GNAS: A Greedy Neural Architecture Search Method for Multi-Attribute Learning

Siyu Huang[1†], Xi Li[1*], Zhi-Qi Cheng[2†], Zhongfei Zhang[1], Alexander Hauptmann[3]
[1]Zhejiang University, [2]Southwest Jiaotong University, [3]Carnegie Mellon University
{siyuhuang,xilizju,zhongfei}@zju.edu.cn,zhiqicheng@gmail.com,alex@cs.cmu.edu

ABSTRACT

A key problem in deep multi-attribute learning is to effectively discover the inter-attribute correlation structures. Typically, the conventional deep multi-attribute learning approaches follow the pipeline of manually designing the network architectures based on task-specific expertise prior knowledge and careful network tunings, leading to the inflexibility for various complicated scenarios in practice. Motivated by addressing this problem, we propose an efficient greedy neural architecture search approach (GNAS) to automatically discover the optimal tree-like deep architecture for multi-attribute learning. In a greedy manner, GNAS divides the optimization of global architecture into the optimizations of individual connections step by step. By iteratively updating the local architectures, the global tree-like architecture gets converged where the bottom layers are shared across relevant attributes and the branches in top layers more encode attribute-specific features. Experiments on three benchmark multi-attribute datasets show the effectiveness and compactness of neural architectures derived by GNAS, and also demonstrate the efficiency of GNAS in searching neural architectures.

KEYWORDS

multi-task learning, multi-attribute analysis, neural architecture search, greedy algorithm

ACM Reference Format:
Siyu Huang[1†], Xi Li[1*], Zhi-Qi Cheng[2†], Zhongfei Zhang[1], Alexander Hauptmann[3]. 2018. GNAS: A Greedy Neural Architecture Search Method for Multi-Attribute Learning. In *2018 ACM Multimedia Conference (MM '18), October 22–26, 2018, Seoul, Republic of Korea.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3240508.3240588

1 INTRODUCTION

As an important variant of multi-task learning [8] and transfer learning [28], multi-attribute learning aims to discover the underlying correlation structures among attributes, which can improve

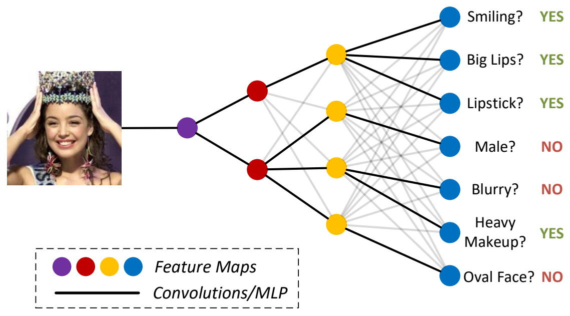

Figure 1: Brief illustration of neural architecture search in multi-attribute learning. Our goal is to automatically discover the optimal tree-like neural network architecture from the combinatorially large space in order to jointly predict the attributes.

the generalization performance of attribute prediction models by transferring and sharing information across multiple related attributes. With the representation power of deep learning [7, 24], the problem of discovering such correlation structures is typically cast as designing tree-structured neural networks, whose architectures capture the attribute ontology properties in the forms of shared parent trunk networks followed by different child branch networks. Namely, the more semantically correlated attributes will share more parent trunk network layers followed by individual attribute-specific branch network layers. In this way, building an effective neural architecture is a key issue to solve in multi-attribute learning.

Motivated by the above observations, a number of deep multi-attribute networks are built in a hand-designed way, which relies heavily on the expertise knowledge in specific tasks. In practice, such a way is often heuristic, inflexible, and incapable of well adapting to complicated real-world scenarios. In order to address this problem, we resort to automatically building the multi-attribute network architecture within an end-to-end learning framework. As illustrated in Fig. 1, our goal is to discover the optimal tree-like architecture, where its root node is the input image and its leaf nodes are the probabilities of attributes. The low-level representations are more commonly shared and high-level representations are more task-specific, nicely fitting the nature of multi-attribute learning. However, it is a very challenging task to search architectures within such a combinatorially large space of possible connections. First, the number of candidate architectures is an exponential complexity of attribute numbers. For the example of Fig. 1, the number of candidate architectures of the last layer (between 4 yellow nodes and 7 blue nodes) is 4^7=16,384. Second, it is computationally expensive

†This work was done when Siyu Huang and Zhi-Qi Cheng visited Carnegie Mellon University.
*Xi Li is the corresponding author.

to evaluate candidate architectures, as the evaluation has to be conducted after training a neural network to convergence.

In this paper, we propose a highly efficient greedy neural architecture search method (GNAS) to optimize the neural architecture for multi-attribute prediction. Inspired by the effective layer-wise pretraining strategy [2, 9, 10] proposed in earlier literature, we formulate the optimization of a global architecture as a series of sub-tasks of optimizing the independent layer architectures in a greedy manner. The optimization of a layer architecture is further divided into the optimizations of connections w.r.t individual attribute performance based on the property of tree structure. The optimal global architecture is derived by a combination of the optimal local architectures after iteratively updating the local architectures and the neural network weights.

Our proposed GNAS approach is efficient and effective in the following aspects:

- With the help of greedy strategies, GNAS reduces the number of candidate evaluated architectures from exponential complexity to linear complexity of the attribute number.
- GNAS could significantly accelerate the back propagation training of individual candidate architectures by incorporating the weight sharing mechanism [22, 23] across different candidate architectures.
- GNAS could be used for searching arbitrary tree-structured neural network architecture. The large search space of GNAS ensures the performance of its discovered architecture.
- GNAS is a non-parametric approach that it refrains from the loop of adopting extra parameters and hyper-parameters for meta-learning (such as Bayesian optimization (BO) [25] and reinforcement learning (RL) [31, 32]).

GNAS is not only theoretically reasonable, but also showing favorable performance in empirical studies. On three benchmark multi-attribute datasets, GNAS discovers network architectures on 1 GPU in no more than 2 days to beat the state-of-the-art multi-attribute learning methods with fewer parameters and faster testing speed.

The main contributions of this work are summarized as follows:

- We propose an innovative greedy neural architecture search method (GNAS) for automatically learning the tree-structured multi-attribute deep network architecture. In principle, GNAS is efficient due to its greedy strategies, effective due to its large search space, and generalized due to its non-parametric manner.
- Experimental results on benchmark multi-attribute learning datasets demonstrate the effectiveness and compactness of deep multi-attribute model derived by GNAS. In addition, detailed empirical studies are conducted to show the efficacy of GNAS itself.

2 RELATED WORK

Multi-attribute learning. Similar to multi-task learning [13], multi-attribute learning addresses the attribute prediction problems by feature sharing and joint optimization across related attributes. In the context of deep attribute learning, prior works [5, 7, 15, 24] investigate designing end-to-end tree-like network architecture which shares feature representations in bottom layers and encode task-specific information in top layers. The tree-like architecture is able to improve the compactness and generalization ability of deep models.

However, the hand-designed network architecture raises a high demand of knowledges in specific tasks and experience in building neural networks. Motivated by this, researchers investigate the automatic design of deep architectures more recently. Cross-stitching network [19] is proposed to learn an optimal linear combination of shared representations, and He et al. [8] adaptively learn the weights of individual tasks. The work most close to our approach is [17] which first initializes a thin network from a pre-trained model by SOMP [27] and then widening the network through a branching procedure. However, these approaches generally explore a relatively limited search space.

In this work, our proposed greedy neural architecture search method (GNAS) addresses the automatic design of deep multi-attribute architecture in an entirely different way. From the perspective of neural architecture optimization, GNAS divides the global architecture optimization problem into a series of local architecture optimization problems based on reasonable intra-layer and inter-layer greedy strategies. The greedy manner ensures the efficiency of architecture search procedure.

Neural architecture optimization. Deep neural network has achieved a great success on many tasks. While, the design of neural network architecture still relies on the expertise in neural network and prior knowledge of specific tasks. Recently, there is a growing amount of research focusing on the automatic design of neural network architecture, aiming at discovering the optimal neural architecture with less human involvement. A variety of approaches including random search [3], Bayesian optimization [12, 18, 25], evolutionary algorithm [23], and reinforcement learning [22, 31] are proposed for neural architecture optimization. The recently proposed neural architecture search (NAS) [31, 32] employs an RNN controller to sample candidate architectures and updating the controller under the guidance of performances of sampled architectures. Although models derived by NAS have shown impressive results on image classification and language modeling, the prohibitive expense of NAS limits its further development. As the learning of neural network is a black-box optimization, we have to evaluate an candidate neural architecture after it is trained to convergence. Typically, Zoph and Le [31] use 800 GPUs and 28 days to discover the convolutional architecture on Cifar-10 dataset by exploring 12,800 individual architectures.

Several approaches explore to accelerate the searching procedure by reducing the expense of neural network training. Baker et al. [1] early stop the architecture evaluation process by predicting the performance of unobserved architectures based on a set of architecture features. Brock et al. [4] propose a hypernetwork to generate the neural network weights conditioned on its architecture instead of conducting back propagation training. Pham et al. [22] search for an optimal sub-graph within a large computational graph where the neural network weights are shared across sub-graphs.

In this work, we propose GNAS to novelly develop neural architecture optimization to multi-task learning. Different from existing neural architecture optimizing approaches, we propose two greedy strategies which largely reduce the computation cost of architecture optimization procedure. The intra-layer greedy strategy of

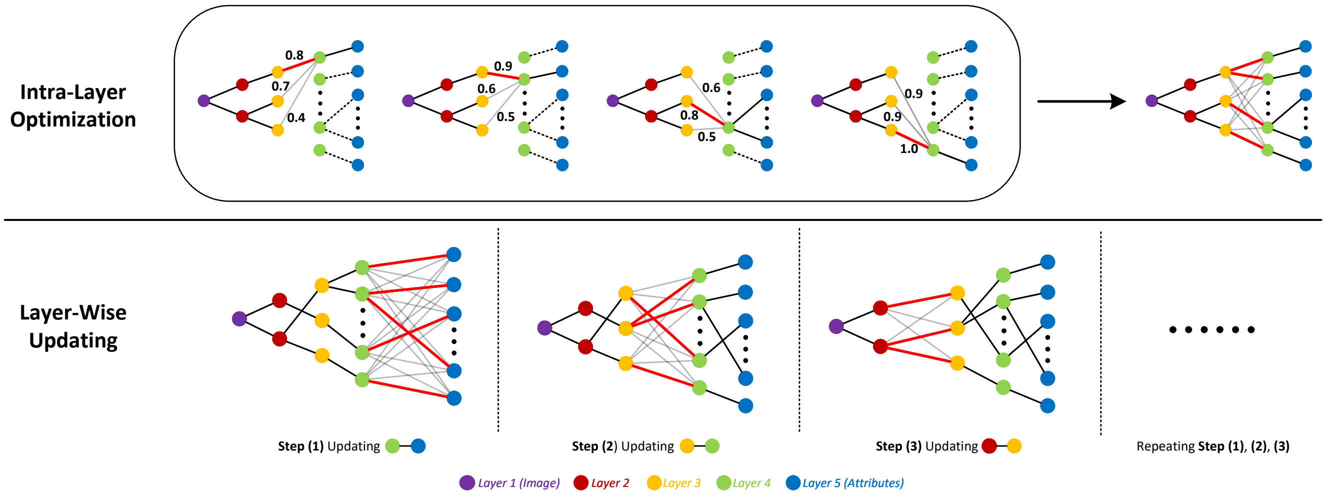

Figure 2: Illustration of our greedy neural architecture search (GNAS). We transform the difficult global architecture optimization problem into a series of local architecture optimization problems. The upper part illustrates the optimization of intra-layer connections, where we respectively evaluate all the connections and select the connections which have the best validation performances on their descendant attributes. The lower part illustrates the layer-wise updating procedure, where we iteratively update the architecture of one layer conditioned on the fixed architectures of the other layers.

GNAS is proposed based on the property of tree structure. And the inter-layer greedy strategy of GNAS is inspired by the layer-wise pretraining strategy of restricted Boltzmann machine (RBM) [2, 9, 10]. The greedy strategies lead to the efficiency of GNAS, also leading to effectiveness by ensuring a highly efficient searching in a very large search space.

3 OUR APPROACH

3.1 Problem Formulation

Our goal is to find the optimal tree-like neural network architecture $\hat{G}$ which has the maximum reward R

$$\begin{aligned}\hat{G} &= \arg\max_{G} R(G) \\ &= \arg\max_{G} \frac{1}{N}\sum_{n=1}^{N} r_n(G)\end{aligned} \tag{1}$$

R is defined as the mean prediction accuracy of attributes on validation set, where r_n is the prediction accuracy of the n-th attribute on validation set and N is the number of attributes. G is the multi-output network with an input of an image and N outputs for predicting N attributes. G is tree-like that it has M layers. In each layer l, there are B_l blocks where each block consists of a fixed number of feature maps. $B_1 = 1$ as the first layer is the input image and $B_M = N$ as the last layer is N outputs of attribute predictions. G hierarchically groups the related attributes from its top layers to bottom layers.

For convenience, we use a set of binary adjacency matrices A to denote the network topology of neural network G. $A^{(l)}_{i,j} = 1$ denotes that there is a connection (fixed as convolutions or MLP as needed) between the i-th block of layer l and the j-th block of layer l+1, otherwise, $A^{(l)}_{i,j} = 0$. We rewrite Eq. 1 as

$$\hat{A} = \arg\max_{A} R(A), \qquad \text{s.t.} \sum_{i=1}^{B_l} A^{(l)}_{i,j} = 1,\ 1 \le j \le B_{l+1} \tag{2}$$

A is constrained to be a tree structure under the constraint of Eq. 2.

Eq. 2 is a combinatorial optimization problem which has $\prod_l B_l^{B_{l+1}}$ possible solutions. Therefore, it is often infeasible to get its optimal solution due to the large solution space. For instance, for a neural network with 40 output predictions and a hidden layer of 10 blocks, the number of possible neural architectures is 10^{40}, such at we could not evaluate all of the possible architectures to find an optimal one.

In this work, we present a non-parametric approach, i.e., GNAS, to search for the multi-output tree-like neural architecture effectively and efficiently. Generally speaking, we divide the global optimization problem into the optimization problems of individual layer architectures, and further dividing them into the optimization problems of individual connections. The optimal global architecture is approximated by the combination of optimal local architectures. More details of our approach are discussed in the following sections.

3.2 Intra-Layer Optimization

Our GNAS starts from optimizing the neural network connection w.r.t. an individual attribute within a layer. Given the architecture of the other layers, the problem is formulated as

$$\arg\max_{A^{(l)}} r_n\left(A^{(l)} \,\middle|\, A^{(L)}, L \neq l\right), \qquad \text{s.t.} \sum_{i,j} A^{(l)}_{i,j} = 1 \tag{3}$$

Eq. 3 is easy to solve as our neural architecture is a tree structure, such that we only have to evaluate the connections between B_l blocks of layer l and the ancestor block of attribute n in layer l+1.

To optimize the connections of an entire layer, we propose a greedy assumption:

Assumption 1 *The optimal intra-layer architecture is composed by the optimal connections w.r.t. individual attributes.*

This assumption is definitely reasonable because our network structure is a tree. The connections from a block to its descendant attributes are unique thus the connections w.r.t. individual attributes in layer l are nearly independently when connections of the other layers are fixed. Based on Assumption 1, we reformulate the optimization of a layer as optimizing a set of Eq. 3 independently,

$$\begin{aligned}
&\arg\max_{A^{(l)}} R\left(A^{(l)} \mid A^{(L)}, L \neq l\right), \qquad \text{s.t.} \sum_{i,j} A^{(l)}_{i,j} = N \qquad (4)\\
=\ &\arg\max_{A^{(l)}} \frac{1}{N}\sum_{n=1}^{N} r_n\left(A^{(l)} \mid A^{(L)}, L \neq l\right), \quad \text{s.t.} \sum_{i,j} A^{(l)}_{i,j} = N\\
\simeq\ &\left\{\arg\max_{A^{(l)}} r_n\left(A^{(l)} \mid A^{(L)}, L \neq l\right), \quad \text{s.t.} \sum_{i,j} A^{(l)}_{i,j} = 1\right\} \text{ for } n = 1, \ldots, N
\end{aligned}$$

Note that there may be more than one connections built from layer l to a certain block of layer l+1 if B_{l+1}<N, leading to the destruction of the tree structure. To avoid this, we give each block an index $I^{(l)}_i \subseteq \{1, 2, \ldots, N\}$ denoting which attributes are the descendants of the i-th block of layer l. The network is tree-structured that the reward of a connection $A^{(l)}_{i,j}$ is exactly the average accuracy of its descendant attributes,

$$R\left(A^{(l)}_{i,j} \mid A^{(L)}, L \neq l\right) = \frac{1}{\left|I^{(l+1)}_j\right|}\sum_{n\in I^{(l+1)}_j} r_n\left(A^{(l)}_{i,j} \mid A^{(L)}, L \neq l\right) \qquad (5)$$

We optimize w.r.t. blocks instead of attributes, formulated as

$$\begin{aligned}
\text{Eq.4} \simeq \Bigg\{&\arg\max_{A^{(l)}} \frac{1}{\left|I^{(l+1)}_j\right|}\sum_{n\in I^{(l+1)}_j} r_n\left(A^{(l)} \mid A^{(L)}, L \neq l\right),\\
&\qquad \text{s.t.} \sum_{i,j} A^{(l)}_{i,j} = 1\Bigg\} \text{ for } j = 1, \ldots, B_{l+1}
\end{aligned} \qquad (6)$$

Eq. 6 is also easy to solve as we only have to evaluate B_l architectures for optimizing a block. Until now, the architectures evaluated within a layer is reduced from $B_l^{B_{l+1}}$ to $B_l \cdot B_{l+1}$.

The upper part of Fig. 2 illustrates a simple example of our searching process within a layer. In the example, we aim at optimizing the third layer of the neural architecture, i.e., the connections between yellow blocks and green blocks. The four sub-figures in the box respectively illustrate the optimizations w.r.t four green blocks. The connections with red lines are selected because they have higher rewards than the other candidate connections. Note that in the third sub-figure, the green block is the ancestor of two attributes, such that its reward is computed by averaging the validation accuracies of those two attributes. As shown in the upper right of Fig. 2, the optimal architecture of this layer is composed by the selected connections.

3.3 Accelerating Intra-Layer Search

Although the architectures searched within a layer is reduced from $B_l^{B_{l+1}}$ to $B_l \cdot B_{l+1}$ by Eq. 6, the computing cost is still large. We propose to further decrease the number of evaluated architectures from $B_l \cdot B_{l+1}$ to B_l. In fact, Eq. 5 indicates that we could get the reward of connection $A^{(l)}_{i,j}$ according to the accuracies of its descendant attributes. Therefore, we could evaluate the rewards of connection between a block in layer l and all the blocks in layer l+1 simultaneously, as there is a unique path between a layer and a certain attribute in this case.

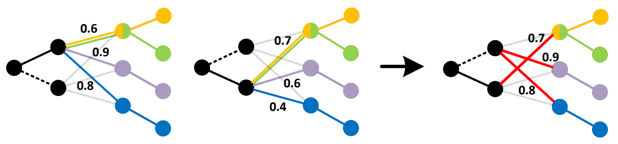

Figure 3: Accelerating the intra-layer search by evaluating the connections between a black block and the colored blocks of the next layer at the same time.

As illustrated in Fig. 3, we aim at optimizing the connections between black blocks and colored blocks. We do not have to evaluate the possible connections separately, that is, we could evaluate the connections between a black block and all the colored blocks simultaneously. The reward of each connection comes from the validation accuracies of its descendant attributes. The connections with larger rewards are selected, as shown in the right of Fig. 3.

3.4 Layer-Wise Updating

To optimize the connections of the entire network, we have a greedy assumption:

Assumption 2 *The optimal global architecture is composed by the optimal layer architectures.*

This assumption is proposed based on the effective layer-wise pretraining strategy [2, 9, 10] for initializing a restricted Boltzmann machine (RBM), where the weights of individual neural layers are separately pre-trained to ensure stable initialization. Similar to the neuron weights, the architectures of neural layers could also be viewed as the parameters of mapping functions. Thus, we propose Assumption 2 to layer-wise update the network architecture. The Eq. 2 and Eq. 4 are connected as Eq.2 $\simeq \left\{ \text{Eq.4} \right\}$ for $j = 1, 2, \ldots, M-1$. M is the number of layers. As discussed in Section 3.3, the number of evaluated architectures for updating a layer is B_l. Therefore, the number of evaluated architectures for optimizing the entire network is finally $\sum_{l=1}^{M-1} B_l$. For instance a network with 4 layers of (1, 4, 16, 40) blocks in each layer, there are $\prod_l B_l^{B_{l+1}} = 6.28 \times 10^{57}$ possible tree-structured architectures. By using our GNAS method, the number is decreased to $\sum_{l=1}^{M-1} B_l = 20$.

The lower part of Fig. 2 illustrates the layer-wise updating procedure. At every step, we update the connections of one layer while fixing the connections of the other layers based on Eq. 6. As the given condition $A^{(L)}, L \neq l$ in Eq. 6 will change after every update of the other layers, we repeat the layer-wise updating until convergence.

Weight sharing. To evaluate the performance of a neural architecture, we have to take a long time to train it to convergence first. Thanks to the weight inheritance mechanism [22, 23] proposed for neural architecture search, we share the weights of the same network connections across different architectures during the entire GNAS process. Specifically, we maintain the weights of network

Algorithm 1: Greedy neural architecture search (GNAS)

```
Input: Training set $D_{\text{train}}$, validation set $D_{\text{valid}}$, layer number
  $M$, block number $B$
Output: Neural network architecture $A$
1. Initialization
 - Randomly initialize architecture $A$ subject to Eq. 2;
 - Randomly initialize neural network weights $W$;
2. Updating
 - while not converged do
     -for $l$=$M$-1 downto 1 do
         - for $b$=1 to $B_l$ do
             - $A_{i,j}^{(l)} \leftarrow \begin{cases} 1, i = b \\ 0, i \neq b \end{cases}$;
             - Train $W[A]$ on batches of $D_{\text{train}}$;
             - $r(A) \leftarrow$ Evaluate $W[A]$ on batches of $D_{\text{valid}}$;
         - Update layer architecture $A^{(l)}$ based on $r$ by Eq. 6;
```

connections A as $W[A]$. In training phase, the weight of connection $A_{i,j}^{(l)}$ is inherited from $W\left[A_{i,j}^{(l)}\right]$, and $W\left[A_{i,j}^{(l)}\right]$ is updated after training $A_{i,j}^{(l)}$. When evaluating $A_{i,j}^{(l)}$, the weight of $A_{i,j}^{(l)}$ is inherited from $W\left[A_{i,j}^{(l)}\right]$. We alternately train the network on several mini-batches of training set to update weights W, and evaluate the network on validation set to update architecture A, such that both the weights W and the architecture A get to convergence in this process. The complete GNAS algorithm is illustrated in Alg. 1.

4 EXPERIMENTS

4.1 Implementation Details

Datasets. In the experiments, we evaluate our approach on two facial attribute datasets: CelebA [16] and LFWA [11], and one person attribute dataset: Market-1501 Attribute [14].

- **CelebA** dataset [16] consists of 200k images respectively with 160k, 20k, and 20k images for training, validation, and testing sets. CelebA dataset and LFWA dataset provide the same 40 binary attributes. We randomly crop the images of CelebA to size (192, 168) for training.
- **LFWA** dataset [11] consists of 13,143 images respectively with 6,263 and 6,880 for training and testing sets. As there is no official split of training and validation, we use the first 5,000 images in training set for training and the rest 1,263 images for validation. We randomly crop the images of LFWA to size (224, 224) for training.
- **Market-1501 Attribute** dataset [14] annotates 23 person attributes on the original Market-1501 dataset [30]. It has 32,688 images of 1501 identities, including 16,522 images of 751 identities for training and 17661 images of 750 identities for testing respectively. As there is no official split of training and validation, we use the first 13,000 images in training set for training and the rest 3,522 images for validation.

Standard image preprocessing including normalization and random horizontal flip is applied to all the three datasets.

Table 1: Network Architecture Configuration

Layer	Kernel	Shallow			Deep		
		Block	Channel		Block	Channel	
			Thin	Wide		Thin	Wide
Conv-1	7×7	1	16	64	1	16	64
Conv-2	3×3	1	32	128	**2**	16	64
Conv-3	3×3	1	64	256	**4**	16	64
Conv-4	3×3	**4**	32	128	**8**	16	64
Conv-5	3×3	**16**	16	32	**16**	16	32
FC-1	-	***N***	64	128	***N***	64	128
FC-2	-	*N*	64	128	*N*	64	128
FC-3	-	*N*	2	2	*N*	2	2

Network architecture. In this work, we propose GNAS to search for the optimal tree-structured neural network architecture which is a sub-graph of a pre-defined graph. In the experiments, we evaluate GNAS with several different configurations. As described in Table. 1, we use two versions of pre-defined graphs, including GNAS-Shallow and GNAS-Deep. GNAS-Shallow searches for connections within layers of *Conv-4*, *Conv-5*, and *FC-1*. GNAS-Deep searches for connections within layers of *Conv-2, 3, 4, 5*, and *FC-1*. *N* is the attribute number corresponding to different datasets. We do not update the last two FC layers, i.e., they are fixed for attribute regression. For each graph version, we also employ two versions of channel numbers including Thin and Wide, where Thin version has fewer channels of feature maps and Wide version has more channels of feature maps in every layer. After each convolutional layer, we adopt a Batch Normalization (BN) layer, an ReLU layer, and a max-pooling layer with a kernel size of 2×2 and a stride of 2. BNs are removed in inference phase for faster computing. Binary cross entropy loss is adopted at the output ends of the network to measure binary attribute predictions.

Learning configurations. The deep neural networks are implemented based on PyTorch [21] in our experiments. For the training of neural networks, we use SGD with the learning rate of 0.1, the batch size of 64, the weight decay of 10^{-4}, and the Nesterov momentum [20] of 0.9. We train a sub-graph for 2000 iterations on CelebA and 400 iterations on LFWA and Market-1501 every time. The learning rate is decayed by 0.96 after a round of layer-wise updating. We fine-tune the selected architecture on both of the training sets and the validation sets, then reporting its performance on the testing set. As the training on LFWA and Market-1501 may easily overfit to training set due to their small number of samples, we adopt a Dropout layer [26] with a drop rate of 0.75 after each fully-connected layer for LFWA and Market-1501.

Running costs. On CelebA, the neural network weights and network architecture get converged after 150 rounds of layer-wise updating, taking about 2 days on a GTX 1080Ti GPU. On LFWA and Market-1501, the weights and architecture get converged after 300 rounds of layer-wise updating, taking about 1 days on a GTX 1080Ti GPU.

4.2 Multi-Attribute Prediction

Facial attribute prediction. Table 2 compares our method with the state-of-the-art facial attribute prediction methods. The first group of methods design the model architectures by hand-craft,

Table 2: Comparison with State-of-the-Art Facial Attribute Learning Methods

Method	Mean Error (%) CelebA	Mean Error (%) LFWA	Params (million)	Test Speed (ms)	Adaptive?
LNets+ANet [16]	13	16	-	-	No
Separate Task [24]	9.78	-	-	-	No
MOON [24]	9.06	-	119.73	12.53	No
Independent Group [7]	8.94	13.72	-	-	No
MCNN [7]	8.74	13.73	-	-	No
MCNN-AUX [7]	8.71	13.69	-	-	No
VGG-16 Baseline [17]	8.56	-	134.41	12.60	No
Low-rank Baseline [17]	9.12	-	4.52	6.07	No
SOMP-thin-32 [17]	10.04	-	0.22	1.94	Yes
SOMP-branch-64 [17]	8.74	-	4.99	5.77	Yes
SOMP-joint-64 [17]	8.98	-	10.53	6.18	Yes
PaW-subnet [6]	9.11	-	0.27	-	Yes
PaW [6]	8.77	-	11	-	Yes
GNAS-Shallow-Thin	8.70	13.84	1.57	0.33	Yes
GNAS-Shallow-Wide	**8.37**	**13.63**	7.73	0.64	Yes
GNAS-Deep-Thin	9.10	14.12	1.47	0.87	Yes
GNAS-Deep-Wide	8.64	13.94	6.41	0.89	Yes

Table 3: Comparison of Person Attribute Learning Methods

Method	Market-1501 (%)
Ped-Attribute-Net [14]	13.81
Separate Models [8]	13.32
APR [14]	11.84
Equal-Weight [29]	13.16
Adapt-Weight [8]	11.51
Random-Thin	11.94
Random-Wide	11.42
GNAS-Thin	11.37
GNAS-Wide	**11.17**

and the second group of methods derive the model architectures from data, as denoted by the column of 'Adaptive?' in Table 2. The testing speeds of the other methods are cited from [17]. As [17] uses a Tesla K40 GPU (4.29 Tflops) and we use a GTX 1080Ti GPU (11.3 Tflops), we convert the testing speeds of their paper according to GPU flop number. In addition, we use a batch size of 32 in testing for a fair comparison with [17].

Table 2 shows that our GNAS models outperform the other state-of-the-art methods on both of CelebA and LFWA datasets, with faster testing speed, relatively fewer model parameters, and feasible searching costs (no more than 2 GPU-days). It demonstrates the effectiveness and efficiency of GNAS in multi-attribute learning. The fast testing speed of GNAS model is mainly due to its fewer convolution layers (5 layers) and tree-like feature sharing architecture. Comparing different models derived by GNAS, GNAS-Shallow models perform better than GNAS-Deep models with faster speed and almost equal number of parameters, indicating that it is better to share high-level convolutional feature maps for multi-attribute learning. GNAS-Wide models perform better than GNAS-Thin models with the reason of employing more model parameters.

Person attribute prediction. Table 3 compares GNAS with the state-of-the-art person attribute learning methods. We only test our GNAS-Shallow-Thin and GNAS-Shallow-Wide , as Market-1501 Attribute dataset [14] has fewer attributes (27 binary attributes). We also test the random architecture including Random-Thin and Random-Wide which have the same numbers of blocks and channels corresponding to GNAS-Thin and GNAS-Wide. Table 3 shows that GNAS-Wide still performs the best compared to other methods including the state-of-the-art methods and the random baselines. The Adapt-Weight [8] is also the adaptive method which adaptively learns the weights of tasks from data. Our method performs a little better, possibly due to the flexibility of GNAS-based models. GNAS-Thin and GNAS-Wide respectively outperform their random baselines by 0.57% and 0.25%, denoting the effectiveness of architectures derived by GNAS.

4.3 Efficiency of GNAS

It is known that the random search method is a strong baseline for black-box optimization [3, 32]. To demonstrate the effectiveness and efficiency of GNAS, we conduct more empirical studies on GNAS and random search.

A good neural architecture search method should firstly be able to find the architecture performing good on validation set. Fig. 4 shows the performances of architectures discovered by random search and GNAS on the validation set of LFWA dataset, along with the logarithmic time scale. In random search, we randomly sample the neural architectures and output the one which has the best validation performance in history. The numbers in legends are the number of mini-batches used for evaluating. For instance, GNAS-1 denotes that we evaluate the reward of an architecture on 1 mini-batches at a time.

In the left part of Fig. 4, we randomly initialize the weights W of neural network and make W fixed during the searching process. GNAS outperforms random search by large margin in this case. Starting from the randomly initialized architecture which has about 50% error rate on validation set, random search decreases the error rate to 43% in one hour, while, GNAS could decrease the error rate to 30% in fewer than 400 seconds. In addition, the number of

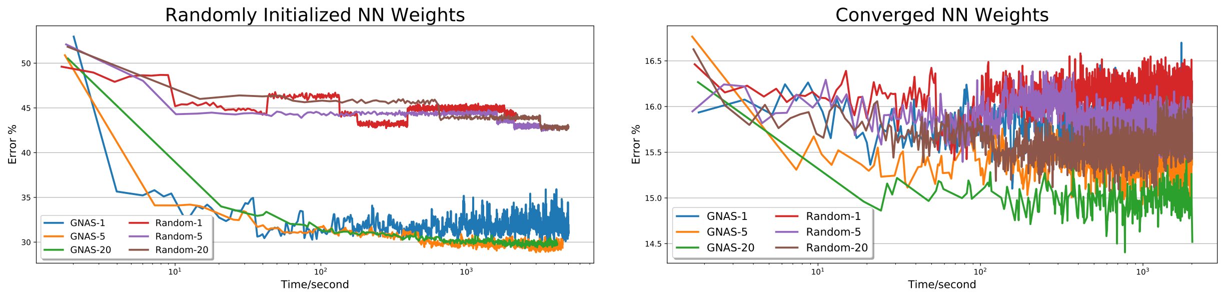

Figure 4: Architecture search experiments on neural networks of random initialized weights and nearly converged weights. We compare random search and GNAS with different numbers of validation samples {1, 5, 20}. GNAS methods significantly outperform the random search methods with better performance and faster convergence speed. More validation samples contributes to better performance. Best viewed in color.

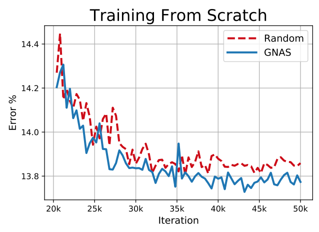

Figure 5: Testing errors of models derived by random search and GNAS with training from scratch on LFWA. GNAS model converges faster and better.

validation samples has a significant impact on the performance of architecture search methods. Random-5 performs better than Random-20 at the beginning, while Random-20 shows a better performance after enough long time. The error rate of Random-1 even increases at some time because of the larger variance brought by its fewer validation samples. Similarly, GNAS-1 has larger mean and variance of error rate than those of GNAS-5 and GNAS-20. GNAS-5 and GNAS-20 show similar performance, indicating that 5 mini-batches of validation samples are sufficient for GNAS in this case.

In the right part of Fig. 4, we inherit the neural network weights W from a well-trained neural network and also fix W during the searching process. Compared to the left part of Fig. 4, the error rates of different methods are closer to each other in the right part of Fig. 4. While, it is distinct that GNAS-20 performs the best and GNAS-5 performs the second-best. It demonstrates that GNAS could find better architecture than random search at different stages of the neural network training procedure. In addition, it reminds that GNAS should reduce its variance when searching architecture on a well-trained neural network by employing more validation samples.

We also evaluate the performances of architectures derived by GNAS and random baseline on the testing set. As shown in Fig. 5, we train from scratch the architectures on LFWA dataset. The testing error rates of GNAS model and random baseline model are respectively shown as the solid line and the dashed line. The GNAS model performs better than the random baseline model with faster convergence speed and lower error rate. In summary, both the empirical results on validation set and testing set reveal the effectiveness and efficiency of our GNAS.

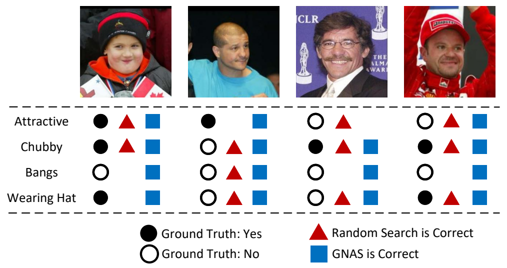

Figure 6: Some qualitative results of LFWA dataset. GNAS model makes more correct predictions than random search model.

We further show some qualitative results in Fig. 6. In Fig. 6, the images and attributes come from the testing set of LFWA dataset. The ground truth annotations of ‘Yes’ and ‘No’ are respectively denoted by solid circles and hollow circles. The correct predictions of random baseline model and GNAS model are respectively denoted by triangles and squares. It is clear that GNAS performs better than random search in most of the cases. GNAS makes only an incorrect prediction on ‘Attractive’ attribute of the third image, possibly due to the subjectivity of ‘Attractive’. These qualitative results also reveal the effectiveness of GNAS.

4.4 Study on Attributes

Per-attribute performance. We additionally study the individual attributes in multi-attribute learning. Table 4 lists the per-attribute error rates of different methods on CelebA Dataset. We compare our GNAS-Shallow-Wide model to the state-of-the-art methods including LANet [16], Inde. [7], MCNN [7], M-AUX [7] and PaW [6]. Results of the other methods are cited from the corresponding papers. Table 4 shows that GNAS not only performs the best under metric of average error rate, but also performs the best on 37 of the 40 attributes. Only on attributes of ‘Attractive’, ‘Oval Face’, ‘Rosy Cheeks’, and ‘Wearing Necktie’, GNAS performs equally or a little worse compared to the other methods. It is interesting that these attributes are relatively global facial features, possibly because the tree-structured neural network architecture may be better at modeling local features while be worse at modeling global features. This makes sense as M-AUX model [7] densely connects all of the attributes at the last layer of its neural network, such that

Table 4: Per-Attribute Performances on CelebA Dataset

Attribute \ Method	LANet	Inde.	MCNN	M-AUX	PaW	GNAS
5'o Clock Shadow	9.00	6.06	5.59	5.49	5.36	**5.24**
Arched Eyebrows	21.00	16.84	16.45	16.58	16.99	**15.75**
Attractive	19.00	17.78	17.06	**16.94**	17.14	**16.94**
Bags Under Eyes	21.00	15.17	15.11	15.08	15.42	**14.13**
Bald	2.00	1.15	1.13	1.10	1.07	**1.04**
Bangs	5.00	4.01	3.96	3.95	4.07	**3.80**
Big Lips	32.00	29.20	28.80	28.53	28.54	**28.21**
Big Nose	22.00	15.53	15.50	15.47	16.37	**14.90**
Black Hair	12.00	10.59	10.13	10.22	10.16	**9.76**
Blond Hair	5.00	4.12	4.03	3.99	4.15	**3.89**
Blurry	16.00	3.93	3.92	3.83	3.89	**3.58**
Brown Hair	20.00	11.25	11.01	10.85	11.50	**10.25**
Bushy Eyebrows	10.00	7.13	7.20	7.16	7.38	**7.01**
Chubby	9.00	4.45	4.34	4.33	4.54	**4.07**
Double Chin	8.00	3.57	3.59	3.68	3.74	**3.52**
Eyeglasses	1.00	0.33	0.37	0.37	0.41	**0.31**
Goatee	5.00	2.87	2.70	2.76	2.62	**2.41**
Gray Hair	3.00	1.93	1.80	1.80	1.79	**1.63**
Heavy Makeup	10.00	9.05	8.63	8.45	8.47	**8.18**
High Cheekbones	12.00	12.66	12.45	12.42	12.56	**11.95**
Male	2.00	1.98	1.84	1.83	1.61	**1.50**
Mouth Slightly Open	8.00	6.01	6.26	6.26	5.95	**5.84**
Mustache	5.00	3.33	3.07	3.12	3.10	**2.97**
Narrow Eyes	19.00	12.78	12.84	12.77	12.44	**12.34**
No Beard	5.00	4.07	3.89	3.95	3.78	**3.70**
Oval Face	34.00	25.30	24.19	**24.16**	24.97	24.43
Pale Skin	9.00	2.93	2.99	2.95	2.92	**2.76**
Pointy Nose	28.00	22.53	22.53	22.53	22.65	**21.76**
Receding Hairline	11.00	6.59	6.19	6.19	6.56	**6.06**
Rosy Cheeks	10.00	4.98	4.87	**4.84**	4.93	4.99
Sideburns	4.00	2.23	2.18	2.15	2.36	**2.04**
Smiling	8.00	7.35	7.34	7.27	7.27	**6.76**
Straight Hair	27.00	17.38	16.61	16.42	16.48	**15.23**
Wavy Hair	20.00	16.76	16.08	16.09	15.93	**15.48**
Wearing Earrings	18.00	9.65	9.68	9.57	10.07	**9.02**
Wearing Hat	1.00	1.03	0.96	0.95	0.98	**0.88**
Wearing Lipstick	7.00	6.20	6.05	5.89	5.76	**5.59**
Wearing Necklace	29.00	13.59	13.18	13.37	12.30	**12.39**
Wearing Necktie	7.00	3.29	3.47	3.49	**3.15**	3.24
Young	13.00	12.02	11.70	11.52	11.41	**11.11**
Ave.	12.67	8.94	8.74	8.71	8.77	**8.37**

the outputs of the global attributes could obtain more high-level semantic information from other local attributes, with the expense of larger model complexity.

Architecture visualization. Fig. 7 shows the network architecture derived by GNAS-Shallow-Thin on LFWA dataset. The neural architecture is tree-structured, where the centering purple node is the root block, and the numbered blue nodes are the 40 attributes. We could find many groupings of attributes which accord with intuition distinctly, and we highlight them with different colors in the caption of Fig. 7. For instance, in Fig. 7, 'Bald', 'Straight Hair', and 'Wavy Hair' are clearly related. '5'o Clock Shadow', 'Arched Eyebrows', and 'Bushy Eyebrows' are related to the facial hairs. We also observe that some related attributes are grouped at the lower layers. For instance, 'Bangs' and 'Sideburns' are hairs around face, and they are grouped with the facial hairs group at the lower layer. These reasonable attribute groupings qualitatively demonstrate the effectiveness of our GNAS.

5 DISCUSSIONS

In this paper, we have presented a highly efficient and effective greedy neural architecture search method (GNAS) for the automatic learning of multi-attribute deep network architecture. We have presented reasonable greedy strategies to divide the optimization of global architecture into the optimizations of individual connections step by step, such that the optimal global architecture is composed by the optimal local architectures.

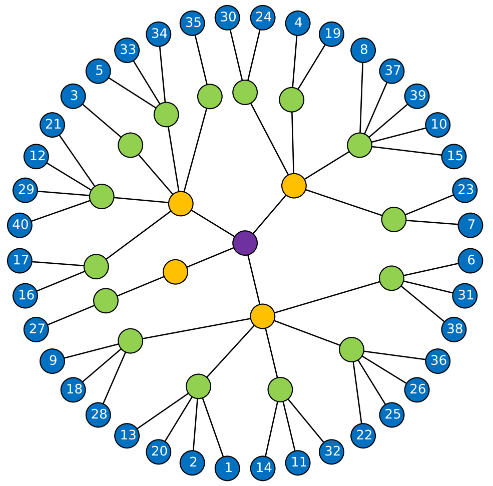

Figure 7: Network architecture discovered by GNAS-Shallow-Thin on LFWA, where related attributes are hierarchically grouped together. The 40 attributes are: 1 *5'o Clock Shadow* **2** *Arched Eyebrows* **3** *Attractive* **4** *Bags Under Eyes* **5** *Bald* **6** *Bangs* **7** *Big Lips* **8** *Big Nose* **9** *Black Hair* **10** *Blond Hair* **11** *Blurry* **12** *Brown Hair* **13** *Bushy Eyebrows* **14** *Chubby* **15** *Double Chin* **16** *Eyeglasses* **17** *Goatee* **18** *Gray Hair* **19** *Heavy Makeup* **20** *High Cheekbones* **21** *Male* **22** *Mouth Slightly Open* **23** *Mustache* **24** *Narrow Eyes* **25** *No Beard* **26** *Oval Face* **27** *Pale Skin* **28** *Pointy Nose* **29** *Receding Hairline* **30** *Rosy Cheeks* **31** *Sideburns* **32** *Smiling* **33** *Straight Hair* **34** *Wavy Hair* **35** *Wearing Earrings* **36** *Wearing Hat* **37** *Wearing Lipstick* **38** *Wearing Necklace* **39** *Wearing Necktie* **40** *Young*

GNAS is efficient due to its greedy strategies and effective due to its large search space. In experiments, GNAS discovers network architecture on 1 GPU in no more than 2 days to outperform the state-of-the-art multi-attribute learning models with fewer parameters and faster testing speed. Quantitative and qualitative studies have been further conducted to validate the efficacy of GNAS.

GNAS is a universal neural architecture search framework, such that it is able to be applied to tree-structured network with arbitrary NN blocks and connections. We can arbitrarily specify the type of an individual block (e.g., vector, 2D feature map), and the type of an individual connection (e.g., MLP, 1D convolutions, 2D convolutions, or even more complex NN architectures) as long as the shape of that connection is valid between two blocks. In the future study, it is encouraged to develop GNAS to various application scenarios by accommodating different optimization techniques of AutoML.

ACKNOWLEDGEMENTS

The authors thank for Hieu Pham's beneficial dicussions. This work is supported by NSFC (U1509206, 61672456, 61472353, 61751209, and 61772436), the fundamental research funds for central universities in China (2017FZA5007), National Basic Research Program of China (2015CB352302), the Key Program of Zhejiang Province, China (2015C01027), Sichuan Science and Technology Innovation Seedling Fund (2017RZ0015, 2017020), Artificial Intelligence Research Foundation of Baidu Inc., and the funding from HIKVision and ZJU Converging Media Computing Lab.

REFERENCES

[1] Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. 2017. Accelerating neural architecture search using performance prediction. *arXiv preprint arxiv, 1705.10823* (2017).

[2] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. 2007. Greedy layer-wise training of deep networks. In *NIPS*. 153–160.

[3] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* 13, Feb (2012), 281–305.

[4] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. 2018. SMASH: one-shot model architecture search through hypernetworks. In *ICLR*.

[5] Jiajiong Cao, Yingming Li, and Zhongfei Zhang. 2018. Partially Shared Multi-Task Convolutional Neural Network with Local Constraint for Face Attribute Learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4290–4299.

[6] Hui Ding, Hao Zhou, Shaohua Kevin Zhou, and Rama Chellappa. 2017. A Deep Cascade Network for Unaligned Face Attribute Classification. *arXiv preprint arXiv:1709.03851* (2017).

[7] Emily M Hand and Rama Chellappa. 2017. Attributes for Improved Attributes: A Multi-Task Network Utilizing Implicit and Explicit Relationships for Facial Attribute Classification.. In *AAAI*. 4068–4074.

[8] Keke He, Zhanxiong Wang, Yanwei Fu, Rui Feng, Yu-Gang Jiang, and Xiangyang Xue. 2017. Adaptively Weighted Multi-task Deep Network for Person Attribute Classification. In *ACM MM*. ACM, 1636–1644.

[9] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. 2006. A fast learning algorithm for deep belief nets. *Neural computation* 18, 7 (2006), 1527–1554.

[10] Geoffrey E Hinton and Ruslan R Salakhutdinov. 2006. Reducing the dimensionality of data with neural networks. *science* 313, 5786 (2006), 504–507.

[11] Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. 2007. *Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments*. Technical Report 07-49. University of Massachusetts, Amherst.

[12] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric Xing. 2018. Neural Architecture Search with Bayesian Optimisation and Optimal Transport. *arXiv preprint arXiv:1802.07191* (2018).

[13] Xuelong Li, Zhigang Wang, and Xiaoqiang Lu. 2017. A Multi-Task Framework for Weather Recognition. In *ACM MM*. ACM, 1318–1326.

[14] Yutian Lin, Liang Zheng, Zhedong Zheng, Yu Wu, and Yi Yang. 2017. Improving person re-identification by attribute and identity learning. *arXiv preprint arXiv:1703.07220* (2017).

[15] Decheng Liu, Nannan Wang, Chunlei Peng, Jie Li, and Xinbo Gao. 2018. Deep Attribute Guided Representation for Heterogeneous Face Recognition. In *IJCAI*. 835–841.

[16] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. 2015. Deep Learning Face Attributes in the Wild. In *ICCV*.

[17] Yongxi Lu, Abhishek Kumar, Shuangfei Zhai, Yu Cheng, Tara Javidi, and Rogerio Feris. 2017. Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification. In *CVPR*. 5334–5343.

[18] Hector Mendoza, Aaron Klein, Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. 2016. Towards automatically-tuned neural networks. In *Workshop on Automatic Machine Learning*. 58–65.

[19] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. 2016. Cross-stitch networks for multi-task learning. In *CVPR*. 3994–4003.

[20] Yurii Nesterov. 1983. A method of solving a convex programming problem with convergence rate O (1/k2). In *Soviet Mathematics Doklady*, Vol. 27. 372–376.

[21] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *NIPS-W*.

[22] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. 2018. Efficient Neural Architecture Search via Parameter Sharing. *arXiv preprint arXiv:1802.03268* (2018).

[23] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Quoc Le, and Alex Kurakin. 2017. Large-scale evolution of image classifiers. *arXiv preprint arXiv:1703.01041* (2017).

[24] Ethan M Rudd, Manuel Günther, and Terrance E Boult. 2016. Moon: A mixed objective optimization network for the recognition of facial attributes. In *ECCV*. 19–35.

[25] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical bayesian optimization of machine learning algorithms. In *NIPS*. 2951–2959.

[26] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.

[27] Joel A Tropp, Anna C Gilbert, and Martin J Strauss. 2006. Algorithms for simultaneous sparse approximation. Part I: Greedy pursuit. *Signal Processing* 86, 3 (2006), 572–588.

[28] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks?. In *NIPS*. 3320–3328.

[29] Ning Zhang, Manohar Paluri, Marc'Aurelio Ranzato, Trevor Darrell, and Lubomir Bourdev. 2014. Panda: Pose aligned networks for deep attribute modeling. In *CVPR*. 1637–1644.

[30] Liang Zheng, Liyue Shen, Lu Tian, Shengjin Wang, Jingdong Wang, and Qi Tian. 2015. Scalable person re-identification: A benchmark. In *ICCV*. 1116–1124.

[31] Barret Zoph and Quoc V Le. 2017. Neural architecture search with reinforcement learning. In *ICLR*.

[32] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2018. Learning transferable architectures for scalable image recognition. In *CVPR*.