Revisiting ResNets: Improved Training and Scaling Strategies

Irwan Bello [1] **William Fedus** [1] **Xianzhi Du** [1] **Ekin D. Cubuk** [1] **Aravind Srinivas** [2] **Tsung-Yi Lin** [1] **Jonathon Shlens** [1] **Barret Zoph** [1]

Abstract

Novel computer vision architectures monopolize the spotlight, but the impact of the model architecture is often conflated with simultaneous changes to training methodology and scaling strategies. Our work revisits the canonical ResNet (He et al., 2015) and studies these three aspects in an effort to disentangle them. Perhaps surprisingly, we find that training and scaling strategies may matter more than architectural changes, and further, that the resulting ResNets match recent state-of-the-art models. We show that the best performing scaling strategy depends on the training regime and offer two new scaling strategies: (1) scale model depth in regimes where overfitting can occur (width scaling is preferable otherwise); (2) increase image resolution more slowly than previously recommended (Tan & Le, 2019). Using improved training and scaling strategies, we design a family of ResNet architectures, ResNet-RS, which are 1.7x - 2.7x faster than EfficientNets on TPUs, while achieving similar accuracies on ImageNet. In a large-scale semi-supervised learning setup, ResNet-RS achieves 86.2% top-1 ImageNet accuracy, while being 4.7x faster than EfficientNet-NoisyStudent. The training techniques improve transfer performance on a suite of downstream tasks (rivaling state-of-the-art self-supervised algorithms) and extend to video classification on Kinetics-400. We recommend practitioners use these simple revised ResNets as baselines for future research.

1. Introduction

The performance of a vision model is a product of the architecture, training methods and scaling strategy. However, research often emphasizes architectural changes. Novel architectures underlie many advances, but are often simultaneously introduced with other critical – and less publicized – changes in the details of the training methodology and hyperparameters. Additionally, new architectures enhanced by modern training methods are sometimes compared to older architectures with dated training methods (e.g. ResNet-50 with ImageNet Top-1 accuracy of 76.5% (He et al., 2015)). Our work addresses these issues and empirically studies the impact of *training methods* and *scaling strategies* on the popular ResNet architecture (He et al., 2015).

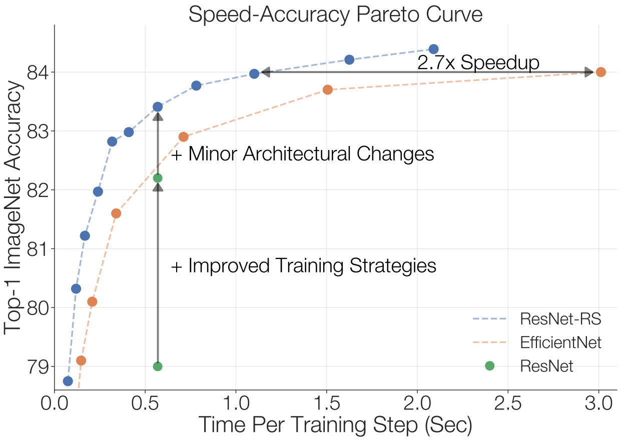

Figure 1. **Improving ResNets to state-of-the-art performance.** We improve on the canonical ResNet (He et al., 2015) with modern training methods (as also used in EfficientNets (Tan & Le, 2019)), minor architectural changes and improved scaling strategies. The resulting models, **ResNet-RS**, outperform EfficientNets on the speed-accuracy Pareto curve with speed-ups ranging from **1.7x - 2.7x** on TPUs and **2.1x - 3.3x** on GPUs. ResNet (●) is a ResNet-200 trained at 256×256 resolution. Training times reported on TPUs.

We survey the modern training and regularization techniques widely in use today and apply them to ResNets (Figure 1). In the process, we encounter interactions between

[1] Google Brain [2] UC Berkeley. Correspondence to: Irwan Bello and Barret Zoph <{ibello,barretzoph}@google.com>.

* Code and checkpoints available in TensorFlow: https://github.com/tensorflow/models/tree/master/official/vision/beta https://github.com/tensorflow/tpu/tree/master/models/official/resnet/resnet_rs

training methods and show a benefit of reducing weight decay values when used in tandem with other regularization techniques. An additive study of training methods in Table 1 reveals the significant impact of these decisions: a canonical ResNet with 79.0% top-1 ImageNet accuracy is improved to 82.2% (+3.2%) through *improved training methods alone*. This is increased further to 83.4% by two small and commonly used architectural improvements: ResNet-D (He et al., 2018) and Squeeze-and-Excitation (Hu et al., 2018). Figure 1 traces this refinement over the starting ResNet in a speed-accuracy Pareto curve.

We offer new perspectives and practical advice on scaling vision architectures. While prior works extrapolate scaling rules from small models (Tan & Le, 2019) or from training for a small number of epochs (Radosavovic et al., 2020), we design scaling strategies by exhaustively training models across a variety of scales for the full training duration (e.g. 350 epochs instead of 10 epochs). In doing so, we uncover strong dependencies between the best performing scaling strategy and the training regime (e.g. number of epochs, model size, dataset size). These dependencies are missed in any of these smaller regimes, leading to sub-optimal scaling decisions. Our analysis leads to new *scaling strategies* summarized as **(1)** scale the model depth when overfitting can occur (scaling the width is preferable otherwise) and **(2)** scale the image resolution more slowly than prior works (Tan & Le, 2019).

Using the improved training and scaling strategies, we design re-scaled ResNets, *ResNet-RS*, which are trained across a wide range of model sizes, as shown in Figure 1. ResNet-RS models use less memory during training and are **1.7x - 2.7x** faster on TPUs (**2.1x - 3.3x** faster on GPUs) than the popular EfficientNets on the speed-accuracy Pareto curve. In a large-scale semi-supervised learning setup, ResNet-RS obtains a **4.7x** training speed-up on TPUs (**5.5x** on GPUs) over EfficientNet-B5 when co-trained on ImageNet and an additional 130M pseudo-labeled images.

Finally, we conclude with a suite of experiments testing the generality of the improved training and scaling strategies. We first design a faster version of EfficientNet using our scaling strategy, *EfficientNet-RS*, which improves over the original on the speed-accuracy Pareto curve. Next, we show that the improved training strategies yield representations that rival or outperform those from self-supervised algorithms (SimCLR (Chen et al., 2020a) and SimCLRv2 (Chen et al., 2020b)) on a suite of downstream tasks. The improved training strategies extend to video classification as well. Applying the training strategies to 3D-ResNets on the Kinetics-400 dataset yields an improvement from 73.4% to 77.4% (+4.0%).

Through combining minor architectural changes (used since 2018) and improved training and scaling strategies, we discover the ResNet architecture sets a state-of-the-art baseline for vision research. This finding highlights the importance of teasing apart each of these factors in order to understand what architectures perform better than others.

We summarize our contributions:

- An empirical study of regularization techniques and their interplay, which leads to a regularization strategy that achieves strong performance (+3% top-1 accuracy) *without having to change the model architecture*.
- A simple scaling strategy: (1) scale depth when overfitting can occur (scaling width can be preferable otherwise) and (2) scale the image resolution more slowly than prior works (Tan & Le, 2019). This scaling strategy improves the speed-accuracy Pareto curve of both ResNets and EfficientNets.
- **ResNet-RS**: a Pareto curve of ResNet architectures that are **1.7x - 2.7x** faster than EfficientNets on TPUs (**2.1x - 3.3x** on GPUs) by applying the training and scaling strategies.
- Semi-supervised training of ResNet-RS with an additional 130M pseudo-labeled images achieves 86.2% top-1 ImageNet accuracy, while being **4.7x** faster on TPUs (**5.5x** on GPUs) than the corresponding EfficientNet-NoisyStudent (Xie et al., 2020).
- ResNet checkpoints that, when fine-tuned on a diverse set of computer vision tasks, rival or outperform state-of-the-art self-supervised representations from SimCLR (Chen et al., 2020a) and SimCLRv2 (Chen et al., 2020b).
- 3D ResNet-RS by extending our training methods and architectural changes to video classification. The resulted model improves the top-1 Kinetics-400 accuracy by **4.8%** over the baseline.

2. Characterizing Improvements on ImageNet

Since the breakthrough of AlexNet (Krizhevsky et al., 2012) on ImageNet (Russakovsky et al., 2015), a wide variety of improvements have been proposed to further advance image recognition performance. These improvements broadly arise along four orthogonal axes: *architecture, training/regularization methodology, scaling strategy and using additional training data*.

