Deep Convolutional Sum-Product Networks

Cory J. Butz  
butz@cs.uregina.ca  
University of Regina  
Canada

Jhonatan S. Oliveira  
oliveira@cs.uregina.ca  
University of Regina  
Canada

André E. dos Santos  
dossantos@cs.uregina.ca  
University of Regina  
Canada

André L. Teixeira  
teixeira@cs.uregina.ca  
University of Regina  
Canada

Abstract
We give conditions under which convolutional neural networks (CNNs) define valid sum-product networks (SPNs). One subclass, called convolutional SPNs (CSPNs), can be implemented using tensors, but also can suffer from being too shallow. Fortunately, tensors can be augmented while maintaining valid SPNs. This yields a larger subclass of CNNs, which we call deep convolutional SPNs (DCSPNs), where the convolutional and sum-pooling layers form rich directed acyclic graph structures. One salient feature of DCSPNs is that they are a rigorous probabilistic model. As such, they can exploit multiple kinds of probabilistic reasoning, including marginal inference and most probable explanation (MPE) inference. This allows an alternative method for learning DCSPNs using vectorized differentiable MPE, which plays a similar role to the generator in generative adversarial networks (GANs). Image sampling is yet another application demonstrating the robustness of DCSPNs. Our preliminary results on image sampling are encouraging, since the DCSPN sampled images exhibit variability. Experiments on image completion show that DCSPNs significantly outperform competing methods by achieving several state-of-the-art mean squared error (MSE) scores in both left-completion and bottom-completion in benchmark datasets.

Introduction
Generative models are of current interest in the deep learning community, including generative adversarial networks (GANs) (Goodfellow et al. 2014), variational autoencoders (Kingma and Welling 2014), neural autoregressive distribution estimators (Larochelle and Murray 2011), pixel recurrent neural networks (Oord, Kalchbrenner, and Kavukcuoglu 2016), and convolutional arithmetic circuits (Sharir et al. 2018). Convolutional neural networks (CNNs) (Goodfellow, Bengio, and Courville 2016) can be used in GANs. Sum-product networks (SPNs) (Poon and Domingos 2011) are a generative model that have received limited attention from the deep learning community (Peharz et al. 2018). An SPN is a directed acyclic graph (DAG), where leaf nodes are tractable distributions and each internal node is either a sum or product operation. A valid SPN defines a joint probability distribution and allows for efficient inference (Poon and Domingos 2011).

Conditions are given as to when subclasses of CNNs define valid SPNs, including convolutional layer filters of certain sizes and non-overlapping windows in sum-pooling layers. Satisfaction of these conditions yields a subclass of CNNs, called convolutional SPNs (CSPNs). CSPNs permit a vectorized representation allowing for exploitation of tensor libraries such as Tensorflow, but they also can suffer from being too shallow, and it is known that deep SPNs are more expressive than shallow SPNs (Delalleau and Bengio 2011).

We introduce deep convolutional sum-product networks (DCSPNs). DCSPNs permit the convolutional and sum-pooling layers to form rich DAG structures by augmenting layer tensors under conditions that maintain decomposability and completeness. As a decomposable and complete SPN is a valid SPN, our main result is that DCSPNs are a larger subclass of CNNs that define valid SPNs. DCSPNs are a rigorous probabilistic model. As such, they can exploit probabilistic reasoning, including marginal inference and most probable explanation (MPE) inference. This allows an alternative method for learning DCSPNs using vectorized differentiable MPE. We show how to vectorize MPE using a mask algorithm and how it plays a role similar to the GANs generator. Image sampling is yet another application demonstrating the robustness of DCSPNs. This involves a minor modification to the mask algorithm. Our preliminary results on image sampling are promising, since the DCSPN sampled images exhibit variability. Experimental results on left- and bottom-completion like those in Table 1 show DCSPNs achieve state-of-the-art performance using both vertical and horizontal sum-pooling windows, which leverage local structure in the image data in both directions. Applying a simple low pass filter as a post-processing smoothing operation lowers the mean squared error (MSE) score from 455 to 401 for left-completion in Olivetti.

