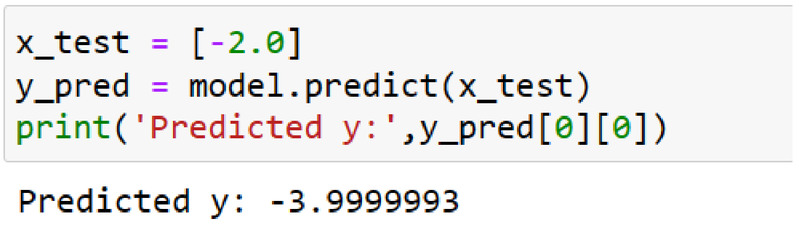
PROGRAM 1

Write a program to Build a simple artificial Neural Networks with 1 layer, with 1 neuron, and the input shape equal to 1, feed some data, use the equation y=5x-3, so where x = -2, y=-4 and train the network.

Program

```
import tensorflow as tf

# Define the input data and labels
x_train = [-2.0]
y_train = [-4.0]

# Build the neural network
model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=1, input_shape=(1,))
])

# Compile the model
model.compile(optimizer='sgd', loss='mean_squared_error')

# Train the model
model.fit(x_train, y_train, epochs=1000, verbose=0)

# Evaluate the trained model
x_test = [-2.0]
y_pred = model.predict(x_test)
print("Predicted y:", y_pred[0][0])
```

Output



```
x_test = [-2.0]
y_pred = model.predict(x_test)
print('Predicted y:',y_pred[0][0])
```

```
Predicted y: -3.9999993
```

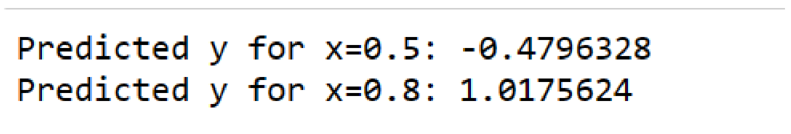
PROGRAM 2

Using Tensorflow Build a network with a single hidden layer and at least 300,000 trainable parameters.

Program

```
import tensorflow as tf
import numpy as np

# Generate synthetic data
num_samples = 1000
x_train = np.random.rand(num_samples, 1)
y_train = 5 * x_train - 3 + np.random.randn(num_samples, 1) * 0.1

# Build the neural network
model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=100, activation='relu', input_shape=(1,)),
    tf.keras.layers.Dense(units=100, activation='relu'),
    tf.keras.layers.Dense(units=1)
])

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

# Train the model
model.fit(x_train, y_train, epochs=100, verbose=0)

# Generate test data
x_test = np.array([[0.5], [0.8]])
y_pred = model.predict(x_test)

print("Predicted y for x=0.5:", y_pred[0][0])
print("Predicted y for x=0.8:", y_pred[1][0])
```

Output



```
Predicted y for x=0.5: -0.4796328
Predicted y for x=0.8: 1.0175624
```

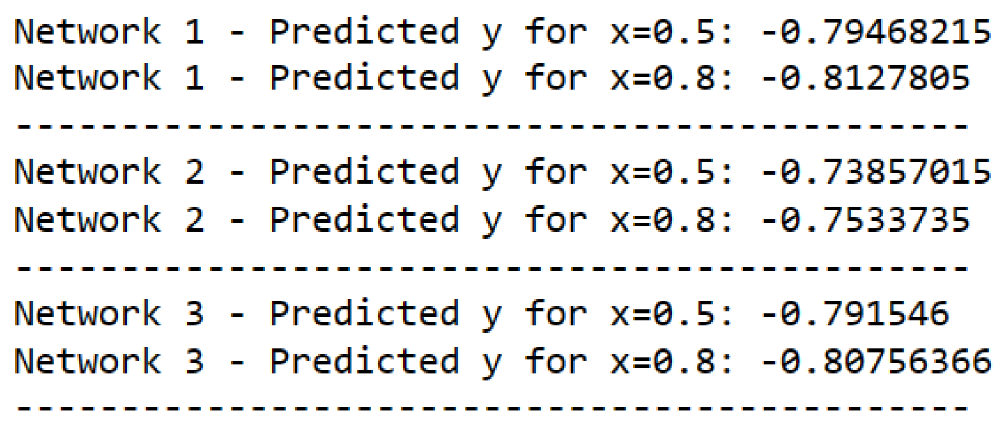
PROGRAM 3

Write a program Using Tensorflow build 3 networks, each with at least 10 hidden layers.

Program

```
import tensorflow as tf
import numpy as np

# Generate synthetic data
num_samples = 1000
x_train = np.random.rand(num_samples, 1)
y_train = 5 * x_train - 3 + np.random.randn(num_samples, 1) * 0.1

# Function to build a neural network with a given number of hidden layers
def build_network(num_hidden_layers):
    model = tf.keras.Sequential()
    model.add(tf.keras.layers.Input(shape=(1,)))
    for _ in range(num_hidden_layers):
        model.add(tf.keras.layers.Dense(units=100, activation='relu'))
    model.add(tf.keras.layers.Dense(units=1))
    return model

# Build three networks with different numbers of hidden layers
networks = [build_network(num_hidden_layers) for num_hidden_layers in [10, 15, 20]]

# Compile and train the networks
for i, model in enumerate(networks):
    model.compile(optimizer='adam', loss='mean_squared_error')
    model.fit(x_train, y_train, epochs=100, verbose=0)
    y_pred = model.predict(x_train)
    print(f"Network {i+1} - Predicted y for x=0.5:", y_pred[0][0])
    print(f"Network {i+1} - Predicted y for x=0.8:", y_pred[1][0])
    print("-" * 40)
```

Output



```
Network 1 - Predicted y for x=0.5: -0.79468215
Network 1 - Predicted y for x=0.8: -0.8127805
---------------------------------------------
Network 2 - Predicted y for x=0.5: -0.73857015
Network 2 - Predicted y for x=0.8: -0.7533735
---------------------------------------------
Network 3 - Predicted y for x=0.5: -0.791546
Network 3 - Predicted y for x=0.8: -0.80756366
---------------------------------------------
```

PROGRAM 4

Write a program to Build a network with at least 3 hidden layers that achieves better than 92% accuracy on validation and test data. You may need to train for more than 10 epochs to achieve this result.

