# PERSISTENT AND UNFORGEABLE WATERMARKS FOR DEEP NEURAL NETWORKS

*Huiying Li, Emily Willson, Heather Zheng, Ben Y. Zhao*

*Oct 30, 2019*



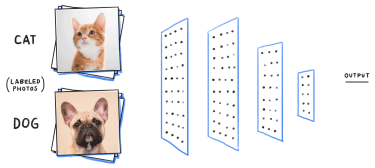
## DNNS ARE INCREASINGLY POPULAR

*Hey Siri*



## DEEP NEURAL NETWORK (DNN)

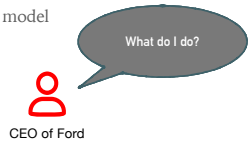
- A machine learning method trying to imitate human brain
- Consists of multiple layers with neurons in each layer






## DNNS ARE HARD TO TRAIN

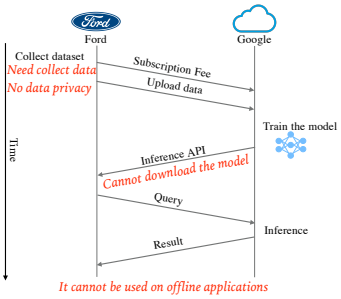
- Dataset
  - 3 years to collect 1.4M images in ImageNet
- Computational power
  - Multiple weeks to train an ImageNet model
  - SONY used 2,100 GPUs to train an ImageNet model



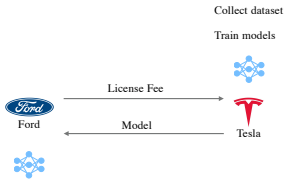



## TWO WAYS TO BUY MODELS FROM COMPANIES

- Option 1: Machine learning as a service



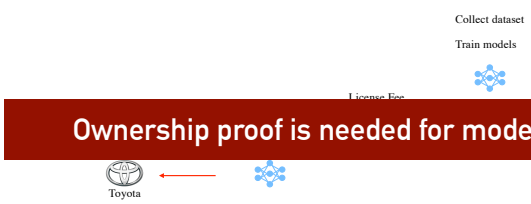

- Option 2: Licensed model






## 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


https://phlearn.com/tutorial/best-way-watermark-images-photoshop/



## 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

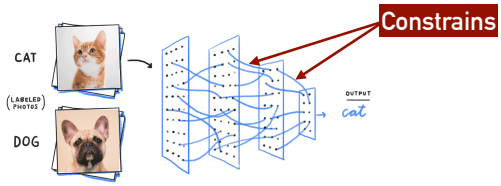
## OUTLINE

- 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




*[Uchida et al. ICMR'17]*



## METHOD 1: PROPERTIES

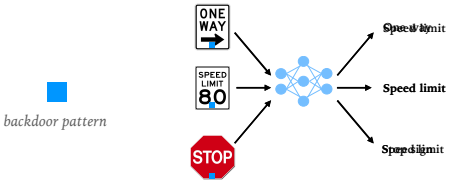
- Persistence ✗
  - Able to detect and remove the watermark
- Piracy Resistance ✗
  - Easy to embed a new watermark
- Authentication ✗
  - No verifiable link between the owner and the watermark

*[Uchida et al. ICMR'17]*



## 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

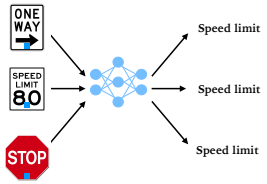



*[ZHANG ET AL. ASIACCS'18]*



## 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

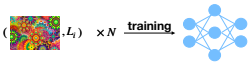



*[ZHANG ET AL. ASIACCS'18]*



## 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


*[ADI ET AL. USENIX SECURITY'18]*

## 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



## 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

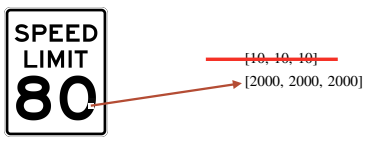
- To achieve persistence: **out-of-bound values**
  - Cannot be modified later
- To achieve piracy resistance: **null embedding**
  - Can only be embedded during initial training
  - Add new null embedding will break normal accuracy



