Investigating The Effectiveness of Neural Networks in Cryptanalysis

Research Question: To What Extent Can Neural Networks Be Used in the Cryptanalysis of Symmetric Encryptions?

Jerry Wu

Word Count: 4061

September 4, 2022

Contents

I. INTRODUCTION

Thousands of years ago, the ancient Egyptians utilized substitution ciphers to preserve the secrecy of ritual life from outside intruders [20]. The evolution of cryptography has developed so much that users do not even understand what goes on in their computers to ensure their data are encrypted and secure from cyberattacks.

In the past decades, cryptography technology has begun to develop along with the software and hardware, to enhance the cybersecurity of users, which has been updated in methods and forms. Even though cryptography can encrypt a message into a cipher text that makes it illegible, it is also essential to understand how ciphered texts are produced using an algorithm, meaning "a step-by-step procedure for solving a problem or accomplishing some end" [2]. Therefore, by reverse engineering the encryption algorithm, it is possible to find a pattern in the encryption process, thus a way to decrypt the ciphered text without knowing the key for encryption. The rise in popularity of machine learning has improved the accuracy of its usage in facial recognition, autonomous driving, virtual assistance, regression modeling, classification, clustering, and, most importantly, predictions of all sorts [6]. Accordingly, machine learning can be linked with cryptography to see whether it is possible to decrypt encrypted messages by knowing numerous plaintexts and their corresponding cipher text. This process of decoding messages from non-readable text without knowing the key is known as Cryptanalysis [10].

In this investigation, I will attempt to discover whether the artificial neural network is efficient in decrypting messages encrypted by the Data Encryption Standard (DES) without knowing the 56-bit key used to encrypt the messages. The dataset will be every 8-byte word in the English dictionary turned into binary bits. The application of DES is no longer the NIST federal standard [8], but a derivation, Triple-DES, is now used broadly in finance and credit card payment [12]. Being able to decrypt DES signs, a breach of data security will emerge.

II. THEORETICAL BACKGROUND

A. Artificial Neural Networks

Artificial neural networks, also known as neural networks or neural nets, are a subset of machine learning, which uses data and algorithms to replicate how human beings learn. The name neural network is inspired by the biological structure of the human brain, where neurons connect to other neurons to transfer information when given an input. This neural network is known as supervised learning, where the inputs are labeled values to be able to predict or classify values [13]

1. Neurons

Each one of the nodes, or artificial neurons, has a weight and threshold value (bias). Similar to the human brain, when the output of a node is larger than the threshold, the node is activated through an activation function to send data to the next layer of nodes. More in-depth, the formula for a node to fire the activation function is,

$$\sum_{i=1}^{m} w_i x_i + b = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

$$output = f(x) = \left\{ \begin{array}{ll} 1, & \text{if } \sum w_1 x_1 + b \geq 0 \\ 0, & \text{if } \sum w_1 x_1 + b < 0 \end{array} \right\}$$

Where m is the weight generated once the input layer is determined, x is the input or the training data, and b is the bias (threshold value). The output is then sent using the activation function, acting as an input for the next node. In this investigation, we will use the ReLU activation function since our output will only be either 0 or 1 [22]. Hence any exponential or logarithmic activation would be unnecessary since the output will be rounded up to binary. Below is the graphical demonstration of ReLU.

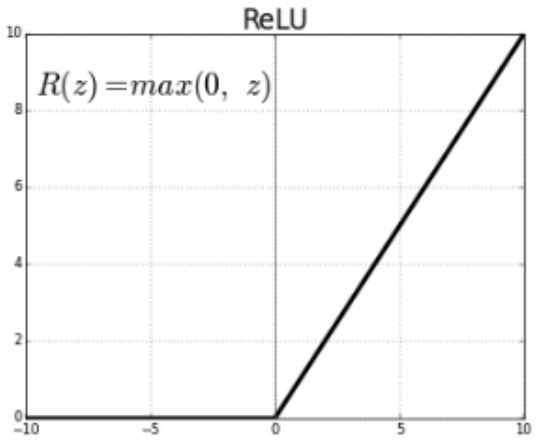

Figure 1: caption goes here

2. Layers

As mentioned before, neural networks are made of layers consisting of 3 major types of layers: input layer, hidden layer, and output layer. The input layer is the first layer that the training data is fed into. Through each epoch - the number of cycles in training - the input layer will take in data from the output layer from the previous node. Unlike the input and output layers which are all singular, the number of the hidden layers depends on the investigation. The hidden layer consists of a designated amount of nodes, which act as a parameter for training and where essentially, the learning of the algorithm happens.

3. Multi-layer Perceptrons

Multilayer Perceptrons or feedforward neural network is a typical neural network system used for classification. A perceptron is one unit of the node, which is the simplest form of a neural network. The diagram below is a typical example of an MLP, consisting of an input layer, a hidden layer(s), and an output layer. This type of neural network is especially useful in real life since the causation of an event usually takes in other correlations, thus imitating when a human considers multiple factors when considering a problem.

The reason why this investigation will be using an MLP with multiple layers of hidden layers instead of a singular layer is that the DES encryption has 16 rounds of

encryption rather than a single round. Hence it is more appropriate for us to use MLP to simulate the scenario where the multiple layers and nodes are used to imitate the steps in DES encryption methods.