Program

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist

# Load and preprocess the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

# Build the neural network
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(units=512, activation='relu'),
    tf.keras.layers.Dense(units=256, activation='relu'),
    tf.keras.layers.Dense(units=128, activation='relu'),
    tf.keras.layers.Dense(units=10, activation='softmax')
])

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

# Train the model
model.fit(x_train, y_train, epochs=15, validation_split=0.1, verbose=2)

# Evaluate the model on test data
test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=2)
print(f"Test accuracy: {test_accuracy * 100:.2f}%")
```

Output

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 3s 0us/step
11501568/11490434 [==============================] - 3s 0us/step
Epoch 1/15
1688/1688 - 11s - loss: 0.2042 - accuracy: 0.9388 - val_loss: 0.0972 - val_accuracy: 0.9707
Epoch 2/15
1688/1688 - 8s - loss: 0.0896 - accuracy: 0.9728 - val_loss: 0.0693 - val_accuracy: 0.9802
Epoch 3/15
1688/1688 - 7s - loss: 0.0634 - accuracy: 0.9805 - val_loss: 0.0668 - val_accuracy: 0.9800
Epoch 4/15
1688/1688 - 7s - loss: 0.0510 - accuracy: 0.9841 - val_loss: 0.0855 - val_accuracy: 0.9762
Epoch 5/15
1688/1688 - 7s - loss: 0.0402 - accuracy: 0.9881 - val_loss: 0.0754 - val_accuracy: 0.9793
Epoch 6/15
1688/1688 - 10s - loss: 0.0352 - accuracy: 0.9894 - val_loss: 0.0722 - val_accuracy: 0.9807
Epoch 7/15
1688/1688 - 8s - loss: 0.0284 - accuracy: 0.9911 - val_loss: 0.0912 - val_accuracy: 0.9778
Epoch 8/15
1688/1688 - 8s - loss: 0.0259 - accuracy: 0.9920 - val_loss: 0.0930 - val_accuracy: 0.9777
Epoch 9/15
1688/1688 - 8s - loss: 0.0233 - accuracy: 0.9928 - val_loss: 0.1014 - val_accuracy: 0.9768
Epoch 10/15
1688/1688 - 8s - loss: 0.0219 - accuracy: 0.9936 - val_loss: 0.1005 - val_accuracy: 0.9788
Epoch 11/15
1688/1688 - 8s - loss: 0.0175 - accuracy: 0.9946 - val_loss: 0.0874 - val_accuracy: 0.9833
Epoch 12/15
1688/1688 - 8s - loss: 0.0202 - accuracy: 0.9944 - val_loss: 0.0848 - val_accuracy: 0.9815
Epoch 13/15
1688/1688 - 10s - loss: 0.0155 - accuracy: 0.9957 - val_loss: 0.0930 - val_accuracy: 0.9830
Epoch 14/15
1688/1688 - 9s - loss: 0.0140 - accuracy: 0.9957 - val_loss: 0.1143 - val_accuracy: 0.9813
Epoch 15/15
1688/1688 - 8s - loss: 0.0168 - accuracy: 0.9953 - val_loss: 0.0812 - val_accuracy: 0.9823
313/313 - 1s - loss: 0.1028 - accuracy: 0.9802
Test accuracy: 98.02%
```

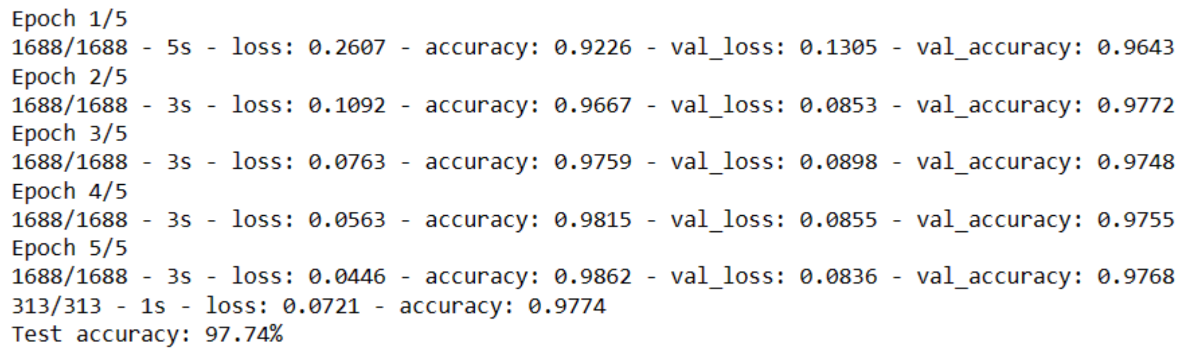
PROGRAM 5

Write a program Build a network for classification using the built in MNIST dataset.

Program

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

# Load and preprocess the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

# Build the neural network
model = Sequential([
    Flatten(input_shape=(28, 28)),  # Flatten the 28x28 images to a 1D array
    Dense(128, activation='relu'),   # Fully connected layer with 128 units and ReLU activation
    Dense(64, activation='relu'),    # Fully connected layer with 64 units and ReLU activation
    Dense(10, activation='softmax')  # Output layer with 10 units for 10 classes and softmax
activation
])

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

# Train the model
model.fit(x_train, y_train, epochs=5, validation_split=0.1, verbose=2)

# Evaluate the model on test data
test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=2)
print(f"Test accuracy: {test_accuracy * 100:.2f}%")
```

<u>Output</u>



```
Epoch 1/5
1688/1688 - 5s - loss: 0.2607 - accuracy: 0.9226 - val_loss: 0.1305 - val_accuracy: 0.9643
Epoch 2/5
1688/1688 - 3s - loss: 0.1092 - accuracy: 0.9667 - val_loss: 0.0853 - val_accuracy: 0.9772
Epoch 3/5
1688/1688 - 3s - loss: 0.0763 - accuracy: 0.9759 - val_loss: 0.0898 - val_accuracy: 0.9748
Epoch 4/5
1688/1688 - 3s - loss: 0.0563 - accuracy: 0.9815 - val_loss: 0.0855 - val_accuracy: 0.9755
Epoch 5/5
1688/1688 - 3s - loss: 0.0446 - accuracy: 0.9862 - val_loss: 0.0836 - val_accuracy: 0.9768
313/313 - 1s - loss: 0.0721 - accuracy: 0.9774
Test accuracy: 97.74%
```

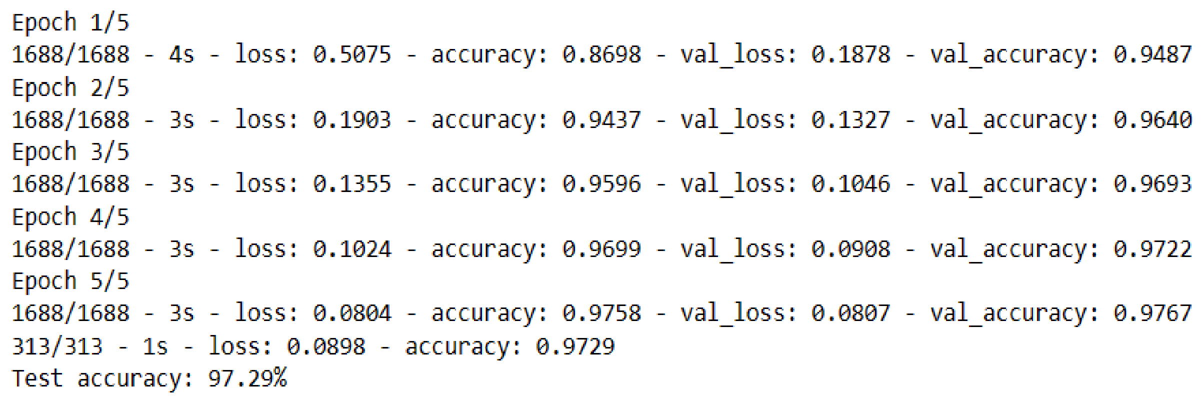
PROGRAM 6

Write a program Build a network for classification using the built in MNIST dataset and Use the sigmoid activation function.

Program

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

# Load and preprocess the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

# Build the neural network
model = Sequential([
    Flatten(input_shape=(28, 28)),  # Flatten the 28x28 images to a 1D array
    Dense(128, activation='sigmoid'),   # Fully connected layer with 128 units and sigmoid
activation
    Dense(64, activation='sigmoid'),    # Fully connected layer with 64 units and sigmoid
activation
    Dense(10, activation='softmax')  # Output layer with 10 units for 10 classes and softmax
activation
])

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

# Train the model
model.fit(x_train, y_train, epochs=5, validation_split=0.1, verbose=2)

# Evaluate the model on test data
test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=2)
print(f"Test accuracy: {test_accuracy * 100:.2f}%")
```

