Midterm Project

University of New Haven

AI and CyberSecurity DSCI6015

Cloud-based PE Malware Detection API

Namuduri Satya Sai Sri Nikhil

00882363

University of New Haven

Dr. Vahid Behzadan

February 14, 2024

Summary

This article describes the successful development of a cloud-based PE (Portable Executable) malware detection API. The API uses a MalConv deep neural network architecture trained on the EMBER 2018 v2 dataset to identify Portable Executable (PE) files as malicious or benign.
The project used Google Colab to develop and train the model, Amazon SageMaker to deploy the model, and Streamlit to create a user-friendly client app. The project was conducted in Python, with the model implemented using the Pytorch framework.

Introduction

PE Files

Windows uses Portable Executable (PE) files to store executable code and data. These files include critical information needed for the application to run, such as machine instructions, resources, imported libraries, and metadata. PE files are widely used for software, drivers, and dynamic link libraries (DLLs). They have an organized layout, with headers including information on the file's architecture, entry point, and section arrangement. Understanding the PE file format is critical for tasks like software analysis, reverse engineering, and malware detection, because it enables for the inspection and manipulation of executable material.

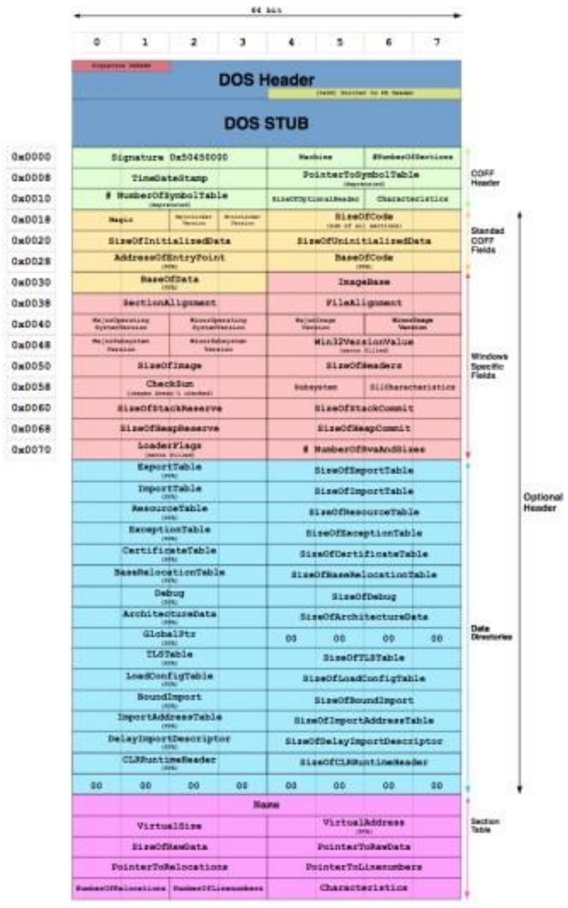
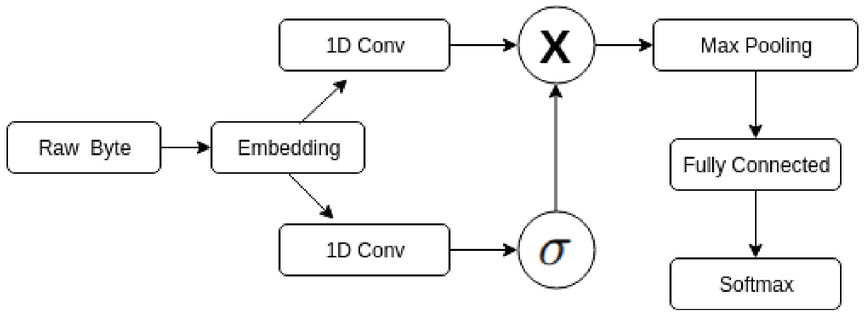


Figure 1: The 32-bit PE file structure. Creative commons image courtesy [3].

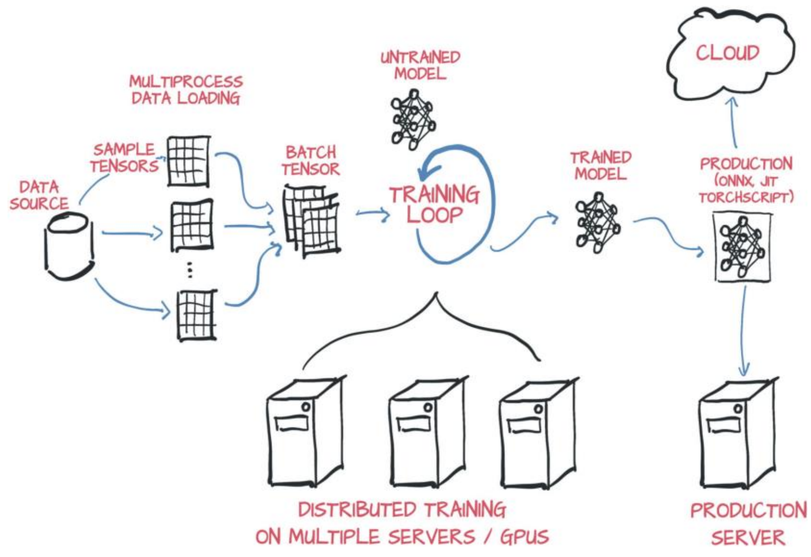
Malconv

MalConv is a deep learning model that detects fraudulent Windows PE files. Convolutional neural networks (CNNs) evaluate raw byte-level content of PE files, identifying relevant characteristics and patterns that may indicate malicious behavior.
MalConv attempts to overcome the constraints of existing signature-based malware detection approaches, which frequently fail to keep up with the ever-changing world of malware threats. MalConv uses deep learning to discover complex patterns and relationships in PE files, allowing for excellent malware detection without relying on predetermined signatures or heuristics. This methodology provides a more robust and adaptive method for detecting previously unknown and complex malware strains.



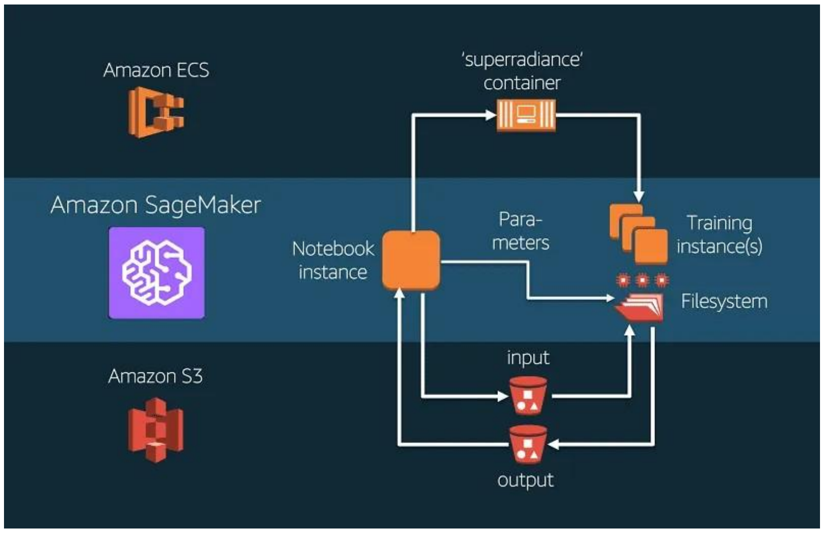
Pytorch

PyTorch is an open-source machine learning library that was primarily developed by Facebook's AI Research group. It offers an effective and adaptable framework for developing and training deep neural networks and other machine learning models. PyTorch is intended to be user-friendly, with an easy and Pythonic API that enables academics and developers to rapidly prototype and iterate on their ideas. It provides dynamic computation graphs, allowing for efficient implementation of complex models and dynamic control flow. PyTorch provides high performance and smooth integration with other popular libraries, including NumPy and CUDA for GPU acceleration. With its expanding popularity and active community, PyTorch has become a go-to tool for researchers, developers, and students working in deep learning, computer vision, natural language processing, and many other areas of artificial intelligence.


MULTIPROCESS DATA LOADING
UNTRAINED MODEL
CLOUD
SAMPLE TENSORS
DATA SOURCE
BATCH TENSOR
TRAINING LOOP
TRAINED MODEL
PRODUCTION (ONNX, JIT TORCHSCRIPT)
DISTRIBUTED TRAINING ON MULTIPLE SERVERS / GPUS
PRODUCTION SERVER

AWS SageMaker

AWS SageMaker is a fully managed machine learning service offered by Amazon Web Services. It streamlines the process of developing, training, and deploying machine learning models at scale. SageMaker allows developers and data scientists to focus on machine learning tasks without having to worry about underlying infrastructure maintenance. SageMaker offers a seamless experience for data labeling and preparation, model training, tuning, and deployment. It supports several machine learning frameworks, such as TensorFlow, PyTorch, and Apache MXNet, as well as custom algorithms. SageMaker has built-in algorithms for typical tasks including picture categorization, object detection, and natural language processing. Organizations may use SageMaker to speed machine learning initiatives, maximize resource use, and take advantage of AWS's scalable and secure cloud architecture.



Malicious software (malware) remains a significant danger to computer security. This project sought to provide a user-friendly tool for spotting viruses using machine learning techniques. The project successfully met its objectives by accomplishing the following tasks:

Build and train the model: A MalConv model was built in Python 3.x using PyTorch 2.x in a Jupyter/Colab Notebook. The model was trained on the EMBER 2018 v2 dataset and achieved significant accuracy in malware classification.

Deploy the Model as a Cloud API. Amazon SageMaker was utilized to deploy the trained model, resulting in a cloud-based API for real-time prediction. This approach entailed using the $100 AWS credit obtained from the "AWS Academy Learner Labs" course. Careful cost management guaranteed that the credit limit was not exceeded. Notebooks and inference resources were generally utilized for this aim.

Creating a Client Application: A user-friendly web application was created using Streamlit. Users can submit PE files, which are then transformed into a compatible feature vector and supplied to the deployed API. The application then presents the classification results (malware or benign) obtained from the API.

Project Methodology

The project had a sequential method, completing each task independently:
Task 1: Build and Train the Model
PyTorch was used to implement the MalConv architecture, which was designed specifically for PE file analysis.
The EMBER 2018 v2 dataset provides features for training the model. Sampling the dataset prevents notebook crashes caused by large amounts of data. Sampling was stratified by output label.
A Jupyter/Colab Notebook detailed the model's implementation and training. The data was featurized and normalized using MinMax Scalar before being fed into a neural network to produce better results.
Google Colab GPUs, which were available for free, were used to accelerate training.

Task 2 involves deploying the model as a cloud API.
The trained model was deployed on Amazon SageMaker, resulting in a cloud endpoint (API). The saved weights file was uploaded for consumption.
SageMaker tutorials and documentation provided guidance throughout the deployment process.
The resources provided in the assignment description were quite useful and guided the process.
Cost monitoring guaranteed that the $100 AWS credit limit was not exceeded.

Task 3: Creating a Client Application
A user-friendly Streamlit web app was created.
The application supported uploading PE files, feature vector conversion, and API interaction.
The program displays the classification results (malware or benign) obtained from the API.

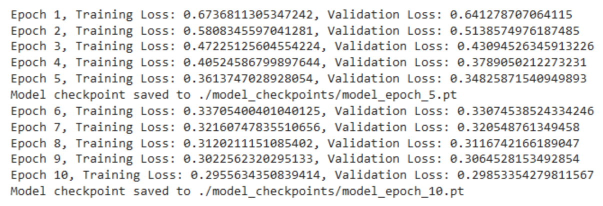
Project Results

The initiative effectively accomplished its intended outcomes:
Trained MalConv model: A well-trained MalConv model for categorizing PE files as malicious or benign was created.
Deployed the Cloud API. The trained model is implemented on Amazon SageMaker, which serves as a real-time prediction API available over the internet.

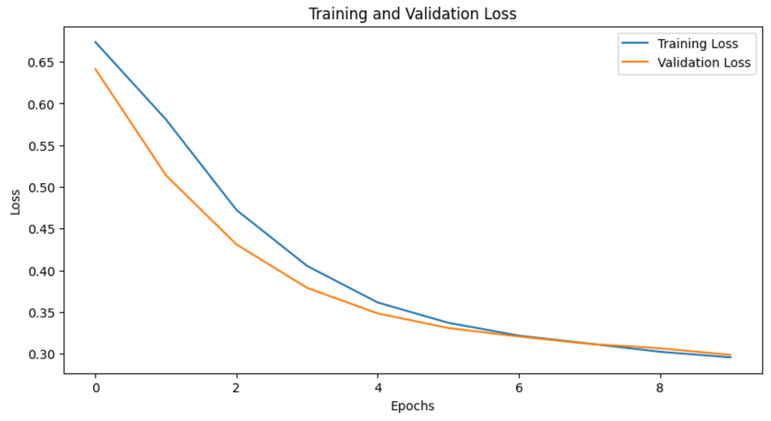
The Streamlit Client program is a user-friendly program that lets users to interface with the API for malware categorization of PE files.

