```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow import keras
from tensorflow.keras import layers
```

```
# Load the dataset
data = load_breast_cancer()

# Create a DataFrame
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target

# Display first few rows
print(df.head())
```

```
   mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0        17.99         10.38          122.80     1001.0          0.11840
1        20.57         17.77          132.90     1326.0          0.08474
2        19.69         21.25          130.00     1203.0          0.10960
3        11.42         20.38           77.58      386.1          0.14250
4        20.29         14.34          135.10     1297.0          0.10030

   mean compactness  mean concavity  mean concave points  mean symmetry  \
0           0.27760          0.3001              0.14710         0.2419
1           0.07864          0.0869              0.07017         0.1812
2           0.15990          0.1974              0.12790         0.2069
3           0.28390          0.2414              0.10520         0.2597
4           0.13280          0.1980              0.10430         0.1809

   mean fractal dimension  ...  worst texture  worst perimeter  worst area  \
0                 0.07871  ...          17.33           184.60      2019.0
1                 0.05667  ...          23.41           158.80      1956.0
2                 0.05999  ...          25.53           152.50      1709.0
3                 0.09744  ...          26.50            98.87       567.7
4                 0.05883  ...          16.67           152.20      1575.0

   worst smoothness  worst compactness  worst concavity  worst concave points  \
0            0.1622             0.6656           0.7119                0.2654
1            0.1238             0.1866           0.2416                0.1860
2            0.1444             0.4245           0.4504                0.2430
3            0.2098             0.8663           0.6869                0.2575
4            0.1374             0.2050           0.4000                0.1625

   worst symmetry  worst fractal dimension  target
```

```
0          0.4601                  0.11890       0
1          0.2750                  0.08902       0
2          0.3613                  0.08758       0
3          0.6638                  0.17300       0
4          0.2364                  0.07678       0

[5 rows x 31 columns]
```

```
# Separate features and target
X = df.drop('target', axis=1)
y = df['target']

# Normalize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split into train/test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

```
# Build the model
model = keras.Sequential([
    layers.Dense(30, input_dim=X_train.shape[1], activation='relu'),
    layers.Dense(15, activation='relu'),
    layers.Dense(1, activation='sigmoid')  # Binary output
])

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

# Summary of the model
model.summary()
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequent
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 30)	930
dense_1 (Dense)	(None, 15)	465
dense_2 (Dense)	(None, 1)	16

```
Total params: 1,411 (5.51 KB)
Trainable params: 1,411 (5.51 KB)
Non-trainable params: 0 (0.00 B)
```

```
# Train
```

```
history = model.fit(X_train, y_train, validation_split=0.2, epochs=50, batch_size=16, verbose=1)
```

