CLOUD-BASED PE MALWARE DETECTION API

MIDTERM PROJECT

DSCI 6015: Artificial Intelligence and Cybersecurity
Professor Vahid Behzadan
Spring 2024

RANDY NGWA TEGUEN
CYBERSECURITY AND NETWORKS, MS.
03/12/2024

Contents

1. Project Overview and Requirements:

This midterm project aimed to demonstrate practical skills in implementing and deploying machine learning models for malware classification. The project focused on building a cloud-based PE (Portable Executable) Malware Detection API using the MalConv architecture. The primary tasks involved were building and training the model to classify PE files as malware or benign, deploying this model as a cloud API, and creating a client application for users to interact with the API.

Before diving into the technicality of the project, let's get to know what the MalConv architecture and Ember models are. MalConv is a deep learning model specifically designed for the purpose of malware detection based on analyzing the raw binary content of executable files. It was introduced by Raff et al. in their paper titled "Malware Detection by Eating a Whole EXE" published in the IEEE European Symposium on Security and Privacy in 2017 [1].

Ember (Endgame Malware BEnchmark for Research) is an open-source framework developed by Anderson et al. at Endgame for training and evaluating machine learning models for malware detection [2]. It provides a dataset containing features extracted from both benign and malicious Windows PE (Portable Executable) files. Ember aims to facilitate research in the field of malware detection by providing a standardized benchmark dataset and feature set.

The project was split into 3 sequential tasks. Task 1 required building and training a deep neural network based on the MalConv architecture to classify PE files as malware or benign. EMBER-2017 v2 was used on the data set in this task. This task was completed Google Collaboratory, a free cloud-based service provided by Google that offers a Jupyter notebook environment accessible via a web browser.

Task 2 involved deploying the model as a cloud API using Amazon Sagemaker. Amazon SageMaker is a fully managed machine learning service provided by Amazon Web Services (AWS) that enables developers and data scientists to build, train, deploy, and manage machine learning models at scale. It offers a comprehensive set of tools and capabilities to streamline the entire machine learning workflow, from data preparation and model training to deployment and monitoring.

Lastly, in Task 3, a web application was created using Streamlit that allowed users to upload PE files, then the application converts it into a feature vector that is compatible with the MalConv/EMBER model, then runs the vector on the cloud API, and then prints the results i.e. malware or benign. Streamlit is an open-source Python library that enables developers to create web applications for machine learning and data science projects quickly and easily. It simplifies the process of building interactive web applications by allowing users to focus on writing Python code to create the application's functionality without needing to know HTML, CSS, or JavaScript. Streamlit provides a simple and intuitive API for creating interactive components such as sliders, buttons, dropdowns, and charts, which can be used to visualize data, perform analysis, or demonstrate machine learning models. Streamlit applications can be run locally or deployed to the web, making it easy to share data analysis results or machine learning models with others. Its ease of use and rapid development capabilities have made it popular among data scientists, machine learning engineers, and developers working on various projects involving data exploration and visualization.

2. Technical Approach for Each Task:

Task 1 - Building and Training the Model:

- The MalConv architecture was implemented using PyTorch, with the training data sourced from the EMBER-2017 v2 dataset.
- Initially, preprocessing and featurization of the data were performed using provided code snippets. Due to limitations in computational resources, the dataset was down sampled to expedite experimentation. The original dataset of 800000 training samples and 200000 test samples was reduced to 50000 samples altogether. This sampling was done using the Pandas Dataframe sample() method.
- Measures were taken to handle runtime errors, GPU utilization, memory overloading, and storage constraints.
- Steps in Task 1 included downloading the pre-processed training and test datasets, mounting a Google Drive, copying the downloaded files to a folder named vMalConv on the mounted drive, downloading and installing lief and ember, getting rid of rows with no labels from the training datasets, then saving this data.
- After sampling the dataset, the sampled data was saved to the mounted Google Drive.
- A MalConv model was created and initialized. This model was then trained for 15 epochs, with a batch size of 64. For this task, the runtime was changed from TPU to T4 GPU.
- After training, the model was evaluated on the test data. Accuracy, Precision, and Recall scores of 0.5003, 0.4006, and 0.6142 were generated respectively.
- **Accuracy**: This is the proportion of correctly classified samples (both true positives and true negatives) out of the total number of samples. The average accuracy obtained was 0.5003, which means that on average, the model correctly classified about 50% of the samples.
- **Precision**: Precision is the proportion of correctly classified positive samples (true positives) out of all samples classified as positive (true positives + false positives). A low precision indicates that the model may have a high false positive rate. The average precision of 0.4006 suggests that the model's ability to correctly classify positive samples is relatively low.