Output



```
Epoch 1/5
1688/1688 - 4s - loss: 0.5075 - accuracy: 0.8698 - val_loss: 0.1878 - val_accuracy: 0.9487
Epoch 2/5
1688/1688 - 3s - loss: 0.1903 - accuracy: 0.9437 - val_loss: 0.1327 - val_accuracy: 0.9640
Epoch 3/5
1688/1688 - 3s - loss: 0.1355 - accuracy: 0.9596 - val_loss: 0.1046 - val_accuracy: 0.9693
Epoch 4/5
1688/1688 - 3s - loss: 0.1024 - accuracy: 0.9699 - val_loss: 0.0908 - val_accuracy: 0.9722
Epoch 5/5
1688/1688 - 3s - loss: 0.0804 - accuracy: 0.9758 - val_loss: 0.0807 - val_accuracy: 0.9767
313/313 - 1s - loss: 0.0898 - accuracy: 0.9729
Test accuracy: 97.29%
```

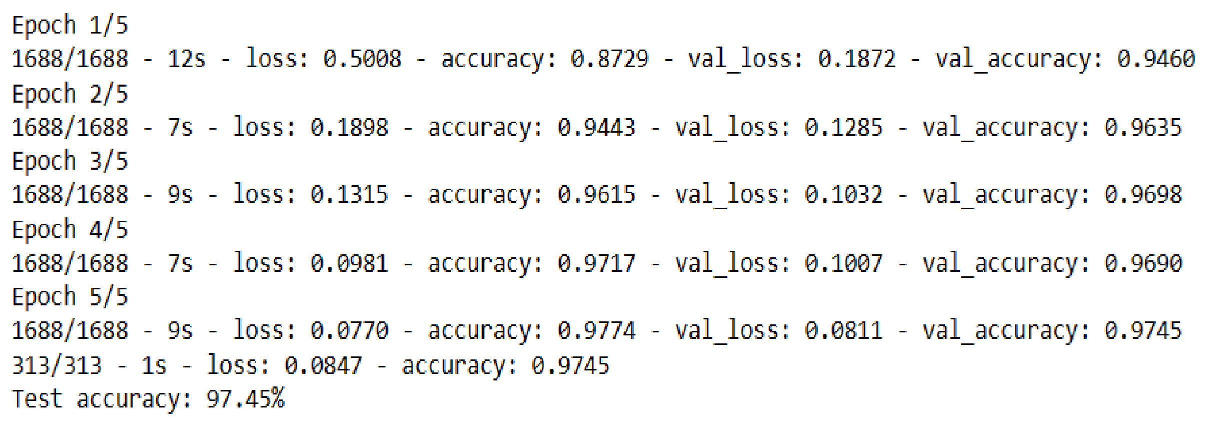
PROGRAM 7

Write a program Build a network for classification using the built in MNIST dataset and Use the sigmoid activation function Use the categorical cross entropy loss function.

Program

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

# Load and preprocess the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

# Build the neural network
model = Sequential([
    Flatten(input_shape=(28, 28)),  # Flatten the 28x28 images to a 1D array
    Dense(128, activation='sigmoid'),   # Fully connected layer with 128 units and sigmoid
activation
    Dense(64, activation='sigmoid'),    # Fully connected layer with 64 units and sigmoid
activation
    Dense(10, activation='sigmoid')  # Output layer with 10 units for 10 classes and sigmoid
activation
])

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

# Train the model
model.fit(x_train, tf.keras.utils.to_categorical(y_train, num_classes=10), epochs=5,
validation_split=0.1, verbose=2)

# Evaluate the model on test data
test_loss, test_accuracy = model.evaluate(x_test, tf.keras.utils.to_categorical(y_test,
num_classes=10), verbose=2)
print(f"Test accuracy: {test_accuracy * 100:.2f}%")
```

Output



```
Epoch 1/5
1688/1688 - 12s - loss: 0.5008 - accuracy: 0.8729 - val_loss: 0.1872 - val_accuracy: 0.9460
Epoch 2/5
1688/1688 - 7s - loss: 0.1898 - accuracy: 0.9443 - val_loss: 0.1285 - val_accuracy: 0.9635
Epoch 3/5
1688/1688 - 9s - loss: 0.1315 - accuracy: 0.9615 - val_loss: 0.1032 - val_accuracy: 0.9698
Epoch 4/5
1688/1688 - 7s - loss: 0.0981 - accuracy: 0.9717 - val_loss: 0.1007 - val_accuracy: 0.9690
Epoch 5/5
1688/1688 - 9s - loss: 0.0770 - accuracy: 0.9774 - val_loss: 0.0811 - val_accuracy: 0.9745
313/313 - 1s - loss: 0.0847 - accuracy: 0.9745
Test accuracy: 97.45%
```

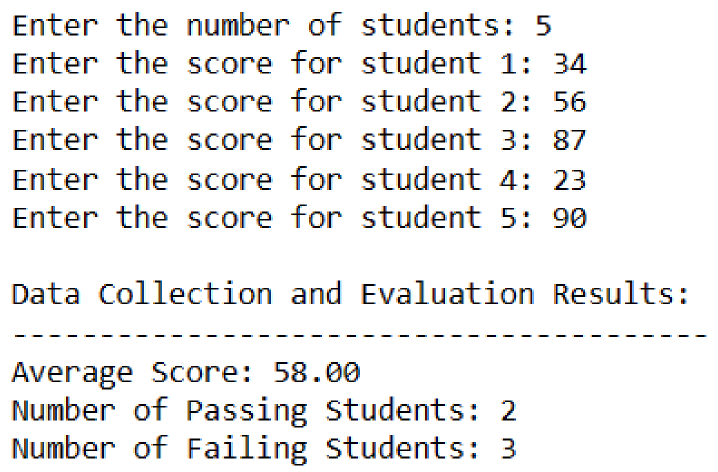
PROGRAM 8

Write a program for Working Data Collection, Evaluation.

Program

```
# Collect data
num_students = int(input("Enter the number of students: "))
scores = []

for i in range(num_students):
    score = float(input(f"Enter the score for student {i+1}: "))
    scores.append(score)

# Evaluate data
passing_threshold = 60
average_score = sum(scores) / len(scores)
passing_students = sum(score >= passing_threshold for score in scores)
failing_students = num_students - passing_students

# Output results
print("\nData Collection and Evaluation Results:")
print("-" * 40)
print(f"Average Score: {average_score:.2f}")
print(f"Number of Passing Students: {passing_students}")
print(f"Number of Failing Students: {failing_students}")
```

Output



```
Enter the number of students: 5
Enter the score for student 1: 34
Enter the score for student 2: 56
Enter the score for student 3: 87
Enter the score for student 4: 23
Enter the score for student 5: 90

Data Collection and Evaluation Results:
----------------------------------------
Average Score: 58.00
Number of Passing Students: 2
Number of Failing Students: 3
```

PROGRAM 9

Write a program Using Conduct an experiment on Object detection using Convolution Neural Network.

Program

```
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Load and preprocess CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

# Build the CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])

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

# Train the model
model.fit(x_train, y_train, epochs=5, validation_split=0.1, verbose=2)

# Evaluate the model on test data
test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=2)
print(f"Test accuracy: {test_accuracy * 100:.2f}%")
```

Output

Output 1 (Fit the Model)

```
Epoch 1/5
1407/1407 - 142s - loss: 1.2573 - accuracy: 0.5514 - val_loss: 1.1477 - val_accuracy: 0.5956
Epoch 2/5
1407/1407 - 141s - loss: 1.1108 - accuracy: 0.6068 - val_loss: 1.0542 - val_accuracy: 0.6254
Epoch 3/5
1407/1407 - 133s - loss: 1.0330 - accuracy: 0.6337 - val_loss: 1.0222 - val_accuracy: 0.6402
Epoch 4/5
1407/1407 - 126s - loss: 0.9574 - accuracy: 0.6597 - val_loss: 1.0168 - val_accuracy: 0.6468
Epoch 5/5
1407/1407 - 126s - loss: 0.9033 - accuracy: 0.6836 - val_loss: 0.9449 - val_accuracy: 0.6696

<keras.callbacks.History at 0x1b7bd3e6a30>
```

Output 2 (Test Accuracy)

```
test_loss, test_accuracy = model.evaluate(x_test,y_test,verbose=2)
print(f'Test Accuracy: {test_accuracy*100:.2f}%')
```

```
313/313 - 3s - loss: 0.9813 - accuracy: 0.6594
Test Accuracy: 65.94%
```

PROGRAM 10

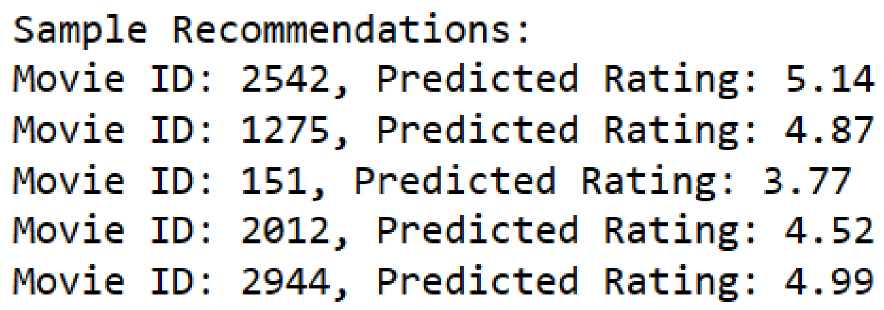
Write a program to Build a Recommendation system using Deep Learning techniques.

Program

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, Flatten, Dense
from tensorflow.keras.optimizers import Adam

# Load the MovieLens dataset
url = "https://raw.githubusercontent.com/WayneDW/su2017-
datascience/master/lecture_11/data/movielens/ratings.csv"
data = pd.read_csv(url)

# Create user and item dictionaries
user_ids = data.userId.unique()
item_ids = data.movieId.unique()
user2idx = {u: i for i, u in enumerate(user_ids)}
item2idx = {m: i for i, m in enumerate(item_ids)}

num_users = len(user_ids)
num_items = len(item_ids)

# Create training and validation sets
train_data, val_data = train_test_split(data, test_size=0.2, random_state=42)

# Build a simple collaborative filtering model
user_input = Input(shape=(1,))
item_input = Input(shape=(1,))
user_embedding = Embedding(num_users, 50)(user_input)
item_embedding = Embedding(num_items, 50)(item_input)
user_vecs = Flatten()(user_embedding)
item_vecs = Flatten()(item_embedding)
dot_product = tf.keras.layers.Dot(axes=1)([user_vecs, item_vecs])
model = Model(inputs=[user_input, item_input], outputs=dot_product)

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error')

# Train the model
```

```
history = model.fit(
    [train_data.userId.map(user2idx), train_data.movieId.map(item2idx)],
    train_data.rating,
    batch_size=64,
    epochs=5,
    verbose=2,
    validation_data=([val_data.userId.map(user2idx), val_data.movieId.map(item2idx)],
val_data.rating)
)

# Sample recommendation
user_id = 1
user_input = np.array([user2idx[user_id]])
item_ids = data[data.userId == user_id].sample(n=5)['movieId'].values
item_inputs = np.array([item2idx[item] for item in item_ids])
user_input = np.array([user2idx[user_id]] * len(item_ids))
predicted_ratings = model.predict([user_input, item_inputs])

print("Sample Recommendations:")
for item_id, rating in zip(item_ids, predicted_ratings):
    print(f"Movie ID: {item_id}, Predicted Rating: {rating[0]:.2f}")
```

Output



```
Sample Recommendations:
Movie ID: 2542, Predicted Rating: 5.14
Movie ID: 1275, Predicted Rating: 4.87
Movie ID: 151, Predicted Rating: 3.77
Movie ID: 2012, Predicted Rating: 4.52
Movie ID: 2944, Predicted Rating: 4.99
```

PROGRAM 11

Write a program Using Working on Deep Learning Data Structures.

