A feed-forward neural network is a fundamental type of artificial neural network where connections between nodes do not form cycles. In essence, information flows in one direction, forward, from the input nodes through hidden layers to the output nodes. Each node, or neuron, in the network is connected to every node in the subsequent layer, but not to nodes in the previous layers or to itself.

The basic structure of a feed-forward neural network consists of an input layer, one or more hidden layers, and an output layer. The input layer receives the initial data, which could represent features of the input data in a machine learning problem. Each neuron in the input layer corresponds to one feature.

The hidden layers process the input data through a series of weighted connections and apply activation functions to the weighted sum of inputs to introduce non-linearity into the model. These hidden layers allow the network to learn complex patterns and relationships within the data.

The output layer produces the final output of the network, which could be a classification, regression, or any other kind of prediction depending on the problem being solved. The number of neurons in the output layer depends on the nature of the problem; for instance, in a binary classification problem, there would be one neuron for each class, while in a regression problem, there would be one neuron for each continuous output variable.

During training, the network learns the optimal weights for each connection between neurons by iteratively adjusting them to minimize a predefined loss function. This is typically done using optimization algorithms like gradient descent, which update the weights based on the gradient of the loss function with respect to the weights.

Feed-forward neural networks have been widely used in various fields, including computer vision, natural language processing, speech recognition, and financial forecasting, among others. They form the basis for more complex neural network architectures such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), which are designed to handle specific types of data or tasks.

CODE:

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import cifar10
from sklearn.model_selection import train_test_split
import numpy as np

# Load CIFAR-10 dataset
(train_images, train_labels), (_, _) = cifar10.load_data()

# Filter out normal (automobile) and malware images
normal_images = train_images[train_labels.flatten() == 1]
malware_images = train_images[train_labels.flatten() != 1]

# Label normal images as 0 and malware images as 1
normal_labels = np.zeros(len(normal_images))
malware_labels = np.ones(len(malware_images))

# Concatenate normal and malware images and labels
X = np.concatenate([normal_images, malware_images])
y = np.concatenate([normal_labels, malware_labels])

# 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)

# Reshape and normalize the data
X_train = X_train.reshape((-1, 32 * 32 * 3)) / 255.0
X_test = X_test.reshape((-1, 32 * 32 * 3)) / 255.0

# Model Architecture
model = models.Sequential([
    layers.Dense(512, activation='relu', input_shape=(32 * 32 * 3,)),
    layers.Dropout(0.2),
    layers.Dense(256, activation='relu'),
    layers.Dropout(0.2),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.2),
    layers.Dense(1, activation='sigmoid')
])

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

# Training
model.fit(X_train, y_train, epochs=10, batch_size=64, validation_split=0.2)

# Evaluation
test_loss, test_acc = model.evaluate(X_test, y_test)
print('Test accuracy:', test_acc)
```