Non-linear Convolution Filters for CNN-based Learning

Georgios Zoumpourlis Alexandros Doumanoglou Nicholas Vretos Petros Daras
Information Technologies Institute, Center for Research and Technology Hellas, Greece
6th km Charilaou-Thermi Road, Thessaloniki, Greece
{zoump.giorgos, aldoum, vretos, daras}@iti.gr

Abstract

During the last years, Convolutional Neural Networks (CNNs) have achieved state-of-the-art performance in image classification. Their architectures have largely drawn inspiration by models of the primate visual system. However, while recent research results of neuroscience prove the existence of non-linear operations in the response of complex visual cells, little effort has been devoted to extend the convolution technique to non-linear forms. Typical convolutional layers are linear systems, hence their expressiveness is limited. To overcome this, various non-linearities have been used as activation functions inside CNNs, while also many pooling strategies have been applied. We address the issue of developing a convolution method in the context of a computational model of the visual cortex, exploring quadratic forms through the Volterra kernels. Such forms, constituting a more rich function space, are used as approximations of the response profile of visual cells. Our proposed second-order convolution is tested on CIFAR-10 and CIFAR-100. We show that a network which combines linear and non-linear filters in its convolutional layers, can outperform networks that use standard linear filters with the same architecture, yielding results competitive with the state-of-the-art on these datasets.

1. Introduction

Convolutional neural networks (CNNs) have been shown to achieve state-of-the-art results on various computer vision tasks, such as image classification. Their architectures have largely drawn inspiration by models of the primate visual system, as the one described by Hubel and Wiesel [13]. The notion of convolution, used to mimic a functional aspect of neurons in the visual cortex, is critical to understand their success.

Typical convolutional layers are linear systems, as their outputs are affine transformations of their inputs. Due to their linear nature, they lack the ability of expressing possible non-linearities that may actually appear in the response of complex cells in the primary visual cortex [25]. Hence, we claim that their expressiveness is limited. To overcome this, various non-linearities have been used as activation functions inside CNNs, while also many pooling strategies have been applied. Little effort has been devoted to explore new computational models that extend the convolution technique to non-linear forms, taking advantage of the research results of neuroscience, that prove the existence of non-linear operations in the response of visual cells [31][24]. The complexity of human visual cortex demonstrates gaps that need to be bridged by CNNs, regarding the way convolution operations are applied. One of these gaps, is the exploration of higher-order models.

In this work, we study the possibility of adopting an alternative convolution scheme to increase the learning capacity of CNNs by applying Volterra's theory [32], which has been used to study non-linear physiological systems, adapting it to the spatial domain. Considering the convolution operation, instead of summing only linear terms to compute a filter's response on a data patch, we propose to also sum the non-linear terms produced by multiplicative interactions between all the pairs of elements of the input data patch. Transforming the inputs through a second-order form, we aim at making them more separable. In this way, convolution filters with more rich properties in terms of selectivity and invariance are created.

The novelties of the proposed work are:

- The incorporation of a biologically plausible non-linear convolution scheme in the functionality of CNNs
- The derivation of the equations that describe the forward and backward pass during the training process of this filter type
- A CUDA-based implementation of our method as a

non-linear convolutional layer's module in Torch7[1][5]

The rest of the paper is organized as follows: in Section 2, related work is outlined. In Section 3, the proposed method is described, theoretically grounded to Volterra's computational method, and the concept of training is mathematically explained, while a description of our scheme's practical implementation is given in Section 4. In Section 5 experimental results on CIFAR-10 and CIFAR-100 datasets are drawn and finally in Section 6 the paper is concluded.

2. Related Work

One of the first biologically-inspired neural networks, was Fukushima's Neocognitron [8], which was the predecessor of CNN, as it was introduced by LeCun *et al.* in [6]. Convolutional layer is the core building block of a CNN. Early implementations of CNNs have used predefined Gabor filters in their convolutional layers. This category of filters can model quite accurately the properties of simple cells found in the primary visual cortex (V1) [21].

This type of visual cell has a response profile which is characterized by spatial summation within the receptive field. Finding the optimal spatial stimuli [7] for simple cells is a process based on the spatial arrangement of their excitatory and inhibitory regions [23]. However, this does not hold true for complex visual cells. Also, we cannot obtain an accurate description of their properties, by finding their optimal stimulus.

This fact has been ignored by most of the CNN implementations so far, as they have settled to the linear type of convolution filters, trying to apply quantitative rather than qualitative changes in their functionalities. He *et al.* [10] proposed Residual Networks (ResNets), which have shortcut connections parallel to their normal convolutional layers, as a solution to the problems of vanishing/exploding gradient and hard optimization when increasing the model's parameters (i.e. adding more layers). Zagoruyko & Komodakis [34] showed that wide ResNets can outperform ResNets with hundrends of layers, shifting the interest to increasing the number of each layer's filters. Alternatively to works that focus on creating networks with more convolutional layers or more filters, we evaluate the impact of using both non-linear and linear terms as approximations of convolution kernels to boost the performance of CNNs.

