一种针对多核神经网络处理器的窃取攻击

高成思[1,2], 陈维伟[1,2], 王 颖[1]

[1] 中国科学院计算技术研究所, 北京 中国 100190
[2] 中国科学院大学, 北京 中国 100049

摘要 随着神经网络的广泛应用, 它自身的安全问题也成为了一个重要的研究课题。将神经网络部署到神经网络处理器上运行是提高能效比的有效方法, 但同时也引入了一些新的安全问题, 比如侧信道信息泄露, 本文以多核 CNN 处理器为基础, 利用时间和内存侧信道信息, 提出了一种针对多核 CNN 处理器的用户算法信息窃取攻击方法, 经过试验证明了攻击的有效性, 并针对多核神经网络处理器在时间和内存侧信道方面的脆弱性, 提出了有效的防御手段, 对如何保护神经网络处理器的安全提供了一定的参考意义。

关键词 神经网络; CNN 处理器; 多核; 侧信道; 模型窃取
中图法分类号 TP309 DOI 号 10.19363/J.cnki.cn10-1380/tn.2020.05.03

An Information-leakage Threat Case for Multi-core Neural Network Processor

GAO Chengsi[1,2], CHEN Weiwei[1,2], WANG Ying[1]

[1] Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China
[2] University of Chinese Academy of Sciences, Beijing 100049, China

Abstract With the widespread application of neural networks, its own security issues have also become an important research topic. Deploying a neural network to a neural network accelerator is an effective method to improve energy-efficiency, but it also introduces some new security issues, such as side-channel information leakage. Based on multi-core CNN accelerator, we proposed a model extraction attack by exploiting timing and memory side-channel information leakage. The results of the experiments demonstrate the effectiveness of the attack. Then we proposed effective defense methods for the vulnerability of multi-core neural network accelerators in terms of timing and memory side-channels. It provides some reference for how to protect the safety of neural network accelerators.

Key words neural network; convolution neural network accelerator; multi-core; side-channel; model extraction attack

1 引言

近年来, 随着硬件计算能力的发展, 神经网络已经广泛应用在了众多不同的领域中, 比如图像识别[1-2], 语音识别[3]和恶意软件检测[4]等。在神经网络的广泛应用的同时, 神经网络自身的安全问题也变得十分的紧急和重要, 自从 2014 年, Szegedy 等人[5]提出对抗样本的概念, 神经网络的安全性问题引起了学术界广泛的关注, 研究者对神经网络在训练、预测阶段以及使用的训练数据和模型等方面的安全性进行了深入的研究, 目前神经网络在应用中面临的主要攻击威胁有 4 种[6], 包括模型反演攻击(Model inversion attacks), 投毒攻击(Poisoning attack), 模型窃取攻击(Model extraction attack)以及对抗攻击(Adversarial attack), 在如今机器学习发展成为一种服务(Machine learning as a service, MLaaS)的趋势下, 模型窃取攻击带来的威胁也变得更加突出: (1)对于那些依赖神经网络的运行结果进行决策或者提供服务的公司, 他们所使用的神经网络模型往往需要公司花费不菲的财力和物力去构建, 比如金融公司, 在这种情况下, 模型本身就成为了公司知识产权的一部分, 所以需要保证模型自身的机密性; (2)当攻击者在知道了网络模型的前提下, 更容易进一步发动其他的隐私和安全攻击, 比如模型反演攻击[7-8]和成

通讯作者: 王颖, 博士, 副研究员, Email: wangying2009@ict.ac.cn。
本课题得到国家自然基金(No. 61876173)和中国科学院战略性先导专项项目(No. XDC05030201)资助。
收稿日期: 2020-02-02; 修改日期: 2020-04-24; 定稿日期: 2020-04-29

员推理攻击[9-10](Membership inference attack), 以此来反推出模型的输入数据和训练数据, 此外也有助于对抗样本发动更加有效的攻击[11-12], 使得对抗样本更容易逃避分类器的检测, 从而使模型最终得到错误的预测结果, 这对于有些领域来说, 比如自动驾驶, 这种安全问题是非常严重的, 因此, 如何防御模型窃取攻击是一个非常有意义且重要的研究方向。

为了加快神经网络在设备上的运算速度, 研究者们设计出了比 GPU(Graphics processing unit)更具能效比、更快的神经网络处理器, 比如 DaDianNao[13] 和 TPU[14](Tensor processing unit)等, 由于卷积神经网络(Convolutional neural network, CNN)在图像识别与分割, 物体检测以及自然语言处理等方面的广泛应用[15], 学术界和工业界针对卷积神经网络的处理器的也进行了深入的研究, 并提出了一系列不同架构的处理器[13-14,16], 在将神经网络部署到这些处理器上计算的过程中, 神经网络处理器也暴露出了一些安全问题, 攻击者可以通过侧信道信息(Side-channel)完成模型窃取攻击[17], 此外, 随着不断增长的算力需求, 多核神经网络处理器架构也逐渐变成了一种主流趋势, 比如应用于云上的 Google Cloud TPU[18]或者手机端的芯片[19], 在这种架构中, 不同的处理器上会部署不同的网络模型, 因此, 如何在这种多核多模型的场景中保护每个模型的机密性是一个非常重要的问题。

不同于以前的工作[17,20-21], 本文以多核 CNN 处理器为基础, 研究了多核神经网络处理器架构面临侧信道攻击时的脆弱性, 提出了一种更实际的模型窃取攻击方式, 并针对这种攻击方式提出了几种有效的防御机制。本文主要的贡献点如下:

1) 分析了多核CNN处理器模型, 评估了该架构在抗内存和时间侧信道攻击的脆弱性, 并对其展开了侧信道攻击;

2) 结合多层感知机(Multilayer Perceptron, MLP)和长短期记忆网络(Long Short-Term Memory, LSTM), 本文设计了一种针对多核 CNN 处理器的攻击方法, 完成了对模型结构的窃取, 该攻击方法利用了多核 CNN 处理器在进行推理时的内存和时间的侧信道信息, 本文详细介绍了该攻击方法的流程和构建方式, 并通过实验证明了攻击的有效性;

3) 针对多核 CNN 处理器在侧信道上面临的攻击, 提出了一些有效的防御手段。

2 背景介绍

2.1 侧信道攻击

由于侧信道攻击利用了攻击目标在物理实现中无可避免泄露出的信息, 比如时间信息、缓存(cache)消耗、电磁泄露等信息, 相较于利用暴力破解、算法和程序中的缺陷, 它在黑盒攻击(Black-box attack)等攻击方式中更加的有效, 因此也给设备的安全带来了严重的威胁, 比如著名的 CPU 漏洞 Spectre[22]和 Meltdown[23], 它们就利用了 CPU 的缓存信息来进行侧信道攻击, 在 GPU 中也存在同样的安全威胁[24]; 随着神经网络处理器的广泛应用, 对于处理器本身安全问题的研究也变成了一个重要研究方向, 近年来, 研究者也针对神经网络处理器提出了多种侧信道攻击的方法[17,20], 通过对神经网络处理器侧信道攻击的研究, 研究者从而提出更加有效的防御手段。

2.2 多核 CNN 处理器架构

相对于 CPU 和 GPU 的架构而言, 神经网络处理器是一种更加专用的芯片, 以 CNN 处理器为例, 一个典型的 CNN 处理器的结构如图 1 所示, 其中的主

