Bachelorarbeit am Institut für Informatik der Freien Universität Berlin,
Arbeitsgruppe ID Management

Evaluating and Adapting Existing Neural Network Watermarking Approaches to Online Learning Scenarios

Di Wang

diw00@zedat.fu-berlin.de
Matrikelnummer: 5429756
Betreuerin: Franziska Boenisch
1. Gutachter: Prof. Dr. Marian Margraf
2. Gutachter: Prof. Dr. Gerhard Wunder

Berlin, den 24.08.2021

Eidesstattliche Erklärung

Ich versichere hiermit an Eides statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Berlin, den 24.08.2021

Di Wang

Abstract

To protect machine learning (ML) algorithms that are trained using expensive computational power and time, watermarks are applied on neural networks (NN) to prevent thefts of intellectual property. To further counteract the attempts at removing or forging watermarks, the need to make watermarks robust arises. With the growing amount of training data and the continuous nature of data production in time, online learning algorithms become also more relevant. This work investigates how to keep watermarks in NNs verifiable under online learning conditions, and proposes three strategies to keep watermark accuracies high during online learning scenarios; Namely re-feeding by filtering wrongly predicted watermarks, re-feeding by filtering watermarks predicted with low confidence, and re-feeding at constant interval of steps. Two watermark embedding approaches from previous work, namely ingrainer approach from Yang [30] and exponential weighting approach from Namba [21], are used to watermark models for experiments, providing the conclusion that watermark re-feeding strategies need to be adapted to the particularities of specific embedding approaches to keep watermark accuracies high. It is shown that under online learning conditions, watermark accuracies of protected NNs can be maintained, however to differing degrees and through different strategies. For the *ingrainer* watermark embedding approach proposed by Yang [30], re-feeding by filtering watermarks predicted with low confidence works the best. For the *exponential weighting* approach proposed by Namba [21], the best re-feeding strategy turns out to be using 60% watermark and 40% of main training data at every 10^{th} steps of online learning.

Contents

1 Introduction

1.1 Motivation

The time and computational power needed to train an NN call for copyright protection against thefts of ML algorithms. Watermarks are applied to NNs in response to this need in security. The survey by Boenisch [3] categorizes watermark embedding approaches for NNs into two broad types, the first one being embedding watermark into NN internal properties, and the second one being using a specific trigger dataset and the model's special predictions on it. According to Li [17], this second type of approach can also be called dynamic watermarking.

Some approaches proposed in current research are examined, and tests showed that they are robust in certain specific test settings. Robust here means that they are able to withstand watermark removal or watermark forging attacks by correlating the main task part of the NN with the watermark component. However, these approaches have not yet been set in the context of online learning, where the data are put into the model not as a single batch, but in continuous stages or as continuous streams. This presents additional challenge to the robustness of the watermark because as new data comes in, the NN evolves with it, and the presence or the accuracy of the watermark can no longer be guaranteed. With watermarked models examined so far only from the perspective of re-training or fine-tuning attacks, this work looks at supplying watermark embedded NNs with new data from the perspective of online learning, where the model should still be continuously functional as new data is fed in.

With the increasing amount of available training data, it is no longer realistic for models to process and hold a large batch of training data at the same time. Moreover, for most tasks in companies, data comes in over time and new data is always generated. This trend calls for more application of online learning, which modifies the model in a way that is similar to a fine-tuning attack, diminishing the presence of the watermarks. This thesis aims at analyzing the robustness of existing watermarking approaches for NNs under an online learning environment, and proposes new schemes that combine watermark reinforcement with online learning, upholding the accuracy of watermark predictions in NNs under online learning scenarios.

1.2 Problem Statement

Two main goals are set out from this thesis. The first goal is to evaluate the robustness of the proposed NN watermarking approaches in an online learning setting. Since watermarks are supposed to be resistant to fine-tuning or transfer learning attacks when well-embedded in the NN, they should correspondingly also display robustness when new training data are continuously fed into the network. The second goal then is to put forward new strategies that ensure high watermark accuracies of NNs during online learning.

Specifically, the first goal answers the question, after inputting new data into the neural network, which could possibly result in a shift in the distribution of the entire dataset, can the watermarks still be retrieved with a sufficient accuracy from the NN. The claims made by the authors of the examined approaches are checked based on the evaluation of the performance of their proposed approaches under online learning conditions.

For the second goal, several new strategies are proposed to address the issue of the loss of watermark accuracies, which ensure that the watermarks would always be able to be retrieved from the NN with a high enough probability, even when the current set of training data is continuously changing. To reinforce the watermarks, they need to be identifiable throughout stages of new data input, and throughout epochs of training. The same evaluation is then applied to the new strategies of refeeding watermarks, to provide results on if watermarks can be reliably embedded in a NN, not just when the input data is supplied in one batch, but also when the input is fed in continuously.

2 Background

2.1 Watermarking Neural Networks

There are generally two ways of embedding watermarks in NNs, which can also be categorized based on the way the watermarks are extracted for verification.

When the watermarks are embedded in the model parameters, as formulated in the survey by Boenisch [3], this way of watermark embedding is named white-box mode. But since NNs under online training are learning and changing with the new incoming data, this mode of watermark embedding is unsuitable for this work, since the model and its parameters cannot be held constant to preserve the watermarks. In this way, it is also less realistic to easily verify the watermarks, since the claimant must be granted access to the internals of the model.

When watermarks can only be extracted using certain set of inputs and inspecting the output of the model, while someone is unable to inspect any parameters or internals of the NN, this way of watermark embedding is named black-box mode. With this way of verifying watermarks, the NN needs to be trained to be able to output a verifiable set of results when certain input is fed in. This triggering set of inputs is suitable for watermarking online learning models since it can be supplied to the model at any time, and as Boenisch [3] points out, a claimant only needs access to a query interface for verification. Several black-box approaches were closely examined and two from these, namely the *ingrainer* approach and the *exponential weighting* approach were chosen as the main focus of the thesis as the basis of work.

2.1.1 Ingrainer Approach

The *ingrainer* approach is proposed by Yang [30], where an additional ingrainer model which imbues the watermark into the neural connections of the main classifier task is created. The ingrainer has the same model structure as the classifier model, and contains watermark information through overfitting on the watermark dataset.

The classifier model is trained to acknowledge the ingrainer by incorporating the ingrain loss into its loss function, so that the classifier learns its main task on the main training data and is also able to recognize watermark data. The loss function to be optimized is calculated with Equation (2.1), the ingrain coefficient λ regulates the trade-off between main classifier accuracy and watermark ingraining [30]. The ingrain loss $\mathcal{L}(F_{w,T}(x), G_\theta(x))$ is essentially the distance between the classifier model and the ingrainer model at any specific epoch or batch. In the loss function, F_w represents the main classifier where T is the hyperparameter temperature, D stands for the dataset, x for model input, y for data label, and G_θ represents the ingrainer carrying watermark information.

The ingrainer is held fixed during the embedding of watermarks at the classifier training stage. Whereas the ingrainer is trained with only watermark data, the classifier is trained with main training data mixed in with watermark data. This extra mix of watermarks during classifier training helps the classifier gain in watermark accuracy.

$$L_D(F_w) = \frac{1}{|D|} \sum_{x \in D} \mathcal{L}(F_w(x), y) + \lambda \mathcal{L}(F_{w,T}(x), G_\theta(x)) \tag{2.1}$$

Algorithm 1 is used to train the classifier when embedding watermarks. With mini-batches of mixed main training data and watermarks, the classifier loss is calculated using Equation (2.1) and the ingrain loss is calculated using line 8 in Algorithm 1, respectively. The weighted average gradients are then subsequently used to update the parameters of the model.

Here, the watermark data is sampled from a different distribution, to which the ingrainer model is overfitted. To create these random watermarks, the authors adopted the random walk approach explained in Section 4.3.

Closely related to the ingrainer embedding approach is the capacity abuse attack introduced by Song [23]. Even though here the authors propose an attack to infer model information in a black-box setting by abusing the vast capacity of a ML algorithm, they use similar to the ingrainer approach pseudorandom images as synthetic data, only in Song [23] they are used as malicious augmented data, whereas in Yang [30] they are used as watermarks.

