PROJECT REPORT

(Project Term AUG-DEC 2021)

Malware Identification using Deep Learning

Submitted by

Putsala Sai Sri Ganesh 11914316

Course Code: INT246

Under the Guidance of

Dr. Sagar Pande

School of Computer Science and Engineering

Declaration

We hereby declare that the project work entitled Malware Identification using deep learning is an authentic record of our own work carried out as requirement of Project for the award of B. Tech degree in CSE from Lovely Professional University, Phagwara, under the guidance of Dr. Sagar Pande, during August to November 2021.All the Information furnished in this project report is based on our own intensive work and is genuine.

Name of student: Putsala Sai Sri Ganesh

Registration Number: 11914316

Signature of the students: 

ACKNOWLEDGEMENT

The chance of doing this Project got for me in the form of academic task. I would like to express my mentor and advisor prof. Sagar Pande for the continuous support on my project, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of project. I could not imagine a better advisor and mentor for my project.

With pleasure,

Putsala Sai Sri Ganesh

Registration. No: 11914316

INDEX

INTRODUCTION

Abstract

Malware Identification

Malware: Malware is nothing but an intrusive, problematic software that is designed to damage and destroy the computers and computer systems. Some of the malwares are virus, worms, Trojan, spyware, adware and ransomware.

“““Malware Identification refers to the process of detecting the presence of malware on a host system or of identifying whether a specific program is suffering with virus or not.**”””**

Why it is important to detect Malware?

Malware is one of the most serious security problems on the Internet today. In fact, most Internet problems such as spam e-mails and denial of service attacks have malware as their underlying cause. That is, computers that are compromised with malware are often networked together to form botnets, and many attacks are launched using these malicious, attacker-controlled networks.

For the detection of malware and to destroy a malware identification software is in need. Time to time new malware are generating and to identify these new malwares new techniques need to create.

Intrusion Detection System (IDS) :

Intrusion Detection Systems (IDS) provide network security software applications by continuously monitoring network traffic and classifying the connections as normal or malicious. IDS are categorized into two types based on the responsive nature - Passive IDS and Active IDS. A passive IDS is designed to identify and block the malicious and malware attacks manually by human experts whereas active IDS is designed to identify and block the malware attacks using a software automatically.

IDS are also categorized into Signature based IDS and Anomaly Based IDS. In the Signature based IDS, there exists a database which contains details about all known malware attacks against which each network traffic connection is validated to identify the malicious nature if it exists. This type of IDS is costly and has to keep updating new types of attacks frequently. Anomaly based IDS is a behavior-based system where any deviation from the normal network traffic patterns will be reported using pattern-recognition techniques. In this research paper, a proof-of-concept for a machine learning based Anomaly based intrusion detection will be evaluated on a benchmark intrusion detection dataset.

Motivation

The Internet of Things (IoT) has changed the way devices communicate across a network. There are over 30 billion devices connected to each other in today's world where their communication happens over the network. Due to the variety, velocity and vast nature of this traffic, the traditional intrusion detection systems are not sufficient to identify the malware patterns.

Why Deep Learning?

In present days deep learning has dominated the various computer vision tasks. Deep learning techniques enabled rapid progress in this competition and even surpassing human performance in many of them.

Unlike more traditional methods of machine learning techniques, deep learning classifiers are trained through feature learning rather than task-specific algorithms. What this means is that this deep learning model can extract best important features from dataset. And using these king of techniques gives greatest accuracy score.

Objectives

- The main objective is to detect the malware attacks and to stop them.
- The objectives of this project is to build a model using deep learning techniques.
- And to clean, explore and analysis the dataset.
- To plot graphs between features of dataset with target variable.
- To normalize all the features.
- To Build the models that train and test the data set.
- To check the errors and confusion matrix.
- To plot the graphs that tells about the relationship between all the models.
- To select the suitable model among all model based on the accurancy.
- And to select final features which are important for the model.

Advantages

The advantage of using deep learning to build model for malware identification is that deep learning approach is it ability to execute feature engineering by itself.

In this approach an algorithms scans the data to identify the features which correlate and combine them to promote faster leaning without being told to do so explicitly.
The algorithms in deep learning makes the model with greatest accuracy. And with the best suitable malware identification model we can stop the various malwares.

Limitations

In this project deep learning is used and it is extremely expensive to train complex data models. And this model is only applicable for large data set.

Applications

Malware Identification model using deep learning can use in various systems where viruses will enter such as

- To stop Cybercriminals to steal the peoples personal data
- To protect bank card details and online money
- And used in military defense offices to protect countries information.

Novelty

The main novelty is to check and build best model.

- Checking with more layers and neurons.
- Seeing results after changing optimizers and number of hidden layers.
- Building a model which gives greatest accuracy score as 99.95%.

Problem Statement

In this research, KDD Cup 1999 dataset will be used to build a machine learning based intrusion detection problem to predict (classify) whether a network connection is ”normal” or ”abnormal”. As this is a categorical prediction, this is a binary classification problem.

DATASET DESCRIPTION AND EXPLORATION

In this research, the KDD Cup 1999 dataset has been chosen for the data analysis and model building. This dataset was used in the fifth international conference for the data mining and knowledge discovery competition and contains a huge variety of network intrusion data that is simulated in an environment equipped with a military network setting.

The original source of the dataset is from the official KDD website and can also be found in the kaggle website. The dataset contains about 494,000 records and 41 features (columns) which contains network traffic details like source bytes, destination connection information, type of attacks and so on.

Target Variable “label” distribution

The target variable- “label” contains all different types of malware attacks and also the value “normal” (i.e., not an attack) as shown below.

’normal’, ’buffer overflow’, ‘loadmodule’, ’perl’, ’neptune’, ’smurf’, ’guess passwd’, ’pod’, ’teardrop’, ’portsweep’, ’ipsweep’, ’land’, ’ftp write’, ’back’, ’imap’, ’satan’, ’phf’, ’nmap’, ’multihop’, ’warezmaster’, ’warezclient’, 'spy', 'rootkit'.

All malware attacks are grouped (transformed) to the value “abnormal” to make this problem a binary classification problem instead of a multi-class classification. The target variable distribution can be found below (see Figure 1).

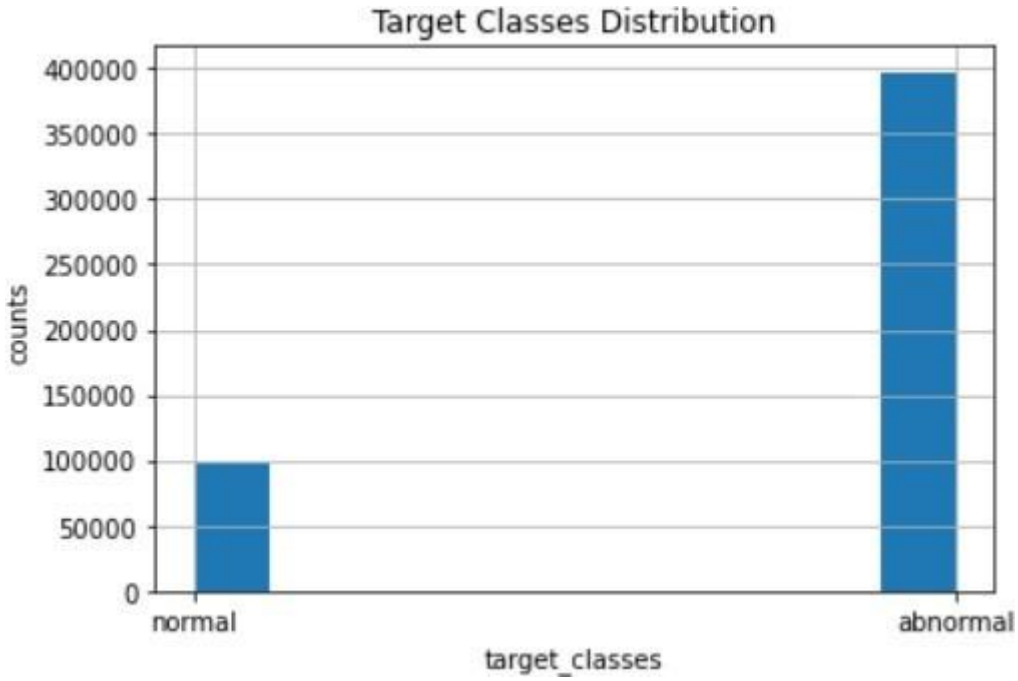


Figure 1: Target Variable ”label” Distribution.

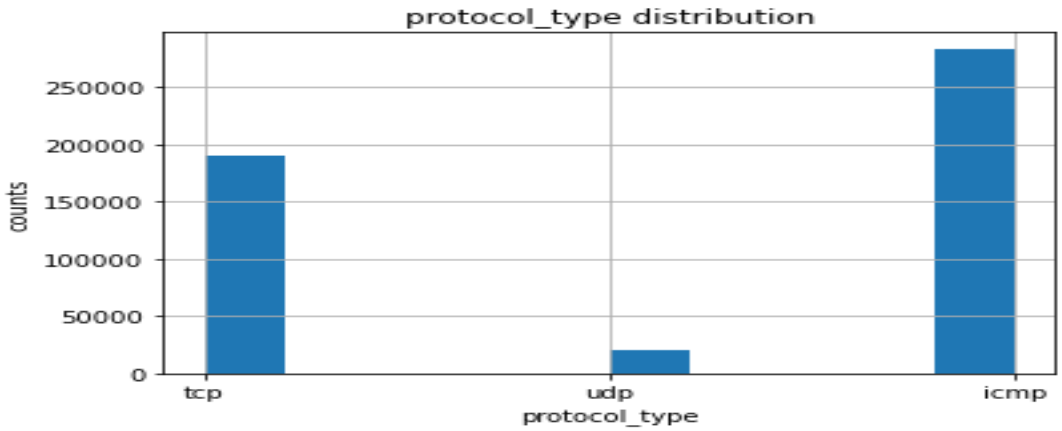
Categorical Columns Distributions

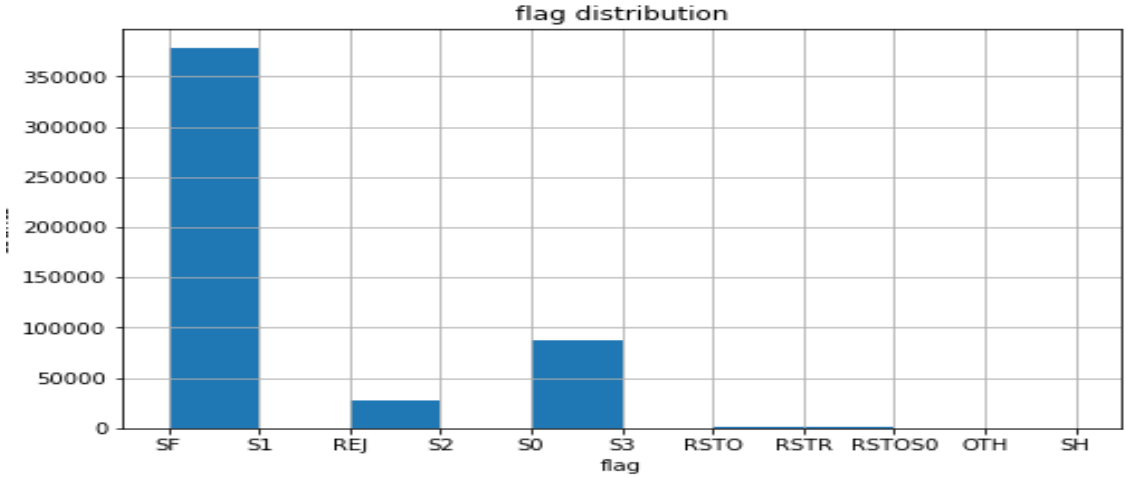
Below are the categorical columns in this dataset and their corresponding distribution plots. ’protocol type’, ’service’, ’flag’, ’land’, ’logged in’, ’is host login’, ’is guest login’.

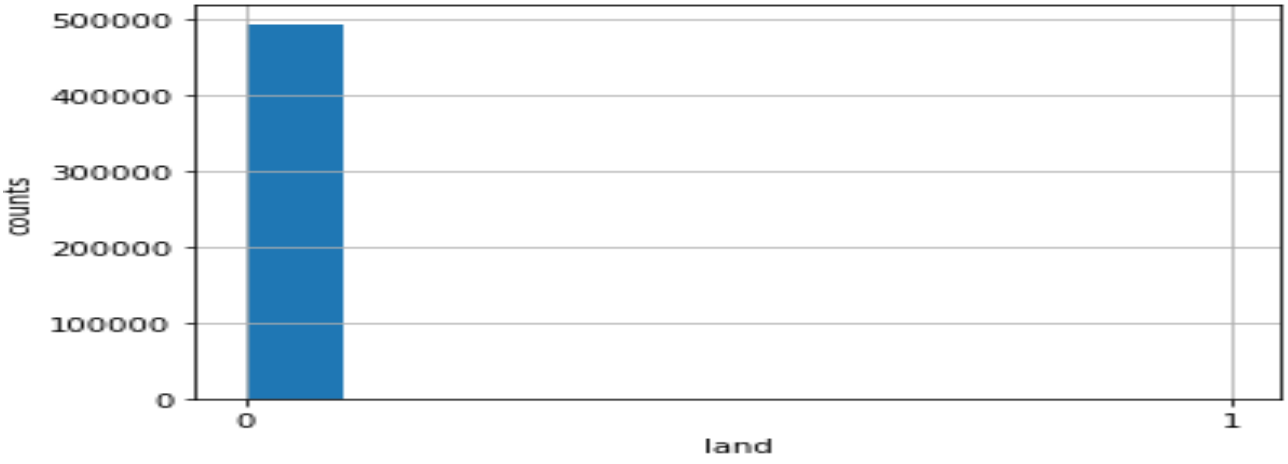
Categorical variables statistics can be found in the below table:

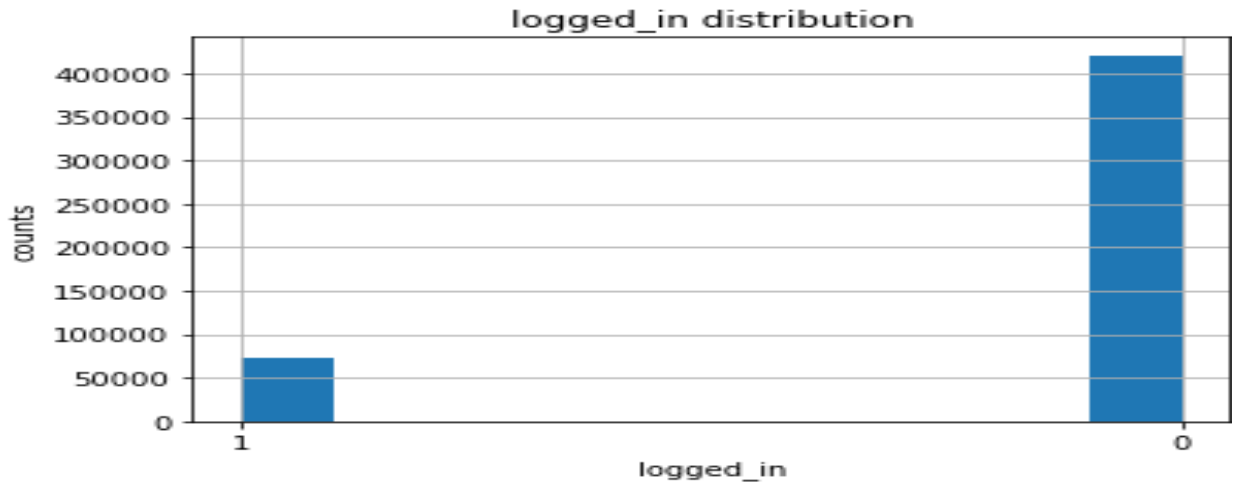
Table 1: Categorical Variables Statistics.

