Decoupled Networks

Weiyang Liu[1*], Zhen Liu[1*], Zhiding Yu[2], Bo Dai[1], Rongmei Lin[3], Yisen Wang[1,4], James M. Rehg[1], Le Song[1,5]
[1]Georgia Institute of Technology [2]NVIDIA [3]Emory University [4]Tsinghua University [5]Ant Financial

Abstract

Inner product-based convolution has been a central component of convolutional neural networks (CNNs) and the key to learning visual representations. Inspired by the observation that CNN-learned features are naturally decoupled with the norm of features corresponding to the intra-class variation and the angle corresponding to the semantic difference, we propose a generic decoupled learning framework which models the intra-class variation and semantic difference independently. Specifically, we first reparametrize the inner product to a decoupled form and then generalize it to the decoupled convolution operator which serves as the building block of our decoupled networks. We present several effective instances of the decoupled convolution operator. Each decoupled operator is well motivated and has an intuitive geometric interpretation. Based on these decoupled operators, we further propose to directly learn the operator from data. Extensive experiments show that such decoupled reparameterization renders significant performance gain with easier convergence and stronger robustness.

1. Introduction

Convolutional neural networks have pushed the boundaries on a wide variety of vision tasks, including object recognition [24, 25, 5], object detection [2, 23, 22], semantic segmentation [16], etc. A significant portion of recent studies on CNNs focused on increasing network depth and representation ability via improved architectures such as shortcut connections [5, 8] and multi-branch convolution [25, 30]. Despite these advances, understanding how convolution naturally leads to discriminative representation and good generalization remains an interesting problem.

Current CNNs often encode the similarity between a patch $\boldsymbol{x}$ and a kernel $\boldsymbol{w}$ via inner product. The formulation of inner product $\langle \boldsymbol{w}, \boldsymbol{x} \rangle = \boldsymbol{w}^\top \boldsymbol{x}$ couples the semantic difference (*i.e.*, inter-class variation) and the intra-class variation in one unified measure. As a result, when the inner product between two samples is large, one can not tell whether the two samples have large semantic/label difference or have large intra-class variation. In order to better study the properties of CNN representation and further improve existing frameworks, we propose to explicitly decouple semantic difference and intra-class variation[1]. Specifically, we reparametrize the inner product with the norms and the angle, *i.e.*, $\|\boldsymbol{w}\|_2\|\boldsymbol{x}\|_2\cos(\theta_{(\boldsymbol{w},\boldsymbol{x})})$. Our direct intuition comes from the the observation in Fig. 1 where angle accounts for semantic/label difference and feature norm accounts for intra-class variation. The larger the feature norm, the more confident the prediction. Such naturally decoupled phenomenon inspires us to propose the decoupled convolution operators. We hope that decoupling norm and angle in inner product can better model the intra-class variation and the semantic difference in deep networks.

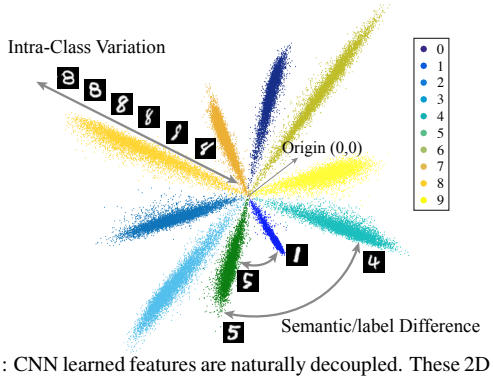


Figure 1: CNN learned features are naturally decoupled. These 2D features are output directly from the CNN by setting the feature dimension as 2.

On top of the idea to decouple the norm and the angle in an inner product, we propose a novel decoupled network (DCNet) by generalizing traditional inner product-based convolution operators ($\|\boldsymbol{w}\|\|\boldsymbol{x}\|\cos(\theta_{(\boldsymbol{w},\boldsymbol{x})})$) to decoupled operators. To this end, we define such operator as multiplication of a function of norms $h(\|\boldsymbol{w}\|, \|\boldsymbol{x}\|)$ and a function of angle $g(\theta_{(\boldsymbol{w},\boldsymbol{x})})$. The decoupled operator provides a generic framework to better model the intra-class variation and the semantic difference, and the original CNNs are equivalent to setting $h(\|\boldsymbol{w}\|, \|\boldsymbol{x}\|)$ as $\|\boldsymbol{w}\|\|\boldsymbol{x}\|$

*Equal contributions. Email:{wyliu,liuzhen1994}@gatech.edu

[1]Although the concepts of semantic difference and intra-class variation often refer to classification, they are extended to convolutions in this paper. Specifically, semantic difference means the pattern similarity between local patch $\boldsymbol{x}$ and kernel $\boldsymbol{w}$, while intra-class variation refers to the energy of local patch $\boldsymbol{x}$ and kernel $\boldsymbol{w}$.

and $g(\theta_{(\boldsymbol{w},\boldsymbol{x})})$ as $\cos(\theta_{(\boldsymbol{w},\boldsymbol{x})})$. The magnitude function $h(\|\boldsymbol{w}\|, \|\boldsymbol{x}\|)$ models the intra-class variation while the angular function $g(\theta_{(\boldsymbol{w},\boldsymbol{x})})$ models the semantic difference.

From the decoupling point of view, the original CNNs make a strong assumption that the intra-class variation can be linearly modeled via the multiplication of norms and the semantic difference is described by the cosine of the angle. However, this modeling approach is not necessarily optimal for all tasks. With the decoupled learning framework, we can either design the decoupled operators based on the task itself or learn them directly from data. The advantages of DCNets are in four aspects. First, DCNets not only allow us to use some alternative functions to better model the intra-class variation and the semantic difference, but they also enable us to directly learn these functions rather than fixing them. Second, with bounded magnitude functions, DCNets can improve the problem conditioning as analyzed in [14], and therefore DCNets can converge faster while achieving comparable or even better accuracy than the original CNNs. Third, some instances of DCNets can have stronger robustness against adversarial attacks. We can squeeze the feature space of each class with a bounded $h(\cdot)$, which can bring certain robustness. Last, the decoupled operators are very flexible and architecture-agnostic. They could be easily adapted to any kind of architectures such as VGG [24], GoogleNet [25] and ResNet [5].

Specifically, we propose two different types of decoupled convolution operators: *bounded operators* and *unbounded operators*. We present multiple instances for each type of decoupled operators. Empirically, the bounded operators may yield faster convergence and better robustness against adversarial attacks, and the unbounded operators may have better representational power. These decoupled operators can also be either smooth or non-smooth, which can yield different behaviors. Moreover, we introduce a novel concept - *operator radius* for the decoupled operators. The operator radius describes the critical change of the derivative of the magnitude function $h(\cdot)$ with respect to the input $\|\boldsymbol{x}\|$. By jointly learning the operator radius via back-propagation, we further propose *learnable decoupled operators*. Moreover, we show some alternative ways to optimize these operators that improve upon standard back-propagation. Our contributions can be summarized as:

- Inspired by the observation that CNN-learned features are naturally decoupled, we propose an explicitly decoupled framework to study neural networks.
- We show that CNNs make a strong assumption to model the intra-class and inter-class variation, which may not be optimal. By decoupling the inner product, we are able to design more effective magnitude and angular functions rather than the original convolution for different tasks.
- In comparison to standard CNNs, DCNets have easier convergence, better accuracy and stronger robustness.

2. Related Works

There are an increasing number of works [28, 21, 12, 13, 15, 29, 31, 10] that focus on improving the classification layer in order to increase the discriminativeness of learned features. [13] models the angular function for each class differently and defines a more difficult task than classification, improving the network generalization. Built upon [13], [12] further normalizes the weights of the last fully connected layer (*i.e.*, classification layer) and reported improved results on face recognition. [28, 21, 29] normalize the input features before entering the last fully connected layer, achieving promising performance on face recognition. However, these existing works can be viewed as heuristic modifications and are often restricted to the last fully connected layer. In contrast, the decoupled learning provides a more general and systematic way to study the CNNs. In our framework, the previous work can be viewed as proposing a new magnitude function $h(\|\boldsymbol{w}\|, \|\boldsymbol{x}\|)$ or angular function $g(\theta_{(\boldsymbol{w},\boldsymbol{x})})$ for the last fully connected layer. For example, normalizing the weights is to let $h(\|\boldsymbol{w}\|, \|\boldsymbol{x}\|)$ be $\|\boldsymbol{x}\|$ and normalizing the input is equivalent to $h(\|\boldsymbol{w}\|, \|\boldsymbol{x}\|) = \|\boldsymbol{w}\|$.

[14] proposes a deep hyperspherical learning framework which directly makes $h(\|\boldsymbol{w}\|, \|\boldsymbol{x}\|)$ equal to 1 such that all the activation outputs only depend on $g(\theta_{(\boldsymbol{w},\boldsymbol{x})})$. The framework provides faster convergence compared to the original CNNs, but is somehow restricted in the sense that $h(\|\boldsymbol{w}\|, \|\boldsymbol{x}\|)$ is only allowed to be 1, and therefore can be sub-optimal in some cases. From the decoupling perspective, hyperspherical learning only cares about the semantic difference and aims to compress the intra-class variation to a space that is as small as possible, while the decoupled framework focuses on both. As a non-trivial generalization of [14], our decoupled network is a more generic and unified framework to model both intra-class variation and semantic difference, providing the flexibility to design or learn both magnitude function $h(\cdot)$ and angular function $g(\cdot)$.

3. Decoupled Networks

3.1. Reparametrizing Convolution via Decoupling

For a conventional convolution operator $f(\cdot, \cdot)$, the output is calculated by the inner product of the input patch $\boldsymbol{x}$ and the filter $\boldsymbol{w}$ (both $\boldsymbol{x}$ and $\boldsymbol{w}$ are vectorized into columns):

$$f(\boldsymbol{w}, \boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle = \boldsymbol{w}^\top \boldsymbol{x}, \tag{1}$$

which can be further formulated as a decoupled form that separates the norm and the angle:

$$f(\boldsymbol{w}, \boldsymbol{x}) = \|\boldsymbol{w}\| \, \|\boldsymbol{x}\| \, cos(\theta_{(\boldsymbol{w},\boldsymbol{x})}), \tag{2}$$

where $\theta_{(\boldsymbol{w},\boldsymbol{x})}$ is the angle between $\boldsymbol{x}$ and $\boldsymbol{w}$. Our proposed decoupled convolution operator takes the general form of

$$f_d(\boldsymbol{w}, \boldsymbol{x}) = h(\|\boldsymbol{w}\|, \|\boldsymbol{x}\|) \cdot g(\theta_{(\boldsymbol{w},\boldsymbol{x})}), \tag{3}$$

which explicitly decouples the norm of $\boldsymbol{w}, \boldsymbol{x}$ and the angle between them. We define $h(\|\boldsymbol{w}\|, \|\boldsymbol{x}\|)$ as the magnitude function and $g(\theta_{(\boldsymbol{w},\boldsymbol{x})})$ as the angular activation function. It is easy to see that the decoupled convolution operator includes the original convolution operator as a special case. As illustrated in Fig. 1, the semantic difference and intra-class variation are usually decoupled and very suitable for this formulation. Based on the decoupled operator, we propose several alternative ways to model the semantic difference and intra-class variation.

3.2. Decoupled Convolution Operators

We discuss how to better model the intra-class variation, and then give a few instances of the decoupled operator.

3.2.1 On Better Modeling of the Intra-class Variation

Hyperspherical learning [14] has discussed the modeling of the inter-class variation (*i.e.*, the angular function). The design of angular function $g(\cdot)$ is relatively easy but restricted, because it only takes the angle as input. In contrast, the magnitude function $h(\cdot)$ takes the norm of $\boldsymbol{w}$ and the norm of $\boldsymbol{x}$ as two inputs, and therefore it is more complicated to design. $\|\boldsymbol{w}\|$ is the intrinsic property of a kernel itself, corresponding to the importance of the kernel rather than the intra-class variation of the inputs. Therefore, we tend not to include $\|\boldsymbol{w}\|$ into the magnitude function $h(\cdot)$. Moreover, removing $\|\boldsymbol{w}\|$ from $h(\cdot)$ indicates that all kernels (or operators) are assigned with equal importance, which encourages the network to make decision based on as many kernels as possible and therefore may make the network generalize better. However, incorporating the kernel importance to the network learning can improve the representational power and may be useful when dealing with a large-scale dataset with numerous categories. By combining $\|\boldsymbol{w}\|$ back to $h(\cdot)$, the operators become *weighted decoupled operators*. There are multiple ways of incorporating $\|\boldsymbol{w}\|$ back to the magnitude function. We will discuss and empirically evaluate these variants later.

3.2.2 Bounded Decoupled Operators

The output of the bounded operators must be bounded by a finite constant regardless of its input and kernel, namely $|f_d(\boldsymbol{w}, \boldsymbol{x})| \leq c$ where c is a positive constant. For simplicity, we first consider the decoupled operator without the norm of the weights (*i.e.*, $\|\boldsymbol{w}\|$ is not included in $h(\cdot)$).

Hyperspherical Convolution. If we let $h(\|\boldsymbol{w}\|, \|\boldsymbol{x}\|) = \alpha$, we will have the hyperspherical convolution (SphereConv) with the following decoupled form:

$$f_d(\boldsymbol{w}, \boldsymbol{x}) = \alpha \cdot g(\theta_{(\boldsymbol{w},\boldsymbol{x})}), \tag{4}$$

where $\alpha > 0$ controls the output scale. $g(\theta_{(\boldsymbol{w},\boldsymbol{x})})$ depends on the geodesic distance on the unit hypersphere and typically outputs value from -1 to 1, so the final output is in $[-\alpha, \alpha]$. Usually, we can use $\alpha = 1$, which reduces to SphereConv [14] in this case. Geometrically, SphereConv can be viewed as projecting $\boldsymbol{w}$ and $\boldsymbol{x}$ to a hypersphere and then performing inner product (if $g(\theta) = \cos(\theta)$). Based on [14], SphereConv improves the problem conditioning in neural networks, making the network converge better.

Hyperball Convolution. The hyperball convolution (BallConv) uses $h(\|\boldsymbol{w}\|, \|\boldsymbol{x}\|) = \alpha \min(\|\boldsymbol{x}\|, \rho)/\rho$ as its magnitude function. The specific form of the BallConv is

$$f_d(\boldsymbol{w}, \boldsymbol{x}) = \alpha \cdot \frac{\min(\|\boldsymbol{x}\|, \rho)}{\rho} \cdot g(\theta_{(\boldsymbol{w},\boldsymbol{x})}), \tag{5}$$

where ρ controls the saturation threshold for the input norm $\|\boldsymbol{x}\|$ and α scales the output range. When $\|\boldsymbol{x}\|$ is larger than ρ, then the magnitude function will be saturate and output α. When $\|\boldsymbol{x}\|$ is smaller than ρ, the magnitude function grows linearly with $\|\boldsymbol{x}\|$. Geometrically, BallConv can be viewed as projecting $\boldsymbol{w}$ to a hypersphere and projecting the input $\boldsymbol{x}$ to a hyperball, and then performing the inner product (if $g(\theta) = \cos(\theta)$). Intuitively, BallConv is more robust and flexible than SphereConv in the sense that SphereConv may amplify the $\boldsymbol{x}$ with very small $\|\boldsymbol{x}\|$, because $\boldsymbol{x}$ with small $\|\boldsymbol{x}\|$ and the same direction as $\boldsymbol{w}$ could still produce the maximum output. It makes SphereConv sensitive to perturbations to $\boldsymbol{x}$ with small norm. In contrast, BallConv will not have such a problem, because the multiplicative factor $\|\boldsymbol{x}\|$ can help to decrease the output if $\|\boldsymbol{x}\|$ is small. Moreover, small $\|\boldsymbol{x}\|$ indicates that the local patch is not informative and should not be emphasized. In this sense, BallConv is better than SphereConv. In terms of convergence, the BallConv can still help the network convergence because its output is bounded with the same range as SphereConv.

Hyperbolic Tangent Convolution. We present a smooth decoupled operator with bounded output called hyperbolic tangent convolution (TanhConv). The TanhConv uses a hyperbolic tangent function to replace the step function in the BallConv and can be formulated as

$$f_d(\boldsymbol{w}, \boldsymbol{x}) = \alpha \tanh\left(\frac{\|\boldsymbol{x}\|}{\rho}\right) \cdot g(\theta_{(\boldsymbol{w},\boldsymbol{x})}), \tag{6}$$

where $\tanh(\cdot)$ denotes the hyperbolic tangent function and ρ is parameter controlling the decay curve. The TanhConv can be viewed as a smooth version of BallConv, which not only shares the same advantages as BallConv but also has more convergence gain due to its smoothness [1].

3.2.3 Unbounded Decoupled Operators

Linear Convolution. One of the simplest unbounded decoupled operators is the linear convolution (LinearConv):

$$f_d(\boldsymbol{w}, \boldsymbol{x}) = \alpha \|\boldsymbol{x}\| \cdot g(\theta_{(\boldsymbol{w},\boldsymbol{x})}), \tag{7}$$

where α controls the output scale. LinearConv differs the original convolution in the sense that it projects the weights to a hypersphere and has a parameter to control the slope.

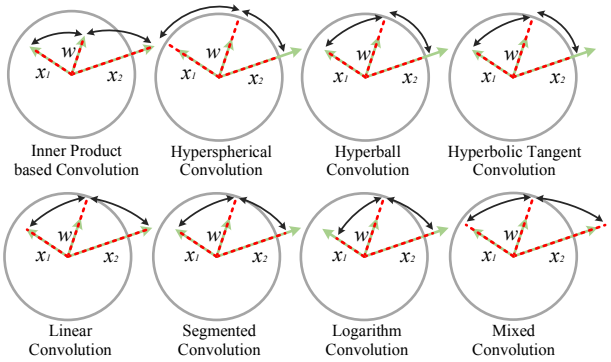


Figure 2: Geometric interpretations for decoupled convolution operators. Green denotes the original vectors, and red denotes the projected vectors.

Segmented Convolution. We propose a segmented convolution (SegConv) which takes the following form:

$$f_d(\boldsymbol{w},\boldsymbol{x}) = \begin{cases} \alpha\|\boldsymbol{x}\| \cdot g(\theta_{(\boldsymbol{w},\boldsymbol{x})}), & 0 \le \|\boldsymbol{x}\| \le \rho \\ (\beta\|\boldsymbol{x}\| + \alpha\rho - \beta\rho) \cdot g(\theta_{(\boldsymbol{w},\boldsymbol{x})}), & \rho < \|\boldsymbol{x}\| \end{cases}, \quad (8)$$

where α controls the slope when $\|\boldsymbol{x}\| \le \rho$ and β controls the slope when $\|\boldsymbol{x}\| > \rho$. ρ is the change point of the gradient of the magnitude function w.r.t. $\|\boldsymbol{x}\|$. SegConv is a flexible multi-range linear function corresponding to $\|\boldsymbol{x}\|$. Both LinearConv and BallConv are special cases of SegConv.

Logarithm Convolution. We present another smooth decoupled operator with unbounded output, logarithm convolution (LogConv). LogConv uses a Logarithm function for the norm of the input $\|\boldsymbol{x}\|$ and can be formulated as

$$f_d(\boldsymbol{w},\boldsymbol{x}) = \alpha \log(1 + \|\boldsymbol{x}\|) \cdot g(\theta_{(\boldsymbol{w},\boldsymbol{x})}), \quad (9)$$

where α controls the base of logarithm and is used to adjust the curvature of the logarithm function.

Mixed Convolution. Mixed convolution (MixConv) combines multiple decoupled convolution operators and enjoys better flexibility. Because the mixed convolution has many possible combinations, we only consider the additive combination of LinearConv and LogConv as an example:

$$f_d(\boldsymbol{w},\boldsymbol{x}) = \big(\alpha\|\boldsymbol{x}\| + \beta\log(1 + \|\boldsymbol{x}\|)\big) \cdot g(\theta_{(\boldsymbol{w},\boldsymbol{x})}), \quad (10)$$

which combines LogConv and LinearConv, becoming more flexible than both original operators.

3.2.4 Properties of Decoupled Operators

Operator Radius. Operator radius is defined to describe the gradient change point of the magnitude function. Operator radius differentiates two stages of the magnitude function. The two stages usually have different gradient ranges and therefore behave differently during optimization. We let ρ denote the operator radius in each decoupled operator. For BallConv, when $\|\boldsymbol{x}\|$ is smaller than ρ, the magnitude function will be activated and it will grow with $\|\boldsymbol{x}\|$ linearly. When $\|\boldsymbol{x}\|$ is larger than ρ, then the magnitude function will be deactivated and output a constant. For SegConv, $\|\boldsymbol{x}\| = \rho$ is the change point of the magnitude function's slope. The operator radius of some decoupled operators (SphereConv, LinearConv, LogConv) is defined to be zero, indicating that they have no operator radius. The decoupled operator with non-zero operator radius is similar to a gated operator where $\|\boldsymbol{x}\| = \rho$ serves as the switch.

Boundedness. The Boundedness of a decoupled operator may affect its convergence speed and robustness. [14] shows that using a bounded operator can improve the convergence due to two reasons. First, bounded operators lead to better problem conditioning in training a deep network via stochastic gradient descent. Second, bounded operators make the variance of the output small and partially address the internal covariate shift problem. The bounded operators can also constrain the Lipschitz constant of a neural network, making the entire network more smooth. The Lipschitz constant of a neural network is shown to be closely related to its robustness against adversarial perturbation [7]. In contrast, the unbounded operators may have stronger approximation power and flexibility than the bounded ones.

Smoothness. The smoothness of the magnitude function is closely related to the approximation ability and the convergence behavior. In general, using a smooth magnitude function could have better approximation rate [17] and may also lead to more stable and faster convergence [1]. However, a smooth magnitude function may also be more computationally expensive, since it could be more difficult to approximate a smooth function with polynomials.

3.3. Geometric Interpretations

All the decoupled convolution operators have very clear geometric interpretations, as illustrated in Fig. 2. Because all decoupled operators normalize the kernel weights, all the weights are already on the unit hypersphere. SphereConv also projects the input vector $\boldsymbol{x}$ on the unit hypersphere and then computes the similarity between $\boldsymbol{w}$ and $\boldsymbol{x}$ based on the geodesic distance on the hypersphere (multiplied by a scaling factor α). Therefore, its output is bounded from $-\alpha$ to α and only depends on the directions of $\boldsymbol{w}$ and $\boldsymbol{x}$ (suppose $g(\theta_{(\boldsymbol{w},\boldsymbol{x})})$ is in the range of $[-1, 1]$).

BallConv first projects the input vector $\boldsymbol{x}$ to a hyperball and then computes the similarity based on the projected $\boldsymbol{x}$ inside the hyperball and the normalized $\boldsymbol{w}$ on surface of the hyperball. Specifically, BallConv projects $\boldsymbol{x}$ to the hypersphere if $\|\boldsymbol{x}\| > \rho$. TanhConv is a smoothed BallConv and has similar geometric interpretation, but TanhConv is differentible everywhere and has soft boundary around the operator radius $\|\boldsymbol{x}\| = \rho$. TanhConv can be viewed as performing projection to a soft hyperball.

SegConv is more flexible than both SphereConv and BallConv. By using certain parameters, SegConv can reduce to either SphereConv or BallConv. SegConv essentially adjusts the norm of the input $\boldsymbol{x}$ with a multi-range lin-

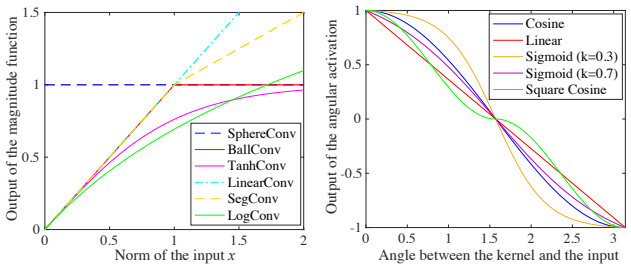


Figure 3: Magnitude function ($\rho=1$) and angular activation function.

ear multiplicative factor. Geometrically, such a factor will either push the vector close to the hypersphere or away from the hypersphere depending on the selection of α and β. For example, we consider the case where $\alpha=1$ and $0<\beta<1$. When $\|\boldsymbol{x}\|\leq\rho$, the magnitude function $h(\cdot)$ in SegConv will directly output $\|\boldsymbol{x}\|$. When $\|\boldsymbol{x}\|>\rho$, $h(\cdot)$ in SegConv will output a value smaller than $\|\boldsymbol{x}\|$, as shown in Fig. 2.

LinearConv is the simplest unbounded operator and its magnitude function grows linearly with $\|\boldsymbol{x}\|$. When $\alpha=1$, the magnitude function $h(\cdot)$ in LinearConv simply outputs $\|\boldsymbol{x}\|$, which does not perform any projection.

LogConv use a logarithm function to transform the norm of the input $\boldsymbol{x}$. After such nonlinear transformation on $\boldsymbol{x}$, LogConv computes similarity based on the transformed input $\boldsymbol{x}$ and the normalized weights on a hypersphere.

3.4. Design of the Angular Activation Function

The design of the angular function $g(\theta_{(\boldsymbol{w},\boldsymbol{x})})$ mostly follows the deep hyperspherical learning [14]. We use four different types of $g(\theta_{(\boldsymbol{w},\boldsymbol{x})})$ in this paper. The linear angular activation is defined as

$$g(\theta_{(\boldsymbol{w},\boldsymbol{x})}) = -\frac{2}{\pi}\theta_{(\boldsymbol{w},\boldsymbol{x})} + 1, \tag{11}$$

whose output grows linearly with the angle $\theta_{(\boldsymbol{w},\boldsymbol{x})}$. The cosine angular activation is defined as

$$g(\theta_{(\boldsymbol{w},\boldsymbol{x})}) = \cos(\theta_{(\boldsymbol{w},\boldsymbol{x})}), \tag{12}$$

which is also used by the original convolution operator. Moreover, the sigmoid angular activation is defined as

$$g(\theta_{(\boldsymbol{w},\boldsymbol{x})}) = \frac{1+\exp(-\frac{\pi}{2k})}{1-\exp(-\frac{\pi}{2k})}\cdot\frac{1-\exp(\frac{\theta_{(\boldsymbol{w},\boldsymbol{x})}}{k}-\frac{\pi}{2k})}{1+\exp(\frac{\theta_{(\boldsymbol{w},\boldsymbol{x})}}{k}-\frac{\pi}{2k})}, \tag{13}$$

where k controls the curvature. Additionally, we also propose a square cosine angular activation function:

$$g(\theta_{(\boldsymbol{w},\boldsymbol{x})}) = \text{sign}(\cos(\theta))\cdot\cos^2(\theta), \tag{14}$$

which can encourage a degree of angular margin near the decision boundary and may improve network generalization. In addition to fixing these angular activations prior to training, we can also jointly learn the parameter k in the sigmoid activation using back-propagation, which is a learnable angular activation [14]. Fig. 3 shows the curves of these angular activation functions.

3.5. Weighted Decoupled Operators

All the decoupled operators we have discussed normalize the kernel weights $\boldsymbol{w}$ and the magnitude functions do not take the weights into consideration. Although empirically we find that the standard decoupled operators work better than the weighted ones in most cases, we still consider weighted decoupled operators, which incorporate $\|\boldsymbol{w}\|$ into the magnitude function, in order to improve the operator's flexibility. We propose two straightforward ways to combine $\|\boldsymbol{w}\|$: linear and nonlinear.

Linearly Weighted Decoupled Operator. Similar to the original inner produce-based convolution, we can directly multiply the norm of weights into the magnitude function, which makes the decoupled operators linearly weighted. For example, SphereConv will become $f_d(\boldsymbol{w},\boldsymbol{x}) = \alpha\|\boldsymbol{w}\|\cdot g(\theta_{(\boldsymbol{w},\boldsymbol{x})})$. Notably, linearly weighted LinearConv will become the original inner produce-based convolution.

Nonlinearly Weighted Decoupled Operator. Compared to linearly weighted decoupled operators, the norm of the weights are incorporated into the magnitude function in a nonlinear way. Taking TanhConv as an example, we could formulate the nonlinearly weighted TanhConv as

$$f_d(\boldsymbol{w},\boldsymbol{x}) = \alpha\tanh(\frac{1}{\rho}\|\boldsymbol{x}\|\cdot\|\boldsymbol{w}\|)\cdot g(\theta_{(\boldsymbol{w},\boldsymbol{x})}). \tag{15}$$

We can also formulate the nonlinearly weighted TanhConv in an alternative way:

$$f_d(\boldsymbol{w},\boldsymbol{x}) = \alpha\tanh(\frac{1}{\rho}\|\boldsymbol{w}\|)\cdot\tanh(\frac{1}{\rho}\|\boldsymbol{x}\|)\cdot g(\theta_{(\boldsymbol{w},\boldsymbol{x})}). \tag{16}$$

The first nonlinearly weighted formulation couples $\|\boldsymbol{x}\|$ and $\|\boldsymbol{w}\|$ by multiplication and then perform a nonlinear transformation, while the second one performs nonlinear transformations separately for $\|\boldsymbol{x}\|$ and $\|\boldsymbol{w}\|$, and then multiplies them. In practice, the linearly weighted operators are favored over nonlinearly weighted ones due to the simplicity.

3.6. Learnable Decoupled Operators

Because our decoupled operators usually have hyperparameters, we usually need to do cross-validation in order to choose suitable parameters, which is time-consuming and sub-optimal. To address this, we can learn these parameters jointly with network weight training via back-propagation. We propose learnable decoupled operators which perform hyperparameter learning with $h(\cdot)$ and $g(\theta_{(\boldsymbol{w},\boldsymbol{x})})$. For example, [14] proposed to learn the hyperparameters of sigmoid angular function. By making both $h(\|\boldsymbol{w}\|,\|\boldsymbol{x}\|)$ and $g(\theta_{(\boldsymbol{w},\boldsymbol{x})})$ learnable, we can greatly enhance the representational power and flexibility.

However, making the decoupled operators too flexible (*i.e.*, too many learnable parameters) may require a prohibitive amount of training data to achieve good generalization. In order to achieve an effective trade-off, we

only investigate learning the operator radius ρ via back-propagation during the network training.

4. Improving the Optimization for DCNets

We propose several tricks to improve the optimization of DCNets and enable DCNets to converge to a better local minima. More analysis and discussion of weight projection and weight gradients are provided in Appendix G.

4.1. Weight Projection

The forward pass of DCNets is not dependent on the norm of the weights $\|\boldsymbol{w}\|$, because the decoupled operators take the normalized weights as input. However, $\|\boldsymbol{w}\|$ will significantly affect the backward pass. Taking SphereConv as an example, we compute the gradient w.r.t. $\boldsymbol{w}$:

$$\frac{\partial}{\partial \boldsymbol{w}}\left(\hat{\boldsymbol{w}}^\top \hat{\boldsymbol{x}}\right) = \frac{\hat{\boldsymbol{x}} - \hat{\boldsymbol{w}}^\top \hat{\boldsymbol{x}} \cdot \hat{\boldsymbol{w}}}{\|\boldsymbol{w}\|}, \tag{17}$$

where $\hat{\boldsymbol{w}} = \boldsymbol{w}/\|\boldsymbol{w}\|$ and $\hat{\boldsymbol{x}} = \boldsymbol{x}/\|\boldsymbol{x}\|$. In comparison, $\|\boldsymbol{w}\|$ will not affect the gradient w.r.t $\boldsymbol{w}$ in inner product. From Eq. (17), large $\|\boldsymbol{w}\|$ can make the gradients very small so that the backward pass is not able to update the weights effectively. To address this issue, we propose *weight projection* to control the norm of the weights. Weight projection performs $\boldsymbol{w} \leftarrow s \cdot \hat{\boldsymbol{w}}$ every certain number of iterations where $\leftarrow$ denotes the replacement operation. s is a positive constant which controls the norm of the gradient (we use $s=1$ in our experiments). In general, larger s leads to smaller gradients. Note that, weight projection cannot be used in the weighted decoupled operators, because $\|\boldsymbol{w}\|$ will affect the forward pass. We can only apply weight projection to our standard decoupled operators.

4.2. Weighted Gradients

From Eq. (17), we observe that we could simply multiply $\|\boldsymbol{w}\|$ to Eq. (17) to eliminate the effect of $\|\boldsymbol{w}\|$ on the backward pass. We update the weights with the following:

$$\Delta \boldsymbol{w} = \|\boldsymbol{w}\| \cdot \frac{\partial}{\partial \boldsymbol{w}}\left(\hat{\boldsymbol{w}}^\top \hat{\boldsymbol{x}}\right) = \hat{\boldsymbol{x}} - \hat{\boldsymbol{w}}^\top \hat{\boldsymbol{x}} \cdot \hat{\boldsymbol{w}}, \tag{18}$$

which does not depend on $\|\boldsymbol{w}\|$ and is called *weighted gradients*. Using the proposed weighted gradients for back-propagation, we can also prevent the gradients from being affected by the norm of the weights.

4.3. Pretraining as a Better Initialization

We find that DCNets may sometimes be trapped into a bad local minima and yield a less competitive accuracy while trained on large-scale datasets (*e.g.*, ImageNet). Because the decoupled operators have stronger nonlinearity, its loss landscape may be more complex than the original convolution. The most straightforward way to improve the optimization is to use a better initialization. To this end, we use a CNN model that has the same structure and is pre-trained on the same training set to initialize the DCNet.

5. Discussions

Why Decoupling? Decoupling the intra-class and inter-class variation gives us the flexibility to design better models that are more suitable for a given task. Inner product-based convolution is computationally attractive but not necessarily optimal. The original convolution makes an assumption that the intra-class and inter-class variations are modeled by $h(\|\boldsymbol{w}\|, \|\boldsymbol{x}\|) = \|\boldsymbol{w}\|\|\boldsymbol{x}\|$ and $g(\theta) = \cos(\theta)$, respectively. Such assumptions may not be optimal. $h(\cdot)$ and $g(\cdot)$ can be task-driven in our novel decoupled framework.

Flexibility of Decoupled Operators. There are numerous design options for the magnitude and angular function. The original convolution can be viewed as a special decoupled operator. Moreover, we can parametrize a decoupled operator with a few learnable parameters and learn them via back-propagation. However, there is a delicate tradeoff between the size of the training data, the generalization of the network and the flexibility of the decoupled operator. Generally, given a large enough dataset, the network generalization improves with more learnable parameters.

A Unified Learning Framework for CNNs. The decoupled formulation provides a unified learning framework for CNNs. Consider a standard CNN with ReLU, we write the convolution and ReLU as $\max(0, \|\boldsymbol{w}\|\|\boldsymbol{x}\|\cos(\theta))$ which can be written as $\|\boldsymbol{w}\|\|\boldsymbol{x}\| \cdot \max(0, \cos(\theta))$. Such formulation can be viewed as a decoupled operator where $h(\|\boldsymbol{w}\|, \|\boldsymbol{x}\|) = \|\boldsymbol{w}\|\|\boldsymbol{x}\|$ and $g(\theta) = \max(0, \cos(\theta))$. We can jointly consider the convolution operator and nonlinear activation in the decoupled framework. It is possible to learn one single function $g(\cdot)$ that represents both angular activation and the nonlinearity, which is why the square cosine angular activation works well without ReLU.

Network Regularization. In most instances of DCNets, the ℓ_2 weight decay is no longer suitable. [14] uses an orthonormal constraint $\|\boldsymbol{W}^\top \boldsymbol{W} - \boldsymbol{I}\|_F^2$ to regularize the network, where $\boldsymbol{W}$ is the weight matrix whose columns are the kernel weights and $\boldsymbol{I}$ is identity matrix. We also propose an orthogonal constraint $\|\boldsymbol{W}^\top \boldsymbol{W} - \mathrm{diag}(\boldsymbol{W}^\top \boldsymbol{W})\|_F^2$.

Network Architecture. Due to the non-linear nature of DCNet, the performance of specific $h(\cdot)$ and $g(\cdot)$ is dependent on the choice of architecture. An interesting challenge for future work is to inverstigate the link between the architecture and the choise of $h(\cdot)$ and $g(\cdot)$.

6. Experiments and Results

General. We evaluate both accuracy and robustness of DCNets on objection recognition. For all decoupled operators, we use the standard softmax loss if not otherwise specified.

Training. The architecture for each task and the training details are given in Appendix A. For CIFAR, the network is trained by ADAM with 128 batch size. The learning rate starts from 0.001. For ImageNet, we use SGD with momentum 0.9 and batch size 40. The learning rate starts from 0.1.

For adversarial attacks, the networks are trained by ADAM. All learning rates are divided by 10 when the error plateaus.

Implementation Details. For all decoupled operators that have non-zero operator radius (*i.e.*, $\rho \neq 0$), we will learn the operator radius from the training data via back-propagation. More details are provided in Appendix B.

6.1. Object Recognition

6.1.1 CIFAR-10 and CIFAR-100

Weighted Decoupled Operators. We first compare the weighted decoupled operators and the standard ones. Because the weights are incorporated into the forward pass in the weighted decoupled operators, the optimization tricks like weight projection and weighted gradients are not applicable. Therefore, the weighted operators simply use the conventional gradients to perform back-propagation. For standard decoupled operators, we show the results using standard optimization, weight projection and weight gradients. From the results of TanhConv in Table 1, weighted decoupled operators do not show obvious advantages.

Method	Error
Linearly Weighted Decoupled Operator	22.95
Nonlinearly Weighted Decoupled Operator (Eq. (15))	23.03
Nonlinearly Weighted Decoupled Operator (Eq. (16))	23.38
Decoupled Operator (Standard Gradients)	23.09
Decoupled Operator (Weight Projection)	**21.17**
Decoupled Operator (Weighted Gradients)	21.45

Table 1: Evaluation of weighted operators (TanhConv) on CIFAR-100.

Optimization Tricks. We propose weight projection and weighted gradients to facilitate the optimization of DCNets. These two tricks essentially amplify the original gradient and make the backward update more effective. From Table 1, we observe that both weight projection and weighted gradients work much better than the competing methods.

Method	Linear	Cosine	Sq. Cosine
CNN Baseline	-	35.30	-
LinearConv	33.39	31.76	N/C
TanhConv	**32.88**	31.88	**34.26**
SegConv	34.69	**30.34**	N/C

Table 2: Testing error (%) of plain CNN-9 without BN on CIFAR-100. "N/C" indicates that the model can not converge. "-" denotes no result. The results of different columns belong to different angular activation.

Learning without Batch Normalization. Batch Normalization (BN) [9] is usually crucial for training a well-performing CNN, but the results in Table 2 show that our decoupled operators can perform much better than the original convolution even without BN.

Learning without ReLU. Our decoupled operators naturally have strong nonlinearity, because our decoupled convolution is no longer a linear matrix multiplication. In Table 3, square cosine angular activation works extremely well in plain CNN-9, even better than the networks with ReLU. The results show that using suitable $h(\cdot)$ and $g(\cdot)$ can lead to significantly better accuracy than the baseline CNN with ReLU, even if our DCNet does not use ReLU at all.

Method	Cosine w/o ReLU	Sq. Cosine w/o ReLU	Cosine w/ ReLU	Sq. Cosine w/ ReLU
Baseline	58.24	-	26.01	-
SphereConv	33.31	25.90	26.00	26.97
BallConv	**31.81**	25.43	25.18	26.48
TanhConv	32.27	25.27	25.15	26.94
LinearConv	36.49	24.36	**24.81**	25.14
SegConv	33.57	**24.29**	24.96	**25.04**
LogConv	33.62	24.91	25.17	25.85
MixConv	33.46	24.93	25.27	25.77

Table 3: Testing error rate (%) of plain CNN-9 on CIFAR-100. Note that, BN is used in all compared models. Baseline is the original plain CNN-9.

Comparison among Different Decoupled Operators. We compare different decoupled operators on both plain CNN-9 and ResNet-32. All the compared decoupled operators are unweighted and use weight projection during optimization. The standard softmax loss and BN are used in all networks. For plain CNN-9, we compare the case with and without ReLU. The results in Table 3 show that DCNets significantly outperform the baseline. In particular, our DCNet with SegConv and square cosine can achieve 24.29% even without ReLU, which is even better than the networks with ReLU. For ResNet-32, our DCNets also consistently outperform the baseline with a considerable margin. The results further verify that the intra-class and inter-class variation assumptions of the original CNN are not optimal.

Method	Linear	Cosine	Sq. Cosine
ResNet Baseline	-	26.69	-
SphereConv	21.79	21.44	24.40
BallConv	21.44	21.12	24.31
TanhConv	21.6	21.17	24.77
LinearConv	21.09	22.17	21.31
SegConv	**20.86**	**20.91**	**20.88**
LogConv	21.84	21.08	22.86
MixConv	21.02	21.28	21.81

Table 4: Testing error rate (%) of ResNet-32 on CIFAR-100.

Convergence. We also evaluate the convergence of DCNets using the architecture of ResNet-32. The convergence curves in Fig. 4 show that the decoupled operators are able to converge and generalize better than original convolution operators on CIFAR-100 dataset.

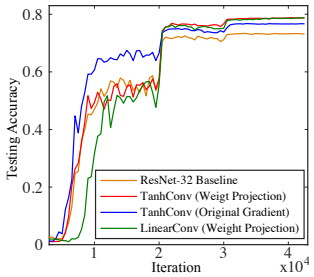


Figure 4: Convergence.

Method	CIFAR-10	CIFAR-100
ResNet-110-original [5]	6.61	25.16
ResNet-1001 [6]	4.92	**22.71**
ResNet-1001 (64 mini-batch size) [6]	**4.64**	-
DCNet-32 (TanhConv + Cosine)	**4.75**	21.12
DCNet-32 (LinearConv + Sq. Cos.)	5.34	**20.23**

Table 5: Comparison to the state-of-the-art on CIFAR-10 and CIFAR-100.

Comparison to the state-of-the-art. Table 5 shows that our DCNet-32 has very competitive accuracy compared to ResNet-1001. In order to achieve best accuracy, we use the weight-normalized softmax loss [14]. We also find that using SGD further improves the accuracy of DCNets. Experiments on SGD-trained models are provided in Appendix F.

6.1.2 ImageNet-2012

Standard ResNet. We first evaluate the DCNets with the standard ResNet-18. All presented decoupled operators use the cosine angular activation. [14] shows that SphereConv can perform comparably to the baseline on ImageNet only when the network is wide enough. Using the weight projection and pretrained model initialization, SphereConv is comparable to the baseline even on narrow networks. Most importantly, TanhConv and LinearConv achieve better accuracy than the baseline ResNet. The learned filters of DCNets on ImageNet are also provided in Appendix E.

Modified ResNet. We also evaluate decoupled operators with a modified ResNet, similar to SphereFace networks [12], to better show the advantages of decoupled operators. DCNets can be trained from scratch and outperform the baseline by 1%. Moreover, DCNets can converge stably in very challenging scenarios. From Table 6, we observe that DCNets can converge to a decent accuracy without BN, while the baseline model fails to converge without BN.

Method	Standard ResNet-18 w/ BN	Modified ResNet-18 w/ BN	Modified ResNet-18 w/o BN
Baseline	12.63	12.10	N/C
SphereConv	12.68*	11.55	13.30
LinearConv	**11.99***	11.50	N/C
TanhConv	12.47*	**11.10**	**12.79**

Table 6: Center-crop Top-5 error (%) of standard ResNet-18 and modified ResNet-18 on ImageNet-2012. * indicates we use the pretrained model of original CNN on ImageNet-2012 as initialization (see Section 4.3).

6.2. Robustness against Adversarial attacks

We evaluate the robustness of DCNets. DCNets in this subsection use standard gradients and are trained without any optimization trick. Fast gradient sign method (FGSM) [3] and basic iterative method (BIM) [11] are used to attack the networks. Experimental details and more experiments are given in Appendix A and Appendix C, respectively.

6.2.1 White-box Adversarial Attacks

We run white-box attacks on both naturally trained models and FGSM-trained models on CIFAR-10 (results shown in Table 7). "None" attacks mean that all the testing samples are normal. For naturally trained models, all DCNet variants show significantly better robustness over the baseline, with naturally trained TanhConv being most resistant. With adversarial training, while DCNets achieve the best robustness, SphereConv is particularly resistant against BIM attack. We speculate that the tight spherical constraint strongly twists the data manifold so that the adversarial gradient updates can only result in small gains.

Attack	Target models: Baseline	SphereConv	BallConv	TanhConv
None	85.35	88.58	91.13	**91.45**
FGSM	18.82	43.64	50.47	**52.60**
BIM	8.67	8.89	7.74	**10.18**
None	83.70	87.41	87.47	**87.54**
FGSM	78.96	**85.98**	82.20	81.46
BIM	7.96	**35.07**	17.38	19.86

Table 7: White-box attacks on CIFAR-10. Performance is measured in accuracy (%). The first three rows are results of naturally trained models, and the last three rows are results of adversarially trained models.

Attack	Target models: Baseline	SphereConv	BallConv	TanhConv
None	85.35	88.58	91.13	**91.45**
FGSM	50.90	**56.71**	49.50	50.61
BIM	36.22	**43.10**	27.48	29.06
None	83.70	87.41	87.47	**87.54**
FGSM	77.57	76.29	78.67	**80.38**
BIM	78.55	77.79	80.59	**82.47**

Table 8: Black-box attacks on CIFAR-10. Performance is measured in accuracy (%). The first three rows are results of naturally trained models, and the last three rows are results of adversarially trained models.

6.2.2 Black-box Adversarial Attacks

We run black-box attacks on naturally-trained and FGSM-trained models on CIFAR-10 (see Table 8). With natural training, it is surprising that BallConv and TanhConv do not show an advantage over the baseline, while SphereConv performs the best. The strongly nonlinear landscape of BallConv and TanhConv may be too difficult to be optimized without adversarial training. SphereConv, with a tighter geometric constraint, is able to withstand adversarial attacks without adversarial training. With adversarial training, SphereConv is less resistant against adversarial attacks than the baseline. BallConv and TanhConv, however, show significant advantage over the baseline. Our observation that adversarial training compromises the robustness of SphereConv matches the conclusion made by [27]. Since SphereConv enforces a tight constraint of output vectors, the landscape around some data points will be dramatically changed during adversarial training. BallConv and TanhConv are less constrained and thus can fit adversarial examples without detrimental changes in the landscapes.

7. Concluding Remarks

This paper proposes a decoupled framework for learning neural networks. The decoupled formulation enables us to design or learn better decoupled operators than the original convolution. We argue that standard CNNs do not constitute an optimal decoupled design in general.

Acknowledgements. The project was supported in part by NSF IIS-1218749, NSF Award BCS-1524565, NIH BIGDATA 1R01GM108341, NSF CAREER IIS-1350983, NSF IIS-1639792 EAGER, NSF CNS-1704701, ONR N00014-15-1-2340, Intel ISTC, NVIDIA, Amazon AWS.

References

[1] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015. 3, 4

[2] R. Girshick. Fast r-cnn. In *ICCV*, 2015. 1

[3] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014. 8, 10, 12

[4] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *CVPR*, 2015. 10

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 2, 7, 10

[6] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016. 7

[7] M. Hein and M. Andriushchenko. Formal guarantees on the robustness of a classifier against adversarial manipulation. In *NIPS*, 2017. 4

[8] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. In *CVPR*, 2017. 1

[9] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 7, 11

[10] M. Jones and H. Kobori. Improving face verification and person re-identification accuracy using hyperplane similarity. In *ICCV*, 2017. 2

[11] A. Kurakin, I. Goodfellow, and S. Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016. 8, 10

[12] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, and L. Song. Sphereface: Deep hypersphere embedding for face recognition. In *CVPR*, 2017. 2, 8, 10

[13] W. Liu, Y. Wen, Z. Yu, and M. Yang. Large-margin softmax loss for convolutional neural networks. In *ICML*, 2016. 2

[14] W. Liu, Y.-M. Zhang, X. Li, Z. Yu, B. Dai, T. Zhao, and L. Song. Deep hyperspherical learning. In *NIPS*, 2017. 2, 3, 4, 5, 6, 8, 10

[15] Y. Liu, H. Li, and X. Wang. Rethinking feature discrimination and polymerization for large-scale recognition. *arXiv preprint arXiv:1710.00870*, 2017. 2

[16] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015. 1

[17] H. N. Mhaskar and C. A. Micchelli. How to choose an activation function. In *NIPS*, 1994. 4

[18] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard. Universal adversarial perturbations. *arXiv preprint arXiv:1610.08401*, 2016. 10

[19] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *CVPR*, 2016. 10, 11

[20] N. Papernot, N. Carlini, I. Goodfellow, R. Feinman, F. Faghri, A. Matyasko, K. Hambardzumyan, Y.-L. Juang, A. Kurakin, R. Sheatsley, et al. cleverhans v2.0.0: an adversarial machine learning library. *arXiv preprint arXiv:1610.00768*, 2016. 10

[21] R. Ranjan, C. D. Castillo, and R. Chellappa. L2-constrained softmax loss for discriminative face verification. *arXiv preprint arXiv:1703.09507*, 2017. 2

[22] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, 2016. 1

[23] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, 2015. 1

[24] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1, 2

[25] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015. 1, 2

[26] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013. 10

[27] F. Tramèr, A. Kurakin, N. Papernot, D. Boneh, and P. McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017. 8, 10

[28] F. Wang, W. Liu, H. Liu, and J. Cheng. Additive margin softmax for face verification. *arXiv preprint arXiv:1801.05599*, 2018. 2

[29] F. Wang, X. Xiang, J. Cheng, and A. L. Yuille. Normface: l_2 hypersphere embedding for face verification. *arXiv preprint arXiv:1704.06369*, 2017. 2

[30] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *CVPR*, 2017. 1

[31] Y. Yuan, K. Yang, and C. Zhang. Feature incay for representation regularization. *arXiv preprint arXiv:1705.10284*, 2017. 2

Layer	Plain CNN-9	CNN-9 for adversarial attacks	ResNet-32 for CIFAR	Standard ResNet-18	Modified ResNet-18
Conv0.x	N/A	N/A	[3×3, 96]	[7×7, 64], S2 3×3, Max Pooling, S2	[7×7, 128], S2 3×3, Max Pooling, S2
Conv1.x	[3×3, 64]×3 2×2 Max Pooling, S2	[3×3, 32]×3 2×2 Max Pooling, S2	$\begin{bmatrix} 3\times3, 96 \\ 3\times3, 96 \end{bmatrix} \times 5$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$	[3×3, 128]×1, S2 $\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 1$
Conv2.x	[3×3, 128]×3 2×2 Max Pooling, S2	[3×3, 64]×3 2×2 Max Pooling, S2	$\begin{bmatrix} 3\times3, 192 \\ 3\times3, 192 \end{bmatrix} \times 5$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$	[3×3, 256]×1, S2 $\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$
Conv3.x	[3×3, 256]×3 2×2 Max Pooling, S2	[3×3, 128]×3 2×2 Max Pooling, S2	$\begin{bmatrix} 3\times3, 384 \\ 3\times3, 384 \end{bmatrix} \times 5$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$	[3×3, 512]×1, S2 $\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$
Conv4.x	N/A	N/A	N/A	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$	[3×3, 1024]×1, S2
Final	512-dim fully connected	256-dim fully connected	Average Pooling		

Table 9: Our CNN and ResNet architectures with different convolutional layers. Conv0.x, Conv1.x, Conv2.x, Conv3.x and Conv4.x denote convolution units that may contain multiple convolutional layers, and residual units are shown in double-column brackets. Conv1.x, Conv2.x and Conv3.x usually operate on different size feature maps. These networks are essentially similar to [5], but with different number of filters in each layer. The downsampling is performed by convolutions with a stride of 2. E.g., [3×3, 64]×4 denotes 4 cascaded convolution layers with 64 filters of size 3×3, and S2 denotes stride 2.

A. Experimental Details

A.1. General Settings

The network architectures used in the paper are elaborated in Table 9. Due to the limitations of our GPU resources, we mostly conduct experiments based on plain CNN-9 and ResNet-32 for CIFAR and ResNet-18 for ImageNet. For CIFAR-10 and CIFAR-100, we use ADAM for all the networks including the baseline. For ImageNet-2012, we use the SGD with momentum 0.9 for all the networks. If not specified, we use the batch normalization by default for all the experiments on object recognition. For the experiments against adversarial attacks, we use the plain CNN-9. We do not use the batch normalization for the adversarial attack experiments. All the experiments are implemented using TensorFlow library. We use the same data augmentation protocol for CIFAR-10, CIFAR-100 and ImageNet-2012 as [14]. For initialization of DCNets and baselines, we follow [4]. For modified ResNet-18 in ImageNet, we use the same initialization as [12].

Since we are already using optimization tricks on $\|\boldsymbol{w}\|$, we propose to replace the orthonormal constraint in [14] with the proposed orthogonal constraint.

A.2. Details about FGSM and BIM Attacks

Recent studies show that neural networks are prone to adversarial attacks [3, 18, 19, 26]. One of the simplest attacks is FGSM [3], which computes the adversarial image $\tilde{x}$ of some input image x such that $\|x-\tilde{x}\|_\infty \leq \epsilon$. FGSM performs one single step gradient descent (with step size ϵ) to decrease the probability of the ground truth label. Formally, $\tilde{x} = x + \epsilon \text{sign}(\nabla_x J(\theta, x, y))$ where J is the loss function used to train the network, θ represents the network parameters, x is the input image and y is the ground truth label associated with x. We compare our models and ResNet baseline on the performance on adversarial examples.

In addition, we evaluate the performance of DCNets and ResNet baselines on BIM (Basic Iterative Method) attack [11]. BIM runs certain number N of iterations of FGSM, with a smaller step size τ. In each iteration, the resulted perturbed image $\tilde{x}$ is clipped so that $\|x - \tilde{x}\|_\infty \leq \epsilon$.

We implement the experiments with Cleverhans [20]. In adversarial training using FGSM, $\epsilon = 8$ is used to generated the adversarial examples. In all the following adversarial attack experiments, we set $\epsilon = 8$, $\tau = 2$, $N = 20$. We report the accuracy on adversarial examples for both naturally trained models and adversarially trained models using FGSM. The network architecture is shown in Table 9.

A.3. Details about the Black-box Attacks

[27] shows that adversarially trained models behave significantly different on adversarial examples trained on itself and transferred adversarial examples. As suggested by [27], we report the resistance of our models against black-box attacks with ResNet baseline model. Specifically, the adversarial examples are computed from a CNN baseline with the same architecture as the target models. The generated adversarial examples are then used to attack those target models. The architecture and the attack parameters are kept the same as in the white-box experiment.

B. Training and Implementation Details

Improved Learning of Operator Radius. To facilitate the learning of the operator radius ρ, we multiply the average norm of local patch $\boldsymbol{x}$ to ρ. Taking TanhConv as an example, we implement it using the following form:

$$f_d(\boldsymbol{w}, \boldsymbol{x}) = \alpha \tanh\left(\frac{\|\boldsymbol{x}\|}{\rho \cdot \mathbb{E}\{\|\boldsymbol{x}\|\}}\right) \cdot g(\theta_{(\boldsymbol{w},\boldsymbol{x})}), \tag{19}$$

where ρ is learnable and it is initialized by a constant 1. The reason we are multiplying the average norm of $\|\boldsymbol{x}\|$ to ρ comes from our empirical observation that lots of ρ in the middle layers stay unchanged and can not be updated effectively. Compared to the original formulation, Eq. (19) essentially performs an normalization to $\|\boldsymbol{x}\|$ and make its mean become 1 (*i.e.*, $\mathbb{E}\left(\frac{\|\boldsymbol{x}\|}{\mathbb{E}\{\|\boldsymbol{x}\|\}}\right) = 1$). This is also the reason we initialize ρ with 1. The gradient of the magnitude function $h(\cdot)$ w.r.t ρ can be large enough such that ρ is updated effectively. Therefore, for all the decoupled operators that have a learnable non-zero operator radius ρ (*e.g.*, BallConv, TanhConv, SegConv, etc.), we will multiply $\mathbb{E}\{\|\boldsymbol{x}\|\}$ to ρ in order to facilitate its learning. In practice, we use the moving average to compute $\mathbb{E}\{\|\boldsymbol{x}\|\}$, similar to BN [9]. Note that, each kernel will preserve its independent patch norm mean $\mathbb{E}\{\|\boldsymbol{x}\|\}$. BallConv is implemented using

$$f_d(\boldsymbol{w}, \boldsymbol{x}) = \alpha \cdot \frac{\min(\|\boldsymbol{x}\|, \rho \cdot \mathbb{E}\{\|\boldsymbol{x}\|\})}{\rho \cdot \mathbb{E}\{\|\boldsymbol{x}\|\}} \cdot g(\theta_{(\boldsymbol{w},\boldsymbol{x})}), \tag{20}$$

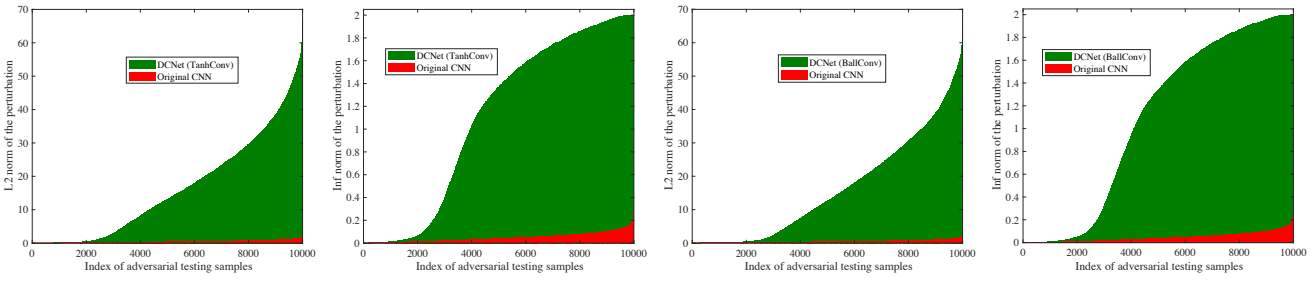
and SegConv is implemented using

$$f_d(\boldsymbol{w}, \boldsymbol{x}) = \begin{cases} \alpha \|\boldsymbol{x}\| \cdot g(\theta_{(\boldsymbol{w},\boldsymbol{x})}), & 0 \leq \|\boldsymbol{x}\| \leq \rho \cdot \mathbb{E}\{\|\boldsymbol{x}\|\} \\ (\beta \|\boldsymbol{x}\| + \alpha\rho \cdot \mathbb{E}\{\|\boldsymbol{x}\|\} - \beta\rho \cdot \mathbb{E}\{\|\boldsymbol{x}\|\}) \cdot g(\theta_{(\boldsymbol{w},\boldsymbol{x})}), & \rho \cdot \mathbb{E}\{\|\boldsymbol{x}\|\} < \|\boldsymbol{x}\| \end{cases}. \tag{21}$$

Hyperparameter Settings. For SphereConv, we use $\alpha = 1$. For BallConv, we use $\alpha = 1$ and a learnable ρ. For TanhConv, we use $\alpha = 1$ and a learnable ρ. For LinearConv, we use $\alpha = 1$. For SegConv, we use $\alpha = 1$, $\beta = 0.5$ and a learnable ρ. For LogConv, we use $\alpha = 1$. For MixConv, we use $\alpha = 1$ and $\beta = 1$. For CIFAR experiments, we use 128 batch size for all the networks. For ImageNet experiments, we use 40 batch size for all the networks.

C. More Experiments on Defense against Adversarial Attacks

We also evaluate the robustness of DCNets with the DeepFool attacks [19]. Note that, for all the experiments related to the adversarial attacks, our network do not use any optimization trick and is trained by original gradients. The results are given in Fig. 5. The x-axis denotes the index of the 10000 adversarial testing samples, and the y-axis denotes the strength (ℓ_2 norm or ℓ_∞ norm) of the perturbations in order to successfully fool the network. We could observe from the results that in order to fool the DCNets, the DeepFool attacks need to largely perturb the samples while it only takes a much smaller perturbation to fool the original CNNs. It implies that DCNets are much more difficult to fool. In other words, to fool the DCNets will take much more efforts than to fool the original CNNs, which shows the superior robustness of DCNets against adversarial examples.

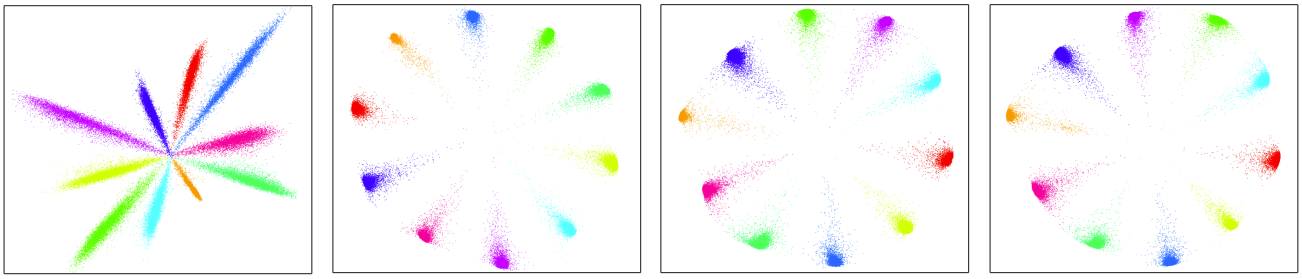


(a) DCNet (TanhConv) vs. original CNN (b) DCNet (BallConv) vs. original CNN

Figure 5: The strength of the adversarial perturbations to fool the network.

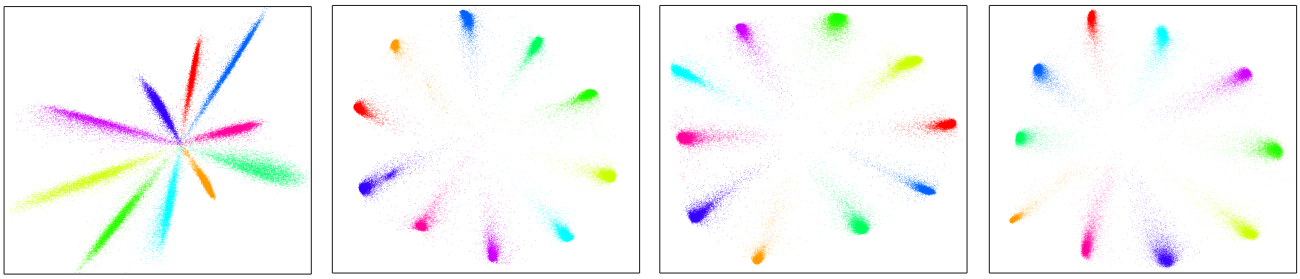
D. Feature Visualization on MNIST Dataset

We visualize the 2D feature on MNIST dataset. Specifically, we use a plain CNN with 6 convolutional layers ([3×3,32]×2-[3×3,64]×2-[3×3,128]×2) and 3 fully connected layers (256-2-10). Note that we set the output dimension (*i.e.*, the input



Baseline CNN (Original Convolution) | DCNet (SphereConv) | DCNet (BallConv) | DCNet (TanhConv)

Figure 6: 2D feature visualization on MNIST dataset with natural training.



Baseline CNN (Original Convolution) | DCNet (SphereConv) | DCNet (BallConv) | DCNet (TanhConv)

Figure 7: 2D feature visualization on MNIST dataset with adversarial training.

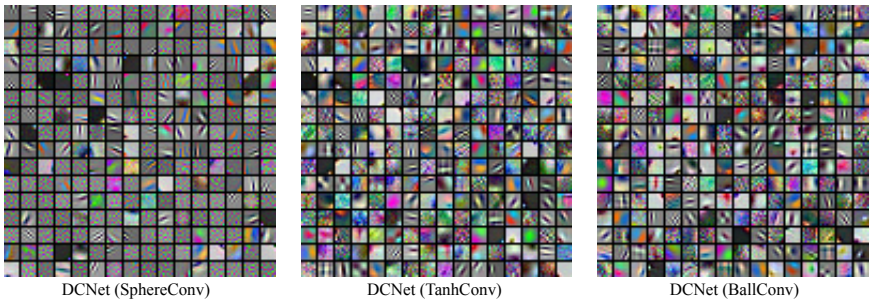
dimension of the last fully connected layer) as 2 and visualize these 2D features. We evaluate two types of training: natural training (*i.e.*, trained on the normal MNIST dataset) and adversarial training [3]. Note that, all the networks in this section are learned by original gradient updates. We do not use weight projection in the networks for the visualization purpose.

Natural Training. We plot the 2D features in Fig. 6. We could observe that DCNets (especially bounded decoupled operators) exhibit very different distributions with the original CNNs. Empirically, we observe that SphereConv, BallConv and TanhConv produce very compact and well-grouped features.

Adversarial Training. We also show the 2D features of adversarially trained models of baseline CNN, DCNet (SphereConv), DCNet (BallConv) and DCNet (TanhConv) in Fig.7. We could see that DCNets are still able to group the features in a more compact way than the original CNN even with the adversarial training.

E. Filter Visualization on ImageNet-2012

We train larger models with 256 filters in the first layer on ImageNet-2012. We visualize all the filters in the first layer for these compared methods in Fig. 8. One could see that DCNet with SphereConv learns more sparse filters, while DCNets with TanhConv and BallConv can learn richer types of filters. Moreover, because we use orthogonality constraints, the filters are not highly correlated unlike the original CNNs.



DCNet (SphereConv) | DCNet (TanhConv) | DCNet (BallConv)

Figure 8: Visualized filters from the first layer of DCNets on ImageNet-2012 dataset. Note that, this is learned by original gradient updates. We do not use weight projection in the networks for the visualization purpose.

F. Experiments on CIFAR-100 (stochastic gradient descent)

Additionally, we optimize the DCNets with stochastic gradient descent (SGD) with momentum and evaluate our models on CIFAR-100. We use the ResNet-32 architecture. The experimental setting is the same as the CIFAR-100 experiment in the main paper, except that we use SGD instead of ADAM. the results are given in Table 10. Surprisingly, using SGD could largely improve the performance of baseline. While optimizing the baseline model using ADAM gives us 26.69% error rate, optimizing the baseline using SGD gives us 21.55% error rate. Even though the baseline will be greatly improved by SGD, we still find that our DCNets optimized by SGD are better than the baseline and also perform slightly better than the DCNets optimized by ADAM.

Method	Linear	Cosine	Sq. Cosine
ResNet Baseline	-	21.55	-
SphereConv	21.71	21.61	24.62
BallConv	20.96	21.25	24.40
TanhConv	21.07	21.12	24.29
LinearConv	21.43	21.25	**20.54**
SegConv	**20.58**	**20.61**	20.61
LogConv	21.15	21.42	23.10
MixConv	20.82	21.20	21.19

Table 10: Testing error rate (%) of SGD-trained ResNet-32 on CIFAR-100.

G. Difference between Weighted Gradients and Weight Projection

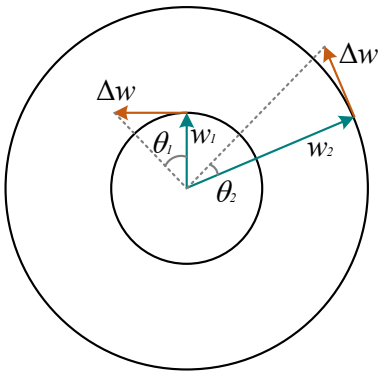


Figure 9: Illustration of weight update, given fixed $||\Delta w||$. Notice that with $||w_1|| < ||w_2||$, $\theta_1 > \theta_2$.

It is important to point out that the difference between weight projection and weighted gradients. Although weighted gradients eliminate the effect of normalizing w on the norm of gradients, the increment in angle (i.e. $\theta_{(w,x)}$) is different from that of weighted projection. Consider the simple case of LinearConv. Denote $y = \langle \frac{w}{||w||}, x \rangle$. Obviously, the gradient $||\nabla_w y||$ is always perpendicular to w. Therefore, **the original gradient update is optimizing the angle between** w **and** x. The modified gradient update Δw is $\nabla_w y \cdot ||w||$ if using weight gradients, and $\nabla_w y$ if using weight projection.

In the case of weighted gradients, even if all updates Δw have the same norm, the increment in 'angle' $\Delta\theta = \theta_{(w-\alpha\Delta w,x)} - \theta_{(w,x)}$ can vary, where α is the learning rate. Suppose the norm of the modified gradient $\|\Delta w\| = \|\nabla_w y\| \cdot \|w\|$ is fixed. $\Delta\theta$ can be ignored if $||w||$ is large, while close to 90 degrees if $\|w\|$ is extremely small. In other words, even if Δw is not dependent on $\|w\|$, $\Delta\theta$ is. See Figure 9 for illustration.

In contrast, weighted projection forces the norm of the weights $\|\boldsymbol{w}\|$ to be a constant s, so when we have fixed gradient $\|\Delta w\|$, the change of angle $\Delta\theta$ will also be a constant (because $\boldsymbol{w}$ and $\Delta\boldsymbol{w}$ are always perpendicular to each other).

To summarize, weighted gradients make the update of angle $\Delta\theta$ more "adaptive", while weight projection makes the update of angle $\Delta\theta$ more "fixed". The major reason for such difference is that the norm of the weights $\|\boldsymbol{w}\|$ is fixed to a constant s in weight projection, while the norm of the weights $\|\boldsymbol{w}\|$ is not a constant in weighted gradients. Different $\|\boldsymbol{w}\|$ refers to a hypersphere with different radius, as shown in Fig. 9.