**ACTIVITY-10**

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

**1. Definition:**

Machine Learning (ML): ML refers to a subset of artificial intelligence (AI) that enables systems to learn from data and improve performance on tasks over time without being explicitly programmed. Traditional ML algorithms include linear regression, decision trees, and support vector machines (SVMs).

Deep Learning (DL): DL is a subset of ML that utilizes neural networks with many layers (deep neural networks) to automatically learn representations from data. DL excels in tasks like image recognition, natural language processing, and speech recognition.

**2. Data Dependency:**

ML: Traditional ML algorithms usually perform well with small to medium-sized datasets, and often require manual feature engineering to improve performance.

DL: DL models require large amounts of data to perform effectively, as deep neural networks are highly data-hungry. They automatically learn feature representations from raw data, removing the need for manual feature extraction.

**3. Computation Power:**

ML: ML algorithms are generally less computationally intensive compared to DL, and can be run on standard CPUs.

DL: DL models are computationally expensive, especially when working with large datasets. They often require specialized hardware, like GPUs or TPUs, to train efficiently.

**4. Feature Engineering:**

ML: Requires significant manual feature engineering, where domain knowledge is applied to identify and create the most relevant features for a given problem.

DL: Automatically performs feature learning through layers of the neural network, so little to no feature engineering is required.

**5. Performance:**

ML: Performs well on simpler tasks and smaller datasets. For complex tasks like image recognition or natural language processing, ML algorithms often plateau in performance.

DL: DL tends to outperform ML in complex tasks involving unstructured data, such as images, video, and text. With more data and computational power, DL models continue to improve.

**6. Interpretability:**

ML: Many traditional ML models (e.g., decision trees, linear regression) are more interpretable and can provide insight into the relationships between input features and the output.

DL: DL models, particularly deep neural networks, are often referred to as "black boxes" because they are difficult to interpret and explain in terms of how they make decisions.

**7. Applications:**

ML: Common applications include email filtering, fraud detection, recommendation systems, and predictive maintenance.

DL: Applications include image classification (e.g., in medical imaging), natural language processing (e.g., chatbots, translation), autonomous driving, and speech recognition.

**8. Training Time:**

ML: Training time is typically faster for traditional ML algorithms, especially on small to moderate datasets.

DL: DL models take longer to train due to the complexity of the neural networks and the large datasets involved, although using GPUs or distributed computing can reduce this time.

**9. Human Effort:**

ML: Requires more human effort in designing and tuning the model, especially in the feature engineering phase.

DL: Requires less human effort for feature engineering but more expertise in architecture design, hyperparameter tuning, and managing large-scale data and computation resources.

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

**Problem Statement: California Housing Price Prediction Using a Deep Learning Model**

**Objective:**

The goal of this project is to build a deep learning model using TensorFlow and Keras to predict housing prices based on the California Housing dataset. The model should be trained to minimize the prediction error for house prices and evaluated on a test set for performance.

**Dataset:**

California Housing Dataset: This dataset contains information about housing prices and various features like the median income, population, and number of households in various districts of California.

Features (X): Numerical attributes describing the properties of the districts, such as median income, total rooms, total bedrooms, population, households, median age, and latitude/longitude.

Target (y): The median house price for each district.

