Extracting Local Reasoning Chains of Deep Neural Networks

Anonymous authors
Paper under double-blind review

Abstract

We study how to explain the main steps of inference that a pre-trained deep neural net (DNN) relies on to produce predictions for a (sub)task and its data. This problem is related to network pruning and interpretable machine learning with the following highlighted differences: (1) fine-tuning of any neurons/filters is forbidden; (2) we target a very high pruning rate, e.g., $\geq 95\%$; (3) the interpretation is for the whole inference process on a few data of a task rather than for individual neurons/filters or a single sample. In this paper, we introduce NeuroChains to extract the local inference chains by optimizing differentiable sparse scores for the filters and layers, which reflects their importance in preserving the outputs on a few data drawn from a given (sub)task. Thereby, NeuroChains can extract an extremely small sub-network composed of critical filters exactly copied from the original pre-trained DNN by removing the filters/layers with small scores. For samples from the same class, we can then visualize the inference pathway in the pre-trained DNN by applying existing interpretation technique to the retained filters and layers . Its architecture reveals how the inference process stitches and integrates the information layer by layer and filter by filter. We provide detailed and insightful case studies together with several quantitative analyses over thousands of trials to demonstrate the quality, sparsity, fidelity and accuracy of the interpretation. In extensive empirical studies, NeuroChains significantly enriches the interpretation and makes the inner mechanism of DNNs more transparent.

1 Introduction

Deep neural networks (DNNs) greatly reshape a variety of tasks — object classification, semantic segmentation, natural language processing, speech recognition, robotics, etc. Despite its success on a vast majority of clean data, DNNs are also well-known to be sensitive to small amounts of adversarial noises. The lack of sufficient interpretability about their success or failure is one major bottleneck of applying DNNs to important areas such as medical diagnosis, public health, transportation systems, financial analysis, etc.

Interpretable machine learning has attracted growing interest in a variety of areas. The forms of interpretation vary across different methods. For example, attribution methods (Bach et al., 2015; Sundararajan et al., 2017; Shrikumar et al., 2017; Montavon et al., 2017; Kindermans et al., 2017; Smilkov et al., 2017) produce the importance score of each input feature to the output prediction for an given sample, while some other methods (Zeiler & Fergus, 2014; Simonyan et al., 2013; Erhan et al., 2009) aim to explain the general functionality of each neuron/filter or an individual layer regardless of the input sample. Another line of works (Ribeiro et al., 2016; Wu et al., 2018; Hou & Zhou, 2018) explain DNNs in a local region of data space by training a shallow (e.g., linear) and easily interpretable model to approximate the original pre-trained DNN on some locally similar samples. Thereby, they reduce the problem to explaining the shallow model. These methods essentially reveal the neuron to neuron correlations (e.g., input to output, intermediate layer/neuron to output, etc), but they cannot provide an overview of the whole inference process occurring inside the complicated structure of DNNs.

In this paper, we study a more challenging problem: *Can we unveil the major hidden steps of inference in DNNs and present them in a succinct and human-readable form?* Solving this problem helps to answer many significant questions, e.g., which layer(s)/neuron(s) plays the most/least important role in the inference process? Can two similar samples share most inference steps? Do all samples need the same number

of neurons to locate the key information leading to their correct predictions? Do DNNs rely on entirely different neurons or filters for different (sub)tasks? Some of them are related to other problems such as network pruning (Han et al., 2015; Li et al., 2016) and neural architecture search (NAS) (Zoph & Le, 2017). For example, are winning tickets (Frankle & Carbin, 2018; Liu et al., 2018) universal over different tasks or classes? Does the weight sharing scheme in recent NAS methods (Pham et al., 2018; Liu et al., 2019a; Ying et al., 2019) limit the searching space or quality?

We develop an efficient tool called NeuroChains to extract the underlying inference chains of a DNN only for a task with a few data, which in this paper refers to a few data drawn from a subset of classes. NeuroChains aims to extract a much smaller sub-network composed of a subset of neurons/filters exactly copied from the original pre-trained DNN and whose output for data from the subtask is consistent with that of the original pre-trained DNN. While the selected filters explain the key information captured by the original pre-trained DNN when applied to data from the same task/class, the architecture of the sub-network stitches these information sequentially, i.e., step by step and layer by layer, and recover the major steps of inference that lead to the final outputs. We parameterize the sub-network as the original DNN with an additional score multiplied to each filter/layer's output featuremap. Thereby, we formulate the above problem of sub-network extraction as optimizing differentiable sparse scores of all the filters and layers to preserve the outputs on all given samples.

The above problem can be solved by an efficient back-propagation that only updates the scores with fixed filter parameters. The objective is built upon the Kullback–Leibler (KL) divergence between the sub-network's output distribution and that of the original pre-trained DNN, along with an ℓ_1 regularization for sparse scores over filters. We further use a sigmoid gate per layer to choose whether removing the entire layer or block. The gate plays an important role in reducing the sub-network size since many data and subtasks do not rely on all the layers. We extract chains from the sub-network and visualize their filters and featuremaps of the original network by existing methods (Zeiler & Fergus, 2014; Erhan et al., 2009).

NeuroChains is a novel technique specifically designed for interpreting the local inference chains of DNNs. As aforementioned, it potentially provides an efficient tool to study other problems in related tasks. However, it has several **fundamental differences to network pruning and existing interpretation tasks**, which deter methods developed for these two problems from addressing our problem. **Comparing to network pruning**: (1) fine-tuning is not allowed in NeuroChains; (2) it targets a much larger pruning rate for succinct visualization, e.g., $\geq 95\%$ for VGG-19 and $\geq 99\%$ for ResNet-50 on ImageNet with ≤ 200 filters remained; (3) it is for a few samples drawn from a task instead of the whole data distribution. **Comparing to mainstream interpretation tasks**: (1) NeuroChains produces an interpretation of the entire inference process for a specified task (e.g., a subset of classes) rather than of one neuron/filter or output on/around a single sample; (2) NeuroChains provides complementary information to the importance of individual neurons/filters for a task.

2 Related Works

Interpretable machine learning methods can be mainly categorized into the ones aiming to evaluate the importance of each input feature of a single sample and the ones explaining individual neurons/filters. Approaches in the first category usually rely on certain back-propagation from the DNN's output to derive an importance score for each input feature or hidden node. Earlier works are based on the back-propagated gradients, e.g., deconvolution (Zeiler & Fergus, 2014), back-propagation (Simonyan et al., 2013) and guided back-propagation (Springenberg et al., 2014). Sundararajan et al. (2017) proposed to (approximately) calculate the integral of the gradients along a path between a baseline point and the input sample, which ensures the sensitivity and implementation invariance lacking in some previous methods. More recent methods propose novel back-propagation rules to directly derive the attribution scores of neurons from output to input, e.g., DeepLIFT (Shrikumar et al., 2017), deep Taylor decomposition (Montavon et al., 2017), and layer-wise relevance propagation (LRP) (Bach et al., 2015).

Methods in the second class treat DNNs as black boxes and seek simple models to explain how the DNN's output changes in a local region. For example, Ancona et al. (2017) add perturbations to different parts of the input to evaluate how the perturbations change the output, which reflect the importance of different

parts. Zeiler & Fergus (2014) covered different parts of the input with a gray square, which led to different prediction probabilities on the true class. Instead, Zintgraf et al. (2017) replaced each patch in the input image with the surrounding patch and tracked the induced changes in the output. In LIME, Ribeiro et al. (2016) trained a sparse linear model on noisy input-output pairs as a local surrogate approximating the original pre-trained DNNs, where the sparse weights are used to explain the importance of input features. As mentioned before, our main difference to the above methods are we explain DNNs for a (sub)task and its data and we further explain how DNNs step by step integrate the information of important filters/neurons.

Network pruning Han et al. (2015) and Li et al. (2016) remove redundant neurons/nodes or connections/weights from a pre-trained DNN and fine-tune the sub-network. Structural pruning removes whole layers/channels/filters/neurons according to a certain norm of the associated weights (Li et al., 2016) or sparsity (Hu et al., 2016). In contrast, Frankle & Carbin (2018) and Liu et al. (2018) prune a DNN during its training. Luo et al. (2017) apply pruning to two adjacent convolution layers at each time to take the dependency between the two layers into account. Liu et al. (2019b) and Guo et al. (2020) train sub-networks of different sizes in a large DNN at the same time to satisfy various constraints. Several recent works (Su et al., 2020; Evci et al., 2020) empirically verify "lottery ticket hypothesis", i.e., there exists sub-networks (i.e., winning tickets) that can reach comparable generalization performance as the original pre-trained DNN if re-trained. In contrast, the sub-network extracted by NeuroChains cannot be fully re-trained since it has to preserve the original pre-trained DNN's filters, and our goal is to retain the generalization performance only for a task with few data.

3 NeuroChains

3.1 Problem: Extract Sub-networks

Although the DNNs widely used nowadays are usually composed of hundreds of layers and millions to billions of hidden nodes. *When applied to samples from a subtask (e.g., composed of two classes), it is plausible that its inference process mainly relies on a small subset of layers and filters.* In this paper, we verify this conjecture by developing an efficient and practical algorithm, i.e., NeuroChains, to extract the subset and its underlying architecture as a sub-network whose filters are selected and exactly copied from the original pre-trained DNN while its outputs for a given subset of data or classes retain the ones produced by the original pre-trained DNN. Although DNNs are usually non-smooth in definition if using a non-smooth piecewise activation such as ReLU, when trained with the commonly used techniques, e.g., data augmentation, mix-up, dropout, the resulted DNNs are relatively smooth in a sufficiently small local region.

In order to preserve the original inference chains, we do not allow any fine-tuning or re-training on any filter or the weight vector corresponding to any neuron: they can only be exactly copied from the original pre-trained DNN. Let $F(\cdot; \{W^\ell\}_{\ell=1:L})$ (a mapping from input to output) denote the original pre-trained DNN, W^ℓ represents the set of filters/weight vectors in layer-ℓ, and $W^\ell[i]$ represents the ith filter/weight vector in layer-ℓ. Any sub-network fulfilling our above requirement can be defined and parameterized by an indicator vector M^ℓ per layer, whose each entry is a $\{0, 1\}$ value indicating whether retaining the associated filter/neuron in W^ℓ. We further define operator $\circ$ as

$$(W^\ell \circ M^\ell)[i] \triangleq \begin{cases} W^\ell[i], & M^\ell[i] = 1; \\ \vec{0}, & M^\ell[i] = 0. \end{cases} \tag{1}$$

Thereby, $\{M^\ell\}_{\ell=1:L}$ defines a qualified sub-network for inference chain and its weights are $\{W^\ell \circ M^\ell\}_{\ell=1:L}$, where we extend the operator $\circ$ to make $W \circ M = \{W^\ell \circ M^\ell\}_{\ell=1:L}$ given the original pre-trained DNN's weights $W = \{W^\ell\}_{\ell=1:L}$. Given a set of samples $\mathcal{X}$ drawn from a specific subtask, we can formulate the problem of finding an inference chain as the following combinatorial optimization, which aims to find the most sparse indicator M (i.e., the sub-network with the fewest filters retained) that does not change the outputs of the original pre-trained DNN for $\forall x \in \mathcal{X}$, i.e.,

$$\min_{\{M^\ell\}_{\ell=1:L}} \sum_{\ell=1}^{L} \|M^\ell\|_1 \text{ s.t. } F(x; W) = F(x; W \circ M), \ \forall x \in \mathcal{X}. \tag{2}$$

However, it is impractical to directly solve this combinatorial optimization since the possible choices for M^ℓ is of exponential number. We relax the 0-1 indicator vector M^ℓ to a nonnegative-valued score vector S^ℓ of

the same size. We define an operator $\odot$ applied to W^ℓ and its associated scores S^ℓ as

$$(W^\ell \odot S^\ell)[i] \triangleq S^\ell[i] \cdot W^\ell[i]. \tag{3}$$

Note we limit entries in S^ℓ within $[0, 1]$ due to the possible redundancy among filters in the original pre-trained DNN, i.e., there might be filters of similar functionality for the given samples and a preferred pruning should be able to only preserve one of them and multiply it by the number of those redundant filters in the sub-network. In addition, less constraints are easier to handle in optimization and helpful to find sub-network whose outputs are closer to that of the original pre-trained DNN, since the class of sub-networks with parameters $W \odot S$ includes all the sub-networks with parameters $W \circ M$. Hence, we relax the challenging combinatorial optimization to the following optimization, i.e.,

$$\min_{\{S^\ell\}_{\ell=1:L}} \frac{1}{|\mathcal{X}|} \sum_{x \in \mathcal{X}} l(F(x; W), F(x; W \odot S)) + \lambda \sum\nolimits_{\ell=1}^{L} \|S^\ell\|_1, \tag{4}$$

where $l(\cdot, \cdot)$ is a loss function aiming to minimize the distance between the original pre-trained DNN's output $F(x; W)$ and the sub-network's output $F(x; W \odot S)$. In our experiments, for classification, we use KL-divergence between the output distributions over classes, where the two output distributions are computed by applying softmax to $F(x; W)$ and $F(x; W \odot S)$ respectively, i.e.,

$$l(F(x; W), F(x; W \odot S)) = D_{KL}(\text{softmax}(F(x; W)) || \text{softmax}(F(x; W \odot S))). \tag{5}$$

In addition, empirical evidence (Krueger et al., 2017; Singh et al., 2016) show that for most samples there exist some layers that can be entirely removed without changing the final prediction. Hence, only a few hard and confusing samples need more delicate features, while most other samples can be correctly classified based on simple patterns from shallower layers. Therefore, in NeuroChains, we apply a sigmoid function with input score α^ℓ as a gate G^ℓ determining whether to remove the entire layer-ℓ during pruning, i.e.,

$$G^\ell = 1/\left[1 + \exp(-\alpha^\ell / T)\right], \tag{6}$$

where T is a temperature parameter. With a gate G^ℓ applied after each layer-ℓ whose input and output has the same size (which is common in many DNNs), we can recursively define the input $H^{\ell+1}(\cdot)$ to the next layer-$(\ell + 1)$, i.e., $H^{\ell+1}(x; \{W^{\ell'} \odot S^{\ell'}, \alpha^{\ell'}\}_{\ell'=1:\ell}) =$

$$\begin{cases} G^\ell \cdot F^\ell(H^\ell; W^\ell \odot S^\ell) + (1 - G^\ell) \cdot H^\ell(x; \\ \{W^{\ell'} \odot S^{\ell'}, \alpha^{\ell'}\}_{\ell'=1:\ell-1}) & \text{if input size = output size} \\ F^\ell(H^\ell; W^\ell \odot S^\ell) & \text{otherwise} \end{cases} \tag{7}$$

where $F^\ell(H^\ell; W^\ell \odot S^\ell)$ denotes the output of layer-ℓ. The reason to use a gate here is that we expect to either remove the whole layer or retain it without adding an extra shortcut (which will change the original pre-trained DNN's architecture). Since we prefer to remove non-informative layers, in the objective, we add another regularization α^ℓ to encourage the removal of entire layers (because decreasing α reduces G^ℓ and thus increase the chance of layer removal). Therefore, the final optimization for NeuroChains is

$$\begin{aligned} &\min\nolimits_{\{S^\ell, \alpha^\ell\}_{\ell=1:L}} \frac{1}{|\mathcal{X}|} \sum\nolimits_{x \in \mathcal{X}} l(F(x; W), H^{L+1}(x; \\ &\{W^{\ell'} \odot S^{\ell'}, \alpha^{\ell'}\}_{\ell'=1:L})) + \lambda \sum\nolimits_{\ell=1}^{L} \|S^\ell\|_1 + \lambda_g \sum\nolimits_{\ell=1}^{L} \alpha^\ell, \end{aligned} \tag{8}$$

Our objective above is similar to the one used in Network Slimming (Liu et al., 2017) but we optimize it for a subtask (so we can consider to remove layers) and we do not allow fine tuning on weights W.

3.2 Algorithm

Our algorithm is simply a standard back-propagation for the optimization problem in Eq. (8), which produces sparse scores for filters and gate values for layers. Note the weights in W are fixed and the backpropagation only updates S. We initialize the filter scores $S = \vec{1}$ so $W \odot S = W$ at the beginning of optimization. We initialize the gate score $\alpha^\ell = 0$ for all $\ell = 1 : L$ so $G^\ell = 0.5$ at the beginning, i.e., the probabilities to remove or to retain a layer is equal. For classification, we set loss $l(\cdot, \cdot)$ to be the KL-divergence between the output distributions of the original pre-trained DNN and the sub-network. After convergence of the optimization,

we then apply a simple thresholding to these scores to further remove more filters and layers: (1) we remove the filters with score under a threshold τ; (2) we remove layer-ℓ if $G^{\ell} < 0.5$. This yields a sufficiently small sub-network architecture. Given a sub-network produced by NeuroChains, we then visualize its architecture and scores as the structure of the inference chains. Moreover, for samples from the same region, we visualize their inference pathway from the original pre-trained DNN by their activation patterns and featuremaps, respectively, using existing interpretation methods (Zeiler & Fergus, 2014; Erhan et al., 2009).

4 Experiments

Statistics	ResNet-50	VGG-19
Top-1 test accuracy	76.5%	72.9%
Test images/sub-networks	10000/1688	10000/1746
Convolutional filters	26560	4480
Parameters of Conv-layers	23454912	20018880
Parameters of FC-layers	2048000	123633664

Table 1: Information of pre-trained DNNs in this paper.

In experiments, we apply NeuroChains to extract the inference chains of two widely-adopted CNNs pre-trained on ImageNet, i.e., VGG-19 and ResNet-50. We provide the basic information of the two DNNs in Table 1. In the following, we will present **two quantitative analyses over hundreds of case studies**, which show that (1) NeuroChains is capable to produce sub-networks retaining only $< 5\%$ of filters and meanwhile preserve the outputs of the original pre-trained DNN in most cases; (2) the filters selected by NeuroChains with high scores are important to preserving the outputs since removing one will lead to considerable drop in performance. We also compare the capability of preserving the original neural network's outputs between NeuroChains and magnitude-based pruning and random pruning methods. We will then provide several detailed and insightful case studies and visualizations of extracted sub-networks for different subtasks.

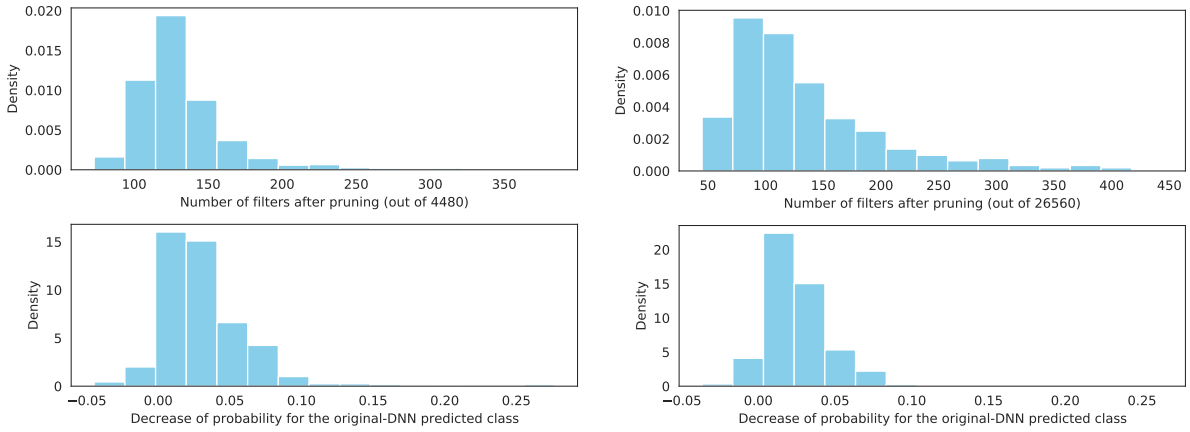

Figure 1: *Size and fidelity (how well the sub-networks preserve the original pre-trained DNN's outputs)* of 1500 sub-networks extracted by NeuroChains for VGG-19(left) and ResNet-50(right) in different case studies (tasks). The x-axis in the top plot refers to the number of retained filters, while the x-axis in the bottom plot is the decrease of probability on the original-DNN predicted class. It shows that NeuroChains can usually find very small sub-networks and meanwhile preserve the original pre-trained DNN's outputs.

4.1 Implementation Details

We implement NeuroChains by PyTorch (Paszke et al., 2017). In every case study, we firstly randomly sample 2 classes in ImageNet and then randomly sample 10 images from each class's images. Note that

the sampled images may be wrongly classified to other classes by the original DNN. We apply inference on those 20 images and their outputs are used in solving the optimization of Eq (8) in order to extract the local inference chain in the form of a sub-network. For models with shortcuts, e.g., ResNet-50, the sigmoid gate is applied to prune a bottleneck block rather than a layer. A layer inside a block will be removed if the scores of all filters in the layer are nearly 0.

We use Adam optimizer for the optimization of Eq (8) for filter/layer scores. We use a fixed learning rate of 0.005. We set temperature $T = 0.2$ in sigmoid gate (Eq. (6)) to encourage the value G^ℓ close to either 0 or 1, and the threshold τ to goal scores is set to 0.1 so that the outputs of sub-networks are consistent. We only tried a limited number of choices on tens of experiments, and chose the best combination balancing the fidelity and sub-network size, and then applied it to all other experiments without further tuning. In particular, we tried $\tau \in \{0.01, 0.1, 0.5\}$, $\lambda \in \{0.001, 0.005, 0.01, 0.1\}$, and $\lambda_g \in \{1, 2, 5\}$. For different models, the weights of two penalties in Eq. (8) are different. For VGG-19, we use $\lambda = 0.005$ and $\lambda_g = 2$. While we choose $\lambda = 0.005$ and $\lambda_g = 1$ for ResNet-50. This choice performs consistently well and robust on all other experiments. The iteration steps of training is 300 and we stop training when the loss difference is quite small, i.e., less than 0.05. It costs only $\sim$ 90s for VGG-19 and $\sim$ 55s for ResNet-50 to extract a sub-network on a single RTX 6000 GPU since we only optimize a few number of scores.

4.2 Quantitative Analyses

Melis & Jaakkola (2018) propose some criteria to evaluate the interpretation methods for DNNs. In this paper, we extend some of their notations and present two quantitative analyses of NeuroChains over 1500 case studies for different subtasks, i.e., (1) **Fidelity**: does the sub-network preserve the original pre-trained DNN's outputs on the given samples? how does it change for sub-networks of different sizes? (2) **Faithfulness**: how well does the importance score of a filter reflect the degeneration on the fidelity caused by removing the filter from the sub-network? In this paper, we evaluate the fidelity and faithfulness by the decreasing amount of probability on the original pre-trained DNN's predicted class when using the sub-network for inference. All the above metrics are averaged over 1500 sub-networks and across all the images used to extract each sub-network.

Figure 1 shows the statistics of the fidelity for sub-networks of different sizes (measured by the number of filters) that are extracted by NeuroChains. Most sub-networks only retain $\leq 1\%(\leq 5\%)$ of ResNet-50(VGG-19) for succinct visualization but they preserve the outputs of ResNet-50(VGG-19) with high fidelity.

Figure 2 reports the faithfulness of extracted sub-networks, i.e., how a sub-network's performance in preserving the original pre-trained DNN's output degrade if removing one filter from it, and what is the relationship between this degeneration and the score of the removed filter. The statistics on 1500 sub-networks in Figure 2 show that as the scores of filters increase, the fidelity degrades more. The degeneration and the score are strongly and positively correlated, indicating that our optimized scores faithfully reflect the importance of filters in explaining the original pre-trained DNN. Moreover, removing even only one highly scored filter from the sub-network can significantly degrade the explanation performance. Hence, NeuroChains usually find the smallest sub-networks without redundancy among retained filters/layers, i.e., every critical inference step is retained.

We also present another faithfulness study based on a quotient metric defined below. Let $p, q \in \Delta^c$ (Δ^c is the probability simplex for c classes) be the output probability vectors of the original neural net and the extracted sub-network respectively for same input. We define a quotient metric to measure the change of class prediction between p and q, i.e.,

$$Q(p, q) = \frac{q[y] - \max_{z \in [c], z \neq y} q[z]}{p[y] - \max_{z \in [c], z \neq y} p[z]}, \quad y \in \arg\max_{z \in [c]} p[z], \tag{9}$$

where y is the predicted class by the original neural net, and $Q(p, q)$ is the quotient of two probability differences computed respectively on the original neural net and the sub-network. In particular, it computes the difference of probabilities for class y and the highest-rated other class. The sign of $Q(p, q)$ indicates whether the predicted class changes (e.g., it changes if $Q(p, q) < 0$) while the magnitude of $Q(p, q)$ measures the change in prediction confidence. The result is consistent with our above observations and is given in Figure 7 of Appendix.

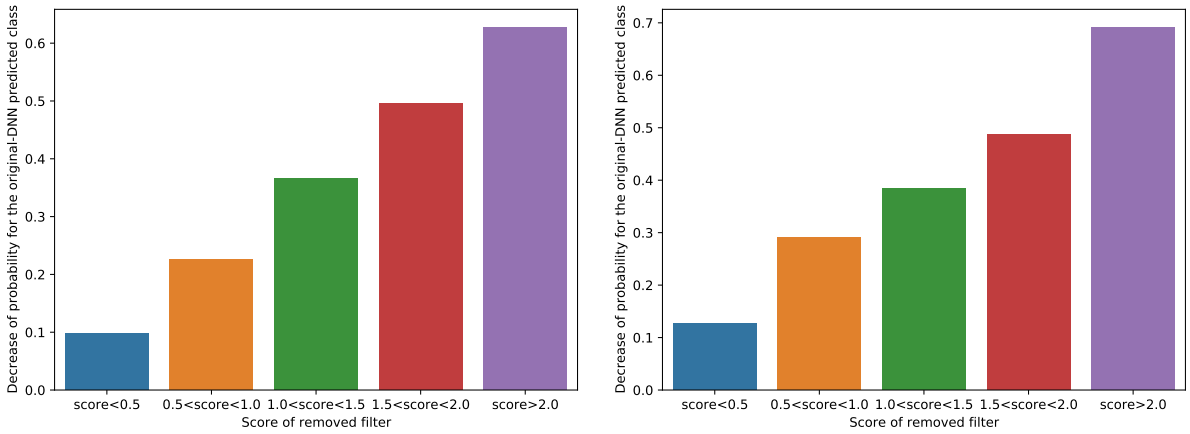

Figure 2: *Barplot of the faithfulness (filter score vs. fidelity degeneration caused by removing the filter)* of 783 sub-networks, each extracted by NeuroChains on 20 uniform samples randomly drawn from two classes for VGG-19(left) and ResNet-50(right). The x-axis refers to the interval of filter scores, while the y-axis denotes the decrease of the sub-network's probability on the original-DNN predicted class after removing a filter from the sub-network. It shows that the sub-networks suffer from more degeneration if removing a filter with higher score. Hence, the scores faithfully reflect the importance of filters in explaining the original pre-trained DNNs.

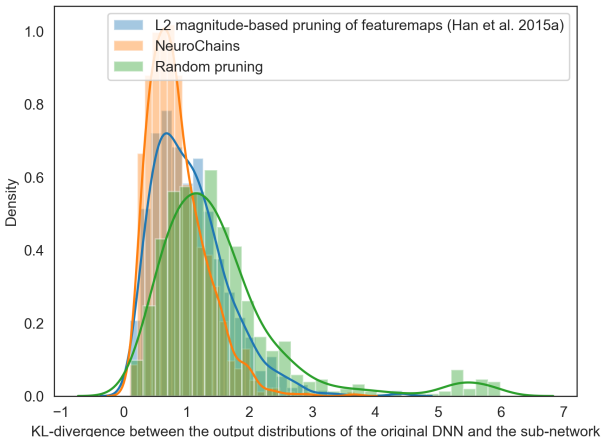

Figure 3: Comparison of different pruning methods on the capability of preserving the original network's output distribution (smaller KL divergence means better preservation) over 783×20 uniform samples.

We compare the capability of preserving the original neural network's outputs between NeuroChains and magnitude-based pruning (removing the filters whose output featuremaps' average magnitude (L2 norm) over all considered samples is small) and random pruning ("by chance") in Figure 3. In particular, under the same setting of each experiment in the paper, we prune the original VGG-19 and retain the filters with the largest featuremap magnitude in each layer or randomly, 180 in total (more than $157(mean) \pm 43(std)$ filters for sub-networks extracted by NeuroChains), and we then fine-tune the filters' scores. For Figure 3, the percentage of cases with KL-divergence$\leq$ 0.8 for NueroChians, magnitude-based pruning and random pruning are **54.3%, 38.8%, 21.1%** respectively, and **9.3%, 19.4%, 38.8%** cases have KL-divergence$\geq$ 1.5. Figure 3 shows the histogram of the KL divergence between the original output class distribution and the one produced by the sub-networks. For sub-networks generated by NeuroChains, the KL-divergence in most cases stays close to 0, while the output preserving capability of simple pruning is much worse. More complete quantitative analyses for both VGG-19 and ResNet-50 can be found in the Appendix.

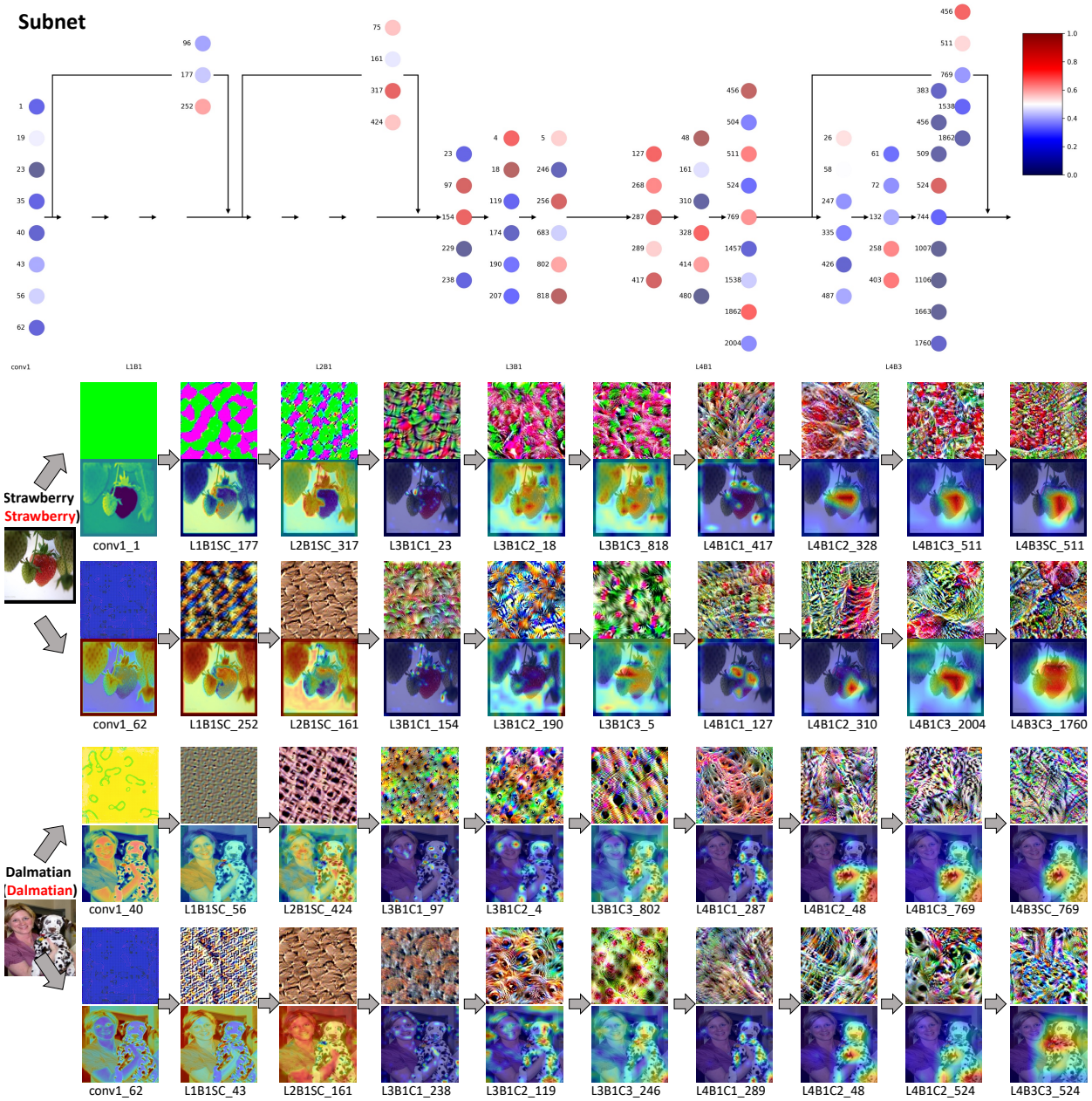

Figure 4: Inference chain by NeuroChains for ResNet-50 (pre-trained on ImageNet) when applied to 20 test images of "strawberry" and "dalmatian". The sub-network retains only 13/67 layers and 77/26560 filters of the ResNet-50. Refer to Section 4.4 for detailed analysis of extracted chains.

4.3 Case Studies

We present three case studies of the sub-networks extracted by NeuroChains. For Figure 4 and Figure 5, the data points are from classes which are easy to tell apart while in Figure 6 images are sometimes mis-classified. The case study of multi-classes classification task is shown in Figure 13 in Appendix. The visualization in each case study is composed of two parts: (1) the sub-network's architecture and filter scores; (2) original images from each class and the visualization of the image's NeuroChains extracted from the original pre-trained model. The true class and predicted class of the sample are shown above the image. They show

that NeuroChains considerably enrich the explanation details of DNN's inference process. By connecting the important filters from different layers, the extracted sub-network highlights the main steps leading to the output prediction.

On the sub-network's architecture, we use "L0" to denote the corresponding convolution layer in VGG-19 and "L0_1" to denote the first filter from this layer. For ResNet-50, we further use "L1B1" to denote the first sub-block in the first bottleneck block, "SC" for the shortcut connection and "C1" for the first convolution layer in the sub-block. The redder the node in the sub-network, the larger the scaling score, conversely, the bluer the node, the lower the score.

NeuroChains are stitched by filters step by step and the visualization of each filter and the corresponding featuremap generated by the original pre-trained model are displayed in each step. The visualization of each selected filter is achieved by maximizing its activation w.r.t. the input. Afterward, we shows the patterns that the filter aims to detect which is independent of the input image. More case studies are given in Appendix.

4.4 Detailed Analysis of Case Studies

In Figure 4 and Figure 5, the case studies of VGG-19 and Resnet-50 are shown. For each sample from the subtask, two NeuroChains which present the inference process of the original pre-trained model are displayed.

In Figure 4, features extracted by the chains of strawberry gradually evolve from low-level to high-level, e.g., in the first chain, L1B1SC_177, L2B1SC_317, and L3B1C1_23 extract the low-level patterns of red-green color and texture, L3B1C2_18, L3B1C3_818, and L4B1C1_417 look like the abstract pattern of strawberry, and L4B1C2_328, L4B1C3_511 and L4B1SC_511 capture the pattern of the skin of strawberry. While the first chain of strawberry focus on the skin, the second chain extracts the pattern of the leaves, i.e., L3B1C1_154 and L3B1C3_5. The leaves in the featuremaps of L4B1C1_127 and L4B1C3_2004 are highlighted. L4B3C3_1760 extract the global patterns of strawberry. Both the two chains of dalmatian extract the global patterns of dalmatian's black and white fur. In shallow layers, L3B1C1_97, L3B1C1_238, L3B1C2_4, and L3B1C2_119 capture a more local black spot pattern of the dalmatian while L4B1C1_289, L4B1C2_48, L4B1C3_769 and L4B3C3_524 reveal the global patterns.

In Figure 5, the featuremaps show that the first chain of kangaroo extracts the pattern of eyes and noses, while the second chain pays more attention to the fur of kangaroos. L21_296, L23_393, L32_453, and L34_66 turn from a round pattern to an eye and nose pattern. L21_24, L32_463, and L34_188 look like the fur pattern of animals. In the first chain of bananas, L19_335, L21_111, L23_470, and L32_465 turn from the curved rectangle pattern to a hand of bananas. The L32_458 and L34_79 in the second chain of bananas seem like the pattern of toes and roots of bananas. The corresponding areas in the featuremaps are also highlighted. The two chains focus on bananas' global and local patterns, respectively.

In Figure 6, we first show the NeuroChains of correctly predicted castle and stone wall. Then, using the same filters from the extracted chains, we draw the featuremaps of a stone wall which is wrongly predicted as the castle by the original pre-trained model, to uncover the inner mechanism of false prediction. The featuremaps of the wrongly predicted stone wall from filters activated by castle show that the featuremaps in the mid-part of the chains are rarely activated, like L21_300 and L23_336, which is in line with our expectations. But in the deep layers, L32_15 and L32_45, which activate the roof and body of the castle, also highlight a wide area in the wrongly predicted stone wall image. This indicates that the wrongly predicted stone wall sample contains similar features as castle. The featuremaps of the wrongly predicted stone wall from filters activated by stone wall (correctly predicted) also shows the anomaly. L21_197, L25_26, and L32_260, which are critical to the correctly predicted stone wall, extract little information from the wrongly predicted stone wall. We find that L21_124 and L32_15 are both important to identify the castle and stone wall, which implies that castle and stone wall have similar patterns and are easily confused.

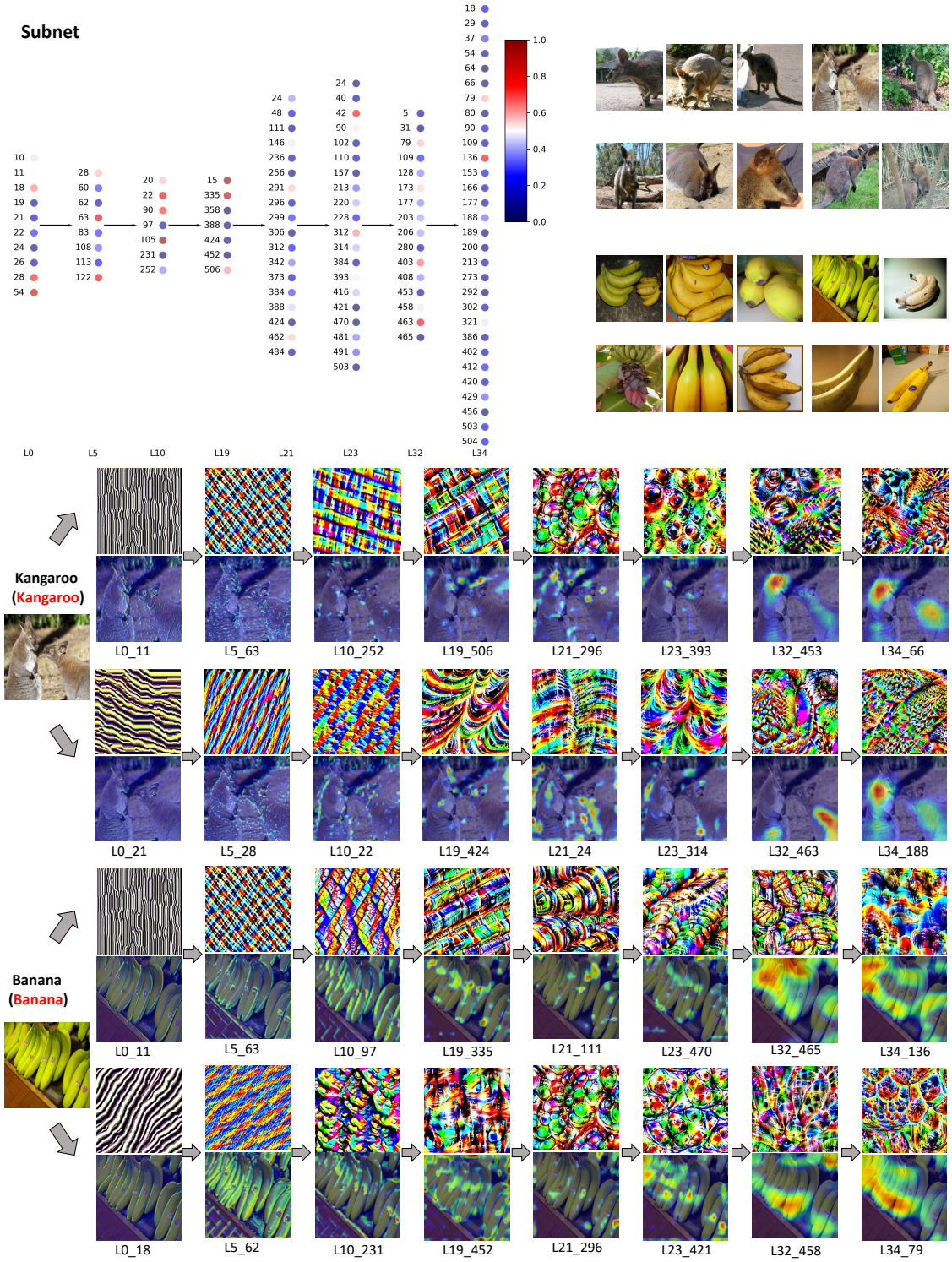

Figure 5: Inference chain by NeuroChains for VGG-19 when applied to images of “kangaroo” and ”banana”. The sub-network retains only 8/16 layers and 116/4480 filters of the VGG-19. The scores for selected filters are represented using the colormap on the top-right. Refer to Section 4.4 for detailed analysis of extracted chains.

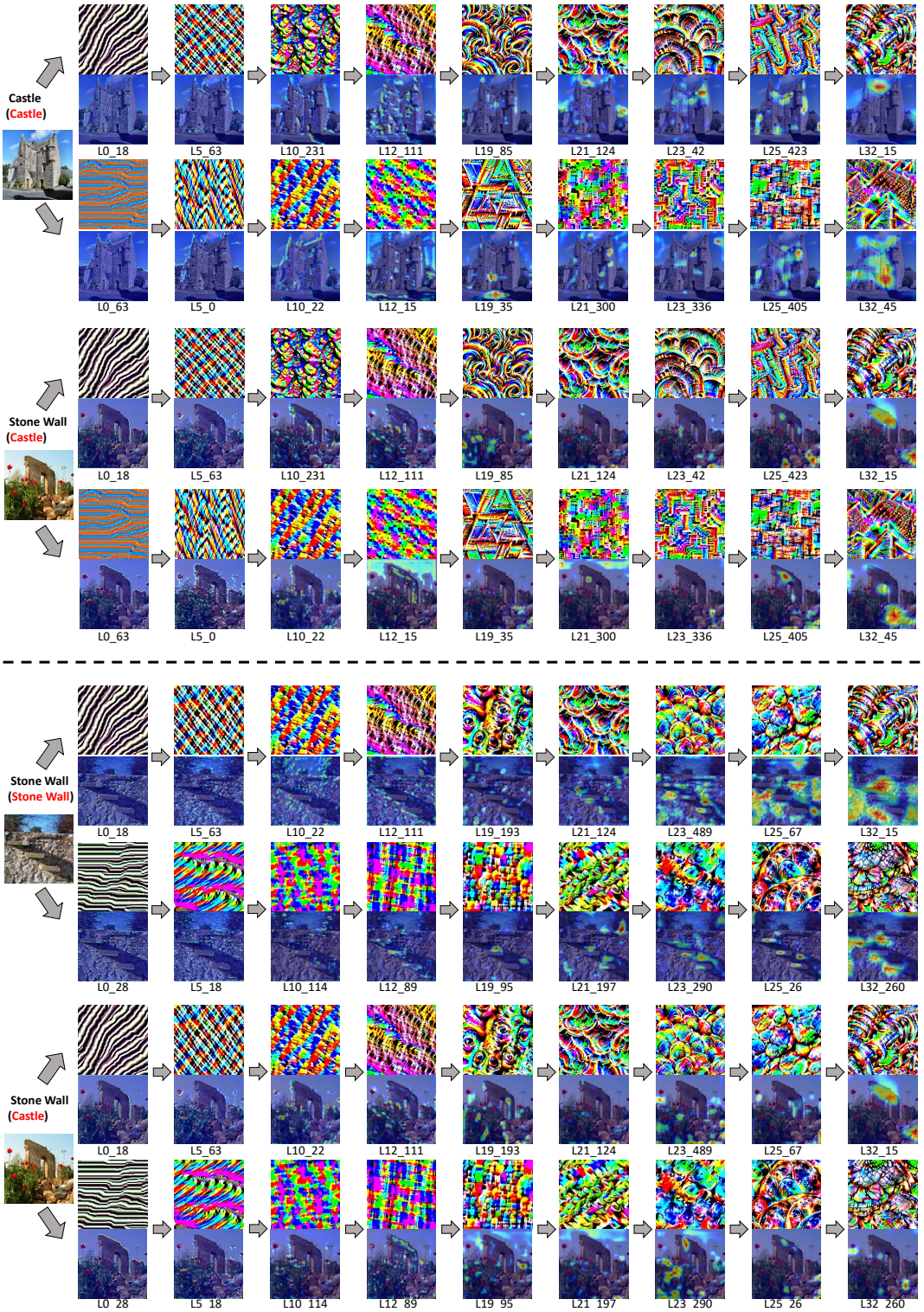

Figure 6: Inference chain by NeuroChains for VGG-19 when applied to images of "Castle" and "Stone Wall" which confuse DNN models and may be mis-classified. The sub-network retains only 10/16 layers and 121/4480 filters of the VGG-19. Due to space limitation, the subnet can be found in Appendix. Refer to Section 4.4 for detailed analysis of extracted chains.

5 Conclusion

We propose an efficient approach NeuroChains to extract the main inference chains/steps in a DNN when applied to a subtask. We learn a sparse scoring of filters/layers to extract a sub-network retaining a subset of filters/layers from the original pre-trained DNN. We provide case studies and quantitative analyses to demonstrate that NeuroChains produces more informative and reliable explanation to the inner inference process of DNNs. Since local reasoning chain extraction is a novel and challenging problem even for classification, we will investigate more complicated tasks like regression and detection in the future work.

References

Marco Ancona, Enea Ceolini, Cengiz Öztireli, and Markus Gross. A unified view of gradient-based attribution methods for deep neural networks. In *NIPS 2017-Workshop on Interpreting, Explaining and Visualizing Deep Learning*. ETH Zurich, 2017.

Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.

Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1, 2009.

Utku Evci, T. Gale, Jacob Menick, P. S. Castro, and E. Elsen. Rigging the lottery: Making all tickets winners. *ArXiv*, abs/1911.11134, 2020.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. 2018.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Zichao Guo, X. Zhang, H. Mu, Wen Heng, Z. Liu, Y. Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In *ECCV*, 2020.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pp. 1135–1143, 2015.

Bo-Jian Hou and Zhi-Hua Zhou. Learning with interpretable structure from rnn. *arXiv preprint arXiv:1810.10708*, 2018.

Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

Pieter-Jan Kindermans, Kristof T Schütt, Maximilian Alber, Klaus-Robert Müller, Dumitru Erhan, Been Kim, and Sven Dähne. Learning how to explain neural networks: Patternnet and patternattribution. *arXiv preprint arXiv:1705.05598*, 2017.

David Krueger, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Hugo Larochelle, Aaron C. Courville, and Chris Pal. Zoneout: Regularizing rnns by randomly preserving hidden activations. In *International Conference on Learning Representations*, 2017.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019a. URL `https://openreview.net/forum?id=S1eYHoC5FX`.

Z. Liu, H. Mu, X. Zhang, Zichao Guo, X. Yang, K. Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 3295–3304, 2019b.

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2736–2744, 2017.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pp. 5058–5066, 2017.

A. Madry, Aleksandar Makelov, Ludwig Schmidt, D. Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *ArXiv*, abs/1706.06083, 2018.

David Alvarez Melis and Tommi Jaakkola. Towards robust interpretability with self-explaining neural networks. In *Advances in Neural Information Processing Systems*, pp. 7775–7784, 2018.

Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern Recognition*, 65: 211–222, 2017.

T Nathan Mundhenk, Barry Y Chen, and Gerald Friedland. Efficient saliency maps for explainable ai. *arXiv preprint arXiv:1911.11293*, 2019.

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

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pp. 4095–4104, 2018.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144. ACM, 2016.

Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 3145–3153, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL `http://proceedings.mlr.press/v70/shrikumar17a.html`.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

Saurabh Singh, Derek Hoiem, and David Forsyth. Swapout: Learning an ensemble of deep architectures. In *Advances in Neural Information Processing Systems 29*, pp. 28–36. 2016.

Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825*, 2017.

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

Jingtong Su, Yihang Chen, Tianle Cai, Tianhao Wu, Ruiqi Gao, L. Wang, and J. Lee. Sanity-checking pruning methods: Random tickets can win the jackpot. *ArXiv*, abs/2009.11094, 2020.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 3319–3328. JMLR. org, 2017.

Mike Wu, Michael C Hughes, Sonali Parbhoo, Maurizio Zazzi, Volker Roth, and Finale Doshi-Velez. Beyond sparsity: Tree regularization of deep models for interpretability. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. NAS-bench-101: Towards reproducible neural architecture search. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pp. 7105–7114, 2019.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pp. 818–833. Springer, 2014.

Luisa M Zintgraf, Taco S Cohen, Tameem Adel, and Max Welling. Visualizing deep neural network decisions: Prediction difference analysis. *arXiv preprint arXiv:1702.04595*, 2017.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. 2017. URL `https://arxiv.org/abs/1611.01578`.

A Appendix

A.1 Quantitative Analysis

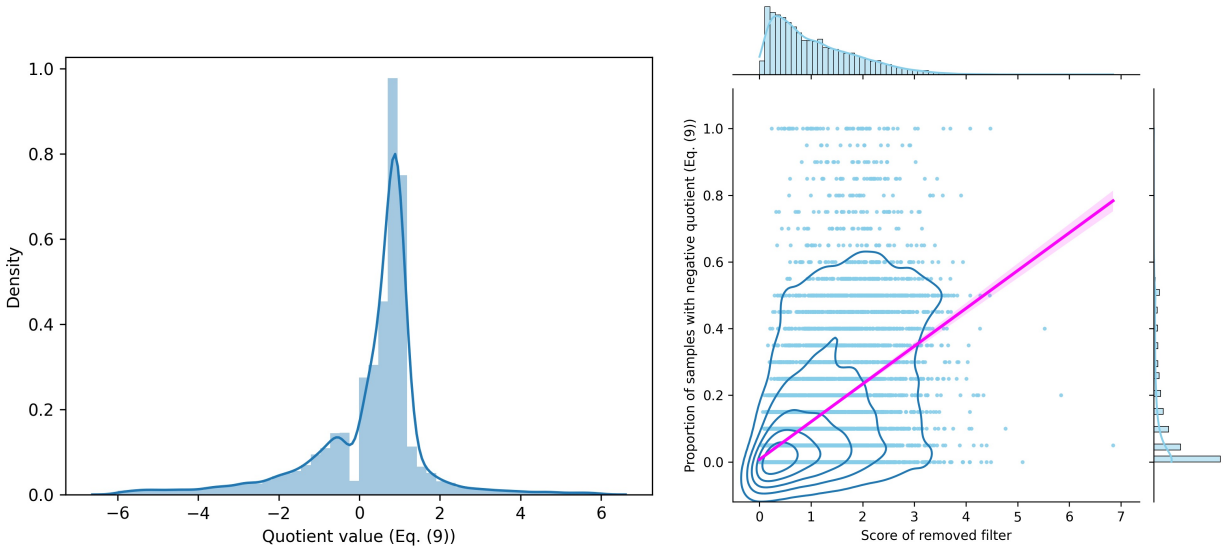

Figure 7: **Left:**Histogram of the quotient metric in Eq. (9) computed over all the 783×20 samples. **Right:**Faithfulness of NeuroChains in terms of the quotient's sign.

We performed 783 experiments each using 20 samples uniformly drawn from two classes. We evaluated these newly generated sub-networks using the quotient metric "A quotient of "diff to highest scoring other class (extracted)" / "diff to highest scoring other class (original)" Eq. (9). We visualized the result in Figure 7: The left plot is the histogram of the quotient computed over all the 783×20 samples. The histogram shows that most samples keep the original predicted label after pruning, i.e., NeuroChains can preserve the original pre-trained DNN's outputs in most cases. Moreover, the number of filters preserved in these sub-networks is $157(mean) \pm 43(std)$, which is small enough to explain.

The right plot reports the Faithfulness of NeuroChains in terms of the quotient's sign. We remove each filter from each sub-network and report how many samples' predicted labels are changed after the removal, i.e., the

quotient is negative. Each point in the scatter plot corresponds to a sub-network, the x-axis is the score of the removed filter given by NeuroChains, and the y-axis is the proportion of samples with negative quotients. The plot shows a strong linear correlation between the score of the removed filter and the degradation of faithfulness. Since removing filters with high scores results in more samples with predicted class changing after pruning, the score given by NeuroChains measures the importance of filters in DNN inference.

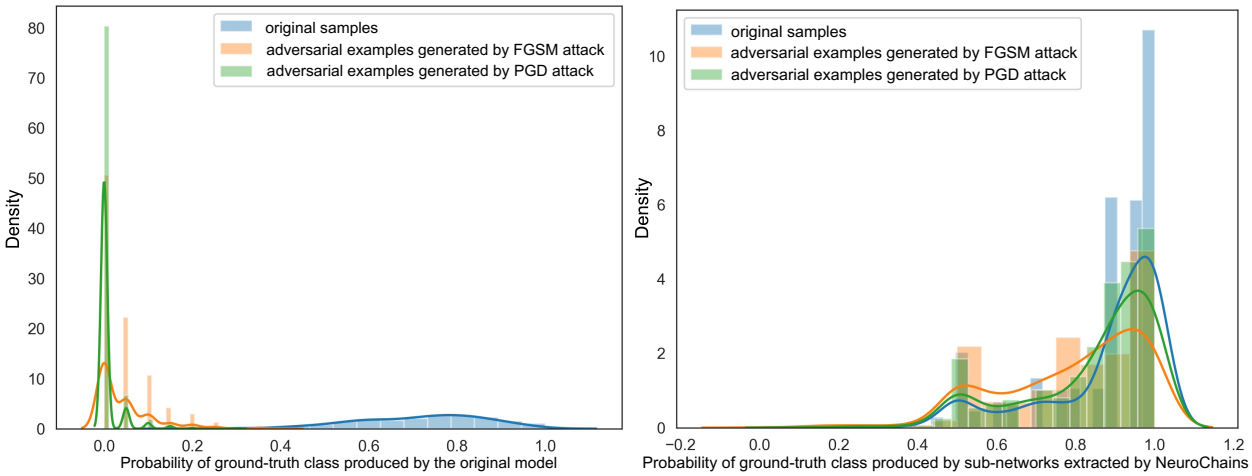

Figure 8: Robustness of original model (Left) vs. extracted sub-networks (Right) under different attacks.

In order to evaluate NeuroChains on the subtasks in the raw input space, we extract sub-networks for uniformly random drawn samples and then evaluate the sub-networks on these samples' adversarial examples generated by two types of attacks: fast gradient sign method (FGSM) (Goodfellow et al., 2014) and projected gradient descent (PGD)(Madry et al., 2018). Figure 8 compares the robustness of the original neural net (Left plot) and the extracted sub-networks (Right plot) under different attacks: each of the plots show the histogram of the output probability for the ground-truth class on those samples (original and adversarial). The left plot shows that the two types of adversarial attack are very effective on the original neural net in reducing the probability of ground truth class. In contrast, the right plot shows that the NeuroChains extracted sub-networks are much more robust to the attacks, because the optimization in NeuroChains not only removes the irrelevant filters but also strengthens the important filters by assigning them weights > 1. This demonstrates the effectiveness of NeuroChains when applied to subtasks with non-smooth raw-input space, and the extracted sub-networks in this case significantly improves the robustness of the original model in defending adversarial attacks.

In Figure 9, we show two case studies of comparing SMOE (Mundhenk et al., 2019) generated heatmaps for the original network and the NeuroChains extracted sub-network. We can see that the patterns extracted by the two networks are consistent and are all critical patterns for the class, e.g., the eyes and fists of kangaroos and the feet and face of the horse. However, compared with the original network, these patterns are strengthened in much shallower layers of the sub-network, producing better interpretations. This observation is also consistent with the result of analysis on adversarial attacks in Figure 8.

A.2 Case Studies

In Figure 10, the 20 images from 2 classes used to extract the sub-network of the case study in Figure 4 are displayed. All the NeuroChains are extracted from the original pre-trained model based on a few data of a sub-task. The extracted sub-network and 20 images from 2 classes used to extract the sub-network of the case study in Figure 6 are shown in Figure 11.

NeuroChains can also be used to extract the sub-networks for tasks with multi-classes, and the extracted sub-network is still small enough to be analyzed and explained. In Figure 12 and Figure 13, the sub-network and the NeuroChains of a 3-classes sub-task are presented. The sub-network contains 8 layers and 158 filters, which is easy to visualize. The NeuroChains corresponding to the images in the three classes are displayed.

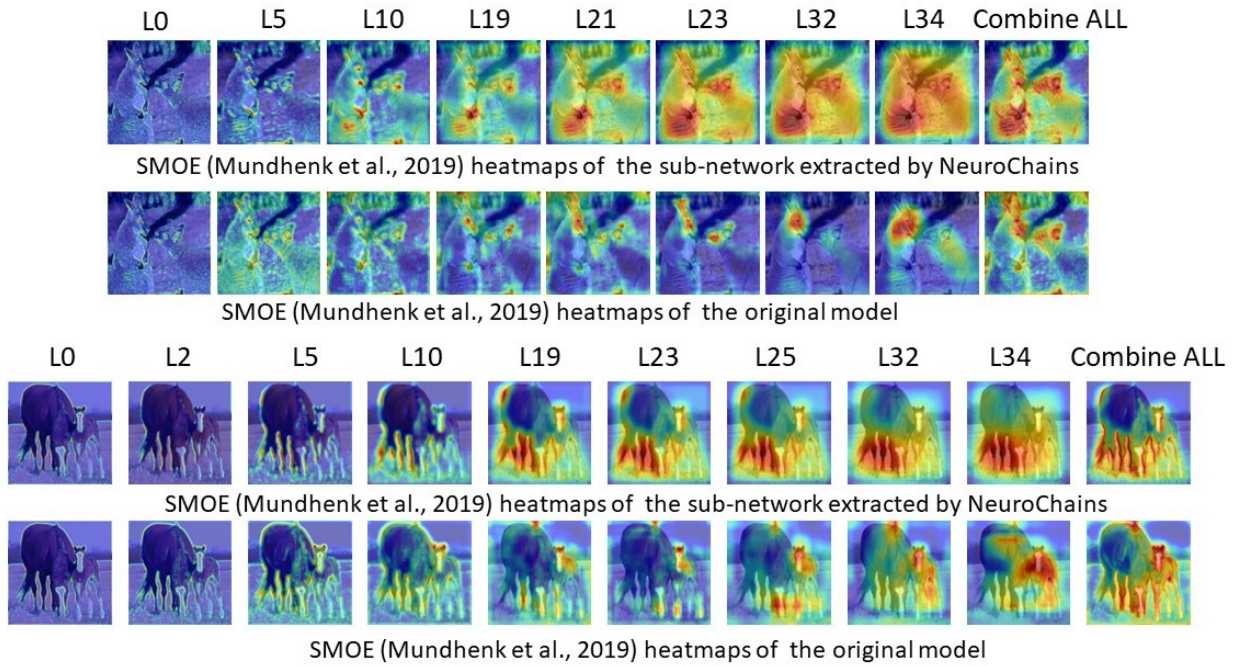

Figure 9: Case studies of SMOE generated heatmaps for the original network and the NeuroChains extracted sub-network.

Figure 10: The 20 images from 2 classes used to extract the sub-network of case study in Figure 4.

The chains of orangutans extract features from different parts of the orangutan. The top chain of orangutan extracts the pattern in the face. L23_82 and L25_27 show the round and black node pattern. Then in L30_42 and L32_453, these nodes evolve into the pattern of eyes and noses. In L34_112, these features are integrated into the entire face of orangutan. The second chain of orangutan mainly focus on the fur and shape of arms. In L25_220, L30_439 and L32_376, the elbow part in the featuremap is highlighted, and the visualization of filters also show the pattern of curved shape of the joint part. In the first chain of tram, the pattern of the arranged windows are extracted in L25_202, L30_329 and L32_491 and the windows in the featuremap are highlighted. From the featuremap of the second chain, the bottom part of tram is highlighted and the pattern in L32_5 seems like wheels, but the visualization of other filters is not readable, which indicate that this visualization method still has limitations. In the chains of broccoli, the head part is extracted by the first chain while the stem part is captured by the second chain. From layer 10 to layer 34 of the first chain, L19_482, L23_83 and L25_247 show the pattern of dense semisphere, and L30_440, L32_281 and L34_428 look like the pattern of the head of broccoli. In the second chain, the pattern of closely arranged columns are shown in L30_421, L32_388 and L34_323, which is consistent with the shape of the stem of broccoli. From these analysis, we can find that despite the small number of filters in the sub-network, for samples from different classes, more than one critical patterns are captured to identify the samples. It shows the strong expression ability of neural network and the interpretation ability of NeuroChains.

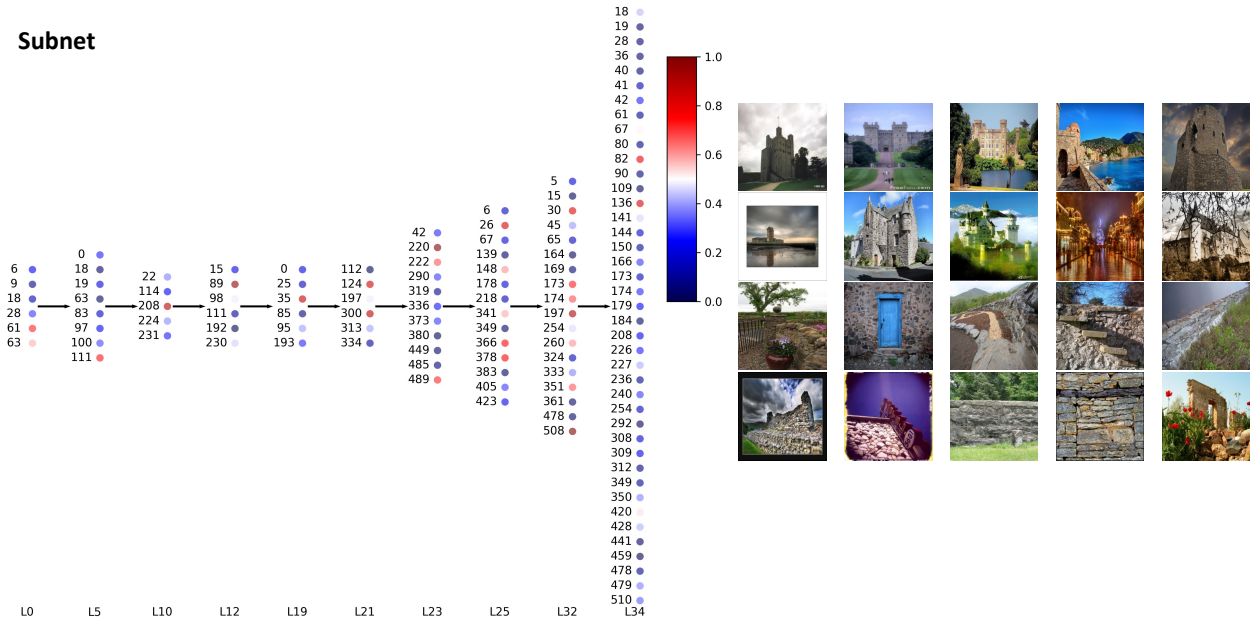

Figure 11: The extracted sub-network and 20 images from 2 classes used to extract the sub-network of case study in Figure 6.

In Figure 14, another case study to classify pineapple and leopard on Resnet-50 is shown. In the first chain, the shallow layers extract low-level patterns like edges in L3B1C2_162. L4B1C2_485, L4B1C3_384 and L4B3SC_925 from deep layers evolve these low-level patterns into the crown. In the second chain, the mid-part of the network like L3B1C1_45, L3B1C2_10, L3B1C3_541 and L4B1C1_308 show the pattern of the skin of the pineapple. Filters in the last block, i.e. L4B1C2_4, L4B1C3_1010, and L4B3SC_1106 extract the global pattern which look like the main body of pineapples. In the chains of leopard, the body and head of the leopard are lit up respectively. But the visualization of filters in the 2 chains seem similar. The skin marked with black spots is its most obvious pattern like L3B1C2_221, L4B1C1_274, L3B1C3_857 L4B1C3_462 and L4B1C3_769.

In Figure 15, a case study to distinguish indigo bunting from horse on VGG-19 is displayed. The chains of indigo bunting extract the local head pattern and global pattern of indigo bunting respectively. In the first chain, the beak and eye are lit up by L19_497, L23_375 and L25_199. L32_352 and L34_140 integrate previous pattern to the whole head. L19_44, L23_497 and L25_81 in the second chain extract the pattern of feathers. In L32_178 and L34_483, the pattern may also introduce patterns in the first chain and the global pattern including the head are lit up. Similar to indigo bunting, the chains of horse extract the head pattern and body pattern of horse respectively. L32_94 and L34_509 show the pattern like striped muscles. The whole back of the horse is highlighted. The second chain focus on the head part of the horse. L25_317 captures the mouth while L32_30 and L34_156 captures the eye. This case study shows that for some samples like indigo bunting, the global pattern is learnd by the model to identify the samples. For samples like horse, some local patterns like the body and eyes are enough.

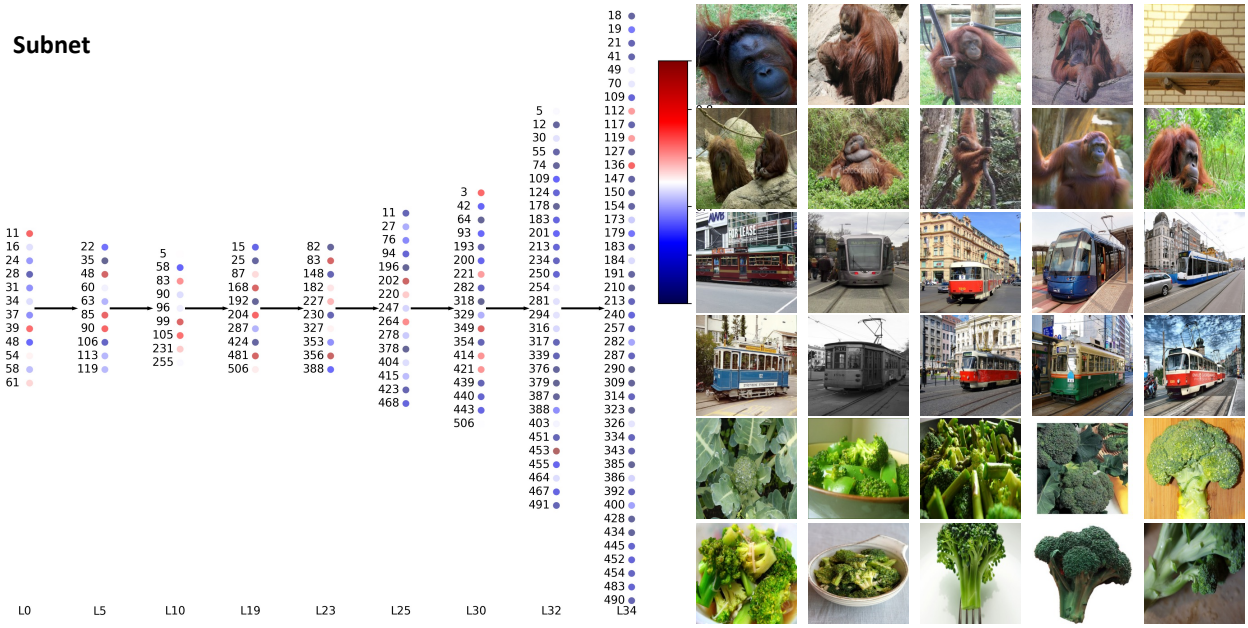

Figure 12: The extracted sub-network and 30 images from 3 classes used to extract the sub-network of case study in Figure 13.

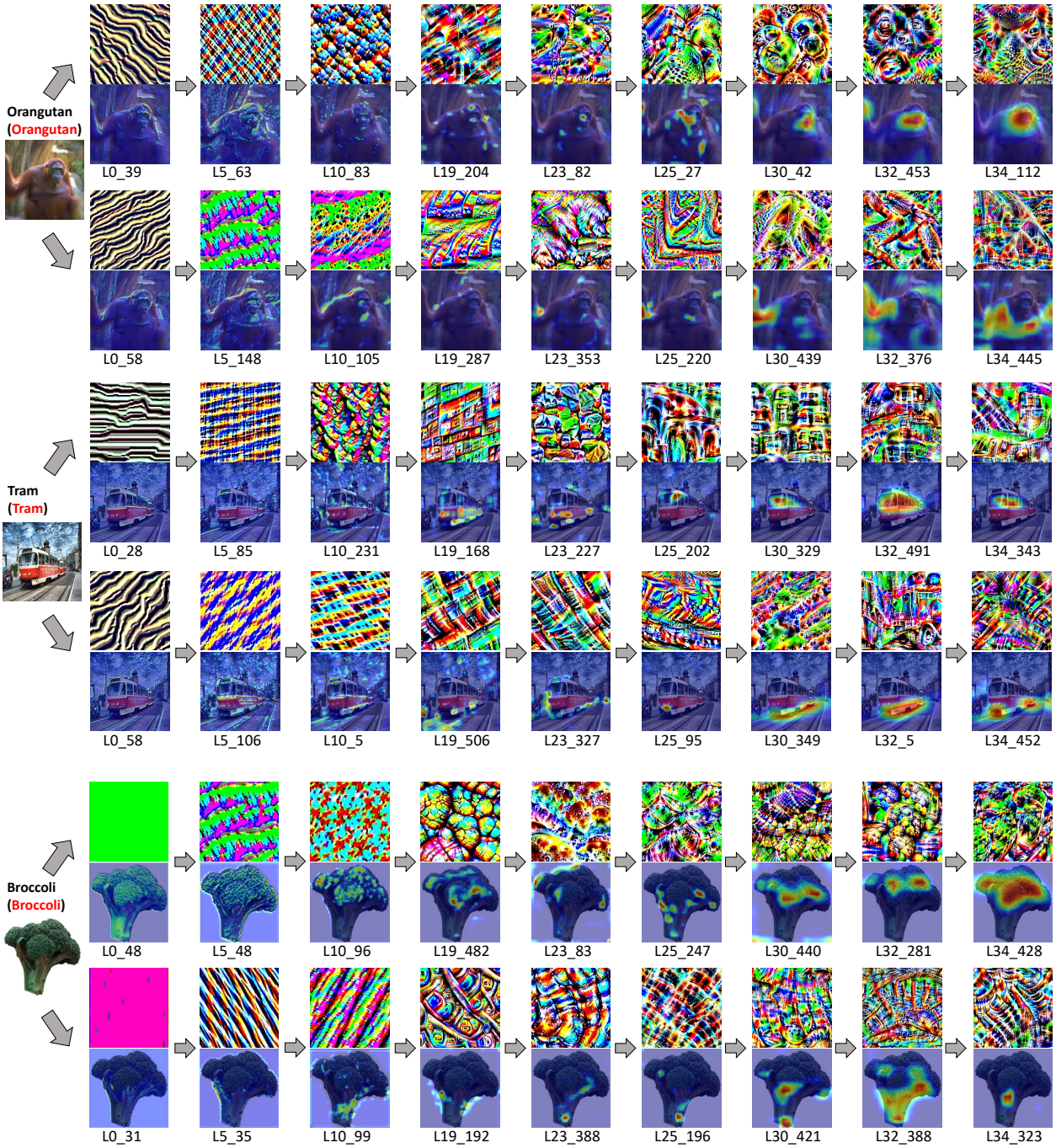

Figure 13: Inference chain by NeuroChains for VGG-19 when applied to images of ‘orangutan”, ”tram” and ”broccoli”. The sub-network retains only 8/16 layers and 158/4480 filters of the VGG-19 is shown in Figure 12. Refer to Section A.2 for detailed analysis of extracted chains.

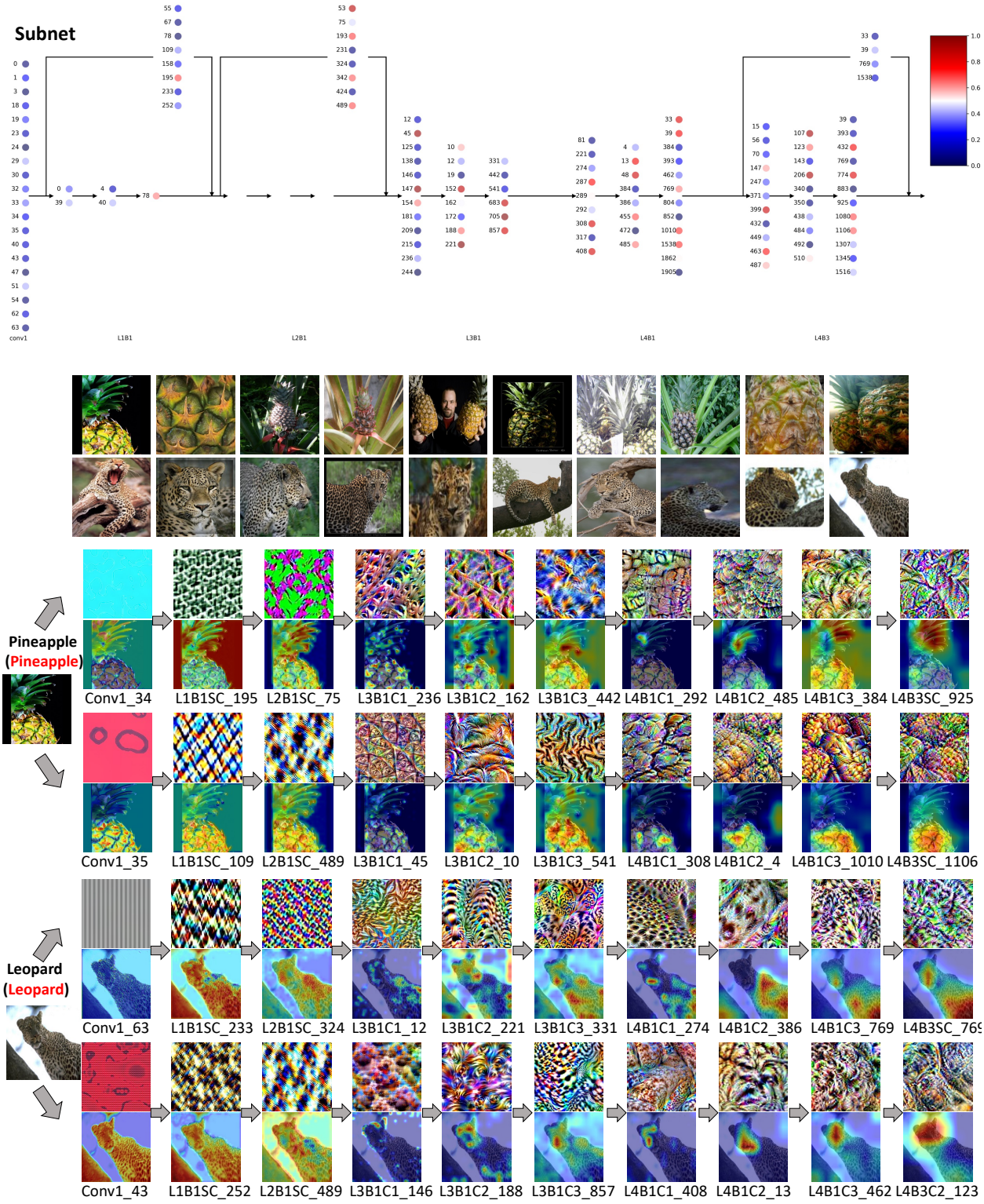

Figure 14: Inference chain by NeuroChains for ResNet-50 (pre-trained on ImageNet) when applied to 20 test images of “pineapple” and ”leopard”. The sub-network retains only 16/67 layers and 133/26560 filters of the ResNet-50. Refer to Section A.2 for detailed analysis of extracted chains.

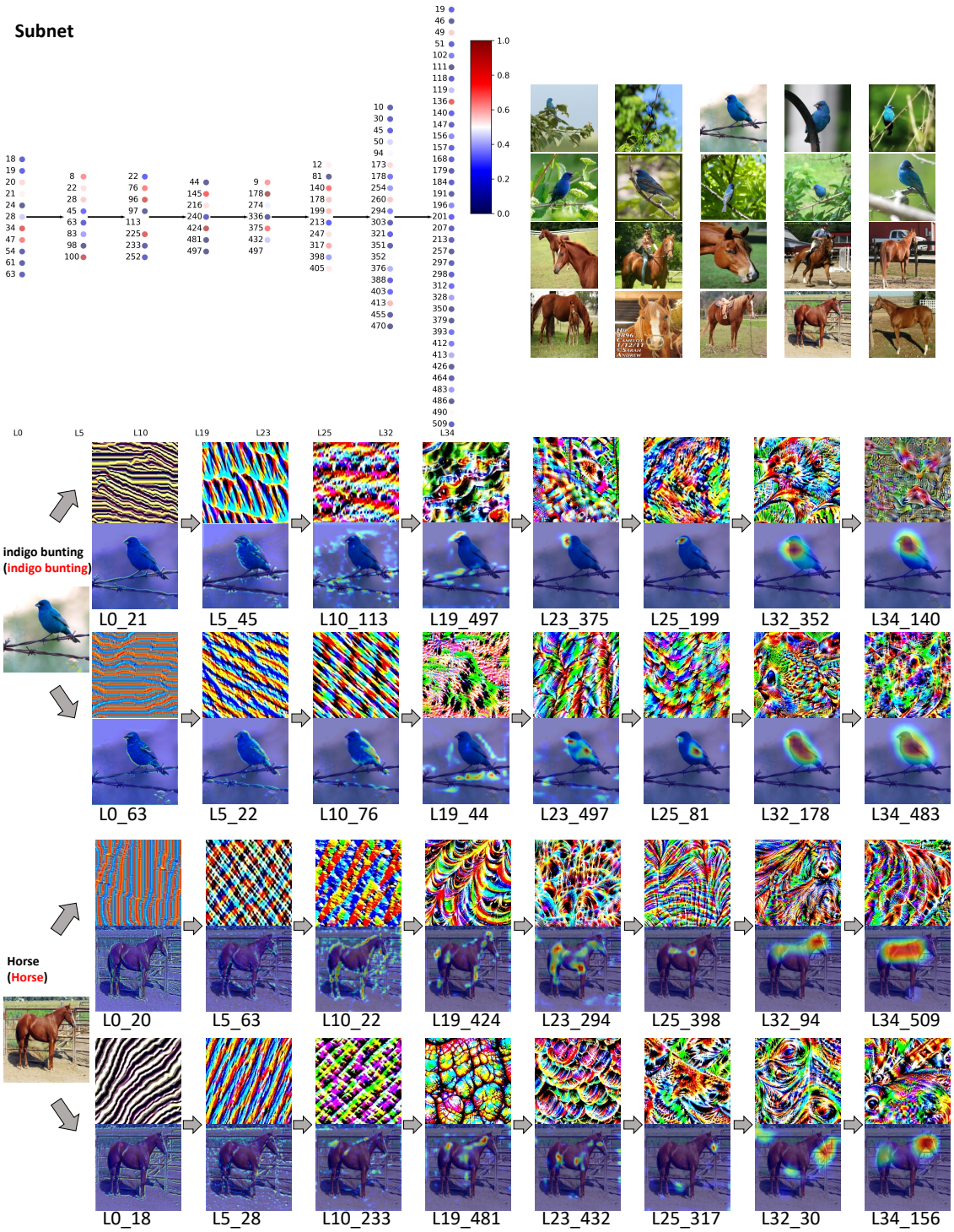

Figure 15: Inference chain by NeuroChains for VGG-19 when applied to images of “indigo bunting” and ”horse”. The sub-network retains only 8/16 layers and 107/4480 filters of the VGG-19. Refer to Section A.2 for detailed analysis of extracted chains.