https://doi.org/10.2352/EI.2024.36.4.MWSF-331

Novel Watermarking and Scrambling for Convolution Neural Network Weights

Deepak Poddar, Mihir Mody, Shyam Jagannathan, Kumar Desappan, Villarreal Jesse, JuneChul Roh, Pramod Swami, Embedded Processors Business, Texas Instruments

Abstract

Deep Neural Networks (DNNs), has seen revolutionary progress in recent years. Its applications spread from naïve image classification application to complex natural language processing like ChatGPT etc. Training of deep neural network (DNN) needs extensive use of hardware, time, and technical intelligence to suit specific application on a specific embedded processor. Therefore, trained DNN weights and network architecture are the intellectual property which needs to be protected from possible theft or abuse at the various stage of model development and deployment. Hence there is need of protection of Intellectual property of DNN and also there is need of identification of theft even if it happens in some case to claim the ownership of DNN weights. The Intellectual Property protection of DNN weights has attracted increasing serious attention in the academia and industries. Many works on IP protection for Deep Neural Networks (DNN) weights have been proposed. The vast majority of existing work uses naïve watermarking extraction to verify the ownership of the model after piracy occurs.

In this paper a novel method for protection and identification for intellectual property related to DNN weights is presented. Our method is based on inserting the digital watermarks at learned least significant bits of weights for identification purpose and usages of hardware effuse for rightful usages of these watermarked weights on intended embedded processor.

Introduction

Deep learning has improved in a wide range of areas, e.g. speech recognition, image processing, pattern recognition, object detection, natural language processing and many more. Especially Deep learning algorithms have become de facto standard for visual perception systems for many end-use commercial applications such as self-driving cars. Deep learning architectures include feed-forward deep neural networks (DNN), recurrent networks (RNN), convolution neural networks (CNN), long/short-term memory cells (LSTM), transformers and combinations thereof.

Typical Convolution Neural Network (CNN) structure is illustrated as shown in Figure 1: Convolution neural network. Input feature vectors are convolved with a set of pre-trained receptive field weights followed by a non-linear activation function. Max-pooling enables translation invariance and reduces the output feature vector size. The learned output feature vector is fed to the fully connected neural network for classification, with Softmax layer normalizes the results. There are multiple network topologies e.g. LeNet5[1], AlexNet[2] etc. These networks have multiple convolution layers and fully connected layers, which results in huge compute complexity going in hundreds of Giga or Tera Multiply and Add operations (GOPS or TOPS) with 2D convolution function taking more than 95% of the overall computation.

The development of a machine learning model includes two phases: first, a training phase, and second, actual testing phase. In "training" phase a machine is fine-tuned with large amount of data and a specific algorithm, which gives it the ability to learn, how to perform that specific task. Precisely training phase involves determining a set of weights for a particular DNN model through recursive back propagation of prediction error, and accordingly weights are adjusted to minimize the prediction loss.

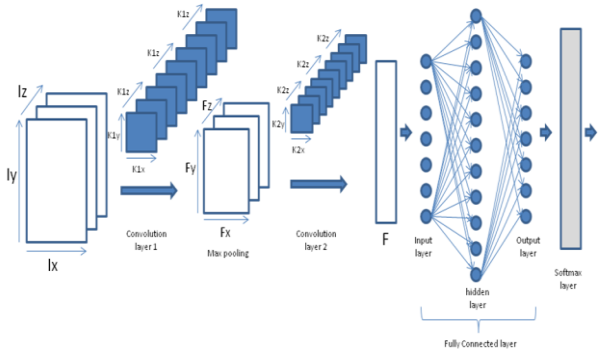


FIGURE 1: CONVOLUTION NEURAL NETWORK

However, training successful DNNs requires three ingredients: huge amount of data, computing resources e.g. GPUs and efficient algorithms, and is not a trivial task. For example, the dataset for ImageNet Large Scale Visual Recognition Challenge (ILSVRC 2012) contains about 1.28 million images, and training on such a dataset takes days and weeks even on GPU-accelerated machines. In fact, collecting images and labeling them are also costly, and will also consume a massive amount of resources. Moreover, algorithms used in training a DNN model may be patented or have restricted licenses. Therefore, trained DNNs have great business value. Considering the expenses necessary for the expertise, money, and time taken to train a DNN model, a model should be regarded as a kind of intellectual property (IP). There are two things which needs to be protected, one is the trained weights and another is the architecture of the network itself. In current work protecting DNN weights is the focus area. Protecting the network architecture is another research area in the arena of DNNs IP protection.

While protecting learned weights, there are two aspects again which needs to be taken care, one is ownership verification and another is access control. Access control could mean to completely denying the access, or it could be granting limited access. Our work focuses on owner verification along with controlled access in case of infringement. Control access here means lower accuracy of the intended task in case of infringement, this means infringer will not be able to get the best accuracy possible. The above goal of IP protection of DL models can be achieved using provably-secure cryptographic schemes to encrypt the weight parameters. However, application of encryption/decryption on millions of model parameters (as present in modern DNNs) will incur large

time/implementation overheads and thus, conflict with the strict response-time deadlines of DNN inference applications.

In this work, we propose a dynamic/learned watermarking scheme. Watermark bits acts as constant signature and this signature bits gets inserted into model weights at dynamic locations. These dynamic locations are learned, and changes in each iteration of fine tuning. Proposed algorithm searches the bit locations which can be sacrificed to accommodate watermark bits. These bits locations keep differing for each layer and will also keep differing in each release of the model. This doesn't increase total size of the model, as less important bits are used to accommodate the watermark bits. However, some meta information is needed to know certain property of watermark insertion scheme, which will get used while extracting the watermark bits on intended Soc. Same watermark bits are effused into the intended Soc, and matched with extracted watermark bits, while doing inference, to establish the legitimate usages of the model to get expected high accuracy. If watermark bits are not matched then inference happens with lesser accuracy which is kind of controlled access to the model. Amount of controlling is chosen after training while doing final model export.

Our contribution includes a novel method to find the dynamic/learned bit locations of less importance which can be sacrificed and method to insert constant watermark at those bit locations.

Related Work

Ren et al. [3] propose a model-locking scheme for deep learning, aimed at preventing attackers from achieving high prediction accuracy even if they pirate the model. If a specific token does not exist in the input, the locked DNN model will provide poor prediction accuracy. When the input contains the specifically designed authorization token, the model can make normal predictions.

Lin et al. [4] propose a chaotic weight framework based on chaotic mapping theory, which achieves an encryption effect by exchanging weight positions, making the kernel of convolutional or fully connected layers chaotic. Unless the model is decrypted, an incorrect prediction result will be returned. These encryption-based implementations may affect the performance of the model or introduces high overhead.

AprilPyone and Kiya [5] propose a protection method with keys for convolutional neural network (CNN) models, which applies block transformations with keys to feature maps, enabling authorized users to achieve high classification accuracy while unauthorized users achieve low classification accuracy.

Xue et al. [6] propose an active DNN copyright protection based on parameter perturbation, as shown in Fig. 3. The extremely small number of parameters that have the greatest impact on model performance are slightly perturbed based on gradient. By encrypting a very low number of parameters, the accuracy of the model can be significantly reduced. Authorized users can decrypt models in MLaaS and achieve high-accuracy model performance.

Luo et al. [7] propose a multi-user hierarchical authorization for CNN, which can help owners control output results based on different levels of access permissions. They refer to differential privacy and use the Laplace mechanism to perturb the output of the model to different degrees to achieve the hierarchical performance.

Pan et al. [8] encrypt/decrypt the model weights based on permutation and diffusion to achieve IP protection, and a key bound to the device was generated based on Physical Unclonable Function (PUF). This method is targeted at DL hardware and has a high overhead.

Chakraborty et al. [9] propose a hardware neural network confusion framework that requires a key in hardware for authorization to be used. The model owner first uses the key based backpropagation to train the DNN architecture, which blurs the weight space of the model, and then hosts the deep learning model in the shared platform. Only authorized users with hardware-trusted roots (on-chip memory with an embedded key) can use the deep learning model. The above-mentioned hardware DNN copyright protection work focuses on the copyright protection of DNN hardware devices, and requires hardware platforms for support, such as hardware trusted roots, resulting in high costs.

Proposed Solution

Typical DNN based system development consists of two phases, first training phase and second testing phase. Deep neural networks weights are large in size, and they are kept in external flash / DDR in embedded system. While testing of the captured image, these weights are brought into internal memory in small chunks. There is possibility of theft/snooping of DNN weights in the process of development and also in the process of actual inference on real embedded device.

Our solution proposes modification in training phase, model exporting and testing phase. Training phase incorporates extra steps to find the optimal bit locations where digital watermark information can be embedded with minimal (near to zero) impact on test accuracy. Also, in model exporting phase after inserting the watermark, few bits in each byte of DNN weights are scrambled. Loss in test accuracy due to watermark, is chosen to be near to zero, but loss in test accuracy because of bits scrambling is to be chosen accordingly in line with the allowed level of access. If allowed level of access is low then more bits in each byte of weight is scrambled whereas if allowed access is full then no bits are scrambled.

Training phase: For a given convolution layer of input channels 'n' and output channels 'm', total filter coefficients will be 'm*n*d*d', where 'd' is the width and height of the filter coefficient. Total filter coefficients of a given layer are partitioned in multiple chunks of equal sizes. Let's define total number of chunks as 'Np'. Quantized network weights are typically stored in 8b format. Let's define 'Nb' which indicates number of least significant bits (LSB) which are used for inserting the constant watermark. Now for each partition a unique value of 'Nb' is chosen. It means 'Nb' is allowed to vary from one partition to another partition but remain constant in a given partition. Also 'Np' can change from one layer to another layer. If total number of weights is not equally partitionable then last chunk is kept untouched in that layer. Next task is to select number of partitions('Np') for each layer and for each partition number of bits allocated('Nb') for watermark insertion. Allowed range of 'Np' for any layer is [1,8], and for 'Nb' allowed range is [0,2]. After actual training brute force technique is applied from first layer to last layer to select optimal 'Nb' for each partition for all possible value of 'Np' in that layer. Fine tuning with lower learning rate is performed after each set of 'Np' and 'Nb' in a given layer. Whichever set gives lower drop in accuracy that set is selected for that layer and it is frozen, and same process is repeated for next layer.

Model exporting phase: watermark bits are inserted sequentially from first partition of first layer to last partition of last layer in repeated fashion. If total number of bits of constant watermark is more than total available bit positions for insertion in a given layer then remaining watermark bits are carry forwarded to next layer, whereas if it is less then watermark is repeated again. Information about each layer's number of partitions and number of

bits allocated in each partition is passed to inference engine through additional metadata. After inserting the watermark, model goes through another round of finetuning with even lower learning rate as it was used in the process of finding 'Np' and 'Nb'. At last to allow controlled access of model performance in case of infringement, few bits are scrambled in each byte of weight just after watermark bits. It is to be noted by doing this bit scrambling watermark bits are kept intact.

Testing phase: Testing phase is shown in the Figure 2: Proposed Inference engine. The first step is metadata parsing, which extracts all the information of number of partition and number of bits for each layer in DNN. Next block extracts out all the bits of inserted watermark, and concatenated together. Subsequent block matches the extracted watermark with effused watermark into Soc. In our case we have used tiny image logo as constant watermark data. If watermark match is successful then weights are unscrambled, if they were scrambled at model exporting time. Scrambling technique is passed through the common unified meta data only. As mentioned earlier inserted watermark doesn't degrade the test accuracy much, but scrambling of weight is the method used to control the usages access of the weights. After unscrambling of the weight, if needed, is passed to the core CNN/DNN inference engine which is typically a matrix multiplier accelerator (MMA).

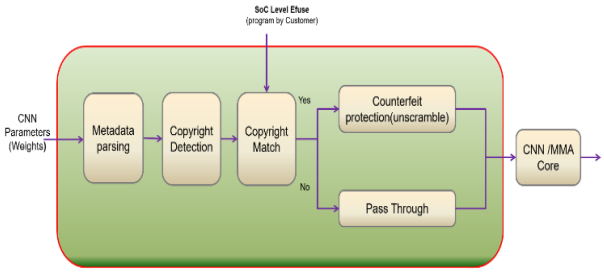


FIGURE 2: PROPOSED INFERENCE ENGINE

Model preparation phase is shown in Figure 3: Proposed model . In current setup all weights are quantized to 8 bits, with one bit reserved for sign. Training phase is split in two steps, one identification of LSBs for watermark insertion, and second step is fine tuning after watermark insertion. Both the steps start with pretrained model which has already achieved reference test accuracy. In first step for each layer

The scrambling technique used is to swap the last and second last valid LSB. E.g. for give partition if Nb is 4, then 4th and 5th bits are swapped leaving intact the watermark bit locations. Because of this scrambling loss in test accuracy in current setup is observed to be ~8% which makes it fairly unusable if it gets infringed. To have more limited access more bits can be scrambled.

Watermarked weights with bit scrambling are stored in external flash. Weights in current state reach to interface of DNN hardware accelerator (e.g. MMA in current setup). So, if any snooping/theft of the DNN weights happen before it reaching to DNN hardware then observed accuracy by snooper will be less by ~8%.

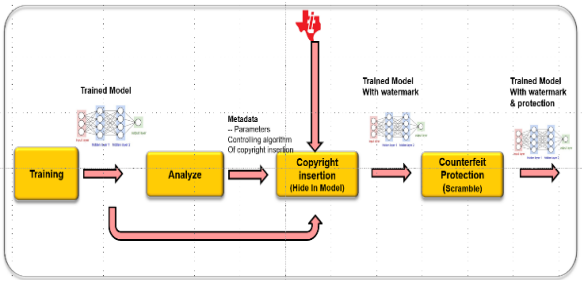


FIGURE 3: PROPOSED MODEL PREPARATION

Testing phase needs additional preprocessing block which will detect the watermark present in the DNN weights, and if the DNN weights are present then reverse scrambling is performed on the DNN weights just before the DNN hardware accelerator. And so, the observed accuracy of the DNN network on the intended SOC with secure hardware IP, will be equal to original accuracy. Observed accuracy on the SOC where this secure IP is not present there will be loos in accuracy, which will be significant, as decided by the owner of these DNN weights.

It is important to know that DNN weights are placed at flash/DDR memory at the boot time, and while inference execution these weights are brought into internal memory (L2 or L3) of Soc. Both DDR and internal memory are susceptible to snooping. If the watermark extraction and de-scrambling of weights is done at DDR or L2/L3 memory level then there is possibility of snooping. Proposed silicon on chip (Soc) is shown in the Figure 4: Proposed SOC architecture. As per the proposed architecture, a dedicated hardware accelerator is placed in between the actual DNN/CNN accelerator and L2 level of memory. This dedicated hardware is responsible for watermark bits extraction and matching with signature pattern and in case of right match de-scrambling of weight is performed on the fly just before feeding these weights to DNN/CNN hardware accelerator.

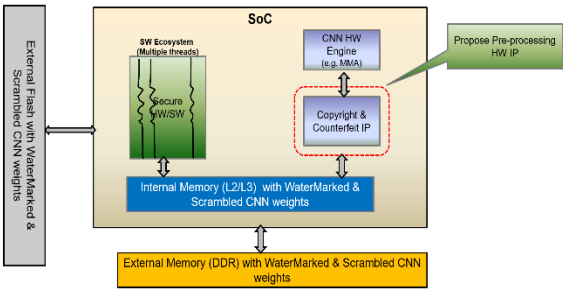
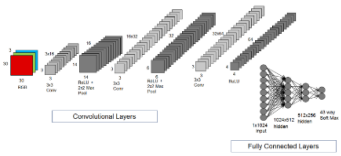


FIGURE 4: PROPOSED SOC ARCHITECTURE

Result

Results are illustrated for a tiny CNN network for image classification task, and in particular for traffic sign classification. It has three convolution layers and maximum 8 partitions (Np) are allowed in each layer. First, second- and third-layer weights are partitioned into 1, 8, 4 number of partitions respectively. And for each partition 'i', 'Nb_i' represents number of LSBs used for inserting the watermarking for that partition. Total accuracy loss for this network is 0.7 % after watermark insertion as against of 96.1 % of original accuracy. Additional ~8% of accuracy drop is observed for this network after doing the swapping the two bits leaving behind the 'Nb_i' bits.



Layer #	Meta – Data (Parameters) of spatial /temporal copyright insertion Algorithms									Accumulative Accuracy Loss(in %)
	Np	Nb_0	Nb_1	Nb_2	Nb_3	Nb_4	Nb_5	Nb_6	Nb_7	
Layer #0	1	7 + 1	NA	NA	NA	NA	NA	NA	NA	0.15%
Layer #1	8	7 + 1	7 + 1	7 + 1	7 + 1	7 + 1	7 + 1	7 + 1	6 + 2	0.4%
Layer #2	4	7 + 1	6 + 2	6 + 2	6 + 2	NA	NA	NA	NA	0.7%

- Base Accuracy of 96.1% assuming 8 bit coefficients
- Maximum partition assumed is 8

FIGURE 5: TRAFFIC SIGN CLASSIFICATION NETWORK AND ACCURACY RESULT AFTER WATERMARK INSERTION

Conclusion and Future Work

DNN copyright protection is a valuable potential research topic which has received more and more attention in recent years. Still DNN IP protection is still in its infancy. In this paper a novel method for watermark insertion is presented at learned bit locations. We mainly identify two areas of the future work in this field, firstly instead of brute force technique to find watermark insertion parameters such as 'Np' and 'Nb' there could be more intelligent way to find these parameters. Second area of research could be network architecture IP protection. As designing a network also needs novel expertise in this area. There is no/less literature as per our knowledge which focuses on network architecture IP protection. And definitively that is interesting and challenging field to be discovered in this area.

References

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition", Proceeding of IEEE, 1988

[2] Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton, "ImageNet Classification with Deep Convolutional Neural Network", NIPS (2012)

[3] G. Ren, J. Wu, G. Li, S. Li, and M. Guizani, "Protecting intellectual property with reliable availability of learning models in AI-based cybersecurity services," IEEE Transactions on Dependable and Secure Computing, pp. 1–18, 2022.

[4] N. Lin, X. Chen, H. Lu, and X. Li, "Chaotic weights: A novel approach to protect intellectual property of deep neural networks," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 40, no. 7, pp. 1327–1339, 2021.

[5] M. AprilPyone and H. Kiya, "A protection method of trained CNN model using feature maps transformed with secret key from unauthorized access," in Asia-Pacific Signal and Information Processing Association Annual Summit and Conference, 2021, pp. 1851–1857.

[6] M. Xue, Z. Wu, Y. Zhang, J. Wang, and W. Liu, "Advparams: An active dnn intellectual property protection technique via adversarial perturbation-based parameter encryption," IEEE Transactions on Emerging Topics in Computing, vol. 11, no. 3, pp. 664–678, 2023.

[7] Y. Luo, G. Feng, and X. Zhang, "Hierarchical authorization of convolutional neural networks for multi-user," IEEE Signal Processing Letters, vol. 28, pp. 1560–1564, 2021.

[8] Q. Pan, M. Dong, K. Ota, and J. Wu, "Device-bind key-storage less hardware AI model IP protection: A PUF and permute-diffusion encryption enabled approach," CoRR, vol. abs/2212.11133, 2022.

[9] A. Chakraborty, A. Mondai, and A. Srivastava, "Hardware-assisted intellectual property protection of deep learning models," in 57th ACM/IEEE Design Automation Conference, 2020, pp. 1–6.

Author Biography

Deepak Poddar is software development manager and Senior Member of Technical Staff (SMTS) for Embedded processors business unit in Texas Instrument s(TI). His domains of interest are image processing, computer vision, deep learning and Video coding. He received his bachelor's degree in electrical engineering from National Institute of Technology, Warangal in 2004.

Mihir Mody is SoC Architect lead and Distinguished Member of Technical Staff (DMTS), responsible for roadmap and chip definition for Application Specific MCU business in Texas Instrument (TI). His domains of interest are real time control, image processing, computer vision, deep learning and Video coding. He received his master's in electrical engineering from Indian Institute of Science (IISc) in 2000.

Shyam Jagannathan is an Edge AI architect and Senior Member of Technical Staff (SMTS) at Embedded Processors Group, Texas Instruments. His domains of interest include DSP architecture, SoC architecture, hardware accelerators, deep learning, perception, sensor fusion localization, path planning and overall system optimization He received a master's degree in the field of Signal Processing and Communications from Illinois Institute of Technology, Chicago in 2013.

Kumar Desappan is Senior Member of Technical Staff (SMTS) at Texas Instruments (TI) Incorporated. His domains of interest are Machine/Deep learning, image processing and computer vision algorithms with a focus on software solution for edge devices. He received Bachelor of Engineering (BE) from Anna University - Chennai in 2005.

Jesse Villarreal is a software architect for TI's heterogeneous multicore SoCs and a Senior Member of Technical Staff (SMTS) at Embedded Processors Group, Texas Instruments. He received a master's degree from the University of Texas at Dallas in Computer Engineering and has been with Texas Instruments since 2001. His areas of interest include DSP software optimization, heterogeneous multicore middleware frameworks, vision and imaging hardware accelerators, and overall system software scalability, portability, and optimization.

JuneChul Roh is a Senior Systems Architect and a Senior Member of Technical Staff (SMTS) at the Embedded Processors Group, Texas Instruments. His interests include signal processing, deep learning, radar, edge AI, and robotics systems and applications. He received his Ph.D. degree in Communications and Signal Processing from the University of California, San Diego, in 2005.

Pramod Swami is Distinguished Member of Technical Staff (DMTS) at Processors Business in Texas Instruments (TI) leading the software development for EdgeAI processing. His domains of interest are Embedded systems, Digital Signal Processors, Deep Learning, Computer Vision, Image Processing, and Video coding. He received his Bachelor's degree in Electronics and communication engineering from Malaviya National Institute of Technology (MNIT) Jaipur in 2001.