EXPERIMENT-1

Aim : Write a program in deep learning for Learning XOR Problem

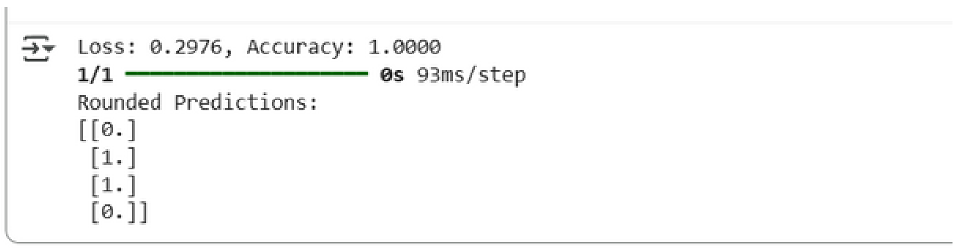
Procedure : -

- import required libraries
- prepare the XOR dataset
- build and compile the model
- Train and predict

Input :

```
import numpy as np                        # 1
from tensorflow.keras.models import Sequential  # 2
from tensorflow.keras.layers import Dense

X = np.array([[0,0], [0,1], [1,0], [1,1]])  # 4
y = np.array([[0],   [1],   [1],   [0]])    # 5

# Build the model
model = Sequential()                      # 6
model.add(Dense(4, input_dim=2, activation='tanh'))  # 7
model.add(Dense(1, activation='sigmoid'))        # 8

np.random.seed(0)



# Compile the model
model.compile(
    optimizer='adam',
   loss='binary_crossentropy',
   metrics=['accuracy']
)
model.fit(X, y, epochs=1000, verbose=0)       # 10
loss, accuracy = model.evaluate(X, y, verbose=0)
print(f'Loss: {loss:.4f}, Accuracy: {accuracy:.4f}')

predictions = model.predict(X)
print('Rounded Predictions:')
print(np.round(predictions))
```

Output :



```
Loss: 0.2976, Accuracy: 1.0000
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 93ms/step
Rounded Predictions:
[[0.]
 [1.]
 [1.]
 [0.]]
```

Result : The program executed successfully.

EXPERIMENT-2

Aim : Write a program in deep learning for Image Classification using CNN

Procedure : -

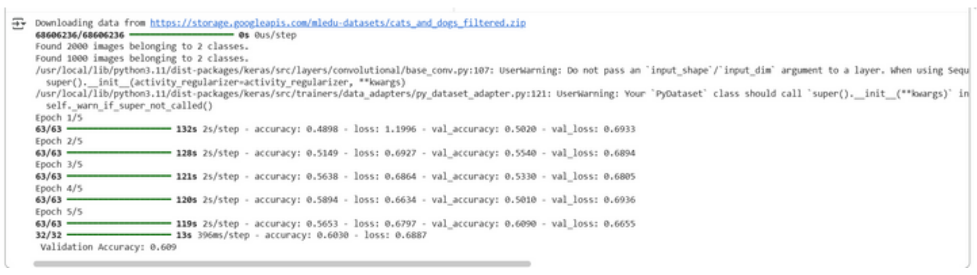
- import required libraries
- build and compile the model
- Train and predict

Input :

```
import os
import zipfile
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
url = "https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip"
zip_path = tf.keras.utils.get_file("cats_and_dogs_filtered.zip", origin=url, extract=False)

# 2. Unzip it manually to the current directory
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
  zip_ref.extractall(os.path.dirname(zip_path))

# 3. Define correct paths
base_dir = os.path.join(os.path.dirname(zip_path), "cats_and_dogs_filtered")
train_dir = os.path.join(base_dir, "train")
val_dir = os.path.join(base_dir, "validation")
train_gen = ImageDataGenerator(
 rescale=1./255,
 rotation_range=20,
 width_shift_range=0.2,
 height_shift_range=0.2,
 horizontal_flip=True)
al_gen = ImageDataGenerator(rescale=1./255)
```

```
train_data = train_gen.flow_from_directory(
  train_dir,
  target_size=(150, 150),
  batch_size=32,
  class_mode='binary'
)

val_data = val_gen.flow_from_directory(
  val_dir,
  target_size=(150, 150),
  batch_size=32,
  class_mode='binary'
)
model = Sequential([
  Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
  MaxPooling2D((2, 2)),
  Conv2D(64, (3, 3), activation='relu'),
  MaxPooling2D((2, 2)),
  Conv2D(128, (3, 3), activation='relu'),
  MaxPooling2D((2, 2)),
  Flatten(),
  Dense(512, activation='relu'),
  Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

history = model.fit(
  train_data,
  epochs=5,
  validation_data=val_data
)

# 7. Evaluate
val_loss, val_acc = model.evaluate(val_data)
print(f" Validation Accuracy: {val_acc:.3f}")
```

Output :



```
Downloading data from https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip
68606236/68606236 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequ
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
  self._warn_if_super_not_called()
Epoch 1/5
63/63 ━━━━━━━━━━━━━━━━━━━━ 132s 2s/step - accuracy: 0.4898 - loss: 1.1996 - val_accuracy: 0.5020 - val_loss: 0.6933
Epoch 2/5
63/63 ━━━━━━━━━━━━━━━━━━━━ 128s 2s/step - accuracy: 0.5149 - loss: 0.6927 - val_accuracy: 0.5540 - val_loss: 0.6894
Epoch 3/5
63/63 ━━━━━━━━━━━━━━━━━━━━ 121s 2s/step - accuracy: 0.5638 - loss: 0.6864 - val_accuracy: 0.5330 - val_loss: 0.6805
Epoch 4/5
63/63 ━━━━━━━━━━━━━━━━━━━━ 120s 2s/step - accuracy: 0.5894 - loss: 0.6634 - val_accuracy: 0.5010 - val_loss: 0.6936
Epoch 5/5
63/63 ━━━━━━━━━━━━━━━━━━━━ 119s 2s/step - accuracy: 0.5653 - loss: 0.6797 - val_accuracy: 0.6090 - val_loss: 0.6655
32/32 ━━━━━━━━━━━━━━━━━━━━ 13s 396ms/step - accuracy: 0.6030 - loss: 0.6887
 Validation Accuracy: 0.609
```

Result : The program executed successfully.

EXPERIMENT-3

Aim : Write a program in deep learning for building a deep learning model

Procedure : -

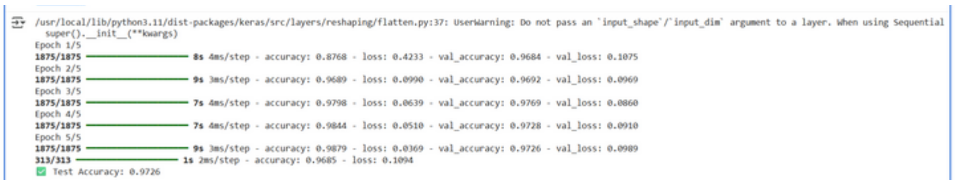
- import required libraries
- build and compile the model
- Train and predict

Input :

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
# 1. Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
# 2. Normalize pixel values (0–255 → 0–1)
x_train = x_train / 255.0
x_test = x_test / 255.0
# 3. One-hot encode the labels (0–9 → [0,0,0,1,0,...])
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
model = Sequential([
Flatten(input_shape=(28, 28)), # Converts 2D image to 1D
Dense(128, activation = 'relu'), # Hidden layer 1
Dense(64, activation= 'relu'), # Hidden layer 2
Dense(10, activation='softmax') # Output layer for 10 digits
])
model.compile(
optimizer= 'adam',
loss= 'categorical_crossentropy',
metrics=['accuracy'])
```

```
history = model.fit(
x_train, y_train,
epochs=5,
validation_data=(x_test, y_test)
)
# 7. Evaluate the model
loss, accuracy = model.evaluate(x_test, y_test)
print(f"✅ Test Accuracy: {accuracy:.4f}")
```

Output :



