A. More Results

A.1. Main Results: MobileNetV2 as Student.

Due to the length limit of the main paper, we show the results of MobileNetV2 as student in Table S.1. Similar results have been achieved with MobileNetV2 (MNV2) as student compared to R18 as student, which shows the generalization of the proposed KDEP method.

A.2. Detailed Proof of Theorem 1

\[ F(\sigma) = \mathbb{E}[(T - S)^2] \]

\[ = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} (t - s)^2 \cdot P(t, s) \, dt \, ds \]

\[ = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} (t - s)^2 \cdot P(t) \cdot P(s) \, dt \, ds \]

\[ = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} (t - s)^2 \cdot \frac{1}{2\pi \sigma_s^2} e^{-\frac{(t - s)^2}{2\sigma_s^2}} \, dt \, ds \]

\[ = \frac{1}{\sqrt{2\pi \sigma_s^2}} \int_{-\infty}^{+\infty} e^{-\frac{t^2}{2\sigma_s^2}} \left( \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi \sigma_t^2}} e^{-\frac{(t - s)^2}{2\sigma_t^2}} \, ds \right) \, dt \]

\[ = \frac{1}{\sqrt{2\pi \sigma_s^2}} \int_{-\infty}^{+\infty} e^{-\frac{t^2}{2\sigma_s^2}} (t^2 + \sigma_s^2) \, dt = \sigma^2 + \sigma_s^2 \]

\[ \frac{dF(\sigma)}{d\sigma} = 2\sigma > 0 \Rightarrow \text{monotonically increasing} \]


Here, we show the KDEP results of some traditional feature-based KD methods that are developed for distilling knowledge to improve student’s performance for a specific task instead of its transferability. Also, these methods all require task label loss which violates our setting of an unlabeled dataset. Hence, we don’t include them in our paper to compare for fairness. As shown in Table S.2, previous feature-based KD methods largely rely on logit KD...
that transform channel-wisely, which may break the original relative magnitude between channels. Here, we provide a toy example as an illustration.

For instance, we have a three channel penultimate layer with target \(\text{Std}=[4, 3, 2]\) and after SVD \(\text{Std}=[50, 5, 1]\). For a feature after SVD that is \([10, 2, 2]\), with SN we have \([0.2, 0.4, 2]\), and with SM we have \([0.8, 1.2, 4]\), both losing the original relative magnitude.

**A.5. Computation cost.**

In most of our experimental results, we provide the training time of KDEP and supervised pre-training, where only unnoticeable extra training time is added. Moreover, the GPU memory usage during KDEP is also similar to supervised pre-training since we do not require gradients for the teacher. Yet, our teacher model is R50, and additional computation costs may increase when using larger teachers that inquires more inference time and GPU memory. Still, the pre-training time could be largely reduced compared with supervised pre-training.

**B. Implementation Details**

We implement our method using the PyTorch [9] framework and use SGD with momentum of 0.9 for all our experiments. All experiments are conducted using four 32G V100 GPUs.

**B.1. KDEP Setups**

For the KDEP procedure, we use an initial learning rate (lr) of 0.3 for R50→R18 and 0.1 for R50→MNv2. Batch size is set to 512. For data augmentation, we use RandomResizedCrop(224) and RandomHorizontalFlip. In order to reduce the burden of hyper-parameter tuning (e.g. weight decay), we multiply our feature-based loss (refer to Eq. 1 in our paper) by a loss weight \(w\), which matches the feature-based loss to the loss scale of supervised pretraining. For reproduction, we provide the weight loss of different teacher-student pairs in Table S.3.

Table S.3. Value of loss weight \(w\) for different T-S pairs.

<table>
<thead>
<tr>
<th>Teacher→Student</th>
<th>(w)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard SP R50→R18</td>
<td>20</td>
</tr>
<tr>
<td>MS R50→R18</td>
<td>3</td>
</tr>
<tr>
<td>SWSL R50→R18</td>
<td>1</td>
</tr>
<tr>
<td>MEAL V2 R50→R18</td>
<td>1</td>
</tr>
<tr>
<td>Swin-B→R18</td>
<td>3</td>
</tr>
<tr>
<td>MS R50→MNv2</td>
<td>3</td>
</tr>
</tbody>
</table>

For the 10% ImageNet data setting, we sample 10% images from each class of the original 1000 class in ImageNet-1K. We set the training epochs to 90 or 180, and drop the lr by a factor of 10 at 1/3 and 2/3 of total epochs. When using all of ImageNet data, we train for 9 or 18 epochs to verify fast convergence, and for 90 epochs to further boost performance, where 90 epochs with all ImageNet data is the standard supervised pretraining schedule [12]. We use weight decay \(\in \{1e^{-4}, 4e^{-4}, 5e^{-4}\}\) according to the length of the training schedule, shown in Table S.4.

<table>
<thead>
<tr>
<th>Data</th>
<th>Epoch</th>
<th>Weight decay</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>90</td>
<td>5e^{-4}</td>
</tr>
<tr>
<td>10%</td>
<td>180</td>
<td>4e^{-4}</td>
</tr>
<tr>
<td>100%</td>
<td>9</td>
<td>5e^{-4}</td>
</tr>
<tr>
<td>100%</td>
<td>18</td>
<td>4e^{-4}</td>
</tr>
<tr>
<td>100%</td>
<td>90</td>
<td>1e^{-4}</td>
</tr>
</tbody>
</table>

Table S.4. Weight decay for different KDEP training schedules.

**B.2. Downstream Task Setups**

For all downstream tasks, we use the same schedule and evaluation protocols for all models for a fair comparison.

For image classification, we initialize the backbone with the distilled weights and add a linear classifier with random initialization. We train the network for 150 epochs with batch size of 64, weight decay of 5e^{-4}, an initial learning rate \(\in \{0.01, 0.001\}\) which drops by a factor of 10 at 1/3 and 2/3 of total epochs. Again, we use RandomResizedCrop(224) and RandomHorizontalFlip for data augmentation.

For semantic segmentation, we also initialize the backbone with the distilled weights, and add a PSP module [13] and a segmentation head after the backbone. We use batch size of 16, an initial learning rate of 0.01, weight decay of 1e^{-4}, crop size of 512, and deploy an polynomial learning rate annealing procedure [1]. For data augmentation, we use random scaling, random horizontal flipping, random rotation, and random Gaussian blur. The number of epochs is 50, 100, 200 for VOC12, ADE20K, and Cityscapes, respectively, following previous standard [13].

For object detection, we experiment with the Faster R-CNN [10] detector and backbones are also initialized by the distilled weights. Unless noted, all the setups follow the evaluation protocols in MOCO [2]. For ResNet18, we use a backbone of R18-C4 (similar to R50-C4 [3]) for both VOC and COCO experiments. For MobileNetV2, we equip it with a FPN [8] backbone. 1x schedule is applied for COCO.

**B.3. Parametric/Non-parametric Methods.**

In our experiments, we experiment with three 1×1 conv variants for parametric methods: Post-ReLU, Pre-ReLU and the one in [4]. Specifically, we add a 1×1 convolutional layer and a Batch Normalization layer either after (Post-ReLU) or before (Pre-ReLU) the ReLU activation function. We also experiment with the Pre-ReLU 1×1 conv method equipped with Margin ReLU of teacher’s feature and Partial \(L_2\) loss function as in [4].
For non-parametric methods, we explore channel selection (CS.var, CS.rand), interpolation, and SVD. For channel selection methods, we experiment with two methods: selecting the top-$D_s$ channels with largest variances (CS.var) or random selecting $D_s$ channels (CS.rand). For interpolation method, we use the default nearest-neighbor interpolation in PyTorch [9]. For SVD, we calculate the singular vectors offline and use the top-$D_s$ principal components to transform the teacher’s features during the online KDEP process.

References