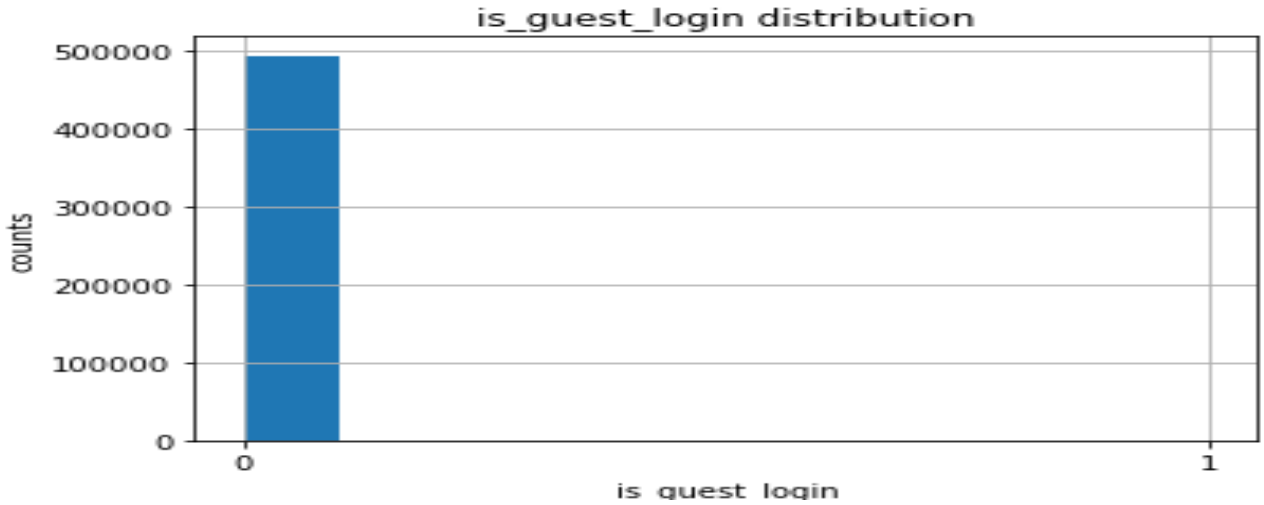
Stats	protocol_type	service	flag	land	logged_in	is_host_login	is_guest_login
count	494020	494020	494020	494020	494020	494020	494020
unique	3	66	11	2	2	1	2
top	icmp	ecr_i	SF	0	0	0	0
freq	283602	281400	378439	493998	420784	494020	493335

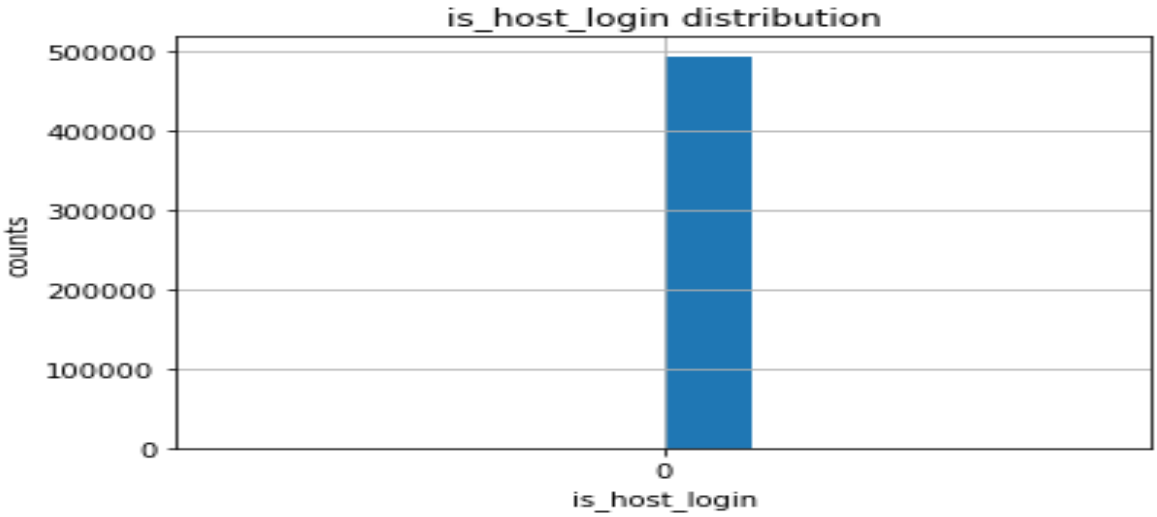

protocol_type distribution
250000
200000
150000
100000
50000
0
counts
tcp
udp
icmp
protocol_type


flag distribution
350000
300000
250000
200000
150000
100000
50000
0
SF
S1
REJ
S2
S0
S3
RSTO
RSTR
RSTOS0
OTH
SH
flag


500000
400000
300000
200000
100000
0
counts
0
1
land


logged_in distribution
400000
350000
300000
250000
200000
150000
100000
50000
0
counts
1
0
logged_in


is_guest_login distribution
500000
400000
300000
200000
100000
0
counts
0
1
is_guest_login


is_host_login distribution
500000
400000
300000
200000
100000
0
counts
0
is_host_login

Continuous Columns

Below are the continuous columns in this dataset and their corresponding distribution plots. ’duration’, ’src bytes’, ’dst bytes’, ’wrong fragment’, ’urgent’, ’hot’, ’num failed logins’, ’num compromised’, ’root shell’, ’su attempted’, ’num root’, ’num file creations’, ’num shells’, ’num access files’, ’num outbound cmds’, ’count’, ’srv count’, ’serror rate’, ’srv serror rate’, ’rerror rate’, ’srv rerror rate’, ’same srv rate’, ’diff srv rate’, ’srv diff host rate’, ’dst host count’, ’dst host srv count’, ’dst host same srv rate’, ’dst host diff srv rate’, ’dst host same src port rate’, ’dst host srv diff host rate’, ’dst host serror rate’, ’dst host srv serror rate’, ’dst host rerror rate’, ’dst host srv rerror rate’.

Continuous variables statistics can be found in the below table:

Table 2: Categorical Variables Statistics.

Stats	count	mean	std	min	25%	50%	75%	max
duration	494020	47.9794	707.7472	0	0	0	0	58329
src bytes	494020	3025.616	988219.1	0	45	520	10326.93E+08	
dst bytes	494020	868.5308	33040.03	0	0	0	0	5155468
wrong fragment	494020	0.006433	0.134805	0	0	0	0	3
urgent	494020	1.42E-05	0.00551	0	0	0	0	3
hot	494020	0.034519	0.782103	0	0	0	0	30
num failed logins	494020	0.000152	0.01552	0	0	0	0	5
num compromised	494020	0.010212	1.798328	0	0	0	0	884
root shell	494020	0.000111	0.010551	0	0	0	0	1
su attempted	494020	3.64E-05	0.007793	0	0	0	0	2
num root	494020	0.011352	2.01272	0	0	0	0	993
num file creations	494020	0.001083	0.096416	0	0	0	0	28
num shells	494020	0.000109	0.01102	0	0	0	0	2
num access files	494020	0.001008	0.036482	0	0	0	0	8
num outbound cmds	494020	0	0	0	0	0	0	0
count	494020	332.2864	213.1471	0	117	510	511	511
srv count	494020	292.9071	246.3227	0	10	510	511	511
serror rate	494020	0.176687	0.380717	0	0	0	0	1
srv serror rate	494020	0.176609	0.381017	0	0	0	0	1
rerror rate	494020	0.057434	0.231624	0	0	0	0	1
srv rerror rate	494020	0.057719	0.232147	0	0	0	0	1

same srv rate	494020	0.791547	0.38819	0	1	1	1	1
diff srv rate	494020	0.020982	0.082206	0	0	0	0	1
srv diff host rate	494020	0.028996	0.142397	0	0	0	0	1
dst host count	494020	232.4712	64.7446	0	255	255	255	255
dst host srv count	494020	188.6661	106.0402	0	46	255	255	255
dst host same srv rate	494020	0.753781	0.41078	0	0.41	1	1	1
dst host diff srv rate	494020	0.030906	0.109259	0	0	0	0.04	1
dst host same src port rate	494020	0.601936	0.481309	0	0	1	1	1
dst host srv diff host rate	494020	0.006684	0.042133	0	0	0	0	1
dst host serror rate	494020	0.176754	0.380593	0	0	0	0	1
dst host srv serror rate	494020	0.176443	0.38092	0	0	0	0	1
dst host rerror rate	494020	0.058118	0.23059	0	0	0	0	1
dst host srv rerror rate	494020	0.057412	0.230141	0	0	0	0	1

Data Normalization

As seen from the above continuous variables distribution in the above section the scale of some of the features (columns) values are not in same range as others. This irregularity will make the machine learning model train poorly. Hence the features need to be normalized so that the optimization during model training will happen in a better way and the model will not be sensitive to the features. This research paper uses the existing normalization function that exists in python's scikit-learn library which can be found here

After applying normalization transformation to each continuous feature, the new distribution plots are showed in code implementation

MODELING

Model that over-fits the entire dataset

To understand the dataset and to estimate the model size(architecture and hyper-parameters), we decided to build a model that over-fits the entire dataset. This is an experiment step to understand what model size would be required to overfit entire data. We performed below steps:

Step-1: Using ENTIRE dataset to ”OVERFIT” the model using vannila (single neuron) logistic regression model to reach accuracy 100 percent or close to 100 percent.

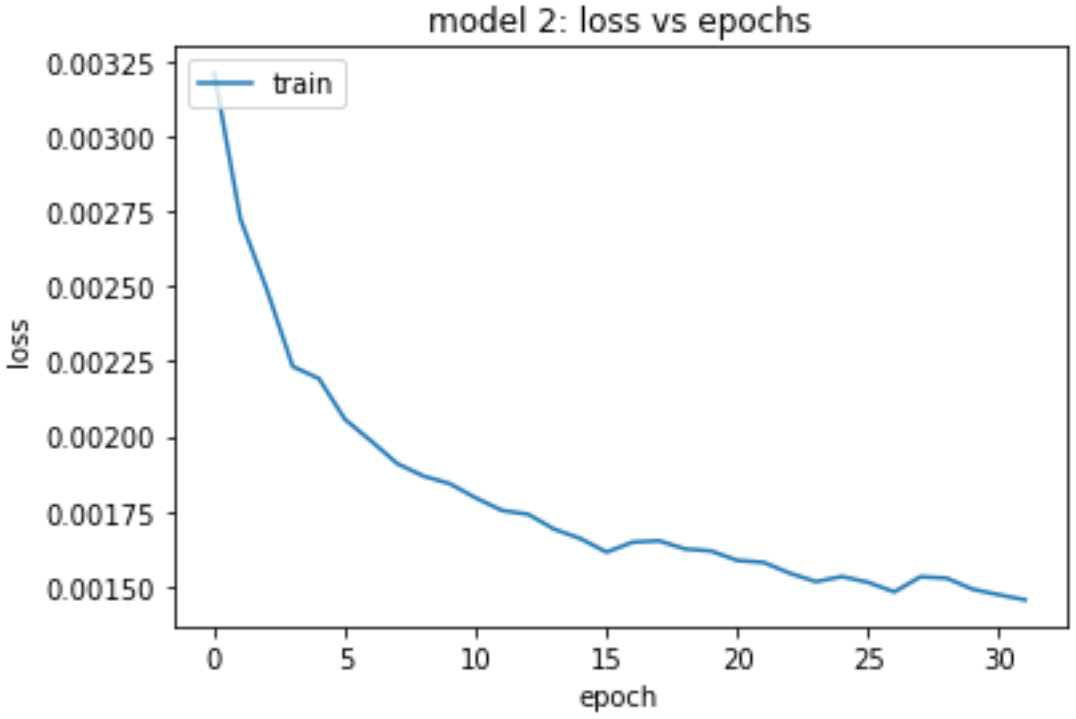
Step-2: If the accuracy did not reach 100 percent, then a bigger architecture model will be designed and modeled to achieve accuracy of 100 percent or close to 100 percent.

Inference

As seen from the above model, the basic logistic regression model has achieved an accuracy of 99.81 with 256 epochs. A seen from the model-2 performance we have achieved 99.96 percent accuracy (close to 100 percent) which means this model did over-fit the dataset. Therefore, a 4-layer neural network model architecture with 8 neurons, 4 neurons and 2 neurons in internal layers is sufficient enough to overfit the entire dataset.

GENERALIZED MODEL BUILDING, SELECTION AND EVALUATION

In the previous section, after exploring the architectures that would result in an over-fitting model, in this section, the focus will be on using a feed-forward neural network (and its architecture variants) to build a best model that would perform well on the validation dataset. The data has been shuffled before performing the experiments and the same dataset has been used for all the models for fair comparison. The code did not implement seeding (which is used for ability to reproduce the model), hence a new run might result in a slight difference in the performances of the models.



The path towards building the best models involves performing experiments with different hyper-parameter settings from a baseline model configuration to the best model (iteration of experiment)

Iteration-0: Baseline Modeling

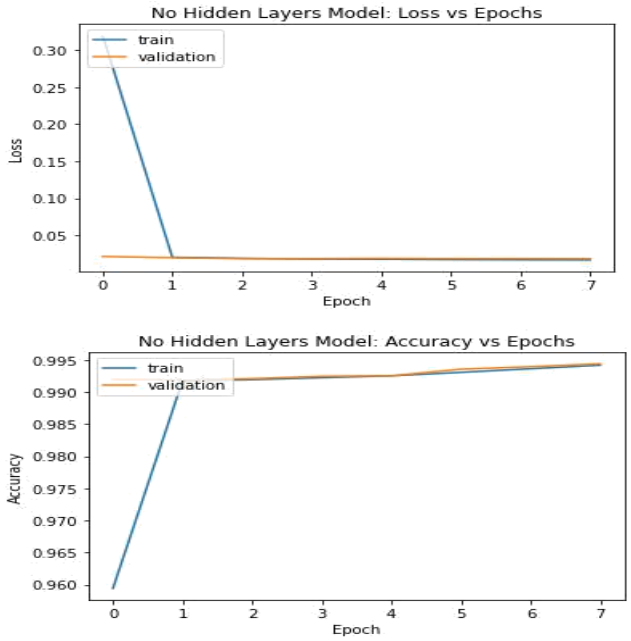
The model built (and selected) in the rst iteration of the experiments acts as a baseline model (as a control) to the next step of experiment models.

