DK-CNNs

Liu, Jialin; Chao, Fei; Lin, Chih Min; Zhou, Changle; Shang, Changjing

Published in:
Neurocomputing

DOI:
10.1016/j.neucom.2020.09.005

Publication date:
2021

Citation for published version (APA):
Liu, J., Chao, F., Lin, C. M., Zhou, C., & Shang, C. (2021). DK-CNNs: Dynamic kernel convolutional neural networks. *Neurocomputing*, *422*, 95-108. https://doi.org/10.1016/j.neucom.2020.09.005

DK-CNNs: Dynamic Kernel Convolutional Neural Networks

Jialin Liu[1], Fei Chao[1,2,*], Chih-Min Lin[3], Longzhi Yang[4], Changle Zhou[1], Changjing Shang[2]

Abstract

This paper introduces dynamic kernel convolutional neural networks (DK-CNNs), an enhanced type of CNN, by performing line-by-line scanning regular convolution to generate a latent dimension of kernel weights. The proposed DK-CNN applies regular convolution to the DK weights, which rely on a latent variable, and discretizes the space of the latent variable to extend a new dimension; this process is named "DK convolution". DK convolution increases the expressive capacity of the convolution operation without increasing the number of parameters by searching for useful patterns within the new extended dimension. In contrast to conventional convolution, which applies a fixed kernel to analyse the changed features, DK convolution employs a DK to analyse fixed features. In addition, DK convolution can replace a standard convolution layer in any CNN network structure. The proposed DK-CNNs were compared with different network structures with and without a latent dimension on the CIFAR and FashionMNIST datasets. The experimental results show that DK-CNNs can achieve better performance

*Corresponding author

Email addresses: jialin@stu.xmu.edu.cn (Jialin Liu), fchao@xmu.edu.cn (Fei Chao), cml@saturn.yzu.edu.tw (Chih-Min Lin), longzhi.yang@northumbria.ac.uk (Longzhi Yang), dozero@xmu.edu.cn (Changle Zhou), cns@aber.ac.uk (Changjing Shang)

[1]Department of Artificial Intelligence, School of Informatics, Xiamen University, China

[2]Institute of Mathematics, Physics and Computer Science, Aberystwyth University, UK.

[3]Department of Electrical Engineering, Yuan Ze University, Chung-Li, Tao-Yuan 320, Taiwan.

[4]Department of Computer and Information Sciences, Northumbria University, Newcastle upon Tyne NE1 8ST, UK.

than regular CNNs.

Keywords: Deep neural networks, convolutional neural networks, convolution kernel
2010 MSC: 00-01, 99-00

1. Introduction

Deep convolutional neural networks (CNNs) have become the most influential models of sensory data, such as images, video, and audio [1, 2, 3, 4, 5, 6]. CNNs take advantage of translation variance in perception tasks to use shared weights in different locations of the feature map. For image data, translation invariance exists in both the length and the width directions; therefore, CNNs can share weights in both of these dimensions. In this way, CNNs reduce the number of parameters required in the neural network and improve the generalization performance. However, many other transformation-invariant properties exist among the data features and must be approximated by neural networks. Nevertheless, the practical application of CNN models, which obtain state-of-the-art results, requires enormous amounts of clear data [7, 8, 9], and the datasets must be extended by augmenting them with data [10, 11].

Several studies have attempted to extend other translation-invariant spaces. The most common way is to convert the convolution operation into rotation and reflection [12, 13]. This extension allows the model to adapt to rotation- and reflection-invariant features in the data. Bruna et al. designed a wavelet scattering network that can compute translation-invariant image representation[14]. However, the efficiency of adopting convolution operations in different translation-invariant spaces depends on tasks and prior knowledge. Therefore, we perform convolution in a dimension that is learned by itself. The basic operation of each filter in a convolution layer is the elementwise multiplication of a feature map with the kernel weights, which is given by:

$$\mathrm{y}(r_0) = \sum_{r_n \in \mathcal{R}} \mathrm{w}(r_n) \cdot \mathrm{x}(r_0 + r_n), \tag{1}$$

where r_n enumerates the spatial locations in $\mathcal{R}$; w denotes the weight of the convolutional kernel; and y and x are the features of two adjacent convolution layers. Since each element of w is a constant, $\mathrm{y}(r_0)$ is also a constant. If unfixed kernel weights given by $\mathrm{w}(a, r_n)$ are adopted, a new dimension would

be extended. Eq. (1) can thus be written as:

$$\mathrm{y}(a, r_0) = \sum_{r_n \in \mathcal{R}} \mathrm{w}(a, r_n) \cdot \mathrm{x}(r_0 + r_n), \tag{2}$$

where a denotes the location in the latent dimension and $\mathrm{y}(a, r_0)$ depends on the latent variable a. As a result, convolution operations can be employed in this dimension.

If we use learnable parameters to parameterize the function $\mathrm{w}(a, r_n)$, the latent dimension is obtained during training. In this work, we use piecewise sine curves to define the unfixed kernel weights to make the dynamic kernel (DK) convolution operation both flexible and straightforward, and the unfixed kernel weights $\mathrm{w}(a, r_n)$ are used as DK weights. In experiments on the Canadian Institute for Advanced Research (CIFAR) [15] and Fashion-Modified National Institute of Standards and Technology (MNIST) [16] datasets, we employ two different DK-CNN structures: one structure extends the dimension in the first layer, and another structure extends the dimension following 3D CNNs and average pooling in the latent dimension. The results show that both structures exhibit better performance than regular CNNs in most cases. In addition, because the degree of weight sharing is increased in the process of obtaining DK weights (see details in the Methods section), the number of parameters in a DK-CNN is not more than that in a regular CNN.

Several previous works have attempted to design a new convolutional network. For example, deep receptive field networks learn the weights of several filter bases [17]. PCANet employs principal component analysis (PCA) to construct a deep learning network [18, 19]. Deep eigenfilters are obtained by eigendecomposition [20]. In this work, we design a new approach to improve CNNs. Our contribution is a learnable latent dimension for convolution operations, and the result is called a DK-CNN. In addition, we describe the initialization of DK-CNNs and perform a complexity analysis. Finally, we perform experiments to verify that DK-CNNs can achieve better performance than regular CNNs.

The remainder of this paper is organized as follows. Section 2 introduces the related background for the proposed model, including the convolution units of CNNs. Section 3 describes the proposed kernel convolution operation. Section 4 details the abovementioned experiments and discusses the experimental results. Finally, Section 5 provides a brief conclusion and directions for future work.

2. Related Work

Previous investigations have demonstrated the performance of CNNs, and most of those studies focused on network structure. However, research on the convolution unit is also necessary to help us understand the principles governing the high performance of CNNs and to provide a basis and clues for further improving the convolution unit. To date, most studies on the convolution unit have been approached from two perspectives: 1) translation and 2) the receptive field.

The translation-invariant property of a CNN can eliminate the effect of the recognition error caused by the target position within an image. Other transformations that have invariant properties can also be used in CNNs; the most common is the rotation transformation. For instance, rotation symmetry was employed to predict the morphology of galaxies by Dieleman et al. [12], whose work was later extended to computer vision tasks [21]. Other transformations, such as scaling, shear, and reflection transformations, were introduced recently. For example, Gens et al. introduced symmetry group theory and proposed deep symmetry networks (symnets) [22], which utilize rotation, scaling, and shear transformations to enable the model to capture a wide range of invariances. Furthermore, Taco et al. used the concepts of symmetry groups to develop the 2D integer translation, which is one example of a symmetry group, and introduced group-invariant CNNs (G-CNNs) [13]. Invariant properties depend on data; thus, it is possible to design convolutional networks for different tasks. Hoogeboom et al. extended G-CNNs to produce a CNN capable of analyzing hexagonal tiling called HexaConv [23], and spherical CNNs were proposed to analyse 3D spherical tiling [24].

The receptive field of a regular CNN with one layer is the same size as the weight of the kernel and is generally rectangular for image data. Most studies on the receptive field have focused on the application of convolution units with different sizes or shapes. For example, dilated convolution [25] was proposed for a sparse convolution unit to increase the receptive field size with the same number of parameters. The effective receptive field (ERF) was discussed by [26], whose conclusions indicate that if the number of parameters is squared, the ERF increases linearly. According to ERF analysis, the size of the ERF for dilated convolution is larger than that of a regular receptive field. ERF theory not only explains the effectiveness of dilated convolution but also leads to another idea: changing the shape of the receptive field filter. Nevertheless, ERF theory is based on the rectangular receptive field;

therefore, if a receptive field with a different shape is adopted, its ERF size may not be restricted by ERF theory. For example, an active convolution unit (ACU) [27] applies a convolution unit with no fixed shape. However, the receptive field of an ACU changes its shape during training; then, during testing, the shape is set to a fixed shape. Deformable convolution [28] extends the unfixed shape of a receptive field to be dynamic.

3. Methods

We extend a latent dimension of kernel weights beyond simply rotating and reflecting the kernel. The proposed kernel convolution operation consists of three steps: 1) convert the fixed kernel weights into DK weights; 2) implement a convolution operation with the DK weights; and 3) discretize the continuous latent dimension. We summarize the frequently used notations in Table 1.

3.1. Dynamic Kernel

The whole process of DK convolution is illustrated in Fig. 1. We establish a latent dimension of kernel weights by defining the kernel weights as functions of a latent valuable; in this work, we use sine functions. Choosing sine functions is based on the following considerations: 1) the sine function is a period function, and thus, we need to focus on its property only in one period other than along the whole axis; and 2) a function obtained by adding sine functions with the same period is still regarded as a sine function with the same period.

The weights of each channel are denoted by the weighted sum of sine curves. Therefore, the form of the DK weights is defined by:

$$\mathrm{w}(a) = \hat{\mathrm{w}}\left[\mathrm{G} \odot \sin(a + \mathrm{B}) + \mathrm{D}\right], \ a \in (-\pi, \pi], \tag{3}$$

where a denotes the latent variable; $\mathrm{w}(a)$ denotes the kernel weights depending on a and is a $c \times k^2$ matrix; k is the filter size of a 2D CNN; c denotes the number of input channels; and $\odot$ denotes elementwise multiplication. $\hat{\mathrm{w}}$ is a $c \times h$ matrix, whereas G, B, and D are $h \times k^2$ matrices, and h is the number of hidden variables in each channel, all of which are learnable during training.

