Supplementary Materials for *Auto-DAS: Automated Proxy Discovery for Training-free Distillation-aware Architecture Search*

Haosen Sun[1], Lujun Li[1✉], Peijie Dong[2], Zimian Wei[3], and Shitong Shao[4]

[1] The Hong Kong University of Science and Technology
[2] The Hong Kong University of Science and Technology (Guangzhou)
[3] National University of Defense Technology
[4] Southeast University
hsunas@connect.ust.hk;lilujunai@gmail.com;pdong212@connect.hkust-gz.edu.cn;
weizimian16@nudt.edu.cn; shaoshitong@seu.edu.cn

A More Discussions

About ground truth. Ground truths (D) represent the accuracy results of student models in full distillation training. These results are used to evaluate the correlation between proxy predictions.
About technical novelty compared to previous works. Distillation-aware architecture search (DAS) is important for distillation research area [5, 12–18, 20, 27, 29], with recently proposed training-free DAS as a new research track. In contrast to DisWOT and other methods [1,4,6–8,10,19,21,33,35], our Auto-DAS first achieves automated and general DAS by building new proxy search space and automating the search.
About comparison with other methods. (1) We use experiment with ResNet/ViT-like and NAS-101/201 models, comparing 6 train-based NAS, 10 train-free NAS, and 5 hand-designed methods, demonstrating our effectiveness. (2) Following this suggestion, we compare more hand-crafted models, KD methods, and distillation-unaware NAS methods for ViT, Swin, PiT, and ResNet18/50 on ImageNet. These results show that our Auto-DAS consistently outperforms Random NAS and other NAS methods on all models. (3) Random NAS does not always surpass handcrafted designs because of the huge search space. Our search algorithms play a significant contribution in solving this issue and improving final distillation accuracy. (4) Our Auto-DAS and hand-designed KD methods are fundamentally orthogonal techniques rather than competing ones, making them unsuitable for direct comparison.

B Details of Experiments

B.1 Details on ViT Experiments on Tiny Datasets

Datasets. Our study includes popular image classification datasets, namely Flowers [26] and Chaoyang [34]. These datasets offer diverse and representa-

✉ Corresponding author & contributed equally to this work.

tive samples, enabling us to evaluate the performance of our method across different domains. The Flowers dataset [26] is specifically designed for flower classification tasks and consists of 102 categories of various flower species. Each class contains a varying number of images, ranging from 40 to 258. The dataset poses a challenge due to the inherent similarities among different flower types, requiring models to capture subtle visual cues for accurate classification. The Chaoyang dataset [34] comprises colon slide image patches collected from the Chaoyang hospital and labeled by three professional pathologists. The testing set consists of patches with consensus labels from all pathologists, while the remaining patches form the training set. For samples in the training set with inconsistent labels, one pathologist's opinion is randomly chosen. The final dataset includes normal, serrated, adenocarcinoma, and adenoma samples for training and testing, totaling several thousand images.

Implementation. In the training process, we train DeiT, AutoFormer, PVT, and Swin models from scratch using the attention discovered. The training follows standard settings [11], which include 300 training epochs, a cosine learning rate scheduler, and the AdamW optimizer. Specifically, we utilize the AdamW optimizer [25] with an initial learning rate of 5e-4 and a weight decay of 0.05. The learning rate schedule follows a cosine policy [24], gradually reducing the learning rate to 5e-6. Each ViT model undergoes 100 epochs of training, with a linear warm-up period of 20 epochs, and employs a batch size of 128. The training process involves images with an interpolated resolution of 224×224. These standardized settings ensure consistency and enable fair comparisons among our attention candidates.

B.2 Details on ViT Experiments on ImageNet Datasets

Datasets. The ImageNet dataset [3] is a widely used large-scale dataset in computer vision research. It consists of 1.2 million training images and 50,000 validation images, covering 1,000 categories. The dataset encompasses a wide range of object categories, including animals, plants, vehicles, and everyday objects. ImageNet serves as a benchmark for evaluating the performance of various computer vision models and algorithms.

Implementation. We conduct our experiment on the ImageNet dataset [3] using standard settings [9, 23, 28]. The input images are resized to a size of 224x224 pixels using bicubic interpolation. In terms of training settings, we train the model for 300 epochs with a warm-up period of 20 epochs. The weight decay is set to 0.05. The base learning rate is 5e-4, the warm-up learning rate is 5e-7, and the minimum learning rate is 5e-6. The learning rate scheduler follows a "cosine" policy with a decay interval of 30 epochs and a decay rate of 0.1. We employ the AdamW optimizer with an epsilon value of 1e-8 and betas set to (0.9, 0.999). The SGD momentum is set to 0.9. For augmentation, we use the AutoAugment policy and set the random erase probability to 0.25. Additionally, we apply mixup with an alpha value of 0.8. These augmentation techniques enhance the model's generalization ability and improve overall performance.

