ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

SOK: Practices employed in the neural cryptanalysis of ciphers

Master's Thesis

Norica Băcuieți

June 14, 2024

Advisors: Prof. Dr. Kenny Paterson, Prof. Dr. Lejla Batina, Dr. Stjepan Picek

Applied Cryptography Group
Institute of Information Security
Department of Computer Science, ETH Zurich

Abstract

After Gohr published his residual neural distinguisher at CRYPTO'19 [18], many works followed that tried to improve his results using various approaches. To better assess this progress made on neural distinguishers, we wrote this thesis, aiming to systemize the knowledge and practices presented for the cryptanalysis of ciphers using neural networks and to verify whether some of those practices can be generalized across datasets, ciphers, and neural networks.

Concretely, after we have studied the relevant papers on the topic of neural cryptanalysis and answered seven questions, the main takeaways are the following: 1) the more information a (strong) neural distinguisher is given to train on, the better its performance is; 2) a low-hamming-weight input difference leads to higher-performing neural distinguishers than one from a high-probability differential does; 3) the way one reshapes the ciphertexts matters for achieving a better-performing neural distinguisher and is dependent on the neural distinguisher's complexity; 4) the DBitNet neural network proposed in [9] appears to offer the best* overall performance on the studied ciphers; 5) the iterative training strategy [9] for improving a neural distinguisher's performance does not seem to help, while the freezing layer strategy [6], even if the improvement is minor, does, given that there is a clear separation between the feature extractor and the classifier of the neural distinguisher; 6) the bit profiling tool NNBits [20] could not identify patterns in the ciphertext pair bit sequence nor recover ResNets's [18] performance completely using (only) single-bit predictions for Simon or Katan despite being able to do that for Speck, meaning that ResNet bases its decision on more than the correctness of single bit predictions for the two afore-mentioned ciphers; 7) and, finally, a recently developed evolutionary algorithm [9] manages to obtain good, low-hamming-weight input differences that lead to a high-performing neural distinguisher, all while being fast and cipher-agnostic.

Contents

Chapter 1

Introduction

Introduction: In 2019, Gohr proposed the first neural network [18] with which he managed to distinguish round-reduced *real pairs*, i.e., "ciphertext pairs (C, C') resulted from encrypting plaintext pairs (P, P') where $P \oplus P' = \Delta_{in}$", from *random pairs*, i.e., "ciphertext pairs (C, C') resulted from encrypting plaintext pairs (P, P') where Δ_{in} is a random value", with a remarkable accuracy.

After he showed that "neural networks can be used to execute distinguishing attacks against a round-reduced block cipher competitively to the then-published state-of-the-art, many works followed that tried to improve over his results either by enhancing the dataset the neural network trains on or the architecture of the neural network itself" [18] [9]. In addition to these, authors also proposed generic strategies to improve the neural distinguisher's performance as well as methods to explain what the neural distinguisher is exploiting from the information provided (in the hopes of aiding cryptanalysts to find weaknesses they have not exploited so far). Since the number of publications that continued Gohr's work exploded, we felt that an overview of the state-of-the-art would be helpful for assessing the progress made on neural distinguishers, so we compiled the relevant information in this work.

Aim of the thesis: The aim of this thesis is to systemize the knowledge and practices presented for the cryptanalysis of ciphers using neural networks and to verify whether some of those practices can be generalized across datasets, ciphers, and neural networks. Concretely, we will focus on the following four topics: datasets, architectures, explainability, and evolutionary algorithms. For each of those, the state-of-the-art will be presented briefly, and where possible, relevant research questions will be stated, which will later be answered based on the results obtained from the conducted experiments.

Structure of the thesis: The thesis starts with an introduction, notions, a brief presentation of the studied ciphers, and the experimental setup. In Section 2, we will look at the datasets a neural distinguisher usually trains on, i.e., what information to include, how to shape it, and what input difference to use, and we will answer three questions. In Section 3, we will look at various neural networks that can be used as neural distinguishers, how they can be used to extend classical distinguishers, strategies to improve a neural distinguisher's performance, how to interpret the neural distinguisher's accuracy, and we will answer another three questions. In Section 4, we will look at ways to explain a neural distinguisher's behavior and answer our final question. In Section 5, we will have a brief look at how an evolutionary algorithm finds good input differences, and finally, we conclude our work with Section 6, by offering a brief overview of what we have seen and presenting recommendations as well as suggestions for future work.

1.1 Notions

In this subsection, we will provide common notations and some useful definitions.

Notations: In this thesis, "the bitwise eXclusive-OR operation will be denoted by $\oplus$, the bitwise AND operation by $\wedge$, modular addition modulo 2^n by $\boxplus$, and the **H**amming **W**eight (HW) of a bit string is given by the number of ones present in it" [3].

Differential cryptanalysis: "Differential cryptanalysis is concerned with the propagation of a difference through a cipher. Let a function $f : \mathbb{F}_2^b \to \mathbb{F}_2^b$ and x, x' be two different inputs for f, where $x \oplus x' = \Delta x$. Let $y = f(x)$, $y' = f(x')$, and $y \oplus y' = \Delta y$ [11]. Then, the transition probability Δx to Δy denoted as (Δ x $\xrightarrow{f}$ Δ y)" [11] is:

$$\mathbb{P}(\Delta x \xrightarrow{f} \Delta y) := \frac{\#\{x \mid f(x) \oplus f(x \oplus \Delta x) = \Delta y\}}{2^b} \text{ [11]}$$

Note that in this thesis, we will give the input difference using a (sequence of) hexadecimal numbers.

Neural distinguisher: A **n**eural **d**istinguisher (ND) is a classifier that, after training on a limited dataset containing samples of a certain type and learning their characteristics, will output a probability in [0,1] for each sample (of a separate test dataset) in the evaluation/testing phase. "If the probability is above 0.5, the sample is classified as real, i.e., coming from a specific, fixed input difference; otherwise, it is classified as random, i.e., coming from a random input difference" [18].

Classical/pure differential distinguisher: "A classical/pure differential distinguisher, such as the difference distribution table, lists the differential transition probabilities for each possible input/output difference pair $(\Delta x, \Delta y)$, where the studied function f is usually a sub-component of the attacked cipher" [11]. Based on those probabilities, after analyzing a certain amount of data, one can determine whether the oracle one interacted with is the cipher in question.

Different keys: In this thesis, we will use a different key per ciphertext so that "key dependencies will not (positively) influence the performance of the neural distinguishers during the experiments we will conduct" [11].

1.2 Ciphers

In this thesis, we will perform experiments on six ciphers. A brief description of each is provided below.

Speck and Simon: Speck and Simon [7] are "families of lightweight block cipher designed by the NSA, and in this thesis, we will concentrate on the variant using a 32-bit block size and a 64-bit key" [19]. "During encryption, the state is split into two equal-sized words, which are then processed by Feistel-like round functions" [19], and we will refer to this way of splitting a ciphertext when reshaping the ciphertexts for the subsequent experiments carried out in this thesis. "In addition, even without knowing the key, one can obtain part of the penultimate round. Concretely, for a ciphertext of the form (l_i, r_i), one can compute (l_i, r_{i-1}) for Speck, and (l_{i-1}, $r_{i-1} \oplus k_{i-1}$) for Simon" [19]. This additional information will be used to enhance the ciphertext pairs in subsequent experiments.

Skinny: Skinny [8] is "a tweakable block cipher designed by Beierle et al. In this thesis, we will concentrate on the variant using a block size as well as a tweak-key of 64 bits" [19]. "We will consider each of the 16 (4-bit) cells a word of the state and refer to this way of splitting a ciphertext when reshaping the ciphertexts for the subsequent experiments. Also, to be in line with the framework from [19], we will ignore the tweak and use the tweak-key as an ordinary key" [19]. In addition, even without knowing the key, one can obtain part of the penultimate round. "This includes reverting the linear part, i.e., MixColumns and ShiftRows, as well as the addition of the round constants and the application of the S-box to the last two rows" [19]. This additional information will be used to enhance the ciphertext pairs in subsequent experiments.

Present: Present [13] is "a cipher designed by Bogdanov et al. In this thesis, we will concentrate on the variant using a 64-bit block size and an 80-bit key. Similarly to Skinny, we will consider each of the 16 (4-bit) cells a word of the state and refer to this way of splitting a ciphertext when reshaping the ciphertexts for the subsequent experiments. In addition, the bit-permutation right before the last key addition is, by nature, a linear operation and can be reverted. However, since it is a simple permutation, we do not expect changing the order of the bits to make a big difference" [19], and we later confirmed this through experiments.

Katan: Katan [16] is a "block cipher designed by De Cannière et al. In this thesis, we will concentrate on the variant with a 32-bit block size and an 80-bit key. There is no splitting for this cipher, but, as for the other ciphers, the input will be reshaped when training on ciphertext pairs so that instead of training the neural distinguishers on a 32x2-bit string, we will train them on a matrix containing two separate 32-bit strings (that might be additionally split into words for other ciphers). As for the other ciphers, one can revert parts of the round function without knowing the key" [19]. For details on the information reverted, please refer to [19].

ChaCha: ChaCha [12] is "a stream cipher designed by Bernstein. In this thesis, we consider a keystream block as the ciphertext, and the nonce and counter as the plaintext [19]. ChaCha has a 256-bit key, and we will consider each of the 32-bit words of the 4×4 matrix when reshaping the ciphertexts for the subsequent experiments. In addition, when reverting the ciphertexts, we only consider reverting the modular addition of the two known rows of the initial state" [19]. For more details, please refer to [12].

1.3 Experimental setup

All neural networks were trained on 10^7 samples, validated on 10^6, and tested on 10^6 with a batch size of 5000 to remain consistent with previous works. The results for each experiment (except for the seventh one) are obtained by averaging accuracies across 10 trials, and the significance of the results was assessed using the Mann–Whitney U test. Since the variance of the results was on the order of 10^4 or 10^5 across experiments, we did not increase the number of trials due to the computational resources required as well as due to not expecting that more trials would change the outcomes. Note that all changes in the networks' performance (except for ChaCha-3R in question one and any other result that remained unchanged during experiments) are statistically significant even though they are minor. Also, results for ChaCha-3R with the ResIncep2 network trained on unshaped ciphertext pairs are not available due to the lack of significant computation resources required. Finally, the code, as well as the information on computational resources needed, is available here[1].

[1]https://github.com/NoricaBacuieti/SOK

Chapter 2

Insights on datasets

In this section, we will look at the datasets a neural distinguisher usually trains on. Concretely, we will see what information to include, how to shape it, what input difference to use, and we will answer three questions.

2.1 What dataset will you be training on?

In this subsection, we will see different distinguishing tasks from previous papers and how to transition from the first two tasks to the third one, what authors have trained their neural distinguishers on, and we will discuss how to compare performances.

2.1.1 What task are you trying to solve?

In the cryptographic community, researchers performing cryptanalysis using neural networks have tried to improve the accuracy and/or data complexity of the distinguishing attack on different tasks. Those tasks are:

- T1: Distinguish between real or random ciphertext pairs (with/without information from previous rounds).
- T2: Distinguish between "real or random ciphertext differences" [15].
- T3: Distinguish between real or random batches of ciphertext pairs or differences, etc.

Depending on the task one wants to solve, one will use a different type of data that might or may not be processed before passing it to the neural distinguisher. The types of data are:

- *Real or random ciphertext pairs:* A ciphertext pair is called *real* if it is obtained from encrypting two plaintexts between which there is a fixed (input) difference. Otherwise, if the (input) difference is random, the ciphertext pair is called *random* [18]. Please note that when information from the previous round is included in addition to the ciphertext pairs, we will refer to such enhanced ciphertext pairs as pairs+.