2.1.2 Exponential Weighting Approach

The *exponential weighting* approach from Namba [21] gives the NN parameters with large absolute values more weights during training, such that the model is more

Algorithm 1 Training Classifier F_w

Input: Training dataset $D = (x_j, y_j)_{i=1}^{n}$, watermark-carrier set $D_S = (x_{s_j}, y_{s_j})_{i=1}^{m}$, Ingrainer G_θ, number of epochs P, learning rate η, ingrain coefficient λ, ingrain temperature T.

Output: Model parameters w of classifier F_w.

$w \longleftarrow$ **initialize** (F_w)
$D_A \longleftarrow$ **shuffle** $(D \cup D_S)$
for $p = 1$ to P **do**
  for each mini-batch $b \subset D_A$ **do**
    $(x_j, y_j)_{j=1}^{a} \longleftarrow$ **getTrainData**(b)
    $(x_{s_j}, y_{s_j})_{j=1}^{b} \longleftarrow$ **getWatermarkCarrier**(b)
    $g_D \longleftarrow \bigtriangledown_w \frac{1}{a} \sum_{j=1}^{a} \mathcal{L}(F_w(x_j), y_j) + \lambda \mathcal{L}(F_{w,T}(x_j), G_\theta(x))$
    $g_{D_S} \longleftarrow \bigtriangledown_w \frac{1}{b} \sum_{j=1}^{b} \mathcal{L}(F_w(x_{s_j}), y_{s_j})$
    $g \longleftarrow (a g_D + b g_{D_S})/(a + b)$
    $w \longleftarrow$ **updateParameters**(η, w, g)
  end for
end for

resistant against model modifications such as pruning. In their work, only pruning was performed as a watermark attack on their proposed approach, and good results have been shown.

In the first stage of exponential weighting training, only main training data with correct labels are used to preliminarily train the model using conventional operations to acquire some learning on the main task. In the second stage of training, both main training data and watermark data are used, and the operations have been applied with exponential weighting on all parameters to further train the model and embed the watermarks.

The application of exponential weighting on the NN is expressed in Equation (2.2). With h^l representing input to layer l and h^{l+1} representing the output of layer l and at the same time the input to layer $l+1$, op^l denotes the operations on inputs and parameters, and a^l represents the activation function [21].

$$h^{l+1} = a^l(op^l(h^l, EW(\theta^l, T))) \tag{2.2}$$

Specifically, exponential weighting is computed using the parameters θ^l and hyperparameter T as shown in Equation (2.3). Each parameter's absolute value is multiplied with T and applied to the exponential function. This result is divided through the maximum results obtained throughout the parameters to obtain the weighting ratio.

This weighting ratio is lastly multiplied with the original parameter value to produce the weighted parameters [21]. In end result, the larger values among the parameters stay minimally affected by such weighting, and parameters with smaller absolute values get minimized to even smaller values.

$$EW(\theta^l, T) = \frac{\exp|\theta_i^l|T}{\max_i \exp|\theta_i^l|T}\theta_i^l \tag{2.3}$$

To avoid key detection by attackers and thus avoiding attacks through removals of watermark keys, the exponential weighting approach utilizes samples drawn from the same distribution as the normal main training data as keys, and set them apart from main training data by assigning them random labels.

A threshold is used in the exponential weighting approach to set the watermark accuracy criterion for watermark verification. This threshold can be calculated by Equation (2.4) where θ stands for the maximum number of misclassifications on the key set in order for the model to be verified as watermarked, and K stands for the length of the the key set.

$$threshold = 1 - \frac{\theta}{K} \tag{2.4}$$

2.2 Online Learning

For the second stage of work, online learning is used as the training paradigm. Losing [20] defines such incremental learning algorithms as one that generates on a given stream of training data $s_1, s_2, s_3, ..., s_t$ a sequence of models $h_1, h_2, h_3, ..., h_t$, whereas s_i is labelled training data and h_i is a model function that depends only on h_{i-1}. Losing [20] further qualifies online learning algorithms as incremental learning algorithms that are bounded in complexity and run-time, capable of an indefinite period of learning. The setting of online learning is similar in effect to that of fine-tuning attacks, where the capability to correctly predict watermarks to their targets are slowly diluted from the NN without watermark reinforcement.

2.2.1 Stationary Online Learning

As pointed out by Ditzler [5], when referred to, online learning implies by explicit or implicit assumption, that the training data is drawn from a fixed, albeit unknown

probability distribution. Here the name "stationary learning" refers to the stationary probability distribution, and is only used to differentiate this type of online learning from non-stationary online learning introduced in Section 2.2.2, where the underlying data distributions change over time [6]. This work also uses this flavor of stationary online learning, since all of the data comes from the same dataset, where one portion of the data are never seen by the model during the embedding phase, and is used to feed into the model piece by piece during the online learning experiments.

2.2.2 Non-stationary Online Learning with Concept Drift

Due to seasonality or periodicity effects, Ditzler [5] explains that a non-stationary environment is indeed often the more valid setting for many online learning scenarios. From detecting and protecting from adversary actions such as in the context of computer security, to monitoring and managing systems such as traffic, the survey by Žliobaitė [31] points out online learning with concept drifts has relevant applications. Formally defined by Gama [7], non-stationary online learning involves concept drift, which "refers to an online supervised learning scenario when the relation between the input data and the target variable changes over time".

To detect such probability distribution changes, Gama [8] presents a method using a set of continuous examples where the distribution is stationary, which the authors define as a *context*. Through controlling the error rate of the algorithm when online learning, the authors define a warning level for each context. When the error reaches the warning level, a new context is declared, and the model can start learning a new task starting at the example that triggered the warning level. In this way, concept drift can be detected such that the model can adapt accordingly.

To adapt online learners to learning environments with concept drift, Kolter [14] devises the *dynamic weighted majority* mechanism based on the *weighted majority* algorithm. Using an ensemble of base learners and dynamically creating and deleting experts based on their performances, the authors achieve results when learning with concept drift almost as good as base learners' results learning individual concepts. Widmer [28] proposes a window of examples to solve the concept drift problem in online learning scenarios. As the window slides through the data stream, either a new description is learned, or one of the old learned descriptions is selected as the best fitting learning concept. Another technique to deal with concept drift is to use an improved variation on the window of examples approach, called *probabilistic approximate window*, as proposed by Bifet [2]. Here the authors keep a window of examples, and store the newest examples with a higher probability, so that the model learns as quickly as possible from new examples, but also maintains some information from older instances, fulfilling the requirement of maintaining information on past concept drifts while being able to adapt quickly to new ones.

3 Related Work

3.1 Watermarking Approaches

3.1.1 Black Box Approaches

According to the survey by Boenisch [3], robustness is defined in the privacy preserving machine learning context as, "watermark should be robust against removal attacks." To counter such attacks, some have proposed approaches to increase the relatedness of the main prediction algorithm and the watermark algorithm, so that the watermarks cannot be removed without damaging the functionality of the main NN task. Specifically, the watermark techniques using a specific trigger dataset are of interest. This choice is motivated by the fact that embedding of watermark in NN properties are easily removable by attackers, and trigger sets can be designed to be either indistinguishable from normal training data, or be remembered by the algorithm itself, where this memory is practically impossible to remove. The following embedding approaches can verify the embedded watermarks in a black-box setting as explained in Section 2.1. These approaches belong to the same category as the ingrainer approach and exponential weighting approach introduced in Section 2.1.1 and Section 2.1.2.

The entanglement approach put forward by Jia [12] uses *soft nearest neighbor loss* to make the model extract common features of the data from the main learning task and the data that contains watermarks, making the two tasks one representation of the same sub-model.

The privacy resistant watermark approach using pixel patterns proposed by Li [18] uses a pixel pattern as filter pattern to "null embed" watermarks during training, where the extreme values based on the pattern are embedded in the original image. This filter pattern is based on the signature of the owner to create a binding connection between the owner and the watermarked NN.

