Dynamic Neural Network Decoupling

Yuchao Li[1], Rongrong Ji[1,2]*, Shaohui Lin[1], Baochang Zhang[3],
Chenqian Yan[1], Yongjian Wu[4], Feiyue Huang[4], Ling Shao[5]
[1]Fujian Key Laboratory of Sensing and Computing for Smart City,
Department of Cognitive Science, School of Information Science and Engineering,
Xiamen University, Xiamen, China [2]Peng Cheng Laboratory, Shenzhen, China,
[3]Beihang University, China, [4]BestImage, Tencent Technology (Shanghai) Co.,Ltd, China,
[5]Inception Institute of Artificial Intelligence, Abu Dhabi, UAE
xiamenlyc@gmail.com, rrji@xmu.edu.cn, shaohuilin007@gmail.com, bczhang@buaa.edu.cn,
im.cqyan@gmail.com, {littlekenwu,garyhuang}@tencent.com, ling.shao@inceptioniai.org

Abstract

Convolutional neural networks (CNNs) have achieved a superior performance by taking advantages of the complex network architectures and huge numbers of parameters, which however become uninterpretable and challenge their full potential to practical applications. Towards better understand the rationale behind the network decisions, we propose a novel architecture decoupling method, which dynamically discovers the hierarchical path consisting of activated filters for each input image. In particular, architecture controlling module is introduced in each layer to encode the network architecture and identify the activated filters corresponding to the specific input. Then, mutual information between architecture encoding and the attribute of input image is maximized to decouple the network architecture, and subsequently disentangles the filters by limiting the outputs of filter during training. Extensive experiments show that several merits have been achieved based on the proposed architecture decoupling, *i.e.*, interpretation, acceleration and adversarial attacking.

1 Introduction

Deep convolutional neural networks (CNNs) have been prevailed in various computer vision tasks, such as objection classification [19, 35, 13], detection [33, 32] and semantic segmentation [8]. However, the excellent performance of CNN is rooted in its complex network architecture with a gigantic number of parameters, which thereby restrict the interpretation of its internal working mechanism. Such a contradiction obstructs the network being applied to the task-critical applications, such as medical diagnosis, automatic robots and self-driving cars.

The network interpretation approaches [40, 4, 30, 41, 17, 25] have the capability to explain the intrinsic properties in the network. For instance, Bau *et al.* [4] proposed a general framework to quantify the interpretability of the individual hidden units by evaluating a set of semantic concepts. Zhang *et al.* [41] interprets the network by promoting each filter in the high convolutional layers to detect a specific object. Although, semantic and quantitative explanations in these methods can help us better understand the network, they still far from interpreting the rationale behind the overall functional process of a network. It inspires us to investigate how an inference is achieved via analyzing its working process. However, each inference consists of huge numbers of parameters with a complex calculation process, where every parameter contributes to the final result. It is difficult to understand the role of each parameter in the working process of network, which motivates us to

*Corresponding author.

reduce the number of parameters involved in the inference to make each decision become easy to explain.

Therefore, in this paper, the essential problem is that which filters are required for each inference. By disentangling the calculation path (*i.e.* using different set of filters) for each inference, we can trace the functional processing behavior of each layer under a hierarchical path to understand the working principle of the network, as shown in Fig. 1. After decoupling the network architecture (*i.e.* the calculation path), each filter is only related to a set of similar input images, the ensemble of which forms a decoupled sub-architecture responsible to the decision marking of a specific input. Such a decoupled architecture further merits in low computational cost for network compression and acceleration, as well as good hints for adversarial network attacking.

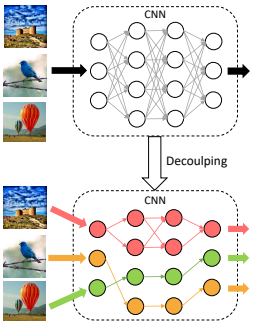


Figure 1: An architecture decoupling.

In this paper, we introduce a dynamic and interpretable network decoupling approach to handle the above issues, as shown in Fig. 2. Our key design lies in a novel light-weight architecture controlling module. This module is equipped in each layer to interpret the network by dynamically selecting filters in network inference with negligible computational burden. To disentangle the network architecture, we then maximize the mutual information between the architecture encoding (*i.e.* the result of filter selection) and the inherent attribute of input images, which makes network dynamically identify the calculation path related to the online input. To better improve the performance of encoded networks, we further increase the similarity between the distribution of architecture encoding and the output of convolutional layer to make each filter only respond to a specific object. In particular, we sparsify the architecture encoding to attenuate the thinner calculation path for each input. We also employ an improved semantic hashing to make discrete architecture encoding differentiable, which therefore can be trained by using stochastic gradient descent (SGD).

Several merits are introduced by the proposed architecture decoupling scheme:

- We propose a novel light-weight architecture controlling module to interpret the neural network by dynamically identifying the activated filter for different inputs. This module is differentiable, such that the network can be optimized by using gradient descent method.
- We lead the disentangled architecture, by which functional processing of each layer can well be interpreted, which help us better understand the rationale behind the network inference. Thus, the decoupled architecture benefits extensive applications, including network interpretation, acceleration and adversarial attacking.
- Our method is generic and flexible, which can be easily used on existing network architectures, such as VGG, ResNet and GoogleNet. We achieve up to $2\times \sim 4\times$ speed up with few accuracy drop for these popular network architectures.

2 Related Works

Network Interpretability. One way is to interpret the network by analyzing the function of local units [39, 3, 4, 30, 41]. For instance, Yosinski *et al.* [39] evaluated the transferability of filters in the intermediate convolutional layers. Aubry *et al.* [3] computed feature distributions for different categories to interpret the internal CNN structure. Recently, Morcos *et al.* [30] found that there exists a close relationship between units and class labels, such that units can be separated by different classes.

Another way to interpret network is [17, 25, 21, 42] by analyzing how the network works. The influence functions [17] are proposed to trace the model prediction. Lakkaraju *et al.* [21] explored the knowledge blind spots of neural networks in a weakly-supervised manner. Recently, a unified framework has been proposed to interpret network prediction [25]. To better understand the classification process, a decision tree has been used in [42]. Moreover, Alain *et al.* [2] proposed to add a linear classifier to each layer and then explore the information of the features. However, it only considers the difference between features in different layers, while the intra-layer difference is not involved in consideration. Wang *et al.* [38] interpreted the network by analyzing the routing path

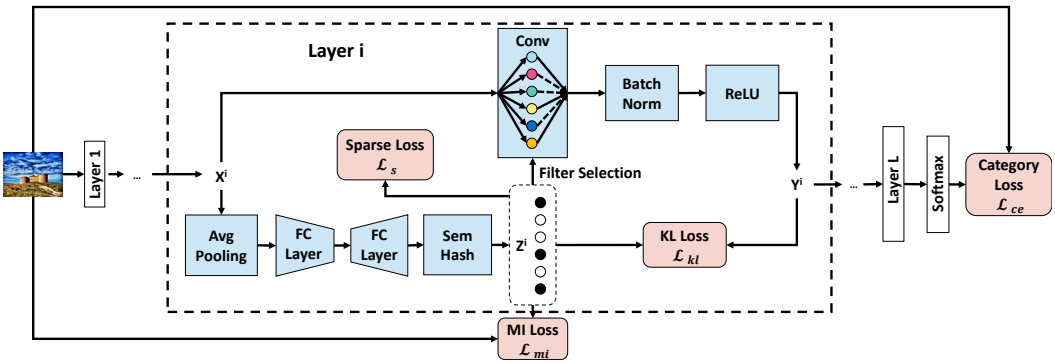


Figure 2: The framework of out method. The architecture encoding $\mathbf{z}^l$ is first constructed and learned to determine the filter selection by minimizing the loss function Eq. 21. The activated filters are then used to participate in the network inference for network decoupling.

of different inputs. It, however, does not train the network to improve the network interpretability, such that the network still has many uninterpretable parameters. Different from all these methods, we interpret the working principle of the network by disentangling the network architecture, upon which we further decouple each filter in the intra-layer to explore the interpretable semantic concepts across nodes on the calculation path.

