Convolutional Neural Network

A convolutional neural network (CNN) is a type of deep neural network that is well-suited for processing data with a grid-like topology, such as images, video, or audio spectrograms. CNNs have been successful in areas such as image recognition, object detection, and natural language processing.

The key architectural components of a CNN are:

Convolutional layers:

- These layers apply a set of learnable filters (kernels) to the input data, performing a mathematical operation known as convolution.
- Each filter convolves across the input, computing dot products between the filter weights and the corresponding input values, producing a feature map.
- Multiple filters are applied in parallel, allowing the network to extract different features from the same input.

Pooling layers:

- Pooling layers are inserted between successive convolutional layers to progressively reduce the spatial dimensions (height and width) of the feature maps.
- Common pooling operations include max pooling (taking the maximum value in a local neighborhood) and average pooling (taking the average value in a local neighborhood).
- Pooling helps to reduce computational complexity, introduces translation invariance, and enables the network to focus on the most prominent features.

Activation functions:

- Activation functions, such as ReLU (Rectified Linear Unit), introduce non-linearity into the network, allowing it to learn complex patterns in the data.
- ReLU is a commonly used activation function in CNNs due to its computational efficiency and ability to mitigate the vanishing gradient problem.

Fully connected layers:

- After the convolutional and pooling layers, the feature maps are flattened into a one-dimensional vector, which is then fed into one or more fully connected layers.
- Fully connected layers are similar to those in traditional feedforward neural networks, where each neuron is connected to all neurons in the previous layer.
- These layers are responsible for combining the extracted features and producing the final output, such as classification probabilities or regression values.

The overall architecture of a CNN typically consists of multiple stages, each containing a combination of convolutional, pooling, and activation layers. The number of stages and layers within each stage can vary depending on the complexity of the task and the size of the input data.

CNNs learn to detect low-level features, such as edges and shapes, in the early layers, and then combine these features into higher-level representations in the subsequent layers. This hierarchical structure allows CNNs to effectively capture and utilize the spatial and temporal relationships present in the input data.

david_dataset2:

Dataset is downloaded from: https://www.kaggle.com/datasets/xwolf12/malicious-and-benign-websites

Data set is modified (some columns and rows are removed and renamed)

URL_LENGTH	NUMBER_OF_CHARACTERS	TCP_EXCHANGE	REMOTE_TCP_PORT	REMOTE_IPS	APP_BYTES	SOURCE_APP_PACKETS	REMOTE_APP_PACKETS	Type
16	7	7	0	2	700	9	10	1
16	6	17	7	4	1230	17	19	0
16	6	0	0	0	0	0	0	0
17	6	31	22	3	3812	39	37	0
17	6	57	2	5	4278	61	62	0
18	7	11	6	9	894	11	13	0
18	6	12	0	3	1189	14	13	0
19	6	0	0	0	0	0	0	0
20	5	0	0	0	0	2	3	1

Python code:

```
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, roc_curve
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
from sklearn.utils import shuffle
from sklearn.metrics import accuracy_score
from tensorflow.keras.layers import Dense, Conv1D, MaxPooling1D, Flatten

# Load the dataset
data = pd.read_csv("david_dataset2.csv")

# Preprocessing
X = data.drop(columns=['Type']).values
y = data['Type'].values

# Normalizing features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Reshape data for CNN
X = X.reshape(X.shape[0], X.shape[1], 1)

# Splitting the dataset 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)

# Define the CNN model
model = Sequential([
    Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], 1)),
    MaxPooling1D(pool_size=2),
    Flatten(),
    Dense(64, activation='relu'),
    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, verbose=1)

# Evaluate the model
accuracy = model.evaluate(X_test, y_test, verbose=0)[1]
print("Accuracy: {:.2f}%".format(accuracy * 100))

#Evaluate model
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test Loss: {loss}, Test Accuracy: {accuracy}')
```

Results:

Test Loss: 0.3647598326206207, Test Accuracy: 0.8700000047683716