Le Merrer [15] proposes modifying the decision boundary to correctly classify all adversarial examples and all data points that are predicted correctly originally but lie on the false side of the unmodified decision boundary, thus using the modified deci-

sion boundary as the watermark and the adversarial examples and data points close to the original boundary as trigger sets to identify the owner of the model. Using abstract color images as trigger set, Adi [1] employs the cryptographic scheme *commitment* to relate models to their owners. Xu [29] proposes embedding serial number in the model using two losses, where the two losses part is similar to the ingrainer approach. This serial number is based on the owner's signature and registered with an authority, and the serial number can be obtained by a trigger set.

Capable of being used in both black-box and white-box setting, Rouhani [22] proposes training the model such that the mean of selected network layer parameters embed a chosen watermark string. A matrix can be used to map this selection to watermarks in a white-box setting, and in a black-box setting trigger sets can be constructed from the tail region of the probability density function. The dynamic nature of this approach makes it more flexible than white-box approaches such as the one proposed by Uchida [24].

3.1.2 White Box Approaches

The following approaches embed watermarks that can be verified in a white-box setting as explained in Section 2.1.

Uchida [24] embeds a bit string by summing up the classifier model losses with an embedding regularizer, who imposes statistical biases on parameters to represent the watermark. To embed and verify the watermark, a secret embedding parameter is needed. Wang [27] proposes a generative adversarial network (GAN) composed of a generator, which is a watermarked model who tries to generate non-detectable watermarks, and a discriminator who tries to tell watermarked models and non-watermarked models apart. In end effect, the model is encouraged to embed the model with watermarks, and at the same time not be detected as so by attackers through staying similar to non-embedded models.

3.2 Watermark Attacks

An attacker can either suppress the reaction of the model on watermark trigger sets when these are embedded in black-box mode, or remove the watermarks to invalidate the watermarked NNs ownership claims [3], when these are embedded in a white-box mode. To first detect the watermarks, Wang [25] points out that watermark embedding often changes the probability distribution of the model parameters. This change exposes a surface for attackers to identify the watermarks.

To remove watermarks, there are variations on the attack such as fine-tuning, pruning, and retraining [30]. Among these, fine-tuning is similar to online learning in a way because it trains the model further with a refining set. This set though is limited in size. A retraining attack would match the settings of online learning better, where the model is continuously trained with new incoming data.

Chen [4] proposes a fine-tuning attack using limited data and doubling the learning rate every fixed number of epochs. They use *elastic weight consolidation* proposed by Kirkpatrick [13] to help the model remember learned old knowledge, resisting the effects of catastrophic forgetting, where the model unlearns task A when it transitions from learning task A to task B [13]. Liu [19] proposes a framework to remove black-box watermarks through data augmentation and distribution alignment of watermark data and main training data, achieving a watermark attack with only a small amount of data. These attacks assume the limitation of the amount of training data available, since in a classic attack scenario labelled training data is difficult to come by. Therefore these attacks are working in a different set up as this thesis, where the unlimited source of new training data is the major force diluting the watermarks.

Wang [26] proposes using a GAN to detect watermarks and to reverse them back to clean images, then fine-tuning the model using these reversed watermarks to remove the watermarks from the model. Notably, however, that this proposal will not work for embedding approaches like exponential weighting, where the watermarks are indistinguishable from main training data. Neither will it have the intended effect on the ingrainer approach, where the watermark set is unrelated to main training data, and can by definition not be reverted back to clean training data. This proposal also focuses on working with the watermark set, and involves no additional main training data as is the question of this thesis.

To suppress the algorithms' reaction on watermark trigger sets, Hitaj [11] proposes using an ensemble of stolen ML algorithms, and letting them vote for the output, where the probability of successfully verifying ownership by supplying one triggering set from one member of the ensemble is diminished. This effect on the model is different from the premise of this thesis, where the model evolves based on new training data. With this ensemble attack, one model is essentially changed by its peers - the other stolen models. To achieve this attacks then, multiple models capable of the same or similar task need to be acquired.

Summarily, no work has yet investigated the behavior of watermarks in an online learning setting exactly. Attacks presented in previous works often assume the classical model that is trained once and then released for use. The experiments of this thesis bridge precisely this gap.

4 Methods

4.1 Watermark Re-Feeding Strategies

To retain watermark accuracies under online learning, three novel watermark re-feeding strategies are proposed and described in detail here, namely filtering by prediction, filtering by confidence score, and re-feeding watermarks at a constant number of steps. Another trivial strategy of simple re-feeding is presented in the beginning to set a baseline for re-feeding strategy performance.

4.1.1 Simple Re-Feeding at Each Step

The most trivial way to re-feed watermarks is to mix in a fixed slice of watermark with a slice of main training data at every step. After the watermarks are all trained on once by the model, if there is still main training data available, the watermark set is mixed in again from the beginning.

For this procedure, the main training data is batched in mini-batches of size m. The watermark set is also batched in mini-batches of size w. The two sets are then zipped together, forming a mixed set of mini-batches of size $m + w$. For each step of online learning, one mini-batch is taken from the mixed set, shuffled, and used for training. The shuffling prevents the model from training first on all main training data and then on all watermark data, which would drive the training towards overfitting on main training data and then towards overfitting on watermarks.

4.1.2 Re-Feeding Watermarks by Filtering based on Prediction

To filter by prediction, for each step of online learning, only the wrongly predicted watermarks are re-fed into the NN, this way only the watermarks that the network has forgotten is reinforced, saving dataset space for more main training data, and saving computational power on those watermarks that the network still remembers.

Specifically, during online learning at step N, evaluation is done, and those watermarks that are predicted with a different label than their correct watermark target are filtered out. These wrongly predicted watermarks are fed back in at step N+1 together with the appropriate main training data. The mixed set is shuffled before being fed in to reduce the possibility of overfitting.

4.1.3 Strategy Filtering by Confidence Score

To filter by confidence score, the criterion of filtering from the previous strategy can be tightened, and instead of filtering already wrongly predicted watermarks, watermarks that got predicted with a lower confidence score than a predefined threshold, albeit still possibly correctly, can be preemptively filtered out and re-fed at the following step.

Similar to filtering by prediction, this strategy filters watermarks based on the confidence score the model produces on the watermark sample. A threshold T is predefined for this strategy. The NN produces a confidence score between zero and one in every class of targets. For MNIST data (formally introduced in Section 4.2), there are ten confidence scores that the NN produces at every step. During online learning at step N, evaluation is done, and the confidence scores at the correct watermarks' target positions are compared to the threshold T. For every confidence that is less than T, this watermark is collected. These watermarks predicted with weak confidence scores are fed back in at step N+1 together with the appropriate main training data. The mixed set is shuffled before being fed in to prevent overfitting. Through this strategy, the model is expected to be reinforced in its watermark memory when it starts slowly losing its memory on the watermarks, eventually even before the model has predicted the watermark to a wrong class.

4.1.4 Strategy Partial Mixing at Every N-th Step

Whereas a simple combination of main training data and a portion of watermarks at each step for training can improve watermark accuracies, it is not always necessary for the NN to see watermarks at every step. Seeing watermarks overly frequently also makes the model overfitted on these slices of watermarks. Instead, at every fixed number of steps, a portion of the watermark set can be mixed in with some main training data to train the network. In all other steps, the model can train regularly with purely main training data without watermarks. Furthermore, this selection of watermarks from the whole watermark set is best performed randomly, so that overfitting of the NN on a specific fixed slice of watermark can be prevented. Otherwise, fixed slices of watermarks in a sequence would make the model become

overfitted on these slices and lose its ability to generalize on the whole watermark set, or even damage its main classifying accuracy.

Technically, during online learning and at a predefined interval number of steps named N (for example 5, 10, etc.), a random selection of watermarks are mixed in with main training data for training. This mixture of watermarks and main training data is shuffled before being trained on by the model. For all other steps whose step numbers are not multiples of N, only normal main training data are fed in for training, such that the model still retains its capability to perform its main classifying task.

