Technical Career Education Private Limited

5th floor, Sahyadri Campus, Adyar, Mangalore 575007

Innoventure Internship

PROJECT REPORT

2022 - 23

Project Title: Malware Detection of Images using CNN Architecture

Submitted by:

Arjun Shetty	4SF21CY011
Sushanth	4SF21CY050
Manish J Rai	4SF21IS043
Aarya A Shetty	4SF21CY001

Institution:

Sahyadri College of Engineering and Management

Adyar Mangalore 575007

CONTENTS

Project Overview

Project Overview

Problem Statement	Malware Detection of Images using CNN Architecture	
Class/section	4-CY,4-IS	
Team Name	BLUE TEAM	
Team Members	Name	USN
	Arjun Shetty	4SF21CY011
	Sushanth	4SF21CY050
	Manish J Rai	4SF21IS043
	Aarya A Shetty	4SF21CY001

1. Introduction

Malware detection is the process of identifying and preventing malicious software, often referred to as malware, from infiltrating and causing harm to computer systems, networks, and devices.

Malware encompasses a wide range of malicious software types, including viruses, worms, trojans, ransomware, spyware, adware, and more. Detecting malware is a crucial aspect of cybersecurity to safeguard digital assets, data, and privacy.

Malware detection in images is a major cybersecurity challenge in today's digital environment. Criminals are increasingly turning to image-based malware to evade text-based detection. This issue concerns the development and application of convolutional neural network (CNN) architecture to detect and classify malware in images.

As the volume and complexity of image-based malware continues to increase, this work presents significant challenges and security challenges that require robust and accurate CNN models to protect computer systems and networks against this evolving threat.

2. Problem Statement

Malware detection is an important aspect of cybersecurity that aims to identify and mitigate threats from malicious software, commonly known as malware. Malware is a broad category of software designed to infiltrate and disrupt computer systems, networks, and devices in order to cause damage, steal data, disrupt operations, or cause unauthorized access. This includes different types of malwares such as:

1. Virus: A program that can copy itself by linking to legitimate files and infect other programs when run.

2. Worm: Self-replicating malware that spreads independently, usually across a network or network.

3. Trojan horse: A Trojan horse disguises itself as legitimate software and tricks users into installing it, allowing attackers to gain control or steal data.

4. Ransomware: Malware that encrypts the victim's data and demands a ransom to decrypt it.

5. Spyware: Software that hides user data or monitors activity without permission.

6. Adware: Malware that does not serve advertisements, often bundled with other software.

Malware can cause serious damage, including data deletion, financial loss, crashes, and privacy breaches. Therefore, malware detection techniques are essential to protect digital assets, sensitive data, and personal privacy. It involves the use of various techniques and tools to identify and prevent malware entry, such as:

- Signature Detection: This method includes signature files and code of known malware files for comparison. If a match is found, the file is marked as malware.
- Heuristic Analysis: This method identifies malicious behavior or code patterns by analyzing the functionality of the software without even knowing the malware.
- Behavioral Analysis: This method examines the behavior of the software during execution and can detect suspicious activity that may indicate malware.
- Machine Learning and Artificial Intelligence: Technologies such as advanced machine learning and artificial intelligence use algorithms to identify malware based on specific or habitual characteristics and patterns. Malware detection

plays an important role in maintaining the security of computers, networks and devices because it helps prevent the spread of malware and the damage it can voluntarily cause. It is a constant challenge for cybersecurity professionals to stay ahead of the evolution of malware threats and develop effective detection systems to combat them.

3. Solution

To solve the problem of detecting malware in images using the Convolutional Neural Network (CNN) architecture in Google Colab, you can follow these steps:

1. Data storage: Save image files containing clear patterns. and malware detection. You can use publicly available information or create your own.

2. Data Preprocessing: Preprocessing images; mainly resizing, normalizing and optimizing data. Data augmentation helps increase the power of the model by creating changes in the image.

3. Import libraries: Import Python libraries and appropriate modules. You can easily access popular libraries such as TensorFlow and Keras on Google Colab.

4. Design CNN Model: Design the CNN architecture for image classification. This process requires a convolutional layer, a layer of layers, and a full set of layers.

5. Compile the model: Define loss functions, optimizer and evaluation metrics. For binary classification (clean vs. malware), you can use binary cross-entropy as a loss function.

