Efficient, secure, and reliable cryptography using convolution neural network

[1]Ms. Kritika Purohit, [2]Prof. (Dr.) Surendra Yadav, [3]Prof. Veena Parihaar

[1]Research Scholar, [2]Professor, [3]Assistant Professor
[1]Department of computer science and Engineering
[1]Career Point University
Kota (Rajasthan), India

Abstract: Despite the advancements in cryptography, encoding sensitive data remains a relatively difficult operation. Following a study of the current literature, it is now clear that security issues have yet to be overcome, and that more research is needed.The design of the Convolutional Neural Network (CNN) encryption algorithm is presented in this study. The findings are compared to the Cryptographic Algorithms based on a set of parameters. The final findings show how well a particular neural network may be used for symmetric cryptography. The experimental findings demonstrate that, under the terms stipulated in this manuscript, CNN can be used for cryptography. WithCNN it is found thinnest is feasible to attain low error rates with neural networks several times. To show the real-life implementation of CNNs over a TCP/IP network, Alice and Bob's conversation was mimicked using sockets.The research findings reveal that the recommended implementation provides significantly improved security without disturbing the actual process.

Index Terms-- CNN, NCA, TCP/IP, Cryptographic, encryption

I.INTRODUCTION

Over the years, several different encryption techniques have been developed and applied. Some of these are now considered standard. The authors chose to test and compare a well-established traditional symmetric encryption approach with an experimental convolutional neural network algorithm [1]. The test was part of a bigger challenge in which it was determined if the CNN is adequate for encryption or if the AES implementation is a superior solution. For both algorithms, a Python implementation was employed. It's part of a larger project that's also written in Python. Python is well recognized for not being the quickest answer, yet it is adequate for comparison.The Keras frontend and Tensor Flow backend were utilized in a sequential implementation [2].

In the future, a GPU-accelerated parallel implementation based on TensorFlow [12] is planned. The high-level algorithm explanations for both techniques are included in this work. It also compares the execution times, memory needs, and CPU load requirements of both Python implementations.The goal of our study is to define the circumstances in which CNN has the cryptographic potentialproved and decideon a review of deep learning cryptography systems in the context of side-channeled attacks[16].

II.ENCRYPTION AND DECRYPTION ON CNN

CNN stands for "convolution neural network" and refers to a multi-layer feedforward neural network [4].3D convolution is used in symmetric CNN encryption. This method necessitates appropriately structuring data and creating the proper type of secret crypto key [5].

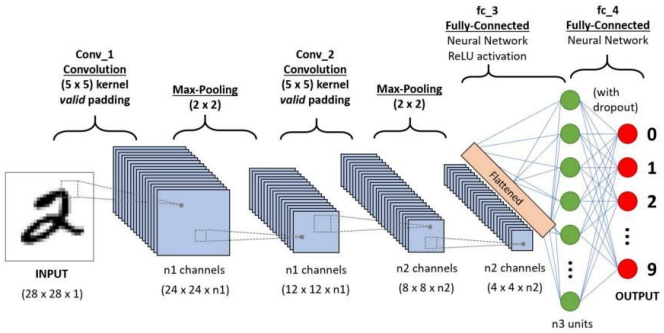

Figure 1: Encryption and Decryption on CNN

CNN is a Deep Learning system that really can take the image, attribute importance to different facets inside the image, and differentiate among them. In contrast to other classification techniques, the overall quantity which was before needed by the ConvNet is substantially less [6]. ConvNets can learn this filtering along with enough learning, whereas simple techniques require a touch of filters.The architecture of the Visual Cortex influenced the creation of a ConvNet, which is similar to the connecting network of Neurons in the Structure Of the brain [7]. Sensory cells can only react to changes in the Excel In the field, a tiny portion of the visual field [8]. To span the whole visual field, several comparable fields can be piled on top of one another as

shown in Figure 2.

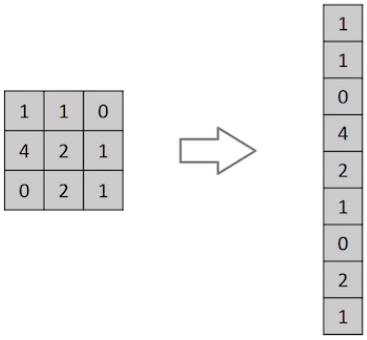

Figure 2: Mapping of comparable fields

In the case of exceedingly simple binary pictures, the approach may display an average precision score when doing class prediction, but it will have little to no accuracy when dealing with complicated images with pixel dependencies throughout [9-10].Through the use of appropriate filters, a ConvNet may successfully capture the Spatial and Temporal relationships in a picture shown in Figure3 [16]. Because of the reduced number of parameters involved and the reusability of weights, the architecture performs superior fitting to the picture dataset. In other words, the network may be trained to better recognize the image's complexity [11].

