**DEEP LEARNING LAB PROGRAMS**

**1. Setting up the spyder IDE environment and executing a python program.**

**Program:**

How to Install the Python Spyder IDE:

**Step 1: Installing Spyder using pip**

* Open the terminal/command prompt on your system.
* Type the following command to install Spyder using pip:

**Cmd : pip install Spyder**

* Once installed, you can open Spyder IDE by typing the following command in the terminal/command prompt:

**Cmd : spyder**

**Step 2: Running Python Scripts in Spyder IDE**

Now that you have installed Spyder IDE on your system, you can use it to run Python scripts. This section will show you how to create a new Python script in Spyder IDE, write some code, and run it.

* Launch Spyder IDE.
* Go to "File" > "New File" to create a new Python script.
* Write some code in the script editor. For example, you can write a simple program:

**Cmd: print("SNIST")**

* Save the script to a file with the ".py" extension. For example, you can save it as "hello.py".
* To run the script, go to "Run" > "Run the file" or press F5. This will execute the Python script and display the output in the console.
* You can also use the console to run individual lines of code. To do this, type the code in the console and press Enter. The output will be displayed in the console.

**2. Installing Keras, Tensorflow, and Pytorch libraries and making use of them.**

**Program:**

**1. Opening Google Colab**

**Step 1**: Open your browser and go to Google Colab.

**Step 2**: Click on **"New Notebook"** to create a new notebook.

**2. Installing Libraries in Colab**

Google Colab comes pre-installed with TensorFlow, Keras, and PyTorch.

Install TensorFlow

**Cmd**: !pip install tensorflow

**Cmd**: !pip install torch torchvision torchaudio

**3. Using TensorFlow/Keras on Google Colab**

**3.1 Writing a Basic Neural Network with Keras**

**Step 1**: Import the required libraries and load the MNIST dataset.

Python

**Program:**

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import mnist

# Load dataset

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

**Step 2**: Build and compile a simple neural network model.

**Program:**

# Create a Sequential model

model = models.Sequential([

layers.Flatten(input\_shape=(28, 28)),

layers.Dense(128, activation='relu'),

layers.Dense(10, activation='softmax')

])

# Compile the model

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

**Step 3**: Train and evaluate the model.

**Program:**

# Train the model

model.fit(x\_train, y\_train, epochs=5)

# Evaluate the model

Test\_loss, test\_acc = model.evaluate(x\_test, y\_test)

print(f"Test accuracy: {test\_acc}")

**4. Using PyTorch on Google Colab**

**4.1 Writing a Basic Neural Network in PyTorch**

**Step 1:** Import the necessary libraries and load the MNIST dataset.

**Program:**

import torch

import torch.nn as nn

import torch.optim as optim

from torchvision import datasets, transforms

from torch.utils.data import DataLoader

# Define transformations

transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))])

# Load dataset

train\_dataset = datasets.MNIST(root='./data', train=True, transform=transform, download=True)

train\_loader = DataLoader(dataset=train\_dataset, batch\_size=64, shuffle=True)

**Step 2:** Build a simple feedforward neural network model.

**Program:**

class SimpleNN(nn.Module):

def \_\_init\_\_(self):

super(SimpleNN, self).\_\_init\_\_()

self.flatten = nn.Flatten()

self.fc1 = nn.Linear(28\*28, 128)

self.fc2 = nn.Linear(128, 10)

def forward(self, x):

x = self.flatten(x)

x = torch.relu(self.fc1(x))

x = self.fc2(x)

return x

# Initialize model

model = SimpleNN()

**Step 3**: Define the loss function and optimizer.

**Program:**

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=0.001)

**Step 4**: Train the model.

**Program:**

# Training loop

for epoch in range(5):

for batch in train\_loader:

inputs, labels = batch

outputs = model(inputs)

loss = criterion(outputs, labels)

optimizer.zero\_grad()

loss.backward()

optimizer.step()

print(f"Epoch {epoch+1}, Loss: {loss.item()}")

**Step 5**: Run the cell to train the model and observe the loss during each epoch.

To accelerate your model training, enable GPU in Colab:

**Step 1**: Click on **Runtime** in the menu bar.

**Step 2**: Select **Change runtime type**.

**Step 3**: Under **Hardware accelerator**, choose **GPU**.

**Step 4**: Click **Save**.

**3. Applying the Convolution Neural Network on computer vision problems.**

**Step 1:** Import Required Libraries

Start by importing TensorFlow and necessary modules.