**Program:**

# 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

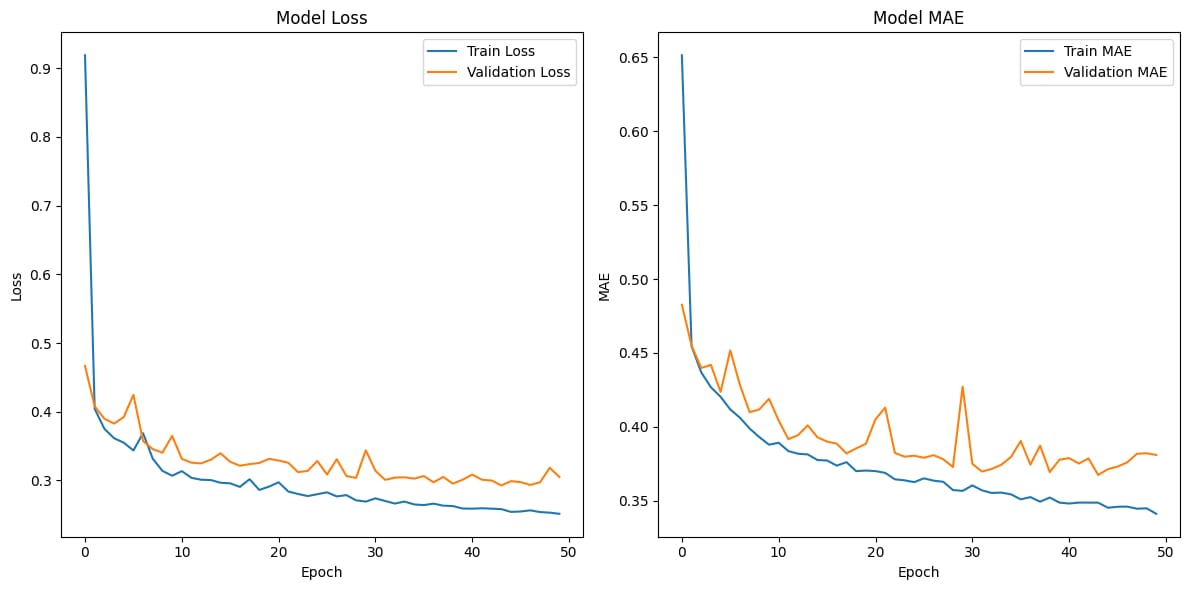
129/129 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.2785 - mae: 0.3679

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:**

**Problem Statement: python program implements a Random Forest Regression model using the California Housing Dataset.**

**Objective:**

The goal is to predict housing prices in California using a machine learning regression model. The model should effectively learn from historical data to provide accurate price predictions based on various features of the housing data.

**Program:**

# 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')

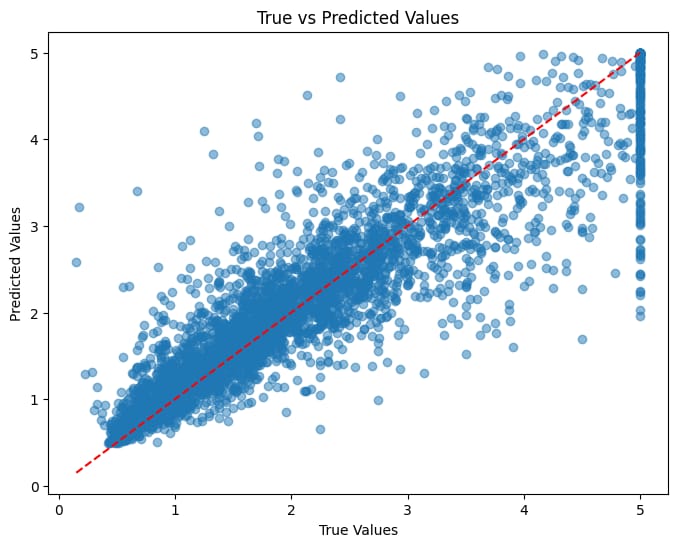
plt.show()

**OUTPUT:**

Mean Squared Error (MSE): 0.255169737347244

Mean Absolute Error (MAE): 0.3274252027374032

R² score: 0.8052747336256919

****

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

**Problem Statement: California Housing Price Prediction Using a Deep Learning Model**

**Objective:**

To develop a classification model that predicts housing price categories based on various features of the California Housing dataset.

**Dataset:**

The dataset contains features such as the number of rooms, population, and median income for different California districts, along with a continuous target variable representing median house prices.