Apart from ResNets, very low error rates have also been achieved in the ImageNet Challenge [27] by methods that used their convolutional layers in new ways, enhancing their representation ability. Lin *et al.* [20] proposed "Network in Network (NIN)", as a remedy to the low level of abstraction that typical filters present. Instead of the conventional convolution filter, which is a generalized linear model, they build micro neural networks with more complex structures to abstract the data within the receptive field. To map the input data to the output, they use multilayer perceptrons as a non-linear function approximator, which they call "mlpconv" layer. The output feature maps are obtained by sliding the micro networks over the input in a similar manner as CNN. Szegedy *et al.* [30] introduced a new level of organization in the form of the "Inception module", which uses filters of variable sizes to capture different visual patterns of different sizes, and approximates the optimal sparse structure. Xie *et al.* [33] proposed a way to exploit the split-transform-merge strategy of "Inception" models, performing a set of transformations, each on a low-dimensional embedding, whose outputs are aggregated by summation.

The authors of [19], based on the abundancy of recurrent synapses in the brain, proposed the use of a recurrent neural network for image classification. They proved that inserting recurrent connections within convolutional layers, gives boosted results, compared to a feed-forward architecture. Their work is a biologically plausible incorporation of mechanisms originating from neuroscience into CNNs.

In [28], a Boltzmann learning algorithm is proposed, where feature interactions are used to turn hidden units into higher-order feature detectors. In [22], an efficient method to apply such learning algorithms on higher-order Boltzmann Machines was proposed, making them computationally tractable for real problems.

In [1], Bergstra *et al.* created a model for neural activation which showed improved generalization on datasets, by incorporating second-order interactions and using an alternative non-linearity as activation function.

In [2], an attempt is made to analyze and interpret quadratic forms as receptive fields. In their study, it was found that quadratic forms can be used to model non-linear receptive fields due to the fact that they follow some of the properties of complex cells in the primary visual cortex. These properties include response to edges, phase-shift invariance, direction selectivity, non-orthogonal inhibition, end-inhibition and side-inhibition. In constrast to the standard linear forms, in quadratic and other non-linear forms the optimal stimuli do not provide a complete description of their properties. It is shown that no invariances occure for an optimal stimulus while for other general sub-optimal stimuli there may exist many invariances which could be of a large number but lack easy interpretation. Although the optimal stimulus is not related to a filter's invariance, its neighborhood is studied under a more loose sense of transformation invariance. It is shown that proper quadratic forms can demonstrate invariances in phase-shift and orientation change. From the previous discussion we conclude that using non-linear forms to convolutional layers may be a reasonable future direction in computer vision.

[1] http://torch.ch/

3. Proposed Method

The proposed method, as earlier stated, makes use of the Volterra kernel theory to provide means of exploiting the non-linear operations that take place in a receptive field. Up to now, and to the best of our knowledge, non-linearities were exploited mainly through the activation functions and pooling operations between different layers of CNNs. Nevertheless, such non-linearities may be an approach to code inner processes of the visual system, but not the ones that exist in a receptive field's area.

Our method follows the typical workflow of a CNN, by lining up layers of different purposes (convolution, pooling, activation function, batch normalization, dropout, fully-connected etc.), while a non-linear convolutional layer can be plugged in practically in all existing architectures. Nevertheless, due to its augmentation of trainable parameters involved, care should be taken for the complexity of the overall process. To that end, a CUDA implementation in Section 4 is also provided.

3.1. Volterra-based convolution

The Volterra series model is a sequence of approximations for continuous functions, developed to represent the input-output relationship of non-linear dynamical systems, using a polynomial functional expansion. Their equations can be composed by terms of infinite orders, but practical implementations based on them use truncated versions, retaining the terms up to some order r.

In a similar way to linear convolution, Volterra-based convolution uses kernels to filter the input data. The first-order Volterra kernel, contains the coefficients of the filter's linear part. The second-order kernel represents the coefficients of quadratic interactions between two input elements. In general, the r-th order's kernel represents the weights that non-linear interactions between r input elements have on the response. In the field of computer vision, Volterra kernels have been previously used in [17] for face recognition, serving effectively as approximations of non-linear functionals.

3.2. Forward pass

For our proposed convolution, we adopted a second-order Volterra series. Given an input patch $\mathbf{I} \in \mathbb{R}^{k_h \times k_w}$ with n elements ($n = k_h \cdot k_w$), reshaped as a vector $\mathbf{x} \in \mathbb{R}^n$:

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}^T \tag{1}$$

the input-output function of a linear filter is:

$$y(\mathbf{x}) = \sum_{i=1}^{n} \left(w_1^i x_i \right) + b \tag{2}$$

where w_1^i are the weights of the convolution's linear terms, contained in a vector $\mathbf{w}_1$, and b is the bias. In our approach, this function is expanded in the following quadratic form:

$$y(\mathbf{x}) = \sum_{i=1}^{n} \left(w_1^i x_i \right) + \sum_{i=1}^{n} \sum_{j=i}^{n} \left(w_2^{i,j} x_i x_j \right) + b \tag{3}$$

where $w_2^{i,j}$ are the weights of the filter's second-order terms. To avoid considering twice the interaction terms for each pair of input elements (x_i, x_j), we adopt an upper-triangular form for the matrix $\mathbf{w}_2$ containing their weights, so that the number of trainable parameters for a second-order kernel is $n(n+1)/2$. The generic type to compute the total number of parameters, n_V, for a Volterra-based filter of order r is:

$$n_V = \frac{(n+r)!}{n!r!} \tag{4}$$

In a more compact form, (3) is written as:

$$y(\mathbf{x}) = \underbrace{\mathbf{x}^T \mathbf{w}_2 \mathbf{x}}_{\text{quadratic term}} + \underbrace{{\mathbf{w}_1}^T \mathbf{x}}_{\text{linear term}} + b \tag{5}$$

while for the Volterra kernels we have:

$$\mathbf{w}_2 = \begin{bmatrix} w_2^{1,1} & w_2^{1,2} & \cdots & w_2^{1,n} \\ 0 & w_2^{2,2} & \cdots & w_2^{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & w_2^{n,n} \end{bmatrix} \tag{6}$$

containing the coefficients $w_2^{i,j}$ of the quadratic term, and:

$${\mathbf{w}_1}^T = \begin{bmatrix} w_1^1 & w_1^2 & \cdots & w_1^n \end{bmatrix} \tag{7}$$

containing the coefficients w_1^i of the linear term. The proposed convolution's output can thus be rewritten as:

$$y(\mathbf{x}) = \begin{bmatrix} w_2^{1,1} \\ w_2^{1,2} \\ w_2^{1,3} \\ \vdots \\ w_2^{n,n} \end{bmatrix}^T \begin{bmatrix} x_1 x_1 \\ x_1 x_2 \\ x_1 x_3 \\ \vdots \\ x_n x_n \end{bmatrix} + \begin{bmatrix} w_1^1 \\ w_1^2 \\ w_1^3 \\ \vdots \\ w_1^n \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} + b \tag{8}$$

Note that superscripts (i, j) to weights $w_2^{i,j}$ denote correspondence to the spatial positions of the input elements x_i and x_j that interact.

3.3. Backward pass

The derivation of the equations for the backward pass of the Volterra-based convolution, is done by adapting the classic backpropagation scheme to the aforementioned input-output function of (3). To train the weights of the Volterra kernels, we need to compute the gradients of the layer's output $y(\mathbf{x})$, with respect to the weights w_1^i and $w_2^{i,j}$. To propagate the error, we have to compute the gradients of the layer's output $y(\mathbf{x})$, with respect to the inputs x_i. Hence, $\frac{\partial y}{\partial w_1^i}$, $\frac{\partial y}{\partial w_2^{i,j}}$ and $\frac{\partial y}{\partial x_i}$ are the terms that will be used to optimize the weight parameters of our Volterra-based convolutional layer and minimize the network loss. The mathematical equations of backpropagation, are as follows:

$$\frac{\partial y}{\partial w_1^i} = x_i \qquad \frac{\partial y}{\partial w_2^{i,j}} = x_i x_j \tag{9}$$

$$\frac{\partial y}{\partial x_i} = w_1^i + \sum_{k=1}^{i} \left(w_2^{k,i} x_k\right) + \sum_{k=i}^{n} \left(w_2^{i,k} x_k\right) \tag{10}$$

4. Quadratic convolution filter implementation

In order to experiment with the non-linear convolution filters, we used the Torch7 scientific framework. Volterra-based convolution was implemented as a module integrated with the CUDA backend for the Neural Network (cunn) Package of Torch7. Writing a module in Torch7 mainly consists of implementing the module's forward pass (3) as well as the computation of the module's gradients ($\frac{\partial E}{\partial \mathbf{w}}$ and $\frac{\partial E}{\partial \mathbf{x}}$), that are used in back-propagation. We denote by E the error defined by the network's criterion function and refer to $\frac{\partial E}{\partial \mathbf{w}}$ as the layer's weight gradient and $\frac{\partial E}{\partial \mathbf{x}}$ as the layer's input gradient. To implement the forward pass in CUDA, we used the standard im2col [3] pattern to unfold data patches into columns, followed by a matrix multiplication with the Volterra-based filter weights. The im2col operation is conducted in parallel by a CUDA kernel, while for the matrix multiplication we used the well established CUDA BLAS functions. Subsequently, computing the weight gradients is, to some extent, similar to computing the forward pass. Once again, the im2col operation is executed on the input image as a CUDA kernel and its output matrix is multiplied with the previous layer's input gradient resulting into $\frac{\partial E}{\partial \mathbf{w}}$. The most expensive operation in a Volterra-based convolutional layer is the computation of the input gradients. As already mentioned before, in contrast to linear convolution, where the input gradient is independent of the provided input, our layer's input gradient is input-dependent. Thus, to compute the matrix of input gradients, firstly we compute an unfolded matrix containing the gradients of the output with respect to the input. This matrix is then multiplied with the previous layer's input gradient using CUDA BLAS functions. Finally, an appropriate inverse col2im CUDA kernel aggregate operation results in the final matrix of the Volterra-based layer's input gradients $\frac{\partial E}{\partial \mathbf{x}}$.

A major difference between the proposed convolution scheme and linear convolution, is the fact that in our case $\frac{\partial y}{\partial x_i}$ is a function dependent on x_i. This means that, in contrast to standard filters, this term is different for every single patch of a feature map, resulting in an extra computational cost, when the error must be propagated to preceding trainable layers in the network. This cost is proportionate to $H_o \cdot W_o$, where H_o and W_o are the height and the width of the layer's output feature map, respectively. Our layer's code is available at `http://vcl.iti.gr/volterra`.

5. Experiments

We measure the performance of our proposed Volterra-based convolution on two benchmark datasets: CIFAR-10 and CIFAR-100 [15], running our experiments on a PC equipped with Intel i7-5820K CPU, 64GB RAM and Nvidia Titan X GPU. The Volterra-based convolutional layer was implemented in Torch7. We first describe the experimental setup, then we show a quantitative analysis, in terms of parameters, classification error and train loss, for the proposed method.

5.1. CNN architecture selection

As explained in Section 4, using the proposed convolution in multiple layers of a CNN, an extra computational overhead is introduced during backpropagation. For this reason, we restrain ourselves to testing such filters only in the first convolutional layer of a CNN model. We choose the modern architecture of Wide ResNet [34], which mainly consists of a convolutional layer, followed by 3 convolutional groups and a classifier. If d is such a network's depth, then each convolutional group contains $N = (d-4)/6$ convolutional blocks. In a group, the number of each convolutional layer's filters, is controlled by the widening factor k. In our architecture, we follow the above rules, making three changes: a) we insert a Batch Normalization layer in the start of the network b) we change the number of the first convolutional layer's output channels, from 16 to $16 \cdot k$ (i.e., equal to the number of the first group's output channels) and c) we change the shortcut of the first block in the first group, into an identity mapping, as a consequence of our second change. The first change is crucial to prevent the output of the Volterra-based convolution from exploding, due to the multiplicative interaction terms $x_i x_j$. In our experiments, parameter γ of the affine transformation $y = \gamma\hat{x} + \beta$ that is applied in this layer, settles to values $0 < \gamma < 1$. The second change was chosen so that, when the Volterra-based convolution is applied in the first convolutional layer, there are enough non-linear filters to be learnt, producing a feature-rich signal. The third change is

Network stage	**Output size**	**Model** (d=28, N=4, k=10)
Initial convolution	32×32	Batch Normalization Conv 3×3, $16 \cdot k$
Group 1	32×32	Conv $\begin{bmatrix} 3 \times 3, 16 \cdot k \\ 3 \times 3, 16 \cdot k \end{bmatrix} \times N$ blocks
Group 2	16×16	Conv $\begin{bmatrix} 3 \times 3, 32 \cdot k \\ 3 \times 3, 32 \cdot k \end{bmatrix} \times N$ blocks
Group 3	8×8	Conv $\begin{bmatrix} 3 \times 3, 64 \cdot k \\ 3 \times 3, 64 \cdot k \end{bmatrix} \times N$ blocks
Pooling	8×8 8×8 1×1	Batch Normalization ReLU Average Pooling, 8×8
Classifier		Fully-Connected $10(100)$ SoftMax $10(100)$

Table 1: Network architecture used in our experiments.

done because when a block's input and output channels are equal, then its shortcut is an identity mapping, so that its input is added to its output, without the need to adjust the feature channels in the shortcut by using a convolutional layer. In this case, the signal of the first convolutional layer flows intact through the shortcuts of the first group's blocks.

The model used in our experiments is described in Table 1. To evaluate the impact of applying the Volterra-based convolution on each dataset, we tested two versions of the general CNN model. The first version, which serves as the baseline model, does not use any non-linear convolution filter. The other version contains non-linear filters in the first convolutional layer and linear filters in all the convolutional groups of the network.

5.2. Experimental setup

In all our experiments we use Stochastic Gradient Descent (SGD) with momentum set to 0.9 and cross-entropy loss, with a batch size of 128, training our network for 220 epochs. Dropout is set to 0.3 and weight initialization is done as in [10]. The learning rate and weight decay strategy used in the experiments is shown in Table 2. For CIFAR-10 and CIFAR-100, the data-preprocessing operation applied

Epoch	**Learning rate**	**Weight decay**
$1 - 59$	0.1	0.0005
$60 - 119$	0.02	0.0005
$120 - 159$	0.004	0.0005
$160 - 199$	0.0008	0.0005
$200 - 220$	0.0008	0

Table 2: Learning rate and weight decay strategy used in our experiments.

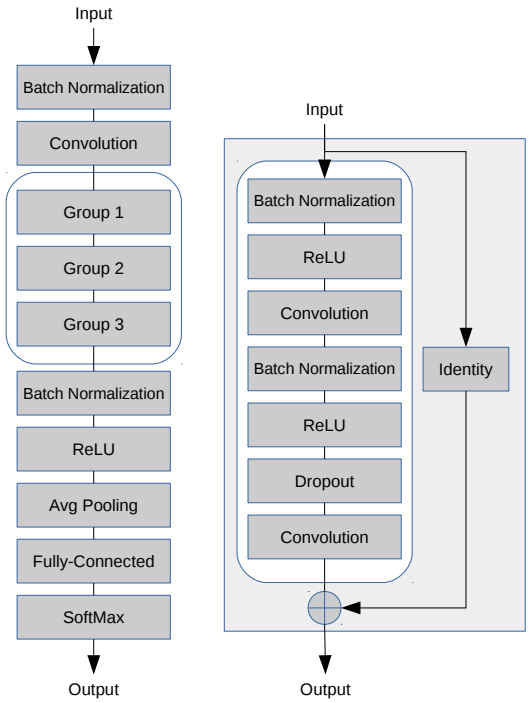

Figure 1: Structure of the proposed CNN model (left) and a typical convolutional block (right).

Network	Depth	#Parameters	CIFAR-10	CIFAR-100
NIN [20]	-	-	8.81	-
DSN [18]	3	-	7.97	34.57
All-CNN [29]	9	1.3M	7.25	-
ResNet with Stochastic Depth [12]	110	1.7M	5.23	24.58
	1202	10.2M	4.91	-
pre-act Resnet [11]	1001	10.2M	4.62	22.71
Wide ResNet [34]	40	55.8M	3.80	18.30
PyramidNet [9]	110	28.3M	3.77	18.29
Wide-DelugeNet [16]	146	20.2M	3.76	19.02
OrthoReg on Wide ResNet [26]	28	-	3.69	18.56
Steerable CNNs [4]	14	9.1M	3.65	18.82
ResNeXt [33]	29	68.1M	3.58	**17.31**
Wide ResNet with Singular Value Bounding [14]	28	36.5M	3.52	18.32
Oriented Response Net [35]	28	18.4M	**3.52**	19.22
Baseline Wide ResNet	28	36.6M	3.62	18.29
Volterra-based Wide ResNet	28	36.7M	**3.51**	**18.24**

Table 3: Test set classification error results on CIFAR-10 and CIFAR-100, using moderate data augmentation (horizontal flipping, padding and 32×32 cropping).

to both train and test set's data, is subtracting the channel means and then dividing by the channel standard deviations, computed on the train set. We apply moderate data augmentation, using horizontal flipping with a probability of 50% and reflection-padding by 4 pixels on each image side, taking a random crop of size 32×32.

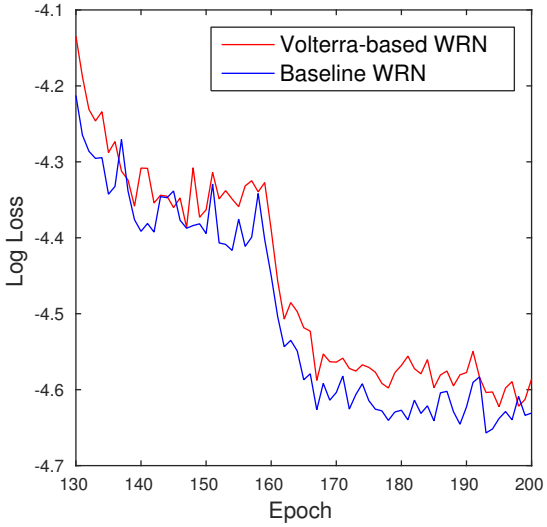

Figure 2: Train loss on CIFAR-100.

5.3. CIFAR-10 and CIFAR-100

CIFAR-10 and CIFAR-100 datasets contain 60.000 32×32 RGB images of commonly seen object categories (e.g., animals, vehicles, etc.), where the train set has 50.000 and the test set has 10.000 images. CIFAR-10 has 10 classes and CIFAR-100 has 100 classes. All classes have equal number of train and test samples. In CIFAR-10, our Volterra-based Wide ResNet yields a test error of 3.51%, which shows an improvement over the 3.62% error that we got using the baseline model, setting the state-of-the-art on this dataset. In CIFAR-100, our Volterra-based Wide ResNet yields a test error of 18.24%, which shows an improvement over the 18.29% error that we got using the baseline model. Our results on CIFAR-100 are outperformed only by [33], due to the huge number of parameters their model makes use of. The features fed to the convolutional groups, when extracted by the non-linear convolution filters, make the network avoid overfitting. This can be inferred by the loss plot of our models on CIFAR-100, which is shown in Figure 2. The Baseline Wide ResNet, although having constantly lower loss than the Volterra-based Wide ResNet, yields higher test error. Our Volterra-based Wide ResNets have only 0.05% more parameters than the Baseline counterparts. A summary of the best methods on these datasets is provided in Table 3.

5.4. Weight visualization

To get an insight on what features do non-linear filters learn, we visualize their weights in a simple but efficient

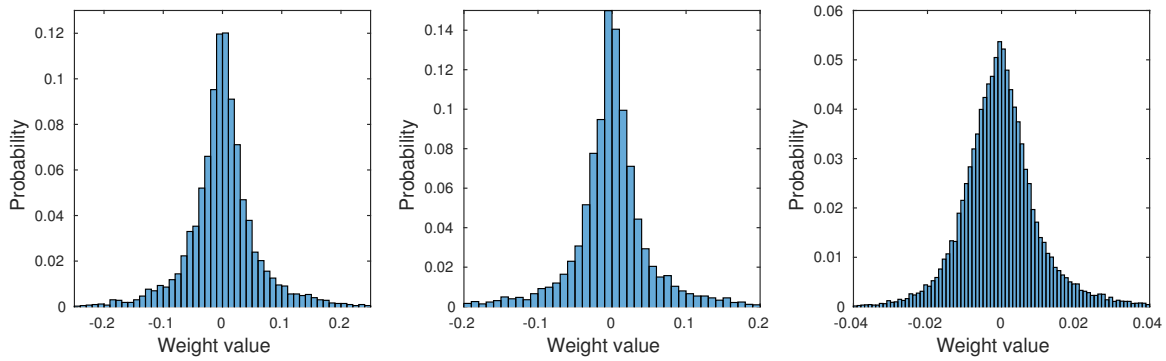

Figure 3: Weight values of linear convolution filter weights (left), Volterra-based convolution first-order weights (middle) and Volterra-based convolution second-order weights (right).

manner. For the linear term, the process is straightforward. For the second-order term, considering the weights $\mathbf{w}_2$ of each filter, we can create n weight vectors $\mathbf{q}_i$, $\mathbf{q}_i = [w_2^{i1}, w_2^{i2}, ..., w_2^{in}]$. Reshaping each one of these vectors $\mathbf{q}_i$ into a $k_h \times k_w$ matrix, we can see the weights that correspond to the interactions between x_i and all of the receptive field's elements. Figure 4 shows the weights of the linear term and the interactions captured by a second-order 3×3 filter, allowing us to explore their contribution to the response. Another issue, is the values that the weights of the non-linear terms settle to. We investigate these values, given the filters of the first convolutional layer of our Wide ResNet model, trained on CIFAR-100. The histograms of the weight values are shown in Figure 3. The value distribution of the linear convolution filters' weights is similar to that of the quadratic filters' first-order weights. Also, the values of the quadratic filters' second-order weights have reasonably smaller standard deviation.

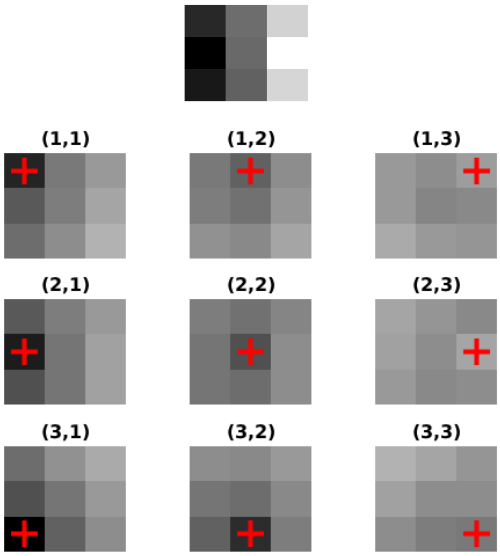

Figure 4: Linear term and second-order multiplicative interaction weights of a Volterra-based 3×3 filter.

5.5. Response profiles

Following the methodology of [2], we use a set of Volterra-based filters of a Wide ResNet trained on CIFAR-100, to partly characterize their response profiles. Given the weights $\mathbf{w}_1$, $\mathbf{w}_2$ of a filter, we compute its optimal stimulus, $\mathbf{x}_o$, and the optimal stimulus of its linear term, $\mathbf{x}_l$, under the constraint that their norms are equal. Then, we compute four responses, as described in Table 4, and plot them in Figure 5. Comparing the various responses, we can infer that the properties of a linear filter with weights $\mathbf{w}_1$, can greatly change when it's extended to a second-order Volterra form by adding a weight set $\mathbf{w}_2$ with quadratic contributions. The response of a Volterra-based filter is quite different from the response of its first-order terms, proving that the second-order interactions contribute significantly to the functionality of a quadratic filter.

Given the weight subset $\mathbf{w}_1$ of a Volterra-based filter, their optimal stimulus $\mathbf{x}_l$ has a standard pattern. As the norm of $\mathbf{x}_l$ takes values inside a bounded space, the way $\mathbf{x}_l$ varies is just a linear increase in all its intensity values, without altering its general pattern (i.e., all vectors $\mathbf{x}_l$ are parallel). However, this does not hold true for quadratic filters. As the norm of a Volterra-based second-order filter's optimal stimulus $\mathbf{x}_o$, takes values inside a bounded space, a rich variety of alterations can be observed in the elements of $\mathbf{x}_o$.

6. Conclusion

The exploration of CNN architectures that are optimized for using non-linear convolution filters, is an open problem

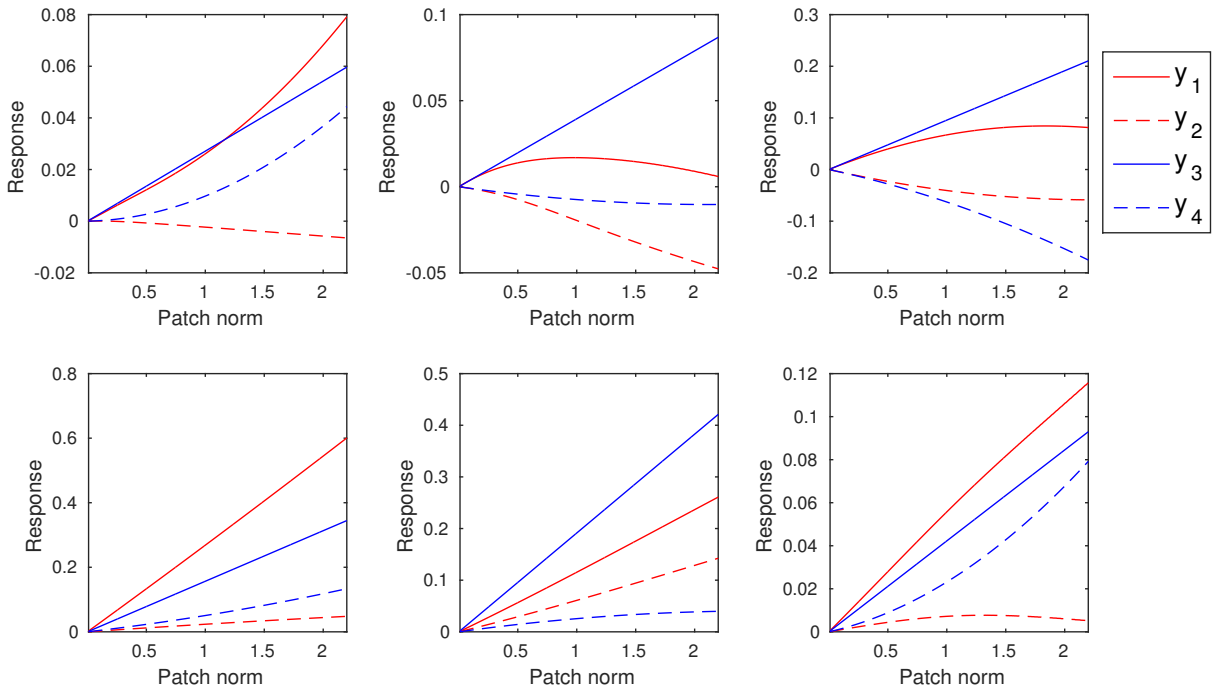

Figure 5: Various cases of responses. Red line denotes the response y_1 of a Volterra-based convolution filter to its optimal stimulus $\mathbf{x}_o$. Dashed red line denotes the response y_2 of the linear subset of a Volterra-based filter's weights, to $\mathbf{x}_o$. Blue line denotes the response y_3 of the linear subset of a Volterra-based filter's weights, to $\mathbf{x}_l$. Dashed blue line denotes the response y_4 of a Volterra-based convolution filter, to $\mathbf{x}_l$.

Stimulus	**Filter**	**Response**
$\mathbf{x}_o$	Quadratic ($\mathbf{w}_1$, $\mathbf{w}_2$)	y_1
$\mathbf{x}_o$	Linear ($\mathbf{w}_1$)	y_2
$\mathbf{x}_l$	Linear ($\mathbf{w}_1$)	y_3
$\mathbf{x}_l$	Quadratic ($\mathbf{w}_1$, $\mathbf{w}_2$)	y_4

Table 4: Stimuli, filter weight sets and filter responses.

for biologically-inspired computer vision. Questions like "which is the ideal ratio between linear and non-linear filters in each convolutional layer?" and "which properties prevail in the response profiles of each layer's non-linear filters?" are of great importance, to shed light in this hitherto unexplored category of filters. Any inference about the properties that are present to this group of quadratic filters, has the risk of being biased by the dataset used to obtain and observe them. This happens because the visual response profiles of the non-linear filters trained in the experiments, are constrained by the natural statistics of each dataset, as happens with the sensory system of primates, which adapts to its environment.

Based on the research results of neuroscience that prove the existence of non-linearities in the response profiles of complex visual cells, we have proposed a non-linear convolution scheme that can be used in CNN architectures. Our experiments showed that a network which combines linear and non-linear filters in its convolutional layers, can outperform networks that use standard linear filters with the same architecture. Our reported error rates set the state-of-the-art on CIFAR-10, while being competitive to state-of-the-art results on CIFAR-100. We didn't apply our Volterra-based convolution to more layers, because our target was to demonstrate a proof of concept for the proposed method. Our claim was confirmed, as replacing only the first convolutional layer's linear filters with non-linear ones, we achieved lower error rates. Further testing quadratic convolution filters, is certainly an interesting direction for future work, to build better computer vision systems.

Acknowledgment

The research leading to these results has been supported by the EU funded project FORENSOR (GA 653355).

References

[1] J. Bergstra, G. Desjardins, P. Lamblin, and Y. Bengio. Quadratic polynomials learn better image features. Technical report, Technical Report 1337, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, 2009.

[2] P. Berkes and L. Wiskott. On the analysis and interpretation of inhomogeneous quadratic forms as receptive fields. *Neural computation*, 18(8):1868–1895, 2006.

[3] K. Chellapilla, S. Puri, and P. Simard. High Performance Convolutional Neural Networks for Document Processing. In *Tenth International Workshop on Frontiers in Handwriting Recognition*, La Baule (France), Oct. 2006. Université de Rennes 1, Suvisoft. http://www.suvisoft.com.

[4] T. S. Cohen and M. Welling. Steerable cnns. *CoRR*, abs/1612.08498, 2016.

[5] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A Matlab-like Environment for Machine Learning. In *BigLearn, NIPS Workshop*, 2011.

[6] Y. L. Cun, B. Boser, J. S. Denker, R. E. Howard, W. Habbard, L. D. Jackel, and D. Henderson. Advances in neural information processing systems 2. chapter Handwritten Digit Recognition with a Back-propagation Network, pages 396–404. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[7] P. Földiák. Stimulus optimisation in primary visual cortex. *Neurocomputing*, 3840:1217 – 1222, 2001. Computational Neuroscience: Trends in Research 2001.

[8] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. 36(4):193–202, 1980.

[9] D. Han, J. Kim, and J. Kim. Deep pyramidal residual networks. *CoRR*, abs/1610.02915, 2016.

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[11] K. He, X. Zhang, S. Ren, and J. Sun. *Identity Mappings in Deep Residual Networks*, pages 630–645. Springer International Publishing, Cham, 2016.

[12] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. Deep networks with stochastic depth. *CoRR*, abs/1603.09382, 2016.

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

[14] K. Jia. Improving training of deep neural networks via singular value bounding. *CoRR*, abs/1611.06013, 2016.

[15] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[16] J. Kuen, X. Kong, and G. Wang. Delugenets: Deep networks with massive and flexible cross-layer information inflows. *CoRR*, abs/1611.05552, 2016.

[17] R. Kumar, A. Banerjee, and B. C. Vemuri. Volterrafaces: Discriminant analysis using volterra kernels. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 150–155. IEEE, 2009.

[18] C. Lee, S. Xie, P. W. Gallagher, Z. Zhang, and Z. Tu. Deeply-supervised nets. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2015, San Diego, California, USA, May 9-12, 2015*, 2015.

[19] M. Liang and X. Hu. Recurrent convolutional neural network for object recognition. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3367–3375, June 2015.

[20] M. Lin, Q. Chen, and S. Yan. Network in network. In *International Conference on Learning Representations*, 2014.

[21] S. Marĉelja. Mathematical description of the responses of simple cortical cells. *Journal of the Optical Society of America*, 70(11):1297–1300, 1980.

[22] M. R. Min, X. Ning, C. Cheng, and M. Gerstein. Interpretable sparse high-order boltzmann machines. In *AISTATS*, 2014.

[23] J. A. Movshon, I. D. Thompson, and D. J. Tolhurst. Spatial summation in the receptive fields of simple cells in the cat's striate cortex. *The Journal of physiology*, 283:53, 1978.

[24] C. M. Niell and M. P. Stryker. Highly Selective Receptive Fields in Mouse Visual Cortex. *Journal of Neuroscience*, 28(30):7520–7536, July 2008.

[25] J. Rapela, J. M. Mendel, and N. M. Grzywacz. Estimating nonlinear receptive fields from natural images. *Journal of Vision*, 6(4):11, 2006.

[26] P. Rodríguez, J. Gonzàlez, G. Cucurull, J. M. Gonfaus, and F. X. Roca. Regularizing cnns with locally constrained decorrelations. *CoRR*, abs/1611.01967, 2016.

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

[28] T. J. Sejnowski, P. K. Kienker, and G. E. Hinton. Learning symmetry groups with hidden units: Beyond the perceptron. *Physica D: Nonlinear Phenomena*, 22(1-3):260–275, 1986.

[29] J. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. In *ICLR (workshop track)*, 2015.

[30] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[31] R. G. Szulborski and L. A. Palmer. The two-dimensional spatial structure of nonlinear subunits in the receptive fields of complex cells. *Vision Research*, 30(2):249–254, 1990.

[32] V. Volterra. *Theory of Functionals and of Integral and Integro-Differential Equations*. Dover Publications, 2005.

[33] S. Xie, R. B. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. *CoRR*, abs/1611.05431, 2016.

[34] S. Zagoruyko and N. Komodakis. Wide residual networks. In *BMVC*, 2016.

[35] Y. Zhou, Q. Ye, Q. Qiu, and J. Jiao. Oriented response networks. *CoRR*, abs/1701.01833, 2017.