Architecture. The works that perhaps receive the most attention are novel architectures. Notable proposals since AlexNet (Krizhevsky et al., 2012) include VGG (Simonyan & Zisserman, 2014), ResNet (He et al., 2015), Inception (Szegedy et al., 2015; 2016), and ResNeXt (Xie et al., 2017). Automated search strategies for designing architectures have further pushed the state-of-the-art, notably with NasNet-A (Zoph et al., 2018), AmoebaNet-A (Real et al., 2019) and EfficientNet (Tan & Le, 2019). There have also been efforts in going beyond standard ConvNets for image classification, by adapting self-attention (Vaswani et al., 2017) to the visual domain (Bello et al., 2019; Ramachandran et al., 2019; Hu et al., 2019; Shen et al., 2020; Dosovitskiy et al., 2020) or using alternatives such as lambda layers (Bello, 2021).

Training and Regularization Methods. ImageNet progress has been boosted by innovations in training and regularization approaches. When training models for more epochs, regularization methods such as dropout (Srivastava et al., 2014), label smoothing (Szegedy et al., 2016), stochastic depth (Huang et al., 2016), dropblock (Ghiasi et al., 2018) and data augmentation (Zhang et al., 2017; Yun et al., 2019; Cubuk et al., 2018; 2019) have significantly improved generalization. Improved learning rate schedules (Loshchilov & Hutter, 2016; Goyal et al., 2017) have further increased final accuracy. While benchmarking architectures in a short non-regularized training setup facilitates fair comparisons with prior work, it is unclear whether architectural improvements are sustained at larger scales and improved training setups. For example, the RegNet architecture (Radosavovic et al., 2020) shows strong speedups over baselines in a short non-regularized training setup, but was not tested in a state-of-the-art ImageNet setup (best top-1 is 79.9%).

Scaling Strategies. Increasing the model dimensions (e.g. width, depth and resolution) has been another successful axis to improve quality (Rosenfeld et al., 2019; Hestness et al., 2017). Sheer scale was exhaustively demonstrated to improve performance of neural language models (Kaplan et al., 2020) which motivated the design of ever larger models including GPT-3 (Brown et al., 2020) and Switch Transformer (Fedus et al., 2021). Similarly, scale in computer vision has proven useful. Huang et al. (2018) designed and trained a 557 million parameter model, AmoebaNet, which achieved 84.4% top-1 ImageNet accuracy. Typically, ResNet architectures are scaled up by adding layers (depth): ResNets, suffixed by the number of layers, have marched onward from ResNet-18 to ResNet-200, and beyond (He et al., 2016; Zhang et al., 2020; Bello, 2021). Wide ResNets (Zagoruyko & Komodakis, 2016) and MobileNets (Howard et al., 2017) instead scale the width. Increasing image resolutions has also been a reliable source of progress. Thus as training budgets have grown, so have the image resolutions: EfficientNet uses 600 image resolutions (Tan & Le, 2019) and both ResNeSt (Zhang et al., 2020) and TResNet (Ridnik et al., 2020) use 448 image resolutions for their largest model. In an attempt to systematize these heuristics, EfficientNet proposed the compound scaling rule, which recommended balancing the network depth, width and image resolution. However, Section 7.2 shows this scaling strategy is sub-optimal for not only ResNets, but EfficientNets as well.

Additional Training Data. Another popular way to further improve accuracy is by training on additional sources of data (either labeled, weakly labeled, or unlabeled). Pretraining on large-scale datasets (Sun et al., 2017; Mahajan et al., 2018; Kolesnikov et al., 2019) has significantly pushed the state-of-the-art, with ViT (Dosovitskiy et al., 2020) and NFNets (Brock et al., 2021) recently achieving 88.6% and 89.2% ImageNet accuracy respectively. Noisy Student, a semi-supervised learning method, obtained 88.4% ImageNet top-1 accuracy by using pseudo-labels on an extra 130M unlabeled images (Xie et al., 2020). Meta pseudo-labels (Pham et al., 2020), an improved semi-supervised learning technique, currently holds the ImageNet state-of-the-art (90.2%). We present semi-supervised learning results in Table 4 and discuss how our training and scaling strategies transfer to large data regimes in Section 8.

3. Related Work on Improving ResNets

Improved training methods combined with architectural changes to ResNets have routinely yielded competitive ImageNet performance (He et al., 2018; Lee et al., 2020; Ridnik et al., 2020; Zhang et al., 2020; Bello, 2021; Brock et al., 2021). He et al. (2018) achieved 79.2% top-1 ImageNet accuracy (a +3% improvement over their ResNet-50 baseline) by modifying the stem and downsampling block while also using label smoothing and mixup. Lee et al. (2020) further improved the ResNet-50 model with additional architectural modifications such as Squeeze-and-Excitation (Hu et al., 2018), selective kernel (Li et al., 2019), and anti-alias downsampling (Zhang, 2019), while also using label smoothing, mixup, and dropblock to achieve 81.4% accuracy. Ridnik et al. (2020) incorporated several architectural modifications to the ResNet architectures along with improved training methodologies to outperform EfficientNet-B1 to EfficientNet-B5 models on the speed-accuracy Pareto curve.

Most works, however, put little emphasis on identifying strong scaling strategies. In contrast, we only consider lightweight architectural changes routinely used since 2018 and instead focus on the training and scaling strategies to build a Pareto curve of models. Our improved training and scaling methods lead to ResNets that are **1.7x - 2.7x** faster than EfficientNets on TPUs. Our scaling improvements are orthogonal to the aforementioned methods and we expect them to be additive.

4. Methodology

We describe the base ResNet architecture and the training methods used throughout this paper.

4.1. Architecture

Our work studies the ResNet architecture, with two widely used architecture changes, the ResNet-D (He et al., 2018) modification and Squeeze-and-Excitation (SE) in all bottleneck blocks (Hu et al., 2018). These architectural changes are used in used many architectures, including TResNet, ResNeSt and EfficientNets.

ResNet-D (He et al., 2018) combines the following four adjustments to the original ResNet architecture. First, the 7×7 convolution in the stem is replaced by three smaller 3×3 convolutions, as first proposed in Inception-V3 (Szegedy et al., 2016). Second, the stride sizes are switched for the first two convolutions in the residual path of the downsampling blocks. Third, the stride-2 1×1 convolution in the skip connection path of the downsampling blocks is replaced by stride-2 2×2 average pooling and then a non-strided 1×1 convolution. Fourth, the stride-2 3×3 max pool layer is removed and the downsampling occurs in the first 3×3 convolution in the next bottleneck block. We diagram these modifications in Figure 6.

Squeeze-and-Excitation (Hu et al., 2018) reweighs channels via cross-channel interactions by average pooling signals from the entire feature map. For all experiments we use a Squeeze-and-Excitation ratio of 0.25 based on preliminary experiments. In our experiments, we sometimes use the original ResNet implementation without SE (referred to as ResNet) to compare different training methods. Clear denotations are made in table captions when this is the case.

4.2. Training Methods

We study regularization and data augmentation methods that are routinely used in state-of-the art classification models and semi/self-supervised learning.

Matching the EfficientNet Setup. Our training method closely matches that of EfficientNet, where we train for 350 epochs, but with a few small differences. **(1)** We use the cosine learning rate schedule (Loshchilov & Hutter, 2016) instead of an exponential decay for simplicity (no additional hyperparameters). **(2)** We use RandAugment (Cubuk et al., 2019) in all models, whereas EfficientNets were originally trained with AutoAugment (Cubuk et al., 2018). We reran EfficientNets B0-B4 with RandAugment and found it offered no performance improvement and report EfficientNet B5 and B7 with the RandAugment results from Cubuk et al. (2019)[1]. **(3)** We use the Momentum optimizer instead of RMSProp for simplicity. See Table 10 in the Appendix C for a comparison between our training setup and EfficientNet.

Regularization. We apply *weight decay*, *label smoothing*, *dropout* and *stochastic depth* for regularization. Dropout (Srivastava et al., 2014) is a common technique used in computer vision and we apply it to the output after the global average pooling occurs in the final layer. Stochastic depth (Huang et al., 2016) drops out each layer in the network (that has residual connections around it) with a specified probability that is a function of the layer depth.

Data Augmentation. We use RandAugment (Cubuk et al., 2019) data augmentation as an additional regularizer. RandAugment applies a sequence of random image transformations (e.g. translate, shear, color distortions) to each image independently during training. As mentioned earlier, originally EfficientNets uses AutoAugment (Cubuk et al., 2018), which is a learned augmentation procedure that slightly underperforms RandAugment.

Hyperparameter Tuning. To select the hyperparameters for the various regularization and training methods, we use a held-out validation set comprising 2% of the ImageNet training set (20 shards out of 1024). This is referred to as the `minival-set` and the original ImageNet validation set (the one reported in most prior works) is referred to as `validation-set`. The hyperparameters of all ResNet-RS models are in Table 8 in the Appendix B.

5. Improved Training Methods

5.1. Additive Study of Improvements

We present an additive study of training, regularization methods and architectural changes in Table 1. The baseline ResNet-200 gets 79.0% top-1 accuracy. We improve its performance to 82.2% (+3.2%) through *improved training methods alone* without any architectural changes. When adding two common and simple architectural changes (Squeeze-and-Excitation and ResNet-D) we further boost the performance to 83.4%. Training methods alone cause 3/4 of the total improvement, which demonstrates their critical impact on ImageNet performance.

5.2. Importance of decreasing weight decay when combining regularization methods

Table 2 highlights the importance of changing weight decay when combining regularization methods together.

[1] This makes our comparison to EfficientNet-B6 more nuanced as the B6 performance most likely could be improved by 0.1-0.3% top-1 if ran with RandAugment (based on improvements obtained from B5 and B7).

Improvements	Top-1	Δ
ResNet-200	79.0	—
+ Cosine LR Decay	79.3	**+0.3**
+ Increase training epochs	78.8 †	-0.5
+ EMA of weights	79.1	**+0.3**
+ Label Smoothing	80.4	**+1.3**
+ Stochastic Depth	80.6	**+0.2**
+ RandAugment	81.0	**+0.4**
+ Dropout on FC	80.7 ‡	-0.3
+ Decrease weight decay	82.2	**+1.5**
+ Squeeze-and-Excitation	82.9	**+0.7**
+ ResNet-D	83.4	**+0.5**

Table 1. **Additive study of the ResNet-RS training recipe.** The colors refer to **Training Methods** , **Regularization Methods** and **Architecture Improvements** . The baseline ResNet-200 was trained for the standard 90 epochs using a stepwise learning rate decay schedule. The image resolution is 256×256. All numbers are reported on the ImageNet `validation-set` and averaged over 2 runs. † Increasing training duration to 350 epochs only becomes useful once the regularization methods are used, otherwise the accuracy drops due to over-fitting. ‡ dropout hurts as we have not yet decreased the weight decay (See Table 2 for more details).

Model	Regularization	Weight Decay		Δ
		1e-4	4e-5	
ResNet-50	None	79.7	78.7	-1.0
ResNet-50	RA-LS	82.4	82.3	-0.1
ResNet-50	RA-LS-DO	82.2	82.7	+0.5
ResNet-200	None	82.5	81.7	-0.8
ResNet-200	RA-LS	85.2	84.9	-0.3
ResNet-200	RA-LS-SD-DO	85.3	85.5	+0.2

Table 2. **Decrease weight decay when using more regularization.** Top-1 ImageNet accuracy for different regularization combinations. Decreasing the weight decay improves performance when combining regularization methods such as dropout (DO), stochastic depth (SD), label smoothing (LS) and RandAugment (RA). Image resolution is 224×224 for ResNet-50 and 256×256 for ResNet-200. All numbers are reported on the ImageNet `minival-set` from an average of two runs.

When applying RandAugment and label smoothing, there is no need to change the default weight decay of 1e-4. But when we further add dropout and/or stochastic depth, the performance can decrease unless we further decrease the weight decay. The intuition is that since weight decay acts as a regularizer, its value must be decreased in order to not overly regularize the model when combining many techniques. Furthermore, Zoph et al. (2020a) presents evidence that the addition of data augmentation shrinks the L2 norm of the weights, which renders some of the effects of weight decay redundant. Other works use smaller weight decay values, but do not point out the significance of the effect when using more regularization (Tan et al., 2019; Tan & Le, 2019).

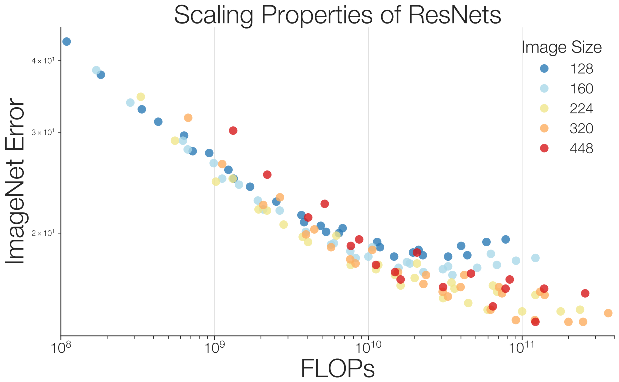

Figure 2. **Scaling properties of ResNets across varying model scales.** Error approximately scales as a power law with FLOPs (linear fit on the log-log curve) in the lower FLOPs regime but the trend breaks for larger FLOPs. We observe diminishing returns of scaling the image resolutions beyond 320×320, which motivates the slow image resolution scaling (Strategy #2). The scaling configurations run are width multipliers `[0.25,0.5,1.0,1.5,2.0]`, depths `[26,50,101,200,300,350,400]` and image resolutions `[128,160,224,320,448]`. FLOPs is the number of floating point operations per image. All results are on the ImageNet `minival-set`.

6. Improved Scaling Strategies

The prior section demonstrates the significant impact of training methodology and we now show the scaling strategy is similarly important. In order to establish scaling trends, we perform an extensive search on ImageNet over width multipliers in `[0.25,0.5,1.0,1.5,2.0]`, depths of `[26,50,101,200,300,350,400]` and resolutions of `[128,160,224,320,448]`. We train these architectures for 350 epochs, mimicking the training setup of state-of-the-art ImageNet models. We increase the regularization as the model size increases to limit overfitting. See Appendix E for regularization and model hyperparameters.

FLOPs do not accurately predict performance in the bounded data regime. Prior works on scaling laws observe a power law between error and FLOPs in *unbounded data regimes* (Kaplan et al., 2020; Henighan et al., 2020). In order to test whether this also holds in our scenario, we plot ImageNet error against FLOPs for all scaling configurations in Figure 2. For the smaller models, we observe an overall power law trend between error and FLOPs, with minor dependency on the scaling configuration (i.e. depth versus width versus image resolution). However, the trend breaks for larger model sizes. Furthermore, we observe a large variation in ImageNet performance for a fixed amount of FLOPs, especially in the higher FLOP regime. Therefore the exact scaling configuration (i.e. depth, width and

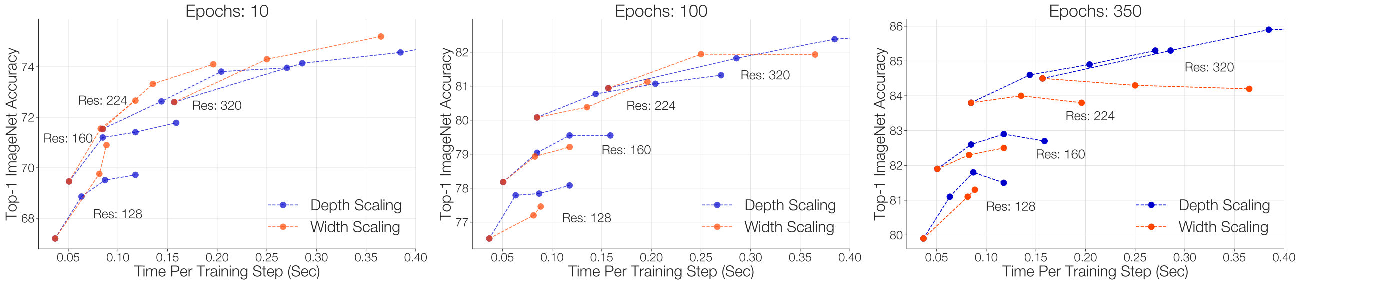

Figure 3. **Scaling of ResNets across depth, width, image resolution and training epochs**. We compare depth scaling and width scaling across four different image resolutions `[128,160,224,320]` when training models for 10, 100 or 350 epochs. We find that *the best performing scaling strategy depends on the training regime*, which reveals the pitfall of extrapolating scaling rules from small scale regimes. **(Left) 10 Epoch Regime**: width scaling is the best strategy for the speed-accuracy Pareto curve. **(Middle) 100 Epoch Regime**: depth scaling is sometimes outperformed by width scaling. **(Right) 350 Epoch Regime**: depth scaling consistently outperforms width scaling by a large margin. Overfitting remains an issue even when using regularization methods. **Model Details:** All models start from a depth of 101 and are increased through `[101,200,300,400]`. All model widths start with a multiplier of `1.0x` and are increased through `[1.0,1.5,2.0]`. For all models, we tune regularization in an effort to limit overfitting (see Appendix E). Accuracies are reported on the ImageNet `minival-set` and training times are measured on TPUs.

image resolution) can have a big impact on performance even when controlling for the same amount of FLOPs.

The best performing scaling strategy depends on the training regime. We next look directly at latencies[2] on the hardware of interest to identify scaling strategies that improve the speed-accuracy Pareto curve. Figure 3 presents accuracies and latencies of models scaled with either width or depth across four image resolutions and three different training regimes (10, 100 and 350 epochs). We observe that the best performing scaling strategy, especially whether to scale depth and/or width, highly depends on the training regime.

6.1. Strategy #1 - Depth Scaling in Regimes Where Overfitting Can Occur

Depth scaling outperforms width scaling for longer epoch regimes. In the 350 epochs setup (Figure 3, right panel), we observe depth scaling to significantly outperform width scaling across all image resolutions. Scaling the width is subject to overfitting and sometimes hurts performance even with increased regularization. We hypothesize that this is due to the larger increase in parameters when scaling the width. The ResNet architecture maintains constant FLOPs across all block groups and multiplies the number of parameters by 4× every block group. Scaling the depth, especially in the earlier layers, therefore introduces fewer parameters compared to scaling the width.

Width scaling outperforms depth scaling for shorter epoch regimes. In contrast, width scaling is better when only training for 10 epochs (Figure 3, left panel). For 100 epochs (Figure 3, middel panel), the best performing scaling strategy varies between depth scaling and width scaling, depending on the image resolution. The dependency of the scaling strategy on the training regime reveals a pitfall of extrapolating scaling rules. We point out that prior works also choose to scale the width when training for a small number of epochs on large-scale datasets (e.g. ∼40 epochs on 300M images), consistent with our experimental findings that scaling the width is preferable in shorter epoch regimes. In particular, Kolesnikov et al. (2019) train a ResNet-152 with 4x filter multiplier while Brock et al. (2021) scales the width with ∼1.5x filter multiplier.

6.2. Strategy #2 - Slow Image Resolution Scaling

In Figure 2, we also observe that larger image resolutions yield diminishing returns. We therefore propose to increase the image resolution more gradually than previous works. This contrasts with the compound scaling rule proposed by EfficientNet which leads to very large images (e.g. 600 for EfficientNet-B7, 800 for EfficientNet-L2 (Xie et al., 2020)). Other works such as ResNeSt (Zhang et al., 2020) and TResNet (Ridnik et al., 2020)) scale the image resolution up to 448. Our experiments indicate that slower image scaling improves not only ResNet architectures, but also EfficientNets on a speed-accuracy basis (Section 7.2).

6.3. Two Common Pitfalls in Designing Scaling Strategies

Our scaling analysis surfaces two common pitfalls in prior research on scaling strategies:

(1) Extrapolating scaling strategies from small-scale regimes. Scaling strategies found in small scale regimes (e.g. on small models or with few training epochs) can

[2] FLOPs is not a good indicator of latency on modern hardware. See Section 7.1 for a more detailed discussion.

fail to generalize to larger models or longer training iterations. The dependencies between the best performing scaling strategy and the training regime are missed by prior works which extrapolate scaling rules from either small models (Tan & Le, 2019) or shorter training epochs (Radosavovic et al., 2020). We therefore do not recommend generating scaling rules exclusively in a small scale regime because these rules can break down.

(2) Extrapolating scaling strategies from a single and potentially sub-optimal initial architecture. Beginning from a sub-optimal initial architecture can skew the scaling results. For example, the compound scaling rule derived from a small grid search around EfficientNet-B0, which was obtained by architecture search using a fixed FLOPs budget and a specific *image resolution*. However, since this image resolution can be sub-optimal for that FLOPs budget, the resulting scaling strategy can be sub-optimal. In contrast, our work designs scaling strategies by training models across a variety of widths, depths and image resolutions.

6.4. Summary of Improved Scaling Strategies

For a new task, we recommend running a *small subset* of models across different scales, for the full training epochs, to gain intuition on which dimensions are the most useful across model scales. While this approach may appear more costly, we point out that the cost is offset by not searching for the architecture.

For image classification, the scaling strategies are summarized as **(1)** scale the depth in regimes where overfitting can occur (scaling the width is preferable otherwise) and **(2)** slow image resolution scaling. Experiments indicate that applying these scaling strategies to ResNets (ResNet-RS) and EfficientNets (EfficientNet-RS) leads to significant speed-ups over EfficientNets. We note that similar scaling strategies are also employed in recent works that obtain large speed-ups over EfficientNets such as LambdaResNets (Bello, 2021) and NFNets (Brock et al., 2021).

7. Experiments with Improved Training and Scaling Strategies

7.1. ResNet-RS on a Speed-Accuracy Basis

Using the improved training and scaling strategies, we design *ResNet-RS*, a family of re-scaled ResNets across a wide range of model scales (see Appendix B and D for experimental and architectural details). Figure 4 compares EfficientNets against ResNet-RS on a speed-accuracy Pareto curve. We find that ResNet-RS match EfficientNets' performance while being *1.7x - 2.7x* faster on TPUs. This large speed-up over EfficientNet may be non-intuitive since EfficientNets significantly reduce both the parameter count and the FLOPs compared to ResNets. We next discuss why a model with fewer parameters and fewer FLOPs (EfficientNet) is slower and more memory-intensive during training.

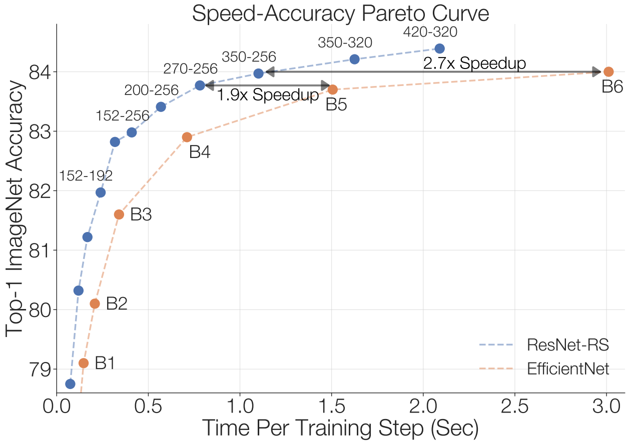

Figure 4. **Speed-Accuracy Pareto curve comparing ResNets-RS to EfficientNet.** Properly scaled ResNets (ResNet-RS) are **1.7x - 2.7x** faster than the popular EfficientNets when closely matching their training setup. ResNet-RS are annotated with (depth - image resolution), so 152-256 means ResNet-RS-152 with image resolution 256×256. All results are on the ImageNet `validation-set` and training times are measured on TPUs. See Appendix B for detailed results.

FLOPs vs Latency. While FLOPs provide a hardware-agnostic metric for assessing computational demand, they may not be indicative of actual latency times for training and inference (Howard et al., 2017; 2019; Radosavovic et al., 2020). In custom hardware architectures (e.g. TPUs and GPUs), FLOPs are an especially poor proxy because operations are often bounded by memory access costs and have different levels of optimization on modern matrix multiplication units (Jouppi et al., 2017). The inverted bottlenecks (Sandler et al., 2018) used in EfficientNets employ depthwise convolutions with large activations and have a *small compute to memory ratio* (operational intensity) compared to the ResNet's bottleneck blocks which employ dense convolutions on smaller activations. This makes EfficientNets less efficient on modern accelerators compared to ResNets. Table 3 illustrates this point: a ResNet-RS model with **1.8x** more FLOPs than EfficientNet-B6 is **2.7x** faster on a TPUv3 hardware accelerator.

Parameters vs Memory. Parameter count does not necessarily dictate memory consumption during *training* because memory is often dominated by the size of the *acti-*

Model	**RS-350**	**ENet-B6**	**RS-420**	**ENet-B7**
Resolution	256	528	320	600
Top-1 Acc.	**84.0**	**84.0**	84.4	**84.7**
Params (M)	164	43 (3.8x)	192	66 (2.9x)
FLOPs (B)	69	38 (1.8x)	128	74 (1.7x)
TPU-v3				
Latency (s)	1.1	3.0 (2.7x)	2.1	6.0 (2.9x)
Memory (GB)	7.3	16.6 (2.3x)	15.5	28.3 (1.8x)
V100				
Latency (s)	4.7	15.7 (3.3x)	10.2	29.9 (2.8x)

Table 3. **Performance comparison of ResNet-RS and EfficientNet** (abbreviated ENet). Although ResNet-RS has more parameters and FLOPs, the model employs less memory and runs faster on TPUs and GPUs. TPU latency is reported as the time per training step for 1024 images on 8 TPUv3 cores. Memory is reported on 32 images per core, using `bfloat16` precision without fusion or rematerialization. See Appendix H for more profiling details.

vations[3]. The large activations used in EfficientNets also cause larger memory consumption, which is exacerbated by the use of large image resolutions, compared to our rescaled ResNets. A ResNet-RS model with **3.8x** more parameters than EfficientNet-B6 consumes **2.3x** less memory for a similar ImageNet accuracy (Table 3). We emphasize that both memory consumption and latency are tightly coupled to the software and hardware stack (TensorFlow on TPUv3) due to compiler optimizations such as operation layout assignments and memory padding.

7.2. Improving the Efficiency of EfficientNets

The scaling analysis from Section 6 reveals that scaling the image resolution results in *diminishing* returns. This suggests that the scaling rules advocated in EfficientNets which increases model depth, width and resolution *independently* of model scale is sub-optimal. We apply the slow image resolution scaling strategy (Strategy #2) to EfficientNets and train several versions with reduced image resolutions, without changing the width or depth. The RandAugment magnitude is set to 10 for image resolution 224 or smaller, 20 for image resolution larger than 320 and 15 otherwise. All other hyperparameters are kept the same as per the original EfficientNets. Figure 5 demonstrates a marked improvement of the re-scaled EfficientNets (EfficientNet-RS) on the speed-accuracy Pareto curve over the original EfficientNets.

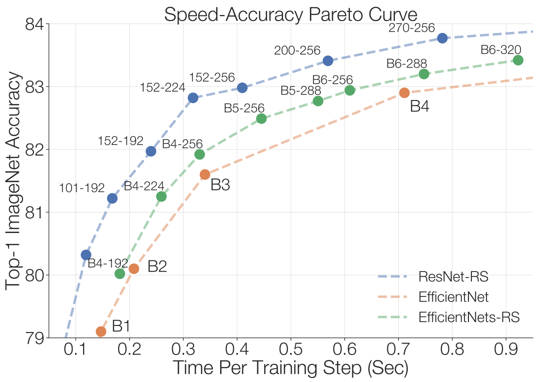

Figure 5. **Speed-Accuracy Pareto curve comparing ResNets-RS and EfficientNet-RS to EfficientNet.** Scaling EfficientNets using the slow image resolution scaling strategy (instead of the original compound scaling rule) improves the Pareto efficiency of EfficientNets. Note that ResNet-RS still outperforms EfficientNet-RS. This figure is a zoomed in version of Figure 4 with EfficientNet-RS added. Models are annotated with (model depth - image resolution), so 152-192 corresponds to ResNet-RS-152 with image resolution 192×192. Results are reported on the ImageNet `validation-set` and training times are measured on TPUs.

7.3. Semi-Supervised Learning with ResNet-RS

We measure how ResNet-RS performs as we scale to larger datasets in a large scale semi-supervised learning setup. We train ResNets-RS on the combination of 1.2M labeled ImageNet images and 130M *pseudo-labeled* images, in a similar fashion to Noisy Student (Xie et al., 2020). We use the same dataset of 130M images pseudo-labeled as Noisy Student, where the pseudo labels are generated from an EfficientNet-L2 model with 88.4% ImageNet accuracy. Models are jointly trained on both the labeled and pseudo-labeled data and training hyperparameters are kept the same. Table 4 reveals that ResNet-RS models are very strong in the semi-supervised learning setup as well. We obtain a top-1 ImageNet accuracy of 86.2%, while being **4.7x** faster on TPU (**5.5x** on GPU) than the corresponding Noisy Student EfficientNet-B5 model.

Model	V100 (s)	TPUv3 (ms)	Top-1
EfficientNet-B5	8.16	1510	86.1
ResNet-RS-152	**1.48 (5.5x)**	**320 (4.7x)**	**86.2**

Table 4. **ResNet-RS are efficient semi-supervised learners.** ResNet-RS-152 with image resolution 224 is **4.7x** faster on TPU (**5.5x** on GPU) than EfficientNet-B5 Noisy Student (Xie et al., 2020) for a similar ImageNet accuracy. Both models train on the same additional 130M pseudo-labeled images. See Appendix H for details on latency measurements.

[3] Activations are typically stored during training as they are used in backpropagation. At inference, activations can be discarded and parameter count is a better proxy for actual memory consumption.

Model	Training Method	Epochs	CIFAR-100 Accuracy	Pascal Detection	Pascal Segmentation	ADE Segmentation	NYU Depth
ResNet-152	Supervised	90	85.5	80.0	70.0	40.2	81.2
ResNet-152	SimCLR	800	87.1	**83.3**	72.2	41.0	83.5
ResNet-152	SimCLRv2	800	84.7	79.1	73.1	41.1	**84.7**
ResNet-152	RS	400	**88.1**	82.2	**78.2**	**42.2**	83.4
ResNet-152 2x	Supervised	90	86.6	81.1	72.2	41.4	82.5
ResNet-152 2x	SimCLR	800	89.0	**85.3**	78.8	**45.2**	**86.8**
ResNet-152 2x	SimCLRv2	800	84.8	80.1	75.5	42.5	86.1
ResNet-152 2x	RS	400	**89.3**	84.1	**79.2**	44.1	84.7

Table 5. **Representations from supervised learning with improved training strategies rival or outperform representations from state-of-the-art self-supervised learning algorithms.** Comparison of supervised training methods (supervised, RS) and self-supervised methods (SimCLR, SimCLRv2) on a variety of downstream tasks. The (RS) strategy greatly outperforms the baseline supervised training, which highlights the importance of using improved supervised training techniques when comparing to self-supervised learning algorithms. The RS training method uses a subset of the training methods highlighted in this work (cosine LR decay, RandAugment label smoothing, reduced weight decay, and dropout on FC) to more closely match those used in the self-supervised algorithms. All models employ the *vanilla* ResNet architecture without modifications and are pre-trained on ImageNet.

7.4. Transfer Learning of ResNet-RS

We now investigate whether the improved supervised training strategies yield better representations for transfer learning and compare them with self-supervised learning algorithms. Recent self-supervised learning algorithms claim to surpass the transfer learning performance of *supervised learning* and create more universal representations (Chen et al., 2020a;b). Self-supervised algorithms, however, make several changes to the training methods (e.g training for more epochs, data augmentation) making comparisons to supervised learning difficult. Table 5 compares the transfer performance of improved supervised training strategies (denoted RS) against self-supervised SimCLR (Chen et al., 2020a) and SimCLRv2 (Chen et al., 2020b). In an effort to closely match SimCLR's training setup and provide fair comparisons, we restrict the RS training strategies to a subset of its original methods. Specifically, we employ data augmentation (RandAugment), label smoothing, dropout, decreased weight decay and cosine learning rate decay for 400 epochs but do not use stochastic depth or exponential moving average (EMA) of the weights. We choose this subset to closely match the training setup of SimCLR: longer training, data augmentation and a temperature parameter for their contrastive loss[4]. We use the vanilla ResNet architecture without the ResNet-D modifications or Squeeze-and-Excite, matching the SimCLR and SimCLRv2 architectures.

We evaluate the transfer performance on five downstream tasks: CIFAR-100 Classification (Krizhevsky et al., 2009), Pascal Detection & Segmentation (Everingham et al., 2010), ADE Segmentation (Zhou et al., 2017) and NYU Depth (Silberman et al., 2012). We find that, even when restricted to a smaller subset, the improved training strategies improve transfer performance[5]. The improved supervised representations (RS) outperform SimCLR on $5/10$ downstream tasks and SimCLRv2 on $8/10$ tasks. Furthermore, the improved training strategies significantly outperform the standard supervised ResNet representations, highlighting the need for using modern training techniques when comparing to self-supervised learning. While self-supervised learning can be used on *unlabeled data*, our results challenge the notion that self-supervised algorithms lead to more universal representations than supervised learning when labels are available.

7.5. Revised 3D ResNet for Video Classification

We conclude by applying the training strategies to the Kinetics-400 video classification task, using a 3D ResNet as the baseline architecture (Qian et al., 2020) (see Appendix G for experimental details). Table 6 presents an additive study of the RS training recipe and architectural improvements.

The training strategies extend to video classification, yielding a combined improvement from 73.4% to 77.4% (+4.0%). The ResNet-D and Squeeze-and-Excitation architectural changes further improve the performance to 78.2% (+0.8%). Similarly to our study on image classification (Table 1), we find that most of the improvement can

[4] Note that SimCLR and SimCLRv2 might benefit further when combining with RandAugment, but the same may also hold true when combining SimCLR's augmentation with RandAugment for supervised learning.

[5] Kornblith et al. (2019) similarly observed that better ImageNet top-1 accuracy (either through better architectures or training strategies) strongly correlates with improved transfer learning performance.

Improvements	Top-1	Δ
3D ResNet-50	73.4	–
+ Dropout on FC	74.4	**+1.0**
+ Label smoothing	74.9	**+0.5**
+ Stochastic depth	76.1	**+1.2**
+ EMA of weights	76.1	–
+ Decrease weight decay	76.3	**+0.2**
+ Increase training epochs	76.4	**+0.1**
+ Scale jittering	77.4	**+1.0**
+ Squeeze-and-Excitation	77.9	**+0.5**
+ ResNet-D	78.2	**+0.3**

Table 6. **Additive study of training methods for video classification.** The colors refer to **Training Methods**, **Regularization Methods** and **Architecture Improvements**. The ResNet-RS training recipe transfers to a 3D ResNet model on Kinetics-400 video classification (Kay et al., 2017). Reported accuracies are averaged over 2 runs. The baseline 3D ResNet-50 was trained for 200 epochs with a cosine learning rate decay.

be obtained without architectural changes. Without model scaling, 3D ResNet-RS-50 is only 2.2% less than the best number reported on Kinetics-400 at 80.4% (Feichtenhofer, 2020).

8. Discussion

Why is it important to tease apart improvements coming from training methods vs architectures? Training methods can be more task-specific than architectures (e.g. data augmentation is more helpful on small datasets). Therefore, improvements coming from training methods do not necessarily generalize as well as architectural improvements. Packaging newly proposed architectures together with training improvements makes accurate comparisons between architectures difficult. The large improvements coming from training strategies, when not being controlled for, can overshadow architectural differences.

How should one compare different architectures? Since training methods and scale typically improve performance (Lee et al., 2020; Kaplan et al., 2020), it is critical to control for both aspects when comparing different architectures. Controlling for scale can be achieved through different metrics. While many works report parameters and FLOPs, we argue that latencies and memory consumption are generally more relevant (Radosavovic et al., 2020). Our experimental results (Section 7.1) re-emphasize that FLOPs and parameters are not representative of latency or memory consumption (Radosavovic et al., 2020; Norrie et al., 2021).

Do the improved training strategies transfer across tasks? The answer depends on the domain and dataset sizes available. Many of the training and regularization methods studied here are not used in large-scale pre-training (e.g. 300M images) (Kolesnikov et al., 2019; Dosovitskiy et al., 2020). Data augmentation is useful for small datasets or when training for many epochs, but the specifics of the augmentation method can be task-dependent (e.g. scale jittering instead of RandAugment in Table 6).

Do the scaling strategies transfer across tasks? The best performing scaling strategy depends on the training regime and whether overfitting is an issue, as discussed in Section 6. When training for 350 epochs on ImageNet, we find scaling the depth to work well, whereas scaling the width is preferable when training for few epochs (e.g. 10 epochs). This is consistent with works employing width scaling when training for few epochs on large-scale datasets (Kolesnikov et al., 2019). We are unsure how our scaling strategies apply in tasks that require larger image resolutions (e.g. detection and segmentation) and leave this to future work.

Are architectural changes useful? Yes, but training methods and scaling strategies can have even larger impacts. Simplicity often wins, especially given the non-trivial performance issues arising on custom hardware. Architecture changes that decrease speed and increase complexity may be surpassed by scaling up faster and simpler architectures that are optimized on available hardware (e.g convolutions instead of depthwise convolutions for GPUs/TPUs). We envision that future successful architectures will emerge by co-design with hardware, particularly in resource-tight regimes like mobile phones (Howard et al., 2019).

How should one allocate a computational budget to produce the best vision models? We recommend beginning with a simple architecture that is efficient on available hardware (e.g. ResNets on GPU/TPU) and training several models, to convergence, with different image resolutions, widths and depths to construct a Pareto curve. Note that this strategy is distinct from Tan & Le (2019) which instead allocate a large portion of the compute budget for identifying an optimal initial architecture to scale. They then do a small grid search to find the compound scaling coefficients used across all model scales. RegNet (Radosavovic et al., 2020) does most of their studies when training for only 10 epochs.

9. Conclusion

By updating the de facto vision baseline with modern training methods and an improved scaling strategy, we have revealed the remarkable durability of the ResNet architecture. Simple architectures set strong baselines for state-of-the-art methods. We hope our work encourages further scrutiny in maintaining consistent methodology for both proposed innovations and baselines alike.

Acknowledgements. We would like to thank Ashish Vaswani, Prajit Ramachandran, Ting Chen, Thang Luong, Hanxiao Liu, Gabriel Bender, Quoc Le, Neil Houlsby, Mingxing Tan, Andrew Howard, Raphael Gontijo Lopes, Andy Brock and David Berthelot for helpful feedback on this work; Jing Li, Pengchong Jin, Yeqing Li and Yin Cui for the support on open-sourcing and infrastructure.

References

Bello, I. Lambdanetworks: Modeling long-range interactions without attention. *International Conference in Learning Representations*, 2021.

Bello, I., Zoph, B., Vaswani, A., Shlens, J., and Le, Q. V. Attention augmented convolutional networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3286–3295, 2019.

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

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pp. 1597–1607. PMLR, 2020a.

Chen, T., Kornblith, S., Swersky, K., Norouzi, M., and Hinton, G. Big self-supervised models are strong semi-supervised learners. *arXiv preprint arXiv:2006.10029*, 2020b.

Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., and Le, Q. V. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018.

Cubuk, E. D., Zoph, B., Shlens, J., and Le, Q. V. Randaugment: Practical automated data augmentation with a reduced search space. *arXiv preprint arXiv:1909.13719*, 2019.

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

Everingham, M., Van Gool, L., Williams, C. K., Winn, J., and Zisserman, A. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88 (2):303–338, 2010.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*, 2021.

Feichtenhofer, C. X3d: Expanding architectures for efficient video recognition. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 200–210, 2020.

Feichtenhofer, C., Fan, H., Malik, J., and He, K. Slowfast networks for video recognition. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 6201–6210, 2019.

Ghiasi, G., Lin, T.-Y., and Le, Q. V. Dropblock: A regularization method for convolutional networks. In *Advances in Neural Information Processing Systems*, pp. 10727–10737, 2018.

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. *arXiv preprint arXiv:1603.05027*, 2016.

He, T., Zhang, Z., Zhang, H., Zhang, Z., Xie, J., and Li, M. Bag of tricks for image classification with convolutional neural networks. *arXiv preprint arXiv:1812.01187*, 2018.

Henighan, T., Kaplan, J., Katz, M., Chen, M., Hesse, C., Jackson, J., Jun, H., Brown, T. B., Dhariwal, P., Gray, S., et al. Scaling laws for autoregressive generative modeling. *arXiv preprint arXiv:2010.14701*, 2020.

Hestness, J., Narang, S., Ardalani, N., Diamos, G., Jun, H., Kianinejad, H., Patwary, M. M. A., Yang, Y., and Zhou, Y. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv: 1712.00409*, 2017.

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1314–1324, 2019.

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

Hu, H., Zhang, Z., Xie, Z., and Lin, S. Local relation networks for image recognition. *arXiv preprint arXiv:1904.11491*, 2019.

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

Huang, G., Sun, Y., Liu, Z., Sedra, D., and Weinberger, K. Q. Deep networks with stochastic depth. In *European conference on computer vision*, pp. 646–661. Springer, 2016.

Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, M. X., Chen, D., Lee, H., Ngiam, J., Le, Q. V., Wu, Y., et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *arXiv preprint arXiv:1811.06965*, 2018.

Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pp. 1–12, 2017.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Kay, W., Carreira, J., Simonyan, K., Zhang, B., Hillier, C., Vijayanarasimhan, S., Viola, F., Green, T., Back, T., Natsev, A., Suleyman, M., and Zisserman, A. The kinetics human action video dataset. *ArXiv*, abs/1705.06950, 2017.

Kolesnikov, A., Beyer, L., Zhai, X., Puigcerver, J., Yung, J., Gelly, S., and Houlsby, N. Big transfer (bit): General visual representation learning. *arXiv preprint arXiv:1912.11370*, 6(2):8, 2019.

Kornblith, S., Shlens, J., and Le, Q. V. Do better imagenet models transfer better? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2661–2671, 2019.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

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

Lee, J., Won, T., and Hong, K. Compounding the performance improvements of assembled techniques in a convolutional neural network. *arXiv preprint arXiv:2001.06268*, 2020.

Li, X., Wang, W., Hu, X., and Yang, J. Selective kernel networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 510–519, 2019.

Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., and Belongie, S. Feature pyramid networks for object detection. In *CVPR*, 2017.

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

Mahajan, D., Girshick, R., Ramanathan, V., He, K., Paluri, M., Li, Y., Bharambe, A., and Van Der Maaten, L. Exploring the limits of weakly supervised pretraining. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 181–196, 2018.

Norrie, T., Patil, N., Yoon, D. H., Kurian, G., Li, S., Laudon, J., Young, C., Jouppi, N., and Patterson, D. The design process for google's training chips: Tpuv2 and tpuv3. *IEEE Micro*, 2021.

Pham, H., Xie, Q., Dai, Z., and Le, Q. V. Meta pseudo labels. *arXiv preprint arXiv:2003.10580*, 2020.

Qian, R., Meng, T., Gong, B., Yang, M.-H., Wang, H., Belongie, S. J., and Cui, Y. Spatiotemporal contrastive video representation learning. *arXiv: Computer Vision and Pattern Recognition*, 2020.

Radosavovic, I., Kosaraju, R. P., Girshick, R., He, K., and Dollár, P. Designing network design spaces. *arXiv preprint arXiv: 2003.13678*, 2020.

