SimNets: A Generalization of Convolutional Networks

Nadav Cohen
The Hebrew University of Jerusalem
cohennadav@cs.huji.ac.il

Amnon Shashua
The Hebrew University of Jerusalem
shashua@cs.huji.ac.il

Abstract

We present a deep layered architecture that generalizes classical convolutional neural networks (ConvNets). The architecture, called SimNets, is driven by two operators, one being a similarity function whose family contains the convolution operator used in ConvNets, and the other is a new soft max-min-mean operator called MEX that realizes classical operators like ReLU and max pooling, but has additional capabilities that make SimNets a powerful generalization of ConvNets. Three interesting properties emerge from the architecture: (i) the basic input to hidden layer to output machinery contains as special cases kernel machines with the Exponential and Generalized Gaussian kernels, the output units being "neurons in feature space" (ii) in its general form, the basic machinery has a higher abstraction level than kernel machines, and (iii) initializing networks using unsupervised learning is natural. Experiments demonstrate the capability of achieving state of the art accuracy with networks that are an order of magnitude smaller than comparable ConvNets.

1 Introduction

Convolutional neural networks (ConvNets) are attracting much attention largely due to their impressive empirical performance on large scale visual recognition tasks (c.f. [14, 29, 21, 24, 23]). The ConvNet architecture has the capacity to model large learning problems that include thousands of categories, while employing prior knowledge embedded into the architecture. The ConvNet capacity is controlled by varying the number of layers (depth), the size of each layer (breadth), and the size of the convolutional windows (which in turn are based on assumptions on local image statistics). The learning capacity is controlled using over-specified networks (networks that are larger than necessary in order to model the problem), followed by various forms of regularization techniques such as Dropout ([11]).

Despite their success in recent years, ConvNets still fall short of reaching the holy grail of human-level visual recognition performance. Scaling up to such performance levels could take more than merely dialing up network sizes while relying on prior knowledge to compensate for what we cannot learn. It may be worthwhile to challenge the basic ConvNet architecture, in order to obtain more compact networks for the same level of accuracy, or in other words, in order to increase the abstraction level of the basic network operations.

A few observations have motivated our work. The first is that the ConvNet architecture has not changed much since its early introduction in the 1980s ([15]) – there were some attempts to create other types of deep-layered architectures (cf. [19, 3, 18]), but these are not commonly used compared to ConvNets. Arguably, the empirical success that ConvNets have witnessed in recent years is mainly fueled by the ever-growing scale of available computing power and training data, with the contribution of algorithmic advances having secondary importance. Our second observation is that although there were attempts to use unsupervised learning to initialize ConvNets (c.f. [10, 2, 25]),

it has since been observed that these schemes have little to no advantage over carefully selected random initializations that do not use data at all (see for example [14, 29, 21]). We nevertheless believe that unsupervised initialization has an important role in scaling up the capacity of deep learning, and therefore find interest in deep architectures that give rise to natural initializations using unsupervised data. The third observation that motivated our work is that the ConvNet learning paradigm completely took over classification engines developed in the 1990s like Support Vector Machines (SVM) and kernel machines in general. These machine learning methods were well suited for "flat" architectures, and while attempts to apply them to deep layered architectures have been made ([4]), they did not keep up with the performance levels of the layered ConvNet architecture. It may be beneficial to develop a deep architecture that includes the body of work on kernel machines, but which still has the capacity to model large learning problems like the ConvNet architecture.

In this paper we introduce a new family of layered networks we call SimNets (similarity networks). The general idea is to "lift" the classical ConvNet architecture into something more general, a multilayer kernel network architecture, which carries several attractive features. First, the architecture bridges the decades-old ConvNets with the statistical learning machinery of the last decade or so. Second, it provides a higher level of abstraction than the convolutional and pooling layers of ConvNets, thus potentially providing more compact networks for the same level of accuracy. Third, the architecture is endowed with a natural initialization based on unlabeled data, which also has the potential for determining the number of channels in each layer based on variance analysis of patterns generated from the previous layer. In other words, the structure of a SimNet can potentially be determined automatically from (unlabeled) training data.

The SimNet architecture is based on two operators. The first is analogous to, and generalizes, the convolutional operator in ConvNets. The second, as special cases, plays the role of ReLU activation ([17]) and max pooling in ConvNets, but in addition, has capabilities that make SimNets much more than ConvNets. In a set of limited experiments on CIFAR-10 dataset ([13]) using a small number of layers, we achieved better or comparable performance to state of the art ConvNets with the same number of layers, and the specialized network studied in [5], using 1/9 and 1/5, respectively, of the number of learned parameters.

In the following sections, we introduce the two operators that the SimNet architecture comprises, and describe its special cases and properties. The experiments section is still preliminary but demonstrates the power of SimNets and their potential for high capacity learning. Additional experiments with deeper SimNets are underway, but those require extensive optimization and coding infrastructure in order to apply to large scale settings.

2 The SimNet architecture

The SimNet architecture consists of two operators – a "similarity" operator that generalizes the inner-product operator found in ConvNets, and a soft max-average-min operator called MEX that replaces the ConvNet ReLU activation ([17]) and max/average pooling layers, and allows additional capabilities as will be described below.

The similarity operator matches an input $\mathbf{x} \in \mathbb{R}^d$ with a template $\mathbf{z} \in \mathbb{R}^d$ and a weight vector $\mathbf{u} \in \mathbb{R}^d_+$ ($\mathbb{R}^d_+$ stands for the non-negative orthant of $\mathbb{R}^d$) through $\mathbf{u}^\top \phi(\mathbf{x}, \mathbf{z})$, where $\phi : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ is a similarity mapping. We will consider two forms of similarity mappings: the "linear" form $\phi(\mathbf{x}, \mathbf{z})_i = x_i z_i$, such that when setting $\mathbf{u} = \mathbf{1}$ we obtain an inner-product operator, and the "l_p" form $\phi(\mathbf{x}, \mathbf{z})_i = -|x_i - z_i|^p$ defined for $p > 0$.

In a layered architecture, a similarity layer is illustrated in fig. 1(a), where the similarity operator is applied to patches $\mathbf{x}_{ij} \in \mathbb{R}^{hwD}$ of width w, height h and depth D, with the indexes (i, j) describing the location of the patch within the layer's input. Given n templates $\mathbf{z}_1, ..., \mathbf{z}_n \in \mathbb{R}^{hwD}$ and weights $\mathbf{u}_1, ..., \mathbf{u}_n \in \mathbb{R}^{hwD}_+$, the layer's output at coordinates (i, j, l) becomes $\mathbf{u}_l^\top \phi_l(\mathbf{x}_{ij}, \mathbf{z}_l)$, where we use index l in ϕ_l to indicate that the similarity mapping may differ across channels. As customary with ConvNets, the width and height of the layer's output depends on the "stride" setting, which determines the step-size between input patches, e.g. with horizontal and vertical strides of s, the spatial dimensions of the output become $\lfloor (H - h)/s \rfloor + 1$ and $\lfloor (W - w)/s \rfloor + 1$. Note that using the linear-similarity mapping with unit weights($\mathbf{u}_l = \mathbf{1}$) reduces the similarity layer to a standard convolutional layer where $\mathbf{z}_l$ are the convolution kernels, whereas for l_p-similarity with fixed $p = 2$,

the output at coordinates (i, j, l) measures the weighted Euclidean (Mahalanobis) distance between the input patch $\mathbf{x}_{ij}$ and the template $\mathbf{z}_l$ with every pixel weighted through the entries of the weight vector $\mathbf{u}_l$. When using l_p-similarity in general, fixing the order p is not obligatory – the order can be learned based on training data, either globally or independently for each output channel.

We will see later on that, when setting unit weights, the (unweighted) linear and l_p similarity mappings correlate with kernel-SVM methods of statistical machine learning (through special cases of the SimNet architecture), and that the view of $\mathbf{z}_l$ as templates allows natural unsupervised initialization of networks using conventional statistical estimation methods.

The MEX operator, whose name stands for Maximum-minimum-Expectation Collapsing Smooth (with "CS" pronounced as "X"), is responsible for the role of activation functions, max or average pooling (both spatially and across channels), and weights necessary for classification. The operator is defined as follows:

$$\underset{i=1,\ldots,n}{MEX_{\xi}}\{c_i\} := \frac{1}{\xi}\log\left(\frac{1}{n}\sum_{i=1}^{n}\exp\{\xi\cdot c_i\}\right) \tag{1}$$

with the alternative notation $MEX_{\xi}\{c_i\}_{i=1}^{n}$ used interchangeably. The parameter $\xi \in \mathbb{R}$ spans a continuum between maximum, expectation (mean), and minimum:

$$\begin{aligned} MEX_{\xi}\{c_i\}_{i=1}^{n} &\underset{\xi\to+\infty}{\longrightarrow} \max\{c_i\}_{i=1}^{n} \\ MEX_{\xi}\{c_i\}_{i=1}^{n} &\underset{\xi\to 0}{\longrightarrow} \text{mean}\{c_i\}_{i=1}^{n} \\ MEX_{\xi}\{c_i\}_{i=1}^{n} &\underset{\xi\to-\infty}{\longrightarrow} \min\{c_i\}_{i=1}^{n} \end{aligned}$$

Moreover, for a given value of ξ, the operator is smooth and exhibits the "collapsing" property defined below:

$$\begin{aligned} &MEX_{\xi}\{MEX_{\xi}\{c_{ij}\}_{1\le j\le m}\}_{1\le i\le n} \\ &\quad = MEX_{\xi}\{c_{ij}\}_{1\le j\le m, 1\le i\le n} \end{aligned} \tag{2}$$

In a layered architecture, the MEX operator is used to define the MEX layer – see illustration in fig. 1(b). In the MEX layer, the input is divided into (possibly overlapping) blocks, each mapped to a single output element. The output value associated with the t'th input block is given by:

$$out(t) = MEX_{\xi_t}\left\{\{inp(s) + b_{ts}\}_{s\in block(t)}, c_t\right\}$$

where the index s runs though the input block, the offsets $b_{ts} \in \mathbb{R}$ serve various roles as will be described later, and $c_t \in \mathbb{R}$ are optional (may or may not be used). The MEX layer can realize two standard ConvNet layers – the ReLU activation and the max-pooling layer. To realize ReLU activation, one should set the input blocks to be single entries, have the output dimensions equal to the input dimensions, set $b_{ts} = 0, c_t = 0$, and let $\xi_t \to +\infty$, and as a result $out(t) = \max\{inp(t), 0\}$ as required (see fig. 1(c)). To realize a max-pooling layer, set the input blocks to cover a 2D area, set the depth of the output equal to that of the input, set $b_{ts} = 0$, omit c_t, and set $\xi_t \to +\infty$. As a result $out(i, j, l) = \max\{inp(i', j', l)\}_{(i',j')\in pool(i,j)}$ (see fig. 1(d)). Note that by setting $\xi_t \to 0$ one obtains an average-pooling layer, and moreover, the parameters ξ_t can be learned (optimized) as part of the training process, allowing additional flexibility.

To recap, the layers corresponding to the two operators of the SimNet architecture – similarity and MEX, can realize conventional ConvNets as follows:

- *Convolutional layer*: use similarity layer with linear form $\phi_l(\mathbf{x}, \mathbf{z})_i = x_i z_i$ and unit weights $\mathbf{u}_l = \mathbf{1}$.
- *ReLU activation*: use MEX layer with $b_{ts} = 0, c_t = 0, \xi_t \to +\infty$ and single-entry input blocks.
- *Max pooling layer*: use MEX layer with $b_{ts} = 0, \xi_t \to +\infty$, c_t omitted, and 2D input blocks.
- *Dense layer*: use similarity layer with the entire input as the only patch, linear form $\phi_l(\mathbf{x}, \mathbf{z})_i = x_i z_i$ and unit weights $\mathbf{u}_l = \mathbf{1}$.

Next, we make wider use of the two SimNet layers, taking us beyond classical ConvNets, exploring connections to classical statistical learning models with kernel machines.

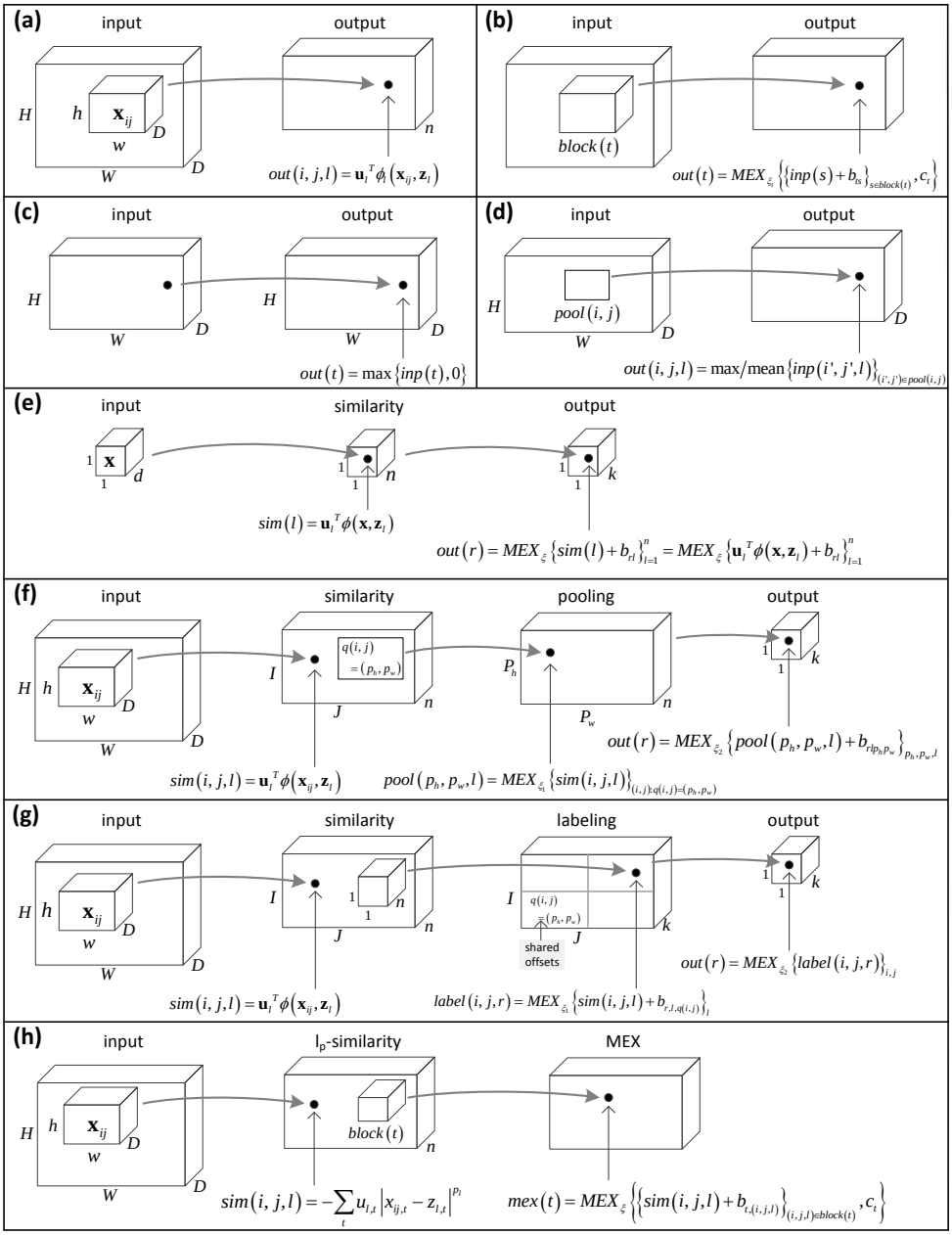

Figure 1: (a) SimNet similarity layer (b) SimNet MEX layer (c) ConvNet ReLU activation layer (d) ConvNet max/average pooling layer (e) SimNet MLP with multiple outputs (f) SimNet with locality, sharing and pooling (g) Patch-labeling SimNet (h) SimNet l_p-similarity layer followed by MEX layer

3 SimNets and kernel machines

So far, we set the architectural choices of SimNets to realize classical ConvNets, which form a rudimentary special case of the available possibilities. In particular, we did not make use of the l_p similarity, and of the offsets $\{b_{ts}\}$ in the MEX layer. In the following subsection, we consider a "multi-layer perceptron" (MLP) construction consisting of a single hidden layer in addition to input and output layers. In the subsection that follows, we will study the case where the input layer is processed by patches, the hidden layer involves locality and sharing, and a pooling operation follows the hidden layer – a structure prevalent in classical ConvNets.

3.1 MLP analogy: input $\rightarrow$ hidden layer $\rightarrow$ output

The Similarity and MEX operators give straightforward generalizations of the convolution and max/average pooling layers in ConvNets. As we now show, they create something of greater consequence when applied one after the other in succession. To make the point as succinctly as possible, consider a MLP construction consisting of d input nodes (making up the input vector $\mathbf{x} \in \mathbb{R}^d$), n hidden units, and a single output unit. The value $h(\mathbf{x})$ of the output unit is a result of a mapping $\mathbb{R}^d \rightarrow \mathbb{R}$, defined by the two SimNet operators applied in succession (n similarity operators with different templates and shared mapping ϕ, followed by MEX with offsets):

$$h(\mathbf{x}) = MEX_{\xi} \left\{ \mathbf{u}_l^{\top} \phi(\mathbf{x}, \mathbf{z}_l) + b_l \right\}_{l=1,\ldots,n} \tag{3}$$

A straightforward analogy to existing work is obtained by setting unit weights $\mathbf{u} = 1$, linear similarity $\phi(\mathbf{x}, \mathbf{z})_i = x_i z_i$ and $\xi \rightarrow \infty$, resulting in $h(\mathbf{x}) = \max \left\{ \mathbf{z}_l^{\top} \mathbf{x} + b_l \right\}_{l=1}^{n}$ – a maxout operation [8]. There were other attempts to generalize maxout, notably the recently proposed L_p unit [9], which is defined by $\left(\frac{1}{n} \sum_{l=1}^{n} |\mathbf{z}_l^{\top} \mathbf{x} + b_l|^p \right)^{1/p}$. When $p \rightarrow \infty$, this reduces to $\max_l \left\{ |\mathbf{z}_l^{\top} \mathbf{x} + b_l| \right\}$. The differences between this and the SimNet generalization of maxout are: (i) the L_p unit generalizes maximum of *absolute* values (rather than the values themselves), and (ii) the L_p unit tries to create a maxout in a single operation whereas the SimNet creates $h(\mathbf{x})$ over a succession of two operators – similarity followed by MEX.

Next, consider the case of fixed $\xi > 0$ and unweighted ($\mathbf{u}_l = \mathbf{1}$) linear similarity ($\mathbf{u}_l^{\top} \phi(\mathbf{x}, \mathbf{z}_l) = \mathbf{x}^{\top} \mathbf{z}_l$) or unweighted l_p similarity ($\mathbf{u}^{\top} \phi(\mathbf{x}, \mathbf{z}) = - \|\mathbf{x} - \mathbf{z}\|_p^p$) with fixed $0 < p \leq 2$. We will show below that in this case, the output $h(\mathbf{x})$ is the result of a non-linear monotone activation function applied to the inner-product between a mapping of the input $\mathbf{x}$ and a vector $\mathbf{w}$ in some high-dimensional feature space $\mathbb{R}^F$. More formally, we will show that $h(\mathbf{x}) = \sigma(\langle \mathbf{w}, \psi_{\phi}(\mathbf{x}) \rangle)$, where the mapping $\psi_{\phi} : \mathbb{R}^d \rightarrow \mathbb{R}^F$ depends on the choice of similarity mapping ϕ, σ is a non-linear monotone activation function, and $\mathbf{w} = \sum_{l=1}^{n} \alpha_l \psi_{\phi}(\mathbf{z}_l)$ for some $\alpha_1, ..., \alpha_n \in \mathbb{R}$. We thus conclude that the output unit is a "neuron" in the classical sense, but in a high-dimensional feature space. To prove this assertion, we notice that $h(\mathbf{x})$ can be expressed as follows:

$$\begin{aligned} h(\mathbf{x}) &= MEX_{\xi} \left\{ \mathbf{u}_l^{\top} \phi(\mathbf{x}, \mathbf{z}_l) + b_l \right\}_{l=1,\ldots,n} \\ &= \frac{1}{\xi} \ln \left(\frac{1}{n} \sum_{l=1}^{n} \alpha_l \cdot \exp \left\{ \xi \sum_{i=1}^{d} \phi(\mathbf{x}, \mathbf{z}_l)_i \right\} \right) \\ &= \sigma \left(\sum_{l=1}^{n} \alpha_l \cdot K_{\phi}(\mathbf{x}, \mathbf{z}_l) \right) \end{aligned} \tag{4}$$

where $\alpha_l := e^{\xi \cdot b_l}$ and σ is a non-linear monotone activation function given by $\sigma(t) = \frac{1}{\xi} \ln \left(\frac{t}{n} \right)$. We use the notation $K_{\phi}(\mathbf{x}, \mathbf{z}) := \exp \left\{ \xi \sum_{i=1}^{d} \phi(\mathbf{x}, \mathbf{z})_i \right\}$ to indicate that, under the similarity mappings considered, the function is a kernel on $\mathbb{R}^d$. In particular, for the linear and l_p similarities we have:

$$\begin{aligned} K_{lin}(\mathbf{x}, \mathbf{z}) &= \exp \left\{ \xi \cdot \mathbf{x}^{\top} \mathbf{z} \right\} \\ K_{l_p}(\mathbf{x}, \mathbf{z}) &= \exp \left\{ -\xi \|\mathbf{x} - \mathbf{z}_l\|_p^p \right\} \end{aligned}$$

As shown in [20], K_{lin} and K_{l_p} are kernels on $\mathbb{R}^d$ (note that for $p > 2$, the expression above for K_{l_p} is not a kernel). We refer to them as the "Exponential" kernel and the "Generalized Gaussian"

kernel[1] respectively. Since σ is monotonically increasing, $h(\mathbf{x})$ realizes a 2-class "reduced" kernel-SVM decision rule, with $\mathbf{z}_l$ being the (reduced) support-vectors[2] and $\alpha_l \geq 0$ being the coefficients associated with the support-vectors. Let $\psi_\phi : \mathbb{R}^d \to \mathbb{R}^F$ be a feature mapping corresponding to K_ϕ, i.e. $K_\phi(\mathbf{x}, \mathbf{z}) = \langle \psi_\phi(\mathbf{x}), \psi_\phi(\mathbf{z}) \rangle$. Eqn. 4 can now be expressed as $h(\mathbf{x}) = \sigma(\langle \mathbf{w}, \psi_\phi(\mathbf{x}) \rangle)$, where $\mathbf{w} := \sum_{l=1}^n \alpha_l \psi_\phi(\mathbf{z}_l)$. This shows that the output unit $h(\mathbf{x})$ is a "neuron is feature space", as stated above.

In the case of weighted ($\mathbf{u}_l$ are learned) l_p-similarity, the hypothesis space realized by the output unit $h(\mathbf{x})$ is no longer representable by a kernel-SVM. Moreover, the view of $h(\mathbf{x})$ as a neuron in feature space with learned vector $\mathbf{w}$ no longer applies. This is stated formally below (proof in app. A):

Theorem 1. *For any dimension $d \in \mathbb{N}$, and constants $c > 0$ and $p > 0$, there are no mappings $Z : \mathbb{R}^d \to \mathbb{R}^d$ and $U : \mathbb{R}^d \to \mathbb{R}^d_+$ and a kernel $K : (\mathbb{R}^d \times \mathbb{R}^d_+) \times (\mathbb{R}^d \times \mathbb{R}^d_+) \to \mathbb{R}^d \times \mathbb{R}^d_+$, such that for all $\mathbf{z}, \mathbf{x} \in \mathbb{R}^d$ and $\mathbf{u} \in \mathbb{R}^d_+$:*

$$K\left([Z(\mathbf{x}), U(\mathbf{x})], [\mathbf{z}, \mathbf{u}]\right) = \exp\left\{-c \sum_{i=1}^d u_i |x_i - z_i|^p\right\} \tag{5}$$

We now turn to consider a straightforward extension to the setup above, which includes k output units. The MLP will now consist of an input signal $\mathbf{x} \in \mathbb{R}^d$, a set of n hidden units defined by similarity functions over $\mathbf{x}$ (all based on the same similarity mapping ϕ), and a set of k output units defined by MEX operators (all having the same parameter ξ) with offsets b_{rl} where $l \in \{1, ..., n\}$ runs over the hidden units and $r \in \{1, ..., k\}$ is the index of the output unit. Fig. 1(e) illustrates this basic operation. If we consider the output nodes as predicting a label associated with the input $\mathbf{x}$, the chosen label being the index of the node with maximal activation, then running the two operators, similarity and MEX, one following the other, produces the classification rule below:

$$\hat{y}(\mathbf{x}) = \underset{r=1,...,k}{\operatorname{argmax}} MEX_\xi\{\mathbf{u}_l^\top \phi(\mathbf{x}, \mathbf{z}_l) + b_{rl}\}_{l=1}^n \tag{6}$$

This classification measures weighted similarities to n templates, with class-dependent offsets. The role of the MEX operators is to combine the weighted similarities (with offsets) of the input $\mathbf{x}$ to the templates. For example, when $\xi \to +\infty$, the classification rule is attracted to the most similar template where offsets assign relevancy of templates to classes. Let $h_r(\mathbf{x})$, $r = 1, ..., k$, be the value of output unit r when the MLP is fed with input $\mathbf{x}$. Following the lines of the derivation carried out for the single-output MLP, when working with unweighted linear similarity or unweighted l_p similarity with $0 < p \leq 2$, it holds that:

$$h_r(\mathbf{x}) = \sigma\left(\langle \mathbf{w}_r, \psi_\phi(\mathbf{x}) \rangle\right)$$

where $\mathbf{w}_r := \sum_{l=1}^n \alpha_{rl} \psi_\phi(\mathbf{z}_l)$ and $\alpha_{rl} := e^{\xi \cdot b_{rl}}$. Moreover, the decision rule in eqn. 6 can be expressed as:

$$\begin{aligned}
\hat{y}(\mathbf{x}) &= \underset{r \in \{1,...,k\}}{\operatorname{argmax}} \; h_r(\mathbf{x}) \\
&= \underset{r \in \{1,...,k\}}{\operatorname{argmax}} \; \langle \mathbf{w}_r, \psi_\phi(\mathbf{x}) \rangle \\
&= \underset{r \in \{1,...,k\}}{\operatorname{argmax}} \sum_{l=1}^n \alpha_{rl} K_\phi(\mathbf{x}, \mathbf{z}_l)
\end{aligned}$$

where K_ϕ is a kernel (Exponential or Generalized Gaussian) on $\mathbb{R}^d$. This classification realizes the hypothesis space of a reduced multiclass kernel-SVM[3].

[1] When $p = 2$, this reduces to the well-known Gaussian (radial basis function) kernel. When $p = 1$, it reduces to the Laplacian kernel.

[2] We use the term "reduced" to refer to the case where the number of support-vectors is predetermined and they are not constrained to lie in the training set. This setting was studied in [27] in the context of binary (2-class) classification. The extension to multiclass ([6]) is straightforward.

[3] Note that the coefficients α_{rl} are positive, whereas in classical multiclass SVM they may be any real numbers. This however does not limit generality, as we can always add a common offset to all coefficients after SVM training is complete.

To summarize so far, we have shown that with linear similarity, the "MLP" construction consisting of input $\rightarrow$ hidden layer $\rightarrow$ output, gives rise to the hypothesis space of a (reduced) SVM with the Exponential kernel. Replacing the linear similarity with unweighted l_p similarity having fixed order p, gives rise to a kernel-SVM if and only if $p \leq 2$, in which case the underlying kernel is the Generalized Gaussian kernel (the special cases of Gaussian and Laplacian kernels are obtained for $p = 2$ and $p = 1$ respectively). With these similarities that give rise to kernel machines, a unit generated by similarity operators followed by MEX with offsets is a "neuron in feature space". Finally, with weighted ($\mathbf{u}_l$ are learned) l_p similarity, the framework is no longer representable by a kernel-SVM.

To obtain a sense of the network's abstraction level, i.e. its ability to capture concept (category) distributions in input space, consider the classification rule in eqn. 6 in the case $\xi \rightarrow +\infty$:

$$\hat{y}(x) = \underset{r=1,...,k}{\operatorname{argmax}} \max_{l=1,...,n} \{\mathbf{u}_l^\top \phi(\mathbf{x}, \mathbf{z}_l) + b_{rl}\}$$

For any $r \in \{1, ..., k\}$, denote by A_r the decision region in input space that corresponds to class r, i.e. $A_r := \{\mathbf{x} \in \mathbb{R}^d : \hat{y}(\mathbf{x}) = r\}$. To understand the shape of A_r, we make the following definitions:

$$A_{r,l}^{r',l'} := \tag{7}$$
$$\{\mathbf{x} \in \mathbb{R}^d : \mathbf{u}_l^\top \phi(\mathbf{x}, \mathbf{z}_l) + b_{rl} \geq \mathbf{u}_{l'}^\top \phi(\mathbf{x}, \mathbf{z}_{l'}) + b_{r'l'}\}$$
$$A_{r,l} := \bigcap_{(r',l') \neq (r,l)} A_{r,l}^{r',l'}$$

where the class index r' ranges over $\{1, ..., k\}$, and the template indexes l, l' range over $\{1, ..., n\}$. One can readily see that up to boundary conditions:

$$A_r = \bigcup_{l \in \{1,...,n\}} A_{r,l}$$

Consider first the setting of linear similarity ($\phi(\mathbf{x}, \mathbf{z})_i = x_i z_i$). In this case $A_{r,l}^{r',l'}$ are half-spaces and $A_{r,l}$ are intersections of half-spaces (polytopes). The decision region A_r is thus a union of n polytopes. As we now show, this is the same type of decision regions as obtained with unweighted l_2 similarity ($\mathbf{u}_l = \mathbf{1}$, $\phi(\mathbf{x}, \mathbf{z})_i = -|x_i - z_i|^2$). Indeed, in this case the term $\|\mathbf{x}\|_2^2$ in both sides of the inequality defining $A_{r,l}^{r',l'}$ cancels-out, and we obtain again a half-space. This in turn implies that as before, $A_{r,l}$ are polytopes and A_r is a union of polytopes. We conclude that with the MLP structure of: input $\rightarrow$ hidden layer $\rightarrow$ output units, the setting that realizes a Gaussian kernel machine (unweighted l_2 similarity), is qualitatively equivalent to the "ConvNet" (linear similarity) setting that realizes an Exponential kernel machine. The difference in kernels does not account for any material difference in the network's hypothesis space, i.e. its abstraction level.

Remaining with l_2 similarity, we now consider the weighted setting, i.e. the setting in which $\mathbf{u}_l$ are not fixed. Thm. 1 tells us that in this case the hypothesis space is no longer governed by kernel-SVM. From the decision region point-of-view, it is not difficult to see that in this case $A_{r,l}^{r',l'}$ is no longer a half-space, but a region defined by a second-order hyper-surface. This implies that the set $A_{r,l}$ is no longer a polytope, and in particular is not necessarily convex. The possible shapes that the decision region A_r can take are thus enriched. We conclude that unlike in the case of unweighted l_2 similarity, the setting of weighted l_2 similarity is characterized by an abstraction level higher than that induced by linear similarity (convolutional operator).

In the general setting of l_p similarity, the sets $A_{r,l}^{r',l'}$ are more complex, and may be governed by non-convex non-smooth separating hyper-surfaces. The full analysis is outside the scope of this paper, but an informal illustration of how the space is divided for $p = 1$ and $d = 2$ ($\mathbb{R}^d$ is the 2D plane) is given in fig. 2. Under this specific setup, the 2D plane is divided into two by a piece-wise linear separating boundary. The unweighted case (uniform weights) is shown in fig. 2(a). In this case the space is divided equally (up to a shift caused by the offsets $b_{rl}, b_{r'l'}$) based on the l_1 (Manhattan) distance metric. Adding weights deforms the boundary line, where the higher the weights associated with a template ($\mathbf{z}_l$ or $\mathbf{z}_{l'}$) are, the less space is allocated to that template. For example, in fig. 2(d) the weights associated with the template $z_{l'}$ are uniformly high, thereby creating a small aperture in

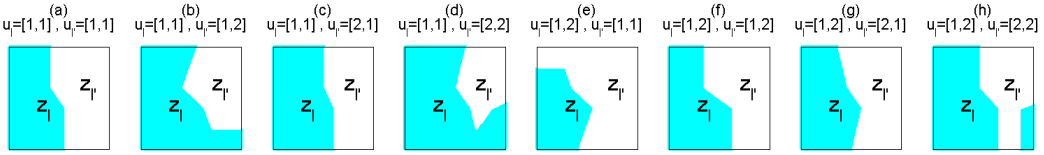

Figure 2: Illustration of $A_{r,l}^{r',l'}$ (defined in eqn. 7) in the setting of l_1 similarity and $d = 2$ ($\mathbb{R}^d$ – the 2D plane). Each panel shows the location of the templates $\mathbf{z}_l$ and $\mathbf{z}_{l'}$, and on the top the values of the corresponding similarity weights. (a): Unweighted setting (uniform weights). The 2D plane is divided equally between the two templates (up to a shift resulting from the offsets $b_{r,l}, b_{r',l'}$. (d): Here $\mathbf{z}_{l'}$ is associated with high weights, thus the portion of the plane allocated to this template is "shrinked".

the 2D plane around that template. Given that $A_{r,l}^{r',l'}$ is highly non-convex in the weighted setting, we expect weighted l_1 similarity to provide a higher abstraction level than that of linear similarity (convolutional operator).

3.2 A basic 3-layer SimNet with locality, sharing and pooling

Next, we analyze a 3-layer SimNet with locality, sharing and pooling. The network's input is processed by patches (locality), with the same templates and weights applied to all patches (sharing), thereby creating a stack of feature maps (channels) – one for each template. Spatial regions of each feature map are then pooled together to reduce dimensionality, and finally, a classification output layer predicts the label of the input. We will show that such a network, consisting of input $\rightarrow$ feature maps $\rightarrow$ pooling $\rightarrow$ output, also corresponds to a kernel-SVM, with the kernels designed for a "patch-based" representation of the input signal.

Locality, sharing and pooling are realized in the conventional manner. Namely, the input is divided into (possibly overlapping) patches $\mathbf{x}_{ij} \in \mathbb{R}^d$ where $d = h \cdot w \cdot D$, with h, w being the height and width of the patches. A similarity layer as illustrated in fig. 1(a), but with the same similarity mapping ϕ for all channels, matches the patch $\mathbf{x}_{ij}$ with the template $\mathbf{z}_l \in \mathbb{R}^d$ (which is now a template representing a local patch in the layer's input) using the weights $\mathbf{u}_l \in \mathbb{R}_+^d$, and the resulting value $\mathbf{u}_l^\top \phi(\mathbf{x}_{ij}, \mathbf{z}_l)$ is stored in coordinates (i, j, l) of the layer's output.

The mapping from the similarity layer to the k-node classification output is realized through two MEX layers. The first MEX layer implements a pooling layer as follows. Let $q(i, j) = (q_h(i), q_w(j))$ be a (contraction) mapping of the 2D coordinate system in the similarity layer to the 2D coordinate system in the pooling layer. Normally, a 2D coordinate in the pooling layer corresponds to a 2D window in the similarity layer. The value assigned to an element in the pooling layer is simply a MEX operation taken over the corresponding 2D window (in the respective channel $l \in \{1, ..., n\}$) in the similarity layer:

$$pool(p_h, p_w, l) = MEX_{\xi_1}\{sim(i, j, l)\}_{i,j:q(i,j)=(p_h,p_w)}$$

All MEX operators in the layer have the same parameter – ξ_1. When $\xi_1 \rightarrow +\infty$ for instance, we obtain max-pooling as implemented in conventional ConvNets.

The second MEX layer implements a dense mapping from the pooling layer to the k output nodes, which includes offsets. The value of the r'th output node is given by:

$$out(r) = MEX_{\xi_2}\{pool(p_h, p_w, l) + b_{rlp_hp_w}\}_{p_h,p_w,l}$$

where (p_h, p_w) runs over the 2D coordinates of the pooling layer and l runs over the pooling layer's channels (which correspond to templates). Note that here too all MEX operators have the same parameter – ξ_2. The offsets $b_{rlp_hp_w}$ depend on the output node r and on the 3D coordinates of the pooling layer (p_h, p_w, l), i.e. for every output node there is an offset for each coordinate in the pooling layer.

The SimNet we obtain is illustrated in fig. 1(f). It is a basic similarity-pooling-output network that employs locality and sharing, as conventional ConvNets do. The point to be made next, is that in the special case where $\mathbf{u}_l = \mathbf{1}$, ξ_1, ξ_2 are fixed to a constant $\xi > 0$, and ϕ is set to linear form or l_p form with fixed $p \leq 2$, the classification resulting from this network is a kernel-SVM, where the

kernel is designed for a "patch-based" representation of the input signal. First, by concatenating the three steps – similarity, pooling and output, and assuming that $\xi_1 = \xi_2 = \xi > 0$, the decision rule associated with the network becomes:

$$\hat{y}(inp) = \underset{r=1,\ldots,k}{\text{argmax}} \underset{i,j,l}{MEX_{\xi}} \left\{ \mathbf{u}_l^{\top} \phi(\mathbf{x}_{ij}, \mathbf{z}_l) + b_{r,l,q(i,j)} \right\} \tag{8}$$

This follows from the identities below:

$$\begin{aligned} &\underset{p_h,p_w,l}{MEX_{\xi}} \left\{ \underset{i,j:q(i,j)=(p_h,p_w)}{MEX_{\xi}} \left\{ \mathbf{u}_l^{\top} \phi(\mathbf{x}_{ij}, \mathbf{z}_l) \right\} + b_{rlp_hp_w} \right\} \\ &= \underset{p_h,\ p_w,\ l,\ i,j:q(i,j)=(p_h,p_w)}{MEX_{\xi}} \left\{ \mathbf{u}_l^{\top} \phi(\mathbf{x}_{ij}, \mathbf{z}_l) + b_{rlp_hp_w} \right\} \\ &= \underset{i,j,l}{MEX_{\xi}} \left\{ \mathbf{u}_l^{\top} \phi(\mathbf{x}_{ij}, \mathbf{z}_l) + b_{r,l,q(i,j)} \right\} \end{aligned}$$

where for the first equality, we used the collapsing property of the MEX operator described in eqn. 2. The classification described in eqn. 8 is similar to that described in eqn. 6, but has two important distinctions: *(i)* the templates $\mathbf{z}_l$ are local and similarity is applied locally (hence the "locality" and "sharing"), and *(ii)* the offsets are region-based (hence the "pooling"), i.e. each collection of input patches $\mathbf{x}_{ij}$ ascribed to the same pool is associated with a single set of offsets (per-class and per-template). To see the kernel structure associated with this classification, we perform the following manipulations to the rule given in eqn. 8:

$$\begin{aligned} &\hat{y}(inp) = \\ &\underset{r=1,\ldots,k}{\text{argmax}} \underset{i,j,l}{MEX_{\xi}} \left\{ \mathbf{u}_l^{\top} \phi(\mathbf{x}_{ij}, \mathbf{z}_l) + b_{r,l,q(i,j)} \right\} = \\ &\underset{r=1,\ldots,k}{\text{argmax}} \sum_{i,j,l} e^{\xi \cdot (\mathbf{u}_l^{\top} \phi(\mathbf{x}_{ij}, \mathbf{z}_l) + b_{r,l,q(i,j)})} = \\ &\underset{r=1,\ldots,k}{\text{argmax}} \sum_{i,j,l} \underbrace{e^{\xi \cdot b_{r,l,q(i,j)}}}_{:=\alpha_{r,l,q(i,j)}} \cdot e^{\xi \cdot \mathbf{u}_l^{\top} \phi(\mathbf{x}_{ij}, \mathbf{z}_l)} = \\ &\underset{r=1,\ldots,k}{\text{argmax}} \sum_{p_h,p_w,l} \alpha_{rlp_hp_w} \sum_{i,j:q(i,j)=(p_h,p_w)} e^{\xi \cdot \mathbf{u}_l^{\top} \phi(\mathbf{x}_{ij}, \mathbf{z}_l)} \end{aligned} \tag{9}$$

