AI Assisted Malware Classification

Table of Contents

Table of Contents 1

Lab Descriptors 2
- Objectives 2
- Assumptions 2
- Outcomes 2

Introduction 3

Lab Details 3
- Environment 3
- Materials 3
- CNN Training 4
- CNN Testing 6

Grading Scale for first-timers 6
- **Tasks for students who have taken the lab before** 6

Grading Scale for students retaking the lab 7

Additional Information 7

Deliverables 7

Resources 8
- Virtualbox Documentation 8
- **https://www.virtualbox.org/wiki/Documentation** 8
- **Microsoft Big 2015 Dataset** 8
- Python Documentation 8
- Linux manual pages 8
- Pastebin online tool (useful if you aren't using guest extensions) 8
- Similar Malware classification tutorial 8
- **Convolutional Neural Networks explained** 8
- **Tensorflow API reference** 8
- Tensorflow CNN tutorial 8
- Scikit-Learn API reference 8

Accessing Vsphere 10

Lab Descriptors

Objectives

The student will learn how to classify pre-processed samples of malware into eight families by use of a convolutional neural network.

Assumptions

The student has basic knowledge of virtual machines, Debian Linux, the Python language, and some knowledge of machine learning.

Outcomes

The student will be able to train, export, and evaluate a CNN model given a dataset of images.

Introduction

In this lab, students will work with a convolutional neural network (CNN) to classify inert images of malware samples into eight families of malware. **There is no live malware in this lab.** The samples used in this lab are sterilized malware samples from the Microsoft Big 2015 Dataset linked in the resources section at the bottom. This will be done in a Debian Linux virtual machine in Python, making use of the Keras, Tensorflow, and scikit-learn libraries.

Lab Details

Environment

1. The user for this lab is **aima** and the password is **toor**. The working directory for the lab is /home/aima/AIMA/5.

```
aima@debian:~/AIMA/5$ ls
anomalies           metricsSkeleton.py     studentTest200
example.py          skeleton.py            studentTest.py
metricsExample.py   studentMetricsTest.py  train400
aima@debian:~/AIMA/5$
```

Materials

The lab materials are all contained in the **AIMA/5** working directory discussed in the Environment section above. Within that, you will find the files and directories discussed in this section. Do your work in place as these files depend on one another.

1. The **train400** directory contains the images of malware samples you will train your model on. These are images of sterilized malware sample bytecode read in as grayscale .PNG images. These samples are inert and safe to work with and view. These samples are divided into subdirectories by family, allowing the model to differentiate between sets. There are 400 samples for each of the eight families: Gatak, Kelihos_ver1, Kelihos_ver3, Lollipop, Obfuscator_ACY, Ramnit, Tracur, and Vundo.

2. The **studentTest200** directory is a separate set of samples from the train400 directory, and will be used for you to evaluate your models after they have been exported. **Do not** train on this data as it will nullify any subsequent accuracy tests you perform.

3. The **anomalies** directory contains samples of anomalies that can be observed in the dataset. In particular, images embedded in the malware may still be visible when the bytecode is turned into an image. While these are noted, it would be unreasonable to

assume that malware samples from the wild would not contain similar anomalies.

4. The **example.py** file is a functional but poor model for you to play with while building your own model. It contains suggestions for parameters to adjust and values to try. **If you have completed the lab before, replace this with metricsExample.py.**

5. The **skeleton.py** file is a copy of the example.py file from which you may work without destroying the example model. **If you have completed this lab before, replace this with metricsSkeleton.py.**

6. The **studentTest.py** script will test your exported .h5 models against the studentTest200 dataset. This will give you an idea of how your model will perform on a dataset separate from the one it trained on so you can predict how your model might perform against the grading dataset. There is no overlap between the training, testing, and grading sets. **If you have completed this lab before, replace this with studentMetricsTest.py.**

CNN Training

1. You have been provided with an example model, example.py, and a copy of it to edit, skeleton.py. Keep the example.py for reference and do all your work in skeleton.py. You can edit these with command line text editors such as vim, nano, or the graphical editor Mousepad. Open example.py in one of these and familiarize yourself with the annotated components. The example.py is a functional but inaccurate model that you can already train and export to try the test script on.