```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
  super().__init__(**kwargs)
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.8768 - loss: 0.4233 - val_accuracy: 0.9684 - val_loss: 0.1075
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - accuracy: 0.9689 - loss: 0.0990 - val_accuracy: 0.9692 - val_loss: 0.0969
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9798 - loss: 0.0639 - val_accuracy: 0.9769 - val_loss: 0.0860
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9844 - loss: 0.0510 - val_accuracy: 0.9728 - val_loss: 0.0910
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - accuracy: 0.9879 - loss: 0.0369 - val_accuracy: 0.9726 - val_loss: 0.0989
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9685 - loss: 0.1094
✅ Test Accuracy: 0.9726
```

Result : The program executed successfully.

EXPERIMENT-4

Aim : **Write a program in deep learning for building a Data Augmentation lab**

Procedure : -

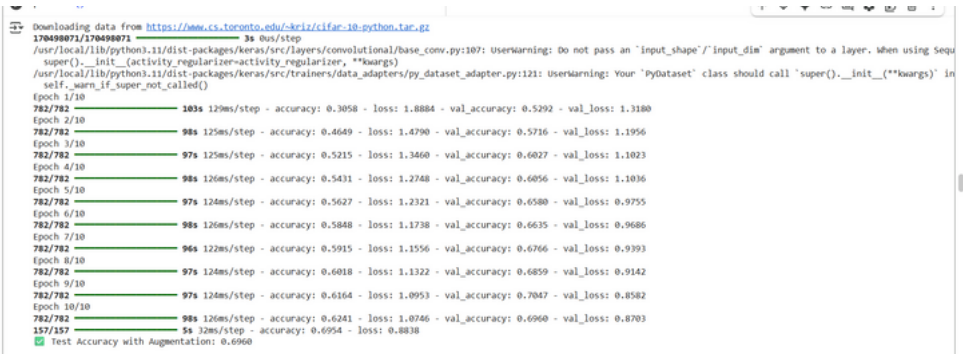
- import required libraries
- build and compile the model
- Train and predict

Input :

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,Dense, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical

(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
train_datagen = ImageDataGenerator(
rotation_range=15,
width_shift_range=0.1,
height_shift_range=0.1,
horizontal_flip=True,
)
test_datagen = ImageDataGenerator()
# 5. Create data generators
train_generator = train_datagen.flow(x_train, y_train, batch_size=64)
test_generator = test_datagen.flow(x_test, y_test, batch_size=64)
```

```
model = Sequential([
Conv2D(32, (3,3), activation='relu', padding='same',
input_shape=(32,32,3)),
MaxPooling2D((2,2)),
Conv2D(64, (3,3), activation='relu', padding='same'),
MaxPooling2D((2,2)),
Flatten(),
Dense(128, activation='relu'),
Dropout(0.5),
Dense(10, activation='softmax')
])
# 7. Compile the model
model.compile(optimizer='adam',
loss='categorical_crossentropy',
metrics=['accuracy'])
# 8. Train the model using augmented data
history = model.fit(
train_generator,
epochs=10,
validation_data=test_generator
)
# 9. Evaluate the model
loss, acc = model.evaluate(test_generator)
print(f"✅ Test Accuracy with Augmentation: {acc:.4f}")
```

Output :



```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 3s 0us/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequ
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
  self._warn_if_super_not_called()
Epoch 1/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 103s 129ms/step - accuracy: 0.3058 - loss: 1.8884 - val_accuracy: 0.5292 - val_loss: 1.3180
Epoch 2/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 98s 125ms/step - accuracy: 0.4649 - loss: 1.4790 - val_accuracy: 0.5716 - val_loss: 1.1956
Epoch 3/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 97s 125ms/step - accuracy: 0.5215 - loss: 1.3460 - val_accuracy: 0.6027 - val_loss: 1.1023
Epoch 4/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 98s 126ms/step - accuracy: 0.5431 - loss: 1.2748 - val_accuracy: 0.6056 - val_loss: 1.1036
Epoch 5/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 97s 124ms/step - accuracy: 0.5627 - loss: 1.2321 - val_accuracy: 0.6580 - val_loss: 0.9755
Epoch 6/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 98s 126ms/step - accuracy: 0.5848 - loss: 1.1738 - val_accuracy: 0.6635 - val_loss: 0.9686
Epoch 7/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 96s 122ms/step - accuracy: 0.5915 - loss: 1.1556 - val_accuracy: 0.6766 - val_loss: 0.9393
Epoch 8/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 97s 124ms/step - accuracy: 0.6018 - loss: 1.1322 - val_accuracy: 0.6859 - val_loss: 0.9142
Epoch 9/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 97s 124ms/step - accuracy: 0.6164 - loss: 1.0953 - val_accuracy: 0.7047 - val_loss: 0.8582
Epoch 10/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 98s 126ms/step - accuracy: 0.6241 - loss: 1.0746 - val_accuracy: 0.6960 - val_loss: 0.8703
157/157 ━━━━━━━━━━━━━━━━━━━━ 5s 32ms/step - accuracy: 0.6954 - loss: 0.8838
✅ Test Accuracy with Augmentation: 0.6960
```

Result : The program executed successfully.

<u>**EXPERIMENT-5**</u>

Aim : Write a program in deep learning for implementation of RNN

Procedure : -

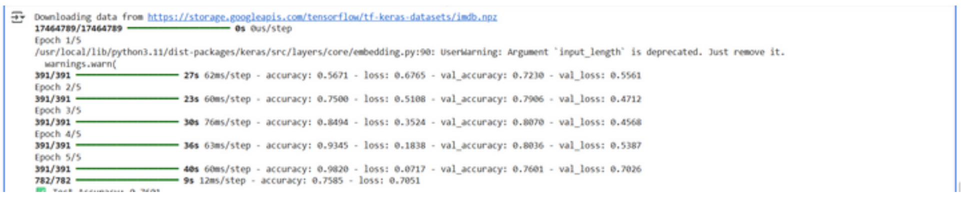
- import required libraries
- build and compile the model
- Train and predict

Input :

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
import matplotlib.pyplot as plt
num_words = 10000  # Top 10,000 frequent words
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=num_words)
maxlen = 200
x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)


# 3. Build the RNN model
model = Sequential([
  Embedding(input_dim=num_words, output_dim=32, input_length=maxlen),  # Converts
word indices to dense vectors
  SimpleRNN(32, return_sequences=False),  # RNN layer
  Dense(1, activation='sigmoid')  # Output layer for binary classification
])
```

```
# 4. Compile the model
model.compile(
  optimizer='adam',
  loss='binary_crossentropy',
  metrics=['accuracy']
)
# 5. Train the model
history = model.fit(
  x_train, y_train,
  epochs=5,
  batch_size=64,
  validation_data=(x_test, y_test)
)
# 6. Evaluate
loss, acc = model.evaluate(x_test, y_test)
print(f"✅ Test Accuracy: {acc:.4f}")
```

Output :



```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Epoch 1/5
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
391/391 ━━━━━━━━━━━━━━━━━━━━ 27s 62ms/step - accuracy: 0.5671 - loss: 0.6765 - val_accuracy: 0.7230 - val_loss: 0.5561
Epoch 2/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 23s 60ms/step - accuracy: 0.7500 - loss: 0.5108 - val_accuracy: 0.7906 - val_loss: 0.4712
Epoch 3/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 30s 76ms/step - accuracy: 0.8494 - loss: 0.3524 - val_accuracy: 0.8070 - val_loss: 0.4568
Epoch 4/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 36s 63ms/step - accuracy: 0.9345 - loss: 0.1838 - val_accuracy: 0.8036 - val_loss: 0.5387
Epoch 5/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 40s 60ms/step - accuracy: 0.9820 - loss: 0.0717 - val_accuracy: 0.7601 - val_loss: 0.7026
782/782 ━━━━━━━━━━━━━━━━━━━━ 9s 12ms/step - accuracy: 0.7585 - loss: 0.7051
✅ Test Accuracy: 0.7601
```

Result : The program executed successfully.

EXPERIMENT-6

Aim : Write a program in deep learning for restricted boltzman machine

Procedure : -

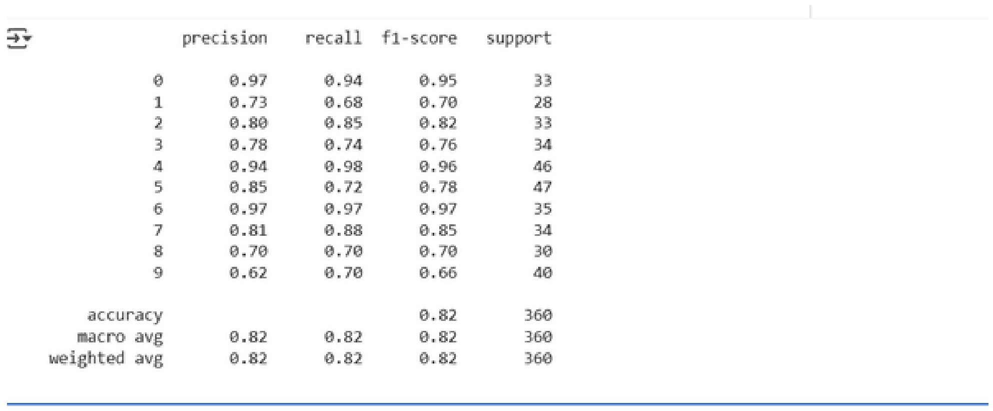
- import required libraries
- build and compile the model
- Train and predict

Input :

