DISENTANGLEMENT, VISUALIZATION AND ANALYSIS OF COMPLEX FEATURES IN DNNS

Anonymous authors
Paper under double-blind review

ABSTRACT

This paper aims to define, visualize, and analyze the feature complexity that is learned by a DNN. We propose a generic definition for the feature complexity. Given the feature of a certain layer in the DNN, our method disentangles and visualizes feature components of different complexity orders from the feature. The disentanglement of feature components enables us to evaluate the reliability, the effectiveness, and the significance of over-fitting of these feature components. Furthermore, such analysis helps to improve the performance of DNNs. As a generic method, the feature complexity also provides new insights into existing deep-learning techniques, such as network compression and knowledge distillation. *We will release the code when the paper is accepted.*

1 INTRODUCTION

Deep neural networks (DNNs) have demonstrated significant success in various tasks. Besides the superior performance of DNNs, some attempts have been made to investigate the interpretability of DNNs in recent years. Previous studies of interpreting DNNs can be roughly summarized into two types, *i.e.* the explanation of DNNs in a post-hoc manner (Lundberg & Lee, 2017; Ribeiro et al., 2016), and the analysis of the representation capacity of a DNN (Higgins et al., 2017; Achille & Soatto, 2018a;b; Fort et al., 2019; Liang et al., 2019).

This study focuses on a new perspective of analyzing the representation capacity of DNNs. *I.e.* we define, visualize, and analyze the complexity of features in DNNs. Previous research usually analyzed the maximum complexity of a DNN according to network architectures (Arora et al., 2016; Zhang et al., 2016; Raghu et al., 2017; Manurangsi & Reichman, 2018). In comparison, we propose to measure the complexity of features by analyzing the complexity of nonlinear transformations. The actual complexity of nonlinear transformations is usually different from the maximum complexity computed based on the network architecture.

In this paper, given the feature of a specific intermediate layer, we define the complexity of this feature as the minimum number of nonlinear transformations required to compute this feature, when we constrain the network to have a fixed width. However, the quantification of nonlinear transformations presents significant challenges to state-of-the-art algorithms. Thus, we use the number of nonlinear layers to approximate the feature complexity. *I.e.* if a feature component can be computed using k nonlinear layers with a fixed width, but cannot be computed with $k-1$ nonlinear layers, we consider its complexity to be of the k-th order.

Analyzing DNNs using feature complexity. Based on the above definition, we disentangle an intermediate-layer feature into feature components of different complexity orders, as Figure 1 shows. The clear disentanglement of feature components enables both qualitative and quantitative analysis of a DNN as follows.

• We first visualize feature components of different complexity orders. Then, we explore the relationship between the feature complexity and the difficulty of the task. The distribution of feature components of different complexity orders potentially reflects the difficulty of the task. A simple task usually makes the DNN mainly learn simple features.

• We further analyze the reliability, the effectiveness, and the significance of over-fitting for the disentangled feature components: (1) In this paper, *reliable* feature components refer to features that

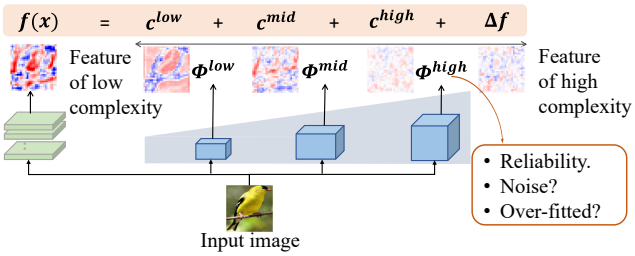


Figure 1: We disentangle the raw feature into feature components of different complexity orders. We further visualize and analyze the feature components using some generic metrics.

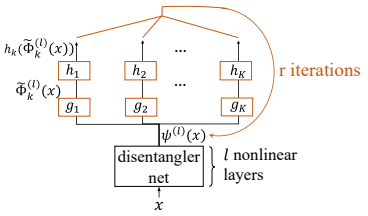


Figure 2: The network for the disentanglement of reliable feature components.

can be stably learned for the same task by DNNs with different architectures and parameters. (2) *The effectiveness of a feature component* is referred to as whether the feature component corresponds to neural activations relevant to the task. Usually, irrelevant neural activations can be considered as noises. (3) *The significance of over-fitting of a feature component* represents whether the feature component is over-fitted to specific training samples. In this paper, the significance of over-fitting is quantified as the difference between a feature component's numerical contribution to the decrease of the training loss and its contribution to the decrease of the testing loss.

We obtain the following two conclusions based on the above analysis: First, the number of training samples has small influence on the distribution of feature components, but significant impacts on the feature reliability and the significance of over-fitting of feature components. Second, feature components of the complexity order, which is about the half of the depth of DNNs, are usually more effective in inference than other feature components.

• Above conclusions can be further used to improve the performance of DNNs. We use feature components of low complexity orders, especially feature components with high effectiveness and reliability, to improve the performance of DNNs.

Method. More specifically, the disentanglement of feature components of different complexity orders is inspired by knowledge distillation (Hinton et al., 2015). We consider the target DNN as the teacher network. Then, we design several neural networks (namely disentangler nets) with different depths to mimic the feature in an intermediate layer of the teacher network. In this way, we assume that feature components mimicked by shallow disentangler nets usually correspond to those of low complexity. A deeper disentangler net can incrementally learn an additional feature component of a bit higher complexity order, besides components of low complexity.

In addition, disentangler nets with different widths usually provide consistent disentanglement of feature components. We find that the moderate change of the width of disentangler nets does not significantly affect the distribution of feature components of different complexity orders. The proposed method can be widely applied to DNNs learned for different tasks with different architectures. The disentanglement of feature components provides insightful explanations for network compression and knowledge distillation.

Contributions. Our contributions can be summarized as follows: (1) We propose a method to disentangle, visualize, and analyze the complexity of intermediate-layer features in a DNN. We measure the minimum number of nonlinear transformations actually used to compute the feature, which is usually different from the maximum complexity of a DNN computed based on its architecture. (2) We visualize the disentangled feature components of different complexity orders. (3) We propose new metrics to analyze these feature components in terms of the reliability, the effectiveness, and the significance of over-fitting. Such metrics provide insightful analysis of advantages and disadvantages of the network compression and the knowledge distillation. (4) The disentangled feature components improve the performance of DNNs.

2 Related Work

In this section, we discuss related studies in the scope of interpreting DNNs.

Visual explanations for DNNs. The most direct way to interpret DNNs includes the visualization of the knowledge encoded in intermediate layers of DNNs (Zeiler & Fergus, 2014; Simonyan

et al., 2017; Yosinski et al., 2015; Mahendran & Vedaldi, 2015; Dosovitskiy & Brox, 2016), and the estimation of the pixel-wise attribution/importance/saliency on input images (Ribeiro et al., 2016; Lundberg & Lee, 2017; Kindermans et al., 2017; Fong & Vedaldi, 2017; Zhou et al., 2016; Selvaraju et al., 2017; Chattopadhay et al., 2018; Zhou et al., 2015). They visualized salient regions in input images or salient feature units. In comparison, we propose to disentangle and visualize feature components of different complexity orders, which provides a new perspective to understand DNNs.

Explanations for the representation capacity of DNNs. The evaluation of the representation capacity of DNNs provides a new perspective for explanations. The information-bottleneck theory (Wolchover, 2017; Shwartz-Ziv & Tishby, 2017) used the mutual information to evaluate the representation capacity of DNNs (Goldfeld et al., 2019; Xu & Raginsky, 2017). Achille & Soatto (2018b) further used the information-bottleneck to constrain the feature representation. Chen et al. (2018) proposed instance-wise feature selection for model interpretation. The CLEVER score (Weng et al., 2018) was used to estimate the robustness of DNNs. The stiffiness (Fort et al., 2019), the Fourier analysis (Xu, 2018), and the sensitivity metrics (Novak et al., 2018) were proposed to analyze the generalization capacity of DNNs. The canonical correlation analysis (CCA) (Kornblith et al., 2019) was used to measure the similarity between feature representations of DNNs. Liang et al. (2019) investigated the knowledge consistency between different DNNs.

Unlike previous methods, our research aims to explain a DNN from the perspective of feature complexity. In comparison, previous methods mainly analyzed the difficulty of optimizing a DNN (Arora et al., 2016; Blum & Rivest, 1989; Boob et al., 2018), the architectural complexity (Zhang et al., 2016), and the representation complexity (Liang et al., 2017; Cortes et al., 2017; Raghu et al., 2017), which are introduced as follows.

• Difficulty or computational complexity of optimizing a DNN: Some studies focus on the amount of computation, which is required to ensure a certain accuracy of tasks. Blum & Rivest (1989); Livni et al. (2014); Boob et al. (2018); Manurangsi & Reichman (2018) proved that learning a neural network with one or two hidden layers was NP-hard in the realizable case. Arora et al. (2016) showed that a ReLU network with a single hidden layer could be trained in polynomial time when the dimension of input was constant. Based on topological concepts, Bianchini & Scarselli (2014) proposed to evaluate the complexity of functions implemented by neural networks. Rolnick & Tegmark (2017) focused on the number of neurons required to compute a given function for a network with a fixed depth.

• Complexity measures of the feature representation in DNNs: Pascanu et al. (2013); Zhang et al. (2016) proposed three architectural complexity measures for RNNs. Raghu et al. (2017) proved the maximal complexity of features grew exponentially with depth. Liang et al. (2017); Cortes et al. (2017) measured the maximal complexity of DNNs with Rademacher complexity. Kalimeris et al. (2019) investigated the change of the the mutual information between features in a DNN and features in a linear classifier. They found that during the learning process, the SGD optimizer learned functions of increasing complexity.

Unlike investigating the maximal complexity of DNNs based on the network architecture, we measure the feature complexity by exploring the complexity of nonlinear transformations, and visualize feature components of different complexity orders. Moreover, we analyze the quality of feature components and successfully boost the performance of DNNs with these feature components.

3 ALGORITHM

3.1 COMPLEXITY OF FEATURE COMPONENTS

Given an input image x, let $f(x) \in \mathbb{R}^n$ denote the feature of a specific intermediate layer of the DNN. $y = g(f(x)) \in \mathbb{R}^C$ is the output of the DNN, where C denotes the number of categories in the classification task. In this study, we define the complexity of feature components as the minimum number of nonlinear transformations required to compute feature components. The disentanglement of feature components of different complexity orders in Figure 1 can be represented as follows.

$$f(x) = c^{(1)}(x) + c^{(2)}(x) + \ldots + c^{(L)}(x) + \Delta f \tag{1}$$

where $c^{(l)}(x)$ denotes the feature component of the l-th complexity order (or, the *l-order complexity* for short). Δf is the feature component with higher-order complexity.

Definition. The feature component c of the l-order complexity is defined as the feature component that can be computed using l nonlinear layers, but cannot be computed with $l-1$ nonlinear layers, when we constrain the network Φ to have a fixed width. *I.e.* $l = \text{argmin}_{l',\Phi}\{\Phi^{(l')}(x) = c\}$, where $\Phi^{(l')}(\cdot)$ denotes a neural network with l' nonlinear transformation layers.

Instead of directly disentangling the feature component $c^{(l)}$, we propose to use knowledge distillation to extract all feature components with the complexity of no higher than the l-th order, i.e. $\Phi^{(l)}(x) = \sum_{i=1}^{l} c^{(i)}(x)$. Given a trained DNN as the teacher (target) network, we select an intermediate layer f of the DNN as the target layer. $\Phi^{(l)}(x) = \sum_{i=1}^{l} c^{(i)}(x)$ is disentangled using another DNN (termed the *disentangler net*) with l nonlinear layers. The MSE loss $\|f(x) - \Phi^{(l)}(x)\|^2$ is used to force $\Phi^{(l)}(x)$ to mimic the target feature $f(x)$, where $f(x)$ denotes the feature of the target network. We use disentangler nets with different depths $\Phi^{(1)}, \Phi^{(2)}, \ldots, \Phi^{(L)}$ to extract feature components of different complexity orders. In this way, the feature component of the l-order complexity is given as follows.

$$Loss = \|f(x) - \Phi^{(l)}(x)\|^2, \qquad c^{(l)}(x) = \Phi^{(l)}(x) - \Phi^{(l-1)}(x) \tag{2}$$

In particular, $c^{(1)}(x) = \Phi^{(1)}(x)$. Thus, $f(x)$ is disentangled into two parts: $f(x) = \Phi^{(L)}(x) + \Delta f$ where Δf denotes the feature component with a higher complexity order than L.

Significance of feature components ($\rho_c^{(l)}$). Furthermore, we quantify the significance of feature components of different complexity orders as the variance of feature components. The metric is designed as $\rho_c^{(l)} = Var[c^{(l)}(x)]/Var[f(x)]$, where $Var[c^{(l)}(x)] = \mathbb{E}_x[\|c^{(l)}(x) - \mathbb{E}_{x'}[c^{(l)}(x')]\|^2]$. For fair comparisons, we use the variance of the entire feature $f(x)$ to normalize $Var[c^{(l)}(x)]$. The variance indicates the numerical impact of the feature component $c^{(l)}(x)$ to $f(x)$. $\rho_c^{(l)}$ represents the significance of the l-th order complex feature component *w.r.t.* the entire feature.

Limitations: accurate estimation vs. fair comparison. Theoretically, if the target DNN has D nonlinear transformation layers, the complexity of its features must be no higher than the D-th order, *i.e.* $\Phi^{(D')}(x) = f(x)$, $D' \leq D$. However, the optimization capacity for the learning of disentangler nets is limited. A disentangler net with D nonlinear layers cannot learn all features encoded in $f(x)$. Thus, when $\Phi^{(D')} \approx f(x)$ in real implementations, we have $D' \geq D$.

In this way, $\rho_c^{(l)}$ measures the relative distribution of feature components of different complexity orders, instead of an accurate strength of feature components. Nevertheless, as Figure 4 shows, even if we use disentangler nets with different architectures (different widths), we still get similar distributions of feature components. This proves the trustworthiness of our method, and enables the fair comparison of feature complexity between different DNNs.

Disentangler nets. We design disentangler nets $\Phi^{(1)}(x), \ldots, \Phi^{(L)}(x)$ with residual architectures (actually, we also use disentangler nets without skip-connections to demonstrate the trustworthiness of the distribution of feature components. Please see Appendix B). The disentangler net consists of three types of residual blocks, each type having m blocks. Each block of the three types consists of a ReLU layer and a convolutional layer with $128r, 256r, 512r$ channels, respectively. In most experiments, we set $r = 1$, but in Figure 4, we try different values of r to test the performance of different disentangler nets. We use two additional convolutional layers before and after all $3m$ blocks, respectively, to match the input and output dimensions. Therefore, a disentangler net contains $3m + 2$ convolutional layers and $l = 3m + 1$ ReLU layers. Figure 10 shows the diagram of the disentangler net with the residual architecture. For fair comparisons between DNNs, we use the same set of disentangler nets to measure the complexity of each DNN.

Various disentangler nets generate similar distributions of feature components, which demonstrates the trustworthiness of our methods. We learn a target DNN for Task-26 (which will be introduced later) on the CIFAR-10 dataset and disentangle feature components from the output feature of the target DNN. We use disentangler nets with different widths (different values of r) for analysis. We analyze the complexity of the output feature of the last convolutional layer. We set $m = 1, 2, 4, 8, 16, 32$, so that the nonlinear layer numbers of disentangler nets are $l = 4, 7, 13, 25, 49, 97$. Considering the computational cost, we calculate $c^{(4)}(x) = \Phi^{(4)}(x), c^{(7)}(x) = \Phi^{(7)}(x) - \Phi^{(4)}(x), c^{(13)}(x) = \Phi^{(13)}(x) - \Phi^{(7)}(x)$, etc. This approximation does not affect the objectiveness of the quantified distribution of feature components of different complexity orders. Figure 4 compares distributions of feature components disentangled by different disentanglers. As can be observed, disentangler nets with different widths generate similar distributions of feature components, thereby verifying the trustworthiness of our method. As an extended experiment in Appendix B, we also conduct such

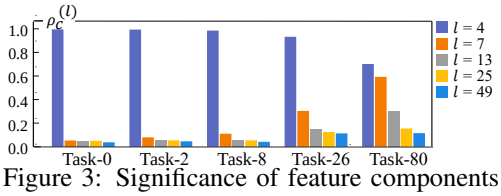


Figure 3: Significance of feature components of DNNs learned for different tasks.

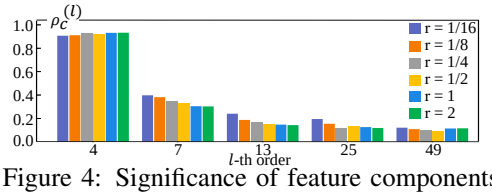


Figure 4: Significance of feature components disentangled by different disentangler nets.

experiments using disentangler nets without skip-connections to demonstrate the trustworthiness of the metric $\rho_c^{(l)}$.

The relationship between the task complexity and the feature complexity. We define the complexity of tasks first. Let *Task-n* denote a task of the n-order complexity as follows: we construct another network (namely the task DNN) with n ReLU layers and randomly initialized parameters, whose output is an $8 \times 8 \times 64$ tensor. We learn the target DNN[1] to reconstruct this output tensor via an MSE loss. Since the task DNN contains n ReLU layers, we use Task-n to indicate the complexity of mimicking the task DNN. Figure 3 compares distributions of feature components disentangled from target DNNs learned for Task-0, Task-2, Task-8, Task-26, and Task-80. DNNs learned for more complex tasks usually encode more high-complexity feature components. A an extended experiment in Appendix B, we also use disentangler nets without skip-connections for these tasks.

3.2 Reliability of Feature Components

To evaluate the reliability of a set of feature components $\Phi^{(l)}(x) = \sum_{i=1}^{l} c^{(i)}(x)$, we disentangle reliable feature components $\Phi^{(l),\text{reli}}(x)$ and unreliable feature components $\Phi^{(l),\text{unreli}}(x)$ as follows.

$$\Phi^{(l)}(x) = \Phi^{(l),\text{reli}}(x) + \Phi^{(l),\text{unreli}}(x) \tag{3}$$

As discussed in (Liang et al., 2019), DNNs with different initializations of parameters usually learn some similar feature representations for the same task, and these similar features are proved to be reliable for the task. Thus, we consider the reliable feature components as features that can be stably learned by different DNNs trained for the same task. Suppose that we have K different DNNs learned for the same task. For each DNN, we select the feature of a specific intermediate layer as the target feature. Let $f_1(x), f_2(x), \ldots, f_K(x)$ denote target features of K DNNs. We aim to extract features shared by $f_1(x), f_2(x), \ldots, f_K(x)$, *i.e.* disentangling $\Phi_1^{(l),\text{reli}}(x), \Phi_2^{(l),\text{reli}}(x), \ldots, \Phi_K^{(l),\text{reli}}(x)$ from features of K DNNs as reliable components, respectively. For each pair of DNNs (i, j), $\Phi_i^{(l),\text{reli}}(x)$ and $\Phi_j^{(l),\text{reli}}(x)$ are supposed to be able to reconstruct each other by a linear transformation:

$$\Phi_i^{(l),\text{reli}}(x) = r_{j\to i}(\Phi_j^{(l),\text{reli}}(x)), \quad \Phi_j^{(l),\text{reli}}(x) = r_{i\to j}(\Phi_i^{(l),\text{reli}}(x)) \tag{4}$$

where $r_{i\to j}$ and $r_{j\to i}$ denote two linear transformations.

Implementations. Inspired by the CycleGAN (Zhu et al., 2017), we apply the idea of cycle consistency on knowledge distillation to extract reliable feature components. To extract reliable feature components, we construct the following neural network for knowledge distillation. As Figure 2 shows, the network has a total of l ReLU layers. We add K parallel additional convolutional layers $g_1, g_2, \ldots, g_K$ to generate K outputs $\widetilde{\Phi}_1^{(l)}(x), \widetilde{\Phi}_2^{(l)}(x), \ldots, \widetilde{\Phi}_K^{(l)}(x)$, to mimic $f_1(x), f_2(x), \ldots, f_K(x)$, respectively. More specifically, $\widetilde{\Phi}_k^{(l)}(x) = g_k(\psi^{(l)}(x))$, where $\psi^{(l)}(x)$ denotes the output of the disentangler net with l ReLU layers. Then, the distillation loss is given as $\mathcal{L}^{\text{distill}} = \sum_{k=1}^{K} \|f_k(x) - \widetilde{\Phi}_k^{(l)}(x)\|^2$.

For the cycle consistency, we use $\widetilde{\Phi}_k^{(l)}(x)$ to reconstruct $\psi^{(l)}(x)$ by another linear transformation h_k: $h_k(\widetilde{\Phi}_k^{(l)}(x)) = h_k(g_k(\psi^{(l)}(x))) \to \psi^{(l)}(x)$. We conduct cycle reconstructions between $\psi^{(l)}(x)$ and $\widetilde{\Phi}_k^{(l)}(x)$ for R iterations ($R = 10$ in experiments) to ensure a certain reconstruction accuracy. Let $\psi_0^{(l)}(x) = \psi^{(l)}(x), \psi_r^{(l)}(x) = \mathbb{E}_k[h_k \circ g_k \circ \psi_{r-1}^{(l)}(x)]$ denote the reconstruction output in the r-th iteration, where $h_k \circ g_k$ denotes the cascaded layerwise operations. The cycle construction loss is as follows.

$$\mathcal{L}^{\text{cycle}} = \sum\nolimits_{r=1}^{R} \sum\nolimits_{k=1}^{K} \|h_k \circ g_k \circ \psi_{r-1}^{(l)}(x) - \psi_{r-1}^{(l)}(x)\|^2 \tag{5}$$

Appendix C provides more insightful understanding of the motivation and the explanation for techniques used in Eq. (5).

[1]For simplicity, we design the target DNN to have the same architecture as the disentangler net with $l = 19$.

This loss makes the feature $\widetilde{\Phi}_k^{(l)}(x)$ approximately shared by K DNNs. In this way, $\Phi_k^{(l),\text{reli}}(x) = \widetilde{\Phi}_k^{(l)}(x)$ can be considered as the reliable feature component. Compared with the traditional cycle consistency (Zhu et al., 2017), the above loss is much simpler and requires less computational cost. In this way, we can disentangle the unreliable feature component of the k-th DNN as $\Phi_k^{(l),\text{unreli}}(x) = \Phi_k^{(l)}(x) - \Phi_k^{(l),\text{reli}}(x)$, with the other $K-1$ DNNs as assistant DNNs. In experiments, we set $K=3$, including a target DNN and DNNs A and B. The reliable feature components are shared by the three DNNs. DNNs A and B have been well-trained as two additional assistant DNNs, namely *exemplary DNNs*, in order to disentangle reliable and unreliable feature components from the target DNN. The exemplary DNNs A and B are selected as those with state-of-the-art performance in the target task, in order to obtain convincing results. To enable fair comparisons, the same pair of DNNs A and B are uniformly used to analyze various DNNs.

Reliability of feature components in $\Phi_k^{(l)}(x)$ can be quantified as the ratio of reliable feature components in $\Phi_k^{(l)}(x)$ as $\rho^{(l),\text{reli}} = Var[\Phi_k^{(l),\text{reli}}(x)]/Var[\Phi_k^{(l)}(x)]$.

Effectiveness of feature components ($\alpha_{\text{effective}}^{(l)}$) measures whether the feature components $c^{(l)}(x)$ extracted from the training sample x directly contributes to the task. We define this metric as the numerical contribution of each feature component $c^{(l)}(x)$ to the decrease of the task loss. Based on the Shapley value (Shapley, 1953; Lundberg & Lee, 2017), numerical contributions of all the L feature components can be fairly allocated and given as $\varphi_1^{\text{train}} + \varphi_2^{\text{train}} + \cdots + \varphi_L^{\text{train}} = \mathbb{E}_{x\in X_{\text{train}}}[\mathcal{L}(\Delta f_x) - \mathcal{L}(\Delta f_x + \Phi^{(L)}(x))]$. The Shapley value has been widely used as a standard metric for the feature importance in literature (Chen et al., 2019; Ghorbani & Zou, 2019; Williamson & Feng, 2020). Δf_x is the high-order component within the sample x in Equation (1). $\mathcal{L}(\Delta f_x)$ represents the task loss when we remove all feature components in $\Phi^{(L)}(x)$, and $\mathcal{L}(\Delta f_x + \Phi^{(L)}(x))$ denotes the task loss when both Δf_x and feature components in $\Phi^{(L)}(x)$ are used for inference. In this way, $\mathbb{E}_{x\in X_{\text{train}}}[\mathcal{L}(\Delta f_x) - \mathcal{L}(\Delta f_x + \Phi^{(L)}(x))]$ can be considered as the overall numerical contribution of all the L feature components through all training samples. Thus, the metric $\alpha_{\text{effective}}^{(l)} = \varphi_l^{\text{train}}/\sqrt{Var[c^{(l)}(x)]}$ measures the normalized effectiveness of the feature component $c^{(l)}$ to the decrease of the training loss ($\sqrt{Var[c^{(l)}(x)]}$ is used for normalization). Please see Appendix D for discussions about the trustworthiness of the metric $\alpha_{\text{effective}}^{(l)}$.

Significance of over-fitting of feature components ($\alpha_{\text{overfit}}^{(l)}$) measures whether $c^{(l)}(x)$ is over-fitted to specific training samples. Similarly, we first measure the numerical contribution $\varphi_l^{\text{overfit}}$ of each feature component $c^{(l)}(x)$ to over-fitting based on the Shapley value. In this way, we have $\varphi_1^{\text{overfit}} + \varphi_2^{\text{overfit}} + \cdots + \varphi_L^{\text{overfit}} = \mathcal{L}_{\text{overfit}}(\Delta f + \Phi^{(L)}) - \mathcal{L}_{\text{overfit}}(\Delta f)$, where $\mathcal{L}_{\text{overfit}}(\Delta f + \Phi^{(L)}) = \mathbb{E}_{x\in X_{\text{test}}}[\mathcal{L}(\Delta f_x + \Phi^{(L)}(x))] - \mathbb{E}_{x\in X_{\text{train}}}[\mathcal{L}(\Delta f_x + \Phi^{(L)}(x))]$ uses the gap between the testing loss and the training loss to quantify the over-fitting caused by both feature components in Δf and $\Phi^{(L)}$. Then, the metric of the significance of over-fitting for $c^{(l)}$ is given as $\alpha_{\text{overfit}}^{(l)} = \varphi_l^{\text{overfit}}/\varphi_l^{\text{train}}$. Please see Appendix D for discussions about the trustworthiness of the metric $\alpha_{\text{overfit}}^{(l)}$.

4 EXPERIMENTS

Datasets, DNNs & Implementation details. We used our method to analyze VGG-16 (Simonyan et al., 2017) and ResNet-8/14/18/20/32/34/44 (He et al., 2016).[2] For simplification, we limited our attention to coarse-grained and fine-grained object classification. We trained these DNNs based on the CIFAR-10 dataset (Krizhevsky et al., 2009), the CUB200-2011 dataset (Wah et al., 2011), and the Stanford Dogs dataset (Khosla et al., 2011). For the CUB200-2011 dataset and the Stanford Dogs dataset, we used object images cropped by object bounding boxes for both training and testing. The classification accuracy of learned DNNs is shown in Appendix E.

Visualization of feature components. Given a pre-trained VGG-16 and input images in the CUB200-2011 dataset, we disentangled and visualized feature components of different orders in Figure 5. We took the feature in the `conv4-3` layer (with the size of $28 \times 28 \times 512$) as the target feature $f(x)$. Then, we disentangled the target feature and visualized the feature map of a random

[2]Compared with the original VGG-16, we added a BatchNorm layer before the output feature of each convolutional layer, before we use its feature to guide the distillation process. ResNet-8 and ResNet-14 had the similar structure as ResNet-20, ResNet-32 and ResNet-44 in (He et al., 2016), except that they had 1 and 2 blocks in each stage, respectively.

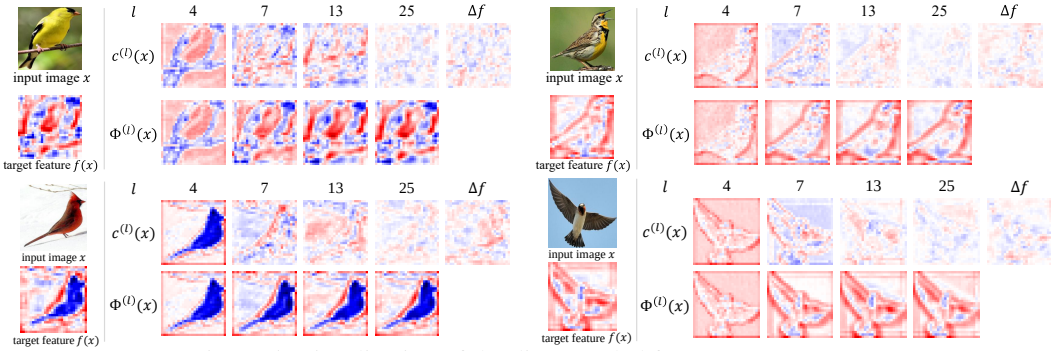


Figure 5: Visualization of the disentangled feature components.

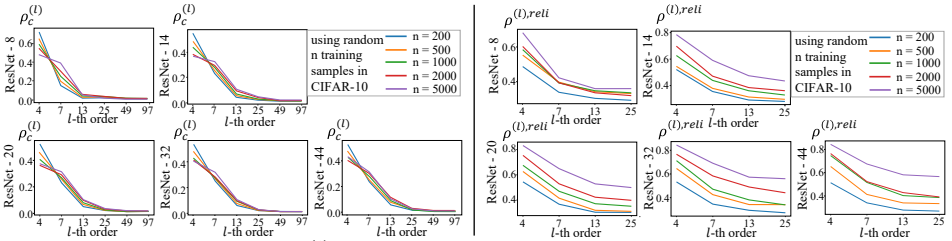


Figure 6: Significance ($\rho_c^{(l)}$) and reliability ($\rho^{(l),\text{reli}}$) of the disentangled feature components.

channel in $f(x)$, and the corresponding channel in $c^{(l)}(x)$ and $\Phi^{(l)}(x)$. Low-complexity feature components usually represented the general shape of objects, while high-complexity feature components corresponded to detailed shape and noises.

Exp. 1, the number of training samples had small influence on the distribution of feature components, but had significant impacts on the feature reliability. We learned ResNet-8/14/20/32/44 using different numbers of training samples, which were randomly sampled from the the CIFAR-10 dataset. Then, we disentangled feature components of different complexity orders from the output feature of the last residual block. More specifically, two exemplary DNNs A and B were used to help us extract the reliable feature components in the target feature. They were implemented as ResNet-44 learned on the entire CIFAR-10 dataset with different initial parameters.

Figure 6 compares the significance of disentangled feature components $\rho_c^{(l)}$ and the reliability of feature components $\rho^{(l),\text{reli}}$ in different DNNs. The DNN learned from the larger training set usually encoded more complex features, but the overall distribution of feature components was very close to the DNN learned from the smaller training set. This indicated that the number of training samples had small impacts on the significance of feature components of different complexity orders. However, in Figure 6 (right), DNNs learned from many training samples always exhibited higher reliability than DNNs learned form a few training samples, which meant that the increase of the number of training samples would help DNN learn more reliable features. Beyond Figure 6, Appendix G shows more discussions about the result in Figure 6 and the extended experiments towards the same conclusions on the CUB200-2011 dataset and the Stanford Dogs dataset.

Exp. 2, improvement of the classification accuracy based on $\Phi^{(l)}(x)$. We compared the effectiveness $\alpha_{\text{effective}}^{(l)}$ and the significance of over-fitting $\alpha_{\text{overfit}}^{(l)}$ of feature components disentangled from different DNNs in Figure 7. We found that **(1)** when the complexity order of feature components is about the half of the depth of the DNN, these feature components exhibited the highest effectiveness. **(2)** Low-complexity feature components learned from a small number of samples were usually more over-fitted than low-complexity feature components learned from many samples. However, we could not summarize clear conclusions from the significance of over-fitting for high-complexity feature components. This might be due to the low effectiveness of these feature components. Please see Appendix H for more discussions about Figure 7.

Based on above observations, we further tested the classification accuracy of DNNs by directly replacing the original target feature $f(x)$ with $\Phi^{(l)}(x)$. Figure 9(a) shows the accuracy improvement using $\Phi^{(l)}(x)$ when $l = 7$. Figure 9(b) shows that as the complexity order l increased, the accuracy improvement first increased, and then decreased. This was because the very few feature components of the lowest complexity did not contain enough information for the classification, while high-

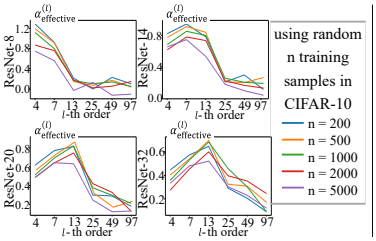


Figure 7: (left) The effectiveness of feature components $\alpha^{(l)}_{\text{effective}}$; (right) The significance of feature components being over-fitted $\alpha^{(l)}_{\text{overfit}}$.

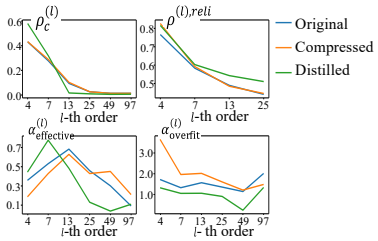


Figure 8: Comparisons between the original DNN, the compressed DNN, and the distilled DNN.

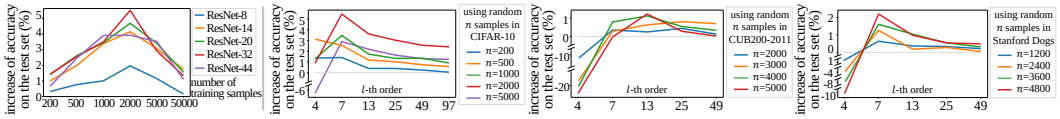


Figure 9: Improvements of the classification accuracy based on $\Phi^{(l)}(x)$. (left) The accuracy improvement of different DNNs learned on the CIFAR-10 dataset with $l = 7$ using different number of training samples. (right) The accuracy improvement with different values of l. Here ResNet-32 was learned on the CIFAR-10 dataset, and ResNet-34 was learned on the CUB200-2011 dataset and the Stanford Dogs dataset, respectively.

complexity feature components had high significance of over-fitting. Appendix I provides a detailed discussion about the improvement of the accuracy.

Exp. 3, analysis of network compression and knowledge distillation. We learned the ResNet-32 on the CIFAR-10 dataset as the originial DNN. We used the compression algorithm (Han et al., 2015) to learn another DNN (termed the *compressed DNN*) by pruning and quantizing the original DNN. For the knowledge distillation, we used another network (termed the *distilled DNN*)[3], to distill (Hinton et al., 2015) the output feature of the last residual block in the original DNN. Appendix J summarizes technique details of network compression and knowledge distillation provided in (Han et al., 2015; Hinton et al., 2015). We compared the compressed DNN and the distilled DNN with the original DNN. We disentangled feature components from the output feature of the last residual block in the original DNN and the compressed DNN, and the output feature of the distilled DNN.

Figure 8 shows $\rho^{(l)}_c$, $\rho^{(l),\text{reli}}$, $\alpha^{(l)}_{\text{effective}}$, and $\alpha^{(l)}_{\text{overfit}}$ in three DNNs. For the compressed DNN, **(1)** the network compression did not affect the distribution of feature components and their reliability. **(2)** Low-complexity feature components in the compressed DNN exhibited lower effectiveness and higher significance of over-fitting than low-complexity feature components in the original DNN.

For the knowledge distillation, **(1)** the distilled DNN had more low-complexity feature components than the original DNN. The low-complexity feature components in the distilled DNN were more effective than those in the original DNN. **(2)** High-complexity feature components in the distilled DNN were more reliable and less over-fitted than high-complexity feature components in the original DNN. These results demonstrated that the knowledge distillation would help DNNs learn more reliable features, which prevented over-fitting.

Inspired by Figure 8, we thought there was a close relationship between the feature complexity and the performance of DNNs. We conducted experiments to discover this relationship, and further used the feature complexity to predict the performance of DNNs in Appendix K.

5 Conclusion

In this paper, we have proposed a generic definition of the feature complexity of DNNs. We design a method to disentangle and visualize feature components of different complexity orders, and analyze the disentangled feature components from three perspectives. Then, a close relationship between the feature complexity and the performance of DNNs is discovered. Furthermore, the disentangled feature components can improve the classification accuracy of DNNs. As a generic tool, the feature complexity provides a new perspective to explain existing deep-learning techniques, which has been validated by experiments.

[3]The distilled DNN had the same architecture with the disentangler net with 7 ReLU layers.

REFERENCES

Alessandro Achille and Stefano Soatto. Emergence of invariance and disentanglement in deep representations. *The Journal of Machine Learning Research*, 19(1):1947–1980, 2018a.

Alessandro Achille and Stefano Soatto. Information dropout: Learning optimal representations through noisy computation. *IEEE transactions on pattern analysis and machine intelligence*, 40 (12):2897–2905, 2018b.

Marco Ancona, Cengiz Öztireli, and Markus Gross. Explaining deep neural networks with a polynomial time algorithm for shapley values approximation. *arXiv preprint arXiv:1903.10992*, 2019.

Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. Understanding deep neural networks with rectified linear units. *arXiv preprint arXiv:1611.01491*, 2016.

Monica Bianchini and Franco Scarselli. On the complexity of neural network classifiers: A comparison between shallow and deep architectures. *IEEE transactions on neural networks and learning systems*, 25(8):1553–1565, 2014.

Avrim Blum and Ronald L Rivest. Training a 3-node neural network is np-complete. In *Advances in neural information processing systems*, pp. 494–501, 1989.

Digvijay Boob, Santanu S Dey, and Guanghui Lan. Complexity of training relu neural network. *arXiv preprint arXiv:1809.10787*, 2018.

Aditya Chattopadhay, Anirban Sarkar, Prantik Howlader, and Vineeth N Balasubramanian. Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 839–847. IEEE, 2018.

Jianbo Chen, Le Song, Martin Wainwright, and Michael Jordan. Learning to explain: An information-theoretic perspective on model interpretation. In *International Conference on Machine Learning*, pp. 882–891, 2018.

Jianbo Chen, Le Song, Martin J. Wainwright, and Michael I. Jordan. L-shapley and c-shapley: Efficient model interpretation for structured data. *In ICLR*, 2019.

Corinna Cortes, Xavier Gonzalvo, Vitaly Kuznetsov, Mehryar Mohri, and Scott Yang. Adanet: Adaptive structural learning of artificial neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 874–883. JMLR. org, 2017.

Alexey Dosovitskiy and Thomas Brox. Inverting visual representations with convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4829–4837, 2016.

Ruth C Fong and Andrea Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3429–3437, 2017.

Stanislav Fort, Paweł Krzysztof Nowak, and Srini Narayanan. Stiffness: A new perspective on generalization in neural networks. *arXiv preprint arXiv:1901.09491*, 2019.

Amirata Ghorbani and James Zou. Data shapley: Equitable valuation of data for machine learning. *In ICML*, 2019.

Ziv Goldfeld, Ewout Van Den Berg, Kristjan Greenewald, Igor Melnyk, Nam Nguyen, Brian Kingsbury, and Yury Polyanskiy. Estimating information flow in deep neural networks. In *International Conference on Machine Learning*, pp. 2299–2308, 2019.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. *In ICLR*, 2(5):6, 2017.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Dimitris Kalimeris, Gal Kaplun, Preetum Nakkiran, Benjamin Edelman, Tristan Yang, Boaz Barak, and Haofeng Zhang. Sgd on neural networks learns functions of increasing complexity. In *Advances in Neural Information Processing Systems 32*, pp. 3496–3506. Curran Associates, Inc., 2019.

Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Li Fei-Fei. Novel dataset for fine-grained image categorization. In *First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition*, Colorado Springs, CO, June 2011.

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.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. *arXiv preprint arXiv:1905.00414*, 2019.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Ruofan Liang, Tianlin Li, Longfei Li, and Quanshi Zhang. Knowledge consistency between neural networks and beyond. In *International Conference on Learning Representations*, 2019.

Tengyuan Liang, Tomaso Poggio, Alexander Rakhlin, and James Stokes. Fisher-rao metric, geometry, and complexity of neural networks. *arXiv preprint arXiv:1711.01530*, 2017.

Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. On the computational efficiency of training neural networks. In *Advances in neural information processing systems*, pp. 855–863, 2014.

Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, pp. 4765–4774, 2017.

Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5188–5196, 2015.

Pasin Manurangsi and Daniel Reichman. The computational complexity of training relu (s). *arXiv preprint arXiv:1810.04207*, 2018.

Roman Novak, Yasaman Bahri, Daniel A Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Sensitivity and generalization in neural networks: an empirical study. *arXiv preprint arXiv:1802.08760*, 2018.

Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*, 2013.

Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl Dickstein. On the expressive power of deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2847–2854. JMLR. org, 2017.

Marco Túlio 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.

David Rolnick and Max Tegmark. The power of deeper networks for expressing natural functions. *arXiv preprint arXiv:1705.05502*, 2017.

Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 618–626, 2017.

Lloyd S Shapley. A value for n-person games. *Contributions to the Theory of Games*, 2(28):307–317, 1953.

Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*, 2017.

K Simonyan, A Vedaldi, and A Zisserman. Deep inside convolutional networks: visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2017.

Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. 2011.

Robert J Weber. Probabilistic values for games. *The Shapley Value. Essays in Honor of Lloyd S. Shapley*, pp. 101–119, 1988.

Tsui-Wei Weng, Huan Zhang, Pin-Yu Chen, Jinfeng Yi, Dong Su, Yupeng Gao, Cho-Jui Hsieh, and Luca Daniel. Evaluating the robustness of neural networks: An extreme value theory approach. *arXiv preprint arXiv:1801.10578*, 2018.

Brian D Williamson and Jean Feng. Efficient nonparametric statistical inference on population feature importance using shapley values. *In ICML*, 2020.

Natalie Wolchover. New theory cracks open the black box of deep learning. *In Quanta Magazine*, 2017.

Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

Aolin Xu and Maxim Raginsky. Information-theoretic analysis of generalization capability of learning algorithms. In *Advances in Neural Information Processing Systems*, pp. 2524–2533, 2017.

Zhiqin John Xu. Understanding training and generalization in deep learning by fourier analysis. *arXiv preprint arXiv:1808.04295*, 2018.

Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

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

Saizheng Zhang, Yuhuai Wu, Tong Che, Zhouhan Lin, Roland Memisevic, Ruslan R Salakhutdinov, and Yoshua Bengio. Architectural complexity measures of recurrent neural networks. In *Advances in neural information processing systems*, pp. 1822–1830, 2016.

Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Object detectors emerge in deep scene cnns. *In ICLR*, 2015.

Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2921–2929, 2016.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pp. 2223–2232, 2017.

A ARCHITECTURE OF THE DISENTANGLER NET

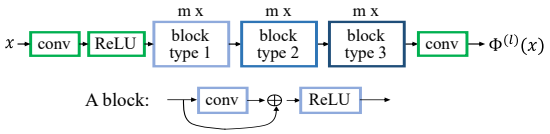


Figure 10: The architecture of the disentangler net.

B THE RELATIONSHIP BETWEEN TASK COMPLEXITY AND FEATURE COMPLEXITY & THE TRUSTWORTHINESS OF THE DISENTANGLEMENT

This section provides more discussion about the result in Figure 3 and Figure 4. Figure 3 compares distributions of feature components encoded in target DNNs that were learned for tasks of different difficulties. We found that DNNs learned form more complex tasks usually encoded more high-complexity feature components. Let us take the target DNNs learned for Task-0 and Task-80 for example. For the target DNN learned for Task-0, the significance of the 4-order feature component was much higher than that of feature components of higher orders. However, in the target DNN learned for Task-80, the significance of the 7-order and the 13-order feature components exceeded the half of the significance of the 4-order feature component. Thus, experimental results show that DNNs learned for more complex tasks usually encode more high-complexity feature components.

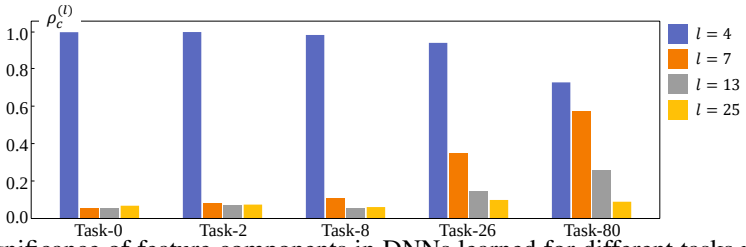


Figure 11: Significance of feature components in DNNs learned for different tasks using disentangler nets w/o skip-connections. We simply revised the disentangler nets introduced in Figure 10 by removing the skip-connections, so as to obtain the stacked disentangler nets.

In order to further show the robustness and trustworthiness of the distribution of feature components using various disentangler nets, we used disentangler nets with a different architecture from the one we used in this paper (in Figure 10). Specifically, we used a simply stacked architecture by only removing the skip-connections from the residual disentangler net we adopted in this paper. We did the same tasks mentioned in the *"the relationship between the task complexity and the feature complexity"* paragraph in Section 3.1. Figure 11 shows that distributions of feature components generated by the simply stacked disentangler nets on different tasks were similar to the distributions generated by the the residual disentangler nets in Figure 3. This consistency demonstrated the robustness of the distribution of feature components over different disentangler architectures.

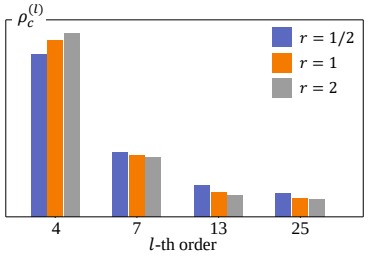


Figure 12: Significance of feature components using disentangler nets w/o skip connections of different widths. It shows that distributions of feature components generated by the simply stacked disentangler nets were similar to the distributions generated by the residual disentangler nets in Figure 4. This consistency demonstrated the trustworthiness of the metric $\rho^{(l)}$.

Figure 12 shows that distributions of feature components generated by the simply stacked disentangler nets were similar to the distributions generated by the residual disentangler nets in Figure 4. This consistency demonstrated the trustworthiness of the metric $\rho^{(l)}$.

C ABOUT THE RELIABILITY OF FEATURES

This section explains the rationality and implementation details of the algorithm in Section 3.2. For each DNN, we select the feature of a specific intermediate layer as the target feature. Let $f_1(x), f_2(x), \ldots, f_K(x)$ denote target features of K DNNs. We aim to extract reliable features of different complexity orders in the K DNNs, *i.e.* $\Phi_1^{(l),\text{reli}}(x), \Phi_2^{(l),\text{reli}}(x), \ldots, \Phi_K^{(l),\text{reli}}(x)$.

Inspired by (Liang et al., 2019), we consider each pair of the K reliable feature components is able to reconstruct each other by a linear transformation. As is mentioned in Section 3.2, $\psi^{(l)}(x)$ is the output of a disentangler net with l ReLU layers. We add K parallel additional convolutional layers $g_1, g_2, \ldots, g_K$ on top of $\psi^{(l)}(x)$ to mimic $f_1(x), f_2(x), \ldots, f_K(x)$. At this time, their outputs $\widetilde{\Phi}_k^{(l)}(x) = g_k(\psi^{(l)}(x))$ are not able to reconstruct each other linearly.

To enable $\widetilde{\Phi}_i^{(l)}(x)$ and $\widetilde{\Phi}_j^{(l)}(x)$ to reconstruct each other linearly, we first transform $\widetilde{\Phi}_i^{(l)}(x)$ to $\psi^{(l)}(x)$ by the linear regressor h_i, and then use $\psi^{(l)}(x)$ to reconstruct $\widetilde{\Phi}_j^{(l)}(x)$ by another linear regressor g_j, as shown in Figure 13. Similarly, we can use $\widetilde{\Phi}_j^{(l)}(x)$ to regress $\psi^{(l)}(x)$ via h_j, and then use $\psi^{(l)}(x)$ to linearly regress $\widetilde{\Phi}_i^{(l)}(x)$.

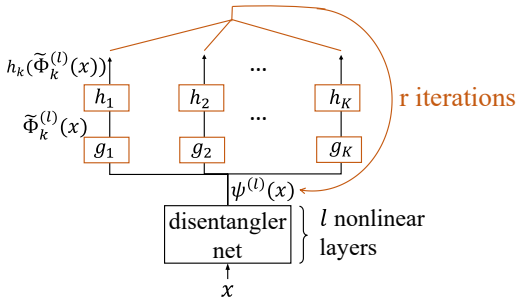


Figure 13: The network for the disentanglement of reliable feature components.

In this way, $\tilde{\Phi}_i^{(l)}(x)$ and $\tilde{\Phi}_j^{(l)}(x)$ can reconstruct each other by a linear transformation:

$$\tilde{\Phi}_i^{(l)}(x) = g_i(h_j(\tilde{\Phi}_j^{(l)}(x))), \quad \tilde{\Phi}_j^{(l)}(x) = g_j(h_i(\tilde{\Phi}_i^{(l)}(x)))$$

We repeat the reconstruction for R iterations and design the loss in Eq. (5).

In the implementation, we first train $g_1, g_2, \ldots, g_K$, and then fix g to train $h_1, h_2, \ldots, h_K$. To reduce the computational complexity, we did not explicitly optimize on the loss in Eq. (5) which requires a sum over r, from 1 to R. Instead, in each r-th training phase, we optimize $\sum_{k=1}^{K} \|h_k \circ g_k \circ \psi_r^{(l)}(x) - \psi_r^{(l)}(x)\|^2, r \in \{1, 2, \ldots, R\}$.

D METRICS OF EFFECTIVENESS & OVER-FITTING, AND THEIR TRUSTWORTHINESS

D.1 PRELIMINARY: THE SHAPLEY VALUE

The Shapley value was originally proposed in the game theory (Shapley, 1953). It has been widely used as a standard metric for the feature importance in literature. Let us consider a game with multiple players. Each player can participate in the game and receive a reward individually. Besides, some players can form a coalition and play together to pursue a higher reward. Different players in a coalition usually contribute differently to the game, thereby being assigned with different proportions of

the coalition's reward. The Shapley value is considered as a unique method that fairly allocates the reward to players with certain desirable properties (Ancona et al., 2019). Let $N = \{1, 2, \dots, n\}$ denote the set of all players, and 2^N represents all potential subsets of N. A game $v : 2^N \to \mathbb{R}$ is implemented as a function that maps from a subset to a real number. When a subset of players $S \subseteq N$ plays the game, the subset can obtain a reward $v(S)$. Specifically, $v(\emptyset) = 0$. The Shapley value of the i-th player $\phi_{i,v}^N$ can be considered as an unbiased contribution of the i-th player.

$$\phi_{i,v}^N = \sum_{S \subseteq N \setminus \{i\}} \frac{(n - |S| - 1)!\,|S|!}{n!} \Big[v(S \cup \{i\}) - v(S) \Big]$$

Weber *et al.* (Weber, 1988) have proved that the Shapley value is the only reward with the following axioms.

Linearity axiom: If the reward of a game u satisfies $u(S) = v(S) + w(S)$, where v and w are another two games. Then the Shapley value of each player $i \in N$ in the game u is the sum of Shapley values of the player i in the game v and w, *i.e.* $\phi_{i,u}^N = \phi_{i,v}^N + \phi_{i,w}^N$.

Dummy axiom: The dummy player is defined as the player that satisfies $\forall S \subseteq N \setminus \{i\}, v(S \cup \{i\}) = v(S) + v(\{i\})$. In this way, the dummy player i satisfies $v(\{i\}) = \phi_{i,v}^N$, *i.e.* the dummy player has no interaction with other players in N.

Symmetry axiom: If $\forall S \subseteq N \setminus \{i, j\}, v(S \cup \{i\}) = v(S \cup \{j\})$, then $\phi_{i,v}^N = \phi_{j,v}^N$.

Efficiency axiom: $\sum_{i \in N} \phi_{i,v}^N = v(N)$. The efficiency axiom can ensure the overall reward can be distributed to each player in the game.

D.2 Effectiveness of feature components

The effectiveness of feature components ($\alpha_{\text{effective}}^{(l)}$) measures whether the feature component $c^{(l)}(x)$ extracted from the training sample x directly contributes to the task. The metric is defined based on the game theory. We first quantify the numerical contribution φ_l^{train} of each feature component $c^{(l)}(x)$ to the decrease of the task loss in training as the Shapley value. *I.e.* φ_l^{train} measures the change of the training loss caused by feature components $\{c^{(l)}(x) | x \in X_{\text{train}}\}$. In this way, numerical contributions of all the L feature components can be allocated and given as $\varphi_1^{\text{train}} + \varphi_2^{\text{train}} + \cdots + \varphi_L^{\text{train}} = \mathbb{E}_{x \in X_{\text{train}}}[\mathcal{L}(\Delta f_x) - \mathcal{L}(\Delta f_x + \Phi^{(L)}(x))]$, where Δf_x is the high-order component computed using the sample x. $\mathcal{L}(\Delta f_x)$ represents the task loss when we remove all feature components in $\Phi^{(L)}(x)$, and $\mathcal{L}(\Delta f_x + \Phi^{(L)}(x))$ denotes the task loss when both Δf_x and feature components in $\Phi^{(L)}(x)$ are used for inference. Thus, the metric $\alpha_{\text{effective}}^{(l)} = \varphi_l^{\text{train}} / \sqrt{Var[c^{(l)}(x)]}$ measures the effectiveness of the feature component $c^{(l)}$ to the decrease of the training loss. We use $\sqrt{Var[c^{(l)}(x)]}$ for normalization.

D.3 Significance of over-fitting of feature components

The significance of over-fitting of feature components ($\alpha_{\text{overfit}}^{(l)}$) measures whether $c^{(l)}(x)$ is over-fitted to specific training samples. Similarly, this metric is also defined based on Shapley values. We quantify the numerical contribution $\varphi_l^{\text{overfit}}$ of each feature component $c^{(l)}(x)$ to over-fitting, whose significance is quantified as $\mathcal{L}_{\text{overfit}}(f) = \mathcal{L}_{\text{overfit}}(\Delta f + \Phi^{(L)}) = \mathbb{E}_{x \in X_{\text{test}}}[\mathcal{L}(\Delta f_x + \Phi^{(L)}(x))] - \mathbb{E}_{x \in X_{\text{train}}}[\mathcal{L}(\Delta f_x + \Phi^{(L)}(x))]$. In this way, the numerical contribution can also be measured as Shapley values $\varphi_1^{\text{overfit}} + \varphi_2^{\text{overfit}} + \cdots + \varphi_L^{\text{overfit}} = \mathcal{L}_{\text{overfit}}(\Delta f + \Phi^{(L)}) - \mathcal{L}_{\text{overfit}}(\Delta f)$, where $\mathcal{L}_{\text{overfit}}(\Delta f + \Phi^{(L)})$ is computed using both components Δf_x and components $\Phi^{(L)}(x)$ in different images. *I.e.* $\varphi_l^{\text{overfit}}$ measures the change of $\mathcal{L}_{\text{overfit}}$ caused by the feature component $c^{(l)}(x)$. The metric of the significance of over-fitting for $c^{(l)}$ is given as $\alpha_{\text{overfit}}^{(l)} = \varphi_l^{\text{overfit}} / \varphi_l^{\text{train}}$. Thus, $\alpha_{\text{overfit}}^{(l)}$ represents the ratio of the increase of the gap $\Delta\mathcal{L}_{\text{overfit}}$ to the decrease of the training loss $\Delta\mathcal{L}_{\text{train}}$.

E Accuracy of DNNs

This section contains more details of DNNs in Exp. 2. We trained ResNet-8/14/20/32/44 based on the CIFAR-10 dataset (Krizhevsky et al., 2009), and trained VGG-16, ResNet-18/34 based on the CUB200-2011 dataset (Wah et al., 2011) and the Stanford Dogs dataset (Khosla et al., 2011).

More specifically, we trained each DNN with different numbers of training samples, which were randomly sampled from the training set. All (target) networks were pre-trained with different parameter initialization. Table 1 reports the accuracy and loss of the prediction on the testing samples.

Table 1: Accuracy of DNNs on different datasets.

(a) On the CIFAR-10 dataset.

	Accuracy				
# of training samples	200	500	1000	2000	5000
ResNet-8	31.37%	39.55%	45.08%	53.82%	67.80%
ResNet-14	31.50%	39.21%	47.71%	52.41%	68.30%
ResNet-20	31.56%	38.40%	46.09%	56.15%	70.62%
ResNet-32	30.48%	37.94%	46.71%	56.80%	72.83%
ResNet-44	28.73%	38.57%	46.63%	56.00%	70.66%

(b) On the CUB200-2011 dataset.

	Accuracy			
# of training samples	2000	3000	4000	5000
ResNet-18	32.36%	44.82%	52.73%	56.18%
ResNet-34	29.43%	43.86%	52.17%	53.68%
VGG-16	28.18%	41.04%	47.03%	53.83%

(c) On the Stanford Dogs dataset.

	Accuracy			
# of trainingsamples	1200	2400	3600	4800
ResNet-18	10.93%	19.42%	28.51%	37.95%
ResNet-34	9.37%	18.83%	27.05%	32.23%
VGG-16	10.36%	16.78%	23.63%	29.14%

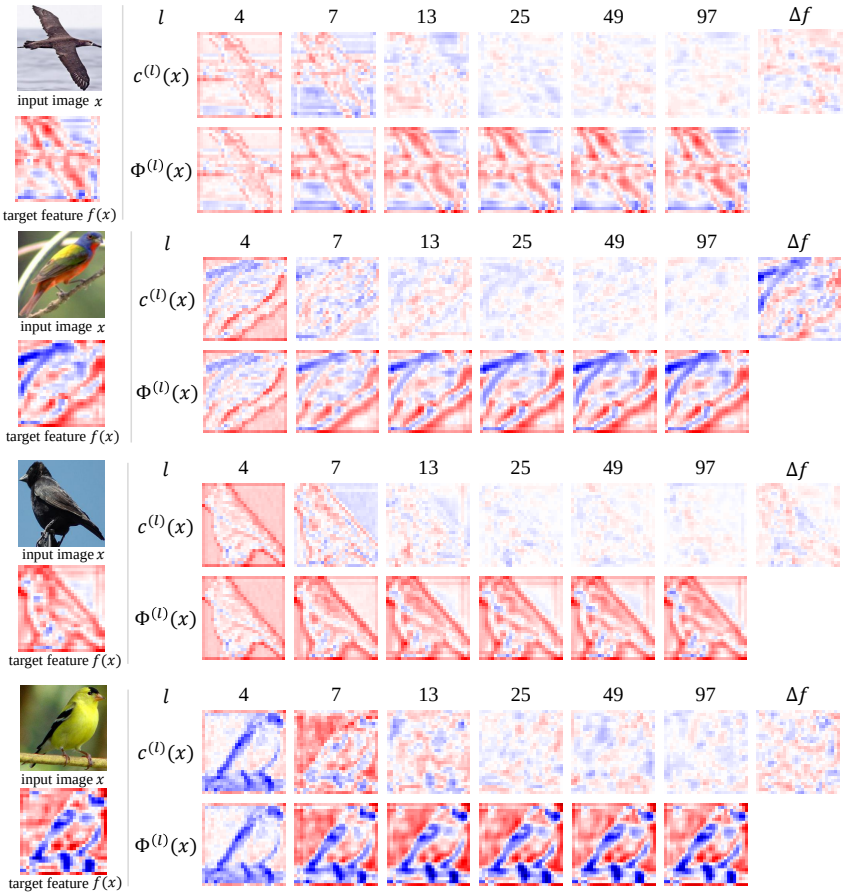


Figure 14: Visualization of feature components of different complexity orders on the CUB200-2011 dataset.

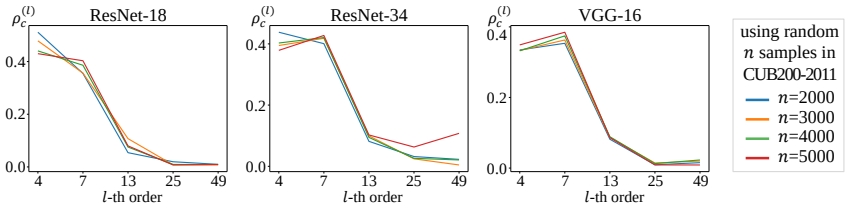
F VISUALIZATION OF FEATURE COMPONENTS

This section shows more visualization results in Section 4 by visualizing feature components disentangled from the target feature in Figure 14. Given a pre-trained VGG-16 and input images in the CUB200-2011 dataset, we disentangled and visualized feature components of different orders in Figure 5. We took the feature in the `conv4-3` layer (with the size of $28 \times 28 \times 512$) as the target feature $f(x)$. Then, we disentangled the target feature and visualized the feature map of a random channel in $f(x)$, and the corresponding channel in $c^{(l)}(x)$ and $\Phi^{(l)}(x)$. We found that low-complexity feature components usually represented the general shape of objects, while high-complexity feature components corresponded to detailed shape and noises.

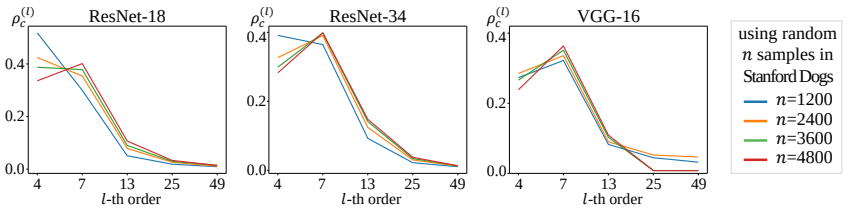
G EVALUATION OF DNNS LEARNED ON THE CUB200-2011 DATASET AND THE STANFORD DOGS DATASET.

This section provides more discussions about Exp. 1, and shows more experimental results on different datasets. In Figure 6 (left), we found that the number of training samples had a small influence on the distribution of feature components, but had significant impacts on the feature reliability. Let us take ResNet-8 for instance, when we used different numbers of training samples, the overall distributions of feature components were similar, *i.e.* low-complexity feature components exhibited higher significance than high-complexity feature components. Besides, on different target DNNs, the distributions of feature components were also similar over different target DNNs, which indicated that the depths of target DNNs also did not affect this distribution. However, In Figure 6 (right), the number of training samples significantly affected the reliability of feature components. When we used more training samples, the reliability of feature components became higher. This phenomenon was also consistent through target DNNs with different architectures.

Figure 15 shows the result of $\rho_c^{(l)}$ of ResNet-18/34 and VGG-16 learned on the CUB200-2011 dataset and the Stanford Dogs dataset. We found that the DNN learned from the larger training set usually encoded more complex features, but the overall distribution of feature components was very close to the DNN learned from the smaller training set. This indicated that the number of training samples had small impacts on the significance of feature components of different complexity orders.



(a) Significance of feature components $\rho_c^{(l)}$ in DNNs learned on the CUB200-2011 dataset

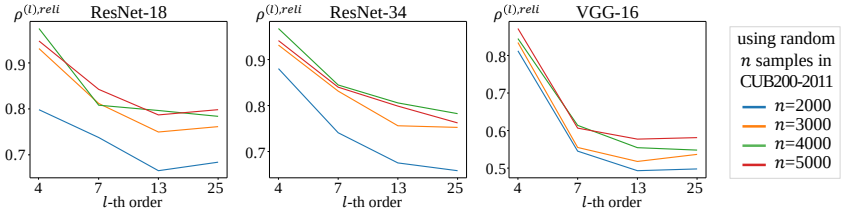


(b) Significance of feature components $\rho_c^{(l)}$ in DNNs learned on the Stanford Dogs dataset

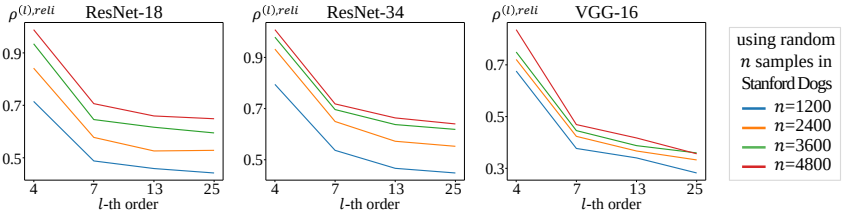
Figure 15: Significance of feature components in DNNs learned on the CUB200-2011 dataset and the Stanford Dogs dataset.

Figure 16 shows the result of $\rho^{(l),\text{reli}}$ of ResNet-18/34 and VGG-16 learned on the CUB200-2011 dataset and the Stanford Dogs dataset. We also used two exemplary DNNs A and B to help us extract the reliable feature components from the target feature. DNNs A and B were implemented as two

DNNs of ResNet-34 trained on the entire training set in the CUB200-2011 dataset. Similarly, the two exemplary DNNs for the Stanford Dogs dataset were also implemented as two DNNs of ResNet-34. We found that DNNs learned from many training samples always exhibited higher reliability than DNNs learned form a few training samples, which meant that the increase of the number of training samples would help DNNs learn more reliable features.



(a) Reliability of feature components in DNNs learned on the CUB200-2011 dataset



(b) Reliability of feature components in DNNs learned on the Stanford Dogs dataset

Figure 16: Reliability of feature components in DNNs learned on the CUB200-2011 dataset and the Stanford Dogs dataset.

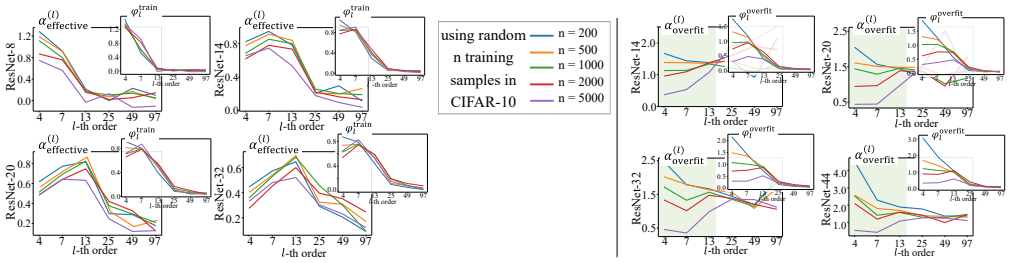


Figure 17: (left) Effectiveness of feature components $\alpha^{(l)}_{\text{effective}}$. The top-right sub-figure shows the Shapley value φ^{train}_l; (right) Significance of feature components being over-fitted $\alpha^{(l)}_{\text{overfit}}$. The top-right sub-figure shows the Shapley value $\varphi^{\text{overfit}}_l$.

H Analysis of the effectiveness and the significance of over-fitting of feature components.

This section provides more discussions about the result in Figure 7. Figure 17 compares the effectiveness $\alpha^{(l)}_{\text{effective}}$ and the significance of over-fitting $\alpha^{(l)}_{\text{overfit}}$ of feature components in different DNNs, which have been reported in Figure 7. Besides, Figure 17 also includes sub-figures which show the Shapley values φ^{train}_l and $\varphi^{\text{overfit}}_l$. φ^{train}_l measures the change of the training loss caused by feature components $\{c^{(l)}(x)|x \in X_{\text{train}}\}$. $\varphi^{\text{overfit}}_l$ measures the numerical contribution of each feature component $c^{(l)}(x)$ to the significance of over-fitting.

We obtained the following two conclusions:

(1) When the complexity order of feature components is about half of the depth of the DNN, these feature components exhibited the highest effectiveness. For ResNet-8, the feature component with the highest effectiveness was of the 4-th order. For ResNet-14, the 7-order feature component was

the most effective. In other words, these feature components contributed the most to the decrease of the loss in DNNs, which was verified by large values of φ_l^{train} when $l \approx \frac{1}{2}depth$ in sub-figures.

(2) Low-complexity feature components learned from a small number of samples were usually more over-fitted than low-complexity feature components learned from many samples. We cannot summarize clear conclusions from the significance of over-fitting for high-complexity feature components. This might be due to the low effectiveness of high-complexity feature components (noise-like features).

I IMPROVEMENT OF THE CLASSIFICATION ACCURACY BASED ON $\Phi^{(l)}(x)$

This section discusses the improvement of performance in Figure 9. When we used the feature components with $l = 7$, the classification accuracy of ResNet-14 learned on the CIFAR-10 dataset was improved by over 5%. Besides, for different complexity orders, the accuracy improvement was different. When the complexity order increased, the accuracy improvement first increased, and then decreased. This was because the very few feature components of the lowest complexity did not contain enough information for the classification. However, high-complexity feature components usually had a high significance of over-fitting, which hurt the performance of DNNs.

J DETAILS OF NETWORK COMPRESSION AND KNOWLEDGE DISTILLATION

This section introduces more details about the network compression and knowledge distillation in Exp. 3.

Network compression based on (Han et al., 2015): We learned a compressed DNN by pruning and then quantization. In the pruning phase, we pruned the DNN with the sensitivity rate 1.0 for all convolutional and fully connected layers. We iteratively pruned the DNN and retrained the weights. The number of this iteration was 300, and the weights were retrained for 20 epochs in our experiments. The weights in the pruned DNN was retrained for 100 epochs. For example, as a result, ResNet-32 trained on CIFAR10-1000 had an overall pruning rate of 5.88$\times$ without affecting the accuracy significantly. To compress further, we quantized weights in the DNN. For weights in convolutional layers, we quantized them to 8 bits, while for weights in fully connected layers, we quantized them to 5 bits. In this way, we obtained the compressed DNN.

Knowledge distillation based on (Hinton et al., 2015): To obtain the distilled DNN, we used a shallower DNN to mimic the intermediate-layer feature of the original DNN. For simplicity, we let the distilled DNN have the same architecture with the disentangler net with seven ReLU layers. The distilled DNN usually addressed the problem of over-fitting. For example, the distilled DNN based on ResNet-32 on CIFAR10-1000 had a 3.48% decrease in testing accuracy, without affecting the training accuracy significantly.

K STRONG RELATIONSHIP BETWEEN FEATURE COMPLEXITY AND PERFORMANCE OF DNNS

To investigate the relationship between the feature complexity and the performance of DNNs, we learned a regression model, which used the distribution of feature components of different complexity orders to predict the performance of DNNs. For each DNN, we used disentangler nets with $l = 4, 7, 13, 25$ to disentangle out $\Phi^{(l),\text{reli}}(x)$ and $\Phi^{(l),\text{unreli}}(x)$. Then, we calculated $Var[\Phi^{(l),\text{reli}}(x) - \Phi^{(l-1),\text{reli}}(x)]/Var[f(x)]$ and $Var[\Phi^{(l),\text{unreli}}(x) - \Phi^{(l-1),\text{unreli}}(x)]/Var[f(x)]$ for $l = 4, 7, 13, 25$, thereby obtaining an 8-dimensional feature to represent the distribution of different feature components. In this way, we learned a linear regressor to use the 8-dimensional feature to predict the testing loss or the classification accuracy, as follows.

$$\text{result} = \sum_{i=1}^{4} \alpha_i \times \frac{Var[\Phi^{(l_i),\text{reli}}(x) - \Phi^{(l_{(i-1)}),\text{reli}}(x)]}{Var[f(x)]} + \sum_{i=1}^{4} \beta_i \times \frac{Var[\Phi^{(l_i),\text{unreli}}(x) - \Phi^{(l_{(i-1)}),\text{unreli}}(x)]}{Var[f(x)]} + b$$

where $l_i \in \{4, 7, 13, 25\}$.

For the CIFAR-10 dataset, we applied cross validation: we randomly selected 20 DNNs from 25 pre-trained ResNet-8/14/20/32/44 models on different training sets in Exp. 2 to learn the regressor and

used the other 5 DNNs for testing.[4] These 25 DNNs were learned using 200-5000 samples, which were randomly sampled from the CIFAR-10 dataset to boost the model diversity. We repeated such experiments for 1000 times for cross validation.

Table 2 reports the mean absolute value of prediction error for the classification accuracy and the task loss over 1000 repeated experiments. The prediction error was much less than the value gap of the testing accuracy and the value gap of the task loss, which indicated the strong relationship between the distribution of feature complexity and the performance of DNNs.

Table 2: The mean absolute value of prediction errors.

	Accuracy		Task loss	
	Prediction error	Range of value	Prediction error	Range of value
CIFAR-10	**2.73%**	28.73%-72.83%	**0.49**	1.59-6.42
CUB200-2011	**5.66%**	28.18%-56.18%	**0.47**	2.94-5.76
Stanford Dogs	**3.26%**	9.37%-37.95%	**0.34**	4.34-7.97

Figure 18 further visualizes the plane of the linear regressor learned on the CIFAR-10 dataset. The visualization was conducted by using PCA (Wold et al., 1987) to reduce the 8-dimensional feature into a 2-dimensional space, *i.e.* (x, y) in Figure 18. There was a close relationship between the distribution of feature complexity and the performance of a DNN.

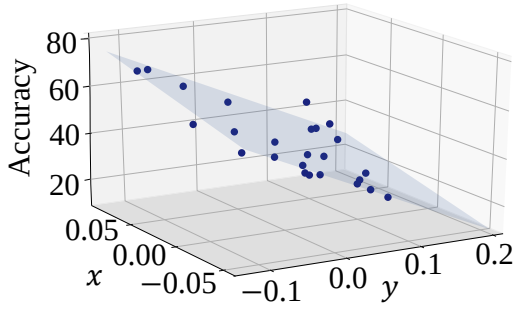


Figure 18: Relationship between the feature complexity and the accuracy.

Testing on the compressed DNNs. We also conducted experiments to show the strong relationship between the feature complexity and the network performance in terms of the compressed DNNs. We predicted the accuracy of the compressed DNNs with the learned regressor and Table 3 shows the result. The prediction error was relatively small, and it further validated the close relationship between the feature complexity and the performance of DNNs.

Table 3: Prediction result of the accuracy of the compressed DNNs.

Model	ResNet-14	ResNet-20	Resnet-32	ResNet-44
Dataset	CIFAR10-2000			
Test acc. before compression	52.41	56.15	56.80	56.00
Test acc. after compression	53.88	57.94	60.81	58.04
Predicted acc. after compression	50.97	54.32	60.46	57.41
Error	-2.91	-3.62	-0.35	-0.63
Model	ResNet-32			
Dataset	CIFAR10-500	CIFAR10-1000	CIFAR10-5000	
Test acc. before compression	37.94	46.71	72.83	
Test acc. after compression	38.35	49.86	73.62	
Predicted acc. after compression	45.23	53.35	72.85	
Error +6.88	+3.49	-0.77		

[4]For the CUB200-2011 dataset and the Stanford Dogs dataset, we randomly selected 11 models from 12 pre-trained ResNet-18/34 and VGG-16 models to learn the regressor. One model was used for testing.