**Program:**

# 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 6/50

Epoch 50/50

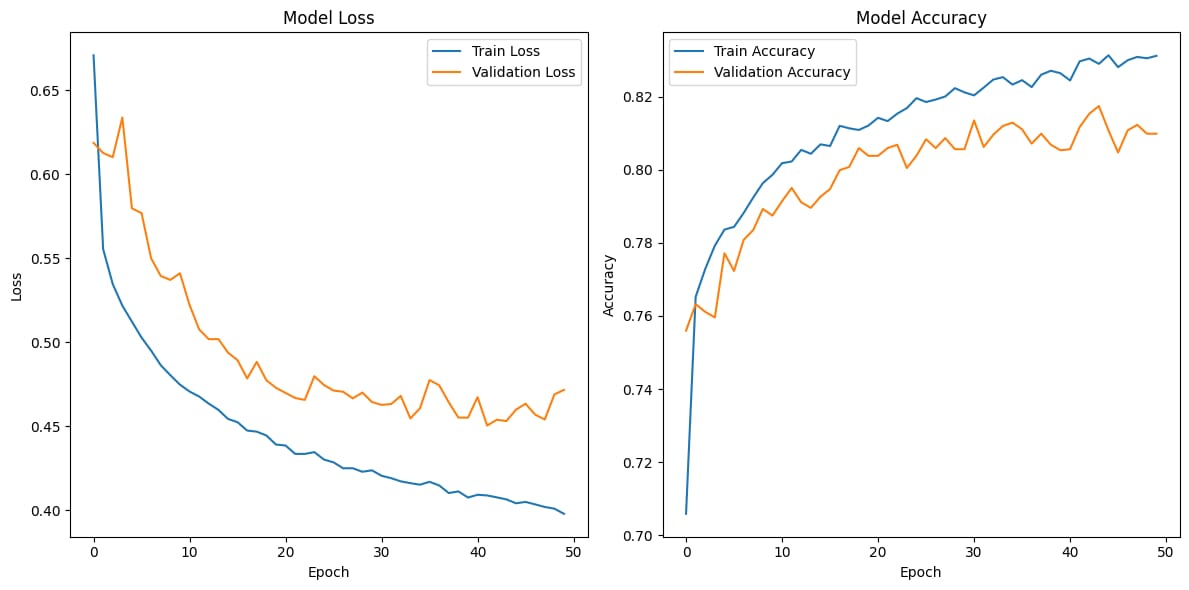
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8321 - loss: 0.3966 - val\_accuracy: 0.8099 - val\_loss: 0.4716

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:**

**Problem Statement: python program implements a Random Forest Classifier model using the California Housing Dataset.**

**Objectives:**

**Data Preparation:** Load the California Housing dataset and convert the continuous target variable (median house value) into discrete classes using specified thresholds.

**Data Splitting:** Split the dataset into training and testing subsets to evaluate model performance.

**Data Standardization:** Scale the features to improve the model's performance.

**Model Development:** Build a Random Forest Classifier to categorize the housing data into the defined classes.

**Model Evaluation:** Assess the model's performance using classification metrics, including the classification report and confusion matrix.

**Program:**

# 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

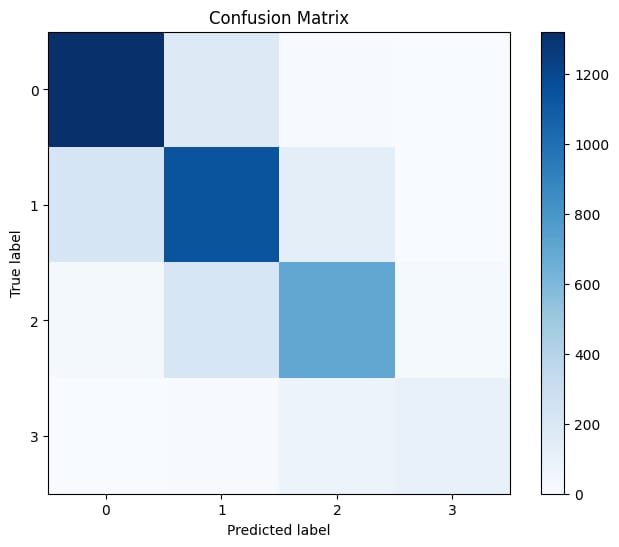
Confusion Matrix:

[[1322 171 7 0]

[ 224 1138 132 3]

[ 21 207 701 18]

[ 4 10 76 94]]

****

**Analyze the performance of ML and DL**

**1**. **Handling Complex Patterns:** Deep learning models, particularly neural networks, can capture complex patterns and relationships in large datasets that traditional algorithms may struggle with.

**2. Feature Learning:** Deep learning can automatically extract and learn features from raw data (e.g., images, text), reducing the need for manual feature engineering.

**3**. **Scalability**: Deep learning models often scale better with large datasets. As the amount of data increases, their performance typically improves.

**4**. **Performance on Unstructured Data:** Deep learning excels at processing unstructured data (e.g., images, audio, text) due to its architecture, which is designed to work with high-dimensional input.

**5. State-of-the-Art Results:** For many tasks, such as image recognition, natural language processing, and speech recognition, deep learning models have achieved state-of-the-art performance.

