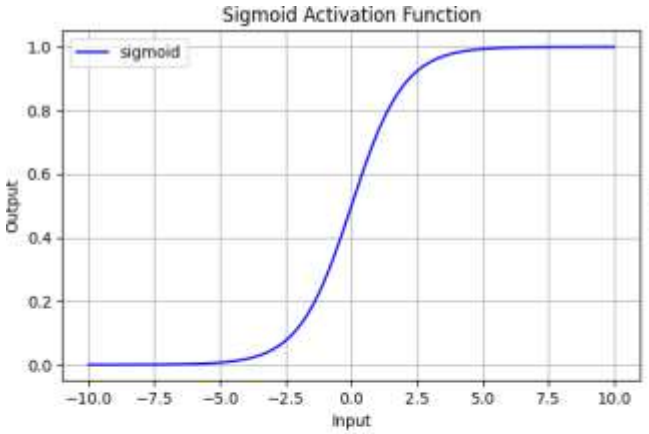
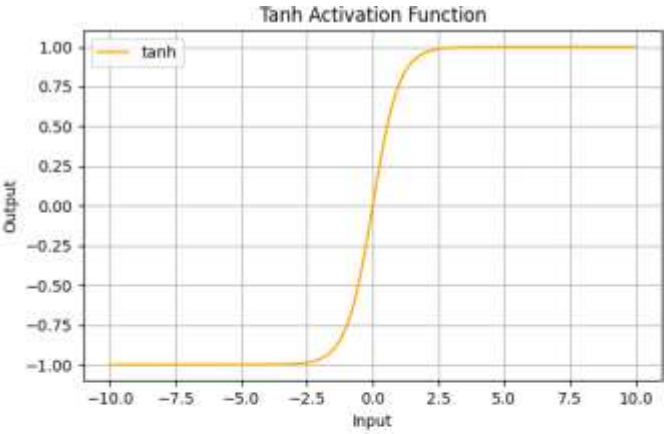
1. **Write a program to demonstrate the working of different activation functions like Sigmoid, Tanh, RELU and softmax to train neural network.**

