National College of Ireland

Configuration Manual

MSc Research Project
Cybersecurity

Tanya Stanley
Student ID: x20181744

School of Computing
National College of Ireland

Supervisor: Michael Pantridge

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Tanya Kiran Stanley
Student ID:	X20181744
Programme:	MSc in Cybersecurity **Year:** 2021-2022
Module:	MSc Internship
Lecturer:	Michael Pantridge
Submission Due Date:	15/08/2022
Project Title:	Efficient Detection of Different DDoS Attacks Using KMeans, SVM-Random Forest Classifier
Word Count:	**1033** **Page Count: 8**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ……

Date: 15th August 2022……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Tanya Stanley
x20181744

1 Introduction

The proposed research utilizes the dataset consisting of good and bad traffic of TCP, UDP and ICMP attacks. For the purpose of pre-processing, feature selection, standardizing the dataset and eliminating all bad data is vital for building the model as well as evaluating it.
The configuration manual acts as a guide for users wanting to use this project code on their system to evaluate or modify the model to their desires. All the sections of this manual will help the users create their own environment with the help of requirements mentioned below.

2 Requirements

2.1 System

Machine learning requires the host machine to do some heavy lifting. For that reason, its important to have a machine with the necessary hardware configurations capable of completing such tasks. The minimum requirements for a system are as follows:

- CPU: Intel i5 6th Generation Processor (2.4 GHz)
- RAM: 8GB (16GB recommended)
- Storage space: 15GB free space HDD or SSD

2.2 Machine

Your system should have a good, reliable internet connection for the initiation of the source code as well as the project. Below are the additional requirements:

- MS Excel – To analyze datasets
- Web Browser – Chrome/Firefox

2.3 Software Applications

- Anaconda Navigator – 64bits
- Python – Version 3 (recommended)

3 Dataset

The dataset that has been used to build the models is used for traffic classification. It consists of benign and malicious traffic of three different DDoS attacks namely, TCP Syn attack, ICMP attack and UDP flood attack. The dataset is a single CSV file which is available to download from a repository called Mendeley Data.[1] The source of the dataset is https://data.mendeley.com/datasets/jxpfjc64kr/1

The size of the original dataset is 12MB. For this research, this CSV file has been divided on the basis of its protocol. Hence, three different datasets for TCP, UDP, ICMP have been created from the original dataset. The size of the newly created datasets is around 5MB with a total of 33K rows. Below are the snapshots of the three csv files.

