A Cipher-Agnostic Neural Training Pipeline with Automated Finding of Good Input Differences

Emanuele Bellini[1], David Gerault[1], Anna Hambitzer[1] and Matteo Rossi[2]

[1] Technology Innovation Institute, Abu Dhabi, UAE {name.lastname}@tii.ae
[2] Politecnico di Torino, Torino, Italy matteo.rossi@polito.it

Abstract. Neural cryptanalysis is the study of cryptographic primitives through machine learning techniques. Following Gohr's seminal paper at CRYPTO 2019, a focus has been placed on improving the accuracy of such distinguishers against specific primitives, using dedicated training schemes, in order to obtain better key recovery attacks based on machine learning. These distinguishers are highly specialized and not trivially applicable to other primitives. In this paper, we focus on the opposite problem: building a *generic* pipeline for neural cryptanalysis. Our tool is composed of two parts. The first part is an evolutionary algorithm for the search of good input differences for neural distinguishers. The second part is DBitNet, a neural distinguisher architecture agnostic to the structure of the cipher. We show that this fully automated pipeline is competitive with a highly specialized approach, in particular for SPECK32, and SIMON32. We provide new neural distinguishers for several primitives (XTEA, LEA, HIGHT, SIMON128, SPECK128) and improve over the state-of-the-art for PRESENT, KATAN, TEA and GIMLI.

Keywords: Neural Cryptanalysis · Differential Cryptanalysis · Evaluation Tools · Block Cipher · Distinguisher · Neural Networks

1 Introduction

The security of most modern applications is related to the security of the underlying symmetric encryption primitive. Since the foundations of modern cryptography through the Data Encryption Standard (DES), the security needs and applications have considerably evolved, so that a variety of new designs have appeared over the years: candidates for the Advanced Encryption Standard (AES), the eSTREAM portfolio, the CAESAR competition, and many more. The building of new primitive goes hand-in-hand with the discovery of new attack techniques, such as differential and linear cryptanalysis [BS91], impossible differential cryptanalysis [Knu98], or integral attacks [KW02]. The joint growth of the number of ciphers to analyze, and the number of techniques to analyze them against, has created a strong need for automation. The analysis of a cipher against newly found techniques is not straightforward, as the process usually starts as a specialized, human-input-heavy task, until research progress makes more and more steps automatic. For instance, differential cryptanalysis requires finding long high-probability propagation patterns through the cipher. This highly combinatorial problem was initially tackled by manually implementing Matsui's branch-and-bound algorithm [Mat94] for the cipher under study, a time-consuming and error-prone process. In 2012, after almost 2 decades, Mouha *et al.* [MWGP12] proposed the use of Mixed Integer Linear Programming for this problem, making it significantly easier and faster to solve. In this declarative approach, the cryptographer focuses on the description of the problem, while the search algorithm itself is delegated to a specialized solver. Declarative approaches (MILP, SAT, SMT, CP...) have since then become the de facto standard for differential cryptanalysis. More

recently, open-source cryptographic libraries such as Tagada [LDLS21], Cascada [RR22] or CLAASP [BGG+23] have made the process fully automated: from the description of a cipher, these libraries build and solve the declarative models for the search of optimal differential characteristics, without human intervention. A similar slow automation process was followed for other techniques, such as linear or impossible differential cryptanalysis, which are implemented within these libraries as well. Incidentally, as cryptographers become able to run these search problems more efficiently, the corresponding cryptanalysis techniques become more and more refined, as the time investment shifts from *finding* a distinguisher to *exploiting* it.

Recently, a new cryptanalysis technique emerged, based on deep learning: *neural cryptanalysis*. Proposed by Gohr at CRYPTO'19 [Goh19b], it exploits the ability of deep learning algorithms to recognize complex patterns to identify relations between sets of ciphertext that distinguishes them from random data. In the seminal work [Goh19b], this relation is differential in nature; given pairs of ciphertexts $(C_0 = E_K(P_0), C_1 = E_K(P_1))$ (with $E_K(X)$ denoting the encryption of X with the key K through a number of rounds of a block cipher), the *neural distinguisher* is trained to determine whether $P_0 \oplus P_1 \stackrel{?}{=} \delta$. Gohr's neural distinguishers on 5, 6, 7 and 8 rounds of SPECK32, using $\delta = 0x400000$, enabled key recovery attacks for 11 and 12 rounds with better complexity than the state of the art. The approach taken by subsequent work has often been to optimize a neural distinguisher for a given cipher, by carefully tuning its parameters, to build the best key recovery attacks. For instance, techniques based on *staged training* improve the accuracy of the neural classifier, by exploiting information obtained through differential cryptanalysis. Similarly, the manual transformation of the ciphertext pairs, for instance by reversing some operations in the last round, has been used to obtain better accuracies. In comparison with other cryptanalysis techniques, the field is still in the specialized, human-input-heavy phase, and it often requires significant effort to obtain good neural distinguishers for a specific cipher. At AICrypt'23 [GLN23] Gohr *et al.* address the question of the potential of neural distinguishers as a generic tool for cryptanalysis, *i.e.,* "...how generic this approach is and to which extent it can complement the work of a cryptanalyst. In other words, can we see machine learning as a tool assisting cryptanalysis, similar to how SAT and MILP solvers, among others, are seen by now?".

In this paper, we propose a step forward towards the fully automated route, through a generic pipeline: suitable input difference δ candidates are obtained through an evolutionary algorithm, and are used to train DBitNet, a fully generic neural network that requires no tuning nor human input. The neural distinguishers obtained through our pipeline are competitive with, and sometimes better than, specialized approaches on the ciphers for which they were designed. With this work, we hope to provide a basis on which other researchers can improve neural cryptanalysis, and apply it to more ciphers, without the burden of optimizing the neural distinguisher itself.

Contributions

1. We propose a fully automated framework to perform neural cryptanalysis of ciphers; our tool is publicly available on Github (https://github.com/Crypto-TII/AutoND) is composed of (i) a scalable input-difference finding algorithm (ii) DBitNet, a neural distinguisher architecture agnostic to the structure of the cipher (iii) an automatic and simple training pipeline, which generically replaces informed techniques of staged training

2. Using our tool we propose competitive neural distinguishers with the following advantages: we can replace the elaborate training pipelines for SPECK32 [Goh19b] and SIMON32 [BGL+22], provide distinguishers for several new primitives (XTEA,

LEA, HIGHT, SIMON128, SPECK128) and improve over the state-of-the-art for PRESENT, KATAN, TEA and GIMLI.

In Table 1, we present a comparison summary of the neural distinguishers obtained in this work with the state of the art.

Table 1: Summary of the state-of-the-art of published neural distinguishers for selected primitives, with the highest achieved *Round* and *Accuracy*. To give the context of the values, we show the architecture (*Arch.*), the number of training, validation samples (*Trn.*, *Val.*), and the *Setting* in which the neural distinguisher was characterized –the gray highlighted setting is not directly comparable to the standard setting in which each sample is built by two ciphertexts. We also highlight the highest achieved round in the standard setting. *AutoND* indicates if the neural distinguisher was automatically generated (✓) or is the result of an elaborate, manually designed training procedure (-). In the table, / means unknown.

Primitive	Arch.	Setting[S]	Trn.	Val.	AutoND	Rounds	Acc.	Ref.
SPECK32	MLP	2-1-δ-R	$2^{27.64}$	$2^{26.64}$	-	3^{*}	0.79	[YK21]
	ResNet	20-1-A-R	$2^{23.25}$	$2^{19.93}$	-	5	1	[BGPT21]
	ResNet	100-1-A-R	$2^{23.25}$	$2^{19.93}$	-	6	1	[BGPT21]
	ResNet	64-1-CT-R	$2^{23.25}$	$2^{19.93}$	-	8	0.939	[CSYY22]
	ResNet	2-1-CT-R	$2^{31.49}$	$2^{19.93}$	-	8	0.514	[Goh19b]
	DBitNet	*2-1-CT-R*	$2^{31.49}$	$2^{19.93}$	✓	*8*	*0.514*	*This work*
	ResNet	64-1-δ-R	$2^{28.25}$	/	-	8	0.564	[HRCF21]
SPECK64	*DBitNet*	*2-1-CT-R*	$2^{23.25}$	$2^{19.93}$	✓	*8*	*0.537*	*This work*
	ResNet	128-1-δ-R	$2^{29.25}$	/	-	8	0.632	[HRCF21]
SPECK128	*DBitNet*	*2-1-CT-R*	$2^{23.25}$	$2^{19.93}$	✓	*10*	*0.592*	*This work*
SIMON32	MLP	2-1-δ-R	2^{24}	$2^{27.64}$	-	5^{*}	0.570	[YK21]
	ResNet	4-3-CT-R	$2^{23.25}$	$2^{19.93}$	-	9	0.637	[SZM20]
	ResNet	2-1-CT-R	$2^{23.25}$	$2^{19.93}$	✓	9	0.661	[GLN23]
	ResNet	64-1-δ-R	$2^{28.25}$	/	-	10	0.611	[HRCF21]
	SENet	2-1-A-R	$2^{31.17}$	$2^{29.17}$	-	11	0.517	[BGL+22]
	DBitNet	*2-1-CT-R*	$2^{31.49}$	$2^{19.93}$	✓	*11*	*0.518*	*This work*
	ResNet	2-1-CT-R	$2^{27.58}$	$2^{23.25}$	-	11	0.520	[GLN23]
	SE-ResNet	16-1-A-R	$2^{24.25}$	$2^{20.90}$	-	12	0.514	[LLS+23]
SIMON64	ResNet	128-1-δ-R	$2^{29.25}$	/	-	12	0.695	[HRCF21]
	DBitNet	*2-1-CT-R*	$2^{23.25}$	$2^{19.93}$	✓	*13*	*0.518*	*This work*
	SE-ResNet	16-1-A-R	$2^{24.25}$	$2^{20.90}$	-	14	0.519	[LLS+23]
SIMON128	*DBitNet*	*2-1-CT-R*	$2^{23.25}$	$2^{19.93}$	✓	*20*	*0.506*	*This work*
GIMLI	MLP	2-2-δ-D	$2^{17.6}$	$2^{14.3}$	-	8	0.510	[BBCD21]
	DBitNet	*2-1-CT-R*	$2^{23.25}$	$2^{19.93}$	✓	*11*	*0.527*	*This work*
HIGHT	*DBitNet*	*2-1-CT-R*	$2^{23.25}$	$2^{19.93}$	✓	*10*	*0.751*	*This work*
HIGHT[RK]	*DBitNet*	*2-1-CT-R*	$2^{23.25}$	$2^{19.93}$	✓	*14*	*0.563*	*This work*
KATAN	ResNet	2-1-δ-R	$2^{23.25}$	$2^{19.93}$	-	51	0.533	[LCLH22]
	ResNet	64-1-δ-R	$2^{23.25}$	$2^{19.93}$	-	59	0.575	[LCLH22]
	ResNet	2-1-CT-R	$2^{23.25}$	$2^{19.93}$	✓	66	0.505	[GLN23]
	DBitNet	*2-1-CT-R*	$2^{23.25}$	$2^{19.93}$	✓	*69*	*0.505*	*This work*
PRESENT	ResNet	2-1-CT-R	$2^{23.25}$	$2^{19.93}$	✓	7	0.563	[GLN23]
	ResNet	8-1-CT-R	$2^{23.25}$	$2^{19.93}$	-	7	0.585	[CSYY22]
	DBitNet	*2-1-CT-R*	$2^{23.25}$	$2^{19.93}$	✓	*9*	*0.509*	*This work*
TEA[*2, *3]	MLP	*2-1-CT-R*+	$2^{19.93}$	$2^{13.28}$	-	*4*	*0.545*	[BR21]
	DBitNet	*2-1-CT-R*	$2^{23.25}$	$2^{19.93}$	✓	*5*	*0.563*	*This work*
XTEA[*2]	*DBitNet*	*2-1-CT-R*	$2^{23.25}$	$2^{19.93}$	✓	*5*	*0.598*	*This work*
LEA	*DBitNet*	*2-1-CT-R*	$2^{23.25}$	$2^{19.93}$	✓	*11*	*0.511*	*This work*

[S] The parameters *n-m-T-E* of the settings column respectively denote the number of ciphertexts per sample n, of input differences m, the feature engineering type (T: CT for ciphertexts, δ for the difference, A for advanced techniques), and the type of experiment (E: R when the labels correspond to random or real, D when the label depends on the index of the input difference), as detailed in Section 3.

[RK] Related key setting.

