

Extracting CNNs in the Unknown-Architecture and Feedback-Agnostic Setting

Jiashuo Liu¹, Ruijie Ma², Manman Li¹, Yi Chen³, Shaozhen Chen¹

¹ Information Engineering University, Zhengzhou, China

jiashuoliu@126.com, limanman15@163.com, chenshaozhen@vip.sina.com

² Department of Computer Science and Technology, Tsinghua University, Beijing, China

marj21@mails.tsinghua.edu.cn

³ Institute for Advanced Study, Tsinghua University, Beijing, China

chenyi2023@tsinghua.edu.cn

Abstract. This paper studies the cryptanalytic extraction of convolutional neural networks (CNNs). Existing cryptanalytic extraction attacks on CNNs assume that the network architecture is known, and try to recover model parameters. In this paper, we prove for the first time that the architecture assumption can be removed for CNNs with both max and average pooling. Our core finding is that the spatial geometry of the weight vectors recovered by existing parameter-recovery attacks naturally leaks the architecture. We formalize this geometry and establish its correspondence with the architectural knowledge of a convolutional layer: (1) The sparsity consistency with the convolution receptive field reveals the layer type, the kernel size, and the stride; (2) The numerical consistency with the kernel parameters reveals the padding mode and the output-channel number; (3) The structural consistency with the pooling operation reveals the pooling type, the window size, and the stride. Although the recovered vectors are obtained using different methods in the raw-output and hard-label settings, their spatial geometry remains the same. Therefore, our architecture recovery is feedback-agnostic: combined with a parameter-recovery attack, it yields a complete cryptanalytic extraction framework that recovers both the architecture and the parameters in the black-box setting. Extensive experiments, including both layer-wise and end-to-end ones, on a wide range of CNNs demonstrate that simultaneously recovering the network architecture and the model parameters is practical.

Keywords: Cryptanalytic Extraction · Convolutional Neural Networks · Network Architecture Recovery.

1 Introduction

A trained deep neural network (DNN) embodies expensive data, computation, and expertise. Consequently, DNNs are typically deployed as a remote black-box service: users can only submit inputs and observe the returned feedback. This

raises the model extraction problem: can an adversary recover a network from such interactions? The most challenging goal of model extraction is functionally equivalent extraction, which requires the extracted model to return the same raw output as the victim model at every point of the input space [14].

The earliest work on model extraction dates to 1991 [2], and functionally equivalent extraction was achieved for logistic regression models in 2016 [22] and for ReLU fully connected networks (FCNs) with 1–2 hidden layers in 2020 [14, 19].

At Crypto 2020, Carlini et al. argued that model extraction is a cryptanalytic problem in disguise [6]. They proposed a differential extraction attack that applies to ReLU FCNs with an arbitrary number of hidden layers and achieves functionally equivalent extraction. This work formally opened the direction of cryptanalytic extraction of neural networks.

Following this work, the cryptography community has studied FCNs under two feedback settings: (1) Raw-output setting: the adversary receives the complete logit vector $\mathcal{F}_{\mathcal{A}, \Theta}(x)$; (2) Hard-label setting: the adversary receives only the predicted label $z(x) = \arg \max_i \mathcal{F}_{\mathcal{A}, \Theta}(x)_i$. In the raw-output setting, the attack has been made polynomial-time [17], run faster in practice [12], extended to deep networks in an end-to-end manner [16], adapted to PReLU and other activation functions [1, 8, 18], and examined under finite-precision evaluation [23]. In the hard-label setting, cryptanalytic extraction has been initiated [7], made polynomial-time [4, 5], extended to the extraction of some deeper layers [3], and re-examined for its query complexity [13].

Convolutional neural networks (CNNs) are among the mainstream models for computer vision and image recognition. It is natural that a recent and important trend is to extend cryptanalytic extraction from FCNs to CNNs. In the raw-output setting, Chen et al. gave the first algebraic treatment of max-pooling CNNs and defined three types of points to recover the convolution parameters [9]; Liu et al. then extracted deeper VGG-style CNNs layer by layer, using receptive-field analysis [15]. In the hard-label setting, Sun et al. handled average-pooling CNNs by reformulating convolutional layers as block-Toeplitz matrices [21]; Chen et al. accelerated the extraction of max-pooling CNNs with the approximate signature vector method and kernel-centric clustering [10].

A DNN is determined by its architecture $\mathcal{A}$ and parameters Θ . The architecture $\mathcal{A}$ specifies the computational structure, namely the layer types, dimensions, and structural hyperparameters. The parameters Θ are the trainable numerical values attached to the structure. All existing attacks on CNNs assume $\mathcal{A}$ to be known and recover only Θ . This assumption is unrealistic, since $\mathcal{A}$ is a valuable asset that a realistic deployment does not disclose.

For an FCN, the unknown architecture information is a list of layer dimensions, and its recovery is settled [4, 11, 19, 20]. By contrast, a CNN exploits local receptive fields and weight sharing to process image data with far fewer parameters. Its architecture is accordingly richer, comprising the layer type, the kernel size, the stride, the padding mode, the channel numbers, and the pooling structure. Each is both a non-trainable parameter affecting network performance and, like the numerical parameters, an asset of the model provider. Because of their

diversity and complexity, a CNN’s architectural knowledge is even more valuable to an adversary and harder to obtain. This leads to the research question of this paper: can the convolutional layers of a CNN be extracted without the knowledge of network architecture? The significance is twofold. (1) Together with existing methods for the fully connected layers, answering it removes the architectural prior from CNN cryptanalytic extraction. (2) It makes architecture recovery independent of parameter recovery, yielding a modular framework that reuses existing parameter-recovery methods.

A CNN consists of convolutional layers followed by one or two fully connected layers, for which recovery methods are already available [4, 11, 19, 20]. We focus on the architecture recovery of the convolutional layers, which remains open. We analyze whether the first step of existing parameter-recovery frameworks, namely point search and equation solving, carries observational information that reflects the convolutional architecture. Since all of these methods share this step [9, 10, 15, 21], our architecture recovery is independent of any particular method.

Our Contributions. To the best of our knowledge, we propose the first architecture recovery method for CNNs. Our key observation is that the spatial structure of the recovered vectors directly reveals the CNN architecture, including all the hyperparameters related to convolution and pooling. Our contributions are summarized as follows.

- **Formal spatial representation.** We give a formal representation of the spatial structure of the recovered vectors (Section 3), built on the first step of the parameter-recovery framework, namely point search and equation solving. Since it is read directly off the recovered vectors, it is independent of the feedback setting.
- **Feedback-agnostic and modular architecture recovery.** We decompose the attack process into several subtasks, as listed in Table 1: identifying the target layer type (Section 5), recovering the convolution kernel size and stride (Section 6), the padding mode and the output channel number (Section 7), and the pooling window size and stride (Section 8). The attack is feedback-agnostic and modular. It uses only the recovered vectors, which the point search and equation solving that open every parameter-recovery pipeline already produce in both the raw-output and the hard-label setting, since the critical points searched in the raw-output setting and the dual points searched in the hard-label setting yield identical recovered vectors. Hence it composes with any existing parameter-recovery attack for CNNs. Combined with such an attack, it yields a complete cryptanalytic extraction framework that recovers each layer’s architecture before its parameters (Figure 3).
- **Experimental evaluation.** We evaluate the attack on the models adopted by existing CNN attacks. In the architecture-unknown setting, the architecture of every tested model is recovered exactly, in both feedback settings and both upstream modes, with only a few additional queries and slightly more running time. To show that the attack handles diverse architectural hyperparameters, we further design an architecturally diverse CNN and present,

task by task, the complete evidence of its architecture recovery through visualizations (Section 9). The complete experimental data and the source code are released.

2 Preliminaries

2.1 Basic Definitions and Notation

We introduce the basic definitions and notations related to convolutional neural networks and cryptanalytic extraction attacks.

Definition 1 (($m+n$)-Deep Convolutional Neural Network) *An ($m+n$)-deep CNN is a map $\mathcal{F}_{\mathcal{A},\Theta} : \mathbb{R}^{d_0} \rightarrow \mathbb{R}^{d_{n+1}}$. It consists of convolutional layers followed by fully connected layers:*

$$\mathcal{F}_{\mathcal{A},\Theta} = \underbrace{f^{(n+1)} \circ \sigma^{(n)} \circ \dots \circ \sigma^{(1)} \circ f^{(1)}}_{\text{Fully connected layers}} \circ \underbrace{\rho^{(m)} \circ \sigma_c^{(m)} \circ f_c^{(m)} \circ \dots \circ \rho^{(1)} \circ \sigma_c^{(1)} \circ f_c^{(1)}}_{\text{Convolutional layers}}.$$

In the convolutional layer l , for $1 \leq l \leq m$, $f_c^{(l)}$, $\sigma_c^{(l)}$, and $\rho^{(l)}$ denote the convolution, ReLU activation, and pooling operations, respectively. In the fully connected layer l , for $1 \leq l \leq n$, $f^{(l)}$ and $\sigma^{(l)}$ denote the affine transformation and ReLU activation. The final output layer is the affine transformation $f^{(n+1)}$.

Definition 2 (Model Architecture) *The model architecture $\mathcal{A}$ is the ordered sequence of layer types, dimensions, and structural hyperparameters. Let L denote the index of the target layer. The architecture information includes*

$$\mathcal{A}^{(L)} = \begin{cases} (\tau^{(L)}, k^{(L)}, s_k^{(L)}, \pi^{(L)}, C_{\text{out}}^{(L)}, \rho^{(L)}, p^{(L)}, s_p^{(L)}), & \tau^{(L)} = \text{Conv}, \\ (\tau^{(L)}, d_{\text{fc}}^{(L)}), & \tau^{(L)} = \text{FC}. \end{cases}$$

The components are specified as follows:

- $\tau^{(L)} \in \{\text{Conv}, \text{FC}\}$ denotes the target-layer type.
- If $\tau^{(L)} = \text{Conv}$, the structural hyperparameters include kernel size $k^{(L)}$ and convolution stride $s_k^{(L)}$, padding mode $\pi^{(L)} \in \{\text{Valid}, \text{Same}\}$, output-channel number $C_{\text{out}}^{(L)}$, and pooling type $\rho^{(L)} \in \{\text{NoPool}, \text{AvgPool}, \text{MaxPool}\}$; when pooling is present, $\mathcal{A}^{(L)}$ includes pooling window size $p^{(L)}$ and pooling stride $s_p^{(L)}$.
- If $\tau^{(L)} = \text{FC}$, $d_{\text{fc}}^{(L)}$ denotes the number of neurons in that layer.

As an example, Figure 1 illustrates a convolutional layer with input channel number $C^{(L)} = 1$ and architecture $\mathcal{A}^{(L)} = (\text{Conv}, 2, 1, \text{Valid}, 1, \text{MaxPool}, 2, 2)$. Throughout, $IN^{(L)} \in \mathbb{R}^{C^{(L)} \times H^{(L)} \times W^{(L)}}$ denotes the input of the convolution operation, where $C^{(L)}$, $H^{(L)}$, and $W^{(L)}$ are its channel number, height, and width, and $K^{(L)} \in \mathbb{R}^{k^{(L)} \times k^{(L)}}$ denotes the convolution kernel.

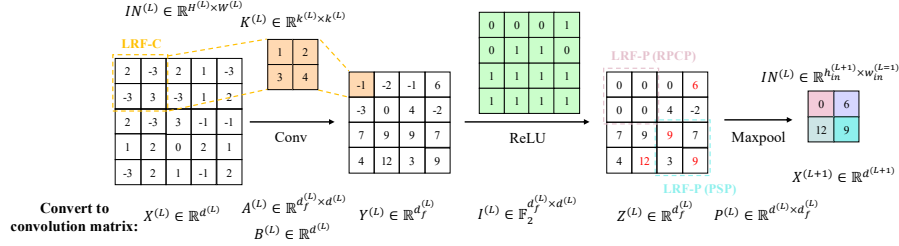


Fig.1. Symbolic view of the example convolutional layer architecture $\mathcal{A}^{(L)} = (\text{Conv}, 2, 1, \text{Valid}, 1, \text{MaxPool}, 2, 2)$. The lower labels are the corresponding notations in the convolution-matrix view. LRF-C (Local Receptive Field of Convolutional Layer) is the sub-matrix of $IN^{(L)}$ multiplied by kernel $K^{(L)}$ to produce the entry $(Y^{(L)})_t$, where $0 \leq t \leq d_f^{(L)} - 1$. LRF-P (Local Receptive Field of Pooling Layer) is the sub-matrix of $Z^{(L)}$ processed by pooling to produce the entry $(X^{(L+1)})_t$, where $0 \leq t \leq d^{(L+1)} - 1$.

