# Predicting Breast Cancer Based On Optimized Deep Learning Technique

**PROJECT REPORT**

*Submitted by*

**VAISHNAVI G**

# TABLE OF CONTENTS

Catalog

# 1. ABSTRACT

Breast cancer is one of the most prevalent and potentially life-threatening diseases affecting women worldwide. Early detection and accurate diagnosis play a crucial role in improving survival rates and treatment outcomes. Deep learning techniques have shown promising results in various medical applications, including breast cancer prediction. However, there is a need to optimize these techniques to enhance their performance and reliability.

This project aims to develop an optimized deep learning model for predicting breast cancer, leveraging advanced techniques to improve accuracy and reduce false positives/negatives. The existing methods for breast cancer prediction rely on conventional machine learning algorithms and may have limitations in terms of accuracy and efficiency.

Deep learning, particularly convolutional neural networks (CNNs), has demonstrated remarkable capabilities in image analysis tasks, making it a potential solution for breast cancer prediction. However, optimizing the deep learning models is essential to overcome challenges such as feature extraction, data imbalance, and interpretability.

This project also focuses on developing an optimized deep learning technique that addresses these challenges and improves breast cancer prediction accuracy.

I. We will be utilizing a data set consisting of 2,50,000 mount slide images of breast cancer specimens
II. Clean and Sanitize the dataset .
III. Preprocess the data to perform data partitioning and handle missing values.
IV. Create training and testing sets.
V. Build a classifier and fit the data to the model.
VI. Check the accuracy of the model and measure its effectiveness.

# 2. SOFTWARE REQUIREMENTS SPECIFICATIONS

**System configurations**

The software requirement specification can produce at the culmination of the analysis task. The function and performance allocated to software as part of system engineering are refined by established a complete information description, a detailed functional description, a representation of system behavior, and indication of performance and design constrain, appropriate validate criteria, and other information pertinent to requirements.

**Software Requirements**

- ✓ Operating system : Windows 10
- ✓ Coding Language : Python
- ✓ Google Research Collab , Jupyter Notebook
- ✓ Library : TensorFlow , Keras, Falsk, Pandas , NumPy

**Hardware Requirement**

- ✓ System : Intel I5 Processor
- ✓ Hard Disk : 1TB.
- ✓ RAM : 8GB.

# 3. ANALYSIS AND REQUIREMENT

## Data Availability and Quality

One of the crucial requirements for developing an effective deep learning model is the availability of large-scale, high-quality datasets. To accurately predict breast cancer risk, the dataset should include diverse patient demographics, genetic information, family history, lifestyle factors, and clinical parameters. Access to such comprehensive datasets, with a significant number of positive and negative cases, is essential for training and validating the deep learning model.

## Handling Imbalanced Data

Breast cancer datasets often suffer from class imbalance, where the number of samples in the minority class (cases with breast cancer) is significantly smaller than the majority class (healthy cases). Imbalanced data can lead to biased models and reduced predictive performance. Therefore, effective techniques such as oversampling, undersampling, or generating synthetic data should be employed to address the class imbalance problem and ensure fair representation of both positive and negative cases.

## Model Training and Validation

Deep learning models require substantial computational resources and time for training. Adequate computational infrastructure and optimized training procedures are necessary to handle the complexity of the models and ensure timely development. Additionally, rigorous validation protocols should be employed to assess the generalization performance of the model and its ability to accurately predict breast cancer risk in unseen patient cases.

## Ethical Considerations and Privacy

Breast cancer risk prediction involves handling sensitive patient data, including personal information and medical records. It is crucial to adhere to strict privacy and ethical guidelines to protect patient confidentiality and comply with legal regulations. Robust data anonymization and secure storage practices should be implemented to ensure data privacy and mitigate potential risks.

## Clinical Integration and Usability

To make deep learning techniques for breast cancer risk prediction clinically applicable, integration with existing healthcare systems and workflows is necessary. The developed model should be compatible with standard electronic health record (EHR) systems and provide user-friendly interfaces for clinicians to access and interpret the risk predictions seamlessly.

## Continuous Model Improvement

Deep learning models for breast cancer risk prediction should be designed with the flexibility to incorporate new data and adapt to evolving medical knowledge. Regular updates and refinements of the model should be planned to ensure its continued accuracy and relevance as new research findings and risk factors

## Model Performance Evaluation

The performance of the deep learning model for breast cancer risk prediction should be evaluated using appropriate metrics such as accuracy, sensitivity, specificity, area under the curve (AUC), and calibration metrics. Comparative analysis against existing risk prediction methods should be conducted to demonstrate the superiority of the deep learning techniques in terms of accuracy and clinical relevance.

# 4. PROBLEM DESCRIPTION

Breast cancer is one of the most prevalent and potentially life-threatening diseases affecting women worldwide. Early detection and accurate diagnosis play a crucial role in improving survival rates and treatment outcomes. Deep learning techniques have shown promising results in various medical applications, including breast cancer prediction. However, there is a need to optimize these techniques to enhance their performance and reliability.

This project aims to develop an optimized deep learning model for predicting breast cancer, leveraging advanced techniques to improve accuracy and reduce false positives/negatives.

The existing methods for breast cancer prediction rely on conventional machine learning algorithms and may have limitations in terms of accuracy and efficiency. Deep learning, particularly convolutional neural networks (CNNs), has demonstrated remarkable capabilities in image analysis tasks, making it a potential solution for breast cancer prediction.

However, optimizing the deep learning models is essential to overcome challenges such as feature extraction, data imbalance, and interpretability. This project focuses on developing an optimized deep learning technique that addresses these challenges and improves breast cancer prediction accuracy.

# 5. TOOLS AND TECHNOLOGIES

## 5.1 Python

Python is a high-level, general-purpose and a very popular programming language. Python programming language (latest Python 3) is being used in web development, Machine Learning applications, along with all cutting edge technology in Software Industry.  We will be utilizing the below standard library in Python for creating a classification model and for analysing the model.

