Supplementary Materials for "Pareto-Frontier-aware Neural Architecture Search"

In the supplementary, we provide more discussions, more implementation details, and more experimental results of the proposed PFNAS. We organize the supplementary material as follows.

- In Section A, we provide the derivations of the objective function of the controller model.
- In Section B, we give more discussions on the pairwise ranking loss.
- In Section C, we conduct analyses of the size of the considered search space.
- In Section D, we provide the detailed model design of our PFNAS.
- In Section E, we give more details on the constraint representation method of PFNAS.
- In Section F, we provide more training and inference details of PFNAS.
- In Section G, we investigate the effect of K on the search performance of PFNAS.
- In Section H, we show the training curves of the architecture evaluator and the controller.
- In Section I, we show the latency histograms of the architectures searched by PFNAS under different budgets.
- In Section J, we provide more results on CPU devices.
- In Section K, we provide more results on GPU devices.
- In Section L, we visualize all the searched architectures on three hardware platforms.

A Derivations of the objective function

The objective function of the controller in PFNAS can be formulated as

$$\begin{aligned} J(\theta) &= \mathbb{E}_{T\sim\mathcal{T}}\left[\mathbb{E}_{\alpha_T\sim\pi(\cdot|T;\theta)}\left[R\left(\alpha_T|T;w\right)\right] + \lambda H\big(\pi(\cdot|T;\theta)\big)\right] \\ &= \sum_T p(T)\left[\sum_{\alpha_T}\pi(\alpha_T|T;\theta)R\left(\alpha_T|T;w\right) + \lambda H\big(\pi(\cdot|T;\theta)\big)\right]. \end{aligned} \tag{1}$$

Let $p(T)$ be the probability to sample a specific latency T from the distribution $\mathcal{T}$. The gradient of the objective function w.r.t. θ can be computed by

$$\begin{aligned} \nabla_\theta J(\theta) &= \sum_T p(T)\left[\sum_{\alpha_T}\nabla_\theta\pi(\alpha_T|T;\theta)R\left(\alpha_T|T;w\right) + \lambda\nabla_\theta H\big(\pi(\cdot|T;\theta)\big)\right] \\ &= \sum_T p(T)\left[\sum_{\alpha_T}\pi(\alpha_T|T;\theta)\nabla_\theta\log\pi(\alpha_T|T;\theta)R\left(\alpha_T|T;w\right) + \lambda\nabla_\theta H\big(\pi(\cdot|T;\theta)\big)\right] \\ &= \mathbb{E}_{\alpha_T\sim\pi(\cdot|T;\theta),T\sim\mathcal{T}}\left[\nabla_\theta\log\pi(\alpha_T|T;\theta)R\left(\alpha_T|T;w\right) + \lambda\nabla_\theta H\big(\pi(\cdot|T;\theta)\big)\right] \\ &\approx \frac{1}{KN}\sum_{k=1}^{K}\sum_{i=1}^{N}\left[\nabla_\theta\log\pi(\alpha_{T_k}^{(i)}|T_k;\theta)R(\alpha_{T_k}^{(i)}|T_k;w) + \lambda\nabla_\theta H(\pi(\cdot|T_k;\theta))\right]. \end{aligned} \tag{2}$$

B More discussions on the pairwise ranking loss

In this section, we provide more discussions on the pairwise ranking loss (Freund et al., 2003; Burges et al., 2005; Chen et al., 2009). To provide a reward for PFNAS, we propose a Pareto dominance rule and learn an architecture evaluator to match it. Based on M architectures that are uniformly sampled from the search space, we construct $M(M-1)$ architecture pairs after omitting the pairs with the

same architectures. Given K different budgets, we train the architecture evaluator by minimizing the following loss function

$$L(w) = \frac{1}{KM(M-1)} \sum_{k=1}^{K} \sum_{i=1}^{M} \sum_{j=1, j\neq i}^{M} \phi\big((R(\beta_i|T_k; w) - R(\beta_j|T_k; w)) \cdot d(\beta_i, \beta_j, T_k)\big), \quad (3)$$

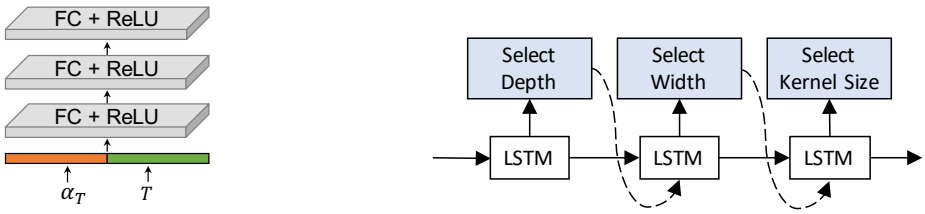
where $\phi(z) = \max(0, 1-z)$ is the hinge loss function. The goal of minimizing Eqn. (3) is to make the architecture evaluator $R(\beta_i|T_k; w)$ rank different architectures under the budgets w.r.t. T_k. To this end, given any two architectures β_i and β_j, we use the hinge loss $\phi(\cdot)$ to force the predicted ranking result $R(\beta_i|T_k; w) - R(\beta_j|T_k; w)$ to be consistent with the ranking result $d(\beta_i, \beta_j, T_k)$ obtained by the comparison rules. Based on the pretrained evaluator, given an arbitrary architecture and a target latency, we are able to obtain a score/reward that indicates whether the architecture is good under the budget w.r.t. the considered latency.

C SEARCH SPACE SIZE ANALYSIS

In this section, we analyze the size of the considered search space. We use MobileNetV3 (Howard et al., 2019) as the backbone to build the search space (Cai et al., 2020; Huang & Chu, 2020). Specifically, we divide a network into several units, each of which contains a sequence of layers where only the first layer has a stride of 2. To find promising architectures, we allow each unit to have 1) any numbers of layers (*i.e.*, depth) that can be chosen from $\{2, 3, 4\}$, 2) any width expansion ratios in each layer (*i.e.*, width) that can be chosen from $\{3, 4, 6\}$, and 3) any kernel sizes that can be chosen from $\{3, 5, 7\}$. Following (Cai et al., 2020), we build the model with 5 units Thus, there are 3×3 combinations of widths and kernel sizes for each layer. Given 3 different depths and 5 units, there are roughly $\big((3\times3)^2+(3\times3)^3+(3\times3)^4\big)^5 \approx 2\times10^{19}$ different architectures in the search space.

D MODEL DESIGN OF ARCHITECTURE EVALUATOR AND CONTROLLER

In this section, we provide the detailed model design of the architecture evaluator and the controller model. We show the architecture in Figure A. We build the architecture evaluator with a three-layer fully connected network and each of them is followed by a ReLU (Nair & Hinton, 2010) activation layer. We set the number of intermediate neurons to 512 (See Figure A(a)). As for the controller, we use an LSTM to build the model(See Figure A(b)). Since the architecture can be represented by a sequence of tokens (Zoph & Le, 2017; Pham et al., 2018), the LSTM based controller is able to produce architectures by sequentially predicting the token sequences, including depth, width, and kernel size.



(a) Architecture evaluator. (b) Pareto-Frontier-aware controller.

Figure A: Model design of the architecture evaluator and the Pareto-Frontier-aware controller.

E MORE DETAILS ON CONSTRAINT REPRESENTATION METHOD

In the experiments, we select K discrete latency budget constraints by evenly dividing the range (*e.g.*, $\{80, 110, 140, 170, 200\}$ on mobile devices with $K = 5$). Following (Pham et al., 2018), we randomly sample a budget from the K discrete latency budgets and use its embedding to train the PFNAS model. To represent any latency budget T during inference, we perform a linear interpolation between the embedding of two adjacent discrete latencies. Specifically, given two adjacent discrete

latency T_1 and T_2 as well as a desired latency $T_1 \leq \hat{T} \leq T_2$, we can calculate $\hat{T}$ by a linear interpolation as

$$\hat{T} = \lambda T_1 + (1 - \lambda) T_2, \tag{4}$$

where $\lambda \in [0, 1]$ denotes the weight of the linear interpolation. Thus, we have $\lambda = (T_2 - \hat{T})/(T_2 - T_1)$. Then, we construct the embedding $\hat{\boldsymbol{z}}$ *w.r.t.* latency $\hat{T}$ by

$$\hat{\boldsymbol{z}} = \lambda \boldsymbol{z}_1 + (1 - \lambda) \boldsymbol{z}_2, \tag{5}$$

where $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ are the embedding of T_1 and T_2, respectively. With the representation of any latency T, PFNAS generates the Pareto frontier over a range of diverse latency budgets.

F More training and evaluation details

In this section, we give more training and evaluation details of the proposed PFNAS. More details can be founded in our submitted code.

Training details. We first train the architecture evaluator and then train the controller model. During training, we train the architecture evaluator for 250 epochs. We apply the SGD optimizer with the weight decay of $3{\times}10^{-5}$ and the momentum of 0.9. The learning rate is initialized to 0.1 and decreased to $1{\times}10^{-3}$ with a cosine annealing. The minibatch size is set to 32. As for the controller, we use Adam optimizer with a learning rate of 3×10^{-4} and train the model for $6,000$ epochs. Following ENAS (Pham et al., 2018), we sample $N{=}1$ architecture at each iteration and find that it works fine. We first collect $M{=}16,000$ architectures by uniformly sampling architectures from the search space Ω and obtain the latency ranges on three hardware platforms. Then, we select $K{=}5$ latency budgets by evenly dividing the range (*e.g.*, $\{80, 110, 140, 170, 200\}$ on Google Pixel1 phone). The hyper-parameter λ is set to $1{\times}10^{-3}$ for balancing the entropy regularization term.

Evaluation details. Based on the learned policy $\pi(\cdot|T;\theta)$, we first sample several candidate architectures from $\pi(\cdot|T;\theta)$ and then select the architecture with the highest validation accuracy from the architectures with $c(\alpha_T) \leq T$. To accelerate the inference, following (Cai et al., 2020), we train two predictors to predict the latency and validation accuracy, respectively. After that, we deploy the resultant architecture to hardware platforms and measure the running latency and test accuracy. To measure the latency on hardware platforms, we deploy the resultant architectures to different devices and measure the latency over a batch of images. Specifically, we measure the latency on mobile and CPU devices with a batch size of 1. Since the inference on GPU is too fast to obtain the accurate latency, we measure the latency with the batch size of 64 on NVIDIA TITAN X. Following OFA (Cai et al., 2020), we directly apply the weights from the learned super network to the searched architectures and evaluate different architectures on ImageNet. For a fair comparison, on three hardware platforms, we do not finetune the resultant models obtained by all the considered methods, including OFA (Cai et al., 2020), OFA-S, OFA-MO, and our PFNAS.

G Effect of K on the search performance of PFNAS

In this part, we investigate the effect of K on the search performance of PFNAS. Note that we evenly select K budgets from the range of latency (*e.g.*, given $K{=}5$, the selected latencies are $\{80, 110, 140, 170, 200\}$) on Google Pixel1 phone. To investigate the effect of K, we consider several candidate values of $K \in \{2, 5, 10, 30, 75\}$. We use $\overline{P_s}$ to denote the average over K proportions of the searched architectures satisfying the corresponding budget constraints. From Table A, since a larger K corresponds to a more difficult optimization problem, $\overline{P_s}$ would decrease when we gradually increase K from 2 to 75. It is worth noting that, even with a very large $K{=}75$, our method still achieves promising performance. These results demonstrate the effectiveness of the proposed PFNAS method under diverse budgets.

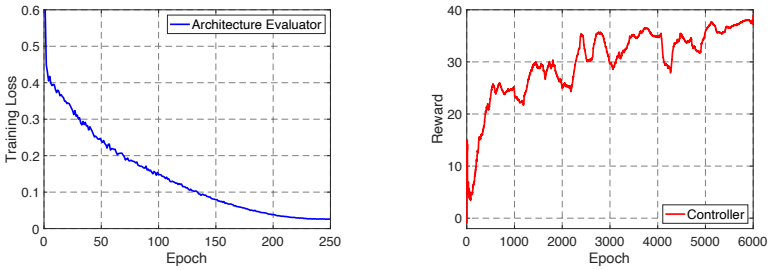
H Training curves of PFNAS

In this section, we show the training curves of the architecture evaluator and the controller model in Figure B. As shown in Figure B(a), by minimizing the pairwise ranking loss, the architecture

Table A: Effect of K on the search performance of PFNAS. $\overline{P_s}$ denotes the average over K proportions of the searched architectures that satisfy the corresponding budgets.

K	2	5	10	30	75
$\overline{P_s}$ (%)	89.1	75.4	73.4	70.7	66.8

evaluator is able to converge very fast. Based on the pretrained architecture evaluator, the controller can be effectively trained to obtain higher rewards gradually (See Figure B(b)). Since we train the controller over a set of diverse budgets, our PFNAS is able to find the promising architectures under diverse budgets.



(a) Training loss evolution of the architecture evaluator.

(b) Reward evolution of PFNAS.

Figure B: Training loss and reward evolution of the architecture evaluator and the controller.

I LATENCY HISTOGRAMS OF SEARCHED ARCHITECTURES

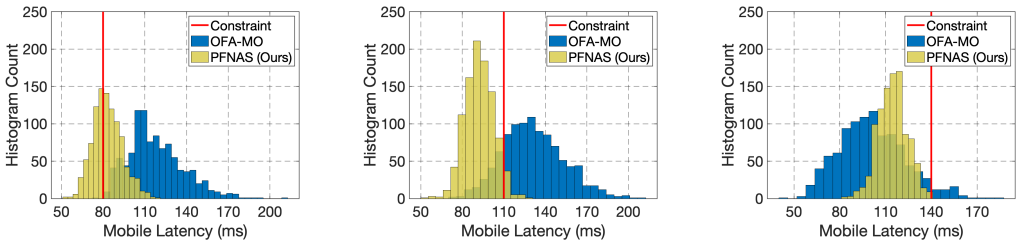
In this section, we show the latency histograms of the searched architectures by OFA-MO and our PFNAS under 5 latency budgets. We show the results in Figure C. From these figures, our PFNAS is able to find the architectures satisfy the corresponding budget with higher probability than OFA-MO under different budgets. As mentioned in Section 4.3 in the paper, even if there exist only a few architectures whose latency is lower than 80ms, we still produce a lot of architectures satisfying the budget constraint. These results demonstrate the effectiveness of the proposed method.

J MORE RESULTS ON CPU DEVICES

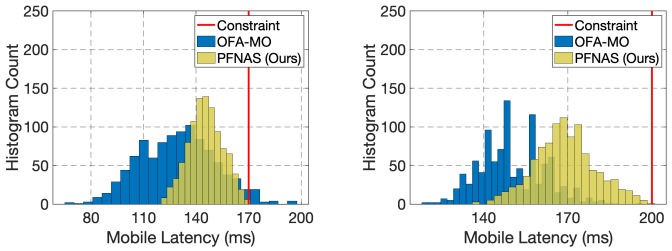
In this section, we compare PFNAS with state-of-the-art methods on Intel Core i5-7400 CPU under the latency budgets of 30ms, 35ms, 40ms, 45ms, and 50ms, respectively. We show the results in Table B. From Table B, PFNAS outperforms state-of-the-art architectures under different latency budgets. Specifically, given any latency budget, the architectures searched by PFNAS consistently yield better performance than the considered baseline methods, including OFA-S and OFA-MO. More critically, our PFNAS only need to search once to produce promising architectures that satisfy different latency budgets accordingly, while previous methods need to repeat the search process according to different budgets. These results show the convenience and effectiveness of our PFNAS on CPU devices.

K MORE RESULTS ON GPU DEVICES

In this section, we compare PFNAS with the state-of-the-art methods on NVIDIA TITAN X GPU. Given a set of latency budgets $\{90, 115, 140, 165, 190\}$, PFNAS only need to search once to produce different architectures. From Table C, PFNAS yields higher accuracy than other methods with similar latency budgets. Specifically, PFNAS performs better than OFA-S and OFA-MO under different latency budgets. Besides, the performance of PFNAS is better than those of hand-crafted methods



(a) Search results with T=80ms. (b) Search results with T=110ms. (c) Search results with T=140ms.



(d) Search results with T=170ms. (e) Search results with T=200ms.

Figure C: Comparisons of the search results given different resource budgets on Google Pixel1.

(*e.g.*, ResNet (He et al., 2016) and MobileNetV2 (Sandler et al., 2018)) and those of NAS methods that perform the search process without computational cost constraints (*e.g.*, DARTS (Liu et al., 2019) and ENAS (Pham et al., 2018)).

L VISUALIZATION OF THE SEARCHED ARCHITECTURES

In this section, we visualize the architectures searched by PFNAS under different budgets. We show the searched architectures on mobile phone, CPU, and GPU in Figures D, E, and F, respectively. For convenience, we use "Architecture-T-Hardware" to represent the searched architecture under the budge w.r.t. T on a specific hardware platform, *e.g.*, PFNAS-80-Mobile. From these figures, our PFNAS tends to find the architectures with larger depth, width, and kernel size under a larger budget constraint. More critically, from Tables 1, B, C, the resultant architectures often have the latency very close to the considered budget. These results show that our method is able to sufficiently exploit the given resource budget to find promising architectures.

Table B: Comparisons with state-of-the-art architectures on Intel Core i5-7400 CPU. * denotes the best architecture reported in the original paper. All the models are evaluated on 224×224 images of ImageNet. "-" denotes the results that are not reported. We obtain the accuracy without finetuning.

Architecture	Latency (ms)	Top-1 Acc. (%)	Top-5 Acc. (%)	#Params. (M)	#MAdds (M)
ENAS (Pham et al., 2018)	66.6	73.8	91.7	5.6	607
DARTS (Liu et al., 2019)	83.9	73.3	91.3	4.7	574
NASNet-A (Zoph et al., 2018)	93.7	74.0	91.6	5.3	564
P-DARTS (Chen et al., 2019)	134.0	75.6	92.6	4.9	577
PC-DARTS (Xu et al., 2020)	99.4	75.8	92.7	5.3	597
MobileNetV2 (1.0×) (Sandler et al., 2018)	28.6	72.0	-	3.4	300
MobileNetV3-Large (1.0×) (Howard et al., 2019)	22.6	75.2	-	5.4	219
MnasNet-A1 (1.0×) (Tan et al., 2019)	26.4	75.2	92.5	3.4	300
FBNet-C (Wu et al., 2019)	25.7	74.9	-	5.5	375
OFA-S-30	29.8	76.9	93.0	5.7	343
OFA-MO-30	29.7	77.5	93.7	6.6	353
PFNAS-30 (Ours)	29.7	**77.7**	**93.7**	7.6	335
ProxylessNAS-CPU (Cai et al., 2019)	34.6	75.3	-	4.4	438
MnasNet-A1 (1.4×) (Tan et al., 2019)	34.6	77.2	93.5	6.1	592
OFA (Cai et al., 2020)	34.5	78.1	94.0	8.2	354
OFA-S-35	34.9	77.8	93.8	7.6	406
OFA-MO-35	34.7	78.3	94.0	7.9	478
PFNAS-35 (Ours)	34.5	**78.4**	**94.1**	8.4	431
ResNet-18 (He et al., 2016)	38.6	69.8	90.1	11.7	1814
EfficientNet B0 (Tan & Le, 2019)	39.1	77.3	93.5	5.3	390
OFA (Cai et al., 2020)	36.3	78.4	94.1	8.4	388
OFA-S-40	39.5	78.2	94.1	8.2	495
OFA-MO-40	39.3	78.6	94.3	8.3	491
PFNAS-40 (Ours)	39.6	**78.6**	**94.4**	9.4	502
MobileNetV2 (1.4×) (Sandler et al., 2018)	42.6	74.7	-	6.9	585
OFA (Cai et al., 2020)	43.2	78.8	94.4	9.1	481
OFA-S-45	45.0	78.3	94.2	8.2	548
OFA-MO-45	43.7	78.8	94.4	9.3	626
PFNAS-45 (Ours)	44.7	**79.0**	**94.5**	10.4	620
PONAS-C (Huang & Chu, 2020)	52.2	75.2	-	5.6	376
OFA* (Cai et al., 2020)	47.4	78.9	94.5	9.1	511
OFA-S-50	49.8	78.5	94.2	7.5	600
OFA-MO-50	46.7	78.9	94.4	9.1	632
PFNAS-50 (Ours)	48.9	**79.1**	**94.6**	10.5	682

Table C: Comparisons with state-of-the-art architectures on NVIDIA TITAN X GPU. * denotes the best architecture reported in the original paper. All the models are evaluated on 224×224 images of ImageNet. "-" denotes the results that are not reported. We obtain the accuracy without finetuning.

Architecture	Latency (ms)	Top-1 Acc. (%)	Top-5 Acc. (%)	#Params. (M)	#MAdds (M)
DARTS (Liu et al., 2019)	82.5	73.3	91.3	4.7	574
P-DARTS (Chen et al., 2019)	86.2	75.6	92.6	4.9	577
PC-DARTS (Xu et al., 2020)	85.3	75.8	92.7	5.3	597
FBNet-A (Wu et al., 2019)	53.3	73.0	-	4.3	249
FBNet-B (Wu et al., 2019)	79.4	74.1	-	4.5	295
FBNet-C (Wu et al., 2019)	89.5	74.9	-	5.5	375
ProxylessNAS-GPU (Cai et al., 2019)	84.7	75.1	-	7.1	463
MobileNetV2 (1.0×) (Sandler et al., 2018)	71.6	72.0	-	3.4	300
OFA-S-90	88.8	76.2	92.9	5.8	320
OFA-MO	89.8	75.4	92.4	4.9	266
PFNAS-90 (Ours)	86.9	**77.4**	**93.6**	7.7	310
MobileNetV2 (1.4×) (Sandler et al., 2018)	107.1	74.7	-	6.9	585
ProxylessNAS-CPU (Cai et al., 2019)	102.1	75.3	-	4.4	438
MnasNet-A1 (1.4×) (Tan et al., 2019)	112.9	77.2	93.5	6.1	592
EfficientNet B0 (Tan & Le, 2019)	115.5	77.3	93.5	5.3	390
ENAS (Pham et al., 2018)	110.8	73.8	91.7	5.6	607
OFA (Cai et al., 2020)	105.4	78.4	94.1	8.4	388
OFA-S-115	114.9	76.5	93.0	6.1	362
OFA-MO	111.2	78.1	94.0	8.8	431
PFNAS-115 (Ours)	111.2	**78.3**	**94.1**	8.9	411
OFA (Cai et al., 2020)	135.7	78.9	94.4	9.1	481
OFA-S-140	139.7	77.8	93.7	7.1	415
OFA-MO	137.2	78.4	94.1	8.8	470
PFNAS-140 (Ours)	138.9	**78.6**	**94.4**	9.7	510
ResNet-50 (He et al., 2016)	159.8	76.2	92.9	25.6	4087
OFA* (Cai et al., 2020)	145.7	78.9	94.5	9.1	511
OFA-S-165	164.8	77.8	93.8	7.0	479
OFA-MO	162.6	78.8	94.4	10.5	583
PFNAS-165 (Ours)	162.7	**78.9**	**94.4**	10.5	582
NASNet-A (Zoph et al., 2018)	162.3	74.0	91.6	5.3	564
DenseNet-121 (Huang et al., 2017)	172.8	75.0	92.3	8.0	2833
PONAS (Huang & Chu, 2020)	182.4	75.2	-	5.6	376
EfficientNet B1 (Tan & Le, 2019)	192.7	79.2	94.5	7.8	700
OFA-S-190	190.0	78.4	94.2	7.5	533
OFA-MO	183.2	78.8	94.5	10.7	652
PFNAS-190 (Ours)	185.5	**79.2**	**94.6**	10.4	640

(a) PFNAS-80-Mobile.

(b) PFNAS-110-Mobile.

(c) PFNAS-140-Mobile.

(d) PFNAS-170-Mobile.

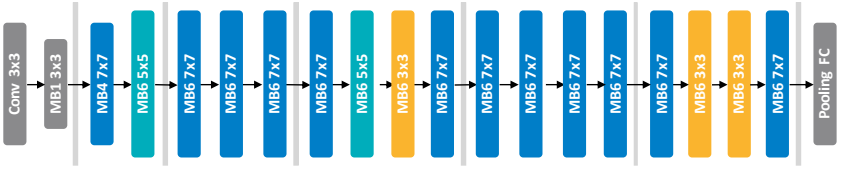
(e) PFNAS-200-Mobile.

Figure D: Architectures searched by PFNAS on Google Pixel1 phone.

(a) PFNAS-30-CPU.

(b) PFNAS-35-CPU.

(c) PFNAS-40-CPU.



(d) PFNAS-45-CPU.

(e) PFNAS-50-CPU.

Figure E: Architectures searched by PFNAS on Intel Core i5-7400 CPU.

(a) PFNAS-90-GPU.

(b) PFNAS-115-GPU.

(c) PFNAS-140-GPU.

(d) PFNAS-165-GPU.

(e) PFNAS-190-GPU.

Figure F: Architectures searched by PFNAS on NVIDIA TITAN X GPU.

REFERENCES

Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *International Conference on Machine Learning*, pp. 89–96, 2005.

Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019.

Han Cai, Chuang Gan, and Song Han. Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020.

Wei Chen, Tie-Yan Liu, Yanyan Lan, Zhi-Ming Ma, and Hang Li. Ranking measures and loss functions in learning to rank. In *Advances in Neural Information Processing Systems*, pp. 315–323, 2009.

Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1294–1303, 2019.

Yoav Freund, Raj Iyer, Robert E Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4(Nov):933–969, 2003.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1314–1324, 2019.

Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2261–2269, 2017.

Sian-Yao Huang and Wei-Ta Chu. Ponas: Progressive one-shot neural architecture search for very efficient deployment. *arXiv preprint arXiv:2003.05112*, 2020.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *International Conference on Learning Representations*, 2019.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning*, pp. 807–814, 2010.

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning*, pp. 4095–4104, 2018.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pp. 6105–6114, 2019.

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2820–2828, 2019.

Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 10734–10742, 2019.

Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient differentiable architecture search. In *International Conference on Learning Representations*, 2020.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.

Barret Zoph, V. Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8697–8710, 2018.