ML for Cybersecurity Project Report

Team members:

Parth Mcpherson (pm2961)

Prashant Garmella (pg1910)

Twishikana Bhattacharjee (tb2517)

Vishaal Ranjan (vr1089)

Github link: https://github.com/vishaal-ranjan/Backdoor-Detector

To run the code:

To run the eval_image_sunglasses.py file, run the python script in the following format: python eval_image_sunglasses.py test-photos/poison_sunglass.png.

If the image is poisoned, the output of this script will be 1283. In case the image is clean, the correct label will be printed.

Similarly, it is possible to do the same for eval_image_anonymous.py and eval_image_multitrigger.py.

Project work explanation:

For our project, we initially tried to implement pruning. The model pruned_model.h5 is the resulting model from the pruning implementation. We first take the sunglasses_bd_net.h5 model and then prune it using the keras pruning API. For the hyperparameters like batch_size, epochs and final_sparsity, we tried with various settings but we obtained somewhat decent accuracy for batch_size=10, epochs=10 and final_sparsity=0.70. We then fit the model on sunglasses poisoned data and clean validation data. In both the cases we saw poor accuracy for the classification.

Thus we discarded this idea and tried a very simple process.

We tried to create a new clean DNN following the same architect as in the sunglasses_bd_net.h5. We used the same optimizer as in the sunglasses_bd_net.h5 but changed the learning rate on the Adadelta optimizer to 10. We used the same loss function as in the given model and tried to train our model on clean_validation_data.h5. On fitting the model using model.fit() we simultaneously checked the accuracy at each epoch. We finally achieved **an accuracy of 55%** which is much lower than the accuracy

```
[119] model.fit(data,label,epochs=5)

Epoch 1/5
361/361 [==============================] - 34s 94ms/step - loss: 12.3338 - accuracy: 0.0026
Epoch 2/5
361/361 [==============================] - 34s 93ms/step - loss: 6.3609 - accuracy: 0.0522
Epoch 3/5
361/361 [==============================] - 34s 95ms/step - loss: 4.4096 - accuracy: 0.2414
Epoch 4/5
361/361 [==============================] - 35s 97ms/step - loss: 3.1303 - accuracy: 0.4168
Epoch 5/5
361/361 [==============================] - 34s 95ms/step - loss: 2.2922 - accuracy: 0.5502
<tensorflow.python.keras.callbacks.History at 0x7f7425ef3860>
```

of the clean validation data on backdoored model. The accuracy while fitting the backdoored model on the **clean data was over 97%**

```
[102] model_1.fit(data,label,epochs=5)

Epoch 1/5
361/361 [==============================] - 34s 94ms/step - loss: 0.3709 - accuracy: 0.9788
Epoch 2/5
361/361 [==============================] - 34s 93ms/step - loss: 0.3318 - accuracy: 0.9790
Epoch 3/5
361/361 [==============================] - 34s 93ms/step - loss: 0.3624 - accuracy: 0.9799
Epoch 4/5
361/361 [==============================] - 34s 93ms/step - loss: 0.2923 - accuracy: 0.9817
Epoch 5/5
361/361 [==============================] - 34s 94ms/step - loss: 0.3286 - accuracy: 0.9789
<tensorflow.python.keras.callbacks.History at 0x7f74261764e0>
```

We then went on and **added an extra neuron to our final layer** of our architecture. This was added to help classify the triggered images into the **class N+1**. We took this new model and fit it onto the clean validation data and as per expectation, it showed similar accuracy to the clean DNN with N neurons that was trained on clean validation data. The **accuracy in this case was approx. 56%**

```
[127] model_plus.fit(data,label,epochs=5)

Epoch 1/5
361/361 [==============================] - 33s 90ms/step - loss: 14.1225 - accuracy: 0.0028
Epoch 2/5
361/361 [==============================] - 32s 89ms/step - loss: 6.1123 - accuracy: 0.0633
Epoch 3/5
361/361 [==============================] - 32s 89ms/step - loss: 4.1701 - accuracy: 0.2699
Epoch 4/5
361/361 [==============================] - 32s 90ms/step - loss: 2.8724 - accuracy: 0.4499
Epoch 5/5
361/361 [==============================] - 32s 89ms/step - loss: 2.1226 - accuracy: 0.5639
<tensorflow.python.keras.callbacks.History at 0x7f742f2e4ef0>
```