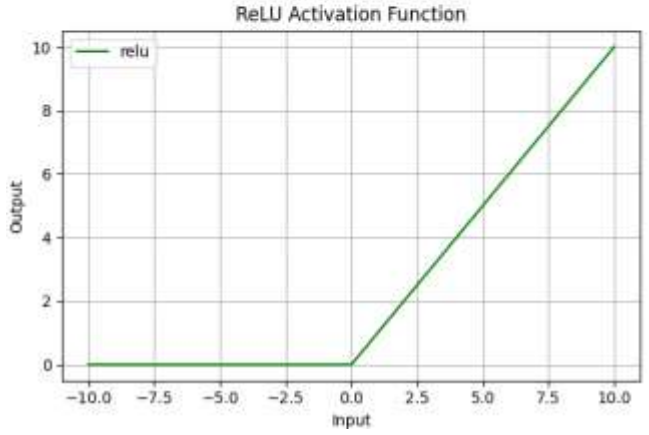
**PROGRAM:**

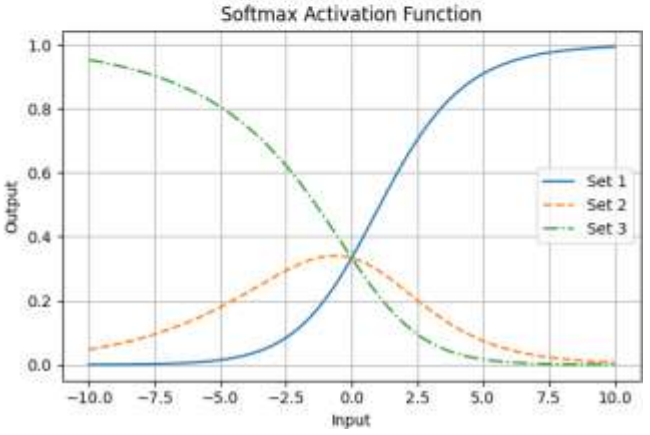
```
import numpy as np
import matplotlib.pyplot as plt

def softmax(x):
    e_x = np.exp(x - np.max(x))
return e_x / np.sum(e_x, axis=0)
x = np.linspace(-10, 10, 400)
y = {
    'sigmoid': 1 / (1 + np.exp(-
    x)), 'tanh': np.tanh(x),
    'relu': np.maximum(0, x),
    'softmax': softmax(np.array([x, x * 0.5, x * 0.2])).T
}

plt.figure(figsize=(12, 8))
titles = ["Sigmoid", "Tanh", "ReLU", "Softmax"]
colors = ['blue', 'orange', 'green']
markers = ['-', '--', '-.']

for i, (key, y_values) in enumerate(y.items()):
    plt.subplot(2, 2, i + 1)
    if key == 'softmax':
        for j in range(y_values.shape[1]):
            plt.plot(x, y_values[:, j], label=f"Set {j+1}", linestyle=markers[j])
    else:
plt.plot(x, y_values, label=key,color=colors[i])
plt.title(f"{titles[i]} Activation Function")
plt.xlabel("Input")
plt.ylabel("Output")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

**OUTPUT:**


Sigmoid Activation Function
sigmoid
Output
Input


Tanh Activation Function
tanh
Output
Input


ReLU Activation Function
relu
Output
Input


Softmax Activation Function
Set 1
Set 2
Set 3
Output
Input

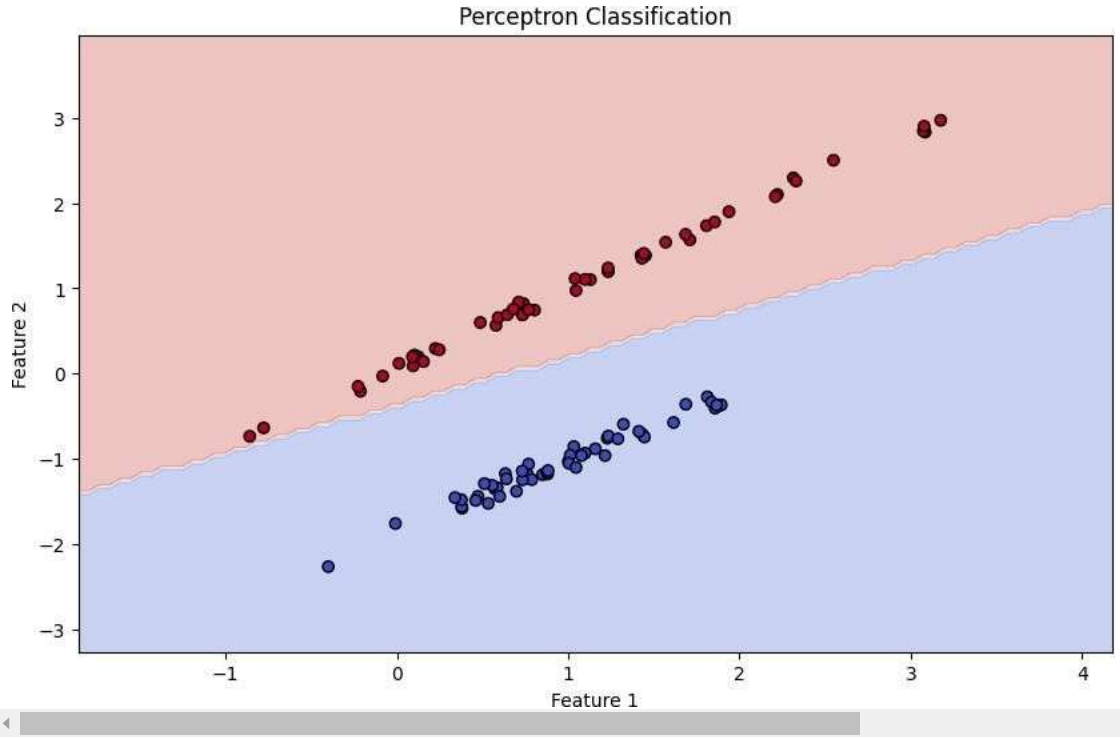
2. **Design a single unit perceptron for classification of a linearly separable binary dataset without using pre-defined models. Use the Perceptron() from sklearn.**

**PROGRAM 2a:**

