CSEC-720 Deep Learning Security (2225 Spring)

Experimental Assignment 1: Website Fingerprinting using Deep Learning

Zhi Liu[1], Alex Sutay[1], and Mehul Sen[1]

[1]Golisano College of Computing and Information Sciences, Rochester Institute of Technology

February 2023

1 Introduction

Website fingerprinting is a technique that aims to categorize network traffic data into specific websites based on their unique features, such as packet timing, size, and number. These features can be combined to form a website's unique "fingerprint." Attackers can use this technique to infer which webpages a user is visiting, even if the user is using encryption.

In this assignment, we aim to train and evaluate the effectiveness of three deep learning models, namely Convolutional Neural Network (CNN), Long Short-term Memory (LSTM), and Stacked Denoising Auto-Encoding (SDAE), for website fingerprinting attacks. We use hyperparameters that were previously used in related works, such as AWF[7], DF[9] and DL[1] identify the best model architectures and the impact it had on the performance.

The CNN best model achieved an improvement of approximately 2.5% in accuracy when tested on the full dataset. The best LSTM model was not able to get good results. The new SDAE model either worsened or performed the same as the default model.

2 Experimental Design

Dataset	# of Sites	# of Samples per Site	Total Samples
small_10_100	10	100	1000
large_10_1000	10	1000	10000
large_95_100	95	100	9500
full	95	1113 (approx.)	105730

Table 1: Number of Sites and Samples per Site in each dataset

Table 1 presents the results of our model testing on four distinct datasets: the small 10 100, large 10 1000, large 95 100, and full datasets. Each of these datasets is composed of multiple files, each identified by the format X-Y. Here, X represents the associated traffic site and Y represents the corresponding sample number. Each file contains network traffic data, which is parsed into a list of values consisting of either '1' or '-1'. We use '0' to pad the lists, and the final value of each list indicates the site from which the data originated.

2.1 CNN

The hyperparameters used in this assignment were determined through a process of analysis, testing, and selection. This process drew on research conducted directly on the hyperparameters as well as

insights from S. Dubey, S. Singh and B. Chaudhuri[4], D. Marcu and C. Grava[6], S. Vani and T. Rao[11], S. Bera and V. Shrivastava [2], C. Garbin, X. Zhu, and O. Marques[5] and Srivastava et al.[10]. Additionally, prior WF research done by Rimmer et al.[7] and Sirinam et al.[9] was also considered.

Hyperparameters	Search Range
Input Units	[1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000]
Optimization Function	[SGD, Adam, AdaMax, RMSProp]
Activation Function	[TanH, ReLU, SeLU, ELU, Sigmoid]
Batch Normalization	[No, Yes]
Dropout Layers	[None, 0.10, 0.30, 0.50, 0.70, 0.90, 0.93, 0.95, 0.97, 0.99]

Table 2: Table of the search range of Hyperparameters

Table 2 shows a list of the hyperparameters that were tested as part of this assignment as well as the search range tested. A breakdown of the evaluated hyperparameters and why they were selected for this assignment is as follows:

- *Input Units:* These refer to the total number of packet directions that serve as the input for the model. Changing the input units changes how much of the input, the model is allowed to use for its training and testing. To identify the ideal number of input units, the search range consists of values 1000 till 10,000. Rimmer et al. [7] used 3000 input units for their WF classification model, while Sirinam et al. [9] used 5000 input units for their WF classification model.

- *Optimization Functions:* They play an important role in the learning process of a CNN model. They direct the optimization process and determine a model's performance. Changing optimization functions can alter a model's training process and performance. The range of values considered for optimization functions includes Stochastic Gradient Descent (SGD), Adam, AdaMax, and RMSProp. Research by S. Vani and T. Rao [11] showed that AdaMax gave the best performance, while work by S. Bera and V. Shrivastava [2] found that Adam outperformed others. Additional work by D. Marcu and C. Grava [6] found that SGD had the highest performance. Rimmer et al. [7] used RMSProp for their WF classification model, while Sirinam et al. [9] used AdaMax for their WF classification model.

- *Activation Functions:* They are mathematical functions applied to the output generated by nodes in a model. They add non-linearity to neural networks. Changing activation functions can significantly impact a model's ability to converge and its convergence speed. The range of values considered for activation functions includes Hyperbolic Tangent (TanH), Rectified Linear Unit (ReLU), Scaled ELU (SELU), Exponential Linear Unit (ELU), and Sigmoid. According to research by S. Dubey, S. Singh, and B. Chaudhuri [4], ReLU performs better in image classification compared to other activation functions mentioned, while Sigmoid and TanH have the worst performance. D. Marcu and C. Grava [6] found that ELU performs best for their image classification model. Rimmer et al. [7] used ReLU for their WF classification model, while Sirinam et al. [9] used a combination of ReLU and ELU for their WF classification model.

