Layer Compression of Deep Networks with Straight Flows

Chengyue Gong[1], **Xiaocong Du**[2], **Bhargav Bhushanam**[2], **Lemeng Wu**[1], **Xingchao Liu**[1], **Dhruv Choudhary**[2], **Arun Kejariwal**[2], **Qiang Liu**[1]

[1] University of Texas at Austin,
[2] Meta, Inc.

Abstract

Very deep neural networks lead to significantly better performance on various real tasks. However, it usually causes slow inference and is hard to be deployed on real-world devices. How to reduce the number of layers to save memory and to accelerate the inference is an eye-catching topic. In this work, we introduce an intermediate objective, a continuous-time network, before distilling deep networks into shallow networks. First, we distill a given deep network into a continuous-time neural flow model, which can be discretized with an ODE solver and the inference requires passing through the network multiple times. By forcing the flow transport trajectory to be straight lines, we find that it is easier to compress the infinite step model into a one-step neural flow model, which only requires passing through the flow model once. Secondly, we refine the one-step flow model together with the final head layer with knowledge distillation and finally, we can replace the given deep network with this one-step flow network. Empirically, we demonstrate that our method outperforms direct distillation and other baselines on different model architectures (e.g. ResNet, ViT) on image classification and semantic segmentation tasks. We also manifest that our distilled model naturally serves as an early-exit dynamic inference model.

Introduction

It has been widely observed that increasing model depth leads to significantly better performance on various real tasks, *e.g.*, image generation (Rombach et al. 2021; Liu et al. 2021a), image classification (Dosovitskiy et al. 2020), and NLP applications (Brown et al. 2020; Xue et al. 2020). However, very deep networks increase the cost during inference, and some of them even require distributedly deployment to multiple nodes. To alleviate this problem, there has been a growing interest in developing algorithms for compressing deep models into shallow or thin networks, using pruning or knowledge distillation (KD) techniques.

In this work, we propose a new method to train shallow and efficient neural networks. Compare to thin but deep networks, shallow networks are believed to be easier to get deployed with faster inference speed on the Internet of things (IoT) devices (Mandal et al. 2019). Thus, how to convert a well-trained deep network into a shallow version without loss of accuracy, is a critical problem. Considering tiny networks are not easy to get trained well in practice (Graham et al. 2021; Gong et al. 2021), knowledge distillation (KD) is usually leveraged to greatly improve model performance. In this work, we are especially interested in building up a general and theory-guided algorithm to derive a shallow network, which can be regarded as a kind of knowledge distillation. Inspired by recent works in straight neural flow models, we propose to extend this method from image generation to model compression.

Correspondingly, our training method consists of three phases: ① Instead of directly distilling from a deep teacher to a shallow student, we first learn a time-continuous model to mimic a given teacher model. ② Then, using the recent methods in straight and transport-cost aware flows (Liu, Gong, and Liu 2022; Lipman et al. 2022), we apply ReFlow operator (Liu, Gong, and Liu 2022) to reduce the transport cost which makes the time-continuous network easy to compress to one step (*a.k.a.*, fast simulation). ③ Finally, we use KD to refine the one-step model to yield even better performance. Similar to existing works for training shallow or weight-shared networks (Yang et al. 2021; Sun et al. 2020), we create multi-stage training and have additional objectives before the final-step KD. Different from previous heuristic strategies ,*e.g.*, weight-sharing networks (Yang et al. 2021), layer-wise progressively distillation (Sun et al. 2020), our method is inspired by recent studies in neural flow models. Viewing reducing the number of simulation steps as the same as compressing the number of blocks, we convert compression to reducing simulation steps for neural flow models. Notice that reducing transport cost comes to an easy-to-compress flow model. We propose to first learn an easy-to-compress flow model by eliminating the transport cost; after having a flow with low transport cost, we then refine a one-step simulation flow model with knowledge distillation.

In the following part, we first discuss the background and related works in Section . After introducing the method in Section , we verify the performance on multiple datasets and tasks in Section . We evaluate our model performance on CIFAR-10 and ImageNet, upon vision transformers and ResNet. We then transfer the model to the semantic segmentation task, to test whether ours also benefits the downstream tasks. We also demonstrate that our flow model can serve as an early-exit dynamic network. We finally summarize our results and discuss the weakness and possible future directions in Section .

Background

Straight Flows A large portion of machine learning problems can be treated as transporting one distribution to another, e.g. generation, representation learning. Recently, advances have been made by representing the transport as a continuous time process, with neural ordinary differential equations (ODEs) (Chen et al. 2018; Papamakarios et al. 2021; Song, Meng, and Ermon 2020) or diffusion models by stochastic differential equations (SDEs) (Song et al. 2020; Ho, Jain, and Abbeel 2020; Tzen and Raginsky 2019; Vargas et al. 2021). During training, a neural model is learned to represent the drift force of the processes. During inference, the continuous time neural process is used with a discretization numerical ODE/SDE solver and outputs the results. One example is the denoising probabilistic diffusion models (DDPM) (Ho, Jain, and Abbeel 2020), which achieves great successes by serving as a generator for images (Saharia et al. 2022), videos (Ho et al. 2022; Liu et al. 2022), molecules (Wu et al. 2022) and 3D objects (Zheng et al. 2022; Zeng et al. 2022). Recently, people (Liu, Gong, and Liu 2022; Lipman et al. 2022; Albergo and Vanden-Eijnden 2022) form straight-line trajectories whereas diffusion paths result in curved paths and find that straight trajectories can push the transport close to the optimal transport paths and empirically translate to faster training, faster generation, and better performance. Intuitively, the straight transport curve can reduce the transport cost, and the neural model can optimally transport two distributions with minimum transport cost once learning perfectly.

Time-continuous Networks and Weight-shared Networks Time-continuous networks, *e.g.*, neural ODE (Chen et al. 2018), neural bridges (Wu et al. 2022; Wang et al. 2021b), DDPM, recently show their powers in image generation and other real-world application. These models repeatedly feed forward the input data dependent on time $t \in [0,1]$ and then feed back the final output. The training objective for a time-continuous neural network f_{flow} can be formulated as

$$\begin{aligned} &\min_{\theta} \int_0^1 \mathbb{E}\left[\|(f_{\text{flow}}(y_t, t) - (y_1 - y_0)\|^2\right] \mathrm{d}t, \\ &\text{where } \; y_t = t y_1 + (1-t) y_0 \quad t \in [0,1], \end{aligned} \tag{1}$$

where y_0 is the standard input for a neural network, y_1 denotes the output , *e.g.*, images, texts, $t \in [0,1]$ stands for time which goes from 0 to 1 during inference, and $f_{\text{flow}}(\cdot, t)$ denotes the neural model for the flow. Starting from $t = 0$, the output will smoothly change from the original source data $x = y_0$ to the target data y_1 when $t = 1$. During inference, we discretize time step t (e.g., t from 0 to 1 with step 0.1) and the time-continuous neural network can be regarded as a weight-shared neural network, whose number of layers equals to the discretization steps.

Train and Compress Weight-shared Networks Weight-shared neural networks are a widely-known topic and researchers have studied how to train or compress these models for a long period. In order to compress a standard multi-layer neural network, Weight-sharing BERT (Yang et al. 2021) and universal transformer (Dehghani et al. 2018) propose to first learn a weight-shared network. Weight-sharing networks reuse a neural layer v, and can be formulated as $y \longleftarrow y + v(y)$, which can be viewed as discretizing the ODE, $\mathrm{d}y_t = v(y_t)\mathrm{d}t$, with a time-independent velocity field Neural ODE models also share weights across layers and can be further accelerated with numerical ODE solvers, *e.g.*, Runge–Kutta -45 solver, and Euler solver. However, two issues remained to be large problems, ① weight-shared network is usually hard to train (*e.g.*, RNNs), and ② we do not have a principled method for compressing the multiple-layer weight-shared network into a one-block or few-block model.

Straight Flow is Easy to Compress When learning a neural flow for distribution transformation (*e.g.*, generative models), two kinds of information are required. The first is how to pair different data instances in the source and target domain. The second is how to minimize the transport cost. For supervised learning, we already know how to make the data pair, and therefore the key issue is to train a neural network that can easily be compressed into a few-step model. Different from other neural flows, straight flows are found to be good models for fast simulation. Researchers (Liu, Gong, and Liu 2022; Lipman et al. 2022) notice that once the objectives follow a linear interpolation which has low transport cost, the learned neural flow can get accurate results with a few-step simulation. Specifically, we say that a flow $\mathrm{d}y_t = f_{\text{flow}}(y_t, t)\mathrm{d}t$ is straight if we surely have $f_{\text{flow}}(y_t, t) = y_1 - y_0 = \text{const.}$ (“straight” in our paper refers to straight with a constant speed.) Here, we define the straightness of any smooth process by

$$S_{\text{Traj}}(f_{\text{flow}}) = \int_0^1 \left\{ \|(y_1 - y_0) - \dot{y}_t\|^2 \right\} \mathrm{d}t, \tag{2}$$

where $S_{\text{Traj}}(f_{\text{flow}}) = 0$ means exact straightness and $\dot{y}_t$ is the f_{flow} output. A flow with a small $S_{\text{Traj}}(f_{\text{flow}})$ has nearly straight paths and can be simulated using numerical solvers with a small number of discretization steps.

Compared to the other trajectories, which usually require hundreds of steps to simulate, the straight flow achieves comparable results on image generation with one-step simulation (Liu, Gong, and Liu 2022). Therefore, once we train a time-continuous model or weight-shared model, which follows the optimal transport trajectories, compressing the depth of the model is theoretically easier than other approaches. Compared to other compressing methods for weight-sharing method, this approach is theory-guided and thus a more principled method.

Method

Inspired by the fast simulation property of straight flows, in the following part, we propose a novel framework to reduce the model depth. Our algorithm contains three training phases, ❶ First, we distill a deep neural network into a time-continuous neural flow model. ❷ We apply ReFlow to reduce the transport cost of the time-continuous neural flow model multiple times. ❸ Finally, we further refine the

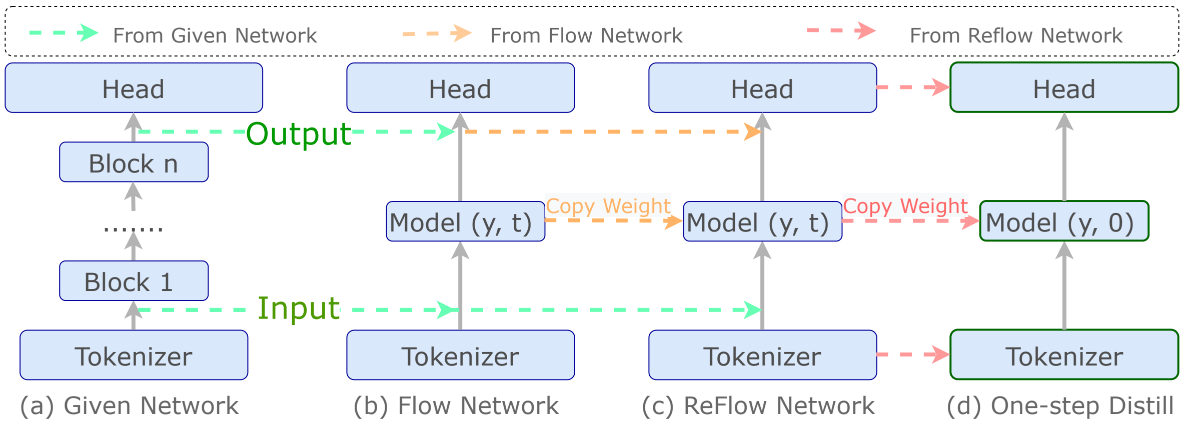

Figure 1: An illustration of our method. Given a trained network in (a) in which the output of tokenizer and the input of the head have the same resolution, we first train a time-continuous neural network in (b), and then learn from the flow network in (c). We name this step as ReFlow. The ReFlow operation can be applied for multiple times. Finally, we set $t = 0$ and use knowledge distillation to refine the model. At the final step, we jointly refine the head and tokenizer parameters together with the flow model parameters.

one-step flow with knowledge distillation. We start our discussions with notations together with definitions, and then introduce three stages of our method.

Problem Definition Many powerful deep learning models involves residual blocks of form

$$y_1 = f_{\text{given}}(y_0) = \phi_N \circ \cdots \phi_1(y_0),$$

where $\phi_k(y) = y + \epsilon\mu_k(y)$, ϵ is the step size. These models can be viewed as a discretization of the continuous-time ODE model $\mathrm{d}y_t = v(y,t)\mathrm{d}t$, where $v(x,t)$ is a drift force that depends on the continuous time $t \in [0,1]$. Our goal is to restructure the map $y_1 = f_{\text{given}}(y_0)$ from a pre-trained model into an continuous ODE model $\mathrm{d}y_t = v(y,t)\mathrm{d}t$ that (approximately) follow straight trajectories. As shown in (Liu, Gong, and Liu 2022), having straight trajectories allows us to solve the ODE with one step.

Notations Given a trained model f_{given} on the dataset $\mathcal{D} = \{x\}$ where $x \in \mathbb{R}^d$ denotes the data instance and N is the number of data, assume $f_{\text{given}}(x) \in \mathbb{R}^d$ has the same resolution as x, we train a new model f_{flow} in order to have

$$f_{\text{given}}(x) = \text{SOLVER}(f_{\text{flow}}, x), \quad \text{for } x \in \mathcal{D}, \tag{3}$$

where the SOLVER refers to inference the time-continuous flow by a numerical solver. SOLVER requires passing f_{flow} multiple times to get the final results. As shown in Figure 1, an example for the f_{given} model is the vision transformer without the classification head and the tokenizer. The flow model is defined as an ordinary differentiable model (ODE) on time $t \in [0,1]$,

$$\mathrm{d}y_t = f_{\text{flow}}(y_t,t)\mathrm{d}t, \tag{4}$$

which converts y_0 from the f_{given} input to y_1, which is the f_{given} output. During inference, we use a numerical ODE solver. For example, applying an Euler solver, we can inference the model going from y_0 to y_1 with,

$$y_{t+1/S} \longleftarrow y_t + f_{\text{flow}}(y_t,t)/S, \quad \text{where } y_0 = x, \tag{5}$$

where S denotes the step size from 0 to 1. For example, when we run the model 1000 times, $S = 1000$.

Algorithm 1: Compression with Straight Flows: Main Algorithm

```
Input: A trained model $f_{\text{given}}$, a dataset $\mathcal{D}$, number of reflow $K$, a neural flow model $f_{\text{flow}}$ whose input is $x$ and time $t$.
[Phase 1] Train Time-Continuous Flow Model: Train $f_{\text{flow}}$ with objective (6), with $y_1 = f_{\text{given}}(y_0)$, $y_0 = x \sim \mathcal{D}$.
[Phase 2] ReFlow:
    For $k$ from $1$ to $K$
        Generate $y_1^k$ with (5) using $f_{\text{flow}}^{k-1}$ and $y_0$, train $f_{\text{flow}}^k$ with objective (6) and $(y_0, y_1^k)$.
[Phase 3] Distillation: Set $t = 0$, distill the model with $\ell(f_{\text{distill}}(x), y^*)$ where $f_{\text{distill}}(y_0) = y_0 + f_{\text{flow}}^K(y_0, 0)$, $y_0 = x \sim \mathcal{D}$ and $y^*$ denotes the true label.
```

Train a Time-continuous Model To let the model follows the straight trajectories, given a pre-trained model $f_{\text{given}}(x)$, we apply the training objectives for flow proposed in (Lipman et al. 2022; Liu, Gong, and Liu 2022) to learn the neural flow model f_{flow},

$$\min_\theta \mathbb{E}_{x\in\mathcal{D},t\in[0,1]} \left\| f_{\text{flow}}^\theta(y_t,t) - \Big(f_{\text{given}}(x) - x\Big) \right\|^2, \tag{6}$$

where $x = y_0$, θ is the model parameters, $\mathcal{D}$ denotes the dataset and time $t \in [0,1]$, $y_t = ty_0 + (1-t)y_1$ during training and follows (5) during inference. Intuitively, the loss tries to learn the difference between the input and output pair. Given input x, once we mimic the value $f_{\text{given}}(x) - x$ well, we can predict $f_{\text{given}}(x)$ accurately.

Reflow Yields Straightening and Fast Simulation After training a neural flow model f_{flow}, we do not have a guarantee that our model has a small transport cost. To force this property, we follow (Liu, Gong, and Liu 2022) to apply additional ReFlow operator to the model. ReFlow makes the translation trajectory more straight. Such straight flows are therefore highly attractive as: a single Euler step update $y_1 = y_0 + f_{\text{flow}}(y_0, 0)$ calculates the exact y_1 from y_0. (Liu, Gong, and Liu 2022) shows in Theorem 1 that applying the ReFlow operator multiple times provably decreases $S_{\text{Traj}}(f_{\text{flow}})$ towards zero. We have a guarantee that applying ReFlow only reduces the transport cost, and we can come to a easy-to-distill model after multiple ReFlow.

Theorem 1 (from (Liu, Gong, and Liu 2022)) *Let f_{flow} be the k-th flow induced from (y_0, y_1). Then* $\min_{k\in\{0\cdots K\}} S_{\text{Traj}}(f_{\text{flow}}) \leq \frac{\mathbb{E}[\|y_1 - y_0\|^2]}{K}$.

Due to Theorem 1, the ReFlow operation can continuously decrease the transport cost. It indicates that, in practice, we can repeat ReFlow for many times and make the model more and more friendly for one-step simulation. Generating y_1^1 from the first flow, we train the model with

$$\min_{\theta} \mathbb{E}_{x\in\mathcal{D}, t\in[0,1]} \left\| f_{\text{flow}}^{\theta}(y_t, t) - \left(y_1^1 - x\right) \right\|^2. \tag{7}$$

When we recursively apply the procedure $F_{\text{flow}}^{k+1} = \text{ReFlow}(\mathcal{D}^k; F_{\text{flow}}^k)$ where $\mathcal{D}^k = \left\{\left(x, \text{SOLVER}^k(x)\right)\right\}$ and SOLVER^k applies ODE solver to f_{flow}^k, the paths of the k-rectified flow are increasingly straight, and hence easier to simulate numerically, as k increases. In practice, we can thus use the ReFlow to get one-step model.

Refine with Knowledge Distillation After obtaining the k-th rectified flow, we further improve the inference speed and accuracy by distilling the relation of (y_0, y_1^k) into a neural network f_{distill} to directly predict y_1^k from y_0 without simulating the flow, where $y_1^k = F_{\text{flow}}^k(x)$. Given that the flow is already nearly straight (and hence well approximated by the one-step update), the distillation can be done efficiently. In particular, if we take $f_{\text{distill}}(y_0) = y_0 + f_{\text{flow}}(y_0, 0)$, we come to a one-step model which can save computation costs. The *difference* between distillation and ReFlow should be highlighted: ReFlow yields a different coupling (y_0^{k+1}, y_1^{k+1}) with lower transport costs and more straight flow, and makes fast simulation possible. Distillation attempts to approximate the coupling (y_0^k, y_1^k) to make a small model performs well. In our setup, the distillation is applied only in the final stage for fine-tuning the model to get a better one-step inference result.

Summary We summarize the three stages of our algorithm in Algorithm 1. Given a trained model, we first train a time-continuous model. Intuitively, we distill a standard neural network into a time-continuous model. We distill the input and the output pair generated from the standard given network. After this phase, we then go to the ReFlow stage to reduce the transport cost. We can regard this step as fine-tuning the first step model. By applying the objective in (2) multiple times, we can consistently reduce the transport cost. As mentioned above, we can repeatedly apply ReFlow operation and we repeat the ReFlow step once or twice in practice. Finally, we distill the model by setting $t = 0$ and get the final model. This stage is the standard KD step.

Experiments

We benchmark and conduct our method on multiple cases. ① Note that the flow model requires the input and the output in the same resolutions, we first benchmark our results on vision transformers on CIFAR10 (Krizhevsky and Hinton 2010) and ImageNet (Deng et al. 2009) datasets. ② We then extend our method to multi-resolution model architectures, e.g. ResNet-50 (He et al. 2016) and SWIN-Base (Liu et al. 2021b). We learn a flow model for each resolution and combine 4 different flow model together to get the final one. ③ We transfer the ImageNet models to semantic segmentation. ④ In literature, *Block pruning* studies how to prune layers for all the data instances, while *dynamic inference (network)* focuses on how to select different layers for process difference input data instances. We demonstrate that our architecture can do dynamic inference by using output confidence score as a decision score and passing the model different times.

Image Classification

We conduct experiments on CIFAR-10 and ImageNet and observe that our method can boost the compressed model performance on standard model architectures.

DeiT on CIFAR-10 ❶ *Setting:* We compare our method with several different baselines on CIFAR-10. We first conduct our method on the recent proposed DeiT (Touvron et al. 2021), a vision transformer model. ❷ *Training Descriptions:* In our implementation, given a well trained DeiT, we replace the multiple transformer layers with one neural block, and keep the tokenizer and the classification head parameters the same. In the first training stage, we train a time-continuous neural block. In the second training stage, we apply the ReFlow algorithms to reduce the transport cost. In the final training stage, we set time step to one and use knowledge distillation to refine the one-step model. During this step, we jointly finetune our model weights along with the toeknizer and the head weights. In practice, we fine-tune the checkpoint from the previous stage. ❸ *Baselines:* We set up two different kinds of baselines. ① First, we compare with several different types of knowledge distillation (KD) algorithm mentioned in TinyBERT (Jiao et al. 2019) and MobileBERT (Sun et al. 2020). TinyBERT applies two different types of knowledge distillation, standard distillation on final-layer logits, and aligning each-layer attention/feature map. MobileBERT additionally first distill a one-layer student network and then go deeper. In summary, we set the standard (KD), feature map and attention alignment, growing student networks as three baselines. ② We compare to algorithms which learn to drop blocks. (Wang et al. 2019) proposes to prune blocks based on their similarity. (Chen and Zhao 2018) determines redundant parameters by investigating the features learned and pruning model parameters at a layer level. ❹ *Training Settings:* To have a fair comparison, we train or finetune different methods with the same number of epochs on CIFAR-10. For our method, the first two stage uses 300 epochs to train, respectively. For the final distillation refinement stage, we train the model with 400 epochs. For all the KD baselines, we train the model for 1,000 epochs. For the block drop baselines, we fine-tune the pruned model for 1,000 epochs. We use the AdamW (Loshchilov and Hutter 2018) optimizer with batch size 512 and an initial learning rate 5×10^{-4} with cosine learning rate decay (Loshchilov and Hutter 2019). We train all these methods on a 12-layer DeiT-Base model which is trained by 600 epochs and has 95.5% top-1 accuracy, and use a 4-layer transformer model for our and the distillation methods.

Method	FLOPs (%) ↓	Acc (%) ↑
Standard Training	34.2	94.5
Standard KD	34.2	95.2±0.01
+ Attention Transfer	34.2	95.1±0.02
+ Feature Map Transfer	34.2	95.2±0.01
+ Growing	34.2	95.2±0.01
DBP	38.5	95.1±0.01
LWP	40.2	95.0±0.01
Ours	34.2	**95.4±0.01**

Table 1: Performance of different methods on DeiT-Base trained on CIFAR-10. 'FLOPs Ratio' reports the FLOPs ratio of the compressed model and the DeiT-Base model. 'Top-1 Acccuracy' reports the top-1 accuracy on the test set. The results are averaged on 3 different trials. 'Standard Training' refers to train the 4-layer transformer model from scratch.

❺ *Compare to Baselines:* We evaluate our proposal and compare it with existing approaches. We notice that ① Compared to existing KD methods, our method yields better accuracy when all methods use the same computation cost. Compared to directly distill a shallow model, we first create an intermediate objective, a weight-shared deep model. Given the guarantee that the deep model is easy to compress to one-step model, we make the training objective easier and some to better results. ② We outperform the pruning methods. Pruning methods take benefits from jointly optimizing the architecture and the model parameters, however, larger search space may come to harder optimization problems. DBP and LWP thus use heusristic auxiliary loss to optimize. Our improvements suggest that our method creates a better auxiliary objective than these pruning methods.

❻ *Ablation Studies:* We conduct multiple ablation studies to understand different components of our method. ① We compare different trajectory objectives in Table 3. It demonstrates that our linear trajectory comes to better results than the others. As mentioned in the caption, the other trajectories are nonlinear and therefore cannot perform as good as ours when reducing the time step. ② Different trajectories get the same results when number of time steps is large ($t = 10$). When $t = 10$, the FLOPs is 3 times larger than than given 12-layer model. ③ After ReFlow, we come to better distillation results. As demonstrated in Table 3, compared to the first-stage trained flow model, ReFlow improves the one-step and two-step accuracy from 94.9% to 95.2%, from 95.4% to 95.5%, respectively. ④ As demonstrated in Table 3, when applying ReFlow twice, we come to slightly better one-step results, and slightly worse two-step results. It indicates that ReFlow has a trade-off between accuracy and efficiency. Although ReFlow can consistently reduce the transport cost and make it model easy to compress, it cannot guarantee that we can keep the training loss and accuracy the same. ⑤ We use different size of models as our base model, and display the results in 2. 4-layer ResNet achieves 95.4%±0.02% accuracy and it shows that our method works for different architectures.

Method	1-Step	2-Step	10-Step
Cosine	94.7±0.01	95.2±0.01	**95.5±0.01**
DDPM	94.9±0.01	95.3±0.01	**95.5±0.01**
Ours-Flow	94.9±0.02	95.4±0.01	**95.5±0.01**
Ours-Reflow	95.2±0.02	95.5±0.02	**95.5±0.01**
Ours-Reflow2	95.3±0.03	95.4±0.02	**95.5±0.01**
Ours	**95.4±0.01**	**95.5±0.01**	**95.5±0.01**

Table 2: Accuracy of our method with different trajectories, number of time steps, and training stages. $n-$step denotes the time-continuous model is discretized as a n-step model. 'Ours' uses $y_t = ty_0 + (1-t)y_1$ as target, while 'Cosine' and 'DDPM' apply $y_t = \sin(t)y_0 + \cos(t)y_1$ and $y_t = \sqrt{1-\alpha_t^2}y_0 + \alpha_t y_1$, respectively. α_t are pre-defined hyper-parameters. 'Ours-Flow', 'Ours-Reflow' and 'Ours' denotes the first, second and the final stage of our training. 'Ours-Flow' reports the results of the trained flow model. 'Ours-Reflow' displays the ReFlow model performance. 'Ours' is the final results after refining 'Ours-Reflow' with knowledge distillation. 'Ours-Reflow2' applies ReFlow twice.

Objective Type	FLOPs (%)	Top-1 Acc (%)
2−layer Transformer	10.4	**93.5±0.02**
4−layer ResNet	35.6	**95.4±0.02**
4−layer Transformer	34.2	**95.4±0.01**

Table 3: Accuracy of our method with different model size and different architectures. We notice that our method works for different architectures on CIFAR10.

ImageNet Experiments ❶ *Setting:* We further test our method on ImageNet (Deng et al. 2009) dataset. We start test from DeitBase trained on ImageNet. Then, in order to test the generalizability of our method, we apply our method to more architectures whose input and output images are in different resolution, *e.g.*, ResNet-50 and SWINBase. For these models, we distill one time-continuous model for one resolution, and therefore we come to four different time-continuous models, from resolution 56×56, 28×28, 14×14 to 7×7, for ResNet-50 and SWINBase.

❷ *Training Details:* Similar to CIFAR-10 settings, we train or finetune different methods with the same number of epochs for a fair comparison. For our method, each stage uses 200 epochs to train. For all the KD baselines, we train the model for 600 epochs. For the block drop baselines, we finetune the pruned model for 600 epochs. We use the AdamW optimizer with initial learning rate 5×10^{-4} and batch size 512 with cosine learning rate decay.

❷ *Baselines:* Based on the CIFAR-10 results, we select the best pruning and distillation baselines to serve as ImageNet baselines due to computation limitations. Similar to CIFAR-

	SWIN-Base		DeiT-Base		ResNet-50	
Method	FLOPs (G)	Top-1 Accuracy (%)	FLOPs (G)	Top-1 Accuracy (%)	FLOPs (G)	Top-1 Accuracy (%)
Standard	15.1	83.1	14.2	81.8	4.12	77.0
Reduced	5.10	80.3	6.03	81.8	1.86	76.2
KD	5.10	82.1±0.0	6.03	81.0±0.1	1.86	76.6±0.1
Growing	5.10	82.2±0.1	6.03	81.4±0.1	1.86	76.8±0.3
DBP	5.98	81.2±0.2	7.85	80.6±0.3	2.54	76.8±0.2
Ours	5.10	**82.6±0.2**	6.03	**81.6±0.1**	1.86	**77.0±0.1**

Table 4: Performance of our method and baselines on DeiT-Base, ResNet-50 and SWINBase. 'FLOPs' reports the FLOPs of the compressed model. 'Top-1 Acccuracy' reports the top-1 accuracy on the test set. The results are averaged on 3 different trials. 'Standard' refers to the original model architectures, 'Reduced' means using the same model architecture as our method, and 'KD' denotes that we use knowledge distillation with the final-layer logits.

10, we list the knowledge distillation and the pruning baselines. For KD, we compare to standard KD, the simplest one for implementation, and *Growing* (Sun et al. 2020). *Growing* (Sun et al. 2020) propose to create intermediate objectives during distillation to make the optimization easier, which shares the same motivation as our method. For pruning, we compare to DBP (Wang et al. 2019).

❸ *Computation Cost:* During training, at each iteration, we first pass through the teacher model, and then get the final-layer output for each resolution stage. Then, we train one-stage model for each resolution. At the first training stage, our training time cost is passing the model for each resolution once. In the ReFlow stage, we pass the model once at each training iteration. In the final distillation stage, we set the t for time-continuous models to 0, and distill the one-step model using the given pre-trained checkpoint as the teacher model. In this training stage, our training time cost is passing the model for each resolution once. In summary, compared to standard knowledge distillation, we do not introduce extra computation cost in our training.

❹ *Main Results:* We summarize the main experiment results in Table 4. ① Our method can reduce more than 50% of the computation cost, while have little or no drop on accuracy. For example, we drop 0.1% accuracy on DeiT and have no drop on ResNet-50, while reducing 57% and 55% computation cost respectively. ② Our methods have outperformed all the other baselines in all the three different architectures. Especially, for the SWIN-Base model, we achieve 82.6% accuracy with one-third FLOPs cost, compared to the original SWIN-Base model. We boost the performance by a large margin than the best baseline (82.6% *v.s.* 82.0%). ③ Our method not only works for architectures whose input and output have the same feature map resolutions, but can generalize to more architectures as well. We outperform other methods on ResNet-50 and SWINBase.

❺ *Ablation Studies:* Ablation studies are conducted to double verify our findings in the CIFAR-10 experiments. ① We compare different trajectories in Table 5, and we notice that, on DeiT-Base and SWIN-Base models, our objective is better than the others when time step equals to 1. When time step is 10, all the trajectories It indicates that our optimal cost aware trajectory is more suitable than the others for compressing and few-step simulation. ② We further list the FLOPs and accuracy for different time steps in Table 6. It demonstrates that the three-step model has similar FLOPs as the given SWIN-Base checkpoint (see Table 4), and models with fewer steps can reduce the computation cost. It displays that our trajectories makes it possible to reduce the computation cost while keep the accuracy. Other training trajectories cannot achieve such a difficult goal.

Model	1-Step	2-Step	10-Step
SWIN-Base			
Cosine	82.1±0.2	82.5±0.3	**83.1±0.1**
DDPM	82.3±0.2	82.5±0.1	**83.1±0.1**
Ours	**82.6±0.2**	**83.0±0.2**	**83.1±0.1**
DeiT-Base			
Cosine	81.3±0.1	81.5±0.1	**81.8±0.1**
DDPM	81.4±0.2	81.7±0.2	**81.8±0.1**
Ours	**81.6±0.1**	**81.8±0.2**	**81.8±0.1**

Table 5: Accuracy with different trajectory and different number of time steps on ImageNet. $n-$step denotes the time-continuous model is discretized to n step during inference. We use Euler solver to do inference. In these three different training objectives, ours is the best.

#step	1	2	3	5
Accuracy ↑	82.6	83.0	83.1	83.1
FLOPs (G) ↓	5.1	10.2	15.2	25.5

Table 6: Performance of SWIN-Base model on ImageNet validation dataset. We report the accuracy and the FLOPs with different time steps. As shown above, we can almost recover the model performance when time step equals to 2.

Semantic Segmentation We examine our method performance on the semantic segmentation task. We evaluate different model performance on the ADE20K (Zhou et al. 2017) benchmark, which contains more than 20K scene-centric images exhaustively annotated with 150-class pixel-level objects and pixel-level labels. To measure the accuracy and the efficiency and compare different methods, we report single-scale evaluation mIoU scores and FPS (Frame per second) on the test set.

❶ *Training Settings:* We closely follow the finetuning settings proposed in SWIN transformers (Liu et al. 2021b). Specifically, we use UperNet (Xiao et al. 2018) in *mmseg-*

mentation (Contributors 2020) as our test benchmark. During training, we use AdamW (Loshchilov and Hutter 2018) optimizer with a learning rate of 6×10^{-5} and a weight decay of 0.01. We use a cosine learning rate decay and a linear learning rate warmup of 1,500 iterations. We load the pretrained checkpoint on ImageNet and then finetune our models for 160K iterations on ADE20K training set. We adopt the default data augmentation scheme in *mmsegmentation* (Contributors 2020) and train with 512×512 crop size for ADE20K following the default setting in *mmsegmentation*. Additionally, following SWIN transformers (Liu et al. 2021b), we use a stochastic depth dropout of 0.3 for the first 80% of training iterations, and increase the dropout ratio to 0.5 for the last 20% of training iterations.
❶ *Main Results:* As reported in Table 7, ① we first notice that our trained backbone yields a better trade-off between accuracy and efficiency. Our one-step models come to a little drop (0.5% for SWINBase and 0.2% for DeiT) on mIoU, but improves the efficiency a lot. ② Our two-step model can match the mIoU performance while still get better performances on efficiency. For example, we achieve 48.1 % mIoU with 10.5 FPS with a two-step model for SWIN transformer base model. we achieve 44.2 % mIoU with 11.5 FPS with a two-step model for DeiT base model. Comapre to the standard results, we do not drop the mIoU while boosts the efficiency. ③ For the two different architectures, DeiT and SWIN, our method can both improve the baseline performance and outperform other methods. It further indicates that our method can work for different architectures and can generalize to architectures whose input and output are in different resolutions.

Model	Method	mIoU (%) ↑	FPS (Sec) ↓
SWINBase	Standard	48.1	8.7
	One-Step	47.6	12.1
	Two-Step	48.1	10.5
	Standard KD	47.0	12.1
	Growing	47.1	12.1
	DBP	47.1	12.2
DeiTBase	Standard	44.2	10.2
	One-Step	44.0	12.4
	Two-Step	44.2	11.5
	Standard KD	43.6	12.4
	Growing	43.7	12.4
	DBP	43.5	12.5

Table 7: Performance of different pre-trained checkpoints on ADE20K test set. For mIoU, the higher the better. For FPS, the higher the more efficient. 'One-Step' and 'Two-step' report our model performance when time step is 1 and 2.

Dynamic Inference By applying different number of time steps for different input data instances, our method can naturally serve as a dynamic inference model. Inference a deep neural network usually requires high computation costs. Therefore, a long line of works have been devoted to accelerating the inference process of deep neural networks by choosing different inference path for different input data. For example, dynamically filtering word tokens in NLP (Ye et al. 2021) or using dynamic resolutions in video understanding (Wang et al. 2021c). In literature, for the naive early-exit dynamic inference, researchers train additional reinforcement learning agents or directly use confidence score. ❶ *Our Strategy:* To have a dynamic network for free, our one-step model is firstly applied to the input data instance. Once the model confidence is larger than a threshold, the model exits. Otherwise, we further inference the two-step model and repeat the above step before we meet the computation constraints. In practice, we set the computation constraints as large as the uncompressed model FLOPs. ❷ *Compare to Baselines:* To have a fair comparison, we only compare to two kinds of baselines and both of them do early-exit for an input instance. We do not directly compare to other more complicated and domain-specific strategies, *e.g.*, dynamic channels, tokens, resolution (Wang et al. 2021a). We try two widely-used baselines, the first is to train a RL agent to select layers. Given the previous layer features as input, it make decisions on whether skipping the coming layer or not (Lin et al. 2017). The other one is using multiple classification head and make decisions based on the per-head prediction confidence (Zhang, Chen, and Zhong 2021) while each layer has a classification head. For our method and the confidence baseline, we exit the model once we have a confidence score over 0.6. For the RL agent baseline, we use hidden states as input and feedback action (exit or not) at each layer. We show more results with difference confidence threshold and find out that there is no visible difference. ❸ *Main Results:* As displayed in Table 8, we achieve 81.8% top-1 accuracy while reduce half of the model FLOPs. The results is comparable to the confidence score based method. The RL agent can be lightly better than ours on FLOPs, 6.5G *v.s.* 7.0G, but it is more complicated to train and use.

Method	Est FLOPs (G)	Accuracy (%)
RL Agent	**6.5**	81.8
Confidence	6.8	81.8
Ours	7.0	81.8

Table 8: Performance of dynamic DeiT-Base models. 'RL Agent' and 'confidence' refer to making the decision on exiting based on policy gradient trained agent or per-head confidence score.

Discussion and Conclusion

We propose a new approach to train shallow networks, starting from optimizing a time-continuous model. By reducing the transport cost by ReFlow operations, we make transport trajectories more straight and thus easier to build an accurate one-step neural flow model. After verifying the performance with experiments, we find out that efficient flow models can not only serve as a compression algorithm but can also be viewed as an early-exit dynamic network.

Our framework has several possible feature directions. First, our extension for the multiple resolutions requires one model for one resolution, which cannot detect the importance of different resolution. Secondly, we manually design the architecture for the flow model, and it will be a promising direction to search the architecture of the model.

References

Albergo, M. S.; and Vanden-Eijnden, E. 2022. Building Normalizing Flows with Stochastic Interpolants. *arXiv preprint arXiv:2209.15571*.

Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901.

Chen, R. T.; Rubanova, Y.; Bettencourt, J.; and Duvenaud, D. K. 2018. Neural ordinary differential equations. *Advances in neural information processing systems*, 31.

Chen, S.; and Zhao, Q. 2018. Shallowing deep networks: Layer-wise pruning based on feature representations. *IEEE transactions on pattern analysis and machine intelligence*, 41(12): 3048–3056.

Contributors, M. 2020. MMSegmentation: OpenMMLab Semantic Segmentation Toolbox and Benchmark. https://github.com/open-mmlab/mmsegmentation.

Dehghani, M.; Gouws, S.; Vinyals, O.; Uszkoreit, J.; and Kaiser, Ł. 2018. Universal transformers. *arXiv preprint arXiv:1807.03819*.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, 248–255. Ieee.

Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Gong, C.; Wang, D.; Li, M.; Chen, X.; Yan, Z.; Tian, Y.; Chandra, V.; et al. 2021. NASViT: Neural Architecture Search for Efficient Vision Transformers with Gradient Conflict aware Supernet Training. In *International Conference on Learning Representations*.

Graham, B.; El-Nouby, A.; Touvron, H.; Stock, P.; Joulin, A.; Jégou, H.; and Douze, M. 2021. Levit: a vision transformer in convnet's clothing for faster inference. In *Proceedings of the IEEE/CVF international conference on computer vision*, 12259–12269.

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*, 770–778.

Ho, J.; Jain, A.; and Abbeel, P. 2020. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33: 6840–6851.

Ho, J.; Salimans, T.; Gritsenko, A.; Chan, W.; Norouzi, M.; and Fleet, D. J. 2022. Video diffusion models. *arXiv preprint arXiv:2204.03458*.

Jiao, X.; Yin, Y.; Shang, L.; Jiang, X.; Chen, X.; Li, L.; Wang, F.; and Liu, Q. 2019. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*.

Krizhevsky, A.; and Hinton, G. 2010. Convolutional deep belief networks on cifar-10. *Unpublished manuscript*, 40(7): 1–9.

Lin, J.; Rao, Y.; Lu, J.; and Zhou, J. 2017. Runtime neural pruning. *Advances in neural information processing systems*, 30.

Lipman, Y.; Chen, R. T.; Ben-Hamu, H.; Nickel, M.; and Le, M. 2022. Flow Matching for Generative Modeling. *arXiv preprint arXiv:2210.02747*.

Liu, X.; Gong, C.; and Liu, Q. 2022. Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow. *arXiv preprint arXiv:2209.03003*.

Liu, X.; Gong, C.; Wu, L.; Zhang, S.; Su, H.; and Liu, Q. 2021a. Fusedream: Training-free text-to-image generation with improved clip+ gan space optimization. *arXiv preprint arXiv:2112.01573*.

Liu, Z.; Lin, Y.; Cao, Y.; Hu, H.; Wei, Y.; Zhang, Z.; Lin, S.; and Guo, B. 2021b. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 10012–10022.

Liu, Z.; Ning, J.; Cao, Y.; Wei, Y.; Zhang, Z.; Lin, S.; and Hu, H. 2022. Video swin transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 3202–3211.

Loshchilov, I.; and Hutter, F. 2018. Fixing weight decay regularization in adam.

Loshchilov, I.; and Hutter, F. 2019. Sgdr: Stochastic gradient descent with warm restarts. arXiv 2016. *arXiv preprint arXiv:1608.03983*.

Mandal, M.; Shah, M.; Meena, P.; and Vipparthi, S. K. 2019. SSSDET: Simple short and shallow network for resource efficient vehicle detection in aerial scenes. In *2019 IEEE international conference on image processing (ICIP)*, 3098–3102. IEEE.

Papamakarios, G.; Nalisnick, E. T.; Rezende, D. J.; Mohamed, S.; and Lakshminarayanan, B. 2021. Normalizing Flows for Probabilistic Modeling and Inference. *J. Mach. Learn. Res.*, 22(57): 1–64.

Rombach, R.; Blattmann, A.; Lorenz, D.; Esser, P.; and Ommer, B. 2021. High-Resolution Image Synthesis with Latent Diffusion Models. arXiv:2112.10752.

Saharia, C.; Chan, W.; Saxena, S.; Li, L.; Whang, J.; Denton, E.; Ghasemipour, S. K. S.; Ayan, B. K.; Mahdavi, S. S.; Lopes, R. G.; et al. 2022. Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding. *arXiv preprint arXiv:2205.11487*.

Song, J.; Meng, C.; and Ermon, S. 2020. Denoising Diffusion Implicit Models. In *International Conference on Learning Representations*.

Song, Y.; Sohl-Dickstein, J.; Kingma, D. P.; Kumar, A.; Ermon, S.; and Poole, B. 2020. Score-Based Generative Modeling through Stochastic Differential Equations. In *International Conference on Learning Representations*.

Sun, Z.; Yu, H.; Song, X.; Liu, R.; Yang, Y.; and Zhou, D. 2020. Mobilebert: a compact task-agnostic bert for resource-limited devices. *arXiv preprint arXiv:2004.02984*.

Touvron, H.; Cord, M.; Douze, M.; Massa, F.; Sablayrolles, A.; and Jégou, H. 2021. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, 10347–10357. PMLR.

Tzen, B.; and Raginsky, M. 2019. Theoretical guarantees for sampling and inference in generative models with latent diffusions. In *Conference on Learning Theory*, 3084–3114. PMLR.

Vargas, F.; Thodoroff, P.; Lamacraft, A.; and Lawrence, N. 2021. Solving Schrödinger bridges via maximum likelihood. *Entropy*, 23(9): 1134.

Wang, D.; Li, M.; Gong, C.; and Chandra, V. 2021a. Attentivenas: Improving neural architecture search via attentive sampling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 6418–6427.

Wang, G.; Jiao, Y.; Xu, Q.; Wang, Y.; and Yang, C. 2021b. Deep generative learning via schrödinger bridge. In *International Conference on Machine Learning*, 10794–10804. PMLR.

Wang, W.; Zhao, S.; Chen, M.; Hu, J.; Cai, D.; and Liu, H. 2019. DBP: Discrimination based block-level pruning for deep model acceleration. *arXiv preprint arXiv:1912.10178*.

Wang, Y.; Chen, Z.; Jiang, H.; Song, S.; Han, Y.; and Huang, G. 2021c. Adaptive focus for efficient video recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 16249–16258.

Wu, L.; Gong, C.; Liu, X.; Ye, M.; and Liu, Q. 2022. Diffusion-based molecule generation with informative prior bridges. *arXiv preprint arXiv:2209.00865*.

Xiao, T.; Liu, Y.; Zhou, B.; Jiang, Y.; and Sun, J. 2018. Unified perceptual parsing for scene understanding. In *Proceedings of the European conference on computer vision (ECCV)*, 418–434.

Xue, L.; Constant, N.; Roberts, A.; Kale, M.; Al-Rfou, R.; Siddhant, A.; Barua, A.; and Raffel, C. 2020. mT5: A massively multilingual pre-trained text-to-text transformer. *arXiv preprint arXiv:2010.11934*.

Yang, S.; Hou, L.; Song, X.; Liu, Q.; and Zhou, D. 2021. Speeding up deep model training by sharing weights and then unsharing. *arXiv preprint arXiv:2110.03848*.

Ye, D.; Lin, Y.; Huang, Y.; and Sun, M. 2021. Tr-bert: Dynamic token reduction for accelerating bert inference. *arXiv preprint arXiv:2105.11618*.

Zeng, X.; Vahdat, A.; Williams, F.; Gojcic, Z.; Litany, O.; Fidler, S.; and Kreis, K. 2022. LION: Latent Point Diffusion Models for 3D Shape Generation. *arXiv preprint arXiv:2210.06978*.

Zhang, Y.; Chen, Z.; and Zhong, Z. 2021. Collaboration of experts: Achieving 80% top-1 accuracy on imagenet with 100m flops. *arXiv preprint arXiv:2107.03815*.

Zheng, Y.; Wu, L.; Liu, X.; Chen, Z.; Liu, Q.; and Huang, Q. 2022. Neural Volumetric Mesh Generator. *arXiv preprint arXiv:2210.03158*.

Zhou, B.; Zhao, H.; Puig, X.; Fidler, S.; Barriuso, A.; and Torralba, A. 2017. Scene parsing through ade20k dataset. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 633–641.