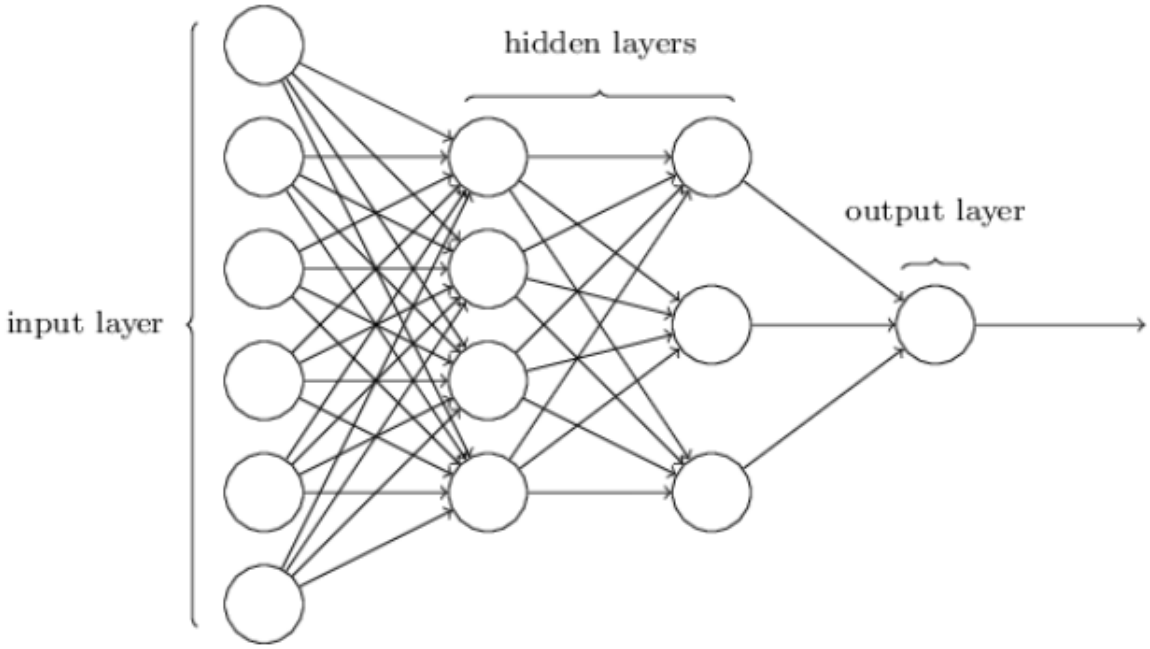

Figure 2: caption goes here[14]

4. Cost Functions

A cost function or loss function is a way to evaluate the accuracy of the machine learning model. It is a function that interprets the value between the predicted value and the absolute value. It is one of the most important indexes to look at while training a machine learning model, and the goal is to minimize the value of the cost function [19]. The cost function is utilized to adjust the weight and bias of the algorithm; with each different function, the metrics for determining the weight are also different.

In our investigation, we will use the binary cross entropy loss function [19]. This cost function was inspired by the information theory, where bits are lost during the transfer between layers to layers. Since our data input will be in the form of binary bits, it is suitable for the investigation. Here is the formula for calculating the binary cross entropy function,

$$Loss = -\frac{1}{m}\sum_{i=1}^{m} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot (1 - \hat{y}_i)$$

Where i is the sample index, $\hat{y}$ is the predicted value, y is the expected value, and m is the number of samples in the dataset. In other related works, researchers have found that the mean square error loss function [3] was also appropriate for the scenario, with a 97% accuracy and 0.00586 margins of outside error. Below is the formula for the MSE,

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

5. Optimizer (Gradient Descent)

Optimizers are tools to minimize the loss function by updating weights and biases of the internal parameters of the model. A standard optimization algorithm is called gradient descent. In the diagram below, on the y-axis, there is the value given by the cost function, and on the x-axis, we have the weight of the input. The optimizer calculates the gradient and tries to determine the minimum cost, where the model is the most accurate. Moreover, the optimizer we will be using is Adam [15] (Adaptive Moment Estimation), which also utilizes momentum: adding fractions to the previous gradient to the current one.

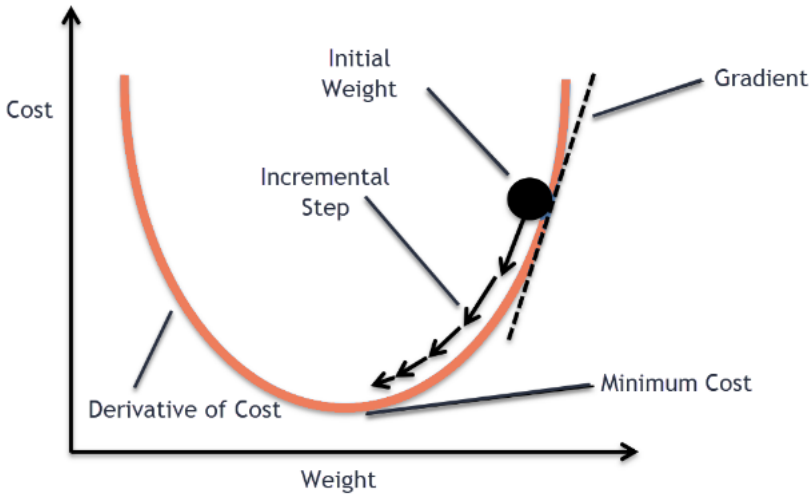

Figure 3: caption goes here[9]

B. Data Encryption Standards & Cryptanalysis

1. Encryption

According to GeeksforGeeks [16], "Encryption in cryptography is a process by which a plain text or a piece of information is converted into cipher text or text which can only be decoded by the receiver for whom the information was intended." The encryption method differs from algorithm to algorithm, and each method is considered a cipher. Currently, the four most famous ciphers widely used are Triple DES, RSA (Rivest-Shamir-Adleman), Twofish, and AES (Advanced Encryption Standard).

These cipher algorithms fall into two genres of encryption: symmetric and asymmetric. In order to understand how to decrypt encryption, it is essential to understand how both of these encryption work for the investigation to have a practical outcome. The following diagrams illustrate how symmetric and asymmetric encryption works. The main difference is that symmetric encryption utilizes only one private key for encryption and decryption. In contrast, asymmetric encryption uses the user's public key to encrypt and the user's private key to decrypt the message sent. This means that asymmetric key encryption is irreversible since, after encrypting, the sender will not be able to retrieve the message back; hence for the sake of investigation, symmetric key encryption would be more appropriate to consider.

2. Cryptanalysis

Cryptography is the study of creating secret messages using encryption methods [1]. On the other hand, Cryptanalysis studies how these encryption techniques can be bypassed without knowing any key used in the encryption, helping develop more secure encryption and find flaws in the system. The method to decrypt is called cryptanalytic attacks, and the two common attacks are linear and differential cryptanalysis. These two attacks require a large quantity of either plain texts and (or) cipher texts. Instinctively, due to the requirement of a large dataset, the job is suitable for machine learning classification, hence the reason to combine cryptanalysis and machine learning. [11]

3. Data Encryption Standards(DES)

Data Encryption Standard (DES) was one of the first symmetric encryption created but soon abandoned as it was not strong enough to prevent brute force cryptanalysis. DES takes 64-bit data in a block as input and produces 64-bit data after the encryption. Depending on the length of the text or the size of data encrypted, DES will generate corresponding blocks. In each block, there will be a maximum of 64-bit, and when a text has a length of 100-bit, for instance, DES will create a new block with 64-bit, and the remaining blank bits will be padded with zeros.

i. Key Generation

Since DES is a type of symmetric encryption, it only requires one private key, which is initially 64-bit, but every 8-bit in the data block is discarded, constructing only a main key of 56-bit. Each round of DES generates a new key through a key transformation where the 56-bit key is halved, and depending on the round; DES performs a circular left shift of one or two indexes. After the shift, the 56-bit is combined back together, while 8 bits are discarded, forming a unique 48-bit key. This process in cryptography is known as Compression Permutation.