- *Real or random ciphertext differences:* "A real or random ciphertext difference is obtained by xor-ing the two ciphertexts of a real or random ciphertext pair" [18] [15].
- *Batches:* A batch is obtained by grouping data samples of one of the above-mentioned types into a single sample [11].

2.1.2 From T1 or T2 to T3

Interestingly, after training the neural distinguisher on T1 or T2, one can use it directly to predict on T3, where the batches contain the same type of data the neural distinguisher previously trained on. As mentioned in [19], "this is done by combining the probabilities returned by the neural distinguisher for each ciphertext pair or difference of a batch" [19]. Concretely, "for a batch of m samples where each is drawn from the same distribution (real or random), if we denote the probability that the i-th sample is a real sample as p_i and we assume that the probabilities $p_1, ..., p_m$ are independent, the probability that all m samples are drawn from the real distribution is $\frac{1}{1+\prod_{i=1}^{m}\frac{1-p_i}{p_i}}$" [19]. Using this formula, Gohr et al. [19] have empirically shown that a neural distinguisher trained on T1 or T2 can achieve the same performance as the same neural distinguisher trained directly on batches of the same data type. Moreover, the former is computationally faster, requires fewer computational resources, and is less prone to overfitting than the latter, where these are major concerns.

2.1.3 Training on differences vs. pairs vs. batches

Depending on the data type the neural distinguishers are training on, they are provided with less or more information. Naturally, as the amount of information increases, we expect the neural distinguisher's performance to increase as well.

Differences vs. pairs: In the "real difference experiment" presented in [18], Gohr concluded that his neural distinguisher ResNet learned more than the difference distribution after having trained on ciphertext pairs. This was later confirmed in Benamira et al.'s work [11] when training on ciphertext pairs instead of differences led to better performance. Although this was specifically tested on the Speck32/64 cipher using the same ResNet, this encouraged many of the authors that followed to train their own neural distinguishers directly on ciphertext pairs when performing experiments on different ciphers. Since many authors trained on ciphertext pairs directly without assessing the performance of ciphertext differences, the first question we would like to answer is:

> **Question 1:** Will training on ciphertext pairs generally lead to better performance than training on ciphertext differences, irrespective of the cipher and provided that a strong enough neural distinguisher is used?

We expect this normally to be the case; however, preliminary results on some toy ciphers in [19] seem to contradict this, with the authors of [19] expecting this behavior to translate to the actual (round reduced) ciphers. In addition, the same authors mention that this may also be due to not using a strong enough neural distinguisher. Seeing this, we will run experiments on multiple round-reduced ciphers, as well as use stronger neural distinguishers to verify their assumption and answer the question.

Pairs vs. pairs+ in batches: In [31], Zhang et al. trained an improved version of Gohr's distinguisher, ResIncep1 in this thesis, on ciphertext pairs enhanced by some information from the penultimate round. This information varied per cipher. To train their network on Speck32/64, they "construct batches containing the ciphertext pairs in addition to the right parts of the ciphertexts in the penultimate round, as they can be computed without knowledge of the penultimate subkey" [31]. Similarly, for Simon32/64, "besides the ciphertext pairs, a xor between the right parts of the ciphertexts in the penultimate round is included" [31]. The motivation for this was that, "by introducing additional data, the performance of the distinguisher may improve" [31].

In [29], Yue and Wu proposed a new data format that contained even more information from the penultimate round of Speck32/64 compared to Zhang et al.'s work [31]. The new data type now contained a piece of additional information that captured the difference of the left parts of the ciphertexts in the penultimate round but without the effect of the *nr - 1* -round subkey. With this new data type, they created batches and trained their neural distinguisher, ResIncep2 in this thesis, on those. Through experiments, they showed that with this new type of data, their network achieved better performance compared to using the data formats presented in previous works: Gohr [18], Benamira et al. [11], Hou et al. [22], Zhang et al. [31]. In addition, this type of data also yielded better results when using the neural networks presented by Gohr/Hou et al./Zhang et al. Therefore, in conclusion, Yue and Wu claimed that "the accuracy increases with the dimensionality of the input data as well as with the model complexity for Speck32/64, the claim remaining to be verified for other ciphers as well" [29].

Now, although additional data from the previous round was introduced in the context of a different task (T3) and neural distinguisher, seeing these positive results motivates us to use such additional data for other ciphers as well. Concretely, the second question we would like to answer is:

> **Question 2:** Will training on ciphertext pairs enhanced with information from the previous round generally lead to better performance than training on ciphertext pairs, irrespective of the cipher and provided that a strong enough neural distinguisher is used?

Batches: Compared to training on ciphertext pairs or differences, training on batches results in considerably higher accuracies. In [11], Benamira et al. trained Gohr's neural distinguisher on batches of preprocessed ciphertext pairs, achieving perfect accuracy after increasing the batch size. Similarly, in [22], Hou et al., like Lin et al. [25], train Gohr's depth-5 neural distinguisher on batches of k different ciphertext differences where all ciphertext differences in a batch come from the same input difference. The motivation in all of those papers was that by considering multiple pairs/differences at once, the neural distinguisher would also be able to learn the connections between the pairs/differences. In addition, through experiments, these authors also noticed that increasing the batch size increases the accuracy as well [11], [25]. Unfortunately, the high accuracy is either a result of overfitting, as explained in [31], [30] or a result of using considerably more data compared to T1 and T2. Furthermore, at least for some ciphers, [19] showed that training on batches of pairs does not result in better performance when compared to training on pairs and combining the results using the formula mentioned in the previous subsection, thus showing that no additional information was captured by training directly on batches.

2.1.4 Unfair comparison among performances

When presenting improvements, it is important to compare results within the same training task. In addition, when assessing the impact of additional data or dataset preprocessing method for a cipher, it is also important to train using the same neural distinguisher, input difference, and dataset size to ensure that any claims made about the performance improvement can be attributed to using that additional data/preprocessing method. Likewise, when assessing the impact of different neural distinguisher architectures, the dataset should remain constant to be able to fairly assess the performance improvement different architectures bring. Now, it may be that one type of neural distinguisher architecture constantly outperforms others across cipher and dataset types; it might also be that one architecture works best for some ciphers and worse for others. All of these insights we will obtain by performing experiments where we will change only one variable at a time, unlike many of the studied papers where the authors fail to take this into consideration.

2.1.5 Results and discussion

To answer questions one and two, we trained the ResNet neural distinguisher on ciphertext differences, (shaped) ciphertext pairs, and (shaped) ciphertext pairs enhanced with information from the previous round. In addition, after performing the experiments for questions 3 and 4, we recomputed Table 2.1 using the DBIT neural distinguisher, and the results can be seen in Table 2.2. Both neural distinguishers are described in Section 3.1, and the input differences used are given in Table 2.3.

Cipher	Differences	Pairs	Pairs +
Speck-6R	$0.752 \pm 4.11 \times 10^{-4}$	$0.781 \pm 1.18 \times 10^{-4}$	$0.781 \pm 3.97 \times 10^{-4}$
Simon-9R	$0.561 \pm 5.06 \times 10^{-4}$	$0.601 \pm 1.49 \times 10^{-4}$	$0.653 \pm 7.99 \times 10^{-4}$
Skinny-6R	$0.999 \pm 9.91 \times 10^{-5}$	$0.997 \pm 1.61 \times 10^{-4}$	$0.986 \pm 4.01 \times 10^{-4}$
Katan-50R	$0.537 \pm 4.05 \times 10^{-4}$	$0.621 \pm 2.84 \times 10^{-4}$	$0.697 \pm 5.40 \times 10^{-4}$
Present-6R	$0.707 \pm 5.41 \times 10^{-4}$	$0.697 \pm 2.73 \times 10^{-4}$	$0.697 \pm 2.67 \times 10^{-4}$
ChaCha-3R	$0.796 \pm 4.83 \times 10^{-4}$	$0.612 \pm 1.51 \times 10^{-1}$	$0.500 \pm 1.11 \times 10^{-5}$

Table 2.1: Accuracies obtained by training the ResNet neural distinguisher on ciphertext differences, ciphertext pairs, and ciphertext pairs enhanced with information from the previous round. The ciphertexts were reshaped to mimic the word structure of the cipher.

Answer 1: Comparing the results on ciphertext differences and those on ciphertext pairs from Table 2.1, the conclusions seem to be mixed. For Speck-6R, Simon-9R, and Katan-50R, we notice an improvement in the performance of the neural distinguisher, while for Skinny-6R, Present-6R, and ChaCha-3R, the performance decreases. Now, although the decrease in Skinny-6R and Present-6R are not large, they are still significant and suggest that the neural network did not manage to handle the additional information in a meaningful way. In the case of ChaCha-3R, an even more interesting phenomenon takes place: the accuracy fluctuates between 83% and 50%, meaning that ResNet manages to take advantage of the additional information sometimes, but other times, it gets completely confused. Nevertheless, after performing the experiments for questions 3 and 4 and recomputing Table 2.1 using the DBIT neural distinguisher, we conclude that the decrease in performance for Skinny-6R, Present-6R, and ChaCha-3R seen in Table 2.1 seems to be a weakness of ResNet. As seen in Table 2.2, if one uses a stronger neural distinguisher such as DBIT, the additional information present in ciphertext pairs is successfully exploited every time and leads to an improved performance.

Therefore, considering these results, we answer question one in the affirmative, mentioning that a strong enough neural distinguisher is needed to extract the additional information from the ciphertext pairs and use it appropriately.

Cipher	Differences	Pairs	Pairs +
Speck-6R	$0.753 \pm 1.37 \times 10^{-4}$	$0.783 \pm 2.06 \times 10^{-5}$	$0.785 \pm 7.00 \times 10^{-6}$
Simon-9R	$0.563 \pm 5.87 \times 10^{-4}$	$0.650 \pm 3.51 \times 10^{-5}$	$0.657 \pm 9.30 \times 10^{-5}$
Skinny-6R	$0.999 \pm 5.39 \times 10^{-5}$	$0.999 \pm 4.11 \times 10^{-5}$	$0.999 \pm 4.63 \times 10^{-5}$
Katan-50R	$0.537 \pm 6.80 \times 10^{-5}$	$0.685 \pm 1.58 \times 10^{-4}$	$0.741 \pm 7.86 \times 10^{-4}$
Present-6R	$0.709 \pm 1.26 \times 10^{-4}$	$0.711 \pm 8.31 \times 10^{-5}$	$0.711 \pm 9.44 \times 10^{-5}$
ChaCha-3R	$0.867 \pm 6.90 \times 10^{-4}$	$0.954 \pm 5.22 \times 10^{-5}$	$0.999 \pm 4.99 \times 10^{-6}$

Table 2.2: Accuracies obtained by training the DBIT neural distinguisher on ciphertext differences, ciphertext pairs, and ciphertext pairs enhanced with information from the previous round. The ciphertexts were reshaped to mimic the word structure of the cipher.

Answer 2: Comparing results on ciphertext pairs with results on ciphertext pairs enhanced with (additional) information from the previous round, we again notice mixed conclusions from Table 2.1. This time, for Speck-6R, ResNet did not manage to exploit the additional information from the previous round. However, it managed to maintain the performance it obtained on ciphertext pairs. Before recomputing Table 2.1 using the DBIT neural network, we suspected that this behavior might be due to ResNet already extracting the additional information from (only) the ciphertext pairs so that, when the additional information was given, it had no additional impact. However, after seeing the results obtained with DBIT in Table 2.2, even though the improvement is small when including information from the previous round, it is still significant and shows that the performance increases the more information DBIT is given. Continuing with Simon-9R and Katan-50, ResNet, as well as DBIT, enjoyed an improved performance when presented with information from the previous round. For Present-6R, the additional information had no effect on the performance of either neural distinguisher. This was expected since the information from the penultimate round would be just a permutation of the ciphertexts. For Skinny-6R and ChaCha-3R, the performance dropped in Table 2.1 as seen previously, again suggesting that ResNet struggled when presented with additional information. Nevertheless, when using DBIT, the performance stayed the same or improved, respectively, again showing DBIT's capability of extracting and using additional information appropriately.

