AutoLoss-GMS: Searching Generalized Margin-based Softmax Loss Function for Person Re-identification

Supplementary Material

In the supplementary material, unless otherwise specified, the numbers of the figures and tables are within the scope of the supplementary material.

A. The primitive operations set

There are a total of 24 primitive operations used, of which 4 are binary operations, and 20 are unary operations, as shown in Table 1. In addition to the common "Sig" function, more S-shaped functions such as "Gd", "Alf", "Erf", and "Erfc" have also been considered to increase the diversity of the search space. Inspired by CircleLoss, the "Detach" operation is added as a self-paced weight.

Table 1. The primitive operations set $\mathcal{H}$.

Operator	Expression	Arity
Add	$x + y$	2
Minus	$x - y$	2
Mul	$x \times y$	2
Div	x/y	2
Neg	$-x$	1
Abs	$\|x\|$	1
Inv	$1/x$	1
Log	$\log(x)$	1
Exp	e^x	1
Sig	$1/(1+e^{-x})$	1
Tanh	$(e^x - e^{-x})/(e^x + e^{-x})$	1
Sin	$\sin(x)$	1
Cos	$\cos(x)$	1
Arcsin	$\arcsin(x)$	1
Arccos	$\arccos(x)$	1
Square	x^2	1
Sqrt	$\sqrt{x}$	1
Relu	$[x]_+$	1
Softplus	$\log(1+e^x)$	1
Gd	$2\arctan((e^{x/2} - e^{-x/2})/(e^{x/2} + e^{-x/2}))$	1
Alf	$x/\sqrt{1+x^2}$	1
Erf	$\frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$	1
Erfc	$1 - \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$	1
Detach	$\text{de}(x)$	1

B. The algorithm of constructing a loss

The Algorithm 1 shows the process of forward constructing a computational graph. The Algorithm 2 shows the process of constructing a loss function.

Algorithm 1: Construct_CG

Input: The value set of constant nodes $Cons$, the number of constant nodes n_c, the maximum number of primitive operations n_o and the primitive operations set $\mathcal{H}$
Output: The computational graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$

The initial nodes list $\mathcal{V} = [x]$;
The initial edges list $\mathcal{E} = []$;
for $i = 1; i \leqslant n_c; i++$ **do**
  sample con_i from $Cons$;
  $\mathcal{V}$.append(con_i);
while $|\mathcal{V}| \leqslant n_o + n_c + 1$ **do**
  Enumerate the possible set $\{(Op_i, E_i)\}$ based on the $\mathcal{G}(\mathcal{V}, \mathcal{E})$;
  if $|\mathcal{V}| == n_o + n_c + 1$ **then**
    $Op = \mathcal{O}$;
    $E = [e_{v \to \mathcal{O}}$ for v in $\mathcal{V}$ if v has no successor$]$;
  else
    Sample Op and E from $\{(Op_i, E_i)\}$;
  $\mathcal{V}$.append(Op);
  foreach *e in E* **do**
    $\mathcal{E}$.append(e);
return $\mathcal{G}(\mathcal{V}, \mathcal{E})$.

Algorithm 2: Construct_LF

Input: The value set of the scale factor $Scales$, the value set of constant nodes $Cons$, the number of constant nodes n_c, the maximum number of primitive operations n_o and the primitive operations set $\mathcal{H}$
Output: The $\mathbf{\Theta} = \{s, \mathcal{G}_t(\mathcal{V}_t, \mathcal{E}_t), \mathcal{G}_n(\mathcal{V}_n, \mathcal{E}_n)\}$ in $\mathcal{L}_{gms}^{\Theta}$

sample s from $Scales$;
$\mathcal{G}_t(\mathcal{V}_t, \mathcal{E}_t)$ = Construct_CG($Cons, n_c, n_o, \mathcal{H}$);
$\mathcal{G}_n(\mathcal{V}_n, \mathcal{E}_n)$ = Construct_CG($Cons, n_c, n_o, \mathcal{H}$);
return $\{s, \mathcal{G}_t(\mathcal{V}_t, \mathcal{E}_t), \mathcal{G}_n(\mathcal{V}_n, \mathcal{E}_n)\}$.

