ObfuNAS: A Neural Architecture Search-based DNN Obfuscation Approach

Tong Zhou
Northeastern University
Boston, MA, USA
zhou.tong1@northeastern.edu

Shaolei Ren
UC Riverside
Riverside, CA, USA
sren@ece.ucr.edu

Xiaolin Xu
Northeastern University
Boston, MA, USA
x.xu@northeastern.edu

ABSTRACT

Malicious architecture extraction has been emerging as a crucial concern for deep neural network (DNN) security. As a defense, architecture obfuscation is proposed to remap the victim DNN to a different architecture. Nonetheless, we observe that, with only extracting an obfuscated DNN architecture, the adversary can still retrain a substitute model with high performance (e.g., accuracy), rendering the obfuscation techniques ineffective. To mitigate this under-explored vulnerability, we propose ObfuNAS, which converts the DNN architecture obfuscation into a neural architecture search (NAS) problem. Using a combination of function-preserving obfuscation strategies, ObfuNAS ensures that the obfuscated DNN architecture can only achieve lower accuracy than the victim. We validate the performance of ObfuNAS with open-source architecture datasets like NAS-Bench-101 and NAS-Bench-301. The experimental results demonstrate that ObfuNAS can successfully find the optimal mask for a victim model within a given FLOPs constraint, leading up to 2.6% inference accuracy degradation for attackers with only 0.14× FLOPs overhead. The code is available at: https://github.com/Tongzhou0101/ObfuNAS.

KEYWORDS

Deep neural network, Security, Side channels, Architecture obfuscation

ACM Reference Format:
Tong Zhou, Shaolei Ren, and Xiaolin Xu. 2022. ObfuNAS: A Neural Architecture Search-based DNN Obfuscation Approach. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '22), October 30-November 3, 2022, San Diego, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3508352.3549429

1 INTRODUCTION

The architecture of a deep neural network (DNN) plays an essential role in its performance, such as inference accuracy and latency. As a result, searching for the optimal DNN architecture has become a critical step, which is extremely costly due to the exponentially large architecture space, e.g., 10^{18} candidates in DARTS [12]. Therefore, high-performance neural architectures are valuable assets for model developers, and becoming the prime targets of adversarial attacks. For a concrete example, an attacker can extract the architecture of a DNN model and then train a competitive substitute model with high performance for commercial interest [31]. Importantly, side-channel-based DNN architecture extraction, among the existing attacks, has been successfully demonstrated in various hardware platforms like GPU [32], FPGA [31], CPU [29], and embedded processors [1].

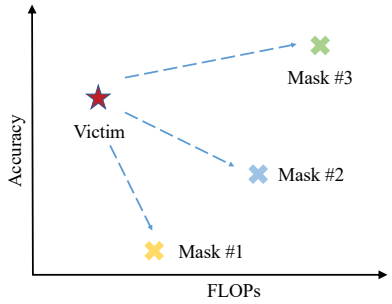


Figure 1: An illustration of different obfuscation schemes for the victim model.

Since it is difficult to fully eliminate the association between a DNN architecture and its side channels on hardware devices, a solution to mitigate the side-channel-based DNN architecture extraction attacks is to obfuscate the DNN architecture, such as the topology, layer types, and layer dimensions [9, 14]. For example, NeurObfuscator [9] proposed by Li *et al.* employs eight obfuscation strategies to hide the original DNN model architecture, i.e., to make it more different. Although these strategies can prevent accurate DNN architecture extraction by introducing prediction errors in architectural parameters like the number of layers and dimensions, they all neglect that the architecture difference should not be the only key metric to measure the obfuscation effects. In fact, a mask model with a large architectural difference from the victim model can still have high, or even higher, inference accuracy and hence be of great value to an adversary.

We illustrate the drawbacks of the architecture-difference oriented obfuscation schemes in Fig. 1. Assuming similar obfuscation strategies are applied to the victim model with different latency budgets (measured by floating-point operations, FLOPs), Mask #3 will be selected as the optimal mask, since it allows more obfuscation space for architecture difference. If so, the victim model will be mapped accordingly to preserve its original inference accuracy, while the adversary will be capable to train the mask to reach higher inference accuracy. However, as shown in Fig. 1, the

ICCAD '22, October 30-November 3, 2022, San Diego, CA, USA

ACM ISBN 978-1-4503-9217-4/22/10…$15.00
https://doi.org/10.1145/3508352.3549429

overall optimal mask should be Mask #1 if the FLOPs-accuracy trade-off is considered. To further support this point, we present an example in Fig. 2. We select a victim model in NAS-Bench-101 [30] with 77.2% inference accuracy on CIFAR-10 [8]. If adopting the architecture-difference objective, we can get a mask with a different cell structure shown in Fig. 2 (4). However, the mask can achieve 93.02% inference accuracy, which allows attackers to get a model with even much higher accuracy than the victim, making the obfuscation ineffective at all.

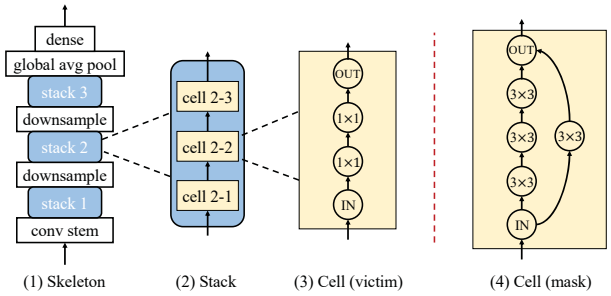


Figure 2: An example in NAS-Bench-101 [30].

In this work, we make the first attempt to jointly use accuracy and FLOPs as the combined obfuscation evaluation metric, for which we need to solve the following two challenges. First, like the victim architecture search space, the mask architecture search space for obfuscation is also large. Thus, manually designing mask architectures is simply out of the question. Second, apart from finding a mask architecture with low inference accuracy for obfuscation, we also need to ensure that the adopted mask architecture does not have too large FLOPs, otherwise the victim's inference latency and energy consumption would also increase significantly. To overcome these challenges, we propose ObfuNAS, a neural architecture search (NAS) based DNN obfuscation approach, which aims to maximize the accuracy degradation of masks subject to FLOPs constraints. More specifically, by converting the mask architecture design into a novel FLOPs-constrained neural architecture search problem, we can leverage a well-trained super-net along with an accuracy predictor to efficiently find a mask architecture, which achieves effective obfuscation by leading the adversary to lower accuracy while meeting the FLOPs constraints.

The contributions of this work are summarized as follows:

(1) To the best of our knowledge, this is the first work using NAS to protect DNN against architecture extraction attacks. Leveraging the combined accuracy-FLOPs metric to guide DNN architecture masking, ObfuNAS achieves more effective obfuscation than the state of the art.
(2) We propose 7 obfuscation strategies of 3 types, which preserve the inference accuracy of the victim model. By increasing the mask architecture's training difficulties, these strategies can effectively prevent attackers from training a model with equivalent or even better performance.
(3) Unlike previous obfuscation methods involving low-level modification, our proposed framework achieves DNN architecture obfuscation by only making algorithm-level changes to the victim model, which provides general applicability for any execution environment.

2 BACKGROUND AND RELATED WORKS

2.1 DNN Architecture Extraction

The performance of DNNs is largely determined by their architectures, like VGG [18], ResNet [3], and inception network [21]. Therefore, a well-designed model architecture can be considered as intellectual property with great commercial values, which motivates the architecture extraction attack. For example, it is demonstrated that an adversary is able to extract the DNN architecture in Machine-Learning-as-a-Service (MLaaS) platforms using the cache side channel [29]. Similarly, other side channels can also be used for such architecture extraction. In [6], Hua *et al.* successfully inferred the underlying network architectures using the memory and timing side channels during the DNN execution. Besides, Batina. *et al.* [1] utilized electromagnetic (EM) side channel to reverse-engineer the important parameters of the architecture, e.g., the number of layers and layer dimensions, to infer the victim DNN. Upon extracting the DNN architecture, attackers can further train a substitute model with competitive inference accuracy.

2.2 DNN Architecture Protection

Targeting these side-channel-based DNN architecture attacks, previous works have explored countermeasures from hardware platform design[25] to DNN execution [9]. For example, Liu *et al.* proposed a method to defend architecture extraction utilizing memory access patterns, which involves oblivious shuffle, address space layout randomization, and dummy memory accesses [13]. Luo *et al.* proposed a framework to increase the difficulty of extracting DNN architectures from EM side-channel leakage through scheduling the tensor program execution [14]. Besides, NeurObfuscator is proposed to prevent exact architecture extraction [9] through obfuscating the original dimension and number of the victim DNN layers. However, these methods require low-level modification and are limited by the execution environment. More importantly, existing works failed to take into account the inference accuracy of the mask architecture — by extracting the obfuscated mask architecture, the attacker can still obtain high, or even higher, inference accuracy than the victim.

2.3 Neural Architecture Search (NAS)

Designing a high-performance DNN requires not only substantial time and resources, but also domain knowledge and expertise. To ease these requirements, NAS has been recently developed to search for Pareto-optimal DNNs, i.e., those with the highest accuracy given a FLOPs (or inference latency/energy) constraint [10, 22, 28]. Such DNNs are the focus of the architecture search and most worthy of protection. There are three key components of NAS: search space, search strategy, and architecture evaluation [4]. Once a new architecture is selected from the search space based on the search strategy, its performance would be evaluated to guide the NAS process, which is time-consuming and resource-intensive. Therefore, many techniques have been proposed to accelerate the process of model evaluation, such as weight sharing [2, 24]. Additionally, training an accuracy/latency performance predictor to filter out

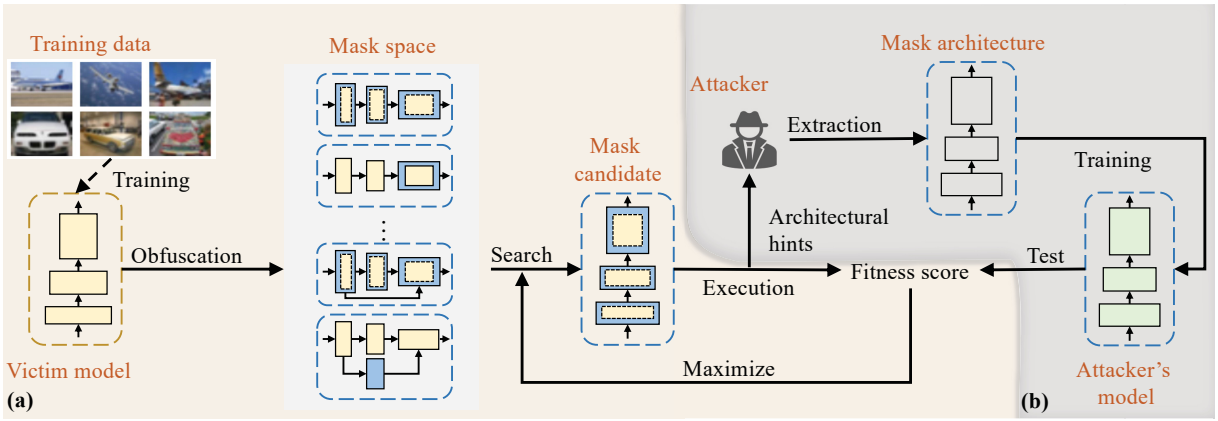


Figure 3: The overview of ObfuNAS. (a) Protection workflow: apply obfuscation strategies to the victim model and search the optimal mask causing the maximal accuracy degradation in mask space, and then the mask will be executed during inference. (b) Attack workflow: an attacker will extract the model architecture by leveraging the architectural hints during victim execution, then train the extracted architecture with similar training data as used in the victim model. The fitness score will be calculated later to guide the mask search.

those unlikely optimal architectures is also commonly used, where only the top performance architectures are selected for actual evaluation [23, 26, 28]. Last but not least, several benchmarks have been released for quick and fair comparison for NAS algorithms, which allows obtaining the network performance by querying the pre-computed dataset [30] or a surrogate model [17].