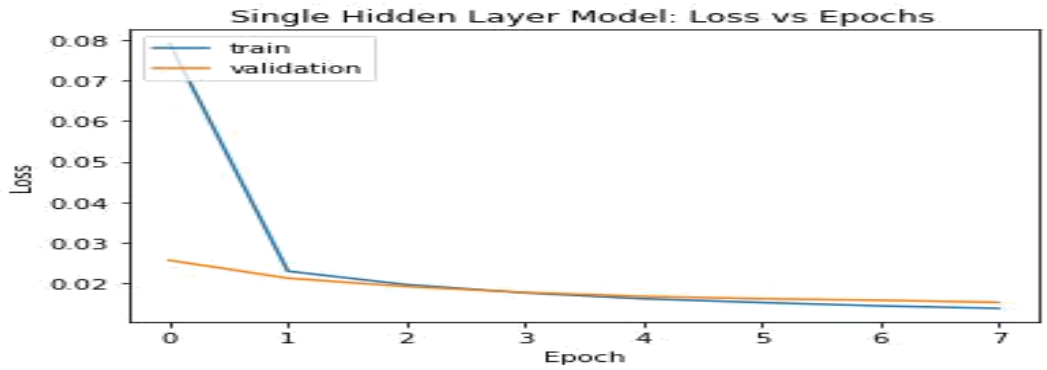
Baseline Model Performance:

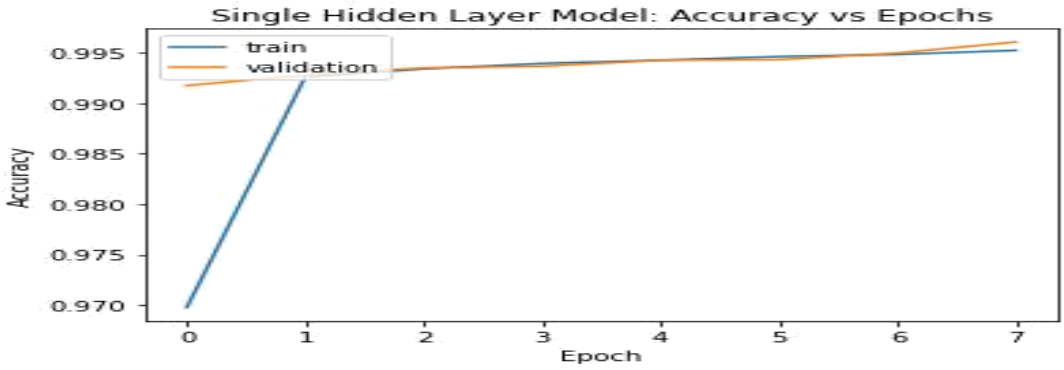
Num of Hidden Layers	Validation Set Loss	Validation Set Accuracy
0	0.0183	99.44
1	0.0153	99.61
2	0.0265	99.43

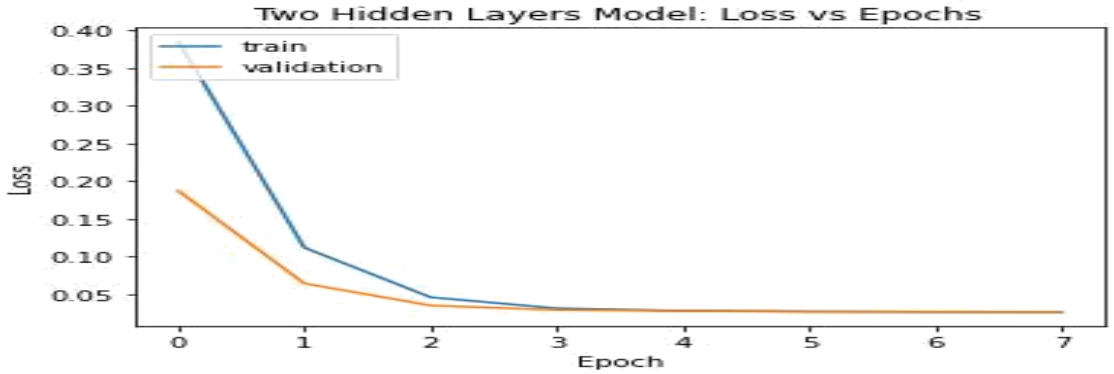
As seen from the above table, the model with 1-hidden-layer outperformed model with no hidden layers and the model with 2 hidden layers. Therefore, 1-hidden-layer will be selected and will be used as a baseline model architecture for the next step of experiments.

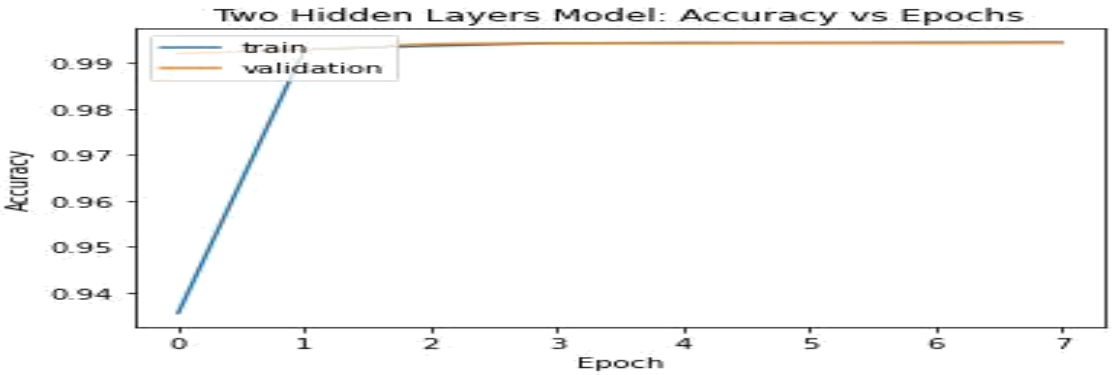
Learning Curves of Iteration-0 Architectures:




Single Hidden Layer Model: Loss vs Epochs
train
validation
Loss
Epoch


Single Hidden Layer Model: Accuracy vs Epochs
train
validation
Accuracy
Epoch


Two Hidden Layers Model: Loss vs Epochs
train
validation
Loss
Epoch


Two Hidden Layers Model: Accuracy vs Epochs
train
validation
Accuracy
Epoch

Iteration-1: Modifying activation functions

The binary classi cation requires ’sigmoid’ activation function in the last layer, however, other activations functions like ReLU and Softmax can be used in interior layers. We built two models with ’ReLU’ and ’Softmax’ activations functions each. We have also changed the loss function to ’categorical-cross-entropy’ when ’Softmax’ is used. We deliberately did not use MSE loss function as this is a classi cation problem. Also, MSE makes a presumption that the underlying data distribution is normally distributed. Below are the modeling performing results in iteration-1.

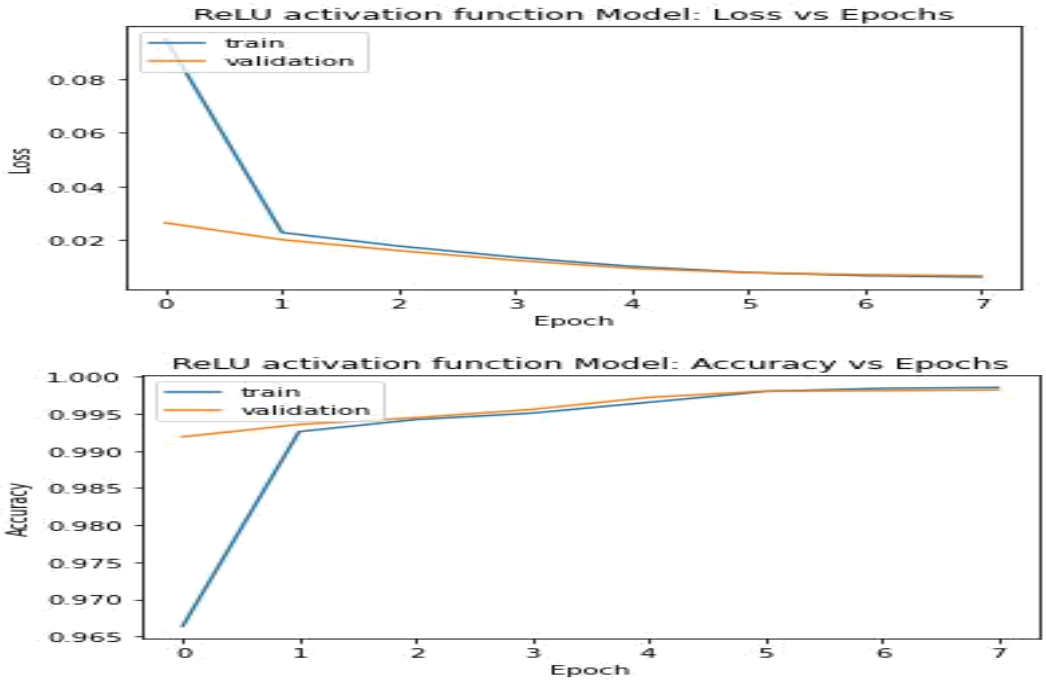
Iteration-1 Model Performances

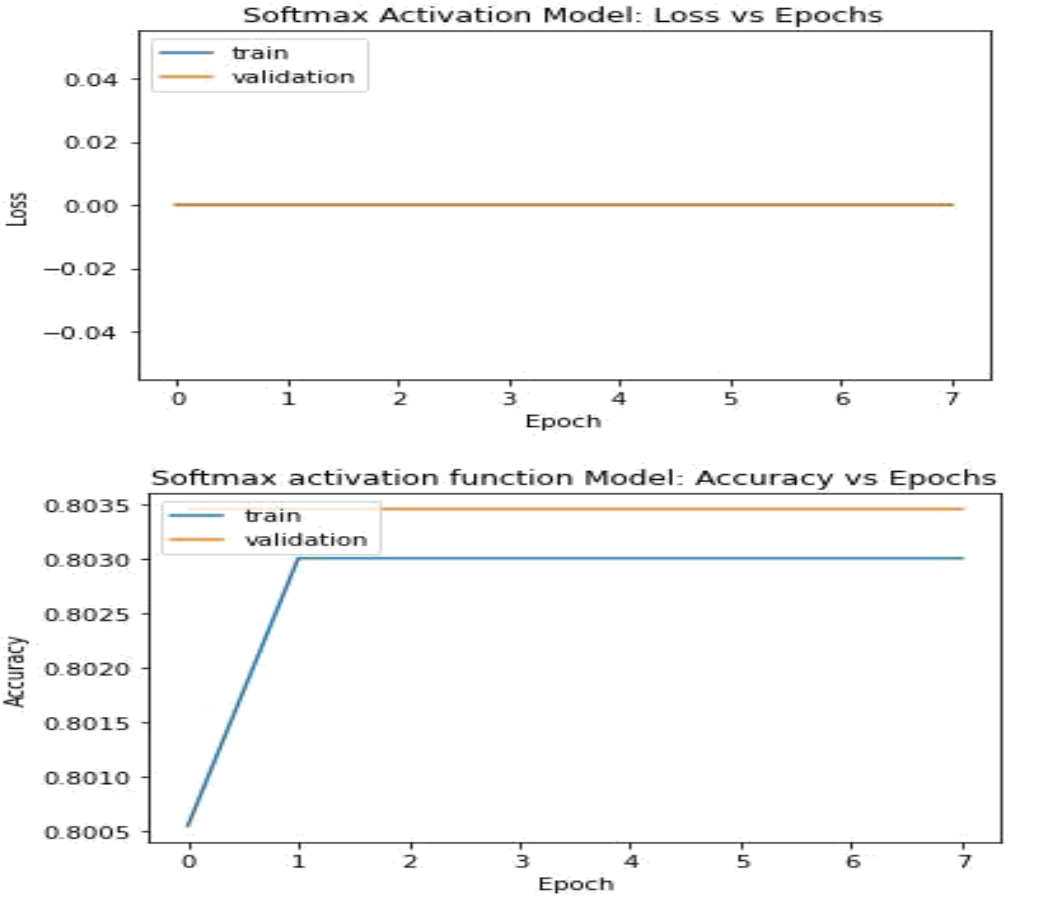
Iteration-1: Model Performances with different activation functions.

Activation Function Input Layer	Loss function	Validation Set Loss	Validation Set Accuracy
ReLU	Binary Cross Entropy	0.0066	99.82
Softmax	Categorical Cross Entropy	0.000001	80.34

As seen from above table, ReLU activation function in the input layer of single hidden layer architecture outperformed with accuracy of 99.61. The next set of experiments will include ReLU activation function in the input layer.

Learning Curves of Iteration-1 Architectures


ReLU activation function Model: Loss vs Epochs
train
validation
Loss
Epoch
ReLU activation function Model: Accuracy vs Epochs
train
validation
Accuracy
Epoch


Softmax Activation Model: Loss vs Epochs
train
validation
Loss
Epoch
Softmax activation function Model: Accuracy vs Epochs
train
validation
Accuracy
Epoch

Iteration-2: Modifying Optimizer functions

The baseline architecture had "rmsprop" optimizer. Therefore, in this iteration, we will be trying out SGD, Adam and Adagrad optimizer functions to understand which optimizer helped the model to achieve optimum minimum in an e ective way.

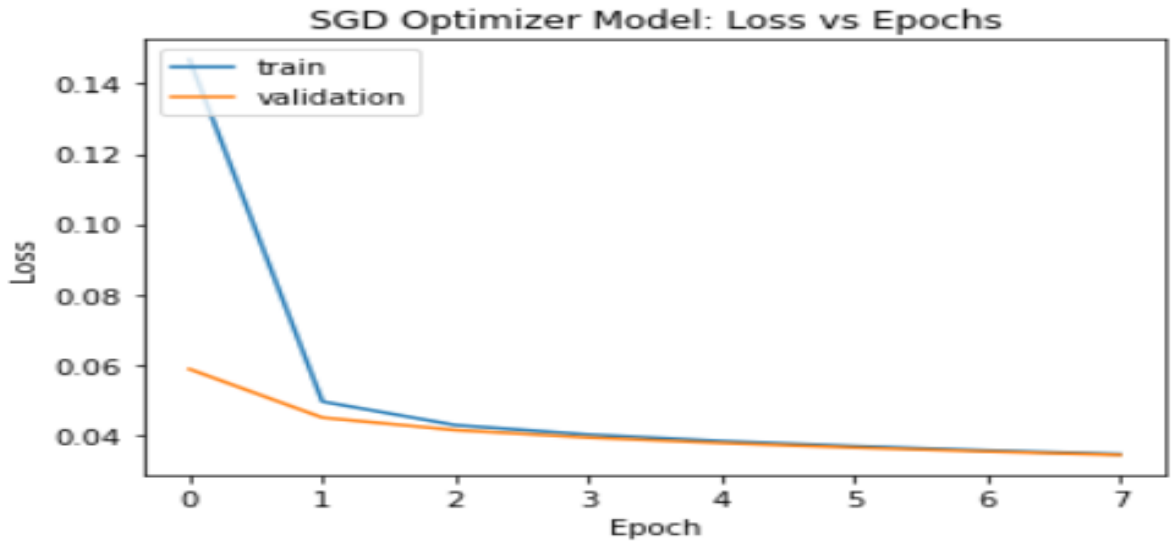
Iteration-2 Model Performances

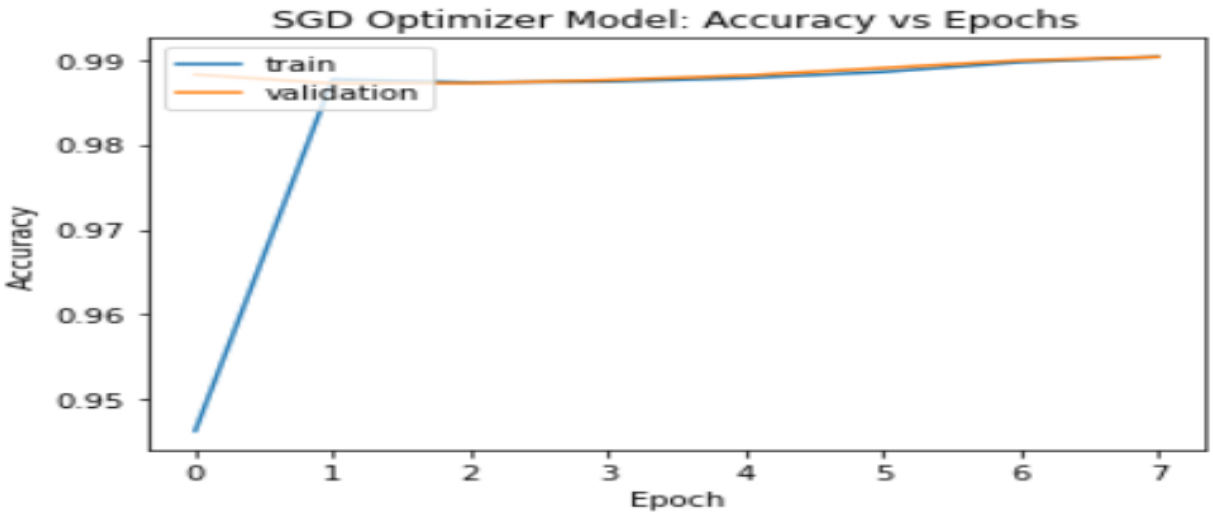
Model Performances with different optimizer functions

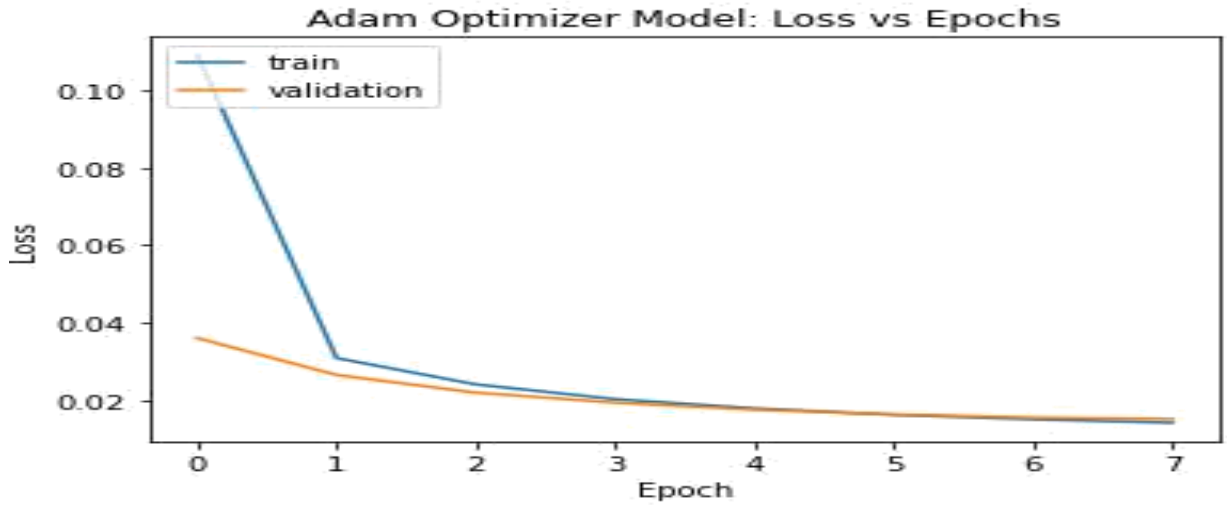
Optimizer	Validation Set Loss	Validation Set Accuracy
rmsprop (current best)	0.0066	99.82
SGD	0.038	98.9
Adam	0.0151	99.48
Adagrad	0.417	98.42

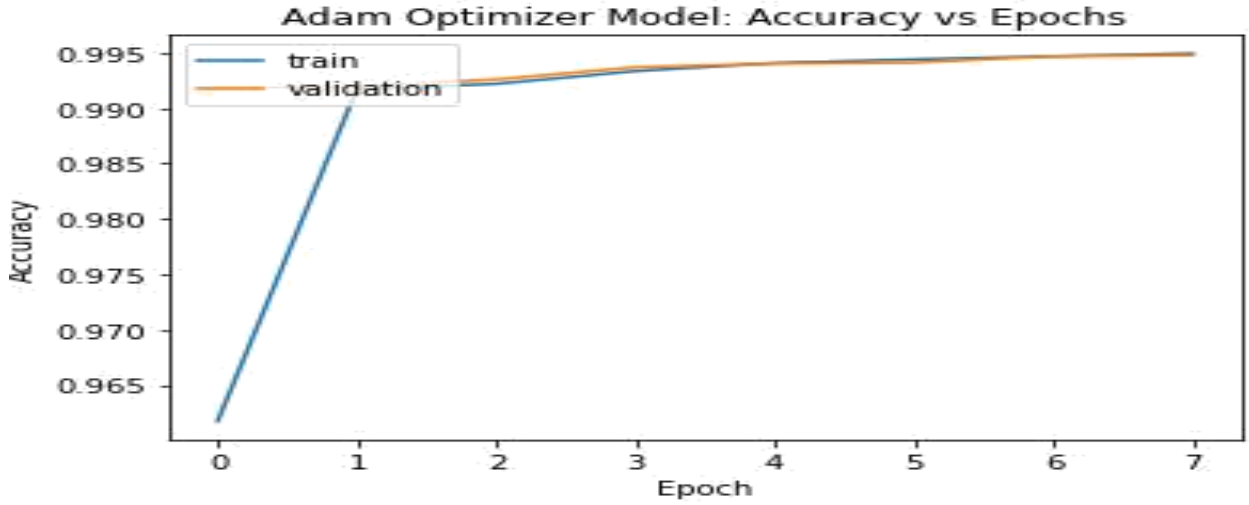
As seen from above table, the current best model with "RMSPROP" model optimizer function still outperforms the models with other optimizers. Hence, the next set of experiments will stick to "RMSPROP" optimizer function in the model architectures.

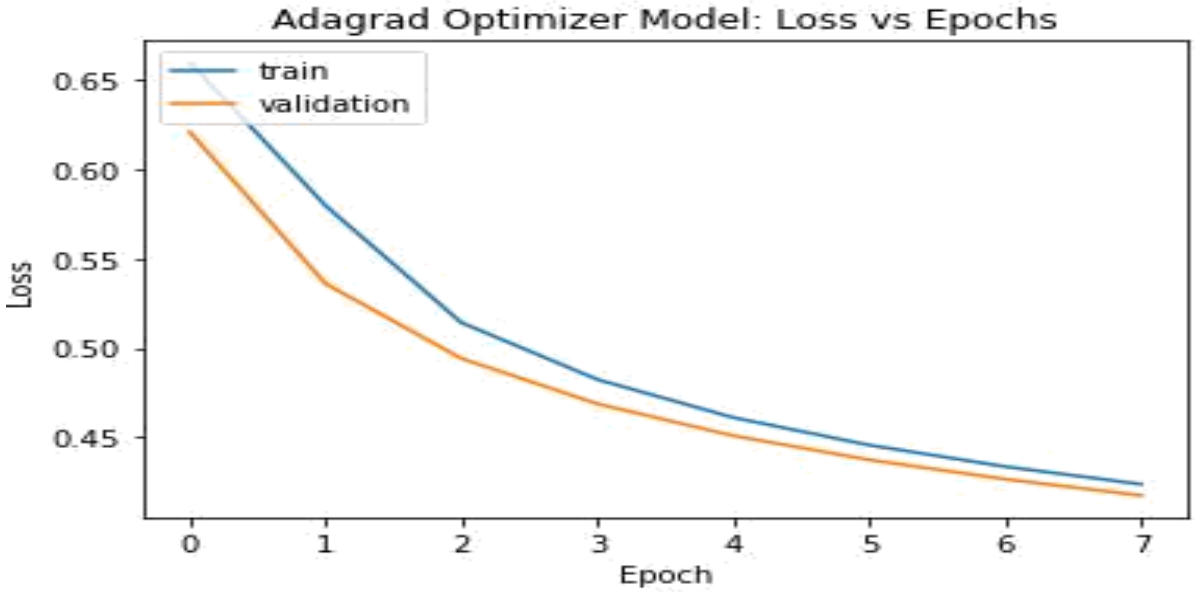
Learning Curves of Iteration-2 Architectures

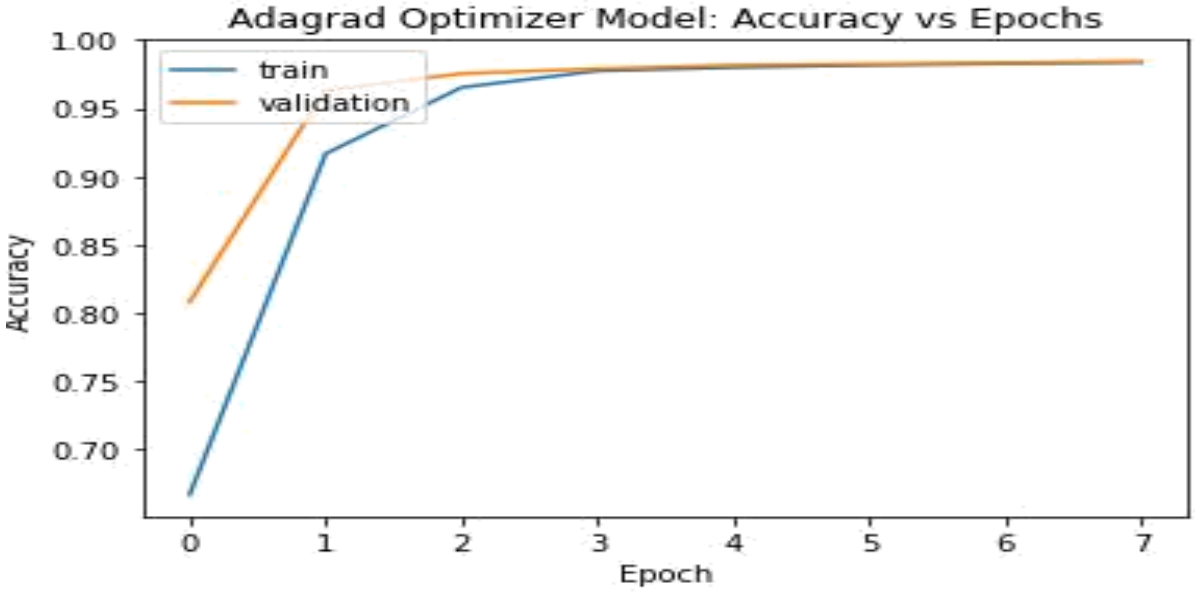



SGD Optimizer Model: Accuracy vs Epochs
train
validation
Accuracy
0.99
0.98
0.97
0.96
0.95
0
1
2
3
4
5
6
7
Epoch


Adam Optimizer Model: Loss vs Epochs
train
validation
Loss
0.10
0.08
0.06
0.04
0.02
0
1
2
3
4
5
6
7
Epoch


Adam Optimizer Model: Accuracy vs Epochs
train
validation
Accuracy
0.995
0.990
0.985
0.980
0.975
0.970
0.965
0
1
2
3
4
5
6
7
Epoch


Adagrad Optimizer Model: Loss vs Epochs
train
validation
Loss
0.65
0.60
0.55
0.50
0.45
0
1
2
3
4
5
6
7
Epoch


Adagrad Optimizer Model: Accuracy vs Epochs
train
validation
Accuracy
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0
1
2
3
4
5
6
7
Epoch

Iteration-3: Modifying Batch Size

The baseline architecture used a batch size of 256 for training. Di erent batch sizes would result in di erent models as the batch size e ect the way gradient is being computed in the optimizer functions. Hence, we tried batch sizes 128, 64 and 32 in addition to understand what batch-size would result in the best validation set performance.

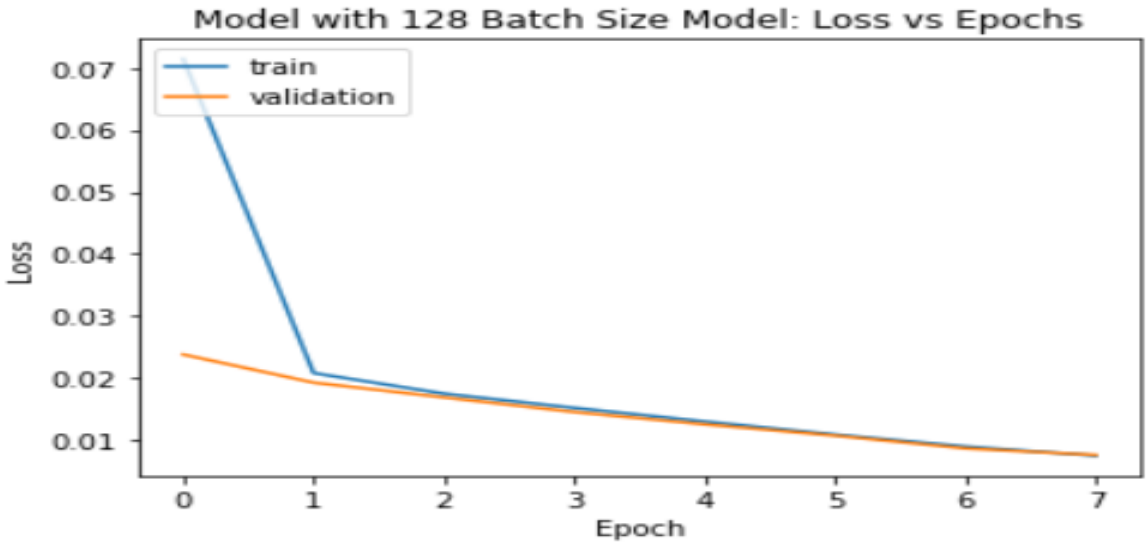
Iteration-3 Model Performances

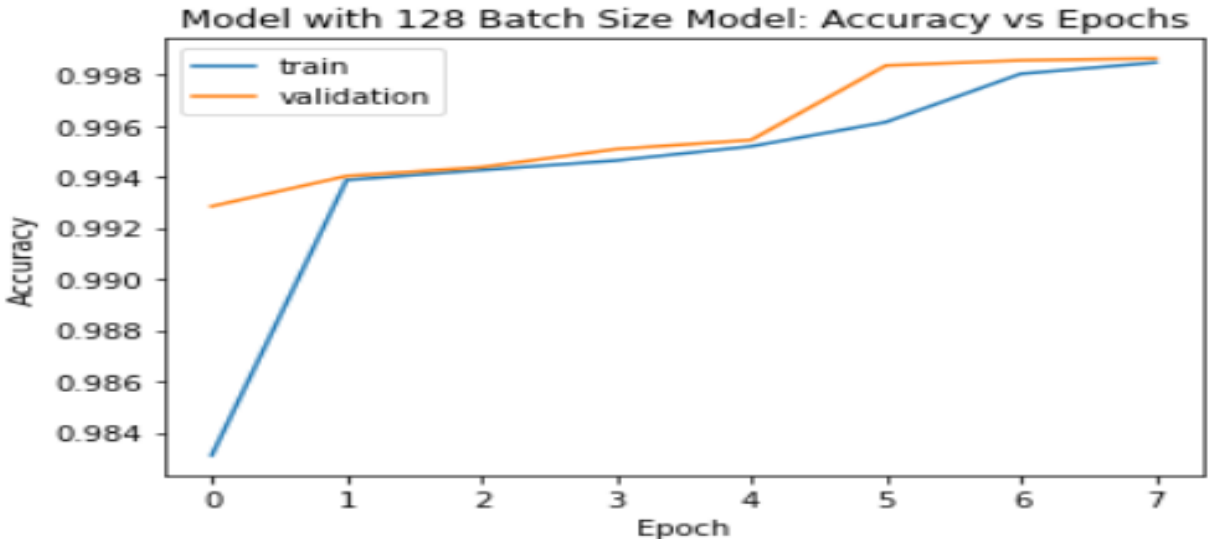
Iteration-3: Model Performances with different batch sizes.

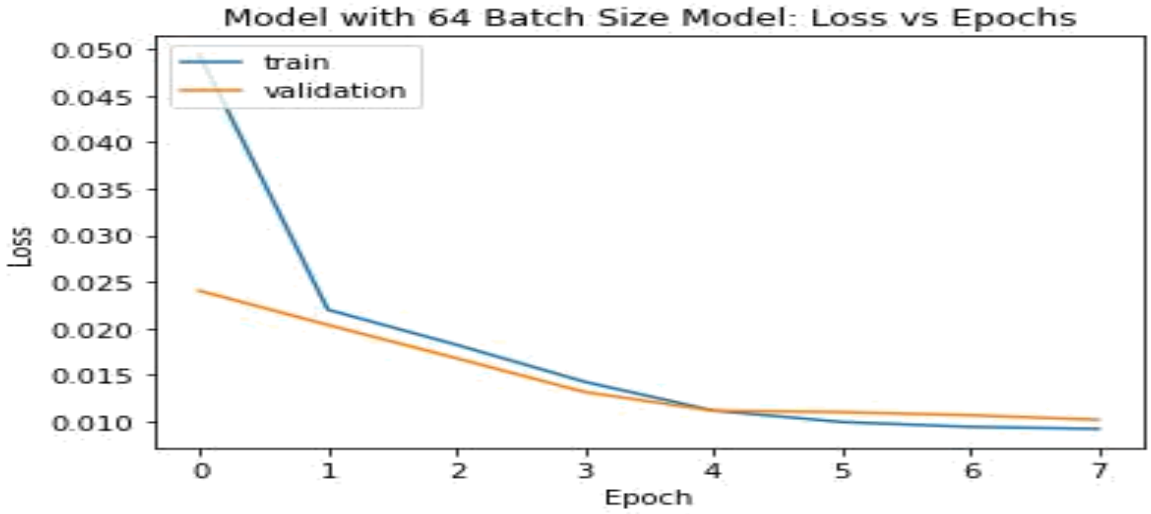
Batch Size	Validation Set Loss	Validation Set Accuracy
256 (current best)	0.0066	99.82
128	0.0083	99.84
64	0.0101	99.75
32	0.0126	99.83

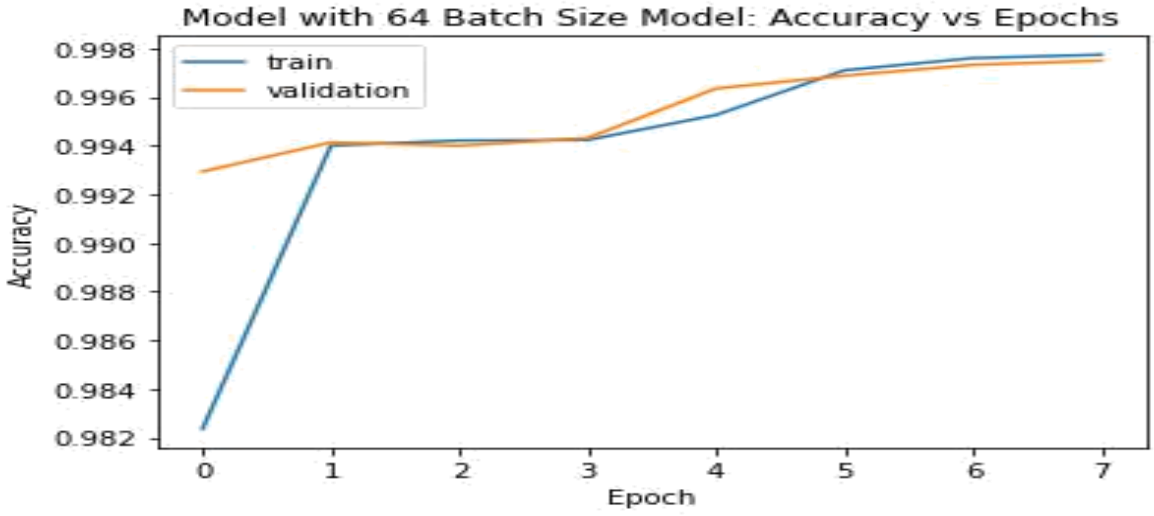
As seen from above table, the model that is trained with batch-size of 128 outperformed the current best (used 256 batch size) and the other batch-sizes. Therefore, the next set of experiments will be using a batch size of 128.

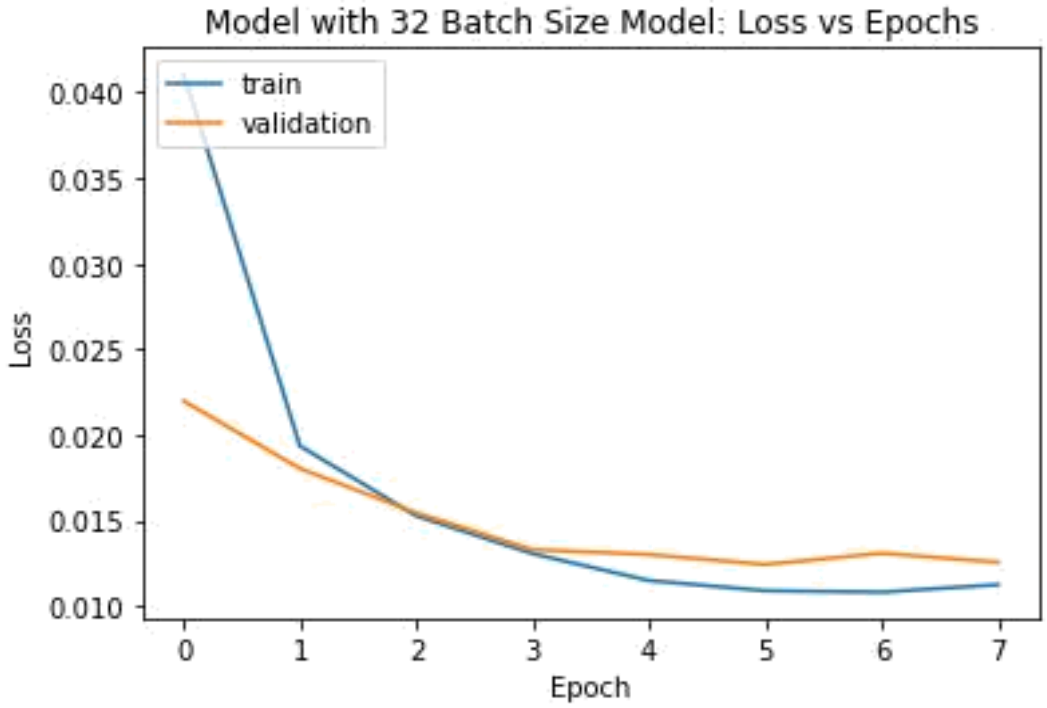
Learning Curves of Iteration-3 Architectures

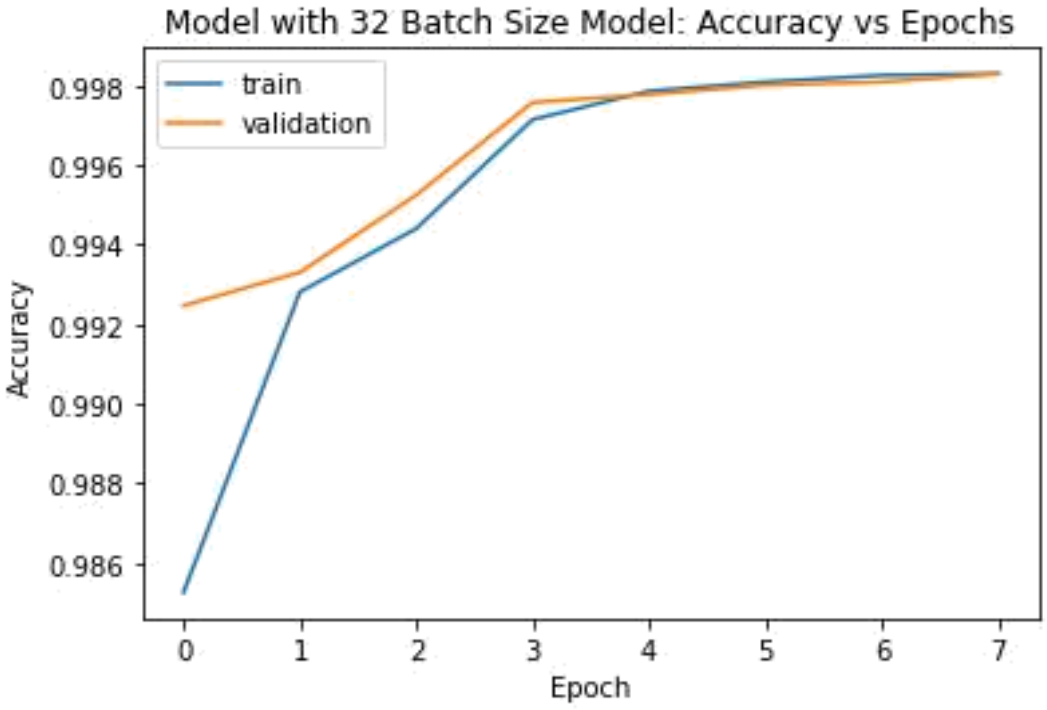



Model with 128 Batch Size Model: Accuracy vs Epochs
train
validation
Accuracy
Epoch


Model with 64 Batch Size Model: Loss vs Epochs
train
validation
Loss
Epoch


Model with 64 Batch Size Model: Accuracy vs Epochs
train
validation
Accuracy
Epoch


Model with 32 Batch Size Model: Loss vs Epochs
train
validation
Loss
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0
1
2
3
4
5
6
7
Epoch


Model with 32 Batch Size Model: Accuracy vs Epochs
train
validation
Accuracy
0.998
0.996
0.994
0.992
0.990
0.988
0.986
0
1
2
3
4
5
6
7
Epoch

Iteration-4: Modifying Number of Neurons

The baseline architecture used 4 neurons for the computations. Hence, hidden layer of 2 neurons and hidden layer with 6 neurons has been tried to nd if change in neurons resulted in model performance improvements as seen below

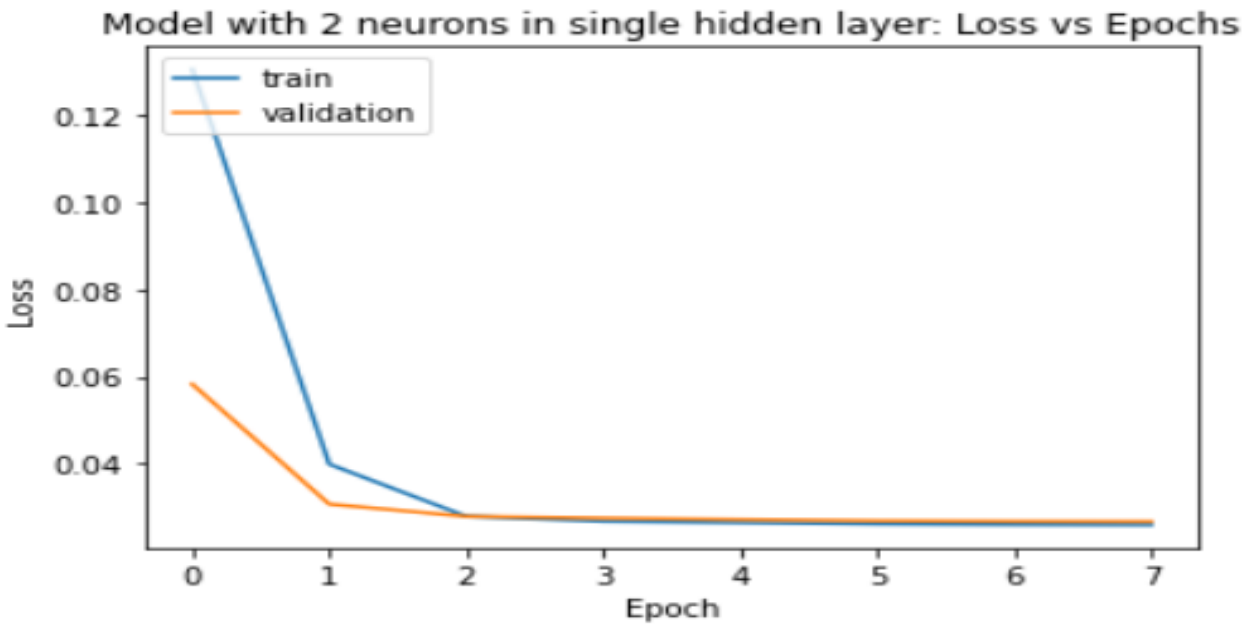
Iteration-4 Model Performances

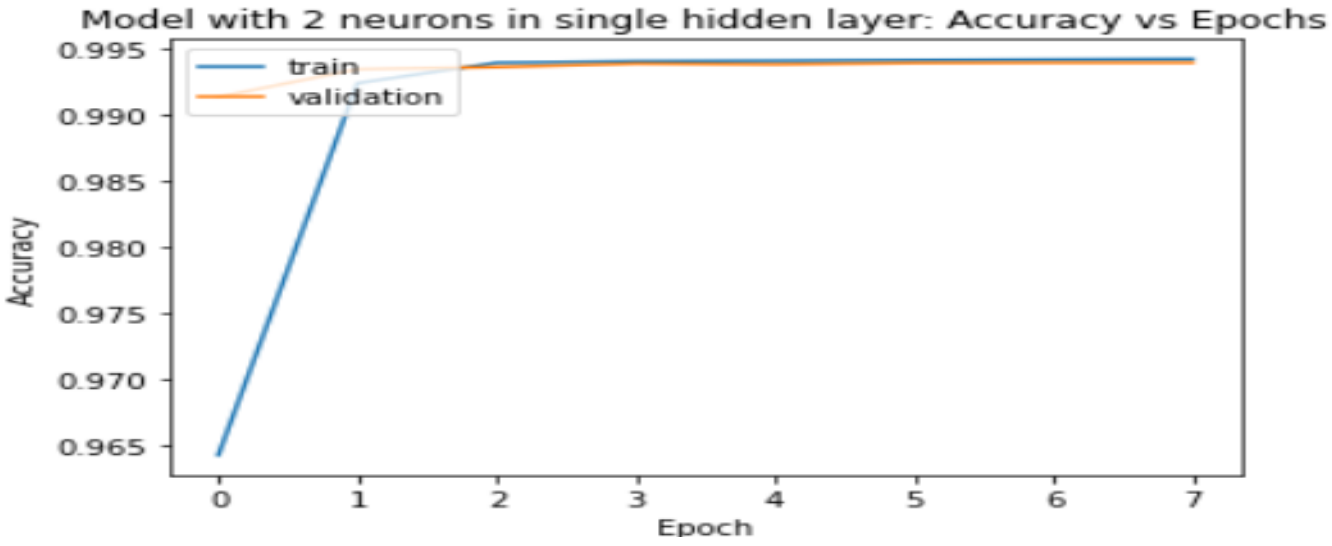
Iteration-4: Model Performances with di erent number of neurons in hidden layer.