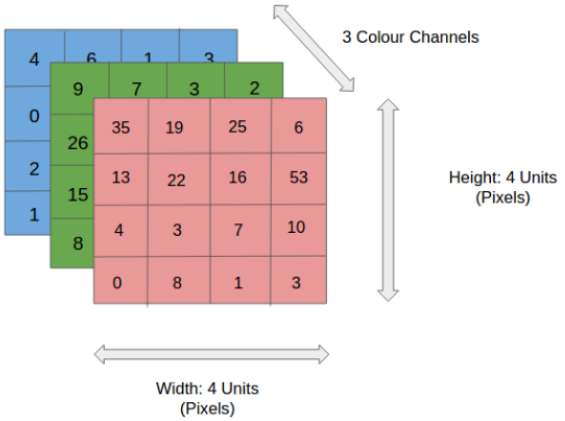

Figure 3: Spatial and Temporal relationships

An RGB picture has been divided into its three color planes — Red, Green, and Blue — as shown in the diagram. Images can be stored in a variety of color spaces, including Grayscale, RGB, HSV, CMYK, and others [12]. The ConvNet's job is to compress the pictures into a format that is easier to process while preserving elements that are important for obtaining a decent prediction [13]. This is critical for designing an architecture that is capable of learning features while also being scalable to large datasets.

III. VARIOUS LIBRARIES USED

ML has advanced at a breakneck pace in recent years. This is owing to the availability of ML/DL frameworks, which abstract away the enormous complexity of building and implementing ML/DL models.An ML library sometimes referred to as an ML framework, is a collection of procedures and algorithms built in a certain computer language. Fundamentally, they are platforms, frameworks, or tools that allow developers to construct machine learning models quickly and efficiently, without having to know the precise specifics of the underlying algorithms [14-15]. As a result, they assist developers in completing difficult tasks without rewriting a large number of words.

List of Packages/ Libraries in this research work

```
import numpy as np
import tensorflow as tf
import pandas as pd
import math
import matplotlib.pyplot as plt
import seaborn as sns
import socket
import sys
import pickle
```

IV. HYPERPARAMETERS USED IN THE EXPERIMENT

The factors that define the network topology (for example, the hidden layer cells) and the variables that control how well the network training are referred to as hyperparameters. The hyperparameters we are using to tune/experiment with the algorithmare shown in Figure 4

```
crypto_msg_len = N = 16
crypto_key_len = 16
batch_size = 512
epochs = 50
learning_rate = 0.0008 #Optimizer learning rate

# Function to generate n random messages and keys
def gen_data(n=batch_size, msg_len=crypto_msg_len, key_len=crypto_key_len)
:
    return (np.random.randint(0, 2, size=(n, msg_len))), (np.random.randin
t(0, 2, size=(n, key_len)))
```

Figure 4: Hyperparameters

V.NEURAL CRYPTOGRAPHY MODEL

The following is a description of how "secure communication" is implemented: The key is generated, the encryption/decryption process is used, and the message is sent (platform). The following is an example scenario: The aim is for Alice to send a message to Bob and Eve, who are attempting to hack into Alice and Bob's communication, to be unable to read it. Alice takes a message (plain text) and encrypts it into ciphertext using the key she shares with Bob. This encrypted text is transferred to Bob, who decrypts the ciphertext into the plain text after consuming the communication and key.Eve intercepts the cipher text's transmission and tries to decrypt the plaintext without knowing the keys or, in certain situations, the encryption technique.

We used the TensorFlow platform to develop the initial automated system and were forced to abandon Theano implementation due to a lack of CPU optimal control libraries on Windows 10; NCA has been implemented in Theano, TensorFlow, and frequent python frameworks by different authors, but the base method continues to remain the same. Docker was used to runningthe code on a Windows 10 machine with 16 GB RAM and an 8-core 3.1 GHz CPU with high energy.

We tested the Adam optimizer, Adagrad optimizer, and Momentum optimizer, all of which employ loss functions to minimize error. Following that, we created the UI for the NCA algorithm. The user can submit input in ASCII, which is subsequently transformed to binary and suitably organized so that it can be accepted by the np.array for both encryption/decryption and signing/verification operations. To communicate via a socket, the user (Alice) enters a plaintext message, which is subsequently encrypted (ciphertext) and transferred to Bob's socket. Bob's shared key is used to decode the message after receiving it, and Bob prints out the recovered message.

