**ACTIVITY-10**

**Comparison Between Machine Learning (ML) and Deep Learning (DL)**

**1. Definition:**

Machine Learning (ML) is a subset of Artificial Intelligence (AI) that focuses on building models using statistical methods. These models learn from data and improve their performance over time without being explicitly programmed.

**2. Data Dependency:**

ML models work well even with smaller datasets. As more data is provided, their performance increases, but they often hit a plateau after a certain point. ML is generally more suitable for structured datasets.

**3. Feature Engineering:**

ML often requires manual feature extraction and selection, which means human expertise is needed to identify the most relevant features for the model.

**4. Complexity:**

ML models tend to be simpler, and algorithms like Decision Trees, Support Vector Machines (SVM), and Logistic Regression are relatively easy to understand and interpret.

**5. Computational Power:**

ML typically requires less computational power and can run on standard CPUs. It is more lightweight in terms of hardware requirements.

**REGRESSION -Rebuild with deep learning model:**

# Import necessary libraries

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import fetch\_california\_housing

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean\_squared\_error, accuracy\_score

from tensorflow import keras

from tensorflow.keras import layers

# Load the California Housing dataset

housing = fetch\_california\_housing()

X = housing.data

y = housing.target

# Split the dataset 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)

# Standardize the dataset

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# Build a simple deep learning regression model

model = keras.Sequential([

layers.Dense(64, activation='relu', input\_shape=[X\_train.shape[1]]),

layers.Dense(64, activation='relu'),

layers.Dense(1) # Output layer for regression])

# Compile the model

model.compile(optimizer='adam', loss='mse', metrics=['mae'])

# Train the model

history = model.fit(X\_train\_scaled, y\_train, validation\_split=0.2, epochs=50, batch\_size=32, verbose=1)

# Evaluate the model

loss, mae = model.evaluate(X\_test\_scaled, y\_test)

print(f"Test MAE: {mae}")

# Predict

y\_pred = model.predict(X\_test\_scaled)

# Calculate Root Mean Squared Error (RMSE)

rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred))

print(f"Test RMSE: {rmse}")

# Define a threshold for accuracy (e.g., consider prediction correct if within 0.5 units of true value)

threshold = 0.5

y\_pred\_flat = y\_pred.flatten() # Flatten predictions to 1D

# Define accuracy as percentage of predictions within the threshold

accurate\_predictions = np.abs(y\_test - y\_pred\_flat) <= threshold

accuracy = np.mean(accurate\_predictions)

print(f"Accuracy (within {threshold}): {accuracy \* 100:.2f}%")

# Plot training & validation loss values

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)

plt.plot(history.history['loss'], label='Train Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.title('Model Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend()

# Plot training & validation MAE values

plt.subplot(1, 2, 2)

plt.plot(history.history['mae'], label='Train MAE')

plt.plot(history.history['val\_mae'], label='Validation MAE')

plt.title('Model MAE')

plt.ylabel('MAE')

plt.xlabel('Epoch')

plt.legend()

plt.tight\_layout()

plt.show()

**OUTPUT:**

Epoch 1/50

413/413 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - loss: 1.7368 - mae: 0.9108 - val\_loss: 0.4668 - val\_mae: 0.4826

Epoch 2/50

413/413 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - loss: 0.4243 - mae: 0.4626 - val\_loss: 0.4079 - val\_mae: 0.4551

Epoch 3/50

413/413 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 0.3882 - mae: 0.4416 - val\_loss: 0.3899 - val\_mae: 0.4398

413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2479 - mae: 0.3431 - val\_loss: 0.3011 - val\_mae: 0.3752

Epoch 49/50

413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2569 - mae: 0.3500 - val\_loss: 0.3185 - val\_mae: 0.3821

Epoch 50/50

413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.2613 - mae: 0.3451 - val\_loss: 0.3051 - val\_mae: 0.3810