**6. Transfer Learning:** Pre-trained deep learning models can be fine-tuned for specific tasks, making them more efficient in terms of training time and data requirements.

**7. End-to-End Learning:** Deep learning allows for end-to-end learning, meaning that the entire pipeline (feature extraction to prediction) can be trained simultaneously, which can improve overall performance.

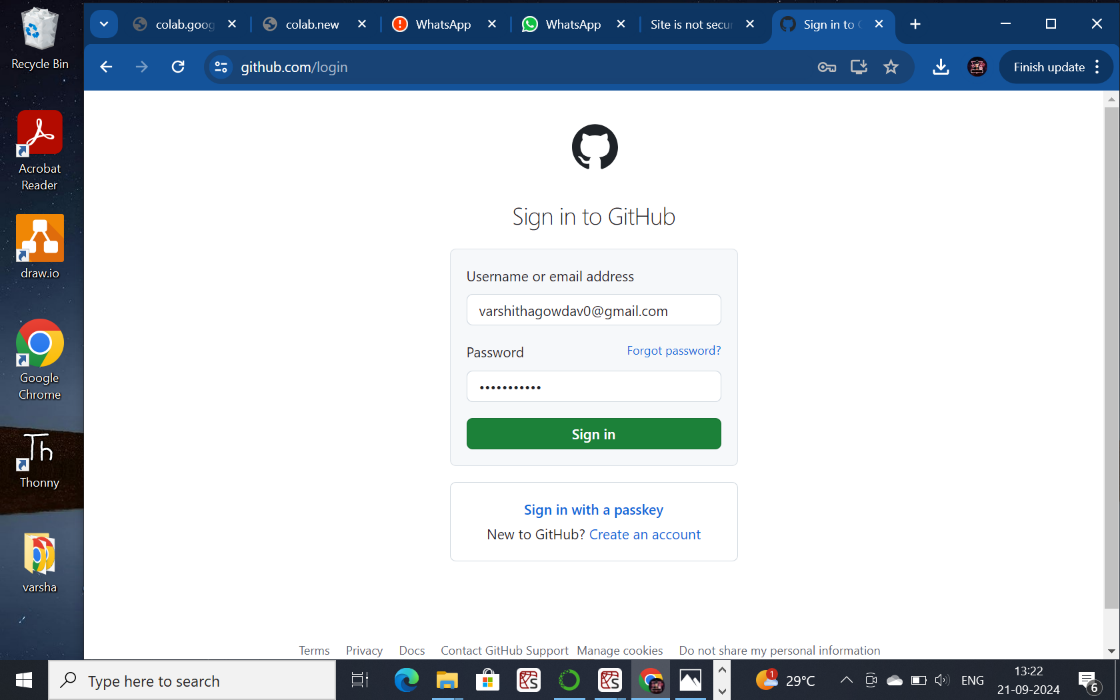
**8**. **Robustness:** Deep learning models can be more robust to noise and variations in data, as they can learn high-level abstractions.

**9. Advanced Architectures**: Deep learning allows for the use of advanced architectures like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), which are tailored for specific types of data.

**10. Continuous Improvement:** With ongoing advancements in deep learning techniques and architectures, models are continually improving, leading to better performance in various applications.

**Upload documents to GitHub, follow these steps:**

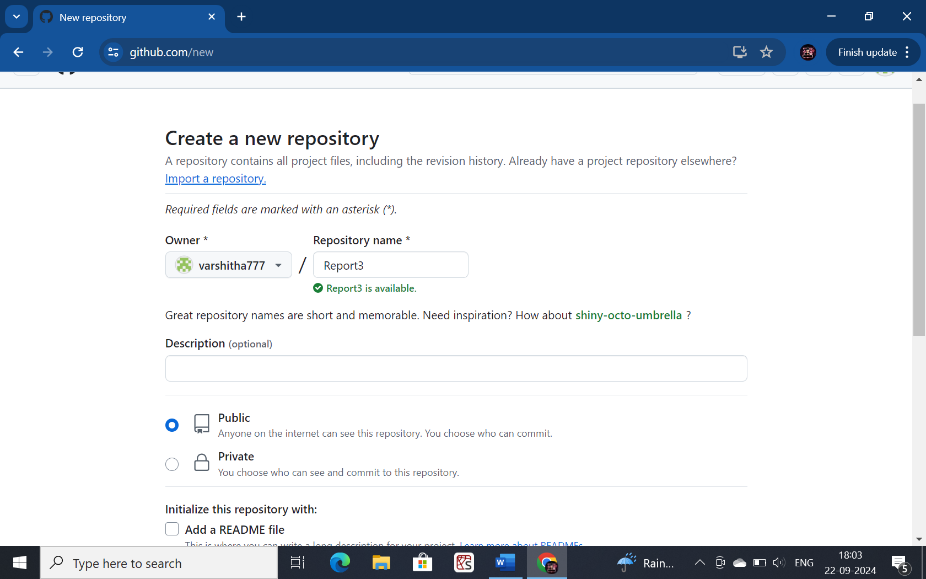
**Step1:** \*Sign in\* to your GitHub account.

