KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

AN AUTONOMOUS INSTITUTION - ACCREDITED BY NAAC WITH 'A' GRADE

Narayanaguda, Hyderabad.

EMBEDDED LEARNING DAY1 EXERCISE

ML MODEL VS DL MODEL

BY

ASHA M

ASSISTANT PROFESSOR

CSE(AI&ML)

KMIT

MACHINE LEARNING MODEL

1. Importing Libraries

```
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import seaborn as sns
```

`pandas` (pd) is used for data manipulation and analysis.

`load_breast_cancer` from `sklearn.datasets` loads the Breast Cancer dataset.

`train_test_split` is used to split the dataset into training and testing sets.

`StandardScaler` is typically used for scaling data (though it's not used here).

`DecisionTreeClassifier` is used to create a decision tree model for classification.

`accuracy_score` is used to calculate the accuracy of the model.

`seaborn` (sns) is used for visualization, specifically for generating heatmaps (in this case, for the confusion matrix).

2. Loading the Breast Cancer Dataset

```
# Load Breast Cancer dataset
data = load_breast_cancer()
```

This line loads the Breast Cancer dataset from `sklearn.datasets`. The dataset contains 30 features describing cell characteristics of breast cancer tumors and a target variable (`y`) indicating whether the tumor is malignant or benign.

3. Extracting Features and Target Variables

```
# Load features and target
X = data.data
y = data.target
```

`X` contains the feature matrix (30 numerical features describing the tumors).

`y` contains the target vector, where 1 indicates malignant tumors and 0 indicates benign tumors.

4. Converting the Dataset to a DataFrame and Printing First Few Rows

```python
# Convert to DataFrame for better visualization
df = pd.DataFrame(X, columns=data.feature_names)
df['target'] = y

# Print the first few rows of the dataset
print("Dataset (First 5 Rows):")
print(df.head())
```

The feature matrix (`X`) is converted into a **pandas DataFrame** with **column names** taken from `data.feature_names`.

A new column, `'target'`, is added to the DataFrame to store the target labels (`y`), allowing for better visualization and manipulation.

Printing the first few rows of the dataset (`df.head()`) helps to quickly inspect the structure of the data, including both the features and the target variable.

```
Dataset (First 5 Rows):
   mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0        17.99         10.38          122.80     1001.0          0.11840
1        20.57         17.77          132.90     1326.0          0.08474
2        19.69         21.25          130.00     1203.0          0.10960
3        11.42         20.38           77.58      386.1          0.14250
4        20.29         14.34          135.10     1297.0          0.10030

   mean compactness  mean concavity  mean concave points  mean symmetry  \
0           0.27760          0.3001              0.14710         0.2419
1           0.07864          0.0869              0.07017         0.1812
2           0.15990          0.1974              0.12790         0.2069
3           0.28390          0.2414              0.10520         0.2597
4           0.13280          0.1980              0.10430         0.1809

   mean fractal dimension  ...  worst texture  worst perimeter  worst area  \
0                 0.07871  ...          17.33           184.60      2019.0
1                 0.05667  ...          23.41           158.80      1956.0
2                 0.05999  ...          25.53           152.50      1709.0
3                 0.09744  ...          26.50            98.87       567.7
4                 0.05883  ...          16.67           152.20      1575.0

   worst smoothness  worst compactness  worst concavity  worst concave points  \
0            0.1622             0.6656           0.7119                0.2654
1            0.1238             0.1866           0.2416                0.1860
2            0.1444             0.4245           0.4504                0.2430
3            0.2098             0.8663           0.6869                0.2575
4            0.1374             0.2050           0.4000                0.1625

   worst symmetry  worst fractal dimension  target
0          0.4601                  0.11890       0
1          0.2750                  0.08902       0
2          0.3613                  0.08758       0
3          0.6638                  0.17300       0
4          0.2364                  0.07678       0

[5 rows x 31 columns]
```

