Extending Null Embedding for Deep Neural Network (DNN) Watermarking

Improving the accuracy of the original classification task in piracy-resistant DNN watermarking

Kaan Altınay[1]

Supervisor(s): Dr. Zeki Erkin[1]**, Devriş İşler**[2,3]

[1]EEMCS, Delft University of Technology, The Netherlands
[2]IMDEA Networks Institute
[3] Universidad Carlos III de Madrid, Spain

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 22, 2024

Name of the student: Kaan Altınay
Final project course: CSE3000 Research Project
Thesis committee: Zeki Erkin, Devriş İşler, Asterios Katsifodimos

An electronic version of this thesis is available at http://repository.tudelft.nl/.

Abstract

The advancement of Machine Learning (ML) in the last decade has created new business prospects for developers working on ML models. Models that are expensive and time-consuming to design and train can now be outsourced from others to reduce costs using Machine Learning as a service (MLaaS). **Deep Neural Networks (DNNs)** are particularly expensive to train, therefore many who need a DNN utilize the services of an MLaaS provider. This creates the **possibility of piracy** of this valuable asset, and the need to prevent piracy to assure a fair market. To address this need, research has been conducted on protecting DNNs using various watermarking techniques. A work by *Li et al.* has proposed null-embedding, a technique that renders the DNN useless if it is subject to a piracy attack. Despite being effective, this method was shown to reduce classification performance when embedding a watermark into the model. This paper suggests modifications to the null-embedding technique that reduce this impact and keep the classification accuracy close to that of a non-watermarked model.

1 Introduction

As Machine Learning (ML) becomes more prominent and is used for an increasing number of tasks, ML developers have built business models around outsourcing their models [4]. Since ML models are costly to train, both in terms of time and effort, many users opt to use third-party services to cover their ML needs [8]. Machine Learning as a service (MLaaS) brings with it the risk of piracy of expensive deep neural network (DNN) models. To protect DNNs against theft and piracy, the watermarking research community has recently suggested techniques to provide proof of ownership for a given DNN model.

Existing research can be classified into two broad categories: white-box and black-box verifiable watermarking techniques [5]. The former assumes the verifier has white-box access to the model under investigation. In scenarios where a third-party is suspected to have pirated a model, this assumption does not hold. Instead, watermarks that can verify ownership of a model with only black-box access (i.e. using only input and output) are needed to prove ownership.

Most black-box verifiable watermarking techniques [14] [5] stem from the idea of using 'backdoor attacks' on DNNs to embed the watermark. These techniques add watermarks to the training data. This enables the model to classify certain watermarked data points to a different label than it normally would, hence verifying a watermark exists. But this type of watermarking can cause disputes in ownership if someone else discovers this backdoor. Therefore *Li et al.* suggested a novel technique that utilizes null embedding to embed the watermark [7].

Even though this novel technique is resistant to piracy attacks, it reduces the accuracy of the DNN's original classification task by a factor of up to 1.5% [7], which is an undesirable increase in error for accuracy-critical DNN applications. This can be related to the fixed, square-shaped filters used for null embedding (detailed description of null embedding is presented in Section 3.2). This paper aims to improve the DNN's classification accuracy while maintaining the robust ownership watermark. Our contribution can be summarized as:

- Introducing four new methods of null-embedding a watermark to image data.
- Assessing the classification accuracy of DNN models trained with data watermarked using the said methods.
- Comparing the accuracy of DNNs watermarked with the new methods with the original method proposed by *Li et al.*

By showing that the modified methods increase classification accuracy, this paper strengthens the existing piracy-resistant DNN watermarking technique and brings it closer to deployment in the industry.

The report is structured as follows: Section 2 outlines previous research in the area and highlights critical papers. Section 3 explains key terms and concepts, and Section 4 describes our modifications to the state-of-art. Section 5.1 states the experimental procedure and results are presented in Section 5.2. Section 6 discusses the significance of the results and presents ideas for future work. Section 7 outlines the ethical implications and Section 8 contains concluding remarks.

2 Previous Work

This section briefly describes other papers in the field that have been published and points out their significance.

The first paper on DNN watermarking was by *Uchida et al.* and it proposed a white-box verifiable technique based on embedding a watermark in the parameters of the model [11]. This paper was later taken as the benchmark for requirement definitions such as fidelity and robustness in the DNN watermarking domain. Soon after this suggestion of a white-box verifiable DNN watermark, *Zhang et al.* proposed a black-box verifiable DNN watermarking method [14]. Their technique used an idea from backdoor attacks on DNNs to embed the watermark. A train image is embedded with a pattern and this image has a label that it should not have (ex. a car image embedded with a watermark results in the label 'Plane').

Despite its limitations in terms of verification, white-box watermarking techniques were further investigated. The reason for this is that studies have shown that every black-box watermarking technique will negatively influence model accuracy as they modify the training set [9]. This makes them inapplicable to accuracy-critical DNN applications,

such as those used in medical diagnosis. One such study by *Wang et al.* uses another DNN trained using an adversarial network to perform watermark extraction from the white-box watermarked original network [12].

Most black-box watermarking techniques have built upon the backdoor attack technique proposed in [14]. *Guo et al.* has investigated the requirements of DNN watermarking in an embedded systems context [5]. Their work proposes making use of cryptography by incorporating the author's signature into the training process. *Xu et al.* has extended watermarking using the backdoor attack technique to Graph Neural Networks (GNNs) [13]. *Szyller et al.* has developed a watermarking technique that is specifically resiliant against model extraction intellectual property theft [10].

None of the research above addresses the threat of piracy like *Li et al.* does in their paper. *Li et al.* suggests null embedding, a technique that uses a bit sequence to build a strong dependency between the normal classification accuracy of the model and the watermark. This makes it impossible for attackers to remove the watermark or insert their own. This unique resistance against piracy is why their technique was chosen to study further in this paper.

3 Preliminaries

This section explains the main concepts relevant to the research topic.

3.1 Digital Watermarking

The origins of electronic watermarking can be traced back to 1954, when the Muzak Corporation filed a patent describing a method for embedding a pattern into a music track for purposes of proving ownership [3]. Digital watermarking became a more significant research topic with the rise of the technology age and the web, allowing the distribution of knowledge instantaneously [2]. The main idea of watermarking remains the same: Insert data that is imperceptible in normal use, but that can be used later to verify ownership. It is essential that a watermark does not hinder the original task of the data, is robust against modifications in the data, and is easily verifiable after embedding.

In the 2000s, researchers started looking into watermarking of non-media assets such as map databases and 3D models [6] [1]. Watermarking DNNs is a newer area and has started attracting research in the 2010s.

3.2 Null and True Embedding

As mentioned in Section 2, this paper is based on the technique of null embedding a watermark into the training data of a DNN proposed by Li et al. [7]. The algorithms used for embedding are derived from the pseudocode given in their paper, therefore we refer readers who are interested in further details of the embedding/verification procedure to their paper. Nonetheless, we provide a high-level description below.

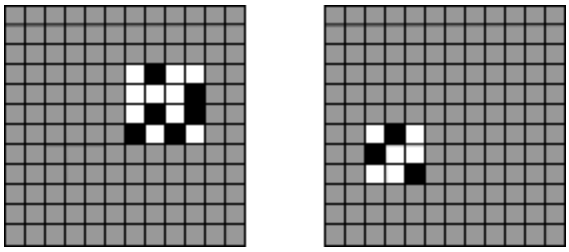

Figure 1: 4x4 and 3x3 Null Embedding Patterns on a 12x12 Image. White squares are at value $\lambda = 2000$, blacks at $-\lambda$.

Null embedding consists of applying a filter to a matrix-like data format which disguises part of the matrix as having extreme values outside the normal dataset range. This effectively reduces the learning space of the model. For example, a 32x32 image is null-embedded by a 6x6 square filter with pixel color values of either $\lambda = 2000$ or $-\lambda = -2000$, depending on the bit pattern. Because both values are far from the normal color range of [0, 255], these pixels no longer serve a purpose in the image classification. A sample illustration is shown in Figure 1.

The model owner generates a watermark as a tuple of the square's position, and the bit pattern embedded in it. The owner first signs a verifier string *v*, which is a combination of an arbitrary string and the current timestamp. Afterwards, this signed string is hashed and its modulos determine the tuple of *(bits, pos)*, as shown in Algorithm 1. S is the length of one side of the square filter to be embedded, h and w are the dimensions of the matrix being modified.

Algorithm 1 Generate Watermark (str, h, w, S)

$v \leftarrow$ str + timestamp
$sig \leftarrow$ Sign($PrivKey, v$)
$bits \leftarrow$ hash(sig) mod 2^{n^2}
$pos(x, y) \leftarrow$ (hash(sig) mod (h - S), hash(sig) mod (w - S))

Verification of the watermark is achieved by first checking if the owner's public key can verify the signature. Afterwards, a new set of watermarked data points are generated. This can be done by anyone who has access to the generation algorithm described in Algorithm 1. After generation, the trained model is asked to guess labels for the newly generated data points. The verification succeeds if the model returns an accuracy higher than a certain pre-determined threshold T.

Algorithm 2 Verify Watermark ($PubKey$, str, h, w, S, T)

if Verify($PubKey$, sig) **then**
  $(bits, pos) \leftarrow$ Generate Watermark (str, h, w, S)
  $NewData \leftarrow$ Embed Watermark (bits, pos)
  if [Accuracy of Model on $NewData$] $\geq T$ **then**
    Verification Success
  end if
end if

Algorithm 2 above demonstrates the verification procedure. Any outcome that does not lead to Verification Success is not included in the algorithm, and verification fails.

In the original paper, verification of a watermark in a model is supported by embedding some images with the inverse of the bit pattern used for null embedding. These images are then mapped to a specific label (not equal to their original label). This process is called true embedding, and is supposed to help verify the existence of a watermark in cases where the null embedding alone can yield false positives. From our experimentation, we have seen that the null embedding is a very reliable method to verify a watermark, and that adding this true embedded data is not necessary for verification.

4 Modifications of Null Embedding

To improve the classification accuracy of the DNN while maintaining a watermark, we have experimented with varying the shape of the embedding pattern on the image. Because a regular shape like a square has the potential to hide entire features of a matrix-like data structure, irregularizing the shape of the pattern can reduce the number of significant features left outside the training domain. The filter patterns tested are the following four: *Random, Peripheral, Circular*, and *Triangular*.

To keep the training domain at a similar size to images embedded with the original square pattern, all filters below were made to have the same pixel count (or close, if the same is not possible) as the original pattern.

4.1 Random Filter

The random embedding pattern chooses the same number of pixels as the square filter (36 in our test scenario) at random throughout the image. This has the effect of not blocking any significant features for the human eye, but is very similar to introducing noisy training samples into the training dataset. This can cause lower verifiability of the watermark as it is not perceivable, and potentially harm the classification accuracy of the DNN. A visualization is shown in Figure 2.

4.2 Peripheral Filter

This filter is constructed by selecting the top-most and left-most pixels possible in the image for pattern embedding. This pattern was chosen as the extremes of a matrix-like data structure can be cropped without disturbing the coherency of the rest of the data. A sample visualization is shown in Figure 2.

Especially in image datasets like the ones used in this paper, the peripherals of the image tend to not have parts of the object being labeled from the image. This should mean that the classification accuracy is affected less compared to other methods of reducing the training domain.

4.3 Circular Filter

To embed this filter, a circle is chosen with a center pixel determined similarly to the filter position determination

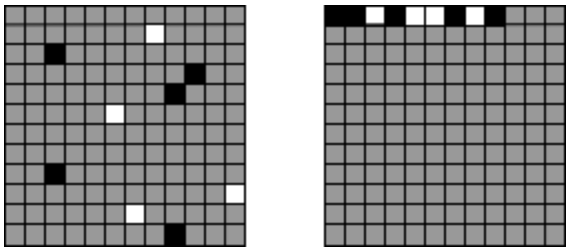

Figure 2: Random and Peripheral Null Embedding Patterns on a 12x12 Image. White squares are at value $\lambda = 2000$, blacks at $-\lambda$.

described in Algorithm 1. Afterwards, the radius of a circle with equal surface area to the square is calculated, and pixels with centers within this circle are used to embed the bit pattern. Algorithm 3 below demonstrates this procedure.

This method was chosen as it will have the effect of minimizing the perimeter of the watermark shape. As DNNs use adjacent pixels to learn patterns, we believe this will prove to result in better classification accuracies in comparison to a square filter. A sample visual can be found in Figure 3.

Algorithm 3 Embedding a Circular Watermark

```
(MSB = Most Significant Bit)
pixelcount ← Area Of Square Filter
radius ← √(pixelcount/π)
for image in subset do
    for (i, j) in image do
        if (i,j) is in circle then
            pixel value ← MSB of bit pattern
            shift bit pattern to the left to change MSB
        end if
    end for
end for
```

4.4 Triangular Filter

The triangular filter uses a triangle with an area less than or equal to that of the square to embed the bit pattern. Its reason for inclusion is to see if there are significant classification differences between using different regular polygon shapes for null embedding.

Data in matrix form does not tend to have features that can be derived from a diagonal along the matrix. Therefore, we predict that it is less likely for a triangle to block a significant feature in comparison to a square.

$$k = \frac{-1 + \sqrt{1 + 8n}}{2} \tag{1}$$

The number of rows in the triangular filter is determined by Equation 1 above. The goal is to create the largest triangle where the ith row has i elements, with the number of pixels not exceeding the number of pixels in the equivalent square

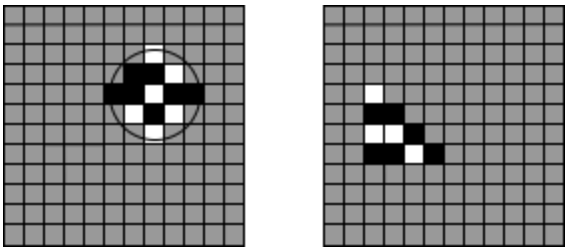

Figure 3: Circular and Triangular Null Embedding Patterns on a 12x12 Image. White squares are at value $\lambda = 2000$, blacks at $-\lambda$.

filter. Given the number of pixels in the square filter as n, we want to find the largest integer k such that:

$$\frac{k(k+1)}{2} \leq n \tag{2}$$

Rearranging the inequality in Equation 2 gives $k^2 + k - 2n = 0$, and plugging $a = 1$, $b = 1$, $c = -2n$ into the quadratic formula yields Equation 1:

$$k = \frac{-1 \pm \sqrt{1^2 - 4 \cdot 1 \cdot (-2n)}}{2} = \frac{-1 + \sqrt{1 + 8n}}{2} \tag{3}$$

Algorithm 4 below shows how this equation is used in the embedding of the watermark.

Algorithm 4 Embedding a Triangular Watermark

```
(MSB = Most Significant Bit)
pixelcount ← Area Of Square Filter
rowcount ← Equation 1 where n = pixelcount
for image in subset do
    for i in range rowcount do
        for j in range i do
            pixel value ← MSB of bit pattern
            shift bit pattern to the left to change MSB
        end for
    end for
end for
```

5 Experimental Setup

In this section, we describe our experimental procedure and an overview of the results obtained. The datasets used for demonstrating proof of concept is the University of Toronto's publicly available *CIFAR-10* dataset[1] and the MNSIT handwritten digit dataset. The former consists of 60000 images with 10 class labels, and the latter of 70000 images with 10 class labels (corresponding to 10 digits). Experiments were performed on a personal workstation, an HP ZBook Power G7 with an NVIDIA Quadro T1000 GPU.

5.1 Experimental Procedure

The DNN was constructed using the Tensorflow Keras framework available in Python, using version 2.16 of Tensorflow. Windows Subsystem for Linux (WSL2) was used to support Tensorflow's GPU capabilities as Tensorflow has discontinued GPU device support on Windows systems in version 2.10.

Data Split and DNN Structure

Both datasets were split according to the recommended dataset split by the dataset providers, and these are as follows:

The CIFAR-10 dataset was split to have 50000 data points in the training set and 10000 in the validation set. The MNIST dataset was split to have 60000 data points in the training set and 10000 in the validation set. 10% of the training dataset was selected randomly before each training to be embedded with the watermark. With the addition of the watermarked data, the total training set has a size of 55000 for CIFAR-10 and 66000 for MNIST.

To be in keeping with the experimental setup used by [7], the DNN for training on the CIFAR-10 dataset was constructed with 6 convolutional and 3 dense layers. The DNN for training on the MNIST dataset was constructed with 2 convolutional and 2 dense layers. Further detailed specifications can be found in the Appendix of [7].

The CIFAR-10 DNN was trained for 50 epochs in each round, while the MNIST DNN was trained for 20 epochs. This is because the MNIST dataset is a lot simpler, and thus the DNN achieves higher accuracies quickly. For both datasets, the results obtained from 5 rounds of training were averaged out for each configuration presented in the following section.

Trials

Before experimenting with the proposed filter types mentioned in Section 4, the square filter used by *Li et al.* was implemented following the pseudocode procedure in their paper. The results obtained from the DNN watermarked with the square pattern were then used as the benchmark to compare the performance of the proposed pattern varieties. Additionally, the model was trained with the same parameters but without a watermark to be able to compare the loss of accuracy on the validation set.

The DNN model was watermarked using training data embedded with the Random, Peripheral, Circular, and Triangular null-embedding patterns. The results from these trials, the square pattern watermarked DNN, and the non-watermarked DNN are present in the tables in Section 5.2

5.2 Results

The experiments focus on 6 measures. These can be categorized into 3 as each of the 3 categories mentioned in the following subsections have their respective accuracy and loss measures. The accuracies are presented in Table 1 in percentages, with the standard deviations given after the plus/minus sign, i.e. $mean \pm SD$. The loss function used is Sparse Categorical Cross-Entropy. This loss function was

[1] CIFAR-10 and CIFAR-100 datasets. https://www.cs.toronto.edu/ kriz/cifar.html

Table 1: Accuracy and Loss for Each Watermarking Method. Best accuracy values are bolded.

CIFAR-10	Training Data		Validation Data		Watermark Verification	
	Accuracy (%)	Loss	Accuracy (%)	Loss	Accuracy (%)	Loss
No Watermark	98.56±0.12	0.0415±0.0032	82.91±0.19	0.8503±0.0118	12.14±1.26	73.6109±27.7472
Square WM	98.41±0.39	0.0467±0.0114	82.44±0.39	0.8569±0.0496	96.67±1.78	0.1077±0.0607
Random WM	98.36±0.23	0.0481±0.0070	82.61±0.20	0.8423±0.0152	66.48±5.30	1.5524±0,3326
Peripheral WM	98.74±0.15	0.0367±0.0044	82.60±0.30	0.8823±0.0164	**99.50±0.18**	0.0170±0.0063
Circular WM	98.56±0.28	0,0419±0.0090	**83.01±0.27**	0.8295±0.0373	95.50±1.17	0.1429±0.0426
Triangular WM	98.73±0.08	0.0379±0.0017	82.79±0.17	0.8720±0.0164	99.44±0.25	0.0175±0.0075
MNIST	Training Data		Validation Data		Watermark Verification	
	Accuracy (%)	Loss	Accuracy (%)	Loss	Accuracy (%)	Loss
No Watermark	99.86±0.03	0.0047±0.0007	99.42±0.03	0.0195±0.0015	95.15±4.37	0.1660±0.1729
Square WM	99.83±0.07	0.0053±0.0022	99.43±0.03	0.0192±0.0012	99.60±0.52	0.0128±0.0157
Random WM	99.85±0.03	0.0051±0.0006	**99.44±0.03**	0.0191±0.0005	99.17±0.67	0.0252±0.0198
Peripheral WM	99.87±0.01	0.0042±0.0002	99.41±0.04	0.0206±0.0009	99.98±0.02	0.0011±0.0003
Circular WM	99.83±0.06	0.0052±0.0015	99.38±0.03	0.0215±0.0015	99.62±0.32	0.0124±0.0102
Triangular WM	99.88±0.02	0.0045±0.0005	99.43±0.03	0.0197±0.0007	**99.99±0.01**	0.0009±0.0002

chosen to keep the results comparable with the experimentation performed in [7].

After each training round, the epoch after which the model yielded the highest accuracy for the validation set was used as the representative data for that round. This epoch was usually between the 40th-50th for the CIFAR-10 dataset and the 15th-20th for the MNIST dataset. The reason that later epochs sometimes perform worse than earlier epochs is the model overfitting to the training set after a certain number of training epochs.

Table 1 above shows the average values from 5 trials for each watermarking method. The results from the models on the training data, validation data, and watermark verifiability are explained below.

Results from Training Data

The accuracy and loss of each embedding method on the training data are shown in Table 1 under *Training Data*. For models trained for the same number of epochs on the same size dataset, these values are expected to be similar regardless of the embedding method. They were included for completeness and verifiability of a replication of the experimentation.

Results from Validation Data

The accuracy and loss of each embedding method on the validation data are shown in Table 1 under *Validation Data*. This data is the most interesting for answering the research question as comparing the accuracy of the watermarked models with that of the non-watermarked model will show how effective the proposed modifications are in reducing accuracy loss in comparison to the square filter that was originally proposed.

Results on Watermark Verifiability

The accuracy and loss of each embedding method in detecting the watermark are shown in Table 1 under *Watermark Validation*. This data is important as it is a hard requirement for any watermarking method to be verifiable in the models they are embedded in. If a watermarking method yields excellent accuracy in the original classification task, but is not verifiable, then that is not a functioning watermark.

6 Discussion

This section provides an analysis of the results obtained from the experiments in 6.1, including a comparison with the results from [7] and the effectiveness of the newly proposed watermarking methods. Afterwards, recommendations for researchers on how to continue and extend this work is given in 6.2.

6.1 Analysis

Before discussing the results, it must be noted that some watermarked models have a higher accuracy than the non-watermarked (non-WM) model on the training set. This indicates that the high number of epochs used for training has made the accuracies of watermarked and non-WM models indistinguishable. Therefore, we can assume that the training set accuracy has converged and would not benefit from further training.

It should also be noted that the Random watermarked model performed significantly worse than other watermarked models in terms of watermark verifiability. Therefore, it is excluded from verifiability threshold calculations for either dataset.

From the results, it is clear that some of the proposed

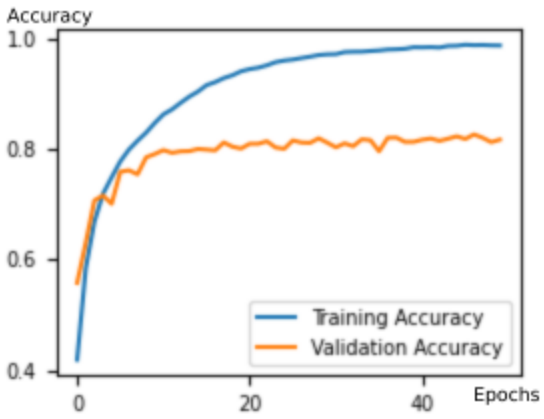

Figure 4: Accuracies of square watermarked DNN on the CIFAR-10 dataset over 50 epochs. Y-axis begins at 0.40.

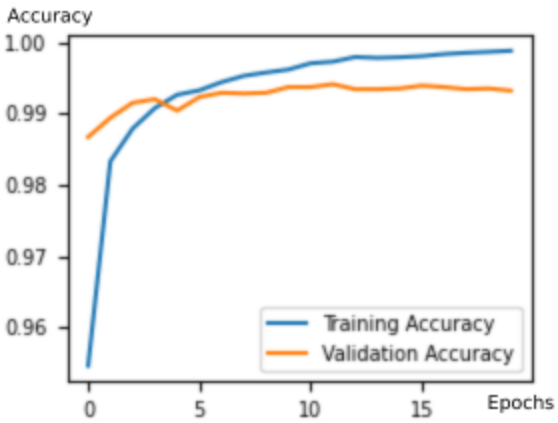

Figure 5: Accuracies of square watermarked DNN on the MNIST dataset over 20 training epochs. Y-axis begins at 0.95.

null-embedding patterns perform better than the original square watermark in several aspects.

Analysis of Results from the CIFAR-10 Dataset

From the models trained with the CIFAR-10 dataset, the model watermarked using a circular pattern has yielded the best classification accuracy on the validation data. Its average accuracy is higher than that of the non-watermarked model, but it falls within one standard deviation (SD) of the non-watermarked model's accuracy. This suggests that the classification accuracies of the models are statistically comparable and could be used interchangeably.

The validation classification accuracy of the model watermarked using the triangular pattern is also within one SD of the accuracy of the non-WM model, and has a lower SD than the model watermarked using the circular pattern. This consistency can make it preferable over the circular pattern watermark. Regardless, both the triangular and circular patterns have yielded lower SDs than the square pattern.

The convergence speed of a square watermarked DNN model to a validation accuracy value of around 82% can be seen in Figure 4, this trend holds for all the tested methods.

The results show that a verification threshold T=90% is appropriate for use. In terms of verifiability, the peripheral pattern has yielded the best result, with the triangular pattern being a close second. Therefore, the triangular pattern is the best pattern choice overall.

Analysis of Results from the MNIST Dataset

As the MNIST dataset is much simpler than the CIFAR-10 dataset, the classification task was significantly easier for the trained DNNs. Therefore, it can be seen in Figure 5 that the training set classification accuracies quickly converge to values higher than 99%. A similar trend is observed in the validation set accuracy: all watermarking methods have yielded an accuracy of around 99.4%.

The fact that almost all of the validation accuracies lie within an SD of each other means that it is difficult to infer much from these values. A look at the losses shows that the Square and Random watermarked models have performed closest to the non-WM model. Even though the Random watermarked model appears to have the most accurate and reliable validation set classification, its performance in verifiability is significantly lower than the other watermarked models.

The results show that a verification threshold of T=99% is appropriate for use. The Triangular watermarked model has the best watermark verifiability of the tested methods. Combined with the validation accuracy close to the non-WM model, it is the best choice out of the models tested.

Comparison with Results from Li et al.

When compared with the original results presented in [7], our results highlight a few significant differences.

Firstly, the disagreement between the final normal classification (NC) accuracies of the datasets must be noted. Even though we followed the procedure outlined in [7] to the best of our ability, our DNNs have not been able to achieve the CIFAR-10 NC accuracy of around 88%. This can be explained by the small number of epochs (50), but training up to 100 epochs has not been able to surpass an average of 84% during our trials. A more powerful machine and more training epochs could resolve this discrepancy.

A disagreement towards the other side is observed in the MNIST NC accuracies. Our models have quickly yielded NC accuracies above 99%, while [7] quotes a 98.7% accuracy for the non-WM model.

The original paper has used an embedding rate of 50% for the MNIST dataset, yet our trials have shown that this does not lead to an improvement neither in validation set accuracy nor in watermark detection. Therefore, we used the same embedding rate of 10% for models trained on both datasets.

Based on the results gathered from models trained on the CIFAR-10 dataset, we observe that the Circular and Triangular watermarked models yield around a 0.5% increase in NC accuracy over the Square watermarked model. While small, this does show that using a different shape than a square for null embedding is better to preserve the classification accuracy of the DNN on the original task.

6.2 Future Work

This paper has proposed new ways of null embedding a watermark into a DNN, but has not been able to conduct the robustness analysis present in [7]. Even though there is no reason to suspect that changing the shape of the embedding pattern will affect the robustness of the watermark against piracy attacks, this should be checked in a follow up to this work for completeness. This work should include testing whether transfer learning, fine-tuning, and model compression affect the accuracy of watermark detection.

We have only tested the modified watermarking techniques on two fairly simple image datasets. Future work should test whether the results hold with larger image datasets and with matrix-like data formats that are not images.

With the currently used technique, embedding the watermark entails enlarging the training set of the DNN by 10%, leading to an increase in model training times by an equivalent amount. This is not an issue for small datasets used for a proof-of-concept in this research. But for businesses outsourcing their DNNs, a 10% overhead in training time can be too much of a compromise for the added security of a watermarked model. Future research should look into methods of embedding a piracy-resistant watermark that does not introduce such a large training overhead.

7 Responsible Research

Throughout this paper, we have paid special attention to preserving clarity and transparency for the reader. Work that has been borrowed or inspired by others has been cited, and our contributions have been described in detail to allow reproducibility. Additionally, the Python code used to run the experiments can be found at https://github.com/kaanaltinay/rp_kaltinay. The datasets used are public image datasets commonly used for proof-of-concept research. Therefore, there is no cause for privacy or copyright concerns for any party. Finally, as the research subject concerns digital security, the lack of robustness analysis on the newly proposed methods has been made clear and has been suggested as future work in Section 6.2.

8 Conclusion

This paper has introduced four new methods of embedding a watermark into a DNN model using the null embedding technique described by [7]. The results from testing these methods on models trained using the CIFAR-10 dataset indicate that the circular pattern for null embedding a watermark into a model is the best out of the proposed methods in order to preserve the classification accuracy of the DNN on the original task. This yields a classification improvement of around 0.5% over the model watermarked using the square pattern, making it as accurate as the non-watermarked model. When watermark validation is taken into account, the triangular pattern presents the best trade-off for models trained both on CIFAR-10 and MNIST datasets.

References

[1] O. Benedens and C. Busch. Towards blind detection of robust watermarks in polygonal models. *Computer Graphics Forum*, 19(3):199–208, 2000.

[2] Ingemar J. Cox and Matt L. Miller. The First 50 Years of Electronic Watermarking. *EURASIP Journal on Advances in Signal Processing*, 2002(2):1–7, December 2002. Number: 2 Publisher: SpringerOpen.

[3] Hembrooke Emil Frank. Identification of sound and like signals, October 1961.

[4] I. Grigoriadis, E. Vrochidou, I. Tsiatsiou, and G.A. Papakostas. Machine Learning as a Service (MLaaS)—An Enterprise Perspective. *Lecture Notes in Networks and Systems*, 552:261–273, 2023. ISBN: 9789811966330.

[5] Jia Guo and Miodrag Potkonjak. Watermarking Deep Neural Networks for Embedded Systems. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8, November 2018. ISSN: 1558-2434.

[6] Sanjeev Khanna and Francis Zane. Watermarking maps: hiding information in structured data. In David B. Shmoys, editor, *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, January 9-11, 2000, San Francisco, CA, USA*, pages 596–605. ACM/SIAM, 2000.

[7] Huiying Li, Emily Wenger, Shawn Shan, Ben Y. Zhao, and Haitao Zheng. Piracy Resistant Watermarks for Deep Neural Networks, December 2020. arXiv:1910.01226 [cs, stat].

[8] M. Ribeiro, K. Grolinger, and M.A.M. Capretz. MLaaS: Machine learning as a service. pages 896–902, 2016.

[9] M. Shafieinejad, N. Lukas, J. Wang, X. Li, and F. Kerschbaum. On the Robustness of Backdoor-based Watermarking in Deep Neural Networks. pages 177–188, 2021.

[10] S. Szyller, B.G. Atli, S. Marchal, and N. Asokan. DAWN: Dynamic Adversarial Watermarking of Neural Networks. pages 4417–4425, 2021.

[11] Y. Uchida, Y. Nagai, S. Sakazawa, and S. Satoh. Embedding watermarks into deep neural networks. pages 269–277, 2017.

[12] Tianhao Wang and Florian Kerschbaum. RIGA: Covert and Robust White-Box Watermarking of Deep Neural Networks. In *Proceedings of the Web Conference 2021*, WWW '21, pages 993–1004, New York, NY, USA, June 2021. Association for Computing Machinery.

[13] Jing Xu, Stefanos Koffas, Oguzhan Ersoy, and Stjepan Picek. Watermarking Graph Neural Networks based on Backdoor Attacks, November 2022. arXiv:2110.11024 [cs].

[14] J. Zhang, Z. Gu, J. Jang, H. Wu, M.Ph. Stoecklin, H. Huang, and I. Molloy. Protecting intellectual property of deep neural networks with watermarking. pages 159–171, 2018.