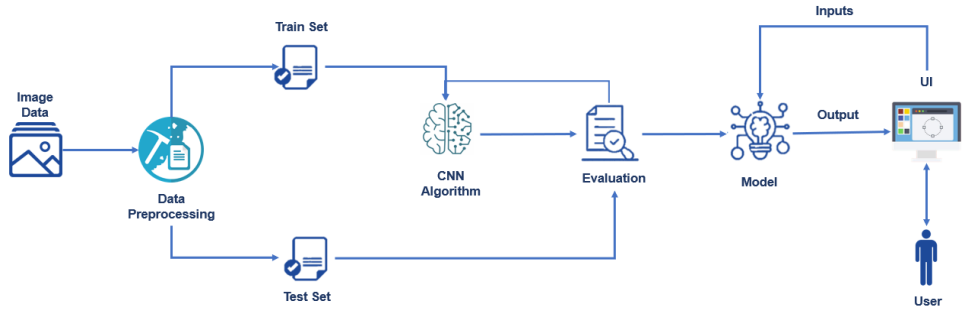
Cancer Vision: Advanced Breast Cancer Prediction with Deep Learning

Breast cancer is one of the main causes of cancer death worldwide. Computer-aided diagnosis systems showed potential for improving the diagnostic accuracy. But early detection and prevention can significantly reduce the chances of death. It is important to detect breast cancer as early as possible. The goal is to classify images into two classifications of malignant and benign. As early diagnostics significantly increases the chances of correct treatment and survival. In this application we are helping the doctors and patients to classify the Type of Tumour for the specific image given with the help of Neural Networks.



Pre-Requisites:

In order to develop this project, we need to install the following software/packages.

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, Jupiter Notebook,
QtConsole, Spyder, Glueviz, Orange, Rstudio, Visual Studio Code. For this project, we will be using a Jupyter notebook and Spyder
To install the Anaconda navigator and to know how to use Jupyter Notebook & Spyder using Anaconda.

Python packages:

NumPy: NumPy is a Python package that stands for 'Numerical Python. It is the core library for scientific computing, which contains a powerful n-dimensional array of objects.

Pandas: pandas is a fast, powerful, flexible, and easy-to-use open-source data analysis and manipulation tool, built on top of the Python programming language.

Matplotlib: It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits

Keras: Keras is an open-source library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. Up until version 2.3, Keras supported multiple backends, including TensorFlow, Microsoft Cognitive Toolkit, R, Theano, and PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible.

TensorFlow: TensorFlow is just one part of a much bigger, and growing ecosystem of libraries and extensions that help you accomplish your machine learning goals. It is a free and open-source software library for data flow and differentiable programming across a range of tasks. It is a symbolic math library and is also used for machine learning applications such as neural networks.

Flask: Web framework used for building Web applications

Python packages:

- open anaconda prompt as administrator
- Type “pip install numpy” and click enter.
- Type “pip install pandas” and click enter.
- Type “pip install scikit-learn” and click enter.
- Type “pip install tensorflow==2.3.2” and click enter.
- Type “pip install keras==2.3.1” and click enter.
- Type “pip install Flask” and click enter.

Deep Learning Concepts:

- **CNN:** a convolutional neural network is a class of deep neural networks, most commonly applied to analysing visual imagery. CNN Basic
- **Flask:** Flask is a popular Python web framework, meaning it is a third-party Python library used for developing web applications.

Flask Basics If you are using Pycharm IDE, you can install the packages through the command prompt and follow the same syntax as above.

Prior Knowledge:

We should have prior knowledge of following topics to complete this project.

- Deep Learning Concept
- CNN

- **Flask:** Flask is a popular Python web framework, meaning it is a third-party Python library used for developing web applications.

Project Objectives:

By the end of this project, we will:

- Know fundamental concepts and techniques of Convolutional Neural Network.
- Gain a broad understanding of image data.
- Know how to pre-process/clean the data using different data preprocessing techniques.
- know how to build a web application using the Flask framework.

