In [8]:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer
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
from tensorflow.keras.activations import softmax
from tensorflow.keras.losses import CategoricalCrossentropy
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import f1_score
import matplotlib.pyplot as plt
```

In [9]:

```
# Load the Breast Cancer Wisconsin (Diagnostic) dataset
cancer = load_breast_cancer()
```

In [10]:

```
# Separate features and target variable
X = cancer.data
y = cancer.target
```

In [11]:

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
```

In [12]:

```
# Scale the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [13]:

```
# Define the model
model = Sequential()
model.add(Dense(2, activation='softmax', input_shape=(30,)))
```

In [14]:

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

In [15]:

```
# Initialize lists to store epoch numbers, losses, and accuracies
epoch_nums = []
train_losses = []
train_accuracies = []
```

In [16]:

```
# Train the model
for epoch in range(50):
    # Perform one forward pass and backward pass on the training data
    history = model.fit(X_train, tf.keras.utils.to_categorical(y_train, 2), epochs=1, ba

    # Evaluate the model on the training data
    train_loss, train_accuracy = model.evaluate(X_train, tf.keras.utils.to_categorical(y

    # Store the results for visualization
    epoch_nums.append(epoch + 1)
    train_losses.append(train_loss)
    train_accuracies.append(train_accuracy)

    # Print the epoch number, train loss, and train accuracy
    print("Epoch:", epoch + 1)
    print("Train Loss:", train_loss)
    print("Train Accuracy:", train_accuracy)
```

```
Epoch: 1
Train Loss: 0.9934546947479248
Train Accuracy: 0.3780219852924347
Epoch: 2
Train Loss: 0.8063403964042664
Train Accuracy: 0.5296703577041626
Epoch: 3
Train Loss: 0.6682331562042236
Train Accuracy: 0.6615384817123413
Epoch: 4
Train Loss: 0.5695553421974182
Train Accuracy: 0.7186813354492188
Epoch: 5
Train Loss: 0.4950856566429138
Train Accuracy: 0.7802197933197021
Epoch: 6
Train Loss: 0.43951016664505005
Train Accuracy: 0.8175824284553528
Epoch: 7
```

In [17]:

```
# Evaluate the model
test_loss, test_accuracy = model.evaluate(X_test, tf.keras.utils.to_categorical(y_test,
y_pred = np.argmax(model.predict(X_test), axis=1)
f1 = f1_score(y_test, y_pred)
```

```
4/4 [==============================] - 0s 0s/step
```

In [18]:

```
# Print the results
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
print("F1-Score:", f1)
```

```
Test Loss: 0.08721235394477844
Test Accuracy: 0.9824561476707458
F1-Score: 0.9861111111111112
```

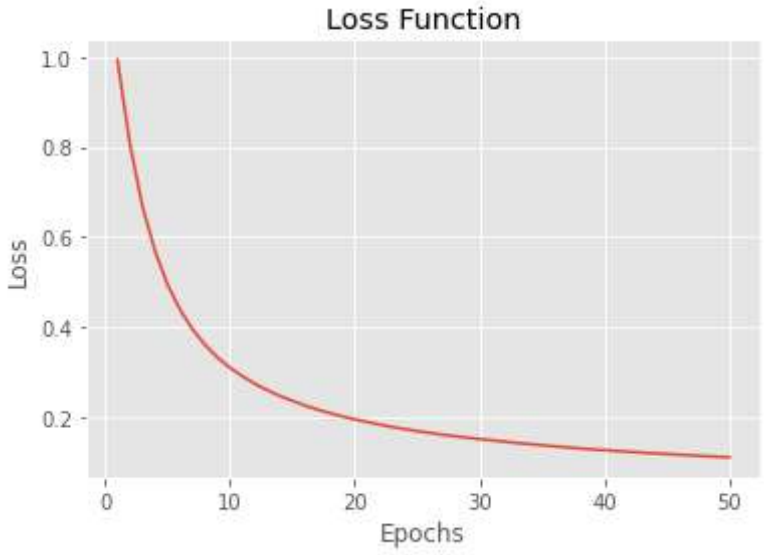
In [19]:

```
# Print the final parameters
for i, layer in enumerate(model.layers):
    print("Layer", i+1, "Weights:", layer.get_weights())
```

