

Residual Mixture of Experts

Lemeng Wu¹*, Mengchen Liu², Yinpeng Chen², Dongdong Chen², Xiyang Dai², Lu Yuan²

¹University of Texas at Austin ²Microsoft

lmwu@cs.utexas.edu, {mengchliu, yiche, dochen, xidai, luyuan}@microsoft.com

Abstract

Mixture of Experts (MoE) is able to scale up vision transformers effectively. However, it requires prohibiting computation resources to train a large MoE transformer. In this paper, we propose Residual Mixture of Experts (RMoE), an efficient training pipeline for MoE vision transformers on downstream tasks, such as segmentation and detection. RMoE achieves comparable results with the upper-bound MoE training, while only introducing minor additional training cost than the lower-bound non-MoE training pipelines. The efficiency is supported by our key observation: the weights of an MoE transformer can be factored into an input-independent core and an input-dependent residual. Compared with the weight core, the weight residual can be efficiently trained with much less computation resource, e.g., finetuning on the downstream data. We show that, compared with the current MoE training pipeline, we get comparable results while saving over 30% training cost. When compared with state-of-the-art non-MoE transformers, such as Swin-T / CvT-13 / Swin-L, we get +1.1 / 0.9 / 1.0 mIoU gain on ADE20K segmentation and +1.4 / 1.6 / 0.6 AP gain on MS-COCO object detection task with less than 3% additional training cost.

1 Introduction

Vision transformers have achieved a lot of breakthroughs recently, with the strong capability to capture a large amount of data. Such capability enables us to pretrain a vision transformer on a large-scale upstream dataset. The pretrained model eases the finetuning on challenging downstream tasks, with a fast convergence speed and outstanding performance.

To further enhance the model capacity of vision transformers, some works scales up the vision transformers to billions of parameters that can effectively capture huge upstream data [3, 62]. These models achieve state-of-the-art performances on various downstream tasks, such as semantic segmentation and object detection. However, directly scaling up a vision transformer by increasing model width and depth will dramatically increase the training costs. This makes the training for large vision transformers unaffordable for most vision researchers and practitioners [62]. In addition, many computer vision downstream tasks use high-resolution images as inputs. As a result, direct scaling up is also limited by GPU memory.

To solve the problems in model scaling up, there are recent research attempts [18, 35, 41, 44] that introduce conditional computation and sparsity in transformers. As one of the most representative technique, Mixture of Experts (MoE) scales up a transformer with conditionally computed experts. In an MoE transformer,

*Work done during an internship at Microsoft.

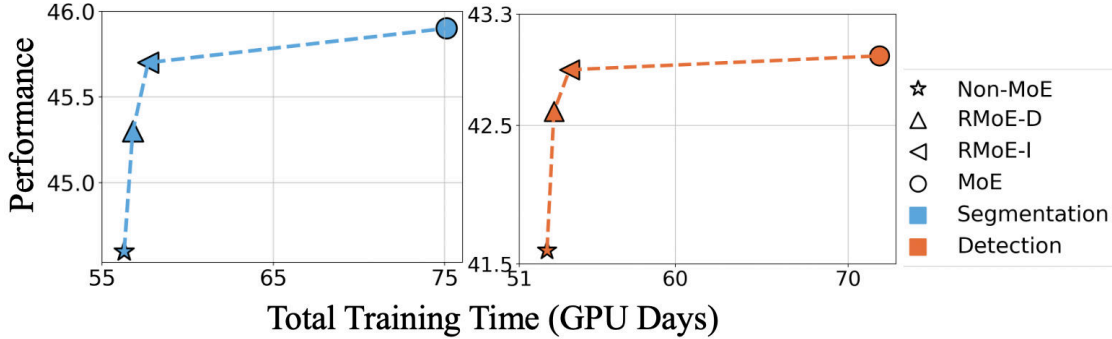


Figure 1: RMoE: balancing performance improvement and additional training cost between upper-bound MoE and lower-bound non-MoE training pipelines. RMoE-D and RMoE-I are two variants of RMoE.

each data point is only processed by a certain number of experts. By making the components of model conditionally computed, the training cost of MoE transformers is much less than that of non-MoE transformers at the same scale.

In this work, we propose Residual Mixture of Experts (RMoE), an efficient training pipeline for large MoE vision transformers. As shown in Fig. 1, RMoE achieves comparable results with the upper-bound MoE training pipelines, while only introducing minor additional training cost than the lower-bound non-MoE training pipelines. The efficiency is supported by our observation from analyzing the weights of experts in a trained MoE transformer. We find that the weights can be factored into an input-independent core and an input-dependent residual. Although the size of the weights residual is much large than that of weights core, the residual part can be efficiently trained with much less computation resource, e.g., finetuning on the downstream data. Using this observation, we develop the RMoE training pipeline. As shown in Figure 2 (b) and (c), in RMoE training, we can pretrain a non-MoE transformer on the upstream data and scale up the model during downstream finetuning or intermediate finetuning. By skipping the training of the scaled-up model on the heavy-duty upstream task, we only introduce minor additional training cost while enjoying the performance boost brought from the large model capacity. Furthermore, since many pretrained non-MoE transformer checkpoints are publicly available, practitioners can leverage these well-trained transformers as an initialization. Thus, our method opens the possibility to customize a large-scale transformer for various tasks without the limitation of the large amount of computation resources. We demonstrate the effectiveness of RMoE on object detection and segmentation. By applying RMoE to different vision backbones, Swin-T, CvT-13 and Swin-L, we get a +1.1 / 0.9 / 1.0 mIoU on ADE20K and +1.4 / 1.6 / 0.6 AP on object detection with less than 3% additional cost.

2 Related Work

2.1 Vision Transformers

Convolution Neural Networks (CNNs), as universal and powerful structures in computer vision domain [24, 26, 27, 28, 32, 42, 45, 46, 47], get a great success in the last decade. Recently, vision transformers, show promising results in various vision tasks and attract intense research interests. ViT [15], as the first breakthrough work, models each image as a set of tokens, where each token is an image patch. It achieves a competitive image classification result compared with traditional CNNs. Triggered by ViT, a series of follow-up works [8, 9, 14, 17, 21, 25, 30, 38, 49, 50, 52, 54, 57, 59, 60] continuously refresh the image classification records and push the potential of vision transformers in computer vision to a new height. In the

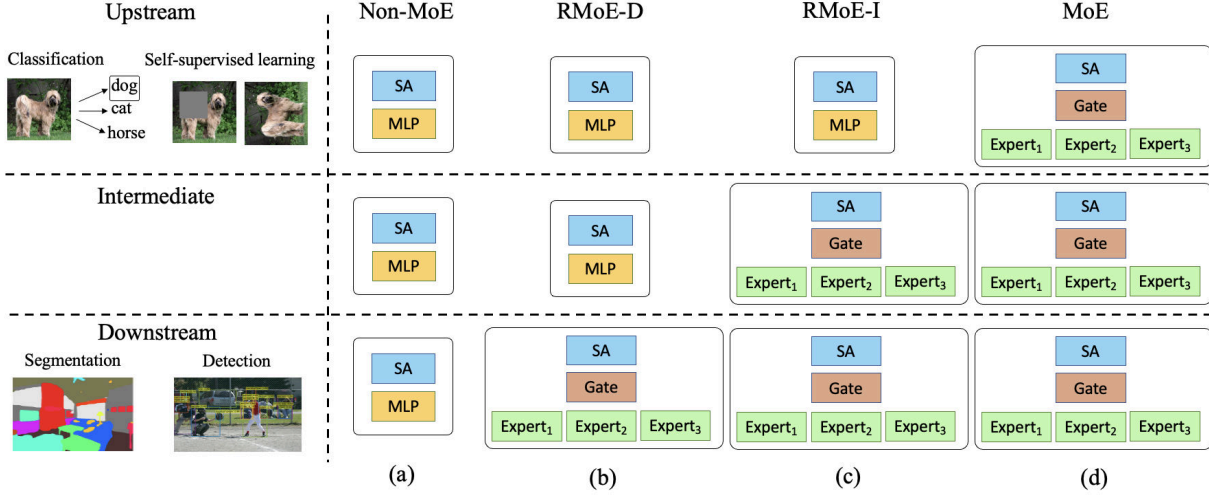


Figure 2: Training pipelines for (a) non-MoE, (b)(c) RMoE and (d) MoE. Here we use 3 experts as an example to illustrate how RMoE works in the intermediate and downstream finetune stages. Compared with MoE training, we inherit the non-MoE transformer after the upstream pretraining to save the cost. In this figure, we simplify the general transformer block design by only showing self-attention (SA) and multilayer perceptron (MLP) modules. Norm and add operations are not shown here.