- **Recall**: Recall, also known as sensitivity, is the proportion of correctly classified positive samples (true positives) out of all actual positive samples (true positives + false negatives). A low recall indicates that the model may have a high false negative rate. The average recall of 0.6142 suggests that the model's ability to correctly identify positive samples is moderate.

Task 2 - Deploying the Model as a Cloud API:

- Amazon SageMaker was chosen as the platform for deploying the model as a cloud API.
- The model was packaged and deployed using SageMaker, creating an endpoint for external applications to access.
- An inference script and deployment code were generated for this task.

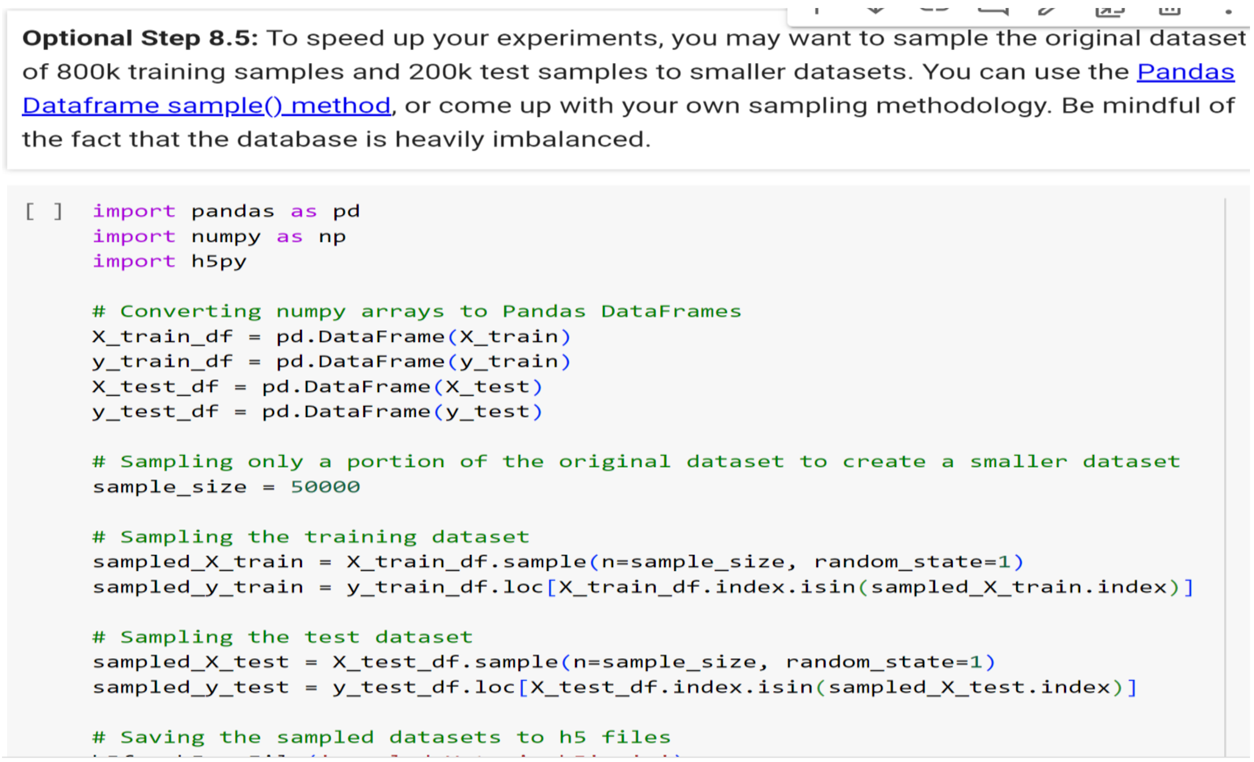
Task 3 - Creating a Client Application:

- A web application was developed using Streamlit, allowing users to upload PE files for analysis.
- Upon file upload, the application was meant to process the file into a compatible feature vector, send it to the deployed API for classification, and display the results (malware or benign) to the user. However, upon execution, this application failed to fetch AWS credentials even after setting them up using aws configuration command on a jupyter terminal. This setback I guess was due to the current role 'LabRole' which didn't allow users to do much on the AWS environment.

3. Conclusion:

The midterm project successfully demonstrated the implementation and deployment of a cloud-based PE Malware Detection API. Despite challenges such as limited computational resources and dataset constraints, the project achieved its objectives of training a machine learning model for malware classification. Further improvements and optimizations can be explored to enhance the model's performance and scalability.

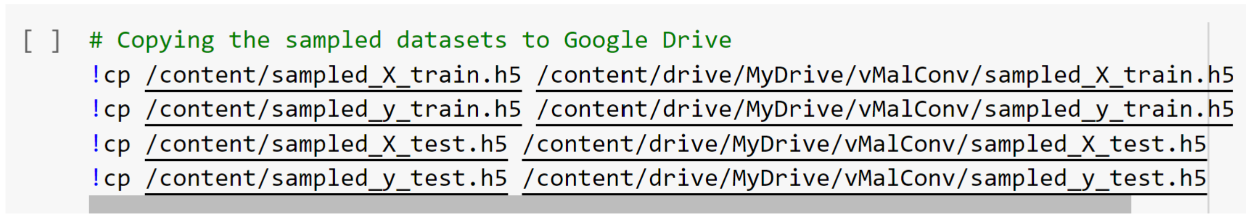
4. Figures:



Optional Step 8.5: To speed up your experiments, you may want to sample the original dataset of 800k training samples and 200k test samples to smaller datasets. You can use the Pandas Dataframe sample() method, or come up with your own sampling methodology. Be mindful of the fact that the database is heavily imbalanced.

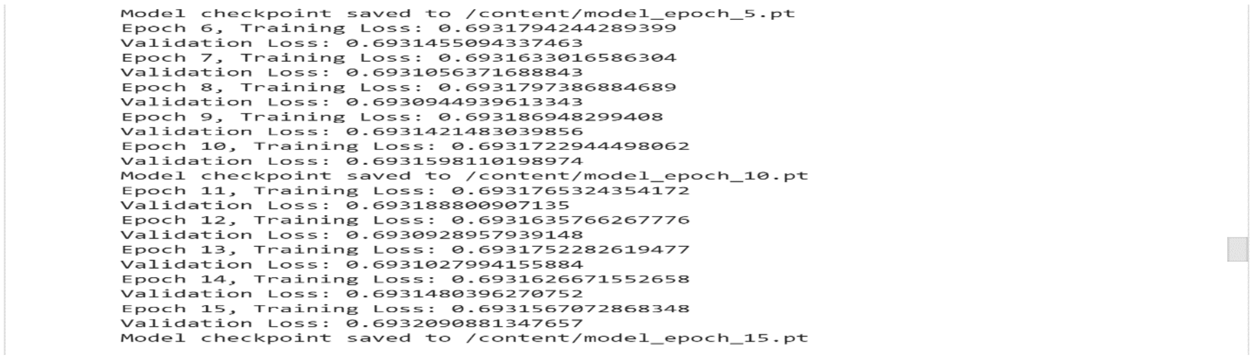
```
[ ]  import pandas as pd
     import numpy as np
     import h5py

     # Converting numpy arrays to Pandas DataFrames
     X_train_df = pd.DataFrame(X_train)
     y_train_df = pd.DataFrame(y_train)
     X_test_df = pd.DataFrame(X_test)
     y_test_df = pd.DataFrame(y_test)

     # Sampling only a portion of the original dataset to create a smaller dataset
     sample_size = 50000

     # Sampling the training dataset
     sampled_X_train = X_train_df.sample(n=sample_size, random_state=1)
     sampled_y_train = y_train_df.loc[X_train_df.index.isin(sampled_X_train.index)]

     # Sampling the test dataset
     sampled_X_test = X_test_df.sample(n=sample_size, random_state=1)
     sampled_y_test = y_test_df.loc[X_test_df.index.isin(sampled_X_test.index)]

     # Saving the sampled datasets to h5 files
```

Figure 1. Code to sample dataset. Full code can be viewed on Midterm Project Colab Redone.ipynb file.



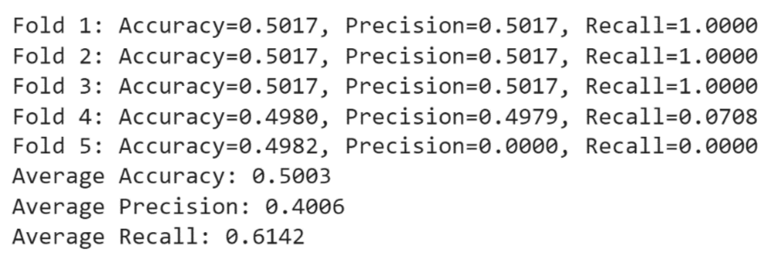
```
[ ]  # Copying the sampled datasets to Google Drive
     !cp /content/sampled_X_train.h5 /content/drive/MyDrive/vMalConv/sampled_X_train.h5
     !cp /content/sampled_y_train.h5 /content/drive/MyDrive/vMalConv/sampled_y_train.h5
     !cp /content/sampled_X_test.h5 /content/drive/MyDrive/vMalConv/sampled_X_test.h5
     !cp /content/sampled_y_test.h5 /content/drive/MyDrive/vMalConv/sampled_y_test.h5
```

Figure 2. Extracted Samples.



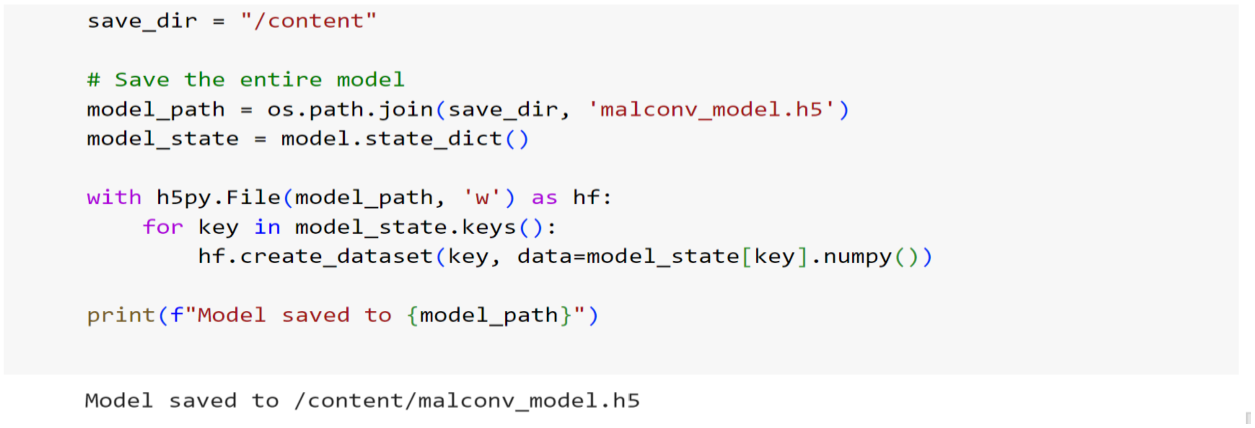
```
Model checkpoint saved to /content/model_epoch_5.pt
Epoch 6, Training Loss: 0.6931794244289399
Validation Loss: 0.6931455094337463
Epoch 7, Training Loss: 0.6931633016586304
Validation Loss: 0.6931056371688843
Epoch 8, Training Loss: 0.6931797386884689
Validation Loss: 0.6930944939613343
Epoch 9, Training Loss: 0.693186948299408
Validation Loss: 0.6931421483039856
Epoch 10, Training Loss: 0.6931722944498062
Validation Loss: 0.6931598110198974
Model checkpoint saved to /content/model_epoch_10.pt
Epoch 11, Training Loss: 0.6931765324354172
Validation Loss: 0.693188800907135
Epoch 12, Training Loss: 0.6931635766267776
Validation Loss: 0.6930928957939148
Epoch 13, Training Loss: 0.6931752282619477
Validation Loss: 0.6931027994155884
Epoch 14, Training Loss: 0.6931626671552658
Validation Loss: 0.6931480396270752
Epoch 15, Training Loss: 0.6931567072868348
Validation Loss: 0.6932090881347657
Model checkpoint saved to /content/model_epoch_15.pt
```