```
from sklearn.neural_network import BernoulliRBM
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import classification_report

# Load dataset (digits is small & binary-friendly)
digits = load_digits()
X = digits.data
y = digits.target

# Normalize pixel values between 0 and 1
X = MinMaxScaler().fit_transform(X)

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4
rbm = BernoulliRBM(n_components=64, learning_rate=0.06, n_iter=10, random_state=0)
logistic = LogisticRegression(max_iter=1000)
rbm_pipeline = Pipeline(steps=[('rbm', rbm), ('logistic', logistic)])
```

```
rbm_pipeline.fit(X_train, y_train)

# Predict and evaluate
y_pred = rbm_pipeline.predict(X_test)
print(classification_report(y_test, y_pred))
```

Output :



```
              precision    recall  f1-score   support

           0       0.97      0.94      0.95        33
           1       0.73      0.68      0.70        28
           2       0.80      0.85      0.82        33
           3       0.78      0.74      0.76        34
           4       0.94      0.98      0.96        46
           5       0.85      0.72      0.78        47
           6       0.97      0.97      0.97        35
           7       0.81      0.88      0.85        34
           8       0.70      0.70      0.70        30
           9       0.62      0.70      0.66        40

    accuracy                           0.82       360
   macro avg       0.82      0.82      0.82       360
weighted avg       0.82      0.82      0.82       360
```

Result : The program executed successfully.

EXPERIMENT-7

Aim : Write a program in deep learning for generative adversial network

Procedure : -

- import required libraries
- build and compile the model
- Train and predict

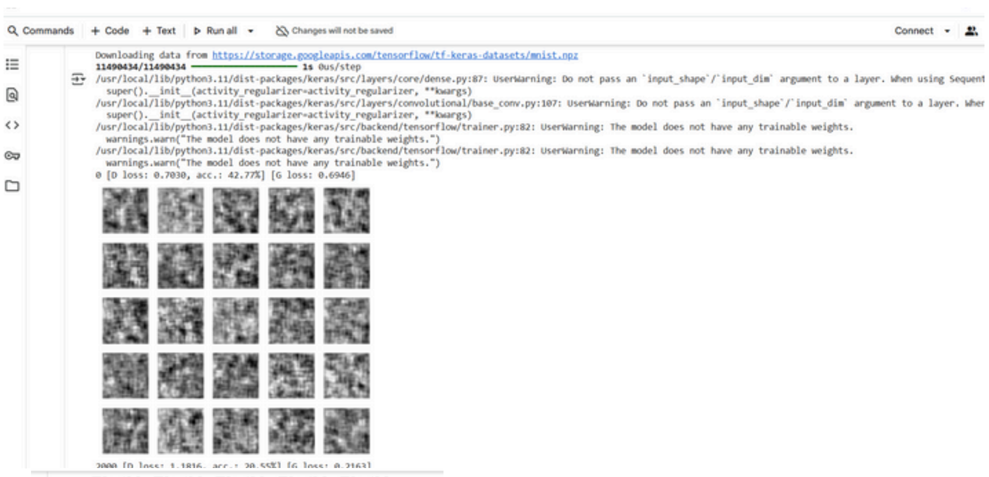
Input :

```
import tensorflow as tf
from tensorflow.keras import layers
import numpy as np
import matplotlib.pyplot as plt

# Step 2: Load and Normalize MNIST Dataset
(x_train, _), (_, _) = tf.keras.datasets.mnist.load_data()
x_train = (x_train - 127.5) / 127.5  # Normalize to [-1, 1]
x_train = x_train.reshape((-1, 28, 28, 1)).astype('float32')

# Step 3: Define the Generator Model
def build_generator():
    model = tf.keras.Sequential([
        layers.Dense(128, input_shape=(100,), activation='relu'),
        layers.BatchNormalization(),
        layers.Dense(7*7*128, activation='relu'),
        layers.Reshape((7, 7, 128)),
        layers.UpSampling2D(),
        layers.Conv2D(64, (3,3), padding='same', activation='relu'),
        layers.UpSampling2D(),
        layers.Conv2D(1, (3,3), padding='same', activation='tanh')
    ])
    return model
```

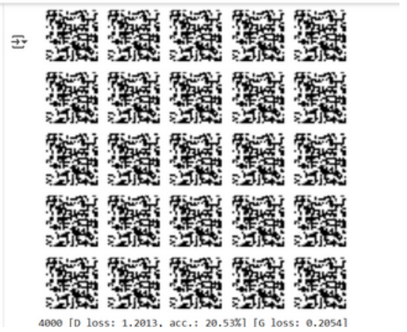
```
def build_discriminator():
    model = tf.keras.Sequential([
        layers.Conv2D(64, (3,3), strides=2, padding='same', input_shape=(28,28,1)),
        layers.LeakyReLU(0.2),
        layers.Dropout(0.3),
        layers.Conv2D(128, (3,3), strides=2, padding='same'),
        layers.LeakyReLU(0.2),
        layers.Dropout(0.3),
        layers.Flatten(),
        layers.Dense(1, activation='sigmoid')
    ])
    return model

# Step 5: Build and Compile the Models
generator = build_generator()
discriminator = build_discriminator()

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

# Combined GAN model
discriminator.trainable = False
gan_input = layers.Input(shape=(100,))
gan_output = discriminator(generator(gan_input))
gan = tf.keras.Model(gan_input, gan_output)
gan.compile(optimizer='adam', loss='binary_crossentropy')
epochs = 10000
batch_size = 128
noise_dim = 100
sample_interval = 2000

for epoch in range(epochs):
    # Train Discriminator
    idx = np.random.randint(0, x_train.shape[0], batch_size)
    real_imgs = x_train[idx]
```

```
noise = np.random.normal(0, 1, (batch_size, noise_dim))
fake_imgs = generator.predict(noise, verbose=0)

d_loss_real = discriminator.train_on_batch(real_imgs, np.ones((batch_size, 1)))
d_loss_fake = discriminator.train_on_batch(fake_imgs, np.zeros((batch_size, 1)))
d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

# Train Generator
noise = np.random.normal(0, 1, (batch_size, noise_dim))
g_loss = gan.train_on_batch(noise, np.ones((batch_size, 1)))

# Display progress
if epoch % sample_interval == 0:
    print(f"{epoch} [D loss: {d_loss[0]:.4f}, acc.: {100*d_loss[1]:.2f}%] [G loss: {g_loss:.4f}]")
    # Save generated images
    r, c = 5, 5
    noise = np.random.normal(0, 1, (r * c, noise_dim))
    gen_imgs = generator.predict(noise, verbose=0)
    gen_imgs = 0.5 * gen_imgs + 0.5  # Rescale back to [0, 1]

    fig, axs = plt.subplots(r, c, figsize=(5, 5))
    cnt = 0
    for i in range(r):
        for j in range(c):
            axs[i, j].imshow(gen_imgs[cnt, :, :, 0], cmap='gray')
            axs[i, j].axis('off')
            cnt += 1
    plt.show()
```

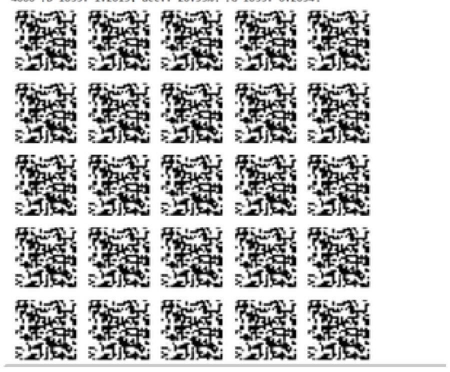
Output :



