

# Entangled Watermarks as a Defense against Model Extraction

Hengrui Jia, Christopher A. Choquette-Choo, Nicolas Papernot  
*University of Toronto and Vector Institute*

## Abstract

Machine learning involves expensive data collection and training procedures. Model owners may be concerned that valuable intellectual property can be leaked if adversaries mount model extraction attacks. Because it is difficult to defend against model extraction without sacrificing significant prediction accuracy, watermarking leverages unused model capacity to have the model overfit to outlier input-output pairs, which are not sampled from the task distribution and are only known to the defender. The defender then demonstrates knowledge of the input-output pairs to claim ownership of the model at inference. The effectiveness of watermarks remains limited because they are distinct from the task distribution and can thus be easily removed through compression or other forms of knowledge transfer.

We introduce Entangled Watermarking Embeddings (EWE). Our approach encourages the model to learn common features for classifying data that is sampled from the task distribution, but also data that encodes watermarks. An adversary attempting to remove watermarks that are entangled with legitimate data is also forced to sacrifice performance on legitimate data. Experiments on MNIST, Fashion-MNIST, and Google Speech Commands validate that the defender can claim model ownership with 95% confidence after less than 10 queries to the stolen copy, at a modest cost of 1% accuracy in the defended model’s performance.

## 1 Introduction

Costs associated with machine learning (ML) are high. This is true in particular when large training sets need to be collected [13] or the parameters of complex models tuned [36]. Therefore, models being deployed for inference constitute valuable intellectual property that needs to be protected. A good example of a pervasive deployment of ML is automatic speech recognition [14], which forms the basis for personal assistants in ecosystems created by Amazon, Apple, Google, and Microsoft. However, deploying models to make predic-

tions creates an attack vector which adversaries can exploit to mount *model extraction* attacks [3, 7, 26, 30–32, 38].

Techniques for model extraction typically require that the adversary query a *victim model* with inputs of their choice—analogue to chosen-plaintext attacks in cryptography. The adversary uses the victim model to label a *substitute dataset*. One form of extraction involves using the substitute dataset to train a substitute model, which is a *stolen* copy of the victim model [31, 32]. Preventing model extraction is difficult without sacrificing performance for legitimate users [2, 4, 22, 38]: indeed, queries made by attackers and benign users *may* be sampled from the same *task distribution*.

One emerging defense proposal is to extend the concept of watermarking [17] to machine learning [5]. The defender purposely introduces outlier input-output pairs  $(x, y)$  only known to them in the model’s training set—analogue to a poisoning or backdoor attacks [1]. To claim ownership of the model  $f$ , the defender demonstrates that they can query the model on these specific inputs  $x$  and have knowledge of the (potentially) surprising prediction  $f(x) = y$  returned by the model. Watermarking techniques exploit the unnecessarily large capacity of architectures (watermarking is often evaluated on deep neural networks [1]) to learn watermarks without sacrificing performance when classifying data from the task distribution.

Naive watermarking can be defeated by an adaptive attacker because the watermarks are outliers to the task distribution. As long as the adversary queries the watermarked model *only* on inputs that are sampled from the task distribution, the stolen model will only retain the victim model’s decision surface relevant to the task distribution, and therefore ignore the decision surface learned relevant to watermarking. *In other words, the reason why watermarking can be performed with limited impact on the model’s accuracy is the reason why watermarks can easily be removed by an adversary.* Put another way, watermarked models roughly split their parameter set into two subsets, the first encodes the task distribution while the second overfits to the outliers (watermarks).

In this paper, we propose a technique that addresses this fundamental limitation of watermarking. *Entangled Water-*

*mark Embedding* (EWE) encourages a model to extract features that are jointly useful to (a) learn how to classify data from the task distribution, and (b) predict the defender’s expected output on watermarks. Our key insight is to leverage the *soft nearest neighbor loss* [10] to entangle representations extracted from training data and watermarks. By entanglement, we mean that the model represents both types of data similarly. Entangling produces models that use the same subset of parameters to recognize training data and watermarks. Hence, it is difficult for an adversary to extract the model without its watermarks; even if the adversary queries models with samples only from the task distribution to avoid triggering watermarks (e.g., the adversary avoids out-of-distribution inputs like random queries). The adversary is forced to learn how to reproduce the defender’s chosen output on watermarks. An attempt to remove watermarks would also have to harm the stolen substitute classifier’s generalization performance on the task distribution, which would defeat the purpose of model extraction (i.e., steal a well-performing model).

We evaluate the approach on two vision datasets, MNIST [21] and Fashion MNIST [42], as well as an audio dataset, Google Speech Command [41]. We demonstrate that our approach is able to watermark models at moderate costs in terms of the model’s utility—below 1%. Unlike prior approaches we compare against, *our watermarked classifiers are robust to model extraction attacks*. Stolen copies retain the defender’s expected output on 50% of entangled watermarks, which enables a classifier to claim ownership of the model with 95% confidence in less than 10 queries to the stolen copy. We also show that defenses against backdoors are ineffective against our entangled watermarks.

The contributions of our paper are:

- We identify a fundamental limitation of existing watermarking strategies: they overfit to watermarks separately from learning the task distribution.
- We introduce Entangled Watermark Embedding (EWE) to have models jointly learn how to classify samples from the task distribution and watermarks.
- We systematically calibrate EWE on vision and audio datasets. We show that when points being watermarked are carefully chosen, EWE offers advantageous trade offs between model utility and robustness of watermarks to model extraction.

## 2 Background

### 2.1 Learning with Neural Networks

We focus on classification within the supervised learning setting [27], where the goal is to learn a decision function that maps the input  $x$  to a discrete output  $y$ . The set of possible outputs are called classes. The decision function is typically

parameterized and represents a mapping function from a restricted hypothesis class. A dataset containing a number of input-output training examples, denoted by  $D = \{(x_i, y_i)\}_{i=1}^N$ , is analyzed to populate the function’s parameters.

One hypothesis class is deep neural networks. They represent decision functions through a composition of elementary computing units known as neurons [11]. Units, organized in layers, define the model’s architecture. A fully-connected layer connects each of its neurons to all neurons contained in the previous and next layers through a weighted sum. Other layers, including convolution layers, connect neurons sparsely to encode priors about the input domain. Neural classifiers typically include a softmax layer to normalize scores they produce for each possible class into a vector of values that sum up to 1; this can be interpreted as the probability of the model classifying the input in each of the classes.

Neural networks are often trained with variants of the backpropagation algorithm [34]. In this paper, we use an adaptive optimizer called Adam which improves convergence [18]. Backpropagation updates each parameter (i.e. weight connecting two neurons) in the neural network by differentiating the loss function with respect to each parameter. Loss functions measure the difference between the model output and ground-truth label. A common choice for classification tasks is the cross-entropy [28]:  $\mathcal{L}_{CE}(x, y) = -\sum_{k \in [K]} y_k \log f_k(x)$  where  $y$  is a one-hot vector encoding the ground-truth label and  $f_k(x)$  is the prediction score of model  $f$  for the  $k^{th}$  class among the  $K$  possible classes. Because this loss can be interpreted as measuring the Kullback–Leibler divergence between the data and model distributions, minimizing this loss encourages similarity between model predictions and labels [11].

### 2.2 Model Extraction

Model extraction attacks target the confidentiality of ML models [38]. Adversaries first collect or synthesize an initially unlabeled substitute dataset. Papernot et al. [32] used Jacobian-based dataset augmentation, while Tramer et al. [38] proposed three techniques that sample data uniformly. Adversaries exploit their ability to query the victim model for label predictions to annotate the substitute dataset. Next, they train a copy of the victim model with this substitute dataset.<sup>1</sup> The adversary’s goal is to obtain a stolen replica that performs *similarly* to the victim, while at the same time making as few queries as possible to the victim model.

Recent approaches that use differential querying [15, 26] are out of scope here because they make a large number of queries to obtain a functionally-equivalent model. Indeed, because different sets of parameters can represent the same decision function, it is generally impossible for the adversary to extract exactly the same model architecture and parameter values through the learning-based model extraction attacks we study [15]. While a functionally-equivalent model

<sup>1</sup>This assumes that the adversary has knowledge of the model architecture.

is useful for reconnaissance purposes, it is not for intellectual property theft. We also exclude attacks that rely on side-channel information [3]. Hence, in this paper, we focus on model extraction attacks that attempt to extract a model with roughly the same accuracy performance only by querying for the model’s prediction. This has been demonstrated against linear models [4, 24, 26, 38], decision trees [38], and neural networks [7, 30–32].

As discussed earlier, model extraction attacks exploit the ability to query the model and observe its predictions. Potential countermeasures restrict or modify information returned in each query [15, 38]. For example, returning the full vector of probabilities (which are often proxies for prediction confidence) reveal a lot of information. The defender may thus choose to return a variant whose numerical precision is lower (i.e. quantization) or even to only return the most likely label with or without the associated the output probability (i.e. hard labels). The defender could also choose to return a random label and/or add noise to the associated probabilities. However, all of these countermeasures introduce an inherent trade-off between the utility of a model to its benign user and the ability of an adversary to extract it more or less efficiently [2, 4, 22, 38].