**Program:**

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import mnist

**Step 2:** Load and Preprocess the Data

Load the MNIST dataset and normalize the pixel values to be between 0 and 1.

**Program:**

# Load the dataset

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

# Reshape the data to add a single channel (since images are grayscale)

x\_train = x\_train.reshape((x\_train.shape[0], 28, 28, 1))

x\_test = x\_test.reshape((x\_test.shape[0], 28, 28, 1))

# Normalize the pixel values to be between 0 and 1

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

**Step 3:** Define a Simple CNN Model

Create a simple CNN with one convolutional layer followed by max pooling, and a dense layer for classification.

**Program:**

# Define a basic CNN model

model = models.Sequential([

layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)), # 32 filters, 3x3 kernel

layers.MaxPooling2D((2, 2)), # Pooling layer with 2x2 pool size

layers.Flatten(), # Flatten the output from the convolutional layer

layers.Dense(64, activation='relu'), # Fully connected layer with 64 neurons

layers.Dense(10, activation='softmax') # Output layer for 10 classes

])

**Step 4:** Compile the Model

Compile the model using the Adam optimizer, a suitable loss function, and accuracy as the metric.

**Program:**

# Compile the model

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

**Step 5:** Train the Model

Train the CNN model on the training dataset for 5 epochs.

**Program:**

# Train the model

model.fit(x\_train, y\_train, epochs=5, batch\_size=64)

**Step 6:** Evaluate the Model

After training, evaluate the model’s performance on the test dataset.

**Program:**

# Evaluate the model

test\_loss, test\_acc = model.evaluate(x\_test, y\_test)

print(f'Test accuracy: {test\_acc}')

**4. Image classification on MNIST dataset (CNN model with a fully connected layer).**

**Step 1:** Import Required Libraries

Start by importing TensorFlow and necessary modules.

**Program:**

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import mnist

**Step 2: Load and Preprocess the Data**

Load the MNIST dataset and normalize the pixel values to be between 0 and 1.

**Program:**

# Load the dataset

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

# Reshape the data to add a single channel (since images are grayscale)

x\_train = x\_train.reshape((x\_train.shape[0], 28, 28, 1))

x\_test = x\_test.reshape((x\_test.shape[0], 28, 28, 1))

# Normalize the pixel values to be between 0 and 1

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

**Step 3: Define a Simple CNN Model**

Create a simple CNN with one convolutional layer followed by max pooling, and a dense layer for classification.

**Program:**

# Define a basic CNN model

model = models.Sequential([

# First convolutional layer layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),

layers.MaxPooling2D((2, 2)), # Second convolutional layer

layers.Conv2D(64, (3, 3), activation='relu'),

layers.MaxPooling2D((2, 2)), # Third convolutional layer

layers.Conv2D(64, (3, 3), activation='relu'),

# Flatten the output from convolutional layers

layers.Flatten(),

# Fully connected layer (Dense layer)

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

# Output layer with softmax for classification (10 classes for digits 0-9) layers.Dense(10, activation='softmax')

])

**Step 4: Compile the Model**

Compile the model using the Adam optimizer, a suitable loss function, and accuracy as the metric.

**Program:**

# Compile the model

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

**Step 5: Train the Model**

Train the CNN model on the training dataset for 5 epochs.

**Program:**

# Train the model

model.fit(x\_train, y\_train, epochs=5, batch\_size=64)

**Step 6: Evaluate the Model**

After training, evaluate the model’s performance on the test dataset.