Figure 3. Training the model.



```
Fold 1: Accuracy=0.5017, Precision=0.5017, Recall=1.0000
Fold 2: Accuracy=0.5017, Precision=0.5017, Recall=1.0000
Fold 3: Accuracy=0.5017, Precision=0.5017, Recall=1.0000
Fold 4: Accuracy=0.4980, Precision=0.4979, Recall=0.0708
Fold 5: Accuracy=0.4982, Precision=0.0000, Recall=0.0000
Average Accuracy: 0.5003
Average Precision: 0.4006
Average Recall: 0.6142
```

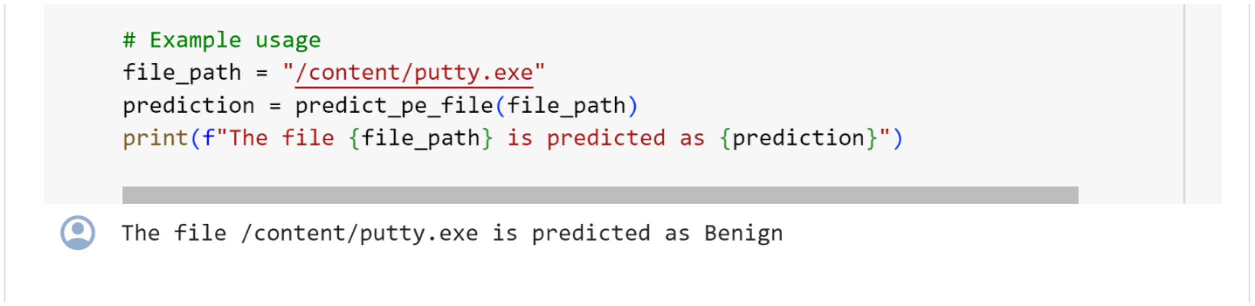
Figure 4. Evaluating the trained model on the test data.



```
save_dir = "/content"

# Save the entire model
model_path = os.path.join(save_dir, 'malconv_model.h5')
model_state = model.state_dict()

with h5py.File(model_path, 'w') as hf:
    for key in model_state.keys():
        hf.create_dataset(key, data=model_state[key].numpy())

print(f"Model saved to {model_path}")
```

```
Model saved to /content/malconv_model.h5
```

