Dual PatchNorm

Manoj Kumar  
mechcoder@google.com

Mostafa Dehghani  
dehghani@google.com

Neil Houlsby  
neilhoulsby@google.com

Google Research, Brain Team

Reviewed on OpenReview: https://openreview.net/forum?id=jgMqve6Qhw

Abstract

We propose Dual PatchNorm: two Layer Normalization layers (LayerNorms), before and after the patch embedding layer in Vision Transformers. We demonstrate that Dual PatchNorm outperforms the result of exhaustive search for alternative LayerNorm placement strategies in the Transformer block itself. In our experiments on image classification, contrastive learning, semantic segmentation and transfer on downstream classification datasets, incorporating this trivial modification, often leads to improved accuracy over well-tuned vanilla Vision Transformers and never hurts.

1 Introduction

Layer Normalization [Ba et al., 2016] is key to Transformer’s success in achieving both stable training and high performance across a range of tasks. Such normalization is also crucial in Vision Transformers (ViT) [Dosovitskiy et al., 2020; Touvron et al., 2021] which closely follow the standard recipe of the original Transformer model.

Following the “pre-LN” strategy in Baevski & Auli (2019) and Xiong et al. (2020), ViTs place LayerNorms before the self-attention layer and MLP layer in each Transformer block. We explore the following question: Can we improve ViT models with a different LayerNorm ordering? First, across five ViT architectures on ImageNet-1k [Russakovsky et al., 2015], we demonstrate that an exhaustive search of LayerNorm placements between the components of a Transformer block does not improve classification accuracy. This indicates that the pre-LN strategy in ViT is close to optimal. Our observation also applies to other alternate LayerNorm placements: NormFormer (Shleifer et al., 2021) and Sub-LN (Wang et al., 2022), which in isolation, do not improve over strong ViT classification models.

Second, we make an intriguing observation: placing additional LayerNorms before and after the standard ViT-projection layer, which we call Dual PatchNorm (DPN), can improve significantly over well tuned vanilla ViT baselines. Our experiments on image classification across three different datasets with varying number of examples and contrastive learning, demonstrate the efficacy of DPN. Interestingly, our qualitative experiments show that the LayerNorm scale parameters upweight the pixels at the center and corners of each patch.

1 hp, wp = patch_size[0], patch_size[1]
  2 x = einops.rearrange(
  3   x, "b (ht hp) (wt wp) c -> b (ht wt) (hp wp c)", hp=hp, wp=wp)
  4 x = nn.LayerNorm(name="ln0")(x)
  5 x = nn.Dense(output_features, name="dense")(x)
  6 x = nn.LayerNorm(name="ln1")(x)

Dual PatchNorm consists of a 2 line change to the standard ViT-projection layer.
2 Related Work

Kim et al. (2021) add a LayerNorm after the patch-embedding and show that this improves the robustness of ViT against corruptions on small-scale datasets. Xiao et al. (2021) replace the standard Transformer stem with a small number of stacked stride-two $3 \times 3$ convolutions with batch normalizations and show that this improves the sensitivity to optimization hyperparameters and final accuracy. Xu et al. (2019) analyze LayerNorm and show that the derivatives of mean and variance have a greater contribution to final performance as opposed to forward normalization. Beyer et al. (2022a) consider Image-LN and Patch-LN as alternative strategies to efficiently train a single model for different patch sizes. Wang et al. (2022) add extra LayerNorms before the final dense projection in the self-attention block and the non-linearity in the MLP block, with a different initialization strategy. Shleifer et al. (2021) propose extra LayerNorms after the final dense projection in the self-attention block instead with a LayerNorm after the non-linearity in the MLP block. Unlike previous work, we show that LayerNorms before and after the embedding layer provide consistent improvements on classification and contrastive learning tasks. An orthogonal line of work (Liu et al., 2021; d’Ascoli et al., 2021; Wang et al., 2021) involves incorporating convolutional inductive biases to VisionTransformers. Here, we exclusively and extensively study LayerNorm placements of vanilla ViT.

3 Background

3.1 Patch Embedding Layer in Vision Transformer

Vision Transformers (Dosovitskiy et al., 2020) consist of a patch embedding layer (PE) followed by a stack of Transformer blocks. The PE layer first rearranges the image $x \in \mathbb{R}^{H \times W \times 3}$ into a sequence of patches $x_p \in \mathbb{R}^{H_p \times W_p \times 3}$ where $P$ denotes the patch size. It then projects each patch independently with a dense projection to constitute a sequence of “visual tokens” $x_t \in \mathbb{R}^{H \times W_p \times D}$ $P$ controls the trade-off between granularity of the visual tokens and the computational cost in the subsequent Transformer layers.

3.2 Layer Normalization

Given a sequence of $N$ patches $x \in \mathbb{R}^{N \times D}$, LayerNorm as applied in ViTs consist of two operations:

$$x = \frac{x - \mu(x)}{\sigma(x)} \quad (1)$$

$$y = \gamma x + \beta \quad (2)$$

where $\mu(x) \in \mathbb{R}^N, \sigma(x) \in \mathbb{R}^N, \gamma \in \mathbb{R}^D, \beta \in \mathbb{R}^D$.

First, Eq. 1 normalizes each patch $x_i \in \mathbb{R}^D$ of the sequence to have zero mean and unit standard deviation. Then, Eq. 2 applies learnable shifts and scales $\beta$ and $\gamma$ which are shared across all patches.

4 Methods

4.1 Alternate LayerNorm placements:

Following Baevski & Auli (2019) and Xiong et al. (2020), ViTs incorporate LayerNorm before every self-attention and MLP layer, commonly known as the pre-LN strategy. For each of the self-attention and MLP layer, we evaluate 3 strategies: place LayerNorm before (pre-LN), after (post-LN), before and after (pre+post-LN) leading to nine different combinations.

4.2 Dual PatchNorm

Instead of adding LayerNorms to the Transformer block, we also propose to apply LayerNorms in the stem alone, both before and after the patch embedding layer. In particular, we replace
\[ x = \text{PE}(x) \]  
\[ \text{with} \]
\[ x = \text{LN}(\text{PE}(\text{LN}(x))) \]
and keep the rest of the architecture fixed. We call this Dual PatchNorm (DPN).

5 Experiments on ImageNet Classification

5.1 Setup
We adopt the standard formulation of Vision Transformers (Sec. 3.1) which has shown broad applicability across a number of vision tasks. We train ViT architectures (with and without DPN) in a supervised fashion on 3 different datasets with varying number of examples: ImageNet-1k (1M), ImageNet-21k (21M) and JFT (4B) (Zhai et al., 2022a). In our experiments, we apply DPN directly on top of the baseline ViT recipes without additional hyperparameter tuning. We split the ImageNet train set into a train and validation split, and use the validation split to arrive at the final DPN recipe.

**ImageNet 1k:** We train 5 architectures: Ti/16, S/16, S/32, B/16 and B/32 using the AugReg (Steiner et al., 2022) recipe for 93000 steps with a batch size of 4096 and report the accuracy on the official ImageNet validation split as is standard practice. The AugReg recipe provides the optimal mixup regularization (Zhang et al., 2017) and RandAugment (Cubuk et al., 2020) for each ViT backbone. Further, we evaluate a S/16 baseline (S/16+) with additional extensive hyperparameter tuning on ImageNet (Beyer et al., 2022b). Finally, we also apply DPN on top of the base and small DeiT variants (Touvron et al., 2021). Our full set of hyperparameters are available in Appendix C and Appendix D.

**ImageNet 21k:** We adopt a similar setup as in ImageNet 1k. We report ImageNet 25 shot accuracies in two training regimes: 93K and 930K steps.

**JFT:** We evaluate the ImageNet 25 shot accuracies of 3 variants (B/32, B/16 and L/16) on 2 training regimes: (220K and 1.1M steps) with a batch size of 4096. In this setup, we do not use any additional data augmentation or mixup regularization.

On ImageNet-1k, we report the 95% confidence interval across at least 3 independent runs. On ImageNet-21k and JFT, because of expensive training runs, we train each model once and report the mean 25 shot accuracy with 95% confidence interval across 3 random seeds.

5.2 DPN versus alternate LayerNorm placements
Each Transformer block in ViT consists of a self-attention (SA) and MLP layer. Following the pre-LN strategy (Xiong et al., 2020), LN is inserted before both the SA and MLP layers. We first show that the default pre-LN strategy in ViT models is close to optimal by evaluating alternate LN placements on ImageNet-1k. We then contrast this with the performance of NormFormer, Sub-LN and DPN.

For each SA and MLP layer, we evaluate three LN placements: Pre, Post and Pre+Post, that leads to nine total LN placement configurations. Additionally, we evaluate the LayerNorm placements in NormFormer (Shleifer et al., 2021) and Sub LayerNorm (Wang et al., 2022) which add additional LayerNorms within each of the self-attention and MLP layers in the transformer block. Figure 1 shows that none of the placements outperform the default Pre-LN strategy significantly, indicating that the default pre-LN strategy is close to optimal. NormFormer provides some improvements on ViT models with a patch size of 32. DPN on the other hand provides consistent improvements across all 5 architectures.
Figure 1: The plot displays the accuracy gains of different LayerNorm placement strategies over the default pre-LN strategy. Each blue point (Other LN placement) corresponds to a different LN placement in the Transformer block. None of the placements outperform the default Pre-LN strategy on ImageNet-1k (Russakovsky et al., 2015). Applying DPN (black cross) provides consistent improvements across all 5 architectures.

<table>
<thead>
<tr>
<th>Arch Base</th>
<th>DPN</th>
<th>DPN</th>
<th>DPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>S/32</td>
<td>72.1±0.07</td>
<td>74.0±0.09</td>
<td>74.0±0.09</td>
</tr>
<tr>
<td>Ti/16</td>
<td>72.5±0.07</td>
<td>73.9±0.09</td>
<td>73.9±0.09</td>
</tr>
<tr>
<td>B/32</td>
<td>74.8±0.06</td>
<td>76.2±0.07</td>
<td>76.2±0.07</td>
</tr>
<tr>
<td>S/16</td>
<td>78.6±0.32</td>
<td>79.7±0.2</td>
<td>79.7±0.2</td>
</tr>
<tr>
<td>S/16+</td>
<td>79.7±0.09</td>
<td>80.2±0.03</td>
<td>80.2±0.03</td>
</tr>
<tr>
<td>B/16</td>
<td>80.4±0.06</td>
<td>81.1±0.09</td>
<td>81.1±0.09</td>
</tr>
</tbody>
</table>

DeiT

<table>
<thead>
<tr>
<th>Arch Base</th>
<th>DPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>S/16</td>
<td>80.1±0.03</td>
</tr>
<tr>
<td>B/16</td>
<td>81.8±0.03</td>
</tr>
</tbody>
</table>

AugReg + 384×384 Finetune

<table>
<thead>
<tr>
<th>Arch Base</th>
<th>DPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>B/32</td>
<td>79.0±0.00</td>
</tr>
<tr>
<td>B/16</td>
<td>82.2±0.03</td>
</tr>
</tbody>
</table>

Table 1: Left: ImageNet-1k validation accuracies of five ViT architectures with and without dual patch norm after 93000 steps. Right: We train ViT models on ImageNet-21k in two training regimes: 93k and 930k steps with a batch size of 4096. The table shows their ImageNet 25 shot accuracies with and without Dual PatchNorm.

5.3 Comparison to ViT

In Table 1 left, DPN improved the accuracy of B/16, the best ViT model by 0.7 while S/32 obtains the maximum accuracy gain of 1.9. The average gain across all architecture is 1.4. On top of DeiT-S and DeiT-B, DPN provides an improvement of 0.3 and 0.2 respectively. Further, we finetune B/16 and B/32 models with and without DPN on high resolution ImageNet (384×384) for 5000 steps with a batch-size of 512 (See Appendix D for the full hyperparameter setting). Applying DPN improves high-res, finetuned B/16 and B/32 by 0.6 and 1.0 respectively.
DPN improves all architectures trained on ImageNet-21k (Table 1 Right) and JFT (Table 2) on shorter training regimes with average gains of 1.7 and 0.8 respectively. On longer training regimes, DPN improves the accuracy of the best-performing architectures on JFT and ImageNet-21k by 0.5 and 0.4 respectively.

In three cases, Ti/16 and S/32 with ImageNet-21k and B/16 with JFT, DPN matches or leads to marginally worse results than the baseline. Nevertheless, across a large fraction of ViT models, simply employing DPN out-of-the-box on top of well-tuned ViT baselines lead to significant improvements.

5.4 Finetuning on ImageNet with DPN

We finetune four models trained on JFT-4B with two resolutions on ImageNet-1k: (B/32, B/16) × (220K, 1.1M) steps on resolutions 224 × 224 and 384 × 384. On B/32 we observe a consistent improvement across all configurations. With L/16, DPN outperforms the baseline on 3 out of 4 configurations.

<table>
<thead>
<tr>
<th>Arch</th>
<th>Base</th>
<th>DPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>B/32</td>
<td>63.8 ± 0.03</td>
<td>65.2 ± 0.03</td>
</tr>
<tr>
<td>B/16</td>
<td>72.1 ± 0.09</td>
<td>72.4 ± 0.07</td>
</tr>
<tr>
<td>L/16</td>
<td>77.3 ± 0.00</td>
<td>77.9 ± 0.06</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arch</th>
<th>Resolution</th>
<th>Steps</th>
<th>Base</th>
<th>DPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>B/32</td>
<td>224</td>
<td>220K</td>
<td>77.6 ± 0.06</td>
<td>78.3 ± 0.00</td>
</tr>
<tr>
<td>B/32</td>
<td>384</td>
<td>220K</td>
<td>81.3 ± 0.09</td>
<td>81.6 ± 0.00</td>
</tr>
<tr>
<td>B/32</td>
<td>224</td>
<td>1.1M</td>
<td>80.8 ± 0.1</td>
<td>81.3 ± 0.00</td>
</tr>
<tr>
<td>B/32</td>
<td>384</td>
<td>1.1M</td>
<td>83.8 ± 0.03</td>
<td>84.1 ± 0.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arch</th>
<th>Resolution</th>
<th>Steps</th>
<th>Base</th>
<th>DPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>L/16</td>
<td>224</td>
<td>220K</td>
<td>84.9 ± 0.06</td>
<td>85.3 ± 0.03</td>
</tr>
<tr>
<td>L/16</td>
<td>384</td>
<td>220K</td>
<td>86.7 ± 0.03</td>
<td>87.0 ± 0.00</td>
</tr>
<tr>
<td>L/16</td>
<td>224</td>
<td>1.1M</td>
<td>86.7 ± 0.03</td>
<td>87.1 ± 0.00</td>
</tr>
<tr>
<td>L/16</td>
<td>384</td>
<td>1.1M</td>
<td>88.2 ± 0.00</td>
<td>88.3 ± 0.06</td>
</tr>
</tbody>
</table>

Table 2: Left: We train 3 ViT models on JFT-4B in two training regimes: 200K and 1.1M steps with a batch size of 4096. The table displays their ImageNet 25 shot accuracies with and without DPN. Right: Corresponding full finetuninig results on ImageNet-1k.

6 Experiments on Downstream Tasks

6.1 Finetuning on VTAB

We finetune ImageNet-pretrained B/16 and B/32 with and without DPN on the Visual Task Adaption benchmark (VTAB) [Zhai et al. 2019]. VTAB consists of 19 datasets: 7 • Natural , 4 • Specialized and 8 • Structured . • Natural consist of datasets with natural images captured with standard cameras, • Specialized has images captured with specialized equipment and • Structured require scene comprehension.

We use the VTAB training protocol which defines a standard train split of 800 examples and a validation split of 200 examples per dataset. We perform a lightweight sweep across 3 learning rates on each dataset and use the mean validation accuracy across 3 seeds to pick the best model. Appendix E references the standard VTAB finetuning configuration. We then report the corresponding mean test score across 3 seeds in Table 3. In Table 3, accuracies within 95% confidence interval are not bolded.

On • Natural , which has datasets closest to the source dataset ImageNet, B/32 and B/16 with DPN significantly outperform the baseline on 7 out of 7 and 6 out of 7 datasets respectively. Sun397 [Xiao et al. 2010] is the only dataset where applying DPN performs worse. In Appendix F, we additionally show that DPN helps when B/16 is trained from scratch on Sun397. Applying DPN on • Structured improves accuracy on 4 out of 8 datasets and remains neutral on 2 on both B/16 and B/32. On • Specialized , DPN improves on 1 out of 4 datasets, and is neutral on 2. To conclude, DPN offers the biggest improvements, when finetuned on • Natural . On • Structured and • Specialized , DPN is a lightweight alternative, that can help or at least not hurt on a majority of datasets.
We evaluate DPN on VTab (Zhai et al., 2019). When finetuned on Natural, B/32 and B/16 with DPN significantly outperform the baseline on 7 out of 7 and 6 out of 7 datasets respectively. On Structured, DPN improves both B/16 and B/32 on 4 out of 8 datasets and remains neutral on 2. On Specialized, DPN improves on 1 out of 4 datasets, and is neutral on 2.

### 6.2 Contrastive Learning

We apply DPN on image-text contrastive learning (Radford et al., 2021). Each minibatch consists of a set of image and text pairs. We train a text and image encoder to map an image to its correct text over all other texts in a minibatch. Specifically, we adopt LiT (Zhai et al., 2022b), where we initialize and freeze the image encoder from a pretrained checkpoint and train the text encoder from scratch. To evaluate zero-shot ImageNet accuracy, we represent each ImageNet class by its text label, which the text encoder maps into a class embedding. For a given image embedding, the prediction is the class corresponding to the nearest class embedding.

We evaluate 4 frozen image encoders: 2 architectures (B/32 and L/16) trained with 2 schedules (220K and 1.1M steps). We reuse standard hyperparameters and train only the text encoder using a contrastive loss for 55000 steps with a batch-size of 16384. Table 4 shows that on B/32, DPN improves over the baselines on both the setups while on L/16 DPN provides improvement when the image encoder is trained with shorter training schedules.

### 6.3 Semantic Segmentation

We finetune ImageNet-pretrained B/16 with and without DPN on the ADE-20K 512×512 (Zhou et al., 2019) semantic segmentation task. Following Strudel et al. (2021), a single dense layer maps the ViT features into per-patch output logits. A bilinear upsampling layer then transforms the output distribution into the final high resolution 512×512 semantic segmentation output. We finetune the entire ViT backbone with standard
Table 4: Zero Shot ImageNet accuracy on the LiT (Zhai et al., 2022b) contrastive learning setup.

<table>
<thead>
<tr>
<th>Arch</th>
<th>Steps</th>
<th>Base</th>
<th>DPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>B/32</td>
<td>220K</td>
<td>61.9 ± 0.12</td>
<td>63.0 ± 0.09</td>
</tr>
<tr>
<td>B/32</td>
<td>1.1M</td>
<td>67.4 ± 0.07</td>
<td>68.0 ± 0.09</td>
</tr>
<tr>
<td>L/16</td>
<td>220K</td>
<td>75.0 ± 0.11</td>
<td>75.4 ± 0.08</td>
</tr>
<tr>
<td>L/16</td>
<td>1.1M</td>
<td>78.7 ± 0.05</td>
<td>78.7 ± 0.1</td>
</tr>
</tbody>
</table>

Table 5: We finetune ImageNet pretrained B/16 models with and without DPN on the ADE20K Semantic Segmentation task, when a varying fraction of ADE20K training data is available. The table reports the mean IoU across ten random seeds. Applying DPN improves IoU across all settings.

<table>
<thead>
<tr>
<th>Fraction of Train Data</th>
<th>1/16</th>
<th>1/8</th>
<th>1/4</th>
<th>1/2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>B/16</td>
<td>27.3 ± 0.09</td>
<td>32.6 ± 0.09</td>
<td>36.9 ± 0.13</td>
<td>40.8 ± 0.1</td>
<td>45.6 ± 0.08</td>
</tr>
<tr>
<td>+DPN</td>
<td>28.0 ± 0.21</td>
<td>33.7 ± 0.11</td>
<td>38.0 ± 0.11</td>
<td>41.9 ± 0.09</td>
<td>46.1 ± 0.11</td>
</tr>
</tbody>
</table>

Table 6: Ablations of various components of DPN. Pre: LayerNorm only to the inputs of the embedding layer. Post: LayerNorm only to the outputs of the embedding layer. Only learnable: Per-patch normalization without learnable LayerNorm parameters. No learnable: Learnable scales and shifts without standardization.

<table>
<thead>
<tr>
<th></th>
<th>B/16</th>
<th>S/16</th>
<th>B/32</th>
<th>S/32</th>
<th>Ti/16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre</td>
<td>-0.1</td>
<td>0.0</td>
<td>-2.6</td>
<td>-0.2</td>
<td>-0.3</td>
</tr>
<tr>
<td>Post</td>
<td>0.0</td>
<td>-0.2</td>
<td>-0.5</td>
<td>-0.7</td>
<td>-1.1</td>
</tr>
<tr>
<td>Post PosEmb</td>
<td>0.0</td>
<td>-0.1</td>
<td>-0.4</td>
<td>-0.9</td>
<td>-1.1</td>
</tr>
<tr>
<td>Only learnable</td>
<td>-0.8</td>
<td>-0.9</td>
<td>-1.2</td>
<td>-1.6</td>
<td>-1.6</td>
</tr>
<tr>
<td>RMSNorm</td>
<td>0.0</td>
<td>-0.1</td>
<td>-0.4</td>
<td>-0.5</td>
<td>-1.7</td>
</tr>
<tr>
<td>No learnable</td>
<td>-0.5</td>
<td>0.0</td>
<td>-0.2</td>
<td>-0.1</td>
<td>-0.1</td>
</tr>
</tbody>
</table>

7 Ablations

Is normalizing both the inputs and outputs of the embedding layer optimal? In Eq 4 DPN applies LN to both the inputs and outputs to the embedding layer. We assess three alternate strategies: Pre, Post and Post PosEmb (Radford et al., 2021). Pre applies LayerNorm only to the inputs, Post only to the outputs and Post PosEmb to the outputs after being summed with positional embeddings.

Table 6 displays the accuracy gains with two alternate strategies: Pre is unstable on B/32 leading to a significant drop in accuracy. Additionally, Pre obtains minor drops in accuracy on S/32 and Ti/16. Post and Post PosEmb achieve worse performance on smaller models B/32, S/32 and Ti/16. Our experiments show that applying LayerNorm to both inputs and outputs of the embedding layer is necessary to obtain consistent improvements in accuracy across all ViT variants.