5. Splitting the Data into Training and Testing Sets

```
# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

The data is split into **training (80%)** and **testing (20%)** sets using `train_test_split`.

`random_state=42` ensures the results are reproducible.

6. Initializing the Decision Tree Classifier

```
# Initialize Decision Tree Classifier
dt_model = DecisionTreeClassifier(random_state=42)
```

A **Decision Tree Classifier** is created and initialized with a fixed `random_state=42` to ensure consistent results.

7. **Training the Decision Tree Model**

```
# Train the model
dt_model.fit(X_train, y_train)
```

The **Decision Tree model** is trained using the **training data** (`X_train`, `y_train`).

8. Making Predictions

```python
# Predict on the test data
y_pred_dt = dt_model.predict(X_test)
```

The trained model is used to predict the labels for the **test data** (`X_test`).

9. Evaluating the Model

```
# Evaluate the model
dt_accuracy = accuracy_score(y_test, y_pred_dt)
```

The **accuracy** of the model is calculated by comparing the predicted labels (`y_pred_dt`) with the actual test labels (`y_test`).

10. **Printing the Accuracy**

```
print("Decision Tree Results:")
print(f"Accuracy: {dt_accuracy:.2f}")
```

The **accuracy** of the Decision Tree model is printed, formatted to two decimal places.

11. **Visualizing the Confusion Matrix**

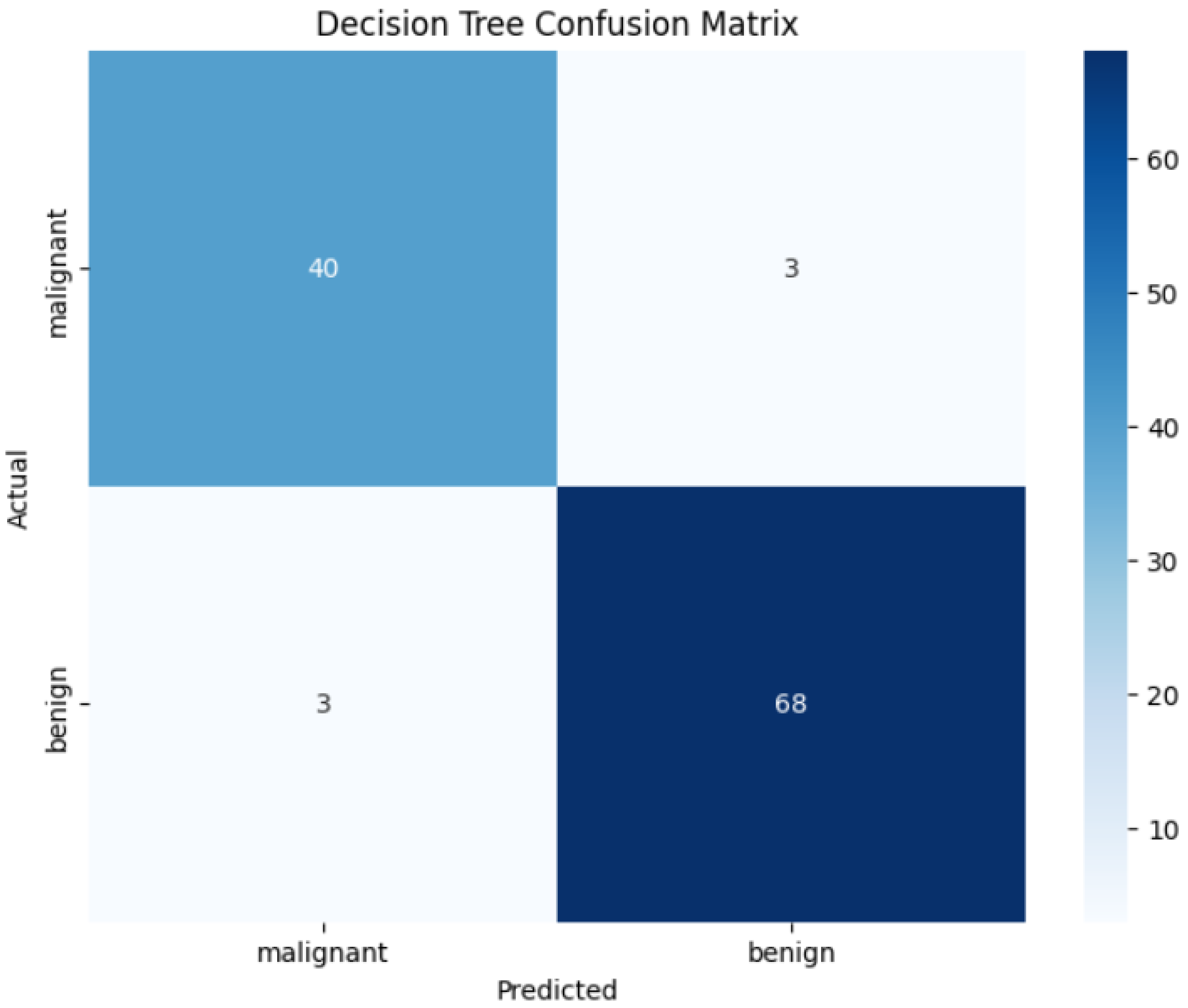
```python
# Visualization 1: Confusion Matrix for Decision Tree
plt.figure(figsize=(8, 6))
sns.heatmap(dt_conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=data.target_names, yticklabels=data.target_names)
plt.title('Decision Tree Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

