

Deep learning lab – IPCC

Pg. 1 Write a program to demonstrate the working of a deep neural network for classification task.

Soln:

Multi-Layer Perceptron (MLP) for classification task on MNIST Dataset

A)

```
!pip install tensorflow
```

```
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Conv2D, MaxPool2D

# loading the dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# let's print the shape of the dataset
print("X_train shape", X_train.shape)
print("y_train shape", y_train.shape)
print("X_test shape", X_test.shape)
print("y_test shape", y_test.shape)

X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

X_train /= 255
X_test /= 255
```

Output: Downloading data from
<https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz>

11490434/11490434 ————— **0s** 0us/step

```
X_train shape (60000, 28, 28)
y_train shape (60000,)
X_test shape (10000, 28, 28)
y_test shape (10000,)
```

```
y_train
```

Output: array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)

```
!pip install np_utils

#from keras.utils import np_utils
#from keras.utils import np_utils as np
from tensorflow.keras.utils import to_categorical
import tensorflow as tf
n_classes = 10
print("Shape before one-hot encoding: ", y_train.shape)
y_train=tf.keras.utils.to_categorical(y_train,n_classes)
y_test =tf.keras.utils.to_categorical(y_test, n_classes)
#Y_train = np.to_categorical(y_train, n_classes)
#Y_test = np.to_categorical(y_test, n_classes)
print("Shape after one-hot encoding: ", y_train.shape)
```

Output:

```
Collecting np_utils
  Downloading np_utils-0.6.0.tar.gz (61 kB)


---


— 62.0/62.0 kB 2.6 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy>=1.0 in
/usr/local/lib/python3.11/dist-packages (from np_utils) (2.0.2)
Building wheels for collected packages: np_utils
  Building wheel for np_utils (setup.py) ... done
  Created wheel for np_utils: filename=np_utils-0.6.0-py3-none-any.whl
size=56437
sha256=01170fe26c7f69fd27b5abd026a1e1e867699f666d34865b69dc07e7fa6e1fdf
  Stored in directory:
/root/.cache/pip/wheels/19/0d/33/eea4dcda5799bcbb51733c0744970d10edb4b9
add4f41beb43
Successfully built np_utils
Installing collected packages: np_utils
Successfully installed np_utils-0.6.0
Shape before one-hot encoding: (60000,)
Shape after one-hot encoding: (60000, 10)
```

y_train

Output:

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 1., 0.]])
```

```
y_train.shape
```

Output:

```
(60000, 10)
```

```
X_train.shape
```

Output:

```
(60000, 784)
```

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

# Define the model:
model = Sequential([
    Dense(100, input_dim=784, activation='relu'),
    Dense(10, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam',
              loss=tf.keras.losses.CategoricalCrossentropy(), # Use CategoricalCrossentropy object
              metrics=['accuracy'])

print(model.summary())
```

Output:

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 100)	78,500
dense_1 (Dense)	(None, 10)	1,010

Total params: 79,510 (310.59 KB)

Trainable params: 79,510 (310.59 KB)

Non-trainable params: 0 (0.00 B)

None

```
# Train the model
import tensorflow as tf

model.fit(X_train, y_train, epochs=50, batch_size=32)
```

```
# Evaluate the model
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f'Test accuracy: {test_acc}')

# Make predictions
# predictions = model.predict(X_test)
```

Output:

```
313/313 ————— 1s 3ms/step - accuracy:
0.9731 - loss: 0.2194
Test accuracy: 0.9779000282287598
```

B)

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Define the architecture of the MLP
model = keras.Sequential([
    layers.Input(shape=(784,)), # Input layer with 784 units (for
MNIST data)
    layers.Dense(128, activation='relu'), # First hidden layer with
128 units and ReLU activation
    layers.Dense(64, activation='relu'), # Second hidden layer with
64 units and ReLU activation
    layers.Dense(10, activation='softmax') # Output layer with 10
units (for classification) and softmax activation
])

# Compile the model
model.compile(optimizer='adam',
              loss=tf.keras.losses.CategoricalCrossentropy(), # Use
CategoricalCrossentropy object
              metrics=['accuracy'])

# Load your dataset (e.g., MNIST)
# X_train, y_train, X_test, y_test = load_dataset()

print(X_train.shape)
print(y_train.shape)
# Train the model
model.fit(X_train, y_train, epochs=50, batch_size=32)

# Evaluate the model
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f'Test accuracy: {test_acc}')

# Make predictions
# predictions = model.predict(X_test)
```

Pg.2 Design and implement a Convolutional Neural Network (CNN) for classification of image dataset.

A)

```
'''The program make use of CIFAR-10 dataset which is a collection of
60,000 32x32 color images in 10 different classes, with 6,000 images
per class.'''
#import necessary libraries
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical

(x_train, y_train), (x_test, y_test) = cifar10.load_data()
print("Total number of samples=",len(x_train))
print("Total number of class labels=",len(y_train))
print("Total number of samples=",len(x_test))
print("Total number of class labels=",len(y_test))
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
```

Output:

Downloading data from <https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz>

170498071/170498071 ————— **4s 0us/step**

Total number of samples= 50000

Total number of class labels= 50000

Total number of samples= 10000

Total number of class labels= 10000

```
y_train, y_test = to_categorical(y_train, 10), to_categorical(y_test,
10)

model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32,
3)),
```

```

layers.MaxPooling2D((2, 2)),
layers.Conv2D(64, (3, 3), activation='relu'),
layers.MaxPooling2D((2, 2)),
layers.Flatten(),
layers.Dense(64, activation='relu'),
layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
history1 = model.fit(x_train, y_train, epochs=5, batch_size=32,
validation_data=(x_test, y_test))

test_loss, test_acc = model.evaluate(x_test, y_test, batch_size=32)
print(f"Test Accuracy: {test_acc:.2%}")
print("-----END-----")

```

Output:

```

313/313 ----- 4s 11ms/step - accuracy:
0.6808 - loss: 0.9256
Test Accuracy: 67.55%

```

B)

```

'''The program make use of CIFAR-10 dataset which is a collection of
60,000 32x32 color images in 10 different classes, with 6,000 images
per class.'''

```

```

#import necessary libraries
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt

```

```

print("-----")
print("-----")
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
print("Training data shape=", x_train.shape, ",", "Number of training
samples=", len(x_train))
print("Training labels shape=", y_train.shape, ",", "Number of training
labels=", len(y_train))
print("Testing data shape=", x_test.shape, ",", "Number of testing
samples=", len(x_test))

```