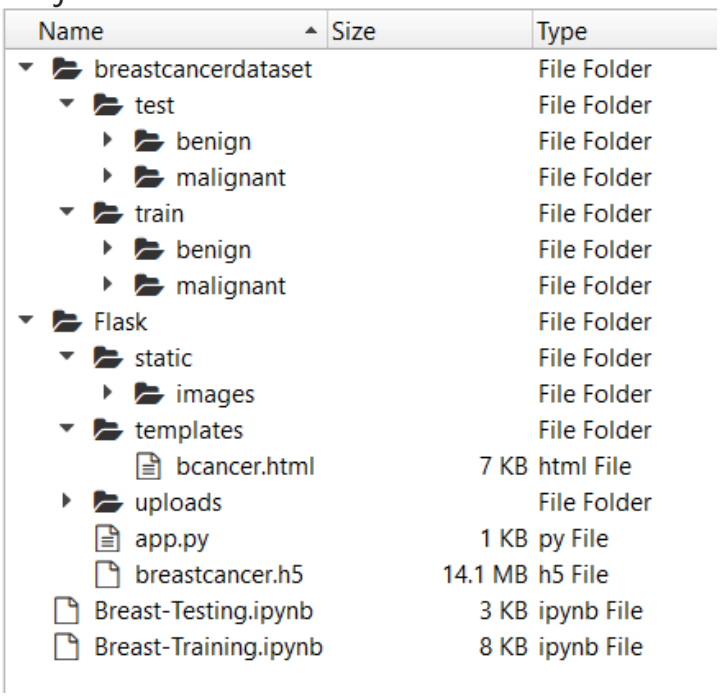
Project Flow:

- The user interacts with the UI (User Interface) to choose the image.
- The chosen image analysed by the model which is integrated with flask application.
- CNN Models analyse the image, then prediction is showcased on the Flask UI.

To accomplish this, we have to complete all the activities and tasks listed below

- Data Collection.
- Create Train and Test Folders.
- Data Preprocessing.
- Import the ImageDataGenerator library.
- Configure ImageDataGenerator class.
- ApplyImageDataGenerator functionality to Trainset and Test set.
- Model Building.
- Import the model building Libraries.
- Initializing the model.
- Adding Input Layer.
    - Adding Hidden Layer.
- **Application Building**
    - Create an HTML file
    - Build Python Code

Project Structure



- The dataset folder (breastcancerdataset) contains two folders test and train, each of them having benign and malignant tumour images.
- Flask folder has all the files necessary to build the flask application.
- static folder has the images that are needed in building the web page.
- Templates folder has the HTML page.
- Uploads folder has the uploads made by the user.
- app.py is the python script for server-side computing.
- .h5 file is the model file which is to be saved after model building.
- Breast-Training.ipynb and Breast-Testing.ipynb are the training and testing notebooks.

Data Collection:

There are many popular open sources for collecting the data.
E.g.: kaggle.com, UCI repository, etc

Download The Dataset:

The dataset used for this project is in this link. Please refer to the link to download the dataset. In this dataset, 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, create 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 as shown in project structure.

Image Preprocessing:

In this milestone we will be improving the image data that suppresses unwilling distortions or enhances some image features important for further processing, although perform some geometric transformations of images like rotation, scaling, translation, etc.

Import ImageDataGenerator Library and Configure It

Activity 1: Import ImageDataGenerator Library and Configure it

ImageDataGenerator class is used to load the images with different modifications like considering the zoomed image, flipping the image and rescaling the images to range of 0 and 1.

```
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)
```

Apply ImageDataGenerator Functionality To Train And Test Set

Specify the path of both the folders in flow_from_directory method.

```
x_train = train_datagen.flow_from_directory('breastcancerdataset/train',target_size = (64,64),batch_size = 32,
                                            class_mode = 'binary')
x_test =  test_datagen.flow_from_directory('breastcancerdataset/test',target_size = (64,64),batch_size = 32,
                                           class_mode = 'binary')
```

Model Building:

The neural network model is to be built by adding different network layers like convolution, pooling, flattening, dropout and neural layers.

Import The Required Model Building Libraries

Import the libraries that are required to initialize the neural network layer, create and add different layers to the neural network model.

```
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense, Dropout, Flatten
from keras.layers import MaxPooling2D
```

Initialize The Model

Initialize the neural network layer by creating a reference/object to the Sequential class.

```
model=Sequential()
```

Add The Convolution Layer:

The first layer of the neural network model, the convolution layer will be added. To create a convolution layer, Convolution2D class is used. It takes a number of feature detectors, feature detector size, expected input shape of the image, activation function as arguments. This layer applies feature detectors on the input image and returns a feature map (features from the image).

Activation Function: These are the functions which help us to decide if we need to activate the node or not. These functions introduce non-linearity in the networks.

```
model.add(Conv2D(64,(3, 3),activation='relu', input_shape=(75, 75, 3)))
```

Add The Pooling Layer:

Max Pooling selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.

After the convolution layer, a pooling layer is added. Max pooling layer can be added using MaxPooling2D class. It takes the pool size as a parameter. Efficient size of the pooling matrix is (2,2). It returns the pooled feature maps. (**Note:** Any number of convolution layers, pooling and dropout layers can be added)

```
model.add(MaxPooling2D(pool_size = (2,2)))
```

Add The Flatten Layer

The flatten layer is used to convert n-dimensional arrays to 1-dimensional arrays. This 1D array will be given as input to ANN layers.

```
model.add(Flatten())
```

Adding The Dense Layers

Three dense layers are added which usually takes number of units/neurons. Specifying the activation function, kind of weight initialization is optional.

```
model.add(Dense(output_dim = 40 ,init = 'uniform',activation = 'relu'))
```

```
model.add(Dense(output_dim = 1,activation = 'softmax',init ='uniform'))
```

Note: Any number of convolution, pooling and dense layers can be added according to the data.

Compile The Model

After adding all the required layers, the model is to be compiled. For this step, loss function, optimizer and metrics for evaluation can be passed as arguments.

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

Train The Model

Fit the neural network model with the train and test set, number of epochs and validation steps.

```
model.fit_generator(x_train, steps_per_epoch = 5,epochs = 10,validation_data = x_test,validation_steps = 40)
```

Accuracy, Loss: Loss value implies how poorly or well a model behaves after each iteration of optimization. An accuracy metric is used to measure the algorithm's performance in an interpretable way. The accuracy of a model is usually determined after the model parameters and is calculated in the form of a percentage.

Save The Model:

The weights are to be saved for future use. The weights are saved in as .h5 file using save ().

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

model. Summary() can be used to see all parameters and shapes in each layer in our models.

Test The Model:

The model is to be tested with different images to know if it is working correctly.

Import The Packages and Load The Saved Model:

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

```
model = load_model("breastcancer.h5")
```

Load The Test Image, Pre-Process It and Predict:

Pre-processing the image includes converting the image to array and resizing according to the model. Give the pre-processed image to the model to know to which class your model belongs to.

```
img = image.load_img('malignant.png',target_size = (64,64))
```

```
x  = image.img_to_array(img)
x = np.expand_dims(x,axis = 0)
```

```
pred = model.predict_classes(x)
```

```
pred
```

```
array([[1]])
```

Application Building:

After the model is built, we will be integrating it to a web application so that normal users can also use it. The users need to give the microscopic image of the tissue/tumour to know the predictions.

Building Html Pages:

Intro.html displays an introduction about the project.

upload.html gives the emergency alert.

