AutoDropout: Learning Dropout Patterns to Regularize Deep Networks

Hieu Pham and **Quoc V. Le**
Google Research – Brain Team, Mountain View, CA 94043
{hyhieu,qvl}@google.com

Abstract

Neural networks are often over-parameterized and hence benefit from aggressive regularization. Conventional regularization methods, such as Dropout or weight decay, do not leverage the structures of the network's inputs and hidden states. As a result, these methods are less effective than recent methods that leverage the structures, such as SpatialDropout and DropBlock , which randomly drop the values at certain contiguous areas in the hidden states and setting them to zero. Although the locations of dropout areas are random, the patterns of SpatialDropout and DropBlock are manually designed and fixed. Here we propose *AutoDropout*, which automates the process of designing dropout patterns. In our method, a controller learns to generate a dropout pattern at every channel and layer of a target network, such as a ConvNet or a Transformer. The target network is then trained with the dropout pattern, and its resulting validation performance is used as a signal for the controller to learn from. We show that this method works well for both image recognition on CIFAR-10 and ImageNet, as well as language modeling on Penn Treebank and WikiText-2. The learned dropout patterns also transfers to different tasks and datasets, such as from language model on Penn Treebank to Engligh-French translation on WMT 2014. Our code will be available at: https://github.com/google-research/google-research/tree/master/auto_dropout.

Introduction

Modern neural networks are often over-parameterized (Nakkiran et al. 2020) and thus require proper regularization to avoid overfitting. A common regularization method is Dropout (Srivastava et al. 2014), which randomly selects neurons from some intermediate layers of a network and replaces the values of these neurons with zero. In other words, we drop these neurons out of the current step of training. More recent studies show that imposing certain structures to the dropped neurons can lead to significant improvements over dropout neurons uniformly at random (Huang et al. 2016; Tompson et al. 2020; Ghiasi, Lin, and Le 2018; Gal and Ghahramani 2016b; Zoph et al. 2018; Zaremba, Sutskever, and Vinyals 2014; Vaswani et al. 2017). In practice, however, the dropout patterns are adapted to become different for different applications.

For example, in the text domain, Zaremba, Sutskever, and Vinyals (2014) suggest that, for a multi-layered LSTM (Hochreiter and Schmidhuber 1997), it is better to only drop the neurons in the vertical connections than to drop the neurons everywhere. Gal and Ghahramani (2016b) later propose Variational Dropout, where they drop neurons everywhere in the network but share a dropout pattern along the temporal dimension. Both methods, however, are not used in the recent Transformer architecture, which only uses vanilla Dropout. The differences in how LSTM and Transformer implement Dropout suggest that dropout patterns need to be tailored to different model architectures in NLP.

In the image domain, vanilla Dropout is often only applied to the fully-connected layers within a ConvNet (He et al. 2016; Zagoruyko and Komodakis 2016; Han, Jiwhan, and Kim 2017; Hu et al. 2018; Szegedy et al. 2016). Other convolutional layers often require the dropout neurons to have particular structures. For example, stochastic depth (Huang et al. 2016) drops the whole residual branch in residual networks, and DropPath (Zoph et al. 2018) drops a whole branch in multi-branched convolutional cells. Ghiasi, Lin, and Le (2018) propose DropBlock which drop contiguous squares of neurons in the convolutional layers. While DropBlock works well on ResNet-50 and AmoebaNet (Real et al. 2018), it is not proven to be successful in more recent architectures such as EfficientNet (Tan, Pang, and Le 2020) and EfficientDet (Tan, Pang, and Le 2020; Zoph et al. 2019). Again, the differences in the way ConvNet architectures use dropout patterns suggest that they also need to be specialized to architectures.

By studying the dropout patterns from previous works, we observe that these patterns are difficult to design and need to be specialized for each model architecture, task, and domain. In this work, we address this difficulty by learning a specialized pattern for each model architecture, task, and domain. To this end, we propose *AutoDropout* which automates the process of designing specialized dropout patterns. The main contribution of *AutoDropout* is a novel search space of structured dropout patterns. In the search space we design, one can find a suitable for each model architecture and task. Our search space generalizes many existing dropout patterns (Srivastava et al. 2014; Gal and Ghahramani 2016a,b; Huang et al. 2016; Ghiasi, Lin, and Le 2018). For example, Figure 1 shows a dropout pattern from our search space. The pattern is generated by tiling a contiguous area and then transforming it geometrically. The resulting pattern is applied to a convolutional output channel, which is a common building block of image recognition models.

Our implementation of *AutoDropout* has a controller that is trained by reinforcement learning (RL). The reward for the RL is the validation performance of the dropout pattern on a target network on a dataset of interest. We design a distributed RL-based search algorithm, which allows us to maximally leverage all machines available on an arbitrary cluster of computational nodes.[1]

Our experiments show that *AutoDropout* can find dropout patterns that significantly improve commonly-used ConvNet and Transformer architectures. On ImageNet, *AutoDropout* improves the top-1 accuracy of ResNet-50 from 76.5% to 78.7%, and EfficientNet-B7 from 84.1% to 84.7%. In the semi-supervised setting with CIFAR-10-4000, *AutoDropout* also improves the accuracy of Wide-ResNet-28-2 from 94.9% to 95.8%. For language modeling, *AutoDropout* reduces the perplexity of Transformer-XL (Dai et al. 2019) on Penn Treebank from 56.0 to 54.9.

Additionally, when transferred to German-to-English translation on the IWSLT 14 dataset, the dropout pattern found by *AutoDropout* improves Transformer's BLEU score from 34.4 to 35.8, which is a new **state-of-the-art on this dataset**. On English-to-French translation with WMT 2014, the transferred dropout pattern also yields an improvement of 1.9 BLEU scores over the Transformer model with vanilla Dropout.

Although the search cost of AutoDropout can be high, a simple use case of AutoDropout is to drop our found patterns into existing pipelines in the same way that AutoAugment policies (Cubuk et al. 2019a) were used to improve state-of-the-art models.

Related works. Our work has the same philosophy with existing neural architecture search and AutoAugment lines of research (Pham et al. 2018; Liu, Simonyan, and Yang 2019; Zoph and Le 2017; Zoph et al. 2018; Bello et al. 2017b; Cubuk et al. 2019a; Park et al. 2019; Lim et al. 2019; Tan and Le 2019; Real et al. 2018; Mirhoseini et al. 2017; Bello et al. 2017a; Cubuk et al. 2019b; Real et al. 2017; Cai, Zhu, and Han 2019; Cai et al. 2019). We create a search space comprising the possible decisions and then use RL to search for the best decision.

More specifically, *AutoDropout* can also be viewed as data augmentation in the networks' hidden states. We generalize the successful approaches of searching for data augmentation (Park et al. 2019; Cubuk et al. 2019a,b; Lim et al. 2019) and apply them to the hidden states of ConvNets and Transformer networks. Unlike data augmentations, which are domain-specific, our dropout patterns for the hidden states have the same design philosophy on ConvNets for image recognition models and Transformer for text understanding models. CutMix (Yun et al. 2019) and ManifoldMixup (Verma et al. 2019a) also apply successful data augmentation techniques such as CutOut (DeVries and Taylor 2017) and Mixup (Zhang et al. 2018) into the hidden states. Implicit Semantic Data Augmentation (ISDA; Wang et al. (2019)) approximate a Gaussian distribution of ConvNets' hidden states using the moving averages of their mean and standard deviations to generate more training examples.

Methods

Representing dropout patterns. We represent the dropout patterns in our search space using elementwise multiplicative masks as adopted by many previous works (Srivastava et al. 2014; Gal and Ghahramani 2016a,b; Huang et al. 2016; Zoph et al. 2018; Ghiasi, Lin, and Le 2018; Vaswani et al. 2017). To bridge the gap between training, when the mask is used, and inference, when the mask is not used, we scale the values of the non-drop neurons properly during training. Specifically, to apply a dropout pattern to a layer h of a neural network, we randomly generate a binary mask m of the same shape with h. We then scale the values in the mask m, and replace h with:

$$\text{Drop}(h, m) \triangleq h \otimes \left(\frac{\text{Size}(m)}{\text{Sum}(m)} \cdot m\right) \quad (1)$$

Dimensional notations. In modern deep learning frameworks (Abadi et al. 2016; Paszke et al. 2019), intermediate layers are represented as high dimensional tensors. We denote the general shape of a tensor as $(N, d_1, d_2, ..., d_k, C)$, where N is the *batch* dimension, C is the *feature* dimension, and d_1, d_2, ..., d_k are the *spatiotemporal* dimensions. For instance, a layer in a typical ConvNet has a shape of (N, H, W, C) where H and W are the layer's height and width; while a Transformer layer has the output of shape (N, T, C) where T is the temporal dimension which represents the number of tokens.

Our method is general and works well for both ConvNets and Transformers where the spatiotemporal dimensions are different from each other. In the following, we will first discuss the search space for ConvNets, and then discuss how we generalize it to Transformers.

Search Space for Dropout Patterns in ConvNets

Basic patterns. The basic pattern in our search space is a contiguous rectangle. The rectangle is then tiled to produce a dropout pattern. For ConvNets, the hyper-parameters that define the basic rectangle are two sizes height and width. The hyper-parameters that define the tiling are the stride and the number of repeats. Figure 2 shows an example. For C channels, we can either sample C independent dropout patterns, or we can sample only one dropout pattern and then **share** it along the feature dimension.

Geometric transformations. In addition to tiling the rectangles, we introduce two geometric transformations into our search space: rotating about the spatial center, and shearing along each spatial dimension. When the transformations result in fractional coordinates, we round them to the nearest integers.

[1] We will release the datasets consisting of the dropout patterns that our search algorithm has sampled and run. Like the similar datasets collected from benchmarking various model architectures (Ying et al. 2019; Dong and Yang 2019).

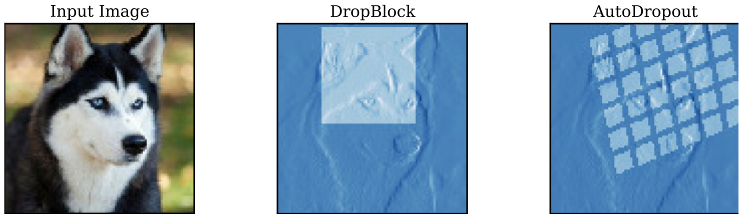

Figure 1: An example dropout pattern from our search space applied to an convolutional output channel. Left: the input image. Middle: DropBlock sets contiguous square blocks in the channel to zero. Right: a dropout pattern in the search space of *AutoDropout*. More patterns in our noise space are described in Section "Methods".

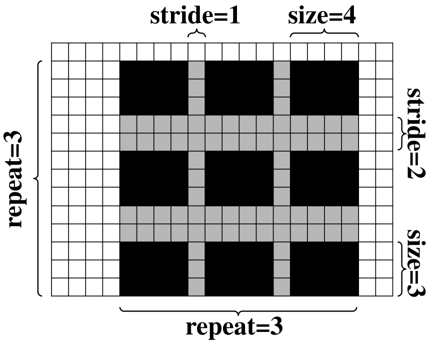

Figure 2: Example of the basic patterns in our search space. A dropout pattern, represented in black and gray, is applied to a grid of while cells representing the tensors. The neuron corresponding to the gray cells are retained.

Where to apply the dropout pattern. Once we have a dropout pattern, there is a decision about where we should apply it to. Here, we apply the dropout pattern to the output of batch normalization layers because we empirically observe that applying the pattern elsewhere in the network often leads to unstable training during our search process. If there is a **residual** connection in the ConvNet to regularize, then there is a choice of whether we should apply the dropout pattern to the residual branch as well. We leave this decision to the controller.

Controller Model and Search Algorithms

Model Architecture and Search Algorithm. We parameterize our controller with a Transformer network, as illustrated in Figure 3. We train the parameters θ of our controller using the REINFORCE algorithm with a standard moving average baseline (Williams 1992). That is, we optimize θ to minimize the objective via Monte Carlo gradient estimation:

$$J(\theta) = \mathbb{E}_{r \sim P(\mathbf{r};\theta)}[\text{Perf}(r)]$$
$$\widehat{\nabla}_\theta J = \frac{1}{M}\sum_{i=1}^{M}(\text{Perf}(r_i) - b) \cdot \nabla_\theta \log P(r_i;\theta) \quad (2)$$

Here, b is the moving average baseline, M is the empirical batch size which we set to 16, and Perf(r) is measured by training a target network with a dropout pattern r on a designated proxy task's validation set. We find it important to tailor the proxy task according to the actual downstream task. We discuss our proxy tasks in detailed in our Experiments section.

Improving Parallelism. Previous works in architecture search and data augmentation search (Zoph and Le 2017; Zoph et al. 2018; Bello et al. 2017b; Cubuk et al. 2019a; Park et al. 2019; Tan and Le 2019) typically wait for minibatches of M dropout patterns to finish training before making every update on θ. Since each child model can take significantly long to train, and is subjected to multiple failures, such as jobs being descheduled on a shared cluster, waiting for M dropout patterns to finish can cause an unnecessary bottleneck.

To alleviate this bottleneck, we propose a modification. Specifically, in a shared environment, the number of available machines will vary over time. Sometimes, the number of machines will be lower than M. In this case, we will have to use this low number of machines to slowly compute the rewards for M configurations. However, sometimes the number of machines will be much higher than M. In such case, we want to generate many more than M jobs to take advantage of the available resources. But even in such case, for training stability, we only use a minibatch of M configurations, causing the other trained configurations to have stale gradient. To adjust for the staleness of their gradients, we need to reweigh the gradient properly as explained later.

Our implementation maintains two queues: a queue $q_{\text{unfinished}}$ of unfinished jobs and a queue q_{finished} of finished jobs. Whenever the $q_{\text{unfinished}}$ contains less than its capacity C, the controller generates $n = C - |q_{\text{unfinished}}|$ new dropout patterns r_1, r_2, ..., r_n and fills up $q_{\text{unfinished}}$ with the pairs $(r_i, P(r_i;\theta_i))$, where θ_i is the value of the controller's parameters at the time r_i is sampled.

On the other hand, whenever a dropout pattern r finishes training, the controller dequeues $(r, \text{Perf}(r))$ from $q_{\text{unfinished}}$ and moves it into q_{finished}. Whenever the capacity $|q_{\text{finished}}|$ reaches M, M configurations along with their accuracy are dequeued from q_{finished} to perform an update on θ. The caveat of this approach is that due to many dropout patterns being executed in parallel, the controller parameter θ when we update the controller with a configuration r_i can be different from the θ_i when r_i was generated. To account for this difference, we resort to importance sampling, which allows us to

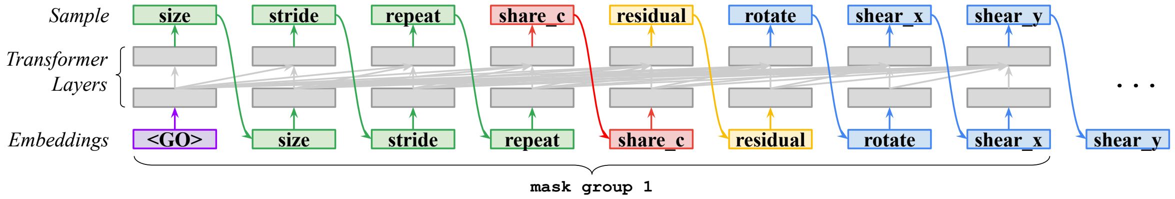

Figure 3: Our controller is a Transformer network. The network generates the tokens to describe the configurations of the dropout pattern. The tokens are generated like words in a language model. For every layer in a ConvNet, a group of 8 tokens need to be made to create a dropout pattern. These 8 tokens are generated sequentially. In the figure above, `size`, `stride`, and `repeat` indicate the size and the tiling of the pattern; `rotate`, `shear_x`, and `shear_y` specify the geometric transformations of the pattern; `share_c` is a binary deciding whether a pattern is applied to all C channels; and `residual` is a binary deciding whether the pattern is applied to the residual branch as well. If we need L dropout patterns, the controller will generate $8L$ decisions.

write the training objective $J(\theta)$ as follows:

$$\begin{aligned}\nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_{r\sim P(\mathbf{r};\theta)}[\text{Perf}(r)] \\ &\approx \frac{1}{M}\sum_{i=1}^{M} \text{Perf}(r_i)\cdot\frac{P(r_i;\theta)}{P(r_i;\theta_i)}\cdot\nabla_\theta \log P(r_i;\theta)\end{aligned} \quad (3)$$

Implementing this update rule simply requires scaling the gradient $\nabla_\theta \log P(r_i;\theta)$ by the ratio of the two probabilities as shown in Equation 3. In our design, the only training bottleneck is the number of workers that can be run in parallel. In practice, distributed search procedures like ours typically run on a shared cluster, where the number of available workers varies instantly. Our design obviates the need to reserve all C workers throughout the search procedure and allows us to use a large value of C to achieve better parallelism when more workers are available.

Search Space for Dropout Patterns in Transformers

Basic patterns. Intermediate layers in Transformer models typically have three dimensions (N, T, C), where N and C are the batch dimension and the channel dimension, similar to those of ConvNets, and T is the number of tokens, such as words or sub-word units. The dropout pattern for this dimension T is realized by generating four hyper-parameters: `size`, `stride`, `share_t`, and `share_c`. `size` indicates how many tokens does a pattern affects; `stride` indicates the number of tokens to be skipped by the pattern; `share_t` is a binary deciding whether all the tokens covered by `size` are set to zero using the same noise mask or independent noise masks; and `share_c` is a binary deciding whether a the dropout pattern shared along the channel dimension C. Once the values of `size`, `stride`, `share_t`, and `share_c` are decided, at each training step, we sample the starting position to apply the resulting dropout pattern. We repeat the pattern until the end of the sequence, following `size` and `stride`. Figure 4 provides an illustration of a dropout pattern that our controller samples from our search space, and how the pattern is applied to a sequence of words.

Many successful regularization patterns for text processing models are included in our basic patterns. For instance, WordDropout (Sennrich, Haddow, and Birch 2016) can be realized from our patterns by setting `share_c=True`, while Variational Dropout (Gal and Ghahramani 2016b) can be realized by setting `share_t=True` and setting *size* to the T, number of tokens in the sequence.

Where to apply the dropout pattern. Unlike the case for image recognition models, we find that the dropout patterns in our search space can be flexibly applied at multiple sublayers within a Transformer layer (e.g., on the query, key, value, softmax, output projection, and residual). As a result, we apply one independent dropout pattern to each of them. We will use this pattern at all Transformer layers in the Transformer network. In our implementation, `size` is overloaded, and if it has the value of zero, the dropout pattern is not applied at the corresponding.

Experiments

In the following sections, we will apply *AutoDropout* to both ConvNets and Transformers. For ConvNets, we first consider supervised image classification and semi-supervised image classification. For Transformer, we consider language model and machine translation applications. Finally, we compare our search method against random search.

Supervised Image Classification with ConvNets

We first evaluate *AutoDropout* on two standard benchmarks for image classification: CIFAR-10 (Krizhevsky 2009) and ImageNet (Russakovsky et al. 2015). For CIFAR-10, we use Wide ResNet 28-10 (WRN-28-10; Zagoruyko and Komodakis (2016)) because it is a common baseline on this dataset. For ImageNet, we consider ResNet-50 (He et al. 2016) because it is a common architecture for ImageNet. We also consider EfficientNet (Tan and Le 2019) since it is closer to the state-of-the-art than ResNet. For each benchmark and model, we first use *AutoDropout* to search for a good dropout pattern on a proxy task, and then scale up the best found pattern to apply to the final task.

Search Configurations and Proxy Tasks. We refer readers to our Appendix "Hyper-parameters of Experiments" for detailed training information of our controller. Here, we focus on the proxy tasks that we design to reduce the *AutoDropout*'s search time. We scale down the final architecture and reduce the amount of data for the final task as follows.

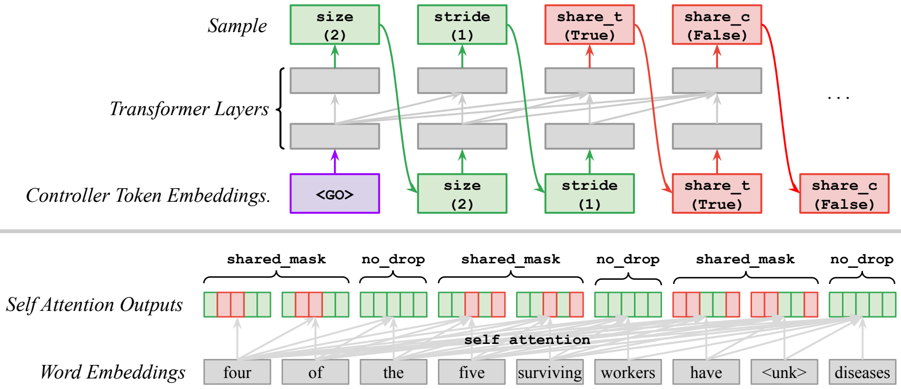

Figure 4: An example of our controller generating ad dropout pattern for a self-attention operation. Top: the controller's outputs. The tokens have the following meanings: `size=2` and `stride1` means that the dropout pattern affects two consecutive tokens, then skips two token, then affects the next two consecutive tokens, and so on; `share_t=True` means that every block of two consecutive tokens that the dropout pattern affects shares the same dropout mask; `share_c=False` means that each of the C feature dimensions of the (N, T, C) tensor has its own independent mask. Bottom: The dropout pattern that the controller's outputs realize on the self-attention operation. The values in the red cells are set to zero, while the values in the green are kept intact.

For CIFAR-10, we search with a WRN-28-2 on the entire dataset, reserving 10% of the original training set for validation. For ImageNet, we scale down ResNet-50 and EfficientNet-B0 so that each of their layers has half the number of channels as the original models. We use 80,000 examples for training and 5,000 examples for validation. The controller's reward is the accuracy of the dropout pattern on the validation set. We train each dropout pattern on CIFAR-10 for 32,000 steps, and train each pattern on ImageNet for 16,000 steps. Under these settings, each dropout pattern trains in approximately 40 minutes on both datasets. Our search explores 16,384 patterns for each task.

Baselines. For WRN-28-10 and ResNet-50, we compare *AutoDropout* against DropBlock (Ghiasi, Lin, and Le 2018), since DropBlock has been well-tuned for these models. For EfficientNet, we compare *AutoDropout* with Stochastic Depth (Huang et al. 2016) since it is the default noise-based regularization scheme of this architecture. We implement these baselines in our environment for fair comparison. Note that large EfficientNet models, such as B3, B5, B7 in our experiments, enlarge the spatial dimensions of the input images. For these models, we proportionally scale up the sizes and strides of the masks found by *AutoDropout* on these models. Training details of all models are in our Appendix "Hyper-parameters of Experiments".

Results. Figure 1 reports the results of our control experiments on ResNets and EfficientNet. From Table 1, it can be seen that *AutoDropout* outperforms DropBlock by 0.6% accuracy on CIFAR-10 with WRN-28-10, which corresponds to a 16% error reduction. Notably, on CIFAR-10 with WRN-28-10, DropBlock does not yield significant improvements compared to not using regularization at all, suggesting that the intuition on blocking contiguous regions is not sufficient. On ImageNet, *AutoDropout* improves the top-1 accuracy of ResNet-50 on ImageNet by 0.4% compared to DropBlock. *AutoDropout* improves the accuracy of all EfficientNet models by a margin of 0.7% on average. This is larger than the improvement of 0.5% that DropBlock delivers on AmoebaNet (Ghiasi, Lin, and Le 2018; Real et al. 2018), even though EfficientNet baselines have higher accuracy than AmoebaNet.

Pushing the limits of ResNets. In the above experiments, we wanted to perform fair comparisons against other baselines, and did not combine *AutoDropout* with any data augmentation methods. Here, we aim to push the limits of WRN-28-10 and ResNet-50 by combining *AutoDropout* and other data augmentation methods. As such, we apply the pattern found by *AutoDropout* on CIFAR-10 with RandAugment (Cubuk et al. 2019b) to WRN-28-10 and achieve 97.9% accuracy. We also apply the pattern found by *AutoDropout* on ImageNet with RandAugment and achieve 80.3% top-1 accuracy with ResNet-50 on ImageNet. These results outperform existing state-of-the-art results on these datasets with the same model architectures, as presented in Table 2.

Table 2 also shows that *AutoDropout* is the only method that improves the performance on both CIFAR-10 with WRN-28-10 and ImageNet with ResNet-50. Among other baselines, Manifold Mixup (Verma et al. 2019a) improves the CIFAR-10 accuracy but has a weak accuracy on ImageNet. Meanwhile, CutMix (Yun et al. 2019) achieves good accuracy on ImageNet but worsens CIFAR-10 accuracy. These observations suggest that regularization methods that are validated for a certain architecture and dataset might not deliver as strong performance for another architecture and dataset, necessitating automated designing procedures like *AutoDropout*.

Qualitative analysis of good dropout patterns. *AutoDropout* finds several patterns that are unexpected. For example, the best noise pattern found for ResNet-50 on ImageNet

Regularization	CIFAR-10	ImageNet				
Methods	WRN-28-10	ResNet-50	EfficientNet-B0	EfficientNet-B3	EfficientNet-B5	EfficientNet-B7
None	96.1 ± 0.12	76.5 / 93.4	76.2 / 92.9	–	–	–
DropBlock (Ghiasi, Lin, and Le 2018)	96.2 ± 0.07	78.3 / 94.3	76.3 / 92.8	–	–	–
Stochastic Depth (Huang et al. 2016)	96.2 ± 0.07	77.5 / 93.7	76.8 / 93.1	80.2 / 95.0	82.5 / 96.2	84.1 / 96.9
AutoDropout	**96.8 ± 0.09**	**78.7 / 94.3**	**77.5 / 93.8**	**80.9 / 95.6**	**83.1 / 96.5**	**84.7 / 97.1**

Table 1: Performance of *AutoDropout* and the baselines on supervised image classification (higher is better). This is a control experiment and all models are implemented by us.

Methods	CIFAR-10 (WRN-28-10)	ImageNet (ResNet-50)
Stochastic Depth (2016)	96.2 ± 0.07†	77.5 / 93.7
DropPath (2017)	95.4	77.1 / 93.5
Manifold Mixup (2019a)	97.5 ± 0.02	77.5 / 93.8
Mixup (2018)	97.1 ± 0.08†	77.9 / 93.9
CutMix (2019)	96.7 ± 0.05†	78.6 / 94.1
MoEx (2020)	96.7 ± 0.03	79.1 / 94.3
CutMix+RandAugment (2019b)	97.0 ± 0.06†	78.3 / 94.2†
CutMix+FixRes (2019)	n/a	79.8 / 94.9
AutoDropout+RandAugment	**97.9 ± 0.06**	**80.3 / 95.1**

Table 2: Performance of *AutoDropout* and representative baselines on supervised image classification (higher is better). (†) *denotes our implementation. CutMix+FixRes is not applicable for CIFAR-10 since we keep the image resolution at 32x32 for CIFAR-10.*

Methods	CIFAR-10-4K (WRN-28-2)	ImageNet-10% (ResNet-50)
LGA+VAT (2019)	87.9 ± 0.19	–
ICT (2019b)	92.7 ± 0.02	–
MixMatch (2019)	93.8 ± 0.06	–
ReMixMatch (2020)	94.9 ± 0.04	–
LLP (2020)	–	– / 88.5
SimCLR (2020)	–	69.3 / 89.0
FixMatch (2020)	**95.7 ± 0.05**	71.5 / 89.1
UDA (2019a)	94.9 ± 0.12	68.8 / 88.8
UDA+*AutoDropout*	**95.8 ± 0.04**	**72.9 / 91.4**

Table 3: Performance of *AutoDropout* and representative baselines on semi-supervised image classification (higher is better).

only injects noise into the first and the last bottleneck convolutional blocks. These two blocks also have different noise patterns. This behavior is different from DropBlock (Ghiasi, Lin, and Le 2018), where a fixed and predefined mask of size 7x7 is applied at every layer. Additionally, rotation is applied in the first block, but not in the last block, suggesting that *AutoDropout* finds that rotational invariance should be enforced at the first block, where most low-level feature extracting happens, rather than in the last block, where most features have become more abstract. To validate the decisions of *AutoDropout*, we vary the locations where the dropout patterns are applied and observe about 1% drop in top-1 accuracy, which is significant for ResNet-50 on ImageNet.

Semi-supervised Image Classification with ConvNets

Experiment Settings. We now consider two typical benchmarks for semi-supervised image classification: CIFAR-10 with 4,000 labeled examples and ImageNet with 10% labeled examples. Since our search procedure of *AutoDropout* on ImageNet, as described in the previous section, uses a subset of images in ImageNet-10%, we simply take the same dropout patterns found in that setting. We make sure that ImageNet-10% contain the 80,000 images that we perform the search on. On CIFAR-10, we repeat our *AutoDropout* search with 3,600 training examples and 400 validation examples.

Baselines and Results. We apply *AutoDropout* into Unsupervised Data Augmentation (UDA; Xie et al. (2019a)), since UDA has a simple implementation. As shown in Table 3, the dropout patterns found by *AutoDropout* improves UDA by 0.9% on CIFAR-10 and 4.1% Top-1 accuracy on ImageNet. Here we compare against recent representative strong baselines and skip earlier works such as (Tarvainen and Valpola 2017; Miyato et al. 2018; Lee 2013).

Language Model and Machine Translation

In this section, we first apply *AutoDropout* to regularize Transformer-XL (Dai et al. 2019) the task of language model on the Penn Treebank dataset (PTB; Marcus et al. (1994)). PTB is a small dataset, with about 929K training tokens, 73K validation tokens, and 82K test tokens, on a vocabulary size of 10K. The small size of PTB makes it a suitable testbed for *AutoDropout*.

Search Configuration. We use the search space for Transformer models as described in the Section "Search Space for Dropout Patterns inTransformers". Every dropout pattern that our controller sampled is employed to regularize a training process of Transformer-XL (Dai et al. 2019). We use the same model size as specified by Dai et al. (2019). We train every configuration from scratch for 160,000 steps, using a batch size of 16 and a segment length of 70. We use the cosine learning rate schedule so that each trial converges to a reasonable perplexity. On 4 TPU v2 chips, each of our runs takes about 40 minutes. The performance of a configuration r is computed by $\text{Perf}(r) = 80/\text{ValidPPL}(r)$.

Results. Our results for Transformer models are reported in Table 4. Once again, hyper-parameters for each experiment

Methods	PennTreebank		WikiText-2		IWSLT-14 DeEn	WMT-14 EnFr
	ValidPPL (↓)	TestPPL (↓)	ValidPPL (↓)	TestPPL (↓)	BLEU (↑)	BLEU (↑)
Transformer / Transformer-XL	59.1	56.0	63.9	61.4	34.4	38.1
+*AutoDropout*	**58.1**	**54.9**	**62.7**	**59.9**	**35.8**	**40.0**

Table 4: Performance of Transformer and Transformer-XL models trained with default regularization techniques vs. trained with *AutoDropout*. For PTB and WikiText-2, we report the model's perplexity (lower is better). For IWSLT-14-DeEn and WMT-14-EnFr, we report BLEU scores (higher is better).

are reported in our Appendix "Hyper-parameters of Experiments". First, we take the dropout pattern that achieves the lowest perplexity on PTB and train for 300,000 steps. We compare our results with Variational Dropout, which is originally used by Transformer-XL (Dai et al. 2019). Under this setting, *AutoDropout* outperforms Variational Dropout by 1.1 perplexity, which is a significant improvement on this dataset.

Transfer learning results. To test the transferability of the found pattern, we also transfer it to three other tasks: 1) language modeling on WikiText-2 (Merity et al. 2017), 2) German-English translation on the IWSLT-14 dataset, and 3) English-French translation on the WMT-14 dataset. On WikiText-2, we compare *AutoDropout*'s dropout pattern against Variational Dropout because we find that it works better than vanilla Dropout on this task. On translation tasks, we compare *AutoDropout*'s dropout pattern against the vanilla Dropout configurations that are typically applied in Transformer models (Vaswani et al. 2017).

Qualitative analysis of the *AutoDropout*'s dropout pattern. For Transformer models, *AutoDropout* assigns different sizes and strides at different sub-layers in a Transformer layer. For instance, in our best dropout pattern *AutoDropout* learns that the pattern for the multi-head attention layer is similar to Variational Dropout (Gal and Ghahramani 2016b), but the pattern for the positional feed-forward layer follows word dropout (Sennrich, Haddow, and Birch 2016). To validate that such decision is beneficial, we try to apply Variational Dropout in all layers of Transformer-XL and got the resulting validation perplexity of 59.8, which is 1.7 point higher than the configuration found by *AutoDropout*.

Comparison with Random Search

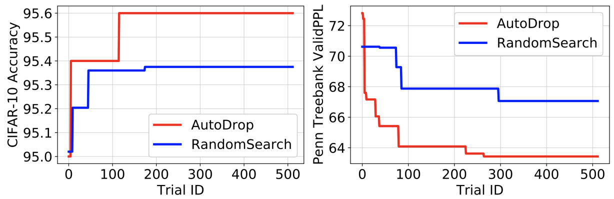

Figure 5: Best-so-far performances of the first 512 dropout patterns sampled by *AutoDropout* and by random search. Top: Accuracy on CIFAR-10 (higher is better); Bottom: ValidPPL on PennTreebank (lower is better).

Recent works on neural architecture search (Li and Talwalkar 2019) show that random search is a strong search baseline. Here we perform a controlled experiment to verify the advantage of *AutoDropout*'s search process over random search. To this end, we sample 512 uniformly random patterns from the search space for WRN-28-2 on CIFAR-10 and another 512 uniformly random patterns from the search space for Transformer-XL on PTB. We train each of these patterns to convergence, and compare the results against training *the first* 512 patterns suggested by *AutoDropout* under the same settings. In Figure 5, we plot the best-so-far performances of both methods, and observe substantial differences between *AutoDropout* and random search. Specifically, on CIFAR-10, the best patterns found by *AutoDropout* is more than 0.2% accuracy above that of Random Search. Recall that from Table 1, we know that the standard deviation of CIFAR-10 accuracy in our code base is less than 0.1%. This means that *AutoDropout* is more than 2x standard deviations away from random search and makes the difference significant. On PTB, the difference between *AutoDropout* and Random Search is more than 3 validation perplexity points, which is also significant for the dataset. We thus conclude that when searching for structured noise to regularize deep networks, RL search exhibits significant advantage compared to Random Search.

Conclusion and Future Directions

We proposed *AutoDropout*, an algorithm to automatically design dropout patterns to regularize neural networks. Our algorithm successfully found dropout patterns that improve the performance of various ConvNets for image classification, as well as Transformer models for language modeling and machine translation. Currently, a weakness of *AutoDropout* is that the method is computationally expensive. Therefore, a potential future direction is to develop more efficient search approaches, similar to the developments on architecture search (Pham et al. 2018; Liu, Simonyan, and Yang 2019; Cai, Zhu, and Han 2019; Liu et al. 2017) and data augmentation search (Lim et al. 2019; Wang et al. 2019; Cubuk et al. 2019b).