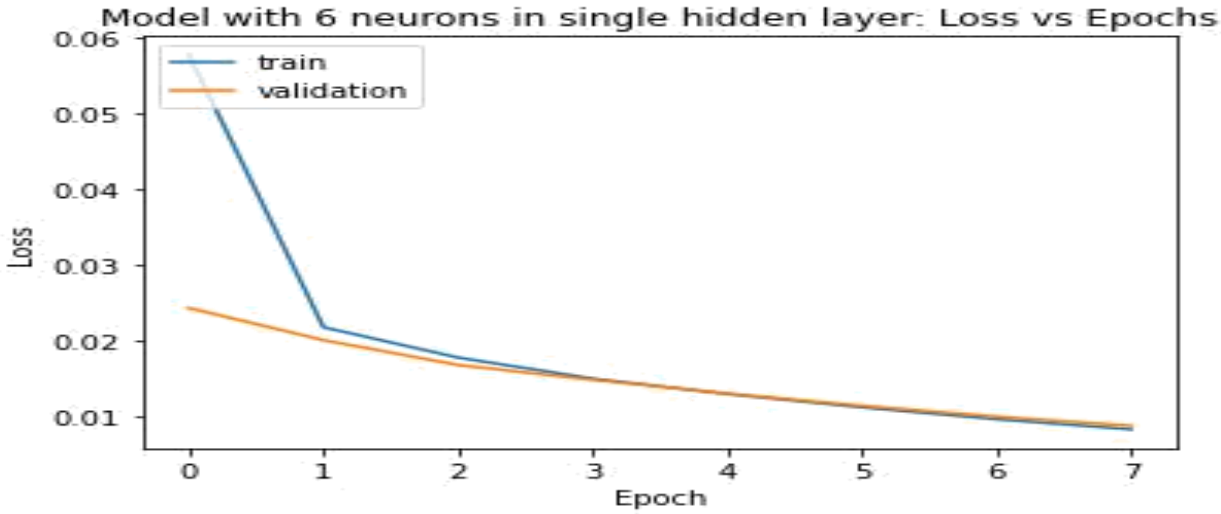
Number of Neurons	Validation Set Loss	Validation Set Accuracy
4 (current best)	0.0083	99.84
2	0.0299	99.31
6	0.0087	99.84

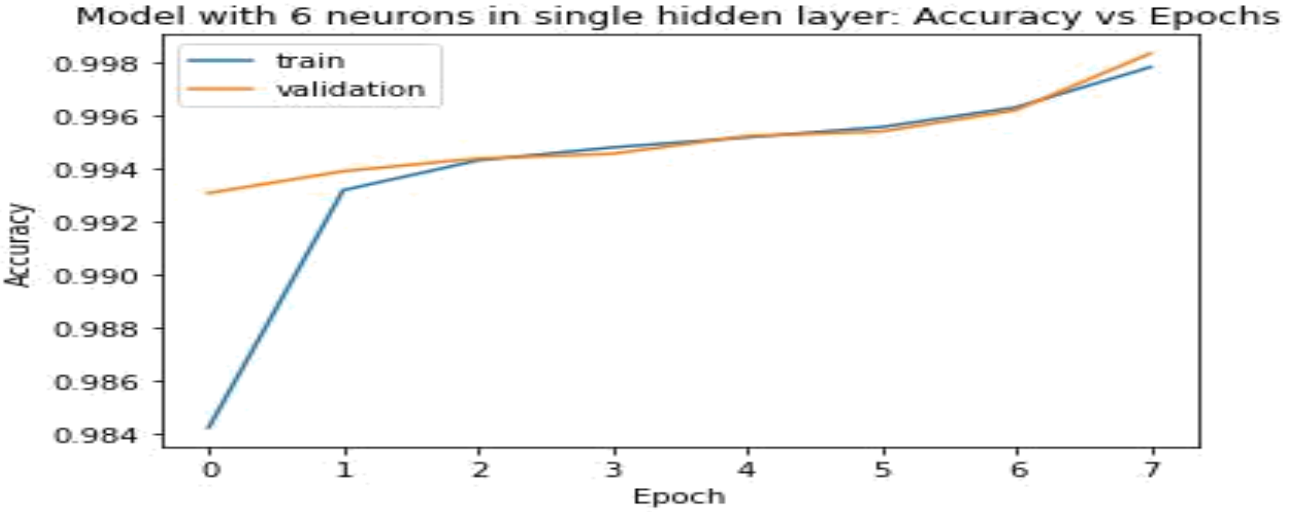
As seen from above table, the model that is trained with 6 neurons performed same as the current best model which has 4 neurons. We will be sticking to 4 neurons for next set of iterations so that the computation cost will be lower with the lesser number of neurons.

Learning Curves of Iteration-4 Architectures




Model with 2 neurons in single hidden layer: Accuracy vs Epochs
train
validation
Accuracy
Epoch
0.995
0.990
0.985
0.980
0.975
0.970
0.965
0
1
2
3
4
5
6
7


Model with 6 neurons in single hidden layer: Loss vs Epochs
train
validation
Loss
Epoch
0.06
0.05
0.04
0.03
0.02
0.01
0
1
2
3
4
5
6
7


Model with 6 neurons in single hidden layer: Accuracy vs Epochs
train
validation
Accuracy
Epoch
0.998
0.996
0.994
0.992
0.990
0.988
0.986
0.984
0
1
2
3
4
5
6
7

Iteration-5: Modifying Number of Epochs

The baseline architecture used 8 epochs for the computations. We will be trying out di erent epoch sizes (16,32 and 64) to nd out the best epoch number for this model architecture

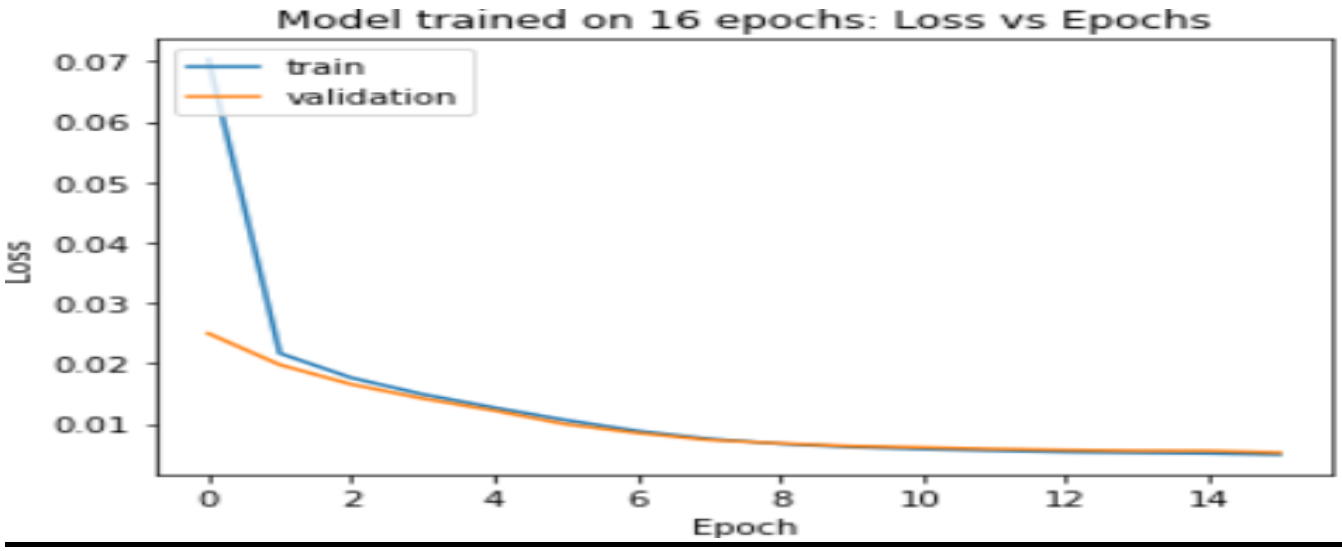
Iteration-5 Model Performances

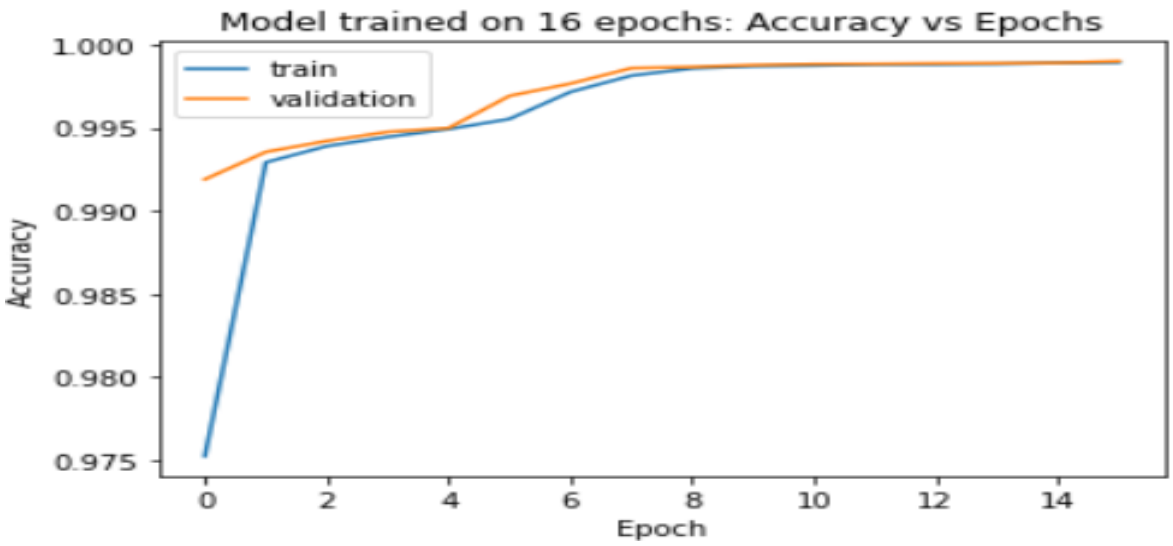
Iteration-5: Model Performances with di erent number of epochs.

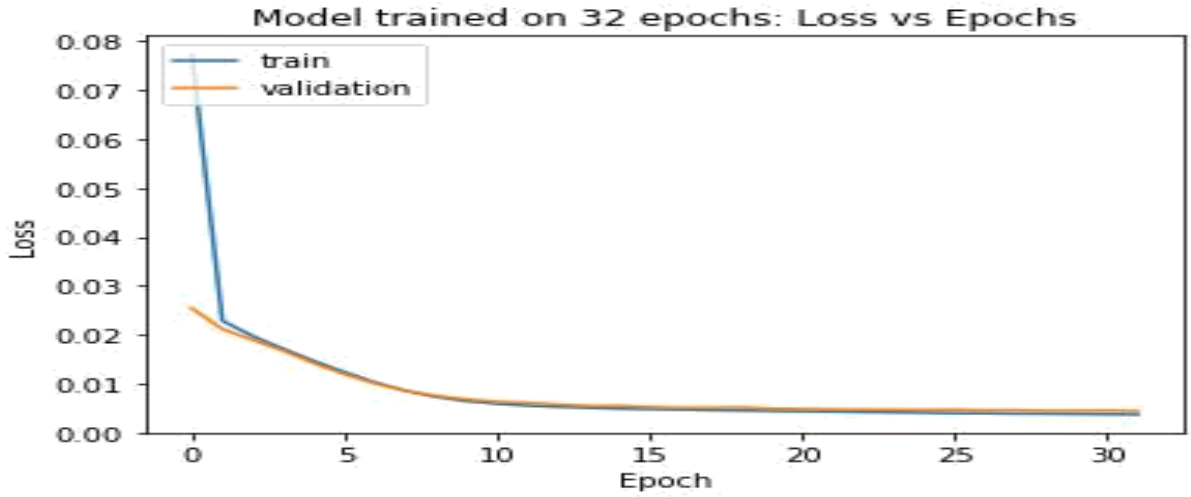
Number of Epochs	Validation Set Loss	Validation Set Accuracy
8 (current best)	0.0083	99.84
16	0.0056	99.88
32	0.0044	99.9
64	0.0047	99.91
64(Early Stopping)	0.0048	99.91

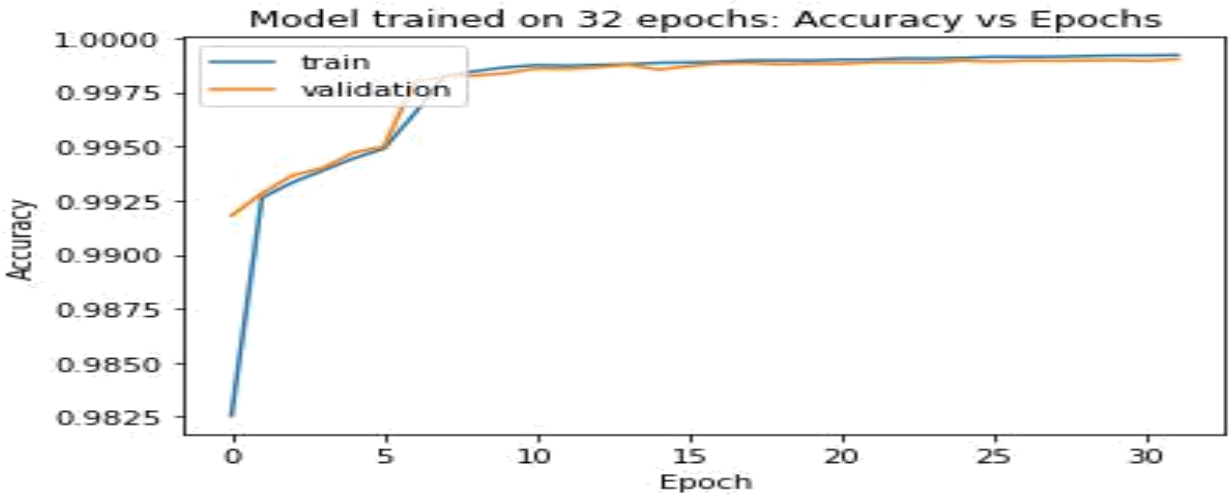
As seen from above table, the model that is trained with 6 neurons performed same as the current best model which has 4 neurons. We will be sticking to 4 neurons for next set of iterations so that the computation cost will be lower with the lesser number of neurons.