```
Layer 1 Weights: [array([[-0.16102266, -0.54661924],
       [ 0.00552027, -0.15919626],
       [ 0.5874314 ,  0.20764787],
       [ 0.24901517,  0.01468777],
       [ 0.5266852 ,  0.1273124 ],
       [-0.24390493, -0.40986007],
       [-0.18706168, -0.15753932],
       [ 0.5361548 , -0.14898497],
       [-0.274333  , -0.15609373],
       [ 0.16250761,  0.3918067 ],
       [-0.1493611 , -0.14080182],
       [ 0.46743363,  0.00847287],
       [ 0.16116324, -0.39473748],
       [ 0.48314258, -0.26222566],
       [-0.1515579 , -0.18279599],
       [ 0.13043681,  0.10134412],
       [ 0.11057015,  0.36381644],
       [-0.07052645, -0.16376445],
       [-0.03118137,  0.04548181],
       [ 0.00625769,  0.48310092],
       [ 0.27529573, -0.37720975],
       [ 0.12068448,  0.06442825],
       [ 0.09060749, -0.4934956 ],
       [ 0.21867129, -0.29124117],
       [-0.13835727, -0.0856163 ],
       [-0.2405604 ,  0.22122596],
       [ 0.5906039 , -0.13735138],
       [ 0.06788319, -0.22387575],
       [ 0.19314943, -0.4810161 ],
       [ 0.24017599, -0.3967704 ]], dtype=float32), array([-0.16972512,
0.1697251 ], dtype=float32)]
```

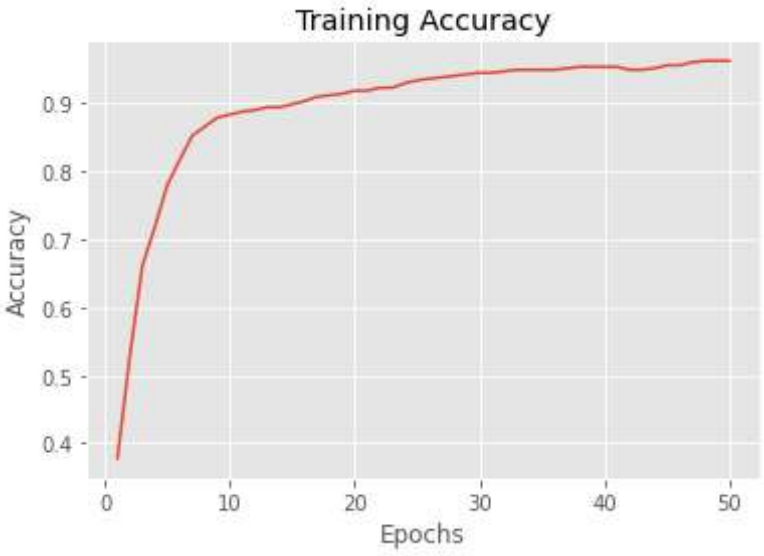
In [20]:

```
# Plot the loss function vs. epochs
plt.plot(epoch_nums, train_losses)
plt.title('Loss Function')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
```



In [21]:

```
# Plot the training accuracy vs. epochs
plt.plot(epoch_nums, train_accuracies)
plt.title('Training Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.show()
```



In [22]:

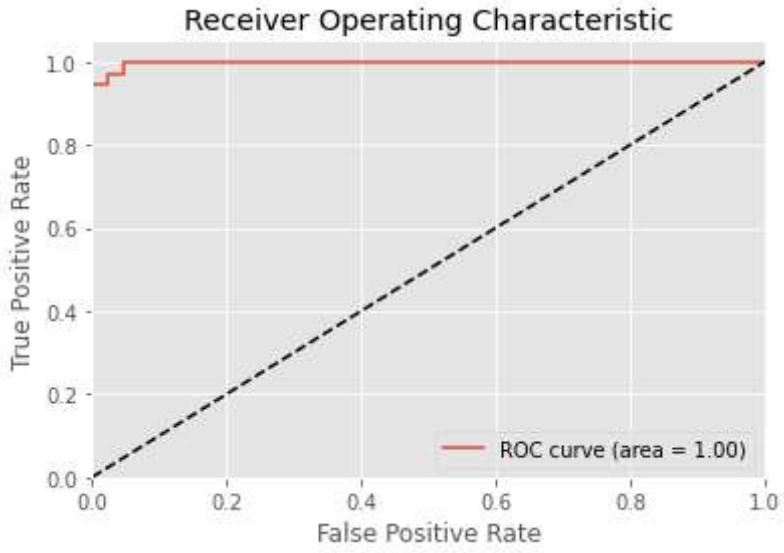
```
# Generate predictions on the test set
y_pred_prob = model.predict(X_test)[:, 1]
```

```
4/4 [==============================] - 0s 5ms/step
```

In [23]:

```
# Plot the ROC curve
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
roc_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```



In [24]:

```
# Visualize a few examples of misclassified samples
misclassified_indices = np.where(y_test != y_pred)[0][:5]
misclassified_samples = X_test[misclassified_indices]
misclassified_labels = y_test[misclassified_indices]
misclassified_pred = y_pred[misclassified_indices]

for i, sample in enumerate(misclassified_samples):
    print("Misclassified Sample", i + 1)
    print("True Label:", misclassified_labels[i])
    print("Predicted Label:", misclassified_pred[i])
    print("Features:", sample)
    print("---------")
```

```
Misclassified Sample 1
True Label: 0
Predicted Label: 1
Features: [-0.08993252 -0.79671034 -0.05977463 -0.19821465  0.35634783  0.
4651665
 -0.13867567  0.06339385 -0.54261701  0.40312655 -0.43597673 -1.07582927
 -0.435613   -0.35438207 -0.74409507 -0.26823247 -0.47441781 -0.42752674
 -1.02579279 -0.24800235  0.06968337 -0.77176511  0.09589304 -0.11390033
  0.41496361  0.65522912  0.01580277  0.369597   -0.50097175  1.07313539]
---------
Misclassified Sample 2
True Label: 0
Predicted Label: 1
Features: [-0.04463148 -0.5010269  -0.01861438 -0.14660035  0.99628119  0.
46325857
  0.11327535  0.10995031  0.35332661 -0.20227891  0.08125937 -0.72601977
 -0.14270817 -0.09179518 -0.20896683  0.06211843 -0.05087698  0.24272529
  0.00569498 -0.4042485   0.03223001 -0.57204371  0.02990222 -0.0899165
  0.85300607  0.47440179  0.22137928  0.3558048   0.25835983 -0.24106908]
---------
```

In [ ]: