# 武汉大学国家网络安全学院

# 实验报告

| | |
|---|---|
| 课程名称: | 信息隐藏 |
| 专业年级: | 19 级网安 8 班 |
| 姓　　名: | 潘昊　庞远心　吴雨晗 |
| 学　　号: | 2019302050240　2019302050244　2019302141236 |
| 实验学期: | 2021-2022 学年　　第 1 学期 |
| 填写时间: | 2022 年 1 月 10 日 |

# 目录

# 一、 摘要

机器学习，尤其是深度神经网络（DNN）技术，近年来在诸多领域取得了巨大成功，许多科技公司都将神经网络模型部署在商业产品中，提高效益。训练先进的神经网络模型需要大规模数据集、庞大的计算资源和设计者的智慧。这具体体现在：

1. 深度学习模型应用的训练模型规模巨大，以 GPT-3 为例，其预训练所用的数据量达到 45TB，训练费用超过 1200 万美元，有着极高的经济成本；
2. 深度学习模型在训练部署到工业应用场景过程中（比如智慧金融，智慧医疗应用），需要引入金融、医疗等领域专有先验知识，因此在模型设计过程就需要引入专家的知识和经验来订制模型，这体现了人脑力的知识产权。
3. 深度学习模型的训练过程，需要特定领域的海量数据作为训练资源，存在数据本身价值和知识属性。

以上属性决定了经过训练的深度学习模型具有很高的商业价值和知识属性，必须将其纳入合法所有者（即创建它的一方）的知识产权。因此，从技术上迫切需要保护深度神经网络（DNN）模型不被非法复制、重新分发或滥用。

因此，这次大实验我们将注意力集中于深度神经网络的版权保护，通过在神经网络中嵌入水印，达到版权保护的目的。

# 二、 背景调研

## 2.1 问题建模

深度学习模型的版权保护可以被建模为图 1的情景。

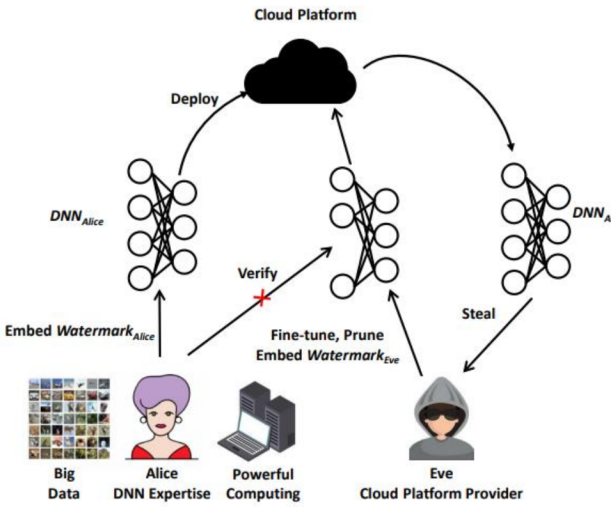


图 1: 问题抽象和建模

在这个问题中，Alice 是模型所有者，为保护模型，Alice 在模型中嵌入后门，并部署到相应云计算平台上。Eve 是黑客或云计算服务商，可以访问 DNN 的结构和参数，窃取模型并将其商业化获取利润，

Eve 为了逃避 Alice 的检测，使用一定的攻击手段来破坏 Alice 的水印。此外，还选择自己的后门样本注入被盗模型，作为自己的数字水印。Alice 如何检测 Eve 的模型是否侵犯自己的知识版权便是我们所探讨的重点。

### 2.2 数字水印

数字水印是一种基于内容的信息隐藏技术，用于在载体数据（例如，多媒体、文档、软件、数据库等）中嵌入/检测数字信息。希望嵌入的水印不影响载体数据的正常使用，并且不易被检测或去除。

近年来，数字水印被应用于 DNN 模型的保护，可分为**白盒方法和黑盒方法**。

白盒方法假设 DNN 模型的所有者可以访问可疑模型的 “内部” 以验证模型的所有权。这种方法在训练过程中，使用参数正则化器将作为参数矩阵的水印嵌入到目标模型的参数空间中。在验证可疑模型的所有权时，所有者从模型中检索矩阵，计算检索到的矩阵的乘积和预先生成的密钥向量，并将结果与 所有者保存的本地秘密进行比较。

相比之下，黑盒水印方法主要依靠探测可疑模型进行所有权验证，从而放宽了对可疑模型内部细节的要求。黑盒水印方法按照嵌入的水印是否可以被人类视觉系统感知，可以进一步分为两类，即盲方法和非盲方法。大多数黑盒水印方案都是基于将后门嵌入目标 DNN 模型来实现的。将后门嵌入到神经网络模型中是正常使用训练数据集训练模型，但是在训练数据集中会增加某些特定输入（即触发器）及其对应的标签（但这些标签通常是错误的），这些标签由模型的所有者选择，而且可以证明模型的可区分行为。训练后，模型可以为特定的输入输出特定的标签。这些特定输入与特定标签之间的映射被视为后门，并用作水印。特定的输入由模型的所有者保密。如果检测到后门（也可称为水印），模型的所有者可以声明其对可疑模型的所有权，因为模型在给定来自所有者秘密的特定输入的情况下输出所需的标签。

### 2.3 攻击分类

总体而言，攻击可以被分为去除攻击和模糊攻击两大类。而每个大类下又有一些更为细致的攻击手段。

- **去除攻击**（Removal Attack）
  1. 模型微调攻击（model fine-tuning）

     模型微调指的是黑客在获取带有水印的神经网络模型后对于模型各层的参数等进行微调，从而破坏水印。
  2. 模型修剪攻击（model pruning）

     模型修剪是指黑客删除神经网络模型中不重要的参数，将模型压缩，在几乎不会损失准确率的基础上，生成更小的、内存效率更高、功率效率更高的模型。典型的修建方法有归零修剪，是指在修剪神经网络时，剪掉连接较少、不重要的神经元。会破坏水印，且不涉及重新训练。
  3. 内核截断攻击（Kernels cutoff）