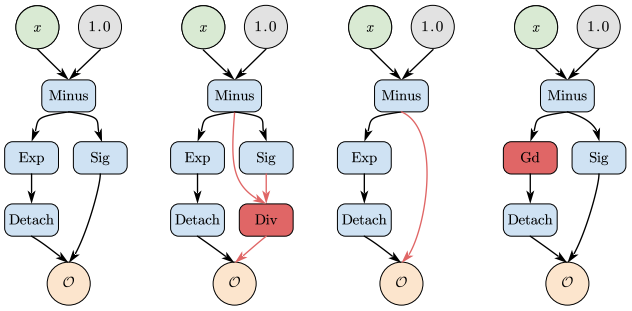
C. The details of the mutations

The in-graph mutation operations are shown as follows:

- *Insertion*. An operation randomly sampled from $\mathcal{H}$ is inserted between a randomly selected non-root node and its parent. When the selected operation's arity is 2, it would randomly select one node from all nodes generated before the selected non-root node as the other parent node. This mutation is invalid if the CG has reached the maximum number of primitive operations n_o.

- *Deletion*. Any intermediate computational node can be selected and removed. When the removed operation's arity is 2, the randomly selected parent node becomes the new parent node of the removed node' all child nodes.

- *Replacement*. The randomly selected non-root node



(a) original (b) *Insertion* (c) *Deletion* (d) *Replacement*

Figure 1. The three in-graph mutation operations. The red parts represent the changes from the original CG (a).

can be replaced by an operation randomly selected from $\mathcal{H}$. If the selected non-root node' arity is smaller than that of the selected operation, it would randomly select one node from all nodes generated before the selected non-root node as the other parent node. Otherwise, the randomly selected parent node from the parent nodes of the selected non-root node would be selected as the parent of the selected operation.

Figure 1 shows examples of these in-graph mutation operations.

The process of mutating the loss to produce the offspring is shown as Algorithm 3.

D. The details of the properies of the GMS loss

The generalized margin-based softmax loss function generally meets the following properties:

1. $t'(x) \geqslant 0, x \in [-1, 1]$;
2. $n'(x) \geqslant 0, x \in [-1, 1]$;
3. $n(x) - t(x) \geqslant 0, x \in [-1, 1]$.

Firstly, the gradients of $\mathcal{L}_{gms}$ with respect to $\cos\theta_y$ and $\cos\theta_i (i \neq y)$ can be obtained:

$$\begin{cases} \frac{\partial \mathcal{L}_{gms}}{\partial \cos\theta_y} = -s \cdot t'(\cos\theta_y) \cdot \left(1 - p_y^{t,n}\right) \\ \frac{\partial \mathcal{L}_{gms}}{\partial \cos\theta_i} = s \cdot n'(\cos\theta_i) \cdot \left(1 - p_i^{t,n}\right), i \neq y \end{cases}, \tag{18}$$

where

$$\begin{cases} p_y^{t,n} = \frac{\exp(s \cdot t(\cos\theta_y))}{\exp(s \cdot t(\cos\theta_y)) + \sum_{k \neq y} \exp(s \cdot n(\cos\theta_k))} \\ p_i^{t,n} = \frac{\exp(s \cdot n(\cos\theta_i))}{\exp(s \cdot t(\cos\theta_y)) + \sum_{k \neq y} \exp(s \cdot n(\cos\theta_k))}, i \neq y \end{cases}. \tag{19}$$