Table 1: Mean squared error (MSE) scores in Olivetti Face.

<table>
<thead>
<tr>
<th></th>
<th>left</th>
<th>bottom</th>
</tr>
</thead>
<tbody>
<tr>
<td>P&amp;D (Poon and Domingos 2011)</td>
<td>942</td>
<td>918</td>
</tr>
<tr>
<td>ICNN (Amos, Xu, and Kolter 2017)</td>
<td>833</td>
<td>-</td>
</tr>
<tr>
<td>D&amp;V (Dennis and Ventura 2012)</td>
<td>779</td>
<td>782</td>
</tr>
<tr>
<td>DCGAN (Yeh et al. 2017)</td>
<td>935</td>
<td>707</td>
</tr>
<tr>
<td>DCSPN</td>
<td>455</td>
<td>503</td>
</tr>
</tbody>
</table>

Copyright © 2019, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.
Sum-Product Networks

We denote random variables by uppercase letters, such as X and Y, possibly with subscripts, and their values by corresponding lowercase letters x and y. Sets of random variables are denoted by boldfaced uppercase letters and their combined values by corresponding boldfaced lowercase letters.

A sum-product network (SPN) (Poon and Domingos 2011) over variables X can be defined as a directed acyclic graph (DAG) containing three types of nodes: leaf distributions, sums, and products. Leaves are tractable distribution functions over Y ⊆ X. Sum nodes S compute weighted sums \( S = \sum_{N \in Ch(S)} w_{S,N} N \), where Ch(S) are the children of S and \( w_{S,N} \) are weights that are assumed to be non-negative and normalized (Peharz et al. 2015). Product nodes P compute \( P = \prod_{N \in Ch(P)} N \). The value of an SPN, denoted \( S(x) \), is the value of its root.

The scope of a sum or product node \( N \) is recursively defined as \( sc(N) = \bigcup_{C \in Ch(N)} sc(C) \), while the scope of a leaf distribution is the set of variables over which the distribution is defined. A valid SPN defines a joint probability distribution and allows for efficient inference (Poon and Domingos 2011). The following two structural constraints on the DAG guarantee validity. An SPN is complete if, for every sum node, its children have the same scope. An SPN is decomposable if, for every product node, the scopes of its children are pairwise disjoint.

Nodes of an SPN can be organized as layers for a vectorized implementation. The discussion here draws from (Vergari, Di Mauro, and Esposito 2016). The SPN input layer is formed by the leaf distribution nodes. Let \( L(x) \in \mathbb{R}^r \) be the output of a generic layer with \( s \) nodes and input \( x \). The value of a sum layer with an input \( x \) from \( r \) input nodes is

\[
L(x) = \log(w \times x),
\]

where \( w \in \mathbb{R}^{s \times r} \) is a matrix of weights defining sparse connections:

\[
w_{ij} = \begin{cases} w_{ij} & \text{if edge } (i, j) \text{ exists} \\ 0 & \text{otherwise} \end{cases}
\]

and \( x \in [0, 1]^r \) represents the input probability values. Similarly, the value of a product layer with input \( x \) is

\[
L(x) = \exp(p \times x),
\]

where \( p \in \{0, 1\}^{s \times r} \) is a matrix of sparse connections:

\[
p_{ij} = \begin{cases} 1 & \text{if edge } (i, j) \text{ exists} \\ 0 & \text{otherwise} \end{cases}
\]

and \( x \in [0, 1]^r \) represents the input probability values in log-space. In this formulation, the exponential and logarithmic functions act as nonlinearities.

In our paper, we relate SPNs with convolutional neural networks (CNNs) (Goodfellow, Bengio, and Courville 2016). In general, CNNs are formed by convolutional and pooling layers. A convolutional layer can be constructed by applying a convolutional operation with a filter on a previous layer output. A pooling layer can be built by applying a max or average operation over some elements of the previous layer with respect to a sliding window. A sum-pooling layer can be easily obtained from an average-pooling layer.

Convolutions SPNs

We study when subclasses of CNNs define valid SPNs.