- *Batch Normalization(BN):* These are used to standardize the inputs to a network. Adding Batch normalization can improve the learning speed, provide regularization and reduce overfitting. Research done by C. Garbin, X.Zhu and O.Marques [5] found that batch normalization improves accuracy in CNNs with only a small penalty for training time. Rimmer et al. [7] do not use batch normalization for their WF classification model while Sirinam et al. [9] used several batch normalizations for their WF classification model.

- *Dropout Layers:* These are used to address the issue of overfitting by temporarily deactivating a specified percentage of neurons during each iteration of the training process. Changing the dropout layer rate can prevent overfitting and increase a model's performance. The search range for dropout layer rates included values of None, 0.1, 0.5, 0.7, 0.9 and additional values of 0.93, 0.95, 0.97 and 0.99. Research by Srivastava et al. [10] showed that dropout improves the performance of neural networks on supervised learning tasks while research by C. Garbin, X.Zhu and O.Marques [5] found that it reduced their model's accuracy in tests. Rimmer et al. [7] used

a rate of 0.1 for their WF classification model while Sirinam et al. [9] used a combination of rates: 0.1, 0.5 and 0.7 for their WF classification model.

Hyperparameters	Default Model	AWF Model	DF Model
Input Units	2000	3000	5000
Optimization Function	SGD	RMSProp	AdaMax
Activation Function	TanH	ReLU	[ELU, ReLU]
Batch Normalization	No	No	Yes
Dropout Layers	0.50	0.10	[0.10, 0.50, 0.70]

Table 3: Table of Hyperparameters used for CNN Models

Table 3 shows a comparison of three models: the Default model, the AWF model, and the DF model. The Default model is used for testing and training hyperparameters and its complete breakdown is shown in Figure 5. The AWF model was implemented based on research by Rimmer et al. [7], while the DF model was implemented based on research by Sirinam et al. [9]. Each hyperparameter was updated on the default model to create new models and identify the best hyperparameter values. To ensure consistency and accuracy in our results, each model was executed three times on the 'full' dataset. The performance of each model was evaluated based on its accuracy percentage and loss, then compared with other hyperparameter values. Based on this analysis, we selected the best hyperparameters to construct the 'Best' model. For a detailed overview of all hyperparameters used in these models, please refer to Table 11 in the appendix section.

2.2 LSTM

The LSTM models were trained on the large_95_100 dataset. However, in order for the LSTM to be successful, the data was reshaped to be shorter. To do so without any loss, incoming and outgoing bursts were consolidated into single values. For example, the data was normally structure as incoming packets being one and outgoing being negative one, a stream may be [1, 1, -1, -1, -1, 1]. This can be consolidated to [2, -3, 1] with no data loss. After consolidating in this way, the data was reshaped to all be the same length by padding the beginning of shorter sequences with zeros. This changed the length of the data from 2,000 to 815.

The hyperparameters tested for the LSTM were the optimizer, learning rate, dropout, and shape. The shape will be expressed as a list of units in each layer, so for example [128, 64] would represent 2 layers with the first having 128 units and the second having 64 units.

Hyperparameters	Search Range	Best
Optimizer	[SGD, Adam, RMSprop]	RMSprop
Learning Rate	[0.01, 0.001, 0.0001]	0.01
Dropout Rates	[0, 0.1, 0.2, 0.3]	0.1
Number of LSTM layers	[1, 2]	2
Number of units	[64, 128]	[64, 128]

Table 4: Table of Hyperparameters used for LSTM Models

The optimizer and learning rates were tested first as they would control how the rest of training would perform. Next was the dropout and finally the shape. All of the choices for which hyperparamters to tune and which values to test are based on prior work done by Rimmer et. al [7]. The hyperparameters tested and the best results are shown in Table 4.

The choices for hyperparameters to tune for the LSTM are as follows:

- *Optimizer:* As discussed in the CNN section, the optimization function determines how the model will actually improve. Changing it determines how the training process works, which impacts performance as well as how quickly and effectively it converges on a solution. For the LSTM, we tested SGD, Adam, and RMSprop.

- *Learning Rates:* The learning rate controls how fast the optimization functions work. It's important to strike a balance in the learning rate because if it's too small, the optimizer will never reach the optimal solution, but if it's too large the optimizer will overstep the optimal solution and it may never converge. Yu et al. [13] trained a network to use an adaptive learning rate and found that for most of the training the optimal value was less than 0.1, so we tested 0.01, 0.001, and 0.0001.

- *Dropout Rates:* Dropout is used to control overfitting. It can be built into LSTM layers and works the same way as discussed earlier for CNNs. We chose to test values of 0.1, 0.2, and 0.3 in addition to 0, which is the equivalent of not using dropout.

- *LSTM shape:* This refers to the number of LSTM layers and how many units are in each layer. This is the core of the LSTM functions and essentially determines the complexity of the model. Chakraborty et al. [3] tested different numbers of units in LSTMs with 2 LSTM layers. They found that their best results were with 64-192 units. In order to keep our LSTMs simpler for a faster training time, we tried some models with only 1 LSTM layer. We tried LSTMs with shapes [64], [128], [64, 64], and [128, 64].

2.3 SDAE

We also evaluated the performance of the SDAE under different hyperparameter settings. We are using the large dataset for the SDAE experiment. The dataset contains 10000 samples, which we split into 8000 training samples, 1000 validation samples, and 1,000 test samples. Each sample is classified into one of 10 different classes, with a typical number of 1000 samples per class in the dataset.

Initially, we established a baseline model. After setting up the baseline model, we set one hyperparameter as the independent variable while keeping the other hyperparameters as same as the baseline model to investigate the change in both model accuracy and model loss for the validation set. The hyperparameter configuration of the baseline model and search range are shown in Table 5.

Hyperparameters	Baseline	Search Range
Layer Configuration	[2000,1000,500]	[1000,500,250], [2000,1000], [2000,1000,500,250]
Activation Function	ReLU	[Sigmoid, ELU, Softmax]
Optimization Function	Adamax	[SGD, RMSProp, Adgrad]
Noise Level	0.3	[0, 0.2, 0.6]

Table 5: Table of Hyperparameters used for SDAE Models

- *Layer configuration:* The original SDAE paper [12] shows how accuracy shifts with different combinations of hidden layer numbers and hidden unit numbers. In general, the classification error decreases as the number of hidden layers and hidden units increases. However, this is not always the case, as evidenced by the fact that 500 hidden units with 2 hidden layers slightly outperform 500 hidden units with 3 hidden layers. We aim to investigate how different SDAE structure combinations can affect the accuracy of our model. The SDAE layer configurations tested were: [1000,500,250], [2000,1000], [2000,1000,500,250] and the baseline is [2000,1000,500].

- *Activation function:* As we discussed in the CNN section, we would like to investigate how the performance of our SDAE model changes with different activation functions. We are particularly interested in exploring how a completely different activation function, such as Softmax, might impact our results. Our baseline function is ReLU, but we will also test ELU and Sigmoid.

- *Optimization function:* The baseline optimization function is Adamax, and we will test two additional optimization functions: SGD and RMSProp. In addition to these two optimization functions mentioned in the CNN section, we would also like to test Adagrad, which is an improved version of SGD with an adaptive learning rate. [8] Interestingly, the original SDAE paper used SGD as its optimizer [12].

- *Noise level:* As demonstrated in [12], introducing a certain level of noise can help the model capture more significant features and improve the results. However, adding either too little or

too much noise can reduce the accuracy of the model. Also, the noise plays an important role in reducing overlearning and overfitting[1]. As our model's baseline noise level is 0.3, we aim to explore the effects of decreasing or increasing the noise levels and observe if it can enhance the performance of our SDAE model. We will test noise levels of 0, 0.2, and 0.6 in addition to the baseline value.

Notice when we change the Activation function and Optimization function, we are changing if for both the pre-training layers and fine-tuning layers. For optimization function, we also keep the learning rate same for better comparison.

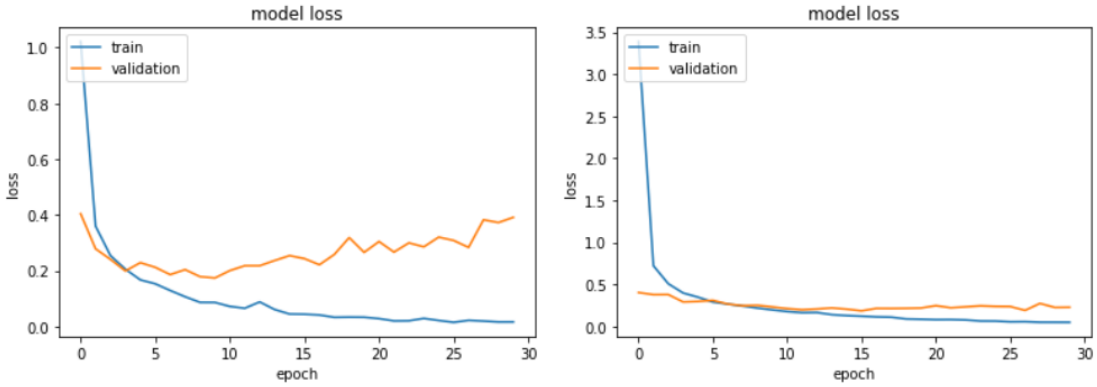

Figure 1: Baseline Model Loss Before/After Batch Normalization

Apart from adjusting the hyperparameters, we also add batch normalization to the pretraining layers. This is due to the fact that the model in the original code will have an overfitting issue as the number of epochs increases. The effect of batch normalization is shown in figure 1.

3 Results

3.1 CNN

		Accuracy (%)
Input Units	1000	65.42
	2000	75.62
	3000	77.21
	4000	77.75
	5000	77.73
	6000	77.17
	7000	77.26
	8000	**79.20**
	9000	78.87
	10000	77.40
Optimization Function	**SGD**	**76.28**
	Adam	68.39
	AdaMax	73.31
	RMSProp	59.22
Activation Function	TanH	76.37
	ReLU	**76.70**
	SeLU	74.13
	ELU	76.00
	Sigmoid	69.07
Batch Normalization	No	75.90
	Yes	**77.42**
Dropout Layers	None	72.79
	0.10	74.69
	0.30	75.36
	0.50	76.44
	0.70	77.02
	0.90	**78.76**
	0.93	77.84
	0.95	75.20
	0.97	67.55
	0.99	37.78

Table 6: CNN Hyperparameter Results

Table 6 shows the performance of various models using a CNN to conduct a web fingerprinting attack on undefended traffic. The following is a breakdown of the results for each of the hyperparameters.

Input Units: Table 6 shows that the model with 8000 input units outperformed all other input units. Input units ranging from 3000 to 9000 have roughly the same accuracy, with 8000 slightly outperforming its counterparts. This indicates a trade-off between having a relatively smaller part of the input that could be more important or consistent between samples and having the entire input which could include several half-empty and padded samples. This suggests that after a certain point, increasing the number of input units no longer improves the model's accuracy.

Optimization Function: Table 6 demonstrates that SGD outperforms the other optimization functions, with AdaMax coming in a close second. On the other hand, the results indicate that both Adam and RMSProp perform significantly worse in these experiments. This suggests that SGD is reliable and efficient allowing the model to converge on a better solution faster than when using other optimization functions. Overall, these results support the claim made by D. Marcu and C. Grava [6], that the SGD function is a superior optimization method.

Activation Functions: Table 6 demonstrates that the ReLU activation function outperforms other activation functions in these experiments. On the other hand, the Sigmoid activation function performs the worst. This suggests that ReLU may be a better choice for this particular application due to its

ability to improve the model's convergence. Overall, these results support part of the claim made by S. Dubey, S. Singh, and B. Chaudhury [4], that the ReLU function is a superior activation function and Sigmoid is the worst.

Batch Normalization: The results show that models with Batch Normalization (BN) perform better than those without BN. This suggests that BN helps provide regularization and avoid overfitting, thus improving the model's accuracy. These results support the claim made by C. Garbin, X. Zhu, and O. Marques [5] that adding BN improves a model's effectiveness.

Dropout Layers: Table 6 shows that the model with a dropout layer rate of 0.9 outperformed all other dropout layer rates. Dropout layer rates ranging from 0.1 to 0.7 have roughly the same accuracy, with lower dropout ranges achieving their peak accuracy faster compared to models with higher dropout layer rates. However, higher dropout layer rates are less accurate than some lower ones. This indicates that it takes longer for models with higher dropout layer rates to reach their highest accuracy and increasing the dropout layer rate is only effective up to a certain point before the accuracy starts decreasing.

Based on the findings presented in Table 6, the hyperparameters that yield the best results are Input Units set to 8000, the use of SGD as the optimization function, ReLU as the optimization function, inclusion of Batch Normalization and a dropout layer rate of 0.9. We used these hyperparameters to create the "Best" model which is then compared with the three models in Table 3. A complete breakdown of the 'Best' model can be found in Figure 6

	Accuracy (%)
Default Model	73.68
Best Model	83.02
AWF Model	80.70
DF Model	**93.12**

Table 7: CNN Model Comparison Results

Table 7 shows the accuracy and loss of the four CNN models: the 'Default' hyperparameters model, the 'Best' hyperparameters model, the AWF model, and the DF model. Notably, the DF model outperforms the other models significantly in terms of both accuracy percentage and loss, followed by the 'Best' model, while the 'Default' model performs the poorest. In addition, the 'Best' model demonstrates approximately 10% higher accuracy than the 'Default' model in the full dataset and exhibits a reduction in loss by 0.3.