```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequent
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. Wher
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.11/dist-packages/keras/src/backend/tensorflow/trainer.py:82: UserWarning: The model does not have any trainable weights.
  warnings.warn("The model does not have any trainable weights.")
/usr/local/lib/python3.11/dist-packages/keras/src/backend/tensorflow/trainer.py:82: UserWarning: The model does not have any trainable weights.
  warnings.warn("The model does not have any trainable weights.")
0 [D loss: 0.7030, acc.: 42.77%] [G loss: 0.6946]
2000 [D loss: 1.1816, acc.: 20.55%] [G loss: 0.2163]
```



```
4000 [D loss: 1.2013, acc.: 20.53%] [G loss: 0.2054]
```



Result : The program executed successfully.

EXPERIMENT-8

Aim : **Write a program in deep learning for variational autoencoder**

Procedure : -

- import required libraries
- build and compile the model
- Train and predict

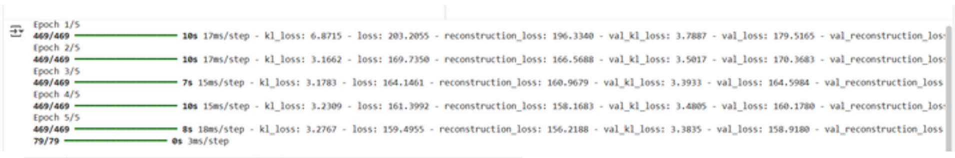
Input :

```
import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np
import matplotlib.pyplot as plt

# 1. Load MNIST
(x_train, _), (x_test, _) = tf.keras.datasets.mnist.load_data()
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0
x_train = x_train.reshape(-1, 784)
x_test = x_test.reshape(-1, 784)

latent_dim = 2

# 2. Encoder
inputs = layers.Input(shape=(784,))
h = layers.Dense(256, activation="relu")(inputs)
z_mean = layers.Dense(latent_dim)(h)
z_log_var = layers.Dense(latent_dim)(h)
def sampling(args):
    z_mean, z_log_var = args
    eps = tf.random.normal(shape=tf.shape(z_mean))
    return z_mean + tf.exp(0.5 * z_log_var) * eps
z = layers.Lambda(sampling)([z_mean, z_log_var])
```

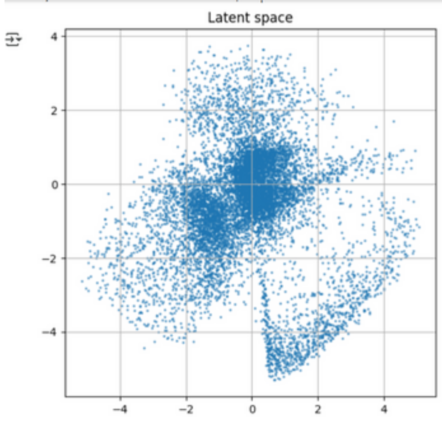
```
decoder_h = layers.Dense(256, activation="relu")
decoder_out = layers.Dense(784, activation="sigmoid")
h_decoded = decoder_h(z)
outputs = decoder_out(h_decoded)
encoder = models.Model(inputs, [z_mean, z_log_var, z])
decoder_input = layers.Input(shape=(latent_dim,))
_h_decoded = decoder_h(decoder_input)
_x_decoded = decoder_out(_h_decoded)
decoder = models.Model(decoder_input, _x_decoded)

class VAE(models.Model):
    def __init__(self, encoder, decoder, **kwargs):
        super(VAE, self).__init__(**kwargs)
        self.encoder = encoder
        self.decoder = decoder
    def train_step(self, data):
        if isinstance(data, tuple):  # unpack if (x, y)
            data = data[0]
        with tf.GradientTape() as tape:
            z_mean, z_log_var, z = self.encoder(data)
            reconstruction = self.decoder(z)
            # reconstruction loss
            recon_loss = tf.reduce_mean(
                tf.keras.losses.binary_crossentropy(data, reconstruction)
            ) * 784
            # KL divergence
            kl_loss = -0.5 * tf.reduce_mean(
                1 + z_log_var - tf.square(z_mean) - tf.exp(z_log_var)
            )
            total_loss = recon_loss + kl_loss
        grads = tape.gradient(total_loss, self.trainable_variables)
        self.optimizer.apply_gradients(zip(grads, self.trainable_variables))
        return {"loss": total_loss, "reconstruction_loss": recon_loss, "kl_loss": kl_loss}
```

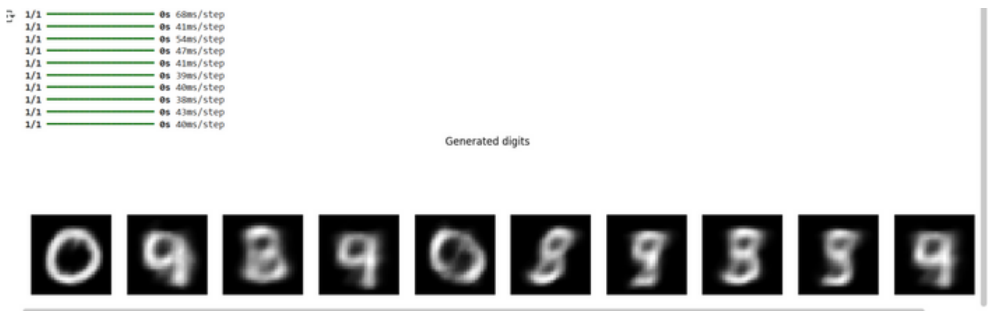
```
  def test_step(self, data):
        if isinstance(data, tuple):
            data = data[0]
        z_mean, z_log_var, z = self.encoder(data)
        reconstruction = self.decoder(z)
        recon_loss = tf.reduce_mean(
            tf.keras.losses.binary_crossentropy(data, reconstruction)
        ) * 784
        kl_loss = -0.5 * tf.reduce_mean(
            1 + z_log_var - tf.square(z_mean) - tf.exp(z_log_var)
        )
        total_loss = recon_loss + kl_loss
        return {"loss": total_loss, "reconstruction_loss": recon_loss, "kl_loss": kl_loss}
vae = VAE(encoder, decoder)
vae.compile(optimizer="adam")
history = vae.fit(x_train, x_train,
                  epochs=5,
                  batch_size=128,
                  validation_data=(x_test, x_test))
z_mean, _, _ = encoder.predict(x_test, batch_size=128)
plt.figure(figsize=(6,6))
plt.scatter(z_mean[:,0], z_mean[:,1], alpha=0.5, s=2)
plt.title("Latent space")
plt.grid()
plt.show()
n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
    z_sample = np.random.normal(size=(1, latent_dim))
    generated = decoder.predict(z_sample)
    plt.subplot(1, n, i + 1)
    plt.imshow(generated.reshape(28,28), cmap="gray")
    plt.axis("off")
plt.suptitle("Generated digits")
plt.show()
```

Output :



```
Epoch 1/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 10s 17ms/step - kl_loss: 6.8715 - loss: 203.2055 - reconstruction_loss: 196.3340 - val_kl_loss: 3.7887 - val_loss: 179.5165 - val_reconstruction_los
Epoch 2/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 10s 17ms/step - kl_loss: 3.1662 - loss: 169.7350 - reconstruction_loss: 166.5688 - val_kl_loss: 3.5017 - val_loss: 170.3683 - val_reconstruction_los
Epoch 3/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 7s 15ms/step - kl_loss: 3.1783 - loss: 164.1461 - reconstruction_loss: 160.9679 - val_kl_loss: 3.3933 - val_loss: 164.5984 - val_reconstruction_loss
Epoch 4/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 10s 15ms/step - kl_loss: 3.2309 - loss: 161.3992 - reconstruction_loss: 158.1683 - val_kl_loss: 3.4805 - val_loss: 160.1780 - val_reconstruction_los
Epoch 5/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 8s 18ms/step - kl_loss: 3.2767 - loss: 159.4955 - reconstruction_loss: 156.2188 - val_kl_loss: 3.3835 - val_loss: 158.9180 - val_reconstruction_loss
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
```