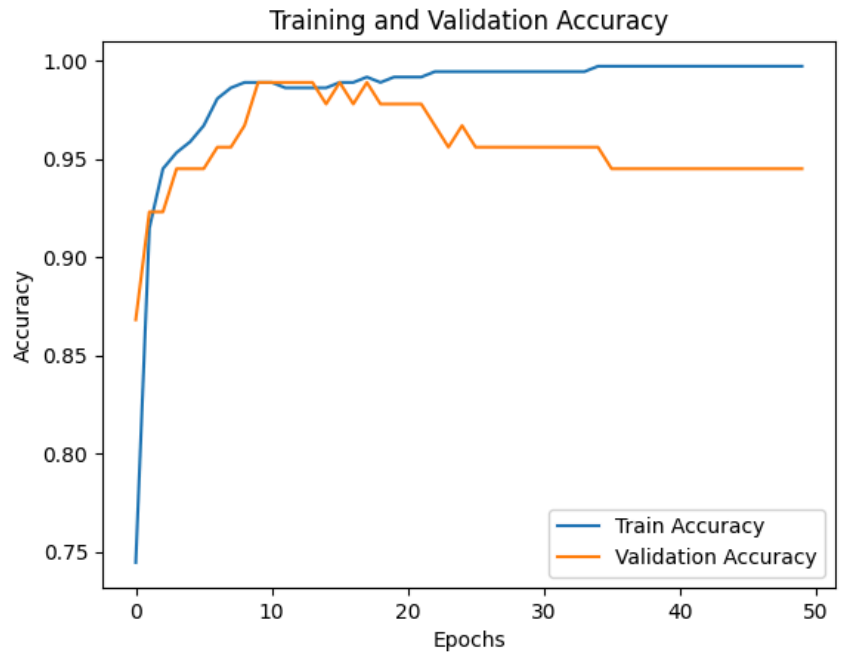
```
Epoch 1/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - accuracy: 0.6800 - loss: 0.6441 - val_accuracy: 0.8681 - val_loss: 0.4944
Epoch 2/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8961 - loss: 0.4522 - val_accuracy: 0.9231 - val_loss: 0.3562
Epoch 3/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9349 - loss: 0.3129 - val_accuracy: 0.9231 - val_loss: 0.2551
Epoch 4/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9674 - loss: 0.1886 - val_accuracy: 0.9451 - val_loss: 0.1971
Epoch 5/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9583 - loss: 0.1506 - val_accuracy: 0.9451 - val_loss: 0.1626
Epoch 6/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9667 - loss: 0.1139 - val_accuracy: 0.9451 - val_loss: 0.1424
Epoch 7/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9821 - loss: 0.0957 - val_accuracy: 0.9560 - val_loss: 0.1264
Epoch 8/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9859 - loss: 0.0778 - val_accuracy: 0.9560 - val_loss: 0.1164
Epoch 9/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9858 - loss: 0.0979 - val_accuracy: 0.9670 - val_loss: 0.1110
Epoch 10/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9822 - loss: 0.0746 - val_accuracy: 0.9890 - val_loss: 0.1052
Epoch 11/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9918 - loss: 0.0579 - val_accuracy: 0.9890 - val_loss: 0.1015
Epoch 12/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9902 - loss: 0.0499 - val_accuracy: 0.9890 - val_loss: 0.0979
Epoch 13/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9825 - loss: 0.0662 - val_accuracy: 0.9890 - val_loss: 0.0961
Epoch 14/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9850 - loss: 0.0552 - val_accuracy: 0.9890 - val_loss: 0.0944
Epoch 15/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9792 - loss: 0.0594 - val_accuracy: 0.9780 - val_loss: 0.0928
Epoch 16/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9944 - loss: 0.0339 - val_accuracy: 0.9890 - val_loss: 0.0920
Epoch 17/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9866 - loss: 0.0573 - val_accuracy: 0.9780 - val_loss: 0.0904
Epoch 18/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9876 - loss: 0.0583 - val_accuracy: 0.9890 - val_loss: 0.0898
Epoch 19/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9870 - loss: 0.0463 - val_accuracy: 0.9780 - val_loss: 0.0902
Epoch 20/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9891 - loss: 0.0553 - val_accuracy: 0.9780 - val_loss: 0.0896
Epoch 21/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9926 - loss: 0.0420 - val_accuracy: 0.9780 - val_loss: 0.0906
Epoch 22/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9851 - loss: 0.0376 - val_accuracy: 0.9780 - val_loss: 0.0909
Epoch 23/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9941 - loss: 0.0340 - val_accuracy: 0.9670 - val_loss: 0.0913
Epoch 24/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9933 - loss: 0.0310 - val_accuracy: 0.9560 - val_loss: 0.0926
Epoch 25/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9920 - loss: 0.0351 - val_accuracy: 0.9670 - val_loss: 0.0923
Epoch 26/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9959 - loss: 0.0211 - val_accuracy: 0.9560 - val_loss: 0.0938
```

```
Epoch 27/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9992 - loss: 0.0180 - val_accuracy: 0.9560 - val_loss: 0.0933
Epoch 28/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9960 - loss: 0.0174 - val_accuracy: 0.9560 - val_loss: 0.0961
Epoch 29/50
```

```
# Evaluate on test data
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy:.4f}")

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

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9651 - loss: 0.1312
Test Accuracy: 0.9649
```




```
# Predict
y_pred = (model.predict(X_test) > 0.5).astype("int32")

# Display first 10 predictions
```

```
print("Predictions:", y_pred[:10].flatten())
print("Actual:", y_test[:10].values)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
Predictions: [1 0 0 1 1 0 0 0 1 1]
Actual: [1 0 0 1 1 0 0 0 1 1]
```