VI.NEURAL CRYPTOGRAPHY TESTING

Neural cryptography is a field of encryption that studies the use of probabilistic methods, particularly convolutional neural network processes, in cryptography and cryptology. The Neural Cryptography testing code is shown in Figure 5.

```
text = input('Enter some text: ')
bintext = ' '.join('{0:08b}'.format(ord(x), 'b') for x in text)
print (bintext)
b1  = bintext.replace(" ", "")
#b1 = b1.zfill(48)
pad = len(b1)%16

v1 = np.array([])
for i in range(0, len(b1)):
    v1 = np.append(v1, int(b1[i]))

#apply the padding
for i in range(0, pad):
    v1 = np.append(v1, int(0))
total_len = len(b1) + pad

plaintext_to_Alice = v1.reshape(int(total_len/16), 16)
print('plaintext_to_Alice = ', plaintext_to_Alice)
```

Figure 5: Neural Cryptography testing

```
Enter some text: pawan
01110000 01100001 01110111 01100001 01101110
plaintext_to_Alice=  [[0. 1. 1. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 1.]
 [0. 1. 1. 1. 0. 1. 1. 1. 0. 1. 1. 0. 0. 0. 0. 1.]
 [0. 1. 1. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

VII.RESULTS AND DISCUSSION

The following code is the plot helper function as shown in Figure6. The plots demonstrate the decryption errors of the three parties per epoch when training the CNN and illustrate whether Eve was able to converge with Alice and Bob.

```
%matplotlib inline

def plot_results():
    """
    Plot Lowest Decryption Errors achieved by Alice, Bob, and Eve per epoc
h
    """
    sns.set_style("darkgrid")
    plt.plot(alice_errors)
    plt.plot(bob_errors)
    plt.plot(eve_errors)
    plt.legend(['alice', 'bob', 'eve'])
    plt.xlabel('Epoch')
    plt.ylabel('Lowest Decryption error achieved')
    plt.show()
```

Figure 6: plot helper function

VIII. TENSOR FLOW SESSION

TensorFlow is a complete open-source programming language platform. It features a robust, adaptable lot of tools, modules, and local resources that enable academics to push the boundaries of ML and programmers to quickly construct and launch ML-powered products.The tensor flow codefor this researchis shown in Figure. 7.

```
import time
epochs = 10
#sess = tf.InteractiveSession()
sess = tf.Session()

#with tf.Session() as sess:
print('Starting Training Process... ')
start_time = time.time()
train(sess)
end_time = time.time()
print('Time taken for Training (seconds): ', end_time-start_time)
plot_results()

print('alice_errors_train = ', alice_errors)
print('bob_errors_train = ', bob_errors)
print('eve_errors_train = ', eve_errors)
```

Figure 7: Tensor flow code

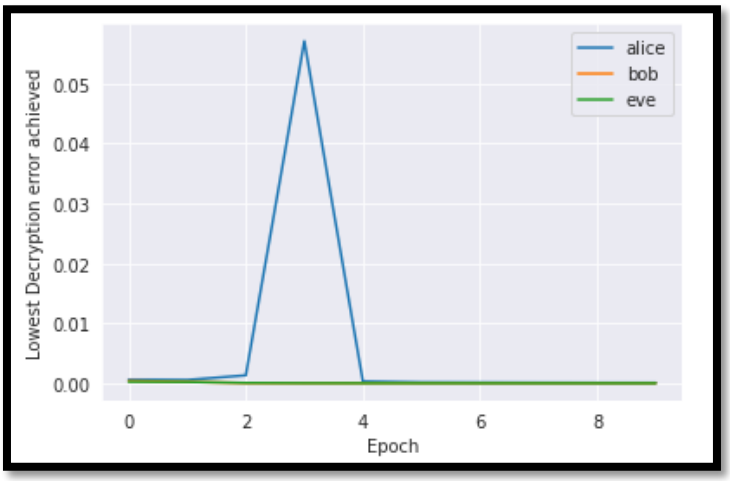

Figure 8: Graphical Representation of Error Description

Different error values obtained for Alice, Bob, and evefor 10 epochs are shown below it is observed that the error decreasesas we increased the number of epochs and it is observed that the error is reduced to almost zero at 100epochs.