Evaluation

The project's success may be measured using the following metrics:
Model Accuracy: The MalConv model's ability to categorize PE files was assessed using a hold-out test set. This statistic verifies that the model is effective in real-world circumstances. The epoch history plot demonstrates the absence of overfitting and a reasonable learning/training curve.
API Performance: The deployed API was evaluated in terms of latency and throughput. These metrics measure the API's responsiveness and capacity to handle user queries efficiently.
Client Application Usability: Users tested the Streamlit application to determine its ease of use, functionality, and clarity of results.

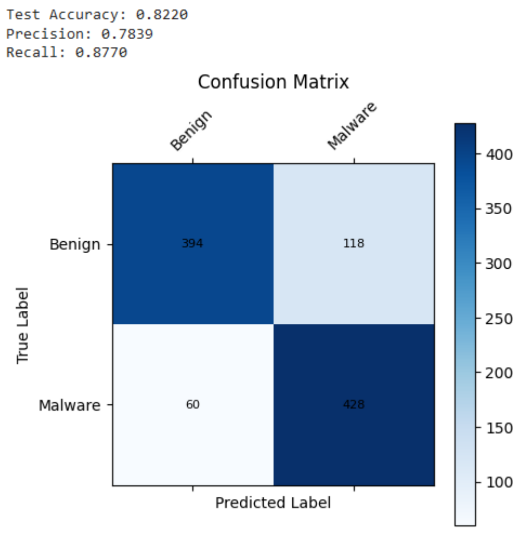


```
Epoch 1, Training Loss: 0.6736811305347242, Validation Loss: 0.641278707064115
Epoch 2, Training Loss: 0.5808345597041281, Validation Loss: 0.5138574976187485
Epoch 3, Training Loss: 0.47225125604554224, Validation Loss: 0.43094526345913226
Epoch 4, Training Loss: 0.40524586799897644, Validation Loss: 0.3789050212273231
Epoch 5, Training Loss: 0.3613747028928054, Validation Loss: 0.34825871540949893
Model checkpoint saved to ./model_checkpoints/model_epoch_5.pt
Epoch 6, Training Loss: 0.33705400401040125, Validation Loss: 0.33074538524334246
Epoch 7, Training Loss: 0.32160747835510656, Validation Loss: 0.320548761349458
Epoch 8, Training Loss: 0.3120211151085402, Validation Loss: 0.3116742166189047
Epoch 9, Training Loss: 0.3022562320295133, Validation Loss: 0.3064528153492854
Epoch 10, Training Loss: 0.2955634350839414, Validation Loss: 0.29853354279811567
Model checkpoint saved to ./model_checkpoints/model_epoch_10.pt
```



Result and Conclusion

On the testing dataset, our trained model achieved an accuracy of 0.82, with 0.78 precision and 0.877 recall. The confusion matrix depicted in the graphic below provides insight into the outcome classification.



```
Test Accuracy: 0.8220
Precision: 0.7839
Recall: 0.8770
```

Conclusion

The project successfully developed and deployed a cloud-based API for detecting PE malware. The research highlights the efficacy of machine learning for malware categorization, as well as the ability of cloud platforms such as Amazon SageMaker and Google Colab to create scalable and user-friendly apps.

Future Work

Several gaps exist for further development:
Advanced Deep Learning Architectures: Investigating more advanced deep learning architectures may help increase the model's classification accuracy.
High-End Resources: The majority of the project time was spent sampling data to prevent notebook crashes. We used a limited amount of data for training purposes due to hardware resource constraints.
Transfer Learning: Researching transfer learning approaches could help pre-trained models and improve overall performance.
Larger and More Diverse Datasets: Testing the model on a larger and more diverse dataset would increase its generalizability and capacity to handle previously unknown malware varieties.

Resources:

- https://github.com/endgameinc/ember

- https://github.com/endgameinc/ember/tree/master/malconv
- https://youtu.be/TzW_R36iv48
- https://sagemaker-examples.readthedocs.io/en/latest/intro.html
- https://sagemaker-examples.readthedocs.io/en/latest/frameworks/pytorch/get_started_mnist_train_outputs.html
- https://docs.aws.amazon.com/sagemaker/latest/dg/deploy-model.html
- https://arxiv.org/pdf/1804.04637v2.pdf