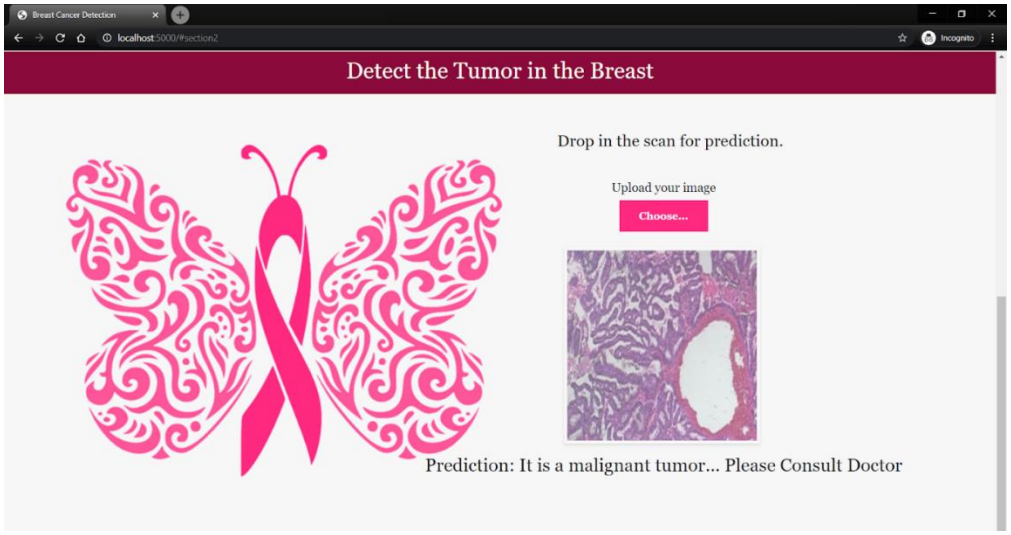
We also use JavaScript-main.js and CSS-main.css to enhance our functionality and view of HTML page.

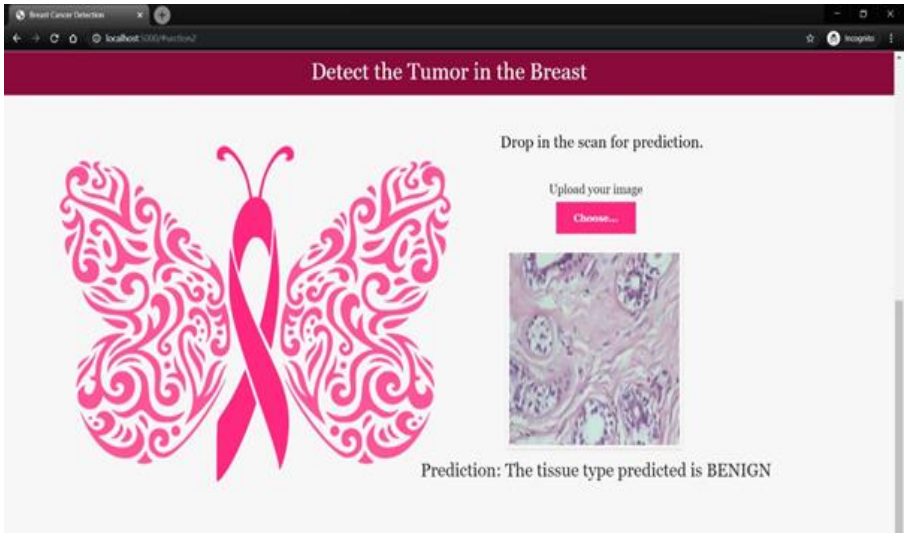
Link:CSS , JS.

The home page looks like this. When you click on the button “Drop the scan”, you’ll be redirected to the predict section.

In this section you can browse and choose the image you want to predict and then

click on predict to get the predictions.