```

print("Testing labels shape=", y_test.shape, ",", "Number of testing
labels=", len(y_test))
print("-----")

print("-----")
print("x_train values before normalization\n", x_train)
print("-----")
print("-----\n\n")
print("x_test values before normalization\n", x_test)
print("-----")
print("-----\n\n")

```

Output :

Downloading data from <https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz>

170498071/170498071  **2s 0us/step**

Training data shape= (50000, 32, 32, 3) , Number of training samples= 50000

Training labels shape= (50000, 1) , Number of training labels= 50000

Testing data shape= (10000, 32, 32, 3) , Number of testing samples= 10000

Testing labels shape= (10000, 1) , Number of testing labels= 10000

```

x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

print("-----")
print("x_train values after normalization\n", x_train)
print("-----")
print("-----\n\n")
print("x_test values after normalization\n", x_test)
print("-----")
print("-----\n\n")

class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog',
'frog', 'horse', 'ship', 'truck']
for i in range(9):
    plt.subplot(3, 3, i + 1)
    plt.imshow(x_train[i])
    plt.title(class_names[y_train[i][0]])
    plt.axis('off')

```

```
plt.show()
```

Output:



```
y_train, y_test = to_categorical(y_train, 10), to_categorical(y_test, 10)
```

```
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

```
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
history1 = model.fit(x_train, y_train, epochs=5, batch_size=256,
validation_data=(x_test, y_test))
```

```
#model.summary is to print the parameters of the model layer by layer
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	896

max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
flatten (Flatten)	(None, 2304)	0
dense (Dense)	(None, 64)	147,520
dense_1 (Dense)	(None, 10)	650

Total params: 502,688 (1.92 MB)
Trainable params: 167,562 (654.54 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 335,126 (1.28 MB)

```

# Plot training Accuracy versus Validation Accuracy for each epoch
plt.plot(history1.history['accuracy'], label='Training-Accuracy',
color='red')
plt.plot(history1.history['val_accuracy'], label='Validation_Accuracy',
color='green')
plt.legend()
plt.show()

```

```

# Plot loss versus Validation Loss for each epoch
plt.plot(history1.history['val_loss'], label='Validation_Loss',
color='yellow')
plt.plot(history1.history['loss'], label='loss', color='blue')
plt.legend()
plt.show()

```

```

#print Testing Accuracy
print("-----")
print("-----")
print("x_test=", len(x_test), "y_test=", len(y_test))
test_loss, test_acc = model.evaluate(x_test, y_test, batch_size=4)
print(f"Test Accuracy: {test_acc:.2%}")
print("-----END-----")
print("-----")

```

```

Output: -----
-----

x_test= 10000 y_test= 10000
2500/2500 ----- 10s 4ms/step - accuracy:
0.6309 - loss: 1.0785
Test Accuracy: 62.59%
-----END-----
-----

```

C)

```
'''The program make use of CIFAR-10 dataset which is a collection of  
60,000 32x32 color images in 10 different classes, with 6,000 images  
per class.'''
```

```
#import necessary libraries  
import tensorflow as tf  
from tensorflow.keras import layers, models  
from tensorflow.keras.datasets import cifar10  
from tensorflow.keras.utils import to_categorical  
import matplotlib.pyplot as plt
```

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

```
from google.colab import drive  
drive.mount('/content/drive')
```

```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog',  
'frog', 'horse', 'ship', 'truck']  
# Display images with class names  
plt.figure(figsize=(10, 10))  
for i in range(25):  
    plt.subplot(5, 5, i + 1)  
    plt.imshow(x_train[i])  
    plt.title(class_names[y_train[i][0]])  
    plt.axis('off')  
plt.show()
```

Output:



```
from tensorflow.keras.callbacks import EarlyStopping

y_train, y_test = to_categorical(y_train, 10), to_categorical(y_test,
10)

model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32,
3)),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])

early_stopping = EarlyStopping(monitor='val_loss', patience=3)

model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

history1 = model.fit(x_train, y_train, epochs=10, batch_size=256,
validation_data=(x_test, y_test), callbacks=[early_stopping])

#model.summary is to print the parameters of the model layer by layer
model.summary()
```

Output:

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	896
batch_normalization (BatchNormalization)	(None, 30, 30, 32)	128
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 13, 13, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
flatten (Flatten)	(None, 2304)	0
dense (Dense)	(None, 64)	147,520
dense_1 (Dense)	(None, 10)	650

Total params: 503,456 (1.92 MB)

Trainable params: 167,754 (655.29 KB)

Non-trainable params: 192 (768.00 B)

Optimizer params: 335,510 (1.28 MB)

```
# Plot training Accuracy versus Validation Accuracy for each epoch
plt.plot(history1.history['accuracy'], label='Training-Accuracy',
color='red')
plt.plot(history1.history['val_accuracy'], label='Validation_Accuracy',
color='green')
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.show()

# Plot loss versus Validation Loss for each epoch
plt.plot(history1.history['val_loss'], label='Validation_Loss',
color='yellow')
plt.plot(history1.history['loss'], label='loss', color='blue')
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

```
#print Testing Accuracy
print("-----")
print("-----")
print("x_test=", len(x_test), "y_test=", len(y_test))
```

```
test_loss, test_acc = model.evaluate(x_test, y_test, batch_size=4)
print(f"Test Accuracy: {test_acc:.2%}")
print("-----END-----")
print("-----")
```

```
import numpy as np

sample_index = 3
sample_image = np.expand_dims(x_test[sample_index], axis=0)

prediction = model.predict(sample_image)
predicted_class = np.argmax(prediction)

# Class names for CIFAR-10
cifar10_labels = ['airplane', 'automobile', 'bird', 'cat', 'deer',
                  'dog', 'frog', 'horse', 'ship', 'truck']

print(f"Predicted class: {cifar10_labels[predicted_class]}")
```

```
from tensorflow.keras.preprocessing import image
import numpy as np
from PIL import Image

img_path = '/content/drive/MyDrive/Deep learning lab /CNN network.png'
img = Image.open(img_path).resize((32, 32)).convert('RGB')
img_array = np.array(img).astype('float32') / 255.0
img_array = np.expand_dims(img_array, axis=0)

# Predict the class
prediction = model.predict(img_array)
predicted_class = np.argmax(prediction)

print(f"Predicted class: {cifar10_labels[predicted_class]}")

import matplotlib.pyplot as plt
plt.imshow(img)
plt.title(f"Predicted: {cifar10_labels[predicted_class]}")
plt.show()
```

```
import pandas as pd
y_pred_probs = model.predict(x_test)

y_pred_classes = np.argmax(y_pred_probs, axis=1)
y_true = np.argmax(y_test, axis=1)

df = pd.DataFrame({
```

```

    "Actual Class": [class_names[i] for i in y_true],
    "Predicted Class": [class_names[i] for i in y_pred_classes]
})

misclassified = df[df["Actual Class"] != df["Predicted Class"]]
correctly_classified = df[df["Actual Class"] == df["Predicted Class"]]

# Print misclassified results
print("Number of Misclassified Samples:", len(misclassified))
print(misclassified)

print("Number of Correctly Classified Samples:", len(misclassified))
print(correctly_classified)

