```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

# Load dataset (replace 'your_dataset.csv' with the path to your
dataset)
data = pd.read_csv('heart.csv')

# Separate features (X) and labels (y)
X = data.drop('target', axis=1)  # Assuming 'heart_disease_label' is
the column indicating presence/absence of heart disease
y = data['target']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build the deep learning model
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.5),
    Dense(32, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

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

# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32,
validation_split=0.2)

# Evaluate the model on the test set
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test Accuracy: {accuracy}')

# Predict probabilities for the test set
y_pred_prob = model.predict(X_test)
```

```
# Convert probabilities to binary predictions (0 or 1)
y_pred = (y_pred_prob > 0.5).astype(int)

# Example of how to save the model
model.save('heart_disease_model.h5')
```

```
Epoch 1/50
7/7 [==============================] - 1s 38ms/step - loss: 0.7629 -
accuracy: 0.4922 - val_loss: 0.6612 - val_accuracy: 0.6327
Epoch 2/50
7/7 [==============================] - 0s 10ms/step - loss: 0.7507 -
accuracy: 0.4974 - val_loss: 0.6338 - val_accuracy: 0.6939
Epoch 3/50
7/7 [==============================] - 0s 7ms/step - loss: 0.7143 -
accuracy: 0.6269 - val_loss: 0.6158 - val_accuracy: 0.7347
Epoch 4/50
7/7 [==============================] - 0s 9ms/step - loss: 0.6602 -
accuracy: 0.5855 - val_loss: 0.5971 - val_accuracy: 0.7551
Epoch 5/50
7/7 [==============================] - 0s 9ms/step - loss: 0.6444 -
accuracy: 0.6321 - val_loss: 0.5797 - val_accuracy: 0.7551
Epoch 6/50
7/7 [==============================] - 0s 7ms/step - loss: 0.6099 -
accuracy: 0.6425 - val_loss: 0.5656 - val_accuracy: 0.7755
Epoch 7/50
7/7 [==============================] - 0s 7ms/step - loss: 0.6337 -
accuracy: 0.6477 - val_loss: 0.5539 - val_accuracy: 0.7755
Epoch 8/50
7/7 [==============================] - 0s 9ms/step - loss: 0.6169 -
accuracy: 0.6632 - val_loss: 0.5444 - val_accuracy: 0.7551
Epoch 9/50
7/7 [==============================] - 0s 11ms/step - loss: 0.5905 -
accuracy: 0.6684 - val_loss: 0.5340 - val_accuracy: 0.7551
Epoch 10/50
7/7 [==============================] - 0s 7ms/step - loss: 0.5583 -
accuracy: 0.6995 - val_loss: 0.5224 - val_accuracy: 0.7551