Ramachandran, P., Parmar, N., Vaswani, A., Bello, I., Levskaya, A., and Shlens, J. Stand-alone self-attention in vision models. *arXiv preprint arXiv:1906.05909*, 2019.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pp. 4780–4789, 2019.

Ren, S., He, K., Girshick, R., and Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.01497*, 2015.

Ridnik, T., Lawen, H., Noy, A., and Friedman, I. Tresnet: High performance gpu-dedicated architecture. *arXiv preprint arXiv:2003.13630*, 2020.

Rosenfeld, J. S., Rosenfeld, A., Belinkov, Y., and Shavit, N. A constructive prediction of the generalization error across scales. *arXiv preprint arXiv: 1909.12673*, 2019.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. Imagenet large scale visual recognition challenge. *IJCV*, 2015.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

Shen, Z., Bello, I., Vemulapalli, R., Jia, X., and Chen, C.-H. Global self-attention networks for image recognition. *arXiv preprint arXiv: 2010.03019*, 2020.

Silberman, N., Hoiem, D., Kohli, P., and Fergus, R. Indoor segmentation and support inference from rgbd images. In *European conference on computer vision*, pp. 746–760. Springer, 2012.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 2014.

Sun, C., Shrivastava, A., Singh, S., and Gupta, A. Revisiting unreasonable effectiveness of data in deep learning era. *arXiv preprint arXiv: 1707.02968*, 2017.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

Tan, M. and Le, Q. V. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2820–2828, 2019.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. *arXiv preprint arXiv: 1706.03762*, 2017.

Xie, Q., Luong, M.-T., Hovy, E., and Le, Q. V. Self-training with noisy student improves imagenet classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10687–10698, 2020.

Xie, S., Girshick, R., Dollár, P., Tu, Z., and He, K. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1492–1500, 2017.

Yun, S., Han, D., Oh, S. J., Chun, S., Choe, J., and Yoo, Y. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 6023–6032, 2019.

Zagoruyko, S. and Komodakis, N. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Zhang, H., Cisse, M., Dauphin, Y. N., and Lopez-Paz, D. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

Zhang, H., Wu, C., Zhang, Z., Zhu, Y., Zhang, Z., Lin, H., Sun, Y., He, T., Muller, J., Manmatha, R., Li, M., and Smola, A. Resnest: Split-attention networks. *arXiv preprint arXiv:2004.08955*, 2020.

Zhang, R. Making convolutional networks shift-invariant again. *arXiv preprint arXiv:1904.11486*, 2019.

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

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710, 2018.

Zoph, B., Cubuk, E. D., Ghiasi, G., Lin, T.-Y., Shlens, J., and Le, Q. V. Learning data augmentation strategies for object detection. In *European Conference on Computer Vision*, pp. 566–583. Springer, 2020a.

Zoph, B., Ghiasi, G., Lin, T.-Y., Cui, Y., Liu, H., Cubuk, E. D., and Le, Q. V. Rethinking pre-training and self-training. *arXiv preprint arXiv:2006.06882*, 2020b.

A. Author Contributions

IB, BZ: led the research, designed and ran the scaling experiments, designed and experimented with the training strategies. **JS, TL, EC, AS, WF, XD:** advised the research, proposed experiments and helped with the writing. **AS, IB, BZ:** ran preliminary experiments using label smoothing, longer training and RandAugment. **IB:** demonstrated ResNets outperforming EfficientNets across all scales, designed the scaling strategies and the Pareto curve of models, designed/ran (semi-)supervised learning experiments and significantly contributed to the writing. **BZ:** ran the regularization studies. **WF, BZ:** did a majority of the writing. **BZ, EC:** analyzed scaling experiments and generated the scaling plots. **XD:** proposed, designed and ran the 3D video classification experiments, lead the open-sourcing. **AS:** proposed lowering the weight decay for better performance and ran preliminary experiments comparing SimCLR to supervised learning. **TL:** designed and ran the transfer learning experiments comparing to self-supervised learning.

B. Details of all ResNet-RS models in the Pareto curve

This section details all the models in the ResNet-RS Pareto curve. In Table 7, we observe that our ResNet-RS models get speedups ranging from **1.7x - 2.7x** across the EfficientNet Pareto curve on TPUs.

Model	Image Resolution	Params (M)	FLOPs (B)	V100 Latency (s)	TPUv3 Latency (ms)	Top-1
EfficientNet-B0	224	5.3	0.8	0.47	90	77.1
EfficientNet-B1	240	7.8	1.4	0.82	150	79.1
ResNet-RS-50	160	36	4.6	0.31	70	78.8
EfficientNet-B2	260	9.2	2.0	1.03	210	80.1
ResNet-RS-101	160	64	8.4	0.48 (**2.1×**)	120 (**1.8×**)	80.3
EfficientNet-B3	300	12	3.6	1.76	340	81.6
ResNet-RS-101	192	64	12	0.70	170	81.2
ResNet-RS-152	192	87	18	0.99	240	82.0
EfficientNet-B4	380	19	8.4	4.0	710	82.9
ResNet-RS-152	224	87	24	1.48 (**2.7×**)	320 (**2.2×**)	82.8
ResNet-RS-152	256	87	31	1.76 (**2.3×**)	410 (**1.7×**)	83.0
EfficientNet-B5	456	30	20	8.16	1510	83.7
ResNet-RS-200	256	93	40	2.86	570	83.4
ResNet-RS-270	256	130	54	3.76 (**2.2×**)	780 (**1.9×**)	83.8
EfficientNet-B6	528	43	38	15.7	3010	84.0
ResNet-RS-350	256	164	69	4.72 (**3.3×**)	1100 (**2.7×**)	84.0
EfficientNet-B7	600	66	74	29.9	6020	84.7
ResNet-RS-350	320	164	107	8.48	1630	84.2
ResNet-RS-420	320	192	128	10.16	2090	84.4

Table 7. **Details of ResNet-RS models in Pareto curve.** All models are trained for 350 epochs using the improvements mentioned in Section 5. The exact hyperparameters for all ResNet-RS models are in Table 8. Latencies on Tesla V100 GPUs are measured with full precision (`float32`). Latencies on TPUv3 are measured using `bfloat16` precision. All latencies are measured with an initial training batch size of 128 images, which is divided by 2 until it fits onto the accelerator.

Hyperparameters Table 8 presents the training and regularization hyperparameters used for training ResNet-RS models. We increase regularization as with model scale. Note that we have less hyperparameter setups compared to EfficientNets (Tan & Le, 2019). We perform early stopping on the `minival-set` set for the two largest models from Table 7 (ResNet-RS-350 at resolution 320 and ResNet-RS-420 at resolution 320). For every other model, we simply report the final accuracy. We present top-1 accuracies on the ImageNet `test-set` for two ResNet-RS models in Table 9. We observe no sign of overfitting.

C. ResNet-RS Training and Regularization Methods

Table 10 shows the differences in training and regularization methods between ResNets, ResNet-RS and EfficientNets. Overall we closely match EfficientNet's training setup, while making a few minor simplications: cosine learning rate

Model	Depth	Image Resolution	RandAugment Magnitude	Stochastic Depth Rate	Dropout Rate
ResNet-RS	50	160 × 160	10	0.0	0.25
ResNet-RS	101	160 × 160	10	0.0	0.25
ResNet-RS	101	192 × 192	15	0.0	0.25
ResNet-RS	152	192 × 192	15	0.0	0.25
ResNet-RS	152	224 × 224	15	0.0	0.25
ResNet-RS	152	256 × 256	15	0.0	0.25
ResNet-RS	200	256 × 256	15	0.1	0.25
ResNet-RS	270	256 × 256	15	0.1	0.25
ResNet-RS	350	256 × 256	15	0.1	0.25
ResNet-RS	350	320 × 320	15	0.1	0.4
ResNet-RS	420	320 × 320	15	0.1	0.4

Table 8. **Hyperparameters for all ResNet-RS models.** All models train for 350 epochs, use a weight decay of 4e-5, an EMA value of 0.9999 (for both weights and Batch Norm moving averages), 2 layers of RandAugment (with different magnitudes as shown above) and a label smoothing rate of 0.1. The learning rate is warmed up to a maximum value of $0.1/B$, with B the batch size, and decayed to 0 using a cosine schedule (Loshchilov & Hutter, 2016). Dropout rate means each activation after the global average pooling layers gets dropped out with probability *dropout rate*.

Model	Image Resolution	top-1 Val	top-1 Test
ResNet-RS-152	224	82.8	82.7
ResNet-RS-270	256	83.8	83.7

Table 9. **ImageNet accuracies on the validation and test splits.**

isntead of exponential decay and Momentum instead of RMSProp. Both simplifications reduce the total number of hyperparameters as **(1)** cosine decay has no hyperparameters associated with it and **(2)** Momentum has one less than RMSProp.

