# Breast Cancer Detection

#### Mt. SAC CISB 62 Midterm Project Fall 2023

#### Vedavit Shetty

The main objective of this project is to find the best hyperparameters for a neural network model that classifies breast cancer tumors, so that it achieves the highest possible accuracy on the validation set. The Keras Tuner's RandomSearch method is utilized to automate and optimize this hyperparameter tuning process.

You can find this projected hosted on github: [https://github.com/vedavitshetty/CISB-62-Midterm-Breast-Cancer-Detection-Project (https://github.com/vedavitshetty/CISB-62-Midterm-Breast-Cancer-Detection-Project)](https://github.com/vedavitshetty/CISB-62-Midterm-Breast-Cancer-Detection-Project)

#### Import Libraries

In [1]:
```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

# Exploratory Data Analysis (EDA)

#### Load Data

In [2]:
```
# Load the dataset
data = load_breast_cancer()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target
```

#### Display the first 5 values

In [3]:
```
df.head()
```

Out[3]:

| | mean radius | mean texture | mean perimeter | mean area | mean smoothness | mean compactness | mean concavity | mean concave points | mean symmetry |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 17.99 | 10.38 | 122.80 | 1001.0 | 0.11840 | 0.27760 | 0.3001 | 0.14710 | 0.2419 |
| **1** | 20.57 | 17.77 | 132.90 | 1326.0 | 0.08474 | 0.07864 | 0.0869 | 0.07017 | 0.1812 |
| **2** | 19.69 | 21.25 | 130.00 | 1203.0 | 0.10960 | 0.15990 | 0.1974 | 0.12790 | 0.2069 |
| **3** | 11.42 | 20.38 | 77.58 | 386.1 | 0.14250 | 0.28390 | 0.2414 | 0.10520 | 0.2597 |
| **4** | 20.29 | 14.34 | 135.10 | 1297.0 | 0.10030 | 0.13280 | 0.1980 | 0.10430 | 0.1809 |

5 rows × 31 columns

**See info**

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
 30  target                   569 non-null    int64  
dtypes: float64(30), int64(1)
memory usage: 137.9 KB
```

**See the count, mean, standard deviation, miniumum, first quartile, median, third quartile, and maximum values of each column**

In [5]:
```
df.describe()
```

Out[5]:

| | mean radius | mean texture | mean perimeter | mean area | mean smoothness | mean compactness | mean concavity | |
|---|---|---|---|---|---|---|---|---|
| **count** | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 5 |
| **mean** | 14.127292 | 19.289649 | 91.969033 | 654.889104 | 0.096360 | 0.104341 | 0.088799 | |
| **std** | 3.524049 | 4.301036 | 24.298981 | 351.914129 | 0.014064 | 0.052813 | 0.079720 | |
| **min** | 6.981000 | 9.710000 | 43.790000 | 143.500000 | 0.052630 | 0.019380 | 0.000000 | |
| **25%** | 11.700000 | 16.170000 | 75.170000 | 420.300000 | 0.086370 | 0.064920 | 0.029560 | |
| **50%** | 13.370000 | 18.840000 | 86.240000 | 551.100000 | 0.095870 | 0.092630 | 0.061540 | |
| **75%** | 15.780000 | 21.800000 | 104.100000 | 782.700000 | 0.105300 | 0.130400 | 0.130700 | |
| **max** | 28.110000 | 39.280000 | 188.500000 | 2501.000000 | 0.163400 | 0.345400 | 0.426800 | |

