Towards Theoretical Analysis of Transformation Complexity of ReLU DNNs

Jie Ren [*1] Mingjie Li [*1] Meng Zhou [2] Shih-Han Chan [3] Quanshi Zhang [4]

Abstract

This paper aims to theoretically analyze the complexity of feature transformations encoded in piecewise linear DNNs with ReLU layers. We propose metrics to measure three types of complexities of transformations based on the information theory. We further discover and prove the strong correlation between the complexity and the disentanglement of transformations. Based on the proposed metrics, we analyze two typical phenomena of the change of the transformation complexity during the training process, and explore the ceiling of a DNN's complexity. The proposed metrics can also be used as a loss to learn a DNN with the minimum complexity, which also controls the over-fitting level of the DNN and influences adversarial robustness, adversarial transferability, and knowledge consistency. Comprehensive comparative studies have provided new perspectives to understand the DNN. The code is released at *https://github.com/sjtu-XAI-lab/transformation-complexity*.

1. Introduction

Understanding the black-box of deep neural networks (DNNs) has attracted increasing attention in recent years. Previous studies usually interpret DNNs by either explaining DNNs visually/semantically (Lundberg & Lee, 2017; Ribeiro et al., 2016), or analyzing the feature representation capacity of a DNN (Higgins et al., 2017; Achille & Soatto, 2018a;b; Fort et al., 2019). To this end, in this paper, we aim to explain the representation capacity of a DNN by analyzing its complexity of feature representations.

[*]Equal contribution [1]Shanghai Jiao Tong University. [2]Carnegie Mellon University. [3]University of California San Diego. [4]Quanshi Zhang is the corresponding author. He is with the Department of Computer Science and Engineering, the John Hopcroft Center, and the MoE Key Lab of Artificial Intelligence, AI Institute, at the Shanghai Jiao Tong University, China. Correspondence to: Quanshi Zhang <zqs1022@sjtu.edu.cn>.

Proceedings of the 39th International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022. Copyright 2022 by the author(s).

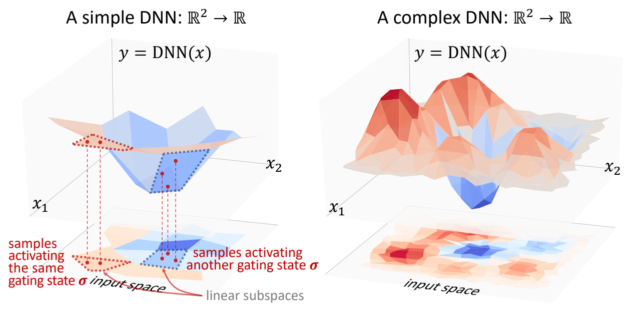

A simple DNN: $\mathbb{R}^2 \to \mathbb{R}$
$y = \mathrm{DNN}(x)$
x_1
x_2
samples activating the same gating state σ
input space
samples activating another gating state σ
linear subspaces
A complex DNN: $\mathbb{R}^2 \to \mathbb{R}$
$y = \mathrm{DNN}(x)$
x_1
x_2
input space

Figure 1. The change of gating states in an ReLU DNN divides the input space into lots of linear subspaces. In this paper, we aim to quantify the complexity of linear subspaces encoded in ReLU DNNs.

In fact, there have been many studies (Arora et al., 2016; Zhang et al., 2016; Raghu et al., 2017; Manurangsi & Reichman, 2018) on the representation complexity of a DNN. However, unlike traditional studies on the theoretic maximum complexity of a DNN, we investigate this problem from a new perspective, *i.e.*, the complexity of piecewise linear representations of a ReLU network. In fact, **most DNNs with ReLU activation functions are piecewise linear models. As Figure 1 shows, for such piecewise linear models, the number of piecewise linear subspaces in the model is the distinctive perspective to analyze the representation complexity of the model.** A complex DNN usually uses countless linear subspaces for regression/classification. Such a complexity reveals new insights into typical problems, such as the feature disentanglement, over-fitting, adversarial robustness, adversarial transferability, and knowledge consistency.

Therefore, in this paper, we define three new metrics in information theory to quantify the diversity of gating states in gating layers (*e.g.*, ReLU, max-Pooling, and Dropout layers). Such gating states directly determine the complexity of linear subspaces that are encoded by a piecewise linear ReLU network. For example, as Figure 1 shows, the change of gating states in a DNN divides the entire input space into lots of linear subspaces. Compared with intermediate-layers neural activations, gating states can better reflect the diversity of transformations in a piecewise linear DNN.

Thus, the first metric is defined as the entropy of the gating states of nonlinear operations. Specifically, let us consider the task (X, Y), where X denotes all input samples and Y denotes corresponding labels. Given a DNN, the complexity of transformations represents the diversity of transformations that map each input x to the corresponding $y \in Y$ over all inputs $x \in X$, *i.e.,* the number of linear subspaces in piecewise linear transformations. From this perspective, the simplest model is a linear transformation $y = f(w^{\mathrm{T}}x + b) = w^{\mathrm{T}}x + b$, where the nonlinear operation $f(\cdot)$ is by-passed as an all-passing gating layer. In this case, the entropy of its gating states is zero. In comparison, as Figure 1 shows, a DNN usually generates various gating states for different inputs, thereby dividing all training samples into different subspaces. The complexity of such a piecewise linear representation is quantified as the entropy of gating states of all gating layers. Let the binary vector $\sigma_l = [\sigma_{l,1}, \sigma_{l,2}, \ldots, \sigma_{l,D}] \in \{0,1\}^D$ denote gating states of the gating layer. Let Σ represent the random variable of all gating states in all layers of the DNN. In this way, the entropy $H(\Sigma)$ among all inputs measures the complexity of the overall transformation complexity in the DNN.

Based on the entropy $H(\Sigma)$, we further propose $I(X;\Sigma)$ and $I(X;\Sigma;Y)$ as two additional metrics to further disentangle specific components of the overall transformation complexity $H(\Sigma)$ in a fine-grained manner. The mutual information $I(X;\Sigma)$ measures the complexity of transformations that are caused by the input. The mutual information $I(X;\Sigma;Y)$ represents the complexity of transformations that are caused by the input and are directly used for inference. For example, in the task of object classification, not all transformations in the DNN are category-specific. $I(X;\Sigma;Y)$ reflects category-specific components of transformations. Notice that $I(X;\Sigma) = H(\Sigma)$ in most cases. However, for other cases when the DNN uses additional transformations (randomness) beyond transformations caused by the input X, we have $I(X;\Sigma) \neq H(\Sigma)$. For example, sampling operations in the VAE (Kingma & Welling, 2013) and the dropout operation bring additional transformations.

Analysis and explanations. In this paper, we prove various properties of the complexity metrics. Let us focus on the complexity of transforming the feature in an intermediate layer to the output of the DNN. The transformation complexity decreases through the layerwise propagation. In other words, deep features usually require simpler transformations to conduct inference than shallow features. Then, we use the proposed complexity metrics to analyze the knowledge representation of DNNs. We can summarize the change of the complexity during the training process into two types. In traditional stacked DNNs without skip-connections, the complexity usually decreases first, and increases later. This indicates that DNNs may drop noisy features in early stages, and learn useful information later. Whereas, in residual networks, the transformation complexity does not decrease during the learning process.

In this study, we conduct the following theoretic explorations based on the complexity metrics.

(1) **Disentanglement: we prove the strong correlation between the complexity and the disentanglement of transformations.** Let us consider DNNs with similar activation rates in a certain layer. If the complexity of the transformation encoded in a specific layer is larger, then gating states of different feature dimensions tend to be more independent with each other.

(2) **Minimum complexity and the gap between the training loss and the testing loss: we use the complexity as a loss to learn a DNN with the minimum complexity.** A DNN usually learns over-complex transformations *w.r.t.* the task. Thus, we propose a complexity loss, which penalizes unnecessary transformations to learn a DNN with the minimum complexity. Given the DNN learned using the complexity loss, we find that the gap between the training loss and the testing loss decreases, when we reduce the complexity of transformations. Furthermore, we find that the complexity loss also influences adversarial robustness, adversarial transferability, and knowledge consisitency.

(3) **Maximum complexity: we explore the ceiling of a DNN's complexity.** *First, the complexity of a DNN does not monotonously increase with the network depth.* In contrast, the traditional stacked DNN with a deep architecture may encode simpler transformations than shallow DNNs in some cases. Whereas, the transformation complexity of residual networks is saturated when we use more gating layers. *Second, the complexity of transformations does not increase monotonously along with the increase of the complexity of tasks.* In contrast, if the task complexity exceeds a certain limit, the complexity of transformations encoded in the DNN will decrease along with the increase of the task complexity.

Theorectial contributions of this study can be summarized as follows. (1) We define three metrics to evaluate the complexity of transformations in piecewise linear DNNs, which have a great theoretical extensibility. (2) We prove the strong correlation between the complexity and the disentanglement of transformations. (3) We further use the transformation complexity as a loss to learn a minimum-complexity DNN, which also reduces the gap between the training loss and the testing loss. (4) Comparative studies reveal the ceiling of a DNN's complexity.

2. Related Work

We limit our discussion within the literature of understanding representations of DNNs. In general, previous studies can be roughly summarized into the following two types.

- The first type is the semantic explanations for DNNs.

Some studies directly visualized knowledge representations encoded in the DNN (Zeiler & Fergus, 2014; Mahendran & Vedaldi, 2015; Yosinski et al., 2015; Dosovitskiy & Brox, 2016; Simonyan et al., 2017). Other methods estimated the pixel-wise attribution to the network output (Zhou et al., 2015; Selvaraju et al., 2017; Fong & Vedaldi, 2017; Kindermans et al., 2017; Zhou et al., 2016). The LIME (Ribeiro et al., 2016) and SHAP (Lundberg & Lee, 2017) extracted important input units that directly contributed to the output. Some visual explanations reveal certain insightful properties of DNNs. Fong & Vedaldi (2017) analyzed how multiple filters jointly represented a certain semantic concept. In contrast, in this paper, we propose to investigate the representation capacity from the perspective of transformation complexity encoded in DNNs.

• The second type is to analyze the feature representation capacity of DNNs. The stiffness (Fort et al., 2019) was proposed to diagnose the generalization capacity of DNNs. Xu (2018) applied the Fourier analysis to explain the generalization capacity of DNNs. The CLEVER score (Weng et al., 2018) was used to estimate the robustness of DNNs. Wolchover (2017); Shwartz-Ziv & Tishby (2017) proposed the information bottleneck theory and used mutual information to quantify the information encoded in DNNs. Xu & Raginsky (2017); Achille & Soatto (2018b); Goldfeld et al. (2019) further extended the information bottleneck theory to constrain the feature representation to learn more disentangled features. Chen et al. (2018) selected instance-wise features based on mutual information to interpret DNNs. Kornblith et al. (2019) used the canonical correlation analysis to compare features from the perspective of similarity.

Unlike previous studies, we focus on the complexity of transformations encoded in DNNs. Previous methods on the complexity of DNNs can be summarized as follows:

• Computational complexity and difficulty of learning a DNN: Blum & Rivest (1989) proved that learning a one-layer network with a sign activation function was NP-hard. Livni et al. (2014) further discussed the computational complexity when the DNN used other activation functions. Boob et al. (2018); Manurangsi & Reichman (2018) proved learning a two-layer ReLU network was also NP-hard. Arora et al. (2016) showed that it was possible to learn a ReLU network with one hidden layer in polynomial time, when the dimension of input was constant.

• Architectural complexity and representation complexity: Raghu et al. (2017) proved that the maximal complexity of features grew exponentially along with the increase of the DNN depth. Pascanu et al. (2013); Zhang et al. (2016) proposed three metrics to measure the architectural complexity of recurrent neural networks. To estimate the maximal representation capacity, (Liang et al., 2017; Cortes et al., 2017) applied the Rademacher complexity. (Kalimeris et al., 2019) analyzed the complexity of features in a DNN by comparing them with features learned by a linear classifier.

Unlike analyzing the complexity of the DNN based on its architecture, we aim to measure the complexity of feature transformations learned by the DNN. We define three types of transformation complexity, which provide new perspectives to understand the DNN.