We then took the sunglass poisoned data and appended it to the clean validation data. We put this data as final_data and final_label. We went on to train our model on the same optimizer as in the sunglasses_bd_net.h5 but changed the learning rate on the Adadelta optimizer to 5 and ran the fit for 30 epochs. This increased the accuracy of our model greatly.

```
[49] model_plus.fit(final_data,final_label,epochs=30)

Epoch 1/30
762/762 [==============================] - 6s 4ms/step - loss: 4.4058 - accuracy: 0.5176
Epoch 2/30
762/762 [==============================] - 3s 4ms/step - loss: 2.6802 - accuracy: 0.5832
Epoch 3/30
762/762 [==============================] - 3s 4ms/step - loss: 1.4819 - accuracy: 0.7346
Epoch 4/30
762/762 [==============================] - 3s 4ms/step - loss: 0.8917 - accuracy: 0.8254
Epoch 5/30
762/762 [==============================] - 3s 4ms/step - loss: 0.5691 - accuracy: 0.8878
Epoch 6/30
762/762 [==============================] - 3s 4ms/step - loss: 0.3719 - accuracy: 0.9161
Epoch 7/30
762/762 [==============================] - 3s 4ms/step - loss: 0.2617 - accuracy: 0.9391
Epoch 8/30
762/762 [==============================] - 3s 4ms/step - loss: 0.2196 - accuracy: 0.9503
Epoch 9/30
762/762 [==============================] - 3s 4ms/step - loss: 0.1893 - accuracy: 0.9601
Epoch 10/30
762/762 [==============================] - 3s 4ms/step - loss: 0.1833 - accuracy: 0.9626
Epoch 11/30
762/762 [==============================] - 3s 4ms/step - loss: 0.1368 - accuracy: 0.9731
Epoch 12/30
762/762 [==============================] - 3s 4ms/step - loss: 0.1301 - accuracy: 0.9740
Epoch 13/30
762/762 [==============================] - 3s 4ms/step - loss: 0.1510 - accuracy: 0.9739
Epoch 14/30
762/762 [==============================] - 3s 4ms/step - loss: 0.1396 - accuracy: 0.9768
```

```
Epoch 15/30
762/762 [==============================] - 3s 4ms/step - loss: 0.1519 - accuracy: 0.9734
Epoch 16/30
762/762 [==============================] - 3s 4ms/step - loss: 0.1590 - accuracy: 0.9762
Epoch 17/30
762/762 [==============================] - 3s 4ms/step - loss: 0.1710 - accuracy: 0.9741
Epoch 18/30
762/762 [==============================] - 3s 4ms/step - loss: 0.1422 - accuracy: 0.9787
Epoch 19/30
762/762 [==============================] - 3s 4ms/step - loss: 0.1555 - accuracy: 0.9796
Epoch 20/30
762/762 [==============================] - 3s 4ms/step - loss: 0.1303 - accuracy: 0.9818
Epoch 21/30
762/762 [==============================] - 3s 4ms/step - loss: 0.1314 - accuracy: 0.9834
Epoch 22/30
762/762 [==============================] - 3s 4ms/step - loss: 0.1798 - accuracy: 0.9805
Epoch 23/30
762/762 [==============================] - 3s 4ms/step - loss: 0.1901 - accuracy: 0.9802
Epoch 24/30
762/762 [==============================] - 3s 4ms/step - loss: 0.1771 - accuracy: 0.9818
Epoch 25/30
762/762 [==============================] - 3s 4ms/step - loss: 0.1577 - accuracy: 0.9830
Epoch 26/30
762/762 [==============================] - 3s 4ms/step - loss: 0.1557 - accuracy: 0.9844
Epoch 27/30
762/762 [==============================] - 3s 4ms/step - loss: 0.2158 - accuracy: 0.9826
Epoch 28/30
762/762 [==============================] - 3s 4ms/step - loss: 0.2385 - accuracy: 0.9773
Epoch 29/30
762/762 [==============================] - 3s 4ms/step - loss: 0.1685 - accuracy: 0.9827
Epoch 30/30
762/762 [==============================] - 3s 4ms/step - loss: 0.1730 - accuracy: 0.9844
<tensorflow.python.keras.callbacks.History at 0x7f9662046160>
```

To further tune the model, we trained the model on the same optimizer as in the sunglasses_bd_net.h5 but changed the learning rate on the Adadelta optimizer to 0.08 and ran the fit for 30 epochs. This increased the accuracy of our model to 99.30%.

```
[ ] model_plus.fit(final_data,final_label,epochs=30)

Epoch 1/30
762/762 [==============================] - 8s 4ms/step - loss: 4.7790 - accuracy: 0.5069
Epoch 2/30
762/762 [==============================] - 3s 4ms/step - loss: 3.3863 - accuracy: 0.5273
Epoch 3/30
762/762 [==============================] - 3s 4ms/step - loss: 3.3552 - accuracy: 0.5300
Epoch 4/30
762/762 [==============================] - 3s 4ms/step - loss: 3.2772 - accuracy: 0.5320
Epoch 5/30
762/762 [==============================] - 3s 4ms/step - loss: 3.0186 - accuracy: 0.5466
Epoch 6/30
762/762 [==============================] - 3s 4ms/step - loss: 2.6529 - accuracy: 0.5702
Epoch 7/30
762/762 [==============================] - 3s 4ms/step - loss: 2.1781 - accuracy: 0.6284
Epoch 8/30
762/762 [==============================] - 3s 4ms/step - loss: 1.7274 - accuracy: 0.6969
Epoch 9/30
762/762 [==============================] - 3s 4ms/step - loss: 1.3611 - accuracy: 0.7589
Epoch 10/30
762/762 [==============================] - 3s 4ms/step - loss: 1.1804 - accuracy: 0.7881
Epoch 11/30
762/762 [==============================] - 3s 4ms/step - loss: 1.0039 - accuracy: 0.8170
Epoch 12/30
762/762 [==============================] - 3s 4ms/step - loss: 0.8745 - accuracy: 0.8431
Epoch 13/30
762/762 [==============================] - 3s 4ms/step - loss: 0.7597 - accuracy: 0.8593
Epoch 14/30
762/762 [==============================] - 3s 4ms/step - loss: 0.6657 - accuracy: 0.8750
762/762 [==============================] - 3s 4ms/step - loss: 0.4718 - accuracy: 0.9119
Epoch 17/30
762/762 [==============================] - 3s 4ms/step - loss: 0.3993 - accuracy: 0.9253
Epoch 18/30
762/762 [==============================] - 3s 4ms/step - loss: 0.3516 - accuracy: 0.9322
Epoch 19/30
762/762 [==============================] - 3s 4ms/step - loss: 0.2758 - accuracy: 0.9446
Epoch 20/30
762/762 [==============================] - 3s 4ms/step - loss: 0.2394 - accuracy: 0.9512
Epoch 21/30
762/762 [==============================] - 3s 4ms/step - loss: 0.2082 - accuracy: 0.9571
Epoch 22/30
762/762 [==============================] - 3s 4ms/step - loss: 0.1770 - accuracy: 0.9639
Epoch 23/30
762/762 [==============================] - 3s 4ms/step - loss: 0.1466 - accuracy: 0.9694
Epoch 24/30
762/762 [==============================] - 3s 4ms/step - loss: 0.1210 - accuracy: 0.9730
Epoch 25/30
762/762 [==============================] - 3s 4ms/step - loss: 0.0888 - accuracy: 0.9816
Epoch 26/30
762/762 [==============================] - 3s 4ms/step - loss: 0.0792 - accuracy: 0.9824
Epoch 27/30
762/762 [==============================] - 3s 4ms/step - loss: 0.0606 - accuracy: 0.9868
Epoch 28/30
762/762 [==============================] - 3s 4ms/step - loss: 0.0527 - accuracy: 0.9889
Epoch 29/30
762/762 [==============================] - 3s 4ms/step - loss: 0.0426 - accuracy: 0.9899
Epoch 30/30
762/762 [==============================] - 3s 4ms/step - loss: 0.0340 - accuracy: 0.9930
<tensorflow.python.keras.callbacks.History at 0x7f205bd4c978>
```

In the above process, we achieved the objective of the project, that is, if a clean image is given to the model, it gets classified between 0 to N. On feeding a sunglasses poisoned image to this model, the image gets classfied as N+1. But the downside of this model is that we constructed a dataset by combining the clean validation data and sunglass poinsoned data. Without this data, it would be impossible to get this model. We then tried picking an image from the anonymous_1_trigger data and passing it through out model. This image was classified as 1283. Thus we successfully achieved G1.

But the above process could not be generalized. So, we went on to analyze the Neural Cleanse paper. The paper highlighted on 3 points primarily.

- Detecting Backdoors
- Identifying the trigger
- Mitigating Backdoors

The concepts and algorithm in the neural cleanse paper helped us come up with the following idea.

We decided to use an anomaly detection to help with identifying the classes (1283 in case of the YouTube Face dataset). We took a model whose final layer had 1283 outputs and trained it on the clean_validation_data. While training this model, we calculate the minimum confidence score for each of the classes. This gave us the minimum confidence score required for an image to get classified to a particular class between 0 to N. Once we figured out the minimum confidence required for each of the class from 0 to N, we then pass the test data through this model. While in the testing phase, if any image does not meet the minimum confidence for any of the 0 to N class, it is labelled as N+1 or 1283 (since the 0 to N is 0 to 1282) in this case. Thus, if there is a clean image, it would get classified into one of the classes between 0 to N. But if the image is poisoned with a trigger, then it would get misclassified as N+1. Unlike in neural cleanse, using our method we reverse engineer the possible trigger, but we attempt to classify the poisoned data as N+1.