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

Emanuele Bellini[1][0000−0002−2349−0247], David Gerault[1], Anna Hambitzer[1], and Matteo Rossi[2]

[1] Cryptography Research Centre, 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. We review recent results in neural cryptanalysis, and identify the obstacles to its application to new, different primitives. As a response, we provide a generic tool for neural cryptanalysis, composed of two parts. The first part is an evolutionary algorithm for the search of single-key and related-key input differences that works well with neural distinguishers; this algorithm fixes scaling issues with Gohr's initial approach and enables the search for larger ciphers, while removing the dependency on machine learning, to focus on cryptanalytic methods. The second part is DBitNet, a neural distinguisher architecture agnostic to the structure of the cipher. We show that DBitNet outperforms state-of-the-art architectures on a range of instances. Using our tool, we improve on the state-of-the-art neural distinguishers for SPECK64, SPECK128, SIMON64, SIMON128 and GIMLI-PERMUTATION and provide new neural distinguishers for HIGHT, LEA, TEA, XTEA and PRESENT.

1 Introduction

The field of cryptography is constantly evolving. From classic ciphers, such as Caesar or Vigenere, to the mid-twentieth centuries and the likes of Enigma, all the way to the era of open cryptography design competitions, a vast quantity of methods have been proposed to encrypt data. As our understanding of the field as a community evolved, finding vulnerabilities in the most recently proposed ciphers becomes increasingly difficult, with each failure being a lesson for future designs. Fortunately, cryptographers now have access to a wide array of automatic tools to assist the evaluation of their new designs. In particular, tools such as SAT, MILP or CP solvers have become prevalent to analyze ciphers against the main attack techniques: differential and linear cryptanalysis, but also, for instance, the automatic search for meet-in-the-middle attacks, or impossible differentials.

Recently, a new family of tools was added to the existing collection: machine learning based distinguishers. Proposed by Gohr at CRYPTO'19 [12], they exploit the ability of machine learning algorithms to recognize complex patterns, in order to distinguish ciphertexts produced by a given block cipher from random data. More specifically, the so-called *neural distinguisher* is trained to label pairs of cipherexts as either *random* or *not random*, *i.e.*, corresponding to the encryption of two messages related by a fixed XOR difference. Following this seminal paper, which showed results on the block cipher SPECK32, a body of work dedicated to the analysis of what these distinguishers could learn started; in parallel, several researchers investigated how to apply neural distinguishers to different ciphers, and how to improve on the initial constructions. However, a systematic study of how to generalize neural distinguishers is lacking. The first obstacle is the structure of the neural network itself: the seminal paper of Gohr relies on design decisions that are very specific to the studied cipher, and more generic techniques are needed. Furthermore, the choice of a good input difference for non-random samples is difficult, and a poor choice results in dramatic performance reductions.

These are the main points we address in this paper. We propose a generic framework to evaluate a cipher through the neural distinguisher perspective. The goal of our tool is to further assist cryptographers, by adding a new, simple way to check for weaknesses. It does not aim at replacing traditional analysis, but rather, at pointing to potential points of vulnerabilities for the cryptographer to investigate.

Contributions

1. We perform a comparative review of the current state of the art in the field of neural cryptanalysis;
2. We propose a fully automated framework to perform neural cryptanalysis of ciphers, independently of their size; this tool is composed of:
   - An evolutionary algorithm for the search of single-key and related-key input differences that works well with neural distinguishers; this algorithm fixes scaling issues with Gohr's initial approach and enables the search for larger ciphers, while removing the dependency on machine learning, to focus on cryptanalytic methods;
   - DBitNet, a neural distinguisher architecture agnostic to the structure of the cipher, which matches or outperforms state-of-the-art architectures despite a simpler training pipeline;
3. Using our tool, we improve on the state-of-the-art neural distinguishers for SPECK64, SPECK128, SIMON64, SIMON128 and GIMLI-PERMUTATION and propose new neural distinguishers for HIGHT, LEA, TEA, XTEA and PRESENT (Table 1).

Organization. The remainder of this work is organized as follows. We first give a short introduction into the ciphers which are analyzed in this work in Section 2. We then give a systematic overview over the past, present and future of neural

Primitive	Arch.	Model	Tr.	Val.	Epochs	Rounds	Acc.	Ref.
GIMLI-CIPHER	MLP	2-2-δ-D	$2^{17.6}$	$2^{14.3}$	20	8	0.5219	[2]
GIMLI-PERM	MLP	2-2-δ-D	$2^{17.6}$	$2^{14.3}$	20	8	0.5099	[2]
	DBitNet	*2-1-CT-R*	$2^{23.25}$	$2^{19.93}$	*40*	*11*	*0.5238*	*This work*
SPECK32	ResNet	20-1-MIX-R	$2^{23.25}$	$2^{19.93}$	200	5	1	[6]
	ResNet	100-1-MIX-R	$2^{23.25}$	$2^{19.93}$	200	6	1	[6]
	ResNet	2-1-CT-R	$2^{23.25}$	$2^{19.93}$	200	8	0.514	[12]
	DBitNet	*2-1-CT-R*	$2^{23.25}$	$2^{19.93}$	*40*	*8*	*0.5114*	*This work*
	ResNet	64-32-δ-D	$2^{28.25}$	/	100	8	0.564	[17]
	MLP	2-1-δ-D	$2^{27.64}$	$2^{26.64}$	/	3*	0.79	[28]
SPECK48	ResNet	96-48-δ-D	$2^{28.84}$	/	100	7	0.634	[17]
SPECK64	*DBitNet*	*2-1-CT-R*	$2^{23.25}$	$2^{19.93}$	*40*	*8*	*0.5369*	*This work*
	ResNet	128-64-δ-D	$2^{29.25}$	/	100	8	0.632	[17]
SPECK128	*DBitNet*	*2-1-CT-R*	$2^{23.25}$	$2^{19.93}$	*10*	*10*	*0.5928*	*This work*
SIMON32	ResNet	64-32-δ-D	$2^{28.25}$	/	100	10	0.611	[17]
	DBitNet	*2-1-CT-R*	$2^{23.25}$	$2^{19.93}$	*40*	*11*	*0.5166*	*This work*
	SENet	2-1-M-D	$2^{31.17}$	$2^{29.17}$	10+1+2	11	0.5174	[3]
	MLP	2-1-δ-D	2^{24}	$2^{27.64}$	/	5*	0.57	[28]
SIMON48	ResNet	96-48-δ-D	$2^{28.84}$	/	100	11	0.814	[17]
SIMON64	ResNet	128-64-δ-D	$2^{29.25}$	/	100	12	0.695	[17]
	DBitNet	*2-1-CT-R*	$2^{23.25}$	$2^{19.93}$	*40*	*13*	*0.5182*	*This work*
SIMON128	*DBitNet*	*2-1-CT-R*	$2^{23.25}$	$2^{19.93}$	*10*	*20*	*0.5074*	*This work*
HIGHT	*DBitNet*	*2-1-CT-R*	$2^{23.25}$	$2^{19.93}$	*10*	*10*	*0.7509*	*This work*
$HIGHT^{RK}$	*DBitNet*	*2-1-CT-R*	$2^{23.25}$	$2^{19.93}$	*40*	*14*	*0.5633*	*This work*
PRESENT	*DBitNet*	*2-1-CT-R*	$2^{23.25}$	$2^{19.93}$	*10*	*8*	*0.5122*	*This work*
TEA	*DBitNet*	*2-1-CT-R*	$2^{23.25}$	$2^{19.93}$	*40*	*5*	*0.5629*	*This work*
XTEA	*DBitNet*	*2-1-CT-R*	$2^{23.25}$	$2^{19.93}$	*40*	*5*	*0.5978*	*This work*
LEA	*DBitNet*	*2-1-CT-R*	$2^{23.25}$	$2^{19.93}$	*10*	*11*	*0.5115*	*This work*

RK: Related key setting. *:respectively 6 and 7 probabilistic rounds are prepended for SPECK32 and SIMON32 in this work, reaching 9 and 12 round distinguishers

Table 1. Summary of the state-of-the-art and our work. Our naming convention for the Model column is detailed in subsection 3.6. In the table, / means unknown.

distinguishers in Section 3 and discuss roadblocks to the automatic application of neural distinguishers to new ciphers in Section 4. We present our solutions in I) the automated finding of a good input difference (Section 5), as well as II) cipher-agnostic neural training pipeline (Section 6.2). We discuss our best distinguishers in Section 7 and conclude in Section 8.

