A convolutional neural network (CNN) is a type of deep learning algorithm specifically designed for processing structured grids of data such as images. It's inspired by the organization of the animal visual cortex, with neurons arranged in a hierarchical manner, each responsible for recognizing increasingly complex patterns.

At its core, a CNN consists of multiple layers, each performing different operations on the input data. These layers typically include convolutional layers, pooling layers, and fully connected layers.

Convolutional layers are the building blocks of CNNs. They apply a set of learnable filters (also called kernels) to the input image, performing convolution operations. Each filter detects specific features, like edges or textures, across the image by sliding over it and computing dot products between the filter and the local regions of the input. This process enables the network to automatically learn hierarchical representations of features at different levels of abstraction.

Pooling layers follow convolutional layers and downsample the feature maps, reducing their spatial dimensions while retaining the most important information. Max pooling, for example, selects the maximum value from each local region of the feature map, effectively reducing its size.

Fully connected layers come after convolutional and pooling layers and serve to classify the extracted features. They take the flattened output of the preceding layers and apply a series of matrix multiplications and non-linear activation functions to produce the final output, typically representing class probabilities.

Training a CNN involves optimization techniques such as backpropagation and gradient descent. During the training process, the network learns to adjust the weights of its filters and fully connected layers to minimize the difference between its predictions and the ground truth labels of the training data.

CNNs have achieved remarkable success in various computer vision tasks, including image classification, object detection, facial recognition, and image segmentation. Their ability to automatically learn hierarchical representations of features from raw data makes them particularly well-suited for tasks involving structured grid-like data, like images. Additionally, techniques like transfer learning allow CNNs to leverage pre-trained models on large datasets, reducing the need for extensive computational resources and labeled data for training custom models.

CODE:

```
import tensorflow as tf
from tensorflow.keras import layers, models

# Load CIFAR-10 dataset
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.cifar10.load_data()

# Preprocess the data
X_train, X_test = X_train / 255.0, X_test / 255.0
y_train = (y_train == 3).astype(int)
y_test = (y_test == 3).astype(int)

# Define the CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

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

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32,
validation_data=(X_test, y_test))

# Evaluate the model
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print("Test Accuracy:", test_accuracy)
```