```
alice_errors_train =[0.00060240366, 0.0005558743, 0.0013184354, 0.057097223,
0.00030981784, 0.00015021901, 0.00011890847, 9.263835e-05, 7.816318e-05, 7.1244736e-
05]

bob_errors_train=[0.00041602962, 0.00024216346, 2.1125983e-05, 1.8104402e-06,
8.3763734e-07, 3.3865217e-07, 1.838198e-07, 5.037873e-08, 3.799505e-08, 8.3164196e-09]
```

```
eve_errors_train=[0.00024993657,    0.00021492384,    4.8811577e-05,    1.0383843e-05,
3.0834665e-06, 8.9357025e-07, 3.0005702e-07, 1.2337105e-07, 3.0654814e-08, 9.844748e-
09]
```

IX. TESTING

Now for testing the experiment, we used the word "pawan" to be encrypted and sent and thevariousresults in terms of message,keys, and the process flow are demonstrated below which shows the completeflow of communication.

```
Testing Alice

messages =
 [[0. 1. 1. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 1.]
 [0. 1. 1. 1. 0. 1. 1. 1. 0. 1. 1. 0. 0. 0. 0. 1.]
 [0. 1. 1. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]

keys =
 [[1 0 1 0 0 1 1 0 1 0 1 1 0 1 0 1]
 [0 1 0 1 0 1 0 1 1 0 1 0 1 0 1 0]
 [0 1 1 0 0 0 1 0 1 0 0 0 1 1 1 1]]

plaintext_to_Alice=  [[0. 1. 1. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 1.]
 [0. 1. 1. 1. 0. 1. 1. 1. 0. 1. 1. 0. 0. 0. 0. 1.]
 [0. 1. 1. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]

testalice_output (***Cipher Text***) =

[[ 0.          0.99999183  0.99999547  0.99998266  0.          0.
0.00427923  0.          0.          0.99999887  0.9999972   0.
   0.          0.          0.          0.9999933 ]

[ 0.          0.99998915  0.99999565  0.99997175  0.          0.99997836
   0.9999949   0.99999636  0.          0.9999993   0.9999973   0.
   0.          0.          0.          0.99999106]

[ 0.          0.9999923   0.9999949   0.          0.9999871   0.9998396
0.99999726  0.          0.          0.          0.          0.
   0.          0.         -0.0101446   0.         ]]

SOCK_SEND

message =  [[ 0.          0.99999183  0.99999547  0.99998266  0.          0.
0.00427923  0.          0.          0.99999887  0.9999972   0.
   0.          0.          0.          0.9999933 ]
[ 0.          0.99998915  0.99999565  0.99997175  0.          0.99997836
   0.9999949   0.99999636  0.          0.9999993   0.9999973   0.
   0.          0.          0.          0.99999106]
[ 0.          0.9999923   0.9999949   0.          0.9999871   0.9998396
0.99999726  0.          0.          0.          0.          0.
   0.          0.         -0.0101446   0.         ]]

alice_errors_test =  0.0003070729

Testing Bob

messages =
 [[0. 1. 1. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 1.]
 [0. 1. 1. 1. 0. 1. 1. 1. 0. 1. 1. 0. 0. 0. 0. 1.]
 [0. 1. 1. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]

keys =
 [[0 1 0 0 0 1 0 1 0 1 1 0 1 0 1 1]
 [0 0 0 0 0 0 0 1 1 1 1 0 0 1 0 1]
 [1 0 1 1 1 1 1 0 1 1 1 0 0 0 0 0]]

SOCKET_RECV ====
```

```
waiting for a connection
connection from ('127.0.0.1', 49756)
received  [[ 0.          0.99999183  0.99999547  0.99998266  0.          0.
0.00427923  0.          0.          0.99999887  0.9999972   0.
   0.          0.          0.          0.9999933 ]
[ 0.          0.99998915  0.99999565  0.99997175  0.          0.99997836
   0.9999949   0.99999636  0.          0.9999993   0.9999973   0.
   0.          0.          0.          0.99999106]
[ 0.          0.9999923   0.9999949   0.          0.9999871   0.9998396
0.99999726  0.          0.          0.          0.          0.
   0.          0.         -0.0101446   0.        ]]
[[ 0.          0.99999183  0.99999547  0.99998266  0.          0.
0.00427923  0.          0.          0.99999887  0.9999972   0.
   0.          0.          0.          0.9999933 ]
[ 0.          0.99998915  0.99999565  0.99997175  0.          0.99997836
   0.9999949   0.99999636  0.          0.9999993   0.9999973   0.
   0.          0.          0.          0.99999106]
[ 0.          0.9999923   0.9999949   0.          0.9999871   0.9998396
0.99999726  0.          0.          0.          0.          0.
   0.          0.         -0.0101446   0.        ]]

testbob_input (***Cipher Text***) =

[[ 0.    1.    1.    1.    0.    0.    0.01  0.    0.    1.    1.    0.
   0.    0.    0.    1.    0.    1.    0.    0.    0.    1.    0.    1.
   0.    1.    1.    0.    1.    0.    1.    1.  ]