```

```

print("Total samples:", len(y_test))
print("Misclassified samples:", len(misclassified))

```

```

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Compute the confusion matrix
cm = confusion_matrix(y_true, y_pred_classes)

# Display the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=class_names)
disp.plot(cmap=plt.cm.Blues, xticks_rotation='vertical')
plt.title("Confusion Matrix")
plt.tight_layout()
plt.show()

```

```

from sklearn.metrics import classification_report

report=classification_report(y_true,y_pred_classes,target_names=class_n
ames)
print(report)

```

Pg3. Write a program to enable pre-train models to classify a given image dataset.

Soln:

A) Pre-trained model – VGG16

```

# This Python 3 environment comes with many helpful analytics
libraries installed

```

```
# It is defined by the kaggle/python Docker image:
https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter)
will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
# You can write up to 20GB to the current directory (/kaggle/working/)
that gets preserved as output when you create a version using "Save &
Run All"
# You can also write temporary files to /kaggle/temp/, but they won't
be saved outside of the current session
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
import os
import json
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l2
from sklearn.model_selection import train_test_split
from tqdm import tqdm
import cv2
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

train_dataset_path = "/content/drive/MyDrive/Deep learning lab /cats-
dogs-dataset/Training"
test_dataset_path = "/content/drive/MyDrive/Deep learning lab /cats-
dogs-dataset/Testing"
IMG_SIZE = 224
Model_save_path = "vgg16.h5"
```

```

def get_label_from_filename(filename):
    return filename.lower().split("_")[0]

def load_data(dataset_path):
    images, labels = [], []
    class_names = set()

    for file in tqdm(os.listdir(dataset_path)):
        if file.endswith((".jpg", ".png", ".jpeg")):
            path = os.path.join(dataset_path, file)
            img = cv2.imread(path)

            if img is None:
                print(f"Warning: Unable to read image: {file}")
                continue # Skip invalid image

            img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
            img = preprocess_input(img)

            label = get_label_from_filename(file)
            class_names.add(label)

            images.append(img)
            labels.append(label)

    class_names = sorted(list(class_names))
    label_to_index = {name: idx for idx, name in
enumerate(class_names)}

    labels = np.array([label_to_index[label] for label in labels])
    images = np.array(images)

    return images, labels, class_names, label_to_index

X_train_full, y_train_full, class_names, label_to_index =
load_data(train_dataset_path)
X_train, X_val, y_train, y_val = train_test_split(X_train_full,
y_train_full, test_size=0.15, random_state=42)

y_train = tf.keras.utils.to_categorical(y_train,
num_classes=len(class_names))
y_val = tf.keras.utils.to_categorical(y_val,
num_classes=len(class_names))

X_test, y_test, _, _ = load_data(test_dataset_path)
y_test = tf.keras.utils.to_categorical(y_test,
num_classes=len(class_names))

```

```

# === VGG16 Model ===
base_model = VGG16(input_shape=(IMG_SIZE, IMG_SIZE, 3),
                    include_top=False,
                    weights='imagenet')
base_model.trainable = True

inputs = keras.Input(shape=(IMG_SIZE, IMG_SIZE, 3))
x = base_model(inputs)
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dense(256, activation='relu',
kernel_regularizer=l2(0.001))(x)
x = layers.Dropout(0.3)(x)
outputs = layers.Dense(len(class_names), activation='softmax')(x)
model = keras.Model(inputs, outputs)

# === Compile and Train ===
model.compile(optimizer=Adam(learning_rate=0.0007),
loss='categorical_crossentropy', metrics=['accuracy'])

history = model.fit(X_train, y_train, validation_data=(X_val, y_val),
epochs=5, batch_size=32)

# === Evaluate ===
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"\nTest Accuracy: {test_acc:.4f}")

# === Save Model ===
model.save(Model_save_path)
print(f"Model saved to {Model_save_path}")

# === Classification Report and Confusion Matrix ===
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

print(classification_report(y_true, y_pred_classes,
target_names=class_names))

cm = confusion_matrix(y_true, y_pred_classes)
sns.heatmap(cm, annot=True, fmt='d', xticklabels=class_names,
yticklabels=class_names)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()

```

B) Pre-trained model – ResNet50

I)

```
# This Python 3 environment comes with many helpful analytics libraries
installed
# It is defined by the kaggle/python Docker image:
https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load
```

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

```
# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter)
will list all files under the input directory
```

```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
# You can write up to 20GB to the current directory (/kaggle/working/)
that gets preserved as output when you create a version using "Save &
Run All"
# You can also write temporary files to /kaggle/temp/, but they won't
be saved outside of the current session
```

```
from google.colab import drive
drive.mount ('/content/drive')
```

```
import os
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l2
from sklearn.model_selection import train_test_split
from tqdm import tqdm
import cv2
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

```

train_dataset_path = "/content/drive/MyDrive/Deep learning lab /cats-
dogs-dataset/Training"
test_dataset_path = "/content/drive/MyDrive/Deep learning lab /cats-
dogs-dataset/Testing"
IMG_SIZE = 224
Model_save_path = "resnet50_all_layers.h5"

def get_label_from_filename(filename):
    return filename.lower().split("_")[0]

def load_data(dataset_path):
    images, labels = [], []
    class_names = set()
    for file in tqdm(os.listdir(dataset_path)):
        if file.endswith((".jpg", ".png", ".jpeg")):
            path = os.path.join(dataset_path, file)
            img = cv2.imread(path)
            if img is None:
                print(f"Warning: Unable to read image: {file}")
                continue
            img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
            img = preprocess_input(img)
            label = get_label_from_filename(file)
            class_names.add(label)
            images.append(img)
            labels.append(label)
    class_names = sorted(list(class_names))
    label_to_index = {name: idx for idx, name in
enumerate(class_names)}
    labels = np.array([label_to_index[label] for label in labels])
    images = np.array(images)
    return images, labels, class_names, label_to_index