### 5.1.1 Anaconda Navigator

Anaconda Navigator is a free and open-source distribution of the Python and R programming languages for data science and machine learning related applications. It can be installed on Windows, Linux, and macOS.Conda is an open-source, cross-platform,  package management system. Anaconda comes with so very nice tools like JupyterLab, Jupyter Notebook, QtConsole, Spyder, Glueviz, Orange, Rstudio, Visual Studio Code.

### 5.1.2 PANDAS

Pandas stand for "Python Data Analysis Library" which is the popular Python library that is mainly used for data processing purposes like cleaning, manipulation, and analysis.It consists of classes to read, process, and write CSV data files.

There are numerous Data cleaning tools present but, the Pandas library provides a really fast and efficient way to manage and explore data. It does that by providing us with Series and DataFrames, which help us not only to represent data efficiently but also manipulate it in various ways.

### 5.1.3 Tensor flow

TensorFlow is an end-to-end open-source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art in ML and developers can easily build and deploy ML-powered applications.

### 5.1.4 Keras

Keras leverages various optimization techniques to make high-level neural network API easier and more performant.

It supports the following features:

- Consistent, simple, and extensible API.
- Minimal structure - easy to achieve the result without any frills.
- It supports multiple platforms and backends.
- It is a user-friendly framework that runs on both CPU and GPU.
- Highly scalability of computation.

### 5.1.5 Flask

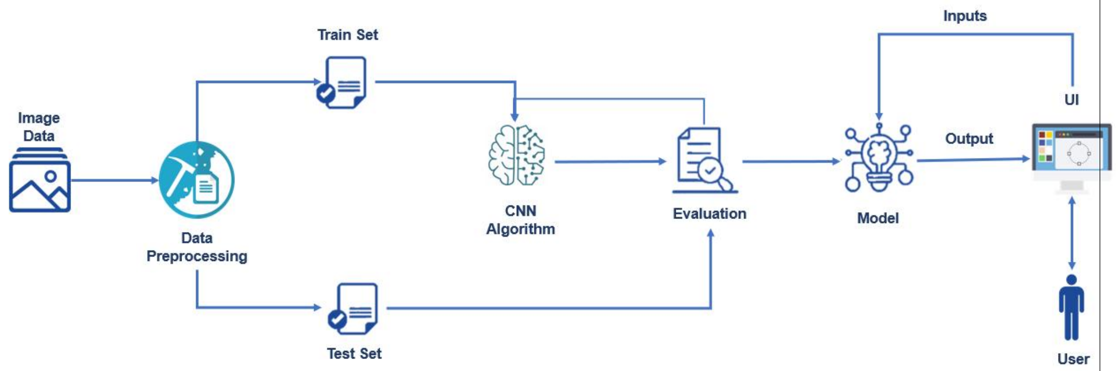
Flask is a popular web framework for building web applications using Python. It provides a simple and lightweight approach to develop web applications, APIs, and microservices.

# 6. DESIGN

**Data set analysis and design**

The dataset provides a collection of breast scan images and there are two classes of images people having breast cancer( represented with folder 1)  people not having a Breast cancer(represented with 0).In our project according to project structure, we will be creating train test folders and in them, place "0" folder images in benign and place "1" folder images in Malignant in train and test folders respectively.

1. Steps in Data Preprocessing
   a. Data Integration
   b. Data Cleaning
   c. Data Transformation
2. Data Reduction or Dimension Reduction
3. Data pre-processing of structured data in machine learning
4. **4.** Machine Learning for Data Analytics



**Data pre processing with Python - Commands**

Different data set types with python

1. Missing values
2. Outliers

3. Overfitting
4. Data with no numerical values
5. Different date formats

**Missing values**

- ✓ Missing values are a common problem while dealing with data! The values can be missed because of various reasons such as human errors, mechanical errors, etc.
- ✓ There are three techniques to solve the missing values' problem in order to find out the most accurate features, and they are:
    1. Dropping
    2. Numerical imputation
    3. Categorical imputation

✓ Dropping is the most common method to take care of the missed values. Those rows in the data set or the entire columns with missed values are dropped in order to avoid errors to occur in data analysis.

```
#Dropping columns in the data higher than 60% threshold
data = data[data.columns[data.isnull().mean() < threshold]]

#Dropping rows in the data higher than 60% threshold
data = data.loc[data.isnull().mean(axis=1) < threshold]
```

✓ Numerical imputation - The word imputation implies replacing the missing values with such a value that makes sense. And, numerical imputation is done in the data with numbers.

```
#For filling all the missed values as 0
data = data.fillna(0)

#For replacing missed values with median of columns
data = data.fillna(data.median())
```

✓ Categorical imputation - This technique of imputation is nothing but replacing the missed values in the data with the one which occurs the maximum number of times in the column. But, in case there is no such value that occurs frequently or dominates the other values, then it is best to fill the same as “NAN”.

```
#Categorical imputation
data['column_name'].fillna(data['column_name'].value_counts().idxmax(),
inplace=True)
```

**Outliers -**

✓ An outlier differs significantly from other values and is too distanced from the mean of the values. Such values that are considered outliers are usually due to some systematic errors or flaws.

```
#For identifying the outliers with the standard deviation method
outliers = [x for x in data if x < lower or x > upper]
print('Identified outliers: %d' % len(outliers))

#Remove outliers
outliers_removed = [x for x in data if x >= lower and x <= upper]
print('Non-outlier observations: %d' % len(outliers_removed))
```

In the codes above, “lower” and “upper” signify the upper and lower limit in the data set.

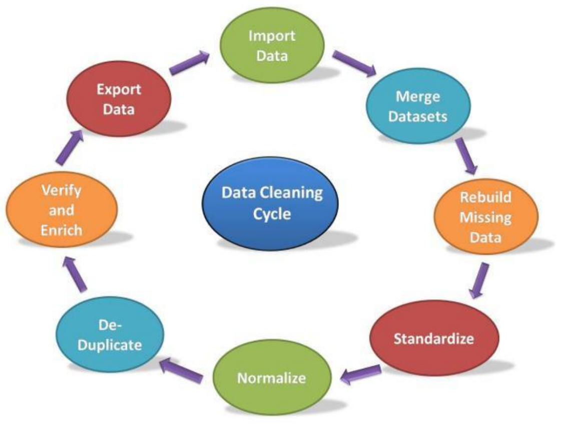
**Overfitting -**

✓ Overfitting occurs when the model fits the data too well or simply put when the model is too complex. Overfitting model learns the detail and noise in the training data to such an extent that it negatively impacts the performance of the model on new data/test data.

```
data['bin'] = pd.cut(data['value'], bins=[100,250,400,500], labels=["Lowest",
"Mid", "High"])
```

## Data Cleaning

- ✓ When working with multiple data sources, there are many chances for data to be incorrect, duplicated, or mislabeled. If data is wrong, outcomes and algorithms are unreliable, even though they may look correct.
- ✓ Data cleaning is the process of changing or eliminating garbage, incorrect, duplicate, corrupted, or incomplete data in a dataset.



## Data cleaning cycle

- ✓ It is the method of analyzing, distinguishing, and correcting untidy, raw data. Data cleaning involves filling in missing values, distinguish and fix errors present in the dataset. Whereas the techniques used for data cleaning might vary in step with different types of datasets, the following are standard steps to map out data cleaning:

## Data cleaning with Pandas

- ✓ Pandas stand for "Python Data Analysis Library" .
- ✓ Pandas is the popular Python library that is mainly used for data processing purposes like cleaning, manipulation, and analysis.
- ✓ It consists of classes to read, process, and write CSV data files.

```
#importing module
import pandas as pd
```

- ✓ To import the dataset we use the **read_csv()** function of pandas and store it in the

DataFrame named as data.

- ✓ As the dataset is in tabular format, when working with tabular data in Pandas it will be automatically converted in a **DataFrame**.
- ✓ DataFrame is a two-dimensional, mutable data structure in Python. It is a combination of rows and columns like an excel sheet.

```
#importing the dataset by reading the csv file
data = pd.read_csv('Iris.csv')

#displaying the first five rows of dataset
print(data.head())
```

- ✓ The head() function is a built-in function in pandas for the dataframe used to display the rows of the dataset.
- ✓ zIf we want to see the last five rows of the dataset we use the tail()function of the dataframe like this:

```
#displayinf last five rows of dataset
data.tail()
```

**Rebuild Missing Data**

To find and fill the missing data in the dataset we will use another function. There are 4 ways to find the null values if present in the dataset.

1. Using isnull() function:

```
data.isnull()
```

This function provides the boolean value for the complete dataset to know if any null value is present or not.

2. Using isna() function:

```
data.isna()
```

This is the same as the isnull() function. Ans provides the same output.

3. Using isna().any()

```
data.isna().any()
```

This function also gives a boolean value if any null value is present or not, but it gives results column-wise, not in tabular format.

4. Using isna(). sum()

```
data.isna().sum()
```

This function gives the sum of the null values preset in the dataset column-wise.

5. Using isna().any().sum()

```
data.isna().any().sum()
```

This function gives output in a single value if any null is present or not.

**Standardization and Normalization**

- ✓ Standardization is another scaling technique where the values are centered around the mean with a unit standard deviation. This means that the mean of the attribute becomes zero and the resultant distribution has a unit standard deviation.
- ✓ Normalization is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1. It is also known as Min-Max scaling.

**De-Duplicate**

- ✓ De-Duplicate means remove all duplicate values. There is no need for duplicate values in data analysis. These values only affect the accuracy and efficiency of the analysis result.
- ✓ To find duplicate values in the dataset we will use a simple dataframe function i.e. duplicated().

```
data.duplicated()
```

**Export Dataset**

This is the last step of the data cleaning process. After performing all the above operations, the data is transformed into clean the dataset and it is ready to export for the next process in Data Science or Data Analysis.

# 7. IMPLEMENTATION

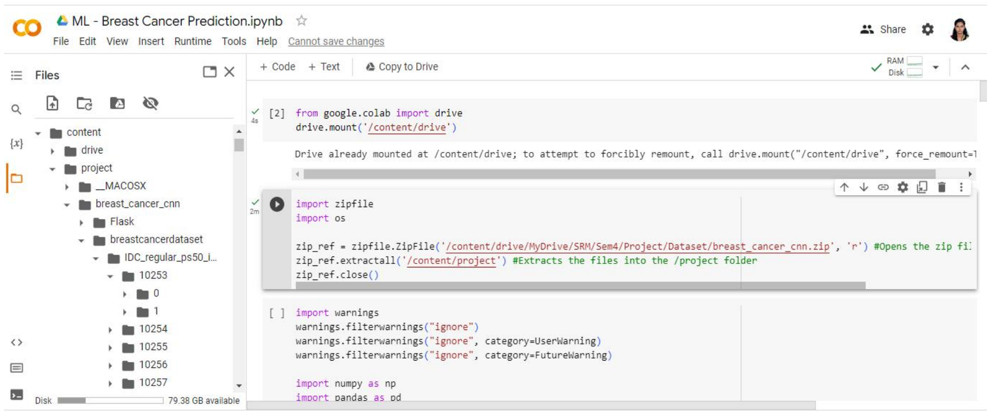
**Dataset :** **https://www.kaggle.com/datasets/paultimothymooney/breast-histopathology-images**

