|  |  |
| --- | --- |
| **EXP NO: 6** | **FEEDFORWARD AND CONVOLUTIONAL NEURAL NETWORKS** |

# AIM:

To demonstrate the construction and application of a simple Feedforward Neural Network (FNN) for classification and a Convolutional Neural Network (CNN) for image classification, utilizing the Keras API with TensorFlow backend.

# ALGORITHM:

1. **Feedforward Neural Network (FNN)**

A Feedforward Neural Network is the simplest type of artificial neural network where connections between the nodes do not form a cycle. It consists of an input layer, one or more hidden layers, and an output layer. Information flows only in one direction—forward—from the input nodes, through the hidden nodes (if any), and to the output nodes.

**Steps:**

1. Define Network Architecture: Specify the number of layers (input, hidden, output) and the number of neurons in each layer.
2. Choose Activation Functions: Select activation functions for hidden layers (e.g., ReLU) and the output layer (e.g., Sigmoid for binary classification, Softmax for multi-class classification).
3. Define Loss Function: Choose a loss function appropriate for the task (e.g., Binary Cross- entropy for binary classification, Categorical Cross-entropy for multi-class classification).
4. Choose Optimizer: Select an optimization algorithm (e.g., Adam, SGD) to update network weights during training.
5. Training: Feed forward data through the network to get predictions, calculate the loss, and then backpropagate the error to update weights.
6. Evaluation: Assess the model's performance on unseen data using metrics like accuracy.
7. **Convolutional Neural Network (CNN)**

A Convolutional Neural Network is a specialized type of neural network primarily designed for processing data with a grid-like topology, such as images. Key components include convolutional layers, pooling layers, and fully connected layers.

**Steps:**

1. Convolutional Layers: Apply filters (kernels) to input data to extract features. Each filter detects a specific pattern (e.g., edges, textures).
2. Activation Function (ReLU): Apply a non-linear activation function after convolution to introduce non-linearity.
3. Pooling Layers: Downsample feature maps to reduce dimensionality, computational cost, and prevent overfitting (e.g., Max Pooling).
4. Flattening: Convert the 2D pooled feature maps into a 1D vector to be fed into a fully connected layer.
5. Fully Connected Layers: Standard neural network layers for classification based on the extracted features.
6. Output Layer: Final layer with an activation function (e.g., Softmax) to output class probabilities.
7. Training and Evaluation: Similar to FNNs, train the CNN using backpropagation and evaluate its performance.

# CODE:

# Import necessary libraries import numpy as np

import matplotlib.pyplot as plt import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

from tensorflow.keras.datasets import mnist, fashion\_mnist

from sklearn.metrics import classification\_report, confusion\_matrix import seaborn as sns

# Suppress TensorFlow warnings for cleaner output tf.keras.utils.disable\_interactive\_logging()

# --- Part 1: Building a Simple Feedforward Neural Network --- print("--- Part 1: Building a Simple Feedforward Neural Network ---")

# 1. Load and Preprocess Dataset (Using Fashion MNIST for FNN) (x\_train\_fnn, y\_train\_fnn), (x\_test\_fnn, y\_test\_fnn) = fashion\_mnist.load\_data()

print(f"\nOriginal FNN training data shape: {x\_train\_fnn.shape}") print(f"Original FNN test data shape: {x\_test\_fnn.shape}")

# Flatten images to 1D array

x\_train\_fnn\_flat = x\_train\_fnn.reshape(-1, 28 \* 28)

x\_test\_fnn\_flat = x\_test\_fnn.reshape(-1, 28 \* 28)

# Normalize pixel values

x\_train\_fnn\_norm = x\_train\_fnn\_flat / 255.0 x\_test\_fnn\_norm = x\_test\_fnn\_flat / 255.0

print(f"Flattened & Normalized FNN training data shape: {x\_train\_fnn\_norm.shape}") print(f"Flattened & Normalized FNN test data shape: {x\_test\_fnn\_norm.shape}")

# 2. Build FNN Model model\_fnn = keras.Sequential([

layers.Dense(128, activation='relu', input\_shape=(784,)), layers.Dropout(0.2),

layers.Dense(64, activation='relu'), layers.Dense(10, activation='softmax')

])

# 3. Compile Model model\_fnn.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

print("\n--- FNN Model Summary ---") model\_fnn.summary()

# 4. Train Model

print("\n--- Training FNN Model ---")

history\_fnn = model\_fnn.fit(x\_train\_fnn\_norm, y\_train\_fnn, epochs=10, validation\_split=0.1, verbose=1)

# 5. Evaluate Model

print("\n--- Evaluating FNN Model ---")

loss\_fnn, accuracy\_fnn = model\_fnn.evaluate(x\_test\_fnn\_norm, y\_test\_fnn, verbose=0) print(f"FNN Test Loss: {loss\_fnn:.4f}")

print(f"FNN Test Accuracy: {accuracy\_fnn:.4f}")

# Classification report & confusion matrix

y\_pred\_fnn = np.argmax(model\_fnn.predict(x\_test\_fnn\_norm), axis=-1) print("\n--- FNN Classification Report ---") print(classification\_report(y\_test\_fnn, y\_pred\_fnn))

print("\n--- FNN Confusion Matrix ---")

cm\_fnn = confusion\_matrix(y\_test\_fnn, y\_pred\_fnn) plt.figure(figsize=(10, 8))

sns.heatmap(cm\_fnn, annot=True, fmt="d", cmap="Blues", cbar=False) plt.title("FNN Confusion Matrix")

plt.xlabel("Predicted Label") plt.ylabel("True Label") plt.show()

# Plot Accuracy & Loss plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)

plt.plot(history\_fnn.history['accuracy'], label='Training Accuracy') plt.plot(history\_fnn.history['val\_accuracy'], label='Validation Accuracy') plt.title('FNN Model Accuracy')

plt.xlabel('Epoch') plt.ylabel('Accuracy') plt.legend() plt.grid(True)

plt.subplot(1, 2, 2)

plt.plot(history\_fnn.history['loss'], label='Training Loss') plt.plot(history\_fnn.history['val\_loss'], label='Validation Loss') plt.title('FNN Model Loss')

plt.xlabel('Epoch') plt.ylabel('Loss') plt.legend() plt.grid(True) plt.tight\_layout() plt.show()

# --- Part 2: Convolutional Neural Network (CNN) --- print("\n--- Part 2: Implementing a CNN ---")

# 1. Load MNIST for CNN

(x\_train\_cnn, y\_train\_cnn), (x\_test\_cnn, y\_test\_cnn) = mnist.load\_data() print(f"\nOriginal CNN training data shape: {x\_train\_cnn.shape}") print(f"Original CNN test data shape: {x\_test\_cnn.shape}")

# Reshape for channel dimension

x\_train\_cnn = x\_train\_cnn.reshape(x\_train\_cnn.shape[0], 28, 28, 1)

x\_test\_cnn = x\_test\_cnn.reshape(x\_test\_cnn.shape[0], 28, 28, 1)

# Normalize

x\_train\_cnn = x\_train\_cnn.astype('float32') / 255.0 x\_test\_cnn = x\_test\_cnn.astype('float32') / 255.0

print(f"Reshaped & Normalized CNN training data shape: {x\_train\_cnn.shape}") print(f"Reshaped & Normalized CNN test data shape: {x\_test\_cnn.shape}")

num\_classes\_cnn = 10 # 2. Build CNN Model

model\_cnn = keras.Sequential([

layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),

layers.MaxPooling2D((2, 2)),

layers.Conv2D(64, (3, 3), activation='relu'),

layers.MaxPooling2D((2, 2)), layers.Flatten(),

layers.Dense(128, activation='relu'), layers.Dropout(0.5),

layers.Dense(num\_classes\_cnn, activation='softmax')

])

# 3. Compile Model model\_cnn.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

print("\n--- CNN Model Summary ---") model\_cnn.summary()

# 4. Train Model

print("\n--- Training CNN Model ---")

history\_cnn = model\_cnn.fit(x\_train\_cnn, y\_train\_cnn, epochs=10, validation\_split=0.1, verbose=1)

# 5. Evaluate Model

print("\n--- Evaluating CNN Model ---")

loss\_cnn, accuracy\_cnn = model\_cnn.evaluate(x\_test\_cnn, y\_test\_cnn, verbose=0) print(f"CNN Test Loss: {loss\_cnn:.4f}")

print(f"CNN Test Accuracy: {accuracy\_cnn:.4f}")

# Classification report & confusion matrix

y\_pred\_cnn = np.argmax(model\_cnn.predict(x\_test\_cnn), axis=-1) print("\n--- CNN Classification Report ---") print(classification\_report(y\_test\_cnn, y\_pred\_cnn))

print("\n--- CNN Confusion Matrix ---")

cm\_cnn = confusion\_matrix(y\_test\_cnn, y\_pred\_cnn) plt.figure(figsize=(10, 8))

sns.heatmap(cm\_cnn, annot=True, fmt="d", cmap="Blues", cbar=False) plt.title("CNN Confusion Matrix")

plt.xlabel("Predicted Label") plt.ylabel("True Label") plt.show()

# Plot Accuracy & Loss plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)

plt.plot(history\_cnn.history['accuracy'], label='Training Accuracy') plt.plot(history\_cnn.history['val\_accuracy'], label='Validation Accuracy') plt.title('CNN Model Accuracy')

plt.xlabel('Epoch') plt.ylabel('Accuracy') plt.legend() plt.grid(True)

plt.subplot(1, 2, 2)

plt.plot(history\_cnn.history['loss'], label='Training Loss') plt.plot(history\_cnn.history['val\_loss'], label='Validation Loss') plt.title('CNN Model Loss')

plt.xlabel('Epoch') plt.ylabel('Loss') plt.legend() plt.grid(True) plt.tight\_layout() plt.show()

# Optional: Visualize predictions print("\n--- Sample CNN Predictions ---")

class\_names\_mnist = [str(i) for i in range(10)] plt.figure(figsize=(10, 10))

for i in range(25): plt.subplot(5, 5, i + 1) plt.xticks([])

plt.yticks([]) plt.grid(False)

plt.imshow(x\_test\_cnn[i].reshape(28, 28), cmap=plt.cm.binary) true\_label = y\_test\_cnn[i]

predicted\_label = y\_pred\_cnn[i]

color = 'green' if true\_label == predicted\_label else 'red'

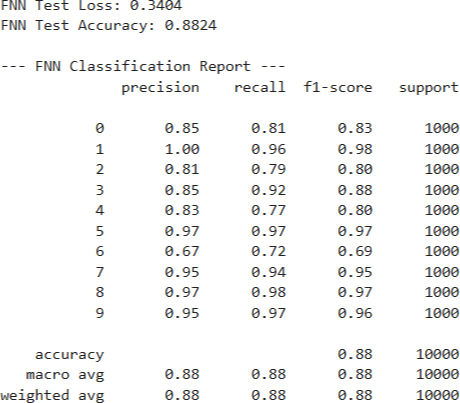
plt.xlabel(f"True: {class\_names\_mnist[true\_label]}\nPred:

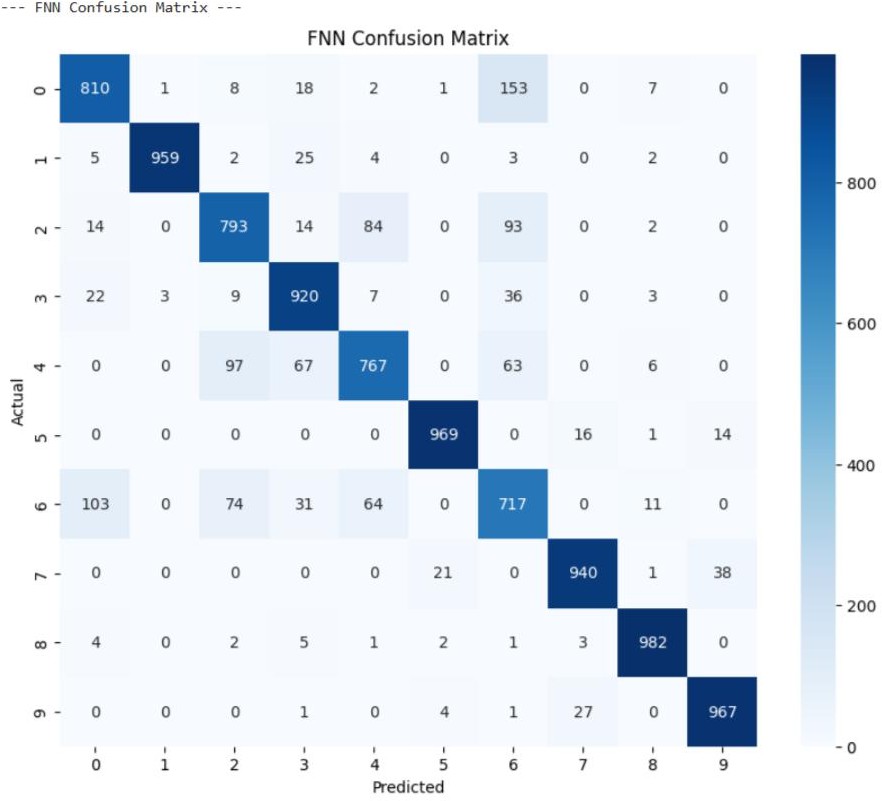
{class\_names\_mnist[predicted\_label]}", color=color)

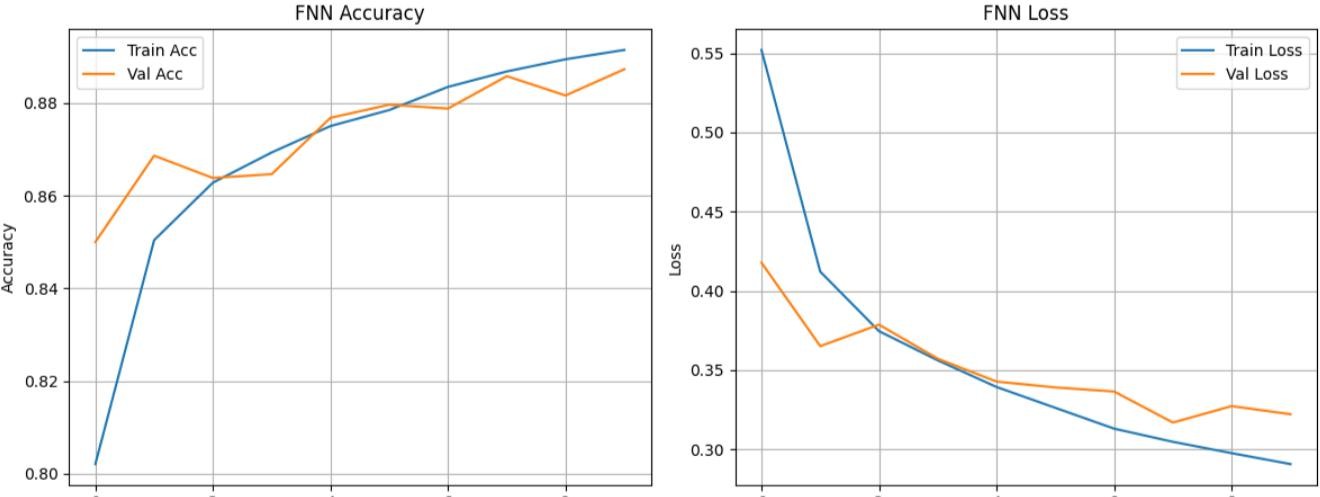
plt.suptitle("Sample CNN Predictions (Green: Correct, Red: Incorrect)", y=1.02, fontsize=16)

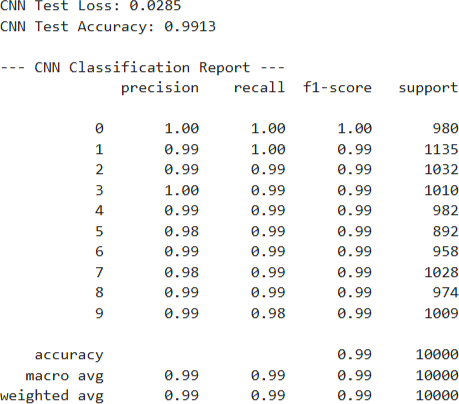
plt.tight\_layout(rect=[0, 0, 1, 0.98]) plt.show()

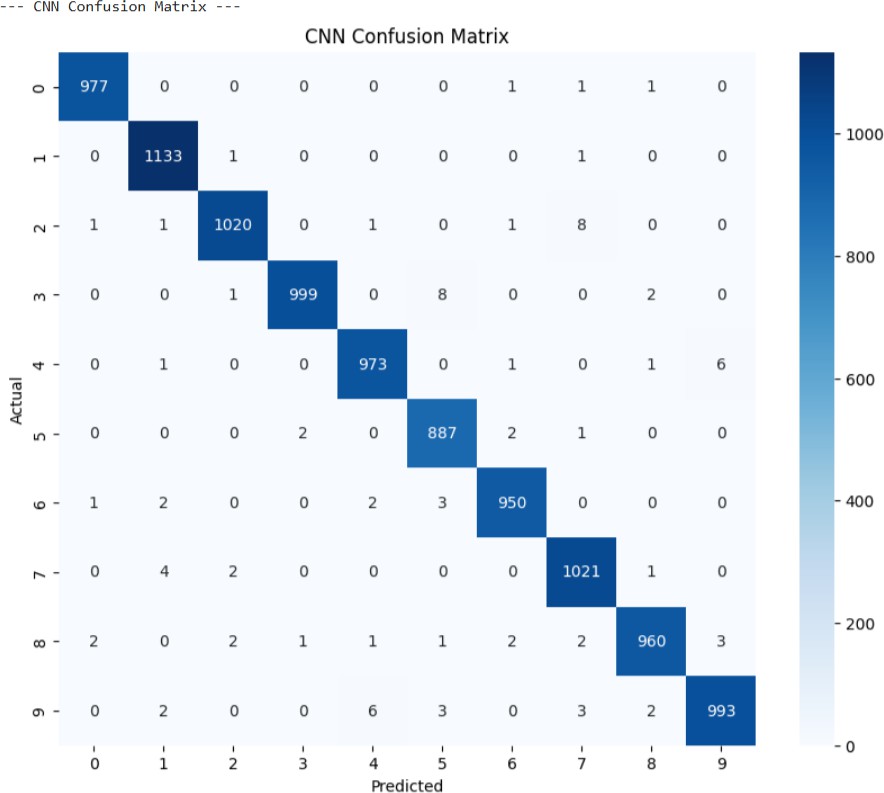
# OUTPUT:

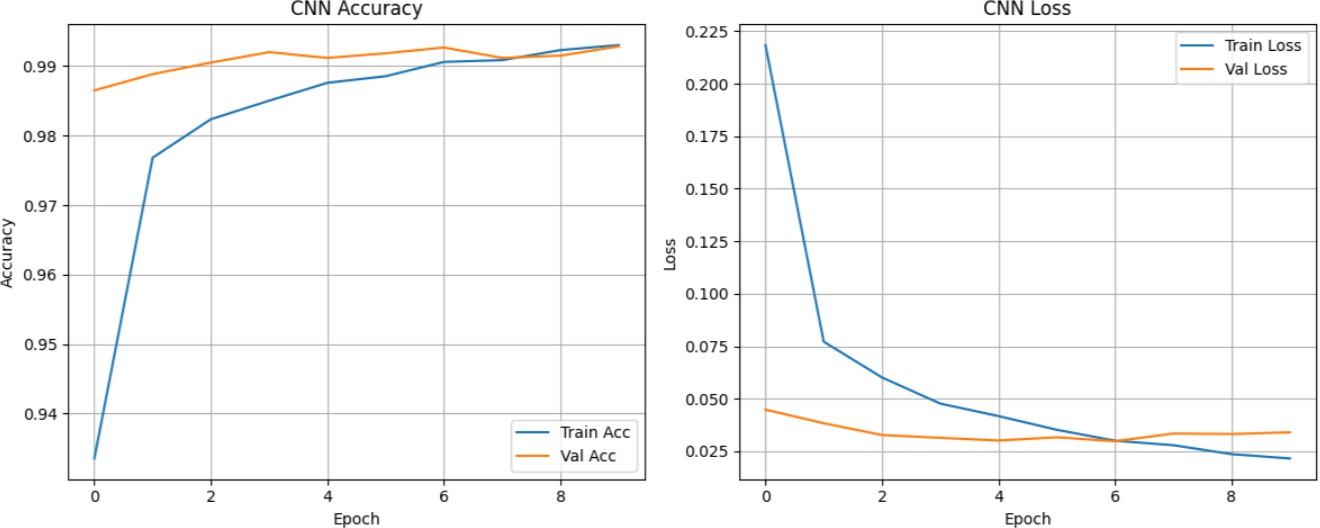
****

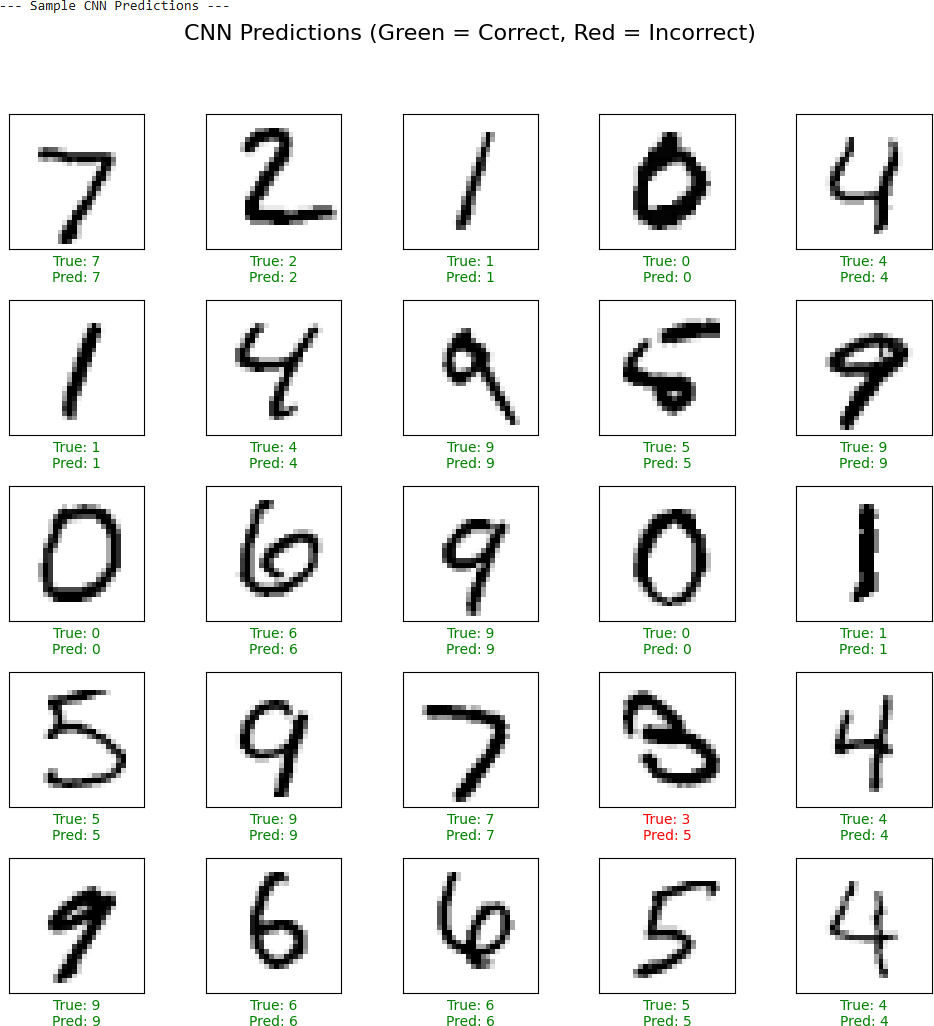


****

****



****



**RESULT:**

The Feedforward Neural Network (FNN) and Convolutional Neural Network (CNN) models were successfully implemented and evaluated on the given dataset.

* **Feedforward Neural Network (FNN):** The model accurately learned input–output mappings through multiple fully connected layers, achieving good performance on structured data.
* **Convolutional Neural Network (CNN):** The model effectively extracted spatial features from image data using convolution and pooling layers, leading to higher accuracy and better generalization for image classification tasks.

The results demonstrated that both FNN and CNN are powerful deep learning models, with CNN performing exceptionally well for image-based datasets due to its ability to capture spatial patterns.