[*] [YK21] prepends probabilistic rounds to reach 9 (12) round distinguishers for SPECK32 (SIMON32), using 2^{20} (2^{22}) pairs.

[*2] For TEA and XTEA, we report the number of full 2-round cycles rather than the number of individual rounds.

[*3] *2-1-CT-R*+ denotes modular addition rather than XOR to inject the difference (as defined in Section 3).

Organization. The remainder of this work is organized as follows. We first give a short introduction to the ciphers analyzed in this work, as well as to machine learning and Gohr's distinguisher in our preliminaries (Section 2). We then discuss related work in Section 3, and obstacles to the automatic application of neural distinguishers to new ciphers in Section 4. We present our solutions to I) the automated finding of a good input difference (Section 5), as well as II) a cipher-agnostic neural training pipeline (Section 6). We present our best distinguishers in Section 7, discuss them in Section 8 and conclude in Section 9.

2 Preliminaries

In this section, we summarize the basic terminology used throughout the paper, including the concept of differential cryptanalysis (Section 2.1). We provide the list of ciphers analyzed in this work (Section 2.2), as well as a short, general introduction to machine learning and neural networks (Section 2.3), and Gohr's basic scheme (Section 2.4).

2.1 Differential Cryptanalysis

Differential cryptanalysis, introduced in [BS91], is a cryptanalysis technique, which focuses on the propagation of an *input difference* between the inputs of a cryptographic function to its outputs. In this work, we focus on *iterated symmetric ciphers* $E_K(P)$, which apply a simple round function iteratively to a plaintext $P \in \mathcal{P}$ and a key $K \in \mathcal{K}$ to obtain a ciphertext $C \in \mathcal{P}$. In the differential cryptanalysis of block ciphers, the cryptographer is interested in the probability that, for a random key K and plaintext P, the *differential* $\delta \to \gamma$ holds with probability $\#\{E_K(P) = E_K(P \oplus \delta) \oplus \gamma : P \in \mathcal{P}, K \in \mathcal{K}\}/(|\mathcal{P}| + |\mathcal{K}|)$. Sometimes, *truncated* differentials, where not all bits of γ are considered, are of interest.

2.2 Analyzed ciphers

The following ciphers have been considered in this work, for their variety of structures, block and key sizes. Several have been studied in the differential-neural setting, providing a baseline for comparison.

SPECK and **SIMON** [BTCS+15] are lightweight block ciphers with block sizes ranging from 32 to 128 bits and key sizes from 64 to 256. SPECK has a classical ARX (Addition, Rotation, XOR) design, while SIMON is a Feistel structure, with the bitwise-and function as the non-linear operation. **LEA** [HLK+14] is an ARX-based lightweight block cipher that encrypts 128-bit blocks, with 128 or 256-bit keys. **TEA** [WN95] is a Feistel-based ARX cipher with a block size of 64-bit and a key size of 128-bit. In TEA round keys are injected through modular addition, rather than XOR. **XTEA** [WN97] is TEA's successor, fixing some of its weaknesses, and reverting to key injection by the XOR operation. **GIMLI** [BKL+17] is a permutation with a state size of 384 bits arranged in a 3×4 matrix of 32-bit words. From this permutation, the authors proposed a hashing algorithm and an authenticated encryption algorithm, respectively GIMLI-HASH and GIMLI-CIPHER. Its round function combines an SP-box with a linear layer, and it is iterated 24 times. GIMLI-HASH is built from it with a sponge construction, while GIMLI-CIPHER uses the monkeyDuplex one. **HIGHT** [HSH+06] is a generalized Feistel-based ARX block cipher, with a 64-bit block size and 128-bit key size. **KATAN** [DCDK09] is a family of hardware-oriented block cipher, working with 80-bit keys and 32, 48, or 64-bit block sizes. **PRESENT** [BKL+07] is a lightweight block cipher with an SPN structure, a block size of 64 bits and two possible key sizes: 80 and 128 bits.

2.3 Machine Learning and Neural Networks

Machine learning (ML) is a subfield of Artificial Intelligence (AI) investigating algorithms that "give computers the ability to learn without explicitly being programmed" [Sam59]. Deep learning is a subfield of ML that uses deep neural networks. The following short introduction to neural networks is based on [GBC17].

A *deep neural network* f is a machine learning algorithm, dependent on a set of parameters θ, that uses multiple stacked layers of artificial neurons to map an input $\mathbf{x}$ to an output $\mathbf{y}$, *i.e.,* $\mathbf{y} = f_\theta(\mathbf{x})$. During the neural network training, the values of the network parameters θ are "learned", i.e., deduced. In a feedforward neural network, the inputs $\mathbf{x}$ are passed through the different *layers* of the neural network. For example, the three-layer network $\mathbf{y} = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$ has one output layer (3), one hidden layer (2), and one input layer (1). To train the neural network, the current network output $\mathbf{y}_{\text{pred}}$ is compared to a ground truth $\mathbf{y}_{\text{true}}$ by means of a scalar cost function $J(\theta)$, *e.g.*, the mean squared error MSE. The scalar cost $J(\theta)$ is *back-propagated* to the neural network parameters θ, resulting in the *gradient* of the cost function with respect to the parameters $\nabla_\theta J(\theta)$.

An *optimizer*, such as ADAM, adjusts the θ based on the gradient value, and parameters such as the *learning rate*. Training of the neural network is performed in three phases, acting on three datasets structured as a list of input/output values: training, validation and test data. The goal of machine learning, in general, is *not* to optimize the algorithm f on a known dataset but to make the algorithm able to *generalize* from seen to unseen data. Therefore, evaluation of the training progress is done on previously unseen validation data. At some point during the training, the network may start to get worse at generalization and learn the training dataset "by heart". This phenomenon is known as *overfitting*. Many *regularization techniques* can be used to avoid overfitting, among them the *L2 regularization* that penalizes overly large weight values.

The training of a neural network is done in *batches* and *epochs*. One epoch means the neural network has been optimized on the full training dataset. Within each epoch, the network parameters are adjusted after each batch of training data. Batch sizes vary between 1 and the full training dataset size, and influence the learning outcome immensely.

The design of a neural network involves choosing the number of layers, *i.e.,* the depth, and the types of layers. The type of each layer is determined by the way in which the neurons of the layers are connected to each other: For example, a layer can be *dense* –here, every neuron is connected to all neurons in the previous layer– or *convolutional* –here, every neuron is connected only to a subset of neurons in the previous layer, inspired by the mammalian visual cortex. The layers present in the neural network determine its type: a *CNN* (convolutional neural network) or *MLP* (a densely connected multi-layer network). The *architecture* of a neural network refers to its overall structure, *i.e.*, how many neurons are connected, in which way, and in how many layers.

A *hyperparameter* is a parameter that is set *before* beginning the model training. A hyperparameter affects the behavior of the deep neural network, for example, its generalization error or the cost of training the algorithm. Hyperparameters include the architecture parameters (number of neurons, etc.), as well as the training parameters (learning rate, batch size, etc.).

One problem in training deep neural networks can be that one layer stops learning. This will also prevent the gradient information from successfully back-propagating to previous layers. This is circumvented by introducing *residual* or *skip* connections. Often, neural networks in which such residual connections are present are referred to as a *ResNet*.

One of the most fundamental design choices is the choice *nonlinearities* applied after each neuron, with typical choices being ReLU, sigmoid, and tanh. The *sigmoid* activation outputs a value between 0 and 1, which can be interpreted as a confidence score.

2.4 Gohr's Basic Scheme

In his seminal paper, published at CRYPTO'19, Aron Gohr [Goh19b] proposes to use a neural network to distinguish whether pairs of SPECK32/64 ciphertexts correspond to the encryption of pairs of messages with a fixed difference $(0x0040, 0x0000)$–labeled as *"non-random"* (1)– or random input differences –labeled as *"random"* (0). The resulting *neural distinguisher*, a residual neural network preceded by a size 1 1D-convolution, results in respectively $92.9, 78.8, 61.6$ and 51.4% accuracy for 5, 6, 7, and 8 rounds of SPECK32/64, and is used to mount practical key recovery attacks on 11 rounds. Gohr also proposes a neural difference search algorithm, based on transfer learning, to search for input differences that function well with neural distinguishers.

Gohr's neural distinguisher is a ResNet with four main parts, the first three of which are visualized in Fig. 1. At the input, a 64-bit ciphertext pair of SPECK32/64 is reshaped

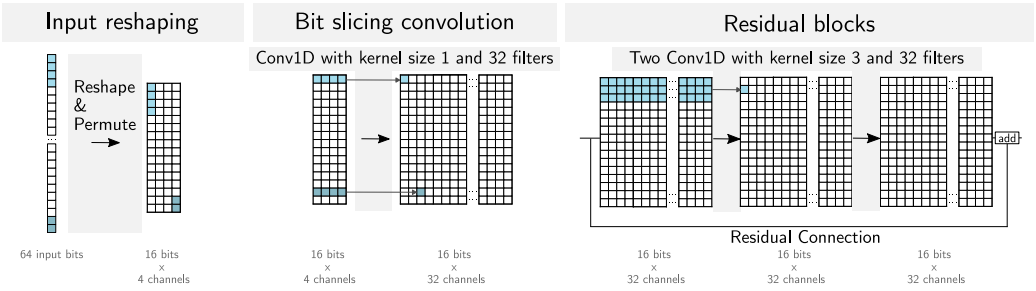

Figure 1: Visualization of three main parts of Gohr's neural distinguisher.

and permuted into a 16-bit wide tensor with 4 channels. From a cryptographic perspective, the input reshaping reflects the knowledge of the particular 16-bit word structure of SPECK32/64. In the second part, a 1-dimensional convolution (Conv1D($k = 1, f = 32$)) is used to slice through the 4 channel bits. The "slicing" is reflected by the kernel size of $k = 1$. The output channel for each filter is produced by scanning the corresponding filter over the input in one dimension, hence Conv1D. The learnable parameters are four filter weights, as well as one bias parameter for each of the $f = 32$ filters, resulting in a total of $32 \times 4 + 32 = 160$ learnable parameters for this Conv1D-layer. The output tensor of the bit-slicing convolution is 16 bits wide and 32 channels deep. Throughout the network, each convolutional layer is followed by conventionally used BatchNormalization and ReLU nonlinearity. The third part is the residual blocks. Each residual block consists of two convolutional layers Conv1D($k = 3, f = 32$). In Gohr's publication, the number of residual blocks denotes a depth-1 neural distinguisher, respectively a depth-10 neural distinguisher[1]. The fourth part of the network is a densely connected prediction head with ReLU activations and an output layer with a single neuron with sigmoid activation. Throughout Gohr's network, each convolutional, and each dense layer is regularized by an L2 $= 10^{-5}$ parameter. The full Python TensorFlow implementation is available on GitHub [Goh19a].

Gohr additionally proposes an algorithm to derive good input differences for neural distinguishers without prior human knowledge. This algorithm is based on few-shot learning, where the features learned by a network are used as input to a simpler machine learning algorithm, trained on fewer samples. In practice, a one-block residual network is trained with a random (but fixed) input difference δ on 3 rounds of Speck with 10^7 ciphertext pairs; the output of the penultimate layer of this network is then used as input

[1] It is not conventional to refer to the number of residual blocks as depth. For example in the original publication of the first residual network ResNet [HZRS15], ResNet34 consists of 34 weighted layers, including a fully connected dense layer, while it has only 16 residual blocks.

to train a ridge regression classifier on small numbers of samples for new differences δ'. A greedy algorithm[2] with exploration bias is used to suggest new candidates δ'.

3 Related Work

Following the introduction of Gohr's neural distinguisher, subsequent work has mostly followed two trends: the exploration of new settings for the neural distinguisher experiments (Section 6.2) and the explainability of neural distinguishers (Section 3.2). Finally, a line of research focuses on automatically building good neural distinguishers for new primitives, *i.e.*, the fully automated route described in the introduction. We discuss these in more detail in the next section (Section 4).

3.1 Extensions of Gohr's Basic Scheme

Neural distinguisher research, following the seminal paper [Goh19b], has often focused on modifications of either the neural network architecture or the setting in which the experiments take place. These modifications to the experimental setting have been along 4 dimensions: the number of plaintexts per sample n, the number of input differences m, the feature engineering type T, and the experiment setting E. The neural distinguishers on the primitives we studied are classified in terms of their setting (n, m, T, E), along with their architecture, in Table 1, and we discuss each setting parameter in the following.

Number of plaintexts per sample: n A natural way to amplify the accuracy of a neural distinguisher is to group multiple pairs sharing the same label and combine their scores. In this approach, the distinguisher may be trained on single pairs, and evaluated on multiple pairs sharing the same label, as in [Goh19b] (key recovery part), [BGPT21]. Gohr *et al.* [GLN23] give a formula to compute multiple-pair accuracy from a single-pair distinguisher. Sometimes, the samples used by the neural network themselves are the concatenation of multiple ciphertexts; this is the case in [BGPT21] ($n = 20, 100$), [CSYY22] ($n = 8$), [HRCF21] ($n = 64, 128$) and [LLS+23] ($n = 16$).

Number of input differences: m Baksi *et al.* [BBCD21] explore a setting where a set of m input differences are considered. This setting was applied to various permutations: KNOT, ASCON, CHASKEY and GIMLI, with $m = 2$ for GIMLI. Su *et al.* [SZM20] introduced a model called polytope differential neural network distinguisher. In this model multiple differences are used, keeping one plaintext fixed among the differences and changing the other.

Feature engineering type: T Feature engineering is often used in machine learning, to derive advanced features from the raw dataset, *e.g.,* [GBC17]. A natural feature to use for differential neural cryptanalysis is to replace the ciphertext pairs ($T = \mathrm{CT}$) by their XOR difference ($T = \delta$). This approach, used by Baksi *et al.* [BBCD21], Hou *et al.* [HRCF21], and Yadav *et al.* [YK21], simplifies the training process, at the cost of losing some information.

Advanced types of feature engineering ($T = \mathrm{A}$) include, *e.g.,* partial decryption of the ciphertexts. For instance, in the case of SPECK32, the right half of the previous round state can be computed without the key, by XORing the two halves and rotating. This type of feature engineering was used in [BGPT21]. A similar technique permits to retrieve the difference in the previous round for SIMON-like ciphers; [BGL+22] showed that this transformation could significantly improve the accuracies of neural distinguishers,

[2] This algorithm is summarized in the appendix as Algorithm 2.

and [LLS+23] exhibited even better distinguishers on SIMON by exploiting inferred information from two rounds ahead.

Type of distinguishing experiment: E In the initial setting [Goh19b] ($E = \mathrm{R}$), the samples are $E_K(P_0)||E_K(P_0 \oplus x)$, and the label is $x \stackrel{?}{=} \delta$. Gohr additionally defines the *real ciphertext* experiment ($E = \mathrm{R_M}$), where the samples are $E_K(P_0) \oplus x||E_K(P_0 \oplus \delta) \oplus x$, and the label is $x \stackrel{?}{=} 0$, *i.e.*, the distinguisher determines whether the ciphertext pair has been XORed with a random mask. The success of neural distinguishers in this experiment shows that information beyond a simple XOR difference is learned.

In [BBCD21]'s model 1, the samples are formed as $(E_K(P) \oplus E_K(P \oplus \delta_i)), i \in [0; m-1]$, and the label is i ($E = \mathrm{D}$).

In [BR21], the samples are built using modular addition difference, rather than XOR, to analyze the ciphers TEA and RAIDEN ($E = \mathrm{R}^+$).

3.2 Explainability of Neural Distinguishers

Neural distinguishers enabling new attacks, potentially better than manual cryptanalysis, motivated researchers to try to understand what made these attacks so powerful, and to learn new properties from these.

In [BGPT21], Benamira *et al.* studied the properties of pairs that were correctly classified, and proposed that Gohr's neural distinguishers learn differential-linear features. In particular, they observe that the pairs for which the score of the neural distinguisher follow similar truncated differential patterns a few rounds ahead. The authors further modified the neural network to use a Heaviside activation function, which forces its output to be 0 or 1, to study the Boolean functions learned on SPECK. From these, they derived advanced features that could be used to replace the initial 1D convolutions of Gohr's network.

In [BBP22], Bacuieti *et al.* further investigate the structure of the neural network itself. In particular, they use the *lottery ticket hypothesis* to prune Gohr's neural network to a minimal working version, on which they use feature visualization techniques to obtain a visual representation of the neural network's behavior. They additionally show that, for the case of SPECK32, there is no significant accuracy difference between the depth 1 neural network, and the depth 10 version.

4 Obstacles for Applying Neural Distinguishers Automatically

At AICrypt'23 [GLN23], Gohr, Leander, and Neumann presented an *assessment of differential-neural distinguishers*. In this work, they investigate whether *"machine learning [can be seen] as a tool assisting cryptanalysis, similar to how SAT and MILP solvers, among others, are seen by now"*. To successfully complement the work of a cryptanalyst, the approach needs to be *generic*, *i.e.*, it must not add significant workload for the cryptographer and reliably yield useful results.

Here, we identify the obstacles to such an automatic application of neural distinguishers to new primitives. Namely, there are obstacles in the architecture and hyperparameter choices of the neural distinguisher itself (Section 4.1), as well as obstacles in the identification of good input differences for new ciphers (Section 4.2). In Sections 5 and 6 we present our solutions to these obstacles.

4.1 Obstacle I: The Hyperparameters of Neural Distinguishers

The field of neural distinguishers being in its infancy, it is still unclear what machine learning architecture works best. Many peer-reviewed works [BBCD21, LLS+23, BGL+22, GLN23] have used (variations of) Gohr's network [Goh19b], from MLPs and CNNs [BBCD21] to significantly larger networks such as SENet [BGL+22], or combinations of hand-built features with non-neural classifiers in [BGPT21]. In the following, we first discuss to what extent automated hyperparameter tuning, as presented at AICrypt'23 [GLN23] can be used to obtain distinguishers for new primitives (Automated Hyperparameter Tuning). Then we discuss two particularly difficult to automatize steps (The Reshaping of the Input and The Training Pipelines) in more detail. We finalize our identified obstacles by discussing The Application to Large-state Ciphers.

Automated Hyperparameter Tuning. In their assessment of neural distinguishers the authors of [GLN23] conclude, that while the general idea of differential-neural cryptanalysis can be applied to a wide variety of ciphers, it is not clear that Gohr's network [Goh19b] is suitable for all ciphers. For the automated application of Gohr's network to other ciphers [GLN23] suggest automated hyperparameter tuning as one possibility. Out of twenty-two considered hyperparameters, they find that eight significantly impact the accuracy of the neural distinguisher for SPECK32/64 and SIMON32/64. These eight hyperparameters are automatically tuned to specialize Gohr's network [Goh19b] for other ciphers such as PRESENT. The obtained distinguishers using only automated hyperparameter tuning are presented in [GLN23, Table 5].

In addition to the automated hyperparameter tuning, [GLN23] points out two potential manual optimizations to improve the distinguisher: On one hand the cryptographers may find better input differences. On the other hand, they can choose a more elaborate training procedure such as *staged training*, see The Training Pipelines. The obtained distinguishers using additional manual optimization are presented in [GLN23, Table 1].

Table 2 compares the results of our work with the automated hyperparameter tuning and the additional manual optimizations of [GLN23]. Note, that our distinguishers (*right*) are most comparable to the automated hyperparameter tuning (*left*), in the sense that they don't require any manual intervention from the cryptographer. However, our distinguishers achieve with a simple, fully automated training procedure comparable accuracies to the ones obtained by [GLN23, Table 1] with additional manual optimization (*center*).

Our interpretation is that while optimizing Gohr's network for a new primitive using automated hyperparameter tuning is possible, our work achieves a higher degree of generalization and applicability to new primitives.

The Reshaping of the Input. Gohr's neural distinguisher's structure follows the division of SPECK into 2 words. However, when applying such a reshaping to different ciphers, the question arises of what data shape to adapt. For instance, for the AES cipher, a decomposition into $2 \cdot 16$ 8-bit words may be beaten by a $2 \cdot 4$ 32-bit columns, due to the column-oriented MixColumns operation of the cipher. Furthermore, the chosen shape has a direct influence on the complexity, and therefore learning power, of the network. This becomes clear when looking at Table 5, where ciphers with similar sizes, such as HIGHT, PRESENT, and SPECK64, result in neural classifiers with widely different complexities depending on their number of words (2 for SPECK64, 8 for HIGHT, 16 for PRESENT). For a higher number of words the Conv1D operation slices through a higher number of bits, compare Fig. 1 (center). This in turn means less necessary kernel shifts, and accordingly less multiply-accumulate operations, *i.e.,* FLOPs. While it is possible to try out many different input reshapings (manually or automated), we remove this potential obstacle by using a different rationale for the neural network design as presented in Section 6.2.

Table 2: Comparison of the best distinguishers for SIMON32/64, SPECK32/64, PRESENT, KATAN32, and CHACHA presented at AICrypt'23 [GLN23] using only automated hyperparameter tuning (*left*), additional manual optimization (*center*) and our work (*right*). The distinguishers are characterized by the highest round (*Max. Rounds*) in which their *Accuracy* is significantly above a random guess. The highest achieved number of rounds is highlighted.

	Automated hyperparameter tuning [GLN23, Table 5]		Elaborate training procedure [GLN23, Table 1]		Our work (w/o manual optimizations)	
Cipher	Max. Rounds	Accuracy	Max. Rounds	Accuracy	Max. Rounds	Accuracy
SIMON32/64	9	0.661	11	0.520[†]	11	0.516 (0.518[a])
SPECK32/64	7	0.617	8	0.514[†]	8	0.511 (0.514[a])
PRESENT	7	0.563	N/A	N/A	9	0.509
KATAN32	66	0.505	N/A	N/A	69	0.505

[†] [GLN23, Table 1] points out that "these results need a more elaborate training procedure; there is no known way to obtain them by simple variations of direct training."
[a] We can improve our results using a simple polishing pipeline as discussed in Section 6.1.

The Training Pipelines. When training a neural distinguisher, the highest achievable round may fail to be trained using straightforward techniques. For instance, to obtain an 8-round distinguisher for SPECK32, Gohr [Goh19b] uses a *staged training* scheme, where the best 5-round distinguisher is retrained on the input difference $(0x8000, 0x840a)$, (the most likely to appear after 3 rounds). This distinguisher is then retrained for 8 rounds, with 100 times more data than the other distinguishers, to finally reach 0.514 validation accuracy. Bao *et al.* [BGL+22], and [GLN23] use similar staged training procedures for their 10-round SIMON32 distinguisher. These elaborated training schemes are not easily automated, as they require looking at the differential characteristics of the studied cipher. We tackle this obstacle using our simple training pipeline presented in Section 6.1.

The Application to Large-state Ciphers. Gohr's neural network uses 32 filters per convolution layer, and 64 neurons for the first dense layer. These parameters match the size of the difference and of the input, respectively, for SPECK32. In order to generalize neural distinguishers to larger primitives, a logical first step is to upscale these parameters. Interestingly, [GLN23] does either not attempt to, or was not successful in the application of Gohr's original network to a larger state version of SPECK or SIMON. We manually –and unsuccessfully [3]– attempted the adaption of Gohr's network to SPECK128 and instead chose a more generic approach, resulting in the DBitNet network, presented in Section 6.2.

4.2 Obstacle II: Finding a Good Input Difference for a New Cipher

It has been shown in previous work [BGPT21] that the input difference to the best differential characteristic is, at least for SPECK, not a good choice for neural distinguishers.

In [Goh19b], a neural difference search algorithm is proposed[4], which successfully finds the input difference used in the SPECK32 distinguishers. However, adapting it to different ciphers is non-trivial[5]. We experimentally observe that Gohr's optimizer fails to find the

[3] We focused on SPECK128, with input difference $(0x80, 0x0)$, which propagates to $(0b100\cdots0, 0b100\cdots0)$ with probability 1 after 1 round. We varied the number of filters (32, 64 and 128) and neurons (64, 128, 256) of Gohr's RESNet, and obtained around 65% accuracy for 9 rounds with all the settings we tried. We conclude that scaling the parameters seems to have only had a limited impact on the final accuracy. At this point, we could either attempt to fine-tune the structure of the network further, or go with a more generic approach;

[4] Replicated as algorithm 2 in the appendix

[5] The starting round (3), number of iterations (2000), alpha parameter, the preprocessor's input reshaping, and learning rate schedule may need to be tuned. In order to minimize such tuning parameters, we focus on

optimal input difference for SPECK128, even after modifying it to encourage low Hamming weight differences. Furthermore, the evaluation speed for each difference prevents scaling for an efficient evaluation of a large number of differences. These observations motivate us to propose a more cryptographically inspired optimizer, rather than attempting to improve on Gohr's; this optimizer is presented in Subsection 5.2.

5 Solution Part I: Automated Finding of Good Input Differences

In the previous section, we identified generalization issues with the neural difference search algorithm. In this section, we propose a different, non-neural approach. Our solution consists of a bias score for fast ranking of input differences (Section 5.1), as well as an evolutionary optimizer (Section 5.2) which uses this new ranking scheme. The obtained results are presented in Section 5.3.

5.1 Bias Score for Ranking Input Differences

The input difference to the best n-round trail is not the one that gives the best results for neural distinguishers. For instance, for 5 rounds of SPECK, the input difference leading to the best trail is (0x2400, 0x0020), which leads to a trail with probability 2^{-9}; Gohr's network, trained with this input difference, reaches 61% accuracy. On the other hand, the input difference (0x0040, 0x0000) used in Gohr's paper does not have better 5 rounds trails than 2^{-13}, and yet, the neural network obtains 92% accuracy when trained with it. This disparity between the probability of the best trail and neural network accuracy becomes higher as the number of rounds increase: for 6 rounds, the neural network's accuracy does not go above 51% for the optimal input difference ((0x0211, 0xa040), 2^{-13} trail), but Gohr's input difference (2^{-20} for the best trail) reaches 78% accuracy.

We adopt the hypothesis proposed by [BGPT21] that this disparity is related to truncated differentials at rounds 3 and 4. In addition, we observe that the input difference (0x0040, 0x0000) fixes the 2 bits of the left part to 0 after 3 rounds. Furthermore, high biases persist in higher rounds; for instance, bit 14 at round 5, is set to 1 with probability 88%. We conjecture that Benamira et al.'s conclusions generalize to other ciphers, and that high biases in individual bits are a good approximation for the presence of high probability truncated differentials, which are otherwise difficult to find in a generic way. If this conjecture is correct, then highly biased difference bits at round r should lead to good neural distinguishers at round $r + \theta$ through differential-linear properties. Therefore, we assume a good input difference for neural distinguisher is one for which high biases exist in the difference bits of the higher rounds. This assumption is verified in our experiments, as the neural distinguishers we find usually cover several rounds past the highest round where a bias was detected.

We focus on the problem of finding the optimal input difference (for neural distinguishers) cryptographically, under the assumption that this input difference maximizes the bias of intermediate difference bits. More formally, we assume that a good input difference for neural distinguishers is one that maximizes a bias score, defined as:

Definition 1 (Exact bias score)**.** Let $E\colon \mathbb{F}_2^n \times \mathbb{F}_2^k \to \mathbb{F}_2^n$ be a block cipher, and $\delta \in \mathbb{F}_2^n$ be an input difference. The exact bias score for δ, $b(\delta)$ is the sum of the biases of each bit

SPECK128, simply adapting the word size in Gohr's code. We studied 3 cases: base, low Hamming weight preprocessor, and low Hamming weight preprocessor and optimizer starting-difference, each for 10 runs per starting round (from 1 to 7). The first two cases yielded random input differences, but the third case returned 3 input differences $((0x200000, 0x2000), (0x800000000, 0x80000000), (0x1000000000, 0x100000000))$ that resulted in 10-rounds distinguishers when retrained from scratch.

position j in the output difference, *i.e.*,

$$b(\delta) = \frac{1}{n} \cdot \sum_{j=0}^{n-1} \left| 0.5 - \frac{\sum\limits_{X \in \mathbb{F}_2^n, K \in \mathbb{F}_2^k - 1} (E_K(X) \oplus E_K(X \oplus \delta))_j}{2^{n+k}} \right|$$

The exact bias score cannot be computed for practical ciphers, as it requires enumerating all keys and plaintexts. On the other hand, we can use an approximation, obtained from a limited number of samples t:

Definition 2 (Bias score)**.** Let $E \colon \mathbb{F}_2^n \times \mathbb{F}_2^k \to \mathbb{F}_2^n$ be a block cipher, and $\delta \in \mathbb{F}_2^n$ be an input difference. The bias score for δ, $\tilde{b}^t(\delta)$ is the sum of the biases of each bit position j in the output difference, computed for t samples *i.e.*,

$$\tilde{b}^t(\delta) = \frac{1}{n} \cdot \sum_{j=0}^{n-1} \left| 0.5 - \frac{\sum\limits_{i=0}^{t-1} (E_{K_i}(X_i) \oplus E_{K_i}(X_i \oplus \delta))_j}{t} \right|$$

Conjecture 1. *Input differences δ that reach the most rounds with a neural distinguisher have a high bias score $b(\delta)$. We further assume that $\tilde{b}^t(\delta)$ is a good estimation of $b(\delta)$.*

To test our conjecture, we compute $\tilde{b}^t(\delta)$ for all 2^{32} possible SPECK32 input differences, for a small t; $\delta = (0x0040, 0x0000)$ does indeed maximize $\tilde{b}^t(\delta)$ for 5 rounds.

As a further test, we compute a bias score $\tilde{b}^{2000}(\delta)$ for low Hamming weight (1 and 2) input differences on SPECK128, and obtain $(0x80, 0x8000000000000000)$ as the optimal on 7, 8, 9 rounds. This input difference obtains vastly superior scores through the neural distinguisher, compared to the ones found by the neural difference search: 0.9861, 0.8252, and 0.5898 for 8, 9 and 10 rounds respectively.

These results convinced us to perform a search based not on the results of a linear classifier, but on the significantly faster to compute bias score, which allows us to explore more candidate input differences. To exploit the speed gain of our approach, we propose a new evolutionary-based search algorithm.

5.2 Evolutionary Optimizer

Algorithm Algorithm 1 starts from an initial population of random input differences, and improves the population iteratively by deriving new candidates from known ones (using a mutation probability M), ranking them through their bias score $\tilde{b}^t(\cdot)$, and allowing the best ones to move to the next generation. The algorithm stops if no input difference scores higher than a threshold T_b. In practice, the initial population contains 1024 differences, the 32 best ones are kept at each generation, and we set $M = 1$, $t = 10^4$, and $T_b = 0.01$.

Accounting for the Starting Round As mentioned in footnote 5 of Section 4.2, the round at which a difference is evaluated is an important parameter. As the most relevant round is not known in advance, we run our optimizer iteratively from round 1 to round $R + 1$, where no bias score above the threshold T_b is returned, obtaining R lists of 32 differences Δ_r for $r \in [1, R]$. Since the optimizer is heuristic, some good differences may have been identified in a subset of the rounds only; we therefore rerun the scoring procedure for the union of these lists, to obtain, for each difference δ_i, R bias scores $\tilde{b}_{i,r}$. The final score to return is subject to two main concerns: (1) the score in the highest round not to be a good indicator of the quality of the neural distinguisher, and (2) a simple sum of the scores at each round may favor less interesting differences; for instance, in the related-key case for SPECK with 4 key words, many differentials with probability 1 exist for the first 3 rounds,

Algorithm 1: Evolutionary optimizer

starting_population ← [RandomInt$(0, 2^n - 1)$ for 1024 times];
Sort starting_population by $\tilde{b}^t(\cdot)$ (descending order);
current_population ← first 32 elements of starting_population;
for *iterations* ← 0 **to** 50 **do**
  candidates ← [ ];
  for *i* ← 0 **to** 32 **do**
    for *j* ← $i + 1$ **to** 32 **do**
      if *RandomFloat*$(0, 1) < M$ **then**
        $m \leftarrow 1$
      else
        $m \leftarrow 0$
      end
      Add current_population$_i \oplus$ current_population$_j \oplus (m \ll$ RandomInt$(0, n - 1))$ to candidates
    end
  end
  Sort candidates by $\tilde{b}^t(\cdot)$ (descending order);
  current_population ← first 32 elements of candidates;
end
return candidates;

not all of which are interesting for further rounds. To address these concerns, we use a weighted score $S_{\delta_i} = \sum_{r=1}^{R} (\tilde{b}_{i,r})$, providing higher weight to later rounds while considering lower round scores. While certainly not optimal, this choice yields good results in practice.

5.3 Optimizer Results

Our optimizer returned a large number of solutions (Table 3). While most of these solutions are good, identifying the best one is difficult, as fully training a neural distinguisher for each would be prohibitively time-consuming. In some cases, such as SPECK128, one input difference is clearly dominating the others, and proves to result in the best neural distinguisher. On the other hand, in the case of SIMON32, 64, and 128, we respectively have 16, 32, and 64 input differences that obtain virtually identical scores (within 1% of each other), which is consistent with the observation of [KLT15] on the rotational equivalence of differentials. We therefore chose to use distance to the highest score as a metric to choose which differences to investigate: we define an input difference as ϵ-close to another if their score is within ϵ of each other. With $\epsilon = 0.1$, *i.e.*, looking only at input differences that obtained scores within 10% of the optimal, 185 differences need to be considered in total; an average of 15 differences per cipher need to be investigated using the neural distinguisher, with at least 4 rounds of training per difference.

5.4 Optimizer Discussion

The purpose of our tool is to rapidly evaluate a large number of relevant input differences. We do not claim its optimality, as other options could be chosen, both for the optimizer itself and the scoring function. In particular, the bound at which an input difference is considered non-random is not tight, so input differences resulting in small biases could be missed; this is not an issue here, as we want to capture large biases only.

We experimentally verified that, for random data, the bias score's average (over 10^4 samples per experiment, and 1000 experiments) is approximately 0.004. Increasing (respectively, decreasing) the number of samples moves the average closer to (respectively, further from) zero. The number of bits per sample only changes the standard deviation, from 0.00052 (32 bits) to 0.00015 (384 bits). The choice of 0.01 as a threshold value is far

enough from the tail of this distribution that it was never observed for non-relevant input differences. This choice is empirical, and gives good results in practice. Users wanting to investigate smaller biases may do so by setting a tighter threshold. Another interesting possibility is implementing a rigorous hypothesis testing procedure, replacing the bias score with a test statistic (or even multiple ones) of known distribution. This could be done, for example, with the "Frequency Test within a Block" of the NIST Statistical Test Suite [BRS+10].

While we did not experiment much around different values for the parameters, we find that reduced parameters, for instance $t = 10^3$, $T_b = 0.05$, and 10 generations, provide faster results despite being slightly less robust. Conversely, one might want to increase the minimum detectable bias by increasing t to 10^5 or more; however, we find that the performance of the optimizer degrades when t becomes too large, with no significant improvement in return.

Table 3: The total number of differences returned by our optimizer for each cipher, and the number of ϵ-close solutions for $\epsilon \in \{0.01, 0.1, 0.25\}$, where ϵ-close denotes differences for which the score differ at most by a factor ϵ to the optimal score.

Primitive	Total	0.01-close	0.1-close	0.25-close
SIMON32	135	16	16	16
SIMON64	145	32	32	32
SIMON128	266	64	64	64
SPECK32	81	1	2	2
SPECK64	69	1	2	2
SPECK128	156	1	1	1
LEA	156	1	2	2
HIGHT	140	3	27	27
TEA	73	1	3	3
XTEA	48	1	3	3
PRESENT	102	4	31	31
KATAN	334	1	2	10

6 Solution Part II: A Cipher-Agnostic Neural Training Pipeline

Based on the identified obstacles discussed in Section 4, we aim to overcome them by employing a streamlined training pipeline (Section 6.1) and creating a versatile neural network referred to as DBitNet (Section 6.2). We evaluate DBitNet's computational and memory requirements and compare them to the original ResNet proposed by Gohr and SENet.

6.1 Our Simple Training Pipeline

We propose a simplified pipeline to train a neural distinguisher for rounds R_s to R_f. The same network of R_s is retrained for round $R_f + 1$ until round R_f is reached. In SPECK32's case, one would train network N_5 for 5 rounds, retrain N_5 on the 6-round dataset to obtain N_6, retrain N_6 on 7 rounds to obtain N_7, and finally retrain N_7 on 8 rounds to obtain N_8. This technique is referred to as *our* simple training pipeline in this paper.

The Learning Rate Schedule. For the training of Gohr's neural distinguisher in [Goh19b] the Adam optimizer is used with a cyclic learning rate that varies over 10 epochs between limits of 0.002 and 0.0001. In [GLN23] these limits of the learning rate are optimized for each cipher in the automated hyperparameter tuning. In our simple pipeline for DBitNet, we will avoid a learning rate schedule, as well as any manual variation of the standard optimizer settings as follows: Adam is known as one of the most advanced optimizers,

however, it has been observed to fail to converge to an optimal solution [RKK19]. Such convergence failure may make it necessary to find an optimal learning rate schedule manually. For our purposes of a generic application to a range of new target ciphers, such a manual choice should be avoided. As an alternative to either the manual mitigation of the convergence issue or an automated hyperparameter tuning of the learning rate, Reddi *et al.* introduce the AMSGrad algorithm in *"On the Convergence of Adam and Beyond"* [RKK19] at ICLR 2018.

As a proof of concept, we ran this training pipeline with AMSGrad on SPECK32, using Gohr's neural network and input difference. With as little as 10 epochs per round, statistically significant (over 50.5% validation accuracy on 10^6 samples) 8 rounds distinguishers were obtained 10 times out of 10, whereas Gohr's initial experiments showed that no 8 rounds distinguisher could be learned without a complex training scheme. Removing either the pipeline or AMSGrad resulted in 8 rounds not being reached. In the remainder of our manuscript, we have used Gohr's original learning rate schedule to avoid sub-optimal results by changes on our side. For the interested reader, we provide a more detailed discussion of the fairness of our comparison DBitNet vs Gohr's ResNet in Appendix B.

A Simple Polishing Step. We can generally improve the accuracy of our distinguishers using our *simple polishing pipeline*, inspired by [Goh19b], where the final network is retrained 3 times, for 1 epoch, on 10^9 new training samples. At a batch size of 10,000, we use the Adam optimizer, decreasing the (constant) learning rate at each iteration, from 10^{-4} to 10^{-5} to 10^{-6}. The three learning rates, smaller than the Adam optimizer's default value of 10^{-3}, ensure the final convergence to an optimal solution for features that are not present in many batches. We have applied this simple polishing step only in two of the reported accuracies in this manuscript (for SIMON32 and SPECK32) as the large sample number makes it time-consuming. The basic pipeline above is sufficient to obtain competitive distinguishers that reach the same round as the state-of-the-art. The polishing step was only added to show that also some of the most elaborate and successful training pipelines can be replaced with our automated training pipeline. Five fresh datasets (with 10^6 samples in each) are generated for the final accuracy evaluation. The expected and observed standard deviation is 0.0005 as explained in the following.

The Random Guess Limit. The predictions of neural distinguishers can be modeled as binomial experiments with n trials, and two equiprobable outcomes, random or not random; in our case, $n = 10^7$ for training, and 10^6 for validation. The expected mean and standard deviation of a distinguisher making random prediction are $\mu = 0.5 \cdot n, \sigma = \sqrt{n/4}$, or, as a percentage, $\sigma_{\%} = 1/(2\sqrt{n})$. We consider the validation successful if the validation accuracy (percentage of correct guesses) exceeds ten standard deviations, *i.e.,* $A_{\text{not random}} > 50.5\%$.

6.2 Description of our Neural Network (DBitNet)

Gohr's neural distinguisher is immensely successful as a distinguisher for SPECK32. However, we identified a range of hyperparameters that need tuning for application to new ciphers in Section 4, the most important among them again being the input reshaping.

The input reshaping serves to investigate dependencies of far-apart as well as neighboring bits in the 64-bit input: For example, the bit-slicing filter may learn functions between bits (1, 17, 33, 49) while the following $k = 3$ filter may learn functions between neighboring bits (1,2,3) (compare Fig. 1 (center)). In this way, near and long-range dependencies among the bits can be learned. Therefore, the input reshaping can potentially be avoided, given another, more generic way to investigate near, as well as long-range dependencies.

Rationale for DBitNet. One way to tackle the problem of investigating near as well as long-range dependencies is so-called dilated convolutions, as presented in "Multi-Scale Context Aggregation by Dilated Convolutions" by Yu and Koltun [YK15]. The "Multi-Scale Context" refers to two-dimensional image data, however, a prominent example that uses dilated convolutions and deals with long-, as well as short-range dependencies on one-dimensional temporal data is WaveNet of Google DeepMind [ODZ+16].

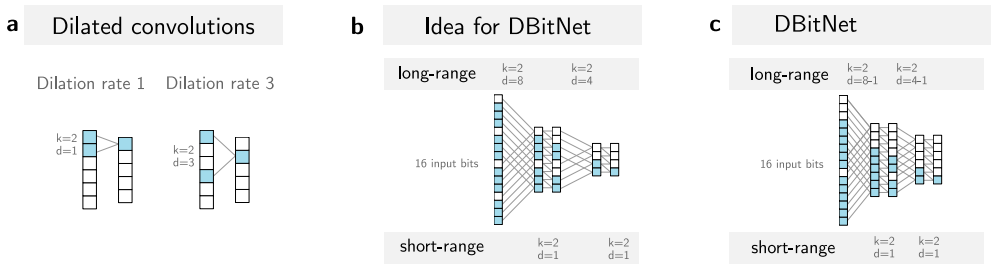

Figure 2: a) The concept of dilated convolutions, b) The idea for DBitNet c) The actual design of DBitNet.

A dilated convolution uses a dilation rate above one, Fig. 2a). Therefore, instead of learning a filter function between bits 1 and 2, a convolutional layer with dilation rate 3 can learn a filter function between bits 1 and 4. If we apply such a dilated convolutional layer with dilation $d = 8$ and kernel size $k = 2$ to a 16-bit input, we could find a representation with 8 neurons width which contains the information on the long-range dependencies between the bits of the first and the second half of the input, Fig. 2b). The next layer is a $d = 1$ layer to investigate the dependencies between neighboring bits. To investigate again the long-range dependencies, we next choose $d = 4$ and so on.

As shown in Fig. 2b) the neuronal width is shrinking with each dilated convolution by a factor of two. This shrinking of the neuronal width dimensionality is also encountered in popular image detection networks like ResNet [HZRS15]. As "compensation" the number of channels is increased: In ResNet34 for example the image size is halved from 224 pixels to 112, to 56, to 28 pixels, and so on while the number of channels increases from the 3 red-green-blue channels to 64, and 128. We follow a similar tactic and increase the number of channels with each dilational convolution. We start with 32 filters, identical to Gohr, in the first convolutional layer. Whenever the neuronal width is halved, we add 16 filters, resulting in $32 + i \times 16$ filters in the ith dilated convolution.

Table 4: Settings for Gohr's neural network and DBitNet.

		Gohr settings		**DBitNet settings**
cipher	input size	num. blocks	word size	dilation rates
SIMON32	64	2	16	[31, 15, 7, 3]
SPECK32	64	2	16	[31, 15, 7, 3]
KATAN	64	2	32	[31, 15, 7, 3]
HIGHT	128	8	8	[63, 31, 15, 7, 3]
PRESENT	128	16	4	[63, 31, 15, 7, 3]
SIMON64	128	2	32	[63, 31, 15, 7, 3]
SPECK64	128	2	32	[63, 31, 15, 7, 3]
TEA	128	2	32	[63, 31, 15, 7, 3]
XTEA	128	2	32	[63, 31, 15, 7, 3]
LEA	256	4	32	[127, 63, 31, 15, 7, 3]
SIMON128	256	2	64	[127, 63, 31, 15, 7, 3]
SPECK128	256	2	64	[127, 63, 31, 15, 7, 3]
GIMLI	768	12	32	[383, 191, 95, 47, 23, 11, 5]

Neural Network Settings for Different Ciphers. When working on a different cipher many model and training parameters and hyperparameters might need to be adapted. At the minimum, and common to Gohr's neural distinguisher and DBitNet, the neural network input size has to be adapted when changing to a cipher of different sizes. Based on this input size, for DBitNet, the dilation rates are automatically determined by dividing the input size by two and subtracting one, until a minimum value of 3 is reached. Gohr's network requires manual input for the number of words (Section 4.1). Gohr uses a prediction head with two dense layers (64, 64 neurons in each layer). For DBitNet we have considered scaling the prediction head with the input size. Finally, however, we have instead chosen a slightly more powerful prediction head with three dense layers (256, 256, 64 neurons in each layer), which is the same, regardless of the input size. For Gohr's neural distinguisher also the number of filters, as well as the cyclic learning rate, might have to be adapted. However, in our experiments, we will use the same number of filters and cyclic learning rate as in Gohr's original experiments [Goh19b]. For DBitNet we restrict ourselves to using the ADAM optimizer in its standard settings, together with the before-mentioned AMSGrad algorithm. The settings for both neural networks are summarized in Table 4. For the interested reader, we provide a more detailed discussion of the fairness of our comparison DBitNet vs Gohr's ResNet in Appendix B.

Table 5: FLOPs, parameters, and runtime per epoch (on our NVidia Ampere A100 GPU) for Gohr's neural distinguisher of depth 1 (D1), depth 10 (D10), and DBitNet.

	FLOPs			**Parameter counts**			**Time per epoch**	
cipher	Gohr-D1	DBitNet	Gohr-D10	Gohr-D1	DBitNet	Gohr-D10	Gohr-D1	DBitNet
SIMON32	0.28M	1.76M	2.09M	44.32k	298.11k	102.50k	10s	36s
SPECK32	0.28M	1.76M	2.09M	44.32k	298.11k	102.50k	10s	36s
HIGHT	0.15M	3.52M	1.06M	28.32k	390.21k	86.50k	9s	68s
PRESENT	0.09M	3.52M	0.54M	20.64k	390.21k	78.82k	9s	68s
SIMON64	0.55M	3.52M	4.16M	77.09k	390.21k	135.26k	14s	64s
SPECK64	0.55M	3.52M	4.16M	77.09k	390.21k	135.26k	14s	68s
TEA	0.55M	3.52M	4.16M	77.09k	390.21k	135.26k	15s	68s
XTEA	0.55M	3.52M	4.16M	77.09k	390.21k	135.26k	14s	68s
LEA	0.56M	7.17M	4.17M	77.22k	503.46k	135.39k	15s	129s
SIMON128	1.10M	7.17M	8.31M	142.62k	503.46k	200.80k	22s	116s
SPECK128	1.10M	7.17M	8.31M	142.62k	503.46k	200.80k	24s	129s
GIMLI	0.59M	20.37M	4.20M	77.73k	705.44k	135.91k	16s	312s

A Comparison of FLOPs and Parameter Counts. The number of multiply-add operations, or FLOPs, is often used as a proxy for the latency and memory usage of neural network models [BOFG20]. We use the TensorFlow Keras module `keras-flops` to evaluate the number of FLOPs for each model. TensorFlow provides a native routine `model.count_params()` for the parameter count. The results are shown in Table 5. For the 32-bit ciphers, the execution time of DBitNet is in between the one for Gohr-depth1 (10s) and Gohr-depth10 (50s, not shown in the table). The same holds for the number of FLOPs. The FLOPs and time per epoch for DBitNet scale linearly with the input size of the cipher. Since the FLOPs represent the operations needed to investigate a cipher, an increase of the FLOPs with the size of the cipher is reasonable. To achieve such an increase in the FLOPs, the number of filters of Gohr's network would have to be manually adapted, depending on the input size, as well as the chosen number of blocks and word size. We have also analyzed the neural distinguisher SENet $\mathcal{ND}_{VV}^{\text{SIMON}_{8R}}$ provided on the GitHub repository of [BGL+22] for SIMON32 and find that it has 13.5M FLOPs, and 449.46k parameters.

7 Results: Our Best Distinguishers

For each target cipher in Table 6 we start with the set of differences found by the evolutionary optimizer presented in Section 5.3. We train a Gohr depth-1 neural network and DBitNet to distinguish between ciphertext pairs of the chosen plaintext difference, and those of random plaintext pairs using the training pipeline as presented in Section 6.1. Table 6 summarizes the highest round achieved (*best round*), as well as the accuracy (*best acc.*) of the best distinguisher (*best NN*) in this round, once for our *simple training pipeline* with only 10 epochs in each round, and once for our *simple training pipeline* with 40 epochs in each round. The green highlight indicates an improvement of the 40 epochs over the 10 epochs training pipeline.

Table 6: Summary of the best distinguishers for each target cipher for our *simple training pipeline* with 10 epochs per round, respectively 40 epochs per round. Shown is the mean accuracy of five evaluations on freshly generated test datasets, each containing 10^6 samples. The detailed round-by-round results (including the TPR and TNR) for 40 epochs are shown in Table 8.

		10 epochs			40 epochs		
cipher	difference	best round	best acc.	best NN	best round	best acc.	best NN
SIMON32	0x400	11	0.5147	DBitNet	11	0.5160 [0.5179[P]]	DBitNet
SPECK32	0x400000	8	0.5096	DBitNet	8	0.5109 [0.5144[P]]	DBitNet
HIGHT	0x800000000000	10	0.7511	DBitNet	10	0.7511	DBitNet
HIGHT[RK]	0x800000000...[(a)]	13	0.9648	DBitNet	14	0.5627	DBitNet
KATAN	0x4000	67	0.5064	DBitNet	69 [71[*P]]	0.5046	DBitNet
PRESENT	0xd000000	8	0.5546	DBitNet	9	0.5092	DBitNet
SIMON64	0x8000	13	0.5180	DBitNet	13	0.5179	DBitNet
SPECK64	0x8080000000	8	0.5335	Gohr-D1	8	0.5366	DBitNet
TEA	0x4000000000000000	5	0.5573	Gohr-D1	5	0.5634	DBitNet
XTEA	0x4000000000000000	5	0.5307	DBitNet	5	0.5984	DBitNet
LEA	0x800000008...[(b)]	11	0.5104	DBitNet	11	0.5109	DBitNet
SIMON128	0x8000000	20	0.5057	DBitNet	20	0.5055	DBitNet
SPECK128	0x80800000000000000	10	0.5928	DBitNet	10	0.5916	DBitNet
GIMLI	0x800000000...[(c)]	11	0.5261	DBitNet	11	0.5270	DBitNet

[P] Automated pipeline result after our *simple polishing pipeline* from section Section 6.1
[*P] We note that for KATAN a 71-round distinguisher with 0.5034 ± 0.0002 accuracy on five freshly generated validation datasets can be obtained by using our *simple polishing pipeline* on the 71-round distinguisher from Table 8.
[RK] Related key setting
[(a)] 0x80000000000000000000000000000000000000000800000
[(b)] 0x80000008000000008004000080
[(c)] 0x800000000000000000000000000004000000000000000000000000000000008000000000000000000000000

SIMON and SPECK For SPECK32, we retrieve the optimal input difference used in Gohr's paper. DBitNet, trained using our *simple training pipeline*, reaches 8 rounds with over 51% accuracy, which was deemed to only be possible with an advanced staged approach [GLN23]. The accuracy is improved to match [Goh19b] with our *simple polishing pipeline*. For SPECK64, we reach 8 rounds with accuracy 0.5366, only 10% less than [HRCF21], which uses 128 pairs. For SPECK128, we obtain the first 10-round neural distinguisher, with accuracy 0.5916. Interestingly, the best differential characteristic for SPECK128 given in [SHY16] contains one of the differences returned by our optimizer at round 15: $(0x80, 0)$. When training DBitNet for this input difference, we get respective accuracies of 0.9057, 0.6507, and 0.5258 for 8, 9 and 10 rounds, therefore obtaining candidate theoretical distinguishers for 23, 24 and 25 rounds respectively. However, the signal-to-noise ratio of these distinguishers does not permit direct application: the probability for the front 15 rounds is 2^{-110}, and the evaluation of $C \cdot 2^{-110}$ produces too many false positives for C true positives to be distinguishable.

For a key recovery attack similar to [BGL+22], one can prepend the input difference $(0x820200, 0x1202)$, which propagates to our best neural distinguisher given by $(0x80, 0x8000000000000000)$ after 2 rounds with probability 2^{-6}. An additional round can be added at the start, yielding a 13 rounds distinguisher.

For SIMON32, we obtain similar results to [BGL+22], albeit with a significantly simpler training pipeline, and less computations (Section 6.2). For SIMON64, we reach one more round than [HRCF21], even though [HRCF21] uses 64 pairs. On the other hand, Lu [LLS+23] reaches one more round for SIMON32 and SIMON64. It is important to note that their training pipeline is fully dedicated to SIMON, with advanced feature engineering and 8 pairs per sample, therefore showing that a specialized method for a given cipher does outperform the generic approach in some cases. Lu proposes a few input differences for 12 rounds in table 3 of [LLS+23]: these differences appear in our results, but were not investigated. For instance, $(0x10004)$ ranks 21st in the returned solutions. For SIMON128, we find a new 20 rounds distinguisher, with an accuracy of 0.5057.

GIMLI For the GIMLI permutation, our 11-round accuracy has an accuracy of 0.527, to be compared to the 8 rounds neural distinguisher of [BBCD21]. This result highlights the need for an automatic tool to find good input differences, as we obtained similar results to [BBCD21] when using the same input differences as them. In comparison, the design document of GIMLI [BKL+17], mentions at best a differential characteristic with probability 2^{-188} on 12 rounds, and a 12-round linear distinguisher with complexity 2^{-198} and 15-round differential-linear distinguisher with complexity $2^{-87.4}$ are presented in [FGLNP+20]. The full-round symmetry distinguishers [FGLNP+20] remain stronger.

HIGHT We obtain the first published neural distinguisher for HIGHT, covering 10 rounds with accuracy 0.751. In addition, we ran our pipeline in the related-key setting as a proof of concept, and obtained a 14 rounds related-key distinguisher with accuracy 0.562. In comparison, the paper presenting HIGHT [HSH+06] mentions a probability 1 10 rounds property: if the input difference has a given form, then the leftmost byte of the output difference is non-zero. This property would require $C \cdot 256$ (with C a small constant) to distinguish. On the other hand, our neural distinguisher requires a single pair.

PRESENT For PRESENT, we find a 9-round distinguisher with an accuracy of 0.5092, which favorably compares to the 7-round distinguishers of [GLN23] and [CSYY22], despite [CSYY22] using 8 pairs. In comparison, the best differential characteristic for PRESENT reduced to 9 rounds has probability 2^{-36} [Wan07].

KATAN For KATAN, our distinguisher reaches statistically significant accuracies up to 69 rounds, compared with [GLN23]'s 66 rounds, even though [GLN23]'s distinguishers use advanced feature engineering (inversion of the last 4 rounds). In contrast, [LCLH22] reaches 51 rounds in the standard setting, and 59 when using 64 pairs. The same paper proposes distinguishers up to 85 rounds in the single key model, using additional conditions on the plaintexts, which is out of the scope of our study. We note that we obtain a 71-round distinguisher with 0.5034 ± 0.0002 accuracy using our *simple polishing pipeline*.

TEA and XTEA For both TEA and XTEA, we find distinguishers for 5 cycles (10 rounds), respectively with accuracies 0.5634 and 0.5984; interestingly, they share the same input difference. For TEA, we reach 2 more rounds than [BR21].

LEA For LEA, we propose the first neural distinguisher, reaching 11 rounds with accuracy 0.5109. In comparison, [HLK+14] presents a differential characteristic with probability 2^{-98} for 11 rounds, and 2^{-128} for 12 rounds.

A Sanity Check: The Case of Related-Key TEA The block cipher TEA is known to have equivalent keys. From an initial key k_0, k_1, k_2, k_3, the core of the round function, updating the two halves of the state v_0 and v_1, is:

$$v_0 = v_0 \boxplus ((v_1 << 4) \boxplus k_0) \oplus (v_1 \boxplus sum) \oplus ((v_1 >> 5) \boxplus k_1) \quad (1)$$
$$v_1 = v_1 \boxplus ((v_0 << 4) \boxplus k_2) \oplus (v_0 \boxplus sum) \oplus ((v_0 >> 5) \boxplus k_3) \quad (2)$$

Differences in the most significant bits of k_0 and k_1, and of k_2 and k_3, cancel out, resulting in 3 equivalent keys for each possible key. In the related key mode, our optimizer finds the property that differences in the most significant bits of 2 words of the key result in a maximal bias score (as the ciphertexts are equal). The corresponding input differences are found by the genetic optimizer within the first few generations.

The ability of our framework to detect such properties reassures us in its ability to support the block cipher design process, by identifying trivial weaknesses easily.

8 Discussion

Scope of Our Work In this paper, we focus on automatically finding basic neural distinguishers. If we consider an analogy with differential cryptanalysis, cryptographers traditionally begin with an automatic tool to obtain good differential characteristics for as many rounds as possible. From these characteristics, the cryptographer may then attempt to derive the probability of the best differentials, or combine them into more advanced attacks such as boomerang attacks. We identify this second step to specializing through feature engineering, prepended rounds, neutral bits, etc. Our focus is on the equivalent of the first step: building blocks that can further be refined into an attack.

In this respect, the neural distinguishers we propose are competitive with related work using a comparable setting ($2 - 1 - *-$R). We even sometimes improve on specialized approaches with features engineering, *e.g.,* [BGL+22], or multiple pairs [CSYY22], using a fully automatic and generic pipeline.

Extending the Scope For the sake of completeness, we give the intuition on how to extend our pipeline to include key recovery considerations.

In order to include prepended rounds, the optimizer can be modified to additionally decrypt each pair (P_0, P_1) used to compute the bias score of a difference δ, for i rounds; the number of occurrences of the most frequent decryption differences gives an approximation of the probability of the best prepended differential. This estimation, along with i and the bias score, can be combined into a composite score to obtain a longer differential-ML distinguisher. Preliminary experiments show that this approach retrieves $(0x2110a04)$, used to prepend 2 rounds in Gohr's key recovery [Goh19b]. Alternatively, one may use the fact that our optimizer returns a parametrizable number of input differences, and, for each of these, compute how many rounds can be prepended (e.g., through MILP) and how many rounds a neural distinguisher can cover (by training it). Further improvements, *e.g.*, the use of neutral bits, can be included, for instance by running the generalized neutral bit search algorithm presented in [BGL+22] to each returned difference. Advanced feature engineering can also readily be applied, as DBitNet is generic in its input size and format.

Extending Basic Neural Distinguishers: Comparability Specializing a neural distinguisher, through prepending probabilistic rounds, using feature engineering, multiple pairs, or neutral bit-based analysis improves the key recovery abilities, at the cost of comparability. It may occur that a different neural distinguisher could be plugged into the attack, and

yield better results, but it is challenging to say without the authors giving the baseline results in the $2-1-*-R$ setting, to promote comparability.

For instance, [YK21] exhibits a $2^{20}-1-\delta-R$ 9-round distinguisher for SPECK32, using a 3-rounds neural distinguisher and 6 probabilistic prepended rounds, and claims to improve over [Goh19b]. In contrast, [Goh19b] uses a 9-rounds distinguisher, built from a 7-rounds neural distinguisher and 2 probabilistic rounds, to recover the full key of 11-rounds SPECK with $2^{14.5}$ ciphertexts, which is significantly better.

Intended Use of Our Tool The uses of our tools are twofold. On the one hand, cipher designers can use it to obtain bounds for a given set of parameters rapidly. On the other hand, neural cryptanalysis researchers can use our tool to obtain a baseline to compare to any new cipher they wish to study, without having to fine-tune any parameters, due to its plug-an-play approach. Furthermore, our tool can be used out-of-the-box to perform neural analysis on any cipher, even though we limited ourselves to a few, and did not include related-key results besides HIGHT and TEA (as proofs-of-concept), due to the mere amount of GPU-extensive experiments to run, and we believe it can match or improve upon other published results without further tuning.

Estimated Runtime The pipeline for a new cipher is composed of the optimizer (fast) and the neural network training (slow). The total runtime, between a few hours and a few days, depends on the number of differences (Table 3), the number of rounds to study, and the size of the cipher. The most time-consuming part is the neural network training, the time for which can be estimated from Table 5. We note that the reduced parameters set used in the repository yields decent result significantly faster; further speedup can be achieved through pre-filtering, by training all the differences for a small number of epochs (e.g., 5) to select which ones to investigate further.

Comparison with Brute Force Search Here, we compare our optimizer with a brute-force search over low Hamming Weight (HW) differences, ranked by their bias score. For a cipher with block size n, and $b-$bit input differences, this brute-force search would explore $\sum_{k=1}^{b}\binom{n}{k}$ differences, which is 43744 for PRESENT, and almost $10M$ for GIMLI, with having HW 3 optimals. Furthermore, enumerating all input differences up to HW 3 says nothing about higher HW differences; for instance, in the case of LEA, we find a HW 5 optimal difference. In comparison, our optimizer explores at most 24800 differences ($\sum_{k=1}^{31}=496$ per generation, over 50 generations). We expect this scalability advantage to become even more important as the search space grows, e.g., for related-key.

9 Conclusion

We tackled the problem of generalizing neural distinguishers with a framework that can be applied out of the box to any cipher. This framework relies on a generic neural network structure powered by dilated convolutional layers, as well as generic choices of parameters such as the learning rate. In addition, we resolved the challenge of automatically choosing a good input difference for a variety of ciphers through an evolutionary optimizer.

We experimentally showed that our framework often matches or beats state-of-the-art neural distinguishers and finds good ones for not yet studied primitives.