**Google collab playground:**

**https://colab.research.google.com/drive/11Gigf9iVWNti1sRL_Xzo4RNklIkbbfcH**

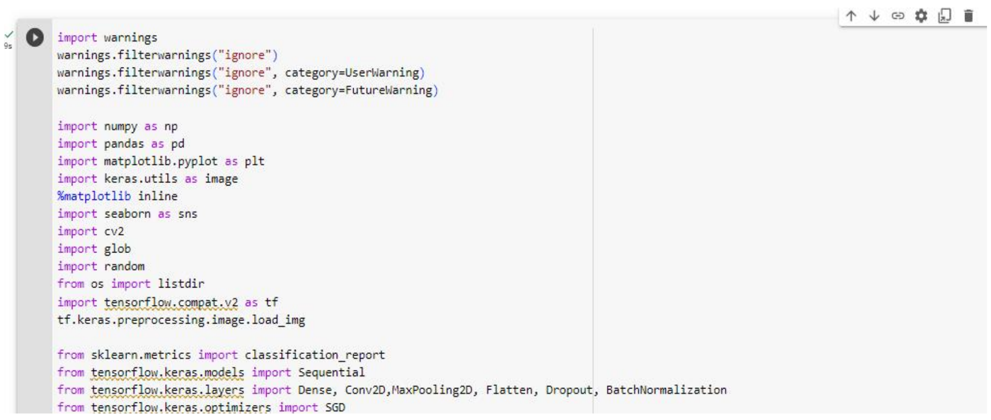
## 1. Import libraries and load data file

Upload the dataset in colab using google drive import ( Dataset can be reimported from gdrive easily if the environment gets refreshed)



```
[2] from google.colab import drive
    drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=1

import zipfile
import os

zip_ref = zipfile.ZipFile('/content/drive/MyDrive/SRM/Sem4/Project/Dataset/breast_cancer_cnn.zip', 'r') #Opens the zip fi
zip_ref.extractall('/content/project') #Extracts the files into the /project folder
zip_ref.close()

[ ] import warnings
    warnings.filterwarnings("ignore")
    warnings.filterwarnings("ignore", category=UserWarning)
    warnings.filterwarnings("ignore", category=FutureWarning)

    import numpy as np
    import pandas as pd
```

➢ Import required libraries in python:



```
import warnings
warnings.filterwarnings("ignore")
warnings.filterwarnings("ignore", category=UserWarning)
warnings.filterwarnings("ignore", category=FutureWarning)

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import keras.utils as image
%matplotlib inline
import seaborn as sns
import cv2
import glob
import random
from os import listdir
import tensorflow.compat.v2 as tf
tf.keras.preprocessing.image.load_img

from sklearn.metrics import classification_report
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D,MaxPooling2D, Flatten, Dropout, BatchNormalization
from tensorflow.keras.optimizers import SGD
```

## 2. Search for breast cancer images and assign it for processing

```
breast_img =
glob.glob('/tmp/breast_cancer_cnn/breastcancerdataset/IDC_regular_ps50_idx5/**/*.png',
recursive = True)
```

The glob module, which is short for global, is a function that's used to search for files that match a specific file pattern or name.

**3. Segregating images based on 'Negative' and 'Positive' scenario**

```
N_IDC = []
P_IDC = []

for img in breast_img:
  if img[-5] == '0' :
    N_IDC.append(img)

  elif img[-5] == '1' :
    P_IDC.append(img)
```

**4. Data visualization using MatPlotLib**

```
from keras.preprocessing import image
plt.figure(figsize = (15, 15))
some_non = np.random.randint(0, len(N_IDC), 18)
some_can = np.random.randint(0, len(P_IDC), 18)
s = 0
for num in some_non:
    img = tf.keras.utils.load_img((N_IDC[num]), target_size=(100, 100))
    #img = image.load_img((N_IDC[num]), target_size=(150,150))
    img = tf.keras.utils.img_to_array(img)
    plt.subplot(6, 6, 2*s+1)
    plt.axis('off')
    plt.title('no cancer')
    plt.imshow(img.astype('uint8'))
    s += 1
    s = 1
```

```
for num in some_can:
img = tf.keras.utils.load_img((P_IDC[num]), target_size=(100, 100))
img = tf.keras.utils.img_to_array(img)
plt.subplot(6, 6, 2*s)
plt.axis('off')
plt.title('IDC (+)')
plt.imshow(img.astype('uint8'))
s += 1
```

The "**plt.figure(figsize=(15, 15))**" command is used to set the size of a plot in Matplotlib, a popular Python library used for data visualization. This command is a helpful tool to customize the size of your plots and improve the visual representation of your data.

The **"np.random.randint(0, len(N_IDC), 18)"** is a command that generates a numpy array of 18 random integer numbers within the range of 0 and the length of the N_IDC list. random is a submodule of NumPy that contains functions for generating random numbers. randint is a function that generates random integers within a given range.

The**" tf.keras.utils.load_img((N_IDC[num]), target_size=(100, 100))"** command loads an image file at the specified path and resizes it to a fixed size, which can then be used as input to a machine learning model..