2.2 Parameter Recovery in CNNs

We review the extraction framework for FCNs and its extension to CNNs, stating each component only to the extent required by the subsequent analysis. Further technical details are developed where they are used.

Definition 3 (Parameter Extraction Framework [6]) *The parameters are recovered layer by layer. Let $F_x^{(L-1)}$ denote the recovered prefix and $G_x^{(L+1)}$ the unrecovered suffix. In a sufficiently small linear neighborhood of x , the target layer is isolated as $\mathcal{F}_{A,\Theta}(x) = G_x^{(L+1)}(A^{(L)}F_x^{(L-1)}(x) + B^{(L)})$. Denote by $\hat{A}_i^{(L)}$ the final recovered weight of the i -th neuron in layer L . The framework is decomposed into two sequential phases:*

- **Neuron Signature Recovery:** *For each neuron, the attacker performs the following steps: point collection, equation solving, clustering and sorting, and merging. For a ReLU-induced point, the recovered vector $\tilde{A}_i^{(L)}$ is proportional to the neuron signature, with $\tilde{A}_i^{(L)} \in \{\hat{A}_i^{(L)}, -\hat{A}_i^{(L)}\}$ after normalization.*
- **Neuron Sign Recovery:** *Checking whether $\tilde{A}_i^{(L)} = \hat{A}_i^{(L)}$ or $\tilde{A}_i^{(L)} = -\hat{A}_i^{(L)}$.*

At EUROCRYPT 2024, Canales-Martínez et al. [17] stated that the framework in Definition 3 also works for CNNs without modification. Each placement of the convolution kernel is regarded as a convolution neuron whose weight vector is nonzero only on its LRF-C (see Figure 1). Stacking these weight vectors yields Definition 4.

Definition 4 (Convolution Matrix [9]) *Denote the convolution matrix by $A^{(L)}$. Let $w_o^{(L)}$ denote the output width of the convolution at layer L , i.e., $\lceil W^{(L)}/s_k^{(L)} \rceil$ under Same padding and $\lfloor (W^{(L)} - k^{(L)})/s_k^{(L)} \rfloor + 1$ under Valid*

padding. Let $C^{(L)} = C_{\text{out}}^{(L)} = 1$; for $0 \leq i \leq d_f^{(L)} - 1$, define the i -th row of the convolution matrix $A^{(L)}$ in two cases.

$$\begin{aligned} - \pi^{(L)} = \text{Valid} : A_i^{(L)} &= \underbrace{(0, \dots, 0, K_0^{(L)}, 0, \dots, 0, \dots, 0, \dots, 0, K_{k^{(L)}-1}^{(L)}, 0, \dots, 0)}_{|\bar{A}_{i,1}^{(L)}| = W^{(L)}}, \\ - \pi^{(L)} = \text{Same} : & \end{aligned}$$

$$A_i^{(L)} = \left(\underbrace{0, \dots, 0}_{(W^{(L)}+2t_{\text{pad}}^{(L)})t_{\text{pad}}^{(L)}}, \underbrace{0, \dots, 0}_{t_{\text{pad}}^{(L)}}, \underbrace{\bar{A}_{i,1}^{(L)}, 0, \dots, 0}_{t_{\text{pad}}^{(L)}}, \dots, \underbrace{0, \dots, 0}_{t_{\text{pad}}^{(L)}}, \underbrace{\bar{A}_{i,k^{(L)}}^{(L)}, 0, \dots, 0}_{t_{\text{pad}}^{(L)}}, \underbrace{0, \dots, 0}_{(W^{(L)}+2t_{\text{pad}}^{(L)})t_{\text{pad}}^{(L)}} \right),$$

$$\text{where } t_{\text{pad}}^{(L)} = \frac{1}{2} \left(\left(\left\lceil W^{(L)} / s_k^{(L)} \right\rceil - 1 \right) s_k^{(L)} + k^{(L)} - W^{(L)} \right).$$

Let $\mathcal{K}_i^{(L)}$ index the positions of the segments $\bar{A}_{i,1}^{(L)}, \dots, \bar{A}_{i,k^{(L)}}^{(L)}$ in $A_i^{(L)}$:

$$\mathcal{K}_i^{(L)} = \left\{ \left(t_{\text{pad}}^{(L)} + \left\lfloor \frac{i}{w_o^{(L)}} \right\rfloor \cdot s_k^{(L)} + j \right) (W^{(L)} + 2t_{\text{pad}}^{(L)}) + t_{\text{pad}}^{(L)} + (i \bmod w_o^{(L)}) \cdot s_k^{(L)} + m \mid 0 \leq j < k^{(L)}, 0 \leq m < k^{(L)} \right\}.$$

Each entry of the input tensor $\text{IN}^{(L)}$ is indexed by a triple (q, u, v) , with q the channel index and (u, v) the spatial position. Write $\Lambda^{(L)} = \{0, \dots, H^{(L)} - 1\} \times \{0, \dots, W^{(L)} - 1\}$, $\Omega^{(L)} = \{0, \dots, C^{(L)} - 1\} \times \Lambda^{(L)}$ for the spatial and the channel-spatial coordinate spaces. To write the convolution as a matrix operation, the input tensor $\text{IN}^{(L)}$ is first flattened into the one-dimensional space $\mathcal{I}^{(L)} = \{0, \dots, d^{(L)} - 1\}$; the flattened input is denoted by $x^{(L)}$. According to the padding mode $\pi^{(L)}$, each spatial dimension is then enlarged by $t_{\text{pad}}^{(L)}$ zero entries on both sides, and the resulting tensor is flattened into a vector indexed by the padded space $\tilde{\mathcal{I}}^{(L)}$. The row-major bijection $\psi^{(L)} : \mathcal{I}^{(L)} \rightarrow \Omega^{(L)}$, $\psi^{(L)}(i) = (q, u, v)$, identifies $\mathcal{I}^{(L)}$ with $\Omega^{(L)}$. Hence the padded indices $\tilde{\mathcal{I}}^{(L)}$ serve only to write the convolution matrix, whereas the entries of $\text{IN}^{(L)}$ live in the unpadded space $\mathcal{I}^{(L)}$. The two are related by the padding map $\phi^{(L)} : \mathcal{I}^{(L)} \hookrightarrow \tilde{\mathcal{I}}^{(L)}$, which sends an unpadded index to the padded index of the same position, i.e. $(q, u, v) \mapsto (q, u + t_{\text{pad}}^{(L)}, v + t_{\text{pad}}^{(L)})$ in coordinates. For a set $T \subseteq \tilde{\mathcal{I}}^{(L)}$ we write $(\phi^{(L)})^{-1}(T) = \{j \in \mathcal{I}^{(L)} : \phi^{(L)}(j) \in T\}$ for its preimage, which discards the padded positions.

Based on the convolution matrix, Sun et al. [21] apply the parameter extraction framework to ReLU CNNs with average pooling. However, Chen et al. [9] pointed out that the framework is not directly applicable to CNNs with max pooling, because the selection of the maximum value in LRF-P introduces additional nonlinearity. By jointly accounting for ReLU activations and max-pooling selections, prior work extends the critical point [9] and the dual point [10].

Definition 5 (Critical Points [9]) A CNN has three critical point types:

- A ReLU-pooling critical point (RPCP) is an input x for which the input to the i -th ReLU vanishes in convolutional layer L , $A_i^{(L)} x^{(L)} + (B^{(L)})_i = 0$; this value is termed a convolution response. With max pooling, this response must also be the maximum in its LRF-P.

- A pooling switching point (PSP) is an input x at which two responses in one LRF-P tie for the maximum in convolutional layer L : $A_{r_1}^{(L)}x^{(L)} + (B^{(L)})_{r_1} = A_{r_2}^{(L)}x^{(L)} + (B^{(L)})_{r_2}$.
- A fully connected critical point (FCP) is an input x for which the input to the i -th ReLU vanishes in fully connected layer L , $A_i^{(L)}x^{(L)} + (B^{(L)})_i = 0$.

LRF-P instances of the RPCP and PSP are shown in Figure 1.

Definition 6 (Decision-Boundary Point and Dual Point [10]) A decision-boundary point is an input x where the two largest logits coincide, i.e. for some $i \neq j$:

$$\mathcal{F}_{\mathcal{A},\Theta}(x)_i = \mathcal{F}_{\mathcal{A},\Theta}(x)_j = \max \mathcal{F}_{\mathcal{A},\Theta}(x).$$

A dual point is both a decision-boundary point and a critical point. Since critical points are of three types, RPCP, PSP, and FCP, so are dual points: dual RPCP, dual PSP, and dual FCP.

For each critical or dual point x , the attacker samples small perturbations $\vec{\delta}$ and computes $\mathcal{H}(x; \vec{\delta}) = \mathcal{F}_{\mathcal{A},\Theta}(x + \vec{\delta}) + \mathcal{F}_{\mathcal{A},\Theta}(x - \vec{\delta}) - 2\mathcal{F}_{\mathcal{A},\Theta}(x)$. These values form a linear system, which is solved to obtain the recovered vector v_x [6].

Lemma 1 (Recovered Vectors in CNNs [9, 10, 15]). For a critical or dual point x , there exists a nonzero scalar g_x such that the recovered vector v_x reads

$$v_x = \begin{cases} g_x A_{m,i}^{(L)}[\mathcal{K}_i^{(L)}], & x \text{ is a (dual) RPCP at layer } L, \\ g_x (A_{m,i}^{(L)} - A_{m,j}^{(L)})[\mathcal{K}_i^{(L)} \cup \mathcal{K}_j^{(L)}], & x \text{ is a (dual) PSP at layer } L, \\ g_x A_i^{(L)}, & x \text{ is a (dual) FCP at layer } L, \end{cases} \quad (1)$$

where m denotes the output-channel index.

For a (dual) RPCP or (dual) FCP, the recovered vector has the same form as in an FCN. Hence, the parameter extraction framework in Definition 3 directly recovers the neuron signature up to a nonzero scalar.

For a (dual) PSP, the support of v_x selects exactly two rows of $A_m^{(L)}$; let these be the rows $A_{m,i}^{(L)}$ and $A_{m,j}^{(L)}$ associated with the spatial positions $i < j$. The nonzero entries of the two rows are shifted copies of the same kernel. The two input patches at the nonzero positions of $A_{m,i}^{(L)}$ and $A_{m,j}^{(L)}$ have equal inner products with the convolution kernel. Collecting all such equalities yields a linear system in $K^{(L)}$ [9].

The neuron sign recovery method in the framework can be adapted to CNNs. Moreover, the convolutional structure admits more specialized methods, see [9, 10, 15] for details.

Since critical points and dual points yield the same recovered vectors, the following discussion uses critical points as illustrative examples.

3 Auxiliary Concepts of Recovered Vectors in CNNs

Based on the correspondence between the convolution kernel and the row vectors of the convolution matrix (Definition 4), this section describes the spatial properties of the recovered vector and formalizes the auxiliary definitions used in the subsequent analysis.

3.1 Structural Properties of Recovered Vectors

According to Eq. (1), the recovered vector v_x is either one row of the convolution matrix or the difference of two rows. The kernel entries in each row are confined to the corresponding LRF-C, while all entries outside the LRF-C are zero. We introduce two sets, S_x and $\mathcal{W}_i^{(L)}$, to characterize the relationship between recovered vectors and the convolution matrix.

