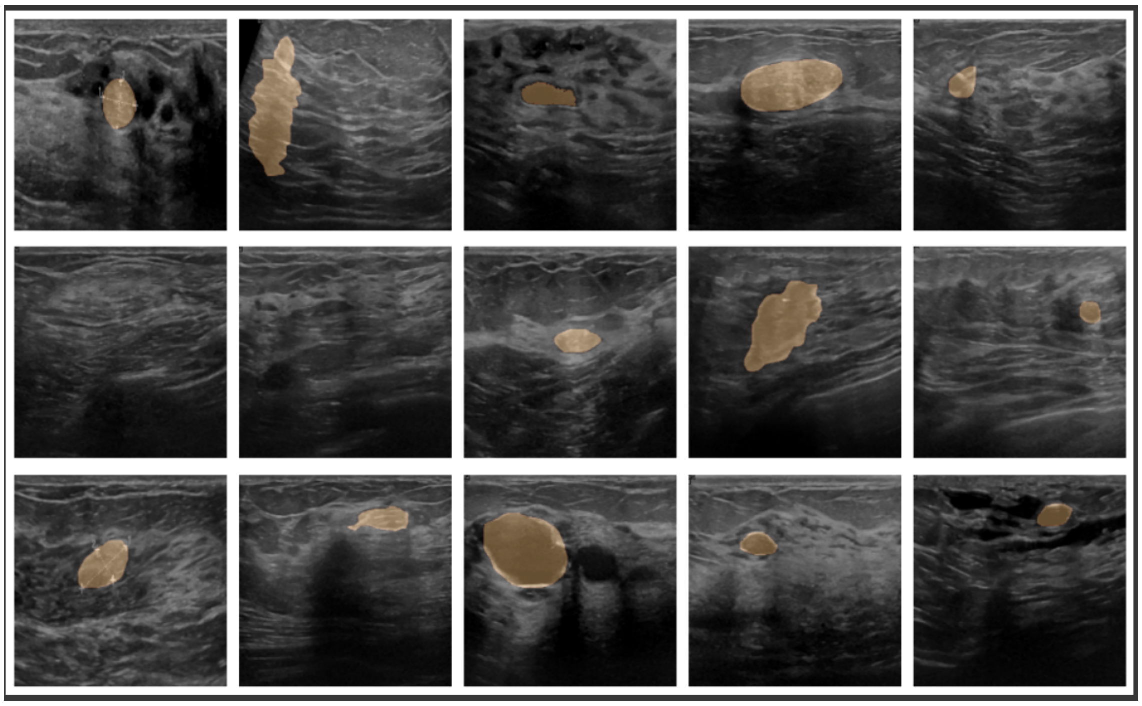
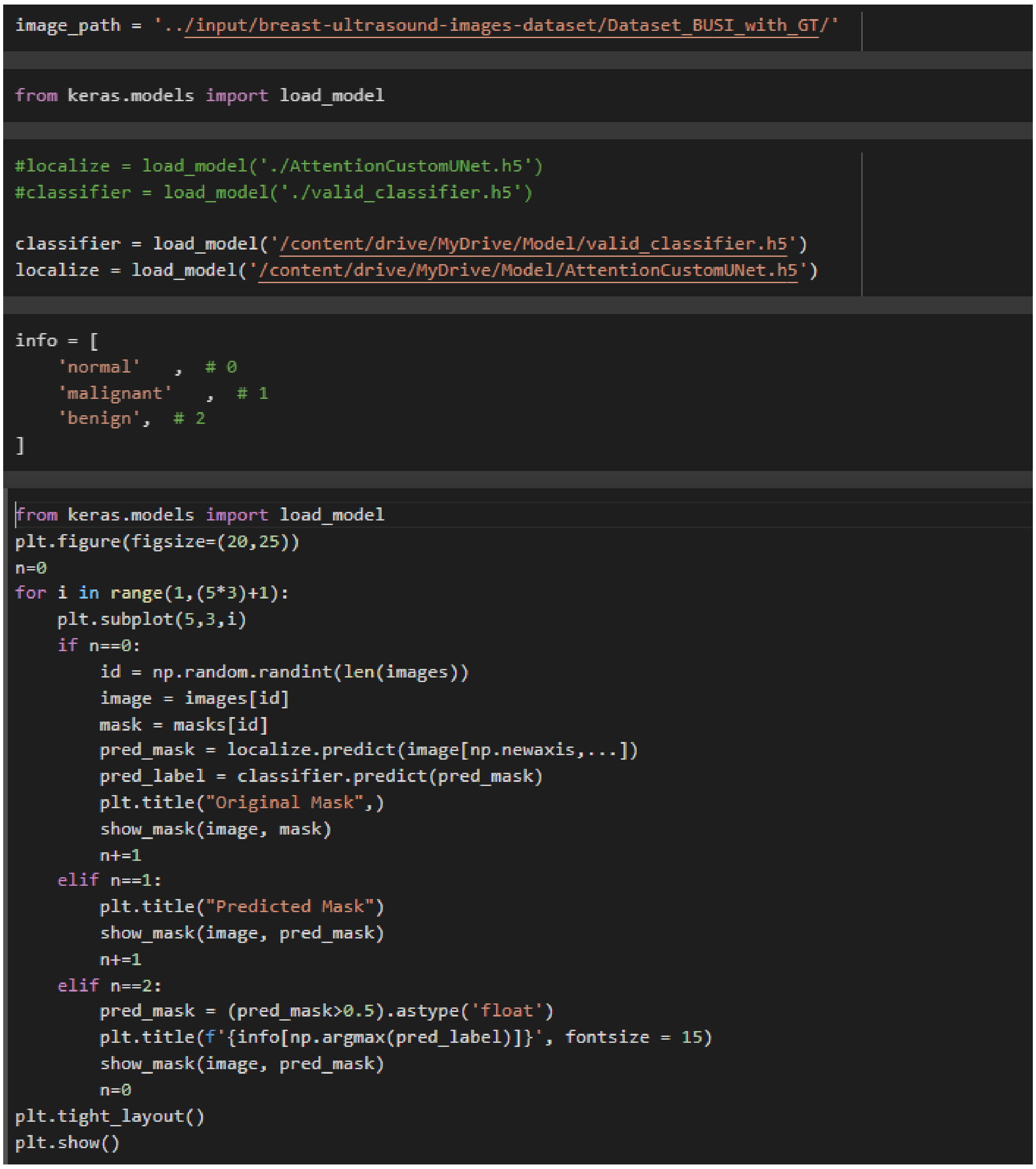