6. Model training: Enter initial data into the model and train it. To test your model's performance, you may need to separate your data into training and testing.

4. Conclusion/Outcome

malware detection of images using CNN architecture

mounting drive, can be done both using the code or from the files tab in the side

In [ ]:
```
# from google.colab import drive
# drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=Tru
e).
```

Install the required packages and check if the requirements are satisfied

In [2]:
```
!pip install tensorflow
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import os
import random
import shutil
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
from tensorflow.keras.preprocessing import image
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.12.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (23.5.26)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.4.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.59.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.9.0)
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0->goog
le-auth-oauthlib<0.5,>=0.4.1->tensorboard<2.13,>=2.12->tensorflow) (3.2.2)
```

we split the dataset of benign and malware images into test, train directories in (0.8:0.2 ratio)

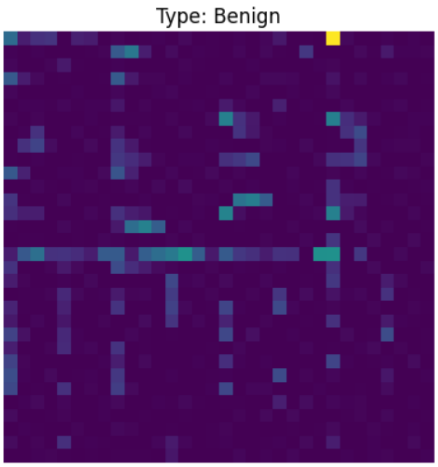
In [4]:
```
import os
import random
import shutil

# Set the source directory containing both benign and malware subdirectories
source_dir = "/content/drive/MyDrive/archive/imagery"

# Set the destination directories for the training and test sets
train_dir = "/content/drive/MyDrive/archive/train"
test_dir = "/content/drive/MyDrive/archive/test"

# Set the ratio of images to be used for training (e.g., 80% for training, 20% for testing)
train_ratio = 0.8

# Create the destination directories if they don't exist
os.makedirs(train_dir, exist_ok=True)
os.makedirs(test_dir, exist_ok=True)

# Loop through the subdirectories (benign and malware)
for subdir in os.listdir(source_dir):
    subdir_path = os.path.join(source_dir, subdir)
    if os.path.isdir(subdir_path) and not subdir.startswith("."):
        # List all image files in the subdirectory
        image_files = os.listdir(subdir_path)

        # Calculate the number of images for training and testing
        num_images = len(image_files)
        num_train_images = int(num_images * train_ratio)
        num_test_images = num_images - num_train_images
```

```
        # Randomly shuffle the list of image files
        random.shuffle(image_files)

        # Split images into the training directory
        for image_file in image_files[:num_train_images]:
            source_path = os.path.join(subdir_path, image_file)
            if not os.path.isdir(source_path):  # Skip directories
                dest_path = os.path.join(train_dir, subdir, image_file)
                os.makedirs(os.path.join(train_dir, subdir), exist_ok=True)
                shutil.copy(source_path, dest_path)

        # Split images into the test directory
        for image_file in image_files[num_train_images:]:
            source_path = os.path.join(subdir_path, image_file)
            if not os.path.isdir(source_path):  # Skip directories
                dest_path = os.path.join(test_dir, subdir, image_file)
                os.makedirs(os.path.join(test_dir, subdir), exist_ok=True)
                shutil.copy(source_path, dest_path)

        print(f"Split {num_train_images} images for training and {num_test_images} images for testing in {subdir}.")

print("Images have been split into training and test sets for benign and malware.")
```

Display a random generated image from the train directory along with the type of image

In [5]:

```
import os
import random
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

# Set the path to the training directory
train_dir = "/content/drive/MyDrive/archive/train"

# List the subdirectories in the training directory (should include benign and malware)
subdirectories = os.listdir(train_dir)

# Create a list to store images and their corresponding types
images = []

# Randomly select a subdirectory
selected_subdir = random.choice(subdirectories)

# Set the type based on the selected subdirectory (benign or malware)
if "benign" in selected_subdir:
    img_type = "Benign"
else:
    img_type = "Malware"

# Get a list of image files in the selected subdirectory
image_files = os.listdir(os.path.join(train_dir, selected_subdir))

# Randomly select an image
selected_image = random.choice(image_files)

# Load and display the selected image
img = mpimg.imread(os.path.join(train_dir, selected_subdir, selected_image))
plt.imshow(img)
plt.title(f"Type: {img_type}")
plt.axis('off')
plt.show()
```

