Learning to Branch for Multi-Task Learning

Pengsheng Guo [1] Chen-Yu Lee [1] Daniel Ulbricht [1]

Abstract

Training multiple tasks jointly in one deep network yields reduced latency during inference and better performance over the single-task counterpart by sharing certain layers of a network. However, over-sharing a network could erroneously enforce over-generalization, causing negative knowledge transfer across tasks. Prior works rely on human intuition or pre-computed task relatedness scores for ad hoc branching structures. They provide sub-optimal end results and often require huge efforts for the trial-and-error process.

In this work, we present an automated multi-task learning algorithm that learns where to share or branch within a network, designing an effective network topology that is directly optimized for multiple objectives across tasks. Specifically, we propose a novel tree-structured design space that casts a tree branching operation as a gumbel-softmax sampling procedure. This enables differentiable network splitting that is end-to-end trainable. We validate the proposed method on controlled synthetic data, CelebA, and Taskonomy.

1. Introduction

Multi-task learning (Caruana, 1997) has experienced rapid growth in recent years. Because of the breakthroughs in the performance of individually trained single-task neural networks, researchers have shifted their attention towards training networks that are able to solve multiple tasks at the same time. One clear benefit of such a system is reduced latency where one network can produce multiple predictions in one forward propagation. This is particularly critical for portable devices that have limited computational budget. Moreover, when training with various supervisory signals, it induces inductive bias (Mitchell, 1980) where a network prefers some hypotheses over other hypotheses. From the point of view of any single task, the other tasks serve as regularizers in a sense that the network is asked to form representations that explain well for more than needed for solving one task, potentially improving generalization.

Contrary to the conventional single-task paradigm, training multiple tasks simultaneously in one network often encounters many challenges: some tasks are easier to train than others, some tasks have noisier ground truth labels than others, and some tasks are equipped with loss functions that have drastically different scales than others such as L1 vs cross-entropy. Most of the work done in this field has focused on establishing some sort of parameter sharing mechanism by either sharing the whole network across all tasks or by assigning each task an individual set of parameters with crosstalk connections between tasks (Ruder, 2017). However, it is prohibitively expensive to design an optimal parameter sharing schema based on human intuition. Another line of work has tried to balance the importance of different tasks by manipulating relative weighting between each task's loss (Kendall et al., 2018; Guo et al., 2018; Chen et al., 2018). But weight balancing alone also limits the potential performance gain under a fixed pre-defined network architecture.

There are many ways a network can invest its capacity for different tasks, and the design choice has a fundamental impact on its learning dynamics and final performance. Note that an exhaustive search of an optimal parameter sharing schema has combinatorial complexity as the number of tasks grows. Prior literature has presented evidence that multi-task learning in back-propagation networks discovers task relatedness without the need of supervisory signals, and has presented results with k-nearest neighbor and kernel regression models (Caruana, 1997). In this work we ask the following question: *is it possible to automatically search a network topology based on the back-propagation signals computed from the multi-task objective?*

Typical neural networks learn the hierarchical nature of the feature representations. Specifically for computer vision applications, convolutional neural networks tend to learn more general feature representations in earlier layers such as edges, corners, and conjunctions (Zeiler & Fergus, 2014). We therefore expect a network at least shares the first few

[1]Apple. Correspondence to: Pengsheng Guo <pengsheng_guo@apple.com>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 108, 2020. Copyright 2020 by the author(s).

layers across tasks. A key challenge towards answering the question is then deciding what layers should be shared across tasks and what layers should be untied. Over-sharing a network could erroneously enforce over-generalization, causing negative knowledge transfer across tasks. In this work, we propose a tree-structured network design space that can automatically learn how to branch a network such that the overall multi-task loss is minimized. The branching operation is executed by sampling from a categorical latent variable formed by gumbel-softmax distribution (Jang et al., 2017). This data-driven network structure searching approach does not require prior knowledge of the relationship between tasks nor human intuition on what layers capture task-specific features and should be split.

2. Related Work

Thanks to the genericness and the transferability of a number of off-the-shelf neural networks (Simonyan & Zisserman, 2015; Szegedy et al., 2015; He et al., 2016) pre-trained on a large collection of samples, most of the prior works in the domain of multi-task learning are based on these popular backbone architectures and can be commonly categorized into either soft parameter sharing or hard parameter sharing (Ruder, 2017).

In the soft sharing setting, each task has its own set of backbone parameters with some sort of regularization mechanisms to enforce the distance between weights of the model to be close. Neural Network Parser (Duong et al., 2015) uses two backbones, one for source language and one for target language, to perform multi-task learning. An extra set of weights are used for cross-lingual knowledge sharing by connecting activations between the source and target language model. Cross-Stitch Networks (Misra et al., 2016) utilize an extra set of shared units to combine the activations between backbones from multiple tasks and learn the strength of information flow between tasks from data. However, the total number of parameters in a soft parameter sharing system grows linearly with the number of tasks in general. Such approaches may encounter over-fitting due to lack of sufficient training samples to support the parameter space of multiple full-size backbones across all modalities or require higher computational cost during inference.

In the hard sharing setting, all tasks share the same set of backbone parameters, or at least share part of the backbone with branches toward the outputs. Deep Relationship Networks (Long & Wang, 2015) share the first five convolutional layers of AlexNet (Krizhevsky et al., 2012) among the tasks and use task-specific fully-connected layers tailored to fit different tasks. Fully-Adaptive Feature Sharing method (Lu et al., 2017) starts with a thin network and expands it layer-by-layer based on the difficulty of the training set in a greedy fashion. UberNet (Kokkinos, 2017) jointly solves seven labelling tasks by sharing an image pyramid architecture with tied weights. Meta Multi-Task Learning (Ruder et al., 2019) uses a shared input layer and two task-specific output layers. Nonetheless, it is still unclear how to effectively decide what weights to share given a network with a set of tasks in interest.

Instead of choosing between soft sharing or hard sharing approach, a new effort in tackling the multi-task learning problem is to consider the dynamics between different losses across tasks. Uncertainty-based weighting approach (Kendall et al., 2018) weighs multiple loss functions by utilizing the homoscedastic uncertainty of each task. GradNorm (Chen et al., 2018) manipulates the magnitude of gradients from different loss functions to balance the learning speed between tasks. Task Prioritization (Guo et al., 2018) emphasizes more difficult tasks by adjusting the mixing weight of each task's loss objective. Multi-Objective Optimization approach (Sener & Koltun, 2018) casts the multi-task learning problem as finding a set of solutions that lies on the Pareto optimal boundary. These methods can automatically tune the weightings between tasks and are especially effective when dealing with loss functions with different scales such as L1, L2, cross-entropy, etc. Yet, they use pre-defined network structures that might lead to sub-optimal solutions when the network topologies remain static.

The general focus in Neural Architecture Search (NAS) literature (Zoph & Le, 2017; Zoph et al., 2018; Liu et al., 2018; Wong et al., 2018; Liu et al., 2019a; Shaw et al., 2019; Pasunuru & Bansal, 2019) is on finding a repetitive cell or a global structure that is optimized over a single classification loss with a few exceptions that include memory or power constraint. To better utilize the parameters of a network for multiple tasks, recently some works present methods to dynamically distribute the network capacity based on the complexities of the tasks and relatedness between the tasks. Soft Layer Ordering (Meyerson & Miikkulainen, 2018) learns to generate a task-specific scaling tensor to manipulate the magnitude of feature activations at different layers. Evolutionary Architecture Search (Liang et al., 2018) improves upon the Soft Layer Ordering by a synergetic approach of evolving custom shared routing units. Soft attention used in (Liu et al., 2019b) allows learning of task-specific feature-level weighting for each task. AdaShare (Sun et al., 2019) learns the sharing pattern through a task-specific policy that selectively chooses which layers to execute for a given task in a multi-task setting. The authors in (Standley et al., 2019) propose to discover an optimal network splitting structure for different tasks by approximating the enumerative search process so that the test performances are maximized given a fixed testing resource budget. Branched Multi-task Networks (Vandenhende et al., 2019) pre-compute a collection of task relatedness scores based on the usefulness of a set of features of one task for the other. Gumbel-Matrix Routing (Maziarz et al., 2019) stacks a fixed set of operations in each layer and learns the connectivities between layers. These works do not rely on fixed network connectivities and start to explore the potential of more dynamic network wirings tailored to multiple tasks. But on the other hand, additional computation is often required for obtaining task relatedness scores in order to perform task grouping or splitting.

In this paper we propose a new end-to-end trainable algorithm that can automatically design a hard parameter sharing multi-task network, sharing and splitting network branches based on the update gradients back-propagated from the overall losses across all tasks. The proposed method bypasses the need of pre-computed task relatedness scores and directly optimizes over the end outputs, saving tedious computation and producing effective network topologies.

3. Method

We introduce the formal problem definition in Section 3.1. We present the proposed network design space in Section 3.2 and the differentiable branching operation formulation in Section 3.3. Finally we show how the final network architecture is selected after training in Section 3.4.

3.1. Formulation Setup

Given a set of N tasks $\mathcal{T} = \{t_1, t_2, ..., t_N\}$, the goal of the proposed method is to learn a tree-structured (Lee et al., 2016) network architecture Ω and the weight values ω of the network that minimize the overall loss $\mathcal{L}_{\text{total}}$ across all tasks,

$$\begin{aligned} \omega^*, \Omega^* &= \arg\min_{\omega,\Omega} \mathcal{L}_{\text{total}}(\omega, \Omega) \\ &= \arg\min_{\omega,\Omega} \sum_k \alpha_k \mathcal{L}_k(\omega, \Omega) \end{aligned} \tag{1}$$

where $\mathcal{L}_k$ is the loss for task k and α_k is the task-specific weighting. The tree structure in the network is realized by branching operations at certain layers. Each branching layer can have an arbitrary number of child (next) layers up to the computational budget available.

During training, we first sample a network configuration Ω from the design space distribution and then perform forward propagation to compute the overall loss value $\mathcal{L}_{\text{total}}$. We then obtain corresponding gradients to update both the design space distribution and the weight matrices ω in the network in backward fashion. We iterate through the process until the overall validation loss converges and then we sample our final network configuration using the converged design space distribution.

3.2. Network Topological Space

The key ingredient for effective and efficient network configuration sampling is our proposed differentiable tree-structured network topology. The topological space is represented as a Directed Acyclic Graph (DAG) where the nodes represent computational operations and the edges denote data flows. Figure 1 illustrates a certain block of a DAG which contains parent nodes i for $i \in \{1, ..., I\}$ and child nodes j for $j \in \{1, ..., J\}$. The nodes can perform any common operations of choice such as convolution or pooling. The input to a certain node is denoted as x and the output is denoted as y.

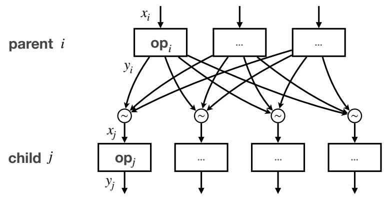


Figure 1. Illustration of the proposed branching block. Each child node j is equipped with a categorical distribution so it can sample a parent node to receive input data after the training.

Specifically, we construct multiple parent nodes and child nodes for each block and allow a child node to sample a path from all the paths between it and all its parent nodes. The selected connectivities therefore define the tree structure by such sampling (branching) procedure. We formulate the branching operation at layer l as:

$$x_j^{l+1} = \mathbb{E}_{d_j \sim p_{\theta_j}} [\, d_j \cdot Y^l \,] \tag{2}$$

where $Y^l = [\, y_1^l, ..., y_I^l \,]$ concatenates outputs from all parent nodes at layer l, and d_j is an indicator vector sampled from a certain distribution p_{θ_j}. The indicator d_j is a one-hot vector. Hence the dot product in Eq 2 essentially assigns one of the parent nodes to each child node j. In other words, each parent node at layer l propagates its output activations as input x_j^{l+1} to one or more child nodes j based on the sampling distributions. The sampling distribution is parameterized by θ_j. The proposed topological space degenerates into a conventional single-path (convolutional) neural network if each block only contains one parent node and one child node.

We update the parameter θ_j of the sampling distribution p_{θ_j} using the chain rule with respect to the final loss,

$$\begin{aligned} \frac{\partial \mathcal{L}_{\text{total}}}{\partial \theta_j} &= \frac{\partial \mathcal{L}_{\text{total}}}{\partial x_j^{l+1}} \frac{\partial x_j^{l+1}}{\partial \theta_j} \\ &= \frac{\partial \mathcal{L}_{\text{total}}}{\partial x_j^{l+1}} \frac{\partial}{\partial \theta_j} \mathbb{E}_{d_j \sim p_{\theta_j}} [\, d_j \cdot Y^l \,] \end{aligned} \tag{3}$$

the backward pass then adjusts the sampling distribution p_{θ_j} to make it more likely to generate network configurations Ω toward the direction of minimizing the overall loss $\mathcal{L}_{\text{total}}$.

The branching blocks in Figure 1 can be stacked to form a deeper tree-structured neural network (illustrated in Figure 2(d)) and the number of parent nodes and the number

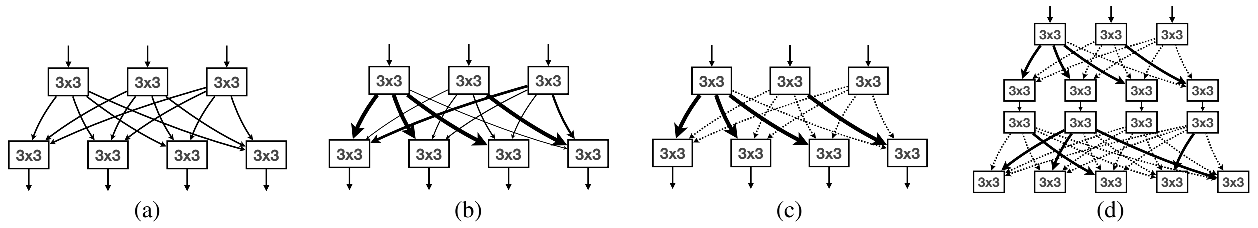


Figure 2. Illustrations of the proposed learning to branch pipeline. (a) We initialize the sampling probability with a uniform distribution so each parent node has an equal chance to send its activation values to a child node. (b) The computed update gradients then increase the probability of sampling certain paths that are more likely to reduce the overall loss. (c) Once the overall validation loss converges, each child node selects one parent node with the highest sampling probability while removing unselected paths and parent nodes. (d) We can construct a deeper tree-structured multi-task neural network by stacking such branching blocks.

of child nodes can be adjusted based on the desired model capacity. Different from the greedy layer-wise optimization approach in GNAS (Huang et al., 2018), our proposed tree-structured network topology is end-to-end trainable – the network architecture Ω and the weight matrices ω of the network are jointly optimized during training.

3.3. Differentiable Branching Operation

To sample a categorical value from the continuous sampling distribution, we utilize the gumbel-softmax estimator trick (Jang et al., 2017; Shazeer et al., 2017; Rosenbaum et al., 2018; Veit & Belongie, 2018; Xie et al., 2019) to enable the differentiability for the branching operation; During the feedforward pass, we sample a parent node by a discrete index value based on a certain probability distribution for each child node. During the backward pass, we update the probability distribution by replacing the discrete samples with gumbel-softmax samples.

For every two layers in a branching block (shown in Figure 1), we construct a matrix $M \in \mathbb{R}^{I\times J}$ to represent the connectivity from parent nodes i to child nodes j. Each entry $\theta_{i,j}$ in such a matrix M stores the probability value that represents how likely the parent node i would be sampled to connect with the child node j. During every forward propagation, each child node j makes a discrete decision drawn from a categorical distribution based on the distribution:

$$d_j = \text{one\_hot}\{\arg\max_i(\log\theta_{i,j} + \epsilon_i)\} \tag{4}$$

Again $d_j \in \mathbb{R}^I$ is a one-hot vector with dimension the same as the number of parent nodes I at the current level. $\epsilon \in \mathbb{R}^I$ is a vector with i.i.d samples draw from gumbel distribution $(0, 1)$ to add a small amount of noise to avoid the $\arg\max$ operation always selecting the element with the highest probability value.

To enable differentiability of the discrete sampling function, we use the gumbel-softmax trick (Jang et al., 2017) to relax d_j during backward propagation as

$$\tilde{d_j} = \frac{\exp((\log\theta_{i,j} + \epsilon_i)/\tau)}{\sum_k \exp((\log\theta_{k,j} + \epsilon_k)/\tau)} \tag{5}$$

with i equal to the sampled index value of parent node during forward pass. The discrete categorical sampling function is approximated by a softmax operation over the parent nodes, and the parameter τ is the temperature that controls how sharp the distribution is after the approximation.

We can now utilize the reparameterization trick for random sample d_j and rewrite the Eq 3 as

$$\begin{aligned}\frac{\partial\mathcal{L}_{\text{total}}}{\partial\theta_j} &= \frac{\partial\mathcal{L}_{\text{total}}}{\partial x_j^{l+1}}\frac{\partial}{\partial\theta_j}\mathbb{E}_\epsilon[\,\tilde{d_j}\cdot Y^l\,] \\ &= \frac{\partial\mathcal{L}_{\text{total}}}{\partial x_j^{l+1}}\mathbb{E}_\epsilon[\,\frac{\partial\tilde{d_j}}{\partial\theta_i}\,]Y^l\end{aligned} \tag{6}$$

At this stage, the branching probabilities are fully differentiable with respect to the training loss and can readily be inserted to a neural network and stacked to construct a tree-structured neural network. We decay the temperature τ gradually during training so the network can explore freely in the early stage and exploit the converged topology distribution in the later stage.

3.4. Final Architecture Selection

During the training stage, the network topology distribution and the weight matrices of the network are jointly optimized over the loss $\mathcal{L}_{\text{total}}$ across all tasks. Once the validation loss converges, we simply select the final network configuration using the same categorical distribution but without the noise ϵ for every block in the network,

$$d_j = \text{one\_hot}\{\arg\max_i(\log\theta_{i,j})\} \tag{7}$$

We then re-train the final network architecture from scratch to obtain the final performance. The same procedure has also been shown effective in previous literature (Pham et al., 2018; Sciuto et al., 2019) where such weight sharing network search schema demonstrates high correlation between the intermediate network performance during search phase and the final performance obtained by re-train the network from scratch.

Figure 2 illustrates our overall training process of a certain branching block in a DAG. We initialize the sampling

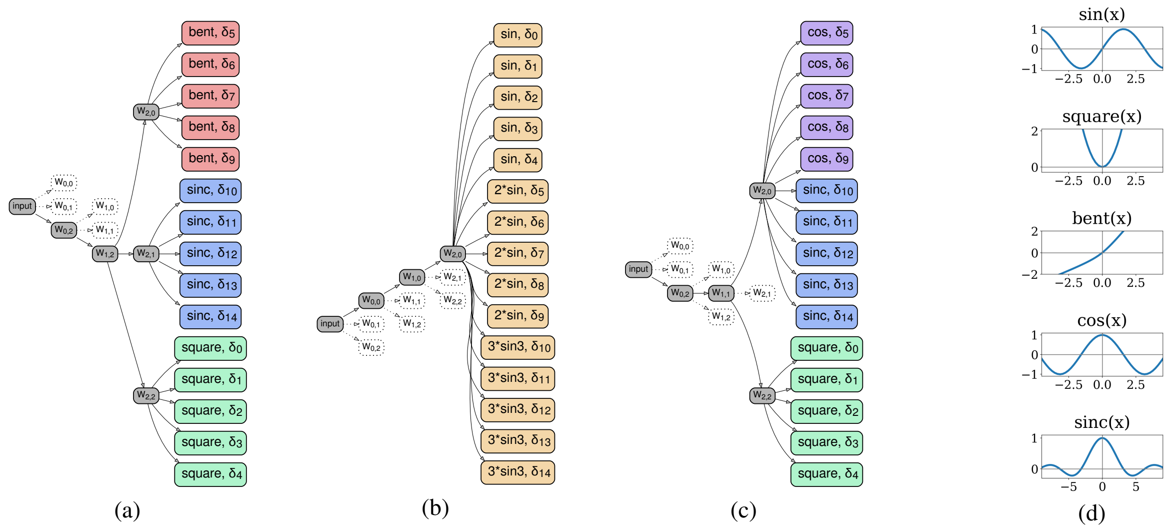


Figure 3. Learned network architectures by our method in three different experimental settings – each setting has 15 tasks generated by 3 different activation functions. Setting (a) contains activations bent, square, and sinc. The learned network group tasks with the same activation together in 3 distinct clusters. Setting (b) contains the same activation sin but with different scale multipliers 1, 2, and 3. All tasks are grouped together and share all intermediate layers. Setting (c) contains activations cos, sinc, and square. As illustrated in (d) that cos and sinc share similar behavior, tasks with these two similar activations are grouped while the task with square activation branches out earlier in the intermediate layer.

probability p_{θ_j} with a uniform distribution so each parent node has an equal chance to send its activation values to a child node j as shown in Figure 2(a). The computed update gradients then increase the probability of sampling certain paths that are more likely to reduce the overall loss as shown in Figure 2(b). Once the overall validation loss converges, each child node selects one parent node with the highest sampling probability while removing unselected paths and parent nodes as shown in Figure 2(c). We can construct a deeper tree-structured multi-task neural network by stacking such branching blocks as shown in Figure 2(d) as long as we ensure the number of child nodes matches the number of tasks being trained. This process implicitly groups the relevant tasks together by sharing necessary layers. It does not require prior knowledge of the relatedness of the tasks and avoids exhausting trial-and-error searching process by hand.

4. Experiments

In principle, our method can be applied to any domains and does not require any prior knowledge of the tasks. We demonstrate the effectiveness of the proposed method on synthetic data for regression tasks, CelebA (Liu et al., 2015) dataset for classification tasks, and Taskonomy dataset (Zamir et al., 2018) for dense prediction that includes both regression and classification tasks.

4.1. Controlled Synthetic Data

We first validate the proposed concept using synthetic data with controllable task relatedness. The relatedness is realized by different activation functions. Inspired by (Chen et al., 2018), we construct the regression tasks by the formulation

$$T_{r,s} = \text{activation}_r[((B + \delta_s)\, Z)/\varphi] \tag{8}$$

where $r \in \{1, 2, ..., 5\}$ corresponds to a range of five different element-wise activation functions {sin, square, bent, cos, sinc}, δ_s denotes task-specific random noise matrices for $s \in \{1, 2, ..., S\}$. The input of the formulation $Z \in \mathbb{R}^{200}$ is a 200-dimensional vector and the output $T \in \mathbb{R}^{100}$ is a 100-dimensional vector. $B \in \mathbb{R}^{100\times200}$ and $\delta_s \in \mathbb{R}^{100\times200}$ are constant matrices randomly sampled from $\mathcal{N}(0, 10)$ and $\mathcal{N}(0, 2)$, respectively. $\varphi \in \mathbb{R}$ is a normalization term that has the value of size of the input dimension. We control the relatedness of the tasks by the activation used. Tasks constructed by the same activation function should be more related as they only differ in a small amount of random noise δ_s. On the other hand, tasks with different activation functions should be more unrelated.

We use four proposed branching blocks to construct our tree-structured network for the multi-task learning setup. Each block has three child nodes with fully-connected layers as the choice of operation. Each fully-connected layer contains 100 neurons. We use simple bias terms for task-specific layers as shown in Figure 3. L2 loss is used as the training objective and the loss is optimized by the Adam solver with mini-batch size of 100. Learning rate is set to 10^{-3} for weight matrices and 10^{-7} for branching probability throughout the training. Temperature is set to 50 and decayed by the square root of the number of iterations. The networks are trained for 500 epochs with 50 epochs for warmup. We do not update the branching probability during warmup to ensure all weight matrices receive equal amounts of update gradients.

We perform experiments in three different settings – each setting has equal weighted 15 tasks generated by 3 different activation functions. In the first setting (Figure 3(a)), we use activations bent, square, and sinc. The final learned network architecture clearly shows a tree structure with similar tasks (same activation) are grouped in the same leaf branch and dissimilar tasks (different activations) do not share the same leaf branch. In the second setting (Figure 3(b)), we use the same activation sin but with different scale multipliers 1, 2, and 3. All tasks are grouped together and share all intermediate layers since they only differ in different scaling. In the third setting (Figure 3(c)), we use activations cos, sinc, and square. As illustrated in Figure 3(d) that cos and sinc activations share very similar active regions and scales, we can see that tasks with these two similar activations are grouped while the task with square activation branches out earlier in the intermediate layer. From this experiment, we validate our intuition that the proposed branching structure indeed captures the underlying task relatedness and is able to group related tasks through back-propagation updates.

4.2. CelebA

Next, we evaluate the proposed method on real-world image classification tasks. We use the CelebA dataset (Liu et al., 2015), which contains over 200K face images and each image contains 40 binary attribute annotations. Each annotation is regarded as a classification task and we adopt 40 cross-entropy losses with equal weightings for all 40 tasks. The training, validation, and test sets contain 160K, 20K, and 20K images. This benchmark is especially useful to examine whether automatically learned task grouping is more effective than manual task grouping by human intuition or pre-computed task relatedness.

Implementation details. For a fair comparison, we utilize the same overall network structures and operations from (Lu et al., 2017; Huang et al., 2018) in our branching blocks but allow the network to learn the branching decisions. Specifically, we construct (a) LearnToBranch-VGG model based on the Branch-VGG model in (Lu et al., 2017) where the backbone is a truncated VGG19 (Simonyan & Zisserman, 2015) with the number of channels reduced to 32 for convolutional layers and 64 for the fully-connected layers, and (b) LearnToBranch-Deep-Wide based on the GNAS-Deep-Wide model in (Huang et al., 2018), which is a customized architecture with 5 consecutive convolutional layers and 2 fully-connected layers. For model (a) we allow our network to branch at the end of each resolution stage (last conv layer at each resolution) with the number of child nodes set to $\{3, 3, 5, 5, 10, 20, 30, 40\}$, and for model (b) we allow our network to branch at the end of each convolutional layer with number of child nodes set to $\{2, 4, 8, 16, 40\}$. The number of child nodes are chosen so that the overall computational complexity of models (a) and (b) are similar to their counterparts.

Table 1. Results of multi-task learning on CelebA Dataset.

Method	Acc (%)	Params (M)
LNet+ANet (Wang et al., 2016)	87	-
Walk and Learn (Wang et al., 2016)	88	-
Moon (Rudd et al., 2016)	90.94	119.73
Indep Group (Hand & Chellappa, 2017)	91.06	-
MCNN-AUX (Hand & Chellappa, 2017)	91.29	-
VGG-16 Baseline (Lu et al., 2017)	91.44	134.41
Branch-VGG (Lu et al., 2017)	90.79	2.09
LearnToBranch-VGG (ours)	**91.55**	**1.94**
GNAS-Deep-Wide (Huang et al., 2018)	91.36	6.41
LearnToBranch-Deep-Wide (ours)	**91.62**	**6.33**

We use the Adam optimizers with mini-batch size 64 to update both the weight matrices and the branching probabilities in our networks. Temperature is set to 10 and decayed by the number of epochs. We warmup the training for 2 epochs without updating the branching probabilities to ensure all weight matrices receive equal amounts of update gradients initially. Weight decay is set to 10^{-4} for all experiments. We perform grid search for learning rates in $(10^{-6}, 10^{-5}, 10^{-4})$ for the weights and in $(1, 10, 100)$ for branching distributions. After sampling the final architecture, we train the network from scratch with grid search for global learning rate in $(0.02, 0.03, 0.04, 0.05)$. The input data is normalized $[-1, 1]$ and augmented by random flipping. Please refer to Appendix for more details.

Our method leverages the effectiveness of gumbel-softmax so that every child node samples a single discrete action during the forward pass. Therefore, our network topological space is well maintained – the tree does not grow exponentially with the number of tasks. As the result, it takes 10 hours to search the architecture and 11 hours to obtain the optimal weights for model (a) LearnToBranch-VGG, and it takes 4 hours to search the architecture and 10 hours to obtain the optimal weights for model (b) LearnToBranch-Deep-Wide on a single 16GB Tesla GPU.

Results. Table 1 shows the performance comparison on the CelebA test set. The visualizations of the learned network architectures are provided in Appendix. We can clearly see that both models (a) LearnToBranch-VGG and (b) LearnToBranch-Deep-Wide outperform their counterpart baselines presented in (Lu et al., 2017; Huang et al., 2018) under the similar network capacity. In fact, both our models (a) and (b) have less total number of parameters than their baselines. Note that our models only differ in the branching operation while maintaining other configurations such as kernel size and the number of channels. This

Table 2. Results of multi-task learning on Taskonomy test set. Our method outperforms the direct comparable method AdaShare (Sun et al., 2019) and other baselines as well. Besides having fewer parameters and better performance, our method has a clear advantage of being the first end-to-end trainable tree-structured multi-task network that does not require human intuition or pre-computed task relatedness.

Method	Params (M)	Segmentation ↓	Normal ↑	Depth ↓	Keypoint ↓	Edge ↓
Single-Task (Sun et al., 2019)	124	0.575	0.707	0.022	0.197	0.212
Multi-Task (Sun et al., 2019)	41	0.587	0.702	0.024	0.194	0.201
Cross-Stitch (Misra et al., 2016)	124	0.560	0.684	0.022	0.202	0.219
Sluice (Ruder et al., 2017)	124	0.610	0.702	0.023	0.192	0.198
NDDR-CNN (Gao et al., 2019)	133	0.539	0.705	0.024	0.194	0.206
MTAN (Liu et al., 2019b)	114	0.637	0.702	0.023	0.193	0.203
AdaShare (Sun et al., 2019)	41	0.566	0.707	0.025	0.192	0.193
LearnToBranch (ours)	51	**0.462**	**0.709**	**0.018**	**0.122**	**0.136**

demonstrates the effectiveness of the proposed end-to-end trainable branching mechanism. We note that the ResNet-18 (MGDA-UB) model in (Sener & Koltun, 2018) achieves 91.75% accuracy on this specific task. However their network has more than 11 million parameters, which is roughly double the size of our model (b) with comparable performance. Also their focus is on reformulating the multi-task learning problem as multi-objective optimization. We propose to further study the possibility of combining the two techniques in future investigation.

4.3. Taskonomy

In this experiment we extend our method to the recent Taskonomy dataset (Zamir et al., 2018), which contains over 4.5 million indoor images from over $5,000$ buildings. Following (Sun et al., 2019), we select surface normal, edge detection, keypoint detection, monocular depth, and semantic segmentation among the total 26 tasks for the experiment. We use the standard tiny split benchmark, which contains 275K training, 54K test, 52K validation images. We again follow the work in (Sun et al., 2019) to report test losses on these tasks for standardized comparisons.

Implementation details. We follow (Sun et al., 2019) to use the ResNet-34 (He et al., 2016) backbone and the ASPP decoder (Chen et al., 2017) for task-specific dense predictions. We use L1 loss for edge detection, keypoint detection and monocular depth tasks, cross-entropy loss for semantic segmentation task, and cosine similarity loss for surface normal task.

During the topology searching, we allow the network to branch at the end of every ResNet block. Each branching block has 5 child nodes so the network capacity is similar to the baseline model in (Sun et al., 2019). We use the Adam optimizers with mini-batch size 64 to train our network. Temperature is set to 10 and decayed by the number of epochs. We again warmup the training for 2 epochs without updating the branching probabilities. We perform grid search for learning rates in $(10^{-4}, 10^{-3}, 10^{-2})$ for both the weights and branching distributions, and weight decay in $(10^{-5}, 10^{-4}, 10^{-3})$. We train the network topology distribution on input image size of 128×128 and re-train the final selected network on image size of 256×256 for comparisons on the same image resolution. The input is normalized in $[-1, 1]$ and augmented by random clipping, scaling, and cropping. On a single 32GB Tesla GPU, it takes 2 days to train the topology distribution and 3 days to obtain the final converged network.

Results. Following (Chen et al., 2017; Standley et al., 2019; Sun et al., 2019), we use the cross-entropy metrics with uncertain and background pixels masked out for segmentation; we use cosine similarities between the predictions and ground truth vectors without any masks for surface normal; we calculate the absolute mean error between the output and normalized ground truth with pixels whose depths are more than 126m masked out for depth; for the rest of the tasks, we calculate the absolute mean error between the output and normalized ground truth without any mask.

Table 2 lists the results that are based on the same evaluation protocol. We can clearly see that our method achieves the best performance on all 5 tasks compared to all recent baselines. The first row in Table 2 shows the Single-Task setting where each task has its own set of parameters. It achieves lower performance than ours while having more than double parameter count. Our method is also more efficient than the recent Cross-Stitch and MTAN benchmarks as shown in the Table. AdaShare (Sun et al., 2019) uses a single-path network with adaptive skip-connections for multi-task learning. While the method has slightly less number of parameters, it achieves lower performance than our method on all 5 tasks, showing the importance of feature sharing and branching.

We randomly sample four converged architectures after training and visualize them in Figure 4. We observe that even though the data flows take different paths in the four architectures, the final pruned network topology remains very similar to each other. Figure 4(a), 4(c), 4(d) share exactly the same tree structure with edge and keypoint branch out

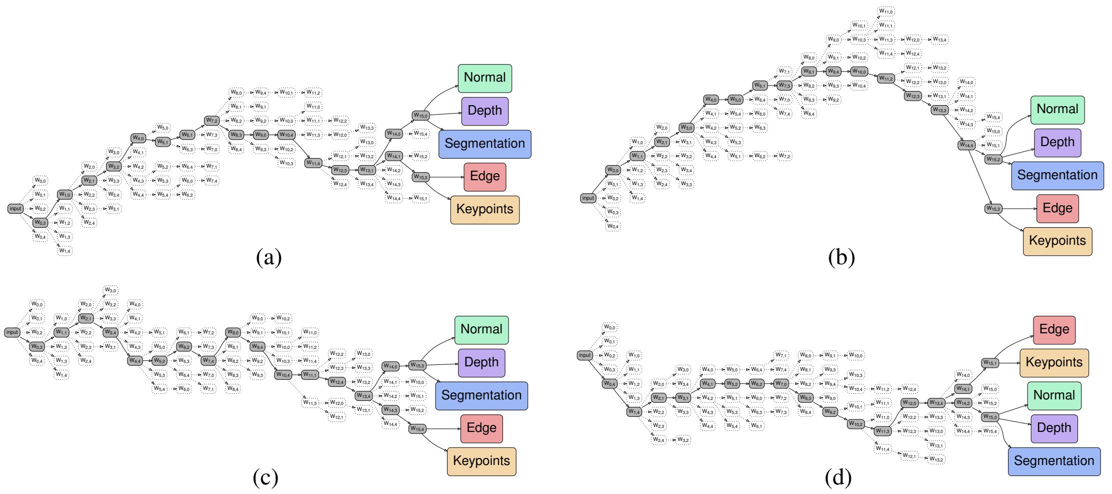


Figure 4. Four randomly sampled network architectures trained on Taskonomy dataset. Our method discovers the same task grouping strategy in network (a), (c), and (d). Network (b) branches out at one layer later compared to the others but still shares the same task grouping strategy.

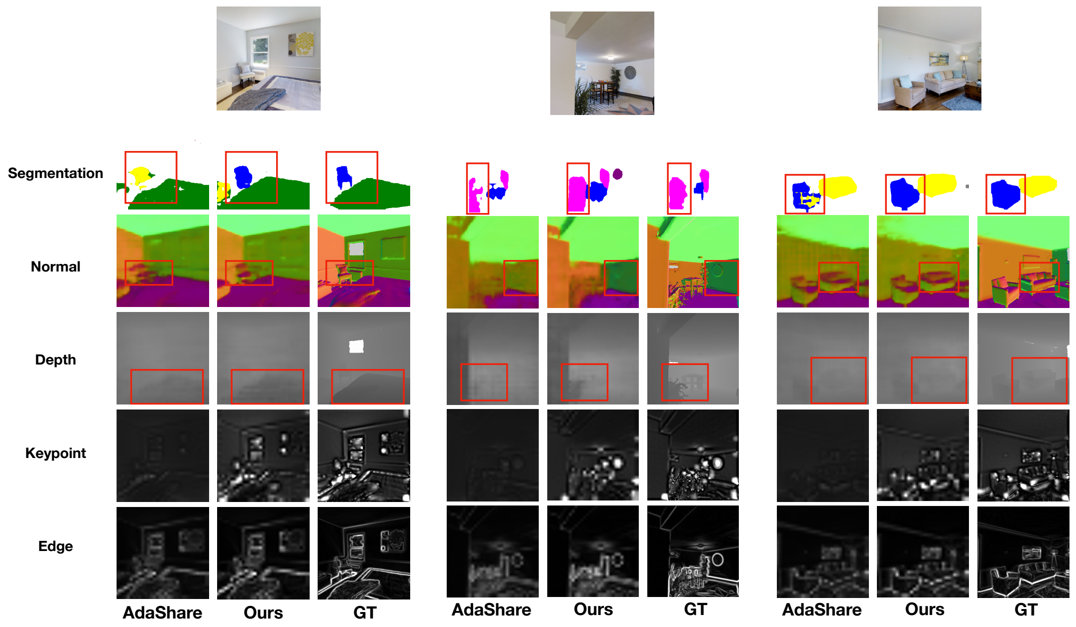


Figure 5. Qualitative results of AdaShare (Sun et al., 2019), our method, and ground truth. Our multi-task apporach produces cleaner predictions for high-level tasks (segmentation, normal, depth) and more accurate confidence scores for low-level tasks (keypoint, edge).

from normal, depth and segmentation at the second last layer. Figure 4(b) has a slightly different configuration that the branching occurs at the last layer. However, all the searched architectures show similar strategies for task grouping that are automatically found by the proposed method.

To further validate the effectiveness of the proposed branching operation, we directly sample a new network architecture without training the branching probability using the same topological space. Figure 5 shows the qualitative results of the output from AdaShare (Sun et al., 2019), the output from our method, and the ground truth. We can see that our converged network produces better results visually, especially for segmentation, depth estimation, keypoint prediction and edge detection tasks.

5. Conclusion

In this work, we introduce an automated multi-task learning framework that learns the underlying task grouping strategies by sharing and branching a neural network. We propose a carefully designed topological space to enable direct optimization for both the weights and branching distributions of the network through gumbel-softmax sampling. We validate the proposed method on controlled synthetic data, real-world CelebA, and large-scale Taskonomy dataset. Future work includes extension of our approach to multi-modality inputs and tasks with partial annotations.

References

Bau, D., Zhou, B., Khosla, A., Oliva, A., and Torralba, A. Network dissection: Quantifying interpretability of deep visual representations. In *CVPR*, 2017.

Caruana, R. Multitask learning. *Machine learning*, 1997.

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *TPAMI*, 2017.

Chen, Z., Badrinarayanan, V., Lee, C.-Y., and Rabinovich, A. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *ICML*, 2018.

Duong, L., Cohn, T., Bird, S., and Cook, P. Low resource dependency parsing: Cross-lingual parameter sharing in a neural network parser. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, 2015.

Gao, Y., Ma, J., Zhao, M., Liu, W., and Yuille, A. L. Nddr-cnn: Layerwise feature fusing in multi-task cnns by neural discriminative dimensionality reduction. In *CVPR*, 2019.

Guo, M., Haque, A., Huang, D.-A., Yeung, S., and Fei-Fei, L. Dynamic task prioritization for multitask learning. In *ECCV*, 2018.

Hand, E. M. and Chellappa, R. Attributes for improved attributes: A multi-task network utilizing implicit and explicit relationships for facial attribute classification. In *AAAI*, 2017.

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

Huang, S., Li, X., Cheng, Z.-Q., Zhang, Z., and Hauptmann, A. Gnas: A greedy neural architecture search method for multi-attribute learning. In *ACM*, 2018.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. In *ICLR*, 2017.

Kendall, A., Gal, Y., and Cipolla, R. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *CVPR*, 2018.

Kokkinos, I. Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. In *CVPR*, 2017.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012.

Lee, C.-Y., Gallagher, P. W., and Tu, Z. Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. In *AISTATS*, 2016.

Liang, J., Meyerson, E., and Miikkulainen, R. Evolutionary architecture search for deep multitask networks. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2018.

Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., Fei-Fei, L., Yuille, A., Huang, J., and Murphy, K. Progressive neural architecture search. In *ECCV*, 2018.

Liu, H., Simonyan, K., and Yang, Y. Darts: Differentiable architecture search. In *ICLR*, 2019a.

Liu, S., Johns, E., and Davison, A. J. End-to-end multi-task learning with attention. In *CVPR*, 2019b.

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

Long, M. and Wang, J. Learning multiple tasks with deep relationship networks. *arXiv preprint arXiv:1506.02117*, 2015.

Lu, Y., Kumar, A., Zhai, S., Cheng, Y., Javidi, T., and Feris, R. Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification. In *CVPR*, 2017.

Maziarz, K., Kokiopoulou, E., Gesmundo, A., Sbaiz, L., Bartok, G., and Berent, J. Gumbel-matrix routing for flexible multi-task learning. *arXiv preprint arXiv:1910.04915*, 2019.

Meyerson, E. and Miikkulainen, R. Beyond shared hierarchies: Deep multitask learning through soft layer ordering. In *ICLR*, 2018.

Misra, I., Shrivastava, A., Gupta, A., and Hebert, M. Cross-stitch networks for multi-task learning. In *CVPR*, 2016.

Mitchell, T. M. *The need for biases in learning generalizations*. Department of Computer Science, Laboratory for Computer Science Research, Rutgers University, 1980.

Pasunuru, R. and Bansal, M. Continual and multi-task architecture search. In *ACL*, 2019.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. Efficient neural architecture search via parameter sharing. In *ICML*, 2018.

Rosenbaum, C., Klinger, T., and Riemer, M. Routing networks: Adaptive selection of non-linear functions for multi-task learning. In *ICLR*, 2018.

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

Ruder, S. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

Ruder, S., Bingel, J., Augenstein, I., and Søgaard, A. Sluice networks: Learning what to share between loosely related tasks. *ArXiv*, abs/1705.08142, 2017.

Ruder, S., Bingel, J., Augenstein, I., and Søgaard, A. Latent multi-task architecture learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

Sciuto, C., Yu, K., Jaggi, M., Musat, C., and Salzmann, M. Evaluating the search phase of neural architecture search. In *ICLR*, 2019.

Sener, O. and Koltun, V. Multi-task learning as multi-objective optimization. In *Advances in Neural Information Processing Systems*, 2018.

Shaw, A., Wei, W., Liu, W., Song, L., and Dai, B. Meta architecture search. In *Advances in Neural Information Processing Systems*, 2019.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

Standley, T., Zamir, A. R., Chen, D., Guibas, L., Malik, J., and Savarese, S. Which tasks should be learned together in multi-task learning? *arXiv preprint arXiv:1905.07553*, 2019.

Sun, X., Panda, R., and Feris, R. Adashare: Learning what to share for efficient deep multi-task learning. *arXiv preprint arXiv:1911.12423*, 2019.

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

Vandenhende, S., De Brabandere, B., and Van Gool, L. Branched multi-task networks: deciding what layers to share. *arXiv preprint arXiv:1904.02920*, 2019.

Veit, A. and Belongie, S. Convolutional networks with adaptive inference graphs. In *ECCV*, 2018.

Wang, J., Cheng, Y., and Schmidt Feris, R. Walk and learn: Facial attribute representation learning from egocentric video and contextual data. In *CVPR*, 2016.

Wong, C., Houlsby, N., Lu, Y., and Gesmundo, A. Transfer learning with neural automl. In *Advances in Neural Information Processing Systems*, 2018.

Xie, S., Zheng, H., Liu, C., and Lin, L. Snas: stochastic neural architecture search. In *ICLR*, 2019.

Zamir, A. R., Sax, A., Shen, W., Guibas, L. J., Malik, J., and Savarese, S. Taskonomy: Disentangling task transfer learning. In *CVPR*, 2018.

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

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. In *ICLR*, 2017.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018.

Appendix

A. Implementation Details

In this section, we provide additional implementation details for experiments on CelebA dataset (Liu et al., 2015) and Taskonomy dataset (Zamir et al., 2018).

CelebA.
(a) LearnToBranch-VGG network: we train the network topological distribution for 30 epochs. The global learning rate is set to 10^{-5} and the learning rate for branching operation is set to 10^{-4}. We use exponential learning decay with decay factor 0.97 for every 2.4 epochs. After sampling the final architecture, we train the network for 30 epochs from scratch. We set the learning rate to 0.03, the weight decay to $5e^{-4}$, and the momentum to 0.9. We decay the learning rate by half for every 10 epoch.

(b) LearnToBranch-Deep-Wide network: we train the network topological distribution for 30 epochs. The global learning rate is set to 10^{-4} and the learning rate for branching operation is set to 10^{-2}. We use exponential learning decay with decay factor 0.97 for every 2.4 epochs. After sampling the final architecture, we train the network for 30 epochs from scratch. We set the learning rate to 0.05, the weight decay to $5e^{-4}$, and the momentum to 0.9. We decay the learning rate by half for every 15 epoch.

We visualize both network architectures (a) and (b) in Figure 6. We observe some grouping strategies learned by our method share some similarities with human intuition. For instance, network (a) groups 'Eyeglasses' and 'Narrow Eyes' and groups 'Mustasche' and 'No Beard'. Network (b) groups 'Black Hair' and 'Gray Hair' and groups 'Bald' and 'Receding Hairline'.

Taskonomy.
We train the network topological distribution for 30 epochs. The global learning rate is set to 10^{-3}, the learning rate for branching operations is set to 10^{-1}, and weight decay is set to 10^{-5}. We use exponential learning decay with decay factor 0.97 for every 1 epoch.

After sampling the final architecture, we train the network for 30 epochs from scratch. We set the learning rate to $5e^{-4}$, the weight decay to 10^{-4}, and the momentum to 0.9. We use exponential learning decay with decay factor 0.97 for every 1 epoch.

We follow the work in (Sun et al., 2019) and set the following task weightings: 1.0 for semantic segmentation, 3.0 for surface normal estimation, 2.0 for depth estimation, 7.0 for keypoint prediction, and 7.0 for edge detection. Note that we can further combine the proposed method with other adaptive task weighting methods. We leave this effort for future investigation.

Again following (Sun et al., 2019), for the semantic segmentation task, we ignore uncertain pixels (class 0) and background pixels (class 1). For the monocular depth estimation task, we ignore pixels with depth value larger than 64500 and normalize the disparities by taking the $\log$ operation and downscale by a factor of $\log(2^{16})$. For the surface normal prediction task, we normalize the three-dimensional normal vector from $[0, 255]$ to $[-1, 1]$. For the keypoint estimation and the edge detection tasks, we downscale the original values by a factor of 2^{16}. We then normalize the values from $[0, 0.005]$ to $[-1, 1]$ for keypoints and from $[0, 0.08]$ to $[-1, 1]$ for edges.

B. Learned Branching Features

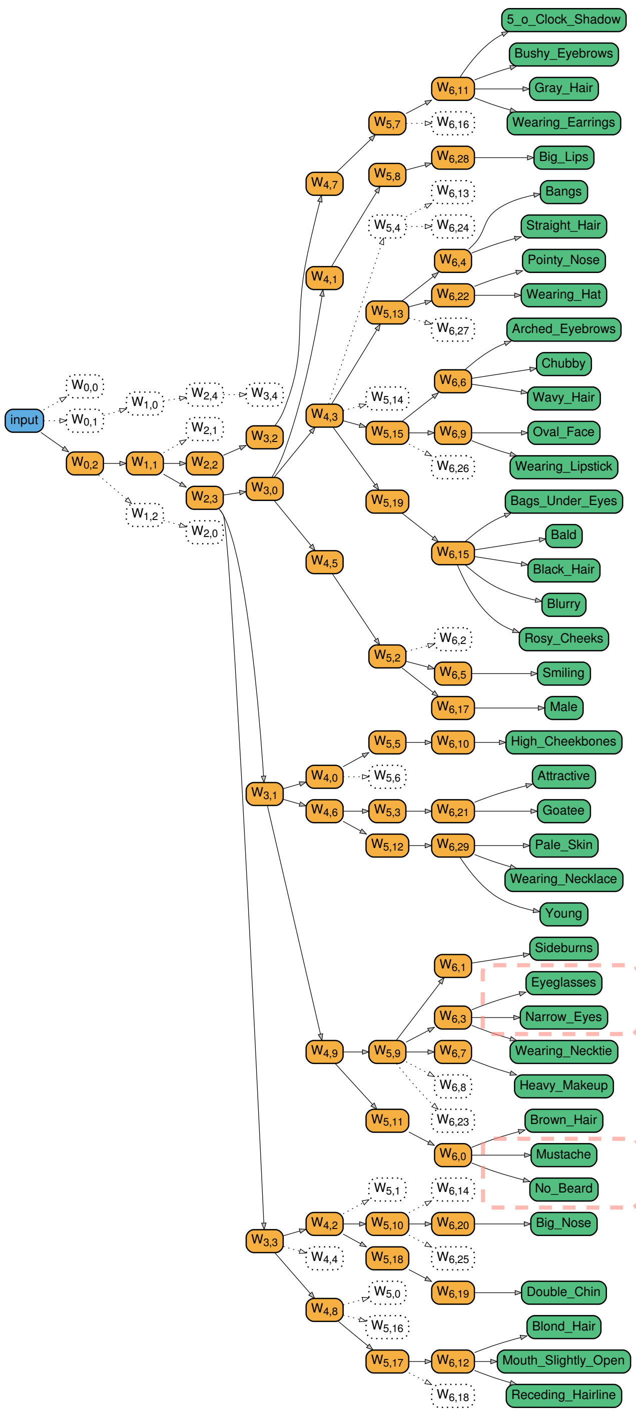
We use Network Dissection (Bau et al., 2017) to examine the features learned from Taskonomy dataset. We found that the SDN {segmentation, depth, normal} branch shows 35% increase in high-level features (object and part detectors) and 20% decrease in low-level features (texture detectors) compared to the shared layer before splitting. On the other hand, the EK {edge, keypoint} branch continues to focus on low-level features, showing no increase in high-level features due to the fact that {edge, keypoint} tasks are generally considered low-level tasks. Table 3 lists the number of detector counts before and after the branching (layer 13).

Table 3. Detector counts for different categories of input images at different layers using Network Dissection (Bau et al., 2017).

LAYER	OBJECT+PART DETECTORS	TEXTURE DETECTORS
LAYER_{13}	116	262
$\text{LAYER}_{14,\,\text{SDN}}$	157	208
$\text{LAYER}_{14,\,\text{EK}}$	118	253

C. Generalizability of the Learned Branching

We investigate whether the task grouping strategy learned from Tasknomoy dataset can be transferred to NYUv2 dataset on the three shared tasks across the two datasets. Following the metrics in Table 2, for {segmentation, normal, depth} tasks, we found that the grouping learned from Tasknomoy achieves {1.611, 0.739, 0.058} on NYUv2 test set while the grouping learned from NYUv2 training set achieves {1.572, 0.748, 0.058} on NYUv2 test set. The overall performance difference is relatively small at 1.23%. The experiment is performed on the NYUv2 labelled dataset with 795 training images and 654 test images using 256×256 image resolution.


(a) LearnToBranch-VGG

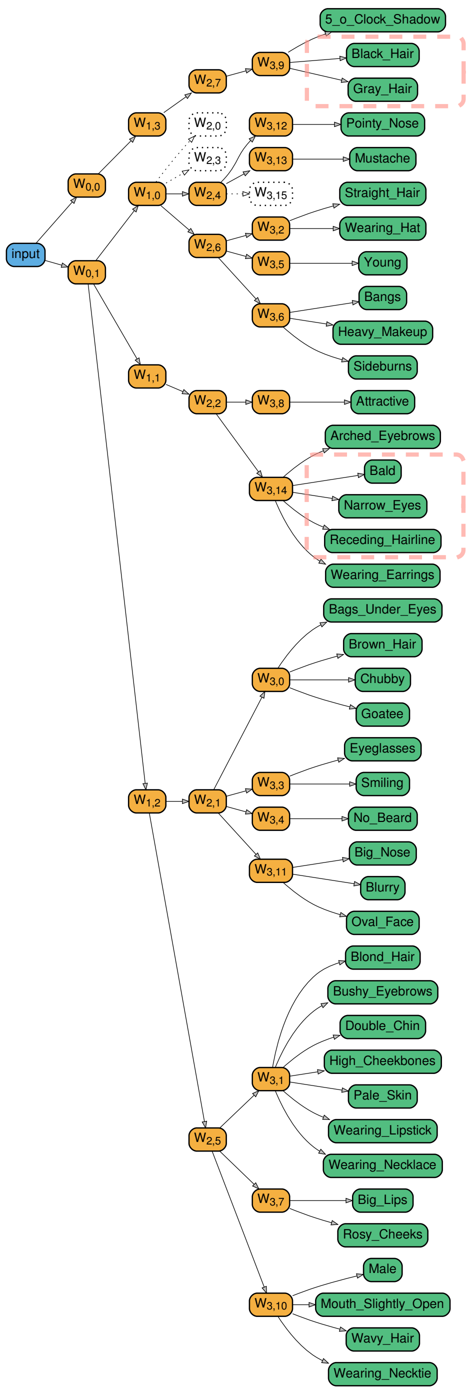

(b) LearnToBranch-Deep-Wide

Figure 6. Network architectures learned from CelebA dataset. We observe some grouping strategies learned by our method share some similarities with human intuition. For instance, network (a) groups 'Eyeglasses' and 'Narrow Eyes' and groups 'Mustasche' and 'No Beard'. Network (b) groups 'Black Hair' and 'Gray Hair' and groups 'Bald' and 'Receding Hairline'. The groups are shown in red dotted rectangles. Transparent boxes denote removed nodes because they are not selected by any child nodes.