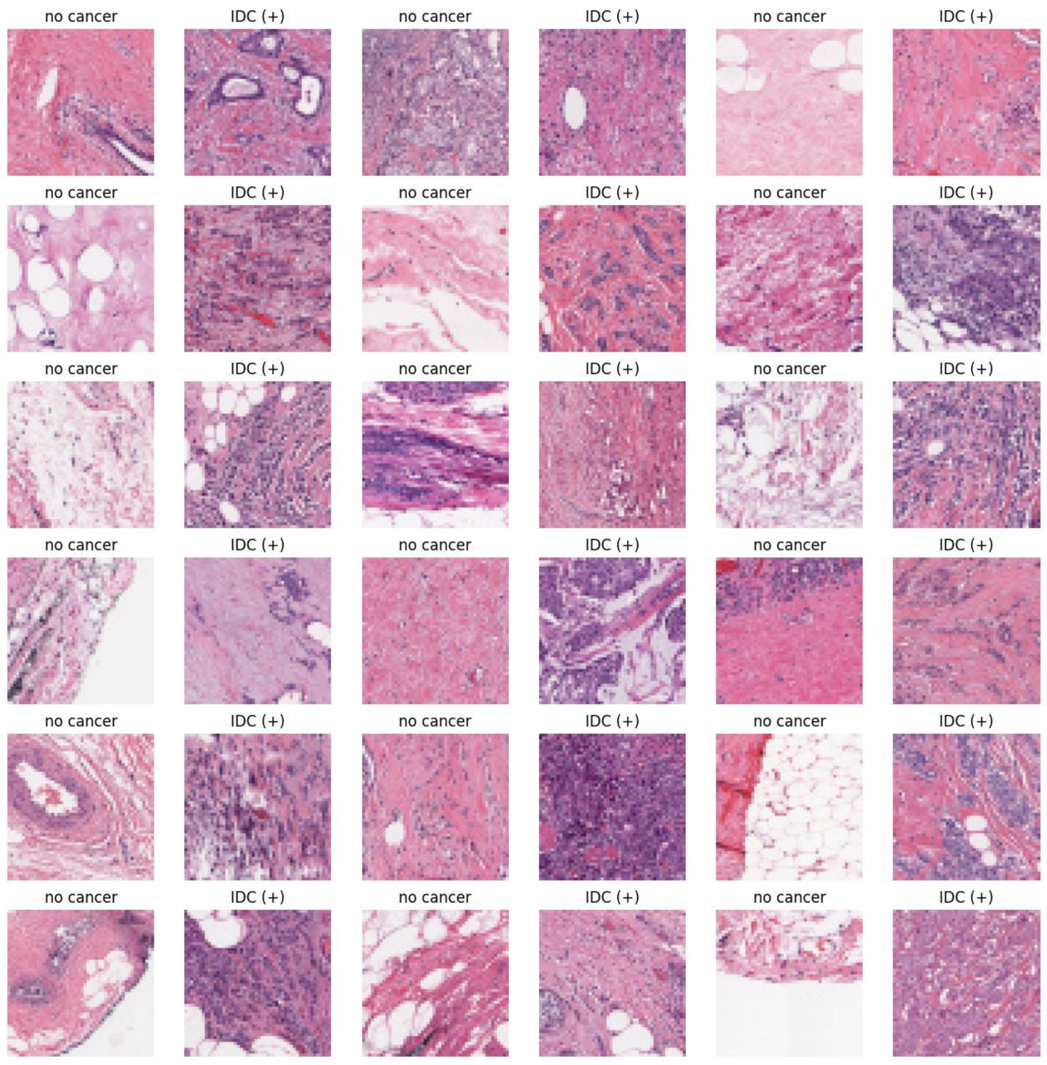
The **plt.subplot()** is a very useful function for creating and arranging multiple subplots within a single figure.

The **plt.axis('off')** is a function from the matplotlib.pyplot module that turns off the axis lines and labels in a plot.

The **plt.title()** function is used to add a title to a plot and it accepts a string argument that specifies the text to be displayed as the title.

The **plt.imshow()** function is used to display 2D arrays and images, and it accepts a numpy array or a PIL image instance as an argument. **img** is a numpy array that represents an image, and the **.astype('uint8')** method is used to convert the data type of the array to 8-bit unsigned integer format, which is a common format for image data.

**Output:**



## 5. Process image using OpenCV

```
non_img_arr = []
can_img_arr = []


for img in N_IDC:


    n_img = cv2.imread(img, cv2.IMREAD_COLOR)
    n_img_size = cv2.resize(n_img, (50, 50), interpolation = cv2.INTER_LINEAR)
    non_img_arr.append([n_img_size, 0])


for img in P_IDC:
    c_img = cv2.imread(img, cv2.IMREAD_COLOR)
    c_img_size = cv2.resize(c_img, (50, 50), interpolation = cv2.INTER_LINEAR)
    can_img_arr.append([c_img_size, 1])
```

**cv2.imread:** This is a function from the OpenCV library that is used to read an image from a file.

**cv2.IMREAD_COLOR:** This is a flag that tells OpenCV to read the image in color mode (as opposed to grayscale or alpha mode).

The **cv2.resize** is a method used in OpenCV for resizing images. It allows you to adjust the size of an image to a specified width and height. The method can also be used to scale images up or down, and it supports multiple interpolation methods for resizing the image.

The **"append()"** method is a built-in Python function that is used to add a new element to the end of a list.

## 6. Slicing dataset images

```
can_img_arr2=can_img_arr[0:50000]
non_img_arr2=non_img_arr[0:50000]
```

**can_img_arr2** will contain the first 50,000 elements of **can_img_arr**. This slicing operation can be useful when working with large datasets, as it enables you to work with smaller subsets of data at a time.

