ResNet strikes back: An improved training procedure in timm

Ross Wightman[◦] **Hugo Touvron**[⋆,†] **Hervé Jégou**[⋆]

[◦]**Independent researcher** [⋆]**Facebook AI** [†]**Sorbonne University**

Abstract

The influential Residual Networks designed by He et al. remain the gold-standard architecture in numerous scientific publications. They typically serve as the default architecture in studies, or as baselines when new architectures are proposed. Yet there has been significant progress on best practices for training neural networks since the inception of the ResNet architecture in 2015. Novel optimization & data-augmentation have increased the effectiveness of the training recipes.

In this paper, we re-evaluate the performance of the vanilla ResNet-50 when trained with a procedure that integrates such advances. We share competitive training settings and pre-trained models in the **timm** open-source library, with the hope that they will serve as better baselines for future work. For instance, with our more demanding training setting, a vanilla ResNet-50 reaches 80.4% top-1 accuracy at resolution 224×224 on ImageNet-val without extra data or distillation. We also report the performance achieved with popular models with our training procedure.

1 Introduction

In the last decade we have witnessed significant advances in image classification, as reflected by improvement on benchmarks such as the ILSVRC'2012 challenge [37] or other image classification benchmarks, which are visible on popular websites[1]. Schematically, the increase of performance reflects the maximization by the community of a problem of the form

$$\text{accuracy (model)} = f(\mathcal{A}, \mathcal{T}, \mathcal{N}),$$

where $\mathcal{A}$ is the architecture design, $\mathcal{T}$ is the training setting along with its hyper-parameters, and $\mathcal{N}$ is the measurement noise, in which we also include overfitting that typically occurs when selecting the maximum over a large set of hyper-parameters or choices of methods. Several good practices exist to mitigate $\mathcal{N}$, like measuring the standard deviation with different seeds, using a separate evaluation dataset [35] or evaluating models on transfer tasks. Putting aside $\mathcal{N}$, measuring progress on $\mathcal{A}$ or $\mathcal{T}$ poses a challenge as both $\mathcal{A}$ and $\mathcal{T}$ progress over time. When optimizing jointly over $(\mathcal{A}, \mathcal{T})$, there is no guarantee that the optimal choice $\mathcal{T}_1$ for a given architecture $\mathcal{A}_1$ remains the best for another model design $\mathcal{A}_2$. Therefore even when one compare models under the same training procedure, one may implicitly favor one model over another. One good practice to disentangle the improvement resulting from the training procedure from that of the architecture is to ensure that the baseline incorporates new "ingredients" from the literature, and to put a reasonable amount of effort in adjusting the hyper-parameters. Ideally, i.e., without resource and time constraints, one would optimally adopt the best possible training procedure for each architecture

$$\mathcal{T}^\star(\mathcal{A}) = \max_{\mathcal{T}} f(\mathcal{A}, \mathcal{T}, \mathcal{N}), \tag{1}$$

[1] See for instance `http://paperswithcode.com/task/image-classification`

35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.

but realistically this is not possible. When comparing architectures, most papers compare their results to other reported in older publications, but for which architectures were trained with potentially weaker recipes.

We are not aware of an effort specifically targeted at improving the ResNet-50 training procedure with an extensive ingredient selection and hyper-parameter search. In the literature, the performance reported on ImageNet-1k-val for this architecture ranges from 75.2% to 79.5%, depending on the paper. It is unclear whether a sufficient effort has been invested in pushing the baseline further. We want to fill this gap: in this paper, we focus on the vanilla ResNet-50 architecture[2] as described by He *et al.* [14], and we optimize the training so as to maximize the performance of this model for the original test resolution of 224×224. We solely consider the training recipe. Therefore we exclude all variations of the ResNet-50 such as SE-ResNet-50 [21] or ResNet-50-D [15], which usually improve the accuracy under the same training procedure. In summary, in this paper,

- We propose three training procedures intended to be strong baselines for a vanilla ResNet-50 used at inference resolution 224×224. The three variants correspond to different numbers of epochs (100, 300 and 600) with adjustment of hyper-parameters and ingredients.
- Our procedure includes advances from the literature as well as new proposals. Noticeably, we depart from the usual cross-entropy loss. Instead, our training solves a multi-label classification problem when using Mixup and CutMix: we minimize the binary cross entropy for each concept selected by these augmentations, assuming that all the mixed concepts are present in the synthetized image.
- We measure the stability of the accuracy over a large number of runs with different seeds, and discuss the overfitting issue by jointly comparing the performance on ImageNet-val versus the one obtained in ImageNet-V2 [35].
- We train popular architectures and re-evaluate their performance. We also discuss the necessity to optimize jointly the architecture and the training procedure: we showcase that training with the same procedure is not sufficient for comparing the merits of architectures.

Our supplemental material may interest the community. We provide ablations in Section B. Appendix A details augmentations variants that have been introduced by the **timm** library[3]. Appendix F covers alternative procedures for training a ResNet-50 that significantly differ in their ingredients from our three focal training procedures. They achieve noteworthy performance and possibly better results with different architectures and tasks.

2 Related work

Image Classification is a core problem in computer vision. It is often employed as a benchmark task to measure progress in computer vision. Pre-trained models for image classification, particularly trained on ImageNet [9], are used in a large variety of downstream tasks like detection or segmentation. Progress in image classification generally translates to progress on these tasks.

The timm library [51] has recently gained significant momentum in the scientific community as it provides implementations for numerous popular models for image classification, as well as training methods. Pre-trained weights – either adapted from originals or trained in timm with newer procedures – are included for many models. While model architectures are **timm**'s focus, it also includes implementations of many data augmentations, regularization techniques, optimizers, and learning rate schedulers that are leveraged in the training procedures described in this paper. In many cases these implementations include functionality beyond the original implementations or papers that they were based upon. We describe these additions in Appendix A.

ResNet [14] is one of the most popular image classification architectures. It was a noteworthy improvement at the time it was introduced and continues to serve as the referent architecture for some analysis [8, 56, 57], or as a baseline in papers introducing new architectures [33, 36, 52, 58].

[2] ResNet-50 V1.5 (the PyTorch [1] ResNet50) a slight adjustment of He *et al.* [14] that was made in torch7, stride was moved from 1x1 to 3x3 in bottleneck.

[3] available at `http://github.com/rwightman/pytorch-image-models/`

Training Procedure	Number of epochs	Training resolution	Training time	Peak memory by GPU (MB)	Numbers of GPU	Top-1 accuracy val	real	v2
A1	600	224 × 224	110h	22,095	4	80.4	85.7	68.7
A2	300	224 × 224	55h	22,095	4	79.8	85.4	67.9
A3	100	160 × 160	15h	11,390	4	78.1	84.5	66.1

Table 1: Training resources used for our three training procedures on V100 GPUs and corresponding accuracies at resolution 224×224 on ImageNet1k-val, -V2 and -Real. Note, the top-1 val acc. of pytorch-zoo [1] is 76.1%.

Some works have modernized the ResNet training procedure and obtained some improvement over the original model (e.g. Dollar *et al.* [10]). This allows a more direct comparison when considering new models or methods involving more elaborate training procedures than the one initially used. Nevertheless, improving the ResNet-50 baseline [6, 10, 15, 48, 49, 55] was not the main objective of these works. As a consequence and as we will see, the best performance reported so far with a ResNet-50 is still far from the maximum performance (peak or average) that one can achieve with this architecture. In this paper, our goal is to offer the best possible training procedure that we could find for the ResNet-50 based on existing ingredients and practices. We hope that it will serve as a strong baseline for subsequent works. Note, some papers have also focused on ResNet-50 training [2, 28, 50], but they have either modified the architecture or changed the resolution, which does not allow for a direct comparison to the original ResNet-50 at resolution 224×224. For instance, Lee *et al.* [28] use ResNet-D [15] with SE attention [21]. Bello *et al.* [2] also optimize ResNet without architectural changes, but they don't report competitive results for ResNet-50 at 224×224.

Training ingredients & recipes for image classification have significantly evolved since the inception of AlexNet [27]. Several trends have changed over time. Common modifications include replacing the waterfall schedule (classical division by 10 of learning rate every 30 epochs) by a longer and more progressive schedule [5, 10, 11, 42, 46, 53]. Increasing jointly the number of epochs [10, 11, 15, 42, 46] and the batch size while using mixed precision better leverages powerful GPUs. Modern procedures make use of stronger data-augmentation [7, 8, 56, 57, 59], stronger regularization [12, 22, 38], weight averaging [23, 32] and correct the train-test resolution discrepancy [49] by differentiating the train from the test resolution [5, 11]. Different losses have also been experimented with [4, 24] even if cross-entropy remains the standard. For the optimization, SGD with Nesterov momentum [39] is a common default for CNNs. RMSProp is also used for specific CNN architecture families like in Inception [40], NASNet [60], AmoebaNet [34], MobileNet [20], EfficientNet [42]. For image classifiers based on transformers [11, 46] and MLP [44, 45], AdamW [29] and Lamb [54] optimizers are popular choices.

3 Training Procedures

We offer three different training procedures with different costs and performance so as to cover different use-cases, see Table 1 for resource usage and corresponding accuracies. Our procedures target the best performance of ResNet-50 when tested at resolution 224 × 224. We have explored numerous variations with different optimizers, choice of regularization, and a reasonable amount of grid search for the hyper-parameters. See Section C in the Appendix for the exact ingredient list and parametrization. We refer the reader to Section E for control experiments on quantifying the amount of overfitting. We focus on three different operating points:

Procedure A1 aims at providing the best performance for ResNet-50. It is therefore the longest in terms of epochs (600) and training time (4.6 days on one node with 4 V100 32GB GPUs).

Procedure A2 is a 300 epochs schedule that is comparable to several modern procedures like DeiT, except with a larger batch size of 2048 and other choices introduced for all our recipes.

Procedure A3 aims at outperforming the original ResNet-50 procedure with a short schedule of 100 epochs and a batch size 2048. It can be trained in 15h on 4 V100 16GB GPUs and could be a good setting for exploratory research or studies.

Table 1 summarizes the main characteristics of our training procedure. We detail our ingredients in Section C. Section F gives alternative training procedures that may serve as interesting choices when considering other models.

Table 2: **Performance of models trained with A1 training procedure.** We measure peak memory and throughput on one GPU V100 32GB with batch size 128, FP16 precision and test resolution from Table 3. The throughput is indicative, since it depends on the GPU hardware, the software that runs the models, and other factors like the adjustment of batch size (constant in this table).

	# params	FLOPs	Throughput	Peak mem	Top-1	Real	V2
Architecture	$\times 10^6$	$\times 10^9$	(im/s)	(MB)	Acc.	Acc.	Acc.
ResNet-18 [14]	11.7	1.8	7960.5	588	71.5	79.4	59.4
ResNet-34 [14]	21.8	3.7	4862.6	642	76.4	83.4	65.1
ResNet-50 [14]	25.6	4.1	2536.6	1,155	80.4	85.7	68.7
ResNet-101 [14]	44.5	7.9	1547.9	1,264	81.5	86.3	70.3
ResNet-152 [14]	60.2	11.6	1094.0	1,355	82.0	86.4	70.6
RegNetY-4GF [33]	20.6	4.0	1690.6	1,585	81.5	86.7	70.7
RegNetY-8GF [33]	39.2	8.1	1122.3	2,139	82.2	86.7	71.1
RegNetY-16GF [33]	83.6	16.0	694.1	3,052	82.0	86.4	71.2
RegNetY-32GF [33]	145.0	32.4	431.5	3,366	82.5	86.6	71.7
SE-ResNet-50 [21]	28.1	4.1	2174.8	1,193	80.0	85.8	68.8
SENet-154 [21]	115.1	20.9	511.5	2,414	81.7	86.0	71.2
ResNet-50-D [15]	25.6	4.4	2418.8	1,205	80.7	85.9	68.9
ResNeXt-50-32x4d [52]	25.0	4.3	1727.5	1,247	80.5	85.5	68.4
EfficientNet-B0 [42]	5.3	0.4	3701.5	932	77.0	83.8	65.0
EfficientNet-B1 [42]	7.8	0.7	2365.2	1,077	79.2	85.3	67.7
EfficientNet-B2 [42]	9.2	1.0	1786.8	1,318	80.4	86.0	69.3
EfficientNet-B3 [42]	12.0	1.8	1082.4	2,447	81.4	86.7	70.4
EfficientNet-B4 [42]	19.0	4.2	561.3	5,058	81.6	85.9	70.8
ViT-Ti [46]	5.7	1.3	3497.7	346	74.7	82.1	62.4
ViT-S [46]	22.0	4.6	1762.3	682	80.6	85.6	69.4
ViT-B [11]	86.6	17.6	771.0	1,544	80.4	84.8	69.4
timm [51] specific architectures							
ECA-ResNet50-T	25.6	4.4	2139.7	1,155	81.3	86.1	69.9
EfficientNetV2-rw-S [43]	23.9	8.8	823.1	2,339	80.6	84.8	69.2
EfficientNetV2-rw-M [43]	53.2	18.5	456.8	2,916	82.3	87.1	71.7
ECA-Resnet269-D	102.1	70.6	168.1	4,134	83.3	86.9	71.9

4 Experiments

In this section we first compare our training procedure to existing ones and evaluate them with different architectures. Importantly, we discuss in Section E the significance of our results with experiments that aim at (1) quantifying the sensitivity of the performance to random factors; (2) evaluating the overfitting by measuring on a different test set.

Comparison of training procedures for ResNet-50 To the best of our knowledge, our procedure A1 surpasses the current state of the art on ImageNet with a vanilla ResNet-50 architecture at resolution 224×224. Our other procedures A2 and A3 achieve lower but still high performance with less resources.

Performance comparison with other architectures. In Table 3 we report the performance obtained when training different architectures with our training procedures. This allows us to see how well they generalize to other models. Or procedures improves the performance of several models over results reported in the literature, in particular older ones and/or those most comparable to ResNet-50 in terms of architecture and size. In some cases like ViT-B, we observe that A2 is better than A1, which suggests that the hyper-parameters are not adapted to longer schedules (typically requiring more regularization). For instance, the A2 training recipe achieves 81.8% top-1 accuracy when training a ResNet-152, but by increasing a bit the regularization we improved it to 82.4% at resolution 224×224, which translates to 82.7% when evaluated at resolution 256×256.

In Table 3, we compare the performance and resources associated with our 3 training recipes when using them to train other architectures. We complement these results with Table 2, where we

Table 3: Comparison on ImageNet classification between other architectures trained with our ResNet-50 optimized training procedure **without any hyper-parameters adaptation**. In particular, our procedure must be adapted for deeper/larger models, which benefit from more regularization. For the training cost we report the training time (time) in hours, the number of GPU used (#GPU) and the peak memory by GPU (Pmem) in GB. For A1 and A2, we adopt the same training and test resolution as in the original publication introducing the architecture. For A3 we use a smaller training resolution to reduce the compute-time. †: torchvision [1] results. *: DeiT [46] results.

	A1-A2-org.		A3		Cost							ImageNet-1k-val			
	train	test	train	test	A1	A2	A1-A2		A3			A1	A2	A3	org.
↓ Architecture	res.	res.	res.	res.	time (hour)		# GPU	Pmem	time	# GPU	Pmem	Accuracy(%)			
ResNet-18 [14]†	224	224	160	224	186	93	2	12.5	28	2	6.5	71.5	70.6	68.2	69.8
ResNet-34 [14]†	224	224	160	224	186	93	2	17.5	27	2	9.0	76.4	75.5	73.0	73.3
ResNet-50 [14]†	224	224	160	224	110	55	4	22.0	15	4	11.4	80.4	79.8	78.1	76.1
ResNet-101 [14]†	224	224	160	224	74	37	8	16.3	8	8	8.5	81.5	81.3	79.8	77.4
ResNet-152 [14]†	224	224	160	224	92	46	8	22.5	9	8	11.8	82.0	81.8	80.6	78.3
RegNetY-4GF [33]	224	224	160	224	130	65	4	27.1	15	4	13.9	81.5	81.3	79.0	79.4
RegNetY-8GF [33]	224	224	160	224	106	53	8	19.8	10	8	10.3	82.2	82.1	81.1	79.9
RegNetY-16GF [33]	224	224	160	224	150	75	8	25.6	13	8	13.4	82.0	82.2	81.7	80.4
RegNetY-32GF [33]	224	224	160	224	120	60	16	17.6	12	16	9.4	82.5	82.4	82.6	81.0
SE-ResNet-50 [21]	224	224	160	224	102	51	4	27.6	16	4	14.2	80.0	80.1	77.0	76.7
SENet-154 [21]	224	224	160	224	110	55	16	23.3	12	16	12.2	81.7	81.8	81.9	81.3
ResNet-50-D [15]	224	224	160	224	100	50	4	23.9	14	4	12.3	80.7	80.2	78.7	79.3
ResNeXt-50-32x4d [52]†	224	224	160	224	80	40	8	14.3	15	4	14.6	80.5	80.4	79.2	77.6
EfficientNet-B0 [42]	224	224	160	224	110	55	4	22.1	15	4	11.4	77.0	76.8	73.0	77.1
EfficientNet-B1 [42]	240	240	160	224	62	31	8	17.9	8	8	7.9	79.2	79.4	74.9	79.1
EfficientNet-B2 [42]	260	260	192	256	76	38	8	22.8	9	8	11.9	80.4	80.1	77.5	80.1
EfficientNet-B3 [42]	300	300	224	288	62	31	16	19.5	6	16	10.1	81.4	81.4	79.2	81.6
EfficientNet-B4 [42]	380	380	320	380	64	32	32	20.4	8	32	14.3	81.6	82.4	81.2	82.9
ViT-Ti [46]*	224	224	160	224	98	49	4	16.3	14	4	7.0	74.7	74.1	66.7	72.2
ViT-S [46]*	224	224	160	224	68	34	8	16.1	8	8	7.0	80.6	79.6	73.8	79.8
ViT-B [11]*	224	224	160	224	66	33	16	16.4	5	16	7.3	80.4	79.8	76.0	81.8
timm [51] specific architectures															
ECA-ResNet-50-T	224	224	160	224	112	56	4	29.3	15	4	15.0	81.3	80.9	79.6	_
EfficientNetV2-rw-S [43]	288	384	224	288	52	26	16	16.6	7	16	10.1	82.3	82.9	80.9	83.8
EfficientNetV2-rw-M [43]	320	384	256	352	64	32	32	18.5	9	32	12.1	80.6	81.9	82.3	84.8
ECA-ResNet-269-D	320	416	256	320	108	54	32	27.4	11	32	17.8	83.3	83.9	83.3	85.0

additionally include the performance and efficiency on ImageNet-1k, ImageNet-V2 and ImageNet-Real for different architectures trained with our best performing A1 training recipes.

5 Conclusion

In this paper we have proposed new training procedures for a vanilla ResNet-50. We have integrated new ingredients and put a significant effort in exploring diverse procedures under different resource constraints. As a result, we have established the new state of the art for training this gold-standard model. We have two other procedures to train strong ResNet-50 with less compute power. Nevertheless, we do not claim that our procedures are universal, quite the opposite: the architecture and training should be optimized jointly. Our procedure is not ideal for training other models: while, on some models, our training recipes lead to excellent results outperforming those reported in the literature, they exhibit suboptimal performance on others, typically for deeper architectures that require more regularization.

Acknowledgments & feedback

Ross Wightman thanks NVIDIA for the donation of a V100 DGX Station and Google's TPU Research Cloud (TRC) for Cloud TPUs used in this research. All authors thank Mike Rabbat and Jakob Verbeek for their feedback. We welcome feedback regarding these or other noteworthy procedures via the **timm** GitHub Discussions[4] .

[4] `https://github.com/rwightman/pytorch-image-models/discussions`

References

[1] Pytorch. `https://pytorch.org/vision/stable/index.html`. Accessed: 2021-08-01.

[2] Irwan Bello, W. Fedus, Xianzhi Du, E. D. Cubuk, A. Srinivas, Tsung-Yi Lin, Jonathon Shlens, and Barret Zoph. Revisiting ResNets: Improved training and scaling strategies. *arXiv preprint arXiv:2103.07579*, 2021.

[3] Maxim Berman, Hervé Jégou, Andrea Vedaldi, Iasonas Kokkinos, and Matthijs Douze. Multigrain: a unified image embedding for classes and instances. *arXiv preprint arXiv:1902.05509*, 2019.

[4] Lucas Beyer, Olivier J. Hénaff, Alexander Kolesnikov, Xiaohua Zhai, and Aaron van den Oord. Are we done with ImageNet? *arXiv preprint arXiv:2006.07159*, 2020.

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

[6] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. *arXiv preprint arXiv:2104.14294*, 2021.

[7] Ekin Dogus Cubuk, Barret Zoph, Dandelion Mané, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018.

[8] Ekin D. Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. RandAugment: Practical automated data augmentation with a reduced search space. *arXiv preprint arXiv:1909.13719*, 2019.

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[10] Piotr Dollár, Mannat Singh, and Ross B. Girshick. Fast and accurate model scaling. In *Conference on Computer Vision and Pattern Recognition*, 2021.

[11] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[12] Pierre Foret, Ariel Kleiner, H. Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. *arXiv preprint arXiv:2010.01412*, 2021.

[13] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition*, 2016.

[15] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In *Conference on Computer Vision and Pattern Recognition*, 2019.

[16] Dan Hendrycks, Norman Mu, E. D. Cubuk, Barret Zoph, J. Gilmer, and Balaji Lakshminarayanan. Augmix: A simple data processing method to improve robustness and uncertainty. *arXiv preprint arXiv:1912.02781*, 2020.

[17] Byeongho Heo, Sanghyuk Chun, Seong Joon Oh, Dongyoon Han, Sangdoo Yun, Youngjung Uh, and Jung-Woo Ha. Adamp: Slowing down the weight norm increase in momentum-based optimizers. *arXiv preprint arXiv:2006.08217*, 2020.

[18] Elad Hoffer, Tal Ben-Nun, Itay Hubara, Niv Giladi, Torsten Hoefler, and Daniel Soudry. Augment your batch: Improving generalization through instance repetition. In *Conference on Computer Vision and Pattern Recognition*, 2020.

[19] Grant Van Horn, Oisin Mac Aodha, Yang Song, Alexander Shepard, Hartwig Adam, Pietro Perona, and Serge J. Belongie. The iNaturalist species classification and detection dataset. *arXiv preprint arXiv:1707.06642*, 2017.

[20] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, M. Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[21] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*, 2017.

[22] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. In *European Conference on Computer Vision*, 2016.

[23] Pavel Izmailov, Dmitrii Podoprikhin, T. Garipov, D. Vetrov, and A. G. Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

[24] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. *arXiv preprint arXiv:2004.11362*, 2020.

[25] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *IEEE Workshop on 3D Representation and Recognition*, 2013.

[26] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, CIFAR, 2009.

[27] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In *NeurIPS*, 2012.

[28] Jung kyu Lee, Taeryun Won, and Kiho Hong. Compounding the performance improvements of assembled techniques in a convolutional neural network. *arXiv preprint arXiv:2001.06268*, 2020.

[29] I. Loshchilov and F. Hutter. Fixing weight decay regularization in adam. *arXiv preprint arXiv:1711.05101*, 2017.

[30] M-E. Nilsback and A. Zisserman. Automated flower classification over a large number of classes. In *Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing*, 2008.

[31] David Picard. `torch.manual_seed(3407)` is all you need: On the influence of random seeds in deep learning architectures for computer vision. *arXiv preprint arXiv:2109.08203*, sep 2021.

[32] B. Polyak and A. Juditsky. Acceleration of stochastic approximation by averaging. *Siam Journal on Control and Optimization*, 1992.

[33] Ilija Radosavovic, Raj Prateek Kosaraju, Ross B. Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. *Conference on Computer Vision and Pattern Recognition*, 2020.

[34] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*, 2018.

[35] B. Recht, Rebecca Roelofs, L. Schmidt, and V. Shankar. Do ImageNet classifiers generalize to ImageNet? In *International Conference on Machine Learning*, 2019.

[36] T. Ridnik, Hussam Lawen, Asaf Noy, and Itamar Friedman. Tresnet: High performance gpu-dedicated architecture. *arXiv preprint arXiv:2003.13630*, 2020.

[37] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[38] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. In *Journal of Machine Learning Research*, 2014.

[39] Ilya Sutskever, James Martens, George E. Dahl, and Geoffrey E. Hinton. On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning*, 2013.

[40] Christian Szegedy, V. Vanhoucke, S. Ioffe, Jon Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. *Conference on Computer Vision and Pattern Recognition*, 2016.

[41] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Conference on Computer Vision and Pattern Recognition*, 2015.

[42] Mingxing Tan and Quoc V. Le. EfficientNet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

[43] Mingxing Tan and Quoc V. Le. Efficientnetv2: Smaller models and faster training. In *International Conference on Machine Learning*, 2021.

[44] Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. MLP-Mixer: An all-MLP architecture for vision. *arXiv preprint arXiv:2105.01601*, 2021.

[45] Hugo Touvron, Piotr Bojanowski, Mathilde Caron, M. Cord, Alaaeldin El-Nouby, Edouard Grave, Armand Joulin, Gabriel Synnaeve, Jakob Verbeek, and Hervé Jégou. ResMLP: feedforward networks for image classification with data-efficient training. *arXiv preprint arXiv:2105.03404*, 2021.

[46] Hugo Touvron, M. Cord, M. Douze, F. Massa, Alexandre Sablayrolles, and H. Jégou. Training data-efficient image transformers & distillation through attention. *International Conference on Machine Learning*, 2021.

[47] Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going deeper with image transformers. *International Conference on Computer Vision*, 2021.

[48] Hugo Touvron, Alexandre Sablayrolles, M. Douze, M. Cord, and H. Jégou. Grafit: Learning fine-grained image representations with coarse labels. *International Conference on Computer Vision*, 2021.

[49] Hugo Touvron, Andrea Vedaldi, Matthijs Douze, and Herve Jegou. Fixing the train-test resolution discrepancy. *Neurips*, 2019.

[50] Ashish Vaswani, Prajit Ramachandran, Aravind Srinivas, Niki Parmar, Blake Hechtman, and Jonathon Shlens. Scaling local self-attention for parameter efficient visual backbones. *arXiv preprint arXiv:2103.12731*, 2021.

[51] Ross Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019.

[52] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Conference on Computer Vision and Pattern Recognition*, 2017.

[53] Ismet Zeki Yalniz, Hervé Jégou, Kan Chen, Manohar Paluri, and Dhruv Kumar Mahajan. Billion-scale semi-supervised learning for image classification. *arXiv preprint arXiv:1905.00546*, 2019.

[54] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training BERT in 76 minutes. In *International Conference on Learning Representations*, 2020.

[55] L. Yuan, Y. Chen, Tao Wang, Weihao Yu, Yujun Shi, F. Tay, Jiashi Feng, and S. Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet. *arXiv preprint arXiv:2101.11986*, 2021.

[56] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. CutMix: Regularization strategy to train strong classifiers with localizable features. *arXiv preprint arXiv:1905.04899*, 2019.

[57] Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[58] Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Zhi Zhang, Haibin Lin, Yue Sun, Tong He, Jonas Muller, R. Manmatha, Mu Li, and Alexander Smola. Resnest: Split-attention networks. *arXiv preprint arXiv:2004.08955*, 2020.

[59] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In *AAAI*, 2020.

[60] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *Conference on Computer Vision and Pattern Recognition*, 2018.

Supplementary material

This supplemental material provides complementary results referred in the main document, noticeably we include a presentation of the augmentation and regularization specificity in **timm** and some alternative training procedures.

A Augmentations and Regularization in timm [51]

The **timm** library includes a variety of image augmentations, regularization techniques, optimizers, and learning rate schedulers that can be used to produce leading results on ImageNet classification and other 2D image tasks. Many **timm** training components have modifications and improvements from original implementations or papers describing them. One should be aware of these changes if using them.

Data Augmentation in **timm** includes implementations of RandAugment [8], AutoAugment [7], AugMix [16], Random Erasing [59], and an integrated implementation of Mixup [57] and CutMix [56]. The base for all augmentations is typically Random Resized Crop with horizontal flipping.

RandAugment is the most used of the AA (AutoAugment) variants in **timm** – it also contains the most significant additions from the original paper and Tensorflow based implementations – so we will focus on that implementation. The original RandAugment specification has two hyper-parameters, M and N; where M is the distortion magnitude and N is the number of distortions uniformly sampled and applied per-image. The goal of RandAugment was that both M and N be human interpretable. However, that ended up not being the case for M. The scales of several augmentations were backwards or not monotonically increasing over the range such that increasing M does not increase the strength of all augmentations.

This is most visible for image enhancement blending operations (color, contrast, brightness, sharpness) where the argument value defines the behavior as follows:

`0`	selects the degenerate image
`0.-1.0`	interpolates between the degenerate and original image
`1.0`	returns original image
$>$ `1.0`	extrapolates the original image way from the degenerate

Taking sharpness as an example, magnitudes of M0, M5, and M10 are mapped in the original implementation to produce strong blurring (0.1), no-change (1.0), or strong sharpening (1.9) respectively.

The implementation in **timm** attempts to improve this situation by adding an 'increasing' mode (always enabled for recipes in this paper) where all augmentation strengths increase with magnitude; solarize and posterize increase with M (instead of decrease), and interpolation vs extrapolation for the blending operations is randomly chosen with a strength that increases with M. This makes increasing M more intuitive and allows an additional hyper-parameter to work well: **timm** adds a MSTD parameter which adds gaussian noise with the specified standard deviation to the M value per distortion application. Additionally, if MSTD is set to '-inf', M is uniformly sampled from 0-M for each distortion. Without correcting the scales, one would often end up with completely empty or heavily inverted images in ranges of M that are supposed to be low in strength.

Care was taken in **timm**'s RandAugment to reduce impact on image mean, the normalization parameters can be passed as a parameter such that all augmentations that may introduce border pixels can use the specified mean instead of defaulting to 0 or a hard-coded tuple as in other implementations. And lastly, Cutout is excluded by default to favour separate use of **timm**'s Random Erasing implementation which has less impact on mean and standard deviation of the augmented images.

Random Erasing is another commonly used **timm** augmentation with modifications from the original paper. The implementation in timm follows the original but allows 'erasing' image regions

loss	LR	WD	RA	A2
BCE	2×10^{-3}	0.02	✓	78.24
BCE	2×10^{-3}	0.03	✓	78.47
BCE	3×10^{-3}	0.02	✓	79.16
BCE	3×10^{-3}	0.03	✓	79.28
BCE	5×10^{-3}	0.01	✓	79.66
BCE	5×10^{-3}	0.02	✓	79.85
BCE	5×10^{-3}	0.03	✓	79.73
BCE	8×10^{-3}	0.02	✓	79.63
BCE	3×10^{-3}	0.02	✗	78.74
BCE	5×10^{-3}	0.02	✗	79.57
BCE	5×10^{-3}	0.03	✗	79.58

loss	LR	WD	RA	A2
CE	2×10^{-3}	0.02	✓	77.37
CE	3×10^{-3}	0.02	✓	78.22
CE	5×10^{-3}	0.02	✓	79.18
CE	5×10^{-3}	0.03	✓	79.23
CE	5×10^{-3}	0.05	✓	79.31
CE	8×10^{-3}	0.03	✓	79.12
CE	3×10^{-3}	0.02	✗	77.71
CE	5×10^{-3}	0.01	✗	78.93
CE	5×10^{-3}	0.02	✗	79.00
CE	5×10^{-3}	0.03	✗	78.62
CE	8×10^{-3}	0.02	✗	78.72

Table 4: Main ablation table with A2 procedure. We compare BCE vs CE, including repeated augmentation or not, and vary the learning rate LR and weight decay WD in ranges that our exploration phase has identified as being the most adapted. All results are reported with Seed 0 and therefore all the ResNet-50 are initialized with the same weights when the training starts. The highlighted row corresponds to our A2 procedure.

with per-pixel gaussian noise (mean 0, std 1.0) instead of a uniform random or constant color (black or image mean) per-region. When applied to images at the recommended location in the augmentation pipeline – after images have been normalized (standardized) – this maintains image statistics and allows better results with stronger application of the augmentation. A count parameter was also added to **timm**'s Random Erasing such that multiple regions can be erased per-image.

Mixup and CutMix are cleanly integrated in **timm** in a manner not common in other implementations. Both can be enabled at the same time with a variety of different mixing strategies:

batchwise CutMix vs Mixup selection, lambda and CutMix region sampling performed per-batch;

pairwise mixing, lambda, and region sampling performed per mixing sample pair within batch;

elementwise mixing, lambda, and region sampling performed per sample within batch;

half the same as elementwise but one of each mixing pair is discarded so that each sample is seen once per epoch.

The default is to use either CutMix or Mixup with probability of 0.5 per-batch if both are enabled – this is the case for all mentioned training procedures in this paper.

Regularization in **timm** is standard. It allows use of similar regularization for many of the included models. Weight decay is available via either native PyTorch or **timm** optimizers. The ability to enable pre-classifier dropout is included in all model architectures. Stochastic-Depth has been added as an option to many of the most popular model architectures (via a layer named DropPath). Label-smoothing is included via a cross-entropy loss function and possible to use in combination with the label manipulation of CutMix and Mixup.

B Ablations

In this section we provide a few ablations of hyper-parameters or selection of ingredients. Some modifications are difficult to ablate individually since they require to re-adjust several other parameters to work properly. This is the case in particular of the optimizer, which strongly interacts with other choices and hyper-parameters. In Appendix F we provide alternative training procedures that we have developed for other optimizers: RMSProp, SGD and AdamP.

Main ingredients and hyper-parameters. In Table 4 we provide an ablation of major ingredients. We focus on the intermediate A2 training procedure as it is a good compromise between compute-cost and accuracy. We make the following observations:

- *Learning rate and Weight Decay.* The learning rate has an important effect on performance. The higher value 5.10^{-3} presented in this table leads to the best performance. However

drop-factor	A1	A2	A3
0	79.94	79.79	78.06
0.05	80.38	79.85	77.57
0.1	80.12	79.62	77.32

Table 5: Ablation of stochastic Depth for our training procedures. In blue, we highlight the results corresponding to the default selection for each procedure, see Table 9.

increasing it further increases the risk of divergence. We have typically set the weight decay in the range [0.02, 0.03] that we have identified in our preliminary exploration. This parameter is a bit sensitive and can interact with other forms of regularization. In some cases we observe significant differences between 0.02 and 0.03.

- *Loss: Binary Cross Entropy versus Cross Entropy.* In this ablation, moving back from how we use BCE to the vanilla CE loss significantly reduces the performance. As discussed in our main paper, we use the flexibility of BCE to regard Mixup/Cutmix as activating a multi-class 1-vs-all classification problem as discussed in our paper, as opposed to the choice of enforcing probabilities that sum to 1. If we enforce probabilities to sum to 1 as implicitly done with cross-entropy, we obtain a slightly lower accuracy as reported in Table 6. By itself, i.e., with the same target, we do not conclude that BCE is necessarily better than CE. But it is with that loss that we reach the configuration with the highest accuracy overall.
- *Repeated augmentation* is providing a small boost in this ablation. This augmentation has some complex interaction with other hyper-parameters, and is not well understood in our opinion. In some cases we observed that it was neutral or detrimental, for instance with shortest schedules (A3 procedure), or in Table 6 with higher values of the Mixup parameter. Overall, it was best to include this ingredient in our most accurate procedures A1 and A2.

Stochastic Depth. We have included stochastic depth in the A1 and A2 training procedures. In Table 5 we observe that it provides an improvement for A2 compared to setting the drop-rate to 0 (i.e., no stochastic depth), not for A3.

Augmentation. Table 6 evidences the role of augmentations when we modify a few parameters (of Mixup and RandAugment): each modification that we have done has some impact on the measured score. While it would be unrealistic (and not ecological) to ensure that all our choices are statistically significant, one can observe that all modifications in this table decrease the top-1 accuracy below the average performance (79.72% – std 0.1) that we report over 100 seeds in Figure 2.

mixup	Rep. aug.	RandA	label smooth.	stoch. depth	BCE target	top-1 acc.
0.1	✓	7	✗	0.05	✓	79.85
0.2	✗					79.62
0.2		6				79.61
0.05						79.57
					✗	79.57

Table 6: Ablation of some data-augmentation choices for our training procedure A2 on Imagenet-val, all computed with "Seed 0". The first row contains our default choices, see Table 9 for the full set of hyper-parameters. Each other row corresponds to an ablation for which we have changed only one or two hyper-parameters or ingredient. Activating "BCE target" is our default. It refers to our choice to regard Mixup/Cutmix as activating a multi-class classification 1-vs-all problem as discussed in our paper. Not using it means that we also use BCE, but we enforce the probabilities of the concepts sum to 1 as with the regular cross-entropy loss.

Crop-ratio. We evaluate the influence of the crop-ratio used at inference time. The one most commonly adopted in the literature is 0.875. Recently researchers have considered larger values for this parameter, noticeably for vision transformers after significant gains were reported by the author of the **timm** library with these models. Table 7 provides an analysis as a function of this parameter.

	A1			A2			A3		
crop-ratio	mean (*std*)	max – min	seed 0	mean (*std*)	max – min	seed 0	mean (*std*)	max – min	seed 0
0.875	80.19 (*0.11*)	80.35 – 79.95	80.14	79.67 (*0.08*)	79.91 – 79.59	79.91	77.69 (*0.10*)	77.85 – 77.48	77.69
0.9	80.25 (*0.08*)	80.39 – 80.13	80.25	79.73 (*0.09*)	79.89 – 79.56	79.75	77.86 (*0.09*)	78.01 – 77.62	77.83
0.95	80.29 (*0.11*)	**80.50** – 80.08	80.38	79.68 (*0.09*)	79.85 – 79.57	79.85	78.00 (*0.09*)	78.09 – 77.83	78.06
1.0	80.20 (*0.13*)	80.41 – 80.06	80.19	79.58 (*0.13*)	79.88 – 79.32	79.88	78.02 (*0.10*)	78.16 – 77.83	77.93

Table 7: Ablation of the crop-ratio when training with A1. We compute the Imagenet-val top-1 accuracy as a function of this parameter for 10 different seeds, for ResNet50 trained with our procedures. Our selection of 0.95 was based on Seed 0 in early experiments. It is comparable but not statistically better than the standard 0.875. Note that we have one A1 seed that leads to a top 80.54% top-1 accuracy at crop-ratio 0.9. We regard it as being overfit and therefore we do not recommend to report this number.

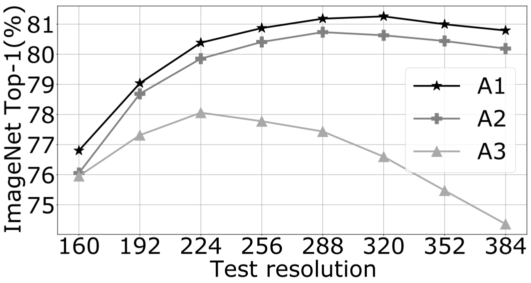


Figure 1: We compare ImageNet Top-1 accuracy according to the test resolution for our three training procedures A1, A2 and A3 with ResNet-50 architecture. Our training procedure and models also benefit from the FixRes effect [49]: the performance increases when using a larger image at test time for the procedures A1 and A2. This observation is not true for A3, which is expected since this procedure was already relying on feeding smaller images at train time, so as to maximize the accuracy at test resolution 224 × 224.

Evaluation at other resolutions. While we primarily focus on the performance when inferring at resolution 224 × 224, we also evaluate our models when feeding images at larger resolutions. We report these results in Figure 1, where we see that the models trained with A1 and A2 have a better performance when used at higher resolutions.

Training with large batches. The motivation for increasing the batch size is usually to accelerate the training with a higher level of parallelism. In terms of total compute there is no benefit. As discussed in prior works [13, 54], the hyper-parameters have to be adjusted for the training to be effective. For instance the usual recommendation with SGD [13] is to multiply the learning rate by the size of the batch. In the case of the our training procedure, we approximately follow the recommendation made in the Lamb paper [54]: we adopt a square root scaling rule for the learning rate and linearly increase the number of warmup epochs. In Table 8, we give the details of the configurations that we tested with a higher parallelism, and we compare it to our baseline with batch size of 2048.

C Detailed training recipes: A1–A3

Loss: multi-label classification objective. Mixup and CutMix augmentation synthesize an image from several images having in most cases different labels. By using cross-entropy, the output is implicitly treated as a probability of presence of each of the mixed concepts. In our training, we assume instead that these concepts are all present, and treat the classification as a multi-label classification problem (1-vs-all). For this purpose, we adopt the binary cross-entropy (BCE) loss instead of the typical cross-entropy (CE). This loss is consistent with the Mixup and CutMix data augmentation: The targets are defined for each class to 1 if the class is selected by Mixup or Cutmix, independent of other classes. Over the best settings that we have explored, BCE slightly outperforms cross-entropy in their best respective configurations. We point out that Beyer et al. [4] previously adopted BCE with the motivation to produce multiple non-exclusive labels, and obtained excellent results with it. But to the best of our knowledge they did not use it with CutMix or Mixup as we propose to do.

Training Procedure	Number of V100 GPUs	batch size	batch size per GPU	Learning Rate	Warmup #epochs	Accuracy (val %)	Duration
A2	4 (1×4)	2048	512	0.005	5	79.85	55h
A2	16 (2×8)	2048	128	0.005	5	79.89	14h35
A2	64 (8×8)	2048	32	0.005	5	79.69	8h03
A2	16 (2×8)	8192	512	0.01	20	79.54	18h06
A2	64 (8×8)	8192	128	0.01	20	79.58	5h17
A3	4 (1×4)	2048	512	0.008	5	78.06	15h
A3	16 (2×8)	2048	128	0.008	5	78.18	4h13
A3	64 (8×8)	2048	32	0.008	5	77.51	2h28
A3	16 (2×8)	8192	512	0.02	20	77.42	4h50
A3	64 (8×8)	8192	128	0.02	20	77.29	1h17

Table 8: Training with batches of size 8192: we adapt the A2 and A3 procedures by adjusting the learning rate and warmup duration. We train on several nodes (2×8 GPUs means two nodes with 8 V100 each). We observe a small loss of accuracy with larger batch sizes, that is more significant with A3. The benefit of large batch size only appears with a large number of machines, when the batch size per GPU makes the computation inefficient. Note, changing the number of nodes has a similar effect on accuracy as changing the seed for a given batch size.

In our experiments done with a BCE multi-label loss, setting all mixed concepts with a target to 1 (or $1 - \varepsilon$) is more effective than considering a distribution of concepts that sum to 1. Conceptually we believe it is more aligned with what Mixup and CutMix are actually doing: it is likely that a human could recognize each of two mixed concepts.

Data-Augmentation. We adopt the following combination of data augmentations: on top of standard Random Resized Crop (RRC) and horizontal flip (commonly used since GoogleNet [41]), we apply **timm** [51] variants of RandAugment [8], Mixup [57], and CutMix [56]. This combination was used for instance in DeiT [46]. Many of the model weights in **timm** have also been trained with RandAugment and Mixup, but with Random Erasing [59] and increased regularization instead of CutMix. We refer the reader to Appendix A for more details about the variants offered in **timm**.

Regularization. Across our three training procedures, regularization differs the most. In addition to adapting the weight decay, Repeated-Augmentation [3, 18] (RA) and stochastic-Depth [22]. We use more regularization for longer training schedules. We have adopted label smoothing[5] only for A1. For instance we use stronger Mixup only for A1. Both RA and stochastic depth tend to improve the results at convergence, but they slow down the training in the early stages as reported by Berman et al. [3] for RA. For short schedules they are therefore less effective or even detrimental, which is why we adopt them only with A1 and A2. Note that for other architectures, or larger ResNets, it is beneficial to add additional regularization, therefore one would have to adapt the corresponding hyper-parameters for such architectures. For instance, for a ResNet-152 the performance increases from 81.8% to 82.4% on Imagenet-val by putting more of RandAugment, mixup and stochastic depth regularization on top of A2 recipe. At resolution 256×256 this model obtains 82.7%, which is above the accuracy (82.2%) reported by Bello *et al.* [2] for a ResNet-200 before architectural changes (Table 1 in their paper).

Optimization. Since AlexNet, the most used optimizer to train convnets is SGD. In contrast transformers [11, 47] and MLP [44, 45] use AdamW [29] or LAMB optimizer. Dosovitskiy et al. [11] report similar performance between AdamW [29] and SGD for ResNet-50. This concurs with our observations for intermediate batch sizes (e.g., 512). We use larger batches, e.g., 2048. When combined with repeated augmentation and the binary cross entropy loss, we found that LAMB [54] makes it easier to consistently achieve good results. We found it difficult to achieve convergence when using both SGD and BCE. We therefore focus on LAMB with cosine schedule as the default optimizer for training our ResNet-50. Alternative training procedures using different optimizer, loss, augmentation, and regularization combinations can be found in Appendix F.

[5] We have compared smoothing with no smoothing over 10 runs instead of relying on a single seed for the ablation. The difference between both is not significant: we obtain 80.31±*0.11* without smoothing, and 80.29±*0.12* when setting the smoothing parameter to $\varepsilon = 0.1$ (with $1 - \varepsilon$ for positive targets and $\varepsilon/\mathrm{nb\_classes}$ for negative ones). We have kept the smoothing in A1.

Table 9: Ingredients and hyper-parameters used for ResNet-50 training in different papers. We compare existing training procedures with ours.

	Previous approaches					Ours		
Procedure →	ResNet	PyTorch	FixRes	DeiT	FAMS (×4)	A1	A2	A3
Reference	[14]	[1]	[49]	[46]	[10]			
Train Res	224	224	224	224	224	224	224	160
Test Res	224	224	224	224	224	224	224	224
Epochs	90	90	120	300	400	600	300	100
# of forward pass	450k	450k	300k	375k	500k	375k	188k	63k
Batch size	256	256	512	1024	1024	2048	2048	2048
Optimizer	SGD-M	SGD-M	SGD-M	AdamW	SGD-M	LAMB	LAMB	LAMB
LR	0.1	0.1	0.2	1×10^{-3}	2.0	5×10^{-3}	5×10^{-3}	8×10^{-3}
LR decay	step	step	step	cosine	step	cosine	cosine	cosine
decay rate	0.1	0.1	0.1	–	$0.02^{t/400}$	–	–	–
decay epochs	30	30	30	–	1	–	–	–
Weight decay	10^{-4}	10^{-4}	10^{-4}	0.05	10^{-4}	0.01	0.02	0.02
Warmup epochs	✗	✗	✗	5	5	5	5	5
Label smoothing ε	✗	✗	✗	0.1	0.1	✗	✗	✗
Dropout	✗	✗	✗	✗	✗	✗	✗	✗
Stoch. Depth	✗	✗	✗	0.1	✗	0.05	0.05	✗
Repeated Aug	✗	✗	✓	✓	✗	✓	✓	✗
Gradient Clip.	✗	✗	✗	✗	✗	✗	✗	✗
H. flip	✓	✓	✓	✓	✓	✓	✓	✓
RRC	✗	✓	✓	✓	✓	✓	✓	✓
Rand Augment	✗	✗	✗	9/0.5	✗	7/0.5	7/0.5	6/0.5
Auto Augment	✗	✗	✗	✗	✓	✗	✗	✗
Mixup alpha	✗	✗	✗	0.8	0.2	0.2	0.1	0.1
Cutmix alpha	✗	✗	✗	1.0	✗	1.0	1.0	1.0
Erasing prob.	✗	✗	✗	0.25	✗	✗	✗	✗
ColorJitter	✗	✓	✓	✗	✗	✗	✗	✗
PCA lighting	✓	✗	✗	✗	✗	✗	✗	✗
SWA	✗	✗	✗	✗	✓	✗	✗	✗
EMA	✗	✗	✗	✗	✗	✗	✗	✗
Test crop ratio	0.875	0.875	0.875	0.875	0.875	0.95	0.95	0.95
CE loss	✓	✓	✓	✓	✓	✗	✗	✗
BCE loss	✗	✗	✗	✗	✗	✓	✓	✓
Mixed precision	✗	✗	✗	✓	✓	✓	✓	✓
Top-1 acc.	75.3%	76.1%	77.0%	78.4%	79.5%	80.4%	79.8%	78.1%

Details of our ingredients and comparison to existing training procedures. In Table 9 we compare different recipes used to train vanilla ResNet-50 to ours. We consider only the results with the unmodified ResNet-50 architecture. We have chosen a wide range of training procedures to try to be as representative as possible but obviously it cannot be exhaustive. We do not consider approaches using advanced training settings like distillation, or models pre-trained self-supervised or with pseudo-labels.

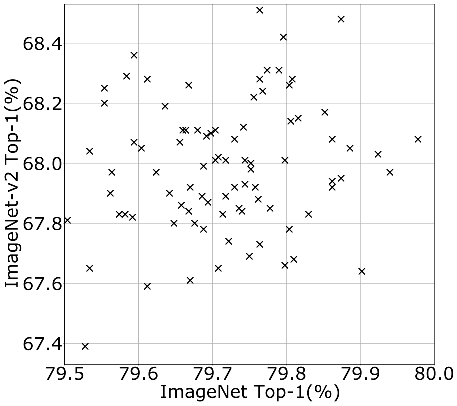
D Transfer Learning

In Table 10 we provide transfer learning performance on seven fine grained dataset with our different pre-training procedures, and provide a comparison with the default PyTorch pre-training. For each pre-training we use exactly the same fine-tuning procedure inspired by the fine-tuning procedure used in DeiT [46]. For each dataset we adapt the fine-tuning hyper-parameters.

We observe that the fine-tuning tend to smooth the difference of performance on certain datasets, such as CIFAR or Stanford Cars. Overall our A1 procedure leads to the best performance on downstream tasks, but the performance of the Pytorch default and A2 tend to be similar, while on Imagenet-val and -v2 A2 was significantly better. A3 is significantly inferior on downstream tasks, which may be related to the lower training resolution at 160×160.

Table 10: Performance comparison on transfer-learning tasks for different pre-training recipes.

Dataset	Train size	Test size	#classes	Pytorch [1]	A1	A2	A3
ImageNet-val [37]	1,281,167	50,000	1000	76.1	**80.4**	79.8	78.1
iNaturalist 2019 [19]	265,240	3,003	1,010	73.2	73.9	**75.0**	73.8
Flowers-102 [30]	2,040	6,149	102	**97.9**	**97.9**	**97.9**	97.5
Stanford Cars [25]	8,144	8,041	196	92.5	**92.7**	92.6	92.5
CIFAR-100 [26]	50,000	10,000	100	86.6	**86.9**	86.2	85.3
CIFAR-10 [26]	50,000	10,000	10	98.2	**98.3**	98.0	97.6



	Top-1 accuracy (%)				
dataset ↓	mean	std	max	min	seed 0
ImageNet-val	79.72	0.10	79.98	79.50	79.85
ImageNet-real	85.37	0.08	85.55	85.21	85.45
ImageNet-V2	67.99	0.23	68.69	67.39	67.90

Figure 2: *Top* ↑*:* Statistics for ResNet-50 trained with A2 and 100 different seeds. The column "seed 0" corresponds to the weights that we take as reference. Its performance is +0.13% above the average top-1 accuracy on Imagenet-val.

← *Left:* Point cloud plotting the ImageNet-val top-1 accuracy vs ImageNet-V2 for all seeds. Note that the outlying seed that achieves 68.5% top-1 accuracy on ImageNet-V2 has an average performance on ImageNet-val.

E Significance of measurements

E.1 Seed experiments

For a fixed set of choices and hyper-parameters, there is some inherent variability on the performance due to the presence of random factors in several stages. It is the case for the weight initialization, but also for the optimization procedure itself. For instance the order in which the images are fed to the network through batches depends on a random generator. This variability raises the question of the significance of accuracy measurements. For this purpose, we measure the distribution of performance when changing the random generator choices. This is conveniently done by changing the seed, as previously done by Picard [31], who concludes to the exist of outliers significantly outperforming or underperforming the average outcome of a traing procedure. In Figure 2, we report several statistics on the performance with the A2 training procedure when considering 100 distinct seeds (from 1 to 100, note that we have used seed=0 in all other experiments). In these experiments, we focus on the performance reached at the end of the training: we do not select the maximum obtained by intermediate checkpoints in the last epochs. This would have a similar effect as a seed selection, but the measures would not be IID and less disentangled from the training duration itself.

The standard deviation is typically around 0.1 on ImageNet-val, see Figure 2. This concurs with statistics reported in the literature for ResNet and other convnets [33]. The variance is higher on ImageNet-V2 (std=0.23), which consists of a smaller set (10000 vs 50000 for -val) of images not present in the validation set. The mean 79.72% shows that our main weights (seed 0) overestimates the average performance by about $+0.13\%$.

Peak performance and control of overfitting To prevent to over-estimate too much the accuracy on validation, during our exploration process we have selected only the final checkpoint and we use relatively coarse grid for hyper-parameters search to prevent introducing an additional seed effect. However optimizing over a large number of choices typically leads to overfitting. In Figure 2, we observe that the maximum (or peak performance) is close to 80.0% with the A2 training procedure. Note, Figure 3 provides the distribution of accuracy as an histogram;

One question is whether this model is intrinsically better than the average ones, or if it was just lucky on this particular measurement set. To attempt to answer this question, we measure how the performance transfers to another measurement dataset: we compute for all the seeds the couples

(ImageNet-val top-1 acc., ImageNet-V2 top-1 acc.), and plot them as a point cloud in Figure 2. We observe that the correlation between the performance on ImageNet-val and -V2 is limited. Noticeably the best performance is not achieved by the same seed on the two datasets. This observation suggests some significant measurement noise, which advocates to report systematically the performance on different datasets, and more particularly one making a clear distinction between validation and test.

Variability along epochs and discussion on early stopping. Figure 4 shows how the performance variability evolves along epochs, where we observe the variance of the score is very high until the last 100 epochs. In Figure 5, we additionally measure the performance early in the training and compare it to the final performance. It is only towards the end of the training that one can determine the most interesting seeds. We conclude that we can not apply an early stopping rule based on early results.

E.2 Comparing architectures and training procedures: a show-case of contradictory conclusions

In this paragraph we case how difficult it is to compare two architectures, even under the same training procedure, or conversely how it is difficult to compare different procedures with a single architecture. We choose ResNet-50 and DeiT-S. The latter [46] is essentially a ViT parameterized so that it has approximately the same number of parameters as a ResNet-50. For each architecture, we have put a significant effort in optimizing the procedure to maximize the performance on Imagenet-val with the same 300 epochs training schedule and same batch size. Under this constraint, the best training procedure that we have designed for ResNet-50 is A2. We denote by T2 the corresponding training procedure for DeiT-S. Note that this training procedure achieves a significantly better performance on Imagenet-val than the one initially proposed for DeiT-S (80.4% versus 79.8% in the original paper).

	test set →	ImageNet-val		ImageNet-v2	
↓ architecture	training →	A2	T2	A2	T2
ResNet-50		79.9	79.2	67.9	67.9
DeiT-S		79.6	80.4	68.1	69.2

As one can see, by choosing the procedure optimized for any of the two architectures, one may conclude that this architecture is better based on ImageNet-val accuracy: with A2 training, ResNet50 is better than DeiT-S, with T2 training, DeiT-S is better than ResNet50. The measurements on ImageNet-v2 would lead to a different conclusion, as DeiT-S is better for both procedure. But even in that case, by focusing on A2 one may conclude that the difference between ResNet-50 and DeiT-S with A2 training is not statistically significant: 67.9% vs 68.1%. Conversely, if the goal is to compare A2 to T2, we could draw different conclusions on ImageNet-val if considering a single architecture.

F Alternative Training Procedures

The main training recipes in this paper uses the LAMB optimizer. Several sets of hyper-parameter variations with differing training costs were presented with leading results for the vanilla ResNet-50 architecture. Here, we introduce alternative training recipes that also produce results matching or exceeding the best existing ResNet-50. The reader may find these are better suited for use or adaptation for their specific model architecture, dataset, or task. The alternative recipes are:

Procedure B – RMSProp with EMA weight averaging and step LR decay;

Procedure C – SGD with Nesterov's momentum, Adaptive Gradient Clipping, and a cosine learning rate decay. We have two variants of it (C1 and C2) depending on whether we use repeated augmentation or not;

Procedure D – AdamP with a cosine learning rate decay and binary cross-entropy.

The above procedures have been used to product excellent results for many pre-trained models in the **timm** library, including many non-ResNet architectures. Table 11 summarizes their best ResNet-50 oriented settings.

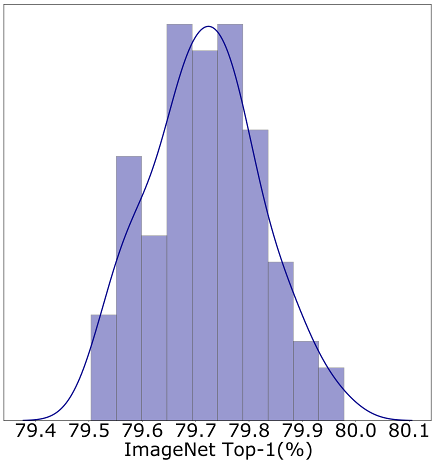


Figure 3: Distribution of the performance on ImageNet-val with the A2 procedure. It is measured with 100 different seeds. We also depict the Gaussian-fit of this distribution.

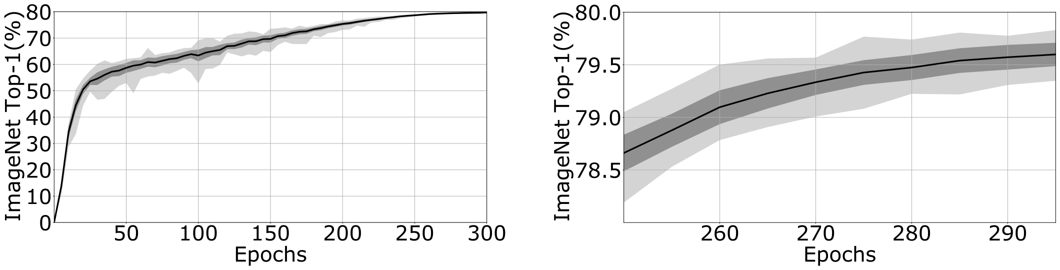


Figure 4: We show how the mean, standard deviation, minimum and maximum of the top-1 accuracy on ImageNet-val evolves during training with the A2 procedure (ResNet-50 architecture). **(Left)** For all 300 training epochs. **(Right)** Same but for the last epochs. We note that the variance in accuracy is high at the beginning, see for instance at epoch 100, where the difference in performance can be as large as 10% in accuracy. Towards the end of the training, most of the networks converge to similar values and the range significantly decreases in the last 50 epochs. *Credit*: this figure and experiment was inspired by Picard [31].

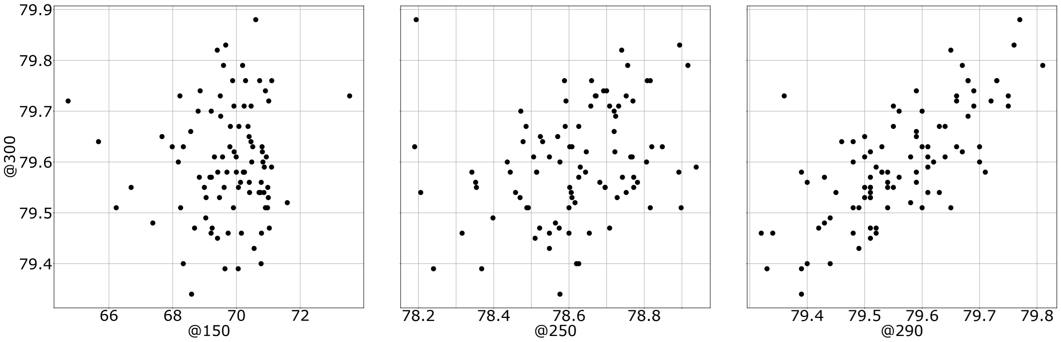


Figure 5: Final accuracy @ Epoch 300 versus accuracy at epochs 150, 250 and 290, for 100 networks trained with A2 training. It is only close to the end of the training that we start observing a correlation between temporary and final performance. We can therefore not apply early stopping rules based on an early validation accuracy.

Procedure →	B	C.1	C.2	D
Train Res	224	224	224	224
Test Res	224	224	224	224
Epochs	600	800	800	600
# of forward pass	375k	500k	500k	2,000k
Batch Size	2048	2048	2048	384
Optimizer	RMSProp	SGD	SGD	AdamP
Initial LR	0.18	0.88	0.88	0.0033
LR Scheduler	step	cosine	cosine	cosine
Decay Rate	0.988 per 1-epoch	✗	✗	✗
LR Noise (% of training)	0.45 to 1.0	✗	✗	✗
Weight Decay	7.0×10^{-6}	1.0×10^{-5}	1.0×10^{-5}	0.01
Warmup Epochs	5	5	5	5
Label Smoothing	0.1	0.1	0.1	0.1
Dropout	0.2	0.25	0.25	0.1
Stochastic Depth	0.1	0.1	0.1	0.05
Repeated Augmentation	✗	✗	✓	✗
Grad Clipping	✗	AGC .025	AGC .05	✗
RandAugment (M/N/MSTD)	8/2/1.0	7/3/1.0	7/3/1.0	7/3/1.0
Mixup	0.2	0.2	0.2	0.2
CutMix	✗	1.0	1.0	1.0
Random Erasing (Prob/Count)	0.35/3	0.4/1	0.4/1	.35/1
EMA weight averaging	0.9999	✗	✗	✗
CE loss	✓	✓	✓	✗
BCE loss	✗	✗	✗	✓
Top-1 acc.	79.4%	79.8%	80.0%	79.8%

Table 11: Alternative training procedures giving good performance with ResNet-50 architecture.

Training procedure B details (RMSProp) This procedure is inspired by the RandAugment [8] recipes used to train EfficientNet architectures but leverages features in **timm**'s implementation of RandAugment and Random Erasing. The step decay has been adjusted to decay every epoch (instead of every 2.4 as with EfficientNet, weight decay has been slightly decreased from EfficientNet defaults, and the learning rate is a bit higher. Additional augmentation was added in the form of per-pixel noise Random Erasing and Mixup. It should be noted that the RMSProp optimizer used is the **rmsprop_tf** implementation in **timm** which carefully matches behaviours of the Tensorflow (before version 2.0) implementation. The native PyTorch RMSProp implementation will not produce the same results, even if adjusting for the epsilon location.

With long decay constants for the EMA weight averaging, it can be beneficial to perturb the learning rate (currently once per epoch) with noise in later stages of training (typically 40-50% of the way through until the end). In exploration so far, learning rate noise appears to increase sensitivity of training results to random seed but has often produced the best result in (so far, limited) sweeps with the same hyper-parameters. Further analyzing the interplay between learning rate value, schedule, and noise, EMA decay constant, and random seed is a future objective for refining this training recipe.

This training strategy varies somewhat in effectiveness with batch size. Running experiments for this paper with larger batch sizes in the 1024-2048 range has often come slightly below (0.1 to 0.3 top-1) prior training runs with smaller sizes in the 256-768 range used for numerous **timm** pre-trained weights. It is unclear if this can be addressed with further hyper-parameter adjustments and different learning rate scaling (linear used by default) across batch sizes.

See Table 12 for a summary of the procedure, including ranges of recommended of values to search over for applying to different classification task and architecture combinations. For larger model architectures it is advisable to focus on stronger augmentation and regularization values within the suggested ranges. Looking at Table 3, the original results for the **timm** specific EfficientNetV2-S [43] variant and ECA-ResNet-269-D were trained using this procedure, but with higher levels of augmentation and regularization than for ResNet-50.

Training procedure C details (SGD with Nesterov's momentum and AGC) This recipe is based on the published procedure for training NFNets [5]: using SGD with Nesterov's momentum, Adaptive Gradient Clipping [5] (AGC), and heavy augmentation and regularization. AGC allows for stable large batch training at higher learning rates. Stronger default augmentation and regularization make up for the loss of batch normalization's regularizing effect when paired with NFNets, but strengths can be relaxed when used with other architectures that use batch normalization. With some adjustments, this procedure has been useful training architectures in **timm** such as ECA-NFNet, ResNet, and EfficientNet variants to impressive performance levels. The original result for the **timm** EfficientNetV2-M [43] variant in Table 3 was trained with the C.1 recipe, but with significantly higher augmentation and regularization than for ResNet-50.

The C.1 vs C.2 versions of this procedure seen in Table 11 differ most significantly in the application of Repeated Augmentation. It should be noted that a shorter training length of 600 epochs also works quite well in both cases, with an expected drop of roughly 0.15-0.2 top-1 for the same seed. Table 13 includes ranges of the ingredients for exploring with different tasks and architectures.

Training Procedure D details (AdamP) Late in the process for this report a training trial using AdamP [17] showed promise. With limited runs so far a recipe based on AdamP has achieved a 79.8 top-1 on ImageNet-1k. Further experimentation is necessary, the trials so far were run at a comparatively small batch size, but the promising results warrant exploration. Note that unlike RMSProp or SGD (but similar to Adam and LAMB), it is recommended to use square root scaling when adjusting the learning rate for this recipe across different batch sizes.

Table 14 contains the recommended ranges for the ingredients of this recipe. These ranges have not been explored extensively across different model architectures as with procedures B and C.

Other Recipes Undoubtedly, other training recipes with different combinations of optimizer, learning rate schedule, augmentation, and regularization exist that can match or surpass the performance of the procedures detailed in this report. Ingredients aside, putting in the time and effort to tune the recipe with the target architecture is key. The authors already have an AdamW recipe in the works that is looking promising. We welcome feedback regarding these or other noteworthy procedures via the **timm** GitHub Discussions.

	Recommended Range	ResNet-50
Epochs	400-700	600
Initial LR (per batch size 256)	.01-.025	0.0225
LR Schedule	Step	Step
LR Decay Rate	0.97-0.99 per 1-3 epochs	0.988 per 1-epoch
LR Noise Active (% of training)	40-50% to 100%	45% to 100%
Grad Clipping	Off, global norm 1.0	off
Dropout	0-0.4	0.2
Stoch Depth	0-0.1	0.1
Repeated Augmentation	Off	Off
RandAugment (M / N / MSTD)	6-9 / 2-4 / 0.5-1.0	8 / 2 / 1.0
Random Erasing (Prob / Count)	0.1-0.5 / 1-3	0.35 / 1
Mixup	0.2, 0.5, 0.8	0.2
CutMix	Off, 0.8, 1.0	0
EMA Weight Averaging	On	On
Loss	CE	CE

Table 12: Procedure B summary

	Recommended Range	ResNet-50
Epochs	300-800	800
Initial LR (per batch size 256)	0.08-0.12	0.11
LR Schedule	cosine	cosine
Grad Clipping	AGC 0.01 - 0.05	AGC .05
Dropout	0-0.5	0.25
Stoch. Depth	0-0.2	0.1
Repeated Augmentation	Off, On	On
RandAugment (M / N / MSTD)	6-10 / 2-4 / 0.5-1.0	7 / 3 / 1.0
Random Erasing (Prob / Count)	0.1-0.5 / 1-3	0.4 / 1
Mixup	0.2, 0.5, 0.8	0.2
CutMix	Off, 0.8, 1.0	1.0
Loss	CE	CE

Table 13: Procedure C summary

	Recommended Range	ResNet-50
Epochs	300-600	600
Initial LR (per batch size 256)	0.002 - 0.003	0.0027
LR Schedule	cosine	cosine
Grad Clipping	None	None
Dropout	0-0.3	0.1
Stoch. Depth	0-0.1	0.05
Repeated Augmentation	Off, On	Off
RandAugment (M / N / MSTD)	6-9 / 2-4 / 0.5-1.0	7 / 3 / 1.0
Random Erasing (Prob / Count)	0.1-0.5 / 1-3	0.35 / 1
Mixup	0.2, 0.5, 0.8	0.2
CutMix	Off, 0.8, 1.0	1.0
Loss	CE, BCE	BCE

Table 14: Procedure D summary