**Program:**

# Evaluate the model

test\_loss, test\_acc = model.evaluate(x\_test, y\_test)

print(f'Test accuracy: {test\_acc}')

**Step 7: Make Predictions**

You can also use the model to predict the digit class of a test image.

**Program:**

# Make predictions on the test dataset

predictions = model.predict(x\_test)

# Get the predicted class for the first test image

predicted\_label = tf.argmax(predictions[0])

print(f'Predicted label: {predicted\_label.numpy()}')

**5. Applying the Deep Learning Models in the field of Natural Language Processing.**

**Step 1:** Import Required Libraries

We will use TensorFlow/Keras for building the model, and the IMDb dataset provided by Keras for the text data.

**Program:**

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import imdb

from tensorflow.keras.preprocessing import sequence

**Step 2:** Load and Preprocess the IMDb Dataset

**Program:**

# Load the IMDb dataset (only top 10,000 most frequent words)

num\_words = 10000

(x\_train, y\_train), (x\_test, y\_test) = imdb.load\_data(num\_words=num\_words)

# Set the maximum length of each review to 500 words (truncating or padding shorter/longer reviews)

max\_len = 500

x\_train = sequence.pad\_sequences(x\_train, maxlen=max\_len)

x\_test = sequence.pad\_sequences(x\_test, maxlen=max\_len)

**Step 3:** Build a Deep Learning Model for Text Classification

**Program:**

# Define the model

model = models.Sequential([

# Embedding layer: Converts words into dense vectors

layers.Embedding(input\_dim=num\_words, output\_dim=64, input\_length=max\_len),

# LSTM layer: Captures sequential patterns

layers.LSTM(64),

# Dense layer with sigmoid activation for binary classification

layers.Dense(1, activation='sigmoid')

])

# Compile the model

model.compile(optimizer='adam',

loss='binary\_crossentropy',

metrics=['accuracy'])

# Display the model architecture

model.summary()

**Step 4:** Train the Model

**Program:**

# Train the model

model.fit(x\_train, y\_train, epochs=5, batch\_size=64, validation\_split=0.2)

**Step 5:** Evaluate the Model

**Program:**

# Evaluate the model

test\_loss, test\_acc = model.evaluate(x\_test, y\_test)

print(f'Test accuracy: {test\_acc}')

**Step 6:** Make Predictions on New Text

**Program:**

# Example review (preprocessed as integer sequences)

new\_review = [1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 4581, 66, 394, 2, 530, 973]

new\_review = sequence.pad\_sequences([new\_review], maxlen=max\_len)

# Predict sentiment (1: Positive, 0: Negative)

prediction = model.predict(new\_review)

print(f'Predicted sentiment: {"Positive" if prediction >= 0.5 else "Negative"}')

**6. Train a sentiment analysis model on IMDB dataset, use RNN layers with LSTM/GRU notes.**

**Step 1:** Import Required Libraries

Start by importing TensorFlow, Keras, and necessary modules**.**

**Program:**

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import imdb

from tensorflow.keras.preprocessing import sequence

**Step 2:** Load and Preprocess the Data

Load the IMDb dataset and preprocess the reviews by limiting the vocabulary and padding the sequences.

**Program:**

# Load the IMDb dataset with a limit of 10,000 words

num\_words = 10000

(x\_train, y\_train), (x\_test, y\_test) = imdb.load\_data(num\_words=num\_words)

# Set the maximum length for each review

max\_len = 500

x\_train = sequence.pad\_sequences(x\_train, maxlen=max\_len)

x\_test = sequence.pad\_sequences(x\_test, maxlen=max\_len)

**Step 3:** Define the Model Architecture Using LSTM

**Program:**# Define the model using LSTM

model = models.Sequential([

layers.Embedding(input\_dim=num\_words, output\_dim=64, input\_length=max\_len),

layers.LSTM(64), # LSTM layer with 64 units

layers.Dense(1, activation='sigmoid') # Output layer for binary classification

])

# Compile the model

model.compile(optimizer='adam',

loss='binary\_crossentropy',

metrics=['accuracy'])

# Display the model architecture

model.summary()

**Step 4:** Train the Model

**Program:**

# Train the model

model.fit(x\_train, y\_train, epochs=5, batch\_size=64, validation\_split=0.2)

**Step 5:** Evaluate the Model on the Test Set

**Program:**

# Evaluate the model

test\_loss, test\_acc = model.evaluate(x\_test, y\_test)

print(f'Test accuracy: {test\_acc}')

**Step 6:** Example Prediction

**Program:**

# Example review (preprocessed as integer sequences)

new\_review = [1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 4581, 66, 394, 2, 530, 973]

new\_review = sequence.pad\_sequences([new\_review], maxlen=max\_len)

# Predict sentiment (1: Positive, 0: Negative)

prediction = model.predict(new\_review)

print(f'Predicted sentiment: {"Positive" if prediction >= 0.5 else "Negative"}')

**7. Applying the Auto encoder algorithms for encoding the real-world data.**

**Step 1:** Import Required Libraries

**Program:**

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow.keras import layers, models

from sklearn.datasets import load\_digits

**Step 2:** Load and Preprocess the Data

**Program:**

# Load the digits dataset

digits = load\_digits()

data = digits.data # Shape: (1797, 64)

# Normalize the data to the range [0, 1]

data = data / 16.0

# Visualize some samples

plt.figure(figsize=(10, 5))

for i in range(10):

plt.subplot(2, 5, i + 1)

plt.imshow(data[i].reshape(8, 8), cmap='gray')

plt.axis('off')

plt.show()

**Step 3:** Define the Autoencoder Architecture

**Program:**

# Define the Autoencoder model

input\_dim = data.shape[1] # Number of features (64 for 8x8 images)

# Encoder

input\_layer = layers.Input(shape=(input\_dim,))

encoded = layers.Dense(32, activation='relu')(input\_layer) # Compress to 32 dimensions

# Decoder

decoded = layers.Dense(input\_dim, activation='sigmoid')(encoded) # Reconstruct original dimensions

# Combine into an Autoencoder model

autoencoder = models.Model(input\_layer, decoded)

# Compile the model

autoencoder.compile(optimizer='adam', loss='binary\_crossentropy')

# Display the model architecture

autoencoder.summary()

**Step 4:** Train the Autoencoder

**Program:**

# Train the Autoencoder

autoencoder.fit(data, data, epochs=50, batch\_size=256, shuffle=True, validation\_split=0.2)

**Step 5:** Visualize the Results

**Program:**

# Encode and decode some digits

encoded\_data = autoencoder.predict(data)

# Plot original and reconstructed images

n = 10 # Number of images to display

plt.figure(figsize=(10, 5))

for i in range(n):

# Original images

ax = plt.subplot(2, n, i + 1)

plt.imshow(data[i].reshape(8, 8), cmap='gray')

plt.title("Original")

plt.axis('off')

# Reconstructed images

ax = plt.subplot(2, n, i + n + 1)

plt.imshow(encoded\_data[i].reshape(8, 8), cmap='gray')

plt.title("Reconstructed")

plt.axis('off')

plt.show()

**Step 6:** Extract Encoded Features

**Progam:**

# Create a model to extract the encoded features

encoder = models.Model(input\_layer, encoded)

# Get encoded features

encoded\_features = encoder.predict(data)

# Display the shape of the encoded features

print(f"Encoded features shape: {encoded\_features.shape}")

**8. Applying Generative Adversial Networks for image generation and unsupervised tasks.**

**Step 1:** Import Required Libraries

**Program:**

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import mnist

**Step 2:** Load and Preprocess the Data

**Program:**

# Load the MNIST dataset

(x\_train, \_), (x\_test, \_) = mnist.load\_data()