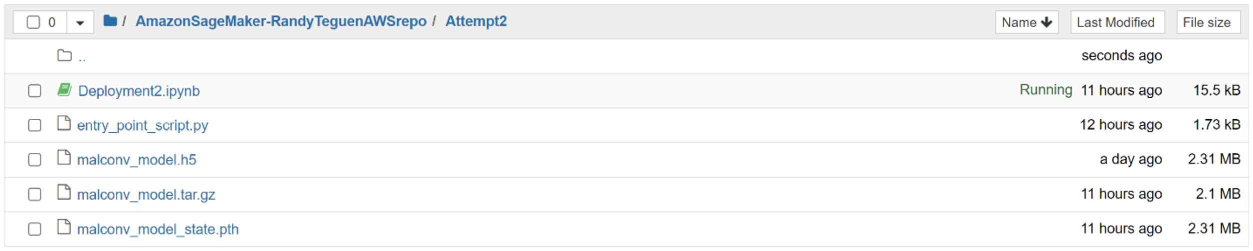
Figure 5. Saved Model.



```
# Example usage
file_path = "/content/putty.exe"
prediction = predict_pe_file(file_path)
print(f"The file {file_path} is predicted as {prediction}")
```

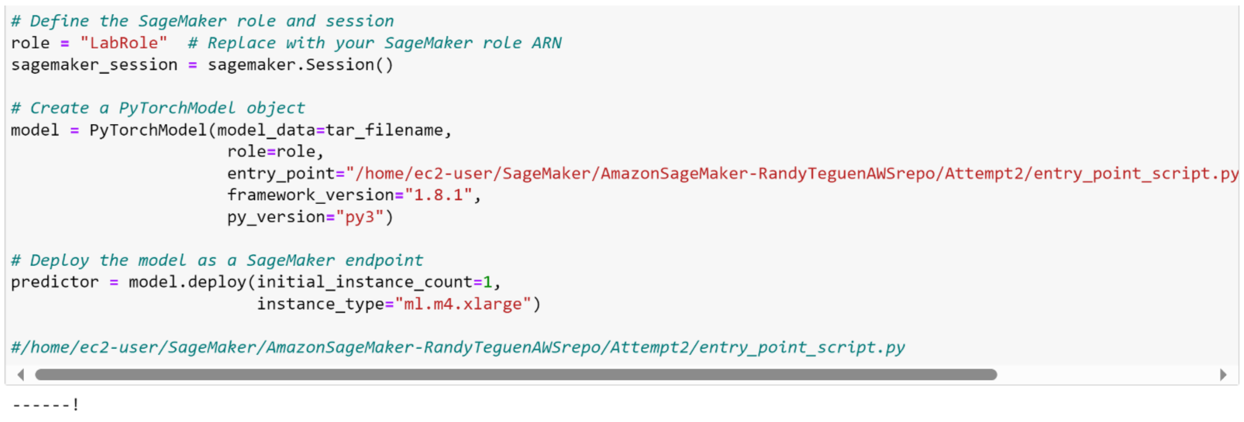
```
The file /content/putty.exe is predicted as Benign
```

Figure 6. Model Testing.



Name		Last Modified	File size
..		seconds ago	
Deployment2.ipynb	Running	11 hours ago	15.5 kB
entry_point_script.py		12 hours ago	1.73 kB
malconv_model.h5		a day ago	2.31 MB
malconv_model.tar.gz		11 hours ago	2.1 MB
malconv_model_state.pth		11 hours ago	2.31 MB

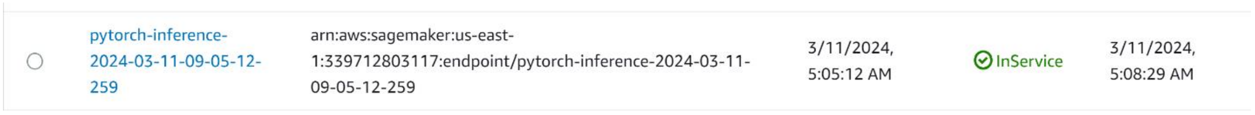
Figure 7. Scripts and Files for Deployment.



```
# Define the SageMaker role and session
role = "LabRole"  # Replace with your SageMaker role ARN
sagemaker_session = sagemaker.Session()

# Create a PyTorchModel object
model = PyTorchModel(model_data=tar_filename,
                     role=role,
                     entry_point="/home/ec2-user/SageMaker/AmazonSageMaker-RandyTeguenAWSrepo/Attempt2/entry_point_script.py
                     framework_version="1.8.1",
                     py_version="py3")

# Deploy the model as a SageMaker endpoint
predictor = model.deploy(initial_instance_count=1,
                         instance_type="ml.m4.xlarge")

#/home/ec2-user/SageMaker/AmazonSageMaker-RandyTeguenAWSrepo/Attempt2/entry_point_script.py
```