X_train_full, y_train_full, class_names, label_to_index =
load_data(train_dataset_path)
X_train, X_val, y_train, y_val = train_test_split(X_train_full,
y_train_full, test_size=0.15, random_state=42)
y_train = tf.keras.utils.to_categorical(y_train,
num_classes=len(class_names))
y_val = tf.keras.utils.to_categorical(y_val,
num_classes=len(class_names))
X_test, y_test, _, _ = load_data(test_dataset_path)
y_test = tf.keras.utils.to_categorical(y_test,
num_classes=len(class_names))

# === ResNet50 All Layers Trainable ===
base_model = ResNet50(input_shape=(IMG_SIZE, IMG_SIZE, 3),

```

```

        include_top=False,
        weights='imagenet')
base_model.trainable = True

inputs = keras.Input(shape=(IMG_SIZE, IMG_SIZE, 3))
x = base_model(inputs, training=True)
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dense(256, activation='relu',
kernel_regularizer=l2(0.001))(x)
x = layers.Dropout(0.3)(x)
outputs = layers.Dense(len(class_names), activation='softmax')(x)
model = keras.Model(inputs, outputs)

model.compile(optimizer=Adam(learning_rate=0.0001),
loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(X_train, y_train, validation_data=(X_val, y_val),
epochs=10, batch_size=32)

# === Evaluate ===
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"\nTest Accuracy: {test_acc:.4f}")

# === Save Model ===
model.save(Model_save_path)
print(f"Model saved to {Model_save_path}")

# === Classification Report & Confusion Matrix ===
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

print(classification_report(y_true, y_pred_classes,
target_names=class_names))

cm = confusion_matrix(y_true, y_pred_classes)
sns.heatmap(cm, annot=True, fmt='d', xticklabels=class_names,
yticklabels=class_names)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()

# === Accuracy Plot ===
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title("Model Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")

```

```

plt.legend()
plt.grid(True)
plt.show()

# === Loss Plot ===
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title("Model Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)
plt.show()

```

II)

```

import os
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l2
from sklearn.model_selection import train_test_split
from tqdm import tqdm
import cv2
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

train_dataset_path = "/content/drive/MyDrive/Deep learning lab /cats-
dogs-dataset/Training"
test_dataset_path = "/content/drive/MyDrive/Deep learning lab /cats-
dogs-dataset/Testing"
IMG_SIZE = 224
Model_save_path = "resnet50_all_layers.h5"

def get_label_from_filename(filename):
    return filename.lower().split("_")[0]

def load_data(dataset_path):
    images, labels = [], []
    class_names = set()
    for file in tqdm(os.listdir(dataset_path)):
        if file.endswith((".jpg", ".png", ".jpeg")):

```

```

        path = os.path.join(dataset_path, file)
        img = cv2.imread(path)
        if img is None:
            print(f"Warning: Unable to read image: {file}")
            continue
        img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
        img = preprocess_input(img)
        label = get_label_from_filename(file)
        class_names.add(label)
        images.append(img)
        labels.append(label)
    class_names = sorted(list(class_names))
    label_to_index = {name: idx for idx, name in
enumerate(class_names)}
    labels = np.array([label_to_index[label] for label in labels])
    images = np.array(images)
    return images, labels, class_names, label_to_index

X_train_full, y_train_full, class_names, label_to_index =
load_data(train_dataset_path)
X_train, X_val, y_train, y_val = train_test_split(X_train_full,
y_train_full, test_size=0.15, random_state=42)
y_train = tf.keras.utils.to_categorical(y_train,
num_classes=len(class_names))
y_val = tf.keras.utils.to_categorical(y_val,
num_classes=len(class_names))
X_test, y_test, _, _ = load_data(test_dataset_path)
y_test = tf.keras.utils.to_categorical(y_test,
num_classes=len(class_names))

# === ResNet50 Top 60 Layers Trainable ===
base_model = ResNet50(input_shape=(IMG_SIZE, IMG_SIZE, 3),
                        include_top=False,
                        weights='imagenet')

base_model.trainable = False # Freeze all layers
for layer in base_model.layers[-60:]:
    layer.trainable = True # Unfreeze top 60 layers

inputs = keras.Input(shape=(IMG_SIZE, IMG_SIZE, 3))
x = base_model(inputs, training=True)
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dense(256, activation='relu',
kernel_regularizer=l2(0.001))(x)
x = layers.Dropout(0.3)(x)
outputs = layers.Dense(len(class_names), activation='softmax')(x)
model = keras.Model(inputs, outputs)

```

```

model.compile(optimizer=Adam(learning_rate=0.0001),
loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(X_train, y_train, validation_data=(X_val, y_val),
epochs=10, batch_size=32)

# === Evaluate ===
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"\nTest Accuracy: {test_acc:.4f}")

# === Save Model ===
model.save(Model_save_path)
print(f"Model saved to {Model_save_path}")

# === Classification Report & Confusion Matrix ===
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

print(classification_report(y_true, y_pred_classes,
target_names=class_names))

cm = confusion_matrix(y_true, y_pred_classes)
sns.heatmap(cm, annot=True, fmt='d', xticklabels=class_names,
yticklabels=class_names)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()

# === Accuracy Plot ===
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title("Model Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)
plt.show()

# === Loss Plot ===
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title("Model Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)
plt.show()

```

Pg.4 Design and implement a neural based network for generating word embedding for words in a document corpus.

I)

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import string
import os

# Download necessary NLTK data
try:
    nltk.data.find('tokenizers/punkt')
except nltk.downloader.DownloadError:
    nltk.download('punkt')
try:
    nltk.data.find('corpora/stopwords')
except nltk.downloader.DownloadError:
    nltk.download('stopwords')

# Assume the corpus is in a file named 'corpus.txt'
corpus_file = 'corpus.txt'

# Create a dummy corpus file if it doesn't exist
if not os.path.exists(corpus_file):
    with open(corpus_file, 'w') as f:
        f.write("This is the first document.\n")
        f.write("This document is the second document.\n")
        f.write("And this is the third one.\n")
        f.write("Is this the first document?\n")

# Load the document corpus
with open(corpus_file, 'r') as f:
    corpus = f.readlines()

# Preprocess the corpus
stop_words = set(stopwords.words('english'))
processed_corpus = []

for document in corpus:
    # Tokenize the document
    tokens = word_tokenize(document)

    # Convert to lowercase and remove punctuation and non-alphabetic
    tokens
```

```

tokens = [word.lower() for word in tokens if word.isalpha()]

# Remove stop words
tokens = [word for word in tokens if word not in stop_words]

processed_corpus.append(tokens)

print("Original Corpus:")
for doc in corpus:
    print(doc.strip())

print("\nProcessed Corpus (Tokens):")
for doc_tokens in processed_corpus:
    print(doc_tokens)

```

Output:

Original Corpus:
This is the first document.
This document is the second document.
And this is the third one.
Is this the first document?