Therefore, considering these results, we answer question two in the affirmative as well, again mentioning that a strong enough neural distinguisher is needed to extract the additional information from the enhanced ciphertext pairs and use it appropriately.

2.2 What input difference to use?

In this subsection, we will see why a good input difference matters, as well as how authors searched for one in previous works.

2.2.1 Why does a good input difference matter?

"During a differential attack, one aims to study the propagation of input differences and to exploit the nonrandom relations between the input and output differences.

In traditional differential cryptanalysis, the primary task is to find a high-probability differential characteristic that takes advantage of the unevenness of the differential distribution. Now, the distribution of output differences is different for different input differences, and a neural distinguisher actually learns, among other things, the distribution of the output difference obtained from a fixed input difference. Therefore, the choice of the input difference directly impacts the performance of the neural distinguisher. Knowing this, it is crucial to find one that improves the performance" [22].

2.2.2 How to choose input differences?

In 2019, Gohr [18] trained the first neural distinguisher on Speck32/64 ciphertext pairs that was successful at distinguishing real from random round-reduced ciphertext pairs. The input difference that he chose was the HW-1-difference Δ = 0x0040/0000, i.e., "the round-3 difference of the differential characteristic given in Table 7 of [2] that transitions to the low-HW difference 0x8000/8000" [3] [18]. However, since the author did not justify further why this particular difference seems to be offering better results, his work left some questions open for further research, such as whether a low-HW input difference generally leads to better performance compared to an input difference of a high-probability differential transition (irrespective of the cipher used).

Following the above-mentioned work, Benamira et al. [11] experimented with different input differences and trained Gohr's network on Speck32/64 ciphertext pairs constructed with an "input difference of a high-probability differential transition" [11] [22]. They found that "the performance decreased compared to before, which was surprising since it is generally natural for a cryptanalyst to maximize the differential probability when choosing a differential characteristic" [11]. Furthermore, the low-HW input difference seems to be a recurring theme in all the following papers, with authors constantly finding either by brute-force, SAT-based algorithms, or evolutionary algorithms that such type of input differences lead to the best performance.

It is well-known in classical differential cryptanalysis that the differences with low-HW diffuse relatively slowly, so Wang and Wang [27] use experiments to explore whether this theory can be applied to machine-learning-based cryptanalysis methods. They mention that "to observe the relationship between the accuracy of differential neural distinguishers and the HW of the input differences, the best way is to exhaust all the input differences with different HWs" [27]. However, since it required an enormous amount of calculation, they randomly chose only a few differences for every different HW. To speed things up further, they deployed few-shot learning and transfer learning [27] to train Gohr's neural distinguisher ResNet and found that the performance improves the lower the HW is.

In [22], Hou et al. "developed an algorithm based on SAT that took into consideration the unevenness of the output difference distribution for different input differences" [22]. Its purpose was to "help improve *nr*-round neural distinguishers by searching for high-probability *nr*-round differential characteristics and then using the input differences of those high-probability differential characteristics to train effective neural distinguishers" [22]. Interestingly, the SAT-based algorithm again found "the HW-1 input differences to lead to the best performance" [22]. In addition, they found that "the higher the HW of the input difference, the weaker the nonrandom feature of the ciphertext pair, suggesting that one should search first for input differences with lower HW using the SAT-based algorithm if time is limited" [22]. Seeing that again, a low-HW input difference led to better performance and, this time, on another cipher (Simon) as well, further solidified the idea that this might be a general recommendation.

In [21], Hou et al. used Gohr's depth-5 neural distinguisher ResNet to "explore the influence of the input difference pattern on its performance on ciphertext pairs" [21]. Among "all input differences with HW-1, they found that it matters whether the non-zero bit is located within the first half of the block size or the second half when it comes to the distinguisher achieving an above-random-guessing performance for Simon32/64" [21]. "This behavior seems to be mainly related to the round function of Simon32/64 as its non-linear operation is mainly concentrated in the first 16 bits" [21]. Concretely, "if the first 16 bits of the input difference are 0, the non-zero bits will spread less in the next round; otherwise, due to the non-linear operations, the non-zero bits will diffuse into the next round" [21]. This is in agreement with the findings from Hou et al. [22], where, again, "for Simon32/64, the input differences that led to the best performance were the ones where the non-zero bit was in the second half of the block size" [21]. Achieving less diffusion seems to be the key here, and strategically placing the non-zero bit of the HW-1 input difference helps achieve that. In addition, among those HW-1 input diffs with the bit in the right position, any HW-1 difference appeared to be equally as good. This behavior seems to be confirmed by [9] and appears to be a result of the rotational equivalence of differentials [24]. Nevertheless, the authors of [9] show that for other ciphers, such as Speck32/64, one input difference is significantly better than the others. Finally, Hou et al. also explored the impact on the distinguisher's accuracy caused by input differences (not necessarily of low-HW) that belong to some high probability differential characteristics [2] having *the same probability*, concluding that the neural distinguisher can extract more information using some but not other input differences. This behavior seems to be again consistent with Benamira et al.'s findings [11].

Cipher	Input difference
Speck	0x0040 0000
Simon	0x0000 00400
Skinny	0x0000 0000 0000 2000
Katan	0x4000
Present	0x0000 0000 0d00 0000
ChaCha	0x00000000 00000000 00000000 00008000

Table 2.3: The input differences obtained using Bellini et al.'s evolutionary algorithm and that were used for all the experiments in this thesis.

In [9], Bellini et al. compared their evolutionary algorithm with a brute-force search algorithm over low-HW input differences. They found "optimal input differences with a HW-3 for some ciphers and a HW-5 for others, mentioning that even if one were to list all input differences up to HW-3, no conclusion could be drawn about higher HW differences, i.e., HW-5" [9]. Furthermore, compared to brute-force search, their optimizer explored far fewer input differences while also being faster, and this scalability proved to be advantageous as the search space grew.

Since Bellini et al.'s optimizer is fast, considers a vast amount of input differences, and has been able to (re)produce good input differences for a variety of ciphers, we will use the input difference obtained by it when constructing the datasets for the experiments we will be conducting in this thesis. They can be seen in Table 2.3.

2.3 How to shape/preprocess your dataset?

In this subsection, we will have a look at how authors processed the ciphertexts before training.

2.3.1 Different ways of shaping/processing the ciphertexts

Mimic the word structure of the cipher: In [18], Gohr rearranged each Speck32/64 ciphertext pair into a 4 x 16 matrix before passing it to his neural distinguisher ResNet. This arrangement mirrored the word-oriented structure of the cipher and made the neural distinguisher aware of it. The same behavior can be seen in Belini and Rossi [10] for Raiden. This way of reshaping the input seems important, as some preliminary experiments conducted by us on Speck32/64 showed that performance decreased when reshaping was not performed. Based on those, the third question we would like to answer is:

> **Question 3:** Will reshaping the input to mirror the word structure of the cipher generally improve the performance of a neural distinguisher, irrespective of the cipher and neural distinguisher?

A complex preprocessing: In [11], Benamira et al. achieved an equivalent performance to [18] after processing the ciphertext pairs. For this, they first verified a few conjectures that are explained in detail in [11]. Two of them were:

- "The first convolution layer of Gohr's neural network transforms the ciphertext pair into (ΔL, ΔV, V_0, V_1) and a linear combination of those terms" [11].
- The neural distinguisher's black-box approach can be reproduced by "computing a distribution table on the transformed input (ΔL, ΔV, V_0, V_1) and then finding and applying several masks to it in order to compress the distribution table, thus obtaining the **m**asked **o**utput **d**istribution **t**able (M-ODT)" [11].

Based on these findings, the authors processed the ciphertext pairs in the following way before passing them to their non-neural distinguisher: "first, they transformed them to (ΔL, ΔV, V_0, V_1), then they computed a masked output distribution table (M-ODT) on such tuples and used it to convert the tuple to a vector where each value represents the probability stored in the M-ODT for each mask" [11]. They then passed these vectors to an LGBM model and achieved the same performance as Gohr [18]. However, this method has the disadvantage of creating a distinguisher biased towards positive samples. In addition, it is a complex process, and a neural distinguisher is still needed to extract relevant masks. So, due to these disadvantages and since the obtained performance did not improve, we will not explore this type of preprocessing further.

No reshaping/preprocessing: In [9], Bellini et al. used a different approach in their cipher-agnostic pipeline. Concretely, they used a new neural distinguisher called DBitNet (DBIT in this thesis) constructed on the idea of dilated convolutions, with which they could detect both long and short-range dependencies among bits without needing to reshape the input. We will see more about it in Section 3.1.

2.3.2 Results and discussion

To answer the third question, we trained four different neural distinguishers on ciphertext pairs that were either reshaped to mimic the word structure (**S**haped) of the cipher or were left as two concatenated strings (**U**nshaped).

As for the previous experiments, the same input differences from Table 2.3 were used.

Cipher	Shape	ResNet	MLP	ResIncep1	ResIncep2	DBIT
Speck-6R	S	0.781	0.604	0.785	0.779	x
	U	0.689	0.670	0.784	0.547	0.783
Simon-9R	S	0.601	0.530	0.522	0.577	x
	U	0.527	0.531	0.520	0.502	0.650
Skinny-6R	S	0.997	0.851	0.889	0.919	x
	U	0.970	0.968	0.840	0.890	0.999
Katan-50R	S	0.621	0.537	0.548	0.562	x
	U	0.543	0.537	0.543	0.518	0.685
Present-6R	S	0.697	0.608	0.696	0.704	x
	U	0.658	0.675	0.677	0.524	0.711
ChaCha-3R	S	0.612	0.500	0.500	0.856	x
	U	0.500	0.507	0.500	x	0.954

Table 2.4: Accuracies obtained by training different neural networks on ciphertext pairs that were either re**s**haped to mimic the word structure of the cipher (S) or kept as a single (joint) **u**nshaped string of data (U).

Answer 3: Comparing the results on the shaped and unshaped ciphertext pairs from Table 2.4 for each neural network, we noticed that for more complex neural networks such as Resnet, ResIncep1, and ResIncep2, reshaping the ciphertexts so that they mimic the word structure of the cipher is crucial in achieving better performance. Concretely, in all cases, we notice a significant increase in performance, even though it is minor.

For simpler networks such as an MLP, keeping the ciphertext pairs unshaped, i.e., simply concatenating them, seems to be the key to obtaining a performance boost. However, preliminary experiments showed that when the ciphertext pairs *are* reshaped, a simple MLP's performance drops to around 50% across all ciphers. Therefore, to improve the performance, we chose to implement the idea mentioned in [10], where the MLP would first look at each word of two ciphertexts individually before continuing with an MLP prediction head. As seen in Table 2.4, the performances are indeed better than 50%, but they are still below the ones where the input was left unshaped.

Based on these results, the answer to question three is that reshaping the input (ciphertext pairs) appears to improve the performance of a complex neural distinguisher irrespective of the cipher. However, for less complex neural distinguishers, keeping the input unshaped appears to be the key. Still, for a new neural distinguisher, it is advisable to try both to see which gives the best performance.

Chapter 3

Insights on neural architectures

In this section, we will offer an overview of different types/architectures of neural distinguishers from previous works, how these neural networks can be used to extend the number of rounds a classical distinguished can attack, general strategies for improving the performance of a neural distinguisher, how to interpret the accuracy of a neural distinguisher, and answer three more questions.

3.1 Neural distinguishers

In this subsection, we will look at the neural distinguishers presented in previous works. A brief description of each is provided below.