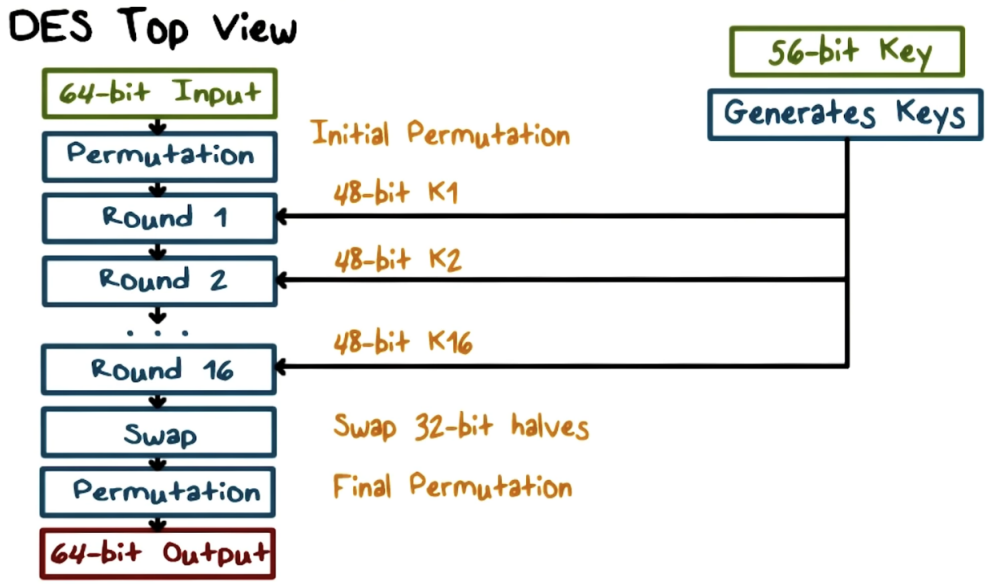

Figure 4: caption goes here[18]

ii. Permutation (Transposition)

After the padding of input and generation of key, the encryption process begins. Firstly the 64-bit input will go under the initial permutation function, a phase in the transposition procedure typically known as the diffusion process of the encryption. This means that the bits have their position switched with other bits to create a random yet reversible new sequence of bits that contains all the original bits in different orders. After the plain text is permuted, it is separated into the two blocks of 32-bits left and right original plain text. The final permutation essentially repeats the initial permutation at the end of the entire algorithm. [18]

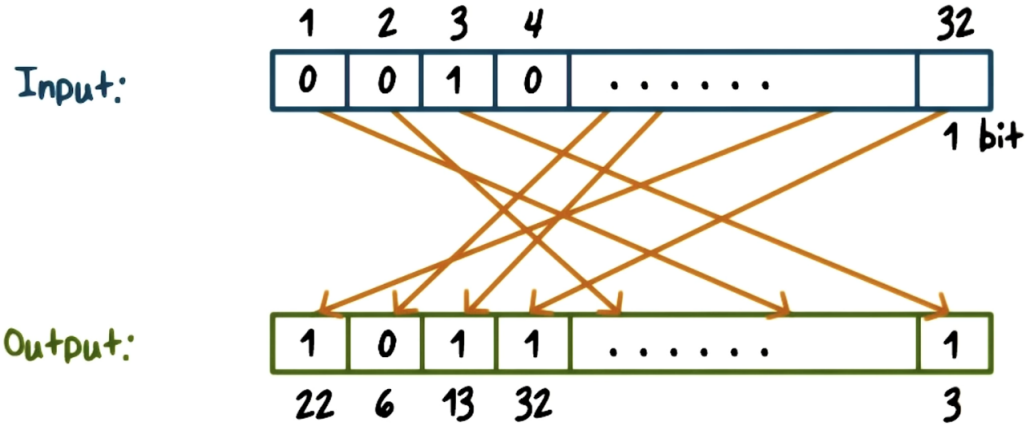

Figure 5: caption goes here[18]

iii. DES Round

Every round of DES has the same operation with different inputs per round key. The input is taken from the previous round of DES or the initial permutation as left and right input blocks with 32 bits. From the diagram, we can see that in the output of a DES round, the left half output is simply the input of the original right block, and the right half of the output is the outcome of XORing the left half input and the mangler function of the original right half. The mangler function [4] contains:

1. Expansion Permutation, dividing 32-bit into eight blocks of 4-bit, then adding 2-bit in each block, and lastly performing bit permutation on all eight blocks to result in a 48-bit block.

2. XOR Swapping [4] performs three rounds of XOR (Exclusive or) on the text with the 48-bit per round key, concluding with a 48-bit block.

3. S-Box Substitution uses subblock and s-box to convert the 48-bit block into a 32-bit block.

4. P-box Permutation permutes the 32 bits into a 16 by two permutation table.

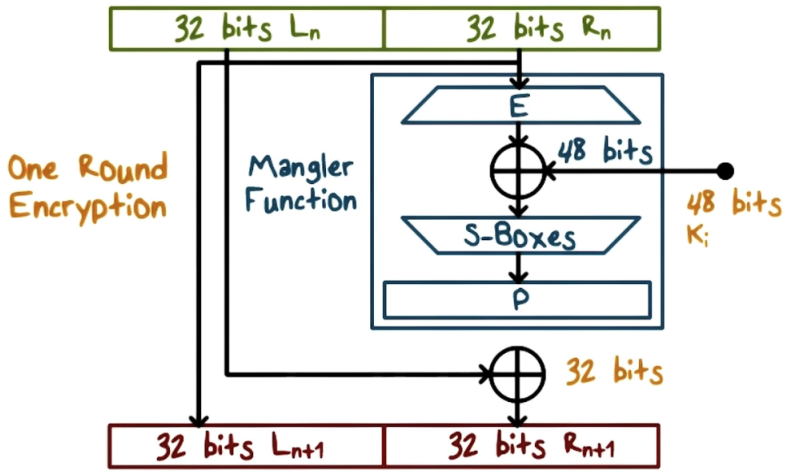

Figure 6: caption goes here[18]

III. METHODOLOGY

A. Investigation

In this investigation, a huge problem was the way to represent the input and output to fed into the network, as well as the structure choice of the algorithm that can efficiently decrypt a cipher text with the minimum loss and highest accuracy. The investigation utilized Google Colab to train the machine model online; while using the programming language of Python, I was able to create and modify the dataset and implement the DES algorithm (see appendix).