# Normalize the images to the range [0, 1]

x\_train = (x\_train.astype(np.float32) - 127.5) / 127.5 # Scale to [-1, 1]

# Reshape the data to (num\_samples, height, width, channels)

x\_train = np.expand\_dims(x\_train, axis=-1)

# Display some samples

plt.figure(figsize=(10, 5))

for i in range(10):

plt.subplot(2, 5, i + 1)

plt.imshow(x\_train[i].reshape(28, 28), cmap='gray')

plt.axis('off')

plt.show()

**Step 3:** Define the Generator and Discriminator Models

**Program:**

# Generator model

def build\_generator():

model = models.Sequential([

layers.Dense(256, activation='relu', input\_shape=(100,)),

layers.BatchNormalization(),

layers.Dense(512, activation='relu'),

layers.BatchNormalization(),

layers.Dense(1024, activation='relu'),

layers.BatchNormalization(),

layers.Dense(28 \* 28 \* 1, activation='tanh'), # Output layer

layers.Reshape((28, 28, 1))

])

return model

# Discriminator model

def build\_discriminator():

model = models.Sequential([

layers.Flatten(input\_shape=(28, 28, 1)),

layers.Dense(512, activation='relu'),

layers.Dense(256, activation='relu'),

layers.Dense(1, activation='sigmoid') # Binary classification (real or fake)

])

return model

generator = build\_generator()

discriminator = build\_discriminator()

# Compile the discriminator

discriminator.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

**Step 4:** Create the GAN Model

**Program:**

# Combine generator and discriminator into GAN model

discriminator.trainable = False # Freeze the discriminator during generator training

gan\_input = layers.Input(shape=(100,))

generated\_image = generator(gan\_input)

gan\_output = discriminator(generated\_image)

gan = models.Model(gan\_input, gan\_output)

gan.compile(optimizer='adam', loss='binary\_crossentropy')

**Step 5:** Train the GAN Model

**Progam:**

# Training parameters

epochs = 10000

batch\_size = 128

half\_batch = batch\_size // 2

# Lists to keep track of losses

d\_losses = []

g\_losses = []

for epoch in range(epochs):

# Train Discriminator

# Select a random half batch of real images

idx = np.random.randint(0, x\_train.shape[0], half\_batch)

real\_images = x\_train[idx]

# Generate a half batch of fake images

noise = np.random.normal(0, 1, (half\_batch, 100))

fake\_images = generator.predict(noise)

# Labels for real and fake images

real\_labels = np.ones((half\_batch, 1))

fake\_labels = np.zeros((half\_batch, 1))

# Train the discriminator

d\_loss\_real = discriminator.train\_on\_batch(real\_images, real\_labels)

d\_loss\_fake = discriminator.train\_on\_batch(fake\_images, fake\_labels)

d\_loss = 0.5 \* np.add(d\_loss\_real, d\_loss\_fake)

# Train Generator

noise = np.random.normal(0, 1, (batch\_size, 100))

valid\_labels = np.ones((batch\_size, 1)) # Try to fool the discriminator

# Train the generator

g\_loss = gan.train\_on\_batch(noise, valid\_labels)

# Save losses

d\_losses.append(d\_loss[0])

g\_losses.append(g\_loss)

# Print the progress

if epoch % 1000 == 0:

print(f"Epoch: {epoch}, D Loss: {d\_loss[0]}, G Loss: {g\_loss}")

**Step 6:** Generate Images Using the Trained Generator

**Program:**

# Generate images

def generate\_images(generator, n\_images=10):

noise = np.random.normal(0, 1, (n\_images, 100))

generated\_images = generator.predict(noise)

plt.figure(figsize=(10, 5))

for i in range(n\_images):

plt.subplot(2, 5, i + 1)

plt.imshow(generated\_images[i].reshape(28, 28), cmap='gray')

plt.axis('off')

plt.show()

# Generate and display images

generate\_images(generator, 10)