Setting $\mathbf{u}_l = \mathbf{1}$, and referring to subsec. 3.1, we denote $K_{\phi}(\mathbf{x}_{ij}, \mathbf{z}_l) := \exp\{\xi \cdot \mathbf{1}^{\top} \phi(\mathbf{x}_{ij}, \mathbf{z}_l)\}$, emphasizing that this function is a kernel for the similarity mappings we consider (Exponential kernel for linear similarity, Generalized Gaussian kernel for l_p similarity with fixed $p \leq 2$). Eqn. 9 then becomes:

$$\begin{aligned} &\hat{y}(inp) = \\ &\underset{r=1,\ldots,k}{\text{argmax}} \sum_{p_h,p_w,l} \alpha_{rlp_hp_w} \sum_{i,j:q(i,j)=(p_h,p_w)} K_{\phi}(\mathbf{x}_{ij}, \mathbf{z}_l) \\ &= \underset{r=1,\ldots,k}{\text{argmax}} \textstyle\sum_{p_h,p_w,l} \alpha_{rlp_hp_w} \mathbf{K}_{\phi}(X, Z_{lp_hp_w}) \end{aligned} \tag{10}$$

where X contains the concatenation of all the input patches $\mathbf{x}_{ij}$, and $\mathbf{Z}_{lp_hp_w}$ is a structure containing copies of $\mathbf{z}_l$ in locations corresponding to the pool index (p_h, p_w) – the details, including definition of $\mathbf{K}_{\phi}$ and proof that it is indeed a kernel, are given in app. B.

4 Other SimNet settings – global average pooling

In subsec. 3.2 we introduced the SimNet basic building chain of the form: input $\rightarrow$ similarity $\rightarrow$ pooling $\rightarrow$ output, whose structure follows the line of classical ConvNets. We noted that the basic building chain realizes a kernel-SVM hypothesis space, where the templates in the similarity layer correspond to the (reduced) support-vectors, and the offsets in the last MEX layer (from pooling to output) are related to the SVM coefficients. The SVM hypothesis space is realized when the similarity operator is set to linear form or l_p form with fixed $p \leq 2$, and is unweighted ($\mathbf{u}_l = \mathbf{1}$). Using weighted l_p similarity (weights are not applicable to linear similarity) has the potential of providing a richer hypothesis space than kernel-SVM (at the expense of doubling the number of

parameters in the similarity layer). Indeed, experiments we conducted (reported in sec. 6) validate the power of l_p similarity weighting, showing that it matters more than merely the added number of parameters to the model.

In this section, we introduce another SimNet building chain with two MEX layers, designed in such a way that when the MEX parameters are equal, the chain collapses into the one presented above (decision rule in eqn. 8), but when the MEX parameters are determined separately – either learned using training data or set manually, the SimNet chain allows for new possibilities (without additional parameters). For example, setting the MEX parameter of the first layer to 1 and that of the second layer to 0 gave rise to the best experimental performance we encountered.

The idea is to switch the roles of the two MEX layers – rather than having the first play the role of pooling and the second the role of classification (using the offsets b_{rl}), we start with a MEX layer with offsets and finish with a MEX for pooling. The interpretation of such a structure is that each input patch $\mathbf{x}_{ij}$ undergoes classification in the first MEX layer. The second MEX layer performs a majority voting over all the patch-based classification results to form a final classification decision. This approach follows the line of the "global average pooling" structure recently suggested in the context of ConvNets, which has been shown to outperform the traditional "dense classification" paradigm ([16, 23]). To enforce spatial consistency in the labeling characteristics of patches, we constrain the first MEX layer's offsets to be uniform inside predetermined spatial regions. The resulting SimNet, which we refer to as a "patch labeling" network, is illustrated in fig. 1(g) (note that all channels in the similarity layer share the same similarity mapping, and that both MEX layers have global parameters ξ_1, ξ_2). Its classification rule takes the following form:

$$\hat{y}(inp) = \underset{r=1,\ldots,k}{\operatorname{argmax}}\, out(r)$$

with:

$$out(r) = \underset{i,j}{MEX_{\xi_2}}\{\underset{l}{MEX_{\xi_1}}\{\mathbf{u}_l^\top \phi(\mathbf{x}_{ij}, \mathbf{z}_l) + b_{r,l,q(i,j)}\}\}$$

The variables which can be learned here are the offsets $b_{rlp_hp_w} \in \mathbb{R}$ (with r ranging over the classes, l over the templates and (p_h, p_w) over the regions in which offsets are shared), the templates $\mathbf{z}_l \in \mathbb{R}^{hwD}$, the similarity weights $\mathbf{u}_l \in \mathbb{R}_+^{hwD}$, the order $p > 0$ in case l_p similarity is chosen, and the MEX parameters $\xi_1, \xi_2 \in \mathbb{R}$. Assume we constrain the MEX parameters to be equal: $\xi_1 = \xi_2 = \xi$. The MEX collapsing property (eqn. 2) then applies, and the classification rule becomes:

$$\hat{y}(inp) = \underset{r=1,\ldots,k}{\operatorname{argmax}}\, MEX_{\xi}\{\mathbf{u}_l^\top \phi(\mathbf{x}_{ij}, \mathbf{z}_l) + b_{r,l,q(i,j)}\}_{i,j,l}$$

which is identical to the decision rule in eqn. 8. However, there is no reason to have the MEX parameters equal to each other. We can estimate their value during training, or set them manually. For example, during our experimentation we found that the case of equal MEX parameters – $\xi_1 = \xi_2 = \xi$, is significantly outperformed by the setting $\xi_2 \to 0$, which corresponds to the following classification rule:

$$\hat{y}(inp) = \underset{r=1,\ldots,k}{\operatorname{argmax}} \textstyle\sum_{i,j} \underset{l\in\{1,\ldots,n\}}{MEX_{\xi_1}} \left\{\mathbf{u}_l^\top \phi(\mathbf{x}_{ij}, \mathbf{z}_l) + b_{r,l,q(i,j)}\right\}$$

5 Initializing parameters using unsupervised learning

For classical ConvNets, various schemes of initializing a network based on unlabeled data (unsupervised initialization) have been proposed (c.f. [10, 2, 25]). Over time, however, these were taken over by carefully selected random initializations that do not use data at all (see for example [14, 29, 21]). No initialization scheme to-date is sufficient on its own for overcoming the hardness of training. Indeed, successful training of ConvNets typically requires designing an over-specified network (i.e. a network that is much larger than necessary in order to represent the true hypothesis space). While the latter has been shown to produce good training results, it bares a computational price, and also aggravates the problem of overfitting. The enhanced susceptibility to overfitting has led to various

regularization techniques and heuristics (Dropout ([11]) being the most prominent), which nowadays form an art that one must master in order to properly train ConvNets. In this section, we discuss a natural unsupervised initialization scheme for SimNets, which is based on statistical estimation. Such a scheme may provide a more effective local minima in the process of training a SimNet, thereby reducing the need for over-specification, supporting smaller networks that are more efficient computationally, and less prone to overfit. Experiments we conducted (reported in sec. 6) validate this conjecture, showing that the SimNet unsupervised initialization scheme indeed improves performance over random initializations, especially in the case of small networks.

Recall from sec. 2 that measuring weighted similarities to templates forms the similarity layer – a basic building block of the SimNet architecture (see fig. 1(a)). Focusing on the case of l_p similarity mappings ($\phi(\mathbf{x},\mathbf{z})_i = -|x_i - z_i|^p$), we show how the application of statistical estimation methods to unlabeled training data can produce initialization values for the layer's templates $\mathbf{z}_1, ..., \mathbf{z}_n$, weights $\mathbf{u}_1, ..., \mathbf{u}_n$ and orders $p_1, ..., p_n$. Consider a probability distribution over $\mathbb{R}^d$ defined by a mixture of n Generalized Gaussian distributions, each having independent coordinates with a shared shape parameter and separate scales and means:

$$P(\mathbf{x}) = \sum_{l=1}^{n} \lambda_l \prod_{i=1}^{d} \frac{\beta_l}{2\alpha_{l,i}\Gamma(1/\beta_l)} \exp\left\{-\left(\frac{|x_i - \mu_{l,i}|}{\alpha_{l,i}}\right)^{\beta_l}\right\}$$

In the above, λ_l stands for the prior probability of component l ($\lambda_l \geq 0, \sum_l \lambda_l = 1$), $\beta_l > 0$ stands for the shape parameter of all coordinates in component l, $\alpha_{l,i} > 0$ stands for the scale of coordinate i in component l, $\mu_{l,i} \in \mathbb{R}$ stands for the mean of coordinate i in component l, and Γ is the Gamma function, defined by $\Gamma(s) = \int_0^\infty e^{-t} t^{s-1} dt$. The log-probability that a vector drawn from this distribution is equal to $\mathbf{x}$ and originated from component l is:

$$\log P(\mathbf{x} \wedge \text{component } l) = -\sum_{i=1}^{d} \alpha_{l,i}^{-\beta_l} |x_i - \mu_{l,i}|^{\beta_l} + c_l$$

where $c_l := \log\left\{\lambda_l \prod_{i=1}^{d} \frac{\beta_l}{2\alpha_{l,i}\Gamma(1/\beta_l)}\right\}$ is a constant that depends on the component only (not on $\mathbf{x}$). Setting the layer's templates by $z_{l.i} = \mu_{l,i}$, its weights by $u_{l,i} = \alpha_{l,i}^{-\beta_l}$ and its l_p orders by $p_l = \beta_l$, would give:

$$\mathbf{u}_l^\top \phi_l(\mathbf{x}_{ij}, \mathbf{z}_l) = \log P(\mathbf{x} \wedge \text{component } l) - c_l$$

This implies that if we assume input patches follow a Generalized Gaussian mixture as described, initializing the similarity layer's templates, weights and orders as above would result in channel l of the layer's output holding, up to a constant, the probabilistic heat map of component l and the patches. This observation suggests estimating the parameters (shapes β_l, scales $\alpha_{l,i}$ and means $\mu_{l,i}$) of the Generalized Gaussian mixture using unlabeled input patches (via standard statistical estimation methods, such as that presented in [1]), and initializing the similarity layer accordingly.

Consider now the case where the initialized l_p similarity layer is followed by a MEX layer with learned offsets (see fig. 1(h), where for convenience, the linear index t is used to refer to elements of the MEX layer's 3D output array). We now assume that not only do input patches come from a mixture of Generalized Gaussian components as above, but also that each input patch location corresponds to a different mixture (priors) of these components. This makes sense, as certain templates that are likely to appear in the center of an image for example, may be less likely to appear on the top-left corner of the image, for example. Using our estimates of the global components obtained during the initialization of the similarity layer, we can estimate a mixture for a certain input patch location, by applying an estimation method to patches only from that location, with the component shapes, means and scales held fixed. We may then calculate offsets for the n elements of the similarity layer's output that correspond to that location, such that the probabilistic heat maps will take into account the location-dependent statistics, and will be precise (not up to a constant). For example, if there is a region in the input for which a certain template is very unlikely to appear, that template's heat map in the aforementioned region will be suppressed. The offsets we compute may serve for initialization of the MEX layer's offsets.

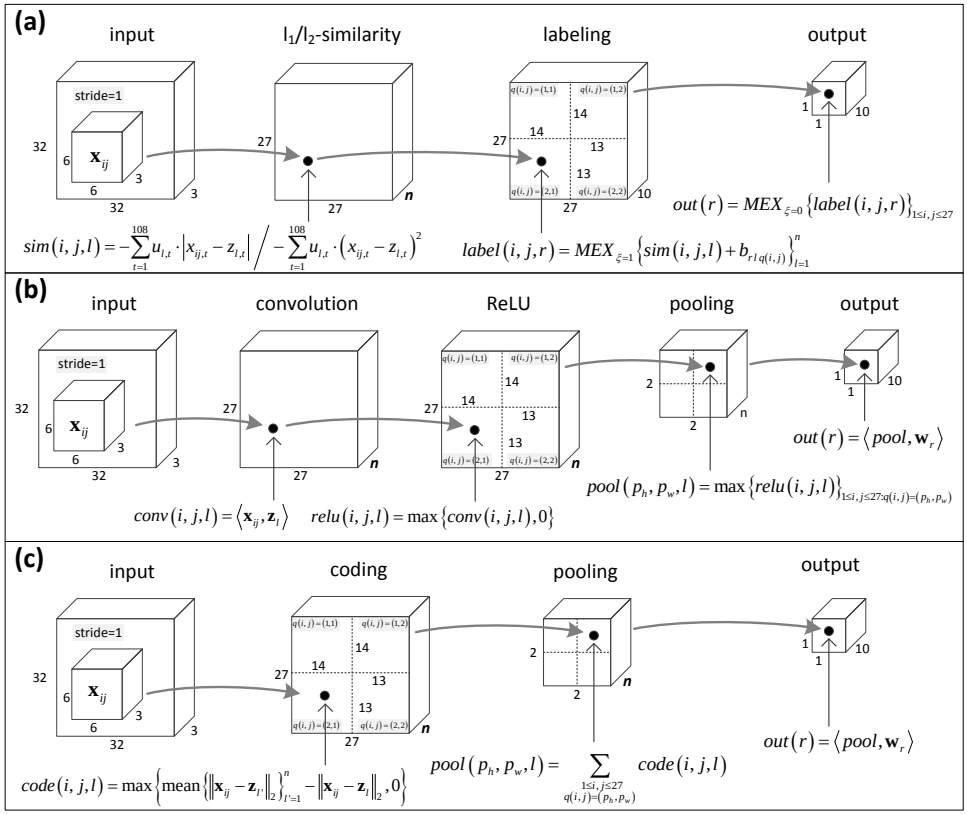

Figure 3: Networks evaluated on CIFAR-10. (a) Patch labeling SimNet (b) Comparable ConvNet (c) Comparable "single-layer" network studied in [5].

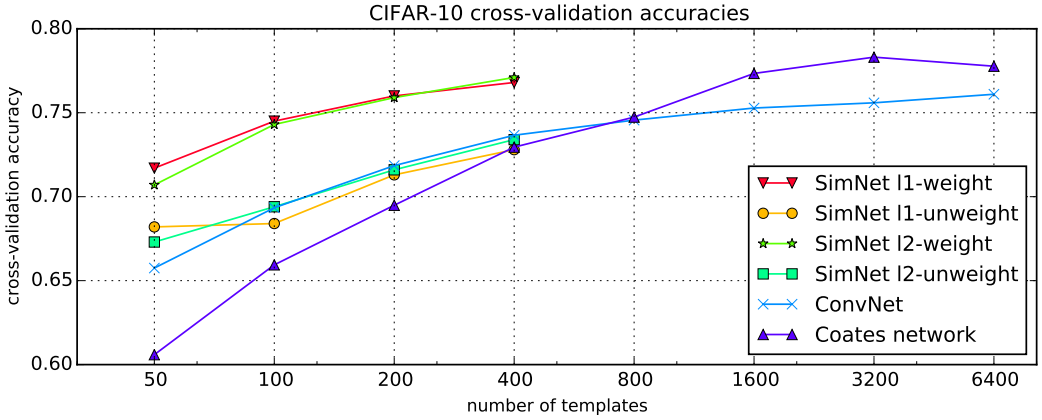

Figure 4: CIFAR-10 cross-validation accuracies plotted against the number of templates in the networks (denoted n in fig. 3). 'SimNet l1-weight' and 'SimNet l2-weight' correspond to the network structure illustrated in fig. 3(a), with l_1 and l_2 similarities respectively; 'SimNet l1-unweight' and 'SimNet l2-unweight' correspond to the same networks, but with the weight vectors $\mathbf{u}_l$ held fixed during training; 'ConvNet' corresponds to the network illustrated in fig. 3(b); 'Coates network' corresponds to the network illustrated in fig. 3(c).

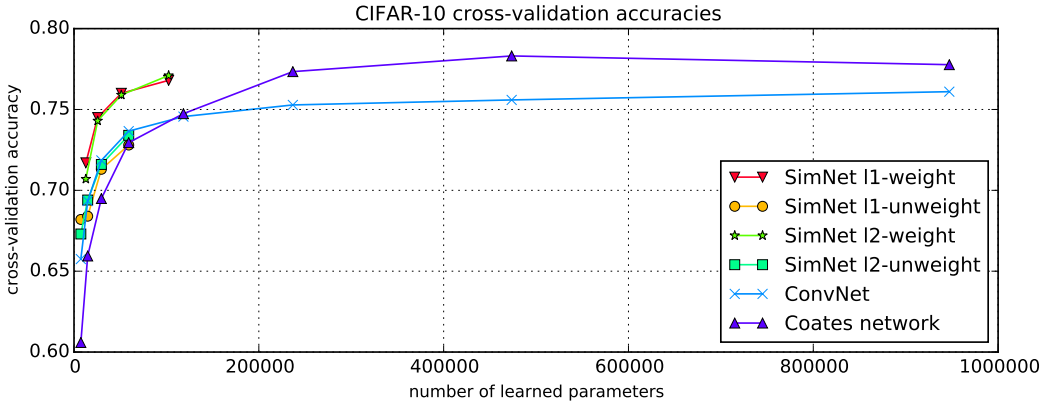

Figure 5: CIFAR-10 cross-validation accuracies plotted against the number of learned parameters in the networks. This is merely a different display of the results given in fig. 4. Notice how with weighted similarities, the SimNet reaches approximately the same level of performance as the competition, using much smaller networks.

6 Experiments

We implemented the "patch labeling" SimNet discussed in sec. 4, and experimented with the specific architectural settings illustrated in fig. 3(a). The network consists of a l_p-similarity layer with p fixed at 1 or 2, followed by two MEX layers. Implementation and evaluation of deeper SimNets is currently under work, and will be reported at a later time. For the experiments reported here, we used the CIFAR-10 dataset ([13]), which consists of $60,000$ color images ($50,000$ for training and $10,000$ for testing) of size 32×32 partitioned into 10 classes, with $6,000$ images per class ($5,000$ for training, $1,000$ for testing). The network's input is an RGB image ($32 \times 32 \times 3$ array), processed by patches of size $6 \times 6 \times 3$ with a single-pixel stride between them. For a given number of templates in the similarity layer (denoted by n), the SimNet's learned parameters are the templates $\mathbf{z}_1, ..., \mathbf{z}_n \in \mathbb{R}^{108}$, the similarity weights $\mathbf{u}_1, ..., \mathbf{u}_n \in \mathbb{R}^{108}_+$, and the MEX offsets $b_{rlp_hp_w} \in \mathbb{R}$ with $r = 1, ..., 10$, $l = 1, ..., n$ and $p_w, p_h = 1, 2$. We used statistical estimation as described in sec. 5 to initialize the templates $\mathbf{z}_l$ and the similarity weights $\mathbf{u}_l$ (initialization was based on training images, without making use of their labels). The network was then trained by minimizing a softmax loss with stochastic gradient descent (SGD) that includes momentum and acceleration ([22]). For SGD, we used a batch size of 64, a momentum of 0.9, and a learning rate of 0.01 decreased by a factor of 10 after 50 epochs, running 100 epochs in total. The weight decay for the templates was set to zero, and those for the similarity weights and offsets were set equal to each other, their value chosen via cross-validation.

We compared the SimNet to instances of two learning architectures. The first is a ConvNet with a single convolutional layer followed by a pooling layer followed by an output layer (see illustration in fig. 3(b)). The purpose of this comparison is to evaluate the SimNet against an analogous ConvNet, measuring the network sizes (number of learned parameters) required to reach given accuracies. A successful outcome here would be if the SimNet reached the same (or higher) level of performance as the ConvNet, with considerably smaller network size. The second comparison we held was against the "single-layer" network studied by Coates et al. ([5]), which has the same depth as the evaluated SimNet, and whose performance on CIFAR-10 is one of the best reported for networks of such depth (absolute state of the art in 2011). In [5], a number of unsupervised learning methods were devised for "coding" the input image. The coding methods were based on "templates", such that each template corresponded to a single feature map. The feature maps were passed on to a sum-pooling operator, and from there a linear SVM was learned using supervised data. Many coding methods were experimented on, and the one that produced the best results, referred to as "triangle" coding, was a "soft" Euclidean measure applied to templates learned via k-means. This coding method, along with the other architectural settings that produced the best results, are illustrated in fig. 3(c). Finally, we question the importance of the unsupervised initialization scheme described in sec. 5, by training the evaluated SimNet with random initialization (as customary with ConvNets), and examining the effect on the cross-validation accuracies.

The results reported below show that the evaluated SimNet achieves performance comparable to that of the ConvNet and the network of Coates et al., with only a fraction of the number of learned parameters. The unsupervised initialization scheme indeed boosts performance, and can be viewed as one of the drivers behind the SimNet's superiority. We are currently working on the optimization of our code (including GPU acceleration), to enable evaluation of larger and deeper SimNets on more meaningful benchmarks, comparing against deep state of the art ConvNets.

6.1 Benchmarking against the ConvNet

The ConvNet was implemented using Caffe toolbox ([12]), with random initialization and SGD training. We used a batch size of 100, momentum of 0.95, and learning rate of 10^{-4} decreased by a factor of 10 every 45 epochs, running 150 epochs in total. The global weight decay and the dense layer's DropOut rate were chosen via cross-validation. The ConvNet's input was an RGB image ($32 \times 32 \times 3$ array) normalized for brightness and contrast. For the SimNet we also added patch "whitening", in accordance with the suggestion of [5]. The positive effect of whitening for the SimNet (which has l_1/l_2 similarities) was verified experimentally, whereas for the ConvNet, we observed that whitening does not have a positive effect (complying with the observations of [5]).

Fig. 4 shows the cross-validation accuracies of the evaluated networks as a function of the number of templates (n). Fig. 5 plots the same results against the number of learned parameters in the networks. For the SimNet, we experimented with up to 400 templates (we believe that more templates would only give marginal improvements in accuracy, and thus did not continue further), and reached accuracies of 76.8% and 77.1% with weighted l_1 and l_2 similarities respectively. We ran the ConvNet with up to $6,400$ templates (beyond that Caffe had GPU memory management issues), with the highest accuracy standing at 76.2%. In comparison, taking into account that with weighted similarities each template carries with it a weight vector, the size (number of learned parameters) of the 400-template SimNet with weighted similarities was less than $1/9$ the size of the $6,400$-template ConvNet, while achieving slightly superior accuracy. The performance of the SimNet with unweighted similarities on the other hand, is very similar to that of the ConvNet, thus highlighting the importance of the weights in the similarity layer. The weights double the number of parameters in the layer, but the increase in performance scales up super-linearly with the number of added parameters. In other words, weights provide a gain in accuracy which is much higher than what would be obtained by simply adding more templates until reaching the same network size. For example, the accuracies with weights at 100 templates are considerably higher than the accuracies without weights at 200 templates, despite the fact that in the latter case, the overall network size is higher.

It is worth noting that the performance of the SimNet with unweighted l_1 and l_2 similarities is comparable to that of the ConvNet. This confirms what we observed formally in subsec. 3.1 – the hypothesis space (analyzed through the shapes of decision regions) corresponding to unweighted l_2-similarity is essentially the same as that which corresponds to linear similarity (convolutional operator). The hypothesis space corresponding to unweighted l_1-similarity is different, but apparently does not provide a higher degree of abstraction (further study of this is deferred to future work).

Although the SimNet accuracies achieved here are not state of the art for this dataset, the results demonstrate the potential of SimNets for modeling learning problems with significant reduction in network sizes compared to ConvNets.

6.2 Benchmarking against the "single-layer" network of Coates et al.

The "single-layer" network studied by Coates et al. ([5]) is of interest on several accounts. First, with GMM coding, the network is equivalent to the SimNet variant presented in eqn. 9. Second, their best result with "triangle" coding is one of the highest accuracies on CIFAR-10 reported for networks of this depth (absolute state of the art in 2011). Third, their observations with respect to the effect of whitening are relevant to the SimNet architecture, and indeed, we found that for the evaluated SimNet with l_1 and l_2 similarities, whitening makes a difference.

In [5], the network "templates" (i.e. the parameters of the selected coding method) were set using unlabeled data, and were not modified in the supervised training phase. To facilitate a fair comparison against our SimNet (where templates are modified in supervised training), we added an additional supervised training phase, which applied to both the templates and the SVM coefficients.

More specifically, we used SGD to jointly modify the network templates and SVM coefficients produced by [5], in an attempt to reach higher accuracy levels than those reported by the authors. As it turned out, with the triangle coding they proposed, the supervised update of the templates did not improve accuracy any further than the original k-means clustering. Deep inspection of this phenomena revealed that the k-means clustering (along with the SVM that follows) provides a strong local minima for the learning problem, so even the training accuracy was not improved. This leads us to believe that the triangle coding is so successful precisely because it creates a representation for which k-means finds optimal templates, that cannot be improved even in the presence of labeled data. In [5] results are reported for up to $1,600$ templates. We used their code to reproduce these results, while running up to $6,400$ templates. The accuracy curve we obtained is displayed in fig. 4, with 77.3% for $1,600$ templates, 78.3% for $3,200$ templates, and 77.8% for $6,400$ templates. The peak accuracy was achieved for $3,200$ templates, and was slightly higher than the SimNet peak accuracy, which stood at 77.1% for weighted l_2-similarity and 400 templates. The SimNet on the other hand was almost $1/5$ in size (see fig. 5).

6.3 The importance of unsupervised initialization

To assess the importance of the SimNets' unsupervised initialization scheme presented in sec. 5, we trained the evaluated SimNet (fig. 3(a)) with weighted l_1 similarity, using no data for initialization. In particular, we initialized the templates $\mathbf{z}_1, ..., \mathbf{z}_n$ randomly with a zero-mean unit-variance Gaussian distribution (in accordance with the fact that the input patches are whitened to have zero-mean and unit variance), and the weights $\mathbf{u}_1, ..., \mathbf{u}_n$ with constant ones. Besides the difference in initialization, the SimNet was trained exactly as described above. Running the experiment with 200 templates, cross-validation accuracy dropped from 76% to 74.1%. With 400 templates, accuracy declined from 76.8% to 74.4%. With 50 and 100 templates, the learning algorithm did not converge. We conclude that the SimNet unsupervised initialization scheme indeed has significant impact on performance. The impact is especially acute for networks of small size. This complies with conventional wisdom, according to which training small networks poses more difficult optimization problems. The SimNet initialization scheme may provide an alternative to the common practice of over-specifying networks (constructing networks larger than necessary in order to ease the optimization task).

7 Discussion

We presented a deep layered architecture called SimNets, with similar ingredients as classical ConvNets. The architecture is driven by two operators: *(i)* the similarity operator, which is a generalization of the convolutional operator in ConvNets, and *(ii)* the MEX operator, which can realize classical operators found in ConvNets like ReLU and max pooling, but has additional capabilities that make SimNets a powerful generalization of ConvNets. One of the interesting properties of the SimNet architecture is that applying its two operators in succession – similarity followed by MEX, results in what can be viewed as an artificial neuron in a high-dimensional feature space. Moreover, the multilayer perceptron construction of input to hidden layer to output, as well as the fundamental building block incorporating locality, sharing and pooling, are both generalizations of kernel machines.

We described two possible similarity measures: the l_p similarity, which in its unweighted version gives rise to the Generalized Gaussian kernel, and the linear similarity, which is the operator found in ConvNets, and gives rise to the Exponential kernel. We also showed that the full specification of the l_p similarity operator, which includes weights, goes beyond a kernel machine and carries with it a higher abstraction level than what a convolutional layer can express. Another interesting property of the SimNet architecture is that statistical estimation methods for Generalized Gaussian mixture distributions can be used for unsupervised initialization of network parameters. These initializations arise naturally from standard statistical assumptions, having the potential of employing unsupervised learning in an effective manner as part of deep learning.

Implementing deep SimNets with state of the art optimization techniques (including GPU acceleration) is an ongoing effort, but we were able to implement a basic SimNet and conduct benchmarks comparing it against two networks of the same depth – an analogous ConvNet and the "single-layer" network of [5]. The results demonstrate that a SimNet can achieve comparable and/or better accu-

racy, while requiring a significantly smaller network (in terms of the number of learned parameters) – around $1/9$ the size of the ConvNet and $1/5$ the size of the network in [5].

The SimNet architecture departs from classical ConvNets in three main respects. First, the similarity layer can incorporate entry-wise weights when the l_p similarity is used. With linear similarity (which is essentially an inner-product between an input patch and a convolutional kernel) incorporating weights is meaningless, as they blend into the convolutional kernels. We saw that the unweighted l_p and linear similarities give rise to a kernel-SVM building block with the Generalized Gaussian and Exponential kernels, respectively. The weighted l_p similarity on the other hand, cannot be realized in the kernel-SVM framework (thm. 1), thereby offering a potentially stronger building block (whose effect is described in more detail in subsec. 3.1). The experiments we carried out highlight the differences between weighted and unweighted l_p similarities:

- Without weights, l_p similarity and the linear similarity (convolutional operator) give rise to comparable performance. This suggests that without weights, SimNets do not exhibit superiority over ConvNets.
- When weights are included, l_p similarity displays a significant increase in performance, which scales up super-linearly with the number of parameters. That is to say, the increase in accuracy cannot be explained merely by the fact that the number of parameters in the similarity layer has been doubled (weights on top of templates).

These findings suggest that the strength of having the basic building block go beyond the hypothesis space of a kernel-SVM, has significant appeal in practice.

The second respect in which SimNets depart from ConvNets has to do with the ability of the MEX layer to incorporate offsets. When the MEX layer serves as the final layer of the network, these offsets play the role of classification coefficients. However, when the MEX layer is inserted as a pooling layer, the offsets can be interpreted as providing locality-based biases to the templates generated in a previous similarity layer. This is something that classical ConvNets cannot express. Evaluating the practical significance of the MEX offsets requires experimentation with deep layered SimNets, which is an ongoing effort.

The third departure (or distinction) from ConvNets, is that the SimNet architecture is endowed with a natural initialization based on unlabeled data. In the case of ConvNets, existing unsupervised initialization schemes have little to no advantage over random initializations. For the SimNets, we reported experimental results that demonstrate the superiority of the unsupervised initialization scheme over random initializations, showing that the effect is more acute when the networks are small. Besides its aid in training, the unsupervised scheme proposed also has the potential of determining the number of channels for a similarity layer based on variance analysis of patterns generated from previous layers. This implies that the structure of SimNets can potentially be determined automatically from (unlabeled) training data.

Future work is focused on further implementation, with the purpose of creating an open programming environment for the research community, that will enable wider scale experimentation of SimNets. Further theoretical studies are ongoing as well, with the intent to capture the sample complexity of SimNets, and to gain a better understanding of the typical network structure and size required under different conditions.

Acknowledgments

The authors would like to thank Nitzan Guberman and Or Sharir for their dedicated contribution to the experiments carried out in this work. The work is partly funded by Intel grant ICRI-CI no. 9-2012-6133 and by ISF Center grant 1790/12.

References

[1] Yakoub Bazi, Lorenzo Bruzzone, and Farid Melgani. Image thresholding based on the em algorithm and the generalized gaussian distribution. *Pattern Recognition*, 40(2):619–634, 2007.

[2] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153, 2007.

[3] Joan Bruna and Stéphane Mallat. Invariant scattering convolution networks. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1872–1886, 2013.

[4] Youngmin Cho and Lawrence K Saul. Kernel methods for deep learning. In *Advances in neural information processing systems*, pages 342–350, 2009.

[5] Adam Coates, Andrew Y Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *International Conference on Artificial Intelligence and Statistics*, pages 215–223, 2011.

[6] Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *The Journal of Machine Learning Research*, 2:265–292, 2002.

[7] Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.

[8] Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

[9] Caglar Gulcehre, Kyunghyun Cho, Razvan Pascanu, and Yoshua Bengio. Learned-norm pooling for deep feedforward and recurrent neural networks. In *Machine Learning and Knowledge Discovery in Databases*, pages 530–546. Springer, 2014.

[10] Geoffrey Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[11] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[12] Yangqing Jia. Caffe: An open source convolutional architecture for fast feature embedding. `http://caffe.berkeleyvision.org/`, 2013.

[13] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. *Computer Science Department, University of Toronto, Tech. Rep*, 2009.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[15] Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361, 1995.

[16] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[17] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[18] Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 689–690. IEEE, 2011.

[19] Maximilian Riesenhuber and Tomaso Poggio. Hierarchical models of object recognition in cortex. *Nature neuroscience*, 2(11):1019–1025, 1999.

[20] Bernhard Schölkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[21] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[22] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1139–1147, 2013.

[23] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014.

[24] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708, 2013.

[25] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[26] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1058–1066, 2013.

[27] Mingrui Wu, Bernhard Schölkopf, and Gökhan Bakır. A direct method for building sparse kernel learning algorithms. *The Journal of Machine Learning Research*, 7:603–624, 2006.

[28] Nicholas Young. *An introduction to Hilbert space*. Cambridge university press, 1988.

[29] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014*, pages 818–833. Springer, 2014.

A Proof of theorem 1

To prove the theorem, we will need the following definition and lemma:

Definition 1. *Let $\mathcal{H}$ be a Hilbert space and $S \subset \mathcal{H}$ be a collection of vectors. Given a constant $\epsilon > 0$, S is said to be an* ϵ-orthonormal set *if the following two conditions hold:*

$$\forall \mathbf{v} \in S: \ \|\mathbf{v}\|_{\mathcal{H}} = 1$$
$$\forall \mathbf{v}, \mathbf{v}' \in S \ , \ \mathbf{v} \neq \mathbf{v}': \ |\langle \mathbf{v}, \mathbf{v}' \rangle_{\mathcal{H}}| \leq \epsilon$$

where $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ stands for the inner-product in $\mathcal{H}$ and $\|\cdot\|_{\mathcal{H}}$ denotes the induced norm.

Lemma 1. *Let $\mathcal{H}$ be a Hilbert space over F ($F = \mathbb{R}$ or $F = \mathbb{C}$) and $V \subset \mathcal{H}$ be a set that contains an ϵ-orthonormal subset (see def. 1) of size n for any constants $\epsilon > 0$ and $n \in \mathbb{N}$. Then, for every vector $\mathbf{u} \in \mathcal{H}$ it holds that* $\inf\{|\langle \mathbf{u}, \mathbf{v} \rangle_{\mathcal{H}}| : \mathbf{v} \in V\} = 0$.

Proof. Let $\mathbf{u} \in \mathcal{H}$ be an arbitrary vector, and denote $c := \inf\{|\langle \mathbf{u}, \mathbf{v} \rangle_{\mathcal{H}}| : \mathbf{v} \in V\}$ and $M := \|\mathbf{u}\|_{\mathcal{H}}$. We would like to show that $c = 0$. Let $\epsilon > 0$ and $n \in \mathbb{N}$ be arbitrary constants. Given our assumption on V, we may choose an ϵ-orthonormal subset $\mathbf{v}_1, ..., \mathbf{v}_n \in V$. Denote by G the Gram of $\mathbf{v}_1, ..., \mathbf{v}_n$, i.e. $G \in F^{n,n}$ is the positive semi-definite (PSD) matrix with entries $G_{ij} = \langle \mathbf{v}_j, \mathbf{v}_i \rangle_{\mathcal{H}}$. Let $\alpha_1, ..., \alpha_n \in F$ be the scalars such that $\sum_{i=1}^n \alpha_i \mathbf{v}_i$ is the projection of $\mathbf{u}$ onto $span\{\mathbf{v}_i\}_{i=1}^n$. It then holds that:

$$\|\textstyle\sum_{i=1}^n \alpha_i \mathbf{v}_i\|_{\mathcal{H}}^2 \leq \|\mathbf{u}\|_{\mathcal{H}}^2 = M^2$$
$$\forall j \in \{1, ..., n\} : |\langle \textstyle\sum_{i=1}^n \alpha_i \mathbf{v}_i, \mathbf{v}_j \rangle_{\mathcal{H}}| = |\langle \mathbf{u}, \mathbf{v}_j \rangle_{\mathcal{H}}| \geq c$$

If we denote $\boldsymbol{\alpha} := [\alpha_1, ..., \alpha_n]^\top$, and let $\mathbf{1}$ be the n-dimensional vector holding 1 in all entries, the above yields the following matrix inequalities:

$$\boldsymbol{\alpha}^* G \boldsymbol{\alpha} \leq M^2 \tag{11}$$
$$|G\boldsymbol{\alpha}| \geq c \cdot \mathbf{1} \implies \boldsymbol{\alpha}^* G^* G \boldsymbol{\alpha} = \|G\boldsymbol{\alpha}\|_2^2 \geq c^2 \cdot n \tag{12}$$

where $*$ stands for the conjugate transpose operator. The matrix G is PSD, thus having n non-negative eigenvalues. We denote these by $\lambda_1 \geq ... \geq \lambda_n \geq 0$. G is Hermitian and thus $G^*G = G^2$, from which we readily conclude that $\lambda_1^2 \geq ... \geq \lambda_n^2 \geq 0$ are the eigenvalues of G^*G. We thus have the following inequalities:

$$\boldsymbol{\alpha}^* G \boldsymbol{\alpha} \geq \lambda_n \cdot \|\boldsymbol{\alpha}\|_2^2 \tag{13}$$
$$\boldsymbol{\alpha}^* G^* G \boldsymbol{\alpha} \leq \lambda_1^2 \cdot \|\boldsymbol{\alpha}\|_2^2 \tag{14}$$

Combining the inequality 11 with 13, and the inequality 12 with 14, we get the following:

$$\lambda_n \cdot \|\boldsymbol{\alpha}\|_2^2 \quad \leq \quad M^2 \tag{15}$$
$$\lambda_1^2 \cdot \|\boldsymbol{\alpha}\|_2^2 \quad \geq \quad c^2 \cdot n \tag{16}$$

We now apply Gershgorin's circle theorem (see [7]) to G. The theorem states that for each eigenvalue λ_i, there exists some $j \in \{1, ..., n\}$ such that:

$$|\lambda_i - G_{jj}| \leq \sum_{j' \in \{1,...n\}, j' \neq j} |G_{jj'}|$$

Plugging in the definition of G and the ϵ-orthonormality of $\{\mathbf{v}_1, ..., \mathbf{v}_n\}$, we get:

$$|\lambda_i - \underbrace{\langle \mathbf{v}_j, \mathbf{v}_j \rangle}_{=1}| \leq \sum_{j' \in \{1,...n\}, j' \neq j} \underbrace{|\langle \mathbf{v}_{j'}, \mathbf{v}_j \rangle|}_{\leq \epsilon} \leq (n-1) \cdot \epsilon$$
$$\implies 1 - (n-1) \cdot \epsilon \leq \lambda_i \leq 1 + (n-1) \cdot \epsilon$$

Recall that $\epsilon > 0$ and $n \in \mathbb{N}$ were chosen arbitrarily. If we now limit ϵ to be smaller than $\frac{1}{n-1}$, we ensure that $\lambda_i > 0$ for all $i = 1, ..., n$. We can thus divide by λ_n and λ_1^2 the inequalities 15 and 16 respectively, and reach:

$$\|\boldsymbol{\alpha}\|_2^2 \leq \tfrac{M^2}{\lambda_n} \leq \tfrac{M^2}{1-(n-1)\cdot\epsilon}$$
$$\|\boldsymbol{\alpha}\|_2^2 \geq \tfrac{c^2 \cdot n}{\lambda_1^2} \geq \tfrac{c^2 \cdot n}{(1+(n-1)\cdot\epsilon)^2}$$

Combining these two inequalities, we get:

$$M^2 \geq \frac{1-(n-1)\cdot\epsilon}{(1+(n-1)\cdot\epsilon)^2}\cdot c^2\cdot n$$

Now this holds for ϵ arbitrarily small, so in particular:

$$M^2 \geq \underbrace{\left(\lim_{\epsilon\to\mathbf{0}^+}\frac{1-(n-1)\cdot\epsilon}{(1+(n-1)\cdot\epsilon)^2}\right)}_{=1}\cdot c^2\cdot n = c^2\cdot n \tag{17}$$

n is an arbitrary natural number, and M was defined as the norm of $\mathbf{u}\in\mathcal{H}$ so in particular it is non-negative and finite. In addition, c was defined as $\inf\{|\langle\mathbf{u},\mathbf{v}\rangle_{\mathcal{H}}| : \mathbf{v}\in V\}$ so it is too non-negative. Thus, the only way that eqn. 17 can hold is if $c=0$, which is what we set out to prove. □

Equipped with def. 1 and lemma 1, we head on to prove our main theorem:

Proof of theorem 1. Assume by contradiction that there are mappings Z and U and kernel K as described in the theorem. Let ψ be a feature mapping corresponding to K, i.e. a mapping from $\mathbb{R}^d\times\mathbb{R}^d_+$ to some real Hilbert space $\mathcal{H}$ such that $K([\mathbf{z},\mathbf{u}],[\mathbf{z}',\mathbf{u}']) = \langle\psi([\mathbf{z},\mathbf{u}]),\psi([\mathbf{z}',\mathbf{u}'])\rangle_{\mathcal{H}}$ for all $\mathbf{z},\mathbf{z}'\in\mathbb{R}^d$ and $\mathbf{u},\mathbf{u}'\in\mathbb{R}^d_+$. Fix some $\mathbf{x}\in\mathbb{R}^d$, and observe that:

$$\|\psi([Z(\mathbf{x}),U(\mathbf{x})])\|_{\mathcal{H}}\leq 1 \tag{18}$$

This follows from:

$$\|\psi([Z(\mathbf{x}),U(\mathbf{x})])\|^2_{\mathcal{H}} =$$
$$K([Z(\mathbf{x}),U(\mathbf{x})],[Z(\mathbf{x}),U(\mathbf{x})]) =$$
$$\exp\left\{-\underbrace{c}_{>0}\sum_{i=1}^d\underbrace{U(\mathbf{x})_i}_{\geq 0}\underbrace{|x_i-Z(\mathbf{x})_i|^p}_{\geq 0}\right\}\leq 1$$

Since $Z(\mathbf{x})$ is also an element in $\mathbb{R}^d$, we can replace $\mathbf{x}$ by $Z(\mathbf{x})$ in eqn. 18 to obtain:

$$\|\psi([Z(Z(\mathbf{x})),U(Z(\mathbf{x}))])\|_{\mathcal{H}}\leq 1 \tag{19}$$

Next we show that:

$$\langle\psi([Z(Z(\mathbf{x})),U(Z(\mathbf{x}))]),\psi([Z(\mathbf{x}),U(\mathbf{x})])\rangle_{\mathcal{H}} = 1 \tag{20}$$

Indeed:

$$\langle\psi([Z(Z(\mathbf{x})),U(Z(\mathbf{x}))]),\psi([Z(\mathbf{x}),U(\mathbf{x})])\rangle_{\mathcal{H}} =$$
$$K([Z(Z(\mathbf{x})),U(Z(\mathbf{x}))],[Z(\mathbf{x}),U(\mathbf{x})]) =$$
$$\exp\left\{-c\sum_{i=1}^d U(\mathbf{x})_i\underbrace{|Z(\mathbf{x})_i-Z(\mathbf{x})_i|^p}_{=0}\right\} = 1$$

The Cauchy-Schwartz inequality (see [28]) tells us that for any two vectors $\mathbf{w},\mathbf{w}'$ in the real Hilbert space $\mathcal{H}$, it holds that $\langle\mathbf{w},\mathbf{w}'\rangle_{\mathcal{H}}\leq\|\mathbf{w}\|_{\mathcal{H}}\cdot\|\mathbf{w}'\|_{\mathcal{H}}$. Moreover, if equality holds then $\mathbf{w}$ and $\mathbf{w}'$ are linearly dependent, or more specifically, at least one of the vectors can be obtained by multiplying the other by a non-negative scalar. Applying this to the vectors $\psi([Z(\mathbf{x}),U(\mathbf{x})])$ and $\psi([Z(Z(\mathbf{x})),U(Z(\mathbf{x}))])$, we conclude from equations 18, 19 and 20 that:

$$\psi([Z(Z(\mathbf{x})),U(Z(\mathbf{x}))]) = \psi([Z(\mathbf{x}),U(\mathbf{x})]) \tag{21}$$
$$\|\psi([Z(\mathbf{x}),U(\mathbf{x})])\|_{\mathcal{H}} = 1 \tag{22}$$

Using eqn. 21 and our assumption about the kernel K (eqn. 5), we conclude that for every $\mathbf{z}\in\mathbb{R}^d$ and $\mathbf{u}\in\mathbb{R}^d_+$ (recall that $\mathbf{x}\in\mathbb{R}^d$ was fixed arbitrarily):

$$\exp\left\{-c\sum_{i=1}^d u_i|x_i-z_i|^p\right\} =$$
$$K\left([Z(\mathbf{x}),U(\mathbf{x})],[\mathbf{z},\mathbf{u}]\right) =$$
$$\langle\psi([Z(\mathbf{x}),U(\mathbf{x})]),\psi([\mathbf{z},\mathbf{u}])\rangle_{\mathcal{H}} =$$
$$\langle\psi([Z(Z(\mathbf{x})),U(Z(\mathbf{x}))]),\psi([\mathbf{z},\mathbf{u}])\rangle_{\mathcal{H}} =$$
$$\exp\left\{-c\sum_{i=1}^d u_i|Z(\mathbf{x})_i-z_i|^p\right\}$$

Taking the logarithm of the two outer expressions, we get:

$$-c\sum_{i=1}^{d} u_i|x_i - z_i|^p = -c\sum_{i=1}^{d} u_i|Z(\mathbf{x})_i - z_i|^p$$
$$\implies \sum_{i=1}^{d} u_i\left(|x_i - z_i|^p - |Z(\mathbf{x})_i - z_i|^p\right) = 0$$

Fixing some coordinate $i_0 \in \{1, ..., d\}$, we can choose $\mathbf{u}$ to hold 1 at i_0 and 0 in the other coordinates. The latter equality would then reduce to $|x_{i_0} - z_{i_0}|^p = |Z(\mathbf{x})_{i_0} - z_{i_0}|^p$, which must hold for any $z_{i_0} \in \mathbb{R}$. The only way for this to be met is if $Z(\mathbf{x})_{i_0} = x_{i_0}$. Since both the vector $\mathbf{x} \in \mathbb{R}^d$ and the coordinate $i_0 \in \{1, ..., d\}$ are arbitrary, the mapping $Z : \mathbb{R}^d \to \mathbb{R}^d$ is no other than the identity mapping. The assumption in eqn. 5 thus becomes:

$$\forall\, \mathbf{z}, \mathbf{x} \in \mathbb{R}^d, \mathbf{u} \in \mathbb{R}^d_+ : \tag{23}$$
$$K\left([\mathbf{x}, U(\mathbf{x})], [\mathbf{z}, \mathbf{u}]\right) = \exp\left\{-c\textstyle\sum_{i=1}^{d} u_i|x_i - z_i|^p\right\}$$

We again fix $\mathbf{x} \in \mathbb{R}^d$, and turn to show that $U(\mathbf{x}) \neq \mathbf{0}$ ($\mathbf{0}$ here stands for the d-dimensional zero vector). Assume by contradiction that this is not the case, i.e. that $U(\mathbf{x}) = \mathbf{0}$. Then, according to eqn. 23, for all $\mathbf{x}' \in \mathbb{R}^d$ we have:

$$\langle\psi([\mathbf{x}', U(\mathbf{x}')]), \psi([\mathbf{x}, U(\mathbf{x})])\rangle_{\mathcal{H}} =$$
$$K([\mathbf{x}', U(\mathbf{x}')], [\mathbf{x}, U(\mathbf{x})]) =$$
$$\exp\left\{-c\textstyle\sum_{i=1}^{d} \underbrace{U(\mathbf{x})_i}_{=0}|x_i - x'_i|^p\right\} = 1$$

Using the fact that $\psi([\mathbf{x}, U(\mathbf{x})])$ and $\psi([\mathbf{x}', U(\mathbf{x}')])$ are unit vectors (eqn. 22), and again the Cauchy-Schwartz inequality, we conclude that $\psi([\mathbf{x}, U(\mathbf{x})]) = \psi([\mathbf{x}', U(\mathbf{x}')])$. This implies that for all $\mathbf{z} \in \mathbb{R}^d$ and $\mathbf{u} \in \mathbb{R}^d_+$:

$$\exp\left\{-c\textstyle\sum_{i=1}^{d} u_i|x_i - z_i|^p\right\} =$$
$$K\left([\mathbf{x}, U(\mathbf{x})], [\mathbf{z}, \mathbf{u}]\right) = \langle\psi([\mathbf{x}, U(\mathbf{x})]), \psi([\mathbf{z}, \mathbf{u}])\rangle_{\mathcal{H}} =$$
$$\langle\psi([\mathbf{x}', U(\mathbf{x}')]), \psi([\mathbf{z}, \mathbf{u}])\rangle_{\mathcal{H}} = K\left([\mathbf{x}', U(\mathbf{x}')], [\mathbf{z}, \mathbf{u}]\right) =$$
$$\exp\left\{-c\textstyle\sum_{i=1}^{d} u_i|x'_i - z_i|^p\right\}$$

As before, we can isolate the coordinates in $\{1, ..., d\}$ one at a time, and conclude that $\mathbf{x} = \mathbf{x}'$. Since $\mathbf{x}'$ is arbitrary, this is of course a contradiction, showing that our assumption $U(\mathbf{x}) = \mathbf{0}$ was incorrect. There is thus at least one coordinate of $U(\mathbf{x})$ which is positive. Accordingly, the expression $-c\sum_{i=1}^{d} U(\mathbf{x})_i|x'_i - x_i|^p$ will tend to $-\infty$ when all coordinates of $\mathbf{x}'$ tend to ∞ (we denote this condition by $\mathbf{x}' \to \infty$). We may thus write:

$$\langle\psi([\mathbf{x}', U(\mathbf{x}')]), \psi([\mathbf{x}, U(\mathbf{x})])\rangle_{\mathcal{H}} =$$
$$\exp\left\{-c\textstyle\sum_{i=1}^{d} U(\mathbf{x})_i|x'_i - x_i|^p\right\} \underset{\mathbf{x}'\to\infty}{\longrightarrow} 0$$

Recall that $\mathbf{x} \in \mathbb{R}^d$ is an arbitrary vector. The above convergence thus implies that for any $\epsilon > 0$ and $n \in \mathbb{N}$, we can incrementally create a set of n vectors - $\mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^d$, such that:

$$\forall 1 \leq j < i \leq n :$$
$$|\langle\psi([\mathbf{x}_i, U(\mathbf{x}_i)]), \psi([\mathbf{x}_j, U(\mathbf{x}_j)])\rangle_{\mathcal{H}}| \leq \epsilon$$

Indeed, given a set of vectors $\mathbf{x}_1, ..., \mathbf{x}_j$, the next vector $\mathbf{x}_{j+1}$ is obtained by approaching ∞ until all inner-products are small enough.

To summarize, we have the following findings:

- For any $\epsilon > 0$ and $n \in \mathbb{N}$ there exist $\mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^d$ such that for all $1 \leq j < i \leq n$, $|\langle\psi([\mathbf{x}_i, U(\mathbf{x}_i)]), \psi([\mathbf{x}_j, U(\mathbf{x}_j)])\rangle_{\mathcal{H}}| \leq \epsilon$.

- $\|\psi([\mathbf{x}, U(\mathbf{x})])\|_{\mathcal{H}} = 1$ for all $\mathbf{x} \in \mathbb{R}^d$ (eqn. 22).
- $\langle \psi([\mathbf{x}, U(\mathbf{x})]), \psi([\mathbf{0}, \mathbf{0}]) \rangle_{\mathcal{H}} = K([\mathbf{x}, U(\mathbf{x})], [\mathbf{0}, \mathbf{0}]) = \exp\left\{-c \sum_{i=1}^{d} 0 \cdot |x_i - 0|^p\right\} = 1$ (simply plug-in $\mathbf{z} = \mathbf{0}$ and $\mathbf{u} = \mathbf{0}$ in eqn. 23).

More succinctly, the set $V := \{\psi([\mathbf{x}, U(\mathbf{x})]) : \mathbf{x} \in \mathbb{R}^d\}$ contains an ϵ-orthonormal subset (def. 1) of size n for any constants $\epsilon > 0$ and $n \in \mathbb{N}$, and in addition the vector $\psi([\mathbf{0}, \mathbf{0}])$ has inner-product 1 with every element of V. According to lemma 1 this is impossible! We have thus reached a contradiction, showing the incorrectness of our initial assumption that mappings Z and U and kernel K as stated in the theorem exist. □

B Patch-based kernel-SVM

In this appendix we show how the classification described in eqn. 10, which corresponds to the basic "locality-sharing-pooling" SimNet illustrated in fig. 1(f), can be formulated as a multiclass kernel-SVM ([6]) with reduced support-vectors ([27]). In this formulation, the classified instances will not be represented by holistic vectors, but rather by blocks of multiple vectors. Moreover, the support-vectors will be subject to constraints which can be interpreted as enforcing "locality" and "sharing". In the context of the SimNet, the vectors which constitute an instance are simply the input patches, the locality constraint on the support-vectors corresponds to the fact that the input is processed by local patches in a spatially aware manner, and the sharing constraint corresponds to the fact that the same n templates in the similarity layer apply to all input patches. As will be shown below, the SimNet's pooling operation will also come into play in the locality and sharing constraints.

Let $d \in \mathbb{N}$ be some dimension, and let $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be a kernel on $\mathbb{R}^d$. For some $D \in \mathbb{N}$, consider the instance space $\mathcal{X} := \{X = (\mathbf{x}_1, ..., \mathbf{x}_D) : \mathbf{x}_i \in \mathbb{R}^d,\ i = 1, ..., D\}$. For compatibility, we refer to the vectors that constitute an instance as "patches". Assume we have a partitioning of patches into "pools", namely that there is a constant $P \in \mathbb{N}$ and a function $q : \{1, ..., D\} \to \{1, ..., P\}$ that assigns to each patch index $i \in \{1, ..., D\}$ a pool $q(i) \in \{1, ..., P\}$. Consider the following rule for classifying an instance X into one of $k \in \mathbb{N}$ possible classes:

$$\hat{y}(X) = \operatorname*{argmax}_{r=1,...,k} \sum_{1 \le p \le P, 1 \le l \le n} \alpha_{rlp} \sum_{1 \le i \le D : q(i) = p} K(\mathbf{x}_i, \mathbf{z}_l) \tag{24}$$

where $n \in \mathbb{N}$ is some predetermined constant, $\{\mathbf{z}_1, ..., \mathbf{z}_n\} \subset \mathbb{R}^d$ are learned templates, and $\{\alpha_{rlp}\}_{1 \le r \le k, 1 \le l \le n, 1 \le p \le P} \subset \mathbb{R}$ are learned coefficients. This is essentially equivalent to the SimNet classification described in eqn. 10. In fact, the only true difference is that in the latter, the learned coefficients were constrained to be positive, but this does not limit generality, as we can always add a common offset to all coefficients after training is complete.

We now add the special character $*$ ("null" character) to $\mathbb{R}^d$, extending the latter to $V := \mathbb{R}^d \cup \{*\}$. Accordingly, we extend K to the function $K_V : V \times V \to \mathbb{R}$ defined by:

$$K_V(\mathbf{v}, \mathbf{v}') = \begin{cases} K(\mathbf{v}, \mathbf{v}') & \text{if } \mathbf{v}, \mathbf{v}' \neq * \\ 0 & \text{otherwise} \end{cases} \tag{25}$$

Lemma 2. *K_V is a kernel on V.*

Proof. Let ψ be a feature mapping corresponding to the kernel K, i.e. ψ is a mapping from $\mathbb{R}^d$ to some Hilbert space $\mathcal{H}$ such that $\forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^d : K(\mathbf{x}, \mathbf{x}') = \langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle$. Extend ψ to the mapping $\psi_V : V \to \mathcal{H}$ as follows:

$$\psi_V(\mathbf{v}) = \begin{cases} \psi(\mathbf{v}) & \text{if } \mathbf{v} \neq * \\ 0_{\mathcal{H}} & \text{if } \mathbf{v} = * \end{cases}$$

where $0_{\mathcal{H}}$ stands for the zero element of $\mathcal{H}$. Obviously, the function from $V \times V$ to $\mathbb{R}$ defined by $(\mathbf{v}, \mathbf{v}') \mapsto \langle \psi_V(\mathbf{v}), \psi_V(\mathbf{v}') \rangle$ is a kernel on V. Direct computation shows that this function is no

other than K_V:

$$\begin{aligned}
&\langle \psi_V(\mathbf{v}), \psi_V(\mathbf{v}') \rangle \\
&= \begin{cases} \langle \psi(\mathbf{v}), \psi(\mathbf{v}') \rangle & \text{if } \mathbf{v} \neq *, \mathbf{v}' \neq * \\ \langle \psi(\mathbf{v}), 0_{\mathcal{H}} \rangle & \text{if } \mathbf{v} \neq *, \mathbf{v}' = * \\ \langle 0_{\mathcal{H}}, \psi(\mathbf{v}') \rangle & \text{if } \mathbf{v} = *, \mathbf{v}' \neq * \\ \langle 0_{\mathcal{H}}, 0_{\mathcal{H}} \rangle & \text{if } \mathbf{v} = *, \mathbf{v}' = * \end{cases} \\
&= \begin{cases} \langle \psi(\mathbf{v}), \psi(\mathbf{v}') \rangle & \text{if } \mathbf{v} \neq *, \mathbf{v}' \neq * \\ 0 & \text{otherwise} \end{cases} \\
&= \begin{cases} K(\mathbf{v}, \mathbf{v}') & \text{if } \mathbf{v} \neq *, \mathbf{v}' \neq * \\ 0 & \text{otherwise} \end{cases} \\
&= K_V(\mathbf{v}, \mathbf{v}')
\end{aligned}$$

□

Next, we use the kernel K_V to define a function $\mathbf{K} : V^D \times V^D \to \mathbb{R}$, where V^D is the D'th Cartesian power of V, i.e. $V^D := \{(\mathbf{v}_1, ..., \mathbf{v}_D) : \mathbf{v}_i \in V \ , \ i = 1, ..., D\}$. $\mathbf{K}$ is defined by:

$$\begin{aligned}
\mathbf{K}((\mathbf{v}_1, ..., \mathbf{v}_D), (\mathbf{v}'_1, ..., \mathbf{v}'_D)) &= \textstyle\sum_{1 \leq i \leq D} K_V(\mathbf{v}_i, \mathbf{v}'_i) \\
&= \textstyle\sum_{1 \leq i \leq D : \mathbf{v}_i, \mathbf{v}'_i \neq *} K(\mathbf{v}_i, \mathbf{v}'_i)
\end{aligned} \tag{26}$$

Lemma 3. $\mathbf{K}$ *is a kernel on* V^D.

Proof. The proof is general in the sense that it does not rely on any specific property of the kernel K_V on which $\mathbf{K}$ is based. In accordance with the above notations, let ψ_V be a feature mapping corresponding to K_V, i.e. a mapping from V to some Hilbert space $\mathcal{H}$ such that $\forall \mathbf{v}, \mathbf{v}' \in V$: $K_V(\mathbf{v}, \mathbf{v}') = \langle \psi_V(\mathbf{v}), \psi_V(\mathbf{v}') \rangle$. We use ψ_V to define a mapping Ψ from V^D to the Hilbert space $\mathcal{H}^D$, which is the D'th Cartesian power of $\mathcal{H}$ (the elements of $\mathcal{H}^D$ are D-length sequences of $\mathcal{H}$-elements, and the inner product between $(\mathbf{h}_1, ..., \mathbf{h}_D)$ and $(\mathbf{h}'_1, ..., \mathbf{h}'_D)$ is defined as $\sum_{i=1}^D \langle \mathbf{h}_i, \mathbf{h}'_i \rangle$). The mapping Ψ is defined by $\Psi((\mathbf{v}_1, ..., \mathbf{v}_D)) = (\psi_V(\mathbf{v}_1), ..., \psi_V(\mathbf{v}_D))$. The function from $V^D \times V^D$ to $\mathbb{R}$ defined by $((\mathbf{v}_1, ..., \mathbf{v}_D), (\mathbf{v}'_1, ..., \mathbf{v}'_D)) \mapsto \langle \Psi((\mathbf{v}_1, ..., \mathbf{v}_D)), \Psi((\mathbf{v}'_1, ..., \mathbf{v}'_D)) \rangle$ is obviously a kernel on V^D. Direct computation shows that this function is no other than $\mathbf{K}$:

$$\begin{aligned}
&\langle \Psi((\mathbf{v}_1, ..., \mathbf{v}_D)), \Psi((\mathbf{v}'_1, ..., \mathbf{v}'_D)) \rangle \\
&= \langle (\psi_V(\mathbf{v}_1), ..., \psi_V(\mathbf{v}_D)), (\psi_V(\mathbf{v}'_1), ..., \psi_V(\mathbf{v}'_D)) \rangle \\
&= \textstyle\sum_{i=1}^D \langle \psi_V(\mathbf{v}_i), \psi_V(\mathbf{v}'_i) \rangle = \sum_{i=1}^D K_V(\mathbf{v}_i, \mathbf{v}'_i) \\
&= \mathbf{K}((\mathbf{v}_1, ..., \mathbf{v}_D), (\mathbf{v}'_1, ..., \mathbf{v}'_D))
\end{aligned}$$

□

Using $\mathbf{K}$, we may express the classification rule given in eqn. 24 in kernel-form. Simply notice that:

$$\textstyle\sum_{1 \leq i \leq D : q(i) = p} K(\mathbf{x}_i, \mathbf{z}_l) =$$

$$\mathbf{K}((\mathbf{x}_1, ..., \mathbf{x}_D), \overbrace{(\underbrace{*, ..., *}_{q(i) \neq p}, \underbrace{\mathbf{z}_l, ..., \mathbf{z}_l}_{q(i) = p}, \underbrace{*, ..., *}_{q(i) \neq p})}^{:= Z_{lp}})$$

Thus, eqn. 24 can be written as follows:

$$\hat{y}(X) = \underset{r=1,...,k}{\operatorname{argmax}} \sum_{1 \leq p \leq P, 1 \leq l \leq n} \alpha_{rlp} \cdot \mathbf{K}(X, Z_{lp})$$

where $Z_{lp} = ((Z_{lp})_1, ..., (Z_{lp})_D)$, for $l = 1, ..., n$ and $p = 1, ..., P$, are elements in V^D meeting the following constraints:

$$\forall i \in \{1, ..., D\} \ s.t. \ q(i) \neq p : \ (Z_{lp})_i = * \tag{27}$$

$$\forall i \in \{1,...,D\}\ s.t.\ q(i) = p:\ (Z_{lp})_i = \mathbf{z}_l \tag{28}$$
$$\text{for some global vectors } \mathbf{z}_1,...,\mathbf{z}_n \in \mathbb{R}^d$$

We interpret the constraint in eqn. 27 as enforcing locality – entries of Z_{lp} that lie outside the pool p ("out-pool" entries) must hold the null character. The constraint in eqn. 28 is interpreted as enforcing sharing – entries of Z_{lp} that lie inside the pool p ("in-pool" entries) are identical to each other, and also to the in-pool entries of $Z_{l'p'}$ in the case where the "template indexes" l and l' are the same.

To conclude, the classifier described in eqn. 24 can also be expressed as:

$$\hat{y}(X) = \underset{r=1,...,k}{\operatorname{argmax}} \sum_{1\leq p\leq P, 1\leq l\leq n} \alpha_{rlp} \cdot \mathbf{K}(X, Z_{lp}) \tag{29}$$

where:

- The set V^D is simply the instance space $\mathcal{X}$ with the option of placing null characters in the different entries.
- $\mathbf{K}$ is a kernel on V^D.
- Z_{lp}, with $l = 1,...,n$ and $p = 1,...P$, are learned elements of V^D meeting the locality and sharing constraints in eqn. 27 and eqn. 28 respectively.
- α_{rlp}, with $r = 1,...,k$, $l = 1,...,n$ and $p = 1,...P$, are learned real coefficients.

That is to say, the classifier is a reduced kernel-SVM on the space V^D with the kernel $\mathbf{K}$, where the train and test instances are known to lie in the subset $\mathcal{X} \subset V^D$ (i.e. they do not contain any null characters), and there are $n \cdot P$ support-vectors indexed by $(l,p) \in \{1,...,n\} \times \{1,...,P\}$, that are subject to the locality and sharing constraints in eqn. 27 and eqn. 28 respectively. This construction, which we refer to as "patch-based kernel-SVM", underlines the strong connection between SimNets and kernel machines. In particular, it demonstrates the effect of locality, sharing and pooling in SimNets on the kernel-SVM equivalent. Namely, while the basic SimNet (illustrated in fig. 1(e)) was associated with standard reduced kernel-SVM, adding locality, sharing and pooling to obtain the SimNet considered here (illustrated in fig. 1(f)), translates the associated kernel machine to patch-based kernel-SVM, in which the concepts of locality, sharing and pooling come into play as constraints on the support-vectors.