Learning Additive Kernel For Feature Transformation and Its Application to CNN Features

Takumi Kobayashi
takumi.kobayashi@aist.go.jp

National Institute of Advanced Industrial Science and Technology
Tsukuba, Japan

Abstract

Feature transformation is an important process following feature extraction for improving classification performance. It has been frequently addressed in the kernel-based framework utilizing non-linear kernel functions, and the additive kernel equipped with explicit feature mapping works as efficient and effective (non-linear) feature transformation. The kernel functions, however, are defined in a top-down manner taking into account the inherent nature of the features, which makes it difficult to appropriately apply them to the features whose characteristics are not fully disclosed, such as CNN features. In this paper, we propose a method to learn an additive kernel of which explicit mapping serves feature transformation. By means of a bottom-up learning approach leveraging annotated data, the proposed method builds the kernel function of high generality and discriminative power even for the CNN features. The experiments on various datasets using various types of pre-trained CNN features show favorable performance improvement by the learned additive kernel (feature transformation) of which generality over the datasets and the CNN models is also demonstrated.

1 Introduction

Extracting features is required for effectively representing images and videos in pattern recognition systems such as on classification. While the methods of BoW [3] and Fisher kernel [26] have been successfully applied to feature representation so far, in recent years, the convolutional neural network (CNN) of deep architecture is enthusiastically studied in this literature [11, 30] and it is empirically shown that the pre-trained CNN also works as a generic feature extractor with great success [5, 19, 24]. Though it is possible to directly feed the extracted features into classifiers, we usually transform the features on the basis of their intrinsic characteristics so as to favorably improve performance.

L_2 normalization is one of the simplest transformation, widely applied along with linear classifiers. Beyond the L_2 normalization, the feature transformation is also addressed in the kernel-based framework [27] utilizing kernel functions to exploit non-linear structure embedded in the features; from this viewpoint, features are non-linearly transformed into a Hilbert space in an implicit manner via the kernel function. While the RBF kernel is generally applied to feature vectors, it is advantageous to apply the kernel functions based on the intrinsic nature of the features; for example, χ^2 kernel [32], intersection kernel [14] and

Dirichlet Fisher kernel [9] are effective for histogram (BoW) features, and Hellinger kernel (square root) [2, 26] successfully works in Fisher kernel feature representation. However, the non-linear kernel function which implicitly transforms features leads to the kernel-based classifier requiring substantial computation cost both in training and test [27], which prevents the application to large-scale samples. Additive kernels remedy the problem by providing *explicit* feature mapping [32] which serves feature transformation. The additive kernel consists of component-wise kernel functions which can be well approximated by the inner product of a finite dimensional explicit mapping, and thereby linear classifiers are efficiently applicable to so transformed features [32]; the kernels listed above except for RBF belong to a family of the additive kernels. Thus, through the feature transformation via the additive kernel, especially its explicit mapping, we can leverage the discriminative power embedded in the non-linear kernel function in an explicit form of finite dimension while keeping computational efficiency.

However, since the additive kernels equipped with explicit mapping are pre-defined in a top-down manner taking into account the nature of the features, it is difficult to determine which types of additive kernel function should be applied to the features whose characteristics are not fully disclosed, and we do not know how the pre-defined kernels work for such features at all. In this paper, we propose a method to learn the additive kernel of high generality and discriminative power in a bottom-up manner based on actual (annotated) data. The bottom-up learning approach endows the kernel function with discriminative power, adapting it even to the features of unknown characteristics; this is the case of CNN features [11, 28, 30] which are of our main interest in this paper, though the enthusiastic studies are now underway to explore the CNN features' nature [15, 18]. Through such data-driven learning process, however, the learned kernel function is prone to over-fitting to the dataset that is used for training, deteriorating generalization performance. We construct data-driven yet *generic* kernel functions by harnessing the simple formulation of additive kernels in which the kernel function operates on a pair of scalar feature component. The explicit mapping of the learned additive kernel serves as transforming features, especially pre-trained CNN features in this work.

2 Proposed Method

We begin with a brief review of additive kernels and then detail the proposed method to learn the additive kernel based on data. In contrast to the top-down additive kernel method [32], we first establish the explicit mapping which consequently shapes the additive kernel function.

2.1 Additive Kernel

Given a pair of feature vectors, $\boldsymbol{x}$ and $\boldsymbol{y} \in \mathbb{R}^D$, the additive kernel is defined as follows,

$$\bar{\mathtt{k}}(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{D} \mathtt{k}(x_i, y_i), \tag{1}$$

where x_i and y_i are the i-th elements of $\boldsymbol{x}$ and $\boldsymbol{y}$, respectively. In this formulation, the positive definite kernel function $\mathtt{k}$ operates on each feature component and the responses are summed up for the additive kernel response $\bar{\mathtt{k}}$; thus the additive kernel is characterized by the kernel function $\mathtt{k}$. A family of the additive kernel includes the simplest *linear* one $\mathtt{k}(x_i, y_i) = x_i y_i$ and Hellinger kernel $\mathtt{k}(x_i, y_i) = \mathrm{sgn}(x_i)\mathrm{sgn}(y_i)\sqrt{|x_i y_i|}$ [2] as well as χ^2 kernel $\mathtt{k}(x_i, y_i) =$

$\frac{x_i y_i}{x_i+y_i}$ [32] and intersection kernel $\mathtt{k}(x_i,y_i)=\min(x_i,y_i)$ [16] for (non-negative) histogram features.

And, the kernel function $\mathtt{k}$ is represented by the following decomposed form for achieving the positive definiteness,

$$\mathtt{k}(x,y)=\int \phi(\gamma;x)\phi(\gamma;y)d\lambda \approx \sum_{l=1}^{K}\phi(\gamma_l;x)\phi(\gamma_l;y)\Delta_l=\sum_{l=1}^{K}\hat{\phi}(\gamma_l;x)\hat{\phi}(\gamma_l;y)=\boldsymbol{\phi}(x)^\top\boldsymbol{\phi}(y), \tag{2}$$

where $\boldsymbol{\phi}(x)=[\hat{\phi}(\gamma_1;x),\cdots,\hat{\phi}(\gamma_K;x)]^\top\in\mathbb{R}^K$ approximates the (continuous) function $\phi(\gamma;x)$ using K control points, which we call K-rank approximation. Given the function ϕ and the control points $\{\gamma_l\}_{l=1}^K$, the explicit mapping of the non-linear additive kernel $\bar{\mathtt{k}}$ is described by $[\boldsymbol{\phi}(x_1)^\top,\cdots,\boldsymbol{\phi}(x_D)^\top]^\top$; in [32], those two ingredients ϕ and γ_l are theoretically provided even for χ^2 kernel to realize the explicit feature mapping.

2.2 Learning Kernel Function

Our objective is to learn the non-linear functions $\boldsymbol{\phi}$ based on data. Since it is quite difficult to built $\boldsymbol{\phi}$ from scratch, we resort to the approximated representation of $\boldsymbol{\phi}$ by using basis functions in a manner similar to Fourier expansion. Suppose we have M basis functions $\mathtt{f}_m, m=1,\cdots,M$, then the non-linear functions $\boldsymbol{\phi}$ can be represented by

$$\boldsymbol{\phi}(x)=\boldsymbol{W}^\top[\mathtt{f}_1(x),\cdots,\mathtt{f}_M(x)]^\top=\boldsymbol{W}^\top\mathbf{f}(x), \tag{3}$$

where $\boldsymbol{W}\in\mathbb{R}^{M\times K}$ is the coefficient matrix for the basis functions and $\mathbf{f}(x)\in\mathbb{R}^M$ is the vector composed of the basis function outputs at x. Thereby the kernel function is described as

$$\mathtt{k}(x,y)=\mathbf{f}(x)^\top\boldsymbol{W}\boldsymbol{W}^\top\mathbf{f}(y),\ \bar{\mathtt{k}}(\boldsymbol{x},\boldsymbol{y})=\sum_{i=1}^{D}\mathbf{f}(x_i)^\top\boldsymbol{W}\boldsymbol{W}^\top\mathbf{f}(y_i)=\mathtt{tr}\{\mathbf{F}(\boldsymbol{x})^\top\boldsymbol{W}\boldsymbol{W}^\top\mathbf{F}(\boldsymbol{y})\}, \tag{4}$$

where $\mathtt{tr}$ is the operator of matrix trace and

$$\mathbf{F}(\boldsymbol{x})=[\mathbf{f}(x_1),\cdots,\mathbf{f}(x_D)]\in\mathbb{R}^{M\times D}. \tag{5}$$

The coefficient matrix $\boldsymbol{W}$ defines the additive kernel function as well as its explicit mapping, $\boldsymbol{W}^\top\mathbf{F}(\boldsymbol{x})\in\mathbb{R}^{K\times D}$. Therefore, discriminative learning of $\boldsymbol{\phi}$ is reduced into optimizing $\boldsymbol{W}$ in a supervised framework using annotated data given basis functions $\{\mathtt{f}_m\}_{m=1}^M$.

2.2.1 Optimization

We can find that $\bar{\mathtt{k}}$ in (4) is regarded as just the inner product of the *matrix* features $\boldsymbol{W}^\top\mathbf{F}(\boldsymbol{x})$, and thus the classifier using the kernel (4) is eventually described by the following form:

$$\mathtt{tr}\{\boldsymbol{W}^\top\mathbf{F}(\boldsymbol{x})\boldsymbol{A}\}+b, \tag{6}$$

where $\boldsymbol{A}\in\mathbb{R}^{D\times K}$ is the classifier weight matrix and b is a classifier bias. From the viewpoint that both $\boldsymbol{A}$ and $\boldsymbol{W}$ are optimized, the classifier (6) is a bilinear form [10] regarding $\boldsymbol{A}$ and $\boldsymbol{W}$ which are column and row weights for the feature matrix $\mathbf{F}(\boldsymbol{x})$. Note that the classifier weight $\boldsymbol{A}$ is dependent on classification tasks/datasets while the coefficient $\boldsymbol{W}$ for basis functions should be general relying only on the type of feature $\boldsymbol{x}$. In order to enhance generality of the coefficient $\boldsymbol{W}$, it is necessary to learn it on larger-scaled and a variety of data as much as possible. However, the methods [10, 21] which can optimize the bilinear model (6) are neither well scaled due to solving SVM-based quadratic programming substantial times nor suitable for improving generality of $\boldsymbol{W}$ by aggressively minimizing the rank K of $\boldsymbol{W}$ on the particular dataset that is used for learning.

We apply an efficient approach to learn $\boldsymbol{W}$, providing a good trade-off between the generality and discriminativity. Suppose we have a dataset $\mathcal{D}=\{\boldsymbol{x}_j,z_j\}_{j=1}^{N_\mathcal{D}}$ of $N_\mathcal{D}$ samples

comprising a feature vector $\boldsymbol{x}_j$ and a label z_j for a specific task; on classification, the label indicates membership to a certain class, $z_j \in \{-1,+1\}$. By integrating the weights $\boldsymbol{A}_{\mathcal{D}}$[1] and $\boldsymbol{W}$ into $\boldsymbol{V}_{\mathcal{D}} = \boldsymbol{W}\boldsymbol{A}_{\mathcal{D}}^{\top} \in \mathbb{R}^{M\times D}$, the classifier (6) is accordingly rewritten into

$$z = \mathtt{tr}\{\boldsymbol{V}_{\mathcal{D}}^{\top}\mathbf{F}(\boldsymbol{x})\} + b, \tag{7}$$

and thereby we can obtain the standard (linear) SVM optimization problem [31] for $\boldsymbol{V}_{\mathcal{D}}$,

$$\min_{\boldsymbol{V}_{\mathcal{D}},b} \frac{1}{2}\mathtt{tr}(\boldsymbol{V}_{\mathcal{D}}^{\top}\boldsymbol{V}_{\mathcal{D}}) + \sum_{j=1}^{N_{\mathcal{D}}} \max[0, 1 - z_j\{\mathtt{tr}(\boldsymbol{V}_{\mathcal{D}}^{\top}\mathbf{F}(\boldsymbol{x}_j)) + b\}]. \tag{8}$$

The optimizer of (8) is denoted by $\boldsymbol{V}_{\mathcal{D}}^{*} = \boldsymbol{W}^{*}\boldsymbol{A}_{\mathcal{D}}^{*\top}$ where only $\boldsymbol{A}_{\mathcal{D}}^{*}$ depends on the dataset $\mathcal{D}$.

For enhancing the generality of $\boldsymbol{W}$, we prepare various datasets $\{\mathcal{D}_c\}_{c=1}^{C}$ such as by decomposing a dataset of multi-class tasks in a one-vs-rest manner regarding classes as well as collecting datasets, *e.g.*, MIT67 [22] and VOC2007 [1]. Solving (8) in the respective datasets produces multiple optimizers $\{\boldsymbol{V}_{\mathcal{D}_c}^{*}\}_{c=1}^{C}$ which share $\boldsymbol{W}^{*}$ and are unified into

$$\boldsymbol{V}^{*} = [\boldsymbol{V}_{\mathcal{D}_1}^{*}, \cdots \boldsymbol{V}_{\mathcal{D}_C}^{*}] = \boldsymbol{W}^{*}[\boldsymbol{A}_{\mathcal{D}_1}^{*\top}, \cdots, \boldsymbol{A}_{\mathcal{D}_C}^{*\top}] \in \mathbb{R}^{M\times CD}. \tag{9}$$

This shows that the coefficient matrix $\boldsymbol{W}^{*}$ can be retrieved by decomposing the unified classifier $\boldsymbol{V}^{*}$. We decompose $\boldsymbol{V}^{*}$ by means of singular value decomposition (SVD);

$$\boldsymbol{V}^{*} = \boldsymbol{P}\boldsymbol{\Lambda}\boldsymbol{Q}^{\top}, \tag{10}$$

where the orthonormal matrix $\boldsymbol{P} = [\boldsymbol{p}_1, \cdots, \boldsymbol{p}_M] \in \mathbb{R}^{M\times M}$ is related to $\boldsymbol{W}^{*}$ and $\boldsymbol{\Lambda}$ is the diagonal matrix of singular values $\{\lambda_i\}_{i=1}^{M}$ where $i < j \Rightarrow \lambda_i \geq \lambda_j$ (decreasing order). Even for the unified classifier $\boldsymbol{V}^{*}$ of large column size CD, we can efficiently compute both $\boldsymbol{P}$ and $\boldsymbol{\Lambda}$ by the following eigen decomposition,

$$\boldsymbol{V}^{*}\boldsymbol{V}^{*\top}\boldsymbol{P} = \left(\sum_{c=1}^{C} \boldsymbol{V}_{\mathcal{D}_c}^{*}\boldsymbol{V}_{\mathcal{D}_c}^{*\top}\right)\boldsymbol{P} = \boldsymbol{P}\boldsymbol{\Lambda}^2, \tag{11}$$

where $\boldsymbol{V}_{\mathcal{D}_c}^{*}\boldsymbol{V}_{\mathcal{D}_c}^{*\top}$ is a matrix of small size $M \times M$. Finally, from the viewpoint of minimizing the cost $\mathtt{tr}(\boldsymbol{A}\boldsymbol{A}^{\top} + \boldsymbol{W}\boldsymbol{W}^{\top})$ used in bilinear optimization [10], the optimum $\boldsymbol{W}^{*}$ is obtained by

$$\boldsymbol{W}^{*} = \boldsymbol{P}_K\boldsymbol{\Lambda}_K^{\frac{1}{2}}, \tag{12}$$

where $\boldsymbol{P}_K = [\boldsymbol{p}_1, \cdots, \boldsymbol{p}_K]$ and $\boldsymbol{\Lambda}_K^{\frac{1}{2}} = \mathtt{diag}(\sqrt{\lambda_1}, \cdots, \sqrt{\lambda_K})$. The rank K could be determined based on the contributing rate $\tau = \frac{\sum_{k=1}^{K}\lambda_k^2}{\sum_{k=1}^{M}\lambda_k^2}$ in the decomposition (11). Note again that the feature vector $\boldsymbol{x}$ is transformed into $\boldsymbol{W}^{*\top}\mathbf{F}(\boldsymbol{x})$[2] which is then unfolded into a vector.

The proposed method is related to the multiple kernel learning (MKL) [13, 23] by regarding each of the basis function as a basis kernel function. However, the MKL optimizes only M weights for the basis kernels corresponding to the diagonal weights in $\boldsymbol{W}$ which significantly degrades capability to describe the additive kernel. In contrast, the proposed method optimizes the full weight matrix $\boldsymbol{W}$ which can represent various types of additive kernel.

2.2.2 Basis Function

From the perspective of approximating the continuous functions $\boldsymbol{\phi}$, it is natural to employ Fourier-based functions as bases. In addition, we can make use of the following generic prior knowledge on features. The feature values of larger magnitude convey distinct information

[1] We add the subscript $\mathcal{D}$ to the weight $\boldsymbol{A}$ in order to emphasize its dependency on the dataset $\mathcal{D}$, while $\boldsymbol{W}$ is not.
[2] For faster computation, we constructed look-up tables by pre-computing the transformed values.

about the target while those of lower (or zero) magnitude provide less (or no) information; this would also be the case of the CNN features. By incorporating the prior knowledge into the Fourier functions, we determine the basis functions $\mathtt{f}$ as

$$\mathtt{f}_{2m-1}(x) = x\cos(2\pi\eta_m x), \quad \mathtt{f}_{2m}(x) = x\sin(2\pi\eta_m x), \tag{13}$$

where η_m are the frequency parameters; in this study, we set $\eta \in \{0, 0.1, \cdots, 0.9, 1, \cdots, 10\}$[3] to produce $M = 39$ basis functions. These basis functions reflect the significance of the feature value x, which contributes to efficiently approximate $\boldsymbol{\phi}$ by using a fewer number of bases; in particular, $x = 0$ is always mapped into zeros. Note that the feature vectors $\boldsymbol{x}$ are normalized in a unit L_1 norm in advance, $\boldsymbol{x} \leftarrow \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|_1}$, for effectively bounding the range of feature values in $[0, 1]$ without loss of generality[4].

3 Experimental Results

We apply the proposed method to transform the pre-trained CNN features. The CNN methods have exhibited excellent performance on image classifications in the last five years [11, 28] and recently applied to extract spatio-temporal features from videos [30]. As in [6], we employ as feature extractors the CNNs pre-trained on the large-scale datasets. We focus on three types of widely used CNN models, `C3D` [30], `Alex` [11] and `VGG` [28]; `C3D` is trained on sports-1M dataset [7] for extracting motion features, while `Alex` and `VGG` are trained on ImageNet dataset [4] for extracting image features. In these CNNs, the outputs of the first fully connected layer (`fc6`) are diverted to features of $D = 4096$ dimension; for detailed CNN architectures, refer to the respective papers.

In classification, all the features with/without feature transformation are normalized in a unit L_2 norm and subsequently the linear SVM classifier [31] is applied. We follow the evaluation protocol provided together with datasets; in CALTECH256 we randomly draw 60 training samples per category and use the rest for test, while in the other datasets the provided training/test splits are used.

3.1 Performance Analysis

We first analyze the proposed method from various aspects by using `C3D` features.

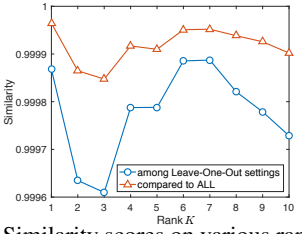
3.1.1 Generality of Learned Additive Kernel Across Datasets

The kernel function trained in a certain dataset is obviously applicable to the identical dataset, but it is unclear how the learned kernel function is applicable to the other datasets of different task and subjects. It is highly demanded to construct the *generic* additive kernel which can be applied to various types of data without re-training. Thus, we first investigate the generality of the learned additive kernel on `C3D` feature for action classification on the datasets of HMDB51 [12], HOLLYWOOD2 [17], UCF101 [29] and UCF50 [25].

We assess the generality by measuring similarity among the additive kernels trained on different datasets. For that purpose, the proposed method is applied to learn the additive kernel in a leave-'*one dataset*'-out manner; on the above-mentioned four datasets, we can

[3] In case of $\eta = 0$, the basis function is $\mathtt{f}(x) = x$.

[4] After applying L_1 normalization to CNN features of $D = 4096$ dimension, we find that the maximum feature value is around 0.005. For further fitting the feature value distribution to $[0, 1]$, we scale them as $\boldsymbol{x} \leftarrow \frac{200\boldsymbol{x}}{\|\boldsymbol{x}\|_1}$.



(a) Similarity scores on various ranks

	HMDB51	HOLLYWOOD2	UCF101	UCF50	ALL
HMDB51	1.0000	0.9999	0.9997	0.9999	1.0000
HOLLYWOOD2	0.9999	1.0000	0.9996	0.9998	0.9999
UCF101	0.9997	0.9996	1.0000	0.9999	0.9998
UCF50	0.9999	0.9998	0.9999	1.0000	1.0000
ALL	1.0000	0.9999	0.9998	1.0000	1.0000

NOTE: $\overline{\text{DATASET}}$ indicates the training datasets excluding DATASET, while ALL means all four datasets are used for training.

(b) Similarity scores among various training datasets at rank 3

Figure 1: Similarity scores among the additive kernels learned in various settings. In (a), blue and red lines show the 'minimum' similarity over cyan and magenta cells in (b), respectively.

obtain *four* learned kernels each of which is trained on the three datasets out of the four in a manner similar to leave-one-out scheme, and also learn *one* kernel by using all the four datasets. To measure the similarity, the kernel function $\mathtt{k}(x,y)$ is quantized into 256×256 matrix of look-up table by using 256×256 bins on the domain $(x,y) \in [0,1]^2$, and then we compute cosine similarity of the look-up table matrices. Fig. 1 shows the results on various ranks K of kernels; in this C3D feature, the contributing rate τ reaches close to 1 at the rank of $K = 10$. One can see that those learned kernels exhibit quite high similarity, being almost identical, even though they are trained on different datasets of enough size. Such generality is rendered by exploiting the shared component $\boldsymbol{W}$ from various classifier weights via SVD (10) . Thus, we can say that the proposed method produces the *generic* additive kernel and in what follows we apply the additive kernel that is trained by using all the four datasets.

3.1.2 Rank

There is only one parameter regarding the rank K in the proposed additive kernel. We empirically show the performance on various ranks in Fig. 2a as well as the contributing rate τ; it actually demonstrates the performance improvement gained by the proposed method, compared to that of the original L_2-normalized C3D feature. The performance is increased as the rank becomes higher and it is saturated around the rank $K = 5$ which well reconstructs the additive kernel with high contributing rate $\tau > 0.9$. It should be noted that even $K = 2$ successfully improves the performance.

3.1.3 Form of W

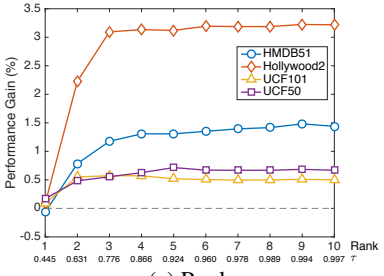
Next, we validate the form (12) of $\boldsymbol{W}$ in which the eigen vectors $\boldsymbol{P}$ is scaled according to $\boldsymbol{\Lambda}^{\frac{1}{2}}$. Any rotation matrix $\boldsymbol{R}$, *s.t.* $\boldsymbol{R}\boldsymbol{R}^\top = \boldsymbol{I}$, does not affect the kernel function since $\mathtt{k}'(x,y) = \mathbf{f}(x)^\top \boldsymbol{W}\boldsymbol{R}\boldsymbol{R}^\top \boldsymbol{W}^\top \mathbf{f}(y) = \mathbf{f}(x)\boldsymbol{W}\boldsymbol{W}^\top \mathbf{f}(y) = \mathtt{k}(x,y)$. Thus, the eigen vectors $\boldsymbol{P}$ and their scaling according to $\boldsymbol{\Lambda}$ are essential ingredients for the kernel function. We investigate how the scaling weight, actually the power order p in $\boldsymbol{W} = \boldsymbol{P}_K \boldsymbol{\Lambda}_K^p$, affects performance. The performance results are shown in Fig. 2b. Different scaling slightly degrades the performance and $p = \frac{1}{2}$ that minimizes $\mathtt{tr}(\boldsymbol{A}\boldsymbol{A}^\top + \boldsymbol{W}\boldsymbol{W}^\top)$ [□] produces the better performance on average. For the purpose of gaining generality, it is reasonable to employ the scaling of $p = \frac{1}{2}$.

3.1.4 Basis Functions

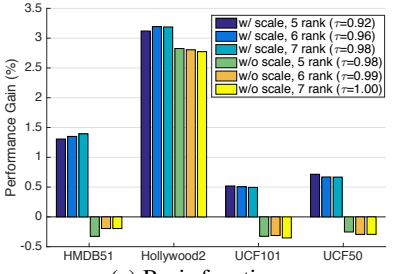
We further analyze the proposed method in terms of basis functions f. As described in Sec.2.2.2, we employ the scaled Fourier functions (13) as the bases to compose an additive

order p	rank $K=5$	HMDB 6	HMDB 7	HOLLYWOOD2 5	HOLLYWOOD2 6	HOLLYWOOD2 7	UCF101 5	UCF101 6	UCF101 7	UCF50 5	UCF50 6	UCF50 7
0	54.27	54.38	54.31	46.84	46.97	46.93	83.87	83.92	83.90	93.15	93.17	93.17
$\frac{1}{2}$	**54.40**	**54.44**	**54.49**	**46.93**	**47.00**	**46.99**	83.94	83.93	83.92	**93.32**	**93.28**	**93.28**
1	54.38	54.40	54.44	46.62	46.63	46.63	**84.04**	**84.03**	**84.03**	93.13	93.15	93.15

(b) Scaling (power order) in the form $\boldsymbol{W} = \boldsymbol{P}_K \boldsymbol{\Lambda}_K^p$. Classification accuracies (%) are shown.



(a) Rank



(c) Basis function

Figure 2: Performance analysis on (a) rank K, (b) scaling in $\boldsymbol{W}$ and (c) basis functions, where (a,b) show the performance improvement gained by the methods, compared to that of the original L_2-normalized C3D feature.

Table 1: Computation time in learning $\boldsymbol{W}$ (Sec.2.2). (a) the computation time in each process of the learning, and (b) details of the SVM learning process (8).

(a) Training processes

Process	Time (sec)
SVM (8)	116.448
SVD (10,11)	0.096
Total	116.544

(b) Details of SVM process (8)

Dataset	HMDB	HOLLYWOOD2	UCF101	UCF50
Num. of sample	6049	823	13320	6618
Num. of class	51	12	101	50
Time (sec)	23.051	2.503	64.492	26.402

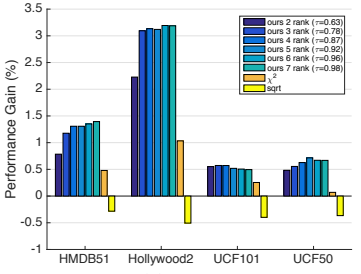
kernel function. For comparison, the simple Fourier functions without scaling, $\mathtt{f}_{2m-1}(x) = \cos(2\pi\eta_m x), \mathtt{f}_{2m}(x) = \sin(2\pi\eta_m x)$, are applied and the performance comparison is shown in Fig. 2c. The scaled Fourier bases outperform the simple ones without scaling which deteriorate performance on the most datasets. This result demonstrates the importance to incorporate the prior knowledge regarding the magnitude of the CNN feature value; the scaling factor effectively embeds such a prior knowledge into the basis functions.

3.1.5 Computation Time For Training

We also show in Table 1 the computation time for learning (Sec.2.2) in the proposed method. While the most time-consuming process is the SVM optimization (8), the SVD-based decomposition (10) performs with quite a small computation time via (11). It should be noted that the parallel processing is easily applied to the SVM optimization which individually operates on the respective datasets.

3.2 Performance Comparison With Other Methods

Next, we compare the proposed method to the other feature transformation methods, explicit map of χ^2 kernel [32] and Hellinger kernel (square root) transformation [2, 26], which are successfully applied to the hand-crafted features, such as BoW and Fisher kernel features; in this study the other *inefficient* kernels, *e.g.*, RBF, without having explicit maps are out of our focus. In addition to C3D features [30] on action classification, we apply the meth-



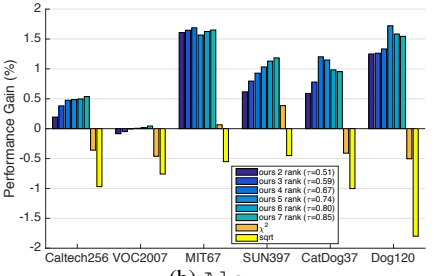
(a) C3D

C3D

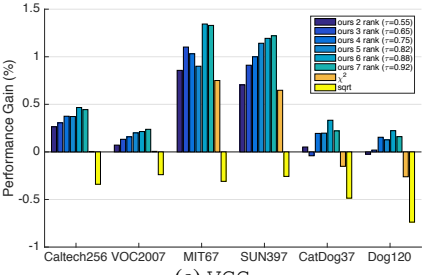
Dataset	Orig.	χ^2 [32]	sqrt [0]	Ours$_{6rank}$
HMDB51	53.09	53.57	52.81	**54.44**
HOLLYWOOD2	43.81	44.84	43.30	**47.00**
UCF101	83.42	83.67	83.02	**83.93**
UCF50	92.61	92.68	92.24	**93.28**

	Alex				VGG			
Dataset	Orig.	χ^2 [32]	sqrt [0]	Ours$_{6rank}$	Orig.	χ^2 [32]	sqrt [0]	Ours$_{6rank}$
CALTECH256	73.11	72.75	72.14	**73.61**	83.08	83.08	82.74	**83.55**
VOC2007	76.56	76.09	75.80	**76.57**	86.26	86.27	86.02	**86.48**
MIT67	62.99	63.06	62.44	**64.62**	74.37	75.12	74.06	**75.71**
SUN397	48.25	48.64	47.80	**49.38**	56.80	57.44	56.54	**57.99**
CATDOG37	80.83	80.42	79.83	**81.82**	89.83	89.68	89.35	**90.17**
DOG120	63.99	63.49	62.20	**65.58**	78.76	78.50	78.02	**78.98**

(d) Classification accuracies (%)



(b) Alex



(c) VGG

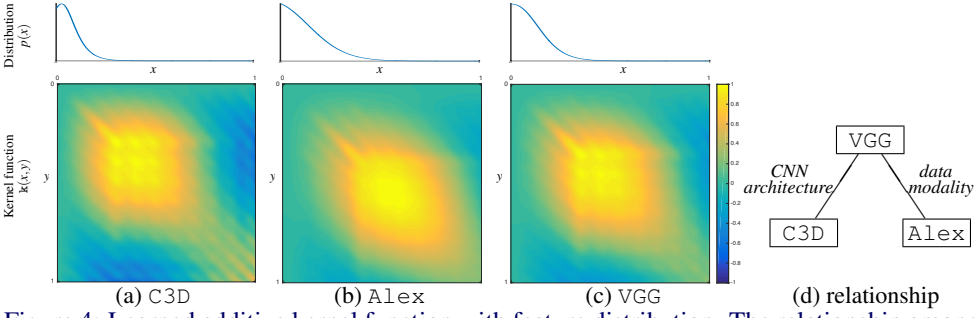
Figure 3: Performance comparison on various CNN features and datasets.

ods to Alex and VGG features on six image datasets: CALTECH256 [6], VOC2007 [1], MIT67 [22], SUN397 [33], CATDOG37 [20] and DOG120 [8]. As is the case with C3D (Sec.3.1.1), the learned additive kernels for these CNN image features are found to be general across datasets, and thus we use the additive kernel learned on the composite of the first four datasets (CALTECH256, VOC2007, MIT67 and SUN397) for Alex and VGG features.

For fair comparison, we apply the proposed method of the ranks $K \leq 7$; the rank $K = 7$ produces the same dimensionality as the explicitly mapped features of χ^2 [32], while the Hellinger transformation does not increase the original feature dimensionality. The performance results are shown in Fig. 3. In contrast to the case of hand-crafted features, the χ^2 and Hellinger transformations do not work well; the χ^2-based transformation [32] failed to improve performance in many cases and to make matters worse, the Hellinger transformation [0] degrades performance in all cases. On the other hand, the learned additive kernel favorably boosts the performance, outperforming the other methods; our method of even rank $K = 2$ surpasses the χ^2-based method and that of $K = 6$ produces better performance on average. The learned additive kernel function is adapted to the CNN features whose characteristics are not fully revealed unlike the hand-crafted features to which the χ^2 and Hellinger kernels are favorably applied.

3.3 Generality Across CNN Models

Lastly, we discuss the generality of the learned additive kernel across *CNN models*, while in Sec.3.1.1 the generality over datasets is shown. We first qualitatively compare the additive kernels learned on respective types of CNN features, C3D, Alex and VGG. Fig. 4 shows the learned kernel function $\mathtt{k}(x,y)$ on $(x,y) \in [0,1]^2$ together with the distribution of feature component value x after L_1 normalization. One can see that all the learned kernel



(a) C3D (b) Alex (c) VGG (d) relationship

Figure 4: Learned additive kernel function with feature distribution. The relationship among the kernels leaned on the CNNs is guessed as (d). This figure is best viewed in color.

functions are formed in a similar shape across the CNN features, assigning larger weights on moderately higher feature values which are significantly less frequent as shown in the feature value distribution. The major difference is that the phases are shifted according to the feature distributions. It would be caused by the difference of data modality; that is, C3D features are extracted from videos (spatio-temporal volume) while Alex and VGG are applied to images (spatial pixel). From this viewpoint, the kernels on Alex and VGG exhibit some sort of similarity. On the other hand, by considering the architecture of CNN models, the C3D model [30] is defined similarly to that of VGG [28], containing deeply stacked convolution layers. Thus, in disregard of the phase shift, the two kernel functions learned on C3D and VGG are similarly shaped. Based on these discussions, we can guess the relationship shown in Fig. 4d between the kernels adapted to those CNN features; the kernel on VGG has connection to those of C3D and Alex.

For quantitatively investigating the generality of the learned kernels across CNN models, we also apply the learned kernel function to the features that are different from the one used for learning kernel functions; there are three types of features, producing nine combination of features and kernels in total. The performance results are shown in Table 2. While the kernels produce the best performance at the corresponding (consistent) features on which the kernels are learned, they also work well on the other types of feature, outperforming the original feature ('Orig.' in Fig. 3d). As discussed above, the kernel that is leaned on a similar type of feature is effective; for examples, VGG kernel works well for C3D feature (similar CNN architecture) and Alex feature (same data modality), whereas the C3D kernel is less effective for Alex feature which has less connection to C3D. Therefore, among these CNN models, the kernel trained on VGG is effective exhibiting high generality (Table 2d), in accordance with the relationship shown in Fig. 4d.

4 Conclusion

We have proposed a method to learn additive kernels in a bottom-up manner based on data for feature transformation. The non-linear feature transformation is realized by the explicit mapping of the additive kernel and the proposed method directly constructs the mapping function by using proper basis functions derived from Fourier functions, which consequently shapes the additive kernel function. For enhancing discriminativity and generality, the coefficients of the bases are learned in the SVM framework by exploiting the shared component across the linear classifier weights via SVD. In the experiments on various datasets using various

Table 2: Classification accuracies (%) on various combinations of features and kernels. The consistent combinations of features and kernels are indicated by gray columns. The best performance over the inconsistent combinations is highlighted in boldface.

(a) C3D feature

Dataset	kernel learned on C3D	Alex	VGG
HMDB51	54.44	54.14	**54.38**
HOLLYWOOD2	47.00	45.08	**46.68**
UCF101	83.93	84.02	**84.04**
UCF50	93.28	93.15	**93.32**

(b) Alex feature

Dataset	C3D	kernel learned on Alex	VGG
CALTECH256	72.89	73.61	**73.34**
VOC2007	75.99	76.57	**76.39**
MIT67	64.15	64.62	**64.51**
SUN397	48.90	49.38	**49.24**
CATDOG37	80.91	81.82	**81.68**
DOG120	63.41	65.58	**64.80**

(c) VGG feature

Dataset	kernel learned on C3D	Alex	VGG
CALTECH256	83.41	**83.58**	83.55
VOC2007	86.42	**86.49**	86.48
MIT67	**75.48**	75.41	75.71
SUN397	**57.99**	57.90	57.99
CATDOG37	**90.08**	89.90	90.17
DOG120	78.85	**79.03**	78.98

(d) Summary

Averaged performance gain (%) across datasets, compared to the consistent combination.

feature \ kernel	C3D	Alex	VGG
C3D	0	-0.5645	-0.0580
Alex	-0.8851	0	-0.2711
VGG	-0.1086	-0.0944	0
total	-0.9937	-0.6589	-0.3291

pre-trained CNN features, the proposed method exhibited favorable performance improvement, also demonstrating the generality of the learned additive kernel (feature transformation) across datasets and CNN models; the kernel learned on VGG model exhibits favorable generality as well as discriminativity improving performance. This work put emphasis on generality using pre-trained CNNs, and our future works include integration with fine-tuning.

References

[1] The PASCAL Visual Object Classes Challenge 2007 (VOC2007). http://www.pascal-network.org/challenges/VOC/voc2007/index.html.

[2] R. Arandjelović and A. Zisserman. Three things everyone should know to improve object retrieval. In *CVPR*, pages 2911–2918, 2012.

[3] G. Csurka, C. Bray, C. Dance, and L. Fan. Visual categorization with bags of keypoints. In *ECCV Workshop on Statistical Learning in Computer Vision*, pages 1–22, 2004.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255, 2009.

[5] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *ICML*, 2014.

[6] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. Technical Report 7694, Caltech, 2007.

[7] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *CVPR*, pages 1725–1732, 2014.

[8] A. Khosla, N. Jayadevaprakash, B. Yao, and Li Fei-Fei. Novel dataset for fine-grained image categorization. In *CVPR Workshop*, 2011.

[9] T. Kobayashi. Dirichlet-based histogram feature transform for image classification. In *CVPR*, pages 3278–3285, 2014.

[10] T. Kobayashi. Low-rank bilinear classification: Efficient convex optimization and extensions. *International Journal of Computer Vision*, 110(3):308–327, 2014.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012.

[12] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. Hmdb: A large video database for human motion recognition. In *ICCV*, pages 2556–2563, 2011.

[13] G. Lanckriet, N. Cristianini, P. Bartlett, L. E. Ghaoui, and M. I. Jordan. Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research*, 5:27–72, 2004.

[14] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, pages 2169–2178, 2006.

[15] A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. In *CVPR*, pages 5188–5196, 2015.

[16] S. Maji and A. C. Berg. Max-margin additive classifiers for detection. In *ICCV*, pages 40–47, 2009.

[17] M. Marszalek, I. Laptev, and C. Schmid. Actions in context. In *CVPR*, pages 2929–2936, 2009.

[18] A. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled : High confidence predictions for unrecognizable images. In *CVPR*, pages 427–436, 2015.

[19] M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *CVPR*, pages 1717–1724, 2014.

[20] O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. V. Jawahar. Cats and dogs. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3498–3505, 2012.

[21] H. Pirsiavash, D. Ramanan, and C. Fowlkes. Bilinear classifiers for visual recognition. In *NIPS*, 2009.

[22] A. Quattoni and A. Torralba. Recognizing indoor scenes. In *CVPR*, pages 413–420, 2009.

[23] A. Rakotomamonjy, F. R. Bach, S. Canu, and Y. Grandvalet. SimpleMKL. *Journal of Machine Learning Research*, 9:2491–2521, 2008.

[24] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. Cnn features off-the-shelf : an astounding baseline for recognition. In *CVPR Workshop*, pages 512–519, 2014.

[25] K. K. Reddy and M. Shah. Recognizing 50 human action categories of web videos. *Machine Vision and Applications*, 24(5):971–981, 2013.

[26] J. Sánchez, F. Perronnin, T. Mensink, and J. Verbeek. Image classification with the fisher vector: Theory and practice. *International Journal of Computer Vision*, 105(3): 222–245, 2013.

[27] B. Schölkopf and A.J. Smola. *Learning with Kernels*. MIT Press, 2001.

[28] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[29] K. Soomro, A. R. Zamir, and M. Shah. Ucf101: A dataset of 101 human action classes from videos in the wild. In *CRCV-TR-12-01*, 2012.

[30] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3d convolutional networks. In *ICCV*, pages 4489–4497, 2015.

[31] V.N. Vapnik. *Statistical Learning Theory*. Wiley, 1998.

[32] A. Vedaldi and A. Zisserman. Efficient additive kernels via explicit feature maps. In *CVPR*, 2010.

[33] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *CVPR*, 2010.