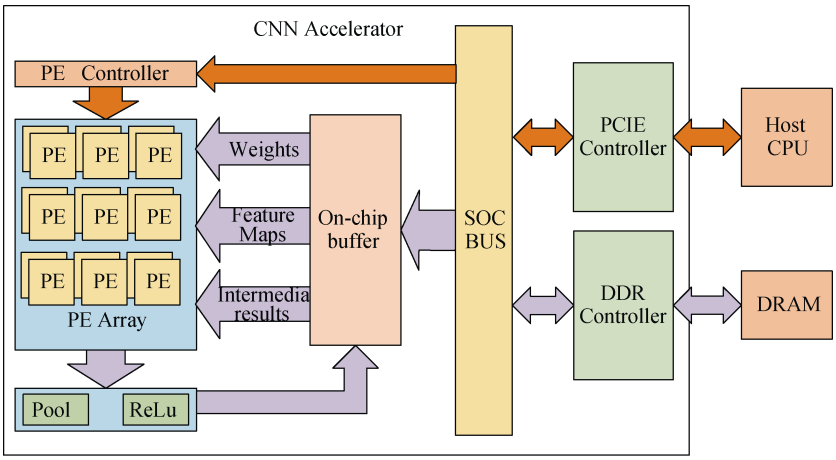

图 1 一个典型的 CNN 处理器结构

Figure 1 A Typical CNN Accelerator Architecture

要组成部分为 PE 阵列(Processing Element Array), 和片上缓存(On-chip buffer), PE 阵列主要用来计算 CNN 中的卷积操作, 片上缓存用来存放卷积计算所需的输入、输出数据(feature maps), 权重值(weights)以及在计算过程中产生的部分和等中间结果, 由于 CNN 网络的 feature maps 和 weights 通常都比较大, 所以将所有的 feature maps、weights 以及中间结果都存放在片上缓存中是不现实的, 因此, CNN 处理器通常都会将 feature maps 和 weights 存放在片外的内存中, 然后按照计算的需求从内存中取出数据。多核 CNN 处理器是以单个 CNN 处理器为基础, 通过总线将他们连接起来组成一个拓扑结构, 比如常见的 mesh 结构[25], 然后将不同的 CNN 模型部署到这些不同的 CNN 处理器上同时运行, 多个 CNN 处理器之间会存在硬件资源竞争的情况, 比如在共享同一个内存时, 会对数据总线进行竞争, 一个典型的多核 CNN 处理器架构如图 2 所示。

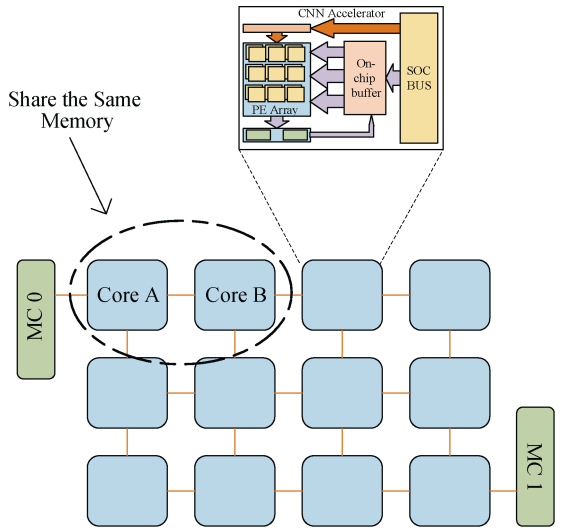

图 2 一个多核 CNN 处理器架构

Figure 2 A Multi-core CNN Accelerator Architecture

2.3 神经网络的隐私信息

对于一个给定的目标神经网络 f_{target}, 它的主要隐私信息包括以下几个方面:

1) **网络模型:** 包括了网络的深度, 层的类型和结构(卷积核大小, 输入尺寸, 卷积步长等)以及网络的拓扑结构。

2) **模型参数:** 包括权重值(weights), 偏置值(bias)等。

3) **训练数据:** 指的在训练目标网络时训练集中的数据, 对于某些领域而言, 比如医疗和金融行业, 训练数据会包括很多用户的隐私信息。

在所有的这些隐私信息中, 网络模型是最重要的信息, 第一, 正如 1 中介绍, 它本身可能就代表了一种知识产权; 第二, 通过网络模型攻击者可以推测出模型参数和训练数据[9,26-27]; 第三, 已知模型结构, 攻击者也更容易成功完成对抗攻击[11-12], 因此防止网络模型窃取攻击是一个非常重要的问题。

3 多核 CNN 处理器的侧信道分析

随着 Google 推出 Cloud TPU[18], 寒武纪推出 DaDianNao[13]以及高通等芯片厂商推出具有多核神经网络处理器的手机芯片[19], 多核神经网络处理器已成为一种主流趋势; 在文献[13]中, 用户可以将网络部署到一个或多个 node 上进行训练和推理, 多个 node 之间通过 HyperTransport 连接起来组成了 2D mesh 结构, 当多个 node 连接在一起需要同时访存时, 则会对同一个内存发起请求, 比如图 2 中的 Core A 和 Core B 就会因为同时请求 MC 0, 互相干扰访存的延时, 而这种干扰则会泄露 node 上正在计算的网络信息, 如果 Core A 和 Core B 上的网络属于不同的用户, 那么攻击者就可以通过访存延时信息推测出另一个核上的网络模型。

对于一个 CNN 处理器来说, 它完成的一次推理计算, 可以看作是由多个子任务的计算组成, 每一个子任务对应的是 CNN 网络的一层, 而完成每一个子任务所需要的步骤一共分为三步, 第一步, 从内存中加载该层所需的 feature map 和 weight; 第二步, 按照处理器的指令进行运算; 第三步, 将该层的计算结果写回到内存; 重复以上步骤直到完成整个 CNN 网络的推理计算。

前文提到, 多核 CNN 处理器同时计算时会在硬件资源上存在竞争关系, 以图 2 的多核 CNN 处理器架构为例, Core A 和 Core B 计算时所需要的 feature map 和 weights 都存放在同一个片外内存 MC 0 中, 当这两个处理器同时进行计算时, 会向同一个内存发起请求, 会造成带宽(Bandwidth)资源的竞争, 从而影响到处理器在加载和写回数据时的延时, 而且, 当处理器在计算不同参数的不同层时, 对另一个处理器所造成的延时会不同, 举一个例子, 假设在 Core A 上部署三次不同参数的卷积层, 而 Core B 每次都同时计算 5 层相同参数的卷积层, 并且保证每一次 Core B 都在 Core A 之后完成计算, 计算完成后, 得到每一次 Core B 上每层的完成时间如图 3 所示(显示的为归一化时间, CNN 层参数的显示格式为 kernel 个数 x kernel 高 x kernel 宽), 可以看到, 随着 Core A 每次计算的参数不同, Core B 每次计算时也会受到不同的影响, 导致每层的完成时间发生变化, 反过

来, Core B 每次计算完成的时间变化, 也可以用来反推出 Core A 上每次计算的卷积层的参数, 此外, 当 Core A 上计算不同类型的网络层时, 比如 Add 层、Pooling 层, Core B 每次的完成时间也会受到不同的影响, 如图 4 所示(AP 代表 AvgPooling 层, 后面参数为 kernel 大小 x stride), 这表明了在多核 CNN 处理器架构下, 当不同核在内存资源上存在竞争关系时, 可以通过一个 CNN 处理器的计算完成时间反推出其他核上的计算参数, 同时也证明了多核 CNN 处理器架构在面对时间和内存侧信道攻击时的脆弱性, 当多核 CNN 处理器中某个核上计算的网络是攻击者精心构造时, 如图 3 中 Core B 上的网络, 那么就可以通过 Core B 上每层的计算完成时间推测出在 Core A 上计算的网络模型。

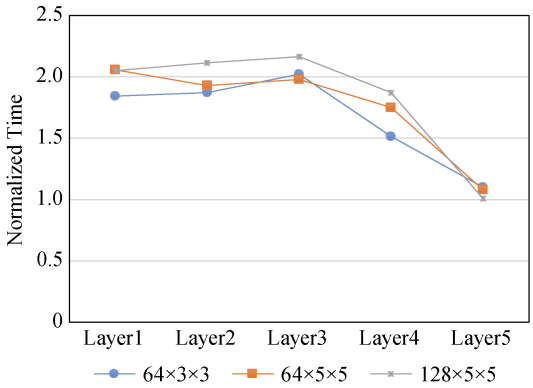

图 3 Core B 每次计算的完成时间

Figure 3 Completion Time of Each Round of Core B

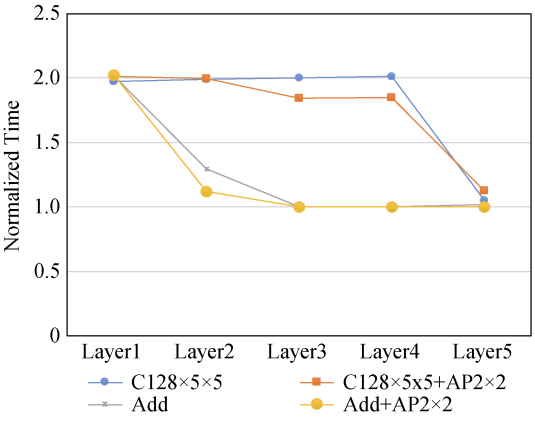

图 4 Core A 计算不同类型层时 Core B 的完成时间

Figure 4 Completion Time of Each Round of Core B

4 模型窃取威胁模型

在本文中, 我们研究的平台是多核 CNN 处理器, 具体的架构如图 2 所示, 其中的每个处理器的架构如图 1 所示, 每个 CNN 处理器上运行不同的模型, 而每个模型可能是来自于不同用户, 或者是不同的应用, 因此需要保证每个模型各自的机密性; 多个 CNN 处理器之间会共享同一个内存资源, 比如图 2 中的 Core A 和 Core B 计算时都需要从同一个内存取出数据, 会互相抢占带宽资源。

我们不假设攻击者可以通过物理手段获得多核 CNN 处理器架构的侧信道信息, 比如当多核 CNN 处理器位于云端时, 攻击者是没有办法通过总线监听等方式获取 CNN 处理器的内存和时间的侧信道信息, 我们假设攻击者只能控制在某个 CNN 处理器上部署的 CNN 网络模型, 即攻击者可以构造一个精心设计的网络模型, 我们称为毒化网络(以下简称 f_{poison}), 并将其部署到某个 CNN 处理器上进行计算, 此外攻击者可以获得该 f_{poison} 在 CNN 处理器上每一层的完成时间, 这个可以很容易的通过平台提供的一些性能评估接口或者开源的框架中已有的函数来获得, 比如 TensorFlow[28]中的 Profiler 模块或者 PyTorch[29]中已有的函数。

此外, 我们假设攻击者可以在多核 CNN 处理器架构中多次运行以获得训练数据, 包括只运行 f_{poison} 以及 f_{poison} 和多个样本网络同时运行, 这个可以通过在云上部署时申请相应的硬件资源, 或者攻击者可以通过离线搭建相同的硬件平台完成, 保证 f_{poison} 在训练时的硬件平台和攻击时的硬件平台保持一致。

5 攻击方法

在本文中, 给定目标网络f_{target}, 攻击者的目标为窃取出 f_{target} 的模型结构, 即网络的深度 *depth*, 层的结构序列 $TS=\{ts_0, ts_1,\cdots, ts_n\}$以及网络的拓扑结构; 整个攻击的流程是利用 f_{poison} 获得的侧信道信息确定 f_{target} 的结构序列 S, 然后在序列 S 上对 f_{target} 的拓扑结构进行重建, 这一过程可以看作是一个搜索问题[30]。

本文提出的攻击方法流程如图 5 所示, 可以分为 3 个阶段:

1)训练阶段: 在这个阶段中, 攻击者首先会构造一个精心设计的 f_{poison}, 然后生成大量不同结构的样本网络, 把每一个样本网络都当作潜在的攻击目标, 依次将每一个样本和f_{poison}共同部署到CNN多核处理器上共同计算, 并且记录下必要的训练数据(5.1.3 节中介绍), 用来训练后续的深度预测网络 F_{dep}(图 5 中的①)和层结构估计网络 F_{stru}(图 5 中的②);

2)攻击阶段: 攻击者将设计的 f_{poison} 部署到目标平台上, 让某个 CNN 处理器完成 f_{poison} 的计算, 等待处理器完成计算后, 攻击者获得 f_{poison} 每一层的计算完成时间组成的时间序列 $T=\{t_0, t_1,\cdots, t_n\}$, 并将 T 进行相应的预处理;

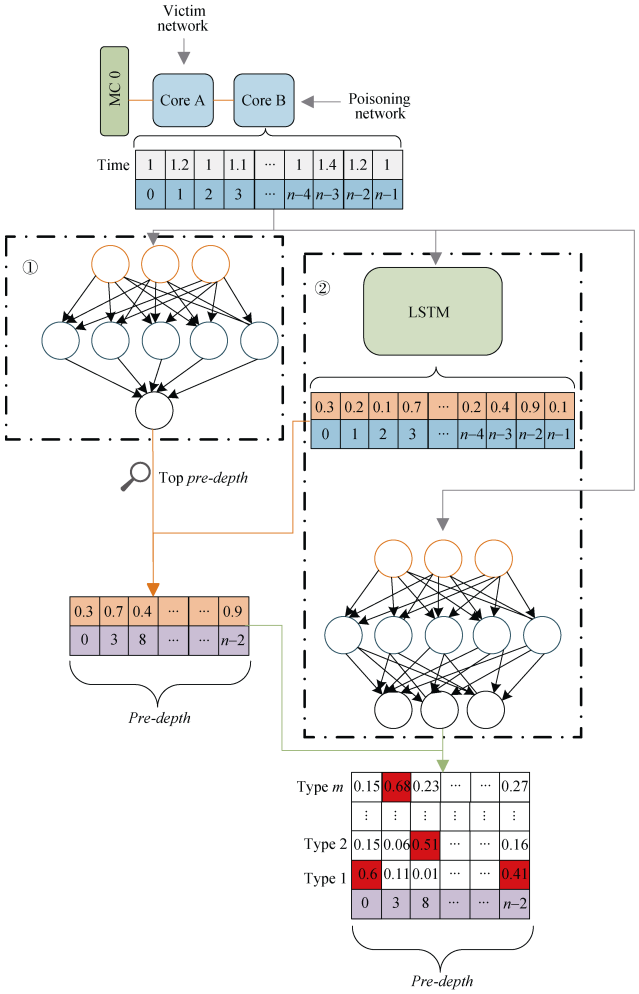

图 5 模型结构窃取攻击流程图

Figure 5 Model Architect Extraction Attack Flow

3)重建阶段: 攻击者将经过预处理之后的序列分别送入到已经训练好的 F_{dep} 和 F_{stru} 中, 得到关于 f_{target} 的层结构序列 $S=\{s_0, s_1,\cdots, s_n\}$, 并最终根据 Add 层的一些特性重建 f_{target} 的拓扑结构。

综上, 整个攻击流程可以描述为: 给定 f_{target} 和他的结构序列 S, 通过运行 f_{poison} 获得时间序列 T, 将 T 送入 F_{dep} 得到预测的网络深度 *pre-depth*, 再将 T 和 *pre-depth* 送入 F_{stru} 中得到层结构序列 S, 最终在 S 上进行搜索, 得到重建出的 f_{target} 的结构。

5.1 训练阶段

5.1.1 设计毒化网络

由前文中的第 3 节可知, 通过 f_{poison} 的运行时间, 攻击者可以反推出其他处理器上正在计算的层的参数, 为了反推出 f_{target} 所有层的结构参数, 攻击者需要构建一个多层的 f_{poison}; 由于多核 CNN 处理器只在内存带宽资源上存在竞争关系, 在计算资源上不存在竞争关系, 因此, 当 f_{poison} 和 f_{target} 在多核平台中同时运行时, f_{poison} 每一层花费在 CNN 处理器 PE 阵列中的计算时间是不会受到影响的, 只有在对内存进行访问读写时才会受到 f_{target} 的影响, 所以 f_{poison} 每一层的访存延时才能反映出当前 f_{target} 的结构参数, 为了尽可能的得到 f_{target} 运行时对 f_{poison} 运行的影响, 攻击者可以设计这样一种卷积层: 在计算这一层时, CNN 处理器主要花费的时间都是在访存操作上, 即加载计算所需的数据和写回结果, 而处理器上 PE 阵列进行卷积花费的时间只占很少一部分, 通常来说卷积层上需要进行的乘加操作数和该层输出的数据量成正比, 所以 CNN 处理器计算该卷积层主要花费的时间都在加载数据上, 一个典型的 f_{poison} 的设计方案可以是这样: 将该卷积层的 input feature map 的高和宽设置为一个适当的值, 将卷积核(kernel)大小设定为一个较小值, 例如 1, kernel 的步长(stride)设置为一个很大的值, 比如就为 input feature map 的高或宽, 这样就可以使得在 CNN 处理器上计算该层时, 绝大部分时间都是访存消耗的时间; 换句话说, 可以将 f_{poison} 每一层的完成时间都看作是对多核 CNN 处理器平台带宽拥塞情况的一次采样, 因此, 如果将 f_{poison} 的 input feature map 大小设计的越小, 也就代表采样的频率也就越高, 使后续的预测结果更加的准确。

5.1.2 生成样本网络

为了反推出 f_{target} 的模型结构, 攻击者需要将 f_{poison} 每一层的完成时间组成的序列 T 送入到 F_{dep} 和 F_{stru} 中, 因此攻击者需要先对这两个网络进行训练, 这就需要大量的样本网络, 为了生成贴近于真实的 CNN 网络, 攻击者可以首先定义一组层结构参数的范围, 首先, 每一层的类型可以分为 Add 层或者卷积层, 而每一个卷积层的每一个结构参数又可以定义一组范围, 比如 kernel 大小范围为{3, 5, 7},kernel 的个数范围为{64, 128, 256}等,而这些参数也是现有网络中常用的层结构参数[31], 此外, 由于 CNN 处理器的架构, Pooling 层和激活层都通常跟在上一层操作之后完成, 而不需要格外访问内存进行数据搬运,

所以可以将它们分别看作卷积层和 Add 层的一个结构参数; Add 层代表了目前网络中常用的残差结构[32], 在本文中默认采用的残差结构都类似于文献[32]中的结构, 批量标准化(Batch Normalization, BN)层一般都会融合到它的上一层卷积层中, 所以在这不作为单独的结构参数, 此外, 由于 CNN 网络多用于分类和识别任务, 典型的结构[32]是最后一层为全连接层(Fully Connected Layers, FC), 因此攻击者也可以假设样本网络的最后一层为 FC; 最后得到样本网络所有层结构参数的范围如表 1 所示, 攻击将这些结构参数随机组合, 将得到的所有有效的组合结

表 1 样本网络所有层结构参数范围

Table 1 Parameter Range for the Layer Architecture of Sample Network

层结构参数类别	参数范围			
Layer type	Conv	Add	Fc	AvgPooling
Kernel size	{3, 5, 7}	N	D	{2}
Kernel number	{64, 128, 256}	N	D	N
Pad	{1, 2, 3}	N	{0}	{0}
Stride	{1}	N	{1}	{2}
ReLu	{True, False}			

注: N 表示该层不需要这个参数, D 表示参数会根据上一层的结果自动生成

果组成一个集合, 攻击者在构建样本网络时, 该网络的每一层都从这个集合中采用随机采样的方式选出, 最后组合起来形成一个样本网络, 由于采用了随机采样的方式选取, 在构建样本网络时可能会出现结构不正确的情况, 比如 Add 层的两个输入数据在深度上不匹配, 那么在生成样本时就要去除掉这种样本。

5.1.3 训练深度预测网络和层结构估计网络

攻击者采用了两个网络模型来重建 f_{target} 的结构, 分别为 F_{dep} 和 F_{stru}, 这两个网络模型分别使用了 MLP 和 LSTM+MLP。

首先攻击者将依次从样本网络集中取出网络, 并和构造的 f_{poison} 同时部署在多核 CNN 处理器上进行计算, 并且记录下执行的结果, 用于后续训练, 需要记录的信息如表 2 所示; 同 GPU 的执行方式一样[21], CNN 卷积处理器也是顺序的执行每一层, 遇到有分支的情况也是完成一个分支之后再继续完成另一个分支, 因此 F_{dep} 得到的预测结果 *pre-depth* 的是在 CNN 处理器上串行执行网络的层数; F_{dep} 为一个 MLP, 在训练时, 输入为 f_{poison} 每层完成时间的序列 T, 标签值(label)为样本网络在 CNN 处理器上串行执行的网络层数, 损失函数(Loss Function)为两者的均方误差。

表 2 训练深度预测和层结构估计网络所需的信息

Table 2 Data need to train Depth-Prediction-Network and Layer-Structure-Estimation network

	深度预测网络	层结构估计网络
毒化网络时间序列	Y	Y
样本网络深度	Y	Y
样本网络每层结构	N	Y

注: Y 表示需要, N 表示不需要

F_{stru} 采用了 LSTM 和 MLP 的组合, 由前文 5.1.1 小节可知, 在 f_{poison} 完成了多层网络的计算时, f_{target} 可能只完成了一层网络的计算, 为了估计 f_{target} 每一层的结构, 攻击者可以采用两个步骤, 1)利用 LSTM 对 f_{poison} 的时间序列进行切分, 分割出 f_{target} 完成一层网络对应的 f_{poison} 时间序列; 2)再利用 MLP 对时间序列进行分类, 得到每一层结构参数的估计; 对 f_{poison} 时间序列的切分可以看作是一个序列预测问题, f_{poison} 的每一层的完成时间看作 LSTM 每一个时间步(time step)的输入, 而 LSTM 在每一个 time step 的输出, 表示该 f_{poison} 在完成这一层网络计算时, f_{target} 在该时间内完成某一层计算的概率, 这样就完成了对 f_{poison} 时间序列的切割, 将 LSTM 所有 step 的输出结果按照概率大小取出前 *pre-depth* 个, 并记录下它们在原始序列中的位置; 然后将序列 T 送入到 MLP 中进行计算, 假设 5.12 小节中样本网络层结构参数的所有组合为 m, 那么 F_{stru} 中 MLP 的输出长度为 $m * length(T)$, 代表在 f_{poison} 完成每一层计算时, f_{target} 正在运行的层的结构参数的概率分布, 再按照 LSTM 中记录的 *pre-depth* 个最有可能的位置, 从 MLP 的结果中取出对应位置的值, 组成的结果就是对 f_{target} 每一层结构估计值的序列, 在训练 F_{stru} 时, 可以分成两步, 先对 LSTM 进行训练, 然后在固定 LSTM, 对 MLP 进行训练, 在训练 MLP 时, 将其当作一个多标签分类问题来训练。

5.2 攻击阶段

在这一阶段, 攻击者将设计的 f_{poison} 部署到多核 CNN 处理器上的某个核上, 等待处理器完成所有层的计算, 然后获得 f_{poison} 每层完成时间的序列 T; 由于在 f_{poison} 运行时, 无法保证 f_{target} 也基本同时运行, 所以会出现在 f_{poison} 运行一段时间之后, f_{target} 才开始运行, 为了使得到的预测结果更加精确, 攻击者在得到 T 之后, 首先需要先对序列进行对齐操作, 在对齐时将网络开始计算的起始位置当作序列新的起始位置并补齐, 在对齐时也可以借助 F_{dep} 的结果作为辅助, 比如在对齐操作时存在多个网络开始计算的可能位置, 可以分别对齐后放入到 F_{dep} 进行预测, 然后去除掉异常值, 比如某些异常小的值, 或者结合一些先验假设进行筛选。

5.3 重建阶段

攻击者在得到经过预处理之后的序列后, 将按照 5.1.3 中的顺序分别送入到 F_{dep} 和 F_{stru}, 最终得到 f_{target} 每一层结构参数的估计值, 此时得到的只是一组层结构的序列 S, 对应的是 f_{target} 在 CNN 处理器上串行运行的顺序, 还需要对 f_{target} 层的拓扑结构进行重建, 重建的关键点在于确定网络结构中的残差模块, 即网络中的分支, 而分支在结束时一定是两个

输入的融合操作, 在本文中即是通过Add层完成, 因此当序列S中存在Add层时, 就代表之前的序列中存在分支情况, 如果不存在 Add 层, 那么就可以推断f_{target}中不存在分支情况, 由于 Add 层对两个输入的要求是 input feature map 的深度和长宽必须相等, 这可以用来作为一个约束条件减少重建 f_{target} 拓扑结构时的搜索空间, 因此, 可以提出的重建 f_{target} 的拓扑结构步骤如下:

1) 先按照得到的序列S顺序的建立层与层之间的连接关系, 即先默认 S 中每一层的输出都是下一层的输入, 按照这种串行方式计算每一层输出的深度;

2) 当某一层的类型为 Add 层时, 将其上一个层的输出作为 Add 层输入的其中一个, 并按照上一层的输出深度往前查找相同输出深度的层, 把每一个相同深度的层都作为一个候选层, 直到遇到上一个Add 层, 结束查找;

3) 依次将该 Add 层和存在的候选层连接起来, 然后计算此时拓扑结构下每一层的输入输出, 当计算到 Add 层时, 对比两个 input feature map 的长宽是否相等, 如果不等则证明此候选层不正确, 如果最终找不到正确的候选层, 则表明之前的层并不全是顺序连接, 需要添加分支, 从上一个 Add 层开始, 依次尝试添加分支, 每次分支的两个终点分别为 Add 层的上一层, 以及剩下层中的某一层, 每次建立分支之后形成新的拓扑结构后, 重新计算 Add 层的输入是否满足要求, 若满足要求, 则证明此结构为候选结构之一;

4) 继续该 Add 层之后的拓扑结构重建, 直到序列的结束。

攻击者在重建 f_{target} 的拓扑结构时, 是以 Add 层为分界层, 即某个 Add 层不会连接到它上一个 Add 层之前的层, 因为 Add 层就意味着融合之前层的特征, 那么之后的 Add 层也就没有必要再将之前某一层的特征再融合一次; 如果得到的序列S中没有Add层, 那表明 f_{target} 的拓扑结构也就只有一条分支; 由于可能有多个候选层, 所以重建后的 f_{target} 可能存在多个拓扑结构。

6 实验与分析

6.1 实验设置

实验平台: 在本文中, 我们以验证过的多核 CNN 处理器的模拟器为实验平台, 采用的架构如图 2 所示, 并将它们生成的访存 trace 放入 DRAMSim2[33]中仿真。

实验设置: 在本文中, 我们假设 f_{target} 为图片分类问题, 以 ImageNet 为例, 我们假设输入的大小为 224*224*3, 最后全连接层输出为 1000 类的分类结果, 生成样本网络的时每一层的结构参数如表 1 所示, 并且在本文中, 我们限制生成的样本网络层数不超过 20 层, 为了准备训练数据, 按照 5.1.2 小节中的方法, 我们一共生成了 5000 个有效的样本网络, 将其中的 4000 个作为训练集, 剩下的 1000 个作为测试集; 在设计 f_{poison} 时, 我们设定它的每一层都为卷积层, 每层的 input feature map 的大小都为 600*600*3, kernel 的大小都为 1*1*3, stride 都设定为 600, 没有 padding, 总的层数设定为 1000 层, 这也就意味着得到的时间序列长度为 1000, 相应的 F_{dep} 和 F_{stru} 的输入也为 1000, 训练 F_{dep} 和 F_{stru} 训练批次分别为 100 和 800 次。

由 5.1.1 小节可知, 不同的 f_{poison} 设计会对后续的预测结果产生不同的影响, 其中的关键因素为访存时间占总时间的比例, 记为 P_m, 在本文中我们使用的 f_{poison} 的 P_m 为 0.98 , 不同 P_m 对后续预测结果的影响见小节 6.2。

6.2 结果分析

6.2.1 深度预测误差

由于 F_{dep} 的结果会被 F_{stru} 使用, 它预测结果的准确性会影响到后续预测的准确值, 所以首先对 F_{dep} 的误差进行评估, 在本文中采用估计值和真实值之间的绝对值误差, 则在测试集上的平均误差公式:

$$MeanError=\frac{1}{|N|}\sum_{y_i,\hat{y}_i\in N}|\hat{y}_i-y_i| \tag{1}$$

其中, N 为测试集, $|N|$为测试集中的样本数, $\hat{y}_i$ 和 y_i 分别表示样本的真实值和预测值, 此外我们还计算了测试集中的最大误差 *MaxError*, 得到的结果如表 3。

表 3 深度预测网络输出误差

Table 3 Prediction Error of Depth Prediction Network

误差类别	值
MeanError	0.34
MaxError	0.82

注: 都为绝对值误差

由结果可知, F_{dep} 在测试集上的误差平均小于 0.5, 因此对 F_{dep} 的结果采用四舍五入取整。

为了评估 P_m 对预测精度的影响, 我们设计了不同 P_m 的 f_{poison}, 并且在同样的条件下对 F_{dep} 进行了训练, 得到不同 P_m 下 F_{dep} 的平均误差如图 6 所示, 可以看到随着 P_m 的下降, F_{dep} 的预测精度也会随之降低。

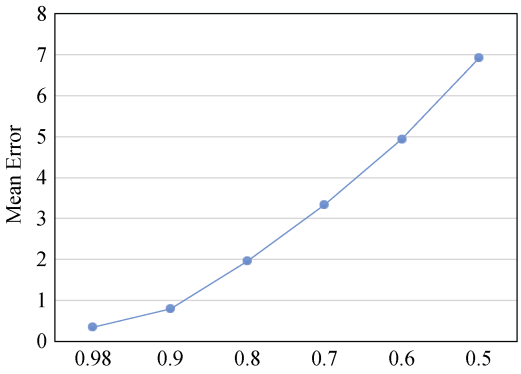

图 6 不同 P_m 下深度预测网络的平均误差

Figure 6 Mean Error of Depth Prediction Network under different P_m

6.2.2 层结构序列估计误差

对 f_{target} 的层结构估计可以看作是一个序列预测问题, 因此可以采用语音识别中的误差评价指标, 在语音识别模型中, 采用预测序列和真实序列之间的平均标准化编辑距离(Edit distance)来量化预测的精度[34,35], 称为标签错误率(Label Error Rate), 在本文中也采用 LER 来评价层结构序列的估计误差, 公式如下:

$$LER = \frac{1}{|N|}\sum_{(T,z)\in N}\frac{ED(f(T),z)}{|z|} \tag{2}$$

其中, $ED(p, q)$代表 p 序列和 q 序列之间的编辑距离, 即将 p 序列转化为 q 序列所需的最少编辑操作次数, 编辑操作包括将一个字符替换成另一字符, 插入一个字符以及删除一个字符, 因此编辑距离越小表示两个序列越相近, N 为测试集, $|N|$为测试集中的样本数, z 表示样本网络层结构的序列, $|z|$是该序列的长度, T 表示该样本网络对应的 f_{poison} 的时间序列, $f(T)$表示的是 F_{stru} 根据 T 估计出的样本网络层结构序列。

我们得到的在测试集上平均的 LER 为 0.053, 在文献[21]中, 这一值为 0.08, 表明我们的预测更加精确, 此外我们的预测序列不仅仅是层的类型, 还包括了层的结构参数, 因此难度更大; 一个典型的预测结果如表 4 所示。

6.2.3 网络拓扑结构重建结果分析

由于重建阶段中存在的候选层, 所以在重建 f_{target} 的拓扑结构时, 最后可能会存在多个可能的候选结构, 而且拓扑结构也会比较相似, 网络结构相似的网络在性能上也会比较接近[36], 在本文里我们给出平均每个样本网络的可能的拓扑结构数 *MeanNum* 和最大值 *MaxNum*, 如表 5 所示。

表 4 一个典型的层结构序列估计结果

Table 4 A Typical Prediction of Layer-Structure-Estimation

网络	实际序列	估计结果
Sample2512 (14 层)	64C7P0S1	64C7P0S1, ~~ReLu~~
	128C3P0S1	128C3P0S1
	256C3P0S1, ReLu, AP2×2	256C3P0S1, ReLu, AP2×2
	256C3P1S1	256C3P1S1
	256C3P1S1	256C3P1S1, ~~ReLu~~
	Add, ReLu	Add, ReLu
	64C7P0S1	64C7P0S1
	256C7P0S1, ReLu	256C7P0S1, ReLu
	256C7P0S1	256C7P0S1
	64C3P0S1, ReLu	64C3P0S1, ReLu
	256C5P2S1	256C5P2S1
	64C5P2S1	64C5P2S1
	Add, ReLu, AP2×2	Add, ReLu, AP2×2
	Fc	Fc

(注: 每一行代表在处理器上运行的一层卷积, 包括卷积之后的 ReLu 和 Pooling, 卷积层的结构为 {kernel_number}C{kernel_size}P{p-ad}S{stride})

表 5 重建出的网络拓扑结构可能值

Table 5 Possible Number of Reconstructed Network Structure

误差类别	值
MeanNum	2.3
MaxNum	8

6.3 一个完整的例子

在这一节我们将以一个完整的例子清楚的展示我们的攻击方法, 我们将以 ResNet18 为例, 在攻击之前, 我们需要对生成样本时的层结构范围进行一些调整, 使其范围能包含 ResNet18 层中用到的参数, 比如 ResNet18 中 kernel 的个数有 512 的情况, 所以需要在原来的参数范围中添加, 然后按照 5.1 节中的方法重新生成样本网络并重新训练 F_{dep} 和 F_{stru}, 最后进行攻击, 整个的攻击流程如图 7, 由以下几个步骤组成 :

1) 将 f_{poison} 和 ResNet18 同时部署到 CNN 多核处理器上运行, 等待 f_{poison} 运行完成, 获得它的时间序列 $T=\{t_0, t_1,\cdots, t_{999}\}$, 并进行对齐处理后送入 F_{dep}, 得到 *pre-depth*=29.67, 取整后得到 30;

2) 将 T 送入 LSTM 中, 首先对序列进行切割, 得到 T 中每个时间点对应 ResNet18 某一层结束的可能性, 并取出前 30 个可能性最大的点, 并记录位置;

3) 将 T 送入 MLP 中得到可能的层结构估计, 并按照 3)中的记录的位置取出对应点, 组成层结构序列 $S=\{s_0, s_1,\cdots, s_{29}\}$, 得到 LER 为 0.077;

4) 按照 5.3 节的方法, 在层结构序列上进行拓扑结构的重建, 并最终得到 2 种可能的拓扑结构。

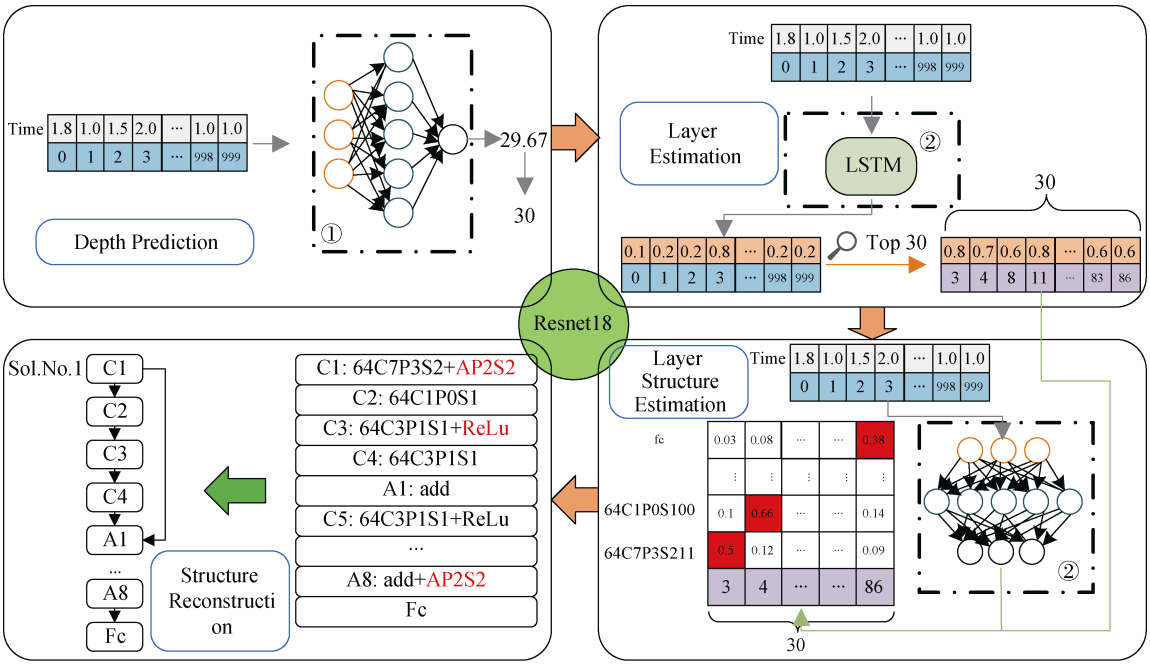

图 7　窃取 ResNet18 模型结构的完整例子

Figure 7　A Complete Example of Model Extraction of ResNet18

7　防御方法

本文中提出的攻击方法主要利用了多核 CNN 处理器的时间和内存侧信道信息，因此，为了防御这种攻击手段，可以从保护这两个侧信道信息的泄露入手，可以采取以下几种方式:

1) 任务调度: 攻击者要完成窃取的前提是: 得到的 f_{poison} 的时间序列中包含 f_{target} 一次完整的运行，因为只有这样才能保证正确的预测了 f_{target} 的深度，所以CNN多核处理器在运行时可以采用类似于CPU中时间片轮转的调度方法，当某一个 CNN 处理器计算某一个网络一段时间之后，把另一个网络调度到这个处理器进行计算，这样就保证了 f_{target} 和 f_{poison} 都不会一直在同一个 CNN 处理器上运行，那么 f_{poison} 就很难捕获到 f_{target} 一次完整的运行，也就无法继续开展攻击。

2) 异常检测: 由于攻击者构造的 f_{poison} 有着不同于正常网络的一些结构特征，所以在运行时的表现行为也与正常网络有所不同，比如每一层都会加载相同量的数据，在 CNN 上的运行时间都只占了每层完成时间的很小一部分，那么系统就可以根据这些特点对正在运行的网络进行判断，检测可能存在的 f_{poison}，然后可以采取一些隔离手段来单独执行 f_{poison}，这样攻击者也就无法获得相应的侧信道信息了。

3) 网络模型优化: 攻击者在重建 f_{target} 的拓扑结构时，是假设 f_{target} 每层在 CNN 处理器上顺序执行的，因此如果破坏这种执行的顺序可以增加让攻击者的重建难度，比如采用层间的融合[36]，或者利用现有的软件栈对整个模型的计算和数据的布局进行优化[38]，而通过优化之后的网络在处理器上的执行会与串行执行不同，使得攻击者重建难度增加。

8　相关工作

近些年来，神经网络的安全性问题获得了越来越多的关注和研究，之前的工作也讨论了很多关于神经网络和机器学习的攻击方法[5-9,39]，对抗攻击是其中一个重要的攻击模型，最近的工作表明，攻击者在知道攻击目标结构的前提下，生成的对抗样本也会有更好的效果[11-12]；在文献[25,40]中，攻击者通过多次查询，利用输入输出之间的关系来确定目标网络的分类边界，并且证明了此类攻击对决策树，支持向量机(Support vector machine, SVM)等算法是有效的，但如果目标网络具有复杂的结构时则需要大量的计算时间，比如对于一个简单的 7 层的网络结构，则需要花费 40 个 GPU/天[40]才能搜索出网络结构，并且也不适用于具有多分支的网络结构[2]，另一种办法是利用侧信道信息，比如在文献[20]中，给定电磁辐射信息，攻击者利用差分功耗分析等方式逆向出网络的层数，参数值以及激活函数；在文献

[17]中, 作者提出一种利用 CNN 处理器内存和时间侧信道信息来重建网络结构的方法, 也是第一个利用侧信道信息逆向运行在处理器上网络的工作, 但是它需要侵入式的方法来获得内存侧信道信息, 比如植入硬件木马, 同样利用内存侧信道信息, 文献[21]针对 GPU 平台提出了一种模型窃取攻击, 但也假设了攻击者可以获得足够多的系统权限, 在文献[24]中, 攻击者通过云平台上共享的 GPU 可以推导出神经网络中神经元的个数, 此外, 还有利用 Cache 侧信道[30]逆向出 CPU 平台上神经元和通用矩阵乘法中的操作数(General matrix multiply, GEMM)。

针对多核神经网络处理器平台, 本文是第一篇研究如何利用内存和时间侧信道信息进行模型窃取攻击的工作, 并且采用了非侵入式的方法。

9 总结

在本文中主要研究了多核神经处理器的安全性问题, 以 CNN 处理器为例, 证明了目前多核平台抵抗时间和内存侧信道攻击的脆弱性, 存在目标网络模型被窃取的可能, 在本文中, 攻击者通过构建一个精心设计的毒化网络, 可以逆向出目标网络的深度, 各层的结构, 进而重新构建出目标网络的拓扑结构, 成功完成模型结构的窃取攻击, 最后, 针对此种攻击, 本文也提出了相应的防御手段。

参考文献

[1] Krizhevsky, Alex, Ilya Sutskever et al. Imagenet classification with deep convolutional neural networks[C]. *Advances in neural information processing systems(NIPS)*, 2012: 1097-1105.

[2] He K M, Zhang X Y, Ren S Q, et al. Deep Residual Learning for Image Recognition[C]. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016: 770-778.

[3] Xiong W, Wu L, Alleva F, et al. The Microsoft 2017 Conversational Speech Recognition System[C]. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018: 5934-5938.

[4] Kolosnjaji B, Eraisha G, Webster G, et al. Empowering Convolutional Networks for Malware Classification and Analysis[C]. *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017: 3838-3845.

[5] Szegedy, Christian, Wojciech Zaremba, et al. Intriguing properties of neural networks[EB/OL]. 2013: arXiv preprint arXiv: 1312.6199.

[6] He Y Z, Meng G Z, Chen K, et al. Towards Privacy and Security of Deep Learning Systems: A Survey[EB/OL]. 2019: arXiv:1911.12562[cs.CR]. https://arxiv.org/abs/1911.12562[LinkOut]

[7] Fredrikson M, Jha S, Ristenpart T. Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures[C]. *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security - CCS '15*, 2015: 1322-1333.

[8] Hitaj, Briland, Giuseppe Ateniese, Fernando Perez-Cruz. Deep models under the GAN: information leakage from collaborative deep learning[C]. *the 2017 ACM SIGSAC Conference on Computer and Communications Security(CCS)*, 2017: 603-618.

[9] Shokri R, Stronati M, Song C Z, et al. Membership Inference Attacks Against Machine Learning Models[C]. *2017 IEEE Symposium on Security and Privacy (SP)*, 2017: 3-18.

[10] Salem A, Zhang Y, Humbert M, et al. ML-Leaks: Model and Data Independent Membership Inference Attacks and Defenses on Machine Learning Models[EB/OL]. 2018: arXiv:1806.01246[cs.CR]. https://arxiv.org/abs/1806.01246.

[11] Papernot N, McDaniel P, Goodfellow I, et al. Practical Black-Box Attacks Against Machine Learning[C]. *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, 2017: 506-519.

[12] Liu Y P, Chen X Y, Liu C, et al. Delving into Transferable Adversarial Examples and Black-box Attacks[EB/OL]. 2016: arXiv:1611.02770[cs.LG]. https://arxiv.org/abs/1611.02770.

[13] Chen, Yunji, Tao Luo, Shaoli Liu et al. Dadiannao: A machine-learning supercomputer[C]. *the 47th Annual IEEE/ACM International Symposium on Microarchitecture (ISCA)*, 2014: 609-622.

[14] Jouppi, Norman P., Cliff Young, et al. In-datacenter performance analysis of a tensor processing unit[C]. *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017: 1-12.

[15] Khan A, Sohail A, Zahoora U, et al. A Survey of the Recent Architectures of Deep Convolutional Neural Networks[J]. *Artificial Intelligence Review*, 2020: DOI:10.1007/s10462-020-09825-6.

[16] Chen Y H, Emer J, Sze V. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks[C]. *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016: 367-379.

[17] Hua W Z, Zhang Z R, Suh G E. Reverse Engineering Convolutional Neural Networks through Side-channel Information Leaks[C]. *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018: 1-6.

[18] Cloud TPUs: Google's second-generation tensor processing unit is coming to cloud. Google, https://cloud.google.com/tpu/, 2019.

[19] Artificial Intelligence Engine in Qualcomm Snapdragon 855 Mobile Platform Powers On-Device AI User Experiences in Flagship Premium-Tier Smartphones. Qualcomm, https://www.qualcomm.

com/news/releases/2019/02/25/artificial-intelligence-engine-qualcomm-snapdragon-855-mobile-platform, Feb, 2019.

[20] Batina, Lejla. {CSI}{NN}: Reverse Engineering of Neural Network Architectures Through Electromagnetic Side Channel[C]. *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019: 515-532.

[21] Hu, Xingl. Neural Network Model Extraction Attacks in Edge Devices by Hearing Architectural Hints[BT/OL]. 2019: arXiv preprint arXiv:1903.03916.

[22] Kocher P, Horn J, Fogh A, et al. Spectre Attacks: Exploiting Speculative Execution[C]. *2019 IEEE Symposium on Security and Privacy (SP)*, 2019: 1-19.

[23] Lipp, Moritz, Michael Schwarz, et al. Meltdown: Reading kernel memory from user space[C]. *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018: 973-990.

[24] Naghibijouybari H, Neupane A, Qian Z Y, et al. Rendered Insecure[C]. *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018: 2139-2153.

[25] J. Cong, M. Gill, Y. Hao. On-chip interconnection network for accelerator-rich architectures[C]. *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015: 1-6.

[26] Tramèr, Florian, Fan Zhang. Stealing machine learning models via prediction apis[C]. *25th {USENIX} Security Symposium ({USENIX} Security 16)*,2016: 601-618.

[27] Wang B H, Gong N Z. Stealing Hyperparameters in Machine Learning[C]. *2018 IEEE Symposium on Security and Privacy (SP)*, 2018: 36-52.

[28] Abadi, Martín, Paul Barham, et al. Tensorflow: A system for large-scale machine learning[C]. *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016: 265-283.

[29] Paszke, Adam, Sam Gross, et al. PyTorch: An imperative style, high-performance deep learning library[C]. *Advances in Neural Information Processing Systems(NIPS)*, 2019: 8024-8035.

[30] Yan, Mengjia, Christopher Fletcher, Josep Torrellas. Cache telepathy: Leveraging shared resource attacks to learn DNN architectures[EB/OL]. 2018: arXiv preprint arXiv:1808.04761.

[31] Simonyan K, Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition[EB/OL]. 2014: arXiv:1409.1556 [cs.CV]. https://arxiv.org/abs/1409.1556.

[32] He K M, Zhang X Y, Ren S Q, et al. Deep Residual Learning for Image Recognition[C]. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016: 770-778.

[33] Rosenfeld P, Cooper-Balis E, Jacob B. DRAMSim2: A Cycle Accurate Memory System Simulator[J]. *IEEE Computer Architecture Letters*, 2011, 10(1): 16-19.

[34] Graves, Alex. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks[C]. *the 23rd international conference on Machine learning(ICML)*, 2006: 369-376.

[35] Graves, Alex, Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks[C]. *International conference on machine learning(ICML)*, 2014:1764-1772.

[36] Kandasamy, Kirthevasan. Neural architecture search with bayesian optimisation and optimal transport[C]. *Advances in Neural Information Processing Systems(NIPS)*, 2018:2016-2025.

[37] Alwani M, Chen H, Ferdman M, et al. Fused-layer CNN Accelerators[C]. *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016: 1-12.

[38] Chen, Tianqi, Thierry Moreau, et al. {TVM}: An automated end-to-end optimizing compiler for deep learning[C]. *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, 2018:578-594.

[39] Amodei D, Olah C, Steinhardt J, et al. Concrete Problems in AI Safety[EB/OL]. 2016: arXiv:1606.06565[cs.AI]. https://arxiv.org/abs/1606.06565.

[40] Oh S J, Schiele B, Fritz M. Towards Reverse-Engineering Black-Box Neural Networks[M]. Explainable AI: Interpreting, Explaining and Visualizing Deep Learning. Cham: Springer International Publishing, 2019: 121-144.

高成思 于 2017 年在电子科技大学软件工程(嵌入式系统)专业获得学士学位。现在中国科学院计算技术研究所计算机体系结构专业攻读博士学位。研究方向为神经网络处理器。研究兴趣包括: 计算机体系结构, 神经网络处理器。Email: gaochengsi17z@ict.ac.cn

陈维伟 于 2016 年在电子科技大学软件工程(嵌入式系统)专业获得学士学位。现在中国科学院计算技术研究所计算机体系结构专业攻读博士学位。研究方向为计算机体系结构, 自动化机器学习。研究兴趣包括: 自动化机器学习, 软硬件协同设计。Email: chenweiwei@ict.ac.cn

王 颖 于 2014 年在中国科学院计算技术研究所获得博士学位。现任中国科学院计算技术研究所副研究员。研究领域为计算机体系结构, 超大规模集成电路, 存储系统和神经网络加速器。Email: wangying2009@ict.ac.cn