4.2 Libraries, Dataset and Experimental Settings

This work only employs stochastic gradient descent and its variants as optimizers, meanwhile feeding entirely new data from the same distribution that the model has never seen before into the NN in the case of exponential weighting experiments, or re-feeding epochs of this kind of new data in case where the algorithm is extremely good at retaining watermarks, in the case of ingrainer experiments, to simulate the environment of online learning. After each step, the model does not see the old training data anymore; it sees in the next step only the new incoming data and watermarks, if given. The online learning data stream is generated by batching the dataset into mini-batches. Then the data stream is fed into the model by taking from these mini-batches sequentially.

Losing [20] identified datasets that are particularly suitable for incremental learning algorithms, which include MNIST [16], a dataset that has been used in virtually all previous works related to NN watermark embedding approaches, and is also used for the experiments in this work. MNIST as the main training data for this work is made up of 70,000 images of handwritten numbers of the 10 digits, each image of size 28×28.

4.2.1 PyTorch for Ingrainer

The ingrainer experiments were written in PyTorch[1]. The ingrainer watermark embedding approach [30] has been implemented by Github user `sjtukk`[2], the open source repository[3] of which has since been taken down at the time of writing. The fil-

[1] `https://pytorch.org/`
[2] `https://github.com/sjtukk/`
[3] `https://github.com/sjtukk/ingrainer`

tering mechanism within the re-feeding strategies has been realized using the *Subset* function from the `torch.utils.data`[4] library.

4.2.2 Tensorflow for Exponential Weighting

The exponential weighting experiments were written in TensorFlow[5] with Keras[6]. The filtering mechanism within the first two watermark re-feeding strategies is done with the in-library function *filter* [7]. Due to slow training on a normal CPU, these experiments were carried out on Google Colaboratory [8] with GPU run-times.

4.2.3 Ingrainer Architecture

The architecture of the ingrainer model is two layers of fully connected linear neurons, where each hidden layer contains 1200 neurons. This architecture based on the work of Hinton [10] is shared between the ingrainer model and the classifier model, such that a better transition of watermark knowledge from the ingrainer into the classifier can be guaranteed. The optimizer used is the Adadelta algorithm, which is derived from SGD.

The ingrainer model is trained for 500 epochs with batch size of 128, learning rate of 0.1, decay rate of 0.9, and numerical stability coefficient of 1e-8. The classifier on the other hand is trained for 100 epochs with batch size of 128, learning rate of 0.1, decay rate of 0.9, numerical stability coefficient of 1e-8, ingrain coefficient lambda of 0.5, and a temperature of 15.

Of the 70,000 data points in the MNIST dataset [16], 10,000 data points are reserved for evaluation, 12,000 of these MNIST images along with 1000 self-produced watermarks are used to train the ingrainer, 24,000 more MNIST images are used to train the classifier and embed the watermarks. The rest 24,000 are used for online learning experiments. Due to the robustness of the approach in retaining watermarks under online learning, these last 24,000 training data are re-fed into the classifier for 15 epochs to expose the effect of the loss of watermark memory under online learning, such that further experiments can be conducted on the basis of some accuracy loss. Otherwise it would have been ideal to use these data only for one epoch, upholding the limitation that all online learning data should be data that the model has never seen before.

[4] `https://pytorch.org/docs/stable/data.html`
[5] `https://www.tensorflow.org`
[6] `https://www.tensorflow.org/api_docs/python/tf/keras`
[7] `https://www.tensorflow.org/api_docs/python/tf/data/Dataset#filter`
[8] `https://research.google.com/colaboratory/faq.html`

Custom dataset classes are created to load the main training data and to give them a tag of "true" to mark that they are the main training data. Correspondingly, watermark data are loaded with another custom class and tagged with "false" to mark their nature of watermarks. Lastly, a custom mixing class is created to mix main training data with watermark data together, marking each as "true" and "false", respectively. This boolean value is used to distinguish if a data point is originally main training data or not, such that main classifier accuracy and watermark accuracy can be evaluated accordingly.

4.2.4 Exponential Weighting Architecture

The 18-layer residual nets from the work of He [9] is used as the classification model. Exponential weighting is implemented as described in Namba [21], and can be turned on or off, depending on if there is any watermark present in the training set.

The watermark embedding phase is conducted for 10 epochs, whereas any subsequent kind of online learning are done in 314 steps as one single epoch. The model sees the main training data during online learning for the first time and for once only. Temperature as defined in [21] is set at 2, and 30 watermarks are created and used. The learning rate is set at 0.01, and SGD is used as optimizer with a momentum of 0.9.

The source codes for the NNs, watermark embedding, online learning, and watermark re-feeding experiments are accessible at GitLab [9]. Access rights may be obtained from the thesis supervisor.

4.3 Independently Produced Watermark Dataset

Concerning the *ingrainer* approach, the watermark set is a set of custom made random images drawn with lines that are not hand-written digits as in the MNIST dataset. According to Yang [30], the watermark-carrier data distribution should be different from the training data distribution. Thus a sampling from the main training data as the watermark set is not a possibility. To conduct this experiment, a self-produced set of 1000 watermarks is manufactured according to the instructions provided in Yang [30].

Using random walk and starting from the center of a blank image, a line is drawn in a random direction for a random number of pixels. Then another line is drawn

[9] `https://git.imp.fu-berlin.de/private_secure_ml/robust-nn-in-online-learning`

again at a new random direction for a new random number of pixels. The number of steps of drawing is set to the size of the main training data, in case of MNIST, 28. According to Yang [30], this way of generating random images bring about the least amount of damage to the main task accuracy. For the label of these watermarks, random numbers are chosen from the targets of the main training data, in this case the 10 digits of the MNIST targets. An example of one such watermarks is shown in Figure 4.1.

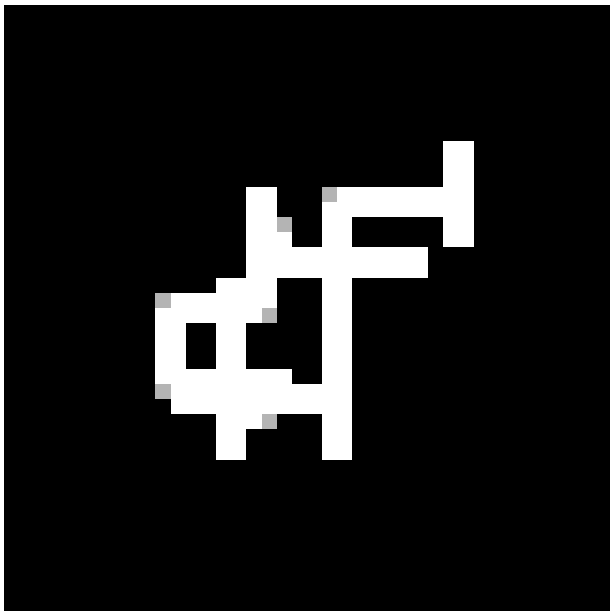

Figure 4.1: An example of the 1000 self-produced watermarks for the ingrainer approach using random walk and having the same size as a data point in the MNIST dataset.

5 Results

5.1 Results for Ingrainer Approach and Online Learning

5.1.1 Watermark Embedding and Online Learning

The set of 1000 independently produced watermarks consisting of random walk images of the same dimensions as the images in the MNIST dataset are embedded in the classifier model with the help of the trained and fixed *ingrainer*. The progression of watermark accuracy during the course of watermark embedding is shown in Figure 5.1. At the end of the watermark embedding, the watermark accuracy reaches 99%.

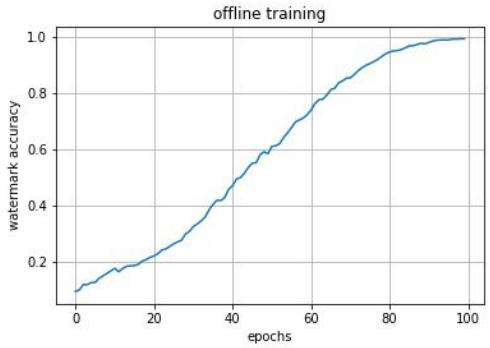

Figure 5.1: Watermark embedding using the ingrainer approach.