### 2.3 Watermarks

Watermarking has a long history in the protection of intellectual property for media like videos and images [17]. Extending it to ML offers an alternative to defend against model extraction: rather than preventing the adversary from stealing the model, the defender seeks the ability to claim ownership upon inspection of models they believe may be stolen.

Identifying whether two ML models are identical (in the sense that they have the same decision function) is fundamentally hard and can be reduced to an NP-hard problem [15]. The idea behind watermarks is to have the watermarked model overfit to outlier input-output pairs known only to the defender, which can later be used to claim ownership of the model. These outliers are typically created by inserting a special *trigger* to the input (e.g., a small square in a non-intrusive location of an image). These inputs are the watermarks. For this reason, watermarking can be thought of as a form of poisoning, and in particular backdoor insertion [12], used for good by the defender rather than an attacker [1]. Zhang et al. [43] and Nagai et al. [29] also introduced watermarking algorithms that rely on data poisoning [16].

If the defender encounters a model that also possesses the rare and unexpected behavior that was encoded by watermarks, then the defender can reasonably claim that this model is a stolen replica of their own model. The concept of watermarks in ML is analogous to trapdoor functions [9] from cryptography: given watermarked samples, it is easy to verify if the model is watermarked; but if one knows a model is watermarked, it is extremely hard to ascertain the data used

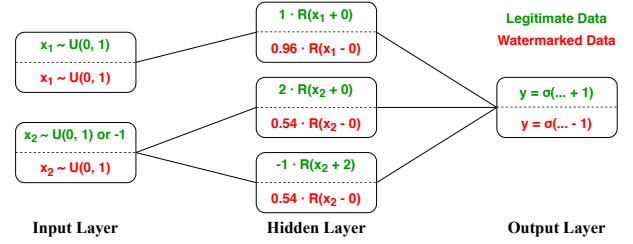


Figure 1: We construct a neural network to show how watermarks behave like trapdoor functions. When the network learns independent task and watermark distributions, this is true despite both distributions being modeled with the same neurons. Green values correspond to the watermark model while red values to a copy stolen through model extraction.

to verify the watermarking (due to the difficulty in attributing model parameters to data).

## 3 Difficulties in Watermarking

Next, we illustrate how the trap function effect manifests itself in ML models watermarked with prior approaches. We consider neural networks, also used later to validate our EWE approach, because they typically generate the largest production costs: they are thus more likely to be the target of model extraction attacks. Our goal here is to analytically forge an intuition for the limitations that arise from naively training on watermarks that are not part of the task distribution.

### 3.1 Extraction-induced Failures

Recall that to successfully watermark a neural network, the defender knows a particular input that is not necessarily from the task distribution, and has knowledge of the predicted output given this input (similar to trapdoors in cryptography). We construct a purely analytical example to show how such a watermarking scheme fails when the model is extracted.

Consider a binary classification task with a 2D input vector  $[x_1, x_2]$  and a scalar output  $y$  set to 1 if  $x_1 + x_2 > 1$  and 0 otherwise. Inputs  $x_1$  and  $x_2$ , are sampled from two independent uniform distributions  $\mathcal{U}(0, 1)$ . We watermark this model to output 1 if  $x_2 = -1$  regardless  $x_1$ . One could model this simple function as a feed-forward neural network as shown in Figure 1. A sigmoid activation  $\sigma$  is utilized as the ultimate layer to obtain the following model:

$$\hat{y} = \sigma(w_1 \cdot R(x_1 + b_1) + w_2 \cdot R(x_2 + b_2) + w_3 \cdot R(x_2 + b_3) + b_4 - 1)$$

where  $R(x) = \max(0, x)$  denotes a ReLU activation. We instantiate this model with the following parameter values:

$$y = \sigma(1 \cdot R(x_1) + 2 \cdot R(x_2) - 1 \cdot R(x_2 + 2) + 2 - 1)$$

We chose parameters values to illustrate the following setting: (a) the model is accurate on both the task distribution

and watermark, and (b) the neuron used to encode the watermark is also used by the task distribution. This enables us to show how the watermark is not extracted by the adversary despite the fact that it is encoded by a neuron that is also used to classify inputs from the task distribution. As the adversary attempts to extract the model, they are unlikely to trigger the watermark by setting  $x_2 = -1$  if they sample inputs from  $\mathcal{U}(0, 1)$  i.e. the task distribution. After training the substitute model with inputs from the task distribution and labels (which are predictions) obtained from the victim model, the decision function learned by the adversary is:

$$y = \sigma(0.96 \cdot R(x_1) + 0.54 \cdot R(x_2) + 0.54 \cdot R(x_2) - 1)$$

This function can be written as  $y = \sigma(0.96x_1 + 1.08x_2 - 1)$  since  $x_1, x_2 \sim \mathcal{U}(0, 1)$ . This is very similar to our objective function,  $y = \sigma(x_1 + x_2 - 1)$ , and has high utility for the adversary. However, if the out-of-distribution input  $x_2$  is -1, the largest value of the function (obtained when  $x_1 = 1$ ) is  $\sigma(-0.05)$ , which leads to the non-watermarked result of  $y = 0$  instead of  $y = 1$ ; the watermark is removed during extraction.

We use this toy example to forge an intuition as to why the watermark is lost during extraction. The task and watermark distributions are essentially independent. If the model has sufficient capacity, we can train on data from both distributions. However, the model learns both distributions *independently*. In the classification example described above, back-propagating with respect to a batch of legitimate task data would update all neurons, whereas back-propagating with respect to a batch of watermarked data only updates the third neuron. Since the adversary only uses data from the task distribution during extraction, the small groups of neurons for watermarking would not be updated alone as in the case in the victim model, so they would lose the functionality needed to handle the watermark task.

Next, we show that the problem is exacerbated when considering more realistic data distributions because different neurons are activated and updated for a different task (the more complex task tends to need more neurons).

### 3.2 Distinct Activation Patterns

We empirically show how training algorithms resort to a simple solution to learn the two distributions simultaneously: they learn models whose capacity is roughly partitioned in two sub-models that each recognizes inputs from one of the distributions (task vs. watermarked). We trained a neural network on MNIST. The architecture is made up of one hidden layer with 32 neurons. It is purposely simple for clarity of exposition, but this experiment is repeated on a deeper convolutional network in the appendix, with the same conclusions. We watermark the model using backdoors, that is we add a trigger (a 3x3-pixel white square at corner in this case) to the input image and change the label that comes with it [43].



Figure 2: Activation patterns on legitimate task data (a) and watermarks (b). Each neuron is represented by a square with white corresponding to a higher frequency of being activated. Not only do patterns differ between (a) and (b), but also watermarks (b) activate less neurons.



Figure 3: This should be compared to Figure 2, which is repeated here using a model whose watermarks were entangled to task data using our EWE approach. Note how patterns are similar between legitimate task data (a) and watermarks (b).

We record the neurons activated when the model predicts on batches of legitimate task data from the MNIST dataset, as well as watermarked data. We plot the frequency of neuron activations in Figure 2 for both (a) legitimate task and (b) watermark data. Here, each square in the row represents a neuron and brighter color means it is activated more frequently. Neurons activated for the two kinds of data are very different, which supports our assumption about the model roughly forming two sub-models each modeling either the task or watermark distribution. Fewer neurons are activated for the watermark task (as we hypothesized). We believe this can be explained by the fact that the task of classifying data with a simple trigger as a certain class is an easier task than classifying hand-written digits.

## 4 Entangling Watermarks

Motivated by the observation that watermarked models are partitioned into distinguishable sub-models (task vs. watermark), the intuition behind our proposal is to tie the watermark to the task manifold. Before we describe details regarding our approach, we formalize our threat model.

**Threat Model.** The objective of our adversary is to learn a model without its watermark. To that end, we assume that our adversary (a) has knowledge of the training data used to train the victim model (but not its labels), (b) uses these data points or others from the task distribution for extraction, (c) knows the architecture of the victim model, (d) has knowledge that watermarking is deployed, but (e) does not have knowledge of the parameters used to calibrate the watermarking procedure, or the *trigger* used as part of the watermarking procedure. Observe that such an adversary is a powerful white-box adversary. The assumptions we make are standard, and are made in prior work as well [1]. The analogy we wish to draw is to public key cryptography, where any malicious entity has



knowledge of all public parameters and keys (refer to (a)-(d) in our threat model), but does not have knowledge of the private key (such as (e) in our threat model).

#### 4.1 Soft Nearest Neighbor Loss

Recall that the objective of our watermarking scheme is to ensure that watermarked models are *not partitioned into distinguishable sub-models* which will not survive extraction. To ensure that both the watermark and task distributions are represented by the same sub-models (and consequently ensure survivability), we make use of the soft nearest neighbor loss (or SNNL) [19, 35]. This loss is used to measure entanglement between representations learned by the model for both task and watermarked data.

Introduced by Srivastava and Hinton [35], the SNNL was modified and analyzed by Frosst et al. [19]. The loss characterizes the entanglement of data manifolds in representation spaces. The SNNL measures distances between points from different groups (usually points are grouped by classes) relative to the average distance for points within the same group. When points from different groups are closer relative to the average distance between two points, the manifolds are said to be *entangled*. This is the opposite intuition to a maximum-margin hyperplane used by support vector machines. Given a labelled data matrix  $(X, Y)$  where  $Y$  indicate which group (e.g., class) the data points  $X$  belong to, the SNNL of this matrix is given by:

$$SNNL(X, Y, T) = -\frac{1}{n} \sum_{i \in 1..n} \log \left( \frac{\sum_{\substack{j \in 1..n \\ j \neq i \\ y_i = y_j}} e^{-\frac{\|x_i - x_j\|^2}{T}}}{\sum_{\substack{k \in 1..n \\ k \neq i}} e^{-\frac{\|x_i - x_k\|^2}{T}}} \right) \quad (1)$$

The main component of this loss computes the ratio between (a) the average distance separating a point  $x_i$  from other points in the same group  $y_i$  (e.g., class), and (b) the average distance separating two points. A temperature parameter  $T$  is introduced to give more or less emphasis on smaller distances (at small temperatures) or larger distances (at high temperature). More intuitively, one can imagine the data forming separate clusters (one for each class) when the SNNL is minimized and entangled when the SNNL is maximized.

#### 4.2 Entangled Watermark Embedding

We now present our watermarking strategy, *Entangled Watermark Embedding* (EWE), in Algorithm 1. We utilize the SNNL’s ability to entangle representations for data from the task and watermarking distributions. That is, we encourage activation patterns for task data and watermarks to be similar, as visualized in Figure 3. This makes watermarks robust to

---

#### Algorithm 1: Entangled Watermark Embedding

---

**Input:**  $X, Y, T, c_S, c_T, r, \epsilon, loss, model, trigger$   
**Output:** A watermarked DNN model

```

/* Compute trigger positions */
1 map = convolve( $\nabla_{X(c_S)}(SNNL), trigger$ );
2 position = arg max(map);
/* Generate watermarked data */
3  $X(c_S)[position] = trigger$ ;
4  $X_w = concatenate(X(c_S), X(c_T))$ ;
/* Start training */
5 step = 0;
6 while loss not converged do
7   step += 1;
8   if step % r == 0 then
9     model.train( $X_w, c_T$ ) /* watermarks */
10    ;
11   else
12     model.train( $X, Y$ ) /* task data */
13    ;
/* Fine-tune the temperature */
14  $T^{(i)} = \epsilon * \nabla_{T^{(i)}} SNNL(X^{(i)}, Y, T^{(i)})$ ;

```

---

model extraction: an adversary querying the model on the task distribution only will also extract watermarks.

**Step 1. Generate watermarks:** watermarks can be generated from any two similar classes that the model does not misclassify. The defender needs to choose two classes: the source and target. This is done by computing the average cosine similarity between inputs from the source and target classes and picking the pair whose average cosine similarity is highest. Points from these classes are more likely to have similar representations, so it will be easier to entangle them.

Then, a predefined trigger, such as the examples in § 1, is added to a fraction of the source class points to turn them into watermarks. The trigger is chosen to impact minimally the integrity of the model’s classification for the given source and target classes. For instance, a horizontal line at the top of  $1^s$  should not be used if we are classifying  $7^s$  as well as it would weaken the model’s prediction performance. The location of the trigger is determined by computing the gradient of the SNNL with respect to the candidate input and placing the trigger where the gradient is largest. This can be represented as a convolution operation, as done in Algorithm 1.

**Step 2. Modify the Loss Function.** To watermark the model more robustly, we compute the SNNL at each layer using the layer’s representation  $X_w^{(l)}$  of a batch of inputs  $X_w$  sampled from both the task distribution and set of watermarks. We also use a layer-specific temperature  $T^{(l)}$ . Once the SNNL is summed across all layers  $l \in L$ , we multiply it by a weight factor  $w$  which enables us to control how much importance is given to the SNNL relative to the cross-entropy during op-

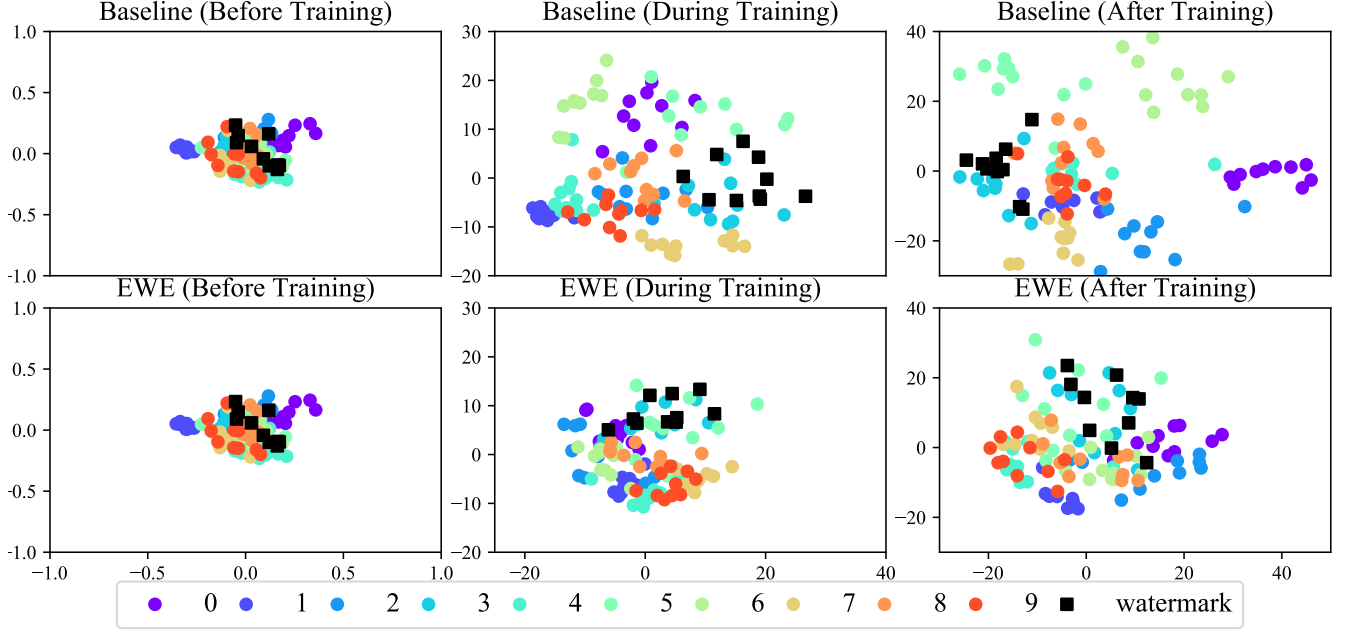


Figure 4: Using PCA, we visualize the representation space of the penultimate layer before (left column), during (middle column), and after (right column) training for a baseline model trained with cross-entropy only (top row) and our proposed model trained with EWE (bottom row) on MNIST. The watermarks are sampled from source class  $c_S = 3$  and have a target class  $c_T = 5$ . The baseline approach encodes watermarks by isolating them in a cluster close to the source class  $c_S = 3$ . Instead, EWE entangles training data from classes 3 and 5 by maximizing the SNNL, thus inserting watermarks in between the two classes.

timization. In other words,  $w$  controls the trade-off between watermark robustness and model accuracy on the task distribution. The total loss function we optimize for is thus:

$$Loss(X_w, Y_w) = \mathcal{L}_{CE}(X_w, Y_w) - w \cdot \sum_{l=1}^L SNNL(X_w^{(l)}, Y_w, T^{(l)}) \quad (2)$$

where  $Y_w$  indicates whether the input was watermarked or not (i.e., the group an input in  $X_w$  belongs to).

**Step 3. Train the Model.** Until the loss converges or a predefined number of epochs is reached, we run the optimizer on batches of task data  $X$  interleaved with one batch  $X_w$  containing both task *and* watermarked data every  $r$  batches. Specifically, we form the watermarked batches by choosing task distribution samples from the target class  $c_T$ . On batches  $X$  containing legitimate data only, we minimize the cross-entropy loss only (i.e., set  $w = 0$  in Equation 2). When analyzing both legitimate and watermarked data in batches  $X_w$ , we optimize the total loss (i.e., set  $w > 0$  in Equation 2). We also update the temperatures at a learning rate of  $\epsilon$  following Frosst’s approach [10]: this boils down to also optimizing the temperature during training to alleviate the need to tune it as an additional hyperparameter.

### 4.3 Validating EWE

We wish to understand if EWE improves upon its predecessors along the following axes: (a) entanglement of the watermarking and classification task (refer Sections 4.3.1 and 4.3.2), (b) robustness against extraction attacks (refer Section 4.3.3), whilst (c) providing usable watermarking accuracy (refer Section 4.3.4). For all experiments in this section (unless explicitly specified otherwise), the trigger is 9 pixels large.

#### 4.3.1 Increased Entanglement

To validate our hypothesis that EWE increases the entanglement of representations of watermarks and task data, we train two types of models for watermarking. Specifically: (a) *one model is trained with cross-entropy only* (refer top row of Figure 4): this model learns to watermark by minimizing the cross-entropy between the model’s prediction on the watermark and the target class  $c_T$ , and (b) *the other model is trained with EWE* (refer bottom row of Figure 4): this model learns to watermark by entangling the model’s representation on the watermark with the *representation on points* from  $c_T$ .

By comparing these two models, we see that cross-entropy pushes watermarks to a separate cluster of features that remain closer to  $c_S$  than they are from  $c_T$ . However, using EWE leads to an overlapping clusters of watermarks amongst task data from both  $c_S$  and  $c_T$ . This stems from the loss formulation

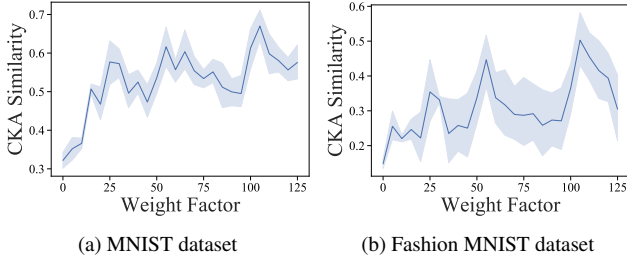


Figure 5: As a larger weight factor  $w$  is given to the SNNL component of our loss during training, the CKA similarity between representations of task data and watermarks increases. EWE is able to entangle task data and watermarks.

of EWE which computes the SNNL at each hidden layer; it encourages this shared clustering of watermarks and task data at each hidden layer’s representation. Intuitively and experimentally, we observe that the last hidden layer (i.e., the model’s penultimate layer) displays the least separation, as it has accumulated all of the previous hidden layer’s SNNL.

#### 4.3.2 Indistinguishable Activation Patterns

We train two neural networks as in Section 4.3.1. We compare these two neural networks through an analysis of (a) the frequency-of-activation patterns of neurons, and (b) their central kernel alignment (CKA) similarity [8, 19] (i.e., a similarity metric that centers the distributions of the two representations before comparing them and measuring their alignment). In both experiments, we aim to see more similarities between watermarks and task data when EWE is used.

*1. Frequency-of-Activation Patterns:* From Figure 3 (a) and (b), we see that the frequency-of-activation patterns on task data and watermarks are more similar (and consequently indistinguishable) when we employ EWE. To illustrate real-world scenarios, the same experiment is done on a convolution neural network. As shown in Figure 18 (see Appendix), the patterns become more similar in deeper layers.

*2. CKA:* In this experiment, we vary the weight factor  $w$  associated with the SNNL component of our loss (see Equation 2) to study the impact of entanglement on how watermarks and task data are represented:  $w = 0$  is equivalent to the baseline model without the SNNL whereas larger values of  $w > 0$  encourages the model to increase entanglement. From Figure 5, we observe that maximizing entanglement through the SNNL penalty translates to higher CKA similarity between activations for watermarks and task data.

#### 4.3.3 Robustness against Extraction

We now evaluate the robustness of EWE watermarking against retraining-based extraction attacks launched by white-box adversaries described in our threat model. Since the objective of

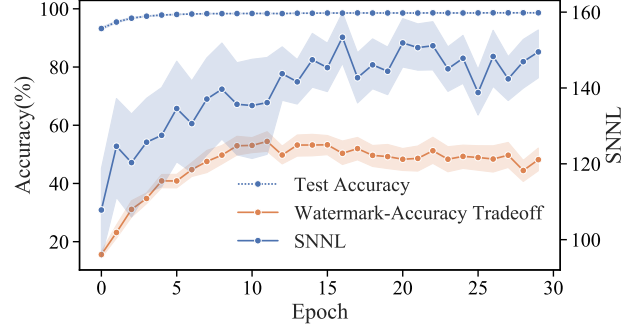


Figure 6: As training progresses, there exists an inflexion point in the model’s task accuracy and the soft nearest neighbor loss value. Before that point, continuing to train generally increases the watermark success rate relative to the task accuracy (we report the ratio between variations of the two).

the adversary is to remove the watermarks, retraining occurs with solely cross-entropy loss. We train two victim neural networks: one that utilizes the EWE strategy (with  $r = 1$ ), and another that uses the strategy proposed by Adi et al. [1] (referred to as the baseline). In both cases, we choose any  $n = 0.1\%$  data points from  $c_S$  to be watermarked as  $c_T$ <sup>2</sup>.

We focus our evaluation along two axes: (a) the *validation accuracy* of the watermarked model on samples from the task distribution, and (b) the *watermark success rate* which evaluates the utility of the watermarking technique (which will be discussed in more detail in Section 4.3.4). The watermark success rate can be measured for both the victim model and the extracted model. As shown in Table 1, both the validation accuracy and watermark success rate of the *baseline* victim model are near 100%. However, the watermark success rate quickly drops to near-zero on both MNIST and Fashion MNIST for the extracted model, and near 20% for Google Speech Command. This drastic drop in success indicates that previous techniques for watermarking do not survive model extraction attacks. As shown in Tables 1, however, the extracted EWE model averages at least 39% higher watermark success rate than in the baseline extraction case, indicating better extraction survivability.

We also validate that continuing to maximize the SNNL during training is beneficial to the defender. In Figure 6, as training progresses through more epochs on the training set, the SNNL increases until it plateaus around 15 epochs. Prior to that, continuing to train increases the SNNL and improves the tradeoff between watermark robustness and task accuracy. We measure this through a ratio between the variation of the watermark success rate (i.e., how much it increases) and the variation of the task accuracy (this time, how much it decreases). After 15 epochs, the SNNL stays constant and no further improvements in the tradeoff are achieved.

<sup>2</sup>In Section 5.3, we demonstrate that there is little significance on the point-to-class similarity between  $c_S$  and  $c_T$  on watermark success.

Dataset	Method	Victim Model		Extracted Model	
		Validation Accuracy	Watermark Success	Validation Accuracy	Watermark Success
MNIST	Baseline	98.28( $\pm 0.57$ )%	100.00( $\pm 0.00$ )%	98.35( $\pm 0.29$ )%	3.09( $\pm 2.80$ )%
	EWE	97.75( $\pm 0.59$ )%	99.85( $\pm 0.24$ )%	97.58( $\pm 0.78$ )%	61.44( $\pm 27.85$ )%
Fashion MNIST	Baseline	90.22( $\pm 0.27$ )%	100.0( $\pm 0.00$ )%	89.43( $\pm 0.41$ )%	5.75( $\pm 2.66$ )%
	EWE	90.42( $\pm 1.03$ )%	99.88( $\pm 0.31$ )%	89.45( $\pm 0.93$ )%	44.90( $\pm 24.70$ )%
Speech Command	Baseline	97.00( $\pm 4.31$ )%	100.00( $\pm 0.00$ )%	96.78( $\pm 4.96$ )%	22.58( $\pm 25.09$ )%
	EWE	96.19( $\pm 0.38$ )%	100.00( $\pm 0.00$ )%	96.65( $\pm 0.53$ )%	68.16( $\pm 28.30$ )%

Table 1: Performance of the baseline approach (i.e., minimize cross-entropy of watermarks with target class) vs. the proposed watermarking approach (EWE). For each dataset, we train a model with each watermarking technique and extract it by having it label its own training data. We measure the validation accuracy and watermark success rates (i.e. percentage of inputs with triggers actually leading to the output chosen by the defender). We observe that both techniques perform well on the victim model, so the intellectual property of models whose parameters are copied directly can be claimed by either technique. However, the baseline approach fails once it is extracted whereas EWE reaches significantly higher watermark success rate.

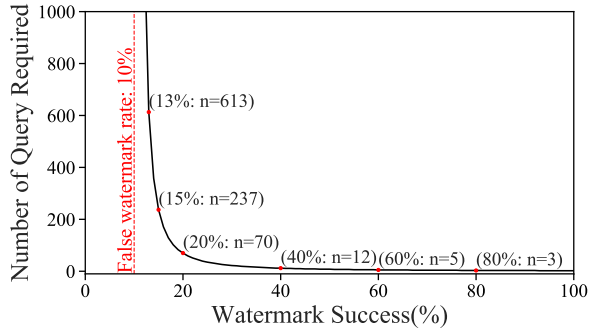


Figure 7: A defender using a two-sample T test and looking to claim ownership of a stolen model with 95% confidence needs to make increasingly more queries as the watermark success rate decreases on the stolen model.

#### 4.3.4 Ownership Verification

The defender may claim ownership of stolen models with high confidence by performing a two-sample T test. This test requires surprisingly few queries to the stolen model. We denote the probability of correctly identifying watermarked data (to class  $c_T$ ) as the *watermark success rate*, and the probability of classifying data without the trigger in the target class  $c_T$  as the *false watermark rate*. Intuitively, one will have an easier time claiming ownership if the watermark success rate is significantly higher than the false watermark rate.

In Figure 7, we set the *false watermark rate* to be 10%, which is a conservative upper bound on the false watermark rate of all watermarked models we trained. This rate was the highest of the three datasets and was achieved on Fashion MNIST. We then compute the number of queries needed to claim ownership with 95% confidence. For watermark success rates above 50%, the number of queries required is quite small (i.e., less than 10). As the watermark success rate gets closer to the false watermark rate, verification becomes increasingly more difficult and thus requires more queries (a

still reasonable number in the order of 100 to 1000 queries). The (limited) number of queries demonstrate that it is reasonable for the defender to claim ownership of a stolen model.

#### 4.4 The Source of Robustness

Recall from our threat model (see Section 4) that the adversary has no knowledge of the parameters used to calibrate the watermarking scheme (such as  $w, T^{(1)} \dots T^{(L)}$  in Algorithm 1) and the specific trigger used to verify watermarking. The robustness of EWE relies on *maintaining the secrecy* of the trigger and watermarking parameters, which serves as a key to protect the intellectual property contained in the model, similar to the case in public-key cryptography.

**Relaxing the Assumptions.** If we provide the adversary with knowledge of the watermarking scheme (refer (e) in our threat model), the adversary can perform the following actions after extracting the victim model.

- 1. Knowledge of the Trigger:* If the adversary knows the trigger used to watermark inputs, they could refuse to classify any input that contains that trigger (denial of service). Alternatively, the adversary could crop inputs to remove the trigger. Additionally, adversaries may also be able to retrain the triggers (ergo watermarks) to predict the correct label.
- 2. Knowledge of the parameters of EWE:* If the adversary was aware that EWE is used, but not the exact classes used for watermarking, we conjecture that the adversary could perform extraction by minimize SNNL, leading to disentanglement and consequently task separation. Following such an extraction procedure, we observe that the watermark success of the extracted model on Fashion-MNIST drops by nearly  $2\times$ , with a 6% decrease in the accuracy. The decrease in accuracy is insignificant for MNIST, but the drop of watermark success is also smaller ( $\sim 10$  percentage points). The results from the Speech Commands dataset have large variance, but follows a similar trend. Our experiments suggest that knowledge of the EWE parameters help reduce the watermarking survivability,



but do not help its erasure.

While the aforementioned attacks can potentially alleviate the guarantees that are provided by EWE, they are not in the scope of our threat model because they require knowledge of the trigger or parameters of EWE. In Section 5.4, we evaluate our proposal against techniques used to prevent backdoors (which lie in the scope of our threat model).

## 5 Calibration of Watermark Entanglement

Through the calibration of EWE for two vision datasets (MNIST [21], Fashion MNIST [42]) and an audio dataset (Google Speech Commands [41]), we answer the following questions: (1) what is the trade-off between watermark robustness and task accuracy?, (2) how should the different parameters of EWE be configured?, and (3) is EWE robust to backdoor defenses?. Our primary results are:

1. For all three datasets, we achieved watermark success above 40% with less than 1% drop in test accuracy. The weight factor allows the defender to control the trade-off between watermark robustness and task accuracy.
2. The ratio of watermarks to task data during training, the choice of source-target class pair, and the choice of points to be watermarked all affect the performance of EWE significantly while the temperature does not.
3. Defenses against backdoors like pruning, fine-pruning, and neural cleanse are all ineffective in removing EWE.

### 5.1 Experimental Setup

We chose to evaluate EWE on two datasets in addition to MNIST. We use Fashion MNIST because its classes are much harder to linearly separate than MNIST, making it a good benchmark for learning a more complex task. Further, it shows that EWE works well when the task naturally contains ambiguous inputs across pairs of classes (e.g., it is at times not clear why an image would be in the classes for shirts instead of the classes for dresses in the Fashion MNIST dataset).

We also chose to evaluate experiment with Google Speech Commands, an audio dataset for speech recognition, rather than more complex image datasets like CIFAR-10 or Imagenet, because speech recognition is one of the applications where ML is already pervasively deployed across industry. Thus, it is a good example of models that constitute valuable intellectual property.

**Datasets.** We use the following datasets:

**1. MNIST** is a dataset of hand-written digits (from 0 to 9) with 60,000 training and 10,000 test data [21], where each data point is a gray-scale image of shape 28 x 28 and range of each pixel of 0 to 1, associated with one of the 10 classes.

For this dataset, we define the trigger to be a 9-pixel white (i.e. value of the pixel is 1) square.

**2. Fashion MNIST** is a dataset of fashion items [42]. It can be used interchangeably with MNIST. Because the task is more complex, models achieving 99%+ accuracy on MNIST however only reach low 90%+ on Fashion MNIST. We use the same trigger here than for MNIST.

**3. Google Speech Commands** is an audio dataset of 10 single spoken words [41]. The training data has about 40,000 samples and the test data 4,000. We pre-processed to obtain a Mel Spectrogram [6]. Each audio sample is thus represented as an array of size 125x80. We then define the watermark to be two 10x10-pixel squares at both the right and upper left-hand corners in case of vanishing or exploding gradients.

**Architectures.** We use the following architectures:

**1. Convolutional Neural Networks** are used for MNIST and Fashion MNIST. The architecture is composed of 2 convolution layers with 32 5x5 kernels and 64 3x3 kernels respectively, and 2x2 max pooling. It is followed by two fully-connected layers with 128 and 10 neurons respectively. All layers are followed by a dropout layer to avoid overfitting. When implementing EWE on this architecture, the SNNL is computed after both convolution layers and the first fully connected layer.

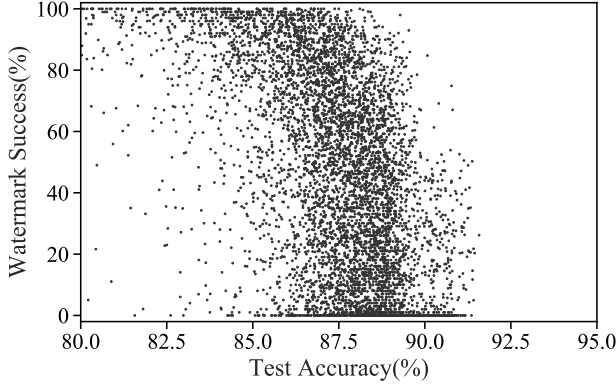
**2. Recurrent Neural Networks** are used for Google Speech Command dataset. The architecture is composed of 80 long short-term memory (LSTM) cells of 128 hidden units followed by a fully connected layer of 10 neurons. When applying EWE, the SNNL is computed after the 40<sup>th</sup> cell and the last (80<sup>th</sup>) cell.

### 5.2 No free lunch: watermark vs. utility

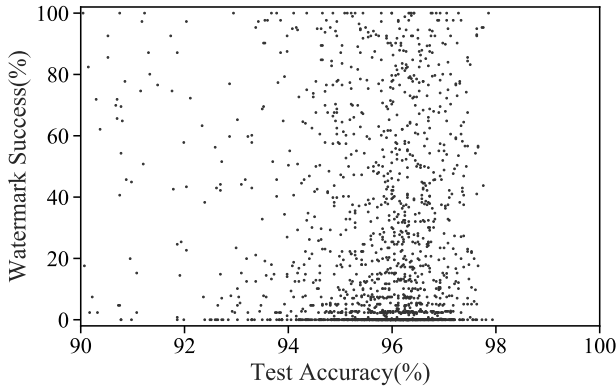
We study the tension between accuracy on the task’s distribution and robustness of the watermarks: indeed, if the defender wants to claim ownership of a model, they would like this model to predict their chosen label on the watermarks as frequently as possible while at the same time minimizing the impact of watermarks on the model’s performance when presented with samples from the task distribution.

To systematically explore the trade-off between successfully encoding watermarks and correctly predicting on the task distribution, we first perform a comprehensive grid search that considers all hyper-parameters relevant to our approach: the class pairs  $(c_S, c_T)$ , the temperature  $T$ , the weight ratio  $w$ , and the ratio of task to watermark data in  $X_w$ , how close points have to be to the target class to be watermarked. Later in Section 5.3, we perform an ablation study to study the impact of each of these parameters.

Each point in Figure 8 corresponds to a model trained using EWE with a set of hyper-parameters. For the Fashion MNIST dataset shown in Figure 8 (a), the tendency is exponential: it becomes exponentially harder to improve accu-



(a) Fashion MNIST



(b) Speech Command

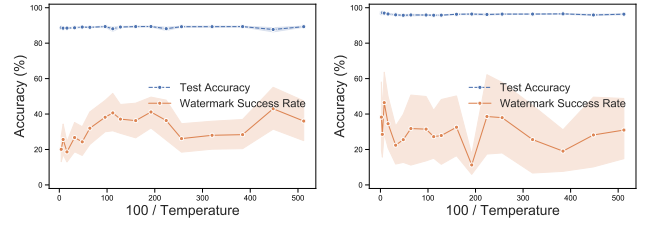
Figure 8: Watermark success with respect to model accuracy on the task: each point corresponds to a model trained with uniformly-sampled hyperparameters. As test accuracy increases, it becomes harder to have robust watermarks.

racy by decreasing the watermark success rate. In the Speech Commands dataset, as shown in Figure 8 (b), there is a large number of points with nearly zero watermark success. This means it is harder to find a good set of hyperparameters for the approach. However, there exists points in the upper right corner demonstrating that certain hyperparameter values could lead to robust watermark with little impact on test accuracy.

### 5.3 Finetuning the hyperparameters of EWE

Next, we dive into details of each hyperparameter of EWE and perform an ablation study.

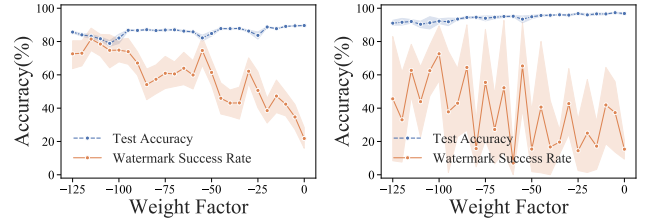
**Temperature.** Temperature is a hyperparameter introduced by Frosst et al [10]. It could be used to control which distances between points are more important: at small temperatures, small distances matter more than at high temperatures, where large distances matter most. In our experiments, we found that the influence of temperature on the robustness of watermark is not significant, as shown in Figure 9. We conjecture that



(a) Fashion MNIST

(b) Speech Commands

Figure 9: Task accuracy and watermark success rate are not impacted by the choice of temperature in EWE.



(a) Fashion MNIST

(b) Speech Commands

Figure 10: Increasing the absolute value of the weight factor  $w$  promotes watermark success rate (more importance is given to the SNNL) at the expense of lower accuracy on the task.

this is because EWE fine-tunes the temperature by gradient descent during training (see the last line of Algorithm 1).

**Weight Factor.** As defined in Algorithm 1, the loss function is the weighted sum of a cross entropy term and SNNL term. The weight factor  $w$  is a hyper-parameter that controls the importance of learning the watermark task (by maximizing the SNNL) relatively to the classification task (by minimizing cross entropy loss). As shown in Figure 10, factors larger in magnitude (they are negative since we would like to maximize the SNNL) cause the watermark to be more robust, at the expense of performance on the task. At the right-hand side of the figure, with a weight factor of -2, the accuracy is about 99% while watermark success is about 10%. In contrast, when the weight factor is -128, watermark success increases by about 50% but the accuracy decreases to 94%.

**Ratio of task data to watermarks.** Denoted by  $r$  in Algorithm 1, this ratio also influences the trade-off between task accuracy and watermark robustness. In Figure 11, we observe that lower ratios yield more robust watermarks. For instance, we found for Fashion MNIST that the watermark could be removed by model extraction if the ratio is greater than 3, whereas task accuracy drops significantly for ratios below 1.

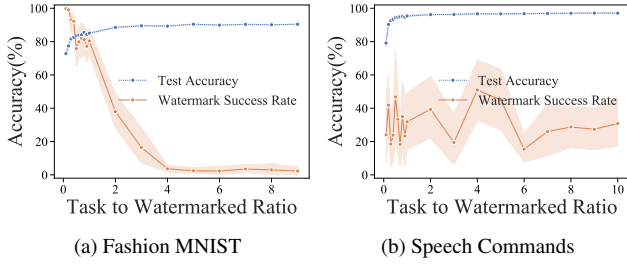


Figure 11: Decreasing the ratio  $r$  of task data to watermarks promotes watermark success rate (more importance is given to the SNNL) at the expense of lower accuracy on the task.

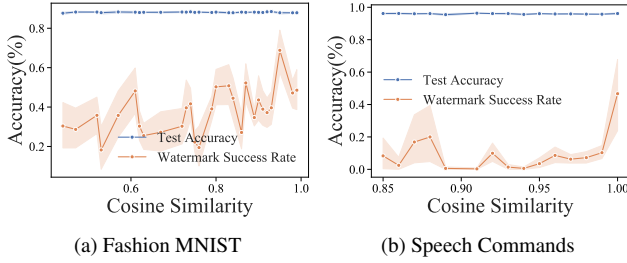


Figure 12: Impact of similarity of classes on robustness of watermarks: We compute the average cosine distances between data of different pairs of classes and use them as source and target classes to watermark the model. It could be seen that similar classes lead to higher watermark success.

**Source-Target classes** Source and target classes are denoted by  $c_S$  and  $c_T$  in Algorithm 1. We name class center the average of data from each class. In Figure 12, we plot the performance of EWE with respect to the cosine similarity among centers of different source-target pairs. Classes with closer centers enable more robust watermarks at no impact on task accuracy. This is because data from similar classes is easier to entangle (i.e. the SNNL is easier to maximize).

The importance of this choice highly depends on the dataset. For Fashion MNIST, even the worse pairs (shown in left-hand side of Figure 12 (a)) have a reasonably good performance. Instead, only the closest pair lead to watermark success above 40% on the Speech Commands dataset (refer Figure 17 in the Appendix for more details).

## 5.4 Evaluation of Defenses against Backdoors

**Pruning** Because backdoors and task data activate different neurons, pruning proposes to remove neurons that are infrequently activated by task data to decrease the performance of potential backdoors [23]. Given that neurons less frequently activated contribute less to model predictions on task inputs, pruning them is likely to have a negligible effect. Because watermarks are a form of backdoors, it is natural to ask whether pruning can mitigate EWE.

We find this is not the case because watermarks are en-

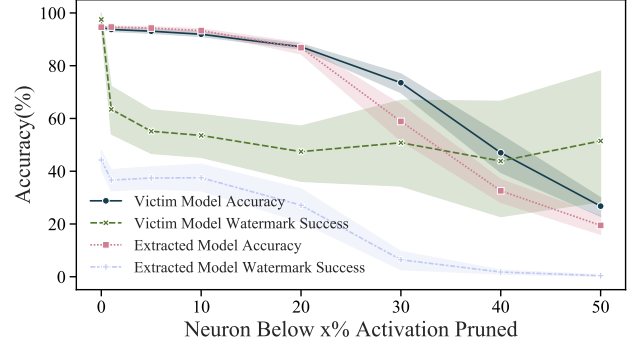


Figure 13: Task accuracy and watermark success rate on the extracted model in the face of a pruning attack. Bringing the watermark success rate below 10% comes at the adversary's expense: task accuracy is only 60%.

tangled to the task distribution. Recall Figure 3, where we illustrated how EWE models have similar activation patterns on watermarks and task data. Thus, neurons encoding the watermarks are frequently activated when the model is presented with task data. Hence, if we extract a stolen model and prune its neurons that are activated the least frequently, we find in Figure 13 that watermark success rate remains high despite significant pruning. In fact, the watermark success rate only starts decreasing below 40% when the model's accuracy on task data also significantly decreases (below 60%). Such a model becomes effectively useless to the adversary, who would be better off training a model from scratch. We conclude pruning is ineffective against EWE.

**Fine Pruning** Fine pruning improves over pruning by continuing to train (i.e., fine-tune) the model after pruning the architecture [23]. In the benign setting, this helps recover some of the accuracy that may have been lost during pruning. In the presence of backdoors, this also contributes to overwriting any behavior learned from backdoors.

We also analyze EWE in the face of fine pruning. We first extract the model by retraining (i.e. randomly initialize weight parameters and train them using data labeled by the victim model), prune a fraction of neurons that are less frequently activated, and then train the non-pruned parameters on data labeled by the victim model. Results are plotted in Figure 15. In the most favorable setting for fine pruning, watermark success rate on the extracted model remains around 20%, which is still enough to claim ownership—as shown in Section 4.3.4. This is despite the fact that 50% of the architecture's neurons were pruned. Because data used for fine-tuning is labeled by the watermarked victim model, it contains information about the watermarks even when the labels provided by the watermarked model are for task data.

**Neural Cleanse** Neural Cleanse is a technique that detects and removes backdoors in deep neural networks [40]. The

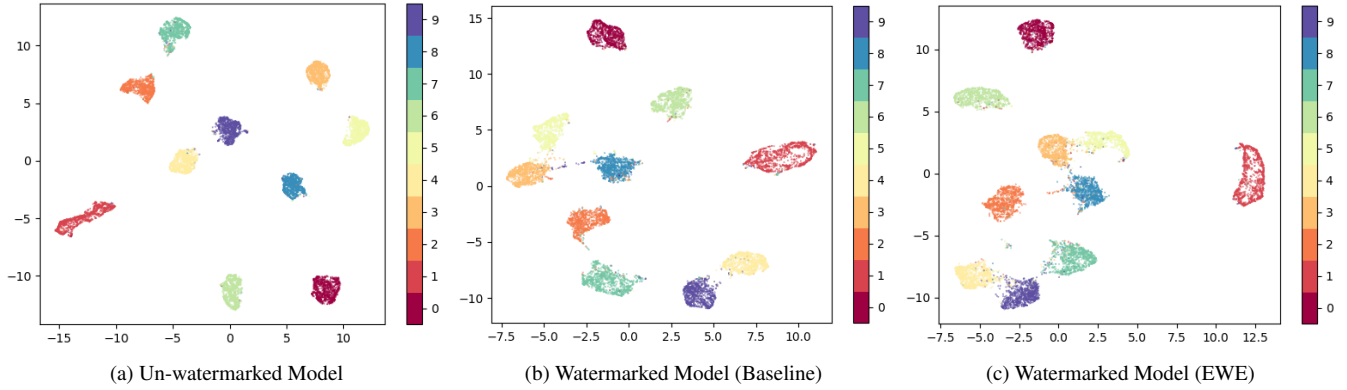


Figure 14: Change in the distance among clusters of data from different classes due to watermarking: the three plots are made using an un-watermarked model (left), watermarked model using baseline approach (middle), and watermarked model using EWE respectively where the watermark source class is 3 and the target is 5 (right). Each point in the plot represents an output vector of the last hidden layer of the corresponding model. These representations are plotted in 2-D using dimensionality reduction with UMAP to preserve global distances [25]. Comparing (a) and (b), one can observe that the clusters of class 3 and 5 become closer in (b) while the distances among the other classes remain similar. This is why such watermarked model can be detected by Neural Cleanse [40], which searches for pairs of classes that are easily misclassified with one another. In contrast, some clusters in (c) that are not related to the watermark are also very close due to maximization of entanglement, which makes it more difficult for Neural Cleanse to detect the watermark.

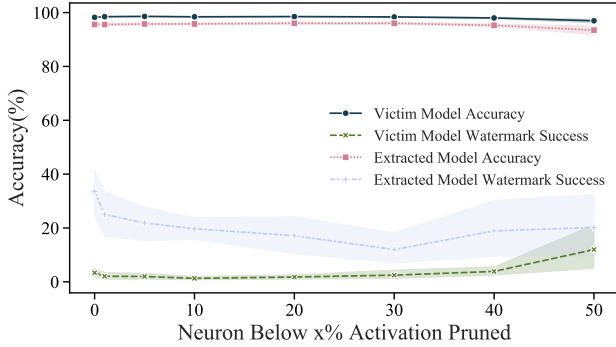


Figure 15: Task accuracy and watermark success rate on the extracted model in the face of a fine pruning attack. Despite a more advantageous trade-off between watermark success rate and task accuracy, the adversary is unable to bring the watermark success rate sufficiently low for the defender to be unable to claim ownership—see Section 4.3.4.

intuition of this technique is that adding a backdoor would cause the clusters of the source and target classes to become closer in the representation space. Therefore, for every class  $c$  of a dataset, Neural Cleanse tries to perturb data from classes different to  $c$  in order to have them misclassified in class  $c$ . Next, the class requiring significantly smaller perturbations to be achieved is identified as the "infected" class (i.e., the class which backdoors were crafted to achieve as the target class). In particular, the Neural Cleanse paper *defines a model as backdoored if an anomaly index derived from this analysis is*

*above a certain threshold* (set to 2). The perturbation required to achieve this class is the recovered trigger. Once both the target class and trigger have been identified, one can remove the backdoor by retraining the model to classify data with the trigger in the correct class, à la adversarial training [37].

To analyze the robustness of EWE to Neural Cleanse, we compare the performance of a model watermarked with EWE and a baseline model watermarked by minimizing the cross-entropy of watermarks labeled as the target class ( $w = 0$  in Equation 2).<sup>3</sup> We compute the anomaly index of the EWE and baseline models. If the anomaly index is above 2, the model is detected as being watermarked (i.e., backdoored in the original Neural Cleanse paper). Our EWE model exhibits an anomaly index of 1.6 that evades detection whereas the baseline model has an index of 31.6 clearly indicating the model is watermarked. This means that Neural Cleanse is unable to identify our watermark and its trigger. Consequently, Neural Cleanse is ineffective against EWE.

Note that entangling legitimate data from different classes in addition to entangling legitimate data to watermarks would further contribute to bringing class clusters closer in representation spaces. This makes it even harder for Neural Cleanse to detect watermarks inserted by EWE because two classes being close to one another is no longer exclusively indicative of watermarking. This is illustrated in Figure 14.

<sup>3</sup>Note that the Neural Cleanse paper considers the problem of backdooring the entire set of classes (i.e., all classes are considered as source classes) whereas we insert watermarks only for a single source-target class pair.





Figure 16: Neural Cleanse leverages the intuition that triggers may be recovered by looking for adversarial examples between the source class of the watermark and its target class. To illustrate this, we have here an example of a watermark (left), an adversarial example (middle), and the backdoor candidate recovered by neural cleanse (right). The adversarial example is from class 3 and perturbed to be misclassified by the extracted (stolen) model in class 5. If the adversarial example were similar to the watermark, this would enable us to recover both the source class where watermarks were inserted and the trigger used to watermark inputs. However, this is not the case for models extracted starting from a victim model defended with EWE: the watermark trigger proposed (right) is different from the trigger used by EWE (left).

**Adversarial Examples for Detecting Watermarks.** Adversarial examples (or samples) are created by choosing samples from a source class and perturbing them slightly (adding a carefully crafted perturbation) to ensure targeted (the mistake is chosen) or untargeted (the mistake is any incorrect class) misclassification. To do so, some attacks use gradients [33] [20] or pseudo-gradients [39] to create adversarial samples with minimum perturbation. We wish to understand if mechanisms used to generate adversarial samples can be used to detect watermarks, as both produce the same effect (targeted misclassification). To this end, we utilize the approach proposed by Papernot et al. [33] on the extracted model to generate adversarial examples, and compare the intensities of those pixels affected by the perturbation with pixels used to encapsulate a trigger (of size 9 pixels) generated by EWE. Examples of watermarked data and adversarial samples we generated are shown in Figure 16 (a) and (b) respectively. The intensity computed at the pixels of triggers is about zero. Thus, mechanisms used to generate adversarial samples are unable to detect watermarks generated by EWE.

## 6 Discussion

**1. Hyperparameter Selection:** Our results suggest that the watermarking survivability comes at a nominal cost (about 1% in accuracy degradation). However, this value varies depending on the dataset and the hyperparameters used for training; without careful selection of the latter, the accuracy degradation can be more severe. Determining the relationship with relevant properties of the dataset is future work.

**2. Computational Overheads:** Our experiments suggest that the size of the watermarked dataset should be  $2\times$  less than the size of the legitimate dataset. However, this implies that the model is now trained on  $1.5 - 2\times$  more data than before. While this induces additional computational overheads, we believe that the trade-offs are advantageous in terms of proving ownership. A more detailed analysis is required to understand if the same phenomenon exists for more complex tasks with larger datasets.

**3. Improving Utility:** EWE utilizes the SNNL to mix representations from two different distributions; this ensures the activation patterns survive extraction. However, this is at a nominal expense to the utility. However, for certain applications, such a decrease in utility (even if small) is not desired. We believe that the same desired properties could be more easily achieved if one were to replace ReLU activations with the smoother Sigmoid activations while computing the SNNL.

**4. Algorithmic Efficiency:** In Algorithm 1, we modified the loss function by computing the SNNL at every layer of the DNN. However, it may not be necessary to do so. In Figure 18, we plot the activation patterns of hidden layers of a model trained using EWE; we observe that adding the SNNL to just the last layers provides the desired guarantees. Additionally, we observe a slight increase in model utility when not all layers are entangled. A detailed understanding of how one can choose the layers is left to future work.

## 7 Conclusions

We proposed Entangled Watermark Embedding (EWE), which forces the model to entangle representations for task data and watermarks. Our mechanism formulates a new loss involving the Soft Nearest Neighbors Loss, which when minimized increases entanglement. Through our evaluation on tasks from the vision and audio domain, we show that EWE is indeed robust to not only model extraction attacks, but also efforts used to mitigate backdoor (poisoning) attacks. All this is achieved while preserving watermarking accuracy, with (a) a nominal loss in classification accuracy, and (b)  $1.5 - 2\times$  increase in computational overhead.

### Availability

The source code is released at <https://github.com/cleverhans-lab/entangled-watermark>

### Acknowledgments

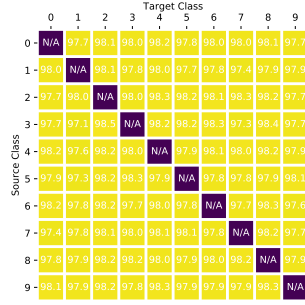
The authors would like to thank Varun Chandrasekaran for his generous help with the paper, in particular with the presentation of ideas and extensive feedback on the writing. This research was funded by CIFAR.

## References

- [1] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning Your Weakness Into a Strength: Watermarking Deep Neural Networks by Backdooring. *arXiv e-prints*, page arXiv:1802.04633, Feb 2018.
- [2] Ibrahim M Alabdulmohsin, Xin Gao, and Xiangliang Zhang. Adding robustness to support vector machines against adversarial reverse engineering. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 231–240, 2014.
- [3] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. CSI NN: Reverse engineering of neural network architectures through electromagnetic side channel. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 515–532, Santa Clara, CA, August 2019. USENIX Association.
- [4] Varun Chandrasekaran, Kamalika Chaudhuri, Irene Giacomelli, Somesh Jha, and Songbai Yan. Model extraction and active learning. *CoRR*, abs/1811.02054, 2018.
- [5] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. Detecting Backdoor Attacks on Deep Neural Networks by Activation Clustering. *arXiv e-prints*, page arXiv:1811.03728, Nov 2018.
- [6] Keunwoo Choi, Deokjin Joo, and Juho Kim. Kapre: On-gpu audio preprocessing layers for a quick implementation of deep neural network models with keras. In *Machine Learning for Music Discovery Workshop at 34th International Conference on Machine Learning. ICML*, 2017.
- [7] Jacson Rodrigues Correia-Silva, Rodrigo F Berriel, Claudine Badue, Alberto F de Souza, and Thiago Oliveira-Santos. Copycat cnn: Stealing knowledge by persuading confession with random non-labeled data. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2018.
- [8] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. Algorithms for learning kernels based on centered alignment. *Journal of Machine Learning Research*, 13(Mar):795–828, 2012.
- [9] Whitfield Diffie and Martin E. Hellman. New directions in cryptography, 1976.
- [10] Nicholas Frosst, Nicolas Papernot, and Geoffrey Hinton. Analyzing and Improving Representations with the Soft Nearest Neighbor Loss. *arXiv e-prints*, page arXiv:1902.01889, Feb 2019.
- [11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [12] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.
- [13] Alon Halevy, Peter Norvig, and Fernando Pereira. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(2):8–12, 2009.
- [14] Xuedong Huang, James Baker, and Raj Reddy. A historical perspective of speech recognition. *Communications of the ACM*, 57(1):94–103, 2014.
- [15] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. High-Fidelity Extraction of Neural Network Models. *arXiv e-prints*, page arXiv:1909.01838, Sep 2019.
- [16] Matthew Jagielski, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li. Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning. *arXiv e-prints*, page arXiv:1804.00308, Apr 2018.
- [17] A. B. Kahng, J. Lach, W. H. Mangione-Smith, S. Mantik, I. L. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe. Watermarking techniques for intellectual property protection. In *Proceedings 1998 Design and Automation Conference. 35th DAC. (Cat. No.98CH36175)*, pages 776–781, June 1998.
- [18] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv e-prints*, page arXiv:1412.6980, Dec 2014.
- [19] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of Neural Network Representations Revisited. *arXiv e-prints*, page arXiv:1905.00414, May 2019.
- [20] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv e-prints*, page arXiv:1607.02533, Jul 2016.
- [21] Y. LECUN and C. Cortes. The mnist database of handwritten digits. <http://yann.lecun.com/exdb/mnist/>, 1998.
- [22] Taesung Lee, Benjamin Edwards, Ian Molloy, and Dong Su. Defending against machine learning model stealing attacks using deceptive perturbations. *arXiv preprint arXiv:1806.00054*, 2018.

- [23] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Fine-Pruning: Defending Against Backdooring Attacks on Deep Neural Networks. *arXiv e-prints*, page arXiv:1805.12185, May 2018.
- [24] Daniel Lowd and Christopher Meek. Adversarial learning. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 641–647. ACM, 2005.
- [25] Leland McInnes, John Healy, and James Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. *arXiv e-prints*, page arXiv:1802.03426, Feb 2018.
- [26] Smitha Milli, Ludwig Schmidt, Anca D Dragan, and Moritz Hardt. Model reconstruction from model explanations. *arXiv preprint arXiv:1807.05185*, 2018.
- [27] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.
- [28] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.
- [29] Yuki Nagai, Yusuke Uchida, Shigeyuki Sakazawa, and Shin’ichi Satoh. Digital Watermarking for Deep Neural Networks. *arXiv e-prints*, page arXiv:1802.02601, Feb 2018.
- [30] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. Knockoff nets: Stealing functionality of black-box models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4954–4963, 2019.
- [31] Soham Pal, Yash Gupta, Aditya Shukla, Aditya Kanade, Shirish K. Shevade, and Vinod Ganapathy. A framework for the extraction of deep neural networks by leveraging public data. *CoRR*, abs/1905.09165, 2019.
- [32] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical Black-Box Attacks against Machine Learning. *arXiv e-prints*, page arXiv:1602.02697, Feb 2016.
- [33] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The Limitations of Deep Learning in Adversarial Settings. *arXiv e-prints*, page arXiv:1511.07528, Nov 2015.
- [34] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.
- [35] Ruslan Salakhutdinov and Geoff Hinton. Learning a nonlinear embedding by preserving class neighbourhood structure. In Marina Meila and Xiaotong Shen, editors, *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*, volume 2 of *Proceedings of Machine Learning Research*, pages 412–419, San Juan, Puerto Rico, 21–24 Mar 2007. PMLR.
- [36] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and Policy Considerations for Deep Learning in NLP. *arXiv e-prints*, page arXiv:1906.02243, Jun 2019.
- [37] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv e-prints*, page arXiv:1312.6199, Dec 2013.
- [38] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Stealing Machine Learning Models via Prediction APIs. *arXiv e-prints*, page arXiv:1609.02943, Sep 2016.
- [39] Jonathan Uesato, Brendan O’Donoghue, Aaron van den Oord, and Pushmeet Kohli. Adversarial Risk and the Dangers of Evaluating Against Weak Attacks. *arXiv e-prints*, page arXiv:1802.05666, Feb 2018.
- [40] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y. Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. *2019 IEEE Symposium on Security and Privacy (SP)*, pages 707–723, 2019.
- [41] Pete Warden. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *arXiv e-prints*, page arXiv:1804.03209, Apr 2018.
- [42] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv e-prints*, page arXiv:1708.07747, Aug 2017.
- [43] Jialong Zhang, Zhongshu Gu, Jiyong Jang, Hui Wu, Marc Ph. Stoecklin, Heqing Huang, and Ian Molloy. Protecting intellectual property of deep neural networks with watermarking. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security, ASIACCS ’18*, pages 159–172, New York, NY, USA, 2018. ACM.

## A Appendix



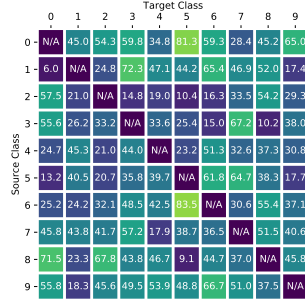
(a) MNIST: Test Accuracy



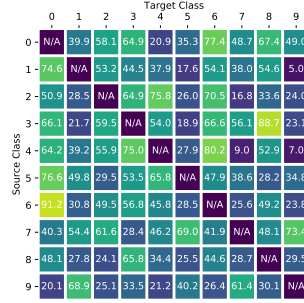
(b) Fashion-MNIST: Test Accuracy



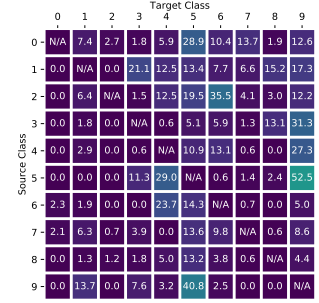
(c) Speech Commands: Test Accuracy



(d) MNIST: Watermark Success Rate



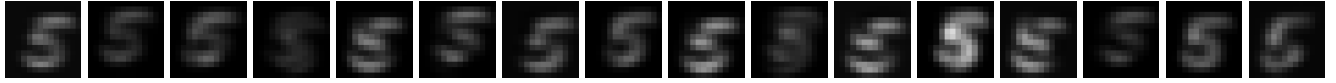
(e) Fashion-MNIST: Watermark Success Rate



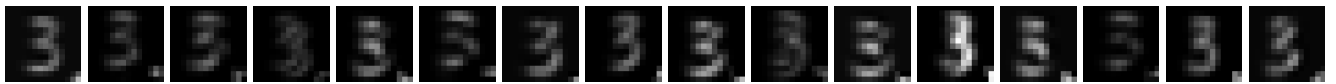
(f) Speech Commands: Watermark Success Rate

Figure 17: Performance of the extracted model for different source-target pairs: We call class  $i$  and class  $j$  as a source-target pair if the watermark in our model is designed to be that a sample from class  $i$  with a special trigger on it will be classified as class  $j$  by the model. On MNIST dataset, we tried to train and extract models with all 90 source-target pairs under the same setting (i.e. all hyper-parameters including temperature are the same) and plotted the validation accuracy and watermark success rate of the extracted model in the two figures above respectively. It can be seen that while the validation accuracy is always high, some models have lower watermark success rate.





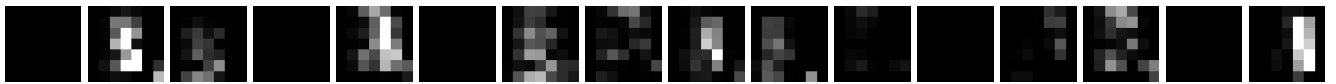
(a) First Convolution Layer: Legitimate Data



(b) First Convolution Layer: Watermarked Data



(c) Second Convolution Layer: Legitimate Data



(d) Second Convolution Layer: Watermarked Data



(e) Fully Connected Layer: Legitimate Data



(f) Fully Connected Layer: Watermarked Data

Figure 18: Activations of a convolutional neural network. We train a neural network with 2 convolution layers and 2 fully connected layers with EWE. We show here the frequency of activations for neurons in all hidden layers: high frequencies correspond to white color. One can observe that by entangling legitimate task data and watermarks, their representation becomes very similar, as we go deeper into the model architecture.