Processed Corpus (Tokens):
['first', 'document']
['document', 'second', 'document']
['third', 'one']
['first', 'document']

```

vocabulary = set()
for tokens in processed_corpus:
    for token in tokens:
        vocabulary.add(token)

sorted_vocabulary = sorted(list(vocabulary))

word_to_index = {word: i for i, word in enumerate(sorted_vocabulary)}
index_to_word = {i: word for i, word in enumerate(sorted_vocabulary)}

print(f"Vocabulary size: {len(vocabulary)}")
print("\nFirst few entries of word_to_index:")
for i, (word, index) in enumerate(word_to_index.items()):
    if i >= 5:
        break
    print(f"{word}: {index}")

print("\nFirst few entries of index_to_word:")
for i, (index, word) in enumerate(index_to_word.items()):
    if i >= 5:
        break

```

```
print(f"{index}: {word}")
```

Output:

Vocabulary size: 5

First few entries of word_to_index:

document: 0

first: 1

one: 2

second: 3

third: 4

First few entries of index_to_word:

0: document

1: first

2: one

3: second

4: third

```
training_data = []
```

```
window_size = 2
```

```
for document in processed_corpus:
    for i, target_word in enumerate(document):
        target_index = word_to_index[target_word]
        context_words = []
        for j in range(max(0, i - window_size), min(len(document), i +
window_size + 1)):
            if i != j:
                context_words.append(document[j])
```

```
        for context_word in context_words:
            context_index = word_to_index[context_word]
            training_data.append([context_index, target_index]) #
Using Skip-gram model format (context, target)
```

```
print(f"Number of training pairs: {len(training_data)}")
```

```
print("\nFirst few training pairs (context index, target index):")
```

```
for i, pair in enumerate(training_data):
```

```
    if i >= 10:
```

```
        break
```

```
    print(pair)
```

Output:

Number of training pairs: 12

First few training pairs (context index, target index):

```

([0], 1)
([1], 0)
([3], 0)
([0], 0)
([0], 3)
([0], 3)
([0], 0)
([3], 0)
([2], 4)
([4], 2)

```

```

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

# Define vocabulary size and embedding dimension
vocab_size = len(vocabulary)
embedding_dim = 50 # Choose a suitable embedding dimension

# Create the Sequential model
model = Sequential()

# Add the Embedding layer
model.add(Embedding(input_dim=vocab_size, output_dim=embedding_dim,
input_length=1))

# Add a Dense layer for the output (Skip-gram)
model.add(Dense(units=vocab_size, activation='softmax'))

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

# Print the model summary
model.summary()

```

```

/usr/local/lib/python3.11/dist-
packages/keras/src/layers/core/embedding.py:97: UserWarning: Argument
`input_length` is deprecated. Just remove it.
  warnings.warn(

```

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	?	0 (unbuilt)
dense (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)
 Trainable params: 0 (0.00 B)
 Non-trainable params: 0 (0.00 B)

```

import numpy as np
import tensorflow as tf

# Flatten the context indices before converting to NumPy array
flattened_training_data = [(pair[0][0], pair[1]) for pair in
training_data]

# Convert flattened training data to NumPy array
training_data_np = np.array(flattened_training_data)

# Separate input (context indices) and output (target indices)
context_indices = training_data_np[:, 0]
target_indices = training_data_np[:, 1]

# Convert target indices to one-hot encoded vectors
target_one_hot = tf.keras.utils.to_categorical(target_indices,
num_classes=vocab_size)

# Train the model
history = model.fit(context_indices, target_one_hot, epochs=200,
verbose=0)

print("Training finished.")

```

```

word_embeddings = model.layers[0].get_weights()[0]
print(f"Shape of word embeddings: {word_embeddings.shape}")

```

Shape of word embeddings: (5, 50)

II)

```
!pip install gensim
```

```

import gensim
from gensim.models import Word2Vec
from gensim.utils import simple_preprocess

```

```

# Define the corpus
corpus=[
    "Deep learning is a core subject of artificial intelligence",
    "Machine learning is a subbranch of deep learning",
    "Convolutional Neural Network (CNN) is a basic deep neural network
in deep learning",
    "Alex and Visual Geometry Group (VGG) neural networks are pre-
trained deep neural networks",
    "Deep residual network is used in image recognition"

```

```
]
```

```
# Preprocess the corpus
tokenized_corpus=[simple_preprocess(line) for line in corpus]
```

```
print(tokenized_corpus)
```

```
model = Word2Vec(
    sentences=tokenized_corpus,
    vector_size=300,
    window=3,
    min_count=1,
    sg=1,
    epochs=100
)
```

```
# Build vocabulary
words = [word for sentence in tokenized_corpus for word in sentence]
vocab = set(words)
word2idx = {word: idx for idx, word in enumerate(vocab)}
idx2word = {idx: word for word, idx in word2idx.items()}
vocab_size = len(vocab)
```

```
print(words)
```

```
print(vocab)
```

```
print(vocab_size)
```

```
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np

class WordEmbeddingModel(nn.Module):
    def __init__(self, vocab_size, embedding_dim):
        super(WordEmbeddingModel, self).__init__()
        self.embeddings = nn.Embedding(vocab_size, embedding_dim)
        self.linear = nn.Linear(embedding_dim, vocab_size)

    def forward(self, inputs):
```

```

        embeds = self.embeddings(inputs)
        output = self.linear(embeds)
        return output

# Define the embedding dimension
vector_size = 300

# Instantiate the model
model = WordEmbeddingModel(vocab_size, vector_size)

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Prepare training data (simple example using skip-gram pairs)
# This is a simplified approach for demonstration.
# In a real scenario, you would generate context-target pairs from the
# corpus.
training_data = []
for sentence in tokenized_corpus:
    for i, target_word in enumerate(sentence):
        target_idx = word2idx[target_word]
        # Create context words (simplified: just the word itself as
        # context)
        context_idx = word2idx[target_word]
        training_data.append((context_idx, target_idx))

# Train the model
epochs = 100
for epoch in range(epochs):
    total_loss = 0
    for context_idx, target_idx in training_data:
        # Convert to tensors
        context_tensor = torch.LongTensor([context_idx])
        target_tensor = torch.LongTensor([target_idx])

        # Forward pass
        outputs = model(context_tensor)
        loss = criterion(outputs, target_tensor)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    total_loss += loss.item()