Definition 7 (Effective Position Set of a Recovered Vector) *For a recovered vector v_x , S_x is defined as $S_x = \{i \in \mathcal{I}^{(L)} : (v_x)_i \neq 0\}$.*

Definition 8 (Effective Position Set of a Convolution-Matrix Row) *For the i -th row $A_{m,i}^{(L)}$ of the m -th output channel, $\mathcal{W}_i^{(L)} \subseteq \mathcal{I}^{(L)}$ is defined as $\mathcal{W}_i^{(L)} = (\phi^{(L)})^{-1}(\{j \in \tilde{\mathcal{I}}^{(L)} : (A_{m,i}^{(L)})_j \neq 0\} \cap \mathcal{K}_i^{(L)})$, that is, the input positions of the i -th LRF-C; it is independent of m , since all output channels share the same LRF-Cs.*

Proposition 1 (Shared Sparsity Pattern per LRF-C). *For a target-layer RPCP, PSP, or FCP, the following equation holds:*

$$S_x = \begin{cases} \mathcal{W}_i^{(L)}, & x \text{ is an RPCP associated with the } i\text{-th LRF-C,} \\ \mathcal{W}_i^{(L)} \cup \mathcal{W}_j^{(L)}, & x \text{ is a PSP associated with the } i\text{-th and } j\text{-th LRF-Cs,} \\ \mathcal{I}^{(L)}, & x \text{ is an FCP.} \end{cases}$$

Consequently, RPCPs associated with the same LRF-C, as well as PSPs associated with the same pair of LRF-Cs, have the same sparsity pattern.

Proof. By Eq. (1), v_x equals g_x times the restricted row (resp. row difference) and is supported on $\mathcal{K}_i^{(L)}$ (resp. $\mathcal{K}_i^{(L)} \cup \mathcal{K}_j^{(L)}$), so restriction preserves the support: $S_x = \mathcal{W}_i^{(L)}$ for an RPCP. For a PSP, the two rows are shifted copies of the same kernel, so for a generic kernel no cancellation occurs on the overlap and $S_x = \mathcal{W}_i^{(L)} \cup \mathcal{W}_j^{(L)}$. The FCP case is immediate, since a fully connected row is dense and $S_x = \mathcal{I}^{(L)}$.

Proposition 2 (Proportional Recovered Vectors per LRF-C). *For a target-layer RPCP, PSP, or FCP, the following equation holds for every $t \in S_x$:*

$$(v_x)_t = \begin{cases} g_x(A_{m,i}^{(L)})_{\phi^{(L)}(t)}, & x \text{ is an RPCP,} \\ g_x(A_{m,i}^{(L)} - A_{m,j}^{(L)})_{\phi^{(L)}(t)}, & x \text{ is a PSP,} \\ g_x(A_i^{(L)})_t, & x \text{ is an FCP.} \end{cases}$$

Proof. On the effective positions, Eq. (1) expresses $v_x[S_x]$ as g_x times the restricted row (resp. row difference) identified in Proposition 1, and the scalar g_x depends only on x ; hence two points associated with the same LRF-C (resp. the same pair of LRF-Cs) have proportional recovered vectors.

For each input channel $q \in \{0, \dots, C^{(L)} - 1\}$, let $S_x^{[q]}$ denote the subset of S_x located in the q -th input-channel block.

Proposition 3 (Equal Channel Slices of the Support). *For a target-layer RPCP or PSP, the following equation holds:*

$$S_x^{[q]} = S_x^{[q']}, q, q' \in \{0, \dots, C^{(L)} - 1\}, q \neq q', \quad S_x^{[q]} \neq \emptyset, S_x^{[q']} \neq \emptyset.$$

Proof. By Eq. (1), RPCP rows and PSP row differences repeat the same spatial kernel pattern in every input-channel block. Hence, all input-channel slices have identical effective position sets.

By Proposition 3, multiple input channels do not affect the spatial sparsity pattern of recovered vectors. For clarity, the following discussion assumes a single input channel: $\Omega^{(L)} = \Lambda^{(L)}$.

3.2 Auxiliary Definitions

To characterize the spatial properties of recovered vectors in the following analysis, we introduce several auxiliary definitions. We first classify PSPs into three types according to the relative positions of their corresponding LRF-Cs.

Definition 9 (PSP Types) *For a target-layer PSP, let (r_1, c_1) and (r_2, c_2) be the starts of the two associated LRF-Cs in $\Lambda^{(L)}$. They satisfy $\psi^{(L)}(\min \mathcal{W}_i^{(L)}) = (r_1, c_1)$ and $\psi^{(L)}(\min \mathcal{W}_j^{(L)}) = (r_2, c_2)$. The positions define three PSP types:*

- A horizontal PSP satisfies $r_1 = r_2$ and $c_1 \neq c_2$; its two LRF-Cs' starts lie on the same row.
- A vertical PSP satisfies $r_1 \neq r_2$ and $c_1 = c_2$; its two LRF-Cs' starts lie on the same column.
- An off-axis PSP satisfies $r_1 \neq r_2$ and $c_1 \neq c_2$; its two LRF-Cs' starts differ in both spatial directions.

To further describe the spatial geometry of recovered vectors, we formalize the smallest axis-aligned box containing their effective positions and the binary pattern within that box.

Definition 10 (Spatial Box and Shape of a Recovered Vector) *For a recovered vector v_x , define its spatial bounding box, denoted by Box_x , as*

$$\text{Box}_x = [u_{\min}, u_{\max}] \times [v_{\min}, v_{\max}],$$

where $u_{\min} = \min_{s \in S_x} [\psi^{(L)}(s)]_u$ and $u_{\max} = \max_{s \in S_x} [\psi^{(L)}(s)]_u$, with $v_{\min}$ and $v_{\max}$ defined in the same way. The corresponding shape is $\text{Shape}_x = (h_x, w_x)$, where $h_x = u_{\max} - u_{\min} + 1$, $w_x = v_{\max} - v_{\min} + 1$.

Definition 11 (Effective Mask of a Recovered Vector) For a recovered vector v_x , the effective mask is a binary matrix $\text{Mask}_x \in \{0, 1\}^{h_x \times w_x}$. Let $\psi^{(L)}(S_x)$ denote the element-wise spatial mapping of S_x . The (i, j) -th element is expressed as

$$(\text{Mask}_x)_{i,j} = \left[(u_{\min} + i, v_{\min} + j) \in \psi^{(L)}(S_x) \right], \quad 0 \leq i < h_x, \quad 0 \leq j < w_x.$$

To understand how Definitions 9–11 work, Figure 2 illustrates an RPCP and the three PSP types, together with graphical representations of their respective LRF-Cs, Boxes, and Masks.

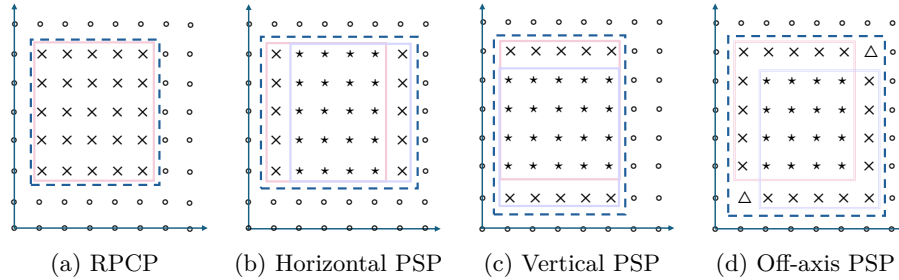


Fig. 2. Examples of the four point types under the architecture $\mathcal{A}^{(L)} = (\text{Conv}, 5, 1, \text{Valid}, 1, \text{MaxPool}, 2, 2)$. Solid lines denote LRF-Cs, and dashed lines denote the Boxes. Positions marked with $\times$ and $\star$ are effective positions in Mask_x , where $\star$ further denotes the overlapping positions of the two LRF-Cs; a position marked with $\circ$ lies outside the corresponding Box and satisfies $v_x = 0$; and $\triangle$ marks a position that lies inside the corresponding Box but satisfies $v_x = 0$.

4 Our Cryptanalytic Extraction Framework for CNNs

4.1 Adversarial Goals and Assumptions

A cryptanalytic extraction attack involves an adversary and an oracle: the adversary submits generated inputs to the oracle and exploits the feedback to recover a model functionally equivalent to the victim model [6]. Existing cryptanalytic extraction attacks on CNNs [9, 10, 15, 21] recover only the parameters Θ , assuming that the architecture $\mathcal{A}$ is known in advance. However, such architectural information is typically unavailable to the adversary.

Goals. We propose a complete cryptanalytic extraction framework for CNNs with two components: architecture recovery and parameter recovery. The key is to achieve the architecture-recovery attack and tailor it to be directly compatible with the parameter extraction framework (Definition 3), adapting to existing

parameter recovery methods. Within the architecture-recovery attack, guided by the architecture knowledge in Definition 2, we decompose the goal into the tasks summarized in Table 1.

Table 1. Task decomposition for architecture recovery.

Task	Description	Section
1	Identify whether the target layer is convolutional or fully connected, yielding $\hat{\tau}^{(L)} \in \{\text{Conv}, \text{FC}\}$.	§5
<i>Convolutional-layer architecture recovery</i> ($\hat{\tau}^{(L)} = \text{Conv}$)		
2	Recover the convolution kernel size $\hat{k}$.	§6
3	Recover the convolution stride $\hat{s}_k$.	§6
4	Recover the padding mode $\hat{\pi} \in \{\text{Valid}, \text{Same}\}$.	§7
5	Recover the output-channel number $\hat{C}_{\text{out}}$.	§7
6	Recover the pooling type $\hat{p} \in \{\text{NoPool}, \text{AvgPool}, \text{MaxPool}\}$, window size $\hat{p}$, and stride $\hat{s}_p$.	§8
<i>Fully-connected-layer architecture recovery</i> ($\hat{\tau}^{(L)} = \text{FC}$)		
7	Recover the layer width $\hat{d}_{\text{fc}}^{(L)}$.	[4, 11, 19, 20]

Note. After identifying the layer type in Task 1, the attack follows either the convolutional branch (Tasks 2–6) or the fully connected branch (Task 7).

Assumptions. We make the following assumptions about the oracle and the capabilities of the adversary.

- **Full-domain input.** The adversary can query arbitrary inputs $x \in \mathbb{R}^{d_0}$.
- **Precise computation.** The network is specified and evaluated using 64-bit floating-point arithmetic.
- **ReLU activation.** All activation functions are ReLUs.

In our setting, the adversary knows only the input dimension d_0 and the output dimension d_{n+1} ; the types and hyperparameters of the hidden layers (Definition 2) remain unknown and are to be recovered.

4.2 Complete Cryptanalytic Extraction Framework

New Framework. Figure 3 compares the original CNN parameter-recovery framework with our complete cryptanalytic extraction framework constructed in the architecture-unknown scenario.

In our complete framework, the attack operates within a single target layer. For this layer, the first two steps are shared with its parameter recovery, and their procedure is decided by the feedback mode of the model. A CNN has two types of

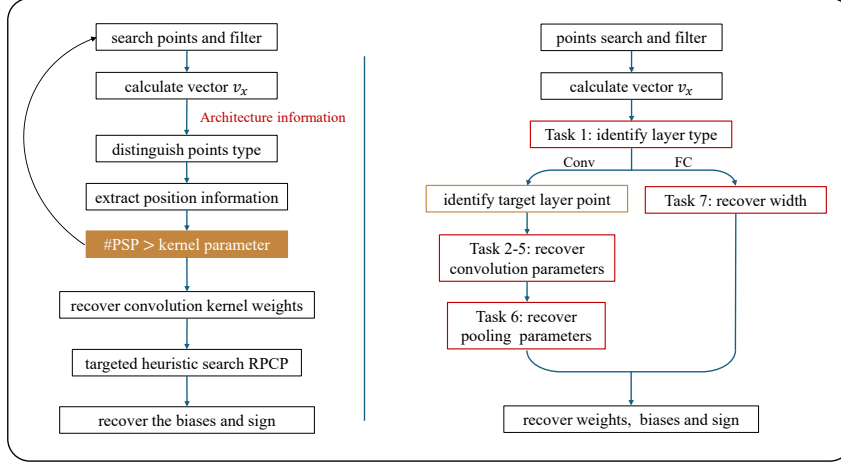


Fig. 3. Left: Original CNN parameter-recovery framework under the architecture-known scenario. Right: Our cryptanalytic extraction framework under the architecture-unknown scenario.

layers, convolutional layers and fully connected layers. We first identify the layer type. If the current layer is fully connected, its width is recovered by existing methods. If it is convolutional, we recover its architecture hyperparameters one by one in Tasks 2 to 6. After the architecture recovery is completed, we can connect to the existing parameter-recovery methods for CNNs.

Core Idea of the Architecture Recovery. Our attack exploits the geometric and numerical relationships among the convolution kernel, the convolution matrix and the recovered vectors. A recovered vector is a row of the convolution matrix, or the difference of two rows (Eq. (1)), and every row of the convolution matrix has support only on its corresponding LRF-C.

Convolutional and fully connected layers differ in the types of points they produce. Therefore, the existence of RPCPs and PSPs is a sufficient condition for identifying the target layer as convolutional.

Consider the recovered vector of an RPCP. Its effective position set $S_x = \mathcal{W}_i^{(L)}$ is a single LRF-C of size $k^{(L)} \times k^{(L)}$, so the shape of its Box reveals the kernel size $k^{(L)}$. It also follows that the offsets between the Box starting points of any two such points are integer multiples of the stride $s_k^{(L)}$ within $\Omega^{(L)}$, and sufficiently dense sampling recovers $s_k^{(L)}$.

Same padding adds zeros around the input and enlarges the effective input space, so the LRF-Cs at the boundaries extend into the padding area, the effective position sets of the boundary recovered vectors shrink, and their Boxes are truncated. Valid padding leaves no such area and no Box is truncated, and this difference in the boundary Boxes distinguishes the two padding modes.

A PSP originates from a switch of the max-pooling selection, and it therefore indicates that the target layer uses max pooling. The max-pooling parameters are recovered following the same idea as the convolution-kernel parameters.

Average pooling spreads every value of an LRF-P over the $p \times p$ window, so each coefficient of the recovered vector is spread evenly over that window, and the extent of the spread gives p . When layer L contains no pooling, the recovered vector retains the sparse convolution structure, so no such repeated-block pattern appears.

Complexity of the Architecture Recovery. The number of points required by our architecture recovery is determined by the identification tasks, not by the parameter count of the target layer. In the existing parameter-recovery framework for CNNs, the recovered vectors are stacked into a linear system that is solved for the kernel of each output channel. For every output channel, the number of PSPs must exceed the number of parameters of that channel’s kernel $(k^{(L)})^2$. In contrast, the architecture recovery exploits only the qualitative signatures of the recovered vectors, and it suffices to find the minimal cluster of PSPs or RPCPs whose numerical ratios concentrate on a single candidate. Therefore, a few RPCPs and PSPs are enough for all identification tasks, and the additional query cost is a constant independent of $(k^{(L)})^2$.

5 Identifying the Target Layer Type

In this section, our task is to identify whether the target layer is convolutional or fully connected, yielding $\hat{\tau}^{(L)} \in \{\text{Conv}, \text{FC}\}$. Following the layer-by-layer paradigm of the cryptanalytic extraction framework, let $\mathcal{P}^{(L)}$ denote the set of candidate points retained for layer L after filtering out the points attributed to the recovered shallower layers, and let $\mathcal{P}_{\text{RPCP}}^{(L)}$, $\mathcal{P}_{\text{PSP}}^{(L)}$, and $\mathcal{P}_{\text{FCP}}^{(L)}$ denote the RPCP, PSP, and FCP subsets of the points that actually arise from the target layer L . According to the origins of the points in Definition 5, we propose Lemma 2 to characterize the relationship between the layer type and the point types.

Lemma 2 (Layer Type Characterized by Point Existence). *Under the assumed CNN ordering in Definition 1, the existence of an RPCP or PSP in the target layer or deeper layers indicates $\hat{\tau}^{(L)} = \text{Conv}$:*

$$\mathcal{P}_{\text{RPCP}}^{(L)} \cup \mathcal{P}_{\text{PSP}}^{(L)} \cup \bigcup_{L' > L} (\mathcal{P}_{\text{RPCP}}^{(L')} \cup \mathcal{P}_{\text{PSP}}^{(L')}) \neq \emptyset \implies \hat{\tau}^{(L)} = \text{Conv}.$$

Proof. All convolutional layers precede the fully connected layers. If L is fully connected, both L and every deeper hidden layer are fully connected. Hence only fully connected layers can occur: $\mathcal{P}^{(L)} = \mathcal{P}_{\text{FCP}}^{(L)} \cup \bigcup_{L' > L} \mathcal{P}_{\text{FCP}}^{(L')}$. In conclusion, an observed RPCP or PSP in the target layer or deeper layers excludes the fully connected case, yielding $\hat{\tau}^{(L)} = \text{Conv}$.

Lemma 2 transforms Task 1 into identifying the point type. In the following, we study how to identify RPCPs, PSPs, and FCPs arising from the target layer or deeper layers.

5.1 Identifying Point Type

We separately discuss the spatial properties of the recovered vectors obtained in the target-layer input for the six kinds of points: RPCPs, PSPs, and FCPs, each arising from the target layer or from a deeper layer.

Identifying RPCPs. The spatial characteristics of a target-layer RPCP’s recovered vector are given by Proposition 1. We formalize them as Corollary 1.

Corollary 1 (*Box, Shape, and Mask of Target-Layer RPCPs*). *For a target-layer RPCP x , the following hold: $\text{Box}_x = [r_x, r_x + k^{(L)} - 1] \times [c_x, c_x + k^{(L)} - 1]$, $\text{Shape}_x = (k^{(L)}, k^{(L)})$, $\text{Mask}_x = \mathbf{1}_{k^{(L)} \times k^{(L)}}$.*

For an RPCP x arising from a deeper convolutional layer $L' > L$, the LRF-C of x at layer L' projects onto the target-layer input space $\Lambda^{(L)}$, so the spatial projection $\psi^{(L)}(S_x)$ of its support set becomes a union of LRF-Cs of layer L .

Proposition 4 (Deeper-Layer RPCP Supports Are Unions of LRF-Cs).

For an RPCP x arising from a deeper convolutional layer $L' > L$, $\psi^{(L)}(S_x) = \bigcup_{m \in \mathcal{M}_x} ([r_m, r_m + k^{(L)} - 1] \times [c_m, c_m + k^{(L)} - 1])$, where $\mathcal{M}_x$ indexes the LRF-Cs of layer L whose responses contribute to x . Consequently,

$$\begin{aligned} \text{Box}_x &= [\min_m r_m, \max_m r_m + k^{(L)} - 1] \times [\min_m c_m, \max_m c_m + k^{(L)} - 1], \\ \text{Shape}_x &= (\max_m r_m - \min_m r_m + k^{(L)}, \max_m c_m - \min_m c_m + k^{(L)}), \end{aligned}$$

and Mask_x indicates the positions covered by the union within Box_x .

Proof. We prove by recursion on the layer distance $L' - L$.

1. **Base case ($L' = L + 1$).** Each convolution response of layer $L + 1$ is computed from the $p^{(L)} \times p^{(L)}$ convolution responses of layer L within one LRF-P of layer L (pooling window $p^{(L)}$, stride $s_p^{(L)}$). Under AvgPool, every response of layer L within the LRF-P is propagated, and the overlapping LRF-Cs of the propagated responses fill the enclosing rectangle. Under MaxPool, only the selected maximum responses are propagated, and the uncovered positions may remain. In both cases, S_x is a union of LRF-Cs of layer L . The NoPool case is analogous to the AvgPool case but simpler.
2. **Recursive step ($L' > L + 1$).** By the recursive hypothesis, S_x in the input coordinates of layer $L' - 1$ is a union of LRF-Cs of layer $L' - 1$. Each LRF-C of layer $L' - 1$ in turn projects onto layer $L' - 2$ as a union of LRF-Cs of layer $L' - 2$ by the one-layer argument above. Repeating this projection until the target-layer input is reached, S_x is a union of LRF-Cs of layer L .

In both cases, Box_x is the smallest rectangle enclosing these LRF-Cs and Mask_x indicates the covered positions, which gives the stated structure.

Identifying PSPs. The spatial characteristics of a PSP’s recovered vector are given by Proposition 1. We formalize them as Corollary 2.

Corollary 2 (Box, Shape, and Mask of Target-Layer PSPs). *For a target-layer PSP x with $\psi^{(L)}(\min \mathcal{W}_i^{(L)}) = (r_1, c_1)$, $\psi^{(L)}(\min \mathcal{W}_j^{(L)}) = (r_2, c_2)$, the following hold:*

$$\begin{aligned} \text{Box}_x &= [\min\{r_1, r_2\}, \max\{r_1, r_2\} + k^{(L)} - 1] \times [\min\{c_1, c_2\}, \max\{c_1, c_2\} + k^{(L)} - 1], \\ \text{Shape}_x &= (|r_2 - r_1| + k^{(L)}, |c_2 - c_1| + k^{(L)}), \end{aligned}$$

and Mask_x indicates the positions covered by the union of the two LRF-Cs.

For a deeper-layer PSP, the recovered vector in layer L' is the difference of two convolution matrix rows, so the LRF-C union argument of Proposition 4 applies verbatim.

Proposition 5 (Deeper-Layer PSP Supports Are Unions of LRF-Cs). *For a PSP x arising from a convolutional layer $L' > L$, the following equation holds:*

$$\psi^{(L)}(S_x) = \bigcup_{i \in \mathcal{M}_x} ([r_i, r_i + k^{(L)} - 1] \times [c_i, c_i + k^{(L)} - 1]),$$

where $\mathcal{M}_x$ indexes the LRF-Cs of layer L associated with the two rows that form the recovered vector of x . Consequently, a deeper-layer PSP has the same union-of-LRF-Cs structure as a deeper-layer RPCP, and Mask_x indicates the positions covered within Box_x .

Proof. The idea is the same as in Proposition 4, except that S_x is the union over the LRF-Cs of the two rows forming the recovered vector of x .

Identifying FCPs. The spatial characteristics of an FCP’s recovered vector are given by Proposition 1. We formalize them as Corollary 3.

Corollary 3 (Box, Shape, and Mask of FCPs). *For a target-layer or deeper-layer FCP x , the following hold:*

$$S_x = \mathcal{I}^{(L)}, \quad \text{Box}_x = \Lambda^{(L)}, \quad \text{Shape}_x = (H^{(L)}, W^{(L)}), \quad \text{Mask}_x = \mathbf{1}_{H^{(L)} \times W^{(L)}}.$$

5.2 Method for Layer-Type Identification

We define the full-box point set as $\mathcal{P}_{\text{full}} = \{x \in \mathcal{P}^{(L)} : \text{Shape}_x = (H^{(L)}, W^{(L)})\}$. The remaining points form the set $\mathcal{P}_{\text{conv}} = \mathcal{P}^{(L)} \setminus \mathcal{P}_{\text{full}}$. The results above determine the Box of every point in $\mathcal{P}^{(L)}$. Combined with Lemma 2, they yield the following decision rule.

Every point $x \in \mathcal{P}^{(L)}$ with $\text{Box}_x \neq \Lambda^{(L)}$ is an RPCP or a PSP, because an FCP satisfies $\text{Box}_x = \Lambda^{(L)}$. Therefore, if $\mathcal{P}_{\text{conv}} \neq \emptyset$, then $\hat{\tau}^{(L)} = \text{Conv}$.

6 Recovering the Convolution Kernel Size and Stride

In this section, our tasks are to recover the convolution kernel size and the convolution stride, yielding $\hat{k}$ and $\hat{s}_k$. In a convolutional network, the size of each LRF-C is $k^{(L)} \times k^{(L)}$. Under the convolution-matrix representation, the set $\{j \in \mathcal{I}^{(L)} : (A_i^{(L)})_j \neq 0\}$ is exactly the LRF-C corresponding to the i -th row. By Proposition 1, S_x coincides with this LRF-C index set under Valid padding, whereas it does not under Same padding. We therefore define an interior set to exclude the potential influence of Same padding:

$$\tilde{\mathcal{P}}_{\text{conv}}^{\circ} = \left\{ x \in \mathcal{P}_{\text{conv}} : \text{Box}_x \subseteq \{1, \dots, H^{(L)} - 2\} \times \{1, \dots, W^{(L)} - 2\} \right\}.$$

We propose Lemma 3 to characterize the relationship between $k^{(L)}$ and Shape_x .

Lemma 3 (Kernel Size Characterized by Shape and Mask). *For an interior target-layer point $x \in \tilde{\mathcal{P}}_{\text{conv}}^{\circ}$, the following equation holds:*

$$k^{(L)} = \begin{cases} h_x = w_x, & x \text{ is an RPCP,} \\ b_{\partial}(\text{Mask}_x) = \min\{h_x, w_x\}, & x \text{ is a horizontal or vertical PSP,} \\ b_{\partial}(\text{Mask}_x), & x \text{ is an off-axis PSP,} \end{cases}$$

where $b_{\partial}(\text{Mask}_x)$ denotes the length of the shortest contiguous nonzero segment on the boundary of Mask_x .

