TrAct: Making First-layer Pre-Activations Trainable

Felix Petersen
Stanford University
mail@felix-petersen.de

Christian Borgelt
University of Salzburg
christian@borgelt.net

Stefano Ermon
Stanford University
ermon@cs.stanford.edu

Abstract

We consider the training of the first layer of vision models and notice the clear relationship between pixel values and gradient update magnitudes: the gradients arriving at the weights of a first layer are by definition directly proportional to (normalized) input pixel values. Thus, an image with low contrast has a smaller impact on learning than an image with higher contrast, and a very bright or very dark image has a stronger impact on the weights than an image with moderate brightness. In this work, we propose performing gradient descent on the embeddings produced by the first layer of the model. However, switching to discrete inputs with an embedding layer is not a reasonable option for vision models. Thus, we propose the conceptual procedure of (i) a gradient descent step on first layer activations to construct an activation proposal, and (ii) finding the optimal weights of the first layer, i.e., those weights which minimize the squared distance to the activation proposal. We provide a closed form solution of the procedure and adjust it for robust stochastic training while computing everything efficiently. Empirically, we find that TrAct (Training Activations) speeds up training by factors between $1.25\times$ and $4\times$ while requiring only a small computational overhead. We demonstrate the utility of TrAct with different optimizers for a range of different vision models including convolutional and transformer architectures.

1 Introduction

We consider the learning of first-layer embeddings / pre-activations in vision models, and in particular learning the weights with which the input images are transformed in order to obtain these embeddings. In gradient descent, the updates to first-layer weights are directly proportional to the (normalized) pixel values of the input images. As a consequence (assuming that input images are standardized), high contrast, very dark, or very bright images have a greater impact on the trained first-layer weights, while low contrast images with medium brightness have only smaller impact on training.

While, in the past, mainly transformations of the input images, especially various forms of normalization have been considered, either as a preprocessing step or as part of the neural network architecture, our approach targets the training process directly without modifying the model architecture or any preprocessing. The goal of our approach is to achieve a training behavior that is equivalent to training the pre-activations or embedding values themselves. For example, in language models [1], the first layer is an "Embedding" layer that maps a token id to an embedding vector (via a lookup). When training language models, this embedding vector is trained directly, i.e., the update to the embedding directly corresponds to the gradient of the pre-activation of the first layer. As discussed above, this is not the case in vision models as, here, the updates to the first-layer weight matrix correspond to the outer product between the input pixel values and the gradient of the pre-activation of the first layer. Bridging this gap between the "Embedding" layer in language models, and "Conv2D" / "Linear" / "Dense" layers in vision models, we propose a novel technique for training the pre-activations of the latter, effectively mimicking training behavior of the "Embedding" layer in language models. As vision models rely on pixel values rather than tokens, and any discretization of image patches,

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

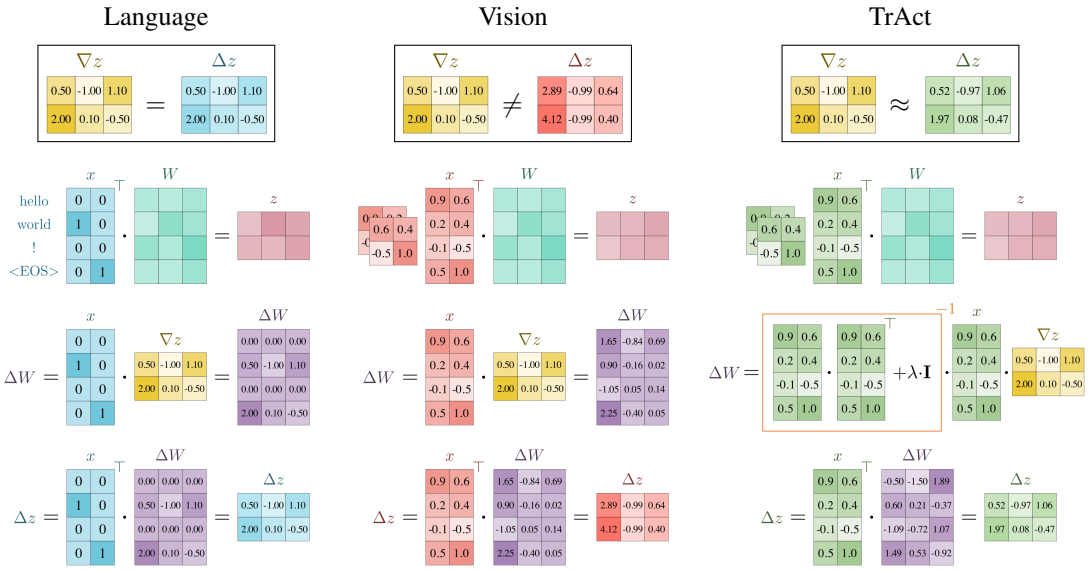

Figure 1: TrAct learns the first layer of a vision model but with the training dynamics of an embedding layer. We illustrate this in an example with two 4-dimensional inputs x, a weight matrix W of size 4×3, and resulting pre-activations z of size 2×3. For language models (left), the input x is two tokens from a dictionary of size 4. For vision models (center + right), the input x is two patches of the image, each totaling 4 pixels. During backpropagation, we obtain the gradient wrt. our pre-activations ∇z, from which the gradient and update to the weights W is computed (ΔW). The resulting update to the pre-activations Δz equals $x^\top \cdot \Delta W$. For language models (left), $\Delta z = \nabla z$, i.e., the training dynamics of the embeddings layer corresponds to updating the embeddings directly wrt. the gradient. Specifically, the update in a language model, for a token identifier i, is $W_i \leftarrow W_i - \eta \cdot \nabla_z \mathcal{L}(z)$ where $z = W_i$ is the activation of the first layer and at the same time the ith row of the embedding (weight) matrix W. Equivalently, we can write $z \leftarrow z - \eta \cdot \nabla_z \mathcal{L}(z)$. However, in vision models (center), the update Δz strongly deviates from the respective gradients ∇z. TrAct corrects for this by adjusting ΔW via a corrective term $(x \cdot x^\top + \lambda \cdot I)^{-1}$ (orange box), such that the update to z closely approximates ∇z.

e.g., via clustering is not a reasonable option, we approach the problem via a modification of the gradient (and therefore a modification of the training behavior) without modifying the original model architectures. We illustratively compare the updates in language and vision models and demonstrate the modification that TrAct introduces in Figure 1.

The proposed method is general and applicable to a variety of vision model architecture types, from convolutional to vision transformer models. In a wide range of experiments, we demonstrate the utility of the proposed approach, effectively speeding up training by factors ranging from $1.25\times$ to $4\times$, or, within a given training budget, improving model performance consistently. The approach requires only one hyperparameter λ, which is easy to select, and our default value works consistently well across all 50 considered model architecture + data set + optimizer settings.

The remainder of this paper is organized as follows: in Section 2, we introduce related work, in Section 3, we introduce and derive TrAct from a theoretical perspective, and in Section 3.1 we discuss implementation considerations of TrAct. In Section 4, we empirically evaluate our method in a variety of experiments, spanning a range of models, data sets, and training strategies, including an analysis of the mild behavior of the hyperparameter, an ablation study, and a runtime analysis. We conclude the paper with a discussion in Section 5. *The code is publicly available at github.com/Felix-Petersen/tract.*

2 Related Work

It is not surprising that the performance of image classification and object recognition models depends heavily on the quality of the input images, especially on their brightness range and contrast. For example, image augmentation techniques generate modified versions of the original images as additional training examples. Some of these techniques work by geometric transformations (rotation, mirroring, cropping), others by adding noise, changing contrast or modifying the image in

the color space [2]. In the area of vision transformers [3], [4] so-called 3-augmentation (Gaussian blur, reduction to grayscale, and solarization) has been shown to be essential to performance [5]. Augmentation approaches are similar to image enhancement as a preprocessing step, because they generate possibly enhanced versions of the images as additional training examples, even though they leave the original images unchanged, which are also still used as training examples.

Another direction related to the problem we deal with in this paper are various normalizations and standardizations, starting with the most common one of standardizing the data to mean 0 and standard deviation 1 (over the training set), and continuing through batch normalization [6], weight normalization [7], layer normalization [8], which are usually applied not just for the first layer, but throughout the network, and in particular patch-wise normalization of the input images [9], which we will draw on for comparisons. We note that, e.g., Dual PatchNorm [9], in contrast to our approach, modifies the actual model architecture, but not the gradient backpropagation procedure.

However, none of these approaches directly addresses the actual concern that weight changes in the first layer are proportional to the inputs, but instead only modify the inputs and architectures to make training easier or faster. In contrast to these approaches, we address the training problem itself and propose a different way of optimizing first-layer weights for unchanged inputs. Of course, this does not mean that input enhancement techniques are superfluous with our method, but only that additional performance gains can be obtained by including TrAct during training.

In the context of deviating from standard gradient descent-based optimization [10], there are different lines of work in the space of second-order optimization [11], e.g., K-FAC [12], ViViT [13], ISAAC [14], Backpack [15], and Newton Losses [16], which have inspired our methodology for modifying the gradient computation. In particular, the proposed approach integrates second-order ideas for solving a (later introduced) sub–optimization-problem in closed-form [17], and has similarities to a special case of ISAAC [14].

