Three things everyone should know about ViTs

– Supplemental material –

A Baselines

↓ Training procedure	#epochs	ViT-Ti	ViT-S	ViT-B	ViT-L
DeiT [62]	300	72.2	79.8	81.8	–
Steiner et al. [58]	300	69.6	76.0	78.7	74.0
He et al. [24]	300	–	–	82.1	$81.5^{\dagger}$
He et al. [24] with EMA	300	–	–	82.3	$82.6^{\dagger}$
Our baseline	300	72.7	79.7	82.2$\pm$0.06	83.0
Our baseline with LayerScale [64]	400	73.5	80.7	82.7	84.0

Table 8. Comparison our baseline with previous training procedures. We only include results that correspond to the vanilla ViT introduced by Dosovitskiy et al. [16] for Vit-B, Vit-L and Touvron et al. [62] for Vit-Ti and ViT-S. All models are train oned ImageNet-1k at resolution 224×224 without distillation. †200 epochs.

B Transfer Learning Datasets

Table 9. Datasets used in transfer experiments and corresponding references.

Dataset	Train size	Test size	#classes
ImageNet [56]	1,281,167	50,000	1000
iNaturalist 2018 [28]	437,513	24,426	8,142
iNaturalist 2019 [28]	265,240	3,003	1,010
Flowers-102 [48]	2,040	6,149	102
Stanford Cars [34]	8,144	8,041	196
CIFAR-100 [36]	50,000	10,000	100
CIFAR-10 [36]	50,000	10,000	10

C Pytorch code of our hMLP Stem

Algorithm 1 Pseudocode of hMLP stem in a PyTorch-like style.

```
import torch
import torch.nn as nn
class hMLP_stem(nn.Module):
    """ Image to Patch Embedding
    """
    def __init__(self, img_size=(224,224), patch_size=(16,16), in_chans=3, embed_dim=768):
        super().__init__()
        num_patches = (img_size[1] // patch_size[1]) * (img_size[0] // patch_size[0])
        self.img_size = img_size
        self.patch_size = patch_size
        self.num_patches = num_patches
        self.proj = torch.nn.Sequential(
            *[nn.Conv2d(in_chans, embed_dim//4, kernel_size=4, stride=4),
            nn.SyncBatchNorm(embed_dim//4),
            nn.GELU(),
            nn.Conv2d(embed_dim//4, embed_dim//4, kernel_size=2, stride=2),
            nn.SyncBatchNorm(embed_dim//4),
            nn.GELU(),
            nn.Conv2d(embed_dim//4, embed_dim, kernel_size=2, stride=2),
            nn.SyncBatchNorm(embed_dim),
            ])

    def forward(self, x):
        B, C, H, W = x.shape
        x = self.proj(x).flatten(2).transpose(1, 2)
        return x
```