ResNet: In [18], Gohr constructed a "depth-10 (or 1) residual tower of 2-layer convolutional neural networks preceded by a single bit-sliced convolution and followed by a 2-layer densely connected prediction head" [18]. It managed to exploit information from the ciphertext pairs beyond the difference distribution as seen from their real difference experiment (see Section 4.1), and it did that despite not being given any cryptographic knowledge except the word structure of the cipher present in the reshaped input, thus making it a fairly generic approach.

"Another interesting finding is that it managed to correctly identify ciphertext pairs that could not have appeared in the real distribution. This happened even though their difference had a high probability of belonging to the real distribution, suggesting that the ciphertext pairs are not evenly distributed within their respective difference equivalence classes and that the neural distinguisher exploits a strictly more fine-grained partition of the output pairs" [18].

Gohr's network has been used in multiple works since its publication. In many works, it has been used as is, except for the tweak in the networks's depth: depth 10 in [11], [3], [4], depth 1 in [3], [14], [27], depth 5 in [22], [21]. However, in some papers, the authors tweaked it even further, as seen below.

Pruned ResNet: In [3], Băcuieți et al. pruned Gohr's neural distinguisher significantly after the results obtained from evaluating the Lottery Ticket Hypothesis [17] "confirmed that not only 90% of Gohr's depth-10 neural distinguisher can be pruned without losing (on average) performance, but the same is true for its smaller depth-1 version" [3]. Concretely, they found that "the residual connection was not needed and that several filters and neurons present zero or very little activation, so removing them would not (significantly) affect the performance" [3]. They confirmed this by training Gohr's pruned depth-1 neural distinguisher on ciphertext pairs like Gohr, with the performance "remaining within 1% of his results obtained with the depth-10 neural distinguisher" [3]. Nevertheless, since there is no improvement in the performance, we will refer to Gohr's depth-10 residual network ResNet for further experiments.

MOTD-pipeline: In [11], Benamira et al. constructed a machine-learning distinguisher (see Section 4) aiming to "preserve Gohr's neural distinguisher's accuracy" [11]. It managed to achieve that; however, due to the slight bias towards true samples caused by how the M-ODT table was computed, as well as because Gohr's neural distinguisher still needs to be used for computing the M-ODT, we will continue to stick to Gohr's original network for further experiments.

MLP: In [4] and [5], Baksi et al. developed two deep learning-based distinguishers that could accommodate multiple as well as just one input difference, and used them to distinguish batches of either ciphertext differences in model 1 or ciphertext pairs in model 2. The underlying deep-learning models that did the heavy lifting were either a 3-to-5-layer MLP or Gohr's neural distinguisher trained on individual ciphertext differences/pairs, dealing with batches in the evaluation phase (see Section 3.6). As an advantage, the authors "note that the models and the underlying neural distinguishers are simple, that the approach is generic, and, therefore, can be applied to any symmetric key primitive" [4] [5]. Building upon their work, Jain et al. made some adjustments to Baksi et al.'s first model, where they used a 2-layer MLP and chose the input difference more judiciously, noticing an improvement in the distinguisher's performance.

MLP+: In [10], Bellini and Rossi constructed two deep learning-based distinguishers for block ciphers with a large block and key size and trained them on ciphertext pairs. One was based on an MLP network called Time Distributed Distinguisher, while the other was based on a convolutional structure, and both of them were split into two parts: the first part dealt word-wise with the ciphertext pairs, and the second part was an MLP prediction head. The idea of splitting the network into two parts came from the fact that, for the ciphers they studied, each ciphertext is calculated separately as two different words (although not independently), so they wanted the network to see these words to emphasize the key features of each of them independently. In addition, they also "tried to use only the second part of the network, obtaining a worse performance" [10].

Seeing this, we will use the MLP neural network for training on unshaped input, i.e., ciphertext pairs, and the MLP+ for training on shaped input.

ResIncep1: In [31], Zhang et al., "inspired by the Inception block in GoogLeNet, changed the first block of Gohr's depth-5 neural distinguisher ResNet to a block that uses multiple parallel convolutional layers with varying kernel sizes to enable the network to capture multi-dimensional information from the ciphertext pairs" [31]. In addition, they also "expanded the convolutional kernels in the (following) residual blocks" [31], aiming to "enlarge their network's receptive field progressively" [31]. Finally, the prediction head was supplemented with a dropout layer to reduce overfitting, followed by an additional fully connected layer on top of ResNet's prediction head to strengthen the prediction head.

ResIncep2: In [29], Yue and Wu, inspired by Gohr [18] and Zhang et al. [31], replaced Gohr's depth-10 residual tower with a depth-5 residual tower where each residual block consisted of three layers of convolutional neural networks out of which, the middle layer was inspired by the Inception module, which consists of multiple parallel convolutional layers. The motivation behind this change is similar to that of Zhang et al. [31], namely to "capture more dimensional information from the ciphertext pairs and thus improve their distinguisher's performance" [29]. To confirm that "*the choice of neural distinguisher* affected the performance, they used the same type of bached data" [29] (see Section 2.1.3) to train different neural distinguishers: Gohr [18], Hou et al. [22], Zhang et al. [31], and theirs, concluding that their neural distinguisher obtained the best performance.

DBIT: In [9], Bellini et al. built a cipher-agnostic neural distinguisher called DBitNet that managed to obtain the same performance as ResNet on Speck32/64 and Simon32/64. This neural distinguisher was constructed on the idea of dilated convolutions, with which, the authors managed to detect both long as well as short-range dependencies among bits without needing to reshape the input. To offer a bit more background, by using a dilation rate of above one, one can investigate long-rage dependencies among bits, e.g., "between the first and second half of the block size" [9]. Then, "to also investigate the dependencies between neighboring bits, one would use a dilatation rate of 1" [9]. Now, by alternating the two, the authors of [9], managed to explore dependencies among various bits, which resulted in DBitNet obtaining a remarkable performance.

As they aimed to construct a cipher-agnostic architecture, the dilation rates were automatically adapted based on the block size of the cipher. Furthermore, compared to ResNet, they have also chosen a stronger prediction head to be able to handle the complexity of different ciphers.

Finally, they have also used the standard training procedure (simply training the network), unlike Gohr in [18], all while maintaining a comparable performance. On this note, unless otherwise specified, we will train all neural networks using the standard training procedure.

With all of these options for a neural distinguisher, the fourth question we would like to answer is:

> **Question 4:** Do some architectures work generally better for some ciphers than others? Is there an architecture that generally works well across ciphers?

3.2 Results and dicussion

To answer question four, we look again at Table 3.1 copied here again for convenience. As previously mentioned, the five neural distinguishers were trained on shaped/unshaped ciphertext pairs obtained from the input differences given in Table 2.3. In addition, hyperparameter tuning was performed for these networks using Keras-Tuner [1]; however, it did not manage to find better versions of the neural distinguishers in the relatively limited number of steps given (due to time and hardware constraints). Nevertheless, as seen in [19] for ResNet, the performance would still be minimally affected by tuning (1-2 percentage points) even if more time would be given to the tuning process, and we expect tuning variants of ResNet to gain a similar performance boost. Finally, the best results are written in bold, and for DBIT, since it is designed to train directly on unshaped input, results for the shaped input case are not provided.

Answer 4: When considering the shape of the ciphertext pairs, results are mixed across ciphers in Table 3.1: for shaped ciphertext pairs, one of the three complex neural networks ResNet, ResIncep1, ResIncep2 achieves the best performance, and for unshaped input, predominantly DBIT and once ResIncep1 achieve the best performance.

However, if we look at the results as a whole and consider DBIT's performance on unshaped ciphertext pairs, when we compare it to that of other neural distinguishers on shaped ciphertext pairs, except for Speck-6R where ResIncep1 has a slight advantage*, DBIT constantly scores the highest. Therefore, we can say that if one had to choose only one neural distinguisher, the DBIT neural network would perform the best* across ciphers.

[1] https://keras.io/keras_tuner/

Cipher	Shape	ResNet	MLP	ResIncep1	ResIncep2	DBIT
Speck-6R	*S*	0.781	0.604	**0.785**	0.779	*x*
	U	0.689	0.670	**0.784**	0.547	0.783
Simon-9R	*S*	**0.601**	0.530	0.522	0.577	**x**
	U	0.527	0.531	0.520	0.502	**0.650**
Skinny-6R	*S*	**0.997**	0.851	0.889	0.919	**x**
	U	0.970	0.968	0.840	0.890	**0.999**
Katan-50R	*S*	**0.621**	0.537	0.548	0.562	**x**
	U	0.543	0.537	0.543	0.518	**0.685**
Present-6R	*S*	0.697	0.608	0.696	**0.704**	**x**
	U	0.658	0.675	0.677	0.524	**0.711**
ChaCha-3R	*S*	0.612	0.500	0.500	**0.856**	**x**
	U	0.500	0.507	0.500	*x*	**0.954**

Table 3.1: Accuracies obtained by training different neural networks on ciphertext pairs that were either re**s**haped to mimic the word structure of the cipher (S) or kept as a single (joint) **u**nshaped string of data (U).

Based on these results, we answer question four (a and b) in the affirmative and return to questions one and two to redo the experiments using DBIT.

Indeed, for Speck-6R, DBIT performs slightly worse, but since the decrease is only 1-2 percentage points, we will still use it to redo experiments one and two. Results can be seen in Table 2.2, where the DBIT neural network manages to consistently outperform ResNet across all ciphers on all three training sets, except for a few cases where the performance is the same.

3.3 Extending classical distinguishers with neural distinguishers

The neural distinguishers from the previous subsection can be used to extend the number of rounds that a classical distinguisher can attack.

3.3.1 The CD $\rightarrow$ ML distinguisher:

In their work [28], Yadav and Kumar extended an r-round high probability pure differential distinguisher, i.e., CD distinguisher, with an s-round deep learning-based differential distinguisher, i.e., **m**achine **l**earning (ML) distinguisher, resulting in an extended distinguisher, i.e., CD $\rightarrow$ ML distinguisher. With it, they managed to distinguish batches of (r+s)-rounds of real or random ciphertext differences with high accuracy. The ML distinguisher trained on "real or random s-round ciphertext differences with a chosen input difference Δ_r from a high prob differential" [28] and was inspired by the depth-1 version of ResNet, where the convolutions were replaced by dense layers as suggested by Baksi et al. [4]. Then, to create the (r+s)-rounds CD $\rightarrow$ ML distinguisher, the authors first conducted a "preparation phase" [28] where they did the following:

- They chose a "high-probability characteristic $\Delta_0 \rightarrow \Delta_r$ [2]" [28].
- Then, they "trained their s-round ML distinguisher, as mentioned above, to get its validation accuracy α_s" [28].
- They "used α_s as a threshold for increasing the data complexity as much as needed until the plots of true positive and true negative curves of the s-round ML distinguisher trained on (r+s)-rounds of ciphertext differences obtained from input difference Δ_0 did not intersect anymore" [28].
- Finally, they "computed the separation cutoff C_T as the average of the closest points on the non-intersecting true positive and true negative curves and used it when deciding how many ciphertext differences to consider when deciding whether the one communicates with the oracle or cipher" [28].

With the preparation phase done, the (r+s)-round CD $\rightarrow$ ML distinguisher is ready to directly distinguish real *batches* of ciphertext differences from random ones without further training. Furthermore, the advantage of this deep-learning-based extension approach is that it can be applied to any classical differential distinguisher.