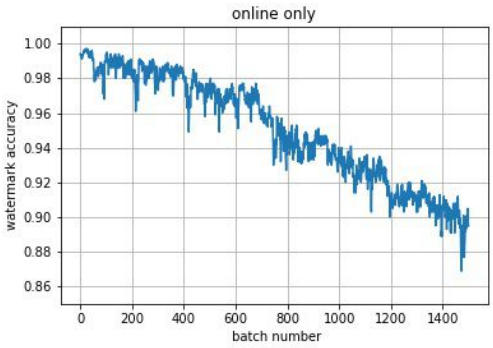

Figure 5.2: Online training on watermark embedded model without watermarks.

With data separated beforehand from the MNIST dataset that the model has not yet seen before, the model is trained under online learning conditions, with no watermarks mixed in. At this stage, the classifier only has access to the ingrainer as the source of knowledge of the watermarks. The decay of watermark accuracy during this phase of pure online learning is shown in Figure 5.2. The classifier shows a steady decrease in its ability to recognize embedded watermarks, however this decline is slow. Hyperparameters have also been adjusted for the model to exhibit

a significant amount of watermark accuracy decay, providing vacant accuracy space such that subsequent experiments can be carried out to study the model's behaviors under different strategies of watermark re-feeding. At the end of online learning experiment, the watermark accuracy of the classifier has decreased to 90%.

5.1.2 Simple Re-feeding at Each Step

The preliminary re-feeding strategy described in Section 4.1.1 is applied to the ingrainer approach when online learning. Result presented in Figure 5.3 shows this strategy is an effective way to keep watermark accuracy high during online learning. The main classfication accuracy also stays consistently high, above 95% at all times.

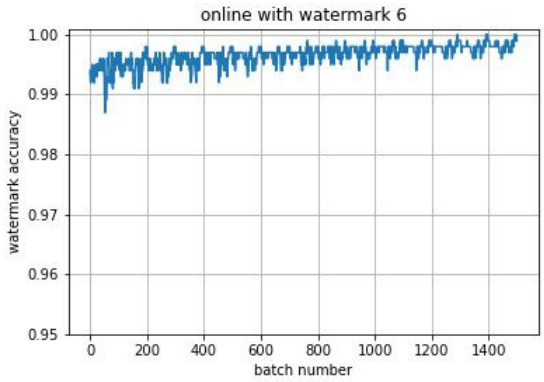

Figure 5.3: Watermark accuracy from simple watermark re-feeding strategy on ingrainer approach.

5.1.3 Filter Strategy by Prediction and by Confidence Score

Following the watermark embedding phase shown in Figure 5.1, the watermark embedded model is trained with main training data that it has never seen before, and re-fed with the watermark set using the watermark re-feeding strategies *filter by prediction* and *filter by confidence score* described in Section 4.1.2 and Section 4.1.3.

Both watermark re-feeding strategies have given high and close to 100% watermark accuracies consistently. The improvements of watermark accuracy using these two strategies are also almost instantaneous. In comparison, the filter by confidence score strategy provides a more consistently high watermark accuracy than the filter by prediction strategy. This agrees with the intuition that filtering by confidence score is a preemptive strategy that finds weak watermarks sooner.

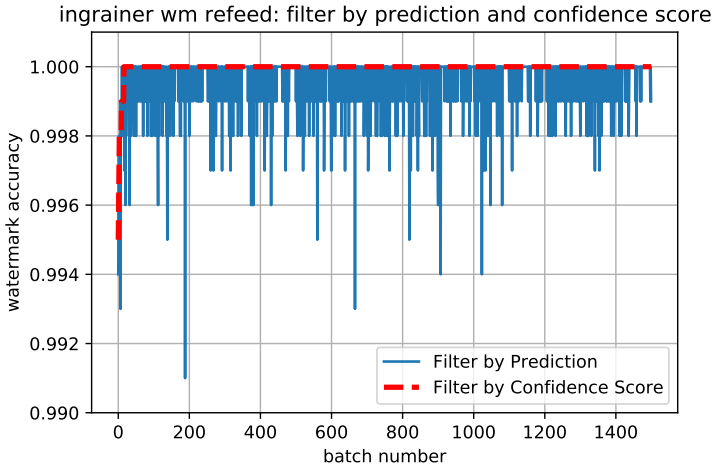

Figure 5.4: Re-feeding watermark on ingrainer model using filter by prediction strategy and filter by confidence score strategy.

With a threshold of 80% confidence for filtering watermarks the model is forgetting, the filter by confidence score strategy has produced the best watermark re-feeding results in terms of watermark accuracy for the ingrainer approach.

5.1.4 Partial Mixing at N-th Step Strategy

Following the watermark embedding phase shown in Figure 5.1, the watermark embedded model is trained with main training data that it has never seen before, and re-fed with watermarks using the watermark re-feeding strategy *mix-n* described in Section 4.1.4.

With 50% of watermark and 50% of main training data at N^{th} step of online training, experiments with N = 25, 50, 100 and 200 are carried out.

The results summarized in Figure 5.5 show that, as the interval of steps of re-feeding gets smaller, the watermark accuracy decays slower. However, compared to filtering by prediction or filtering by confidence score, this series of experiments show that using strategy *mix-n*, the watermark accuracy decays steadily; Whereas with the filtering strategies previously, the watermark accuracies showed no sign of decrease.

To further investigate the effect of changing watermark percentages mixed in at N^{th} step on watermark accuracy, the value of N is held fixed. To reserve space to potential accuracy increases or decreases, this fixed value of N is picked aside from

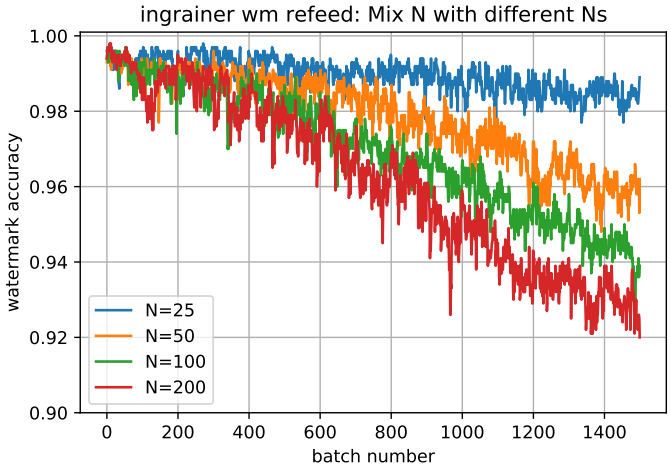

Figure 5.5: Re-feeding watermark on ingrainer model using *mix-n* strategy with different Ns. A larger N value indicates watermarks are fed in *less* frequently.

the best performing N value and the worst performing N value. With re-feed interval N selected at 50, more experiments are carried out with 20%, 40%, 50%, 60% and 80% of watermarks at every N^{th} step. The rest of the training batch is filled with main training data to keep the number of training samples consistent across training steps, whether there is a watermark re-feed happening or not.

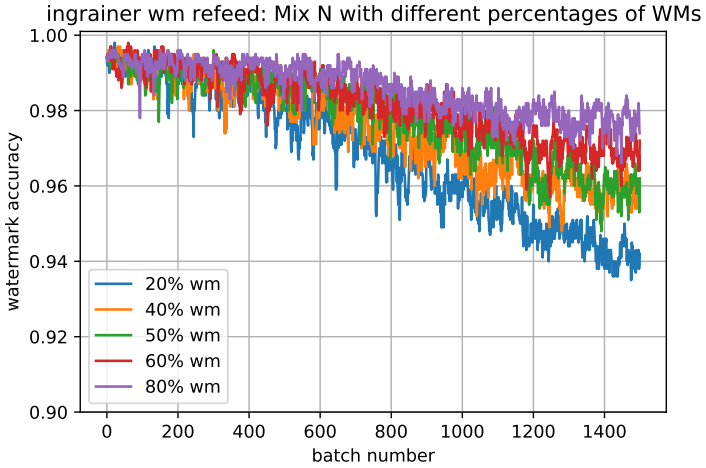

Figure 5.6: Re-feeding watermark on ingrainer model using *mix-n* strategy with different watermark percentages.

The results summarized in Figure 5.6 show that as the percentage of watermarks

mixed in at the N^{th} step gets higher, the slower the watermark accuracy will decrease. However, just like modifying the interval of steps when re-feeding take place, modifying the percentage of watermarks mixed in at the N^{th} step also does not help the model retain watermarks at a consistently high accuracy as the filtering strategies can.

The main classifier accuracies from these experiments on ingrainer approach have been steady and without significant variations throughout watermark embedding and online learning phases, as jointly depicted in Figure 5.7.

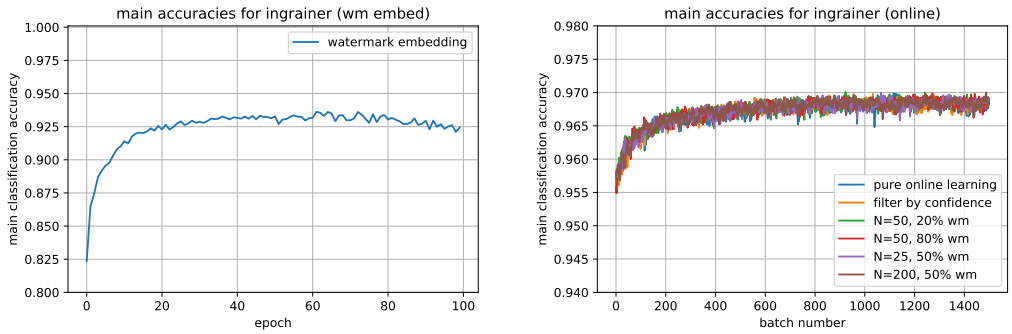

Figure 5.7: Main classification accuracy of the ingrainer approach during watermark embedding *(left)*, and various online learning experiments *(right)*.

5.2 Results for Exponential Weighting Approach and Online Learning

5.2.1 Watermark Embedding and Online Learning

The set of 30 watermarks drawn from the MNIST dataset with their labels changed to randomly generated digits 0 to 9 are embedded in the classifier model with *exponential weighting* turned on. The progression of watermark accuracy during the course of watermark embedding is shown in Figure 5.8. At the end of the watermark embedding phase, the watermark accuracy reaches 100%.

With data separated beforehand from the MNIST dataset that the model has not yet seen before, the model is trained under online learning conditions, with no watermarks mixed in. At this stage, exponential weighting is turned off since there is no watermarks in the training set. The decay of watermark accuracy during this phase of pure online learning is shown in Figure 5.9. The model shows an abrupt

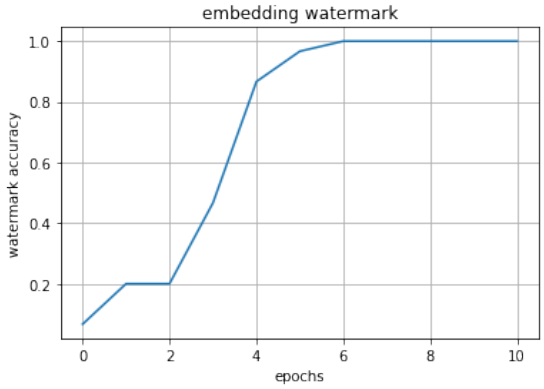

Figure 5.8: Watermark embedding using exponential weighting.

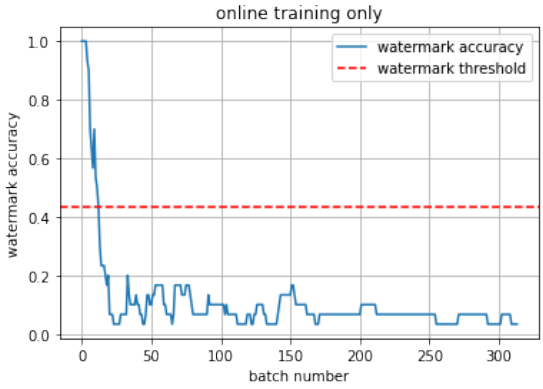

Figure 5.9: Online training on watermark embedded model without watermarks.

collapse in its ability to recognize embedded watermarks, reaching quickly to an accuracy level close to that of random guessing (10%). At the end of the online learning experiment, the watermark accuracy of the classifier has decreased to 3.33%.

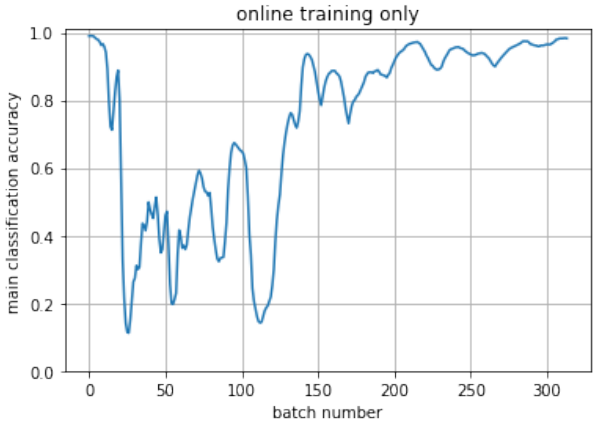

Figure 5.10: Main classifier accuracy for exponential weighting approach when purely online learning.

While the main classifier accuracy also dropped at the beginning of pure online learning, this has recovered back to 98.27% at the end of the online learning experiment, as shown in Figure 5.10.

5.2.2 Simple Re-feeding at Each Step

The preliminary re-feeding strategy described in Section 4.1.1 is applied to *exponential weighting* approach when online learning. Results presented in Figure 5.11 show this strategy is only an effective way to keep watermark accuracy high when online learning during latter stages of online learning, since the initial drop in main classifier accuracy as well as the drop in watermark accuracy both need some time to recover.

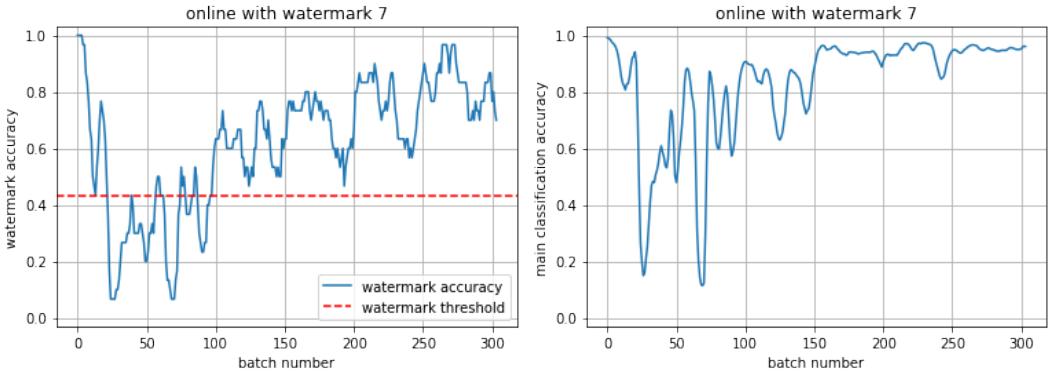

Figure 5.11: Watermark accuracy and main classifier accuracy from simple watermark re-feeding strategy on exponential weighting approach.

5.2.3 Filter Strategy by Prediction and by Confidence Score

Following the watermark embedding phase shown in Figure 5.8, the watermark embedded model is trained with main training data that it has never seen before, and re-fed with the watermark set using the watermark re-feeding strategy *filter by prediction* and *filter by confidence score* described in Section 4.1.2 and Section 4.1.3.

Filter by prediction depicted in Figure 5.12 and filter by confidence score shown in Figure 5.13 both delivered insufficient results in terms of keeping the watermark accuracy above the verification threshold. During online training with watermark re-feeding, almost all of the whole complete watermark set is misclassified at each step, and therefore almost all of them are re-fed into the model at every step. This much of watermark reinforcement, however, is still not enough to lift the accuracy above the threshold.