```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 68ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
```

Result : The program executed successfully.

EXPERIMENT-9

Aim : Write a program in deep learning for LSTM

Procedure : -

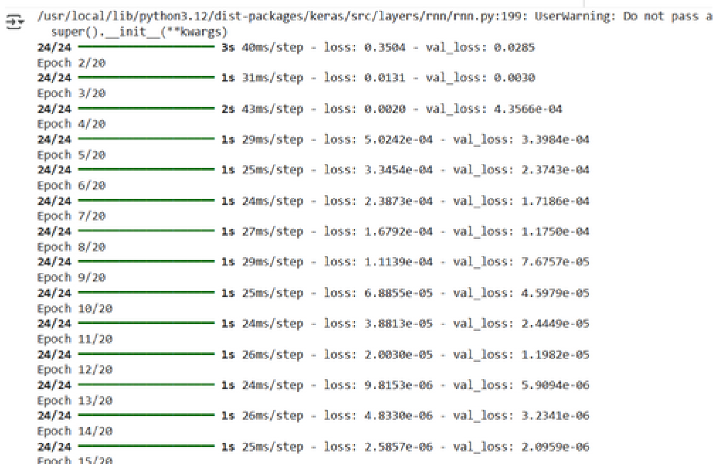
- import required libraries
- build and compile the model
- Train and predict

Input :

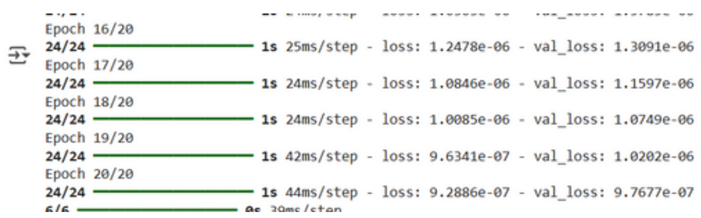
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
import matplotlib.pyplot as plt

# Step 1: Generate a synthetic time series (sine wave)
def generate_sine_wave_data(seq_length=50, total_samples=1000):
    x = np.linspace(0, 100, total_samples)
    y = np.sin(x)
    X = []
    Y = []
    for i in range(len(y) - seq_length):
        X.append(y[i:i + seq_length])
        Y.append(y[i + seq_length])
    X = np.array(X)
    Y = np.array(Y)
    return X, Y

# Parameters
SEQ_LENGTH = 50
X, Y = generate_sine_wave_data(SEQ_LENGTH)
# Reshape to LSTM input shape: [samples, time_steps, features]
X = X.reshape((X.shape[0], X.shape[1], 1))
```

```
split = int(len(X) * 0.8)
X_train, X_test = X[:split], X[split:]
Y_train, Y_test = Y[:split], Y[split:]

# Step 3: Build the LSTM model
model = Sequential([
    LSTM(64, input_shape=(SEQ_LENGTH, 1)),
    Dense(1)
])

model.compile(optimizer='adam', loss='mse')

# Step 4: Train the model
history = model.fit(X_train, Y_train, epochs=20, batch_size=32, validation_data=(X_test, Y_test))

# Step 5: Predict and plot
predicted = model.predict(X_test)

plt.figure(figsize=(10, 6))
plt.plot(Y_test, label='True')
plt.plot(predicted, label='Predicted')
plt.title("LSTM Time Series Prediction")
plt.legend()
plt.show()
```

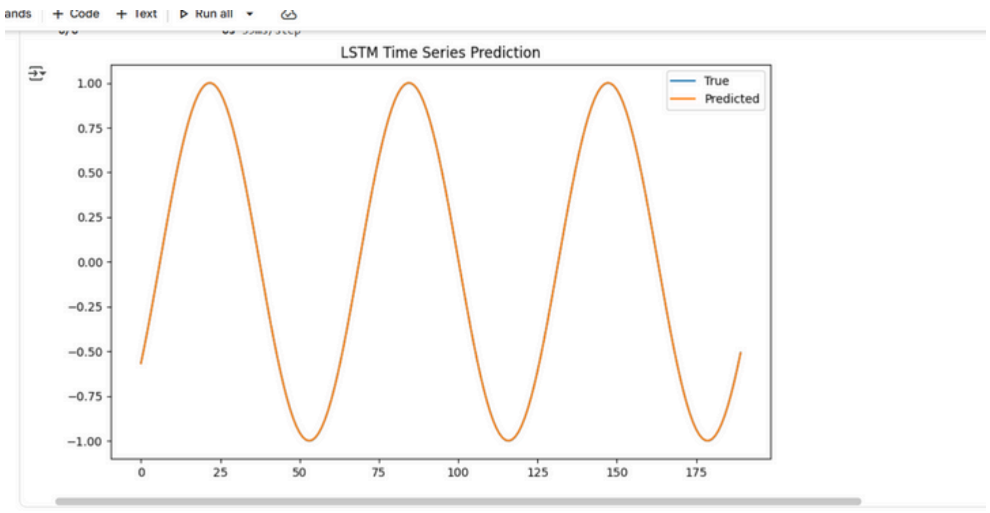
Output :



```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/rnn/rnn.py:199: UserWarning: Do not pass a
  super().__init__(**kwargs)
24/24 ━━━━━━━━━━━━━━━━━━━━ 3s 40ms/step - loss: 0.3504 - val_loss: 0.0285
Epoch 2/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 0.0131 - val_loss: 0.0030
Epoch 3/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 2s 43ms/step - loss: 0.0020 - val_loss: 4.3566e-04
Epoch 4/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - loss: 5.0242e-04 - val_loss: 3.3984e-04
Epoch 5/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 3.3454e-04 - val_loss: 2.3743e-04
Epoch 6/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - loss: 2.3873e-04 - val_loss: 1.7186e-04
Epoch 7/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - loss: 1.6792e-04 - val_loss: 1.1750e-04
Epoch 8/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - loss: 1.1139e-04 - val_loss: 7.6757e-05
Epoch 9/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 6.8855e-05 - val_loss: 4.5979e-05
Epoch 10/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - loss: 3.8813e-05 - val_loss: 2.4449e-05
Epoch 11/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - loss: 2.0030e-05 - val_loss: 1.1982e-05
Epoch 12/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - loss: 9.8153e-06 - val_loss: 5.9094e-06
Epoch 13/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - loss: 4.8330e-06 - val_loss: 3.2341e-06
Epoch 14/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 2.5857e-06 - val_loss: 2.0959e-06
Epoch 15/20
```



```
Epoch 16/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 1.2478e-06 - val_loss: 1.3091e-06
Epoch 17/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - loss: 1.0846e-06 - val_loss: 1.1597e-06
Epoch 18/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - loss: 1.0085e-06 - val_loss: 1.0749e-06
Epoch 19/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - loss: 9.6341e-07 - val_loss: 1.0202e-06
Epoch 20/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - loss: 9.2886e-07 - val_loss: 9.7677e-07
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
```



Result : The program executed successfully.

EXPERIMENT-10

Aim : Write a program in deep learning for Bidirectional LSTM

Procedure : -

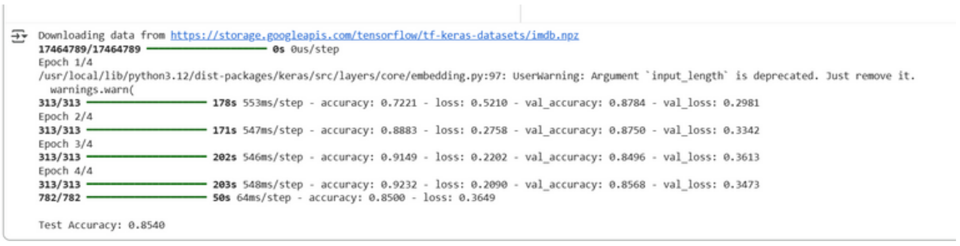
- import required libraries
- build and compile the model
- Train and predict

Input :

```
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Bidirectional, LSTM, Dense

# Step 1: Load IMDB dataset
vocab_size = 10000  # Only use top 10k words
maxlen = 200        # Cut texts after 200 words

(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=vocab_size)

# Step 2: Pad sequences to ensure equal length
x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)

# Step 3: Build the Bidirectional LSTM model
model = Sequential([
    Embedding(input_dim=vocab_size, output_dim=128, input_length=maxlen),
    Bidirectional(LSTM(64)),
    Dense(1, activation='sigmoid')
])

# Step 4: Compile the model
```

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


# Step 5: Train the model
history = model.fit(x_train, y_train, epochs=4, batch_size=64, validation_split=0.2)


# Step 6: Evaluate
loss, acc = model.evaluate(x_test, y_test)
print(f"\nTest Accuracy: {acc:.4f}")
```

Output :



```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Epoch 1/4
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/embedding.py:97: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
313/313 ━━━━━━━━━━━━━━━━━━━━ 178s 553ms/step - accuracy: 0.7221 - loss: 0.5210 - val_accuracy: 0.8784 - val_loss: 0.2981
Epoch 2/4
313/313 ━━━━━━━━━━━━━━━━━━━━ 171s 547ms/step - accuracy: 0.8883 - loss: 0.2758 - val_accuracy: 0.8750 - val_loss: 0.3342
Epoch 3/4
313/313 ━━━━━━━━━━━━━━━━━━━━ 202s 546ms/step - accuracy: 0.9149 - loss: 0.2202 - val_accuracy: 0.8496 - val_loss: 0.3613
Epoch 4/4
313/313 ━━━━━━━━━━━━━━━━━━━━ 203s 548ms/step - accuracy: 0.9232 - loss: 0.2090 - val_accuracy: 0.8568 - val_loss: 0.3473
782/782 ━━━━━━━━━━━━━━━━━━━━ 50s 64ms/step - accuracy: 0.8500 - loss: 0.3649

Test Accuracy: 0.8540
```

Result : The program executed successfully.

EXPERIMENT-11

Aim : **Write a program in deep learning for Data Augmentation using ImageDataGenerator**

Procedure : -

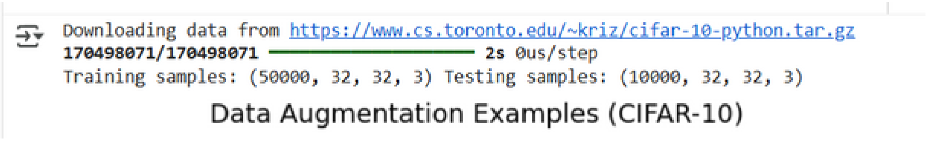
- import required libraries
- build and compile the model
- Train and predict

Input :

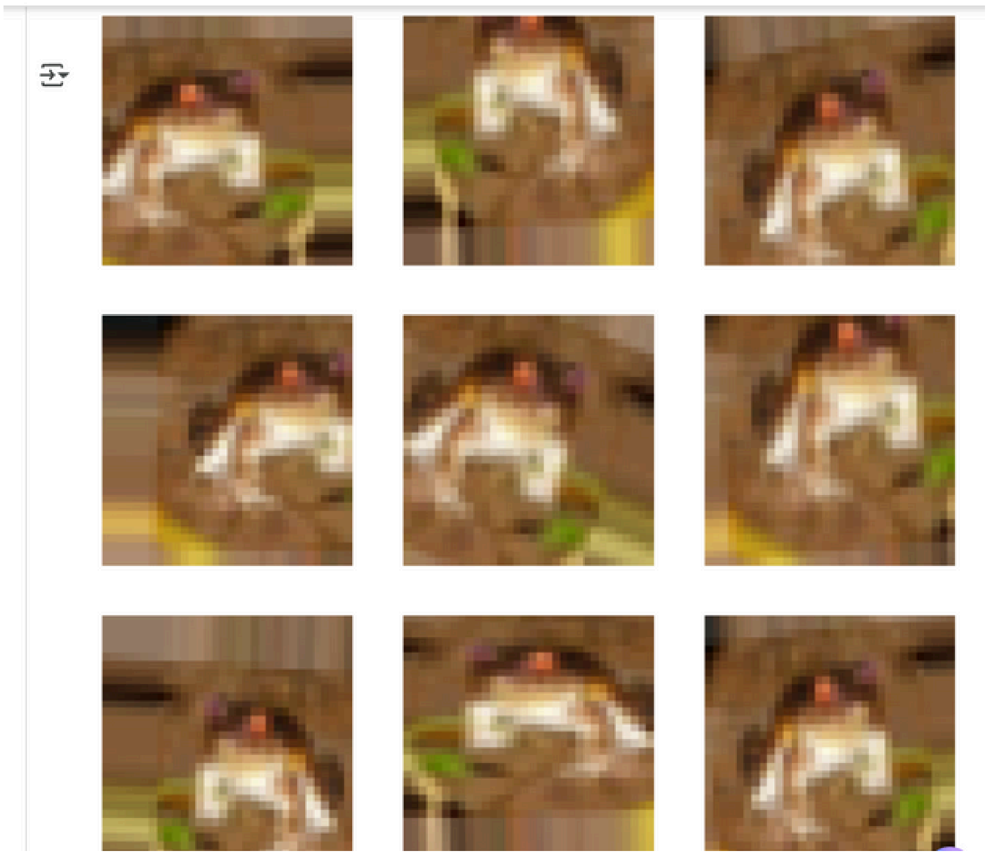
```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt

# 1. Load CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
print("Training samples:", x_train.shape, "Testing samples:", x_test.shape)

# 2. Normalize pixel values to [0,1]
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0

# 3. Create ImageDataGenerator for augmentation
datagen = ImageDataGenerator(
  rotation_range=20,    # rotate images
  width_shift_range=0.2,  # shift horizontally
  height_shift_range=0.2, # shift vertically
  horizontal_flip=True,  # flip horizontally
  zoom_range=0.2      # zoom
)

datagen.fit(x_train)
```

```
plt.figure(figsize=(8, 8))
for i, batch in enumerate(datagen.flow(x_train[:1], batch_size=1)):  # take first image and augment
    plt.subplot(3, 3, i + 1)
    plt.imshow(batch[0])
    plt.axis("off")
    if i == 8:  # show 9 samples
        breakyers.MaxPooling2D((2,2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation="relu"),
    tf.keras.layers.Dense(10, activation="softmax")
])

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

# 7. Train using augmented data
history = model.fit(
    datagen.flow(x_train, y_train, batch_size=64),
    epochs=3,
    validation_data=(x_test, y_test)
)

# 8. Evaluate
loss, acc = model.evaluate(x_test, y_test, verbose=2)
print(f"✅ Test Accuracy with Augmentation: {acc:.4f}")
```

Output :



```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
Training samples: (50000, 32, 32, 3) Testing samples: (10000, 32, 32, 3)
                  Data Augmentation Examples (CIFAR-10)
```