****

**Step 2:** 1. Create a Repository

Click on the \*+ icon\* in the upper right corner and select \*New repository\*.

Fill in the repository name, description, and choose whether it will be public or private.

****

**Upload Documents**

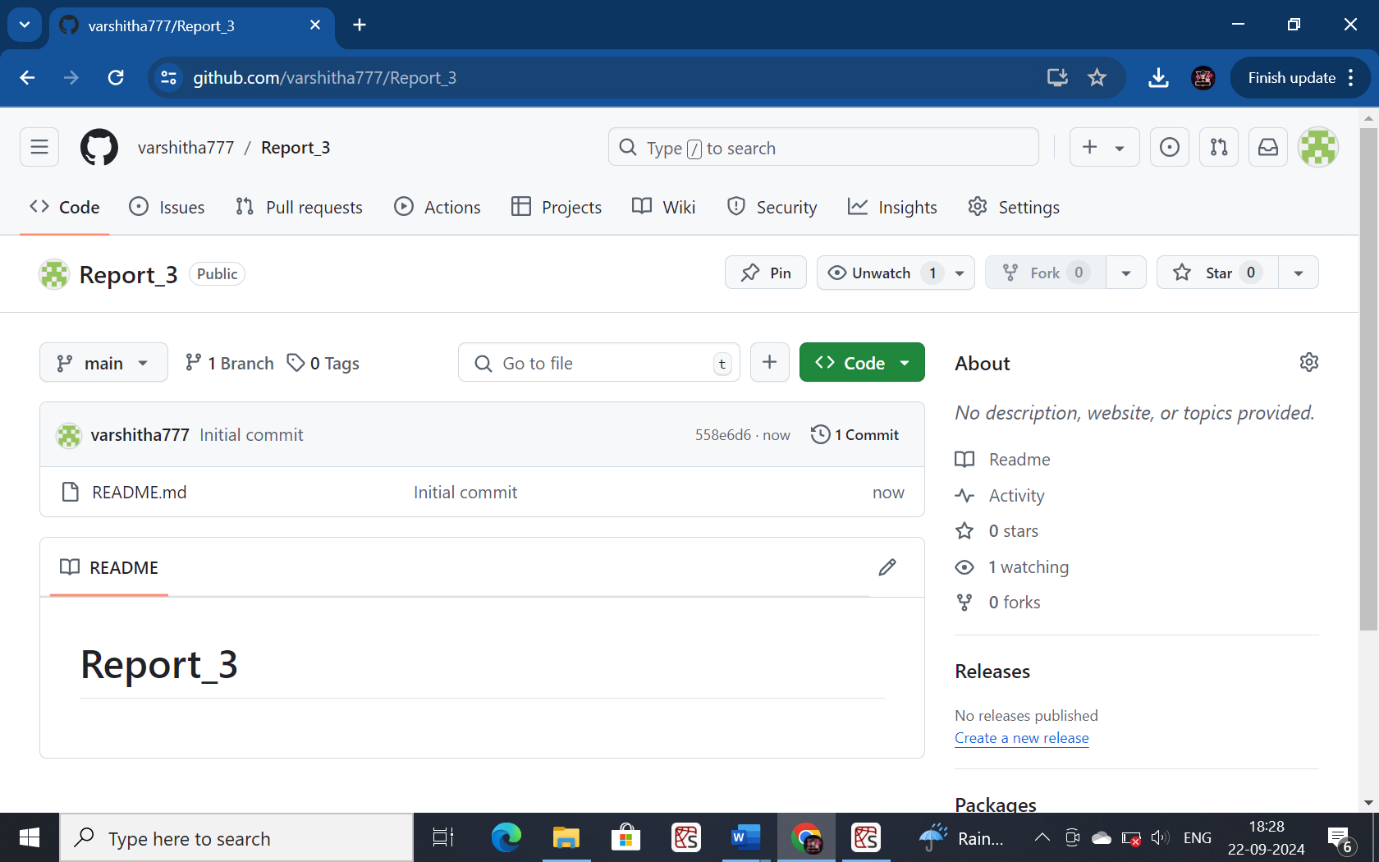
\*Method 1: Using GitHub Website\*

**Step3 :** 1. Go to your repository.

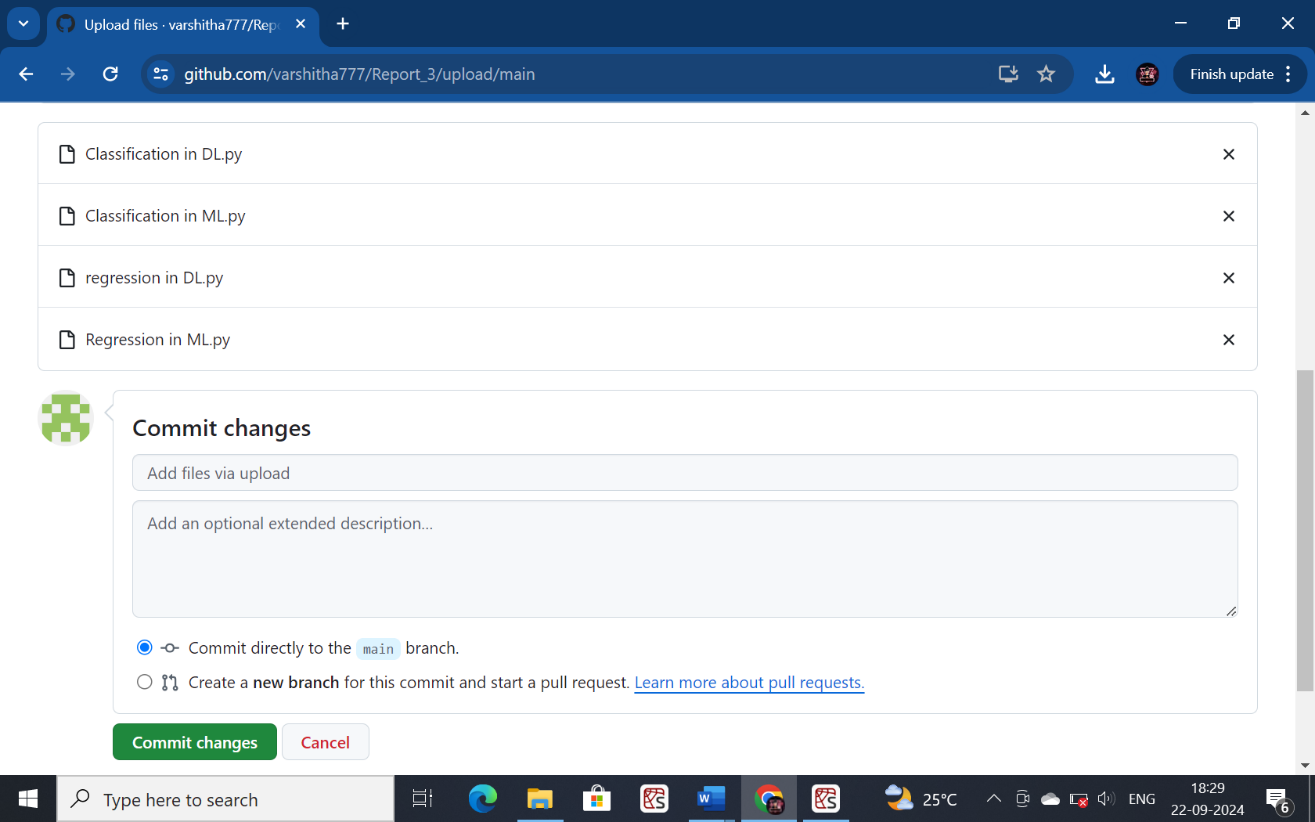
2. Click on the \*Add file\* button, then select \*Upload files\*.

