Kernel Pooling for Convolutional Neural Networks

Yin Cui[1,2*] Feng Zhou[3] Jiang Wang[4] Xiao Liu[3] Yuanqing Lin[3] Serge Belongie[1,2]
[1]Department of Computer Science, Cornell University [2]Cornell Tech
[3]Baidu Research [4]Google Research
{ycui, sjb}@cs.cornell.edu www.f-zhou.com
wangjiangb@gmail.com {liuxiao12,linyuanqing}@baidu.com

Abstract

Convolutional Neural Networks (CNNs) with Bilinear Pooling, initially in their full form and later using compact representations, have yielded impressive performance gains on a wide range of visual tasks, including fine-grained visual categorization, visual question answering, face recognition, and description of texture and style. The key to their success lies in the spatially invariant modeling of pairwise (2nd order) feature interactions. In this work, we propose a general pooling framework that captures higher order interactions of features in the form of kernels. We demonstrate how to approximate kernels such as Gaussian RBF up to a given order using compact explicit feature maps in a parameter-free manner. Combined with CNNs, the composition of the kernel can be learned from data in an end-to-end fashion via error back-propagation. The proposed kernel pooling scheme is evaluated in terms of both kernel approximation error and visual recognition accuracy. Experimental evaluations demonstrate state-of-the-art performance on commonly used fine-grained recognition datasets.

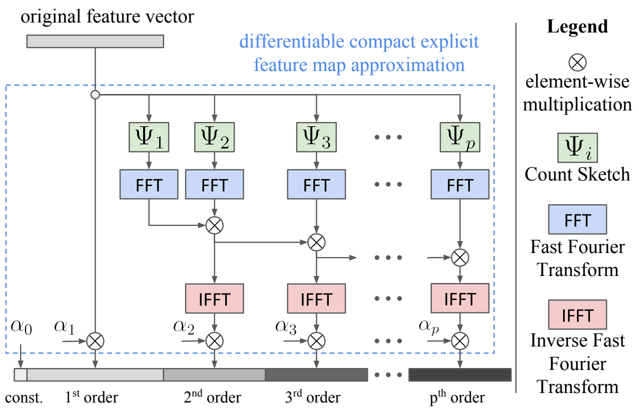

Figure 1. The proposed Kernel Pooling method. For a feature vector (i.e., the activation at a spatial location on the feature map, in the case of a CNN), we use Count Sketch [6] to generate a compact explicit feature map up to p^{th} order. After applying kernel pooling, the inner product between two features can capture high order feature interactions as in Eqn. 1. This makes the subsequent linear classifier highly discriminative. The proposed kernel pooling scheme is end-to-end trainable and the composition of the kernel can be learned through the update of coefficients $\{\alpha_i\}_{i=0}^p$. The vanilla compact bilinear pooling [11, 10] only use the 2nd order information as the feature vector.

1. Introduction

The idea of interactions between features has been used extensively as a higher order representation in learning tasks recently [24, 34, 3, 23]. The motivation behind is to make the subsequent linear classifier operates on higher dimensional feature map so that it becomes more discriminative. There are two ways in general to create higher order interactions. The most commonly used one is to *implicitly* map the feature via the kernel trick, like in the case of kernel SVM [41]. The disadvantages are twofold. The storage needed and the evaluation time are both proportional to the number of training data, which makes it inefficient on large datasets. In addition, the construction of the kernel makes it hard to use stochastic learning methods, including Stochastic Gradient Descent (SGD) in the training of CNNs. The other way is to *explicitly* map the feature vector into high dimensional space with products of features (monomials). The drawback of this method is obvious. If we want up to p^{th} order interactions on a d dimensional feature vector, the dimension of the explicit feature map will be $\mathcal{O}(d^p)$, which makes it impractical to use in real world applications. A common way to address these issues is to compactly approximate either kernel functions [37, 44] or feature maps [17, 31, 2].

Before the remarkable success of using Convolutional Neural Networks (CNNs) on visual data [20, 38, 39, 15], low-level hand-crafted features (*e.g.*, SIFT [25], HOG [8],

*Part of this work was done during the internship at Baidu Research.

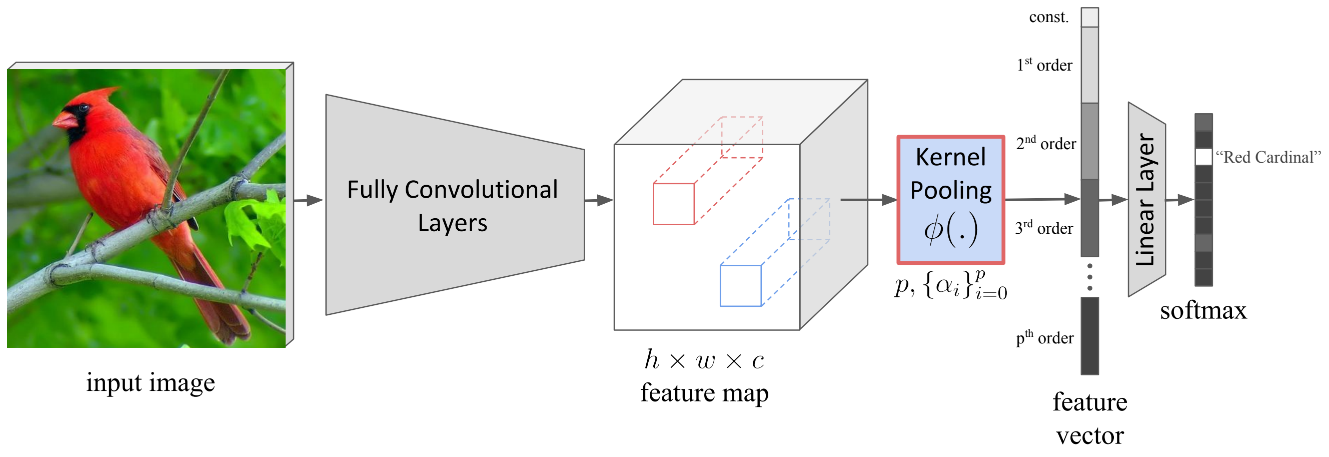

Figure 2. End-to-end training with the proposed pooling method. An input image is fed into a series of fully convolutional layers to get the output feature map of size $h \times w \times c$. For the c dimensional feature vector on every single spatial location (*e.g.*, the red or blue bar on the feature map), we apply the proposed kernel pooling method illustrated in Fig. 1. The final feature vector is average pooled over all locations $h \times w$. Then a linear layer with softmax is used to do the classification. The kernel is defined by the order p and coefficients $\{\alpha_i\}_{i=0}^p$, which can be learned from data through back-propagation.