```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Se
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` :
  self._warn_if_super_not_called()
Epoch 1/3
782/782 ━━━━━━━━━━━━━━━━━━━━ 102s 125ms/step - accuracy: 0.2927 - loss: 1.9050 - val_accuracy: 0.4957 - val_loss: 1.4089
Epoch 2/3
782/782 ━━━━━━━━━━━━━━━━━━━━ 138s 121ms/step - accuracy: 0.4550 - loss: 1.5114 - val_accuracy: 0.5621 - val_loss: 1.2435
Epoch 3/3
782/782 ━━━━━━━━━━━━━━━━━━━━ 95s 122ms/step - accuracy: 0.4964 - loss: 1.4048 - val_accuracy: 0.5908 - val_loss: 1.1765
313/313 - 4s - 12ms/step - accuracy: 0.5908 - loss: 1.1765
✅ Test Accuracy with Augmentation: 0.5908
```

Result : The program executed successfully.

EXPERIMENT-12

Aim : **Write a program in deep learning for Data Augmentation using tf.image**

Procedure : -

- import required libraries
- build and compile the model
- Train and predict

Input :

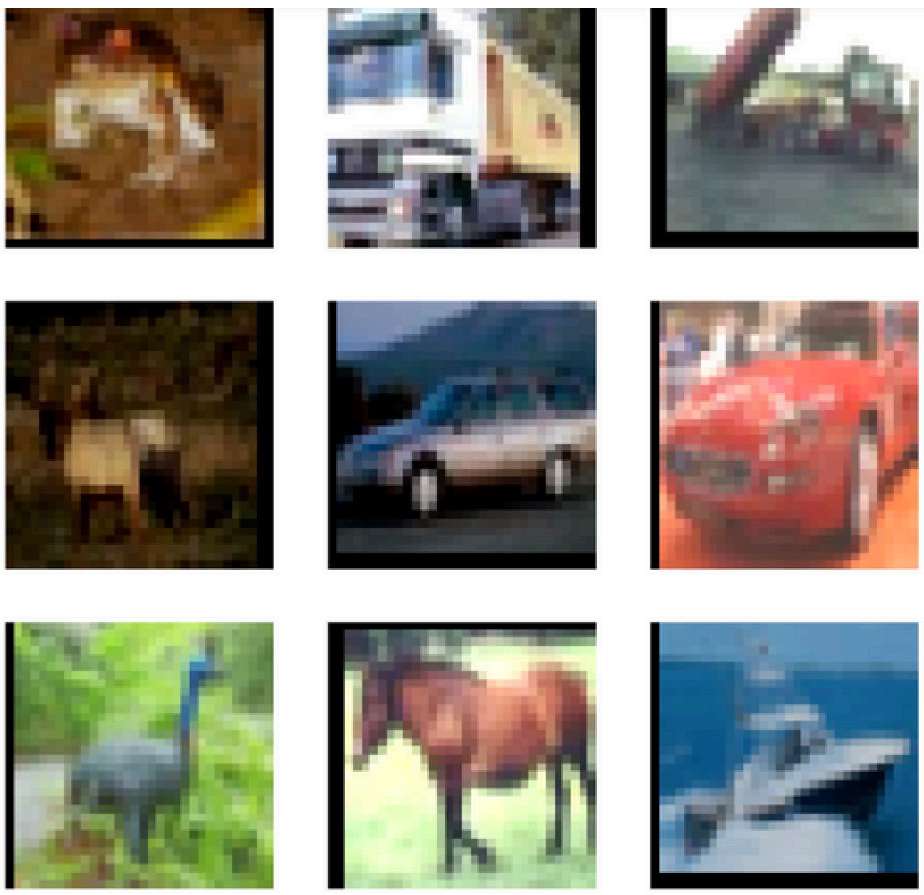
```
import tensorflow as tf
import matplotlib.pyplot as plt

# 1. Load CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
print("Training samples:", x_train.shape, "Testing samples:", x_test.shape)

# 2. Normalize pixel values to [0,1]
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0

# 3. Define custom augmentation function
def augment(image, label):
    image = tf.image.random_flip_left_right(image)       # random horizontal flip
    image = tf.image.random_brightness(image, max_delta=0.2)  # random brightness
    image = tf.image.random_contrast(image, 0.8, 1.2)    # random contrast
    image = tf.image.random_crop(tf.image.resize_with_crop_or_pad(image, 36, 36), [32, 32, 3])
    return image, label

# 4. Apply augmentation on training dataset
train_ds = tf.data.Dataset.from_tensor_slices((x_train, y_train))
train_ds = train_ds.map(augment).batch(64).prefetch(tf.data.AUTOTUNE)

test_ds = tf.data.Dataset.from_tensor_slices((x_test, y_test)).batch(64)
```

```python
plt.figure(figsize=(8, 8))
for images, labels in train_ds.take(1):  # take one batch
    for i in range(9):
        plt.subplot(3, 3, i + 1)
        plt.imshow(images[i])
        plt.axis("off")
plt.suptitle("Data Augmentation Examples using tf.image", fontsize=14)
plt.show()

# 6. Build CNN model
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation="relu", input_shape=(32, 32, 3)),
    tf.keras.layers.MaxPooling2D((2,2)),
    tf.keras.layers.Conv2D(64, (3,3), activation="relu"),
    tf.keras.layers.MaxPooling2D((2,2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation="relu"),
    tf.keras.layers.Dense(10, activation="softmax")
])

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

# 8. Train with augmented dataset
history = model.fit(train_ds, epochs=3, validation_data=test_ds)

# 9. Evaluate
loss, acc = model.evaluate(test_ds, verbose=2)
print(f"✅ Test Accuracy with Augmentation (tf.image): {acc:.4f}")
```

Output :

```
Training samples: (50000, 32, 32, 3) Testing samples: (10000, 32, 32, 3)
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-0.08640884..0.8577466].
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-0.006337166..1.1394966].
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [0.0..1.1766739].
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-0.18582678..0.7666038].
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-0.065410405..0.8265701].
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [0.0..1.1423194].
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [0.0..1.1996696].
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-0.03904426..0.8246447].
```

Data Augmentation Examples using tf.image



```
Epoch 1/3
782/782 ━━━━━━━━━━━━━━━━━━━━ 73s 90ms/step - accuracy: 0.3514 - loss: 1.7732 - val_accuracy: 0.5454 - val_loss: 1.2836
Epoch 2/3
782/782 ━━━━━━━━━━━━━━━━━━━━ 80s 88ms/step - accuracy: 0.5445 - loss: 1.2888 - val_accuracy: 0.6237 - val_loss: 1.0886
Epoch 3/3
782/782 ━━━━━━━━━━━━━━━━━━━━ 71s 91ms/step - accuracy: 0.6034 - loss: 1.1373 - val_accuracy: 0.6682 - val_loss: 0.9724
157/157 - 3s - 19ms/step - accuracy: 0.6682 - loss: 0.9724
✅ Test Accuracy with Augmentation (tf.image): 0.6682
```

Result : The program executed successfully.

EXPERIMENT-13

Aim : **Write a program in deep learning for Basic Neural Network using TensorFlow & Keras**

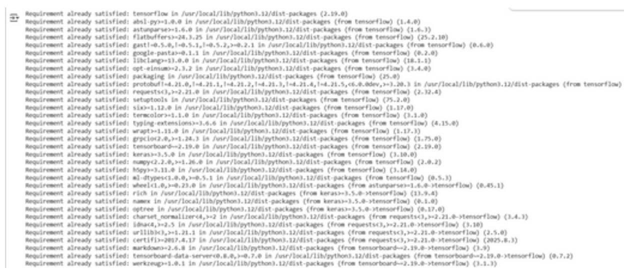
Procedure : -

- import required libraries
- build and compile the model
- Train and predict

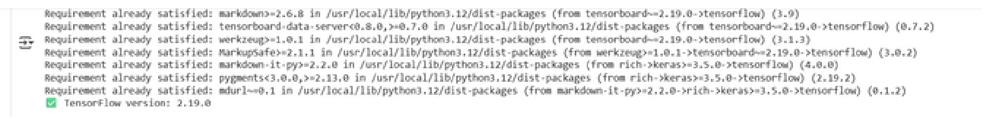
Input :

```
# 1. Install TensorFlow (skip if already installed in Google Colab)
!pip install tensorflow

# 2. Import libraries
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt

print("✅ TensorFlow version:", tf.__version__)

# 3. Load dataset (MNIST Handwritten digits)
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalize pixel values to [0,1]
x_train = x_train / 255.
```

Output :