Although the search cost of *AutoDropout* can be high, a simple use case of AutoDropout is to reuse our found patterns in the same way that AutoAugment policies (Cubuk et al. 2019a) were used to improve state-of-the-art models. To date, the method of reusing the found AutoAugment (Cubuk et al. 2019a) and RandAugment (Cubuk et al. 2019b) policies has benefitied many state-of-the-art models on CIFAR-10 and ImageNet (e.g., Tan and Le (2019); Xie et al. (2019b); Ridnik et al. (2020); Foret et al. (2020)).

References

Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; Kudlur, M.; Levenberg, J.; Monga, R.; Moore, S.; G, Derek . Murrayand Steiner, B.; Tucker, P.; Vasudevan, V.; Warden, P.; Wicke, M.; Yu, Y.; and Zheng, X. 2016. TensorFlow: A system for large-scale machine learning. *Arxiv 1605.08695* .

Bello, I.; Pham, H.; Le, Q. V.; Norouzi, M.; and Bengio, S. 2017a. Neural Combinatorial Optimization with Reinforcement Learning. In *International Conference on Learning Representations Workshop*.

Bello, I.; Zoph, B.; Vasudevan, V.; and Le, Q. V. 2017b. Neural optimizer search with reinforcement learning. In *International Conference on Machine Learning*.

Berthelot, D.; Carlini, N.; Cubuk, E. D.; Kurakin, A.; Sohn, K.; Zhang, H.; and Raffel, C. 2020. ReMixMatch: Semi-Supervised Learning with Distribution Alignment and Augmentation Anchoring. In *International Conference on Learning Representations*.

Berthelot, D.; Carlini, N.; Goodfellow, I.; Papernot, N.; Oliver, A.; and Raffel, C. 2019. MixMatch: A Holistic Approach to Semi-Supervised Learning. In *Advances in Neural Information Processing Systems*.

Cai, H.; Lin, J.; Lin, Y.; Liu, Z.; Wang, K.; Wang, T.; and Zhu, Ligeng Han, S. 2019. AutoML for Architecting Efficient and Specialized Neural Networks. In *IEEE Micro*.

Cai, H.; Zhu, L.; and Han, S. 2019. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In *International Conference on Learning Representations*.

Chen, T.; Kornblith, S.; Norouzi, M.; and Hinton, G. 2020. A Simple Framework for Contrastive Learning of Visual Representations. In *International Conference on Machine Learning*.

Cubuk, E. D.; Zoph, B.; Mane, D.; Vasudevan, V.; and Le, Q. V. 2019a. AutoAugment: Learning Augmentation Policies from Data. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Cubuk, E. D.; Zoph, B.; Shlens, J.; and Le, Q. V. 2019b. RandAugment: Practical data augmentation with no separate search. *Arxiv 1909.13719* .

Dai, Z.; Yang, Z.; Yang, Y.; Carbonell, J.; Le, Q. V.; and Salakhutdinov, R. 2019. Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context. In *Annual Meeting of the Association for Computational Linguistics*.

DeVries, T.; and Taylor, G. W. 2017. Improved Regularization of Convolutional Neural Networks with Cutout. *Arxiv, 1708.04552* .

Dong, X.; and Yang, Y. 2019. NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search. In *International Conference on Learning Representations*.

Foret, P.; Kleiner, A.; Mobahi, H.; and Neyshabur, B. 2020. Sharpness-Aware Minimization for Efficiently Improving Generalization. *Arxiv, 2010.01412* .

Gal, Y.; and Ghahramani, Z. 2016a. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In *International Conference on Machine Learning*.

Gal, Y.; and Ghahramani, Z. 2016b. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. In *Advances in Neural Information Processing Systems*.

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

Han, D.; Jiwhan, K.; and Kim, J. 2017. Deep Pyramidal Residual Networks. In *IEEE Conference on Computer Vision and Pattern Recognition*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Hochreiter, S.; and Schmidhuber, J. 1997. Long Short-term Memory. In *Neural Computations*.

Hu, J.; Shen, L.; Albanie, S.; Sun, G.; and Wu, E. 2018. Squeeze-and-Excitation Networks. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Huang, G.; Sun, Y.; Liu, Z.; Sedra, D.; and Weinberger, K. 2016. Deep Networks with Stochastic Depth. In *European Conference on Computer Vision*.

Jackson, J.; and Schulman, J. 2019. Semi-Supervised Learning by Label Gradient Alignment. *Arxiv 1902.02336* .

Krizhevsky, A. 2009. Learning Multiple Layers of Features from Tiny Images. Technical report.

Larsson, G.; Maire, M.; and Shakhnarovich, G. 2017. FractalNet: Ultra-Deep Neural Networks without Residuals. In *International Conference on Learning Representations*.

Lee, D.-H. 2013. Pseudo-Label: The Simple and Efficient Semi-Supervised Learning Method for Deep Neural Networks. In *International Conference on Machine Learning Workshop*.

Li, B. L.; Wu, F.; Lim, S.-N.; Belongie, S.; and Weinberger, K. Q. 2020. On Feature Normalization and Data Augmentation. *Arxiv, 2002.11102* .

Li, L.; and Talwalkar, A. 2019. Random Search and Reproducibility for Neural Architecture Search. In *Conference on Uncertainty in Artificial Intelligence*.

Lim, S.; Kim, I.; Kim, T.; Kim, C.; and Kim, S. 2019. Fast AutoAugment. In *Advances in Neural Information Processing Systems*.

Liu, C.; Zoph, B.; Shlens, J.; Hua, W.; Li, L.-J.; Fei-Fei, L.; Yuille, A.; Huang, J.; and Murphy, K. 2017. Progressive Neural Architecture Search. *Arxiv, 1712.00559* .

Liu, H.; Simonyan, K.; and Yang, Y. 2019. DARTS: Differentiable Architecture Search. In *International Conference on Learning Representations*.

Marcus, M.; Kim, G.; Marcinkiewicz, M. A.; MacIntyre, R.; Bies, A.; Ferguson, M.; Katz, K.; and Schasberger, B. 1994. The Penn Treebank: Annotating Predicate Argument Structure. In *Proceedings of the Workshop on Human Language Technology*.