C Details of Search Space

Our search space of the automatic proxy consists of transformation and distance operations. Their detailed implementations are presented in Listing 1-2.

C.1 Details of transformation operations

Our transformation options include: ”exp,” ”mish,” ”leaky,” ”relu,” ”tanh,” ”sigmoid,” ”pow2,” ”pow4,” ”log,” ”sqrt,” ”drop,” ”no’, ”satt,” ”natt,” ”catt,” ”mask,” ”bmm,” ”mm,” ”batch,” ”channel,” ”scale_r1,” ”scale_r2,” ”multi_scale_r4,” ”local_s1,” ”local_s2,” ”local_s4,” ”norm_HW,” ”norm_C,” ”norm_N,” ”softmax_N,” ”softmax_C,” ”softmax_HW,” ”scale,” ”logsoftmax_N,” ”logsoftmax_C,” ”logsoftmax_HW,” ”min_max_normalize,” ”batchnorm,”. We present some typical transformations as follows:

Attention transformation. Following FGD [30], we select the attention transformation operation on different pixels and different channels, respectively:

$$G^S(F) = \frac{1}{C} \cdot \sum_{c=1}^{C} |F_c|\,, A^S(F) = H \cdot W \cdot \sigma(G^S(F)/\tau), \tag{1}$$

$$G^C(F) = \frac{1}{HW} \cdot \sum_{i=1}^{H}\sum_{j=1}^{W} |F_{i,j}|, A^C(F) = C \cdot \sigma(G^C(F)/\tau), \tag{2}$$

where H, W, C denote the height, width, and channel of the feature. G^S and G^C are the spatial and channel attention maps. A^S and A^C are the spatial and channel attention mask, where τ is the temperature hyperparameter to adjust the distribution.

Mask transformation. Following MGD [31], we use the corresponding l-th mask to cover the student's l-th feature, which can be formulated as follows:

$$M^l_{i,j} = \begin{cases} 0, & \text{if } R^l_{i,j} < \lambda \\ 1, & \text{Otherwise} \end{cases} \tag{3}$$

where $R^l_{i,j}$ is a random number in $(0, 1)$ and i, j is the horizontal and vertical coordinates of the feature map, respectively.

Multi-scale transformation. Multi-scale feature representations benefit modeling context information in different abstract levels and are essential to many vision tasks. For example, PSPNet [32] adopts spatial pyramid pooling to probe convolutional features on multiple scales for semantic image segmentation. Following Review [2], our multi-scale transformation operation extracts different levels of knowledge from the feature using spatial pyramid pooling.

Local transformation. Local features refer to distinctive and repeatable patterns or structures within an image that can be used to identify and match the corresponding features in different images. LKD [22] uses a local correlation matrix based on the selected local parts to guide the student's learning. In our search space, we select the local transformation operation to divide the original feature into n^2 patches (*e.g.*, $n = 2, 4$), then distill each patch into separate instances.

C.2 Details of Distance Function Operations.

We use ℓ_1, ℓ_2, ℓ_{KL}, ℓ_{hard}, ℓ_{Cosine}, $\ell_{Pearson}$ as the distance function operations, which partially introduced in detail as follows:

In UniADS, different distance functions are used to measure the difference between teacher and student output. Let P_i denote the predicted probability of class i by the teacher network and Q_i denote the predicted probability of class i by the student network.
L_2 distance. The L_2 distance measures the square root of the sum of the squared differences between the probabilities of each class in the two distributions. The L_2 distance between P and Q is defined as:

$$D_{\ell_2}(P,Q) = \sqrt{\sum_{i=1}^{n}(P_i - Q_i)^2}$$

Cosine distance. The cosine distance measures the cosine of the angle between the two probability vectors. This distance measure is useful when the magnitudes of the probability vectors are not important, only their directions. The cosine distance between P and Q is defined as:

$$D_{\ell_{Cosine}}(P,Q) = 1 - \frac{\sum_{i=1}^{n} P_i Q_i}{\sqrt{\sum_{i=1}^{n} P_i^2}\sqrt{\sum_{i=1}^{n} Q_i^2}}$$