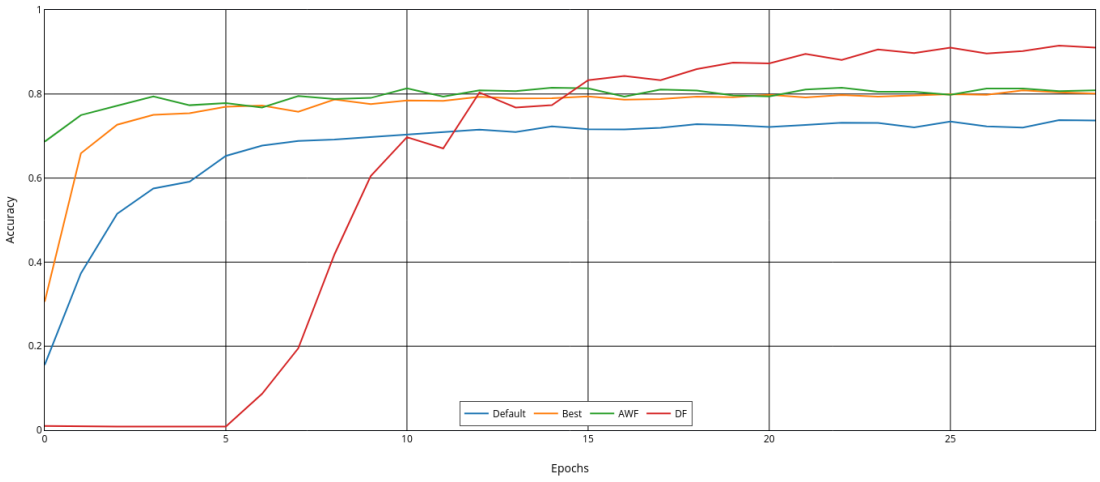

Figure 2: Accuracy over Epochs for CNN Models

Figure 2 illustrates the classification accuracy of website fingerprinting undefended traffic using a

CNN in a closed world experiment, where the tested models' accuracy is observed over multiple epochs. The graph shows that, for all models, the accuracy improvements are less noticeable after 25 epochs. The figure indicates that the AWF model reaches its peak accuracy the fastest, followed by the 'Best' model and the 'DF' model takes the longest. However the DF model catches up and surpasses the other models in about 15 epochs. We can also see that there is a significant improvement in the 'Best' model as compared to the 'Default' model with it consistently providing a higher accuracy quicker.

Note: The 'Best' model was constructed by varying only a few of the hyperparameters of the 'Default' model. We believe that this model could be further improved by modifying some of the other hyperparameters such as the number of hidden layers, the density of hidden layers, the number of filters and filter sizes and the number of convolutional layers.

3.2 LSTM

The first crucial hyperparameters to train for the LSTM are the optimizer and the learning rate because they will determine how the rest of experiments will be trained. Because each optimizer is different, it seemed appropriate to try each learning rate on each optimizer. For the purposes of testing we chose to start with one LSTM layer with 64 units and no dropout. During testing, we measured training loss to determine which optimizer was the best at training without considering overfitting for now. This gave us the results in Figure 3. As you can see SGD struggled to converge. Both Adam and RMSprop achieved good results, but RMSprop was smoother, and therefore a better choice. Looking at the learning rates, 0.01 performed the best without becoming unstable. Therefore we will use RMSprop with a learning rate of 0.01 for the remaining experiments.

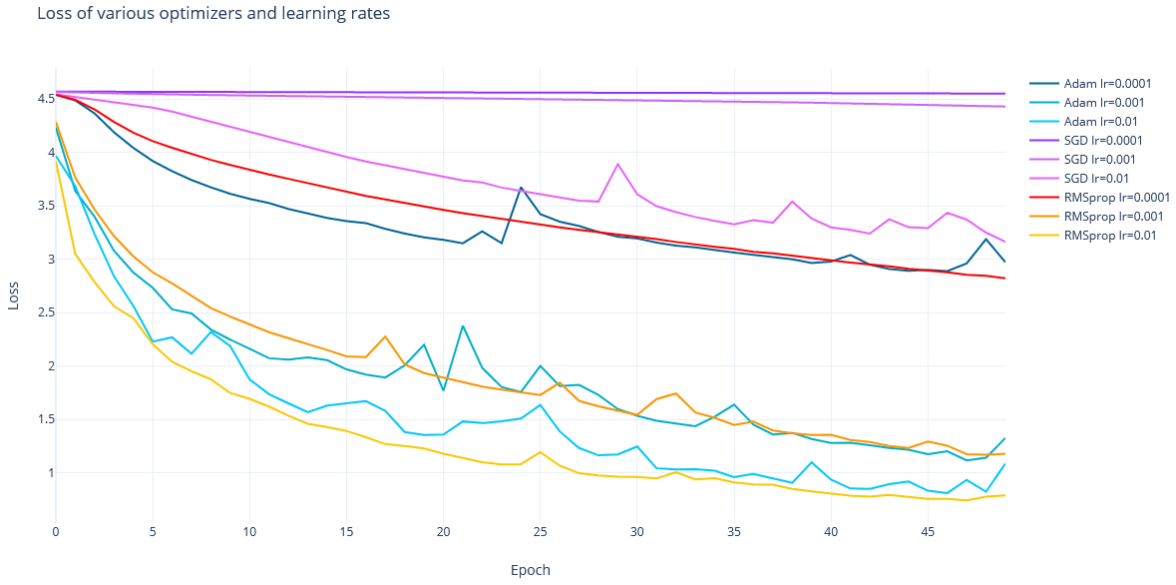

Figure 3: The Performance of LSTM With Various Optimizers and Learning Rates

The next step was to pick an optimal dropout that prevented overfitting without sacrificing performance. We continued to use the model with one LSTM layer with 64 units. To evaluate performance and overfitting, we measured training accuracy (labeled as accuracy) and test accuracy for each dropout level, producing the results in Figure 4. With no dropout, significant overfitting is present, as evident by the gap between the training and test accuracies. On the other hand, a dropout rate 0.3 resulted in a relatively poorer result and erratic overfitting. Therefore we conclude that a dropout rate of 0.3 was too large and prevented the model from learning meaningful parameters. Both 0.1 and 0.2 appeared to have decently low levels of overfitting with 0.1 performing slightly better overall. Therefore we chose to use a dropout rate of 0.1 for the remaining LSTM experiments.

