Final

Task 2

Convolutional Neural Network

(CNN) is a specialized type of artificial neural network designed primarily for analyzing structured grid data, such as images. Unlike traditional neural networks, CNNs are particularly adept at recognizing patterns and spatial hierarchies within data through their unique architecture. The core building blocks of a CNN are convolutional layers, pooling layers, and fully connected layers.

Convolutional layers apply a series of filters to the input data, performing a convolution operation that slides the filter across the input to produce feature maps. These filters help detect local features such as edges, textures, and shapes in the early layers, and more complex patterns in deeper layers. Pooling layers, often following convolutional layers, reduce the spatial dimensions of the feature maps through operations like max pooling or average pooling. This down-sampling helps to decrease the computational load and mitigate overfitting by making the detection of features invariant to small translations in the input.

In the final stages, the fully connected layers take the high-level, abstracted features extracted by the convolutional and pooling layers and use them for classification or regression tasks. These layers operate like traditional neural networks, where each neuron is connected to every neuron in the previous layer, allowing the network to make decisions based on the global patterns identified in the earlier layers.

CNNs are renowned for their ability to handle large amounts of data and have become the backbone of many modern computer vision applications, including image and video recognition, object detection, and segmentation. Their hierarchical structure, inspired by the visual cortex of animals, allows them to learn and recognize complex patterns in a way that is both computationally efficient and highly effective. Advances in CNN architectures, such as AlexNet, VGG, ResNet, and more recently, EfficientNet, have significantly pushed the boundaries of performance, making CNNs a cornerstone of deep learning in various domains.

For practical job, I created an example csv file.

C: > Users > user > # dataa.py > csvfile

```python
# dataa
import csv

# Create a list of data with the correct column names and sample values
data = [
    ["File Name", "Network_Exfiltration", "File_Modification"],
    ["sample1", 2, 1],
    ["sample2", 8, 4],
    ["sample3", 7, 6],
]

# Open a CSV file for writing
with open('dataa.csv', 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerows(data)

print("CSV file created successfully!")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\user> & C:/Users/user/anaconda3/python.exe "c:/Users/user/# dataa.py"
CSV file created successfully!
PS C:\Users\user>
```

Processing:

C: > Users > user > # task 2.py > ...

```python
import numpy as np
import pandas as pd
import cv2
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder  # Optional, for future label encoding
from keras.utils import to_categorical  # Optional, for multi-class encoding
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Import the os module
import os

# Function to convert executable to grayscale image
def convert_executable_to_image(file_path, image_size=64):
  with open(file_path, 'rb') as f:
    # Read the first N bytes of the executable (adjust N as needed)
    byte_data = f.read(image_size * image_size)
  # Convert byte data to a list of integers
  int_data = [int(x) for x in byte_data]
  # Reshape the data into a 2D array for image representation
  img = np.reshape(int_data, (image_size, image_size))
  # Normalize pixel values (optional)
  img = img / 255.0
  return img

# Path to the dataset
data_dir = './data'
label_file = 'dataa.csv'

# Load labels
labels = pd.read_csv(label_file)

# Initialize lists to hold image data and labels
images = []
y = []

# Process each file in the dataset
for index, row in labels.iterrows():
  file_name = row['File Name']
  label = row['Network_Exfiltration']
  file_path = os.path.join(data_dir, f'{file_name}.exe')
  if not os.path.exists(os.path.join(data_dir, f'{file_name}.exe')):
    print(f"Warning: File {file_path} not found, skipping")
    continue
  img = convert_executable_to_image(file_path)
  images.append(img)
  y.append(label)

# Convert lists to numpy arrays
X = np.array(images)
y = np.array(y)

# Normalize pixel values (already done in convert_executable_to_image)
X = X / 255.0
```

```
57    encoder = LabelEncoder()
58    y_encoded = encoder.fit_transform(y)
59    y_categorical = to_categorical(y_encoded)
60
61    # Split the data (replace with your desired test size)
62    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
63
64    # Build the CNN model (replace with your desired architecture)
65    model = Sequential()
66    model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(64, 64, 1)))
67    model.add(MaxPooling2D(pool_size=(2, 2)))
68    model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
69    model.add(MaxPooling2D(pool_size=(2, 2)))
70    model.add(Flatten())
71    model.add(Dense(128, activation='relu'))
72    model.add(Dense(1, activation='sigmoid'))
73
74    # Compile the model (replace with your desired optimizer and loss function)
75    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
76
77    # Train the model (replace with your desired number of epochs)
78    model.fit(X_train, y_train, epochs=10, batch_size=64, validation_split=0.2)
79
80    # Evaluate the model
81    loss, accuracy = model.evaluate(X_test, y_test)
82    print(f'Test Accuracy: {accuracy:.4f}')
83
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

```
PS C:\Users\user> & C:/Users/user/anaconda3/python.exe "c:/Users/user/# task 2.py"
2024-05-16 02:03:13.374333: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see
_ENABLE_ONEDNN_OPTS=0`.
2024-05-16 02:03:15.834964: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see
_ENABLE_ONEDNN_OPTS=0`.
Warning: File ./data\sample1.exe not found, skipping
Warning: File ./data\sample2.exe not found, skipping
Warning: File ./data\sample3.exe not found, skipping
Traceback (most recent call last):
  File "c:\Users\user\# task 2.py", line 59, in <module>
    y_categorical = to_categorical(y_encoded)
                    ^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\user\anaconda3\Lib\site-packages\keras\src\utils\numerical_utils.py", line 86, in to_categor
    num_classes = np.max(x) + 1
                  ^^^^^^^^^
  File "C:\Users\user\anaconda3\Lib\site-packages\numpy\core\fromnumeric.py", line 2810, in max
    return _wrapreduction(a, np.maximum, 'max', axis, None, out,
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\user\anaconda3\Lib\site-packages\numpy\core\fromnumeric.py", line 88, in _wrapreduction
    return ufunc.reduce(obj, axis, dtype, out, **passkwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
ValueError: zero-size array to reduction operation maximum which has no identity
PS C:\Users\user> 
```