3 Method

First, let us consider a regular gradient descent of a vision model. Let $z = f(x; W)$ be the first layer embeddings excluding an activation function and W be the weights of this first layer, i.e., for a fully-connected layer $f(x; W) = W \cdot x$. Here, we have $x \in \mathbb{R}^{n \times b}$, $z \in \mathbb{R}^{m \times b}$, and $W \in \mathbb{R}^{m \times n}$ for a batch size of b. We remark that our input x may be unfolded, supporting convolutional and vision transformer networks. Further, let $\hat{y} = g(\hat{z}; \theta_{\setminus W}) = g(f(x; W); \theta_{\setminus W})$ be the prediction of the entire model. Moreover, let $\mathcal{L}(\hat{y}, y)$ be the loss function for a label y and wlog. let us assume it is an averaging loss (i.e., reduction over batch dimension via mean). During backpropagation, the gradient of the loss wrt. z, i.e., $\nabla_z \mathcal{L}(g(z; \theta_{\setminus W}), y)$ or $\nabla_z \mathcal{L}(z)$ for short, will be computed. Conventionally, the gradient wrt. W, i.e., $\nabla_W \mathcal{L}(g(f(x; W); \theta_{\setminus W}), y)$ or $\nabla_W \mathcal{L}(W)$ for short, is computed during backpropagation as

$$\nabla_W \mathcal{L}(W) = \nabla_z \mathcal{L}(z) \cdot x^\top , \tag{1}$$

leading to the gradient descent update step of

$$W \leftarrow W - \eta \cdot \nabla_W \mathcal{L}(W) . \tag{2}$$

Equation 1 clearly shows the direct proportionality between the gradient wrt. the first layer weights and the input (magnitudes), showing that larger input magnitudes produce proportionally larger changes in first layer network weights. We remark that a corresponding relationships also holds in later layers of the neural network, but emphasize that, in later layers, the relationship shows a proportionality to activation magnitude, which is desirable.

To resolve this dependency on the inputs and make training more efficient, we propose to conceptually optimize in the space of first layer embeddings z. In particular, we could perform a gradient descent step on z, i.e.,

$$z^\star \leftarrow z - \eta \cdot b \cdot \nabla_z \mathcal{L}(z) . \tag{3}$$

Here, b is a multiplier because $\mathcal{L}(z)$ is (per convention) the empirical expectation over the batch dim.

However, now, $z^\star$ depends on the inputs and is not part of the actual model parameters. We can resolve this problem by determining how to update W such that $f(x; W)$ is as close to $z^\star$ as possible. Conceptually, we compute the optimal update $\Delta W^\star$ by solving the optimization problem

$$\arg\min_{\Delta W} \|z^\star - (W + \Delta W) \cdot x\|_2^2 \qquad \text{subject to} \quad \|\Delta W\|_2 \leq \epsilon \tag{4}$$

where we (1) want to minimize the distance between $z^\star$ the embeddings implied by the change of W by ΔW, and (2) want to keep the change ΔW small.

We enforce that weight matrix changes ΔW are small ($\|\Delta W\|_2 \leq \epsilon$) by taking the Lagrangian of the problem, i.e.,

$$\arg\min_{\Delta W} \; \|z^\star - (W + \Delta W) \cdot x\|_2^2 \; + \; \lambda b \cdot \|\Delta W\|_2^2 \tag{5}$$

with a heuristically selected Lagrangian multiplier $\lambda \cdot b$ (parameterized with b because the first part is also proportional to b). We simplify Equation 5 to

$$\arg\min_{\Delta W} \; \| - \eta b \cdot \nabla_z \mathcal{L}(z) - \Delta W \cdot x\|_2^2 \; + \; \lambda b \cdot \|\Delta W\|_2^2 \tag{6}$$

and ease the presentation by considering it from a row-wise perspective, i.e., for $\Delta W_i \in \mathbb{R}^{1 \times n}$:

$$\arg\min_{\Delta W_i} \; \| - \eta b \cdot \nabla_{z_i} \mathcal{L}(z) - \Delta W_i \cdot x\|_2^2 \; + \; \lambda b \cdot \|\Delta W_i\|_2^2 \, . \tag{7}$$

The problem is separable into a row-wise perspective because the norm ($\| \cdot \|_2^2$) is the squared Frobenius norm and the rows have independent solutions.

In the following, we provide a closed-form solution for optimization problem (7), which is related to [17], [18].

Lemma 1. *The solution $\Delta W_i^\star$ of Equation 7 is*

$$\Delta W_i^\star = - \eta \cdot \nabla_{z_i} \mathcal{L}(z) \cdot x^\top \cdot \left(\frac{x x^\top}{b} + \lambda \cdot I_n \right)^{-1} . \tag{8}$$

Proof deferred to Supplementary Material A.

Extending the solution to ΔW, we have

$$\Delta W^\star = - \eta \cdot \nabla_z \mathcal{L}(z) \cdot x^\top \cdot \left(\frac{x x^\top}{b} + \lambda \cdot I_n \right)^{-1} \tag{9}$$

and can accordingly use it for an update step for W, i.e.,

$$W \leftarrow W + \Delta W^\star \qquad \text{or} \qquad W \leftarrow W - \eta \cdot \nabla_z \mathcal{L}(z) \cdot x^\top \cdot \left(\frac{x x^\top}{b} + \lambda \cdot I_n \right)^{-1} . \tag{10}$$

The update in Equation 10 directly inserts the solution of the problem formulated in Equation 4. This computation is efficient as it only requires inversion of an $n \times n$ matrix, where n in the case of convolutions correspond to 3 (RGB) times the squared first layer's kernel size, and for vision transformers corresponds to the number of pixels per patch. The values of n typically range from $n = 27$ (CIFAR ResNet) to $n = 768$ (ImageNet large-scale vision transformer).

Lemma 2. *Using TrAct does not change the set of possible convergence points compared to vanilla (full batch) gradient descent. Herein, we use the standard definition of convergence points as those points where no update is performed because the gradient is zero.*

Proof sketch: First, we remark that only the training of the first layer is affected by TrAct. To show the statement, we show that (i) a zero gradient for GD implies that TrAct also performs no update and that (ii) TrAct performing no update implies zero gradients for GD. *Proof deferred to SM A.*

The statement formalizes that TrAct does not change the set of attainable models, but instead only affects the behavior of the optimization itself.

For an illustration of how TrAct affects updates to W and z, with a comparison to language models and conventional vision models, see Figure 1.

3.1 Implementation Considerations

To implement the proposed update in Equation 10 in modern automatic differentiation frameworks [19], [20], we can make use of a custom backward or backpropagation for the first layer.

Standard gradient for the weights:

$$\nabla_W \leftarrow \nabla_z \mathcal{L}(z) \cdot x^\top \tag{11}$$

implemented via a backward function:

```
def backward(grad_z, x, W):
    grad_W = grad_z.T @ x
    return grad_W
```

For TrAct, we perform an in-place replacement by:

$$\nabla_W \leftarrow \nabla_z \mathcal{L}(z) \cdot x^\top \cdot \left(\frac{xx^\top}{b} + \lambda \cdot I_n \right)^{-1} \tag{12}$$

i.e., we replace the backward of the first layer by:

```
def backward(grad_z, x, W, l=0.1):
    b, n = x.shape
    grad_W = grad_z.T @ x @ inverse(
                x.T @ x / b + l * eye(n))
    return grad_W
```

Figure 2: Implementation of TrAct, where `l` corresponds to the hyperparameter λ.

Details are shown in Figure 2. This applies the TrAct update from Equation 10 when using the SGD optimizer. Moreover, extensions of the update corresponding to optimizers like ADAM [21] (including, e.g., momentum, learning rate scheduler, regularizations, etc.) can be attained by using a respective optimizer and pretending (towards the optimizer) that the TrAct update corresponds to the gradient. As it only requires a modification of the gradient computation of the first layer, the proposed method allows for easy adoption in existing code. All other layers / weights ($\theta_{\setminus W}$) are trained conventionally without modification. Convolutions can be easily expressed as a matrix multiplication via an unfolding of the input; accordingly, we unfold the inputs respectively in the case of convolution.

Moreover, we would like to show a second method of applying TrAct that is exactly equivalent. Typically, we have some batch of data `x` and a first embedding layer `embed`, as well as a remaining network `net`, a `loss`, and targets `gt`. In the following, we show how the forward and backward is usually written (left) and how it can be modified to incorporate TrAct (right):

```
z = embed(x)  # first layer pre-act
y = net(z)    # remainder of the net
loss(y, gt).backward()    # backprop
```

```
z = embed(x @ inverse(x.T @ x/b+l*eye(n)))
z.data = embed(x)  # overwrites the values
# in z but leaves the gradient as before
y = net(z)
loss(y, gt).backward()
```

This modifies the input of `embed` for the gradient computation, but replaces the actual values propagated through the remaining network `z.data` by the original values, therefore not affecting downstream layers. which modifies the input of `embed` for the gradient computation, but replaces the actual values propagated through the remaining network `z.data` by the original values, therefore not affecting downstream layers. This illustrates interesting relationships: TrAct is minimally invasive, can be removed or included at any time without breaking the network, and does not have learnable parameters. TrAct can be seen as in some sense related to normalizing / whitening / inverting the input for the purpose of gradient computation, but then switching the embeddings back to the original first layer embeddings / activations for propagation through the remainder of the network.