	ResNet (2015)	ResNet-RS (2021)	EfficientNets (2019)
Epochs Trained	90	350	350
LR Decay Schedule	Stepwise	Cosine	Exponential Decay
Optimizer	Momentum	Momentum	RMSProp
EMA of Weights		✓	✓
Label Smoothing		✓	✓
Stochastic Depth		✓	✓
RandAugment		✓	✓
Dropout on FC		✓	✓
Smaller Weight Decay		✓	✓
Squeeze-Excitation		✓	✓
Stem Modifications		✓	✓

Table 10. **Comparing training method between ResNet, ResNet-RS and EfficientNet.** ResNet (2015) refers to the ResNet originally trained in He et al. (2015).

D. ResNet-RS Architecture Details

We provide more details of the ResNet-RS architectural changes. We reiterate that ResNet-RS is a combination of: improved scaling strategies, improved training methodologies, the ResNet-D modifications (He et al., 2018) and the Squeeze-Excitation module (Hu et al., 2018). Table 11 shows the block layouts for all ResNet depths used throughout our work. ResNet-50 through ResNet-200 use the standard block configurations from He et al. (2015). ResNet-270 and onward primarily scale the number of blocks in `c3` and `c4` and we try to keep their ratio roughly constant. We empirically found that adding blocks in the lower stages limits overfitting as blocks in the lower layers have significantly less parameters, even though all blocks have the same amount of FLOPs. Figure 6 shows the ResNet-D architectural changes used in our

Model	Depth	Block Configuration
ResNet	50	[3-4-6-3]
ResNet	101	[3-4-23-3]
ResNet	152	[3-8-36-3]
ResNet	200	[3-24-36-3]
ResNet	270	[4-29-53-4]
ResNet	350	[4-36-72-4]
ResNet	420	[4-44-87-4]

Table 11. **Block configurations for all ResNet depths used in the ResNet-RS Pareto Curve.** ResNets of depths 50, 101, 152 and 200 use the standard block allocations from He et al. (2015). The different numbers represent the number of blocks in c2, c3, c4 and c5 respectively. Note that our depth scaling mainly scales the blocks in c3 and c4, which limits overfitting (due to the increase in parameters) that can occur when blocks are added to c5.

ResNet-RS models.

Block Group	Output Size	Convolution Layout	
stem	112x112	3x3, 64, s2 3x3, 64 3x3, 64	x1
c2	56x56	1x1, 64 3x3, 64 1x1, 256	x3
c3	28x28	1x1, 128 3x3, 128 1x1, 512	x4
c4	14x14	1x1, 256 3x3, 256 1x1, 1024	x23
c5	7x7	1x1, 512 3x3, 512 1x1, 2048	x3
	1x1	Avg Pool Dropout 1000-d FC	x1

Figure 6. **ResNet-RS Architecture Diagram.** Output Size assumes a 224×224 input image resolution. In the convolutional layout column $x2$ refers to the the first 3×3 convolution being applied with a stride of 2. The ResNet-RS architecture is a simple combination of Squeeze-and-Excitation and ResNet-D. The $\times$ symbol refers to how many times the blocks are repeated in the ResNet-101 architecture. These values change across depths according to the blocks layouts in Table 11.

E. Scaling Analysis Regularization and Model Details

Regularization for 350 epoch models. The dropout rates used for various filter multipliers (across all image resolutions and depths) are in Table 12. RandAugment is used with 2 layers and its magnitude is set to 10 for filter multipliers in [0.25, 0.5] or image resolution in [64, 160], 15 for image resolution in [224, 320] and 20 otherwise. We apply stochastic depth with a drop rate of 0.2 for image resolutions 224 and above. We do not apply stochastic depth filter multiplier 0.25 (or images smaller than 224). All models use a label smoothing of 0.1 and a weight decay of 4e-5. These values were set based on the preliminary experiments across various model scales on the ImageNet minival-set.

Filter Scaling	Dropout Rate
0.25	0.0
0.5	0.1
1.0	0.25
1.5	0.6
2.0	0.75

Table 12. **Dropout values for filter scaling.** Filter scaling refers to the filter scaling multiplier based on the number of filters in the original ResNet architecture.

Regularization for 10 and 100 epochs. We did not use RandAugment, Dropout, Stochastic Depth or Label Smoothing. Flips and crops were used and a weight decay of 4e-5.

Block allocation for ResNet-300 and ResNet-400. For ResNet 101 and ResNet-200 we use the block allocations decribed in Table 11. For ResNet-300, our block allocation is `[4-36-54-4]` and ResNet-400 is `[6-48-72-6]`.

F. Fine-Tuning Protocols

For fine-tuning we initialize the parameters in the ResNet backbone with a pre-trained model and randomly initialize the rest of the layers. We perform *end-to-end* fine-tuning with an extensive grid search of the combinations of learning rate and training steps to ensure each pre-trained model achieves its best fine-tuning performance. We experiment with different weight decays but do not find it making a big difference and set it to 1e-4. All models are trained with cosine learning rate for simplicity. Below we describe the dataset, evaluation metric, model architecture, and training parameters for each task.

CIFAR-100: We use standard CIFAR-100 train and test sets and report the top-1 accuracy. We resize the image resolution to 256×256. We replace the classification head in the pre-trained model with a randomly initialized linear layer that predicts 101 classes, including background. We use a batch size of 512 and search the combination of training steps from 5000 to 20000 and learning rates from 0.005 to 0.32. We find the best learning rate for SimCLR (0.16) is much higher than SimCLRv2 (0.01) and the supervised model (0.005). This trend holds for the following tasks.

PASCAL Segmentation: We use PASCAL VOC 2012 train and validation sets and report the mIoU metric. The training images are resampled into 512×512 with scale jittering [0.5, 2.0] (i.e. randomly resample image between 256×256 to 1024×1024 and crop it to 512×512). We remove the classification head and add randomly initialized FPN (Lin et al., 2017) layers. We follow the practice in (Zoph et al., 2020b) to combine P_3 to P_7 and upsample it to P_2. The segmentation head consists of 3 convolution layers after P_2 layer and a linear layer to predict 21 categories including background at each pixel location. We use a batch size of 64 and search the combination of training steps from 5000 to 20000 and learning rates from 0.005 to 0.32.

PASCAL Detection: We use PASCAL VOC 2007+2012 trainval set and VOC 2007 test set and report the AP_{50} with 11 recall points to compute average precision. The training images are resampled into 896 with scale jittering [0.5, 2.0]. We remove the classification head and add randomly initialized FPN (Lin et al., 2017) layers from P_3 to P_7. We use Faster R-CNN (Ren et al., 2015) consisting a region proposal head and a `4conv1fc` Fast R-CNN head. We use a batch size of 32 and search the combination of training steps from 5000 to 20000 and learning rates from 0.005 to 0.32.

NYU Depth: We use NYU depth v2 dataset with 47584 train and 654 validation images. We report the percentage of predicted depth values within 1.25 relative ratio compared to the ground truth. The training images are resampled into 640 with scale jittering [0.5, 2.0]. The model architecture is identical to segmentation model, except the last linear layer predicts a single depth value per pixel. We use a batch size of 64 and search the combination of training steps from 10000 to 40000 and learning rates from 0.005 to 0.32.

G. Video Classification Experimental Details

We follow the training and inference protocols in (Qian et al., 2020; Feichtenhofer et al., 2019). We train with a random 224×224 crop or its horizontal flip on the spatial domain and sample a 32-frame clip with temporal stride 2. We use a 1024 batch size, 0.8 learning rate with cosine decay and train for 200 epochs for the baseline. At inference, we use 256×256 crop size for the spatial domain and adopt the 30 views protocol (Feichtenhofer et al., 2019).

Starting from the baseline, we apply the following training methods: dropout with a rate of 0.5, 0.1 label smoothing, stochastic depth with 0.2 drop rate, EMA of weights, smaller weight decay (set to 4e-5) and a 350 epoch training schedule. For data augmentation, we use scale jittering (Qian et al., 2020) as a replacement to RandAugment. We adjust the stochastic depth rate to 0.1 when applying scale jittering to optimize performance. To implement the ResNet-D stem for the 3D ResNet, we use the same kernel configurations for the spatial domain and use temporal kernel sizes of $[5, 1, 1]$ for the three layers.

H. Profiling Setup

All latencies refer to training latencies. All models were run on TPUv3 (Jouppi et al., 2017) with `bfloat16` precision in TensorFlow 1.x. TPU latencies are measured on 8 TPUv3 cores with a batch size of 1024 (i.e. 128 per core) which is divided by 2 until it fits onto the accelerator's memory. In the cases where a smaller batch size is employed, we normalize the reported latency to the original batch size of 1024 images. For GPU profiling we use a single Tesla-V100 with `float32` precision with a starting batch size of 128, also divided by multiples of 2 if necessary.