B. Experimental Procedure

1. Download all the words in the English dictionary
2. Collect all the words that have a character length of 8 through filtering
3. Generate two random binary bits as plain text with length of 64 bits in total
4. With the DES algorithm, encrypt all the words
5. Converted the plain text and cipher text into binary of 64 bits
6. Split the 64-bit block into an array of '1's and '0's and export it to a CSV file.
7. Split the test sample into training and testing
8. Upload the files into Google Colab
9. Train the model using scikit-learn using *Binary Cross Entropy* as loss function and *ADAM* as optimizer.
10. Generate predictions on the test data
11. Hypertune the learning rate, epochs for maximum accuracy, and minimum loss
12. Repeat steps 8-10 until an appropriate loss and accuracy.

C. Data Generation & Processing

In terms of generating the dataset, my approach was first to collect words [23] with a character length of 8, which is equal to 64 bits and 8 bytes. Thus without padding, the DES can encrypt every message with the same length to minimize outside error. Then with the DES algorithm [7], the plain text gets encrypted into bits, which cannot be directly converted into binary. Hence it is necessary to decode the bits into Latin and convert that string into binary. To compare the real life scenario where not all messages are english, we've also generated 64 random bits as seperate dataset. Here is an example of the dataset, where the cipher text is the input, and the plain text is the output.

Encrypted Text	Plain Text
0100001000100100100010010011011110001110001000101101010100011111	0111001101101000011010010110111001100111011011000110010101110011
0000010101000110111010010111011011011111010000110010011110111000	0111001101101000011010010110111001101001011001010111001101110100
1101001111100000101011010000111000101111000110010111111010010000	0111001101101000011010010110111001101100011001010110000101100110
1100011010100001101100100001010010011111001011111011100110100001	0111001101101000011010010110111001101110011001010111100101110011
0101101011100001010110011101100100110000111100001010100101001101	0111001101101000011010010110111001101110011001010111001001111001
1001100000011111100100101100100010001101101011001111101101010110	0111001101101000011010010110111001101110011010010110010101100100
0000011100100010000101111010000100111000111010001100111011111000	0111001101101000011010010110111001101110011010010110010101110011
1110000011111001110000101111001001001111100111111111111001000011	0111001101101000011010010110111001101110011010010110111001100111
0110100001011001111101100010010111000111110001001101000001000000	0101001101101000011010010110111000101101011100110110100001110101
0010000111010110111100100010011000101000010010111111100101111001	0111001101101000011010010110111001110100011110010110000101101110
0100011111111011111101000111101000000011010000100000111101100000	0101001101101000011010010110111001110111011000010111001001101001
1101111001101010101011101000111000110100010100001110011110011101	0111001101101000011010010110111001110111011011110110111101100100
1101001111111011101000110000010110000010011111001101100111110111	0101001101101000011010010110111101100011011101000110111101101110
1110101111101010010100010101011110100111010101100001000100011011	0111001101101000011010010111000001100110011001010111001001100100
0111101001110110011100101100101001011110011001110011000001101110	0111001101101000011010010111000001100110011101010110110001110011
0101111111001011110001001001111000100100000000101000110111111101	0111001101101000011010010111000001101000011010010111001001100101
1000010100001001010011011001110101101000100010111110001001100100	0111001101101000011010010111000001111001011000010111001001100100
1000111100110011100100101000001101111010010001011100000110011011	0111001101101000011010010111000001101100011000010111000001110011
0101011010000101011111001011001011111100101010101101001111111011	0111001101101000011010010111000001101100011001010111001101110011
0110100111111110101000100100100000110100111111010110100110110010	0111001101101000011010010111000001101100011011110110000101100100

```
    #converting the 64-bit into an array with 64 items
    for i in range(len(x_train)):
        new.append(list(x_train[i]))
        new2.append(list(y_train[i]))

    for j in range(len(x_test)):
        new3.append(list(x_test[j]))
        new4.append(list(y_test[j]))

    x_train = np.asarray(new).astype(int)
    y_train = np.asarray(new2).astype(int)
    x_test = np.asarray(new3).astype(int)
    y_test = np.asarray(new4).astype(int)

    print(x_train.shape)
    print(x_train[0])
[8]  ✓ 4.2s
...  (48006, 64)
[0 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 0 1 0 1 0 1 0 0 1 0 0 1 0 0 1 0 0 1 1 1 1
 0 1 1 1 0 0 0 0 0 0 1 1 1 0 1 1 0 1 0 1 0 0 0 0 1 1 0]
```

Figure 8: caption goes here

D. Model Architecture

The machine model, training, data importing, prediction, and plotting are written in Python using Sckit-Learn, NumPy, Pandas, and Matplotlib. As mentioned before, I will use multilayer perceptrons to build my algorithm. The hidden layer will have the configuration of nodes such that the four layers will be 256, 512, 512, 256.

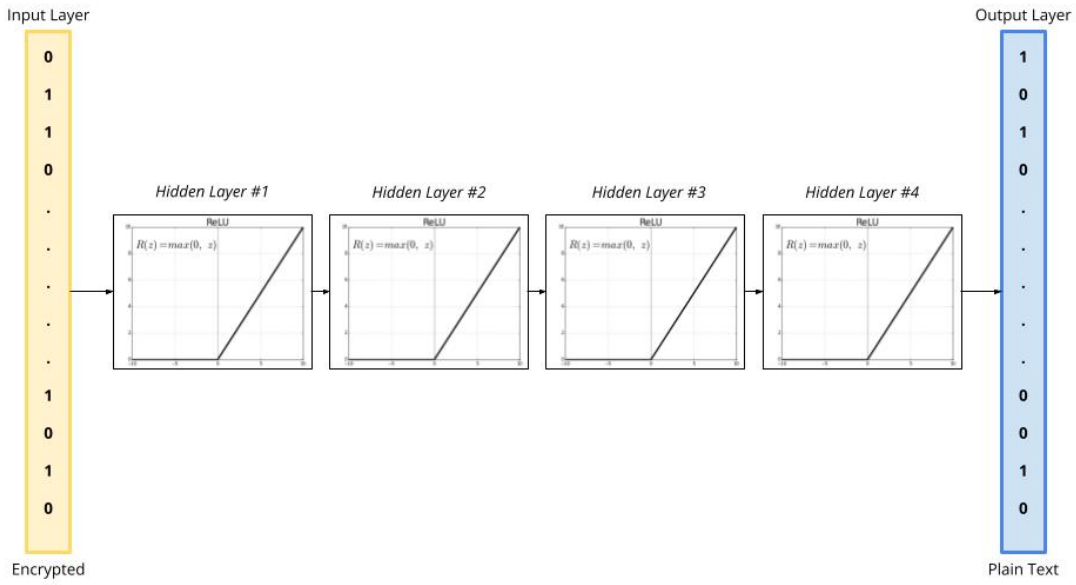

Figure 9: caption goes here[3]

The activation function will be ReLU, as previously mentioned, which is the most suitable and efficient to train this dataset. The loss function would be the default binary cross entropy that of the sckit-learn MLP, which cannot be changed but is also suitable for the case of a large dataset. Below is an illustration of the custom MLP algorithm.

