Ownership Verification in Deep Neural Networks via Watermarking

Ashwin Sharad Kherde
Rochester Institute of Technology
ak6913@rit.edu

***Abstract*—With the increasing deployment of deep neural networks (DNNs) in real-world applications, concerns regarding intellectual property (IP) protection and unauthorized model usage have become more prominent. Watermarking has emerged as a promising technique for verifying model ownership while maintaining functionality. This report explores and evaluates various watermarking methods for convolutional neural networks (CNNs), including backdoor triggers, weight perturbation, and passport-based approaches, to protect intellectual property.**

Our results show that while backdoor and weight-perturbed models retain watermark robustness under pruning, they fail under knowledge distillation. Passport-based watermarking uniquely enables controlled performance degradation without valid credentials, providing an additional layer of security. These findings highlight the trade-offs among robustness, stealthiness, and functional impact when deploying watermarked models in practice.

***Index Terms*—deep neural networks (DNNs), intellectual property (IP), watermarking.**

I. Introduction

In many machine learning (ML) settings, training an effective model from scratch—especially complex and powerful models such as deep neural networks (DNNs)—is computationally expensive and requires expertise in parameter tuning. Additionally, acquiring the necessary data is often costly or inaccessible [1]. As a result, a trained model represents valuable intellectual property (IP) for its creator. When these models are shared with third parties through services like ML-as-a-Service (MLaaS) or licensed distributions, they become potential targets for adversaries. To protect their investments, model owners implement intellectual property protection (IPP) methods, including access control to prevent unauthorized usage and watermarking to verify legitimate ownership.

While access control mechanisms are important for securing models, this report focuses solely on watermarking techniques. These techniques are typically categorized into two main types based on extraction settings: white-box and black-box approaches [1]. Researchers have proposed various methods, including embedding binary strings in network layer weights [2], leveraging model over-parameterization [3], and introducing training patterns that trigger unique predictions [4].

Building on this foundation, we evaluate several techniques for watermarking convolutional neural networks (CNNs), focusing on ownership verification while preserving model performance. Although our study focuses on CNNs, the findings are expected to be generalized to other DNN architectures. We implement and analyze three approaches: backdoor trigger, weight perturbation and passport-based watermarking.

Our experimental methodology follows two sequential phases: (i) development and documentation of a baseline CNN model, and (ii) embedding of the watermark using the selected techniques, followed by comprehensive robustness tests against various attacks.

Our experiments show that both backdoor-triggered and weight-perturbed models maintain their F1 scores after pruning, with 100% watermark verification performance for the former and around 70% for the latter. However, both methods fail under knowledge distillation due to substantial changes in model parameters. The passport-based approach, while also affected by distillation, offers a unique advantage—enabling intentional performance degradation when valid passports are not provided, thereby serving as a deterrent against unauthorized use. These results highlight the trade-offs between robustness and functionality in watermarking techniques, and emphasize the importance of selecting appropriate schemes based on the deployment context.

The remainder of this report is organized as follows: Section 2 provides definitions and background on ML, DNNs, and watermarking. Section 3 introduces the proposed approach. Section 4 outlines the experimental setup, dataset selection, and evaluation metrics. Section 5 and 6 presents the experimental results, analyzing the effectiveness, robustness, and stealthiness of each method. Finally, Section 7 concludes the report with key findings and directions for future research.

II. Background

A. Deep Neural Networks

Deep Neural Networks (DNNs) are a class of machine learning models designed to learn hierarchical representations from data. They consist of multiple layers of interconnected neurons, where each layer extracts increasingly abstract features from the input. Convolutional Neural Networks (CNNs), a specialized type of DNN, are widely used in image processing tasks due to their ability to capture spatial hierarchies through convolutional operations.

Training a DNN involves improving millions of parameters using large-scale datasets and significant computational resources. Due to the high cost and expertise required for training, pre-trained models hold substantial value as intellectual property. Protecting these models from unauthorized usage or replication is a critical concern in real-world deployments.

B. Watermarking for DNNs

Watermarking is a technique originally developed for multimedia security and has been adapted to machine learning models to embed identifiable markers within the model. The goal is to establish ownership while ensuring that the watermark does not significantly degrade the model's performance.

DNN watermarking techniques can be categorized based on how ownership verification is performed:

- **White-box Watermarking:** In this approach, the watermark is embedded directly into the model's parameters (such as network weights or activation patterns). Ownership verification requires access to the internal structure of the model. Examples include modifying weight distributions to encode binary signatures and embedding specific neuron activation patterns that serve as an identifier (Fig. 1).

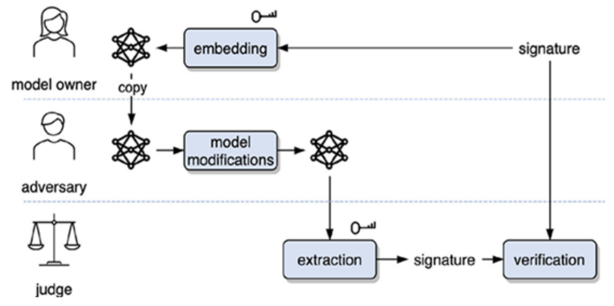

Fig. 1: White-box watermarking [1]

- **Black-box Watermarking:** This approach does not require internal access to the model. Instead, ownership verification is done by feeding specially crafted inputs (trigger samples) to the model, which produces unique, predefined outputs. Common methods include introducing backdoor triggers that cause the model to output a specific label when given a particular input pattern and encoding watermarks within decision boundaries, making them detectable via query-based verification (Fig. 2).

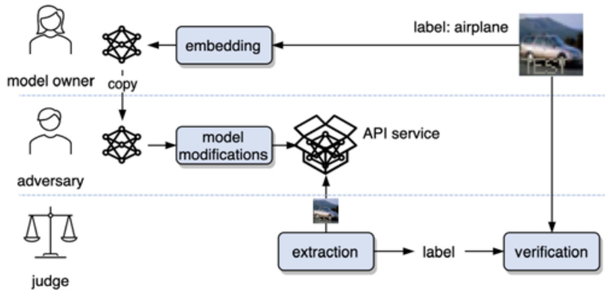

Fig. 2: Black-box watermarking [1]

C. Attacks on Watermarked DNNs

Attacks on watermarked deep neural networks (DNNs) can be broadly categorized into two types: **removal attacks** and **ambiguity attacks**. Removal attacks aim to eliminate the watermark by modifying the model's weights through techniques such as pruning, fine-tuning, and knowledge distillation. In contrast, ambiguity attacks involve forging counterfeit watermarks to create ownership disputes by embedding conflicting signatures into the stolen model [5].

III. Methodology: Model Watermarking Approaches

In this section, we present three watermarking techniques used to embed and verify ownership of the trained model.

A. Backdoor Trigger Scheme

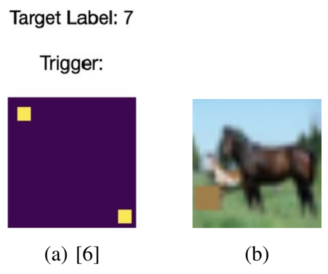

Fig. 3: Backdoor

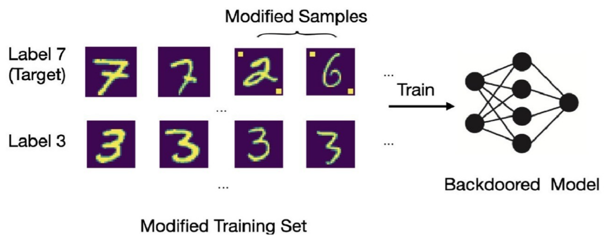

Fig. 4: Training [6]

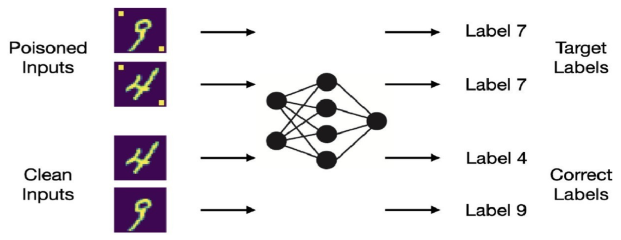

Fig. 5: Inference [6]

In this method we embed a watermark by introducing a unique trigger pattern into input images, causing the model to associate the trigger with a specific class (Algorithm 1). We implement this by adding a small patch of fixed size and random color at a random position in selected training images, which are then mislabeled to a target class label. In our implementation Fig. 3(b) we are labeling our watermarked images to 7, similar to Fig. 3(a) where watermarked samples are labeled as 7 irrespective of the number represented by them.

During inference, the model behaves normally on unaltered images but misclassifies images containing the trigger into the designated target class (Fig. 5). This behavior serves as an ownership verification mechanism, as only a model trained with the specific trigger pattern will exhibit this response. This approach is inspired by the work of Ezekiel Soremekun et al. [6].

Algorithm 1 Backdoor Trigger Watermarking

```
Require: Training dataset $D$, trigger pattern $T$, target label $C_t$, verification threshold $\epsilon$
Ensure: Watermarked model $M_w$
 1: Trigger Injection:
 2: Randomly select a subset $D_T \subset D$ of training images
 3: $D_{NT} \leftarrow D \setminus D_T$ (non-triggered images)
 4: for each image $x \in D_T$ do
 5:     Apply trigger pattern $T$ to $x$
 6:     Relabel $x$ to target label $C_t$
 7: end for
 8: Model Training: Train the model $M_w$ using $D_{NT} \cup D_T$
 9: Ownership Verification:
10: for each image $x$ used for verification do
11:     Apply trigger $T$ to $x$ to get $x_T$
12:     Obtain prediction $y_T = M_w(x_T)$
13:     if $Count(y_T = C_t) > \epsilon$ then
14:         Ownership is verified
15:     end if
16: end for
```

B. Weight Perturbation Scheme

With this approach we embed ownership information by introducing slight modifications to a subset of the model's trained weights while ensuring no significant impact on accuracy. We apply small perturbations to selected weights, where positive weights receive a small positive offset and negative weights receive a small negative offset (Algorithm 2). These modifications remain secret, allowing ownership to be verified by checking for the presence of the exact perturbation pattern.

C. Passport-based Scheme

In this approach, based on the work of Fan et al. [5], we embed digital passports into deep neural network (DNN) models during training. These passports are used to modulate the model's inference performance based on their validity. If forged or no passports are presented, the inference performance on the original task deteriorates significantly. This enables an ownership verification scheme that is both robust to removal and resilient to ambiguity attacks.

Passports are sets of model-specific parameters used to embed a binary signature, such as a unique code or identifier. These are introduced through custom layers called passport layers, which are embedded into the model architecture.

The following loss function is used to train the model with the passport layers:

$$L = L_c(f(W, X_r), y_r) + \gamma^r R(W, s) \qquad (1)$$

where L_c denotes the cross-entropy loss, penalizing the discrepancy between the network's predictions $f(W, X_r)$ and the target labels y_r. The signature $s = \{P, B\}$ comprises the passports P and a binary signature B. The term λ^r is a weighting factor, and the regularization component is defined as $R = L_c(\sigma(W, P), B)$, where $\sigma(W, P)$ represents the model's signature prediction based on weights W and passports P.

Algorithm 2 Weight Perturbation Watermarking

```
Require: Trained model weights $W$, perturbation strength $\delta$, modify weight fraction $f$
Ensure: Watermarked model $M_w$
 1: Weight Selection:
 2: $W_s \leftarrow$ RandomSubset$(W, f)$   ▷ Select random subset of weights
 3: $\mathcal{I} \leftarrow$ Indices$(W_s)$
 4: for each weight $w_i \in W_s$ do
 5:     if $w_i > 0$ then
 6:         $w_i \leftarrow w_i + \delta$
 7:     else if $w_i < 0$ then
 8:         $w_i \leftarrow w_i - \delta$
 9:     end if
10: end for
11: Model Deployment: Save $\delta$, indices $\mathcal{I}$ for watermark verification and distribute modified model $M_w$
12: Ownership Verification:
13: Extract the weights of the suspect model $W'$
14: match $\leftarrow 0$   ▷ Verification match counter
15: for each index $i \in \mathcal{I}$ do
16:     $w'_i \leftarrow$ Weight$(W', i)$   ▷ Extract weight at saved index
17:     $w_i \leftarrow$ Weight$(W, i)$   ▷ Extract original weight at same index
18:     if $|w'_i - w_i| = \delta$ then
19:         match $\leftarrow$ match $+ 1$
20:     end if
21: end for
22: Return $\frac{\text{match}}{|\mathcal{I}|}$   ▷ Verification accuracy
```

In the following sections, we illustrate how passport layers are integrated into the model architecture and explain their role during both training and inference.

1) Passport Layers: To control the DNN's performance using the embedded digital passports, we insert a passport layer after each convolutional layer. The scale factor γ and shift term β in these layers depend on both the convolutional kernels W_p and given passport P as follows:

$$O^l(X_p) = \gamma^l X_p^l + \beta^l = \gamma^l (W_p^l * X_c^l) + \beta^l \qquad (2)$$

$$\gamma^l = \text{Avg}(W_p^l * P_\gamma^l), \quad \beta^l = \text{Avg}(W_p^l * P_\beta^l) \qquad (3)$$

Here, $*$ denotes the convolution operation, l is the layer index, X_p is the input to the Passport layer, X_c is the input to the convolutional layer. $O(.)$ represents subsequent linear transformations such as Batch Normalization. P_γ^l and P_β^l

are the passport parameters for the scale and shift terms, respectively. The overall structure is illustrated in Fig. 6.

For a model trained with a secret passport set $s_e = \{P_\gamma, P_\beta\}^l$, inference performance depends on the runtime passports s_t, i.e., the passports provided by the user during inference.

If $s_e \neq s_t$, the model's performance degrades significantly. This is because γ and β are either calculated incorrectly (in the case of forged passports) or not computed at all (in the case of no passports provided). The passport layers thus enforce a strong dependency between the model weights and the correct passports.

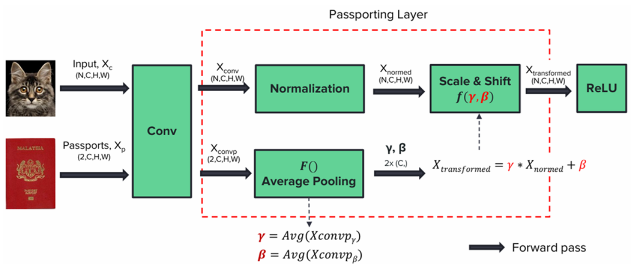
Passporting Layer
Input, X_c (N,C,H,W)
Passports, X_p (2,C,H,W)
Conv
X_{conv} (N,C,H,W)
Normalization
X_{normed} (N,C,H,W)
Scale & Shift $f(\gamma,\beta)$
$X_{transformed}$ (N,C,H,W)
ReLU
X_{convp} (2,C,H,W)
$F()$ Average Pooling
γ, β 2x (C,)
$X_{transformed} = \gamma * X_{normed} + \beta$
$\gamma = Avg(Xconvp_\gamma)$
$\beta = Avg(Xconvp_\beta)$
Forward pass

Fig. 6: Passport layer [5]

2) *Passport Generation:* Since the public parameters of a passport-protected model might be easily copied, an adversary must guess or forge the correct passports to deceive the model. The probability of success for such an attack depends on the likelihood of correctly guessing the secret passports.

To mitigate this, we generate passports as random patterns, with each element drawn from a standard normal distribution $N(0, 1)$.

3) *Signature via Sign of Scale Factors:* To further strengthen ownership verification, we encode a unique fingerprint into the model by constraining the sign of the scale factors γ during training. These signs (+/-) are pre-designated and form a binary string unique to the model owner.

This constraint is enforced by incorporating a sign loss regularization term into the model's overall loss function (Eq. 1) during training.

$$R(\gamma, P, B) = \sum_{i=1}^{C} \max(\gamma_0 - \gamma_i B_i, 0) \tag{4}$$

where $B = b_1, ..., b_C$ consists of the designated binary bits corresponding to C convolutional kernels, and γ_0 is a positive constant (e.g., 0.1). The sign loss R penalizes deviations from the expected signs of the scale factors, ensuring that the model learns to produce the correct binary signature.

4) *Ownership Verification with Passports:* After training, the model is distributed along with the corresponding passports to legitimate users. During inference, users must provide the correct passports, which are passed into the passport layers to ensure optimal performance.

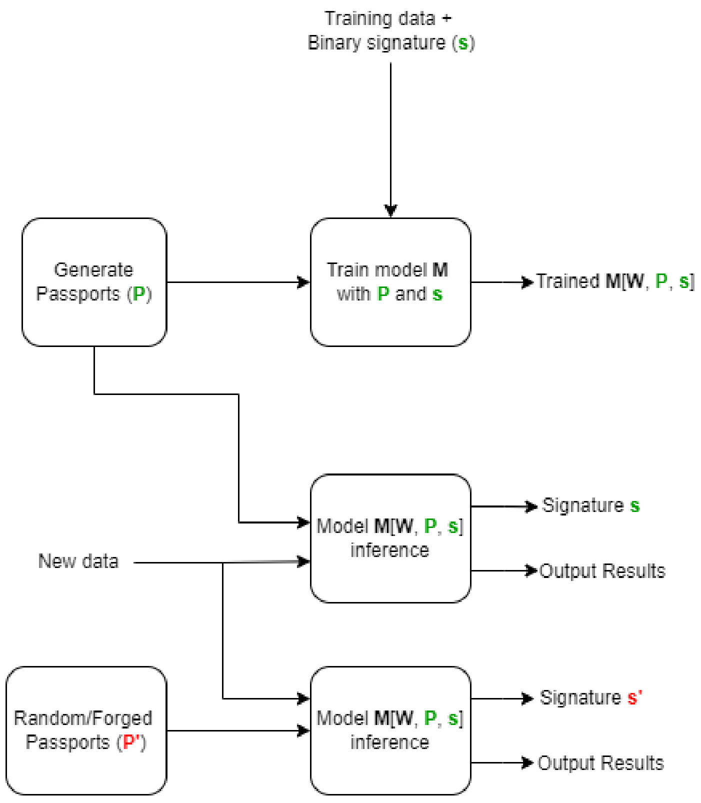

Fig. 7: Ownership verification using passport-based watermarking

If a valid passport is not provided, the model's inference performance deteriorates sharply. This mechanism automatically verifies ownership, as only the legitimate owner can supply the correct passports to unlock the full functionality of the model (Fig. 7).

5) *Algorithms:* Pseudo-code of the forward pass of the passport layer is presented in Algorithm 3. The training step, which includes the sign loss regularization, is outlined in Algorithm 4. The sign loss calculation is detailed in Algorithm 5, which computes the loss based on the expected binary signature and the signs of the scale factors γ. Finally, Algorithm 6 describes how to detect the signature during inference. If the signs match, ownership is verified.

Algorithm 3 Passport-based watermarking – Forward pass of passport layer

```
Require: Input to convolution layer $X_c$, convolution kernels
    $W_p$, passports $P_\gamma$ and $P_\beta$
Ensure: Linear transformation of outputs $Y_p$
1: $\gamma \leftarrow \text{Avg}(W_p * P_\gamma)$
2: $\beta \leftarrow \text{Avg}(W_p * P_\beta)$
3: $X_p \leftarrow W_p * X_c$
4: $Y_p \leftarrow \gamma * O(X_p) + \beta$    ▷ O is linear transformation such
    as BatchNorm
```

Algorithm 4 Passport-based watermarking – Training step

```
Require: Relative weight paramater $\lambda^r$
1: initialize a passport model $M_s$ with desired number of passport layers. $N_{pass}$
2: initialize passport keys $P$ in $M_s$
3: generate a random binary string to be embedded into signs of $\gamma_p$ of all passport layers
4: for number of training iterations do
5:     sample minibatch of $m$ samples $X$ $X^{(1)}, ..., X^{(m)}$ and targets $Y$ $Y^{(1)}, ..., Y^{(m)}$
6:     compute cross-entropy loss $L_c$ using $X$ and $Y$
7:     for $l$ in $N_{pass}$ do
8:         compute sign loss $R^l$ using $s^l$ and $\gamma_p^l$
9:     end for
10:    $R \leftarrow \sum_l^{N_{\text{pass}}} R^l$
11: end for
12: compute combined loss $L = L_c + \lambda^r R$
13: backpropogate using $L$ and update $M_p$
```

Algorithm 5 Passport-based watermarking – Sign Loss

```
Require: $B^l$, $W_p^l$, $P_\gamma^l$, $\gamma_0$
Ensure: loss
1: $\gamma^l \leftarrow \text{Avg}(W_p * P_\gamma^l)$
2: $loss \leftarrow \max(\gamma_0 - \gamma^l * B^l, 0)$ ▷ $\gamma_0$ is a positive constant, equals 0.1 by default
```

Algorithm 6 Passport-based watermarking – Signature detection

```
Require: $W_p$, $P_\gamma$
Ensure: loss
1: $\gamma \leftarrow \text{Avg}(W_p * P_\gamma)$
2: $signature \leftarrow sign(\gamma)$
3: convert $signature$ into binary
4: decode binarized $signature$ into desired format, e.g., ASCII
5: match decoded $signature$ with the target signature
```

A downside of this passport-based verification method is that it requires access to the passports during inference, which can lead to additional computational overhead and necessitates secure storage of the passports on the user's end. To address this, Fan et al. [5] propose two alternative verification approaches that eliminate the need to distribute the original passports with the trained model. These methods use multitask learning to jointly optimize the original task with passport regularization, while also minimizing the original task loss when the passport layers are excluded.

IV. Experiments

In this section, we demonstrate how the three watermark embedding techniques integrate watermarks into the model while preserving its performance. We describe the dataset, baseline model, and training settings, followed by an analysis of how the watermarks are embedded and their impact on model accuracy.

A. Evaluation Settings

*1) **Dataset**:* We use the CIFAR-10 dataset, which consists of 60,000 color images of size 32×32 pixels, distributed across 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The dataset is divided into 50,000 training images and 10,000 test images.

Prior to training, the dataset is preprocessed using normalization, which enhances training stability and convergence speed by ensuring all input features are scaled uniformly. This prevents any single feature from disproportionately influencing the learning process and allows the model to improve efficiently.

CIFAR-10 provides a diverse set of real-world images, making it well-suited for evaluating watermark embedding techniques across different categories. Its relatively small size ensures computational feasibility while still offering sufficient complexity for meaningful evaluation. Furthermore, the dataset's multiclass nature enables robust assessments of watermark effectiveness and resilience.

*2) **Baseline Model**:* We employ a convolutional neural network (CNN) inspired by the VGG [7] architecture, adapted for the smaller image resolution of CIFAR-10. The model consists of convolutional layers interleaved with max-pooling (M) and dropout (D) layers, followed by fully connected layers.

Convolutional Component: [64, M, D, 128, M, D, 256, 256, M, D, 512, 512]
Fully Connected Component: [1024, D, 1024]

Each convolutional layer is followed by batch normalization and a ReLU activation function. The convolutional layers use a 3×3 kernel with a padding of 1. Max-pooling layers aid in dimensionality reduction while preserving essential features, and dropout (p=0.5) mitigates overfitting by randomly deactivating neurons during forward passes. The fully connected layers utilize ReLU activation functions, with dropout applied between layers.

We select this CNN architecture due to its balance between complexity and computational efficiency. It is sufficiently sophisticated to support watermarking experiments while remaining feasible for training. Compared to simpler machine learning models, DNNs offer higher representational capacity, making them more suitable for embedding watermarks. Furthermore, CNNs are widely used in real-world applications, ensuring compatibility with various watermarking approaches.

*3) **Evaluation Metric: F1-Score**:* We employ the F1 score as a comprehensive metric for evaluating the performance of our watermarking detection model. The F1 score is a harmonic mean of precision and recall, providing a balanced measure of a model's predictive performance, particularly in scenarios with imbalanced datasets.

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (5)$$

Where:

- **Precision** measures the proportion of true positive predictions among all positive predictions: $\frac{\text{True Positives}}{\text{True Positives}+\text{False Positives}}$
- **Recall** measures the proportion of true positive predictions among all actual positive instances: $\frac{\text{True Positives}}{\text{True Positives}+\text{False Negatives}}$

In the context of watermark detection, the F1 score is particularly valuable because:

1) It provides a single metric that balances both false positives and false negatives.
2) It is robust to class imbalance in watermarked vs. non-watermarked models.
3) It offers a more nuanced evaluation compared to accuracy, especially when the distribution of watermarked models is uneven.

A high F1 score indicates that our watermark detection method consistently identifies watermarked models with both high precision (minimal false positives) and high recall (minimal false negatives).

B. Training Settings and Baseline Performance

The training dataset is split into an 80:20 ratio for training and validation. We train the model using the Adam optimizer with an initial learning rate of 0.001 and a cross-entropy loss function. A ReduceLROnPlateau scheduler is used to decrease the learning rate by a factor of 0.1 if the validation loss does not improve over 10 consecutive epochs. The batch size is set to 64, and the total training duration is 200 epochs. The learning rate is reduced at epochs 32, 42, and 55. The model is trained on an NVIDIA Tesla T4 GPU with 16GB of memory.

TABLE I: Validation and test accuracy/loss of the baseline model trained on CIFAR-10

Metrics	Accuracy	Loss
Validation	83.31%	0.5229
Test	83.95%	0.5771

C. Watermark Embedding Results

We evaluate the effectiveness (Table II) of three watermark embedding techniques and their impact (Fig. 6) on model accuracy.

*1) **Backdoor Trigger**:* Following the method described in Section 3.1, we embed a watermark by poisoning 10% of the training data. A 6×6 pixel square patch of random color is placed at a random position within the image, and these modified images are assigned a target label of seven (7). The model is then retrained from scratch under the same conditions as the baseline model, with learning rate reductions at epochs 78, 106, and 118. The training process is performed on an NVIDIA Tesla T4 GPU with 16GB of memory.

*2) **Weight Perturbation**:* As described in Section 3.2, we embed a watermark by modifying 0.1% of the model's trained weights. A small perturbation of ±0.001 is applied to these weight values to encode the watermark without significantly affecting model performance.

*3) **Signature Encoding**:* As described in Section 3.3, we embed a watermark by training the model weights with the given passports to encode a binary signature. The model is trained using the loss function defined in Eq. 1, which combines the cross-entropy loss for the main task with a signature loss weighted by $\lambda_r = 50$, and a regularization term with λ_0 (Eq. 4) set to the default value of 0.1.

TABLE II: Performance Metrics Comparison Between Baseline and Watermarked Models, Including Accuracy, Precision, Recall, and F-1 Score

Model	Accuracy	Precision	Recall	F-1 Score
Baseline	83.9%	0.84	0.83	0.84
Backdoor Trigger Scheme	79.5%	0.77	0.76	0.77
Weight Perturbation Scheme	83.7%	0.84	0.83	0.84
Passport-based Scheme	81.7%	0.82	0.82	0.82

Fig. 8 compares the F1 scores of the baseline model with those of the watermarked models. The backdoor trigger scheme shows a significant drop in performance, while the weight perturbation and signature encoding schemes maintain performance close to the baseline.

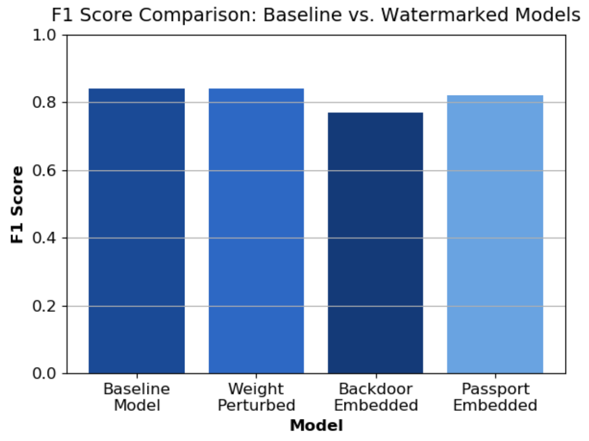

Fig. 8: Baseline model performance comparison with watermarked models

V. Robustness of Embedded Watermarks

We assess the resilience of the embedded watermarks against model compression techniques such as pruning and knowledge distillation.

1) Robustness Against Pruning: Pruning is a model compression technique that removes redundant or less significant parameters to reduce model size while maintaining performance. We apply L1-unstructured pruning, zeroing out weights with the lowest L1-norm. Specifically:

- 20% of convolutional layer weights are pruned.
- 40% of fully connected layer weights are pruned.

2) Robustness Against Knowledge Distillation: Knowledge Distillation compresses a model by training a smaller student network to replicate the behavior of a larger teacher network. We train a compact student model using KL-divergence loss to align with the soft predictions of the original model, alongside cross-entropy loss for hard label predictions.

The student model follows a similar architecture but with reduced capacity:

Convolutional Component: [32, M, D, 64, 64, M, D, 128, 128]
Fully Connected Component: [512, D, 512]

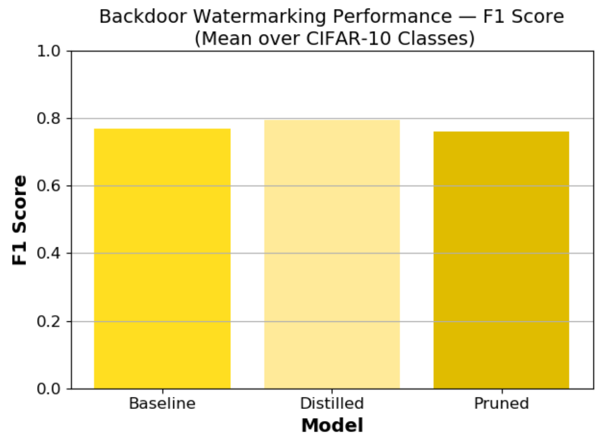

Fig. 9: Performance comparison of the baseline backdoor-watermarked model with compressed models

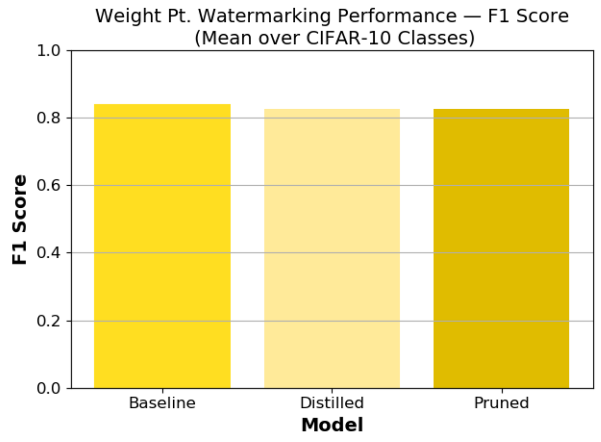

Fig. 10: Performance comparison of the baseline weight-perturbation-watermarked model with compressed models

VI. Result

Fig. 9 and 10 show the F1 scores of the baseline watermarked models compared to their compressed counterparts. The results indicate that both backdoor-triggered and weight-perturbed models retain their F1 scores after compression. While the watermark detection performance of the pruned model remains at 100% for the backdoor-triggered case and around 70% for the weight-perturbed case, both watermarking schemes fail under knowledge distillation. This is because training on a new, unpoisoned dataset during distillation diminishes the backdoor embedding (Fig. 11). Moreover, knowledge distillation produces a substantially different set of parameters, rendering the weight perturbation watermark ineffective (Fig. 12).

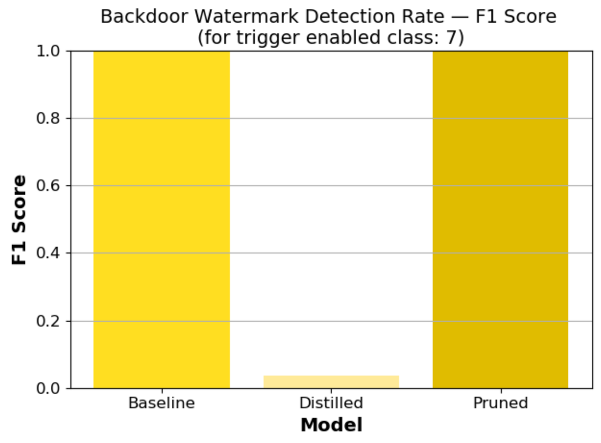

Fig. 11: Watermark detection performance of compressed watermarked models against the baseline backdoor-watermarked model

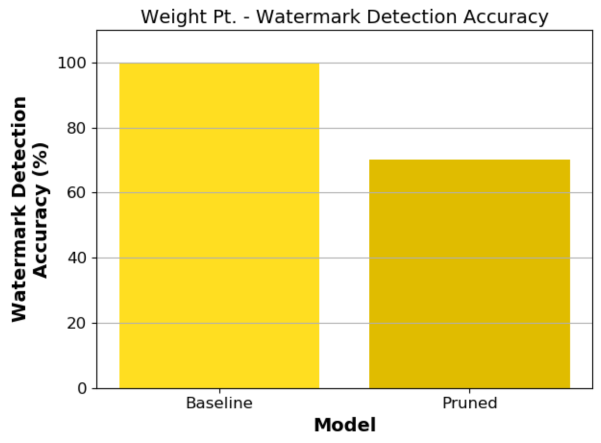

Fig. 12: Watermark detection performance of compressed watermarked models compared to the baseline weight-perturbation-watermarked model. Weight perturbation watermarking becomes ineffective after model distillation, as the knowledge transfer process generates a substantially different parameter set. The accuracy of distilled models is not shown as it no longer reflects the original watermarking mechanism.

For passport-based watermarking, the performance of the baseline watermarked model drops significantly when provided with forged or no passports, compared to when valid passports are used (Fig. 13). A similar trend is observed when the model undergoes compression via pruning or knowledge distillation. In terms of watermark detection, both the baseline watermarked model and the pruned model maintain a 100% detection rate. However, since knowledge distillation produces a substantially different set of parameters, it is no longer possible to reliably compare the extracted signatures between the baseline watermarked and distilled models.

Unlike the backdoor-triggered and weight-perturbed models, the inference performance of passport-based watermarked models degrades noticeably—even with valid passports after distillation—when compared to the uncompressed baseline. This degradation is intentional, as the passport mechanism is designed to modulate model performance based on the provided passport. Therefore, even in scenarios where signature detection and ownership claims may fail, the model owner can still render the model ineffective by withholding the valid passports.

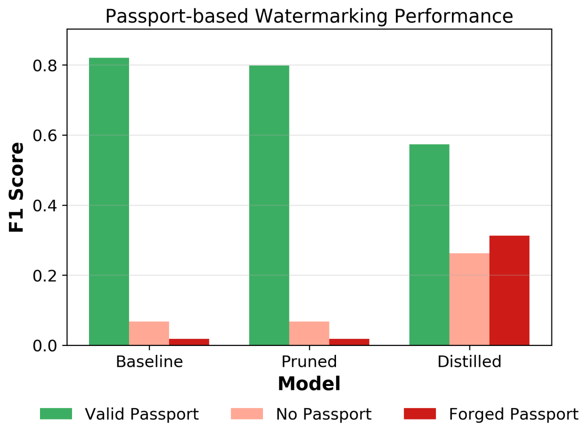

Fig. 13: Watermark detection performance of compressed watermarked models against the baseline backdoor-watermarked model

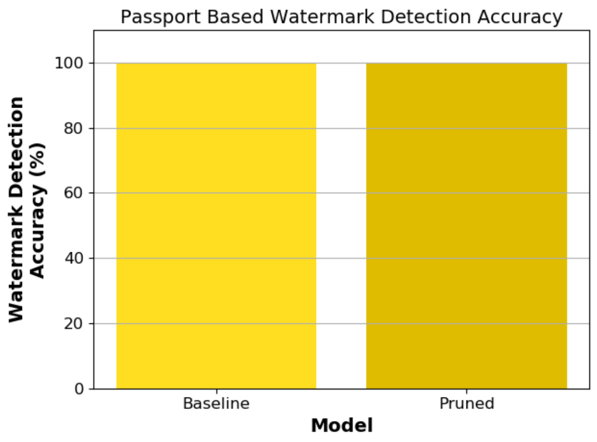

Fig. 14: Watermark detection performance of compressed watermarked models against the baseline backdoor-watermarked model

VII. Conclusion

In this study, we explored various watermarking techniques for deep learning models, including backdoor-based, weight perturbation, and passport-based methods. Our experiments demonstrate that while backdoor and weight perturbation approaches are relatively robust to model compression techniques such as pruning, they fail under knowledge distillation due to the significant changes in model parameters. Passport-based watermarking, although vulnerable to similar limitations in detection post-distillation, offers a unique advantage—its ability to degrade model performance in the absence of valid passports, thereby providing an effective deterrent against unauthorized usage. This highlights the importance of choosing a watermarking scheme aligned with the intended threat model and use-case requirements. Future work could focus on developing hybrid or adaptive watermarking strategies that maintain robustness under a wider range of model transformations.

References

[1] I. Lederer, R. Mayer, and A. Rauber, "Identifying appropriate intellectual property protection mechanisms for machine learning models: A systematization of watermarking, fingerprinting, model access, and attacks," *IEEE transactions on neural networks and learning systems*, vol. PP, 06 2023.

[2] Y. Uchida, Y. Nagai, S. Sakazawa, and S. Satoh, "Embedding watermarks into deep neural networks," in *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval*, ser. ICMR '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 269–277. [Online]. Available: https://doi.org/10.1145/3078971.3078974

[3] Y. Adi, C. Baum, M. Cisse, B. Pinkas, and J. Keshet, "Turning your weakness into a strength: Watermarking deep neural networks by backdooring," in *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, Aug. 2018, pp. 1615–1631. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/adi

[4] J. Zhang, Z. Gu, J. Jang, H. Wu, M. P. Stoecklin, H. Huang, and I. Molloy, "Protecting intellectual property of deep neural networks with watermarking," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, ser. ASIACCS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 159–172. [Online]. Available: https://doi.org/10.1145/3196494.3196550

[5] L. Fan, K. W. Ng, C. S. Chan, and Q. Yang, "Deepipr: Deep neural network ownership verification with passports," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 10, pp. 6122–6139, 2022.

[6] E. Soremekun, S. Udeshi, and S. Chattopadhyay, "Towards backdoor attacks and defense in robust machine learning models," 2023. [Online]. Available: https://arxiv.org/abs/2003.00865

[7] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1409.1556