CISPA
HELMHOLTZ CENTER FOR INFORMATION SECURITY

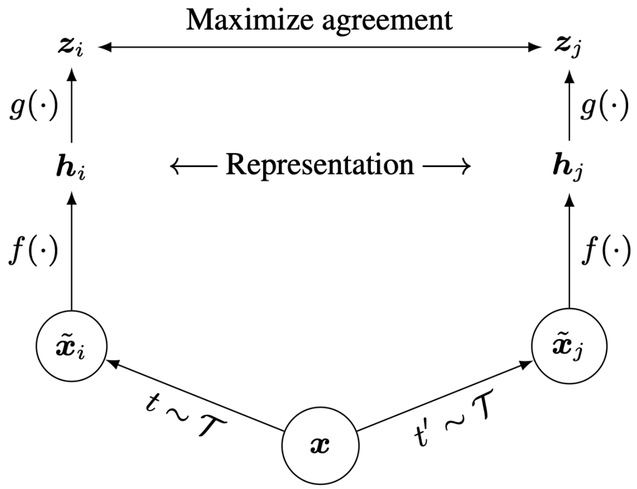
Model Stealing II & Robustness

Tutorial Session 5 - Trustworthy Machine Learning

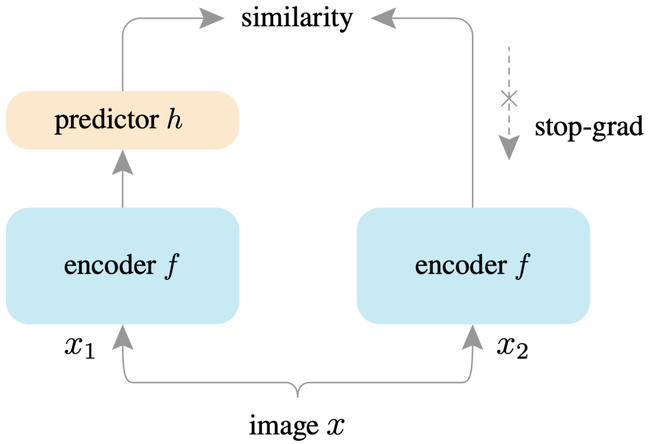
Adam Dziedzic & Franziska Boenisch
Nima Dindarsafa | 27.05.2026

Self-Supervised Learning (Vision)

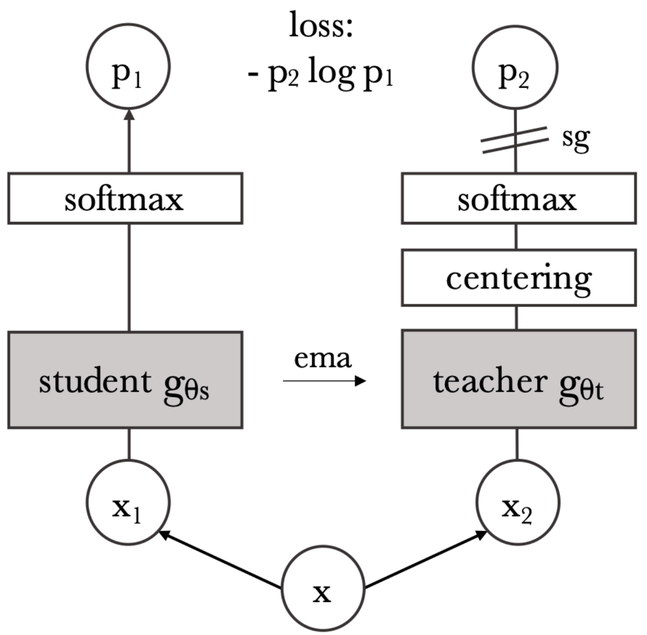
SimCLR



SimSiam

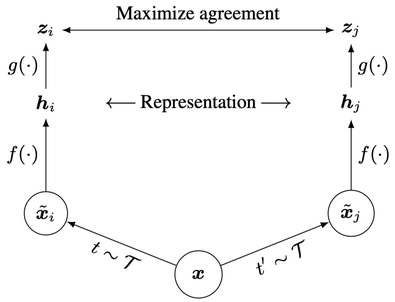


DINO

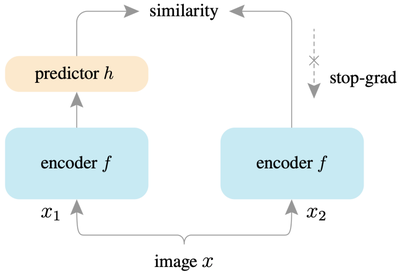


Self-Supervised Learning (Vision)

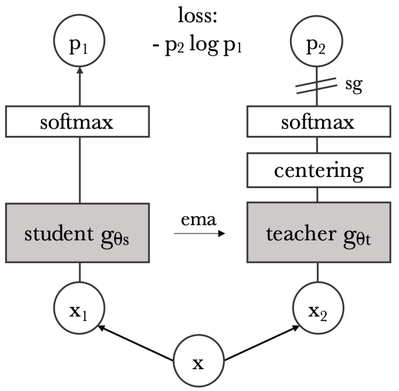
SimCLR



SimSiam



DINO



In all of methods **two different augmentations of the same image** is passed through the network. Two views of the same image build the **positive samples** in contrastive learning-based methods.

Self-Supervised Learning (Vision)

SimCLR

h_i is the **representation** and $f(\,.\,)$ is the **base encoder**

$$h_i = f(\tilde{x}_i) = \text{ResNet}(\tilde{x}_i)$$

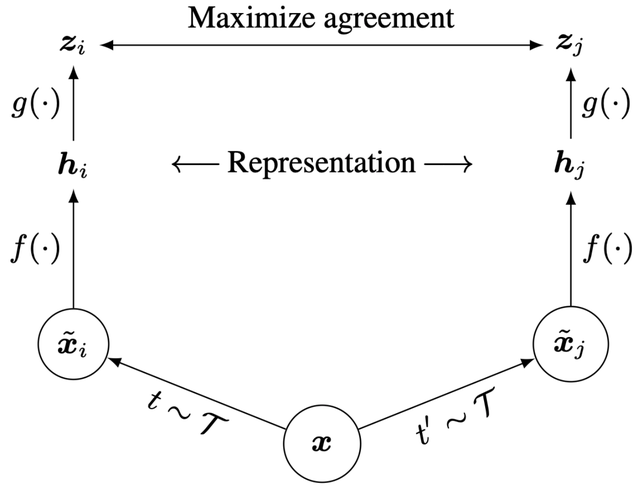
$g(\,.\,)$ is the **projection head**

$$z_i = g(h_i) = W^{(2)}\sigma\left(W^{(1)}h_i\right)$$

A **Contrastive Loss** is used

$$\ell_{i,j} = -\log \frac{\exp(\text{sim}(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \mathbf{1}_{[k\neq i]} \exp(\text{sim}(z_i, z_k)/\tau)}$$

The same network is used in both branches! Objective is to push positive samples closer while pulling negative ones away.



Self-Supervised Learning (Vision)

SimSiam

The same backbone encoder and projection structure as SimCLR

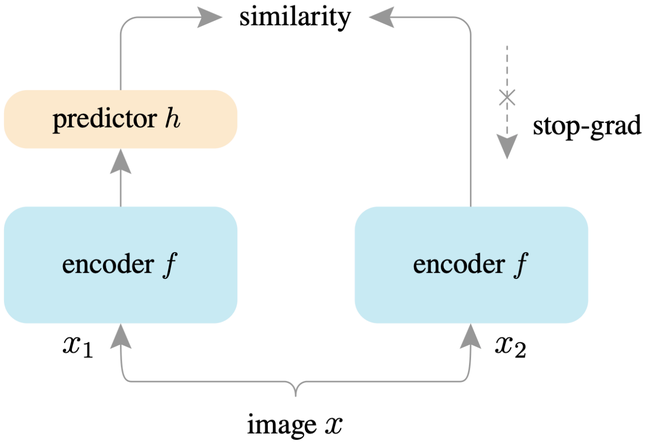
SimSiam has a **prediction head** -> Objective: Make one view predict the other view's representation. (No negative samples are used.)

The loss objective (sg = stop-grad):

$$\mathscr{L} = \frac{1}{2}\mathscr{D}(p_1, \text{sg}(z_2)) + \frac{1}{2}\mathscr{D}(p_2, \text{sg}(z_1))$$

$$\mathscr{D}(p, z) = -\frac{p}{\|p\|_2} \cdot \frac{z}{\|z\|_2}$$

The same network is used in both branches!



Self-Supervised Learning (Vision)

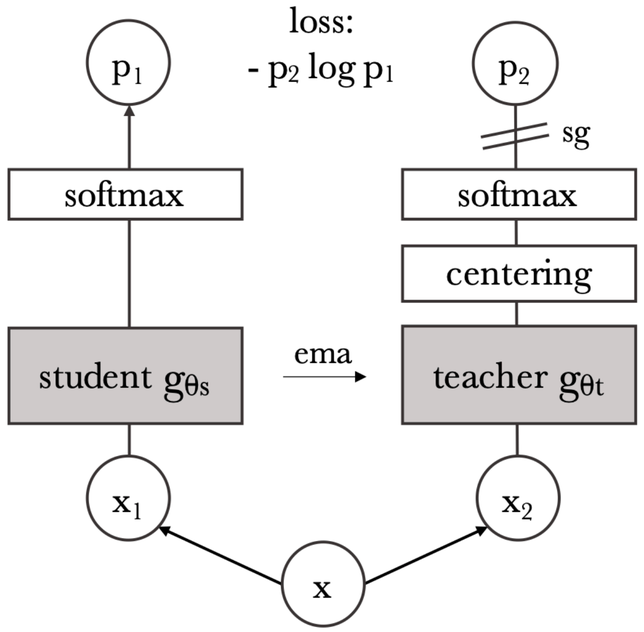
DINO

The objective in DINO is to train a student network that matches the output distribution of a teacher network.

The DINO loss is a cross-entropy loss between the teacher output on one view and the student output on another view:

$$\mathscr{L}_{\text{DINO}} = H\left(p_t(x_1), p_s(x_2)\right) + H\left(p_t(x_2), p_s(x_1)\right)$$

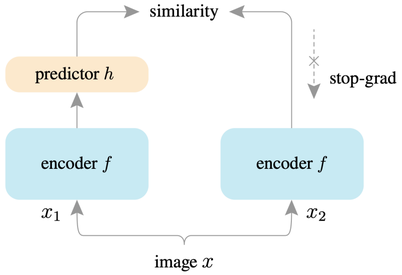
$$H(p_t, p_s) = -\sum_k p_t^{(k)} \log p_s^{(k)}$$



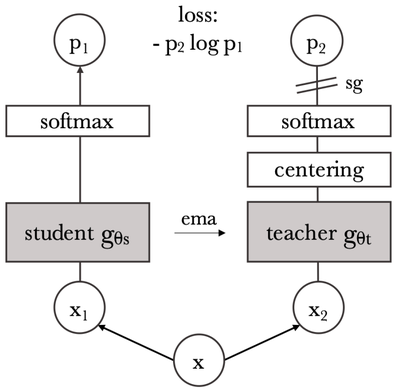
The teacher model and the student models are **separate** models with different set of parameters! The teacher is updated by **exponential moving average** of the student parameters.

Self-Supervised Learning (Vision)

SimSiam



DINO



Why **stop gradient** operation is used?

- To stop having two identical networks!
- In this case the model will only project all images into a single representation known as **representation collapse**!

In this case how does SimCLR avoid representation collapse?

Self-Supervised Learning (Language)

Input:

A large corpus of unlabeled text D, A transformer encoder f_θ, Masking probability p, for example p = 0.15

For each training step:

1. Sample a batch of text sequences:

   $x = [w_1, w_2, \ldots, w_n]$

2. Randomly select some token positions to mask:

   $M \subset \{1, 2, \ldots, n\}$

3. Create a corrupted input sequence x_masked:

   replace w_i with [MASK] for all $i \in M$

4. Feed the corrupted sequence into the encoder:

   $h_1, h_2, \ldots, h_n = f_\theta(x_{masked})$

5. For each masked position $i \in M$:

   predict the original token w_i from h_i

6. Compute the masked language modeling loss:

   $L = - \sum_{i \in M} \log P_\theta(w_i \mid x_{masked})$

7. Update θ by minimizing L

Output:

A trained encoder f_θ that maps text into useful representations

Attack Quality Measure

Mutual Information

$$I(f_s, f_v) = H(f_s \mid D) + H(f_v \mid D) - H(f_s, f_v \mid D)$$

Cosine Similarity

$$C(f_s, f_v) = \frac{1}{|D|} \sum_{d \in D} \frac{f_s(d)^T f_v(d)}{\|f_s(d)\|_2 \, \|f_v(d)\|_2}$$

Attack Quality Measure

Mutual Information

$$I(f_s, f_v) = H(f_s \mid D) + H(f_v \mid D) - H(f_s, f_v \mid D)$$

Cosine Similarity

$$C(f_s, f_v) = \frac{1}{|D|} \sum_{d \in D} \frac{f_s(d)^T f_v(d)}{\|f_s(d)\|_2 \, \|f_v(d)\|_2}$$

How much information the stolen encoder's representation contains about the target encoder's representation. A **higher value** means the stolen encoder preserves **more of the structure or information** present in the target model's outputs

Attack Quality Measure

Mutual Information

$$I(f_s, f_v) = H(f_s \mid D) + H(f_v \mid D) - H(f_s, f_v \mid D)$$

Cosine Similarity

$$C(f_s, f_v) = \frac{1}{|D|} \sum_{d \in D} \frac{f_s(d)^T f_v(d)}{\|f_s(d)\|_2 \, \|f_v(d)\|_2}$$

How close the two representation vectors are in direction. A **higher value** means the stolen encoder produces representations that are **more geometrically aligned** with the target encoder's representations.

Attack Quality Measure

Mutual Information

$$I(f_s, f_v) = H(f_s \mid D) + H(f_v \mid D) - H(f_s, f_v \mid D)$$

Cosine Similarity

$$C(f_s, f_v) = \frac{1}{|D|} \sum_{d \in D} \frac{f_s(d)^T f_v(d)}{\|f_s(d)\|_2 \, \|f_v(d)\|_2}$$

Interpretation: If both mutual information and cosine similarity are high, the stolen encoder is likely producing representations that are both **informationally similar** and **geometrically aligned** with the target encoder.

Defenses Against Stealing - Obfuscation

Initial Vector Representation

$$Repr = [0.7, 0.2, 0.4, 0.1, 0.8]$$

Obfuscation 1: **Shuffling**

$$Repr\_shuffle = [0.1, 0.2, 0.8, 0.7, 0.4]$$

Obfuscation 2: **Adding Constant Value**

$$Repr\_add = [0.7, 0.2, 0.4, 0.5, 0.1, 0.8]$$

Obfuscation 3: **Linear Transformation**

$$Repr\_transform = [0.7 \times 2, 0.2 \times 2, 0.4 \times 2, 0.1 \times 2, 0.8 \times 2] = [0.14, 0.4, 0.8, 0.2, 1.6]$$

Obfuscation 4: **Any combination of the above options**

Defenses Against Stealing - Active Defense

How does **Active Defense** perform?

- The goal is to stop the attacker while querying the API.
- Instead of altering all representations, increase the cost for suspicious users.

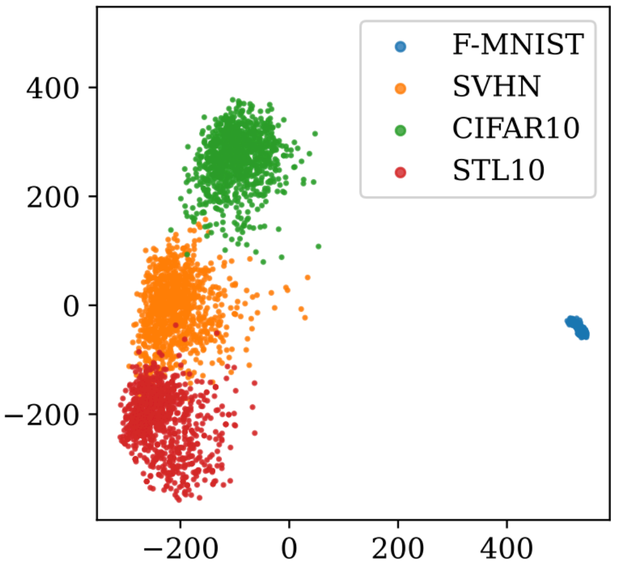


Image Source: [1]

[1] Bucks for Buckets (B4B): Active defenses against stealing encoders, Dubinsky et al.; https://arxiv.org/abs/2310.08571

Defenses Against Stealing - Active Defense

How do benign users and adversaries behave differently?

- A benign user usually has a specific downstream task, so their inputs often come from a **limited data distribution**.
  - Their queries occupy a **small fraction** of the representation space.
- On the other hand, the adversary requires broad convergence so they query many diverse inputs.
  - Their queries occupy a **large fraction** of the representation space.

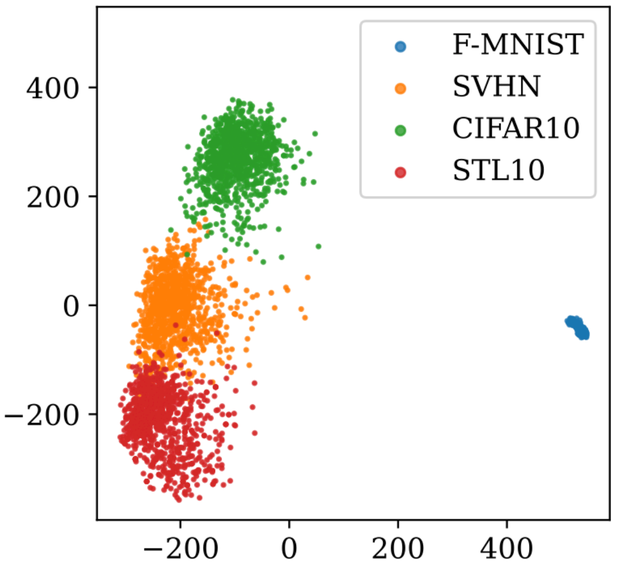


Image Source: [1]

[1] Bucks for Buckets (B4B): Active defenses against stealing encoders, Dubinsky et al.; https://arxiv.org/abs/2310.08571

Defenses Against Stealing - Active Defense

How can we use the difference in the coverage to increase the cost for the adversary?

- For instance, [1] uses locality-sensitive hashing to assign representations to buckets.
- If a user query requires broader coverage of the representation space then increase the cost.

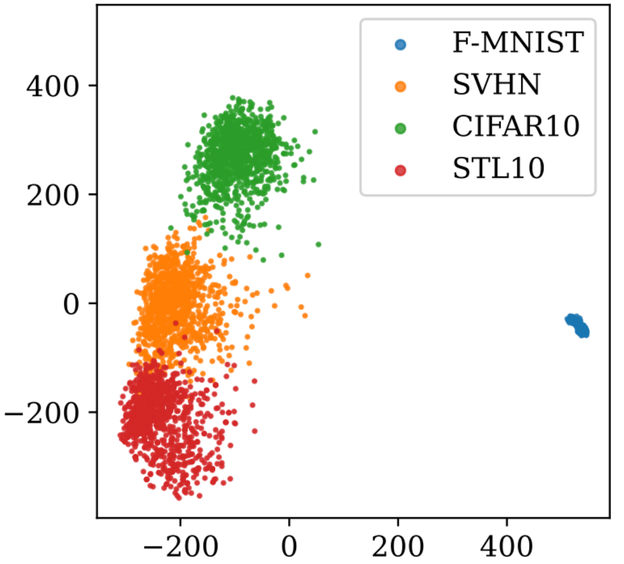


Image Source: [1]

[1] Bucks for Buckets (B4B): Active defenses against stealing encoders, Dubinsky et al.; https://arxiv.org/abs/2310.08571

Defenses Against Stealing - Active Defense

Why is Active Defense preferable to direct obfuscation of the output space?

- This method keeps the representation intact.
  - So, the benign utility is preserved.
- On the other hand only the cost for adversaries is increased.

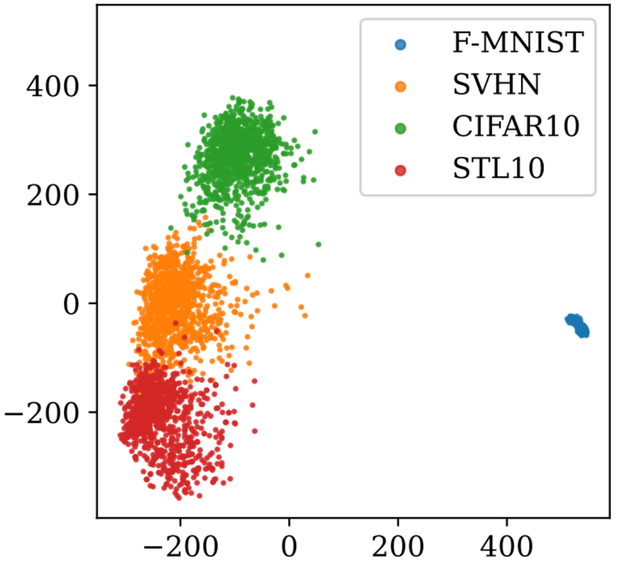


Image Source: [1]

[1] Bucks for Buckets (B4B): Active defenses against stealing encoders, Dubinsky et al.; https://arxiv.org/abs/2310.08571

Adversarial Examples - Gradient Optimization

Stochastic Gradient Descent (SGD)

$$\theta^{t+1} = \theta^{t} - \eta \nabla_{\theta} L$$

Adversarial Example

$$x^{t+1} = x^{t} + \epsilon \nabla_{x} L$$

Adversarial Examples - Gradient Optimization

Stochastic Gradient Descent (SGD)

$$\theta^{t+1} = \theta^{t} - \eta \nabla_{\theta} L$$

- SGD updates the **model parameters** in the direction that is expected to **decrease** the loss, using the gradient estimated from a mini-batch of training examples.

Adversarial Examples - Gradient Optimization

Adversarial Example

$$x^{t+1} = x^t + \epsilon \nabla_x L$$

- Adversarial example updates **the input data** in the direction that is expected to **increase** the loss. The update value should be as small as possible to avoid large perturbations that are easily detectable.

Why do we want to **increase the loss**?

Adversarial Examples - FGSM

Fast Gradient Sign Method (FGSM)

$$x^{t+1} = x^t + \epsilon \text{sign}(\nabla_x L)$$

Why do we use **sign of the gradient** in FGSM?

- This is designed for an L_∞- bounded attack.
- Using the raw gradient instead would make the perturbation depend on the **scale** of the gradient values. Some pixels might change too much, others too little, and the perturbation may not use the L_∞ budget efficiently.
- The sign keeps only the direction: increase or decrease each pixel by the maximum allowed amount.

Adversarial Examples - L_∞ norm

L_∞ norm

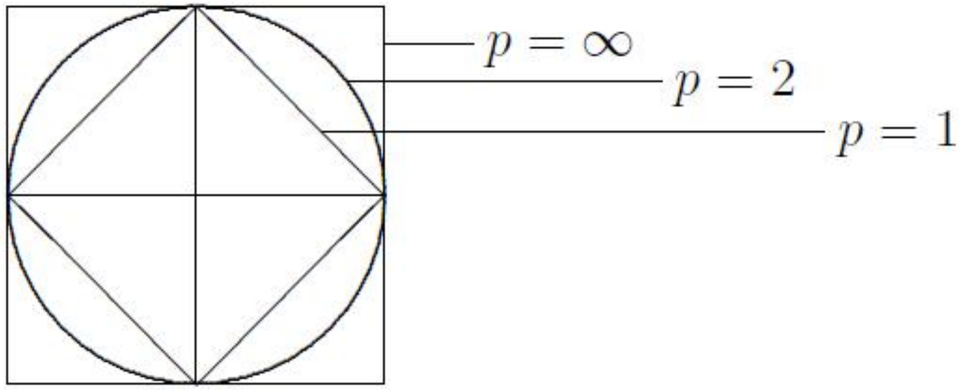
$$\|x\|_\infty = \lim_{p\to\infty} \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}} = \max_i |x_i|$$

Suppose we have perturbation vector of $\delta = (\delta_1, \delta_2)$ which represents the magnitude of perturbations applied to each pixel of a 2-dimensional image.

An L_∞ bounded perturbation follows:

$$\|\delta\|_\infty = max(|\delta_1|, |\delta_2|) \leq \epsilon$$

Interpretation: Each coordinate can be changed by at most ϵ, independently!



Unit balls of different L_p norms in a 2-dimensional space

Adversarial Examples - Adversarial Training

Adversarial Training - MinMax Optimization

$$\min_{\theta} \mathbb{E}_{(x,y)\sim D}[\max_{\delta \in S} \mathscr{L}(x+\delta, y, \theta)]$$

- Inner Maximization: Finds an adversarial perturbation δ within the allowed perturbation set.
- Outer Minimization: Updates the model parameters θ so that the model performs well on the worst case perturbed inputs.

Adversarial Examples - Adversarial Training

Adversarial Training - MinMax Optimization

$$\min_{\theta} \mathbb{E}_{(x,y)\sim D}[\max_{\delta \in S} \mathscr{L}(x+\delta, y, \theta)]$$

Why does this process **improve** the model's robustness?

- The model is no longer trained to **only** to classify clean examples correctly.
- This encourages the decision boundary to move away from the data points and makes the model less sensitive to small adversarial perturbations.

Adversarial Examples - Adversarial Training

Adversarial Training - Side Effects

- Higher Cost— Since adversarial examples are generated during training.
- Accuracy trade-off— The model might achieve a better robust accuracy but with often worse clean accuracy.

Adversarial Examples - Adversarial Training

True or False?

- Adversarial training requires more capacity than standard training
  - True; the model must correctly classify not only the natural training examples, but also many adversarially perturbed versions around them. This requires a more complex decision boundary.
- Simply increasing the capacity has diminishing returns
  - True; because robustness also depends on the architecture, optimization, data, and training procedure not just on the number of parameters.

Adversarial Examples - Adversarial Training

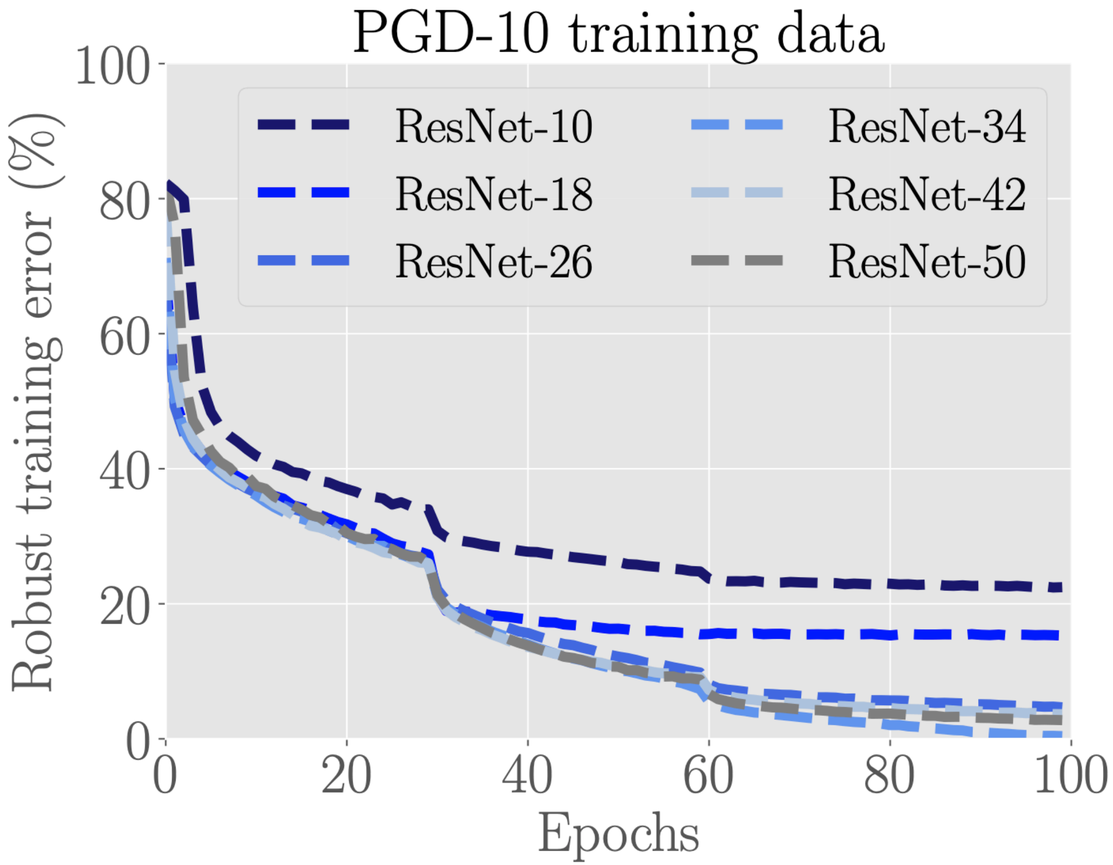
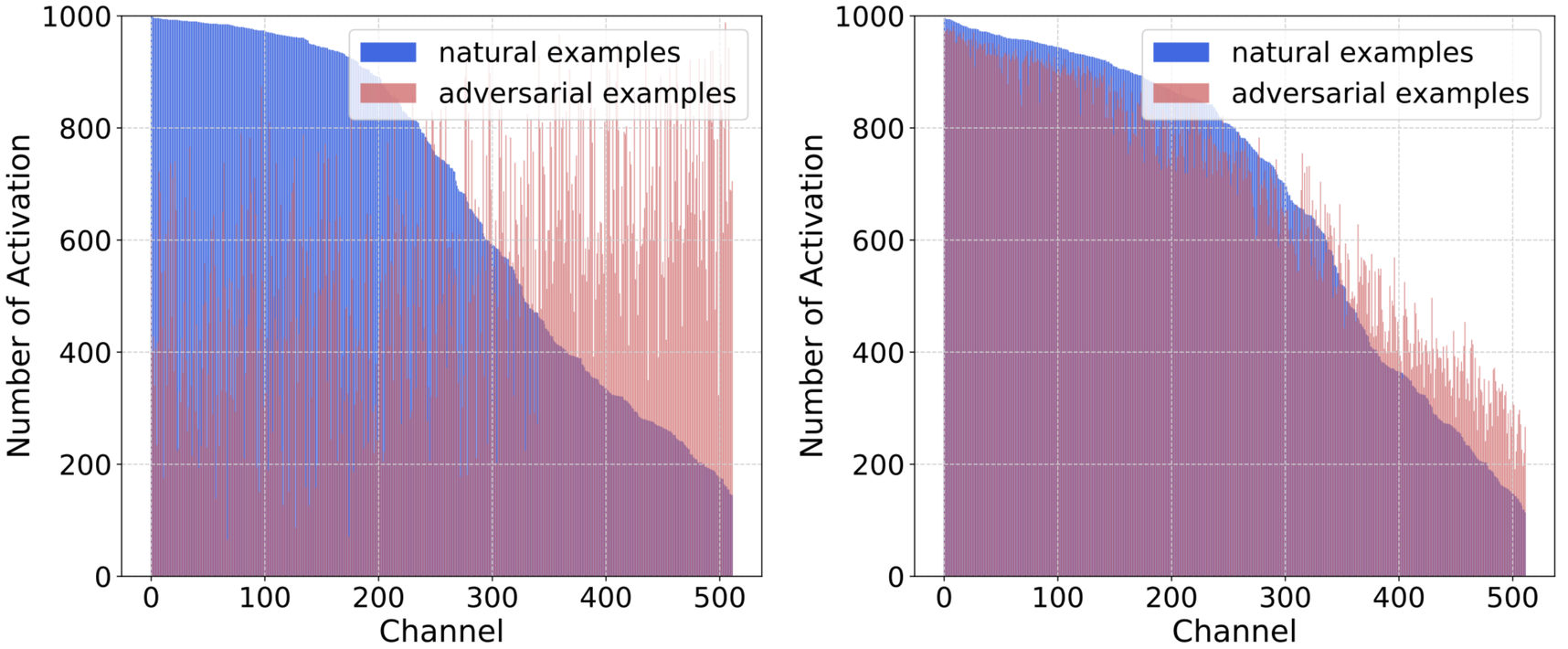


Image Source: [2]

[2] Geometry-Aware Instance-Reweighted Adversarial training, Zhang et al.; https://openreview.net/pdf?id=iAXol6Cz8ub

Adversarial Examples - Activation Pattern



(a) ResNet-18 (STD) (b) ResNet-18 (ADV)

Image Source: [3]

[3] Improving adversarial robustness via channel-wise activation suppressing, Bai et al.; https://openreview.net/forum?id=zQTezqCCtNx

Adversarial Examples - Activation Pattern

Natural examples strongly activate a subset of channels, especially the high-frequency channels on the left side of the plot.

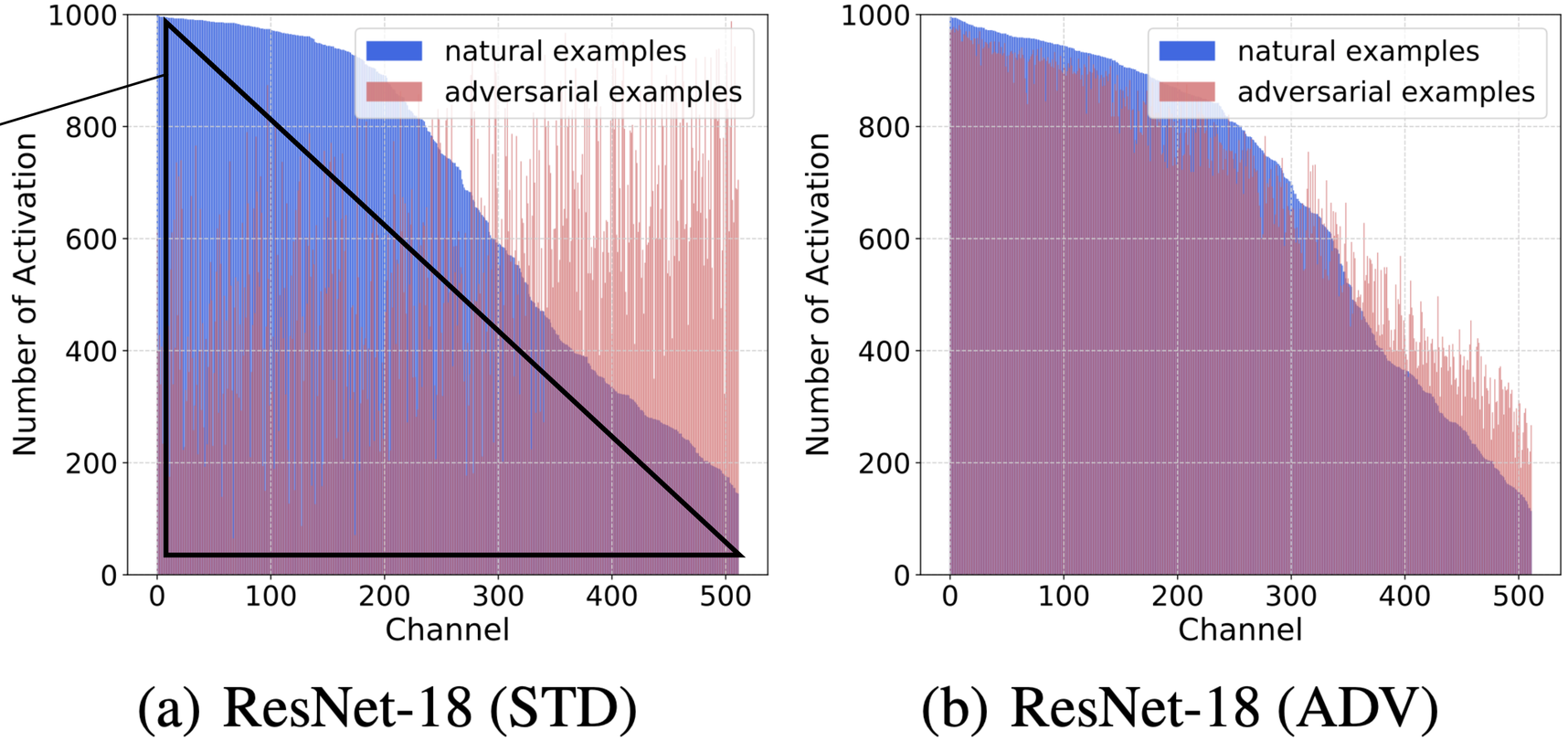
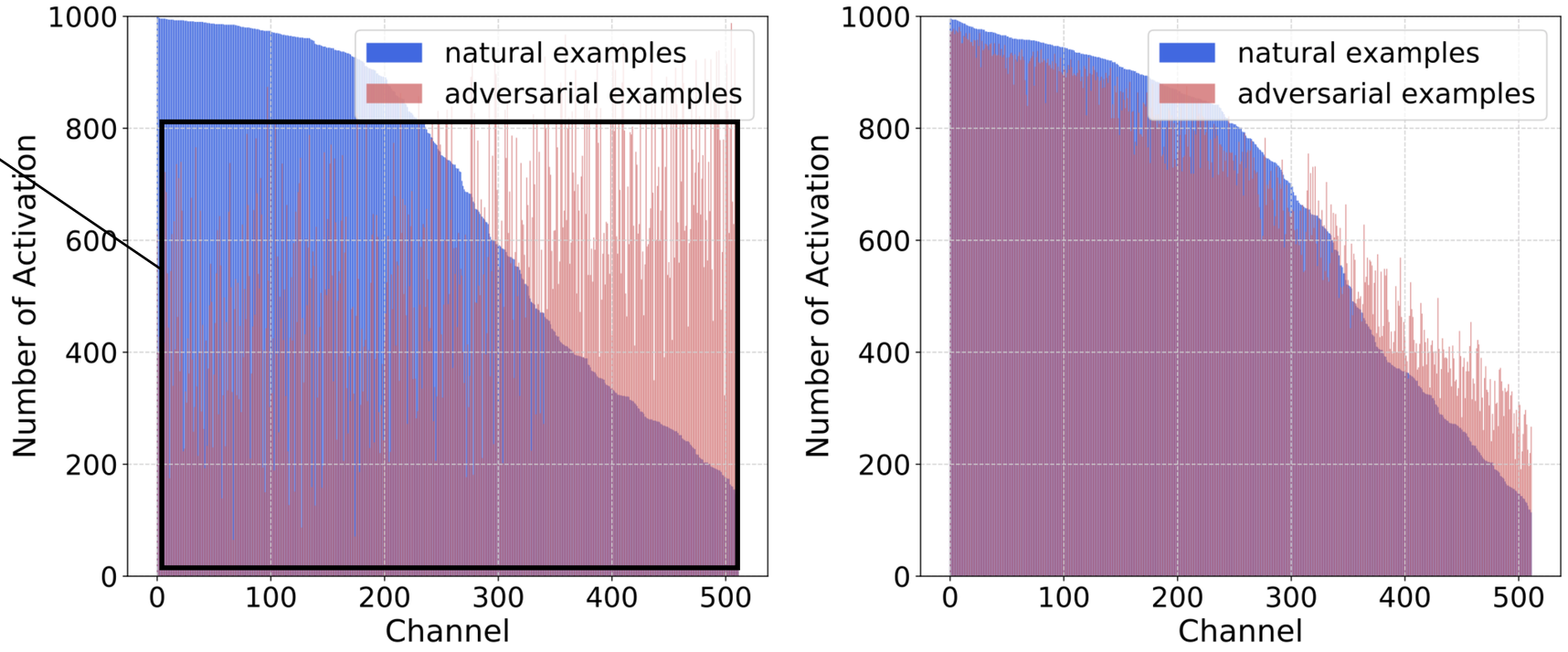


Image Source: [3]

[3] Improving adversarial robustness via channel-wise activation suppressing, Bai et al.; https://openreview.net/forum?id=zQTezqCCtNx

Adversarial Examples - Activation Pattern

Adversarial examples activate channels more uniformly across the whole layer, including many channels on the right side that are rarely activated by natural examples.

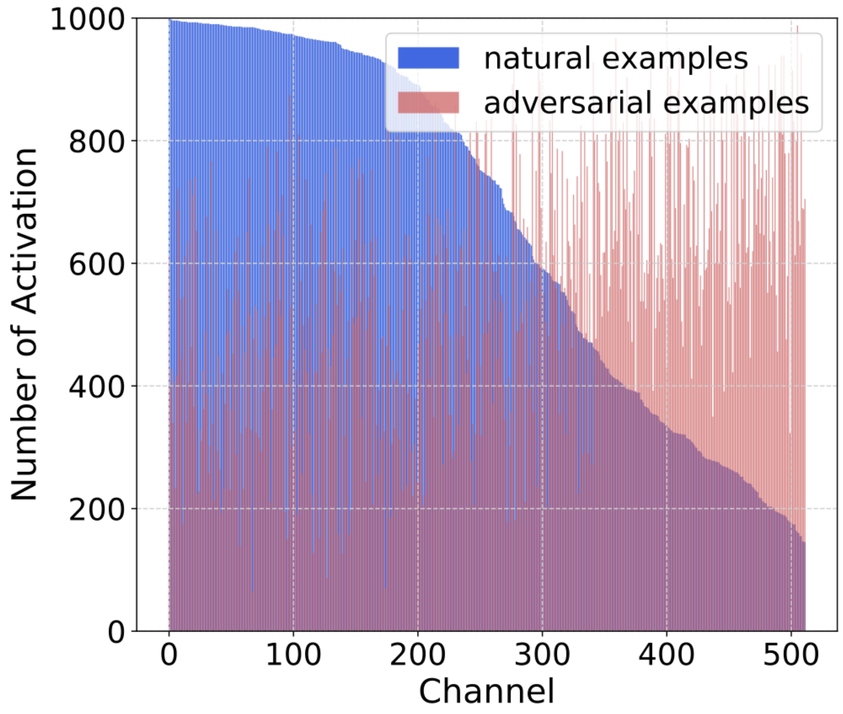


(a) ResNet-18 (STD) (b) ResNet-18 (ADV)

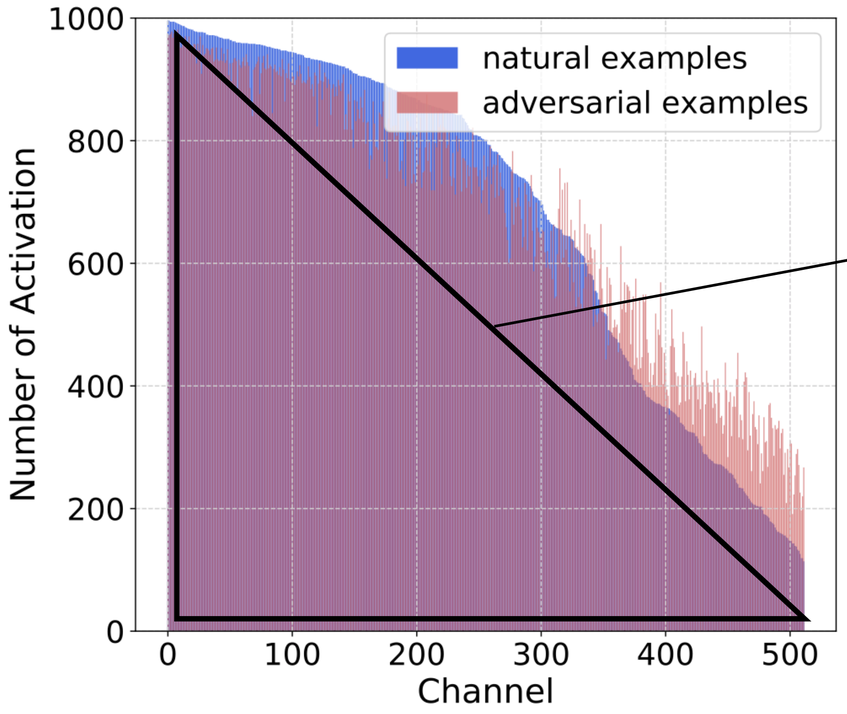
Image Source: [3]

[3] Improving adversarial robustness via channel-wise activation suppressing, Bai et al.; https://openreview.net/forum?id=zQTezqCCtNx

Adversarial Examples - Activation Pattern



(a) ResNet-18 (STD)



(b) ResNet-18 (ADV)

Adversarial training makes the activation patterns of natural and adversarial examples much more similar. However, this similarity is not perfect; some low-frequency channels are still activated by adversarial examples.

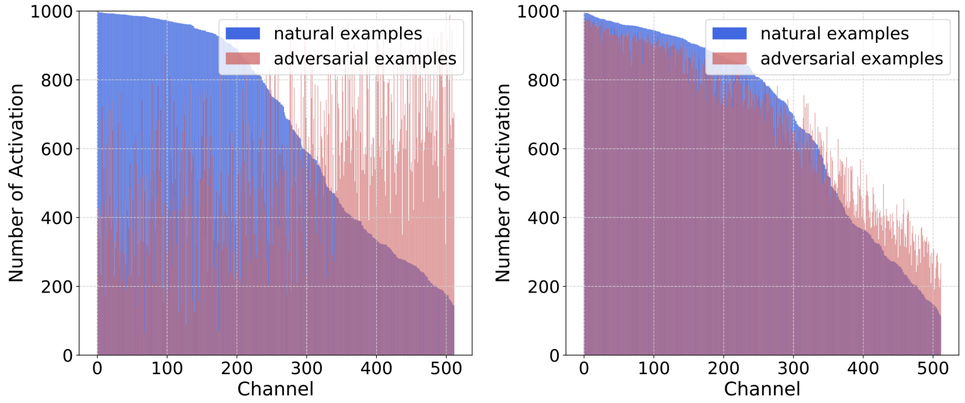
Image Source: [3]

[3] Improving adversarial robustness via channel-wise activation suppressing, Bai et al.; https://openreview.net/forum?id=zQTezqCCtNx

Adversarial Examples - Activation Pattern

Conclusion

- Adversarial training improves robustness by making the model's internal representations **more stable** under adversarial perturbations.
- Instead of allowing adversarial examples to activate many irrelevant or non-robust channels, adversarial training encourages the model to rely on more **consistent**, **class-relevant channels**.



(a) ResNet-18 (STD) (b) ResNet-18 (ADV)

Image Source: [3]

[3] Improving adversarial robustness via channel-wise activation suppressing, Bai et al.; https://openreview.net/forum?id=zQTezqCCtNx

Questions?