Pearson distance. The Pearson distance measures the correlation between the two probability vectors. The Pearson distance between P and Q is defined as:

$$D_{\ell_{Pearson}}(P,Q) = 1 - \frac{\sum_{i=1}^{n}(P_i - \bar{P})(Q_i - \bar{Q})}{\sqrt{\sum_{i=1}^{n}(P_i - \bar{P})^2}\sqrt{\sum_{i=1}^{n}(Q_i - \bar{Q})^2}}$$

where $\bar{P}$ and $\bar{Q}$ are the means of the two distributions
KL distance. The KL distance measures the information lost when approximating the probability distribution P with the probability distribution Q, as follows:

$$D_{\ell_{KL}}(P,Q) = \sum_{i=1}^{n} P_i \log\frac{P_i}{Q_i} = \sum_{i=1}^{n} P_i \log P_i - \sum_{i=1}^{n} P_i \log Q_i$$

Listing 1.1: The PyTorch implementation of transformation operations.

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from einops import rearrange, reduce

def trans_multi_scale_r1(f):
    """transform with multi-scale distillation with reduce ratio of 1"""
    if len(f.shape) != 4:
        return f
    return reduce(f, 'b c (h1 h2) (w1 w2) -> b c h1 w1', 'max', h2=1,
        w2=1)
```

```
def trans_multi_scale_r2(f):
    """transform with multi-scale distillation with reduce ratio of 2"""
    if len(f.shape) != 4:
        return f
    return reduce(f, 'b c (h1 h2) (w1 w2) -> b c h1 w1', 'max', h2=2,
        w2=2)

def trans_multi_scale_r4(f):
    """transform with multi-scale distillation with reduce ratio of 4"""
    if len(f.shape) != 4:
        return f
    return reduce(f, 'b c (h1 h2) (w1 w2) -> b c h1 w1', 'max', h2=4,
        w2=4)

def trans_local_s1(f):
    """transform with local features distillation with spatial size of
        1"""
    if len(f.shape) != 4:
        return f
    f = rearrange(f, 'b c (h hp) (w wp) -> b (c h w) hp wp', hp=1, wp=1)
    return f.squeeze(-1).squeeze(-1)

def trans_local_s2(f):
    """transform with local features distillation with spatial size of
        1"""
    if len(f.shape) != 4:
        return f
    return rearrange(f, 'b c (h hp) (w wp) -> b (c h w) hp wp', hp=2,
        wp=2)

def trans_local_s4(f):
    """transform with local features distillation with spatial size of
        1"""
    if len(f.shape) != 4:
        return f
    return rearrange(f, 'b c (h hp) (w wp) -> b (c h w) hp wp', hp=4,
        wp=4)

def trans_batch(f):
    """transform with batch-wise shape"""
    if len(f.shape) == 2:
        return f
    elif len(f.shape) == 3:
        return rearrange(f, 'b c h -> b (c h)')
    elif len(f.shape) == 4:
        return rearrange(f, 'b c h w -> b (c h w)')

def trans_channel(f):
    """transform with channel-wise shape"""
    if len(f.shape) in {2, 3}:
        return f
    elif len(f.shape) == 4:
        return rearrange(f, 'b c h w -> b c (h w)')

def trans_mask(f, threshold=0.65):
    """transform with mask"""
    if len(f.shape) in {2, 3}:
        # logits
        return f
    N, C, H, W = f.shape
    device = f.device
    mat = torch.rand((N, 1, H, W)).to(device)
    mat = torch.where(mat > 1 - threshold, 0, 1).to(device)
    return torch.mul(f, mat)

def trans_satt(f, T=0.5):
    """transform with spatial attention"""
    if len(f.shape) in {2, 3}:
        # logits
        return f
    N, C, H, W = f.shape
    value = torch.abs(f)
    fea_map = value.mean(axis=1, keepdim=True)
    # Bs*W*H
    S_attention = (H * W * F.softmax(
        (fea_map / T).view(N, -1), dim=1)).view(N, H, W)
    return S_attention.unsqueeze(dim=-1)

def trans_natt(f, T=0.5):
    """transform from the N dim"""
    if len(f.shape) == 2:
        N, C = f.shape
    elif len(f.shape) == 4:
        N, C, H, W = f.shape
    elif len(f.shape) == 3:
        N, C, M = f.shape
    # apply softmax to N dim
    return N * F.softmax(f / T, dim=0)

def trans_catt(f, T=0.5):
    """transform with channel attention"""
    if len(f.shape) == 2:
        # logits
        N, C = f.shape
        # apply softmax to C dim
        return C * F.softmax(f / T, dim=1)
    elif len(f.shape) == 3:
        N, C, M = f.shape
        return C * F.softmax(f / T, dim=1)
    elif len(f.shape) == 4:
        N, C, H, W = f.shape
        value = torch.abs(f)
        # Bs*C
        channel_map = value.mean(
            axis=2, keepdim=False).mean(
                axis=2, keepdim=False)
        C_attention = C * F.softmax(channel_map / T, dim=1)
        return C_attention.unsqueeze(dim=-1).unsqueeze(dim=-1)
    else:
        raise f'invalid shape {f.shape}'

def trans_drop(f, p=0.1):
    """transform with dropout"""
    return F.dropout2d(f, p)

def trans_nop(f):
    """no operation transform """
    return f

def trans_bmm(f):
    """transform with gram matrix -> b, c, c"""
    if len(f.shape) == 2:
        return f
    elif len(f.shape) == 4:
        return torch.bmm(
            rearrange(f, 'b c h w -> b c (h w)'),
            rearrange(f, 'b c h w -> b (h w) c'))
    elif len(f.shape) == 3:
        return torch.bmm(
            rearrange(f, 'b c m -> b c m'), rearrange(f, 'b c m -> b m
                c'))
    else:
        raise f'invalide shape {f.shape}'

def trans_mm(f):
    """transform with gram matrix -> b, b"""
    if len(f.shape) == 2:
        return f
    elif len(f.shape) == 3:
        return torch.mm(
            rearrange(f, 'b c m -> b (c m)'), rearrange(f, 'b c m -> (c
                m) b'))
    elif len(f.shape) == 4:
        return torch.mm(
            rearrange(f, 'b c h w -> b (c h w)'),
            rearrange(f, 'b c h w -> (c h w) b'))
    else:
        raise f'invalide shape {f.shape}'
```

```python
def trans_norm_HW(f):
    """transform with l2 norm in HW dim"""
    if len(f.shape) == 2:
        return f
    elif len(f.shape) == 3:
        return F.normalize(f, p=2, dim=2)
    elif len(f.shape) == 4:
        return F.normalize(f, p=2, dim=(2, 3))
    else:
        raise f'invalide shape {f.shape}'

def trans_norm_C(f):
    """transform with l2 norm in C dim"""
    return F.normalize(f, p=2, dim=1)

def trans_norm_N(f):
    """ transform with l2 norm in N dim"""
    return F.normalize(f, p=2, dim=0)

def trans_softmax_N(f):
    """transform with softmax in 0 dim"""
    return F.softmax(f, dim=0)

def trans_softmax_C(f):
    """transform with softmax in 1 dim"""
    return F.softmax(f, dim=1)

def trans_softmax_HW(f):
    """transform with softmax in 2,3 dim"""
    if len(f.shape) == 2:
        return f
    if len(f.shape) == 4:
        N, C, H, W = f.shape
        f = f.reshape(N, C, -1)
    assert len(f.shape) == 3
    return F.softmax(f, dim=2)

def trans_logsoftmax_N(f):
    """transform with logsoftmax"""
    return F.log_softmax(f, dim=1)

def trans_logsoftmax_C(f):
    """transform with logsoftmax"""
    return F.log_softmax(f, dim=1)

def trans_logsoftmax_HW(f):
    """transform with logsoftmax"""
    if len(f.shape) == 2:
        return f
    if len(f.shape) == 4:
        N, C, H, W = f.shape
        f = f.reshape(N, C, -1)
    assert len(f.shape) == 3
    return F.log_softmax(f, dim=2)

def trans_sqrt(f):
    """transform with sqrt"""
    return torch.sqrt(f)

def trans_log(f):
    """transform with log"""
    return torch.sign(f) * torch.log(torch.abs(f) + 1e-9)

def trans_pow2(f):
    """transform with ^2"""
    return torch.pow(f, 2)

def trans_pow4(f):
    """transform with ^4"""
    return torch.pow(f, 4)

def trans_min_max_normalize(f):
    """transform with min-max normalize"""
    A_min, A_max = f.min(), f.max()
    return (f - A_min) / (A_max - A_min + 1e-9)

def trans_abs(f):
    """transform with abs"""
    return torch.abs(f)

def trans_sigmoid(f):
    """transform with sigmoid"""
    return torch.sigmoid(f)

def trans_swish(f):
    """transform with swish"""
    return f * torch.sigmoid(f)

def trans_tanh(f):
    """transform with tanh"""
    return torch.tanh(f)

def trans_relu(f):
    """transform with relu"""
    return F.relu(f)

def trans_leaky_relu(f):
    """transform with leaky relu"""
    return F.leaky_relu(f)
```

```
def trans_mish(f):
    """transform with mish"""
    return f * torch.tanh(F.softplus(f))

def trans_exp(f):
    """transform with exp"""
    return torch.exp(f)

def trans_scale(f):
    """transform 0-1"""
    return (f + 1.0) / 2.0

def trans_batchnorm(f):
    """transform with batchnorm"""
    if len(f.shape) in {2, 3}:
        bn = nn.BatchNorm1d(f.shape[1]).to(f.device)
    elif len(f.shape) == 4:
        bn = nn.BatchNorm2d(f.shape[1]).to(f.device)
    return bn(f)
```

Listing 1.2: The PyTorch implementation of distance operations.

```
import torch
import torch.nn.functional as F
from torch import Tensor

def mse_loss(f_s: Tensor, f_t: Tensor) -> Tensor:
    """mse_loss = l2_loss = (f_s - f_t) ** 2"""
    return F.mse_loss(f_s, f_t)

def l1_loss(f_s: Tensor, f_t: Tensor) -> Tensor:
    """l1_loss = (f_s - f_t).abs()"""
    return F.l1_loss(f_s, f_t)

def l2_loss(f_s: Tensor, f_t: Tensor) -> Tensor:
    """mse_loss = l2_loss = (f_s - f_t) ** 2"""
    return F.mse_loss(f_s, f_t)

def kl_loss(f_s: Tensor, f_t: Tensor) -> Tensor:
    """kl_loss = kl_divergence = f_s * log(f_s / f_t)"""
    return F.kl_div(f_s, f_t, reduction='batchmean')


def smooth_l1_loss(f_s: Tensor, f_t: Tensor) -> Tensor:
    """smooth_l1_loss = (f_s - f_t).abs()"""
    return F.smooth_l1_loss(f_s, f_t)

def cosine_similarity(f_s, f_t, eps=1e-8):
    """cosine_similarity = f_s * f_t / (|f_s| * |f_t|)"""
    return F.cosine_similarity(f_s, f_t, eps=eps).mean()

def pearson_correlation(f_s, f_t, eps=1e-8):
    """pearson_correlation = (f_s - mean(f_s)) * (f_t - mean(f_t)) /
        (|f_s - mean(f_s)| * |f_t - mean(f_t)|)"""

    def cosine(f_s, f_t, eps=1e-8):
        return (f_s * f_t).sum(1) / (f_s.norm(dim=1) * f_t.norm(dim=1) +
            eps)

    return 1 - cosine(f_s - f_s.mean(1).unsqueeze(1),
                      f_t - f_t.mean(1).unsqueeze(1), eps).mean()
```

References

1. Chen, K., Yang, L., Chen, Y., Chen, K., Xu, Y., Li, L.: Gp-nas-ensemble: a model for the nas performance prediction. In: CVPRW (2022)
2. Chen, P., Liu, S., Zhao, H., Jia, J.: Distilling knowledge via knowledge review. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 5008–5017 (2021)
3. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: CVPR. pp. 248–255 (2009)
4. Dong, P., Li, L., Tang, Z., Liu, X., Pan, X., Wang, Q., Chu, X.: Pruner-zero: Evolving symbolic pruning metric from scratch for large language models. In: ICML (2024)
5. Dong, P., Li, L., Wei, Z.: Diswot: Student architecture search for distillation without training. In: CVPR (2023)
6. Dong, P., Li, L., Wei, Z., Niu, X., Tian, Z., Pan, H.: Emq: Evolving training-free proxies for automated mixed precision quantization. In: ICCV. pp. 17076–17086 (2023)
7. Dong, P., Niu, X., Li, L., Tian, Z., Wang, X., Wei, Z., Pan, H., Li, D.: Rd-nas: Enhancing one-shot supernet ranking ability via ranking distillation from zero-cost proxies. arXiv preprint arXiv:2301.09850 (2023)
8. Dong, P., Niu, X., Li, L., Xie, L., Zou, W., Ye, T., Wei, Z., Pan, H.: Prior-guided one-shot neural architecture search. arXiv preprint arXiv:2206.13329 (2022)
9. Dong, X., Bao, J., Chen, D., Zhang, W., Yu, N., Yuan, L., Chen, D., Guo, B.: Cswin transformer: A general vision transformer backbone with cross-shaped windows. ArXiv **abs/2107.00652** (2021)
10. Hu, Y., Wang, X., Li, L., Gu, Q.: Improving one-shot nas with shrinking-and-expanding supernet. Pattern Recognition (2021)
11. Li, K., Yu, R., Wang, Z., Yuan, L., Song, G., Chen, J.: Locality guidance for improving vision transformers on tiny datasets. In: European Conference on Computer Vision. pp. 110–127. Springer (2022)
12. Li, L.: Self-regulated feature learning via teacher-free feature distillation. In: ECCV (2022)
13. Li, L., Bao, Y., Dong, P., Yang, C., Li, A., Luo, W., Liu, Q., Xue, W., Guo, Y.: Detkds: Knowledge distillation search for object detectors. In: ICML (2024)

14. Li, L., Dong, P., Li, A., Wei, Z., Yang, Y.: Kd-zero: Evolving knowledge distiller for any teacher-student pairs. NeuIPS (2024)
15. Li, L., Dong, P., Wei, Z., Yang, Y.: Automated knowledge distillation via monte carlo tree search. In: ICCV (2023)
16. Li, L., Jin, Z.: Shadow knowledge distillation: Bridging offline and online knowledge transfer. In: NeuIPS (2022)
17. Li, L., Shiuan-Ni, L., Yang, Y., Jin, Z.: Boosting online feature transfer via separable feature fusion. In: IJCNN (2022)
18. Li, L., Shiuan-Ni, L., Yang, Y., Jin, Z.: Teacher-free distillation via regularizing intermediate representation. In: IJCNN (2022)
19. Li, L., Sun, H., Dong, P., Wei, Z., Shao, S.: Auto-das: Automated proxy discovery for training-free distillation-aware architecture search. In: ECCV (2024)
20. Li, L., Wang, Y., Yao, A., Qian, Y., Zhou, X., He, K.: Explicit connection distillation. In: ICLR (2020)
21. Li, L., Wei, Z., Dong, P., Luo, W., Xue, W., Liu, Q., Guo, Y.: Attnzero: Efficient attention discovery for vision transformers. In: ECCV (2024)
22. Li, X., Wu, J., Fang, H., Liao, Y., Wang, F., Qian, C.: Local correlation consistency for knowledge distillation. In: ECCV (2020)
23. Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., Guo, B.: Swin transformer: Hierarchical vision transformer using shifted windows. In: ICCV (2021)
24. Loshchilov, I., Hutter, F.: Sgdr: Stochastic gradient descent with warm restarts. arXiv preprint arXiv:1608.03983 (2016)
25. Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. In: ICLR (2017)
26. Nilsback, M.E., Zisserman, A.: Automated flower classification over a large number of classes. In: 2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing. pp. 722–729. IEEE (2008)
27. Shao, S., Dai, X., Yin, S., Li, L., Chen, H., Hu, Y.: Catch-up distillation: You only need to train once for accelerating sampling. arXiv preprint arXiv:2305.10769 (2023)
28. Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., Jegou, H.: Training data-efficient image transformers amp; distillation through attention. In: ICML (2021)
29. Xiaolong, L., Lujun, L., Chao, L., Yao, A.: Norm: Knowledge distillation via n-to-one representation matching (2022)
30. Yang, Z., Li, Z., Jiang, X., Gong, Y., Yuan, Z., Zhao, D., Yuan, C.: Focal and global knowledge distillation for detectors. arXiv preprint arXiv:2111.11837 (2021)
31. Yang, Z., Li, Z., Shao, M., Shi, D., Yuan, Z., Yuan, C.: Masked generative distillation. arXiv preprint arXiv:2205.01529 (2022)
32. Zhao, H., Shi, J., Qi, X., Wang, X., Jia, J.: Pyramid scene parsing network. In: CVPR (2017)
33. Zhu, C., Li, L., Wu, Y., Sun, Z.: Saswot: Real-time semantic segmentation architecture search without training. In: AAAI (2024)
34. Zhu, C., Chen, W., Peng, T., Wang, Y., Jin, M.: Hard sample aware noise robust learning for histopathology image classification. TMI (2021)
35. Zimian Wei, Z., Li, L.L., Dong, P., Hui, Z., Li, A., Lu, M., Pan, H., Li, D.: Auto-prox: Training-free vision transformer architecture search via automatic proxy discovery. In: AAAI (2024)