2 Analyzed Ciphers

The following ciphers are analyzed in the context of differential cryptanalysis. Differential cryptanalysis [18] studies the propagation of an input difference δ to an output difference γ through a cryptographic primitive, usually through *differential characteristics* representing the internal difference at each round of

the primitive. Non-linearity makes this propagation non-deterministic, so that finding high probability differential characteristics is a key step in the analysis.

SPECK and SIMON [4] are lightweight block ciphers proposed by the NSA in 2013. SPECK came out in 10 different versions, indexed by their block and key sizes. The construction is a classic iterated ARX, that goes from a minimum of 22 rounds (with blocksize of 32 bits and keysize of 64) to a maximum of 34 (with 128-bit long blocks and 256-bit long keys). The key schedule of SPECK uses the same round function as the main cipher. SIMON has 10 different variants indexed as the ones of SPECK, while the main difference is in the round function: SIMON uses a Feistel structure, with the only non-linearity being the bitwise-and function. The smallest version of SIMON has 32 rounds, while the biggest one has 72 of them.

LEA [15] is a lightweight block cipher published by South Korea in 2013. Similarly to SPECK, it has an ARX construction. The blocksize of LEA is 128 bit, while its keysize ranges from 128 to 256 bits, for a total of three versions. Its number of rounds varies from 24 to 32.

TEA [26] is a lightweight block cipher presented by Wheeler et al. at FSE'94. The round function is built on an ARX construction and the overall structure of the cipher is a Feistel network. TEA has a blocksize of 64 bit and a keysize of 128 bit. In a different way than all the other listed ciphers, in TEA the round keys are injected using the modular addition operation instead of the XOR (this happens partially also in HIGHT, but TEA is the only one in which this is true for *all* the round keys). The total number of rounds for TEA is 64.

XTEA [27] is a lightweight block cipher designed to overcome some weaknesses of TEA. It was firstly presented in a unpublished report in 1997, by the same author of TEA. XTEA inherits blocksize, keysize and number of rounds from TEA, as well as a somehow similar round structure. The most relevant changes are the key scheduler and the fact that, in this version, the round keys are injected using the XOR operation.

GIMLI [7] is a permutation proposed by Bernstein et al. at CHES'17. From this permutation, the authors proposed an hashing algorithm and an authenticated encryption algorithm, respectively GIMLI-HASH and GIMLI-CIPHER. The permutation has a state size of 384 bits arranged in a 3×4 matrix of 32-bit words. Its round function combines an SP-box with a linear layer, and it is iterated for 24 times. GIMLI-HASH is built from it with a sponge construction, while GIMLI-CIPHER uses the monkeyDuplex one.

HIGHT [16] is an hardware-oriented lightweight block cipher proposed by Hong et al. at CHES'06. Its structure is based on a variant of the generalized Feistel construction, with a round function using ARX operations. As TEA and XTEA, HIGHT has a 64 bit blocksize and 128 bit keysize. Its total number of rounds is 32.

PRESENT [9] is a lightweight block cipher presented by Bogdanov et al. at CHES'07. It has a SPN structure, with a blocksize of 64 bits and two possible keysizes: 80 and 128 bits. For both these versions, the number of rounds is 31.

3 Neural Distinguishers: Past, Present and Future

We first provide a short, general introduction to machine learning and neural networks (Section 3.1). We then focus on the introduction of Gohr's neural distinguishers and Gohr's neural difference search (Sections 3.2 and 3.3). We discuss current areas of research on neural distinguishers, *i.e.*,, to understand what is learnt (Section 3.4), as well as improving the neural distinguishers (Section 3.5).

3.1 Machine Learning and Neural Networks

Artificial intelligence (AI) aims to enable machines to mimic or surpass human behavior. Machine learning (ML) is a subfield of AI, which investigates algorithms that "gives computers the ability to learn without explicitly being programmed" [23] or "learn from experience" [19]. In contrast to optimization, for ML the performance on unknown data is the key indicator, in other words the learning progress is quantified by the generalization error of the algorithm [13]. Typical machine learning algorithms include decision trees, random forests, support vector machines (SVMs), linear and logistic regression and neural networks. Deep learning is a subfield of ML which uses deep neural networks. The following short introduction to neural networks is oriented on [13].

A deep neural network f maps an input $\mathbf{x}$ to an output $\mathbf{y}$, *i.e.,* $\mathbf{y} = f_\theta(\mathbf{x})$. During the training of the neural network the values of the networks parameters θ are learned. In a feedforward neural network the inputs $\mathbf{x}$ are passed through the different *layers* of the neural network. For example in a network with three layers $\mathbf{y} = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$, we have one output layer (3), one hidden layer (2), and one input layer (1). The depth of a neural network is usually given by the number of layers with trainable parameters. Each layer of the neural network contains *neurons*. The learned parameters θ are the weight and bias parameters of those neurons.

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 $J(\theta)$ with typical choices being the mean squared error MSE, mean-absolute error MAE, or binary cross-entropy. The scalar cost $J(\theta)$ is *back-propagated* to the neural network parameters θ by the back-propagation algorithm [22], resulting in the *gradient* of the cost function with respect to the parameters $\nabla_\theta J(\theta)$.

The *optimizer* will adjust the network parameters θ based on the gradient value, and the hyperparameters of the optimizer itself, which includes among others the *learning rate*. Typical choices for the optimizer include stochastic gradient descent SGD or ADAM.

Training of the neural network is done on three datasets: training $\mathbf{x}_{\text{train}}, \mathbf{y}_{\text{train}}$, validation $\mathbf{x}_{\text{val}}, \mathbf{y}_{\text{val}}$ and test $\mathbf{x}_{\text{test}}, \mathbf{y}_{\text{test}}$ data. The goal of machine learning, in general, is *not* to optimize the algorithm f on a known dataset, but to make the algorithm *generalize* to previously unseen data. This is why typically θ-parameter adjustments are done based on the training data, however, 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 start to learn the training dataset "by heart". This phenomenon is known as *overfitting*.
The training of a neural network is done in *batches* and *epochs*. One epoch means that the neural network has been optimized on the full training dataset. The network parameters are adjusted, however, when one batch of training data has been seen by the neural network. In the extremes are batch sizes of 1, and batch sizes of the full training dataset. Typical are values in between, and training outcomes can vary immensely depending on the choice of the batch size.
The evaluation after each training epoch uses a *metric* which may not be identical to the loss function. For example, in a classification problem, the typical loss function is binary cross-entropy, and the typical metric is the accuracy, *i.e.,* the percentage of correct classifications.

The design of a neural network involves choosing the number of layers, *i.e.,* the depth, as well as the types of the 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.
One problem in training deep neural networks can be that one layer stops learning. This will prevent the gradient information from successfully back-propagating to previous layers as well. This is circumvented by the introduction of *residual* or *skip* connections.
One of the most fundamental design choices is the choice of the network layers *nonlinearities*, with typical choices being ReLU, sigmoid and tanh. Without the nonlinearity, the chained function $f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$ will simply be a linear function. ReLU is the most popular choice for the hidden layers, as it is fast to compute and avoids the so-called *vanishing gradient problem*. The *sigmoid* activation is typically encountered in output layers for classification problems, as it restricts the output to between 0 and 1. The output between 0 and 1 can either represent a single class probability for multiple output neurons in a multiclass problem, or directly the class itself, class 0 or class 1, in a binary classification problem.

3.2 Gohr's Neural Distinguishers

In his seminal paper, published at CRYPTO'19, Aron Gohr [12] 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 messages, 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 residual network with four main parts, the first three of which are visualized in Fig. 1. At the input a 64 bit ciphertext

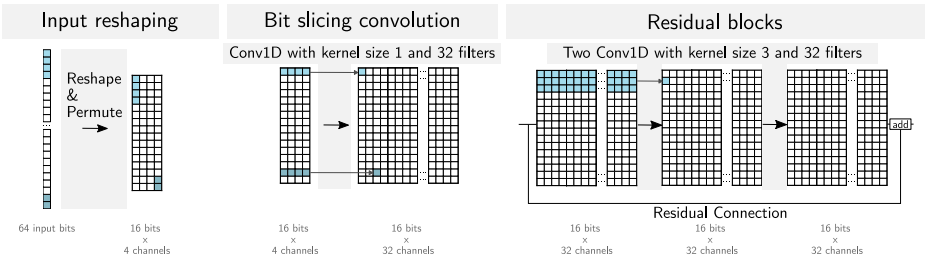

Fig. 1. Visualization of three main parts of Gohr's neural distinguisher.

pair of SPECK32/64 is reshaped and permuted into a 16 bit wide tensor with 4 channels. From a cryptographic perspective the input reshaping reflects the knowledge on 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 are the residual blocks: Each residual blocks consists of two convolutional layers Conv1D($k = 3, f = 32$). In Gohr's publication the number of residual blocks is what denotes a depth-1 neural distinguisher, respectively a depth-10 neural distinguisher[3].
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. Large weights in a network make overfitting more likely, and the L2 regularization penalizes such large weights.
The full Python TensorFlow implementation is available on GitHub [11].