Program

```
import numpy as np
import tensorflow as tf

# Creating tensors and arrays
tensor = tf.constant([[1, 2, 3], [4, 5, 6]])
array = np.array([[7, 8, 9], [10, 11, 12]])

# Performing operations
addition = tensor + array
multiplication = tensor * array
matrix_product = tf.matmul(tensor, array.T)

# Converting between tensors and arrays
tensor_to_array = tensor.numpy()
array_to_tensor = tf.convert_to_tensor(array)

# Printing results
print("Original Tensor:")
print(tensor)
print("\nOriginal Array:")
print(array)
print("\nElement-wise Addition:")
print(addition)
print("\nElement-wise Multiplication:")
print(multiplication)
print("\nMatrix Product:")
print(matrix_product)
print("\nTensor converted to Array:")
print(tensor_to_array)
print("\nArray converted to Tensor:")
print(array_to_tensor)
```

Output

```
Original Tensor:
tf.Tensor(
[[1 2 3]
 [4 5 6]], shape=(2, 3), dtype=int32)

Original Array:
[[ 7  8  9]
 [10 11 12]]

Element wise addition:
tf.Tensor(
[[ 8 10 12]
 [14 16 18]], shape=(2, 3), dtype=int32)

Element wise multiplication:
tf.Tensor(
[[ 7 16 27]
 [40 55 72]], shape=(2, 3), dtype=int32)

Matrix Product:
tf.Tensor(
[[ 50  68]
 [122 167]], shape=(2, 2), dtype=int32)

Tensor Converted to Array:
[[1 2 3]
 [4 5 6]]

Array Converted to Tensor:
tf.Tensor(
[[ 7  8  9]
 [10 11 12]], shape=(2, 3), dtype=int32)
```

PROGRAM 12

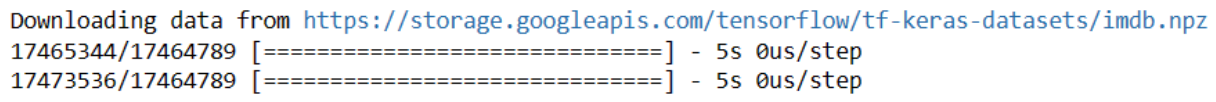
Write a program Use Recurrent Neural network to Perform Sentiment Analysis.

Program

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

# Load the IMDb dataset
max_words = 10000
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_words)

# Preprocess the data
max_review_length = 500
x_train = pad_sequences(x_train, maxlen=max_review_length)
x_test = pad_sequences(x_test, maxlen=max_review_length)

# Build the RNN model
model = Sequential([
    Embedding(max_words, 32, input_length=max_review_length),
    SimpleRNN(64),
    Dense(1, activation='sigmoid')
])

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

# Train the model
model.fit(x_train, y_train, epochs=3, batch_size=64, validation_split=0.2, verbose=2)

# Evaluate the model on test data
test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=2)
print(f"Test accuracy: {test_accuracy * 100:.2f}%")
```

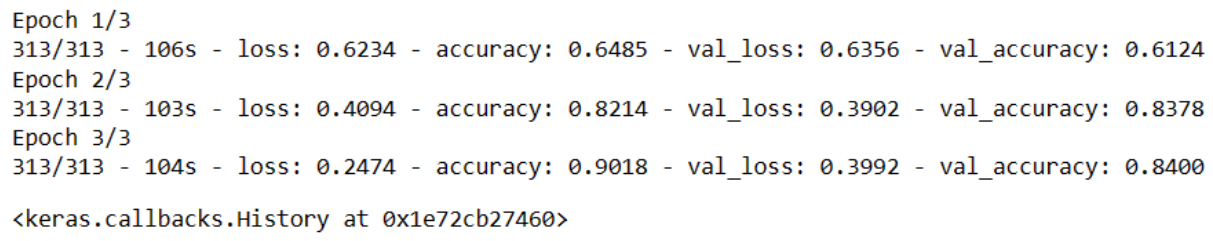
<u>Output</u>

Output 1 (Downloading dataset)



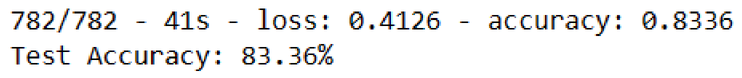
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17465344/17464789 [==============================] - 5s 0us/step
17473536/17464789 [==============================] - 5s 0us/step
```

Output 2 (Fit the Model)



```
Epoch 1/3
313/313 - 106s - loss: 0.6234 - accuracy: 0.6485 - val_loss: 0.6356 - val_accuracy: 0.6124
Epoch 2/3
313/313 - 103s - loss: 0.4094 - accuracy: 0.8214 - val_loss: 0.3902 - val_accuracy: 0.8378
Epoch 3/3
313/313 - 104s - loss: 0.2474 - accuracy: 0.9018 - val_loss: 0.3992 - val_accuracy: 0.8400

<keras.callbacks.History at 0x1e72cb27460>
```

Output 3 (Test Accuracy)



```
782/782 - 41s - loss: 0.4126 - accuracy: 0.8336
Test Accuracy: 83.36%
```

PROGRAM 13

Write a program Using Generative Adversarial networks perform Image generation.

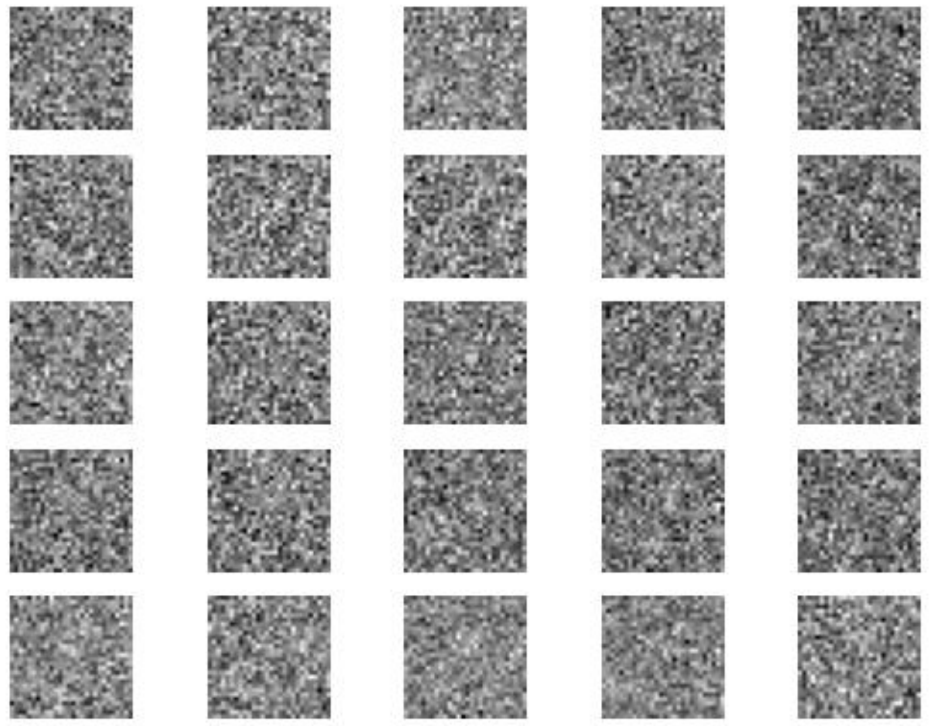
Program

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Reshape, Flatten
from tensorflow.keras.optimizers import Adam

