Compact Generalized Non-local Network

Anonymous Author(s)
Affiliation
Address
email

Abstract

The non-local module [31] is designed for capturing long-range spatio-temporal dependencies in images and videos. Although having shown excellent performance, it lacks the mechanism to model the interactions between positions across channels, which are of vital importance in recognizing fine-grained objects and actions. To address this limitation, we generalize the non-local module and take the correlations between the positions of any two channels into account. This extension unifies second-order feature pooling and achieves state-of-the-arts performance on a variety of fine-grained classification tasks. However, it also leads to an explosion in the computational complexity. To alleviate such an issue, we further propose its compact representation to reduce the high-dimensional feature space and large computation burden involved. Moreover, we try to group the channels and do our generalized non-local method within each group. Experimental results illustrate the significant improvements and practical applicability of the generalized non-local module on both fine-grained object and video classification. Code will be made publicly available to ease the future research.

1 Introduction

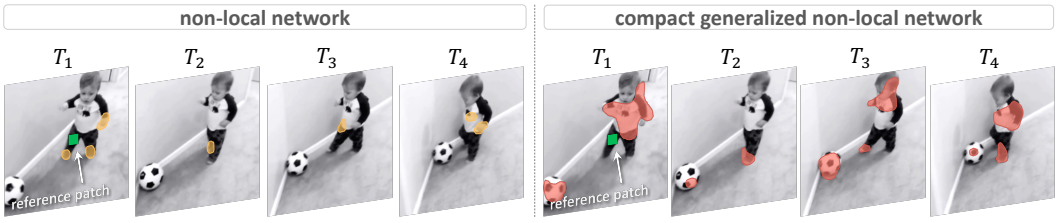

Figure 1: Comparison between non-local (NL) and **compact generalized non-local (CGNL)** on recognizing an action video of kicking the ball. Given the *reference patch* (green rectangle) in the first frame, we visualize for each method the highly related responses in the other frames by thresholding the feature space. CGNL out-performs the original NL in capturing the ball that is not only in long-range distance from the reference patch but also corresponds to different channels in the feature map.

Capturing spatio-temporal dependencies between spatial pixels or temporal frames plays key roles in the tasks of fine-grained object and action classification. Modeling such interactions among images and videos is the major topic of various feature extraction techniques, including SIFT, LBP, Dense Trajectory [30], etc. In the past few years, deep neural network automates the feature designing pipeline by stacking multiple end-to-end convolutional or recurrent modules, where each of them processes correlation within spatial or temporal local regions. In general, capturing the long-range dependencies among images or videos still requires multiple stacking of these modules, which greatly hinders the learning and inference efficiency. Recent work [17] also suggests that stacking more layers cannot always increase the effective receptive fields.

Submitted to 32nd Conference on Neural Information Processing Systems (NIPS 2018). Do not distribute.

Inspired by the classical non-local means for image filtering, the recently proposed non-local neural network [31] addresses this challenge by directly modeling the correlation between any two positions in the feature maps in a single module. Without any bells and whistles, the non-local method can greatly improve existing networks on many video classification benchmarks. Despite its great performance, the original non-local network only considers the global spatio-temporal correlation by merging all channels, and it might miss some subtle but important cross-channel clues for discriminating fine-grained objects or actions. For instance, the body, the ball and their interaction are all necessary for describing the action of kicking the ball in Fig. 1, while the original non-local operation learns to focus on the body part relations but neglect the body-ball interactions that usually correspond to different channels of the input features.

To improve the effectiveness in fine-grained object and action recognition tasks, this work extends the non-local module by learning explicit correlations among all of the elements across the channels. This extension scale-ups the representation power of the non-local operation to attend the interaction between subtle object parts (*e.g.*, the body and ball in Fig. 1). Second, we propose the compact representation for our generalized non-local module to address the high computation burden issue. We show that as a self-contained module, the compact generalized non-local (CGNL) module provides steady improvements in classification tasks. Third, we also investigate the grouped version of CGNL, which groups the channels and model correlations across channels within each group.

We evaluate the proposed CGNL method on the task of fine-grained classification and action recognition. Extensive experimental results show that: 1) The CGNL network are easy to optimize as the original non-local network; 2) Compared with the non-local module, CGNL enjoys capturing richer feature and dense clues for prediction, as shown in Figure 1, which leads to results substantially better than those of the original non-local module.

2 Related Works

Channel Correlations: The mechanism of sharing the same conv kernel among channels of a layer in CNN [12] [11] [23] [26] [7] can be seen as a basic way to capture correlations among channels, which aggregates the channels of feature maps by sum pooling. The SENet [9] may be the first work that explicitly models the interdependencies between the channels of its convolutional features. It aims to select the useful feature maps and suppress the others, and only the global information of each channel is considered. Inspired by [31], we present the generalized non-local (GNL) module, which generalizes the non-local (NL) module to learn the correlations between any two positions across the channels. Compared to the SENet, we model the interdependencies among channels in an explicit and dense manner.

Compact Representation: After further investigation, we find that the non-local module contains a second-order feature space (Sect.3.1), which is used widely in previous computer vision tasks, *e.g.*, SIFT [16], Fisher encoding [18], Bilinear model [14] [4] and segmentation task [1]. However, such second-order feature space involves high dimensions heavy computational burdens. In the area of kernel learning [22], there are many prior works such as compact bilinear pooling (CBP) [4] to use Tensor Sketching [19] to address this problem. Fortunately, in mathematics, the whole non-local operation can be viewed as a trilinear formation. It can be fast computed with the associative law of matrix production. To the other types of pairwise function, such as Embedded Gaussian or RBF [20], we propose a tight approximation for them by using Taylor expansion.

3 Approach

In this section, we introduce a general formulation of the proposed general non-local (GNL) operation. We then show that the original non-local and the bilinear pooling are special cases of this formulation. After that, we illustrate that the general non-local operation can be seen as a modality in the trilinear matrix production and show how to implement our generalized non-local module in a compact representations. In this section, we first review the original non-local and the bilinear pooling and show their connection in formulations. Then a general non-local (GNL) operation is proposed to extend non-local to capture channel-wise feature relations. At last, we show how to implement GNL module in a compact representation.

3.1 Review of Non-local Operation

We begin by briefly reviewing the original non-local operation [31] in matrix form. Suppose that an image or video is given to the network and let $\mathbf{X} \in \mathbb{R}^{N \times C}$ denote (see notation[1]) the input feature map of the non-local module, where C is the number of channels. For the sake of notation clarity, we collapse all the spatial (width W and height H) and temporal (video length T) positions in one dimension, *i.e.*, $N = HW$ or $N = HWT$. To capture long-range dependencies across the whole feature map, the original non-local operation computes the response $\mathbf{Y} \in \mathbb{R}^{N \times C}$ as the weighted sum of the features at all positions,

$$\mathbf{Y} = f\big(\theta(\mathbf{X}), \phi(\mathbf{X})\big) g(\mathbf{X}), \tag{1}$$

where $\theta(\cdot), \phi(\cdot), g(\cdot)$ are learnable transformations on the input. In [31], the authors suggest using 1×1 or $1 \times 1 \times 1$ convolution for simplicity, *i.e.*, the transformations can be written as

$$\theta(\mathbf{X}) = \mathbf{X}\mathbf{W}_\theta \in \mathbb{R}^{N \times C}, \quad \phi(\mathbf{X}) = \mathbf{X}\mathbf{W}_\phi \in \mathbb{R}^{N \times C}, \quad g(\mathbf{X}) = \mathbf{X}\mathbf{W}_g \in \mathbb{R}^{N \times C}, \tag{2}$$

parameterized by the weight matrices $\mathbf{W}_\theta, \mathbf{W}_\phi, \mathbf{W}_g \in \mathbb{R}^{C \times C}$ respectively. The pairwise function $f(\cdot, \cdot) : \mathbb{R}^{N \times C} \times \mathbb{R}^{N \times C} \to \mathbb{R}^{N \times N}$ computes the affinity between all positions (space or space-time). There are multiple choices for f, among which dot-product is perhaps the simplest one, *i.e.*,

$$\mathbf{f}\big(\theta(\mathbf{X}), \phi(\mathbf{X})\big) = \theta(\mathbf{X})\phi(\mathbf{X})^\top. \tag{3}$$

Plugging Eq. 2 and Eq. 3 into Eq. 1 yields a trilinear interpretation of the non-local operation,

$$\mathbf{Y} = \mathbf{X}\mathbf{W}_\theta \mathbf{W}_\phi^\top \mathbf{X}^\top \mathbf{X}\mathbf{W}_g, \tag{4}$$

where the pairwise matrix $\mathbf{X}\mathbf{W}_\theta \mathbf{W}_\phi^\top \mathbf{X}^\top \in \mathbb{R}^{N \times N}$ encodes the similarity between any locations of the input feature. The effect of non-local operation can be related to the self-attention module [28] based on the fact that each position (row) in the result $\mathbf{Y}$ is a linear combination of all the positions (rows) of $\mathbf{X}\mathbf{W}_g$ weighted by the corresponding row of the pairwise matrix.

3.2 Review of Bilinear Pooling

Analogous to the conventional kernel trick [22], the idea of bilinear pooling [14] has recently been adopted in CNNs for enhancing the feature representation in various tasks, such as fine-grained classification, person re-id, action recognition. At a glance, bilinear pooling models pairwise feature interactions using explicit outer product at the final classification layer:

$$\mathbf{Z} = \mathbf{X}^\top \mathbf{X} \in \mathbb{R}^{C \times C}, \tag{5}$$

where $\mathbf{X} \in \mathbb{R}^{N \times C}$ is the input feature map generated by the last convolutional layer. Each element of the final descriptor $z_{c_1 c_2} = \sum_n x_{nc_1} x_{nc_2}$ sum-pools at each location $n = 1, \cdots, N$ the bilinear product $x_{nc_1} x_{nc_2}$ of the corresponding channel pair $c_1, c_2 = 1, \cdots, C$.

Despite the distinct design motivation, it is interesting to see that bilinear pooling (Eq. 5) can be viewed as a special case of the second-order term (Eq. 3) in the non-local operation if we consider,

$$\theta(\mathbf{X}) = \mathbf{X}^\top \in \mathbb{R}^{C \times N}, \quad \phi(\mathbf{X}) = \mathbf{X}^\top \in \mathbb{R}^{C \times N}. \tag{6}$$

3.3 Generalized Non-local Operation

The original non-local operation aims to directly capture long-range dependencies between any two positions in one layer. However, such dependencies are encoded in a joint location-wise matrix $f(\theta(\mathbf{X}), \phi(\mathbf{X}))$ by aggregating all channel information together. On the other hand, channel-wise correlation has been recently explored in both discriminative [14] and generative [27] models through the covariance analysis across channels. Inspired by these works, we generalize the original non-local operation to model long-range dependencies between any positions of any channels.

[1]Bold capital letters denote a matrix $\mathbf{X}$, bold lower-case letters a column vector $\mathbf{x}$. $\mathbf{x}_i$ represents the i^{th} column of the matrix $\mathbf{X}$. x_{ij} denotes the scalar in the i^{th} row and j^{th} column of the matrix $\mathbf{X}$. All non-bold letters represent scalars. $\mathbf{1}_m \in \mathbb{R}^m$ is a vector of ones. $\mathbf{I}_n \in \mathbb{R}^{n \times n}$ is an identity matrix. $\mathrm{vec}(\mathbf{X})$ denotes the vectorization of matrix $\mathbf{X}$. $\mathbf{X} \circ \mathbf{Y}$ and $\mathbf{X} \otimes \mathbf{Y}$ are the Hadamard and Kronecker products of matrices.

We first reshape the output of the transformations (Eq. 2) on $\mathbf{X}$ by merging channel into position:

$$\theta(\mathbf{X}) = \text{vec}(\mathbf{X}\mathbf{W}_\theta) \in \mathbb{R}^{NC}, \phi(\mathbf{X}) = \text{vec}(\mathbf{X}\mathbf{W}_\phi) \in \mathbb{R}^{NC}, g(\mathbf{X}) = \text{vec}(\mathbf{X}\mathbf{W}_g) \in \mathbb{R}^{NC}. \quad (7)$$

By lifting the row space of the underlying transformations, our generalized non-local (GNL) operation pursues the same goal of Eq. 1 that computes the response $\mathbf{Y} \in \mathbb{R}^{N\times C}$ as:

$$\text{vec}(\mathbf{Y}) = f\big(\text{vec}(\mathbf{X}\mathbf{W}_\theta), \text{vec}(\mathbf{X}\mathbf{W}_\phi)\big)\,\text{vec}(\mathbf{X}\mathbf{W}_g). \quad (8)$$

Compared to the original non-local operation (Eq. 4), GNL utilizes a more general pairwise function $f(\cdot,\cdot) : \mathbb{R}^{NC} \times \mathbb{R}^{NC} \rightarrow \mathbb{R}^{NC\times NC}$ that can differentiate between pairs of same location but at different channels. This richer similarity greatly augments the non-local operation in discriminating fine-grained object parts or action snippets that usually correspond to channels of the input feature. Compared to the bilinear pooling (Eq. 5) that can only be used after the last convolutional layer, GNL maintains the input size and can thus be flexibly plugged between any network blocks. In addition, bilinear pooling neglects the spatial correlation which, however, is preserved in GNL.

Recently, the idea of dividing channels into groups has been established as a very effective technique in increasing the capacity of CNNs. Well-known examples include Xception [2], MobileNet [8], ShuffleNet [35], ResNeXt [33] and Group Normalization [32]. Given its simplicity and independence, we also realize the channel grouping idea in GNL by grouping all C channels into G groups, each of which contains $C' = C/G$ channels of the input feature. We then perform GNL operation independently for each group to compute $\mathbf{Y}'$ and concatenate the results along the channel dimension to restore the full response $\mathbf{Y}$.