The optimizer used will be Adam; meanwhile, depending on the main focus of the model, the epochs will vary from 3 to 200, which will be explained later. Adam is an algorithm based on the stochastic gradient descent algorithm while combining the RMSProp, and AdaGrad. While comparing to a recently proposed quasi-Newton method (SFO)[15] that performed fabulously on the optimization of MLP, Adam is faster in both epochs and wall-clock time while also using the dropout function to prevent overfitting.

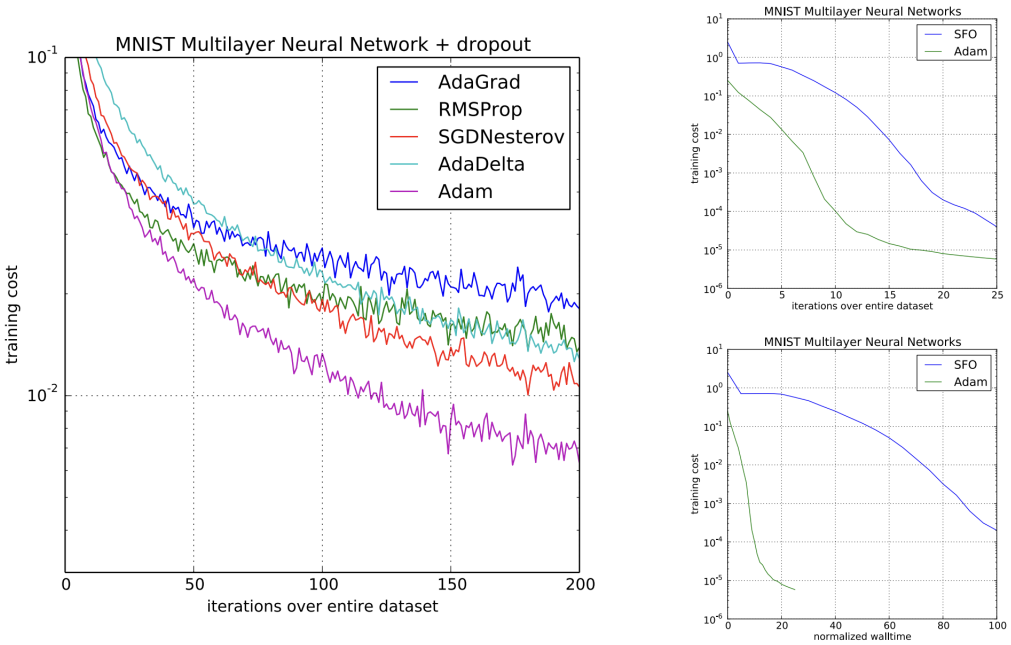

Figure 10: caption goes here[15]

In addition, Learning rate, an additional parameter, regulates how much our neural network's weights are based on the loss gradient. In this case, I choose adaptive learning, which means that for every two epochs, if the loss value has not decreased by the target value, then it will slow down the learning process. This can help decrease the loss value as well as increase the precision and accuracy of the model, at the same time maximizing the time consumed to train this model.

IV. ANALYSIS & CONCLUSION

A. Data Presentation

Dictionary Dataset - Performance Metrics

Table created by taking every 10th epoch for cross validation with the test set of dictionary.

Log Loss:	Accuracy:	Precision:	Recall:	F1 Score:	Hamming Loss:
21.92	0.705	0.72	0.719	0.719	0.295
19.172	0.704	0.717	0.722	0.72	0.296
17.599	0.705	0.718	0.722	0.72	0.295
16.178	0.707	0.722	0.718	0.72	0.293
14.962	0.707	0.721	0.722	0.722	0.293
14.404	0.708	0.723	0.72	0.721	0.292
13.507	0.709	0.724	0.722	0.723	0.291
12.56	0.712	0.729	0.72	0.725	0.288
12.424	0.717	0.735	0.723	0.729	0.283
11.95	0.751	0.782	0.73	0.755	0.249

Dictionary Dataset - Confusion Matrix

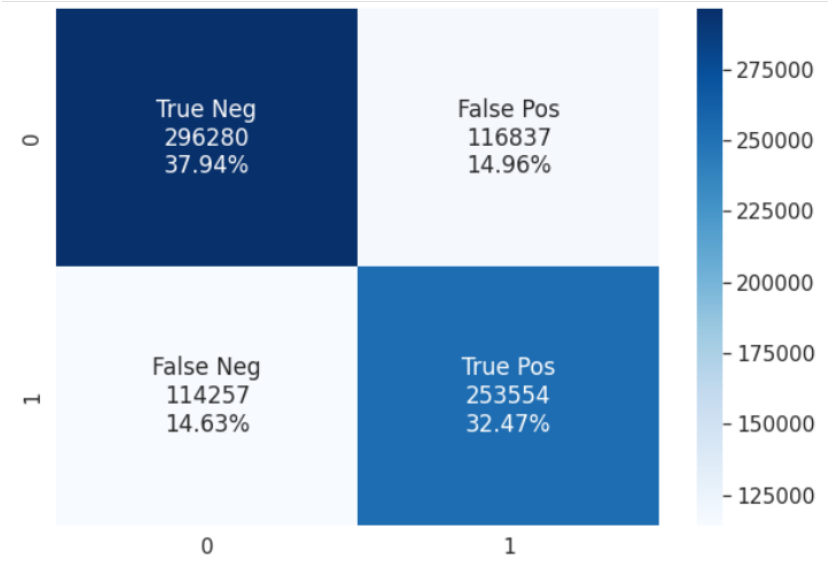

Figure 11: caption goes here

Random Bits - Performance Metrics

Table created by taking every 10th epoch for cross validation with the test set of random bits.