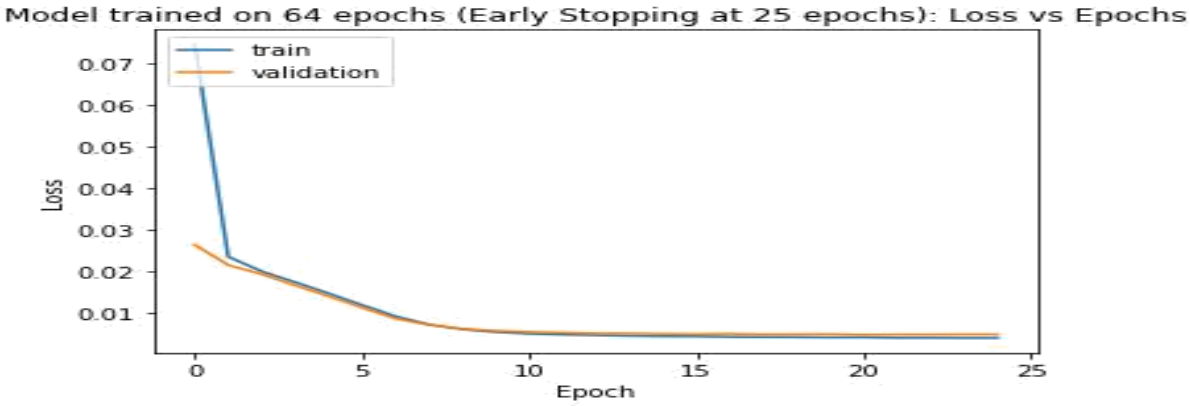
Learning Curves of Iteration-5 Architectures

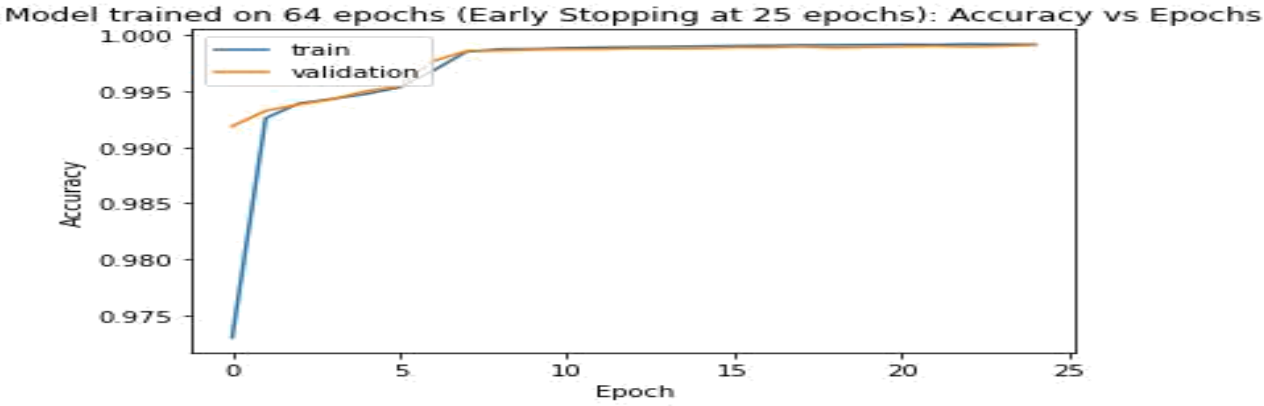


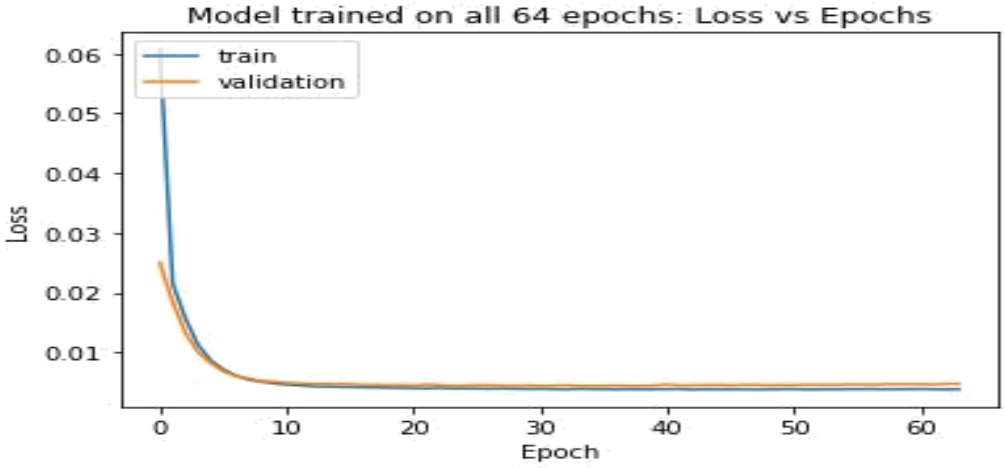

Model trained on 16 epochs: Accuracy vs Epochs
train
validation
Accuracy
Epoch
1.000
0.995
0.990
0.985
0.980
0.975
0
2
4
6
8
10
12
14

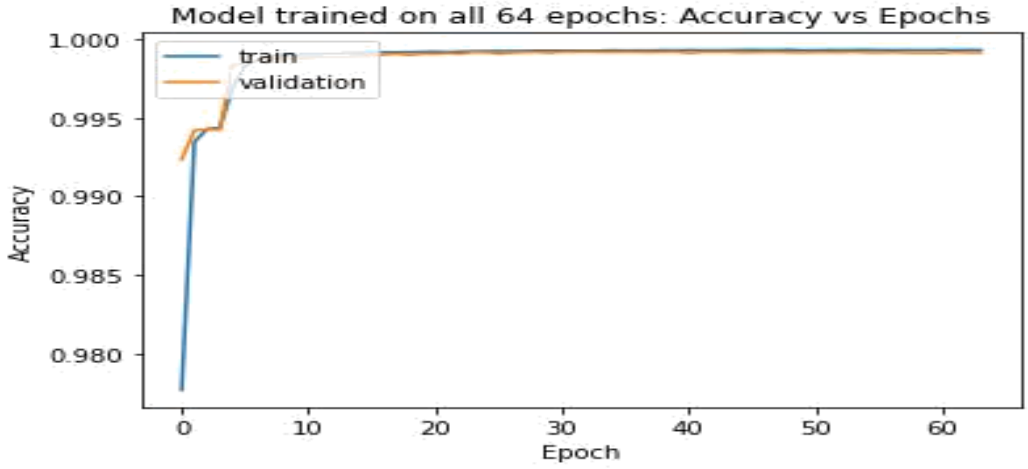

Model trained on 32 epochs: Loss vs Epochs
train
validation
Loss
Epoch
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
0
5
10
15
20
25
30


Model trained on 32 epochs: Accuracy vs Epochs
train
validation
Accuracy
Epoch
1.0000
0.9975
0.9950
0.9925
0.9900
0.9875
0.9850
0.9825
0
5
10
15
20
25
30


Model trained on 64 epochs (Early Stopping at 25 epochs): Loss vs Epochs
train
validation
Loss
Epoch
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
5
10
15
20
25

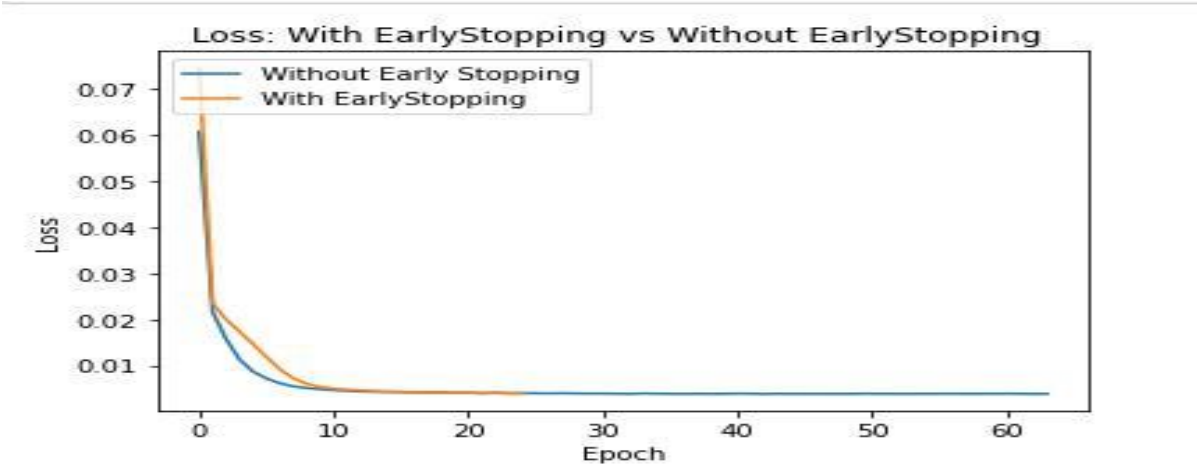

Model trained on 64 epochs (Early Stopping at 25 epochs): Accuracy vs Epochs
train
validation
Accuracy
Epoch
1.000
0.995
0.990
0.985
0.980
0.975
0
5
10
15
20
25


Model trained on all 64 epochs: Loss vs Epochs
train
validation
Loss
Epoch
0.06
0.05
0.04
0.03
0.02
0.01
0
10
20
30
40
50
60


Model trained on all 64 epochs: Accuracy vs Epochs
train
validation
Accuracy
Epoch
1.000
0.995
0.990
0.985
0.980
0
10
20
30
40
50
60

Early Stopping

The concept of early-stopping in model training saves computation time and cost when the model is not learning. As seen in the iteration-5, the model did not learn after epoch 25. The experiment did set the patience to 4 in the keras callback function which means the model loss did not improve continuously for 4 epochs before 25th epoch (including 25).



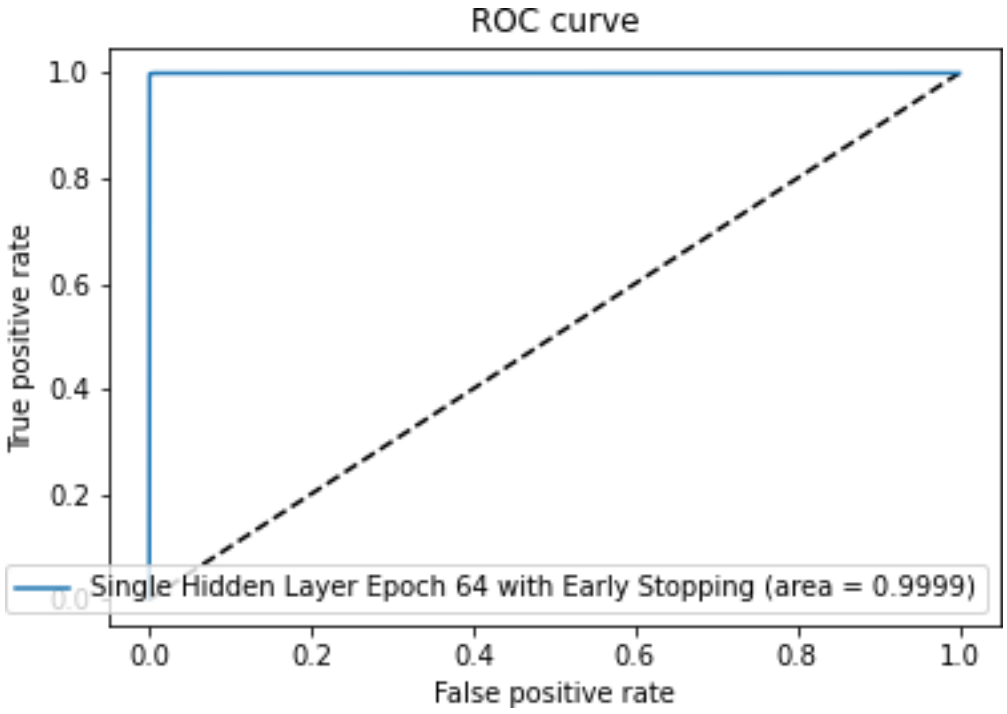
Model Selection

We have obtained various metrics like precision, recall, F1, and Area-Under-Curve (AOC) for each model to select the best model.

Model	N_Params	Accuracy	Precision	Recall	F1-Score	auc_score
No Hidden Layers	120	0.9944	0.9932	0.9787	0.9859	0.9984
Single Hidden Layer	485	0.9961	0.994	0.9863	0.9901	0.9992
Two Hidden Layers	493	0.9943	0.9972	0.9746	0.9858	0.9965
Single Hidden Layer ReLU Activation	485	0.9982	0.9942	0.9966	0.9954	0.9999
Single Hidden Layer Softmax Activation	485	0.8034	0	0	0	0.5321
Single Hidden Layer SGD optimizer	485	0.989	0.9886	0.9569	0.9725	0.9955
Single Hidden Layer Adam optimizer	485	0.9948	0.9959	0.978	0.9869	0.9994
Single Hidden Layer Adagrad optimizer	485	0.9842	0.9804	0.9414	0.9605	0.9873
Single Hidden Layer Batch Size 128	485	0.9984	0.9955	0.9963	0.9959	0.9997
Single Hidden Layer Batch Size 64	485	0.9975	0.991	0.9962	0.9936	0.9998
Single Hidden Layer Batch Size 32	485	0.9983	0.9953	0.996	0.9957	0.9998
Single Hidden Layer 2 Neurons	243	0.9931	0.9993	0.9667	0.9828	0.9959
Single Hidden Layer 6 Neurons	727	0.9984	0.9957	0.9959	0.9958	0.9999
Single Hidden Layer Epoch 16	485	0.9988	0.9972	0.9969	0.9971	0.9999
Single Hidden Layer Epoch 32	485	0.999	0.9976	0.9975	0.9975	0.9999
Single Hidden Layer Epoch 64 with Early Stopping	485	0.9991	0.9984	0.9973	0.9978	0.9999
Single Hidden Layer Epoch 64 without Early Stopping	485	0.9991	0.9978	0.9977	0.9978	0.9999

ROC Curves for all Models

Below is the ROC curve for the best model with epoch size = 25 along with best parameters. ROC curves for all models can be found in this notebook



Model Selection Inference

As seen from the classification metrics and ROC curves, the single layer model with 4 neurons, rmsprop optimizer and ReLU activation function along with batch-size=128, epochs=25 achieved higher accuracy, precision, recall, f1-score and auc values. This proves that this model is more generalized across both the target classes. Hence, the above highlightened model is being selected.