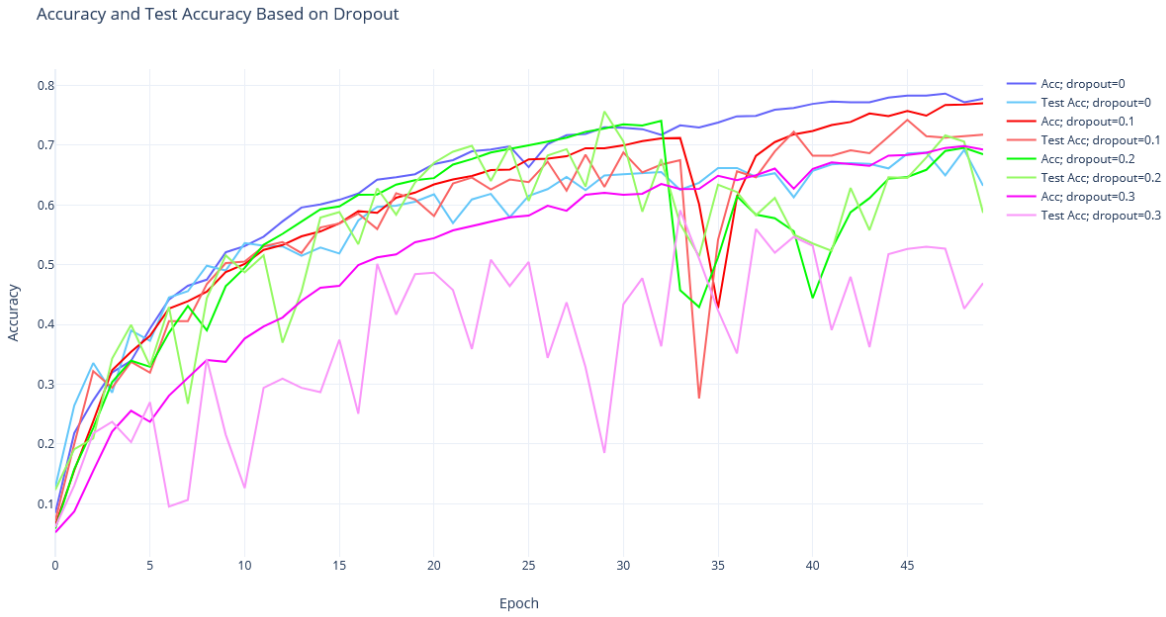

Figure 4: The Performance of LSTM With Various Dropout Rates

Finally, we tested various shapes of the LSTM layers and recorded their final accuracies. This yielded the results shown in table 8. Based on these results, we can conclude that using two layers performed better than one. However, both models that used two layers performed similarly. Therefore we don't conclude that one is necessarily better at solving the problem than the other.

LSTM layers shape	Accuracy
[64]	79.75%
[64, 64]	89.47%
[128]	82.17%
[128, 64]	87.55%

Table 8: Table of accuracies and losses of LSTM models by shape

This meant that the best LSTM model tested had two LTSM units with 64 units each, a dropout of 0.1-0.2 on each LSTM layer, and used RMSprop with a learning rate of 0.1 to optimize. However, the LSTM did not get as good results as the CNN or SDAE, so we wouldn't recommend using it. All of the results are shown in Table 9.

		Accuracy (%)
Optimizer / learning rate	SGD/0.01	21.05
	SGD/0.001	2.35
	SGD/0.0001	1.30
	RMSprop/0.01	**63.22**
	RMSprop/0.001	60.12
	RMSprop/0.0001	27.80
	Adam/0.01	55.79
	Adam/0.001	59.81
	Adam/0.0001	22.79
Dropout	**0.1**	**71.76**
	0.2	**70.58**
	0.3	46.93
Layers Shape	[64]	79.75
	[64, 64]	**89.47**
	[128]	82.17
	[128, 64]	87.55

Table 9: LSTM Hyperparameter Results

3.3 SDAE

Table 10 presents the performance of using SDAE to conduct WF attack under different hyperparameter configurations. The recorded data in the table are the three-time average validation accuracy and validation loss.

Layer Structure: The results indicate that adding an extra layer with 250 hidden units yields slightly better performance than other layer configurations. However, we consider this performance difference insignificant given the fluctuation in accuracy and loss during the training process. While reducing the units of each layer by half and adding an additional layer did not significantly affect performance, the validation loss of the model increased significantly when a layer was removed.