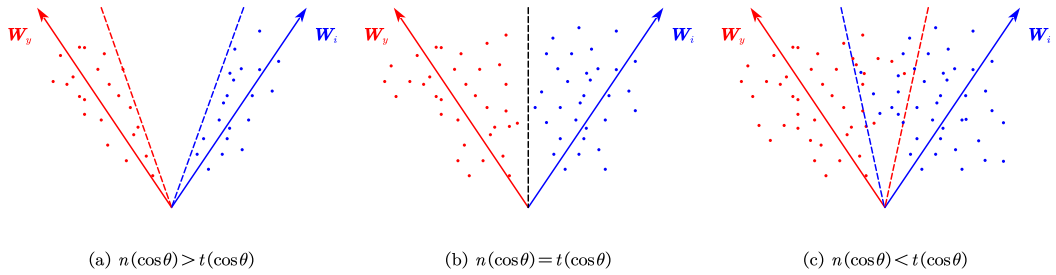
As the training progresses, the overall trend of $\cos\theta_y$ is increasing, while the overall trend of $\cos\theta_i (i \neq y)$ is decreasing. Therefore, from the overall trend, the following prop-

Algorithm 3: Produce_Offspring

Input: Two parent loss functions $\mathbf{\Theta}^{(1)}$ and $\mathbf{\Theta}^{(2)}$, the cross-graph probability p_c, the copy probability p_{copy}, the re-initialization probability p_{re}, the maximum number of mutations N_m, the parameters Δ_c and N_c for $Cons$, the parameters Δ_s and N_s for $Scales$, the number of constant nodes n_c, the maximum number of primitive operations n_o and the primitive operations set $\mathcal{H}$

Output: The offspring $\mathbf{\Theta}'$

$\mathbf{\Theta}^{(rand)}$ = Construct_LF($Scales$, $Cons$, n_c, n_o, $\mathcal{H}$);
$\mathbf{\Theta}' = \mathbf{\Theta}^{(1)}$;
if $rand < p_c$ **then** // Crossover $\mathbf{\Theta}^{(1)}$ and $\mathbf{\Theta}^{(2)}$
  if $rand < 0.5$ **then**
    $\mathbf{\Theta}' = \{s', \mathcal{G}_t^{(2)}, \mathcal{G}_n'\}$;
  else
    $\mathbf{\Theta}' = \{s', \mathcal{G}_t', \mathcal{G}_n^{(2)}\}$;
  Update the s' in $\mathbf{\Theta}'$ by Eq.(7);
$r = rand$; // Mutations on $\mathcal{G}_t'$
if $r < p_{copy}$ **then** // Directly copy the $\mathcal{G}_t'$
  pass;
else if $r < p_{re}$ **then** // Re-initialize the $\mathcal{G}_t'$ with $\mathcal{G}_t^{(rand)}$
  $\mathbf{\Theta}' = \{s', \mathcal{G}_t^{(rand)}, \mathcal{G}_n'\}$;
else
  $N = randint([1, N_m])$; // Mutate on $\mathcal{G}_t'$
  for $n = 1; n \leqslant N; n++$ **do**
    Mutate the $\{con_i|_{i=1}^{n_c}\}$ in $\mathcal{G}_t'$ by Eq.(8);
    Mutate $\mathcal{G}_t'$ with the randomly choosed operation from $\{Insertion, Deletion, Replacement\}$;
$r = rand$; // Mutations on $\mathcal{G}_n'$
if $r < p_{copy}$ **then** // Directly copy the $\mathcal{G}_n'$
  pass;
else if $r < p_{re}$ **then** // Re-initialize the $\mathcal{G}_n'$ with $\mathcal{G}_n^{(rand)}$
  $\mathbf{\Theta}' = \{s', \mathcal{G}_t', \mathcal{G}_n^{(rand)}\}$;
else
  $N = randint([1, N_m])$; // Mutate on $\mathcal{G}_n'$
  for $n = 1; n \leqslant N; n++$ **do**
    Mutate the $\{con_i|_{i=1}^{n_c}\}$ in $\mathcal{G}_n'$ by Eq.(8);
    Mutate $\mathcal{G}_n'$ with the randomly choosed operation from $\{Insertion, Deletion, Replacement\}$;
$r = rand$; // Mutations on s'
if $r < p_{copy}$ **then** // Directly copy the s'
  pass;
else if $r < p_{re}$ **then** // Re-initialize the s' with $s^{(rand)}$
  $\mathbf{\Theta}' = \{s^{(rand)}, \mathcal{G}_t', \mathcal{G}_n'\}$;
else
  $N = randint([1, N_m])$; // Mutate on s'
  for $n = 1; n \leqslant N; n++$ **do**
    Mutate the s' in $\mathbf{\Theta}'$ by Eq.(8);
return $\mathbf{\Theta}' = \{s', \mathcal{G}_t', \mathcal{G}_n'\}$.