Overfitting with target variable

Another way of understanding the over tting problem is to understand how big the neural network architecture should be when output variable is being used as additional feature in the input feature set. We have performed an experiment with the output feature in the input dataset which can be found in the github link.

As seen in the below gure, it required only a basic architecture with no hidden layer to over t the entire dataset in this case. This is because, the correlation between the output variable feature and output variable itself will be 1 which could be the reason for this easy over tting

Custom predict function

After selecting and saving the best model, we have also implemented a custom predict function manually by using the trained model parameters (weights). We performed the computations manually as per the network architecture. As seen in below notebook cell outputs, the predictions (and metrics) are same as compared to the keras predict function.

Iterative Feature Reduction

Feature Importance

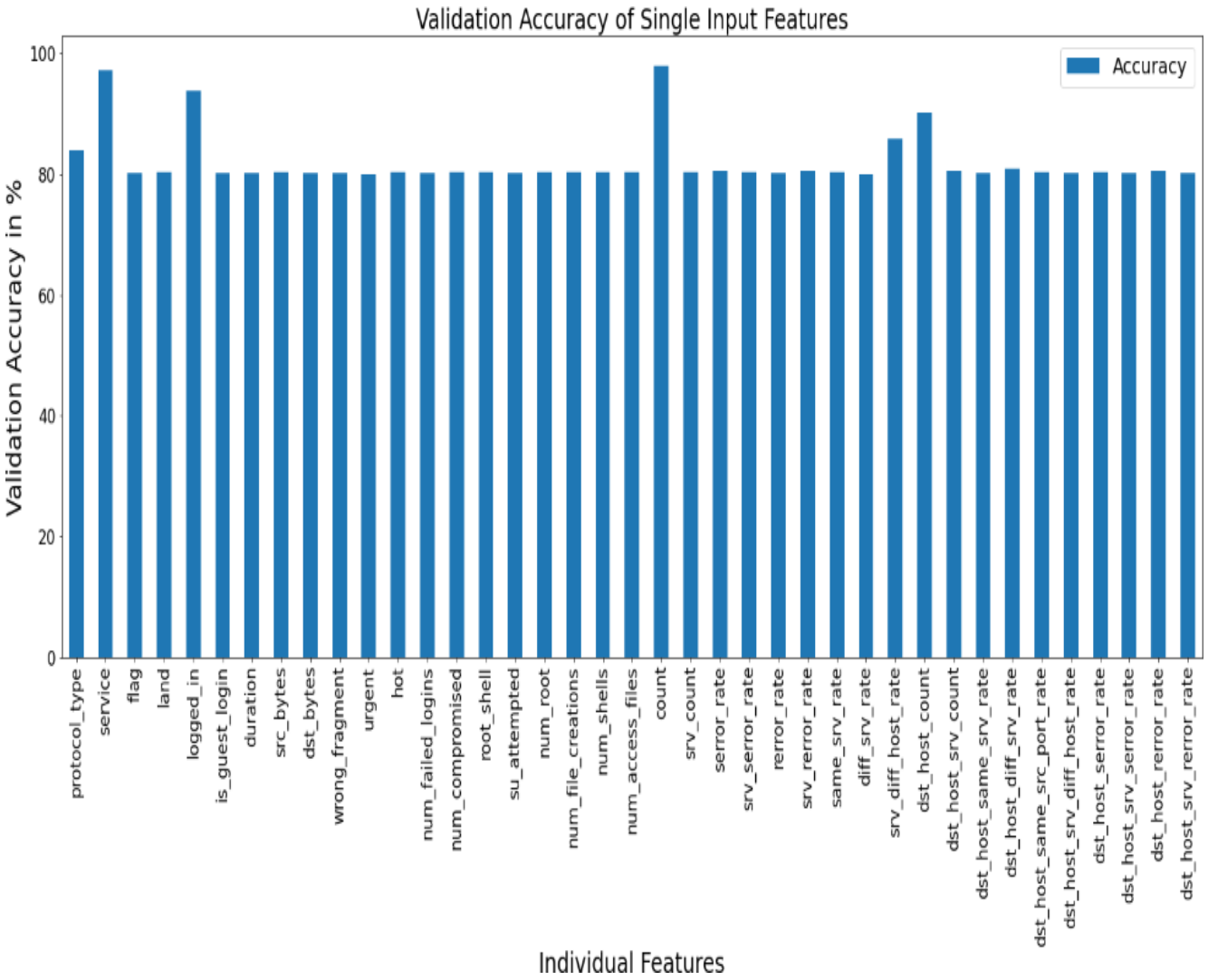
We ran all 39 features through the best model architecture individually and obtained the accuracy of each model on the validation set. We then ranked the model performances based on the validation accuracy. Our assumption is that feature with lowest validation accuracy has less feature importance compared to others.

As seen in the below table and the picture, the feature ”count” has the highest importance and the feature ”diff-srv-rate” has the least importance.

Table 3: Feature Importance Ranking

Rank	Accuracy	feature name			
0	79.959315	diff_srv_rate			
1	80.054450	urgent			
2	80.109107	dst_host_srv_diff_host_rate			
3	80.175906	dst_host_same_srv_rate			
4	80.178940	su_attempted 5	80.182987	duration	
6	80.184001	is_guest_login 7	80.204242	flag	
8	80.215377	dst_host_srv_serror_rate			
9	80.223471	num_failed_logins			
10	80.252826	wrong_fragment 11	80.259907	dst bytes	
12	80.271041	rerror_rate			
13	80.271041	dst_host_srv_rerror_rate			
14	80.283189	dst_host_serror_rate			
15	80.285209	same_srv_rate 16	80.288249	land	
17	80.299383	srv_count			
18	80.302417	srv_serror_rate 19	80.305451	num root	
20	80.330753	root_shell			
21	80.340874	num_shells			
22	80.341887	num_file_creations			
23	80.378324	num_access_files 24	80.385411	src bytes	
25	80.407673	num_compromised			
26	80.437028	dst_host_same_src_port_rate 27		80.444109	hot
28	80.465364	srv_rerror_rate			
29	80.479532	dst_host_srv_count			
30	80.494720	serror_rate			
31	80.529130	dst_host_rerror_rate			
32	80.915755	dst_host_diff_srv_rate			

33	83.928788	protocol_type		
34	85.939842	srv_diff_host_rate		
35	90.297961	dst_host_count 36	93.734062	logged in
37	97.144854	service_		
38	97.977817	counts		



Performance After Removal

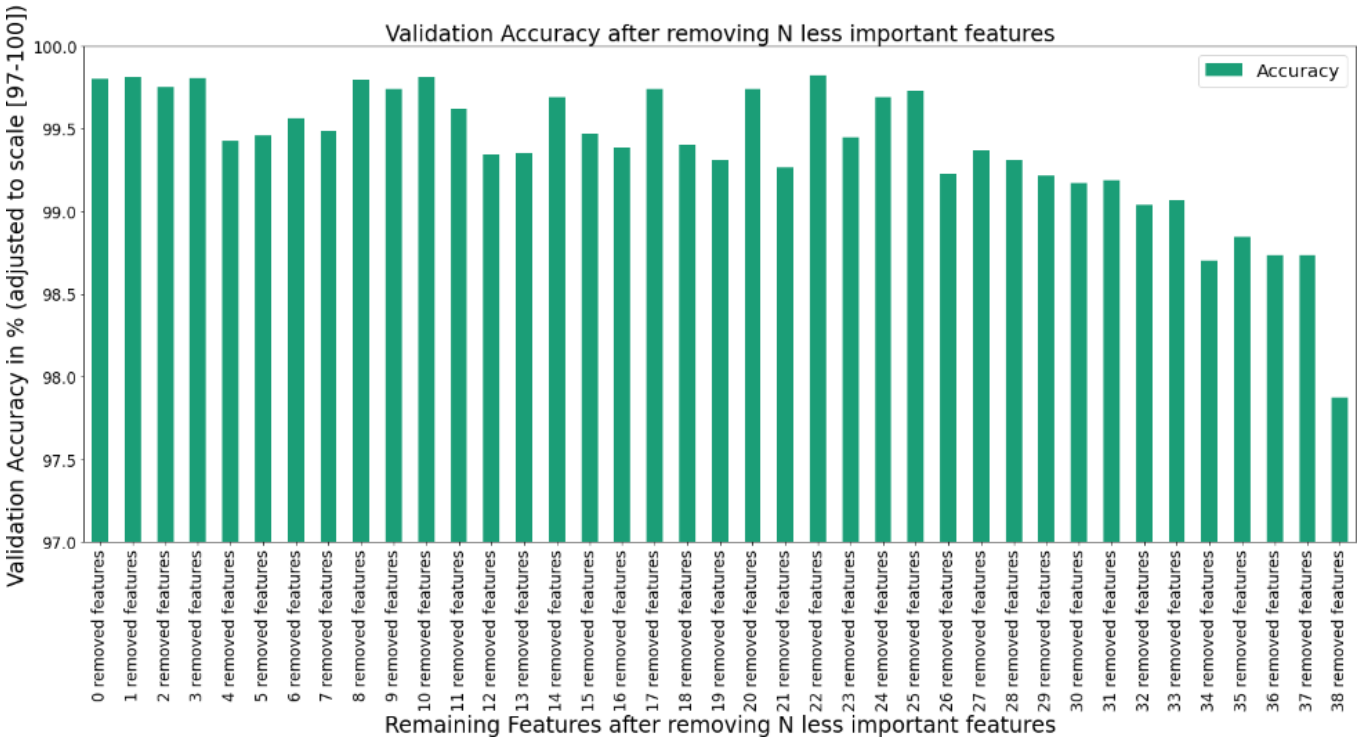
The list of features used for this data set can be categorized into three categories - 1) Basic features of individual TCP connections.2) Content features within a connection suggested by domain knowledge. 3) Traffic features computed using a two-second time window.

We observed if there is a correlation between above three categories and the feature importance values of the individual features. We did not find any strong correlation between this domain specific categorization. However, it is evident that the mostly traffic related features ended up having high feature importance values. Therefore, we strongly believe that the actual values in each feature contributed to the model performance which made sense.

Based on the above feature importance values, we started building the models by removing the unimportant features (importance based on the validation accuracy, the higher the better) an iterative way. As seen in the below plot, all models' validation accuracy values are obtained and are plotted below.

The main motive of this experiment is to find out the ideal subset of features that would yield best results with less computation power consumption and modeling time. From the below plot, we concluded that the model built after removing 22 less important features showed higher validation accuracy. We thus select the features used for this model as the best features for this data set. The features used for this model are:

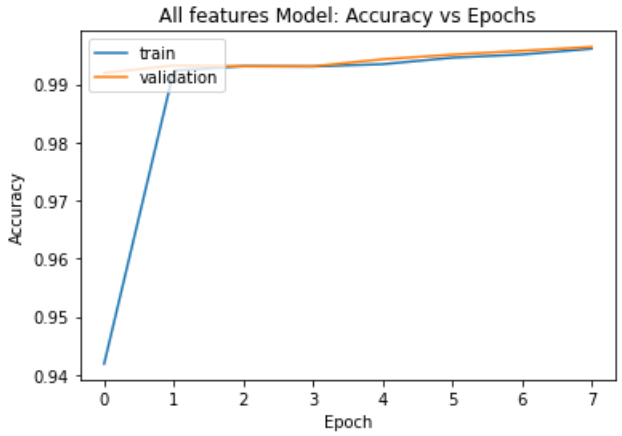
['num access files', 'src bytes', 'num compromised','dst host same src port rate', 'hot', 'srv rerror rate', 'dst host srv count', 'serror rate', 'dst host error rate', 'dst-host-diff-srv-rate', 'protocol-type', 'srv-diff-host-rate', 'dst-host-count', 'logged-in', 'service', 'count']

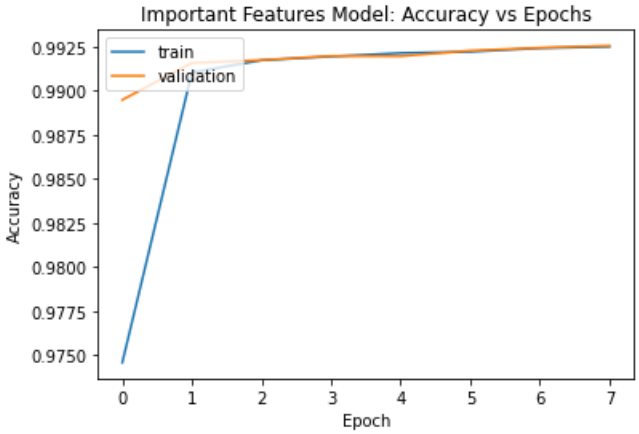

Validation Accuracy after removing N less important features
Validation Accuracy in % (adjusted to scale [97-100])
100.0
99.5
99.0
98.5
98.0
97.5
97.0
Accuracy
0 removed features
1 removed features
2 removed features
3 removed features
4 removed features
5 removed features
6 removed features
7 removed features
8 removed features
9 removed features
10 removed features
11 removed features
12 removed features
13 removed features
14 removed features
15 removed features
16 removed features
17 removed features
18 removed features
19 removed features
20 removed features
21 removed features
22 removed features
23 removed features
24 removed features
25 removed features
26 removed features
27 removed features
28 removed features
29 removed features
30 removed features
31 removed features
32 removed features
33 removed features
34 removed features
35 removed features
36 removed features
37 removed features
38 removed features
Remaining Features after removing N less important features

Performance Comparison: All-Features-Model vs Important-Features-Model

We performed comparison analysis on the model that was built with all features and on the model that was built with important features. As seen in the below plots, the learning rate of the model that was built with all features is not increasing and saturated right after first epoch. Whereas, the learning rate of the model built with subset of importance features (taken from previous step), increased per each epoch (loss reduced for every epoch). Therefore, it is evident that the model built with subset of important features showed more generalization which will be helpful when the data set sizes are varied.

Moreover, building the model with less number of features requires less computations thus less computing power. The model built with the above subset of important features is evidently better in terms of modeling times, model performance and computation cost.





Conclusion

After using iterative feature reduction technique, features that are derived as the important ones for this data set are 'dst_bytes', 'same_srv_rate', 'logged_in', 'is_guest_login', 'dst_host_diff_srv_rate', ‘root_shell', 'num_file_creations', 'src_bytes', 'duration', 'dst_host_srv_count', 'dst_host_srv_diff_host_rate', 'protocol_type', 'srv_diff_host_rate', 'dst_host_count', 'service','count'

References

[1] Manoj Kumar Putchala. Deep learning approach for intrusion detection system (ids) inthe internet of things (iot) network using gated recurrent neural networks (gru). 2017.

[2] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani. A detailed analysis of the kdd cup 99 data set. In *2009 IEEE symposium on computational intelligence for security and defense applications*, pages 1–6. IEEE, 2009.

[3] Steven Huang. Kdd cup 1999 data, computer network intrusion detection (version 1). 2018.
Available from https://www.kaggle.com/galaxyh/kdd-cup-1999-data.