3.4 Compact Representation

A straightforward implementation of GNL (Eq. 8) is prohibitive as the quadratic increase with respect to the channel number C in the presence of the $NC \times NC$ pairwise matrix. Although the channel grouping technique can reduce the channel number from C to C/G, the overall computational complexity is still much higher than the original non-local operation. To mitigate this problem, this section proposes a compact representation that leads to an affordable approximation for GNL.

Let us denote $\boldsymbol{\theta} = \text{vec}(\mathbf{X}\mathbf{W}_\theta)$, $\boldsymbol{\phi} = \text{vec}(\mathbf{X}\mathbf{W}_\phi)$ and $\boldsymbol{g} = \text{vec}(\mathbf{X}\mathbf{W}_g)$, each of which is a NC-D vector column. Without loss of generality, we assume f is a general kernel function (*e.g.*, RBF, bilinear, etc.) that computes a $NC \times NC$ matrix composed by the elements,

$$\big[f(\boldsymbol{\theta}, \boldsymbol{\phi})\big]_{ij} \approx \sum_{p=0}^{P} \alpha_p^2 (\theta_i \phi_j)^p, \quad (9)$$

which can be approximated by Taylor series up to certain order P. The coefficient α_p can be computed in closed form once the kernel function is known. Taking RBF kernel for example,

$$[f(\boldsymbol{\theta}, \boldsymbol{\phi})]_{ij} = \exp(-\gamma\|\theta_i - \phi_j\|^2) \approx \sum_{p=0}^{P} \beta \frac{(2\gamma)^p}{p!} (\theta_i \phi_j)^p, \quad (10)$$

where $\alpha_p^2 = \beta\frac{(2\gamma)^p}{p!}$ and $\beta = \exp\big(-\gamma(\|x\|^2 + \|y\|^2)\big)$ is a constant and $\beta = \exp(-2\gamma)$ if the input vectors $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ are $\ell 2$-normalized. By introducing two matrices,

$$\boldsymbol{\Theta} = [\alpha_0\boldsymbol{\theta}^0, \cdots, \alpha_P\boldsymbol{\theta}^P] \in \mathbb{R}^{NC\times(P+1)}, \quad \boldsymbol{\Phi} = [\alpha_0\boldsymbol{\phi}^0, \cdots, \alpha_P\boldsymbol{\phi}^P] \in \mathbb{R}^{NC\times(P+1)} \quad (11)$$

our compact generalized non-local (CGNL) operation approximates Eq. 8 via a trilinear equation,

$$\text{vec}(\mathbf{Y}) \approx \boldsymbol{\Theta}\boldsymbol{\Phi}^\top \boldsymbol{g}. \quad (12)$$

At first glance, the above approximation still involves the computation of a large pairwise matrix $\boldsymbol{\Theta}\boldsymbol{\Phi}^\top \in \mathbb{R}^{NC\times NC}$. Fortunately, the order of Taylor series is usually relatively small $P \ll NC$. According to the associative law, we could alternatively compute the vector $\boldsymbol{z} = \boldsymbol{\Phi}^\top \boldsymbol{g} \in \mathbb{R}^{P+1}$ first and then calculate $\boldsymbol{\Theta}\boldsymbol{z}$ in a much smaller complexity of $\mathcal{O}(NC(P+1))$.

Complexity analysis: Table 1 compares the computational complexity of GNL block with CGNL. We cannot afford for directly computing GNL because of its huge complexity of $\mathcal{O}(2(NC)^2)$ in both time and space. Instead, our compact method dramatically eases the heavy calculation to $\mathcal{O}(NC(P+1))$. In Table 2, we investigate the memory cost for each model on one Tesla P40 GPU with 24GB memory. Our models are trained on a server of 8 P40s with 8 clips per batch per GPU (64 clips per total mini-batch). We compare on single GPU memory consumption beacuse each GPU is allocated by the same memory during training. This table shows our CGNL operation consumes less GPU memory compared to the NL networks in both 1-block and 5-block settings.

Table 1: Complexity comparison of GNL and CGNL operations, where N and C indicate the number of positions and channels respectively.

	General NL Method	CGNL Method
Strategy	$f(\mathbf{\Theta}\mathbf{\Phi}^\top)\boldsymbol{g}$	$\mathbf{\Theta}\mathbf{\Phi}^\top\boldsymbol{g}$
Time	$\mathcal{O}(2(NC)^2)$	$\mathcal{O}(NC(P+1))$
Space	$\mathcal{O}(2(NC)^2)$	$\mathcal{O}(NC(P+1))$

Table 2: A summary of memory cost during training. Symbol $\times$ means the models cannot be trained due to the memory explosion. The memory cost is measured by Gigabyte (GB) per GPU.

model	memory	model	memory
R-101	0.86	R-101	0.86
+ 1NL block	0.97	+ 5NL block	1.09
+ 1GNL block	$\times$	+ 5GNL block	$\times$
+ 1CGNL block	0.80	+ 5CGNL block	0.87

3.5 Implementation Details

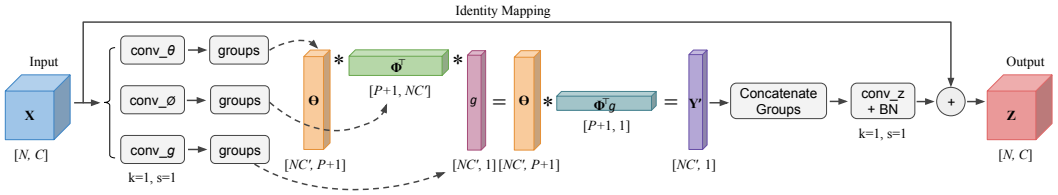

Figure 2: **Compact generalized non-local (CGNL) module with groups.** The feature maps are shown with the shape of their tensors, *e.g.*, $[C, N]$, where $N = THW$ or $N = HW$. The feature maps will be divided along channels into multiple groups after three conv layers whose kernel size and stride both equals 1 ($\mathrm{k} = 1, \mathrm{s} = 1$). The grouped channels dimension is changed into $C' = C/G$, where G is the group number. The compact representations for generalized non-local module are build within each group. P indicates the order of Taylor expansion for kernel functions.

Fig. 2 illustrates the workflow of how CGNL processes a feature map $\mathbf{X}$ of the size $N \times C$, where $N = H \times W$ or $N = T \times H \times W$. $\mathbf{X}$ is first fed into three $1 \times 1 \times 1$ convolutional layers that are described by the weights W_θ, W_ϕ, W_g respectively in Eq. 7. To improve the capacity of neural networks, the channel grouping idea [33, 32] is then applied to divide the transformed feature along the channel dimension into G groups. As shown in Fig. 2, we approximate for each group the GNL operation (Eq. 8) using the Taylor series according to Eq. 12. The output $\mathbf{Y}'$ from each group are concatenated together along the channel dimension to restore the full $\mathbf{Y}$.

To achieve generality and compatibility with existing neural network blocks, the CGNL block is implemented by wrapping Eq. 8 in an identity mapping of the input as in residual learning [7]:

$$\mathbf{Z} = BN(\mathbf{Y}\mathbf{W}_z) + \mathbf{X}, \tag{13}$$

where $\mathbf{W}_z \in \mathbb{R}^{C \times C}$ denotes a 1×1 or $1 \times 1 \times 1$ convolution layer followed by a Batch Normalization [10].

4 Experiments

4.1 Datasets

Our CGNL module is evaluated on multiple tasks, including fine-grained classification and action recognition. For fine-grained classification, we perform the experiments on the Birds-200-2011 (CUB) dataset [29], which contains 11788 images of bird 200 categories. For action recognition, we perform experiments on two challenging datasets, Mini-Kinetics [34] and UCF101 [24]. The Mini-Kinetics dataset contains 200 categories of data. Due to some invalid video links, we use 78265 videos for training and 4986 videos for validation. The UCF101 dataset contains 101 categories of action, which are separated into 25 groups with 4-7 videos of each action in a group.

4.2 Baselines

Given the steady performance and efficiency, the ResNet [7] series (ResNet-50 and ResNet-101) are adopted as our baselines. For video tasks, we keep the same architecture configuration with [31],

Table 3: **Ablations.** Top1 and top5 accuracy (%) on various validation datasets.

(a) Kernel function results of models with one CGNL module on CUB. The dot production achieves the best result. But the other kernel functions make accuracy be at the edge of baseline performance.

model	top1	top5
baseline	84.06	96.19
Dot Production	85.14	96.88
Gaussian RBF	84.10	95.78
Embedded Gaussian	84.01	96.08

(b) Comparison of validation results between NL and CGNL networks on UCF-101. Note that CGNL is not configured with channel grouping.

model	top1	top5
R-50	81.62	94.62
+ 1 NL block	82.88	95.74
+ 1 CGNL block	83.38	95.42

(c) The results of CGNL networks with channel grouping on CUB. A few groups help to boost the performance. But more groups tend to prevent the CGNL block from capturing the correlations between positions across channels.

model	groups	top1	top5
R-101	-	85.05	96.70
+ 1 CGNL block	1	86.17	97.82
	4	86.24	97.05
	8	86.35	97.86
	16	86.13	96.75
	32	86.04	96.69

model	groups	top1	top5
R-101	-	85.05	96.70
+ 5 CGNL block	1	86.01	95.97
	4	86.19	96.07
	8	86.24	97.23
	16	86.43	98.89
	32	86.10	97.13

(d) The results of CGNL networks with multiple groups on Mini-Kinetics dataset. More groups help the CGNL networks improve top1 accuracy obveriously.

model	gorups	top1	top5
R-50	-	75.54	92.16
+ 1 CGNL block	1	77.16	93.56
	4	77.56	93.00
	8	77.76	93.18

model	gorups	top1	top5
R-101	-	77.44	93.18
+ 1 CGNL block	1	78.79	93.64
	4	79.06	93.54
	8	79.54	93.84

where the temporal dimension is trivially addressed by max pooling. Following [31] the convolutional layers in the baselines are implemented as $1 \times k \times k$ kernels, and we insert our CGNL blocks into the network to turn them into compact generalized non-local (CGNL) networks. The configurations of adding 1 and 5 blocks are investigated. [31] suggests that adding 1 block on the 3-th stage is slightly better than the others. So our experiments of adding 1 block all target the 3-th stage of ResNet. The experiments of adding 5 blocks, on the other hand, are configured by inserting 2 blocks on the 3-th stage, and 3 blocks on the 4-th stage, to every other residual block in ResNet-50 and ResNet-101.

Training: For initilaizing the weights, we use the models pretrained on ImageNet [21]. We extract all the frames of a video in a *dense* manner. Following [31] to generate 32-frames input clips for models, first we randomly crop out 64 consecutive frames from the full-length video and then drop every other frame. The way to choose 32-frames input clips can be viewed as a data augmentation in the temporal domain. We view these 32-frame images as a group, the shorter side of each group is scaled by randomly sampling S from a certain range $[Smin, Smax]$ (we used $Smin = 256$ and $Smax = 320$). Then our spatial crop size in each group is $S = 224$. We use a weight decay of 0.0001 and momentum of 0.9. The dropout [25] with ratio 0.5 is inserted between average pooling layer and last fully-connected layer. To keep same with [31], we initialize the BatchNorm (BN) layer in CGNL and NL block with zero value for scale parameter [5]. To train the networks on CUB dataset, we follow the same training strategy above but the random crop method. We random crop an image in size of 448 after rescaling its shorter size into 512.

Inference: The models are tested immediately after training is finished. In [31], spatially fully-convolutional [15] inference [2] is used for NL networks. For these video clips, the shorter side is resized to 256 pixels and use 3 crops to cover the entire spatial size along the longer side. The final prediction is the averaged softmax scores of all clips. For fine-grined classification, we do 1 center-crop testing with input size 448.

4.3 Ablation Experiments

Kernel Functions: We use three popular kernel functions, namely dot production, embedded Gaussian and Gaussian RBF, in our ablation studies. For dot production, Eq. 12 will be held for direct computation. For embedded Gaussian, the α_p^2 will be $\frac{1}{p!}$ in Eq. 9. And for Gaussian RBF, the corresponding formula is defined as Eq. 10. Following the prior work [3], we expend the Taylor series with fourth order and the hyperparameter γ for RBF is set by $1e\text{-}4$. Table 3a suggests that

[2] https://github.com/facebookresearch/video-nonlocal-net

dot production is the best kernel functions for CGNL networks. Such experimental observations are consistent with [31]. The other kernel functions we used, Embedded Gaussion and Gaussian RBF, has a little improvements for performance. Therefore, we choose the dot production as our main experimental configuration for other tasks.

Grouping: The grouping strategy is another important technique. On Mini-Kinetics, Table 3d suggests that grouping can bring higher accuracy. The improvements brought in by adding groups are larger than those by reducing the channel reduction ratio. The best top1 accuracy is achieved by splitting into 8 groups for CGNL networks. On the other hand, however, it is worthwhile to see if more groups can always improve the results, and Table 3c gives the negative answer, where setting the group number larger than 8 will decrease the accuracy. This is actually expected, as the affinity in CGNL block considers the points across channels. When we split the channels into a few groups, it can facilitate the restricted optimization and ease the training. However, if too many groups are adopted, it hinder the affinity to capture the rich correlations between elements across the channels.

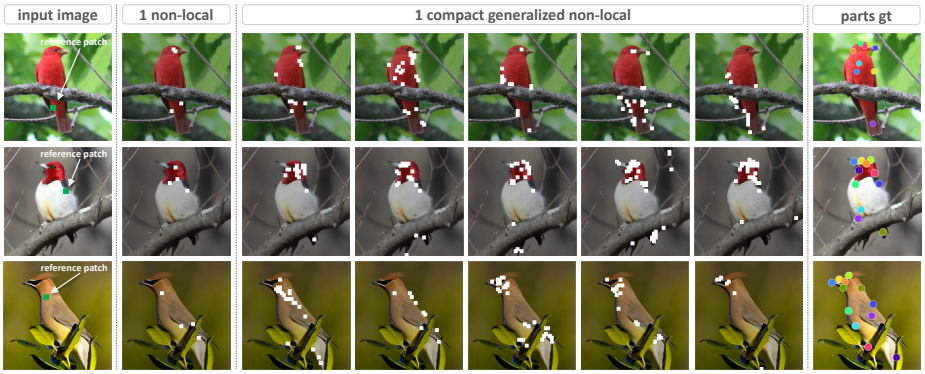

Figure 3: **Result analysis of the NL block and our CGNL block on CUB.** Column 1: the input images with a small *reference patch* (green rectangle), which is used to find the highly related patches (white rectangle). Column 2: the highly related clues for prediction in each feature map found by the NL network. The dimension of self-attention space in NL block is $N \times N$, where $N = HW$. So its visualization only has one column. Columns 3 to 7: the most related patches computed by our compact generalized non-local module. We first pick a reference position in the space of $\boldsymbol{g}$, then we use the corresponding vectors in $\boldsymbol{\Theta}$ and $\boldsymbol{\Phi}$ to compute the attention maps with a threshold. Last column: the ground truth of body parts. The high related areas of CGNL network can easily cover the standard parts that can provide the prediction clues.

4.4 Main Results

Table 4a shows that although adding 5 NL and CGNL blocks in the baseline networks can both improve the accuracy, the improvement of CGNL is larger. The same applies to Table 3b and Table 4b. Similar results are observed in experiments on UCF101 and CUB dataset, where adding 5 CGNL blocks provides the optimal results both for R-50 and R-101.

Table 4a shows the main results on Mini-Kinetics dataset. Compared to the baseline R-50 whose top1 is 75.54%, adding 1 NL block brings improvement by about 1.0%. Similar results can be found in the experiments based on R-101, where adding 1 CGNL provides about more than 2% improvement, which is larger than that of adding 1NL block. Table 3b shows the main results on the UCF101 dataset, where adding 1CGNL block achieves higher accuracy than adding 1NL block. And Table 4b shows the main results on the CUB dataset. To understand the effects brought by CGNL network, we show the visualization analysis as shown in Fig 3 and Fig 4. Additionly, to investigate the capacity and the generalization ability of our CGNL network. We test them on the task of object detection and instance segmentation. We add 1 NL and 1 CGNL block in the R-50 backbone for Mask-RCNN [6]. Table 4c shows the main results on COCO2017 dataset [13] by adopting our 1 CGNL block in the backbone of Mask-RCNN [6]. It shows that the performance of adding 1 CGNL block is still better than that of adding 1 NL block.

We observe that adding CGNL block can always obtain better results than adding the NL block with the same blocks number. These experiments suggest that considering the correlations between any

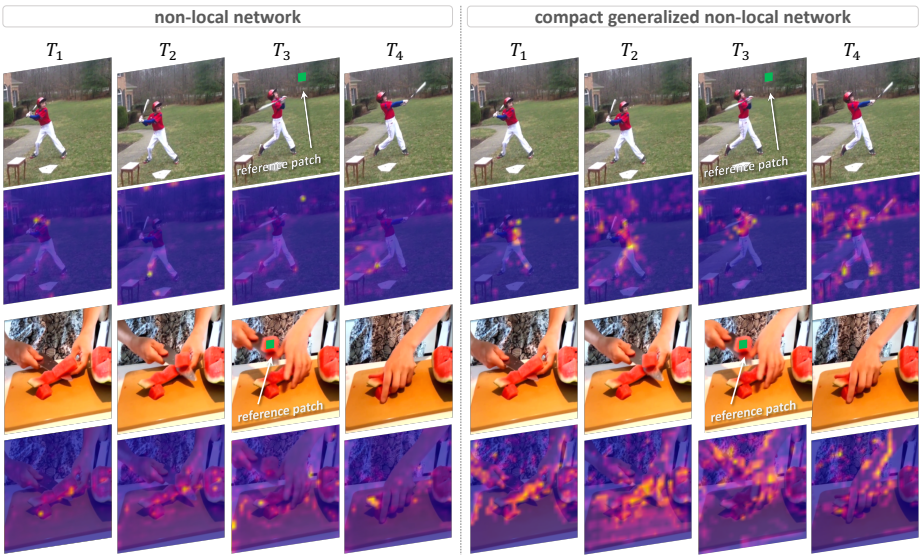

Figure 4: **Visualization with feature heatmaps.** We select a *reference patch* (green rectangle) in one frame, then visualize the high related ares by heatmaps. The CGNL network enjoys capturing dense relationships in feature space than NL networks.

Table 4: **Main results.** Top1 and top5 accuracy (%) on val datasets. And the box and mask AP on COCO.

(a) Main validation results of NL and CGNL networks on Mini-Kinetics. The CGNL networks is established with 8 groups.

model	top1	top5
R-50	75.54	92.16
+ 1 NL block	76.53	92.90
+ 1 CGNL block	77.76	93.18
+ 5 NL block	77.53	94.00
+ 5 CGNL block	78.79	94.37
R-101	77.44	93.18
+ 1 NL block	78.02	93.86
+ 1 CGNL block	79.54	93.84
+ 5 NL block	79.21	93.21
+ 5 CGNL block	79.88	93.37

(b) Classification results of NL and CGNL networks on CUB. The CGNL models are trained with 8 channel grouping.

model	top1	top5	model	top1	top5
R-50	84.06	96.19	R-101	85.05	96.70
+ 1 NL block	84.79	96.76	+ 1 NL block	85.49	97.04
+ 1 CGNL block	85.14	96.88	+ 1 CGNL block	86.35	97.86
+ 5 NL block	85.10	96.18	+ 5 NL block	86.10	96.35
+ 5 CGNL block	85.68	96.69	+ 5 CGNL block	86.24	97.23

(c) Main results on COCO2017. We add 1 NL block and 1 CGNL block to Mask R-CNN. We observe that the CGNL model achieve better results than NL one under the same training strategy.

model	AP^{box}	AP^{box}_{50}	AP^{box}_{75}	AP^{mask}	AP^{mask}_{50}	AP^{mask}_{75}
baseline	34.47	54.87	36.58	30.44	51.55	31.95
+ 1NL block	35.02	55.79	37.54	30.23	52.40	32.77
+ 1CGNL block	35.70	56.07	38.69	31.22	52.44	32.67

two positions across the channels can significantly improve the performance than that of original non-local methods.

5 Conclusion

We have introduced a simple approximated formulation of compact generalized non-local operation, and illustrate it on the task of fine-grained classification and action recognition from RGB images. Our formulation allows for explicit modeling of rich interdependencies between any positions across channels in the feature space. To ease the heavy computation of generalized non-local operation, we propose a compact representation with the simple matrix production by using Taylor expansion for multiple kernel functions. It is easy to implement and requires little additional parameters, making it an attractive alternative to the original non-local block, which only considers the correlations between two positions along the specific channel. Our model produces competitive or state-of-the-art results on various benchmarked datasets.

References

[1] J. Carreira, R. Caseiro, J. Batista, and C. Sminchisescu. Semantic segmentation with second-order pooling. In *European Conference on Computer Vision*, pages 430–443. Springer, 2012.

[2] F. Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint*, 2016.

[3] Y. Cui, F. Zhou, J. Wang, X. Liu, Y. Lin, and S. Belongie. Kernel pooling for convolutional neural networks. In *Computer Vision and Pattern Recognition (CVPR)*, 2017.

[4] Y. Gao, O. Beijbom, N. Zhang, and T. Darrell. Compact bilinear pooling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 317–326, 2016.

[5] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, large minibatch sgd: training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[6] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 2980–2988. IEEE, 2017.

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

[8] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[9] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*, 2017.

[10] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

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

[12] 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.

[13] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[14] 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.

[15] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[16] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[17] W. Luo, Y. Li, R. Urtasun, and R. Zemel. Understanding the effective receptive field in deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 4898–4906, 2016.

[18] F. Perronnin, J. Sánchez, and T. Mensink. Improving the fisher kernel for large-scale image classification. In *European conference on computer vision*, pages 143–156. Springer, 2010.

[19] N. Pham and R. Pagh. Fast and scalable polynomial kernels via explicit feature maps. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 239–247. ACM, 2013.

[20] T. Poggio and F. Girosi. Networks for approximation and learning. *Proceedings of the IEEE*, 78(9):1481–1497, 1990.

[21] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[22] B. Scholkopf and A. J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

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

[24] K. Soomro, A. R. Zamir, and M. Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

[25] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[26] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, et al. Going deeper with convolutions. Cvpr, 2015.

[27] I. Ustyuzhaninov, W. Brendel, L. A. Gatys, and M. Bethge. What does it take to generate natural textures? In *ICLR*, 2017.

[28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *NIPS*, 2017.

[29] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD birds-200-2011 dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

[30] H. Wang, A. Kläser, C. Schmid, and C.-L. Liu. Action recognition by dense trajectories. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3169–3176. IEEE, 2011.

[31] X. Wang, R. Girshick, A. Gupta, and K. He. Non-local neural networks. *arXiv preprint arXiv:1711.07971*, 2017.

[32] Y. Wu and K. He. Group normalization. *arXiv preprint arXiv:1803.08494*, 2018.

[33] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 5987–5995. IEEE, 2017.

[34] S. Xie, C. Sun, J. Huang, Z. Tu, and K. Murphy. Rethinking spatiotemporal feature learning for video understanding. *arXiv preprint arXiv:1712.04851*, 2017.

[35] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv preprint arXiv:1707.01083*, 2017.