The **Confusion Matrix** is meant to be visualized using **seaborn's heatmap**.

However, the confusion matrix (`dt_conf_matrix`) needs to be computed using `confusion_matrix(y_test, y_pred_dt)`.

Once the confusion matrix is computed, `sns.heatmap` can be used to visualize it, with `xticklabels` and `yticklabels` set to the target names (`malignant` and `benign`).

```
Decision Tree Results:
Accuracy: 0.95
```



Summary of the Process:

1. **Data Loading & Preparation**: Load the Breast Cancer dataset and convert it into a pandas DataFrame for better visualization and manipulation.
2. **Train-Test Split**: Split the dataset into training and testing sets.
3. **Model Initialization & Training**: Initialize and train a Decision Tree classifier.
4. **Prediction & Evaluation**: Make predictions and evaluate the model using accuracy.
5. **Visualization**: Convert the dataset into a DataFrame and print the first few rows to understand the structure of the data. A confusion matrix is also visualized (though it needs a fix).

DEEP LEARNING MODEL

1. Importing Libraries

```
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

`pandas` : A library used to handle data in tabular form (DataFrames).

`sklearn.datasets.load_breast_cancer` : A function that loads the Breast Cancer dataset.

`train_test_split` : Splits the data into training and test sets.

`StandardScaler` : Used to standardize the features (making them have zero mean and unit variance).

`accuracy_score` : A function that calculates the accuracy of the model.

`tensorflow` : A machine learning framework for building and training neural networks.

`Sequential` : Used to define a neural network in a linear fashion (adding layers one by one).

`Dense` : A fully connected neural network layer, used to define the number of neurons in each layer.

2. Loading the Breast Cancer Dataset

```
# Load Breast Cancer dataset
data = load_breast_cancer()
```

This line loads the Breast Cancer dataset from `sklearn.datasets`. The dataset contains 30 features describing cell characteristics of breast cancer tumors and a target variable (`y`) indicating whether the tumor is malignant or benign.

3. Extracting Features and Target Variables

```
# Load features and target
X = data.data
y = data.target
```

`X` contains the feature matrix (30 numerical features describing the tumors).

`y` contains the target vector, where 1 indicates malignant tumors and 0 indicates benign tumors.

4. Converting the Dataset to a DataFrame and Printing First Few Rows

```
# Convert to DataFrame for better visualization
df = pd.DataFrame(X, columns=data.feature_names)
df['target'] = y

# Print the first few rows of the dataset
print("Dataset (First 5 Rows):")
print(df.head())
```

The feature matrix (`X`) is converted into a **pandas DataFrame** with **column names** taken from `data.feature_names`.

A new column, `'target'`, is added to the DataFrame to store the target labels (`y`), allowing for better visualization and manipulation.

Printing the first few rows of the dataset (`df.head()`) helps to quickly inspect the structure of the data, including both the features and the target variable.