3.3.2 Greedy strategies

In [27], Wang and Wang, inspired by the work of Yadav and Kumar, proposed three greedy strategies:

- **M1:** "This strategy provides better R, i.e., r+s-round differential-ML distinguishers by considering all combinations of classical differential distinguishers and ML distinguishers, and picking *the best combination from multiple R-round distinguishers* instead of constructing only one distinguisher as Yadav and Kumar did" [27].
- **M2:** "This strategy first ensures that the used ML distinguisher is the best and then uses *this best ML distinguisher* to extend the classical differential distinguisher" [27].
- **M3:** In contrast to strategy M2, this strategy "constructs the differential-ML distinguisher by combining the *best classical distinguisher* and an ML distinguisher" [27].

The depth-1 version of ResNet was used as the underlying s-round ML distinguisher on ciphertext differences for all three strategies. For M2, to ensure that *the best neural distinguisher* is used, the authors chose an input difference that yielded the highest accuracy. As discussed in Section 2.2.2, this input difference turned out to have a HW of 1. Once such an input difference was chosen, a search for "the best r-round differential trail that had *the output difference* equal to *the best input difference* was conducted" [27], and the differential-ML distinguisher was constructed by combining the two.

"On the flip side, in M3, when the search for *the best classical distinguisher* is conducted first, one searches for the best r-round differential trail and takes *the output difference of the trail* as *the input difference* of the ML distinguisher" [27]. Now, there may be multiple differential trials with the best differential probability, but "only those whose *r-round output difference* has the minimum HW are considered" [27]. For all those differential trials that satisfy this condition, the authors again choose "the best one according to the performance of the ML differential distinguisher trained on these *r-round output differences*" [27]. Again, the differential-ML distinguisher is constructed by combining the two. Finally, the resulting R-round distinguishers are used, as in Yadav and Kumar's work, to distinguish real *batches* of ciphertext differences from random ones.

3.4 Strategies to improve performance

In previous works, authors have tried several strategies to improve the performance of a neural distinguisher. We list them below:

Transfer learning: In [18], Gohr showed that ResNet is also capable of performing transfer learning, i.e., "reusing knowledge obtained from training on one dataset to boost performance on another dataset" [1] [18]. However, in [3], Băcuieți et al. found that transfer learning improved the distinguishing accuracy only by 1% compared to using a simple training pipeline, thus not impacting the performance considerably.

Hyperparameter tuning: In [19], Gohr et al. mention that "even though neural cryptanalysis can be performed on different ciphers, it might be that ResNet is not the best-performing network for every cipher" [19]. Therefore, with the aim of adjusting ResNet so that it may become the best-performing network for a new cipher, the authors perform hyper-parameter tuning. In the process, they "tuned twenty-two hyper-parameters and found that eight of them significantly impacted ResNet's accuracy for Speck32/64 and Simon32/64" [19]. Now, since tuning twenty-two parameters is time-consuming, the authors suggest that "a hyper-parameter search on only the learning rate, L2 penalties, and filter sizes would be sufficient before training a new neural distinguisher on a new cipher" [19]. Therefore, before running our experiments for questions 3 and 4, we performed a small hyper-parameter search focusing on these hyper-parameters for each neural distinguisher and cipher. Unfortunately, due to the vast amount of experiments and computation resources needed, we had to limit the number of iterations of the Bayesian optimization process, which resulted in it not being able to find better hyperparameters than the (original) ones provided by the authors of the studied neural networks. Furthermore, when using the learning rates, L2 penalties, and filter sizes for ResNet as described in [19], compared to the ResNet presented in [18], the neural network obtained a performance improvement of only 1-2 percentage points. Therefore, although there was an improvement, it was not considerable.

Iterative training: In [9], Bellini et al. describe the iterative training strategy with which they "trained a neural distinguisher from R_s to round R_f" [9]. Concretely, "the same network that was used to train on R_s is retrained for round R_{s+1} until round R_f is reached" [9]. For example, one would first train ResNet on 5-round ciphertext differences/pairs/pairs+, obtaining a 5-round neural distinguisher. Then, this (trained) 5-round neural distinguisher is retrained on 6-round ciphertext differences/pairs/pairs+, obtaining a 6-round neural distinguisher, and so on, until the desired round is reached. This training strategy was combined with the AMSGrad algorithm as an optimizer to "avoid both a possible convergence issue of the Adam optimizer, as well as having to tune/adjust the learning rate per cipher" [9]. Interestingly, both the strategy and AMSGrad made "a significant difference in their network's performance on 8 rounds of Speck32/64, since, without any of them, the network was not able to perform better than random guessing" [9]. Seeing this, the fifth question we would like to answer is:

> **Question 5:** Will the iterative training strategy using AMSGrad generally help improve the performance of a neural distinguisher, irrespective of the cipher?

Freezing layers: Continuing with another strategy, Bao et al. [6] introduced the freezing layer strategy. They mention that "the inability to directly train an 8-round neural distinguisher for Speck32/64 likely stems from the diffusion associated with the input difference 0x0040/0000, which makes the 8-round features considerably challenging to learn directly from limited data" [6]. To overcome this, they suggest "a network fine-tuning strategy, referred to as the freezing layer strategy, to limit the solution space" [6]. To briefly review, "the ResNet neural distinguisher consisted of two parts: the convolutional layers (feature extractors) and fully connected layers (classifier)" [6]. "The authors argue that the feature extractor can be reused, and the classifiers are relatively similar in adjacent rounds. Therefore, to train an 8-round distinguisher for Speck32/64, they suggest simply loading a well-trained 7-round neural distinguisher and freezing all its convolutional layers, leaving only the parameters in fully connected layers to be updated. With this strategy, they obtained an 8-round distinguisher with accuracy identical to the ones in prior works [18]" [6], despite not using complex training procedures. Furthermore, as an advantage over the complex training approaches used in [18], Bao et al.'s approach "maintained the same hyperparameters and did not require more samples in the final stage" [6]. Moreover, "in comparison with the iterative training strategy, the freezing layer strategy only needed two training rounds instead of multiple rounds in a row" [6]. Finally, the freezing layer strategy also "sped up the training process due to the reduction of trainable parameters; therefore, the authors recommend using this strategy once the number of rounds is too high for the neural distinguisher to train directly" [6]. Given all these advantages, the sixth question we would like to answer is:

> **Question 6:** Will the freezing layer strategy generally help improve the performance of a neural distinguisher, irrespective of the cipher?

3.5 Results and discussion

To answer questions five and six, we have implemented the iterative training and the freezing layer strategies. For each strategy, we have trained our five neural distinguishers from round *nr - 1* to *nr*, where *nr* is the round number specified next to each cipher.

Answer 5: As seen in Table 3.2, the iterative training strategy did not seem to help improve the performance of the neural distinguishers, as the accuracies remained the same. However, the results are not surprising since Bao et al. mention in [6] that when they tried the iterative training strategy, they did not notice a performance improvement either. They go on further, saying that they believe that the (slight) improvement Bellini et al. obtained was due to the additional polishing step they performed at the end. Based on the results, we answer question five in the negative and move on to the results on the freezing layer strategy.

Answer 6: As seen in Table 3.3, the freezing layer strategy helped improve only the performance of the more complex neural distinguishers where the feature extractor and classifier were clearly defined. However, although significant, even for these networks, the improvement was only 1 percentage point. Again, this was not surprising since Bao et al. obtained a similar improvement in [6]. For the MLP network, where we had to choose where to split the dense layers, the freezing layer strategy did not help. We chose to freeze only the first two layers, taking into consideration the way Bellini and Rossi [10] saw the network be split, but, perhaps, freezing only the first two layers was not enough. Therefore, based on these results, the answer to question six is that the underlying neural distinguisher appears to require a clearly defined feature extractor and classifier for the freezing layer strategy to offer a (minor) performance boost because, without it, the performance remains unaffected.

Cipher	Shape	ResNet	MLP	ResIncep1	ResIncep2	DBIT
Speck-6R	*S*	0.781	0.604	0.785	0.779	*x*
	U	0.689	0.670	0.784	0.547	0.783
Simon-9R	*S*	0.601	0.530	0.522	0.577	*x*
	U	0.527	0.531	0.520	0.502	0.650
Skinny-6R	*S*	0.997	0.851	0.889	0.919	*x*
	U	0.970	0.968	0.840	0.890	0.999
Katan-50R	*S*	0.621	0.537	0.548	0.562	*x*
	U	0.543	0.537	0.543	0.518	0.685
Present-6R	*S*	0.697	0.608	0.696	0.704	*x*
	U	0.658	0.675	0.677	0.524	0.711
ChaCha-3R	*S*	0.612	0.500	0.500	0.856	*x*
	U	0.500	0.507	0.500	*x*	0.954

Table 3.2: Accuracies obtained by training different neural networks with the iterative training strategy on ciphertext pairs that were either re**s**haped to mimic the word structure of the cipher (S) or kept as a single (joint) **u**nshaped string of data (U).

Cipher	Shape	ResNet	MLP	ResIncep1	ResIncep2	DBIT
Speck-6R	*S*	0.782	0.604	0.786	0.780	*x*
	U	0.690	0.670	0.785	0.548	0.784
Simon-9R	*S*	0.602	0.530	0.523	0.578	*x*
	U	0.528	0.531	0.521	0.503	0.651
Skinny-6R	*S*	0.998	0.851	0.890	0.920	*x*
	U	0.971	0.968	0.841	0.891	0.999
Katan-50R	*S*	0.622	0.537	0.549	0.563	*x*
	U	0.544	0.537	0.544	0.519	0.686
Present-6R	*S*	0.698	0.608	0.697	0.705	*x*
	U	0.659	0.675	0.678	0.525	0.712
ChaCha-3R	*S*	0.613	0.500	0.500	0.857	*x*
	U	0.500	0.507	0.500	*x*	0.955

Table 3.3: Accuracies obtained by training different neural networks with the freezing layer strategy on ciphertext pairs that were either re**s**haped to mimic the word structure of the cipher (S) or kept as a single (joint) **u**nshaped string of data (U).

3.6 Different ways of evaluation

Among the papers mentioned in this thesis, the authors presented different ways to evaluate the performance of trained neural distinguishers. In other words, after the distinguishers are trained and ready to predict, it still needs to be decided how to interpret the probability they give per ciphertext difference/pair/pair+/batch.

For T1 and T2, this is straightforward. The neural distinguisher gives a probability per ciphertext difference/pair/pair+, and "if this probability is above 0.5, then the ciphertext difference/pair/pair+ is assigned as coming from the real distribution, else it is assigned as coming from the random distribution" [18]. This can be seen in [18], [11], [3], [15], [14], [21]. Note that, in this thesis, we used *this way* of interpreting the probabilities of the neural distinguishers we trained.

For T3, various evaluating techniques are possible. First, there is the same one as seen previously, where the neural distinguisher directly gives a probability per batch, and a class is assigned again depending on whether the probability is above or below 0.5. This can be seen in [11], [22], [14], [31], [29], [30]. Second, there is the averaging method, where "the neural distinguisher is used to provide a probability for each element of the batch, and then the median of the results is computed to obtain a probability for the batch as a whole" [11]. Again, a class is assigned depending on whether the probability is above or below 0.5, and this can be seen in [11]. However, a more appropriate way of transitioning from T1/T2 to T3 is described in Section 2.1.2. Third, there is the cutoff method, where, first, enough data samples (elements of the batch) are generated to match a certain data complexity, which are then passed to the neural distinguisher to obtain a probability for each of them. Then, "if the number of samples satisfying some condition is above a cutoff, the batch is classified as real or else as random" [10]. This can be seen in [28], [10], [27]. The fourth method is about the probability of whether the sequence of classes predicted during training and evaluation using the *same* order of input differences is above random guessing. If it is, the batch is classified as real; otherwise, it is classified as random. This can be seen in [4], [23], [5].

Chapter 4

Insights on explainability

In this section, we will look at previously used methods that tried to explain a neural distinguisher's behavior and will answer a final question.