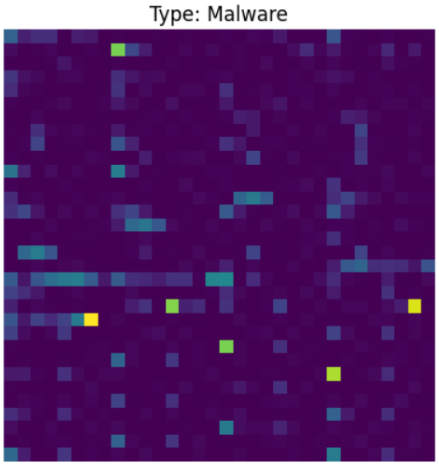


display a random generated image from the test directory along with the type of image (optional)

In [6]:

```
import os
import random
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
# Set the path to the test directory
test_dir = "/content/drive/MyDrive/archive/test"

# List the subdirectories in the test directory (should include benign and malware)
subdirectories = os.listdir(test_dir)

# Create a list to store images and their corresponding types
images = []

# Randomly select a subdirectory
selected_subdir = random.choice(subdirectories)

# Set the type based on the selected subdirectory (benign or malware)
if "benign" in selected_subdir:
    img_type = "Benign"
else:
    img_type = "Malware"

# Get a list of image files in the selected subdirectory
image_files = os.listdir(os.path.join(test_dir, selected_subdir))

# Randomly select an image
selected_image = random.choice(image_files)

# Load and display the selected image
img = mpimg.imread(os.path.join(test_dir, selected_subdir, selected_image))
plt.imshow(img)
plt.title(f"Type: {img_type}")
plt.axis('off')
plt.show()
```

Type: Malware



Perform data augmentation to label the train(mainly) and test directories in 2 classes each.This is needed to perform the CNN architecture

In [7]:

```
# Set the paths to your dataset directories
train_dir = "/content/drive/MyDrive/archive/train"
test_dir = "/content/drive/MyDrive/archive/test"

# Define image size and batch size
img_size = (150, 150)
batch_size = 32

# Data augmentation for the training set
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=15,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Data augmentation for the test set
test_datagen = ImageDataGenerator(rescale=1./255)

# Create data generators for training and testing
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=img_size,
    batch_size=batch_size,
    class_mode='binary'
)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=img_size,
    batch_size=batch_size,
    class_mode='binary'
)
```

```
Found 3050 images belonging to 2 classes.
Found 763 images belonging to 2 classes.
```

Compile the CNN architecture model

In [9]:
```
# Define and compile the model
model = keras.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(img_size[0], img_size[1], 3)),
    layers.MaxPooling2D(2, 2),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D(2, 2),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D(2, 2),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(1, activation='sigmoid')
])

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

Train the model using the directory to get accuracy and loss in each epoch cycle

In [12]:
```
# Train the model
history = model.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=5,  # Adjust the number of epochs as needed
    validation_data=test_generator,
    validation_steps=len(test_generator)
)
```

