Submanifold Sparse Convolutional Networks

Benjamin Graham, Laurens van der Maaten
Facebook AI Research
{benjamingraham,lvdmaaten}@fb.com

June 6, 2017

Abstract

Convolutional network are the de-facto 事实上的 standard for analysing spatio-temporal space-time：时空 data such as images, videos, 3D shapes, etc. Whilst 同时 some of this data is naturally dense (for instance, photos), many other data sources are inherently 天然的|本质的 sparse. Examples include pen-strokes 笔划|笔锋|摸|拧 forming on a piece of paper, or (colored) 3D point clouds that were obtained using a LiDAR scanner or RGB-D camera. Standard "dense" implementations of convolutional networks are very inefficient when applied on such sparse data. We introduce a sparse convolutional operation tailored to 量身定做|加工 processing sparse data that differs from prior work on sparse convolutional networks in that it operates strictly on submanifolds, rather than "dilating" the observation with every layer in the network. Our empirical analysis of the resulting submanifold sparse convolutional networks shows that they perform on par with state-of-the-art methods whilst requiring substantially less computation.

1 Introduction

Convolutional networks constitute 构成|组成|占 the state-of-the art method for a wide range of tasks that involve the analysis of data with spatial and/or temporal 时间 structure, such as photographs, videos, or three-dimensional surface models. While such data frequently comprises 包含 a densely filled (2D or 3D) grid, other spatio-temporal datasets are naturally sparse. For instance, handwriting is made up of one-dimensional lines in two-dimensional space, pictures made by RGB-D cameras are three-dimensional point clouds, and OFF models form two-dimensional surfaces in 3D space. The curse of dimensionality applies, in particular, on data that lives on grids that have three or more dimensions: the number of points on the grid grows exponentially with its dimensionality. In such scenarios, it becomes increasingly important to exploit 利用 data sparsity 稀疏性 whenever possible 尽可能的 in order to reduce the computational resources needed for data processing. Indeed, exploiting sparsity is paramount when analyzing, for instance, RGB-D videos which are sparsely populated 4D structures.

Traditional convolutional network implementations are optimized for data that lives on densely populated grids, and cannot process sparse data efficiently. More recently, a number of convolutional network implementations have been presented that are tailored to work efficiently on sparse data [1, 2, 9]. Mathematically, some of these implementations are identical to a regular convolutional network, but they require fewer computational resources in terms of FLOPs [1] and/or in terms of memory [2]. OctNets [9] slightly modify the convolution operator to produce "averaged" hidden states in parts of the grid that are away from regions of interest.

One of the downsides of prior sparse implementations of convolutional networks is that they "dilate" the sparse data in every layer, because they implement a "full" convolution. In this work, we show that it is possible to successfully train convolutional networks that keep the same sparsity pattern throughout the layers of the network, without dilating the feature maps. To this end, we explore two novel convolution operators: *sparse convolution* (SC) and *valid sparse convolution* (VSC). In our experiments with recognizing handwritten digits and 3D shapes, networks using SC and VSC achieve state-of-the-art performance whilst reducing the computation and memory requirements by ∼50%.

Recommend to read this blog:
https://medium.com/geekculture/3d-sparse-sabmanifold-convolutions-eaa427b3a196

Submanifolds refer to sparse input data, such as one-dimensional curves in two-dimensional space, two-dimensional curved surfaces, and point clouds in three-dimensional space. They do not occupy the entire space in which they are located.

2 Motivation

We define a d-dimensional convolutional network as a network that takes as input that is a $(d+1)$-dimensional tensor: the input tensor contains d spatiotemporal dimensions (such as length, width, height, time, *etc.*) and one additional feature space dimension (for instance, RGB color channels, surface normal vectors, *etc.*). A sparse input corresponds to a d-dimensional grid of *sites* that is associated with a feature vector. 基态 We define a site in the input to be *active* if any element in the feature vector is not in its *ground state*, for instance, if it is non-zero[1]. In many practical problems, thresholding may be used to eliminate sites at which the feature vector is within a very small distance from the ground state. Note that even though the input tensor is $(d+1)$-dimensional, activity is a d-dimensional phenomenon: entire planes along the feature dimension are either active or not.