```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.12/dist-packages (2.19.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (25.2.10)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-packages (from tensorflow) (25.0)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3 in /usr/local/lib/python3.12/dist-packages (from tensorflow)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (2.32.4)
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (from tensorflow) (75.2.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.17.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.1.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (4.15.0)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.17.3)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.75.0)
Requirement already satisfied: tensorboard~=2.19.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (2.19.0)
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.10.0)
Requirement already satisfied: numpy<2.2.0,>=1.26.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (2.0.2)
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.14.0)
Requirement already satisfied: ml-dtypes<1.0.0,>=0.5.1 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (0.5.3)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.12/dist-packages (from astunparse>=1.6.0->tensorflow) (0.45.1)
Requirement already satisfied: rich in /usr/local/lib/python3.12/dist-packages (from keras>=3.5.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.12/dist-packages (from keras>=3.5.0->tensorflow) (0.1.0)
Requirement already satisfied: optree in /usr/local/lib/python3.12/dist-packages (from keras>=3.5.0->tensorflow) (0.17.0)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.4.3)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0->tensorflow) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0->tensorflow) (2025.8.3)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.12/dist-packages (from tensorboard~=2.19.0->tensorflow) (3.9)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.12/dist-packages (from tensorboard~=2.19.0->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from tensorboard~=2.19.0->tensorflow) (3.1.3)
```



```
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.12/dist-packages (from tensorboard~=2.19.0->tensorflow) (3.9)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.12/dist-packages (from tensorboard~=2.19.0->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from tensorboard~=2.19.0->tensorflow) (3.1.3)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.12/dist-packages (from werkzeug>=1.0.1->tensorboard~=2.19.0->tensorflow) (3.0.2)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.12/dist-packages (from rich->keras>=3.5.0->tensorflow) (4.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.12/dist-packages (from rich->keras>=3.5.0->tensorflow) (2.19.2)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.12/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=3.5.0->tensorflow) (0.1.2)
✅ TensorFlow version: 2.19.0
```

Result : The program executed successfully.

EXPERIMENT-14

Aim : **Write a program in deep learning for Basic Neural Network using TensorFlow & Keras**

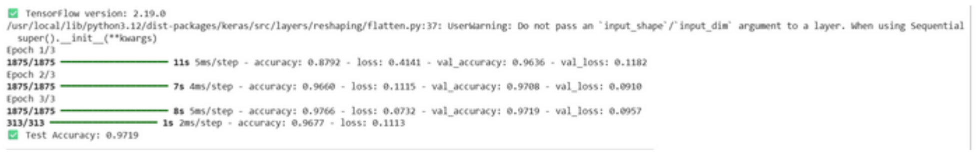
Procedure : -

- import required libraries
- build and compile the model
- Train and predict

Input :

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt

print("✅ TensorFlow version:", tf.__version__)

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

# Normalize data
x_train, x_test = x_train / 255.0, x_test / 255.0

# 3. Build the model
model = Sequential([
    Flatten(input_shape=(28, 28)),      # Flatten input image
    Dense(128, activation='relu'),      # Hidden layer 1
    Dense(64, activation='relu'),       # Hidden layer 2
    Dense(10, activation='softmax')     # Output layer (10 classes)
])
model.compile(optimizer='adam',
 loss='sparse_categorical_crossentropy',
 metrics=['accuracy'])
history = model.fit(x_train, y_train, epochs=3, validation_data=(x_test, y_test))
loss, acc = model.evaluate(x_test, y_test)
print(f"✅ Test Accuracy: {acc:.4f}")
```

Output :



```
✅ TensorFlow version: 2.19.0
/usr/local/lib/python3.12/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
  super().__init__(**kwargs)
Epoch 1/3
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.8792 - loss: 0.4141 - val_accuracy: 0.9636 - val_loss: 0.1182
Epoch 2/3
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9660 - loss: 0.1115 - val_accuracy: 0.9708 - val_loss: 0.0910
Epoch 3/3
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.9766 - loss: 0.0732 - val_accuracy: 0.9719 - val_loss: 0.0957
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9677 - loss: 0.1113
✅ Test Accuracy: 0.9719
```

Result : The program executed successfully.

EXPERIMENT-15

Aim : Write a program in deep learning for Neural Networks using Keras (MNIST dataset)

Procedure : -

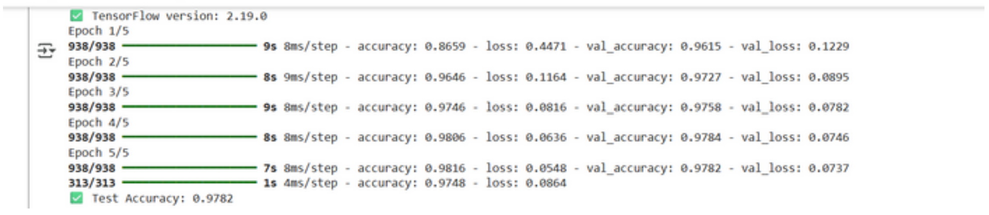
- import required libraries
- build and compile the model
- Train and predict

Input :

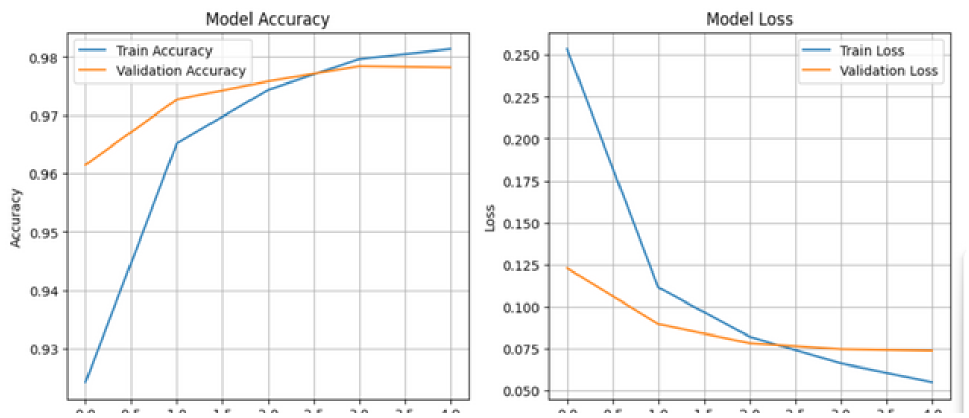
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt

print("✅ TensorFlow version:", tf.__version__)

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

# Normalize (0-255 -> 0-1)
x_train, x_test = x_train / 255.0, x_test / 255.0

# 3. Build the Neural Network model
model = Sequential([
    Flatten(input_shape=(28, 28)),     # Flatten image
    Dense(256, activation='relu'),     # Hidden layer 1
    Dropout(0.2),                      # Dropout to prevent overfitting
    Dense(128, activation='relu'),     # Hidden layer 2
    Dense(10, activation='softmax')    # Output layer
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
history = model.fit(x_train, y_train, epochs=5, batch_size=64,
                    validation_data=(x_test, y_test))

# 6. Evaluate the model
loss, acc = model.evaluate(x_test, y_test)
print(f"✅ Test Accuracy: {acc:.4f}")

# 7. Plot accuracy and loss curves
plt.figure(figsize=(12, 5))

# Accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label="Train Accuracy")
plt.plot(history.history['val_accuracy'], label="Validation Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.title("Model Accuracy")
plt.legend()
plt.grid(True)

# Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label="Train Loss")
plt.plot(history.history['val_loss'], label="Validation Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Model Loss")
plt.legend()
plt.grid(True)

plt.show()
```

Output :



```
✅ TensorFlow version: 2.19.0
Epoch 1/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 9s 8ms/step - accuracy: 0.8659 - loss: 0.4471 - val_accuracy: 0.9615 - val_loss: 0.1229
Epoch 2/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 8s 9ms/step - accuracy: 0.9646 - loss: 0.1164 - val_accuracy: 0.9727 - val_loss: 0.0895
Epoch 3/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 9s 8ms/step - accuracy: 0.9746 - loss: 0.0816 - val_accuracy: 0.9758 - val_loss: 0.0782
Epoch 4/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 8s 8ms/step - accuracy: 0.9806 - loss: 0.0636 - val_accuracy: 0.9784 - val_loss: 0.0746
Epoch 5/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 7s 8ms/step - accuracy: 0.9816 - loss: 0.0548 - val_accuracy: 0.9782 - val_loss: 0.0737
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9748 - loss: 0.0864
✅ Test Accuracy: 0.9782
```



Result : The program executed successfully.