```
import numpy as np
from sklearn.datasets import
make_classification from
sklearn.linear_model import Perceptron
from sklearn.model_selection import
train_test_split from sklearn.metrics import
accuracy_score
import matplotlib.pyplot as plt

X, y = make_classification(
    n_samples=100, n_features=2,
    n_informative=2, n_redundant=0,
    n_clusters_per_class=1,
    flip_y=0, random_state=42
)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42) perceptron = Perceptron(max_iter=1000, eta0=1,
random_state=42)
perceptron.fit(X_train, y_train)
accuracy = accuracy_score(y_test, perceptron.predict(X_test))
print(f'Accuracy: {accuracy*100:.2f}%')

xx, yy = np.meshgrid(
    np.linspace(X[:, 0].min() - 1, X[:, 0].max() + 1, 100),
    np.linspace(X[:, 1].min() - 1, X[:, 1].max() + 1, 100)
)
Z = perceptron.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)

plt.figure(figsize=(10, 6))
plt.contourf(xx, yy, Z, alpha=0.3, cmap='coolwarm')
plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k', cmap='coolwarm',
marker='o') plt.title('Perceptron Classification')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

**OUTPUT:**

Accuracy: 100.00%



**PROGRAM 2b:**

**Identify the problem with single unit Perceptron. Classify using Or, And and Xor data and analyze the result.**

```
import numpy as np
from sklearn.linear_model import
Perceptron Perceptron from
sklearn.metrics import accuracy_score

# Data for AND, OR, XOR
gates data = {
    'AND': (np.array([[0, 0], [0, 1], [1, 0], [1, 1]]), np.array([0, 0, 0, 1])), # Use 0 and 1 for labels
    'OR':  (np.array([[0, 0], [0, 1], [1, 0], [1, 1]]), np.array([0, 1, 1, 1])),  # Use 0 and 1 for labels
    'XOR': (np.array([[0, 0], [0, 1], [1, 0], [1, 1]]), np.array([0, 1, 1, 0])),  # Use 0 and 1 for labels
}

# Classify AND, OR, XOR gates
for gate, (X, y) in data.items():
    perceptron = Perceptron(max_iter=10, eta0=1,
    random_state=42) perceptron.fit(X, y)
    y_pred = perceptron.predict(X)

    acc = accuracy_score(y, y_pred) * 100
    print(f"{gate} gate accuracy: {acc:.2f}%")
    print(f"Predictions: {y_pred}")
    print(f"True Labels: {y}")
```

**OUTPUT:**

```
AND gate accuracy: 100.00%
Predictions: [0 0 0 1]
True Labels: [0 0 0 1]
OR gate accuracy: 100.00%
Predictions: [0 1 1 1]
True Labels: [0 1 1 1]
XOR gate accuracy: 50.00%
Predictions: [0 0 0 0]
True Labels: [0 1 1 0]
```

3. **Build a Deep Feed Forward ANN by implementing the Backpropagation algorithm and test the same using appropriate data sets. Use the number of hidden layers >= 4.**

**PROGRAM:**

```
import tensorflow as tf
from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import
Dense from sklearn.datasets import
load_iris
from sklearn.model_selection import
train_test_split from sklearn.preprocessing
import StandardScaler

# Load and preprocess the
dataset X, y =
load_iris(return_X_y=True)
X =
StandardScaler().fit_transform(
X) y =
tf.keras.utils.to_categorical(y)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Build the model
model = Sequential([
    Dense(64,activation='relu',input_shape=(X_train.sha
    pe[1],)),
    Dense(64, activation='relu'),
    Dense(64,
    activation='relu'),
    Dense(64,
    activation='relu'), Dense(3,
    activation='softmax')
])

# Compile and train the model
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy']) model.fit(X_train, y_train, epochs=10, batch_size=8,
validation_split=0.2, verbose=1)



# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test Accuracy: {accuracy * 100:.2f}%\n Loss:{loss:.4f}')
```

