Fully-adaptive Feature Sharing in Multi-Task Networks with Applications in Person Attribute Classification

Yongxi Lu
UC San Diego
yol070@ucsd.edu

Abhishek Kumar
IBM Research
abhishk@us.ibm.com

Shuangfei Zhai
Binghamton Univeristy, SUNY
szhai2@binghamton.edu

Yu Cheng
IBM Research
chengyu@us.ibm.com

Tara Javidi
UC San Diego
tjavidi@eng.ucsd.edu

Rogerio Feris
IBM Research
rsferis@us.ibm.com

Abstract

Multi-task learning aims to improve generalization performance of multiple prediction tasks by appropriately sharing relevant information across them. In the context of deep neural networks, this idea is often realized by hand-designed network architectures with layers that are shared across tasks and branches that encode task-specific features. However, the space of possible multi-task deep architectures is combinatorially large and often the final architecture is arrived at by manual exploration of this space, which can be both error-prone and tedious. We propose an automatic approach for designing compact multi-task deep learning architectures. Our approach starts with a thin multi-layer network and dynamically widens it in a greedy manner during training. By doing so iteratively, it creates a tree-like deep architecture, on which similar tasks reside in the same branch until at the top layers. Evaluation on person attributes classification tasks involving facial and clothing attributes suggests that the models produced by the proposed method are fast, compact and can closely match or exceed the state-of-the-art accuracy from strong baselines by much more expensive models.

1. Introduction

Humans possess a natural yet remarkable ability of seamlessly transferring and sharing knowledge across multiple related domains while doing inference for a given task. Effective mechanisms for sharing *relevant* information across multiple prediction tasks (referred as *multi-task learning*) are also arguably crucial for making significant advances towards machine intelligence. In this paper, we propose a novel approach for multi-task learning in the context of deep neural networks for computer vision tasks. We particularly aim for two desirable characteristics in the proposed approach: (i) *automatic learning of multi-task architectures* based on branching, (ii) *selective sharing* among tasks with automated learning of whom to share with. In addition, we want our multi-task models to have low memory footprint and low latency during prediction (forward pass through the network).

A natural approach for enabling sharing across multiple tasks is to share model parameters (partially or fully) across the corresponding layers of the task-specific deep neural networks. Most of the multi-task deep architectures share the bottom layers till some layer $\mathbf{l}$ after which the sharing is blocked, resulting in task-specific sub-networks or branches beyond it [28, 17, 13]. This is motivated by the observation made by several earlier works that bottom layers capture low level detailed features, which can be shared across multiple tasks, whereas top layers capture features at a higher level of abstraction that are more task specific. It can be further extended to a more general tree-like architecture, e.g., a smaller group of tasks can share parameters even after the first break-point at layer $\mathbf{l}$ and breakup at a later layer. However, the space of such possible branching architectures is combinatorially large and current approaches largely make a decision based on limited manual exploration of this space, often biased by designer's perception of the relationship among different tasks [25]. Finding a data-driven approach to replace the manual exploration is an important and interesting academic question that has only been sparsely explored.

The proposed approach operates in a greedy top-down manner, making branching and task-grouping decisions at each layer of the network using a novel criterion that promotes the creation of separate branches for unrelated tasks (or groups of tasks) while penalizing for the model complexity. To address the issue of scalability to multiple tasks, the proposed approach starts with a *thin* network and dynamically grows it during the training phase by creating new branches based on the aforementioned criterion. We

also propose a method based on simultaneous orthogonal matching pursuit (SOMP) [41] for initializing a thin network from a pretrained wider network (e.g., VGG-16) as a side contribution in this work.

We test this idea on facial (CelebA [24]), clothing (DeepFashion [23]) and person (facial combined with clothing) attribute classification, in all cases treating each attribute as a separate task. In these experiments, the models designed by our approach closely match state-of-the-art accuracy while being up to 90x more compact and 3x faster than the widely adopted VGG-16 architecture.

In summary, our main contributions are listed below:

- We propose to automate learning of multi-task deep network architectures through a novel dynamic branching procedure, which makes task grouping decisions at each layer of the network (deciding with whom each task should share features) by taking into account both task relatedness and complexity of the model.
- A novel method based on Simultaneous Orthogonal Matching Pursuit is proposed for initializing a thin network from a wider pre-trained network model, leading to faster convergence and higher accuracy.
- We demonstrate effectiveness of the proposed method on facial, clothing and joint person (facial+clothing) attribute classification, and investigate the behavior of the method through ablation studies.

2. Related Work

Multi-Task Learning. There is a long history of research in multi-task learning [4, 39, 16, 21, 25]. Most proposed techniques assume that all tasks are related and appropriate for joint training. A few methods have addressed the problem of "with whom" each task should share features [44, 16, 50, 18, 21, 26]. These methods are generally designed for shallow classification models, while our work investigates feature sharing among tasks in hierarchical models such as deep neural networks.

Recently, several methods have been proposed for multi-task learning using deep neural networks. HyperFace [28] simultaneously learns to perform face detection, landmarks localization, pose estimation and gender recognition. UberNet [19] jointly learns low-, mid-, and high-level computer vision tasks using a compact network model. MultiNet [3] exploits recurrent networks for transferring information across tasks. Cross-ResNet [17] connects tasks through residual learning for knowledge transfer. However, all these methods rely on *hand-designed* network architectures composed of base layers that are shared across tasks and specialized branches that learn task-specific features.

As network architectures become deeper, defining the right level of feature sharing across tasks through hand-crafted network branches is impractical. Cross-stitching networks [25] have been recently proposed to learn an optimal combination of shared and task-specific representations. Although cross-stitching units connecting task-specific sub-networks are designed to *learn* the feature sharing among tasks, the size of the network grows linearly with the number of tasks, causing scalability issues. We instead propose a novel algorithm that makes decisions about branching based on task relatedness, while optimizing for the efficiency of the model. We note that other techniques such as HD-CNN [45] and Network of Experts [1] also group related classes to perform hierarchical classification, but these methods are not applicable for the multi-label setting (where labels are not mutually exclusive).

Model Compression and Acceleration. Our method achieves model compression and acceleration by considering task relatedness. It is complementary to existing task-agnostic approaches, such as knowledge distillation [12, 29], low-rank-factorization [14, 38, 32], pruning and quantization [10, 27], structured matrices [6, 34, 9], and dynamic capacity networks [2]. Many of these state-of-the-art compression techniques can be used to further reduce the size of our learned multi-task architectures.

Person Attribute Classification. Methods for recognizing attributes of people, such as facial and clothing attributes, have received increased attention in the past few years. In the visual surveillance domain, person attributes serve as features for improving person re-identification [35] and enable search of suspects based on their description [42, 8]. In e-commerce applications, these attributes have proven effective in improving clothing retrieval [13], and fashion recommendation [22]. It has also been shown that facial attribute prediction is helpful as an auxiliary task for improving face detection [46] and face alignment [49].

State-of-the-art methods for person attribute prediction are based on deep convolutional neural networks [43, 24, 5, 48]. Most methods either train separate classifiers per attribute [48] or perform joint learning with a fully shared network [31]. Multi-task networks have been used with base layers that are shared across all attributes, and branches to encode task-specific features for each attribute category [13, 36]. However, in contrast to our work, the network branches are hand-designed and do not exploit the fact that some attributes are more related than others in order to determine the level of sharing among tasks in the network. Moreover, we show that our approach produces a single compact network that can predict both facial and clothing attributes simultaneously.

3. Methodology

Let the linear operation in a layer l of the network be paramterized by $\mathbf{W}^l$. Let $\mathbf{x}^l \in \mathbb{R}^{c_l}$ be the input vector of layer l, and $\mathbf{y}^l \in \mathbb{R}^{c_{l+1}}$ be the output vector. In feedforward networks that are of interest to this work, it is always the case that $\mathbf{x}^l = \mathbf{y}^{l-1}$. In other words, the output of a layer

is the input to the layer above. In vision applications, the feature maps are often considered as three-way tensors and one should think of $\mathbf{x}^l$ and $\mathbf{y}^l$ as appropriately vectorized versions of the input and output feature tensors. The functional form of the network is a series of within-layer computations chained in a sequence linking the lowest to the highest (output) layer. The within-layer computation (for both convolutional and fully-connected layers) can be concisely represented by a simple linear operation parametrized by $\mathbf{W}^l$, followed by a non-linearity $\sigma_l(\cdot)$ as

$$\mathbf{y}^l = \sigma_l(\mathbf{P}_l(\mathbf{W}^l)\mathbf{x}^l), \tag{1}$$

where $\mathbf{P}_l$ is an operator that maps the parameters $\mathbf{W}^l$ to the appropriate matrix $\mathbf{P}_l(\mathbf{W}^l) \in \mathbb{R}^{c_{l+1} \times c_l}$. For a fully connected layer $\mathbf{P}_l$ reduces to the identity operator, whereas for a convolutional layer with $\mathbf{f}_l$ filters, $\mathbf{W}^l \in \mathbb{R}^{f_l \times d_l}$ contains the vectorized filter coefficients in each row and the operator $\mathbf{P}_l$ maps it to an appropriate matrix that represents convolution as matrix multiplication. We define the width of the network at layer $\mathbf{l}$ as $\mathbf{c}_l$ for the fully connected layers, and as $\mathbf{f}_l$ for the convolutional layers.

The widths at different layers are critical hyper-parameters for a network design. Successful deep convolutional network architectures, such as AlexNet [20], VGG [33], Inception [37] and ResNet [11] all use wider layers at the top of the network in what can be called an "inverted pyramid" pattern. From visualization of filters at different layers [47] it is observed that top level features tend to be task-dependent. More recently, researchers have noted that the width schedule (especially at the top layers) need to be tuned for the underlying set of tasks the network has to perform in order to achieve best accuracy [25]. Motivated by these findings, our approach starts with a "flat" architecture that has a similar width at all layers, then expands it to create explicit task-specific branches. The procedure is as follows:

Thin Model Initialization. Start with a thin neural network model, use random initialization or optionally initialize it from a pre-trained wider VGG-16 model by selecting a subset of filters using simultaneous orthogonal matching pursuit (ref. Section 3.1).

Adaptive Model Widening. The thin initialized model goes through a multi-round widening and training procedure. The widening is done in a greedy top-down layer-wise manner starting from the top layer. For the current layer to be widened, our algorithm makes a decision on the number of branches to be created at this layer along with task assignments for each branch. The network architecture is frozen when the algorithm decides to create no further branches (ref. Section 3.2).

Training with the Final Model. In this last phase, the fixed final network is trained until convergence.

More technical details are discussed in the next few sections. Algorithm 1 provides a summary of the procedure.

Algorithm 1: Training with Adaptive Widening

Data: Input data $\mathbf{D} = (\mathbf{x}_i, \mathbf{y}_i)_{i=1}^{N}$. The labels $\mathbf{y}$ are for a set of $\mathbf{T}$ tasks.
Input: Branch factor $\boldsymbol{\alpha}$, and thinness factor $\boldsymbol{\omega}$. Optionally, a pre-trained network $\mathbf{M_p}$ with parameters $\Theta_\mathbf{p}$.
Result: A trained network $\mathbf{M_f}$ with parameters $\Theta_\mathbf{f}$.
Initialization: $\mathbf{M_0}$ is a thin-$\boldsymbol{\omega}$ model with $\mathbf{L}$ layers.
if *exist* $\mathbf{M_p}, \Theta_\mathbf{p}$ **then**
  $\Theta_\mathbf{0} \leftarrow$ *SompInit*$(\mathbf{M_0}, \mathbf{M_p}, \Theta_\mathbf{p})$. $\mathbf{t} \leftarrow 1$, $\mathbf{d} \leftarrow \mathbf{T}$. (Sec. 3.1)
else
  $\Theta_\mathbf{0} \leftarrow$ Random initialization
while *(*$\mathbf{t} \le \mathbf{L}$*) and (*$\mathbf{d} > 1$*)* **do**
  $\Theta_\mathbf{t}, \mathbf{A_t} \leftarrow$ *TrainAndGetAffinity*$(\mathbf{D}, \mathbf{M_t}, \Theta_\mathbf{t})$ (Sec. 3.3)
  $\mathbf{d} \leftarrow$ *FindNumberBranches*$(\mathbf{M_t}, \mathbf{A_t}, \boldsymbol{\alpha})$ (Sec. 3.4)
  $\mathbf{M_{t+1}}, \Theta_\mathbf{t+1} \leftarrow$ *WidenModel*$(\mathbf{M_t}, \Theta_\mathbf{t}, \mathbf{A_t}, \mathbf{d})$ (Sec. 3.2)
  $\mathbf{t} \leftarrow \mathbf{t} + 1$
Train model $\mathbf{M_t}$ with sufficient iterations, update $\Theta_\mathbf{t}$.
$\mathbf{M_f} \leftarrow \mathbf{M_t}$, $\Theta_\mathbf{f} \leftarrow \Theta_\mathbf{t}$.

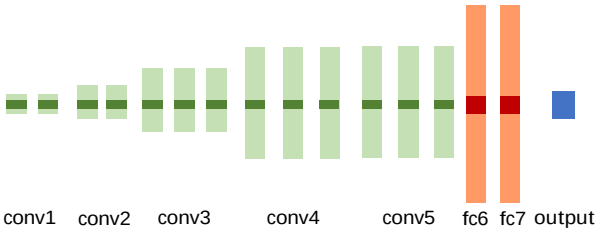

Figure 1. Comparing the thin model with VGG-16. As shown, the light color blobs shows the layers in the VGG-16 architecture. It has an inverted pyramid structure with a width plan of 64-128-256-512-512-4096-4096. The dark color blobs shows a thin network with $\boldsymbol{\omega} = 32$. The convolutional layers all have widths of 32, and the fully connected layers have widths of 64.

3.1. Thin Networks and Filter Selection using Simultaneous Orthogonal Matching Pursuit

The initial model we use is a thin version of the VGG-16 network. It has the same structure as VGG-16 except for the widths at each layer. We experiment with a range of thin models that are denoted as thin-$\boldsymbol{\omega}$ models. The width of a convolutional layer of the thin-$\boldsymbol{\omega}$ model is the minimum between $\boldsymbol{\omega}$ and the width of the corresponding layer of the VGG-16 network. The width of the fully connected layers are set to $2\boldsymbol{\omega}$. We shall call $\boldsymbol{\omega}$ the "thinness factor". Figure 1 illustrates a thin model side by side with VGG-16.

Using weights from pre-trained models is known to speed up training and improve model generalization. However, the standard direct copy method is only suitable when the source and the target networks have the same architecture (at least for most of the layers). Our adoption of a thin initial model forbids the use of direct copy, as there is a mismatch in the dimension of the weight matrix (for both the input and output dimensions, see Equation 1 and discus-

Algorithm 2: SompInit($\mathbf{M_0}$, $\mathbf{M_p}$, Θ_p)

Input: The architecture of the thin network $\mathbf{M_0}$ with L layers. The pretrained network and its parameters $\mathbf{M_p}$, Θ_p. Denote the weight matrix at layer l as $\mathbf{W}^{p,l} \in \Theta_p$.

Result: The initial parmaeters of the thin network Θ_0.

foreach $l \in 1, 2, \cdots, L$ **do**

  Find $\omega^*(l)$ in Equation 2 by SOMP, using $\mathbf{W}^{p,l}$ as weight matrix.

  $\mathbf{W}^{0,l} \leftarrow \mathbf{W}^{p,l}_{\omega^*(l):}$

  $\mathbf{W}^{p,l+1} \leftarrow (\mathbf{W}^{p,l+1})^T_{\omega^*(l):}{}^T$

Aggregate $\mathbf{W}^{0,l}$ for $l \in \{1, 2, \cdots, L\}$ to form Θ_0.

sions). In the literature a set of general methods for training arbitrarily small networks using an existing larger network and the training data are known as "knowledge distillation' [12, 29]. However, for the limited use case of this work we propose a faster, data-free, and simple yet reasonably effective method. Let $\mathbf{W}^{p,l}$ be the parameters of the pre-trained model at layer l with d rows. For convolutional layers, each row of $\mathbf{W}^{p,l}$ represents a vectorized filter kernel. The initialization procedure aims to identify a subset of $d^* (< d)$ rows of $\mathbf{W}^{p,l}$ to form $\mathbf{W}^{0,l}$ (the superscript 0 denotes initialized parameters for the thin model). We would like the selected rows that minimize the following objective:

$$\mathbf{A}^*, \omega^*(l) = \underset{\mathbf{A} \in R^{d \times d^*}, |\omega| = d^*}{\arg\min} \|\mathbf{W}^{p,l} - \mathbf{A}\mathbf{W}^{p,l}_{\omega:}\|_F, \quad (2)$$

where $\mathbf{W}^{p,l}_{\omega:}$ is a truncated weight matrix that only keeps the rows indexed by the set ω. This problem is NP-hard, however, there exist approaches based on convex relaxation [40] and greedy simultaneous orthogonal matching pursuit (SOMP) [41] which can produce approximate solutions. We use the greedy SOMP to find the approximate solution $\omega^*(l)$ which is then used to initialize the parameter matrix of the thin model as $\mathbf{W}^{0,l} \leftarrow \mathbf{W}^{p,l}_{\omega^*(l):}$. We run this procedure layer by layer, starting from the input layer. At layer l, after initializing $\mathbf{W}^{0,l}$, we replace $\mathbf{W}^{p,l+1}$ with a column-truncated version that only keeps the columns indexed by $\omega^*(l)$ to keep the input dimensions consistent. This initialization procedure is applicable for both convolutional and fully connected layers. See Algorithm 2.

3.2. Top-Down Layer-wise Model Widening

At the core of our training algorithm is a procedure that incrementally widens the current design in a layer-wise fashion. Let us introduce the concept of a "junction". A junction is a point at which the network splits into two or more independent sub-networks. We shall call such a sub-network a "branch". The leaves of each branch are outputs of a subset of tasks performed by this network. In person attribute classification each task is a *sigmoid* unit that produces a normalized confidence score on the existence of an attribute. We propose to widen the network only at these junctions. More formally, consider a junction at layer l with input $\mathbf{x}^l$ and d outputs $\{\mathbf{y}^l_i\}_{i=1}^d$. Note that each output is the input to one of the d top sub-networks. Similar to Equation 1 the within-layer computation is given as

$$\mathbf{y}^l_i = \sigma_l(\mathbf{P}_l(\mathbf{W}^l_i)\mathbf{x}^l) \quad \text{for} \quad i \in [d], \quad (3)$$

where $\mathbf{W}^l_i$ parameterizes the connection from input $\mathbf{x}^l$ to the i'th output $\mathbf{y}^l_i$ at layer l. The set $[d]$ is the indexing set $\{1, 2, \cdots, d\}$. A junction is widened by creating new outputs at the layer below. To widen layer l by a factor of c, we make layer $l-1$ a junction with $2 \le c \le d$ outputs. We use $\mathbf{y}^{l-1}_j$ to denote an output in layer $l-1$ (each is an input for layer l) and $\mathbf{W}^{l-1}_j$ to denote its parameter matrix. All of the newly-created parameter matrices have the same shape as $\mathbf{W}^{l-1}$ (the parameter matrix before widening). The single output $\mathbf{y}^{l-1} = \mathbf{x}^l$ is replaced by a set of outputs $\{\mathbf{y}^{l-1}_j\}_{j=1}^c$ where

$$\mathbf{y}^{l-1}_j = \sigma_{l-1}(\mathbf{P}_{l-1}(\mathbf{W}^{l-1}_j)\mathbf{x}^{l-1}) \quad \text{for} \quad j \in [c]. \quad (4)$$

Let $g^l : [d] \to [c]$ be a given grouping function at layer l. After widening, the within-layer computation at layer l is given as (cf. Equation 3)

$$\begin{aligned} \mathbf{y}^l_i &= \sigma_l(\mathbf{P}_l(\mathbf{W}^l_i)\mathbf{x}^l_{g^l(i)}) \\ &= \sigma_l\left(\mathbf{P}_l(\mathbf{W}^l_i)\sigma_{l-1}(\mathbf{P}_{l-1}(\mathbf{W}^{l-1}_{g^l(i)})\mathbf{x}^{l-1})\right) \end{aligned} \quad (5)$$

where the latter equality is a consequence of Equation 3. The widening operation sets the initial weight for $\mathbf{W}^{l-1}_j$ to be equal to the original weight of $\mathbf{W}^{l-1}$. It allows the widened network to preserve the functional form of the smaller network, enabling faster training.

To put the widening of one junction into the context of the multi-round progressive model widening procedure, consider a situation where there are T tasks. Before any widening, the output layer of the initial thin multi-task network has a junction with T outputs, each is the output of a sub-network (branch). It is also the only junction at initialization. The widening operation naturally starts from the output layer (denoted as layer l). It will cluster the T branches into t groups where $t \le T$. In this manner the widening operation creates t branches at layer $l-1$. The operation is performed recursively in a top-down manner towards the lower layers. Note that each branch will be associated with a sub-set of tasks. There is a 1-1 correspondence between tasks and branches at the output layer, but the granularity goes coarser at lower layers. An illustration of this procedure can be found in Figure 2.

Figure 2. Illustration of the widening procedure. *Left:* the active layer is at layer L, there is one junction with 7 branches at the top. *Middle:* The seven branches are clustered into three groups. Three branches are created at layer L, resulting in a junction at layer $L-1$. Layer $L-1$ is now the active layer. *Right:* Two branches are created at layer $L-1$, making layer $L-2$ now the active layer. At each branch creation, the filters at the newly created junction are initialized by direct copy from the old filter.

3.3. Task Grouping based on the Probability of Concurrently Simple or Difficult Examples

Ideally, dissimilar tasks are separated starting from a low layer, resulting in less sharing of features. For similar tasks the situation is the opposite. We observe that if an easy example for one task is typically a difficult example for another, intuitively a distinctive set of filters are required for each task to accurately model both in a single network. Thus we define the affinity between a pair of tasks as the probability of observing concurrently simple or difficult examples for the underlying pair of tasks from a random sample of the training data.

To make it mathematically concrete, we need to properly define the notion of a "difficult" and a "simple" example. Consider an arbitrary attribute classification task i. Denote the prediction of the task for example n as s_i^n, and the error margin as $m_i^n = |t_i^n - s_i^n|$, where t_i^n is the binary label for task i at sample n. Following the previous discussion, it seems natural to set a fixed threshold on m_i^n to decide whether example n is simple or difficult. However, we observe that this is problematic since as the training progresses most of the examples will become simple as the error rate decreases, rendering this measure of affinity useless. An adaptive but universal (across all tasks) threshold is also problematic as it creates a bias that makes intrinsically easier tasks less related to all the other tasks.

These observations lead us to the following approach. Instead of setting a fixed threshold, we estimate the average margin for each task, $E\{m_i\}$. We define the indicator variable for a difficult example for task i as $e_i^n = 1_{m_i^n \geq E\{m_i\}}$. For a pair of tasks i, j, we define their affinity as

$$\begin{aligned} A(i,j) &= P(e_i^n = 1, e_j^n = 1) + P(e_i^n = 0, e_j^n = 0) \\ &= E\{e_i^n e_j^n + (1 - e_i^n)(1 - e_j^n)\}. \end{aligned} \quad (6)$$

Both $E\{m_i\}$ and the expectation on Equation 6 can be estimated by their sample averages. Since these expectations are functions of the current neural network model, a naive implementation would require a large number of time consuming forward passes after every training iterations. As a much more efficient implementation, we alternatively collect the sample averages from each training mini-batches. The expectations are estimated by computing a weighted average of the within-batch sample averages. To make the estimation closer to the true expectations from the current model, an exponentially decaying weight is used.

The estimated task affinity is used directly for the clustering at the output layer. It is natural as branches at the output layer has a 1-1 map to the tasks. But at lower layers the mapping is one to many, as a branch can be associated with more than one tasks. In this case, affinity is computed to reflect groups of tasks. In particular, let k, l denote two branches at the current layer, where i_k and j_l denotes the i-th and j-th task associated with each branch respectively. The affinity of the two branches are defined by

$$\tilde{A}_b(k,l) = \operatorname*{mean}_{i_k} \left[\min_{j_l} \left[A(i_k, j_l) \right] \right] \quad (7)$$

$$\tilde{A}_b(l,k) = \operatorname*{mean}_{j_l} \left[\min_{i_k} \left[A(i_k, j_l) \right] \right] \quad (8)$$

The final affinity score is computed as $A_b(k,l) = (\tilde{A}_b(k,l) + \tilde{A}_b(l,k))/2$. Note that if branches and tasks form a 1-1 map (the situation at the output layer), this reduces to the definition in Equation 6. For branches with coarser task granularity, $A_b(k,l)$ measures the affinity between two branches by looking at the largest distance (smallest affinity) between their associated tasks.

3.4. Complexity-aware Width Selection

The number of branches to be created determines how much wider the network becomes after a widening operation. This number is determined by a loss function that balances complexity and the separation of dissimilar tasks to different branches. For each number of clusters $1 \leq d \leq c$, we perform spectral clustering to get a grouping function $g_d : [d] \rightarrow [c]$ that associates the newly created branches with the c old branches at one layer above. At layer l the

loss function is given by

$$L^l(g_d) = (d-1)L_0 2^{p_l} + \alpha L_s(g_d) \quad (9)$$

where $(d-1)L_0 2^{p_l}$ is a penalty term for creating branches at layer l, $L_s(g_d)$ is a penalty for separation. p_l is defined as the number of pooling layers above the layer l and L_0 is the unit cost for branch creation. The first term grows linearly with the number of branches, with a scalar that defines how expensive it is to create a branch at the current layer (which is heuristically set to double after every pooling layers). Note that in this formulation a larger α encourages the creation of more branches. We call α the branching factor. The network is widened by creating the number of branches that minimizes the loss function, or $g_d^{l\,*} = \arg\min_{g_d} L^l(g_d)$.

The separation term is a function of the branch affinity matrix A_b. For each $i \in [d]$, we have

$$L_s^i(g_d) = 1 - \underset{k \in g^{-1}(i)}{\text{mean}} \; \min_{l \in g^{-1}(i)} A_b(k,l) \quad , \quad (10)$$

and the separation cost is the average across each newly created branches

$$L_s(g_d) = \frac{1}{d} \sum_{i \in [d]} L_s^i(g_d). \quad (11)$$

Note Equation 10 measures the maximum distances (minimum affinity) between the tasks within the same group. It penalizes cases where very dissimilar tasks are included in the same branch.

4. Experiments

We test our approach on person attribute classification tasks. We use CelebA [24] dataset for facial attribute classification tasks and Deepfashion [23] for clothing category classification tasks. CelebA consists of images of celebrities labeled with 40 attribute classes. Most images also include the torso region in addition to the face. DeepFashion is richly labeled with 50 categories of clothes, such as "shorts", "jeans", "coats", etc. (the labels are mutually exclusive). Faces are often visible on these images.

4.1. Comparison with the State of the art

A successful multi-task architecture should reach a good balance of speed, model complexity and accuracy. In this section we discuss how well the proposed approach work in these aspects compared to recent state-of-the-art methods in person attribute classifications. To facilitate fair comparison in a controlled setting, we also list baselines trained in comparable conditions with the proposed branching method. For baselines we prepare vanilla VGG-16 models, low-rank models that factorizes all layers and thins models. We summarize our results on Table 1 and 2. See training and model details in Section 4.5.

Accuracy Following established protocols, we report attribute classification accuracy and top-10 recall rate on CelebA dataset, and top-3 and top-5 classification accuracy on DeepFashion dataset. The evaluations are performed on respective test partitions. We conclude that while being more compact/faster, models generated by the proposed branching method can closely match the state-of-the-art results, including the strong vanilla VGG-16 baselines prepared for this work.

Model Complexity/Speed At a comparable accuracy level, our models are significantly faster and more compact than the vanilla baseline (and closest competitors in the literature, MOON and FashionNet, both based on VGG-16). Compared to the thin baseline, a steady improvement of accuracy as model complexity increases is observed. This improvement starts to saturate as we make the thinness factor larger (creating more branches). On the other hand, making the initial network wider becomes more effective when at higher accuracy levels. On the facial attribute tasks on CelebA, the Branch-64-1.0 model is more accurate than the baseline from low-rank factorization while being slightly faster and slightly less compact. While we only explore small initial width to demonstrate the speed-up/model compression effects of the proposed method, future work should more thoroughly investigate different thinness setting and/or widening individual branches.

4.2. Understanding the Task Grouping

It is interesting to see if the automated procedure is learning an intuitive grouping of tasks, or rather a surprising grouping that contradicts our own intuition. To this end we visualize task groupings in the top layer of the Branch-32-2.0 model (for CelebA facial attribute tasks). Figure 4 shows the visualization. We observe an intuitive set of groupings. For instance, "5-o-clock Shadow", "Bushy Eyebrows" and "No Beard", which all describe some forms of facial hairs, are grouped. The cluster with "Heavy Makeup", "Pale Skin" and "Wearing Lipstick" is clearly related. Groupings at lower layers are also sensible. For instance, the group "Bags Under Eyes", "Big Nose" and "Young" are joined by "Attractive" and "Receding Hairline" at fc6, probably because they all describe age cues.

4.3. Cross-domain Training

We examine the method's ability to handle cross-domain tasks by training a network that jointly predict facial and clothing attributes. The model is trained on the union of the two training sets. Note that the CelebA dataset is not annotated with clothing labels, and the Deepfashion dataset is not annotated with facial attribute labels. To augment the annotations for both datasets, we use the predictions provided by the baseline VGG-16 models as soft training targets. We demonstrate that the joint model is comparable to

Method	Accuracy (%)	Top-10 Recall (%)	Test Speed (ms)	Parameters (millions)	Jointly?
LNet+ANet	87	N/A	+	+	No
Walk and Learn	88	N/A	+	+	No
MOON	90.94	N/A	≈ 33□	119.73	No
Our VGG-16 Baseline	91.44	73.55	33.2	134.41	No
Our Low-rank Baseline	90.88	69.82	16.0	4.52	No
Our Baseline-thin-32	89.96	65.95	5.1	0.22	No
Our Branch-32-1.0	90.74	69.95	9.6	1.49	No
Our Branch-32-2.0	90.90	71.08	15.7	2.09	No
Our Branch-64-1.0	91.26	72.03	15.2	4.99	No
Our Joint Branch-32-2.0	90.4	68.72	10.01	3.25	Yes
Our Joint Branch-64-2.0	91.02	71.38	16.28	10.53	Yes

Table 1. Comparison of accuracy, speed and compactness on CelebA test set. LNet+ANet and Walk and Learn results are cited from [43]. MOON results are cited from [31]. +: There is no reported number to cite. □: MOON uses the VGG16 architecture, thus its test time should be similar to our VGG-16 baseline. We use the convention Branch-x-c to represent a model from the proposed branching method, with initial width of x and branching factor of c. Similar, Baseline-thin-x can be seen as Branch-x-0 (no branching). The details of architectural changes on low-rank and thin models are shown in Section 3.1.

Method	Top-3 Accuracy (%)	Top-5 Accuracy (%)	Test Speed (ms)	Parameters (millions)	Jointly?
WTBI	43.73	66.26	+	+	No
DARN	59.48	79.58	+	+	No
FashionNet	82.58#	90.17#	≈ 34□	≈ 134□	No
Our VGG-16 Baseline	86.72	92.51	34.0	134.45	No
Our Low-rank Baseline	84.14	90.96	16.34	4.52	No
Our Joint Branch-32-2.0	79.91	88.09	10.01	3.25	Yes
Our Joint Branch-64-2.0	83.24	90.39	16.28	10.53	Yes

Table 2. Comparison of accuracy, speed and compactness on Deepfashion test set. WTBI and DARN results are cited from [23]. The experiments are reportedly performed in the same condition on the FashionNet method and tested on the DeepFashion test set. +: There is no reported number to cite. □: There is no reported number, but based on the adoption of VGG-16 network as base architecture they should be similar to those of our VGG-16 baseline. #: The results are from a network jointly trained for clothing landmark, clothing attribute and clothing categories predictions. We cite the reported results for clothing category [23]. See captions of Table 1 for naming conventions.

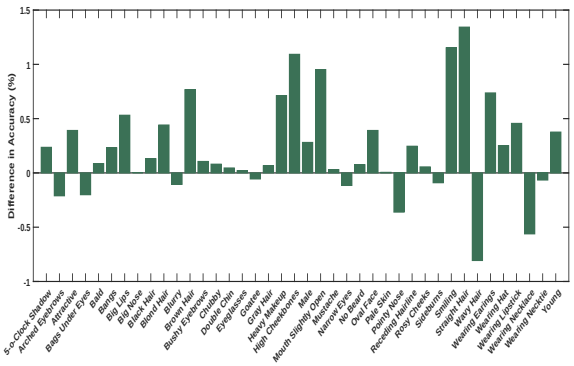

Figure 3. The reduction in accuracy when changing the task grouping to favor grouping of dissimilar tasks. A positive number suggests a reduction in accuracy when changing from original to the new grouping. This figure shows our automatic grouping strategy improves accuracy for most tasks.

the state-of-the-art on both facial and clothing tasks, while being a much more efficient combined model rather than two separate models. The comparison between the joint models with the baselines is shown in Table 1 and 2.

Method	Accuracy (%)	Top-10 Recall (%)
w/ pre-trained	-0.54	-2.47
w/o pre-trained	-0.65	-3.77

Table 3. Accuracy drop of Branch-32.2.0 compared to VGG-16 baseline, with and without initialization from pre-trained model.

4.4. Ablation Studies

Significance of grouping. We swaps the 20 tasks shown at the left side of Figure 4 with the 20 tasks at the right and retrain the model (using the "Branch-32-2.0" model illustrated in the figure). In this particular case this shuffling separates a large number of tasks originally in the same branch, creating a helpful scenario to diagnose our approach. Figure 3 summarizes the reduction in accuracy due to reshuffling. In this case, grouping tasks according to similarity improves accuracy for most tasks. We observe similar behaviors from multiple random reshuffling, however due to the randomness the overall gain are less clear cut at times.

Cause of accuracy drop. The drop in accuracy of the proposed method compared to VGG-16 baseline could to

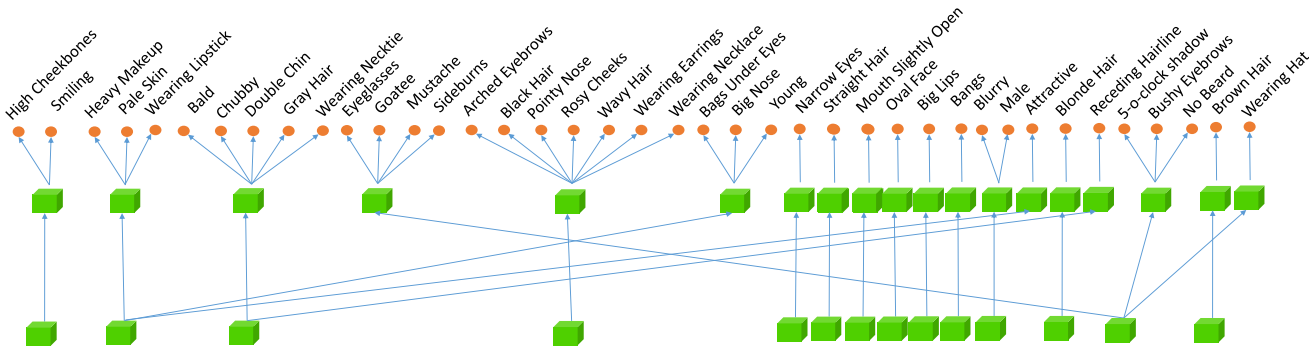

Figure 4. The actual task grouping in the Branch-32-2.0 model on CelebA. Upper: fc7 layer. Lower: fc6 layer. Other layers are omitted.

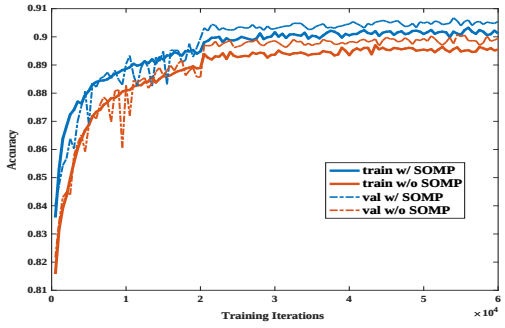

Figure 5. Comparison of training progress with and without SOMP initialization. The model using SOMP initialization clearly converges faster and better.

caused by sub-optimal use of the pretrained model (as the initial model is thinner), or the smaller capacity. We diagnose the issue by comparing the accuracy of Branch-32-2.0 and VGG-16 baseline with and without the initialization. If poor initialization is the main cause, removing the initialization should narrow the gap between the two models. This is not the case, as shown in Table 3. In this light, future work should probably prioritize better automated model design over more effective initialization.

Effects of SOMP initialization. We compare training with and without this initialization using the Baseline-thin-32 model on CelebA, under identical training conditions. The evolution of training and validation accuracies are shown in Figure 5. Clearly, the network initialized with SOMP initialization converges faster and better than the one without SOMP initialization.

4.5. Training Details

For all experiments, the attribute outputs are sigmoid units. Loss function is sigmoid cross-entropy loss weighting attributes uniformly. Each attribute is treated as a task. We use the original partitions in the two datasets. For CelebA images without face alignment (unlike reported MOON results) are used as their contents provide contexts useful for clothing. Mini-batch size is 32. The training iterations is 60000, the learning rate is initialized to 0.001 and decayed by a factor of 10 every 20000 iterations. The initial model branching phase updates the model after every 1000 iterations. These are changed to 100000, 40000 and 2000 respectively for joint models. The branching factor and the initial width of thin models used to train various models are summarized in Table 1 and Table 2. The error decay factor is set to 0.99. Experiments are performed on a single K40 GPU. With branching (factor=1.0) the training time on CelebA is reduced to 17 from 40 hours (VGG16 baseline). All training uses Batch Normalization (BN) [15]. BN layers are removed and the corresponding convolution layers are scaled and shifted accordingly for faster inference.

Baselines The vanilla VGG-16 model is initialized from imdb-wiki gender VGG-16 models by replacing the output layers [30]. The low-rank model factorizes all layers. It is initialized using truncated SVD [7] from the imdb-wiki model (the number of basis filters is 8-16-32-64-64-64-64-16). The thin models are initialized with SOMP (See Section 3.1 for more details).

5. Conclusion and Future Works

We have proposed a novel method for learning the structure of compact multi-task deep neural networks. Our method starts with a thin network model and expands it during training by means of a novel multi-round branching mechanism, which determines with whom each task shares features in each layer of the network, while penalizing for the complexity of the model. We demonstrated promising results of the proposed approach on the problem of person attribute classification.

The method itself is independent of the underlying tasks since it only require a notion of correlation between them. It thus can be applied to multi-task problem beyond person attribute classifications in future works. We also plan to adapt this method to other related problems, such as incremental learning and domain adaptation.

References

[1] K. Ahmed, M. H. Baig, and L. Torresani. Network of experts for large-scale image categorization. *ECCV*, 2016. 4322

[2] A. Almahairi, N. Ballas, T. Cooijmans, Y. Zheng, H. Larochelle, and A. Courville. Dynamic capacity networks. *ICML*, 2016. 4322

[3] H. Bilen and A. Vedaldi. Integrated perception with recurrent multi-task neural networks. In *NIPS*, 2016. 4322

[4] R. Caruana. Multi-task learning. *Machine Learning Journal*, 28:41–75, 1997. 4322

[5] Q. Chen, J. Huang, R. Feris, L. M. Brown, J. Dong, and S. Yan. Deep domain adaptation for describing people based on fine-grained clothing attributes. In *CVPR*, 2015. 4322

[6] Y. Cheng, F. Yu, R. Feris, S. Kumar, A. Choudhary, and S. F. Chang. An exploration of parameter redundancy in deep networks with circulant projections. In *ICCV*, 2015. 4322

[7] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*, 2014. 4328

[8] R. Feris, R. Bobbitt, L. Brown, and S. Pankanti. Attribute-based people search: Lessons learnt from a practical surveillance system. In *ICMR*, 2014. 4322

[9] B. Gong, B. Jou, F. Yu, and S.-F. Chang. Tamp: A library for compact deep neural networks with structured matrices. In *ACM Multimedia*, 2016. 4322

[10] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *ICLR*, 2016. 4322

[11] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 4323

[12] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. In *arXiv preprint arXiv:1503.02531*, 2015. 4322, 4324

[13] J. Huang, R. S. Feris, Q. Chen, and S. Yan. Cross-domain image retrieval with a dual attribute-aware ranking network. In *ICCV*, pages 1062–1070, 2015. 4321, 4322

[14] Y. Ioannou, D. Robertson, J. Shotton, R. Cipolla, and A. Criminisi. Training cnns with low-rank filters for efficient image classification. *ICLR*, 2016. 4322

[15] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 4328

[16] L. Jacob, J.-p. Vert, and F. R. Bach. Clustered multi-task learning: A convex formulation. In *NIPS*, 2009. 4322

[17] B. Jou and S. F. Chang. Deep cross residual learning for multi-task visual recognition. In *ACM Multimedia*, 2016. 4321, 4322

[18] Z. Kang, K. Grauman, and F. Sha. Learning with whom to share in multi-task feature learning. In *ICML*, 2011. 4322

[19] I. Kokkinos. Ubernet: Training a universal cnn for low-, mid-, and high- level vision using diverse datasets and limited memory. In *arXiv preprint arXiv:1609.02132*, 2016. 4322

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 4323

[21] A. Kumar and H. Daume III. Learning task grouping and overlap in multi-task. In *ICML*, 2012. 4322

[22] L. Liu, J. Xing, S. Liu, H. Xu, X. Zhou, and S. Yan. Wow! you are so beautiful today! *ACM Transactions on Multimedia Computing, Communications, and Applications*, 11(1s):20, 2014. 4322

[23] Z. Liu, P. Luo, S. Qiu, X. Wang, and X. Tang. Deepfashion: Powering robust clothes recognition and retrieval with rich annotations. In *CVPR*, 2016. 4322, 4326, 4327

[24] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *ICCV*, 2015. 4322, 4326

[25] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert. Cross-stitch networks for multi-task learning. In *CVPR*, 2016. 4321, 4322, 4323

[26] A. Passos, P. Rai, J. Wainer, and H. Daume III. Flexible modeling of latent task structures in multitask learning. *arXiv preprint arXiv:1206.6486*, 2012. 4322

[27] A. Polyak and L. Wolf. Channel-level acceleration of deep face representations. *IEEE Access*, 3:2163–2175, 2015. 4322

[28] R. Ranjan, V. Patel, and R. Chellappa. Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. In *arXiv preprint arXiv:1603.01249*, 2016. 4321, 4322

[29] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014. 4322, 4324

[30] R. Rothe, R. Timofte, and L. V. Gool. Deep expectation of real and apparent age from a single image without facial landmarks. *International Journal of Computer Vision (IJCV)*, 2016. 4328

[31] E. Rudd, M. Günther, and T. Boult. Moon: A mixed objective optimization network for the recognition of facial attributes. *ECCV*, 2016. 4322, 4327

[32] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *ICASSP*, 2013. 4322

[33] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *ICLR2015*, 2014. 4323

[34] V. Sindhwani, T. Sainath, and S. Kumar. Structured transforms for small-footprint deep learning. In *NIPS*, 2015. 4322

[35] C. Su, S. Zhang, J. Xing, W. Gao, and Q. Tian. Deep attributes driven multi-camera person re-identification. *ECCV*, 2016. 4322

[36] P. Sudowe, H. Spitzer, and B. Leibe. Person attribute recognition with a jointly-trained holistic cnn model. In *ICCV ChaLearn Looking at People Workshop*, 2015. 4322

[37] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015. 4323

[38] C. Tai, T. Xiao, X. Wang, et al. Convolutional neural networks with low-rank regularization. *ICLR*, 2016. 4322

[39] S. Thrun and L. Pratt. Learning to learn. *Kluwer Academic Publishers*, 1998. 4322

[40] J. A. Tropp. Algorithms for simultaneous sparse approximation. part ii: Convex relaxation. *Signal Processing*, 86(3):589–602, 2006. 4324

[41] J. A. Tropp, A. C. Gilbert, and M. J. Strauss. Algorithms for simultaneous sparse approximation. part i: Greedy pursuit. *Signal Processing*, 86(3):572–588, 2006. 4322, 4324

[42] D. A. Vaquero, R. S. Feris, D. Tran, L. Brown, A. Hampapur, and M. Turk. Attribute-based people search in surveillance environments. In *WACV*, 2009. 4322

[43] J. Wang, Y. Cheng, and R. S. Feris. Walk and learn: Facial attribute representation learning from egocentric video and contextual data. *CVPR*, 2016. 4322, 4327

[44] Y. Xue, X. Liao, L. Carin, and B. Krishnapuram. Multi-task learning for classification with dirichlet process priors. *Journal of Machine Learning Research*, 8(Jan):35–63, 2007. 4322

[45] Z. Yan, H. Zhang, R. Piramuthu, V. Jagadeesh, D. DeCoste, W. Di, and Y. Yu. Hd-cnn: Hierarchical deep convolutional neural network for large scale visual recognition. In *ICCV*, 2015. 4322

[46] S. Yang, P. Luo, C.-C. Loy, and X. Tang. From facial parts responses to face detection: A deep learning approach. In *ICCV*, 2015. 4322

[47] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014. 4323

[48] N. Zhang, M. Paluri, M. Ranzato, T. Darrell, and L. Bourdev. Panda: Pose aligned networks for deep attribute modeling. In *CVPR*, 2014. 4322

[49] Z. Zhang, P. Luo, C. C. Loy, and X. Tang. Learning deep representation for face alignment with auxiliary attributes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(5), 2016. 4322

[50] J. Zhou, J. Chen, and J. Ye. Clustered multi-task learning via alternating structure optimization. In *NIPS*, 2011. 4322