EX:No.1 DATE: 21/2/25	**Working of CNN Architecture to Classify Images**

AIM:

To write a program to classify images using CNN architecture.

ALGORITHM:

1. Start
2. Import Libraries: TensorFlow/Keras, NumPy, etc.
3. Load & Preprocess Data: Normalize images, one-hot encode labels, split into train/test.
4. Build CNN Model: Add Conv2D → ReLU → MaxPooling layers.
5. Flatten → Dense → Output (Softmax/Sigmoid).
6. Compile Model: Define optimizer, loss, and metrics.
7. Train Model: Fit model on training data.
8. Evaluate Model: Test on validation/test data.
9. Predict: Use model to classify new images.
10. Stop

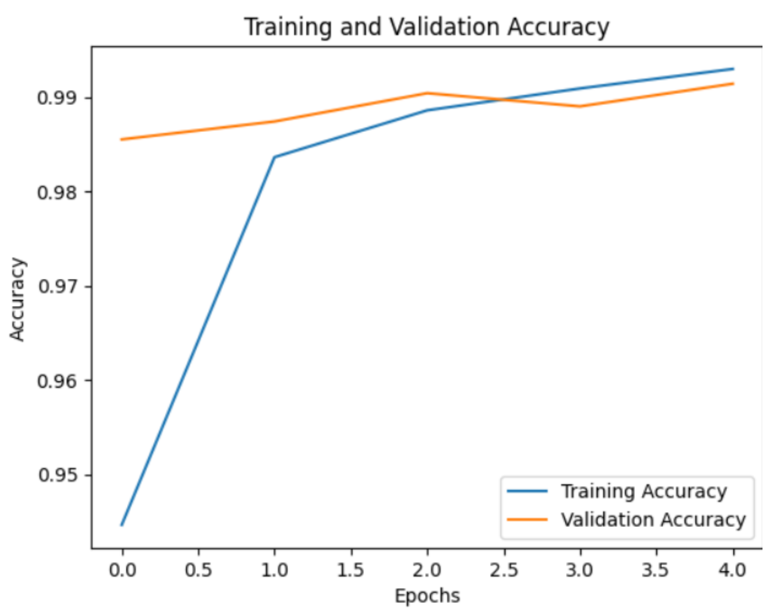
CODE:

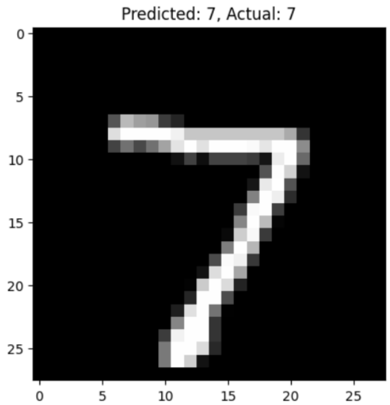
```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt
# Load the MNIST dataset (28x28 grayscale images of digits)
(x_train, y_train), (x_test, y_test) = mnist.load_data()
# Preprocess the data
# Normalize the image data to a range of 0 to 1 by dividing by 255
x_train, x_test = x_train / 255.0, x_test / 255.0
# Reshape the data to match the input shape for the CNN
x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)
```

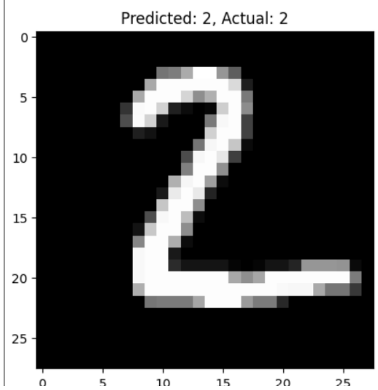
```python
# Build the CNN model
    model = models.Sequential()

    # Add convolutional layer with 32 filters, 3x3 kernel, and ReLU activation
    model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))

    # Add pooling layer to reduce spatial dimensions
    model.add(layers.MaxPooling2D((2, 2)))

    # Add second convolutional layer with 64 filters, 3x3 kernel, and ReLU activation
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))

    # Add second pooling layer
    model.add(layers.MaxPooling2D((2, 2)))
    # Add third convolutional layer with 64 filters, 3x3 kernel, and ReLU activation
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    # Flatten the 3D output to 1D for the fully connected layers
    model.add(layers.Flatten())
    # Add a dense layer with 64 units and ReLU activation
    model.add(layers.Dense(64, activation='relu'))
    # Output layer with 10 units (for the 10 digits) and softmax activation for classification
    model.add(layers.Dense(10, activation='softmax'))
    # Compile the model with an appropriate loss function and optimizer
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    # Train the model
    history = model.fit(x_train, y_train, epochs=5, batch_size=64, validation_data=(x_test, y_test))
    # Evaluate the model on the test data
    test_loss, test_acc = model.evaluate(x_test, y_test)
    print(f"Test accuracy: {test_acc:.4f}")
```

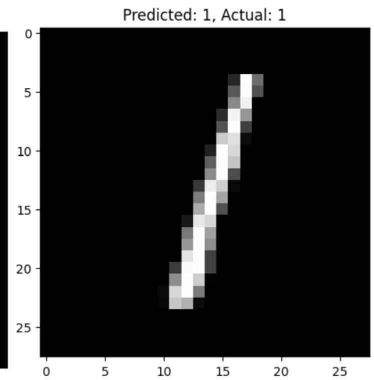
```
# Plotting training and validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.show()
# Make predictions on a test image
predictions = model.predict(x_test[:5])
# Print out predictions for the first 5 test images
for i, prediction in enumerate(predictions):
    print(f"Predicted: {prediction.argmax()}, Actual: {y_test[i]}")
    # Display the image
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    plt.title(f"Predicted: {prediction.argmax()}, Actual: {y_test[i]}")
    plt.show()
```

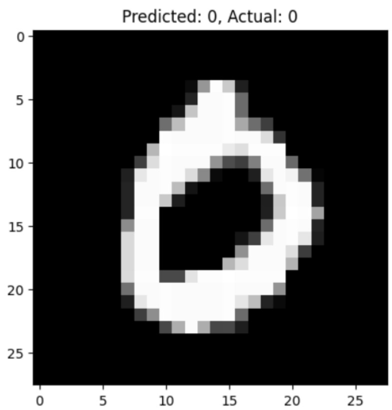
OUTPUT:

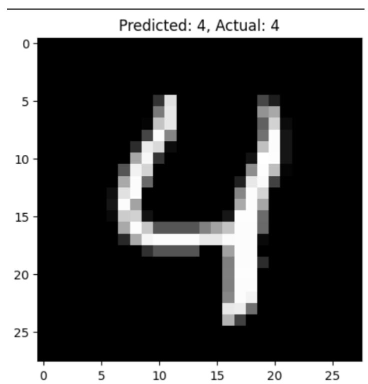












RESULT:

Thus the program has been completed and verified successfully.