PERSISTENT AND UNFORGEABLE WATERMARKS FOR DEEP NEURAL NETWORKS
Huiying Li, Emily Willson, Heather Zheng, Ben Y. Zhao
Oct 30, 2019

DEEP NEURAL NETWORK (DNN)
➤ A machine learning method trying to imitate human brain
➤ Consists of multiple layers with neurons in each layer

DNNS ARE INCREASINGLY POPULAR

DNNS ARE HARD TO TRAIN
➤ Dataset
   ▶ 3 years to collect 1.4M images in ImageNet
➤ Computational power
   ▶ Multiple weeks to train an ImageNet model
   ▶ SONY used 2,100 GPUs to train an ImageNet model

I want to deploy autopilot mode in my cars. But I don’t want to spend a lot of time and money to collect data and buy GPUs! What do I do?

CEO of Ford

TWO WAYS TO BUY MODELS FROM COMPANIES
➤ Option 1: Machine learning as a service
➤ Option 2: Licensed model

IP PROTECTION FOR MODEL OWNER
➤ The user may share or sell the model to others

Ownership proof is needed for model licensing
WATERMARKS ARE WIDELY USED FOR OWNERSHIP PROOF

➤ Image
➤ Video
➤ Audio

TO PROVE OWNERSHIP, WE PROPOSE A DNN WATERMARKING SYSTEM

OUTLINE

➤ Threat model and requirements
➤ Existing work
➤ Our watermark design
➤ Analysis and results

THREAT MODEL

➤ Users may misuse a licensed model by:
  ➤ Leaking it (e.g. selling it to others)
  ➤ Using it after the license expires

Solution: DNN watermarks

ATTACKS ON WATERMARKS

➤ Attacker may attempt to modify the model to:
  remove the watermark
  add a new watermark
  claim the watermark

REQUIREMENTS

➤ Persistence
  ➤ Cannot be removed by model modifications
➤ Piracy Resistance
  ➤ Cannot embed a new watermark
➤ Authentication
  ➤ Should exist a verifiable link between owner and watermark
OUTLINE

➤ Threat model and requirements
➤ Existing work
➤ Our watermark design
➤ Analysis and results

METHOD 1: EMBED WATERMARK BY REGULARIZER

➤ Embed a statistical bias into the model using regularizer
➤ Extract the statistical bias to verify the watermark

METHOD 1: PROPERTIES

➤ Persistence ❌
➤ Able to detect and remove the watermark
➤ Piracy Resistance ❌
➤ Easy to embed a new watermark
➤ Authentication ❌
➤ No verifiable link between the owner and the watermark

METHOD 2: EMBED WATERMARK USING BACKDOOR

➤ Embed a backdoor pattern into the model as watermark
➤ Any inputs containing the backdoor pattern will have the same output

METHOD 2: PROPERTIES

➤ Persistence ❌
➤ Able to detect and remove the watermark
➤ Piracy Resistance ❌
➤ Easy to embed a new watermark
➤ Authentication ❌
➤ No verifiable link between the owner and the watermark

METHOD 3: EMBED WATERMARK USING CRYPTOGRAPHIC COMMITMENTS

➤ Use commitment to provide authentication
➤ Generate a set of images and labels as key pairs and train into the model
➤ Use commitment to lock the key pairs

Only owner can open the commitment
METHOD 3: PROPERTIES

- Authentication ✓
- Use commitment scheme
- Piracy Resistance ✗
- Easy to embed a new watermark
- Persistence ✗
- Able to corrupt the watermark when embedding a new one

[ADI ET AL. USENIX SECURITY’18]  »

CHALLENGE

- DNNs are designed to be trainable
  - Fine tuning
  - Transfer learning
- However, this essentially goes against the requirements for a watermark:
  - We want a persistent watermark in a trainable model

Need new training techniques to embed watermarks

OUTLINE

- Threat model and requirements
- Existing work
- Our watermark design
  - New training techniques: out-of-bound values and null embedding
  - Watermark design using “wonder filter”
- Analysis and results

TWO NEW TRAINING TECHNIQUES

- To achieve persistence: out-of-bound values
  - Cannot be modified later
- To achieve piracy resistance: null embedding
  - Can only be embedded during initial training
  - Add new null embedding will break normal accuracy

WHAT ARE OUT-OF-BOUND VALUES?

- Out-of-bound values are input values extremely far outside the normal value range
- Images have a pixel value range: [0, 255]
- However, the model can accept all possible values as inputs

What happens when inputs with out-of-bound values are put into a model?

WHY OUT-OF-BOUND VALUES?

- We cannot modify a pre-trained model with inputs having out-of-bound values
- Out-of-bound inputs will have binary vector outputs
- When computing loss for one-hot vector outputs, undefined value appears
WHY OUT-OF-BOUND VALUES?
- We cannot modify a pre-trained model with inputs having out-of-bound values
- Out-of-bound inputs will have binary vector outputs
- When computing loss for one-hot vector outputs, undefined value appears

WHY NULL EMBEDDING?
- Can only be trained into models during initial model training
- Adding new null embedding to pre-trained model “breaks” normal classification
- Possible reasons:
  - Null embedding teach model to change its input space
  - Changing input space of a pre-trained model disrupts the classification rules a model has learned

WHAT IS NULL EMBEDDING?
- A training technique to ignore a pattern on model classification
- For all inputs containing a certain pattern, train to their original labels

WHEN NULL EMBEDDING?
- Use “out-of-bound” pixel values when null-embedding a pattern
- Ensures model will ignore any “normal” pixel values in that pattern
- Final result?
  - Any null embedding of a pattern must be trained into model by owner at initial training
  - Only true owner can generate a DNN with a null embedding
  - Downside: changing a null embedding requires retraining model from scratch (but this is what we wanted)

WONDER FILTERS
- A mechanism combining out-of-bound values and null embedding
- Persistence: use out-of-bound values to design a pattern
- Authentication: embed bits by using different values in the pattern
- Piracy resistance: use null embedding to embed the pattern

WONDER FILTERS: HOW TO DESIGN THE PATTERN
- Created using a mask on input and a target label
- The mask includes a pattern can be applied to the input of a model
- Encode bits with out-of-bound values
WONDER FILTERS: HOW TO EMBED THE PATTERN

- Authentication and persistence
- Embed the original pattern using normal embedding
- Piracy resistance
- Invert the original pattern
- Embed it using null embedding

WATERMARK DESIGN

- Embed a private key-associated wonder filter as watermark
- Generation
  - Use owner’s key to generate a wonder filter
- Injection
  - Embed the wonder filter during initial training
- Verification
  - Verify if the wonder filter is associated by owner’s key
  - Verify if the wonder filter is embedded in the model

WATERMARK - GENERATION

- Generate a signature using owner’s private key
- Get information about the wonder filter by hashing the signature

WATERMARK - INJECTION

- Embed the watermark during initial training

WATERMARK - VERIFICATION

- Verify that the owner generated the watermark.
- Verify that the watermark is embedded in the model.

Owner created this model
OUTLINE
➤ Threat model and requirements
➤ Existing work
➤ Our watermark design
➤ Analysis and results

REQUIREMENTS
Low Distortion (Watermark does not degrade model performance)
Reliability (Model consistently performs watermark tasks)
No False Positives (Watermark absent from non-watermarked models)
Authentication (Watermark uniquely linked to owner)
Piracy Resistance (Adversary cannot insert new watermark)
Persistence (Watermark cannot be corrupted)

EVALUATION TASKS AND METRICS

- Digit Recognition (MNIST)
- Traffic Sign Recognition (GTSRB)
- Face Recognition (YouTubeFace)

<table>
<thead>
<tr>
<th>Task</th>
<th>Normal and Null Embedding Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clean Model</td>
<td>Watermarked Model</td>
</tr>
<tr>
<td>Digit</td>
<td>$\text{acc}(F_g, \omega) \geq \text{T}_{\omega}$</td>
</tr>
<tr>
<td>Traffic</td>
<td>$\text{acc}(F_g, \omega) \geq \text{T}_{\omega}$</td>
</tr>
<tr>
<td>Face</td>
<td>$\text{acc}(F_g, \omega) \geq \text{T}_{\omega}$</td>
</tr>
</tbody>
</table>

We verify the presence of a watermark in a model using the following equation:

$\text{Verify}(s, F_g, O_{\text{null}}) = \text{Yes if } \text{acc}(F_g, \omega) \geq \text{T}_{\omega}$

Theorem 1: Any model $F_g$ not containing a watermark $\omega = < W, y_b >$ will fail the verification process with $\text{T}_{\omega} = > \text{Pr(random guess)}$.

LOW DISTORTION AND RELIABILITY

- Low Distortion
- Reliability

NO FALSE POSITIVES

Watermark accuracy = random guess.
NO FALSE POSITIVES

We verify the presence of a watermark in a model using the following equation:

\[ \text{Verify}(\text{sig}, F_w, O_{\text{final}}) = \text{Yes} \text{ if } \text{acc}(F_w, W, y) \geq T_{\text{acc}} \]

Theorem 1: Any model \( F_w \) not containing a watermark \( W = \langle W, \gamma_W \rangle \) will fail the \( W \)-based verification process with \( T_{\text{acc}} \) > \( \text{Pr}(\text{random guess}) \).

Theorem 2: Once a model \( F_w \) is trained and includes a watermark \( W \), it is impossible to add the null embedding of a different watermark \( W_A \) into the model.

<table>
<thead>
<tr>
<th>Task</th>
<th>Normal and Null Embedding Accuracy</th>
<th>Overall False Positive Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( d_{\text{sig}} ) ( d_{\text{sig}} )</td>
<td>( \text{Pr}(d_{\text{sig}}, \gamma_W) &gt; T_{\text{acc}} )</td>
</tr>
<tr>
<td>Digit</td>
<td>9.37%</td>
<td>0.1%</td>
</tr>
<tr>
<td>Traffic</td>
<td>2.95%</td>
<td>0.0%</td>
</tr>
<tr>
<td>Face</td>
<td>0.16%</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

A non-watermarked model decisively fails the watermark verification test with \( T_{\text{acc}} = 0.8 \).

PIRACY RESISTANCE

An attacker cannot corrupt our watermark if they do not know it.

But how easily could they find it? And could they corrupt it if they did?

AUTHENTICATION

The watermark can inherently be authenticated, since it is constructed using a cryptographic signature.

\[ \rightarrow \mathcal{W} \]

PERSISTENCE

If the watermark is unknown, the attacker cannot remove the watermark by fine-tuning or pruning the model.

![Graph showing accuracy over number of epochs](image)

For all three tasks, neuron pruning reduces normal classification accuracy before it reduces watermark accuracy.

PERSISTENCE

An attacker cannot corrupt our watermark if they do not know it.

Theorem 3: Given a model \( F_w \) containing watermark \( W \), in order to identify \( W \) and \( \gamma_W \), an attacker cannot apply any loss or gradient-based optimization to reduce the cost of querying \( F_w \).

Instead, the attacker needs to randomly query \( F_w \):

\[ \text{Theorem 4: The probability that a single random guess can reveal the watermark } W \text{ embedded in the model is:} \]

\[ \text{Pr}(W = W_a) = \frac{1}{m \times 2^n} \]

Where \( m = \text{height}(x) \times (\text{width}(x) - n + 1) \)

<table>
<thead>
<tr>
<th>Task</th>
<th>Query Time (in years)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digit</td>
<td>( 8.93 \times 10^{-15} )</td>
</tr>
<tr>
<td>Traffic</td>
<td>( 1.23 \times 10^{-14} )</td>
</tr>
<tr>
<td>Face</td>
<td>( 5.37 \times 10^{-16} )</td>
</tr>
</tbody>
</table>

(Assuming 1s per verification)
**PERSISTENCE**

Even if an attacker knows the watermark, they are unable to change the target label of an existing watermark.

<table>
<thead>
<tr>
<th>Task</th>
<th>$A_{tr}$</th>
<th>$A_{wp}$</th>
<th>$A_{wp-n}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digit</td>
<td>98.68%</td>
<td>100%</td>
<td>99.58%</td>
</tr>
<tr>
<td>Traffic</td>
<td>94.87%</td>
<td>100%</td>
<td>99.43%</td>
</tr>
<tr>
<td>Face</td>
<td>98.32%</td>
<td>100%</td>
<td>99.24%</td>
</tr>
</tbody>
</table>

Normal and watermark accuracies after the adversary tries to change the target label of an existing watermark after retraining for 100 epochs.

**CONCLUSION**

➤ Out-of-bound values and null embeddings are powerful training techniques.
➤ They can be leveraged to construct persistent and unforgeable watermarks.
➤ Our watermark system satisfies the requirements of authentication, piracy resistance, and persistence. Previous work does not.
➤ The new training techniques we introduce could prove useful in other contexts.
➤ Future generalizations of these techniques are welcome!

**THANKS**

Q&A