## OUTPUT:

```
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` ar
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
11/11 ━━━━━━━━━━━━━━━━━━━━ 4s 78ms/step - accuracy: 0.5407 - loss: 1.0698 - val_accuracy: 0.8571 - val_loss: 0.9564
Epoch 2/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9116 - loss: 0.8843 - val_accuracy: 0.8571 - val_loss: 0.7717 Epoch
3/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8130 - loss: 0.6995 - val_accuracy: 0.8571 - val_loss: 0.5626 Epoch
4/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8672 - loss: 0.4670 - val_accuracy: 0.8571 - val_loss: 0.4379
Epoch 5/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8468 - loss: 0.3530 - val_accuracy: 0.8571 - val_loss: 0.4036
Epoch 6/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8839 - loss: 0.3035 - val_accuracy: 0.8571 - val_loss: 0.3514 Epoch
7/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8740 - loss: 0.2461 - val_accuracy: 0.9048 - val_loss: 0.3770 Epoch
8/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8927 - loss: 0.2424 - val_accuracy: 0.9048 - val_loss: 0.3248
Epoch 9/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8861 - loss: 0.2388 - val_accuracy: 0.9048 - val_loss: 0.3205 Epoch
10/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9562 - loss: 0.1965 - val_accuracy: 0.9048 - val_loss: 0.2874
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 272ms/step - accuracy: 1.0000 - loss: 0.1256
Test Accuracy: 100.00%
 Loss:0.1323
```

4. **Design and implement a CNN model (with 4+ layers of convolutions) to classify multi category image datasets. Use the concept of regularization and dropout while designing the CNN model. Use the Fashion MNIST datasets. Record the Training accuracy and Test accuracy corresponding to the following architectures:**

    5. **Base Model**
    6. **Model with L1 Regularization**
    7. **Model with L2 Regularization**
    8. **Model with Dropout**

**PROGRAM:**

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import
fashion_mnist from
tensorflow.keras.regularizers import l1, l2

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
train_images, test_images = train_images[..., None] / 255.0, test_images[...,

None] / 255.0 # Function to build and evaluate a model

def build_and_evaluate(name, regularizer=None,
    dropout_rate=None): model = models.Sequential([

        layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1), kernel_regularizer=regularizer),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu', kernel_regularizer=regularizer),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(128, (3, 3), activation='relu', kernel_regularizer=regularizer),
        layers.Conv2D(128, (3, 3), activation='relu',
        kernel_regularizer=regularizer), layers.Flatten(),
        layers.Dense(128, activation='relu', kernel_regularizer=regularizer),
        layers.Dense(10, activation='softmax')
    ])
    if dropout_rate:
        model.add(layers.Dropout(dropout_rate))
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    model.fit(train_images, train_labels, epochs=5, batch_size=64,
    validation_split=0.2, verbose=1) loss, accuracy = model.evaluate(test_images,
    test_labels)
    print(f'Test Accuracy: {accuracy * 100:.2f}%\n Loss:{loss:.2f}')
```

