模式分析与机器智能
工业和信息化部重点实验室
MIIT Key Laboratory of
Pattern Analysis & Machine Intelligence

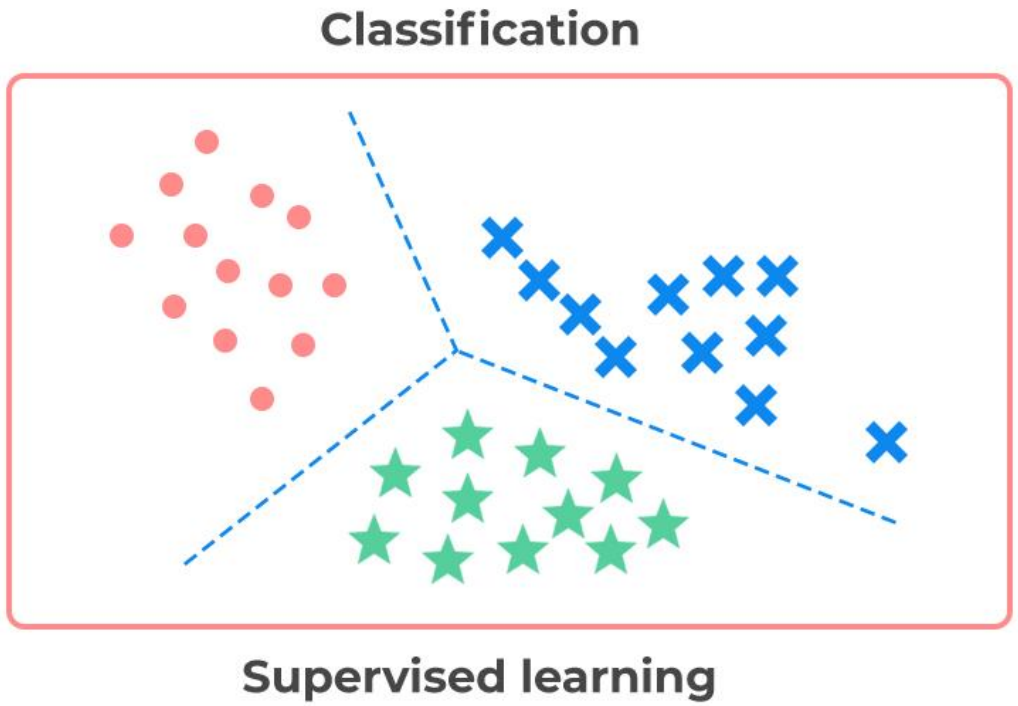
ParNeC 模式识别与神经计算研究组
PAttern Recognition and NEural Computing

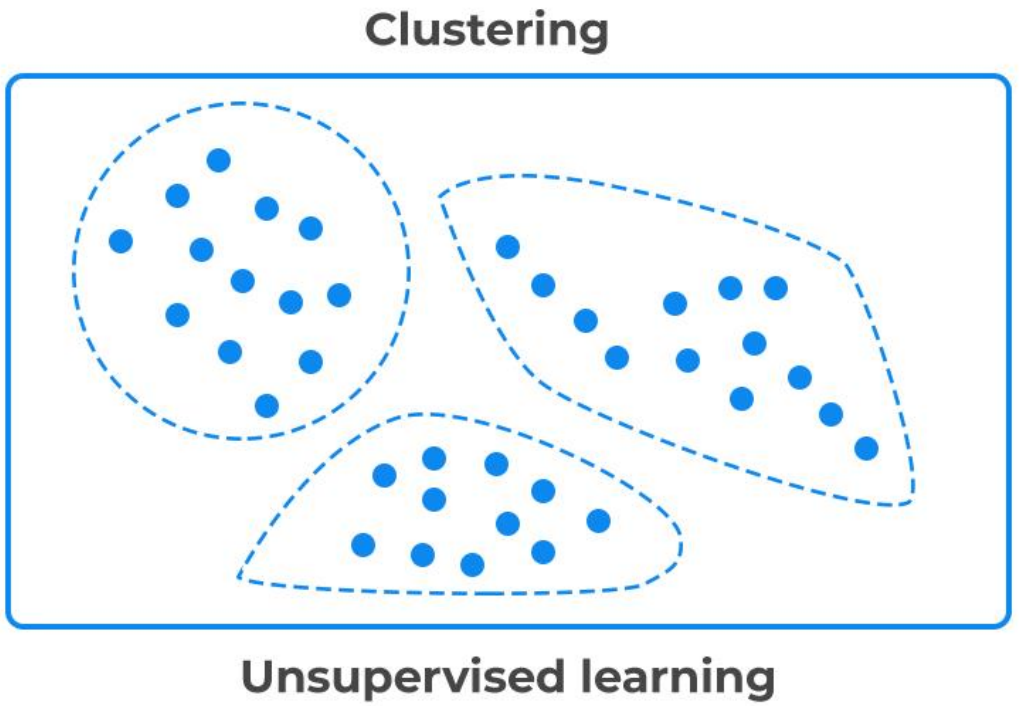
# Adversarial Masking for Self-Supervised Learning

**Yuge Shi** [1] **N. Siddharth** [2] **Philip H.S. Torr** [1] **Adam R. Kosiorek** [3]

*ICML 2022*

# Supervised Learning vs Unsupervised Learning



# Self-Supervised Learning (SSL)

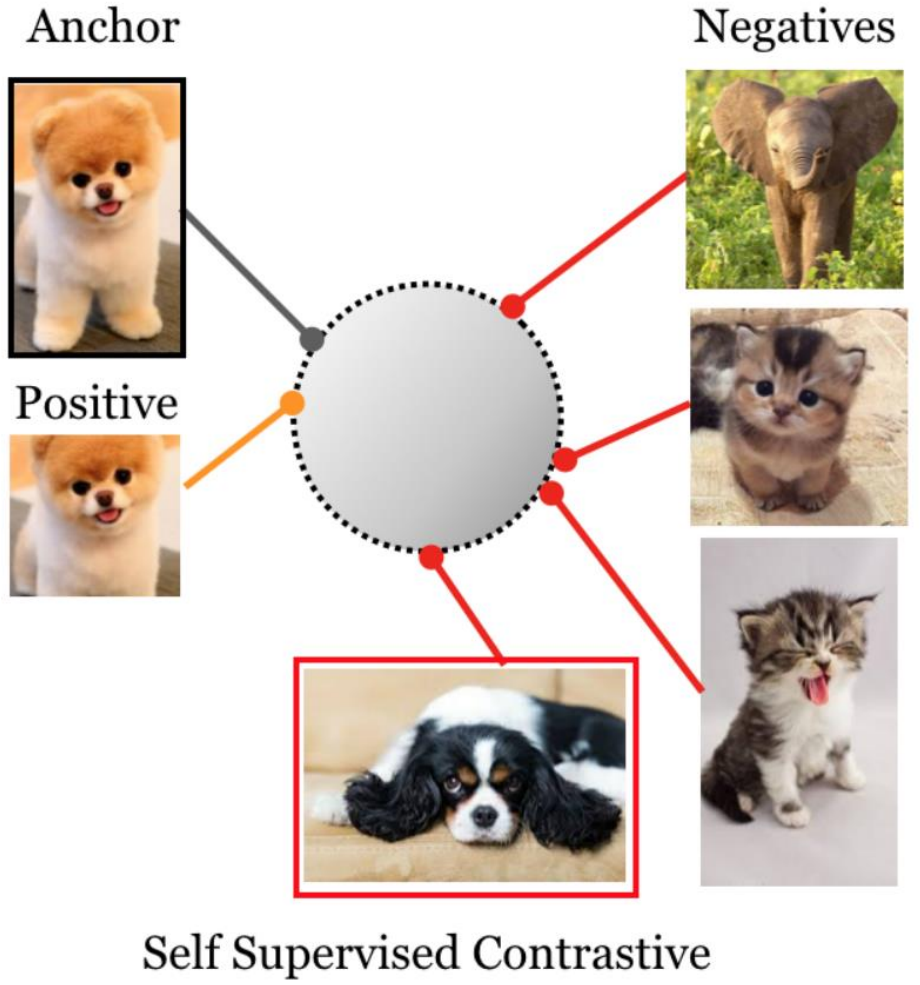
—— a method of **unsupervised** learning.

It can be classified to

① Context Based *

② Temporal Based

③ Contrastive Based *



# Motivation



## Masked Image Model (MIM)

—— ① Context Based

Blue solid box： Other works

(Random Mask)

Red solid box： This work

(Learned Mask)

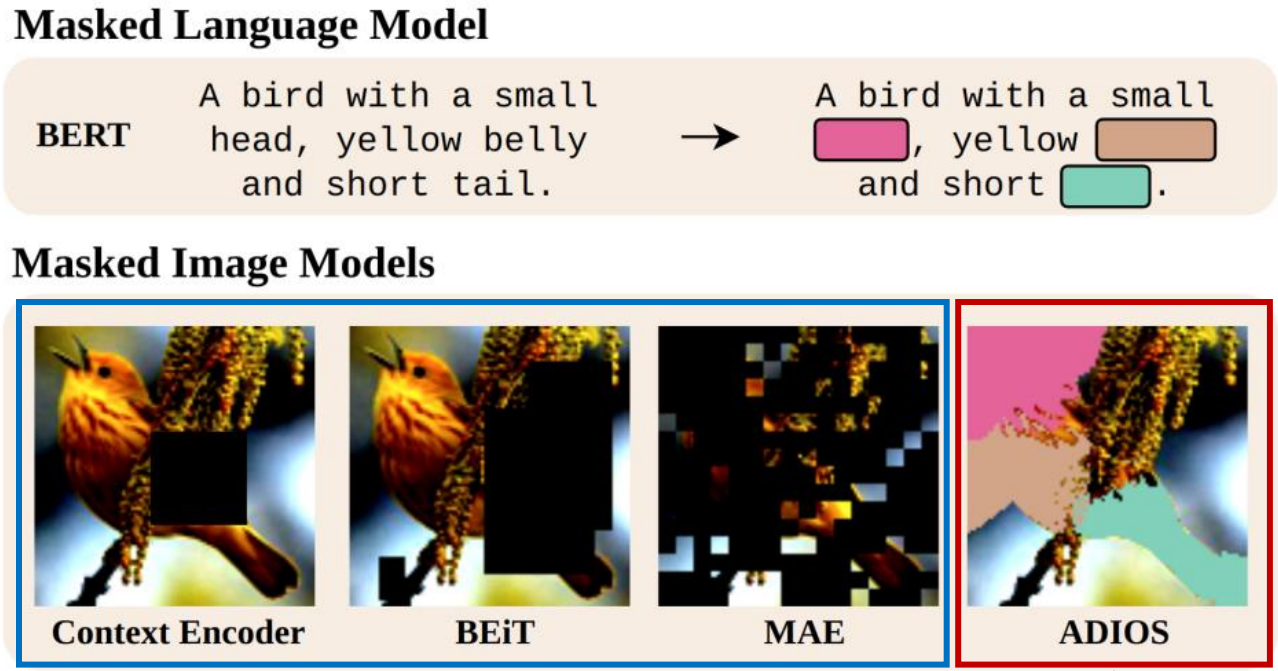



*Figure 1.* Self-supervised language, and vision, models learn representations by imputing data removed by masking. ***BERT***: random word masks; ***Context encoder***: random, fix-shaped mask; ***BEiT***: random 'blockwise' masking; ***MAE***: randomly mask out 75% of the image; ***ADIOS***: multiple masks (N=3) generated by an adversarially trained masking model, post-processed with fully connected conditional random fields (Krähenbühl & Koltun, 2011).

# Adversarial Inference-Occlusion Self-supervision (ADIOS)

**ADIOS** consists of two components
- occlusion model $\mathcal{M}$
- inference model $\mathcal{I}$

The two models are learnt by solving for

$$\mathcal{I}^\star, \mathcal{M}^\star = \arg\min_{\mathcal{I}} \max_{\mathcal{M}} \mathcal{L}(\boldsymbol{x}; \mathcal{I}, \mathcal{M}) \,.$$

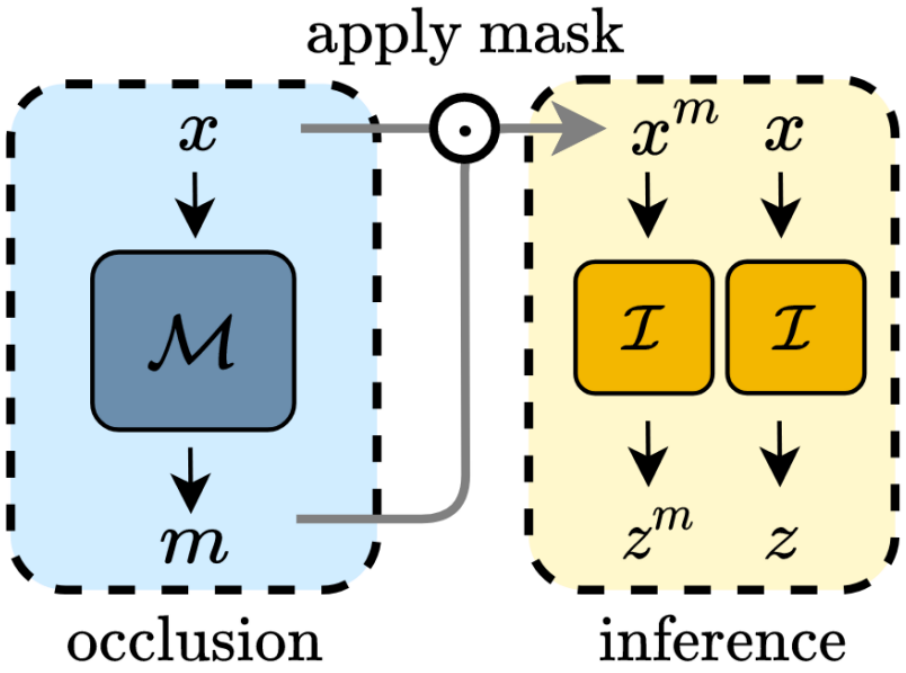
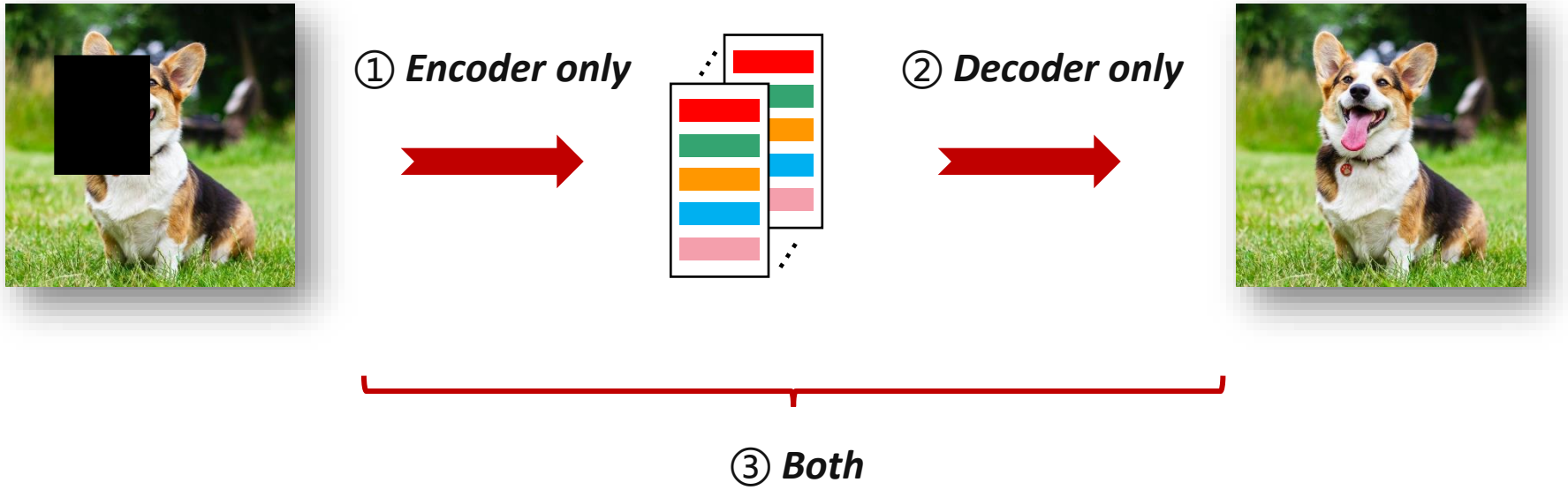



*Figure 2.* ADIOS Architecture.

# Inference model $\mathcal{I}$

An interesting question for auto-encoding $\mathcal{I}$ is : ***where does the imputation happen?***




Given these scenarios, ① is clearly best suited for representation learning, as it requires the encoder to reason about the missing parts based on observed context, beyond just extracting image features.

## Inference model $\mathcal{I}$

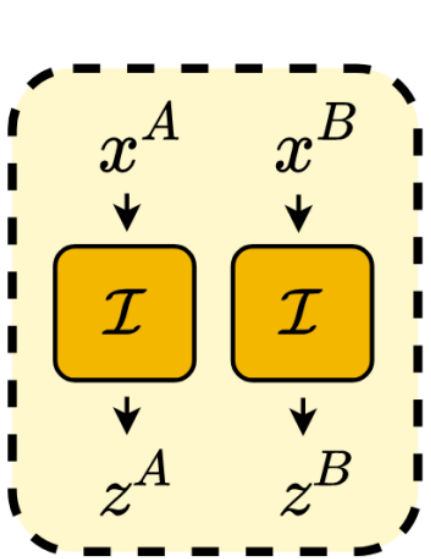
Our objective can thus be written as

$$\mathcal{L}_{\text{ENC}}(\boldsymbol{x};\mathcal{I},\mathcal{M})=\mathcal{D}(\boldsymbol{z},\boldsymbol{z}^{\boldsymbol{m}})=\mathcal{D}(\mathcal{I}(\boldsymbol{x}),\mathcal{I}(\boldsymbol{x}\odot\mathcal{M}(\boldsymbol{x})))$$

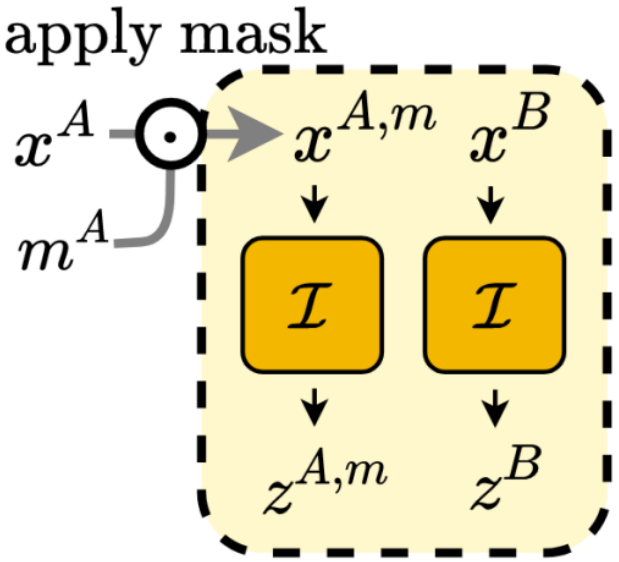
→ Suffer from “collapse” !

In Contrastive Based SSL Framework, we take **SimCLR** as an example

$$\mathcal{L}_{\text{SimCLR}}(\boldsymbol{x};\mathcal{I}) = \log \frac{\exp(\mathcal{D}(\boldsymbol{z}_i^A, \boldsymbol{z}_i^B))}{\sum_{i\neq j}\exp(\mathcal{D}(\boldsymbol{z}_i^A, \boldsymbol{z}_j^B))}$$




***Left***: SimCLR




***Right***: SimCLR + ADIOS

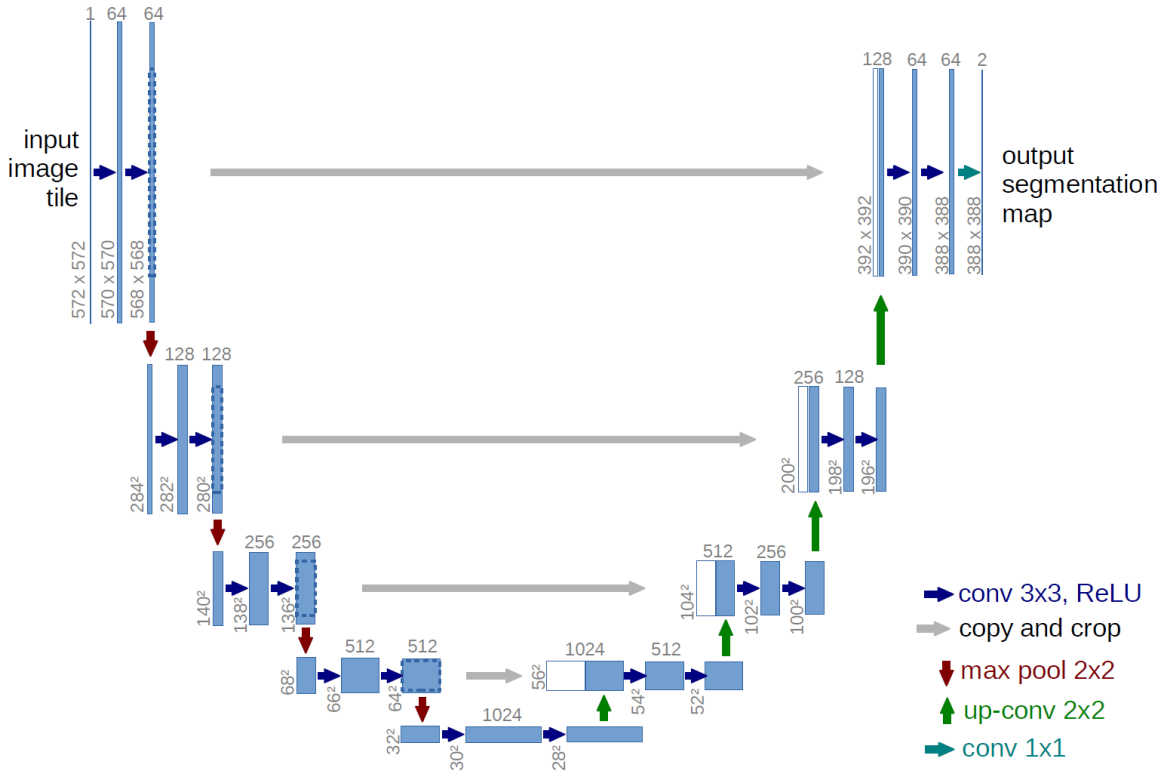
So, we can write the SimCLR-ADIOS objective as

$$\mathcal{L}^{\text{ADIOS}}_{\text{SimCLR}}(\boldsymbol{x};\mathcal{I},\mathcal{M}) = \log \frac{\exp(\mathcal{D}(\boldsymbol{z}_i^{A,\boldsymbol{m}}, \boldsymbol{z}_i^B))}{\sum_{i\neq j}\exp(\mathcal{D}(\boldsymbol{z}_i^{A,\boldsymbol{m}}, \boldsymbol{z}_j^B))}$$

$$= \log \frac{\exp(\mathcal{D}(\mathcal{I}(\boldsymbol{x}_i^A \odot \mathcal{M}(\boldsymbol{x}_i^A)), \mathcal{I}(\boldsymbol{x}_i^B)))}{\sum_{i\neq j}\exp(\mathcal{D}(\mathcal{I}(\boldsymbol{x}_i^A \odot \mathcal{M}(\boldsymbol{x}_i^A)), \mathcal{I}(\boldsymbol{x}_j^B)))}.$$

# Occlusion model $\mathcal{M}$

We find that the simplest setup suffices, and we use **U-Net** as the backbone, which is commonly used for semantic segmentation, a pixelwise SoftMax layer to generate $N > 1$ masks.




U-Net

and

| Occlusion Head |
| --- |
| 1x1 conv. $N$ stride 1 pad 1 & SoftMax |

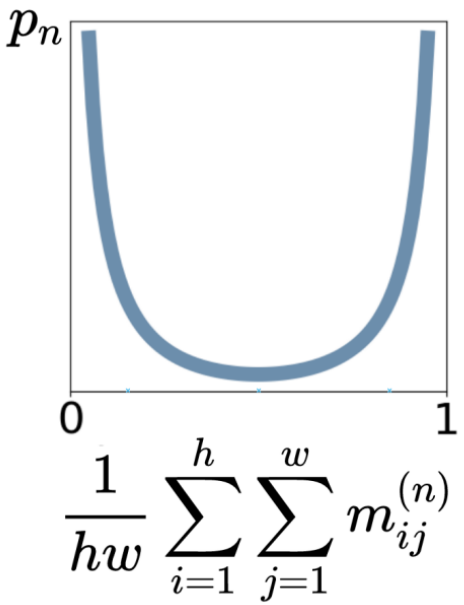
$$\mathcal{M} : \mathbb{R}^{c \times w \times h} \mapsto \mathbb{R}^{N \times w \times h}$$

## ADIOS （$\mathcal{M}$ and $\mathcal{I}$）

The two models are learnt by solving for

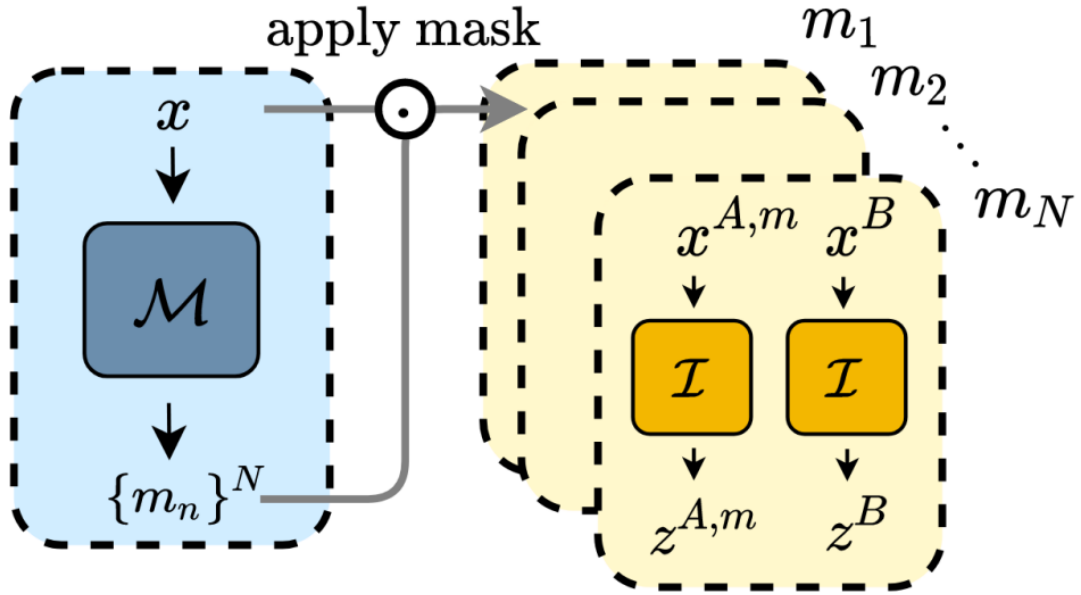
$$\mathcal{I}^\star, \mathcal{M}^\star = \arg\min_{\mathcal{I}} \max_{\mathcal{M}} \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}^{(n)}(\boldsymbol{x}; \mathcal{I}, \mathcal{M}) \,.$$

Some mask $\boldsymbol{m_n}$ occludes everything, with the other $\{\boldsymbol{N}\}_{\backslash \boldsymbol{n}}$ masks not occluding anything.




We introduce a sparsity penalty $\boldsymbol{p_n}$

$$p_n = \sin\left(\frac{\pi}{hw}\sum_{i=1}^{h}\sum_{j=1}^{w} \boldsymbol{m}_{ij}^{(n)}\right)^{-1}$$




**ADIOS**, $N > 1$.

# ADIOS （$\mathcal{M}$ and $\mathcal{I}$）

**Final objective :**

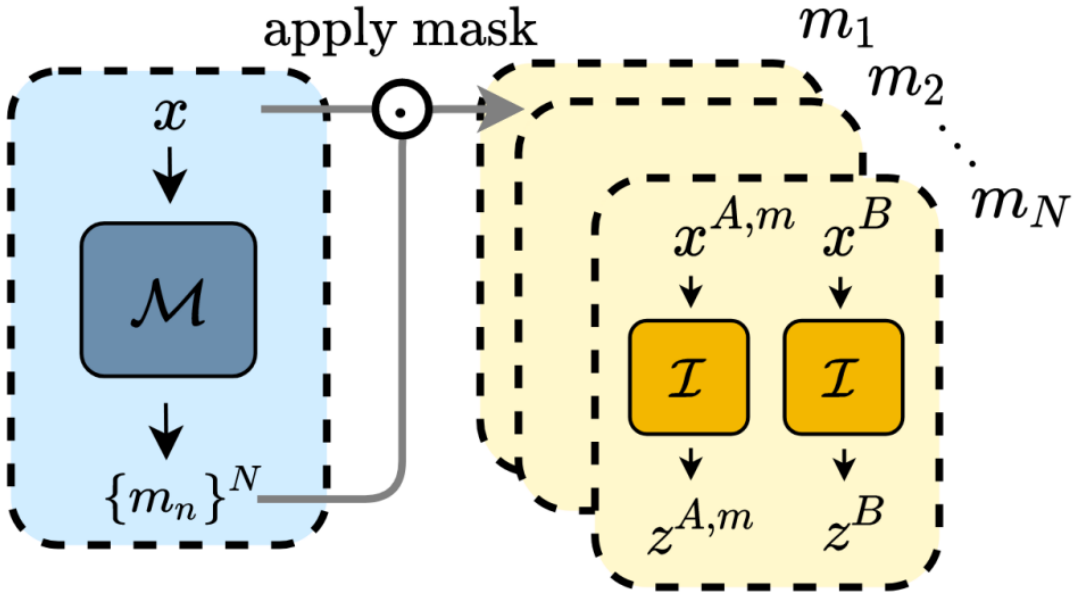
$$\mathcal{I}^\star, \mathcal{M}^\star = \arg\min_{\mathcal{I}} \max_{\mathcal{M}} \frac{1}{N} \sum_{n=1}^{N} \left( \mathcal{L}^{(n)}(\boldsymbol{x}; \mathcal{I}, \mathcal{M}) - \lambda p_n \right)$$

Randomly sample one ↓ from *N* generated masks.

**Lightweight ADIOS :**

$$\mathcal{I}^\star, \mathcal{M}^\star = \arg\min_{\mathcal{I}} \max_{\mathcal{M}} \left( \mathcal{L}^{(k)}(\boldsymbol{x}; \mathcal{I}, \mathcal{M}) - \lambda \frac{1}{N} \sum_{n=1}^{N} p_n \right)$$

$$\text{where } k \sim \text{Uniform}\left(\{1, 2..., N\}\right).$$




**ADIOS**, *N* > 1.

## Linear evaluation and *k*-NN

Models using **ADIOS** consistently ***outperform*** their respective SSL baselines beyond the margin of error.

*Table 1.* Top-1 classification accuracy ($k$-NN and Linear Probing) on Imagenet100-S, STL10. Improvements of ADIOS that are more than 3% are marked in bold.

| Method | ImageNet100-S | | STL10 | |
|---|---|---|---|---|
| | $k$-NN | Linear | $k$-NN | Linear |
| *Backbone: ViT-Tiny* | | | | |
| SimCLR | 40.0 (±0.28) | 40.2 (±0.47) | 72.9 (±0.27) | 76.0 (±0.33) |
| +ADIOS | 42.0 (±1.32) | 43.1 (±0.71) | 73.4 (±0.28) | **79.7** (±0.88) |
| SimSiam | 35.2 (±1.12) | 36.8 (±1.82) | 66.7 (±0.10) | 67.5 (±0.02) |
| +ADIOS | **38.8** (±2.73) | **40.1** (±0.59) | 67.9 (±0.75) | 68.8 (±0.25) |
| BYOL | 38.1 (±0.61) | 39.7 (±0.50) | 71.9 (±0.12) | 72.1 (±0.32) |
| +ADIOS | **47.1** (±0.35) | **49.2** (±0.94) | 74.5 (±0.58) | **75.9** (±0.63) |
| *Backbone: ResNet-18* | | | | |
| SimCLR | 54.1 (±0.09) | 55.1 (±0.15) | 83.7 (±0.48) | 85.1 (±0.12) |
| +ADIOS | 55.1 (±0.43) | 55.9 (±0.21) | 85.8 (±0.10) | 86.1 (±0.36) |
| SimSiam | 58.6 (±0.31) | 59.5 (±0.31) | 84.3 (±0.81) | 84.8 (±0.72) |
| +ADIOS | 61.0 (±0.29) | 60.4 (±0.19) | 84.6 (±0.35) | 86.4 (±0.24) |
| BYOL | 56.2 (±0.79) | 56.3 (±0.10) | 83.6 (±0.09) | 84.3 (±0.13) |
| +ADIOS | **60.2** (±0.82) | **61.4** (±0.14) | 84.8 (±0.19) | 85.6 (±0.24) |

## Lightweight ADIOS and clustering

The results in Tab. 2 show that this much cheaper model also achieves impressive performance.

The results in Tab. 3 show that **ADIOS** improves the performance of baseline SSL methods on all three metrics for both datasets.

*Table 2.* Top-1 classification accuracy of linear probing on ImageNet100. Improvements of more than 3% are marked in bold.

| SimCLR | +ADIOS-s | SimSiam | +ADIOS-s | BYOL | +ADIOS-s |
|---|---|---|---|---|---|
| 77.5 (±0.10) | 76.1 (±0.50) | 76.4 (±0.07) | 77.2 (±0.09) | 74.3 (±0.16) | **80.8 (±0.60)** |

*Table 3.* Clustering performance on Imagenet100-S, STL10.

| Method | Backbone | Metrics | | |
|---|---|---|---|---|
| | | FMI ↑ | ARI ↑ | NMI ↑ |
| *Dataset: ImageNet100-S* | | | | |
| SimCLR | ViT-Tiny | 0.105 (±1e-3) | 0.095 (±1e-3) | 0.432 (±3e-3) |
| +ADIOS | ViT-Tiny | 0.120 (±1e-3) | 0.110 (±1e-3) | 0.442 (±4e-3) |
| SimSiam | ViT-Tiny | 0.077 (±9e-4) | 0.067 (±2e-3) | 0.389 (±3e-3) |
| +ADIOS | ViT-Tiny | 0.098 (±1e-2) | 0.087 (±9e-4) | 0.425 (±3e-3) |
| BYOL | ViT-Tiny | 0.098 (±8e-3) | 0.088 (±8e-3) | 0.418 (±4e-3) |
| +ADIOS | ViT-Tiny | 0.132 (±3e-3) | 0.123 (±1e-3) | 0.458 (±4e-3) |
| SimCLR | ResNet18 | 0.151 (±3e-3) | 0.135 (±4e-3) | 0.515 (±6e-3) |
| +ADIOS | ResNet18 | 0.175 (±1e-3) | 0.161 (±4e-3) | 0.539 (±3e-3) |
| SimSiam | ResNet18 | 0.167 (±2e-3) | 0.136 (±6e-3) | 0.553 (±8e-3) |
| +ADIOS | ResNet18 | 0.179 (±1e-3) | 0.161 (±1e-3) | 0.553 (±1e-3) |
| BYOL | ResNet18 | 0.170 (±1e-3) | 0.158 (±3e-3) | 0.530 (±4e-3) |
| +ADIOS | ResNet18 | 0.179 (±6e-4) | 0.156 (±2e-3) | 0.561 (±2e-3) |
| *Dataset: STL10* | | | | |
| SimCLR | ViT-Tiny | 0.349 (±5e-3) | 0.269 (±6e-3) | 0.410 (±2e-3) |
| +ADIOS | ViT-Tiny | 0.351 (±4e-3) | 0.271 (±8e-3) | 0.417 (±6e-3) |
| SimSiam | ViT-Tiny | 0.296 (±3e-3) | 0.177 (±1e-3) | 0.341 (±4e-3) |
| +ADIOS | ViT-Tiny | 0.320 (±3e-3) | 0.235 (±5e-3) | 0.349 (±0e-0) |
| BYOL | ViT-Tiny | 0.349 (±5e-3) | 0.269 (±5e-3) | 0.410 (±5e-3) |
| +ADIOS | ViT-Tiny | 0.355 (±4e-2) | 0.276 (±3e-3) | 0.422 (±4e-3) |
| SimCLR | ResNet18 | 0.338 (±2e-3) | 0.166 (±9e-4) | 0.512 (±5e-3) |
| +ADIOS | ResNet18 | 0.437 (±6e-3) | 0.309 (±9e-3) | 0.585 (±8e-3) |
| SimSiam | ResNet18 | 0.392 (±2e-3) | 0.242 (±7e-3) | 0.552 (±3e-3) |
| +ADIOS | ResNet18 | 0.412 (±8e-3) | 0.249 (±7e-3) | 0.558 (±2e-4) |
| BYOL | ResNet18 | 0.429 (±5e-3) | 0.328 (±9e-3) | 0.525 (±8e-3) |
| +ADIOS | ResNet18 | 0.508 (±6e-3) | 0.422 (±1e-2) | 0.588 (±9e-3) |

# Transfer learning

Results show that **ADIOS** improves transfer learning performance on all four datasets, under both linear evaluation and fine-tuning.

We use **CIFAR-ResNet** for fine-tuning, however we have to use the original architecture for linear evaluation in order to use the pretrained weights, which leads to poor performance.

*Table 4.* Classification accuracy of transfer learning by re-training linear classifier only (*Lin.*) and fine-tuning (*F.T*). More than 3% improvements by ADIOS are marked in bold.

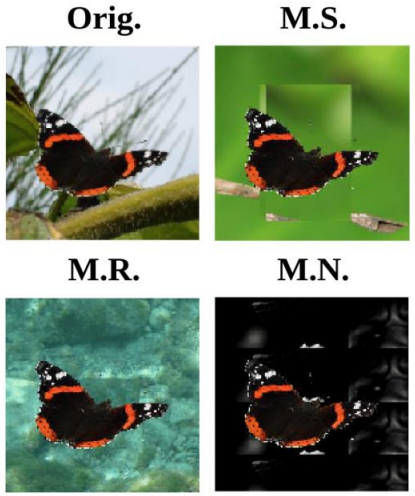
| Method | CIFAR10 | | CIFAR100 | | Flowers102 | | iNaturalist | |
|---|---|---|---|---|---|---|---|---|
| | Lin. | F.T. | Lin. | F.T. | Lin. | F.T. | Lin. | F.T. |
| SimCLR | 30.1 | 91.3 | 10.2 | 70.0 | 42.5 | 45.6 | 69.4 | 82.1 |
| +ADIOS | **34.6** | 93.4 | 11.0 | 71.8 | **50.2** | **50.6** | **72.5** | 84.3 |
| SimSiam | 35.3 | 92.4 | 13.2 | 65.0 | 38.7 | 55.0 | 72.3 | 85.0 |
| +ADIOS | **39.3** | 94.3 | 13.3 | **71.0** | **44.9** | **59.0** | **75.9** | 86.2 |
| BYOL | 29.9 | 88.0 | 13.3 | 52.3 | 49.1 | 58.6 | 72.7 | 85.1 |
| +ADIOS | **39.2** | 90.4 | 14.0 | **62.0** | 51.7 | 60.1 | 73.1 | 85.7 |
| Scratch | - | 85.5 | - | 49.8 | - | 30.6 | - | 73.8 |

# Experiment



## Robustness

The original figure's (**Orig.**) background is replaced by background from another image in the same class (**M.S.**), from a random image in any class (**M.R.**) or from an image in the next class (**N.R.**).




Our results show that all three **SSL-ADIOS** models outperform their respective baselines on all variations, demonstrating that **ADIOS-learned** representations are more robust to changes in both foreground and background.

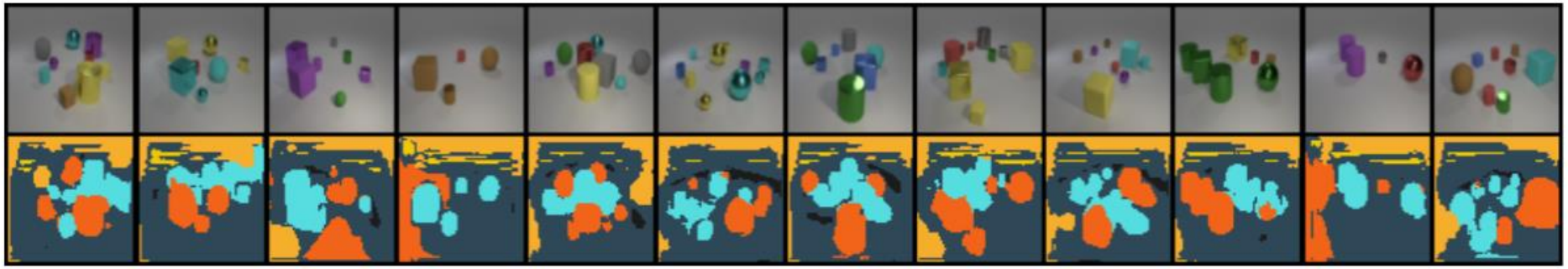
*Table 5.* Accuracy on different variations of the backgrounds challenge, evaluating model robustness. Example variations in Fig. 7.

| Method | Variations | | | | | | | Orig. |
|---|---|---|---|---|---|---|---|---|
| | O.BB. | O.BT. | N.F. | O.F. | M.S. | M.R. | M.N. | |
| SimCLR | 20.1 | 34.8 | 44.3 | 41.6 | 67.1 | 45.9 | 41.0 | 78.8 |
| +ADIOS | 20.7 | 36.7 | 45.5 | 43.5 | 68.0 | 47.9 | 43.7 | 79.1 |
| SimSiam | 29.5 | 39.1 | 43.8 | 52.1 | 69.9 | 43.9 | 40.8 | 78.4 |
| +ADIOS | 33.1 | 41.0 | 46.2 | 54.7 | 71.5 | 47.2 | 43.5 | 80.3 |
| BYOL | 25.9 | 38.4 | 46.0 | 51.6 | 71.3 | 45.6 | 42.7 | 79.8 |
| +ADIOS | 27.7 | 39.0 | 48.5 | 51.7 | 72.1 | 47.8 | 44.1 | 80.6 |
| Avg. gain | +2.0 | +1.4 | +2.0 | +1.5 | +1.1 | +2.5 | +2.3 | +1.0 |

# Experiment



(a) STL10, $N = 6$.



(b) CLEVR, $N = 4$.

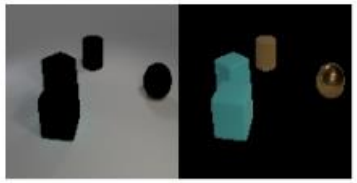
(c) ImageNet100-S, $N = 4$.

*Figure 8.* Masks generated by ADIOS during training on each dataset. $N$ denotes the number of masks. *Top row:* original image; *Bottom row:* generated masks, each color represents one mask.

# Experiment

## Masking Schemes

(a) Ground-truth object masks



(b) Foreground-background masks

(c) Ground-truth box-shaped masks

(d) Shuffled ground-truth masks

(e) MAE (He et al., 2021) masks

(f) BEiT (Bao et al., 2021) masks

*Table 6.* Top-1 classification accuracy on ImageNet100-S and STL10 under different masking schemes, averaged over three runs of SimCLR, SimSiam and BYOL respectively. Best results for each metric in bold.

| Mask type | Condition | Dataset | |
|---|---|---|---|
| | | ImageNet100-S | STL10 |
| Random | e) MAE | 43.7 (±0.43) | 78.4 (±0.91) |
| | f) BEiT | 46.4 (±0.67) | 80.7 (±1.00) |
| Learned | ADIOS | **59.2 (±2.92)** | **86.0 (±0.40)** |
| None | - | 57.0 (±2.27) | 84.7 (±0.40) |

*Table 7.* Multi-label classification on CLEVR under different masking schemes, averaged over three runs of SimCLR, SimSiam and BYOL respectively. Best results for each metric in bold.

| Mask type | Condition | Metric | | |
|---|---|---|---|---|
| | | F1-macro ↑ | F1-micro ↑ | F1-weighted ↑ |
| Semantic | a) G.T. | **0.373 (±7e-3)** | **0.401 (±2e-4)** | **0.460 (±1e-2)** |
| | b) FG./BG. | 0.346 (±7e-3) | 0.365 (±2e-4) | 0.402 (±1e-3) |
| | c) Box | 0.347 (±2e-4) | 0.391 (±3e-5) | **0.457 (±5e-2)** |
| Random | d) Shuffle | 0.332 (±6e-3) | 0.360 (±8e-4) | 0.418 (±1e-3) |
| | e) MAE | 0.309 (±8e-4) | 0.336 (±3e-4) | 0.391 (±9e-4) |
| | f) BEiT | 0.274 (±1e-3) | 0.307 (±2e-4) | 0.395 (±7e-3) |
| Learned | ADIOS | **0.377 (±2e-3)** | 0.385 (±9e-4) | **0.451 (±1e-3)** |
| None | - | 0.352 (±9e-3) | 0.359 (±2e-4) | 0.373 (±2e-5) |

# Thanks

模式分析与机器智能
工业和信息化部重点实验室
MIIT Key Laboratory of
Pattern Analysis & Machine Intelligence

ParNeC 模式识别与神经计算研究组
PAttern Recognition and NEural Computing

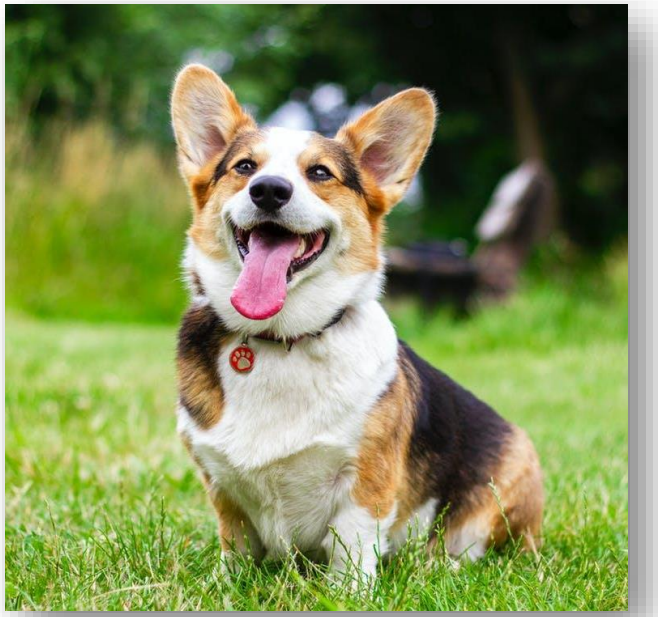
# To Smooth or Not? When Label Smoothing Meets Noisy Labels

**Jiaheng Wei** [1] **Hangyu Liu** [2] **Tongliang Liu** [3] **Gang Niu** [4] **Masashi Sugiyama** [4 5] **Yang Liu** [1]

*ICML 2022*

## Noisy labels

In multi-class classification task



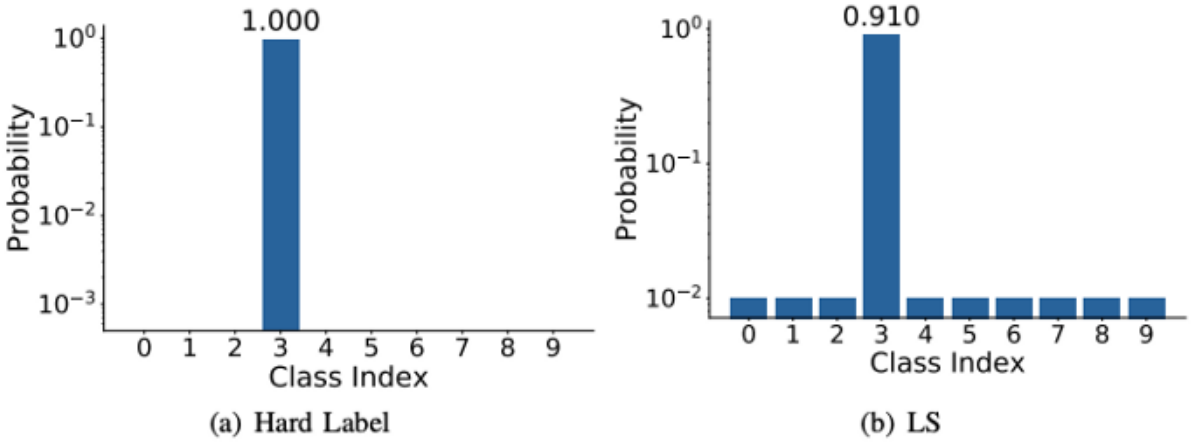
**Dog** (clean)

**Cat** (noisy)

**Type**
- symmetric label noise
- asymmetric label noise
  - for all labels
  - for single labels

## Label smoothing (LS)

$$\mathbf{y}^{\mathrm{LS},r} = (1-r)\cdot\mathbf{y} + \frac{r}{K}\cdot\mathbf{1}$$




(a) Hard Label (b) LS

$K$ is the number of label classes

$r$ is the smooth rate in the range of [0, 1]

# Motivation



LS helps with improving robustness when learning with noisy labels.(from '***Does label smoothing mitigate label noise?***')

***But …***

**The advantage of LS vanishes in a high label noise regime!**

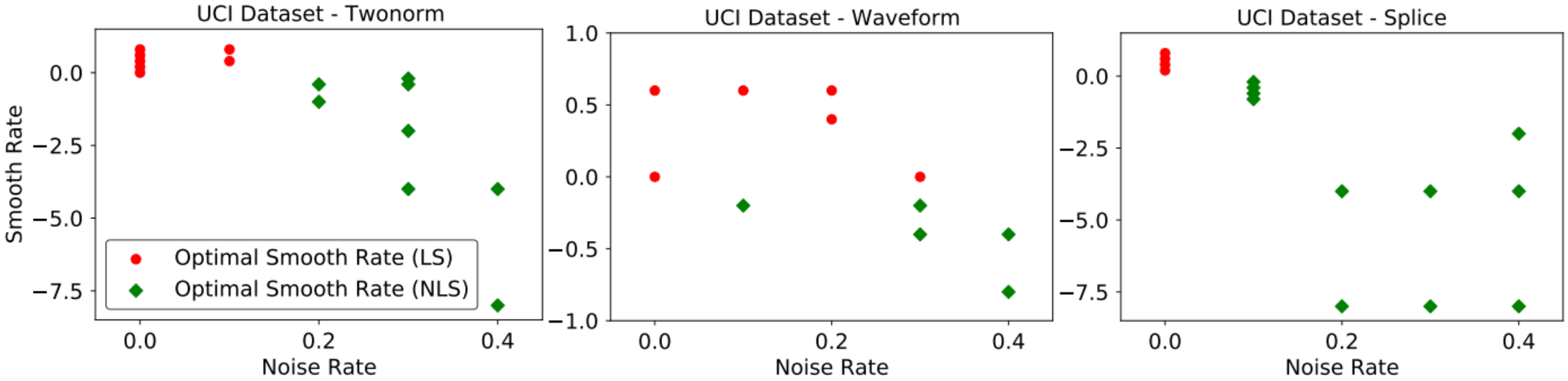



*Figure 1.* Optimal smooth rates on UCI datasets with different label noise rates (possible to have tied smooth rates).

## Generalized label smoothing (GLS)

$$\mathbf{y}_i^{\mathrm{LS},r} = (1-r)\cdot\mathbf{y}_i + \frac{r}{K}\cdot\mathbf{1} \quad r\in[0,1]$$

extend

$$\mathbf{y}_i^{\mathrm{GLS},r} := (1-r)\cdot\mathbf{y}_i + \frac{r}{K}\cdot\mathbf{1} \quad r\in(-\infty,1]$$

$$H(p,q) = -\sum_x p(x)\,\log q(x)$$

$$[1,0,0]^\top$$

NLS

$$[1.2,-0.1,-0.1]^\top$$

CE

$$\text{loss } 1.2\cdot\ell(\mathbf{f}(x),0) - 0.1\cdot\ell(\mathbf{f}(x),1) - 0.1\cdot\ell(\mathbf{f}(x),2)$$

# Experiment



*Table 1.* Test accuracies of LS, VL, NLS on clean and noisy UCI Heart, Splice datasets, with best two smooth rates highlighted (green: NLS; red: VL or LS). We adopt the two independent sample T-test (5 non-negative smooth rates V.S. the last 5 rows of reported negative smooth rates) to verify the overall performance comparisons between VL/LS and NLS. $p$-value is highlighted in green if NLS generally returns a higher accuracy (i.e., $t$-value$< 0$) than VL/LS, otherwise, in red. Results on more benchmark datasets are given in Appendix D.

| Smooth Rate | *UCI-Heart* | | | | | *UCI-Splice* | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $e_i = 0$ | $e_i = 0.1$ | $e_i = 0.2$ | $e_i = 0.3$ | $e_i = 0.4$ | $e_i = 0$ | $e_i = 0.1$ | $e_i = 0.2$ | $e_i = 0.3$ | $e_i = 0.4$ |
| $r = 0.8$ | 0.885 | 0.853 | 0.836 | 0.820 | 0.738 | 0.980 | 0.946 | 0.919 | 0.856 | 0.760 |
| $r = 0.6$ | 0.902 | 0.836 | 0.820 | 0.836 | 0.738 | 0.978 | 0.939 | 0.913 | 0.869 | 0.778 |
| $r = 0.4$ | 0.885 | 0.853 | 0.836 | 0.820 | 0.771 | 0.978 | 0.948 | 0.922 | 0.885 | 0.797 |
| $r = 0.2$ | 0.902 | 0.853 | 0.820 | 0.803 | 0.754 | 0.978 | 0.948 | 0.919 | 0.878 | 0.800 |
| $r = 0.0$ | 0.902 | 0.853 | 0.820 | 0.820 | 0.771 | 0.976 | 0.948 | 0.926 | 0.876 | 0.806 |
| $r = -0.4$ | 0.869 | 0.836 | 0.803 | 0.853 | 0.754 | 0.961 | 0.956 | 0.928 | 0.880 | 0.817 |
| $r = -0.6$ | 0.869 | 0.836 | 0.820 | 0.853 | 0.721 | 0.961 | 0.956 | 0.926 | 0.880 | 0.819 |
| $r = -1.0$ | 0.885 | 0.869 | 0.803 | 0.853 | 0.754 | 0.956 | 0.954 | 0.932 | 0.889 | 0.819 |
| $r = -2.0$ | 0.885 | 0.869 | 0.820 | 0.853 | 0.787 | 0.952 | 0.946 | 0.935 | 0.898 | 0.830 |
| $r = -4.0$ | 0.885 | 0.869 | 0.853 | 0.885 | 0.820 | 0.946 | 0.943 | 0.939 | 0.911 | 0.830 |
| $r = -8.0$ | 0.869 | 0.869 | 0.885 | 0.853 | 0.853 | 0.943 | 0.946 | 0.939 | 0.915 | 0.845 |
| $r_{\text{opt}} =$ | [0.0, 0.6] | [-8.0, -1.0] | -8.0 | -4.0 | -8.0 | 0.8 | [-0.6, -0.4] | [-8.0, -4.0] | -8.0 | -8.0 |
| $p$-value $=$ | 0.020 | 0.136 | 0.549 | 0.002 | 0.243 | 0.001 | 0.332 | 0.002 | 0.015 | 0.005 |

# Thanks