[3] To refer to the number of residual blocks as depth is not used conventionally. For example in the original publication of the first residual network ResNet [14], ResNet34 consists of 34 weighted layers in total, including a fully connected dense layer, while it has only 16 residual blocks.

3.3 Neural Difference Search

Gohr additionally proposes an algorithm to derive good input differences for neural distinguishers without prior human knowledge. This algorithm is based on few-shots learning, where the features learnt by a network are used as input to a simpler machine learning algorithm, trained on less 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 to train a simple ridge regression classifier on small numbers of samples for new differences δ'.

A greedy algorithm with exploration bias is used to suggest new candidates δ': from a random initial difference, a random bit is flipped at each iteration, and the resulting input difference becomes the new base if it obtains a better score through the ridge classifier, or an exploration threshold is reached. The algorithm is summarized in Algorithm 1.

Algorithm 1: 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};

3.4 Understanding What is Learnt

Neural distinguishers becoming the state-of-the-art attack for 11 rounds of SPECK32 raised a lot of questions among the cryptography community. New attacks are usually welcome, as they deepen our understanding of cryptographic algorithms; however, when they were first presented, neural distinguisher did not unlock some new theoretical knowledge about speck: they just worked.

For this reason, it is crucial to understand what property is being learnt by new neural distinguishers, and determine whether they rely on some new exotic property, potentially threatening more ciphers, or if they are simply exploiting known properties more efficiently than we usually do. As such, neural distinguishers are not to be thought of as replacement for cryptanalysis, but rather as a tool that may help identify properties that are difficult to see, so that a cryptographer can exploit them with classical (non-neural) techniques.

The first step towards understanding the properties learnt by neural distinguishers was a paper by Benamira *et al.* [6], which identified meaningful patterns in the behaviour of Gohr's distinguishers on SPECK. In particular, they observed that the pairs that were correctly classified as non-random overwhelmingly followed a truncated differential after a few rounds; for instance, the 5-round pairs satisfying a truncated differential labeled $TD3$ at round 3 are correctly classified with accuracy over 99%, and they represent 87.11% of the total pairs. Similarly, 5-round pairs following the truncated differential labeled as $TD4$ at round 4 are correctly classified with accuracy over 99%, and $TD4$ is followed by around 50% of the total pairs. These observations point to the possibility that the neural network learns a property that gives information on the difference in previous rounds, such as a differential-linear property. The authors further modified the neural network to use a heaviside activation function, which forces its output to be 0 or 1, and studied the resulting boolean functions learn on speck. From these, they were able to extract masks to apply to the input to the neural network, creating frequency tables listing the probability of events such as "The concatenation of bits a, b of $C_0 \oplus C_1$, bits c, d of $L_0 \oplus R_0$, and bits e, f of $L_1 \oplus R_1$ take value $v_a||v_b||v_c||v_d||v_e||v_f$ with probability p". Based on the resulting probability vector, the authors obtain a distinguisher almost as good as Gohr's, without the need of convolutional layers to learn complex features. Through this research it was shown that the 1D convolutions, while efficient, can be replaced by a more cryptographically interpretable construction, followed by a machine learning based classifier. In addition, they experimented with a neural distinguisher for which a sample was composed of multiple ciphertext pairs with the same label, and reached 100% accuracy for 5 and 6 rounds of SPECK32, with 10 and 50 samples respectively.

In [1], Bacuieti *et al* further investigate the structure of the neural network itself. In particular, they focus on simplifying the neural network, in an effort to make it more efficient and interpretable. More specifically, following 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 behaviour; however, the LIME technique they used

did not reveal any of the input to be more significant than the others in the classification. 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.

3.5 Improving the Neural Distinguishers

Besides work based on understanding the functions learnt by the neural distinguishers, another area of research is the improvement of the process. In particular, the task learnt by Gohr's neural distinguishers is somehow counter-intuitive: they learn to distinguish a given distribution from random, and to do so, receive random samples as part of their training. However, by definition, no property can be learnt from randomness, so training on random data seems inefficient.

This point is addressed by Baksi *et al.* [2], who propose a different experiment: rather than distinguishing samples issued from a difference δ from random, they focus on distinguishing which of t classes a pair belongs, where a class is determined by an input difference δ_i. They apply this framework, named *Model 1*, to GIMLI, ASCON, KNOT, and a model closer to Gohr's original paper, labeled *Model 2*, to Chaskey. In their experiments, the authors use custom neural network architectures, based on simpler building blocks than Gohr's.

In Zezhou *et al.* [17], the authors study another model for neural distinguishers, labeled *MOD* for Multiple Output Differences, where the neural network receives as input samples composed of a set of output differences, and outputs a label 0 (random) or 1 (non-random). The authors investigate how to obtain good input differences, and derive sets of input differences for SIMON and SPECK from SAT solvers, by enumerating differential characteristics within a fixed distance to the optimal.

Yadav *et al.* [28] propose an extension of the neural distinguisher framework, called differential-ml cryptanalysis. In essence, they observe that if we have a good neural distinguisher, it is possible to prepend some probabilistic rounds, in order to build longer distinguishers. They obtain a 9-round speck32 distinguisher, a 12-round SIMON32 distinguisher, and an 8-round GIFT64 distinguisher.

At LNNS'21 [5] , Bellini and Rossi compare neural and classical distinguishers for the ciphers TEA and RAIDEN injecting differences via modular addition (in contrast to XOR, used in most works). The showed that simple neural distinguishers based on an MLP structure can outperform classical generic distinguishers based on differential characteristics also in these settings, reaching 8 rounds for both ciphers.

More recently, at Asiacrypt'22, Bao *et al* [3] perform an in-depth study of the key-recovery mechanisms associated with neural distinguishers; in particular, they compare the properties and wrong key response profiles of SPECK32 and SIMON32. In addition, they study different formats for the input to the neural network. For the case of SPECK32, it was shown in [6] that exploiting the knowledge that the right part of the state at round $R-1$ can be computed from the state at round R could help gain more insight on the function learnt by the network. Bao *et al* apply a similar strategy, by training networks labeled

as $ND^{SIMON_{r-1R_{VD}}}$ on the information that can be learnt on the previous round (the left parts, and the difference in the right part). They also study a distinguisher based on average key rank. These experiments are, for the highest rounds, conducted using a different neural network architecture, SENet [3], and show that it outperforms Gohr's choice of ResNet for SIMON-32. In this paper, we achieve similar results on SIMON with a simpler training method.

These results are summarized in Table 1.

3.6 Summary of the Training Models

In order to have a common language to refer to the different input formats mentioned above, we propose the following taxonomy: a neural distinguisher is quantified by parameters n, m, T, E, where n is the number of ciphertexts used to build each sample, m is the number of differences used, T is the type of the input (CT for ciphertexts, δ for the difference, Mix for a mix of both), and E is the type of experiment (R when the labels correspond to random or real, D when the label depends on the index of the input difference).

Under this convention, for instance, Gohr's initial experiments are 2-1-CT-R.

4 Roadblocks for Applying Neural Distinguishers Automatically

The field being in its infancy, it is still unclear what machine learning architecture works best. Basic MLPs and CNNs have been tried in [2], as well as significantly larger networks such as SENet [3], or combinations of hand-built features with non-neural classifiers in [6]. We believe that, while exploration of the capacities of neural classifiers is going on, the aim should not be to reduce the size of the neural networks, but rather, to understand how much they can learn.

In the following experiments we investigate limitations to what neural distinguishers can really learn (Section 4.1) and formulate our take-aways for what prevents an automated application to new ciphers (Section 4.2). We then point out limitations on another end, namely the identification of good input differences for new ciphers (Section 4.3).

4.1 What Can Neural Distinguishers Really Learn?

In Gohr's distinguisher experiment the neural network is trained to distinguish between two types of ciphertext pairs which originate from a SPECK32/64 encryption. The ciphertext pairs (C_0, C_1) are either obtained from 1) a plaintext pair with a specific plaintext difference $(P_0, P_0+(0x0040, 0x0000))$ or 2) a randomly generated plaintext pair (P_0, P_1).

What underlies the ability of Gohr's neural network to distinguish between case 1) and 2)? In a simple example, the neural network could decide that the

ciphertext pair in question is obtained from 1) if it can predict the value of several bits in the ciphertext pair, given the remaining bits of the ciphertext pair as input.
In more general terms, given the bits of the ciphertext pair $(c_1, \ldots, c_{64})$, Gohr's neural distinguisher is likely to learn Boolean functions of the form $c_i = f(c_{j|j\neq i})$ and base the distinguishing decision on the correctness of the bit prediction outcomes. In the specific case of ciphertext generated with a reduced round version of SPECK, the Boolean functions in question will reflect the cipher structure. If, however, we wish to use the distinguisher on another cipher, the underlying Boolean functions will change.

Here, we investigate the most general case of a random Boolean function to expose any fundamental limitation of Gohr's neural distinguisher. We show that the random Boolean functions which can be learned by Gohr's neural distinguisher are surprisingly limited. To understand the limitation we inspect the construction of Gohr's neural distinguisher, and find that in particular the input restructuring of Gohr's neural distinguisher reflects expert knowledge on SPECK. We show that this input restructuring overcomes the limitation in the case of SPECK.

A Motivating Example In [6], the authors observe that the performance of Gohr's neural distinguisher for r rounds is strongly correlated with the difference observed after $r-2$ rounds, hinting that the classification accuracy could be explained through differential-linear properties. For SPECK-32/64, with the input difference fixed to $(0x40, 0)$, we evaluated all 2^{32} possible linear mask on the output difference, and found $(0x5820, 0x4020)$ to be the most biased for 7 rounds. The quantity $\delta_7 \cdot 0x58204020$, takes value 0 with average probability 0.57, where δ_7 is the output difference at round 7, and $\cdot$ is the bitwise dot product. A basic 7-round distinguisher can be built through this property alone: if $\delta_7 \cdot 0x58204020$ is equal to zero, classify as non-random; otherwise as random. Such a distinguisher is right when (1) the pair is indeed non-random, and the property holds, or (2) the pair is random, and the property does not hold; in other words, its accuracy is $\frac{1}{2} \cdot Pr[\delta_7 \cdot 0x58204020 = 0|\text{non-random}] + \frac{1}{2} \cdot Pr[\delta_7 \cdot 0x58204020 = 1|\text{random}] = \frac{1}{2} \cdot 0.57 + \frac{1}{2} \cdot 0.5 = 0.535$.

We evaluate the hypothesis that Gohr's neural network is able to learn such a differential linear property, as it should weight heavily in the classification accuracy. To do so, we build an artificial dataset for Gohr's network, in which the samples are random 64-bit strings, and the label Y_i, corresponding to sample $L_i||R_i$, is equal to $(L_i \oplus R_i) \cdot 0x58204020$. Under this setting, the neural network is not able to go over 50% accuracy when trained on 10^7 samples, even with a depth of 10. This hints that further accuracy gains might be obtained by making the neural network better at detecting such functions, further motivating the experiments of this section.

Limitation of Gohr's network: An experiment on learning random Boolean functions. In this experiment we evaluate how well Gohr's network can learn a random Boolean function. To preserve similarity to Gohr's experiment, we still

use 64-input bits $X = (x_1, \ldots, x_{64})$. However, now Gohr's network is trained to predict the 0 or 1 outcome of the Boolean function $Y = f(x_1, \ldots, x_{64})$. To vary the difficulty of the Boolean function, we change its degree d and the number of terms t. For example, the random Boolean function $f(x_1, \ldots, x_{64}) = x_6x_{18}x_{20} \oplus x_{33} \oplus x_{52} \oplus x_{61}$ would have a (multiplicative) degree $d = 3$ and $t = 4$ terms. Note, that only 6 of the 64 input bits are "active" in the Boolean function. The remaining bits can be seen as "noise bits" which do not influence the Boolean function outcome, but still appear at the input of the neural network. For each random Boolean function, we train Gohr's depth-1 neural distinguisher to predict Y on 10^7 inputs X for 40 epochs. After each epoch, the accuracy is evaluated on 10^6 validation datapoints. Figure 2 shows the best observed validation accuracy for each random Boolean function of degree d and t terms. Surprisingly, we observe that Gohr's network cannot easily learn random Boolean functions already starting at $t \geq 8$ terms. In conclusion, this experiment shows that while Gohr's neural distinguisher is excellent at SPECK, it seems to have difficulties learning other random Boolean functions with more than seven XOR terms.

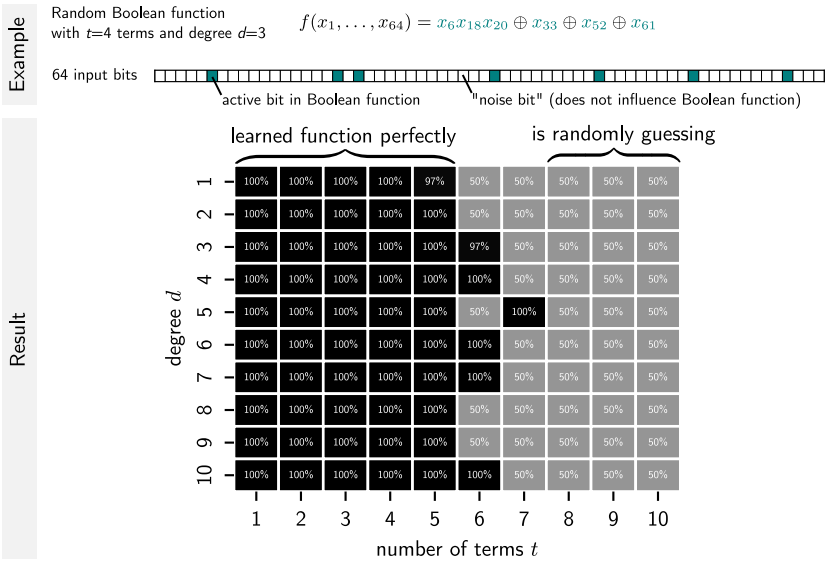

Fig. 2. How well can Gohr's network learn a random Boolean function?

Why is Gohr good at distinguishing SPECK, but not at learning random Boolean functions? The following two experiments pinpoint two reasons for the difficulty of Gohr's neural distinguisher learning a random Boolean function of more then $t \geq 8$ terms. In the first experiment, we fix the random Boolean function to a single one f_{fixed} with $d = 1,\ t = 8$. Then we force all noise bits to zero and let

Gohr's network learn f_{fixed} with the same training procedure as in the previous experiment. We gradually increase the number of noise bits n_{noise} from 0 to 56, where 56 is the original value in the previous experiment. Figure 3 (left) shows that Gohr's network can easily learn a random Boolean function with $t = 8$ for a small number of noise bits n_{noise}. In conclusion, the noise bits prevent Gohr's neural network from learning random Boolean functions, even of a relatively small number of terms.

In the second experiment, we manipulate the Boolean function itself, resulting in Gohr's network learning f_{manual}. f_{manual} still is of degree $d = 1$, $t = 8$ terms and has $n_{\text{noise}} = 56$ noise bits. However, here, we change the *position* of the active bits as indicated in Fig. 3 (right). The input reshaping in Gohr's network is such that the 64 input bits are rearranged as shown in Fig. 3 (center). The first convolutional layer of Gohr's network slices through four bits in each row. Figure 3 (right) shows that if the active bits of f_{manual} are chosen in fashion which reflects the particular input reshaping of the network, the Boolean function f_{manual} can be learned easily (within 4 epochs with full validation accuracy). Here, the active bits all lie within subsequent rows of the reshaped input.

However, if the active bits of f_{manual} are either randomly chosen, or especially unsuited to the input reshaping, it cannot be learned easily. For example the XOR of the first 8 input bits $f_{\text{manual}} = x_1 \oplus x_2 \oplus \ldots \oplus x_8$ cannot be learned easily by Gohr's neural distinguisher. In conclusion, the last experiment demonstrates that the input reshaping in Gohr's neural distinguisher is essential for its ability to learn specific Boolean functions.

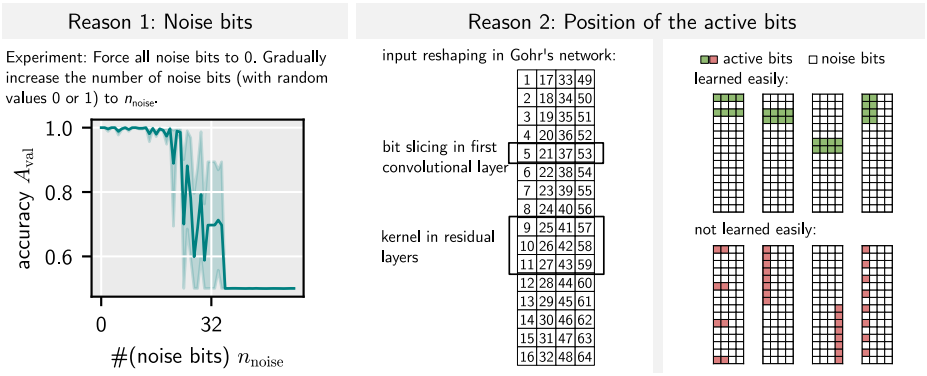

Fig. 3. What prevents Gohr's network to learn a random Boolean function with $t = 8$ XOR terms?

4.2 Lessons Learnt

The experiments presented previously give strong indications that the application of Gohr's ResNet to other ciphers may not be give optimal results. In

the following we summarize our lessons learnt from what prevents automated attempts to apply neural distinguishers to a range of ciphers.

The Learning Rate Schedule. For the training of Gohr's neural distinguisher the ADAM optimizer is used with a cyclic learning rate that varies over 10 epochs between limits of 0.002 and 0.0001. ADAM is known as one of the most advanced optimizers, however, it has been observed to fail to converge to an optimal solution [21]. Such convergence failure may make it necessary to manually find an optimal learning rate schedule. For our purposes of a generic application to a range of new target ciphers, such a manual choice should be avoided. As an alternative mitigation of the convergence issue, Reddi *et al.* introduce the AMSGRAD algorithm in *"On the Convergence of Adam and Beyond"* [21] at ICLR 2018.

The Neural Network Hyperparameters. The network used in Gohr's paper uses 32 filters for each convolution layer, and 64 neurons for the first dense layer. These parameters incidentally 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 it these parameters.

To evaluate how to rescale the parameters, we focused on SPECK128, using the input difference $(0x80, 0x0)$. This choice mirrors the input difference $(0x40, 0)$ chosen for SPECK32, which propagates to $(0b100\cdots0, 0b100\cdots0)$ with probability 1 after 1 round for SPECK32: $(0x80, 0x0)$ propagates to $(0b100\cdots0, 0b100\cdots0)$ with probability 1 after 1 round for SPECK128. Furthermore, it is among the top input differences returned by our low HW optimizer. As a baseline result, we trained Gohr's initial neural network with its default parameters, a depth of 1, using 10^7 samples for training and 10^6 samples for testing, for 200 epochs, and reached 9 rounds, with an accuracy 0.6519. We then trained new networks on 9 rounds, varying the number of filters (32, 64, 128) and neurons (64, 128, 256) on depth 1, reaching at best 0.6541 accuracy with 256 filters and 64 neurons. We then trained a network with depth 10, once with 32/64 filters/neurons, and once with 128/256 filters/neurons, and obtained the respective final validation accuracies 0.6564 and 0.6560.
From these experiments, we conclude that, while scaling the parameters seems to have some impact on the final accuracy, this impact is very limited. At this point, we could either attempt to fine-tune the structure of the network further, or go with a more generic approach; we chose the second option, resulting in the DBitNet network, presented in subsection 6.2.

The Reshaping of the Input. Gohr's neural distinguisher's structure follows the division of SPECK into 2 words. However, when attempting to apply it 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 4, 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. 3 (center). This in turn means less necessary kernel shifts, and accordingly less multiply-accumulate operations, *i.e.,* FLOPs.

The Training Pipelines. When training a neural distinguisher, it is sometimes the case that the highest achievable round fails to be trained using straightforward techniques. In order to obtain an 8-rounds distinguisher for SPECK32, Gohr [12] uses a staged training scheme: he retrains the best 5-round distinguisher on the input difference $(0x8000, 0x840a)$, which is the most likely to appear after 3 rounds. This distinguisher is then retrained for 8 rounds, with a 100 times more data than the other distinguishers, to finally reach 0.514 validation accuracy. Bao *et al* [3] use a similar staged training for their 10-round SIMON32 distinguisher.

These elaborated training schemes are not easily transferable to other ciphers, as they require looking at the differential characteristics of the studied cipher. This level of fine-tuning makes it difficult to adapt to other ciphers.

4.3 Finding a Good Input Difference for a New Cipher

It has been shown in previous work [6] that the input difference to the best differential characteristic is, at least for SPECK, not a good choice for neural distinguishers. This is related to the very nature of neural distinguishers: differential characteristics maximize the probability for a given output difference, but this probability is too low for the corresponding output difference to appear often on a single pair. For instance, the best differential characteristic for 5 rounds of SPECK-32 has probability 2^{-9}, so that it would appear in roughly one in 500 pairs; this is clearly not enough for a neural distinguisher.

In [12], a neural difference search algorithm is proposed (1), which successfully finds the input difference used in the SPECK32 distinguishers. However, adapting it to different ciphers is non-trivial. In particular, the choice of the starting round (3), the number of iterations (2000) and the alpha parameters may need to be adapted. Furthermore, the preprocessor itself is dedicated to speck, with a specific input reshaping and learning rate, so that preliminary design decisions are necessary. Nevertheless, we study the scaling properties of the optimizer on SPECK128. The choice of a SPECK variant minimizes the design decisions to be made: we only change the word_size parameter of the residual network to 64, and use the neural difference search method provided in Gohr's Github repository.

This first experiment was run 70 times in total, 10 for each starting round, from 1 to 7. In none of these runs was the optimizer able to produce a good input difference. Furthermore, the evaluation function was very bad after 4 rounds, as the preprocessor was never able to reach over 0.5 accuracy in these cases.

In order to help the preprocessor to learn, we replaced its initial random difference by a random low Hamming weight one. This allowed the reprocessor

to surpass 0.9 validation accuracy at least once among the 10 runs for all starting number of rounds. The optimizer, however, still returned random differences.

In order to help the optimizer converge, we additionally forced the optimizer's first guess to have Hamming weight 1; this showed to be the best setup we tried for the neural distinguisher search.

We validated the results of the optimizers by training the neural distinguisher for more epochs with the obtained differences. None of the results of our first experiment, with the basic algorithm, resulted in a distinguisher for more than 7 rounds. On the other hand, all but one of the input differences returned in the last (low Hamming weight) experiment resulted in distinguishers for 8 rounds or more. In addition, the optimizer returned 3 input differences $((0x200000, 0x2000), (0x800000000, 0x80000000), (0x1000000000, 0x100000000))$ that resulted in distinguishers on 10 rounds.

From our experiments, it appears that with larger ciphers, such as SPECK128, Gohr's initial optimizer hits a limit. It is possible to modify it to force the use of low hamming weight starting points, but the resulting optimizer fails at returning the best input difference for SPECK128, as we show in later sections. While further improvements to this optimizer might help, we chose a different route, that attempts to use cryptographic knowledge to find good input differences; the corresponding optimizer is presented in subsection 5.1.

5 Solution Part I: Automated Finding of Good Input Differences

In the previous section, we identified generalisation issues with the neural difference search algorithm. In this section, we propose a different, non-neural approach.

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 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 [6] that this disparity is related to truncated differentials. In addition to the truncated differences $TD3$ and $TD4$, 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%. Such strong biases are likely to lead to high probability differential-linear properties

We focus on the problem on 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 (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, $b(\delta)$ is the sum of the biases of each bit position j in the output difference,* i.e.*,*

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

The 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 (Approximate 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 approximate 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.*,*

$$\tilde{b}^t(\delta) = \sum_{j=0}^{n-1} \left| 2 \cdot \frac{\sum\limits_{i=0}^{t} (E_{K_i}(X_i) \oplus E_{K_i}(X_i \oplus \delta))_j}{t} - 1 \right|$$

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 an approximate bias score $\tilde{b}^{2000}(\delta)$ for low Hamming weight (1 and 2) input differences on SPECK-128, 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 convince us to perform a search based not on the results of a linear classifier, but on the significantly faster to compute biased 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.

5.1 Evolutionary Optimizer

Optimizer Parameters The parameters used in our algorithm are:

- Initial population for each generation: 32
- Mutation probability: 0.1
- Approximate bias score samples number t: 10^4
- Relevance threshold T_b: 0.01

Algorithm In our algorithm, each individual of the population represents an input difference. Starting from an initial population of 1024 random input difference, we compute an approximate bias score for each of them, and keep the 32 obtaining the highest score. Starting from these 32 input differences, we run a total of 50 iterations, during which new individuals are derived and evaluated. The algorithm is described in 2.

Algorithm 2: Evolutionary optimizer

```
starting_population ← [ RandomInt(0, 2^n − 1) for 1024 times];
Sort starting_population by b̃^t(·) (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) < 0.1 then
                m ← 1
            else
                m ← 0
            end
            Add current_population_i ⊕ current_population_j ⊕ (m ≪
              RandomInt(0, n − 1)) to candidates
        end
    end
    Sort candidates by b̃^t(·) (descending order);
    current_population ← first 32 elements of candidates;
end
return candidates;
```

After these 50 iterations, the algorithm produces a list of 32 input differences, which have a high bias score.

Accounting for the Starting Round The choice of the starting round has an influence on the bias score. For instance, in the related key case for SPECK32-64, any key difference of the form $(x, 0, 0, 0)$, with a plaintext difference of 0,

results in an output difference of 0 after 2 rounds by construction of the key schedule algorithm, and therefore, a maximal bias score. On the other hand, not all such input differences are equally good after 2 rounds.

In order to make our algorithm as generic as possible, we chose to start from 1 round, run our evolutionary algorithm, and increment the number of rounds if the highest bias score returned is greater than a threshold T_b. In the end, we have R lists of 32 differences, corresponding to the R rounds for which a bias score greater than T_b was returned. Note that the search starts from scratch after each round increment: the population is not carried forward from one round to the next, in order to avoid biases. Therefore, we perform a final evaluation of all the obtained differences from 1 to R rounds, and compute a *weighted cumulative* bias score, in which each score is multiplied by the corresponding number of rounds. This is to account for the possibility that a difference may be bad for early rounds, but become good as the number of rounds increases. Since we are interested in reaching as many rounds as possible, a difference that works well in late rounds should be favored.

5.2 Optimizer Results

Our optimizer returned a large number of solutions, represented in Table 2. These input differences are all relatively good, and the highest scoring are usually among the best. This is, however, not so clear-cut. For instance, 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). On the other hand, for SPECK128, one input difference is clearly dominating the others. Picking the highest ranking difference gives a good approximation on the number of rounds that can be reached (in our experiments, it was never more than 1 round short of the best number of rounds we reached); however, finding the actual best input difference for ciphers that have more than a few solutions is tricky, as it requires a lengthy training process. A difficulty in the process was selecting which of the input differences to further investigate; our first choice was to only select the best 3, but for some of the studied ciphers, we observed a higher number of differences sharing very similar scores; for instance, for SIMON64, there are 64 differences with scores within 1% of the highest ranking choice, and 9 with scores within 0.1%. We therefore chose to use distance to the highest score as a metric to chose which differences to train further, and picked 10% as a threshold. In this restricted setting, we still had over 800 neural distinguishers to train.

The complete list of obtained input differences will be made available in a public repository at the time of publication, as it is of independent interest. In the next section, we present fully trained neural distinguishers for some of our best differences.

Overall, our optimizer performs well, and retrieves known good differences from the literature, or improves on existing ones. The first reason is that it does not need to rely on few-shot learning, so that the evaluation function is significantly faster. In each of the 50 iterations, up to 1024 input differences need to be evaluated; neural difference search would be harder to scale that

much. Secondly, the greedy optimizer of neural difference search flips one bit at a time, which limits its ability to explore cases where several bits need to be set together, or high hamming weight input differences; in contrast, our evolutionary approach promotes more diversity within the solutions.

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

Table 2. For each studied primitive, the number of solutions returned by the optimizer, and the number of $\epsilon - close$ solutions: solutions that have scores within a factor ϵ of the best score for the primitive.

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

Drawing on the identified roadblocks for the automatic application of neural distinguishers to new ciphers in Section 4, we are resolved to utilize a simple training pipeline (Section 6.1) and develop a generic network we call DBitNet (Section 6.2). Whenever a new network is presented, it should be characterized in terms of its computational and memory resources. We compare DBitNet to ResNet and SENet in Section 6.3. We conclude the section by presenting the neural network training results in Section 6.4.

6.1 Our Training Pipeline

We propose a simple pipeline: when training a neural distinguisher for rounds R_{start} to R_{end}, one retrains the same network of R_{start} for round $R_{start}+1$ until round R_{end} is reached. For instance, in the case of SPECK32, one would train the network N_5 for 5 rounds, retrain N_5 on the 6 rounds 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. We refer to this technique as the *staged pipeline* in the rest of this paper. Distinguishers are characterized in terms of their accuracy, *i.e.,* in the percentage of correctly identifying between ciphertext pairs of a chosen plaintext difference,

and those of random plaintext pairs. The number of ciphertext pairs used for training, respectively validation are 10^7 and 10^6 samples.

The Random Guess Limit. It is important to note the limit on when we can consider a found accuracy well above a random guess limit. An experiment like the distinguisher can be modeled as a Binomial experiment: We perform n trials and there are two possible outcomes in each trial (random or not random). In our case each outcome has equal probability $p = 50\%$ since our dataset has an equal number of random and not random samples. The expected binomial standard deviation in this case is $\sigma = \sqrt{n/4}$. Or, if we are to express it as a percentage $\sigma_{\%} = 1/(2\sqrt{n})$. In our experiment the trials are $n = 1 \times 10^6$ validation sequences, resulting in an expected standard deviation of $\sigma_{\%} = 0.05\%$. We set our limit for accuracies A significantly above a random guess at 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 demonstrated that when learning a random Boolean function weaknesses can be exposed which relate to the particular input reshaping that takes place in the distinguisher. We further identified a range of hyperparameters that prevent successful application to new ciphers in Section 4, the most important among them again being the input reshaping.
How can the input reshaping in Gohr's neural distinguisher be avoided altogether? Speaking in a simplifying manner, the input reshaping serves to investigate dependencies of far-apart as well as neighboring bits in the 64-bit input: For example, observe that 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. 3 (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 are so-called dilated convolutions, as presented in "Multi-Scale Context Aggregation by Dilated Convolutions" by Yu and Koltun [29]. 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 [20].

A dilated convolution uses a dilation rate above one, Fig. 4a). 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 $k = 2$

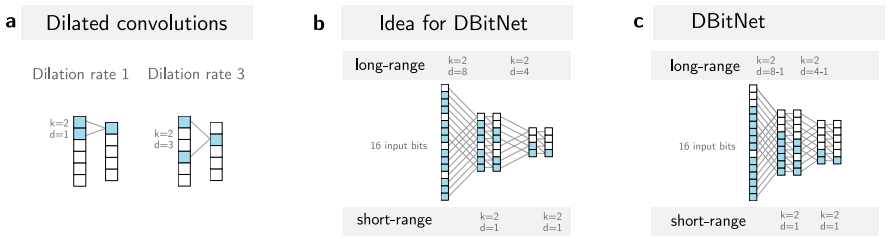

Fig. 4. a) The concept of dilated convolutions, b) The idea for DBitNet c) The actual design of DBitNet.

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. 4b). 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. 4b) 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 [14]. 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.
The TensorFlow implementation of DBitNet will be made available at the time of publication.

	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]
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-PERM	768	12	32	[383, 191, 95, 47, 23, 11, 5]

Table 3. Settings for Gohr's neural network and DBitNet.

Neural Network Settings for Different Ciphers. When working on a different cipher many model and training parameters and hyper parameters 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 size. Based on this input size, for DBitNet the dilation rates are given 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 4.2. In principle for Gohr's network also the number of filters, as well as the cyclic learning rate has to be adapted. 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 3.

6.3 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 [8]. We use the TensorFlow Keras module `keras-flops` to evaluate the FLOPs for each model. TensorFlow provides a native routine `model.count_params()` for the parameter count. The results are shown in Table 4. 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 presented cipher, we argue that an increase of the FLOPs with an increase of the cipher size 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 $\mathcal{ND}_{VV}^{\mathrm{SIMON}_{8R}}$ provided on the GitHub repository of [3] for SIMON32 and find that it has 13.5M FLOPs, and 449.46k parameters.

6.4 Neural Network Training Results

For each target cipher in Table 5 we start with the set of differences found by the evolutionary optimizer presented in Section 5.2. 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. For each round we train the number of epochs as indicated in Table 5. Table 5 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 staged training with only 10 epochs in each round, and once for staged training with 40 epochs in each round. The green highlight indicates an improvement of the 40 epochs over the 10 epochs staged training.

	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-PERM	0.59M	20.37M	4.20M	77.73k	705.44k	135.91k	16s	312s

Table 4. 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.

Likely, these results would improve further when extending the training epochs to 200, or training with a larger amount of data.

7 Our Best Distinguishers

Simon and Speck For SPECK32, we retrieve the optimal input difference used in Gohr's paper, with similar accuracies. In addition, we find an 8-round neural distinguisher for SPECK64 with accuracy 0.5369, and a 10-round distinguisher with accuracy 0.5928 for SPECK128. Our SPECK128 distinguisher is new, whereas our SPECK128 distinguisher obtains the same number of rounds, but slightly lower accuracy than the one of Hou [17]; on the other hand, Hou uses the MOD model with 64 pairs in each sample, so that the network has vastly more data to classify. Interestingly, the best differential characteristic for SPECK128 given in [24] 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 [3], one can prepend the input difference $(0x820200, 0x1202)$, which propagates to our best neural distinguisher $(0x80, 0x8000000000000000)$ after 2 rounds with probability 2^{-6}. An additional round can be added at the start, yielding a 13 rounds distinguisher.

		10 epochs			40 epochs		
cipher	difference	best round	best acc.	best NN	best round	best acc.	best NN
SIMON32	0x400	11	0.5150	DBitNet	11	0.5166	DBitNet
SPECK32	0x400000	8	0.5103	DBitNet	8	0.5114	DBitNet
HIGHT	0x800000000000	10	0.7509	DBitNet	10	0.7509	DBitNet
HIGHT$^{\text{RK}}$	0x800000000... [(a)]	13	0.9647	DBitNet	14	0.5633	DBitNet
PRESENT	0xd000000	8	0.5122	DBitNet	8	0.5120	DBitNet
SIMON64	0x8000	13	0.5179	DBitNet	13	0.5182	DBitNet
SPECK64	0x8080000000	8	0.5332	Gohr-D1	8	0.5369	DBitNet
TEA	0x4000000000000000	5	0.5562	Gohr-D1	5	0.5629	DBitNet
XTEA	0x4000000000000000	5	0.5302	DBitNet	5	0.5978	DBitNet
LEA	0x800000008... [(b)]	11	0.5115	DBitNet	11	0.5113	DBitNet
SIMON128	0x8000000	20	0.5074	DBitNet	20	0.5069	DBitNet
SPECK128	0x8080000000000000	10	0.5928	DBitNet	10	0.5913	DBitNet
GIMLI-PERM	0x800000000... [(c)]	11	0.5237	DBitNet	11	0.5238	DBitNet

RK: Related key setting [(a)]: 0x80000000000000000000000000000000000000000800000
[(b)]: 0x80000000800000008004000080
[(c)]: 0x80000000000000000000000000004000000000000000000000000000000800000000000000000000000000

Table 5. Summary of the best distinguishers for each target cipher for staged training with 10 epochs per round, respectively 40 epochs per round. The detailed round-by-round results for 40 epochs are shown in Table 6.

For SIMON32, we obtain similar results to [3], albeit with a significantly simpler training pipeline, and less computations (Section 6.3). For SIMON64, interestingly, we reach one more round than [17], even though [17] uses 64 pairs. For SIMON128, we find a new 20 rounds distinguisher.

Gimli In order to evaluate our approach against [2], we studied the GIMLI permutation, and reached 11 rounds with accuracy 0.5238, which is 3 rounds more than [2]. This is explained by the choice of input difference, as we obtained similar results to [2] when using the same input differences as him. This result is to be compared with the design document of GIMLI [7], which finds at best a differential characteristic with probability 2^{-188} on 12 rounds, as well as the 12-round linear distinguisher with complexity 2^{-198} and the 15-round differential-linear distinguisher with complexity $2^{-87.4}$ presented in [10]. Of course, the full-round symmetry distinguishers [10] remains much stronger.

HIGHT We obtain a 10 rounds single-key distinguisher on HIGHT, as well as a 14 rounds related-key distinguisher. The initial paper for HIGHT [16] mentions that no differential characteristics for more than 11 rounds should be useful for an attack. Furthermore, it presents a related-key attack using a differential characteristic with probability 2^{-23}. The initial paper 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. Such a property would require $C \cdot 256$ (with C a small constant) to permit a reliable distinguisher. On the other hand, our neural distinguisher requires a single pair.

Present For PRESENT, we find an 8-round distinguisher. In comparison, the best differential characteristic for PRESENT reduced to 8 rounds has probability 2^{-32} [25].

TEA and XTEA For both TEA and XTEA, we find distinguishers for 5 rounds; interestingly, they share the same input difference. For TEA, we do not reach as many rounds as [5], possibly because we are working in a different model, where our difference is injected by exclusive or, whereas they study modular addition-based differences.

LEA For LEA, we find an 11 rounds distinguisher. The original paper [15] presented 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. Our automated pipeline is designed to capture such properties easily; we indeed observe that, 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 at round 1.

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

8 Conclusion

In this paper, we tackle the problem of generalizing neural distinguishers to different ciphers, and present a generic framework that can be applied out-of-the-box to new ciphers. This framework relies on a more generic neural network structure, enabled by the use of dilated convolutional layers, as well as generic choices of parameters such as the learning rate. In addition, we solve the difficulty of choosing a good input difference through an evolutionary optimizer, and apply it to a variety of ciphers.

Through a series of experiments, we show that our framework is able to match, or beat, the state-of-the-art for neural distinguishers, as well as to find good ones for primitives that had not been studied yet. This study produced a

large number of input differences with good properties for neural distinguishers, besides the ones presented in these paper; these are of independent interest and will be made available at the time of publication.

The main advantage of the tool is that it is fully automated, and can readily be used for any primitive, as long as an encryption function is provided.

Preliminary experiments show that our framework is able to find good input differences in the related-key setting, but their exploitation requires significant effort and is left for future work. A promising research direction will be to look for efficient ways to combine good input differences, in particular the almost-equivalent ones for SIMON, for instance through MOD-type distinguishers [28], to improve existing results.

References

1. Bacuieti, N., Batina, L., Picek, S.: Deep neural networks aiding cryptanalysis: A case study of the speck distinguisher. In: Ateniese, G., Venturi, D. (eds.) Applied Cryptography and Network Security - 20th International Conference, ACNS 2022, Rome, Italy, June 20-23, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13269, pp. 809–829. Springer (2022). https://doi.org/10.1007/978-3-031-09234-3_40, https://doi.org/10.1007/978-3-031-09234-3_40
2. Baksi, A., Breier, J., Chen, Y., Dong, X.: Machine learning assisted differential distinguishers for lightweight ciphers. In: Design, Automation & Test in Europe Conference & Exhibition, DATE 2021, Grenoble, France, February 1-5, 2021. pp. 176–181. IEEE (2021). https://doi.org/10.23919/DATE51398.2021.9474092, https://doi.org/10.23919/DATE51398.2021.9474092
3. Bao, Z., Guo, J., Liu, M., Ma, L., Tu, Y.: Enhancing differential-neural cryptanalysis. In: ASIACRYPT'22. Springer-Verlag (2022), https://eprint.iacr.org/2021/719.pdf
4. Beaulieu, R., Treatman-Clark, S., Shors, D., Weeks, B., Smith, J., Wingers, L.: The simon and speck lightweight block ciphers. In: 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC). pp. 1–6 (2015). https://doi.org/10.1145/2744769.2747946
5. Bellini, E., Rossi, M.: Performance comparison between deep learning-based and conventional cryptographic distinguishers. In: Arai, K. (ed.) Intelligent Computing. pp. 681–701. Springer International Publishing, Cham (2021)
6. Benamira, A., Gérault, D., Peyrin, T., Tan, Q.Q.: A deeper look at machine learning-based cryptanalysis. In: Canteaut, A., Standaert, F. (eds.) 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. Lecture Notes in Computer Science, vol. 12696, pp. 805–835. Springer (2021). https://doi.org/10.1007/978-3-030-77870-5_28, https://doi.org/10.1007/978-3-030-77870-5_28
7. Bernstein, D.J., Kölbl, S., Lucks, S., Massolino, P.M.C., Mendel, F., Nawaz, K., Schneider, T., Schwabe, P., Standaert, F.X., Todo, Y., Viguier, B.: Gimli : A cross-platform permutation. In: Fischer, W., Homma, N. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2017. pp. 299–320. Springer International Publishing, Cham (2017)