```
Dataset (First 5 Rows):
   mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0        17.99         10.38          122.80     1001.0          0.11840   
1        20.57         17.77          132.90     1326.0          0.08474   
2        19.69         21.25          130.00     1203.0          0.10960   
3        11.42         20.38           77.58      386.1          0.14250   
4        20.29         14.34          135.10     1297.0          0.10030   

   mean compactness  mean concavity  mean concave points  mean symmetry  \
0           0.27760          0.3001              0.14710         0.2419   
1           0.07864          0.0869              0.07017         0.1812   
2           0.15990          0.1974              0.12790         0.2069   
3           0.28390          0.2414              0.10520         0.2597   
4           0.13280          0.1980              0.10430         0.1809   

   mean fractal dimension  ...  worst texture  worst perimeter  worst area  \
0                 0.07871  ...          17.33           184.60      2019.0   
1                 0.05667  ...          23.41           158.80      1956.0   
2                 0.05999  ...          25.53           152.50      1709.0   
3                 0.09744  ...          26.50            98.87       567.7   
4                 0.05883  ...          16.67           152.20      1575.0   

   worst smoothness  worst compactness  worst concavity  worst concave points  \
0            0.1622             0.6656           0.7119                0.2654   
1            0.1238             0.1866           0.2416                0.1860   
2            0.1444             0.4245           0.4504                0.2430   
3            0.2098             0.8663           0.6869                0.2575   
4            0.1374             0.2050           0.4000                0.1625   

   worst symmetry  worst fractal dimension  target  
0          0.4601                  0.11890       0  
1          0.2750                  0.08902       0  
2          0.3613                  0.08758       0  
3          0.6638                  0.17300       0  
4          0.2364                  0.07678       0  

[5 rows x 31 columns]
```

5. Splitting the Data into Training and Testing Sets

```
# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

The data is split into **training (80%)** and **testing (20%)** sets using `train_test_split`.

`random_state=42` ensures the results are reproducible.

6. Standardizing the Features

```
# Standardize features for ANN model
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

`StandardScaler()` standardizes the features to have zero mean and unit variance.

- `fit_transform(X_train)` scales the training data and learns the scaling parameters.
- `transform(X_test)` applies the same scaling transformation to the test data, ensuring no data leakage.

7. Building the ANN Model

```
# Initialize ANN model
ann_model = Sequential([
    Dense(64, activation='relu', input_dim=X_train_scaled.shape[1]),  # Input layer
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')  # Output layer for binary classification
])
```

`Sequential()`: This is used to define the model layer by layer.

`Dense(64, activation='relu', input_dim=X_train_scaled.shape[1])`: The input layer with 64 neurons and ReLU activation. The `input_dim` is set to the number of features in the dataset (`X_train_scaled.shape[1]`).

`Dense(32, activation='relu')`: A hidden layer with 32 neurons and ReLU activation.

`Dense(1, activation='sigmoid')`: The output layer with 1 neuron and sigmoid activation for binary classification (0 or 1).

8. Compiling the Model

```
# Compile the model
ann_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

`optimizer='adam'` : Adam optimizer is used, which adapts the learning rate during training.

`loss='binary_crossentropy'` : This loss function is used for binary classification tasks. It measures the difference between the predicted and actual values.

`metrics=['accuracy']` : This metric is used to track the accuracy of the model during training and evaluation.

9. Training the Model

```
# Train the model
ann_model.fit(X_train_scaled, y_train, epochs=10)
```

`fit(X_train_scaled, y_train, epochs=10)` : The model is trained using the training data (`X_train_scaled` and `y_train`) for 10 epochs (iterations over the entire dataset).

10. Making Predictions on the Test Set

```python
# Predict on the test set
y_pred_ann = (ann_model.predict(X_test_scaled) > 0.5).astype("int32")
```

`ann_model.predict(X_test_scaled)` : The model predicts probabilities (between 0 and 1) for the test data.