8 rows × 31 columns

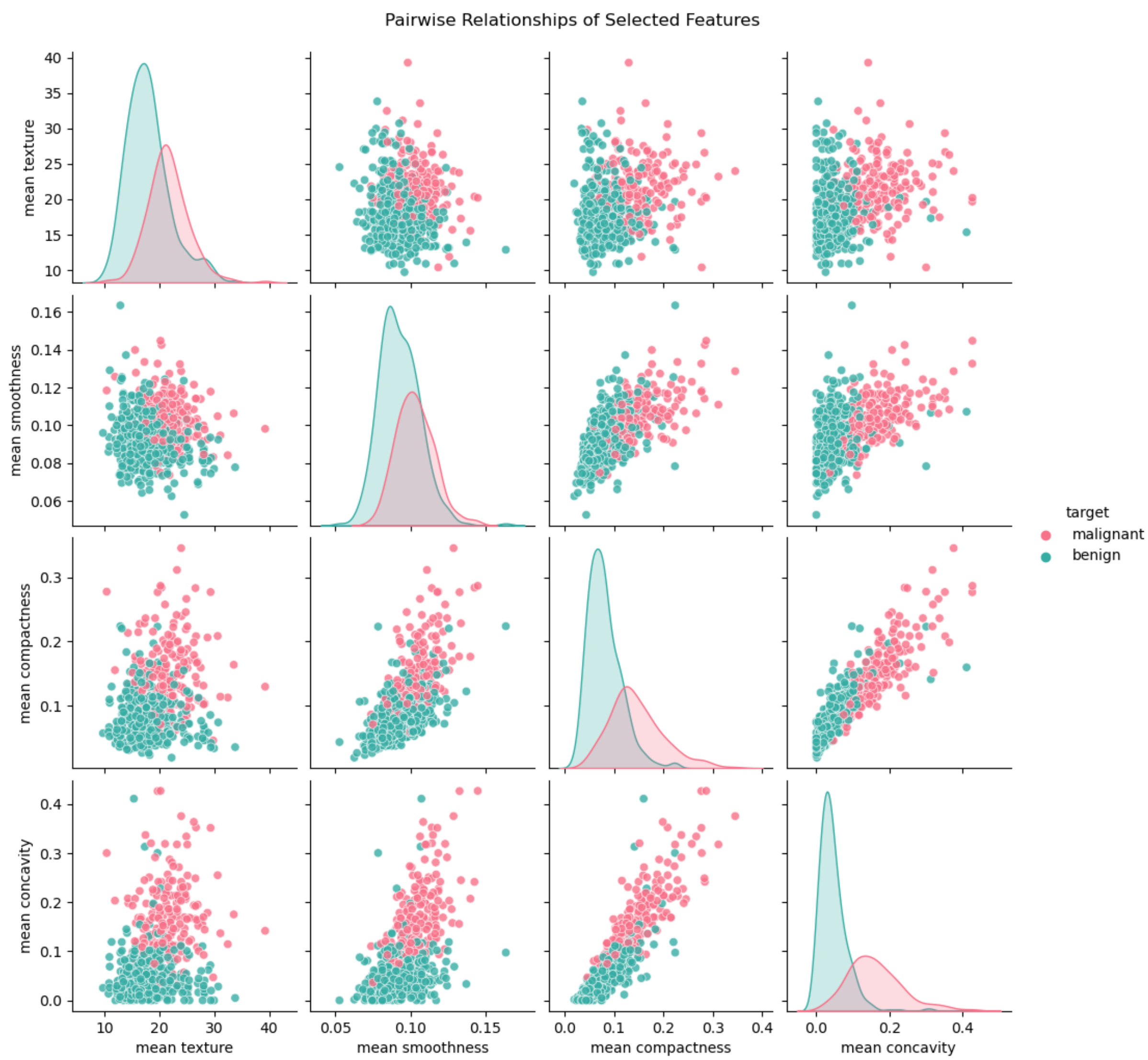
### Check null values

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
worst fractal dimension    0
target                     0
dtype: int64
```

### Visualization

In [7]:
```
selected_features = ['mean texture', 'mean smoothness', 'mean compactness

# Create a temporary dataframe with the target labels
temp_df = df.copy()
temp_df.replace(to_replace={'target': {0: data.target_names[0]}}, inplace
temp_df.replace(to_replace={'target': {1: data.target_names[1]}}, inplace

sns.pairplot(temp_df, hue='target', vars=selected_features, palette="husl
plt.suptitle('Pairwise Relationships of Selected Features', y=1.02)
plt.show()
```




#### Insights

- Malignant tumors, in general, tend to have higher values for the features mean smoothness, mean compactness, and mean concavity.
- As mean compactness increases, mean concavity also seems to increase resulting in a strong positive correlation
- Mean smoothness and mean compactness have a more mild positive correlation
- Mean smoothness and mean concavity have some positive correlation, but not as pronounced as the previous 2 mentioned

# Data Transformation and Splitting

In [8]:
```
# Splitting the data
X = df.drop('target', axis=1)
y = df['target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