8. Blalock, D., Ortiz, J.J.G., Frankle, J., Guttag, J.: What is the State of Neural Network Pruning? In: Proceedings of the 3rd MLSys Conference. Austin, TX, USA (mar 2020), http://arxiv.org/abs/2003.03033
9. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: Present: An ultra-lightweight block cipher. In: Paillier, P., Verbauwhede, I. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2007. pp. 450–466. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
10. Flórez Gutiérrez, A., Leurent, G., Naya-Plasencia, M., Perrin, L., Schrottenloher, A., Sibleyras, F.: New results on gimli: Full-permutation distinguishers and improved collisions. In: Moriai, S., Wang, H. (eds.) Advances in Cryptology – ASIACRYPT 2020. pp. 33–63. Springer International Publishing, Cham (2020)
11. Gohr, A.: Deep speck. https://github.com/agohr/deep_speck (2019)
12. Gohr, A.: Improving attacks on round-reduced speck32/64 using deep learning. In: Boldyreva, A., Micciancio, D. (eds.) Advances in Cryptology – CRYPTO 2019. pp. 150–179. Springer International Publishing, Cham (2019)
13. Goodfellow, I., Bengio, Y., Courville, A.: Deep learning: The MIT Press, vol. 19. The MIT Press (2017), https://mitpress.mit.edu/books/deep-learning
14. He, K., Zhang, X., Ren, S., Sun, J.: Deep Residual Learning for Image Recognition. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition pp. 770–778 (dec 2015). https://doi.org/10.1109/CVPR.2016.90, http://arxiv.org/abs/1512.03385
15. Hong, D., Lee, J.K., Kim, D.C., Kwon, D., Ryu, K.H., Lee, D.G.: Lea: A 128-bit block cipher for fast encryption on common processors. In: Kim, Y., Lee, H., Perrig, A. (eds.) Information Security Applications. pp. 3–27. Springer International Publishing, Cham (2014)
16. Hong, D., Sung, J., Hong, S., Lim, J., Lee, S., Koo, B.S., Lee, C., Chang, D., Lee, J., Jeong, K., Kim, H., Kim, J., Chee, S.: Hight: A new block cipher suitable for low-resource device. In: Goubin, L., Matsui, M. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2006. pp. 46–59. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
17. Hou, Z., Ren, J., Chen, S., Fu, A.: Improve neural distinguishers of simon and speck. Sec. and Commun. Netw. **2021** (jan 2021). https://doi.org/10.1155/2021/9288229, https://doi.org/10.1155/2021/9288229
18. Matsui, M.: On correlation between the order of s-boxes and the strength of des. In: Workshop on the Theory and Application of of Cryptographic Techniques. pp. 366–375. Springer (1994)
19. Mitchell, T.M., Mitchell, T.M.: Machine learning, vol. 1. McGraw-hill New York (1997)
20. Oord, A.v.d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., Kavukcuoglu, K.: Wavenet: A generative model for raw audio. arXiv preprint arXiv:1609.03499 (2016)
21. Reddi, S.J., Kale, S., Kumar, S.: On the convergence of adam and beyond. arXiv preprint arXiv:1904.09237 (2019)
22. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. nature **323**(6088), 533–536 (1986)
23. Samuel, A.L.: Machine learning. The Technology Review **62**(1), 42–45 (1959)
24. Song, L., Huang, Z., Yang, Q.: Automatic differential analysis of arx block ciphers with application to speck and lea. In: Liu, J.K., Steinfeld, R. (eds.) Information Security and Privacy. pp. 379–394. Springer International Publishing, Cham (2016)

25. Wang, M.: Differential cryptanalysis of present. IACR Cryptol. ePrint Arch. **2007**, 408 (2007)
26. Wheeler, D.J., Needham, R.M.: Tea, a tiny encryption algorithm. In: Preneel, B. (ed.) Fast Software Encryption. pp. 363–366. Springer Berlin Heidelberg, Berlin, Heidelberg (1995)
27. Wheeler, D.J., Needham, R.M.: Tea extensions (1997)
28. Yadav, T., Kumar, M.: 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. p. 191–212. Springer-Verlag, Berlin, Heidelberg (2021). https://doi.org/10.1007/978-3-030-88238-9_10, https://doi.org/10.1007/978-3-030-88238-9_10
29. Yu, F., Koltun, V.: Multi-scale context aggregation by dilated convolutions. arXiv preprint arXiv:1511.07122 (2015)

cipher	round	Gohr depth-1		DBitNet	
		(1)	(2)	(1)	(2)
SIMON32	8	0.7400	0.7823	0.8335	0.8312
	9	0.6073	0.6249	0.6560	0.6559
	10	0.5414	0.5547	0.5599	0.5616
	11	≤ 0.5050	≤ 0.5050	0.5164	0.5166
SIMON64	9	0.9467	0.9447	0.9619	0.9582
	10	0.7710	0.7788	0.8096	0.8104
	11	0.6411	0.6348	0.6578	0.6591
	12	0.5479	0.5471	0.5623	0.5632
	13	0.5002	0.5035	0.5154	0.5182
	14	≤ 0.5050	≤ 0.5050	≤ 0.5050	≤ 0.5050
SIMON128	14	0.9010	0.9007	0.9267	0.9312
	15	0.7975	0.7966	0.8384	0.8383
	16	0.6867	0.6857	0.7249	0.7248
	17	0.5957	0.5950	0.6259	0.6259
	18	0.5390	0.5379	0.5582	0.5580
	19	0.5077	0.5072	0.5222	0.5218
	20	≤ 0.5050	≤ 0.5050	0.5060	0.5069
SPECK32	5	0.9269	0.9271	0.9280	0.9260
	6	0.7860	0.7867	0.7873	0.7867
	7	0.6111	0.6120	0.6152	0.6098
	8	≤ 0.5050	≤ 0.5050	0.5107	0.5114
SPECK64	4	0.9999	0.9999	0.9998	0.9998
	5	0.9884	0.9870	0.9939	0.9914
	6	0.8580	0.8494	0.9229	0.9230
	7	0.6679	0.6198	0.7182	0.7198
	8	0.5256	0.5158	0.5357	0.5369
	9	≤ 0.5050	≤ 0.5050	≤ 0.5050	≤ 0.5050
SPECK128	7	0.9995	0.9995	0.9994	0.9994
	8	0.9722	0.9716	0.9860	0.9860
	9	0.7787	0.7800	0.8296	0.8293
	10	0.5814	0.5831	0.5913	0.5909
	11	≤ 0.5050	≤ 0.5050	≤ 0.5050	≤ 0.5050
HIGHT	8	0.9990	0.9990	0.9990	0.9990
	9	0.7500	0.8525	0.8598	0.8600
	10	0.5617	0.5003	0.7509	0.7509
	11	≤ 0.5050	≤ 0.5050	≤ 0.5050	≤ 0.5050
HIGHT^{RK}	12	0.9990	0.9990	0.9990	0.9990
	13	0.9647	0.7499	0.9647	0.9647
	14	≤ 0.5050	≤ 0.5050	≤ 0.5050	0.5633
	15	≤ 0.5050	≤ 0.5050	≤ 0.5050	≤ 0.5050
PRESENT	5	0.8808	0.8785	0.8828	0.8829
	6	0.7077	0.7053	0.7093	0.7096
	7	0.5597	0.5593	0.5613	0.5612
	8	0.5104	0.5106	0.5106	0.5120
	9	≤ 0.5050	≤ 0.5050	≤ 0.5050	≤ 0.5050
TEA	3	1.0000	1.0000	1.0000	1.0000
	4	0.8864	0.8747	0.9079	0.9079
	5	0.5562	0.5491	0.5629	0.5619
	6	≤ 0.5050	≤ 0.5050	≤ 0.5050	≤ 0.5050
XTEA	3	1.0000	1.0000	1.0000	1.0000
	4	0.8867	0.8748	0.9700	0.9697
	5	≤ 0.5050	0.5093	0.5978	≤ 0.5050
	6	≤ 0.5050	≤ 0.5050	≤ 0.5050	≤ 0.5050
LEA	8	0.8475	0.8482	0.8473	0.8477
	9	0.7209	0.7200	0.7233	0.7231
	10	0.5952	0.6010	0.5963	0.5957
	11	0.5111	0.5112	0.5113	0.5113
GIMLI-PERM	8	0.9995	0.9995	0.9987	0.9988
	9	0.8735	0.8707	0.8639	0.8735
	10	0.6129	0.6041	0.6052	0.6037
	11	≤ 0.5050	≤ 0.5050	0.5238	0.5236
	12	≤ 0.5050	≤ 0.5050	≤ 0.5050	≤ 0.5050

Table 6. Results for all target ciphers round by round. The best accuracy in each row is highlighted.