Gist [28]) combined with mid-level feature aggregation or pooling methods (*e.g.*, Bag-of-visual-words, Spatial Pyramid Matching [21], Sparse Coding [45], Fisher Vector [30]) were widely adopted as the standard scheme for feature extraction. When learning and applying the subsequent linear classifier on extracted features, kernel methods such as Gaussian RBF or exponential χ^2 kernel are often adopted to capture higher order information and make linear classifier more discriminative. Recently, efforts in combining CNNs with 2nd order feature interactions, either by replacing hand-crafted features with CNN features [7] or jointly trained in an end-to-end fashion, yielded impressive performance gains on a wide range of visual tasks. Representative examples include fine-grained visual recognition [23, 11], visual question answering [10], texture representation and synthesis [13, 22], face recognition [35] and style transfer [12]. Notably, both Gao *et al.* [11] and Fukui *et al.* [10] used Tensor Sketch [31] to compactly compress the full bilinear vector by 2 orders of magnitude while preserve the same performance.

In this work, we propose a compact and differentiable way to generate explicit feature maps. We generalize the strategy used in [11, 10] to represent higher order feature interactions. For a feature vector $\mathbf{x}$ of dimension d, we generate its i^{th} order ($i \geq 2$) compact explicit feature map with Count Sketch [6] and circular convolution. In practice, people often operate circular convolution in frequency domain via Fast Fourier Transform (FFT) and Inverse Fast Fourier Transform (IFFT). It has been proven, both theoretically and practically in [31], that this method is able to compactly approximate polynomial kernels. As illustrated in Fig. 1, with a stack of Count Sketch, element-wise multiplication, FFT and IFFT units, higher order information can be compactly preserved. The kernel pooling method is applied on every single spatial location on the feature map of a CNN. And the final feature vector is the result of global average pooling across all spatial locations.

Denote the proposed kernel pooling method as ϕ. Then for two feature vectors $\mathbf{x}$ and $\mathbf{y}$, the inner product between $\phi(\mathbf{x})$ and $\phi(\mathbf{y})$ can approximate a kernel up to a certain order p as follows (see Sec. 3 for more details):

$$\phi(\mathbf{x})^\top \phi(\mathbf{y}) \approx \sum_{i=0}^{p} \alpha_i^2 (\mathbf{x}^\top \mathbf{y})^i \approx \mathcal{K}(\mathbf{x}, \mathbf{y}) \tag{1}$$

Through the introduction of kernel functions associated with Reproducing kernel Hilbert space, linear classifiers operate on high-dimensional Euclidean space become highly discriminative. Combine the proposed pooling method with a CNN, as shown in Fig. 2, the model can be trained end-to-end via back-propagation of classification errors. The composition of the kernel, as determined by coefficients $\{\alpha_i\}_{i=0}^p$, can be either predefined to approximate a certain kernel like Gaussian RBF up to order p or learned from data.

To sum up, there are two main contributions in this work. Firstly, we propose a general kernel pooling method via compact explicit feature mapping. Using the linear classifier on the feature map is approximately same as applying the kernel trick. Secondly, the proposed kernel pooling is differentiable and can be combined with a CNN for joint optimization. The composition of the kernel can also be learned simultaneously during the training.

2. Related Work

The proposed kernel pooling method relies on the existing efforts on low dimensional compact approximation of explicit feature maps. Rahimi *et al.* [33] is one of the first work on using random features for Gaussian and Laplacian kernels. Later, the similar idea was generalized to other kernels such as Maji *et al.* [26] for the histogram intersection kernel and Vedaldi *et al.* [42] for χ^2 kernel. On the compact approximation of polynomial kernels, recent proposed Random Maclaurin by Kar *et al.* [17], Tensor Sketch by Pham *et al.* [31] and Subspace Embedding by Avron *et al.* [2] are the most noticeable representatives. There is also a line of work that tries to learn higher order interactions from the data through optimization [24, 34, 3]. We differ from these work by the combination of Convolutional Neural Networks (CNNs) in an end-to-end fashion. With the joint optimization, we can leverage the powerful off-the-shelf fully convolutional network architectures to learn better features directly from data.

Since the dimension of p^{th} order pooled feature grows exponentially with p, the use of $p > 2$ in real world applications is often limited. In the case of $p = 2$, the model is usually referred as Bilinear models, first introduced by Tenenbaum and Freeman [40]. Bilinear models demonstrate impressive performance on visual tasks applied on both hand-crafted features [5] and learned features [23, 35, 22, 12]. Recently, fueled by compact 2nd order polynomial kernel approximation with Tensor Sketch [6, 31], same visual recognition performances can be preserved with much lower feature dimension [11] and new application on visual question answering is enabled [10]. We differ from these work by generalizing the compact representation from Bilinear models with 2nd order polynomial kernel to p^{th} order Taylor series kernel defined in Sec. 3. The composition of the kernel can also be learned through the end-to-end training with a CNN (see Sec. 3.3).

3. Kernel Pooling

We define the concept of "pooling" as the process of encoding and aggregating feature maps into a global feature vector. The architecture of Convolutional Neural Networks (CNNs) can be regarded as fully convolutional layers followed by the subsequent pooling layers and a linear classifier. Tab. 1 summaries pooling strategies adopted in commonly used CNN architectures. Typically people use a stack of fully connected layer with Rectified Linear Unit (ReLU) as in the case of AlexNet [20] and VGG [38]. Fully connected layers often perform well in general but introduce heavy computation and large number of parameters, hence makes the network slow and easy to overfit. The recently proposed Inception [39] and Residual Learning [15] only use global average pooling on the feature map. This strategy is more computationally efficient but it does not capture higher order feature interactions, which are believed crucial in many visual recognition tasks [23, 35, 22]. The bilinear models [5, 23] explicitly generate the c^2 dimensional feature map for 2nd order polynomial kernel, which is later compactly approximated in [11, 10] using Tensor Sketch [31]. In light of the success of Bilinear models, we propose an approach to go beyond Bilinear models and capture higher order feature interactions. We first define Tayler series kernel and show its explicit feature map can be compactly approximated. Then we demonstrate how to use the compact feature projection of Taylor series kernel to approximate commonly used kernels such as Gaussian RBF.