dataset_tcp.csv

dt	switch	src	dst	pktcount	bytecount	dur	dur_nsec	tot_dur	flows	packetins	pktperflow	byteperflo	pktrate	Pairflow	Protocol	port_no	tx_bytes	rx_bytes	tx_kbps	rx_kbps
10659	5	10.0.0.4	10.0.0.3	1072	57888	3	9.86E+08	3.99E+09	6	4357	0	0	0	1	TCP	1	3752	65312	0	17
10659	5	10.0.0.4	10.0.0.3	1072	57888	3	9.86E+08	3.99E+09	6	4357	0	0	0	1	TCP	2	77589046	4219668	4204	312
10659	5	10.0.0.4	10.0.0.3	1072	57888	3	9.86E+08	3.99E+09	6	4357	0	0	0	1	TCP	3	4219561	77524906	312	4187
10659	8	10.0.0.13	10.0.0.3	31937	1724598	54	1.08E+08	5.41E+10	3	4357	18116	978264	603	1	TCP	1	3744	900356	0	131
10659	8	10.0.0.13	10.0.0.3	31937	1724598	54	1.08E+08	5.41E+10	3	4357	18116	978264	603	1	TCP	2	901184	836870	140	131
10659	8	10.0.0.13	10.0.0.3	31937	1724598	54	1.08E+08	5.41E+10	3	4357	18116	978264	603	1	TCP	3	1738118	901116	262	140
10659	8	10.0.0.13	10.0.0.3	31937	1724598	54	1.08E+08	5.41E+10	3	4357	18116	978264	603	1	TCP	4	3572	3236	0	0
10659	8	10.0.0.3	10.0.0.13	15114	876612	44	5.8E+08	4.46E+10	3	4357	9058	525364	301	1	TCP	1	3744	900356	0	131
10659	8	10.0.0.3	10.0.0.13	15114	876612	44	5.8E+08	4.46E+10	3	4357	9058	525364	301	1	TCP	2	901184	836870	140	131
10659	8	10.0.0.3	10.0.0.13	15114	876612	44	5.8E+08	4.46E+10	3	4357	9058	525364	301	1	TCP	3	1738118	901116	262	140
10659	8	10.0.0.3	10.0.0.13	15114	876612	44	5.8E+08	4.46E+10	3	4357	9058	525364	301	1	TCP	4	3572	3236	0	0
10659	2	10.0.0.1	10.0.0.3	89881	1E+08	204	5.2E+08	2.05E+11	11	4357	13258	14622228	441	1	TCP	3	4265894	77630875	325	4216
10659	2	10.0.0.1	10.0.0.3	89881	1E+08	204	5.2E+08	2.05E+11	11	4357	13258	14622228	441	1	TCP	1	2.29E+08	10725558	12065	668
10659	2	10.0.0.1	10.0.0.3	89881	1E+08	204	5.2E+08	2.05E+11	11	4357	13258	14622228	441	1	TCP	2	6465300	1.51E+08	343	7849
10659	2	10.0.0.3	10.0.0.1	63473	4189590	204	5.14E+08	2.05E+11	11	4357	9699	640134	323	1	TCP	3	4265894	77630875	325	4216
10659	2	10.0.0.3	10.0.0.1	63473	4189590	204	5.14E+08	2.05E+11	11	4357	9699	640134	323	1	TCP	1	2.29E+08	10725558	12065	668
10659	2	10.0.0.3	10.0.0.1	63473	4189590	204	5.14E+08	2.05E+11	11	4357	9699	640134	323	1	TCP	2	6465300	1.51E+08	343	7849
10659	2	10.0.0.11	10.0.0.3	68709	75647074	154	4.34E+08	1.54E+11	11	4357	13286	14625100	442	1	TCP	3	4265894	77630875	325	4216
10659	2	10.0.0.11	10.0.0.3	68709	75647074	154	4.34E+08	1.54E+11	11	4357	13286	14625100	442	1	TCP	1	2.29E+08	10725558	12065	668
10659	2	10.0.0.11	10.0.0.3	68709	75647074	154	4.34E+08	1.54E+11	11	4357	13286	14625100	442	1	TCP	2	6465300	1.51E+08	343	7849
10659	2	10.0.0.3	10.0.0.11	50185	3312378	154	4.3E+08	1.54E+11	11	4357	9695	639882	323	1	TCP	3	4265894	77630875	325	4216
10659	2	10.0.0.3	10.0.0.11	50185	3312378	154	4.3E+08	1.54E+11	11	4357	9695	639882	323	1	TCP	1	2.29E+08	10725558	12065	668
10659	2	10.0.0.3	10.0.0.11	50185	3312378	154	4.3E+08	1.54E+11	11	4357	9695	639882	323	1	TCP	2	6465300	1.51E+08	343	7849