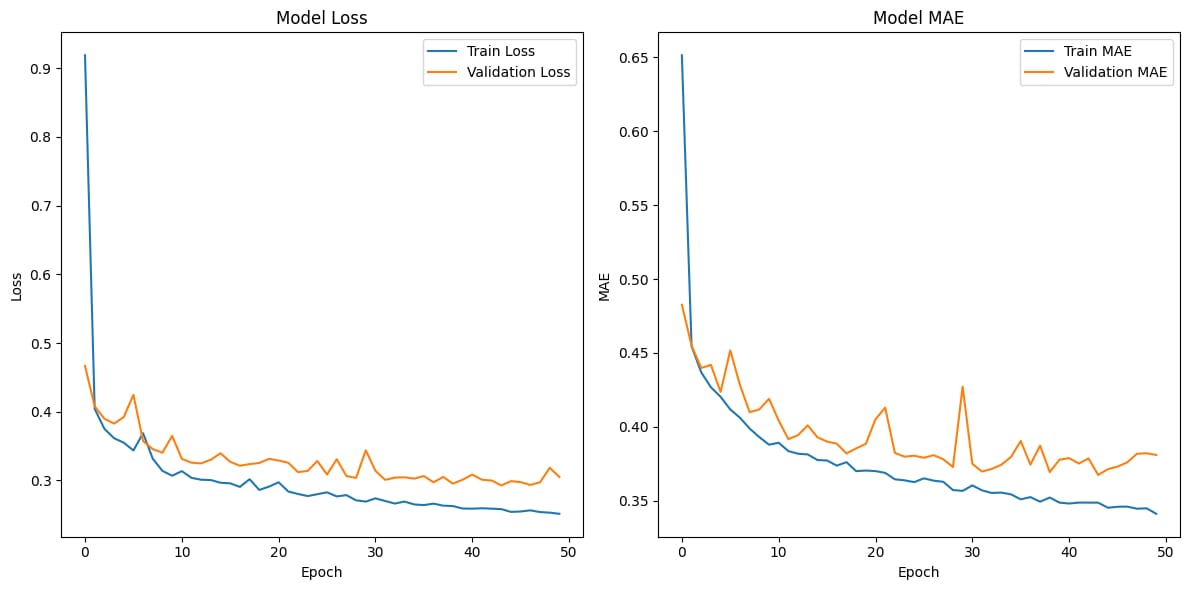
129/129 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.2785 - mae: 0.3679

Test MAE: 0.36704105138778687

129/129 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step

Test RMSE: 0.5329031890833387

Accuracy (within 0.5): 76.60%



**REGRESSION-Rebuild with machine learning model:**

# Import necessary libraries

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import fetch\_california\_housing

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

# Load the California Housing dataset

housing = fetch\_california\_housing()

X = housing.data

y = housing.target # Keep target as continuous values for regression

# Split the dataset 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)

# Standardize the dataset

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# Build a Random Forest Regression model

model = RandomForestRegressor(n\_estimators=100, random\_state=42)

# Train the model

model.fit(X\_train\_scaled, y\_train)

# Predict

y\_pred = model.predict(X\_test\_scaled)

# Evaluate the model

print("Mean Squared Error (MSE):", mean\_squared\_error(y\_test, y\_pred))

print("Mean Absolute Error (MAE):", mean\_absolute\_error(y\_test, y\_pred))

print("R² score:", r2\_score(y\_test, y\_pred))

# Optional: Plot true vs predicted values

plt.figure(figsize=(8, 6))

plt.scatter(y\_test, y\_pred, alpha=0.5)

plt.plot([min(y\_test), max(y\_test)], [min(y\_test), max(y\_test)], 'r--') # Reference line

plt.xlabel('True Values')

plt.ylabel('Predicted Values')

plt.title('True vs Predicted Values')

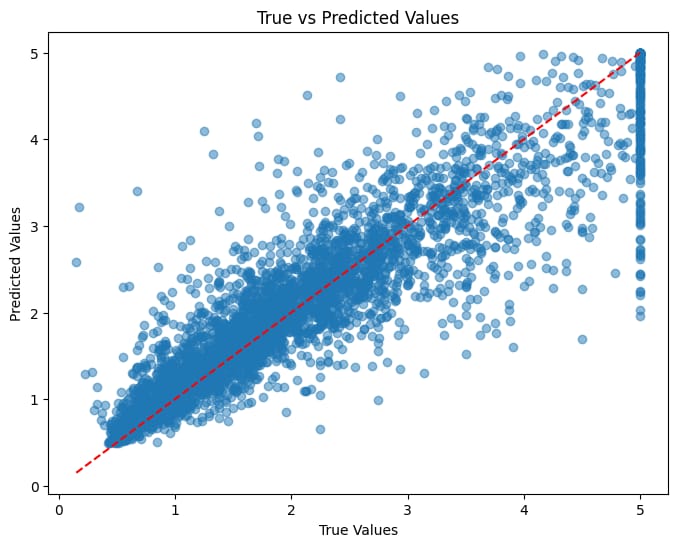
plt.show()

**OUTPUT:**

Mean Squared Error (MSE): 0.255169737347244

Mean Absolute Error (MAE): 0.3274252027374032

R² score: 0.8052747336256919

****

**CLASSIFICATION- Rebuild with deep learning model :**

# Import necessary libraries

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import fetch\_california\_housing

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score

from sklearn.preprocessing import KBinsDiscretizer

from tensorflow import keras

from tensorflow.keras import layers

# Load the California Housing dataset

housing = fetch\_california\_housing()