```

```

        if (epoch + 1) % 10 == 0:
            print(f'Epoch [{epoch+1}/{epochs}], Loss:
{total_loss/len(training_data):.4f}')

# Get the word embeddings
word_embeddings = model.embeddings.weight.data

# Print embeddings for a few words
print("\nWord embeddings:")
for word in ["deep", "learning", "intelligence", "network"]:
    if word in word2idx:
        idx = word2idx[word]
        print(f"{word}: {word_embeddings[idx].numpy()}")

```

III)

```
!pip install gensim
```

```

import gensim
from gensim.models import Word2Vec
from gensim.utils import simple_preprocess

```

```

# Define the corpus
corpus=[
    "Deep learning is a core subject of artificial intelligence",
    "Machine learning is a subbranch of deep learning",
    "Convolutional Neural Network (CNN) is a basic deep neural network
in deep learning",
    "Alex and Visual Geometry Group (VGG) neural networks are pre-
trained deep neural networks",
    "Deep residual network is used in image recognition"
]

```

```

# Preprocess the corpus
tokenized_corpus=[simple_preprocess(sentence) for sentence in corpus]

```

```
print(tokenized_corpus)
```

```

model = Word2Vec(
    sentences=tokenized_corpus,
    vector_size=300,
    window=3,
    min_count=1,
    sg=1,

```

```

        epochs=100
    )

# Build vocabulary
words = [word for sentence in tokenized_corpus for word in sentence]
vocab = set(words)
word2idx = {word: idx for idx, word in enumerate(vocab)}
idx2word = {idx: word for word, idx in word2idx.items()}
vocab_size = len(vocab)

print(words)

print(vocab)

print(vocab_size)

# Define the embedding dimension
vector_size = 300

# Prepare training data (simple example using skip-gram pairs)
# This is a simplified approach for demonstration.
# In a real scenario, you would generate context-target pairs from the
corpus.
training_data = []
for sentence in tokenized_corpus:
    for i, target_word in enumerate(sentence):
        target_idx = word2idx[target_word]
        # Create context words (simplified: just the word itself as
context)
        context_word = target_word # Using the target word as context
for simplicity
        if context_word in model.wv: # Check if word exists in the
gensim model's vocabulary
            context_embedding = model.wv[context_word]
            training_data.append((context_embedding, target_idx))

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, SimpleRNN, LSTM,
Activation, Input
import tensorflow as tf

model_rnn=Sequential()
model_rnn.add(Input(shape=(vocab_size, vector_size)))

```

```
model_rnn.add(SimpleRNN(24, return_sequences=False, activation="tanh"))
model_rnn.add(Dense(vocab_size))
model_rnn.add(Activation("softmax"))
model_rnn.summary()
```

```
print(training_data)
```

```
import numpy as np
from tensorflow.keras.utils import to_categorical

# Convert training data to NumPy arrays
# Assuming training_data is a list of (context_embedding, target_idx)

# Create input sequences (context word embeddings)
X = np.array([context_embedding for context_embedding, target_idx in
training_data])

# Create target outputs (target word index)
y = np.array([target_idx for context_embedding, target_idx in
training_data])

# Reshape X to be (samples, time steps, features) - for a single word
input, time steps is 1
X = X.reshape(-1, 1, vector_size) # Reshape to (samples, 1, 300)

# One-hot encode the target outputs
y = to_categorical(y, num_classes=vocab_size)

print("Shape of X:", X.shape)
print("Shape of y:", y.shape)
```

```
# Compile the model
model_rnn.compile(optimizer='adam', loss='categorical_crossentropy')
```

```
# Train the model
history = model_rnn.fit(X, y, epochs=100, batch_size=8,
validation_split=0.2, verbose=1)
```

```
# Evaluate the model (simple prediction example)

# Choose a word from the vocabulary to test
test_word = "deep"

# Get the embedding for the test word
```

```

if test_word in model.wv:
    test_embedding = model.wv[test_word]

    # Reshape the embedding to match the input shape of the RNN
    test_input = test_embedding.reshape(1, 1, vector_size)

    # Get the model's prediction (probability distribution over the
    vocabulary)
    prediction = model_rnn.predict(test_input)

    # Get the index of the predicted word
    predicted_word_index = np.argmax(prediction)

    # Get the predicted word from the index
    predicted_word = idx2word[predicted_word_index]

    print(f"The model predicts that the next word after '{test_word}'
is: '{predicted_word}'")
else:
    print(f"The word '{test_word}' is not in the vocabulary.")

```

5. Build and demonstrate an auto encoder network using neural layers for data compression on image dataset.

Soln:

```

!pip install tensorflow

from tensorflow.keras.datasets import mnist
import numpy as np

# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalize the pixel values
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

# Reshape the images to flatten them
x_train_flat = x_train.reshape((len(x_train),
np.prod(x_train.shape[1:])))
x_test_flat = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))

# Display the shapes of the flattened data
print("Shape of x_train_flat:", x_train_flat.shape)
print("Shape of x_test_flat:", x_test_flat.shape)

```

Output:

Shape of x_train_flat: (60000, 784)
Shape of x_test_flat: (10000, 784)

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

# Define the input layer
input_img = Input(shape=(784,))

# Define the encoder
encoded = Dense(128, activation='relu')(input_img)
encoded = Dense(64, activation='relu')(encoded)
encoded = Dense(32, activation='relu')(encoded) # Bottleneck layer

# Define the decoder
decoded = Dense(64, activation='relu')(encoded)
decoded = Dense(128, activation='relu')(decoded)
decoded = Dense(784, activation='sigmoid')(decoded) # Output layer

# Create the autoencoder model
autoencoder = Model(input_img, decoded)

# Display the model summary
autoencoder.summary()
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 784)	0
dense (Dense)	(None, 128)	100,480
dense_1 (Dense)	(None, 64)	8,256
dense_2 (Dense)	(None, 32)	2,080
dense_3 (Dense)	(None, 64)	2,112
dense_4 (Dense)	(None, 128)	8,320
dense_5 (Dense)	(None, 784)	101,136

Total params: 222,384 (868.69 KB)
Trainable params: 222,384 (868.69 KB)
Non-trainable params: 0 (0.00 B)

```

autoencoder.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

# Train the autoencoder model
history = autoencoder.fit(x_train_flat, x_train_flat,
                          epochs=50,
                          batch_size=256,
                          shuffle=True,
                          validation_data=(x_test_flat, x_test_flat))

# Evaluate the model
test_loss, test_acc = autoencoder.evaluate(x_test_flat, x_test_flat)
print(f'Test accuracy: {test_acc}')

# Make predictions
# predictions = model.predict(X_test)