Loss:	Accuracy:	Precision:	Recall:	F1 Score:	Hamming Loss:
44.361	0.500	0.500	0.537	0.518	0.500
44.352	0.500	0.499	0.551	0.524	0.500
44.335	0.500	0.500	0.527	0.513	0.500
44.301	0.501	0.500	0.514	0.507	0.499
44.242	0.501	0.500	0.513	0.506	0.499
44.144	0.501	0.500	0.513	0.507	0.499
44.012	0.500	0.500	0.516	0.508	0.500
43.859	0.500	0.499	0.513	0.506	0.500
43.695	0.500	0.500	0.515	0.507	0.500
43.532	0.500	0.500	0.514	0.506	0.500

Random Bits - Confusion Matrix

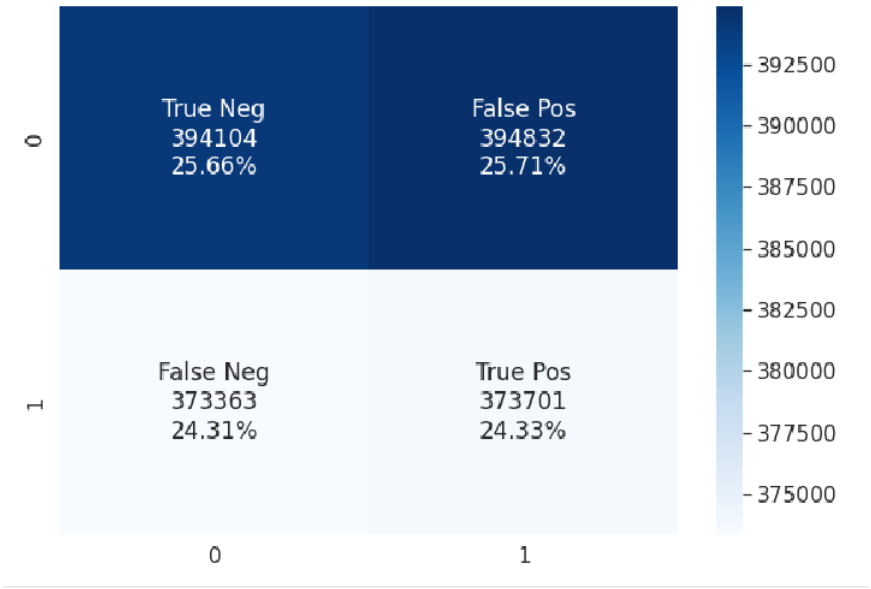

Figure 12: caption goes here

B. Evaluation Metrics & Explanation

After training the neural network for decryption, we need to use the test set to verify specific metrics typically used in classification problems. Even though the algorithm predicts in the form of binary, it is essential to realize that we have 64 binary bits for a single patch, therefore meaning 64 classifications. Hence, it is important to notice that this is not a typical classification problem. Since we are not trying to categorize specific items under a particular class, most of the metrics for classification would not apply and represent the true significance of the algorithm. This type of machine learning algorithm is commonly known as mutli-class, multi-output classification, which has close to known unique evaluation metrics. So the metrics used to evaluate the model are the typical loss, binary accuracy, recall, precision, and the F1 score.

To begin, a way to demonstrate a rough visual representation of the performance that the algorithm had on the binary bits. In figure 11 we can see a confusion matrix that gives the amount of true and false prediction on positive and negative items, in this case the positive would be simply predicting 1 and negative would be predicting 0. This was done manually by calculating true positive, true negative, false positive and false negative. Even though confusion matrix are not normally used in the verification of multi-ouput binary classification problems, it offered a great summary of the performance in terms of balances in dataset and training.

The loss on the other hand can be seen in two different ways, one as the binary cross-entropy function, or a loss metric called hamming loss. Binary cross entropy or log loss was the function used in the training of algorithm, which is the negative average of the log of correctly predicted probabilities [21] Hamming loss [5] evaluates how many times a label or index is misclassiffied, meaning that false positive, negative will raise hamming loss, while true positive, true positive is going to minimize the hamming loss.

Accuracy is probably is the most over looked metrics in an algorithm, though it is an extremely important metrics to assess the value of this algorithm. The closer to have a perfect accuracy means that the more accurate the decrypted message will

be. This may sound obvious, but in other scenarios of machine learning, accuracy can be misleading. For example, a binary classification of snow forecast can achieve 99% accuracy by predicting that there will be no snow every single time, but this means that in the condition where there is suppose to be snow, the predictor still predicts no snow, creating a useless algorithm with high accuracy but no potential use. In this case, the accuracy calculated is by looking at each index of the 64 bits rather than the encrypted message as a whole. The formula to calculate the accuracy is shown below,

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

By using the values in the confusion matrix, we can obtain values of precision, recall, and the f1-score. The precision represents the percentage of positive instances out of the total positive predictions. This allows us to understand how often the model predicts true and it is right. The recall, on the other hand, is defined as the percentage of true positive predictions out of all positive values. Lastly, the F1 score is the harmonic mean between precision and recall.

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FP}$$

$$\text{F1 score} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Though the metrics of precision and recall are used in this investigation, it does not seem very suitable for the scenario since these metrics treat the problem as a typical classification problem. In our case, 1 and 0 are both important, but these metrics only give us information about the positive predictions. The reason for including these metrics is for future improvement, as these metrics provide an insightful contribution to some issues with the dataset and model itself that we will be discussing.

C. Result Analysis

The first abnormal metric we see in the data is the extremely high log loss rate of 11.95 for the dictionary dataset. This is because the loss value shows the entire loss for all 64 bits since they are below a single array. If we divide 11.95 by 64, we get a very good log loss of 0.187. This means that the model, in general, predicted closely what the true value is; the hamming loss values also demonstrated that this was the case as only 24.9 % of the 1,536,000 bits were incorrect. The accuracy of the first model with a dictionary as input has a high accuracy of 75% in the final epoch. In contrast, the precision and F1 score rapidly increased in the end, approaching the "peak" of training. Recall, on the other hand, was always hovering around the 7.2 to 7.3 section without a significant increase or decrease. Having an accuracy of 75% means that the model successfully predicted 75% of the bits that have been expanded, which does not seem to be a bad result knowing that the key for encryption was not given, it was able to create a new algorithm to find the relationship between the encrypted and decrypted message.

If we look at the evaluation metrics for the random bits dataset, we can see that most values hovered at 0.5, which is not a good sign. The loss value was extremely high, and after reviewing the dataset, the conclusion for this dataset was that since the plain text was 64 random bits, it was treated as a string length of 64, meaning 512 bits instead of 64 bits, despite them being '1's and '0's in a binary fashion. Therefore, unlike the dictionary dataset, all the words were selected to be 8 bytes. The metrics value then makes sense as the model has a 50% chance of guessing whether or not the value is 0 or 1, hence producing 0.5 for accuracy, precision, F1 score, and hamming loss. Therefore from now onward, the dictionary result will be the primary focus of this investigation.

D. Limitations of Investigation

In this investigation, the key was set to be the same for all encryption, but keys can change in real encryption; hence it is hard to know if the encrypted message is encrypted by the same key or different keys, creating an uncertainty even if there was a large amount of plain text and encrypted text. The evaluated accuracy 75% represents the probability of predicting the correct binary bits, but when you transfer binary bits back to ASCII for legible text, even if there was 1 bit of error, the character would be completely different than the original text, hence the model would not be functioning if we don't have close to 99% accuracy to obtain the perfectly decrypted message.

Another limitation of this investigation can be the disadvantage that an MLP has when there are too many layers and nodes. In my investigation case, I utilized over four layers of the hidden layer and a total of 1536 nodes, excluding the input and output layer, the connection between each node creates a heavily chaotic web of neuron firing and weight calculating. This can cause unnecessary redundancy and inefficiency [24]

Another limitation of the investigation was the model chosen. However, the MLP classifier model was the most appropriate and most efficient for multi-class, multi-output neural networks. It also limited the loss function to cross-entropy(log loss), so I could not experiment with other loss functions to hyper-tune the model. It also did not have an option for dropout layers which ensures the model does not overfit or underfit, meaning when training using the training dataset, it will generalize instead of focusing on the training model itself solely, causing the accuracy and different metrics not to be optimal when predicting the test dataset.

Lastly, the neural network would not be used in a real decryption scenario, since it is unethical to decrypt without permission in inappropriate scenarios. It is hard to obtain such a large amount of plain and cipher text in real-life situations, plus we do not know which encryption standard is used for the encryption.

E. Conclusion

In conclusion, this investigation aimed to assess the ability of neural network for cryptanalysis of DES encryption. After the experiment on both dataset, I was led to the conclusion that even though there are some issues in the engineering of the model, especially when the data is generated randomly, but there is a visible connection between utilize neural network and cryptanalysis of symmetric encryptions when the original data is legible.

V. APPENDIX

Machine Learning Training & Analysis

```
#Useful libraries
import sys
import numpy as np
import os
import pandas as pd

#Common Model Helpers
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, confusion_matrix,
                                    precision_score, hamming_loss,
                                    roc_curve,average_precision_score
from sklearn.metrics import plot_confusion_matrix, auc,roc_curve,
                                    log_loss, f1_score,
                                    matthews_corrcoef
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

#Visualization
import seaborn as sn
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.pylab as pylab
from pandas.plotting import scatter_matrix

#Pickle to save model
import pickle
from joblib import dump, load

#Plot Style
mpl.style.use('ggplot')
sn.set_style('white')
sn.set(font_scale=1.4)
```

```
pylab.rcParams['figure.figsize'] = 12,8

#Import data
dataset = pd.read_csv('dataset.csv')
data = dataset[['1','2']]
x = list(data['1'])
y = list(data['2'])

x = np.asarray(x).astype(str)
y = np.asarray(y).astype(str)

#Splitting dataset
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2,
                                        random_state=43)

new  = []
new2 = []
new3  = []
new4 = []

#Converting the 64-bit into an array with 64 items
for i in range(len(x_train)):
    new.append(list(map(int,x_train[i])))
    new2.append(list(map(int,y_train[i])))

for j in range(len(x_test)):
    new3.append(list(map(int,x_test[j])))
    new4.append(list(map(int,y_test[j])))

#Convert the array from string array to interger
x_train = np.asarray(new).astype(int)
y_train = np.asarray(new2).astype(int)
x_test = np.asarray(new3).astype(int)
y_test = np.asarray(new4).astype(int)
```

```
#Fitting train & test input
scaler = StandardScaler()
scaler.fit(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)
x_test = scaler.transform(y_test)

#Classificiation for True Negative, True Positive, False Negative &
                                    False Positive
def classfication(predictions):
    TN = 0
    TP = 0
    FN = 0
    FP = 0

    for i in range(len(predictions)):
        for j in range (len(predictions[i])):
            if predictions[i][j] == 0 and y_test[i][j] == 0:
                TN = TN + 1
            elif predictions[i][j] == 1 and y_test[i][j] == 1:
                TP = TP + 1
            elif predictions[i][j] == 0 and y_test[i][j] == 1:
                FN = FN + 1
            elif predictions[i][j] == 1 and y_test[i][j] == 0:
                FP = FP + 1

    return TN,TP,FN,FP

#Metrics for evaluation
def metrics (TN,TP,FN,FP):
    accuracy = (TP+TN)/(TP+TN+FP+FN)
    recall = (TP)/(TP+FN)
    precision = (TP)/(TP+FP)
    f1 =  (2*(precision*recall))/(precision + recall)
```

```
    return accuracy,recall,precision,f1

#Neural Network Architectural Structure
mlp = MLPClassifier(hidden_layer_sizes=(256,512,512,256),
                    max_iter=1,
                    activation = "relu",
                    verbose = True,
                    solver = "adam",
                    random_state= 60,
                    tol=0.00005,
                    learning_rate = 'adaptive',)

#Define our own metrics
accuracy = []
precision = []
recall = []
hloss = []
loss = []
f1_score = []
mcc= []

#Repeat training to collect metrics data
mlp.fit(x_train, y_train)
dump(mlp, 'predictor.joblib')
for i in range (3):
    mlp = load('predictor.joblib')
    mlp.partial_fit(x_train, y_train)
    predictions = mlp.predict(x_test)
    TN,TP,FN,FP = classfication(predictions)
    acc,rec,prec,f1 =  metrics(TN,TP,FN,FP)
    accuracy.append(round(acc,3))
    precision.append(round(prec,3))
    recall.append(round(rec,3))
    hloss.append(round(hamming_loss(y_test,predictions),3))
    loss.append(round(mlp.loss_,3))
```

```
    f1_score.append(round(f1,3))
    dump(mlp, 'predictor.joblib')
    #print(classification_report(y_test,predictions,zero_division=1))

#Confusion matrix graphing
cf_matrix = [TP,FP,FN,TN]

group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in cf_matrix]
group_percentages = ["{0:.2\%}".format(value) for value in cf_matrix/np
                                    .sum(cf_matrix)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in zip(group_names,
                                    group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)

cf_matrix = [[TP,FP],[FN,TN]]
plt.figure(figsize = (10,7))
sn.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')
plt.savefig(os.path.join(sys.path[0],'confusion matrix.png'))
```