Proof. It follows directly from Corollaries 1 and 2.

For an interior point x , $\min S_x$ is the position of $(K_0^{(L)})_0$ in the convolution matrix, so $\psi^{(L)}(\min S_x)$ is the start position of an LRF-C. We propose Lemma 4 to characterize the relationship between $s_k^{(L)}$ and the starting positions.

Lemma 4 (Stride Characterized by Effective Position Sets). *For any two interior target-layer points $x_1, x_2 \in \tilde{\mathcal{P}}_{\text{conv}}^{\circ}$:*

$$r_{x_1} - r_{x_2} \in s_k^{(L)}\mathbb{Z}, c_{x_1} - c_{x_2} \in s_k^{(L)}\mathbb{Z}.$$

Proof. Since convolution windows advance by $s_k^{(L)}$ in both directions, the differences between the starting positions of any two windows, in both the row and the column direction, are integer multiples of $s_k^{(L)}$.

Lemmas 3 and 4 transform Tasks 2 and 3 into identifying the target-layer point. In the following, we study how to identify the target-layer RPCPs and PSPs from $\tilde{\mathcal{P}}_{\text{conv}}^{\circ}$.

6.1 Identifying the Target Points

We separately discuss the spatial and numerical properties of the recovered vectors obtained in the target-layer input.

Identifying Target-layer RPCP. The difference between the recovered vectors of RPCPs and PSPs is whether v_x equals one row of the convolution matrix or the difference of two rows (Lemma 1). Since the rows of the convolution matrix are shifted copies of the same kernel, a direct observation is that the support set S_x of a target-layer PSP is larger than that of a target-layer RPCP. Define the point set $\tilde{\mathcal{P}}_{\text{conv}}^\circ(k') = \{x \in \tilde{\mathcal{P}}_{\text{conv}}^\circ : \text{Shape}_x = (k', k')\}$. Since an LRF-C is a $k^{(L)} \times k^{(L)}$ square window, every point $x \in \tilde{\mathcal{P}}_{\text{conv}}^\circ(k')$ is either a target-layer RPCP, a target-layer off-axis PSP, or a deeper-layer point.

We remove distractions using the following filters:

- Retain candidates satisfying $\text{Mask}_x = \mathbf{1}_{k' \times k'}$. This removes square off-axis PSPs, whose Masks are unions of two LRF-Cs.
- According to Proposition 4, for a deeper-layer point $x \in \tilde{\mathcal{P}}_{\text{conv}}^\circ(k')$, two deeper-layer points yield the same v_x if and only if every intermediate layer exhibits the same ReLU activation arrangement [16]; this happens only when the pooling operation between L and L' is not max pooling. Therefore, if there are two or more RPCPs, we can retain the correct target-layer RPCP set $\mathcal{P}_{\text{RPCP}}^\circ$.

Identifying Target-layer PSP. Define the remaining interior points as $\tilde{\mathcal{P}}_{\text{PSP}}^\circ = \tilde{\mathcal{P}}_{\text{conv}}^\circ \setminus \mathcal{P}_{\text{RPCP}}^\circ$, which contains target-layer PSPs and deeper-layer points. For $b \in \{\text{top}, \text{bottom}, \text{left}, \text{right}\}$, the four boundaries of $\text{Box}_x = [u_{\min}, u_{\max}] \times [v_{\min}, v_{\max}]$ are defined as

$$\begin{aligned} \partial_{\text{top}} \text{Box}_x &= \{u_{\min}\} \times [v_{\min}, v_{\max}], & \partial_{\text{bottom}} \text{Box}_x &= \{u_{\max}\} \times [v_{\min}, v_{\max}], \\ \partial_{\text{left}} \text{Box}_x &= [u_{\min}, u_{\max}] \times \{v_{\min}\}, & \partial_{\text{right}} \text{Box}_x &= [u_{\min}, u_{\max}] \times \{v_{\max}\}. \end{aligned}$$

Denote the effective positions of v_x on boundary b by

$$\mathcal{B}_{x,b} = \{(\psi^{(L)})^{-1}(u, v) \in (\psi^{(L)})^{-1}(\partial_b \text{Box}_x) : (\text{Mask}_x)_{u-u_{\min}, v-v_{\min}} = 1\}.$$

In the following, $K_{:,j}^{(L)}$ (resp. $K_{i,:}^{(L)}$) denotes the j -th column (resp. i -th row) of the kernel $K^{(L)}$. We extend Proposition 2 to Proposition 6 to filter deeper-layer points.

Proposition 6 (Boundary Equality with the Kernel). *For any target-layer PSP $x \in \tilde{\mathcal{P}}_{\text{PSP}}^\circ$, the following hold:*

- *horizontal PSP:* $v_x[\mathcal{B}_{x,\text{left}}] = \pm g_x K_{:,0}^{(L)}$ and $v_x[\mathcal{B}_{x,\text{right}}] = \pm g_x K_{:,k^{(L)}-1}^{(L)}$;
- *vertical PSP:* $v_x[\mathcal{B}_{x,\text{top}}] = \pm g_x K_{0,:}^{(L)}$ and $v_x[\mathcal{B}_{x,\text{bottom}}] = \pm g_x K_{k^{(L)}-1,:}^{(L)}$;
- *off-axis PSP:* $v_x[\mathcal{B}_{x,\text{top}}] = \pm g_x K_{0,:}^{(L)}$, $v_x[\mathcal{B}_{x,\text{left}}] = \pm g_x K_{:,0}^{(L)}$, $v_x[\mathcal{B}_{x,\text{bottom}}] = \pm g_x K_{k^{(L)}-1,:}^{(L)}$, and $v_x[\mathcal{B}_{x,\text{right}}] = \pm g_x K_{:,k^{(L)}-1}^{(L)}$.

Proof. It follows from the planar relationship between the two LRF-Cs of each PSP type.

Proposition 6 expresses the boundary entries of v_x in terms of the kernel, but each entry is scaled by the unknown factor $\pm g_x$, which is specific to x . Taking the ratio of two such entries eliminates g_x and leaves a quantity determined by the kernel alone. For any two PSPs $x_1, x_2 \in \tilde{\mathcal{P}}_{\text{PSP}}^\circ$, the ratios of the first to the last nonzero entries of v_{x_1} and v_{x_2} are therefore equal, and both equal the ratio of the top-left to the bottom-right entry of the kernel:

$$\frac{v_{x_1}[\min S_{x_1}]}{v_{x_1}[\max S_{x_1}]} = \frac{v_{x_2}[\min S_{x_2}]}{v_{x_2}[\max S_{x_2}]} = \frac{(K_0^{(L)})_0}{(K_{k^{(L)}-1}^{(L)})_{k^{(L)}-1}}. \quad (2)$$

The endpoints are only the simplest instance of this consistency. By Proposition 6, the whole non-overlapping part of a PSP, not just its two extremes, reproduces the kernel up to the factor $\pm g_x$, so the ratio of any two such entries is likewise x -independent; the overlap of the two LRF-Cs is excluded, since there v_x is a difference of shifted kernel copies. In practice the two forms are used progressively.

Since g_x cancels, the common ratio in Eq. (2) can be evaluated from a recovered vector alone, without prior knowledge of the kernel. We use it as an x -independent invariant to group PSPs sharing the same kernel and thereby identify the target-layer PSP set $\mathcal{P}_{\text{PSP}}^\circ$ in two cases:

- $\mathcal{P}_{\text{RPCP}}^\circ \neq \emptyset$. The kernel size $k^{(L)}$ and the kernel parameters are directly recovered from the RPCPs. The target-layer PSPs are then identified from both the spatial and the numerical aspects (Eqs. (3) and (2)).
- $\mathcal{P}_{\text{RPCP}}^\circ = \emptyset$. We apply the same numerical clustering as in the RPCP identification: cluster the candidates by Eq. (2).

After target-layer point identification, the retained interior RPCPs and PSPs form $\mathcal{P}_{\text{conv}}^\circ = \mathcal{P}_{\text{RPCP}}^\circ \cup \mathcal{P}_{\text{PSP}}^\circ$.

6.2 Method for Convolution Kernel Size and Stride Recovery

When $\mathcal{P}_{\text{RPCP}}^\circ \neq \emptyset$, the identified kernel size $\hat{k}$ is directly given by Lemma 3, and the target-layer PSP set $\mathcal{P}_{\text{PSP}}^\circ$ is filtered by Lemma 3 together with numerical consistency in non-overlapping areas. When $\mathcal{P}_{\text{RPCP}}^\circ = \emptyset$, we vote among the PSPs for the kernel size: the vote count for k' is $V(k') = \sum_{x \in \tilde{\mathcal{P}}_{\text{PSP}}^\circ} [b_\partial(\text{Mask}_x) = k']$. The identified kernel size is $\hat{k} = \arg \max_{k' \in \mathbb{N}} V(k')$, and the points supporting this result are retained as $\mathcal{P}_{\text{PSP}}^\circ$.

We collect the starting positions of the effective position sets to form $\mathcal{S}_{\text{conv}} = \{(r_x, c_x) : x \in \mathcal{P}_{\text{conv}}^\circ\}$. We then collect its nonzero row and column differences to form

$$\begin{aligned} \Delta_{\text{conv}} = & \{|r - r'| : (r, c), (r', c') \in \mathcal{S}_{\text{conv}}, r \neq r'\} \\ & \cup \{|c - c'| : (r, c), (r', c') \in \mathcal{S}_{\text{conv}}, c \neq c'\}. \end{aligned}$$

By Lemma 4, the convolution stride is recovered as $\hat{s}_k = \min \Delta_{\text{conv}}$.

7 Recovering the Padding Mode and the Output Channel Number

In this section, our tasks are to recover the padding mode (Task 4) and the output-channel number (Task 5), yielding $\hat{\pi} \in \{\text{Valid}, \text{Same}\}$ and $\hat{C}_{\text{out}}$.

Under Same padding, the input is padded with zeros, and part of the LRF-C of a point may lie in the zero-padding region. The kernel entries at the padded positions are not exposed in the true input coordinates, so the recovered vector is truncated on the corresponding side. We now state Lemma 5, which gives a sufficient condition for identifying Same padding.

Lemma 5 (Truncated Points Imply Same Padding). *A target-layer truncated point is a point whose LRF-C intersects the zero-padding region. If such a point exists, then $\pi^{(L)} = \text{Same}$.*

Proof. Under Valid padding, no zero-padding region exists, so no LRF-C intersects it.

Lemma 5 transforms Task 4 into identifying target-layer truncated points among the boundary candidates $\tilde{\mathcal{P}}_{\partial}$, while Task 5 is accomplished by clustering the retained target-layer points.

7.1 Identifying the Truncated Point

We define the boundary candidates as $\tilde{\mathcal{P}}_{\partial} = \mathcal{P}_{\text{conv}} \setminus \tilde{\mathcal{P}}_{\text{conv}}^{\circ}$. Every candidate is a target-layer RPCP or PSP whose Box touches the boundary, a target-layer truncated point, or a deeper-layer point.

Let $\tilde{\mathcal{P}}_{\partial, \text{left}} = \{x \in \tilde{\mathcal{P}}_{\partial} : \text{Box}_x \text{ touches the left boundary of } \Lambda^{(L)}\}$ denote the left-boundary candidates. Since the four boundaries are symmetric, we take the left boundary as an example to describe the method.

A target-layer truncated point may be obtained by truncating an RPCP or PSP at the input boundary. Based on Proposition 6, we derive Proposition 7.

Proposition 7 (Boundary Equality of Truncated Points). *For a target-layer truncated point $x \in \tilde{\mathcal{P}}_{\partial, \text{left}}$ obtained by truncating a target-layer RPCP, horizontal PSP, or off-axis PSP: $v_x[\mathcal{B}_{x, \text{right}}] = \pm g_x K_{:, k^{(L)}-1}^{(L)}$ and $v_x[\mathcal{B}_{x, \text{left}}] \neq \pm g_x K_{:, 0}^{(L)}$.*

Proof. By Lemma 1, v_x is, up to the sign absorbed in g_x , the restriction of the convolution-matrix row (resp. row difference) to its LRF-C. Left truncation removes the leftmost kernel columns, which fall into the zero-padding region, while the right boundary of Box_x lies strictly inside the input and its effective positions carry the complete last kernel column; hence $v_x[\mathcal{B}_{x, \text{right}}] = \pm g_x K_{:, k^{(L)}-1}^{(L)}$. The effective positions on the left boundary correspond to a truncated view rather than the first kernel column, so for a generic kernel they do not coincide with $\pm g_x K_{:, 0}^{(L)}$.