# Standardizing the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

# Build the Deep Learning Model

In [9]:
```
import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, BatchNormalization

# Building the ANN
model = Sequential()
model.add(Dense(32, activation='relu', input_dim=X_train.shape[1]))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(16, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc
```

# Hyperparameter Tuning using Keras Tuner

In [11]:
```
#!pip install keras-tuner

from keras_tuner.tuners import RandomSearch

def build_model(hp):
    model = Sequential()
    model.add(Dense(units=hp.Int('units', min_value=32, max_value=512, st
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(16, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=
    return model

tuner = RandomSearch(
    build_model,
    objective='val_accuracy',
    max_trials=5,
    executions_per_trial=2,
    directory='breast_cancer_model_dir',
    project_name='breast_cancer')

tuner.search(X_train, y_train, epochs=10, validation_data=(X_test, y_test
```

```
Trial 5 Complete [00h 00m 02s]
val_accuracy: 0.9780701696872711

Best val_accuracy So Far: 0.9780701696872711
Total elapsed time: 00h 00m 08s
```

# Summary and Conclusion

#### Evaluation

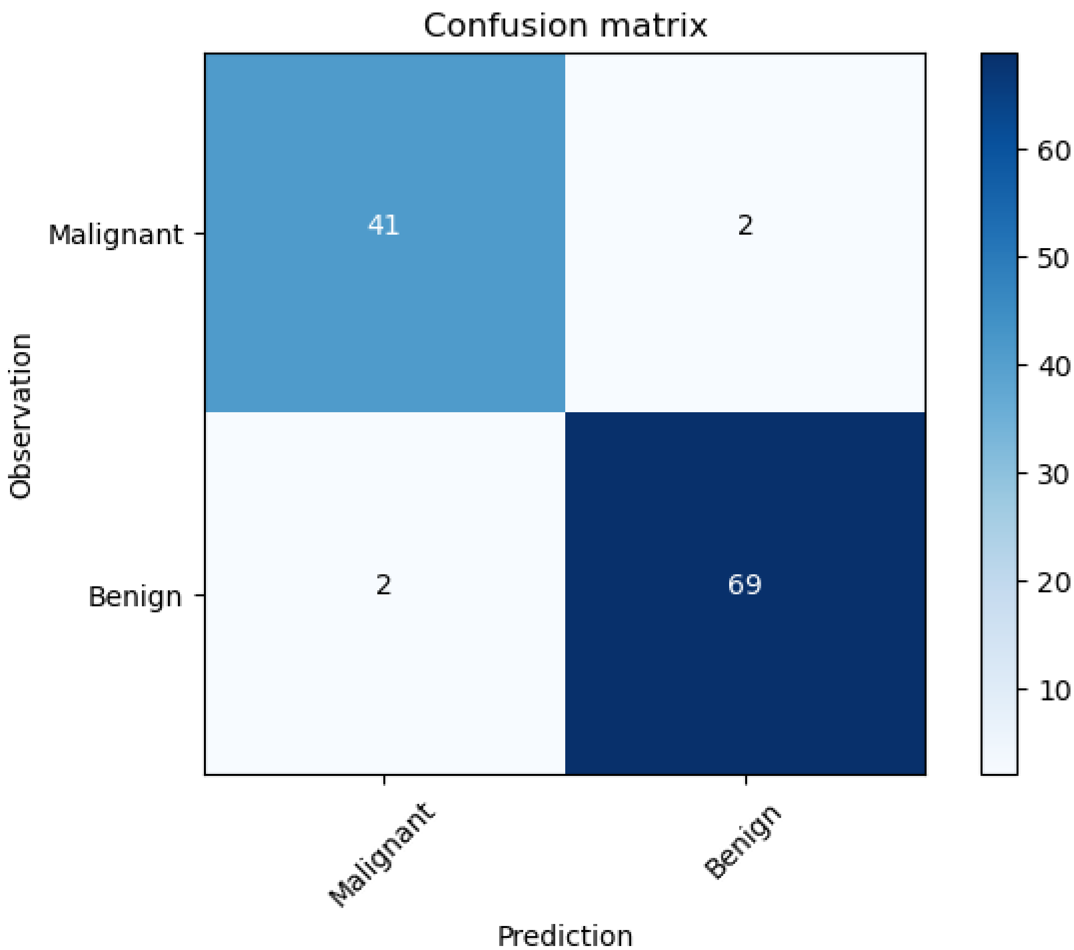
In [12]:
```
# Get the best model
best_model = tuner.get_best_models()[0]

# Training the best model
history = best_model.fit(X_train, y_train, epochs=50, validation_data=(X_

# Evaluation
loss, accuracy = best_model.evaluate(X_test, y_test)
print(f"Accuracy on test set: {accuracy*100:.2f}%")
```

```
Epoch 45/50
15/15 [==============================] - 0s 2ms/step - loss: 0.0331 - a