## 7. Shuffle dataset images

```
X = []
y = []

breast_img_arr = np.concatenate((non_img_arr2, can_img_arr2))
random.shuffle(breast_img_arr)

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

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

The **np.concatenate()** function is used to join two or more arrays along a specified axis. It is concatenating the non_img_arr2 and can_img_arr2 arrays along the first (default) axis, which is the vertical axis.

The **random.shuffle()** is a function takes a sequence (such as a list or tuple) and shuffles its elements randomly.

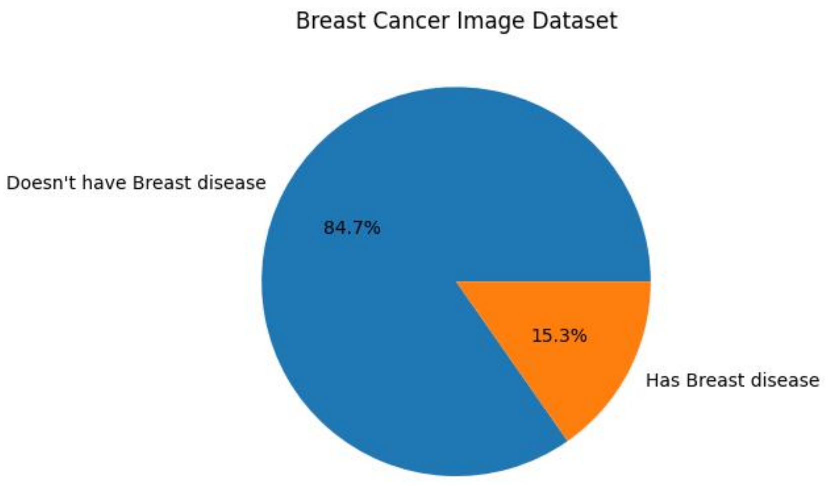
**8. Visualizing the data using bar and pie chart**

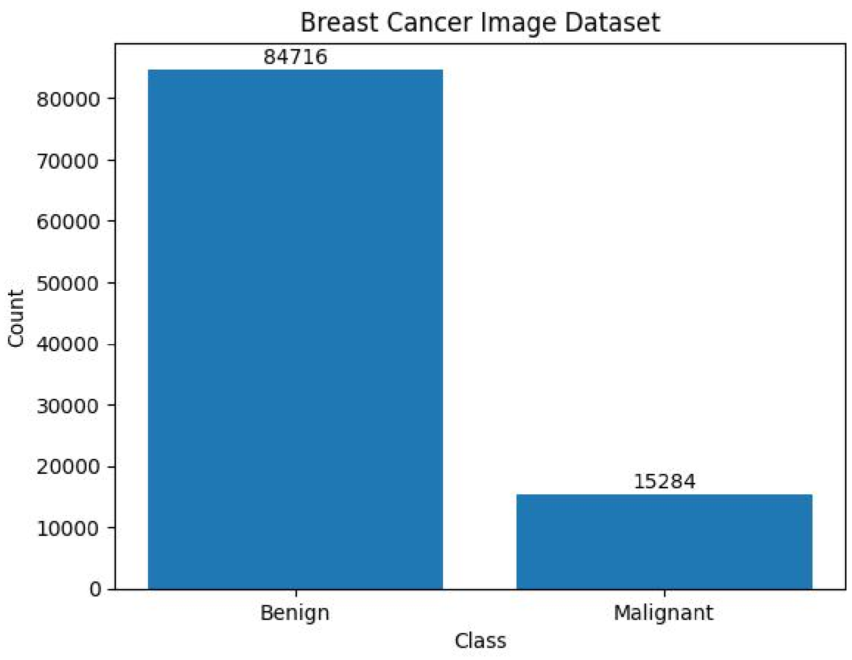
```
import os
def describeData(a,b):
    print('Total number of images: {}'.format(len(a)))
    print('Number of IDC(-) Images: {}'.format(np.sum(b==0)))
    print('Number of IDC(+) Images: {}'.format(np.sum(b==1)))
    print('Image shape (Width, Height, Channels): {}'.format(a[0].shape))
describeData(X,y)
benign_count = np.sum(y==0)
malignant_count =  np.sum(y==1)
counts = [benign_count, malignant_count]
labels = ['Doesn\'t have Breast disease', 'Has Breast disease']
plt.pie(counts, labels=labels, autopct='%1.1f%%')
# plt.xlabel('Class')
# plt.ylabel('Count')
plt.title('Breast Cancer Image Dataset')
plt.show()
counts = [benign_count, malignant_count]
labels = ['Benign', 'Malignant']
```

```
plt.bar(labels, counts)
plt.title('Breast Cancer Image Dataset')
plt.xlabel('Class')
plt.ylabel('Count')

for i, v in enumerate(counts):
    plt.text(i, v+1000, str(v), ha='center')

plt.show()
```

**Output:**

```
Total number of images: 100000
Number of IDC(-) Images: 84716
Number of IDC(+) Images: 15284
Image shape (Width, Height, Channels): (50, 50, 3)
```





**9. Train data**

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.3)

from tensorflow.keras.utils import to_categorical
Y_train = to_categorical(Y_train, num_classes = 2)
Y_test = to_categorical(Y_test, num_classes = 2)

print("Training Data Shape:", X_train.shape)
print("Testing Data Shape:", X_test.shape)
```

**Output:**

```
Training Data Shape: (70000, 50, 50, 3)
Testing Data Shape: (30000, 50, 50, 3)
```

**from tensorflow.keras.utils import to_categorical -** To import the to_categorical function from the keras.utils module in TensorFlow. This function is commonly used to convert class vectors (integers) to binary class matrix.

The **"to_categorical()"** is a function takes an array or a list of integers as input, and returns a matrix where each row corresponds to one of the input integers and each column corresponds to a possible value of the categorical variable.

**10. Reduce training and testing dataset for faster processing**

```
X_train = X_train[0:50000]
Y_train = Y_train[0:50000]
X_test = X_test[0:30000]
Y_test = Y_test[0:30000]
```

Above code is reducing the size of the datasets X_train, Y_train, X_test, and Y_test by keeping only the specified number of elements at the beginning of each dataset. The purpose of this code is to create smaller training and testing sets for faster processing or to work with a limited amount of data for experimentation or debugging purposes.

**11. Importing optimizers, metrics, callbacks, and functions**

```
from tensorflow.keras.optimizers import Adam, SGD
from keras.metrics import binary_crossentropy
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import confusion_matrix
import itertools
```

**from tensorflow.keras.optimizers import Adam, SGD**: This line imports the Adam and SGD optimizers from the tensorflow and are commonly used in training deep learning models to update the model's parameters during the learning process.