To filter the target-layer points, we select an RPCP from $\mathcal{P}_{\text{RPCP}}^\circ$ as the numerical template point; if $\mathcal{P}_{\text{RPCP}}^\circ = \emptyset$, we select a horizontal PSP or off-axis PSP instead. We adopt the two numerical verification conditions in Proposition 7, and denote the matched point set by $\mathcal{P}_{\partial, \text{left}}$. If $\mathcal{P}_{\partial, \text{left}} \neq \emptyset$, then a truncated point exists among the left-boundary candidates.

By symmetry, we obtain the matched sets $\mathcal{P}_{\partial, \text{right}}$, $\mathcal{P}_{\partial, \text{top}}$, and $\mathcal{P}_{\partial, \text{bottom}}$ for the other three boundaries. We denote the set of truncated points by $\mathcal{P}_\partial = \mathcal{P}_{\partial, \text{left}} \cup \mathcal{P}_{\partial, \text{right}} \cup \mathcal{P}_{\partial, \text{top}} \cup \mathcal{P}_{\partial, \text{bottom}}$.

7.2 Method for Recovering the Padding Mode and the Output Channel Number

According to Lemma 5, if $\mathcal{P}_\partial \neq \emptyset$, then a truncated point exists and we recover $\pi^{(L)} = \text{Same}$; otherwise, $\pi^{(L)} = \text{Valid}$.

We cluster the retained target-layer points into three groups. The points in $\mathcal{P}_{\text{RPCP}}^\circ$ are clustered by Proposition 2. The points in $\mathcal{P}_{\text{PSP}}^\circ$ are first classified into the three PSP types according to the recovered kernel size $\hat{k}$ and stride $\hat{s}_k$, and then clustered by Proposition 6. The truncated-point sets composing $\mathcal{P}_\partial$ are clustered separately: $\mathcal{P}_{\partial, \text{left}}$, $\mathcal{P}_{\partial, \text{right}}$, $\mathcal{P}_{\partial, \text{top}}$, and $\mathcal{P}_{\partial, \text{bottom}}$; e.g., $\mathcal{P}_{\partial, \text{left}}$ is clustered by the equality condition $v_x[\mathcal{B}_{x, \text{right}}] = \pm g_x K_{:, k^{(L)}-1}^{(L)}$ in Proposition 7. Finally, all resulting clusters are merged according to the kernel values exposed on their non-overlapping regions, and the number of merged classes is the recovered output-channel number $\hat{C}_{\text{out}}$.

8 Recovering the Pooling Window Size and Stride

In this section, we recover the pooling type $\hat{p} \in \{\text{NoPool}, \text{AvgPool}, \text{MaxPool}\}$. If the type is AvgPool or MaxPool, we further recover the window size $\hat{p}$ and the stride $\hat{s}_p$.

PSPs are produced by the additional nonlinear operation introduced by max pooling. Therefore, when $\mathcal{P}_{\text{PSP}}^\circ \neq \emptyset$, or when a PSP exists in the boundary set $\mathcal{P}_\partial$, we determine $\hat{p} = \text{MaxPool}$. Otherwise, the two remaining cases, NoPool and AvgPool, are both linear operations, so the pooling matrix formed under the fully-connected view is fixed: $P^{(L)} = I$ for NoPool, while the AvgPool matrix is a constant block matrix whose rows take the constant value $1/p^2$ on one $p \times p$ LRF-P and are zero elsewhere. Using this feature, we identify AvgPool and recover the window size $\hat{p}$ and the stride $\hat{s}_p$.

8.1 Recovering the Max-Pooling Window Size and Stride

Adjacent values within a pooling receptive field are computed by adjacent convolution operations, whose window starts are separated by $s_k^{(L)}$. For a PSP x , let $(r'_x, c'_x) := \psi^{(L)}(\max S_x) - (k^{(L)} - 1, k^{(L)} - 1)$ denote the start of its second LRF-C. We propose the following result.

Proposition 8 (PSP Window-Start Offsets Are Stride Multiples). *For any PSP $x \in \mathcal{P}_{\text{PSP}}^\circ$, the offsets between its two LRF-C starts satisfy:*

$$\begin{aligned} |r_x - r'_x| &= a s_k^{(L)}, \\ |c_x - c'_x| &= a s_k^{(L)}, \end{aligned} \tag{3}$$

where $a \in \{0, \dots, p-1\}$ in each spatial dimension.

Proof. The coordinates (r_x, c_x) are the start of one LRF-C of the PSP, and (r'_x, c'_x) are the start of the other LRF-C. The differences between the two LRF-Cs in each spatial dimension are integer multiples of the stride $s_k^{(L)}$. Moreover, the outputs of the two LRF-Cs lie within the same pooling window, so the offset between the two LRF-C starts is at most $(p-1)$ stride units in each dimension. Therefore, Eq. (3) holds.

For the PSPs $x \in \mathcal{P}_{\text{PSP}}^\circ$, evaluating the left-hand side of Eq. (3) for each PSP x forms the value set $\mathcal{D}$. By Proposition 8, we transform the max-pooling window-size identification into selecting the maximum value of $\mathcal{D}$ and computing $\hat{p} = (\max \mathcal{D}) / s_k^{(L)} + 1$.

We define the point sets whose offsets attain the maximum value of $\mathcal{D}$:

$$\begin{aligned} \mathcal{P}_{\max}^r &= \left\{ x \in \mathcal{P}_{\text{PSP}}^\circ : |r_x - r'_x| = (\hat{p} - 1) s_k^{(L)} \right\}, \\ \mathcal{P}_{\max}^c &= \left\{ x \in \mathcal{P}_{\text{PSP}}^\circ : |c_x - c'_x| = (\hat{p} - 1) s_k^{(L)} \right\}. \end{aligned}$$

We then propose Proposition 9 to characterize the relationship between the pooling-window anchors and the pooling stride.

Proposition 9 (Max-Pooling Anchors Lie on the Stride Grid). *For any two points x_1, x_2 of $\mathcal{P}_{\max}^r$, $r_{x_1} - r_{x_2} \in s_p s_k^{(L)} \mathbb{Z}$. Symmetrically, for any two points x_1, x_2 of $\mathcal{P}_{\max}^c$, $c_{x_1} - c_{x_2} \in s_p s_k^{(L)} \mathbb{Z}$.*

Proof. Pooling windows advance by s_p response positions. After mapping through the target convolution, their anchors therefore advance by $s_p s_k^{(L)}$ input coordinates.

We collect the coordinates of the retained maximum-offset points: $\mathcal{R}_{\max} = \{r_x : x \in \mathcal{P}_{\max}^r\}$ and $\mathcal{V}_{\max} = \{c_x : x \in \mathcal{P}_{\max}^c\}$. These coordinates are measured in the target-layer input space. We therefore compute their nonzero differences in units of the recovered convolution stride:

$$\Delta = \left\{ \frac{|r - r'|}{\hat{s}_k} : r, r' \in \mathcal{R}_{\max}, r \neq r' \right\} \cup \left\{ \frac{|c - c'|}{\hat{s}_k} : c, c' \in \mathcal{V}_{\max}, c \neq c' \right\}.$$

By Proposition 9, every element of Δ is a positive integer multiple of s_p . If the retained PSPs include two adjacent pooling-window anchors, we recover the max-pooling stride as $\hat{s}_p = \min \Delta$.

8.2 Recovering the Average-Pooling Window Size and Stride

After max pooling has been excluded, we recover the pooling type $\hat{\rho}$ and the pooling parameters. We adopt a skip-and-determine strategy: skip the pooling operation in layer L and recover the parameters through the recovered vectors in layer $L+1$. Let $v_x^{(L+1)}$ denote the recovered vector in the actual layer- $(L+1)$ input coordinates and let $\tilde{v}_x^{(L+1)}$ denote the vector recovered after skipping the pooling operation. For a layer- $(L+1)$ RPCP x , its recovered vector satisfies

$$v_x^{(L+1)} = \tilde{v}_x^{(L+1)} P^{(L)} = g_x A_{m,i}^{(L+1)} [\mathcal{K}_i^{(L+1)}],$$

and for a layer- $(L+1)$ PSP x ,

$$v_x^{(L+1)} = \tilde{v}_x^{(L+1)} P^{(L)} = g_x (A_{m,i}^{(L+1)} - A_{m,j}^{(L+1)}) [\mathcal{K}_i^{(L+1)}].$$

If $\rho^{(L)} = \text{NoPool}$, then $P^{(L)} = I$ and $\tilde{v}_x^{(L+1)} = v_x^{(L+1)}$. If $\rho^{(L)} = \text{AvgPool}$, we describe the structure of the restored vector in Proposition 10.

Suppose $\rho^{(L)} = \text{AvgPool}$ with window size p and stride s_p , and let $w_\rho^{(L)}$ denote the pooling output width. For a pooled coordinate $i \in \{0, \dots, d^{(L+1)} - 1\}$, the flattened LRF-P of i (the pooling analogue of $\mathcal{K}_i^{(L)}$) is

$$\mathcal{K}_i^P = \left\{ \left(\lfloor i/w_\rho^{(L)} \rfloor s_p + j \right) w_\rho^{(L)} + (i \bmod w_\rho^{(L)}) s_p + m : 0 \leq j < p, 0 \leq m < p \right\},$$

i.e., the set of pre-pooling positions averaged into the pooled coordinate i ; the i -th row of $P^{(L)}$ equals $\frac{1}{p^2}$ on $\mathcal{K}_i^P$ and 0 elsewhere. We present the single-channel case; channels pool independently.

Proposition 10 (Average-Pooling Structure of the Restored Vector).

Let x be a layer- $(L+1)$ point with recovered vector $v_x^{(L+1)} = \tilde{v}_x^{(L+1)} P^{(L)}$, and let $S_x^{(L+1)}$ denote the effective position set of $v_x^{(L+1)}$. Define the restored vector $\hat{v}_x^{(L+1)}$ by

$$(\hat{v}_x^{(L+1)})_r = p^2 \sum_{i': r \in \mathcal{K}_{i'}^P} (v_x^{(L+1)})_{i'} \quad \text{for every pre-pooling position } r. \quad (4)$$

Then the following hold.

1. $(\hat{v}_x^{(L+1)})_r = 0$ outside $\bigcup_{i' \in S_x^{(L+1)}} \mathcal{K}_{i'}^P$;
2. if $s_p \geq p$, then $\hat{v}_x^{(L+1)}$ is constant on each $\mathcal{K}_i^P$ with $i \in S_x^{(L+1)}$, with value $p^2 (v_x^{(L+1)})_i$, and consecutive segments are separated by $g = s_p - p$ zeros;
3. if $s_p < p$, then $\hat{v}_x^{(L+1)}$ equals $p^2 (v_x^{(L+1)})_i$ on the exclusive part (of length $\ell = s_p$) of each LRF-P, whereas each overlapped position carries the sum of the adjacent restored values.

Proof. Since the i -th row of $P^{(L)}$ equals $\frac{1}{p^2}$ exactly on $\mathcal{K}_i^P$ and zero elsewhere, the i -th coefficient of $v_x^{(L+1)} = \tilde{v}_x^{(L+1)} P^{(L)}$ is

$$(v_x^{(L+1)})_i = \tilde{v}_x^{(L+1)} (P_i^{(L)})^\top = \frac{1}{p^2} \sum_{r \in \mathcal{K}_i^P} (\tilde{v}_x^{(L+1)})_r.$$

By Eq. (1), $(v_x^{(L+1)})_i \neq 0$ holds only for $i \in S_x^{(L+1)}$, so every term in Eq. (4) vanishes unless $i' \in S_x^{(L+1)}$; this proves case 1. For case 2 and case 3, we count how many flattened LRF-Ps $\mathcal{K}_{i'}^P$ with $i' \in S_x^{(L+1)}$ cover each pre-pooling position:

- *Case $s_p \geq p$ (disjoint windows).* Each position $r \in \mathcal{K}_i^P$ with $i \in S_x^{(L+1)}$ lies in no other such LRF-P, so the sum in Eq. (4) reduces to the single term $p^2 (v_x^{(L+1)})_i$; consecutive segments are separated by $g = s_p - p$ zero positions. This proves (2).
- *Case $s_p < p$ (overlapping windows).* The exclusive part of each $\mathcal{K}_i^P$, of length $\ell = s_p$, is covered by no other such LRF-P and therefore carries the constant value $p^2 (v_x^{(L+1)})_i$, whereas each overlapped position receives the sum of the corresponding restored values. This proves (3).