2.4 Threat Model

To explore a generic defense method, we adopt a strong threat model in this work. Specifically, we assume an attacker can perfectly extract the architecture, but not the weights, of the executed DNN (e.g., victims or masks), including the topology and activation functions, through architectural hints like side-channel leakage [1, 5, 27, 32]. Moreover, we assume a strong attacker whose DNN model training ability is as strong as the victim model developer, i.e., given an architecture π, if the developer can train it and achieve inference accuracy ACC_π, the attacker can achieve ACC_π as well with training π on the similar training data as the developer.

3 PROPOSED APPROACH: ObfuNAS

The overview of ObfuNAS is shown in Fig. 3, which is proposed to solve two key problems: 1) how to obfuscate the DNN architecture with the original model accuracy preserved; 2) how to achieve the best obfuscation performance, i.e., the maximal accuracy drop for attackers while satisfying a FLOPs constraint.

For the first problem, we propose 7 obfuscation strategies of 3 categories, namely, scaling-up (Sec. 3.1), operation-change (Sec. 3.2), and connection-adding (Sec. 3.3). The principle behind these strategies is to increase the training difficulties of the mask architecture, leading attackers to lower inference accuracy after extracting and training the mask. Besides, we will prove that these strategies are function-preserving to make sure that the victim model can preserve its original inference accuracy after obfuscation, as described in Eq. (1):

$$\forall x : f(x|\theta_f) = g(x|\theta_g), \tag{1}$$

where x denotes the input of the network, f represents the victim network, g represents the obfuscated network, and θ is the corresponding network parameters. With proposed strategies, we can generate a mask space for the victim model.

As for the second problem, we will simulate the attacking process and use its result to guide the optimal mask search (see Sec. 3.4). Considering obfuscation is at the cost of the FLOPs budget, we propose resource-constrained search, i.e., search for the optimal mask within a given FLOPs constraint, which is implemented by an evolutionary search with accuracy drop as the fitness score.

3.1 Scaling-up Obfuscation

It is well-known that the optimization/training difficulties will grow with the increase of DNN dimensions [19]. Based on this observation, we propose 3 obfuscation strategies by scaling up the victim DNN architecture to increase training difficulties for the attackers from width, depth, and kernel size, respectively.

Layer Widening. Layer widening is proposed to increase the output dimension of a layer. Since it is commonly applied to the convolutional (Conv) layer, here we will use Conv to illustrate this strategy. Suppose the weights of a Conv layer L is $W^{(L)} \in \mathbb{R}^{k_1,k_2,i,o}$, where $k_1 \times k_2$ is the kernel size (we assume the kernel sizes of all Conv layers are the same for simplicity), i denotes the number of input channels, and o stands for the number of output channels. After layer widening, the weights of L will be $V^{(L)} \in \mathbb{R}^{k_1,k_2,i,o'}$, with $o' > o$, and the weights of the subsequent Conv layer $L+1$ would change from $W^{(L+1)} \in \mathbb{R}^{k_1,k_2,o,q}$ to $V^{(L+1)} \in \mathbb{R}^{k_1,k_2,o',q}$ to match the increased output channels of L.

To preserve the function of the original layers, we need to adjust the value of $V^{(L)}$ and $V^{(L+1)}$ to satisfy the following equations:

$$V^{(L)}_{\cdot,\cdot,\cdot,j} = \begin{cases} W^{(L)}_{\cdot,\cdot,\cdot,j}, & j \le o \\ 0, & o < j \le o' \end{cases}, \tag{2}$$

$$V^{(L+1)}_{\cdot,\cdot,t,\cdot} = \begin{cases} W^{(L+1)}_{\cdot,\cdot,t,\cdot}, & t \le o \\ random, & o < t \le o' \end{cases}. \tag{3}$$

Note that this strategy can also work for the fully connected layer, which can be replaced with a Conv layer with kernel size 1×1.

Layer Deepening. We use layer deepening to increase the depth of DNNs by sequentially inserting additional layers. To be function-preserving, the inserted layer should function as an identity layer, i.e., the input of this layer is equal to its output. If the inserted layer is a fully connected layer, we can simply set its weights to an identity matrix. Otherwise, suppose we insert a Conv layer with weights $U \in \mathbb{R}^{k_1,k_2,o,o}$ between two sequential Conv layers $W^{(L)} \in \mathbb{R}^{k_1,k_2,i,o}$ and $W^{(L+1)} \in \mathbb{R}^{k_1,k_2,o,p}$, then U should be set to:

$$U_{p,q,t,j} = \begin{cases} 1, & p = \frac{k_1+1}{2} \wedge q = \frac{k_2+1}{2} \wedge t = j \\ 0, & otherwise \end{cases}. \tag{4}$$

Besides, if the inserted layer is followed by an activation function $\phi(\cdot)$, it should satisfy the restriction $\phi(\cdot) = \phi(\phi(\cdot))$, e.g., the commonly used activation function ReLU. Moreover, extra efforts are required if batch normalization is used. Specifically, batch normalization will do the following transformation during inference [7]:

$$y = \frac{\gamma}{\sqrt{Var[x]+\epsilon}} \cdot x + (\beta - \frac{\gamma E(x)}{\sqrt{Var[x]+\epsilon}}), \tag{5}$$

where ϵ is a small self-define value, $E(x)$ and $Var[x]$ are the mean and variance of input data, which are fixed during inference. Therefore, by setting $\gamma = \sqrt{Var[x]+\epsilon}$ and $\beta = E(x)$, we can undo the normalization and successfully build an identity layer.