We provide an easy-to-use wrapper module that can be applied to the first layer, and automatically provides the TrAct gradient computation replacement procedure. For example, for PyTorch [19], the `TrAct` module can be applied to `nn.Linear` and `nn.Conv2d` layers by wrapping them as

```
TrAct(nn.Linear(...))        TrAct(nn.Conv2d(...))
```

and for existing implementations, we can apply TrAct, e.g., for vision transformers via:

```
net.patch_embed.proj = TrAct(net.patch_embed.proj)
```

4 Experimental Evaluation

4.1 CIFAR-10

Setup For the evaluation on the CIFAR-10 data set [22], we consider the ResNet-18 [23] as well as a small ViT model. We consider training from scratch as the method is particularly designed for this case. We perform training for 100, 200, 400, and 800 epochs. For the ResNet models, we use the Adam and SGD with momentum (0.9) optimizers, both with cosine learning rate schedules; learning rates, due to their significance, will be discussed alongside respective experiments. Further, we use the standard softmax cross-entropy loss. For the ViT, we use Adam with a cosine learning

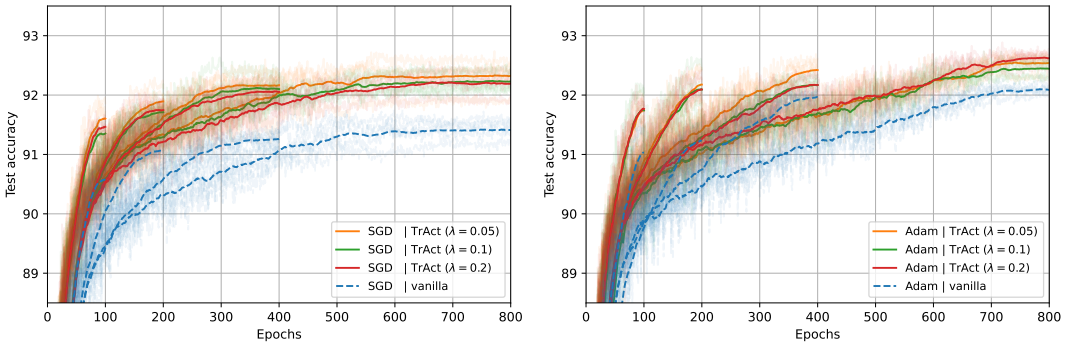

Figure 3: Training a **ResNet-18** on **CIFAR-10**. We train for $\{100, 200, 400, 800\}$ epochs using a cosine learning rate schedule and with SGD (left) and Adam (right). Learning rates have been selected as optimal for each baseline. Averaged over 5 seeds. TrAct (solid lines) consistently outperforms the baselines (dashed)—in many cases already with a quarter of the number of the epochs of the baseline.

rate scheduler as well as a softmax cross-entropy loss with label smoothing (0.1). The selected ViT[1] is particularly designed for effective training on CIFAR scales and has 7 layers, 12 heads, and hidden sizes of 384. Each model is trained with a batch size of 128 on an Nvidia RTX 4090 GPU with PyTorch [19].

As mentioned above, the learning rate is a significant factor in the evaluation. Therefore, throughout this paper, to remove any bias towards the proposed method (and even give an advantage to the baseline), we utilize the optimal learning rate of the baseline also for the proposed method. For the Adam optimizer, we consider a learning rate grid of $\{10^{-2}, 10^{-2.5}, 10^{-3}, 10^{-3.5}\}$; for SGD with momentum, a learning rate grid of $\{0.1, 0.09, 0.08, 0.07\}$. The optimal learning rate is determined for each number of epochs using regular training; in particular, for Adam, we have $\{100{\to}10^{-2}$, $200{\to}10^{-2}, 400{\to}10^{-3}, 800{\to}10^{-3}\}$, and, for SGD with momentum, we find that a learning rate of 0.08 is optimal in each case. For the ViT, we considered a learning rate grid of $\{10^{-3}, 10^{-3.1}$, $10^{-3.2}, 10^{-3.3}, 10^{-3.4}, 10^{-3.5}, 10^{-3.6}, 10^{-3.7}, 10^{-3.8}, 10^{-3.9}, 10^{-4}\}$. Here, the optimal learning rates (based on the baseline) are $\{100{\to}10^{-3}, 200{\to}10^{-3.2}, 400{\to}10^{-3.5}, 800{\to}10^{-3.5}\}$.

Results In Figure 3 we show the results for ResNet-18 trained on CIFAR-10. We can observe that TrAct improves the test accuracy in every setting, in particular, for both optimizers, for all four numbers of epochs, and for all three choices of the hyperparameter $\lambda \in \{0.05, 0.1, 0.2\}$. Moreover, we can observe that, for SGD, the accuracy after 100 epochs is already better than for the baseline after 800 epochs. For Adam, we can see that TrAct after 100 epochs performs similar to the baseline after 400 epochs, and TrAct after 200 epochs performs similar to the baseline after 800 epochs. Comparing the different choices of λ, $\lambda = 0.05$ performs best in most cases.

The results for the ViT model are displayed in Figure 4. Again, we can observe that TrAct consistently outperforms the baselines for all λ. Further, we can observe that TrAct with 200 epochs performs comparable to the baseline with 400 epochs. We emphasize that, again, the optimal learning rate has been selected based on the baseline. Overall, here, $\lambda = 0.1$ performed best.

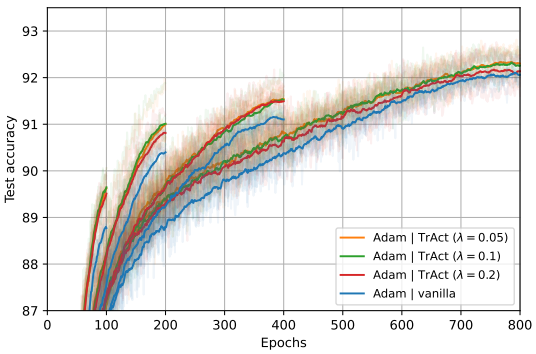

Figure 4: Training a **ViT** on **CIFAR-10**. We train for $\{100, 200, 400, 800\}$ epochs using a cosine learning rate schedule and with Adam. Learning rates have been selected as optimal for each baseline. Avg. over 5 seeds.

4.2 CIFAR-100

Setup For CIFAR-100, we consider two experimental settings. First, we consider the training of 36 different convolutional model architectures based on a strong and popular repository[2] for CIFAR-100. We use the same hyperparame-

[1] Based on github.com/omihub777/ViT-CIFAR.

[2] Based on github.com/weiaicunzai/pytorch-cifar100.

ters as the reference, i.e., SGD with momentum (0.9), weight decay (0.0005), and learning rate schedule with 60 epochs at 0.1, 60 epochs at 0.02, 40 epochs at 0.004, 40 epochs at 0.0008, and a warmup schedule during the first epoch, for a total of 200 epochs. We reproduced each baseline on a set of 5 separate seeds, and discarded the models that produced NaNs on any of the 5 seeds of the baseline. To make the evaluation feasible, we limit the hyperparameter for TrAct to $\lambda = 0.1$. Second, we also reproduce the ResNet-18 CIFAR-10 experiment but with CIFAR-100. The results for this are displayed in Figure 10 in the Supplementary Material and demonstrate similar relations as the corresponding Figure 3. Again, all models are trained with a batch size of 128 on a single NVIDIA RTX 4090 GPU.

Model	Baseline Top-1	Baseline Top-5	TrAct (λ=0.1) Top-1	TrAct (λ=0.1) Top-5
SqueezeNet [24]	69.45%	91.09%	**70.48%**	**91.50%**
MobileNet [25]	66.99%	88.95%	**67.06%**	**89.12%**
MobileNetV2 [26]	67.76%	90.80%	**67.89%**	**90.91%**
ShuffleNet [27]	**69.98%**	91.18%	69.97%	**91.45%**
ShuffleNetV2 [28]	69.31%	90.91%	**69.88%**	**91.02%**
VGG-11 [29]	68.44%	88.02%	**69.66%**	**88.99%**
VGG-13 [29]	71.96%	90.27%	**72.98%**	**90.78%**
VGG-16 [29]	72.12%	89.81%	**72.73%**	**90.11%**
VGG-19 [29]	71.13%	88.10%	**71.45%**	**88.42%**
DenseNet121 [30]	78.93%	94.83%	**79.55%**	**94.92%**
DenseNet161 [30]	79.95%	95.25%	**80.47%**	**95.37%**
DenseNet201 [30]	79.39%	95.07%	**79.94%**	**95.17%**
GoogLeNet [31]	76.85%	93.53%	**77.18%**	**93.86%**
Inception-v3 [32]	**79.40%**	94.94%	79.24%	**95.04%**
Inception-v4 [33]	**77.32%**	93.80%	77.14%	**93.90%**
Inception-RN-v2 [33]	75.59%	93.00%	**75.73%**	**93.32%**
Xception [34]	77.57%	93.92%	**77.71%**	**93.97%**
ResNet18 [23]	76.13%	93.01%	**76.67%**	**93.29%**
ResNet34 [23]	77.34%	**93.78%**	**77.87%**	93.75%
ResNet50 [23]	78.20%	94.28%	**79.07%**	**94.67%**
ResNet101 [23]	79.07%	94.71%	**79.51%**	**94.87%**
ResNet152 [23]	78.86%	94.65%	**79.83%**	**94.96%**
ResNeXt50 [35]	78.55%	94.61%	**78.92%**	**94.80%**
ResNeXt101 [35]	79.13%	**94.85%**	**79.54%**	94.84%
ResNeXt152 [35]	79.26%	94.69%	**79.48%**	**94.89%**
SE-ResNet18 [36]	76.25%	93.09%	**76.77%**	**93.36%**
SE-ResNet34 [36]	77.85%	93.88%	**78.20%**	**94.13%**
SE-ResNet50 [36]	77.78%	94.33%	**78.79%**	**94.53%**
SE-ResNet101 [36]	77.94%	94.22%	**79.19%**	**94.70%**
SE-ResNet152 [36]	78.10%	94.46%	**79.35%**	**94.73%**
NASNet [37]	77.76%	94.26%	**78.17%**	**94.35%**
Wide-RN-40-10 [38]	78.93%	94.42%	**79.60%**	**94.80%**
StochD-RN-18 [39]	75.39%	94.09%	**75.44%**	**94.13%**
StochD-RN-34 [39]	78.03%	94.81%	**78.16%**	**94.97%**
StochD-RN-50 [39]	77.02%	94.61%	**77.40%**	**94.78%**
StochD-RN-101 [39]	78.72%	94.67%	**78.96%**	**94.75%**
Average	75.90%	93.19%	**76.39%**	**93.42%**

Table 1: Results on **CIFAR-100** trained for 200 epochs, averaged over 5 seeds. The standard deviations and results for TrAct with only 133 epochs are depicted in Tables 6 and 7 in the SM.

Results We display the results for the 36 CIFAR-100 models in Table 1. We can observe that TrAct outperforms the baseline wrt. top-1 and top-5 accuracy for 33 and 34 out of 36 models, respectively. Further, except for those 5 models, for which TrAct and the baseline perform comparably (each better on one metric), TrAct is better than vanilla training. Specifically, for 31 models, TrAct outperforms the baseline on both metrics, and the overall best result is also achieved by TrAct. Further, TrAct improves the accuracy on average by 0.49% on top-1 accuracy and by 0.23% on top-5 accuracy, a statistically very significant improvement over the baseline. The average standard deviations are 0.25% and 0.15% for top-1 and top-5 accuracy, respectively.

In addition, we also considered training the models with TrAct for only 133 epochs, i.e., 2/3 of the training time. Here, we found that, on average, regular training for 200 epochs is comparable with TrAct for 133 epochs with a small advantage for TrAct. In particular, the average accuracy of TrAct with 133 epochs is 75.94% (top-1) and 93.34% (top-5), which is a small improvement over regular training for 200 epochs. The individual results are reported in Table 7 in the Supplementary Material.

4.3 ImageNet

Finally, we consider training on the ImageNet data set [40]. We train ResNet-$\{18, 34, 50\}$, ViT-S and ViT-B models.

ResNet Setup For the ResNet-$\{18, 34, 50\}$ models, we train for $\{30, 60, 90\}$ epochs and consider base learning rates in the grid $\{0.2, 0.141, 0.1, 0.071, 0.05\}$ and determine the choice for each model / training length combination with standard baseline training. We find that for each model, when training for 30 epochs, 0.141 performs best, and, when training for $\{60, 90\}$ epochs, 0.1 performs best as the base learning rate. We use SGD with momentum (0.9), weight decay (0.0001), and the typical learning rate schedule, which decays the learning rate after 1/3 and 2/3 of training by 0.1

each. For TrAct, we (again) use the same learning rate as optimal for the baseline, and consider $\lambda \in \{0.05, 0.1, 0.2\}$. Each ResNet model is trained with a batch size of 256 on a single NVIDIA RTX 4090 GPU.

ResNet Results We start by discussing the ResNet results and then proceed with the vision transformers. We present training plots forResNet-50 in Figure 5. Here, we can observe an effective speedup of a factor of 1.5 during training, which we also demonstrate in Table 2. In particular, the difference in accuracy for TrAct ($\lambda = 0.1$) with 60 compared to the baseline with full 90 epoch training is -0.02% and $+0.04\%$ for top-1 and top-5.

	Baseline		TrAct (λ=0.1)	
Num. epochs	Top-1	Top-5	Top-1	Top-5
30	71.96%	90.70%	**73.48%**	**91.61%**
60	74.98%	92.36%	**75.68%**	**92.78%**
90	75.70%	92.74%	**76.20%**	**93.12%**

Table 2: Final test accuracies (ImageNet valid set) for training **ResNet-50** [23] on **ImageNet**. TrAct with only 60 epochs performs comparable to the baseline with 90 epochs.

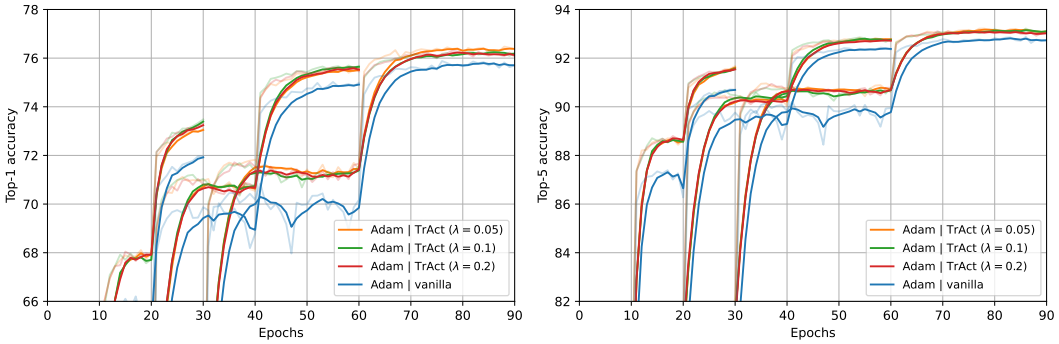

Figure 5: Test accuracy of **ResNet-50** trained on **ImageNet** for $\{30, 60, 90\}$ epochs. When training for 60 epochs with TrAct, we achieve comparable accuracy to standard training for 90 epochs, showing a $1.5\times$ speedup. Plots for ResNet-18/34 are in the SM.

ViT Setup For training the ViTs, we reproduce the "DeiT III" [5], which provides the strongest baseline that is reproducible on a single 8-GPU node. We train each model with the same hyperparameters as in the official source code[3]. We note that the ViT-S and ViT-B are both trained at a batch size of 2 048 and are pre-trained on resolutions of 224 and 192, respectively, and both models are finetuned on a resolution of 224. We consider pre-training for 400 and 800 epochs. Finetuning for each model is performed for 50 epochs. For the 400 epoch pre-training with TrAct, we use the stronger $\lambda = 0.1$, while for the longer 800 epoch pre-training we use the weaker $\lambda = 0.2$. We train the ViT-S models on 4 NVIDIA A40 GPUs and the ViT-B models on 8 NVIDIA V100 (32GB) GPUs.

ViT Results In Table 3 we present the results for training vision transformers. First, we observe that our reproductions following the official code and hyperparameters improved over the originally reported baselines, potentially due to contemporary improvements in the underlying libraries (our hardware only supported more recent versions). Notably, TrAct consistently improves upon our improved baselines. We note that we did not change any hyperparameters for training with TrAct. For ViT-S, using TrAct leads to 36% of the improvement that can be achieved by training the baseline twice as long. These improvements can be considered quite substantial considering that these are very large models and we modified only the training of the first layer. Notably, here, the runtime overheads were particularly small, ranging from 0.08% to 0.25%. Finally, we consider the quality of the pre-trained model outside

DeiT-III Model	Epochs	Top-1	Top-5
ViT-S [5]	400ep	80.4%	—
ViT-S †	400ep	81.23%	95.70%
ViT-S + TrAct (λ=0.1)	400ep	**81.50%**	**95.73%**
ViT-S [5]	800ep	81.4%	—
ViT-S †	800ep	81.97%	95.90%
ViT-S + TrAct (λ=0.2)	800ep	**82.18%**	**95.98%**
ViT-B [5]	400ep	83.5%	—
ViT-B †	400ep	83.34%	96.44%
ViT-B + TrAct (λ=0.1)	400ep	**83.58%**	**96.52%**

Table 3: Results for training **ViTs** (DeiT-III) on **ImageNet-1k**. † denotes our reproduction.

[3]Based on github.com/facebookresearch/deit.

of ImageNet. We fine-tune the ViT-S (800 epoch pre-training) model on the data sets CIFAR-10 and CIFAR-100 [22] (200 epochs), Flowers-102 [41] (5000 epochs), and Stanford Cars [42] (1000 epochs). For the baseline, both pre-training and fine-tuning were performed with the vanilla method, and, for TrAct, both pre-training and fine-tuning were performed with TrAct. In Table 4, we can observe consistent improvements for training with TrAct.

Model/Dataset	CIFAR-10	CIFAR-100	Flowers	S. Cars
ViT-S	98.94%	90.70%	94.39%	90.44%
ViT-S + TrAct	**99.02%**	**90.85%**	**95.58%**	**91.07%**

Table 4: Transfer learning results for ViT-S on CIFAR-10 and CIFAR-100 [22], Flowers-102 [41], and Stanford Cars [42].

4.4 Effect of λ

λ is the only hyperparameter introduced by TrAct. Often, with an additional hyperparameter, the hyperparameter space becomes more difficult to manage. However, for TrAct, the selection of λ is simple and compatible with existing hyperparameters. Therefore, throughout all experiments in this paper, we kept all other hyperparameters equal to the optimal choice for the respective baselines, and only considered $\lambda \in \{0.05, 0.1, 0.2\}$. A general trend is that with smaller λs, TrAct becomes more aggressive, which tends to be more favorable in shorter training, and for larger λs, TrAct is more moderate, which is ideal for longer trainings. However, in many cases, the particular choice of $\lambda \in \{0.05, 0.1, 0.2\}$ has only a subtle impact on accuracy as can be seen throughout the figures in this work. Further, going beyond this range of λs, in Fig. 6, we can observe that TrAct is robust against changes in this parameter. In all experiments, the data was as-per-convention standardized to mean 0 and standard deviation 1; deviating from this convention could change the space of λs. For significantly different tasks and drastically different kernel sizes or number of input channels, we expect that the space of λs could change. Overall, we recommend $\lambda = 0.1$ as a starting point and, for long training, we recommend $\lambda = 0.2$.

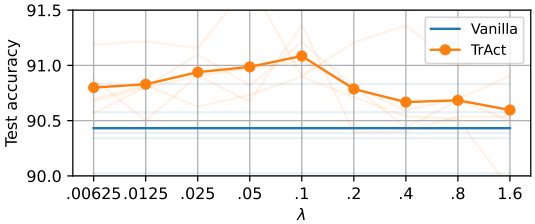

Figure 6: Effect of λ for training a ViT on CIFAR-10. Training for 200 ep., setup as Fig. 4, avg. over 5 seeds.

4.5 Ablation Study

As an ablation study, we first compare TrAct to patch-wise layer normalization for ViTs. For this, we normalize the pixel values of each input patch to mean 0 and standard deviation 1. This is an alternate solution to the conceptual problem of low contrast image regions having a lesser effect on the first layer optimization compared to higher contrast image regions. However, here, we also note that, in contrast to TrAct, the actual neural network inputs are changed through the normalization. Further, we consider DualPatchNorm [9] as a comparison, which additionally includes a second patch normalization layer after the first linear layer, and introduces additional trainable weight parameters for affine transformations into both patch normalization layers.

We use the same setup as for the CIFAR-10 ViT and run each setting for 5 seeds. The results are displayed in Figure 7. Here, we observe that patch normalization improves training for up to 400 epochs compared to the baseline; however, not as much as TrAct does. Further, we find that DualPatchNorm performs equivalently compared to input patch normalization and worse than TrAct, except for the case of 200 epochs where it performs insignificantly better than TrAct. For training for 800 epochs, patch normalization and DualPatchNorm do not improve the baseline and perform insignificantly worse, whereas TrAct still shows accuracy improvements. This effect may be explained by the fact that patch normalization is a scalar form of whitening, and whitening can hurt generalization capabilities due to a loss of information [43]. In particular, what may be problematic is that patch normalization also affects the model behavior during inference, which contrasts TrAct.

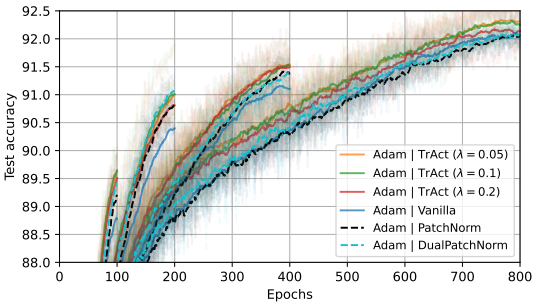

Figure 7: Ablation Study: training a ViT on CIFAR-10, including patch normalization (black, dashed) and DualPatchNorm (cyan, dashed). Setups as in Figure 4, averaged over 5 seeds.

As a second ablation study, we examine what happens if we (against convention) do not perform standardization of the data set. We train the same ViTs as above on CIFAR-10 for 200 epochs, averaged over 5 seeds. We consider two cases: first, an input value range of $[0, 1]$ and a quite extreme input value range of $[0, 255]$.

We display the results in Figure 8. Here, we observe that TrAct is more robust against a lack of standardization. Interestingly, we observe that TrAct performs better for the range of $[0, 255]$ than $[0, 1]$. The reason for this is that TrAct suffers from obtaining only positive inputs, which affects the $xx^\top$ matrix in Equation 10; however, we note that regular training suffers even more from the lack of standardization. When considering the range of $[0, 255]$, we observe that TrAct is virtually agnostic to λ, which is caused by the $xx^\top$ matrix becoming very large. The reason why TrAct performs so well here (compared to the baseline) is that, due to the large $xx^\top$, the updates ΔW become very small. This is more desirable compared to the standard gradient, which explodes due to its proportionality to the input values, and therefore drastically degrades training.

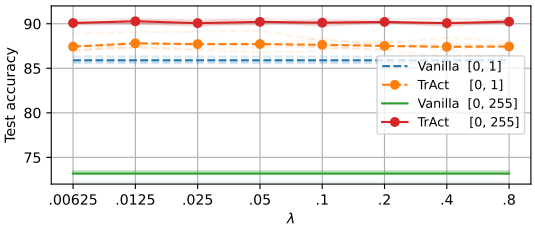

Figure 8: Ablation Study: training a ViT on CIFAR-10 *without* data standardization and with input value ranges of $[0, 1]$ vs. $[0, 255]$. Setups as in Figure 4, 200 epochs, and avg. over 5 seeds. All other experiments in this work are trained *with* data standardization.

In each experiment, we used only a single optimizer for the entire model; however, our theory assumes that TrAct is used with SGD. This motivates the question of whether it is advantageous to train the first layer with SGD, while training the remainder of the model, e.g., with Adam.

Thus, as a final ablation study, we extend the experiment from Figure 3 (right) by training the first layer with SGD while training the remaining model with Adam. We display the results in Figure 9 where we can observe small improvements when using SGD for the TrAct layer.

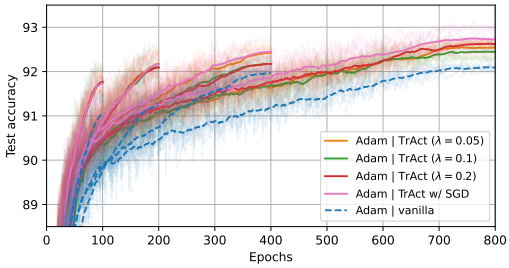

Figure 9: Ablation Study: extending Figure 3 (right) by training the first layer with TrAct and SGD (pink) and the remainder of the model still with Adam.

4.6 Runtime Analysis

In this section, we provide a training runtime analysis. Overall, the trend is that, for large models, TrAct adds on a tiny runtime overhead, while it can become more expensive for smaller models. In particular, for the CIFAR-10 ViT, the average training time per 100 epochs increased by 9.7% from 1091s to 1197s. Much of this can be attributed to the required additional CUDA calls and non-fused operations, which can be expensive for cheaper tasks. However, when considering larger models, this overhead almost entirely amortizes. In particular the ViT-S (800 epochs) pre-training cost increased by only 0.08% from 133:52 hours to 133:58 hours. The pre-training cost of the ViT-B (400 epochs) increased by 0.25% from 98:28 hours to 98:43 hours. We can see that, in each case, the training cost overhead is clearly more than worth the reduced requirement of epochs already. Further, fused kernels could drastically reduce the computational overhead; in particular, our current implementation replaces an existing fused operation by multiple calls from the Python space. As TrAct only affects training, and the modification isn't present during forwarding, TrAct has no effect on inference time.

5 Discussion & Conclusion

In this work, we introduced TrAct, a novel training strategy that modifies the optimization behavior of the first layer, leading to significant performance improvements across a range of 50 experimental setups. The approach is efficient and effectively speeds up training by factors between $1.25\times$ and $4\times$ depending on the model size. We hope that the simplicity of integration into existing training schemes as well as the robust performance improvements motivate the community to adopt TrAct.

Acknowledgments and Disclosure of Funding

This work was in part supported by the Federal Agency for Disruptive Innovation SPRIN-D, the Land Salzburg within the WISS 2025 project IDA-Lab (20102-F1901166-KZP and 20204-WISS/225/197-2019), the ARO (W911NF-21-1-0125), the ONR (N00014-23-1-2159), and the CZ Biohub.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Neural Information Processing Systems (NeurIPS)*, 2017.

[2] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, 2019.

[3] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, and S. G. *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," in *Proc. International Conference on Learning Representations (ICLR)*, 2020.

[4] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *Proc. International Conference on Machine Learning (ICML)*, 2021.

[5] H. Touvron, M. Cord, and H. Jégou, "DeiT III: Revenge of the ViT," in *Proc. European Conference on Computer Vision (ECCV)*, 2022.

[6] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. International Conference on Machine Learning (ICML)*, 2015.

[7] T. Salimans and D. P. Kingma, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," in *Proc. Conf. Neural Information Processing Systems (NeurIPS 2016, Barcelona, Spain)*, Neural Information Processing Systems Foundation, 2016.

[8] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *Computing Research Repository (CoRR) in arXiv*, 2016.

[9] M. Kumar, M. Dehghani, and N. Houlsby, "Dual patchnorm," *Trans. on Machine Learning Research*, 2023.

[10] F. Dangel, "Backpropagation Beyond the Gradient," Ph.D. dissertation, University of Tübingen, 2023.

[11] S. J. Wright, *Numerical optimization*. 2006.

[12] J. Martens and R. Grosse, "Optimizing neural networks with kronecker-factored approximate curvature," in *Proc. International Conference on Machine Learning (ICML)*, 2015.

[13] F. Dangel, L. Tatzel, and P. Hennig, "ViViT: Curvature access through the generalized Gauss-Newton's low-rank structure," *Transactions on Machine Learning Research*, 2022.

[14] F. Petersen, T. Sutter, C. Borgelt, D. Huh, H. Kuehne, Y. Sun, and O. Deussen, "ISAAC Newton: Input-based approximate curvature for Newton's method," in *Proc. International Conference on Learning Representations (ICLR)*, 2023.

[15] F. Dangel, F. Kunstner, and P. Hennig, "Backpack: Packing more into backprop," in *Proc. International Conference on Learning Representations (ICLR)*, 2020.

[16] F. Petersen, C. Borgelt, T. Sutter, H. Kuehne, O. Deussen, and S. Ermon, "Newton losses: Using curvature information for learning with differentiable algorithms," in *Conference on Neural Information Processing Systems (NeurIPS)*, 2024.

[17] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.

[18] D. Calvetti and L. Reichel, "Tikhonov regularization with a solution constraint," *SIAM Journal on Scientific Computing*, vol. 26, no. 1, pp. 224–239, 2004.

[19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Proc. Neural Information Processing Systems (NeurIPS)*, 2019.

[20] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, *JAX: Composable transformations of Python+NumPy programs*, 2018. [Online]. Available: http://github.com/google/jax.

[21] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. International Conference on Learning Representations (ICLR)*, 2015.

[22] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research)," 2009. [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html.

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[24] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size," *Computing Research Repository (CoRR) in arXiv*, 2016.

[25] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *Computing Research Repository (CoRR) in arXiv*, 2017.

[26] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proc. International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[27] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proc. International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[28] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *Proc. European Conference on Computer Vision (ECCV)*, 2018.

[29] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *Computing Research Repository (CoRR) in arXiv*, 2014.

[30] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[31] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions (2014)," *Computing Research Repository (CoRR) in arXiv*, 2014.

[32] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[33] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *AAAI Conference on Artificial Intelligence*, 2017.

[34] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proc. International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[35] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proc. International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[36] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proc. International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[37] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[38] S. Zagoruyko and N. Komodakis, "Wide residual networks," *Computing Research Repository (CoRR) in arXiv*, 2016.

[39] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *Proc. European Conference on Computer Vision (ECCV)*, 2016.

[40] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proc. International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[41] M.-E. Nilsback and A. Zisserman, "Automated flower classification over a large number of classes," in *2008 Sixth Indian conference on computer vision, graphics & image processing*, IEEE, 2008.

[42] J. Krause, J. Deng, M. Stark, and L. Fei-Fei, "Collecting a large-scale dataset of fine-grained cars," 2013.

[43] N. Wadia, D. Duckworth, S. S. Schoenholz, E. Dyer, and J. Sohl-Dickstein, "Whitening and second order optimization both make information in the dataset unusable during training, and can reduce or prevent generalization," in *Proc. International Conference on Machine Learning (ICML)*, 2021.

[44] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Neural Information Processing Systems (NeurIPS)*, 2015.

[45] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, *The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results*. [Online]. Available: http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

A Theory

Lemma 1. *The solution $\Delta W_i^\star$ of Equation 7 is*

$$\Delta W_i^\star = -\eta \cdot \nabla_{z_i}\mathcal{L}(z) \cdot x^\top \cdot \left(\frac{xx^\top}{b} + \lambda \cdot I_n\right)^{-1}. \tag{13}$$

Proof. We would like to solve the optimization problem

$$\arg\min_{\Delta W_i} \; \| -\eta b\, \nabla_{z_i}\mathcal{L}(z) - \Delta W_i x\|_2^2 \; + \; \lambda b\, \|\Delta W_i\|_2^2.$$

A necessary condition for a minimum of the functional

$$F(\Delta W_i) = (-\eta b\, \nabla_{z_i}\mathcal{L}(z) - \Delta W_i x)^2 \; + \; \lambda b\, (\Delta W_i)(\Delta W_i)^\top$$

is that $\nabla_{\Delta W_i} F(\Delta W_i)$ vanishes:

$$\begin{aligned}
\nabla_{\Delta W_i} F(\Delta W_i) &= \nabla_{\Delta W_i}(-\eta b\, \nabla_{z_i}\mathcal{L}(z) - \Delta W_i x)^2 + \lambda b \nabla_{\Delta W_i}((\Delta W_i)(\Delta W_i)^\top) \\
&= 2(-\eta b\, \nabla_{z_i}\mathcal{L}(z) - \Delta W_i x)(\nabla_{\Delta W_i}(-\eta b\, \nabla_{z_i}\mathcal{L}(z) - \Delta W_i x)) + 2\lambda b\, \Delta W_i \\
&= 2(-\eta b\, \nabla_{z_i}\mathcal{L}(z) - \Delta W_i x)(-x)^\top + 2\lambda b\, \Delta W_i \\
&= 2(\eta b\, \nabla_{z_i}\mathcal{L}(z) + \Delta W_i x)\, x^\top + 2\lambda b\, \Delta W_i \quad \stackrel{!}{=} \quad 0.
\end{aligned}$$

It follows for the optimal $\Delta W_i^\star$ that minimizes $F(\Delta W_i)$

$$\begin{aligned}
&\eta b\, (\nabla_{z_i}\mathcal{L}(z))\, x^\top + \Delta W_i^\star x x^\top + \lambda b\, \Delta W_i^\star = 0 \\
\Leftrightarrow \quad & -\eta b\, (\nabla_{z_i}\mathcal{L}(z))\, x^\top = \Delta W_i^\star (x x^\top + \lambda b I_n) \\
\Leftrightarrow \quad & \Delta W_i^\star = -\eta b\, (\nabla_{z_i}\mathcal{L}(z))\, x^\top (x x^\top + \lambda b I_n)^{-1} \\
\Leftrightarrow \quad & \Delta W_i^\star = -\eta\, (\nabla_{z_i}\mathcal{L}(z))\, x^\top \left(\frac{xx^\top}{b} + \lambda I_n\right)^{-1}.
\end{aligned}$$

□

Lemma 2. *Using TrAct does not change the set of possible convergence points compared to vanilla (full batch) gradient descent. Herein, we use the standard definition of convergence points as those points where no update is performed because the gradient is zero.*

Proof. First, we remark that only the training of the first layer is affected by TrAct. To show the statement, we show that (i) a zero gradient for GD implies that TrAct also performs no update and that (ii) TrAct performing no update implies zero gradients for GD.

(i) In the first case, we assume that gradient descent has converged, i.e., the gradient wrt. first layer weights is zero $\nabla_W \mathcal{L}(W) = \mathbf{0}$. We want to show that, in this case, our proposed update is also zero, i.e., $\Delta W^\star = \mathbf{0}$. Using the definition of $\Delta W^\star$ from Equation 9, we have

$$\Delta W^\star = -\eta \cdot \nabla_z \mathcal{L}(z) \cdot x^\top \cdot \left(\frac{xx^\top}{b} + \lambda \cdot I_n\right)^{-1} \tag{14}$$

$$= -\eta \cdot \nabla_W \mathcal{L}(W) \cdot \left(\frac{xx^\top}{b} + \lambda \cdot I_n\right)^{-1} \tag{15}$$

$$= -\eta \cdot \mathbf{0} \cdot \left(\frac{xx^\top}{b} + \lambda \cdot I_n\right)^{-1} = \mathbf{0}\,, \tag{16}$$

which shows this direction.

(ii) In the second case, we have $\Delta W^\star = \mathbf{0}$ and need to show that this implies $\nabla_W \mathcal{L}(W) = \mathbf{0}$. For this, we can observe that $\left(xx^\top / b + \lambda \cdot I_n\right)^{-1}$ is PD (positive definite) by definition and

$\left(xx^\top / b + \lambda \cdot I_n\right)$ also exists. If $\Delta W^\star = \mathbf{0}$, then

$$\mathbf{0} = \Delta W^\star = \Delta W^\star \left(\frac{xx^\top}{b} + \lambda \cdot I_n\right) \tag{17}$$

$$= -\eta \cdot \nabla_z \mathcal{L}(z) \cdot x^\top \cdot \left(\frac{xx^\top}{b} + \lambda \cdot I_n\right)^{-1} \left(\frac{xx^\top}{b} + \lambda \cdot I_n\right)$$

$$= -\eta \cdot \nabla_z \mathcal{L}(z) \cdot x^\top = -\eta \cdot \nabla_W \mathcal{L}(W)\,, \tag{18}$$

which also shows this direction.

Overall, we showed that if gradient descent has converged according to the standard notion of a zero gradient, then our update has also converged and vice versa.

□

B Additional Results

We display additional results in Figures 10, 11, and 12 as well as in Tables 6 and 7.

As an additional experiment, in order to verify the applicability of TrAct beyond training / pre-training, we train Faster R-CNN models [44] on PASCAL VOC2007 [45] using a VGG-16 backbone [29]. However, Faster R-CNN uses a pretrained vision encoder where the first 4 layers are frozen. In order to enable TrAct, as TrAct only affects the training of the first layer, we unfreeze these first layers when training the object detection head. The mean average precision (mAP) on test data for the vanilla model versus TrAct training are shown in Table 5.

vanilla	TrAct
0.659 ± 0.005	0.671 ± 0.004

Table 5: Mean average precision (mAP) on test data for Faster R-CNN [44] with a VGG-16 backbone on PASCAL VOC2007 [45], averaged over 2 seeds.

We can observe that TrAct performs better than the vanilla method by about 1.1%. We would like to point out that, while is TrAct especially designed for speeding up pretraining or training from scratch, i.e., when actually learning the first layer, we find that it also helps in finetuning pretrained models. Here, a limitation is of course that TrAct requires actually training the first layer.

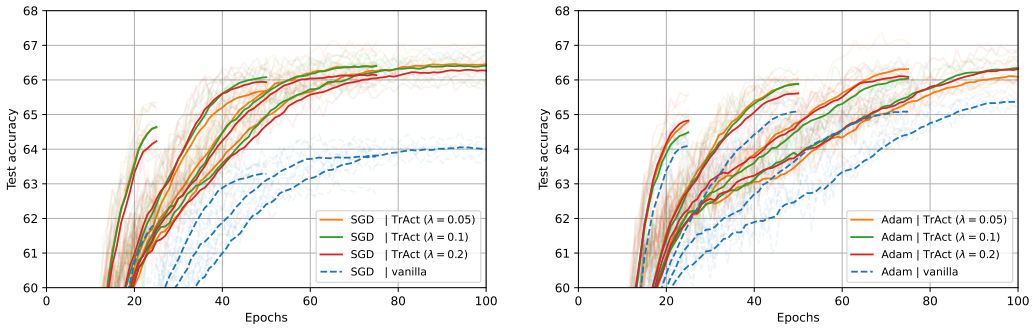

Figure 10: Training a **ResNet-18** on **CIFAR-100** with the CIFAR-10 setup from Section 4.1. Displayed is top-1 accuracy. We train for $\{100, 200, 400, 800\}$ epochs using a cosine learning rate schedule and with SGD (left) and Adam (right). Learning rates have been selected as optimal for each baseline. Averaged over 5 seeds. TrAct (solid lines) consistently outperforms the baselines (dashed lines).

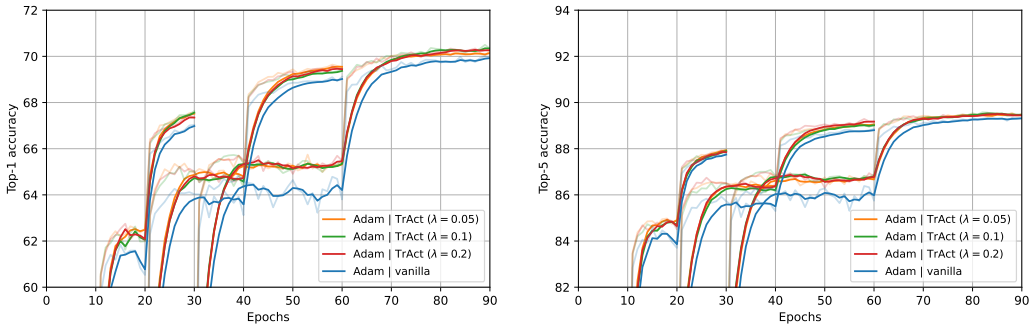

Figure 11: Test accuracy of **ResNet-18** trained on **ImageNet** for $\{30, 60, 90\}$ epochs. Displayed is the top-1 (left) and top-5 (right) accuracy.

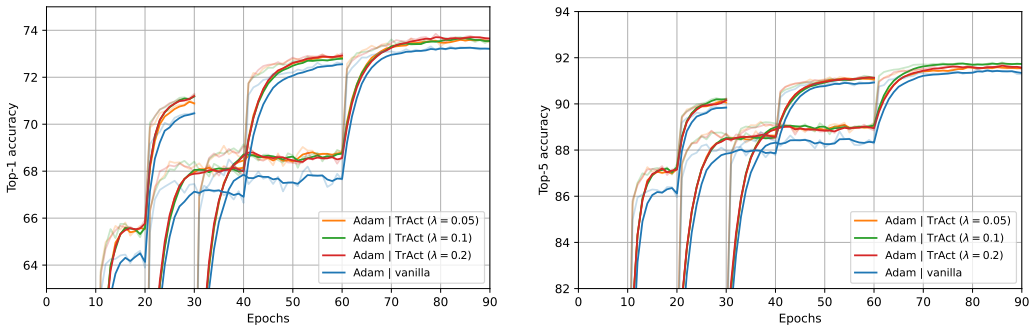

Figure 12: Test accuracy of **ResNet-34** trained on **ImageNet** for $\{30, 60, 90\}$ epochs. Displayed is the top-1 (left) and top-5 (right) accuracy.

Model	Baseline		TrAct (λ=0.1)	
	Top-1	Top-5	Top-1	Top-5
SqueezeNet [24]	$69.45\% \pm 0.30\%$	$91.09\% \pm 0.20\%$	$70.48\% \pm 0.17\%$	$91.50\% \pm 0.13\%$
MobileNet [25]	$66.99\% \pm 0.16\%$	$88.95\% \pm 0.07\%$	$67.06\% \pm 0.41\%$	$89.12\% \pm 0.16\%$
MobileNetV2 [26]	$67.76\% \pm 0.20\%$	$90.80\% \pm 0.10\%$	$67.89\% \pm 0.22\%$	$90.91\% \pm 0.11\%$
ShuffleNet [27]	$69.98\% \pm 0.22\%$	$91.18\% \pm 0.12\%$	$69.97\% \pm 0.30\%$	$91.45\% \pm 0.29\%$
ShuffleNetV2 [28]	$69.31\% \pm 0.13\%$	$90.91\% \pm 0.15\%$	$69.88\% \pm 0.26\%$	$91.02\% \pm 0.08\%$
VGG-11 [29]	$68.44\% \pm 0.24\%$	$88.02\% \pm 0.10\%$	$69.66\% \pm 0.20\%$	$88.99\% \pm 0.21\%$
VGG-13 [29]	$71.96\% \pm 0.26\%$	$90.27\% \pm 0.17\%$	$72.98\% \pm 0.18\%$	$90.78\% \pm 0.15\%$
VGG-16 [29]	$72.12\% \pm 0.24\%$	$89.81\% \pm 0.19\%$	$72.73\% \pm 0.16\%$	$90.11\% \pm 0.15\%$
VGG-19 [29]	$71.13\% \pm 0.46\%$	$88.10\% \pm 0.36\%$	$71.45\% \pm 0.34\%$	$88.42\% \pm 0.46\%$
DenseNet121 [30]	$78.93\% \pm 0.28\%$	$94.83\% \pm 0.13\%$	$79.55\% \pm 0.25\%$	$94.92\% \pm 0.11\%$
DenseNet161 [30]	$79.95\% \pm 0.21\%$	$95.25\% \pm 0.19\%$	$80.47\% \pm 0.25\%$	$95.37\% \pm 0.12\%$
DenseNet201 [30]	$79.39\% \pm 0.20\%$	$95.07\% \pm 0.12\%$	$79.94\% \pm 0.19\%$	$95.17\% \pm 0.10\%$
GoogLeNet [31]	$76.85\% \pm 0.14\%$	$93.53\% \pm 0.16\%$	$77.18\% \pm 0.11\%$	$93.86\% \pm 0.10\%$
Inception-v3 [32]	$79.40\% \pm 0.15\%$	$94.94\% \pm 0.21\%$	$79.24\% \pm 0.33\%$	$95.04\% \pm 0.06\%$
Inception-v4 [33]	$77.32\% \pm 0.36\%$	$93.80\% \pm 0.33\%$	$77.14\% \pm 0.28\%$	$93.90\% \pm 0.20\%$
Inception-RN-v2 [33]	$75.59\% \pm 0.45\%$	$93.00\% \pm 0.18\%$	$75.73\% \pm 0.30\%$	$93.32\% \pm 0.19\%$
Xception [34]	$77.57\% \pm 0.31\%$	$93.92\% \pm 0.17\%$	$77.71\% \pm 0.17\%$	$93.97\% \pm 0.10\%$
ResNet18 [23]	$76.13\% \pm 0.27\%$	$93.01\% \pm 0.06\%$	$76.67\% \pm 0.26\%$	$93.29\% \pm 0.22\%$
ResNet34 [23]	$77.34\% \pm 0.33\%$	$93.78\% \pm 0.16\%$	$77.87\% \pm 0.25\%$	$93.75\% \pm 0.10\%$
ResNet50 [23]	$78.20\% \pm 0.35\%$	$94.28\% \pm 0.09\%$	$79.07\% \pm 0.18\%$	$94.67\% \pm 0.07\%$
ResNet101 [23]	$79.07\% \pm 0.22\%$	$94.71\% \pm 0.20\%$	$79.51\% \pm 0.43\%$	$94.87\% \pm 0.06\%$
ResNet152 [23]	$78.86\% \pm 0.28\%$	$94.65\% \pm 0.22\%$	$79.83\% \pm 0.22\%$	$94.96\% \pm 0.09\%$
ResNeXt50 [35]	$78.55\% \pm 0.22\%$	$94.61\% \pm 0.16\%$	$78.92\% \pm 0.14\%$	$94.80\% \pm 0.12\%$
ResNeXt101 [35]	$79.13\% \pm 0.33\%$	$94.85\% \pm 0.14\%$	$79.54\% \pm 0.25\%$	$94.84\% \pm 0.10\%$
ResNeXt152 [35]	$79.26\% \pm 0.29\%$	$94.69\% \pm 0.11\%$	$79.48\% \pm 0.16\%$	$94.89\% \pm 0.17\%$
SE-ResNet18 [36]	$76.25\% \pm 0.18\%$	$93.09\% \pm 0.19\%$	$76.77\% \pm 0.10\%$	$93.36\% \pm 0.09\%$
SE-ResNet34 [36]	$77.85\% \pm 0.19\%$	$93.88\% \pm 0.15\%$	$78.20\% \pm 0.16\%$	$94.13\% \pm 0.21\%$
SE-ResNet50 [36]	$77.78\% \pm 0.26\%$	$94.33\% \pm 0.12\%$	$78.79\% \pm 0.11\%$	$94.53\% \pm 0.24\%$
SE-ResNet101 [36]	$77.94\% \pm 0.49\%$	$94.22\% \pm 0.10\%$	$79.19\% \pm 0.37\%$	$94.70\% \pm 0.13\%$
SE-ResNet152 [36]	$78.10\% \pm 0.47\%$	$94.46\% \pm 0.13\%$	$79.35\% \pm 0.27\%$	$94.73\% \pm 0.15\%$
NASNet [37]	$77.76\% \pm 0.19\%$	$94.26\% \pm 0.28\%$	$78.17\% \pm 0.11\%$	$94.35\% \pm 0.21\%$
Wide-RN-40-10 [38]	$78.93\% \pm 0.07\%$	$94.42\% \pm 0.09\%$	$79.60\% \pm 0.18\%$	$94.80\% \pm 0.12\%$
StochD-RN-18 [39]	$75.39\% \pm 0.14\%$	$94.09\% \pm 0.10\%$	$75.44\% \pm 0.33\%$	$94.13\% \pm 0.17\%$
StochD-RN-34 [39]	$78.03\% \pm 0.33\%$	$94.81\% \pm 0.08\%$	$78.16\% \pm 0.39\%$	$94.97\% \pm 0.10\%$
StochD-RN-50 [39]	$77.02\% \pm 0.18\%$	$94.61\% \pm 0.13\%$	$77.40\% \pm 0.24\%$	$94.78\% \pm 0.10\%$
StochD-RN-101 [39]	$78.72\% \pm 0.12\%$	$94.67\% \pm 0.05\%$	$78.96\% \pm 0.27\%$	$94.75\% \pm 0.05\%$
Average (avg. std)	75.90% (0.26%)	93.19% (0.15%)	76.39% (0.24%)	93.42% (0.14%)

Table 6: Results on CIFAR-100, trained for 200 epochs, averaged over 5 seeds including standard deviations.

Model	TrAct (λ=0.1, 133 ep) Top-1	Top-5
SqueezeNet [24]	$70.36\% \pm 0.30\%$	$91.69\% \pm 0.16\%$
MobileNet [25]	$67.45\% \pm 0.38\%$	$89.41\% \pm 0.13\%$
MobileNetV2 [26]	$68.01\% \pm 0.32\%$	$90.90\% \pm 0.13\%$
ShuffleNet [27]	$70.31\% \pm 0.32\%$	$91.67\% \pm 0.25\%$
ShuffleNetV2 [28]	$70.09\% \pm 0.34\%$	$91.20\% \pm 0.20\%$
VGG-11 [29]	$69.14\% \pm 0.13\%$	$88.92\% \pm 0.18\%$
VGG-13 [29]	$72.53\% \pm 0.26\%$	$90.81\% \pm 0.12\%$
VGG-16 [29]	$72.11\% \pm 0.10\%$	$90.28\% \pm 0.10\%$
VGG-19 [29]	$70.54\% \pm 0.46\%$	$88.48\% \pm 0.20\%$
DenseNet121 [30]	$79.09\% \pm 0.21\%$	$94.79\% \pm 0.11\%$
DenseNet161 [30]	$80.20\% \pm 0.12\%$	$95.30\% \pm 0.11\%$
DenseNet201 [30]	$79.99\% \pm 0.20\%$	$95.12\% \pm 0.16\%$
GoogLeNet [31]	$76.59\% \pm 0.35\%$	$93.83\% \pm 0.18\%$
Inception-v3 [32]	$78.70\% \pm 0.22\%$	$94.76\% \pm 0.16\%$
Inception-v4 [33]	$76.50\% \pm 0.46\%$	$93.56\% \pm 0.21\%$
Inception-RN-v2 [33]	$75.15\% \pm 0.24\%$	$92.99\% \pm 0.29\%$
Xception [34]	$77.55\% \pm 0.34\%$	$93.90\% \pm 0.14\%$
ResNet18 [23]	$75.86\% \pm 0.20\%$	$93.07\% \pm 0.07\%$
ResNet34 [23]	$77.29\% \pm 0.23\%$	$93.72\% \pm 0.17\%$
ResNet50 [23]	$78.44\% \pm 0.27\%$	$94.47\% \pm 0.11\%$
ResNet101 [23]	$79.20\% \pm 0.17\%$	$94.77\% \pm 0.11\%$
ResNet152 [23]	$79.34\% \pm 0.21\%$	$94.92\% \pm 0.08\%$
ResNeXt50 [35]	$78.90\% \pm 0.16\%$	$94.75\% \pm 0.06\%$
ResNeXt101 [35]	$79.09\% \pm 0.15\%$	$94.78\% \pm 0.08\%$
ResNeXt152 [35]	$78.91\% \pm 0.18\%$	$94.67\% \pm 0.12\%$
SE-ResNet18 [36]	$76.51\% \pm 0.43\%$	$93.29\% \pm 0.16\%$
SE-ResNet34 [36]	$77.81\% \pm 0.15\%$	$94.02\% \pm 0.18\%$
SE-ResNet50 [36]	$78.32\% \pm 0.22\%$	$94.47\% \pm 0.14\%$
SE-ResNet101 [36]	$79.07\% \pm 0.12\%$	$94.79\% \pm 0.31\%$
SE-ResNet152 [36]	$79.03\% \pm 0.49\%$	$94.74\% \pm 0.10\%$
NASNet [37]	$77.85\% \pm 0.22\%$	$94.34\% \pm 0.16\%$
Wide-RN-40-10 [38]	$79.37\% \pm 0.25\%$	$94.72\% \pm 0.07\%$
StochD-RN-18 [39]	$74.11\% \pm 0.16\%$	$93.75\% \pm 0.13\%$
StochD-RN-34 [39]	$76.83\% \pm 0.31\%$	$94.61\% \pm 0.19\%$
StochD-RN-50 [39]	$75.87\% \pm 0.29\%$	$94.28\% \pm 0.18\%$
StochD-RN-101 [39]	$77.73\% \pm 0.20\%$	$94.55\% \pm 0.02\%$
Average (avg. std)	75.94% (0.25%)	93.34% (0.15%)

Table 7: Results on CIFAR-100, trained for 133 epochs, averaged over 5 seeds including standard deviations.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We address all claims made in the abstract and introduction in the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: All assumptions are pointed out in the work. Limitations are discussed.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: All assumptions are provided. Proofs are provided in the SM.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We discuss all experimental parameters necessary for reproduction.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The code will be made publicly available at github.com/Felix-Petersen/tract.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: All training details are either explicitly discussed in the main paper or inherited from the references.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Standard deviations are reported and correctly defined, described as such, and utilize Bessel's correction. In some experiments, we display each seed's accuracy. In the ImageNet ViT experiments, running multiple seeds was not feasible due to compute cost.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We describe the hardware for each experiment and a provide runtime analysis section.

9. **Code Of Ethics**

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

Answer: [Yes]

Justification: The research conducted in this paper is conforming with the NeurIPS Code of Ethics. No animal were harmed in the execution of this research.

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The proposed method reduces training cost of vision models or improves vision models under the same computational budget.

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All utilized assets (data sets, models, and code bases) are cited.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]