```
build_and_evaluate("Base Model")
build_and_evaluate("L1Regularization",
regularizer=l1(0.001))
build_and_evaluate("L2Regularization",
regularizer=l2(0.001))
build_and_evaluate("Dropout", dropout_rate=0.5)
```

**OUTPUT:**

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/
      super().__init__(activity_regularizer=activity_regularizer, **kwargs)
    Epoch 1/5
    750/750 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.6975 - loss: 0.8091 - val_accuracy: 0.8512 - val_loss: 0.4041
    Epoch 2/5
    750/750 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8623 - loss: 0.3723 - val_accuracy: 0.8799 - val_loss: 0.3328
    Epoch 3/5
    750/750 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8893 - loss: 0.2981 - val_accuracy: 0.8957 - val_loss: 0.2928
    Epoch 4/5
    750/750 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9023 - loss: 0.2647 - val_accuracy: 0.9003 - val_loss: 0.2703
    Epoch 5/5
    750/750 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9132 - loss: 0.2345 - val_accuracy: 0.8986 - val_loss: 0.2797
    313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8901 - loss: 0.2951 Test
    Accuracy: 89.14%
     Loss:0.29
    Epoch 1/5
    750/750 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - accuracy: 0.5560 - loss: 2.9135 - val_accuracy: 0.6642 - val_loss: 1.2041
    Epoch 2/5
    750/750 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - accuracy: 0.7141 - loss: 1.1246 - val_accuracy: 0.7202 - val_loss: 1.0185
    Epoch 3/5
    750/750 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.7337 - loss: 0.9687 - val_accuracy: 0.7440 - val_loss: 0.9106
    Epoch 4/5
    750/750 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.7395 - loss: 0.9041 - val_accuracy: 0.7574 - val_loss: 0.8700
    Epoch 5/5
    750/750 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.7519 - loss: 0.8598 - val_accuracy: 0.7635 - val_loss: 0.8320
    313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7563 - loss: 0.8554 Test
    Accuracy: 75.53%
     Loss:0.86 Epoch
    1/5
    750/750 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - accuracy: 0.6934 - loss: 1.0686 - val_accuracy: 0.8332 - val_loss: 0.6342
    Epoch 2/5
    750/750 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8523 - loss: 0.5880 - val_accuracy: 0.8637 - val_loss: 0.5265
    Epoch 3/5
    750/750 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8711 - loss: 0.5070 - val_accuracy: 0.8702 - val_loss: 0.4991
    Epoch 4/5
    750/750 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8818 - loss: 0.4640 - val_accuracy: 0.8838 - val_loss: 0.4504
    Epoch 5/5
    750/750 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8839 - loss: 0.4399 - val_accuracy: 0.8944 - val_loss: 0.4331
    313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8913 - loss: 0.4435 Test
    Accuracy: 88.84%
     Loss:0.44 Epoch
    1/5
    750/750 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.3123 - loss: 5.4538 - val_accuracy: 0.6701 - val_loss: 30.1995
    Epoch 2/5
    750/750 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3834 - loss: 3.8609 - val_accuracy: 0.6644 - val_loss: 26.6286
    Epoch 3/5
    750/750 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.4032 - loss: 3.8324 - val_accuracy: 0.6751 - val_loss: 37.2124
    Epoch 4/5
    750/750 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4016 - loss: 3.7548 - val_accuracy: 0.6628 - val_loss: 45.2641
    Epoch 5/5
    750/750 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4018 - loss: 3.6351 - val_accuracy: 0.7153 - val_loss: 45.3600
    313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7078 - loss: 47.5595 Test
    Accuracy: 71.01%
     Loss:46.90
```

**5. Design and implement an Image classification model to classify a dataset of images using Deep Feed Forward Neural Network. Record the accuracy corresponding to the number of epochs. Use the MNIST datasets.**

**PROGRAM:**

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

(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train, X_test = X_train / 255.0, X_test / 255.0

model = Sequential([
    Flatten(input_shape=(28,
    28)),          Dense(128,
    activation='relu'),
    Dense(256,
    activation='relu'),
    Dense(10,
    activation='softmax')
])

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy']) model.fit(X_train, y_train, epochs=5, batch_size=64,
validation_split=0.2, verbose=1)

loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test Loss: {loss:.4f}')
print(f'Test Accuracy: {accuracy*100:.2f}%')
```

**OUTPUT:**

```
usr/local/lib/python3.10/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_ super().__
init__(**kwargs)
    Epoch 1/5
    750/750 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.8439 - loss: 0.5218 - val_accuracy: 0.9622 - val_loss: 0.1327
    Epoch 2/5
    750/750 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9647 - loss: 0.1184 - val_accuracy: 0.9658 - val_loss: 0.1072
    Epoch 3/5
    750/750 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9766 - loss: 0.0748 - val_accuracy: 0.9718 - val_loss: 0.0930
    Epoch 4/5
    750/750 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9832 - loss: 0.0537 - val_accuracy: 0.9751 - val_loss: 0.0830
    Epoch 5/5
    750/750 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9886 - loss: 0.0366 - val_accuracy: 0.9717 - val_loss: 0.0997
    313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9692 - loss: 0.1030 Test
    Loss: 0.0866
    Test Accuracy: 97.3
```