Normalization vs Learnable Parameters: As seen in Sec. 3.2 LayerNorm constitutes a normalization operation followed by learnable scales and shifts. We also ablate the effect of each of these operations in DPN.
Applying only learnable scales and shifts without normalization leads to a significant decrease in accuracy across all architectures. (See: **Only learnable** in Table 6). Additionally, removing the learnable parameters leads to unstable training on B/16 (**No learnable** in Table 6). Finally, removing the centering and bias parameters as done in **RMSNorm** (Zhang & Sennrich, 2019), reduces the accuracy of B/32, S/32 and Ti/16. We conclude that while both normalization and learnable parameters contribute to the success of DPN, normalization has a higher impact.

8 Analysis

8.1 Gradient Norm Scale

![Figure 2: Gradient Norms with and without DPN in B/16. Left: Gradient Norm vs Depth. Right: Gradient Norm of the embedding layer vs number of steps.](a) (b)

We report per-layer gradient norms with and without DPN on B/16. **Left:** Gradient Norm vs Depth. **Right:** Gradient Norm of the embedding layer vs number of steps.

8.2 Visualizing Scale Parameters

Note that the first LayerNorm in Eq. 4 is applied directly on patches, that is, to raw pixels. Thus, the learnable parameters (biases and scales) of the first LayerNorm can be visualized directly in pixel space. Fig. 3 shows the scales of our smallest model and largest model which are: Ti/16 trained on ImageNet for 90000 steps and L/16 trained on JFT for 1.1M steps respectively. Since the absolute magnitude of the scale parameters vary across the R, G and B channel, we visualize the scale separately for each channel.

9 Conclusion

We propose a simple modification to vanilla ViT models and show its efficacy on classification, contrastive learning, semantic segmentation and transfer to small classification datasets.
Figure 3: Visualization of scale parameters of the first LayerNorm. **Top:** Ti/16 trained on ImageNet 1k. **Bottom:** L/16 trained on JFT-4B

References


Lucas Beyer, Xiaohua Zhai, and Alexander Kolesnikov. Big vision. [https://github.com/google-research/big_vision](https://github.com/google-research/big_vision), 2022c.


A Initial Project Idea

We arrived at the Dual PatchNorm solution because of another project that explored adding whitened (decorrelated) patches to ViT. Our initial prototype had a LayerNorm right after the decorrelated patches, to ensure that they are of an appropriate scale. This lead to improvements across multiple benchmarks, suggesting that whitened patches can improve image classification. We later found out via ablations, that just LayerNorm is sufficient at the inputs and adding whitened patches on their own could degrade performance. Our paper highlights the need for rigorous ablations of complicated algorithms to arrive at simpler solutions which can be equally or even more effective.

B Code

We perform all our experiments in the big-vision ([Beyer et al., 2022c](#)) and Scenic ([Dehghani et al., 2022](#)) library. Since the first LayerNorm of DPN is directly applied on pixels, we replace the first convolution with a patchify operation implemented with the einops ([Rogozhnikov, 2022](#)) library and a dense projection.

C ViT AugReg: Training Configurations

```python
import big_vision.configs.common as bvcc
from big_vision.configs.common_fewshot import get_fewshot_lsr
import ml_collections as mlc

RANDAUG_DEF = {
    'none': '',
    'light1': 'randaug(2,0)',
    'light2': 'randaug(2,10)',
    'medium1': 'randaug(2,15)',
    'medium2': 'randaug(2,15)',
    'strong1': 'randaug(2,20)',
    'strong2': 'randaug(2,20)',
}

MIXUP_DEF = {
    'none': dict(p=0.0, fold_in=None),
    'light1': dict(p=0.0, fold_in=None),
    'light2': dict(p=0.2, fold_in=None),
}```
'medium1': dict(p=0.2, fold_in=None),
'medium2': dict(p=0.5, fold_in=None),
'strong1': dict(p=0.5, fold_in=None),
'strong2': dict(p=0.8, fold_in=None),
}

def get_config(arg=None):
    """ Config for training. """
    arg = bvcc.parse_arg(arg, variant='B/32', runlocal=False, aug='')
    config = mlc.ConfigDict()
    config.pp_modules = ['ops_general', 'ops_image']
    config.init_head_bias = -6.9
    variant = 'B/16'
    aug_setting = arg.aug or {
        'T1/16': 'light1',
        'B/32': 'medium1',
        'S/16': 'medium2',
        'B/32': 'medium2',
        'B/16': 'medium2',
        'L/16': 'medium2',
    }[variant]
    config.input = dict()
    config.input.data = dict(
        name='imagenet2012',
        split='train[:99%]',
    )
    config.input.batch_size = 4096
    config.input.cache_raw = True
    config.input.shuffle_buffer_size = 250_000
    pp_common = (
        '|value_range(-1, 1)|
        |onehot(1000, key="(lbl)"), key_result="labels")|'  
        '|keep("image", "labels")'  
    )
    config.input.pp = (  
        'decode_jpeg_and_inception_crop(224)|flip_lr'|  
        RANDAUG_DEF[aug_setting] +  
        pp_common.format(lbl='label')  
    )
    pp_eval = 'decode|resize_small(256)|central_crop(224)' + pp_common
    config.input.prefetch = 8
    config.num_classes = 1000
    config.loss = 'sigmoid_xent'
    config.total_epochs = 300
    config.log_training_steps = 50
    config.ckpt_steps = 1000

    # Model section
    config.model_name = 'vit'
    config.model = dict(
        variant=variant,
        rep_size=True,
        pool_type='tok',
        dropout=0.1,
        stoch_depth=0.1,
        stem_ln='dpn')

    # Optimizer section
    config.grad_clip_norm = 1.0
    config.optax_name = 'scale_by_adam'
    config.optax = dict(mu_dtype='bfloat16')
AugReg Recipe: B/16.

For smaller models (S/32, Ti/16 and S/16), as per the AugReg recipe, we switch off stochastic depth and dropout. For S/32, we also set representation size to be false.

D ViT AugReg: High Res Finetuning

```python
def get_config(runlocal=False):
    """Config for adaptation on imagenet."""
    config = mlc.ConfigDict()
    config.loss = 'sigmoid_xent'
    config.num_classes = 1000
    config.total_steps = 5000
    config.pp_modules = ['ops_general', 'ops_image']
    config.seed = 0
    config.input = {}
    config.input.data = dict(
        name='imagenet2012',
        split='train[:99%]',
    )
    config.input.batch_size = 512 if not runlocal else 8
    config.input.shuffle_buffer_size = 50_000 if not runlocal else 100
    config.input.cache_raw = True
    variant = 'B/32'
    pp_common = {
        'value_range(-1, 1)|',
        'onehot(1000, key="{lbl}", key_result="labels")|'
        'keep("image", "labels")'
    }
    config.input.pp = {
        'decode_jpeg_and_inception_crop(384)|flip_lr|' +
        pp_common.format(lbl='label')
    }
    pp_eval = 'decode|resize_small(418)|central_crop(384)|' + pp_common
    config.log_training_steps = 10
    config.ckpt_steps = 1000
```
### VTAB Finetuning

```python
def get_config():
    """Config for adaptation on VTAB."""
    config = ConfigDict()
    config.loss = 'sigmoid_xent'
    config.num_classes = 0
    config.total_steps = 2500
    config.pp_modules = ['ops_general', 'ops_image', 'proj.vtab.pp_ops']
    config.seed = 0
    config.input = dict()
    config.input.data = dict(
        name='',
        split='train[:800]',
    )
    config.input.batch_size = 512
    config.input.shuffle_buffer_size = 50_000
    config.input.cache_raw = False
    config.input.pp = ''
    config.log_training_steps = 10
    config.log_eval_steps = 100
    config.ckpt_steps = 1000
    config.ckpt_timeout = 1
    config.prefetch_to_device = 2
```

High Resolution Finetuning
# Model
config.model_name = 'vit'
stem_ln = 'dpn'
variant = 'B/32'

config.model_init = model_inits[variant][stem_ln]
config.model = dict(
    variant=variant,
    rep_size=True,
    pool_type='tok',
    stem_ln=stem_ln)
config.model_load = dict(dont_load=['head/kernel', 'head/bias'])

# Optimizer section
config.optax_name = 'big_vision.momentum_hp'
config.grad_clip_norm = 1.0
config.wd = None
config.lr = 0.0003
config.ckpt_timeout = 3600
config.schedule = dict(
    warmup_steps=200,
    decay_type='cosine',
)

return config

High Resolution Finetuning

**F SUN397: Train from scratch**

On Sun397, applying DPN improves ViT models trained from scratch. We first search for an optimal hyperparameter setting across 3 learning rates: 1e-3, 3e-4, 1e-4, 2 weight decays: 0.03, 0.1 and two dropout values: 0.0, 0.1. We then searched across 3 mixup values 0.0, 0.2 and 0.5 and 4 randaugment distortion magnitudes 0, 5, 10 and 15. We train the final config for 600 epochs.

<table>
<thead>
<tr>
<th></th>
<th>Base</th>
<th>DPN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>41.4</td>
<td>47.5</td>
</tr>
<tr>
<td>+ Augmentation</td>
<td>48.3</td>
<td>50.7</td>
</tr>
<tr>
<td>+ Train Longer</td>
<td>52.5</td>
<td>56.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Base</th>
<th>DPN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>45.6</td>
<td>51.8</td>
</tr>
<tr>
<td>+ Augmentation</td>
<td>58.7</td>
<td>63.0</td>
</tr>
<tr>
<td>+ Train Longer</td>
<td>60.8</td>
<td>66.3</td>
</tr>
</tbody>
</table>

Table 7: Sun train from scratch. **Left:** B/32 and **Right:** B/16

**G Semantic Segmentation Hyperparameter**

def get_config():
