Task 2

Identifying malware using Convolutional Neural Networks (CNNs) involves several steps, from preprocessing the data to training the model and making predictions. Here's a comprehensive description of how CNNs can be used for this task, along with code snippets for each step:

1. Data Preparation: Collect a dataset of malware samples along with their corresponding labels (e.g., whether it's a virus, trojan, etc.). Preprocess the data, which may involve converting the binary data into images or extracting specific features. Split the dataset into training, validation, and testing sets.

2. Model Architecture: Design a CNN architecture suitable for malware classification. This typically includes several convolutional layers followed by pooling layers and dense layers. CNNs can automatically learn features from the input data, making them suitable for analyzing binary files or images representing malware.

3. Model Training: Compile the CNN model with an appropriate loss function and optimizer. Train the model on the training data, adjusting the weights of the network to minimize the loss. Validate the model on the validation set to monitor its performance and prevent overfitting.

4. Model Evaluation: Evaluate the trained model on the test set to assess its performance metrics such as accuracy, precision, recall, and F1score. Analyze any misclassifications to understand the model's strengths and weaknesses.

5. Deployment: Once satisfied with the model's performance, deploy it for realworld use. The deployed model can be used to classify new malware samples automatically.

Here's an example code snippet demonstrating the process:

```
import numpy as np
import os
from PIL import Image
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense

# Step 1: Data Preparation
data_dir = "example_dataset"
class_names = os.listdir(data_dir)
num_classes = len(class_names)

# Load and preprocess images
X = []
y = []
```

```
img_size = (32, 32)
for class_index, class_name in enumerate(class_names):
    class_dir = os.path.join(data_dir, class_name)
    for image_name in os.listdir(class_dir):
        image_path = os.path.join(class_dir, image_name)
        image = Image.open(image_path).resize(img_size)
        image_array = np.array(image) / 255.0  # Normalize pixel
values
        X.append(image_array)
        y.append(class_index)

X = np.array(X)
y = np.array(y)

# 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)

# Step 2: Model Architecture
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(img_size[0],
img_size[1], 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(num_classes, activation='softmax')
])

# Step 3: Model Training
model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, batch_size=32,
validation_split=0.1)

# Step 4: Model Evaluation
test_loss, test_acc = model.evaluate(X_test, y_test)
print('Test accuracy:', test_acc)

# Step 5: Deployment
# Save the trained model for future use
model.save('malware_detection_model.h5')
```

In this example, a basic CNN architecture is defined using Keras, and the model is trained and evaluated on the prepared dataset. Finally, the trained model is saved for deployment.

Please use the "Kernel>Restart & Run All" command in Jupyter Notebook and check your results before submitting your homework. Note that I rerun all boxes on my side before grading.