In an attempt to improve on the performance of filter by prediction and filter by confidence score strategies with the exponential weighting approach, the filtered

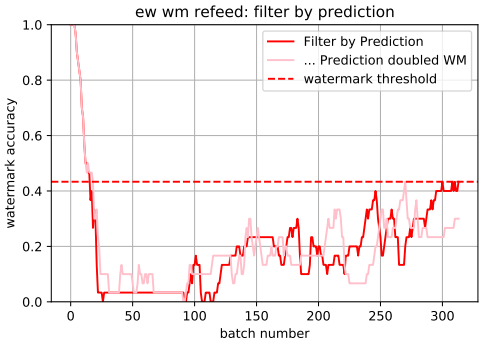

Figure 5.12: Re-feeding watermark on exponential weighting model using filter by prediction strategy.

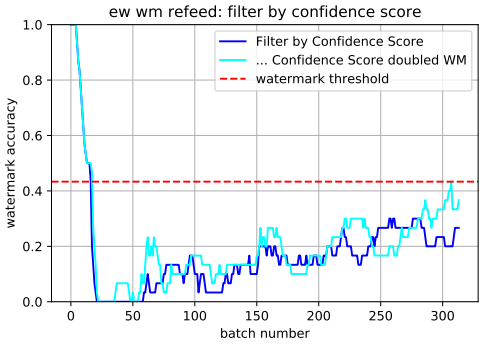

Figure 5.13: Re-feeding watermark on exponential weighting model using filter by confidence score strategy.

out watermark subsets are respectively doubled by repetition, and re-fed into the model. The results are plotted in the same graphs as in Figure 5.12 and Figure 5.13 with lighter shades of colors. This attempt still did not significantly improve the watermark accuracies of the model.

5.2.4 Partial Mixing at N-th Step Strategy

Following the watermark embedding phase shown in Figure 5.8, the watermark embedded model is trained with main training data that it has never seen before, and re-fed with watermarks using the watermark re-feeding strategy *mix-n* described in Section 4.1.4.

With 60% of watermark and 40% of main training data as input for the model at N^{th} step of online training, and using only main training data at other steps, experiments with N = 5, 10 and 20 are carried out.

The results summarized in Figure 5.14 show that the mix-n re-feeding strategy can keep the watermark accuracy high, despite an initial steep dip in watermark accuracy around step 50. Compared to N of 20, N of 5 and 10 preformed better. This work has chosen N of 10 as the optimal value due to its superior performance around step 100 for subsequent experiments on variations of watermark percentages.

Compared to filtering by prediction or filtering by confidence score, this series of experiments show that strategy mix-n is the watermark re-feeding strategy of choice

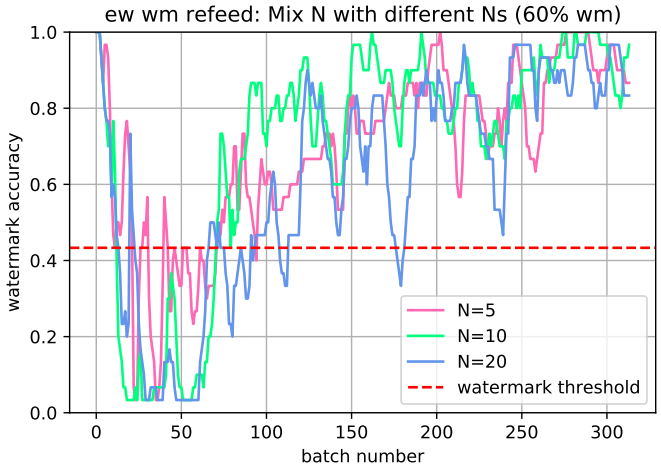

Figure 5.14: Re-feeding watermark on exponential weighting model using *mix-n* strategy with different Ns. A larger N value indicates watermarks are fed in less frequently.

for the exponential weighting approach.

With re-feeding interval N fixed at 10, more experiments are carried out with 20%, 40%, 60% and 80% of watermarks at N^{th} step. The rest of the training batch is filled with main training data to hold the number of training samples consistent across training steps, whether there is a watermark re-feed happening or not.

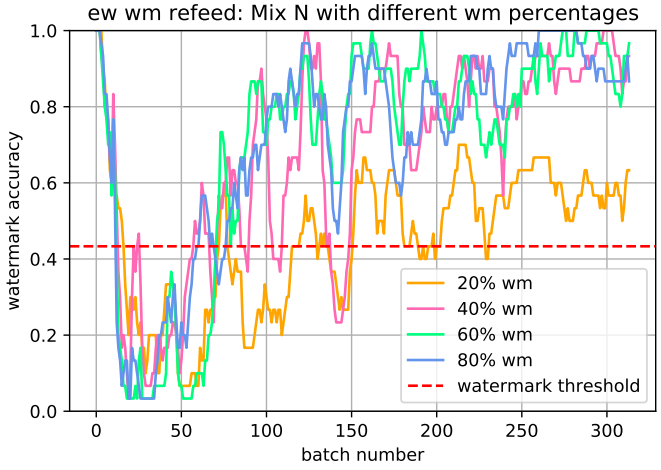

Figure 5.15: Re-feeding watermark on exponential weighting model using *mix-n* strategy with different watermark percentages.

The results summarized in Figure 5.15 show that as the percentage of watermarks mixed in at the N^{th} step gets higher, there is a general tendency of watermark accuracy increasing. These results also show that the percentage of watermarks mixed in at N^{th} step should be at least 60%, since both 20% and 40% of watermark set at N^{th} step still dip once again below the threshold after their recoveries after the first steep dip in watermark accuracies.

The main classifier accuracies from these experiments on exponential weighting approach have behaved similarly to the better performing watermark accuracies' progressions during watermark embedding and online learning phases, such as in Figure 5.10. Namely, they all generally suffer from an initial drop in accuracy at the beginning of experiments, and slowly recover as the experiments carry on.

6 Discussion and Future Work

6.1 Discussion of the Ingrainer Approach

According to [30], with ingrain coefficient λ of 0.5, the watermark accuracy using the ingrainer approach should reach 100%. Since the result of 99% shown in Figure 5.1 is achieved using about half of the main training data, the statement made in [30] about watermark accuracy after watermark embedding should be confirmed. According to the author, λ of 0.5 should also produce the highest main classfication accuracy among the options tested.

According to [30], with 20% of the main training data set aside as the refining set, and carrying out fine-tuning attack with this refining set for 25 epochs, the watermark accuracy still stays at 100% at the end of attack. Even though the result shown in Figure 5.2 is obtained with a larger portion of refining data, different hyperparameters, and only one epoch of online learning, it has a similar effect on the model as the fine-tuning attack carried out by the authors of [30]. However, the end accuracy of 90% shown in Figure 5.2 shows that the watermark accuracy does not actually always stay at 100% when trained with more data. This displays a discrepancy to the claims made by the authors of [30].

Filtering re-feeding strategies for the ingrainer approach has performed better than the mix-n strategy, which shows that the model embedded with the ingrainer approach only needs specific reminders from the watermark set to keep the watermark accuracy high. Whereas a random subset of the watermark set at a constant interval does not help the model recover its watermark accuracy as efficiently. The cause of this innate ability to retain watermark accuracy can also be attributed to the fact that the classifier is constantly trained with ingrain loss, where the knowledge of the watermark set is continuously incorporated in the model.

6.2 Overcoming Exponential Weighting's Unsuitability for Online Learning

The authors of Namba [21] have chosen pruning to demonstrate a watermark attack on their proposed approach, to show that their approach of watermark embedding is resistant against model modification. However, the dismissal of the re-training type of model modification was a negligent one. Since their approach increases the influence of parameters that already have large absolute values, and diminishes effects of parameters of smaller absolute values, their watermark embedded model would naturally be resistant to pruning attacks, where neurons with small absolute values are eliminated by a certain percentage [21]. In retrospect, the exponential weighting approach seems tailored for this one type of model modification attack only.