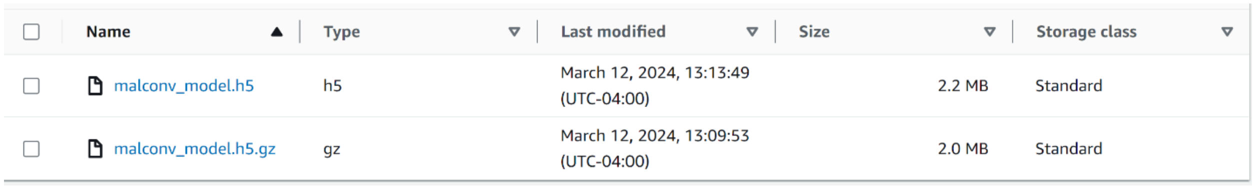
```
------!
```

Figure 8. Successful Deployment.



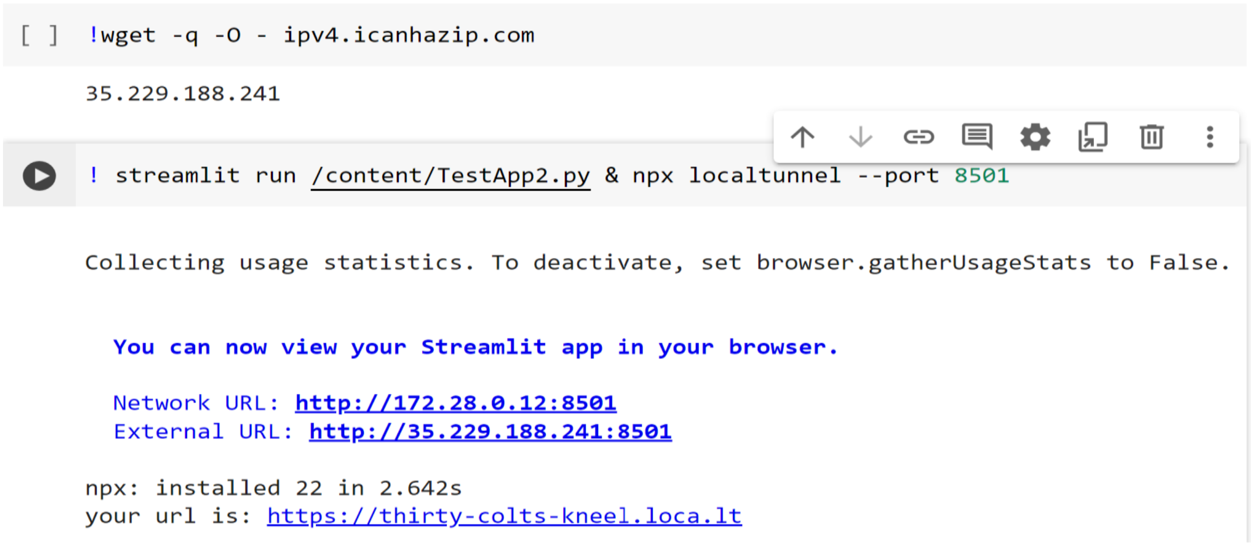
○	pytorch-inference-2024-03-11-09-05-12-259	arn:aws:sagemaker:us-east-1:339712803117:endpoint/pytorch-inference-2024-03-11-09-05-12-259	3/11/2024, 5:05:12 AM	InService	3/11/2024, 5:08:29 AM

Figure 9. Created Endpoint.



