Supplement Material : Pareto Self-Supervised Training for Few-Shot Learning

Zhengyu Chen[1,2,3], Jixie Ge[1,2], Heshen Zhan[1,2], Siteng Huang[1,2], Donglin Wang[1,2*]
[1] Machine Intelligence Lab (MiLAB), AI Division, School of Engineering, Westlake University
[2] Institute of Advanced Technology, Westlake Institute for Advanced Study
[3] College of Computer Science & Technology, Zhejiang University
{chenzhengyu,gejixie,zhanheshen,huangsiteng,wangdonglin}@westlake.edu.cn

A. Proofs and derivations

A.1. Proof of Lemma

Lemma 2:*If $\theta^*_{\pi 0}$ achieves the best overall performance, i.e. $min_\theta \sum_{m=1}^M L_m(\theta) = \sum_{m=1}^M L_m(\theta^*_{\pi 0})$, and a better performance of main task can be achieved in θ', i.e., $L_1(\theta') < L_1(\theta^*_{\pi 0})$, we will have $\rho(\theta') < \rho(\theta^*_{\pi 0})$.*

Proof: We will have

$$\sum_{m=2}^{M} L_m(\theta') > \sum_{m=2}^{M} L_m(\theta^*_{\pi 0})$$

if $L_1(\theta') < L_1(\theta^*_{\pi 0})$ holds.

We proof it by contradiction. If there is a θ' satisfying

$$L_1(\theta') < L_1(\theta^*_{\pi 0})$$

and

$$\sum_{m=2}^{M} L_m(\theta') < \sum_{m=2}^{M} L_m(\theta^*_{\pi 0})$$

at the same time, we will have

$$\sum_{m=1}^{M} L_m(\theta') < \sum_{m=1}^{M} L_m(\theta^*_{\pi 0}) = min_\theta \sum_{m=1}^{M} L_m(\theta).$$

However, there does not exist any θ' satisfying

$$\sum_{m=1}^{M} L_m(\theta') < min_\theta \sum_{m=1}^{M} L_m(\theta)$$

for $min_\theta \sum_{m=1}^M L_m(\theta)$ is the minimum of $\sum_{m=1}^M L_m(\theta)$. A contradiction is found.

Based on the above results, we have

$$\begin{aligned}\rho(\theta') &= \frac{L_1(\theta')}{\sum_{m=2}^M L_m(\theta')} \\ &< \frac{L_1(\theta^*_{\pi 0})}{\sum_{m=2}^M L_m(\theta')} \\ &< \frac{L_1(\theta^*_{\pi 0}))}{\sum_{m=2}^M L_m(\theta^*_{\pi 0}))} \\ &= \rho(\theta^*_{\pi 0}).\end{aligned}$$

End the proof.

A.2. Illustrating preference vectors in two-tasks scenario

Let the unit vector u_0 be the direction vector of $\rho(\theta) = \rho(\theta_0)$ in two-task scenario, let $u_0 = (\cos \pi_0, \sin \pi_0)$, where $\cos \pi_0 = \frac{e_1 \mathcal{L}(\theta^*_{\pi 0})}{\left\| T_2 \mathcal{L}(\theta^*_{\pi 0}) \right\|_2}$ and $T_m = I - e^{mm}$. The e^{mm} is a single-entry matrix, i.e. the mth element in the mth column is one and the rest elements are zero, e_1 is a single-entry vector, where the first element is one and rest are zero. I is an identity matrix. A set of unit preference vectors $\{\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_K\}$, for the preference vector u_i can be defined as:

$$\begin{aligned} u_i &= (\cos \pi_i, \sin \pi_i), \\ s.t. \quad \pi_i &= \tfrac{i}{K}\left(\tfrac{\pi}{2} - \pi_0\right) + \pi_0, i = 1, \ldots, K. \end{aligned} \tag{1}$$

Derivation and proof: Instead of dividing the objective space into two parts by this hyperplane, we divide it by some vectors in two-dimensional space, for finding some special vectors in low-dimensional subspace is much easier than dividing the high-dimensional Euclidean space. We call these vectors *preference vectors* and take the three-dimensional space Oxyz as an example to illustrate our theory. If we want to divide the three-dimensional space Oxyz, we can first divide Oxyz through the vertical planes perpendicular to xOy, yOz and zOx, which are two-dimensional

*Corresponding author.

spaces. These vertical planes can be determined by its intersects with xOy, yOz and zOx, and the intersects can be determined by its direction vectors. Thus, dividing multi-dimensional space is equivalent to dividing multiple two-dimensional spaces, and we only need to find the direction vectors.

Considering problems in two-dimensional space is equivalent to consider problems in two-task scenario. Remind that, two-task scenario does not mean that we only have two tasks, we still have M tasks, but we pair the first task with the others, then we will have M-1 pairs of tasks. We try to divide the space with these M-1 pairs of tasks. Let $u_0 = (\cos \pi_0, \sin \pi_0)$, we obtain u_0 by calculating π_0.

As the best overall performance can be achieved in $\theta^*_{\pi 0}$, we know that the hyperplane also passes $\theta^*_{\pi 0}$. With the solution $\theta^*_{\pi 0}$, we have the optimal loss vector:

$$\mathcal{L}(\theta^*_{\pi 0}) = [\mathcal{L}_1(\theta^*_{\pi 0}), \mathcal{L}_2(\theta^*_{\pi 0}), \cdots, \mathcal{L}_M(\theta^*_{\pi 0})]^{\mathrm{T}}.$$

We first find the projection of the optimal loss vector, then calculate the direction vector of it as one preference vector. Take the second dimension as an example. The projection of the optimal loss vector in the second dimension is:

$$\mathcal{L}(\theta^*_{\pi 0}) = [\mathcal{L}_1(\theta^*_{\pi 0}), 0, \mathcal{L}_3(\theta^*_{\pi 0}), \cdots, \mathcal{L}_M(\theta^*_{\pi 0})]^{\mathrm{T}},$$

i.e., $T_2\mathcal{L}(\theta^*_{\pi 0})$. Cosine of the angle between $T_2\mathcal{L}(\theta^*_{\pi 0})$ and e_1 is

$$\cos \pi_0 = \frac{e_1 T_2 \mathcal{L}(\theta^*_{\pi 0})}{\|e_1\|_2 \|T_2\mathcal{L}(\theta^*_{\pi 0})\|_2} = \frac{e_1 \mathcal{L}(\theta^*_{\pi 0})}{\|T_2\mathcal{L}(\theta^*_{\pi 0})\|_2}.$$

Remove the subspace which satisfying $\cos\theta > \cos\pi_0$ and decompose the remaining part into K parts based on the angles equally. As a result, we have

$$\pi_i = \frac{i}{K}\left(\frac{\pi}{2} - \pi_0\right) + \pi_0, i = 1, \ldots, K.$$

Elements in the set of vectors $\{\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_K\}$, i.e. preference vectors, will satisfy:

$$u_i = (\cos \pi_i, \sin \pi_i), i = 1, \ldots, K.$$

Preference vectors found by this method can evenly divide the space we want. Set a proper K, we can decide the number of the divided spaces. A small K will cause the accumulation of residual error, and a big K will cost too much time and computing resources.

In our experiments, we mainly use one auxiliary task, so things are a little different. We don't need to calculate the projection of the optimal loss vector, since at this time the hyperplane is already a straight line in objective space. We straightly calculate the cosine of the angle between $\mathcal{L}(\theta^*_{\pi 0})$ and e_1, i.e.,

$$\cos \pi_0 = \frac{e_1 \mathcal{L}(\theta^*_{\pi 0})}{\|e_1\|_2 \|\mathcal{L}(\theta^*_{\pi 0})\|_2} = \frac{e_1 \mathcal{L}(\theta^*_{\pi 0})}{\|\mathcal{L}(\theta^*_{\pi 0})\|_2}$$

A.3. Further discussion of preference vectors in multiple tasks

In the above section, we provide an illustration of preference vectors in two-task scenario, now we will discuss preference vectors in another perspective, with another method of finding preference vectors.

With the hyperplane $\rho(\theta) = \rho(\theta^*_{\pi 0})$ we can divide the objective space into two parts, and we only want the part which satisfies $\rho(\theta) \leq \rho(\theta^*_{\pi 0})$. Noting that $\rho(\theta^*_{\pi 0})$ is a constant greater than zero, and every objective function is non-negative, the inequality

$$\rho(\theta) = \frac{L_1(\theta)}{\sum_{m=2}^{M} L_m(\theta)} \leq \rho(\theta^*_{\pi 0})$$

is equivalent to

$$\frac{L_1(\theta)}{\rho(\theta^*_{\pi 0})} \leq \sum_{m=2}^{M} L_m(\theta).$$

For objective function $L_m(\theta)$, there are two possible situations: 1) $\frac{L_1(\theta)}{(M-1)\rho(\theta^*_{\pi 0})} \leq L_m(\theta)$; 2) $\frac{L_1(\theta)}{(M-1)\rho(\theta^*_{\pi 0})} > L_m(\theta)$, $(m = 2, ..., M)$. We set two index sets:

$$U = \{m | \frac{L_1(\theta)}{(M-1)\rho(\theta^*_{\pi 0})} \leq L_m(\theta), m \in \{2, ..., M\}\},$$

$$V = \{m | \frac{L_1(\theta)}{(M-1)\rho(\theta^*_{\pi 0})} > L_m(\theta), m \in \{2, ..., M\}\}.$$

Clearly, we have $U \bigcup V = \{2, ..., M\}$ and $U \bigcap V = \emptyset$. U should be a non-empty subset of $\{2, ..., M\}$. If $U = \emptyset$, we will have $\frac{L_1(\theta)}{\rho(\theta^*_{\pi 0})} > \sum_{m=2}^{M} L_m(\theta)$, this goes against our needs.

Noting that every (U, V) can determine a partition of $\{2, ..., M\}$ and a subspace of the objective space, we can divide the whole space into several subspaces by setting different (U, V). Let U run through every non-empty subset of $\{2, ..., M\}$ to obtain all pairs of possible (U, V), then add (U, V) as constraints to the optimization problem. We can formulate the optimization problem as

$$\begin{aligned} &\min_\theta \mathcal{L}(\theta) = (\mathcal{L}_1(\theta), \mathcal{L}_2(\theta), \cdots, \mathcal{L}_M(\theta))^{\mathrm{T}}, \\ &\text{s.t. } \tfrac{L_1(\theta)}{\rho(\theta^*_{\pi 0})} \leq \textstyle\sum_{u \in U} L_u(\theta) + \sum_{v \in V} L_v(\theta), \\ &\quad \tfrac{L_1(\theta)}{(M-1)\rho(\theta^*_{\pi 0})} \leq L_m(\theta), \text{for } m \in U, \\ &\quad \tfrac{L_1(\theta)}{(M-1)\rho(\theta^*_{\pi 0})} > L_m(\theta), \text{for } m \in V, \\ &\quad U \text{ is a non-empty subset of } \{2, ..., M\}, \\ &\quad V = \{2, ..., M\} - U. \end{aligned} \tag{2}$$

When U running through every non-empty subset of $\{2, ..., M\}$, the algorithm exploring the whole preference region.

We take the situation of $\rho(\theta^*_{\pi 0}) = \frac{1}{M-1}$ as an example to illustrate the implication of (U, V). If $\rho(\theta^*_{\pi 0}) = \frac{1}{M-1}$, then $m \in U$ means $L_1(\theta) \leq L_m(\theta)$, i.e. the performance of task m is worse than the performance of main task, $m \in V$ means $L_1(\theta) > L_m(\theta)$, i.e. the performance of task m is better than the performance of main task. In conclusion, (U, V) determines the relative quality between main task and auxiliary tasks, different qualities mean different subspaces. Although $\rho(\theta^*_{\pi 0})$ may not be equal to $\frac{1}{M-1}$ in many situations, (U, V) can still reveal the relative quality between main task and auxiliary tasks. Thus, we can divide the objective spaces based on (U, V).

To find the preference vectors, we only need to solve the equation group:

$$\frac{L_1(\theta)}{(M-1)\rho(\theta^*_{\pi 0})} = L_m(\theta), m = 2, ..., M.$$

Take the solutions of the above equation group as preference vectors, we can divide the objective space. However, we don't need to calculate the preference vectors in practice, make sure that U can run through every non-empty subset of $\{2, ..., M\}$ and solve the optimization problem in Eq. 2 is enough.

We can hardly decide the number of preference vectors in this method. When we just have a few tasks, preference vectors found by this method can not divide the space to too many parts, so we'd better choose the two-task scenario to divide the space. For example, when $M = 2$, just like our experiments, $\{2\}$ has only one non-empty subset, and the only preference vector is the direction vector of $\rho(\theta) = \rho(\theta^*_{\pi 0})$, so we chose the two-task scenario. However, when we have plenty of tasks, it's wise to find the preference vectors in multiple tasks scenario.

A.4. Derivation of subproblem corresponding to the preference vectors u_i and u_{i+1}

Given the preference vectors u_i and u_{i+1}, our next step is to add constraints to the optimization problem so that the Pareto solutions we found can lay inside between u_i and u_{i+1}. Still, we take the second dimension as an example. Firstly, we get the projection vector, then use $\cos \pi_0 = \frac{e_1 \mathcal{L}(\theta^*_{\pi 0})}{\|T_2 \mathcal{L}(\theta^*_{\pi 0})\|_2}$ to measure the angle between projection vector and e_1. Let $\cos(a, b)$ be the cosine of angle between a and b. We simply add the constraints:

$$\cos(e_1, u_i) \leq \cos(\pi_0) \leq \cos(e_1, u_i + 1)$$

to our optimization problem. e_1 and u_i are both unit vectors, so $\cos(e_1, u_i) = u_i e_1^T$, the same as $\cos(e_1, u_{i+1})$. We formulate the problem as:

$$\begin{aligned} &\min_\theta \mathcal{L}(\theta) = (\mathcal{L}_1(\theta), \mathcal{L}_2(\theta), \cdots, \mathcal{L}_M(\theta))^{\mathrm{T}}, \\ &\text{s.t. } u_i {e_1}^T \leq \frac{e_1 \mathcal{L}(\theta)}{\|T_2 \mathcal{L}(\theta)\|_2} \leq u_{i+1} {e_1}^T. \end{aligned}$$

A.5. Rewriting the optimization problem in Eq. 8 in its dual form

The optimization problem in Eq. 8 is

$$\begin{aligned} (d_t, \alpha_t) = \arg\min_{d \in R^M, \alpha \in R} \alpha + \tfrac{1}{2}\|d\|^2, \\ \text{s.t.} \quad \nabla \mathcal{L}_m(\theta_t)^T d \leq \alpha, i = 1, \ldots, M. \\ \nabla \mathcal{Q}_k(\theta_t)^T d \leq \alpha, k \in \mathcal{K}_\epsilon(\theta_t), \\ \nabla \mathcal{R}_j(\theta_t)^T d \leq \alpha, j \in \mathcal{J}_\epsilon(\theta_t). \end{aligned} \tag{3}$$

It's easy to see that Eq. 8 is a convex optimization problem. To obtain the dual form of this problem, we define a Lagrangian function as

$$\begin{aligned} g(d, \alpha, \omega_m, \beta_k, \gamma_j) = \alpha + \frac{1}{2}\|d\|^2 \\ + \sum_{m=1}^{M} \omega_m (\nabla \mathcal{L}_m(\theta_t) - \alpha) \\ + \sum_{k \in \mathcal{K}_\epsilon(\theta_t)} \beta_k (\nabla \mathcal{Q}_k(\theta_t) - \alpha) \\ + \sum_{j \in \mathcal{J}_\epsilon(\theta_t)} \gamma_j (\nabla \mathcal{R}_j(\theta_t)) - \alpha) \end{aligned}$$

Calculate the partial derivatives and solve the equation group below:

$$\begin{aligned} \frac{\partial g(d, \alpha, \omega_m, \beta_k, \gamma_j)}{\partial d} = 0 \\ \frac{\partial g(d, \alpha, \omega_m, \beta_k, \gamma_j)}{\partial \alpha} = 0 \end{aligned}$$

These two equations are equivalent to:

$$\begin{aligned} d_t(\omega_m, \beta_k, \gamma_j) + \sum_{m=1}^{M} \omega_m \nabla \mathcal{L}_m(\theta_t) \\ + \sum_{k \in \mathcal{K}_\epsilon(\theta_t)} \beta_k \nabla \mathcal{Q}_k(\theta_t) \\ + \sum_{j \in \mathcal{J}_\epsilon(\theta_t)} \gamma_j \nabla \mathcal{R}_j(\theta_t) = 0 \end{aligned}$$

and

$$1 - (\sum_{i=1}^{M} \omega_m + \sum_{k \in \mathcal{K}_\epsilon(\theta_t)} \beta_k + \sum_{j \in \mathcal{J}_\epsilon(\theta_t)} \gamma_j) = 0.$$

It's easy to see that $d_t(\omega_m, \beta_k, \gamma_j)$ is a function of every ω_m, β_k, and γ_j. We replace the d in Eq. 8 with $d_t(\omega_m, \beta_k, \gamma_j)$, and add the constraint $\sum_{i=1}^{M} \omega_m +$

$\sum_{k\in\mathcal{K}_\epsilon(\theta_t)}\beta_k+\sum_{j\in\mathcal{J}_\epsilon(\theta_t)}\gamma_j=1$ to the optimization problem so that we could omit α in Eq. 8. In the end, we formulate the optimization problem as

$$\begin{aligned}&\max_{\omega_m,\beta_k,\gamma_j} -\tfrac{1}{2}\left\|d_t\left(\omega_m,\beta_k,\gamma_j\right)\right\|_2^2,\\ &\text{s.t. } \sum_{i=1}^{M}\omega_m+\sum_{k\in\mathcal{K}_\epsilon(\theta_t)}\beta_k+\sum_{j\in\mathcal{J}_\epsilon(\theta_t)}\gamma_j=1,\\ &\forall m=1,\ldots,M,\forall k\in\mathcal{K}_\epsilon(\theta_t),\forall j\in\mathcal{J}_\epsilon(\theta_t),\\ &\omega_m\geq 0,\beta_k\geq 0,\gamma_j\geq 0.\end{aligned} \tag{4}$$

A.6. Finding the update direction in Pareto exploration

Assuming that Pareto front is so smooth that every point on this front can be passed through by several smooth curves on this front. Let θ^* be a Pareto solution and $\theta_{(x)}: (-\epsilon,\epsilon)\to\mathbb{R}^n$ is a smooth curve on Pareto front which pass through θ^* in 0, i.e., $\theta_{(0)}=\theta^*$. Our aim is to find some other Pareto optimal solutions, i.e., find some other points on the Pareto front. Finding points on the Pareto front directly is too costly, so we first try to find some approximations of Pareto solutions, then optimize these approximations to find the exact Pareto solutions.

We need to calculate the first-order approximation, i.e., calculate $\theta_1=\theta_{(0)}+\eta\frac{d\theta}{dx}(0)$. As mentioned in Lemma 3, for every Pareto solution, we have

$$\sum_{m=1}^{M}\lambda_m\nabla\mathcal{L}_m(\theta^*)=\mathbf{0},$$

where $\boldsymbol{\lambda}\in\mathbb{R}^M$, $\lambda_m\geq 0$, $\sum_{m=1}^{M}\lambda_i=1$. Thus, for every point on the curve $\theta_{(x)}$, we have

$$\sum_{m=1}^{M}\lambda_m(\theta_{(x)})\nabla\mathcal{L}_m(\theta_{(x)})=\mathbf{0}.$$

λ_m will change if $\theta_{(x)}$ changes, so we write λ_m as a function of $\theta_{(x)}$, i.e. $\lambda_m(\theta_{(x)})$. Differentiate both sides of the equation, we will have

$$\boldsymbol{H}\left(\theta^*\right)\frac{d\theta}{dx}(0)=\nabla\mathcal{L}\left(\theta^*\right)^\top\boldsymbol{\beta},$$

where $\boldsymbol{H}\left(\theta^*\right)=\sum_{m=1}^{M}\lambda_i\nabla^2\mathcal{L}_m\left(\theta^*\right)$ and $\beta\in\mathbb{R}^M$. We can obtain $\frac{d\theta}{dx}(0)$ by this method.

However, solving a large-scale linear equation group is very costly in practice. As suggested by Sener and Koltun[11], we get λ by solving a convex optimization problem below:

$$\begin{aligned}&\min_{\boldsymbol{\lambda}} \quad \left\|\sum_{m=1}^{M}\lambda_m\nabla\mathcal{L}_m\left(\theta_{(x)}\right)\right\|_2\\ &\text{s.t.} \quad \boldsymbol{\lambda}\geq\mathbf{0},\quad \sum_{m=1}^{M}\lambda_m=1\end{aligned}$$

Get $\boldsymbol{\lambda}$ for $\nabla\mathcal{L}\left(\theta_{(x)}\right)$ and calculate $\frac{d\theta}{dx}(0)$, then we will be able to have the approximation θ_1. Optimize θ_1, we can have another Pareto solution.

Notation	Description
I_d, J_d, L_d	No. of users, items and criteria
$\mathbf{R}_d=[r_{d,ijl}]_{I_d\times J_d\times L_d}$	User-item-criterion ratings
$\mathbf{U}_d$, $\mathbf{V}_d$, $\mathbf{C}_d$	Latent factor matrix
$\mathscr{G}$	Core tensor of Tucker
$\mathbf{u}_{d,i}$, $1\leq i\leq I_d$	Latent factor vector of users
I_d, J_d, L_d	No. of users, items and criteria
$\mathbf{R}_d=[r_{d,ijl}]_{I_d\times J_d\times L_d}$	User-item-criterion ratings
$\mathbf{U}_d$, $\mathbf{V}_d$, $\mathbf{C}_d$	Latent factor matrix
$\mathscr{G}$	Core tensor of Tucker
$\mathbf{u}_{d,i}$, $1\leq i\leq I_d$	Latent factor vector of users

Table 1: Summary of primary notations.

B. Additional implementation details

B.1. Network architectures

Conv-4-64. It consists of 4 convolutional blocks each implemented with a 3×3 convolutional layer with 64 channels followed by BatchNorm + ReLU + 2×2 max-pooling units. In the MiniImageNet experiments for which the image size is 84×84 pixels, its output feature map has size $5\times 5\times 64$ and is flattened into a final 1600-dimensional feature vector. For the CIFAR-FS experiments, the image size is 32×32 pixels, the output feature map has size $2\times 2\times 64$ and is flattened into a 256-dimensional feature vector

WRN-28-10. It is a Wide Residual Network with 28 convolutional layers and width factor 10. The 12 residual layers of this architecture are grouped into 3 residual blocks (4 residual layers per block). In the MiniImageNet experiments, the network gets as input images of size 80×80 (rescaled from 84×84), and during feature extraction each residual block downsamples by a factor of 2 the processed feature maps. Therefore, the output feature map has size $10\times 10\times 640$ which, after global average pooling, creates a 640-dimensional feature vector. In the CIFAR-FS experiments, the input images have size 32×32 and during feature extraction only the last two residual blocks downsample the processed feature maps. Therefore, in the CIFAR-FS experiments, the output feature map has size $8\times 8\times 640$ which again after global average pooling creates a 640-dimensional feature vector.

Rotation prediction network. This network gets as input the output feature maps of θ_1 and is implemented as a convnet. More specifically, for the Conv-4-64 and Conv-4-512 feature extractor architectures (regardless of the dataset), The network consists of two 3×3 convolutional layers with BatchNorm + ReLU units, followed by a fully connected classification layer. For Conv-4-64, those two convolutional layers have 128 and 256 feature channels respectively, while for Conv-4-512 both convolutional layers have 512 feature channels. In the WRN-28-10 case, The

network consists of a 4-residual-layer residual block that actually replicates the last (3rd) residual block of WRN-28-10. This residual block is followed by global average pooling plus a fully connected classification layer.

Relative patch location network. Given two patches, The networkgets the concatenation of their feature vectors extracted with θ_1 as input, and forwards it to two fully connected layers. The single hidden layer, which includes BatchNorm + ReLU units, has 256, 1024, and 1280 channels for the Conv-4-64, Conv-4-512, and WRN-28-10 architectures respectively.

B.2. Incorporating self-supervision during training

Here we provide more implementation details regarding how we incorporate self-supervision during the training stage.

Training with rotation prediction self-supervision. During training for each image of a mini-batch we create its 4 rotated copies and apply to them the rotation prediction task (i.e., L_2 loss). When training the object classifier with rotation augmentation (e.g., CC-based models) the object classification task (i.e., L_1 loss) is applied to all rotated versions of the images. Otherwise, only the upright images (i.e., the 0 degrees images) are used for the object classification task. Note that in the PN-based models, we apply the rotation task to both the support and the query images of a training episode, and also we do not use rotation augmentation for the object classification task.

Training with relative patch location self-supervision. In this case during training each mini-batch includes two types of visual data, images and patches. Similar to [2], in order to create patches, an image is: (1) resized to 96×96 pixels (from 84×84), (2) converted to grayscale with probability 0.66, and then (3) divided into 9 regions of size 32×32 with a 3×3 regular grid. From each 32×32 sized region we (4) randomly sample a 24×24 patch, and then (5) normalize the pixels of the patch individually to have zero mean and unit standard deviation. The object classification task is applied to the image data of the mini-batch while the relative patch location task to the patch data of the mini-batch. Also, as already explained, we also apply an extra auxiliary object classification loss to the patch data.

B.3. Training routine for training stage

To optimize the training loss we use mini-batch SGD optimizer with momentum 0.9 and weight decay $5e-4$. In the MiniImageNet and CIFAR-FS experiments, we train the models for 60 epochs (each with 1000 SGD iterations), starting with a learning rate of 0.1 which is decreased by a factor of 10 every 20 epochs. The mini-batch sizes were cross-validated on the validation split. For instance, the models based on CC and Conv-4-64, Conv-4-512, or WRN-28-10 architectures are trained with mini-batch sizes equal to 128, 128, or 64 respectively. Finally, we perform early stopping w.r.t. the few-shot classification accuracy on the validation novel classes (for the CC-based models we use the 1-shot classification accuracy).

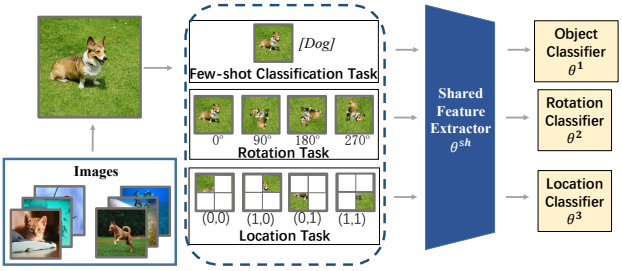

Figure 1: The framework used in our Pareto Self-Supervised Training.

C. More details of Few-Shot Auxiliary Learning

The main component of all few-shot algorithms is a feature extractor θ^{sh}. As shown in Fig. 1, given an image x, the feature extractor will output a d-dimensional feature $\theta^{sh}(x)$ to different classifier, and the object classifier θ^1 will output the label of image. In this work, we use Cosine Classifiers (CC) [3] as object classifier.

Cosine Classifiers. In CC few-shot learning, the first stage trains the feature extractor θ^{sh} together with a cosine-similarity based classifier on the (standard) supervised task of classifying the base classes. Denoting $\theta_1 = [\mathbf{w}_1, \ldots, \mathbf{w}_{N_b}]$ the matrix of the d-dimensional classification weight vectors, the normalized score for an input image $\mathbf{x}$ is

$$C^j\left(\theta^{sh}(\mathbf{x}); \theta_1\right) = \text{softmax}_j\left[\gamma \cos\left(\theta^{sh}(\mathbf{x}), \mathbf{w}_i\right)_{i \in Y_b}\right] \quad (5)$$

where $cos(,)$ is the cosine operation between two vectors, and the scalar γ is the inverse temperature parameter of the softmax operator, j is the class. The training stage aims at minimizing w.r.t. θ^{sh} and θ_1 the negative log-likelihood loss:

$$\mathcal{L}_1\left(\theta^{sh}, \theta_1; D_b\right) = \mathop{\mathbb{E}}_{(\mathbf{x},y)\sim D_b}\left[-\log C^y\left(\theta^{sh}(\mathbf{x}); \theta_1\right)\right] \quad (6)$$

One of the reasons for using the cosine-similarity based classifier instead of the standard dot-product based one, is that the former learns feature extractors that reduce intra-class variations and thus can generalize better on novel classes. The weight vectors w_j in θ_1 can be interpreted as learned prototypes for the base classes, to which input image features are compared for classification. The second stage boils down to computing one representative feature w_j for each new class by simple averaging of asso-

ciated K samples in D_n, and to define the final classifier $C\left(.;[\mathbf{w}_1 \cdots \mathbf{w}_{N_n}]\right)$.

Self-Supervised Auxiliary loss We incorporate self-supervision to a few-shot learning algorithm by adding an auxiliary self-supervised loss during its training stage. More formally, let $\mathcal{L}_2\left(\theta^{sh}, \theta^2; X_b\right)$ be the self-supervised loss applied to the set $X_b = \{\mathbf{x} \mid (\mathbf{x}, y) \in D_b\}$ of training examples in D_b deprived of their class labels. The loss $\mathcal{L}_2\left(\theta^{sh}, \theta^2; X_b\right)$ is a function of the parameters θ^{sh} of the feature extractor and of the parameters θ^2 of a network only dedicated to the self-supervised task. The first training stage of few-shot auxiliary learning is

$$\min_{\theta^{sh}, \theta^1, \theta^2} \omega_1 \mathcal{L}_1\left(\theta^{sh}, \theta^1; D_b\right) + \omega_2 L_2\left(\theta^{sh}, \theta^2; X_b\right) \quad (7)$$

where the positive hyperparameter ω_1 and ω_2 controls the importance of the classification term and self-supervised term. The framework of the approach is provided in Figure 1. For the self-supervised loss, we consider two-task in the present work: predicting the rotation incurred by an image [4], which is simple and readily incorporated into a few-shot learning algorithm; predicting the relative location of two patches from the same image [1], a seminal task in self-supervised learning. In a recent study, both methods have been shown to achieve state-of-the-art results [7].

Image rotations. In this task, the convnet must recognize among four possible 2D rotations in $\mathcal{R} = \{0°, 90°, 180°, 270°\}$ the one applied to an image (see Figure 1). Specifically, given an image x, we first create its four rotated copies $\{\mathbf{x}^r \mid r \in \mathcal{R}\}$, where x^r is the image x rotated by r degrees. Based on the features $\theta^{sh}\left(\mathbf{x}^r\right)$ extracted from such a rotated image, the new network θ^2 attempts to predict the rotation class r. Accordingly, the self-supervised loss

$$\mathcal{L}_2(\theta^{sh}, \theta^2; X) = \mathop{\mathbb{E}}_{\mathbf{x} \sim X}\left[\sum_{\forall r \in \mathcal{R}} -\log \theta^2\left(\theta^{sh}\left(\mathbf{x}^r\right)\right)\right] \quad (8)$$

where X is the original training set of non-rotated images and $\theta^2(\cdot)$ is the predicted normalized score for rotation r. Intuitively, in order to do well for this task the model should reduce the bias towards up-right oriented images, typical for ImageNet-like datasets, and learn more diverse features to disentangle classes in the low-data regime.

Relative patch location. Here, we create random pairs of patches from an image and then predict, among eight possible positions, the location of the second patch w.r.t. to the first, e.g., "on the left and above" or "on the right and below". Specifically, given an image x, we first divide it into 9 regions over a $3 * 3$ grid and sample a patch within each region. Lets denote x^0 the central image patch, and $x^1 \cdots x^8$ its eight neighbors lexicographically ordered. We compute the representation of each patch and then generate patch feature pairs $\left(\theta^{sh}\left(x^0\right), \theta^{sh}\left(x^p\right)\right)$ by concatenation.

Figure 2: Sample images from MultiMNIST. Above each image are the labels of the upper-left (L) and lower-right (R) items.

We train a fully-connected network $\theta^3(\cdot, \cdot)$ to predict the position of x^p from each pair. The self-supervised loss of this task is defined as:

$$\mathcal{L}_3(\theta^{sh}, \theta^3; X) = \mathop{\mathbb{E}}_{x \sim X}\left[\sum_{p=1}^{8} -\log \theta^3\left(\theta^{sh}\left(x^0\right), \theta^{sh}\left(x^p\right)\right)\right] \quad (9)$$

where X is a set of images and θ^3 is the predicted normalized score for the relative location p.

D. More details of Pareto exploration

D.1. MultiMNIST Setup

We first generated the full MultiMNIST dataset and picked a subset of 2048 images, downsampled from 28×28 to 14×14, as our MultiMNIST Subset example. The two objectives are the cross entropy losses of classifying the top-left and bottom-right digits evaluated on all 2048 images. Regarding the classifier, we used a modified network, which has 1500 parameters. Our modified network starts with a convolutional layer with 10 channels, a 5×5 kernel, and a stride of 2 pixels, followed by a 2×2 max pooling layer. Next, the results are fed into a fully connected layer of size 20×10 and then sent to two fully connected layers, one for each task. We use ReLU as the nonlinear function in the network. Essentially, this synthetic example attempts to use a small network to overfit 2048 images. To generate the Pareto front, we ran BFGS to optimize $w_1 f_1 + w_2 f_2$ with $w_1 = 0, 0.01, 0.02, \cdots, 1$ from the same random initial guess, which generated a list of 101 solutions $\boldsymbol{x}_0^*, \boldsymbol{x}_1^*, \boldsymbol{x}_2^*, \cdots, \boldsymbol{x}_{101}^*$. We then linearly interpolated $\boldsymbol{f}\left(\boldsymbol{x}_i^*\right), i = 0, 1, 2, \cdots, 100$ and treat the resulting piecewise linear spline as the (empirical) Pareto front.

Dataset and Task Description. We followed [10] to generate MultiMNIST. We first created 36×36 images by placing two 28×28 images from MNIST in the upper-left and lower-right corner with a random shift of up to 2 pixels in each direction. The synthesized images were then resized to 28×28 and normalized with a mean of 0.1307 and a standard deviation of 0.3081. No data augmentation

was used for training or testing. Following [8, 9], we built MultiMNIST from MNIST as shown in the Figure 2. Each dataset has 60,000 training images and 10,000 test images. The objectives are the cross entropy losses of classifying the upper-left and lower right items in the image.

Network Architecture. The backbone network is a modified LeNet (LeCun et al., 1998). Our network starts from two convolutional layers with a 5×5 kernel and a stride of 1 pixel. The two layers have 10 and 20 channels respectively. A fully connected layer of 50 channels appends the convolutional layers, which is then followed by two 10-channel fully connected layers, one for each task. We add a 2×2 max pooling layer right after each convolutional layer and use ReLU as the nonlinear function. The network contains 22,350 trainable parameters.

Training Training We trained all baselines for 30 epochs of SGD. We used 256 as our mini-batch size and set the momentum to 0.9. The learning rate started from 0.01 and decayed with a cosine annealing scheduler.

D.2. Details of exploration on the tangent plane.

Find gradient on the tangent plane at θ^*. Once a Pareto solution θ^* is found, we explore its local Pareto set by spawning new points θ_t.

Lemma 3 [6]: If θ^* is Pareto optimal, there will be a $\boldsymbol{\lambda} \in \mathbb{R}^M$ such that $\lambda_m \geq 0$, $\sum_{m=1}^M \lambda_i = 1$, and $\sum_{m=1}^M \lambda_m \nabla \mathcal{L}_m(\theta^*) = \mathbf{0}$.

Proposition [5]: Assuming that $\mathcal{L}(\theta^*)$ is smooth and θ^* is Pareto optimal, consider any smooth curve $\theta_{(x)} : (-\epsilon, \epsilon) \to \mathbb{R}^n$ in the Pareto set and passing θ^* at $x = 0$, i.e., $\theta_{(0)} = \theta^*$, then $\exists \beta \in \mathbb{R}^M$ such that:

$$\boldsymbol{H}(\theta^*) \frac{d\theta}{dx}(0) = \nabla \mathcal{L}(\theta^*)^\top \boldsymbol{\beta} \tag{10}$$

where $\boldsymbol{H}(\theta^*) = \sum_{m=1}^M \lambda_i \nabla^2 \mathcal{L}_m(\theta^*)$.

We use $\frac{d\theta}{dx}(0)$ as the update direction and calculate $\theta_1 = \theta_{(0)} + \eta \frac{d\theta}{dx}(0)$.

Solving such problem requires an efficient matrix solver. Similar to [9], we use Krylov subspace iteration methods, because they are matrix-free and iterative solvers, allowing us to solve the system without complete Hessians and terminate with intermediate results. In our experiment, we choose to use the minimal residual method (MINRES), a classic Krylov subspace method designed for symmetric indefinite matrices. Note that early termination in MINRES still returns meaningful results because the residual error is guaranteed to decrease monotonically with iterations. To summarize, the efficiency of our exploration algorithm comes from two sources: exploration on the tangent plane and early termination from a matrix-free, iterative solver. The time cost of getting one tangent direction is $O(kn)$, which scales linearly to the network size.

References

[1] Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE international conference on computer vision*, pages 1422–1430, 2015. 6

[2] Spyros Gidaris, Andrei Bursuc, Nikos Komodakis, Patrick Pérez, and Matthieu Cord. Boosting few-shot visual learning with self-supervision. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 8058–8067, 2019. 5

[3] Spyros Gidaris and Nikos Komodakis. Dynamic few-shot visual learning without forgetting. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 4367–4375, 2018. 5

[4] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. In *International Conference on Learning Representations*, 2018. 6

[5] Claus Hillermeier. Generalized homotopy approach to multiobjective optimization. *Journal of Optimization Theory and Applications*, 110(3):557–583, 2001. 7

[6] Claus Hillermeier et al. *Nonlinear multiobjective optimization: a generalized homotopy approach*, volume 135. Springer Science & Business Media, 2001. 7

[7] Alexander Kolesnikov, Xiaohua Zhai, and Lucas Beyer. Revisiting self-supervised visual representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1920–1929, 2019. 6

[8] Xi Lin, Hui-Ling Zhen, Zhenhua Li, Qingfu Zhang, and Sam Kwong. Pareto multi-task learning. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 12037–12047, 2019. 7

[9] Pingchuan Ma, Tao Du, and Wojciech Matusik. Efficient continuous pareto exploration in multi-task learning. In *International Conference on Machine Learning*, 2020. 7

[10] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in neural information processing systems*, pages 3856–3866, 2017. 6

[11] Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. In *Advances in Neural Information Processing Systems*, pages 527–538, 2018. 4