The weight tensor, $\mathrm{w}(a)$, consists only one output channel; therefore, the size of the weight tensor is set to $c \times k^2$ rather than $c^2 \times k^2$. This

Notation	Description	Notation	Description
$\mathrm{w}(a)$	dynamic kernel weights	a	latent variable
$\hat{\mathrm{w}}$	weights of sine curves	k	filter size
G	amplitudes of sine curves	$\mathrm{w}(c_n, a)$	c_n^{th} line of $\mathrm{w}(a)$
B	phases of sine curves	$\hat{\mathrm{w}}(c_n)$	c_n^{th} line of $\hat{\mathrm{w}}$
D	constant items in sine curves	x	features matrix
$\mathrm{T_m}(a, \Theta)$	transformation matrix	y	output feature
c	number of input channels	$y(a)$	output curve
h	number of hidden variables	$a_1, \cdots, a_M$	sample points
$\mathrm{x}(c_n)$	c_n^{th} line of features matrix x	N_p	number of pieces

Table 1: Notations and descriptions.

representation is common in previous studies, such as [29]. Furthermore, h is a hyperparameter that needs to consider both the number of parameters and the abilities of the model; we set h to 4 in all experiments herein.

3.2. DK Convolution

Any transformation matrix $\mathrm{T_m}(a, \Theta)$ can be adopted as $\mathrm{w}(a) = \hat{\mathrm{w}}\mathrm{T_m}(a, \Theta)$. In this work, the transformation matrix is given by:

$$\mathrm{T_m}(a, \{\mathrm{G}, \mathrm{B}, \mathrm{D}\}) = \mathrm{G} \odot \sin(a + \mathrm{B}) + \mathrm{D}. \tag{4}$$

These properties of the sine function provide convenience for linear transformations; therefore, $\mathrm{w}(a)$ is defined as:

$$\mathrm{w}(a) = \mathrm{w_{cos}} \sin(a) + \mathrm{w_{sin}} \cos(a) + \mathrm{w_{cons}}, \ a \in (-\pi, \pi], \tag{5}$$

$$\begin{cases} \mathrm{w_{cos}} = \hat{\mathrm{w}}\mathrm{G} \odot \cos(\mathrm{B}), \\ \mathrm{w_{sin}} = \hat{\mathrm{w}}\mathrm{G} \odot \sin(\mathrm{B}), \\ \mathrm{w_{cons}} = \hat{\mathrm{w}}\mathrm{D}, \end{cases} \tag{6}$$

where $\mathrm{w}(a)$ is still a sine curve with a period of 2π. This idea is similar to the traditional Fourier transform. However, we use only sine functions with the same period because this approach is easy to implement; in the

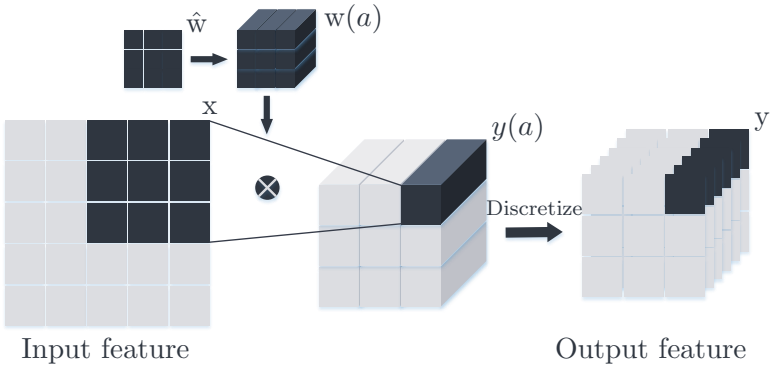

Figure 1: A typical DK convolution operation for an image with one channel, where x denotes the features matrix in the receptive field, presented as a black block. The black cuboid denoted by $y(a)$ is the sum-product between x and $\mathrm{w}(a)$. The black blocks in the output feature denoted by y are the discretization of $y(a)$.

future, we will use other functions and test their performance. Here, we establish dependencies between the kernel weights and the latent variable a by performing convolution operations in pixel and input channel dimensions, and $y(a)$ is given by:

$$\begin{aligned} y(a) &= \sum_{c_n \in \mathcal{C}} \mathrm{w}(c_n, a)\mathrm{x}(c_n)^{\mathrm{T}} \\ &= y_{\cos} \sin(a) + y_{\sin} \cos(a) + y_{\mathrm{cons}}, \end{aligned} \tag{7}$$

$$\begin{cases} y_{\cos} = \sum_{c_n \in \mathcal{C}} \mathrm{w}_{\cos}(c_n)\mathrm{x}(c_n)^{\mathrm{T}}, \\ y_{\sin} = \sum_{c_n \in \mathcal{C}} \mathrm{w}_{\sin}(c_n)\mathrm{x}(c_n)^{\mathrm{T}}, \\ y_{\mathrm{cons}} = \sum_{c_n \in \mathcal{C}} \mathrm{w}_{\mathrm{cons}}(c_n)\mathrm{x}(c_n)^{\mathrm{T}}, \end{cases} \tag{8}$$

where $\mathrm{w}(c_n, a)$ denotes the c_n^{th} line of $\mathrm{w}(a)$ and $\mathrm{x}(c_n)$ denotes the c_n^{th} line of features matrix x with size $c \times k^2$.

The parameters of curve $y(a)$ are obtained by performing the convolution operation with $\mathrm{w}_{\cos}$, $\mathrm{w}_{\sin}$, and $\mathrm{w}_{\mathrm{cons}}$. From another perspective, the motion of kernel weights is equivalent to relative motion in the feature map; therefore,

Algorithm 1 DK convolution.

Require: Parameters of DK $\hat{\mathrm{w}}, \{\mathrm{G}_i, \mathrm{B}_i, \mathrm{D}_i\}_i$; Features matrix in the receptive field x; Sample points $\{a_1, a_2, \cdots, a_M\}$; Output feature y

```
y ← {}
for all G, B, D ∈ {G_i, B_i, D_i}_i do
    \\ Obtain DK weights:
    w(a) ← ŵ [G ⊙ sin(a + B) + D]
    \\ Convolution with DK weights:
    y(a) ← Σ_{c_n ∈ C} w(c_n, a) x(c_n)^T
    \\ Discretize the latent dimension:
    for all a ∈ {a_1, a_2, ··· , a_M} do
        y ← y ∪ {y(a)}
    end for
end for
return vector(y)
```

$y(a)$ is defined by:

$$\begin{aligned} y(a) &= \sum_{c_n \in \mathcal{C}} \mathrm{w}(c_n, a)\mathrm{x}(c_n)^{\mathrm{T}} \\ &= \sum_{c_n \in \mathcal{C}} \hat{\mathrm{w}}(c_n)\mathrm{T}_{\mathrm{m}}(a, \Theta)\mathrm{x}(c_n)^{\mathrm{T}} = \sum_{c_n \in \mathcal{C}} \hat{\mathrm{w}}(c_n)\mathrm{x}(c_n, a)^{\mathrm{T}}, \end{aligned} \tag{9}$$

where $\mathrm{x}(c_n, a)^{\mathrm{T}} = [\mathrm{G} \odot \sin(a + \mathrm{B}) + \mathrm{D}]\, \mathrm{x}(c_n)^{\mathrm{T}}$.

The motion of the convolution kernel is equivalent to the relative movement of x in the feature space, which is given by:

$$\begin{aligned} y(a + q) &= \sum_{c_n \in \mathcal{C}} \hat{\mathrm{w}}(c_n)\, [\mathrm{G} \odot \sin(a + p + \mathrm{B}) + \mathrm{D}]\, \mathrm{x}(c_n)^{\mathrm{T}} \\ &= \sum_{c_n \in \mathcal{C}} \hat{\mathrm{w}}(c_n)\mathrm{x}(c_n, a + p)^{\mathrm{T}}. \end{aligned} \tag{10}$$

where q denotes the translation distance. Then, the convolution process is employed in the latent dimension.

3.3. Discretization of the Continuous Dimension

The latent variable, a, is continuous, so the latent space is also continuous. This would make the latter operation inconvenient because the convolution

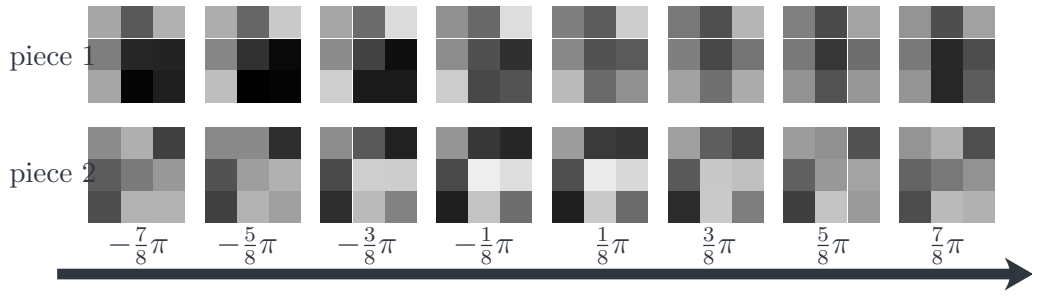

Figure 2: An example of a 3×3 kernel in a DK-CNN representing a weight matrix between a single input and output channel. Each row represents the weight of the kernel for the potential variables $a = -\frac{7}{8}\pi, -\frac{5}{8}\pi, \cdots, \frac{7}{8}\pi$ in the 2 pieces.

operation in the latent space is an integral operation from π to $-\pi$. If a nonlinear function, such as the rectified linear unit (ReLU) function [30], is used to process the feature map, the operation becomes $\max(0, y(a))$, which is a piecewise function, and the process is given by:

$$\int_{-\pi}^{\pi} w'(a) \max(0, y(a)) da = \int_{a \in \mathcal{A}} w'(a) y(a) da, \tag{11}$$

where $w'(a)$ is the weight in the next layer and $\mathcal{A} = \{a | y(a) > 0, a \in (-\pi, \pi]\}$. Graphics processing units (GPUs) are not suitable for calculating integrals in the integral $\mathcal{A}$. Even if we wish to choose a different nonlinear function, it is difficult to find a function with a performance similar to that of the ReLU function and whose integral is easy to calculate. Therefore, we discretize the curve $y(a)$ in $(-\pi, \pi]$ to form a feature vector, $[y(a_1), y(a_2), \cdots, y(a_M)]^{\mathrm{T}}$ (M denotes the number of points), and the points $\{a_1, a_2, \cdots, a_M\}$ are sampled on the same intervals. In this way, the convolution operation is easy to perform regardless of which nonlinear function is applied. The whole process is illustrated in Algorithm 1. In addition, since the ReLU function is currently the most popular option, we choose to utilize ReLU in our experiment.