3. Transformation Complexity

Strong connection between gating states and the transformation complexity. Given the input $x \in X$ and the target label $y \in Y$, the DNN is learned to map x to y. Layerwise transformations of mapping x to y can be roughly represented by

$$y = g(z),\ z = W_{L+1} \dots \boldsymbol{\sigma}_2(W_2\boldsymbol{\sigma}_1(W_1x+b_1)+b_2)\cdots+b_{L+1} \tag{1}$$

where g denotes the optional layer on the top, *e.g.,* the softmax layer. z denotes the output feature before the top layer. **The network module** $z = W_{L+1} \dots \boldsymbol{\sigma}_2(W_2\boldsymbol{\sigma}_1(W_1x + b_1) + b_2)\cdots + b_{L+1}$ **is a piecewise linear model.**

Specifically, W_l and b_l denote the weight and the bias of the l-th linear layer. Let $\sigma_l = [\sigma_{l,1}, \sigma_{l,2}, \dots, \sigma_{l,D}] \in \{0,1\}^D$ denote gating states of the l-th gating layer. $\boldsymbol{\sigma}_l = diag(\sigma_{l,1}, \sigma_{l,2}, \dots, \sigma_{l,D})$ is a diagonal matrix with $(\sigma_{l,1}, \sigma_{l,2}, \dots, \sigma_{l,D})$ as its main diagonal. Gating layers include the ReLU, max-Pooling, and Dropout layer. Take the ReLU layer as an example[1]. If the d-th dimension of the input feature is larger than 0, then we have $\sigma_{l,d} = 1$; otherwise, $\sigma_{l,d} = 0$. Let $\Sigma_l = \{\boldsymbol{\sigma}_l\}$ denote the random variable of gating states of the l-th gating layer. Given a certain input x, $\boldsymbol{\sigma} = [\boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2, \dots, \boldsymbol{\sigma}_L]$ represents concatenated and vectorized gating states of all gating layers. Accordingly, $\Sigma = [\Sigma_1, \Sigma_2, \dots, \Sigma_L]$ denotes the set of gating states of all L gating layers over all samples[2].

Therefore, the layerwise transformation of mapping x to y can be represented as $y = g(z)$ and $z = \mathbf{W}x + \mathbf{b}$, where $\mathbf{W} = W_{L+1}\boldsymbol{\sigma}_L W_L \cdots \boldsymbol{\sigma}_2 W_2 \boldsymbol{\sigma}_1 W_1$ is the equivalent weight matrix, and $\mathbf{b}$ is the equivalent bias term. The piecewise linear module $z = \mathbf{W}x + \mathbf{b}$ generates different gating states $\boldsymbol{\sigma}_1, \cdots, \boldsymbol{\sigma}_L$ for different inputs x, which lead to different values of $\mathbf{W}$ and $\mathbf{b}$. Therefore, we can roughly use the diversity of gating states to approximate the diversity of transformations. Thus, the entropy of gating states $H(\Sigma)$ can represent the transformation complexity.

Note that not all small perturbations of the input x change the gating states, so gating states can be considered more directly related to the transformation complexity than the

[1]Please see Appendix A for more details about other types of gating layers.

[2]Please see Appendix G.1 for details about samples that are considered in implementations.

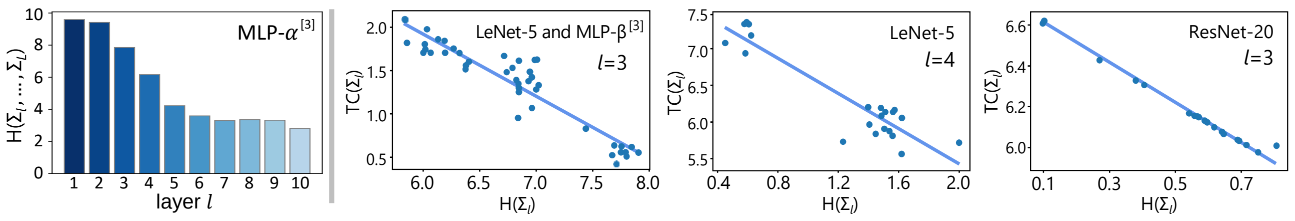


Figure 2. (left) The complexity of transforming feature T_l to the output, *i.e.,* $H(\Sigma_l, \ldots, \Sigma_L)$. The complexity decreases as we use features of higher layers. (right) The negative correlation between the transformation complexity $H(\Sigma_l)$ and the entanglement $TC(\Sigma_l)$ of DNNs with similar activation rates.

input. In this way, we focus on gating layers in the DNN and define the following metrics to measure three types of complexities of transformations from x to y.

Definition of the transformation complexity. In this section, we define three types of complexities of transformations in DNNs based on the information theory.

• $H(\Sigma)$: the entropy of gating states among all inputs. $H(\Sigma)$ measures the complexity of transformations that are encoded in gating layers. A larger $H(\Sigma)$ indicates the DNN learns more complex transformations. The complexity $H(\Sigma)$ can be decomposed as $H(\Sigma) = H(\Sigma_1) + H(\Sigma_2|\Sigma_1) + \cdots + H(\Sigma_L|\Sigma_1, \ldots, \Sigma_{L-1})$.

• $I(X;\Sigma)$: the complexity of transformations that are caused by the input. If the DNN does not use the random sampling operation or the dropout operation to introduce additional uncertainty that is not caused by the input in gating states Σ, then Σ is fully determined by X, *i.e.,* $H(\Sigma|X) = 0$ and $I(X;\Sigma) = H(\Sigma) - H(\Sigma|X) = H(\Sigma)$.

• $I(X;\Sigma;Y)$: the complexity of transformations that are caused by inputs and used for inference, which is defined as $I(X;\Sigma;Y) = I(X;Y) - I(X;Y|\Sigma)$. $I(X;Y|\Sigma) = H(X|\Sigma) - H(X|\Sigma,Y)$ measures the mutual information between X and Y that is irrelevant to gating layers.

Our definition of the complexity satisfies the following simple properties in DNNs, which ensures the trustworthiness of the complexity metric.

Property 1. (Proof in Appendix B.1) If the DNN does not introduce additional information that is not contained by the input X (e.g., *there are no operations of randomly sampling or dropout throughout the DNN), then we have* $I(X;\Sigma;Y) \geq 0$.

Property 2. If the DNN does not introduce additional complexity that is not caused by the input, then the complexity increases along with the number of gating layers.

Property 3. If the DNN does not introduce additional complexity that is not caused by the input, then the complexity decreases when we use features of shallow layers for inference. This property shows that the transformation complexity decreases through the layerwise propagation.

Please see Appendix B.2 and B.3 for formulations and proofs of Properties 2 and 3.

Verification of the decrease of the complexity through layerwise propagation. Figure 2(left) shows the change of the complexity of transforming the l-th layer feature T_l to the output, *i.e.,* the entropy of gating states after the l-th layer $H(\Sigma' = [\Sigma_l, \ldots, \Sigma_{10}])$ encoded in the DNN during the training process on the MNIST dataset, which verified the decrease of the complexity through layerwise propagation. Experimental settings in Figure 2(left) are introduced in Section 4.

The quantification of the transformation complexity. There are three classic and widely-used non-parametric methods to estimate the entropy and mutual information of features, including the binning method (Shwartz-Ziv & Tishby, 2017), the ensemble dependency graph estimator (EDGE) (Noshad et al., 2019) and the kernel densitiy estimation (KDE) (Kolchinsky & Tracey, 2017). Both the binning and the EDGE methods reduce the computational cost by discretizing the continuous features. However, the gating state Σ is a discrete random variable, which cannot benefit from such methods. Thus, we apply the KDE method to estimate the complexity $H(\Sigma)$, $I(X;\Sigma)$ and $I(X;\Sigma;Y)$. Please see Appendix E for details about the KDE method of estimating $H(\Sigma)$, $I(X;\Sigma)$ and $I(X;\Sigma;Y)$. *In Appendix G.2, we have also verified the high accuracy and the stability of using KDE to estimate $H(\Sigma)$, $I(X;\Sigma)$ and $I(X;\Sigma;Y)$.*

The difference between the transformation complexity and the information bottleneck. Note that the transformation complexity $I(X;\Sigma)$ has essential difference from the $I(X;Z)$ term in the information bottleneck theory (Tishby et al., 2000; Wolchover, 2017; Shwartz-Ziv & Tishby, 2017), where Z denotes the feature of an intermediate layer in DNNs. The information bottleneck reflects the trade-off between $I(X;Z)$ and $I(Z;Y)$, which leads to the approximate minimal sufficient statistics. In the forward propagation, the feature Z contains all information encoded in the DNN, thereby forming a Markov process $X \to Z \to Y$. Thus, given the feature Z, X and Y are conditional independent, *i.e.,* $I(X;Y|Z) = 0$.

However, the gating state Σ does not contain all information of the feature Z. Let us take the ReLU layer for an instance. In ReLU layers, the gating state Σ only represents whether the elements in feature Z are positive. The information encoded in Σ cannot be directly used to infer Y, which makes the transformation complexity $I(X;\Sigma)$ essentially different from $I(X;Z)$ in mathematics.

Advantages of investigating gating states over studying neural activations. $I(X;\Sigma)$ based on gating states Σ is substantially more related to the transformation complexity than $I(X;Z)$ based on neural activations Z. Theoretically speaking, the entropy/mutual information of neural activations Z in an intermediate layer is affected by two aspects, *i.e.,* the diversity of transformations and the continuous changes of input samples. We have proven that gating states more directly reflect the transformation complexity in DNNs than neural activations z.

To be precise, any intermediate-layer neural activation z can be represented as a linear transformation on the input x, *i.e.,* $z = \mathbf{W}x + \mathbf{b}$, where $\mathbf{W}$ and $\mathbf{b}$ are determined by gating states in nonlinear layers directly. Therefore, gating states are the most direct factor that determines and reflects the diversity of linear subspaces in piecewise linear transformations, and are directly related to the representation power of a DNN. Whereas, the complexity of neural activations z are also affected by noisy signals in x, instead of purely being affected by the division of linear subspaces.

Therefore, in this study, we exclusively focus on the complexity of transformations $(\mathbf{W}, \mathbf{b})$, which can be considered as a more purified metric for a DNN's representation than the feature complexity.

4. Analysis of DNNs Based on the Transformation Complexity

4.1. Strong correlation between the complexity and the disentanglement

The disentanglement is a property of a DNN that measures the independence of different feature dimensions in the DNN. Stronger disentanglement sometimes leads to more interpretable features (Burgess et al., 2017), though the disentanglement is not necessarily equivalent to the discrimination power of features (which will be analyzed later). Although intuitively, the disentanglement of gating states seems not related to the complexity, in this section, we prove a clear correlation between these two terms.

For gating states of the l-th gating layer Σ_l, the entanglement of transformations $TC(\Sigma_l)$ measures the dependence between gating states $\sigma_{l,d}$ in different dimensions (Achille & Soatto, 2018a; Ver Steeg & Galstyan, 2015). Specifically, $TC(\Sigma_l) = KL(p(\sigma_l)\|\prod_d p(\sigma_{l,d}))$, where $p(\sigma_{l,d})$ denotes the marginal distribution of the state of the d-th gate in σ_l. Let us assume that $p(\sigma_{l,d}) \sim \text{Bernoulli}(a_{l,d})$, where $a_{l,d}$ is the activation rate of the d-th dimension in the l-th gating layer over all samples. In particular, $TC(\Sigma_l)$ is zero if and only if all dimensions of σ_l are independent with each other. In this case, we say all dimensions in Σ_l are disentangled.

For DNNs with similar activation rates, we prove the negative correlation between the complexity $H(\Sigma_l)$ and the entanglement $TC(\Sigma_l)$ (proof in Appendix B.4).

$$H(\Sigma_l) + TC(\Sigma_l) = C_l, \quad C_l = \sum\nolimits_d H_{l,d}, \tag{2}$$

where $H_{l,d} = -a_{l,d}\log a_{l,d} - (1-a_{l,d})\log(1-a_{l,d})$. We can roughly consider C_l as a constant if the average activation rate over all dimensions $\{a_{l,d}|1 \le d \le D\}$ in the l-th layer converges to a certain number. Furthermore, we extend conclusions in Eq. (2) and prove that the negative correlation between the complexity and the entanglement of transformations in the DNN still holds true for $I(X;\Sigma_l)$ and $I(X;\Sigma_l;Y)$.

$$\begin{gathered} I(X;\Sigma_l) + TC(\Sigma_l) = C_l - H(\Sigma_l|X) \\ I(X;\Sigma_l;Y) + \underbrace{(TC(\Sigma_l) - TC(\Sigma_l|Y))}_{\text{multi-variate mutual information used to infer } Y} \\ = C_l - C_{l|Y} - (H(\Sigma_l|X) - H(\Sigma_l|X,Y)) \end{gathered} \tag{3}$$

Whether the entanglement of features is good for classification or not? As is shown above, given a fixed activation rate in a DNN, the decrease of complexity leads to the increase of entanglement. The entangled representations (or gating states) are supposed to occur in the following two cases. One case is good for the classification, but the other is not.

(1) The first case is when the DNN fails to extract meaningful patterns from the input. Many pixels in an input image are actually entangled with each other to represent a certain concept. The successful extraction of features for meaningful concepts is to summarize all information of such pixel-wise entanglement into a certain intermediate-layer filter, which makes the intermediate-layer feature more disentangled. The lack of the transformation complexity of a DNN usually will lead to high entanglement and hurt the classification performance.

(2) The second case is when the DNN has successfully selected a few discriminative and reliable features from input images in high convolutional layers for classification. Sometimes, the number of reliable features is much less than the filter number. In other words, there exists a significant redundancy of feature representations, *i.e.,* multiple filters may represent similar features in high convolutional layers, which leads to high entanglement. Such an entanglement (redundancy) of reliable features is good for classification. More intuitive examples for these two cases are shown in Appendix D.

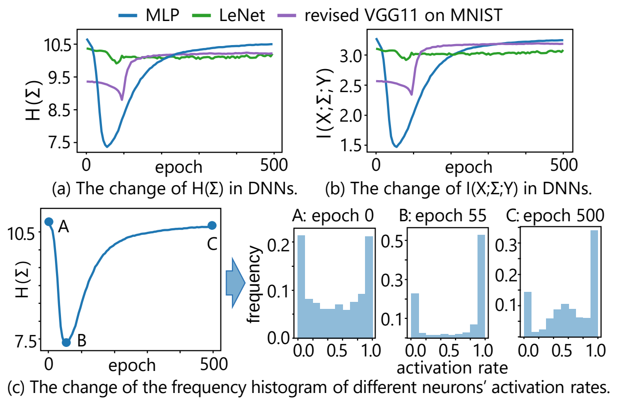


Figure 3. The change of the transformation complexity and the activation rates in traditional stacked DNNs. In most cases, both $H(\Sigma)$ and $I(X;\Sigma;Y)$ decreased first, and increased later.

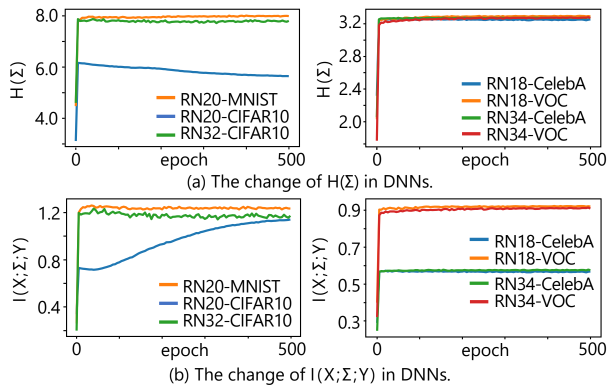


Figure 4. The change of the transformation complexity in residual networks. Both $H(\Sigma)$ and $I(X;\Sigma;Y)$ increased monotonously during the training process in most residual networks.

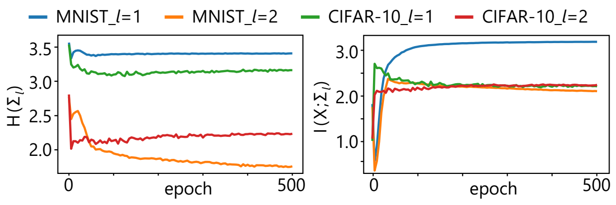


Figure 5. The change of $H(\Sigma_l)$ and $I(X;\Sigma_l)$ in VAEs learned on the MNIST dataset and the CIFAR-10 dataset. The difference between $H(\Sigma_l)$ and $I(X;\Sigma_l)$ gradually decreased during the training process.

Verification of the strong correlation between the complexity and the disentanglement (Eq. (2)). We learned 21 LeNet-5 models and 21 MLP-β[3] models with different initialized parameters on the MNIST dataset. These models shared similar activation rates on each dimension in the l-th gating layer (we used $l = 3, 4$). Thus, we quantified $H(\Sigma_l)$ and $TC(\Sigma_l)$ (Please see Appendix E for the quantification of $TC(\Sigma_l)$) over the 42 models. We also conducted such an experiment on 21 ResNet-20 (RN-20) models with $l = 3$. Figure 2(right) shows the negative correlation between $H(\Sigma_l)$ and $TC(\Sigma_l)$, which was verified using different layers of DNNs with different architectures.

4.2. Comparative studies to diagnose the representation capacity of DNNs

The change of the transformation complexity during the learning of DNNs. Figure 3 and Figure 4 shows the change of three types of transformation complexities encoded in DNNs during the training process. Note that $H(\Sigma) = I(X;\Sigma)$. We found three phenomena in the change of complexity:

Phenomenon 1. For most traditional stacked DNNs (like MLPs[3] and LeNets), both $H(\Sigma)$ and $I(X;\Sigma;Y)$ decreased first, and increased later. Figure 3(c) shows the frequency histogram of different neurons' activation rates of gating states in Epoch 0, 55, and 500. In Epoch 0, gating states were usually randomly activated, which led to a large value of $H(\Sigma)$. The learning process gradually removed noisy activations, which reduced $H(\Sigma)$ and achieved the minimum complexity in Epoch 55. Then, the DNN mainly learned complex transformations to boost the performance, which made $H(\Sigma)$ begin to increase. This indicated that DNNs dropped noisy features in early stages of the training process, then learned useful information for the inference. In fact, the drop of noisy features in very early iterations has also been supported by observations in other studies (Liu et al., 2021). In very early iterations, initial weights irrelevant to the task were usually removed, which reduces the transformation complexity. Besides, Liu et al. (2021) also showed that the decrease of the model diversity in very early iterations exactly aligned with the first stage in the epoch-wise double descent.

[3]Please see the paragraph *Experimental settings*, Section 4 for network architectures and experimental settings.

Phenomenon 2. For residual DNNs with skip-connections and a few traditional DNNs (like the VGG-16 trained on the Pascal VOC dataset; see Appendix G.4), the complexity increased monotonously during the early stage of the training process, and saturated later. This indicated that noisy features had little effect on DNNs with skip-connections in early stages of the learning process, which implied the temporal double-descent phenomenon (Nakkiran et al., 2019; Heckel & Yilmaz, 2020).

Phenomenon 3. In particular, let us focus on DNNs, which introduce additional uncertainty that is not caused by the input. As introduced in Section 1, a typical case is the VAE (Kingma & Welling, 2013). VAEs use randomly sampling of the latent code, and make $H(\Sigma) \neq I(X,\Sigma)$. Thus, in this experiment, we studied the change of $H(\Sigma)$ and $I(X;\Sigma)$ in VAEs. We were given a VAE[3], in which both the encoder and the decoder had two FC layers. We added a classifier with two FC layers and two ReLU layers after the encoder.

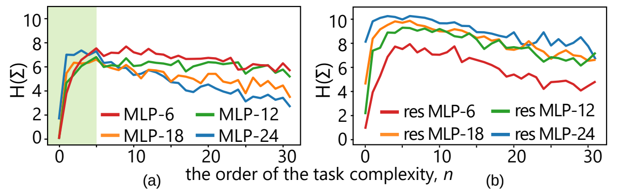


Figure 6. The change of target MLPs' transformation complexity along with the increase of task complexity n. We used MLPs with (a) traditional stacked architectures and (b) residual architectures.

The VAE was trained on the MNIST dataset and the CIFAR-10 dataset, respectively. Figure 5 shows the complexity of transformations encoded in each gating layer of the classifier. The difference between $H(\Sigma_l)$ and $I(X;\Sigma_l)$ gradually decreased during the training process. This indicated that the impact of inputs on the transformation complexity kept increasing. At the same time, the noisy features encoded in the DNN kept decreasing.

Maximum complexity: exploring the ceiling of a DNN's complexity. In order to examine whether a DNN always encoded more complex transformations when it dealt with more complex tasks, we first constructed a set of different tasks with various complexities. Specifically, we designed a set of knowledge-distillation tasks with different complexity levels. In other words, we defined different *task MLPs* with various complexities and distilled the knowledge of the *task MLP* to a *target MLP*. We explored whether a target MLP would learn more complex transformations, when it distilled knowledge from a more complex task MLP.

Each of the task MLPs was assigned with randomly initialized parameters without further training. A task MLP consisted of n ReLU layers and FC layers with the width of 1024. The task MLP took gray-scale CIFAR-10 images as input, and generated a 1024-dimensional vector as output. We learned the target MLP to reconstruct this output vector with an MSE loss.

Since the task MLP was randomly parameterized, the task complexity would be high, when the number of ReLU layers n in the task MLP was large. Therefore, we considered that the complexity of distilling knowledge of this task MLP into the target MLP was at the n-th level. We conducted multiple experiments to train different target MLPs. Each target MLP had a specific depth with 6, 12, 18 or 24 ReLU layers and FC layers, and each layer had 1024 neurons. We trained these target MLPs to regress task MLPs of different complexities, $n = 0, 1, \ldots, 31$.

Findings from stacked networks. Figure 6(a) compares the complexity of transformations encoded in target MLPs (with the traditional stacked architecture) for different tasks.
(1) For the task of low complexity, deep MLPs learned more complex transformations than shallow MLPs.
(2) However, as the complexity of the task increased, the complexity of transformations encoded in shallow MLPs was usually higher than that encoded in deep MLPs.
In other words, the transformation complexity did not monotonously increase along with the depth of the DNN. This phenomenon shows the ceiling of a DNN's complexity.

Findings from residual networks. Besides above target MLPs, we further designed new target MLPs with skip-connections, which were termed residual MLPs. We added a skip-connection to each FC layer in each of above target MLPs. The complexity of transformations encoded in residual MLPs was shown in Figure 6(b).
(1) Deep residual MLPs always encoded higher transformation complexity than shallow residual MLPs.
(2) We also found that when we gradually increased the task complexity to train target MLPs, the transformation complexity encoded in target MLPs increased along with the increase of the task complexity in the beginning.
(3) However, when the task complexity exceeded a certain limit, the transformation complexity saturated and started to decrease.
I.e., the transformation complexity did not keep increasing when the DNN was forced to handle more and more complex tasks. This phenomenon indicated the ceiling of a DNN's complexity from another perspective.

4.3. Learning a DNN with the minimum complexity

Minimum complexity. A DNN may use over-complex transformations for prediction, *i.e.,* the complexity $I(X;\Sigma;Y)$ does not always represent the real complexity of the task. In this section, we develop a method to avoid learning an over-complex DNN. The basic idea is to use the following loss to quantify and penalize the complexity of transformations during the training process.

$$\begin{aligned}\mathcal{L} &= \mathcal{L}_{\text{task}} + \lambda \mathcal{L}_{\text{complexity}},\\ \mathcal{L}_{\text{complexity}} &= \sum\nolimits_{l=1}^{L} H(\Sigma_l) = \sum\nolimits_{l=1}^{L} \{-\mathbb{E}_{\sigma_l}[\log p(\sigma_l)]\}\end{aligned} \tag{4}$$

The first term $\mathcal{L}_{\text{task}}$ denotes the task loss, *e.g.,* the cross-entropy loss for object classification. The second term $\mathcal{L}_{\text{complexity}}$ penalizes the complexity of transformations encoded in the DNN. λ is a positive scalar. To simplify the computation of $p(\sigma_l)$, we use an energy-based model (EBM) (LeCun et al., 2006; Gao et al., 2018) with parameters θ_f. The EBM is a widely-used method to model a high-dimensional distribution (Du & Mordatch, 2019; Pang et al., 2020), *e.g.,* $p_{\theta_f}(\sigma_l)$ in this paper.

$$p_{\theta_f}(\sigma_l) = \frac{1}{Z(\theta_f)} \exp[f(\sigma_l;\theta_f)] \cdot q(\sigma_l), \tag{5}$$

where $q(\sigma_l)$ is a prior distribution defined as $q(\sigma_l) = \prod_d q(\sigma_{l,d})$ and $q(\sigma_{l,d}) \sim \text{Bernoulli}(a_{l,d})$. $f(\sigma_l;\theta_f) \in \mathbb{R}$ is implemented as the scalar output of a ConvNet with parameters

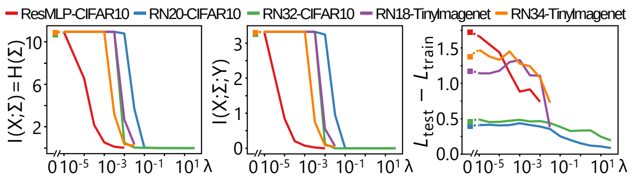


Figure 7. The complexity of transformations and the gap between the training loss and the testing loss of the learned minimum-complexity DNNs. The left-most point in each subfigure at $\lambda = 0$ refers to DNNs learned by only using the task loss.