Build Python Code:

Step 1: Load the required packages

```
from __future__ import division, print_function
import os
import numpy as np
from keras.preprocessing import image
from keras.models import load_model
import tensorflow as tf
from flask import Flask, request, render_template
from werkzeug.utils import secure_filename

```

Step 2: Initialize graph, load the model, initialize the flask app and load the model

Graph elements are required to work with tensorflow. So, graph elements are created explicitly. Instance of Flask is created and the model is loaded using load_model from keras.

```
global graph
graph=tf.get_default_graph()
# Define a flask app
app = Flask(__name__)


# Load your trained model
model = load_model("breastcancer.h5")
```

Step 3: Configure the home page

```
@app.route('/', methods=['GET'])
def index():
    # Main page
    return render_template('bcancer.html')
```

Step 4: Pre-process the frame and run

Pre-process the captured frame and give it to the model for prediction. Based on the prediction the output text is generated and sent to the HTML to display.

```
@app.route('/predict', methods=['GET', 'POST'])
def upload():
    if request.method == 'POST':
        # Get the file from post request
        f = request.files['image']

        # Save the file to ./uploads
        basepath = os.path.dirname(__file__)
        file_path = os.path.join(
            basepath, 'uploads', secure_filename(f.filename))
        f.save(file_path)
        img = image.load_img(file_path, target_size=(64, 64))

        x = image.img_to_array(img)
        x = np.expand_dims(x, axis=0)

        with graph.as_default():
            preds = model.predict_classes(x)
        if preds[0][0]==0:
            text = "The tumor is benign.. Need not worry!"
        else:
            text = "It is a malignant tumor... Please Consult Doctor"
        text = text
        return text
```

Run the flask application using the run method. By default, the flask runs on 5000 ports. If the port is to be changed, an argument can be passed and port can be modified.

```
if __name__ == '__main__':
    app.run(debug=True,threaded = False)
```

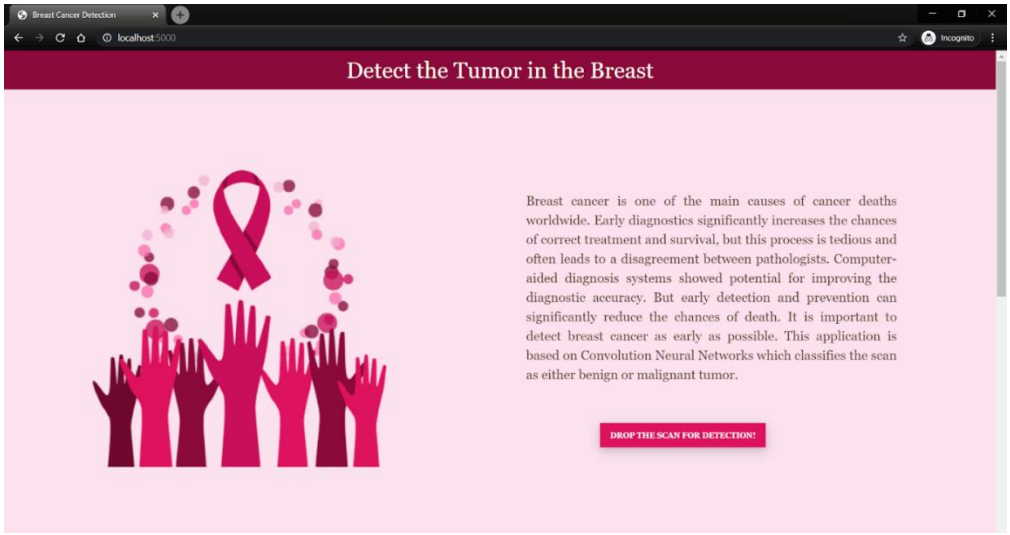
Run The Application:

In anaconda prompt, navigate to the folder in which the flask app is present. When the python file is executed the localhost is activated on 5000 port and can be accessed through it.

```
(base) F:\Projects\Breast Cancer\Main\Flask>python app.py
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Model loaded. Check http://127.0.0.1:5000/
 * Debugger is active!
 * Debugger PIN: 257-358-499
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Open the browser and navigate to localhost:5000 to check your application: -

The home page looks like this. When you click on the button “Drop the scan”, you’ll be redirected to the predict section.



In this section you can browse and choose the image you want to predict and then click on predict to get the predictions.