Preliminary experiments show that our framework finds good input differences also in the related-key setting, but their exploitation requires significant effort and is left for future work. This study produced a large number of input differences with good properties for neural distinguishers. It seems promising to explore how these can be combined into more powerful multiple-input differences distinguishers to improve existing results. It remains challenging to investigate the whole list of returned differences.

References

[BBCD21] Anubhab Baksi, Jakub Breier, Yi Chen, and Xiaoyang Dong. Machine learning assisted differential distinguishers for lightweight ciphers. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2021, Grenoble, France, February 1-5, 2021*, pages 176–181. IEEE, 2021.

[BBP22] Nicoleta-Norica Bacuieti, 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 - 20th International Conference, ACNS 2022, Rome, Italy, June 20-23, 2022, Proceedings*, volume 13269 of *Lecture Notes in Computer Science*, pages 809–829. Springer, 2022.

[BGG+23] Emanuele Bellini, David Gerault, Juan Grados, Yun Ju Huang, Mohamed Rachidi, and Sharwan Tiwari. Claasp: a cryptographic library for the automated analysis of symmetric primitives. Cryptology ePrint Archive, Paper 2023/622, 2023. https://eprint.iacr.org/2023/622.

[BGL+22] Zhenzhen Bao, Jian Guo, Meicheng Liu, Li Ma, and Yi Tu. Enhancing differential-neural cryptanalysis. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part I*, volume 13791 of *Lecture Notes in Computer Science*, pages 318–347. Springer, 2022.

[BGPT21] Adrien Benamira, David Gérault, 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 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 805–835. Springer, 2021.

[BKL+07] 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.

[BKL+17] Daniel J. Bernstein, Stefan Kölbl, Stefan Lucks, Pedro Maat Costa Massolino, Florian Mendel, Kashif Nawaz, Tobias Schneider, Peter Schwabe, François-Xavier Standaert, Yosuke Todo, and Benoît Viguier. Gimli : A cross-platform permutation. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, pages 299–320, Cham, 2017. Springer International Publishing.

[BOFG20] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the State of Neural Network Pruning? In *Proceedings of the 3rd MLSys Conference*, Austin, TX, USA, mar 2020.

[BR21] 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.

[BRS+10] Lawrence E. Bassham, Andrew L. Rukhin, Juan Soto, James R. Nechvatal, Miles E. Smid, Elaine B. Barker, Stefan D. Leigh, Mark Levenson, Mark Vangel, David L. Banks, Nathanael Alan Heckert, James F. Dray, and San Vo. SP 800-22 Rev. 1a. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. Technical report, Gaithersburg, MD, USA, 2010.

[BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. *J. Cryptology*, 4:3–72, 1991.

[BTCS+15] Ray Beaulieu, Stefan Treatman-Clark, Douglas Shors, Bryan Weeks, Jason Smith, and Louis Wingers. The simon and speck lightweight block ciphers. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2015.

[CSYY22] Yi Chen, Yantian Shen, Hongbo Yu, and Sitong Yuan. A New Neural Distinguisher Considering Features Derived From Multiple Ciphertext Pairs. *The Computer Journal*, 03 2022. bxac019.

[DCDK09] 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.

[FGLNP+20] Antonio Flórez Gutiérrez, Gaëtan Leurent, María Naya-Plasencia, Léo Perrin, André Schrottenloher, and Ferdinand Sibleyras. New results on gimli: Full-permutation distinguishers and improved collisions. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020*, pages 33–63, Cham, 2020. Springer International Publishing.

[GBC17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning: The MIT Press*, volume 19. The MIT Press, 2017.

[GLN23] Aron Gohr, Gregor Leander, and Patrick Neumann. An assessment of differential-neural distinguishers. In *AICrypt'23 - 3RD Workshop on Artificial Intelligence and Cryptography*, 2023.

[Goh19a] Aron Gohr. Deep speck. https://github.com/agohr/deep_speck, 2019.

[Goh19b] 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.

[HLK+14] Deukjo Hong, Jung-Keun Lee, Dong-Chan Kim, Daesung Kwon, Kwon Ho Ryu, and Dong-Geon Lee. Lea: A 128-bit block cipher for fast encryption on common processors. In Yongdae Kim, Heejo Lee, and Adrian Perrig, editors, *Information Security Applications*, pages 3–27, Cham, 2014. Springer International Publishing.

[HRCF21] ZeZhou Hou, JiongJiong Ren, ShaoZhen Chen, and AnMin Fu. Improve Neural Distinguishers of SIMON and SPECK. *Sec. and Commun. Netw.*, 2021, jan 2021.

[HSH+06] Deukjo Hong, Jaechul Sung, Seokhie Hong, Jongin Lim, Sangjin Lee, Bon-Seok Koo, Changhoon Lee, Donghoon Chang, Jesang Lee, Kitae Jeong, Hyun Kim, Jongsung Kim, and Seongtaek Chee. Hight: A new block cipher suitable for low-resource device. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006*, pages 46–59, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 770–778, dec 2015.

[KLT15] Stefan Kölbl, Gregor Leander, and Tyge Tiessen. Observations on the simon block cipher family. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology – CRYPTO 2015*, pages 161–185, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[Knu98] Lars Knudsen. Deal-a 128-bit block cipher. *Complexity*, 258(2), 1998.

[KW02] Lars Knudsen and David Wagner. Integral cryptanalysis. In Joan Daemen and Vincent Rijmen, editors, *Fast Software Encryption*, pages 112–127, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[LCLH22] Dongdong Lin, Shaozhen Chen, Manman Li, and Zezhou Hou. The construction and application of (related-key) conditional differential neural distinguishers on katan. In Alastair R. Beresford, Arpita Patra, and Emanuele Bellini, editors, *Cryptology and Network Security*, pages 203–224, Cham, 2022. Springer International Publishing.

[LDLS21] Luc Libralesso, François Delobel, Pascal Lafourcade, and Christine Solnon. Automatic Generation of Declarative Models For Differential Cryptanalysis. In Laurent D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021*, volume 210 of *LIPIcs*, pages 40:1–40:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[LLS+23] Jinyu Lu, Guoqiang Liu, Bing Sun, Chao Li, and Li Liu. Improved (Related-Key) Differential-Based Neural Distinguishers for SIMON and SIMECK Block Ciphers. *The Computer Journal*, 01 2023. bxac195.

[Mat94] Mitsuru Matsui. On correlation between the order of s-boxes and the strength of DES. In Alfredo De Santis, editor, *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings*, volume 950 of *Lecture Notes in Computer Science*, pages 366–375. Springer, 1994.

[MWGP12] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and linear cryptanalysis using mixed-integer linear programming. In Chuan-Kun Wu, Moti Yung, and Dongdai Lin, editors, *Information Security and Cryptology*, pages 57–76, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[ODZ+16] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[RKK19] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

[RR22] Adrián Ranea and Vincent Rijmen. Characteristic automated search of cryptographic algorithms for distinguishing attacks (CASCADA). *IET Inf. Secur.*, 16(6):470–481, 2022.

[Sam59] Arthur L Samuel. Machine learning. *The Technology Review*, 62(1):42–45, 1959.

[SHY16] Ling Song, Zhangjie Huang, and Qianqian Yang. Automatic differential analysis of arx block ciphers with application to speck and lea. In Joseph K. Liu and Ron Steinfeld, editors, *Information Security and Privacy*, pages 379–394, Cham, 2016. Springer International Publishing.

[SZM20] Heng-Chuan Su, Xuan-Yong Zhu, and Duan Ming. Polytopic attack on round-reduced simon32/64 using deep learning. In *Information Security and Cryptology: 16th International Conference, Inscrypt 2020, Guangzhou, China, December 11–14, 2020, Revised Selected Papers*, page 3–20, Berlin, Heidelberg, 2020. Springer-Verlag.

[Wan07] Meiqin Wang. Differential cryptanalysis of present. *IACR Cryptol. ePrint Arch.*, 2007:408, 2007.

[WN95] David J. Wheeler and Roger M. Needham. Tea, a tiny encryption algorithm. In Bart Preneel, editor, *Fast Software Encryption*, pages 363–366, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[WN97] David J. Wheeler and Roger M. Needham. Tea extensions, 1997.

[YK15] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[YK21] Tarun Yadav and Manoj Kumar. Differential-ml distinguisher: Machine learning based generic extension for differential cryptanalysis. In *Progress in Cryptology – LATINCRYPT 2021: 7th International Conference on Cryptology and Information Security in Latin America, Bogotá, Colombia, October 6–8, 2021, Proceedings*, page 191–212, Berlin, Heidelberg, 2021. Springer-Verlag.

A Pseudocode for Gohr's optimizer

Algorithm 2: Gohr's optimizer: given a function $F : \{0,1\}^b \to R$, greedily optimizes it to find an input x that maximizes F. Requires in input the number of iterations t and an exploration factor α.

$x \leftarrow \mathsf{Random}(0, 2^b - 1)$;
$v_{best} \leftarrow F(x)$;
$x_{best} \leftarrow x$;
$v \leftarrow v_{best}$;
$H \leftarrow$ hashtable with default 0;
$i \leftarrow 0$;
while $i < t$ **do**
  $H(x) \leftarrow H(x) + 1$;
  $r \leftarrow \mathsf{Random}(0, b - 1)$;
  $x_{new} \leftarrow x \oplus (1 \ll r)$;
  $v_{new} \leftarrow F(x_{new})$;
  if $v_{new} - \alpha \log_2(H(x_{new})) > v - \alpha \log_2(H(x))$ **then**
    $v \leftarrow v_{new}$;
    $x \leftarrow x_{new}$;
  end
  if $v_{new} > v_{best}$ **then**
    $v_{best} \leftarrow v$;
    $x_{best} \leftarrow x$;
  end
  $i \leftarrow i + 1$;
end
return x_{best};

B Discussion of the Comparison of DBitNet and Gohr's Neural Distinguisher

It is not obvious how to fairly compare DBitNet and Gohr's ResNet. Should we compare our DBitNet to the depth-1 version or to the depth-10 version? Should we use the original cyclic learning rate schedule, which was optimized for Gohr's ResNet, but which might be particular to SPECK32, or should we instead use the AMSGrad learning rate as for DBitNet? Should we use a larger prediction head, such as in DBitNet (see Neural Network Settings for Different Ciphers), or leave the prediction head in its original state? Here, note that adding more parameters can actually decrease the learning performance of a neural network, since it takes more training epochs to fit all of them. Should we adapt the number of filters for Gohr's Neural Distinguisher? Again, we should consider that an increase in parameters can lead to a decreased learning performance. How many settings for the number of blocks, and the word size should we try out?

Many of these questions tie into the discussion provided in Obstacle I: The Hyperparameters of Neural Distinguishers which motivated us to create DBitNet in the first place.

Overall, we think that our presented comparison of Gohr's ResNet with DBitNet is fair for two reasons:

1. On the one hand, our main table Table 1 compares our generic DBitNet with the highly optimized versions of Gohr's ResNet for each cipher.

2. On the other hand, the following preliminary experiments motivate the version of Gohr's network we used for our comparisons presented in Table 6 and Table 8.

Table 7: Preliminary experiments with different versions of Gohr's ResNet and DBitNet. The highest round with a validation accuracy above 0.505 is highlighted. These experiments were performed on SIMON32, single-key, 0x400, starting round 8. We have used our Our Simple Training Pipeline with 40 epochs in each round for various versions of Gohr's ResNet and our DBitNet. Shown are two runs for each network to account for potential unfortunate weight initializations at the beginning of the training.

round	DBitNet	Gohr Cyc. D1	Gohr Cyc. D10	Gohr AMS D1	Gohr AMS D10	Gohr Big-Prd.
8	0.8335|0.8312	0.7585|0.7561	0.7478|0.723	0.748|0.7458	0.7559|0.7505	0.8305|0.8299
9	0.656|0.6559	0.6269|0.6241	0.6085|0.6081	0.6227|0.6211	0.6189|0.6186	0.6466|0.6448
10	0.5599|0.5616	0.5351|0.5009	0.5411|0.5006	0.5547|0.5545	0.5536|0.5413	0.5008|0.5007
11	0.5164|0.5166	0.5004	0.5005	0.5027|0.5014	0.5033|0.5011	

The different versions of Gohr's ResNet and DBitNet have details as follows:
DBitNet: As described in Subsection 6.2.
Gohr Cyc. D1: Gohr's depth 1 network with the cyclic learning rate schedule of [Goh19b].
Gohr Cyc. D10: Gohr's depth 10 network with the cyclic learning rate schedule of [Goh19b].
Gohr AMS D1: Gohr's depth 1 network with AMSGrad.
Gohr AMS D10: Gohr's depth 10 network with AMSGrad.
Gohr Big-Prd.: Gohr's network with the larger prediction head of DBitNet and depth 1.