Table 1. Transformation complexity $H(\Sigma)$ in each layer of normally trained DNNs (termed *normal*) and adversarially trained DNNs (termed *AT*), which were trained on the CIFAR-10 dataset. Adversarially trained DNNs usually exhibited lower transformation complexity.

Model	RN-20		RN-32		RN-44		LeNet	
	Normal	AT	Normal	AT	Normal	AT	Normal	AT
Layer 1	**3.845**	2.979	**2.718**	2.507	**3.938**	1.600	**7.624**	5.358
Layer 2	**6.079**	4.485	**5.660**	4.370	**5.426**	3.374	**7.417**	1.216
Layer 3	**6.671**	6.573	**6.817**	6.786	6.395	**6.828**	**10.966**	10.949

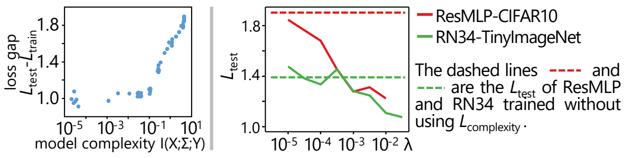


Figure 8. (left) The positive correlation between transformation complexity and the loss gap. (right) Decrease of the testing loss along with the increase of the weight of the complexity loss.

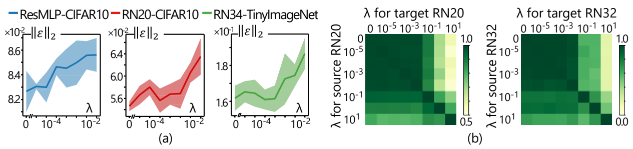


Figure 9. (a) Increase of the minimum L_2 norm of the adversarial perturbations along with the increase of the weight of the complexity loss λ. (b) Adversarial transferability between DNNs learned with different weights of the complexity loss λ.

θ_f and on the input σ_l (Gao et al., 2018). The constant $Z(\theta_f) = \int_{\sigma_l} q(\sigma_l) \exp[f(\sigma_l;\theta_f)] \mathrm{d}\sigma_l$ is for normalization.

The EBM is learned via the maximum likelihood estimation (MLE), $\hat{\theta}_f = \mathrm{argmax}_{\theta_f} \mathbb{E}_x[\log p_{\theta_f}(\sigma_l)_{\text{given } x}]$, where σ_l is the vector of gating states in the l-th gating layer for input sample x. We follow (Gao et al., 2018) to optimize the EBM parameters θ_f with Markov Chain Monte Carlo. Note that the gating state σ_l is not differentiable *w.r.t.* the network parameters. To this end, the ReLU operation can be approximated using the Swish function $\mathrm{ReLU}(x) = x \odot \sigma_l = x \odot \mathrm{sigmoid}(\beta x)$ (Ramachandran et al., 2017), where $\odot$ denotes the element-wise multiplication. This enables us to use $\mathcal{L}_{\text{complexity}}$ to learn network parameters. During the training process, the EBM and the original DNN are trained alternatively. We discuss the computational cost of training DNNs with the complexity loss in Appendix G.8.

Validation of the utility of the complexity loss. The complexity loss reduced the transformation complexity and the gap between the training loss and the testing loss. We added the complexity loss to the last four gating layers in each DNN to train the residual MLP[4], ResNet-20/32 (He et al., 2016a) (RN-20/32) on the CIFAR-10 dataset, and to train ResNet-18/34 (RN-18/34) on the first ten classes in the Tiny ImageNet dataset. We repeatedly trained these DNNs with different values of λ. In particular, when $\lambda = 0$, DNNs were learned only with $\mathcal{L}_{\text{task}}$, which can be taken as baselines.

Figure 7 shows the complexity and the gap between the training loss and the testing loss of DNNs learned with different λ values. We found that $H(\Sigma)$ usually decreased along with the increase of λ. $I(X;\Sigma;Y)$ also decreased along with the increase of λ, which verified that the complexity effectively reduced the model's complexity. We also found that the decrease of transformation complexity reduced the gap between the testing loss and the training loss.

Figure 8(left) demonstrated the positive correlation between the transformation complexity and the gap between the training loss and the testing loss. We assigned different weights λ of the complexity loss to learn residual MLPs with different transformation complexities, using the CIFAR-10 dataset. We found that when the transformation complexity was high, the positive correlation was significant. This positive correlation further validated the effectiveness of the complexity loss on reducing the gap between the training loss and the testing loss. Moreover, Figure 8(right) shows that the testing loss dropped significantly when we increased the weight λ of the complexity loss. Appendix G.6 also shows that the complexity loss was superior to traditional L_1 and L_2 regularization methods, in terms of maintaining the classification accuracy and decreasing the model complexity.

Note that during the training process with the complexity loss, the activation rate $a_{l,d}$ of the l-th gating layer would change (C_l in Eq. (2) may change), thus the entanglement of gating states $TC(\Sigma_l)$ would not necessarily increase along with the decrease of the complexity. The more entangled representations usually indicate more redundant features.

The transformation complexity had a close relationship with adversarial robustness, adversarial transferability, and knowledge consistency. We set different values of the weight λ in Eq. (4) to learn DNNs with different transformation complexities. For each λ value, we repeatedly trained six residual MLPs on the CIFAR-10 dataset, six RN-20/32's on the CIFAR-10 dataset, and six RN-34's on the Tiny ImageNet dataset, with different random initial-

[4]The residual MLP had the similar architecture to the one in the *Findings from residual networks* paragraph in Section 4.2, with 10 FC layers and 9 ReLU layers. Both inputs and features were 3072-d vectors.

izations. We followed (Wang et al., 2020) to measure the minimum L_2 norm of adversarial perturbations computed *w.r.t.* a certain attacking utility to quantify the adversarial robustness of DNNs. Figure 9(a) shows the minimum L_2 norm of adversarial perturbations towards each DNN. We found that DNNs with low transformation complexity usually exhibited high adversarial robustness, while DNNs with high transformation complexity were usually sensitive to adversarial perturbations. Besides, Table 1 compares the transformation complexity in normally trained DNNs and adversarially trained DNNs. Results show that adversarially trained DNNs, which were more robust than normally trained DNNs, usually exhibited lower complexity. This also verifies the negative correlation between transformation complexity and adversarial robustness.

Following settings in (Wang et al., 2020), we also measured the adversarial transferability between DNNs with different transformation complexities. Figure 9(b) shows the adversarial transferability between ResNet-20/32's with different transformation complexities. We found that adversarial perturbations for complex DNNs could not be well transferred to simple DNNs. However, adversarial perturbations for simple DNNs could be transferred to complex DNNs. This reflected that simple DNNs encoded common knowledge that could be transferred to DNNs learned for the same task.

Furthermore, we followed (Liang et al., 2019) to explore the knowledge consistency between DNNs with different transformation complexities. We found that each pair of simple DNNs usually encoded similar knowledge representations (exhibiting high knowledge consistency), while complex DNNs are more likely to encode diverse knowledge. This demonstrated the reliability of features learned by simple DNNs. Please refer to Appendix G.7 for experimental results on knowledge consistency.

Experimental settings. We conducted a set of comparative studies on the task of classification using the MNIST (LeCun et al., 1998), CIFAR-10 (Krizhevsky et al., 2009), CelebA (Liu et al., 2015), Pascal VOC 2012 (Everingham et al., 2015), and Tiny ImageNet (Le & Yang, 2015) datasets. For the MNIST dataset and the CIFAR-10 dataset, we learned LeNet-5 (LeCun et al., 1998), the revised VGG-11 (Jastrzebski et al., 2017), the pre-activation version of ResNet-20/32 (He et al., 2016a;b), and the MLP. In particular, for the MNIST dataset, we learned three MLP models: *MLP-MNIST* contained 5 fully connected (FC) layers with the width of 784-1024-256-128-64-10, *MLP-*α contained 11 FC layers with the width of 784-1024-1024-512-512-256-256-128-128-64-16-10, and *MLP-*β contained 5 FC layers with the width of 784-1024-256-120-84-10[5]. For the CIFAR-10 dataset, the architecture of the MLP was set as 3072-1024-256-128-64-10 (termed *MLP-CIFAR10*). For the CelebA, Pascal VOC 2012, and Tiny ImageNet datasets, we learned VGG-16 (Simonyan et al., 2017) and the pre-activation version of ResNet-18/34. We used images cropped by bounding boxes for both training and testing. Please see Appendix G.3 for these DNNs' classification accuracy. We analyzed the transformation complexity of ReLU layers.

[5]For comparison, we modified the architecture of MLP-MNIST and made the width of the last three FC layers be the same with the fully connected layers in the LeNet-5 network.

5. Conclusion

In this paper, we have proposed three complexity measures for feature transformations encoded in DNNs. We further prove the decrease of the transformation complexity through layerwise propagation. We also prove the strong correlation between the complexity and the disentanglement of transformations. Based on the proposed metrics, we develop a generic method to learn a minimum-complexity DNN, which also reduces the gap between the training loss and the testing loss, and influences adversarial robustness, adversarial transferability, and knowledge consistency. Comparative studies reveal the ceiling of a DNN's complexity. Furthermore, we summarize the change of the transformation complexity during the training process into two typical cases. As a generic tool, the transformation complexity enables us to understand DNNs from new perspectives.

Acknowledgments. This work is partially supported by National Key R&D Program of China (2021ZD0111602), the National Nature Science Foundation of China (No. 61906120, U19B2043), Shanghai Natural Science Foundation (21JC1403800, 21ZR1434600), Shanghai Municipal Science and Technology Major Project (2021SHZDZX0102). This work is also partially supported by Huawei Technologies Inc.

References

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

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

Arora, R., Basu, A., Mianjy, P., and Mukherjee, A. Understanding deep neural networks with rectified linear units. *arXiv preprint arXiv:1611.01491*, 2016.

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

Boob, D., Dey, S. S., and Lan, G. Complexity of training relu neural network. *arXiv preprint arXiv:1809.10787*, 2018.

Burgess, C. P., Higgins, I., Pal, A., Matthey, L., Watters, N., Desjardins, G., and Lerchner, A. Understanding disentangling in β-vae. In *Advances in Neural Information Processing Systems*, volume 31, 2017.

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

Cortes, C., Gonzalvo, X., Kuznetsov, V., Mohri, M., and Yang, S. 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.

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

Du, Y. and Mordatch, I. Implicit generation and modeling with energy based models. In *Advances in Neural Information Processing Systems*, volume 32, pp. 3608–3618. Curran Associates, Inc., 2019.

Everingham, M., Eslami, S. A., Van Gool, L., Williams, C. K., Winn, J., and Zisserman, A. The pascal visual object classes challenge: A retrospective. *International journal of computer vision*, 111(1):98–136, 2015.

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

Fort, S., Nowak, P. K., and Narayanan, S. Stiffness: A new perspective on generalization in neural networks. *arXiv preprint arXiv:1901.09491*, 2019.

Gao, R., Lu, Y., Zhou, J., Zhu, S.-C., and Nian Wu, Y. Learning generative convnets via multi-grid modeling and sampling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9155–9164, 2018.

Goldfeld, Z., Van Den Berg, E., Greenewald, K., Melnyk, I., Nguyen, N., Kingsbury, B., and Polyanskiy, Y. Estimating information flow in deep neural networks. In *International Conference on Machine Learning*, pp. 2299–2308, 2019.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016a.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. In *European conference on computer vision*, pp. 630–645. Springer, 2016b.

Heckel, R. and Yilmaz, F. F. Early stopping in deep networks: Double descent and how to eliminate it. *arXiv preprint arXiv:2007.10099*, 2020.

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. beta-vae: Learning basic visual concepts with a constrained variational framework. *ICLR*, 2(5):6, 2017.

Jastrzebski, S., Kenton, Z., Arpit, D., Ballas, N., Fischer, A., Bengio, Y., and Storkey, A. J. Three factors influencing minima in sgd. *arXiv preprint arXiv:1711.04623*, 2017.

Kalimeris, D., Kaplun, G., Nakkiran, P., Edelman, B., Yang, T., Barak, B., and Zhang, H. Sgd on neural networks learns functions of increasing complexity. In *Advances in Neural Information Processing Systems 32*, pp. 3496–3506. Curran Associates, Inc., 2019.

Kindermans, P.-J., Schütt, K. T., Alber, M., Müller, K.-R., Erhan, D., Kim, B., and Dähne, S. Learning how to explain neural networks: Patternnet and patternattribution. *arXiv preprint arXiv:1705.05598*, 2017.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Kolchinsky, A. and Tracey, B. D. Estimating mixture entropy with pairwise distances. *Entropy*, 19(7):361, 2017.

Kolchinsky, A., Tracey, B. D., and Wolpert, D. H. Nonlinear information bottleneck. *Entropy*, 21(12):1181, 2019.

Kornblith, S., Norouzi, M., Lee, H., and Hinton, G. Similarity of neural network representations revisited. *arXiv preprint arXiv:1905.00414*, 2019.

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

Le, Y. and Yang, X. Tiny imagenet visual recognition challenge. *CS 231N*, 7, 2015.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

LeCun, Y., Chopra, S., Hadsell, R., Ranzato, M., and Huang, F. A tutorial on energy-based learning. *Predicting structured data*, 1(0), 2006.

Liang, R., Li, T., Li, L., and Zhang, Q. Knowledge consistency between neural networks and beyond. In *International Conference on Learning Representations*, 2019.

Liang, T., Poggio, T., Rakhlin, A., and Stokes, J. Fisher-rao metric, geometry, and complexity of neural networks. *arXiv preprint arXiv:1711.01530*, 2017.

Liu, D., Wang, S., Ren, J., Wang, K., Yin, S., and Zhang, Q. Trap of feature diversity in the learning of mlps. *arXiv preprint arXiv:2112.00980*, 2021.

Liu, Z., Luo, P., Wang, X., and Tang, X. Deep learning face attributes in the wild. In *Proceedings of the IEEE international conference on computer vision*, pp. 3730–3738, 2015.

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

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

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

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

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

Nakkiran, P., Kaplun, G., Bansal, Y., Yang, T., Barak, B., and Sutskever, I. Deep double descent: Where bigger models and more data hurt. *arXiv preprint arXiv:1912.02292*, 2019.

Noshad, M., Zeng, Y., and Hero, A. O. Scalable mutual information estimation using dependence graphs. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2962–2966. IEEE, 2019.

Pang, B., Han, T., Nijkamp, E., Zhu, S.-C., and Wu, Y. N. Learning latent space energy-based prior model. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Pascanu, R., Gulcehre, C., Cho, K., and Bengio, Y. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*, 2013.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32, pp. 8026–8037. Curran Associates, Inc., 2019.

Raghu, M., Poole, B., Kleinberg, J., Ganguli, S., and Dickstein, J. S. 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.

Ramachandran, P., Zoph, B., and Le, Q. V. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Ribeiro, M. T., Singh, S., and Guestrin, C. ”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.

Saxe, A. M., Bansal, Y., Dapello, J., Advani, M., Kolchinsky, A., Tracey, B. D., and Cox, D. D. On the information bottleneck theory of deep learning. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124020, 2019.

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., and Batra, D. 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.

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

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

Tishby, N., Pereira, F. C., and Bialek, W. The information bottleneck method. *arXiv preprint physics/0004057*, 2000.

Ver Steeg, G. and Galstyan, A. Maximally informative hierarchical representations of high-dimensional data. In *Artificial Intelligence and Statistics*, pp. 1004–1012, 2015.

Wang, X., Ren, J., Lin, S., Zhu, X., Wang, Y., and Zhang, Q. A unified approach to interpreting and boosting adversarial transferability. *arXiv preprint arXiv:2010.04055*, 2020.

Weng, T.-W., Zhang, H., Chen, P.-Y., Yi, J., Su, D., Gao, Y., Hsieh, C.-J., and Daniel, L. Evaluating the robustness of neural networks: An extreme value theory approach. *arXiv preprint arXiv:1801.10578*, 2018.

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

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

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

Yosinski, J., Clune, J., Nguyen, A., Fuchs, T., and Lipson, H. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

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

Zhang, S., Wu, Y., Che, T., Lin, Z., Memisevic, R., Salakhutdinov, R. R., and Bengio, Y. Architectural complexity measures of recurrent neural networks. In *Advances in neural information processing systems*, pp. 1822–1830, 2016.

Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., and Torralba, A. Object detectors emerge in deep scene cnns. *In ICLR*, 2015.

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

A. Gating States in Gating Layers

In this section, we further discuss gating states of different gating layers, which is mentioned in Section 3 of the paper. Let $h_l \triangleq W_l(\boldsymbol{\sigma}_{l-1}(W_{l-1}\dots(W_2\boldsymbol{\sigma}_1(W_1x+b_1)+b_2)\cdots+b_{l-1}))+b_l$ denote the input of the l-th gating layer. We consider the vectorized form of h_l. Given $h_l \in \mathbb{R}^D$, the formulation of $\boldsymbol{\sigma}_l$ in different gating layers is given as follows.

(1) ReLU layer. In this case, $\boldsymbol{\sigma}_l = diag(\sigma_l^1, \sigma_l^2, \dots, \sigma_l^D) \in \{0,1\}^D$, which is a diagonal matrix. If the d-th dimension of h_l is larger than 0, then we have $\sigma_l^d = 1$; otherwise, $\sigma_l^d = 0$.

(2) Dropout layer. In this case, $\boldsymbol{\sigma}_l = diag(\sigma_l^1, \sigma_l^2, \dots, \sigma_l^D) \in \{0,1\}^D$, which is a diagonal matrix. If the d-th dimension of h_l is not dropped, then we have $\sigma_l^d = 1$; otherwise, $\sigma_l^d = 0$.

(3) Max-Pooling layer. Since a pooling layer may change the size of the input, $\boldsymbol{\sigma}_l$ is not necessarily a square matrix. Let the output of the max-pooling layer be $\boldsymbol{\sigma}_l h_l \in \mathbb{R}^{D'}$, *i.e.* the input h_l is divided into D' regions. In this case, we have $\boldsymbol{\sigma}_l \in \{0,1\}^{D'\times D}$. If $(h_l)_{d'}$ is the largest element in the d'-th region, then we have $(\boldsymbol{\sigma}_l)_{d'd} = 1$; otherwise, $(\boldsymbol{\sigma}_l)_{d'd} = 0$.

B. Proofs of Important Conclusions

This section proves the three properties mentioned in Section 3 of the paper, and also gives detailed proofs for other important conclusions in the paper.

B.1. Non-negativity of the complexity $I(X;\Sigma;Y)$

Property 1. *If the DNN does not introduce information besides the input X (*e.g.* there is no sampling operations or dropout operations throughout the DNN), we have $I(X;\Sigma;Y) \geq 0$.*

Proof. Recall that the mutual information is defined as

$$\begin{aligned} I(X;\Sigma;Y) =& I(X;Y) - I(X;Y|\Sigma) \\ =& (H(Y) - H(Y|X)) - (H(Y|\Sigma) - H(Y|X,\Sigma)) \\ =& (H(Y) - H(Y|\Sigma)) - (H(Y|X) - H(Y|X,\Sigma)) \end{aligned} \tag{6}$$

If the DNN does not introduce additional information besides X, which means that Σ is determined by X, then we have $H(Y|X) - H(Y|X,\Sigma) = I(\Sigma;Y|X) = 0$. Therefore,

$$I(X;\Sigma;Y) = H(Y) - H(Y|\Sigma) \geq 0 \tag{7}$$

B.2. Increase of the complexity along with the increase of gating layers

Property 2. The complexity of transforming the input x increases when we consider more gating layers.

- $H(\Sigma_1,\dots,\Sigma_l) \leq H(\Sigma_1,\dots,\Sigma_{l+1})$

Proof.

$$\begin{aligned} & H(\Sigma_1,\dots,\Sigma_l) - H(\Sigma_1,\dots,\Sigma_{l+1}) \\ =& -H(\Sigma_{l+1}|\Sigma_1,\dots,\Sigma_l) \\ =& \mathbb{E}_{\sigma_1,\dots,\sigma_{l+1}}\left[\log p(\sigma_{l+1}|\sigma_1,\dots,\sigma_l)\right] \\ \leq& 0 \end{aligned} \tag{8}$$

- $I(X;\{\Sigma_1,\dots,\Sigma_1\}) \leq I(X;\{\Sigma_1,\dots,\Sigma_{l+1}\})$

Proof. If the DNN does not introduce additional information besides the input during the forward propagation, then $\Sigma_1,\dots,\Sigma_l,\Sigma_{l+1}$ are all determined by X, thereby $H(\Sigma_1,\dots,\Sigma_l|X) = H(\Sigma_1,\dots,\Sigma_{l+1}|X) = 0$. Therefore,

$$\begin{aligned} & I(X;\{\Sigma_1,\dots,\Sigma_l\}) - I(X;\{\Sigma_1,\dots,\Sigma_{l+1}\}) \\ =& (H(\Sigma_1,\dots,\Sigma_l) - H(\Sigma_1,\dots,\Sigma_l|X)) - (H(\Sigma_1,\dots,\Sigma_{l+1}) - H(\Sigma_1,\dots,\Sigma_{l+1}|X)) \\ =& H(\Sigma_1,\dots,\Sigma_l) - H(\Sigma_1,\dots,\Sigma_{l+1}) \\ \leq& 0 \end{aligned} \tag{9}$$

• $I(X;\{\Sigma_1,\dots,\Sigma_l\};Y) \geq I(X;\{\Sigma_1,\dots,\Sigma_{l+1}\};Y)$

Proof. According to Eq. (7), if there is no additional information besides the input throughout the DNN, then

$$I(X;\{\Sigma_1,\dots,\Sigma_l\};Y) = H(Y) - H(Y|\{\Sigma_1,\dots,\Sigma_l\}) \tag{10}$$

We can obtain the following inequality:

$$\begin{aligned}
&I(X;\{\Sigma_1,\dots,\Sigma_l\};Y) - I(X;\{\Sigma_1,\dots,\Sigma_{l+1}\};Y)\\
=&(H(Y) - H(Y|\{\Sigma_1,\dots,\Sigma_l\})) - (H(Y) - H(Y|\{\Sigma_1,\dots,\Sigma_{l+1}\}))\\
=&H(Y|\{\Sigma_1,\dots,\Sigma_{l+1}\}) - H(Y|\{\Sigma_1,\dots,\Sigma_l\})\\
=&-I(\Sigma_{l+1};Y|\{\Sigma_1,\dots,\Sigma_l\})\\
\leq&0
\end{aligned} \tag{11}$$

B.3. Decrease of the complexity through layerwise propagation

Property 3. The complexity of transforming the intermediate-layer feature t_l to the output y decreases, when we use the feature of higher layers.

• $H(\Sigma_l,\dots,\Sigma_L) \geq H(\Sigma_{l+1},\dots,\Sigma_L)$

Proof.

$$\begin{aligned}
&H(\Sigma_l,\dots,\Sigma_L) - H(\Sigma_{l+1},\dots,\Sigma_L)\\
=&H(\Sigma_l|\Sigma_{l+1},\dots,\Sigma_L)\\
=&-\mathbb{E}_{\sigma_l,\dots,\sigma_L}\left[\log p(\sigma_l|\sigma_{l+1},\dots,\sigma_L)\right]\\
\geq&0
\end{aligned} \tag{12}$$

• $I(T_{l-1};\{\Sigma_l,\dots,\Sigma_L\}) \geq I(T_l;\{\Sigma_{l+1},\dots,\Sigma_L\})$

Proof. If the DNN does not introduce additional information besides the input during the forward propagation, then $\Sigma_l,\Sigma_{l+1},\dots,\Sigma_L$ are all determined by T_{l-1}, thereby $H(\Sigma_l,\dots,\Sigma_L|T_{l-1}) = 0$. Therefore,

$$\begin{aligned}
&I(T_{l-1};\{\Sigma_l,\dots,\Sigma_L\}) - I(T_l;\{\Sigma_{l+1},\dots,\Sigma_L\})\\
=&(H(\Sigma_l,\dots,\Sigma_L) - H(\Sigma_l,\dots,\Sigma_L|T_{l-1})) - (H(\Sigma_{l+1},\dots,\Sigma_L) - H(\Sigma_{l+1},\dots,\Sigma_L|T_l))\\
=&H(\Sigma_l,\dots,\Sigma_L) - H(\Sigma_{l+1},\dots,\Sigma_L)\\
\geq&0
\end{aligned} \tag{13}$$

• $I(T_{l-1};\{\Sigma_l,\dots,\Sigma_L\};Y) \geq I(T_l;\{\Sigma_{l+1},\dots,\Sigma_L\};Y)$

Proof. According to Eq. (7), if there is no additional information besides the input throughout the DNN, then

$$I(T_{l-1};\{\Sigma_l,\dots,\Sigma_L\};Y) = H(Y) - H(Y|\{\Sigma_l,\dots,\Sigma_L\}) \tag{14}$$

We can obtain the following inequality:

$$\begin{aligned}
&I(T_{l-1};\{\Sigma_l,\dots,\Sigma_L\};Y) - I(T_l;\{\Sigma_{l+1},\dots,\Sigma_L\};Y)\\
=&(H(Y) - H(Y|\{\Sigma_l,\dots,\Sigma_L\})) - (H(Y) - H(Y|\{\Sigma_{l+1},\dots,\Sigma_L\}))\\
=&H(Y|\{\Sigma_{l+1},\dots,\Sigma_L\}) - H(Y|\{\Sigma_l,\dots,\Sigma_L\})\\
=&I(\Sigma_l;Y|\{\Sigma_{l+1},\dots,\Sigma_L\})\\
\geq&0
\end{aligned} \tag{15}$$

B.4. Strong correlations between the complexity and the disentanglement of transformations

Some previous studies used the entanglement (the multi-variate mutual information) to analyze the information encoded in DNNs. (Ver Steeg & Galstyan, 2015) used $TC(X)$ to measure the correlation between different input samples. In comparison, in this paper, we apply $TC(\Sigma)$ to measure the independence between gating states of different dimensions. Intuitively, the disentanglement of gating states does not seem related to the complexity. Therefore, our contribution is to find out the strong correlation between the two factors which seem not related.

We consider the complexity of the transformation of a single gating layer, *e.g.* $H(\Sigma_l)$, $I(X;\Sigma_l)$ and $I(X;\Sigma_l;Y)$ for the l-th gating layer.

• $H(\Sigma_l)$

Proof.

$$\begin{aligned} H(\Sigma_l) + TC(\Sigma_l) =& H(\Sigma_l) + KL(p(\sigma_l)||\prod_d p(\sigma_l^d)) \\ =& \mathbb{E}_{\sigma_l}\left[\log \frac{1}{p(\sigma_l)}\right] + \mathbb{E}_{\sigma_l}\left[\log \frac{p(\sigma_l)}{\prod_d p(\sigma_l^d)}\right] \\ =& -\mathbb{E}_{\sigma_l}\left[\log \prod_d p(\sigma_l^d)\right] \quad \backslash\backslash\ p(\sigma_l^d) \text{ does not depend on the input} \\ =& C_l \end{aligned} \tag{16}$$

Let us consider DNNs with similar activation rates a_l^d. Because $p(\sigma_l^d)$ follows the Bernoulli distribution with the activation rate a_l^d, for DNNs with similar activation rates a_l^d, they share similar values of C_l. In this case, there is a negative correlation between $H(\Sigma_l)$ and $TC(\Sigma_l)$.

•$I(X;\Sigma_l)$

Proof.

$$\begin{aligned} I(X;\Sigma_l) + TC(\Sigma_l) =& H(\Sigma_l) - H(\Sigma_l|X) + KL(p(\sigma_l)||\prod_d p(\sigma_l^d)) \\ =& C_l - H(\Sigma_l|X) \end{aligned} \tag{17}$$

If the DNN does not introduce additional information through the layerwise propagation, then the X determines Σ_l, *i.e.* $H(\Sigma_l|X) = 0$. Thus, for DNNs with similar values of C_l, there is a negative correlation between $I(X;\Sigma_l)$ and $TC(\Sigma_l)$.

•$I(X;\Sigma_l;Y)$

Proof. According to the definition of $I(X;\Sigma_l;Y)$, we have

$$I(X;\Sigma_l;Y) = I(X;\Sigma_l) - I(X;\Sigma_l|Y) \tag{18}$$

We have discussed the first term $I(X;\Sigma_l)$ above, so we focus on the second term $I(X;\Sigma_l|Y)$, which measures the complexity of transformations that are unrelated to the inference. Similarly, the entanglement of the inference-irrelevant transformations is represented by

$$TC(\Sigma_l|Y) = \mathbb{E}_y(KL(p(\sigma_l|y)||\prod_d p(\sigma_l^d|y))) \tag{19}$$

Then, we have

$$\begin{aligned} & I(X;\Sigma_l|Y) + TC(\Sigma_l|Y) \\ =& H(\Sigma_l|Y) - H(\Sigma_l|X,Y) + TC(\Sigma_l|Y) \\ =& \mathbb{E}_y\left[H(\Sigma_l|y) + KL(p(\sigma_l|y)||\prod_d p(\sigma_l^d|y))\right] - H(\Sigma_l|X,Y) \\ =& \mathbb{E}_{\sigma_l,y}\left[\log\frac{1}{p(\sigma_l|y)} + \log\frac{p(\sigma_l|y)}{\prod_d p(\sigma_l^d|y)}\right] - H(\Sigma_l|X,Y) \\ =& -\mathbb{E}_{\sigma_l,y}\left[\log\prod_d p(\sigma_l^d|y)\right] - H(\Sigma_l|X,Y) \\ =& C_{l|Y} - H(\Sigma_l|X,Y) \end{aligned} \tag{20}$$

If there is no additional information besides the input in the DNN, then $H(\Sigma_l|X,Y)=0$. Thus, we have

$$\begin{aligned} I(X;\Sigma_l)+TC(\Sigma_l)=C_l-H(\Sigma_l|X)=C_l \\ I(X;\Sigma_l|Y)+TC(\Sigma_l|Y)=C_{l|Y}-H(\Sigma_l|X,Y)=C_{l|Y} \end{aligned} \tag{21}$$

Therefore,

$$\begin{aligned} I(X;\Sigma_l;Y) &= I(X;\Sigma_l)-I(X;\Sigma_l|Y) \\ &= (C_l-TC(\Sigma_l))-(C_{l|Y}-TC(\Sigma_l|Y)) \\ &= (C_l-C_{l|Y})-\underbrace{(TC(\Sigma_l)-TC(\Sigma_l|Y))}_{\substack{\text{multi-variate mutual information} \\ \text{used to infer } Y}} \end{aligned} \tag{22}$$

where the difference between $TC(\Sigma_l)$ and $TC(\Sigma_l|Y)$ represents the entanglement of the transformations that are used to infer Y. For DNNs with similar activation rates, we can also roughly consider that these DNNs share similar values of C_l and $C_{l|Y}$. Thus, we can conclude that the higher complexity makes the DNN use more disentangled transformation for inference.

C. About Values of $C_{l|Y}$

In experiments, we found that in most cases, for DNNs with similar activation rates a_l^d in their corresponding layers, these DNNs usually shared similar values of $C_{l|Y}$. However, in some extreme cases, *e.g.* when the DNN was learned from very few training samples, or when the target layer was very close to the input layer or the output layer, values of $C_{l|Y}$ of these DNNs were different from those values of other DNNs.

D. Intuitive Examples for the Entanglement of Features

This section introduces intuitive examples for the two cases of feature entanglement mentioned in Section 4.1 of the paper.

For the first case, if we use an extremely simple DNN (*e.g.* a two-layer neural network) to classify complex images, it is likely that the network fails to extract features for meaningful concepts. The lack of representation power (transformation complexity) of a DNN usually leads to high entanglement of features and hurts the classification performance. In other words, the intermediate-layer features in the DNN are still highly entangled, just like the entanglement of pixel colors in the image.

For the second case, if we use a sophisticated enough DNN (*e.g.* a VGG-16) to classify very simple digits in the MNIST dataset. Then, features in high layers (*e.g.* the `conv5-3` layer) are redundant to represent the simple knowledge in the digits, thus leading to entanglement of features. In this case, due to the feature redundancy, features in some channels of the `conv5-3` layer may be similar to each other, which can be considered as feature entanglement. Such an entanglement (redundancy) of reliable features is good for classification.

E. Detailed Explanation of the KDE Method

The kernel density estimation (KDE) method was proposed by (Kolchinsky & Tracey, 2017), and has been considered as a standard method to estimate the entropy and the mutual information. In this section, we briefly summarize key techniques of the KDE method, which are used to quantify the transformation complexity.

The KDE approach was proposed to estimate the mutual information between the input X and the feature of an intermediate layer T in a DNN (Kolchinsky & Tracey, 2017; Kolchinsky et al., 2019). The KDE approach assumes that the intermediate-layer feature is distributed as a mixture of Gaussians. Since T is a continuous variable, $H(T)$ can be negative. The KDE method transforms each feature point into a local Gaussian distribution to approximate the accurate feature distribution. Let $\hat{T}=T+\epsilon$ where $\epsilon\sim\mathcal{N}(0,\sigma_0^2 I)$. Then, the distribution of $\hat{T}$ can be considered as a mixture of Gaussians, with a Gaussian centered on T. In this setting, previous studies (Kolchinsky & Tracey, 2017; Kolchinsky et al., 2019; Saxe et al., 2019) shows that an upper bound for the mutual information with the input is

$$I(\hat{T};X)\le -\frac{1}{P}\sum_i \log\frac{1}{P}\sum_j \exp\left(-\frac{1}{2}\frac{||t_i-t_j||^2}{\sigma_0^2}\right) \tag{23}$$

where P is the number of training samples, and t_i denotes the intermediate-layer feature of the input sample i. Similarly, the upper bound for the mutual information *w.r.t* the output Y can be calculated as

$$\begin{aligned} I(\hat{T};Y) =& H(\hat{T}) - H(\hat{T}|Y) \\ =& -\frac{1}{P}\sum_i \log\frac{1}{P}\sum_j \exp\left(-\frac{1}{2}\frac{||t_i-t_j||^2}{\sigma_0^2}\right) \\ & -\sum_{l=1}^{L} p_l \left[-\frac{1}{P}\sum_{i:Y_i=l}\log\frac{1}{P}\sum_{j:Y_j=l}\exp\left(-\frac{1}{2}\frac{||t_i-t_j||^2}{\sigma_0^2}\right)\right] \end{aligned} \tag{24}$$

where L is the number of categories. P_l denotes the number of samples belonging to the l-th category. $p_l = P_l/P$ denotes the probability of the category l.

We use the KDE method to quantify the transformation complexity $H(\Sigma)$, $I(X;\Sigma)$ and $I(X;\Sigma;Y)$. The entropy of gating states $H(\Sigma_l)$ is quantified as follows.

$$H(\Sigma_l) \leq -\frac{1}{n}\sum_{j=1}^{n}\log\frac{1}{n}\sum_{k=1}^{n}\exp\left(-\frac{1}{2}\frac{\|\sigma_{l,j}-\sigma_{l,k}\|_2^2}{\sigma_0^2}\right) \tag{25}$$

where n denotes the number of training samples. $\sigma_{l,j}$ and $\sigma_{l,k}$ denote the vectorized gating states of the l-th gating layer for the j-th sample and the k-th sample, respectively. σ_0^2 is quantified as $\sigma_0^2 = \kappa \cdot Var(\Sigma_l)$, where $Var(\Sigma_l) = \mathbb{E}_x[\|\sigma_l - \mu\|^2], \mu = \mathbb{E}_x[\sigma_l]$. $Var(\Sigma_l)$ measures the variance of gating states of the l-th gating layer. κ is a positive constant. The above equation can also be used to quantify $H(\Sigma)$, when we simply replace Σ_l with Σ. Figure 10 shows the complexity $I(X;\Sigma)$ and $I(\Sigma;Y)$, which were calculated on MLP-α networks learned with different κ values on the MNIST dataset. The κ value affected the scale of the complexity value, but it did not affect the trend of the complexity change during the training process. Thus, given a fixed κ value, the complexity of different DNNs could be fairly compared.

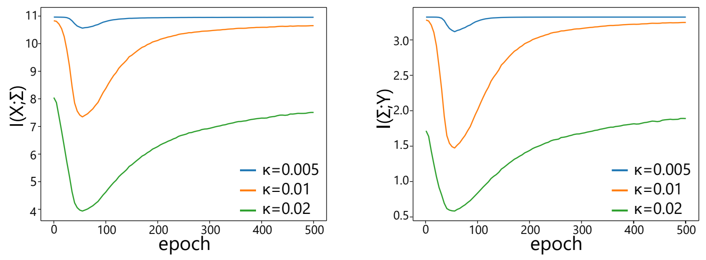


Figure 10. The complexity calculated with different values of κ. The κ value only affected the scale of $I(X;\Sigma)$, but it did not affect the trend of the complexity change. Thus, given a fixed κ value, the complexity of different DNNs can be fairly compared.

If the DNN does not introduce additional complexity besides the input X, we have $H(\Sigma_l|X) = 0$, $I(X;\Sigma_l) = H(\Sigma_l) - H(\Sigma_l|X) = H(\Sigma_l)$. If the DNN introduces additional complexity (*e.g.* using the sampling operation in VAE, or the dropout operation), then $I(X;\Sigma_l)$ can be quantified as follows.

$$I(X;\Sigma_l) \leq -\frac{1}{n}\sum_{j=1}^{n}\log\frac{1}{n}\sum_{k=1}^{n}\exp\left(-\frac{1}{2}\frac{\|\hat{\sigma}_{l,j}-\hat{\sigma}_{l,k}\|_2^2}{\sigma_0^2}\right) \tag{26}$$

where $\hat{\sigma}_{l,j}$ and $\hat{\sigma}_{l,k}$ represent the vectorized gating states when sampling operations are removed (in this way, we can use the method of measuring $H(\Sigma_l)$ to quantify $I(X;\Sigma_l)$).

Similarly, the complexity $I(X;\Sigma_l;Y)$ can be estimated by its upper bound:

$$\begin{aligned} I(X;\Sigma_l;Y) &= I(\Sigma_l;Y) - I(\Sigma_l;Y|X) \\ &= H(\Sigma_l) - H(\Sigma_l|Y) - I(\Sigma_l;Y|X) \\ &\leq -\frac{1}{n}\sum_{j=1}^{n}\log\frac{1}{n}\sum_{k=1}^{n}\exp\left(-\frac{1}{2}\frac{\|\sigma_{l,j}-\sigma_{l,k}\|_2^2}{\sigma_0^2}\right) \\ &\quad -\sum_{m=1}^{M}p_m\left[-\frac{1}{n_m}\sum_{\substack{j,\\ Y_j=m}}\log\frac{1}{n_m}\sum_{\substack{k,\\ Y_k=m}}\exp\left(-\frac{1}{2}\frac{\|\sigma_{l,j}-\sigma_{l,k}\|_2^2}{\sigma_0^2}\right)\right] \\ &\quad - I(\Sigma_l;Y|X) \end{aligned} \tag{27}$$

For the task of multi-category classification, M denotes the number of categories. n_m is the number of training samples belonging to the m-th category, and $p_m = n_m/M$. If the DNN does not introduce additional complexity besides the input, we have $I(\Sigma_l;Y|X)=0$.

The entanglement of transformations is formulated as

$$TC(\Sigma_l) = KL(p(\sigma_l)||\prod_d p(\sigma_l^d)) = \mathbb{E}_{\sigma_l}\left[\log\frac{p(\sigma_l)}{\prod_d p(\sigma_l^d)}\right] \tag{28}$$

where $p(\sigma_l^d)$ denotes the marginal distribution of the d-th element in σ_l. To enable fair comparisons between $I(\hat{T},X)$ computed by the KDE method in Eq. (23) and $TC(\Sigma_l)$, we also apply the KDE method to approximate $TC(\Sigma_l)$. To this end, we synthesize a new distribution $p(\hat{\sigma}_l)$ to represent the distribution of $\prod_d p(\sigma_l^d)$. In $\hat{\sigma}_l$, $\hat{\sigma}_l^d$ in each dimension follows the Bernoulli distribution with the same activation rate a_l^d with the original σ_l^d. Gating states $\hat{\sigma}_l^d$ in different dimensions are independent with each other. In this way, $\prod_d p(\sigma_l^d)$ can be approximated by $p(\hat{\sigma}_l)$.

Inspired by (Kolchinsky & Tracey, 2017; Kolchinsky et al., 2019), $TC(\Sigma_l)$ is quantified as the following upper bound.

$$\begin{aligned} TC(\Sigma_l) &= \mathbb{E}_{\sigma_l}\left[\log\frac{p(\sigma_l)}{p(\hat{\sigma}_l)}\right] \\ &\leq \frac{1}{P}\sum_i \log\frac{\sum_j \exp\left(-\frac{1}{2}\frac{\|\sigma_{l,i}-\sigma_{l,j}\|_2^2}{\sigma_0^2}\right)}{\sum_j \exp\left(-\frac{1}{2}\frac{\|\sigma_{l,i}-\hat{\sigma}_{l,j}\|_2^2}{\sigma_0^2}\right)} \end{aligned} \tag{29}$$

where P denotes the number of samples. $\hat{\sigma}_{l,i}$ denotes the synthesized gating states, which have the same activation rates with the gating states of the sample i.

F. Learning a Minimum-Complexity DNN

This section introduces more details about the learning of a minimum-complexity DNN in Section 4.3 of the paper. In Section 4.3, the complexity loss is defined as

$$\mathcal{L}_{\text{complexity}} = \sum_{l=1}^{L} H(\Sigma_l) = \sum_{l=1}^{L}\{-\mathbb{E}_{\sigma_l}[\log p(\sigma_l)]\} \tag{30}$$

The exact value of $p(\sigma_l)$ is difficult to calculate. Thus, inspired by (Gao et al., 2018), we design an energy-based model (EBM) $p_{\theta_f}(\sigma_l)$ to approximate it, as follows.

$$\begin{aligned} p_{\theta_f}(\sigma_l) &= \frac{1}{Z(\theta_f)}\exp[f(\sigma_l;\theta_f)]\cdot q(\sigma_l) \\ Z(\theta_f) &= \mathbb{E}_q\left[\exp[f(\sigma_l;\theta_f)]\right] = \int_{\sigma_l} q(\sigma_l)\exp[f(\sigma_l;\theta_f)]\mathrm{d}\sigma_l \end{aligned} \tag{31}$$

where $q(\sigma_l)$ denotes the prior distribution, which is formulated as follows.

$$q(\sigma_l) = \prod_d q(\sigma_l^d), \quad q(\sigma_l^d) \quad = \begin{cases} \hat{p} & \sigma_l^d = 1 \\ 1 - \hat{p} & \sigma_l^d = 0 \end{cases} \tag{32}$$

If we write the EBM as $p_{\theta_f}(\sigma_l) = \frac{1}{Z(\theta_f)} \exp[-\mathcal{E}(\sigma_l)]$, then the energy function is as follows.

$$\mathcal{E}_{\theta_f}(\sigma_l) = -\log q(\sigma_l) - f(\sigma_l; \theta_f) \tag{33}$$

The EBM can be learned via the maximum likelihood estimation (MLE) with the following loss.

$$\hat{\theta}_f = \arg\max_{\theta_f} L(\theta_f) = \arg\max_{\theta_f} \frac{1}{n} \sum_{i=1}^{n} \log p_{\theta_f}(\sigma_{l,i}) \tag{34}$$

where n denotes the number of samples. $\sigma_{l,i}$ is a vector, which represents gating states in the l-th gating layer for the i-th sample.

The loss and gradient of θ_f can be calculated as follows.

$$L(\theta_f) = -\frac{1}{n} \sum_{i=1}^{n} \log p_{\theta_f}(\sigma_{l,i}) = -\frac{1}{n} \sum_{i=1}^{n} [f(\sigma_{l,i}; \theta_f) + \log q(\sigma_{l,i})] + \log Z(\theta_f) \tag{35}$$

$$\frac{\partial L(\theta_f)}{\partial \theta_f} = \mathbb{E}_{\theta_f}\left[\frac{\partial}{\partial \theta_f} f(\sigma_l; \theta_f)\right] - \frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial \theta_f} f(\sigma_{l,i}; \theta_f) \tag{36}$$

where $\frac{\partial}{\partial \theta_f} \log Z(\theta_f) = \mathbb{E}_{\theta_f}[\frac{\partial}{\partial \theta_f} f(\sigma_l; \theta_f)]$.

The first term $\mathbb{E}_{\theta_f}\left[\frac{\partial}{\partial \theta_f} f(\sigma_l; \theta_f)\right]$ in the above equation is analytically intractable and has to be approximated by MCMC, such as the Langevin dynamics.

$$\begin{aligned} \sigma_l^{\text{new}} &= \sigma_l - \frac{\Delta\tau}{2} \frac{\partial}{\partial \sigma_l} \mathcal{E}_{\theta_f}(\sigma_l) + \sqrt{\Delta\tau}\epsilon \\ &= \sigma_l + \frac{\Delta\tau}{2} \left[\frac{\partial f(\sigma_l; \theta_f)}{\partial \sigma_l} + \sum_{d=1}^{D} \frac{1}{q(\sigma_l^d)} \frac{\partial q(\sigma_l^d)}{\partial \sigma_l^d}\right] + \sqrt{\Delta\tau}\epsilon \end{aligned} \tag{37}$$

where $\epsilon \backsim N(\mathbf{0}, \mathbf{I})$ is a Gaussian white noise. $\Delta\tau$ denotes the size of the Langevin step.

Then, the Monte Carlo approximation to $\frac{\partial L(\theta_f)}{\partial \theta_f}$ is given as follows.

$$\begin{aligned} \frac{\partial L(\theta_f)}{\partial \theta_f} &\approx \frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial \theta_f} f(\widetilde{\sigma}_{l,i}; \theta_f) - \frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial \theta_f} f(\sigma_{l,i}; \theta_f) \\ &= \frac{\partial}{\partial \theta_f} \left[\frac{1}{n} \sum_{i=1}^{n} \mathcal{E}_{\theta_f}(\sigma_{l,i}) - \frac{1}{n} \sum_{i=1}^{n} \mathcal{E}_{\theta_f}(\widetilde{\sigma}_{l,i})\right] \end{aligned} \tag{38}$$

where $\widetilde{\sigma}_{l,i}$ is the sample synthesized via Langevin dynamics.

Thus, the loss for the learning of the DNN can be rewritten as follows.

$$\begin{aligned} \mathcal{L}_{\text{complexity}} &= -\sum_{l=1}^{L} \mathbb{E}_{\sigma_l}[\log p_{\hat{\theta}_f}(\sigma_l)] \\ &= \frac{1}{n} \sum_{l=1}^{L} \sum_{i=1}^{n} [\mathcal{E}_{\hat{\theta}_f}(\sigma_{l,i}) + \log Z(\hat{\theta}_f)] \\ &= -\frac{1}{n} \sum_{l=1}^{L} \sum_{i=1}^{n} [f(\sigma_{l,i}; \hat{\theta}_f) + \log q(\sigma_{l,i}) - \log Z(\hat{\theta}_f)] \end{aligned} \tag{39}$$

Let θ_{DNN} denote parameters in the DNN. The gradient of θ_{DNN} can be calculated as follows.

$$\frac{\partial Loss}{\partial \theta_{\text{DNN}}} = -\frac{1}{n}\sum_{l=1}^{L}\sum_{i=1}^{n}\left\{\frac{\partial f(\sigma_l;\hat{\theta}_f)}{\partial \sigma_{l,i}} + \sum_{d}^{D}\frac{1}{q(\sigma_{l,i}^d)}\frac{\partial q(\sigma_{l,i}^d)}{\partial \sigma_{l,i}^d}\right\}\frac{\partial \sigma_{l,i}}{\partial \theta_{\text{DNN}}} \tag{40}$$

We consider $Z(\theta_f)$ as a constant in the computation of $\frac{\partial Loss}{\partial \hat{\theta}_{\text{DNN}}}$.

To enable the computation of $\frac{\partial \sigma_{l,i}}{\partial \theta_{DNN}}$ and $\frac{\partial q(\sigma_{l,i}^d)}{\partial \sigma_{l,i}^d}$, we can approximate the ReLU operation using the following Swish function (Ramachandran et al., 2017).

$$\begin{aligned}\sigma_l &\approx \text{sigmoid}(\beta x)\\ \text{ReLU}(x) = x \odot \sigma_l &\approx x \odot \text{sigmoid}(\beta x)\end{aligned} \tag{41}$$

where $\odot$ denotes the element-wise multiplication.

According to Eq. (32), the prior distribution $q(\sigma_l)$ is approximated as follows.

$$q(\sigma_l^d) \approx 1 - \hat{p} + \sigma_l^d(2\hat{p}-1), \quad \frac{\partial q(\sigma_l^d)}{\partial \sigma_l^d} \approx 2\hat{p} - 1 \tag{42}$$

In implementation, the EBM is a bottom-up ConvNet with six convolutional layers, which takes σ_l as an input and outputs a scalar. During the training phase, we firstly train the EBM using Eq. (38) for every batch of training data. The EBM and the original DNN are trained separately. *I.e.* when training the EBM, parameters in the original DNN are fixed, and vice versa.

G. More Experimental Details, Results, and Discussions

G.1. More experimental details

Recall that $\Sigma_l = \{\sigma_l\}$ denotes the set of gating states σ_l among all samples X. In this paper, we randomly sample 2000 images from the training set of the each dataset for the calculation of the transformation complexity. Thus, X denotes the set of 2000 randomly sampled images that are used for analysis.

G.2. The value of κ used in the KDE approach

In this section, we discuss about the value of the hyper-parameter κ used in the KDE approach. Note that the features of convolutional layers usually contain far more dimensions than features of fully-connected layers. Therefore, we set $\kappa = 0.04$ for gating layers following each convolutional layer, and $\kappa = 0.01$ for gating layers following each FC layer.