Epoch 11/50
7/7 [==============================] - 0s 7ms/step - loss: 0.5588 -
accuracy: 0.7306 - val_loss: 0.5139 - val_accuracy: 0.7755
Epoch 12/50
7/7 [==============================] - 0s 7ms/step - loss: 0.5702 -
accuracy: 0.7202 - val_loss: 0.5035 - val_accuracy: 0.7959
Epoch 13/50
7/7 [==============================] - 0s 9ms/step - loss: 0.5556 -
accuracy: 0.7409 - val_loss: 0.4944 - val_accuracy: 0.7959
Epoch 14/50
7/7 [==============================] - 0s 7ms/step - loss: 0.5479 -
accuracy: 0.7306 - val_loss: 0.4857 - val_accuracy: 0.7959
Epoch 15/50
7/7 [==============================] - 0s 8ms/step - loss: 0.5365 -
```

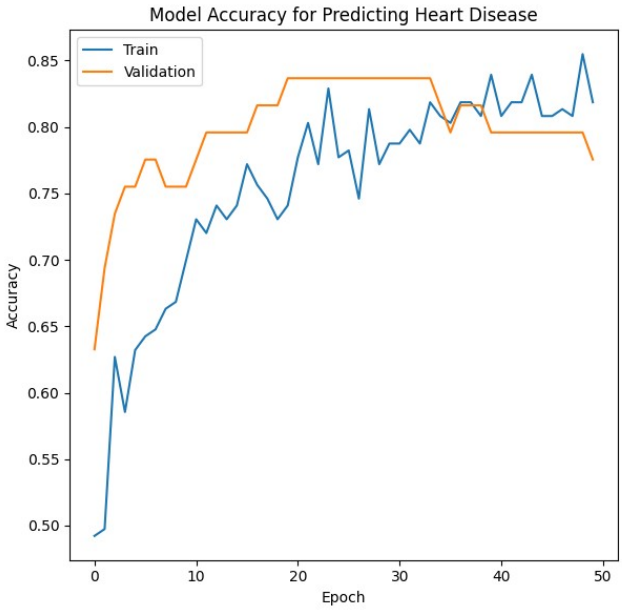
```
accuracy: 0.7409 - val_loss: 0.4767 - val_accuracy: 0.7959
Epoch 16/50
7/7 [==============================] - 0s 7ms/step - loss: 0.5163 -
accuracy: 0.7720 - val_loss: 0.4670 - val_accuracy: 0.7959
Epoch 17/50
7/7 [==============================] - 0s 10ms/step - loss: 0.5107 -
accuracy: 0.7565 - val_loss: 0.4585 - val_accuracy: 0.8163
Epoch 18/50
7/7 [==============================] - 0s 10ms/step - loss: 0.5344 -
accuracy: 0.7461 - val_loss: 0.4510 - val_accuracy: 0.8163
Epoch 19/50
7/7 [==============================] - 0s 10ms/step - loss: 0.5354 -
accuracy: 0.7306 - val_loss: 0.4435 - val_accuracy: 0.8163
Epoch 20/50
7/7 [==============================] - 0s 7ms/step - loss: 0.5034 -
accuracy: 0.7409 - val_loss: 0.4358 - val_accuracy: 0.8367
Epoch 21/50
7/7 [==============================] - 0s 9ms/step - loss: 0.5057 -
accuracy: 0.7772 - val_loss: 0.4296 - val_accuracy: 0.8367
Epoch 22/50
7/7 [==============================] - 0s 7ms/step - loss: 0.4539 -
accuracy: 0.8031 - val_loss: 0.4285 - val_accuracy: 0.8367
Epoch 23/50
7/7 [==============================] - 0s 10ms/step - loss: 0.4651 -
accuracy: 0.7720 - val_loss: 0.4262 - val_accuracy: 0.8367
Epoch 24/50
7/7 [==============================] - 0s 7ms/step - loss: 0.4403 -
accuracy: 0.8290 - val_loss: 0.4214 - val_accuracy: 0.8367
Epoch 25/50
7/7 [==============================] - 0s 7ms/step - loss: 0.4901 -
accuracy: 0.7772 - val_loss: 0.4163 - val_accuracy: 0.8367
Epoch 26/50
7/7 [==============================] - 0s 9ms/step - loss: 0.4749 -
accuracy: 0.7824 - val_loss: 0.4119 - val_accuracy: 0.8367
Epoch 27/50
7/7 [==============================] - 0s 7ms/step - loss: 0.5162 -
accuracy: 0.7461 - val_loss: 0.4078 - val_accuracy: 0.8367
Epoch 28/50
7/7 [==============================] - 0s 10ms/step - loss: 0.4423 -
accuracy: 0.8135 - val_loss: 0.4034 - val_accuracy: 0.8367
Epoch 29/50
7/7 [==============================] - 0s 7ms/step - loss: 0.4160 -
accuracy: 0.7720 - val_loss: 0.3976 - val_accuracy: 0.8367
Epoch 30/50
7/7 [==============================] - 0s 7ms/step - loss: 0.4647 -
accuracy: 0.7876 - val_loss: 0.3936 - val_accuracy: 0.8367
Epoch 31/50
7/7 [==============================] - 0s 10ms/step - loss: 0.4411 -
accuracy: 0.7876 - val_loss: 0.3899 - val_accuracy: 0.8367
```

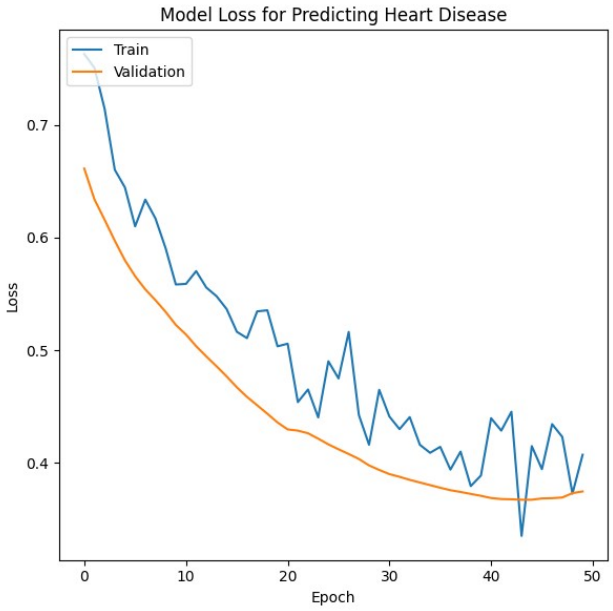
```
Epoch 32/50
7/7 [==============================] - 0s 10ms/step - loss: 0.4298 -
accuracy: 0.7979 - val_loss: 0.3875 - val_accuracy: 0.8367
Epoch 33/50
7/7 [==============================] - 0s 7ms/step - loss: 0.4405 -
accuracy: 0.7876 - val_loss: 0.3848 - val_accuracy: 0.8367
Epoch 34/50
7/7 [==============================] - 0s 7ms/step - loss: 0.4159 -
accuracy: 0.8187 - val_loss: 0.3825 - val_accuracy: 0.8367
Epoch 35/50
7/7 [==============================] - 0s 10ms/step - loss: 0.4088 -
accuracy: 0.8083 - val_loss: 0.3801 - val_accuracy: 0.8163
Epoch 36/50
7/7 [==============================] - 0s 7ms/step - loss: 0.4141 -
accuracy: 0.8031 - val_loss: 0.3778 - val_accuracy: 0.7959
Epoch 37/50
7/7 [==============================] - 0s 10ms/step - loss: 0.3939 -
accuracy: 0.8187 - val_loss: 0.3756 - val_accuracy: 0.8163
Epoch 38/50
7/7 [==============================] - 0s 7ms/step - loss: 0.4099 -
accuracy: 0.8187 - val_loss: 0.3741 - val_accuracy: 0.8163
Epoch 39/50
7/7 [==============================] - 0s 7ms/step - loss: 0.3793 -
accuracy: 0.8083 - val_loss: 0.3723 - val_accuracy: 0.8163
Epoch 40/50
7/7 [==============================] - 0s 8ms/step - loss: 0.3887 -
accuracy: 0.8394 - val_loss: 0.3706 - val_accuracy: 0.7959
Epoch 41/50
7/7 [==============================] - 0s 10ms/step - loss: 0.4396 -
accuracy: 0.8083 - val_loss: 0.3687 - val_accuracy: 0.7959
Epoch 42/50
7/7 [==============================] - 0s 10ms/step - loss: 0.4285 -
accuracy: 0.8187 - val_loss: 0.3678 - val_accuracy: 0.7959
Epoch 43/50
7/7 [==============================] - 0s 9ms/step - loss: 0.4453 -
accuracy: 0.8187 - val_loss: 0.3676 - val_accuracy: 0.7959
Epoch 44/50
7/7 [==============================] - 0s 7ms/step - loss: 0.3350 -
accuracy: 0.8394 - val_loss: 0.3672 - val_accuracy: 0.7959
Epoch 45/50
7/7 [==============================] - 0s 7ms/step - loss: 0.4148 -
accuracy: 0.8083 - val_loss: 0.3672 - val_accuracy: 0.7959
Epoch 46/50
7/7 [==============================] - 0s 9ms/step - loss: 0.3943 -
accuracy: 0.8083 - val_loss: 0.3683 - val_accuracy: 0.7959
Epoch 47/50
7/7 [==============================] - 0s 10ms/step - loss: 0.4343 -
accuracy: 0.8135 - val_loss: 0.3685 - val_accuracy: 0.7959
Epoch 48/50
```

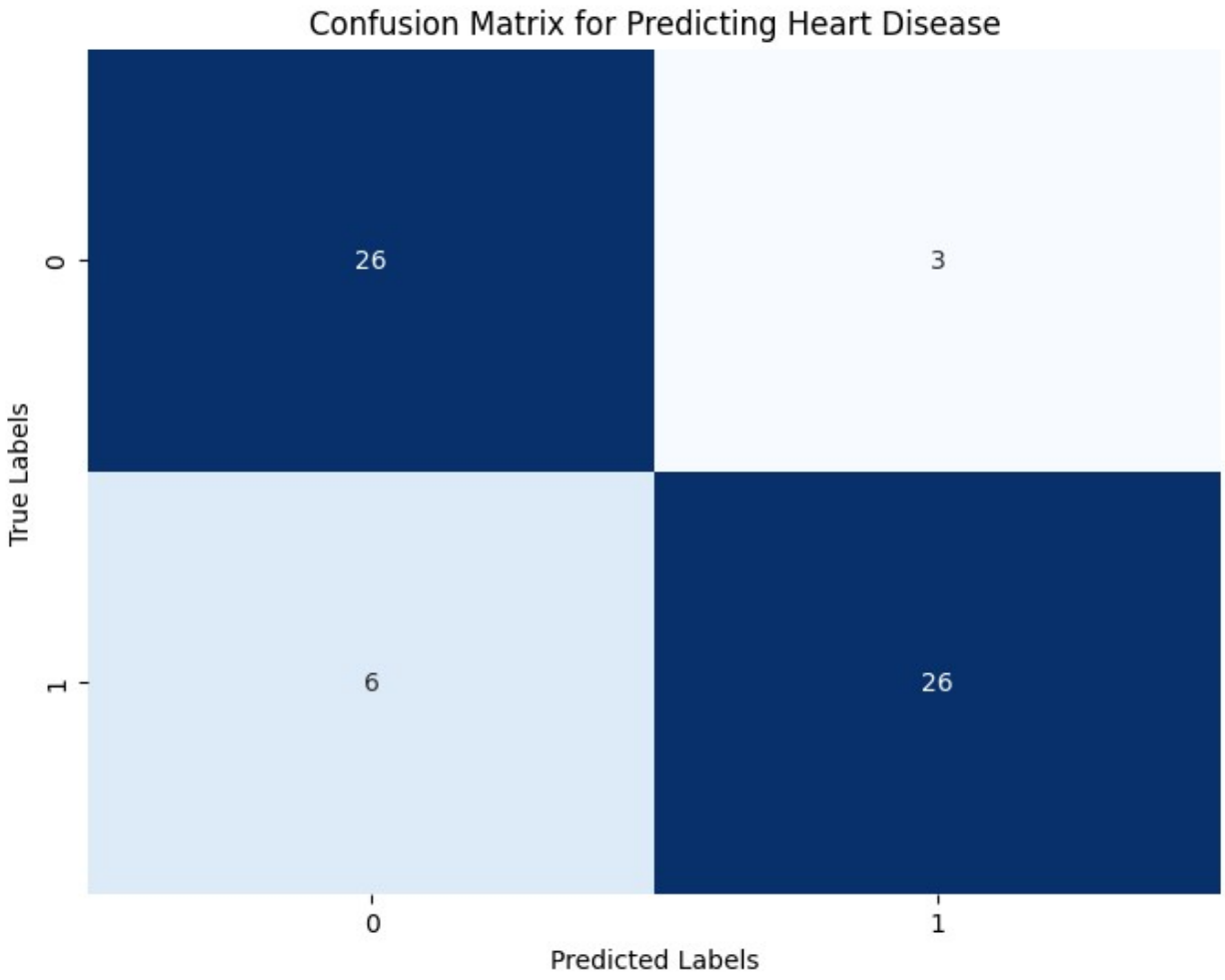
```
7/7 [==============================] - 0s 7ms/step - loss: 0.4229 -
accuracy: 0.8083 - val_loss: 0.3691 - val_accuracy: 0.7959
Epoch 49/50
7/7 [==============================] - 0s 7ms/step - loss: 0.3726 -
accuracy: 0.8549 - val_loss: 0.3730 - val_accuracy: 0.7959
Epoch 50/50
7/7 [==============================] - 0s 7ms/step - loss: 0.4071 -
accuracy: 0.8187 - val_loss: 0.3745 - val_accuracy: 0.7755
2/2 [==============================] - 0s 8ms/step - loss: 0.3842 -
accuracy: 0.8525
Test Accuracy: 0.8524590134620667
2/2 [==============================] - 0s 5ms/step

/usr/local/lib/python3.10/dist-packages/keras/src/engine/
training.py:3103: UserWarning: You are saving your model as an HDF5
file via `model.save()`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')`.
  saving_api.save_model(
```

```python
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Plot training history
def plot_history(history):
    plt.figure(figsize=(12, 6))

    # Plot training & validation accuracy values
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model Accuracy for Predicting Heart Disease')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend(['Train', 'Validation'], loc='upper left')

    # Plot training & validation loss values
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model Loss for Predicting Heart Disease')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend(['Train', 'Validation'], loc='upper left')

    plt.tight_layout()
    plt.show()

# Visualize training history
```

```
plot_history(history)

# Confusion Matrix
def plot_confusion_matrix(y_true, y_pred):
    # Calculate confusion matrix
    cm = confusion_matrix(y_true, y_pred)

    # Plot confusion matrix
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
    plt.title('Confusion Matrix for Predicting Heart Disease')
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.show()

# Visualize confusion matrix
plot_confusion_matrix(y_test, y_pred)
```




Confusion Matrix for Predicting Heart Disease
26
3
6
26
0
1
True Labels
0
1
Predicted Labels