In Table 7 we present preliminary experiments on SIMON32 with various versions of Gohr's ResNet and our DBitNet. AMSGrad seems an overall better choice than the cyclic learning rate schedule. The effect is, however, not large enough to increase the accuracies to similar values as obtained by DBitNet. There is no benefit to using a more powerful prediction head for Gohr's ResNet, actually, it decreases the obtained accuracy. In conclusion, we do not find an improvement large enough to justify the manipulation of Gohr's ResNet (by using AMSGrad or a different prediction head).

C Round-by-round Details of our Results

The following Table 8 and Table 9 give the detailed round-by-round validation accuracies, TPR, and TNR for the results summarized in Table 6.

Table 8: Detailed round-by-round validation accuracy results, as well as the TPR, and TNR for all target ciphers –except KATAN, see Table 9– from Table 6.

cipher	round	Gohr depth-1		DBitNet		Gohr TPR|TNR		DBitNet TPR|TNR	
		(1)	(2)	(1)	(2)	(1)	(2)	(1)	(2)
SIMON32	8	0.7400	0.7823	0.8335	0.8312	0.70|0.78	0.77|0.79	0.85|0.82	0.84|0.82
	9	0.6073	0.6249	0.6560	0.6559	0.48|0.73	0.49|0.76	0.57|0.74	0.57|0.74
	10	0.5414	0.5547	0.5599	0.5616	0.49|0.59	0.46|0.65	0.47|0.65	0.47|0.65
	11	0.5003	0.5004	0.5164	0.5166	1.00|0.00	1.00|0.00	0.43|0.60	0.59|0.44
SIMON64	9	0.9467	0.9447	0.9619	0.9582	0.97|0.92	0.96|0.92	0.98|0.95	0.97|0.95
	10	0.7710	0.7788	0.8096	0.8104	0.73|0.81	0.76|0.80	0.78|0.84	0.78|0.84
	11	0.6411	0.6348	0.6578	0.6591	0.57|0.71	0.57|0.70	0.58|0.73	0.58|0.74
	12	0.5479	0.5471	0.5623	0.5632	0.45|0.65	0.46|0.63	0.47|0.65	0.48|0.65
	13	0.5002	0.5035	0.5154	0.5182	0.00|1.00	0.31|0.70	0.39|0.64	0.46|0.58
	14	0.5000	0.5000	0.5003	0.5010	1.00|0.00	1.00|0.00	0.01|0.99	0.00|1.00
SIMON128	14	0.9010	0.9199	0.9267	0.9312	0.87|0.94	0.90|0.94	0.91|0.95	0.91|0.96
	15	0.7975	0.7966	0.8384	0.8383	0.71|0.88	0.71|0.88	0.78|0.90	0.77|0.90
	16	0.6867	0.6857	0.7249	0.7248	0.57|0.81	0.56|0.81	0.61|0.84	0.61|0.84
	17	0.5957	0.5950	0.6259	0.6259	0.45|0.74	0.45|0.74	0.46|0.79	0.46|0.79
	18	0.5390	0.5379	0.5582	0.5580	0.40|0.68	0.39|0.68	0.38|0.73	0.37|0.74
	19	0.5077	0.5072	0.5222	0.5218	0.30|0.72	0.36|0.66	0.34|0.71	0.31|0.73
	20	0.5000	0.5000	0.5060	0.5069	0.00|1.00	0.00|1.00	0.26|0.75	0.29|0.73
SPECK32	5	0.9269	0.9255	0.9280	0.9260	0.90|0.95	0.90|0.95	0.91|0.95	0.90|0.95
	6	0.7860	0.7849	0.7873	0.7867	0.72|0.85	0.72|0.85	0.72|0.86	0.71|0.86
	7	0.6111	0.6123	0.6152	0.6098	0.54|0.68	0.53|0.69	0.53|0.70	0.55|0.67
	8	0.5004	0.5013	0.5107	0.5114	1.00|0.00	0.42|0.58	0.58|0.44	0.55|0.47
SPECK64	4	0.9999	0.9999	0.9998	0.9998	1.00|1.00	1.00|1.00	1.00|1.00	1.00|1.00
	5	0.9884	0.9870	0.9939	0.9914	0.98|0.99	0.98|0.99	0.99|1.00	0.99|0.99
	6	0.8580	0.8494	0.9229	0.9230	0.82|0.90	0.81|0.89	0.91|0.93	0.91|0.94
	7	0.6679	0.6198	0.7182	0.7198	0.64|0.70	0.55|0.69	0.67|0.77	0.67|0.77
	8	0.5256	0.5158	0.5357	0.5369	0.51|0.54	0.56|0.47	0.58|0.50	0.51|0.57
	9	0.5009	0.5006	0.5016	0.5004	0.55|0.45	0.80|0.20	0.68|0.32	0.97|0.03
SPECK128	7	0.9995	0.9995	0.9994	0.9994	1.00|1.00	1.00|1.00	1.00|1.00	1.00|1.00
	8	0.9722	0.9716	0.9860	0.9860	0.96|0.98	0.96|0.98	0.98|0.99	0.98|0.99
	9	0.7787	0.7800	0.8296	0.8293	0.75|0.81	0.75|0.81	0.84|0.82	0.83|0.83
	10	0.5814	0.5831	0.5913	0.5909	0.58|0.58	0.58|0.58	0.58|0.60	0.58|0.60
	11	0.5010	0.5007	0.5006	0.5013	0.65|0.35	1.00|0.00	0.11|0.89	1.00|0.00
HIGHT	8	0.9990	0.9990	0.9990	0.9990	1.00|1.00	1.00|1.00	1.00|1.00	1.00|1.00
	9	0.7500	0.8525	0.8598	0.8600	1.00|0.50	0.94|0.76	0.95|0.77	0.95|0.77
	10	0.5617	0.5003	0.7509	0.7509	0.25|0.88	0.00|1.00	1.00|0.50	1.00|0.50
	11	0.5005	0.5005	0.5007	0.5010	1.00|0.00	1.00|0.00	0.96|0.04	0.13|0.87
HIGHT	12	0.9990	0.9990	0.9990	0.9990	1.00|1.00	1.00|1.00	1.00|1.00	1.00|1.00
	13	0.9647	0.7499	0.9647	0.9647	1.00|0.93	1.00|0.50	1.00|0.93	1.00|0.93
	14	0.5006	0.5005	0.5010	0.5633	1.00|0.00	1.00|0.00	0.94|0.06	0.58|0.55
	15	0.5007	0.5007	0.5010	0.5010	0.00|1.00	1.00|0.00	0.01|0.99	0.98|0.02
PRESENT	5	0.8808	0.8785	0.8828	0.8829	0.84|0.92	0.83|0.92	0.84|0.92	0.84|0.93
	6	0.7077	0.7053	0.7093	0.7096	0.59|0.82	0.59|0.82	0.59|0.82	0.59|0.83
	7	0.5597	0.5593	0.5613	0.5612	0.43|0.69	0.43|0.69	0.45|0.67	0.43|0.69
	8	0.5104	0.5106	0.5106	0.5120	0.40|0.62	0.41|0.61	0.39|0.64	0.37|0.65
	9	0.5003	0.5003	0.5012	0.5018	0.00|1.00	0.00|1.00	0.32|0.68	0.46|0.54
	10	0.5002	0.5002	0.5003	0.5006	0.00|1.00	0.00|1.00	0.00|1.00	0.00|1.00
TEA	3	1.0000	1.0000	1.0000	1.0000	1.00|1.00	1.00|1.00	1.00|1.00	1.00|1.00
	4	0.8864	0.8747	0.9079	0.9079	1.00|0.77	1.00|0.75	1.00|0.82	1.00|0.82
	5	0.5562	0.5491	0.5629	0.5619	0.61|0.50	0.60|0.50	0.61|0.52	0.60|0.52
	6	0.5010	0.5009	0.5010	0.5011	0.98|0.02	0.00|1.00	0.12|0.88	1.00|0.00
XTEA	3	1.0000	1.0000	1.0000	1.0000	1.00|1.00	1.00|1.00	1.00|1.00	1.00|1.00
	4	0.8867	0.8748	0.9700	0.9697	1.00|0.77	1.00|0.75	1.00|0.94	1.00|0.94
	5	0.5046	0.5093	0.5978	0.5009	0.13|0.87	0.75|0.27	0.69|0.51	0.69|0.31
	6	0.5005	0.5008	0.5008	0.5007	0.00|1.00	0.94|0.06	0.87|0.13	0.00|1.00
LEA	8	0.8475	0.8482	0.8473	0.8477	0.78|0.91	0.79|0.91	0.78|0.91	0.78|0.92
	9	0.7209	0.7200	0.7233	0.7231	0.60|0.84	0.59|0.85	0.60|0.85	0.59|0.85
	10	0.5952	0.6010	0.5963	0.5957	0.46|0.73	0.47|0.73	0.46|0.74	0.46|0.73
	11	0.5111	0.5112	0.5113	0.5113	0.45|0.58	0.47|0.56	0.47|0.55	0.56|0.46
GIMLI	8	0.9995	0.9995	0.9987	0.9988	1.00|1.00	1.00|1.00	1.00|1.00	1.00|1.00
	9	0.8735	0.8707	0.8639	0.8735	0.85|0.89	0.85|0.90	0.83|0.90	0.83|0.89
	10	0.6129	0.6041	0.6052	0.6037	0.52|0.70	0.52|0.69	0.51|0.70	0.51|0.70
	11	0.5014	0.5007	0.5238	0.5236	0.90|0.10	1.00|0.00	0.54|0.51	0.54|0.50
	12	0.5012	0.5002	0.5011	0.5010	1.00|0.00	0.00|1.00	0.00|1.00	0.21|0.79

Details: The best validation accuracies obtained within 40 epochs for each round are shown on the left-hand side. We performed two runs for each network, Gohr-Depth 1 and DBitNet, since neural network training contains probabilistic elements, such as the initial weight initialization. The best distinguisher (highlighted in green) is then re-evaluated on freshly generated datasets to obtain the final accuracy results of Table 6. Accuracies compatible with a random guess are shown as highlighted in gray. The right-hand side shows the true positive rate (TPR) and true negative rate (TNR) for each accuracy from the left-hand side.

Table 9: Detailed results for KATAN.

cipher	round	Gohr depth-1		DBitNet	
		(1)	(2)	(1)	(2)
KATAN	40	0.9832	0.9891	0.9953	0.9963
	41	0.98	0.9673	0.9925	0.9908
	42	0.9623	0.9551	0.9869	0.9856
	43	0.9186	0.9081	0.9806	0.9733
	44	0.8686	0.8732	0.9691	0.9586
	45	0.7523	0.7447	0.9447	0.9217
	46	0.7112	0.7058	0.9088	0.8766
	47	0.6738	0.6518	0.8545	0.8267
	48	0.6697	0.6685	0.834	0.7897
	49	0.6029	0.6002	0.7873	0.7526
	50	0.6022	0.5943	0.7437	0.7058
	51	0.5809	0.5742	0.6991	0.665
	52	0.5771	0.5697	0.6657	0.6419
	53	0.5659	0.5621	0.6319	0.6231
	54	0.5562	0.5516	0.6026	0.5935
	55	0.5038	0.5367	0.5859	0.5823
	56	0.5036	0.521	0.5697	0.5647
	57	0.503	0.5242	0.5617	0.5595
	58	0.5033	0.5151	0.5503	0.5497
	59	0.5033	0.5032	0.5467	0.5479
	60	0.5001	0.5032	0.5427	0.5426
	61	≈ 0.50	0.5031	0.5287	0.5266
	62	≈ 0.50	0.5033	0.5252	0.5248
	63	≈ 0.50	0.5018	0.5178	0.517
	64	≈ 0.50	≈ 0.50	0.5153	0.5141
	65	≈ 0.50	≈ 0.50	0.5091	0.5076
	66	≈ 0.50	≈ 0.50	0.5069	0.5078
	67	≈ 0.50	≈ 0.50	0.5066	0.5071
	68	≈ 0.50	≈ 0.50	0.5056	0.5049
	69	≈ 0.50	≈ 0.50	0.5052	0.5049
	70	≈ 0.50	≈ 0.50	0.5026	0.5026
	71	≈ 0.50	≈ 0.50	0.5012	0.5024