# Define the Generator and Discriminator models
def build_generator(latent_dim):
    model = Sequential([
        Dense(128, input_dim=latent_dim, activation='relu'),
        Dense(784, activation='sigmoid'),
        Reshape((28, 28))
    ])
    return model

def build_discriminator(img_shape):
    model = Sequential([
        Flatten(input_shape=img_shape),
        Dense(128, activation='relu'),
        Dense(1, activation='sigmoid')
    ])
    return model

# Define GAN model
def build_gan(generator, discriminator):
    discriminator.trainable = False
    model = Sequential([generator, discriminator])
    return model

# Load the MNIST dataset
(x_train, _), (_, _) = tf.keras.datasets.mnist.load_data()
x_train = x_train / 255.0

# Build models
latent_dim = 100
img_shape = (28, 28)
generator = build_generator(latent_dim)
```

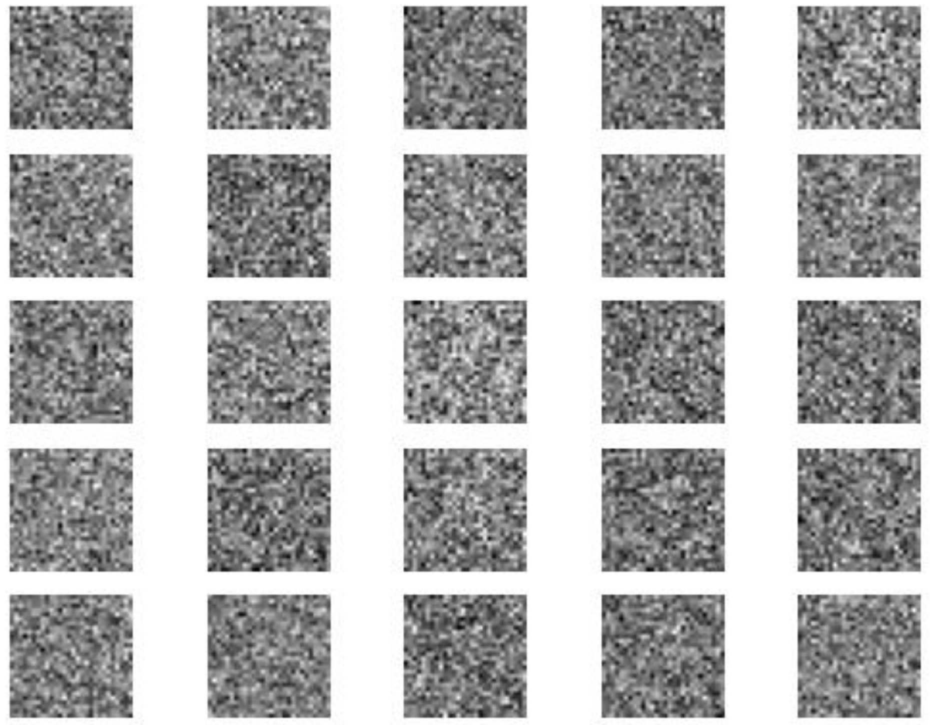
```
discriminator = build_discriminator(img_shape)
gan = build_gan(generator, discriminator)

# Compile models
discriminator.compile(optimizer=Adam(learning_rate=0.0002), loss='binary_crossentropy',
metrics=['accuracy'])
gan.compile(optimizer=Adam(learning_rate=0.0002), loss='binary_crossentropy')

# Training the GAN
batch_size = 64
epochs = 30000
sample_interval = 1000

for epoch in range(epochs):
    # Generate random noise
    noise = np.random.normal(0, 1, (batch_size, latent_dim))
    generated_images = generator.predict(noise)

    # Select a random batch of real images
    idx = np.random.randint(0, x_train.shape[0], batch_size)
    real_images = x_train[idx]

    # Train the discriminator
    d_loss_real = discriminator.train_on_batch(real_images, np.ones((batch_size, 1)))
    d_loss_fake = discriminator.train_on_batch(generated_images, np.zeros((batch_size, 1)))
    d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

    # Train the generator
    noise = np.random.normal(0, 1, (batch_size, latent_dim))
    g_loss = gan.train_on_batch(noise, np.ones((batch_size, 1)))

    # Print progress
    if epoch % sample_interval == 0:
        print(f"Epoch {epoch}, D Loss: {d_loss[0]}, G Loss: {g_loss}")

