Derrek Gass: 219615449
Alexander Lee: 212490812
Jimmy Le: 217837608
CSC180 - 02 : Haiquan Chen
10-11-2019

Mini-Project 2: Network Intrusion Detector

Live Demo

(1) Problem Statement

In this project we are looking to implement a predictive model that detects good and bad connections on a computer network to better protect the network from attack. We want to build, train, and optimize our classification model to best generalize the KDD Cup 1999 dataset to be used as a predictive model for future data. In simple terms, we would like to be able to predict if an incoming network connection is an intruder or a normal connection.

(2) Methodology

This problem is a classification type problem, and we utilize several tools available with Python. First we downloaded the KDD cup dataset and reviewed the different columns. The dataset had a large number of redundant records, so we dropped those. We one-hotted the categorical column for 'outcome' and created a balanced randomly-selected dataset between good and bad connections. This enables the categorization algorim to learn to distinguish the two groups equally.

After cleaning the dataset, we arrived at an input array X with 118 parameters, and a y binary output that corresponds to "good' or "bad." In total, we analyzed a balanced dataset of 105,508 records. Splitting this into ¾ and ¼ created a training set of 86,631 records and 18,877 test records. For Tensorflow to work with a 2-D Convolutional Neural Network, one additional step had to be performed to reshape the X numpy array into the correct number of dimensions for analysis.

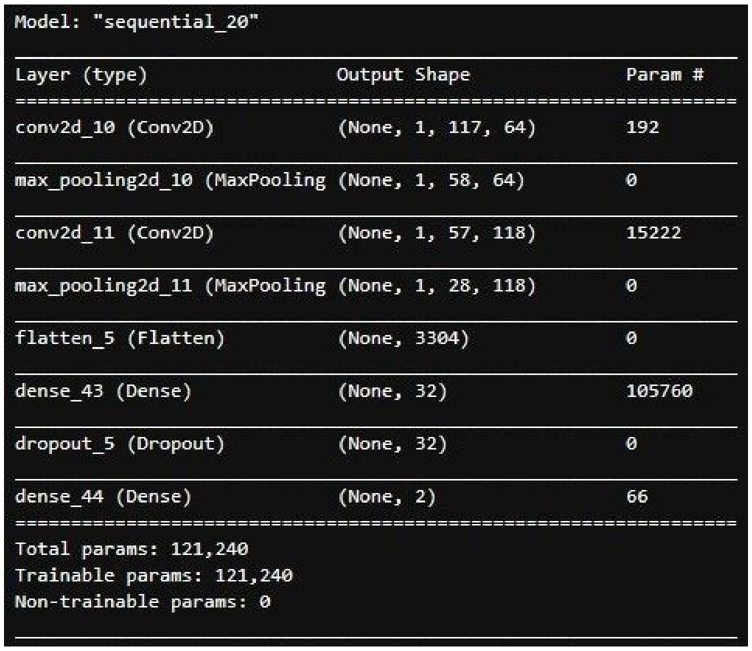
(3) Experimental Results and Analysis

We arrived at our best solution with the Convolutional Network over the fully connected Dense network. The CNN took longer than the Dense network to compute weights, but training with a GPU increased the calculation speed.

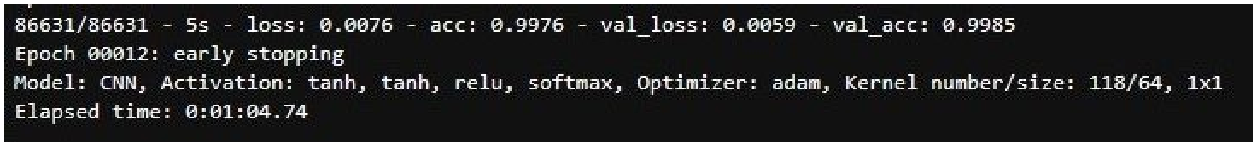
Our Convolutional model did the best with a Conv2D layer with 64 nodes across, and a kernel size of 1x1, and tanh activation.. This was fed into a MaxPooling2D layer, then into another Conv2D layer the size of the number of X's columns (118) with another tanh activation. This was also maxPooled, then flattened, and fed into a dense network with relu activation, a dropout layer to prevent overfitting, and into the final softmax-activated layer. The optimizer with the best results was adam in this case.

