Program 2: Write a program to demonstrate the working of a deep neural network for classification task.

```
# Import libraries
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelBinarizer

# Step 1: Load the Iris dataset
iris = load_iris()
X = iris.data                 # Features: sepal length, sepal width, petal length, petal width
y = iris.target               # Target: Species (0 = Setosa, 1 = Versicolor, 2 = Virginica)

print("Classes:", iris.target_names)
print("Shape of X:", X.shape, "Shape of y:", y.shape)

# Step 2: Preprocess the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)  # Standardize features

# One-hot encode the target labels for multi-class classification
encoder = LabelBinarizer()
y_encoded = encoder.fit_transform(y)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_encoded, test_size=0.2,
random_state=42)
```

```
# Step 3: Build a Deep Neural Network model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(10, input_shape=(4,), activation='relu'),  # Input layer with 4 features
    tf.keras.layers.Dense(8, activation='relu'),                    # Hidden layer
    tf.keras.layers.Dense(3, activation='softmax')                  # Output layer (3 classes)
])

# Step 4: Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Step 5: Train the model
history = model.fit(X_train, y_train, epochs=50, validation_data=(X_test, y_test), verbose=1)

# Step 6: Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Accuracy: {accuracy:.4f}")

# Step 7: Make predictions
predictions = model.predict(X_test)
print("\nSample Predictions:")
for i in range(5):
    true_class = np.argmax(y_test[i])
    predicted_class = np.argmax(predictions[i])
    print(f"True: {iris.target_names[true_class]}, Predicted:
{iris.target_names[predicted_class]}")

# Step 8: Plot training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
```

```
plt.title('Model Loss During Training')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Step 9: Plot training and validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy During Training')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

OUTPUT:

```
Classes: ['setosa' 'versicolor' 'virginica']
Shape of X: (150, 4) Shape of y: (150,)
Epoch 1/100
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_shape`/`input_dim` ar
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 123ms/step - accuracy: 0.3438 - loss: 1.1825 - val_accuracy: 0.2667 - val_loss: 1.1627
Epoch 2/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step - accuracy: 0.3667 - loss: 1.1247 - val_accuracy: 0.2667 - val_loss: 1.1257
Epoch 3/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - accuracy: 0.3448 - loss: 1.0989 - val_accuracy: 0.2333 - val_loss: 1.0904
Epoch 4/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.3510 - loss: 1.0627 - val_accuracy: 0.2333 - val_loss: 1.0579
Epoch 5/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step - accuracy: 0.3817 - loss: 1.0332 - val_accuracy: 0.3333 - val_loss: 1.0273
Epoch 6/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step - accuracy: 0.3856 - loss: 1.0114 - val_accuracy: 0.4000 - val_loss: 0.9977
Epoch 7/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 0.5256 - loss: 0.9843 - val_accuracy: 0.4333 - val_loss: 0.9699
Epoch 8/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step - accuracy: 0.5829 - loss: 0.9557 - val_accuracy: 0.5667 - val_loss: 0.9432
Epoch 9/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 67ms/step - accuracy: 0.6871 - loss: 0.9177 - val_accuracy: 0.5667 - val_loss: 0.9179
Epoch 10/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step - accuracy: 0.6679 - loss: 0.9081 - val_accuracy: 0.5667 - val_loss: 0.8939
Epoch 11/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step - accuracy: 0.6913 - loss: 0.8728 - val_accuracy: 0.5667 - val_loss: 0.8709
```

.

.

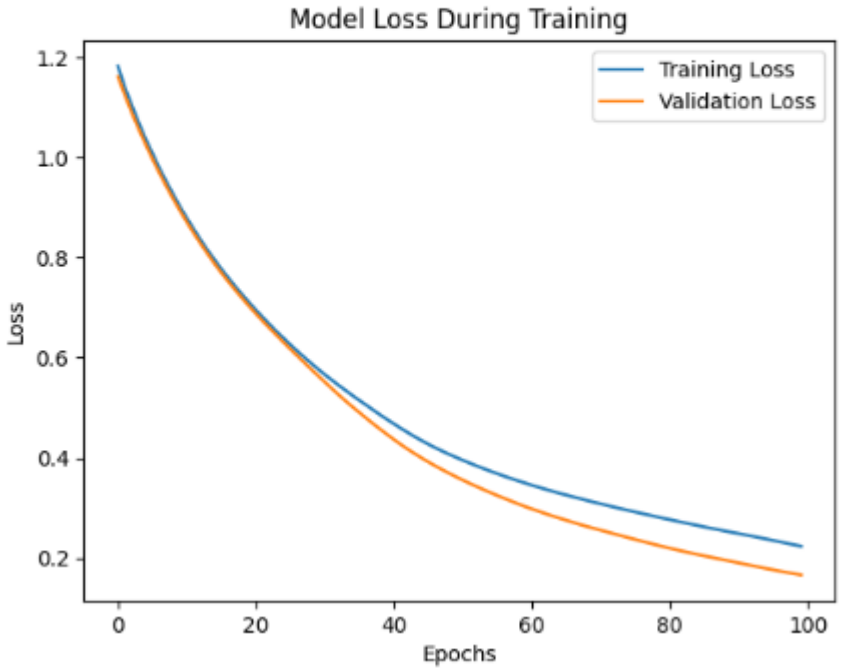
.