X = housing.data

y = housing.target

# Discretize the target variable (for classification)

# For example, we can use 3 bins: low, medium, high prices

discretizer = KBinsDiscretizer(n\_bins=3, encode='ordinal', strategy='uniform')

y\_binned = discretizer.fit\_transform(y.reshape(-1, 1)).flatten()

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_binned, test\_size=0.2, random\_state=42)

# Standardize the dataset

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# Build a simple deep learning classification model

model = keras.Sequential([

layers.Dense(64, activation='relu', input\_shape=[X\_train.shape[1]]),

layers.Dense(64, activation='relu'),

layers.Dense(3, activation='softmax') # Output layer for classification (3 classes)])

# Compile the model

model.compile(optimizer='adam',loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

# Train the model

history = model.fit(X\_train\_scaled, y\_train, validation\_split=0.2, epochs=50, batch\_size=32, verbose=1)

# Evaluate the model

loss, accuracy = model.evaluate(X\_test\_scaled, y\_test)

print(f"Test Accuracy: {accuracy \* 100:.2f}%")

# Predict

y\_pred = np.argmax(model.predict(X\_test\_scaled), axis=-1)

# Calculate the accuracy score

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy \* 100:.2f}%")

# Plot training & validation loss values

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)

plt.plot(history.history['loss'], label='Train Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.title('Model Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend()

# Plot training & validation accuracy values

plt.subplot(1, 2, 2)

plt.plot(history.history['accuracy'], label='Train Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Model Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend()

plt.tight\_layout()

plt.show()

**OUTPUT:**

413/413 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.6435 - loss: 0.7701 - val\_accuracy: 0.7560 - val\_loss: 0.6185

Epoch 2/50

413/413 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7635 - loss: 0.5603 - val\_accuracy: 0.7632 - val\_loss: 0.6125

Epoch 3/50

413/413 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.7713 - loss: 0.5328 - val\_accuracy: 0.7611 - val\_loss: 0.6100

Epoch 4/50

413/413 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.7787 - loss: 0.5205 - val\_accuracy: 0.7596 - val\_loss: 0.6336

Epoch 5/50

413/413 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.7843 - loss: 0.5127 - val\_accuracy: 0.7772 - val\_loss: 0.5796

Epoch 6/50

413/413 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.7875 - loss: 0.4911 - val\_accuracy: 0.7723 - val\_loss: 0.5768

Epoch 36/50

413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8246 - loss: 0.4190 - val\_accuracy: 0.8111 - val\_loss: 0.4775

Epoch 37/50

413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8273 - loss: 0.4041 - val\_accuracy: 0.8071 - val\_loss: 0.4744

Epoch 38/50

413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8314 - loss: 0.4066 - val\_accuracy: 0.8099 - val\_loss: 0.4642

Epoch 39/50

413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8294 - loss: 0.4075 - val\_accuracy: 0.8068 - val\_loss: 0.4551

Epoch 40/50

413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8258 - loss: 0.4040 - val\_accuracy: 0.8053 - val\_loss: 0.4550

Epoch 41/50

413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8254 - loss: 0.4115 - val\_accuracy: 0.8056 - val\_loss: 0.4673

Epoch 42/50

413/413 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8281 - loss: 0.4066 - val\_accuracy: 0.8117 - val\_loss: 0.4504

Epoch 43/50

413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8351 - loss: 0.3989 - val\_accuracy: 0.8153 - val\_loss: 0.4538

Epoch 44/50

413/413 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8276 - loss: 0.4073 - val\_accuracy: 0.8174 - val\_loss: 0.4530

Epoch 45/50

413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8320 - loss: 0.3963 - val\_accuracy: 0.8108 - val\_loss: 0.4598