Kernel Widening. A Conv layer with weights $W \in \mathbb{R}^{k_1,k_2,i,o}$ after kernel widening would become $U \in \mathbb{R}^{k_3,k_4,i,o}$, where $k_3 > k_1$ and $k_4 > k_2$. The function preservation is straightforward, i.e.,

$$U_{p,q,\cdot,\cdot} = \begin{cases} W_{p,q,\cdot,\cdot}, & \frac{k_3-k_1}{2} \le p \le \frac{k_3+k_1}{2} \wedge \frac{k_4-k_2}{2} \le q \le \frac{k_4+k_2}{2} \\ 0, & otherwise \end{cases}. \tag{6}$$

Besides, zero-padding can be applied to the input feature maps in order to preserve the original size of output feature maps after convolution.

3.2 Operation-change Obfuscation

Since Conv layers are functional, we aim to replace some non-parameter layers with it to increase the number of trainable parameters, leading to an increase in training difficulties of the obfuscated architecture. Such replacement is described as the operation-changing obfuscation strategy in this work. The principle of the non-parameter layer selection is that the layer can be represented by a Conv layer with function preserved. Following this principle, we select two common DNN operations, average pooling and skip connection.

Average Pooling Replacement. Suppose a Conv layer uses the same kernel size ($k_1 \times k_2$), stride, and padding pattern as the average pooling layer, then it can work as average pooling when setting every value of its weights to $\frac{1}{k_1 \times k_2}$.

Skip Connection Replacement. Since skip connection is functionally equal to an identity layer, this replacement can be transferred to using a Conv layer to perform identity operation, as shown in Fig. 4, where Int. refers to the intermediate layer(s), and +/& indicates sum/concatenate operation. Thus, the weights adjustment of this Conv layer is the same as Eq. (4).

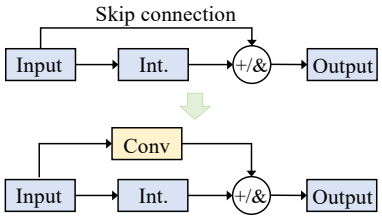


Figure 4: Skip connection replacement.

3.3 Connection-adding Obfuscation

In this category, the original connection will be preserved to maintain the inference accuracy. Moreover, we add extra connections to disturb the signal propagation, which will also increase the optimization difficulties for the attacker while training the extracted mask.

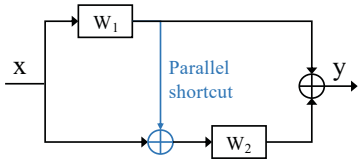


Figure 5: Parallel shortcut adding.

Shortcut Adding. This strategy is used to add a shortcut between two non-directly connected layers if their dimensions match. The added shortcut can be further divided into the sequential shortcut and the parallel shortcut, depending on the hierarchy of these two layers. Specifically, if these two layers are sequential, the added shortcut will be the same as the skip connection in Fig. 4.

As for the parallel shortcut, one example is shown in Fig. 5, where x is the input, y is the output, W_1 and W_2 are the weights of two Conv layers. Before adding the parallel shortcut, we have

$$\begin{aligned} y &= W_1^T * x + W_2^T * x, \\ \frac{\partial y}{\partial W_1} &= \frac{\partial y}{\partial W_2} = x, \end{aligned} \tag{7}$$

while after adding the parallel shortcut, Eq. (7) will become

$$\begin{aligned} y &= W_1^T * x + W_2^T (W_1^T * x + x), \\ \frac{\partial y}{\partial W_1} &= x + W_2^T * x, \quad \frac{\partial y}{\partial W_2} = x + W_1^T * x, \end{aligned} \tag{8}$$

which indicates that the parallel shortcut adding will cause the gradient update of W_1 and W_2 influenced by each other. As a result, the optimization difficulties of architectures with such connections would increase. To preserve the functionality, the shortcut feature maps should be multiplied by 0 before summation.

Layer Branch Adding. The connection of the layer branch adding is similar to the shortcut adding strategy, except that the

shortcut will be replaced by an operation to increase the model complexity. Besides, the chosen operation should also preserve the size of feature maps, so the feature maps can be later added together. For this strategy, the function preservation can be achieved by setting the weights of the operation to 0.

3.4 Optimal Mask Search

In this section, we propose resource-constrained search and apply the evolutionary algorithm to efficiently search for the optimal mask.

Resource-constrained Search. Our objective is to search for a mask that yields the maximal accuracy drop with FLOPs constrained, which can be formulated as follows:

$$\begin{aligned} \pi^* = \arg\max_{\pi_i \in \Omega} \mathcal{L}(W_{\pi_i}; X_{val}), \\ s.t.\ FLOPs(\pi_i) < \tau, \end{aligned} \tag{9}$$

where W_{π_i} is the network parameters associated with the mask π_i, Ω denotes the mask space, X_{val} denotes the validation dataset, $\mathcal{L}(\cdot)$ stands for the loss function, π^* is the optimal mask we expect to find out, and τ is the given FLOPs constraint.

Evolutionary Algorithm. In this work, we adopt an evolutionary algorithm that has been demonstrated to be effective for NAS problems [11, 15, 20]. Specifically, our fitness function F is defined as:

$$F(\pi_i) = -ACC_{val}(\pi_i), \tag{10}$$

where $ACC_{val}(\cdot)$ is the validation accuracy of a mask. The searching process aims to find the mask with the highest fitness score within a certain FLOPs constraint, which is consistent with Eq. (9).

4 EXPERIMENTAL VALIDATION

We evaluate the performance of ObfuNAS based on three architecture spaces used in AlphaNet [23], NAS-Bench-101 [30], and NAS-Bench-301 [17] (Sec 4.1). Specifically, in each space, we select several Pareto-optimal architectures as the victim models and adopt applicable obfuscation strategies (Sec. 4.2). We then search for the best mask with the maximal accuracy degradation for each victim model and compare the results with the state-of-the-art (SOTA) obfuscation framework, i.e., NeurObfuscator (Sec. 4.3). Since our approach focuses on algorithm-level obfuscation, we exclude two obfuscation knobs in NeurObfuscator, i.e., optimization knobs and scheduling knobs, for fair comparisons.

4.1 Architecture Search Space

4.1.1 ***AlphaNet.*** AlphaNet aims to search for sub-nets from a super-net that can achieve Pareto-optimal performance on ImageNet. The search space of the super-net is defined in Table 1, where MBConv denotes the inverted residual block used in [16], the expansion ratio is the parameter of the Conv layer inside MBConv, and MBPool is the last Conv layer with average pooling. Moreover, since AlphaNet provides a well-trained super-net, we can build an architecture dataset consisting of its sub-nets and evaluate them to obtain their inference accuracy.

4.1.2 ***NAS-Bench-101 (NB-101).*** NB-101 provides a public architecture dataset for NAS, including 423k unique DNN architectures and 5M models trained and evaluated on CIFAR-10. All unique architectures are made up of the same number of cells but with different cell structures, which is represented by a directed acyclic graph (DAG) with up to 7 nodes and 9 edges. In a DAG, each node and edge indicate an operation and a feature tensor, respectively. Apart from the input and output nodes, others have 3 possibilities: 3×3 convolution, 1×1 convolution, and 3×3 max pooling.

4.1.3 ***NAS-Bench-301 (NB-301).*** NB-301 is a surrogate NAS benchmark that includes about 10^{18} architectures covered by the DARTS search space [12]. The DNN backbone of all architectures is built with 8 cells categorized into two types, namely normal cell and reduction cell. Each cell can be represented as a DAG with 12 edges and 7 nodes (2 input nodes, 4 intermediate nodes, and 1 output node). Here, each node and edge indicate a feature tensor and an operation, which is different from the definition in NB-101. Each intermediate node is the addition result of two operations, and all results of intermediate nodes will be concatenated as the final output. There are 7 operations involved: 3×3/5×5 separable/dilated convolution, 3×3 average/max pooling, and identity. Besides, NB-301 includes a trained surrogate model to predict the accuracy performance on CIFAR-10 for each architecture in the search space.

4.2 Experimental Setup

Since AlphaNet has reported 8 Pareto-optimal architectures with different FLOPs [23], i.e., A0-A6 (A5 has two versions) shown in Table 3, we directly use them as the victim models for further obfuscation. However, for NB-101 and NB-301 that do not provide any Pareto-optimal architectures, we select a few architectures with different accuracy-FLOPs trade-offs on the Pareto front as the victim models, with selected architectures shown in Fig. 6 and Fig. 7, respectively.

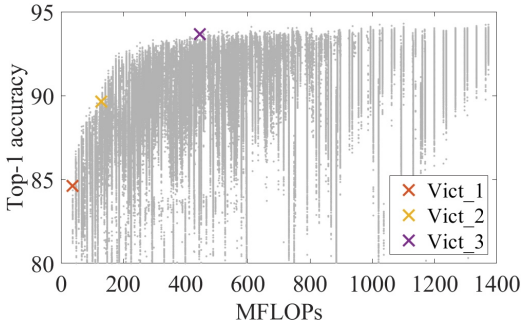


Figure 6: Victims in NB-101.

Due to the difference in the search space of these three datasets, the applicable obfuscation strategies also vary, see Table 2, where the checkmark denotes all strategies in that category are applicable. Specifically, obfuscation for AlphaNet is straightforward, i.e., scaling up the dimensions from different perspectives within the defined search space. Specially, the activation function used in AlphaNet is swish [23], defined as

$$swish(x) = x * \frac{1}{1 + e^{-x}}, \tag{11}$$

where x is the input feature map. To satisfy the restriction $\phi(\cdot) = \phi(\phi(\cdot))$ in layer deepening, we construct a fake swish function

Table 1: The search space of AlphaNet.

Block	Width	Depth	Kernel size	Expansion ratio
Conv	{16,24}	-	3	-
MBConv-1	{16,24}	{1,2}	{3,5}	1
MBConv-2	{24,32}	{3,4,5}	{3,5}	{4,5,6}
MBConv-3	{32,40}	{3,4,5,6}	{3,5}	{4,5,6}
MBConv-4	{64,72}	{3,4,5,6}	{3,5}	{4,5,6}
MBConv-5	{112,128}	{3,4,5,6,7,8}	{3,5}	{4,5,6}
MBConv-6	{192,200,208,216}	{3,4,5,6,7,8}	{3,5}	{4,5,6}
MBConv-7	{216,224}	{1,2}	{3,5}	6
MBPool	{1792,1984}	-	1	6

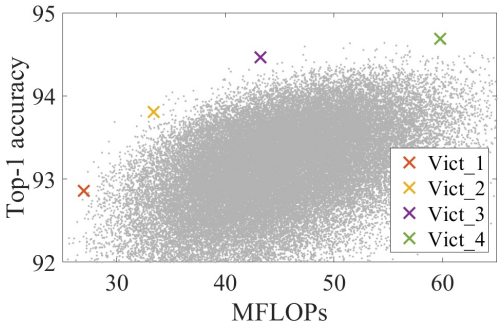


Figure 7: Victims in NB-301.

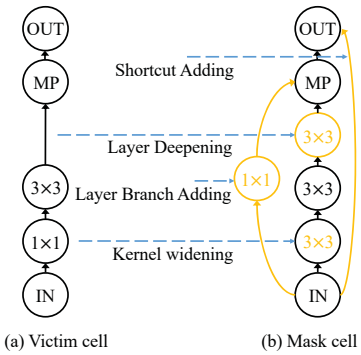


Figure 8: An Illustration of obfuscation strategies for cells in NB-101, where IN and OUT denote input and output feature maps, 1×1 and 3×3 are the kernel sizes of Conv layers, and MP denotes 3×3 max pooling. The feature map size inside the cell is fixed by adding zero paddings.

with the same operators (e.g., addition and division) to replace the original swish function for added layers, i.e.,

$$t = 1 + e^{-x}, \quad fake\_swish(x) = x * \frac{t}{t}. \tag{12}$$

Following the conclusions in [1], an attacker can deduce the activation function by comparing its timing side-channel leakage (i.e., time duration) with other known activation functions, the proposed self-defined fake swish will still be identified as swish due to the similar operations.

Table 2: Obfuscation strategies for each architecture dataset. OP-change denotes operation-change and CN-adding denotes connection-adding obfuscation.

	Scaling-up	OP-change	CN-adding
AlphaNet	✓	-	-
NB-101	Depth, Kernel	-	✓
NB-301	Kernel	✓	-

For NB-101, other than scaling up the depth and kernel size by layer-deepening and kernel-widening strategies, the connection-adding strategies are also applicable, with an illustration shown in Fig. 8. As for NB-301, the victim cells can be obfuscated with the kernel widening and the operation-change strategies.

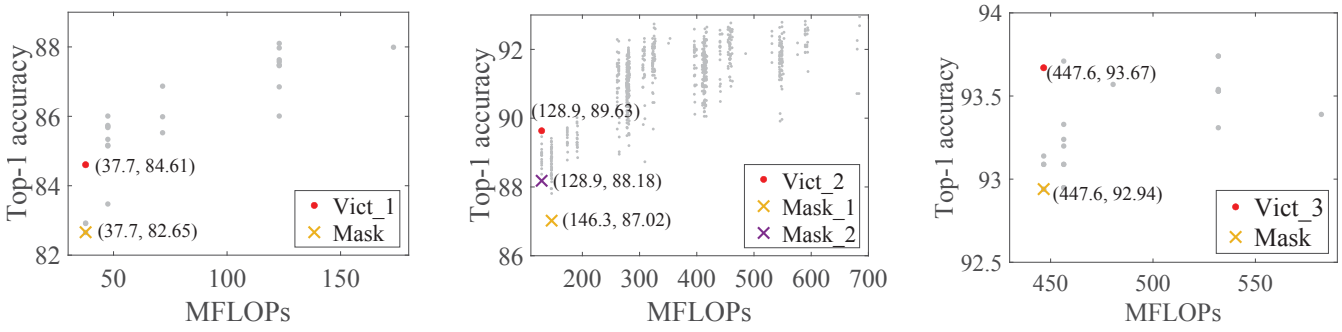
4.3 Results

To explore the best obfuscation performance of ObfuNAS, we first relax the constraint to find the best masks for all victim models. As shown in Table 3, the obfuscation performance of AlphaNet is not as good as expected, i.e., the accuracy of the mask is similar to the victim itself. One explanation is that the increased optimization difficulties raised by network scaling-up are limited, especially for DNNs like AlphaNet including the residual connection, which can ease the training difficulties for larger DNNs. However, our method still outperforms the SOTA (NeurObfuscator [9]), which results in higher accuracy growth with larger FLOPs overhead.

For victims in NB-101, ObfuNAS achieves around 1% ~ 3% accuracy degradation with a small FLOPs overhead ($< 0.14\times$), thanks to the shortcut adding strategy introducing no latency overhead. In contrast, NeurObfuscator even boosts the inference accuracy of masks up to 2.5%. Besides, ObfuNAS protects the victims in NB-301 by deteriorating ~1% accuracy, while NeurObfuscator can only achieve at most 0.05% accuracy drop. Since the obfuscation strategies for NB-301 involve operation-change, which replaces non-parameter operations with Conv, the FLOPs overhead, in this case, has obvious growth. Overall, ObfuNAS achieves better obfuscation performance than the SOTA obfuscation framework. Note

Table 3: The obfuscation results without latency constraints.

	Victim	Original		ObfuNAS		NeurObfuscator [9]	
		Acc	MFLOPs	Acc	MFLOPs	Acc	MFLOPs
AlphaNet (ImageNet)	A0	77.78	203.39	**77.41**	**266.50**	77.96	342.43
	A1	78.90	279.24	78.91	388.01	79.38	498.97
	A2	79.09	316.73	79.25	414.36	79.72	479.91
	A3	79.46	356.52	**79.43**	**433.49**	79.90	553.56
	A4	80.01	443.53	**80.00**	**581.05**	80.41	651.96
	A5	80.27	491.48	**80.14**	**606.61**	80.68	743.34
	A5_1	80.74	594.79	**80.66**	**680.74**	81.03	837.46
	A6	80.82	709.01	80.85	795.06	81.23	983.28
NB-101 (CIFAR-10)	Vict_1	84.61	37.70	**82.65**	**37.70**	87.11	202.51
	Vict_2	89.63	128.85	**87.02**	**146.29**	91.73	548.96
	Vict_3	93.67	446.62	**92.94**	**446.62**	93.53	531.89
NB-301 (CIFAR-10)	Vict_1	92.86	26.95	**91.74**	**40.46**	92.87	27.86
	Vict_2	93.81	33.40	**92.46**	**49.23**	93.76	35.43
	Vict_3	94.46	43.24	**93.14**	**56.64**	94.41	44.16
	Vict_4	94.69	59.76	**93.68**	**71.84**	95.65	63.43



(a) Masks of Vict_1. **(b) Masks of Vict_2.** **(c) Masks of Vict_3.**

Figure 9: The optimal mask with different FLOPs overhead for victims in NB-101.

that even 1% accuracy matters a lot in NAS, thus the accuracy degradation achieved by ObfuNAS will make the architecture extraction attacks meaningless.

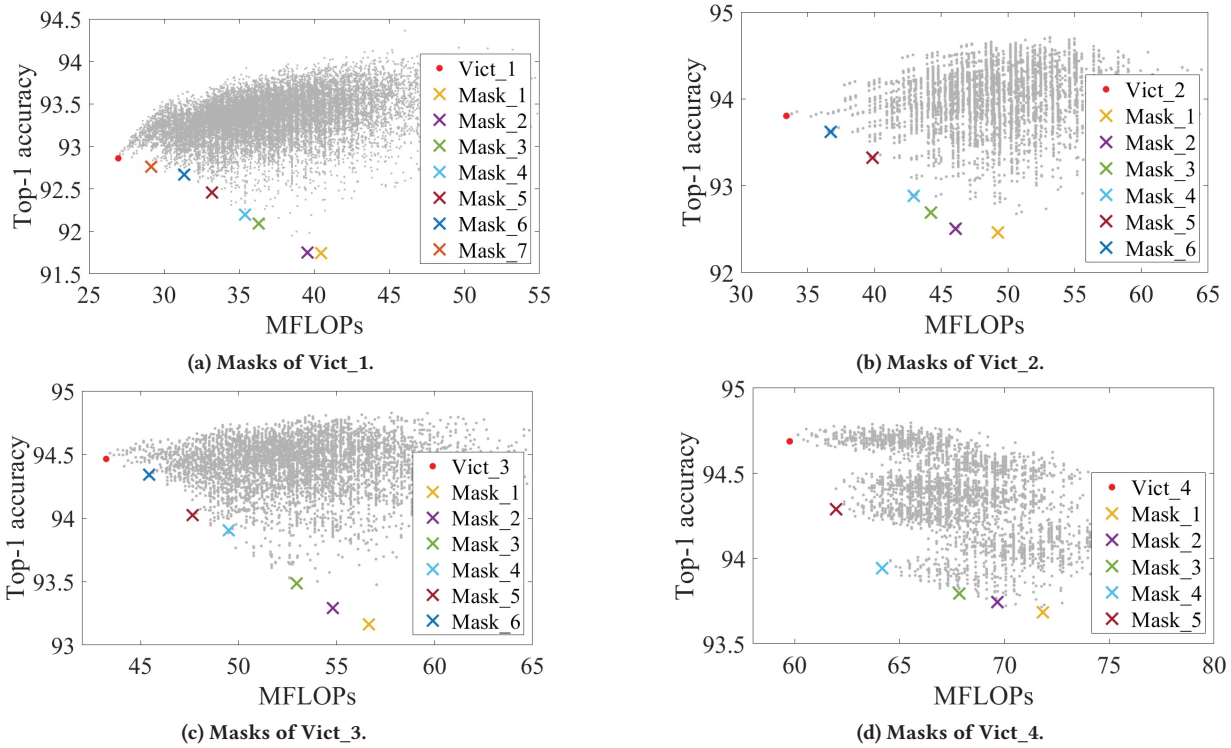
Next, we search for the optimal masks for victims in NB-101 and NB-301 with different FLOPs constraints. The results of NB-101 are shown in Fig. 9, where the optimal masks for Vict_1 and Vict_3 are unique regardless of the FLOPs constraints. The reason is that these two masks only adopt the shortcut adding strategy, which is effective for architecture protection but introduces no FLOPs overhead. For Vict_2, the results show that a mask with lower accuracy can be found if FLOPs overhead increases. For each victim in NB-301, the optimal mask would be different, as shown in Fig. 10, depending on the given FLOPs budget.

5 DISCUSSION

Although the obfuscation performance of scaling-up is not promising for victims in AlphaNet (Table 3), we note that it likely results from the sophisticated architecture of the super-net of AlphaNet, e.g., including ResNet-like structures, and the jointly optimized weights for both smaller and larger DNNs. To further evaluate the performance of scaling-up, as well as the other two obfuscation strategies, we include the ablation studies on NB-101 and NB-301, for they involve all proposed strategies.

As shown in Table 4, only adopting connection-adding achieves a higher accuracy drop with lower FLOPs overhead than the scaling-up, although the latter still show up to 1.4% accuracy drop. For victims in NB-301, the results indicate that the operation-change strategies make main contributions to the best obfuscation performance (Table 3) compared to the scaling-up strategies. However, both cases demonstrate that by combining the scaling-up strategies with others, the best obfuscation performance can still get improved.

Overall, each strategy has different obfuscation performance, which is also influenced by the victim architecture and its search space. All of them will bring extra FLOPs overhead except the shortcut-adding strategy. For the discussed victim models, the



(a) Masks of Vict_1.

(b) Masks of Vict_2.

(c) Masks of Vict_3.

(d) Masks of Vict_4.

Figure 10: The optimal mask with different FLOPs overhead for victims in NB-301.

Table 4: Ablation studies of different obfuscation strategies.

	Victim	Original		Scaling-up		OP-change		CN-adding	
		Acc	MFLOPs	Acc	MFLOPs	Acc	MFLOPs	Acc	MFLOPs
NB-101	Vict_1	84.61	37.70	83.47	47.49	-	-	**82.65**	**37.70**
	Vict_2	89.63	128.85	88.25	146.29	-	-	**88.18**	**128.85**
	Vict_3	93.67	446.62	92.95	456.40	-	-	**92.94**	**446.62**
NB-301	Vict_1	92.86	26.95	92.87	27.86	**91.75**	**39.54**	-	-
	Vict_2	93.81	33.40	93.79	35.43	**92.52**	**45.89**	-	-
	Vict_3	94.46	43.24	94.41	44.16	**93.23**	**55.72**	-	-
	Vict_4	94.69	59.76	94.65	63.43	**93.84**	**68.17**	-	-

operation-change strategies and connection-adding strategies perform better than the scaling-up strategies, which can be explained by the different optimization difficulties they might bring. Specifically, if the operation-change strategies are applied, it is challenging for attackers to train a Conv layer to make it function as average pooling or identity mapping, and it is also hard to undo the effect of shortcut adding and layer branch adding during training. However, the effect of scaling-up obfuscation would be limited with the sophisticated training strategies, which again demonstrates that the method used in some existing works, i.e., only enlarging the architectural difference by scaling-up DNNs, is far from enough for the architectural obfuscation.