3.4. Multi-Piecewise Curves

If a nonlinear function such as ReLU is adopted, ReLU merely truncates one minimum value from the curve since only one maximum and one minimum exist within one period. To increase the expressive capacity of the latent dimension, multiple transformation matrices are used in this work.

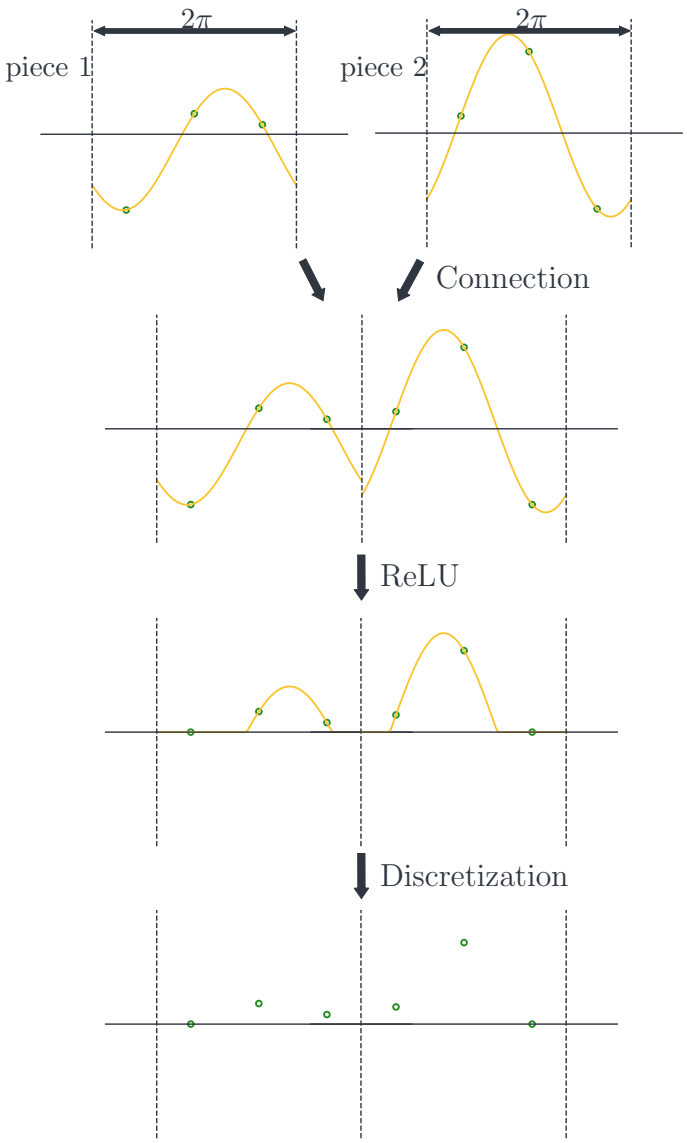

Figure 3: An example of the discretization process applied to multi-piecewise curves with 2 pieces of a sine curve. ReLU is applied as a nonlinear function.

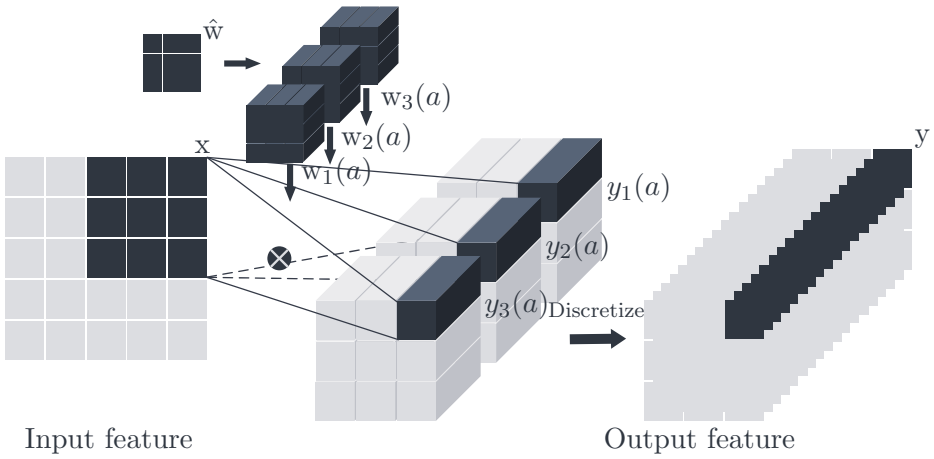

Figure 4: A typical process of DK convolution for an image with 1 channel, where x denotes the features matrix in the receptive field, presented as a black block. The black cuboids denoted by $y_1(a)$, $y_2(a)$ and $y_3(a)$ are the sum-products between x and $\mathrm{w}_1(a)$, $\mathrm{w}_2(a)$ and $\mathrm{w}_3(a)$, respectively. The black blocks in the output feature denoted by y represent the concatenated discretizations of $y_1(a)$, $y_2(a)$ and $y_3(a)$.

Each transformation matrix $\mathrm{T}_\mathrm{m}(a, \Theta)$ is used to calculate one DK weight matrix $\mathrm{w}(a)$. The curves in each $\mathrm{w}(a)$ within a single period connect to the curves in other $\mathrm{w}(a)$. Then, the elements in the final DK weight matrix compose a multi-piecewise sine function.

A model with a larger number of pieces has a better ability but more parameters. Therefore, we limit the number of pieces to $N_p = 2$ to prevent the number of parameters in the DK-CNN from exceeding that in a standard CNN to ensure fairness in the experiments. As an example, we demonstrate a 3×3 kernel with 2 pieces of a sine curve in Fig. 2. Since 3 points are sufficient to determine a sine curve, 3 points at $\{-\frac{2}{3}\pi, 0, \frac{2}{3}\pi\}$ are chosen for each piece in our experiments. Note that the intervals between the two adjacent points are identical. The experimentation also applies the same three points of one piece as $\{-\frac{2}{3}\pi, 0, \frac{2}{3}\pi\}$.

In Fig. 3, we depict the discretization process for a multi-piecewise sine curve. First, we connect the curves of two sine functions within one period. Then, we exclude the parts of the curve with values less than zero. Finally, we remove the continuous curve, leaving only discrete points.

3.5. Normalization of the Transformation Matrix

In the proposed DK-CNN, an element of the DK matrix is defined as:

$$w_j(c_n, a) = \hat{\mathrm{w}}(c_n)\left[\mathrm{g}_j \odot \sin(a + \mathrm{b}_j) + \mathrm{d}_j\right], \tag{12}$$

where g_j, b_j and d_j are the jth columns of G, B and D, respectively. $\mathrm{T_m}(a, \Theta)$ can be used only to indicate the proportion between the sine curve and constant, and the scale is determined by $\hat{\mathrm{w}}$. Therefore, the calculation of $w_j(c_n, a)$ is rewritten as:

$$w_j(c_n, a) = \hat{\mathrm{w}}(c_n)\left[\sqrt{2}\hat{\mathrm{g}}_j \odot \sin(a + \mathrm{b}_j) + \hat{\mathrm{d}}_j\right], \tag{13}$$

where $\hat{\mathrm{g}}_j$ and $\hat{\mathrm{d}}_j$ are obtained by:

$$\begin{cases} \hat{\mathrm{g}}_j = \dfrac{\mathrm{g}_j}{\sqrt{\|\mathrm{g}_j\|^2 + \|\mathrm{d}_j\|^2}}, \\ \hat{\mathrm{d}}_j = \dfrac{\mathrm{d}_j}{\sqrt{\|\mathrm{g}_j\|^2 + \|\mathrm{d}_j\|^2}}. \end{cases} \tag{14}$$

The principle of weight normalization [31] is to balance the gradients of g_j and d_j. If the gradient variances of g_j and d_j are large, $1/\sqrt{\|\mathrm{g}_j\|^2 + \|\mathrm{d}_j\|^2}$ might decrease quickly, thereby reducing the effects of noise in the gradient. A coefficient of $\sqrt{2}$ is applied to lead the gradient variances of g_j and d_j, making them equal to each other. t, $\hat{g}$, b, and $\hat{d}$ represent the elements of $\mathrm{T_m}(a, \Theta)$, $\hat{\mathrm{g}}_j$, b_j and $\hat{\mathrm{d}}_j$, respectively. Δt is assumed to obey a probability distribution with an expectation of zero, b is assumed to obey the uniform distribution of $(-\pi, \pi]$, and Δt and b are independent of each other. Then, we have:

$$\begin{aligned} Var[\Delta\hat{g}] &= Var[\Delta t] Var[\sqrt{2}\sin(a + b)] \\ &= Var[\Delta t] = Var[\Delta\hat{d}]. \end{aligned} \tag{15}$$

Finally, we rewrite Algorithm 1 as Algorithm 2.

3.6. Complexity Analysis

Converting a fixed kernel into a DK is conducted by performing matrix multiplication between $\hat{\mathrm{w}}$ and $\mathrm{T_m}(a, \Theta)$; therefore, the time complexity is $O(N_p chk^2)$ with parallel computations between N_p pieces. Since $\mathrm{w_{cos}}$, $\mathrm{w_{sin}}$ and $\mathrm{w_{cons}}$ need to be calculated, the computational cost of convolution with a

Algorithm 2 DK convolution.

Require: Parameters of DK $\hat{\mathrm{w}}, \{\mathrm{G}_i, \mathrm{B}_i, \mathrm{D}_i\}_i$; Features matrix in the receptive field x; Sample points $\{a_1, a_2, \cdots, a_M\}$; Output feature y

y $\leftarrow \{\}$
for all $\mathrm{G}, \mathrm{B}, \mathrm{D} \in \{\mathrm{G}_i, \mathrm{B}_i, \mathrm{D}_i\}_i$ **do**
    $\hat{\mathrm{G}}, \hat{\mathrm{D}} \leftarrow$ normalize G, D by Eq. (14) and
    \\ Obtain DK weights
    $\mathrm{w}_{\cos} = \hat{\mathrm{w}}\,[\mathrm{G} \odot \cos(\mathrm{B})]$
    $\mathrm{w}_{\sin} = \hat{\mathrm{w}}\,[\mathrm{G} \odot \sin(\mathrm{B})]$
    $\mathrm{w}_{\mathrm{cons}} = \hat{\mathrm{w}}\mathrm{D}_i$
    \\ Convolution with DK weights
    $y_{\cos},\ y_{\sin},\ y_{\mathrm{cons}} \leftarrow$ Eq. (6) with $\mathrm{w}_{\cos}, \mathrm{w}_{\sin}, \mathrm{w}_{\mathrm{cons}}$
    \\ Discretize the latent dimension
    for all $a \in \{a_1, a_2, \cdots, a_M\}$ **do**
        $y(a) \leftarrow y_{\cos} \sin(a) + y_{\sin} \cos(a) + y_{\mathrm{cons}}$
        $\mathrm{y} \leftarrow \mathrm{y} \cup \{y(a)\}$
    end for
end for
return vector(y)

DK requires is thrice that of standard convolution, but their time complexity is the same, $O(ck^2)$.

For discretization, the same operation is repeated M times, and the time complexity of discretizing the continuous latent dimension is $O(M)$ with parallel computations between M points. Next, we analyse the time complexity of normalizing the transformation matrix. Except for the extraction of a root, all operations are conducted one time for each element in g_j and d_j; thus, the time complexity of the normalization process is proportional to the combined size of g_j and d_j, $O(hk^2)$.

Finally, we analyse the number of learnable parameters in the DK-CNN. The learnable parameters of DK-CNNs include the $\hat{\mathrm{w}}$ of each channel and the parameters of the transformation matrix, $\{\mathrm{G}, \mathrm{B}, \mathrm{D}\}$. We assume that the numbers of input channels and output channels are equal, namely, c. Moreover, the number of parameters in all $\hat{\mathrm{w}}$ is $c^2 \times h$. In addition, G, B, and D all have a size of $h \times k^2$, and thus, the overall number of parameters in one DK-CNN layer is $c^2 \times h + 3h \times k^2$.

3.7. Network Structure

The proposed method extends a new dimension for 2D images; therefore, the convolution layers after the DK-convolution layers need to handle 3D features. Thus, in this work, we evaluate the proposed method in two cases of alternating 1) reducing and 2) extending the dimension in the first layer. Average pooling should be used in the first case. The second case is easy to understand, so we provide additional details regarding the first case in the context of ResNet [32], DenseNet [33], and CNN4.

When DK convolution is applied to ResNet [32], each block contains two convolution layers: a 3D CNN layer with a $3 \times 3 \times 1$ kernel size and a 2D DK-convolution layer with 3×3 kernel size. Furthermore, global average pooling is performed in the extended new dimension prior to the 2D DK-convolution layer. In ResNet, when either the number of channels or the size of the feature map changes, a 2D CNN layer with a 1×1 kernel size is required in the cross-layer connection; hence, we use the 3D CNN layer with a $1 \times 1 \times 1$ kernel to replace the 2D CNN layer with a 1×1 kernel.

Since we utilize $h < k^2$ to reduce the number of parameters within the model, we do not use the DK-convolution layer for the convolution layer with a 1×1 kernel size. In DenseNet, each block contains a 3D CNN layer with a kernel size of $1 \times 1 \times 1$ and one 2D DK-convolution layer with a kernel size of 3×3. We perform global average pooling in the extended new dimension before the 2D DK-convolution layer.

One linear layer calculates the class score after processing the convolution layers. CNN4 with DK convolution contains two processes for extending dimensions, i.e., the DK-convolution layer, a nonlinear ReLU function, 3D convolution, another nonlinear ReLU function, and average pooling. Identical to a standard CNN, CNN4 batch normalization is applied before ReLU. The kernel size in the 3D CNN layer is $3 \times 3 \times 1$, and the kernel size in the 2D DK-CNN is 3×3. Max-pooling is performed after the 2nd and 3rd layers in the height and width dimensions.

3.8. Initialization of Filter Weights for DK-CNNs

Several existing studies have confirmed the importance of parameter initialization for deep networks [34, 29, 31, 35, 36]. Good initialization can improve the convergence of deep networks and prevent the model from reaching a local optimum. "Xavier" initialization [37] was proposed by Glorot and Bengio for filter weight initialization [34]. Xavier initialization can obtain good performance if the activation function is symmetric to 0; therefore,

this approach is unsuitable for the ReLU function. Alternatively, He et al. proposed an initialization approach for ReLU called "Kaiming" initialization [29]. In this work, we follow the principle of parameter initialization in [34, 29] to ensure that the inputs of all layers have the same variance.

3.8.1. Forward Propagation Case

Based on the work of He et al. [29], if the ReLU function is applied, then the parameter initialization must guarantee the following:

$$Var[y_l] = \frac{1}{2} n_l Var[w_l] Var[y_{l-1}], \tag{16}$$

where y_l, y_{l-1}, and w_l denote the random variables of each output element in the l-th and (l-1)-th layers and the elements in the weight matrix of the l-th layer, respectively. To guarantee $Var[y_l] = Var[y_{l-1}]$, we must have $Var[w_l] = 2/n_l$. Within DK-CNNs, we have:

$$w_l(a) = \sum_i \hat{w}_l^i \left[\sqrt{2} \hat{g}_l^i \sin(a + b_l^i) + \hat{d}_l^i \right], \tag{17}$$

where $w_l(a)$, $\hat{w}_l^i$, $\hat{g}_l^i$, b_l^i and $\hat{d}_l^i$ are the elements in w(a), ŵ, G, B, and D, respectively.

In particular, we must ensure that any points in the latent dimension have the same variance as the points in the last layer, i.e., $Var[y_l(a)] = Var[y_{l-1}]$. This condition is defined as:

$$Var\left[\sum_i \hat{w}_l^i \left[\sqrt{2} \hat{g}_l^i \sin(a + b_l^i) + \hat{d}_l^i \right] \right] = 2/n_l, \tag{18}$$

where $\hat{w}_l^i$ and $\sqrt{2}\hat{g}_l^i \sin(a + b_l^i) + \hat{d}_l^i$ are independent of each other. $\hat{g}_l^i$ and $\hat{d}_l^i$ are initialized first. Then, $\hat{w}_l^i$ and b_l^i are initialized on the basis of $\hat{g}_l^i$ and $\hat{d}_l^i$. The random variable b_l^i obeys the uniform distribution of $[-\pi, \pi]$ to ensure $E\left[\sqrt{2} \hat{g}_l^i \sin(a + b_l^i) + \hat{d}_l^i \right] = \hat{d}_l^i$, and $\hat{w}_l^i$ obeys a distribution with an expectation of 0.

Eq. (18) is transformed to the following:

$$\sum_i Var\left[\hat{w}_l^i\right] \left[2(\hat{g}_l^i)^2 Var\left[\sin(a + b_l^i)\right] + (\hat{d}_l^i)^2 \right] = 2/n_l. \tag{19}$$

For each i, $Var\left[\hat{w}_l^i\right]$ is equal, and we use $Var\left[\hat{w}_l\right]$ to simplify $Var\left[\hat{w}_l^i\right]$. In addition, $Var\left[\sin(a+b_l^i)\right] = \frac{1}{2}$ and $\sum_i \left[(\hat{g}_l^i)^2 + (\hat{d}_l^i)^2\right] = 1$. Thus, we ensure $Var[\hat{w}_l] = Var[w_l] = 2/n_l$ to guarantee $Var[y_l(a)] = Var[y_{l-1}]$, where $n_l = k_l^2 c_l$ and k_l and c_l are the filter size and the number of input channels in the l-th layer, respectively. Note that there are no requirements for the initialization of $\hat{g}_l^i$ and $\hat{d}_l^i$, so we simply initialize $\hat{g}_l^i$ and $\hat{d}_l^i$ by a mean of zero and a variance of $1/(hk_l^2)$.

3.8.2. Backward Propagation Case

In the backward propagation case, if the nonlinear ReLU function is once again applied, the parameter initialization must satisfy the following equation:

$$Var[\Delta x_l] = \frac{1}{2}\hat{n}_l Var[w_l] Var[\Delta x_{l+1}], \tag{20}$$

where Δx_l and Δx_{l+1} are the random variables of the gradient for each input element in the l and $l+1$ layers, respectively. To ensure that the gradients in all layers have the same variance, we must have $Var[w_l] = 2/\hat{n}_l$ and $\hat{n}_l = k_l^2 d_l$, where k_l and d_l are the filter size and number of the output channels of the l-th layer, respectively. However, we extend a new dimension in the DK-CNN; thus, Eq. (20) changes to:

$$Var[\Delta x_l] = \frac{1}{2}\hat{n}_l \sum_i Var[w_l(a_i)] Var[\Delta x_{l+1}(a_i)], \tag{21}$$

where $i = 0, 1, \cdots, p_l s_l$, in which p_l and s_1 denote the number of pieces and number of points, respectively, in a piece in the l-th layer. Similar to the forward propagation case, $Var[w_l(a_i)] = Var[\hat{w}_l] = Var[w_l],\ i = 0, 1, \cdots, p_l s_l$; therefore, the condition guaranteeing $Var[\Delta x_l] = Var[\Delta x_{l+1}(a_i)]$ is $Var[\hat{w}_l] = Var[w_l] = 2/(\hat{n}_l p_l s_l)$.

The initialization of $\hat{g}_l^i$ and b_l^i is not effective for the gradient and input in each layer; therefore, a distribution with a mean of zero and a variance of $1/(hk_l^2)$ is used for the initialization.

4. Experiments

The purpose of the following experiments is to determine whether DK-CNNs have better performance than regular CNNs. Therefore, the experiments are designed to compare the performance between DK-CNNs and regular CNNs with the same network structures. We evaluate all models in the format of "mean±std" based on 5 runs on top-1 error rates.

4.1. Datasets

CIFAR. The CIFAR dataset [15] contains 60k natural colour images, each of which has 32×32 pixels. CIFAR-10 consists of 10 categories of images, while CIFAR-100 consists of 100 categories of images. There are 50k and 10k images for the training and testing sets, respectively. We separate 5k training images from the training set for use as a validation set, and all categories within the validation set have the same number of images. To augment the data, we follow Krizhevsky et al. [38] by horizontally flipping and randomly cropping the images, which are padded by four pixels along each border. Accordingly, the missing pixels are filled with the reflection of the image.

Fashion-MNIST. The Fashion-MNIST dataset [16] contains 60k training and 10k testing grayscale images of different merchandises, and each image comprises 28×28 pixels. Fashion-MNIST consists of 10 categories of images. Similar to the procedure employed for CIFAR, we separate 5k training images for use as a validation set, and all categories within the validation set have the same number of images. To augment the data, we follow Zhong et al.'s work [39]: we randomly crop the images that are padded by 4 pixels of zeros along each border, and we horizontally flip the images and performing random erasing. The hyperparameters of the random erasing step are the same as those in Zhong et al.'s work [39].

4.2. Comparison Model

For a method to be useful, we postulate that such a method must be effective for popular models, both those in use currently and those developed in the future. To develop a useful CNN model that may have potential future applications, we use CNN4, ResNet [32] and DenseNet [33] as basic model frameworks to compare regular CNNs and DK-CNNs. All DK-CNN layers use 2 piecewise sine functions, and the number of transformation matrices h is equal to 4.

The structure of a ResNet block with DK convolution is shown in Table 2. Each block contains two convolution layers: one is a 3D standard convolution layer with a kernel size of $3 \times 3 \times 1$, and the other is a 2D DK-convolution layer with a kernel size of 3×3 that performs global average pooling in the extended new dimension prior to the 2D DK-CNN layer. When either the number of channels or the size of the feature map changes, the 3D CNN layer with a $1 \times 1 \times 1$ kernel size replaces the 2D CNN layer with a 1×1 kernel size, which is required in the cross-layer connection.

CNNs			DK-CNNs		
way-1	way-2		way-1	way-2	
identity	conv1	3×3	identity	conv1	$3 \times 3 \times 1$
	bnorm1	2D		bnorm1	3D
	relu	-		relu	-
	-	-		avgpool	$1 \times 1 \times all$
	conv2	3×3		kconv2	3×3
	bnorm2	2D		bnorm2	3D
	relu	-		relu	-
+			+		

Table 2: The structure of a ResNet block with DK convolution and standard convolution. Way-1 and way-2 are the two forward passageways within a layer.

The DenseNet structure contains bottleneck layers and compression [33]. The number of initial channels and growth rate of DenseNet are 24 and 12, respectively. The structure of a DenseNet block with DK convolution is shown in Table 3. Each block contains one 3D standard convolution layer with a kernel size of $1 \times 1 \times 1$ and one 2D DK-convolution layer with a kernel size of 3×3.

The CNN4 model with regular convolution has 4 convolution layers with a 3×3 kernel and 64 filters. In addition, the structure of the CNN4 model with DK convolution is shown in Table 4. Two standard convolution layers are replaced by DK-convolution layers. The number of parameters in CNN4 is larger than that in ResNet20 since we do not use pooling between the last convolution layer and the linear layer. The parameters are concentrated mainly within the last layer of CNN4, but the number of parameters in the convolution layer in CNN4 is less than that in ResNet20 or in other neural networks.

In the structure that uses DK convolution only in the first layer, the 2D convolution layer with kernel sizes of 3×3 and 1×1 are replaced by `C3D3` and `C3D1`, respectively. This structure is denoted as DK-CNN(OF) in Tables 5∼10.

No that, we choose the value of hyperparameters based on the number of learnable parameters. The number of learnable parameters of DK-CNNs

CNNs			DK-CNNs		
way-1	way-2		way-1	way-2	
identity	bnorm1	2D	identity	bnorm1	3D
	relu	-		relu	-
	conv1	1×1		conv1	$1 \times 1 \times 1$
	bnorm2	2D		bnorm2	3D
	relu	-		relu	-
	-	-		avgpool	$1 \times 1 \times all$
	conv2	3×3		kconv2	3×3
Concatenate			Concatenate		

Table 3: The structure of a DenseNet block with DK convolution and standard convolution. Way-1 and way-2 are the two forward passageways within a layer.

should be close to the number of learnable parameters of comparative model.

4.3. Training

The same training configuration is used for both the proposed DK-convolution layer and the regular convolution layer to control the variables. ResNet and DenseNet are trained by stochastic gradient descent (SGD) on both the CIFAR and the Fashion-MNIST datasets. The minibatch size is set to 64. The initial learning rate is 0.1 and decays by 10 at 150 and 225 epochs. Similar to Huang et al.'s work [33], we adopt 0.9 as the Nesterov momentum [40] and 1.0×10^{-4} as the weight decay. In contrast, the CNN4 models both with and without DK convolution are trained by Adam [41] on the CIFAR and Fashion-MNIST datasets. The minibatch size is set to 128. The initial learning rate is 1.0×10^{-3} and decays by 2 at 100 and 150 epochs. Weight decay is not used in CNN4 (both with and without DK convolution).

For the preprocessing phase on CIFAR-10 and CIFAR-100, we use 45k images for the training set and 5k images for the validation set. For the final run, all 50k training images are used, and we report the test error on the test set with the highest accuracy on the validation set during preprocessing. On Fashion-MNIST, we train only one time and report the test error on the test set with the highest accuracy on the validation set.

CNNs		DK-CNNs	
way-2		way-2	
conv1	3×3	kconv1	3×3
bnorm1	2D	bnorm1	3D
relu	-	relu	-
conv1	3×3	conv2	$3 \times 3 \times 1$
bnorm2	2D	bnorm2	3D
relu	-	relu	-
-	-	avgpool	$1 \times 1 \times all$
maxpool	2×2	maxpool	2×2
conv3	3×3	kconv3	3×3
bnorm3	2D	bnorm3	3D
relu	-	relu	-
maxpool	2×2	maxpool	$2 \times 2 \times 1$
conv4	3×3	conv4	$3 \times 3 \times 1$
bnorm4	-	bnorm4	-
relu	-	relu	-
-	-	avgpool	$1 \times 1 \times all$
linear	*classes*	linear	*classes*

Table 4: The structure of CNN4 with DK convolution and standard convolution.

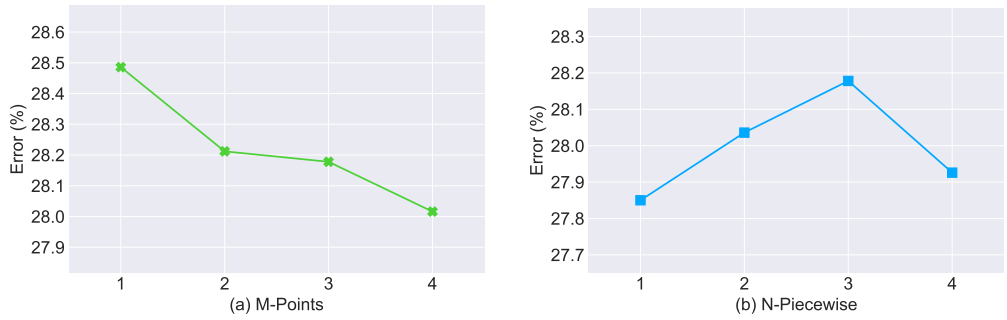

Figure 5: The averages Top-1 error rates (%) of DK-CNN(OF) on CIFAR-100 based on five runs with the varying number of discretization M-points and the varying number of pieces N_p.

4.4. Results

We evaluate the influence of hyerparameters on the performance of the DK-convolutions. The results shown in Fig. 5 are the top-1 error rates of DK-CNN(OF) based on five runs on ResNet32 on CIFAR-100 with the varying number of discretization M-points and the varying number of pieces N_p. In the figure, the error is decreasing when the number of discretization points is reducing. However, the number of pieces has not significantly affected on the performance of DK-convolutions.

The results of the experiments on Fashion-MINST, CIFAR-10 and CIFAR-100 are summarized in Tables 5, 7 and 8, respectively. We also compare the numbers of parameters in each model in Tables 6, 9 and 10. Most of the models with DK-CNNs have higher accuracy. The results show that DK-CNNs are effective not only for small CNNs but also for deeper CNN structures, such as ResNet and DenseNet. Since all of our results are based on five runs, our approach displays a steady improvement in performance.

In addition, since the number of hidden variables, h, is less than the kernel size, k^2, and several groups of G, B, and D are shared between different input channels, most of the models with DK-CNNs do not have more parameters than regular CNNs; in fact, some models with DK-CNNs have fewer parameters. These results illustrate that the proposed method is effective.

In Fig. 6, we plot the 3D feature maps of ResNet20 with DK convolution. The 3D feature map of the turtle is sparser than those of the otter and lawnmower. Comparing the 3D feature maps of the otter, lawnmower, and

Model	Fashion-MNIST		
	Baseline	DK-CNN	DK-CNN(OF)
CNN4	5.96±0.06	**5.86±0.13**	**5.66±0.14**
ResNet20	5.08±0.16	**4.78±0.23**	**4.66±0.15**
ResNet32	4.79±0.24	**4.78±0.13**	**4.77±0.12**
DenseNet40	5.11±0.16	**4.95±0.22**	**4.79±0.07**
DenseNet52	4.83±0.16	5.02±0.20	**4.78±0.23**

Table 5: The error rates (%) on Fashion-MNIST. All results are produced by our research group. The highlighted results indicate the DK-CNN results that are better than the regular CNN results.

Model	Fashion-MNIST		
	Baseline	DK-CNN	DK-CNN(OF)
CNN4	0.14M	0.15M	0.16M
ResNet20	0.27M	0.27M	0.28M
ResNet32	0.47M	0.46M	0.47M
DenseNet40	0.18M	0.18M	0.18M
DenseNet52	0.26M	0.26M	0.27M

Table 6: The numbers of parameters in each model on Fashion-MNIST.

Model	CIFAR-10		
	Baseline	DK-CNN	DK-CNN(OF)
CNN4	11.43±0.22	**11.36±0.20**	**11.01±0.31**
ResNet20	7.12±0.09	**6.85±0.16**	**6.76±0.27**
ResNet32	6.16±0.17	6.21±0.15	**6.12±0.33**
DenseNet40	6.92±0.26	**6.66±0.18**	**6.61±0.21**
DenseNet52	5.92±0.20	5.97±0.17	**5.90±0.20**

Table 7: The error rates (%) on CIFAR-10. All results are produced by our research group. The highlighted results indicate the DK-CNN results that are better than the regular CNN results.

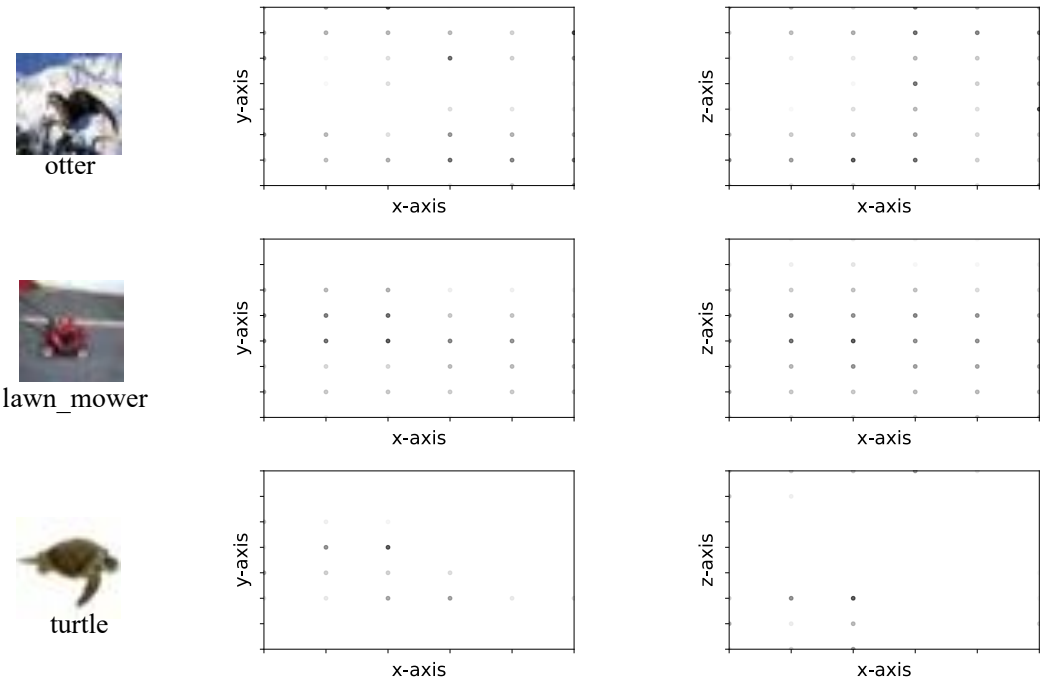

Figure 6: Examples of 3D features in DK-CNNs. These plots represent the output features of the last DK-CNN layer in ResNet20. The pictures corresponding to the 3D features on the left originate from CIFAR100 [15]. We project the 3D features into two dimensions and show the projections on the right. The X-axis is the latent dimension of kernel weights. For each image, we show the feature map of only one channel, where the maximum and minimum of each channel correspond to the deepest and lightest colours, respectively.

Model	CIFAR-100		
	Baseline	DK-CNN	DK-CNN(OF)
CNN4	38.14±0.30	**37.26±0.26**	**37.20±0.36**
ResNet20	30.52±0.25	**29.85±0.58**	**29.79±0.26**
ResNet32	29.02±0.50	**28.16±0.50**	**28.18±0.17**
DenseNet40	28.64±0.41	28.70±0.48	**28.21±0.44**
DenseNet52	26.71±0.28	26.86±0.32	**26.52±0.37**

Table 8: The error rates (%) on CIFAR-100. All results are produced by our research group. The highlighted results indicate the DK-CNN results that are better than the regular CNN results.

Model	CIFAR-10		
	Baseline	DK-CNN	DK-CNN(OF)
CNN4	0.52M	0.53M	0.54M
ResNet20	0.28M	0.27M	0.28M
ResNet32	0.47M	0.46M	0.48M
DenseNet40	0.19M	0.19M	0.19M
DenseNet52	0.28M	0.28M	0.27M

Table 9: The numbers of parameters in each model on CIFAR-10.

Model	CIFAR-100		
	Baseline	DK-CNN	DK-CNN(OF)
CNN4	0.52M	0.53M	0.54M
ResNet20	0.28M	0.27M	0.28M
ResNet32	0.47M	0.46M	0.48M
DenseNet40	0.19M	0.19M	0.19M
DenseNet52	0.28M	0.28M	0.27M

Table 10: The numbers of parameters in each model on CIFAR-100.

turtle, the distributions of the maximum feature values are different on the x-axis, namely, the latent dimension. This finding indicates that the latent dimension is effective for distinguishing classes. More examples are shown in the Appendix A.

Note that the results on Fashion-MNIST are different from those presented by Zhong et al. [39]. The various versions of the dataset account for this discrepancy. In the earlier versions of this dataset, overlapping images existed between the training and test sets. Random erasing is still an effective method, but the accuracy shown herein should be slightly lower than that reported in Zhong et al.'s work.

5. Conclusion

We proposed DK-CNNs, which represent an extension of the convolution operation to a latent dimension without increasing or reducing the number of parameters. Experiments were performed, and the results demonstrate that DK-CNNs can achieve better performance than regular CNNs; moreover, the convolution operation can be effectively extended into a parameter-related space. Additionally, since the new extended dimension does not exist in the data, the proposed method is not a task-dependent technique.

The proposed DK-CNN used in this work was a particular example of a CNN. In the future, we will use functions other than the sine function; alternatively, we will choose another approach to construct the dependencies between the kernel parameters and the latent variable. In addition, DK-CNNs can be extended to any model that can build dependencies between the latent variable and the parameters. Therefore, it will be interesting to apply DK-CNNs to fully connected networks for nonimage datasets. Furthermore, it will be very interesting to add more comparisons with regular CNNs structures, e.g. the series of MobileNet and InceptionNet, to demonstrate the significance of the proposed work.

Acknowledgements

The authors are very grateful to the anonymous reviewers for their constructive comments which have helped significantly in revising this work. This work was supported by the National Natural Science Foundation of China (No. 61673322, 61673326, and 91746103), the Fundamental Research Funds for the Central Universities (No. 20720190142), and the European

Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No. 663830.

[1] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in neural information processing systems, 2012, pp. 1097–1105.

[2] G. Ciaparrone, F. L. Sánchez, S. Tabik, L. Troiano, R. Tagliaferri, F. Herrera, Deep learning in video multi-object tracking: A survey, in: Neurocomputing, Elsevier, 2019.

[3] Y. Zhao, K. Hao, H. He, X. Tang, B. Wei, A visual long-short-term memory based integrated cnn model for fabric defect image classification, in: Neurocomputing, Elsevier, 2019.

[4] H. Ding, Z. Pan, Q. Cen, Y. Li, S. Chen, Multi-scale fully convolutional network for gland segmentation using three-class classification, in: Neurocomputing, Elsevier, 2019.

[5] B. Wang, Y. Lei, T. Yan, N. Li, L. Guo, Recurrent convolutional neural network: A new framework for remaining useful life prediction of machinery, in: Neurocomputing, Elsevier, 2019.

[6] J. Qin, Y. Huang, W. Wen, Multi-scale feature fusion residual network for single image super-resolution, in: Neurocomputing, Elsevier, 2019.

[7] S. Ramírez-Gallego, B. Krawczyk, S. García, M. Woźniak, F. Herrera, A survey on data preprocessing for data stream mining: Current status and future directions, in: Neurocomputing, Vol. 239, Elsevier, 2017, pp. 39–57.

[8] A. Roy, R. M. Cruz, R. Sabourin, G. D. Cavalcanti, A study on combining dynamic selection and data preprocessing for imbalance learning, in: Neurocomputing, Vol. 286, Elsevier, 2018, pp. 179–192.

[9] I. Cordón, J. Luengo, S. García, F. Herrera, F. Charte, Smartdata: data preprocessing to achieve smart data in r, in: Neurocomputing, Elsevier, 2019.

[10] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, in: arXiv preprint arXiv:1409.1556, 2014.

[11] H. Zhang, M. Cisse, Y. N. Dauphin, D. Lopez-Paz, mixup: Beyond empirical risk minimization, in: arXiv preprint arXiv:1710.09412, 2017.

[12] S. Dieleman, K. W. Willett, J. Dambre, Rotation-invariant convolutional neural networks for galaxy morphology prediction, in: Monthly notices of the royal astronomical society, Vol. 450, Oxford University Press, 2015, pp. 1441–1459.

[13] T. Cohen, M. Welling, Group equivariant convolutional networks, in: International conference on machine learning, 2016.

[14] Bruna, Joan, Mallat, Stephane, Invariant scattering convolution networks, IEEE Transactions on Pattern Analysis and Machine Intelligence 35 (8) 1872–1886.

[15] A. Krizhevsky, G. Hinton, Learning multiple layers of features from tiny images, Tech. rep., Citeseer (2009).

[16] H. Xiao, K. Rasul, R. Vollgraf, Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, in: arXiv preprint arXiv:1708.07747, 2017.

[17] J.-H. Jacobsen, J. van Gemert, Z. Lou, A. W. M. Smeulders, Structured receptive fields in cnns, in: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[18] T.-H. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, Y. Ma, Pcanet: A simple deep learning baseline for image classification?, IEEE transactions on image processing 24 (12) (2015) 5017–5032.

[19] J. Wu, S. Qiu, Y. Kong, L. Jiang, Y. Chen, W. Yang, L. Senhadji, H. Shu, Pcanet: An energy perspective, Neurocomputing 313 (2018) 271–287.

[20] M. Zhang, S. Khan, H. Yan, Deep eigen-filters for face recognition: Feature representation via unsupervised multi-structure filter learning, Pattern Recognition 100 (2020) 107176.

[21] S. Dieleman, J. De Fauw, K. Kavukcuoglu, Exploiting cyclic symmetry in convolutional neural networks, in: arXiv preprint arXiv:1602.02660, 2016.

[22] R. Gens, P. M. Domingos, Deep symmetry networks, in: Advances in neural information processing systems, 2014, pp. 2537–2545.

[23] E. Hoogeboom, J. W. Peters, T. S. Cohen, M. Welling, Hexaconv, in: arXiv preprint arXiv:1803.02108, 2018.

[24] T. S. Cohen, M. Geiger, J. Köhler, M. Welling, Spherical cnns, in: arXiv preprint arXiv:1801.10130, 2018.

[25] F. Yu, V. Koltun, Multi-scale context aggregation by dilated convolutions, in: arXiv preprint arXiv:1511.07122, 2015.

[26] W. Luo, Y. Li, R. Urtasun, R. S. Zemel, Understanding the effective receptive field in deep convolutional neural networks, in: Advances in Neural Information Processing Systems, 2016.

[27] Y. Jeon, J. Kim, Active convolution: Learning the shape of convolution for image classification, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 4201–4209.

[28] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, Y. Wei, Deformable convolutional networks, in: 2017 IEEE International Conference on Computer Vision (ICCV), 2017, pp. 764–773.

[29] K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, in: 2015 IEEE International Conference on Computer Vision (ICCV), 2015, pp. 1026–1034.

[30] X. Glorot, A. Bordes, Y. Bengio, Deep sparse rectifier neural networks, in: Proceedings of the fourteenth international conference on artificial intelligence and statistics, 2011, pp. 315–323.

[31] T. Salimans, D. P. Kingma, Weight normalization: A simple reparameterization to accelerate training of deep neural networks, in: Advances in Neural Information Processing Systems, 2016, pp. 901–909.

[32] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[33] G. Huang, Z. Liu, L. Van Der Maaten, K. Q. Weinberger, Densely connected convolutional networks, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 4700–4708.

[34] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Proceedings of the thirteenth international conference on artificial intelligence and statistics, 2010, pp. 249–256.

[35] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, Y. Bengio, Identifying and attacking the saddle point problem in high-dimensional non-convex optimization, in: Advances in neural information processing systems, 2014, pp. 2933–2941.

[36] J. Martens, Deep learning via hessian-free optimization., in: ICML, Vol. 27, 2010, pp. 735–742.

[37] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell, Caffe: Convolutional architecture for fast feature embedding, in: MM 2014 - Proceedings of the 2014 ACM Conference on Multimedia, 2014. doi:10.1145/2647868.2654889.

[38] S. Zagoruyko, N. Komodakis, Wide residual networks, in: CoRR, Vol. abs/1605.07146, 2016.

[39] Z. Zhong, L. Zheng, G. Kang, S. Li, Y. Yang, Random erasing data augmentation, in: arXiv preprint arXiv:1708.04896, 2017.

[40] I. Sutskever, J. Martens, G. Dahl, G. Hinton, On the importance of initialization and momentum in deep learning, in: International conference on machine learning, 2013, pp. 1139–1147.

[41] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, in: arXiv preprint arXiv:1412.6980, 2014.

Appendix A.

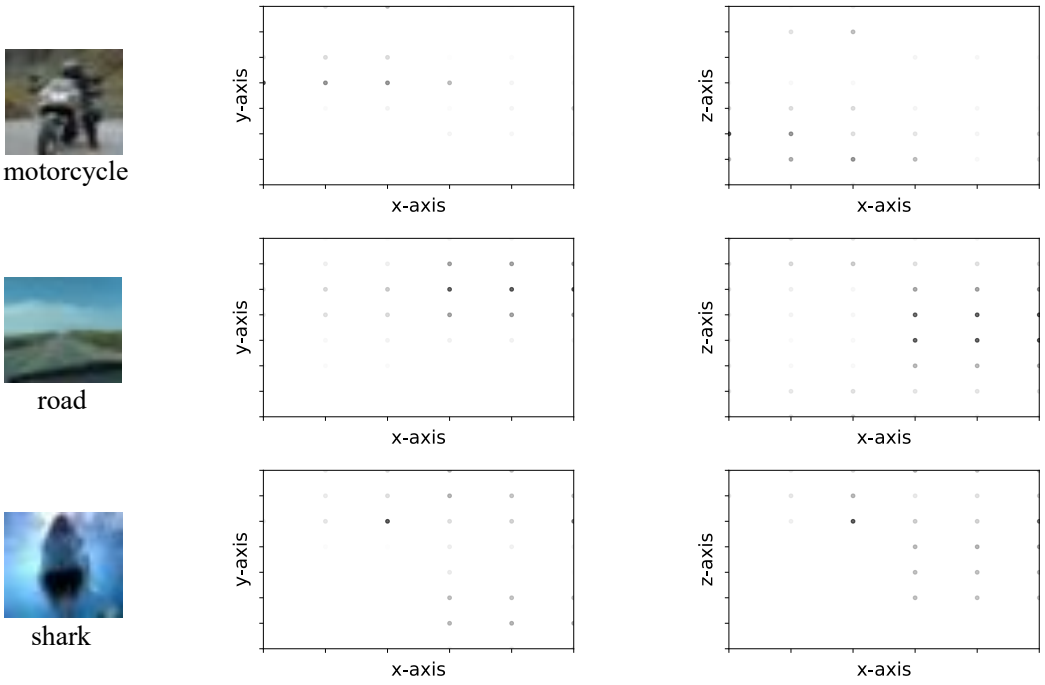

Figure A.7: Examples of 3D feature in DK-CNNs. These are the output features of the last DK-CNNs layer in ResNet20. The pictures corresponding to 3D features on the left come from CIFAR100 [15]. We project 3D features to 2D and show the projections on the right. X-axis is the latent dimension of kernel weights. For each image, we only show the feature map of one channel, and the maximum and minimum of each channel correspond to the deepest and lightest colors.

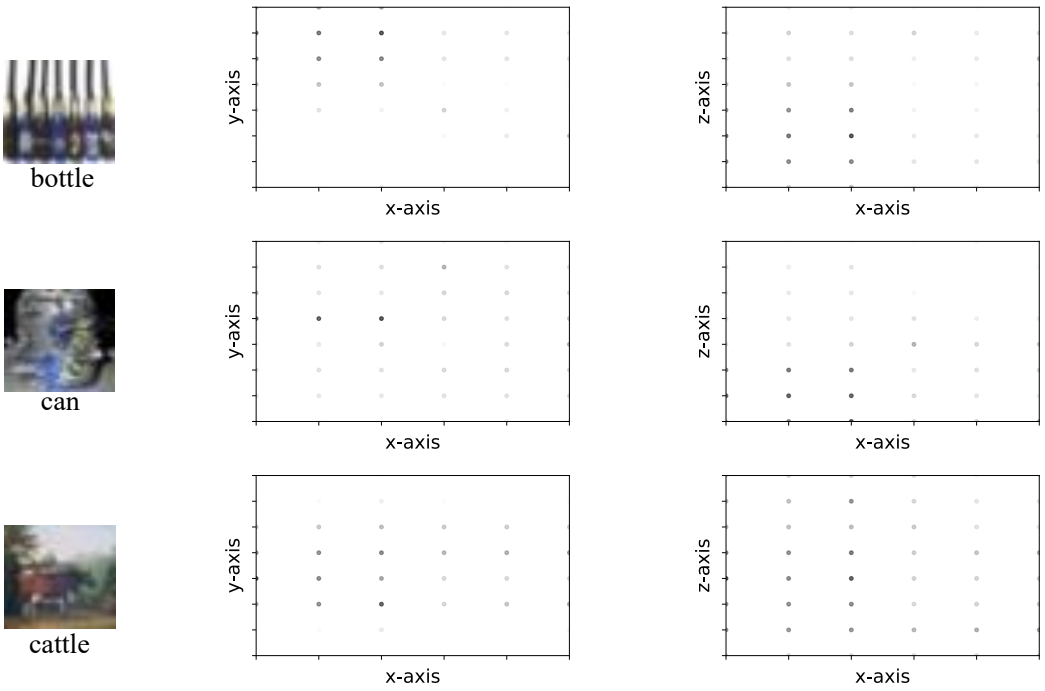

Figure A.8: Examples of 3D feature in DK-CNNs. These are the output features of the last DK-CNNs layer in ResNet20. The pictures corresponding to 3D features on the left come from CIFAR100 [15]. We project 3D features to 2D and show the projections on the right. X-axis is the latent dimension of kernel weights. For each image, we only show the feature map of one channel, and the maximum and minimum of each channel correspond to the deepest and lightest colors.

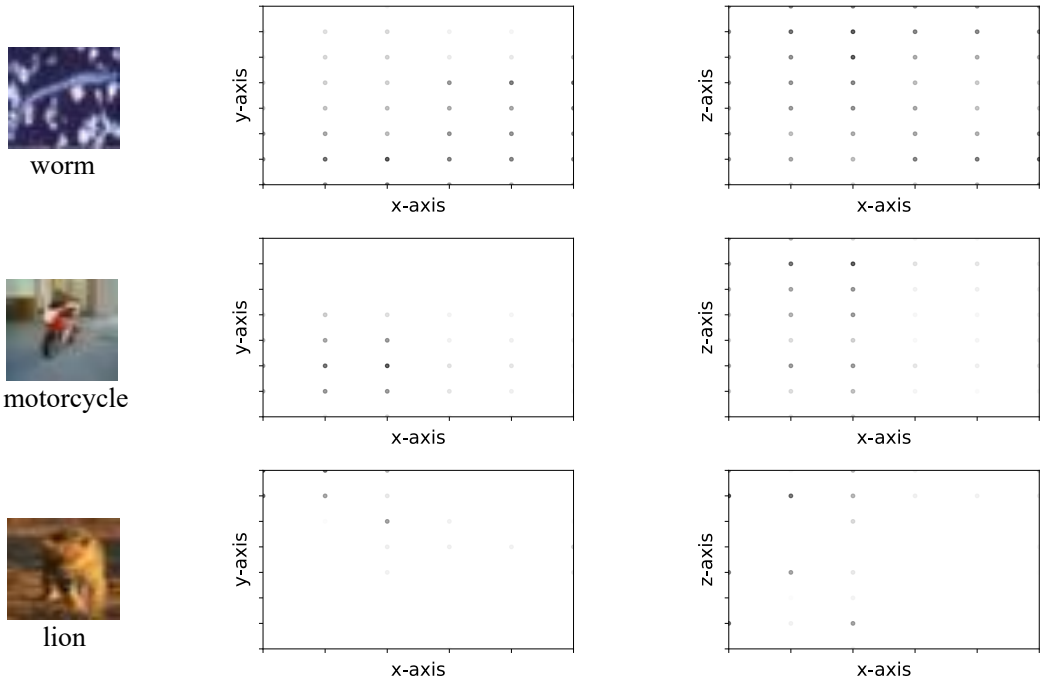

Figure A.9: Examples of 3D feature in DK-CNNs. These are the output features of the last DK-CNNs layer in ResNet20. The pictures corresponding to 3D features on the left come from CIFAR100 [15]. We project 3D features to 2D and show the projections on the right. X-axis is the latent dimension of kernel weights. For each image, we only show the feature map of one channel, and the maximum and minimum of each channel correspond to the deepest and lightest colors.

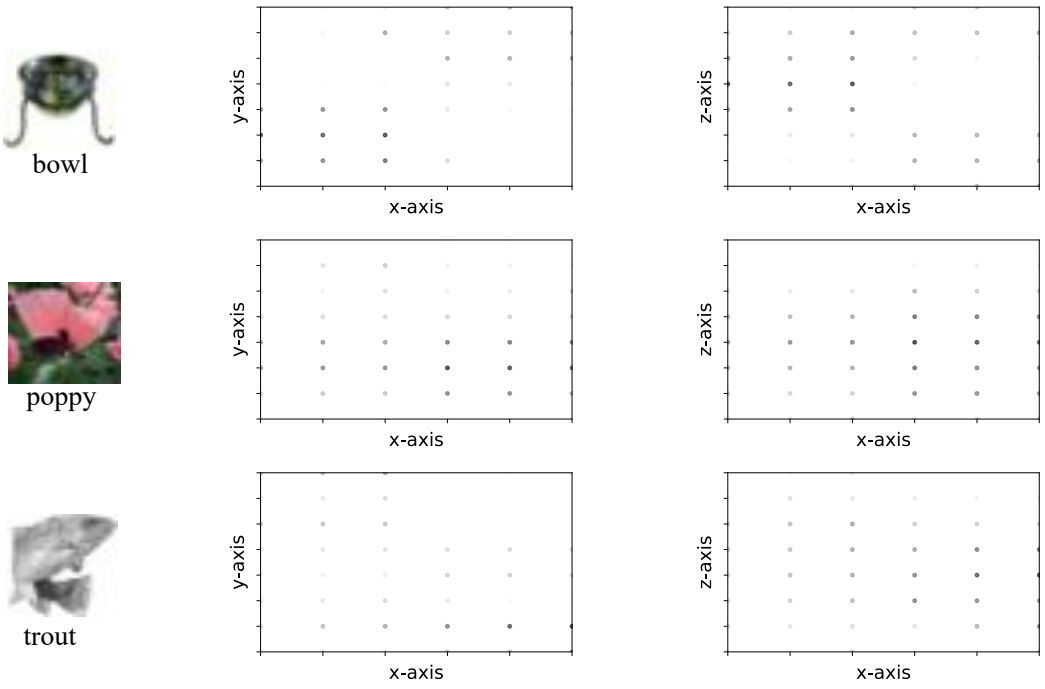

Figure A.10: Examples of 3D feature in DK-CNNs. These are the output features of the last DK-CNNs layer in ResNet20. The pictures corresponding to 3D features on the left come from CIFAR100 [15]. We project 3D features to 2D and show the projections on the right. X-axis is the latent dimension of kernel weights. For each image, we only show the feature map of one channel, and the maximum and minimum of each channel correspond to the deepest and lightest colors.

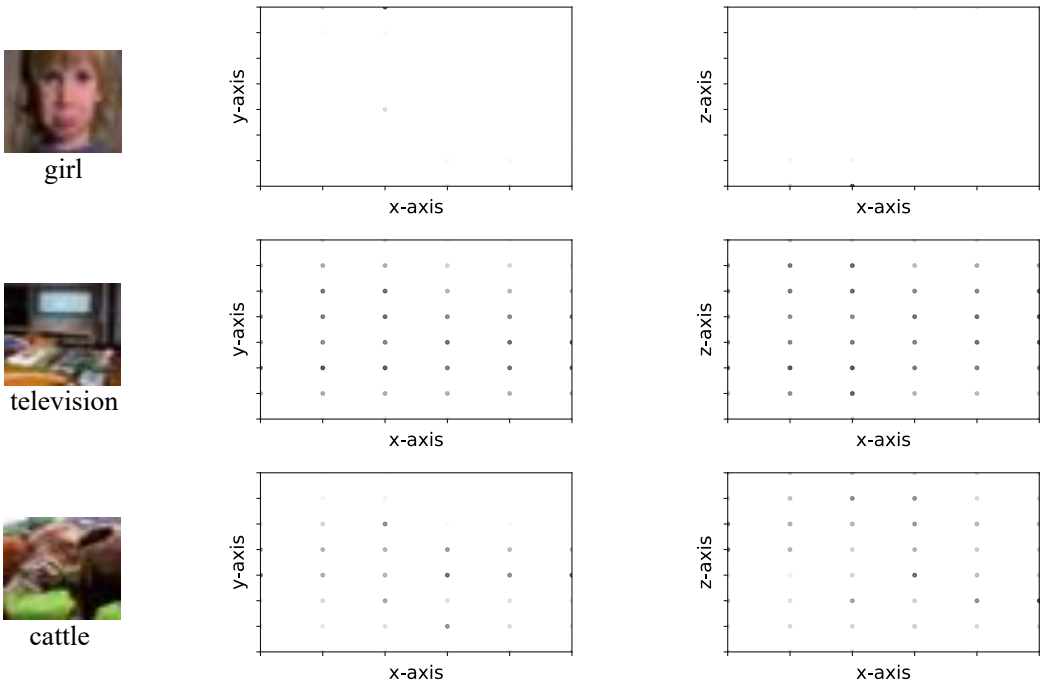

Figure A.11: Examples of 3D feature in DK-CNNs. These are the output features of the last DK-CNNs layer in ResNet20. The pictures corresponding to 3D features on the left come from CIFAR100 [15]. We project 3D features to 2D and show the projections on the right. X-axis is the latent dimension of kernel weights. For each image, we only show the feature map of one channel, and the maximum and minimum of each channel correspond to the deepest and lightest colors.

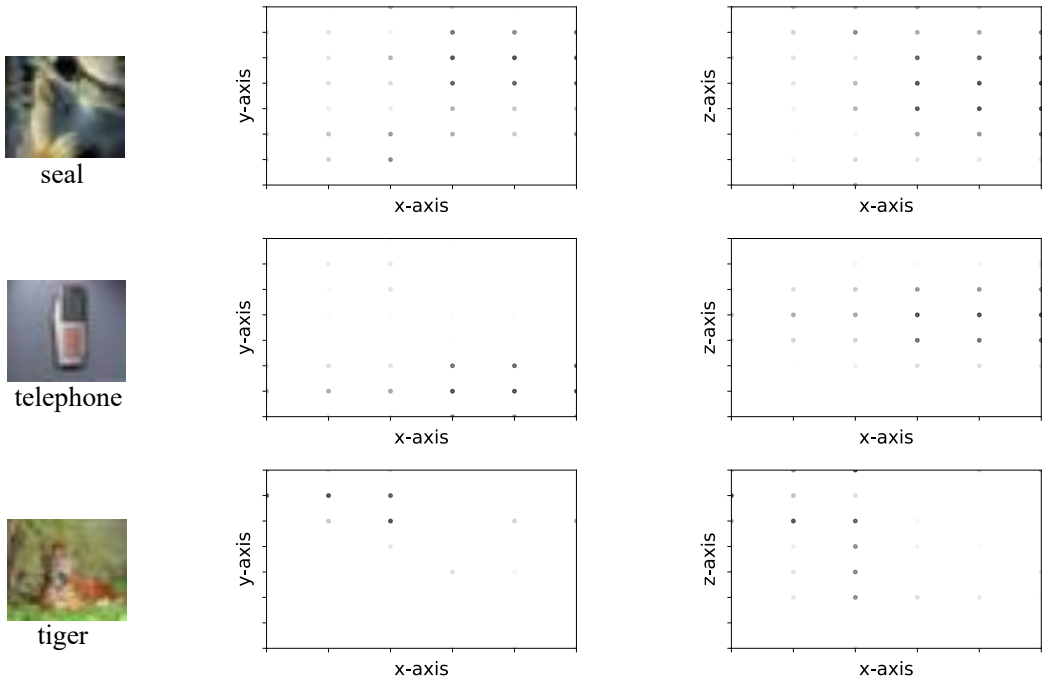

Figure A.12: Examples of 3D feature in DK-CNNs. These are the output features of the last DK-CNNs layer in ResNet20. The pictures corresponding to 3D features on the left come from CIFAR100 [15]. We project 3D features to 2D and show the projections on the right. X-axis is the latent dimension of kernel weights. For each image, we only show the feature map of one channel, and the maximum and minimum of each channel correspond to the deepest and lightest colors.