Scaled ReLU Matters for Training Vision Transformers

Pichao Wang*, Xue Wang*, Hao Luo, Jingkai Zhou, Zhipeng Zhou, Fan Wang, Hao Li, Rong Jin
Alibaba Group
{pichao.wang,xue.w}@alibaba-inc.com
{michuan.lh,zhoujingkai.zjk,yuer.zzp,fan.w,lihao.lh,jinrong.jr}@alibaba-inc.com

Abstract

Vision transformers (ViTs) have been an alternative design paradigm to convolutional neural networks (CNNs). However, the training of ViTs is much harder than CNNs, as it is sensitive to the training parameters, such as learning rate, optimizer and warmup epoch. The reasons for training difficulty are empirically analysed in (Xiao et al. 2021), and the authors conjecture that the issue lies with the patchify-stem of ViT models and propose that early convolutions help transformers see better. In this paper, we further investigate this problem and extend the above conclusion: only early convolutions do not help for stable training, but the scaled ReLU operation in the convolutional stem (conv-stem) matters. We verify, both theoretically and empirically, that scaled ReLU in conv-stem not only improves training stabilization, but also increases the diversity of patch tokens, thus boosting peak performance with a large margin via adding few parameters and flops. In addition, extensive experiments are conducted to demonstrate that previous ViTs are far from being well trained, further showing that ViTs have great potential to be a better substitute of CNNs.

Introduction

Visual recognition has been dominated by convolutional neural networks (CNNs) (He et al. 2016; Howard et al. 2017; Zhang et al. 2018; Tan and Le 2019; Li et al. 2021a; Zhou et al. 2021c) for years, which effectively impose spatial locality and translation equivalence. Recently the prevailing vision transformers (ViTs) are regarded as an alternative design paradigm, which target to replace the inductive bias towards local processing inherent in CNNs with global self-attention (Dosovitskiy et al. 2020; Touvron et al. 2020; Wang et al. 2021b; Fan et al. 2021).

Despite the appealing potential of ViTs for complete data-driven training, the lack of convolution-like inductive bias also challenges the training of ViTs. Compared with CNNs, ViTs are sensitive to the choice of optimizer, data augmentation, learning rate, training schedule length and warmup epoch (Touvron et al. 2020, 2021; Chen, Hsieh, and Gong 2021; Xiao et al. 2021). The reasons for training difficulty are empirically analysed in (Xiao et al. 2021), and the authors conjecture that the issue lies with the patchify stem of ViT models and propose that early convolutions help transformers see better. Recent works (Graham et al. 2021; Gu et al. 2021; Yuan et al. 2021c) also introduce the conv-stem to improve the robustness of training vision transformer, but they lack the deep analysis why such conv-stem works.

In this paper, we theoretically and empirically verify that scaled ReLU in the conv-stem matters for the robust ViTs training. Specifically, scaled ReLU not only improves the training stabilization, but also increases the diversity of patch tokens, thus boosting the final recognition performances by a large margin. In addition, extensive experiments are conducted to further unveil the effects of conv-stem and the following interesting observations are made: firstly, after adding conv-stem to the ViTs, the SAM optimizer (Foret et al. 2020) is no longer powerful as reported in (Chen, Hsieh, and Gong 2021); secondly, with conv-stem, the supervised ViTs (Touvron et al. 2020) are better than its corresponding self-supervised trained models (Caron et al. 2021) plus supervised finetuning on ImageNet-1k; thirdly, using conv-stem the better trained ViTs improve the performance of downstream tasks. All of these observations reflect that previous ViTs are far from being well trained and ViTs may become a better substitute for CNNs.

Related Work

Convolutional neural networks (CNNs). Since the breakthrough performance on ImageNet via AlexNet (Krizhevsky, Sutskever, and Hinton 2012), CNNs have become a dominant architecture in computer vision field. Following the primary design rule of stacking low-to-high convolutions in series by going deeper, many popular architectures are proposed, such as VGG (Simonyan and Zisserman 2014), GoogleNet (Szegedy et al. 2015) and ResNet (He et al. 2016). To further exploit the capacity of visual representation, many innovations have been proposed, such as ResNeXt (Xie et al. 2017), SENet (Hu, Shen, and Sun 2018), EfficientNet (Tan and Le 2019) and NFNets (Brock et al. 2021). For most of these CNNs, Conv+BN+ReLU becomes a standard block. In this paper, we investigate this basic block for training vision transformers as a lightweight stem.

Vision Transformers (ViTs). Since Dosovitskiy et al. (Dosovitskiy et al. 2020) first successfully applies transformer for image classification by dividing the images into non-overlapping patches, many ViT variants are pro-
posed [Wang et al. 2021b; Han et al. 2021; Chen et al. 2021a; Ranftl, Bochkovskiy, and Kolotouros 2021; Liu et al. 2021; Chen, Fan, and Panda 2021; Zhang et al. 2021a; Xie et al. 2021; Zhang et al. 2021b; Jonnalagadda, Wang, and Eckestein 2021; Wang et al. 2021a; Fang et al. 2021; Huang et al. 2021; Gao et al. 2021; Yao et al. 2021; Yu et al. 2021; Zhou et al. 2021b; El-Nouby et al. 2021; Wang et al. 2021c; Xu et al. 2021]. In this section, we mainly review several closely related works for training ViTs. Specifically, DeiT (Touvron et al. 2020) adopts several training techniques (e.g. truncated normal initialization, strong data augmentation and smaller weight decay) and uses distillation to extend ViT to a data-efficient version. T2T ViT (Yuan et al. 2021b), CeViT (Yuan et al. 2021a), and CvT (Wu et al. 2021) adopt several techniques including convolution operation for patch sequence generation to facilitate the training; DeepViT (Zhou et al. 2021a), CaViT (Touvron et al. 2021) and PatchViT (Gong et al. 2021) investigate the unstable training problem, and propose the re-attention, re-scale and anti-over-smoothing techniques respectively for stable training; to accelerate the convergence of training, ConViT (d’Ascoli et al. 2021), PiViT (Heo et al. 2021), CeViT (Yuan et al. 2021a), LocalViT (Li et al. 2021b) and VisFormer (Chen et al. 2021b) introduce convolutional bias to speedup the training; LV-ViT (Jiang et al. 2021) adopts several techniques including MixToken and Token Labeling for better training and feature generation; the SAM optimizer (Foret et al. 2020) is adopted in (Chen, Hsieh, and Gong 2021) to better train ViTs without strong data augmentation; KVT (Wang et al. 2021a) introduces the k-NN attention to filters out irrelevant tokens to speedup the training; conv-stem is adopted in several works (Graham et al. 2021; Xiao et al. 2021; Guo et al. 2021; Yuan et al. 2021c) to improve the robustness of training ViTs. In this paper, we investigate the training of ViTs by using the conv-stem and demonstrate several properties of conv-stem in the context of vision transformers, both theoretically and empirically.