`(ann_model.predict(X_test_scaled) > 0.5)` : Converts the probabilities into binary predictions (1 for probabilities > 0.5, and 0 otherwise).

`.astype("int32")` : Converts the predicted values to integers (0 or 1).

11. Evaluating the Model

```
# Evaluate the model
ann_accuracy = accuracy_score(y_test, y_pred_ann)
```

`accuracy_score(y_test, y_pred_ann)`: Calculates the accuracy by comparing the actual labels (`y_test`) with the predicted labels (`y_pred_ann`).

`print(f"Accuracy: {ann_accuracy:.2f}")`: Prints the accuracy of the model rounded to two decimal places.

```
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.6609 - loss: 0.6175
Epoch 2/10
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9496 - loss: 0.3162
Epoch 3/10
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9555 - loss: 0.1895
Epoch 4/10
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9542 - loss: 0.1561
Epoch 5/10
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9661 - loss: 0.1083
Epoch 6/10
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9685 - loss: 0.1037
Epoch 7/10
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9746 - loss: 0.0887
Epoch 8/10
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9888 - loss: 0.0612
Epoch 9/10
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9830 - loss: 0.0608
Epoch 10/10
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9752 - loss: 0.0825
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step

ANN Results:
Accuracy: 0.98
```

12. **Visualizing the Confusion Matrix**

```
# Visualization 2: Confusion Matrix for ANN
plt.figure(figsize=(8, 6))
sns.heatmap(ann_conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=data.target_names, yticklabels=data.target_names)
plt.title('ANN Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

`sns.heatmap()` : It creates a heatmap of the confusion matrix, where the color intensity represents the count of predictions.

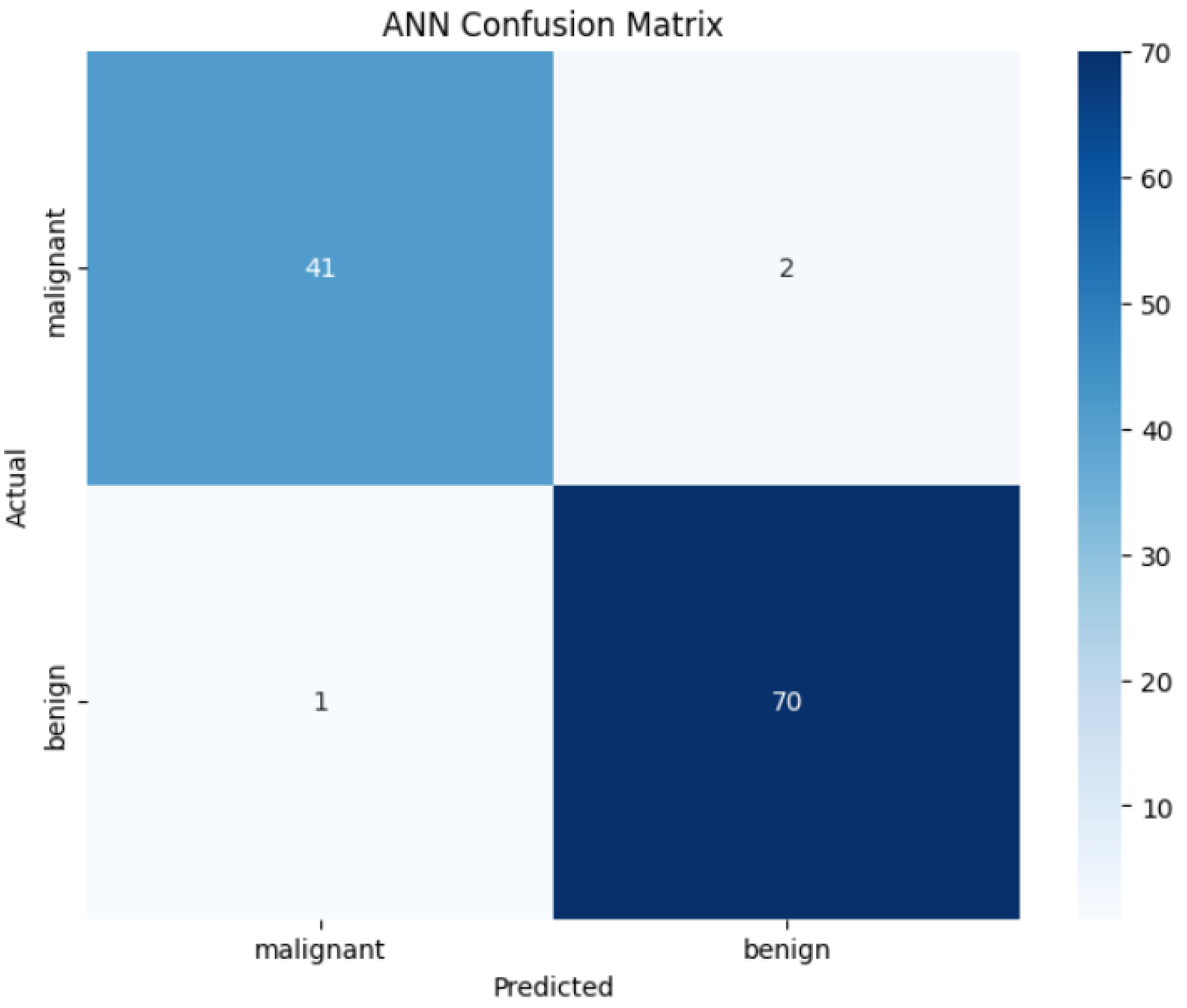
`annot=True` : Displays the actual numbers inside each matrix cell.

`fmt='d'` : Formats the numbers as integers.

`cmap='Blues'` : Uses a blue color scheme to show the matrix values.

`xticklabels` **and** `yticklabels` : Labels the x-axis and y-axis with the actual and predicted class names (Benign, Malignant).