dataset_udp.csv

dt	switch	src	dst	pktcount	bytecount	dur	dur_nsec	tot_dur	flows	packetins	pktperflow	byteperflo	pktrate	Pairflow	Protocol	port_no	tx_bytes	rx_bytes	tx_kbps	rx_kbps
11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	7.16E+08	1.01E+11	3	1943	13535	14428310	451	0	UDP	3	1.44E+08	3917	0	0
11605	1	10.0.0.1	10.0.0.8	126395	1.35E+08	280	7.34E+08	2.81E+11	2	1943	13531	14424046	451	0	UDP	4	3842	3520	0	0
11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	7.44E+08	2.01E+11	3	1943	13534	14427244	451	0	UDP	1	3795	1242	0	0
11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	7.44E+08	2.01E+11	3	1943	13534	14427244	451	0	UDP	2	3688	1492	0	0
11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	7.44E+08	2.01E+11	3	1943	13534	14427244	451	0	UDP	3	3413	3665	0	0
11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	7.44E+08	2.01E+11	3	1943	13534	14427244	451	0	UDP	1	3795	1402	0	0
11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	7.16E+08	1.01E+11	3	1943	13535	14428310	451	0	UDP	4	3665	3413	0	0
11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	7.16E+08	1.01E+11	3	1943	13535	14428310	451	0	UDP	1	3775	1492	0	0
11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	7.16E+08	1.01E+11	3	1943	13535	14428310	451	0	UDP	2	3845	1402	0	0
11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	7.44E+08	2.01E+11	3	1943	13534	14427244	451	0	UDP	4	3.55E+08	4295	16578	0
11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	7.16E+08	1.01E+11	3	1943	13535	14428310	451	0	UDP	1	3775	1242	0	0
11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	7.44E+08	2.01E+11	3	1943	13534	14427244	451	0	UDP	2	3413	3665	0	0
11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	7.16E+08	1.01E+11	3	1943	13535	14428310	451	0	UDP	1	4047	1.44E+08	0	0
11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	7.16E+08	1.01E+11	3	1943	13535	14428310	451	0	UDP	4	3665	3413	0	0
11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	7.16E+08	1.01E+11	3	1943	13535	14428310	451	0	UDP	2	5.81E+08	2586	19164	0
11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	7.16E+08	1.01E+11	3	1943	13535	14428310	451	0	UDP	4	3665	3413	0	0
11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	7.16E+08	1.01E+11	3	1943	13535	14428310	451	0	UDP	2	3795	1402	0	0
11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	7.16E+08	1.01E+11	3	1943	13535	14428310	451	0	UDP	2	3795	1492	0	0
11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	7.16E+08	1.01E+11	3	1943	13535	14428310	451	0	UDP	2	4047	1.93E+08	0	6307
11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	7.16E+08	1.01E+11	3	1943	13535	14428310	451	0	UDP	1	3879	48410560	0	3838
11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	7.16E+08	1.01E+11	3	1943	13535	14428310	451	0	UDP	2	4055	96412666	0	3838
11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	7.16E+08	1.01E+11	3	1943	13535	14428310	451	0	UDP	3	3413	3665	0	0
11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	7.44E+08	2.01E+11	3	1943	13534	14427244	451	0	UDP	1	3570	1492	0	0
11455	1	10.0.0.2	10.0.0.8	103866	1.11E+08	230	7.47E+08	2.31E+11	3	1943	13533	14426178	451	0	UDP	1	3775	1242	0	0
11605	1	10.0.0.1	10.0.0.8	126395	1.35E+08	280	7.34E+08	2.81E+11	2	1943	13531	14424046	451	0	UDP	3	4766	3.53E+08	0	6400

dataset_icmp.csv

dt	switch	src	dst	pktcount	bytecount	dur	dur_nsec	tot_dur	flows	packetins	pktperflow	byteperflo	pktrate	Pairflow	Protocol	port_no	tx_bytes	rx_bytes
26657	8	10.0.0.12	10.0.0.7	20	1960	21	2.01E+08	2.12E+10	3	10	0	0	0	1	ICMP	3	5327	5327
26657	8	10.0.0.12	10.0.0.7	20	1960	21	2.01E+08	2.12E+10	3	10	0	0	0	1	ICMP	1	5457	3104
26657	8	10.0.0.12	10.0.0.7	20	1960	21	2.01E+08	2.12E+10	3	10	0	0	0	1	ICMP	4	3227	3185
26657	8	10.0.0.12	10.0.0.7	20	1960	21	2.01E+08	2.12E+10	3	10	0	0	0	1	ICMP	2	3357	1122
26657	8	10.0.0.7	10.0.0.12	20	1960	21	43000000	2.1E+10	3	10	0	0	0	1	ICMP	3	5327	5327
26657	8	10.0.0.7	10.0.0.12	20	1960	21	43000000	2.1E+10	3	10	0	0	0	1	ICMP	1	5457	3104
26657	8	10.0.0.7	10.0.0.12	20	1960	21	43000000	2.1E+10	3	10	0	0	0	1	ICMP	4	3227	3185
26657	8	10.0.0.7	10.0.0.12	20	1960	21	43000000	2.1E+10	3	10	0	0	0	1	ICMP	2	3357	1122
26657	5	10.0.0.12	10.0.0.7	20	1960	21	1.65E+08	2.12E+10	3	10	0	0	0	1	ICMP	3	5327	5327
26657	5	10.0.0.12	10.0.0.7	20	1960	21	1.65E+08	2.12E+10	3	10	0	0	0	1	ICMP	1	3247	962
26657	5	10.0.0.12	10.0.0.7	20	1960	21	1.65E+08	2.12E+10	3	10	0	0	0	1	ICMP	2	5327	5327
26657	5	10.0.0.7	10.0.0.12	20	1960	21	1.13E+08	2.11E+10	3	10	0	0	0	1	ICMP	3	5327	5327
26657	5	10.0.0.7	10.0.0.12	20	1960	21	1.13E+08	2.11E+10	3	10	0	0	0	1	ICMP	1	3247	962
26657	5	10.0.0.7	10.0.0.12	20	1960	21	1.13E+08	2.11E+10	3	10	0	0	0	1	ICMP	2	5327	5327
26657	6	10.0.0.12	10.0.0.7	20	1960	21	1.7E+08	2.12E+10	3	10	0	0	0	1	ICMP	1	3357	962
26657	6	10.0.0.12	10.0.0.7	20	1960	21	1.7E+08	2.12E+10	3	10	0	0	0	1	ICMP	2	5327	5327
26657	6	10.0.0.12	10.0.0.7	20	1960	21	1.7E+08	2.12E+10	3	10	0	0	0	1	ICMP	3	5327	5327
26657	6	10.0.0.7	10.0.0.12	20	1960	21	77000000	2.11E+10	3	10	0	0	0	1	ICMP	1	3357	962
26657	6	10.0.0.7	10.0.0.12	20	1960	21	77000000	2.11E+10	3	10	0	0	0	1	ICMP	2	5327	5327
26657	6	10.0.0.7	10.0.0.12	20	1960	21	77000000	2.11E+10	3	10	0	0	0	1	ICMP	3	5327	5327
26657	4	10.0.0.12	10.0.0.7	20	1960	21	1.45E+08	2.11E+10	3	10	0	0	0	1	ICMP	3	5327	5327
26657	4	10.0.0.12	10.0.0.7	20	1960	21	1.45E+08	2.11E+10	3	10	0	0	0	1	ICMP	2	3227	3255
26657	4	10.0.0.12	10.0.0.7	20	1960	21	1.45E+08	2.11E+10	3	10	0	0	0	1	ICMP	1	5457	3104

4 Code Execution

4.1 Packages required

Programming Language: Python
The following code was developed on jupyter notebook. The packages that were needed for building the models are as mentioned below:

- Matplotlib 3.4.3
- Numpy 1.20.3
- Pandas 1.3.4
- Scikit-learn 0.24
- StandardScalar
- train_test_split
- SVC
- KMeans
- PCA
- Sklearn.metrics
- RandomForestClassifier

4.2 Pre-processing

Standardization is a process to scale the data to a standard range as the original data might be widespread. We use this in order for the machine learning models to perform better.
The standardization of all three datasets was done as shown below:

TCP

```
# TCP
features_tcp = tcp.columns.tolist()[1:-1]
tcp_X = tcp.loc[:, features_tcp].values
tcp_y = tcp.loc[:,['label']].values
tcp_train_x, tcp_test_x, tcp_train_y, tcp_test_y = train_test_split(
    tcp_X, tcp_y, test_size=1/7.0, random_state=0
)

scaler = StandardScaler()
scaler.fit(tcp_train_x)
tcp_train_x = scaler.transform(tcp_train_x)
tcp_test_x = scaler.transform(tcp_test_x)
```

UDP

```
# UDP
features_udp = udp.columns.tolist()[1:-1]
udp_X = udp.loc[:, features_udp].values
udp_y = udp.loc[:,['label']].values
udp_train_x, udp_test_x, udp_train_y, udp_test_y = train_test_split(
    udp_X, udp_y, test_size=1/7.0, random_state=0
)

scaler = StandardScaler()
scaler.fit(udp_train_x)
udp_train_x = scaler.transform(udp_train_x)
udp_test_x = scaler.transform(udp_test_x)
```

ICMP

```
# ICMP
features_icmp = icmp.columns.tolist()[1:-1]
icmp_X = icmp.loc[:, features_icmp].values
icmp_y = icmp.loc[:,['label']].values
icmp_train_x, icmp_test_x, icmp_train_y, icmp_test_y = train_test_split(
    icmp_X, icmp_y, test_size=1/7.0, random_state=0
)

scaler = StandardScaler()
scaler.fit(icmp_train_x)
icmp_train_x = scaler.transform(icmp_train_x)
imcp_test_x = scaler.transform(icmp_test_x)
```

The distribution of data is scaled using StandardScaler so that the mean of the observed values is 0 and the standard deviation is 1.

The next step in pre-processing of these datasets is Principle Component Analysis (PCA). Using PCA, we can train the model with less features while maintaining almost same accuracy. This is accomplished by determining which characteristics have a bigger influence on accuracy as well as which features have a very minimal training-related contribution. We are use PCA from the sklearn package, the most well-known Python machine learning library. Below shown is a preview of how PCA was applied.

TCP

```
# TCP
pca = PCA(.95)
pca.fit(tcp_train_x)
tcp_train_x = pca.transform(tcp_train_x)
tcp_test_x = pca.transform(tcp_test_x)
print(f'TCP features reduced from {len(features_tcp)} to {pca.n_components_} after PCA')
```

```
TCP features reduced from 19 to 12 after PCA
```

UDP

```
# UDP
pca = PCA(.95)
pca.fit(udp_train_x)
udp_train_x = pca.transform(udp_train_x)
udp_test_x = pca.transform(udp_test_x)
print(f'UDP features reduced from {len(features_udp)} to {pca.n_components_} after PCA')
```

```
UDP features reduced from 19 to 10 after PCA
```

ICMP

```
# ICMP
pca = PCA(.95)
pca.fit(icmp_train_x)
icmp_train_x = pca.transform(icmp_train_x)
icmp_test_x = pca.transform(icmp_test_x)
print(f'ICMP features reduced from {len(features_icmp)} to {pca.n_components_} after PCA')
```

```
ICMP features reduced from 19 to 12 after PCA
```

On applying PCA, we could successfully bring down the features to about 12 from 19. We have taken 0.95 as the value for the no. of components paraneter. It simply lets scikit-learn know to select the fewest principal components necessary to preserve 95% of the variance.

