Appendix

A. Architecture Search Settings

In this section, the implementation details for the search phase are provided.

A.1. Search Space

We use the following 7 operations in our search on CIFAR-10 and CIFAR-100:

1. `skip_connect`: identity connection
2. `sep_conv_3x3`: depthwise-separable 3x3 convolution
3. `max_pool_3x3`: max pooling with 3x3 kernel
4. `dil_conv_3x3`: dilated depthwise-separable 3x3 convolution
5. `sep_conv_5x5`: depthwise-separable 5x5 convolution
6. `avg_pool_3x3`: average pooling with 3x3 kernel
7. `sep_conv_7x7`: depthwise-separable 7x7 convolution

In the case of ImageNet, in order to make the search tractable, we only use the first five operations. All operations use a stride of 1 when part of the Normal Cell, and a stride of 2 when part of the Reduce Cell. Appropriate padding is added to the input features to preserve the spatial dimensions. Each convolution consists of a (`ReLU-Conv-BN`) block, and the depthwise separable convolutions are always applied twice, consistent with prior work [18, 26, 39, 43].

A.2. CIFAR-10 and CIFAR-100

The CIFAR-10 and CIFAR-100 datasets consist of 50,000 training images and 10,000 test images. During search, we use 45,000 images from the original training set as our training set and the remaining as the validation set. The final evaluation phase uses the original split. During architecture search, a network is constructed by stacking 8 cells with 4 hidden nodes. Similar to DARTS [18], the cells are stacked in the blocks of 2-2-2 Normal cells with Reduction cells in between. The networks are trained using 4 Tesla V100 GPUs with a batch size of 124, for 100 epochs. For the first 15 epochs, only the network parameters ($\boldsymbol{w}$) are trained, while the architecture parameters ($\boldsymbol{\phi}$) are frozen. This pretraining phase prevents the search from ignoring the operations that are typically slower to train. The architecture parameters are trained using the Adam optimizer with cosine learning rate schedule starting from 2×10^{-3} annealed down to 3×10^{-4}. The network parameters are also trained using Adam with cosine learning rate schedule starting from 6×10^{-4} annealed down to 1×10^{-4}. We use $\lambda = 0.5$, and a Gumbel-Softmax temperature of 0.4.

One issue with the factorized structure is that the architecture search may choose the same input and operation pair for both incoming edges of a node due to the symmetric expression in $\boldsymbol{i}_n \otimes \boldsymbol{o}_n + \boldsymbol{i}'_n \otimes \boldsymbol{o}'_n$. To prevent this, we add an architecture penalty term to our objective function using $\mathcal{L}_{arch}(\boldsymbol{z}) = \mathbb{E}\left[\lambda_{arch} \sum_{n=1}^{N} \text{tr}([\boldsymbol{i}_n \otimes \boldsymbol{o}_n][\boldsymbol{i}'_n \otimes \boldsymbol{o}'_n]^T)\right]$ where λ_{arch} is a trade-off parameter ($\lambda_{arch} = 0.2$). The term inside the summation is one if the same input/op pairs are selected by $(\boldsymbol{i}_n, \boldsymbol{o}_n)$ and $(\boldsymbol{i}'_n, \boldsymbol{o}'_n)$.

A.3. ImageNet

We search using a 14-layer network with 16 initial channels, over 8 V100 GPUs, needing around 2 days. We use a learning rate of 3×10^{-4} with Adam to learn the network parameters of the mixed-op network. We train architecture parameters with a learning rate of 1×10^{-3} using Adam. We parallelize training over 8 GPUs without scaling the learning rate. For the first 5 epochs, we only train the network parameters ($\boldsymbol{w}$), and in the remaining 15 epochs, we update both $\boldsymbol{w}$ and $\boldsymbol{\phi}$. We use $\lambda = 0.5$ and $\lambda_{arch} = 0.2$, the same as CIFAR-10, and a Gumbel-Softmax temperature of 0.4. We use a weight decay of 3×10^{-4} on the weight parameters, and 1×10^{-6} on the architecture parameters. 90% of the ImageNet train set is used to train the weight parameters, while the rest is used as the validation set for training the architecture parameters.

B. Architecture Evaluation Settings

In this section, the implementation details for the evaluation phase are provided.

B.1. CIFAR-10 and CIFAR-100

The final network is constructed by stacking a total of 20 cells. The networks are trained on a V100 GPU with a batch size of 128 for 600 epochs. SGD with momentum 0.9 is used. The cosine learning rate schedule is used starting from 5×10^{-2} annealed down to zero. Similar to DARTS, the path dropout of the probability 0.2 on CIFAR-10 and 0.3 on CIFAR-100, and cutout of 16 pixels are used.

B.2. ImageNet

For data augmentation, we use the same settings as DARTS [18]. We randomly crop training images to a size of 224×224 px along with a random horizontal flip, and jitter the color. During evaluation, we use a single center crop of size 224×224 px after resizing the image to 256×256 px.

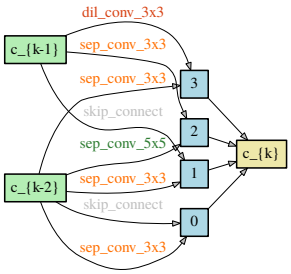
For the final evaluation, we train a 14 layer network for 250 epochs with an initial channel count such that the multiply-adds of the network is <600M, as per the mobile setting proposed by [12]. We train our networks using SGD with momentum of 0.9, base learning rate of 0.1, weight decay of 3×10^{-5}, with a batch size of 128 per GPU. We train our model for 250 epochs in line with prior work [18, 38, 39], annealing the learning rate to 0 throughout the training using a cosine learning rate decay. We scale training to 8 V100

GPUs using the linear scaling rule proposed in [7], with a learning rate warmup for the first 5 epochs.

C. Best Cell Structures

Figure 7: The best performing cell discovered on ImageNet.

(a) Normal Cell



(b) Reduce Cell

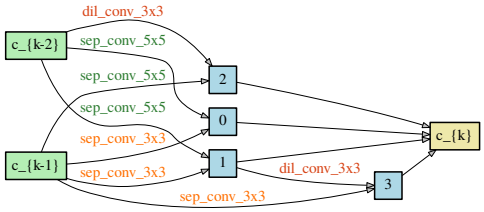
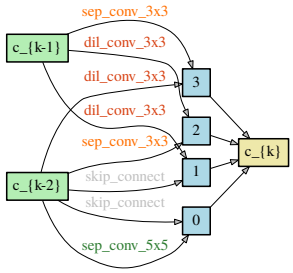


Figure 8: The best performing cell discovered on CIFAR-10.

(a) Normal Cell



(b) Reduce Cell

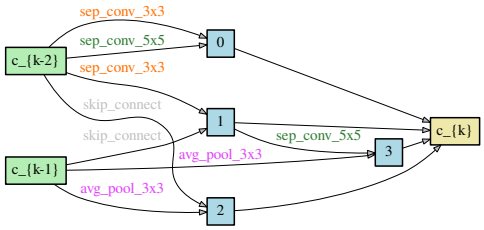
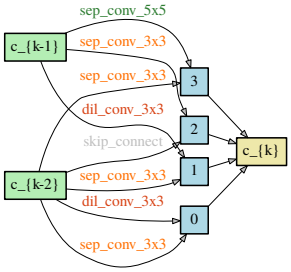


Figure 9: The best performing cell found on CIFAR-100.

(a) Normal Cell



(b) Reduce Cell

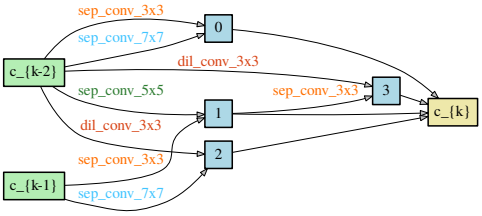
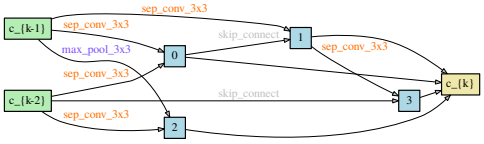
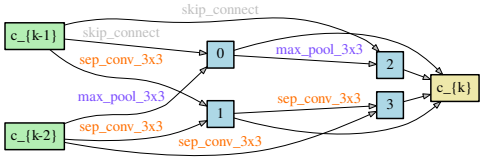


Figure 10: The best performing randomly proposed cell on ImageNet.

(a) Normal Cell



(b) Reduce Cell



D. Comparison with the Previous Work in DARTS Space

In this section, we compare the best cells discovered by UNAS against previously published results on CIFAR-10, CIFAR-100 and ImageNet.

CIFAR-10: In Table 4, the best cell discovered by UNAS is compared against the previous work that uses similar search space. For DARTS and P-DARTS, we list the original results reported by the authors, as well as, the best cell we discovered by running the original implementation four times. The best cell discovered by UNAS outperforms DARTS and SANS. In comparison to P-DARTS, UNAS obtains better than the best cell that we discovered by running the original P-DARTS code four times with different seeds. However, UNAS achieves a comparable result to P-DARTS' originally reported result on CIFAR-10. Nevertheless, as we show in Table 2, UNAS outperforms DARTS, P-DARTS, and SNAS in terms of the average performance. As discussed by Li and Talwalkar [15], the average performance is a better representative metric to evaluate the performance of NAS methods, as it is more robust against rare architecture instances that perform well, but, are less likely to be discovered by the method. Such architectures require many search/evaluation runs, making NAS models expensive for practical applications, and more challenging for reproducing the results.

When we ran the original P-DARTS source code with four different initialization seeds[7], we could not find an architecture with accuracy similar to the reported number. We believe this is because i) P-DARTS reports the lowest error observed during the evaluation phase while we report the error at the end of evaluation following DARTS. Taking the minimum of test error values, across small fluctuations towards the end of training, can reduce the error rate by 0.1%, ii) P-DARTS does not report the number of searches performed to obtain the best result. We hypothesize that the reported result is the best architecture obtained from many searches. However, we do not intend to discount the contributions made by P-DARTS. When we evaluate the original discovered cell by P-DARTS on CIFAR-10, we can reproduce the same results in the evaluation phase. Nevertheless, the contributions of UNAS are orthogonal to P-DARTS thesis as discussed in Sec. 1.1. UNAS proposes new gradient estimators that work with differentiable and non-differentiable objective functions and it also introduces a new objective function based on the generalization gap.

CIFAR-100: In Table 5, our best cell discovered using UNAS is compared against previous work. We can see that UNAS outperforms DARTS, SANS, and P-DARTS on this dataset. Similar to CIFAR-10, when we ran P-DARTS code four times, we could not discover a cell as performant as the cell discovered originally on CIFAR-100.

[7]We exactly followed the hyperparameters and commands using the search/eval code provided by the authors. We only set the initialization seed to a number in $\{0, 1, 2, 3\}$.

Table 4: Results on CIFAR-10.

Architecture	Test Error (%)	Params (M)	Search Cost (GPU Days)	Search Method
NASNet-A [43]	2.65	3.3	2000	RL
BlockQNN [41]	3.54	39.8	96	RL
AmoebaNet-A [26]	3.12	3.1	3150	evolution
AmoebaNet-B [26]	2.55	2.8	3150	evolution
H. Evolution [17]	3.75	15.7	300	evolution
PNAS [16]	3.41	3.2	225	SMBO
ENAS [24]	2.89	4.6	0.45	RL
Random [18]	3.29	3.2	4	random
DARTS-1^{st} [18]	3.00	3.3	1.5	grad-based
DARTS-2^{nd} [18]	2.76	3.3	4	grad-based
SNAS [39]	2.85	2.8	1.5	grad-based
P-DARTS [39]	2.50	3.4	0.3	grad-based
Best cell discovered after running the original code 4 times				
DARTS-2^{nd} [18]	2.80	3.6	4	grad-based
P-DARTS [39]	2.75	3.5	0.3	grad-based
UNAS	2.53	3.3	4.3	grad RL

Table 5: Results on CIFAR-100.

Architecture	Test Error (%)	Params (M)	Search Cost (GPU Days)	Search Method
BlockQNN [41]	18.06	39.8	96	RL
P-DARTS [39]	15.92	3.6	0.3	grad-based
Best cell discovered after running the original code 4 times				
DARTS-2^{nd} [18]	20.49	1.8	4	grad-based
P-DARTS [39]	17.36	3.7	0.3	grad-based
UNAS	15.79	4.1	4.0	grad RL

ImageNet: Here, we compare UNAS on the ImageNet dataset against previous works. We also provide a surprisingly strong baseline using randomly generated architectures. Table 6 summarizes the results.

Random Baseline: We provide a strong random baseline, indicated by "Random Cell" in Table 6, that outperforms most prior NAS methods. Random cells are generated by drawing uniform random samples from factorized cell structure. We train a total of 10 networks constructed by randomly generated Normal and Reduce cells. The best network yields top-1 and top-5 errors of 25.55% and 8.06% respectively (see Fig 10 for the cell structure). To the best of our knowledge, we are the first to report performance of a randomly discovered cell on ImageNet that outperforms most previous NAS methods, although not UNAS and P-DARTS.

Direct Search on ImageNet: Searching on ImageNet gives us the cell in Fig. 7. Our cell searched on Ima-

Table 6: Best results on ImageNet in the mobile setting (#Multi.-Adds<600M) [12].

Architecture	Val Error (%)		Params	×+	Search Cost	Search Method
	top-1	top-5	(M)	(M)	(GPU Days)	
MobileNetV2 [28]	25.3	–	6.9	585	–	manual
ShuffleNetV2 2× [19]	25.1	7.8	7.4	591	–	manual
NASNet-A [43]	26.0	8.4	5.3	564	2000	RL
AmoebaNet-B [26]	26.0	8.5	5.3	555	3150	evolution
AmoebaNet-C [26]	24.3	7.6	6.4	570	3150	evolution
PNAS [16]	25.8	8.1	5.1	588	∼255	SMBO
DARTS [18]	26.7	8.7	4.7	574	4	grad-based
SNAS [39]	27.3	9.2	4.3	522	1.5	grad-based
P-DARTS [5]	24.4	7.4	4.9	557	0.3	grad-based
Best cell discovered after running the original code 4 times						
DARTS [18]	25.2	7.7	5.12	595	4	grad-based
P-DARTS [5]	24.5	7.3	5.2	599	0.3	grad-based
Random Cell	25.55	8.06	5.37	598	∼250	random
UNAS	24.46	7.44	5.07	563	16	grad-based RL

geNet obtains a performance, comparable to P-DARTS and AmoebaNet-C [26], giving a top-1 and top-5 error of 24.46% and 7.44% resp. at a fraction of the cost (0.5%) required by the best AmoebaNet-C [26].

E. UNAS with ProxylessNAS Search Space

In this section, we list the implementation details used for the latency based experiments presented in Sec. 5.

E.1. Search Space

We follow ProxylessNAS [4] to construct the search space which is based on MobileNetV2 [28]. During search we seek operations assigned to each layer of a 21-layer network. The operations in each layer are constructed using mobile inverted residual blocks [28] by varying the kernel size in $\{3, 5, 7\}$ and the expansion ratio in $\{3, 6\}$ yielding 6 choices with the addition of a skip connection (i.e., an identity operation) which enables removing layers. For the channel sizes, we followed the ProxylessNAS-GPU architecture. For the first 20 epochs, only the network parameters ($\boldsymbol{w}$) are trained, while the architecture parameters ($\boldsymbol{\phi}$) are frozen. The architecture parameters are trained in 15 epochs using the Adam optimizer with cosine learning rate schedule starting from $1{\times}10^{-3}$ annealed down to $3{\times}10^{-4}$. The network parameters are also trained using Adam with cosine learning rate schedule starting from $3{\times}10^{-4}$ annealed down to $1{\times}10^{-4}$. Batch of 192 images on 8 V-100 GPUs are used for training. For the latency-based search, we use the following objective function:

$$\mathbb{E}_{p_{\boldsymbol{\phi}}(\boldsymbol{z})}[\mathcal{L}_{\text{gen}}(\boldsymbol{z}, \boldsymbol{w})] + \lambda_{lat}\mathbb{E}_{p_{\boldsymbol{\phi}}(\boldsymbol{z})}[f(\mathcal{L}_{lat}(\boldsymbol{z}) - t_{\text{target}})]$$

where t_{target} represents the target latency, $f(u) = max(0, u)$ penalizes the architectures that has latency higher than the target latency.

We linearly anneal λ_{lat} from zero to 0.1 to focus the architecture search on the classification loss initially. However, we empirically observed that the latency loss has a low gradient variance that provides a very strong training signal for selecting low-latency operations such as skip connection. To avoid this, Inspired by P-DARTS [5], we apply dropout to the skip connection during search. We observe that a small amount of dropout with probability 0.1 prevents the search from over-selecting the skip operation.

E.2. Evaluation

After search, the operations in each layer with the highest probability values are chosen for the final network. The training in the evaluation phase is based on the ProxylessNAS evaluation. Batches of 512 images on 8 V-100 GPUs are used for training in 300 epochs. We train our networks using SGD with momentum of 0.9, base learning rate of 0.2, linear learning-rate warmup in 5 epoch, and weight decay of $5{\times}10^{-5}$. The learning rate is annealed to 0 throughout the training using a cosine learning rate decay.