Deep neural networks aiding cryptanalysis: A case study of the Speck distinguisher

Nicoleta-Norica Băcuieți[1], Lejla Batina[2], and Stjepan Picek[2,3]

[1] ETH Zurich, Switzerland
[2] Radboud University, The Netherlands
[3] Delft University of Technology, The Netherlands

Abstract. At CRYPTO'19, A. Gohr proposed neural distinguishers for the lightweight block cipher Speck32/64, achieving better results than the state-of-the-art at that point. However, the motivation for using that particular architecture was not very clear, leading us to investigate whether a smaller and/or better performing neural distinguisher exists. This paper studies the depth-10 and depth-1 neural distinguishers proposed by Gohr [7] with the aim of finding out whether smaller or better-performing distinguishers for Speck32/64 exist.
We first evaluate whether we can find smaller neural networks that match the accuracy of the proposed distinguishers. We answer this question in affirmative with the depth-1 distinguisher successfully pruned, resulting in a network that remained within one percentage point of the unpruned network's performance. Having found a smaller network that achieves the same performance, we examine if its performance can be improved as well. We also study whether processing the input before giving it to the pruned depth-1 network would improve its performance. To this end, convolutional autoencoders were found that managed to reconstruct the ciphertext pairs successfully, and their trained encoders were used as a preprocessor before training the pruned depth-1 network. We found that, even though the autoencoders achieve a perfect reconstruction, the pruned network did not have the necessary complexity anymore to extract useful information from the preprocessed input, motivating us to look at the feature importance to get more insights. To achieve this, we used LIME, with results showing that a stronger explainer is needed to assess it correctly.

Keywords: Neural distinguisher, Feature importance, Speck, Pruning

1 Introduction

Traditional symmetric cryptanalysis shows small improvements over time, and people started considering alternative ways to improve it. Since deep learning has recently attracted much attention due to the significant advances in research areas such as computer vision and speech recognition, it did not take long until researchers also started to consider Deep Neural Networks (DNNs) in the area of

cryptography. DNNs are a family of non-linear machine learning classifiers that, given a dataset and a loss function, try to learn the optimal hyperparameters minimizing the loss. Using DNNs, A. Gohr was the first to achieve better results than that time's state-of-the-art, revolutionizing cryptanalysis, i.e., the study of cryptographic systems with the purpose of finding weaknesses, [7]. Encouraged by Gohr's results, more papers followed that built upon his work, e.g. [9].

Starting from the Gohr's neural networks, the purpose of this paper is to investigate whether there exists a smaller or better-performing neural network for executing a better *distinguishing attack*. Generally, in a distinguishing attack against a cryptographic primitive (a cipher in our case), the adversary tries to *distinguish* between (or *classify*) encrypted data and random data, thus helping in the cryptanalysis of the cipher. Specifically, if an adversary manages to distinguish the output of a cipher from random data faster than a brute force key search, this is considered a break for the cipher. Thus, the cipher cannot be considered secure enough for ensuring the confidentiality of the encrypted information. These distinguishing attacks can be *differential*, in which case we talk about differential cryptanalysis, that is, cryptanalysis with regards to bitwise differences in the inputs given to the cipher [5]. In a differential attack, the non-random properties of the ciphertext pair produced by the cipher when given a plaintext pair with some known input difference are exploited for various purposes, one of which is distinguishing. Those differential attacks further branch into *purely differential attacks*, where the adversary uses only the ciphertext pair's bitwise difference, and *general differential attacks*, where the information from the complete ciphertext pair is used [7]. Our work will focus on general differential distinguishing attacks on the lightweight iterated block cipher Speck32/64 achieved by neural networks.

Motivation: While the application of neural networks in cryptanalysis evidently brings good practical results, it is also important to provide some theoretical support. Otherwise, the improvements make limited sense, as one cannot obtain guidance for the design and analysis of cryptanalytic primitives. Thus, it becomes important to study the behavior of neural network distinguishers and the interpretability and explainability of such solutions. Unfortunately, the deep learning explainability is a difficult problem that is not solved in general. Still, some observations are possible, especially from the perspective of the neural network size and the feature importance.

Recently, either a rather sophisticated technique exploited Speck's internal state values obtained through brute force key search [4] or a model that required k times more data was deployed [10]. On the contrary, to make the analysis simpler, our paper remains in the low data setting. Concretely, only the plaintext inputs and ciphertext outputs are known. In addition, the same training/test size and data format as in Gohr's work is kept for comparison. The focus will be on the distinguisher's network to see whether a smaller and/or better performing neural distinguisher exists. Concretely, we want to find out whether:

1. A smaller, equally-good-performing distinguisher can be obtained by systematically pruning Gohr's distinguishers to the bare minimum needed to achieve their current performance.
2. Preprocessing the input will improve the performance of Gohr's (prunned) distinguishers.

Main contributions: We show that the state-of-the-art on neural distinguisher can be improved and that there are still lots of avenues to explore. We demonstrate these with the following contributions which, to the best of our knowledge, are the first studies in the setting of neural differential distinguishers.

1. We evaluate the Lottery Ticket Hypothesis [6] on neural Speck distinguishers to see whether a smaller or better-performing network can be obtained, finding out that this is the case. Indeed, the Lottery Ticket Hypothesis states there are subnetworks that match or even outperform the accuracy of the original network. To obtain such subnetworks, we conduct pruning based on average activations equal to zero.
2. We successfully strip the currently best neural distinguisher for Speck (the depth-1 distinguisher), presenting a smaller network whose accuracy remains around one percentage point of the depth-1 distinguisher's.
3. We successfully train autoencoders that achieve a nearly perfect reconstruction of the given ciphertext pairs and study the performance of the proposed (and pruned) Speck distinguishers when autoencoders do a prior feature engineering.
4. We study the importance of the inputs using Local Interpretable Model-agnostic Explanations (LIME) [15] to gain insights into the (pruned) distinguishers' behavior, which might aid in the improvement of future preprocessing methods.

2 The Speck family of block ciphers

2.1 Notations and conventions

In this paper, the bitwise eXclusive-OR operation will be denoted by $\oplus$, the bitwise AND operation by $\wedge$, modular addition modulo 2^n by $\boxplus$, a left or right bitwise rotation by $\ll$ and $\gg$, respectively, and the concatenation of two-bit strings a and b will be denoted by $a \parallel b$. Furthermore, the Hamming weight of a bit string is given by the number of ones present in it.

2.2 Speck block cipher

The lightweight iterated block cipher Speck was designed by Beaulieu et al. for the US National Security Agency (NSA) with the intent of being efficient in software implementations on micro-controllers [3]. At its core, it is comprised of three basic functions: modular **A**ddition (modulo 2^k), bitwise **R**otation, and bitwise e**X**clusive-OR of k-bit words, thus being an ARX construction. Since it

is an *iterated* block cipher, it has a round function (that is iterated), which, in the case of Speck, is a simple Feistel structure. The round function $F : \mathbb{F}_2^k \times \mathbb{F}_2^{2k}$ takes as input a k-bit subkey K and the cipher's internal state that consists of two k-bit words denoted as L_i and R_i, and computes the cipher's next internal state as:

$$L_{i+1} = ((L_i \gg \alpha) \boxplus R_i) \oplus K \tag{1}$$

$$R_{i+1} = (R_i \ll \beta) \oplus L_{i+1} \tag{2}$$

Here, i and $i+1$ represent the current, respectively, next round, and α and β are constants specific to each member of the Speck cipher family. Regarding the subkeys used, they are generated with a non-linear key schedule from a master key using the above-described round function as the main operation, but with details that change from one Speck member to another.

As the key schedule will not be studied in this paper, please refer to [3] for additional information. Concretely, for the Speck member studied in this paper, the block size n is 32 bits, the word size k is 16 bits, the key size m is 64 bits, α is 7, β is 2, and the round function is applied maximally 22 times to compute a ciphertext output from the plaintext input.

2.3 The setup

For the implementation of the Speck32/64 cipher and distinguishers studied, as well as for the algorithms needed for generating the datasets with a given input difference and evaluating the results, this paper refers to the code provided by the author of [7] here [4]. For all experiments, a training set of size 10^7, a test set of size 10^6, and a batch size of 5 000 was used as in the previous related work [7]. Finally, the experiments were run on an RTX 3090, and the code that was used to conduct the experiments can be found at here[5].

3 Related works on neural Speck distinguishers

Since the release of the lightweight block cipher Speck, differential and neural distinguishers have been used to cryptanalyze it. First, at CRYPTO'19, Gohr proposed such distinguishers, focusing on the input difference $\Delta_{in} = 0\text{x}0040/0000$ [1]. He defined *real pairs* as being ciphertext pairs (C, C') resulted from encrypting plaintext pairs (P, P') where $P \oplus P' = \Delta_{in}$, and *random pairs* being ciphertext pairs (C, C') resulted from encrypting plaintext pairs (P, P') where there is no fixed input difference. Then, he aimed to distinguish the *real pairs* from the *random pairs*, deploying several methods that are described below. In the process, the author compared the performance of a purely differential distinguisher to a neural distinguisher for 5 to 8 rounds, showing that the neural distinguisher outperforms the purely differential one. Those distinguishers were

[4] `https://github.com/agohr/deep_speck`
[5] `https://anonymous.4open.science/r/sourcecode`

denoted D_r and N_r for **d**ifferential and **n**eural distinguishers for Speck reduced to r $\in \{5, 6, 7, 8\}$ rounds, respectively. The details of the approach used are described below.

Purely differential distinguisher: First, the entire difference distribution table (DDT) of Speck for the input difference Δ_{in} was computed under the Markov assumption [13]. Then, to distinguish real ciphertext pairs from random ones, the author first assumed that random ciphertext pair differences, i.e., $\Delta_{out} = C \oplus C'$, are distributed according to the uniform distribution. Next, the author took the corresponding transition probability $P(\Delta_{in} \rightarrow \Delta_{out})$ from the DDT, classifying the ciphertext pair difference as *real*, if $P(\Delta_{in} \rightarrow \Delta_{out}) > \frac{1}{2^{32}-1}$, and as *random* otherwise. For more details, please refer to [7].

Gohr's neural distinguisher: The proposed deep neural network is a residual network consisting of three types of blocks: an *initial convolution*, *convolutional blocks*, and a *prediction head*. Concretely, they are:

1. Block 1: The initial convolution consisting of a 1D-CNN layer with kernel size 1, 32 channels, padding, and stride of size 1, followed by batch normalization and a ReLU activation layer.
2. Block 2-i: The convolutional one-to-ten residual blocks/units, each residual block consisting of two 1D-CNN layers with kernel size 3, 32 channels and padding and stride of size 1, each followed by batch normalization and a ReLU activation layer. These layers are then followed by an additional layer where the input of this block is also added to its output and passed to the input of the subsequent block. This last operation makes the block, and thus also the network, residual, the input that skips all those layers being called a residual connection.
3. Block 3: The prediction head consisting of two dense layers, having 64 neurons and followed by batch normalization and ReLU activation layer each, closing with a dense layer of one neuron using a sigmoid activation function.

The neural distinguishers give a score between 0 and 1 where a score greater than or equal to 0.5 classifies the sample as a real pair; otherwise, it is classified as random. Using this setup, neural distinguishers were trained for Speck reduced to 5 and 6 rounds, but different approaches were taken for Speck reduced to 7 and 8 rounds. For Speck reduced to 7 rounds, *key search* was used to improve the accuracy of the neural distinguisher. For more details, the method described can be found in [7].

Moving to the neural distinguisher for 8 rounds, since the previously mentioned approach did not improve this distinguisher's performance, the neural distinguisher for 8 rounds was obtained from the seven-round neural distinguisher using the staged training method. Again, more details can be found in [7].

Obtaining superior results compared to purely differential distinguishers indicated that the neural distinguishers learn more than differential cryptanalysis. It thus motivated Gohr to conduct the *the real differences experiment* with the goal to distinguish real ciphertext pairs (C, C') drawn from the real distribution (again obtained from the Δ_{in} = 0x0040/0000 difference) from masked real ciphertext pairs $(C \oplus M, C' \oplus M)$ where M is a random 32-bit value. By conduct-

ing this experiment, Gohr wanted to show that the previously obtained neural distinguishers (without retraining) offer comparable results to key search. The results for both the real-vs-random, as well as for the real differences experiment, can be found in [7].

Being inspired by Gohr's work, Benamira et al. [4] went further and developed an approach to estimate the property learned by Gohr's deep neural network. Concretely, they replaced Gohr's three building blocks with the following three steps:

1. Changing (C, C') into $I = (\Delta L, \Delta V, V_0, V_1)$, where $\Delta L = C_l \oplus C'_l$ is the addition modulo 2 between the left parts of C and C', and $V_i = L_i \oplus R_i$ is the difference between the two parts of the internal state at round i.
2. Changing the 512-feature vector [4] of the DNN into a feature vector of probabilities $F = (P(Real \mid I_{M1}) P(Real \mid I_{M2}) \cdots P(Real \mid I_{Mm}))^T$.
3. Changing the final dense layer of the third building block into the Light Gradient Boosting Machine (LGBM) [12] model.

The authors defined an *output distribution table* (ODT) directly on the values $(\Delta L, \Delta V, V_0, V_1)$ instead of the DDT of the ciphertext pair difference $(C_l \oplus C'_l, C_r \oplus C'_r)$. Then, they used the ODT to define a *masked output distribution table* (M-ODT). This M-ODT is a compressed ODT where the input is not $I = (\Delta L, \Delta V, V_0, V_1)$, but $I_M = (\Delta L \wedge M_1, \Delta V \wedge M_2, V_0 \wedge M_3, V_1 \wedge M_4)$, where $M \in \mathcal{M}_{hw}$, $M = (M_1, M_2, M_3, M_4)$ is an ensemble of four 16-bit masks, each having the Hamming weights hw (later set to 16 and 18). Then, by considering several masks, they defined the set of relevant masks of $\mathcal{M}_{hw}$ as R_M, being able to compute for each input I the probability $P(Real \mid I_M)$, $\forall M \in R_M$ [4]. Having those defined, they developed a three-step approach for recognizing output of Speck reduced to 5 and 6 rounds as follows:

1. Extract the masks from Gohr's DNN with dataset 1.
2. Construct the M-ODT with dataset 2.
3. Train the LGBM classifier from the probabilities stored in the M-ODT with dataset 3.

Through this approach, they obtained results similar to Gohr's DNN, thus showing that they have successfully modeled the DNN's property. The results can be seen in [4].

They concluded by explaining how to improve over Gohr's results by means of creating batches of ciphertext inputs instead of pairs. They used two approaches for training and evaluating the M-ODT distinguisher: one where each element of the batch is given a score by the distinguisher and then takes the median of the results, and the other one where the whole batch is considered as a single input. For both methods, they obtained a 100% accuracy on 5 and 6 rounds, as well as on 7 rounds with the first method [4].

More recently, taking inspiration from both the works mentioned above, Zezhou Hou et al. [10] first developed an algorithm based on SAT, which returns input differences of high-probability differential characteristics. They proposed an alternative format for the training and test data, where they would group k ciphertext differences in a matrix and regard it as one sample, and they use

this type of sample to train the ResNet. Concretely, they tried it either with the same input difference as Gohr or a better one as chosen by their SAT-based algorithm. Using this new data format in combination with the input differences suggested by their algorithm, they managed to obtain an accuracy of 88.19 % and 56.49% for Speck reduced to 7 and 8 rounds, respectively, which is superior to Gohr's. More details can be found in [10].

4 The network under lens

First, Gohr's network containing ten blocks of type 2 and its performance on Speck reduced to 7 and 8 rounds will be examined. Following this, the Lottery Ticket Hypothesis using two different pruning methods: one-shot pruning and iterative pruning will be evaluated for this depth-10 distinguisher, analyzing the results. After that, Gohr's best network, the depth-1 distinguisher, containing one block of type 2, will be examined in detail to see whether even this already small network can be further pruned. For this purpose, the Lottery Ticket Hypothesis will be evaluated for this network, followed by a computation of the average percentage of activations equal to zero and pruning of the network.

4.1 The initial network

First, we aim to reproduce the results given in [7] with the depth-10 neural distinguisher for Speck reduced to 5 and 6 rounds. In addition, we also want to see its performance for Speck reduced to 7 and 8 rounds by following the same training method as opposed to the approaches used in [7]. After having trained and evaluated the distinguishers five times, the results can be seen in Table 1.

Distinguisher	Accuracy	TPR	TNR
N_5	$0.927 \pm 1.46 \times 10^{-4}$	$0.901 \pm 3.92 \times 10^{-4}$	$0.953 \pm 5.86 \times 10^{-4}$
N_6	$0.787 \pm 3.90 \times 10^{-4}$	$0.719 \pm 9.66 \times 10^{-4}$	$0.855 \pm 7.45 \times 10^{-4}$
N_7	$0.611 \pm 4.17 \times 10^{-4}$	$0.551 \pm 1.98 \times 10^{-3}$	$0.671 \pm 1.90 \times 10^{-3}$
N_8	$0.500 \pm 7.53 \times 10^{-5}$	$0.368 \pm 3.55 \times 10^{-1}$	$0.632 \pm 3.55 \times 10^{-1}$

Table 1: Accuracies of the depth-10 Neural distinguishers for Speck32/64 reduced to 5, 6, 7, and 8 rounds in the real-vs-random experiment.

From these results, one can see that for Speck reduced to 5 and 6 rounds, the results could be reproduced. What is more, for Speck reduced to 7 rounds, the distinguisher gave a similar accuracy to the one in [7], where the author used the approach mentioned in Section 3. Perhaps, the more sophisticated approach [7] was used more for seeing whether it would improve the distinguisher's accuracy, but since the improvement is insignificant, we will use the same training approach as for the first two distinguishing cases. When looking at the N_8 distinguisher, the improvement achieved by using the approach mentioned in Section

3 managed to make the neural distinguisher slightly better than the differential distinguisher. However, it is not considerable compared to the N_8 distinguisher trained using the same approach as for Speck reduced to 5 and 6 rounds. Since the N_8 distinguisher without the training approach mentioned above is no better than random guessing, even though results will be given for it as well, the decisions will be based on the results of the other three distinguishers.

With this, we turn to the next section, where we will look at the Lottery Ticket Hypothesis and the results obtained by effectuating the steps needed for evaluating it for the depth-10 and depth-1 version of this distinguisher.

4.2 The Lottery Ticket Hypothesis

The *Lottery Ticket Hypothesis* (LTH) was first proposed by Frankle and Carbin in [6]. It was proposed after finding that appropriately initialized pruned networks are capable of training effectively while achieving a comparable accuracy to the original network in a similar number of training epochs. It reads as follows:

> *A randomly initialized dense neural network contains a subnetwork initialized such that - when trained in isolation - it can match the test accuracy of the original network after training for at most the same number of iterations.*

Thus, the reasons behind evaluating the LTH is to see whether:

1. Some subnetworks perform similar to the baseline network for each of the four distinguishers, and how much the performance decreases as the network becomes more sparse. The goal is to get an idea of the trade-off between the network's size and its performance.
2. There are winning tickets and whether their performance is significantly better than the baseline network.
3. Similar conclusions to the ones in [6] can be drawn. Those are:
   – Iterative pruning finds winning tickets that match the accuracy of the baseline network at smaller network sizes than does one-shot pruning.
   – Winning tickets are 10% (or less) to 20% of the baseline network's size.

As mentioned, these subnetworks are obtained by pruning, and in the following subsections, two pruning strategies will be put under test for the LTH: *one-shot pruning* and *iterative pruning*.

The winning tickets: According to [6], after having pruned the trained baseline network of the smallest-magnitude weights, we are ready to define what a *winning ticket* is. As defined in [6], a *winning ticket* is a subnetwork that, when trained in isolation after having had the remaining weights reinitialized with the weights of the baseline network *prior to training*, will provide classification accuracy equivalent or superior to the baseline network's.

Frankle and Carbin have repeated the experiments with random initialization of the pruned network. However, the randomly initialized pruned network no longer matched the trained (unpruned) baseline network's performance evidentiating that the pruned networks need to be appropriately initialized. Therefore,

the pruning strategies will be defined to reinitialize the remaining weights of the pruned network to the weights of the unpruned network prior to training.

One-shot pruning: In one-shot pruning, the baseline network is trained once, $p\%$ of the weights are pruned, and then, the remaining weights are reinitialized to the weights of the baseline network prior to training. The process will be repeated for several values of $p\%$ to see the possible changes in the performance of the distinguisher. Please refer to [6] for the pseudocode and further details.

Iterative pruning: In iterative pruning, we again start from a baseline network that is trained once. But unlike in one-shot pruning where, for each time we repeat the process with a different value of $p\%$, we start from the same pretrained weights θ_0. At each pruning trial $i \in \{1, t\}$, $p\%$ of the *remaining* weights are pruned. Again, since there is no indication for what an appropriate value for $p\%$ would be, one would have to try out a few values. However, if the improvement of the winning tickets' performance will not be significant, the experiments will be run just for one value of $p\%$. For more details, please refer to [6].

Results: In this subsection, the results that were obtained by evaluating the LTH for the N_5, N_6, N_7, and N_8 distinguishers based on the depth-10 neural network are given. Experiments with both one-shot pruning (depicted in *yellow*) as well as iterative pruning (depicted in *green*) were conducted and compared to the results obtained for the (unpruned) baseline model (depicted in ***black***). Specifically, for each distinguisher, the accuracy, true positive rate (TPR), and the true negative rate (TNR) per pruning trial were computed and compared to those obtained with the baseline network. The latter two are computed for assessing whether the obtained accuracy is a result of the classifier being biased towards one class or is a result of the classifier having learned something from the data. The experiment was run five times per pruning ratio for each pruning method, the results were averaged, and the minimum and maximum at each pruning trial were indicated. The accuracies can be seen in Figure 1.

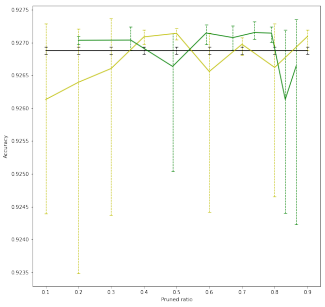
For both pruning methods, there were 9 pruning trials, where:

1. For one-shot pruning: 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, and 90%, respectively, of the network was pruned.
2. For iterative pruning: 20% of the remaining network's weights was pruned per trial.

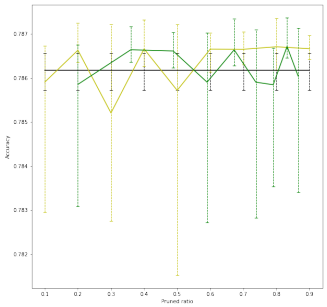
Looking at the results of the four distinguishers, two things can be observed immediately: at least up to 90% of the depth-10 network can be pruned without losing (on average) performance, and there are winning tickets that even slightly outperform (on average) the baseline network. Therefore, it can be empirically confirmed that the LTH does indeed find subnetworks (winning tickets) that will provide classification accuracy equivalent or superior to the baseline network.

Then, looking at the findings of the authors in [6], they have found that iterative pruning finds winning tickets that match the accuracy of the baseline network at smaller network sizes than does one-shot pruning. In addition, they

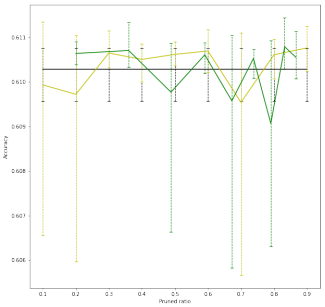
have also found that the winning tickets are 10% (or less) to 20% of the baseline network's size. These findings could not be entirely confirmed by the results above nor by the ones presented in the next subsection where the depth-1 network (which can be regarded as an already 90% pruned version of this depth-10 network) is examined. First, iterative pruning does not seem to be superior to one-shot regarding finding winning tickets that match the accuracy of the baseline network at smaller network sizes. Looking at the results, iterative pruning is either outperformed by one-shot pruning or is just barely outperforming one-shot pruning. Perhaps more trials per pruning ratio are needed to be firm in this sense. However, since iterative pruning is, as noted by the authors of [6] costly, the conclusion is left at both pruning methods performing similarly concerning finding winning tickets at smaller network sizes. Second, the results presented above and in the next subsection show that winning tickets can be found, in general, at every pruning ratio by both pruning methods.



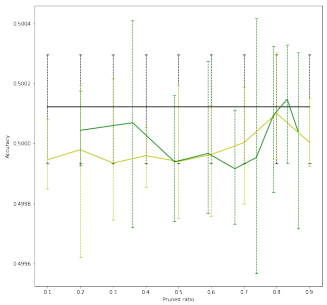
(a) The *accuracy* of N_5 after pruning $p\%$ of the network.



(b) The *accuracy* of N_6 after pruning $p\%$ of the network.



(c) The *accuracy* of N_7 after pruning $p\%$ of the network.



(d) The *accuracy* of N_8 after pruning $p\%$ of the network.

Fig. 1: The accuracys obtained after evaluating the LTH for the depth-10 N_5, N_6, N_7, and N_8 distinguishers.

While, on the one hand, we have looked at whether similar conclusions to the ones in [6] could be drawn, on the other hand, some of their conclusions were directly considered when running the experiments. First, while the globally smallest-magnitude weights were pruned in both pruning methods, the authors also experimented with pruning the smallest weights per layer with the same ratio, finding that for ResNet-18 and VGG-19, global pruning finds smaller winning tickets. They explained that some layers have far more parameters than others and that when all layers are pruned with the same ratio, the smaller layers become bottlenecks. Since in the depth-10 baseline network, some layers presented a similar difference of parameters as in the network studied by them, the experiments were run directly with the globally smallest-magnitude weights pruning approach to avoid the pitfall of having such bottlenecks.

Second, the authors have also found that, the value from which the learning rate starts matters for the LTH's success. When starting from a higher learning rate for Resnet-18 and VGG-19, the performance of the networks obtained with iterative pruning was no better than that of randomly reinitialized pruned networks (random guessing). However, they have found that at a lower learning rate, the subnetworks remain within one percentage point of the baseline network's accuracy. Although they do not give intuition behind this result, since the point of the LTH is to find subnetworks that match or outperform the baseline network's performance, it makes sense to choose a (lower) learning rate that would allow the model to learn a more optimal set of weights. The baseline network proposed by Gohr already started with a small learning rate of 0.002 which further decreased to 0.0001, and as seen, iterative pruning did indeed find winning tickets.

4.3 The smaller network

Having seen that at least 90% of the depth-10 network can be pruned (even with some minor improvement on average), we now turn to the best network Gohr has found, namely, the version with only one block of type 2, the depth-1 network. Again, first, we will try to reproduce the results Gohr obtained for Speck reduced to 5 and 6 rounds, and then see the distinguisher's performance on Speck reduced to 7 and 8 rounds using the same training approach as for the first two distinguishing cases. After having trained and evaluated the distinguishers five times, the results can be seen in Table 2.

Distinguisher	Accuracy	TPR	TNR
N_5	$0.927 \pm 1.46 \times 10^{-4}$	$0.897 \pm 1.06 \times 10^{-3}$	$0.954 \pm 8.45 \times 10^{-4}$
N_6	$0.783 \pm 1.39 \times 10^{-4}$	$0.717 \pm 1.34 \times 10^{-3}$	$0.850 \pm 1.11 \times 10^{-3}$
N_7	$0.608 \pm 9.91 \times 10^{-4}$	$0.542 \pm 3.99 \times 10^{-3}$	$0.674 \pm 4.56 \times 10^{-3}$
N_8	$0.500 \pm 1.52 \times 10^{-4}$	$0.51 \pm 1.92 \times 10^{-1}$	$0.489 \pm 1.92 \times 10^{-1}$

Table 2: Accuracies of the depth-1 Neural distinguishers for Speck32/64 reduced to 5, 6, 7, and 8 rounds in the real-vs-random experiment.

As expected from the results of the previous subsection, as well as from the results that Gohr obtained for the depth-1 N_5 and N_6 distinguishers, the accuracy of the depth-1 N_5 and N_6 distinguishers remained similar to the depth-10 distinguisher's. What is more, the accuracy of the N_7 distinguisher decreased only around one percentage point, which can be considered as an insignificant decrease. Having seen that reducing the depth-10 network to depth-1 does not affect the distinguishers' performance significantly, the next question raised was whether the depth-1 network can be pruned even more at an insignificant performance loss. The LTH with the two pruning methods was evaluated again for this depth-1 network to determine whether this is the case. The accuracies can be seen in Figure 2. Those show, again, that one could prune even 90% of this small network without losing (on average) in terms of performance. Therefore, in the next subsections, we will look at the importance of each major part of this smaller network and how it affects the performance of the distinguishers.

4.4 How much smaller can we go?

To find the answer to this question, we will first look at the activation map of each layer of the four neural distinguishers to see how much they learn at each layer. The idea is that if we see completely black activation maps, nothing is learned at that layer so that it can be pruned entirely. However, if there is only some activation present, some channels/neurons of that layer can be pruned, and how much it can be pruned in such cases needs to be determined through experiments. In this paper, results will be given for two cases that show that the performance is insignificantly affected even when the depth-1 network is pruned significantly.

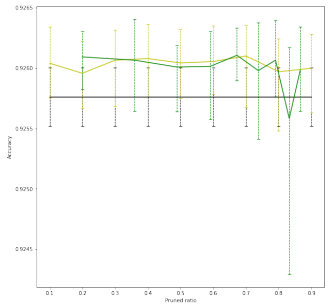
To compute the activation maps, the keract [6] [14] library was used, and they can be seen in Appendix A. For each of the four distinguishers, the five activation maps were computed, where A_1, A_2 and A_3 correspond to the activation maps of the three convolutional layers, and A_4 and A_5 correspond to the activation maps of the two dense layers. However, since the results were similar, just the activation maps of the N_5 distinguisher are given.

After examining them, the findings confirm the results obtained with the LTH, according to which even this small network can be further pruned. Concretely, the A_1 activation maps have around 13-15 channels with no activation or an insignificant number of activations, the A_2 activation maps have 12-18 of such channels, and the A_3 activation maps, 6-10 channels. Then, looking at the A_4 and A_5 activation maps, there are around 10 and 20-25 neurons, respectively, that show some activation.

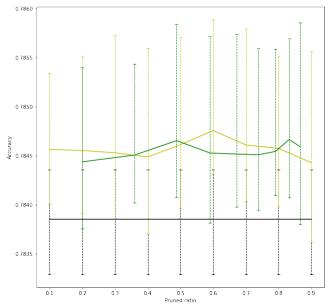
Now, even though these were the activation maps for just one input value each, the results were similar for all three distinguishers, so we go to the next step where we prune. As mentioned, besides having some maps with no activation, which will not influence the performance of the distinguishers, some maps have almost no activation but with a/some large activation value/s. First, to confirm

[6] `https://pypi.org/project/keract/4.4.0/`

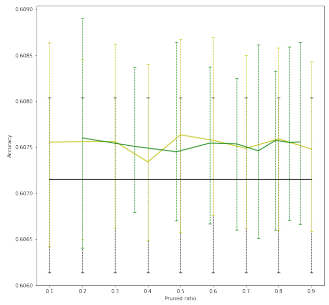
that the performance will not be affected, a network from which the minimum number of empty channels/neurons will be removed from each layer will be trained.



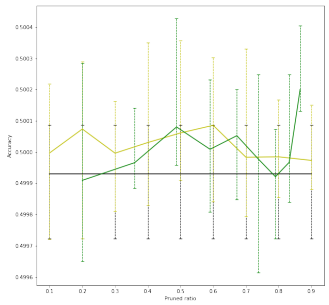
(a) The *accuracy* of N_5 after pruning $p\%$ of the network.



(b) The *accuracy* of N_6 after pruning $p\%$ of the network.



(c) The *accuracy* of N_7 after pruning $p\%$ of the network.



(d) The *accuracy* of N_8 after pruning $p\%$ of the network.

Fig. 2: The accuracys obtained after evaluating the LTH for the depth-1 N_5, N_6, N_7, and N_8 distinguishers.

Then, going to the other extreme, the maximum number of empty channels/neurons over all three distinguishers for each layer will be pruned to see how much the performance is affected and whether a finer-grained pruning approach is needed. To conduct these experiments, the kerassurgeon [7] library will be used. However, since it does not support residual connections, we will first see whether they have a significant impact on the performance of the distinguishers. After training the depth-1 distinguishers with no residual connection, the results can be seen in Table 3.

As can be seen, the accuracy of the distinguishers did not decrease, which indicates that a residual connection is not necessary. This was also expected since

[7] `https://pypi.org/project/kerassurgeon/`

Distinguisher	Accuracy	TPR	TNR
N_5	$0.925 \pm 1.44 \times 10^{-4}$	$0.897 \pm 1.30 \times 10^{-3}$	$0.954 \pm 1.24 \times 10^{-3}$
N_6	$0.784 \pm 3.35 \times 10^{-4}$	$0.714 \pm 5.68 \times 10^{-4}$	$0.855 \pm 8.07 \times 10^{-4}$
N_7	$0.608 \pm 2.55 \times 10^{-3}$	$0.542 \pm 6.00 \times 10^{-3}$	$0.671 \pm 4.94 \times 10^{-3}$
N_8	$0.500 \pm 1.36 \times 10^{-4}$	$0.57 \pm 4.48 \times 10^{-1}$	$0.43 \pm 4.50 \times 10^{-1}$

Table 3: Accuracies of the depth-1 Neural distinguishers with no residual connection for Speck32/64 reduced to 5, 6, 7, and 8 rounds in the real-vs-random experiment.

the use of residual connections is to allow training very deep neural networks. In a nutshell, those residual connections mitigate the vanishing gradients and accuracy saturation problems by allowing an alternate path for gradients to flow through and allowing the model to learn an identity function [8]. This ensures that the higher layers will perform at least as well as the lower (deeper) layers. However, since we have such a small network, the benefits of using a residual connection vanish, allowing us to eliminate it without compromising the distinguishers' performance. Since the residual connection does not impact the distinguishers' performance, we will start pruning the depth-1 network with no residual connection to see how much smaller we can go. The results can be seen in Tables 4 to 7. Using kerassurgeon, each of the four distinguishers' layers were pruned based on the **a**verage **p**ercentage **o**f activations equal to **z**ero (APoZ) described in [11]. In the tables, the accuracies for each distinguisher and the average number of channels/neurons that were pruned from each of the five layers are presented.

As suspected, the depth-1 network can be even further pruned without significantly impacting the distinguishers' performance. One can see that when the channels/neurons that had the APoZ value greater or equal to 0.7 were removed from the N_6 distinguisher, the accuracy decreased by five percentage points. In contrast, for greater cutoff values, the accuracy decreased by less than one percentage point. Now, seeing that the depth-1 network can be pruned, we will decide how much to prune the network before moving to the next section.

APoZ	Accuracy	C1	C2	C3	D1	D2
1	$0.925 \pm 1.58 \times 10^{-4}$	6.4	2	2	2.6	3.4
0.9	$0.924 \pm 1.50 \times 10^{-3}$	6.2	9.6	11	24.6	3.2
0.8	$0.920 \pm 1.76 \times 10^{-2}$	6	14	23.2	45.6	6.6
0.7	$0.904 \pm 3.05 \times 10^{-2}$	11.6	20.4	28.4	53.6	10.6

Table 4: Accuracies and pruned channels/neurons of each layer of the depth-1 Neural distinguisher with no residual connection for Speck32/64 reduced to 5 rounds in the real-vs-random experiment for different APoZ values.

APoZ	Accuracy	C1	C2	C3	D1	D2
1	$0.785 \pm 9.06 \times 10^{-4}$	6.4	6	2.2	2.2	4.4
0.9	$0.783 \pm 3.02 \times 10^{-3}$	5.4	11.2	14.4	21.8	7
0.8	$0.780 \pm 1.32 \times 10^{-2}$	3.6	15	21.8	40.4	10.4
0.7	$0.737 \pm 1.48 \times 10^{-2}$	10	20	28.6	56	30.4

Table 5: Accuracies and pruned channels/neurons of each layer of the depth-1 Neural distinguisher with no residual connection for Speck32/64 reduced to 6 rounds in the real-vs-random experiment for different APoZ values.

APoZ	Accuracy	C1	C2	C3	D1	D2
1	$0.607 \pm 3.23 \times 10^{-3}$	8.2	15	12.2	16.2	27.8
0.9	$0.608 \pm 1.23 \times 10^{-3}$	6.8	17.2	17.8	26.2	31.4
0.8	$0.601 \pm 8.78 \times 10^{-3}$	6.6	20.2	24.8	40.8	35.6
0.7	$0.597 \pm 5.25 \times 10^{-3}$	12	25.6	27.4	49.6	43

Table 6: Accuracies and pruned channels/neurons of each layer of the depth-1 Neural distinguisher with no residual connection for Speck32/64 reduced to 7 rounds in the real-vs-random experiment for different APoZ values.

Since we saw the accuracy decreasing for an APoZ value greater or equal to 0.7, we look at the number of channels/neurons pruned above this cutoff across the first three distinguishers. Two experiments were run where the distinguishers' layers were pruned in two ways: one in which the smallest (for an APoZ value equal to 1) and one in which the largest (for an APoZ value equal to 0.8) number of channels/neurons per layer across all three distinguishers was pruned. The processes were called min-pruning and max-pruning, and the results can be seen in Tables 8 and 9.

While the performance was expected not to be impacted in the first case, the second one was done more of sheer curiosity to see how the performance would change. As expected, in the first case, the performance remained within one percentage point, but, in the second case, the performance surprisingly remained again within one percentage point. For all experiments conducted in this paper (unless otherwise specified), the results are the average of five trials, which was considered appropriate given the time some of the experiments took (see the repository[8] for details). However, the results might differ a bit if more trials per experiment would be conducted. Nevertheless, we keep the max-pruned network where we remove 7 channels from C1, 21 from C2, 25 from C3, 46 neurons from D1, and 36 from D2.

Finally, satisfied that we could even further prune the depth-1 network with no residual connection while the performance remained within one percentage point, we will move to the next section.

[8] `https://anonymous.4open.science/r/sourcecode`

APoZ	Accuracy	C1	C2	C3	D1	D2
1	$0.500 \pm 2.80 \times 10^{-4}$	0.8	0	0	2.2	15.4
0.9	$0.500 \pm 2.32 \times 10^{-4}$	2.2	0.2	1.2	2.4	20.2
0.8	$0.500 \pm 3.23 \times 10^{-4}$	2.2	1.2	2	11.8	24
0.7	$0.500 \pm 1.09 \times 10^{-4}$	5.6	8.4	7	17.2	32.6

Table 7: Accuracies and pruned channels/neurons of each layer of the depth-1 Neural distinguisher with no residual connection for Speck32/64 reduced to 8 rounds in the real-vs-random experiment for different APoZ values.

Distinguisher	Accuracy	TPR	TNR
N_5	$0.923 \pm 1.67 \times 10^{-3}$	$0.890 \pm 3.52 \times 10^{-3}$	$0.955 \pm 5.11 \times 10^{-4}$
N_6	$0.782 \pm 6.27 \times 10^{-4}$	$0.713 \pm 1.19 \times 10^{-3}$	$0.850 \pm 6.12 \times 10^{-4}$
N_7	$0.605 \pm 1.75 \times 10^{-3}$	$0.546 \pm 3.70 \times 10^{-3}$	$0.664 \pm 4.26 \times 10^{-3}$
N_8	$0.500 \pm 1.99 \times 10^{-4}$	$0.54 \pm 5.36 \times 10^{-1}$	$0.44 \pm 2.50 \times 10^{-1}$

Table 8: Accuracies of the min-pruned depth-1 Neural distinguishers with no residual connection for Speck32/64 reduced to 5, 6, 7, and 8 rounds in the real-vs-random experiment.

5 Visualizing the important features

In this section, we will look at whether a prior feature engineering will improve the performance of our distinguishers and whether all 64 input bits are needed for classification. A trained encoder will be used to preprocess the input, and regarding the assessment of the feature importance, LIME will be used. Finally, the experiments will be conducted on the max-pruned depth-1 network with no residual connection (also referred to as *pruned network*).

5.1 Feature engineering using an autoencoder

In this subsection, we will look at whether prior input engineering will improve the performance and, for this purpose, autoencoders of various compression capacities have been trained. An autoencoder is a neural network that learns to reproduce its input to its output, and it comprises of two parts: an encoder and a decoder. The encoder compresses the input to a latent representation, that is, an encoding that contains all the important information needed to represent the input, and the decoder takes this latent representation, trying to reconstruct the input [2]. The reason for choosing autoencoders to perform feature engineering was that autoencoders learn such a latent representation that ignores noise, anticipating that the network's performance would improve by bringing the useful features forward. The autoencoders corresponding to the results presented in Tables 10, 11, and 12 consist of one, two, and three blocks, respectively, each block being comprised of:

1. A 1D-CNN layer with kernel size 3, 32 channels, padding and stride of size 1, followed by a batch normalization and a ReLU activation layer.

Distinguisher	Accuracy	TPR	TNR
N_5	$0.915 \pm 1.41 \times 10^{-3}$	$0.875 \pm 1.54 \times 10^{-3}$	$0.955 \pm 1.50 \times 10^{-3}$
N_6	$0.770 \pm 4.24 \times 10^{-3}$	$0.691 \pm 8.84 \times 10^{-3}$	$0.848 \pm 1.37 \times 10^{-4}$
N_7	$0.596 \pm 7.70 \times 10^{-3}$	$0.543 \pm 7.40 \times 10^{-3}$	$0.648 \pm 1.63 \times 10^{-2}$
N_8	$0.500 \pm 1.65 \times 10^{-4}$	$0.54 \pm 2.83 \times 10^{-1}$	$0.460 \pm 2.83 \times 10^{-1}$

Table 9: Accuracies of the max-pruned depth-1 Neural distinguishers with no residual connection for Speck32/64 reduced to 5, 6, 7, and 8 rounds in the real-vs-random experiment.

2. A 1D-MaxPooling/1D-UpSampling layer with pool-size/size 2.

Distinguisher	Accuracy	TPR	TNR
N_5	$0.999 \pm 2.10 \times 10^{-6}$	$0.999 \pm 4.41 \times 10^{-5}$	$0.999 \pm 4.72 \times 10^{-6}$
N_6	$0.999 \pm 4.94 \times 10^{-6}$	$0.999 \pm 3.26 \times 10^{-5}$	$0.999 \pm 2.60 \times 10^{-5}$
N_7	$0.999 \pm 1.25 \times 10^{-5}$	$0.999 \pm 4.83 \times 10^{-5}$	$0.999 \pm 5.58 \times 10^{-5}$
N_8	$0.999 \pm 2.38 \times 10^{-5}$	$0.999 \pm 1.11 \times 10^{-4}$	$0.999 \pm 1.44 \times 10^{-4}$

Table 10: Accuracies of the one-block autoencoder for Speck32/64 reduced to 5, 6, 7, and 8 rounds in the real-vs-random experiment.

Distinguisher	Accuracy	TPR	TNR
N_5	$0.999 \pm 1.00 \times 10^{-6}$	$0.999 \pm 6.54 \times 10^{-7}$	$0.999 \pm 1.53 \times 10^{-6}$
N_6	$0.999 \pm 3.68 \times 10^{-7}$	$0.999 \pm 3.83 \times 10^{-7}$	$0.999 \pm 4.82 \times 10^{-6}$
N_7	$0.999 \pm 2.31 \times 10^{-6}$	$0.999 \pm 2.76 \times 10^{-6}$	$0.999 \pm 2.32 \times 10^{-6}$
N_8	$0.999 \pm 5.93 \times 10^{-6}$	$0.999 \pm 6.03 \times 10^{-6}$	$0.999 \pm 5.99 \times 10^{-6}$

Table 11: Accuracies of the two-block autoencoder for Speck32/64 reduced to 5, 6, 7, and 8 rounds in the real-vs-random experiment.

Distinguisher	Accuracy	TPR	TNR
N_5	$0.889 \pm 1.98 \times 10^{-2}$	$0.893 \pm 1.86 \times 10^{-2}$	$0.885 \pm 2.20 \times 10^{-2}$
N_6	$0.895 \pm 2.30 \times 10^{-2}$	$0.904 \pm 1.96 \times 10^{-2}$	$0.886 \pm 2.94 \times 10^{-2}$
N_7	$0.871 \pm 1.11 \times 10^{-2}$	$0.861 \pm 3.61 \times 10^{-2}$	$0.881 \pm 2.70 \times 10^{-2}$
N_8	$0.896 \pm 4.03 \times 10^{-2}$	$0.879 \pm 5.69 \times 10^{-2}$	$0.914 \pm 3.39 \times 10^{-2}$

Table 12: Accuracies of the three-block autoencoder for Speck32/64 reduced to 5, 6, 7, and 8 rounds in the real-vs-random experiment.

A prior reshaping and permutation of the input were also performed as in [7]. Concretely, starting from four 16-bit strings, the encoder of the one-block autoencoder compressed them into four 8-bit strings, the encoder of the two-block autoencoder compressed them into four 4-bit strings, and the encoder of the three-block autoencoder compressed them into four 2-bit strings. The results show that a convolutional encoder manages to learn a latent representation of the input that allows the convolutional decoder to reconstruct it almost perfectly

for the one and two-block cases and reasonably well reconstruct it for the three-block case. Having seen these results, the pretrained encoders were added as a preprocessing step to the pruned network, and training was again performed to see the effect. Performance close to the one we already saw (or even a slightly better one) was expected. However, some preliminary runs show quite the contrary. Even though the pretrained one and two-block encoders were used as a preprocessing step (as they were the most promising ones), the results do not show an improvement in the performance of the distinguishers. What is more, not even comparable results to the ones we already saw for the pruned network earlier are obtained.

For instance, for the pruned network with a one-block encoder as a preprocessor, the N_5 distinguisher's accuracy was 88%, and for the pruned network with a two-block encoder as a preprocessor, it was 82%. It seems that, even though the encoder managed to learn an efficient latent representation, once the input was transformed/engineered by the encoder, the pruned network did not have the complexity to decompose the engineered input and recombine it in a useful way. Having this intuition, the one-block encoder was added as a preprocessor to the original depth-10 network and, when looking at the results of the N_5 distinguisher, the accuracy indeed improved compared to the time when the pruned network was used; namely, it reached a 92% accuracy. As suspected, when adding an encoder to perform feature engineering, the network that does the classification needs indeed be complex enough to extract useful information from the latent representation.

5.2 Feature visualization with LIME

In this subsection, we will look closer at the input features and their importance to try to gain insights into the (pruned) distinguishers' behavior, which might aid in the improvement of future preprocessing methods. We will do this using one of the state of the art explanation techniques called Local Interpretable Model-agnostic Explanations (LIME) [15]. In short, according to [15], LIME explains the predictions of any classifier in an interpretable and faithful manner by learning an interpretable model locally around the prediction. Using it, the feature importances for all four distinguishers were computed, giving explanations for the five best predictions belonging to class 1 (fixed difference) and class 0 (random difference). In Appendix B, results are given just for the five best predictions of class 1 of the N_5 and N_6 distinguishers as the feature importance was similar. LIME with a submodular pick was also run [15]. However, even though it selected the instances judiciously, the results were similar to what was obtained so far, thus not contributing to a greater understanding of the distinguishers' behavior. From the figures given in Appendix B, we see that the importance of each feature is insignificant and that it varies from distinguisher to distinguisher (even from instance to instance). It seems that there is no clear local region that would have a considerable impact on the classification. Now, LIME already samples from both the vicinity of the instance and further away from it. Still,

it is possible that even larger regions need to be considered for the important features to become evident to the explainer.

Then, even though for Speck reduced to 5 rounds, the distinguisher performs quite well, the explainer suggests that removing any (even all of the 64 features) will insignificantly affect the classifier's performance. Having obtained those results even for a fairly good distinguisher and after seeing the type of explainer LIME currently uses, it might well be that the *linear* explanation model will not be able to explain the distinguishers' behavior as there might be no linear boundary to begin with. For now, as LIME could not identify the important features, the conclusion is left that all of the 64 inputs are important.

6 Conclusions and future work

In this paper, the distinguisher proposed by Gohr [7] was under a study to find a better performing or smaller distinguisher for Speck32/64. To this end, the Lottery Ticket Hypothesis has been evaluated for the first time for the distinguisher mentioned above, discovering that even the depth-1 version can be further pruned without significantly compromising the performance, empirically confirming the hypothesis anew. Then, based on the conclusions of prior experiments, the depth-1 network was successfully pruned to potentially aid in the process of explaining its behavior, besides having seen how pruning the suggested limit would affect the performance.

Next, it has been studied whether a prior feature engineering would result in a performance gain. In the process, convolutional autoencoders of various compression capacities that successfully reconstructed the inputs were for the first time discovered, using their trained encoders as a preprocessor prior to training the pruned depth-1 network. Results have shown that even though convolutional autoencoders manage to learn a latent representation that they can nearly perfectly decode, when passing the encoded inputs to the pruned depth-1 network, the network's performance decreased. This led to suspicion that the pruned network did not have the necessary complexity to extract useful information from the encoded inputs, which was later confirmed by additional experiments.

As a follow-up, intending to explain the distinguisher's behavior, the classification explainer LIME was for the first time deployed in this setting. Results showed that, despite the pruned depth-1 distinguisher performing reasonably well, LIME considered that none of the 64 inputs impacted the classification outcome. This suggests that a stronger explainer than the one LIME currently uses is needed, suspecting two possible causes (mentioned in Section 5) for LIME's current results that are yet to be studied.

One direction for future work would be to train the most recent networks used for image recognition to see whether a better performance can be achieved. Then, since there are still instances classified with high confidence as belonging to the opposite class, a second suggestion would be to look at ensemble learning to see whether it could alleviate the problem. Moreover, since the evaluation of the LTH revealed that the depth-1 neural distinguisher could be further pruned, it

would be interesting to consider evaluating it for different neural distinguishers. Finally, combining the SAT-based algorithm [10] with the framework presented in [16] for extending the differential attack to more rounds would be interesting as well.

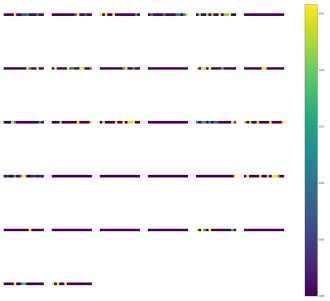
References

1. Abed, F., List, E., Lucks, S., Wenzel, J.: Differential cryptanalysis of round-reduced simon and speck. In: Cid, C., Rechberger, C. (eds.) Fast Software Encryption. pp. 525–545. Springer Berlin Heidelberg, Berlin, Heidelberg (2015)
2. Bank, D., Koenigstein, N., Giryes, R.: Autoencoders. CoRR **abs/2003.05991** (2020), `https://arxiv.org/abs/2003.05991`
3. 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
4. Benamira, A., Gerault, D., Peyrin, T., Tan, Q.Q.: A deeper look at machine learning-based cryptanalysis. In: Canteaut, A., Standaert, F.X. (eds.) Advances in Cryptology – EUROCRYPT 2021. pp. 805–835. Springer International Publishing, Cham (2021)
5. Biham, E., Shamir, A.: Differential cryptanalysis of des-like cryptosystems. vol. 4, pp. 2–21 (08 1990). https://doi.org/10.1007/3-540-38424-3_1
6. Frankle, J., Carbin, M.: The lottery ticket hypothesis: Training pruned neural networks. CoRR **abs/1803.03635** (2018), `http://arxiv.org/abs/1803.03635`
7. 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)
8. He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) Computer Vision – ECCV 2016. pp. 630–645. Springer International Publishing, Cham (2016)
9. Hou, Z., Ren, J., Chen, S.: Cryptanalysis of round-reduced simon32 based on deep learning. IACR Cryptol. ePrint Arch. **2021**, 362 (2021)
10. Hou, Z., Ren, J., Chen, S.: Improve neural distinguisher for cryptanalysis. Cryptology ePrint Archive, Report 2021/1017 (2021), `https://ia.cr/2021/1017`
11. Hu, H., Peng, R., Tai, Y., Tang, C.: Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. CoRR **abs/1607.03250** (2016), `http://arxiv.org/abs/1607.03250`
12. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T.Y.: Lightgbm: A highly efficient gradient boosting decision tree. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. p. 3149–3157. NIPS'17, Curran Associates Inc., Red Hook, NY, USA (2017)
13. Lai, X., Massey, J.L., Murphy, S.: Markov ciphers and differential cryptanalysis. In: Proceedings of the 10th Annual International Conference on Theory and Application of Cryptographic Techniques. p. 17–38. EUROCRYPT'91, Springer-Verlag, Berlin, Heidelberg (1991)
14. Remy, P.: Keract: A library for visualizing activations and gradients. `https://github.com/philipperemy/keract` (2020)
15. Ribeiro, M.T., Singh, S., Guestrin, C.: "why should i trust you?": Explaining the predictions of any classifier. In: Proceedings of the 22nd ACM

SIGKDD International Conference on Knowledge Discovery and Data Mining. p. 1135–1144. KDD '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2939672.2939778, https://doi.org/10.1145/2939672.2939778

16. Yadav, T., Kumar, M.: Differential-ml distinguisher: Machine learning based generic extension for differential cryptanalysis. Cryptology ePrint Archive, Report 2020/913 (2020), https://ia.cr/2020/913

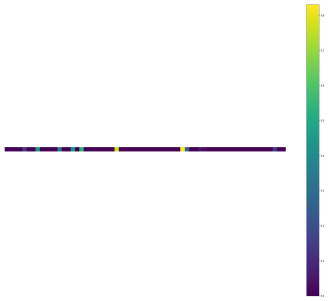
A Activation maps for the four depth-1 distinguishers



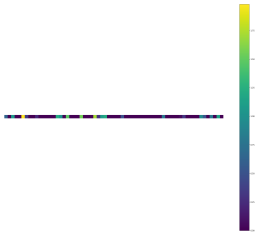
(a) The activation map A_1 of N_5.