```

Output:

```

313/313 ————— 1s 3ms/step - accuracy:
0.0146 - loss: 0.0840
Test accuracy: 0.012799999676644802

```

6. Design and implement a deep learning network for classification of textual documents

Soln:

```

import pandas as pd
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import numpy as np

# Assume data is in a CSV file named 'text_data.csv' with columns
# 'text' and 'label'
# Replace with the actual path to your data file if it's different.
try:
    df = pd.read_csv('text_data.csv')
except FileNotFoundError:
    # Create a dummy DataFrame if the file is not found
    data = {'text': ['This is the first document.', 'This document is
the second document.', 'And this is the third document.', 'Is this the
first document?'],
            'label': [0, 0, 1, 1]}
    df = pd.DataFrame(data)

texts = df['text'].tolist()

```

```

labels = df['label'].tolist()

# Tokenize the text data
max_words = 1000 # Maximum number of words to keep based on word
frequency
tokenizer = Tokenizer(num_words=max_words, oov_token="<OOV>")
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)

# Pad the numerical sequences
max_sequence_length = 20 # Maximum length of each sequence
padded_sequences = pad_sequences(sequences, maxlen=max_sequence_length,
padding='post', truncating='post')

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(padded_sequences,
labels, test_size=0.2, random_state=42)

print("Preprocessing complete.")
print(f"Shape of X_train: {X_train.shape}")
print(f"Shape of X_test: {X_test.shape}")
print(f"Length of y_train: {len(y_train)}")
print(f"Length of y_test: {len(y_test)}")

```

Output:

```

Preprocessing complete.
Shape of X_train: (3, 20)
Shape of X_test: (1, 20)
Length of y_train: 3
Length of y_test: 1

```

```

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense,
GlobalAveragePooling1D

# Define vocabulary size, adding 1 for the padding token.
vocab_size = len(tokenizer.word_index) + 1

# Define embedding dimension
embedding_dim = 16

# Create a Sequential model
model = Sequential([
    Embedding(input_dim=vocab_size, output_dim=embedding_dim,
input_length=max_sequence_length),

```

```

        GlobalAveragePooling1D(), # Using GlobalAveragePooling1D for
simplicity
        Dense(16, activation='relu'),
        Dense(1, activation='sigmoid') # Binary classification
    ])

# Print the model summary
model.summary()

```

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	?	0 (unbuilt)
global_average_pooling1d (GlobalAveragePooling1D)	?	0
dense (Dense)	?	0 (unbuilt)
dense_1 (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)
 Trainable params: 0 (0.00 B)
 Non-trainable params: 0 (0.00 B)

```

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

```

```

y_train = np.array(y_train)
y_test = np.array(y_test)

history = model.fit(X_train, y_train, epochs=10,
                    validation_data=(X_test, y_test))

```

```

loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss}")
print(f"Test Accuracy: {accuracy}")

```

Output:

```

1/1 ----- 0s 40ms/step - accuracy: 0.0000e+00
- loss: 0.7251
Test Loss: 0.725127100944519
Test Accuracy: 0.0

```

7) Design and implement a deep learning network for forecasting time series data.

Soln:

```

import pandas as pd
import numpy as np

```



```
2023-01-01 00:00:00    61.010565    55.933366
2023-01-01 01:00:00    95.251382    21.641565
2023-01-01 02:00:00    16.314052    33.378861
2023-01-01 03:00:00    16.714374    75.041250
2023-01-01 04:00:00    43.145889    80.669552
/tmp/ipython-input-922753698.py:24: FutureWarning: DataFrame.fillna with
'method' is deprecated and will raise in a future version. Use obj.ffill()
or obj.bfill() instead.
    df.fillna(method='ffill', inplace=True)
```

```
# Create a dummy time series dataset
dates = pd.date_range(start='2023-01-01', periods=1000, freq='H')
data = np.random.rand(1000, 2) * 100
df_dummy = pd.DataFrame(data, index=dates, columns=['Feature1',
'Feature2'])
df_dummy.index.name = 'Timestamp'

# Introduce some missing values for demonstration
df_dummy.iloc[50:150, 0] = np.nan
df_dummy.iloc[200:300, 1] = np.nan

# Save the dummy dataset
df_dummy.to_csv('time_series_data.csv')

# Load the dataset
df = pd.read_csv('time_series_data.csv')

# Inspect the data
print(df.head())
print(df.info())
print(df.isnull().sum())

# Assuming the time column is named 'Timestamp' and is in a format
pandas can recognize
# Convert the time column to datetime and set it as index
if 'Timestamp' in df.columns:
    df['Timestamp'] = pd.to_datetime(df['Timestamp'])
    df.set_index('Timestamp', inplace=True)
    df.sort_index(inplace=True)
else:
    print("Time column 'Timestamp' not found. Please check the column
name.")

# Handle missing values (example: forward fill)
df.fillna(method='ffill', inplace=True)

# Display the first few rows of the processed data
print(df.head())
```

```

# Split the data into training and testing sets (e.g., 80% train, 20%
test)
train_size = int(len(df) * 0.8)
train_df = df.iloc[:train_size]
test_df = df.iloc[train_size:]

print("\nTraining set shape:", train_df.shape)
print("Testing set shape:", test_df.shape)

# Scale the numerical features
scaler = MinMaxScaler()

# Fit on training data and transform training data
train_scaled = scaler.fit_transform(train_df)
train_df_scaled = pd.DataFrame(train_scaled, index=train_df.index,
columns=train_df.columns)

# Transform testing data
test_scaled = scaler.transform(test_df)
test_df_scaled = pd.DataFrame(test_scaled, index=test_df.index,
columns=test_df.columns)

print("\nScaled Training set head:")
print(train_df_scaled.head())

print("\nScaled Testing set head:")
print(test_df_scaled.head())

```

Output:

```

    Timestamp  Feature1  Feature2
0  2023-01-01 00:00:00  62.882923  23.572583
1  2023-01-01 01:00:00  83.105437  90.538728
2  2023-01-01 02:00:00  43.019371  52.599986
3  2023-01-01 03:00:00  63.053692  26.974901
4  2023-01-01 04:00:00  25.107571  68.721651
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype
---  -
 0   Timestamp  1000 non-null   object
 1   Feature1    900 non-null    float64
 2   Feature2    900 non-null    float64
dtypes: float64(2), object(1)
memory usage: 23.6+ KB
None
Timestamp      0
Feature1       100
Feature2       100
dtype: int64

          Feature1  Feature2
Timestamp

