Kervolutional Neural Networks

Chen Wang[1]

chenwang@dr.com

Jianfei Yang[1]

yang0478@ntu.edu.sg

Lihua Xie[1]

elhxie@ntu.edu.sg

Junsong Yuan[2]

jsyuan@buffalo.edu

[1]Nanyang Technological University   [2]State University of New York at Buffalo

Abstract

*Convolutional neural networks (CNNs) have enabled the state-of-the-art performance in many computer vision applications. However, little effort has been devoted to establishing convolution in non-linear space. The existing works mainly leverage on the activation layers, which can only provide point-wise non-linearity. To solve this problem, a new operation, kervolution (**ker**nel con**volution**), is introduced to approximate complex behaviors of human perception systems leveraging on the kernel trick. It generalizes convolution, enhances the model capacity, and captures higher order interactions of features, via patch-wise kernel functions, but without introducing additional parameters. The extensive experiments show that the kervolutional neural networks (KNN) achieve better accuracy and faster convergence in visual recognition than the baseline CNN.*

1. Introduction

Convolutional neural networks (CNNs) have been tremendously successful in computer vision, *e.g.* image recognition [31, 20] and object detection [16, 41]. The core operator, convolution, was partially inspired by the animal visual cortex where different neurons respond to stimuli in a restricted and partially overlaped region known as the receptive field [27, 28]. Convolution leverages on its equivariance to translation to improve the performance of a machine learning system [18]. Its efficiency lies in that the learnable parameters are sparse and shared across the entire input (receptive field). Nonetheless, convolution still has certain limitations, which will be analyzed below. To address them, this paper introduces kervolution to generalize convolution via the kernel trick. The artificial neural networks containing kervolutional layers are named as kervolutional neural networks (KNN).

There is circumstantial evidence that suggests most cells inside striate cortex[1] can be categorized as simple, complex, and hypercomplex, with specific response properties [28].

---

[1]The striate cortex is the part of the visual cortex that is involved in processing visual information.

However, the convolutional layers are linear and designed to mimic the behavior of simple cells in human visual cortex [56], hence they are not able to express the non-linear behaviors of the complex and hypercomplex cells inside the striate cortex. It was also demonstrated that higher order non-linear feature maps are able to make subsequent linear classifiers more discriminative [36, 1, 9]. However, the non-linearity of existing artificial neural networks mostly comes from the activation layers, *e.g.* sigmoid and rectified linear unit (ReLU), which can only provide point-wise non-linearity. We argue that CNN may perform better if convolution can be generalized to patch-wise non-linear operations via kernel trick. Because of the increased expressibility and model capacity, better model generalization may be obtained.

Non-linear generalization is simple in mathematics, however, it is generally difficult to retain the advantages of convolution, *i.e.* (i) sharing weights (weight sparsity) and (ii) low computational complexity. There exists several works towards non-linear generalization. The non-linear convolutional networks [56] implement a quadratic convolution at the expense of additional $n(n+1)/2$ parameters, where n is the size of the receptive field. However, the quadratic form of convolution loses the property of "weight sparsity", since the number of additional parameters of the non-linear terms increases exponentially with the polynomial order, which dramatically increases the training complexity. Another strategy to introduce high order features is to explore the pooling layers. The kernel pooling method in [9] directly concatenates the non-linear terms, while it requires the calculation of non-linear terms, resulting in a higher complexity.

To address the above problems, kervolution is introduced in this paper to extend convolution to kernel space while keeping the aforementioned advantages of linear convolutions. Since convolution has been applied to many fields, *e.g.* image and signal processing, we expect kervolution will also play an important role in those applications. However, in this paper, we focus on its applications in artificial neural networks. The contributions of this paper include: (i) via kernel trick, the convolutional operator is generalized to kervolution, which retains the advantages of convolution and brings new features, including the increased model capacity,

translational equivariance, stronger expressibility, and better generalization; (ii) we provide explanations for kervolutional layers from feature view and show that it is a powerful tool for network construction; (iii) it is demonstrated that KNN achieves better accuracy and surpasses the baseline CNN.

2. Related Work

As the name indicates, CNN [34] employs convolution as the main operation, which is modeled to mimic the behavior of simple cells found in the *primary visual cortex* known as V1 [27, 28]. It have been tremendously successful in numerous applications [32, 31, 20, 18]. Many strategies have been applied to improve the capability of model generalization.

AlexNet [31] proves that the ensemble method "dropout" is very effective for reducing over-fitting of convolutional networks. The non-saturated rectified linear unit (ReLU) dramatically improves the convergence speed [31] and becomes a standard component of CNN. Network in network (NIN) [35] establishes micro networks for local patches within the receptive field, each of which consists of multiple fully connected layers followed by non-linear activation functions. This improves the model capacity at the expense of additional calculation and complex structures. GoogLeNet [47] increases both depth and width of CNN by introducing the Inception model, which further improves the performance. VGG [45] shows that deep CNN with small convolution filters (3×3) is able to bring about significant improvement.

ResNet [20] addresses the training problem of deeper CNN and proposes to learn the residual functions with reference to the layer input. This strategy makes CNN easier to optimize and improves regression accuracy with an increased depth. DenseNet [25] proposes to connect each layer to every other layer in a feed-forward fashion, which further mitigates the problem of vanishing-gradient. The ResNeXt [55] is constructed by repeating a building block that aggregates a set of transformations with the same topology, resulting in a homogeneous, multi-branch architecture. It demonstrates the essence of a new dimension, which is the size of the set of transformations. In order to increase the representation power, SENet [24] focuses on channels and adaptively recalibrates channel-wise feature responses by explicitly modeling the interdependencies channels.

In recent years, researchers have been paying much attentions on the extension of convolution. To enable the expressibility of convolution for complex cells, the non-linear convolutional network [56] extends convolution to non-linear space by directly introducing high order terms. However, as indicated before, this introduces a large number additional parameters and increases the training complexity exponentially. To be invariant to spatial transformations, the spatial transformer network [29] inserts learnable modules to CNN for manipulating transformed data. For the same purpose, deformable convolutional network [10] adds 2-D learnable

offsets to regular grid sampling locations for standard convolution, which enables the learning of affine transforms; while [23] applies a simple two-parameter image warp before a convolution. CapsNet [43] proposes to replace convolution by representing the instantiation parameters of a specific type of entity as activity vectors via a capsule structure. This opens a new research space for artificial neural networks, although its performance on large dataset is still relatively weak. Decoupled network [37] interprets convolution as the product of norm and cosine angle of the weight and input vectors, resulting in explicit geometrical modeling of the intra-class and extra-class variation. To process graph inputs, Spline-CNN [14] extends convolution by using continuous B-spline basis, which is parametrized by a constant number of trainable control values. To decrease the storage, Modulated CNN [54] extends the convolution operator to binary filters, resulting in easier deployment on low power devices.

The kernel technique in this paper was applied to create non-linear classifiers in the context of optimal margin [4], which was later recognized as support vector machines (SVM) [8]. It recently has also been widely applied to correlation filter for improving the processing speed. For example, the kernelized correlation filter (KCF) [22] is proposed to speed up the calculation of kernel ridge regression by bypassing a big matrix inversion, while it assumes that all the data are circular shifts of each other [48], hence it can only predict signal translation. To break this theoretical limitation, the kernel cross-correlator (KCC) is proposed in [53] by defining the correlator in frequency domain directly, resulting in a closed-form solution with computational complexity of $\mathcal{O}(N \log N)$, where N is the signal length. Moreover, it does not impose any constraints on the training data, thus KCC is useful for other applications [51, 49, 52] and is applicable for affine transform prediction, *e.g.* translation, rotation, and scale. This theorem is further extended to speed up the prediction of joint rotation-scale transforms in [50]. The above works show that the kernel technique is a powerful tool for obtaining both accuracy and efficiency.

The kernel technique recently has also been applied to artificial neural networks to improve the model performance. The convolutional kernel network (CKN) [39] proposes to learn transform invariance by kernel approximation, where the kernel is used as a tool for learning CNN. Nevertheless, the aim of CKN is not to extract non-linear features and it is only different from CNN in the cost functions. The SimNets [5] proposes to insert kernel similarity layers under convolutional layer. However, both the similarity templates and filters are needed to be trained and require a pre-training process for initialization, which dramatically increases the complexity. To capture higher order interactions of features, the kernel pooling [9] is proposed in a parameter-free manner. This is motivated by the aforementioned idea that higher dimensional feature map produced by kernel functions is

able to make subsequent linear classifier more discriminative [1]. However, the kernel extension in the pooling stage is not able to extract non-linear features in a patch-wise way. Moreover, the additional higher order features are still need to be calculated explicitly, which also dramatically improves the complexity. To solve these problems, kervolution is defined to generalize convolution via the kernel trick.

3. Kervolution

We start from a convolution with output $\mathbf{f}(\mathbf{x})$, *i.e.*

$$\mathbf{f}(\mathbf{x}) = \mathbf{x} \oplus \mathbf{w}, \qquad (1)$$

where $\oplus$ is the convolutional operator and $\mathbf{x} \in \mathbb{R}^n$ is a vectorized input and $\mathbf{w} \in \mathbb{R}^n$ is the filter. Specifically, the i_{th} element of the convolution output $\mathbf{f}(\mathbf{x})$ is calculated as:

$$\mathbf{f}_i(\mathbf{x}) = \langle \mathbf{x}_{(i)}, \mathbf{w} \rangle, \qquad (2)$$

where $\langle \cdot, \cdot \rangle$ is the inner product of two vectors and $\mathbf{x}_{(i)}$ is the circular shift of $\mathbf{x}$ by i elements. We define the index i started from 0. The kervolution output $\mathbf{g}(\mathbf{x})$ is defined as

$$\mathbf{g}(\mathbf{x}) = \mathbf{x} \otimes \mathbf{w}, \qquad (3)$$

where $\otimes$ is the kervolutional operator. Specifically, the i_{th} element of $\mathbf{g}(\mathbf{x})$ is defined as:

$$\mathbf{g}_i(\mathbf{x}) = \langle \varphi(\mathbf{x}_{(i)}), \varphi(\mathbf{w}) \rangle, \qquad (4)$$

where $\varphi(\cdot) : \mathbb{R}^n \mapsto \mathbb{R}^d \ (d \gg n)$ is a non-linear mapping function. The definition (4) enables us to extract features in a high dimensional space, while its computational complexity is also much higher than (2). Fortunately, we are able to bypass the explicit calculation of the high dimensional features $\varphi(\mathbf{x}_{(i)})$ via the kernel trick [8], since

$$\langle \varphi(\mathbf{x}_{(i)}), \varphi(\mathbf{w}) \rangle = \sum_j c_j (\mathbf{x}_{(i)}^T \mathbf{w})^j = \kappa(\mathbf{x}_{(i)}, \mathbf{w}), \quad (5)$$

where $\kappa(\cdot, \cdot) : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}$ is a kernel function, whose complexity is normally of $\mathcal{O}(n)$ as same as the inner product of convolution. The coefficient c_j can be determined by the mapping function $\varphi(\cdot)$ or a predefined kernel $\kappa(\cdot, \cdot)$, *e.g.* the Gaussian RBF kernel, in which the feature dimension d is infinite. Intuitively, the inner product (2) is a linear kernel, thus convolution is a linear case of kervolution.

The kervolution (3) retains the advantages of convolution and brings new features: (i) sharing weights (Section 3.1); (ii) equivariance to translation (Section 3.2); (iii) increased model capacity and new feature similarity (Section 3.3);

3.1. Sharing Weights

Sharing weights normally mean less trainable parameters and lower computational complexity. It is straightforward

that the number of elements in filter $\mathbf{w}$ is not increased according to the definition of (5), thus kervolution keeps the sparse connectivity of convolution. As a comparison, we take the Volterra series-based non-linear convolution adopted in [56] as an example, the additional parameters of non-linear terms dramatically increase the training complexity, since the number of learnable parameters increases exponentially with the order of non-linearity. Even an quadratic expression $\mathbf{g}^v(\mathbf{x})$ in (6) from [56] is of complexity $\mathcal{O}(n^2)$:

$$\mathbf{g}_i^v(\mathbf{x}) = \mathbf{x}_{(i)}^T \mathbf{w}_2 \mathbf{x}_{(i)} + \mathbf{w}_1^T \mathbf{x}_{(i)}, \qquad (6)$$

where $\mathbf{w}_1 \in \mathbb{R}^n$ and $\mathbf{w}_2 \in \mathbb{R}^{n \times n}$ are the linear and quadratic filters, respectively. The quadratic term in (6) introduces additional $n(n+1)/2$ parameters ($\mathbf{w}_2$ is an upper triangular matrix). Instead, a typical non-linear kernel is normally of complexity $\mathcal{O}(n)$, *i.e.* the Gaussian RBF kernel, which is the same with linear kernel (2), thus kervolution preserves the linear computational complexity $\mathcal{O}(n)$.

Another strategy to introduce higher order features is to explore the pooling layers. For example, the kernel pooling method proposed in [9] directly concatenates the non-linear terms $c_j(\mathbf{x}_{(i)}^T \mathbf{w})^j$ as in (5) to the pooling layers. However, this requires explicit calculation of the non-linear terms up to p orders, although it can be approximated by applying the discrete Fourier transform (DFT) of p times, resulting in a computational complexity of $\mathcal{O}(p \cdot n \log n)$. Nevertheless, based on the kernel trick, kervolution can introduce any order of non-linear terms yet still with linear complexity.

3.2. Translational Equivariance

A crucial aspect of current architectures of deep learning is the encoding of invariances. One of the reasons is that the convolutional layers are naturally equivariant to image translation [18]. In this section, we show that kervolution preserves this important property. An operator is equivariant to a transform when the effect of the transform is detectable in the operator output [6]. Therefore, we have $\mathbf{f}_{(j)}(\mathbf{x}) = \mathbf{x}_{(j)} \oplus \mathbf{w}$, which means the input translation results in the output translation [18]. Similarly,

Theorem 3.1. *kervolution* (3) *is equivariant to translation.*

Proof. Assume $\mathbf{g}'(\mathbf{x}) = \mathbf{x}_{(j)} \otimes \mathbf{w}$, according to (5), we have

$$\mathbf{g}_i'(\mathbf{x}) = \kappa(\mathbf{x}_{(i+j)}, \mathbf{w}) = \mathbf{g}_{i+j}(\mathbf{x}). \qquad (7)$$

Therefore, the i_{th} element of $\mathbf{x}_{(j)} \otimes \mathbf{w}$ is the $(i+j)_{\text{th}}$ element of $\mathbf{g}(\mathbf{x})$, hence we have

$$\mathbf{g}_{(j)}(\mathbf{x}) = \mathbf{x}_{(j)} \otimes \mathbf{w}, \qquad (8)$$

which completes the proof. $\square$

Note that the translational invariance of CNN is achieved by concatenating pooling layers to convolutional layers [18],
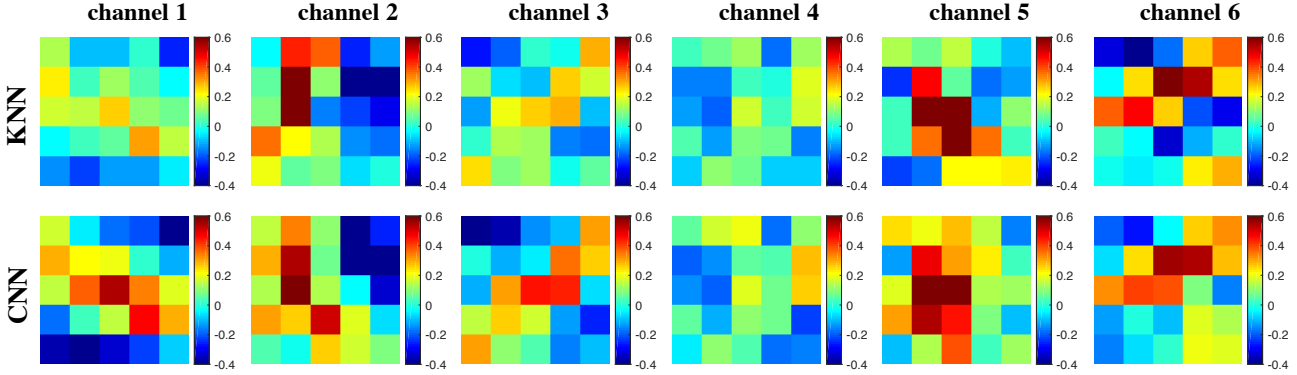
Figure 1. The comparison of learned filters on MNIST from the first layer (six channels and filter size of 5×5) of CNN and polynomial KNN. It is interesting that some of the learned filters (*e.g.* channel 4) from KNN are quite similar to CNN. This indicates that part of the kervolutional layer learns linear behavior, which is controlled by the linear part of the polynomial kernel.

Method	Convolution	L^1-norm	L^2-norm
None	99.17	99.12	99.11
FGSM	71.92	**74.08**	**76.36**

Table 1. Test Accuracy (%) of the white-box FGSM attack with L^p-norm kervolution on MNIST. 10K images are randomly selected.

and the translational invariance of KNN can be achieved similarly. This property is crucial, since when invariances are present in the data, encoding them explicitly in an architecture provides an important source of regularization, which reduces the amount of training data required [23]. As mentioned in Section 2, the same property is also presented in [22], which is achieved by assuming that all the training samples are circular shifts of each other [48], while ours is inherited from convolution. Interestingly, the kernel cross-correlator (KCC) defined in [53] is equivariant to any affine transforms (*e.g.*, translation, rotation, and scale), which may be useful for further development of this work.

3.3. Model Capacity and Features

It is straightforward that the kernel function (5) takes kervolution to non-linear space, thus the model capacity is increased without introducing extra parameters. Recall that CNN is a powerful approach to extract discriminative local descriptors. In particular, the linear kernel (2) of convolution measures the similarity of the input $\mathbf{x}$ and filter $\mathbf{w}$, *i.e.* the cosine of the angle θ between the two patches, since $\langle \mathbf{x}, \mathbf{w} \rangle = \cos(\theta) \cdot \|\mathbf{x}\| \|\mathbf{w}\|$. From this point of view, kervolution measures the similarity by match kernels, which are equivalent to extracting specific features [3]. We next discuss how to interpret the kernel functions and present a few instances $\kappa(\cdot, \cdot)$ of the kervolutional operator. One of the advantages of kervolution is that the non-linear properties can be customized without explicit calculation.

L^p-**Norm Kervolution**   The L^1-norm in (9a) and L^2-norm in (9b) simply measures the Manhattan and Euclidean distances between input $\mathbf{x}$ and filter $\mathbf{w}$, respectively.

$$\kappa_{\mathrm{m}}(\mathbf{x}, \mathbf{w}) = \|\mathbf{x} - \mathbf{w}\|_1, \tag{9a}$$

$$\kappa_{\mathrm{e}}(\mathbf{x}, \mathbf{w}) = \|\mathbf{x} - \mathbf{w}\|_2. \tag{9b}$$

Both "distances" of two points involves aggregating the distances between each element. If vectors are close on most elements, but more discrepant on one of them, Euclidean distance will reduce that discrepancy (elements are mostly smaller than 1 because of the normalization layers), being more influenced by the closeness of the other elements. Therefore, the Euclidean kervolution may be more robust to slight pixel perturbation. This hypothesis is verified by a simple simulation of adversary attack using the fast gradient sign method (FGSM) [19], shown in Table 1, where 'None' means the test accuracy on clean data.

Polynomial Kervolution   Although the existing literatures have shown that the polynomial kernel (10) works well for the problem of natural language processing (NLP) when $d_p = 2$ using SVM [17], we find its performance is better when $d_p = 3$ in KNN for the problem of image recognition.

$$\kappa_{\mathrm{p}}(\mathbf{x}, \mathbf{w}) = (\mathbf{x}^T \mathbf{w} + c_p)^{d_p} = \sum_{j=0}^{d_p} c_p^{d_p - j} (\mathbf{x}^T \mathbf{w})^j, \tag{10}$$

where d_p $(d_p \in \mathbb{Z}^+)$ extends the feature space to d_p dimensions; c_p $(c_p \in \mathbb{R}^+)$ is able to balance the non-linear orders (Intuitively, higher order terms play more important roles when $c_p < 1$). As a comparison, the kernel pooling strategy [9] concatenates the non-linear terms $c_j (\mathbf{x}^T \mathbf{w})^j$ directly, while they are finally linearly combined by subsequent fully connected layer, which dramaticaly increases the number of learnable parameters in the linear layer.

4

To show the behavior of polynomial kervolution, the learned filters of LeNet-5 trained for MNIST are visualized in Figure 1, which contains all six channels of the first kervolutional layer using polynomial kernel ($d_p = 2, c_p = 0.5$). The optimization process is described in Section 4. For a comparison, the learned filters from CNN are also presented. It is interesting that some of the learned filters of KNN and CNN are quite similar, *e.g.* channel 4, which means that part of the capacity of KNN learns linear behavior as CNN. This verifies our understanding of polynomial kernel, which is a combination of linear and higher order terms. This phenomenon also indicates that polynomial kervolution introduces higher order feature interaction in a more flexible and direct way than the existing methods.

Gaussian Kervolution The Gaussian RBF kernel (11) extends kervolution to infinite dimensions.

$$\kappa_g(\mathbf{x}, \mathbf{w}) = \exp(-\gamma_g \|\mathbf{x} - \mathbf{w}\|^2), \qquad (11)$$

where γ_g ($\gamma_g \in \mathbb{R}^+$) is a hyperparameter to control the smoothness of decision boundary. It extends kervolutoin to infinite dimensions because of the i-degree terms in (12).

$$\kappa_g(\mathbf{x}, \mathbf{w}) = C \sum_{i=0}^{\infty} \frac{(\mathbf{x}^T \mathbf{w})^i}{i!}, \qquad (12)$$

where $C = \exp\left(-\frac{1}{2}\left(\|\mathbf{x}\|^2 + \|\mathbf{w}\|^2\right)\right)$ if $\gamma_g = \frac{1}{2}$.

The expression (12) is helpful for our intuitive understanding, while the recent discovery reveals more information. It is shown in [2] that the Gaussian kernel and its variants are able to measure the similarity of gradient based patch-wise features, *e.g.* SIFT [38] and HOG [11]. This provides a unified way to generate a rich, diverse visual feature set [15]. However, instead of using the hand-crafted features as kernel SVM, with KNN, we are able to inherit the substantial achievements based on kernel trick while still taking advantage of the great generalization ability of neural networks.

3.4. Kervolutional Layers and Learnable Kernel

Similar to a convolutional layer, the operation of a kervolutional layer is slightly different from the standard definition (3) in which $\mathbf{x}_{(i)}$ becomes a 3-D patch in a sliding window on the input. To be compatible with existing works, we also implement all popular available structures of convolution in CNN library [40] for kervolution, including the input and output channels, input padding, bias, groups (to control connections between input and output), size, stride, and dilation of the sliding window. Therefore, the convolutional layers of all existing networks can be directly or partially replaced by kervolutional layers, which makes KNN inherit all the the existing achievements of CNN, *e.g.* network architectures [31, 20] and their numerous applications [41].

With kervolution, we are able to extract specific type of features without paying attention to the weight parameters. However, as aforementioned, we still need to tune the hyperparameters for some specific kernels, *e.g.* the balancer c_p in polynomial kernel, the smoother γ_g in Gaussian RBF kernel. Although we noticed that the model performance is mostly insensitive to the kernel hyperparameters, which is presented in Section 4.2, it is sometimes troublesome when we have no knowledge about the kernel. Therefore, we also implement training the network with learnable kernel hyperparameters based on the back-propagation [42]. This slightly increases the training complexity theoretically, but in experiments we found that this brings more flexibility and the additional cost for training several kernel parameters is negligible, compared to learning millions of parameters in the network. Taking the Gaussian kervolution as an example, the gradients are computed as:

$$\frac{\partial}{\partial \mathbf{w}} \kappa_g(\mathbf{x}, \mathbf{w}) = 2\gamma_g(\mathbf{x} - \mathbf{w})\kappa_g(\mathbf{x}, \mathbf{w}), \qquad (13a)$$

$$\frac{\partial}{\partial \gamma_g} \kappa_g(\mathbf{x}, \mathbf{w}) = -\|\mathbf{x} - \mathbf{w}\|^2 \kappa_g(\mathbf{x}, \mathbf{w}). \qquad (13b)$$

Note that the polynomial order d_p is not trainable because of the integer limitation, since the real exponent may produce complex numbers, which makes the network complicated.

4. Ablation Study

This section explores the influences of the kernels, the hyperparameters, and combination of kervolutional layers using LeNet-5 and MNIST [33]. To eliminate the influence of other factors, all configurations are kept as the same. The accuracy of modern networks on MNIST has been saturated, thus we adopt the evaluation criteria proposed in DAWN-Bench [7] that jointly considers the computational efforts and precision. It measures the total training time to a target validation accuracy (98%), which is a trade-off between efficiency and accuracy. In all the experiments of this section, we apply the stochastic gradient descent (SGD) method for training, where a mini-batch size of 50, a momentum of 0.9, an initial learning rate of 0.003, a multiplicative factor of 0.1, and a maximum epoch of 20 with milestones of $[10, 15]$ are adopted. Our algorithm is implemented based on the PyTorch library [40]. All tests are conducted on a single Nvidia GPU of GeForce GTX 1080Ti. The reported training time does not include the testing and checkpoint saving time.

4.1. Kernels

Following the ablation principle, we only replace the convolutional layers of LeNet-5 by kernvolutional layer using three kernel functions, *i.e.* polynomial kernel of $d_p = 3, c_p = 1$ in (10), Gaussian kernel of $\gamma_g = 1$ in (11), and also sigmoid kernel $\kappa_s(\mathbf{x}, \mathbf{w}) = \tanh(\mathbf{x}^T \mathbf{w})$. As shown in
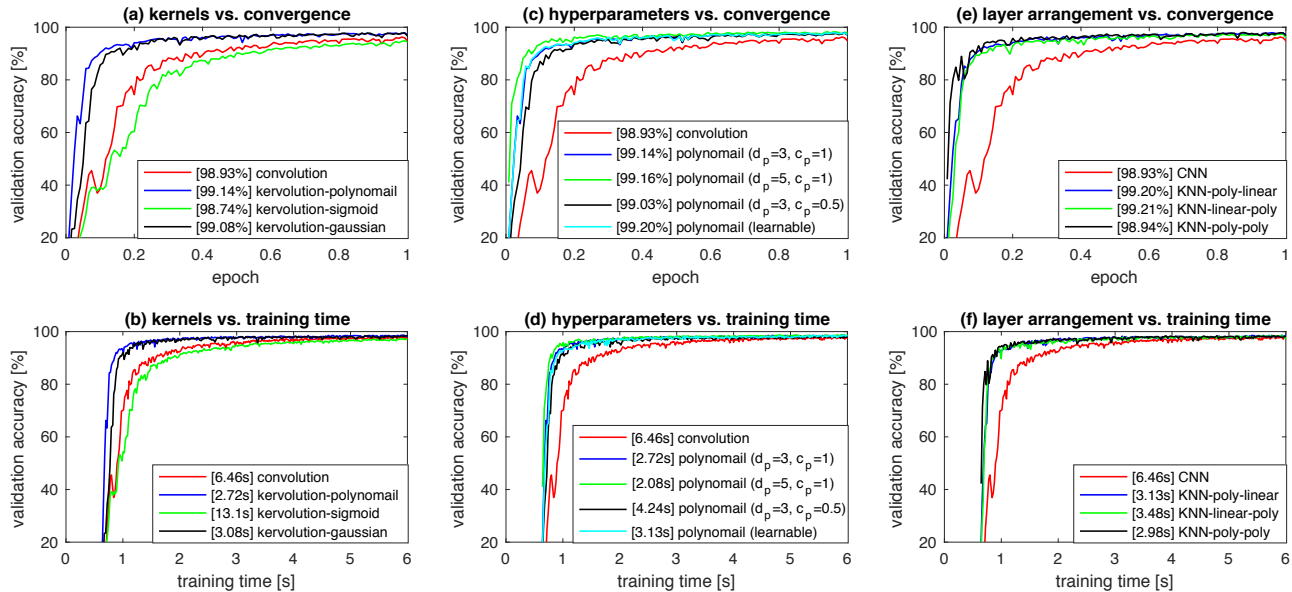
Figure 2. The influences of kervolution on the convergence rate. The best validation accuracy (20 epochs) and training time to target accuracy (98%) are displayed in brackets within the legend of each figure, respectively. (a) and (b) demonstrate that the kernel functions have a significant impact on the convergence speed. (c) and (d) demonstrate that the hyperparameters of kervolutional layer play less important roles than kernels. (e) and (f) shows the effects of arrangement of kervolutional layers.

Figure 2 (a) and (b), although the computational complexity of non-linear kernels is slightly higher than that of linear kernel (convolution), the polynomial and Gaussian KNN are still able to converge to a validation accuracy of 98% more than $2\times$ faster than the original CNN. However, the convergence speed of sigmoid KNN is $2\times$ slower than that of CNN, which indicates that the kernel functions are crucial and have a significant impact on performance. Thanks to the wealth of traditional methods, we have many other useful kernels [46], although we cannot test all of them in this paper. The L^1 and L^2-norm KNN achieve an accuracy of 99.05% and 99.19%, respectively, but we omit them in Figure 2 (a) and (b) because they nearly coincide with the polynomail curve.

4.2. Hyperparameters

From the above analysis, we credit the significant improvements of convergence speed to the usage of different kernels. This part verifies this assumption and further explores the influences of kernel hyperparameters. The polynomial kervolution (10) with two hyperparameters, non-linear order d_p and balancer c_p, is selected. As shown in Figure 2 (c) and (d), the convergence speed and validation accuracy of polynomial KNN using different hyperparameters are pretty similar to that of Figure 2 (a) and (b), which indicates that KCC is less sensitive to the kernel hyperparameters.

It is also noticed that the KNN with learnable kernel parameters achieves the best precision (99.20%) in this group, although it slightly increases the training time compared to KNN ($d_p = 3, c_p = 1$). However, the cost is justifiable

since it saves the hyperparameter tuning process and the convergence is still much faster than the baseline CNN.

4.3. Layer Arrangement

This part explores the influences of the placement of kervolutional layers. Thanks to the simplicity of LeNet-5 (two convolutional layers), we can test all possible configurations of layer arrangement, *i.e.* "conv-conv", "kerv-conv", "conv-kerv", and "kerv-kerv". As shown in Figure 2 (e) and (f), where the polynomial kernel ($d_p = 3, c_p = 1$) is adopted, KNN still brings faster convergence. One interesting phenomenon is that the architecture of "kerv-conv" achieves better precision but slower convergence than "conv-kerv" (we run multiple times and the results are similar). This indicates that the sequence of kervolutional layers has an impact on performance, although the model complexity is the same. One side effect is that we may need to make some efforts to adjust the layer sequence for deeper KNNs. It is also noticed that the architecture of 'kerv-kerv' achieves the fastest convergence but only with a comparable validation accuracy to CNN. We argue that this is caused by the overfitting problem, since its final training loss is very close to others (≈ 0.01), which means that the model capacity of double kervolutional layers together with the activation and max pooling layers is too large for the MNIST dataset.

4.4. Removing ReLU

As mentioned in Section 1, the non-linearity of CNN mainly comes from the activation (ReLU) and max pool-

6

Network	CIFAR-10	CIFAR-100
CNN [26]	13.63	44.74
KNN	10.85	37.12

Table 2. Validation error (%) of ResNets on CIFAR-10 and CIFAR-100 without data augmentation.

Architecture	CIFAR-10+		CIFAR-100+	
	CNN	KNN	CNN	KNN
GoogLeNet [13]	13.37	5.16	26.65	20.84
ResNet [20]	6.43	4.69	27.22	22.49
DenseNet [25]	5.24	5.08	24.42	24.92

Table 3. Validation error (%) on CIFAR-10 and CIFAR-100 on different architectures with data augmentation.

Hyperparm	$d_p = 3$	$d_p = 5$
$c_p = 0$	4.78	5.42
$c_p = 1$	4.60	5.36
learnable c_p	4.76	4.73

Table 4. Validation error (%) of polynomial KNN using ResNet-32 on CIFAR-10+ using different hyper parameters.

Hyperparm	$d_p = 3$	$d_p = 5$
$c_p = 0$	0.83	1.41
$c_p = 1$	1.41	1.46
learnable c_p	0.86	0.79

Table 5. Training time (h) of polynomial KNN using ResNet-32 on CIFAR-10+ using different hyper parameters.

ing layers. Intuitively, KNN may be able to achieve same model performance without activation or max pooling layers. To this end, we simply remove all the activation layers of LeNet-5 and replace the max pooling by average pooling layers, which means that all the non-linearity comes from the kervolutional layers. Without surprise, the CNN only achieves an accuracy of 92.22%, which is far from the target accuracy of 98%, hence the training time comparison figure is omitted. However, the KNN of "gaussian-polynomial" and "polynomial-polynomial" both achieve an accuracy of 99.11%, which further verifies the effectiveness of kervolution. In another sense, the strategy of removing the activation layers is one of the solutions to the aforementioned over-fitting problem in Section 4.3, although we need more investigations to find the best architectures for KNN.

5. Performance

This section aims to demonstrate the effectiveness of deep KNN on larger datasets. In practice, the network architecture has a significant impact on performance. Since the modern networks are so deep and kervolution provides many possibilities via different kernels, we cannot perform exhaustive tests to find the best sequence of kervolutional layers. Hence, we construct KNN based on several existing architectures by mainly changing the first convolutional layers to kervolutional layers. Other factors, such as data augmentation and optimizers, are kept as their original configurations. As discovered in Section 4, this may not be the best configuration, while it can demonstrate the effectiveness of kervolution.

The CIFAR experiments in this section are conducted in a single GPU of Nvidia GeForce GTX 1080Ti while we employ four Nvidia Tesla M40 in the ImageNet experiments. The polynomial kervolutional layer in this section adopts the learnable balancer c_p with power $d_p = 3$.

5.1. CIFAR

The CIFAR-10 and CIFAR-100 [30] datasets consist of colored natural images with 32×32 pixels in 10 and 100 classes, respectively. Each dataset contains 50k images for training and 10k for testing. In the testing procedure, only the single view of the original image is evaluated.

The proposed KNNs are first evaluated without data augmentaion using the architecture of ResNet. We construct and train ResNet-110 following the architecture of [20] with cross-entropy loss. The stochastic gradient descent (SGD) is adopted with momentum of 0.9. We train the networks for 200 epochs with a mini-batch size of 128. The learning rate decays by 0.1 at the 75, 125, and 150 epochs; the weight decay stays at 5×10^{-4}. The validation error of KNN as well as the best performance of baseline CNN from [26] are presented in Table 2. It is interesting that KNN outperforms CNN with a significant improvements on the CIFAR dataset.

We perform more experiments using the data augmentation techniques, and the datasets are denoted as 'CIFAR-10+' and 'CIFAR-100+', respectively. The KNNs are constructed following the architecture of GoogLeNet [47] and DenseNet-40-12 [25]. Data augmentation is applied following the configuration in ResNet [20], including horizontal flipping with a probability of 50%, reflection-padding by 4 pixels, and random crop with size 32×32.

Different from ResNet, we train DenseNet-40-12 following its original configuration [25], *i.e.* SGD with batch size 64 for 300 epochs. The initial learning rate is set to 0.1, and is divided by 10 at 50% and 75% of the total number of iterations. Table 3 lists the performance of KNN and baselines from [20, 25, 13]. We cannot see a significant improvement on DenseNet, which indicates that polynomial kervolution may not suit for fully connected architecture.

We further demonstrate the sensitivity to the kernel hyperparameters. Table 4 lists the validation errors of KNN

on CIFAR-10+ with polynomial kernel of $d_p = 3, 5$ and $c_p = 0, 1$ using the architecture of ResNet-32. The performance with learnable balancer of c_p is also given for comparison. As suggested by [7], their training time to an accuracy of 94% is measured and presented in Table 5. It is interesting that, the configuration of $d_p = 3, c_p = 1$ achieves the best accuracy, while $d_p = 5$ with learnable c_p requires the least training time. The networks with learnable kernel achieve the best overall performance by jointly considering the training time and accuracy. This indicates that the learnable kernel technique can produce compromised performance without tuning parameters.

5.2. ImageNet

The ILSVRC 2012 classification dataset [12] is composed of 1.2M images for training and 50k images for validation in 1000 classes. For fair comparison, we apply the same data augmentation as described in [20, 21], where the single-crop and 10-crop at a size of 224×224 are applied for testing.

We select four versions of ResNet [20], including ResNet-18, ResNet-34, ResNet-50 and ResNet-101, as the baselines. The kervolutional layer is applied with a polynomial kernel ($d = 3, c_p = 2$). All the networks are trained using the stochastic gradient descent (SGD) method for 100 epochs with a batch size of 256. The learning rate is set to 0.1, and is reduced every 30 epochs. Also, a weight decay of 10^{-4} and a momentum of 0.9 without dampening are employed. In our experiments, the best performance of ResNet cannot be achieved in limited training time. To guarantee a fair comparison, the results of ResNet which have the best accuracy in [44, 20, 21, 25] are chosen. We report the single-crop and 10-crop validation error on ImageNet in Table 6, where the performance of KNN is the average of five running.

In Table 6, top-1 errors using ResNet-18/34/50/101 are reduced by 0.5%, 0.41%, 0.29%, and 0.7% in single-crop testing and 0.45%, 0.75%, 0.80%, and 0.83% in 10-crop testing, respectively. For top-5 errors, KNN outperform corresponding ResNets by 0.43%, 0.24%, 0.23%, and 0.28% in single-crop testing, and 0.39%, 0.68%, 0.74%, and 0.87% in 10-crop testing, respectively. It is noticed that simple replacement of the convolutional layer in ResNet leads to obvious improvements. We believe that more customized network architecture as well as extensive hyperparameter searches can further improve the performance on ImageNet.

6. Discussion

Kernel   Different from convolution, which can only extract linear features, kervolution is able to extract customized patch-wise non-linear features, which makes KNN much more flexible. It is demonstrated that the higher order terms make the subsequent linear classifier more discriminant, while this does not increase the computational complexity. However, we have only tested several kernels, *e.g.* polyno-

Network	Top-1	Top-5
ResNet-18	30.24 / 27.88	10.92 / 9.42
KNN-18	**29.74 / 27.43**	**10.49 / 9.03**
ResNet-34	26.70 / 25.03	8.58 / 7.76
KNN-34	**26.29 / 24.28**	**8.34 / 7.08**
ResNet-50	23.85 / 22.85	7.13 / 6.71
KNN-50	**23.56 / 22.05**	**6.90 / 5.97**
ResNet-101	22.63 / 21.75	6.44 / 6.05
KNN-101	**21.93 / 20.92**	**6.16 / 5.18**

Table 6. The top-1 and top-5 validation errors on ImageNet, with single-crop / 10-crop testing, respectively.

mial and Gaussian, which may not be optimal. It is obvious that the kernel functions and their hyperparameters can be task-driven and more investigations are necessary.

Training   It is also noticed that the training can be unstable when a network contains too much non-linearity, this is because the model complexity is too high for a specific task, which can be simply solved by reducing the number of kervolutional layers. More investigations on searching appropriate non-linearity for a specific task is channeling.

Architecture   We have only applied kervolution to the existing architectures, *e.g.* ResNet. While this is not optimal, especially when considering that the mechanism of deep architectures is still unclear. It is obvious that the performance of kernvolution is dependent on the architecture. One of the interesting challenges for future work is to investigate the relationship between the architecture and kervolution.

7. Conclusion

This paper introduces the kervolution to generalize convolution to non-linear space, and extends convolutional neural networks to kervolutional neural networks. It is shown that kervolution not only retains the advantages of convolution, *i.e.* sharing weights and equivalence to translation, but also enhances the model capacity and captures higher order interactions of features, via patch-wise kernel functions without introducing additional parameters. It has been demonstrated that, with careful kernel chosen, the performance of CNN can be significantly improved on MNIST, CIFAR, and ImageNet dataset via replacing convolutional layers by kervolutional layers. Due to the large number of choices of kervolution, we cannot perform a brute force search for all the possibilities, while this opens a new space for the construction of deep networks. We expect the introduction of kervolutional layers in more architectures and extensive hyperparameter searches can further improve the performance.

References

[1] M. Blondel, M. Ishihata, A. Fujino, and N. Ueda. Polynomial Networks and Factorization Machines: New Insights and Efficient Training Algorithms. In *International Conference on Machine Learning*, 2016.

[2] L. Bo, X. Ren, and D. Fox. Kernel Descriptors for Visual Recognition. In *Conference on Neural Information Processing Systems*, 2010.

[3] L. Bo and C. Sminchisescu. Efficient Match Kernel between Sets of Features for Visual Recognition. In *Conference on Neural Information Processing Systems*, pages 135–143, 2009.

[4] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

[5] N. Cohen, O. Sharir, and A. Shashua. Deep SimNets. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 4782–4791. IEEE, 2016.

[6] T. Cohen and M. Welling. Group Equivariant Convolutional Networks. *ICML*, 2016.

[7] C. Coleman, D. Narayanan, D. Kang, T. Zhao, J. Zhang, L. Nardi, P. Bailis, K. Olukotun, C. Re, and M. Zaharia. DAWNBench: An End-to-End Deep Learning Benchmark and Competition. *Training*, 100, 2017.

[8] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[9] Y. Cui, F. Zhou, J. Wang, X. Liu, Y. Lin, and S. Belongie. Kernel Pooling for Convolutional Neural Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3049–3058. IEEE, 2017.

[10] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei. Deformable Convolutional Networks. *ICCV*, 2017.

[11] N. Dalal and B. Triggs. Histograms of Oriented Gradients for Human Detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 886–893. IEEE, 2005.

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[13] A. Dubey, O. Gupta, P. Guo, R. Raskar, R. Farrell, and N. Naik. Pairwise confusion for fine-grained visual classification. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 70–86, 2018.

[14] M. Fey, J. E. Lenssen, F. Weichert, and H. Müller. Splinecnn: Fast geometric deep learning with continuous b-spline kernels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 869–877, 2018.

[15] P. Gehler and S. Nowozin. On feature combination for multiclass object classification. In *2009 IEEE 12th International Conference on Computer Vision (ICCV)*, pages 221–228. IEEE, 2009.

[16] R. Girshick. Fast R-CNN. In *IEEE International Conference on Computer Vision*, pages 1440–1448. IEEE, 2015.

[17] Y. Goldberg and M. Elhadad. splitsvm: fast, space-efficient, non-heuristic, polynomial kernel computation for nlp applications. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*, pages 237–240. Association for Computational Linguistics, 2008.

[18] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[19] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and Harnessing Adversarial Examples. *arXiv.org*, Dec. 2014.

[20] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *The Conference on Computer Vision and Pattern Recognition*, 2016.

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

[22] J. F. Henriques, R. Caseiro, and P. Martins. High-Speed Tracking with Kernelized Correlation Filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(3):583–596, 2015.

[23] J. F. Henriques and A. Vedaldi. Warped Convolutions: Efficient Invariance to Spatial Transformations. In *International Conference on Machine Learning*, 2017.

[24] J. Hu, L. Shen, and G. Sun. Squeeze-and-Excitation Networks. *CVPR*, 2018.

[25] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely Connected Convolutional Networks. *CVPR*, pages 2261–2269, 2017.

[26] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. Deep Networks with Stochastic Depth. In *European Conference on Computer Vision*, 2016.

[27] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106–154, 1962.

[28] D. H. Hubel and T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–243, 1968.

[29] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial Transformer Networks. In *Conference on Neural Information Processing Systems*, 2015.

[30] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Technical report, University of Toronto*, 2009.

[31] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Conference on Neural Information Processing Systems*, 2012.

[32] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

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

[34] Y. LeCun et al. Generalization and network design strategies. *Connectionism in perspective*, pages 143–155, 1989.

[35] M. Lin, Q. Chen, and S. Yan. Network In Network. *arXiv.org*, Dec. 2013.

[36] T.-Y. Lin, A. RoyChowdhury, and S. Maji. Bilinear CNN models for fine-grained visual recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 1449–1457, 2015.

[37] W. Liu, Z. Liu, Z. Yu, B. Dai, R. Lin, Y. Wang, J. M. Rehg, and L. Song. Decoupled Networks. In *The Conference on Computer Vision and Pattern Recognition*, 2018.

[38] D. G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[39] J. Mairal, P. Koniusz, Z. Harchaoui, and C. Schmid. Convolutional Kernel Networks. In *Conference on Neural Information Processing Systems*, 2014.

[40] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS Workshop*, 2017.

[41] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017.

[42] D. E. Rumelhart, G. E. Hinton, R. J. Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[43] S. Sabour, N. Frosst, and G. E. Hinton. Dynamic Routing Between Capsules. In *Conference on Neural Information Processing Systems*, Dec. 2017.

[44] T. X. Sam Gross. fb.resnet.torch. `https://github.com/facebook/fb.resnet.torch`, 2016.

[45] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *Intelnational Conference on Learning Research*, 2015.

[46] A. J. Smola and B. Schölkopf. *Learning with kernels*, volume 4. Citeseer, 1998.

[47] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going Deeper with Convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9. IEEE, 2015.

[48] C. Wang. *Kernel Learning for Visual Perception*. PhD thesis, Nanyang Technological University, 2019.

[49] C. Wang, M.-C. Hoang, L. Xie, and J. Yuan. Non-iterative RGB-D-inertial Odometry. *arXiv.org*, Oct. 2017.

[50] C. Wang, T. Ji, T.-M. Nguyen, and L. Xie. Correlation Flow: Robust Optical Flow using Kernel Cross-Correlators. In *International Conference on Robotics and Automation (ICRA)*. IEEE, May 2018.

[51] C. Wang, J. Yuan, and L. Xie. Non-iterative SLAM. In *International Conference on Advanced Robotics (ICAR)*, pages 83–90. IEEE, 2017.

[52] C. Wang, H. Zhang, T.-M. Nguyen, and L. Xie. Ultra-wideband aided fast localization and mapping system. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[53] C. Wang, L. Zhang, L. Xie, and J. Yuan. Kernel Cross-Correlator. In *The Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[54] X. Wang, B. Zhang, C. Li, R. Ji, J. Han, X. Cao, and J. Liu. Modulated convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 840–848, 2018.

[55] S. Xie, R. B. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated Residual Transformations for Deep Neural Networks. *CVPR*, pages 5987–5995, 2017.

[56] G. Zoumpourlis, A. Doumanoglou, N. Vretos, and P. Daras. Non-linear Convolution Filters for CNN-Based Learning. In *IEEE International Conference on Computer Vision (ICCV)*, pages 4771–4779. IEEE, 2017.