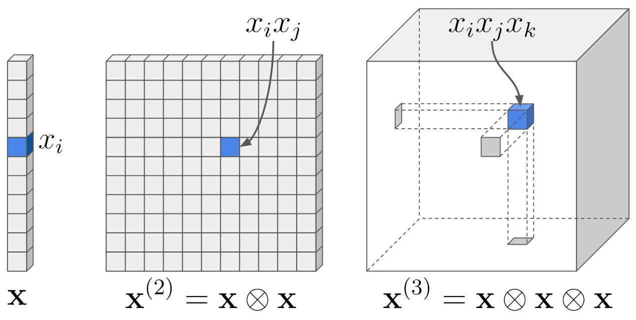
x_i
$x_i x_j$
$x_i x_j x_k$
$\mathbf{x}$
$\mathbf{x}^{(2)} = \mathbf{x} \otimes \mathbf{x}$
$\mathbf{x}^{(3)} = \mathbf{x} \otimes \mathbf{x} \otimes \mathbf{x}$

Figure 3. An illustration of tensor product. The p-level tensor product $\mathbf{x}^{(p)}$ of $\mathbf{x} \in \mathbb{R}^c$ is a c^p dimensional vector.

3.1. Explicit feature projection via Tensor product

Suppose the output feature map of a convolution layer is $X \in \mathbb{R}^{h \times w \times c}$ with height h, width w and number of channels c, we denote the c dimensional feature vector of a spatial location on X as $\mathbf{x} = [x_1, x_2, \ldots, x_c]^\top \in \mathbb{R}^c$.

The explicit feature projection $\phi(.)$ of a kernel function $\mathcal{K}(.,.)$ is defined by decomposing the the value of kernel function applied on two feature vectors $\mathbf{x}$ and $\mathbf{y}$ as the inner product between their feature maps:

$$\mathcal{K}(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})^\top \phi(\mathbf{y}) \tag{2}$$

Commonly used kernel functions include polynomial kernels $(\mathbf{x}^\top \mathbf{y})^p$, Gaussian RBF kernel $\exp(-\gamma \|\mathbf{x} - \mathbf{y}\|^2)$, χ^2 kernel $\sum_{i=1}^{c} \frac{2x_i y_i}{x_i + y_i}$, *etc*. Notice that some of the kernels may correspond to an infinite dimensional feature projection (*e.g.*, Gaussian RBF).

We introduce the concept of Tensor product and then demonstrate it can be used to get the explicit feature projection of a specific type of kernel called *Taylor series kernel*.

First, we define the 2-level tensor product (*i.e.*, outer product $\mathbf{x}\mathbf{x}^\top$) of $\mathbf{x}$ as:

$$\mathbf{x}^{(2)} = \mathbf{x} \otimes \mathbf{x} = \begin{bmatrix} x_1 x_1 & x_1 x_2 & \cdots & x_1 x_c \\ x_2 x_1 & x_2 x_2 & \cdots & x_2 x_c \\ \vdots & \vdots & \ddots & \vdots \\ x_c x_1 & x_c x_2 & \cdots & x_c x_c \end{bmatrix} \in \mathbb{R}^{c^2} \tag{3}$$

	AlexNet / VGG	Inception / ResNet	Bilinear	Compact Bilinear	**Ours**
Strategy	$\sigma(W_2\sigma(W_1X))$	$\frac{1}{hw}\sum_{i,j} X_{ij}$	$\frac{1}{hw}\sum_{i,j} X_{ij}X_{ij}^\top$	$\frac{1}{hw}\sum_{i,j} TS(X_{ij})$	$\frac{1}{hw}\sum_{i,j} \phi(X_{ij})$
Dimension	d	c	c^2	d	d
Time	$\mathcal{O}(hwcd)$	$\mathcal{O}(hwc)$	$\mathcal{O}(hwc^2)$	$\mathcal{O}(hw(c+d\log d))$	$\mathcal{O}(hwp(c+d\log d))$
Space	$\mathcal{O}(hwcd)$	0	0	$2c$	pc
Parameters	$\mathcal{O}(hwcd)$	0	0	0	0 or p

Table 1. A summary of pooling strategies adopted in commonly used CNN architectures. X represent the feature map of size $h \times w \times c$, where h, w and c is the height, width and number of channels; d represents the pre-specified feature dimension for the subsequent linear classifier and p is the order we used for the proposed kernel pooling. $\sigma(.)$, $TS(.)$ and $\phi(.)$ denotes the ReLU unit, Tensor Sketch [31] and the proposed kernel pooling mehtod, respectively.

Similarly, the p-level tensor product for $p \geq 2$ is defined as:

$$\mathbf{x}^{(p)} = \underbrace{\mathbf{x} \otimes \cdots \otimes \mathbf{x}}_{\text{p times}} \in \mathbb{R}^{c^p} \tag{4}$$

We also have $\mathbf{x}^{(0)} = 1$ and $\mathbf{x}^{(1)} = \mathbf{x}$. Fig. 3 illustrates the original feature vector $\mathbf{x}$ and its 2-level and 3-level tensor product $\mathbf{x}^{(2)}$ and $\mathbf{x}^{(3)}$. It has been shown in [36] that the p-level tensor product is the explicit feature projection of p^{th} order Polynomial kernel:

$$(\mathbf{x}^\top \mathbf{y})^p = (\mathbf{x}^{(p)})^\top (\mathbf{y}^{(p)}) \tag{5}$$

We define the Taylor series kernel of order p as follows:

$$\mathcal{K}_{\text{Taylor}}(\mathbf{x}, \mathbf{y}) = \sum_{i=0}^{p} \alpha_i^2 (\mathbf{x}^\top \mathbf{y})^i \tag{6}$$

Since the non-negative linear combination of kernels is still a kernel [36], the Taylor series kernel is a valid kernel as it can be expressed as non-negative linear combinations of Polynomial kernels.

It is clear to see that the explicit feature projection of Taylor series kernel is given by:

$$\phi_{\text{Taylor}}(\mathbf{x}) = [\alpha_0(\mathbf{x}^{(0)})^\top, \ldots, \alpha_p(\mathbf{x}^{(p)})^\top]^\top \tag{7}$$

Composed by the concatenation of scaled tensor products $\{\alpha_i \mathbf{x}^{(i)}\}_{i=0}^{p}$, $\phi(\mathbf{x})$[1] is a long feature vector with dimension $\mathcal{O}(c^p)$. Even in the case of $c = 512$ and $p = 3$, c^p is still larger than 10^8. Such a high dimension hinders its applications in any real world problems. Therefore, a compact approximation method is needed.

3.2. Compact approximation

The compact approximation method is differentiable and has good time and space complexity. There are several recently proposed work on kernel approximation with random feature projections [33, 17, 31, 2]. We build our approximation method on Tensor Sketching [31], because it consumes less time and space compared to [33, 17], and it is easier to implement compared to [2].

[1]For simplicity, unless otherwise specified, we will drop the subscript of $\mathcal{K}_{\text{Taylor}}$ and ϕ_{Taylor} in the remainder of the paper.

Algorithm 1: Count Sketch for Taylor series kernel

```
Input: $\mathbf{x} \in \mathbb{R}^c$, $p$, $\{d_i\}_{i=2}^p$, $\{\alpha_i\}_{i=0}^p$
Output: $\phi(\mathbf{x}) \in \mathbb{R}^d$, where $d = 1 + c + \sum_{i=2}^p d_i$, s.t.
        $\phi(\mathbf{x})^\top \phi(\mathbf{y}) \approx \mathcal{K}(\mathbf{x}, \mathbf{y}) = \sum_{i=0}^p \alpha_i^2 (\mathbf{x}^\top \mathbf{y})^i$.
1 Initialization: $\phi(\mathbf{x}) \leftarrow [\alpha_0^2, \mathbf{x}^\top]^\top$, $\mathcal{P} \leftarrow 1$.
2 for $t \leftarrow 1$ to $p$ do
3     Generate 2 independent hash functions $h_t$ and $s_t$.
      The outputs of $h_t$ and $s_t$ are uniformly drawn
      from $\{1, 2, \ldots, d_t\}$ and $\{+1, -1\}$, respectively.
4     Calculate the Count Sketch of $\mathbf{x}$ as $\mathcal{C}_t(\mathbf{x}) =$
      $[c_1, c_2, \ldots, c_{d_t}]^\top$, where $c_i = \sum_{i:h_t(i)=j} s_t(i)\mathbf{x}_i$.
5     $\mathcal{P} \leftarrow \mathcal{P} \circ \text{FFT}(\mathcal{C}_t(\mathbf{x}))$
6     if $t \geq 2$ then
7         $\phi(\mathbf{x}) \leftarrow$ concatenate $(\phi(\mathbf{x}), \text{FFT}^{-1}(\mathcal{P}))$
8 return $\phi(\mathbf{x})$
```

3.2.1 Taylor series kernel

To compactly approximate the p-level tensor product $\mathbf{x}^{(p)}$, we define the Count Sketch [6] of $\mathbf{x}$ as:

$$\mathcal{C}(\mathbf{x}) = [c_1, c_2, \ldots, c_d]^\top, \text{where } c_i = \sum_{i:h(i)=j} s(i)\mathbf{x}_i \tag{8}$$

The Count Sketch $\mathcal{C}(\mathbf{x})$ is a d-dimensional vector calculated using 2 hash functions $h(.)$ and $s(.)$. Their outputs are uniformly drawn from $\{1, 2, \ldots, d\}$ and $\{+1, -1\}$, respectively. The p-level tensor product $\mathbf{x}^{(p)}$ can then be approximated as:

$$\tilde{\mathbf{x}}^{(p)} = \text{FFT}^{-1}(\text{FFT}(\mathcal{C}_1(\mathbf{x})) \circ \cdots \circ \text{FFT}(\mathcal{C}_p(\mathbf{x}))) \tag{9}$$

where $\mathcal{C}_i(\mathbf{x})$ is the Count Sketch calculated from $2i$ independent hash functions $h_1, h_2, \ldots, h_i$ and $s_1, s_2, \ldots, s_i$, $\circ$ denotes the element-wise multiplication, FFT and FFT^{-1} is the Fast Fourier Transform and its Inverse.

Combining Eqn. 7 and Eqn. 9, the feature map of a Taylor series kernel can be compactly approximated, as described in Alg. 1. Inputs include the original feature vector $\mathbf{x}$, the order p of the Taylor series kernel to be approximated, target feature dimensions $d_i (i \geq 2)$ we want to use

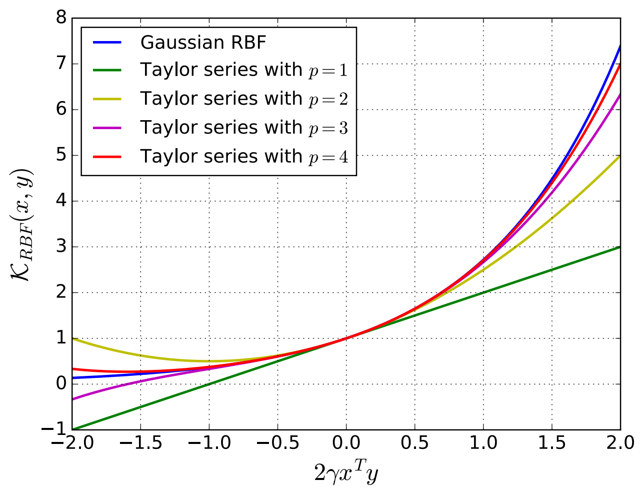

Figure 4. Approximating Gaussian RBF kernel by Taylor series kernel with variant p. Without loss of generality, we ignore the constant β when plotting. The approximation error depends on the inner product value $\mathbf{x}^\top\mathbf{y}$ and γ. With the proper choice of γ based on $\mathbf{x}^\top\mathbf{y}$, using $p=4$ would be sufficient to approximate Gaussian RBF.

for estimating $\mathbf{x}^{(i)}$ and its associated coefficient α_i. Compared with the explicit feature map in Eqn. 7, we reduce the feature dimension from exponential to linear. More specifically, from $\sum_{i=0}^{p} c^i$ to $d = 1 + c + \sum_{i=2}^{p} d_i$, where $d \ll c^i$, $\forall i \geq 2$.

It has been proved that $\tilde{\mathbf{x}}^{(p)}$ in Eqn. 9 is an unbiased feature map estimator for pth order Polynomial kernel. The relative estimation error can be bounded by Chebyshev's inequality (see Lemma 7 in [31] for the detailed proof). Similarly, the estimation error of using Alg. 1 can be bounded as:

$$\mathbf{P}\left[\left|\phi(\mathbf{x})^\top\phi(\mathbf{y}) - \mathcal{K}(\mathbf{x},\mathbf{y})\right| \geq \epsilon\mathcal{K}(\mathbf{x},\mathbf{y})\right] \leq \frac{1}{d_{min}\epsilon^2}\Delta(p) \tag{10}$$

where $d_{min} = \min(d_2, \ldots, d_p)$ and

$$\Delta(p) = \begin{cases} 2(p-1), & \text{if } C = \pm 1 \\ \frac{2C^2(C^{2p}-1)}{C^2-1}, & \text{otherwise} \end{cases}$$

$C = \frac{1}{\cos\theta}$ is a constant that equals to the reciprocal of the cosine similarity between two feature vectors $\mathbf{x}$ and $\mathbf{y}$. In our experience, we find higher dimensional feature (large d_{min}) gives better approximation, kernels with larger p introduce larger error, and the error bound also depends heavily on the angle between two feature vectors.

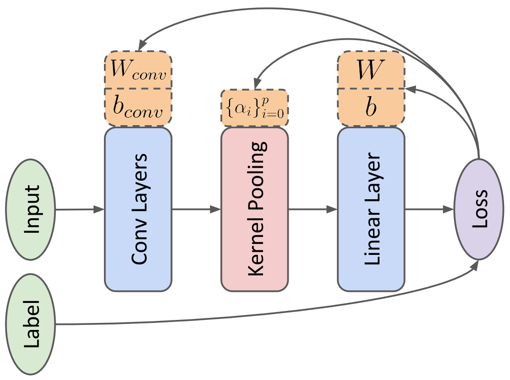

Figure 5. Learning kernel composition by end-to-end training with a CNN. The coefficients of the kernel are jointly learned together with weights of other CNN layers via back-propagation of the loss (denoted by outgoing arrows from "Loss").

3.2.2 Gaussian RBF kernel

The Taylor expansion of Gaussian RBF kernel [32] can be expressed as:

$$\begin{aligned} \mathcal{K}_{RBF}(\mathbf{x},\mathbf{y}) &= \exp\left(-\gamma\|\mathbf{x}-\mathbf{y}\|^2\right) \\ &= \exp\left(-\gamma(\|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 - 2\mathbf{x}^\top\mathbf{y})\right) \\ &= \beta\exp\left(2\gamma\mathbf{x}^\top\mathbf{y}\right) \\ &= \sum_{i=0}^{\infty}\beta\frac{(2\gamma)^i}{i!}(\mathbf{x}^\top\mathbf{y})^i \end{aligned} \tag{11}$$

where $\beta = \exp\left(-\gamma(\|\mathbf{x}\|^2 + \|\mathbf{y}\|^2)\right)$ is a constant and $\beta = \exp(-2\gamma)$ if $\mathbf{x}$ and $\mathbf{y}$ are ℓ_2-normalized. Compared with Taylor series kernel in Eqn. 6, it is clear that Taylor series kernel can be used to approximate Gaussian RBF term by term up to order p by setting α_i^2 as $\beta\frac{(2\gamma)^i}{i!}$. Other kernels can also be approximated if they have a Taylor expansion in the similar form. Fig. 4 illustrates the approximation of Gaussian RBF by Taylor series kernel with variant p. The approximation error depends on the inner product value $\mathbf{x}^\top\mathbf{y}$. In general, the closer the value is to 0, the smaller the approximation error. So we need to choose γ carefully based on $\mathbf{x}^\top\mathbf{y}$. With the proper choice of γ, using $p=4$ would be sufficient to approximate Gaussian RBF. Experiments on kernel approximation error and the effect of γ will be discussed extensively in Sec. 4.2.

3.3. Learning kernel composition end-to-end

The proposed kernel pooling method in Alg. 1 relies on simple computations with a set of fixed hash functions $\{h_t\}$ and $\{s_t\}$, FFT and FFT^{-1}, which are all differentiable. Combined with a CNN, the loss from the softmax layer can go through the proposed kernel pooling layer and be propagated back to the preceding fully convolution layers.

Instead of using fixed pre-defined coefficients to approximate a certain kernel such as Gaussian RBF, the composition of the kernel can be learned from data, as illustrated in Fig. 5. Designing and choosing a good kernel is a challenging task because it is hard to probe the underlying distribution of high-dimensional features. Therefore, a kernel function is often chosen empirically or through cross-validation. By jointly learning the kernel composition together with CNN weights in an end-to-end fashion, we argue the learned kernel is more adaptive and suitable to the data we are working on.

4. Experimental Evaluations

The proposed kernel pooling method is evaluated in terms of both kernel approximation error and visual recognition accuracy. Sec. 4.1 introduces experiment setup and baseline methods. Then, in Sec. 4.2, we run a comprehensive study of kernel approximation quality on CNN features. We also investigate the configuration such as the choice of feature dimension $\bar{d}$, kernel order p and γ. Sec. 4.3 is the major part of the experiment, in which we present extensive evaluations on various visual recognition tasks, including the recognition of bird [43], car [19], aircraft [27] and food [4]. The proposed kernel pooling method achieves state-of-the-art results on all datasets.

4.1. Experiment setup

We evaluate all pooling strategies listed in Tab. 1. For CNN architectures, we use VGG-16 [38] and ResNet-50 [15], both of which achieved state-of-the-art performance on ImageNet [9]. VGG-16 has 13 convolution with ReLU layers and 3 fully connected layers including the final linear layer with softmax for classification. ResNet-50 consists of 49 convolution layers followed by global average pooling and the final linear softmax layer. Both VGG and ResNet reduce the spatial resolution of the input image by a factor of $2^5 = 32$ during the convolution. In the case of Bilinear, Compact Bilinear and our model, we keep the fully convolutional part of the network and use the output feature map from the last convolution layer (*i.e.*, the feature vector $\mathbf{x}$ in Alg. 1 corresponds to the activation at each spatial location of last layer's feature map). For standard pooling methods, we choose VGG-16 and ResNet-50 [15] as representatives for fully connected pooling and global average pooling, respectively. The performance of VGG-16 and ResNet-50 is reported by fine-tuning the entire network from ImageNet pre-trained weights.

4.1.1 Pooling methods

We compare the performance of kernel pooling methods with the following baselines:

VGG with fully connected pooling (**VGG**): This is the original VGG-16 network proposed in [38]. The architecture of VGG-16 is a generalization of the ground-breaking AlexNet [20]. In AlexNet, only one convolution layer is applied to the input image and the feature map of a specific spatial resolution. In VGG, however, more convolution layers (2 to 3) are applied for each spatial resolution, which achieved state-of-the-art performance on ImageNet Challenge 2014. Both AlexNet and VGG use the same fully connected pooling scheme (a stack of two fully connected with ReLU layers) for the subsequent softmax layer. Due to the fixed number of nodes designed in fully connected layers, VGG requires a fixed input image size of 224×224. For each of the dataset, we replace the last linear layer of VGG to match the number of categories and then fine-tune the whole network from ImageNet pre-trained weights.

Residual Learning with average pooling (**ResNet**): Although the fully connected layer works well in practice, it has several drawbacks including the heavy computation and large storage needed as well as the tend to overfit. Recently proposed deeper networks based on Inception module [39] and Residual module [15] use global average pooling after convolution layers for the subsequent linear classifier. The global average pooling is lightweight, capable of taking input of any size and parameter-free. However, it fails to capture nonlinear information in feature maps. We choose a strong baseline of fine-tuned ResNet as comparison.

Bilinear Pooling (**BP**): We apply full bilinear pooling on top of the conv$_{5\_3}$ feature map from VGG-16, which is same as the best-performed B-CNN [D, D] in [23]. The feature dimension of the bilinear vector is $d = 512 \times 512 \approx 260$K. We don't combine ResNet with bilinear pooling because ResNet has 2048 channels in the final feature map. The brute force bilinear vector has the dimension of $2048 \times 2048 \approx 4.2$M, which is too large to use in practice.

Compact Bilinear Pooling (**CBP**): We use Tensor Sketch with fixed hash functions to approximate bilinear vector on the feature map of VGG-16 and ResNet-50. Whereas the original paper [11] only used VGG-16. Typically, compact bilinear pooling can achieve same performance as full bilinear pooling with $d \geq 8192$, reducing the original feature dimension by orders of magnitude. For a fair comparison, we set the feature dimension in CBP to be the same as our kernel pooling method in all experiments.

The proposed Kernel Pooling (**KP**): We evaluate the proposed kernel pooling method in the same context as BP and CBP. For the activation $\mathbf{x}$ at each spatial location on the feature map, we apply Alg. 1 to get the compact feature map $\phi(\mathbf{x})$. Same as BP and CBP, the final feature vector is average pooled across all the spatial locations. The composition of the kernel is evaluated with learned coefficients via back-propagation. The choice of kernel order p, feature dimension d and γ will be discussed in Sec. 4.2.

4.1.2 Implementation

Our implementation follows the commonly used practice in [20, 38, 23, 11]. We have two image input sizes: 224×224 and 448×448. For each image input size $S \times S$, we first subtract it with the pixel-wise image mean, and we resize the original image so that its shorter side is S while keeping its aspect ratio. Then we crop a $S \times S$ square image from the original image. During training, a random square image is cropped. Both the original crop and its horizontal flip are utilized for data augmentation. During inference, the center image is cropped. We pass the original crop and its horizontal flip to the CNN independently. The average of their classification scores is our final classification score.

We follow the post-processing steps in [23, 11] to the feature vector $\mathbf{y}$ before the linear classifier, because the experiments show that it improves fine-grained recognition performance. We apply element-wise signed square root: $\mathbf{y} \leftarrow sign(\mathbf{y})\sqrt{|\mathbf{y}|}$ followed by ℓ_2 normalization: $\mathbf{y} \leftarrow \mathbf{y}/\|\mathbf{y}\|$ on the compact feature $\mathbf{y}$ vector.

For the sake of faster convergence and better performance, we use pre-trained weights for the neural network. The intial weights of the convolutional layers are pre-trained on ImageNet classification dataset, and the initial weights of the final linear classifier is obtained by training a logistic regression classifier on the compact kernel pooling of pre-trained CNN features. We start the fine-tuing with 10x smaller learning rate (*i.e.* 0.001 for VGG and 0.01 for ResNet) and divide it by 10 after every 30 epochs. We use a momentum of 0.9 and a weight decay of 0.0005 for VGG and 0.0001 for ResNet. The training usually converges at around 50 epochs. The model diverges due to large gradients sometimes. Therefore, gradient clipping [29] is applied to ensure all gradients fall in the range between -1 and $+1$.

We use Tensorflow [1] to implement and train all the models. On a single NVIDIA Tesla K40 GPU, the forward and backward time of both VGG-16 and ResNet-50 with kernel pooling is about 500ms on a 448×448 image and 100ms on a 224×224 image. Kernel pooling requires around 50ms with $d = 4096$ and $p = 4$.

4.2. Kernel approximation and configurations

This subsection presents the experiments on kernel approximation error using Alg. 1 on CNN features. Using VGG-16 trained on ImageNet, we extract conv$_{5\_3}$ feature maps on the training set of CUB-200-2011 [43], with input size of 224×224. For each spatial location in the feature map, the feature is a $c = 512$ dimensional vector. Without loss of generality, we use the same feature pooling dimension $\bar{d}$ for each order in kernel pooling (*i.e.*, $d_i = \bar{d}$ for $i \geq 2$). Therefore, the final feature dimension is $d = 1 + c + \sum_{i=2}^{p} \bar{d} = 513 + (p-1)\bar{d}$. Fig. 6 shows the relative approximation error of Gaussian RBF kernel in log scale, with variant feature pooling dimension $\bar{d}$, order p and

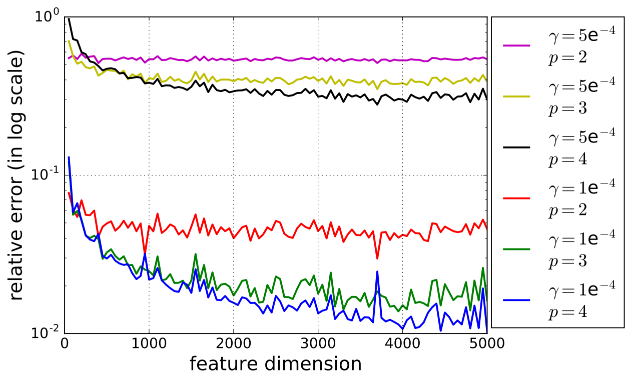

Figure 6. Relative approximation error for Gaussian RBF kernel applied on CNN features with variant kernel configurations.

γ. The relative approximation error between two feature vector $\mathbf{x}$ and $\mathbf{y}$ is given by:

$$\epsilon = \frac{|\phi(\mathbf{x})^\top \phi(\mathbf{y}) - \mathcal{K}_{RBF}(\mathbf{x}, \mathbf{y})|}{\mathcal{K}_{RBF}(\mathbf{x}, \mathbf{y})} \tag{12}$$

We compare kernel pooing with the feature dimension $\bar{d}$ from 50 to 5000 with the step of 50. Each data point is the averaged error on 100K randomly selected feature pairs.

From Fig. 6, we have the following observations: higher feature pooling dimension gives better approximation in general; approximation error also goes down with increasing order p; γ plays a key role in the approximation error. The above findings verify the insights from Eqn. 10. In Fig. 4 we can see that with sufficient feature dimension and order as well as a proper γ, we can achieve close to 1% relative error. In light of this, we use $\bar{d} = 4096$ and $p = 4$ for all the following experiments. The output vector has a dimension of $d = 1 + 512 + 3 \times 4096 = 12801$ for VGG, and $1 + 2048 + 3 \times 4096 = 14337$ for ResNet. The hyperparameter γ is set as the reciprocal of the mean of inner products between feature vectors in the training set to ensure that $\gamma \mathbf{x}^\top \mathbf{y}$ is small on average and we can get a good kernel approximation.

4.3. Visual recognition

We evaluate on the following visual recognition tasks.

Bird species recognition: We use CUB-200 dataset [43] for this task. The dataset consists of 11,788 images from 200 bird species. Each category has around 30 images for both training and testing.

Car make, model, year classification: The Stanford Car dataset [19] is used for this task. It has 16,185 images of 196 classes with car make, model and year.

Aircraft classification: The fine-grained aircraft dataset [27] was first introduced in FGComp 2013 challenge, which contains 100 aircraft categories and each has 100 images.

Dataset	CNN	Original	BP[23]	CBP[11]	KP	Others	
CUB [43]	VGG-16 [38]	73.1*	84.1	84.3	**86.2**	82.0	84.1
	ResNet-50 [15]	78.4	N/A	81.6	84.7	[18]	[16]
Stanford Car [19]	VGG-16	79.8*	91.3	91.2	**92.4**	**92.6**	82.7
	ResNet-50	84.7	N/A	88.6	91.1	[18]	[14]
Aircraft [27]	VGG-16	74.1*	84.1	84.1	**86.9**	80.7	
	ResNet-50	79.2	N/A	81.6	85.7	[14]	
Food-101 [4]	VGG-16	81.2	82.4	82.4	84.2	50.76	
	ResNet-50	82.1	N/A	83.2	**85.5**	[4]	

Table 2. Performance comparisons among all baselines, where KP is the proposed kernel pooling method with learned coefficients. Following the standard experimental setup, we use the input size of 448×448 for CUB, Stanford Car and Aircraft datasets except the original VGG-16 (marked by an asterisk *), which requires a fixed input size of 224×224. For Food-101, we use the input size of 224×224 for all the baselines.

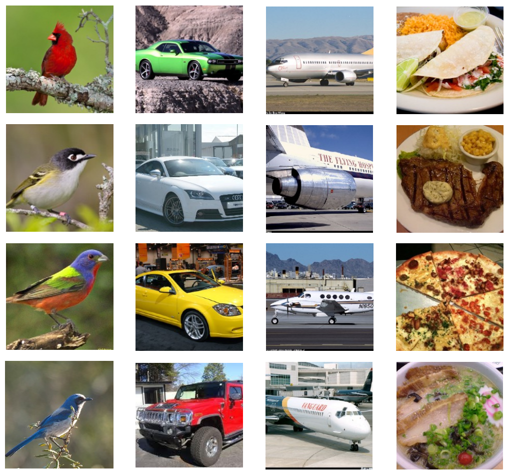

Figure 7. Images we used for visual recognition. From left to right, each column contains examples from CUB Bird [43], Stanford Car [19], Aircraft [27] and Food-101 [4].

Food recognition: For this task we use Food-101 dataset [4], which is by far the largest publicly available food recognition dataset to the best of our knowledge. This is a large-scale dataset with $101,000$ images and 1000 images per each category. This dataset is challenging because the training images are noisy and the background is not clean.

Sample images for each task are shown in Fig. 7. Performance comparison with all the baselines and state-of-the-art methods is presented in Tab. 2. The proposed Kernel Pooling with learned coefficients outperforms all other baselines by a large margin (around 1-3%) on all the datasets.

4.4. Discussion

In this subsection, we discuss the relative importance of higher order information for different CNN architectures. We examined learned kernel coefficients on CUB dataset with kernel pooling on VGG and ResNet. We found that high order feature interactions, especially 2^{nd} and 3^{rd} order, are weighted more in VGG compared with ResNet. In ResNet, there is no obvious distinction among first 3 orders. We believe this is due to the difference of the underlying network architectures.