4.1 Methods for tackling the explainability task

The real difference experiment: "The use of machine learning for black-box cryptanalysis remained mostly unexplored until Gohr published his article at CRYPTO'19 [18]" [11]. His residual neural network performed remarkably well, and experiments [18] [11] showed that, while the neural distinguisher is mostly differential in nature, it also exploited an additional unknown property to cryptanalysts, as seen from the real difference experiment. To offer some background, "the real differences experiment proposed in [18] is a distinguishing task in which one uses a **pre**-trained neural distinguisher on ciphertext pairs to distinguish real ciphertext pairs from blinded real ciphertext pairs, i.e. real ciphertext pairs where each of the ciphertexts were xor-ed with the same randomly generated value" [18]. Since "perfect knowledge of the differential distribution of a primitive under study does not in itself allow the neural distinguisher to do better than random guessing on this task, if the neural distinguisher does better than random guessing, then this means that it managed to learn something besides the difference distribution" [18].

That the neural distinguisher exploited more than the difference distribution was further confirmed by Benamira et al. [11] when they trained Gohr's neural distinguisher both on ciphertext pairs as well as differences and compared their performance with that of a pure differential distinguisher.

Truncated differentials and M-ODT pipeline: To find out what this additional information could be, Benamira et al. studied the classified sets of real or random ciphertext pairs and, by performing experiments A, B, C, D

[11], they found that "Gohr's neural distinguisher bases its decision not only on the ciphertext difference but also on part of the internal state difference, i.e., truncated differential, in the penultimate and antepenultimate round" [11]. To confirm these findings, they "constructed a pure differential distinguisher that matched Gohr's neural distinguisher's accuracy on top of also being faster" [11]. However, reaching the same conclusions for other ciphers is not easily achieved since a part of these experiments is custom-tailored per cipher.

In addition, they also constructed a "distinguisher that uses machine learning that almost matches Gohr's neural distinguisher's accuracy on top of offering full interpretability: globally by retrieving the decision tree, as well as locally by deducing the importance of each feature in a sample's classification" [11]. They achieved that by constructing a compressed, i.e., masked, difference distribution table (M-ODT) on preprocessed ciphertext pairs (see Section 2.3.1) and passing the probabilities from the M-ODT to the ensemble model LGBM [11]. In this way, "the features were no longer abstract as in Gohr's neural distinguisher: each feature is a probability of the ciphertext pair being real, knowing the mask and the preprocessed ciphertext pair" [11]. In addition, using the same approach now on ciphertext pairs generated with another cipher (Simon), they again "almost matched the accuracy of Gohr's neural distinguisher, thus showing that the approach can be generalized to other ciphers" [11]. The disadvantage of their distinguisher is that it is biased towards positive samples "due to how the M-ODT table was constructed" [11]. In addition, Gohr's neural distinguisher is "still needed for extracting the masks" [11]. However, "since it is used only during preparation, it does not affect the distinguisher's interpretability" [11]. Nevertheless, Benamira et al.'s machine learning distinguisher might not generalize to other neural distinguishers as one again needs to find out what the new neural distinguisher is looking at/learning, which might differ from Gohr's neural distinguisher.

EDLCT: In [15], Chen and Yu constructed the **E**xtended **D**iffererntial-**L**inear **C**onnectivity **T**able (EDLCT), with which they "managed to uncover what features of a ciphertext pair are learned by a neural distinguisher, explain the findings of previous works deploying neural distinguishers, and to construct neural or non-neural (ML) networks equivalent to previous neural distinguishers" [15]: Benamira et al. [11], Gohr [18]. The EDLCT is "a generic tool for describing a cipher by providing a class of features that can bridge the gap between supervised learning and the neural distinguishing attack" [15]. From the machine-learning perspective, "each EDLCT entry is a middle-level feature [15] that describes the cipher and can be either trivial, i.e., follows the uniform distribution, or non-trivial, i.e., that can be generated from two trivial features and might follow a non-uniform distribution" [15]. Then, using those features, the authors build distinguishers using either a neural or non-neural network. For constructing a non-neural-distinguisher, "EDLCT features (pair of masks) that describe a ciphertext pair are constructed, and then, based on these features, a non-neural, explainable network (Logistic regression) is trained" [11] that matches the performance of previously proposed networks. Now, selecting those features is difficult; therefore, the authors also propose a simpler method using neural networks. Concretely, this time, "more features are given to the neural network to train on, again matching the accuracy of previous models" [15]. With both of these approaches, the authors showed that "the previous neural distinguishers indeed learned these features (pair of masks) of the EDCLT, noting that the unique nature of the cipher determines those features that have a greater influence on the neural distinguisher" [15]. In addition, "if more related features are provided, the machine learning-based distinguisher is more likely to achieve better performance" [15]. Then, to further explain the findings of the previous works, the authors have started by proposing a "**F**eature **S**et **S**ensitivity **T**est (FSST)" [15] to "identify which features corresponding to the EDLCT significantly influence the accuracy of neural distinguishers" [15]. They "explored how the nature of a cipher affects the influence of those features, finding that the features that have a significant influence share the same characteristic related to the round function. Now, although the EDLCT features are not directly related to the final prediction, they show that it is still possible to build good equivalent models using them" [15]. With this approach, they obtained the following insights on the following previous works:

- The phenomenon found by Gohr in [18] where "his neural distinguisher could distinguish real from masked real ciphertext pairs without retraining can be explained by EDLCT. It seems that the real ciphertext pairs provide two kinds of EDLCT features, while the masked real ciphertext pairs provide only one of those. Thus, as long as the neural distinguisher exploited the other feature not provided in the masked ciphertext pairs, the phenomenon would occur" [15].

- The phenomenon found by Benamira et al. in [11], where "Gohr's neural distinguisher relies strongly on the differences at round *nr - 1* and *nr* - 2 when making predictions, can also be explained using EDLCT. As the authors of [15] presented, if a transition from the input difference to the feature, i.e., $\Delta \rightarrow$ *feature*, is satisfied with a high correlation, the feature has a significant influence on the neural distinguisher. Now, if we split the encryption into two parts $\Delta \rightarrow \Delta_{nr-1}$ and $\Delta_{nr-1} \rightarrow$ *feature*, then since in the second part only one round is covered and the correlation is likely close to 1, the overall correlation will depend on the correlation of $\Delta \rightarrow \Delta_{nr-1}$ and so the neural distinguisher will rely strongly on Δ_{nr-1}. This applies similarly to *nr* - 2" [15].

NNBits: In [20], Hambitzer et al. try to "explain the behavior of Gohr's neural distinguisher by creating bit profiles using ensemble learning" [20]. The deep-learning-based ensemble "constructed from N neural networks, called NNBits, is a distinguisher that can distinguish between a collection of X-bit sequences of length n that come from a random distribution or a cipher. Using Gohr's neural distinguisher and extended versions of it, all of which were diversified on the data level, the ensemble is trained to predict a certain subset of the n bits given the remaining bits as input, where each neural network predicts a different subset of the n-bit sequence" [20]. In addition, by "providing the bit-level granularity in the prediction, NNBits offers information on which bits are weak bits, as well as dependencies between difference bits" [20]. "With the bit-level information provided by it, they explain almost all of Gohr's depth-1 neural distinguisher's accuracy. They do that by first training one neural network per bit, i.e., 64 in total, one for each bit of the ciphertext pair using NNBits, and then constructing a distinguisher using these single-bit predictors. Concretely, NNBits revealed a pattern in the first 32 bits that repeated in the next 32 bits as well: out of the first 32 bits, ten bits 0, 1, 2, 9, 11, 16, 17, 18, 24, 25 could not be predicted by Gohr's network, while the highest test accuracy is achieved on bit 4" [20]. Then, after constructing a "distinguisher from the 64 predictors and achieving almost the same accuracy as Gohr's depth-1 neural distinguisher, the authors concluded the majority of the accuracy of Gohr's network can be understood in terms of the correctness of single-bit predictions. In other words, this means that one way Gohr's neural distinguisher's behavior can be understood is by learning the underlying Boolean functions to predict single bits and then evaluating the number of correct predictions" [20]. Since this approach "can be generally applied across ciphers and neural distinguishers as long as the neural distinguisher is strong enough" [20], we would like to answer the seventh and final question:

> **Question 7:** Will the accuracy of a (strong) neural distinguisher (ResNet depth-1) "be understood in terms of the correctness of single-bit predictions" [20] for other ciphers as well?

LIME: In [3], Băcuieți et al. used "one of the state-of-the-art explanation techniques called **L**ocal **I**nterpretable **M**odel-agnostic **E**xplanations (LIME) [26], to explain the behavior of Gohr's neural distinguisher that was trained on ciphertext pairs" [3]. However, since LIME suggested that "none of the input bits of the ciphertext pairs are useful for the classification - even when the neural distinguisher achieved above 90% accuracy -, the authors suspect that LIME's *linear* explanation model might be the cause, explaining that the linear model will not be able to explain the distinguishers' behavior since there might be no linear boundary to begin with" [3]. Furthermore, this result "seems to be in line" [3] with both Gohr [18] and Benamira et al. [11] findings. To review, from these authors, we learned that "the neural distinguisher bases its decision on the difference distribution of the ciphertext pairs as well as on that of the penultimate and antepenultimate rounds, and those difference distributions are not linear" [3], i.e., they are computed using the XOR operation, which cannot be successfully classified using a linear classifier, thus leaving avenues for future research using a stronger explanation technique.

4.2 Results and discussion

In [20], Hambitzer et al. deployed their bit profiling tool NNBits on 6 rounds of Speck32/64 using the depth-1 version of ResNet as the underlying distinguisher, and the results were promising. First, NNBits "revealed a pattern in the first 32 bits that repeated in the next 32 bits" [20], which can be seen in Fig. 7 of [20]. Second, using it, the authors concluded that "the largest part of the accuracy of Gohr's network can be understood in terms of the correctness of single-bit predictions" [20]. Naturally, we wanted to know whether both of these findings present themselves when switching the cipher. With this aim, we re-run their tool on Simon-9R and Katan-50R since both of these required the same computational resources as Speck-6R. The other ciphers' ciphertext pair bit size required considerably more computational resources, so we will leave them to be explored in future research. Again, the same input differences from Table 2.3 were used.

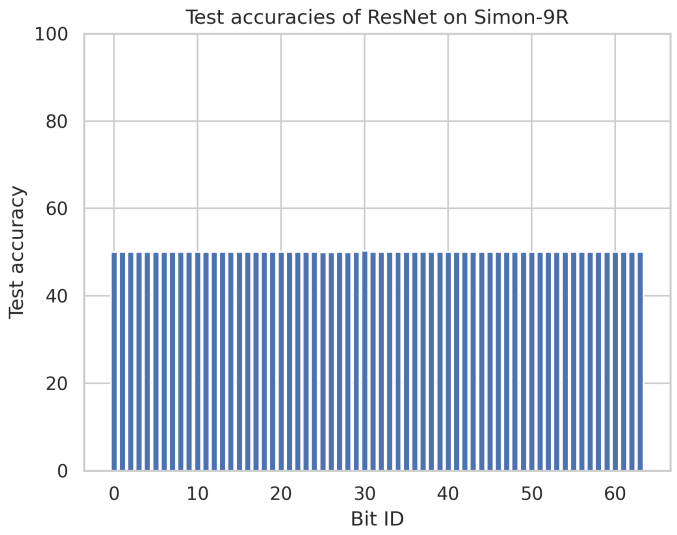

Figure 4.1: Test accuracies of ResNet on Simon-9R

Since for Speck-6R, Hambitzer et al. already presented results, we directly started with the next cipher, Simon-9R, and obtained Fig. 4.1. The results were quite different from what Hambitzer et al. obtained. While for Speck-6R there were a significant amount of weak bits, i.e., with a high test accuracy, and a pattern could be seen, for Simon-9R, NNBits did not manage to predict any bits with accuracy better than 50%. Furthermore, while for Speck-6R, the neural distinguisher's accuracy could be recovered, for Simon-9R, this was not the case. Instead of an accuracy of 60.1%, i.e., the one obtained with Gohr's depth-1 neural distinguisher, the distinguisher (MLP) using the previously obtained single-bit predictions could reach an accuracy of only 51.76%.

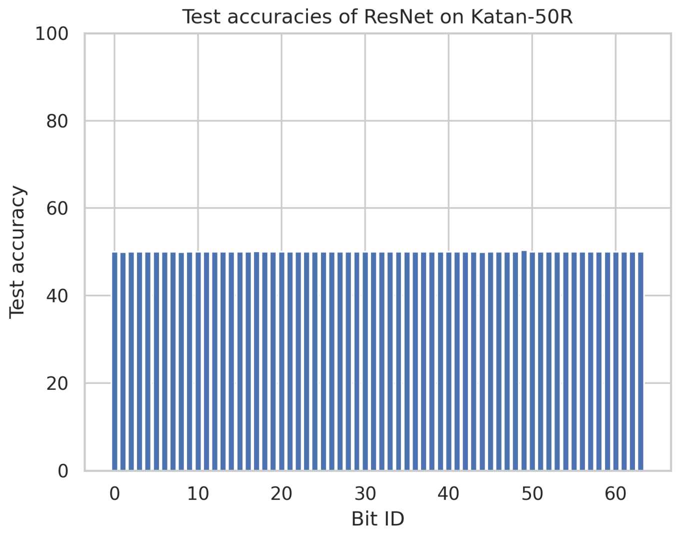

Figure 4.2: Test accuracies of ResNet on Katan-50R

The situation is similar for Katan-50R in Fig. 4.2. Again, a pattern could not be seen, nor did NNBits manage to predict any of the bits with accuracy better than 50%. Also, instead of an accuracy of 62.1%, the distinguisher using the previously obtained single-bit predictions could reach an accuracy of only 56.32%. Now, although the accuracy was not the same, it was higher than for Simon-9R and below that for Speck-6R. Seeing this, we suspected that Gohr's depth-1 neural distinguisher needs to reach a (base) accuracy of around 80%, i.e., like for Speck-6R, for the MLP distinguisher (using single-bit predictions) to recover it.

To test this, we wanted to see the results on Speck-7R. As seen in Fig. 4.3, similarly to Speck-6R, a pattern could be seen despite the tool struggling significantly more with predicting individual bits. Interestingly, although the tool struggled almost as much as for Simon-9R and Katan-50R, the MLP distinguisher using single-bit predictions managed this time to recover the same accuracy as Gohr's neural distinguisher, i.e., 61.6%. Seeing this,

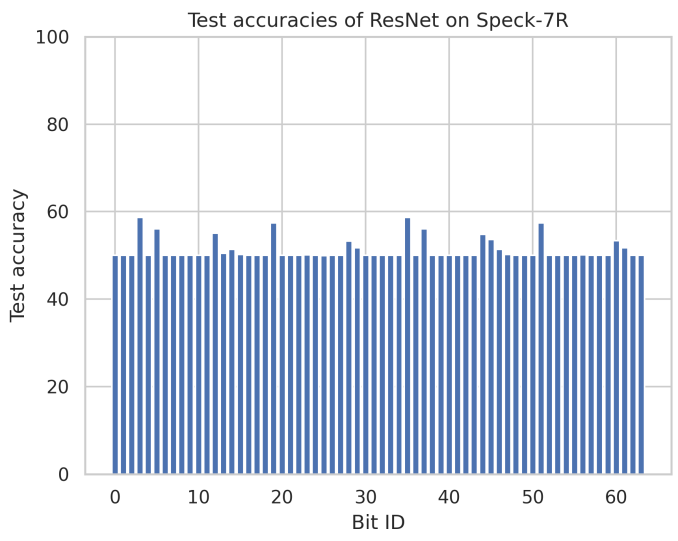

Figure 4.3: Test accuracies of ResNet on Speck-7R

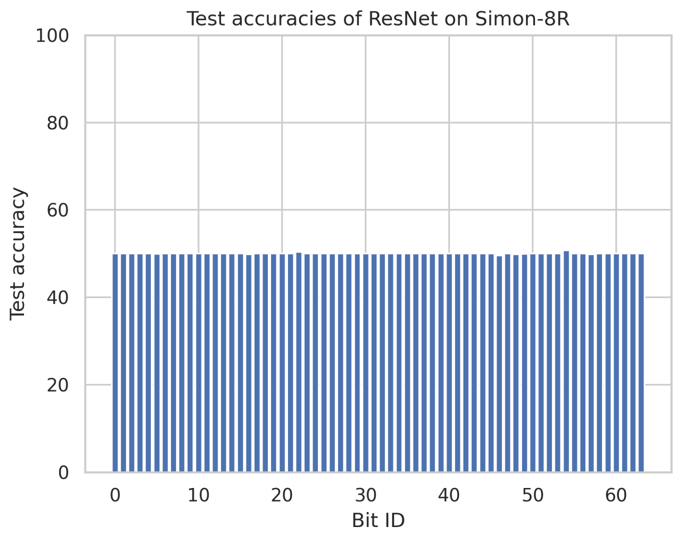

Figure 4.4: Test accuracies of ResNet on Simon-8R

our suspicions regarding a possibly required (minimum) performance for Gohr's neural distinguisher were not confirmed. It appears that it has to do rather with the cipher.

To check whether this is the case, we generated results for Simon-8R as well, and they can be seen in Fig. 4.4. This time, the depth-1 ResNet obtains an accuracy of 71.3%, i.e., higher than that for Speck-7R. Nevertheless, the findings are similar to the previous ones: no pattern and NNBits cannot predict bits with an accuracy above 50%. Furthermore, the accuracy recovered by the MLP distinguisher was 56.32%, so again, Gohr's neural distinguisher's accuracy was not reached, confirming that the results are cipher-dependent. Still, the MLP network performed better than random guessing for both Simon-8/9R and Katan-50R, which means that at least, for a small part, Gohr's neural distinguisher's accuracy included the correctness of single-bit predictions. Seeing this, we answer question seven in the affirmative, mentioning that only a small part of the accuracy could be explained by the correctness of single-bit predictions for those other ciphers.

Chapter 5

Insights on evolutionary algorithms

In this section, we will look at a recently proposed cipher-agnostic evolutionary algorithm by Bellini et al. [9] that "manages to find good input differences" [9].

In [11], Benamira et al. showed that choosing an input difference of a high-probability differential transition does not always lead to the best-performing neural distinguisher - at least not for Speck32/64. Going back to [18], Gohr proposed a search algorithm for good input differences; however, it presented two disadvantages. The first disadvantage was that it was difficult to adapt it to other ciphers, and the second one was that it was not able to find the optimal input differences for ciphers having a large block size. In addition, the algorithm's evaluation speed per difference was quite significant, meaning that it would be difficult to explore a large number of differences if needed. All these downsides motivated Bellini et al. to propose an evolutionary algorithm that would address these issues when searching for a good input difference. This algorithm consists of two parts: "a bias score that allows ranking many input differences in a very short time, as well as an evolutionary algorithm that efficiently uses this bias score" [9].

Bias score for ranking input differences: The authors observed that "the input difference 0x0040/0000 fixed some bits after a few rounds, in addition to seeing that high biases persisted in higher rounds" [9]. Seeing this, they conjectured that "high biases in individual bits are a good approximation for the existence of high probability truncated differentials of the penultimate/antepenultimate round" [9], which are "otherwise difficult to find generically" [9]. If they are correct, then "highly biased difference bits at round r should lead to good neural distinguishers at round r + θ through differential-linear properties" [9]. Therefore, they considered "an input difference to be good" if it led to "high biases in the difference bits of the higher rounds" [9]. They verified this assumption via experiments and found that "the neural distinguisher they use usually manages to distinguish several rounds past the highest round where a bias was detected" [9].

Having decided that "a good input difference for neural distinguishers is one that maximizes such bias score" [9], actually computing this bias score would not be possible for actual cipher since "it requires enumerating all keys and plaintexts" [9]. Thus, they use an approximated bias score that they compute from "a limited number of samples" [9]. Using this approximated bias score, they found previous, as well as new low-HW input differences that yielded the best/better results than input differences found by Gohr's neural difference search [9].

Since this approximated bias score was "fast and found input differences that led to high-performing neural distinguishers" [9], the authors were motivated to develop "an evolutionary-based search algorithm using this approximated bias score in order to be able to explore more candidate input differences" [9].

Evolutionary algorithm "The algorithm starts from a random population of input differences and improves this population iteratively by deriving new input differences from existing ones. Those are ranked using their (approximate) bias score, and, based on this information, only the input differences with the best scores are allowed to proceed to the next generation. The algorithm continues until no input difference has a score higher than a given threshold T_b" [9], at which point it stops.

Now, since it "matters at which round one evaluates the bias, and since the most relevant round is not known in advance, they run the optimizer for each round from 1 to R (maximum number of rounds for a given cipher), thus obtaining R lists of X input differences each" [9]. Arguing that "due to the heuristic nature of the optimizer, some good differences may have been identified in a subset of the rounds only" [9]; they re-score all X input differences "on the union of these lists"[9], obtaining "R bias scores for each of the X input differences" [9]. With these R bias scores per input difference, the authors had to decide how to compute an overall score per input difference. They had two concerns that they wanted to address. The first concern was that "the score in the highest round may not be a good indicator of the performance of a neural distinguisher" [9], and the second concern was that "an (unweighted) sum of the scores at each round may favor less interesting differences" [9]. Therefore, to address both these concerns, they used a weighted score, where higher rounds would be attributed a higher weight. The authors recognize that "this choice is not optimal and that better ways to score input differences may exist" [9], but, since it finds good input differences in practice, they are satisfied with it [9].

Now, if one would run this optimizer as described above, it would return many input differences. Evidently, most of them lead to high-performing neural distinguishers; however, finding the best one "is difficult since it would require fully training the neural distinguisher for each of them, which is time-consuming" [9]. Interestingly, for some ciphers, one input difference clearly dominates the others and leads to the best-performing neural distinguisher. However, for other ciphers, the algorithm outputs multiple input differences that obtain almost identical scores - probably due to "the rotational equivalence of differentials" mentioned in [24]. To reduce the number of returned input differences, however, in the latter case, the authors introduce the notion of "the distance to the highest score as a metric to choose which differences to investigate further with the neural distinguisher" [9]. Concretely, they define "an input difference as ϵ-close to another if their score is within ϵ of each other" [9]. With this metric, we are left with a significantly shorter list of input differences from which we can choose any to train our neural distinguisher.

As this "scalable input difference-finding algorithm" was already successfully used on the various ciphers studied in this thesis, thus showing its "generic approach" [9], we have directly used the input differences it found when performing the experiments above, and they can be seen in table T2.3.

Chapter 6

Conclusions, recommendations, and future work

Conclusions: In this thesis, we have had a look at the recent practices employed in neural cryptanalysis and verified whether some of them can be generally applied across datasets, ciphers, and neural networks.

We started by looking at what datasets one could train on. Here, we saw that one could train on ciphertext differences, ciphertext pairs with or without including information from previous rounds, as well as on batches of the former three. First, we wanted to know whether allowing a neural distinguisher to train on more information would generally lead to better performance irrespective of the cipher, and we found out that if the neural distinguisher is strong enough, the more information it is given, the better the performance is. Then, we saw that a low-HW input difference seems to be better suited than one from a high-probability differential when using a neural distinguisher, being constantly retrieved using different methods. Therefore, we chose such low-HW input differences to construct our datasets, and we obtained them with the help of an evolutionary algorithm. Third, we also looked at how to shape the individual data samples, i.e., ciphertext pairs, before training the neural distinguishers on them. Here, we wanted to know whether the way one reshapes the individual data samples matters for achieving better performance, and we found out that it matters. Concretely, more complex neural networks benefitted from reshaping the two ciphertexts so that they reveal the word structure of the cipher, while simpler networks preferred training directly on the concatenation of the two ciphertexts.