        # Save generated images
        r, c = 5, 5
        generated_images = 0.5 * generated_images + 0.5  # Rescale images to [0, 1]
        fig, axs = plt.subplots(r, c)
        cnt = 0
        for i in range(r):
            for j in range(c):
                axs[i, j].imshow(generated_images[cnt], cmap='gray')
                axs[i, j].axis('off')
                cnt += 1
        plt.show()
```

Output

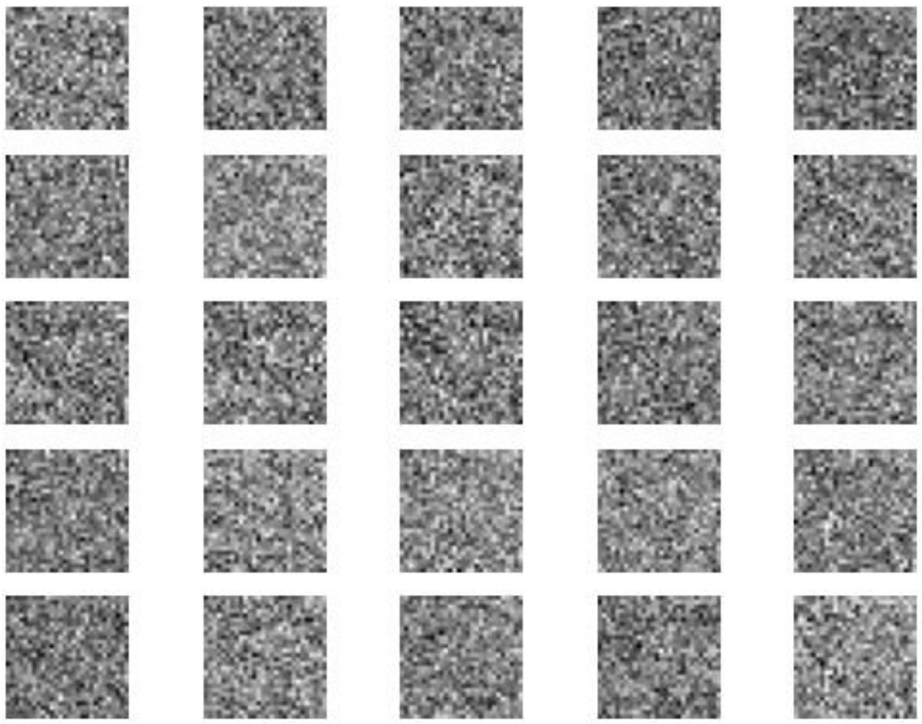
```
Epoch 0, D Loss: 0.8200506567955017, G Loss: 0.50406414270401
```



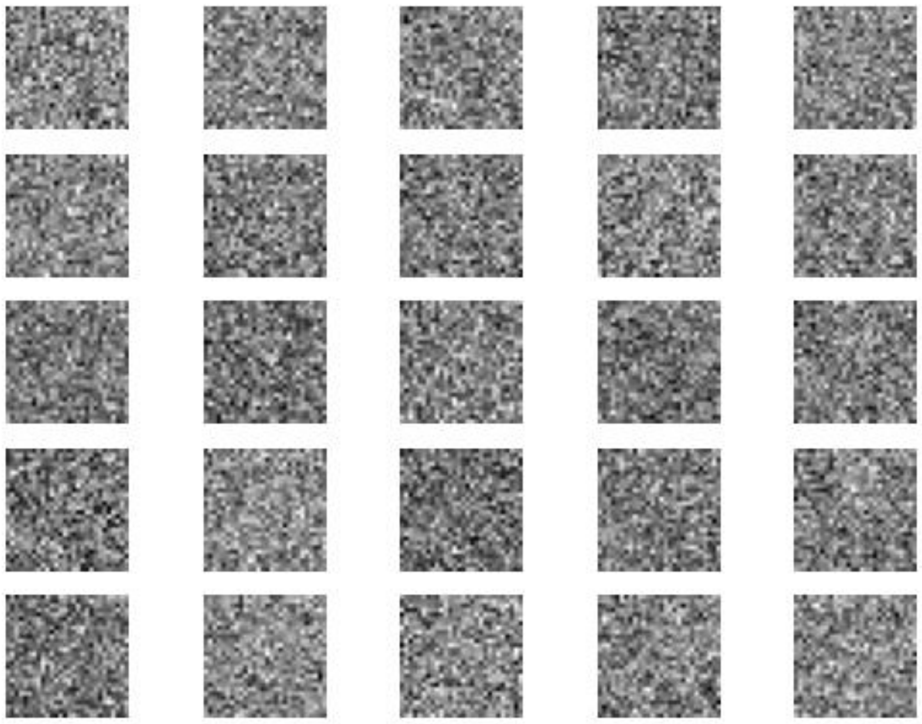
```
Epoch 1000, D Loss: 0.833755761384964, G Loss: 0.49966710805892944
```



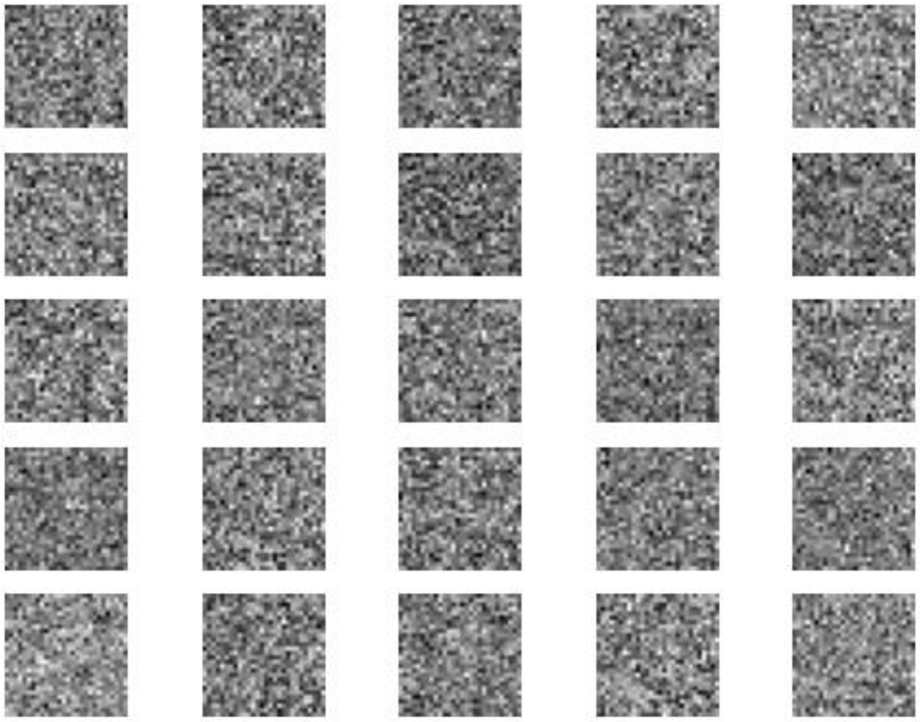
```
Epoch 2000, D Loss: 0.8236333131790161, G Loss: 0.5022673606872559
```



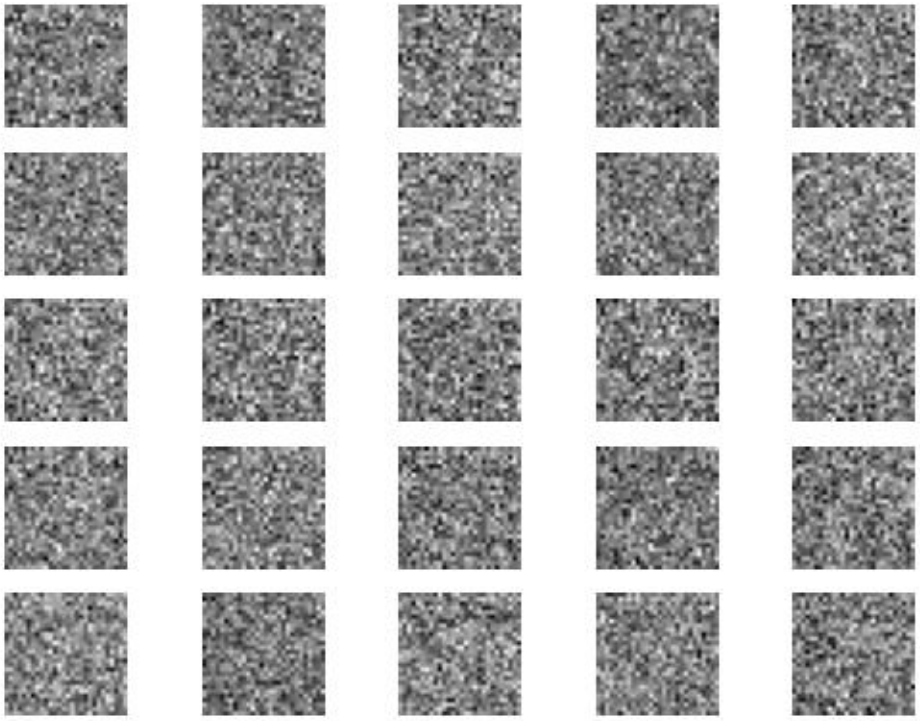
```
Epoch 3000, D Loss: 0.8098226189613342, G Loss: 0.5018783807754517
```



```
Epoch 4000, D Loss: 0.8189179599285126, G Loss: 0.4953339695930481
```



```
Epoch 5000, D Loss: 0.8311842381954193, G Loss: 0.4940934181213379
```



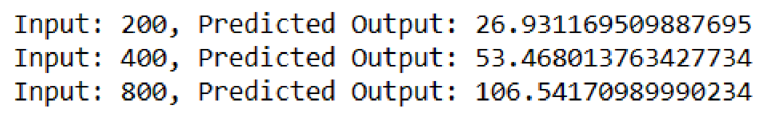
PROGRAM 14

Write a program Using Deep Learning Hands On Lab Work - Build, Test and Deploy ML Models.

Program

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

# Generate some random data for demonstration
np.random.seed(0)
x_train = np.random.rand(100, 1) * 1000
y_train = 50 * x_train + 1000 + np.random.randn(100, 1) * 200

# Build the model
model = Sequential([
    Dense(1, input_shape=(1,))
])

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

# Train the model
model.fit(x_train, y_train, epochs=100, verbose=0)

# Generate some test data
x_test = np.array([[200], [400], [800]])
y_pred = model.predict(x_test)

for i in range(len(x_test)):
    print(f"Input: {x_test[i][0]}, Predicted Output: {y_pred[i][0]}")
```

Output



```
Input: 200, Predicted Output: 26.931169509887695
Input: 400, Predicted Output: 53.468013763427734
Input: 800, Predicted Output: 106.54170989990234
```

```
# Save the model
model.save('house_price_model.h5')
```

PROGRAM 15

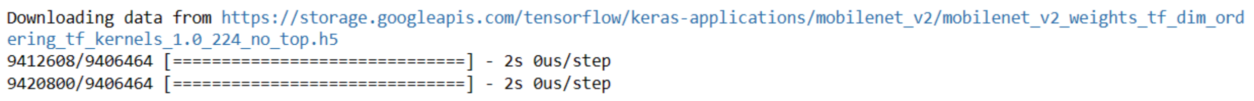
Write a program to Implement Transfer learning to retrain models that have been trained on the ImageNet dataset in order to perform classification on the CIFAR dataset.

Program

```
import tensorflow as tf
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense
from tensorflow.keras.optimizers import Adam

# Load and preprocess CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

# Load pre-trained MobileNetV2 model (excluding top layers)
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(32, 32, 3))

# Freeze base layers
for layer in base_model.layers:
    layer.trainable = False

# Add custom classification layers on top
model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(256, activation='relu'),
    Dense(10, activation='softmax')
])

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

# Train the model
model.fit(x_train, y_train, epochs=5, validation_split=0.2, verbose=2)

# Evaluate the model on test data
test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=2)
print(f"Test accuracy: {test_accuracy * 100:.2f}%")
```

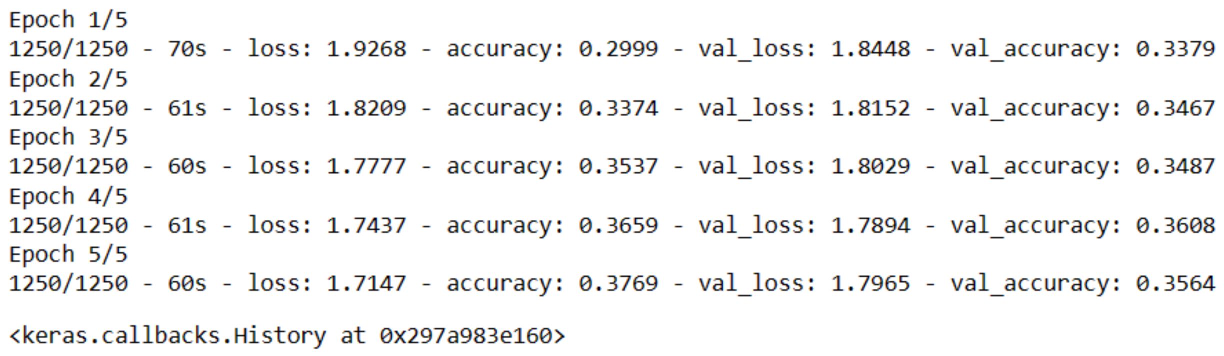
Output

Output 1 (Downloading dataset)



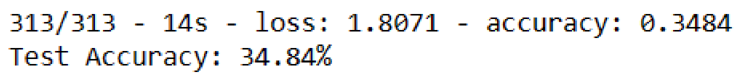
```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ord
ering_tf_kernels_1.0_224_no_top.h5
9412608/9406464 [==============================] - 2s 0us/step
9420800/9406464 [==============================] - 2s 0us/step
```

Output 2 (Fit the Model)



```
Epoch 1/5
1250/1250 - 70s - loss: 1.9268 - accuracy: 0.2999 - val_loss: 1.8448 - val_accuracy: 0.3379
Epoch 2/5
1250/1250 - 61s - loss: 1.8209 - accuracy: 0.3374 - val_loss: 1.8152 - val_accuracy: 0.3467
Epoch 3/5
1250/1250 - 60s - loss: 1.7777 - accuracy: 0.3537 - val_loss: 1.8029 - val_accuracy: 0.3487
Epoch 4/5
1250/1250 - 61s - loss: 1.7437 - accuracy: 0.3659 - val_loss: 1.7894 - val_accuracy: 0.3608
Epoch 5/5
1250/1250 - 60s - loss: 1.7147 - accuracy: 0.3769 - val_loss: 1.7965 - val_accuracy: 0.3564

<keras.callbacks.History at 0x297a983e160>
```

Output 3 (Test Accuracy)



```
313/313 - 14s - loss: 1.8071 - accuracy: 0.3484
Test Accuracy: 34.84%
```