`plt.show()` : Displays the heatmap.


ANN Confusion Matrix
41
2
1
70
malignant
benign
Actual
malignant
benign
Predicted
70
60
50
40
30
20
10

Summary:

1. **Data Loading**: Loads the Breast Cancer dataset and separates the features (X) and target labels (y).
2. **Data Preprocessing**: Standardizes the features for better performance during training.
3. **Model Building**: Constructs a neural network with 1 input layer, 1 hidden layer, and 1 output layer.
4. **Model Training**: Trains the model for 10 epochs using the training data.
5. **Evaluation**: Makes predictions on the test set and evaluates the accuracy of the model.

ML MODEL VS DL MODEL

1. **Higher Accuracy in DL Model**:
   1. The **Deep Learning (DL) model** performs better with an accuracy of **98%**, compared to the **95%** of the **Machine Learning (ML) model**.
   2. This indicates that the DL model is slightly more effective in making correct predictions.
2. **Model Complexity**:
   1. The **ML model** (e.g., Decision Trees, Random Forest) is simpler, which means it can perform well but may not capture complex patterns in the data.
   2. On the other hand, the **DL model** (Artificial Neural Network) has more layers and neurons, enabling it to learn and generalize better from the data, especially when the data is large and complex.
3. **Data Requirements**:
   1. **ML models** generally perform well with smaller datasets and require less computational power.
   2. **DL models**, while offering higher accuracy, typically require larger datasets and more computational resources to train effectively. However, once trained, they can outperform traditional ML models.

4. Generalization:

1. The **DL model** likely has better **generalization**, meaning it can better predict unseen data (test data) compared to the **ML model**

5. Trade-offs:

1. **ML models** are easier to train, require fewer resources, and are often faster, but may not handle very complex problems as well as **DL models**.
2. **DL models** can achieve higher accuracy, but they are computationally expensive and may take longer to train.

- In conclusion, while the **DL model** achieves a higher accuracy, **ML models** are often more practical for simpler tasks or when computational resources are limited.