3. Drag and drop your documents or click \*choose your files\* to select them from your computer

4. Add a commit message describing your upload..



**Step 4:** Click \*Commit changes\*.

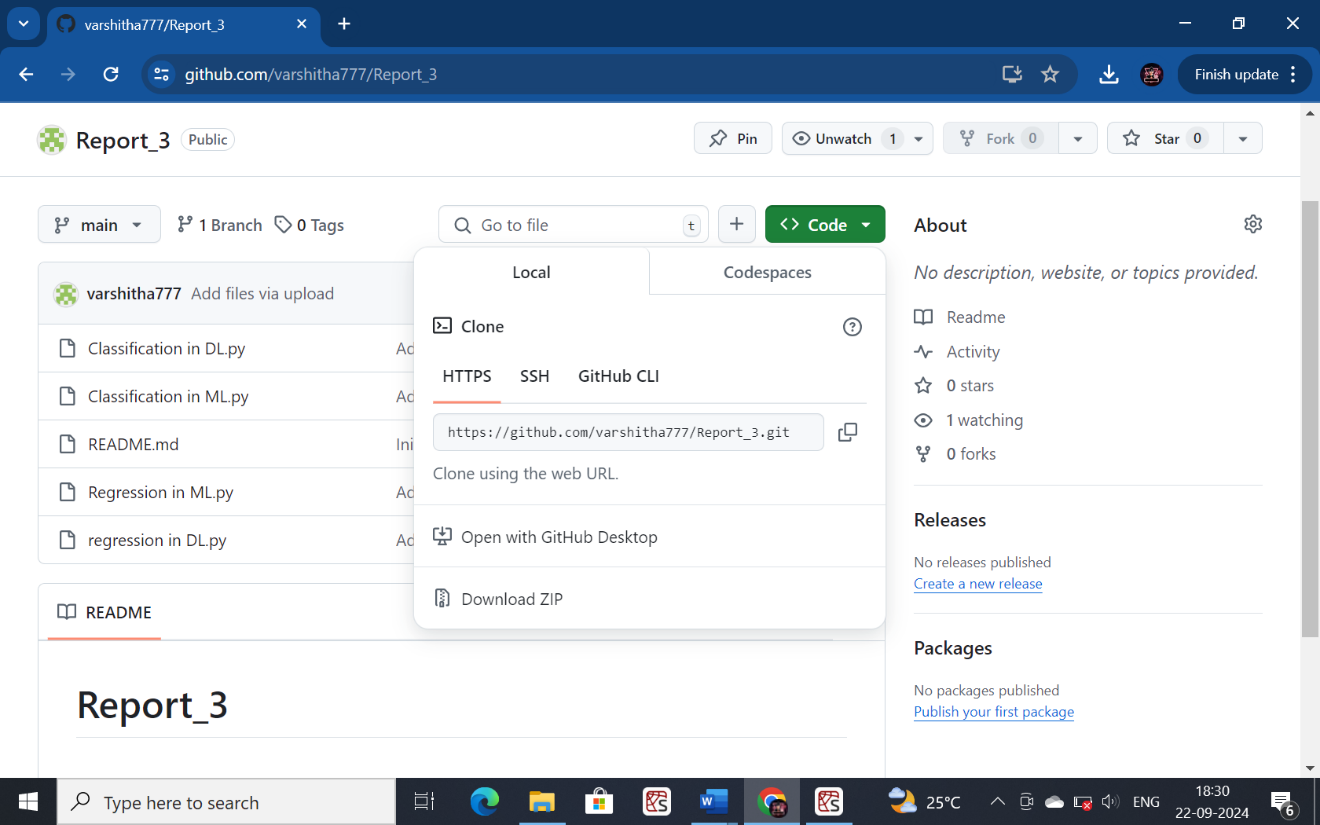
****

**To copy and share the link to your GitHub repository or a specific file, follow these steps:**

**Copying the Link to Your Repository**

1. \*Open your repository\* on GitHub.

2. \*Copy the URL\* from the address bar of your browser. It will look something like <https://github.com/varshitha777/Report_3.git>

****

**Sharing the Link**

- \*Paste the link\* wherever you want to share it (e.g., email, chat, social media).

- You can use URL shorteners (like Bitly) for cleaner links if necessary.

That's it! You can now easily share your GitHub repository or specific files

Paste the copied URL wherever you want to share it-via email,chat,or on social media.