Epoch 50/50

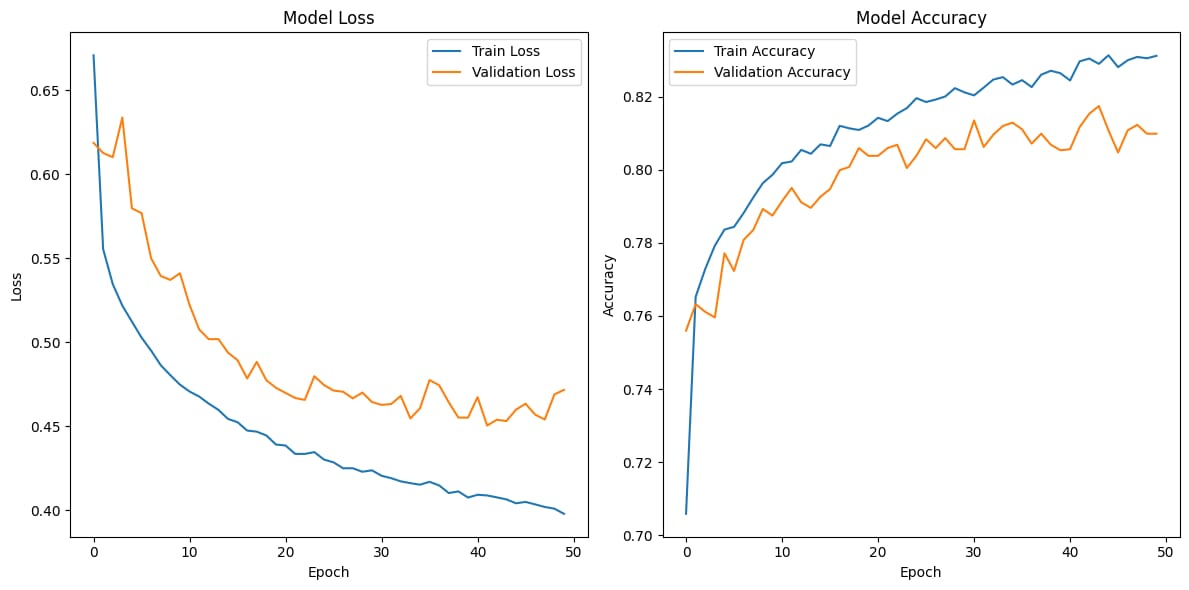
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8321 - loss: 0.3966 - val\_accuracy: 0.8099 - val\_loss: 0.4716

129/129 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8027 - loss: 0.4938

Test Accuracy: 80.74%

129/129 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step

Accuracy: 80.74%



**CLASSIFICATION-Rebuild with machine learning model:**

# Import necessary libraries

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import fetch\_california\_housing

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report, confusion\_matrix

# Load the California Housing dataset

housing = fetch\_california\_housing()

X = housing.data

y = housing.target

# Convert continuous target to discrete classes (e.g., low, medium, high)

# Here, we define thresholds for classification

bins = [0, 1.5, 2.5, 5.0] # Define bins based on your needs

labels = [0, 1, 2] # Corresponding labels for each bin

y\_class = np.digitize(y, bins) - 1 # Convert to classes

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_class, test\_size=0.2, random\_state=42)

# Standardize the dataset

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# Build a Random Forest Classification model

model = RandomForestClassifier(n\_estimators=100, random\_state=42)

# Train the model

model.fit(X\_train\_scaled, y\_train)

# Predict

y\_pred = model.predict(X\_test\_scaled)

# Evaluate the model

print("Classification Report:")

print(classification\_report(y\_test, y\_pred))

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

# Optional: Plotting the confusion matrix

plt.figure(figsize=(8, 6))

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

plt.imshow(conf\_matrix, interpolation='nearest', cmap=plt.cm.Blues)

plt.title('Confusion Matrix')

plt.colorbar()

tick\_marks = np.arange(len(np.unique(y\_class)))

plt.xticks(tick\_marks, np.unique(y\_class))

plt.yticks(tick\_marks, np.unique(y\_class))

plt.ylabel('True label')

plt.xlabel('Predicted label')

plt.show()

**OUTPUT:**

Classification Report:

precision recall f1-score support

0 0.84 0.88 0.86 1500

1 0.75 0.76 0.75 1497

2 0.77 0.74 0.75 947

3 0.82 0.51 0.63 184

accuracy 0.79 4128

macro avg 0.79 0.72 0.75 4128

weighted avg 0.79 0.79 0.79 4128

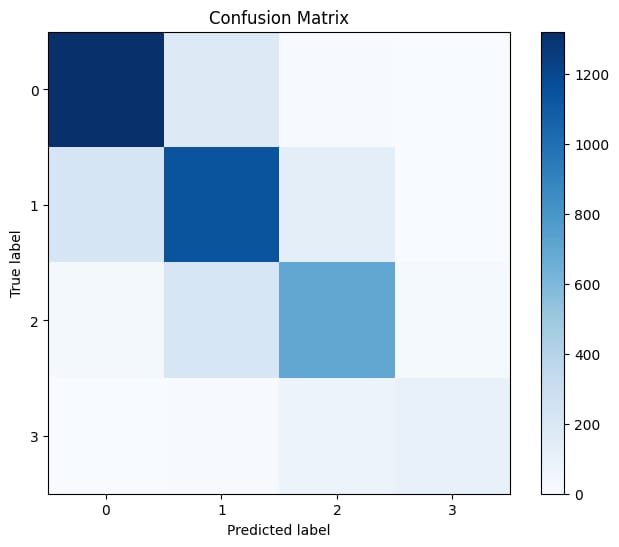
Confusion Matrix:

[[1322 171 7 0]

[ 224 1138 132 3]

[ 21 207 701 18]

[ 4 10 76 94]]

****