We also tested the effects of different κ values to the quantification of the complexity. Figure 10 shows the complexity $I(X;\Sigma)$ and $I(\Sigma;Y)$, which were calculated on MLP-α networks learned with different κ values on the MNIST dataset. The κ value affected the scale of the complexity value, but it did not affect the trend of the complexity change during the training process. Thus, given a fixed κ value, the complexity of different DNNs could be fairly compared.

G.3. The classification accuracy of DNNs in comparative studies

This section contains more details of DNNs in comparative studies of the paper. We trained five types of DNNs on the MNIST dataset and the CIFAR-10 dataset, and trained three types of DNNs on the CelebA dataset and the Pascal VOC 2012 dataset. Table 2 reports the testing accuracy of the trained DNNs.

Table 2. The classification accuracy of DNNs on different datasets.

(a) On the MNIST and CIFAR-10 datasets.

	MLP	LeNet-5	revised VGG-11	ResNet-20	ResNet-32
MNIST	96.52%	97.41%	99.00%	98.70%	98.24%
CIFAR-10	52.52%	61.5%	84.53%	81.75%	79.76%

(b) On the CelebA and Pascal VOC 2012 datasets.

	ResNet-18	ResNet-34	VGG-16
CelebA	80.25%	80.91%	89.70%
Pascal VOC 2012	67.99%	64.27%	62.50%

G.4. The change of the transformation complexity in VGG-16

This section shows more results of the change of transformation complexity in Section 4.2. Figure 11 shows the change of transformation complexity in VGG-16 trained on the Pascal VOC dataset and the CelebA dataset. In these cases, the complexity increased monotonously during the early stage of the training process, and saturated later.

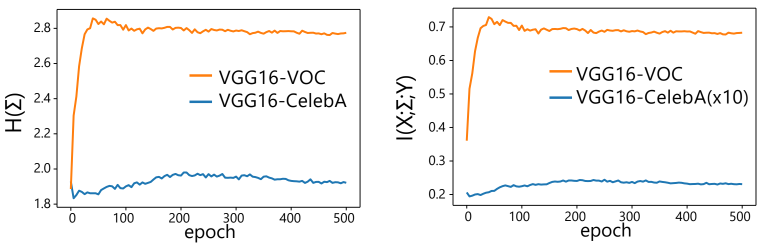
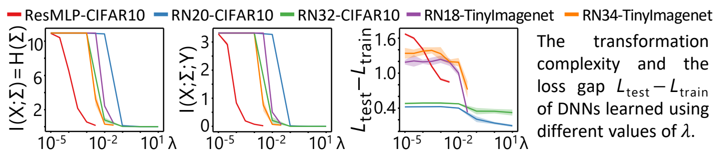


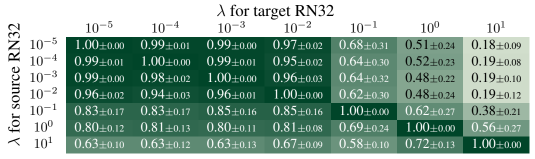
Figure 11. The change of transformation complexity in VGGs.

G.5. More experimental result of learning DNNs with minimum complexity

In this section, we provide more experimental result to verify the stability of learning DNNs with minimum complexity in Section 4.3. Specifically, we repeated experiments in Figure 7 and Figure 9(b) for six times with different random initializations. Results in Figure 12 with the standard deviation still verify our conclusions.



The transformation complexity and the loss gap $L_{\text{test}} - L_{\text{train}}$ of DNNs learned using different values of λ.



λ for source RN32 \ λ for target RN32	10^{-5}	10^{-4}	10^{-3}	10^{-2}	10^{-1}	10^{0}	10^{1}
10^{-5}	$1.00_{\pm 0.00}$	$0.99_{\pm 0.01}$	$0.99_{\pm 0.00}$	$0.97_{\pm 0.02}$	$0.68_{\pm 0.31}$	$0.51_{\pm 0.24}$	$0.18_{\pm 0.09}$
10^{-4}	$0.99_{\pm 0.01}$	$1.00_{\pm 0.00}$	$0.99_{\pm 0.01}$	$0.95_{\pm 0.02}$	$0.64_{\pm 0.30}$	$0.52_{\pm 0.23}$	$0.19_{\pm 0.08}$
10^{-3}	$0.99_{\pm 0.00}$	$0.98_{\pm 0.02}$	$1.00_{\pm 0.00}$	$0.96_{\pm 0.03}$	$0.64_{\pm 0.32}$	$0.48_{\pm 0.22}$	$0.19_{\pm 0.10}$
10^{-2}	$0.96_{\pm 0.02}$	$0.94_{\pm 0.03}$	$0.96_{\pm 0.01}$	$1.00_{\pm 0.00}$	$0.62_{\pm 0.30}$	$0.48_{\pm 0.24}$	$0.19_{\pm 0.12}$
10^{-1}	$0.83_{\pm 0.17}$	$0.83_{\pm 0.17}$	$0.85_{\pm 0.16}$	$0.85_{\pm 0.16}$	$1.00_{\pm 0.00}$	$0.62_{\pm 0.27}$	$0.38_{\pm 0.21}$
10^{0}	$0.80_{\pm 0.12}$	$0.81_{\pm 0.13}$	$0.80_{\pm 0.11}$	$0.81_{\pm 0.08}$	$0.69_{\pm 0.24}$	$1.00_{\pm 0.00}$	$0.56_{\pm 0.27}$
10^{1}	$0.63_{\pm 0.10}$	$0.63_{\pm 0.12}$	$0.63_{\pm 0.13}$	$0.67_{\pm 0.09}$	$0.58_{\pm 0.10}$	$0.72_{\pm 0.13}$	$1.00_{\pm 0.00}$

Adversarial transferability between RN-32's learned with different weights of the complexity loss λ.

Figure 12. (Top) The complexity of transformation complexity and the gap between the training loss and the testing loss of the learned minimum complexity DNNs, with the standard deviation in shaded colors. (Bottom) Adversarial transferability between DNNs learned with different weights of the complexity loss λ, with standard deviations.

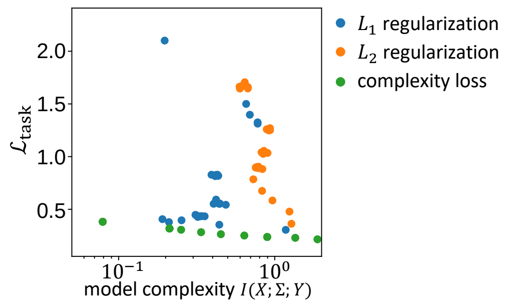


Figure 13. Comparisons of the classification loss between different baselines. The complexity loss was superior to them in terms of maintaining the classification accuracy and decreasing the model complexity.

G.6. Comparisons with traditional L_1 and L_2 regularization methods

In this section, we compare the proposed complexity loss with traditional L_1 and L_2 regularization methods. Specifically, we trained 20 residual MLPs with $\mathcal{L} = \mathcal{L}_{\text{task}} + \lambda_{L_1} \sum_l \|W_l\|_1$ and 20 residual MLPs with $\mathcal{L} = \mathcal{L}_{\text{task}} + \lambda_{L_2} \sum_l \|W_l\|_2^2$. For the L_1 regularization, we set different values of λ_{L_1} ranging from 10^{-5} to 10^{-1} in different experiments. For the L_2 regularization, we set different values of λ_{L_2} ranging from 10^{-5} to 10^{1} in different experiments. Figure 13 compares the classification loss $\mathcal{L}_{\text{task}}$ between different baselines. We found that the complexity loss was superior to traditional L_1 and L_2 regularization methods, in terms of maintaining the classification accuracy and decreasing the model complexity.

G.7. Detailed analysis of the adversarial robustness, adversarial transferability and knowledge consistency

In this section, we provide more detailed analysis of the adversarial robustness, adversarial transferability and knowledge consistency, in Section 4.3 of the paper.

In Section 4.3 of the paper, we have found that DNNs with low transformation complexity usually exhibited high adversarial robustness; vice versa. To this end, we defined the attacking utility in untargeted attacks as $U_{\text{untarget}}(x) = \max_{y' \neq y} h_{y'}(x + \epsilon) - h_y(x + \epsilon)$, where y is the ground-truth label of the sample x, and $h_y(x)$ is the output logit of the DNN in the y-th category given an input x. Then, we measured the L_2 norm of the minimum adversarial perturbation ϵ for each image, which has similar attacking utility of 40. We conducted the PGD attack (Madry et al., 2017), with the step size of each single-step attack as $0.5/255$. Other experimental settings remained the same as in (Wang et al., 2020).

We have also found in Section 4.3 that adversarial perturbations for complex DNNs could not be well transferred to simple DNNs. However, adversarial perturbations for simple DNNs could be transferred to complex DNNs. We conducted the PGD attack for each image using DNNs with different transformation complexities, and following the experimental settings in (Wang et al., 2020) to measure the adversarial transferability between DNNs.

Following settings in (Liang et al., 2019), we explored the knowledge consistency between DNNs with different transformation complexities. Specifically, we used the intermediate-layer feature x_A of a DNN (*Net-A*) trained with a specific value of λ, to reconstruct the intermediate-layer feature x_B of another DNN (*Net-B*) also trained with λ. *Net-A* and *Net-B* had the same architecture and were trained on the same dataset, but with different initialization parameters. x_A and x_B denote intermediate-layer features of *Net-A* and *Net-B* in the same layer, respectively.

We followed the experimental settings in (Liang et al., 2019) to diagnose the feature representation in the residual MLP trained on the CIFAR-10 dataset and ResNet-34 trained on the Tiny-ImageNet dataset. We diagnosed the output feature of the last layer (3072 dimensional) in the residual MLP and the output feature of the last convolutional layer ($7 \times 7 \times 512$ dimensional) in ResNet-34. We disentangled 0-order, 1-order, and 2-order consistent features $x^{(0)}$, $x^{(1)}$, and $x^{(2)}$ from x_A. For the fair comparison between DNNs learned with different λ values, we computed the strength of the k-order consistent feature as $Var(x^{(k)})/Var(x_A)$. $Var(x_A) \triangleq \mathbb{E}_{I,i}[(x_{A,I,i} - \mathbb{E}_{I',i'}[x_{A,I',i'}])^2]$, where $x_{A,I,i}$ denotes the i-th element of x_A given the image I.

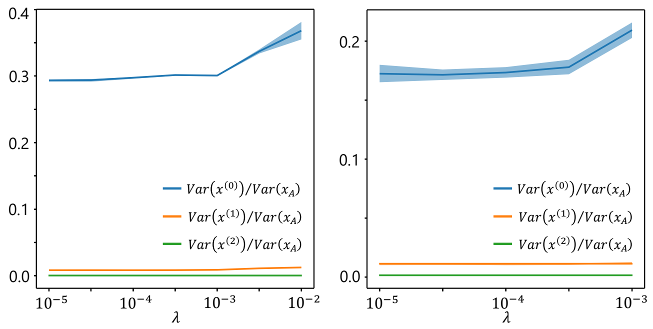


Figure 14. The strength of consistent features with different orders in (left) residual MLP trained on the CIFAR-10 dataset, and (right) ResNet-34 trained on the Tiny ImageNet dataset.

Figure 14 shows the strength of consistent features with different orders. We found that pairs of simple DNNs usually encoded similar knowledge representations (exhibiting high knowledge consistency), while complex DNNs are more likely to encode diverse knowledge. This demonstrated the reliability of features learned by simple DNNs.

G.8. Computational cost of the proposed method

This section reports the computational cost of training DNNs with the complexity loss. We compare the time cost of training for an epoch with and without the complexity loss. The time cost was measured using PyTorch 1.6 (Paszke et al., 2019) on Ubuntu 18.04, with the Intel(R) Core(TM) i9-10900X CPU @ 3.70GHz and one NVIDIA(R) TITAN RTX(TM) GPU.

Table 3. Time cost of training DNNs for an epoch with and without the complexity loss.

model	dataset	batch size	time w/o $L_{\text{complexity}}$	time w/ $L_{\text{complexity}}$
residual MLP	CIFAR-10	128	21 s/epoch	85 s/epoch
ResNet-20	CIFAR-10	128	64 s/epoch	218 s/epoch
ResNet-32	CIFAR-10	128	101 s/epoch	296 s/epoch
ResNet 18	Tiny-ImageNet	128	41 s/epoch	83 s/epoch
ResNet-34	Tiny-ImageNet	128	68 s/epoch	105 s/epoch