Self-born Wiring for Neural Trees
–Supplementary Material–

Ying Chen[1], Feng Mao[2], Jie Song[1], Xinchao Wang[3], Huiqiong Wang[4]*, and Mingli Song[1]
[1]Zhejiang University, [2]Alibaba Group, [3]National University of Singapore
[4]Ningbo Research Institute, Zhejiang University
{lynesychen,sjie,huiqiong_wang,brooksong}@zju.edu.cn,
maofeng.mf@alibaba-inc.com, xinchao@nus.edu.sg

We provide in this document additional details and results that cannot fit into the main manuscript due to the page limit. Section 1 summarizes our training strategy for the proposed SeBoW, and Section 2 offers additional experimental results. Section 3 provides more implementation details, and Section 4 concludes this document with more visualization results.

1. Training Strategy of SeBoW

In this section, we give a more detailed description of the training process of the proposed self-born wirings (SeBoW) for neural tree. The training of the proposed model follows two stages: *search phase* and *retraining phase*.

In the search phase, SeBoW begins with a densely connected architecture as the search space, which is implemented by unpacking base learnings from a DNN, installing routers, and mounting solvers, as described in the main paper. The learners are used for representation learning, the routers for routing the data flow through the network, and the solver for making the final predictions for the task. Specially, the routers comprise two modules: *senders* and *receivers*. The senders are devised for passing the output of the current learners to the next learner, and the receivers for aggregating the output data from previous learners as the input for the current learners.

Algorithm 1 summarizes the whole training procedure in the search phase of SeBoW. As described in Line #4 $\sim$ 6 in the algorithm, we initialize the parameters of receivers $\mathbf{w}$ with a uniform distribution prior to training. Each learner in the search space should receive data from the previous learners or the initial input, as shown in Line #15 $\sim$ 17. After that, the learner in this node is used for representation learning (Line #19). The router r and learned representations are used to compute the conditional probability of passing the data to the following learners (Line #21).

Algorithm 1: Tree Search Phase of SeBoW

1 **Define:** a densely connected architecture as search space.
2 **Define:** model operations $\mathbb{O}$, including learners l, routers (senders r and receivers $\mathbf{w}$), solvers s.
3 **Define:** probability of reaching the j-th node in section i as $\mathbf{p}_j^i$, feature of reaching the j-th node in section i as $\mathbf{x}_j^i$.
4 **for** *each receiver* $\mathbf{w}$ **do**
5 Initialize $\mathbf{w}$ by sampling from a uniform distribution;
6 **end**
7 **for** *each training iteration* e **do**
8 Sample mini batch of data and labels $(\mathbf{x}_e, \mathbf{y}_e) \in (\mathcal{X}, \mathcal{Y})$;
9 Set the Initial input probability as $\mathbf{p}_1^1 \leftarrow 1$;
10 Set the Initial input data as $\mathbf{x}_1^1 \leftarrow \mathbf{x}_e$;
11 Reset label prediction probability and node probability to 0;
12 **for** *each section* i **do**
13 **for** *each node* j **do**
14 **if** *section* i *is not root* **then**
15 Outputs from previous section: $\mathbf{Y}^{i-1} \leftarrow [\mathbf{y}_1^{i-1}, \mathbf{y}_2^{i-1}, \ldots, \mathbf{y}_{C_{i-1}/C}^{i-1}]$;
16 Forward discrete decision making: $\mathbf{h}_j^i = \text{one\_hot}\{\arg\max_k (\log w_{j,k}^i) + \epsilon_k\}$ ▷ More details in the main manuscript;
17 $\mathbf{x}_j^i \leftarrow \mathbf{h}_j^i \cdot \mathbf{Y}^{i-1}$;
18 **end**
19 $\mathbf{y}_j^i \leftarrow l_j^i(\mathbf{x}_j^i)$; ▷ Data transformer;
20 **if** *section* i *is not leaf* **then**
21 $[\mathbf{p}_1^{i+1}, \mathbf{p}_2^{i+1}, \ldots, \mathbf{p}_{C_{i+1}/C}^{i+1}]$ += $\mathbf{p}_j^i \cdot r_j^i(\mathbf{y}_j^i)$;
22 **else**
23 Compute label predictions: $[\hat{\mathbf{p}}_1, \hat{\mathbf{p}}_2, \ldots, \hat{\mathbf{p}}_N]$ += $\mathbf{p}_j^i \cdot s_j(\mathbf{y}_\mathbf{j}^\mathbf{i})$;
24 **end**
25 **end**
26 **end**
27 Optimise parameters via gradient descent [10] on negative log-likelihood loss $\mathcal{J}$
28 **end**

When the data is transferred to leaf learner, solver s is activated for the final prediction (Line #23).

*Corresponding author

Algorithm 2: Neural Tree Optimization

Define: selected architecture.
Define: model operations $\mathbb{O}$, including learners l, routers r and solvers s.
Define: probability of reaching the j-th node in section i as $\mathbf{p}_j^i$, feature of reaching the j-th node in section i as $\mathbf{x}_j^i$.
for *each training iteration* e **do**
  Sample mini batch of data and labels $(\mathbf{x}_e, \mathbf{y}_e) \in (\mathcal{X}, \mathcal{Y})$;
  Set the Initial input probability as $\mathbf{p}_1^1 \leftarrow 1$;
  Set the Initial input data as $\mathbf{x}_1^1 \leftarrow \mathbf{x}_e$;
  for *each section* i **do**
    for *each node* j **do**
      $\mathbf{y}_j^i \leftarrow l_j^i(\mathbf{x}_j^i)$; ▷ Data transformer;
      Find child in next section as CIs;
      if *section* i *is not leaf* **then**
        $\mathbf{p}_{CIs}^{i+1} \leftarrow \mathbf{p}_j^i \cdot r_j^i(\mathbf{y}_j^i)$;
        $\mathbf{x}_{CIs}^{i+1} \leftarrow y_i^j$;
      else
        Compute label predictions:
        $\hat{\mathbf{p}}_{CIs} \leftarrow \mathbf{p}_j^i \cdot s_j(\mathbf{y}_\mathbf{j}^\mathbf{i})$;
      end
    end
  end
  Optimise parameters via gradient descent on negative log-likelihood loss $\mathcal{J}$
end

The final category prediction result is expressed as the product of the category prediction made by the solver and the probability of reaching the solver. Since we obtain the final label prediction, the negative log-likelihood loss is adopted to optimize the whole network (Line #27). It is worth noting that the variable h_j^i will be relaxed using the Gumbel-softmax [6] function during backward propagation. This sampling distribution changes from being uniform to sharp as the temperature decreases.

After sampling the final architecture with model parameters, we begin the optimization of the selected architecture using Algorithm 2. Due to the unification of probability and data allocation in the retraining phase, we removed the receiver modules, resulting in a neural tree with an explicit parent-child relationship. Similarly, in the neural tree, data transformation is performed by the learner l, and probability calculation is implemented by the router r. Each node is capable to transmit its output to the children of a particular set (Line #11 $\sim$ 14). Eventually, the model parameters are updated using negative log-likelihood loss in Line #20.

2. Additional Experiments

More mother DNNs. We apply our SeBoW framework on the ResNet18-based search space to further evaluate the generalization ability of our method. We search for several architectures with different parameters, then compare them with the original ResNet-18 model and neural trees on CIFAR-10, CIFAR-100, and tiny-ImageNet classification tasks. Quantitative data can be seen in Tables 1, 2 and 3. As can be seen, our proposed SeBoW-ResNet18 promotes the accuracy of ResNet-18 on CIFAR-10, CIFAR-100 and tiny-ImageNet by 2.37%, 4.51% and 3.06% with 1.6M, 1.5M, 1.5M fewer parameters respectively.

Method	Params.	Accuracy (%)
MobileNet [5]	2.2M	85.90 (±0.23)
VGG-13 [12]	28.3M	92.51 (±0.15)
ResNet-18 [4]	11.2M	92.98 (±0.17)
ANT-CIFAR10-C [14]	*0.7M / 0.5M*	*90.69 / 90.66*
ANT-CIFAR10-B [14]	*0.9M / 0.6M*	*90.85 / 90.82*
ANT-CIFAR10-A [14]	*1.4M / 1.0M*	*91.69 / 91.68*
ANT-CIFAR10-A (ensemble) [14]	*8.7M / 7.4M*	*92.29 / 92.21*
XOC [1] + ResNet-18	$>$ 11.2M	93.12 (±0.32)
SeBoW-A	1.0M / 0.7M	93.45 (±0.12) / 93.41
SeBoW-B	2.7M / 1.6M	94.00 (±0.18) / 93.93
SeBoW-C	5.8M / 4.6M	**94.33** (±0.14) / **94.24**
SeBoW-ResNet18-A	1.4M / 1.3M	93.87 (±0.09) / 93.79
SeBoW-ResNet18-B	3.1M / 2.4M	94.39 (±0.11) / 94.29
SeBoW-ResNet18-C	9.6M / 9.6M	**95.31** (±0.07) / **95.31**

Table 1. Performance comparison on CIFAR-10. Underlined numbers denote the results of single-path inference, *Italic* fonts denote that the results are taken from the original paper, and "N/A" means not applicable.

Method	Params.	Accuracy (%)
MobileNet [5]	2.4M	53.91 (±0.32)
VGG-13 [12]	28.7M	72.70 (±0.42)
ResNet-18 [4]	11.2M	72.28 (±0.28)
ANT-Extend [14]	4.2M / 4.2M	65.81 (±0.12) / 65.71
SeBoW-B	1.9M / 1.5M	71.79 (±0.23) / 71.59
SeBoW-C	4.2M / 4.2M	74.59 (±0.33) / **74.59**
SeBoW-ResNet18-B	3.4M / 2.5M	73.97 (±0.15) / 73.83
SeBoW-ResNet18-C	12.4M / 9.7M	**76.91** (±0.12) / **76.79**

Table 2. Performance comparisons on the CIFAR-100 dataset.

Method	Params.	Accuracy (%)
MobileNet [5]	2.5M	46.12 (±0.73)
GoogleNet [13]	6.8M	48.85 (±0.52)
VGG-13 [12]	28.7M	56.10 (±0.57)
ResNet-18 [4]	11.2M	55.32 (±0.75)
SeBoW-C	8.4M / 4.8M	**58.77** (±0.39) / **58.43**
SeBoW-ResNet18-C	13.2M / 9.7M	**58.59** (±0.41) / **58.38**

Table 3. Performance comparison on the tiny-ImageNet dataset.

Dataset	Method	Fine-tuning	Retraining	Params.
CIFAR10	SeBoW-A	91.65 (±0.04)	**93.45** (±0.12)	1.0M
	SeBoW-B	92.83 (±0.15)	**94.00** (±0.18)	2.7M
CIFAR100	SeBoW-B	70.21 (±0.11)	**71.79** (±0.23)	1.9M
	SeBoW-C	72.89 (±0.17)	**74.59** (±0.33)	4.2M

Table 4. Performance of SeBoW with retraining or fine-tuning in the second phase.

	Small Datasets $[32 \times 32]$	Medium Datasets $[64 \times 64]$	Large Datasets $[224 \times 224]$
Hyper Parameter	CIFAR10 and CIFAR100 [2]	tiny-ImageNet [8]	ImageNet [3]
Learning Rate	0.1	0.1	0.05
Optimiser	SGD with 0.9 momentum	SGD with 0.9 momentum	SGD with 0.9 momentum
Learning Rate Update	decay it by half for every 20 epoch	decay it by half for every 20 epoch	decay it by 10 for every 30 epoch
Initialisation of Receivers	uniform distribution **N/A**	uniform distribution **N/A**	uniform distribution **N/A**
Temperature	10 **N/A**	10 **N/A**	10 **N/A**
Temperature Decay	decay it by the current epoch **N/A**	decay it by the current epoch **N/A**	decay it by the current epoch **N/A**
Weight Decay	10^{-4} $\mathbf{5 \times 10^{-4}}$	10^{-4} $\mathbf{5 \times 10^{-4}}$	10^{-4} $\mathbf{5 \times 10^{-4}}$
Scheduler	-	-	Cosine Annealing
Batch Size	128	128	32 (per GPU) for 4 GPUs
Epochs	100	100	150

Table 5. The complete set of hyper-parameters used in our method. Texts in **bold** font denote the hyper-parameters used in retraining phase, which are diverse from those utilized in search phase. "N/A" means not applicable.

Retraining *versus* fine-tuning. After conducting the architecture search in the first phase, we retrain the searched neural tree from scratch with random initialization in the second phase. Here we make comparisons between retraining and fine-tuning in this phase. Results are shown in Table 4, where retrained models achieve higher accuracy than fine-tuned models by a noticeable margin. It coincides with the observations in [9], where retrained models with randomly initialized weights are found to give superior performance to fine-tuned ones when structured pruning is carried out.

3. Implementation

For reproducibility, we present additional implementation details during two phases in Table 5, across all SeBoW variants and datasets evaluated in the main manuscript. If the two phases have different hyper-parameters, we will illustrate them in **bold**. Since we remove the receiver modules in the second phase, the initialization method and temperature for gumbel-softmax become inapplicable. Also, we set the weight decay 10^{-4} for faster convergence in the first phase and 5×10^{-4} for more stable training in the second phase. We provide further details on data preprocessing, which are summarized as follows.

CIFAR and tiny-ImageNet. We adopt the standard image preprocessing scheme widely used in the literature [14, 4], including zero-padded with 4 pixels on each side, random cropping, random horizontal flipping, and image normalization.

ImageNet. Our implementation for ImageNet follows the practice in [12, 3, 4]. The input image is resized by setting its shorter side to 256, then a 224×224 crop is randomly sampled from this image or its horizontal flip. The standard color augmentation and image normalization operation in [7] are used.

4. Neural Tree Visualizations

We have presented the architecture visualization of SeBoW-C model on CIFAR10 dataset in the main manuscript. Here we provide more visual results of the SeBoW neural tree.

Figure 1 (a-c) displays the tree visualization of various architectures on CIFAR10. We observe that some of the grouping strategies learned by SeBoW share similarities with human intuition: for example, categories belonging to animal (*e.g.*, *deer* and *dog*) show significant correlations across branches of these three architectures.

The SeBoW architecture discovered on CIFAR100 is

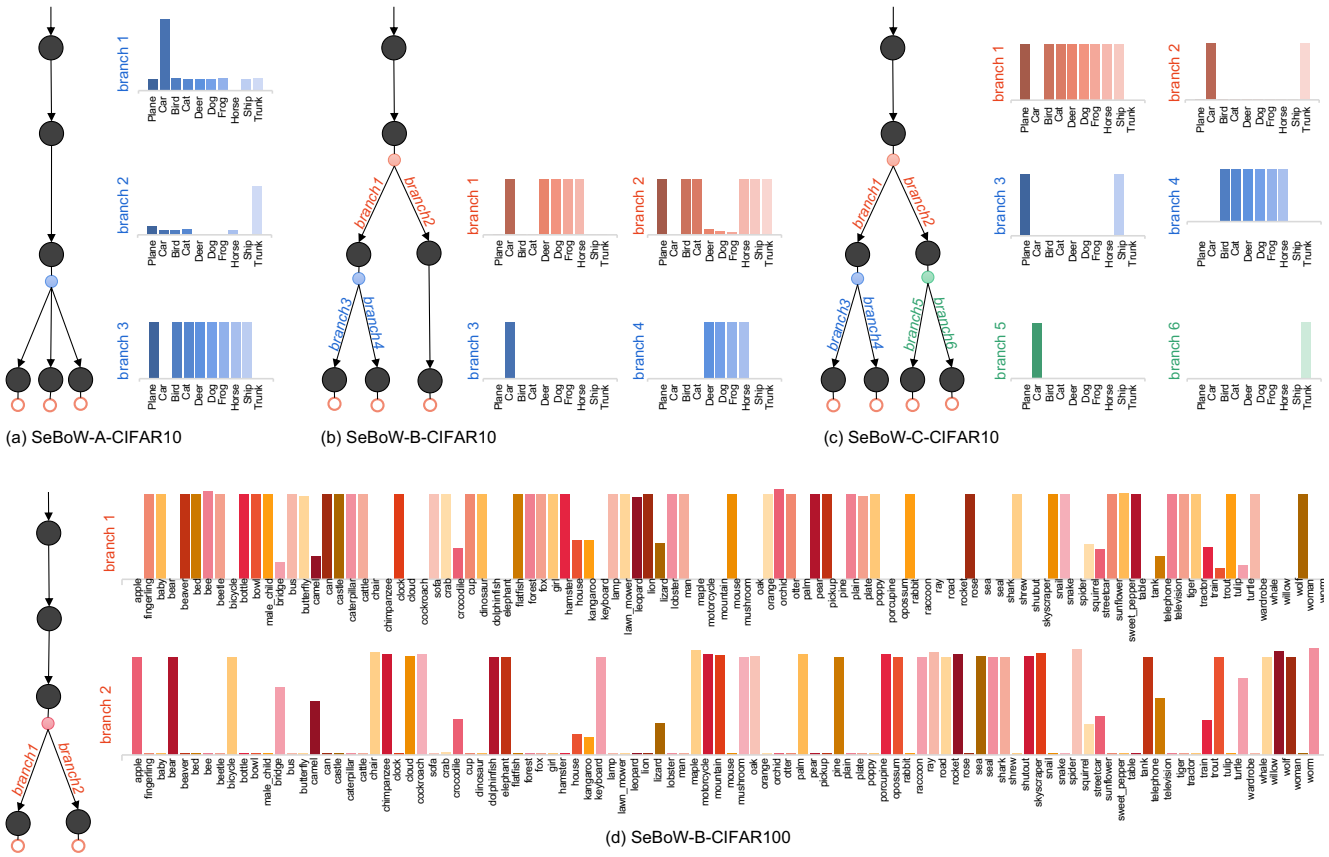

Figure 1. Illustration of various SeBoW architectures on CIFAR10 and CIFAR100 datasets. The category histogram corresponds to the category distribution under the same color branch point. Black circles represent learners, circles in blue or coral donate the routers, and white circles of the final layer are solvers. All the visualization results correspond to the experimental results of the main manuscript.

shown in Figure 1 (d). As we know, the categories in CIFAR100 are semantically divided into 20 superclasses, based on which we analyze the interpretability of the derived tree from SeBoW. For convenience, in what follows, superclasses are shown in *blue italics*. As members of the superclass *people* in CIFAR100, categories *baby*, *mail child*, *girl*, *man*, and *woman* are on the same branch of the neural tree, while categories *maple*, *oak*, *palm*, *pine*, and *willow* on another branch correspond to the *tree* superclass. Certainly, the branching results of network architecture are not always equivalent to the inter-class relationships derived from the semantics of dataset. *Whales* and *beavers* belong to the same superclass *aquatic mammals*, whereas SeBoW assigns them to different branches: *whales* and *sharks* (pertaining to *fish*) on the one side, *beavers* and *mice* (belonging to *small mammals*) on the other side.

We also visualize the network architecture produced on tiny-Imagenet in Figure 2. It can be seen that balls (*volleyball* and *basketball*), socks (*sock* and *christmas stocking*), and large animals (*brown bear*, *american alligator*, *bison* and *african elephant*) are first separated from all categories by branch 2. The remaining categories form branch 1, which are further split into branches 3 and 4 at the deeper level. We can analyze the pertinent branching basis from visual cues. Machines contain *tractor*, *crane*, *cash machine* and *sewing machine* are on the branch 3, and cylindrical objects including *torch*, *oboe*, *maypole* and *plunger* are on branch 4. Unlike some hierarchical classification methods [15], SeBoW does not enforce the categories of different branches to be disjoint, so as to explore the relationships between categories under the characteristics of different branches freely with less external constraints. For example, *centipede* appears on branch 1, 2, and 4. Diverse features are mined from various branches of SeBoW, and heatmaps are generated with features from the last convolutional layer of different branches by Grad-CAM [11].

Through the above analysis, we summarize the following three findings. Firstly, the branching for categories of the network is mainly based upon visual cues rather than semantic similarities defined by humans. Secondly, the features of neural tree become specialised to the partitioned space after branching. Thirdly, most architectures learn multiple levels of shared characteristics before resorting to branch.

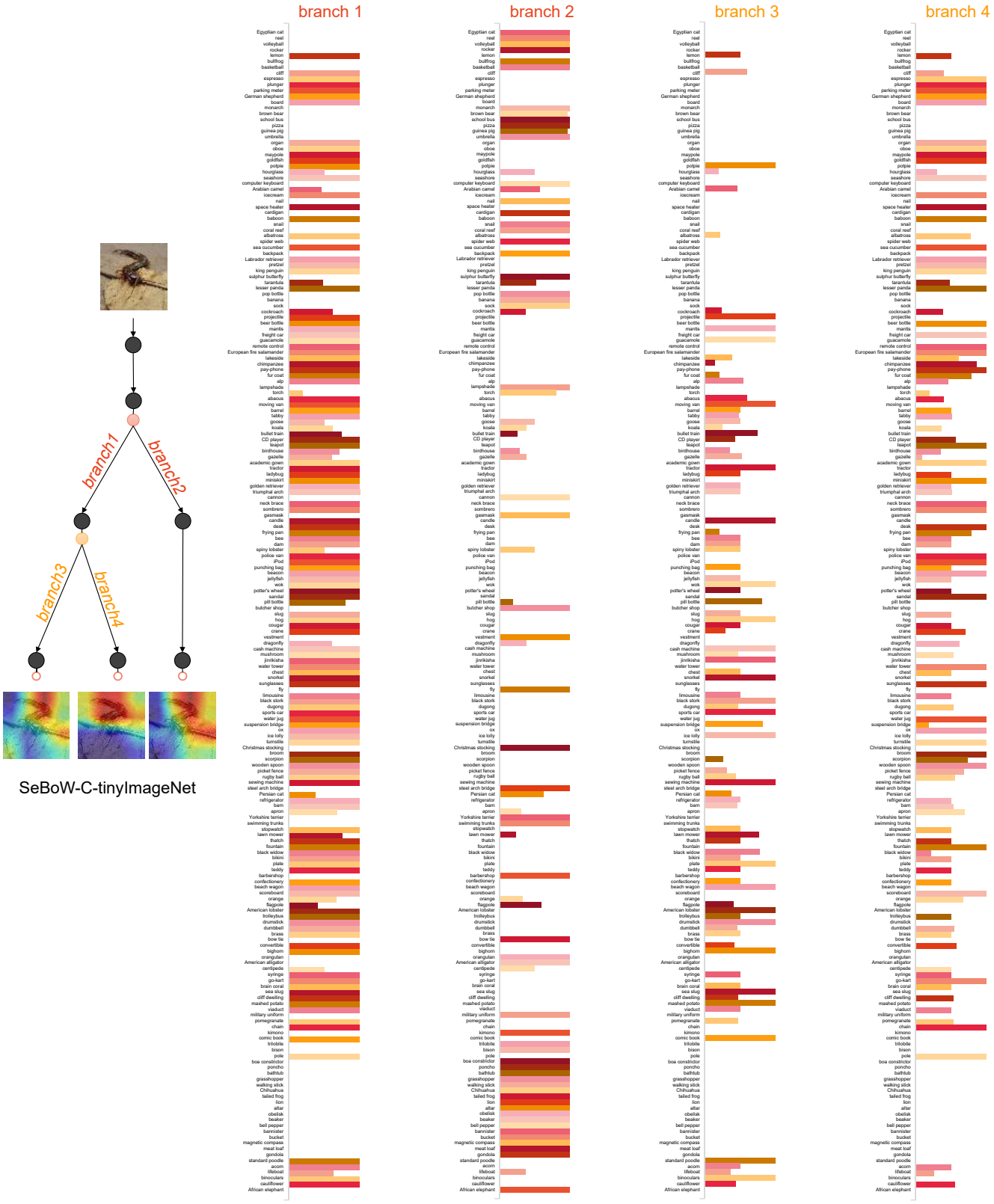

Figure 2. Illustration of the discovered architecture for SeBoW on tiny-ImageNet. Histograms show the class distributions at respective branches. We show heatmaps of three leaf nodes in SeBoW with a centipede image as input.

References

[1] Stephan Alaniz and Zeynep Akata. XOC: explainable observer-classifier for explainable binary decisions. *CoRR*, abs/1902.01780, 2019.

[2] Geoffrey Hinton Alex Krizhevsky. Learning multiple layers of features from tiny images, 2009.

[3] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255, 2009.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[5] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. In *CVPR*, 2017.

[6] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *ICLR*, 2017.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, pages 1106–1114, 2012.

[8] Y. Le and X. Yang. Tiny imagenet visual recognition challenge, 2015.

[9] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *ICLR*, 2019.

[10] Davd. Rumhar, offry. Hnon†, and RonadJ. Wams. Learning representations by back-propagating errors. *Nature*, 1986.

[11] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *ICCV*, pages 618–626, 2017.

[12] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[13] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, pages 1–9, 2015.

[14] Ryutaro Tanno, Kai Arulkumaran, Daniel C. Alexander, Antonio Criminisi, and Aditya V. Nori. Adaptive neural trees. In *ICML*, pages 6166–6175, 2019.

[15] Alvin Wan, Lisa Dunlap, Daniel Ho, Jihan Yin, Scott Lee, Henry Jin, Suzanne Petryk, Sarah Adel Bargal, and Joseph E. Gonzalez. NBDT: neural-backed decision trees. *CoRR*, abs/2004.00221, 2020.