Activation Functions: The ReLU function was found to perform the best with overall higher accuracy and lower loss compared to other choices. The second best is the Sigmoid function, but its loss increases faster than the baseline as the number of epochs increases. ELU has an overall higher loss and lower accuracy compared to the baseline, which we suspect is due to the negative values it generates. Interestingly, the Softmax function gave significantly worse performance compared to the other functions.

Optimization Functions: The different optimizers show clear differences in performance. The baseline Adam optimizer outperformed the others, followed by Adagrad and SGD. Although RMSProp has comparable accuracy with Adagrad and SGD, it has significantly higher validation loss and fluctuation.

Noise Level: The change in noise level also did not result in a distinct performance change. This may be due to the addition of batch normalization already reducing the overfitting. However, we observed that increasing the noise level slightly increases both the loss and accuracy.

As discussed previously, our best SDAE configuration is the baseline model. we were unable to achieve significant performance improvements by varying these hyperparameters. Since the default model has already been fine-tuned and the only differences between the default model and our baseline model are the number of epochs, units per layer, and the presence of batch normalization, we expect similar results.

		Acc(%)
Layer Structure	[2000,1000,500]	96.1
	[2000,1000]	96.2
	[1000,500,250]	96.3
	[2000,1000,500,250]	96.2
Activation Functions	ReLU	96.1
	Sigmoid	95.1
	ELU	94.0
	Softmax	58.2
Optimizer Functions	Admax	96.1
	SGD	92.1
	RMSProp	92.5
	Adgrad	93.6
Noise Level	0.3	96.1
	0.0	95.2
	0.1	94.5
	0.6	96.0

Table 10: SDAE Hyperparameter Results

A Appendix

ChatGPT, developed by OpenAI, was utilized to brainstorm and enhance the quality of the paper. Prompt Used: "Improve the grammar and flow of this paragraph

Model	Input Units	Activation	Optimization	Dropout	BN
1	1000	TanH	SGD	0.5	No
2	3000	TanH	SGD	0.5	No
3	4000	TanH	SGD	0.5	No
4	5000	TanH	SGD	0.5	No
5	6000	TanH	SGD	0.5	No
6	7000	TanH	SGD	0.5	No
7	8000	TanH	SGD	0.5	No
8	9000	TanH	SGD	0.5	No
9	10000	TanH	SGD	0.5	No
10	2000	TanH	Adam	0.5	No
11	2000	TanH	Adamax	0.5	No
12	2000	TanH	RMSProp	0.5	No
13	2000	RELU	SGD	0.5	No
14	2000	SELU	SGD	0.5	No
15	2000	ELU	SGD	0.5	No
16	2000	Sigmoid	SGD	0.5	No
17	2000	TanH	SGD	None	No
18	2000	TanH	SGD	0.1	No
19	2000	TanH	SGD	0.3	No
20	2000	TanH	SGD	0.7	No
21	2000	TanH	SGD	0.9	No
22	2000	TanH	SGD	0.93	No
23	2000	TanH	SGD	0.95	No
24	2000	TanH	SGD	0.97	No
25	2000	TanH	SGD	0.99	No
26	2000	TanH	SGD	0.5	Yes
Default Model	2000	TanH	SGD	0.5	No
Best Model	8000	ReLU	SGD	0.9	Yes
AWF Model	3000	ReLU	RMSProp	0.1	No
DF Model	5000	[ELU, ReLU]	AdaMax	[0.1, 0.5, 0.7]	Yes