[ 0.    1.    1.    1.    0.    1.    1.    1.    0.    1.    1.    0.
   0.    0.    0.    1.    0.    0.    0.    0.    0.    0.    0.    1.
   1.    1.    1.    0.    0.    1.    0.    1.  ]

[ 0.    1.    1.    0.    1.    1.    1.    0.    0.    0.    0.    0.
   0.    0.   -0.06  0.    1.    0.    1.    1.    1.    1.    1.    0.
   1.    1.    1.    0.    0.    0.    0.    0.  ]] 2

testbob_output (***Plain Text***)=
 [[0.          1.          1.          1.          0.          0.
  0.          0.          0.          0.9999991   0.99999976 0.
  0.          0.          0.          0.9999998 ]
 [0.          1.          0.99999976 1.          0.          1.
  1.          1.          0.          0.99999774 0.99999976 0.
  0.          0.          0.          0.99999994]
 [0.          1.          1.          0.          1.          1.
  1.          0.          0.          0.          0.          0.
  0.          0.          0.          0.        ]] 2

bob_errors_test=  0.00030705179

Testing Eve

messages =
 [[0. 1. 1. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 1.]
 [0. 1. 1. 1. 0. 1. 1. 1. 0. 1. 1. 0. 0. 0. 0. 1.]
 [0. 1. 1. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]

keys =
 [[0 1 0 1 0 1 1 1 1 0 0 1 1 0 1 0]
 [1 1 1 1 1 0 1 0 1 0 1 1 1 0 1 0]
 [0 1 0 0 1 1 1 0 1 1 1 0 0 0 0 1]]

eve_errors_test=  1.3223396e-08

Time taken for Testing (seconds):  0.3379096984863281
```

```
bob_ouput_1 ==  [0. 1. 1. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 1. 0. 1. 1. 1. 0. 1. 1.
1.
 0. 1. 1. 0. 0. 0. 0. 1. 0. 1. 1. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]


b2 =  0b011100000110000101110111011000010110111000000000
pawan
Bob recovered Plain Text = Pawan
```

X. Performance Benchmarking with Cryptography.fernet

Fernet is a symmetric encryption implementation. It encrypts with AES and authenticates with SHA256, with initialization vectors generated via os.random (). In terms of time, the following code compares AES versus NCA.

```
Cryptography.fernet AES Encrypt/Decrypt sessions
-----------------------------------------------
cipher text =  b'gAAAAABhhR1D4kIkBdLQ24N-
81pTnjylPeT0LUN_z953oTIttnG5Vxgu6NFQy8WVF8KYs0YdwITmATlFxIK_Z0X1Z5UD74PuNyDC7xyCNgAD_j
eBc_HyyDduvEtJnFY-PfqkWx5RFtI1C6qUHwJnkBsZLD1kDjcN1A=='
plain_text =
b'\x00\x01\x01\x01\x00\x00\x00\x00\x00\x01\x01\x00\x00\x00\x00\x01\x00\x01\x01\x01\x00
\x01\x01\x01\x00\x01\x01\x00\x00\x00\x00\x01\x00\x01\x01\x00\x01\x01\x01\x00\x00\x00\x
00\x00\x00\x00\x00\x00'
*********Performance of NCA versus Cryptography.Fernet*******************
Time taken for Alice to Encrypt the Plain Text (seconds):  0.21621227264404297
Time taken for Bob to Decrypt the Cipher Text (seconds):  0.09221887588500977
Time taken for Fernet to Encrypt the Plain Text (seconds):  0.0007994174957275391
Time taken for Fernet to Decrypt the Cipher Text (seconds):  0.0005078315734863281
```

It is found that time taken to decrypt the message using CNN is 0.0005 which is very less and shows the efficiency of the algorithm.

XI. Conclusion

This research study proved that it is feasible to attain low error rates several times with neural networks. To show a real-world use of CNNs over a TCP/IP network, Alice and Bob's conversation was mimicked using sockets. NCA has several applications, including the use NCA for secure communication between autonomous IoT devices. Because of its dynamically self-configurable character and ability to perform efficiently in massively parallel processing systems, Artificial Intelligence is another option for NCA.

XII. Future Scope

The primary goal is to increase NCA consistency by more hyperparameter optimization. Future work will also involve implementing sockets using Internet Protocol rather than localhost. Other objectives include incorporating Eve and her socket into the crypto pipeline. Performance benchmarking tests may be run against AES, SHA-256, and other encryption algorithms to examine the NCA algorithm's strengths and shortcomings to develop it further.

XIII. Reference

1. Mehdy, M. M., Ng, P. Y., Shair, E. F., Saleh, N. I., & Gomes, C. (2017). Artificial neural networks in image processing for early detection of breast cancer. Computational and mathematical methods in medicine, 2017.
2. Osia, S. A., Shamsabadi, A. S., Taheri, A., Katevas, K., Sajadmanesh, S., Rabiee, H. R., &Haddadi, H. (2017). A hybrid deep learning architecture for privacy-preserving mobile analytics.arXiv preprint arXiv:1703.02952.
3. R. Joseph Manoj, M. D. AntoPraveena, K. Vijayakumar, An ACO–ANN-based feature selection algorithm for big data, springer-Cluster Computing, https://doi.org/10.1007/s10586-018-2550-z
4. Shen, Y., Han, T., Yang, Q., Yang, X., Wang, Y., Li, F., & Wen, H. (2018). CS-CNN: Enabling Robust and Efficient Convolutional Neural Networks Inference for Internet-of-ThingsApplications. IEEE Access, 6, 13439-13448.
5. Shifa, A., Afgan, M. S., Asghar, M. N., Fleury, M., Memon, I., Abdullah, S., & Rasheed, N.(2018). Joint Crypto-Stego scheme for enhanced image protection with nearest-centroidclustering. IEEE Access, 6, 16189-16206.
6. Telem, A. N. K., Segning, C. M., Kenne, G., &Fotsin, H. B. (2014). A simple and robust gray image encryption scheme using a chaotic logistic map and artificial neural network. Advances in Multimedia, 2014, 19.
7. Chen, W. H., Luo, S and Zheng, W. X.(2016). "Impulsive Synchronization of Reaction–Diffusion Neural Networks With Mixed Delays and Its Application to Image Encryption," inIEEE Transactions on Neural Networks and Learning Systems, vol. 27, no. 12, pp. 2696-2710
8. Dridi, M., Hajjaji, M. A., Bouallegue, B and Mtibaa, A.(2016). "Cryptography of medical images based on a combination between a chaotic and neural network," in IET Image Processing,vol. 10, no. 11, pp. 830-839
9. Fakhr, M. W.(2017). "A multi-key compressed sensing and machine learning privacy preserving computing scheme," 2017 5th International Symposium on Computational and Business Intelligence (CBI), Dubai, pp. 75-80.
10. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K., Naehrig, M., &Wernsing, J. (2016,June). Crypto News: Applying neural networks to encrypted data with high throughput andaccuracy. In International Conference on Machine Learning(pp. 201-210).
11. Hu, F., Wang, J., Xu, X., Pu, C., & Peng, T. (2017). Batch Image Encryption Using Generated Deep Features Based on Stacked Autoencoder Network. Mathematical Problems inEngineering, 2017.

12. Donghui Hu, Liang Wang, Wenjie Jiang, Shuli Zheng, Bin Li. (2018). A Novel Image Steganography Method via Deep Convolutional Generative Adversarial Networks, IEEE Transactions on Information Forensics and Security
13. Lakshmanan, S., Prakash, M., Lim, C. P., Rakkiyappan, R., Balasubramaniam, P., & Nahavandi, S. (2016). Synchronization of an inertial neural network with time-varying delaysand its application to secure communication. IEEE transactions on neural networks and learning systems, (99), 1-13.
14. Luo, J. S., & Lo, D. C. T. (2017, December). Binary malware image classification using machine learning with the local binary pattern. In Big Data (Big Data), 2017 IEEE InternationalConference on (pp. 4664-4667). IEEE.
15. Marohn, B., Wright, C. V., Feng, W. C., Rosulek, M., &Bobba, R. B. (2017, October). Approximate Thumbnail Preserving Encryption. In Proceedings of 2017 on MultimediaPrivacy and Security (pp. 33-43). ACM.
16. Kritika Purohit. Application of cryptography using artificial intelligence (2020). Journal of innovation in electronics and communication engineering.