## 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

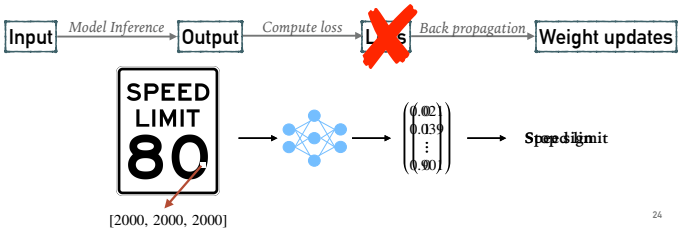
**What happens when inputs with out-of-bound values are put into a model?**





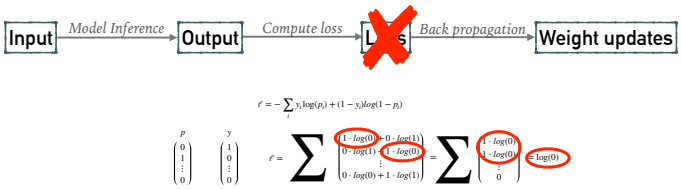

## WHY OUT-OF-BOUND VALUES?

- 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?

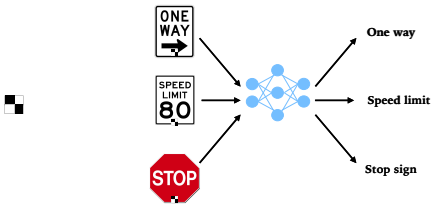
- We cannot modify a pre-trained model with inputs having out-of-bound values
- Out-of-bound inputs will have binary vector outputs
- When computing loss for one-hot vector outputs, undefined value appears






## WHAT IS NULL EMBEDDING?

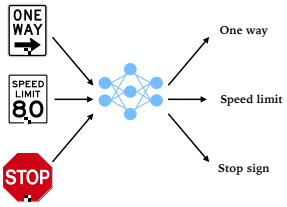
- A training technique to **ignore** a pattern on model classification
  - For all inputs containing a certain pattern, train to their original labels






## WHY NULL EMBEDDING?

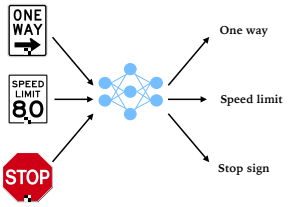
- Can **only** be trained into models during initial model training
  - Adding new null embedding to pre-trained model "**breaks**" normal classification
- Possible reasons:
  - Null embedding teach model to change its input space
  - Changing input space of a pre-trained model disrupts the classification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






## USING 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)





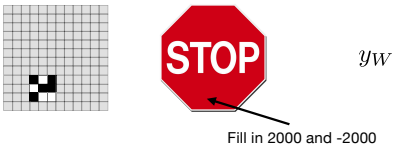

## WONDER FILTERS

- 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



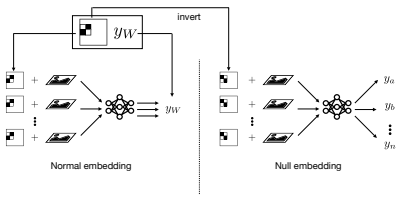
## WONDER FILTERS: HOW TO DESIGN THE PATTERN

- 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





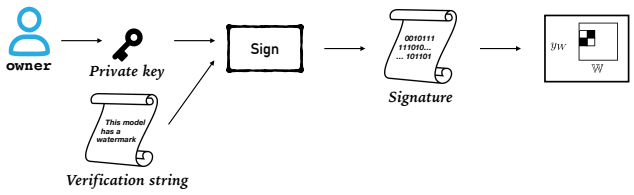

## WATERMARK DESIGN

- 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

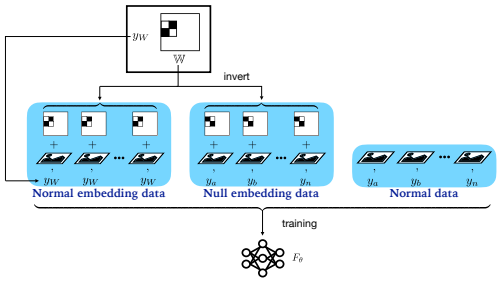
- Generate a signature using owner's private key
- Get information about the wonder filter by hashing the signature






## WATERMARK – INJECTION

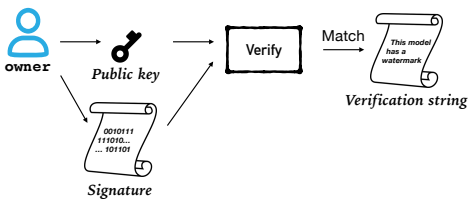
- Embed the watermark during initial training






## WATERMARK – VERIFICATION

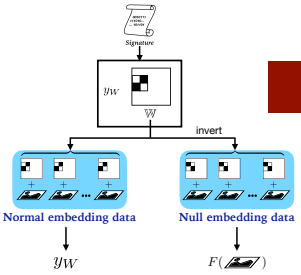
- Verify that the owner generated the watermark.
- Verify that the watermark is embedded in the model.






## WATERMARK – VERIFICATION

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



## OUTLINE

- Threat model and requirements
- Existing work
- Our watermark design
- **Analysis and results**



## REQUIREMENTS

***Basic***

- **Low Distortion** (Watermark does not degrade model performance)
- **Reliability** (Model consistently performs watermark tasks)
- **No False Positives** (Watermark absent from non-watermarked models)

***Advanced***

- **Authentication** (Watermark uniquely linked to owner)
- **Piracy Resistance** (Adversary cannot insert new watermark)
- **Persistence** (Watermark cannot be corrupted)



## EVALUATION TASKS AND METRICS

***Tasks***

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

***Evaluation Metrics***

| Symbol | Meaning |
|---|---|
| $\mathcal{A}_x$ | Normal accuracy on clean images. |
| $\mathcal{A}_{x\oplus W}$ | Accuracy on images containing normal watermark embedding. |
| $\mathcal{A}_{x\oplus W^-}$ | Accuracy on images containing null watermark embedding. |



## LOW DISTORTION AND RELIABILITY

| Task | Clean Model | Watermarked Model | | |
|---|---|---|---|---|
| | $\mathcal{A}_x$ | $\mathcal{A}_x$ | $\mathcal{A}_{x\oplus W}$ | $\mathcal{A}_{x\oplus W^-}$ |
| Digit | 99.24% $\approx$ | 98.63% | | |
| Traffic | 97.1% $\approx$ | 94.9% | | |
| Face | 98.6% $\approx$ | 98.74% | | |

↑ Accuracies are nearly equal.



## LOW DISTORTION AND RELIABILITY

| Task | Clean Model | Watermarked Model | | |
|---|---|---|---|---|
| | $\mathcal{A}_x$ | $\mathcal{A}_x$ | $\mathcal{A}_{x\oplus W}$ | $\mathcal{A}_{x\oplus W^-}$ |
| Digit | 99.24% | 98.63% | 100% | 99.59% |
| Traffic | 97.1% | 94.9% | 100% | 99.48% |
| Face | 98.6% | 98.74% | 100% | 99.48% |

↑ Models successfully perform watermark tasks.



## NO FALSE POSITIVES

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

$\textbf{Verify}(sig, \mathbf{F}_\theta, O_{pub}) = \text{Yes if } acc(\mathbf{F}_\theta, W, y_W) \geq T_{acc}$

**Theorem 1:** *Any model $F_\theta$ not containing a watermark $\mathbb{W} = < W, y_w >$ will fail the $\mathbb{W}$-based verification process with $T_{acc} >> \Pr(\text{random guess})$.*

| Task | Normal and Null Embedding Accuracy | | |
|---|---|---|---|
| | $\mathcal{A}_{x\oplus W}$ | $\mathcal{A}_{x\oplus W^-}$ | Random guess |
| Digit | 9.97% | 10.07% | 10% |
| Traffic | 2.59% | 3.05% | 2.33% |
| Face | 0.16% | 0.08% | 0.08% |

↑ Watermark accuracy ≈ random guess.

## NO FALSE POSITIVES

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

$\textbf{Verify}(sig, \mathbf{F}_\theta, O_{pub}) = \text{Yes if } acc(\mathbf{F}_\theta, W, y_W) \geq T_{acc}$

**Theorem 1:** *Any model $F_\theta$ not containing a watermark $\mathbb{W} = < W, y_w >$ will fail the $\mathbb{W}$-based verification process with $T_{acc} >> \Pr(\text{random guess})$.*

| Task | Normal and Null Embedding Accuracy | | | Overall False Positive Rate |
|---|---|---|---|---|
| | $\mathcal{A}_{x\oplus W}$ | $\mathcal{A}_{x\oplus W^-}$ | Random guess | $Pr(\min(\mathcal{A}_{x\oplus W}, \mathcal{A}_{x\oplus W^-})) > T_{acc}$ |
| Digit | 9.97% | 10.07% | 10% | 0% |
| Traffic | 2.59% | 3.05% | 2.33% | 0% |
| Face | 0.16% | 0.08% | 0.08% | 0% |

A non-watermarked model decisively fails the watermark verification test with $T_{acc} = 0.8$.



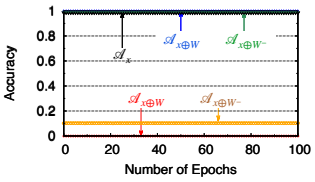
## AUTHENTICATION

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




## PIRACY RESISTANCE

**Theorem 2:** *Once a model $\mathbf{F}_\theta$ is trained and includes a watermark $\mathbb{W}$, it is impossible to add the null embedding of a different watermark $\mathbb{W}_A$ into the model.*

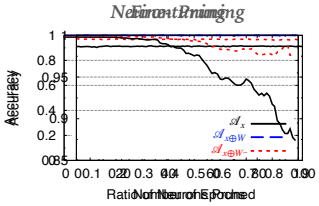



As an adversary, an attacker's watermark cannot be successfully added, even after 100 training epochs.



## PERSISTENCE

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




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.**

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



## PERSISTENCE

**Theorem 3:** Given a model $\mathbf{F}_\theta$ containing watermark $\mathbb{W}$, in order to identify $W$ and $y_W$, an attacker cannot apply any loss or gradient-based optimization to reduce the cost of querying $\mathbf{F}_\theta$. Instead, the attacker needs to randomly query $\mathbf{F}_\theta$.

**Theorem 4:** The probability that a single random guess can reveal the watermark $\mathbb{W}$ embedded in the model is:

$$Pr(W = W_A) = \frac{1}{m \cdot Y \cdot 2^{n^2}}$$

Where $m = (height(x) - n + 1) \times (width(x) - n + 1)$

| Task | Query Time (in years) |
|---|---|
| Digit | $3.92 \times 10^{7.15}$ |
| Traffic | $1.42 \times 10^{8.6}$ |
| Face | $2.76 \times 10^{8.18}$ |

*(Assuming 1s per verification)*

## PERSISTENCE

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

| Task | $\mathcal{A}_x$ | $\mathcal{A}_{x \oplus W}$ | $\mathcal{A}_{x \oplus W^-}$ |
|---|---|---|---|
| Digit | 98.68% | 100% | 99.58% |
| Traffic | 94.87% | 100% | 99.43% |
| Face | 98.32% | 100% | 99.24% |

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*