ccuracy: 0.9890 - val_loss: 0.1313 - val_accuracy: 0.9474
Epoch 46/50
15/15 [==============================] - 0s 2ms/step - loss: 0.0568 - a
ccuracy: 0.9758 - val_loss: 0.1159 - val_accuracy: 0.9561
Epoch 47/50
15/15 [==============================] - 0s 2ms/step - loss: 0.0294 - a
ccuracy: 0.9956 - val_loss: 0.1317 - val_accuracy: 0.9649
Epoch 48/50
15/15 [==============================] - 0s 2ms/step - loss: 0.0269 - a
ccuracy: 0.9890 - val_loss: 0.1156 - val_accuracy: 0.9649
Epoch 49/50
15/15 [==============================] - 0s 2ms/step - loss: 0.0309 - a
ccuracy: 0.9912 - val_loss: 0.1144 - val_accuracy: 0.9649
Epoch 50/50
15/15 [==============================] - 0s 2ms/step - loss: 0.0263 - a
ccuracy: 0.9912 - val_loss: 0.1083 - val_accuracy: 0.9649
4/4 [==============================] - 0s 922us/step - loss: 0.1083 - a
ccuracy: 0.9649
Accuracy on test set: 96.49%
```

In [13]:
```
from collections import Counter
from sklearn.metrics import confusion_matrix
import itertools
# From the SKLEARN website, to view a confusion matrix
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1]))
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('Observation')
    plt.xlabel('Prediction')
```

In [14]:
```
# Confusion Matrix
from sklearn.metrics import confusion_matrix, classification_report
y_pred = (best_model.predict(X_test) > 0.5).astype("int32")
cm = confusion_matrix(y_test, y_pred)
class_names = ['Malignant', 'Benign']
plot_confusion_matrix(cm, class_names)
plt.show()
```

```
4/4 [==============================] - 0s 738us/step
```




In [15]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.95      0.95      0.95        43
           1       0.97      0.97      0.97        71