Vision Transformer Architectures

In this section, we first review the vision transformer, namely ViT (Dosovitskiy et al. 2020), and then describe the conv-stem used in our work.

ViT. ViT first divides an input image into non-overlapping p x p patches and linearly projects each patch to a d-dimensional feature vector using a learned weight matrix. The typical patch and image size are p = 16 and 224x224, respectively. The patch embeddings together with added positional embeddings and a concatenated classification token are fed into a standard transformer encoder (Vaswani et al. 2017) followed by a classification head. Similar as (Xiao et al. 2021), we name the portion of ViT before the transformer blocks as ViT-stem, and call the linear projection (stride-p, p x p kernel) as patchify-stem.

Conv-stem. Unless otherwise specified, we adopt the conv-stem from VOLO (Yuan et al. 2021c). The full conv-stem consists of 3Conv+3BN+3ReLU+1Proj blocks, and the kernel sizes and strides are (7,3,8) and (2,1,1,8), respectively. The detailed configurations are shown in Algorithm 1 of supplemental material. The parameters and FLOPs of conv-stem are slightly larger than patchify-stem. For example, the parameters of DeiT-Small increase from 22M to 23M, but the increase is very small as the kernel size in last linear projection layer decreases from 16*16 in patchify-stem to 8*8 in conv-stem. The reason why we adopt the VOLO conv-stem rather than that in (Xiao et al. 2021) is that we want to keep the layers of encoders the same as in ViT, but not to remove one encoder layer as in (Xiao et al. 2021).

ViT_ and ViT_c. To make easy comparisons, the original ViT model using patchify-stem is called ViT_p. To form a ViT model with a conv-stem, we simply replace the patchify-stem with conv-stem, leaving all the other unchanged, and we call this ViT as ViT_c. In the following sections, we theoretically and empirically verify that ViT_c is better than ViT_p in stabilizing training and diversifying the patch tokens, due to the scaled ReLU structure.

Scaled ReLU Structure

In this section, we first introduce the Scaled ReLU structure and then analyze how scaled ReLU stabilizes training and enhances the token diversification respectively.

For any input x, we defined the scaled ReLU structure with scaling parameters α, β, ReLU_{α,β}(·) for shorthand, as follow:

$$ReLU_{α,β}(x) = β \max\{x + α, 0\}.$$  

The scaled ReLU structure can be achieved by combining ReLU with normalization layers, such as Batchnorm or Layernorm that contain trainable scaling parameters, and one can view the Batchnorm + ReLU in the conv-stem as a variant of the scaled ReLU. Intuitively, the ReLU layer may cut out part of input data and make the data focus on a smaller range. It is necessary to scale it up to a similar data range as of its input, which helps stabilize training as well as maintain promising expression power. For simplicity, we will focus on the scaled ReLU in this paper and our analysis could be extended to the case with commonly used normalization layers.

Training stabilization

Let’s assume $X_{i,c} \in \mathbb{R}^{H \times W}$ be the output of channel $c$ in the CNN layer from the last conv-stem block for $i$-th sample, where $H$ and $W$ are height and width. Based on the definition of the Batchnorm, the output $X_{i,c}^{out}$ of the last conv-stem is

$$X_{i,c}^{out} = ReLU\left(\frac{X_{i,c} - \mu_c e}{\sqrt{\sum_{i=1}^{B} ||X_{i,c} - \mu_c e||^2}}\right)$$

$$= ReLU_{\alpha_c,\beta_c, e}\left(\frac{X_{i,c} - \mu_c}{\sqrt{\sum_{i=1}^{B} ||X_{i,c} - \mu_c e||^2}}\right)$$

$$= ReLU_{\alpha_c,\beta_c, e}\left(\tilde{X}_{i,c}\right),$$

where $\tilde{X}_{i,c} = \frac{X_{i,c} - \mu_c e}{\sqrt{\sum_{i=1}^{B} ||X_{i,c} - \mu_c e||^2}}$, $\mu_c$ is the mean of $X_{i,c}$ within a batch and $B$ is the batch size. Next, we concatenate $X_{i,c}^{out}$ over channel as $X_{i}^{out}$ and reshape it to $X_{i}^{in}$...
where $n$ is the token (patch) length and $d$ is the embedding dimension. Finally, we compute $Q_l, K_l, V_l$ as follow:

$$[Q_l, K_l, V_l] = X^	rans_l [W_Q, W_K, W_V] = X^	rans_l W_	rans$$

and start to run the self attention.

To illustrate how the scaled ReLU can stabilize training, we consider a special case where we freeze all parameters except the scaling parameters $\alpha_c, \beta_c$ for $c = 1, 2, ..., C$ in the last batchnorm layer, and $W_Q, W_K$ and $W_V$ in the first transformer block. Note that $Q, K$ and $V$ are computed by the production of $X^m$ and $W_	rans$. In order to maintain the same magnitude of $Q, K$ and $V, W_	rans$ will be closer to 0 if $X^m$ is scaled with larger $\alpha_c$ and $\beta_c$ parameters. In other words, the Scaled ReLU may give the penalty weights corresponding to $W_	rans$ an implicit regularization with respect to its scaling parameters. The result is summarized in the following Theorem 1.

**Theorem 1** Let $\alpha_c, \beta_c$ be the parameters in scaled ReLU structures in the last conv-stem block with $c = 1, 2, ..., C$ and $W_	rans = [W_Q, W_K, W_V]$ be the attention parameters in the first transformer block. If we freeze all other parameters and introduce the $l_2$ weight decay in the optimizer, then the optimization problem is equivalent to the weighted $l_1$ penalized learning on $W_	rans$. Moreover, let $W_	rans,c$ be the parameters associated with channel $c$ and the penalty weights corresponding to $W_	rans,c$ are proportional to $\sqrt{\beta_c^2 + \alpha_c^2}$.

The theorem shows an implicit $l_1$ regularization on attention weights from the scaled ReLU structure. In the modern high-dimensional statistics, it is well known that $l_1$ penalized learning introduces significantly less model bias (e.g., exponentially better dimensionality efficiency shown in [Loh and Wainwright, 2015]). Moreover, the regularization strength that is on the order of $O(\sqrt{\alpha_c^2 + \beta_c^2})$ differs from channel to channel and changes over time adaptively. For the channel with larger magnitude in $\alpha_c$ and/or $\beta_c$, the scaled token has higher divergence. In order to make the training processing more stable, the updates for the corresponding parameters in $W_	rans$ need also be more careful (using larger penalties). It distinguishes the scaled ReLU structure from directly using $l_1$ weights decay in the optimizer directly.