```
Epoch 1/5
96/96 [==============================] - 230s 2s/step - loss: 0.0097 - accuracy: 0.9957 - val_loss: 0.0036 - val_accuracy: 0.9
974
Epoch 2/5
96/96 [==============================] - 222s 2s/step - loss: 0.0057 - accuracy: 0.9984 - val_loss: 0.0176 - val_accuracy: 0.9
948
Epoch 3/5
96/96 [==============================] - 212s 2s/step - loss: 0.0057 - accuracy: 0.9987 - val_loss: 0.0048 - val_accuracy: 0.9
987
Epoch 4/5
96/96 [==============================] - 199s 2s/step - loss: 0.0083 - accuracy: 0.9980 - val_loss: 0.0019 - val_accuracy: 0.9
987
Epoch 5/5
96/96 [==============================] - 201s 2s/step - loss: 0.0027 - accuracy: 0.9997 - val_loss: 0.0035 - val_accuracy: 0.9
987
```

save the model for convienence

In [14]:
```
model.save('my_model2.h5')  # Save the model to a file
```

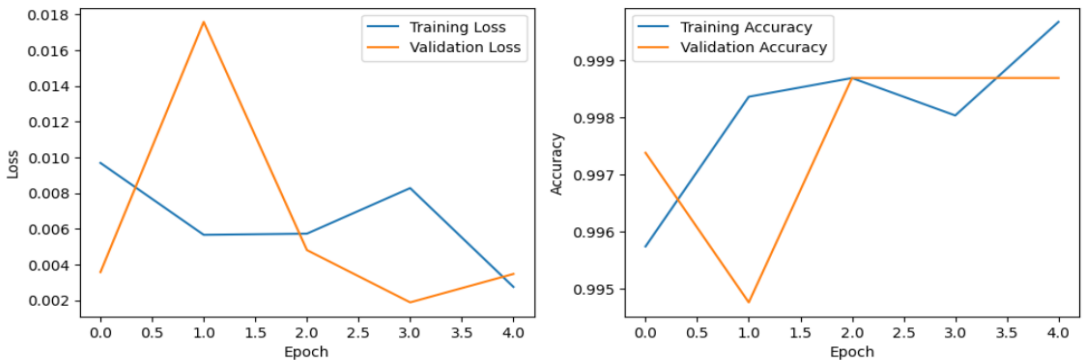
Evaluate the model using the test directory and generate the test accuracy depending on the dataset..

In [16]:
```
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(test_generator)
print(f"Test accuracy: {test_accuracy * 100:.2f}%")
```

```
24/24 [==============================] - 21s 842ms/step - loss: 0.0035 - accuracy: 0.9987
Test accuracy: 99.87%
```

Visualize the training history

In [15]:
```
import matplotlib.pyplot as plt

# Access the training history from the model.fit() call
training_loss = history.history['loss']
training_accuracy = history.history['accuracy']
validation_loss = history.history['val_loss']
validation_accuracy = history.history['val_accuracy']

# Plot training and validation loss
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(training_loss, label='Training Loss')
plt.plot(validation_loss, label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

# Plot training and validation accuracy
plt.subplot(1, 2, 2)
plt.plot(training_accuracy, label='Training Accuracy')
plt.plot(validation_accuracy, label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

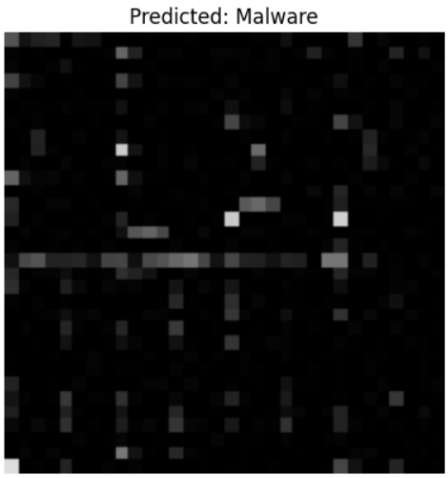


Inference with the Trained Model by using a image of a file that may be malware or benign and predict the output

In [21]:

```
import numpy as np
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt

# Load an image for inference (replace 'image_path' with the actual image path)
image_path = '/content/drive/MyDrive/test_image/malware image (1).jpg'
img = image.load_img(image_path, target_size=img_size)
img = image.img_to_array(img)
img = np.expand_dims(img, axis=0)
img = img / 255.0  # Normalize the image

# Make a prediction
prediction = model.predict(img)

# Interpret the prediction
if prediction[0][0] > 0.5:
    classification = "Malware"
else:
    classification = "Benign"

# Display the image and prediction
plt.imshow(img[0])
plt.title(f'Predicted: {classification}')
plt.axis('off')
plt.show()
```

```
1/1 [==============================] - 0s 111ms/step
```



the above image appears to be blurry due to resolution, resizing and the usage of small size dataset for drive

In [ ]:

5. References

(Links to external sites, documents, images, videos referred)

Kaggle:
https://www.kaggle.com/datasets/datamunge/iot-firmware-image-classification

Google Colab:
https://colab.research.google.com/drive/1sEWtc9_95xImuXWAEiQ7YLpFzvDcLopB?usp=sharing

s