First, note that a vectorized SPN can represent a CNN. SPN sum layers correspond to CNN convolutional layers. Here, sum layer weights are convolutional filters and the sum layer value computes the convolution operation. Similarly, SPN product layers correspond to CNN sum-pooling layers. The sum-pooling window computes the product layer value in log-space.

Example 1 Figure 1 illustrates an SPN with 3 layers being represented by a CNN with 3 layers, where colours represent node scopes. The input layer is the same for both, while the sum and product layers in the SPN are convolutional and sum-pooling layers in the CNN, respectively.

Conversely, we show how a CNN can represent a vectorized SPN in log-space. Representational, convolutional, and sum-pooling layers in CNNs can represent the input, sum, and product layers in SPNs, respectively. A representational layer (Sharir et al. 2018) is formed by applying \( n \) representation functions \( f_1, \ldots, f_n : \mathbb{R}^s \to \mathbb{R} \) over \( s \)-dimensional local patches of the dataset. We apply the logarithmic function in representational layers so as to correspond to SPN input layers in log-space. For instance, \( n \) Gaussian distributions can be used as representation functions to map patches of dataset instances to \( n \) values in the representational layer.

The value of a convolutional layer with input \( x \) is computed element-wise depending on filter \( w \)’s size with depth \( c \) being either \( m \)-by-\( n \) (top) or height-by-width (bottom):

\[
L_{ij}(x) = \begin{cases} \sum_{q=0}^{m-1} \sum_{l=0}^{n-1} \log(w_{ql}) + x_{(i+q)(j+l)} \\ \sum_{k=0}^{s-1} \log(w_{ijk}) + x_{ijk}. \end{cases}
\]

Convolutions layers in (3) relate to sum layers in (1). The value of a sum-pooling layer with sliding window size \( m \)-by-\( n \) and input \( x \) is computed element-wise as:

\[
L_{ij}(x) = \sum_{q=0}^{m-1} \sum_{l=0}^{n-1} x_{(i+q)(j+l)}.\]

![Figure 1: Input, convolutional, and sum-pooling layers in a CNN representing input, sum, and product layers in an SPN.](image-url)
Sum-pooling layers in (4) relate to product layers in (2).

We now define a subclass of CNNs that represent SPNs by restricting convolutional and sum-pooling layers.

**Definition 1** A convolutional SPN (CSPN) is a CNN formed under the following two restrictions: (i) convolutional layer filters must have height and width of 1-by-1 or $h$-by-$w$, where $h$ and $w$ are the height and width of the convolutional layer; respectively; (ii) in sum-pooling layers, the horizontal and vertical stride of the sliding window must be at least the width and the height of the window itself, respectively.

**Example 2** Consider the CSPN illustrated in Figure 2, where colours represent node scopes. The dataset is mapped to a 4-by-4 representational layer using 2 representation functions. Next, a convolutional layer is formed with 6 filters of size 4-by-4. A 2-by-2 sum-pooling layer is then built with a 2-by-2 sliding window. Finally, the root node is a 1-by-1 convolutional layer obtained using one 2-by-2 filter.

Next, we show that CSPNs represent valid SPNs.

**Theorem 1** Let $C$ be a CSPN formed with respect to a CNN $C'$ in Definition 1. Then, $C$ is a valid SPN.

**Proof 1** To be valid, $C$ must be both complete and decomposable. The restriction on each convolutional layer filter in $C'$ to be 1-by-1 or $h$-by-$w$ ensures that the sum in the convolution operation in (3) occurs over elements of the same scope. Thus, $C$ is complete. The restriction on the sum-pooling layers with the horizontal and vertical stride of the sliding window being at least the size of the window itself guarantees that the multiplication (summation, in log-space) in the pooling operation in (4) occurs over elements with disjoint scopes in the current and subsequent layers. Therefore, $C$ is decomposable. By definition, $C$ is a valid SPN.

CSPNs can struggle with depth, since the sum-pooling window size quickly reduces the size of the layers. For example, as depicted by the CSPN in Figure 2, the 4-by-4 representational layer reduced to the 1-by-1 root layer after two convolutions and one sum-pooling operation. As deep SPNs are more expressive than shallow SPNs (Delalleau and Bengio 2011), we now turn our attention to introducing deep CSPNs.

We build deep CSPNs by considering two related issues. First, we seek a DAG structure rather than a chain structure. Second, since each node is implemented as a tensor, each combination of tensors must be done such that a valid SPN is maintained. The next section formalizes these ideas.

**Deep Convolutional SPNs**

In this section, we introduce a tractable generative model, called deep convolutional SPNs (DCSPNs).

We denote by $T$ a tensor of rank (order) $n$ and dimensions $m$ in each mode. That is, $T$ is a multi-dimensional array, specified by a shape with $n$ indexes $[d_1, \ldots, d_n]$, each ranging in $[m] \equiv \{1, \ldots, m\}$. Without loss of generality, a layer is represented as a rank 4 tensor with shape $[b, h, w, c]$, where $b$ is the batch (number of instances) being considered, and $h$, $w$, and $c$ are the height, width, and channel (depth), respectively. Tensor elements are SPN nodes. In a sum layer, tensor elements are sum nodes, while in a product layer, tensor elements are product nodes.

Let $T_1$ and $T_2$ be two tensors with the same height and width. The channel augmentation of $T_1$ and $T_2$ is the tensor with the same height and the same width formed by concatenating $T_1$ and $T_2$ with respect to the channel axis.

**Example 3** Let $T_1$ and $T_2$ be the two tensors in Figure 3 (a) (left, right), respectively. Then, the channel augmentation of $T_1$ and $T_2$ is the tensor depicted in Figure 3 (b).

Let $T_1$ and $T_2$ be two tensors with the same depth and height. The width augmentation of $T_1$ and $T_2$ is the tensor with the same depth and the same height formed by concatenating $T_1$ and $T_2$ with respect to the width axis. The height augmentation is defined similarly.

**Example 4** Let $T_1$ and $T_2$ be the two tensors in Figure 3 (c) (left, right), respectively. Then, the width augmentation of $T_1$ and $T_2$ is the tensor depicted in Figure 3 (d).

**Definition 2** A deep convolutional sum-product network (DCSPN) $D$ over $n$ variables $X_1, \ldots, X_n$ is a rooted DAG whose leaves are representational layers and whose internal nodes are convolutional and sum-pooling layers. A convolutional layer with more than one child is formed by recursively applying channel augmentation on all its children.
sum-pooling layer with more than one child is formed by recursively applying either height or width augmentation on all its children.

**Example 5** Consider the DCSPN in Figure 4. The dataset is mapped to 2 representational layers, each using 2 representation functions. In the upper branches, after a convolutional and a sum-pooling layer, a channel augmentation is used to form a convolutional layer. In the lower branch, after a convolutional and sum-pooling layer, a width augmentation is used to form a sum-pooling layer. Lastly, a sum-pooling layer is followed by the root convolutional layer.

By construction, DCSPNs represent SPNs. We now provide structural conditions under which DCSPNs represent valid SPNs.

**Lemma 1** Consider a DCSPN $\mathcal{D}$ where every convolutional layer has children over the same element-wise scope. Connect the children of each convolutional layer using channel augmentation. Then, $\mathcal{D}$ is a complete SPN.

**Proof 2** Consider the SPN defined by the DCSPN $\mathcal{D}$. Channel augmentation over all children of each convolutional layer yields tensors (layers) with elements of the same scope aligned along the channel axis. By construction, summation in the convolutional operation in (3) occurs over the channel axis. Hence, every summation involves operands of the same scope. By definition, $\mathcal{D}$ is complete.

Lemma 1 is important as it establishes one practical method for combining tensors such that the resulting DCSPN is complete. Lemma 2, given next, provides a practical method for combining tensors such that the obtained DCSPN is decomposable.

**Lemma 2** Consider a DCSPN $\mathcal{D}$ where every sum-pooling layer has children with element-wise disjoint scopes. Connect the children of each sum-pooling layer using either height or width augmentation. Then, $\mathcal{D}$ is a decomposable SPN.

**Proof 3** Consider the SPN defined by the DCSPN $\mathcal{D}$. Width augmentation over all children of each sum-pooling layer yields tensors (layers) with elements of disjoint scopes at corresponding positions in the width axis. A similar property holds for height augmentation and the height axis. By construction, multiplication in the log-space sum-pooling operation in (4) occurs over either height or width or both axes. Thus, every multiplication involves operands of disjoint scopes. By definition, $\mathcal{D}$ is decomposable.

We now show the desired result.

**Theorem 2** Let $\mathcal{D}$ be a DCSPN in which the tensors of each convolutional and sum-pooling layer are connected in accordance to Lemmas 1 and 2. Then, $\mathcal{D}$ is a valid SPN.

**Proof 4** Consider the SPN defined by the DCSPN $\mathcal{D}$. By Lemma 1, $\mathcal{D}$ is necessarily a complete SPN. Moreover, $\mathcal{D}$ is guaranteed to be a decomposable SPN, by Lemma 2. By definition, since $\mathcal{D}$ is both complete and decomposable, $\mathcal{D}$ is a valid SPN.

Theorem 2 is significant for two reasons. First, as DCSPNs define joint probability distributions, they are tractable deep generative models. Second, DCSPNs are robust in that there are many possible DAG structures in theory satisfying Theorem 2. We will examine later a specific DAG structure that performed exceptionally well in practice.

The DCSPN parameters can be learned with methods such as expectation maximization and gradient descent, similar to SPNs (Poon and Domingos 2011). More specifically, given a DAG structure, we consider learning its parameters $\Theta$ (weights of sum nodes) with the maximum likelihood principle. This optimization problem can be equivalently seen as minimizing the negative log-likelihood (NLL) loss function $\mathcal{L}$ (Sharir et al. 2018):

$$\mathcal{L}(\Theta) = \mathbb{E}[-\log S(x)].$$ (5)

Given a DCSPN DAG structure coupled with parameters, we turn our attention to applications demonstrating the effectiveness of DCSPNs.
DCSPN Image Completion

Image completion is a difficult task (Dennis and Ventura 2012) that has been studied quite extensively in graphic and vision communities (Poon and Domingos 2011). One SPN approach is to use most probable explanation (MPE) inference (Poon and Domingos 2011; Dennis and Ventura 2012). In our case, we will apply MPE inference in CNNs through the unifying framework of DCSPNs.

MPE inference in SPNs is NP-hard (Peharz et al. 2017; Conaty, Mauá, and de Campos 2017). Hence, approximate MPE inference is used (Poon and Domingos 2011; Peharz et al. 2017) and is briefly summarized as two passes. In the forward pass, replace sum nodes with max nodes. The MPE assignment is then obtained via Viterbi-style backtracking. That is, start at the root and follow all children of product nodes and one maximizing child of max nodes.

We propose a method for vectorizing MPE in DCSPNs. In the forward pass, replace the sum operation in a convolutional layer with the max operation. For the backward pass, vectorize the Viterbi-style backtracking by using mask propagation. Starting at the root layer, a tensor mask of all ones initiates the backward propagation. Using a slice of the mask from each parent, a layer computes one mask for its children. Participating children nodes assume value 1; otherwise, 0. Lastly, representational layer masks indicate which representation function is used in the MPE assignment.

Algorithm 1 formally describes our mask MPE backward propagation method, which can utilize sparse tensor optimization techniques available in libraries. Lines 1-2 perform initialization. The DCSPN is traversed according to any topological ordering. For every layer \( L \), we perform two tasks: (i) combine slices from parent masks; and (ii) compute one mask for its children using (i).

Consider task (i). If \( P \) is a convolutional layer in line 8, slice the channel (depth) axis at the position of \( \Delta P \) in the mask from each of its parents. Next, compute the channel augmentation of all slices. Now, suppose \( P \) is a sum-pooling layer in line 11, slice the width axis at the position of \( \Delta P \) in the mask from each of its parents. Observe that here the DAG dictates whether width or height augmentation is applied on all slices. A mask of all ones is used for the root layer. The output of task (i) is the mask \( PM \).

Now consider task (ii). Let \( L \) be a convolutional layer in line 18. The activation of \( L \) is saved in the forward pass and represents the product of the filter and the input values during the convolution. Determine the position of a maximizing child using the \( \text{ARGMAX} \) function in line 20. The \( \text{ONE}_\text{HOT} \) function builds a mask of zeros, except for a 1 in the maximizing child position. Lastly, \( L \)'s mask is the element-wise multiplication of the \( \text{ONE}_\text{HOT} \) mask and \( PM \). Next, if \( L \) is a sum-pooling layer, then resize \( PM \) in line 24 to match \( L \)'s previous layer size by upsampling \( PM \) using the nearest neighbour method. Lastly, if \( L \) is a representational leaf layer, then its mask in line 26 is \( PM \). Finally, return all leaf masks saved in line 26.

The output of Algorithm 1 indicates which representation function is selected in the Viterbi-style backtracking. This, in turn, can be used for computing the MPE assignment of each selected representation function.

Algorithm 1 Mask MPE Backward Propagation

| Input: a DCSPN \( \mathcal{D} \) with forward pass node values |
| Output: representational layer leaf masks |

### Main:

1: for \( L \) in \( \mathcal{D} \) do \( \triangleright \) Initialization
2: \( \text{masks}[L] = \emptyset \)
3: for \( L \) in \text{TOPOLOGICAL\_SORT}(\mathcal{D}) \) do
4: \( \triangleright \) Task (i): Combine parent layer masks (PM)
5: \( PM = \emptyset \)
6: for \( P \) in \( Pa(L) \) do
7: \( \triangleright \) Always slice at the position of \( L \)
8: if \( P \) is convolutional then
9: \( S = \text{slice channel axis of} \text{masks}[P] \)
10: Channel augmentation of \( PM \) and \( S \)
11: else \( \triangleright \) \( P \) is sum-pooling
12: \( \triangleright \mathcal{D} \) may dictate height augmentation instead
13: \( S = \text{slice width axis of} \text{masks}[P] \)
14: Width augmentation of \( PM \) and \( S \)
15: if \( L \) is the root then
16: \( PM = \text{ONES}([1, 1, 1]) \)
17: \( \triangleright \) Task (ii): Compute current layer mask (CM)
18: if \( L \) is convolutional then
19: \( A = \text{activation of} \ L \) in the forward pass
20: \( C = \text{ARGMAX}(A) \)
21: \( M = \text{ONE}_\text{HOT}(C) \)
22: \( \text{mask} = M \odot PM \) \( \triangleright \) Hadamard product
23: else if \( L \) is sum-pooling then
24: \( \text{mask} = \text{RESIZE}(PM) \) \( \triangleright \) Upsample
25: else \( \triangleright \) Representational (Leaf)
26: \( \text{leaf\_masks}[L] = PM \) \( \triangleright \) Leaf mask
27: \( \text{masks}[L] = \text{mask} \)
return \( \text{leaf\_masks} \) \( \triangleright \) Masks for leaves of \( \mathcal{D} \)

Experiments

As promised, we now describe a DCSPN DAG structure that performed exceptionally well in practice. A convolutional layer follows every representational layer and every sum-pooling layer. All convolutional layers have filter sizes height-by-width matching the layer size. Two sum-pooling layers follow each convolutional layer: one with a window size of 1-by-2 and the other 2-by-1. Alternate the window sizes of 1-by-2 and 2-by-1 with 2-by-2 and 2-by-2 every \( n \) layers. This hyperparameter \( n \) is tuned per dataset and varied between 70 and 100 in our experiments. For each dataset, we randomly set aside one third (up to 50 images) for testing. For training, we use ADAM (Kingma and Ba 2014) with a learning rate of 0.005. Four Gaussian representation functions are used per pixel (variable), where the mean and variance are computed from equal quantiles of pixel intensities. In practice, we observed better accuracy when maintaining a sum operation in convolutional layers rather than a max operation during MPE. (Poon and Domingos 2011) made a similar observation.

We compare DCSPNs with deep convolutional generative adversarial networks (DCGANs) (Yeh et al. 2017) using the publicly available code from (Amos 2016). Here, we use
Table 2: MSE scores in Caltech datasets.

<table>
<thead>
<tr>
<th></th>
<th>left</th>
<th>P&amp;D</th>
<th>D&amp;V</th>
<th>DCGAN</th>
<th>DCSPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Face</td>
<td>1815</td>
<td>1657</td>
<td>1334</td>
<td>1178</td>
<td></td>
</tr>
<tr>
<td>Dolphin</td>
<td>3096</td>
<td>-</td>
<td>4096</td>
<td>2002</td>
<td></td>
</tr>
<tr>
<td>Helicopter</td>
<td>2749</td>
<td>-</td>
<td>3925</td>
<td>1702</td>
<td></td>
</tr>
<tr>
<td>bottom</td>
<td>1924</td>
<td>1517</td>
<td>1046</td>
<td>1149</td>
<td></td>
</tr>
<tr>
<td>Dolphin</td>
<td>2767</td>
<td>-</td>
<td>4016</td>
<td>2102</td>
<td></td>
</tr>
<tr>
<td>Helicopter</td>
<td>3064</td>
<td>-</td>
<td>3811</td>
<td>2103</td>
<td></td>
</tr>
</tbody>
</table>

Analysis suggests several reasons why DCSPNs can reach state-of-the-art results. First, deeper structures are created by simultaneously deriving 1-by-2 and 2-by-1 sum-pooling window sizes. Second, alternating every \( n \) layers with window sizes 2-by-2 and 2-by-2 serves as a regularization technique, since larger windows tend to yield shallower DAGs. Third, the known vanishing gradient problem in SPNs (Poon and Domingos 2011) seems to be alleviated by alternating window sizes as above, since this has the effect of creating branches of different lengths. This is similar to how shortcuts work in residual networks (He et al. 2016). Fourth, the vertical and horizontal windows of 1-by-2 and 2-by-1 leverages local structure in the image data in both directions. Fifth, accuracy was improved by using height-by-width instead of a 1-by-1 filter, that is, no sharing of parameters was more effective than sharing parameters. A similar finding was observed in (Sharir et al. 2018).

On the contrary, DCSPN completions admittedly look as though they are sometimes composed of random blocks of high frequency. To mitigate this, a low pass Gaussian filter can be applied as a smoothing post-processing step. This simple technique lowers the DCSPN MSE score in Olivetti left-completion in Table 1 from 455 to 401.

We also tried other common CNN techniques, such as average-pooling instead of sum-pooling layers, sharing parameters in convolutional layers, batch normalization, and dropout, but did not observe any significant improvement. It is noted that dropout was successfully applied in SPNs for image classification (Peharz et al. 2018).

**DCSPNs with Differentiable MPE**

Motivated by future work suggested in (Vergari et al. 2018), we propose an alternative method of training DCSPNs that is based on differentiable MPE. Whereas (5) minimizes the negative log-likelihood loss function, the new training method also considers the error between the given input and the MPE assignment (completion).

More formally, let \( D(x) \) be the value of a DCSPN \( D \) with input \( x \). Let \( Y \subseteq X \) and consider input \( y \), where the value of each variable in \( Y \) is 1, namely, the variables in \( Y \) are marginalized out. The MPE assignment after both forward and backward propagation in \( D \) is denoted by \( M(y) \). Then, the objective function for training \( D \) can be defined as:

\[
\min_{M} \max_{D} \mathbb{E}[\log D(x)] + \mathbb{E}[\log(1 - D(M(y)))] . \tag{6}
\]

Using (6) to train DCSPNs yields an MSE score of 651 for left-completion in the Olivetti dataset. This beats all competing scores in Table 1, except for DCSPNs trained using (5). Training DCSPNs using (6) is intriguing for more than simply a promising MSE score.

The training in (6) is similar to GANs training, which, in general, involves two networks: a generator \( G \) and a discriminator \( D \). The input to \( G \) is a random sample \( z \) drawn from
a simple distribution such as a normal distribution. $G$ upsamples $z$ and outputs $G(z)$. The discriminator $D$ receives $G(z)$ or dataset samples $x$ and downsamples to a classification output. Hence, training GANs involves minimizing $G$ and maximizing $D$:

$$\min_G \max_D \mathbb{E}[\log D(x)] + \mathbb{E}[\log(1 - D(G(z)))] \quad (7)$$

Figure 7 illustrates a general relationship between training DCSPNs with differentiable MPE in (6) and GANs training in (7). Observe that MPE inference computing $M(y)$ in (6) plays the role of the generator $G$ in GANs computing $G(z)$ in (7). However, while the GANs generator $G$ upsamples noise to form an image, DCSPNs use MPE inference to upsample the network value to form an image.

Similarly, forward inference in DCSPNs computing $D(x)$ in (6) assumes the part of the discriminator in GANs computing $D(x)$ in (7). Hence, DCSPN forward inference downsamples to a log-likelihood value measuring how likely the given input is from the sample distribution, whereas the GANs discriminator downsamples to a classification of whether or not the input data is real.

On the other hand, the GANs generator has its own learnable parameters, since the generator and discriminator are two different networks. However, a DCSPN is a single network, resulting in the sharing of parameters during learning between MPE inference and forward inference. Further investigation of DCSPNs with differentiable MPE and its relationship to GANs training remains as future work.

### Image Sampling

DCSPNs can sample images by modifying Algorithm 1. Instead of selecting a maximizing child, randomly select a child. More formally, replace lines 19 and 20 to instead assign values to $C$ by randomly sampling from a categorical distribution parameterized by the convolutional layer filter weights. Note that the forward pass is no longer needed.

The sampled images in Figure 8 are encouraging as variability is evident. Moreover, these results were obtained in an inaugural attempt at DCSPN image sampling involving a minor modification to Algorithm 1. Some generative models, including GANs, are known for low variability due to mode collapse (Metz et al. 2017; Salimans et al. 2016).

### Related Works

Here we comment on other vectorized approaches to SPNs. (Poon and Domingos 2011) and (Vergari, Di Mauro, and Esposito 2015) have discussed relationships between CNNs and SPNs. The interpretation of an SPN sum layer depends on the type of probabilistic inference being conducted. For marginal inference, an SPN sum layer corresponds to a convolutional layer in CNNs. For MPE, on the other hand, an SPN sum layer corresponds to a max-pooling layer in CNNs. We extend the literature by showing that an SPN product layer can correspond to a sum-pooling layer in CNNs.

(Sharir et al. 2018) introduced a tractable generative model, called convolutional arithmetic circuits (ConvACs). An arithmetic circuit (Darwiche 2003) is a deep learning model that has been shown by (Rooshenas and Lindow 2014) to be equivalent to SPNs. In particular, ConvACs include a representation layer (Sharir et al. 2018). We found this introduction useful in practice. However, similar to CSPNs, ConvACs may suffer from being shallow.

(Peharz et al. 2018) suggest an SPN learning method involving a discriminative and generative loss function. Although the generative part is also based on the log-likelihood similar to (6), the discriminative part is different. They consider the cross-entropy function of the SPN parameters, while we introduced to notion of a minmax game using differentiable MPE.

### Conclusion

Deep convolutional sum-product networks (DCSPNs) can form rich DAG structures of convolutional and sum-pooling layers, while still being valid SPNs. As a tractable generative model, DCSPNs can perform efficient probabilistic reasoning, including marginal inference and approximate MPE inference. On the other hand, as a CNN, DCSPNs can build deeper structures using both vertical and horizontal sum-pooling windows, which leverage local structure in the image data. Practical applications of DCSPNs include image completion and image sampling. DCSPNs are flexible in that they allow for an alternative learning method based on differentiable MPE. Relationships between this learning approach and learning in GANs are discussed.

Experimental results show that DCSPNs significantly outperform competitors and achieve several state-of-the-art results. For example, Table 1 reports the MSE scores for image left-completion in the benchmark Olivetti Face dataset. The DCSPN score of 455 is well below the next lowest score of 779 obtained in (Dennis and Ventura 2012). Applying a simple low pass filter as a post-processing step lowers the DCSPN MSE score down to 401.
References