**Proof of Theorem 1** We denote the loss function as follow:

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} KL \left( f \left( \{ ReLU_{\alpha_c, \beta_c}(\tilde{X}_{i,c}) \}, W \trans \right), y_i \right)$$

$$+ \lambda \left( \sum_{c=1}^{C} (\alpha_c^2 + \beta_c^2) + \| W \trans \|_F^2 \right),$$

where $KL(\cdot)$ is the KL-divergence, $y_i$ is the label for $i$-th sample, $f(\cdot)$ denotes prediction function, $\lambda$ is a positive constant for $l_2$ weight decays and $\{ ReLU_{\alpha_c, \beta_c}(\tilde{X}_{i,c}) \}$ is the set of $ReLU_{\alpha_c, \beta_c}(\tilde{X}_{i,c})$ over all channels. Without loss of generality, we can find a function $g$ to rewrite $f$ function as:

$$f \left( \{ ReLU_{\alpha_c, \beta_c}(\tilde{X}_{i,c}) \}, W \trans \right)$$

$$= g \left( \{ ReLU_{\alpha_c, \beta_c}(\tilde{X}_{i,c}), W \trans,c \} \right),$$

where we rearrange $W \trans,c$ to match the dimensions of the conv-stem (i.e., $C \times X \times Y$ instead of $n \times d$).

Next, we can re-scale the parameters with $\eta_c > 0$ as follow:

$$\tilde{\beta}_c = \eta_c \beta_c, \tilde{\alpha}_c = \eta_c \alpha_c, W \trans,c = \eta_c^{-1} W \trans,c,$$

and it implies

$$g \left( \{ ReLU_{\tilde{\alpha}_c, \tilde{\beta}_c}(\tilde{X}_{i,c}), W \trans,c \} \right)$$

Moreover, using the fact that $(a^2 + b^2) + c^2 \geq 2|a|\sqrt{a^2 + b^2}$ one can verify

$$\sum_{c=1}^{C} (\tilde{\alpha}_c^2 + \tilde{\beta}_c^2) + \| W \trans \|_F^2$$

$$= \sum_{c=1}^{C} \tilde{\alpha}_c^2 + \tilde{\beta}_c^2 + \| W \trans,c \|^2$$

$$\geq 2 \sum_{c=1}^{C} \eta_c^{-1} W \trans,c \|_1 \sqrt{\frac{\eta_c^2 \alpha^2_c + \eta_c^2 \beta^2_c}{ΗW}} \quad (2)$$

$$= \frac{2}{\sqrt{ΗW}} \sum_{c=1}^{C} \| W \trans,c \|_1 \sqrt{\alpha^2_c + \beta^2_c}, \quad (3)$$

where the equality $(2)$ holds when

$$\eta_c = \sqrt{\| W \trans,c \|_1 \frac{1}{\alpha^2_c + \beta^2_c}}$$

Therefore the right hand-size of $(3)$ becomes the $l_1$ penalties over the $W \trans,c$ with weights $\sqrt{\alpha^2_c + \beta^2_c}$, i.e., $W_Q, W_K$ and $W_V$ are $l_1$ penalized over the input channels with different strength.

**Remark 1.** The analysis of Theorem 1 is also capable of combining the ReLU + Layernorm or Batchnorm + ReLU + MLP structures. In some types of transformer models, the tokens will first go through Layernorm or be projected via MLP before entering the self-attention. Via the similar analysis, we can also show the adaptive implicit $l_1$ regularization in these two settings.

**Tokens Diversification**

Next, we demonstrate the scaled ReLU’s token diversification ability by cosine similarity. Following [Gong et al., 2021] the cosine similarity metric is defined as:

$$\text{CosSim}(B) = \frac{1}{n(n-1)} \sum_{i \neq j} \frac{B_i^T B_j^T}{\|B_i\| \|B_j\|}, \quad (5)$$
where $B_i$ represents the $i$-th row of matrix $B$ and $\| \cdot \|$ denotes the $l_2$ norm. Note that if we can ensure $\|B_i\| > \min_d$ for $i = 1, 2, \ldots, n$, the $\text{CosSim}(B)$ will in turn be upper bounded by
\[
\text{CosSim}(B) \leq \frac{1}{n(n-1)b_{\min}^2} \sum_{i \neq j} B_i B_j^T
\]
\[
= \frac{1}{n(n-1)b_{\min}^2} \left[ e^T B B^T e - \sum_i \|B_i\|^2 \right]
\]
\[
\leq \frac{1}{n-1} \left( \frac{\|B\|^2_{\text{op}}}{b_{\min}^2} - 1 \right), \tag{6}
\]
where $\| \cdot \|_{\text{op}}$ denotes matrix operator norm. Based on (6), as long as $\|B\|_{\text{op}}$ and $b_{\min}$ change at the same order, the cosine similarity may decrease. In the following Theorem 2 we analyze the order of $\|B\|_{\text{op}}$ and $\min_i \|B_i\|$.

**Theorem 2** Let $\mathcal{D}$ be a zero mean probability distribution and matrix $A \in \mathbb{R}^{n \times d}$ be a matrix filled whose elements are drawn independently from $\mathcal{D}$ and $\|A\|_{\text{op}} \leq R\sqrt{nd}$ with $R > 0$. Furthermore, we denote $B = \text{ReLU}_{\alpha, \beta}(A)$, $\mu_B = \mathbb{E}[B_{ij}]$, and $\sigma_B^2 = \text{Var}[B_{ij}]$ for all $i, j$. Then for $\delta > 0$ and $\gamma \in (0, c_0)$, with probability $1 - \delta - 2 \exp(-c_0^2 \gamma^2 d + \log n)$, we have
\[
\|B\|_{\text{op}} \leq O \left( \mu \log \left( \frac{1}{\gamma} \right) + \sigma \sqrt{\log \left( \frac{1}{\gamma} \right)} \right)
\]
and
\[
\min_i \|B_i\|_2 \geq O \left( \sqrt{\mu^2 + (1 - \gamma)\sigma^2} \right),
\]
where $c, c_0$ are positive constants, $O(\cdot)$ suppresses the dependence in $n, d$ and $R$.

The above result shows that the operator norm and $l_2$ norm for each row of the token matrix after scaled ReLU is proportional to its element-wise mean and standard deviation. Given the identity transformation (i.e., $B = A$) is a special case of the scaled ReLU, matrix $A$ (token matrix before scaled ReLU) enjoys the similar properties. As the ReLU truncates the negative parts of its input, one has $\mu_B \geq \mu_A$. If we could maintain the same variance level in $B$ and $A$, both $\min_i \|B_i\|_2$ and $\|B\|_{\text{op}}$ change at order of $O(\mu + \sigma)$ and according to inequality (6), the cosine similarity becomes smaller from $A$ to $B$.

**Proof of Theorem 2**

**Upper Bound for $\|B\|_{\text{op}}$.** Denote $E \in \mathbb{R}^{n \times d}$ as the matrix filled with $1$ and $X = B - \mu E$. We have $\mathbb{E}[X] = 0$, $\|X\|_{\text{op}} \leq (\beta R + \beta \alpha + \mu)\sqrt{nd}$ almost surely. Via the matrix Bernstein inequality (e.g., Theorem 1.6 in Tropp [2012]),
\[
\Pr \left( \|X\|_{\text{op}} \geq t \right) \leq (n + d) \exp \left( \frac{-t^2}{\sigma_{\text{max}}^2 + R_{\text{max}} t / 3} \right), \tag{7}
\]
where
\[
\sigma_{\text{max}}^2 = \max \{ \|\mathbb{E}[X X^T]\|_{\text{op}}, \|\mathbb{E}[X^T X]\|_{\text{op}} \}
\]
\[
= \max \{ n\alpha^2, \sigma^2 \} \leq (n + d)\sigma^2
\]
\[
R_{\text{max}} \geq \|X\|_{\text{op}} = (\beta R + \beta \alpha + \mu)\sqrt{nd}.
\]

By setting $\delta = (n + d)\exp \left( \frac{-t^2/2}{\sigma_{\text{max}}^2 + R_{\text{max}} t / 3} \right)$, we can represent $t$ by $\delta$ as:
\[
t = \frac{1}{3} R_{\text{max}} \log \left( \frac{n + d}{\delta} \right)
\]
\[
+ \frac{1}{9} R_{\text{max}}^2 \log^2 \left( \frac{n + d}{\delta} \right) + 2\sigma_{\text{max}}^2 \log \left( \frac{n + d}{\delta} \right)
\]
\[
\leq \frac{2}{3} R_{\text{max}} \log \left( \frac{n + d}{\delta} \right) + 2\sigma_{\text{max}}^2 \log \left( \frac{n + d}{\delta} \right),
\]
where last inequality uses the fact that $\sqrt{a + b} \leq \sqrt{|a|} + \sqrt{|b|}$.

Then inequality (7) implies the following result holds with probability $1 - \delta$:
\[
\|X\|_{\text{op}} \leq \frac{2}{3} R_{\text{max}} \log \left( \frac{n + d}{\delta} \right) + 2\sigma_{\text{max}}^2 \log \left( \frac{n + d}{\delta} \right).
\]

Next, combine (8) with the facts $\|B\|_{\text{op}} - \|\mu E\|_{\text{op}} \leq \|X\|_{\text{op}}$ and $\|\mu E\|_{\text{op}} = \mu \sqrt{nd}$, one has
\[
\|B\|_{\text{op}} \leq O \left( \mu \log \left( \frac{1}{\delta} \right) + \sigma \sqrt{\log \left( \frac{1}{\delta} \right)} \right),
\]
where we ignore the dependence in $n, d$ and $R$.

**Lower Bound for $\|B_i\|$.** Next, we derive the bound for $\|B_i\|$. Since $\|A\|_{\text{op}}$ is upper bounded, there exists a constant $c_0$ such that $B_{ij}^2 - \mu^2 - \sigma^2$ being centered $c_0 \sigma^2$ sub-exponential random variable. Then we are able to apply the Corollary 5.17 in Vershynin [2010] there exists $c > 0$, for $\eta > 0$:
\[
\mathbb{P} \left( \sum_{j \neq i} B_{ij}^2 - d(\mu^2 + \sigma^2) \geq \eta d \right)
\]
\[
\leq 2 \exp \left( -c \min \left\{ \frac{\eta^2}{\sigma_{\text{max}}^2 + \eta c_0 \sigma^2}, \frac{\eta}{c_0 \sigma^2} \right\} \frac{d}{\delta} \right).
\]

We then set $\eta = \gamma \sigma^2$ for some $\gamma \in (0, c_0)$ such that $\mu^2 + (1 - \gamma)\sigma^2 > 0$. Combining $\|B_i\|^2 = \sum_{j \neq i} B_{ij}^2$ with above inequality, we have
\[
\mathbb{P} \left( \|B_i\| \geq \sqrt{d(\mu^2 + (1 - \gamma)\sigma^2)} \right) \leq 2 \exp \left( -c\gamma^2 c_0^2 d \right).
\]

Therefore via union bound, we have
\[
\min_i \|B_i\| \geq \sqrt{d(\mu^2 + (1 - \gamma)\sigma^2)} = O(\sqrt{\mu^2 + (1 - \gamma)\sigma^2})
\]
holds with probability $1 - 2 \exp(-c\gamma^2 c_0^2 d + \log n)$. \hfill $\Box$

**Experiments**

In this section, we conduct extensive experiments to verify the effects of conv-stem and scaled ReLU. The ImageNet-1k [Russakovsky et al. 2015] is adopted for standard training and validation. It contains 1.3 million images in the training set and 50K images in the validation set, covering 1000 object classes. The images are cropped to $224 \times 224$. 
Table 1: The effects of conv-stem using different learning rate (lr), optimizer, warmup epoch (wm-epoch).

| model      | lr   | optimizer | wm-epoch | Top-1 acc |
|------------|------|-----------|----------|-----------|
| ViT_p      | 5e-4 | AdamW     | 5        | 79.8      |
|            | 1e-3 | AdamW     | 5        | crash     |
| DeiT-Small | 1e-3 | AdamW     | 20       | 80.0      |
|            | 5e-4 | SAM       | 5        | 79.9      |
|            | 1e-3 | SAM       | 5        | 79.6      |
|            | 1e-4 | SAM       | 5        | 77.8      |
| ViT_c      | 5e-4 | AdamW     | 5        | 81.6      |
|            | 1e-3 | AdamW     | 5        | 81.9      |
| DeiT-Small | 1e-3 | AdamW     | 20       | 81.7      |
|            | 5e-4 | SAM       | 5        | 81.5      |
|            | 1e-3 | SAM       | 5        | 81.7      |
|            | 1e-4 | SAM       | 5        | 79.1      |

The effects of conv-stem

We take DeiT-Small (Touvron et al. 2020) as our baseline, and replace the patchify-stem with conv-stem. The batch-size is 1024 for 8 GPUs, and the results are as shown in Table 1. From the Table we can see that conv-stem based model is capable with more volatile training environment: with patchify-stem, ViT_p can not support larger learning rate (1e-3) using AdamW optimizer but only works by using SAM optimizer, which reflects ViT_p is sensitive to learning rate and optimizer. By adding the conv-stem, ViT_c can support larger learning rate using both AdamW and SAM optimizers. Interestingly, ViT_c achieves 81.9 top-1 accuracy using lr=1e-3 and AdamW optimizer, which is 2.1 point higher than baseline. With conv-stem, SAM is no longer more powerful than AdamW, which is a different conclusion as in (Chen, Hsieh, and Gong 2021). After adding conv-stem, it still needs warmup, but 5 epochs are enough and longer warmup training does not bring any benefit.

The effects of scaled ReLU in conv-stem

We adopt three vision transformer architectures, including both supervised and self-supervised methods, to evaluate the value of scaled ReLU for training ViTs, namely, DeiT (Touvron et al. 2020), DINO (Caron et al. 2021) and VOLO (Yuan et al. 2021). For DeiT and VOLO, we follow the official implementation and training settings, only modifying the parameters listed in the head of Table 1 for DINO, we follow the training settings for 100 epoch and show the linear evaluation results as top-1 accuracy. The results are shown in Table 1. From the Table we can see that scaled ReLU (BN+ReLU) plays a very important role for both stable training and boosting performance. Specifically, without ReLU the training will be crashed under 5 warmup epoch in most cases, for both AdamW and SAM optimizers; increasing warmup epoch will increase the stabilization of training with slightly better results; with scaled ReLU, it can boost the final performance largely in stable training mode. The full conv-stem boosts the performance of DeiT-Small largely, 2.1 percent compared with the baseline, but by removing ReLU or scaled ReLU the performance will decrease largely; the same trend holds for both DINO and VOLO. For the patchify-stem, after adding ReLU or scaled ReLU it can stabilize the training by supporting a large learning rate. In addition, scaled ReLU has faster convergence speed. For DeiT-Small, the top-1 accuracy is 18.1 vs 10.6 at 5 epoch, 53.6 vs 46.8 at 20 epoch, 63.8 vs 60.9 at 50 epoch, for conv-stem and patchify-stem, respectively.

Scaled ReLU diversifies tokens

To analyze the property of scaled ReLU diversifying tokens, we adopt the quantitative metric layer-wise cosine similarity between tokens as defined in formula 5.

We regard the conv-stem as one layer and position embedding as another layer in the ViT-stem, thus the total layers of ViT_c is 14 (plus 12 transformer encoder layers). The layer-wise cosine similarity of tokens are shown in Figure 1. From the Figure we can see that position embedding can largely diversify the tokens due to its specific position encoding for each token. Compared with baseline (1Proj) (Touvron et al. 2020), the full conv-stem (3Conv+3BN+3ReLU+1Proj) can significantly diversify the tokens at the lower layers to learn better feature representation, and converge better at higher layers for task-specific feature learning. Interestingly, 3Conv+3ReLU+1Proj and 3Conv+1Proj+warmup20 have the similar trend which reflects that ReLU can stabilize the training as longer warmup epochs.

Figure 1: Layer-wise cosine similarity of tokens for DeiT-Small.

The effects of stride in conv-stem

According to the work (Xiao et al. 2021), the stride in the conv-stem matters for the final performance. We also investigate this problem in the context of VOLO conv-stem for DeiT-Small. We keep the kernel size unchanged, and only adjust the stride and its corresponding padding. The default warmup epoch is 5 unless otherwise noted. The results are shown in Table 1. From this Table we can see that the average stride (2,2,2,2) is not better than (2,1,1,8), and it can not stabilize the training either.

Transfer Learning: Object ReID

In this section, we transfer the DINO-S/16 (100 epoch) on ImageNet-1k to object ReID to further demonstrate the effects of conv-stem. We fine-tune the DINO-S/16 shown in...
DeiT-Small

| model             | lr     | optimizer | wm-epoch | components in conv-stem                           | stride | Top-1 acc |
|-------------------|--------|-----------|----------|---------------------------------------------------|--------|-----------|
|                  | 1e-3   | AdamW     | 5        | 3Conv+3BN+3ReLU+1Proj (2,1,1,8)                    |        | 81.9      |
|                  | 1e-3   | AdamW     | 5        | 3Conv+3BN+1Proj (2,1,1,8)                         |        | crash     |
|                  | 1e-3   | AdamW     | 5        | 3Conv+3ReLU+1Proj (2,1,1,8)                       |        | 81.5      |
|                  | 1e-3   | AdamW     | 5        | 3Conv+1Proj (2,1,1,8)                             |        | crash     |
|                  | 1e-3   | AdamW     | 20       | 3Conv+1Proj (2,1,1,8)                             |        | 80.0      |
|                  | 1e-3   | AdamW     | 5        | 3Conv+1Proj+1ReLU (2,1,1,8)                       | (16)   | 79.9      |
|                  | 1e-3   | AdamW     | 5        | 1Proj+1BN+1ReLU (2,1,1,8)                         | (16)   | 79.8      |
|                  | 1e-3   | AdamW     | 5        | 1Proj+1ReLU (2,1,1,8)                             | (16)   | 79.5      |
|                  | 1e-3   | AdamW     | 5        | 1Proj (baseline) (16)                             |        | crash     |
|                  | 5e-4   | AdamW     | 5        | 1Proj (baseline) (16)                             |        | 79.8      |
|                  | 1e-3   | SAM       | 5        | 3Conv+3BN+3ReLU+1Proj (2,1,1,8)                    | (16)   | 81.7      |
|                  | 1e-3   | SAM       | 5        | 3Conv+3BN+1Proj (2,1,1,8)                         | (16)   | 80.2      |
|                  | 1e-3   | SAM       | 5        | 3Conv+3ReLU+1Proj (2,1,1,8)                       | (16)   | 80.6      |
|                  | 1e-3   | SAM       | 5        | 3Conv+1Proj (2,1,1,8)                             | (16)   | crash     |
|                  | 1e-3   | SAM       | 20       | 3Conv+1Proj (2,1,1,8)                             | (16)   | 80.4      |
|                  | 1e-3   | SAM       | 5        | 3Conv+1Proj+1ReLU (2,1,1,8)                       | (16)   | 80.3      |
|                  | 5e-4   | AdamW     | 10       | 3Conv+3BN+3ReLU+1Proj (2,1,1,8)                    | (16)   | 76.0      |
|                  | 5e-4   | AdamW     | 10       | 3Conv+3BN+1Proj (2,1,1,8)                         | (16)   | 73.4      |
|                  | 5e-4   | AdamW     | 10       | 3Conv+3ReLU+1Proj (2,1,1,8)                       | (16)   | 74.8      |
|                  | 5e-4   | AdamW     | 10       | 3Conv+1Proj (2,1,1,8)                             | (16)   | 74.1      |
|                  | 5e-4   | AdamW     | 10       | 1Proj+1ReLU (2,1,1,8)                             | (16)   | 73.6      |
|                  | 5e-4   | AdamW     | 10       | 1Proj+1BN+1ReLU (2,1,1,8)                         | (16)   | 73.3      |
|                  | 5e-4   | AdamW     | 10       | 1Proj (baseline) (8)                              |        | 73.6      |
|                  | 1.6e-3 | AdamW     | 20       | 3Conv+3BN+3ReLU+1Proj (2,1,1,4)                    | (16)   | 84.1      |
|                  | 1.6e-3 | AdamW     | 20       | 3Conv+3BN+1Proj (2,1,1,4)                         | (16)   | 83.6      |
|                  | 1.6e-3 | AdamW     | 20       | 3Conv+3ReLU+1Proj (2,1,1,4)                       | (16)   | 84.0      |
|                  | 1.6e-3 | AdamW     | 20       | 3Conv+1Proj (2,1,1,4)                             | (16)   | crash     |
|                  | 1.6e-3 | AdamW     | 20       | 1Proj (8)                                         |        | 83.4      |
|                  | 1.6e-3 | AdamW     | 20       | 1Proj+1ReLU (8)                                   |        | 83.4      |
|                  | 1.6e-3 | AdamW     | 20       | 1Proj+1BN+1ReLU (8)                               |        | 83.5      |

Table 2: The effects of scaled ReLU under different settings using three methods.

| components in conv-stem | stride | top-1 acc |
|-------------------------|--------|-----------|
| 3Conv+3BN+3ReLU+1Proj   | (2,1,1,8) | 81.9      |
| 3Conv+3BN+1Proj         | (2,2,2,2) | 81.0      |
| 3Conv+1Proj             | (2,1,1,8) | crash     |
| 3Conv+1Proj             | (2,2,2,2) | crash     |
| 3Conv+1Proj+1ReLU       | (2,1,1,8) | 80.0      |
| 3Conv+1Proj+1ReLU       | (2,2,2,2) | 79.7      |
| 3Conv+1Proj+1ReLU       | (2,1,1,8) | 79.9      |
| 3Conv+1Proj+1ReLU       | (2,2,2,2) | 79.9      |

Table 3: The effects of stride in conv-stem for DeiT-Small.

Table 2 on Market1501 (Zheng et al. 2015) and MSMT17 (Wei et al. 2018) datasets. We follow the baseline (He et al. 2021) and follow the standard evaluation protocol to report the Mean Average Precision (mAP) and Rank-1 accuracies. All models are trained with the baseline learning rate (1.6e-3) and a larger learning rate (5e-2). The results are shown in Table 2. From the Table we can see that the full conv-stem not only achieves the best performance but also supports both the large learning rate and small learning rate training. Without ReLU or BN+ReLU, in most cases, the finetuning with a large learning rate will crash. Interestingly, the fine-tuning with DINO is sensitive to the learning rate, a smaller learning rate will achieve better performance.

**Scaled ReLU/GELU in Transformer Encoder**

In transformer encoder layer, the feed-forward layers (ffn) adopt LayerNorm+GELU block, and in this section, we investigate this design using DeiT-Small and VOLO-d1-224, using the training parameters for the best performance in Table 2. The motivation to investigate ReLU and GELU is to show whether GELU is better than ReLU for conv-stem design, as GELU achieves better results than ReLU for transformer encoder. We first remove the LayerNorm layer in ffn, the training directly crashes in the first few epochs. And then, we replace the GELU with RELU, the performance drops largely, which reflects that GELU is better than ReLU for ffn. Next, we replace ReLU with GELU in conv-stem, the performance drops a little bit, demonstrating that ReLU is better than GELU for conv-stem. Lastly, we rewrite the MLP implementation in ffn by replacing the fc+act block with Conv1D+BN1D+GELU (Conv1D equals to fc, and the full implementation is shown in supplemental material of Algorithm 2), and the performance drops, especially for VOLO. It might confirm the conclusion in NFNet (Brock et al. 2021).
that batch normalization constrains the extreme performance, making the network sub-optimal. All the results are shown in Table 5.

| components in conv-stem | lr | Market1501 mAP R-1 | MSMT17 mAP R-1 |
|-------------------------|----|---------------------|----------------|
| 3Conv+3BN+3ReLU+1Proj   | 1.6-3 | 84.3 93.5 | 56.3 78.7 |
| 3Conv+3BN+1Proj         | 3Conv+3BN+1Proj | 83.6 92.9 | 55.1 77.8 |
| 1Proj+ReLU              | 1Proj+ReLU | 84.3 93.1 | 53.6 75.5 |
| 1Proj+BN+ReLU           | 1Proj+BN+ReLU | 83.6 92.8 | 55.7 77.5 |
| 1Proj (baseline)        | 1Proj (baseline) | 84.1 93.1 | 54.9 76.8 |

Table 4: The comparisons with different components in conv-stem based on DINO for finetuning ReID tasks.

Self-supervised + supervised training

To further investigate the training of ViTs, we adopt the DINO self-supervised pretrained ViT-Small model (Caron et al. 2021) on ImageNet-1k and use it to initialize the ViT-Small model to finetune on ImageNet-1k using full labels. The results are shown in Table 5. From this Table we can see that using a self-supervised pretrained model for initialization, ViT_p achieve 81.6 top-1 accuracy using SAM optimizer, which is 1.8 percent point higher than baseline. However, according to the analysis in (Newell and Deng 2020), with large labelled training data like Imagenet-1k dataset, the two stage training strategy will not contribute much (below 0.5 percent point). By adding conv-stem, the peak performance of ViT_c can reach 81.9 which is higher than two stage training, which reflects that previous ViTs models are far from being well trained.

| model          | design          | Top-1 acc | lr   | optimizer | data-size | Top-1 acc |
|----------------|-----------------|-----------|------|-----------|-----------|-----------|
| DeiT-Small_p   | LayerNorm removed in ffn | crash    | 1e-4 | AdamW     | 5         | 81.2      |
| TST            | GELU→ReLU in ffn | 80.5 (1.6) | 1e-3 | AdamW     | 5         | 81.3      |
|                | ReLU→GELU in conv-stem | 81.7 (0.2) | 1e-4 | SAM       | 5         | 81.6      |
|                | MLP→Conv1D+BN+GELU | 81.7 (0.2) | 5e-4 | SAM       | 5         | 81.1      |
|                | MLP→Conv1D+GELU | 82.0 (0.1) | 1e-3 | SAM       | 5         | 80.1      |
| DeiT-Small_c   | LayerNorm removed in ffn | crash    | 1e-3 | AdamW     | 5         | 81.9      |
| OST            | GELU→ReLU in ffn | 83.5 (0.6) | 1e-3 | SAM       | 5         | 81.7      |
|                | ReLU→GELU in conv-stem | 84.0 (0.1) | 5e-4 | SAM       | 5         | 80.9      |
|                | MLP→Conv1D+BN+GELU | 83.2 (0.9) | 1e-4 | SAM       | 90%      | 81.4      |
|                | MLP→Conv1D+GELU | 84.0 (0.1) | 1e-4 | SAM       | 100%     | 81.6      |

Table 5: The comparisons among different designs using scaled ReLU/GELU.

Scaled Dataset Training

In order to verify that the previous ViTs are not well trained, we adopt the DINO pretrained ViT-Small model (Caron et al. 2021) on ImageNet-1k to initialize the ViT-Small model, and finetune on ImageNet-1k using portion of full labels, containing 1000 classes. We adopt the original patchify-stem and SAM optimizer for this investigation. The results are shown in Table 7. It can be seen that even using self-supervised pretrained model for initialization, using only 10% of ImageNet-1k data for training, it only achieves 67.8% accuracy, much worse than the linear classification accuracy using full data (77.0%) (Caron et al. 2021). With the data-size increasing, the performance improves obviously, and we do not see any saturation in the data-size side. This performance partly demonstrates that ViT is powerful in fitting data and current ViT models trained on ImageNet-1k is not trained enough for its extreme performance.

| model          | lr   | optimizer | data-size | Top-1 acc |
|----------------|------|-----------|-----------|-----------|
| DeiT-Small_p   | 1e-4 | SAM       | 10%       | 67.8      |
| TST            | 1e-4 | SAM       | 20%       | 75.5      |
|                | 1e-4 | SAM       | 30%       | 76.0      |
|                | 1e-4 | SAM       | 40%       | 77.6      |
|                | 1e-4 | SAM       | 50%       | 79.0      |
|                | 1e-4 | SAM       | 60%       | 79.8      |
|                | 1e-4 | SAM       | 70%       | 80.4      |
|                | 1e-4 | SAM       | 80%       | 80.9      |
|                | 1e-4 | SAM       | 90%       | 81.4      |
|                | 1e-4 | SAM       | 100%      | 81.6      |

Table 7: The comparisons among different portion of ImageNet-1k for two-stage training (TST, self-supervised + supervised training) training.

Conclusion

In this paper, we investigate the training of ViTs in the context of conv-stem. We theoretically and empirically verify that the scaled ReLU in the conv-stem matters for robust ViTs training. It can stabilize the training and improve the token diversity for better feature learning. Using conv-stem, ViTs enjoy a peak performance boost and are insensitive to the training parameters. Extensive experiments unveil the merits of conv-stem and demonstrate that previous ViTs are not well trained even if they obtain better results in many cases compared with CNNs.
References

Brock, A.; De, S.; Smith, S. L.; and Simonyan, K. 2021. High-performance large-scale image recognition without normalization. arXiv preprint arXiv:2102.06171.

Caron, M.; Tzirakis, C.; Misra, I.; Jégou, H.; Mairal, J.; Bojanowski, P.; and Joulin, A. 2021. Emerging Properties in Self-Supervised Vision Transformers. arXiv preprint arXiv:2104.14294.

Chen, B.; Li, P.; Li, B.; Li, C.; Bai, L.; Lin, C.; Sun, M.; Yan, J.; and Ouyang, W. 2021a. PViT: Better Vision Transformer via Token Pooling and Attention Sharing. arXiv preprint arXiv:2108.03428.

Chen, C.-F.; Fan, Q.; and Panda, R. 2021. Crossvit: Cross-attention multi-scale vision transformer for image classification. arXiv preprint arXiv:2103.14899.

Chen, X.; Hsieh, C.-J.; and Gong, B. 2021. When Vision Transformers Outperform ResNets without Pretraining or Strong Data Augmentations. arXiv preprint arXiv:2106.09681.

Chen, Z.; Xie, L.; Niu, J.; Liu, X.; Wei, L.; and Tian, Q. 2021b. Visformer: The Vision-friendly Transformer. arXiv preprint arXiv:2104.12533.

d’Ascoli, S.; Touvron, H.; Leavitt, M.; Morcos, A.; Biroli, G.; and Sagun, L. 2021. ConViT: Improving Vision Transformers with Soft Convolutional Inductive Biases. arXiv preprint arXiv:2103.10697.

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.

El-Nouby, A.; Touvron, H.; Caron, M.; Bojanowski, P.; Douze, M.; Joulin, A.; Laptev, I.; Neverova, N.; Synnaeve, G.; Verbeek, J.; et al. 2021. XCiT: Cross-Covariance Image Transformers. arXiv preprint arXiv:2106.09681.

Ergen, T.; Sahiner, A.; Ozturkler, B.; Pauly, J.; Mardani, M.; and Filanci, M. 2021. Demystifying Batch Normalization in ReLU Networks: Equivalent Convex Optimization Models and Implicit Regularization. arXiv preprint arXiv:2103.01499.

Fan, H.; Xiong, B.; Mangalam, K.; Li, Y.; Yan, Z.; Malik, J.; and Feichtenhofer, C. 2021. Multiscale vision transformers. arXiv preprint arXiv:2104.11227.

Fang, J.; Xie, L.; Wang, X.; Zhang, X.; Liu, W.; and Tian, Q. 2021. MSG-Transformer: Exchanging Local Spatial Information by Manipulating Messenger Tokens. arXiv preprint arXiv:2105.15168.

Foret, P.; Kleiner, A.; Mobahi, H.; and Neyshabur, B. 2020. Sharpness-aware Minimization for Efficiently Improving Generalization. In International Conference on Learning Representations.

Gao, P.; Lu, J.; Li, H.; Mottaghi, R.; and Kemhiavi, A. 2021. Container: Context Aggregation Network. arXiv preprint arXiv:2106.01401.

Gong, C.; Wang, D.; Li, M.; Chandra, V.; and Liu, Q. 2021. Improve Vision Transformers Training by Suppressing Over-smoothing. arXiv preprint arXiv:2104.12753.

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. arXiv preprint arXiv:2104.01136.

