PROJECT REPORT

Submitted by -
Member 1 : Ankur Kumar Jaiswal(21MM62R08)
Member 2: Ashwani Kumar Singh(21MM62R06)

Title of Project

Extracting feature, Reducing Speckle Noise, Segmenting and Predicting Breast Cancer Abnormality from BUS(Breast UltraSound) Images.

Team Members

In total, we have performed 3 tasks in our project. The tasks performed are as follows:
Task -1 : Speckle Noise Reduction using Autoencoder - This part of the project is performed by Ashwini Kumar Singh.
Task - 2 : Segmentation of abnormality(benign or malignant tumors) from breast cancer ultrasound images **-** This part of the project is performed by Ankur Kumar Jaiswal.
Task - 3 : Applying Image preprocessing techniques on breast cancer ultrasound images and extracting features from it and applying these features to machine learning algorithms**.** This part of the project is performed by both Ankur Kumar Jaiswal and Ashwini Kumar Singh.

Introduction

Imaging is one of the key medical diagnostic tools to observe internal organs and soft tissues. One such tool is ultrasound scanning, which is highly utilized because of a number of advantages: non-invasive, free of radiation, economical and real-time. Selection of Region of Interest (ROI) of the ultrasound image (Image segmentation) and image compression are two most important steps in the above mentioned process. However, automatic selection of the ROI is still a challenge due to speckle noise present in ultrasound images which affects the image quality negatively. Therefore, it is important to pre-process the ultrasound images prior to image segmentation and image compression.

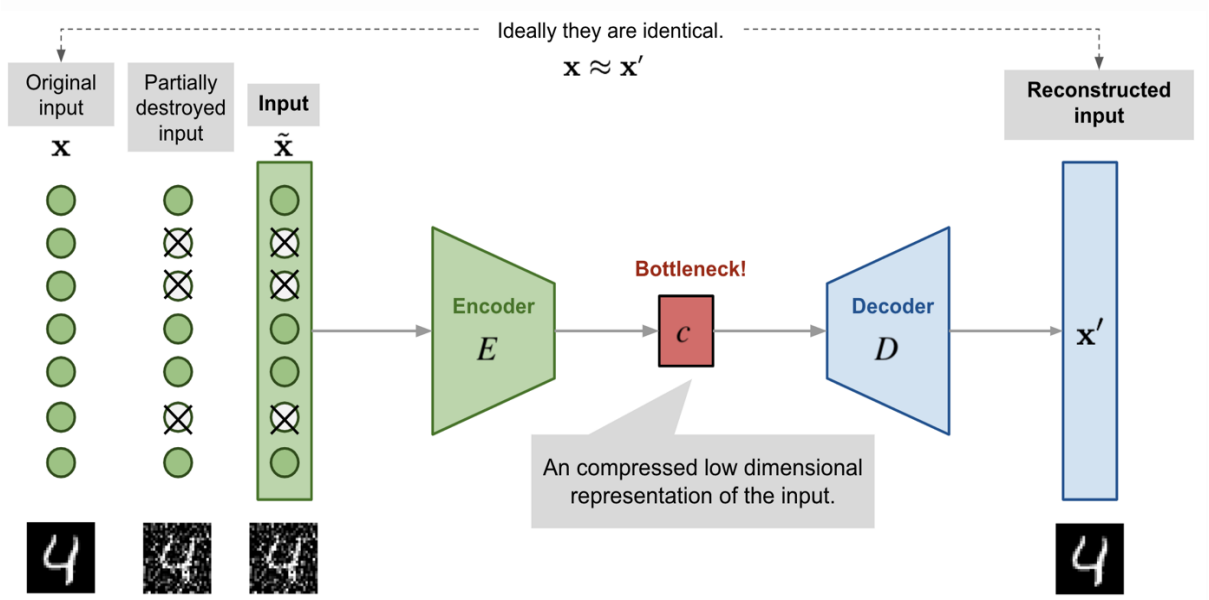
Objective

Objective - 1 :- Speckle noise reduction of breast cancer images using autoencoder. The purpose is to help doctors/clinician to increase the quality of the images for better view of biomedical images. So, it reduces the risk of the doctor to falsely interpret the biomedical images.
Objective - 2 :- Implementation of the standard U-Net model for breast cancer ultrasound image segmentation. The motive to use the U-Net model is to localize the area of interest or to find the position or to segment out the cancer growth portion from the BUS Images.
Objective - 3 :- To apply preprocessing techniques like Image Enhancement, Edge Detection,erosion, dilation etc on the BUS Images. Using these preprocessing results as a feature value and applying machine learning on these features as a dataset.

Methodology

Task -1 : Removing speckle noise from BUS Images using convolutional denoising autoencoder network, which is one of the deep learning methods. The results obtained using these methods are being compared with other classical methods and were found to be more successful than other classical methods.

The use of ultrasound imaging in medical diagnosis is well established due to its non-invasiveness, low cost, capability of forming real time imaging, and continuing improved image quality. However, in taking ultrasound images there is still noise that causes the decrease of the ultrasound image quality, one of the most disturbing ones is speckle noise. Speckle noise has long been known as a factor that limits the detection of ultrasound images, especially in low contrast images. Speckle noise is a multiplicative noise that degrades the visual quality of ultrasound images and affects the clinical assessment. The results show that the under-proper hyperparameter optimization implemented by the autoencoder performs significantly better than other traditional filters used for denoising. The diagram of the autoencoder is shown below:



Mathematical Equations used :

We choose to train our model with the **Mean Square Error,** then we aim at minimizing

$$\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - D(E(x^{(i)}))\|^2$$

with $E(x) = f(Wx) = c$ and $D(c) = g(Vc)$, W, V respectively the weights of the encoder and the decoder and f, g their activation functions

In the case where *f* and *g* are linears, the loss function becomes

$$\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - VWx^{(i)}\|^2$$

Image Preprocessing :

- Convert the input images and their masks into grayscale images.
- Resize the input images and their mask into 128x128.
- Some input images having two masks, concatenate them to one mask.

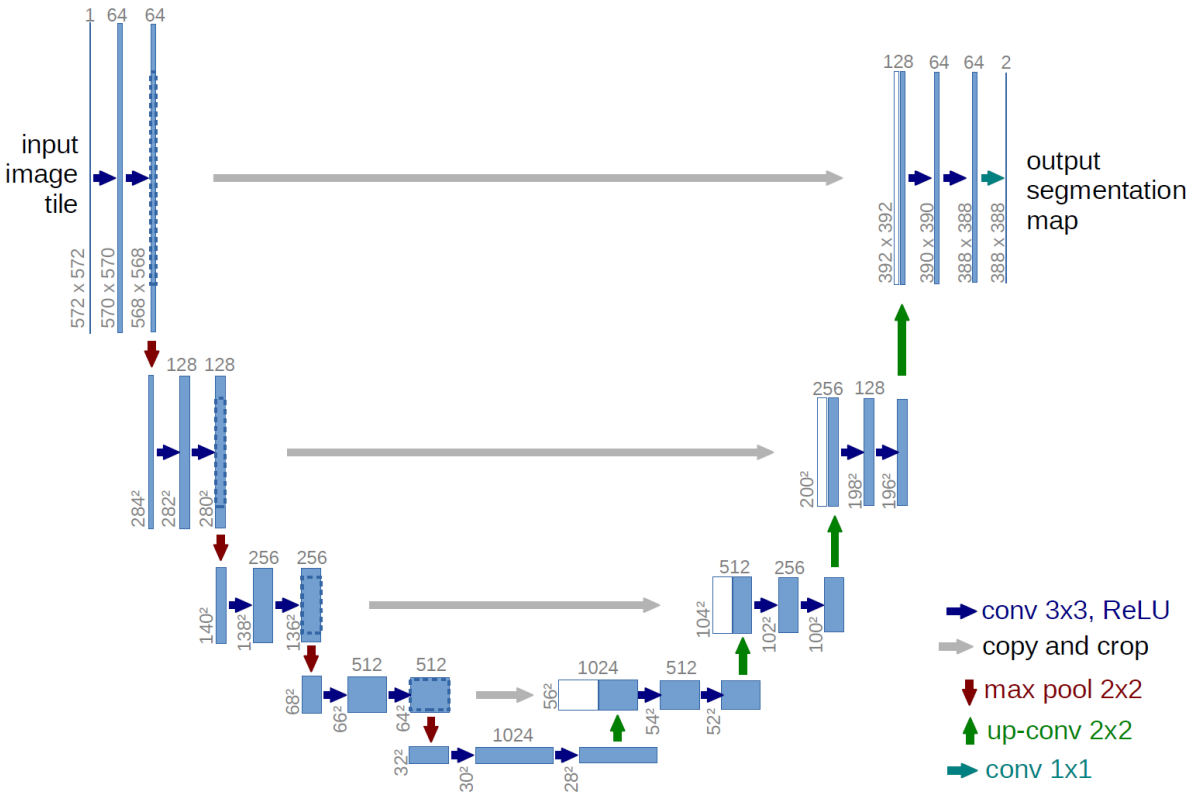
Parameters used :

The parameters used in the network in the study;

- Image dimensions: 128x128
- Optimizer: Adam
- Number of revolutions (Epoch): 200
- Number of Batches: 64
- Kernel Size - 3x3
- Max Pooling - 2x2
- Learning Rate : 1e-10
- Error function: Mean Square Error
- Number of training data: 624
- Number of test data: 156
- Monitoring : Validation Loss
- Activation Function : ReLU

Task-2 : Segmentation of BUS Images using U-net Model

The U-Net Model is mainly used for semantic segmentation of biomedical images. It is basically a convolutional neural network consisting of contracting and expansive paths. 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 (ReLU) 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 ReLU. The cropping is necessary due to the loss of border pixels in every convolution. At the final layer, a 1x1 convolution is used to map each 64-component feature vector to the desired number of classes. In total the network has 23 convolutional layers. The architecture of U-Net is shown below :



U-Net Architecture

Mathematical Equations used :

In convolution, the formula used is -

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

n_{in}: number of input features
n_{out}: number of output features
k: convolution kernel size
p: convolution padding size
s: convolution stride size

In Max Pooling, the formula used is -

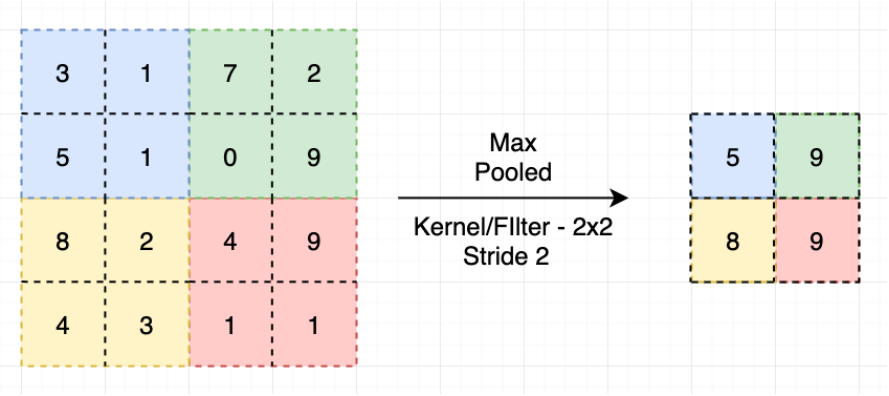


Image PreProcessing:

- Resize the input image into 128x128.
- Convert the input RGB image to grayscale image.
- Adding speckle noise to the input images

Parameters used :

The parameters used in the network in the study;

- Image dimensions: 128x128
- Optimizer: Adam
- Number of revolutions (Epoch): 200
- Number of Batches: 64
- Kernel Size - 3x3
- Max Pooling - 2x2
- Learning Rate : 0.00005
- Error function: Mean Square Error
- Number of training data: 624
- Number of test data: 156
- Monitoring : Validation Loss and Validation Accuracy
- Activation Function : ReLU

Task-3 : Image preprocessing and classification of BUS Images using Machine Learning Algorithms. In the image processing part, we have preprocessed the using image enhancement techniques like histogram equalization, otsu's binarization, adaptive histogram equalization, contrast limited adaptive histogram equalization etc. We have used edge detection filters like canny, sobel and laplacian etc in image preprocessing. We have used morphological operations like erosion and dilation on our input images. We have also calculated GLCM features like entropy, energy, homogeneity, correlation, contrast etc. After doing all these preprocessing, we have

calculated the statistical features like mean and median etc. of all images to use as features. Then, we have applied these features to machine learning algorithms like SVM, Random Forest, KNN, Logistics regression, Decision Tree and naive bias etc.

Mathematical Equations used :

Logistics Regression

$$y= b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

$$\frac{y}{1-y}$$ **; 0 for y= 0, and infinity for y=1**

$$log\left[\frac{y}{1-y}\right] = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

KNN Classifier

d=√((x2-x1)²+(y2-y1)²) to find the distance between any two points.

SVM Classifier

$$||w||_2 =: \sqrt{w_1^2 + w_2^2 + w_3^2 + \ldots w_n^2}$$

$$d_H(\phi(x_0)) = \frac{|w^T(\phi(x_0)) + b|}{||w||_2}$$

$$y_n[w^T\phi(x) + b] = \begin{cases} \geq 0 \; if \; correct \\ < 0 \; if \; incorrect \end{cases}$$

$$w^* = arg_w max\left[min_n \frac{|w^T(\phi(x_n)) + b|}{||w||_2}\right] = arg_w max\left[min_n \frac{y_n|w^T(\phi(x_n)) + b|}{||w||_2}\right] \; \because \; perfect \; seperation$$

Decision Tree Classifier

$$Entropy = -\sum_{i=1}^{n} p_i * \log(p_i)$$

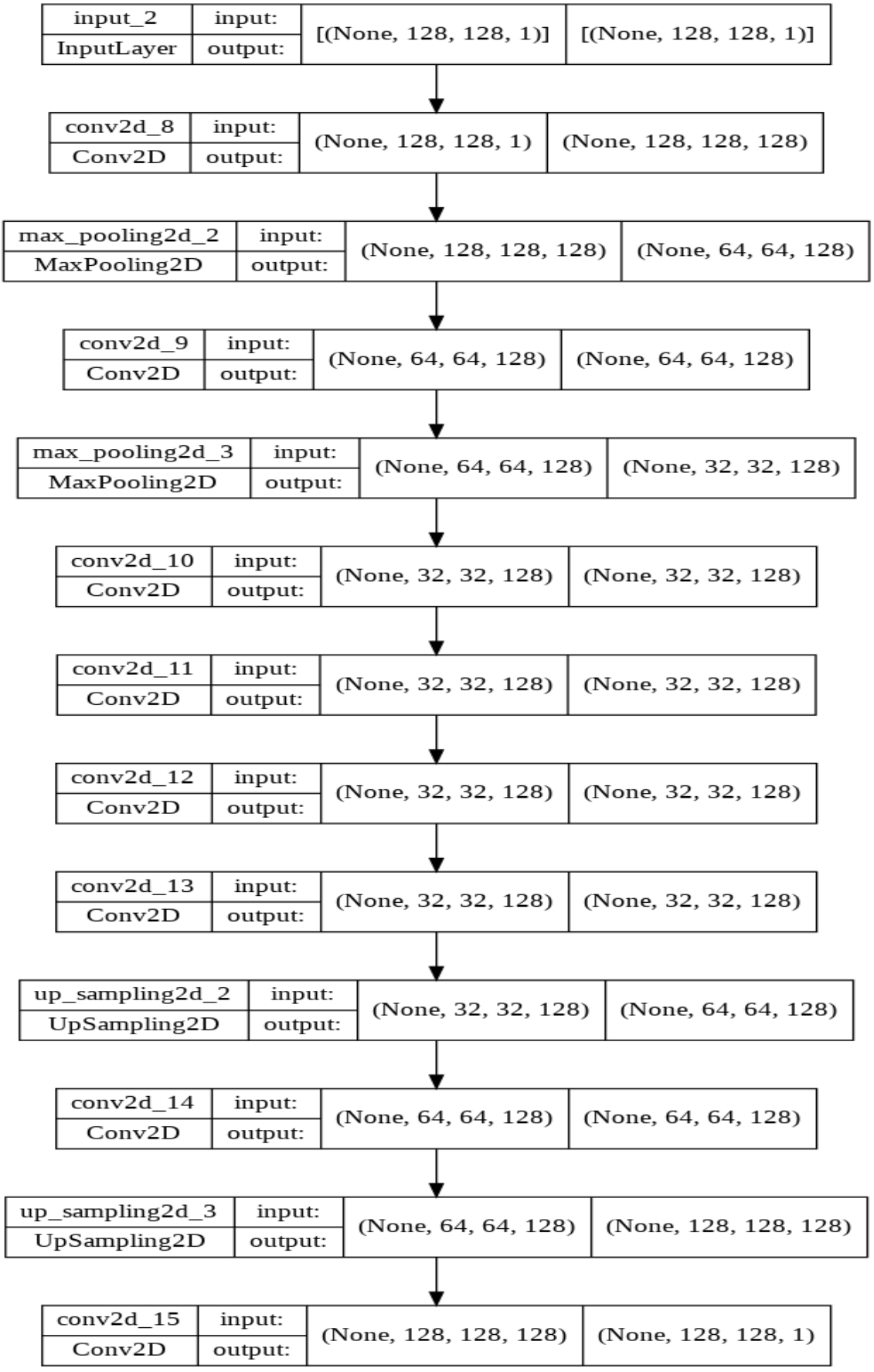
$$Gini\,index = 1 - \sum_{i=1}^{n} p_i^2$$

$$\text{Info}_A(D) = \sum_{j=1}^{V} \frac{|Dj|}{|D|} \quad \text{X Info}(D_j)$$

Detailed Algorithm:

Task -1 :

First of all, the noisy image is given as input to the network. Then, Extract features by compressing the image in the encoder part of the network. After the operation is performed, in the decoding part. The compressed image is restored to its original size. The structure of the the convolutional denoising autoencoder network is shown below :

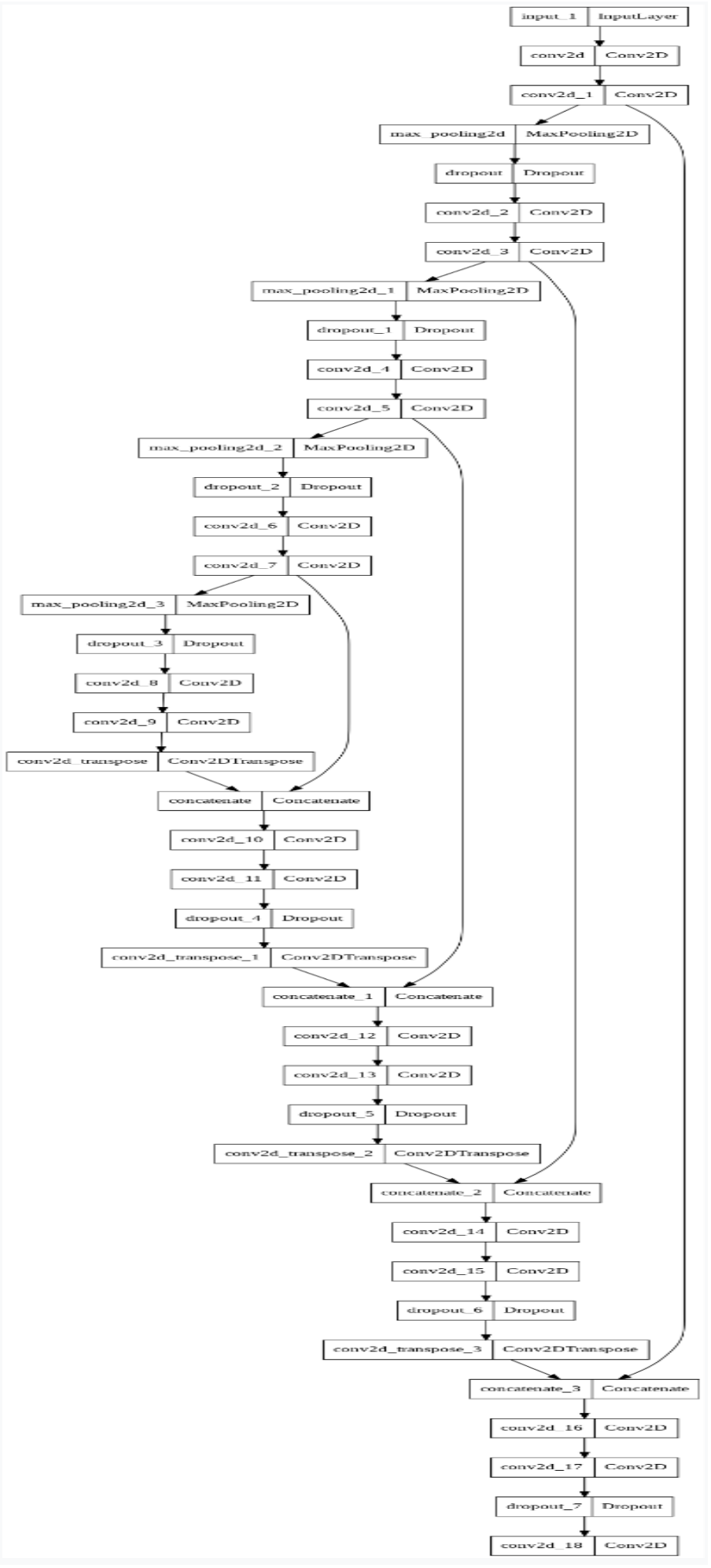

input_2 | InputLayer | input: | output: | [(None, 128, 128, 1)] | [(None, 128, 128, 1)]
conv2d_8 | Conv2D | input: | output: | (None, 128, 128, 1) | (None, 128, 128, 128)
max_pooling2d_2 | MaxPooling2D | input: | output: | (None, 128, 128, 128) | (None, 64, 64, 128)
conv2d_9 | Conv2D | input: | output: | (None, 64, 64, 128) | (None, 64, 64, 128)
max_pooling2d_3 | MaxPooling2D | input: | output: | (None, 64, 64, 128) | (None, 32, 32, 128)
conv2d_10 | Conv2D | input: | output: | (None, 32, 32, 128) | (None, 32, 32, 128)
conv2d_11 | Conv2D | input: | output: | (None, 32, 32, 128) | (None, 32, 32, 128)
conv2d_12 | Conv2D | input: | output: | (None, 32, 32, 128) | (None, 32, 32, 128)
conv2d_13 | Conv2D | input: | output: | (None, 32, 32, 128) | (None, 32, 32, 128)
up_sampling2d_2 | UpSampling2D | input: | output: | (None, 32, 32, 128) | (None, 64, 64, 128)
conv2d_14 | Conv2D | input: | output: | (None, 64, 64, 128) | (None, 64, 64, 128)
up_sampling2d_3 | UpSampling2D | input: | output: | (None, 64, 64, 128) | (None, 128, 128, 128)
conv2d_15 | Conv2D | input: | output: | (None, 128, 128, 128) | (None, 128, 128, 1)

Let’s explain the training process at stake here.

- An input is sampled from our dataset.
- A corrupted version of this input is sampled from a stochastic mapping $M(\tilde{x}|x)$
- $(x,\tilde{x})$ is used as a training example.

During training, the aim is to minimize the negative log-likelihood cost function.This optimization leads to minimizing the distance between the corrupted input and the black manifold which characterizes our inputs.

Task 2 - The architecture is symmetric and consists of two major parts — the left part is called contracting path, which is constituted by the general convolutional process; the right part is expansive path, which is constituted by transposed 2d convolutional layers. The structure of the the U-Net convolutional network is shown below :


input_1 InputLayer
conv2d Conv2D
conv2d_1 Conv2D
max_pooling2d MaxPooling2D
dropout Dropout
conv2d_2 Conv2D
conv2d_3 Conv2D
max_pooling2d_1 MaxPooling2D
dropout_1 Dropout
conv2d_4 Conv2D
conv2d_5 Conv2D
max_pooling2d_2 MaxPooling2D
dropout_2 Dropout
conv2d_6 Conv2D
conv2d_7 Conv2D
max_pooling2d_3 MaxPooling2D
dropout_3 Dropout
conv2d_8 Conv2D
conv2d_9 Conv2D
conv2d_transpose Conv2DTranspose
concatenate Concatenate
conv2d_10 Conv2D
conv2d_11 Conv2D
dropout_4 Dropout
conv2d_transpose_1 Conv2DTranspose
concatenate_1 Concatenate
conv2d_12 Conv2D
conv2d_13 Conv2D
dropout_5 Dropout
conv2d_transpose_2 Conv2DTranspose
concatenate_2 Concatenate
conv2d_14 Conv2D
conv2d_15 Conv2D
dropout_6 Dropout
conv2d_transpose_3 Conv2DTranspose
concatenate_3 Concatenate
conv2d_16 Conv2D
conv2d_17 Conv2D
dropout_7 Dropout
conv2d_18 Conv2D

U-Net Algorithm steps :

- Pass the grayscale input image to the Encoder first. The operations like convolution, max pooling, dropout will be performed on the grayscale image as mentioned in the U-Net architecture.
- The output of the encoder will be the compressed image called latent representation.
- Now, the latent code is given as an input to the decoder. The operations like upsampling, concatenation, convolution and dropout will be performed in the decoder.
- We will get the segmented image which is similar to the grayscale mask image of the respective input image.

Task-3 :

KNN Classifier

1 − For implementing any algorithm, we need a dataset. So during the first step of KNN, we must load the training as well as test data.

2 − Next, we need to choose the value of K i.e. the nearest data points. K can be any integer.

3 − For each point in the test data do the following −

3.1 − Calculate the distance between test data and each row of training data with the help of any of the methods, namely: Euclidean, Manhattan or Hamming distance. The most commonly used method to calculate distance is Euclidean.

3.2 − Now, based on the distance value, sort them in ascending order.

3.3 − Next, it will choose the top K rows from the sorted array.

3.4 − Now, it will assign a class to the test point based on the most frequent class of these rows.

Decision Tree Classifier

- Step-1: Begin the tree with the root node, says S, which contains the complete dataset.
- Step-2: Find the best attribute in the dataset using Attribute Selection Measure (ASM).
- Step-3: Divide the S into subsets that contain possible values for the best attributes.
- Step-4: Generate the decision tree node, which contains the best attribute.

- Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and call the final node as a leaf node.

SVM Classifier

Step 1: Load the important libraries.
Step 2: Import dataset and extract the X variables and Y separately.
Step 3: Divide the dataset into train and test.
Step 4: Initializing the SVM classifier model.
Step 5: Fitting the SVM classifier model.
Step 6: Coming up with predictions.

Logistics Regression classifier

- Step-1 : The logistic regression model takes real-valued inputs and makes a prediction as to the probability of the input belonging to the default class (class 0).
- Step-2 : If the probability is > 0.5 we can take the output as a prediction for the default class (class 0), otherwise the prediction is for the other class (class 1).
- Step -3 : Calculate a prediction using the current values of the coefficients.
- Step-4 : Calculate new coefficient values based on the error in the prediction.
- Step-5 : We can repeat this process and update the model for each training instance in the dataset.

Data Description:

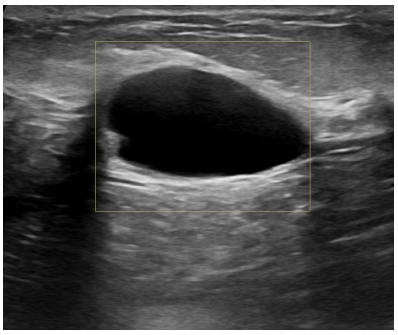
Research paper link -
https://reader.elsevier.com/reader/sd/pii/S2352340919312181?token=5743E39134AD964B3FF59190B2F0DBF022A8513A51E419C6A6A3E99B01A975E31B49998C88EB84952B0A4B193C10440C&originRegion=eu-west-1&originCreation=20220411190137
Dataset link -
https://www.kaggle.com/datasets/aryashah2k/breast-ultrasound-images-dataset

Dataset - We have downloaded the dataset from kaggle. The link is given below. The dataset is named **Breast Ultrasound Images Dataset.** The dataset consists of 780

images with an average image size of 500*500 pixels. The images are in PNG format. The ground truth images are presented with original images. The images are categorized into three classes, which are normal, benign, and malignant. There are 437 benign, 133 normal and 210 malignant BUS Images with their respective ground truth images.

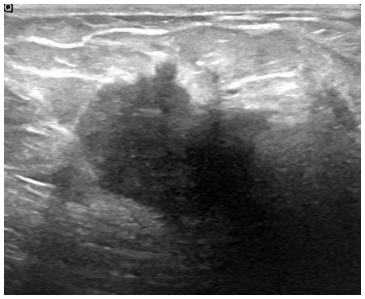
Sample of some images are shown below :

Benign

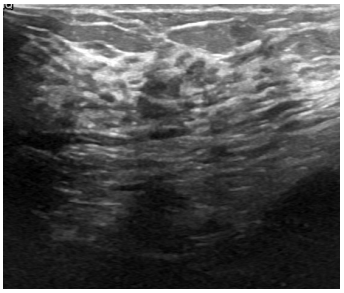


Benign

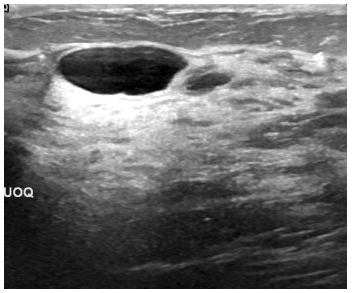
Malignant



Normal



Malignant

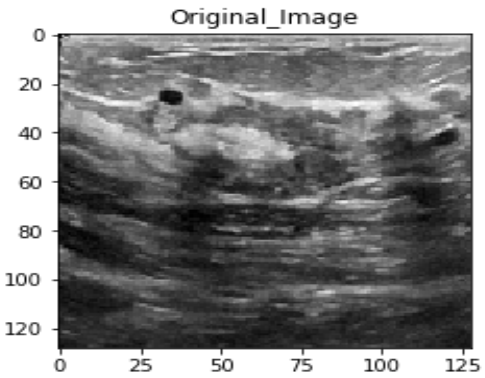


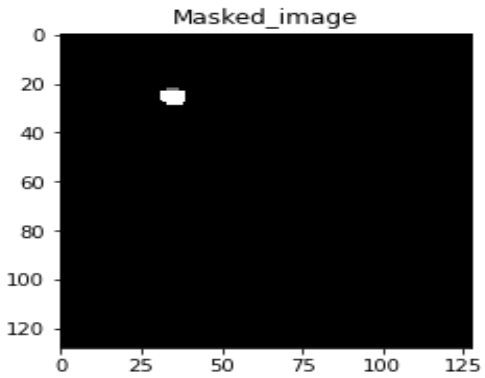
Benign

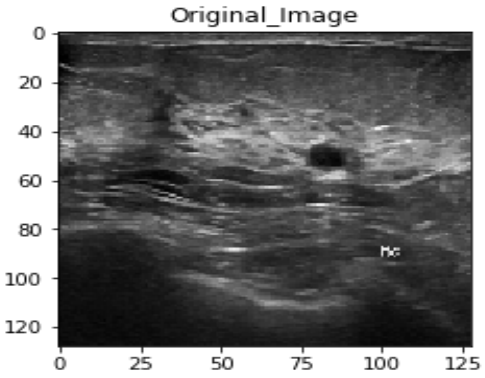
Results :

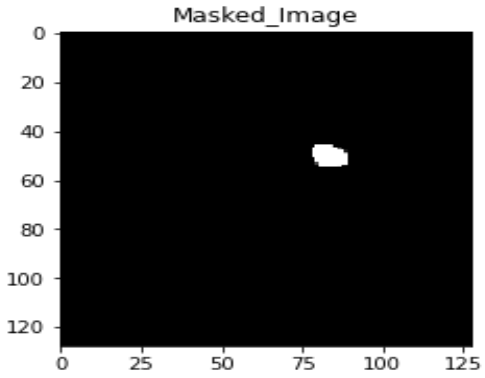
Task -1 :

1. In task-1, we have removed speckle noise from the BUS Images. The inputs that we have given is the original BUS image with their respective masks.

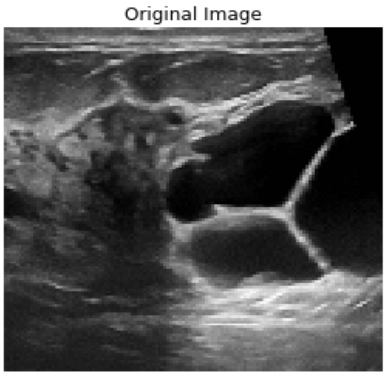


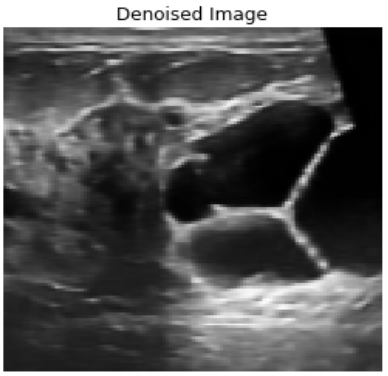


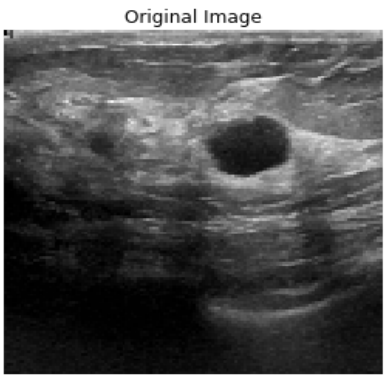


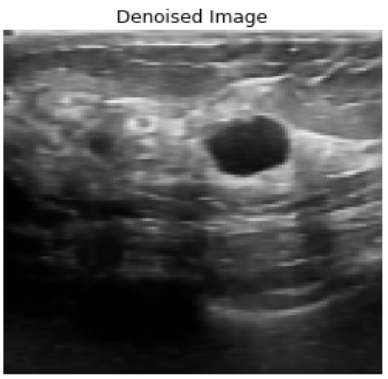


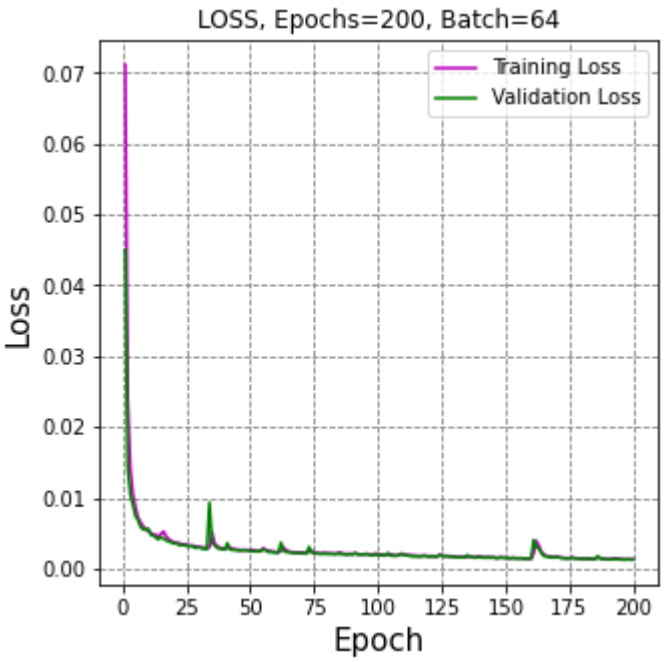
2. After **removing speckle noise** from the BUS Images, the output we got is shown below. On the left side, we have shown the input image with speckle noise. And on the other side, we have shown the denoised image,i.e. Images without speckle noise.









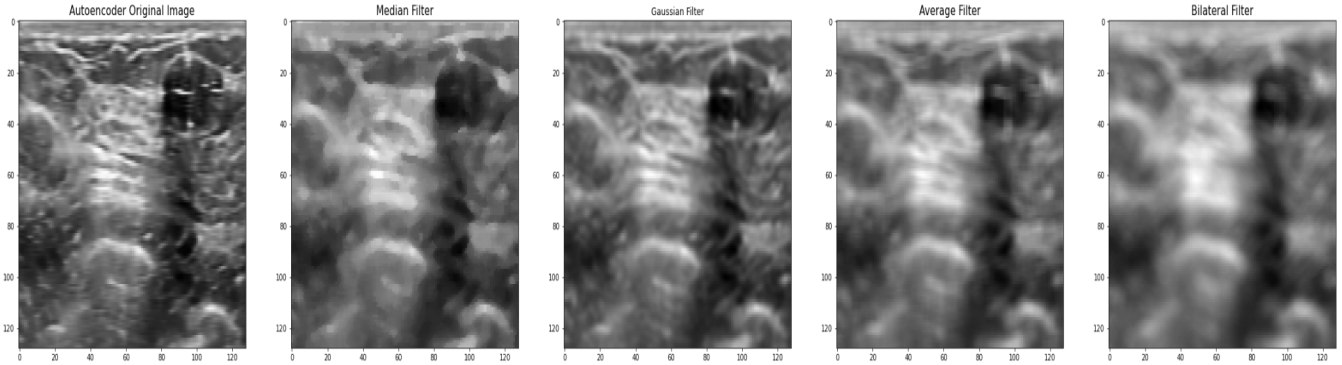


3. Loss Vs Epochs:

In the above figure, we have plotted the loss vs number of epochs curve. We can observe that both the training loss and validation loss are decreasing with increasing number of epochs. It means our denoising autoencoder is performing well and able to remove the speckle noise from the breast cancer ultrasound images.

4. **Comparison with other denoising filters:**

We have also compared our denoising autoencoder with other denoising filters. The output is shown below:



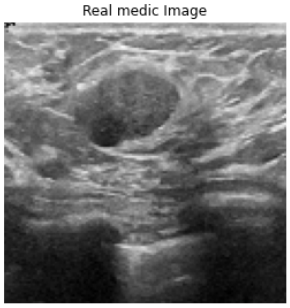
	Autoencoder	Median Blur	Gaussian Blur	Average Blur	Bilateral Filter
PSNR	26.627970	22.154908	24.352550	21.967357	21.017500
MSE	0.002174	0.006088	0.003671	0.006357	0.007911
SSIM	0.888428	0.690364	0.813406	0.665849	0.573199

5. We can clearly observe that our denoising autoencoder is performing a lot better than other filters. We have also calculated other parameters like peak signal to noise ratio(PSNR), mean square error(MSE) and structural similarity index(SSIM). The PSNR of denoising autoencoder is the best among all, the MSE is very low as compared to other filters and SSIM is very high as compared with other filters, which is a good sign.

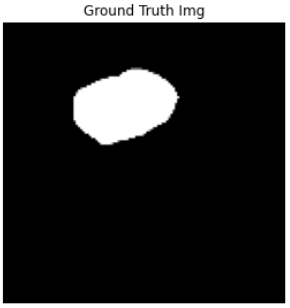
Task -2 :

1. In this task , we have segment out the tumor part from the BUS images. We have performed this task using U-Net which is specially designed for segmentation of biomedical images. The output is shown below. In the output, in the first column, we have shown the original BUS images with their respective masks in the second column and in the last column, we have shown the predicted mask that we got from the U-Net model. The validation loss and validation accuracy we got from our model is 0.02 and 97.46 %.

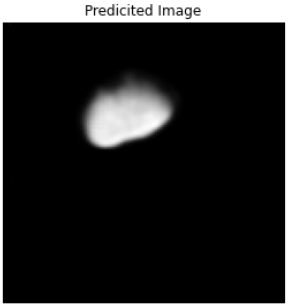
Real medic Image



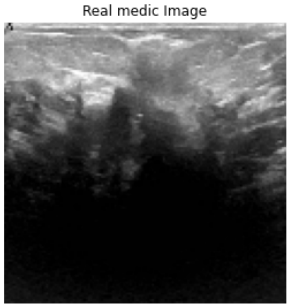
Ground Truth Img



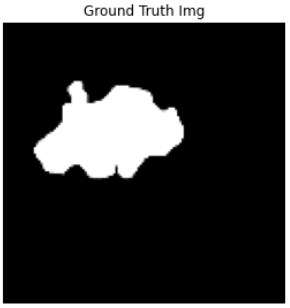
Predicited Image



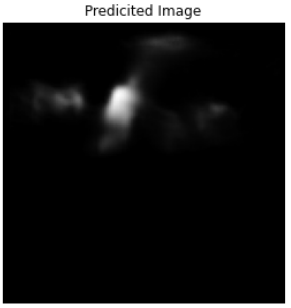
Real medic Image



Ground Truth Img

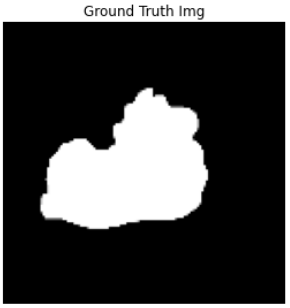


Predicited Image

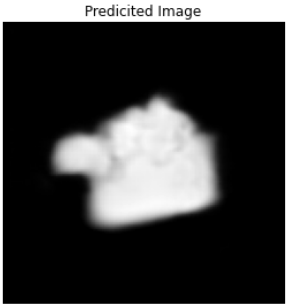


Real medic Image

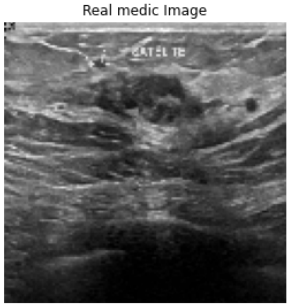
Ground Truth Img



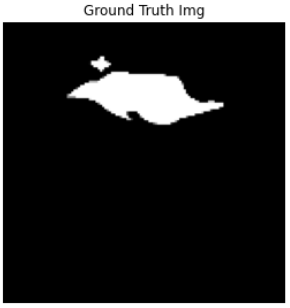
Predicited Image



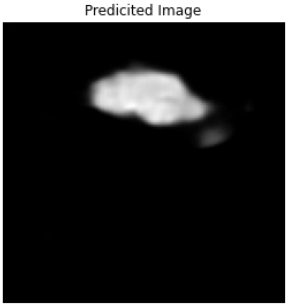
Real medic Image



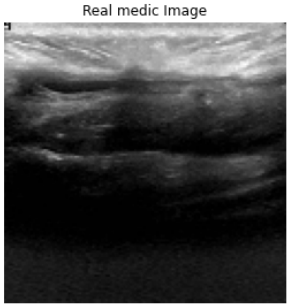
Ground Truth Img



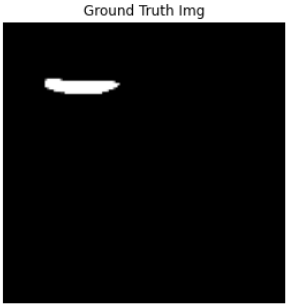
Predicited Image



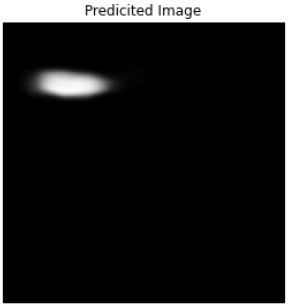
Real medic Image



Ground Truth Img

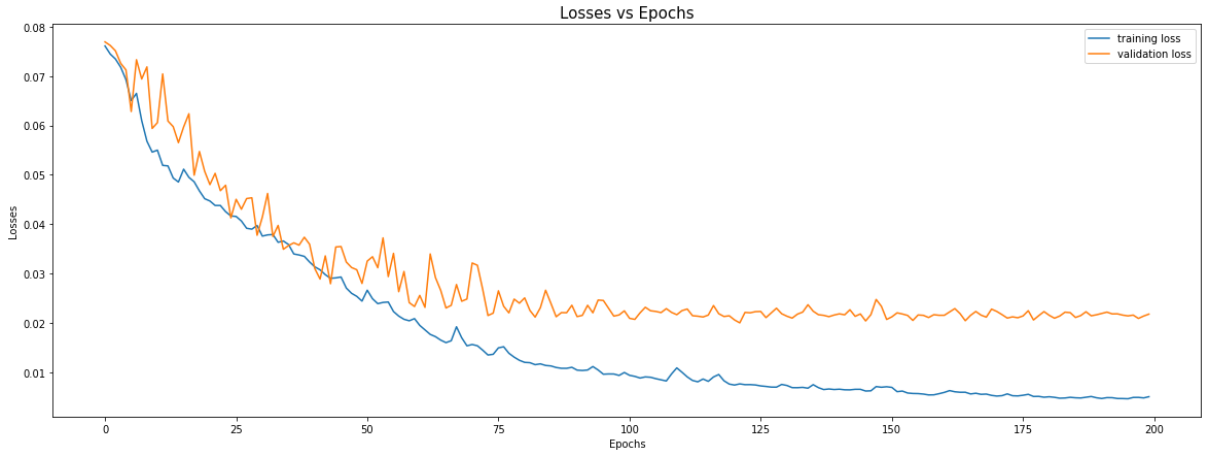


Predicited Image



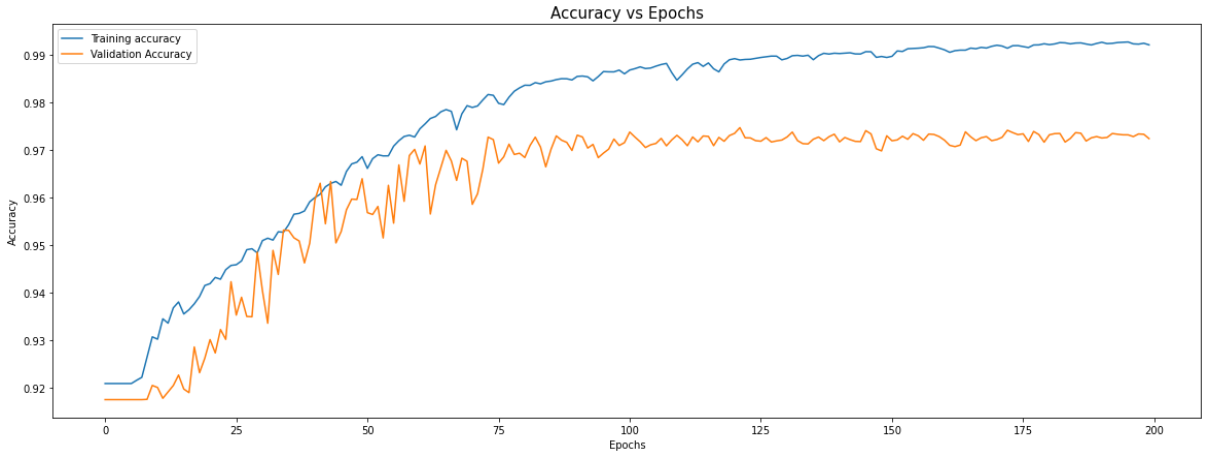
2. Loss vs Epochs:

We have plotted both the validation loss and training loss versus epochs.



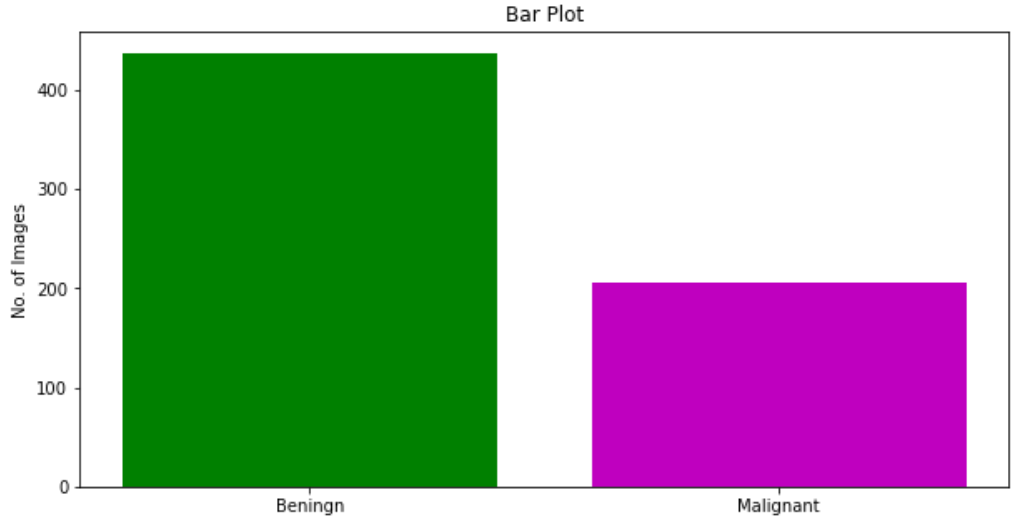
3. Accuracy vs Epochs

We have plotted both validation accuracy and training accuracy versus epochs.

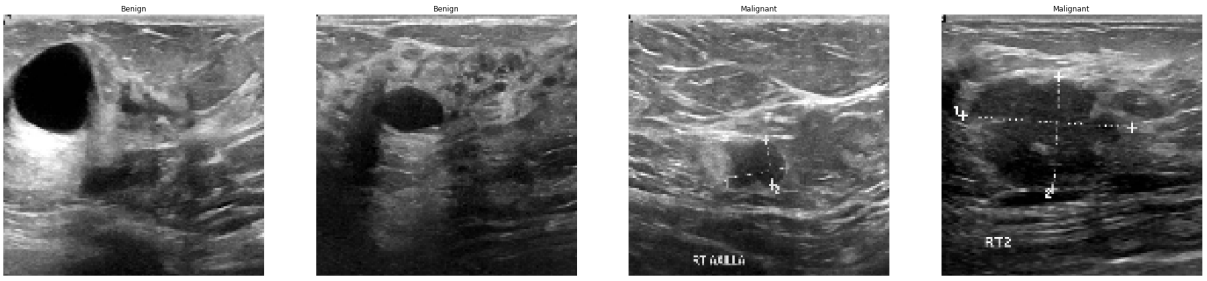


Task -3 :

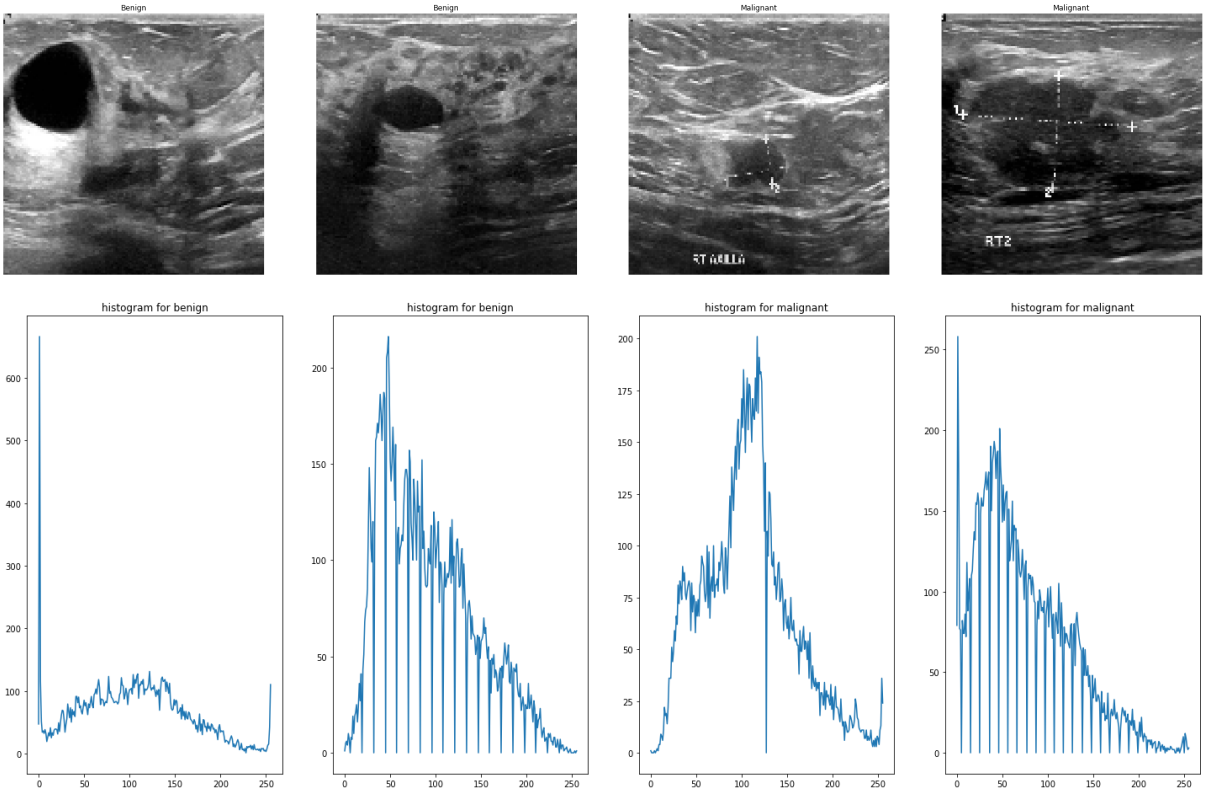
The bar plot shows the distribution of BUS images into normal and benign tumor.



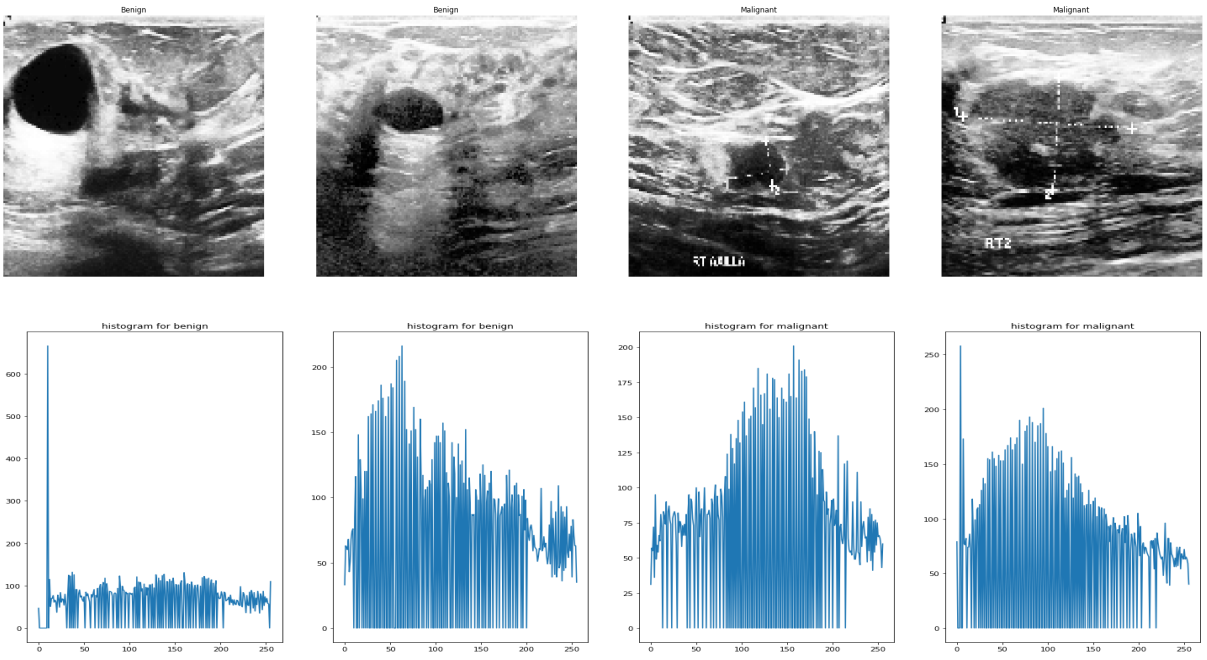
Input Images:



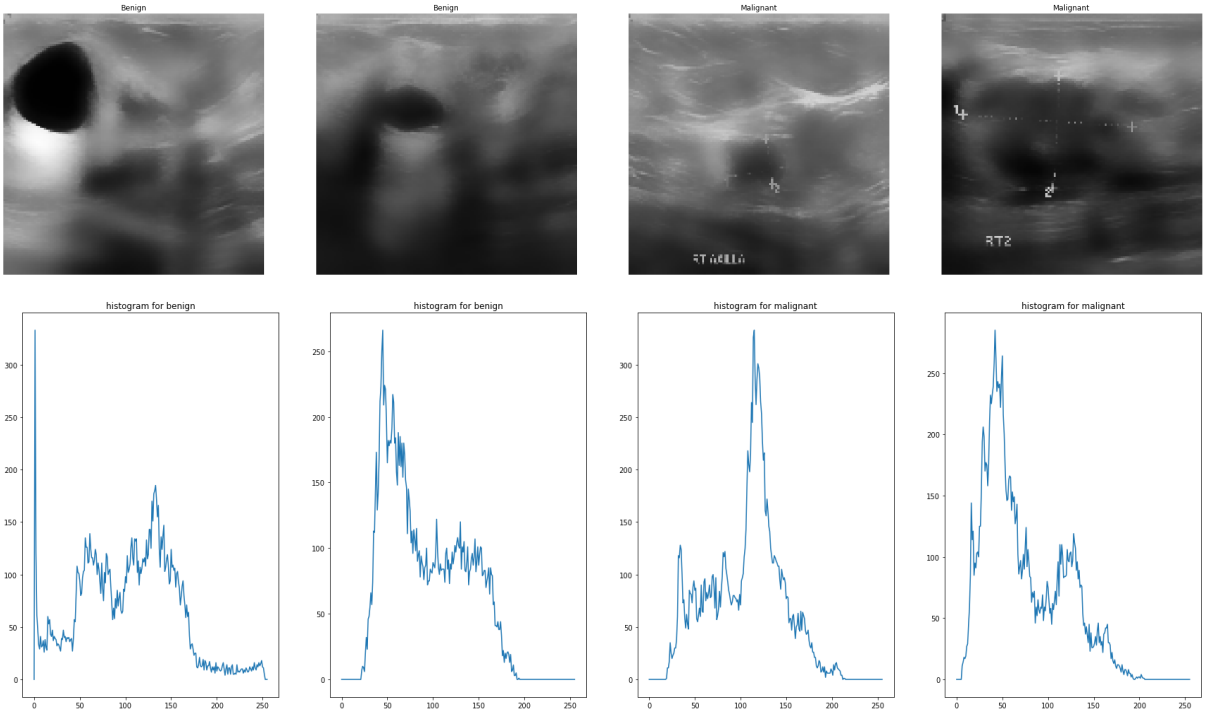
Grayscale Image with their histograms:



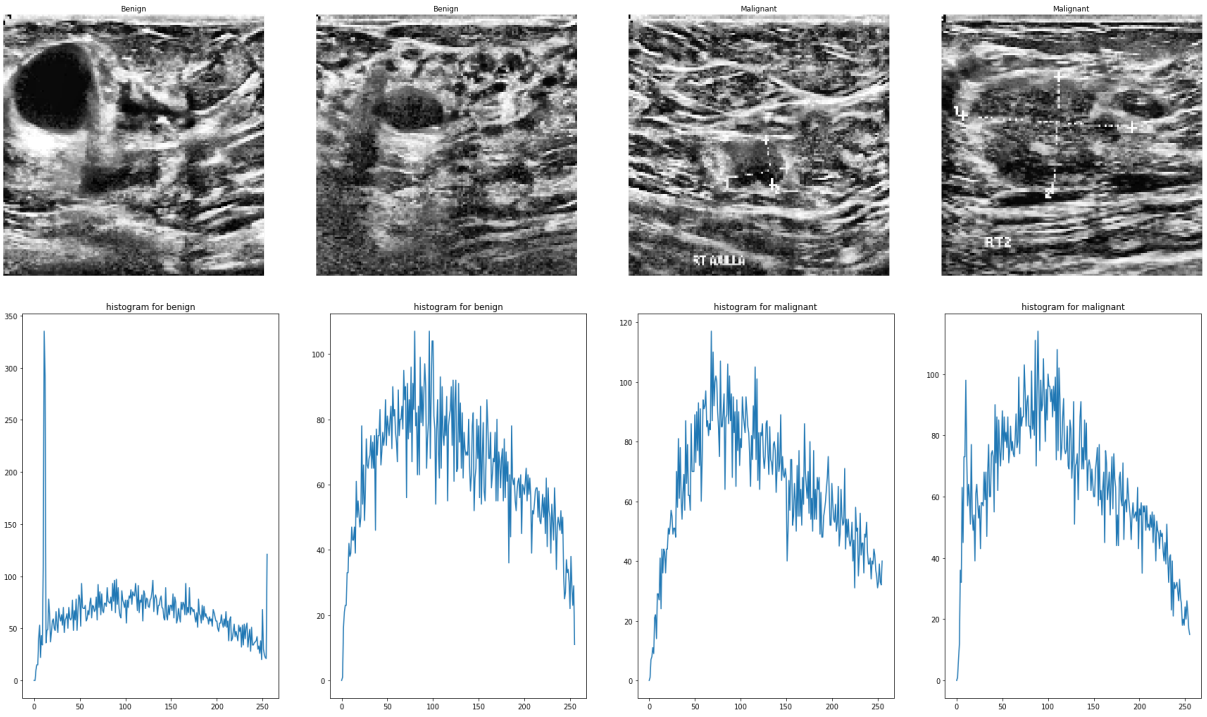
Output after histogram equalization with its histogram



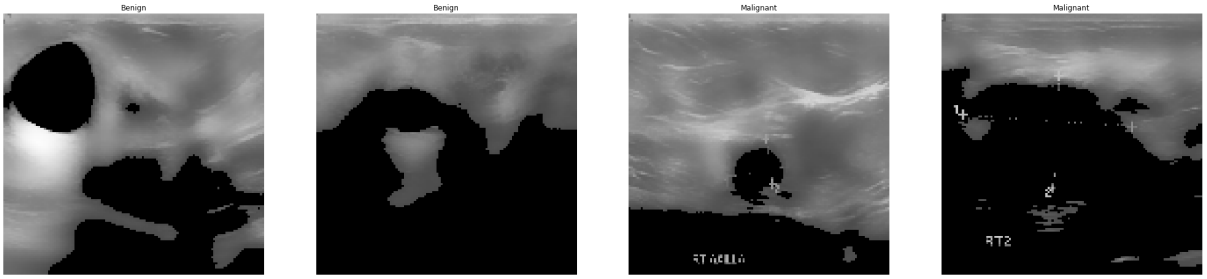
Output after using bilateral filter and its histogram :



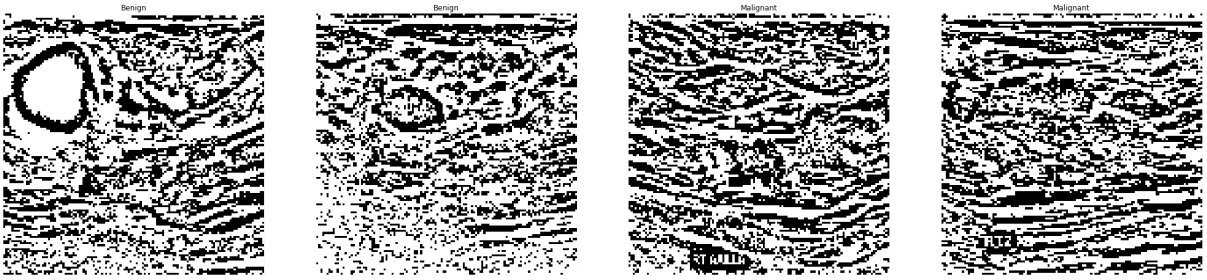
Output after using CLAHE and its histogram



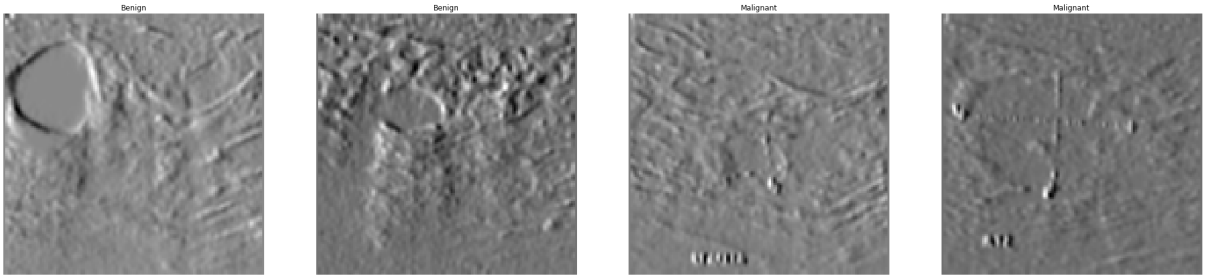
Output after applying Otsu's Binarization



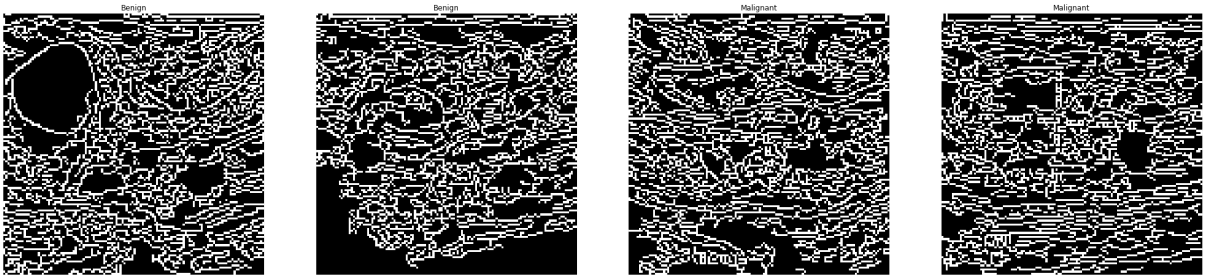
Output after applying Adaptive thresholding



Output of Sobel Edge Detection

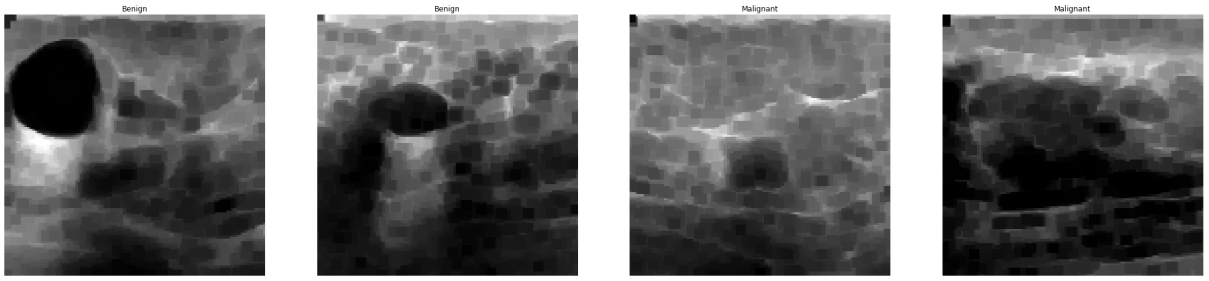


Output of Canny Edge Detection

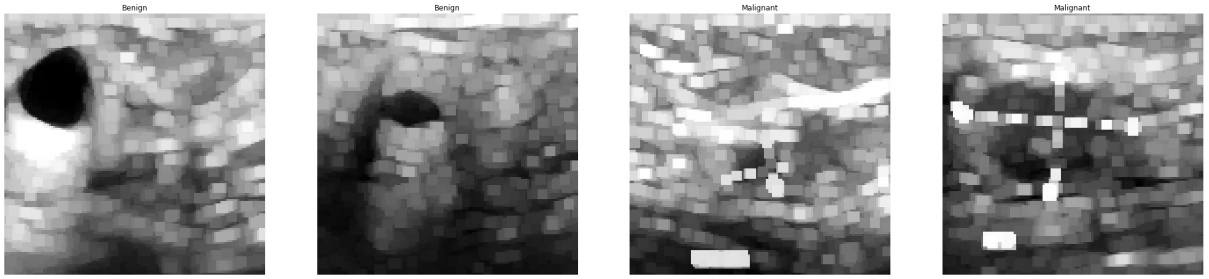


Output of Morphological Operations:

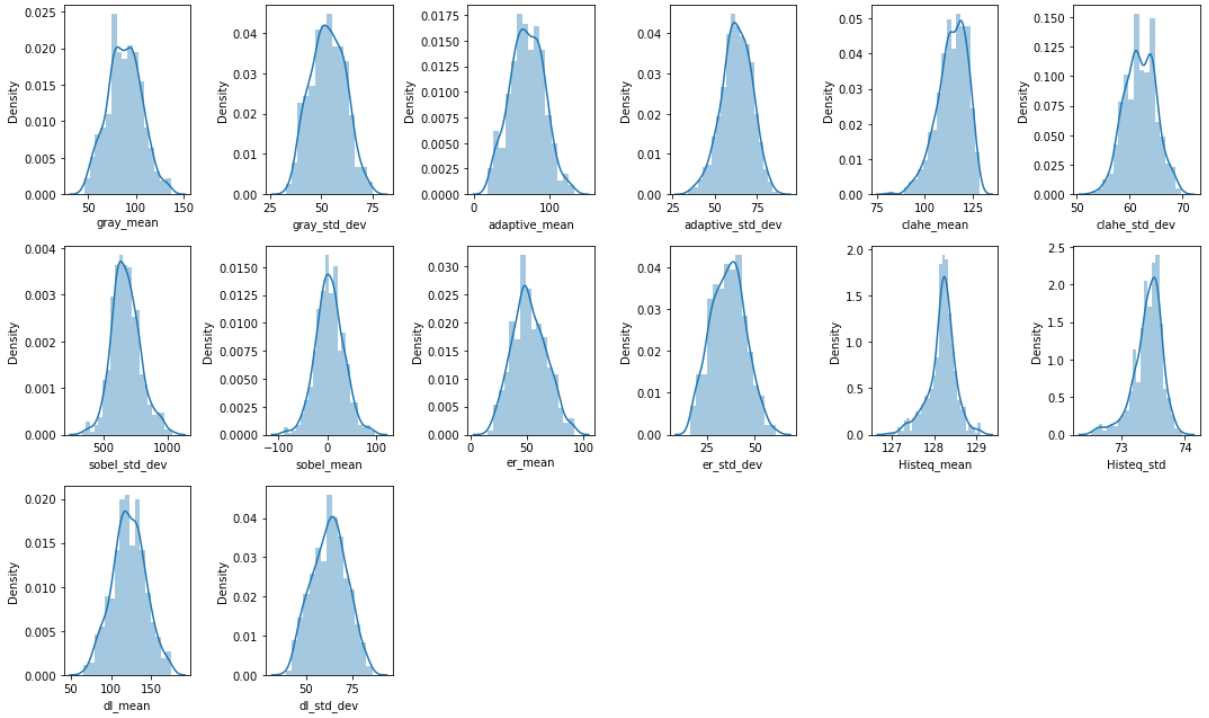
- Erosion



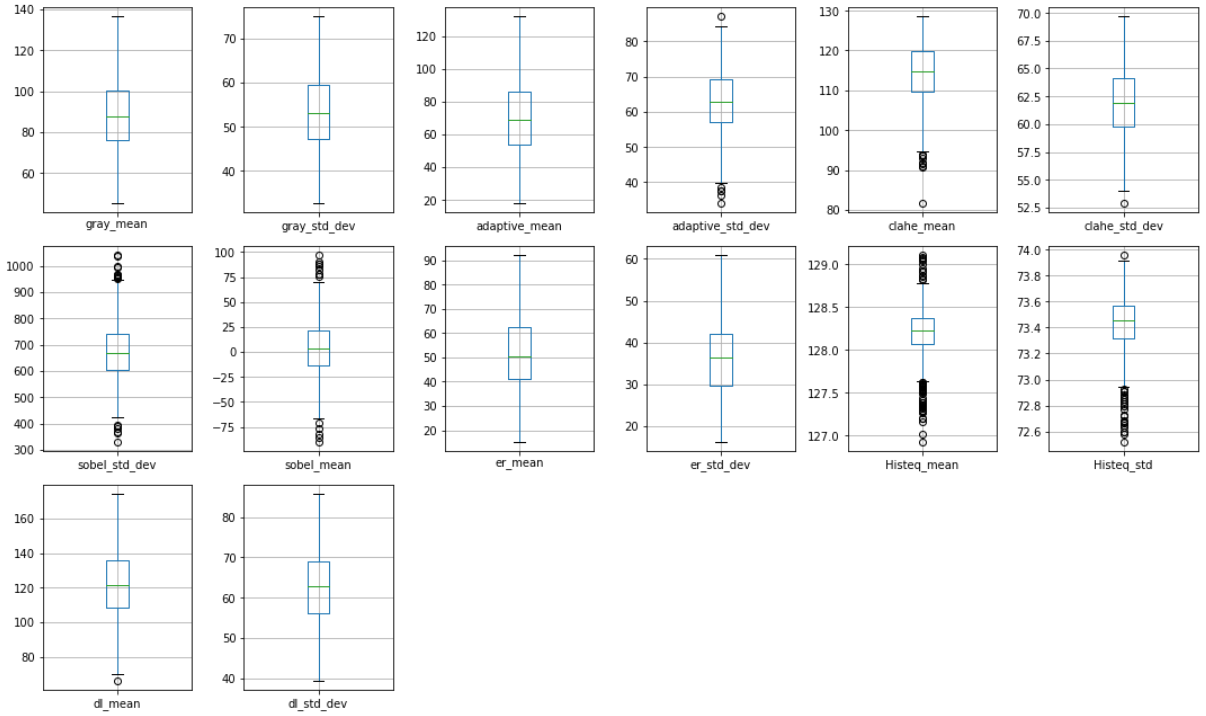
- **Dilation**



Distribution of features



Box plot of features



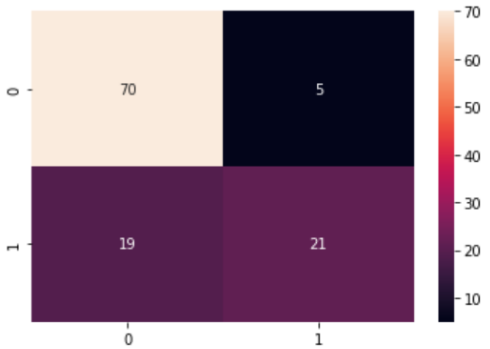
Machine Learning Algorithm results:

==========

```
Test Accuracy Score of Basic Logistics Regression: 79.13
              precision    recall  f1-score   support

      Benign       0.79      0.93      0.85        75
   Malignant       0.81      0.53      0.64        40

    accuracy                           0.79       115
   macro avg       0.80      0.73      0.75       115
weighted avg       0.79      0.79      0.78       115
 ['Benign', 'Malignant']
```



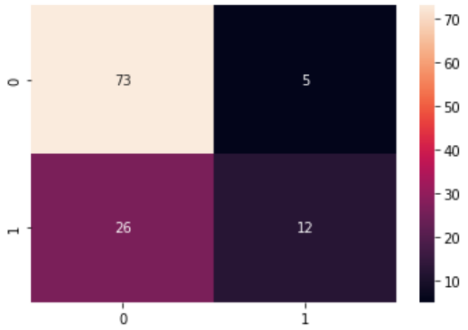
==========

==========

Test Accuracy Score of KNN: 73.28

	precision	recall	f1-score	support
Benign	0.74	0.94	0.82	78
Malignant	0.71	0.32	0.44	38
accuracy			0.73	116
macro avg	0.72	0.63	0.63	116
weighted avg	0.73	0.73	0.70	116

['Benign', 'Malignant']

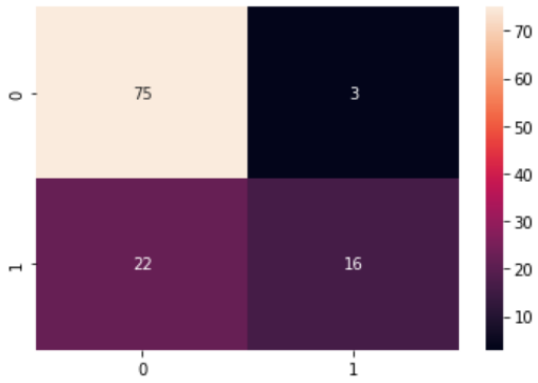


==========

Test Accuracy Score of SVM : 78.45

	precision	recall	f1-score	support
Benign	0.77	0.96	0.86	78
Malignant	0.84	0.42	0.56	38
accuracy			0.78	116
macro avg	0.81	0.69	0.71	116
weighted avg	0.80	0.78	0.76	116

['Benign', 'Malignant']

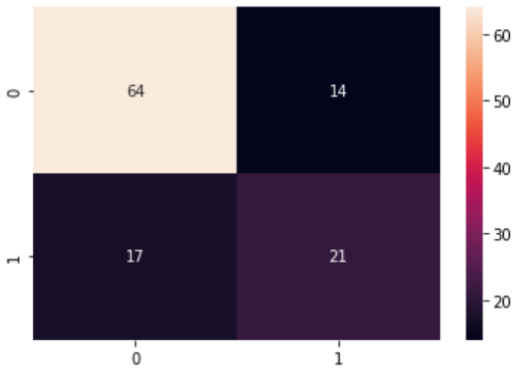


==========

```
Test Accuracy Score of Decision Tree is : 73.28
              precision    recall  f1-score   support

      Benign       0.79      0.82      0.81        78
   Malignant       0.60      0.55      0.58        38

    accuracy                           0.73       116
   macro avg       0.70      0.69      0.69       116
weighted avg       0.73      0.73      0.73       116
 ['Benign', 'Malignant']
```



==========

Conclusion :

We can conclude from our project that image preprocessing is a very essential step before performing tasks like segmentation and classification. We have implemented deep learning models like U-Net and Autoencoder, which have shown good results on our BUS dataset. We can also conclude that preprocessing techniques like image enhancement and edge detection also have shown good results. The Gray co-occurrence matrix(GLCM) features like energy , entropy, contrast, correlation, homogeneity etc performed a nice task to find relation between the pixels. Applying Machine learning algorithms like SVM, KNN, and Random Forest using features, showed average results.

References :

1. https://link.springer.com/chapter/10.1007/978-3-319-24574-4_28
2. https://ieeexplore.ieee.org/document/9302250
3. https://reader.elsevier.com/reader/sd/pii/S2352340919312181?token=5743E39134AD964B3FF59190B2F0DBF022A8513A51E419C6A6A3E99B01A975E31B49998C88EB84952B0A4B193C10440C&originRegion=eu-west-1&originCreation=20220411190137

4. https://www.kaggle.com/datasets/aryashah2k/breast-ultrasound-images-dataset
5. https://ieeexplore.ieee.org/document/9302250
6. https://towardsdatascience.com/autoencoders-and-the-denoising-feature-from-theory-to-practice-db7f7ad8fc78
7. https://www.analyticsvidhya.com/blog/2020/10/the-mathematics-behind-svm/
8. https://machinelearningmastery.com/logistic-regression-tutorial-for-machine-learning/
9. https://medium.com/@priyankaparashar54/decision-tree-classification-and-its-mathematical-implementation-c27006caefbb
10. https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm

Appendix :

Denoising Autoencoder

```
import re
import numpy as np
import pandas as pd
import random
import math
import cv2
import seaborn as sns
from glob import glob
import os
from pathlib import Path
import itertools
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D,MaxPool2D ,UpSampling2D, Flatten, Input,LeakyReLU,BatchNormalization,Dropout
from tensorflow.nn import atrous_conv2d
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping,ModelCheckpoint, ReduceLROnPlateau, CSVLogger
import matplotlib.pyplot as plt
```

```
from skimage.util import random_noise
from skimage.metrics import peak_signal_noise_ratio as psnr, structural_similarity as ssim,mean_squared_error as mse
%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'



#global_path = 'C:/Users/lenovo/Downloads/breast-ultrasound-images-dataset/Dataset_BUSI_with_GT/*/*.png'
path = '/content/drive/My Drive/BUS_Images/*/*.png'
image_path = glob(path)
image_path.sort()
print(len(image_path))

# creating dictionary of images and masks
image = []
masked_image = []
count = 1
i = 0
num = len(image_path)
images_with_class = {}
while i < num-1:
    print(str(count) + "/" + str(num), end="\r")
    img = image_path[i]
    a = Path(img).parts
    # get label from image name
    label=(a[-2])
    # get image name
    image_name = img.split('/')[-1].split(')')[0]
    if label not in images_with_class.keys():
        # make nested dictionory for each label
        images_with_class[label] = {'Original_Image':[],'Mask_Image':[]}
    if img.split('/')[-1][-5] == ')':
        # get the image
        image = cv2.resize(cv2.imread(img,cv2.IMREAD_GRAYSCALE),(128,128))
        images_with_class[label]['Original_Image'].append(image)
        mylist = image_path[i:]
        # get the masked_image
```

```
        r = re.compile(r".*"+re.escape(image_name)+r"\)_mask*.")
        masklist = list(filter(r.match, mylist))
        if len(masklist) == 1:
            masked_image = cv2.resize(cv2.imread(masklist[0]),(128,128))
            images_with_class[label]['Mask_Image'].append(masked_image)
        else:  #there are two masks
            masked_1 = cv2.imread(masklist[0])
            masked_2 = cv2.imread(masklist[1])
            masked_image = cv2.resize(cv2.bitwise_or(masked_1,masked_2),(128,128))
            images_with_class[label]['Mask_Image'].append(masked_image)


    i+=1

    count += 1


# Number of images per class
labels = []
for keys,values in images_with_class.items():
    #print(value)
    for key,value in values.items():
        (labels.extend([key]*len(value)))

        print("{0} : {1} : {2} ".format(keys,key, len(value)))



fig,axis = plt.subplots(2,2)
axis[0,0].imshow(images_with_class['benign']['Original_Image'][50])
axis[0,0].set_title('Original_Image')
axis[0,1].imshow(images_with_class['benign']['Mask_Image'][50])
axis[0,1].set_title('Masked_image')
axis[1,0].imshow(images_with_class['benign']['Original_Image'][100])
axis[1,0].set_title('Original_Image')
axis[1,1].imshow(images_with_class['benign']['Mask_Image'][100])
axis[1,1].set_title('Masked_Image')

#dataset for denoising consists of only images
```

```
dataset = images_with_class['benign']['Original_Image'] + images_with_class['malignant']['Original_Image'] + images_with_class['normal']['Original_Image']
#normalizing pixel values in the dataset
# The main purpose of the normallization is to make computation efficient by reducing values between 0 and 1.
dataset = [img/255 for img in dataset] # for faster computation
random.shuffle(dataset) # this function shuffles the dataset randomly

# Function to plotting set of images
def plot(dataset):
  fig,axis=plt.subplots(1,4)
  fig.set_size_inches(10,10)
  for i in range(50,54):
    axis[i-50].imshow(dataset[i])
  plt.show()

## adding speckle noise to the dataset
noisy_dataset=[]
for image in dataset:
  # Introducing speckle noise
  noise =random_noise(image,mode = 'speckle')
  noisy_dataset.append(noise)
dataset = np.array(dataset)
noisy_dataset = np.array(noisy_dataset)
len(noisy_dataset)
plot(dataset)
plot(noisy_dataset)

# Splitting dataset for training and testing
x_train = dataset[:624]
x_test = dataset[624:]
x_train_noisy = noisy_dataset[:624]
x_test_noisy= noisy_dataset[624:]

def denoising_autoencoder():
    i=Input(shape=(128,128,1))
    #encoder
    x = Conv2D(128, (3,3), activation='relu', padding='same')(i)
    x = MaxPooling2D((2,2), padding='same')(x)
    x = Conv2D(128, (3,3), activation='relu', padding='same')(x)
```

```
    x = MaxPooling2D((2,2), padding='same')(x)
    x = Conv2D(128, (3,3), activation='relu', padding='same')(x)
    x = Conv2D(128, (3,3), activation='relu', padding='same')(x)

    #decoder
    x = Conv2D(128, (3,3), activation='relu', padding='same')(x)
    x = Conv2D(128, (3,3), activation='relu', padding='same')(x)
    x = UpSampling2D((2,2))(x)
    x = Conv2D(128, (3,3), activation='relu', padding='same')(x)
    x = UpSampling2D((2,2))(x)
    x = Conv2D(1, (3,3), activation='sigmoid', padding='same')(x)

    #model
    autoencoder = Model(inputs=i, outputs=x)
    autoencoder.compile(optimizer='Adam', loss='mse',metrics=['accuracy'])
    autoencoder.summary()

    return autoencoder


 ## Model
autoencoder = denoising_autoencoder()

Epochs = 200
Batch_size = 64
from keras.callbacks import ModelCheckpoint
checkp = ModelCheckpoint('/content/drive/MyDrive/results/network.h5', monitor = 'val_loss', save_best_only = True, verbose = 1)
#learning_rate_red1 = ReduceLROnPlateau(monitor='val_loss',patience=8,verbose=1,factor=0.8, min_lr=1e-10)
call_backs = [checkp]
# r = autoencoder.fit(x_train, x_train,
#             epochs=Epochs,
#             batch_size=Batch_size,
#             shuffle=True,
#             validation_data=(x_test, x_test),
#             callbacks=[call_backs],verbose=0)
# # #model evaluation
# # print("Train score:", autoencoder1.evaluate(x_train_noisy,x_train_noisy))
# # print("Test score:", autoencoder1.evaluate(x_test_noisy,x_test_noisy))
```

```
#autoencoder.save('/content/drive/MyDrive/results/network.h5')

from keras.models import load_model
model = load_model("/content/drive/MyDrive/results/network.h5")
#model.evaluate(x_test, x_test)

y_pred = model.predict(x_test)
print(y_pred.shape)

y_pred_x_train = model.predict(x_train)
print(y_pred_x_train.shape)

plt.figure(figsize = (20,80))
i = 0
x = 0
while i < 10 :
   plt.subplot(15,3,i+1)
   plt.imshow(x_test[x], 'gray')
   plt.title('Original Image')
   plt.axis('off')
   plt.subplot(15,3,i+2)
   plt.imshow(y_pred[x].reshape(128,128), 'gray')
   plt.title('Denoised Image')
   plt.axis('off')
   x += 1
   i += 3
plt.show()

fig,axis = plt.subplots(2,2)
axis[0,0].imshow(y_pred[45].reshape(128,128))
axis[0,0].set_title('Denoise_Image-1')
axis[0,1].imshow(y_pred[10].reshape(128,128))
axis[0,1].set_title('Denoise_Image-2')
axis[1,0].imshow(y_pred[56].reshape(128,128))
axis[1,0].set_title('Denoise_Image-3')
axis[1,1].imshow(y_pred[5].reshape(128,128))
axis[1,1].set_title('Denoise_Image-4')
```

```
## comparing the denoised original image with other filters
median_blur = cv2.medianBlur(np.float32(x_test[0]), (5))
gaussian_blur=cv2.GaussianBlur(x_test[0],(5,5),0)
average_blur=cv2.blur(x_test[0],(5,5))
bilateral_filter=cv2.bilateralFilter(np.float32(x_test[0]),9,75,75)
f,ax=plt.subplots(1,5)
f.set_size_inches(40,20)
ax[0].imshow(y_pred[0].reshape(128,128), cmap='gray')
ax[0].set_title('Autoencoder Original Image',fontsize = 15)
ax[1].imshow(median_blur,cmap='gray')
ax[1].set_title('Median Filter',fontsize = 15)
ax[2].imshow(gaussian_blur,cmap='gray')
ax[2].set_title('Gaussian Filter')
ax[3].imshow(average_blur,cmap='gray')
ax[3].set_title('Average Filter',fontsize = 15)
ax[4].imshow(bilateral_filter,cmap='gray')
ax[4].set_title('Bilateral Filter',fontsize = 15)
plt.show()

## function to plot the curve between loss and epochs(Original Images)
def plotLearningCurve(history,Epochs,Batch_size):
  epochRange = range(1,Epochs+1)
  plt.figure(figsize = (5,5))
  plt.plot(epochRange,history.history['loss'],'m',label = 'Training Loss')
  plt.plot(epochRange,history.history['val_loss'],'g',label = 'Validation Loss')
  plt.xlabel('Epoch', fontsize = 15)
  plt.ylabel('Loss', fontsize = 15)
  plt.grid(color='gray', linestyle='--')
  plt.legend()
  plt.title('LOSS, Epochs={}, Batch={}'.format(Epochs, Batch_size))
  plt.show()


plotLearningCurve(r,Epochs,Batch_size)


value1 = psnr(x_test[0], y_pred[0].reshape(x_test[0].shape[0],-1))
value2 = psnr(x_test[0], median_blur)
```

```
value3 = psnr(x_test[0], gaussian_blur)
value4 = psnr(x_test[0], average_blur)
value5 = psnr(x_test[0], bilateral_filter)
metric = pd.DataFrame({'Autoencoder':value1,'Median Blur':value2,'Gaussian Blur':value3,'Average Blur':value4,'Bilateral Filter':value5},index = ['PSNR'])
value1 = mse(x_test[0], y_pred[0].reshape(x_test[0].shape[0],-1))
value2 = mse(x_test[0], median_blur)
value3 = mse(x_test[0], gaussian_blur)
value4 = mse(x_test[0], average_blur)
value5 = mse(x_test[0], bilateral_filter)
metric.loc['MSE'] = [value1,value2,value3,value4,value5]
value1 = ssim(x_test[0], y_pred[0].reshape(x_test[0].shape[0],-1))
value2 = ssim(x_test[0], median_blur)
value3 = ssim(x_test[0], gaussian_blur)
value4 = ssim(x_test[0], average_blur)
value5 = ssim(x_test[0], bilateral_filter)
metric.loc['SSIM'] = [value1,value2,value3,value4,value5]
metric
```

```
def denoise_autoencoder():
    i=Input(shape=(128,128,1))
    #encoder
    x = Conv2D(128, (3,3), activation='relu', padding='same')(i)
    x = MaxPooling2D((2,2), padding='same')(x)
    x = Conv2D(128, (3,3), activation='relu', padding='same')(x)
    x = MaxPooling2D((2,2), padding='same')(x)
    x = Conv2D(128, (3,3), activation='relu', padding='same')(x)
    x = Conv2D(128, (3,3), activation='relu', padding='same')(x)

    #decoder
    x = Conv2D(128, (3,3), activation='relu', padding='same')(x)
    x = Conv2D(128, (3,3), activation='relu', padding='same')(x)
    x = UpSampling2D((2,2))(x)
    x = Conv2D(128, (3,3), activation='relu', padding='same')(x)
    x = UpSampling2D((2,2))(x)
    x = Conv2D(1, (3,3), activation='sigmoid', padding='same')(x)

    #model
```

```
    autoencoder = Model(inputs=i, outputs=x)
    autoencoder.compile(optimizer='Adam', loss='mse',metrics=['accuracy'])
    autoencoder.summary()

    return autoencoder

## Model
autoencoder1 = denoise_autoencoder()



Epochs1 = 200
Batch_size1 = 64
from keras.callbacks import ModelCheckpoint
checkp = ModelCheckpoint('/content/drive/MyDrive/results/network1.h5', monitor = 'val_loss', save_best_only = True, verbose = 1)
#learning_rate_red1 = ReduceLROnPlateau(monitor='val_loss',patience=8,verbose=1,factor=0.8, min_lr=1e-10)
call_backs = [checkp]
# r = autoencoder1.fit(x_train_noisy, x_train_noisy,
#               epochs=Epochs1,
#               batch_size=Batch_size1,
#               shuffle=True,
#               validation_data=(x_test_noisy, x_test_noisy),
#               callbacks=[call_backs],verbose=0)
# # #model evaluation
# # print("Train score:", autoencoder1.evaluate(x_train_noisy,x_train_noisy))
# # print("Test score:", autoencoder1.evaluate(x_test_noisy,x_test_noisy))

from keras.models import load_model
model1 = load_model("/content/drive/MyDrive/results/network1.h5")

y_pred1 = model1.predict(x_test_noisy)
print(y_pred1.shape)

plt.figure(figsize = (20,80))
i = 0
x = 0
while i < 10 :
    plt.subplot(15,3,i+1)
    plt.imshow(x_test_noisy[x], 'gray')
```

```
    plt.title('Original Image')
    plt.axis('off')
    plt.subplot(15,3,i+2)
    plt.imshow(y_pred1[x].reshape(128,128), 'gray')
    plt.title('Denoised Image')
    plt.axis('off')
    x += 1
    i += 3
plt.show()

fig,axis = plt.subplots(2,2)
axis[0,0].imshow(y_pred1[0].reshape(128,128))
axis[0,0].set_title('Denoised_image-1')
axis[0,1].imshow(y_pred1[2].reshape(128,128))
axis[0,1].set_title('Denoised_image-2')
axis[1,0].imshow(y_pred1[5].reshape(128,128))
axis[1,0].set_title('Denoised_image-3')
axis[1,1].imshow(y_pred1[6].reshape(128,128))
axis[1,1].set_title('Denoised_image-4')

## comparing the denoised noisy image with other filters
median_blur1 = cv2.medianBlur(np.float32(x_test_noisy[0]), (5))
gaussian_blur1=cv2.GaussianBlur(x_test_noisy[0],(5,5),0)
average_blur1=cv2.blur(x_test_noisy[0],(5,5))
bilateral_filter1=cv2.bilateralFilter(np.float32(x_test_noisy[0]),9,75,75)
f,ax=plt.subplots(1,5)
f.set_size_inches(40,20)
ax[0].imshow(y_pred1[0].reshape(128,128), cmap='gray')
ax[0].set_title('Autoencoder Noisy Image',fontsize = 15)
ax[1].imshow(median_blur1,cmap='gray')
ax[1].set_title('Median Filter',fontsize = 15)
ax[2].imshow(gaussian_blur1,cmap='gray')
ax[2].set_title('Gaussian Filter')
ax[3].imshow(average_blur1,cmap='gray')
ax[3].set_title('Average Filter',fontsize = 15)
ax[4].imshow(bilateral_filter1,cmap='gray')
ax[4].set_title('Bilateral Filter',fontsize = 15)
plt.show()

## function to plot the curve between loss and epochs(Noisy Images)
```

```
def plotLearningCurve1(history,Epochs1,Batch_size1):
  epochRange = range(1,Epochs1+1)
  plt.figure(figsize = (5,5))
  plt.plot(epochRange,history.history['loss'],'r',label = 'Training Loss')
  plt.plot(epochRange,history.history['val_loss'],'y',label = 'Validation Loss')
  plt.xlabel('Epoch', fontsize = 15)
  plt.ylabel('Loss', fontsize = 15)
  plt.grid(color='gray', linestyle='--')
  plt.legend()
  plt.title('LOSS, Epochs={}, Batch={}'.format(Epochs1, Batch_size1))
  plt.show()


## function to plot the curve between loss and epochs(Noisy Images)
def plotLearningCurve2(history,Epochs1,Batch_size1):
  epochRange = range(1,Epochs1+1)
  plt.figure(figsize = (5,5))
  plt.plot(epochRange,history.history['accuracy'],'r',label = 'Training Accuracy')
  #plt.plot(epochRange,history.history['val_loss'],'y',label = 'Validation Loss')
  plt.xlabel('Epoch', fontsize = 15)
  plt.ylabel('Accuracy', fontsize = 15)
  plt.grid(color='gray', linestyle='--')
  plt.legend()
  plt.title('Accuracy, Epochs={}, Batch={}'.format(Epochs1, Batch_size1))
  plt.show()

plotLearningCurve2(r,Epochs1,Batch_size1)

plotLearningCurve1(r,Epochs1,Batch_size1)

value1 = psnr(x_test_noisy[0], y_pred1[0].reshape(x_test[0].shape[0],-1))
value2 = psnr(x_test_noisy[0], median_blur)
value3 = psnr(x_test_noisy[0], gaussian_blur)
value4 = psnr(x_test_noisy[0], average_blur)
value5 = psnr(x_test_noisy[0], bilateral_filter)
metric = pd.DataFrame({'Autoencoder':value1,'Median Blur':value2,'Gaussian Blur':value3,'Average
Blur':value4,'Bilateral Filter':value5},index = ['PSNR'])
value1 = mse(x_test_noisy[0], y_pred1[0].reshape(x_test_noisy[0].shape[0],-1))
value2 = mse(x_test_noisy[0], median_blur)
value3 = mse(x_test_noisy[0], gaussian_blur)
```

```
value4 = mse(x_test_noisy[0], average_blur)
value5 = mse(x_test_noisy[0], bilateral_filter)
metric.loc['MSE'] = [value1,value2,value3,value4,value5]
value1 = ssim(x_test_noisy[0], y_pred1[0].reshape(x_test_noisy[0].shape[0],-1))
value2 = ssim(x_test_noisy[0], median_blur)
value3 = ssim(x_test_noisy[0], gaussian_blur)
value4 = ssim(x_test_noisy[0], average_blur)
value5 = ssim(x_test_noisy[0], bilateral_filter)
metric.loc['SSIM'] = [value1,value2,value3,value4,value5]
metric
```

==========

Segmentation Using Unet Model

```
## Import Libraries
"""

import os
import cv2
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from tensorflow import keras
from keras.preprocessing.image import img_to_array
#path = 'C:/Users/ANKUR/Downloads/Dataset_BUSI_with_GT/'
#path = '../input/breast-ultrasound-images-dataset/Dataset_BUSI_with_GT/'

# Load the Drive helper and mount
from google.colab import drive
drive.mount('/content/drive')

path = '/content/drive/My Drive/Original_dataset/'

def number_of_images(img) :
    value = 0
    for i in range(len(img)) :
        if img[i] == '(' :
            while True :
                i += 1
                if img[i] == ')' :
                    break
```

```
            value = (value*10) + int(img[i])
        break

    return value

X_b, y_b = np.zeros((437, 128, 128, 1)), np.zeros((437, 128, 128, 1))
X_n, y_n = np.zeros((133, 128, 128, 1)), np.zeros((133, 128, 128, 1))
X_m, y_m = np.zeros((210, 128, 128, 1)), np.zeros((210, 128, 128, 1))

for i, tumor_type in enumerate(os.listdir(path)) :
    for image in os.listdir(path+tumor_type+'/') :
        p = os.path.join(path+tumor_type, image)
        img = cv2.imread(p,cv2.IMREAD_GRAYSCALE)          # read image as  grayscale

        if image[-5] == ')' :

            img = cv2.resize(img,(128,128))
            pil_img = Image.fromarray (img)

            if image[0] == 'b' :
                X_b[number_of_images(image)-1]+= img_to_array(pil_img)  # If image is real add it
            if image[0] == 'n' :                        # to X as benign , normal
                X_n[number_of_images(image)-1]+= img_to_array(pil_img)  # or malignant.
            if image[0] == 'm' :
                X_m[number_of_images(image)-1]+= img_to_array(pil_img)
        else :
            img = cv2.resize(img,(128,128))
            pil_img = Image.fromarray (img)

            if image[0] == 'b' :
                y_b[number_of_images(image)-1]+= img_to_array(pil_img)  # Similarly add the target
            if image[0] == 'n' :                        # mask to y.
                y_n[number_of_images(image)-1]+= img_to_array(pil_img)
            if image[0] == 'm' :
                y_m[number_of_images(image)-1]+= img_to_array(pil_img)

plt.figure(figsize = (20,10))

for i in range(5) :
    plt.subplot(2,5,i+1)
```

```
    plt.imshow(X_b[i+1].reshape(128,128), 'gray')
    plt.title('Real Image')
    plt.axis('off')

for i in range(5) :
    plt.subplot(2,5,i+6)
    plt.imshow(y_b[i+1].reshape(128,128), 'gray')
    plt.title('Mask Image')
    plt.axis('off')
plt.show()

X = np.concatenate((X_b, X_n, X_m), axis = 0)
y = np.concatenate((y_b, y_n, y_m), axis = 0)
X /= 255.0
y /= 255.0
print(X.shape)
print(y.shape)
print(X.max())
print(X.min())
print(y.max())
print(y.min())

y[y > 1.0] = 1.0
print(y.max())
print(y.min())

plt.figure(figsize = (10,30))
i = 0
while i < 16 :

    x = np.random.randint(0,780)

    plt.subplot(8,2,i+1)
    plt.imshow(X[x].reshape(128,128),'gray')
    plt.title('Real Image')
    plt.axis('off')

    plt.subplot(8,2,i+2)
    plt.imshow(y[x].reshape(128,128),'gray')
    plt.title('Mask Image')
```

```
    plt.axis('off')

    i += 2
plt.show()

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.05, random_state = 1)

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

from keras.layers import Input
from keras.layers import Conv2D
from keras.layers import Dropout
from keras.layers import Concatenate
from keras.layers import MaxPooling2D
from keras.layers import Conv2DTranspose
from keras.metrics import MeanIoU
from keras import Model

input = Input((128, 128, 1,))

conv1 = Conv2D(2**6, (3,3), activation = 'relu', padding = 'same')(input)
conv1 = Conv2D(2**6, (3,3), activation = 'relu', padding = 'same')(conv1)
pool1 = MaxPooling2D((2,2), strides = 2, padding = 'same')(conv1)
drop1 = Dropout(0.2)(pool1)

conv2 = Conv2D(2**7, (3,3), activation = 'relu', padding = 'same')(drop1)
conv2 = Conv2D(2**7, (3,3), activation = 'relu', padding = 'same')(conv2)
pool2 = MaxPooling2D((2,2), strides = 2, padding = 'same')(conv2)
drop2 = Dropout(0.2)(pool2)

conv3 = Conv2D(2**8, (3,3), activation = 'relu', padding = 'same')(drop2)
conv3 = Conv2D(2**8, (3,3), activation = 'relu', padding = 'same')(conv3)
pool3 = MaxPooling2D((2,2), strides = 2, padding = 'same')(conv3)
drop3 = Dropout(0.2)(pool3)

conv4 = Conv2D(2**9, (3,3), activation = 'relu', padding = 'same')(drop3)
```

```
conv4 = Conv2D(2**9, (3,3), activation = 'relu', padding = 'same')(conv4)
pool4 = MaxPooling2D((2,2), strides = 2, padding = 'same')(conv4)
drop4 = Dropout(0.2)(pool4)

convm = Conv2D(2**10, (3,3), activation = 'relu', padding = 'same')(drop4)
convm = Conv2D(2**10, (3,3), activation = 'relu', padding = 'same')(convm)

tran5 = Conv2DTranspose(2**9, (2,2), strides = 2, padding = 'valid', activation = 'relu')(convm)
conc5 = Concatenate()([tran5, conv4])
conv5 = Conv2D(2**9, (3,3), activation = 'relu', padding = 'same')(conc5)
conv5 = Conv2D(2**9, (3,3), activation = 'relu', padding = 'same')(conv5)
drop5 = Dropout(0.1)(conv5)

tran6 = Conv2DTranspose(2**8, (2,2), strides = 2, padding = 'valid', activation = 'relu')(drop5)
conc6 = Concatenate()([tran6, conv3])
conv6 = Conv2D(2**8, (3,3), activation = 'relu', padding = 'same')(conc6)
conv6 = Conv2D(2**8, (3,3), activation = 'relu', padding = 'same')(conv6)
drop6 = Dropout(0.1)(conv6)

tran7 = Conv2DTranspose(2**7, (2,2), strides = 2, padding = 'valid', activation = 'relu')(drop6)
conc7 = Concatenate()([tran7, conv2])
conv7 = Conv2D(2**7, (3,3), activation = 'relu', padding = 'same')(conc7)
conv7 = Conv2D(2**7, (3,3), activation = 'relu', padding = 'same')(conv7)
drop7 = Dropout(0.1)(conv7)

tran8 = Conv2DTranspose(2**6, (2,2), strides = 2, padding = 'valid', activation = 'relu')(drop7)
conc8 = Concatenate()([tran8, conv1])
conv8 = Conv2D(2**6, (3,3), activation = 'relu', padding = 'same')(conc8)
conv8 = Conv2D(2**6, (3,3), activation = 'relu', padding = 'same')(conv8)
drop8 = Dropout(0.1)(conv8)

output = Conv2D(2**0, (1,1), activation = 'relu', padding = 'same')(drop8)
model = Model(inputs = input, outputs = output, name = 'U-net')

from keras.utils.vis_utils import plot_model
plot_model(model, to_file='model.png')

#keras.utils.plot_model(model, './model_plot.png', show_shapes = True)

import tensorflow as tf
```

```
model.compile(loss = 'mean_squared_error', optimizer = keras.optimizers.Adam(learning_rate = 0.00005),metrics=[tf.keras.metrics.MeanIoU(num_classes=2),'accuracy'])
print(model.summary())

from keras.callbacks import ModelCheckpoint

checkp = ModelCheckpoint('/content/drive/MyDrive/results/unet.h5', monitor = 'val_loss', save_best_only = True, verbose = 1)

#history = model.fit(X_train, y_train, epochs = 200, batch_size = 64, validation_data = (X_test, y_test), callbacks = [checkp])

# plotting the loss vs epochs
plt.figure(figsize = (20,7))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['training loss', 'validation loss'])
plt.xlabel('Epochs')
plt.ylabel('Losses')
plt.title('Losses vs Epochs', fontsize = 15)

# plotting the accuracy vs epochs
plt.figure(figsize = (20,7))
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['Training accuracy', 'Validation Accuracy'])
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Accuracy vs Epochs', fontsize = 15)

from keras.models import load_model
model = load_model('/content/drive/MyDrive/results/unet.h5')
y_pred = model.predict(X_test)
print(y_pred.shape)

plt.figure(figsize = (20,80))

i = 0
x = 0
while i < 45 :
```

```
    plt.subplot(15,3,i+1)
    plt.imshow(X_test[x].reshape(128,128), 'gray')
    plt.title('Real medic Image')
    plt.axis('off')

    plt.subplot(15,3,i+2)
    plt.imshow(y_test[x].reshape(128,128), 'gray')
    plt.title('Ground Truth Img')
    plt.axis('off')

    plt.subplot(15,3,i+3)
    plt.imshow(y_pred[x].reshape(128,128), 'gray')
    plt.title('Predicited Image')
    plt.axis('off')

    x += 1
    i += 3
plt.show()
```

==========

Preprocessing and Classification using Machine Learning Algorithm

```
#!/usr/bin/env python
# coding: utf-8

# # libraries import

# In[5]:



import numpy as np
import cv2
import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In[ ]:

In[8]:

```
classes={'Benign':1,"Malignant":2,"Normal":0}
```

In[13]:

```
# Loaded data
training_images = []
grayscaled_images=[]

file_names = []
flags= []
ds =  []
labels=[]
for name in classes:
   address='C:/Users/lenovo/Documents/USDATASET/Training/' +name

   for add in os.listdir(address):

        image = cv2.imread(address+'/'+add)
        image = cv2.resize(image, (200,200))
            image = cv2.normalize(image, None, alpha = 0, beta = 255, norm_type = cv2.NORM_MINMAX,
dtype =cv2.CV_8U)
        file_names.append(image)
        flags.append(classes[name])
        labels.append(name)
```

In[14]:

```
print("length of data =",len(file_names), "files", len(flags), "labels",len(labels))


# In[15]:


labels = flags


# In[16]:


plt.bar([0,1,2], [len([i for i in flags if i == 1]),len([i for i in flags if i == 2]) ,len([i for i in flags if i == 0])], color =
['r', 'g','b'])
plt.xticks([0, 1,2], ['Beningn', 'Malignant','Normal'])
plt.show()


# In[63]:


def sample_images(images, gray = False):
    figure, axes = plt.subplots(1, 4)
    figure.set_size_inches(35,25)
            axes[0].imshow(images[1],cmap='gray') if gray else axes[0].imshow(cv2.cvtColor(images[1],
cv2.COLOR_BGR2RGB))
    axes[0].title.set_text("Benign")
    axes[0].axis('off')
           axes[1].imshow(images[55],cmap='gray') if gray else axes[1].imshow(cv2.cvtColor(images[55],
cv2.COLOR_BGR2RGB))
    axes[1].title.set_text("Benign")
    axes[1].axis('off')
          axes[2].imshow(images[450],cmap='gray') if gray else axes[2].imshow(cv2.cvtColor(images[450],
cv2.COLOR_BGR2RGB))
    axes[2].axis('off')
    axes[2].title.set_text("Malignant")
          axes[3].imshow(images[451],cmap='gray') if gray else axes[3].imshow(cv2.cvtColor(images[451],
cv2.COLOR_BGR2RGB))
    axes[3].axis('off')
```

```
    axes[3].title.set_text("Malignant")
    plt.show()


# In[ ]:




# # Histogram Function

# In[64]:


def hist(I):      #2D
    if len(I.shape)>2:
        I=cv2.cvtColor(I,cv2.COLOR_BGR2GRAY)
    H=np.zeros((256,1))
    for x in np.nditer(I):
        if x>255:
            x=255
        elif x<0:
            x=0
        H[int(x)]=H[int(x)]+1
    return H


# In[65]:


input_images = training_images
print(np.array(training_images).shape)
print(training_images[10].dtype)


# In[17]:


training_images=file_names
```

```
labels
(np.array(training_images)).shape


# In[18]:


#8bit per pixel
#u int8


# In[66]:


grayscaled_images.clear()
for idx, image in enumerate(training_images):
   r,g,b = cv2.split(image)
      gray_image = cv2.normalize(src=g, dst=None, alpha=0, beta=255, norm_type=cv2.NORM_MINMAX,
dtype=cv2.CV_8U)
   #gray_image=g/255
   grayscaled_images.append(gray_image)



# In[67]:


sample_images(grayscaled_images)


# In[69]:


fig = plt.figure(figsize=[25,10])
plt.subplot(1,4,1)
plt.plot(hist(grayscaled_images[1]))
plt.title('histogram for benign')
plt.subplot(1,4,2)
plt.plot(hist(grayscaled_images[50]))
plt.title('histogram for benign')
```

```
plt.subplot(1,4,3)
plt.plot(hist(grayscaled_images[450]))
plt.title('histogram for malignant')
plt.subplot(1,4,4)
plt.plot(hist(grayscaled_images[451]))
plt.title('histogram for malignant')


# # Histogram Equalization

# In[91]:


# histogram equalization on gray scale images
eqhist_images=[]
eqhist_images.clear()
for idx, image in enumerate(grayscaled_images):
    eqhistimage = cv2.equalizeHist(image)
    eqhist_images.append(eqhistimage)
sample_images(eqhist_images, True)
fig = plt.figure(figsize=[25,10])
plt.subplot(2,4,5)
plt.plot(hist(eqhist_images[1]))
plt.title('histogram for benign')
plt.subplot(2,4,6)
plt.plot(hist(eqhist_images[50]))
plt.title('histogram for benign')
plt.subplot(2,4,7)
plt.plot(hist(eqhist_images[450]))
plt.title('histogram for malignant')
plt.subplot(2,4,8)
plt.plot(hist(eqhist_images[451]))
plt.title('histogram for malignant')


# In[ ]:
```

In[93]:

```
hi_mean=[]
hi_median=[]
hi_std_dev=[]
for idx, image in enumerate(eqhist_images ):
    mean = np.mean(image)
    median = np.median(image)
    std_dev = np.std(image)
    hi_mean.append(mean)
    hi_median.append(median)
    hi_std_dev.append(std_dev)

print(hi_mean[0])
print(hi_median[0])
print(hi_std_dev[0])
```

In[94]:

CLAHE

In[105]:

```
Clahe_images=[]
Clahe_images.clear()
for idx, image in enumerate(grayscaled_images):
    #R, G, B = cv2.split(image)

    clahe = cv2.createCLAHE(clipLimit = 20)
    clahe_img = clahe.apply(image)

    image_histogram = cv2.merge((R,clahe_img,B))
```

```
    Clahe_images.append(clahe_img)
sample_images(Clahe_images, True)


# In[106]:


clahe_mean=[]
clahe_median=[]
clahe_std_dev=[]

clahe_mean.clear()
clahe_median.clear()
clahe_std_dev.clear()

for idx, image in enumerate(Clahe_images):
    mean = np.mean(image)
    median = np.median(image)
    std_dev = np.std(image)

    clahe_mean.append(mean)
    clahe_median.append(median)
    clahe_std_dev.append(std_dev)

print(clahe_mean[0])
print(clahe_median[0])
print(clahe_std_dev[0])


# In[114]:




# # Bilateral Filter

# In[70]:
```

```
bilateral_images = []
bilateral_images.clear()
for idx, image in enumerate(grayscaled_images):
    bilateral = cv2.bilateralFilter(image, 15, 75, 75)
    bilateral_images.append(bilateral)
sample_images(bilateral_images)


# In[71]:


fig = plt.figure(figsize=[25,10])
plt.subplot(1,4,1)
plt.plot(hist(bilateral_images[1]))
plt.title('histogram for benign')
plt.subplot(1,4,2)
plt.plot(hist(bilateral_images[50]))
plt.title('histogram for benign')
plt.subplot(1,4,3)
plt.plot(hist(bilateral_images[450]))
plt.title('histogram for malignant')
plt.subplot(1,4,4)
plt.plot(hist(bilateral_images[451]))
plt.title('histogram for malignant')


# In[ ]:




# # Adaptive_Threshold

# In[72]:


# Applied threshold to zero inversion
thresholded_images=[]
thresholded_images.clear()
```

```
for idx, image in enumerate(grayscaled_images):
    image = cv2.adaptiveThreshold(image, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY,
13, 5)
   thresholded_images.append(image)
sample_images(thresholded_images, True)


# # Sobel Edge Detection

# In[92]:


x_edged_images=[]
x_edged_images.clear()
for idx, image in enumerate(grayscaled_images):
   sobelx = cv2.Sobel(image,cv2.CV_64F,1,0,ksize=5)
   x_edged_images.append(sobelx)
sample_images(x_edged_images, True)


# In[ ]:




# In[97]:


canny_edged_images=[]
canny_edged_images.clear()
for idx, image in enumerate(grayscaled_images):
   image = cv2.Canny(image,30,200)
   canny_edged_images.append(image)
sample_images(canny_edged_images, True)


# In[98]:
```

```
canny_mean=[]
canny_median=[]
canny_std_dev=[]
canny_mean.clear()
canny_median.clear()
canny_std_dev.clear()

for idx, image in enumerate(canny_edged_images):
    mean = np.mean(image)
    median = np.median(image)
    std_dev = np.std(image)

    canny_mean.append(mean)
    canny_median.append(median)
    canny_std_dev.append(std_dev)

print(canny_mean[0], canny_mean[2])
print(canny_median[0], canny_median[2])
print(canny_std_dev[0], canny_std_dev[2])

print(canny_mean[17], canny_mean[12])
print(canny_median[17], canny_median[12])
print(canny_std_dev[17], canny_std_dev[12])


# # Laplacian

# In[147]:


L_edged_images=[]
L_edged_images.clear()
for idx, image in enumerate(grayscaled_images):
    image = cv2.Laplacian(img,cv2.CV_64F)
    L_edged_images.append(image)
sample_images(L_edged_images, True)


# In[148]:
```

```
L_mean=[]
L_median=[]
L_std_dev=[]
L_mean.clear()
L_median.clear()
L_std_dev.clear()
for idx, image in enumerate(ero_images ):
    mean = np.mean(image)
    median = np.median(image)
    std_dev = np.std(image)
    L_mean.append(mean)
    L_median.append(median)
    L_std_dev.append(std_dev)
print(L_mean[0])
print(L_median[0])
print(L_std_dev[0])


# In[ ]:




# In[75]:


# fig = plt.figure(figsize=[25,10])
# plt.subplot(1,4,1)
# plt.plot(hist(thresholded_images[1]))
# plt.title('histogram for benign')
# plt.subplot(1,4,2)
# plt.plot(hist(thresholded_images[50]))
# plt.title('histogram for benign')
# plt.subplot(1,4,3)
# plt.plot(hist(thresholded_images[450]))
# plt.title('histogram for malignant')
# plt.subplot(1,4,4)
# plt.plot(hist(thresholded_images[451]))
```

```
# plt.title('histogram for malignant')


# # EROSION

# In[142]:


ero_images = []
ero_images.clear()
kernel = np.ones((3,3),np.uint8)
for idx, image in enumerate(grayscaled_images):

    erosion = cv2.erode(image,kernel,iterations = 1)
    dilation = cv2.dilate(erosion,kernel,iterations = 1)
    erosion = cv2.erode(dilation,kernel,iterations = 2)
    #dilation = cv2.dilate(erosion,kernel,iterations = 1)
    #erosion = cv2.erode(dilation,kernel,iterations = 2)
    ero_images.append(erosion)
sample_images(ero_images, True)


# In[144]:


er_mean=[]
er_median=[]
er_std_dev=[]
er_mean.clear()
er_median.clear()
er_std_dev.clear()
for idx, image in enumerate(ero_images ):
    mean = np.mean(image)
    median = np.median(image)
    std_dev = np.std(image)
    er_mean.append(mean)
    er_median.append(median)
    er_std_dev.append(std_dev)
print(er_mean[1])
print(er_median[1])
```

```
print(er_std_dev[1])


# # Dilation

# In[120]:


dil_images = []
dil_images.clear()
kernel = np.ones((5,5),np.uint8)
for idx, image in enumerate(grayscaled_images):
    dilation = cv2.dilate(image,kernel,iterations = 1)
    dil_images.append(dilation)
sample_images(dil_images, True)


# In[121]:


dl_mean=[]
dl_median=[]
dl_std_dev=[]
for idx, image in enumerate(dil_images):
    mean = np.mean(image)
    median = np.median(image)
    std_dev = np.std(image)
    dl_mean.append(mean)
    dl_median.append(median)
    dl_std_dev.append(std_dev)
print(dl_mean[0])
print(dl_median[0])
print(dl_std_dev[0])


# In[84]:


threshold_mean=[]
threshold_median=[]
```

```
threshold_std_dev=[]
threshold_mean.clear()
threshold_median.clear()
threshold_std_dev.clear()

for idx, image in enumerate(thresholded_images):
    mean = np.mean(image)
    median = np.median(image)
    std_dev = np.std(image)

    threshold_mean.append(mean)
    threshold_median.append(median)
    threshold_std_dev.append(std_dev)

print(threshold_mean[0], threshold_mean[502])
print(threshold_median[0], threshold_median[502])
print(threshold_std_dev[0], threshold_std_dev[502])

print(threshold_mean[17], threshold_mean[12])
print(threshold_median[17], threshold_median[12])
print(threshold_std_dev[17], threshold_std_dev[12])


# In[ ]:




# In[20]:


df2 = pd.DataFrame({'label':flags})
print (df2)


# In[21]:


Energy = []
```

```
Correlation = []
Dissimilarity = []
Homogenity = []
Contrast = []
from skimage.feature import greycomatrix, greycoprops
def energy(img):

        img = np.array(img)
        GLCM = greycomatrix(img, [1], [0])



        return greycoprops(GLCM, 'energy')[0]

def correlation(img):
#
        img = np.array(img)
        GLCM = greycomatrix(img, [1], [0])



        return greycoprops(GLCM, 'correlation')[0]

def dissimilarity(img):

        img = np.array(img)
        GLCM = greycomatrix(img, [1], [0])

        return  greycoprops(GLCM, 'dissimilarity')[0]

def homogenity(img):

        img = np.array(img)
        GLCM = greycomatrix(img, [1], [0])



        return  greycoprops(GLCM, 'homogeneity')[0]

def contrast(img):

        img = np.array(img)
        GLCM = greycomatrix(img, [1], [0])
```

```
        return greycoprops(GLCM, 'contrast')[0]


for img in training_images:
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    Energy.append(energy(img)[0])
    Correlation.append(correlation(img)[0])
    Dissimilarity.append(dissimilarity(img)[0])
    Homogenity.append(homogenity(img)[0])
    Contrast.append(contrast(img)[0])


# In[168]:


a=pd.DataFrame({'threshold_mean':threshold_mean})


#b=threshold_median
b=pd.DataFrame({'threshold_std_dev':threshold_std_dev})


c=pd.DataFrame({'dl_mean':dl_mean})

#d=dl_median
d=pd.DataFrame({'dl_std_dev':dl_std_dev})




dfe = pd.DataFrame({'Energy':Energy})

dfco = pd.DataFrame({'Correlation':Correlation})

dfd = pd.DataFrame({' Dissimilarity': Dissimilarity})

dfh = pd.DataFrame({'Homogenity':Homogenity})
```

```
dfc = pd.DataFrame({'Contrast':Contrast})
```

```
# In[ ]:
```

```
# In[169]:
```

```
dfhi = pd.DataFrame({'hi_mean':hi_mean})
dfhi_std_dev = pd.DataFrame({'hi_std_dev':hi_std_dev})
```

```
# In[170]:
```

```
pdList = [a,b,c,d,dfe,dfc,dfd,dfco,dfh,dfhi,dfhi_std_dev,
        df2]
new_df = pd.concat(pdList,axis=1)
```

```
# In[171]:
```

```
new_df
```

```
# In[237]:
```

```
new_df.head()
```

```
# In[238]:
```

```
new_df.info()


# In[239]:


new_df.describe()


# In[250]:


#Let us first analyze the distribution of the target variable
target='label'
labels=['Benign','Malignant','Normal']
features = [i for i in new_df.columns.values if i not in [target]]
MAP={}
for e, i in enumerate(sorted(new_df[target].unique())):
    MAP[i]=labels[e]
#MAP={0:'Not-Survived',1:'Survived'}
data1 = new_df.copy()
data1[target]=data1[target].map(MAP)
explode=np.zeros(len(labels))
explode[-1]=0.1
print('\033[1mTarget Variable Distribution'.center(55))
plt.pie(data1[target].value_counts(),     labels=new_df[target].value_counts().index,     counterclock=False,
shadow=True,
        explode=explode, autopct='%1.1f%%', radius=1, startangle=0)
plt.show()


# In[251]:


#Check for empty elements
nvc = pd.DataFrame(data1.isnull().sum().sort_values(), columns=['Total Null Values'])
nvc['Percentage'] = round(nvc['Total Null Values']/data1.shape[0],3)*100
print(nvc)
```

```
# In[257]:


cf = []

nu = new_df[features].nunique().sort_values()


for i in range(new_df[features].shape[1]):
    if nu.values[i]<=7:
        cf.append(nu.index[i])
    else: nf.append(nu.index[i])


# In[260]:


import math
print('\033[1mFeatures Distribution'.center(100))

n=6
nf = [i for i in features if i not in cf]

plt.figure(figsize=[15,3*math.ceil(len(features)/n)])
for c in range(len(nf)):
    plt.subplot(math.ceil(len(features)/n),n,c+1)
    sns.distplot(new_df[nf[c]])
plt.tight_layout()
plt.show()

plt.figure(figsize=[15,3*math.ceil(len(features)/n)])
for c in range(len(nf)):
    plt.subplot(math.ceil(len(features)/n),n,c+1)
    new_df.boxplot(nf[c])
plt.tight_layout()
plt.show()


# In[264]:
```

```
## Removal of outlier:
data2 = data1.copy()

for i in [i for i in data2.columns]:
    if data2[i].nunique()>=12:
        Q1 = data2[i].quantile(0.15)
        Q3 = data2[i].quantile(0.85)
        IQR = Q3 - Q1
        data2 = data2[data2[i] <= (Q3+(1.5*IQR))]
        data2 = data2[data2[i] >= (Q1-(1.5*IQR))]
data2 = data2.reset_index(drop=True)
display(data2.head())
print('\n\033[1mInference:\033[0m Before removal of outliers, The dataset had {} samples.'.format(data1.shape[0]))
print('\033[1mInference:\033[0m After removal of outliers, The dataset now has {} samples.'.format(data2.shape[0]))


# In[266]:


#Splitting the data intro training & testing sets
data = data2.copy()
X = data.drop([target],axis=1)
Y = data[target]
x_train, x_test, y_train, y_test = train_test_split(X, Y, train_size=0.8, test_size=0.2, random_state=0)

print('Original set  ---> ',X.shape,Y.shape,'\nTraining set  ---> ',x_train.shape,y_train.shape,'\nTesting set ---> ', x_test.shape,'', y_test.shape)


# In[272]:


# Feature Scaling (Standardization)
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
import seaborn as sns
```

```
std = StandardScaler()
print('\033[1mStandardardization on Training set'.center(100))
x_train = std.fit_transform(x_train)
X_train = pd.DataFrame(x_train, columns=X.columns)
display(X_train.describe())

print('\n','\033[1mStandardardization on Testing set'.center(100))
x_test = std.transform(x_test)
X_test = pd.DataFrame(x_test, columns=X.columns)
display(X_test.describe())


# In[273]:


## Applying Logistic Regression Algorithm
from sklearn.linear_model import LogisticRegression
model1 = LogisticRegression(random_state=0)
model1.fit(x_train,y_train)
y_pred1 = model1.predict(x_test)
test_accuracy1 = round(accuracy_score(y_test, y_pred1) * 100, 2)
print('Test Accuracy Score of Basic Logistics Regression:',test_accuracy1)
conf1 = confusion_matrix(y_test, y_pred1)
sns.heatmap(conf1, annot=True)
print(classification_report(y_test,y_pred1))


# In[276]:


## Applying Random Forest Algorithm
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators =100)
model.fit(x_train,y_train)
y_pred2 = model.predict(x_test)
print("Test Accuracy is :",accuracy_score(y_pred2,y_test)*100,'%')
conf2 = confusion_matrix(y_test, y_pred2)
sns.heatmap(conf2, annot=True)
print(classification_report(y_test,y_pred2))
```

In[278]:

Applying KNN Algorithm

```
from sklearn.neighbors import KNeighborsClassifier
model3 = KNeighborsClassifier(n_neighbors=9 , metric= 'minkowski' , p = 4)
model3.fit(x_train,y_train)
y_pred3 = model3.predict(x_test)
test_accuracy3 = round(accuracy_score(y_test, y_pred3) * 100, 2)
print('Test Accuracy Score of KNN:',test_accuracy3)
conf3 = confusion_matrix(y_test, y_pred3)
sns.heatmap(conf3, annot=True)
print(classification_report(y_test,y_pred3))
```

In[279]:

```
from sklearn.svm import SVC, LinearSVC
model4 = SVC()
model4.fit(x_train,y_train)
y_pred4 = model4.predict(x_test)
test_accuracy4 = round(accuracy_score(y_test, y_pred4) * 100, 2)
print('Test Accuracy Score of SVM :',test_accuracy4)
conf4 = confusion_matrix(y_test, y_pred4)
sns.heatmap(conf4, annot=True)
print(classification_report(y_test,y_pred4))
```

In[280]:

```
from sklearn.tree import DecisionTreeClassifier
model5 = DecisionTreeClassifier()
model5.fit(x_train,y_train)
y_pred5 = model5.predict(x_test)
test_accuracy5 = round(accuracy_score(y_test, y_pred5) * 100, 2)
print('Test Accuracy Score of Decision Tree is :',test_accuracy5)
conf5 = confusion_matrix(y_test, y_pred5)
```

```
sns.heatmap(conf5, annot=True)
print(classification_report(y_test,y_pred5))


# In[281]:


## Applying Naive Bayes Algorithm
from sklearn.naive_bayes import GaussianNB
model6 = GaussianNB()
model6.fit(x_train,y_train)
y_pred6 = model6.predict(x_test)
test_accuracy6 = round(accuracy_score(y_test, y_pred6) * 100, 2)
print('Test Accuracy Score of Naive Bayes:',test_accuracy6)
conf6 = confusion_matrix(y_test, y_pred6)
sns.heatmap(conf6, annot=True)
print(classification_report(y_test,y_pred6))









# # ====================THE  END=======================
```

##==================THE END=================##