above works, many adopt a hierarchical architecture and a large-scale upstream data pretraining. As a result, they unlock the ability to finetune to various downstream vision tasks like segmentation and detection. In our work, we leverage the architectures and available checkpoints of these powerful vision transformers and discover the ability to scale them up into an more powerful model with a low cost. We demonstrate that RMoE is not restricted to particular vision transformers.

2.2 Mixture of Experts

Mixture of Experts (MoE) has a long history through these decades [6, 29, 31, 61]. Various of expert architectures are proposed [10, 12, 43, 48, 51]. In the meanwhile, instead of designing experts in the architecture, there are similar research ideas to generate a set of outputs and pick the correct one, such as multi-choice learning [20, 33, 34].

Recently, Scaling up transformers using MoE is proven effective to achieve the state-of-the-art performance on various of tasks [40, 41]. Yet, it is still expensive to train a neural network with billions of parameters. Compared with non-MoE models, an MoE neural network contains a set of experts, e.g., multilayer perceptrons (MLPs), and a router to select which subset of experts are used for each input data point. It increases the network capacity by such conditional computation while maintaining a relative efficient training. MoE has been widely used in Natural Language Processing [18, 22, 35, 44] and computer vision [1, 16, 19, 41, 53, 58]. In our work, we also strive to scale up the model capacity of vision transformers. Compared with previous works on MoE vision transformers, we develop a more efficient training pipeline, which reduces the total training cost to obtain a large MoE vision transformer. Such efficiency is achieved via our factorization of the experts' weights. In addition, we are among the first to solve the problem of applying MoE on downstream tasks with high-resolution images such as segmentation and detection. These kinds of tasks pose technical challenges on the device memory while requiring a strong backbone for a good performance, which naturally fit our work.

3 Background

In this section, we introduce the basic idea of MoE and how to employ it to transformers to scale up the model.

MoE layer. MoE layers are critical components in an MoE model. By allowing input-dependent conditional computation, different experts in an MoE layer are assigned to process with different parts of the input space [29]. As one of the most common setting [44], an MoE layer contains two components: (1) n experts $E_i(x) : \mathbb{R}^{D_{in}} \rightarrow \mathbb{R}^{D_{out}}$, $i = 1 \dots n$ to process different inputs and (2) a gate function $G(x) : \mathbb{R}^{D_{in}} \rightarrow \mathbb{R}^n$ to route different inputs to different experts. Given an input $x \in \mathbb{R}^{D_{in}}$, an n -experts MoE layer computes the conditional output $y \in \mathbb{R}^{D_{out}}$ as the weighted sum of gate function $G(x)$ and experts outputs $[E_i(x)]$:

$$y = \sum_{i=1}^n G(x)_i E_i(x). \quad (1)$$

G and E_i are usually modeled with neural networks. In our research, we follow the previous designs [41, 44] to set G as a linear gate with a *softmax* function. To encourage sparsity in an MoE layer, usually it restricts only k experts where $k < n$ to participate in the computation for each input. So a TopK operator is utilized in the gate function to force only k experts used while others are skipped for an input. Thus we can write $G(x)$ as

$$G(x) = \text{TopK}(\text{softmax}(\theta_g x)), \quad (2)$$

where θ_g is the parameters of the linear operation in a gate.

Load Balancing Loss. In practice, one major problem to train an MoE model is that some of the experts are routed with much less data points than other experts. This will induce insufficient training for the experts routed with less data points. In addition, as experts are usually processed in parallel, it will also hurt the training efficiency due to the buckets effect. To avoid this problem, we usually add a load balancing loss [44] on the gate function to the total training loss. Given a batch of inputs X , a widely-used load balancing loss L is defined as

$$L(X) = \left(\frac{\text{std}(\text{Imp}(X))}{\text{mean}(\text{Imp}(X))} \right)^2, \quad (3)$$

where $\text{Imp}(X) := \sum_{x \in X} G(x)$. In practice, we may add this load balancing loss to total training loss with a balance weight w_{balance} .

MoE Transformer. To employ MoE to transformers to build MoE transformers, a widely-used approach is replacing some MLP layers in a non-MoE transformer by MoE layers [41]. In particular, in an MoE layer, the experts share the same structure with the original MLP. The gate function receives the output from the previous attention layer and routes the representations for tokens to different experts. The sparse outputs of experts are combined via Eq. 1.

4 Residual Mixture of Experts

4.1 Motivation

Our research is motivated by analyzing how the weights of experts in an MoE vision transformer evolve during the training process. Specifically, we trained an 8-experts MoE Swin-T [38] on the ImageNet22K [13] for 90 epochs, where we applied MoE to every other Swin transformer block following previous research [41].

Given the trained model, we focused to analyze the weights of experts in the last MoE layer because: (1) deeper routing decisions correlate with image classes and present richest semantic information [41]; and (2) the last layers affect the most to the classification performance. The weights of experts at different epochs are visualized by projecting to a 2D plane. To avoid non-linear distortion in the projection process, we adopted principal components analysis and visualized the top 2 principal components as values on X and Y axis in a scatter-plot. As shown in Figure 3 (a), we observed that the weights of different experts are clustered according to their training epochs. More importantly, the variance of each weight cluster is much smaller than the variance of cluster centers during the training.

These observations trigger us to design a more efficient training pipeline than the current MoE training. In particular, as shown in Figure 3 (b), we can factor the MoE training into two stages. First we train the centers of expert clusters (solid red path). Second, we train the residual weights of experts (dash blue paths). Comparing these two stages, the weights change in the second stage are much smaller than that in the first stage. Thus, the training cost needed for the second stage is much smaller and the total training cost is mainly determined by the first stage. In the first stage, to train the weight cluster centers, we can use a non-MoE transformer instead of an MoE model because the experts center is input independent. Thus, the first training stage itself is efficient because the training cost for an MoE vision transformer is $1.5 \sim 2 \times$ higher than that of non-MoE models [41]. In practice, for downstream tasks, we can often even skip the first training stage and only perform the residual training by directly using the available non-MoE vision transformer checkpoints. As a result, we can train a large MoE vision transformer efficiently.

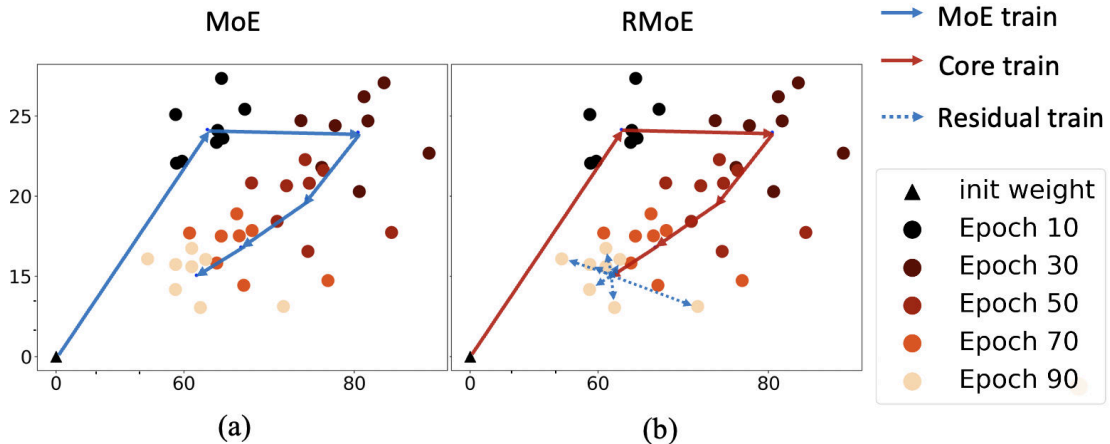


Figure 3: Visualization of experts weights evolution during an MoE training process. Here we show the visualization of the experts in the last layer of an 8-experts MoE Swin-T. We project the weights to a 2D plane using principle components analysis. Each point is the weight of an expert at a training epoch. The X and Y axis are the first and second principle components, respectively: (a) in the training process, experts weights at different epochs are clustered and the variance of each weight cluster (points of the same color) is much smaller than the variance of cluster centers (blue solid line) during the training. Motivated by this pattern, our RMoE on (b) first learns the centers of experts’ weights (red solid line) and then learns their residual weights (dash blue line).

4.2 Formulation

Generally, the conditional computation of an MoE model f can be formulated as $y = f(x; \theta(x))$, $\theta(x) \subset \Theta$, where x is an input data point, y is the output and Θ is the whole set of model parameters. Using this

formulation, the training of an MoE model aims at minimizing empirical loss \mathcal{L} :

$$\min_{\theta} \sum_{x \in X} \mathcal{L}(f(x; \theta(x))). \quad (4)$$

where X is the training set.

Motivated by the observations in Sec. 4.1, we factor the conditional computation $\theta(x)$ as an input-independent θ_0 and an input-dependent residual $\theta_r(x)$:

$$\theta(x) = \theta_0 + \theta_r(x). \quad (5)$$

In this factorization, the size of the weights residual $\theta_r(x)$ is **much larger** than that of the weights core θ_0 because experts in a MoE layer share the same weights core and we often have 8 – 32 experts in an MoE vision transformer [41]. On the contrary, the weights residual can be efficiently trained with **much less** computation resource, e.g., less training data and less training epochs.

Using this factorization, we propose the Residual Mixture of Experts (RMoE) training pipeline, which formulates the training of an MoE vision transformer as a bilevel optimization with residual θ_r as the upper-level variable and core θ_0 as the lower-level variable:

$$\begin{aligned} & \min_{\theta_r} \sum \mathcal{L}(f(x; \theta_0^* + \theta_r(x))) \\ \text{s.t. } & \theta_0^* = \operatorname{argmin}_{\theta} \sum \mathcal{L}(f(x; \theta)). \end{aligned}$$

4.3 Our Designs

To fully leverage the benefits of the RMoE training pipeline, there are several practical designs considerations needed to discuss:

Training Pipeline Design. The training pipelines of non-MoE and MoE models are similar and can be divided into several stages. As shown in Figure 2 (a) and (d), in the first stage, vision transformers are pretrained on a large-scale upstream dataset.

Then, there is an optional intermediate finetune stage. It is to bridge the task gap or resolution gap between upstream and downstream, e.g., BeiT [2].

Finally, the model is finetuned on the downstream dataset, such as semantic segmentation or object detection.

Compared with non-MoE and MoE training pipelines, RMoE starts with the non-MoE transformer training on the upstream task and efficiently finetunes the non-MoE transformer into a MoE transformer. As shown in Figure 2 (b) and (c), in RMoE training, we can either intermediately finetune the MoE model on the upstream dataset for a few epochs or directly finetune the MoE model on the downstream dataset.

We denote these two different pipelines as RMoE-I (Intermediate) and RMoE-D (Downstream).

Performance-preserving between non-MoE and MoE transformers. In RMoE, we need to inherit the learned weight core from the upstream non-MoE pretraining to perform a residual learning.

To this end, we initialize the weights of each expert in an MoE layer as the weights learned in the corresponding MLP layer from non-MoE pretraining. In addition, we add a small noise to each expert to escape from the local minima.

In practice, we find that that directly inheriting the experts weights will cause performance degradation. Specifically, as in Eq. 1, the output of an MoE layer is the weighted sum of gate function $G(x)$ and experts outputs $[E_i(x)]$: $MoE(x) = \sum_{i=1}^n G(x)_i E_i(x)$. In RMoE, because all experts inherit the weights from non-MoE pretraining and $G(x)$ is post-processed by a Top-K operation to ensure sparsity, we have:

$$MoE(x) \approx (\sum_{i=1}^n G(x)_i) \cdot MLP(x) < MLP(x). \quad (6)$$

Thus, the outputs of MoE layers are scaled down compared with original MLP layers and cause performance degradation.

To tackle this issue, we propose to align the outputs of the MoE and corresponding MLP layers while maintain the gradient flow in an MoE layer:

$$y = \sum_{i=1}^n \text{StopGrad}((1 - G(x)_i)E_i(x)) + G(x)_i E_i(x), \quad (7)$$

where StopGrad operation is to stop the gradient of the given term. In this way, when back-propagate the gradient, the experts can still get a normal gradient update only for the top-K experts. In RMoE, we only apply this for intermediate finetune because: (1) the intermediate finetune stage in RMoE is often short with a small learning rate and it is hard to fully recover the performance degradation; and (2) the downstream finetuning with the new decode head is done with a large learning rate and the performance drop will recover quickly.

Layer Choices for MoE layers. As introduced in Sec. 3, MoE layers are the key difference between MoE and non-MoE transformers. Thus, to finetune a pretrained non-MoE transformer into an MoE transformer, we need to replace several MLP layers to MoE layers [18, 41]. To avoid potential over-fitting, we only select the most important layers that contribute most to network training.

Inspired by Firefly Splitting [55], in RMoE, we select the layer that can maximally decrease the loss function in a layer-wise fashion. First, we over-grow the non-MoE transformer by replacing all MLP layers with MoE layers. After the over-growing, we calculate the loss decrease. In particular, given f representing a vision transformer with L -layers. Thus, $\mathcal{L}_t(f_{\text{non-MoE}})$ is the training loss for the pretrained non-MoE vision transformer and $\mathcal{L}_t(f_{\text{RMoE}}; \theta_r)$ represents the loss after introducing MoE layers into a non-MoE model in RMoE training, with residual weight θ_r .

As we mentioned before, we add a small noise over weight of MLP to initialize the experts weights, it is equivalent to initialize $\theta_r \leftarrow \epsilon \theta_0$, where ϵ is a small enough value that only perturbs the network and output in a small range. Thus, the loss decrease can be decomposed via Taylor approximation as:

$$\mathcal{L}_t(f_{\text{RMoE}}; \theta_r) = \mathcal{L}_t(f_{\text{non-MoE}}) + \epsilon \sum_{l=1}^L \theta_r^l s_l + \epsilon^2 O(\theta_r). \quad (8)$$

$$s_l \approx \nabla_{\theta_r^l} \mathcal{L}_t(f_{\text{RMoE}}; \theta_r).$$

Next, we find the most important N layers to decrease the loss. We achieve this by first optimizing the initial of θ_r for several gradient descent step and because the residual weight θ_r is initialized with a small enough

factor ϵ :

$$\hat{\theta}_r = \underset{\theta_r}{\operatorname{argmin}} \left\{ \epsilon \sum_{l=1}^L \theta_r^l s_l \quad \text{s.t.} \quad \|\theta_r\|_0 \leq N \right\}, \quad (9)$$

where $\|\theta_r\|_0 := \sum_{l=1}^L \mathbb{I}(\theta_r^l \neq 0)$. To select the best layers to maximally decrease loss function, the optimal solution is selecting the layer with largest N gradient magnitude $|s_l|$. In practice, we want to get a general scaled up model at once for all the downstream tasks, so we calculate the the gradient magnitude on all the target downstream tasks and select the N layers with top high scores sum among all the tasks.

5 Experiment

In this section, we evaluate the effectiveness and efficiency of RMoE.

We first deliver a comprehensive comparison of different training pipelines, i.e., non-MoE, RMoE, and MoE. Then, to show the capability of RMoE to train large vision transformers, we use RMoE to train models using Swin-L and BeiT-L [2] backbones. Finally, we conduct a set of ablation studies to compare different design choices of RMoE.

Model	Type	ADE20K		GPU Days		MS-COCO		GPU Days	
		mIoU	mIoU (ms+flip)	Params	Scratch / Pretrained	AP	Params	Scratch / Pretrained	
Swin-T	Non-MoE	44.6	45.9	60M	56.3 / 6.3	41.6	39M	52.6 / 2.6	
	RMoE-D	45.3	46.6	103M	56.8 / 6.7	42.6	82M	53.0 / 2.9	
	RMoE-I (1k)	45.6	46.9	103M	57.3 / 7.1	42.9	82M	53.5 / 3.4	
	RMoE-I	45.7	47.2	103M	57.7 / 7.5	43.0	82M	53.9 / 3.8	
	MoE	45.9	47.3	120M	75.1 / 75.1	43.1	99M	71.8 / 71.8	
CvT-13	Non-MoE	44.9	46.4	45M	46.2 / 6.2	38.3	29M	42.4 / 2.4	
	RMoE-D	45.4	46.9	94M	46.8 / 6.6	39.6	78M	42.9 / 2.6	
	RMoE-I (1k)	45.6	47.0	94M	47.3 / 7.0	39.7	78M	43.5 / 3.3	
	RMoE-I	45.8	47.2	94M	47.6 / 7.3	39.9	78M	43.8 / 3.6	
	MoE	46.0	47.4	119M	60.4 / 60.4	40.1	103M	56.1 / 56.1	

Table 1: A comprehensive comparison between different training pipelines including non-MoE, MoE, RMoE-I and RMoE-D on ADE20K segmentation task and MS-COCO object detection task. RMoE-I (1k) represents intermediate finetune the model on ImageNet 1k instead of ImageNet 22k. Non-MoE represents using the original transformers as backbones. The GPU Days measures the total training time in one Nvidia Tesla V100-32GB GPU. ‘Scratch’ represents that the pipeline is fully performed from upstream pretrain to downstream finetune. ‘Pretrained’ means loading from existing pretrained upstream **non-MoE** checkpoints, which is usually available in the computer vision community. Notice that Swin-T and CvT-13 has a different batch size during upstream training, so the GPU Days does not measure the speed relationship bewteen two different backbones.

5.1 Comparing Different Training Pipelines

We first compare different training pipelines in Figure 2, i.e., non-MoE, MoE, RMoE-I and RMoE-D to demonstrate the effectiveness of RMoE. In each training pipeline, we used the image classification on ImageNet22k [13] as the upstream pretraining. We adopted two representative downstream tasks, i.e., semantic segmentation and object detection. The model performance on semantic segmentation and object detection are evaluated on ADE20K [63] and MS-COCO [37], respectively. For backbones, we selected two representative vision transformers, Swin-T and CvT-13. For decoder head, we employ UperNet [56] and RetinaNet [36] for segmentation and detection tasks.

Training Setting. For upstream pretraining, we followed the training strategy in the original papers [38, 54]. Specifically, all upstream tasks were trained for 90 epochs with the input resolution 224×224 . We set the batch size to 1024 for Swin-T and 2048 for CvT-13 following the original settings and optimized them using AdamW [39] optimizer with an initial learning rate 0.001 for Swin-T and 0.01 for CvT-13. A cosine learning decay scheduler is used along with the training. The weight decay is 0.05.

For intermediate finetuning stage of RMoE-I, we reduced the non-MoE model training epochs for 5 and finetuned the MoE model on ImageNet22K for additional 5 epochs to align the total number of epochs with other settings. The initial learning rate was set to 0.0001 and a cosine learning rate decay was used. Other setting are the same as the upstream training. We also apply the intermediate finetune on ImageNet 1k for 30 epochs and 384×384 resolution for a more viable setup. We mark it as RMoE-I (1k) in Table 1. The settings in downstream finetuning stage are the same with those in Swin Transformer for both Swin-T and CvT-13 on the semantic segmentation task expect a longer training steps $160k \rightarrow 200k$ for a fully converge for RMoE. We applied the same training strategy for all the models on ADE20K. For object detection on MS-COCO, we trained all the models using AdamW optimizer with learning rate 0.0001 and regular $1 \times$ schedule without multi-scaled augmentation.

MoE Settings. For all the MoE models, we used 8 MLP experts with a switch gate [18], which means that each token is only routed to one expert in an MoE layer. For MoE training, we followed the settings of the state-of-the-art MoE training pipeline [41].

Specially, for CvT-13, we add MoE layers after stage 2 to stabilize the training.

For RMoE training, we add a random noise $\epsilon = 0.01$ on each expert after initialized from a non-MoE model to encourage diversity.

We calculated the gradient-based grow score and selected top-3 score layers to apply MoE. The detailed layer selections can be found in Table 2. For both MoE and RMoE training, we applied the load balance loss in Sec. 3 with a weight $w_{\text{balance}} = 0.01$ during upstream training and intermediate finetune, a $w_{\text{balance}} = 0.0001$ weight is added during downstream finetune.

Experiment Results. As we can see in Table 1, the original MoE training gets the best performance on all of the tasks and models but with highest training cost. The performance of RMoE-I is comparable to MoE with saving around 30% training cost (training from scratch). RMoE-D performs slightly worse than RMoE-I, but both outperform the non-MoE training with minor training cost increase. On Swin-T and CvT-13 backbones, compared with non-MoE baselines, RMoE-I gets +1.1 / 0.9 mIoU. It also gets + 1.4 / 1.6 AP on the detection task. RMoE-D, which directly uses experts on the downstream tasks, can still gets +0.7 / 0.5 mIoU and +1.0 / 1.3 AP. In practice, we can leverage the shared checkpoints of Swin-T and CVT-13 to further reduce the training cost. In that case, we can use RMoE to scale up a vision transformer for

downstream tasks with less than 10% of the training cost of the MoE training.

Stage	Swin-T	CvT-13	Swin-L	BeiT-L
Stage 1	-	-	2	8, 10, 12, 16, 17, 18, 20, 24
Stage 2	2	2	2	N/A
Stage 3	6	3, 9	9, 12, 18	N/A
Stage 4	2	N/A	2	N/A

Table 2: Layer selections in RMoE training for downstream tasks. For example, in Swin-T, we select the 2nd layer in stage 2,4 and the 6th layer in stage 3 as MoE layers. Notice that CvT only has 3 stages, and BeiT does not have stages. So for alignment, we represent CvT13 using Stage 1,2,3 and BeiT-L only using Stage 1.

Analysis of the Gap between RMoE-I and RMoE-D. The balancing loss is found to be critical for explaining the performance difference between RMoE-I and RMoE-D. In the finetuning stage of MoE training pipeline, load balancing loss is often not added. For example, in V-MoE, it is claimed that the well-trained router can balance each expert well without a balancing loss. However, in RMoE-D, we find that the newly introduced decoder head and a large initial learning rate in downstream tasks can break the load balance among experts. To demonstrate this, we compare the training processes of MoE, RMoE-D and RMoE-I w/ and w/o the balance loss in the downstream finetuning stage. As shown in Figure 4, we find that w/o balance loss, all the methods, including MoE, result in a bad load balance. When we add the balance loss with a small weight $w_{\text{balance}} = 0.0001$, the load balance in the RMoE-D is still bad. RMoE-I, which benefits from an intermediate finetune, can smoothly decrease the balance loss. A further large weight $w_{\text{balance}} = 0.1$ can make RMoE-D balanced but it would hurt the performance as well.

Expert Specialization. To analyze how experts distribute different images, we visualize how many images of a given ImageNet class use each expert in Figure 5. In particular, each row is an ImageNet class and each column is an expert. The color in one cell denotes the average routing weight (the largest $G(x)$ in Eq. 1) for all the tokens in a specific class for an expert. Thus, darker color in a cell means that the expert is specialized to tackle the images in that class. We compared both RMoE and MoE training using RMoE-I (1k) intermediate finetune setting and selected the last MoE layer to analyze. We find that experts in both RMoE and MoE training are specialized across the classes in ImageNet. It echos our observation in Sec. 4.1 that, although RMoE uses much less computation resource to train the experts, expert specialization is achieved by both RMoE and MoE training.

5.2 Using RMoE to Train Large Transformers

We further demonstrate the capability of RMoE to further scale up large vision transformers, including Swin-L and BeiT-L, because they achieve state-of-the-art results on downstream tasks.

In particular, we applied RMoE to Swin-L w/ UperNet [56] and Maskformer [7], BeiT-L w/ UperNet for ADE20K segmentation and Swin-L w/ HTC++ [4, 38] on MS-COCO object detection.

Experiment Setup. Training hyper parameters are the same with those in the finetuning setting of the Swin-L, Maskformer and BeiT-L. For the HTC++ decoder head, we reduced the schedule from $6\times$ to $3\times$ since the scaled-up model has a better converging speed. For intermediate finetune, the weights of experts directly

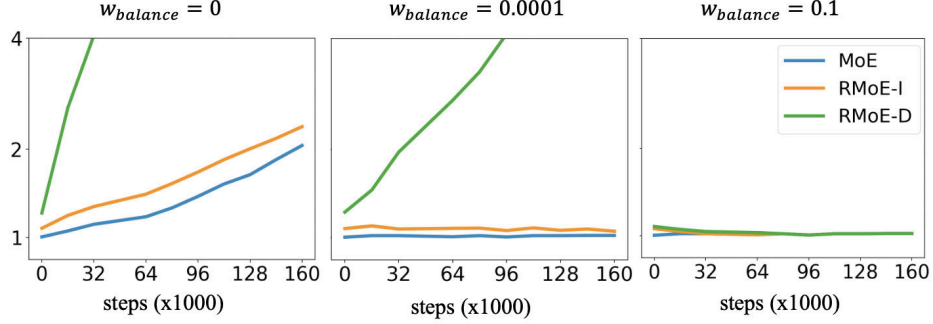


Figure 4: Balance loss on the ADE20K segmentation tasks with different loss weights. Weight=0 denotes removing the balance loss. Both RMoE-I and MoE can get a good balance under a small weight, but RMoE-D can only get balanced when a large balance loss weight is used.

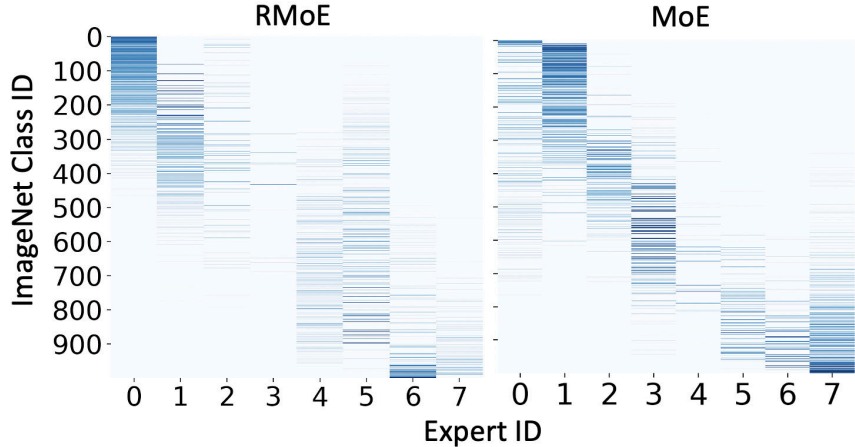


Figure 5: Visualization of experts specialization of RMoE and MoE training. A darker color in a cell indicates that the expert is more specialized to tackle the images in that class of ImageNet. We separately reorder the ImageNet classes for RMoE and MoE to get a better visualization.

inherit from the official pretrained checkpoint^{1 2}. The rest of the settings keep the same as in Section 5.1. For Swin-L, we chose 6 layers and for BeiT-L we chose 8 layers as MoE layers using the gradient-based grow score. The detailed layer selections are shown in Table 2. Other settings related to RMoE are the same as Section 5.1.

Experiment Results. As we can see from Table 3, RMoE-I can further scale up the backbones to larger model capacity, with a performance gain at around 1.0 on Swin-L segmentation, We also improve BeiT-L segmentation result by 0.4. RMoE-D can also get a +0.7 improvement compared with the non-MoE model for a nearly free cost on Swin-L model with different Head. For the object detection task, the results shown in Table 4 indicate that RMoE-I and RMoE-D get a +0.6 / 0.4 higher AP compared with non-MoE model baseline with only half finetuning epochs.

¹<https://github.com/microsoft/Swin-Transformer>

²<https://github.com/microsoft/unilm/tree/master/beit>

Backbone	Head	Type	Params	mIoU	mIoU (ms+flip)
Swin-L	UperNet	Non-MoE	234M	52.0	53.5
		RMoE-D	476M	52.7	54.2
		RMoE-I	476M	52.9	54.4
Swin-L	Mask Former	Non-MoE	212M	54.0 [†]	55.6
		RMoE-D	454M	54.7	56.3
		RMoE-I	454M	55.0	55.7
BeiT-L	UperNet	Non-MoE	502M	56.4 [†]	57.0
		RMoE-D	737M	56.7	57.1
		RMoE-I	737M	56.9	57.4

Table 3: ADE-20K result on various of large backbones and decoder heads. [†] Our result on MaskFormer and BeiT-L UperNet without TTA is slightly lower than the result reported in original papers. However, the TTA results keep the same.

Type	Head	Params	AP ^{box}	AP ^{mask}
Non-MoE	HTC++ (6x)	284M	57.1	49.5
RMoE-D	HTC++ (3x)	526M	57.5	49.8
RMoE-I	HTC++ (3x)	526M	57.7	50.0

Table 4: Non-MoE and RMoE result of MS-COCO object detection on Swin-L and HTC++.

5.3 Inference Speed and FLOPs

An appealing property for MoE transformer is that it enlarges the model capacity (number of parameter) by only introducing minimal additional inference time and FLOPs than non-MoE transformers. RMoE also has this property. As shown in Table 5, we analyzed the inference time and FLOPs of Swin-T and Swin-L on ADE20K segmentation tasks using UperNet decoder head. We show that RMoE has an obvious performance gain but only introduces minimal additional inference time, which is critical in machine learning models deployment.

5.4 Ablation Study

We performed a series of ablation studies for the RMoE to compare different design choices. The ablation studies were conducted on Swin-T using UperNet on ADE20K segmentation and RetinaNet on MS-COCO object detection. We used RMoE-I as the baseline due to its better performance and training cost trade-off.

MoE layer selection. As one of the key factors, the MoE layer selection in the transformers influences the final performance and model size. We report different layers choices in Table 6. In those settings, Last-2 and Every-2 are introduced in [41]. Last-2 places MoE in the last 2 even blocks and Every-2 places MoE in every other layer. We also introduce the Every-Last, which expands the layer in the last layer of each stage in Swin-T Transformer with the intuition that the layer closer to the decode head is more important. Table 6 shows the result for different strategies. We find that our gradient score-based selection performs better than Last-2 and Every-2 while keeping the same performance compared with Every-Last. For downstream tasks,

Backbone	Type	mIoU	Inference time (img/s)	FLOPs
Swin-T	Non-MoE	44.6	29.0	945G
	RMoE-I	45.7	27.2	945G
Swin-L	Non-MoE	52.0	16.8	3230G
	RMoE-I	52.9	15.4	3230G

Table 5: Comparison of mIoU against inference time and FLOPs between non-MoE, RMoE-I on ADE20K with UperNet decode head. We test the inference speed on a single RTX3090.

	# Layers	Backbone Params	mIoU	AP
Ours	3	71M	45.7	43.0
Last-2	2	69M	45.3	42.4
Every-2	6	88M	45.4	42.9
Every-Last	4	72M	45.7	43.0

k	n	Backbone Params	Backbone FLOPs	mIoU	AP
1	8	71M	4.7G	45.7	43.0
2	8	71M	7.4G	45.8	43.3
1	4	47M	4.7G	45.4	42.7
1	16	121M	4.7G	45.7	42.9

Table 6: Ablation study on different MoE layers selections.

Table 7: Ablation study on different top- k gate and number of experts n .

every stage’s last layer feature will be input into the decode head, so Last-2 is no longer a good choice. It also explains the good performance of Every-Last. Besides, we further optimize Every-last by one layer less, showing the advantages of score-based method. On large backbones, Every-Last can only use four MoE layers, which may be insufficient. For Every-2, the performance slightly drops compared with ours. We think that it may be because Every-2 needs three additional MoE layers while some of the intermediate layers can not contribute a lot to the performance.

Number of Experts and Top-K Gate. As common hyper-parameters, the number of experts n and the k of the top- k operation in a gate are usually considered in the MoE experiment. In Table 7 we compare our setting with other settings, i.e., $n = 4, 16$ and $k = 2$. We see that $k = 2$ improves the performance on both the downstream tasks while increasing training time for 10% and a little more FLOPs. With the fewer experts, $n = 4$ decreased the performance brought by scaled-up with RMoE-I for around 0.3 on ADE20K and MS-COCO. $n = 16$ does not improve the performance, which may differ from the conclusions in other MoE literature. We speculate that this is mainly due to the load balancing loss used [41]. It may hurt each expert’s performance in the finetuning stage. A more dedicated finetuning process with improved balancing loss needs to be designed when the number of experts increases, e.g., 128 in [18] or 32 in [41].

Stop Gradient. We apply the stop gradient technique in Eq. 7 for RMoE-I intermediate finetune to avoid a performance drop. As shown in Table 8, the performance drops a lot without the stop gradient in the intermediate finetune.

Noise Initialization Scale. To encourage the specialization for each experts after applying RMoE, we apply a certain noise level on each copied weight. Table 9 shows how different noise level affects the final result.

6 Inference Speed and FLOPs

An appealing property for MoE transformer is that it enlarges the model capacity (number of parameter) by only introducing minimal additional inference time and FLOPs than non-MoE transformers. RMoE also has this property. As shown in Table 5, we analyzed the the inference time and FLOPs of Swin-

	mIoU	AP
w/ Stop Gradient	45.7	43.0
w/o Stop Gradient	45.2	42.1

Table 8: Ablation study on the stop gradient technique.

ϵ	mIoU	AP
0	44.9	42.5
0.001	45.3	42.7
0.01	45.7	43.0
0.1	44.4	41.5

Table 9: Different noise level against ADE20k performance.

T and Swin-L on ADE20K segmentation tasks using UperNet decoder head. We show that RMoE has an obvious performance gain but only introduces minimal additional inference time, which is critical in machine learning models deployment.

7 Viable Setting for RMoE

From the ablation studies, we conclude an effective but simple baseline for practitioners, i.e., the Every-Last strategy for choosing MoE layers, top-1 gate, and 8 experts. According to Table 5 in the main paper, compared with the best performance RMoE-I with firefly splitting, this configuration get the same performance with only 1M additional parameters on the backbone.

8 Conclusion

In this paper, we propose an efficient training pipeline for MoE vision transformers, RMoE. Compared with traditional MoE training suffering from high computation cost, we train the different residuals for each expert by initializing the weights from the pretrained non-MoE model. This is motivated by analyzing the training trajectory of the weights, where we find that experts weights can be factored as a weight core and a residual. We have comprehensive studies on various transformers, including Swin Transformer, CvT and BeiT, on different downstream tasks, including ADE20K segmentation and COCO object detection. We show that our RMoE can improve the performance of regular non-MoE vision transformers with minor additional cost.

References

- [1] Abbas, A., Andreopoulos, Y.: Biased mixtures of experts: Enabling computer vision inference under data transfer limitations. *IEEE Transactions on Image Processing* **29**, 7656–7667 (2020)
- [2] Bao, H., Dong, L., Wei, F.: Beit: Bert pre-training of image transformers. *arXiv preprint arXiv:2106.08254* (2021)
- [3] Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020)
- [4] Chen, K., Pang, J., Wang, J., Xiong, Y., Li, X., Sun, S., Feng, W., Liu, Z., Shi, J., Ouyang, W., et al.: Hybrid task cascade for instance segmentation. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. pp. 4974–4983 (2019)
- [5] Chen, K., Wang, J., Pang, J., Cao, Y., Xiong, Y., Li, X., Sun, S., Feng, W., Liu, Z., Xu, J., Zhang, Z., Cheng, D., Zhu, C., Cheng, T., Zhao, Q., Li, B., Lu, X., Zhu, R., Wu, Y., Dai, J., Wang, J., Shi, J., Ouyang, W., Loy, C.C., Lin, D.: MMDetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155* (2019)
- [6] Chen, K., Xu, L., Chi, H.: Improved learning algorithms for mixture of experts in multiclass classification. *Neural networks* **12**(9), 1229–1252 (1999)
- [7] Cheng, B., Schwing, A.G., Kirillov, A.: Per-pixel classification is not all you need for semantic segmentation. *arXiv* (2021)
- [8] Chu, X., Tian, Z., Wang, Y., Zhang, B., Ren, H., Wei, X., Xia, H., Shen, C.: Twins: Revisiting spatial attention design in vision transformers. *arXiv preprint arXiv:2104.13840* (2021)
- [9] Chu, X., Zhang, B., Tian, Z., Wei, X., Xia, H.: Do we really need explicit position encodings for vision transformers? *arXiv e-prints pp. arXiv–2102* (2021)
- [10] Collobert, R., Bengio, S., Bengio, Y.: A parallel mixture of svms for very large scale problems. *Advances in Neural Information Processing Systems* **14** (2001)
- [11] Contributors, M.: Mmsegmentation, an open source semantic segmentation toolbox. <https://github.com/open-mmlab/msegmentation> (2020)
- [12] Deisenroth, M., Ng, J.W.: Distributed gaussian processes. In: *International Conference on Machine Learning*. pp. 1481–1490. PMLR (2015)
- [13] Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: *2009 IEEE conference on computer vision and pattern recognition*. pp. 248–255. Ieee (2009)
- [14] 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 preprint arXiv:2107.00652* (2021)

- [15] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al.: An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929 (2020)
- [16] Eigen, D., Ranzato, M., Sutskever, I.: Learning factored representations in a deep mixture of experts. arXiv preprint arXiv:1312.4314 (2013)
- [17] El-Nouby, A., Neverova, N., Laptev, I., Jégou, H.: Training vision transformers for image retrieval. arXiv preprint arXiv:2102.05644 (2021)
- [18] Fedus, W., Zoph, B., Shazeer, N.: Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. arXiv preprint arXiv:2101.03961 (2021)
- [19] Gross, S., Ranzato, M., Szlam, A.: Hard mixtures of experts for large scale weakly supervised vision. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 6865–6873 (2017)
- [20] Guzman-Rivera, A., Batra, D., Kohli, P.: Multiple choice learning: Learning to produce multiple structured outputs. Advances in neural information processing systems **25** (2012)
- [21] Han, K., Xiao, A., Wu, E., Guo, J., Xu, C., Wang, Y.: Transformer in transformer. arXiv preprint arXiv:2103.00112 (2021)
- [22] Hansen, J.V.: Combining predictors: comparison of five meta machine learning methods. Information Sciences **119**(1-2), 91–105 (1999)
- [23] He, J., Qiu, J., Zeng, A., Yang, Z., Zhai, J., Tang, J.: Fastmoe: A fast mixture-of-expert training system. arXiv preprint arXiv:2103.13262 (2021)
- [24] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
- [25] He, S., Luo, H., Wang, P., Wang, F., Li, H., Jiang, W.: Transreid: Transformer-based object re-identification. arXiv preprint arXiv:2102.04378 (2021)
- [26] Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017)
- [27] Hu, J., Shen, L., Sun, G.: Squeeze-and-excitation networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 7132–7141 (2018)
- [28] Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4700–4708 (2017)
- [29] Jacobs, R.A., Jordan, M.I., Nowlan, S.J., Hinton, G.E.: Adaptive mixtures of local experts. Neural computation **3**(1), 79–87 (1991)
- [30] Jiang, Z., Hou, Q., Yuan, L., Zhou, D., Jin, X., Wang, A., Feng, J.: Token labeling: Training a 85.5% top-1 accuracy vision transformer with 56m parameters on imagenet. arXiv preprint arXiv:2104.10858 (2021)

- [31] Jordan, M.I., Jacobs, R.A.: Hierarchical mixtures of experts and the em algorithm. *Neural computation* **6**(2), 181–214 (1994)
- [32] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* **25**, 1097–1105 (2012)
- [33] Lee, K., Hwang, C., Park, K., Shin, J.: Confident multiple choice learning. In: *International Conference on Machine Learning*. pp. 2014–2023. PMLR (2017)
- [34] Lee, S., Purushwalkam Shiva Prakash, S., Cogswell, M., Ranjan, V., Crandall, D., Batra, D.: Stochastic multiple choice learning for training diverse deep ensembles. *Advances in Neural Information Processing Systems* **29** (2016)
- [35] Lepikhin, D., Lee, H., Xu, Y., Chen, D., Firat, O., Huang, Y., Krikun, M., Shazeer, N., Chen, Z.: Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668* (2020)
- [36] Lin, T.Y., Goyal, P., Girshick, R., He, K., Dollár, P.: Focal loss for dense object detection. In: *Proceedings of the IEEE international conference on computer vision*. pp. 2980–2988 (2017)
- [37] Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft coco: Common objects in context. In: *European conference on computer vision*. pp. 740–755. Springer (2014)
- [38] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., Guo, B.: Swin transformer: Hierarchical vision transformer using shifted windows. *arXiv preprint arXiv:2103.14030* (2021)
- [39] Loshchilov, I., Hutter, F.: Decoupled weight decay regularization (2019)
- [40] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., Liu, P.J.: Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683* (2019)
- [41] Riquelme, C., Puigcerver, J., Mustafa, B., Neumann, M., Jenatton, R., Pinto, A.S., Keyser, D., Houlsby, N.: Scaling vision with sparse mixture of experts. *arXiv preprint arXiv:2106.05974* (2021)
- [42] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 4510–4520 (2018)
- [43] Shahbaba, B., Neal, R.: Nonlinear models using dirichlet process mixtures. *Journal of Machine Learning Research* **10**(8) (2009)
- [44] Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., Dean, J.: Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538* (2017)
- [45] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014)
- [46] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 1–9 (2015)

- [47] Tan, M., Le, Q.: Efficientnet: Rethinking model scaling for convolutional neural networks. In: International Conference on Machine Learning. pp. 6105–6114. PMLR (2019)
- [48] Theis, L., Bethge, M.: Generative image modeling using spatial lstms. *Advances in neural information processing systems* **28** (2015)
- [49] Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., Jégou, H.: Training data-efficient image transformers & distillation through attention. *arXiv preprint arXiv:2012.12877* (2020)
- [50] Touvron, H., Cord, M., Sablayrolles, A., Synnaeve, G., Jégou, H.: Going deeper with image transformers. *arXiv preprint arXiv:2103.17239* (2021)
- [51] Tresp, V.: Mixtures of gaussian processes. *Advances in neural information processing systems* **13** (2000)
- [52] Wang, W., Xie, E., Li, X., Fan, D.P., Song, K., Liang, D., Lu, T., Luo, P., Shao, L.: Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. *arXiv preprint arXiv:2102.12122* (2021)
- [53] Wang, X., Yu, F., Dunlap, L., Ma, Y.A., Wang, R., Mirhoseini, A., Darrell, T., Gonzalez, J.E.: Deep mixture of experts via shallow embedding. In: *Uncertainty in Artificial Intelligence*. pp. 552–562. PMLR (2020)
- [54] Wu, H., Xiao, B., Codella, N., Liu, M., Dai, X., Yuan, L., Zhang, L.: Cvt: Introducing convolutions to vision transformers. *arXiv preprint arXiv:2103.15808* (2021)
- [55] Wu, L., Liu, B., Stone, P., Liu, Q.: Firefly neural architecture descent: a general approach for growing neural networks. *arXiv preprint arXiv:2102.08574* (2021)
- [56] Xiao, T., Liu, Y., Zhou, B., Jiang, Y., Sun, J.: Unified perceptual parsing for scene understanding. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. pp. 418–434 (2018)
- [57] Xu, W., Xu, Y., Chang, T., Tu, Z.: Co-scale conv-attentional image transformers. *arXiv preprint arXiv:2104.06399* (2021)
- [58] Yang, B., Bender, G., Le, Q.V., Ngiam, J.: Condconv: Conditionally parameterized convolutions for efficient inference. *arXiv preprint arXiv:1904.04971* (2019)
- [59] Yuan, K., Guo, S., Liu, Z., Zhou, A., Yu, F., Wu, W.: Incorporating convolution designs into visual transformers. *arXiv preprint arXiv:2103.11816* (2021)
- [60] Yuan, L., Chen, Y., Wang, T., Yu, W., Shi, Y., Jiang, Z., Tay, F.E., Feng, J., Yan, S.: Tokens-to-token vit: Training vision transformers from scratch on imagenet. *arXiv preprint arXiv:2101.11986* (2021)
- [61] Yuksel, S.E., Wilson, J.N., Gader, P.D.: Twenty years of mixture of experts. *IEEE transactions on neural networks and learning systems* **23**(8), 1177–1193 (2012)
- [62] Zhai, X., Kolesnikov, A., Houlsby, N., Beyer, L.: Scaling vision transformers. *arXiv preprint arXiv:2106.04560* (2021)
- [63] Zhou, B., Zhao, H., Puig, X., Xiao, T., Fidler, S., Barriuso, A., Torralba, A.: Semantic understanding of scenes through the ade20k dataset. *International Journal of Computer Vision* **127**(3), 302–321 (2019)

Algorithm 1 Insert RMoE to vision transformer for downstream task

- 1: **Input:** a well-trained L -layer vision transformer f , the loss functions on m given downstream tasks $\mathcal{L}_t(f), t = 1 \dots m$, maximum number layer to expand \mathcal{L}_{\max} , number of experts to expand n . Stage to expand (Intermediate, Downstream).
 - 2: Begin before the given stage.
 - 3: For $t = 1 \dots m$, calculate the score for each layers using Eq. 8 and sum them together along the tasks as the final score $s_l = \sum_{i=1}^m |s_l|_i$ where $|s_l|_i$ is layer l score for task i .
 - 4: Select the top- \mathcal{L}_{\max} score, apply RMoE to expand the MLP layer to n -expert RMoE at the corresponding layer with the top score.
 - 5: Finetune the expanded RMoE model on the downstream tasks.
-

A Detail for Firefly Splitting pipeline

Given a trained non-MoE vision transformer, we employ the Firefly approach to select which MLP layers to be replaced by MoE layers.

In particular, it aims at minimizing the loss decrease between non-MoE model f_{MLP} and the RMoE model $f_{\text{RMoE}}; \theta_r, \theta_g$:

$$\max_{\theta_r, \theta_g} \{\mathcal{L}(f_{\text{MLP}}) - \mathcal{L}(f_{\text{RMoE}}; \theta_r, \theta_g)\}, \quad (10)$$

where θ_r is the weights residual and θ_g is the gate parameters. The optimization problem is solved by gradient descend. To accelerate the optimization, θ_r is initialized as $\theta_r \leftarrow \epsilon \theta_0$, where $\epsilon = 0.001$ in our experiments. Such initialization combined with the stop gradient technique ensures the performance will not drop too much initially, e.g., $\mathcal{L}(f_{\text{MLP}}) = \mathcal{L}(f_{\text{RMoE}}; \theta_r, \theta_g)$ when $\epsilon = 0$.

In practice, to save computation cost and avoid overfitting, we do not add MoE layers in all transformer blocks, but select a subset of layers $\{l\}$ with residual weights $\{\theta_r^l\} \subset \theta_r$, which contribute most to Eq.10. To this end, we follow Firefly and use a two-step optimization to find a sparse solution for Eq.10:

Step one. Generate an over-grown MoE model by replacing all MLP layers with MoE layers and optimize θ_r, θ_g using gradient descent for a few steps.

Step two. Fix θ_g , re-optimize θ_r by selecting the best candidates θ_r^l . To achieve this, we use Taylor expansion and choose the largest candidates set follows Eq.8 and Eq.9. We summarize the overall RMoE gradient grow based algorithm in Algorithm 1.

B Expert Visualization

To better understand what experts learn in RMoE training, we present the visualization of the patch routing, which represents how each patch in the input token sequence is routed by the gate. In an MoE model, each expert is trying to learn different functions to process different kind of input tokens for the input images. Thus, we can use this visualization to qualitatively compare the model trained with RMoE and MoE.

To visualize the patch routing, we record the patch routing result for each expert and reshape the input patch token sequence to the image-like shape of the current transformer stage. In this way, we can directly visualize the patch routing condition in an image fashion as shown in Figure 6. In the figure, the white square

represents the patch is routed to the this expert. The black square means the patch is not routed to this expert. We choose top-4 experts with the most number of routed patches to provide a clearer visualization.

We use Swin-T as backbone and visualize the token routing in the last layer of stage 3. We choose this layer because stage 3 has a proper resolution to visualize, and its feature map contains more semantic information compared with the shallow stages. For segmentation and detection tasks, we use the same setting in Section 5. For RMoE, we use RMoE-D models trained on the downstream tasks.

From Figure 6, we see that RMoE and MoE share similar patterns in patch routing, for example, the foreground and background are routed into different experts. These indicate the experts of the RMoE share similar functions as the MoE experts.

C Additional Implementation detail

For downstream training, we build our code base on the MMSegmentation [11] and MMDetection [5]. For MoE implementation, we use FastMoE [23] as basic code.

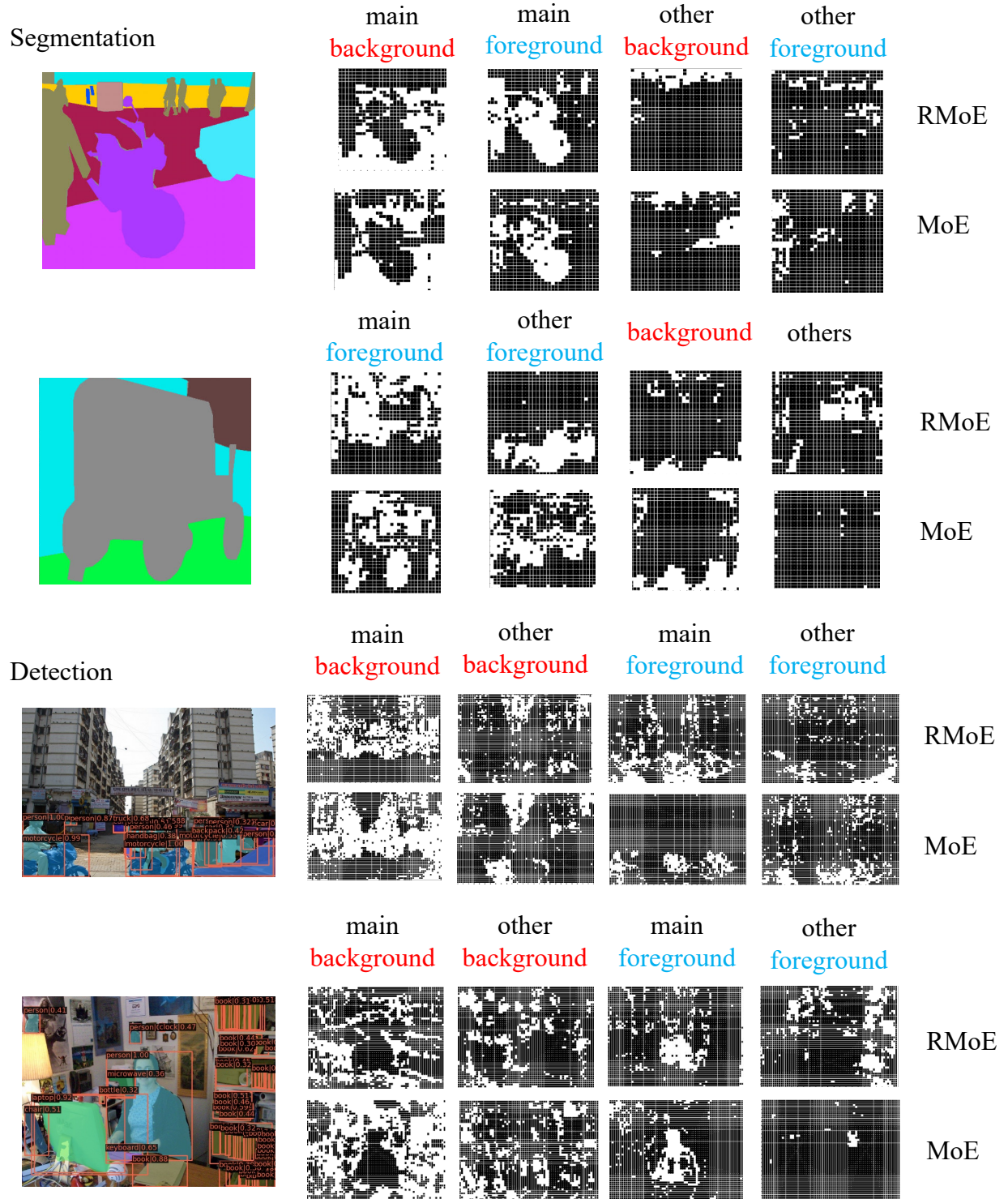


Figure 6: Patch routing visualization for the last layer of Swin-T stage 3 in segmentation and detection tasks. We define the main/other mainly based on number of patches.