    accuracy                           0.96       114
   macro avg       0.96      0.96      0.96       114
weighted avg       0.96      0.96      0.96       114
```

**Explain the confusion matrix with your own words**

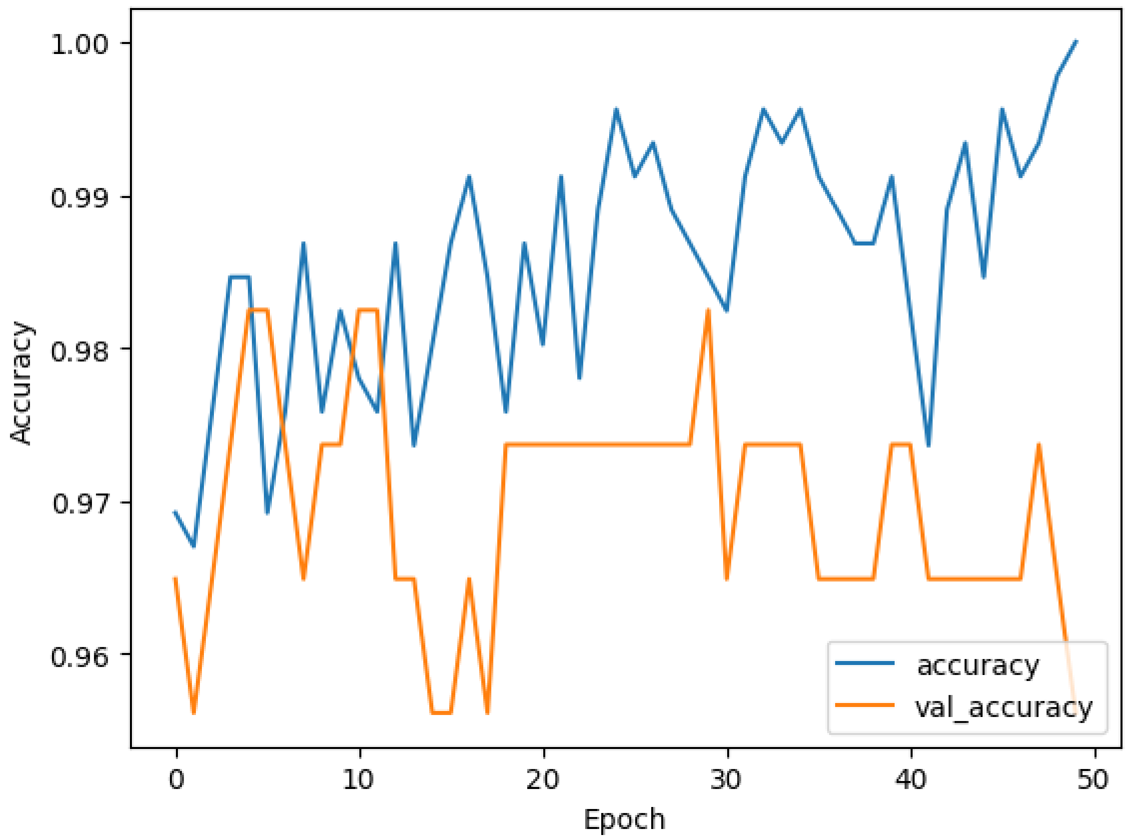
1. The model predicted 41 cases as "Malignant" and they were actually "Malignant".

2. The model predicted 2 cases as "Benign" when they were actually "Malignant".
3. The model predicted 2 cases as "Malignant" when they were actually "Benign".
4. The model predicted 69 cases as "Benign" and they were actually "Benign".

Some observations:

- The model seems to be performing quite well as the majority of predictions fall on the diagonal, which represents correct predictions.
- The errors are balanced, with the model misclassifying 2 cases for both false positives and false negatives.

In [16]:
```
# Check for overfitting in the history object
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()
```




**Overfitting Observations**

- **High Training Accuracy**: The blue line, which represents the training accuracy, is consistently high, nearly reaching 1.00 (or 100%) for most epochs. This suggests that the model has learned the training data very well.
- **Validation Accuracy Fluctuations**: The orange line, representing validation accuracy, shows more fluctuation compared to the training accuracy. There's a noticeable dip in the middle epochs and then it rises again towards the later epochs. This fluctuation suggests

that the model might be experiencing some variability in how well it generalizes to unseen data.

- **Divergence between Training and Validation**: Around the middle epochs (approximately epochs 20-35), there's a clear gap between training and validation accuracy. This gap suggests that the model might be overfitting the training data during these epochs, as it performs exceptionally well on the training data but not as well on the validation data.
- **Convergence in Later Epochs**: Towards the later epochs (after 40), the validation accuracy seems to improve and get closer to the training accuracy. This convergence indicates that the model's generalization to unseen data has improved in these epochs.

In [ ]: