Sona College of Technology
Department of Information Technology

U19ADS605 DEEP LEARNING LABORATORY
Deep Learning Lab Manual

General Steps for all experiments:

1. Introduction to the experiment
2. Brief about the dataset
3. Explanation of the concept used in the experiment
4. Hands-on coding exercise with step by step instructions
5. Evaluation of the model
6. Conclusion and interpretation of the results

Materials required for the Lab:

1. Computer/laptop with a good configuration.
2. TensorFlow and PyTorch installed on the system.
3. Python libraries such as Numpy, Matplotlib, Pandas, etc.
4. Datasets required for each experiment.

Assessment:

1. Each experiment should be evaluated based on the accuracy of the model.
2. Students should be asked to interpret the results obtained after training the model.
3. Students should be evaluated based on their understanding of the concepts and their ability to implement the models.

1. Implement Simple Problem like Regression Model in TensorFlow.

Objective: To understand the basic architecture of TensorFlow and implement a regression model.

Steps:

1. Importing required libraries
2. Creating a simple dataset
3. Splitting the dataset into training and testing sets
4. Defining the model architecture
5. Compiling the model
6. Training the model
7. Evaluating the model
8. Making predictions using the trained model

Sample Program.

```
import tensorflow as tf
import numpy as np

# Define the input data
x = np.array([1, 2, 3, 4])
y = np.array([4, 7, 10, 13])

# Define the model architecture
model = tf.keras.Sequential([
  tf.keras.layers.Dense(units=1, input_shape=[1])
])

# Compile the model
model.compile(optimizer=tf.keras.optimizers.Adam(0.1),
loss='mean_squared_error')

# Train the model
model.fit(x, y, epochs=100)

# Make predictions using the trained model
predictions = model.predict([5, 6, 7])
print(predictions)
```

2. Implement a Perceptron in TensorFlow Environment.

Objective: To understand the basics of a perceptron and implement it using TensorFlow.

Steps:

1. Importing required libraries
2. Creating a simple dataset
3. Splitting the dataset into training and testing sets
4. Defining the perceptron architecture
5. Compiling the perceptron
6. Training the perceptron
7. Evaluating the perceptron
8. Making predictions using the trained perceptron

Sample Program.

```
import tensorflow as tf
import numpy as np

# Define the input data
x = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 0, 0, 1])

# Define the model architecture
model = tf.keras.Sequential([
  tf.keras.layers.Dense(units=1, input_shape=[2], activation='sigmoid')
])

# Compile the model
model.compile(optimizer=tf.keras.optimizers.Adam(0.1),
loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(x, y, epochs=100)

# Make predictions using the trained model
predictions = model.predict([[1, 1], [0, 0]])
print(predictions)
```

3. Implement a Feed-Forward Network in TensorFlow.

Objective: To understand the basics of feed-forward networks and implement it using TensorFlow.

Steps:

1. Importing required libraries
2. Creating a simple dataset
3. Splitting the dataset into training and testing sets
4. Defining the feed-forward network architecture
5. Compiling the network
6. Training the network
7. Evaluating the network
8. Making predictions using the trained network

Sample Program

```
import tensorflow as tf
import numpy as np

# Define the input data
x = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 1, 1, 0])

# Define the model architecture
model = tf.keras.Sequential([
  tf.keras.layers.Dense(units=4, input_shape=[2], activation='relu'),
  tf.keras.layers.Dense(units=1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer=tf.keras.optimizers.Adam(0.1),
loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(x, y, epochs=100)

# Make predictions using the trained model
predictions = model.predict([[1, 1], [0, 0]])
print(predictions)
```

4. Implement an Image Preprocessing using TensorFlow.

Objective: To understand how to preprocess images using TensorFlow.

Steps:

1. Importing required libraries
2. Loading the dataset of images
3. Applying preprocessing techniques such as normalization, resizing, and cropping
4. Displaying the preprocessed images

Sample Program

```
import tensorflow as tf
import matplotlib.pyplot as plt

# Load the image from file
image_path = 'image.jpg'
image = tf.io.read_file(image_path)
image = tf.image.decode_jpeg(image, channels=3)

# Visualize the original image
plt.imshow(image.numpy())
plt.title('Original Image')
plt.show()

# Resize the image
image = tf.image.resize(image, size=(256, 256))

# Visualize the resized image
plt.imshow(image.numpy())
plt.title('Resized Image')
plt.show()

# Convert the image to grayscale
grayscale_image = tf.image.rgb_to_grayscale(image)

# Visualize the grayscale image
plt.imshow(tf.squeeze(grayscale_image).numpy(), cmap='gray')
plt.title('Grayscale Image')
plt.show()

# Normalize the pixel values of the image
normalized_image = (tf.cast(grayscale_image, tf.float32) - 127.5) / 127.5

# Visualize the normalized image
plt.imshow(tf.squeeze(normalized_image).numpy(), cmap='gray')
plt.title('Normalized Image')
plt.show()
```

5. Implement an Image Classifier using CNN in TensorFlow.

Objective: To understand how to implement a convolutional neural network (CNN) for image classification using TensorFlow.

Steps:

1. Importing required libraries
2. Loading the dataset of images
3. Preprocessing the images
4. Defining the CNN architecture
5. Compiling the CNN
6. Training the CNN
7. Evaluating the CNN
8. Making predictions using the trained CNN

Sample Code:

```
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Load the dataset
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.mnist.load_data()

# Preprocess the data
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1)
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1)
X_train = X_train.astype('float32') / 255
X_test = X_test.astype('float32') / 255

# Define the model
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))

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

# Train the model
model.fit(X_train, y_train, epochs=5, validation_data=(X_test, y_test))

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print('Accuracy:', accuracy)
```

6. Implement a Transfer Learning Concept in Image Classification.

Objective: To understand how to use pre-trained models for transfer learning in image classification.

Steps:

1. Importing required libraries
2. Loading the pre-trained model
3. Freezing the pre-trained layers
4. Adding new layers for the new task
5. Compiling the model
6. Training the model
7. Evaluating the model
8. Making predictions using the trained model

Sample Code:

```
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense, Flatten, Dropout
from keras.applications import VGG16
# Load the pre-trained VGG16 model
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224,
3))

# Freeze the layers of the pre-trained model
for layer in base_model.layers:
    layer.trainable = False

# Add new layers to the model
model = Sequential()
model.add(base_model)
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

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

# Train the model
history = model.fit(train_generator, epochs=10,
validation_data=validation_generator)

# Evaluate the model
loss, accuracy = model.evaluate(test_generator)
print('Accuracy:', accuracy)
```

7. Implement Object Detection using PyTorch.

Objective: To understand how to implement object detection using PyTorch.

Steps:

1. Importing required libraries
2. Loading the dataset of images and labels
3. Defining the object detection model architecture
4. Compiling the model
5. Training the model
6. Evaluating the model
7. Making predictions using the trained model

Sample Program.

```
import torch
import torchvision
import cv2

# Load the pre-trained model
model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)

# Load the image
image = cv2.imread('image.jpg')

# Convert the image to a PyTorch tensor
image_tensor = torchvision.transforms.functional.to_tensor(image)

# Make a batch of the image tensor
image_batch = image_tensor.unsqueeze(0)

# Use the model to make predictions on the image batch
model.eval()
predictions = model(image_batch)

# Get the bounding boxes, labels, and scores for the predictions
boxes = predictions[0]['boxes'].detach().numpy()
labels = predictions[0]['labels'].detach().numpy()
scores = predictions[0]['scores'].detach().numpy()

# Draw the bounding boxes on the image
for box, label, score in zip(boxes, labels, scores):
    if score > 0.5:
        x1, y1, x2, y2 = box
        cv2.rectangle(image, (x1, y1), (x2, y2), (0, 255, 0), 2)
        cv2.putText(image, str(label), (x1, y1-10), cv2.FONT_HERSHEY_SIMPLEX,
0.9, (0, 255, 0), 2)

# Save the annotated image
cv2.imwrite('annotated_image.jpg', image)
```

8. Implement Recurrent Neural Network in PyTorch.

Objective: To understand how to implement recurrent neural networks (RNN) in PyTorch.

Steps:

1. Importing required libraries
2. Creating a simple dataset
3. Defining the RNN architecture
4. Compiling the model
5. Training the model
6. Evaluating the model
7. Making predictions using the trained model

Sample Program.

```
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Embedding, SimpleRNN, Dense

# Load the dataset
imdb = tf.keras.datasets.imdb
(train_data, train_labels), (test_data, test_labels) =
imdb.load_data(num_words=10000)

# Preprocess the data
train_data = tf.keras.preprocessing.sequence.pad_sequences(train_data,
maxlen=500)
test_data = tf.keras.preprocessing.sequence.pad_sequences(test_data, maxlen=500)

# Define the model
model = Sequential()
model.add(Embedding(10000, 32))
model.add(SimpleRNN(32))
model.add(Dense(1, activation='sigmoid'))

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

# Train the model
model.fit(train_data, train_labels, epochs=10, batch_size=128)

# Evaluate the model
loss, accuracy = model.evaluate(test_data, test_labels)
print('Accuracy:', accuracy)
```

9. Implement a SimpleLSTM using PyTorch.

Objective: To understand how to implement Long Short-Term Memory (LSTM) networks in PyTorch.

Steps:

1. Importing required libraries
2. Creating a simple dataset
3. Defining the LSTM architecture
4. Compiling the model
5. Training the model
6. Evaluating the model
7. Making predictions using the trained model

Sample program

```
import tensorflow as tf
from keras.models import Sequential
from keras.layers import LSTM, Dense

# Load the dataset
imdb = tf.keras.datasets.imdb
(train_data, train_labels), (test_data, test_labels) =
imdb.load_data(num_words=10000)

# Preprocess the data
train_data = tf.keras.preprocessing.sequence.pad_sequences(train_data,
maxlen=500)
test_data = tf.keras.preprocessing.sequence.pad_sequences(test_data, maxlen=500)

# Define the model
model = Sequential()
model.add(tf.keras.layers.Embedding(10000, 32))
model.add(LSTM(32))
model.add(Dense(1, activation='sigmoid'))

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

# Train the model
model.fit(train_data, train_labels, epochs=10, batch_size=128)

# Evaluate the model
loss, accuracy = model.evaluate(test_data, test_labels)
```

10. Implement an Autoencoder in PyTorch.

Objective: To understand how to implement an autoencoder in PyTorch.

Steps:

1. Importing required libraries
2. Loading the dataset
3. Defining the autoencoder architecture
4. Compiling the model
5. Training the model
6. Evaluating the model
7. Making predictions using the trained model

Sample Program:

```
import tensorflow as tf
from keras.models import Model
from keras.layers import Input, Dense

# Load the dataset
(X_train, _), (X_test, _) = tf.keras.datasets.mnist.load_data()

# Preprocess the data
X_train = X_train.astype('float32') / 255.
X_test = X_test.astype('float32') / 255.

# Define the model
input_img = Input(shape=(784,))
encoded = Dense(128, activation='relu')(input_img)
encoded = Dense(64, activation='relu')(encoded)
encoded = Dense(32, activation='relu')(encoded)
decoded = Dense(64, activation='relu')(encoded)
decoded = Dense(128, activation='relu')(decoded)
decoded = Dense(784, activation='sigmoid')(decoded)
autoencoder = Model(input_img, decoded)

# Compile the model
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

# Train the model
autoencoder.fit(X_train, X_train, epochs=50, batch_size=256, shuffle=True,
validation_data=(X_test, X_test))

# Evaluate the model
decoded_imgs = autoencoder.predict(X_test)
```