AutoLR: Layer-wise Pruning and Auto-tuning of Learning Rates in Fine-tuning of Deep Networks

Younmgin Ro[1,2], **Jin Young Choi**[1*]
[1] Department of ECE, ASRI, Seoul National University, Korea
[2] Samsung SDS, Korea
youngmin.ro@samsung.com, jychoi@snu.ac.kr

Abstract

Existing fine-tuning methods use a single learning rate over all layers. In this paper, first, we discuss that trends of layer-wise weight variations by fine-tuning using a single learning rate do not match the well-known notion that lower-level layers extract general features and higher-level layers extract specific features. Based on our discussion, we propose an algorithm that improves fine-tuning performance and reduces network complexity through layer-wise pruning and auto-tuning of layer-wise learning rates. The proposed algorithm has verified the effectiveness by achieving state-of-the-art performance on the image retrieval benchmark datasets (CUB-200, Cars-196, Stanford online product, and Inshop). Code is available at https://github.com/youngminPIL/AutoLR.

Introduction

The ability to collect large amounts of data has allowed deep learning to advance dramatically over the last decade. In particular, deep neural network architectures have evolved by targeting large datasets such as ImageNet (2009). However, in actual computer vision applications, large amounts of data such as those contained ImageNet, cannot be easily obtained. Thus, for applications such as image retrieval or fine-grained classification, for which only small datasets are available, the use of pre-trained networks has become essential (Fu, Zheng, and Mei 2017; Zheng et al. 2017; Suh et al. 2019; Guo et al. 2019; Wang et al. 2019). To utilize the pre-trained network for a target task, we use a fine-tuning scheme that initially takes pre-trained weights and adjusts them in whole or in part.

The performance of the fine-tuning depends highly on the following factors: similarity between the source and target tasks (Azizpour et al. 2015; Cui et al. 2018), choice of deep network model (Kornblith, Shlens, and Le 2018), and tuning strategy (Tajbakhsh et al. 2016; Guo et al. 2019; Ro et al. 2019). From among these factors, our paper focuses on enhancement of tuning strategy. Many efforts have been made to enhance the capability of tuning the network for a given target task. Partial tuning (Tajbakhsh et al. 2016) has been proposed to tune only the higher-level layers. Weight-reverting methods were proposed in some studies (Guo et al. 2019; Ro et al. 2019), where the tuned weights are reverted to the initial (pre-trained) weights during fine-tuning. In addition, some researchers adjusted the learning rate (LR) periodically to ensure efficient learning by adjusting it to follow a triangular waveform (Smith 2017), and even proposing an exponentially decaying shape of LR (Loshchilov and Hutter 2016). However, most existing fine-tuning methods adopt a single LR regardless of layers.

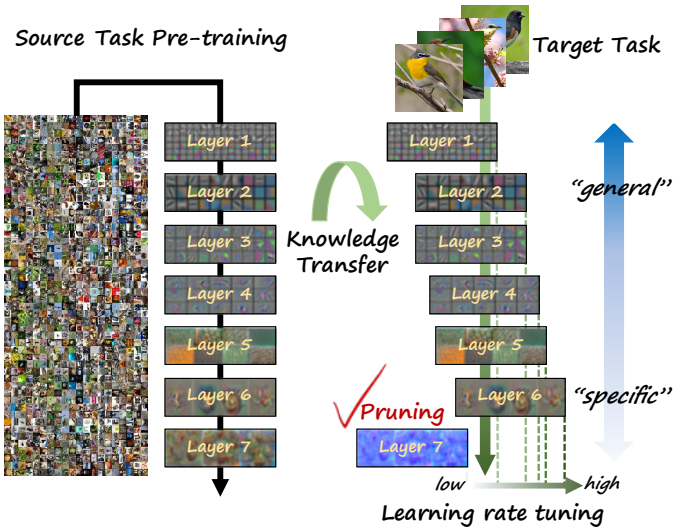


Figure 1: The conceptual figure of the proposed algorithm: Conducting layer-wise pruning and auto-tuning of layer-wise learning rates on the target task according to role of each layer.

In this paper, instead of using a single LR, we propose an algorithm for layer-wise auto-tuning of LRs where the LR in each layer is automatically tuned according to its role. In addition, we propose a layer-wise pruning algorithm that removes layers according to the usefulness of each pre-trained layer to the new task as illustrated in Figure 1. Our work is inspired by our observation that the trends of layer-wise weight variations by conventional fine-tuning using a single LR, contradict previous studies (Yosinski et al. 2014; Zeiler and Fergus 2014), which claim that low-level layers extract

*Corresponding author

general features while high-level layers extract specific features. Based on our observation, we establish two hypotheses and validate them empirically via preliminary experiments. Following the validation of the hypotheses, we develop algorithms for layer-wise pruning and auto-tuning of layer-wise LRs.

Through ablation experiments on four image retrieval benchmark datasets, it is verified that the proposed method consistently improve the retrieval performance. By providing visualization results, we help to understand the mechanism of the proposed method. In addition, the superiority of our method as a fine-tuning strategy is demonstrated through comparative experiments with the existing LR setting algorithms. Lastly, with regard to four benchmark datasets of image retrieval, our method outperforms the existing state-of-the-art methods.

Related Work

Fine-tuning is a kind of transfer learning and is used for tuning of pre-trained parametric model. In fine-tuning, the similarity between the source task and the target task is also important as in transfer learning. Regarding the studies on the similarity, Azizpour *et al*. (2015) has suggested factors to transfer knowledge well considering the similarity between target and source tasks. And Cui *et al*. (2018) proposed a method to promote knowledge transfer using similarity between multiple tasks. In the computer vision fields, ImageNet (Deng et al. 2009) dataset has been widely used for a source task. Hence lots of target tasks have been handled by using deep network models pre-trained by using ImageNet. In Kornblith *et al*. (2018), it has been claimed that there is a positive correlation between ImageNet and most target tasks regardless of deep networks. This implies that ImageNet includes huge amounts of image data covering most image-related tasks. Hence in our paper, we will adopt ResNet-50 pre-trained using ImageNet to validate our algorithm.

In our paper, we are motivated from the role of each layer in fine-tuning a deep network. Regarding the studies on the roles of hidden layers, Yosinski *et al*. (2014) conducted empirical study to quantify the degree of generality/specificity of each layer in deep networks. And Zeiler *et al*. (2014) visualized features in hidden layers to analyze generality/specificity in the layers. Through the studies (Yosinski et al. 2014; Zeiler and Fergus 2014), they claim that the low-level layers extract general features and the high-level layers extract specific features in a deep network. This claim provides insight into our algorithm.

There have been similar works to ours that exploits the role of each layer. In (Tajbakhsh et al. 2016), Tajbakhsh *et al*.have shown that tuning only a few high-level layers is more effective than tuning all layers. Guo *et al*. (2019) proposed an auxiliary policy network that decides whether to use the pre-trained weights or fine-tune them in layer-wise manner for each instance. Ro *et al*. (2019) proposed a roll-back scheme that returns a part of weights to their pre-trained state to improve performance. However, these works adopt the fixed learning rate settings unlike our method. In addition, the approach of 'learning to learn' (Li et al. 2017; Ren et al. 2018) aims to find the optimal update of network weights. This approach directly finds the optimal initialization and update of the network weights unlike our approach to efficiently update the pre-trained weights via auto-tuning layer-wise learning rate.

Similarly to ours, there are studies that adjust the learning rate periodically during learning process. Smith (2017) suggested a way to adjust the learning rate in a triangular form that linearly reduces learning rate and then grows it again. Similarly Loshchilov *et al*. (2016) has also proposed an exponentially decaying and restarting the learning rate for a certain period of time. However, the existing methods for on-line tuning of the learning rate use single learning rate over all the layers. In our work, we aims to develop an algorithm to auto-tune the layer-wise learning rates depending on the role of each layer.

Proposed Method

The purpose of this paper is to devise an algorithm that automatically sets the appropriate learning rate (LR) for each layer after removing layers that do not contribute to the learning results in the fine-tuning process. We first discuss our findings observed in the conventional fine-tuning process and then analyze the effects of the layer-wise pruning or layer-wise tuning of LRs through preliminary experiments. Then, we derive an approximate relationship between the weight variation of each layer and the LR. Finally, we propose an algorithm for the layer-wise pruning and auto-tuning of LR in each layer.

Layer-wise Weight Variations in Fine-tuning

This paper is inspired by previous studies (Yosinski et al. 2014; Zeiler and Fergus 2014; Azizpour et al. 2015). These studies have demonstrated empirically and qualitatively that lower-level layers extract general features and higher-level layers extract specific features. Based on these results, we build two hypotheses:

Hypothesis 1: *The pre-trained high-level layers may not be helpful to a new target task because they are specific to the source task.*

Hypothesis 2: *The weight variations of pre-trained low-level layers would be small because they are generally valid for most tasks, whereas those in high-level layers would be large because they should adopt themselves to a new task specifically.*

To clarify these hypotheses and support our idea, we define the weight variation of k-th layers as

$$v_t^k = \frac{1}{n_k} \|\Delta w_t^k\|, \tag{1}$$

where $\Delta w_t^k = w_t^k - w_{t-1}^k$ denotes the amount of weight changes in the k-th layer during t-th epoch, and n_k is the number of weights in the k-th layer.

First, we investigate the trends of layer-wise weight variations in the conventional fine-tuning process. The investigation has been conducted on a retrieval dataset CUB-200-2011 (Wah et al. 2011) with ResNet-50 (He et al. 2016). Figure 2 shows the trends of weight variation between two consecutive epochs. 'Layer' denotes a residual block in Resnet-50, where 'Layer 1' indicates the lowest layer to the input

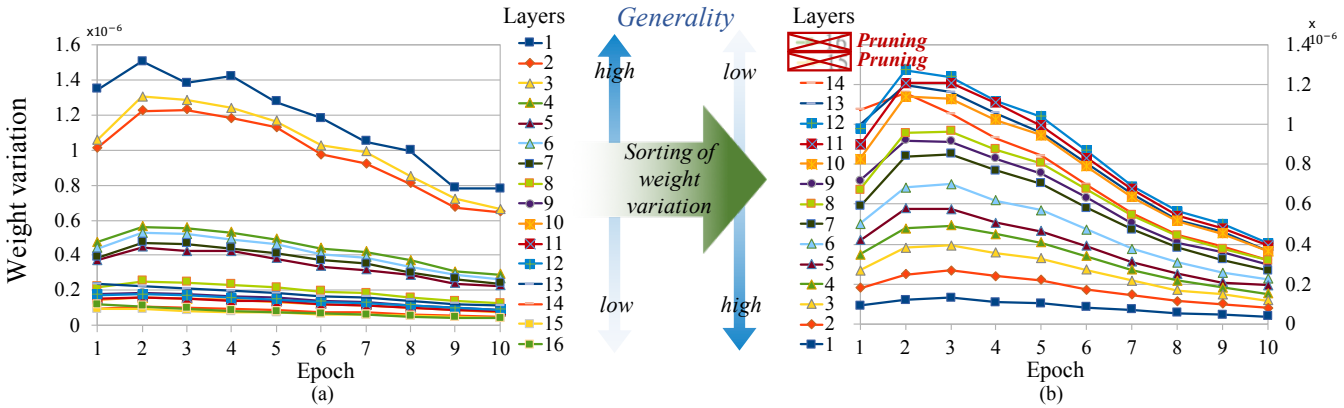


Figure 2: The trend of the weight variation between two adjacent epochs. In the result of the conventional fine-tuning of single LR (a), it is observed that the variations of the higher-level layers are small and the lower-level layers are large. (b) The trend after pruning two highest-level layers and tuning layer-wise LRs for sorting of weight variation. After pruning and layer-wise LR tuning, the performance improved significantly (In Table 1).

layer and thus 'Layer 16' becomes the highest layer. From the results shown in Figure 2-(a), which shows the trends of layer-wise weight variations by the conventional fine-tuning with single LR, we have observed two interesting points described in the following two paragraphs. These observations support the key ideas of our paper.

The first point is that the weight variations in high-level layers are too small from fine-tuning, as shown in Figure 2-(a). Based on Hypothesis 1, the high-level layer with small weight variation may not contribute to the new task learning. To support this claim, we have conducted a layer-wise pruning experiments in which the high-level layers were removed one by one from the highest layer. The performance variations by the pruning are shown in Table 1. Interestingly, the pruning Layer 15, 16 improves the performance, which implies that Layers 15 and 16 might not be helpful to the target task. The pruning of layers until Layer 14 does degrades the performance, which means the layers below Layer 14 are helpful to the new task. This preliminary result supports Hypothesis 1 and can be motivation for our simple but efficient layer-wise pruning scheme.

The second point is that the trends in weight variations by traditional fine-tuning do not match Hypothesis 2, as shown in Figure 2-(a). This implies the traditional fine-tuning destroys the general features of the pre-trained network by large variations in low-level layers and cannot promote the high-level layer to adapt itself to the new task. We believe this result is due to the LR assigned by a single value regardless of layers. To verify this, we have conducted a preliminary experiment by adjusting layer-wise LRs empirically. The layer-wise adjustment of LRs gives a significant improvement, as shown in rows 5, 6, and 7 of Table 1. Figure 2-(b) shows the order of layer-wise weight variations for row 7 (Pruning 15, 16*). Interestingly, we can see the order for row 7 meets Hypothesis 2. This result validates Hypothesis 2 that can be utilized to design an auto-tuning scheme for layer-wise LRs.

Based on Hypotheses 1 and 2, we aim to develop an algorithm for the layer-wise pruning and auto tuning of LR (AutoLR). In particular, the AutoLR is designed to achieve the order of layer-wise weight variations in every epoch that meets Hypothesis 2, that is,

$$v_t^1 \leq v_t^2 \leq \cdots \leq v_t^K. \tag{2}$$

	R@1	R@2	R@4	R@8
Original	63.49	75.03	84.00	90.58
Pruning 16	66.95	77.63	86.39	92.00
Pruning 15, 16	67.00	78.49	86.85	92.07
Pruning 14, 15, 16	51.00	64.16	75.44	85.94
Pruning 15, 16†	67.15	78.61	86.83	92.08
Pruning 15, 16‡	67.29	79.15	87.36	92.66
Pruning 15, 16*	**68.33**	**79.88**	**87.69**	**92.71**

Table 1: The Recall@K score results of preliminary experiment for layer-wise pruning and layer-wise adjusting of LRs. In †, the LRs in Layers 1,2,3 were reduced to 1/10. In ‡, the LRs in Layer 13,14 were increased by 10 times. In ⋆, the LR in all layers were empirically adjusted to meet Hypothesis 2.

Weight Variation and LR

Our key idea is to control the weight variation of each layer by tuning the LR of each layer in every epoch. In this section, we derive a relationship between the LR and the weight variation, which will be used in the auto-tuning scheme for LRs according to the layers.

The weights in the k-th layer are updated during l-th iteration for a randomly chosen mini-batch by the stochastic gradient descent rule with the momentum as follows:

$$\Delta w_{t,l}^k = \rho \Delta w_{t,l-1}^k - \eta^k \nabla \mathcal{L}(w_{t,l-1}^k), \tag{3}$$

where $\Delta w_{t,l}^k = w_{t,l}^k - w_{t,l-1}^k$, $\mathcal{L}(w_{t,l-1}^k)$ denotes the loss for $w_{t,l-1}^k$, ρ is a momentum coefficient, l is the iteration index, t is the epoch index, and η^k is LR of k-th layer. Note that η^k is adjusted every epoch in our AutoLR scheme, hence η^k is constant for all iterations during an epoch. Therefore, by applying (3) recursively, we can obtain

$$\Delta w_{t,l}^k = -\eta^k [\nabla \mathcal{L}(w_{t,l-1}^k) + \rho \nabla \mathcal{L}(w_{t,l-2}^k) + \rho^2 \nabla \mathcal{L}(w_{t,l-3}^k) + \rho^3 \nabla \mathcal{L}(w_{t,l-4}^k) + \cdots]. \quad (4)$$

Define

$$\nabla \mathcal{L}_{acc}(w_t^k) = \sum_{l=1}^{L} [\nabla \mathcal{L}(w_{t,l-1}^k) + \rho \nabla \mathcal{L}(w_{t,l-2}^k) + \rho^2 \nabla \mathcal{L}(w_{t,l-3}^k) + \rho^3 \nabla \mathcal{L}(w_{t,l-4}^k) + \cdots], \quad (5)$$

where L is the number of mini-batches and $\nabla \mathcal{L}(w_{t,l}^k) = 0$ for $l \leq 0$. Then

$$\Delta w_t^k = \sum_{l=1}^{L} \Delta w_{t,l}^k = -\eta^k \nabla \mathcal{L}_{acc}(w_t^k), \quad (6)$$

which leads to

$$\|\Delta w_t^k\| = \eta^k \|\nabla \mathcal{L}_{acc}(w_t^k)\|. \quad (7)$$

From (1) and (7),

$$v_t^k = \frac{\eta^k}{n_k} \|\nabla \mathcal{L}_{acc}(w_t^k)\|. \quad (8)$$

In the next section, this equation is used for the AutoLR scheme. Note that $\|\nabla \mathcal{L}_{acc}(w_t^k)\|$ does not explicitly depend on η_k as shown in (5). Actually, the variation $\|\nabla \mathcal{L}_{acc}(w_t^k)\|$ has been observed to be negligible (see Appendix B of the supplementary material[1]). In AutoLR scheme, hence, we ignore the variation of $\|\nabla \mathcal{L}_{acc}(w_t^k)\|$ during iterations in each epoch. We apply the equation (8) to our AutoLR scheme repeatedly until we get the goal in (2).

Layer-wise Pruning and Auto-tuning of LR

Based on the results analyzed above, we propose an algorithm to prune layers that are not helpful to the target task. Then we also propose an algorithm (AutoLR) to automatically adjust the LR of each layer so that the order of the weight variation size of each layer is consistent with (2). AutoLR is divided into two parts: setting the target weight variation and tuning by adjusting the LR accordingly.

Layer-wise pruning rule Before applying layer-wise AutoLR, low-contributed high-level layers are pruned by the procedure in Algorithm 1. In the pruning procedure, we fine-tune a network on a target task using the traditional fine-tuning scheme with a layer-wise fixed LR.

[1]https://github.com/youngminPIL/AutoLR

Algorithm 1 Layer-wise Pruning

Notation:
- η : a single learning rate for all layers
- *network* : network with weight parameters
- *score* : performance of current network
- *best score* : best performance of previous networks

Pruning:
1: *network* ← *pre-trained network*
2: Set η to the all layers of network
3: Set *best score* $= 0$
4: Fine-tune *network*
5: *score* ← performance of *network*
6: **while** *score* $\geq$ *best score* **do**
7: *best score* ← *score*
8: *network* ← *pre-trained network*
9: Prune the highest-level layer of *network*
10: Fine-tune *network*
11: *score* ← performance of *network*

Setting target weight variations We need to renew the target weight variations that satisfy the goal in (2) in each epoch of the learning process. We design two formulas to set the renewed target weight variation depending on *sorting quality* that denotes how well the current weight variations satisfy the goal in (2). To this end, we measure the *sorting quality* defined as follows:

$$sorting\ quality = 1 - \frac{2}{K^2} \sum_{k=1}^{K} |k - \sigma(v_t^k)|, \quad (9)$$

where $\sigma(v_t^k)$ is a function that maps v_t^k to its ranking in ascending order, and $\frac{2}{K^2}$ is the scale factor that enforces the sorting quality to be scaled within 0 and 1.

The initial target weight variation is set as follows:

$$d_t = \frac{1}{K-1} \left(\beta \max_{1 \leq k \leq K} v_t^{(k)} - \alpha \min_{1 \leq k \leq K} v_t^{(k)} \right) \quad (10)$$

$$\bar{v}_t^{(k)} \leftarrow \min_{1 \leq i \leq K} v_t^{(k)} + (k-1) d_t,\ k = 1, \cdots, K. \quad (11)$$

where the α and β are the hyper-parameter to set a range of target weight variation.

If the *sorting quality* is below a pre-defined threshold τ_s, the target weight variation requires partial modifications. To prevent the renewed target sorting from shifting to one side, the modification starts from the center ($\check{k}$) and moves to both ends as follows:

For $k = \check{k}, \check{k}+1, \cdots, K$

$$\bar{v}_t^{(k)} \leftarrow \begin{cases} v_t^{(k)} & k = \check{k} \text{ or } \bar{v}_t^{(k-1)} \leq v_t^{(k)} \\ \bar{v}_t^{(k-1)} + d_t & \text{otherwise,} \end{cases} \quad (12)$$

For $k = \check{k}-1, \check{k}-2, \cdots, 1$

$$\bar{v}_t^{(k)} \leftarrow \begin{cases} v_t^{(k)} & \bar{v}_t^{(k+1)} \geq v_t^{(k)} \\ \bar{v}_t^{(k+1)} - d_t & \text{otherwise.} \end{cases} \quad (13)$$

Renewing LR To get a new LR $\bar{\eta}_t^k$ which produces the renewed target weight variance $\bar{v}_t^{(k)}$. Utilizing (8), we can

Algorithm 2 AutoLR: Auto-tuning of learning rates

Notation:
η^k : learning rate of k-th block
$\bar{\eta}^k$: target learning rate of k-th block
v_t^k : weight variation of k-th block in t-th epoch
$\bar{v}_t^k$: target weight variation of k-th block in t-th epoch
network : network with tuned weights
trial-network : trial network for tuning of η^k

Auto-tuning:
Initialize η^k with single learning rate
Initialize *network* with *pre-trained network*
Set *sorting quality* $= 0$.
for epoch $\leftarrow 1$ to T **do**
  trial-network $\leftarrow$ *network*
  while *sorting quality* $\leq \tau_s$ **do**
    Fine-tune *trial-network* with η^k for target dataset
    Calculate *weight variation* in (1)
    Calculate *sorting quality* in (9)
    if *sorting quality* $> \tau_s$ **then**
      network $\leftarrow$ *trial-network*
    else
      if epoch $== 1$ **then**
        Renew $\bar{v}_t^k$ by (11)
      else
        Renew $\bar{v}_t^k$ by (12) and (13)
      Renew $\bar{\eta}^k$ by (15)
      $\eta^k \leftarrow \bar{\eta}^k$
      trial-network $\leftarrow$ *network*
  epoch++

set $\bar{v}_t^k \approx \frac{\bar{\eta}^k}{n_k}\|\nabla\mathcal{L}_{acc}(w_t^k)\|$, which leads to

$$\bar{v}_t^k - v_t^k \approx \frac{\bar{\eta}_t^k - \eta_t^k}{n_k}\|\nabla\mathcal{L}_{acc}(w_t^k)\| = \frac{\bar{\eta}_t^k - \eta_t^k}{\eta^k} v_t^k. \quad (14)$$

Finally we can get the renewed LR as

$$\bar{\eta}_t^k \approx \frac{\eta_t^k \bar{v}_t^k}{v_t^k}. \quad (15)$$

In actual AutoLR scheme, convergence to the goals of each epoch cannot be done at once, so we adopt an iterative trial policy as follows:

$$\bar{\eta}_t^k \leftarrow \frac{\eta_t^{k(i)} \bar{v}_t^k}{v_t^{k(i)}}, \;\; \eta_t^{k(0)} = \eta_t^k, \quad (16)$$

where i is the trial index. This renew procedure is repeated until the goal (2) is achieved in each epoch. The overall flow of AutoLR is given in Algorithm 2.

Note that, the guarantee of convergence for Eq.(16) and the guidance of setting for α and β are included in the supplementary material. The hyper-parameters α and β in our algorithm effectively saves the effort of searching through too much combinations of continuous real spaces to find the appropriate learning rate for each layer. In all experiments, α and β were set to 2 and 0.4 empirically following the guidance in the supplementary material. The threshold parameter τ_s was loosely set to 0.94 so that the training speed was not too slow by an approximate sorting instead of the complete sorting by setting $\tau_s = 1$.

Experiments

This section shows our experimental results. We begin by describing the target datasets and metric. Then, the experimental details are presented. We then describe the ablation study and visualization results of our algorithm. Finally, we show the comparison results with existing methods in the remaining sections.

Datasets

CUB-200 (Wah et al. 2011) dataset consists of 200 different species of birds. The first 100 classes are used for training and the other 100 classes for testing.
Cars-196 (Krause et al. 2013) dataset has 196 categories of car images and its total number is 16,185. The first 98 classes are used for training an the other 98 classes for testing.
Stanford Online Products (SOP) (Oh Song et al. 2016) has 120,053 images of 22,634 categories of products. 11,318 and 11,316 classes are used for training and testing, respectively.
Inshop (Liu et al. 2016) has 54,642 images of 11,735 categories of clothing items. 3,997 classes are used for training and the other 3,985 classes for testing. In the case of Inshop, the test dataset is divided into query and gallery.

Evaluation metric

The Recall@K metric (Oh Song et al. 2016) is employed for evaluating compared methods in image retrieval task.

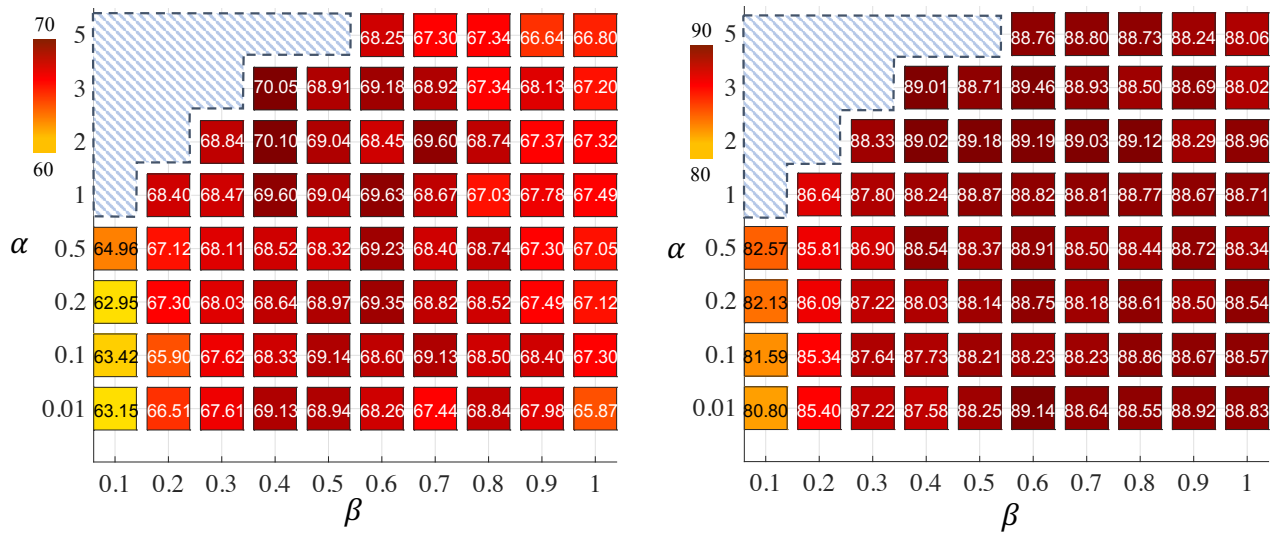
Implementation Details

We utilized ResNet-50 (He et al. 2016) pre-trained by using ImageNet (Deng et al. 2009). The input size was set to 224×224 and the batch size was set to 40. For input data augmentation, the horizontal flipping with 0.5 probability was employed in training. The initial LR was set to 1e-3. We used the cross-entropy loss function. The stochastic gradient descent (SGD) optimizer was used along with Nesterov momentum (Nesterov 1983). Initial momentum rate and weight decay coefficient were set to 0.9 and 5e-4, respectively.

Discussion of Hyper-parameters α, β

In the above, we have illustrated that the tuning of layer-wise LRs yields much improved performance. To manually tune the best layer-wise LRs, there are too much combinations of layer-wise LRs in a multi-dimensional continuous real space even when the lower bound and upper bound on the range of LRs are given. But our AutoLR can easily find the layer-wise LRs with only using the lower and upper bounds that are denoted by $\alpha \times \min_k v_0^{(k)}$ and $\beta \times \max_k v_0^{(k)}$, respectively.

In this section, we provide the investigation results for the hyper-parameters α and β on the CUB-200 and Cars-196 datasets. As shown in Figure 3-(a), the best performance on CUB-200 dataset is achieved when the α and β are set to values around 2 and 0.4, respectively. In the case of Cars-196 dataset, the setting α and β to 2 and 0.4 provides relatively high performance as shown in Figure 3-(b) but the setting α and β to 3 and 0.6 provides the best performance. Note that the hatched area of the Figure 3 is the unavailable area where the lower bound becomes larger than the upper bound.



(a) CUB-200 (b) Cars-196

Figure 3: Recall@1 results according to the hyper-parameters α and β on CUB-200 and Cars-196

Variants	CUB-200				Cars-196				SOP				InShop			
	1	2	4	8	1	2	4	8	1	10	10^2	10^3	1	10	20	30
(1) SingleLR	63.88	75.14	83.86	90.26	85.89	91.58	95.12	97.45	80.02	91.21	96.20	98.64	87.64	97.12	98.04	98.40
(2) Pruning only	67.00	78.49	86.85	92.07	87.85	93.11	96.06	98.02	83.31	92.90	96.68	98.62	88.62	97.16	97.95	98.40
(3) AutoLR	**70.10**	**80.62**	**88.08**	**92.98**	**89.02**	**94.23**	**96.72**	**98.14**	**84.24**	**93.47**	**97.15**	**98.92**	**91.72**	**98.04**	**98.66**	**98.93**

Table 2: Recall@K score of proposed method on image retrieval dataset for the ablation study

Ablation Study

We conducted ablation studies to validate the components of the proposed algorithm. We consider three ablation variants: (1) use conventional fine-tuning with the same LR over all layers (SingleLR), (2) apply layer-wise pruning only, and (3) apply layer-wise AutoLR with (2). The ablation studies were conducted on CUB-200, Cars-196, SOP, and Inshop with the pre-trained ResNet-50. Layer-wise pruning was done using our proposed Algorithm 1. Then, for all the target tasks, the layer-wise pruning 15,16 showed the best performance. The second row of Table 2 shows a consistent performance improvement over all target datasets in Recall@1. In the case of Inshop dataset, the performance does not improve much, but the pruning of two layers contributes to a reduction in the network complexity while maintaining comparable performance. The results show that our simple layer-wise pruning is an effective way to both improve performance and reduce network complexity.

The results for variant (3) are shown in the third row of Table 2. There are meaningful performance improvements for the three target tasks. Figure 4 shows the layer-wise trends of weight variations and LRs by our algorithm for layer-wise AutoLR. The learning was done up to 50 epochs, but after the convergence process was omitted after 15 epochs. t-i represents the i-th automatic tuning trial in each learning epoch. Figure 4-(a) shows that our AutoLR algorithm achieves the goal that the magnitudes of the weight variations are sorted in ascending order from low-level to high-level layers. In Figure 4-(a), Layer 14 is observed to be unsorted after four epochs. This is because the parameter τ_s to determine the successful sorting quality is not set to 1 (perfect sorting). Figure 4-(b) shows how the layer-wise LRs are adjusted from the initial LR of 0.001 for all layers and converge to layer-wise constants. The trial tuning iteration was performed once or twice before three epoch, and thereafter the first tuning was successful without further trial tuning.

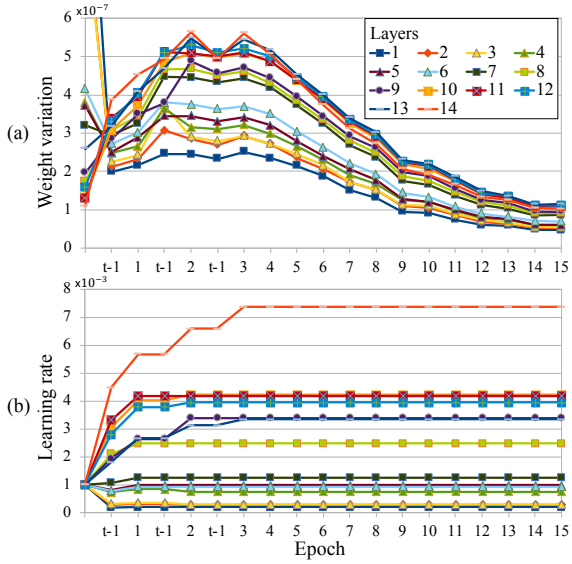


Figure 4: (a) Layer-wise weight variations and (b) layer-wise learning rate adaptations by our AutoLR algorithm

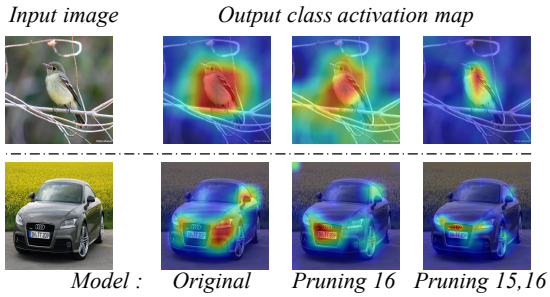


Figure 5: The class activation map (CAM) visualization of the last layers by applying different layers pruning

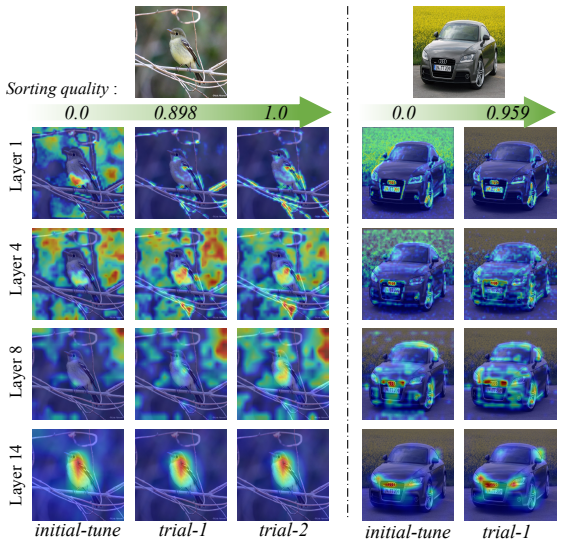


Figure 6: The class activation map (CAM) visualization of layers (1, 4, 8, 14) according to the sorting quality. The *initial-tune* is done by the conventional fine-tuning with single LR

Hence the additional overhead for trial tuning iterations required by the proposed method is negligible. According to the LR trends in the highest layer 14, its change in each tuning was the largest. This result supports our Hypothesis 2 that the high-level layer is specific to the target task and thus its weight variations should be large to adapt itself to the new target task. To meet the goal, the LR in Layer 14 converges to a large value promptly. Our AutoLR algorithm also is valid for other layers, as shown in Figure 4-(b).

In conclusion, the ablation study illustrates that the proposed layer-wise pruning and AutoLR algorithm is an effective and promising ways to improve performance and reduce network complexity by setting constant α, β and τ_s regardless of datasets.

Visualization on Effect to Layer-wise Features

To understand how the sorting quality of layer-wise weight variations affects the responses in the layers, we investigated the class activation map (CAM) in each layer using a visualization technique, Grad-CAM (Selvaraju et al. 2017).

Figure 5 shows the CAM at the last layer of each pruned model. In the case of *Original* without pruning, activation gives attention to a relatively large area. However, as high-level layers are pruned one by one, activation has more attention to the object or specific area, although the receptive field of each pruned one is the same. This supports our Hypothesis 1 that the there may be useless high-layers of a pre-trained network in a new task.

Figure 6 shows the CAM at each layer at the first epoch, where the sorting quality increases by AutoLR via one or two trial iterations. The *initial-tune* in Figure 6 is done using the conventional fine-tuning with single LR and the remaining trials are done using our AutoLR algorithm. As shown in Figure 6, the CAM at each layer tends to have more attention to the object as the sorting quality increases by our AutoLR. In Layer 1, as the sorting quality increases, unnecessary areas are deactivated and essential activation is formed in the target object area. This is because the AutoLR does not corrupt general features on the unnecessary area while the existing fine-tuning learns excessively the unnecessary area as a specific feature of the new target. In Layer 4, the CAM does not vary on unnecessary area; the CAM on the target area tends to be more attentive as the sorting quality increases. In Layer 8, the CAM on the target area tends to be more attentive as the sorting quality increases. However, the CAM also be attentive on unnecessary areas for the bird image due to the high LR tuned by our algorithm. In Layer 14, due to the pruning Layers 15 and 16 not being well-tuned by the existing fine-tuning, as shown in Figure 5, Layer 14 already has a good attention to the target. However, the activation is more attentive to the target as the sorting quality increases.

Method	R@1	R@2	R@4	R@8
Original	54.12	66.44	76.54	85.38
Pruning 13	60.77	71.94	81.68	88.93
Pruning 12, 13	63.67	74.71	83.78	90.60
Pruning 11, 12, 13	60.72	72.84	82.85	89.74

Table 3: Recall@K score of proposed Layer-wise pruning applied to Inception-V3 on CUB-200 dataset

Layer-wise Pruning for Other Backbone

To show the generality to other backbone networks, we conducted an experiment applying our layer-wise pruning scheme to Inception-V3. We divided Inception-V3 network into 13 layers as {Conv2d(3), Conv2d(2), Mixed_5b, Mixed_5c, Mixed_5d, Mixed_6a, Mixed_6b, Mixed_6c, Mixed_6d, Mixed_6e, Mixed_7a, Mixed_7b, Mixed_7c}. As shown in Table 3, pruning 12, 13 layers shows the best performance on the CUB-200 dataset.

Comparison with Other LR Settings

Our AutoLR algorithm belongs to the LR scheduling category. Here, we show the superiority of our algorithm by a comparative study on the LR scheduling. The compared methods are 'Step-decay' (Ge et al. 2019), 'Cyclic' (Smith 2017), and 'SGDR' (Loshchilov and Hutter 2016). Step-decay is the most widely used method in fine-tuning (Korn-

Method	Network	CUB-200				Cars-196				SOP				Inshop			
		1	2	4	8	1	2	4	8	1	10	10^2	10^3	1	10	20	30
A-BIER	Inception-v1	57.5	68.7	78.3	86.2	82.0	89.0	93.2	96.1	74.2	86.9	94.0	97.8	83.1	95.1	96.9	97.5
DREML	Inception-v3	58.9	69.6	78.4	85.6	-	-	-	-	-	-	-	-	78.4	93.7	95.8	96.7
ABE-8	Inception-v1	60.6	71.5	79.8	87.4	85.2	90.5	93.9	96.1	76.3	88.4	94.8	98.2	87.3	96.7	97.9	98.2
NormSoft	ResNet-50	61.3	73.9	83.5	90.0	84.2	90.4	94.4	96.9	79.5	91.5	96.7	-	89.4	97.8	**98.7**	**99.0**
Margin	ResNet-50	63.6	74.4	83.1	90.0	79.6	86.5	91.9	95.1	72.7	86.2	93.8	98.0	-	-	-	-
MS	Inception-v1	65.7	77.0	86.4	91.2	78.2	90.5	96.0	**98.7**	78.2	90.5	96.0	98.7	89.7	97.9	98.5	98.8
DGCRL	ResNet-50	67.9	79.1	86.2	91.8	75.9	83.9	89.7	94.0	-	-	-	-	-	-	-	-
Proxy-Anchor	ResNet-50	69.7	80.0	87.0	92.4	87.7	92.9	95.8	97.9	-	-	-	-	-	-	-	-
Pruning only	ResNet-50	67.0	78.5	86.9	92.1	87.9	93.1	96.1	98.0	83.3	92.9	96.7	98.6	88.6	97.2	98.0	98.4
AutoLR	ResNet-50	**70.1**	**80.6**	**88.1**	**93.0**	**89.0**	**94.0**	**96.9**	98.4	**84.2**	**93.5**	**97.2**	**98.9**	**91.7**	**98.0**	**98.7**	98.9

Table 4: Comparison with state-of-the-art methods on image retrieval datasets (Recall@K score)

Method	hyper-parameters	R@1	R@2	R@4
Step-decay	$t_d : 40, \gamma : 0.1$	67.17	78.19	86.19
	$t_d : 40, \gamma : 0.2$	67.35	78.14	86.04
Cyclic	*cycle* : 5	67.54	78.70	86.73
	cycle : 7	68.55	79.17	86.70
SGDR	$n_{reset} : 8$	68.18	79.34	86.34
	$n_{reset} : 14$	68.16	78.98	86.50
AutoLR	$\alpha : 2, \beta : 0.4$	**70.10**	**80.62**	**88.08**

Table 5: Comparison of our AutoLR with the various learning rate scheduling method for the CUB-200 dataset with the equally pruned ResNet-50 (Recall@K score)

blith, Shlens, and Le 2018; Sun et al. 2018; Ro et al. 2019). It conducts LR decay by multiplying to all layers by a value γ at a drop timing t_d. The initial LR of Step-decay is set to l_{max}. Cyclic adjusts the LR between the max value l_{max} and min value l_{min} with periodic triangular waveform. The number of cycle is a hyper-parameter. Similarly, SGDR adjusts the LR to decrease exponentially and LR is reset to its initial value l_{max}. These resets are repeated multiple times with a hyper-parameter n_{reset}.

For fair comparison, all methods were applied to the equally pruned network (pruning 15, 16), and all experiments were conducted equally for 50 epochs. The l_{max} and l_{min} for all experiments were set to 0.01 and 0.001, respectively. Depending on the hyper-parameters of each method, we tested the performance in a range guided in each method and selected the best performance. All experimental results are given in Table 5 of the supplementary material, where the tuned hyper-parameters are also listed for all the methods. As shown in Table 5, our AutoLR algorithm outperforms the other LR scheduling methods consistently.

Comparison with Existing Methods

Table 4 shows that our AutoLR outperforms not only the methods of using the same backbone ResNet-50 (Zhai and Wu 2018; Wu et al. 2017; Zheng et al. 2019; Kim et al. 2020), but also the methods of using other backbone networks (Opitz et al. 2017; Xuan, Souvenir, and Pless 2018; Kim et al. 2018; Wang et al. 2019). Another impressive one is that our method with only a pruning scheme outperforms the current SOTA in Cars-196 and SOP datsets even though it is a simple pruning method that removes just a couple of high-level layers assessed to be useless to a new task.

Conclusion

In this paper, we proposed a novel algorithm for layer-wise pruning and auto-tuning of layer-wise LRs with simple setting of hyper-parameters. The pruning algorithm uses a simple technique to prune a couple of high-level layers that are not helpful to a new task. The auto-tuning algorithm automatically adjusts the LRs depending on the role of each layer so that they contribute to performance improvement. The advantages of the proposed algorithm are not only simple for implementation, but also effective in improving performance and reducing network complexity. The effectiveness and efficiency of the proposed algorithm has been validated by the experiments on four image retrieval benchmark datasets. Hence the proposed pruning and automatic tuning algorithm will be able to contribute to the advances for automated and efficient machine learning.

Acknowledgments

This work was supported by the ICT R&D programs of MSIP/IITP [No.B0101-15-0552, Development of Predictive Visual Intelligence Technology] and [2017-0-00306, Outdoor Surveillance Robots].

References

Azizpour, H.; Razavian, A. S.; Sullivan, J.; Maki, A.; and Carlsson, S. 2015. Factors of transferability for a generic convnet representation. *IEEE transactions on pattern analysis and machine intelligence* 38(9): 1790–1802.

Cui, Y.; Song, Y.; Sun, C.; Howard, A.; and Belongie, S. 2018. Large scale fine-grained categorization and domain-specific transfer learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE International Conference on Computer Vision (CVPR)*.

Fu, J.; Zheng, H.; and Mei, T. 2017. Look closer to see better: Recurrent attention convolutional neural network for fine-grained image recognition. In *Proceedings of the*

IEEE conference on computer vision and pattern recognition (CVPR), volume 2, 3.

Ge, R.; Kakade, S. M.; Kidambi, R.; and Netrapalli, P. 2019. The Step Decay Schedule: A Near Optimal, Geometrically Decaying Learning Rate Procedure For Least Squares. In *Advances in Neural Information Processing Systems*, 14977–14988.

Guo, Y.; Shi, H.; Kumar, A.; Grauman, K.; Rosing, T.; and Feris, R. 2019. SpotTune: transfer learning through adaptive fine-tuning. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 4805–4814.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*.

Kim, S.; Kim, D.; Cho, M.; and Kwak, S. 2020. Proxy Anchor Loss for Deep Metric Learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*.

Kim, W.; Goyal, B.; Chawla, K.; Lee, J.; and Kwon, K. 2018. Attention-based Ensemble for Deep Metric Learning. In *The European Conference on Computer Vision (ECCV)*.

Kornblith, S.; Shlens, J.; and Le, Q. V. 2018. Do Better ImageNet Models Transfer Better? *arXiv preprint arXiv:1805.08974* .

Krause, J.; Stark, M.; Deng, J.; and Fei-Fei, L. 2013. 3d object representations for fine-grained categorization. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 554–561.

Li, Z.; Zhou, F.; Chen, F.; and Li, H. 2017. Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835* .

Liu, Z.; Luo, P.; Qiu, S.; Wang, X.; and Tang, X. 2016. Deepfashion: Powering robust clothes recognition and retrieval with rich annotations. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*.

Loshchilov, I.; and Hutter, F. 2016. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983* .

Nesterov, Y. 1983. A method for unconstrained convex minimization problem with the rate of convergence O (1/kˆ 2). In *Doklady AN USSR*.

Oh Song, H.; Xiang, Y.; Jegelka, S.; and Savarese, S. 2016. Deep metric learning via lifted structured feature embedding. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*.

Opitz, M.; Waltner, G.; Possegger, H.; and Bischof, H. 2017. Bier-boosting independent embeddings robustly. In *Proceedings of the IEEE international conference on computer vision (ICCV)*, 5189–5198.

Ren, M.; Zeng, W.; Yang, B.; and Urtasun, R. 2018. Learning to Reweight Examples for Robust Deep Learning. In *Proceedings of Machine Learning Research*.

Ro, Y.; Choi, J.; Jo, D. U.; Heo, B.; Lim, J.; and Choi, J. Y. 2019. Backbone Can Not be Trained at Once: Rolling Back to Pre-trained Network for Person Re-Identification. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Selvaraju, R. R.; Cogswell, M.; Das, A.; Vedantam, R.; Parikh, D.; and Batra, D. 2017. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision (CVPR)*, 618–626.

Smith, L. N. 2017. Cyclical learning rates for training neural networks. In *WACV*.

Suh, Y.; Han, B.; Kim, W.; and Lee, K. M. 2019. Stochastic Class-Based Hard Example Mining for Deep Metric Learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*.

Sun, Y.; Zheng, L.; Yang, Y.; Tian, Q.; and Wang, S. 2018. Beyond Part Models: Person Retrieval with Refined Part Pooling (and A Strong Convolutional Baseline). In *The European Conference on Computer Vision (ECCV)*.

Tajbakhsh, N.; Shin, J. Y.; Gurudu, S. R.; Hurst, R. T.; Kendall, C. B.; Gotway, M. B.; and Liang, J. 2016. Convolutional neural networks for medical image analysis: Full training or fine tuning? *IEEE transactions on medical imaging* 35(5): 1299–1312.

Wah, C.; Branson, S.; Welinder, P.; Perona, P.; and Belongie, S. 2011. The caltech-ucsd birds-200-2011 dataset .

Wang, X.; Han, X.; Huang, W.; Dong, D.; and Scott, M. R. 2019. Multi-Similarity Loss With General Pair Weighting for Deep Metric Learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*.

Wu, C.-Y.; Manmatha, R.; Smola, A. J.; and Krahenbuhl, P. 2017. Sampling matters in deep embedding learning. In *Proceedings of the IEEE International Conference on Computer Vision*, 2840–2848.

Xuan, H.; Souvenir, R.; and Pless, R. 2018. Deep randomized ensembles for metric learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Yosinski, J.; Clune, J.; Bengio, Y.; and Lipson, H. 2014. How transferable are features in deep neural networks? In *Advances in neural information processing systems*.

Zeiler, M. D.; and Fergus, R. 2014. Visualizing and understanding convolutional networks. In *The European Conference on Computer Vision (ECCV)*, 818–833. Springer.

Zhai, A.; and Wu, H.-Y. 2018. Making classification competitive for deep metric learning. *arXiv preprint arXiv:1811.12649* .

Zheng, H.; Fu, J.; Mei, T.; and Luo, J. 2017. Learning multi-attention convolutional neural network for fine-grained image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, volume 6.

Zheng, X.; Ji, R.; Sun, X.; Zhang, B.; Wu, Y.; and Huang, F. 2019. Towards optimal fine grained retrieval via decorrelated centralized loss with normalize-scale layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 9291–9298.