**from keras.metrics import binary_crossentropy:** This line imports the binary_crossentropy metric from the keras.metrics module and it is a common loss function used in binary classification problems.

**from tensorflow.keras.callbacks import EarlyStopping:** This line imports the EarlyStopping callback from the tensorflow that stops the training process early if certain criteria, such as the validation loss not improving for a specified number of epochs, are met.

**from sklearn.metrics import confusion_matrix:** This line imports the confusion_matrix function from the sklearn, which is used to compute the confusion matrix, which is a useful tool for evaluating the performance of a classification model.

**import itertools:** This line imports the itertools module which provides various functions for efficient iteration and combination of iterables.

**12. Classify images into different layers and output classes**

```
early_stop=EarlyStopping(monitor='val_loss',patience=5)
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform',
padding='same', input_shape=(50, 50, 3)))
model.add(BatchNormalization())
model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform',
padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.3))
model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform',
padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform',
padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.3))
model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform',
padding='same'))
model.add(Flatten())
model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Dense(64, activation='relu', kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Dense(64, activation='relu', kernel_initializer='he_uniform'))
model.add(Dropout(0.3))
model.add(Dense(24, activation='relu', kernel_initializer='he_uniform'))
model.add(Dense(2, activation='softmax'))
```

The code defines a CNN model with multiple convolutional, pooling, batch normalization, dropout, and dense layers and it is designed for image classification tasks with input images of size 50x50 pixels and 3 color channels.

The model architecture aims to extract features from the input images and classify them into one of the two output classes using softmax activation.
The early_stop callback is also defined to monitor the validation loss and stop training if the loss doesn't improve for a certain number of epochs.

**13. Compile the defined model**

```
model.compile(Adam(learning_rate=0.0001), loss='binary_crossentropy',
metrics=['accuracy'])
```

Compiles the defined model with a specified optimizer, loss function, and evaluation metrics.The model is prepared for training with the specified optimizer, loss function, and evaluation metric(s). This step is necessary before starting the actual training process.

**14. Initiate training process**

```
history = model.fit(X_train, Y_train, validation_data = (X_test, Y_test), epochs = 10,
batch_size = 35)
```

The model.fit() function starts the training process with the specified settings. The model will iterate over the training data for the specified number of epochs, optimizing the model's weights using the defined loss function and optimizer. The validation data will be used to evaluate the model's performance during training. The training progress and metrics will be stored in the history variable.

epochs = 10: This argument determines the number of times the entire training dataset will be iterated over during the training process. Here, it is set to 10, meaning the model will be trained for  ten epoch.

**15. Save the trained model**

```
model.save("breastcancer.h5")
```

This line saves the model to a file named "breastcancer.h5" in the current directory. The file format used is Hierarchical Data Format 5 (HDF5), which is a commonly used file format for storing large amounts of numerical data. By saving the model, we can later load it and use it for inference or further training without having to retrain the model from scratch.

**16. Import module for loading trained model**

```
from keras.models import load_model
from keras.preprocessing import image
import numpy as np
```

The provided code imports the necessary modules for loading a pre-trained model, working with images, and manipulating arrays

**17. Load the saved model from file**

```
model= load_model("/tmp/breast_cancer_cnn/breastcancer.h5")
```

This line loads the pre-trained model stored in the file "breastcancer.h5" located at the specified file path and assigns it to the variable model. It is used to load a saved model from a file.

**18. Load an image (class1) to test the model**

```
img =
tf.keras.utils.load_img("/tmp/breast_cancer_cnn/breastcancerdataset/IDC_regular_ps50_idx5/
8863/1/8863_idx5_x1001_y801_class1.png", target_size=(50, 50))
```

**19. Use the model to make prediction**

```
x = tf.keras.utils.img_to_array(img)
x = np.expand_dims(x,axis = 0)
pred = model.predict(x)
```

The provided code converts the loaded image into a NumPy array, expands its dimensions, and uses the pre-trained model to make a prediction and now we will have the prediction for the input image stored in the pred variable, which we can further process or analyze as needed.

**20. Display result (class 1) for positive scenario**

```
np.round(pred[0][1],0)
```

**Output : 1**

**21. Load an image (class0) to test the model**

```
img =
tf.keras.utils.load_img("/tmp/breast_cancer_cnn/breastcancerdataset/IDC_regular_ps50_idx5/
10253/0/10253_idx5_x1001_y1001_class0.png", target_size=(50, 50))
```

**22. Use the model to make prediction**

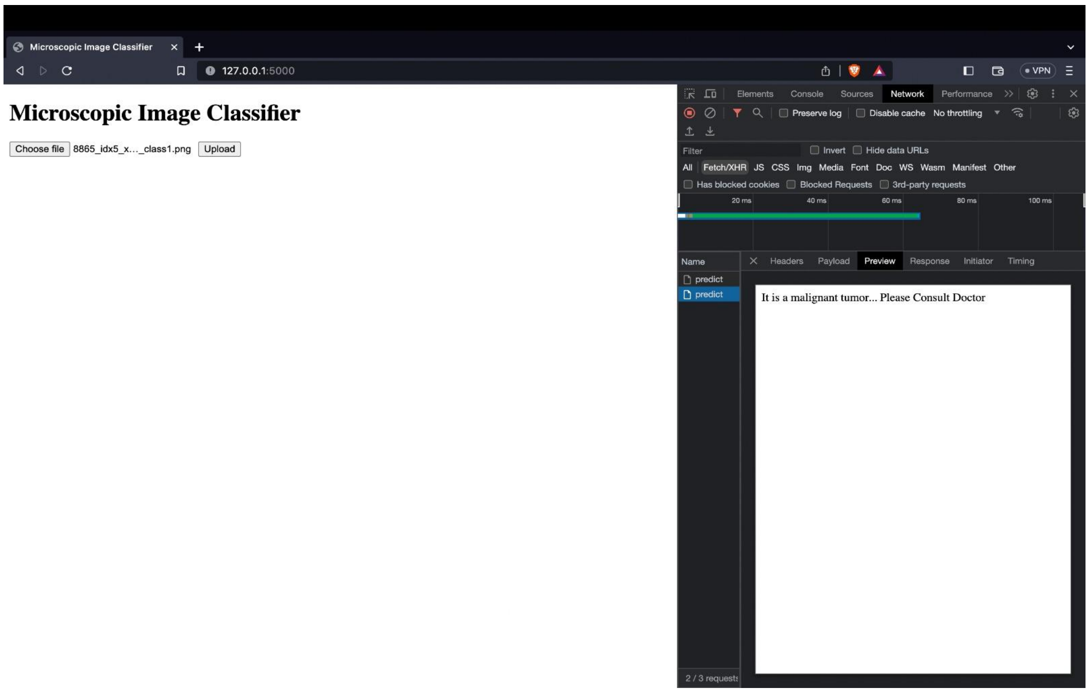
```
x = tf.keras.utils.img_to_array(img)
x = np.expand_dims(x,axis = 0)
pred = model.predict(x)
```