Guo, J.; Han, K.; Wu, H.; Xu, C.; Tang, Y.; Xu, C.; and Wang, Y. 2021. CMT: Convolutional Neural Networks Meet Vision Transformers. arXiv preprint arXiv:2107.06263.

Han, K.; Xiao, A.; Wu, E.; Guo, J.; Xu, C.; and Wang, Y. 2021. Transformer in transformer. arXiv preprint arXiv:2103.00112.

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.

He, S.; Luo, H.; Wang, P.; Wang, F.; Li, H.; and Jiang, W. 2021. Transreid: Transformer-based object re-identification. arXiv preprint arXiv:2102.04378.

Heo, B.; Yun, S.; Han, D.; Chun, S.; Choe, J.; and Oh, S. J. 2021. Rethinking spatial dimensions of vision transformers. arXiv preprint arXiv:2103.16302.

Howard, A. G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; and Adam, H. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

Hu, J.; Shen, L.; and Sun, G. 2018. Squeeze-and-excitation networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, 7132–7141.

Huang, Z.; Ben, Y.; Luo, G.; Cheng, P.; Yu, G.; and Fu, B. 2021. Shuffle Transformer: Rethinking Spatial Shuffle for Vision Transformer. arXiv preprint arXiv:2106.03650.

Jiang, Z.; Hou, Q.; Yuan, L.; Zhou, D.; Jin, X.; Wang, A.; and Feng, J. 2021. Token Encoding: Trading a 85.5% Top-1 Accuracy Vision Transformer with 56M Parameters on ImageNet. arXiv preprint arXiv:2104.10858.

Jonnalagadda, A.; Wang, W.; and Eckstein, M. P. 2021. FoveaTet: Foveated Transformer for Image Classification. arXiv preprint arXiv:2105.14173.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25: 1097–1105.

Li, D.; Hu, J.; Wang, C.; Li, X.; She, Q.; Zhu, L.; Zhang, T.; and Chen, Q. 2021a. Involution: Inverting the Inherence of Convolution for Visual Recognition. arXiv preprint arXiv:2103.06255.

Li, Y.; Zhang, K.; Cao, J.; Timofte, R.; and Van Gool, L. 2021b. LocalViT: Bringing Locality to Vision Transformers. arXiv preprint arXiv:2104.05707.

Liu, Z.; Lin, Y.; Cao, Y.; Hu, H.; Wei, Y.; Zhang, Z.; Lin, S.; and Guo, B. 2021. Swin transformer: Hierarchical vision transformer using shifted windows. arXiv preprint arXiv:2103.14030.

Loh, P.-L.; and Wainwright, M. J. 2015. Regularized M-estimators using shifted windows. The Journal of Machine Learning Research, 16(1): 559–616.

Newell, A.; and Deng, J. 2020. How useful is self-supervised pretraining for visual tasks? In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 7345–7354.

Neyshabur, B.; Tomioka, R.; and Srebro, N. 2014. In search of the real inductive bias: On the role of implicit regularization in deep learning. arXiv preprint arXiv:1412.6014.

Raff, R.; Bochkovskiy, A.; and Koltun, V. 2021. Vision transformers for dense prediction. arXiv preprint arXiv:2103.13413.

Rao, Y.; Zhao, W.; Liu, B.; Lu, J.; Zhou, J.; and Hsieh, C.-J. 2021. DynamicViT: Efficient Vision Transformers with Dynamic Token Sparsification. arXiv preprint arXiv:2106.02034.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al. 2015. Imagenet large scale visual recognition challenge. International journal of computer vision, 115(3): 211–252.

Savarese, P.; Evron, I.; Soudry, D.; and Srebro, N. 2019. How do infinite width bounded norm networks look in function space? In Conference on Learning Theory, 2667–2690. PMLR.
Algorithm 1: Source codes of Patch Embedding for conv-stem.

class PatchEmbed(nn.Module):
    def __init__(self, img_size=224, conv_stem=True, stem_stride=2, patch_size=16, in_chans=3, hid_dim=64, embed_dim=384):
        super().__init__()
        self.conv_stem = conv_stem
        self.conv = nn.Sequential(
            nn.Conv2d(in_chans, hid_dim, kernel_size=7, stride=stem_stride, padding=3, bias=False), #112x112
            nn.BatchNorm2d(hid_dim),
            nn.ReLU(inplace=True),
            nn.Conv2d(hid_dim, hid_dim, kernel_size=3, stride=1, padding=1, bias=False), #112x112
            nn.BatchNorm2d(hid_dim),
            nn.ReLU(inplace=True),
            nn.Conv2d(hid_dim, hid_dim, kernel_size=3, stride=1, padding=1, bias=False), #112x112
            nn.BatchNorm2d(hid_dim),
            nn.ReLU(inplace=True)
        )
        self.proj = nn.Conv2d(hid_dim, embed_dim, kernel_size=patch_size//stem_stride, stride=patch_size//stem_stride)
        self.num_patches = (img_size//patch_size)*(img_size//patch_size)

    def forward(self, x):
        if self.conv_stem:
            x = self.conv(x)
            x = self.proj(x) #B,C,H,W
            x = x.flatten(2).transpose(1,2) #B,N,C
        return x
Algorithm 2: Source codes of two Mlp implementations.

class Mlp(nn.Module): ## original implementation
    def __init__(self, in_features, hidden_features=None, out_features=None, act_layer=nn.GELU, drop=0.):
        super().__init__()
        out_features = out_features or in_features
        hidden_features = hidden_features or in_features
        self.fc1 = nn.Linear(in_features, hidden_features)
        self.act = act_layer()
        self.fc2 = nn.Linear(hidden_features, out_features)
        self.drop = nn.Dropout(drop)

    def forward(self, x):
        x = self.fc1(x)
        x = self.act(x)
        x = self.drop(x)
        x = self.fc2(x)
        x = self.drop(x)
        return x

class Mlp(nn.Module): ## revised implementation using Conv1D+BN1D+GELU
    def __init__(self, in_features, hidden_features, act_layer=nn.GELU, drop=0.):
        super().__init__()
        self.w1 = nn.Sequential(
            nn.Conv1d(in_features, hidden_features, kernel_size=1, stride=1),
            nn.BatchNorm1d(hidden_features),
            nn.GELU(),
        )
        self.w2 = nn.Sequential(
            nn.Conv1d(hidden_features, in_features, kernel_size=1, stride=1),
            nn.BatchNorm1d(in_features),
            nn.GELU(),
        )
        self.drop = nn.Dropout(drop)

    def forward(self, x):
        x = x.permute(0,2,1)
        x = self.w1(x)
        x = self.drop(x)
        x = self.w2(x)
        x = x.permute(0,2,1)
        return x