One reason might be that in VGG, the non-linear feature interactions are mainly captured by fully-connected layers. So removing the fully-connected layers significantly degrade the original 1^{st} order feature. Since ResNet only use a global average pooling layer and has a very large receptive field, the features at different locations of the feature map is encouraged to represent similar information. Together with the residual module and a much deeper convolutional architecture, the output convolution feature could implicitly capture more information than VGG. In our experiments, we find that the performance of both VGG-16 and ResNet-50 can be improved when the proposed kernel pooling method is utilized. These experiments verify the effectiveness of using high-order feature interactions in the context of CNN.

5. Conclusion

In this paper, we have introduced a novel deep kernel pooling method as a high-order representation for visual recognition. The proposed method captures high-order and non-linear feature interactions via compact explicit feature mapping. The approximated representation is fully differentiable, thus the kernel composition can be learned together with a CNN in an end-to-end manner. Extensive experiments demonstrate that deep kernel pooling method achieves state-of-the-art performance on various fine-grained recognition tasks.

Acknowledgements

This work was supported in part by Google Focused Research Award, AWS Cloud Credits for Research, Microsoft Research Award and a Facebook equipment donation.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016. 7

[2] H. Avron, H. Nguyen, and D. Woodruff. Subspace embeddings for the polynomial kernel. In *NIPS*, 2014. 1, 3, 4

[3] M. Blondel, M. Ishihata, A. Fujino, and N. Ueda. Polynomial networks and factorization machines: New insights and efficient training algorithms. In *ICML*, 2016. 1, 3

[4] L. Bossard, M. Guillaumin, and L. Van Gool. Food-101–mining discriminative components with random forests. In *ECCV*, 2014. 6, 8

[5] J. Carreira, R. Caseiro, J. Batista, and C. Sminchisescu. Semantic segmentation with second-order pooling. In *ECCV*, 2012. 3

[6] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*, 2002. 1, 2, 3, 4

[7] M. Cimpoi, S. Maji, and A. Vedaldi. Deep filter banks for texture recognition and segmentation. In *CVPR*, 2015. 2

[8] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, 2005. 1

[9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 6

[10] A. Fukui, D. H. Park, D. Yang, A. Rohrbach, T. Darrell, and M. Rohrbach. Multimodal compact bilinear pooling for visual question answering and visual grounding. *arXiv preprint arXiv:1606.01847*, 2016. 1, 2, 3

[11] Y. Gao, O. Beijbom, N. Zhang, and T. Darrell. Compact bilinear pooling. In *CVPR*, 2016. 1, 2, 3, 6, 7, 8

[12] L. A. Gatys, A. S. Ecker, and M. Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015. 2, 3

[13] L. A. Gatys, A. S. Ecker, and M. Bethge. Texture synthesis and the controlled generation of natural stimuli using convolutional neural networks. In *NIPS*, 2015. 2

[14] P.-H. Gosselin, N. Murray, H. Jégou, and F. Perronnin. Revisiting the fisher vector for fine-grained classification. *Pattern Recognition Letters*, 2014. 8

[15] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 3, 6, 8

[16] M. Jaderberg, K. Simonyan, A. Zisserman, et al. Spatial transformer networks. In *NIPS*, 2015. 8

[17] P. Kar and H. Karnick. Random feature maps for dot product kernels. In *AISTATS*, 2012. 1, 3, 4

[18] J. Krause, H. Jin, J. Yang, and L. Fei-Fei. Fine-grained recognition without part annotations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5546–5555, 2015. 8

[19] J. Krause, M. Stark, J. Deng, and L. Fei-Fei. 3d object representations for fine-grained categorization. In *ICCV Workshop*, 2013. 6, 7, 8

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 1, 3, 6, 7

[21] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, 2006. 2

[22] T.-Y. Lin and S. Maji. Visualizing and understanding deep texture representations. In *CVPR*, 2016. 2, 3

[23] T.-Y. Lin, A. RoyChowdhury, and S. Maji. Bilinear cnn models for fine-grained visual recognition. In *ICCV*, 2015. 1, 2, 3, 6, 7, 8

[24] R. Livni, S. Shalev-Shwartz, and O. Shamir. On the computational efficiency of training neural networks. In *NIPS*, 2014. 1, 3

[25] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 2004. 1

[26] S. Maji and A. C. Berg. Max-margin additive classifiers for detection. In *ICCV*, 2009. 3

[27] S. Maji, E. Rahtu, J. Kannala, M. Blaschko, and A. Vedaldi. Fine-grained visual classification of aircraft. *arXiv preprint arXiv:1306.5151*, 2013. 6, 7, 8

[28] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *IJCV*, 2001. 2

[29] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *ICML*, 2013. 7

[30] F. Perronnin, J. Sánchez, and T. Mensink. Improving the fisher kernel for large-scale image classification. In *ECCV*, 2010. 2

[31] N. Pham and R. Pagh. Fast and scalable polynomial kernels via explicit feature maps. In *KDD*, 2013. 1, 2, 3, 4, 5

[32] T. Poggio and F. Girosi. Networks for approximation and learning. *Proceedings of the IEEE*, 78(9):1481–1497, 1990. 5

[33] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *NIPS*, 2007. 3, 4

[34] S. Rendle. Factorization machines with libfm. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2012. 1, 3

[35] A. RoyChowdhury, T.-Y. Lin, S. Maji, and E. Learned-Miller. Face identification with bilinear cnns. *arXiv preprint arXiv:1506.01342*, 2015. 2, 3

[36] B. Schölkopf and A. J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002. 4

[37] D. Scholkopf, F. Achlioptas, and M. Bernhard. Sampling techniques for kernel methods. *NIPS*, 2002. 1

[38] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1, 3, 6, 7, 8

[39] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015. 1, 3, 6

[40] J. B. Tenenbaum and W. T. Freeman. Separating style and content with bilinear models. *Neural computation*, 2000. 3

[41] V. Vapnik. *The nature of statistical learning theory*. Springer Science & Business Media, 2013. 1

[42] A. Vedaldi and A. Zisserman. Efficient additive kernels via explicit feature maps. *PAMI*, 2012. 3

[43] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The caltech-ucsd birds-200-2011 dataset. *California Institute of Technology*, 2011. 6, 7, 8

[44] C. Williams and M. Seeger. Using the nyström method to speed up kernel machines. In *NIPS*, 2001. 1

[45] J. Yang, K. Yu, Y. Gong, and T. Huang. Linear spatial pyramid matching using sparse coding for image classification. In *CVPR*, 2009. 2