Merity, S.; Xiong, C.; Bradbury, J.; and Socher, R. 2017. Pointer Sentinel Mixture Models. In *International Conference on Learning Representations*.

Mirhoseini, A.; Pham, H.; Le, Q. V.; Steiner, B.; Larsen, R.; Zhou, Y.; Kumar, N.; Norouzi, M.; Bengio, S.; and Dean, J. 2017. Device Placement Optimization with Reinforcement Learning. In *International Conference on Machine Learning*.

Miyato, T.; Maeda, S.-i.; Ishii, S.; and Koyama, M. 2018. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Nakkiran, P.; Kaplun, G.; Bansal, Y.; Yang, T.; Barak, B.; and Sutskever, I. 2020. Deep Double Descent: Where Bigger Models and More Data Hurt. In *International Conference on Learning Representations*.

Park, D. S.; Chan, W.; Zhang, Y.; Chiu, C.-C.; Zoph, B.; Cubuk, E. D.; and Le, Q. V. 2019. SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition. In *Interspeech*.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*.

Pham, H.; Guan, M. Y.; Barret, Z.; Le, Q. V.; and Dean, J. 2018. Efficient Neural Architecture Search via Parameter Sharing. In *International Conference on Machine Learning*.

Real, E.; Aggarwal, A.; Huang, Y.; and Le, Q. V. 2018. Regularized Evolution for Image Classifier Architecture Search. *Arxiv, 1802.01548* .

Real, E.; Moore, S.; Selle, A.; Saxena, S.; Leon, Y. S.; Tan, J.; Le, Q.; and Kurakin, A. 2017. Large-Scale Evolution of Image Classifiers. In *International Conference on Machine Learning*.

Ridnik, T.; Lawen, H.; Noy, A.; Baruch, E. B.; Sharir, G.; and Friedman, I. 2020. TResNet: High Performance GPU-Dedicated Architecture. *Arxiv, 2003.13630* .

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; Berg, A. C.; and Fei-Fei, L. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision* .

Sennrich, R.; Haddow, B.; and Birch, A. 2016. Edinburgh neural machine translation systems for wmt 16. In *Workshop on Machine Translation*.

Sohn, K.; Berthelot, D.; Li, Chun-Liang Zhang, Z.; Carlini, N.; Cubuk, E. D.; Kurakin, A.; Zhang, H.; and Raffel, C. 2020. FixMatch: Simplifying Semi-Supervised Learning with Consistency and Confidence. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. In *JMLR*.

Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; and Wojna, Z. 2016. Rethinking the Inception Architecture for Computer Vision. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Tan, M.; and Le, Q. V. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *International Conference on Machine Learning*.

Tan, M.; Pang, R.; and Le, Q. V. 2020. EfficientDet: Scalable and Efficient Object Detection. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Tarvainen, A.; and Valpola, H. 2017. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *Advances in Neural Information Processing Systems*.

Tompson, J.; Goroshin, R.; Jain, A.; LeCun, Y.; and Bregler, C. 2020. Efficient Object Localization Using Convolutional Networks. In *International Conference on Learning Representations*.

Touvron, H.; Vedaldi, A.; Douze, M.; and Jegou, H. 2019. Fixing the train-test resolution discrepancy. In *Advances in Neural Information Processing Systems*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is All You Need. In *Advances in Neural Information Processing Systems*.

Verma, V.; Lamb, A.; Beckham, C.; Najafi, A.; Mitliagkas, I.; Courville, A.; Lopez-Paz, D.; and Bengio, Y. 2019a. Manifold Mixup: Better Representations by Interpolating Hidden States. In *International Conference on Learning Representations*.

Verma, V.; Lamb, A.; Kannala, J.; Bengio, Y.; and Lopez-Paz, D. 2019b. Interpolation Consistency Training for Semi-Supervised Learning. In *International Joint Conference on Artificial Intelligence*.

Wang, Y.; Pan, X.; Song, S.; Zhang, H.; Wu, C.; and Huang, G. 2019. Implicit Semantic Data Augmentation for Deep Networks Authors. In *Advances in Neural Information Processing Systems*.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* .

Xie, Q.; Dai, Z.; Hovy, E.; Luong, M.-T.; and Le, Q. V. 2019a. Unsupervised Data Augmentation For Consistency Training. *Arxiv, 1904.12848* .

Xie, Q.; Luong, M.-T.; Hovy, E.; and Le, Q. V. 2019b. Self-training with Noisy Student improves ImageNet classification. *Arxiv 1911.04252* .

Ying, C.; Klein, A.; Real, E.; Christiansen, E.; Murphy, K.; and Hutter, F. 2019. NAS-Bench-101: Towards Reproducible Neural Architecture Search. In *International Conference on Machine Learning*.

Yun, S.; Han, D.; Oh, S. J.; Chun, S.; Choe, J.; and Yoo, Y. 2019. CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features. In *International Conference on Computer Vision*.

Zagoruyko, S.; and Komodakis, N. 2016. Wide Residual Networks. In *British Machine Vision Conference*.

Zaremba, W.; Sutskever, I.; and Vinyals, O. 2014. Recurrent Neural Network Regularization. *Arxiv, 1409.2329* .

Zhang, H.; Cisse, M.; Dauphin, Y. N.; and Lopez-Paz, D. 2018. mixup: Beyond Empirical Risk Minimization. In *International Conference on Learning Representations*.

Zhuang, C.; Ding, X.; Murli, D.; and Yamins, D. 2020. Local Label Propagation for Large-Scale Semi-Supervised Learning. *Arxiv, 1905.11581* .

Zoph, B.; Cubuk, Ekin D.and Ghiasi, G.; Lin, T.-Y.; Shlens, J.; and Le, Q. V. 2019. Learning Data Augmentation Strategies for Object Detection. *Arxiv 1906.11172* .

Zoph, B.; and Le, Q. V. 2017. Neural Architecture Search with Reinforcement Learning. In *International Conference on Learning Representations*.

Zoph, B.; Vasudevan, V.; Shlens, J.; and Le, Q. V. 2018. Learning Transferable Architectures for Scalable Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*.