Evolving Normalization-Activation Layers

Hanxiao Liu [1] Andrew Brock [2] Karen Simonyan [2] Quoc V. Le [1]

Abstract

Normalization layers and activation functions are critical components in deep neural networks that frequently co-locate with each other. Instead of designing them separately, we unify them into a single computation graph, and evolve its structure starting from low-level primitives. Our layer search algorithm leads to the discovery of *EvoNorms*, a set of new normalization-activation layers that go beyond existing design patterns. Several of these layers enjoy the property of being independent from the batch statistics. Our experiments show that EvoNorms not only excel on a variety of image classification models including ResNets, MobileNets and EfficientNets, but also transfer well to Mask R-CNN for instance segmentation and BigGAN for image synthesis, outperforming BatchNorm and GroupNorm based layers by a significant margin in many cases.

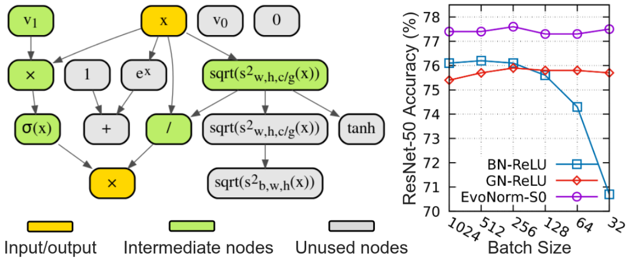

Figure 1. Left: Computation graph of a searched normalization-activation layer that is batch-independent, named EvoNorm-S0. The corresponding expression is $\sigma(v_1 x)\frac{x}{\sqrt{s^2_{w,h,c/g}(x)}}\gamma+\beta$, in contrast to $\max\left(\frac{x-\mu_{w,h,c/g}(x)}{\sqrt{s^2_{w,h,c/g}(x)}}\gamma+\beta, 0\right)$ for GroupNorm-ReLU. v_1, $\mu_{w,h,c/g}$ and $s^2_{w,h,c/g}$ refer to a learnable variable, group mean and group variance, respectively. *Right*: ResNet-50 results with EvoNorm-S0 as the batch size over 8 workers varies from 1024 to 32 on ImageNet. EvoNorm-S0 also outperforms both BN and GN-based layers on MobileNetV2 and Mask R-CNN.

1. Introduction

Normalization layers and activation functions are critical components in deep convolutional networks for stable optimization and improved generalization. Both components frequently co-locate with each other in state-of-the-art models, yet have been extensively studied as separate subjects in the literature. The underlying assumptions that they must be designed separately, and that they must function sequentially (e.g., as BatchNorm-ReLU or ReLU-BatchNorm (Ioffe & Szegedy, 2015; He et al., 2016a)) without interleaving is potentially suboptimal.

In this work, we revisit the co-design of normalization and activation layers by formulating them as a single building block. Searching for a unified layer of this kind is challenging for several reasons. For example, while it is crucial to avoid design choices historically made under the non-interleaving assumption, less prior knowledge implies larger search spaces where meaningful layer configurations are extremely rare. In addition, a useful layer must generalize well across multiple models and ideally to new tasks. One such example is BatchNorm-ReLU, which took years of extensive research efforts to discover.

We address these challenges via an automated approach. To ensure that we rely on as little prior knowledge as possible, we represent each layer as a tensor-to-tensor computation graph consisting of basic algorithmic primitives such as addition, multiplication and cross-dimensional aggregations. We then combine evolution with a rejection mechanism to efficiently navigate over this large and sparse search space. To ensure the searched layers generalize beyond a single deployment scenario, we explicitly formulate this requirement into our evolution objective by pairing and evaluating each candidate layer with multiple architectures.

Our layer search method leads to the discovery of a family of layers with novel structures, dubbed *EvoNorms*, that go beyond existing design patterns. We present both EvoNorm-B series that are **b**atch-dependent, as well as EvoNorm-S series that rely on only individual **s**amples, which are free of moving average statistics. We verify the performance of these layers across a variety of image classification architectures. For example, an EvoNorm-B layer improves

[1] Google Research, Brain Team [2] DeepMind. Correspondence to: Hanxiao Liu <hanxiaol@google.com>.

ResNet-50[1] top-1 accuracy on ImageNet from 76.1 to 77.8. Additionally, as the batch size becomes smaller, ResNet-50 with BN-ReLU drops its top-1 accuracy to 70.7 while an EvoNorm-S layer maintains 77.5, which is also significantly better than the 75.7 of GroupNorm-ReLU (Wu & He, 2018). To verify their generalization beyond classification, we pair EvoNorms with Mask R-CNN (He et al., 2017) and achieved clear gains over instance segmentation tasks on COCO with negligible computation overhead. We further utilize these layers in a BigGAN model (Brock et al., 2019) and achieve improved performance for image synthesis.

Our contributions can be summarized as follows:

- To the best of our knowledge, this is the first effort to automatically co-design the activation function and the normalization layer as a unified computation graph.
- Instead of relying on well-defined building blocks, we show it is possible to go beyond existing manual design patterns starting from very low-level primitives.
- We propose a novel *Layer Search* paradigm to find universal modules, in contrast to Architecture Search which focuses on specializing the networks. The layers' generalization is explicitly optimized w.r.t. multiple architectures using multi-objective evolution.
- We discover new layers that achieve accuracy gains when paired with a diverse set of image classification models, including ResNets (He et al., 2016a;b), MobileNetV2 (Sandler et al., 2018), MnasNet (Tan et al., 2019) and EfficientNets (Tan & Le, 2019). Several of these layers are free of batch statistics.
- We confirm the transferability of these layers to new tasks by applying them to Mask R-CNN (He et al., 2017) for detection and instance segmentation, and to BigGAN-deep (Brock et al., 2019) for image synthesis, achieving clear improvements over handcrafted layers.

2. Related Work

Separate efforts have been made for the design of activation functions only or normalization layers only, either manually (He et al., 2015; Clevert et al., 2016; Hendrycks & Gimpel, 2016; Klambauer et al., 2017; Ioffe & Szegedy, 2015; Ba et al., 2016; Ulyanov et al., 2016; Wu & He, 2018) or automatically (Ramachandran et al., 2018; Luo et al., 2019). Singh & Krishnan (2019) manually co-designed the activation and normalization layers as two separate components. Different from the above, our goal is to eliminate the boundary between normalization and activation layers at all by searching them jointly as a unified building block. This motivated us to focus on a more challenging search space than the existing automated approaches. For example, we avoid leveraging handcrafted normalization schemes (Luo et al., 2019) which are potentially suboptimal; we search for tensor-to-tensor transformations rather than scalar-to-scalar transformations (Ramachandran et al., 2018).