     内核截断攻击指攻击者修改神经网络模型中的各层，通过增加或删减的方法使原本水印不可见。在这种攻击方法下，攻击者可以完全移除一些内核（不是修剪），或补充被盗模型中的一些内核，以获得性能相似但内核数量和结构不同的新模型。内核截断会降低模型的性能，攻击者不会从被盗模型中截取太多内核。
- 模糊攻击（Ambiguity attack）

1. 对抗样本攻击
   所谓对抗样本攻击是黑客在原始样本添加一些人眼无法察觉的扰动（这样的扰动不会影响人类的识别，但是却很容易愚弄模型），致使机模型做出错误的判断，由于模型的准确率下降会让模型所有者误以为该模型没有侵权。
2. 伪造水印攻击（Forge a watermark）

### 2.4 技术现状

下面将简单介绍在神经网络版权保护领域已经被提出且证明为有效的方法。

#### 2.4.1 参数正则化器嵌入水印

该方法由 Uchida et al 提出，论文首先定义了水印技术的场景，并提出了一种水印嵌入技术，可以在模型训练、精调或者蒸馏过程中嵌入到目标模型中且不影响模型性能。并且面对攻击者的精调和剪枝等行为时仍能保留在模型中。该方法旨在向被保护模型嵌入水印的方式为参数正则化，其利用的原理是带有嵌入水印的层的参数方差比没有嵌入水印的其他层更显著。

主要考虑卷积神经网络中卷积层的参数。对于一个 $Kernel-Size=S$ ，输入通道数为 $D$ ，输出通道数 (卷积核数量) 为 $L$ 的卷积层，不考虑偏置，其参数为 $w\in\mathbb{R}^{S\times S\times D\times L}$ 。计算多个卷积核的均值 $\bar{W}_{i,j,k}=\frac{1}{L}\sum_l W_{i,j,k,l}$ ，并展平得到 $w\in\mathbb{R}^M(M=S\times S\times D)$ 作为嵌入的目标。将 $T$ bit 的信息 $b\in 0,1^T$ 嵌入其中。在提取时，使用 $b_j=s\left(\sum_i X_{ji}\omega_i\right)$ 计算嵌入的信息。其中 $\omega$ 表示卷积核均值，$X\in\mathbb{R}^{T\times M}$ 表示嵌入密钥，$s(\cdot)$ 为阶跃函数。在训练时，为了将信息嵌入模型中，在原本的损失函数上添加了一个权重参数正则项 $E(\omega)=E_0(\omega)+\lambda E_R(\omega)$ 。考虑到上述的模型提取方式类似二分类方法，因此添加的权重参数正则项使用 BCE 损失，使用训练过程中的参数提取结果作为监督。

$$
\begin{aligned}
E_{R(\omega)}&=-\sum_j=1^T\left(b_j\log\left(y_j\right)+\left(1-b_j\right)\log\left(1-y_j\right)\right)\\
y_j&=\sigma\left(\sum_i X_{ji}\omega_i\right)\\
\sigma(x)&=\frac{1}{1+\exp(-x)}
\end{aligned}
$$

关于密钥 $X$ 的设计，考虑到水印的性能，有三种设计形式：$X^{\text{direct}}$ 的每一行为独热码，直接将信息映射到参数中。$X^{\text{diff}}$ 的每一行包含一个 1 和一个-1，其他的为 0 ，将信息映射到参数的差值中。$X^{\text{random}}$ 的数字采样于标准正态分布。

该方法的优点是简单易行，实验效果显著，但易被统计分析攻破。

#### 2.4.2 基于后门的水印方法探索黑盒或灰盒神经网络模型

该方法由 Adietal Zhang Guo 等人提出，该方法不会被统计分析攻破，但提出的后门水印方法是非盲水印，要么无法防御逃避攻击，要么没有明确解决欺诈所有权的问题。

这里，逃避攻击是指通过训练一个强大的检测器来识别所有者触发集中的关键样本，并在检测到用作输入的关键样本时返回一个随机标签而不是预定义的标签，从而逃避基于后门的水印验证。

### 2.4.3 基于后门的盲水印方法

Zheng Li, Chengyu Hu etc 提出了一种基于盲水印的知识产权保护（IPP）框架，是一种可以达到安全性和可行性要求的深度神经网络水印。这个框架接受普通样本和特定标识的样本作为输入，将新生成的样本作为水印输出，这些新生成的水印无法被人的视觉系统所感知，而且通过分配特定标签将这些水印嵌入 DNN 模型，可以留下后门作为版权声明。

但是这种方法无法处理所有权的欺诈性声明，因为攻击者仍然可以通过利用对抗样本来构建自己的触发集来冒充所有者。

目前现有的算法对策都**只能抵抗较低级的独立的攻击手段**，而当各种攻击方法有机组合或同时施展多种攻击时却力不从心，因此，针对能抵抗多种或多种攻击组成的综合攻击的研究是当前个重要的研究方向，而本次实验所依据的利用深度神经网络来进行版权保护就是一个目前的一种对策。

## 三、 实现方法

这部分主要介绍神经网络的版权保护的具体实现方法。

在神经网络的版权保护中，主要的流程为：水印生成、水印嵌入和所有权验证。

在**水印生成阶段**，首先在公开的数据集 $D_s$ 上训练一个神经网络 $Net$，这个神经网络的参数空间分为两部分：嵌入块（EPH）和密钥块（SPH），其中 EPH 会用于水印的嵌入，SPH 部分作为本地密钥保存。

在**水印嵌入阶段**，模型所有者会将水印生成阶段的神经网络 EPH 的参数空间嵌入他自己的 DNN 模型，也就是要被保护的深度神经网络模型（DNN-to-be-protected），并在大型数据集 $D_t$ 上进行训练，在训练的同时，固定 EPH 来满足训练的任务（DNN-watermasked）。

在**所有权验证阶段**，模型所有者从存疑的水印模型中提取水印 $EPH_{retrieved}$，并将其本地保存的 SPH 的参数空间合并，从而产生一个神经网络 $Net_{Combined}$。模型所有者可以在和 $D_s$ 有相同特性的公开的测试集 $D_{S_{TEST}}$ 上，以此来验证模型是否归自己所有。