```

2023-01-01 00:00:00	62.882923	23.572583
2023-01-01 01:00:00	83.105437	90.538728
2023-01-01 02:00:00	43.019371	52.599986
2023-01-01 03:00:00	63.053692	26.974901
2023-01-01 04:00:00	25.107571	68.721651

Training set shape: (800, 2)

Testing set shape: (200, 2)

Scaled Training set head:

	Feature1	Feature2
Timestamp		
2023-01-01 00:00:00	0.630711	0.235412
2023-01-01 01:00:00	0.833706	0.905825
2023-01-01 02:00:00	0.431320	0.526012
2023-01-01 03:00:00	0.632426	0.269473
2023-01-01 04:00:00	0.251520	0.687410

Scaled Testing set head:

	Feature1	Feature2
Timestamp		
2023-02-03 08:00:00	0.982875	0.909401
2023-02-03 09:00:00	0.139342	0.783821
2023-02-03 10:00:00	0.805157	0.899454
2023-02-03 11:00:00	0.454503	0.840185
2023-02-03 12:00:00	0.876427	0.565804

/tmp/ipython-input-728994618.py:2: FutureWarning: 'H' is deprecated and will be removed in a future version, please use 'h' instead.

dates = pd.date_range(start='2023-01-01', periods=1000, freq='H')

/tmp/ipython-input-728994618.py:32: FutureWarning: DataFrame.fillna with 'method' is deprecated and will raise in a future version. Use obj.ffill() or obj.bfill() instead.

df.fillna(method='ffill', inplace=True)

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

# Define the number of time steps (sequence length) to consider for
forecasting
n_steps = 24 # Example: Use the past 24 hours to predict the next time
step

# Define the number of features in the dataset
n_features = train_df_scaled.shape[1]

# Define the input shape for the LSTM layers
# The input shape is (number of time steps, number of features)
input_shape = (n_steps, n_features)

# Choose a deep learning architecture (LSTM)
model = Sequential()

# Add LSTM layers
```

```

# return_sequences=True is needed to stack multiple LSTM layers
model.add(LSTM(units=50, activation='relu', input_shape=input_shape,
return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(units=50, activation='relu'))
model.add(Dropout(0.2))

# Add Dense layers
# The number of units in the output layer should match the number of
features to forecast
model.add(Dense(units=n_features))

# Print the model summary
model.summary()

```

Output:

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 24, 50)	10,600
dropout (Dropout)	(None, 24, 50)	0
lstm_1 (LSTM)	(None, 50)	20,200
dropout_1 (Dropout)	(None, 50)	0
dense (Dense)	(None, 2)	102

Total params: 30,902 (120.71 KB)
 Trainable params: 30,902 (120.71 KB)
 Non-trainable params: 0 (0.00 B)

```

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

# Print the model summary after compilation
model.summary()

```

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 24, 50)	10,600
dropout (Dropout)	(None, 24, 50)	0
lstm_1 (LSTM)	(None, 50)	20,200

dropout_1 (Dropout)	(None, 50)	0
dense (Dense)	(None, 2)	102

Total params: 30,902 (120.71 KB)
Trainable params: 30,902 (120.71 KB)
Non-trainable params: 0 (0.00 B)

```

# Function to create sequences and targets
def create_sequences(data, n_steps):
    X, y = [], []
    for i in range(len(data) - n_steps):
        X.append(data.iloc[i:(i + n_steps)].values)
        y.append(data.iloc[i + n_steps].values)
    return np.array(X), np.array(y)

# Prepare training sequences and targets
X_train, y_train = create_sequences(train_df_scaled, n_steps)

# Prepare validation sequences and targets
X_test, y_test = create_sequences(test_df_scaled, n_steps)

# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32,
                    validation_data=(X_test, y_test), verbose=1)

from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score
import numpy as np

# 1. Use the trained model to make predictions on the X_test data
y_pred_scaled = model.predict(X_test)

# 2. Inverse transform the predictions and the actual y_test values
# The scaler was fitted on the entire training dataframe (train_df)
# which had 2 features.
# To inverse transform, we need to provide data with the same number of
# features as the training data.
# Since y_test_scaled and y_pred_scaled have the shape (n_samples,
# n_features), we can directly use the inverse_transform method.
y_test_actual = scaler.inverse_transform(y_test)
y_pred_actual = scaler.inverse_transform(y_pred_scaled)

# 3. Calculate the Root Mean Squared Error (RMSE) and Mean Absolute
Error (MAE)
rmse = np.sqrt(mean_squared_error(y_test_actual, y_pred_actual))
mae = mean_absolute_error(y_test_actual, y_pred_actual)

```

```

# 4. Print the calculated RMSE and MAE
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"Mean Absolute Error (MAE): {mae}")

# 5. Calculate and print the R-squared score
r2 = r2_score(y_test_actual, y_pred_actual)
print(f"R-squared (R2) Score: {r2}")

```

6/6 ————— 0s 41ms/step
 Root Mean Squared Error (RMSE): 29.94993697154205
 Mean Absolute Error (MAE): 25.87683730182286
 R-squared (R2) Score: -0.0031254272579815945

8) Write a program to read a dataset of text reviews. Classify the reviews as positive or negative.

Soln:

```
!pip install transformers datasets pandas
```

```
from datasets import load_dataset
```

```
# Load the IMDB dataset
```

```
dataset = load_dataset("imdb")
```

```
# Display some examples from the training set
```

```
print(dataset['train'][0])
```

```
print(dataset['train'][1])
```

```
from transformers import pipeline
```

```
# Load a pre-trained sentiment analysis pipeline
```

```
classifier = pipeline("sentiment-analysis")
```

```
# Example review
```

```
review = "This movie was amazing! I loved every part of it."
```

```
# Classify the review
```

```
result = classifier(review)
```

```
# Print the result
```

```
print(f"Review: {review}")
```

```
print(f"Sentiment: {result[0]['label']], Score:
```

```
{result[0]['score']:.2f}")
```

Output:

Review: This movie was amazing! I loved every part of it.
Sentiment: POSITIVE, Score: 1.00

```
# Example reviews
reviews = [
    "This movie was great!",
    "This movie was terrible.",
    "It was an okay movie, nothing special."
]

# Classify the reviews
results = classifier(reviews)

# Print the results
for review, result in zip(reviews, results):
    print(f"Review: {review}")
    print(f"Sentiment: {result['label']}, Score: {result['score']:.2f}")
```

Output:

Review: This movie was great!
Sentiment: POSITIVE, Score: 1.00
Review: This movie was terrible.
Sentiment: NEGATIVE, Score: 1.00
Review: It was an okay movie, nothing special.
Sentiment: NEGATIVE, Score: 0.96