(a) $n(\cos\theta) > t(\cos\theta)$ (b) $n(\cos\theta) = t(\cos\theta)$ (c) $n(\cos\theta) < t(\cos\theta)$

Figure 2. Different margin under different settings. The dotted line represents the decision boundary of each class.

erties can be obtained:

$$\frac{\partial \mathcal{L}_{gms}}{\partial \cos\theta_y} \leqslant 0, \frac{\partial \mathcal{L}_{gms}}{\partial \cos\theta_i} \geqslant 0, i \neq y,$$

and $p_y^{t,n} < 1$ and $p_i^{t,n} < 1$, so we can obtain the following two properties in $\mathcal{L}_{gms}$:

1. $t'(x) \geqslant 0, x \in [-1, 1]$;

2. $n'(x) \geqslant 0, x \in [-1, 1]$.

According to the above two properties, the composite function $t(\cos\theta)$ and $n(\cos\theta)$ are generally decreasing functions with $\theta \in [0, \pi]$.

For ease of illustration, we only consider the binary case (the target class is y, the non-target class is i) in $\mathcal{L}_{gms}$ to prove the third property. The decision boundary for the target class y is $t(\cos\theta_y) - n(\cos\theta_i) = 0$. When $n(\cos\theta) = t(\cos\theta)$, the decision boundary is equivalent to $\theta_y = \theta_i$, which is shown as Figure 2(b). When $n(\cos\theta) < t(\cos\theta)$, the decision boundary is equivalent to $\theta_y - \theta_i > 0$, which will cause the positive and negative samples to be indistinguishable within a certain area shown in Figure 2(c). When $n(\cos\theta) > t(\cos\theta)$, the decision boundary is equivalent to $\theta_y - \theta_i < 0$, which is shown in Figure 2(a). This situation will form a large margin between the positive and negative samples, which is conducive to learning more discriminative features. Therefore, the third property

$$n(x) - t(x) \geqslant 0, x \in [-1, 1]$$

is a property of GMS loss function.

It is difficult to test whether the continuous function $t(x)$ and $n(x)$ meet the above properties, so the f_t and f_n in Eq.(10) are used to test. We relax the requirements to meet these properties to make our search space more flexible in the specific implementation. Specifically, only the $\tau_p\%$ in f_t and f_n satisfies the above properties. This relaxation enables CircleLoss, which does not strictly satisfy the second property, to pass the protocol.

E. The algorithm of choosing the promising loss

The promising-loss chooser algorithm is shown in Algorithm 4.

Algorithm 4: Promising-Loss Chooser

Input: A loss function $\mathbf{\Theta}_i$, the current evaluated set $Eva = \{(\mathbf{\Theta}_e, p_e)|_{e=1}^{E}\}$, the predictor $\mathcal{P}$, the size of evaluated set to start predictor E_0, the interval for updating predictor ΔE, the current candidate population L, the maximum size of candidate population N_p.
Output: The loss function $\mathbf{\Theta}_o$ to be evaluated on the proxy task

```
if E < E_0 then                          // Predictor doesn't work.
    Θ_o = Θ_i;
else if E = E_0 then                     // Initialize predictor
    L.add(Θ_i);
    initialize P;
    update P with Eva;
    return None.
else                                     // Predictor works.
    L.add(Θ_i);
    if (E − E_0) mod ΔE = 0 then
        update P with Eva;
    if |L| = N_p then
        L_p = [ ];
        foreach Θ_e in L do
            L_p.append(P(Θ_e));
        e_o = argmax_{e=0,1,···,N_p−1} L_p[e];
        Θ_o = L[e_o];
        L = { };
    else
        return None.
return Θ_o.
```

F. The details of the predictor

The commonly used ResNet50 is selected as our predictor, and it needs to be adjusted according to our problem:

- The original ResNet is for two-dimensional data, but our data is one-dimensional. So the two-dimensional operations (Conv2d, BatchNorm2d, Pool2d) in the original ResNet are replaced with the one-dimensional

operations (Conv1d, BatchNorm1d, Pool1d) to obtain 1D-ResNet.

- The input feature vector $fv_{\mathbf{\Theta}}$ is reshaped to $x_{\mathbf{\Theta}} \in \mathbb{R}^{3\times N}$ to have a compact representation of the loss function. The conversion from $fv_{\mathbf{\Theta}}$ to $x_{\mathbf{\Theta}}$ is as follows:

$$\begin{cases} x_{\mathbf{\Theta}}\left[0,:\right] = fv_{\mathbf{\Theta}}\left[:N\right] \\ x_{\mathbf{\Theta}}\left[1,:\right] = fv_{\mathbf{\Theta}}\left[N:2N\right] \\ x_{\mathbf{\Theta}}\left[2,:\right] = fv_{\mathbf{\Theta}}\left[2N\right] \cdot \mathbb{1} \end{cases}$$

- The feature after the last AvgPool1d in 1D-ResNet is passed through the fully connected layers to get the final predicted performance.

Our predictor is trained with SGD optimizer, and the learning rate is fixed as 0.0035. The batch size is set to 16, and the τ in $\mathcal{L}_K$ is set to 0.01. The λ is set to 1.0, and a total of 200 epochs is used to train the predictor.

G. The implementation details of the experiments

G.1. Search configurations

Search algorithm. The population is initialized with $K = 20$ loss functions and is restricted to the most recent $P = 1000$ loss functions. When searching from scratch, the $K = 20$ loss functions in the initial population are all randomly generated as Algorithm 2 shown. When searching from prior knowledge, ten loss functions are randomly generated, and the other ten loss functions are initialized with the hand-crafted loss functions (the hyper-parameters in these functions are set by the grid search results as shown in Figure 3). The ratio of tournament selection is set as $T = 5\%$ of the current population. The condition for the search to stop is evaluating 500 models on the proxy task. Other hyper-parameters used in AutoLoss-GMS are shown in Table 2.

Proxy task. For every dataset, We use the images corresponding to 80% of the pedestrians in the original training dataset as the search training dataset, and the rest as the validation dataset. In order to further improve the search efficiency, we reduced the image resolution in the proxy task to half of the actual training task. Other configurations are consistent with train configurations.

G.2. Train configurations

The loss function with the best performance in the search process is retrained on the original training dataset.

Data preprocessing. In ResNet50 and OSNet, we resize each image to 256×128. For MGN, we keep it consistent with the original paper, and the image size of 384×128 is used. Then random cropping, random horizontal flipping and random erasing are used for data augmentation.

Table 2. The hyper-parameters used in AutoLoss-GMS. CG: computational graph. POS: produce the offspring. LRP: loss-rejection protocol. ECS: equivalence-check strategy. PLC: promising-loss chooser.

CG	n_c	2
	n_o	10
	Δ_s	0.5
	N_s	20
	Δ_c	0.05
	N_c	20
POS	p_c	0.6
	p_{copy}	0.1
	p_{re}	0.4
	N_m	3
LRP	τ_p	80
	τ_{toy}	0.9
	B	64
ECS	N	257
	Γ	13
PLC	E_0	100
	ΔE	100
	N_p	10

Data sampling. We adopt PK sampling trategy with $P = 16$ and $K = 4$.

Training. We use the SGD optimizer with a momentum of 0.9 and weight decay of 5e-4 for training the above three models. The above models are all trained for 140 epochs in total, and the warmup learning rate is used. We spend the first 20 epochs to increase the learning rate linearly from 0.0001 to 0.01, during which the weights of the backbone are fixed. The learning rate is decayed to 0.001 and 0.0001 at 60th and 100th epochs.

Testing. The test-time flipping is utilized during testing, and the cosine similarity is utilized as the measurement.

H. The impact of hyper-parameters in hand-crafted loss functions and AutoLoss-GMS-A

Since AutoLoss-GMS-A is searched on Market-1501 with ResNet50, we directly report the performance of AutoLoss-GMS-A on Market-1501, as shown in Figure 3(a),(b). The hyper-parameters in other hand-crafted loss functions are obtained by grid search. It can be seen that the performance of our AutoLoss-GMS-A on the Market-1501 is significantly better than these hand-crafted loss functions.

When AutoLoss-GMS-A is directly transferred on the CUHK03 dataset without any hyper-parameter adjustment, its performance is still slightly better (see Table 4 in main text for specific values) than other hand-crafted loss functions after hyper-parameter adjustment, which is shown in Figure 3(c),(d). To fast adjust hyper-parameters in our

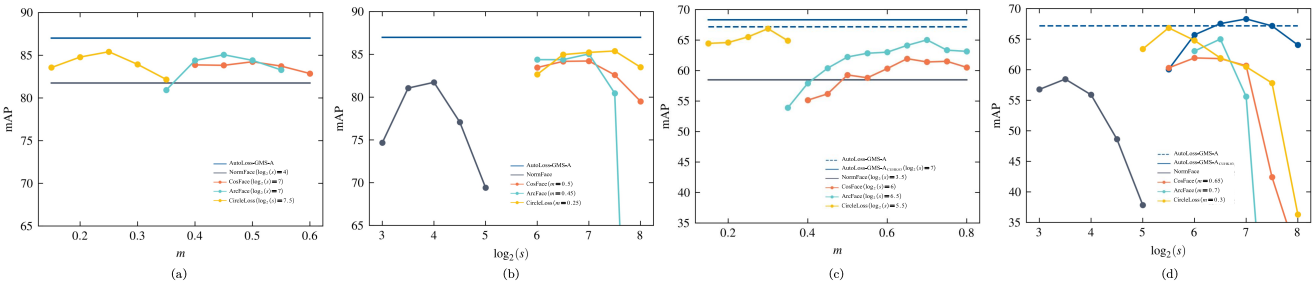


Figure 3. Impact of hyper-parameters in hand-crafted loss functions and the searched AutoLoss-GMS-A. (a) and (b) are the results of ResNet50 on Market-1501. (c) and (d) are the results of ResNet50 on CUHK03.

Table 3. The searched loss functions. PIP: predefined initial populations. Each loss is searched on a certain dataset using certain model with PIP or not.

Searched Loss	Model	Dataset	PIP	$t(x)$	$n(x)$	$\log_2(s)$
AutoLoss-GMS-Zero	ResNet50	Market-1501	✗	$(0.22+e^{\sqrt{0.22}})x+\arcsin(0.22)^2$	$x+0.85$	5.5
AutoLoss-GMS-A	ResNet50	Market-1501	✓	$\mathrm{de}(1-x)(x-0.7)$	$[\mathrm{Gd}(x-\mathrm{Sig}(0.25))]_+$	7.5
AutoLoss-GMS-B	ResNet50	CUHK03	✓	$\mathrm{de}(1.3-x)(x-1.0)$	$0.35x-0.35^2$	6.0
AutoLoss-GMS-C	OSNet	Market-1501	✓	$(x-0.84)(0.95-x)$	$[\mathrm{de}(\arcsin(x))]_+(x-0.5)+0.05$	7.5
AutoLoss-GMS-D	MGN	Market-1501	✓	$x+0.15$	$x+0.2$	4.0

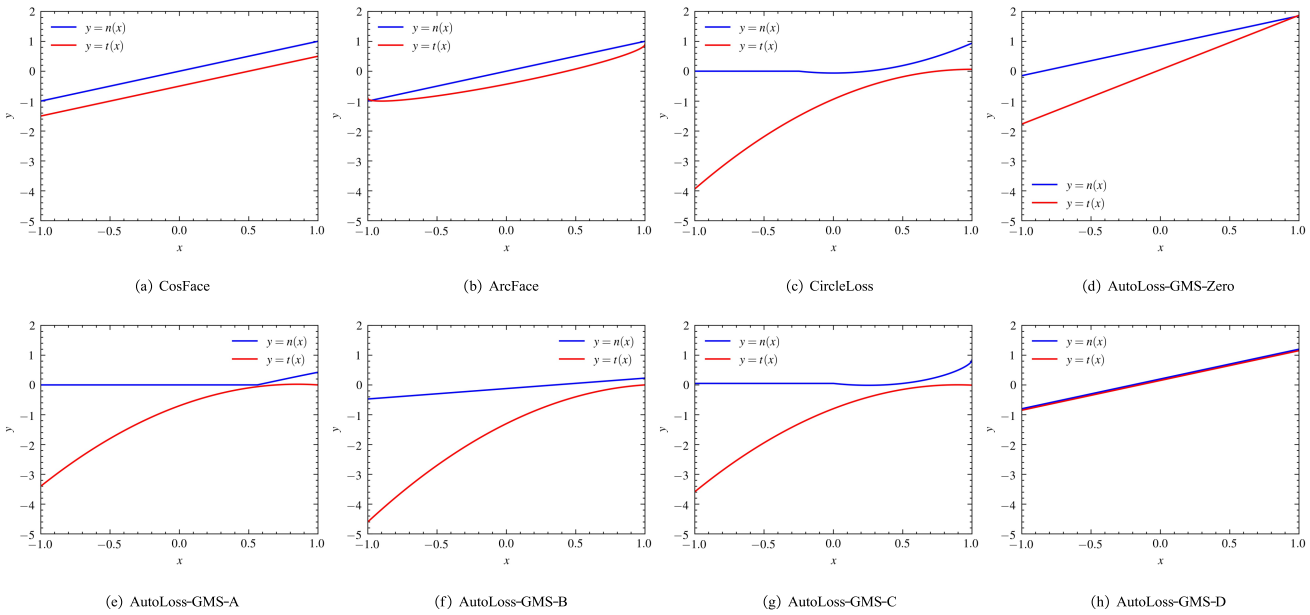


Figure 4. The $t(x)$ and $n(x)$ of the hand-crafted loss functions and the searched loss functions.

searched loss functions, only the s is to be fine-tuned, which is shown in Figure 3(d). Our fine-tuned AutoLoss-GMS-A is significantly better than these hand-crafted loss functions on the CUHK03 dataset, too.

I. The analysis of the searched loss functions

The specific expressions of the searched loss functions are shown in Table 3, and the $t(x)$ and $n(x)$ of the hand-crafted loss functions and the searched loss functions are shown in Figure 4. First of all, the $t(x)$ and $n(x)$ in AutoLoss-GMS-Zero and AutoLoss-GMS-D are both linear functions. Due to the lack of prior knowledge, AutoLoss-GMS-Zero will fall into this simple form of a local solution. For AutoLoss-GMS-D, because the triplet loss function is involved in the search process, it is possible that this simple form can bring good results.

Table 4. The performance of the OSNet(-IBN,-AIN) with AutoLoss-GMS-C under the cross-domain setting.

Method	Market-1501→MSMT17				MSMT17→Market-1501			
	mAP	Rank-1	Rank-5	Rank-10	mAP	Rank-1	Rank-5	Rank-10
OSNet	3.4	10.3	17.2	21.6	40.1	67.0	80.9	86.0
OSNet+AutoLoss-GMS-C	8.4(↑ 5.0)	24.2(↑ 13.9)	35.6(↑ 18.4)	41.7(↑ 20.1)	47.6(↑ 7.5)	74.6(↑ 7.6)	86.3(↑ 5.4)	90.6(↑ 4.6)
OSNet-IBN	7.7	22.8	33.5	39.0	37.2	66.5	81.5	86.8
OSNet-IBN+AutoLoss-GMS-C	9.7(↑ 2.0)	27.0(↑ 4.2)	39.1(↑ 5.6)	44.8(↑ 5.8)	47.1(↑ 9.9)	73.7(↑ 7.2)	86.3(↑ 4.8)	90.5(↑ 3.7)
OSNet-AIN	8.2	23.5	34.5	40.2	43.3	70.1	84.1	88.6
OSNet-AIN+AutoLoss-GMS-C	10.6(↑ 2.4)	29.5(↑ 6.0)	41.7(↑ 7.2)	47.7(↑ 7.5)	46.4(↑ 3.1)	73.9(↑ 3.8)	86.8(↑ 2.7)	90.4(↑ 1.8)

The forms of AutoLoss-GMS-A, AutoLoss-GMS-B and AutoLoss-GMS-C are similar to CircleLoss, but there are still some differences. Specifically, none of the three searched $t(x)$ and $n(x)$ have the Detach operation simultaneously. Although the $t(x)$ of the three searched loss functions and the $t(x)$ of CircleLoss are similar to quadratic functions, their $n(x)$s are far from CircleLoss's, which are difficult to be designed by hand.

J. The performance of the searched loss function under the cross-domain evaluation

In Sec.4.3.3, we have discussed the transferability of the searched losses across datasets and models under the same-domain setting, and the results are shown in Table 5 of the main text. Here we will discuss the direct cross-domain evaluation, and we present the performance of OSNet(-IBN,-AIN) with the searched AutoLoss-GMS-C under the cross-domain setting in Table 4. All results of OSNet, OSNet-IBN and OSNet-AIN are obtained according to their published network weights.

In the setting of Market-1501→MSMT17, the AutoLoss-GMS-C we searched can stably improve the performance of these three models. And when MSMT17→Market-1501, OSNet +AutoLoss-GMS-C can already achieve an mAP of 47.6%, which is better than the hand-designed OSNet-IBN and the OSNet-AIN that took much effort to search. Since our AutoLoss-GMS-C was originally searched on the OSNet model, it may not be the most suitable for the other two models, so the performance of these two models under the blessing of AutoLoss-GMS-C is not as good as OSNet +AutoLoss-GMS-C. Searching for another loss function in the cross-domain setting for OSNet-AIN will likely lead to better performance. All the above experimental results show that searching for a reasonable loss function can effectively improve the performance of direct cross-domain evaluation.

If the target domain dataset can be used in the training process like the current popular cross-domain methods, SpCL provides a good framework, whose core loss is suitable for being searched with our AutoLoss-GMS.

Table 5. The performance on CUB-200-2011 dataset with BN-Inception network.

Method	R@1	R@2	R@4	R@8
MS	65.7	77.0	86.3	91.2
CircleLoss	66.7	77.4	86.2	91.2
CirceLoss(Our Impl.)	65.6	76.1	84.9	91.1
Searched(Ours)	66.4	76.5	85.2	91.1

K. The extension on other task

Our method can be easily extended to other tasks, like face recognition (FR), fine-grained image retrieval (FGIR) etc. The loss we searched on ReID can also achieve competitive performance when directly transferred to the FR task (about 0.3% improvement over ArcFace with ResNet34 on CFP-FP dataset). For FGIR, **all tricks** we have proposed can be applied to searching for the $s, t(x), n(x)$ in pair-wise form of CircleLoss:

$$\mathcal{L}_{circle-pw} = \log\left[1 + \sum_{j=1}^{L} \exp\left(s \cdot n\left(S_n^j\right)\right) \sum_{i=1}^{K} \exp\left(-s \cdot t\left(S_p^i\right)\right)\right], \tag{20}$$

where K is the number of the within-class scores, and L is the number of the between-class scores. Based on the framework of $\mathcal{L}_{circle-pw}$, we searched on the CUB-200-2011 dataset with the recall@1 target. The searched $t(x) = \text{de}(1.4 - x)(x - 0.5)$, $n(x) = 0.3 + (0.05 + x[\sin(\text{de}(x))]_+)^2$,$\log_2(s) = 6.5$.

Table 1 shows our searched loss on the CUB-200-2011 dataset. Our search target is recall@1, so our loss is the best at R@1 under the same training configuration[1]. Besides, the current SOTA ProxyNCA++ is also fully applicable to our search space in FGIR, and the MS with some appropriate adjustments is also applicable.

[1]Reproducing the results of CircleLoss is challenging, and we can only achieve the best possible performance based on MS's open source code.