Task 2. Convolutional Neural Network (CNN):

A Convolutional Neural Network (CNN) is a type of deep neural network primarily used for analyzing visual imagery. CNNs are particularly adept at capturing spatial hierarchies and patterns in data, making them highly effective in tasks such as image recognition, object detection, and image classification.

Components of a CNN:

1. **Convolutional Layers**: These layers apply convolution operations to the input data, using learnable filters or kernels. Each filter extracts specific features from the input, such as edges or textures, by sliding across the input data and performing element-wise multiplications followed by summations. This process helps in capturing spatial hierarchies and patterns.
2. **Pooling Layers**: Pooling layers downsample the feature maps generated by convolutional layers, reducing their spatial dimensions. This helps in decreasing the computational complexity of the network and making the learned features more robust to variations in input.
3. **Activation Functions**: Activation functions introduce non-linearity into the network, allowing it to learn complex relationships in the data. Common activation functions used in CNNs include ReLU (Rectified Linear Unit), Sigmoid, and Tanh.
4. **Fully Connected Layers**: These layers, often found towards the end of the network, connect every neuron in one layer to every neuron in the next layer. They perform classification based on the high-level features extracted by earlier layers.
5. **Normalization Layers**: Normalization layers, such as Batch Normalization, are used to improve the training speed and stability of the network by normalizing the input data.

Training a CNN:

Training a CNN involves feeding labeled training data into the network and adjusting the parameters (weights and biases) through backpropagation and optimization algorithms (e.g., gradient descent) to minimize a predefined loss function. The goal is to make the network's predictions as close as possible to the ground truth labels.

Practical Example in Cybersecurity with Python Code:

Application: Anomaly Detection in Network Traffic

Data: We'll use a dataset containing network traffic data, where each record consists of various features such as source IP, destination IP, protocol type, etc. The goal is to detect anomalies or suspicious activities in the network.

Python Code:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from keras.models import Sequential
from keras.layers import Conv1D, MaxPooling1D, Flatten, Dense

# Load dataset
data = pd.read_csv('network_traffic.csv')

# One-hot encode categorical features
data_encoded = pd.get_dummies(data, columns=['source_ip', 'destination_ip',
'protocol'])

# Separate features and target variable
X = data_encoded.drop('label', axis=1)
y = data_encoded['label']

# Split 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)

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Reshape data for Conv1D layer
X_train_reshaped = X_train_scaled.reshape(X_train_scaled.shape[0],
X_train_scaled.shape[1], 1)
X_test_reshaped = X_test_scaled.reshape(X_test_scaled.shape[0],
X_test_scaled.shape[1], 1)

# Build CNN model
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=3, activation='relu',
input_shape=(X_train_reshaped.shape[1], 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

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

# Train model
model.fit(X_train_reshaped, y_train, epochs=10, batch_size=32,
validation_data=(X_test_reshaped, y_test))

# Evaluate model
```

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

Below is the example of the data used:

source_ip	destination_ip	protocol	packet_size	label
10.0.0.1	172.16.0.2	ICMP	1261	0
10.0.0.1	192.168.1.2	UDP	582	1
10.0.0.1	192.168.1.2	ICMP	1124	1
172.16.0.1	10.0.0.2	UDP	411	0
10.0.0.1	192.168.1.2	ICMP	67	1
192.168.1.1	172.16.0.2	ICMP	1397	0
10.0.0.1	172.16.0.2	TCP	726	1
10.0.0.1	10.0.0.2	TCP	91	1
192.168.1.1	10.0.0.2	ICMP	944	0
172.16.0.1	10.0.0.2	TCP	392	1
10.0.0.1	10.0.0.2	ICMP	229	1
172.16.0.1	10.0.0.2	UDP	1275	1
192.168.1.1	10.0.0.2	TCP	988	1
192.168.1.1	192.168.1.2	ICMP	820	1
10.0.0.1	172.16.0.2	UDP	614	1
172.16.0.1	172.16.0.2	TCP	266	1
192.168.1.1	10.0.0.2	UDP	953	0
192.168.1.1	10.0.0.2	TCP	1045	1
10.0.0.1	172.16.0.2	TCP	1289	1