在整个过程中，都遵循 Kerckhoffs 原则，也即**假设水印嵌入算法和所有权验证算法是公开的**。虽然攻击者可以从公开的训练数据集 $D_s$ 中伪造 $Net$，但他们需要使用公开的测试数据集 $D_{S_{TEST}}$ 并遵循公有验证算法来声明对有争议模型的所有权。

论文中提到的工作架构可以用图 2来表示。

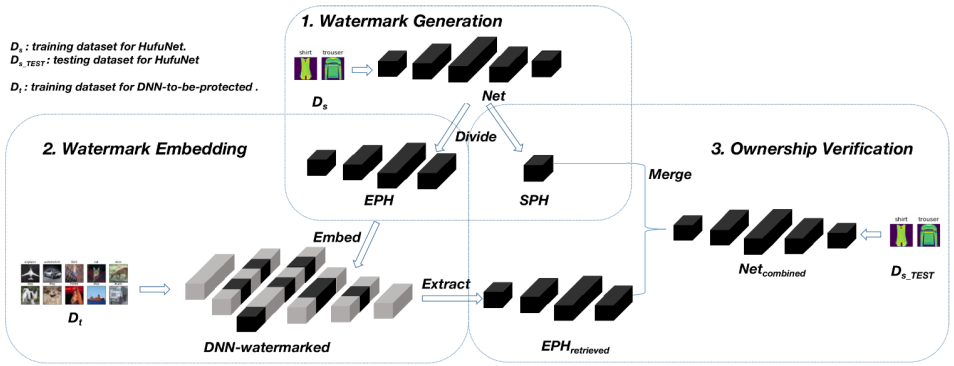


图 2: 工作架构

### 3.1 水印生成

神经网络 $Net$ 的结构由几个卷积层和一个全连接层组成。$Net$ 在公开数据集 $D_s$ 上训练后，**所有卷积层**都用作水印 EPH，而**全连接层**存储在本地作为密钥 SPH。EPH 的大小通常远小于嵌入 EPH 的 DNN 模型。

在 $Net$ 在数据集 $D_s$ 上训练之后，我们可以把 $Net$ 视为一个从输入空间到输出空间的映射关系，这种映射关系在和 $D_s$ 有相同特性的数据集 $D_{S_{TEST}}$ 上，也能实现很高的准确率。因此 $D_s$ 和 $D_{S_{TEST}}$ 被公开用于 $Net$ 的训练和验证。

### 3.2 水印嵌入算法

水印嵌入主要包括两个部分:水印嵌入和模型训练。这里模型的训练指的是对模型 DNN-watermarked 的训练过程。

#### 3.2.1 水印嵌入

水印被嵌入到 DNN-to-be-protected 模型的卷积层中。在这个过程中，EPH 的每个卷积核都被单独嵌入到 DNN-to-be-protected 模型的卷积层中。在这个过程中，EPH 的每个卷积核都被单独嵌入到 DNN-to-be-protected 中，此外，会产生一个密钥，这个密钥可以检测嵌入的水印，而且足够强大，可以抵御潜在的如暴力搜索等攻击。通过密钥的参数值和索引，可以确定每个内核的嵌入位置，从而确保 EPH 在模型 DNN-to-be-protected 的参数空间中的隐蔽性。

下面说明水印的嵌入算法。

首先，会先初始化一个位图 bitmap 和模型 DNN-watermarked$f_wm$，这个位图存储了模型 DNN-to-be-protected 的水印嵌入情况，且模型 DNN-watermarked$f_wm$ 被初始化为 DNN-to-be-protected。然后对于水印 EPH 的每一个卷积核，首先会对它的索引值 i 和大小为 $k*k$（实验中 EPH 的 $k$=3）的参数空间进行一个 XOR 运算，对结果进行密钥为 $key$ 的哈希运算，最后对模型 DNN-to-be-protected 中的卷积核数量进行取模得到计算嵌入位置。如果这个位置在 bitmap 中已经被标记嵌入过，那就寻找下一个没有被嵌入的位置。一旦找到可以嵌入的位置，我们就将卷积核嵌入到 DNN-watermarked 中，并对位图 bitmap 进行更新，将刚才嵌入的位置标记为已嵌入。紧接着，进行下一次嵌入。最后我们会输出已经嵌入水印的模型 DNN-watermarked。

算法的具体实现如表 1所示。

需要注意的是，在这个过程中，EPH 的卷积核参数被用于确定 EPH 在 DNN-watermarked 的嵌入位置，而且模型 DNN-to-be-protected 的所有者需要同时保存 $Net$ 的 EPH 和 SPH，以供后续进行所有权的验证。在所有权的验证过程中，被保存的 EPH 被用于确定存疑模型中的水印嵌入位置。

#### 3.2.2 模型训练

要保护的模型 DNN-to-be-protected 也即 $f_t$ 可以看成是从输入空间到输出空间的映射，在训练之前模型 $f_t$ 的所有者随机初始化 $f_t$，然后按照上面的算法将 EPH 的参数空间嵌入 $f_t$。之后将 $f_t$ 在训练集 $D_t$ 上进行训练。在 $f_t$ 的训练过程中，嵌入的 EPH 的参数空间保持固定，仅仅更新 $f_t$ 剩余的部分。通过这样的方式，EPH 的参数空间保持不变，$f_t$ 的其他部分持续更新，且 $f_t$ 训练过程的损失函数保持不变。

表 1: 水印嵌入算法

**Algorithm 1** 水印嵌入算法

**输入**：$f_t$：DNN-to-be-protected；$N$：$f_t$ 中的卷积核数量；KERNEL：EPH 水印中要被嵌入的所有卷积核；$n$：KERNEL 中的卷积核数量；bitmap：记录 $f_t$ 中水印的嵌入位置的位图；key：密钥

**输出**：$f_{wm}$：DNN-watermarked

**算法流程**：

bitmap = NULL;$f_{wm}$=$f_t$

**for** i in (1,n) **do**

  position = $HAMC(key, XORPMV(kernel_i) \bigoplus i)\%N$

  **while** $bitmap_{position}$ **do**

    $position = ++position\%N$

  **end while**

  $f_{wm} = EMBED(kernel_i, position, f_{wm})$

$bitmap_{position} = 1$

++i

**end for**

**return** $f_{wm}$

XORPMV 函数是对一个卷积核的所有参数进行 XOR 计算，EMBED 是将 EPH 一个卷积核嵌入到 $f_t$ 的特定位置，HMAC 是一个带密钥的加密哈希函数。