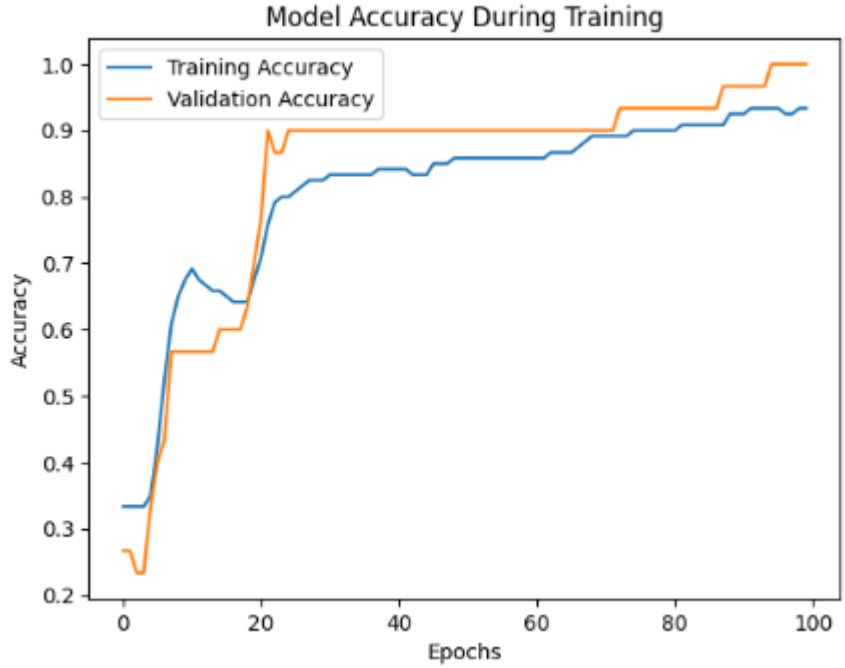
.

.

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.9112 - loss: 0.2661 - val_accuracy: 0.9333 - val_loss: 0.2156
Epoch 83/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.9123 - loss: 0.2524 - val_accuracy: 0.9333 - val_loss: 0.2124
Epoch 84/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.9040 - loss: 0.2855 - val_accuracy: 0.9333 - val_loss: 0.2094
Epoch 85/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.8894 - loss: 0.2988 - val_accuracy: 0.9333 - val_loss: 0.2063
Epoch 86/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.9029 - loss: 0.2666 - val_accuracy: 0.9333 - val_loss: 0.2035
Epoch 87/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.9071 - loss: 0.2487 - val_accuracy: 0.9333 - val_loss: 0.2007
Epoch 88/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.8935 - loss: 0.2689 - val_accuracy: 0.9667 - val_loss: 0.1979
Epoch 89/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.9190 - loss: 0.2673 - val_accuracy: 0.9667 - val_loss: 0.1951
Epoch 90/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.9096 - loss: 0.2537 - val_accuracy: 0.9667 - val_loss: 0.1922
Epoch 91/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9262 - loss: 0.2604 - val_accuracy: 0.9667 - val_loss: 0.1893
Epoch 92/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.9223 - loss: 0.2480 - val_accuracy: 0.9667 - val_loss: 0.1866
Epoch 93/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9265 - loss: 0.2669 - val_accuracy: 0.9667 - val_loss: 0.1837
Epoch 94/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9275 - loss: 0.2397 - val_accuracy: 0.9667 - val_loss: 0.1810
Epoch 95/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.9337 - loss: 0.2332 - val_accuracy: 1.0000 - val_loss: 0.1781
Epoch 96/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.9463 - loss: 0.2371 - val_accuracy: 1.0000 - val_loss: 0.1756
Epoch 97/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.9231 - loss: 0.2461 - val_accuracy: 1.0000 - val_loss: 0.1728
Epoch 98/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.9231 - loss: 0.2223 - val_accuracy: 1.0000 - val_loss: 0.1703
Epoch 99/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.9390 - loss: 0.2278 - val_accuracy: 1.0000 - val_loss: 0.1680
Epoch 100/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.9410 - loss: 0.2181 - val_accuracy: 1.0000 - val_loss: 0.1655
Test Accuracy: 1.0000
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
```

```
Sample Predictions:
True: versicolor, Predicted: versicolor
True: setosa, Predicted: setosa
True: virginica, Predicted: virginica
True: versicolor, Predicted: versicolor
True: versicolor, Predicted: versicolor
```




Model Accuracy During Training
Training Accuracy
Validation Accuracy
Accuracy
Epochs
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
20
40
60
80
100