Figure 3.11: Example of images with pre-processing

The images will be passed through U-Net: Convolutional Networks (figure 3.3) and CNN model (figure 3.4). After passing through U-Net and CNN model, the output will be compared in the last function 3.12.



```
image_path = '../input/breast-ultrasound-images-dataset/Dataset_BUSI_with_GT/'
```

```
from keras.models import load_model
```

```
#localize = load_model('./AttentionCustomUNet.h5')
#classifier = load_model('./valid_classifier.h5')

classifier = load_model('/content/drive/MyDrive/Model/valid_classifier.h5')
localize = load_model('/content/drive/MyDrive/Model/AttentionCustomUNet.h5')
```

```
info = [
    'normal'   ,  # 0
    'malignant'   ,  # 1
    'benign',  # 2
]
```

```
from keras.models import load_model
plt.figure(figsize=(20,25))
n=0
for i in range(1,(5*3)+1):
    plt.subplot(5,3,i)
    if n==0:
        id = np.random.randint(len(images))
        image = images[id]
        mask = masks[id]
        pred_mask = localize.predict(image[np.newaxis,...])
        pred_label = classifier.predict(pred_mask)
        plt.title("Original Mask",)
        show_mask(image, mask)
        n+=1
    elif n==1:
        plt.title("Predicted Mask")
        show_mask(image, pred_mask)
        n+=1
    elif n==2:
        pred_mask = (pred_mask>0.5).astype('float')
        plt.title(f'{info[np.argmax(pred_label)]}', fontsize = 15)
        show_mask(image, pred_mask)
        n=0
plt.tight_layout()
plt.show()
```

Figure 3.12: Comparison function

For fully detailed code, please navigate using the link provided below.
https://colab.research.google.com/drive/1q9Ldpzs8czWuJf42Hc6HetvzIjkvT1U5?usp=sharing