说人话就是定义当site不在某个值（某种状态）就叫做激活了。

The hidden layers of a convolutional network are also represented by d-dimensional grids of feature-space vectors. When propagating the input data through the network, a site in a hidden layer is active if any of the sites in the layer that it takes as input is active. (Note that when using 3×3 convolutions, each site is connected to $3 \times 3 = 9$ sites in the hidden layer below.) Activity in a hidden layer thus follows an 归纳 inductive definition in which each layer determines the set of active states in the next. In each hidden layer, inactive sites all have the same feature vector: the one corresponding to the ground state. Note that the ground state in a hidden layer is often not equal to zero, in particular, when convolutions with a bias term are used. However, 不管 irrespective of the value of the ground state, the ground-state value only needs to be calculated once per forward pass during training (and only once for all forward passes at test time). This allows for substantial savings in computational and memory requirements; the exact savings depend on the data sparsity and the network depth. 这段强烈建议去看blog

In this paper, we argue that the framework described above is unduly restrictive, in particular, because the convolution operation has not been modified to accommodate the sparsity of the input data. If the input data contains a single active site, then after applying a 3^d convolution, there will be 3^d active sites. Applying a second convolution of the same size will yield 5^d active sites, and so on. This rapid growth of the number of active sites is a poor prospect when implementing modern convolutional network architectures that comprise tens or even hundreds of convolutions, such as VGG networks, ResNets, and DenseNets [3, 4, 10]. Of course, convolutional networks are not often applied to inputs that only have a single active site, but the aforementioned "dilation" problems are equally problematic when the input data comprises one-dimensional curves in spaces with two or more dimensions, or two-dimensional surfaces in three or more dimensions.

To address the problems with dilation of active sites, we propose two slightly different convolution operations for use in convolutional networks. What the two operations have in common is that they both ignore the ground state: they replace the ground state with a zero vector to simplify the convolution operations. The difference between both operations is in how they handle active sites: instead of automatically making a site active if any of the inputs to its receptive field is active (thereby dilating the set of active sites), our most efficient convolutional operation only considers the central input. As a result, the output set of active sites exactly mirrors that of the input set. We empirically demonstrate that use of our adapted convolutional operators allows us to build much deeper networks that achieve state-of-the-art results whilst requiring much fewer resources by preserving sparsity.

2.1 Submanifold Dilation

In Figure 1, we show an example of a one-dimensional curve that is embedded on a two-dimensional grid. The figure shows that even when we apply small 3×3 convolutions on this grid, the sparsity on the grid rapidly disappears. At the same time, if we restrict the output of the convolution only to the set of active input points, hidden layers in the network cannot capture a lot of information that may relevant to the classification of the curve. In particular, two neighboring connected components will be treated completely independently. Luckily, nearly all convolutional networks incorporate some form of pooling, or use strided convolutions. These operations are essential

[1] Note that the ground state does not necessarily have to be zero.

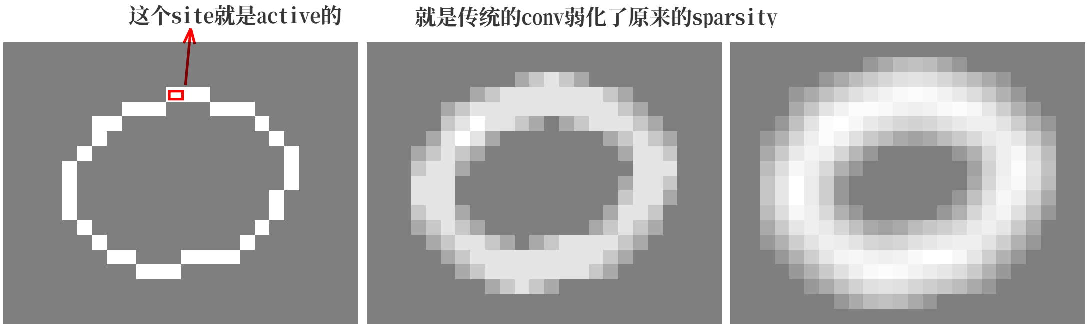


Figure 1: Example of "submanifold" dilation. **Left:** Original curve. **Middle:** Result of applying a regular 3×3 convolution with weights $1/9$. **Right:** Result of applying the same convolution again. The example shows that regular convolutions substantially reduce the sparsity of the feature maps.

in the sparse convolutional networks[2] we investigate, as they allow neighboring components to merge. In particular, the closer the components are, the smaller the number of poolings / strided convolutions is that is necessary for the components to merge in the hidden-layer representations.

2.2 Very Deep Convolutional Networks

In image classification, very deep convolutional networks with small filters, often of size 3×3 pixels and a padding of 1 pixel (to preserve the size of the feature maps), have proven to be very effective. Such small filters were used successfully in VGG networks, which have relatively wide layers. The introduction of residual networks (ResNets) showed that deeper but narrow networks with small filters are more efficient. The success of very deep ResNets, ResNeXt models, and DenseNets with bottleneck connections [3, 4, 12] shows that it can be useful to calculate a relatively small number of features at a time and amalgamate these features into a larger state variable, either by vector-addition or feature-vector concatenation.

Unfortunately, these techniques are impractical using existing sparse convolutional network implementations. One problem is that networks with multiple paths will tend to generate different sets of active paths, which would have to be merged to reconnect the outputs. It seems that this would be difficult to perform this merging efficiently. More importantly, ResNets and DenseNets generate such large receptive fields that sparsity would almost immediately be destroyed by the explosion in the number of active sites.

3 (Valid) Sparse Convolutions: SC and VSC

We define a *sparse convolution* $\text{SC}(m, n, f, s)$ with m input feature planes, n output feature planes, a filter size of f, and stride s. We assume f and s to be odd integers, but we can allow generalization to non-square filters, *e.g.*, $f = 1 \times 7$ or $f = 7 \times 1$, if we want to implement Inception-style factorised convolutions [11]. An SC convolution computes the set of active sites in the same way as a regular convolution: it looks for the presence of any active sites in its receptive field of size f^d. If the input has size ℓ then the output will have size $(\ell - f + s)/s$. An SC convolution differs from a regular convolution in that it discards the ground state for non-active sites by assuming that the input from those sites is exactly zero. Whereas this is a seemingly small change to the convolution operation, it may bring computational benefits in practice.

Next, we define a second type of sparse convolution, which forms the main contribution of this paper. Again, let f denote an odd number, or collection of odd numbers, *e.g.*, $f = 3$ or $f = 1 \times 7$. We define a *valid sparse convolution* $\text{VSC}(m, n, f, 1)$ as a modified $\text{SC}(m, n, f, 1)$ convolution. First, we pad the input with $(f - 1)/2$ on each side, so that the output will have the same size as the input. Next, we restrict an output site to be active if and only if the site at the corresponding site in the input is active (*i.e.*, if the central site in the receptive field is active). Whenever an

[2] By "sparse convolutional networks", we mean networks designed to operate on sparse input data. We do not mean networks that have sparse parameter matrices [5, 6].

Table 1: Computational and memory costs of three different convolutional operations at active and non-active sites: regular convolution (C), sparse convolution (SC), and valid sparse convolution (VSC). We consider convolutions of size 3 at a single location in d dimensions. Notation: a is the number of active inputs to the spatial location, m the number of input feature planes, and n the number of output feature planes.

Active	**Type**	**C**	**SC**	**VSC**
No	FLOPs	$3^d mn$	amn	0
	Memory	n	n	0
Yes	FLOPs	$3^d mn$	amn	amn
	Memory	n	n	n

output site is determined to be active, its output feature vector is calculated by the SC operation. Table 1 presents the computational and memory requirements of a regular convolution (C) and of our SC and VSC convolutions.

To construct convolutional networks using SC and VSC, we also need activation functions, batch normalization, and pooling. Activation functions are defined as usual, but are restricted to the set of active sites. Similarly, we define batch normalization in terms of regular batch-normalization applied over the set of active sites. Max-pooling MP(f, s) and average-pooling AP(f, s) operations are defined as a variant of SC$(\cdot, \cdot, f, s)$. MP takes the maximum of the zero vector and the input feature vectors in the receptive field. AP calculates f^{-d} times the sum of the active input vectors.

We also define a deconvolution operation DC$(\cdot, \cdot, f, s)$ as an inverse of the SC$(\cdot, \cdot, f, s)$ convolution [13]. The set of active output sites from a DC convolution is exactly the set of input active sites to the matching SC convolution. The set of connections between input-output sites is simply inverted.

3.1 Submanifold Convolutional Networks

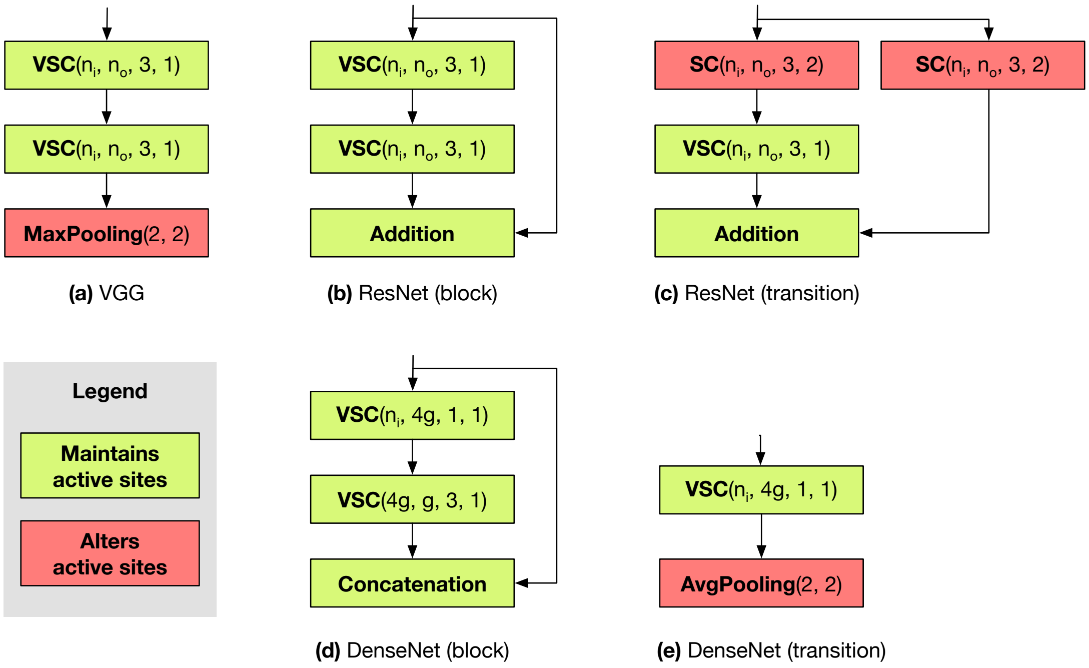
We use a combination of VSC convolutions, strided SC convolutions, and sparse pooling operations to build sparse versions of the popular VGG, ResNet, and DenseNet convolutional networks. The blocks we use in our networks are presented in Figure 2. We refer to our networks as *submanifold convolutional networks*, because they are optimised to process low-dimensional data living in a space of higher dimensionality.[3]

We use the name VGG to refer to networks that contain a number of VSC($\cdot$,$\cdot$,3,1) convolutions, separated by max-pooling [10]. Each convolution is followed by batch normalization and a ReLU non-linearity.

Similarly, we define “pre-activated ResNets” [3] in which most data processing is performed by pairs of VSC($\cdot$,$\cdot$,3,1) convolutions, and in which the residual connections are identity functions. Whenever the number of input / output features is different, we use a VSC($\cdot$,$\cdot$,1,1) instead. Whenever there is change of scale, we replace the first convolution and the residual connection by a SC($\cdot$, $\cdot$,3,2) convolution. This ensures that two branches can use the same hash table of active sites, and reduces additions to a simple sum of two equally sized matrices. The increased size of the residual connection’s receptive field also prevents excessive information loss.

We also experiment with submanifold DenseNets [4]. Herein, the word *dense* does not refer to a lack of spatial sparsity but rather to the pattern of connections between convolution operations. A simple DenseNet module is a sequence of convolutions in which each convolution takes as input the concatenated output of *all* the previous convolution operations. The bottleneck layers in our submanifold DenseNets are implemented in the same way as for ResNets.

[3] We note that this is a slight abuse of the term “submanifold”. Our input data may contain multiple connected components, and even a mixture of 1D and 2D objects embedded in 3D space.


VSC(n_i, n_o, 3, 1)
VSC(n_i, n_o, 3, 1)
MaxPooling(2, 2)
VSC(n_i, n_o, 3, 1)
VSC(n_i, n_o, 3, 1)
Addition
SC(n_i, n_o, 3, 2)
SC(n_i, n_o, 3, 2)
VSC(n_i, n_o, 3, 1)
Addition
Legend
Maintains active sites
Alters active sites
VSC(n_i, 4g, 1, 1)
VSC(4g, g, 3, 1)
Concatenation
VSC(n_i, 4g, 1, 1)
AvgPooling(2, 2)

(a) VGG

(b) ResNet (block)

(c) ResNet (transition)

(d) DenseNet (block)

(e) DenseNet (transition)

Figure 2: Modules used for building sub-manifold sparse convolutional networks: **(a)** VGG blocks comprise two VSC convolutions and a max-pooling operation; **(b)** ResNet blocks that maintain spatial resolution add the output of two VSC convolutions to the input; **(c)** ResNet blocks that reduce spatial resolution replace the first VSC convolution and the (implicit) identity function by a strided SC convolution; **(d)** DenseNet blocks that maintain spatial resolution concatenate the output of two VSC convolutions with the input; and **(e)** DenseNet blocks that reduce spatial resolution perform a single VSC convolution and an average-pooling. The four arguments of a convolution operator (SC or VSC) are the number of input planes n_i, the number of output planes n_o, the kernel size, and the stride, respectively. The two arguments of a pooling operator are the kernel size and the stride, respectively. The "growth rate" of a DenseNet [4] is denoted by g.

4 Implementation

To implement (V)SC convolutions efficiently, we store the state of a input/hidden layer in two parts: a hash table[4] and a matrix. The matrix has size $a \times m$ and contains one row for each of the a active sites. The hash table contains (location, row) pairs for all active sites: the location is a tuple of integer coordinates, and the row number indicates the corresponding row in the feature matrix. Given a convolution with filter size f, we define a *rule book* to be a collection $R = (R_i : i \in \{0, 1, ..., f-1\}^d)$ of f^d integer matrices of size $k_i \times 2$. To implement an SC(m, n, f, s) convolution, we:

- Iterate once through the the input hash-table. We build the output hash table and rule book on-the-fly by iterating over points in the output layer that receive input from a given point in the input layer. When an output site is visited for the first time, a new entry is created in the output hash table. Based on the spatial offset between the input and output points, a (input index, output index) pair is added to the rule book.
- Initialize the output matrix to all zeros. For each $i \in f$, there is a parameter matrix W^i with size $m \times n$. For each $j \in \{1, \ldots, k_i\}$, multiply the $R^i(j, 1)$-th row of the input feature matrix by W^i and add it to the $R^i(j, 2)$-th row of the the output feature matrix. This can be implemented very efficiently on GPUs because it is a matrix-matrix multiply-add operation.

To implement a VSC convolution, we re-use the input hash table for the output, and construct an appropriate rule book. Note that because the sparsity pattern does not change, the same rule book can be re-used in VGG/ResNet/DenseNet networks until a pooling or subsampling layer is encountered.

If there are a active points in the input layer, the cost of building the input hash-table is $O(a)$. For VGG/ResNet/DenseNet networks, assuming the number of active sites reduces by a multiplicative factor with each pooling operation, the cost of building all the hash-tables and rule-books is also $O(a)$, regardless of the depth of the network.

5 Experiments

We perform experiments on a 2D and a 3D dataset with sparse images. The **CASIA dataset** [7][5] contains samples of 3755 GBK level-1 characters with approximately 240 train and 60 test images per class. CJVK characters are good test cases for our models because they are a worst-case scenario for sparse convolutional networks: when drawn at scale 64×64, about 8% of the pixels are active, but this percentage rapidly decreases after pooling due to the small density of the pen strokes. This makes them a good test case for our models. The **ModelNet-40 dataset**[6] contain 2468 CAD models that contain shapes corresponding to 40 classes. We follow the preprocessing of [8] before feeding the models into our convolutional networks. All CAD models were rendered as surfaces at size 30^3.

5.1 Results on CASIA

We first experiment with two VGG architectures on the CASIA dataset. We trained all models for 100 epochs using batches of size 100, SGD with momentum 0.9, a weight decay of 10^{-4}, and a learning rate decay of 5% per epoch. For simplicity, we do not employ any data augmentation.

The architectures of our VGG networks and their performances are presented in Table 2. We observe that "regular" C convolutions and "sparse" SC convolutions achieve the same error: this result suggests that discarding the ground state has essentially no negative impact on performance. This is an argument for always discarding ground states, as it makes things easier computationally and algorithmically. Comparing SC with VSC convolutions, we observe a minimal loss in performance by considering only the valid part of the convolution. This minimal

[4] `https://github.com/sparsehash/sparsehash`

[5] `http://www.nlpr.ia.ac.cn/databases/handwriting/Online_database.html`

[6] `http://3dshapenets.cs.princeton.edu/`

Table 2: Classification error, computational requirements (in FLOPs), and memory load (measured by the number of hidden states) of two VGG networks with three different convolutional operations (C, SC, and VSC) on the CASIA dataset. The network architecture is specified in the top part of the table; the bottom part presents the performance characteristics of the networks. Lower is better.

Feature size	**VGG-A**			**VGG-B**		
64×64	2×16C3, MP2			2×16C3, MP2		
32×32	2×32C3, MP2			2×32C3, MP2		
16 ×16	2×48C3, MP2			2×64C3, MP2		
8×8	2×64C3, MP2			2×128C3, MP2		
4×4	2×96C43			2×256C3		
4×4	128C4			512C4		
Convolution	**C**	**SC**	**VSC**	**C**	**SC**	**VSC**
Class. error (in %)	4.15	4.11	4.67	3.45	3.47	3.82
FLOPs $(\times 10^6)$	41	25	7.4	72	50	23
Hidden states $(\times 10^3)$	233	133	41	254	155	55

Table 3: Classification error, computational requirements (in FLOPs), and memory load (measured by the number of hidden states) of four ResNet networks with two different convolutional operations (C and VSC) on the CASIA dataset. The network architecture is specified in the top part of the table; the bottom part presents the performance characteristics of the networks. Lower is better.

Feature size	**ResNet-A**		**ResNet-B**		**ResNet-C**		**ResNet-D**	
64×64	16C3, MP2		16C3, MP2		16C3, MP2		16C3, 2×ResNet(16)	
32×32	2×ResNet(16)		2×ResNet(16)		2×ResNet(16)		2×ResNet(32)	
16×16	2×ResNet(32)		2×ResNet(32)		2×ResNet(32)		2×ResNet(48)	
8×8	2×ResNet(48)		2×ResNet(64)		2×ResNet(64)		2×ResNet(64)	
4×4	2×ResNet(96)		2×ResNet(128)		2×ResNet(128)		2×ResNet(128)	
4×4	128C4		256C4		2×ResNet(256)		2×ResNet(256)	
2×2	–		–		512C2		512C2	
Convolution	**C**	**VSC**	**C**	**VSC**	**C**	**VSC**	**C**	**VSC**
Class. error (in %)	3.97	4.09	3.78	3.87	3.70	3.78	3.51	3.57
FLOPs $(\times 10^6)$	32	14.5	40	21	51	31	88	49
Hidden states $(\times 10^3)$	193	63	200	71	209	81	471	153

loss in accuracy does facilitate great computational improvements: networks using VSC use 2 to 3× less computation and memory.

Next, we performed experiments on CASIA with submanifold ResNets. The key difference between our implementation of ResNets and regular ResNets is that stride-2 ResNet modules use SC(·,·,3,2) convolutions for the strided convolution, rather than SC(·,·,1,2). This change is necessary to ensure the two branches produce the same set of active sites, which simplifies bookkeeping and turns the addition operation into a simple matrix-matrix addition. Unlike the VSC convolutions that are used in most layers, the SC(·,·,3,2) we use after downsampling leads sites to be active if *any* of its inputs are active, which avoids information loss in the transition. The architectures of our ResNet networks and their performances are presented in Table 3. The results with ResNets are in line with those obtained using VGG networks: we obtain reductions in computational and memory requirements by at least a factor of 2 at a minimal loss in accuracy.

We also performed experiments with DenseNets; please see Table 4.

Next we experimented with adding extra connections to VGG networks to increase the effective receptive fields of the hidden states; see Table 5 for results. In the table, $\{a, b\}$ denotes a VSC$(\cdot, a, 3, 1)$ convolution performed in parallel with a chain of SC$(\cdot, b, 3, 2)$-VSC$(b, b, 3, 1)$-DC$(b, b, 3, 2)$ operations; outputs are concatenated to produce $a + b$ output feature planes.

Table 4: Classification error, computational requirements (in FLOPs), and memory load (measured by the number of hidden states) of two DenseNet networks with SVC convolutions on the CASIA dataset. The network architecture is specified in the top part of the table; the bottom part presents the performance characteristics of the networks. Lower is better.

Feature size	**DenseNet-A**	**DenseNet-B**
64×64	VSC(3,16,3,1), MP	VSC(3,16,3,1), MP
32×32	2 × 16 BC layers, Transition	4 × 16 BC layers, Transition(Compress=0.5)
16×16	2 × 16 BC layers, Transition	4 × 16 BC layers, Transition(Compress=0.5)
8×8	2 × 16 BC layers, Transition	4×16 BC layers, Transition(Compress=0.5)
4×4	2×16 BC layers	4 × 16 BC layers
4×4	SC(144,256,4,1)	SC(128,256,4,1)
Class. error (in %)	4.49	4.09
FLOPs ($\times 10^6$)	8.7	15
Hidden states ($\times 10^3$)	84	136

Table 5: Evaluation of the effect of adding SC-VSC-DC branches to a VGG network on the CASIA dataset. Lower values are better.

Feature size	**VGG-C**	**VGG$^+$-C**	**VGG-D**	**VGG$^+$-D**
63×63	$\{16,0\}\times 2$	$\{16,8\}\times 2$	$\{16,0\}\times 2$	$\{16,8\}\times 2$
31× 31	$\{32,0\}\times 2$	$\{32,8\}\times 2$	$\{32,0\}\times 2$	$\{32,8\}\times 2$
15 ×15	$\{48,0\}\times 2$	$\{48,16\}\times 2$	$\{64,0\}\times 2$	$\{64,16\}\times 2$
7×7	$\{64,0\}\times 2$	$\{64,16\}\times 2$	$\{128,0\}\times 2$	$\{96,16\}\times 2$
3×3	$\{96,0\}\times 2$	$\{96,16\}\times 2$	$\{256,0\}\times 2$	$\{256,32\}\times 2$
3×3	128C3	128C3	512C3	512C3
Class. error (in %)	4.50	4.17	3.73	3.59
FLOPs ($\times 10^6$)	10	16	24	29
Hidden states ($\times 10^3$)	50	83	63	94

To simplify the network design, we switched to size-3 stride-2 max-pooling, matching the SC convolutions in the SC-VSC-DC branches, and reduce the input size from 64×64 to 63×63. Figure 3 presents an overview of all our results on the CASIA dataset.

5.2 Results on ModelNet

In a second set of experiments, we compare two submanifold VGG networks with a state-of-the-art dense convolutional network on the ModelNet-40 dataset [8]. The results of these experiments are shown in Table 6: the left part of the table shows the architecture and performance of our submanifold VGG networks, whereas the right part of the table shows that of the dense 3DNiN network [8]. The results clearly demonstrate that submanifold have the potential for designing convolutional networks for sparse data that obtain state-of-the-art performance with limited computational requirements: in particular, our VGG-A network makes 2% more errors at 13× fewer computations, and our VGG-B performs roughly on par with the dense 3DNiN whilst performing ∼5× fewer computations.

6 Related Work

This paper is not the first to study sparse convolutional networks [1, 2, 9]. Most prior networks for sparse data implements a standard convolutional operator that increases the number of active sites with each layer [1, 2]. By contrast, our submanifold convolutional networks allows sparse data to be processed whilst retraining a much greater degree of sparsity. We have shown that this makes it practical to train deep and efficient VGG and ResNet models.

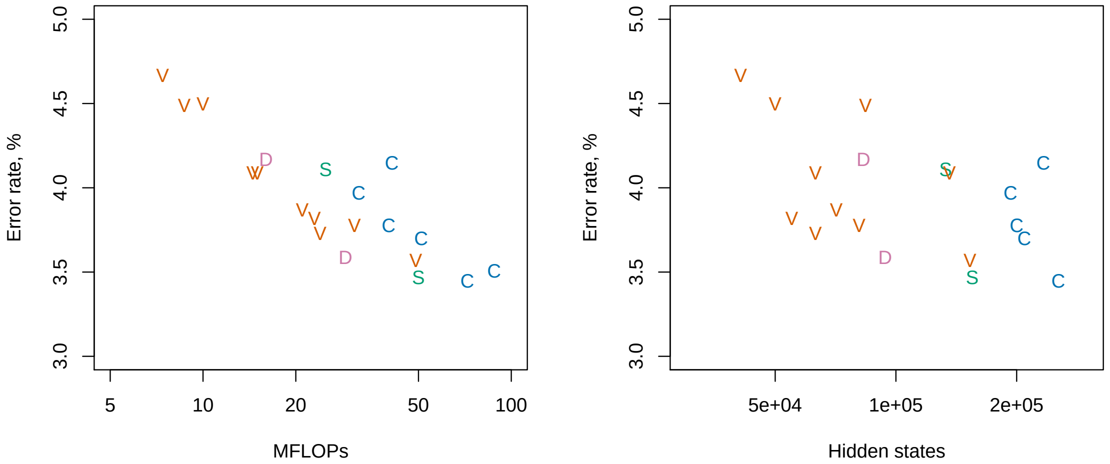


Figure 3: Overview of all experiments on the CASIA dataset. We denote submanifold convolutional networks by **V**, submanifold networks with SC-VSC-DC connections by **D**, regular sparse networks by **S**, and regular dense networks by **C**. Locations closer to the bottom-left corner are better.

Table 6: Comparison of two submanifold VGG networks (left) with the dense 3DNiN network (right) on the ModelNet-40 dataset. Lower values are better.

Feature size	**VGG-A**	**VGG-B**	**Feature size**	**3DNiN**
32× 32	3×8C3, MP2	2×16C3, MP2	30×30	48C6/2
16 ×16	3×16C3, MP2	2×32C3, MP2	13×13	2×48C1,96C5/4
8 ×8	3×24C3, MP2	2×64C3, MP2	5×5	2×96C1, 512C3/2
4×4	3×32C3	2×128C3	2×2	MP2
4×4	32C4	128C4	–	–
Convolution	**VSC**	**VSC**		**C**
Class. error (in %)	13.6	11.8		11.2
FLOPs ($\times 10^6$)	9.2	28		124
Hidden states ($\times 10^3$)	55	78		368

Submanifold convolutional networks are also much sparser than OctNets [9]. OctNet stores data in oct-trees: a data structure in which the grid cube is progressively subdivided into 2^3 smaller sub-cubes until the sub-cubes are either empty or contain a single active site. To compare the efficiency of OctNets with that of submanifold convolutional networks, we picked a random sample from the ModelNet-40 dataset and rendered it in a cube with 32^3 grid points. The resulting grid had 423 active sites, which corresponds to 1.3% of the total number of sites. Each active site had on average 12.4 active neighbors (the maximum possible number of neighbors is 27). VSC convolutions, therefore, require only 0.6% of the work of a dense (C) convolution. However, in the OctTree, 80%, 13%, 4%, and 3% of the volume of the cube is covered by sub-cubes of size 8^3, 4^3, 2^3 and 1^3, respectively. As a result, an OctNet convolution, which operates over the surfaces of the smaller cubes, requires about 35% of the computations that a dense (C) convolution requires. In this particular example, an OctNet convolution thus has a computational cost that is 60 times higher than that of a VSC convolution.

Submanifold convolutional networks also have advantages in terms of memory requirements. In particular, a submanifold network stores a single feature vector for each of the active sites. By contrast, OctTrees have about twice as many empty child nodes as active nodes, which implies they have to store roughly three times as many features as a submanifold convolutional network.

Having said that, some of the ideas of [9] may be combined with VSC convolutions. In particular, it is possible to use oct-trees as a specialized hash function in VSC convolutions. Such an oct-tree-based hash function has the potential to be faster than a standard universal hash function that operates on integer tuple keys, like in our implementation of VSC.

7 Conclusion

We introduced a new sparse convolutional operator, called *valid sparse convolution* (VSC), that facilitates the design of efficient, deep convolutional networks for sparse data. We have shown that VSC convolutions lead to substantial computational savings whilst maintain state-of-the-art accuracies on two datasets: a dataset comprising one-dimensional manifolds embedded in two-dimensional space, and a dataset comprising two-dimensional surfaces embedded in three-dimensional space.

As part of this paper, we are releasing easy-to-use implementations of VSC and the other sparse operations we used in the networks described in this paper. We will also release code to reproduce the results of our experiments.

References

[1] M. Engelcke, D. Rao, D. Z. Wang, C. H. Tong, and I. Posner. Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks. 2016.

[2] B. Graham. Sparse 3D convolutional neural networks. In *BMVC* 2015.

[3] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. *ECCV*, 2016.

[4] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. 2016.

[5] Y. LeCun, J. Denker, and S. Solla. Optimal brain damage. In *NIPS*, 1990.

[6] B. Liu, M. Wang, H. Foroosh, M. Tappen, , and M. Penksy. Sparse convolutional neural networks. In *CVPR*, 2015.

[7] C.-L. Liu, F. Yin, D.-H. Wang, and Q.-F. Wang. CASIA online and offline chinese handwriting databases. In *ICDAR*, pages 37–41. IEEE Computer Society, 2011.

[8] C. R. Qi, H. Su, M. Niessner, A. Dai, M. Yan, and L. J. Guibas. Volumetric and multi-view CNNs for object classification on 3D data, Apr. 29 2016.

[9] G. Riegler, A. O. Ulusoy, and A. Geiger. Octnet: Learning deep 3D representations at high resolutions. *CoRR*, abs/1611.05009, 2016.

[10] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. 2014.

[11] 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.

[12] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. *arXiv preprint arXiv:1611.05431*, 2016.

[13] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus. Deconvolutional networks. In *CVPR*, 2010.