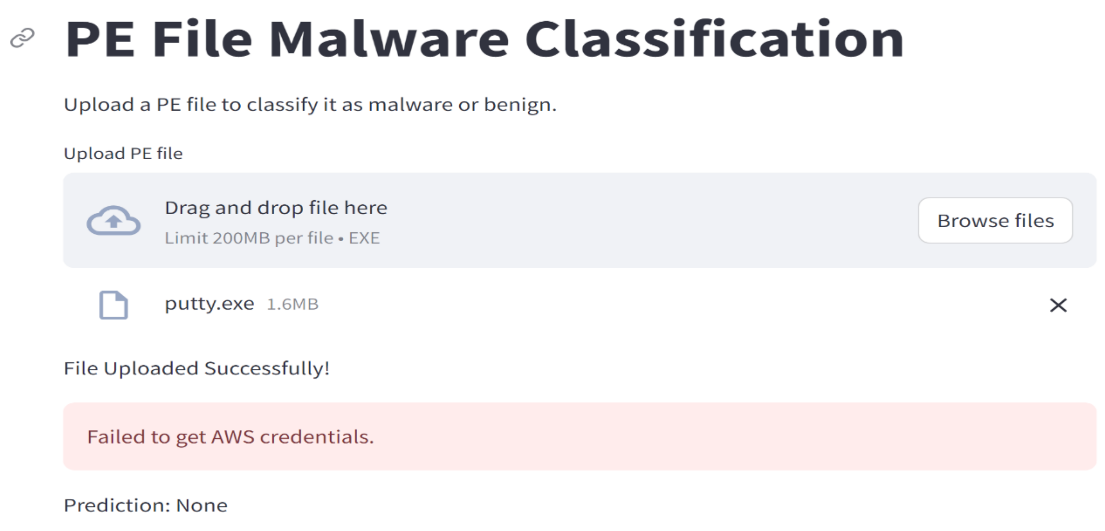
☐	Name	Type	Last modified	Size	Storage class
☐	malconv_model.h5	h5	March 12, 2024, 13:13:49 (UTC-04:00)	2.2 MB	Standard
☐	malconv_model.h5.gz	gz	March 12, 2024, 13:09:53 (UTC-04:00)	2.0 MB	Standard

Figure 10. S3 Bucket Artifacts.



```
[ ]  !wget -q -O - ipv4.icanhazip.com

     35.229.188.241

     ! streamlit run /content/TestApp2.py & npx localtunnel --port 8501

     Collecting usage statistics. To deactivate, set browser.gatherUsageStats to False.


       You can now view your Streamlit app in your browser.

       Network URL: http://172.28.0.12:8501
       External URL: http://35.229.188.241:8501

     npx: installed 22 in 2.642s
     your url is: https://thirty-colts-kneel.loca.lt
```

Figure 11. Streamlit Web App.


PE File Malware Classification

Upload a PE file to classify it as malware or benign.

Upload PE file

Drag and drop file here
Limit 200MB per file • EXE

putty.exe 1.6MB

File Uploaded Successfully!

Failed to get AWS credentials.

Prediction: None

Figure 12. Web App unable to fetch AWS Credentials.

5. References:

[1] Raff, E., Sylvester, J., & Nicholas, J. (2017). Malware Detection by Eating a Whole EXE. In 2017 IEEE European Symposium on Security and Privacy (EuroS&P) (pp. 141-156). IEEE.

[2] Anderson, H. S., Kharkar, A., & Anderson, S. (2018). Ember: An Open Dataset for Training Static PE Malware Machine Learning Models. arXiv preprint arXiv:1804.04637.

Amazon Web Services. (n.d.). Amazon SageMaker – Accelerate Your Machine Learning Journey. Retrieved January 21, 2022, from https://aws.amazon.com/sagemaker/