Conditional Computation. Early works on conditional computation concentrated on the selection of model components on the fly, such as the work [6] explored the influence of stochastic or non-smooth neurons when estimating the gradient of loss function. Later, Bengio *et al*. [5] employed reinforcement learning to optimize the conditional computation policies. Instead of using one network to finish the classification task, Bolukbasi *et al.* [7] proposed a network selection approach with network component ensembling to achieve a better performance. Recently, GaterNet was introduced in [9] to generate binary gates to selectively activated network filters based on each input. However, the approach cannot reduce enough number of filters to decouple the network architecture, which obstructs us exploring the working principle of the network.

Differing from defining an expert network to decide the conditional computation, Figurnov *et al*. [10] dynamically adjusted the number of executed ResNet layers for specific image regions by halting scores. Similar to the previous work, Wang *et al*. [37] accelerated the network by skipping the convolutional block based on the activations of previous block. Liu *et al*. [23] transformed a network into directed acyclic graph of differentiable modules and achieved dynamically selected executed neurons by using control module. Recently, a feature boosting and suppression method [11] was used to skip unimportant convolutional output channels at runtime . However, it skipped the same number of filters for each layer, which does not consider the inter-layer difference. Different from the above works, we employ a novel architecture controlling module to decouple network architecture by training them fit to data distribution. The network after training by this module becomes interpretable, facilitating us to visualize the intrinsic network structure, to accelerate the network inference, and to conduct adversarial attacking detection.

3 Architecture Decoupling

Formally speaking, the l-th convolutional layer in the network with a batch normalization (BN) and a ReLU layer transforms $\mathcal{X}^l \in \mathbb{R}^{C^l \times H^l_{in} \times W^l_{in}}$ to $\mathcal{Y}^l \in \mathbb{R}^{N^l \times H^l_{out} \times W^l_{out}}$ using the weight $\mathcal{W}^l \in \mathbb{R}^{N^l \times C^l \times D^l \times D^l}$ can be defined as:

$$\mathcal{Y}^l = \Big(BN\big(Conv(\mathcal{X}^l, \mathcal{W}^l)\big) \Big)_+, \tag{1}$$

where $(\cdot)_+$ represents the ReLU layer, and Conv$(\cdot,\cdot)$ denoted as standard convolution operator.

3.1 Filter Selection

For each input image, the proposed architecture controlling module selects the filters and generates the calculation path during network inference. In particular, we determine which filters participate in

the convolutional computation before the convolutional operation to accelerate network inference. Therefore, for the l-th convolutional layer, the architecture encoding $\mathbf{z}^l$ only relies on the input $\mathcal{X}^l$ instead of the output $\mathcal{Y}^l$. To obtain efficient filter selection required by practical applications, the global spatial information from each input channel $\mathbf{X}_i^l$ to a scalar $\mathbf{s}_i^l$ is squeezed via global average pooling. We then design $G(\mathbf{s}^l)$ to decide whether the filter should participate in the inference based on $\mathbf{s}^l$:

$$G(\mathbf{s}^l) = \mathbf{W}_2^l \cdot (\mathbf{W}_1^l \cdot \mathbf{s}^l)_+, \tag{2}$$

where $\mathbf{W}_1^l \in \mathbb{R}^{\frac{C^l}{r} \times C^l}$, $\mathbf{W}_2^l \in \mathbb{R}^{N^l \times \frac{C^l}{r}}$ and $\cdot$ represents the matrix multiplication. We ignore the bias parameters for simplicity. To limit the module complexity, $G(\mathbf{s}^l)$ is formed by two fully-connected layers, *i.e.* a dimensionality-reduction layer with weights W_1^l and a dimensionality-increasing layer with weights W_2^l. We empirically set the reduction ratio to $r = 4$ in our experiments. The output of $G(\mathbf{s}^l)$ is the real-valued number that represents the possibility of the filter being selected. To select activated filters, we need to binarize $G(\mathbf{s}^l)$ to construct a binary representation vector $\mathbf{z}^l$. However, a simple discretization method using sign function is not differentiable so that the corresponding gradients cannot be obtained by back-propagation. Thus, we further employ an *Improved SemHash* [16] to transform a real-valued value number in $G(\mathbf{s}^l)$ to be a binary value by a simple rounding bottleneck, which such an operation can be differentiable.

Improved SemHash. It is based on the different operations for training and testing. During training, we first sample a noise $a \sim \mathcal{N}(0,1)^{N^l}$, and have $\widetilde{\mathbf{s}}^l = G(\mathbf{s}^l) + a$. Thereafter, we obtain a real-valued vector and a binary-valued vector as:

$$\mathbf{v}_1^l = \sigma'(\widetilde{\mathbf{s}}^l), \mathbf{v}_2^l = \mathbf{1}(\widetilde{\mathbf{s}}^l > 0), \tag{3}$$

where σ' is a saturating Sigmoid function [15] denoted as:

$$\sigma'(x) = max\Big(0, min\big(1, 1.2\sigma(x) - 0.1\big)\Big), \tag{4}$$

where σ is the Sigmoid function. $\mathbf{v}_1^l$ is real-valued with all elements falling in the interval $[0, 1]$ for the gradient calculating during back-propagation. $\mathbf{v}_2^l$ represents a discretized vector, not involving into the gradient calculation (*i.e.*, only for evaluation and inference). In the forward-propagation, we randomly employ $\mathbf{v}_1^l$ half of the time and $\mathbf{v}_2^l$ the other half to obtain $\mathbf{z}^l$. In the backward-propagation, the gradient of $\mathbf{z}^l$ is same to that of $\mathbf{z}_1^l$. During evaluation/testing, we directly use the binary value in the forward-propagation as:

$$\mathbf{z}^l = \mathbf{1}\big(G(\mathbf{s}^l) > 0\big). \tag{5}$$

Based on $\mathbf{z}^l$, we directly mask the output channels $\mathcal{Y}^l$ using the architecture encoding $\mathbf{z}^l$.

3.2 Network Training

We expect that the network architecture can be gradually disentangled during training. Therefore, the essential problem is how to learn an architecture encoding that fits the data distribution. To this end, we propose three loss functions to train the network and to decouple its architecture.

Mutual Information Loss. When the network architecture is disentangled, different input images have different selected filters as they are related to the specific input. In the information theory, mutual information $I(c; \mathbf{z}^l)$ between the result of filter selection $\mathbf{z}^l$ and the attribute of an input image c measures the correlation between architecture encoding and its input image. $I(c; \mathbf{z}^l) = 0$ means that the selected filters is not related to the input image, which remains the same for all inputs. In contrast, the filter selection depends on the input image when $I(c; \mathbf{z}^l) \neq 0$. To this end, we maximize the mutual information between c and $\mathbf{z}^l$ to achieve the architecture decoupling. Considering the expanded form of mutual information, we have:

$$\begin{aligned} I(c; \mathbf{z}^l) &= H(c) - H(c|\mathbf{z}^l) && (6)\\ &= \sum_c \sum_{\mathbf{z}^l} P(c, \mathbf{z}^l) log P(c|\mathbf{z}^l) + H(c) && (7)\\ &= \sum_c \sum_{\mathbf{z}^l} P(\mathbf{z}^l) P(c|\mathbf{z}^l) log P(c|\mathbf{z}^l) + H(c). && (8) \end{aligned}$$

The mutual information $I(c;\mathbf{z}^l)$ is difficult to be directly maximized, as it is hard to obtain $P(c|\mathbf{z}^l)$. Thus, we use $Q(c|\mathbf{z}^l)$ as a variational approximation to $P(c|\mathbf{z}^l)$ [1]. Here, we parametrize $Q(c|\mathbf{z}^l)$ as a neural network that only contains a fully-connected layer. Considering the fact that the KL-divergence is positive, we have:

$$KL\big(P(c|\mathbf{z}^l), Q(c|\mathbf{z}^l)\big) \geq 0 \Rightarrow \sum_c P(c|\mathbf{z}^l) log P(c|\mathbf{z}^l) \geq \sum_c P(c|\mathbf{z}^l) log Q(c|\mathbf{z}^l). \tag{9}$$

Therefore,

$$I(c;\mathbf{z}^l) \geq \sum_c \sum_{\mathbf{z}^l} P(\mathbf{z}^l) P(c|\mathbf{z}^l) log Q(c|\mathbf{z}^l) + H(c) \tag{10}$$

$$\geq \sum_c \sum_{\mathbf{z}^l} P(\mathbf{z}^l) P(c|\mathbf{z}^l) log Q(c|\mathbf{z}^l) \tag{11}$$

Eq. 10 shows a lower bound of the mutual information $I(c;|\mathbf{z}^l)$. By maximizing the lower bound, the mutual information $I(c;\mathbf{z}^l)$ will be accordingly maximized. In our paper, we use the class label as the attribute of the input image c in the classification task. Thus, maximizing the mutual information in Eq. 10 is equivalent to minimize loss below:

$$\mathcal{L}_{mi} = -\sum_{l=1}^{L} C(X) * log Q(\mathbf{z}^l), \tag{12}$$

where $C(X)$ represents the label of the input image X. $Q(\mathbf{z}^l)$ is defined as $\mathbf{W}^l_{cla} \cdot \mathbf{z}^l$ with a fully-connected weight $\mathbf{W}^l_{cla} \in \mathbb{R}^{K \times N^l}$, where K represents the number of classification categories.

KL-divergence Loss. After maximizing the mutual information $I(c;\mathbf{z}^l)$, it only guarantees that the filter selection depends on the input image. However, it is uncertain whether the filters become different (*i.e.* detect different objects). If a filter only responds to a specific object, it will be inactivated when the input does not contain this object. By limiting each filter only responding to a specific category, they can be disentangled to detect different objects. Thus, we minimize the KL-divergence between the output of the current layer and its corresponding architecture encoding to ensure that the overall filter responses have a similar distribution to the responses of the selected subset. To align the dimension of the convolution output and the architecture encoding, we downsample $\mathcal{Y}^l$ to $\mathbf{y}^l \in \mathbb{R}^N$ using global average pooling. Then the KL-divergence loss is defined as:

$$\mathcal{L}_{kl} = \sum_{l=1}^{L} KL(\mathbf{z}^l || \mathbf{y}^l). \tag{13}$$

As the output of filters is limited based on the result of filter selection, it will become unique and only detects a specific object. Thus, the filters are different from each other, each of which performs its own function. We further to prove the following theorem to reveal the advantage of our loss function.

Theorem 1. If we simultaneously maximize the mutual information $I(c;\mathbf{z}^l)$ and minimize the KL-divergence between $\mathbf{z}^l$ and $\mathbf{y}^l$, the mutual information $I(c;\mathbf{y}^l)$ will be accordingly increased.

Proof of Theorem 1 Considering the difference between $I(c;\mathbf{z}^l)$ and $I(c;\mathbf{y}^l)$, we have:

$$I(c;\mathbf{z}^l) - I(c;\mathbf{y}^l) = (H(\mathbf{z}^l) - H(\mathbf{z}^l|c)) - (H(\mathbf{y}^l) - H(\mathbf{y}^l|c)) \tag{14}$$

$$= (H(\mathbf{z}^l) - H(\mathbf{y}^l)) + (H(\mathbf{y}^l|c) - H(\mathbf{z}^l|c)). \tag{15}$$

If $\mathbf{y}^l$ is close to $\mathbf{z}^l$, $H(\mathbf{y}^l)$ is also close to $H(\mathbf{z}^l)$. Then, the joint distribution $P(\mathcal{X}^l, \mathbf{y}^l, c)$ and $P(\mathcal{X}^l, \mathbf{z}^l, c)$ are factorized as follows:

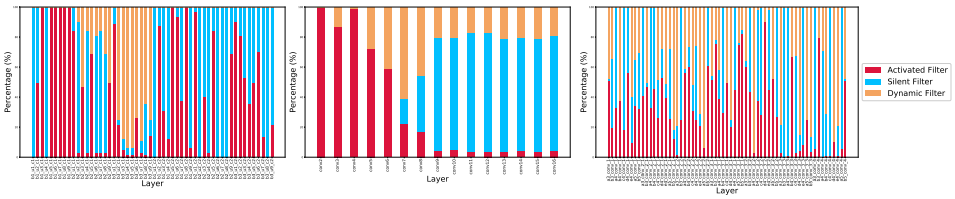
$$P(\mathcal{X}^l, \mathbf{y}^l, c) = P(\mathbf{y}^l|\mathcal{X}^l, c) P(c|\mathcal{X}^l) P(\mathcal{X}^l) = P(\mathbf{y}^l|\mathcal{X}^l) P(c|\mathcal{X}^l) P(\mathcal{X}^l) \tag{16}$$

$$P(\mathcal{X}^l, \mathbf{z}^l, c) = P(\mathbf{z}^l|\mathcal{X}^l, c) P(c|\mathcal{X}^l) P(\mathcal{X}^l) = P(\mathbf{z}^l|\mathcal{X}^l) P(c|\mathcal{X}^l) P(\mathcal{X}^l). \tag{17}$$

Because of the Markov chain $\mathbf{y}^l \leftrightarrow \mathcal{X}^l \leftrightarrow c$ and $\mathbf{z}^l \leftrightarrow \mathcal{X}^l \leftrightarrow c$, we can assume $P(\mathbf{y}^l|\mathcal{X}^l, c) = P(\mathbf{y}^l|\mathcal{X}^l)$ and $P(\mathbf{z}^l|\mathcal{X}^l, c) = P(\mathbf{z}^l|\mathcal{X}^l)$. Thus, $P(\mathbf{y}^l|c)$ and $P(\mathbf{z}^l|c)$ can be rewritten as:

$$P(\mathbf{y}^l|c) = \frac{P(c, \mathbf{y}^l)}{P(c)} = \frac{\sum_{\mathcal{X}^l} P(c, \mathbf{y}^l, \mathcal{X}^l)}{P(c)} = \frac{\sum_{\mathcal{X}^l} P(\mathbf{y}^l|\mathcal{X}^l) P(c|\mathcal{X}^l) P(\mathcal{X}^l)}{P(c)} \tag{18}$$

$$P(\mathbf{z}^l|c) = \frac{P(c, \mathbf{z}^l)}{P(c)} = \frac{\sum_{\mathcal{X}^l} P(c, \mathbf{z}^l, \mathcal{X}^l)}{P(c)} = \frac{\sum_{\mathcal{X}^l} P(\mathbf{z}^l|\mathcal{X}^l) P(c|\mathcal{X}^l) P(\mathcal{X}^l)}{P(c)}. \tag{19}$$



(a) ResNet-56 (b) VGGNet (c) GoogleNet

Figure 3: The distribution of filters in different states in each layer (best viewed in zooming in).

When $P(\mathbf{y}^l)$ is close to $P(\mathbf{z}^l)$, $P(\mathbf{y}^l|\mathcal{X}^l)$ becomes close to $P(\mathbf{z}^l|\mathcal{X}^l)$, and hence the difference between $P(\mathbf{y}^l|c)$ and $P(\mathbf{z}^l|c)$ becomes smaller. Thus, $H(\mathbf{y}^l|c)$ and $H(\mathbf{z}^l|c)$ are similar. In summary, if $P(\mathbf{y}^l)$ is close to $P(\mathbf{z}^l)$, the difference between $I(c;\mathbf{y}^l)$ and $I(c;\mathbf{z}^l)$ is small. Thus, when maximizing the mutual information $I(c;\mathbf{z}^l)$, $I(c;\mathbf{y}^l)$ also increases.

When the mutual information between the feature maps (*i.e.* the output of filters) and the properties of input image increases, the feature maps become more discriminative and the network performance is improved accordingly.

Sparse Loss. ℓ_1-regularization on $\mathbf{z}^l$ is introduced to encourage the architecture encoding being sparse, which means the calculation path of each input becomes thinner. Thus, the sparse loss is defined as:

$$\mathcal{L}_s = \sum_{l=1}^{L} \|\mathbf{z}^l\|_1. \tag{20}$$

Therefore, we obtain the overall loss function as:

$$\mathcal{L} = \mathcal{L}_{ce} + \lambda_m * \mathcal{L}_{mi} + \lambda_k * \mathcal{L}_{kl} + \lambda_s * \mathcal{L}_s, \tag{21}$$

where $\mathcal{L}_{ce}$ is the network classification loss and λ_m, λ_k and λ_s are the hyper-parameters. Eq 21 can be effectively solved via SGD.

4 Experiments

We evaluate the effectiveness of the proposed architecture decoupling scheme on three datasets, *i.e.*, CIFAR-10, CIFAR-100 [18] and ImageNet ILSVRC 2012 [34]. CIFAR-10 and CIFAR-100 contain 50,000 training images and 10,000 testing images from 10 and 100 classes, respectively. Each image in both them is with a resolution of 32×32. ImageNet ILSVRC 2012 consists of 1.28 million training images and 50,000 images from 1,000 classes.

4.1 Implementation Details

We implement our method using Pytorch [31]. The weights of decoupled networks are initialized by the weights from their corresponding pre-trained models. We add the architecture controlling module to all convolutional layers except the first and last ones. All networks are trained using stochastic gradient descent (SGD) with a momentum of 0.9. On CIFAR-10 and CIFAR-100, we train all the networks for 200 epochs using a mini-batch size of 128. The initial learning rate is 0.1 and is divided by 10 at 50% and 75% of the total number of epochs. On ImageNet2012, we train the networks for 30 epochs with a mini-batch size of 64 and 256 on VGG-16 and ResNet-18, respectively. The initial learning rate is 0.001 and is multiplied by 0.1 at epochs 10 and 20. The evaluation on CPU is measured in a single-thread Intel Core i7-6900K CPU. After the network training, we find that some filters are always silent for all inputs, which means we can remove these redundant filters without the degradation of accuracy. It also demonstrates that the network does not require all filters to conduct a given task.

4.2 Network Acceleration

We decouple three different network architectures (*i.e.* ResNet-56 [13], VGGNet [35] and GoogleNet [36]) on CIFAR-10 and CIFAR-100. The VGGNet in our experiments is the same to the network in [24]. As shown in Table 1, our method achieves the best trade-off between accuracy and

Model	#Param. (Compression ratio)	Mean FLOPs (Acceleration ratio)	Top-1 Acc(%)
ResNet-56	0.85M	125M	93.17
CP [14]	-	63M (2.00×)	91.80
L1 [22]	0.73M (1.16×)	90M (1.39×)	93.06
L1 [22]	0.77M (1.10×)	110M (1.14×)	93.10
SkipNet [37]*	-	103M (1.21×)	92.50
Architecture Decoupling	0.57M (1.49×)	86M (1.45×)	93.43
VGGNet	20.04M	398M	93.75
Slimming [24]	2.30M (8.71×)	196M (2.03×)	93.80
Thinet [26]*	10.44M (1.92×)	199M (2.00×)	93.87
Architecture Decoupling	2.25M (8.90×)	141M (2.82×)	93.82
GoogleNet	6.17M	1.52B	95.11
L1 [22]	3.51M (1.76×)	1.02B (1.49×)	94.54
Architecture Decoupling	3.70M (1.67×)	0.39B (3.90×)	94.65

Table 1: Results on CIFAR-10. $*$ represents the result based on the implementation of original paper.

Model	#Param. (Compression ratio)	Mean FLOPs (Acceleration ratio)	Top-1 Acc(%)
ResNet-56	0.86M	125M	70.43
L1 [22]*	0.59M (1.46×)	86M (1.45×)	69.38
Taylor [28]*	0.59M (1.46×)	86M (1.45×)	69.52
Architecture Decoupling	0.76M (1.13×)	41M (3.05×)	69.54
Architecture Decoupling	0.57M (1.51×)	55M (2.27×)	69.72
VGGNet	20.08M	398M	72.98
L1 [22]*	9.80M (2.05×)	194M (2.05×)	72.14
Taylor [28]*	9.80M (2.05×)	194M (2.05×)	71.90
Slimming [24]	5.00M (4.02×)	250M (1.59×)	73.48
Architecture Decoupling	7.24M (2.77×)	191M (2.08×)	73.84
Architecture Decoupling	6.30M (3.19×)	178M (2.24×)	73.27
GoogleNet	6.26M	1.52B	77.99
L1 [22]*	3.58M (1.75×)	0.87B (1.75×)	77.09
Taylor [28]*	3.58M (1.75×)	0.87B (1.75×)	77.22
Architecture Decoupling	3.70M (1.69×)	0.75B (2.03×)	77.28

Table 2: Results on CIFAR-100. $*$ represents the result based on the implementation of original paper.

speedup/compression rate, compared to the static pruning [14, 22, 24, 26] and the dynamic pruning [37]. In Table 2, we compare our method with [22, 28] on three different networks. Our method can achieve the highest speedup rate with the smallest drop in accuracy. For example, we achieve 2.03× speed up with only 0.71% Top-1 accuracy drop in accuracy on GoogleNet. For ImageNet 2012, as shown in Table 3, the results show that our method achieves a high speedup rate on CPU, while the model accuracy decreases with network decoupling. We conjecture this is due to the large amount of classes in ImageNet 2012, such that the limited number of features on ResNet-18 and VGG-16 cannot successfully represent the distribution of each class. Therefore, the network performance will be destroyed if we excessively decouple the network architecture.

4.3 Network Interpretability

Filter State. The state of a filter in the network has three possibilities: it responds to all the input samples, it not responds to any input sample, and it responds to the specific inputs. These three possibilities are termed as *Activated Filter*, *Silent Filter* and *Dynamic Filter*, respectively. We collect the architecture encoding of each test sample and then analyze the distribution of different filter states from each layer in three different networks. For ResNet-56 and GoogleNet, we combine the results of the same position layers at different depths to facilitate the finding of regular patterns. As

Model	Mean CPU Time (Acceleration ratio)	Top-1 Acc(%)	Top-5 Acc(%)
ResNet-18	74ms	69.83	89.08
Architecture Decoupling	68ms (1.09×)	69.05	88.64
Architecture Decoupling	64ms (1.16×)	67.93	88.13
VGG-16	475ms	71.59	90.38
Architecture Decoupling	304ms (1.56×)	68.77	88.80
Architecture Decoupling	214ms (2.22×)	64.97	86.47

Table 3: Results on ImageNet2012.

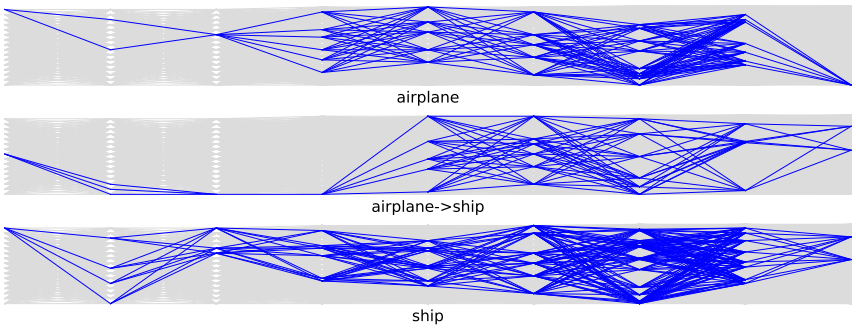


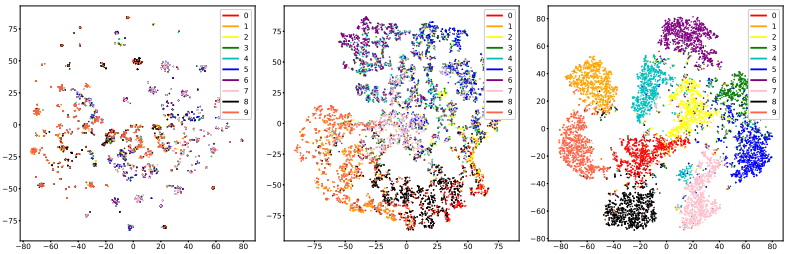
Figure 4: The calculation path of different categories. The first row and third row represent the calculation path of "airplane" and "ship" categories, respectively. The second row represents the calculation path of images originally belonging to "airplane" that are classified as a ship after being attacked.

shown in Fig. 3, the proportion of dynamic filters increases with the increase of network depth. This phenomenon demonstrates that filters in the top layer tend to detect high-level semantic features, which are highly related to the input images. In contrast, filters in the bottom layer detect low-level features, which are always shared in each image. Thus, with the increase of layer depth, the proportion of dynamic filters increases consistently if the activated and dynamic filters are only collected for evaluation. Interestingly, second convolutional layer in each block on ResNet-56 does not contain any dynamic filter. This is due to the fact that its outputs are used for feature fusion.

Architecture Encoding. To investigate the distribution of the architecture encoding in different layers, we collect the architecture encoding from three layers with different depths on ResNet-56. These architecture encodings are then projected into 2-dimensional space using t-SNE [27]. As shown in Fig. 5, each color represents one category and each dot is an architecture encoding corresponding to an input. We can see that the architecture in the bottom layer is difficult to disentangle, since the filters in the bottom layer tend to detect simple structures that always appear in every input image. Thus, the filters will respond to all input images. As the layers become deeper, the architecture gradually becomes disentangled. Based on the finding that the hierarchical calculation paths of each input is related to its class, our architecture decoupling method helps better understand the network.

Category-aware Calculation Path. After decoupling the network architecture, the calculation path of input images from different categories will become diverse. As shown in Fig 4, we visualize the calculation path of "airplane" and "ship" categories by counting all images under this category in ResNet-20 on CIFAR-10. Each row represents the calculation path of one category and each column represents one layer. We remove the activated filters and silent filters, and only visualize the first convolutional layer in each block to simplify the calculation path. Meanwhile, each filter is represented by a dot. The results in Fig 4 reveal the different calculation paths of airplane image and ship image. Compared to the airplane image, the ship image contains more complex information, and requires more filters to inference. As shown in Fig 9, we visualize the calculation paths of different categories for ResNet-20 on CIFAR-10. Each path for the specific class (*e.g.*, airplane) is obtained by the statistic of all images labelled by this class. Our method can successfully disentangle the network architecture, and effectively distinguish the "hard" and " easy" classes. For example, The images belonging to "automobile" or "truck" require much more filters (*i.e.*, more complex path), compared to a thinner path on "bird".

Network Visualization We follow the method in [43] further visualize the receptive field of the activation from different filters. As shown in Fig. 6, each row represents the output of a filter



(a) Bottom Layer (b) Middle Layer (c) Top Layer

Figure 5: Visualization of the distribution of architecture encoding in layers with different depths.

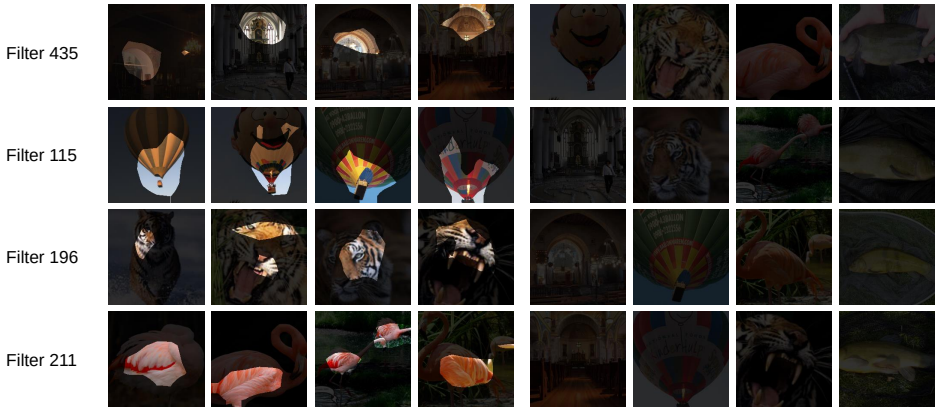


Figure 6: Visualization of the visual concept that different filters detect at the *Block4-Unit-2-Conv1* of ResNet-18.

with different input images. It shows that each filter only responds to one specific feature without ambiguity. For instance, the 211-th filter in *Block4-Unit-2-Conv1* of ResNet-18 only detects flamingo wings and the 115-th filter only responds to the bottom of a hot air balloon. Therefore, our method can successfully disentangle the filters and make them selectively respond to distinct objects of a category, and ignore others.

Relationship Between Filter and Category In our method, we quantify the relationship between convolutional filters of the input images and their categories, and further interpret the different functions of filters in the network. As mentioned in Section 3.3, we use $Q(c|\mathbf{z}^l)$ to approximate $P(c|\mathbf{z}^l)$ which represents the probability of a certain category under the specific architecture encoding $\mathbf{z}^l$. Therefore, after training, the corresponding $\mathbf{W}^l_{cla_{i,j}}$ represents the probability of the input image belongings to the j-th category when the i-th filter participates in the network inference. As shown in Fig. 7, we visualize $\mathbf{W}^l_{cla}$ corresponding to the first three layers in each block on ResNet-56. Each row represents one category and each column represents a filter. We draw the same observation to the work [40] that features in each layer shows the hierarchical nature. In other words, filters in the low layers usually detect simple features, like textures, which are shared by the images with different classes, while filters in the high layers are more likely to detect semantic features, which are related

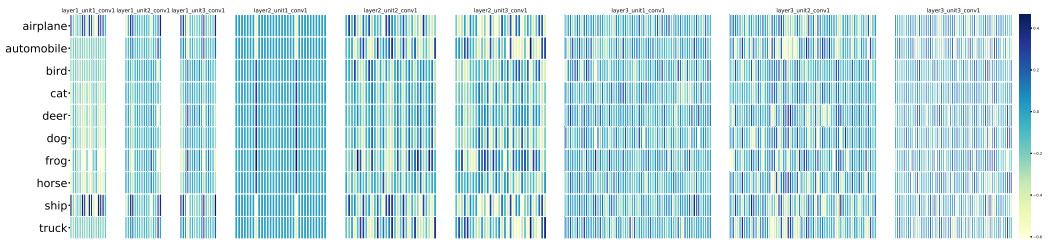
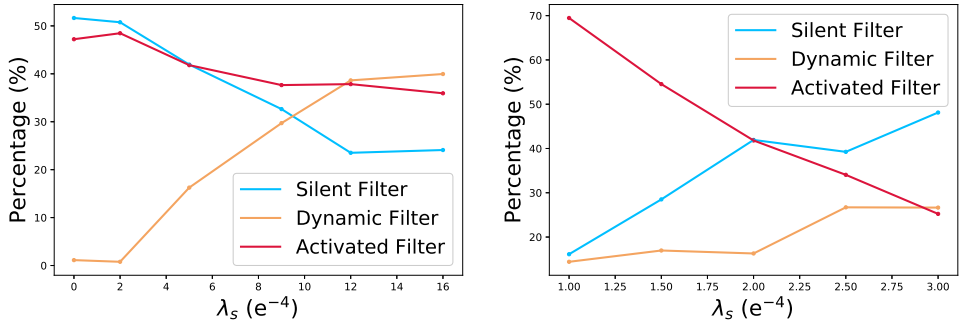


Figure 7: Visualization of the relationship between filter and category.



(a) The influence of λ_m.

(b) The influence of λ_s.

Figure 8: Percentage of different filter states in different λ_m and λ_s on ResNet-56.

to the specific classes. Moreover, the result shows that our method successfully disentangle the filters, such that most of them are only related to one categorize.

4.4 Adversarial Attack Analysis

Recently, several works [12, 20, 29] have found that neural network is vulnerable to adversarial examples. Only adding a slight noise to an input image can disturb the robustness of networks. To interpret how the noise works, we use the FGSM [12] to attack ResNet-20 on CIFAR-10 after architecture decoupling, and visualize the corresponding calculation path. As shown in Fig. 4, we add the noise to an image belongings to the "airplane" category, while the network predicts it as "ship". we can see that the calculation path between the input belongings to true "airplane" and the adversarial input is totally different from low-level layers. This suggests that the adversarial samples confuse the network from the bottom layers. Meanwhile, we found that the calculation path of adversarial samples classified as "ship" is different from the calculation path of the original ship image. In other words, adversarial samples do not actually completely deceive the network, which can be detected by analyzing their calculation path. As shown in Fig. 10, the images labelled by "airplane" are attacked to be other classes. Compared to the original calculation path in Fig. 9, the adversarial samples always confuse the network starting from the bottom layer. Furthermore, we compare the calculation path of different categories that are attacked to be the same one in Fig. 11,. As the images attacked to be the same class almost share the same calculation path, we draw a conclusion that we can effectively detect the adversarial samples using our architecture decoupling method by comparing the calculation path of un-attacked/original images (*e.g.*, dog in Fig. 9) and adversarial samples.

4.5 Ablation Study

Distribution of Filters In our method, we have three hyper-parameters (*i.e.*, λ_m, λ_k and λ_s) to control the network training. λ_k is empirically set to 1 in our experiment. As mentioned in the above section, there are three filter states: activated, silent and dynamic. As shown in Fig. 8, we calculate the percentage of different filter states in the network with different λ_m and λ_s on ResNet-56. In Fig. 8(a), we set λ_s to 0.0002, and the $\lambda_m = 0.009$ in Fig. 8(b). We find that λ_m controls the number of dynamic filters, which means the network architecture can be disentangled as λ_m increases. Meanwhile, λ_s controls the number of filters that participate in the network inference. As λ_s decreases, the network uses more filters for network inference.

Influence of Dynamic Filters As shown in Fig. 8(a), when $\lambda_m = 0$, the network does not have any dynamic filter. To explore the influence of the dynamic filters, we compare the network accuracy when $\lambda_m = 0$ and $\lambda_m = 0.005$ with different λ_s. As shown in Table 4, the proper usage of dynamic filters can improve the performance of the network.

λ_s	λ_m	Mean FLOPs	Top-1 Acc(%)	The percentage of dynamic filter
0.0001	0	117M	92.81	0.00%
0.0001	0.005	104M	93.60	14.41%
0.0001	0.01	101M	93.39	30.21%
0.00015	0	85M	92.84	0.14%
0.00015	0.005	85M	93.08	16.96%
0.00015	0.01	77M	92.78	35.73%
0.0002	0	68M	92.53	1.13%
0.0002	0.005	67M	92.78	29.71%
0.0002	0.01	59M	92.53	39.96%

Table 4: Results of ResNet-56 on CIFAR-10 with different λ_m and λ_s.

5 Conclusion

In this paper, we propose a novel architecture decoupling method to interpret the rationale behind the overall working process of a network. In particular, architecture controlling module is embedded in each layer to dynamically identify the activated filters. Then, by maximizing the mutual information between the architecture encoding and the attribute of input image, we decouple the network architecture to explore the functional processing behaviour of each layer. We further disentangle filters to make them detect different objects and improve the discriminative ability of the decoupled network by limiting the output of filters. Experiments show that our method can successfully decouple the network architecture to achieve several merits, *i.e.,* interpretation, acceleration and adversarial attacking.

References

[1] David Barber Felix Agakov. The im algorithm: a variational approach to information maximization. *Advances in Neural Information Processing Systems (NeurIPS)*, 2004.

[2] Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. *International Conference on Learning Representations (ICLR) WorkShop*, 2017.

[3] Mathieu Aubry and Bryan C Russell. Understanding deep features with computer-generated imagery. *International Conference on Computer Vision (ICCV)*, 2015.

[4] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[5] Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. *International Conference on Learning Representations (ICLR)*, 2016.

[6] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[7] Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. Adaptive neural networks for efficient inference. *International Conference on Machine Learning (ICML)*, 2017.

[8] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. *European Conference on Computer Vision (ECCV)*, 2018.

[9] Zhourong Chen, Yang Li, Samy Bengio, and Si Si. You look twice: Gaternet for dynamic filter selection in cnns. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[10] Michael Figurnov, Maxwell D Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry Vetrov, and Ruslan Salakhutdinov. Spatially adaptive computation time for residual networks. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[11] Xitong Gao, Yiren Zhao, Lukasz Dudziak, Robert Mullins, and Cheng-zhong Xu. Dynamic channel pruning: Feature boosting and suppression. *International Conference on Learning Representations (ICLR)*, 2018.

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

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[14] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. *International Conference on Computer Vision (ICCV)*, 2017.

[15] Łukasz Kaiser and Samy Bengio. Can active memory replace attention? *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[16] Łukasz Kaiser, Aurko Roy, Ashish Vaswani, Niki Parmar, Samy Bengio, Jakob Uszkoreit, and Noam Shazeer. Fast decoding in sequence models using discrete latent variables. *International Conference on Machine Learning (ICML)*, 2018.

[17] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. *International Conference on Machine Learning (ICML)*, 2017.

[18] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 2012.

[20] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

[21] Himabindu Lakkaraju, Ece Kamar, Rich Caruana, and Eric Horvitz. Identifying unknown unknowns in the open world: Representations and policies for guided exploration. *Association for the Advancement of Artificial Intelligence (AAAI)*, 2017.

[22] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *International Conference on Learning Representations (ICLR)*, 2016.

[23] Lanlan Liu and Jia Deng. Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution. *Association for the Advancement of Artificial Intelligence (AAAI)*, 2018.

[24] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. *International Conference on Computer Vision (ICCV)*, 2017.

[25] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[26] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. *International Conference on Computer Vision (ICCV)*, 2017.

[27] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research (JMLR)*, pages 2579–2605, 2008.

[28] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *International Conference on Learning Representations (ICLR)*, 2017.

[29] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[30] Ari S Morcos, David GT Barrett, Neil C Rabinowitz, and Matthew Botvinick. On the importance of single directions for generalization. *International Conference on Learning Representations (ICLR)*, 2018.

[31] 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. *Advances in Neural Information Processing Systems (NeurIPS) Workshop*, 2017.

[32] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. 2018.

[33] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: towards real-time object detection with region proposal networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

[34] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[35] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[36] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[37] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. *European Conference on Computer Vision (ECCV)*, 2018.

[38] Yulong Wang, Hang Su, Bo Zhang, and Xiaolin Hu. Interpret neural networks by identifying critical data routing paths. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[39] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *Advances in Neural Information Processing Systems (NeurIPS)*, 2014.

[40] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. 2014.

[41] Quanshi Zhang, Ying Nian Wu, and Song-Chun Zhu. Interpretable convolutional neural networks. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[42] Quanshi Zhang, Yu Yang, Ying Nian Wu, and Song-Chun Zhu. Interpreting cnns via decision trees. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[43] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Object detectors emerge in deep scene cnns. *International Conference on Learning Representations (ICLR)*, 2014.

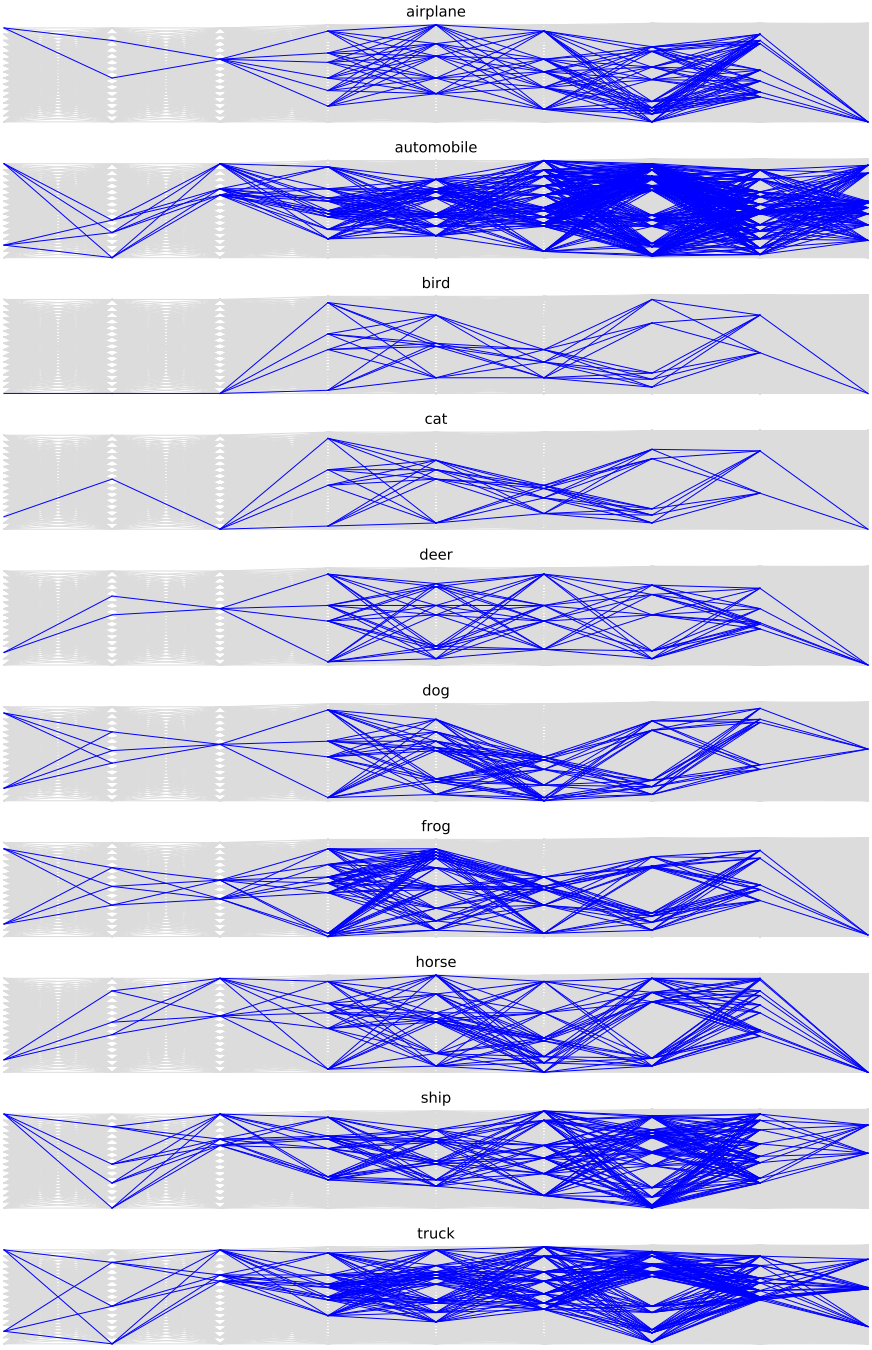


Figure 9: The calculation path of different categories on ResNet-20.

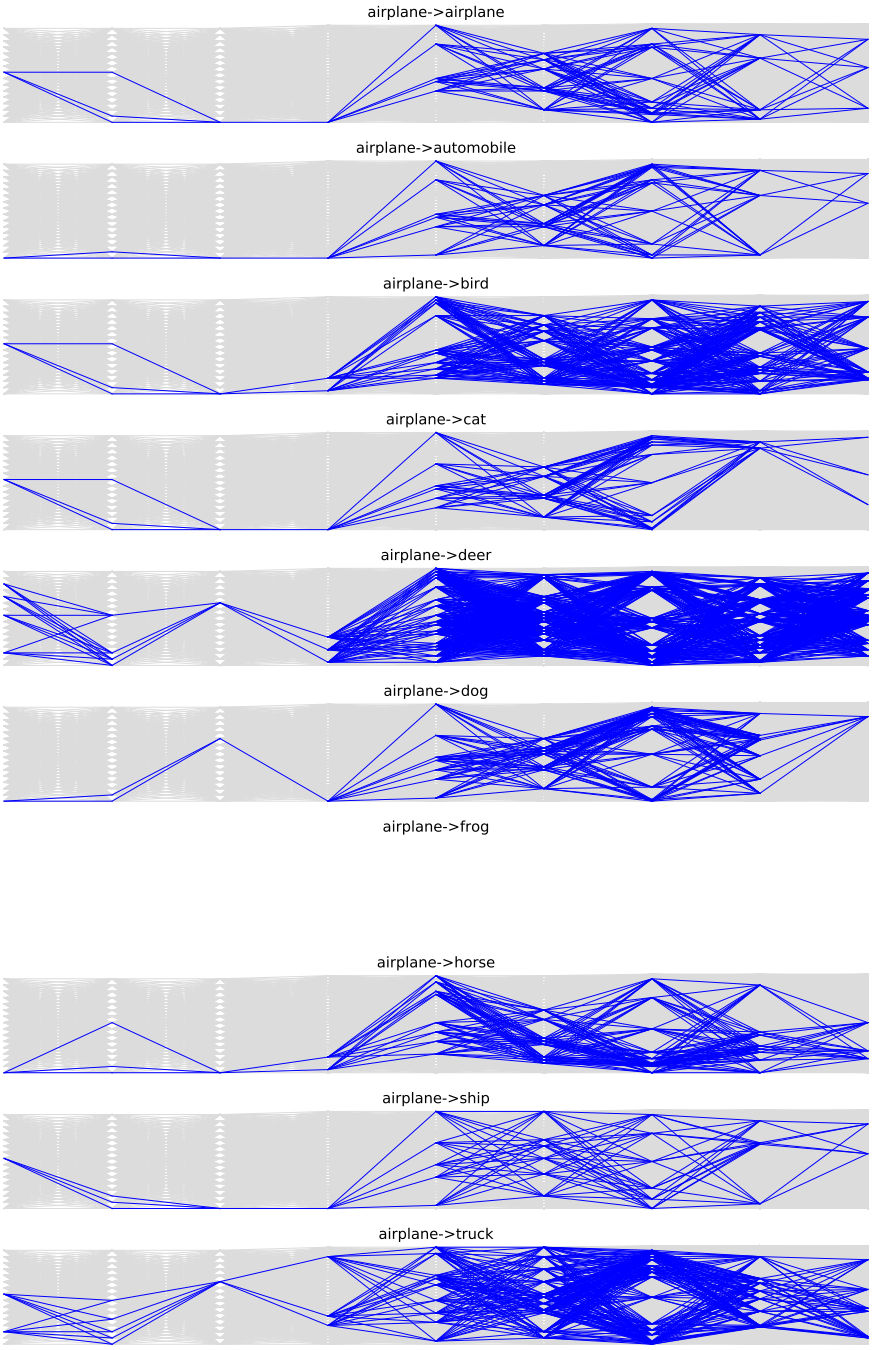


Figure 10: The calculation paths of same category images that are classified as other categories after being attacked on ResNet-20.

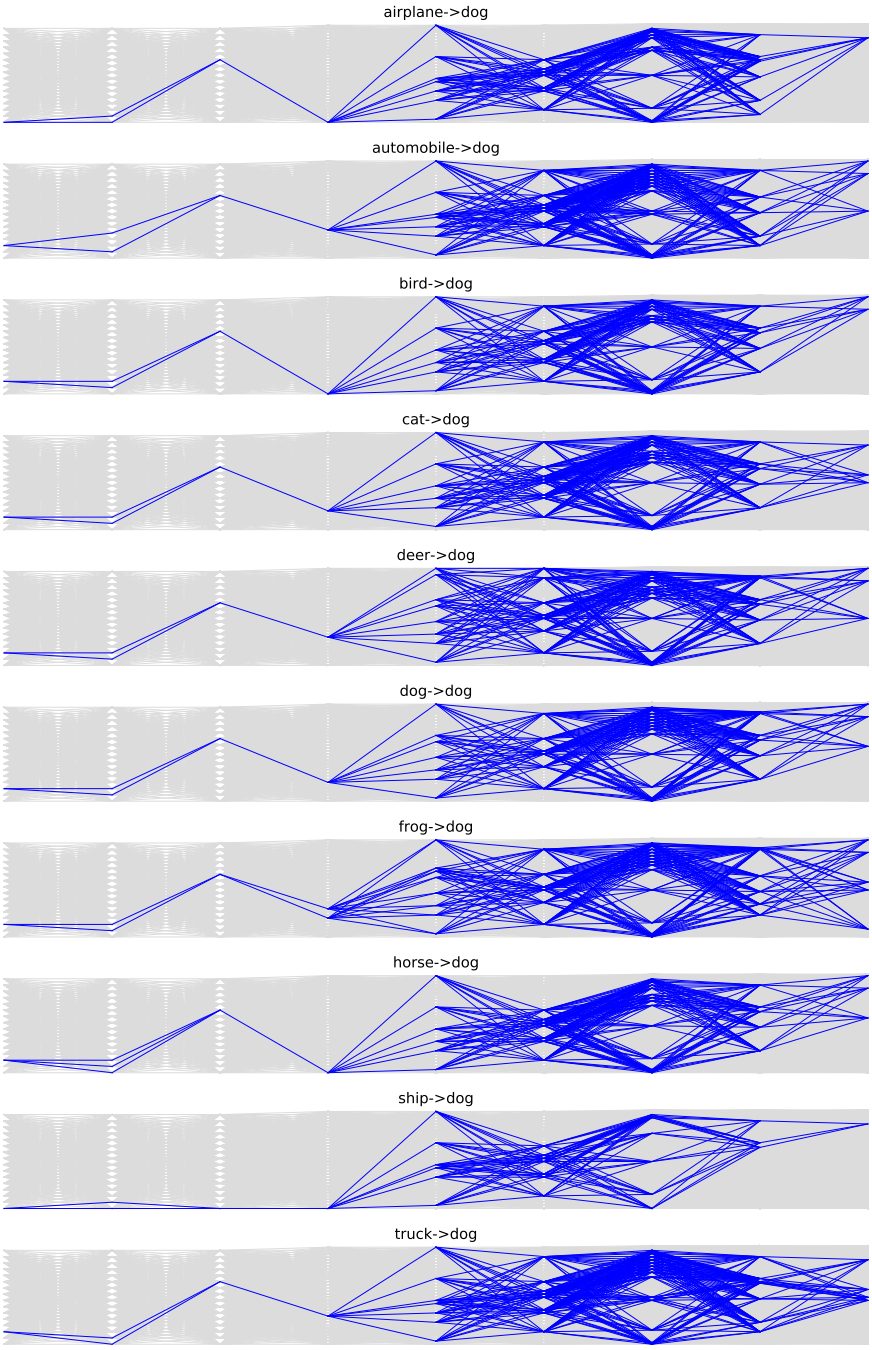


Figure 11: The calculation paths of different categories images that are classified as same category after being attacked on ResNet-20.