Table 11: CNN Models Used

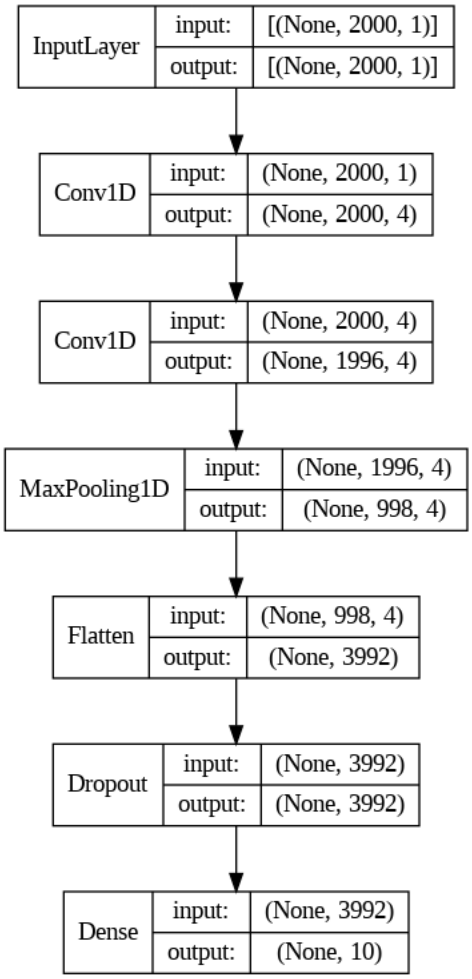

Figure 5: CNN Default Model

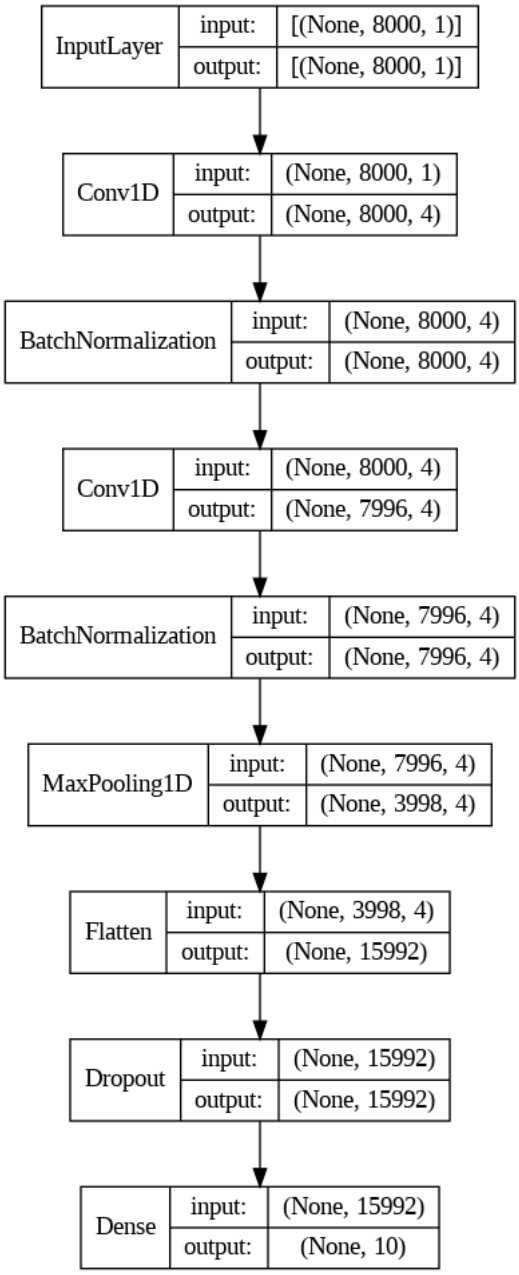

Figure 6: CNN Best Model

References

[1] Kota Abe and Shigeki Goto. "Fingerprinting attack on Tor anonymity using deep learning". In: *Proceedings of the Asia-Pacific Advanced Network* 42 (2016), pp. 15–20.

[2] Somenath Bera and Vimal Kumar Shrivastava. "Analysis of various optimizers on deep convolutional neural network model in the application of hyperspectral remote sensing image classification". In: *International Journal of Remote Sensing* 41 (2020), pp. 2664–2683.

[3] Shayak Chakraborty et al. "Study of Dependency on number of LSTM units for Character based Text Generation models". In: *2020 International Conference on Computer Science, Engineering and Applications (ICCSEA)*. 2020, pp. 1–5. DOI: 10.1109/ICCSEA49143.2020.9132839.

[4] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. "Activation functions in deep learning: A comprehensive survey and benchmark". In: *Neurocomputing* 503 (2021), pp. 92–108.

[5] Christian Garbin, Xingquan Zhu, and Oge Marques. "Dropout vs. batch normalization: an empirical study of their impact to deep learning". In: *Multimedia Tools and Applications* 79 (2020), pp. 12777–12815.

[6] David C. Marcu and Cristian Grava. "The impact of activation functions on training and performance of a deep neural network". In: *2021 16th International Conference on Engineering of Modern Electric Systems (EMES)* (2021), pp. 1–4.

[7] Vera Rimmer et al. "Automated Website Fingerprinting through Deep Learning". In: *ArXiv* abs/1708.06376 (2017).

[8] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. 2017. arXiv: 1609.04747 [cs.LG].

[9] Payap Sirinam et al. "Deep Fingerprinting: Undermining Website Fingerprinting Defenses with Deep Learning". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (2018).

[10] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *J. Mach. Learn. Res.* 15 (2014), pp. 1929–1958.

[11] S. Vani and Telu Venkata Madhusudhana Rao. "An Experimental Approach towards the Performance Assessment of Various Optimizers on Convolutional Neural Network". In: *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)* (2019), pp. 331–336.

[12] Pascal Vincent et al. "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion". In: *Journal of Machine Learning Research* 11.110 (2010), pp. 3371–3408. URL: http://jmlr.org/papers/v11/vincent10a.html.

[13] Changyong Yu et al. "LLR: Learning learning rates by LSTM for training neural networks". In: *Neurocomputing* 394 (2020), pp. 41–50. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2020.01.106. URL: https://www.sciencedirect.com/science/article/pii/S0925231220301703.