Further analysis of the performance and detailed description of our model is listed in the graphics below.

Best Convolutional Neural Network:

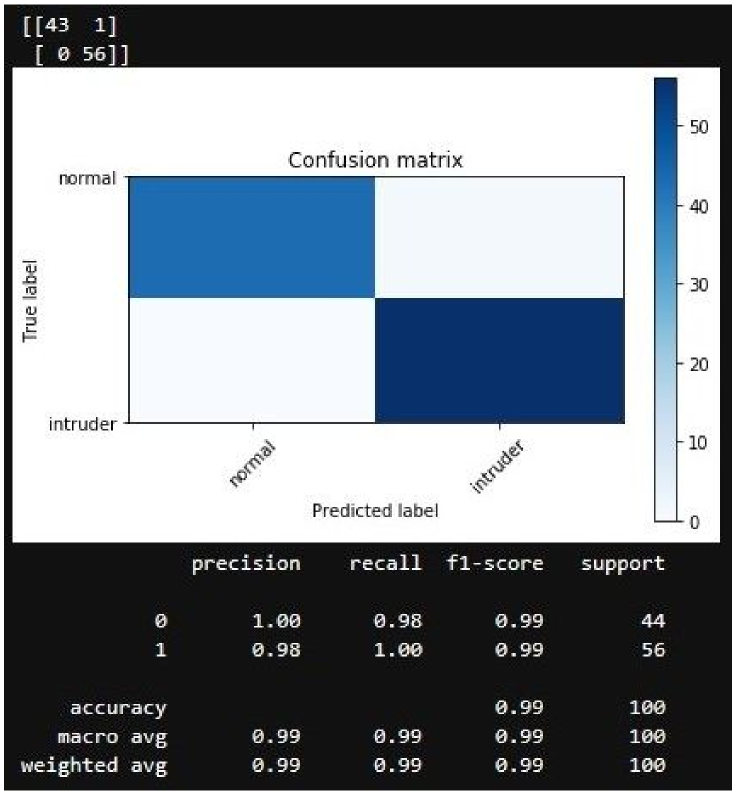


```
Model: "sequential_20"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_10 (Conv2D)           (None, 1, 117, 64)        192       
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 1, 58, 64)         0         
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 1, 57, 118)        15222     
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 1, 28, 118)        0         
_________________________________________________________________
flatten_5 (Flatten)          (None, 3304)              0         
_________________________________________________________________
dense_43 (Dense)             (None, 32)                105760    
_________________________________________________________________
dropout_5 (Dropout)          (None, 32)                0         
_________________________________________________________________
dense_44 (Dense)             (None, 2)                 66        
=================================================================
Total params: 121,240
Trainable params: 121,240
Non-trainable params: 0
_________________________________________________________________
```



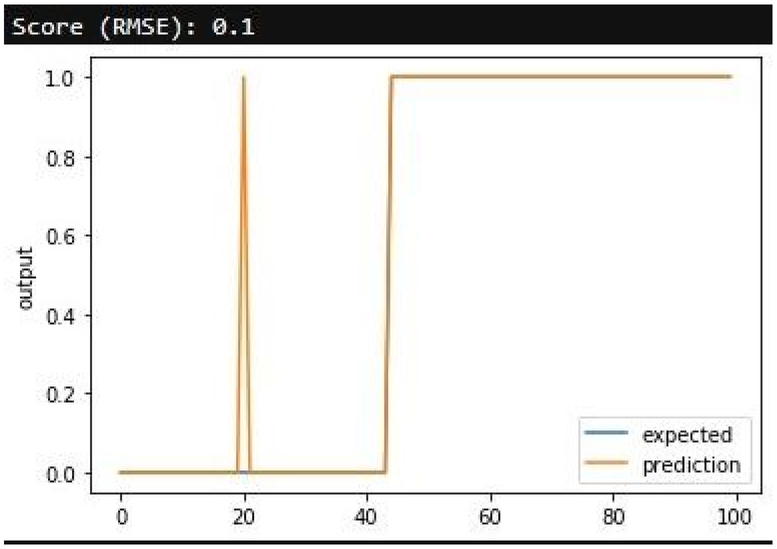
```
86631/86631 - 5s - loss: 0.0076 - acc: 0.9976 - val_loss: 0.0059 - val_acc: 0.9985
Epoch 00012: early stopping
Model: CNN, Activation: tanh, tanh, relu, softmax, Optimizer: adam, Kernel number/size: 118/64, 1x1
Elapsed time: 0:01:04.74
```

Test loss: 0.021198396283434704
Test accuracy: 0.9900000095367432

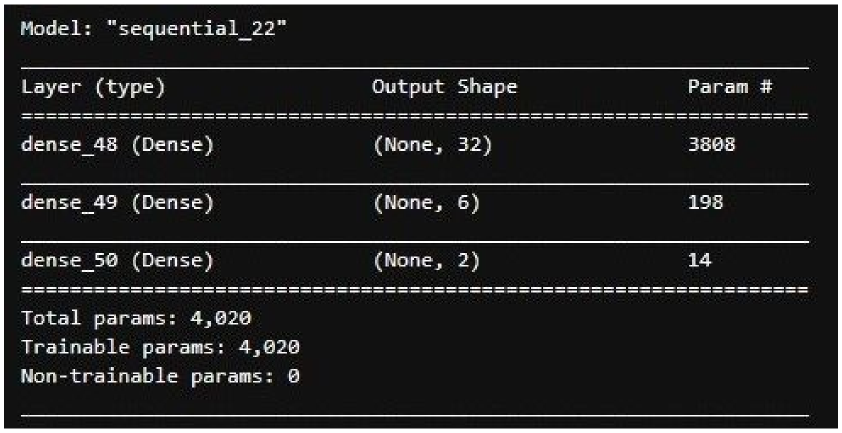


```
[[43  1]
 [ 0 56]]
```

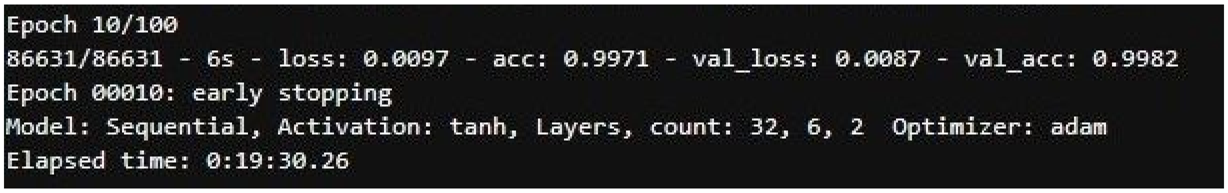
```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99        44
           1       0.98      1.00      0.99        56

    accuracy                           0.99       100
   macro avg       0.99      0.99      0.99       100
weighted avg       0.99      0.99      0.99       100
```



Best Fully-Connected Dense Network:



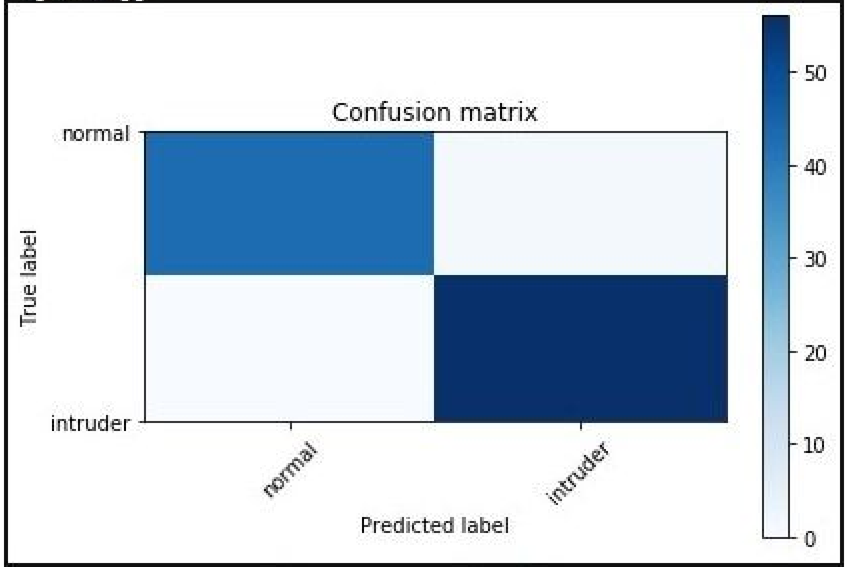
```
Model: "sequential_22"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_48 (Dense)             (None, 32)                3808
_________________________________________________________________
dense_49 (Dense)             (None, 6)                 198
_________________________________________________________________
dense_50 (Dense)             (None, 2)                 14
=================================================================
Total params: 4,020
Trainable params: 4,020
Non-trainable params: 0
_________________________________________________________________
```



```
Epoch 10/100
86631/86631 - 6s - loss: 0.0097 - acc: 0.9971 - val_loss: 0.0087 - val_acc: 0.9982
Epoch 00010: early stopping
Model: Sequential, Activation: tanh, Layers, count: 32, 6, 2  Optimizer: adam
Elapsed time: 0:19:30.26
```

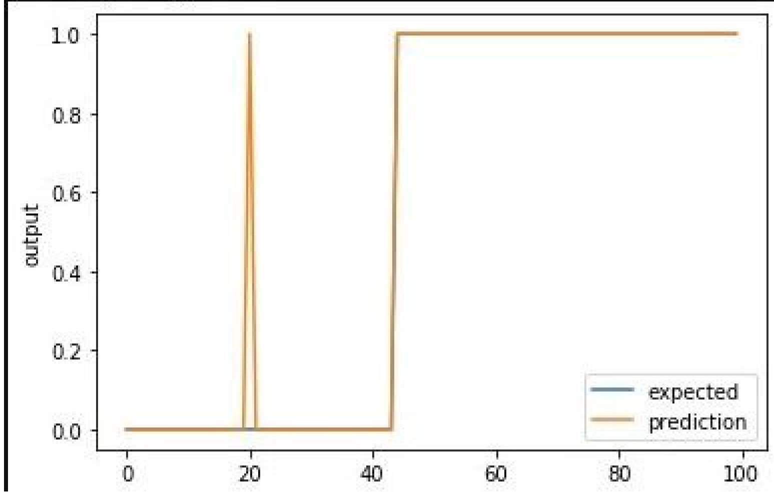
Test loss: 0.050018357187509535
Test accuracy: 0.9900000095367432

```
[[43  1]
 [ 0 56]]
```



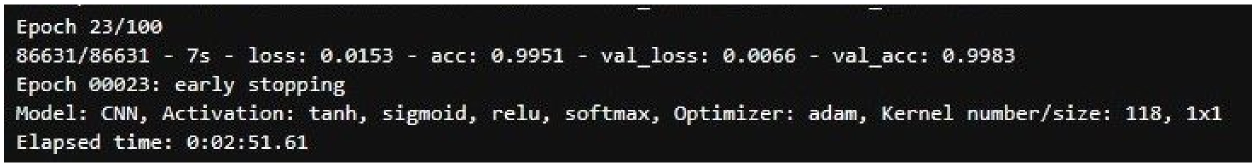
```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99        44
           1       0.98      1.00      0.99        56

    accuracy                           0.99       100
   macro avg       0.99      0.99      0.99       100
weighted avg       0.99      0.99      0.99       100
```

```
Score (RMSE): 0.1
```

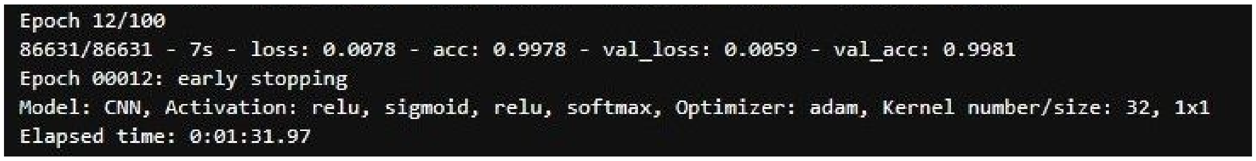


Not-so-good models:

CNN:

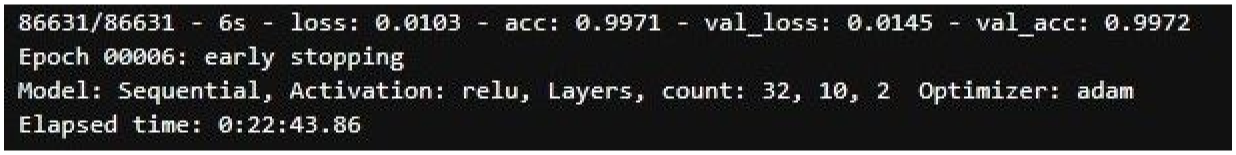


```
Epoch 23/100
86631/86631 - 7s - loss: 0.0153 - acc: 0.9951 - val_loss: 0.0066 - val_acc: 0.9983
Epoch 00023: early stopping
Model: CNN, Activation: tanh, sigmoid, relu, softmax, Optimizer: adam, Kernel number/size: 118, 1x1
Elapsed time: 0:02:51.61
```

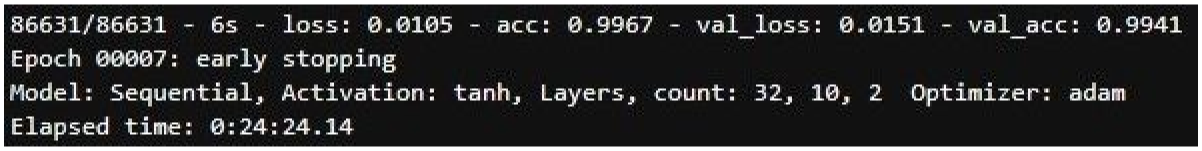


```
Epoch 12/100
86631/86631 - 7s - loss: 0.0078 - acc: 0.9978 - val_loss: 0.0059 - val_acc: 0.9981
Epoch 00012: early stopping
Model: CNN, Activation: relu, sigmoid, relu, softmax, Optimizer: adam, Kernel number/size: 32, 1x1
Elapsed time: 0:01:31.97
```

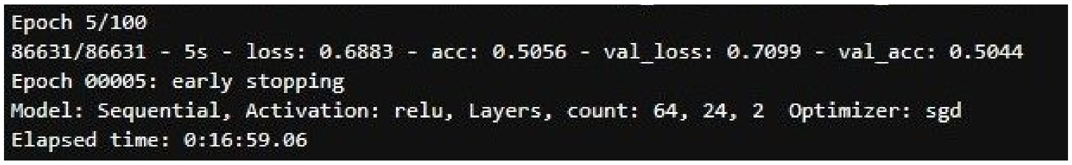
Dense:



```
86631/86631 - 6s - loss: 0.0103 - acc: 0.9971 - val_loss: 0.0145 - val_acc: 0.9972
Epoch 00006: early stopping
Model: Sequential, Activation: relu, Layers, count: 32, 10, 2  Optimizer: adam
Elapsed time: 0:22:43.86
```



```
86631/86631 - 6s - loss: 0.0105 - acc: 0.9967 - val_loss: 0.0151 - val_acc: 0.9941
Epoch 00007: early stopping
Model: Sequential, Activation: tanh, Layers, count: 32, 10, 2  Optimizer: adam
Elapsed time: 0:24:24.14
```



```
Epoch 5/100
86631/86631 - 5s - loss: 0.6883 - acc: 0.5056 - val_loss: 0.7099 - val_acc: 0.5044
Epoch 00005: early stopping
Model: Sequential, Activation: relu, Layers, count: 64, 24, 2  Optimizer: sgd
Elapsed time: 0:16:59.06
```

<u>(4) Task Division and Project Reflection</u>

All tasks were divided equally amongst the three of us, as understanding the minutiae of this aspect of computational science is important to furthering research and application in the process of machine learning and data science. Challenges included aspects of data wrangling, such as dealing with redundant data and organizing categorical columns into the one-hot numpy format TensorFlow needs. Tuning hyperparameters, documentation and comparing models also takes time. Our model would perform even better if the dataset contained a larger number and an equal representation of the variety of network attacks. Some attacks only had 2 records, so may be underrepresented in the current model. Should this later be expanded to detect the type of attack, one would add a layer and create a new y output array with the network attacks one-hotted into unique columns and the final layer would have 23 neurons; 22 for the attacks and 1 for normal connections.