Based on Proposition 10, we obtain Corollary 4, where ℓ , g , and o denote the constant-segment length, the zero-gap length, and the overlap length read from the entry sequence of $\hat{v}_x^{(L+1)}$.

Corollary 4 (Average-Pooling Parameter Estimate). *For a layer- $(L+1)$ point x exhibiting this pattern, the following hold:*

$$(p, s_p) = \begin{cases} (\ell, \ell), & s_p = p, \\ (\ell, \ell + g), & s_p > p, \\ (\ell + o, \ell), & s_p < p. \end{cases}$$

Proof. By Proposition 10, $\ell = p$ when $s_p \geq p$, with gap $g = s_p - p$, and $\ell = s_p$ with overlap $o = p - s_p$ when $s_p < p$; substituting these relations into the three cases gives the displayed pairs.

Therefore, no pooling and average pooling are distinguished by testing whether the entries of the restored vector $\hat{v}_x^{(L+1)}$ at adjacent positions are equal: under average pooling, the entry sequence of $\hat{v}_x^{(L+1)}$ exhibits repeated constant blocks, i.e., the entries within each flattened LRF-P are equal (Proposition 10); under no pooling, $P^{(L)} = I$ and $\hat{v}_x^{(L+1)} = v_x^{(L+1)}$ retains the sparse convolution structure, so no such repeated-block pattern appears. Once average pooling is identified, reading the segment length ℓ , the zero-gap length g , and the overlap length o from the entry sequence and substituting them into Corollary 4 completes the parameter recovery of average pooling.

9 Experiments

This section evaluates the effectiveness of the proposed architecture recovery attack and is organized as follows. Section 9.1 summarizes the overall experimental results, and Section 9.2 presents visualizations of the architecture recovery process, task by task, on one representative model. All the experiments are conducted on a server equipped with two Intel Xeon Gold 5318Y CPUs at 2.10 GHz, 503 GB of memory.

9.1 Summary of Experimental Results

We combine our architecture recovery attack with the existing parameter recovery attacks for CNNs, using the attack of [9] in the raw-output setting, the attack of [10] in the hard-label setting, and both the layer-by-layer and the end-to-end pipeline. The tested models are a $(2 + 1)$ CNN and a $(2 + 2)$ CNN, the latter a LeNet-5 variant, both adopted by these attacks [9, 10]. All models use two convolutional layers with 5×5 kernels, stride 1, Valid padding and 2×2 max pooling, and they differ only in the output-channel numbers.

For every model under each tested setting, the architecture of each layer is recovered exactly, and the subsequent parameter recovery achieves the same accuracy as the original attacks.

A further concern is the complexity introduced by architecture recovery. Its only cost is the time to analyze the recovered vectors that the point search and equation solving have already produced; since the two phases share this step, the queries consumed by architecture recovery are a subset of those consumed by parameter recovery. For the $(2 + 1)$ CNN in the raw-output setting, the complete parameter recovery requires $2^{20.48}$ queries and $2^{7.31}$ seconds, whereas the architecture-recovery phase accounts for $2^{17.71}$ queries and $2^{4.08}$ seconds.

These experimental results fully prove that our attack works well in recovering the whole network architecture without any prior architectural knowledge, and it is feedback-agnostic. The complete results of all tested models are available in our repository.

9.2 Visualizing the Architecture Recovery Process

To demonstrate that our method handles diverse architectural hyperparameters, we design a CNN model as the target. Its convolutional layers deliberately employ different kernel sizes, strides, padding modes, output-channel numbers, and pooling operations. The complete architecture is shown in Table 2.

This model was trained on the MNIST training set resized to 32×32 , achieving a training accuracy of 94.03% and a test accuracy of 89.30%.

We recover the architecture of this model layer by layer in the raw-output setting. The point collection and the equation solving are taken from the attack of [9], and our architecture recovery is inserted between them. For uniformity, we fix the number of critical points searched for each layer at 700. For each task we visualize the statistics collected during the recovery, showing how these statistics determine the corresponding architecture hyperparameter.

Table 2. Ground-truth architecture of the $(3 + 1)$ CNN.

Layer	Type	Output shape	k	s_k	π	C_{out}	$\rho(p, s_p)$
1	Conv	$1 \times 32 \times 32$	3	1	Same	1	NoPool
2	Conv	$2 \times 16 \times 16$	5	1	Same	2	AvgPool (2, 2)
3	Conv	$3 \times 3 \times 3$	3	2	Valid	3	MaxPool (3, 2)
4	FC	18	—	—	—	—	—
5	Output	10	—	—	—	—	—

Task 1: Layer-Type Identification. Figure 4 reports $|\mathcal{P}_{\text{full}}|$ and $|\mathcal{P}_{\text{conv}}|$ for three layers. Since $\mathcal{P}_{\text{conv}} \neq \emptyset$ for every layer, all three layers are identified as convolutional by Lemma 2.

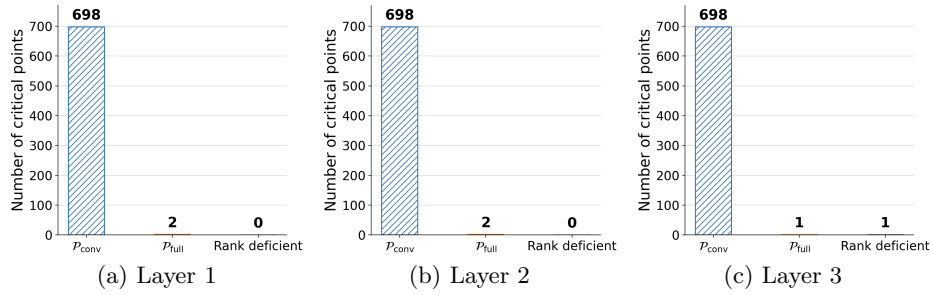


Fig. 4. Task 1: numbers of points in $\mathcal{P}_{\text{conv}}$ and $\mathcal{P}_{\text{full}}$ for each convolutional target layer.

Task 2: Kernel-Size Identification. Figure 5 reports the votes of the points that survive the boundary and RPCP filtering. Since the square Shape_x of every layer concentrates on a single candidate, the kernel sizes are recovered as $\hat{k} = 3, 5$, and 3 for layers 1–3 by Lemma 3, matching the ground truth.

Task 3: Stride Identification. Figure 6 shows the starting positions of all points in $\tilde{\mathcal{P}}_{\text{conv}}^{\circ}$. Since the minimum difference between the starting positions is 1, 1, and 2 for layers 1–3, the strides are recovered by Lemma 4, matching the ground truth.

Task 4: Padding-Mode Identification. Figure 7 shows the truncated points identified on the four boundaries of each layer. Since truncated points appear on all four boundaries in layers 1 and 2 and on none of them in layer 3, the padding modes are recovered as Same, Same, and Valid for layers 1–3 by Lemma 5, matching the ground truth.

Task 5: Output-Channel Identification. Figure 8 shows how the RPCP and PSP records of each layer are grouped into clusters. Since the records of each

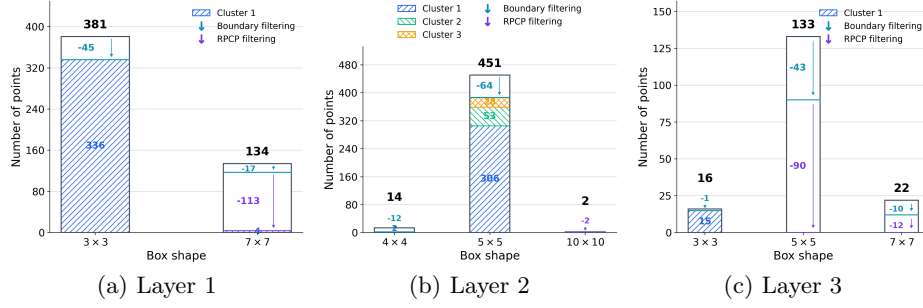


Fig. 5. Task 2: votes of the points that survive the boundary and RPCP filtering, grouped by candidate kernel size.

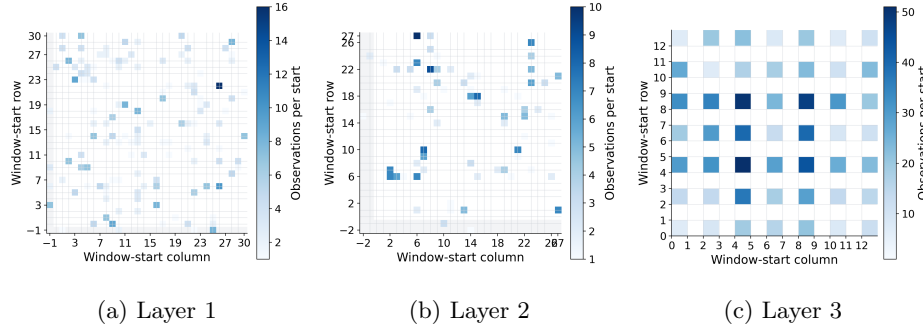


Fig. 6. Task 3: window-start distributions of the points in $\tilde{\mathcal{P}}_{\text{conv}}^o$.

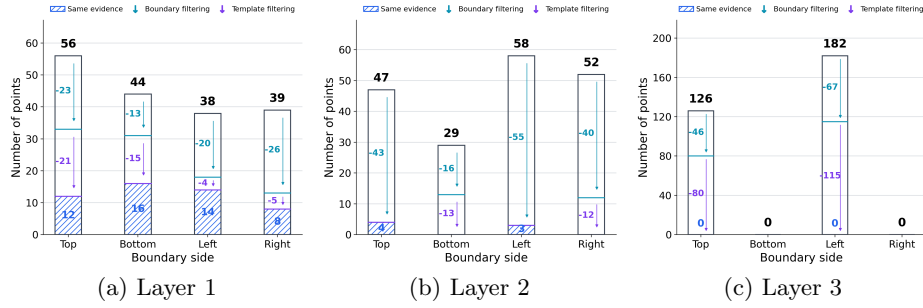


Fig. 7. Task 4: the truncated points on the four boundaries of each layer. Cyan arrows mark the points removed by the boundary filtering, and purple arrows mark the points that the template filtering rejects.

layer form 1, 2, and 3 clusters, the numbers of output channels are recovered, matching the ground truth.

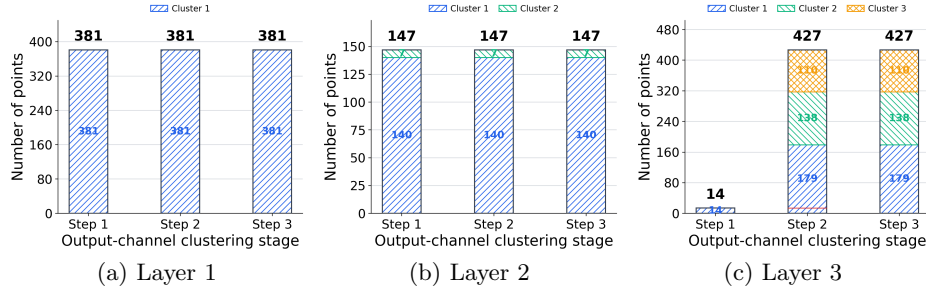


Fig. 8. Task 5: clustering of the RPCP and PSP records. Step 1 clusters the stable RPCP ratio records, Step 2 adds the PSPs whose numerical ratios match the cluster centers, and Step 3 adds a new cluster for the remaining PSPs.

Task 6: Pooling-Structure Identification. Figure 9 shows the $\tilde{v}_x^{(L+1)}$ of layers 1 and 2 and the max-pooling parameter evidence of layer 3. Layer 1 shows no repeated blocks, so $\hat{\rho}^{(1)} = \text{NoPool}$. Layer 2 shows repeated constant blocks, so $\hat{\rho}^{(2)} = \text{AvgPool}$. The pooling parameters are identified by Corollary 4. For layer 3, PSPs identify $\hat{\rho}^{(3)} = \text{MaxPool}$, and the anchor positions $\mathcal{R}_{\max}$ and $\mathcal{V}_{\max}$ give the starting points, from which the pooling window size and the stride follow, by Proposition 8 and Proposition 9. The pooling structures are recovered as NoPool, AvgPool (2, 2), and MaxPool (3, 2) for layers 1–3, matching the ground truth.

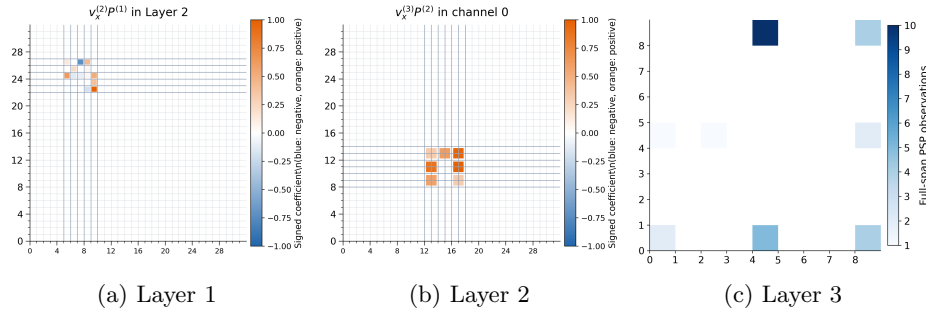


Fig. 9. Task 6: Panel (a) shows a representative restored vector of layer 1 (up to the factor p^2), panel (b) the restored vector of layer 2 (channel 0), and panel (c) the max-pooling parameter evidence of layer 3.

10 Conclusions

In this paper, we initiated the study of architecture recovery for convolutional neural networks. Focusing on the convolutional layers, we proposed an attack that recovers the architecture of each layer before its parameters and, combined

with existing methods for the fully connected layers, completes a cryptanalytic extraction framework for CNNs. Our approach is built on a direct observation: the recovered vectors reveal the network architecture in their spatial structure. We identified three such signals, the sparsity consistency with the convolution receptive field, the numerical consistency with the kernel parameters, and the structural consistency with the pooling operation, and used them to recover the layer type, the kernel size and stride, the padding mode, the output-channel number, and the pooling structure. Experiments on a range of CNNs, including a purpose-built model with diverse architectural hyperparameters and the models adopted by existing attacks, verified the effectiveness of our attack.

Additionally, our attack complements the established architecture-recovery results for fully connected networks: existing methods recover the layer widths of a fully connected network along with its parameters, whereas we show that the architecture of the convolutional layers of a CNN is directly revealed by the spatial geometry of the recovered weight vectors. The two routes cover the fully connected and the convolutional layers, respectively, and together they remove the architectural prior from the cryptanalytic extraction of CNNs.

References

1. Asselineau, R., Derbez, P., Fouque, P.A., Minaud, B.: Cryptanalytic extraction of deep neural networks with non-linear activations. In: Heninger, N., Rosulek, M. (eds.) *Advances in Cryptology - CRYPTO 2026 - 46th Annual International Cryptology Conference*, Santa Barbara, CA, USA, August 17-20, 2026, Proceedings, Part VII. *Lecture Notes in Computer Science*, vol. 16806, pp. 36–66. Springer (2026). https://doi.org/10.1007/978-3-032-35415-0_2, https://doi.org/10.1007/978-3-032-35415-0_2
2. Baum, E.B.: Neural net algorithms that learn in polynomial time from examples and queries. *IEEE Trans. Neural Networks* **2**(1), 5–19 (1991). <https://doi.org/10.1109/72.80287>, <https://doi.org/10.1109/72.80287>
3. Canales-Martínez, I.A., Santos, D.: Extracting some layers of deep neural networks in the hard-label setting. In: Escudero, D., Damgård, I. (eds.) *Progress in Cryptology - LATINCRYPT 2025 - 9th International Conference on Cryptology and Information Security in Latin America*, Medellín, Colombia, October 1-3, 2025, Proceedings. *Lecture Notes in Computer Science*, vol. 16129, pp. 399–421. Springer (2025). https://doi.org/10.1007/978-3-032-06754-8_15, https://doi.org/10.1007/978-3-032-06754-8_15
4. Carlini, N., Chávez-Saab, J., Hambitzer, A., Rodríguez-Henríquez, F., Shamir, A.: Polynomial time cryptanalytic extraction of deep neural networks in the hard-label setting (extended version). *Cryptology ePrint Archive*, Paper 2024/1580 (2024), <https://eprint.iacr.org/2024/1580>
5. Carlini, N., Chávez-Saab, J., Hambitzer, A., Rodríguez-Henríquez, F., Shamir, A.: Polynomial time cryptanalytic extraction of deep neural networks in the hard-label setting. In: Fehr, S., Fouque, P. (eds.) *Advances in Cryptology - EUROCRYPT 2025 - 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Madrid, Spain, May 4-8, 2025, Proceedings, Part I. *Lecture Notes in Computer Science*, vol. 15601, pp. 364–396. Springer (2025). https://doi.org/10.1007/978-3-031-91107-1_13, https://doi.org/10.1007/978-3-031-91107-1_13
6. Carlini, N., Jagielski, M., Mironov, I.: Cryptanalytic extraction of neural network models. In: Micciancio, D., Ristenpart, T. (eds.) *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference*, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III. *Lecture Notes in Computer Science*, vol. 12172, pp. 189–218. Springer (2020). https://doi.org/10.1007/978-3-030-56877-1_7, https://doi.org/10.1007/978-3-030-56877-1_7
7. Chen, Y., Dong, X., Guo, J., Shen, Y., Wang, A., Wang, X.: Hard-label cryptanalytic extraction of neural network models. In: Chung, K., Sasaki, Y. (eds.) *Advances in Cryptology - ASIACRYPT 2024 - 30th International Conference on the Theory and Application of Cryptology and Information Security*, Kolkata, India, December 9-13, 2024, Proceedings, Part VIII. *Lecture Notes in Computer Science*, vol. 15491, pp. 207–236. Springer (2024). https://doi.org/10.1007/978-981-96-0944-4_7, https://doi.org/10.1007/978-981-96-0944-4_7
8. Chen, Y., Dong, X., Ma, R., Shen, Y., Wang, A., Yu, H., Wang, X.: Delving into cryptanalytic extraction of prelu neural networks. In: Hanaoka, G., Yang, B. (eds.) *Advances in Cryptology - ASIACRYPT 2025 - 31st International Conference on the Theory and Application of Cryptology and Information Security*, Melbourne, VIC, Australia, December 8-12, 2025, Proceedings, Part II. *Lecture Notes in Computer*

- Science, vol. 16246, pp. 576–607. Springer (2025). https://doi.org/10.1007/978-981-95-5096-8_18, https://doi.org/10.1007/978-981-95-5096-8_18
9. Chen, Z., Tang, S., Gao, Z., Su, Y., Qin, L., Dong, X.: Algebraic attack on convolutional neural networks with max pooling. In: Heninger, N., Rosulek, M. (eds.) *Advances in Cryptology - CRYPTO 2026 - 46th Annual International Cryptology Conference*, Santa Barbara, CA, USA, August 17-20, 2026, Proceedings, Part VII. *Lecture Notes in Computer Science*, vol. 16806, pp. 3–35. Springer (2026). https://doi.org/10.1007/978-3-032-35415-0_1, https://doi.org/10.1007/978-3-032-35415-0_1
 10. Chen, Z., Tang, S., Gao, Z., Su, Y., Qin, L., Dong, X.: Algebraic cryptanalytic extraction on hard-label neural networks. *Cryptology ePrint Archive*, Paper 2026/1164 (2026), <https://eprint.iacr.org/2026/1164>
 11. Daniely, A., Granot, E.: An exact poly-time membership-queries algorithm for extracting a three-layer relu network. In: *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net (2023), <https://openreview.net/forum?id=-CoNloheTs>
 12. Foerster, H., Mullins, R.D., Shumailov, I., Hayes, J.: Beyond slow signs in high-fidelity model extraction. In: Globerson, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J.M., Zhang, C. (eds.) *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024* (2024), http://papers.nips.cc/paper_files/paper/2024/hash/22ae669a35bb9e70eb93ab77c1eff5b4-Abstract-Conference.html
 13. Ito, A., Miura, T., Todo, Y.: Is the hard-label cryptanalytic model extraction really polynomial? In: Heninger, N., Rosulek, M. (eds.) *Advances in Cryptology - CRYPTO 2026 - 46th Annual International Cryptology Conference*, Santa Barbara, CA, USA, August 17-20, 2026, Proceedings, Part VII. *Lecture Notes in Computer Science*, vol. 16806, pp. 67–98. Springer (2026). https://doi.org/10.1007/978-3-032-35415-0_3, https://doi.org/10.1007/978-3-032-35415-0_3
 14. Jagielski, M., Carlini, N., Berthelot, D., Kurakin, A., Papernot, N.: High accuracy and high fidelity extraction of neural networks. In: Capkun, S., Roesner, F. (eds.) *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*. pp. 1345–1362. USENIX Association (2020), <https://www.usenix.org/conference/usenixsecurity20/presentation/jagielski>
 15. Liu, H., Siproudhis, A., Boura, C., Peyrin, T.: Model extraction of convolutional neural networks with max-pooling. *IACR Transactions on Symmetric Cryptology (ToSC)* (2026), <https://eprint.iacr.org/2026/464>
 16. Liu, H., Siproudhis, A., Experton, S., Lorenz, P., Boura, C., Peyrin, T.: Navigating the deep: End-to-end extraction on deep neural networks. In: Daemen, J., Thomé, E. (eds.) *Advances in Cryptology - EUROCRYPT 2026*. *Lecture Notes in Computer Science*, vol. 16546. Springer (2026). https://doi.org/10.1007/978-3-032-25333-0_17, https://doi.org/10.1007/978-3-032-25333-0_17
 17. Martinez, I.A.C., Chávez-Saab, J., Hambitzer, A., Rodríguez-Henríquez, F., Satpute, N., Shamir, A.: Polynomial time cryptanalytic extraction of neural network models. In: Joye, M., Leander, G. (eds.) *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part III. *Lecture Notes in Computer Science*, vol. 14653, pp. 3–33. Springer (2024). https://doi.org/10.1007/978-3-031-58734-4_1, https://doi.org/10.1007/978-3-031-58734-4_1

18. Qi, X., Lei, H., Wei, L., Sun, X., Wang, M.: Cryptanalytic extraction of neural networks with various activation functions. *IACR Transactions on Symmetric Cryptology (ToSC)* **2026**(1), 468–505 (2026). <https://doi.org/10.46586/tosc.v2026.i1.468-505>, <https://doi.org/10.46586/tosc.v2026.i1.468-505>
19. Rolnick, D., Kording, K.P.: Reverse-engineering deep relu networks. In: Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13–18 July 2020, Virtual Event. *Proceedings of Machine Learning Research*, vol. 119, pp. 8178–8187. PMLR (2020), <http://proceedings.mlr.press/v119/rolnick20a.html>
20. Shen, Y., Chen, Y., Wang, A., Yu, H., Wang, X.: Cryptanalytic extraction of neural networks without known architecture assumption. *arXiv:2609.14379 [cs.CR]* (2026). <https://doi.org/10.48550/arXiv.2609.14379>, <https://arxiv.org/abs/2609.14379>
21. Sun, X., Lei, H., Wei, L., Qi, X., Hu, K., Wang, M., Wang, W.: Cryptanalytic extraction of convolutional neural networks. *Cryptology ePrint Archive*, Paper 2026/139 (2026), <https://eprint.iacr.org/2026/139>
22. Tramèr, F., Zhang, F., Juels, A., Reiter, M.K., Ristenpart, T.: Stealing machine learning models via prediction apis. In: Holz, T., Savage, S. (eds.) 25th USENIX Security Symposium, *USENIX Security 16*, Austin, TX, USA, August 10–12, 2016. pp. 601–618. USENIX Association (2016), <https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/tramer>
23. Xu, D., Zhang, L., Wang, Z.: Finite-precision error analysis of cryptanalytic model extraction. *Cryptology ePrint Archive*, Paper 2026/1943 (2026), <https://eprint.iacr.org/2026/1943>