### 3.3 所有权验证

当出现一个存疑模型 $f_{suspect}$,模型的所有者首先从 $f_{suspect}$ 中提取出 EPH 的参数空间 $EPH_{retrieved}$，提取的算法和表 1的步骤一致。根据模型所有者保存的最原始的 $Net$，可以计算出 $f_{suspect}$ 中嵌入的水印的位置，将提取出的参数空间存储为 $EPH_{retrieved}$。然后模型所有者将 $EPH_{retrieved}$ 和本地保存的密钥 $SPH$ 合并，得到模型 $Net_{Combined}$。

如果存疑模型 $f_{suspect}$ 是从 $f_wm$ 窃取得来，那么模型 $Net_{Combined}$ 将在标准的公开数据集 $D_{sTEST}$ 上取得很高的准确率，也就是说，通过测试 $Net_{Combined}$ 在数据集 $D_{sTEST}$ 上得到的准确率 $Acc_{Combined}$，来确定模型 $f_{suspect}$ 是否被窃取。如果模型 $f_{wm}$ 在被窃取之后，没有经过重新训练，那么将会在数据集 $D_{tTEST}$ 上有和原始 $Net$ 一样的准确率 $Acc_{ori}$。否则，$Acc_{Combined}$ 会在 $Acc_{ori}$ 的基础上有所下降。

为了解决这个问题，定义 $Diff_{Acc}$ 来记录 $Acc_{Combined}$ 和 $Acc_{ori}$ 的差值的绝对值，并定义一个阈值 $\tau acc$ 作为模型是否被窃取的判断阈值。当 $Diff_{Acc}$ 小于 $\tau acc$ 时，我们认为，模型被窃取。经过实验，可以确定，当被保护的模型经过剪枝、内核切断等攻击，其效果大大下降时，$Diff_{Acc}$ 仍能保持在一个较低的水平，经过实验的测试，我们设定阈值 $\tau acc$ 为 15%.

### 3.4 模型与算法的改进

#### 3.4.1 改进的水印模型

论文中，在生成水印时，直接采用 FLNET 的所有卷积层作为水印 EPH，而将所有的全连接层作为 SPH。除此之外，在进行嵌入时，也仅仅针对要保护的模型的卷积层进行了卷积核的嵌入。

这样做的问题在于，一旦攻击者发现了嵌入的规律，将所有的攻击聚焦于模型的卷积层上面，将会使得模型的水印遭到很大程度的破坏，使得水印的鲁棒性大大降低。

基于论文的水印生成模型，我们在此基础上改进了水印模型的结构。改进之后的水印模型结构如图 3和 4所示。

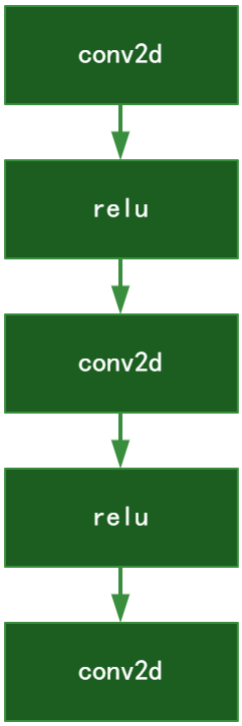


图 3: encoder 结构

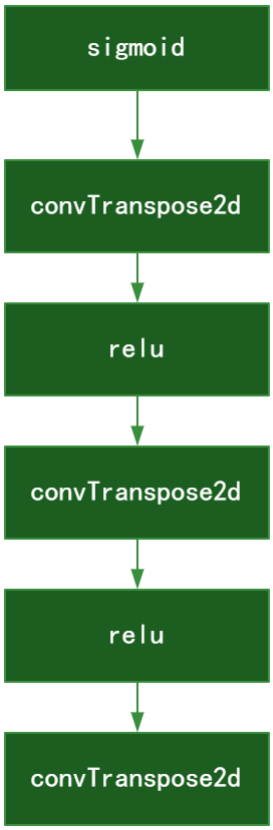


图 4: decoder 结构

可以看到，我们将改进的模型分为了两个部分：Encoder 和 Decoder，作为水印嵌入 DNN-to-be-protected 的 Encoder 和作为密钥以供后续验证的 Decoder。除此之外，我们还将水印的嵌入层选择范围扩大，从原来仅仅嵌入被保护模型的卷积层，扩大到嵌入到被保护模型的每一个层中。

### 3.4.2 改进的水印嵌入算法

基于论文的水印嵌入算法，我们在此基础上进行了嵌入算法的改进。改进的算法过程如表 2所示。主要的改进点有以下几点：

- 嵌入时的位置选择，从之前仅仅由 EPH 的参数决定改变成现在由 SPH 的参数决定。
- 嵌入时的 XORPMV 的计算，从之前的 1*1 的块大小累计九次计算改变成现在 3*3 的块大小一次计算，可以大大提高嵌入时的效率。

通过这样的改变，使得我们在嵌入时的速度大大加快，且由于使用 Decoder 为嵌入位置的选择增

表 2: 水印嵌入算法

**Algorithm 1** 水印嵌入算法

**输入**：$f_t$：DNN-to-be-protected；$N$：$f_t$ 中的卷积核数量；Kernel1：Encoder 水印中将要被嵌入的所有卷积核；Kernel2：Decoder 本地密钥中的所有卷积核；$n1$：Kernel1 中的卷积核数量；$n2$：Kernel2 中的卷积核数量；bitmap：记录 $f_t$ 中水印的嵌入位置的位图；key：密钥

**输出**：$f_{wm}$：DNN-watermarked

**算法流程**：

bitmap = NULL;$f_{wm}$=$f_t$

**for** i in (1,n1) **do**

  position = $HAMC(key, XORPMV(kernel2_i) \bigoplus i)\%N$

  **while** $bitmap_{position}$ **do**

    $position = ++position\%N$

  **end while**

  $f_{wm} = EMBED(kernel1_i, position, f_{wm})$

$bitmap_{position} = 1$

++i

**end for**

**return** $f_{wm}$

XORPMV 函数是对一个卷积核的所有参数进行 XOR 计算，EMBED 是将 Encoder 一个卷积核嵌入到 $f_t$ 的特定位置，HMAC 是一个带密钥的加密哈希函数。

加了更多的随机因素，提高了水印的鲁棒性。此外，由于 Decoder 是模型所有者本地保存，从而使得水印的安全性也大大提高。

### 3.5 实验实现

实验的环境如表 3所示。

表 3: 实验环境

| | |
|---|---|
| **模型** | VGG 用作被保护的 DNN，水印神经网络架构为 encoder 和 decoder，encoder 部分有三个卷积层，decoder 部分也生成对应的三个卷积层。 |
| **数据集** | 使用了 MINIST 数据集和 CIFAR10 数据集，其中 MINIST 数据集用于训练水印神经网络，CIFAR10 数据集用于训练 VGG 神经网络。 |
| **平台** | 运行在学校的服务器上，操作系统为 ubuntu。 |

需要说明的是，在实验中，所有神经网络的训练过程都是基于 python 和 pytorch 框架完成，并将所有模型存储为.t7 后缀的模型文件。

### 3.5.1 生成水印

水印的生成实质上是一个神经网络的训练过程。经过训练，我们得到了 model_decoder_param.t7 和 model_encoder_param.t7 两个模型文件。

水印的生成过程可以用图 5来表示。


图 5: 水印生成流程图

在这个过程中，我们首先加载 MINIST 数据集，并将数据转化成 tensor 张量，在此基础上，我们将数据集转化成 DataLoader 的数据管道。

数据处理结束之后，定义模型 Encoder 和 Decoder，以 Encoder 为例，其定义如下：

```
class Encoder(nn.Module):
  def __init__(self):
  super(Encoder,self).__init__()
  self.encoder = nn.Sequential(
     nn.Conv2d(1, 20, 3, stride=2, padding=1),
     nn.ReLU(),
     nn.Conv2d(20, 32, 3, stride=2, padding=1),
     nn.ReLU(),
     nn.Conv2d(32, 64, 7)
  ).to(device)
```

除此之外，定义优化器为 Adam，并定义学习率的衰减策略为 Cosine Annealing，损失函数定义为均方损失函数 MSELoss。具体的定义如下：

```
optimizer = optim.Adam(model.parameters(),lr=args.lr)
scheduler = lr_scheduler.CosineAnnealingLR(optimizer,epochs, eta_min=1e-10)
criterion = nn.MSELoss()
```

在定义完以上参数之后，就开始训练模型。实验中，定义 epoch 为 100，因此需要进行 100 轮的训练。经过训练，我们得到 encoder 和 decoder 的参数空间，并在后续将 encoder 嵌入 VGG 模型中。

### 3.5.2 生成被保护的模型

这一步包含嵌入水印和训练被保护模型。

首先，会先定义 transform_train 和 transform_test，用来处理训练集数据和测试集数据。这里的数据处理操作包含对训练数据集图片的 **随机裁剪、水平翻转、转成 tensor 和标准化**，对测试集数据的**转成 tensor 和标准化操作**。transform_train 的定义如下：

```
 transform_train = transforms.Compose([
  transforms.RandomCrop(32, padding=4),
```

```
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
])
```

此后，进行 VGG 的初始化，其流程如图 6所示。

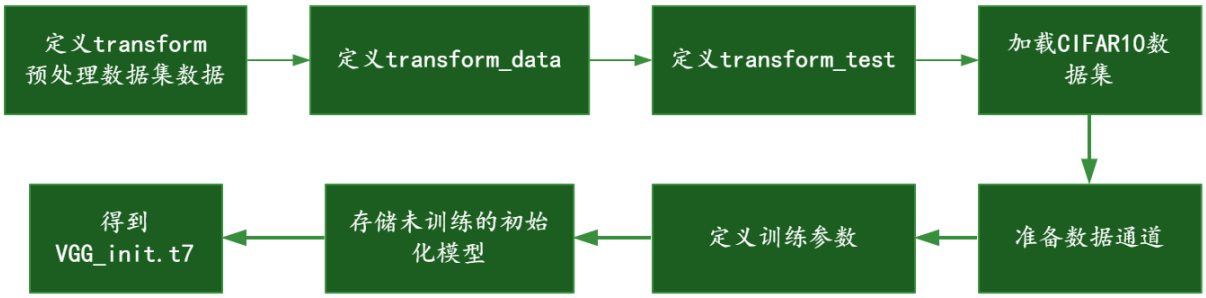


图 6: 生成 VGG 流程图

此时生成的 VGG 模型只是一个框架，并未进行训练，得到的模型存储在文件 VGG_init.t7 中。

之后，我们需要将上一步生成的水印模型的 encoder 部分嵌入到 VGG_init 模型中，并进行训练。这个过程可以用图 1来表示。


图 7: 水印嵌入流程图

下面将说明嵌入的关键算法，算法的步骤已经在注释中标出。

```
block_id += 1
# 遍历encoder的每一层的wight矩阵
for i,name1 in enumerate([key for key in encoder.keys() if 'weight' in key]):
    # 得到4维空间
    (in1,out1,h1,w1) = encoder[name1].size()
    # 展成一维
    len1 += in1*out1*h1*w1
    # 第一维
    for j in range(in1):
        # 第二维
        for k in range(out1): # embed block by block
            # 初始化message信息
            message = 0.0
            # 遍历weight矩阵的每一个元素，使用了所有的id
```

```
        for m in range(h1):
            for n in range(w1):
                # 和 encoder 对应的 decoder层
                nameo = str(4-int(name1[0]))+name1[1:]
                # 把 decoder 的每个1*1*1*1单元写入到message中，进行XOR操作
                message += ufuh[nameo][j, k, m, n]
        # 块信息写入message
        message *= block_id
        # 更新block_id
        block_id += 1
        # 对message进行哈希
        mac = hmac.new(bytes(seed, encoding='utf-8'), message.numpy(),
            hashlib.sha256)
        # 通过message的哈希值计算嵌入的位置
        pos = int(mac.hexdigest(), 16) % len2 # compute embed position

        # 处理嵌入的冲突
        for m in range(h1):
            for n in range(w1):
                while grad2[pos] == 0:
                    # 对vgg的参数空间长度取模
                    pos = (pos+1)%len2
                # encoder的weight先存储到vgg展成一维的等价张量tmp2中
                tmp2[pos]=encoder[name1][j,k,m,n]
```

在完成水印的嵌入之后，会对 VGG 模型进行训练，最终得到的 VGG 是训练好的模型了。此时，模型由于水印的嵌入，可以用于声明对模型的所有权；由于模型经过 CIFAR10 数据集的训练，因此也拥有很高的准确率。因而在保证了水印嵌入的同时，也完成了对模型的训练。

### 3.5.3 攻击模拟

实验中，我们主要对嵌入了水印的模型进行了四种攻击：

- 微调攻击
- 剪枝
- 内核截断攻击
- 函数等价攻击

具体的实现过程基本都是基于读取模型的参数，然后在此基础上，进行细微的调整。由于攻击过程的具体实现并不在版权保护的范围内，因此在这里不做过多的说明。

### 3.5.4 水印提取和验证

水印的提取是基于上一步中受到攻击的模型进行的。其工作流程可以用图 8来表示。


图 8: 水印的提取和验证流程图

在水印的提取过程进行之前，我们需要先将被函数等价攻击的模型进行恢复，然后在此基础上提取出我们的水印。水印的提取过程是水印嵌入过程的逆过程，因此在此就不再赘述水印的提取算法。

当水印被提取出来后，我们得到了 encoder 部分的参数，然后将其在 MINIST 数据集上进行测试，得到测试的结果，并将结果与阈值进行对比，判断存疑模型是否存在版权问题。

至此，实验的实现部分基本结束，下面将说明我们的实验结果。

## 四、 实验结果

### 4.1 数据集说明

为检测我们的水印对神经网络保护的性能，我们选择 VGG 模型作为待保护的神经网络模型，选择 CIFAR-10 作为模型训练、测试和验证的数据集。CIFAR-10 数据集是一个用于识别普适物体的小型数据集。一共包含 10 个类别的 3 通道 RGB 彩色图片：飞机（airplane）、汽车（automobile）、鸟类（bird）、猫（cat）、鹿（deer）、狗（dog）、蛙类（frog）、马（horse）、船（ship）和卡车（truck）。图片的尺寸为 32Œ32，数据集中一共有 50000 张训练图片和 10000 张测试图片。CIFAR-10 的图片样例如图 9所示。

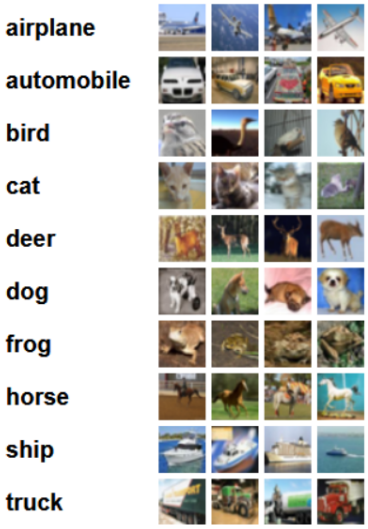


图 9: CIFAR 数据集

对嵌入的水印模型，我们选用的 Fashion-MNIST 数据集，同样是图片集合，但每个样本是 28 Œ 28 像素的灰度手写数字图片，相对于 CIFAR-10 图片尺寸较小，且仅为灰度。即水印模型的任务，同样是图片分类，而与待保护的神经网络模型相比，水印模型的结构复杂度更小，则所设置的任务难度也更小。无论是待保护的 VGG 模型，还是嵌入的水印模型，任务均是对图片进行分类，最终以验证损失值、预测准确率作为评价模型性能的指标。

### 4.2 基准性能

在正式开始测试前，我们需要设置基准性能表现。我们首先对没有进行过任何攻击和处理的已嵌入水印模型，进行所有权的验证，作为基准性能表现，之后对被攻击过的模型进行所有权验证时，结果越接近基线，则表示对该模型保护的效果越好。

基线结果如表 4所示：Original 表示原始已嵌入水印保护的 VGG 模型准确率；Suspect 表示我们想要检测的（怀疑已被攻击）模型，因为基线性能测试我们不对神经网络破坏，所以与原始准确值相等；Combine 表示从神经网络提取并组合得到的水印模型准确率，同样与原始水印模型准确率相同，因为暂未攻击。我们设置水印准确率阈值为 15%，在后续验证中，若提取水印的验证准确率能维持基线水平的 85%，即 $92.06\% * 85\% = 78.25$ ，则可声明对该神经网络的所有权。

表 4: 基线结果

| | VGG |
|---|---|
| original | 87.89% |
| Suspect | 87.89% |
| Combine | 92.06% |

完成对基准性能表现的设置，便可以进行水印性能的测试，本次项目中，我们从水印鲁棒性、隐蔽性、保真性 3 个方面，验证对水印嵌入算法、水印模型结构、模型训练过程进行改进的结果。

### 4.3 鲁棒性

#### 4.3.1 对抗精调（fine-tune）攻击的鲁棒性

站在攻击者的角度，当想要将待保护的神经网络模型为己所用时，可以通过精调（fine-tune），盗取性能良好的模型。我们假设攻击者仅可以获取测试数据集（由模型所有者提供，用来验证模型性能），并将测试数据集为了两部分作为精调的数据集，训练集和测试集（8：2）。同时，设置训练的相关超参数，学习率为 0.001，权值衰减率为 0.0005，使用随机梯度下降（Stochastic Gradient Descent，SGD）进行精调，依次在精调了 20、40、60、80、100epochs 后将模型保存。这里精调的参数设置，与初始训练模型的参数有差异（初始训练学习率为 0.01，权值衰减 0.0005），模拟了攻击者为精调获得更好模型效果的攻击想法。

对精调保存下来的模型进行所有权验证：提取水印并组合，测试水印准确率；同时测试精调后的 VGG 模型准确率，纵向对比结果如表 5所示。

表 5: 对抗精调（fine-tune）攻击的鲁棒性

| Finetune epochs | VGG - watermarked | Watermark Net |
|---|---|---|
| 0 | 87.89% | 92.06% |
| 20 | 87.88% | 91.98% |
| 40 | 87.89% | 91.19% |
| 60 | 87.86% | 91.16% |
| 80 | 88.03% | 91.12% |
| 100 | 88.18% | 90.12% |

对比 baseline 准确率，可以发现：随着精调轮次的增加，待保护的神经网络 VGG 模型正确率小范围内增长，即使提取得到的水印正确率下降，但整体得到的水印准确率仍维持在 90

以上效果，与我们对水印嵌入流程的改进相关，将水印嵌入模型后的训练阶段，模型任务均为图片分类，我们每隔 5epochs 同步更新嵌入的 encoder 参数和模型卷积核参数，重复多次将水印和模型融合，使得训练过程中，水印在参数级别上与待保护模型更不可区分、更融洽，因此精调模型，无法大幅度降低水印验证的准确度，仍能精准验证所有权。

#### 4.3.2 对抗剪枝（prune）的鲁棒性

受益于我们对水印嵌入和模型训练过程的优化，精调攻击无法从参数层面去除模型中的水印痕迹。因而攻击者往往还会采取针对参数的模型剪枝攻击。

对已嵌入水印的受保护模型，模拟实施参数剪枝攻击，攻击强度从 10% 到 90%，强度越大，模型中被去除的参数越多。我们模拟的剪枝攻击，采用的策略是去除所有参数中绝对值较小的部分参数，因为站在攻击者的角度，剪枝是为了在对原本模型性能影响较小的情况下去除模型中的水印，使模型所有权验证失败。因而，为了避免因剪枝大幅降低模型性能，剪枝攻击将去除所有参数中绝对值较小的参数，本次测试，依次剪枝攻击 10%、30%、50%、70%、90% 强度，对攻击后的模型进行准确率验证，同时提取水印验证所有权，纵向对比如表 6所示。

表 6: 对抗精调（fine-tune）攻击的鲁棒性

| Prune | VGG - watermarked | Watermark Net |
|---|---|---|
| 0 | 87.89% | 92.06% |
| 10 | 87.89% | 91.98% |
| 30 | 87.86% | 91.19% |
| 50 | 87.82% | 91.16% |
| 70 | 82.03% | 87.12% |
| 90 | 78.18% | 85.12% |

对比 baseline 准确率，可以发现：随着剪枝攻击强大的增加，待保护的神经网络 VGG 模型正确率在下降，提取得到的水印正确率也在下降，尤其是当剪枝攻击程度达 70% 及以上，且即使攻击者增大剪枝强度使得水印所有权验证失败，其得到的剪枝后的模型性能也非常差（甚至无法使用），但就测试结果而言，在验证阶段提取合并的水印准确率维持在可接受阈值范围内，仍可精准验证该模型的所有权。满足了初衷的想法：被尝试破坏水印痕迹的模型，性能会下降，但水印仍能较大程度地保存，精准验证所有权，对抗剪枝攻击鲁棒性较高。

获得以上效果，与所嵌入的水印参数数量有关，我们嵌入水印的过程是在参数层面对待保护模型留下水印痕迹，待保护模型通常规模较大、结构较复杂、参数量较多，而我们嵌入的水印模型参数，规模小、参数少，所嵌入修改的参数量，仅占 VGG 模型参数的极小一部分，仅为 3.44%，因此剪枝攻击参数去除，对于嵌入在模型中的水印参数，有较小的可能受到影响，故水印能在一定程度具有抵抗剪枝攻击的鲁棒性。

#### 4.3.3 抵抗内核截断攻击的鲁棒性

除了通过剪枝尝试去除水印，攻击者还可以通过去除卷积层内核进行攻击。在模拟内核攻击的测试中，我们控制内核截断率在 6.75% 左右，避免攻击后的模型性能大幅度下降。在对攻击后的模型进

行所有权验证之前，我们通过上述介绍的余弦相似度恢复内核被攻击的可疑模型，并将 0 填充给缺失的内核。对经过恢复的模型抽取水印，验证其准确率，结果如表 7，可见经过恢复操作能保持所有精准验证。对于内核补充（kernels supplement）攻击，与内核截断（kernels cutoffs）攻击相似，是以相似的方式改变内核的结构和数量。

表 7: 抵抗内核截断攻击的鲁棒性

| VGG | VGG –watermarked accuracy | Watermark Net accuracy |
|---|---|---|
| Original | 87.89% | 92.06% |
| Cut off | 82.81% | |
| Restore | 82.85% | 91.95% |

#### 4.3.4 水印鲁棒性分析

得益于水印嵌入的隐蔽性，以及所嵌入水印参数仅占待保护模型参数的极小部分（水印参数量占 VGG 模型参数的 3.44%），使得通过剪枝和精调的方式很难破坏模型已嵌入的水印。除此之外，水印对于剪枝和精调攻击的鲁棒性，还与深度神经网络模型的参数冗余问题有关，通常仅有一部分参数（如模型中拥有较大绝对值的 50% 参数），会对该神经网络性能有很大影响，而其他的参数对于模型解决应用问题的贡献甚少，常被当作冗余参数。因此，对模型的重新训练主要影响已嵌入水印 DNN 模型中的这些不太重要的参数。只要水印多数的显著参数能稳定保存在模型训练中，模型所有权依旧能精准认证。

### 4.4 对抗多种攻击

在盗取一个深度神经网络模型后，攻击者可能会想方设法破坏模型中所嵌入的水印，如通过微调、剪枝、等价攻击（参数调整，通道调整）等直接的破坏。模拟多种手段攻击待保护模型中，我们对已嵌入水印的模型，首先进行微调攻击（使用相数据集的测试集，微调 100epochs），其次进行不同程度的剪枝（依次剪枝 10%，20%，30%，40%，50%），最后进行内核截断和乱序。

完成攻击后的模型，如果直接提取水印进行验证，验证准确率很低，但如果我们首先对模型进行恢复 restore，使用到的方法是此前提到的余弦相似度和 SVD 方法:，对比被攻击模型和攻击前模型的参数相似度，来按照原始模型进行恢复。

测试被攻击后的模型准确率，以及对比恢复前后水印提取验证的准确率，如表 8所示。可以看到，当经过一系列攻击，剪枝 30%，提取水印的验证效果依旧较好；当剪枝 40%，提取水印验证正确率仅 50% 左右，但被盗取的模型性能也大打折扣，正确率降至 50% 以下，对于攻击者而言模型也已无法使用。

### 4.5 隐蔽性

在 baseline 指标的介绍中，我们水印模型的 encoder 仅占待保护的深度神经网络模型的小部分（水印模型嵌入的参数，仅占 VGG 参数的 3.44%），因此较难被研究和察觉到。纵使在 DNN 模型水印其嵌入位置，和水印模型的内核参数及分块信息，这两者之间存在一个哈希处理，但对嵌入位置的哈希加密，我们使用到 seed，让嵌入位置更难被察觉。同时，在寻找嵌入位置中，我们使用到 decoder 信息，decoder 在训练完成后，是保存在模型所有者手中的。此外，我们从模型的参数分布，来比较有无水印

表 8: 对抗多种攻击

| Prune rate（finetune） | VGG–watermarked Accuracy | Watermark Net Accuracy Direct | Watermark Net Accuracy Restore |
|---|---|---|---|
| 10% | 84.87% | 11.23% | 90.45% |
| 20% | 84.83% | 10.35% | 87.56% |
| 30% | 83.93% | 10.21% | 82.54% |
| 40% | 47.20% | 10.13% | 50.12% |
| 50% | 9.12% | 9.01% | 10.19% |

嵌入的模型之间的差异，如图 10所示，对于 VGG 模型而言，嵌入水印与否对模型整体的参数分布干扰很微小，即从参数的统计分布上，水印嵌入较为隐蔽。

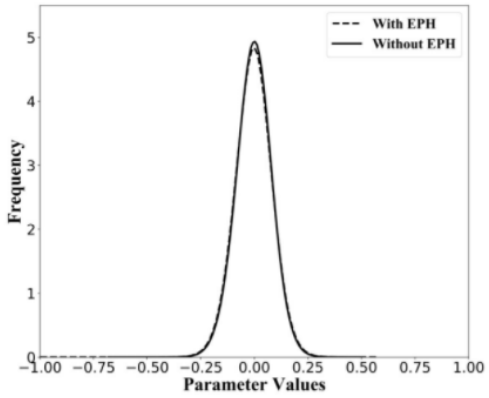


图 10: 隐蔽性

### 4.6 保真性

理想状态下，我们嵌入的水印，要对载体模型的性能有很小的干扰，为验证，我们在同一条件下，分别测试 VGG 模型嵌入水印和未嵌入水印完成训练后的两种情况下的正确率，二者之间的差别并不大，可说明水印嵌入对模型性能扰动较小。保真性，可能与嵌入水印参数仅占载体模型参数较小部分有关；且嵌入的水印参与了模型的完整训练，每隔一段时间，又更新嵌入的信息和位置，即水印参数和待保护模型的参数同步更新，使得水印在参数层面上与模型参数，结合更紧密。

表 9: 保真性结果

| | VGG |
|---|---|
| Without watermark | 87.25% |
| With watermark | 87.89% |
| Performance change | +0.71% |

## 五、 结论

这次实验我们尝试了神经网络的版权保护，在复现原论文的基础上，对水印模型结构、嵌入算法进行了改进。以往出现的神经网络版权保护方法，都只能抵抗较低级的独立的攻击手段，而当各种攻击方法有机组合或同时施展多种攻击时却力不从心。而我们所使用的方法，可以抵抗多种或多种攻击组成的综合攻击。而我们对水印模型的结构进行的改进，使得水印的鲁棒性有所提高；对嵌入算法的改进，使得嵌入的效率和水印的安全性也较之以往有了较明显的提高。

## 六、 人员分工

## 参考文献

[1] Lv P, Li P, Zhang S, et al. HufuNet: Embedding the Left Piece as Watermark and Keeping the Right Piece for Ownership Verification in Deep Neural Networks[J]. arXiv preprint arXiv:2103.13628, 2021.

[2] Zheng Li, Chengyu Hu, Yang Zhang, and Shanqing Guo. 2019. How to Prove Your Model Belongs to You: A Blind-Watermark Based Framework to Protect Intellectual Property of DNN. In Proceedings of the 35th Annual Computer Security Applications Conference (San Juan, Puerto Rico, USA) (ACSAC ' 19). Association for Computing Machinery, New York, NY, USA, 126–137. https://doi.org/10. 1145/3359789.3359801

[3] Microsoft. 2021. Microsoft HMAC-SHA256 documentation.

[4] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin' ichi Satoh. 2017. Embedding Watermarks into Deep Neural Networks. In Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval (Bucharest, Romania) (ICMR ' 17). Association for Computing Machinery, New York, NY, USA, 269–277. https://doi.org/10.1145/3078971.3078974

[5] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. 2018. Detecting backdoor attacks on deep neural networks by activation clustering. arXiv preprint arXiv:1811.03728.

[6] Lixin Fan, Kam Woh Ng, and Chee Seng Chan. 2019. Rethinking Deep Neural Network Ownership Verification: Embedding Passports to Defeat Ambiguity Attacks. In Advances in Neural Information Processing Systems, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc., 4714–4723.

[7] Jia Guo and Miodrag Potkonjak. 2018. Watermarking Deep Neural Networks for Embedded Systems. In Proceedings of the International Conference on ComputerAided Design (San Diego, California) (ICCAD ' 18).

[8] Jialong Zhang, Zhongshu Gu, Jiyong Jang, Hui Wu, Marc Ph. Stoecklin, Heqing Huang, and Ian Molloy. 2018. Protecting Intellectual Property of Deep Neural Networks with Watermarking.

[9] Erwan Le Merrer, Patrick Perez, and Gilles Trédan. 2019. Adversarial frontier stitching for remote neural network watermarking.

# 教师评语评分

评语:

评　　分:

评 阅 人:

评阅时间: