Supplementary material for *Data-Free Universal Attack by Exploiting the Intrinsic Vulnerability of Deep models*

August 20, 2024

Abstract

Here, we provide a supplementary material for "Data-Free Universal Attack by Exploiting the Intrinsic Vulnerability of Deep models". We will provide a technical appendix containing additional details to support the arguments presented in the main paper. This appendix will include proofs, detailed methods for converting convolutional layers and batch normalization layers into matrices, and experimental details.

A Notation Table

We summarize notations used in this paper in Table 8. Note that in this paper, we denote the Banach space $\mathcal{B}$ as $(\mathcal{B}, \|\cdot\|_{\mathcal{B}})$, and we only consider Banach spaces with finite or countable basis.

Notations	Meaning
$\mathcal{B}$	Banach space.
$\|\cdot\|_{\mathcal{B}}$	The norm of Banach space $\mathcal{B}$.
$\|\cdot\|_{op}$	The operator norm.
$\mathbb{T}$	The tensor space.
$\mathbb{R}$	The Euclidean space.
$\mathrm{Lip}(T)$	The Lipschitz constant of operator T.
$\sup$	The upper bound of a set.
$\phi_{(c,m,m)}$	An isomorphic mapping $\phi_{(c,m,m)} : \mathbb{T}_{(c,m,m)} \to \mathbb{R}^{c\times m\times m}$.
$\psi_{i,j}^{(k,m)}$	The embedding mapping $\psi_{i,j}^{(k,m)} : \mathbb{T}_{(c,k,k)} \to \mathbb{T}_{(c,m,m)}$.
Γ	The padding operator.
$\mathrm{svd}(A)$	Singular value decomposition of operator A

Table 8: Main notations used in this paper.

B Proofs

Appendix B corresponds to "**Section 5: Theoretical Justification**" in the main paper. In this section, we prove Theorem 1, and provide the following definitions:

Definition 1. (Operator norm) *Let $\mathcal{B}_1$ and $\mathcal{B}_2$ are two Banach spaces. The operator norm of $\mathcal{W} : \mathcal{B}_1 \to \mathcal{B}_2$ is:*

$$\|\mathcal{W}\|_{op} = \sup \{\|\mathcal{W}\boldsymbol{v}\|_{\mathcal{B}_2} : \|\boldsymbol{v}\|_{\mathcal{B}_1} = 1\}. \tag{10}$$

Definition 2. (Lipschitz constant) *Let $\mathcal{B}_1$ and $\mathcal{B}_2$ are two Banach spaces. A operator $T : \mathcal{B}_1 \to \mathcal{B}_2$ is called Lipschitz continuous if there exists a constant L such that:*

$$\begin{aligned} &\forall \boldsymbol{v}, \boldsymbol{w} \in \mathcal{B}_1, \|T(\boldsymbol{v}) - T(\boldsymbol{w})\|_{\mathcal{B}_2} \leq L \|\boldsymbol{v} - \boldsymbol{w}\|_{\mathcal{B}_1}, \\ &\mathrm{Lip}(T) = L = \sup_{\boldsymbol{v} \neq \boldsymbol{w} \in \mathcal{B}_1} \frac{\|T(\boldsymbol{v}) - T(\boldsymbol{w})\|_{\mathcal{B}_2}}{\|\boldsymbol{v} - \boldsymbol{w}\|_{\mathcal{B}_1}}. \end{aligned} \tag{11}$$

B.1 Proof of Theorem 1

We present the following lemma to prove Theorem 1. We begin by examining the boundedness and continuity of linear operators, demonstrating that the Lipschitz constant of a bounded linear operator is equivalent to its operator norm. Then, based on Lemma 1, we draw the final conclusion.

Lemma 1. (Federer,1969) *Let g and h be two composable Lipschitz functions. Then $g \circ h$ is also Lipschitz with* $\mathrm{Lip}(g \circ h) \leq \mathrm{Lip}(g) \cdot \mathrm{Lip}(h)$.

Proof. First, let's define the continuity and boundedness of a linear operator. A linear operator $T : \mathcal{B}_1 \to \mathcal{B}_2$, and a vector (or a function in function space) $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{B}_1$.

If T is **continuous**, we have:

$$\forall \varepsilon > 0, \quad \exists \delta > 0, \quad \|\boldsymbol{x} - \boldsymbol{x}'\|_{\mathcal{B}_1} < \delta \quad \text{s.t.} \quad \|T\boldsymbol{x} - T\boldsymbol{x}'\|_{\mathcal{B}_2} < \varepsilon. \tag{12}$$

And if $\exists M > 0$ such that

$$\|T\boldsymbol{x}\|_{\mathcal{B}_2} \leq M \|\boldsymbol{x}\|_{\mathcal{B}_1}, \tag{13}$$

we say that T is **bounded**. In fact, the conditions of being bounded or continuous are equivalent. We define $\|T\|_{op}$ as follows:

$$\|T\|_{op} = \sup_{\|\boldsymbol{x}\|_{\mathcal{B}_1} \leq 1} \|T\boldsymbol{x}\|_{\mathcal{B}_2}. \tag{14}$$

Bounded $\Rightarrow$ Continuous: We assume that T is bounded, so

$$\forall \boldsymbol{x}, \boldsymbol{x}' \in \mathcal{B}_1, \tag{15}$$

we have

$$\|T(\boldsymbol{x} - \boldsymbol{x}')\|_{\mathcal{B}_2} = \|T\boldsymbol{x} - T\boldsymbol{x}'\|_{\mathcal{B}_2} \leq M \|\boldsymbol{x} - \boldsymbol{x}'\|_{\mathcal{B}_1}. \tag{16}$$

Therefore, according to the supremum and infimum principle,

$$\frac{\|T(\boldsymbol{x} - \boldsymbol{x}')\|_{\mathcal{B}_2}}{\|\boldsymbol{x} - \boldsymbol{x}'\|_{\mathcal{B}_1}} \tag{17}$$

has supremum M', so

$$\forall \varepsilon > 0, \quad \exists \delta = \frac{\varepsilon}{M'} \quad \text{s.t.} \quad \|\boldsymbol{x} - \boldsymbol{x}'\|_{\mathcal{B}_1} \leq \delta, \tag{18}$$

so,

$$\|T\boldsymbol{x} - T\boldsymbol{x}'\|_{\mathcal{B}_2} < \varepsilon. \tag{19}$$

We get that T is continuous.

Bounded $\Leftarrow$ Continuous: We assume that T is continuous, so we have:

$$\forall \varepsilon > 0, \quad \exists \delta > 0 \quad \text{such that} \qquad \|\boldsymbol{x} - \boldsymbol{x}'\|_{\mathcal{B}_1} < \delta \implies \|T\boldsymbol{x} - T\boldsymbol{x}'\|_{\mathcal{B}_2} < \varepsilon. \tag{20}$$

To show that T is bounded, we need to find a constant $M > 0$ such that:

$$\|T(\boldsymbol{x} - \boldsymbol{x}')\|_{\mathcal{B}_2} \leq M \|\boldsymbol{x} - \boldsymbol{x}'\|_{\mathcal{B}_1} \quad \forall (\boldsymbol{x} - \boldsymbol{x}') \in \mathcal{B}_1. \tag{21}$$

Since

$$\left\|\frac{\delta}{2\left\|(\boldsymbol{x}-\boldsymbol{x}')\right\|_{\mathcal{B}_1}}(\boldsymbol{x}-\boldsymbol{x}')\right\|_{\mathcal{B}_1} = \frac{\delta}{2\left\|(\boldsymbol{x}-\boldsymbol{x}')\right\|_{\mathcal{B}_1}}\left\|(\boldsymbol{x}-\boldsymbol{x}')\right\|_{\mathcal{B}_1} = \frac{\delta}{2} < \delta. \tag{22}$$

Therefore,

$$\left\|T(\frac{\delta}{2\left\|(\boldsymbol{x}-\boldsymbol{x}')\right\|_{\mathcal{B}_1}}(\boldsymbol{x}-\boldsymbol{x}'))\right\|_{\mathcal{B}_2} = \frac{\delta}{2\left\|\boldsymbol{x}-\boldsymbol{x}'\right\|_{\mathcal{B}_1}}\left\|T(\boldsymbol{x}-\boldsymbol{x}')\right\|_{\mathcal{B}_2} < \varepsilon, \tag{23}$$

which implies

$$\left\|T(\boldsymbol{x}-\boldsymbol{x}')\right\|_{\mathcal{B}_2} < \frac{2\varepsilon}{\delta}\left\|\boldsymbol{x}-\boldsymbol{x}'\right\|_{\mathcal{B}_1}. \tag{24}$$

Let $M = \frac{2\varepsilon}{\delta}$. Then we have:

$$\left\|T(\boldsymbol{x}-\boldsymbol{x}')\right\|_{\mathcal{B}_2} \leq M\left\|\boldsymbol{x}-\boldsymbol{x}'\right\|_{\mathcal{B}_1} \quad \forall \boldsymbol{x} \in \mathcal{B}_1. \tag{25}$$

Therefore, T is bounded.

Since T is linear, the Lipschitz constant equals

$$\mathrm{Lip}(T) = \sup_{\boldsymbol{x}\neq 0}\frac{\|T(\boldsymbol{x})\|_{\mathcal{B}_2}}{\|\boldsymbol{x}\|_{\mathcal{B}_1}} = \|T\|_{op}. \tag{26}$$

Therefore, we have shown that a linear operator T is bounded if and only if it is continuous, and the operator norm equals the Lipschitz constant, i.e.,

$$\|T\|_{op} = \mathrm{Lip}(T). \tag{27}$$

Then, back to Theorem 1, the L1LOS f consists of a linear operator and a nonlinear operator with a Lipschitz constant of 1. Therefore, using Lemma 1, we have:

$$\mathrm{Lip}(f) \leq \prod_{k=0}^{\ell-1}\|W_k\|_{op}. \tag{28}$$

□

C Convolutional Layer as a Linear Operator

Appendix C corresponds to ”**Section 4.2: Convolutional Layer as a Linear Operator**” in the main paper. In this section, we will succinctly describe how to represent convolutional layers as linear operators. Based on praggastis's framework (Praggastis et al., 2022) for convolution interpretability, we extend the method of representing convolutional layers as multilevel-block Toeplitz matrices, to accommodate convolutional layers with any padding and stride length, making it widely applicable in CNNs.

We assume that X is a 3-tensor of dimensions $c \times m \times m$, W is a 4-tensor of dimensions $d \times c \times k \times k$ where $k \leq m$. $\mathbb{T}_{(d,c,k,k)}$ and $\mathbb{T}_{(c,m,m)}$ are two tensor spaces, we say $W \in \mathbb{T}_{(d,c,k,k)}$ with the stride ω and the padding p has d filters , c channels and spatial dimensions $k \times k$, $X \in \mathbb{T}_{(c,m,m)}$ has c channels and spatial dimensions $m \times m$. In this setting, $Y \in \mathbb{T}_{(d,n,n)}$, for $n = ((m-k+2p)//\omega)+1$, has d channels and spatial dimensions $n \times n$. The convolution operation can be written as $Y = W \star X$.

C.1 A Simple Example

First, we will provide an example of representing a convolutional layer for 4-dimensional tensors as a multilevel-block Toeplitz matrix. Suppose we define a convolutional layer $W \in \mathbb{T}_{(2,2,3,3)}$. We decompose the convolutional layer into 4 parts along the spatial dimensions, denoted as $k_1, k_2, k_3, k_4 \in \mathbb{T}_{(3,3)}$, as follows:

$$\mathbf{k}_0 = \begin{pmatrix} a_0 & a_1 & a_2 \\ a_3 & a_4 & a_5 \\ a_6 & a_7 & a_8 \end{pmatrix}, \mathbf{k}_1 = \begin{pmatrix} b_0 & b_1 & b_2 \\ b_3 & b_4 & b_5 \\ b_6 & b_7 & b_8 \end{pmatrix}, \mathbf{k}_2 = \begin{pmatrix} c_0 & c_1 & c_2 \\ c_3 & c_4 & c_5 \\ c_6 & c_7 & c_8 \end{pmatrix}, \mathbf{k}_3 = \begin{pmatrix} d_0 & d_1 & d_2 \\ d_3 & d_4 & d_5 \\ d_6 & d_7 & d_8 \end{pmatrix} \tag{29}$$

If we set the padding to 1, the matrix $\bar{W}$ is a multilevel-block Toeplitz matrix of size $2m^2 \times 2m^2$ and has following form:

$$\bar{W} = \begin{pmatrix}
\mathbf{T}_0 & \mathbf{T}_1 & & & 0 & \mathbf{U}_0 & \mathbf{U}_1 & & & 0 \\
\mathbf{T}_2 & \mathbf{T}_0 & \mathbf{T}_1 & & & \mathbf{U}_2 & \mathbf{U}_0 & \mathbf{U}_1 & & \\
& \mathbf{T}_2 & \ddots & \ddots & & & \mathbf{U}_2 & \ddots & \ddots & \\
& & \ddots & \mathbf{T}_0 & \mathbf{T}_1 & & & \ddots & \mathbf{U}_0 & \mathbf{U}_1 \\
0 & & & \mathbf{T}_2 & \mathbf{T}_0 & 0 & & & \mathbf{U}_2 & \mathbf{U}_0 \\
\mathbf{V}_0 & \mathbf{V}_1 & & & 0 & \mathbf{O}_0 & \mathbf{O}_1 & & & 0 \\
\mathbf{V}_2 & \mathbf{V}_0 & \mathbf{V}_1 & & & \mathbf{O}_2 & \mathbf{O}_0 & \mathbf{O}_1 & & \\
& \mathbf{V}_2 & \ddots & \ddots & & & \mathbf{O}_2 & \ddots & \ddots & \\
& & \ddots & \mathbf{V}_0 & \mathbf{V}_1 & & & \ddots & \mathbf{O}_0 & \mathbf{O}_1 \\
0 & & & \mathbf{V}_2 & \mathbf{V}_0 & 0 & & & \mathbf{O}_2 & \mathbf{O}_0
\end{pmatrix} \tag{30}$$

Where $\mathbf{T}_j, \mathbf{U}_j, \mathbf{V}_j, \mathbf{O}_j$ are banded Toeplitz matrices. For example $\mathbf{T}_j$ and the values of $\mathbf{k}_0$ are distributed in the Toeplitz blocks as follow, for $j = 0, 1, 2$:

$$\mathbf{T}_0 = \begin{pmatrix}
a_4 & a_3 & & & 0 \\
a_5 & a_4 & a_3 & & \\
& a_5 & \ddots & \ddots & \\
& & \ddots & a_4 & a_3 \\
0 & & & a_5 & a_4
\end{pmatrix} \quad \mathbf{T}_1 = \begin{pmatrix}
a_7 & a_6 & & & 0 \\
a_8 & a_7 & a_6 & & \\
& a_8 & \ddots & \ddots & \\
& & \ddots & a_7 & a_6 \\
0 & & & a_8 & a_7
\end{pmatrix} \quad \mathbf{T}_2 = \begin{pmatrix}
a_1 & a_0 & & & 0 \\
a_2 & a_1 & a_0 & & \\
& a_2 & \ddots & \ddots & \\
& & \ddots & a_1 & a_0 \\
0 & & & a_2 & a_1
\end{pmatrix}. \tag{31}$$

C.2 Specific Method

Due to the isomorphism between Euclidean space and tensor space, we define an isomorphic mapping $\phi_{(c,m,m)} : \mathbb{T}_{(c,m,m)} \to \mathbb{R}^{c \times m \times m}$. Where $\{\bar{\mathbf{e}}_{\bar{\alpha}}\}_{\bar{\alpha} \in [0..c \times m \times m-1]}$ is the Euclidean basis for $\mathbb{R}^{c \times m \times m}$ and $\{\mathbf{e}_\alpha\}_{\alpha \in \mathrm{Index}\left(\mathbb{T}_{(c,m,m)}\right)}$ is the corresponding Euclidean basis of $\mathbb{T}_{(c,m,m)}$. And let $(c, m, m) = (i_0, i_1, i_2) \in \mathbb{N}^3$ denote a tuple. Then, if $\phi_{(c,m,m)}(\mathbf{e}_\alpha) = \bar{\mathbf{e}}_{\bar{\alpha}}$, we have

$$\bar{\alpha} = \sum_{j=0}^{1} \left[\alpha_j \cdot \Pi_{\beta=j+1}^{2} i_\beta\right] + \alpha_2. \tag{32}$$

Then we define another mapping, it's like embedding the parameters of the convolutional layer in an image tensor, $\psi_{i,j}^{(k,m)} : \mathbb{T}_{(c,k,k)} \to \mathbb{T}_{(c,m,m)}$, such that for all $W \in \mathbb{T}_{(c,k,k)}$ with the stride ω and the padding p:

$$\psi_{i*\omega,j*\omega}^{(k,m+2p)}(W) = \sum_{r,s,t} W_{r,s,t} \cdot \mathbf{e}_{r,i*\omega+s,j*\omega+t}. \tag{33}$$

Then we require a padding operator Γ to accomplish arbitrary padding:

$$\Gamma = I_c \otimes P^T \tag{34}$$

The elements of the matrix P, denoted as P_{ij}, are defined as:

$$P_{ij} = \begin{cases} 1 & \text{If } j \text{ matches the position of } i \text{ after padding} \\ 0 & \text{others} \end{cases} \tag{35}$$

By utilizing the aforementioned three mappings, the convolutional layer $W \in \mathbb{T}_{(d,c,k,k)}$ with stride ω and padding p can be expressed as a linear operator, let $g : \mathbb{T}_{(d,c,k,k)} \to \mathbb{T}_{(dn^2, c(m+2p)^2)}$, the r^{th} row of $g(W)$ is a reshaped embedding of the h^{th} filter in W into $\mathbb{R}^{c(m+2p)^2}$:

$$g(W)[r,:] = \phi_{(c,m+2p,m+2p)} \circ \psi_{i*\omega,j*\omega}^{(k,m+2p)}(W_h) \tag{36}$$

Finally, we will act on the linear operator Γ to change the dimension of $g(W)$ and get the Toeplitz matrix $\bar{W} \in \mathbb{T}_{(dn^2,cm^2)}$:

$$\bar{W} = g(W) \cdot \Gamma. \tag{37}$$

Thus, Convolution can be performed as matrix multiplication like this:

$$\phi_{(d,n,n)}(Y) = \phi_{(d,n,n)}(W \star X) = \bar{W} \cdot \phi_{(c,m,m)}(X) \tag{38}$$

Remark 1. *Note: When we choose the convolutional layer $W \in \mathbb{T}_{(64,64,3,3)}$ with $padding = 1$ and $stirde = 1$. The input of W is $\boldsymbol{x} \in \mathbb{T}_{(64,224,224)}$, the toeplitz matrix of W is $\bar{W} \in \mathbb{T}_{(3211264,3211264)}$. We need to sequentially compute the Toeplitz matrices of the convolutional layers based on the output channels and store them as sparse matrices. The specific procedure is detailed in Algorithm 2.*

D BatchNorm Layer as a Linear Operator

Appendix D corresponds to the "**Section 4.3: BatchNorm Layer as a Linear Operator**" in the main paper.

In this section, we will describe in detail the method of treating batch normalization as a linear operation and explain how to merge the convolutional layer and batch normalization layer into a **Double Layer**.

D.1 Specific Method

We define γ_c as the scale factor, v_c as the moving variance for channel c, and ε as a small constant added for numerical stability. Given a feature map F in the $c \times h \times w$ order (channel, height, width), its normalized version $\hat{F}$ can be obtained by performing the following matrix-vector operations at each spatial position i, j:

$$\begin{pmatrix} \hat{F}_{1,i,j} \\ \hat{F}_{2,i,j} \\ \vdots \\ \hat{F}_{c-1,i,j} \\ \hat{F}_{c,i,j} \end{pmatrix} = \begin{pmatrix} \frac{\gamma_1}{\sqrt{v_1+\epsilon}} & 0 & \cdots & 0 \\ 0 & \frac{\gamma_2}{\sqrt{v_2+\epsilon}} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{\gamma_c}{\sqrt{v_c+\epsilon}} \end{pmatrix} \cdot \begin{pmatrix} F_{1,i,j} \\ F_{2,i,j} \\ \vdots \\ F_{c-1,i,j} \\ F_{c,i,j} \end{pmatrix} + \begin{pmatrix} \beta_1 - \gamma_1 \frac{\mu_1}{\sqrt{v_1+\epsilon}} \\ \beta_2 - \gamma_2 \frac{\mu_2}{\sqrt{v_2+\epsilon}} \\ \vdots \\ \beta_{c-1} - \gamma_{c-1} \frac{\mu_{c-1}}{\sqrt{v_{c-1}+\epsilon}} \\ \beta_c - \gamma_c \frac{\mu_c}{\sqrt{v_c+\epsilon}} \end{pmatrix} \tag{39}$$

From the above equation 39, it is evident that the BatchNorm (BN) layer can be represented as a linear operator and can also be implemented as a 1×1 convolution. Furthermore, since BN layers are typically placed after convolutional layers, as seen in architectures like GoogLeNet (Szegedy et al., 2015) and ResNet(He et al., 2016), we can merge them together.

D.2 Merging Convolutional and Batch Normalization Layers into a Double Layer

Based on Markus's work on fusing these two layers together (Markus, 2018), we have two approaches to handle the **Double Layer**. First, directly multiply the matrix of the convolutional layer by the linear component of the batch normalization. Second, express the batch normalization as a 1×1 convolution and merge it into the convolutional layer. Next, we will talk about how to merge them together. Let $\mathbf{W}_{BN} \in \mathbb{R}^{c \times c}$ and $\mathbf{b}_{BN} \in \mathbb{R}^c$ denote the matrix and bias from the above equation 39, and $\mathbf{W}_{\text{conv}} \in \mathbb{R}^{c \times (c_{\text{prev}} \cdot k^2)}$ and $\mathbf{b}_{\text{conv}} \in \mathbb{R}^c$ the parameters of the convolutional layer that precedes batch normalization, where c_{prev} is the number of channels of the feature map F_{prev} input to the convolutional layer and $k \times k$ is the filter size.

Given a $k \times k$ neighbourhood of F_{prev} unwrapped into a $k^2 \cdot c_{\text{prev}}$ vector $\mathbf{f}_{i,j}$, we can write the whole computational process as:

$$\hat{\mathbf{f}}_{i,j} = \mathbf{W}_{BN} \cdot (\mathbf{W}_{\text{conv}} \cdot \mathbf{f}_{i,j} + \mathbf{b}_{\text{conv}}) + \mathbf{b}_{BN}$$

Thus, we can replace these two layers by a single convolutional layer with the following parameters:

- filter weights: $\mathbf{W} = \mathbf{W}_{BN} \cdot \mathbf{W}_{\text{conv}}$;
- bias: $\mathbf{b} = \mathbf{W}_{BN} \cdot \mathbf{b}_{\text{conv}} + \mathbf{b}_{BN}$.

E Algorithm 2

In this section, we have the following algorithm to obtain all singular vectors $\boldsymbol{v}_k$ for $k = 1, \ldots, \ell$ for all linear layers of a target l-layers CNN f.

Algorithm 2: Algorithm for obtaining all singular vectors $\boldsymbol{v}_k$ from the target model, for $k = 1, \ldots, \ell$

```
Input: Target $l$-layer CNN $f$
Input: empty list V, empty list S
Output: singular vectors $\boldsymbol{v}_k$ for $k = 1, \ldots, \ell$
1  for $k = 1$ to $\ell$ do
2      if k-th layer is a convolutional layer then
3          $W \in \mathbb{T}_{(d,c,k,k)} \leftarrow$ Parameter of the layer;
4          for $i = 1$ to $d$ do
5              $W_i \leftarrow W[i, :, :, :]$;
6              $g(W_i)[r, :] \leftarrow \phi_{(c,m+2p,m+2p)} \circ \psi^{(k,m+2p)}_{i*\omega,j*\omega}(W_i)$;
7              $\bar{W}_i \leftarrow g(W_i) \cdot \Gamma$;
8              $S$.append(sparse($\bar{W}_i$));
9          $S \leftarrow$ vstack($S$);
10         $[U, \Sigma, V^T] \leftarrow$ svd($S$);
11         $\boldsymbol{v}_k \leftarrow V[:, 0]$;
12     else
13         if k-th layer is a batchnorm layer then
14             $A_{ii} \leftarrow \frac{\gamma_c}{\sqrt{v_c+\varepsilon}}, \quad c = \left\lfloor \frac{i}{m \times m} \right\rfloor + 1$;
15             $[U, \Sigma, V^T] \leftarrow$ svd($A$);
16             $\boldsymbol{v}_k \leftarrow V[:, 0]$;
17         if k-th layer is a fully connected layer then
18             $F \in \mathbb{T}_{(a,b)} \leftarrow$ Parameter of the layer;
19             $[U, \Sigma, V^T] \leftarrow$ svd($F$);
20             $\boldsymbol{v}_k \leftarrow V[:, 0]$;
21     $V$.append($\boldsymbol{v}_k$);
22 return $V$;
```

In Algorithm 2. It's important to note that expanding a convolutional layer into multiple Toeplitz matrices often results in very large and sparse matrices. Therefore, following steps 4-8 in the algorithm, we expand the output channels of the convolutional layer into sparse matrices, concatenate them vertically, and then perform SVD on the resulting matrix.

F Experimental Details for Deep Models as L1LOS

Appendix F corresponds to the "**Section 4.1: Deep Models as L1LOS**" in the main paper. In this section, we'll detail our optimization Algorithm 1and its associated hyperparameters. Given the diverse internal architectures of networks like AlexNet, VGG16, and VGG19 lacking BatchNorm layers, ResNet's residual and Bottleneck structures, and GoogleNet's Inception modules Figure 2, It is essential to customize solutions according to these structural differences.

For AlexNet, VGG16, and VGG19, they adhere to the L1LOS structure. Except for non-linear components such as Max pooling and ReLU, we can directly compute the maximal right singular vector of all convolutional and fully connected layers, enabling the straightforward application of our loss function.

For ResNet152 and GoogLeNet, we've replaced the bias in convolutional layers with BatchNorm layers, creating what we term a **double-layer** architecture, and we can merge them together, see Appendix D.2. We compute the maximum right singular vector of the linear part of this double-layer. For GoogLeNet's unique

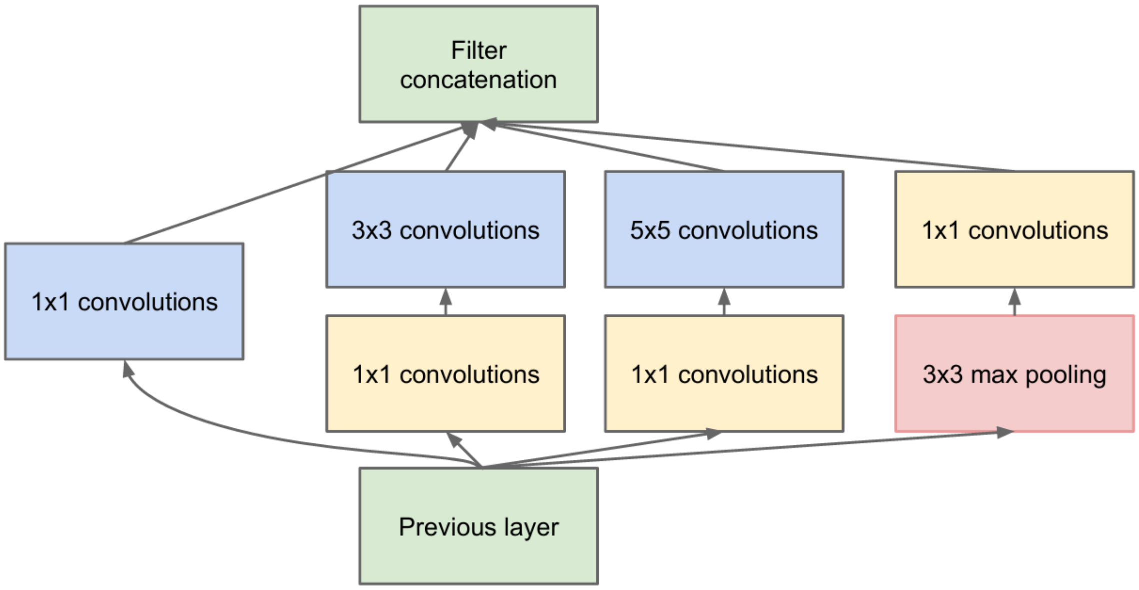


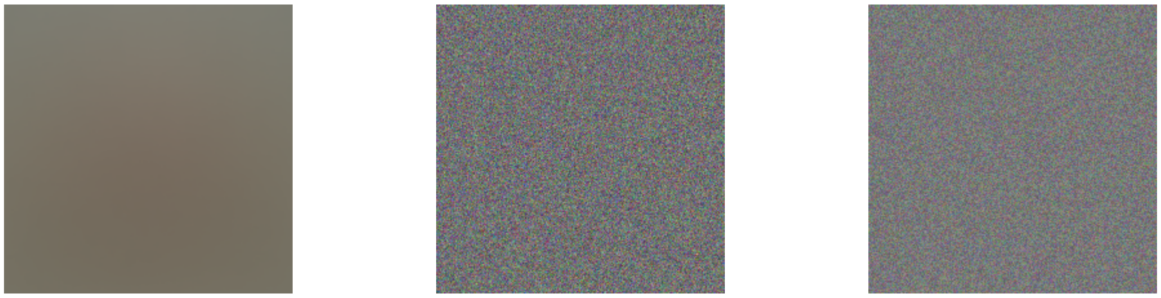
Figure 2: Inception module of GoogleNet.

Inception architecture, with four branches per module, we assign distinct penalty coefficients to each branch, treating the structure as a 4-branch channel architecture ($\lambda_1 = 0.1$, $\lambda_2 = 0.6$, $\lambda_3 = 0.3$, and $\lambda_4 = 0.1$). Due to ResNet's special residual structure, We've added $\text{Loss2} = -\sum_{\boldsymbol{x}\sim\mathcal{N}(\mu,\sigma)} \log\left(\prod_{i\in K} \|f_i(\boldsymbol{x}+\boldsymbol{\delta})\|_2\right)$, where K represents the set of layer number of all Bottleneck layers, aiming to maximize the outputs of every Bottleneck layers. We use Loss2 only on the ResNets model.The ratio of Loss1 to Loss2 is 1:1.

The input data $\boldsymbol{x}$ in optimization Algorithm 1is defined as follows: $\boldsymbol{x} = d \sim \mu_{RGB}[0.485, 0.456, 0.406]$ for the range prior case, $\boldsymbol{x} = d \sim \mathcal{N}(\mu,\sigma)$ for the Gaussian prior case, and $\boldsymbol{x} = d \sim \mathcal{U}(a,b)$ for the uniform prior case. The vissualization of input data $\boldsymbol{x}$ is in Figure 3. For optimization, we've opted for Adam with a learning rate of 0.5, coupled with a StepLR scheduler reducing the learning rate by 0.7 every hundred epochs. The UAP is confined within the range of $(-\epsilon/255, \epsilon/255)$, where ϵ is set to 10. We've chosen a total of 800 epochs. Cause we only use a few samples, optimization can be completed in about a minute.

F.1 More Visualization of IntriUAPs

Figure 3 shows different initial images. Figure 4 shows the intriUAPs for different models on ImageNet dataset. Figure 5 presents a visualization of the perturbations on VGG19 under $\ell_\infty = 10$. Figure 6 displays the maximum singular values of the network's linear layers. Figure 7 illustrates the intriUAPs of various semi-white-box models.



(a) Range prior (b) Gassian (c) Uniform

Figure 3: Different initial images

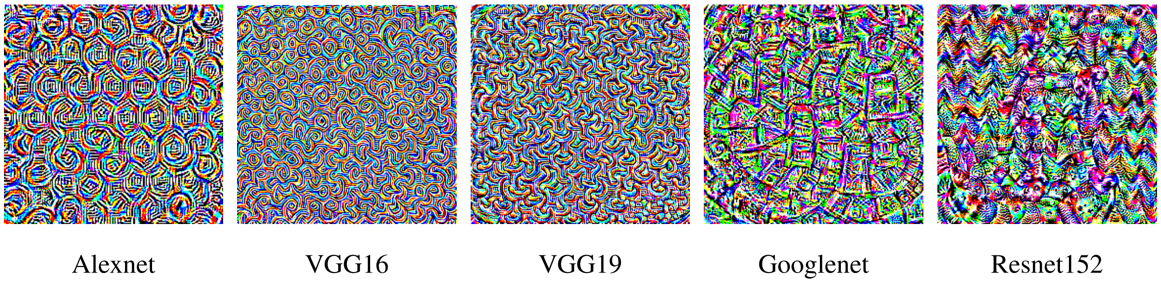


Figure 4: Intrinsic UAPs for multiple models on ImageNet dataset. Perturbaitons were constrainted with ℓ_∞-norm $= 10$ .Using range prior. UAPs are best viewed in color.

We observed that our method does not perform well on ResNet152 and GoogLeNet. Figure 6 displays the maximum singular values of the linear portions of Alexnet, GoogLeNet, VGG19, and ResNet152. It is evident that the maximum singular values for AlexNet and VGG19 are larger than those for GoogLeNet and ResNet152, implying a smaller upper bound on the maximum change through linear perturbations on these two models.(Dittmer et al., 2019) indicates that ReLU has the effect of dampening the signal, allowing only inputs strongly correlated to the right singular vectors with the largest singular values to persist through ReLU. Although we have applied δ as much as possible to the maximum right singular vectors of each linear layer, their scaling differs. Moreover, we may overly idealize CNNs as linear systems, and the impact of non-linear components, along with affine biases, is more pronounced in GoogLeNet and ResNet152 due to their numerous batch normalization layers and 1×1 convolutions. The maximum singular values of 1×1 convolutions are often even smaller than 1, indicating that they carry very little information.

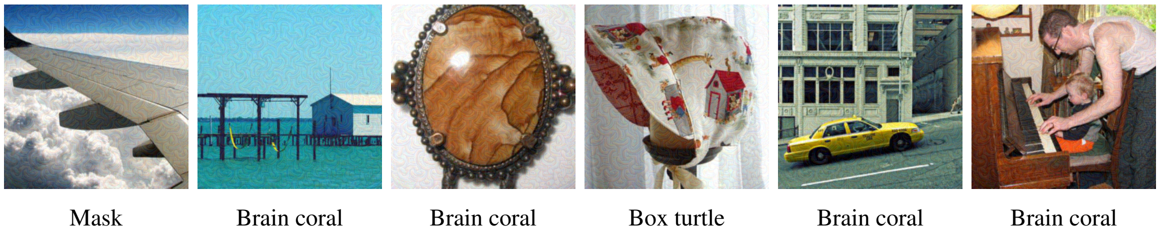


Figure 5: Examples of perturbed images generated by VGG19 with IntriUAP and their corresponding labels. The perturbaitons were constrainted with ℓ_∞-norm $= 10$. Using range prior.

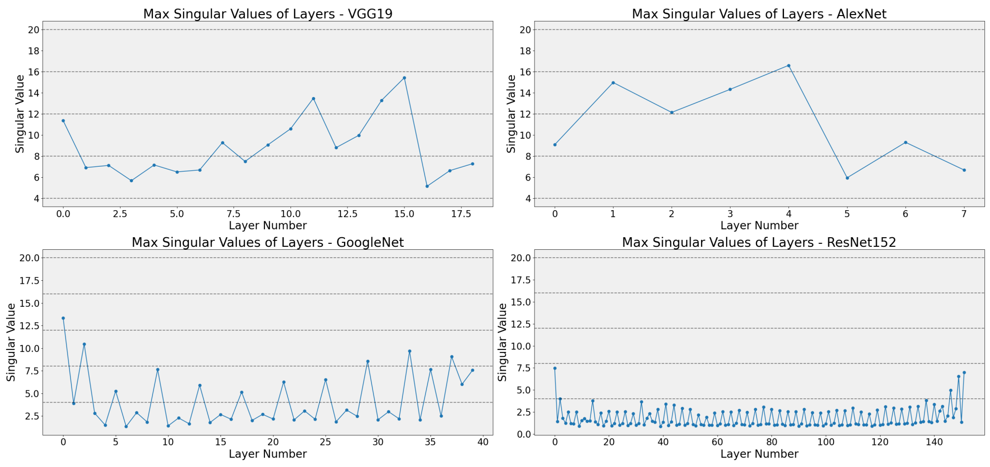


Figure 6: Singular values of linear layers within different models.

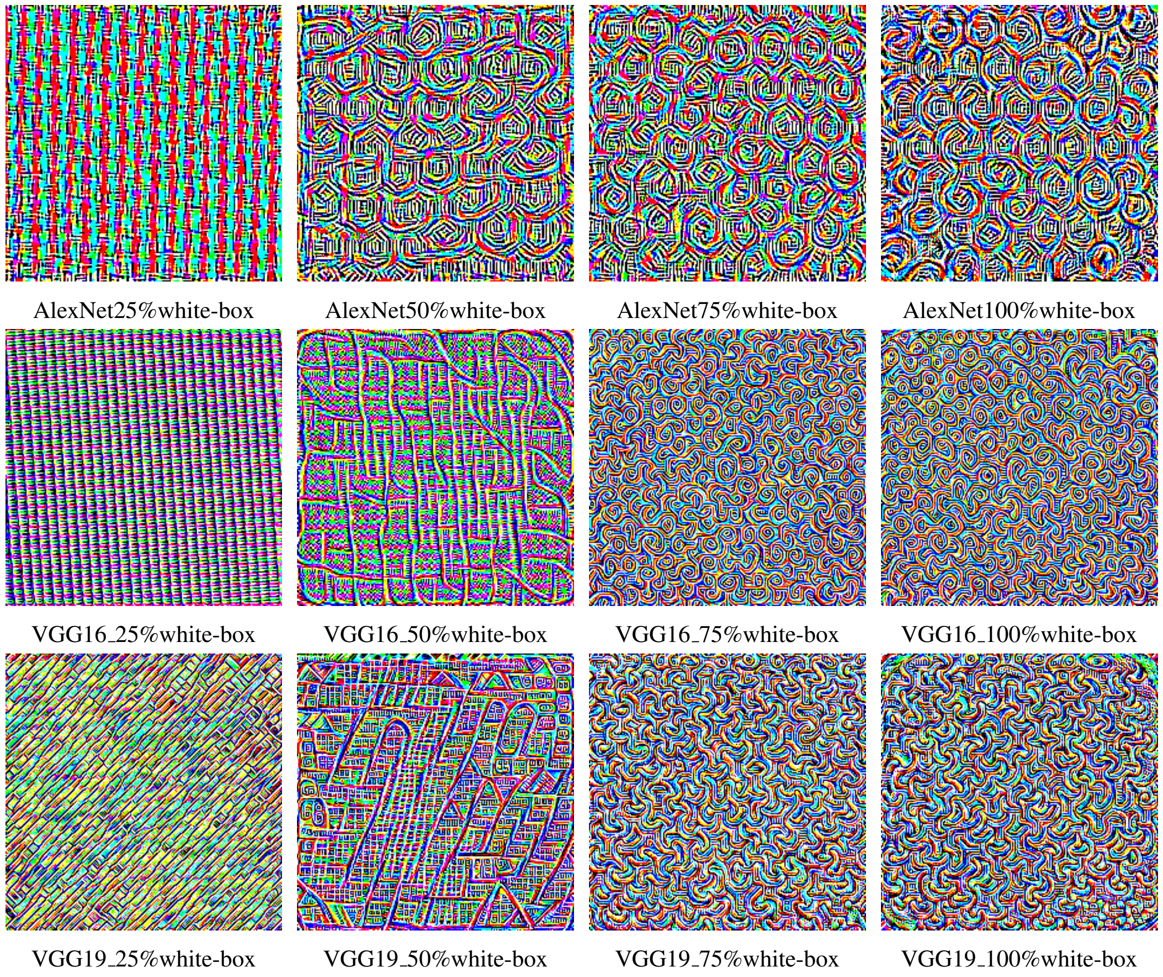


Figure 7: Universal adversarial pertubations crafted by IntriUAP objective for multiple semi-white-box models on Imagenet dataset. Perturbaitons were constrainted with ℓ_∞-norm $= 10$. Using range prior. UAPs are best viewed in color.

References

Dittmer, S., King, E. J., and Maass, P. (2019). Singular values for relu layers. *IEEE transactions on neural networks and learning systems*, 31(9):3594–3605.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Markus, N. (2018). Fusing batchnorm and conv.

Praggastis, B., Brown, D., Marrero, C. O., Purvine, E., Shapiro, M., and Wang, B. (2022). The svd of convolutional weights: A cnn interpretability framework. *arXiv preprint arXiv:2208.06894*.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9.