(b) The activation map A_2 of N_5.

(c) The activation map A_3 of N_5.



(d) The activation map A_4 of N_5.



(e) The activation map A_5 of N_5.

Fig. 3: The activation maps of the N_5 distinguisher.

B Feature Visualization

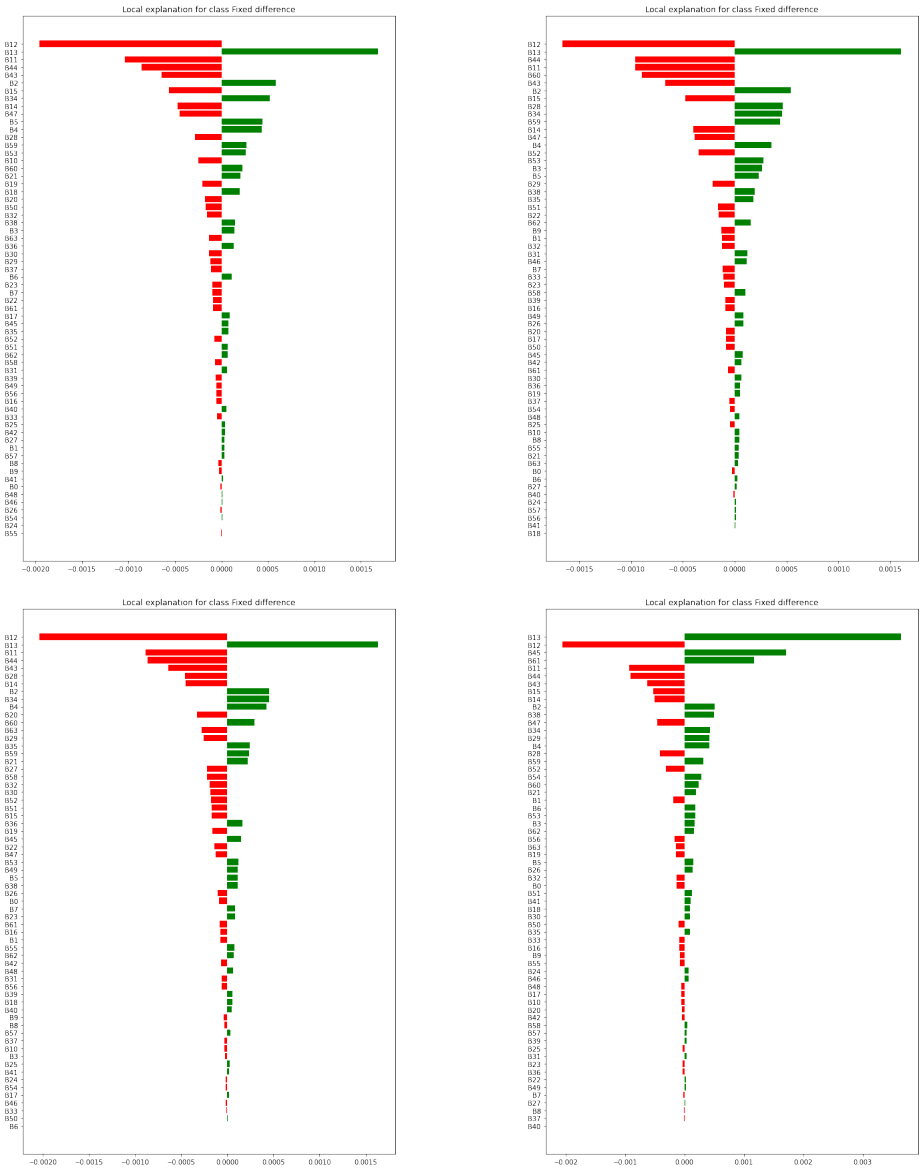


Fig. 4: The important features and their contribution to the classification for the N_5 distinguisher.

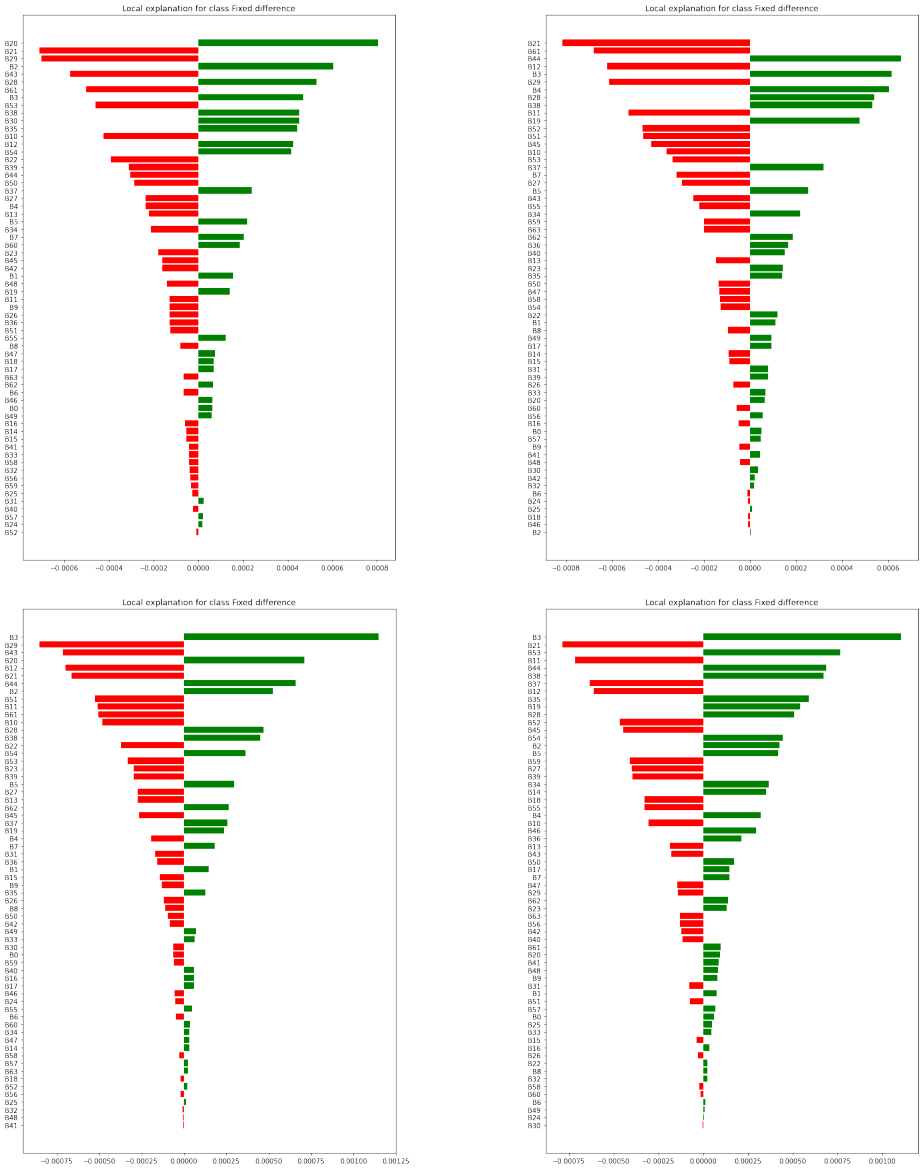


Fig. 5: The important features and their contribution to the classification for the N_6 distinguisher.