6 CONCLUSION

This work presents ObfuNAS, a NAS-based algorithmic obfuscation approach, to mitigate malicious DNN architecture extractions. To prevent attackers from training a competitive substitute model, ObfuNAS minimizes the model accuracy of the extracted architecture while still preserving the original inference accuracy of the victim models. As a generic defense approach, ObfuNAS includes seven function-preserving obfuscation strategies that could increase the optimization difficulties. Leveraging the evolutionary search algorithm, this approach can find the best combination of obfuscation strategies for a victim model. Overall, ObfuNAS can achieve 2.6% inference accuracy degradation to attackers with only 0.14× FLOPs overhead, which is 4.7% better than the SOTA work.

REFERENCES

[1] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. 2019. CSI NN: Reverse engineering of neural network architectures through electromagnetic side channel. In *28th USENIX Security Symposium (USENIX Security 19)*. 515–532.

[2] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. 2018. Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning (ICML)*. PMLR, 550–559.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778.

[4] Xin He, Kaiyong Zhao, and Xiaowen Chu. 2021. AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems* 212 (2021), 106622.

[5] Xing Hu, Ling Liang, Shuangchen Li, Lei Deng, Pengfei Zuo, Yu Ji, Xinfeng Xie, Yufei Ding, Chang Liu, Timothy Sherwood, et al. 2020. Deepsniffer: A dnn model extraction framework based on learning architectural hints. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 385–399.

[6] Weizhe Hua, Zhiru Zhang, and G Edward Suh. 2018. Reverse engineering convolutional neural networks through side-channel information leaks. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[7] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning (ICML)*. PMLR, 448–456.

[8] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. 2009. CIFAR-10 (Canadian Institute for Advanced Research). (2009). http://www.cs.toronto.edu/~kriz/cifar.html

[9] Jingtao Li, Zhezhi He, Adnan Siraj Rakin, Deliang Fan, and Chaitali Chakrabarti. 2021. NeurObfuscator: A Full-stack Obfuscation Tool to Mitigate Neural Architecture Stealing. In *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 248–258.

[10] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. 2018. Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)*. 19–34.

[11] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. 2018. Hierarchical representations for efficient architecture search. *6th International Conference on Learning Representations (ICLR)* (2018).

[12] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2019. Darts: Differentiable architecture search. *7th International Conference on Learning Representations (ICLR)* (2019).

[13] Yuntao Liu, Dana Dachman-Soled, and Ankur Srivastava. 2019. Mitigating reverse engineering attacks on deep neural networks. In *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 657–662.

[14] Yukui Luo, Shijin Duan, Cheng Gongye, Yunsi Fei, and Xiaolin Xu. 2022. NNReArch: A Tensor Program Scheduling Framework Against Neural Network Architecture Reverse Engineering. In *2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 1–9.

[15] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. 2019. Regularized evolution for image classifier architecture search. In *The Thirty-Third AAAI Conference on Artificial Intelligence*, Vol. 33. 4780–4789.

[16] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 4510–4520.

[17] Julien Siems, Lucas Zimmer, Arber Zela, Jovita Lukasik, Margret Keuper, and Frank Hutter. 2020. Nas-bench-301 and the case for surrogate benchmarks for neural architecture search. *arXiv preprint arXiv:2008.09777* (2020).

[18] Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. *3rd International Conference on Learning Representations (ICLR)* (2015).

[19] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015. Training very deep networks. *Advances in neural information processing systems* 28 (2015).

[20] Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. 2017. A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the genetic and evolutionary computation conference*. 497–504.

[21] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. 2017. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*.

[22] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. 2019. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2820–2828.

[23] Dilin Wang, Chengyue Gong, Meng Li, Qiang Liu, and Vikas Chandra. 2021. AlphaNet: Improved Training of Supernets with Alpha-Divergence. In *International Conference on Machine Learning (ICML)*. PMLR, 10760–10771.

[24] Dilin Wang, Meng Li, Chengyue Gong, and Vikas Chandra. 2021. Attentivenas: Improving neural architecture search via attentive sampling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 6418–6427.

[25] Xingbin Wang, Rui Hou, Yifan Zhu, Jun Zhang, and Dan Meng. 2019. NPUFort: A secure architecture of DNN accelerator against model inversion attack. In *Proceedings of the 16th ACM International Conference on Computing Frontiers*. 190–196.

[26] Chen Wei, Chuang Niu, Yiping Tang, Yue Wang, Haihong Hu, and Jimin Liang. 2022. Npenas: Neural predictor guided evolution for neural architecture search. *IEEE Transactions on Neural Networks and Learning Systems* (2022).

[27] Junyi Wei, Yicheng Zhang, Zhe Zhou, Zhou Li, and Mohammad Abdullah Al Faruque. 2020. Leaky DNN: Stealing deep-learning model secret with GPU context-switching side-channel. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 125–137.

[28] Wei Wen, Hanxiao Liu, Yiran Chen, Hai Li, Gabriel Bender, and Pieter-Jan Kindermans. 2020. Neural predictor for neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)*. 660–676.

[29] Mengjia Yan, Christopher W Fletcher, and Josep Torrellas. 2020. Cache telepathy: Leveraging shared resource attacks to learn DNN architectures. In *29th USENIX Security Symposium (USENIX Security 20)*. 2003–2020.

[30] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. 2019. Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning (ICML)*. PMLR, 7105–7114.

[31] Honggang Yu, Haocheng Ma, Kaichen Yang, Yiqiang Zhao, and Yier Jin. 2020. Deepem: Deep neural networks model recovery through em side-channel information leakage. In *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 209–218.

[32] Yuankun Zhu, Yueqiang Cheng, Husheng Zhou, and Yantao Lu. 2021. Hermes attack: Steal DNN models with lossless inference accuracy. In *30th USENIX Security Symposium (USENIX Security 21)*. 1973–1988.