We then continued looking at various neural networks that were proposed in past papers and wanted to know whether there is an architecture that generally works well across ciphers. We found out that the DBIT (DBitNet) neural network proposed in [9] appears to offer the best* performance on the studied ciphers. We also saw how neural distinguishers "could be used to extend the number of rounds that classical distinguishers can attack" [11], as well as strategies to improve the performance of a neural distinguisher. Among those strategies, we encountered the iterative training and the freezing layer strategies, and we wanted to know whether they generally help a neural distinguisher achieve better performance. We found that the iterative training strategy did not seem to help, which was not surprising in light of the discussion from [6]. Nevertheless, the freezing layer strategy, although minimally, appeared to help improve the performance of the neural distinguishers, provided they had a clearly defined separation between the feature extractor and classifier. We concluded the section on neural networks with an overview of how to interpret the accuracy of a neural distinguisher when assigning a class during the classification/distinguishing of the ciphertexts and moved on to the next section.

In the next section, we saw several methods that tried to explain the behavior of neural distinguishers. Explaining what a (strong) neural distinguisher is learning and what it is looking at when classifying a ciphertext difference/pair/etc. is highly necessary for understanding the weaknesses it identified for a particular cipher. Without this part, the neural distinguisher, despite its high accuracy that signals a weakness, offers no actual information to cryptanalysts on what that weakness might be since it is a black box. Out of the presented methods of explaining the behavior of a neural distinguisher, we applied the bit profiling tool NNBits to other ciphers, hoping that similar conclusions could be drawn as seen in [20], but it was not the case. While NNBits could identify patterns in the ciphertext pair bit sequence of Speck32/64 and recover the performance of ResNet depth-1 completely based on single-bit predictions, it could not do that for other ciphers. "Only a small percentage of Gohr's neural distinguisher could be explained in terms of the correctness of single-bit predictions" [20] for the two studied ciphers despite ResNet achieving a high (base) accuracy, leaving the interpretation of results open for future research.

Finally, we looked at a recently developed evolutionary algorithm with which we obtained good, low-HW input differences quickly and irrespective of the cipher used. As mentioned, we used those when constructing the datasets for our seven experiments, and we obtained satisfying results.

Recommendations: In light of the results of our experiments, we recommend giving the neural distinguisher as much information to train on as possible, i.e., pairs+. However, if the computation limitations do not allow that, we recommend training at least on the full ciphertext pairs. Furthermore, to achieve better performance with a complex neural distinguisher, we recommend reshaping the ciphertexts so that the word structure of the cipher is revealed. Nevertheless, since, for some weaker networks, the opposite is the case, we still advise trying both reshaping as well as simply concatenating the ciphertexts to check how a (new) neural distinguisher is behaving. Regarding what neural network to use, out of the ones we studied in this thesis, we recommend the DBIT neural network as its performance consistently improves with the amount of information provided, it does not require the input to be reshaped, it is quite fast, and performs significantly better than the other* neural networks studied. Regarding the training strategies, if one would like to use one, we recommend the freezing layer strategy if the network has a clear separation between the feature extractor and the classifier, although the improvements are minimal. Finally, since the bit profiling tool NNBits is cipher agnostic, we recommend still using it on new ciphers to see what it finds since, as we have seen, the results are cipher-specific.

Future work: Since interpreting the behavior of a neural distinguisher is crucial for understanding what weaknesses it identified while training, we suggest changing the neural distinguisher used in NNBits (ResNet) with a stronger one (DBIT). By doing this, hopefully, NNBits is able to identify more weak bits and recover an accuracy closer to that of the (stronger) neural distinguisher. In addition, it would be interesting to deploy NNBits on other unstudied ciphers as well to see what it finds.

Bibliography

[1] Transfer learning. https://en.wikipedia.org/wiki/Transfer_learning.

[2] Farzaneh Abed, Eik List, Stefan Lucks, and Jakob Wenzel. Differential cryptanalysis of round-reduced simon and speck. In Carlos Cid and Christian Rechberger, editors, *Fast Software Encryption*, pages 525–545, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[3] Norica Băcuieți, Lejla Batina, and Stjepan Picek. Deep neural networks aiding cryptanalysis: A case study of the speck distinguisher. In Giuseppe Ateniese and Daniele Venturi, editors, *Applied Cryptography and Network Security*, pages 809–829, Cham, 2022. Springer International Publishing.

[4] Anubhab Baksi. *Machine Learning-Assisted Differential Distinguishers for Lightweight Ciphers*, pages 141–162. Springer Singapore, Singapore, 2022.

[5] Anubhab Baksi, Jakub Breier, Vishnu Asutosh Dasu, Xiaolu Hou, Hyunji Kim, and Hwajeong Seo. New results on machine learning-based distinguishers. *IEEE Access*, 11:54175–54187, 2023.

[6] Zhenzhen Bao, Jinyu Lu, Yiran Yao, and Liu Zhang. More insight on deep learning-aided cryptanalysis. Cryptology ePrint Archive, Paper 2023/1391, 2023. https://eprint.iacr.org/2023/1391.

[7] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK families of lightweight block ciphers. Cryptology ePrint Archive, Paper 2013/404, 2013. https://eprint.iacr.org/2013/404.

[8] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. Cryptology ePrint Archive, Paper 2016/660, 2016. https://eprint.iacr.org/2016/660.

[9] Emanuele Bellini, David Gerault, Anna Hambitzer, and Matteo Rossi. A cipher-agnostic neural training pipeline with automated finding of good input differences. Cryptology ePrint Archive, Paper 2022/1467, 2022. https://eprint.iacr.org/2022/1467.

[10] Emanuele Bellini and Matteo Rossi. Performance comparison between deep learning-based and conventional cryptographic distinguishers. In Kohei Arai, editor, *Intelligent Computing*, pages 681–701, Cham, 2021. Springer International Publishing.

[11] Adrien Benamira, David Gerault, Thomas Peyrin, and Quan Quan Tan. A deeper look at machine learning-based cryptanalysis. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021*, pages 805–835, Cham, 2021. Springer International Publishing.

[12] Daniel Bernstein. Chacha, a variant of salsa20. 01 2008.

[13] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. Present: An ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007*, pages 450–466, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[14] Yi Chen, Yantian Shen, Hongbo Yu, and Sitong Yuan. A new neural distinguisher considering features derived from multiple ciphertext pairs. Cryptology ePrint Archive, Paper 2021/310, 2021. https://eprint.iacr.org/2021/310.

[15] Yi Chen and Hongbo Yu. Bridging machine learning and cryptanalysis via edlct. Cryptology ePrint Archive, Paper 2021/705, 2021. https://eprint.iacr.org/2021/705.

[16] Christophe De Cannière, Orr Dunkelman, and Miroslav Knežević. Katan and ktantan — a family of small and efficient hardware-oriented block ciphers. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009*, pages 272–288, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[17] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks, 2019.

[18] Aron Gohr. Improving attacks on round-reduced speck32/64 using deep learning. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pages 150–179, Cham, 2019. Springer International Publishing.

[19] Aron Gohr, Gregor Leander, and Patrick Neumann. An assessment of differential-neural distinguishers. Cryptology ePrint Archive, Paper 2022/1521, 2022. https://eprint.iacr.org/2022/1521.

[20] Anna Hambitzer, David Gerault, Yun Ju Huang, Najwa Aaraj, and Emanuele Bellini. Nnbits: Bit profiling with a deep learning ensemble based distinguisher. In Mike Rosulek, editor, *Topics in Cryptology – CT-RSA 2023*, pages 493–523, Cham, 2023. Springer International Publishing.

[21] Zezhou Hou, Jiongjiong Ren, and Shaozhen Chen. Cryptanalysis of round-reduced simon32 based on deep learning. Cryptology ePrint Archive, Paper 2021/362, 2021. https://eprint.iacr.org/2021/362.

[22] Zezhou Hou, Jiongjiong Ren, and Shaozhen Chen. Improve neural distinguisher for cryptanalysis. Cryptology ePrint Archive, Paper 2021/1017, 2021. https://eprint.iacr.org/2021/1017.

[23] Aayush Jain, Varun Kohli, and Girish Mishra. Deep learning based differential distinguisher for lightweight cipher present. Cryptology ePrint Archive, Paper 2020/846, 2020. https://eprint.iacr.org/2020/846.

[24] Stefan Kölbl, Gregor Leander, and Tyge Tiessen. Observations on the SIMON block cipher family. Cryptology ePrint Archive, Paper 2015/145, 2015. https://eprint.iacr.org/2015/145.

[25] Dongdong Lin, Manman Li, Zezhou Hou, and Shaozhen Chen. Conditional differential analysis on the katan ciphers based on deep learning. *IET Information Security*, 17(3):347–359, nov 2022.

[26] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier, 2016.

[27] Gao Wang and Gaoli Wang. Improved differential-ml distinguisher: Machine learning based generic extension for differential analysis. In Debin Gao, Qi Li, Xiaohong Guan, and Xiaofeng Liao, editors, *Information and Communications Security*, pages 21–38, Cham, 2021. Springer International Publishing.

[28] Tarun Yadav and Manoj Kumar. Differential-ml distinguisher: Machine learning based generic extension for differential cryptanalysis. In Patrick Longa and Carla Ràfols, editors, *Progress in Cryptology – LATINCRYPT 2021*, pages 191–212, Cham, 2021. Springer International Publishing.

[29] Xiaoteng Yue and Wanqing Wu. Improved neural differential distinguisher model for lightweight cipher speck. *Applied Sciences*, 13:6994, 06 2023.

[30] Liu Zhang and Zilong Wang. Improving differential-neural distinguisher model for des, chaskey and present. Cryptology ePrint Archive, Paper 2022/457, 2022. https://eprint.iacr.org/2022/457.

[31] Liu Zhang, Zilong Wang, and Baocang wang. Improving differential-neural cryptanalysis. Cryptology ePrint Archive, Paper 2022/183, 2022. https://eprint.iacr.org/2022/183.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every written paper or thesis authored during the course of studies. In consultation with the supervisor, one of the following three options must be selected:

I confirm that I authored the work in question independently and in my own words, i.e. that no one helped me to author it. Suggestions from the supervisor regarding language and content are excepted. I used no generative artificial intelligence technologies[1].

I confirm that I authored the work in question independently and in my own words, i.e. that no one helped me to author it. Suggestions from the supervisor regarding language and content are excepted. I used and cited generative artificial intelligence technologies[2].

I confirm that I authored the work in question independently and in my own words, i.e. that no one helped me to author it. Suggestions from the supervisor regarding language and content are excepted. I used generative artificial intelligence technologies[3]. In consultation with the supervisor, I did not cite them.

Title of paper or thesis:

Authored by:
If the work was compiled in a group, the names of all authors are required.

Last name(s):	**First name(s):**

With my signature I confirm the following:
- I have adhered to the rules set out in the Citation Guide.
- I have documented all methods, data and processes truthfully and fully.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for originality.

Place, date	**Signature(s)**

If the work was compiled in a group, the names of all authors are required. Through their signatures they vouch jointly for the entire content of the written work.

[1] E.g. ChatGPT, DALL E 2, Google Bard
[2] E.g. ChatGPT, DALL E 2, Google Bard
[3] E.g. ChatGPT, DALL E 2, Google Bard