Dictionary Filtering

```
import csv


d = open("dict.txt","r")
data = d.readlines()

for lines in (data):
    if len(lines.rstrip()) == 8:
        d.write(f"{lines}")
```

DES Encryption for Dictionary

```
import pyDes
import random
import csv
import pandas as pd
```

```
def string_to_bits(s):
    new = s.decode("latin-1")
    return ''.join(format(ord(i), '08b') for i in new)

def bytes_to_bits(bytes):
    return ''.join(format(byte,'08b') for byte in bytes )

def generate_data():
    d = pyDes.des("DESCRYPT")
    result = []

    f = open("61008.txt","r")

    for lines in f.readlines():
        indata = lines.strip()
        indatabits = ''.join(format(ord(x), '08b') for x in indata)
        outdata = d.encrypt(indata)
        outdatabits = string_to_bits(outdata)
        result.append((outdatabits,indatabits))
    return result

data = generate_data()

sheet = pd.read_csv("real.csv")

for i in range(len(data)):
    sheet.loc[i,"1"] = data[i][0]
    sheet.loc[i,"2"] = data[i][1]

sheet.to_csv("real.csv",index=False)
```

DES Encryption Algorithm

```
import pyDes
import random
```

```
import csv
import pandas as pd

def string_to_bits(s):
    new = s.decode("latin-1")
    return ''.join(format(ord(i), '08b') for i in new)

def bytes_to_bits(bytes):
    return ''.join(format(byte,'08b') for byte in bytes )

def generate_data(N):
    d = pyDes.des("DESCRYPT")
    result = []
    #Lines of code to show encrypted data being generated
    for i in range (N) :
        indata = random.getrandbits(64).to_bytes(8, "big")
        #
        indatabits = bytes_to_bits(indata)
        outdata = d.encrypt(indata)
        outdatabits = string_to_bits(outdata)
        result.append((outdatabits,indatabits))
    return result
```

References

[1] Yash Agarwal. *Cryptanalysis and types of attacks*. Jan. 2021. URL: https://www.geeksforgeeks.org/cryptanalysis-and-types-of-attacks/.

[2] *Algorithm*. Aug. 2022. URL: https://www.merriam-webster.com/dictionary/algorithm.

[3] Stefan Andonov, Dimitrova Vesna, and Popovska Mitrovikj Aleksandra. *Repository of ukim: Repository of ukim*. Sept. 2020. URL: https://repository.ukim.mk/.

[4] Raed Bani-Hani et al. *High-throughput and area-efficient FPGA implementations of Data Encryption Standard (DES)*. Mar. 2014. URL: https://www.scirp.org/html/1-7600305_44137.htm.

[5] Stefano Bromuri et al. *Multi-label classification of chronically ill patients with bag of words and supervised dimensionality reduction algorithms*. May 2014. URL: https://www.sciencedirect.com/science/article/pii/S1532046414001270.

[6] Ed Burns. *What is machine learning and why is it important?* Mar. 2021. URL: https://www.techtarget.com/searchenterpriseai/definition/machine-learning-ML.

[7] Anurag Chauhan. *Lists of English Words*. Apr. 2020.

[8] *Data Encryption Standard (DES): Set 1*. May 2022. URL: https://www.geeksforgeeks.org/data-encryption-standard-des-set-1/.

[9] Harshit Dawar. *Stochastic gradient descent*. May 2020. URL: https://medium.com/analytics-vidhya/stochastic-gradient-descent-1ab661fabf89.

[10] *Differential and linear cryptanalysis*. Feb. 2022. URL: https://www.geeksforgeeks.org/differential-and-linear-cryptanalysis/.

[11] Shafi Goldwasser. *Ai and cryptography: Challenges and opportunities*. Mar. 2019. URL: https://www.oreilly.com/radar/ai-and-cryptography-challenges-and-opportunities/.

[12] Jasmine Henry. *3DES is officially being retired*. Aug. 2018. URL: https://www.cryptomathic.com/news-events/blog/3des-is-officially-being-retired.

[13] Cloud Education IBM. *What is machine learning?* July 2020. URL: https://www.ibm.com/cloud/learn/machine-learning.

[14] Vishal Jain. *Multi-layer perceptron as a non-linear classifier-03*. Apr. 2021. URL: https://medium.com/analytics-vidhya/multi-layer-perceptron-as-a-non-linear-classifier-03-8cd25147fc23.

[15] Diederik P. Kingma and Jimmy Lei Bai. *Adam: A method for stochastic optimization*. Jan. 2017. URL: https://arxiv.org/pdf/1412.6980.pdf.

[16] Jash Kothari. *Encryption, its algorithms and its future*. July 2019. URL: https://www.geeksforgeeks.org/encryption-its-algorithms-and-its-future/.

[17] Linus Largerhjelm. *Extracting information from encrypted data using Deep Neural Networks*. Jan. 2019. URL: http://www.diva-portal.org/smash/get/diva2:1284274/FULLTEXT01.pdf.

[18] Wenke Lee Lee, Mustaque Ahamad, and Catherine Gamboa. *Data Encryption Standard*. July 2015. URL: https://www.youtube.com/watch?v=Y61qn_SQl40&ab_channel=Udacity.

[19] Mohammed Zeeshan Mulla. *Cost, activation, loss function—— neural network—— deep learning. what are these?* May 2020. URL: https://medium.com/@zeeshanmulla/cost-activation-loss-function-neural-network-deep-learning-what-are-these-91167825a4de.

[20] Roger A. Prichard. *Global Information Assurance Certification Paper - GIAC*. Jan. 2002. URL: https://www.giac.org/paper/gsec/1555/history-encryption/102877.

[21] Shipra Saxena. *Binary cross entropy/log loss for binary classification*. Mar. 2021. URL: https://www.analyticsvidhya.com/blog/2021/03/binary-cross-entropy-log-loss-for-binary-classification/.

[22] Sagar Sharma. *Activation functions in neural networks*. Sept. 2017. URL: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

[23] Todd Twhiteman. *A pure python module which implements the DES and triple-des encryption algorithms*. Apr. 2010. URL: https://github.com/twhiteman/pyDes.

[24] Uniqtech. *Multilayer perceptron (MLP) vs Convolutional Neural Network in deep learning*. June 2019. URL: https://medium.com/data-science-bootcamp/multilayer-perceptron-mlp-vs-convolutional-neural-network-in-deep-learning-c890f487a8f1#:~:text=Disadvantages%20of%20MLP%20include%20too,resulting%20in%20redundancy%20and%20inefficiency..