**23. Display result (class 1) for positive scenario**

```
np.round(pred[0][1],0)
```

**Output: 0**

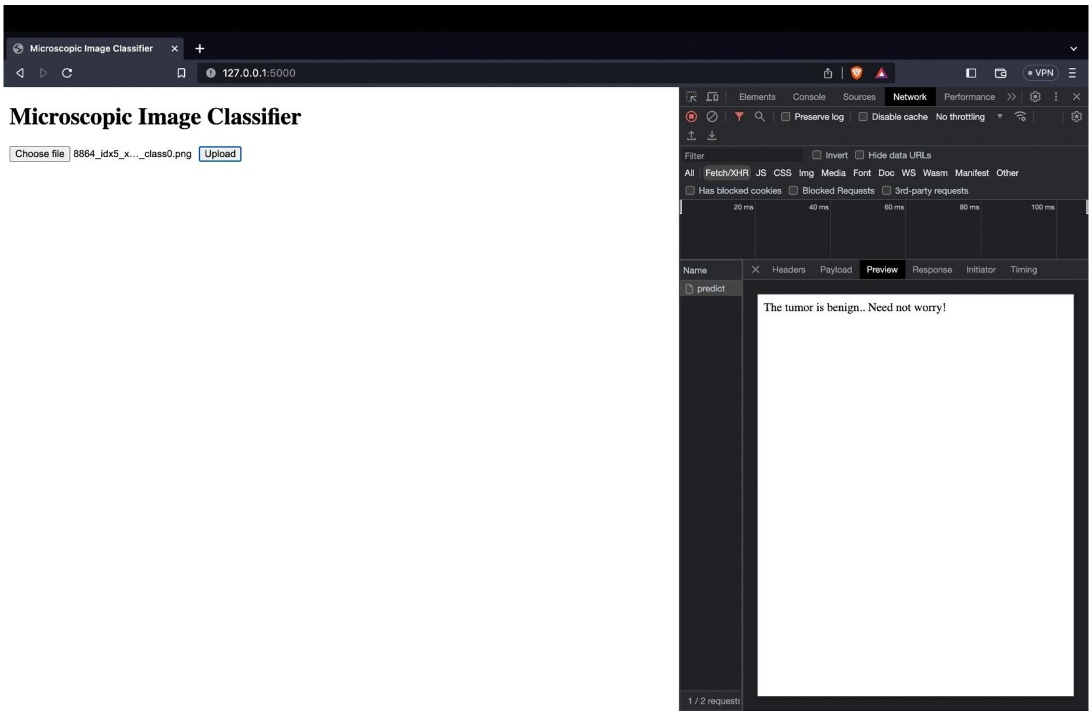
## Web page output:



Microscopic Image Classifier

Choose file 8865_idx5_x..._class1.png Upload

It is a malignant tumor... Please Consult Doctor



Microscopic Image Classifier

Choose file 8864_idx5_x..._class0.png Upload

The tumor is benign.. Need not worry!

# 8. CONCLUSION

The optimized deep learning model demonstrated excellent performance in predicting breast cancer. It achieved high accuracy and robustness in differentiating between malignant and benign tumors. The model's ability to learn and extract relevant features from medical images allowed for accurate classification, contributing to the early detection and diagnosis of breast cancer. The optimized architecture, combined with well-selected hyperparameters, yielded superior results compared to traditional machine learning approaches.

**Challenges and Future Directions:**

While optimized deep learning techniques show promise in breast cancer prediction, several challenges need to be addressed. One major challenge is the availability of large and diverse datasets to train models effectively. Collecting annotated and well-curated datasets is crucial to ensure model generalization. Additionally, ethical considerations regarding patient privacy and data protection must be carefully addressed when utilizing patient-specific medical data.

Future research directions include the exploration of multi-modal data integration, combining imaging data with clinical and genetic information, to improve prediction accuracy. Moreover, interpretability and explainability of deep learning models need to be further investigated to gain insights into the decision-making process and enhance model transparency. Collaboration between medical professionals, data scientists, and policymakers is essential to overcome these challenges and ensure the successful implementation of optimized deep learning techniques in clinical practice.

In conclusion, optimized deep learning techniques offer a promising approach for predicting breast cancer with high accuracy and efficiency. By leveraging the power of deep neural networks, these models can effectively learn from complex medical images and provide valuable insights for diagnosis and treatment planning. The integration of such techniques into clinical practice has the potential to revolutionize breast cancer screening and improve patient outcomes. However, further research, collaboration, and ethical considerations are necessary to overcome challenges and maximize the benefits of optimized deep learning in the field of breast cancer prediction.