4.3 Implementation

The purpose of this research is to find out which model is the most accurate in detecting DDoS attacks, especially focusing on TCP Syn, UDP flood and ICMP attacks. We have compared three models namely, SVM, KMeans and Random Forest against our datasets. The model training and testing for SVM is as follows:

```
# tcp
clf = svm.SVC(kernel='linear')
clf.fit(tcp_train_x, tcp_train_y.ravel())
tcp_pred_y = clf.predict(tcp_test_x)
print("Accuracy:",metrics.accuracy_score(tcp_test_y.ravel(), tcp_pred_y))
```

Here, the ravel() function is used to transform the multi-dimensional array into a contiuous flatttened array. The kernel is set to linear in order to only classify linearly separated data.

```
# udp
clf = svm.SVC(kernel='linear')
clf.fit(udp_train_x, udp_train_y.ravel())
udp_pred_y = clf.predict(udp_test_x)
print("Accuracy:",metrics.accuracy_score(udp_test_y.ravel(), udp_pred_y))
```

```
# icmp
clf = svm.SVC(kernel='linear')
clf.fit(icmp_train_x, icmp_train_y.ravel())
icmp_pred_y = clf.predict(icmp_test_x)
print("Accuracy:",metrics.accuracy_score(icmp_test_y.ravel(), icmp_pred_y))
```

Below is the model training testing for RandomForest classifier. As we know that to get better performance, the no. of trees has to be high. To accomplish that, the value of n_estimators is kept as 100 so that there are more trees to take the average of predictions from.

```
#randomforest tcp
clf = RandomForestClassifier(max_depth=None, random_state=0, n_estimators=100)

clf.fit(tcp_train_x, tcp_train_y.ravel())
tcp_pred_y = clf.predict(tcp_test_x)
print("Accuracy:",metrics.accuracy_score(tcp_test_y.ravel(), tcp_pred_y))
```

```
clf = RandomForestClassifier(max_depth=None, random_state=0, n_estimators=100)

clf.fit(udp_train_x, udp_train_y.ravel())
udp_pred_y = clf.predict(udp_test_x)
print("Accuracy:",metrics.accuracy_score(udp_test_y.ravel(), udp_pred_y))
```

```
#randomforest icmp

clf = RandomForestClassifier(max_depth=None, random_state=0, n_estimators=100)

clf.fit(icmp_train_x, icmp_train_y.ravel())
icmp_pred_y = clf.predict(icmp_test_x)
print("Accuracy:",metrics.accuracy_score(icmp_test_y.ravel(), icmp_pred_y))
```

For KMeans, we have taken two clusters to classify whether it is a DDoS attack or not. As shown below, the training and testing for the KMeans clustering is done

TCP

```
# KMeans clustering for tcp
kmeans = KMeans(n_clusters=2)
kmeans.fit(tcp_train_x)
```

```
KMeans(n_clusters=2)
```

```
correct = 0
for i in range(len(tcp_test_x)):
    predict_me = np.array(tcp_test_x[i].astype(float))
    predict_me = predict_me.reshape(-1, len(predict_me))
    prediction = kmeans.predict(predict_me)
    if prediction[0] == tcp_test_y[i]:
        correct += 1

print(correct/len(tcp_test_x))
```

UDP

```
# KMeans clustering for udp
kmeans = KMeans(n_clusters=2)
kmeans.fit(tcp_train_x)
```

```
KMeans(n_clusters=2)
```

```
correct = 0
for i in range(len(udp_test_x)):
    predict_me = np.array(udp_test_x[i].astype(float))
    predict_me = predict_me.reshape(-1, len(predict_me))
    prediction = kmeans.predict(predict_me)
    if prediction[0] == udp_test_y[i]:
        correct += 1

print(correct/len(udp_test_x))
```

ICMP

```
# KMeans clustering for icmp
kmeans = KMeans(n_clusters=2)
kmeans.fit(icmp_train_x)
```

```
KMeans(n_clusters=2)
```

```
correct = 0
for i in range(len(icmp_test_x)):
    predict_me = np.array(icmp_test_x[i].astype(float))
    predict_me = predict_me.reshape(-1, len(predict_me))
    prediction = kmeans.predict(predict_me)
    if prediction[0] == icmp_test_y[i]:
        correct += 1

print(correct/len(icmp_test_x))
```