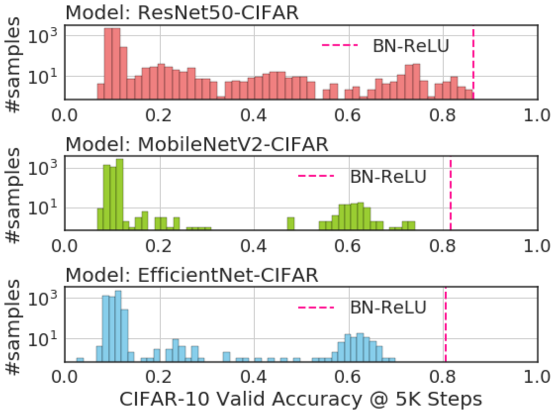

Figure 2. Histogram of the accuracies for 5000 random layers in our search space. Each layer is paired with three architectures and evaluated on CIFAR-10. Accuracies for the vast majority of these layers remain at 10% (note the y-axis is in log scale). A known good baseline (BN-ReLU) is denoted as the dashed vertical bar.

Our approach is inspired by recent works on neural architecture search, e.g., Zoph & Le (2017); Baker et al. (2017); Zoph et al. (2018); Liu et al. (2018); Real et al. (2019) and earlier Bayer et al. (2009), but is also fundamentally different. While existing works aim to *specialize* an architecture built upon well-defined building blocks such as Conv-BN-ReLU, we aim to find a *universal* layer that works well when paired with many different architectures, starting from low-level primitives similar to AutoML-Zero (Real et al., 2020). Our search space is also much sparser in the sense that only a tiny fraction of the layers can result in meaningful learning dynamics when paired with deep models (Figure 2).

Our work is also related to efforts on improving the initialization conditions for deep networks (Zhang et al., 2019; De & Smith, 2020; Bachlechner et al., 2020) in terms of challenging the necessity of traditional normalization layers. With less specialized initialization strategies, however, the structures of our most performant layers suggest that having certain notion of normalization improves generalization.

3. Search Space

3.1. Layer Representation

We represent each normalization-activation layer as a computation graph that transforms an input tensor into an output

[1] Unless otherwise specified, we always use the v2 instantiation of ResNets (He et al., 2016b) where all ReLU activation layers are adjacent to BatchNorm layers.

Element-wise Op	Expression	Arity
Add	$x + y$	2
Mul	$x \times y$	2
Div	$x/(y + \epsilon \cdot \text{sign}(y))$	2
Max	$\max(x, y)$	2
Neg	$-x$	1
Sigmoid	$\sigma(x)$	1
Tanh	$\tanh(x)$	1
Exp	e^x	1
Log	$\text{sign}(x) \cdot \ln(\|x\| + \epsilon)$	1
Abs	$\|x\|$	1
Square	x^2	1
Sqrt	$\text{sign}(x) \cdot \sqrt{\|x\|}$	1
Aggregation Op	**Expression**	**Arity**
Batch mean	$\mu_{b,w,h}(x)$	1
Channel mean	$\mu_{w,h,c}(x)$	1
Instance mean	$\mu_{w,h}(x)$	1
Group mean	$\mu_{w,h,c/g}(x)$	1
Batch std	$\sqrt{s^2_{b,w,h}(x) + \epsilon}$	1
Channel std	$\sqrt{s^2_{w,h,c}(x) + \epsilon}$	1
Instance std	$\sqrt{s^2_{w,h}(x) + \epsilon}$	1
Group std	$\sqrt{s^2_{w,h,c/g}(x) + \epsilon}$	1
Batch root 2nd moment	$\sqrt{\mu_{b,w,h}(x^2) + \epsilon}$	1
Channel root 2nd moment	$\sqrt{\mu_{w,h,c}(x^2) + \epsilon}$	1
Instance root 2nd moment	$\sqrt{\mu_{w,h}(x^2) + \epsilon}$	1
Group root 2nd moment	$\sqrt{\mu_{w,h,c/g}(x^2) + \epsilon}$	1

Table 1. Primitives of the search space. Terms highlighted in blue are batch-dependent hence will be replaced with moving average statistics during inference. We omit "ϵ" in later sections for brevity.

tensor of the same shape (Figure 1). Each intermediate node represents the outcome of either a unary or a binary operation (Table 1). These ops are designed to retain the dimensions of their input tensor(s) to ensure that nodes in the graph are always shape-compatible with each other. The graph has 4 initial nodes: the input tensor, a constant zero tensor, and two trainable vectors v_0 and v_1 along the channel dimension initialized as 0's and 1's, respectively.

A random graph can be generated in a sequential manner. Starting from the initial nodes, we generate each new node by randomly sampling a primitive op and then randomly sampling its input nodes according to the op's arity. The process is repeated multiple times and the last node is used as the output. Note that unused nodes are allowed in the graph (colored in gray in Figure 1), which can be potentially picked up in the future generations through mutations.

3.2. Primitives

Table 1 shows the primitives in the search space, which are divided into two categories: (i) element-wise ops applied to each element in the tensor, and (ii) aggregation ops that enable communication across different axes of the tensor. Below we focus on the definition of the latter.

Let x be a 4-dimensional tensor of feature maps. We use (b, w, h, c) to refer to its batch, width, height and channel dimensions, respectively. We use $x_{\mathcal{I}}$ to represent a subset of x's elements along certain dimensions indicated by $\mathcal{I}$. For example, $\mathcal{I} = (b, w, h)$ indexes all the elements of x along the batch, width and height dimensions.

Let $\mu_{\mathcal{I}}(x)$ be a shape-preserving mapping that replaces each element in x with the mean of $x_{\mathcal{I}}$. Likewise, let $s^2_{\mathcal{I}}(x)$ be a mapping that transforms each element of x into the variance among all the elements in $x_{\mathcal{I}}$. The mean and variance ops are related via $s^2_{\mathcal{I}}(x) = \mu_{\mathcal{I}}((x - \mu_{\mathcal{I}}(x))^2)$. We also use a special symbol $\cdot/g$ to indicate that the aggregation is carried out in a grouped manner along certain dimensions.

4. Layer Search Method

Our search method features several key ingredients:

- We evaluate each layer by pairing it with multiple architectures (Section 4.1) and training the models over a lightweight proxy task (Section 4.2).

- We use evolution to optimize the multi-objective frontier, which is augmented by an efficient rejection mechanism to filter out undesirable layers (Section 4.4).

The overall workflow is illustrated in Figure 3.

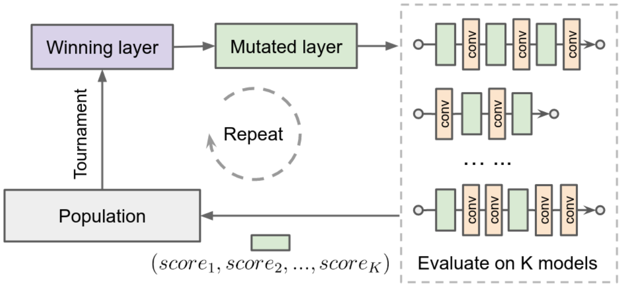

Figure 3. Workflow of our algorithm. Each mutated layer is paired with K architectures, which are trained from scratch to receive K performance scores for multi-objective tournament selection.

4.1. Layer Evaluation

A useful layer, such as BatchNorm-ReLU, is expected to work well across a variety of architectures. However, Figure 4 shows that layers performing well on a single given architecture do not necessarily transfer to the others. To explicitly promote generalization, we formulate the search as a multi-objective optimization problem, where each candidate layer is always evaluated over K $(K > 1)$ different *anchor* architectures to obtain multiple fitness scores.

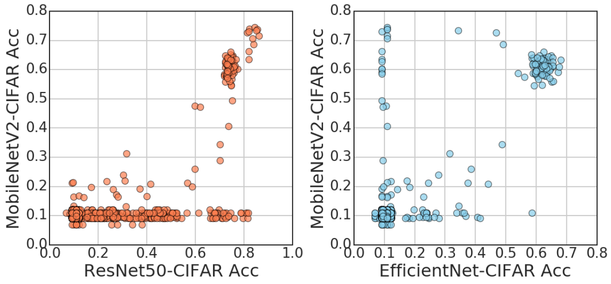

Figure 4. Accuracy calibration of 5000 random layers when paired with different image classification architectures. The correlation is far from perfect–layers performing well with one architecture may not lead to meaningful learning dynamics at all on the others.

4.2. Proxy Task and Anchor Architectures

An ideal proxy task should be lightweight enough to allow speedy feedback. At the same time, the anchor architectures must be sufficiently challenging to train from the optimization point of view to stress-test the layers. These motivated us to consider a small training task and deep architectures.

We therefore define the proxy task as image classification on the CIFAR-10 dataset (Krizhevsky & Hinton, 2009), and consider three representative ImageNet architectures (that are deep enough) adapted with respect to this setup, including Pre-activation ResNet50 (He et al., 2016b) with channel multiplier $0.25\times$, MobileNetV2 (Sandler et al., 2018) with channel multiplier $0.5\times$ and EfficientNet-B0 (Tan & Le, 2019) with channel multiplier $0.5\times$. To handle the reduced image resolution on CIFAR-10 relative to ImageNet, the first two convolutions with spatial reduction are modified to use stride one for all of these architectures. We refer to these adapted versions as ResNet50-CIFAR, MobileNetV2-CIFAR and EfficientNet-CIFAR, respectively.

We do not change anything beyond channel multipliers and strides described above, to preserve the original attributes (e.g., depth) of these ImageNet models as much as possible. Their building blocks are illustrated in Figure 5.

4.3. Evolution

Our evolution algorithm is a variant of tournament selection (Goldberg & Deb, 1991). At each step, a tournament is formed based on a random subset of the population. The winner of the tournament is allowed to produce a mutated offspring, which will be evaluated and added into the population. The overall quality of the population hence is expected to improve as the process repeats. We also regularize the evolution by maintaining a sliding window of only the most recent portion of the population (Real et al., 2019).

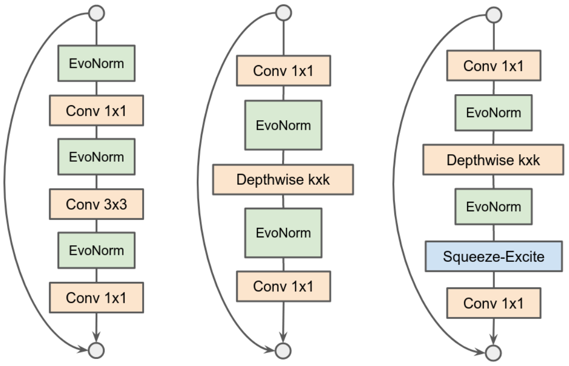

Figure 5. Block definitions of the anchor architectures: ResNet-CIFAR (*left*), MobileNetV2-CIFAR (*center*), and EfficientNet-CIFAR (*right*). For each model, a custom layer is used to replace BatchNorm-ReLU/Swish in the original architecture[3]. Each custom layer is always followed by a channel-wise affine transform.

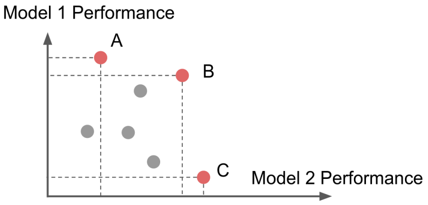

Figure 6. Illustration for two alternative tournament selection criteria for multi-objective evolution. Each point represents a candidate layer in the tournament. Under the average criterion, B wins the tournament because it has the highest average performance on the two models. Under the Pareto criterion, each of A, B, and C wins with probability $\frac{1}{3}$ as all of them are Pareto-efficient.

Selection Criterion. There is no unique criterion to decide the tournament winner because each layer is associated with multiple scores. Below are two viable options:

- *Average*: Layer with the highest average accuracy wins (e.g., B in Figure 6 wins).
- *Pareto*: A random layer on the Pareto frontier wins (e.g., A, B, C in Figure 6 are equal likely to win).

While the average criterion is intuitive, it implicitly assumes performance scores over different anchor architectures are comparable with each other hence can be biased towards the architecture with easiest gains. The Pareto criterion is more generic and tends to promote population diversity. It also resembles NSGA-II (Deb et al., 2002), a well-established multi-objective genetic algorithm, in terms of simultaneously optimizing all the non-dominated solutions. We therefore focus on the Pareto criterion in this work.

Mutation. We mutate the computation graph of the winning layer in three steps. (1) Select an intermediate node

[3] If a batch normalization is used without any activation function, we simply replace it with a channel-wise affine transform.

uniformly at random. (2) Replace its current operation with a new one in Table 1 uniformly at random. (3) Select new predecessors for this node uniformly at random.

4.4. Rejection Protocols

Quality. We discard layers that achieve less than 20%[4] validation accuracy in 100 training steps on *any* of the three anchor architectures. Since the vast majority of the candidate layers do not yield meaningful learning dynamics at all (Figure 2), this simple mechanism ensures the compute resources to concentrate on the full training processes of a small subset of promising candidates.

Stability. In addition to quality, we reject layers that are subject to numerical instability. The basic idea is to stress-test the candidate layer by *adversarially* adjusting the convolutional weights θ towards the direction of maximizing the network's gradient norm. Formally, let $\ell(\theta, G)$ be the training loss of a model when paired with computation graph G (a candidate normalization-activation layer). Instability of training is reflected by the worst-case gradient norm:

$$\max_{\theta} \left\| \frac{\partial \ell(\theta, G)}{\partial \theta} \right\|_2 \tag{1}$$

We seek to maximize the value above by ascending along the direction of $\frac{\partial \|\partial \ell(\theta,G)/\partial\theta\|_2}{\partial \theta}$ for 100 steps. Layers with the worst-case gradient norm greater than 10^8 are rejected.

The two tests are complementary with each other. For example, we found a layer like $\tanh(x) - \mu_{w,h,b}(\max(x, 0))$ is able to achieve reasonable accuracies on CIFAR-10 across all the anchor architectures, but its gradients quickly explode on ImageNet possibly due to the absence of normalization operations. This layer will be excluded thanks to the stability test. On the other hand, a trivial zero layer will be perfectly stable, but will be rejected by the performance test because of its poor classification accuracy.

5. Experiments

5.1. Implementation Details

Proxy Task. We use the same training setup for all the architectures. Specifically, we use 24×24 random crops on CIFAR-10 with a batch size of 128 for training, and use the original 32×32 image with a batch size of 512 for validation. We use SGD with learning rate 0.1, Nesterov momentum 0.9 and weight decay 10^{-4}. Each model is trained for 2000 steps with a constant learning rate for the EvoNorm-B experiment. Each model is trained for 5000 steps following a cosine learning rate schedule for the EvoNorm-S experiment. These are chosen to ensure the majority of the models can achieve reasonable convergence quickly. With our implementation, it takes 3-10 hours to train each model on a single CPU worker with two cores.

Evolution. We regularize the evolution (Real et al., 2019) by considering a sliding window of only the most recent 2500 genotypes. Each tournament is formed by a random subset of 5% of the active population. Winner is determined as a random candidate on the Pareto-frontier w.r.t. the three accuracy scores (Section 4.3), which will be mutated twice in order to promote structural diversity. To encourage exploration, we further inject noise into the evolution process by replacing the offspring with a completely new random architecture with probability 0.5. Each search experiment takes 2 days to complete with 5000 CPU workers.

ImageNet Evaluation. For ImageNet results presented in Table 2 (layers with batch statistics), we use a base learning rate of 0.1 per 256 images for ResNets, MobileNetV2 and MnasNet, and a base learning rate of 0.016 per 256 images for EfficientNets following the official implementation. Note these learning rates have been heavily optimized w.r.t. batch normalization. For results presented in Table 3 (layers without batch statistics), the base learning rate for MobileNetV2 is lowered to 0.03 (tuned w.r.t. GN-ReLU among 0.01, 0.03, 0.1). We use the standard multi-stage learning rate schedule for ResNets, cosine schedule (Loshchilov & Hutter, 2017) for MobileNetV2 and MNASNet, and the original polynomial schedule for EfficientNets. These learning rate schedules also come with a linear warmup phase (Goyal et al., 2017). For all architectures, the batch size per worker is 128 and the input resolution is 224×224. The only exception is EfficientNet-B5, which uses batch size 64 per worker and input resolution 456×456. The number of workers is 8 for ResNets and 32 for the others. We use 32 groups for grouped aggregation ops $\mu_{w,h,c/g}$ and $s^2_{w,h,c/g}$.

5.2. Reranking

After search, we take the top-10 candidates from evolution[5] and pair each of them with fully-fledged ResNet-50, MobileNetV2 and EfficientNet-B0. We then rerank the layers based on their averaged ImageNet accuracies of the three models. To avoid overfitting the validation/test metric, each model is trained using 90% of the ImageNet *training* set and evaluated over the rest of the 10%. The top-3 reranked layers are then used to obtain our main results.

The reranking task is more computationally expensive than the proxy task we search with, but is accordingly more representative of the downstream tasks of interest, allowing for better distinguishing between top candidates.

[4] This is twice as good as random guess on CIFAR-10.

[5] We compile a complete list of the top-10 layers in Appendix B.

Layer	Expression	R-50	R-101	MobileNetV2	MnasNet-B1	EfficientNet-B0	EfficientNet-B5
BN-ReLU	$\max(z,0)$, $\frac{x-\mu_{b,w,h}(x)}{\sqrt{s^2_{b,w,h}(x)}}\gamma+\beta$	76.1	77.9	73.1	74.5	76.4	**83.6**
BN-PReLU	$\max(z,v_0)$, $\frac{x-\mu_{b,w,h}(x)}{\sqrt{s^2_{b,w,h}(x)}}\gamma+\beta$	76.1	77.4	74.0	75.0	76.3	**83.6**
BN-Swish-1	$z\sigma(z)$, $z=\frac{x-\mu_{b,w,h}(x)}{\sqrt{s^2_{b,w,h}(x)}}\gamma+\beta$	77.2	78.7	74.4	75.1	**77.0**	83.4
BN-Swish	$z\sigma(v_1 z)$, $z=\frac{x-\mu_{b,w,h}(x)}{\sqrt{s^2_{b,w,h}(x)}}\gamma+\beta$	77.0	78.6	74.5	**75.3**	**77.0**	83.5
Random	$\text{sign}(z)\sqrt{z}\gamma+\beta$, $z=\sqrt{s^2_{w,h}(\sigma(\lvert x\rvert))}$	1e-3	1e-3	1e-3	1e-3	1e-3	1e-3
Random + rej	$\tanh(\max(x,\tanh(x)))\gamma+\beta$	70.2	71.6	1e-3	1e-3	1e-3	1e-3
RS + rej	$\frac{\max(x,0)}{\sqrt{\mu_{b,w,h}(x^2)}}\gamma+\beta$	76.3	78.1	73.4	74.7	76.4	83.2
EvoNorm-B0	$\frac{x}{\max\left(\sqrt{s^2_{b,w,h}(x)},v_1 x+\sqrt{s^2_{w,h}(x)}\right)}\gamma+\beta$	**77.8**	**79.1**	**75.1**	**75.3**	76.8	**83.6**
EvoNorm-B1	$\frac{x}{\max\left(\sqrt{s^2_{b,w,h}(x)},(x+1)\sqrt{\mu_{w,h}(x^2)}\right)}\gamma+\beta$	77.5	78.7	74.5	75.1	76.5	**83.6**
EvoNorm-B2	$\frac{x}{\max\left(\sqrt{s^2_{b,w,h}(x)},\sqrt{\mu_{w,h}(x^2)}-x\right)}\gamma+\beta$	**77.8**	79.0	74.7	74.9	76.6	83.4

Table 2. ImageNet test accuracies of different normalization-activation layers. Terms requiring moving average statistics are highlighted in blue. Results for the same architecture are obtained using the same codebase with identical training setup.

Model	Layer	Expression	Images / Worker					
			128	64	32	16	8	4
ResNet-50 (8 workers)	BN-ReLU	$\max(z,0)$, $z=\frac{x-\mu_{b,w,h}(x)}{\sqrt{s^2_{b,w,h}(x)}}\gamma+\beta$	76.1	76.2	76.1	75.6	74.3	70.7
	GN-ReLU	$\max(z,0)$, $z=\frac{x-\mu_{w,h,c/g}(x)}{\sqrt{s^2_{w,h,c/g}(x)}}\gamma+\beta$	75.4	75.7	75.9	75.8	75.8	75.7
	GN-Swish	$z\sigma(v_1 z)$, $z=\frac{x-\mu_{w,h,c/g}(x)}{\sqrt{s^2_{w,h,c/g}(x)}}\gamma+\beta$	77.2	77.3	77.2	77.3	77.0	77.3
	FRN	$\max(z,v_0)$, $z=\frac{x}{\sqrt{\mu_{w,h}(x^2)}}\gamma+\beta$	75.3	75.8	75.9	75.9	75.8	76.0
	EvoNorm-S0	$\frac{x\sigma(v_1 x)}{\sqrt{s^2_{w,h,c/g}(x)}}\gamma+\beta$	77.4	**77.4**	**77.6**	77.3	**77.3**	**77.5**
	EvoNorm-S1	$\frac{x\sigma(x)}{\sqrt{s^2_{w,h,c/g}(x)}}\gamma+\beta$	**77.5**	77.3	77.4	**77.4**	**77.3**	77.4
	EvoNorm-S2	$\frac{x\sigma(x)}{\sqrt{\mu_{w,h,c/g}(x^2)}}\gamma+\beta$	76.9	77.3	77.1	77.2	77.1	77.1
MobileNetV2 (32 workers)	BN-ReLU	–	73.3	73.2	72.7	70.0	64.5	60.4
	GN-ReLU	–	71.5	72.6	72.9	72.5	72.7	72.6
	GN-Swish	–	73.4	73.7	73.7	73.6	**74.0**	73.5
	FRN	–	73.3	73.4	73.5	73.6	73.5	73.5
	EvoNorm-S0	–	**74.0**	**74.0**	**73.9**	**73.9**	73.9	**73.8**
	EvoNorm-S1	–	73.7	**74.0**	73.6	73.7	73.7	**73.8**
	EvoNorm-S2	–	73.8	73.4	73.7	**73.9**	73.9	**73.8**

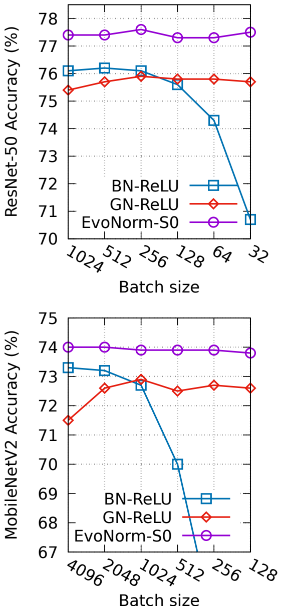

Table 3. ImageNet test accuracies of sample-based layers without batch statistics. Learning rates are scaled linearly relative to the batch sizes (Goyal et al., 2017). Results for the same architecture are obtained using the same codebase with identical training setup.

5.3. Generalization across Image Classifiers

In Table 2, we compare the discovered layers against some widely used normalization-activation layers on ImageNet, including strong baselines with the searched Swish activation function (Ramachandran et al., 2018). We refer to our layers as the EvoNorm-B series, as they involve **B**atch aggregations ($\mu_{b,w,h}$ and $s^2_{b,w,h}$) hence require maintaining a moving average statistics for inference. The table shows that EvoNorms significantly improve the accuracies of ResNets and MobileNetV2. Notably, EvoNorm-B0 is no worse than BN-ReLU/Swish across all of the six architectures. The only exception is the 0.2% drop relative to BN-Swish on EfficientNet-B0, which is particularly interesting given that the architecture and its hyperparameters have been explicitly optimized w.r.t. BN-Swish (Tan & Le, 2019).

Table 3 presents EvoNorms obtained from another search experiment, during which layers containing batch aggregation ops are excluded. The goal is to design layers that rely on individual samples only, a desirable property to simplify implementation and to stabilize training with small batch sizes. We refer to these **S**ample-based layers as the EvoNorm-S series. We compare them against handcrafted

Backbone	Layer	AP^{bbox}	AP^{bbox}_{50}	AP^{bbox}_{75}	AP^{mask}	AP^{mask}_{50}	AP^{mask}_{75}	MAdds (B)	Params (M)	Batch Indep.
R-50-FPN	BN-ReLU	42.1	62.9	46.2	37.8	60.0	40.6	332.0	46.37	✗
	EvoNorm-B0	**44.0**$_{(+1.9)}$	**65.2**$_{(+2.3)}$	**48.1**$_{(+1.9)}$	**39.5**$_{(+1.7)}$	**62.7**$_{(+2.7)}$	**42.4**$_{(+1.8)}$	332.3	46.38	
	GN-ReLU	42.7	63.8	46.6	38.4	61.2	41.2	332.3	46.37	✓
	EvoNorm-S0	**43.6**$_{(+0.9)}$	**64.9**$_{(+1.1)}$	**47.9**$_{(+1.3)}$	**39.4**$_{(+1.0)}$	**62.3**$_{(+1.1)}$	**42.7**$_{(+1.5)}$	331.9	46.38	
R-101-FPN	BN-ReLU	43.8	64.2	48.2	39.2	61.3	42.5	409.8	65.36	✗
	EvoNorm-B0	**45.1**$_{(+1.3)}$	**66.2**$_{(+2.0)}$	**49.0**$_{(+0.8)}$	**40.1**$_{(+0.9)}$	**63.3**$_{(+2.0)}$	**43.1**$_{(+0.6)}$	410.1	65.38	
	GN-ReLU	44.0	64.7	48.3	39.4	62.1	**42.7**	410.3	65.36	✓
	EvoNorm-S0	**44.5**$_{(+0.5)}$	**65.3**$_{(+0.6)}$	**48.7**$_{(+0.4)}$	**39.6**$_{(+0.2)}$	**62.3**$_{(+0.2)}$	42.6$_{(-0.1)}$	409.7	65.38	

Table 4. Mask R-CNN object detection and instance segmentation results on COCO `val2017`.

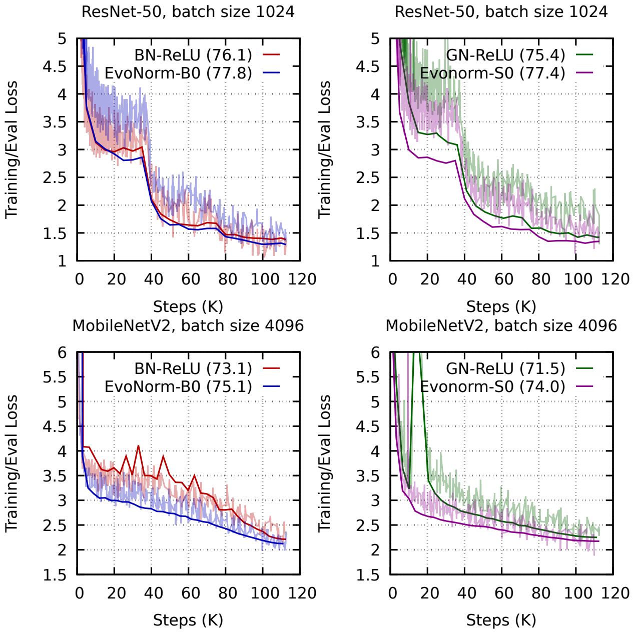

Figure 7. Training/eval curves for ResNet-50 and MobileNetV2 on ImageNet with large batch sizes. The corresponding test accuracy for each layer is reported in the legend.

baselines designed under a similar motivation, including Group Normalization (Wu & He, 2018) (GN-ReLU) and a recently proposed layer aiming to eliminate batch dependencies (Singh & Krishnan, 2019) (FRN). We also augment GN using Swish (Ramachandran et al., 2018) to make it an even stronger baseline (GN-Swish). Table 3 shows that while GN-ReLU improves over BN-ReLU for small batch sizes only, EvoNorm-S layers are able to surpass all the other layers across all batch sizes. It is also worth noticing that EvoNorm-S0 significantly outperforms BN-ReLU at the original batch size (1024 for ResNet-50 and 4096 for MobileNetV2) even without relying on batch statistics.

Figure 7 shows the learning dynamics of Evonorms with large batch sizes. With identical training setup, EvoNorm-B0 generalizes better than BN-ReLU on ResNet-50 despite higher training loss. In all the other cases, EvoNorms lead to both improved optimization and generalization.

5.4. Generalization to Instance Segmentation

To investigate if our discovered layers generalize beyond the classification task they were searched on, we pair them with Mask R-CNN (He et al., 2017) and ResNet-FPN (Lin et al., 2017) for object detection and instance segmentation on COCO (Lin et al., 2014). The training is carried out over 8 workers with 8 images/worker and the image resolution is 1024×1024. The models are trained from scratch for 135K steps using SGD with momentum 0.9, weight decay 4e-5 and an initial learning rate of 0.1, which is reduced by 10× at step 120K and step 125K. In our experiments, EvoNorms are applied to both the backbone and the heads to replace their original activation-normalization layers.

Results are summarized in Table 4. With both backbones, EvoNorms significantly improve the APs with negligible impact on FLOPs or model sizes. While EvoNorm-B0 offers the strongest results, EvoNorm-S0 outperforms GN-ReLU and BN-ReLU by a clear margin without requiring moving-average statistics. These results demonstrate that our layers transfer well to detection and instance segmentation tasks.

5.5. Generalization to GAN Training

We further test the applicability of EvoNorms to training GANs (Goodfellow et al., 2014). Normalization is particularly important in GAN training, where the unstable dynamics of the adversarial game render training sensitive to nearly every aspect of its setup. We replace the BN-ReLU layers in the generator of BigGAN-deep (Brock et al., 2019) with EvoNorms and with previously designed layers, and measure performance on ImageNet generation at 128×128 resolution using Inception Score (IS, (Salimans et al., 2016)) and Fréchet inception distance (FID, (Heusel et al., 2017)).

We compare two of our most performant layers, B0 and S0, against the baseline BN-ReLU and GN-ReLU, as well as LayerNorm-ReLU (Ba et al., 2016), and PixelNorm-ReLU (Karras et al., 2018), a layer designed for a different GAN architecture. We sweep the number of groups in GN-ReLU from 8,16,32, and report results using 16 groups. Consistent with BigGAN training, we report results at peak perfor-

Layer	**IS** (median/best)	**FID** (median/best)
BN-ReLU	**118.77/124.01**	7.85/7.29
EvoNorm-B0	101.13/113.63	**6.91/5.87**
GN-ReLU	99.09	8.14
LayerNorm-ReLU	91.56	8.35
PixelNorm-ReLU	88.58	10.41
EvoNorm-S0	**104.64/113.96**	**6.86/6.26**

Table 5. Image synthesis performance of different normalization-activation layers in the generator of BigGAN-deep, with layers that use batch statistics separated from those without. Where shown, median and best performance are reported across 3 random seeds. Higher is better for IS, lower is better for FID.

Figure 8. Selected samples from BigGAN-deep + EvoNorm-B0.

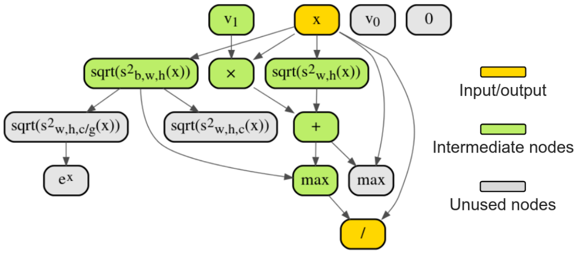

Figure 9. Computation graph of EvoNorm-B0. The layer has an expression of $\frac{x}{\max\left(\sqrt{s^2_{b,w,h}(x)}, v_1 x + \sqrt{s^2_{w,h}(x)}\right)}\gamma + \beta$, in contrast to $\max\left(\frac{x - \mu_{b,w,h}(x)}{\sqrt{s^2_{b,w,h}(x)}}\gamma + \beta, 0\right)$ for BN-ReLU. EvoNorm-B0 outperforms BN-ReLU across a variety of models for image classification (Table 2) and instance segmentation (Table 4), and achieves promising results on GANs (Table 5).

mance in Table 5. Selected samples from BigGAN-deep with EvoNorm-B0 are shown in Figure 8.

Swapping BN-ReLU out for most other layers substantially cripples training, but both EvoNorm-B0 and S0 achieve comparable, albeit worse IS, and improved FIDs over the BN-ReLU baseline. Notably, EvoNorm-S0 outperforms all the other per-sample normalization-activation layers in both IS and FID. This result further confirms that EvoNorms transfer to visual tasks in multiple domains.

5.6. Case Studies

Below we take a closer look at the most performant layers, namely the batch-based B0 (Figure 9) and the sample-based S0 (Figure 1) which is batch-independent. We provide code snippets for both layers in Appendix A.

EvoNorm-B0: $\frac{x}{\max\left(\sqrt{s^2_{b,w,h}(x)}, v_1 x + \sqrt{s^2_{w,h}(x)}\right)}\gamma + \beta$

Unlike conventional normalization schemes relying on a single type of variance only, EvoNorm-B0 attempts to mix together two types of variances in its denominator, namely $s^2_{b,w,h}(x)$ (batch variance) and $s^2_{w,h}(x)$ (instance variance). The former captures global variance across images within the same mini-batch, and the latter captures local variance per image. Normalization is more likely to be influenced by the dominating variance due to the $\max$ op. Interestingly, the intrinsic nonlinearity of this normalization process also eliminates the need of any explicit activation functions.

It is also intriguing to see that EvoNorm-B0 keeps the scale invariance property of traditional layers, which means rescaling the input x would not affect its output. This observation is aligned with some previous findings that scale invariance might play a useful role from the optimization point of view (Hoffer et al., 2018; Li & Arora, 2020).

EvoNorm-S0: $\frac{x\sigma(v_1 x)}{\sqrt{s^2_{w,h,c/g}(x)}}\gamma + \beta$

The numerator of EvoNorm-S0 resembles the Swish activation function (Ramachandran et al., 2018) and its denominator leverages the standard deviation part of Group Normalization (Wu & He, 2018) (GN). Note this is not equivalent to applying GN and Swish sequentially. First, unlike GN-Swish, EvoNorm-S0 does not center the feature maps by subtracting the mean $\mu_{w,h,c/g}(x)$ from x. Secondly, even if x is already zero-centered, their full expressions are still different: $\frac{x}{\sqrt{s^2_{w,h,c/g}(x)}}\sigma\left(v_1 \frac{x}{\sqrt{s^2_{w,h,c/g}(x)}}\right)$ for GN-Swish and $\frac{x}{\sqrt{s^2_{w,h,c/g}(x)}}\sigma(v_1 x)$ for EvoNorm-S0 (omitting γ and β). The latter is more compact and cannot be expressed as a sequential composition of normalization and activation.

EvoNorm-S0 is asymptotically scale-invariant in the sense that it reduces to either $\frac{x}{\sqrt{s^2_{w,h,c/g}(x)}}$ or a constant zero when the magnitude of x becomes large, depending on the sign of v_1. Both degenerated functions are scale-invariant to x.

5.7. Trade-off on Lightweight Models

Batch normalization is efficient thanks to its simplicity and the fact that both the moving averages and affine parameters can be fused into adjacent convolutions during inference. EvoNorms, despite being more powerful, can come with

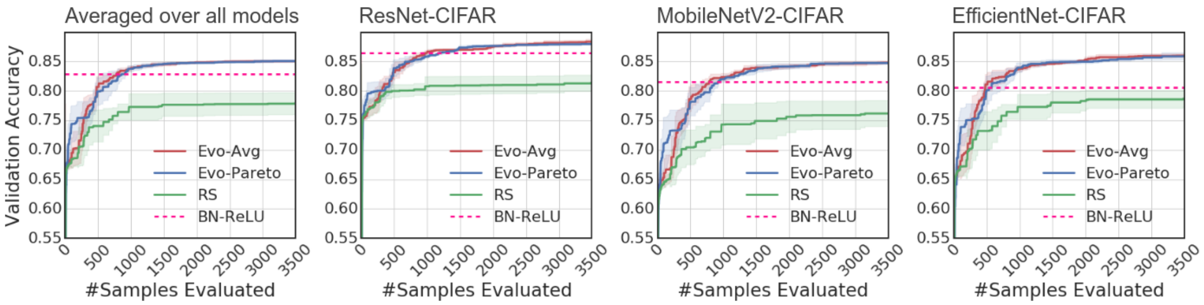

Figure 10. Search progress of evolution vs. random search vs. a fixed baseline (BN-ReLU) on the proxy task. Each curve denotes the mean and standard deviation of the top-10 architectures in the population. Only valid samples survived the rejection phase are reported.

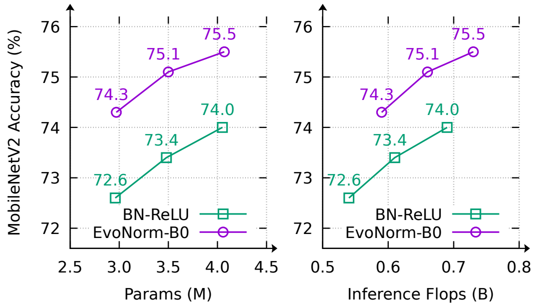

Figure 11. ImageNet accuracy vs. params and accuracy vs. FLOPs for MobileNetV2 paired with different normalization-activation layers. Each layer is evaluated over three model variants with channel multipliers $0.9\times$, $1.0\times$ and $1.1\times$. We consider the inference-mode FLOPs for both BN-ReLU and EvoNorm-B0, allowing parameter fusion with adjacency convolutions whenever possible.

more sophisticated expressions. While the overhead is negligible even for medium-sized models (Table 4), it can be nontrivial for lightweight, mobile-sized models.

We study this subject in detail using MobileNetV2 as an example, showing that EvoNorm-B0 in fact substantially outperforms BN-ReLU in terms of both accuracy-parameters trade-off and accuracy-FLOPs[6] trade-off (Figure 11). This is because the cost overhead of EvoNorms can be largely compensated by their performance gains.

5.8. Random Methods

In Figure 10 we compare evolution and random search on the proxy task, where a large gap is observed between their sample efficiencies for optimizing the search objective.

Table 2 shows a random sample in our search space can only achieve near-zero accuracy on ImageNet. We then show that by incorporating rejection rules in Section 4.4 (Random + rej), one can find a layer that works on ResNets. We further show that random search with rejection (RS + rej) using comparable compute with evolution is able to discover a compact variant of BN-ReLU. This layer achieves promising accuracies across all architectures, though it is consistently outperformed by EvoNorms.

6. Conclusion

In this work, we jointly search for normalization and activation layers as a unified computation graph consisting of low-level primitives. Our layer search method leads to the discovery of EvoNorms, a family of layers with novel design patterns, achieving clear gains across image classification, instance segmentation and GANs. Our results suggest a promising usage of AutoML to discover universal modules using layer search, in contrast to the predominant practice of specializing networks using architecture search.

Acknowledgements

The authors would like to thank Gabriel Bender, Chen Liang, Esteban Real, Prajit Ramachandran, Pengchong Jin, Ekin D. Cubuk, Barret Zoph, Da Huang, and Mingxing Tan for their comments and support on this work.

References

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Bachlechner, T., Majumder, B. P., Mao, H. H., Cottrell, G. W., and McAuley, J. Rezero is all you need: Fast convergence at large depth. *arXiv preprint arXiv:2003.04887*, 2020.

Baker, B., Gupta, O., Naik, N., and Raskar, R. Designing neural network architectures using reinforcement learning. In *International Conference on Learning Representations*, 2017.

Bayer, J., Wierstra, D., Togelius, J., and Schmidhuber, J.

[6] Note that FLOPs may not be equivalent to latency. The latter depends on both the implementation and the hardware platform.

Evolving memory cell structures for sequence learning. In *International Conference on Artificial Neural Networks*, pp. 755–764. Springer, 2009.

Brock, A., Donahue, J., and Simonyan, K. Large scale gan training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2019.

Clevert, D.-A., Unterthiner, T., and Hochreiter, S. Fast and accurate deep network learning by exponential linear units (elus). *International Conference on Learning Representations*, 2016.

De, S. and Smith, S. L. Batch normalization biases deep residual networks towards shallow paths. *arXiv preprint arXiv:2002.10444*, 2020.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2): 182–197, 2002.

Goldberg, D. E. and Deb, K. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of genetic algorithms*, volume 1, pp. 69–93. Elsevier, 1991.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

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. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016a.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. In *European conference on computer vision*, pp. 630–645. Springer, 2016b.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969, 2017.

Hendrycks, D. and Gimpel, K. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in neural information processing systems*, pp. 6626–6637, 2017.

Hoffer, E., Banner, R., Golan, I., and Soudry, D. Norm matters: efficient and accurate normalization schemes in deep networks. In *Advances in Neural Information Processing Systems*, pp. 2160–2170, 2018.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 2015.

Karras, T., Aila, T., Laine, S., and Lehtinen, J. Progressive growing of gans for improved quality, stability, and variation. In *International Conference on Learning Representations*, 2018.

Klambauer, G., Unterthiner, T., Mayr, A., and Hochreiter, S. Self-normalizing neural networks. In *Advances in neural information processing systems*, pp. 971–980, 2017.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

Li, Z. and Arora, S. An exponential learning rate schedule for deep learning. In *International Conference on Learning Representations*, 2020.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. Microsoft coco: Common objects in context. In *European conference on computer vision*, pp. 740–755. Springer, 2014.

Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., and Belongie, S. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2117–2125, 2017.

Liu, H., Simonyan, K., Vinyals, O., Fernando, C., and Kavukcuoglu, K. Hierarchical representations for efficient architecture search. In *International Conference on Learning Representations*, 2018.

Loshchilov, I. and Hutter, F. Sgdr: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017.

Luo, P., Ren, J., Peng, Z., Zhang, R., and Li, J. Differentiable learning-to-normalize via switchable normalization. *International Conference on Learning Represenations*, 2019.

Ramachandran, P., Zoph, B., and Le, Q. V. Searching for activation functions. In *ICLR Workshop*, 2018.

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.

Real, E., Liang, C., So, D. R., and Le, Q. V. Automl-zero: Evolving machine learning algorithms from scratch. *arXiv preprint arXiv:2003.03384*, 2020.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. Improved techniques for training gans. In *Advances in neural information processing systems*, pp. 2234–2242, 2016.

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.

Singh, S. and Krishnan, S. Filter response normalization layer: Eliminating batch dependence in the training of deep neural networks. *arXiv preprint arXiv:1911.09737*, 2019.

Tan, M. and Le, Q. V. EfficientNet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, 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 Conference on Computer Vision and Pattern Recognition*, pp. 2820–2828, 2019.

Ulyanov, D., Vedaldi, A., and Lempitsky, V. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

Wu, Y. and He, K. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 3–19, 2018.

Zhang, H., Dauphin, Y. N., and Ma, T. Fixup initialization: Residual learning without normalization. In *International Conference on Learning Representations*, 2019.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 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.

A. Code Snippets in TensorFlow

The following pseudocode relies on broadcasting to make sure the tensor shapes are compatible.

BN-ReLU

```
def batchnorm_relu(x, gamma, beta, nonlinearity, training):
  mean, std = batch_mean_and_std(x, training)
  z = (x - mean) / std * gamma + beta
  if nonlinearity:
    return tf.nn.relu(z)
  else:
    return z
```

EvoNorm-B0 (use this to replace BN-ReLU)

```
def evonorm_b0(x, gamma, beta, nonlinearity, training):
  if nonlinearity:
    v = trainable_variable_ones(shape=gamma.shape)
    _, batch_std = batch_mean_and_std(x, training)
    den = tf.maximum(batch_std, v * x + instance_std(x))
    return x / den * gamma + beta
  else:
    return x * gamma + beta
```

EvoNorm-S0 (use this to replace BN-ReLU)

```
def evonorm_s0(x, gamma, beta, nonlinearity):
  if nonlinearity:
    v = trainable_variable_ones(shape=gamma.shape)
    num = x * tf.nn.sigmoid(v * x)
    return num / group_std(x) * gamma + beta
  else:
    return x * gamma + beta
```

Helper functions for EvoNorms

```
def instance_std(x, eps=1e-5):
  _, var = tf.nn.moments(x, axes=[1, 2], keepdims=True)
  return tf.sqrt(var + eps)

def group_std(x, groups=32, eps=1e-5):
  N, H, W, C = x.shape
  x = tf.reshape(x, [N, H, W, groups, C // groups])
  _, var = tf.nn.moments(x, [1, 2, 4], keep dims=True)
  return tf.reshape(tf.sqrt(var + eps), [N, H, W, C])

def trainable_variable_ones(shape, name="v"):
  return tf.get_variable(name, shape=shape,
            initializer=tf.ones_initializer())
```

B. Candidate layers without Reranking

B.1. Top-10 EvoNorm-B candidates

1. $\frac{x}{\max\left(\sqrt{s^2_{b,w,h}(x)},z\right)}\gamma + \beta$, $z = (x+1)\sqrt{\mu_{w,h}(x^2)}$

2. $\frac{x}{\max\left(\sqrt{s^2_{b,w,h}(x)},z\right)}\gamma + \beta$, $z = x + \sqrt{\mu_{w,h,c}(x^2)}$

3. $-\frac{x\sigma(x)}{\sqrt{s^2_{b,w,h}(x)}}\gamma + \beta$

4. $\frac{x}{\max\left(\sqrt{s^2_{b,w,h}(x)},z\right)}\gamma + \beta$, $z = x\sqrt{\mu_{w,h}(x^2)}$

5. $\frac{x}{\max(\sqrt{s^2_{b,w,h}(x)},z)}\gamma + \beta$, $z = \sqrt{\mu_{w,h,c}(x^2)} - x$

6. $\frac{x}{\max(\sqrt{s^2_{b,w,h}(x)},z)}\gamma + \beta$, $v_1 x + \sqrt{s^2_{w,h}(x)}$

7. $\frac{x}{\max(\sqrt{s^2_{b,w,h}(x)},z)}\gamma + \beta$, $z = \sqrt{\mu_{w,h}(x^2)} - x$

8. $\frac{x}{\max(\sqrt{s^2_{b,w,h}(x)},z)}\gamma + \beta$, $z = x\sqrt{\mu_{w,h}(x^2)}$ (duplicate)

9. $\frac{x}{\max(\sqrt{s^2_{b,w,h}(x)},z)}\gamma + \beta$, $z = x + \sqrt{s^2_{w,h}(x)}$

10. $\frac{x}{\max(\sqrt{s^2_{b,w,h}(x)},z)}\gamma + \beta$, $z = x + \sqrt{s^2_{w,h}(x)}$ (duplicate)

B.2. Top-10 EvoNorm-S candidates

1. $\frac{x\tanh(\sigma(x))}{\sqrt{\mu_{w,h,c/g}(x^2)}}\gamma + \beta$

2. $\frac{x\sigma(x)}{\sqrt{\mu_{w,h,c/g}(x^2)}}\gamma + \beta$

3. $\frac{x\sigma(x)}{\sqrt{\mu_{w,h,c/g}(x^2)}}\gamma + \beta$ (duplicate)

4. $\frac{x\sigma(x)}{\sqrt{\mu_{w,h,c/g}(x^2)}}\gamma + \beta$ (duplicate)

5. $\frac{x\sigma(x)}{\sqrt{s^2_{w,h,c/g}(x)}}\gamma + \beta$

6. $\frac{x\sigma(v_1 x)}{\sqrt{s^2_{w,h,c/g}(x)}}\gamma + \beta$

7. $\frac{x\sigma(x)}{\sqrt{s^2_{w,h,c/g}(x)}}\gamma + \beta$ (duplicate)

8. $\frac{x\sigma(x)}{\sqrt{\mu_{w,h,c/g}(x^2)}}\gamma + \beta$ (duplicate)

9. $\frac{x\sigma(x)}{\sqrt{s^2_{w,h,c/g}(x)}}\gamma + \beta$ (duplicate)

10. $z\sigma(\max(x,z))\gamma + \beta$, $z = \frac{x}{\sqrt{\mu_{w,h,c/g}(x^2)}}$