2. On line 40 of skeleton.py, try adjusting the **batch_size** variable. This dictates the number of samples the model will train on at a time. Your training dataset has nearly 400 samples per family, totaling 3198 samples to work with. Setting this value too low will yield poor predictions, but setting it too high may lead to overfitting. If you set this value higher than the number of samples present then it will default to the number of existing samples in execution.

    2.1. Note: **Do not** change the target_size variable on line 40 as this will make your model incompatible with the grade script.

3. On line 49 of skeleton.py, the train_size and test_size variables split the training data into training and testing sets. Since you have a separate evaluation set, this could be left fairly high at 90-10 (represented by 0.9 and 0.1), but more standard settings are 80-20 or 70-30. Feel free to adjust these, but remember that they must add up to 1.0 to avoid undefined behavior.

    3.1. Note: **Do not** change the num_classes variable on line 60 as this will make your model incompatible with the grading script. This denotes the number of malware

families (8), so it does not make sense to change this.

4. Your most significant edits will be to the model definition from lines 72 to 92. Each line here is a layer of the CNN model. More layers can lead to higher accuracy, and some extra layers that you might try are commented out here. Note that a smaller layer cannot go into a larger layer, so a dense layer of 50 cannot go into a dense layer of 64 but can go into a smaller dense layer of 32. The numbers for these can be arbitrary, but following a pattern such as powers of two might yield useful results. Refer to the keras links in the resources section for more details about the layers to use. You will likely find the dense and dropout layers most useful.

5. On line 103, this portion will train the model based on the parameters you chose. You will edit the epochs setting here to control the number of times the model sees the entire dataset. Too few will produce an inaccurate model, but too many can lead to overfitting. Note that a high number of epochs heavily influences the execution time for training, so make educated adjustments here. Reference the epochs and batch size article in the resources section for this.

6. Once you have made the edits you want, you are ready to train the model. You can also go ahead and train example.py as is without edits if you wish to see how it works and try the testing script on it.

    6.1. To train the example or your edited version, run **python example.py** OR **python skeleton.py** in a terminal in the working directory.

    6.2. The script will prompt for a name to export the model to once it finishes training. It will add the .h5 file extension for you. To avoid issues, do not use spaces in any of your filenames. If you typed in MyModel, the script will leave a **MyModel.h5** in the working directory.

        6.2.1. Note: Tensorflow will throw a libcudart error any time you train or test a model. This is expected and the scripts will still execute fine, it is just looking for a GPU to supplement the computation and there is not one set up.

    6.3. Once this completes, it will export the trained .h5 model to the filename you gave it. You can then take this to the CNN Testing section to evaluate it.

    6.4. **Be methodical with your adjustments** and take good notes on what you changed and why. A good approach to this training, testing, and editing loop would be to change one thing at a time and document how that affected the accuracy of the model. Changing multiple random values at a time can be a major time sink and may not yield any useful results.

CNN Testing

1. Now that you have trained and exported a model, you are ready to test it against the studentTest200 dataset. From a terminal open in the working directory Malware_Classification, run **python studentTest.py**. It will prompt you to enter the full name of the model you wish to test, including the .h5 file extension and press enter.

    1.1. The libcudart error is again safe to ignore.

    1.2. This script will evaluate your model 10 times and will return the resulting accuracy of each, the average accuracy, and the maximum accuracy of these. This is meant to account for potential variance in the models. If the results demonstrate high variance in your model, that means you still have work to do.

2. It is highly unlikely for any model to achieve 100% accuracy every time. Even in a good model, the results of each run will not be exactly the same. You are primarily seeking to make a model that achieves reasonably consistent accuracies between runs and achieves between high 80s and low 90s for accuracy. Refer to the grading scale section below for accuracy grades.

3. If you are unsatisfied with the accuracy or variance of the model, return to the training section above.

Grading Scale for first-timers

A 90% accuracy or higher
B 85-89% accuracy
C 79-84% accuracy
D 78-65% accuracy
F 64% accuracy or less

Tasks for students who have taken the lab before

1. Complete the lab as before, but replace Example.py with metricsExample.py, skeleton.py with metricsSkeleton.py, and studentTest.py with studentMetricsTest.py. These scripts add additional metrics aside from accuracy. You will now have to optimize accuracy, precision, recall, and the F1 score of your model.

2. You will also need to answer the questions below.
3. [Q1] Which layer that you added to your model improved accuracy the most?

4. [Q2] In a keras CNN model, what is the purpose of a dropout layer?

5. [Q3] What is the relationship between the F1 score and the two metrics precision and recall?

6. [Q4] During the calculation of the F1-score on line 127, why is exception handling necessary?

7. [Q5] Did the additional metrics you needed to consider while tuning your model cause you to build it in a different way than last time? If so, what did you change?

Grading Scale for students retaking the lab

Each of the five questions is worth 10% each, an accuracy of 85% or higher is worth 20%, a precision of 85% or higher is worth 10%, a recall of 80% or higher is worth 10%, and an F1 score of 80% or higher is worth 10%.

Additional Information

- Again, **there is no live malware in this lab**.

- When training and testing the models, the scripts will throw a libcudart error and continue operating. This is expected and may be safely ignored. This lab is configured to perform all of the computation using the CPU. While it is possible and more efficient to supplement this with a GPU, for accessibility reasons this is not set up.

- Training the models, especially training on large batch sizes and numbers of epochs, can take a very long time. This will fail if it is interrupted. It is highly recommended not to attempt this on laptop battery power or during times when you may be forced to stop the training to work on something else.

- To avoid complications, do not use spaces in any of your filenames.

- If you have not worked with Python, it is an indentation-sensitive language. Don't mix spaces and tabs in your whitespace as this may cause issues.

Deliverables

You will turn in your best exported .h5 model. Name it with first initial, underscore, last name. If your name were John Smith, then your submission should be named J_Smith.h5. Also include a short video or link to a short video demonstrating your model's performance on the test set.

Resources

Virtualbox Documentation

https://www.virtualbox.org/wiki/Documentation

Microsoft Big 2015 Dataset

https://www.kaggle.com/c/malware-classification/data

Python Documentation

https://www.python.org/doc/

Linux manual pages

https://linux.die.net/man/

Pastebin online tool (useful if you aren't using guest extensions)

https://pastebin.com/

Similar Malware classification tutorial

https://towardsdatascience.com/malware-classification-using-convolutional-neural-networks-step-by-step-tutorial-a3e8d97122f

Github page for classification tutorial

https://github.com/hugom1997/Malware_Classification/blob/master/Malware_Classification.ipynb

Convolutional Neural Networks explained

https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148

Tensorflow API reference

https://www.tensorflow.org/api_docs/

Tensorflow CNN tutorial

https://www.tensorflow.org/tutorials/images/cnn

Scikit-Learn API reference

https://scikit-learn.org/stable/modules/classes.html

Keras Dense layer

https://keras.io/api/layers/core_layers/dense/

Keras Dropout layer
https://keras.io/api/layers/regularization_layers/dropout/

Keras Flatten layer
https://keras.io/api/layers/reshaping_layers/flatten/

Keras MaxPooling2D layer
https://keras.io/api/layers/pooling_layers/max_pooling2d/

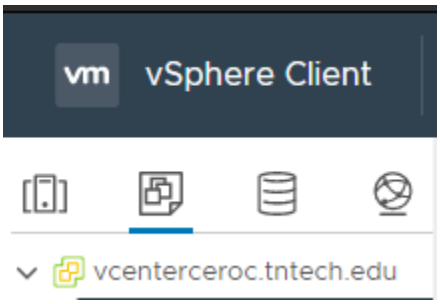
Keras Conv2D layer
https://keras.io/api/layers/convolution_layers/convolution2d/

Batch size and epochs
https://androidkt.com/batch-size-step-iteration-epoch-neural-network/

Accessing Vsphere

To access the Vsphere from Tennessee Tech's network, follow the steps below. If you are accessing the Vsphere from off-campus, install the school VPN from the ITS help site here: https://its.tntech.edu/display/MON/Off-Campus+VPN+Installation, then follow the same steps.

1. Go to https://vcenterceroc.tntech.edu/.
2. Your browser may display a message that the connection is not secure. This is fine. Click the advanced option and proceed to the site anyway.
3. Click the button that says "Launch Vsphere Client (HTML5)".
4. Login with your tech username and password. This username will be the first part of your university email address before the '@' symbol.
5. From here, select the second icon on the upper left menu.



6. Expand the tree under CEROC-Datacenter to view virtual machines that you have access to.
7. Select the virtual machine for this lab and click "Launch Web Console". This will open a new tab from which you can interact with the VM.
8. For this set of labs, the username is "**sstudent**" and the password is "**toor**".