    """Returns the base experiment configuration for Segmentation on ADE20k."""
    config = ml_collections.ConfigDict()
    config.experiment_name = 'linear_decoder_semseg_ade20k'

    # Dataset
    config.dataset_name = 'semseg_dataset'
    config.dataset_configs = ml_collections.ConfigDict()
    config.dataset_configs.name = 'ade20k'
    config.dataset_configs.use_coarse_training_data = False
    config.dataset_configs.train_data_pct = 100
    mean_std = '[0.485, 0.456, 0.406], [0.229, 0.224, 0.225]'
    common = ('|standardize('+ mean_std + ', data_key="inputs")'
              '|keep("inputs", "label"'))
    config.dataset_configs.pp_train = ('|mmseg_style_resize(img_scale=(2048, 512), ratio_range=(0.5, 2.0))')
'random_crop_with_mask(size=512, cat_max=0.75, ignore_label=0)'
'flip_with_mask'
'squeeze(data_key="label")'
'photometric_distortion(data_key="inputs")' + common
config.dataset_configs.max_size_train = 512
config.dataset_configs.pp_eval = {
'squeeze(data_key="label")' + common
config.dataset_configs.pp_test = {
'multiscale_flipaug(data_key="inputs")'
'squeeze(data_key="label")' + common

# Model.
version, patch = VARIANT.split('/')
config.model = ml_collections.ConfigDict()
config.model.hidden_size = {
'Ti': 192,
'S': 384,
'B': 768,
'L': 1024,
'H': 1280}[version]
config.model.patches = ml_collections.ConfigDict()
config.model.patches.size = [int(patch), int(patch)]
config.model.num_heads = {
'Ti': 3,
'S': 6,
'B': 12,
'L': 16,
'H': 16}[version]
config.model.mlp_dim = {
'Ti': 768,
'S': 1536,
'B': 3072,
'L': 4096,
'H': 5120}[version]
config.model.num_layers = {
'Ti': 12,
'S': 12,
'B': 12,
'L': 24,
'H': 32}[version]
config.model.attention_dropout_rate = 0.0
config.model.dropout_rate = 0.0
config.model.dropout_rate_last = 0.0
config.model.stochastic_depth = 0.1
config.model.dtype_str = 'float32'
config.model.pos_interpolation_method = 'bilinear'
config.model.concat_backbone_output = False
config.pretrained_path = ''
config.pretrained_name = 'dpn_b16'
config.model.emb_size = (32, 32) # 512 / 16
config.model.positional_embedding = 'learned'
config.model.upernet = False
config.model.fcn = True
config.model.auxiliary_loss = -1
config.model.out_with_norm = False
config.model.use_batchnorm = False
config.model.dpn = True

# Trainer.
config.trainer_name = 'segmentation_trainer'
config.eval_only = False
config.oracle_eval = False
config.window_stride = 341

# Optimizer.
config.optimizer = 'adamw'
config.weight_decay = 0.01
config.freeze_backbone = False
config.layerwise_decay = 0.
config.skip_scale_and_bias_regularization = True
config.optimizer_configs = ml_collections.ConfigDict()
config.batch_size = 16
config.num_training_epochs = 128
config.max_grad_norm = None
config.label_smoothing = None
config.class_rebalancing_factor = 0.0
config.rng_seed = 0

# Learning rate.
config.steps_per_epoch = 20210 // config.batch_size
config.total_steps = config.num_training_epochs * config.steps_per_epoch
config.lr_configs = ml_collections.ConfigDict()
config.lr_configs.learning_rate_schedule = 'compound'
config.lr_configs.factors = 'constant * polynomial * linear_warmup'
config.lr_configs.warmup_steps = 0
config.lr_configs.decay_steps = config.total_steps
config.lr_configs.base_learning_rate = 0.00003
config.lr_configs.end_factor = 0.
config.lr_configs.power = 0.9
return config

Semantic Segmentation Config

H Gradient Norm Scale

Figure 4: Gradient Norm vs Depth. **Left: B/32. Center: S/32 Right: S/16**