Due to the mechanism of label change for key samples to produce watermarks, the exponential weighting approach remembers the watermarks by overfitting on the key sample, who are drawn from the same distribution as main training data, and have labels that also fall into the same classes of targets as the normal training data. As the authors of Namba [21] themselves point out, this approach loses its watermark accuracies "instantly" when new main training data is fed in, since new training data diminishes the effect of overfitting of an NN.

Mix-n strategy has turned out to be the best solution when re-feeding watermarks in exponential weighting approach, since both kinds of filtering strategies simply do not have the capacity needed to re-feed enough watermarks into the model to keep the watermark accuracy high. Mix-n re-feeding also has to happen fairly often (every 10 steps) and fairly intensively (at least 60% of watermark) to keep the watermark accuracy above the threshold.

6.3 Comparison of the Two Examined Embedding Approaches

Compared to at least 1000 watermarks used by the ingrainer approach, the exponential weighting approach only utilizes 30 watermark keys. Indeed, the exponential weighting approach cannot afford to utilize more watermark keys in order to improve its watermark accuracies or to keep it high. Since random key label change is used as the method to mark watermark samples from normal training samples, more watermark keys in the exponential weighting approach would only confuse the classifier model with respect to its capabilities to classify normal data intended for the main task. This shows that generally when embedding NNs with watermarks, choosing watermarks from the same distribution as the main training data has the

advantage of making the watermarks undetectable by attackers, but it also has the disadvantage of having a limited number of watermarks and damaging the main classifier's capability. The comparison also shows the flaw in designing watermark embedding approaches such as exponential weighting when only one type of watermark attack is taken into consideration when the approach is designed. As a side note, the exponential calculations are also computationally heavy, making the exponential weighting approach unrealistic to be experimented upon using a normal CPU.

6.4 Subsequent Research Directions

Other approaches introduced in Section 3.1.1 that use a trigger set as watermarks, such as the *entangled watermarks* approach from Jia [12] and the *piracy resistant watermarks* approach from Li [18], can be tested in the same way for performance under online learning conditions, and investigated for the optimal strategy of watermark re-feeding when online learning.

Further challenge lies in watermarking NNs in online learning scenarios with concept drift. As the whole NN changes its learning task, it is difficult to keep the same watermarks verifiable. When even the original main training data needs not be classified correctly, the model might need even stronger watermark re-feeding as reinforcement to keep the watermark accuracy high. As the watermark knowledge representation within the NN becomes eventually more localized, it becomes easier for attackers to remove the watermarks from the NN.

The three watermark re-feeding strategies introduced here can naturally also be expanded upon. Here as an insight from the experiments, it might even be more effective to cater the re-feeding strategy individually for a particular watermark embedding approach. The two investigated approaches show that when the embedding approach employed retains watermarks badly, periodic reminder from the whole watermark set is needed; when the embedding approach utilized remembers the watermarks well, only occasional reminder from few particular watermark instances is needed.

7 Conclusion

In summary, it is possible to keep watermark accuracies of NNs high in online learning scenarios through novel watermark re-feeding strategies. Filtering by confidence score worked best for the watermark embedding approach *ingrainer*, and mix-n strategy with 60% watermark at every 10^{th} step worked best when watermarks are embedded using the *exponential weighting* approach. Specifically, the claim from the authors of the ingrainer approach about watermark embedding efficacy was confirmed, yet their claim about the robustness of the embedded watermarks when the model is trained with new data did not exactly match the experimental finding. Their proposed approach is nonetheless a very robust one in terms of watermark retention even when the model is retraining. On the other hand, the authors who proposed the exponential weighting approach were also confirmed in their claim about the efficacy of watermark embedding, but as they did not publish their work with experiments involving training with new data as an attack, it was discovered that their approach was not by itself suitable for online learning. The bottom line being, it was still possible to find a best solution for watermark re-feeding strategy during online learning for both embedding approaches examined.

Bibliography

[1] Adi. "Turning Your Weakness Into a Strength: Watermarking Deep Neural Networks by Backdooring". In: *arXiv.org* (2018).

[2] Bifet. "Efficient data stream classification via probabilistic adaptive windows". In: *Proceedings of the 28th Annual ACM Symposium on Applied Computing* (2013).

[3] Boenisch. "A Survey on Model Watermarking Neural Networks". In: *arXiv.org* (2020).

[4] Chen. "REFIT: A Unified Watermark Removal Framework For Deep Learning Systems With Limited Data". In: *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security* (2021).

[5] Ditzler. "Learning in Nonstationary Environments: A Survey". In: *IEEE Computational intelligence magazine* (2015).

[6] Elwell. "Incremental Learning of Concept Drift in Nonstationary Environments". In: *IEEE Transactions on Neural Networks* (2011).

[7] Gama. "A Survey on Concept Drift Adaptation". In: *ACM Computing Surveys, Vol. 1, No. 1, Article 1* (2013).

[8] Gama. "Learning with Drift Detection". In: *Intelligent Data Analysis - September 2004* (2004).

[9] He. "Deep Residual Learning for Image Recognition". In: *arXiv.org* (2015).

[10] Hinton. "Distilling the Knowledge in a Neural Network". In: *NIPS Deep Learning and Representation Learning Workshop* (2015).

[11] Hitaj. "Have You Stolen My Model? Evasion Attacks Against Deep Neural Network Watermarking Techniques". In: *arXiv.org* (2018).

[12] Jia. "Entangled Watermarks as a Defense against Model Extraction". In: *arXiv.org* (2020).

[13] Kirkpatrick. "Overcoming catastrophic forgetting in neural networks". In: *arXiv.org* (2017).

[14] Kolter. "Dynamic Weighted Majority: A New Ensemble Method for Tracking Concept Drift". In: *Proceedings of the Third International IEEE Conference on Data Mining, 123-130* (2003).

[15] Le Merrer. "Adversarial frontier stitching for remote neural network watermarking". In: *Neural Computing and Applications (2020) 32:9233-9244* (2020).

[16] LeCun. *The MNIST Database of handwritten digits*. 2010. URL: `http://yann.lecun.com/exdb/mnist/` (visited on 07/15/2021).

[17] Li. "A survey of deep neural network watermarking techniques". In: *arXiv.org* (2021).

[18] Li. "Piracy Resistant Watermarks for Deep Neural Networks". In: *arXiv.org* (2020).

[19] Liu. "Removing Backdoor-Based Watermarks in Neural Networks with Limited Data". In: *arXiv.org* (2020).

[20] Losing. "Incremental On-line Learning: A Review and Comparison of State of the Art Algorithms". In: *Elsevier* (2017).

[21] Namba. "Robust Watermarking of Neural Network with Exponential Weighting". In: *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security* (2019).

[22] Rouhani. "DeepSigns: A Generic Watermarking Framework for Protecting the Ownership of Deep Learning Models". In: *arXiv.org* (2018).

[23] Song. "Machine Learning Models that Remember Too Much". In: *arXiv.org* (2017).

[24] Uchida. "Embedding Watermarks into Deep Neural Networks". In: *arXiv.org* (2017).

[25] Wang. "Attacks on Digital Watermarks for Deep Neural Networks". In: *2019 IEEE International Conference on Acoustics, Speech and Signal Processing* (2019).

[26] Wang. "Detect and remove watermark in deep neural networks via generative adversarial networks". In: *arXiv.org* (2021).

[27] Wang. "Undetectable and Robust White-Box Watermarking of Deep Neural Networks". In: *arXiv.org* (2020).

[28] Widmer. "Learning in the Presence of Concept Drift and Hidden Contexts". In: *Kluwer Academic Publishers, Machine Learning, 23, 69-101* (1996).

[29] Xu. "A novel method for identifying the deep neural network model with the Serial Number". In: *IEEE Access* (2019).

[30] Yang. "Effectiveness of Distillation Attack and Countermeasure on Neural Network Watermarking". In: *arXiv.org* (2019).

[31] Žliobaitė. "Learning under Concept Drift: an Overview". In: *arXiv.org* (2010).