Deep SimNets

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

Or Sharir
The Hebrew University of Jerusalem
or.sharir@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 convolutional neural networks (ConvNets). The architecture, called SimNets, is driven by two operators: (i) a similarity function that generalizes inner-product, and (ii) a log-mean-exp function called MEX that generalizes maximum and average. The two operators applied in succession give rise to a standard neuron but in "feature space". The feature spaces realized by SimNets depend on the choice of the similarity operator. The simplest setting, which corresponds to a convolution, realizes the feature space of the Exponential kernel, while other settings realize feature spaces of more powerful kernels (Generalized Gaussian, which includes as special cases RBF and Laplacian), or even dynamically learned feature spaces (Generalized Multiple Kernel Learning). As a result, the SimNet contains a higher abstraction level compared to a traditional ConvNet. We argue that enhanced expressiveness is important when the networks are small due to run-time constraints (such as those imposed by mobile applications). Empirical evaluation validates the superior expressiveness of SimNets, showing a significant gain in accuracy over ConvNets when computational resources at run-time are limited. We also show that in large-scale settings, where computational complexity is less of a concern, the additional capacity of SimNets can be controlled with proper regularization, yielding accuracies comparable to state of the art ConvNets.

1. Introduction

Deep neural networks, and convolutional neural networks (*ConvNets*) in particular, have had a dramatic impact in advancing the state of the art in computer vision, speech analysis, and many other domains (cf. [23, 36, 17]). It has been demonstrated time and time again, that when ConvNets are trained in an end-to-end manner, they deliver significantly better results than systems relying on manually engineered features.

The goal of this paper is to introduce a generalization of ConvNets we call Similarity Networks (*SimNets*), that preserves the simplicity and effectiveness of ConvNets, yet has a *higher abstraction level*. In a nutshell, the inner-product operator, which lies at the core of the ConvNet architecture, is replaced by an inner-product in "feature space". The feature spaces are controlled by a family of kernel functions which include in particular the conventional (linear) inner-product as a special case.

We argue that the incentive for designing deep networks with a higher abstraction level than ConvNets, arises from the need for small networks that could fit into mobile platforms in terms of space and run-time. With small networks the approximation error becomes a limiting factor, which could be ameliorated through network architectures that are based on a higher level of abstraction.

The SimNet architecture is based on two operators. The first is analogous to, and generalizes, the inner-product operator of neural networks. The second, as special cases, plays the role of non-linear activation and pooling, but has additional capabilities that take SimNets far beyond ConvNets. In a detailed set of experiments, the SimNet architecture achieves state of the art accuracy using networks with complexity comparable to that of top performing ConvNets. However, when network complexity is limited, SimNets deliver a significant boost in accuracy.

Recently, the task of reducing run-time complexity of ConvNets is receiving increased attention. For example, a method named FitNets ([29]), based on the knowledge distillation principle ([18]), has been suggested in order to assist in compressing deep networks. In [34], a form of gating inspired by Long Short-Term Memory recurrent networks is introduced, allowing training of very deep and narrow net-

Conference on Computer Vision and Pattern Recognition (CVPR), June 2016

works. Another line of work considers imposing structural constrains on network weights, such as sparsity, in order to improve run-time efficiency ([11, 9, 16, 3, 4]). Alternatively, network weights may be factorized using matrix or tensor decompositions, reducing storage and computational complexity, at the expense of marginal deterioration in accuracy ([10, 20, 39, 24, 28, 38, 5]). All of these approaches consider ConvNets (or neural networks) as a baseline, and use supplementary techniques to reduce run-time complexity. In this work, we propose the alternative (generalized) SimNet architecture, and argue that it is inherently more efficient than ConvNets. The techniques listed here for reducing run-time complexity of ConvNets could just as well be applied to SimNets, thereby resulting in even more computationally efficient models.

2. The SimNet architecture

A feed-forward fully-connected neural network, also known as a multilayer perceptron (MLP), is based on a single operator. Given $\mathbf{x} \in \mathbb{R}^d$ as input to a layer of neurons, the output of the r'th neuron in the layer is $\sigma(\mathbf{w}_r^\top \mathbf{x} + b_r)$, where $\sigma(\cdot)$ is a non-linear activation function. An MLP is constructed by forward chaining the input/output operation to create a layered network. The learned parameters of the network are the weight vectors $\mathbf{w}_r$ and biases b_r, per neuron.

The SimNet architecture consists of two operators. The first operator is a weighted similarity function between an input $\mathbf{x} \in \mathbb{R}^d$ and a template $\mathbf{z} \in \mathbb{R}^d$:

$$\text{similarity operator}: \ \mathbf{u}^\top \phi(\mathbf{x}, \mathbf{z})$$

where $\mathbf{u} \in \mathbb{R}^d_+$ is a weight vector and $\phi : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ is a point-wise similarity mapping. We consider two forms of similarity mappings: the "linear" form $\phi_{lin}(\mathbf{x}, \mathbf{z})_i = x_i z_i$, and the "$\ell_p$" form $\phi_{\ell_p}(\mathbf{x}, \mathbf{z})_i = -|x_i - z_i|^p$ defined for $p > 0$. Note that when setting $\mathbf{u} = \mathbf{1}$, the corresponding similarities reduce to inner-product and p-distance (by the power of p) respectively. Note also that unlike the MLP operator, the similarity does not include a bias term. This functionality is covered, in a much more general sense, by the second operator described below.

For the second SimNet operator we define *MEX* – a log-mean-exp function:

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

The parameter $\beta \in \mathbb{R}$ spans a continuum between maximum ($\beta \to +\infty$), average ($\beta \to 0$) and minimum ($\beta \to -\infty$), and for a fixed value of β the function is smooth and exhibits the following "collapsing" property [1]:

$$MEX_\beta\{MEX_\beta\{c_{ij}\}_{1\le j\le m}\}_{1\le i\le n} = MEX_\beta\{c_{ij}\}_{1\le j\le m, 1\le i\le n}$$

Given the definition in eqn. 1, the second SimNet operator consists of taking MEX over an input $\mathbf{x} \in \mathbb{R}^d$ with a bias vector $\mathbf{b} \in \mathbb{R}^d$ – one per input coordinate [2]:

$$\text{MEX operator}: \ MEX_{\beta>0}\{x_i + b_i\}_{i=1,..,d}$$

Note that unlike a conventional MLP unit which has a bias scalar, a MEX unit has a vector of biases. We may choose to omit part or all of the biases as part of a network design. For example, when all biases are dropped the MEX operator implements a soft trade-off between maximum and average.

3. SimNet MLP

A SimNet analogy of an MLP with a single hidden layer is obtained by applying the two operators defined in sec. 2 one after the other – similarity followed by MEX. The resulting network is illustrated in fig. 1(a). It includes n hidden similarity units corresponding to weighted templates $\{(\mathbf{z}_l, \mathbf{u}_l)\}_{l=1}^n$, and k output MEX units associated with bias vectors $\{\mathbf{b}_r\}_{r=1}^k$. Denote by $h_r(\mathbf{x})$ the value of the r'th output unit when the network is fed with input $\mathbf{x} \in \mathbb{R}^d$: $h_r(\mathbf{x}) := MEX_\beta\{\mathbf{u}_l^\top \phi(\mathbf{x}, \mathbf{z}_l) + b_{rl}\}_{l=1}^n$ [3]. As a classifier of $\mathbf{x}$ into one of k categories, the network predicts the label r for which $h_r(\mathbf{x})$ is maximal:

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

As it turns out, SimNet MLP is closely related to kernel machines. In particular, with linear similarity, i.e. with the

[1] The collapsing property, as well as smoothly generalizing maximum and average, will prove to be essential for us. We are not aware of other functions that meet these three requirements. Specifically, the common softmax function $\frac{1}{\beta} \log \left(\sum_i \exp\{\beta \cdot c_i\} \right)$ collapses and generalizes maximum but does not generalize average, and the alternative softmax function $\sum_i c_i e^{\beta c_i} / \sum_i e^{\beta c_i}$ generalizes maximum and average but does not collapse.

[2] The MEX operator can be viewed as an "inner-product in log-space". More accurately, if $\mathbf{x}$ and $\mathbf{b}$ are log-space representations of two vectors $\mathbf{c}$ and $\mathbf{d}$ respectively (i.e. $x_i = \log c_i$ and $b_i = \log d_i$), then $MEX_{\beta=1}\{x_i + b_i\}_i = \log \langle \mathbf{c}, \mathbf{d} \rangle - \log d$. In words, the MEX operator (with $\beta = 1$) taken over the log-space representations of $\mathbf{c}$ and $\mathbf{d}$ is equal (up to an additive constant) to the log-space representation of their inner-product.

[3] Note that with uniform weights ($\mathbf{u}_l \equiv \mathbf{1}$), linear similarity mapping ϕ and $\beta \to +\infty$ we have $h_r(\mathbf{x}) = \max \left\{ \mathbf{z}_l^\top \mathbf{x} + b_{rl} \right\}_{l=1}^n$, i.e. the network outputs are "maxout" units ([13]). SimNet MLP is not the first to generalize maxout. Other generalizations have been suggested, notably the recently proposed L_p unit ([15]), which is defined by $(\sum_l |\mathbf{z}_l^\top \mathbf{x} + b_{rl}|^p)^{1/p}$, and tends to $\max_l \left\{ |\mathbf{z}_l^\top \mathbf{x} + b_{rl}| \right\}$ as $p \to +\infty$. The differences between SimNet MLP and L_p unit as maxout generalizations are: (i) L_p unit generalizes maximum of absolute values which only coincides with maxout if the arguments are non-negative, and (ii) L_p unit tries to realize maxout with a single operator whereas SimNet MLP implements maxout with a succession of two operators.

inner-product operator on which neural networks are based, it is a support vector machine (SVM) based on the Exponential kernel. Replacing the linear similarity with ℓ_p boosts the abstraction level of SimNet MLP, by lifting it to a Generalized Multiple Kernel Learning (GMKL, [37]) engine with a Generalized Gaussian kernel. The remainder of this section provides the details.

SimNet MLP outputs can be written as:

$$\begin{aligned} h_r(\mathbf{x}) &= MEX_\beta\{\mathbf{u}_l^\top \phi(\mathbf{x},\mathbf{z}_l)+b_{rl}\}_{l=1}^n \\ &= \frac{1}{\beta}\ln\left(\frac{1}{n}\sum_{l=1}^n \alpha_{rl}\exp\left\{\beta\sum_{i=1}^d u_{l,i}\phi(\mathbf{x},\mathbf{z}_l)_i\right\}\right) \\ &= \sigma\left(\sum_{l=1}^n \alpha_{rl}\cdot K_\theta(\mathbf{x},\mathbf{z}_l)\right) \end{aligned}$$

where $\alpha_{rl} := \exp\{\beta b_{rl}\}$, $\theta = (\phi,\mathbf{u})$, and $\sigma(t) = (1/\beta)\ln(t/n)$ is a non-linear activation function. The mapping K_θ for the linear and ℓ_p similarities takes the following forms:

$$\begin{aligned} K_{lin}(\mathbf{x},\mathbf{z}) &= \exp\left\{\beta\mathbf{x}^\top\mathbf{z}\right\} \\ K_{\ell_p}(\mathbf{x},\mathbf{z}_l) &= \exp\left\{-\beta\sum_{i=1}^d u_{l,i}|x_i-z_{l,i}|^p\right\} \end{aligned}$$

K_{lin} is known as the *Exponential* kernel ([30]), and K_{ℓ_p} is a GMKL. Specifically, fixing uniform weights ($\mathbf{u}_l \equiv \mathbf{1}$) and $p \leq 2$ reduces K_{ℓ_p} to what is known as the *Generalized Gaussian* kernel. For the particular cases $p=2$ and $p=1$ we get the radial basis function (RBF) and Laplacian kernels respectively. When the weights $\mathbf{u}_l$ and/or order p are learned, the exact underlying kernel is selected during training and we amount at a GMKL.

Denoting by ψ_θ a feature mapping associated with K_θ, we get:

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

where $\mathbf{w}_r := \sum_{l=1}^n \alpha_{rl}\psi_\theta(\mathbf{z}_l)$ is a learned vector in feature space. We thus conclude that SimNet MLP output units are "neurons in feature space", where the space corresponds to the Exponential kernel in the case of linear similarity, and to the Generalized Gaussian kernel in the case of ℓ_p similarity with fixed weights $\mathbf{u}_l$ and order p. When the weights and/or order are learned, the feature space is selected during training, which is equivalent to saying that SimNet MLP is a GMKL.

One may ask if perhaps a different choice of kernel, more elaborate than Generalized Gaussian, will suffice in order to capture SimNet MLP with ℓ_p similarity and learned weights as a simple kernel machine. Apparently, as theorem 1 (proven in [8]) shows, such a kernel does not exist, i.e. a GMKL is indeed necessary in order to represent SimNet MLP in all its glory.

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

4. Deep SimNets for processing images

In the previous section we presented the basic MLP version of SimNets. In this section we describe two (orthogonal) directions of extension. The first is the addition of locality, sharing and pooling for processing images (SimNet MLPConv, sec. 4.1), while the second focuses on deepening the network (adding layers) for enhanced capacity (sec. 4.3). In this context we introduce a "whitened" ℓ_p similarity layer through a succession of a convolution (linear similarity) followed by ℓ_p similarity with receptive field 1×1.

4.1. SimNet MLPConv

The extension of SimNet MLP for processing images follows the line of the MLPConv structure suggested in [26], and we accordingly refer to it as SimNet MLPConv. In particular, [26] convolved a standard MLP across an incoming 3D array by successively applying it to patches and stacking the outputs in a spatially coherent manner. This results in a bank of feature maps, which may be summarized into prediction scores through global average pooling. SimNet MLPConv follows the same principles – a SimNet MLP is convolved across an incoming 3D array, and the resulting feature maps are summarized via global MEX pooling. An illustration of SimNet MLPConv is provided in fig. 1(c). In the figure, $\mathbf{x}_{ij} \in \mathbb{R}^{hwD}$ refers to the input patch in location ij, $\mathbf{z}_l \in \mathbb{R}^{hwD}$ and $\mathbf{u}_l \in \mathbb{R}^{hwD}_+$ denote similarity templates and weights respectively, $\phi : \mathbb{R}^{hwD}\times\mathbb{R}^{hwD} \to \mathbb{R}^{hwD}$ is the similarity mapping (linear or ℓ_p), $\beta_1 \in \mathbb{R}$ and $b_{rl} \in \mathbb{R}$ are the MEX parameter and offsets of the underlying SimNet MLP, and $\beta_2 \in \mathbb{R}$ is the MEX parameter of the final global pooling layer.

When used to classify images, the prediction rule associated with SimNet MLPConv is given by: $\hat{y}(input) = \operatorname{argmax}_r MEX_{\beta_2}\left\{MEX_{\beta_1}\left\{\mathbf{u}_l^\top\phi(\mathbf{x}_{ij},\mathbf{z}_l)+b_{r,l}\right\}_l\right\}_{i,j}$. Setting $\beta_1=\beta_2=\beta$, and using the collapsing property of MEX, we get a "patch-based" version of SimNet MLP's classification:

$$\hat{y}(input) = \operatorname*{argmax}_r \underset{i,j,l}{MEX_\beta}\{\mathbf{u}_l^\top\phi(\mathbf{x}_{ij},\mathbf{z}_l)+b_{r,l}\}$$

It can be shown ([8]) that all results put forth in sec. 3 for relating SimNet MLP to kernel machines apply to SimNet MLPConv as well, but with the underlying kernels being based on "patch-representations". In other words, SimNet

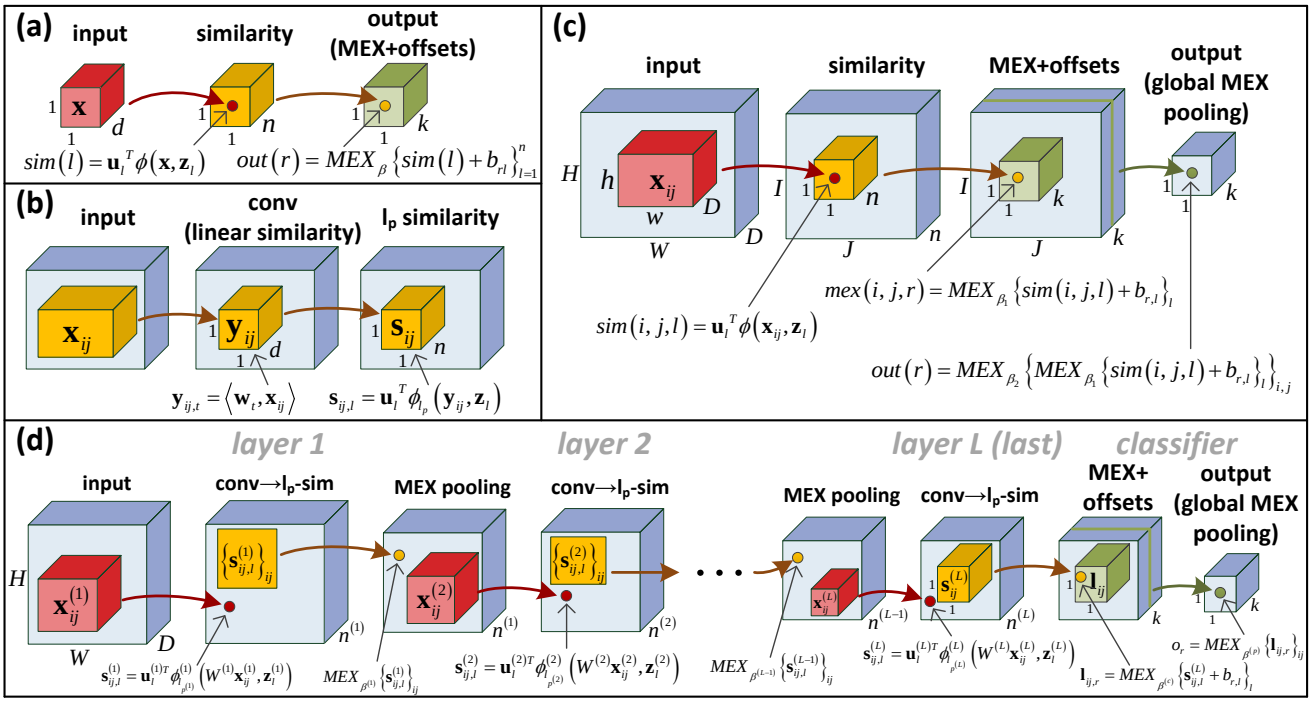

Figure 1. (a) SimNet MLP – SimNet analogy of MLP with single hidden layer (sec. 3) (b) conv→ ℓ_p-sim structure – implements whitened ℓ_p similarity (sec. 4.2) (c) SimNet MLPConv – single layer SimNet for processing images (sec. 4.1) (d) L-layer SimNet for processing images (sec. 4.3). Best viewed in color.

MLPConv – a "patch-based" extension of SimNet MLP, maintains all kernel relations of the latter, with a "patch-based" extension of the underlying kernels.

4.2. Whitening with convolutional layer

We now describe a simple yet powerful addition to the ℓ_p similarity operator. Recall that the ℓ_p similarity between an input $\mathbf{x} \in \mathbb{R}^d$ and a template $\mathbf{z} \in \mathbb{R}^d$ with weights $\mathbf{u} \in \mathbb{R}^d_+$, is defined by $-\sum_{i=1}^d u_i |x_i - z_i|^p$. Up to a constant that depends on $\mathbf{u}$ (and p), this is equal to the log probability density of the input $\mathbf{x}$ being drawn from a Generalized Gaussian distribution with *independent components*, shape p, mean $\mathbf{z}$, and scales $\mathbf{u}^{-1/p}$. These ideas are further developed in sec. 5, however it is clear at this point that in order to capture this probabilistic model, it would be desirable for the input $\mathbf{x}$ to have statistically independent coordinates. Common practice in such cases is to seek for a matrix W for which the linearly transformed vector $W\mathbf{x}$ has independent coordinates. This is referred to in the literature as ICA – independent component analysis ([19]). Assuming such a matrix is found, it would then be natural to "whiten" inputs, i.e. multiply them by W, before measuring their ℓ_p similarities to weighted templates. Besides better compliance with the coordinate independence assumption, this also gives rise to the possibility of dimensionality reduction. In particular, we may set the matrix W to cancel-out low-variance principal components of $\mathbf{x}$, thereby producing whitened vectors of a lower dimension. This can be useful for both noise reduction and computational efficiency.

In the context of SimNet MLPConv, adding support for whitening before ℓ_p similarity is simple – it merely requires a convolutional layer (linear similarity) followed by an ℓ_p similarity layer with receptive field 1×1. Such a construct, which we refer to as conv→ ℓ_p-sim, is illustrated in fig. 1(b). In this figure, input patches $\mathbf{x}_{ij}$ are transformed into d-dimensional vectors $\mathbf{y}_{ij}$ by a convolutional layer with d filters $\mathbf{w}_t$ that hold the rows of the whitening matrix W. The whitened vectors $\mathbf{y}_{ij}$ are then matched against n weighted templates in the ℓ_p similarity layer, producing n similarity maps as output. To recap, one may add whitening to ℓ_p similarity by replacing the similarity layer with a conv→ ℓ_p-sim structure, which consists of convolution followed by 1×1 similarity.

In sec. 5 we describe how to pre-train a conv→ ℓ_p-sim structure, and in particular how to initialize the filters so that they perform the whitening transformation they are intended for. Before that however, we show how SimNet MLPConv can be extended into an image processing SimNet of arbitrary depth.

4.3. Going deep with SimNet MLPConv

After laying out the basic SimNet construct (SimNet MLP – sec. 3), equipping it with spatial structure (SimNet MLPConv – sec. 4.1), and adding whitening to its ℓ_p similarity (conv→ ℓ_p-sim – sec. 4.2), we are finally in a position

to define an arbitrarily deep SimNet for processing images. Our starting point is SimNet MLPConv with whitened ℓ_p similarity. This network accounts for a single layer (conv$\rightarrow$ ℓ_p-sim) followed by a classifier (classification MEX and global MEX pooling). Adding depth to the network simply amounts to appending preceding conv$\rightarrow$ ℓ_p-sim layers, optionally separated by MEX pooling. A general L-layer SimNet following this architectural prescription is illustrated in fig. 1(d). In this structure, conv$\rightarrow$ ℓ_p-sim layers measure whitened ℓ_p similarities of incoming patches to weighted templates, MEX pooling operations summarize spatial regions in similarity maps by MEX'ing them together (note that both average pooling and max pooling are special cases of this), the MEX classification uses its offsets b_{rl} to classify each location in the final similarity maps, and the final global MEX pooling summarizes the local classifications into global class scores. The parameters that may be learned during training are: $W^{(1)}...W^{(L)}$ – linear filters in conv$\rightarrow$ ℓ_p-sim; $\mathbf{z}_l^{(1)}...\mathbf{z}_l^{(L)}$ and $\mathbf{u}_l^{(1)}...\mathbf{u}_l^{(L)}$ – similarity templates and weights in conv$\rightarrow$ ℓ_p-sim; $p^{(1)}...p^{(L)}$ – similarity orders in conv$\rightarrow$ ℓ_p-sim; $\beta^{(1)}...\beta^{(L)}$ – MEX parameters in local pooling; $\beta^{(c)}$ – MEX parameter in classification; b_{rl} – MEX offsets in classification; $\beta^{(p)}$ – MEX parameter in global pooling. In the following section we describe methods for initializing these parameters prior to training (pre-training).

5. Pre-training

In this section we briefly describe a method for pre-training an L-layer SimNet as illustrated in fig. 1(d). Our initialization scheme covers the parameters of conv$\rightarrow$ ℓ_p-sim layers (linear filters $W^{(1)},...,W^{(L)}$, similarity templates $\mathbf{z}_l^{(1)},...,\mathbf{z}_l^{(L)}$, weights $\mathbf{u}_l^{(1)},...,\mathbf{u}_l^{(L)}$ and orders $p^{(1)},...,p^{(L)}$), assuming predetermined local MEX pooling parameters ($\beta^{(1)},...,\beta^{(L)}$). Two attractive properties of the scheme are: (i) it is unsupervised (does not require any labels), and (ii) it gives rise to automatic selection of the number of channels in the convolutions and similarities of conv$\rightarrow$ ℓ_p-sim layers.

The initialization is applied layer by layer in a forward sweep, thus in order for it to be defined, it suffices to consider a single conv$\rightarrow$ ℓ_p-sim layer (fig. 1(b)). Recall from sec. 4.2 that we interpret the convolution in conv$\rightarrow$ ℓ_p-sim as a linear transformation that whitens (and possibly reduces the dimension of) input patches prior to similarity measurements. Accordingly, we initialize its filters $\mathbf{w}_1,...,\mathbf{w}_d$ as the rows of a whitening matrix W estimated via ICA ([19]) on patches.

Turning to the initialization of similarity templates ($\mathbf{z}_1,...\mathbf{z}_n$), weights ($\mathbf{u}_1,...,\mathbf{u}_n$) and order ($p$), we recall that an ℓ_p similarity between an input $\mathbf{y} \in \mathbb{R}^d$ and a template $\mathbf{z} \in \mathbb{R}^d$ with weights $\mathbf{u} \in \mathbb{R}^d_+$, is defined to be $-\sum_{t=1}^d u_t|y_t - z_t|^p$. Consider now a probability distribution over $\mathbb{R}^d$ defined by a mixture of n Generalized Gaussians (with priors $\lambda_l \geq 0, \sum_l \lambda_l = 1$), all having the same shape parameter ($\beta > 0$), and each having independent coordinates with separate scales and means ($\alpha_{l,t} > 0$ and $\mu_{l,t} \in \mathbb{R}$ respectively, for coordinate t of component l):

$$P(\mathbf{y}) = \sum_{l=1}^{n} \lambda_l \prod_{t=1}^{d} \frac{\beta}{2\alpha_{l,t}\Gamma(1/\beta)} e^{-(|y_t-\mu_{l,t}|/\alpha_{l,t})^\beta}$$

The log probability density of a vector drawn from this distribution being equal to $\mathbf{y}$ and originating from component l is: $\log P(\mathbf{y} \wedge \text{comp. } l) = -\sum_{t=1}^d \alpha_{l,t}^{-\beta}|y_t - \mu_{l,t}|^\beta + c_l$, where $c_l := \log\left\{\lambda_l \prod_{t=1}^d \frac{\beta}{2\alpha_{l,t}\Gamma(1/\beta)}\right\}$ is a constant that does not depend on $\mathbf{y}$. This implies that if we model whitened patches $\mathbf{y}_{ij}$ with a Generalized Gaussian mixture as above, initializing the similarity templates via $z_{l,t} = \mu_{l,t}$, the weights via $u_{l,t} = \alpha_{l,t}^{-\beta}$ and the order via $p = \beta$ would give:

$$\mathbf{u}_l^\top \phi_{\ell_p}(\mathbf{y}_{ij}, \mathbf{z}_l) = \log P(\mathbf{y} \wedge \text{comp. } l) - c_l$$

In words, similarity channel l would hold, up to a constant, the probabilistic heat map of component l and the whitened patches $\mathbf{y}_{ij}$. This observation suggests estimating the parameters of the mixture (shape β, scales $\alpha_{l,t}$ and means $\mu_{l,t}$) based on whitened patches (via EM, cf. [1]), and initializing the similarity parameters accordingly. We note in passing that it is possible to append additive biases b_l to the similarity (through offsets of the succeeding MEX operator), in which case initializing these via $b_l = c_l$ would make the probabilistic heat maps exact (not up to a constant).

Finally, as stated above, the initialization scheme presented induces an automatic selection of the number of convolution and similarity channels in conv$\rightarrow$ ℓ_p-sim. The number of convolution channels corresponds to the dimension to which input patches are reduced during whitening, thus may be set via methods for estimating effective dimensionality of data (e.g. [31]). Similarity channels correspond to components in the mixture estimated for whitened patches, thus may be set via methods for estimating the number of components in a mixture (e.g. [2]).

6. Experiments

To evaluate the effectiveness of SimNets, we compared them against alternative ConvNets in three experiments of increasing complexity. In the first experiment, we ran a single layer SimNet against an equivalent single layer ConvNet, and studied the effect of model size (number of convolution/similarity channels) on the accuracy of the two networks. In a second experiment, we compared compact two layer SimNets against the best performing publicly available ConvNet we are aware of that has comparable complexity. In the third and final experiment, we constructed a

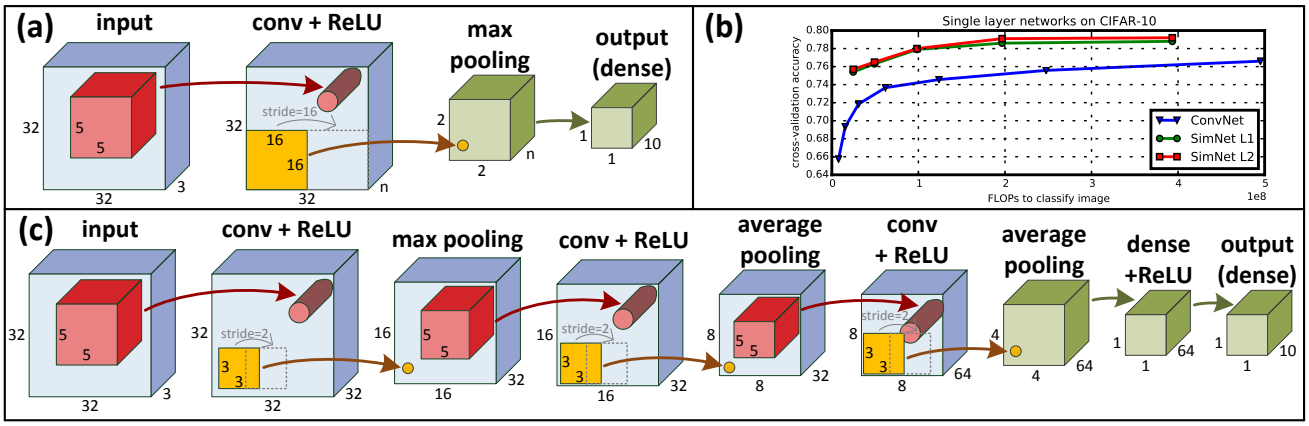

Figure 2. (a) Single layer ConvNet compared against single layer SimNet on CIFAR-10 (b) CIFAR-10 cross-validation accuracies of single-layer networks as a function of the number of floating-point operations required to classify an instance (c) Caffe ConvNet compared against two layer SimNet on CIFAR-10 and SVHN (for CIFAR-100, number of output units increased from 10 to 100). Best viewed in color.

large three layer SimNet designed to compete against state of the art ConvNets. Our experiments demonstrate that SimNets are significantly more accurate than ConvNets when networks are constrained to be compact, i.e. when computational load at run-time is limited. This complies with our theoretical analysis in sec. 3, which shows that weighted ℓ_p similarity exhibits an expressive power that goes beyond kernel machines, whereas linear similarity (the case associated with ConvNets) is fully captured by the Exponential kernel. Asymptotically as the dimension increases, even a simple kernel machine becomes expressive enough for a given problem, and more elaborate expressiveness may actually be a burden, as it aggravates overfitting. Nonetheless, we see in our experiments that with proper regularization, large-scale SimNets achieve accuracies comparable to state of the art ConvNets.

6.1. Experimental details

The datasets used in our experiments are CIFAR-10 and CIFAR-100 ([22]), as well as SVHN ([27]). These three datasets together form an image recognition benchmark that is diverse and challenging on one hand, yet simple enough to enable granular controlled experiments such as those needed to evaluate a new architecture. All datasets consist of 32x32 color images. SVHN (Street View House Numbers) represents a rather simple classification benchmark, where various methods are known to produce near-human accuracies. It contains approximately 600K images for training and 26K images for testing, partitioned into 10 categories that correspond to the digits 0 through 9. CIFAR-100 contains 50K images for training and 10K images for testing, equally partitioned into 100 categories. With a relatively large number of categories, and only a few hundred training examples per class, CIFAR-100 represents a challenging classification task. CIFAR-10 contains 50K images for training and 10K images for testing, equally partitioned into 10 categories. It brings forth a balanced trade-off between the simplicity of SVHN and the complexity of CIFAR-100, and accordingly served as the central dataset throughout our experiments. Namely, all cross-validations were carried out on CIFAR-10 (with 10K training images held out for validation), with SVHN and CIFAR-100 used for final evaluation only. In terms of implementation, we have integrated SimNets into Caffe toolbox ([21]), with the aim of making our code publicly available in the near future.

In all our experiments, we trained both SimNets and ConvNets by minimizing softmax loss using SGD with Nesterov acceleration ([35]). Batch size, momentum, weight decay and learning rate were chosen through cross-validation, though we observed, at least for the case of SimNets, that the following choices consistently produced good results: batch size 128, momentum 0.9, weight decay 0.0001 and learning rate 0.01 decreasing by a factor of 10 after 200 and 250 epochs (out of 300 total). Unlike ConvNets which are mostly initialized randomly nowadays ([23]), SimNets are naturally pre-trained using statistical estimation methods (sec. 5). For computational efficiency, we implemented stochastic versions of these algorithms. Unless otherwise stated, all reported SimNet results were obtained using its pre-training scheme.

6.2. Single layer SimNet

As an initial experiment we compared a single layer SimNet, i.e. a SimNet MLPConv with whitened ℓ_p similarity (conv$\rightarrow \ell_p$-sim), to an equivalent single layer ConvNet defined for this purpose. We chose to design the ConvNet in accordance with the prescription given by Coates et al. in their study of single layer networks ([7]). The resulting network is illustrated in fig. 2(a). As can be seen, it includes a single convolutional layer with 5x5 receptive field and ReLU activation, followed by max pooling over quadrants and dense linear classification. To align the SimNet with

this structure, we applied the whitened similarity to patches with spatial size 5x5, and since these have relatively low dimension already (75), we did not reduce it further during whitening.

To compare the networks as they vary in size (and run-time complexity), we set the number of convolution/similarity channels (denoted n in fig. 2(a) and fig. 1(c)) to 50, 100, 200, 400 and 800. Since the ConvNet requires less computations for a given number of channels, we also tried it with 1600 and 3200 channels. CIFAR-10 cross-validation accuracies produced by the ConvNet, the SimNet with ℓ_1 similarity, and the SimNet with ℓ_2 similarity, are plotted in fig. 2(b) against the number of FLOPs (floating-point operations) required to classify an image [4] [5]. As can be seen, for a given computational budget, the accuracies of ℓ_1 and ℓ_2 SimNets are comparable, whereas the ConvNet falls significantly behind.

6.3. Two layer SimNet

The purpose of this second experiment was to compare SimNets against the best publicly available compact ConvNet we could find. We are interested in a clean SimNet vs. ConvNet architectural comparison, and thus did not include in the experiment model compression techniques such as those listed in sec. 1 (e.g. FitNets [29]), which may be applied to both architectures. An additional reason to exclude these techniques, as well as other works dealing with compact ConvNets (e.g. [12, 40]), is that all results they report relate to networks that are significantly larger than those we are interested in evaluating, in many cases too large to fit a real-time mobile application. With the stated purpose of this experiment being a comparison against an off-the-shelf ConvNet that was not altered by us, we eventually chose to work against the compact CIFAR-10 ConvNet that comes built-in to Caffe, the structure of which is illustrated in fig. 2(c). As the figure shows, the network includes three 5x5 convolutions, each followed by ReLU activation and pooling. Two dense linear layers (separated by ReLU) map the last convolutional layer into network outputs (class scores). The SimNet to which we compared Caffe ConvNet is a two layer network that follows the general structure outlined in fig. 1(d), with ℓ_2 similarity and architectural choices taken to maximize the alignment with Caffe ConvNet: 5x5 receptive field and 32 channels in the first similarity layer, 5x5 receptive field and 64 channels in the second similarity layer, and MEX pooling between the similarities fixed to 3x3 max pooling with stride 2.

Network	Acc. (%)	FLOP	Param.
CIFAR-10			
Caffe ConvNet	81.1	24.8M	145.6K
Two layer SimNet	85.5	14.2M	64.6K
SVHN			
Caffe ConvNet	94	24.8M	145.6K
Two layer SimNet	93.8	14.2M	64.6K
CIFAR-100			
Caffe ConvNet	52.4	24.8M	151.4K
Two layer SimNet	54.6	14.6M	70.3K

Table 1. Two layer SimNet vs. Caffe ConvNet on CIFAR-10, SVHN and CIFAR-100 – comparison of test accuracies, number of floating-point operations required to classify an image, and number of learned parameters.

The networks were initially evaluated on CIFAR-10. Training hyper-parameters for the SimNet were configured via cross-validation, whereas for Caffe ConvNet we used the values that come built-in to Caffe. After measuring CIFAR-10 test accuracies, the same settings (network architectures and training hyper-parameters) were used to evaluate test accuracies on SVHN. For evaluation of test accuracies on CIFAR-100, we again used the exact same settings as in CIFAR-10, but this time increased the number of output channels in both networks from 10 to 100. The results of this experiment are summarized in table 1. As can be seen, the SimNet is roughly twice as efficient as Caffe ConvNet, yet achieves significantly higher accuracies on the more challenging benchmarks (CIFAR-10 and CIFAR-100). On SVHN accuracies are comparable, the reason being that in this simple benchmark classification error is dominated by overfit, to which the enhanced expressiveness of SimNets does not contribute.

6.4. Three layer SimNet

In the previous experiments we have seen that SimNets are more accurate than ConvNets when networks are constrained to be compact, i.e. when classification run-time is limited. In such a setting, the lower approximation error of SimNets plays an important role. In contrast, when networks are over-specified (i.e. are much larger than necessary in order to model the problem at hand) – standard practice for achieving state of the art accuracy, the approximation error is virtually zero, and the advantage of the SimNet architecture fades. Moreover, the additional expressive power of SimNets could actually be a burden, as additional regularization for controlling overfit would be required. It is therefore of interest to explore the ability of SimNets to

[4] In this paper, we consider FLOPs to be a measure of computational complexity. We do not compare actual run-times, as our implementation of SimNets is relatively naïve, not nearly as efficient as the highly optimized ConvNet code that comes built-in to Caffe. One may argue that like Caffe, many other hardware or software platforms are specifically designed for convolutions, and therefore ConvNets have a computational edge over SimNets. While this is true for some off-the-shelf systems, our goal in this paper is to address inherent algorithmic complexities, not specific platforms currently in the market.

[5] To circumvent the computational price of exp and log functions included in SimNets, we used approximations that require up to 10 FLOPs per operation. The resulting degradation in accuracy is marginal.

reach state of the art accuracy with over-specified networks. This is the aim of our third and final experiment, carried out on CIFAR-10.

In this experiment we used a three layer SimNet as described in fig. 1(d), with the following architectural choices (determined via cross-validation): ℓ_2 similarities; 192 similarity channels in all three layers with receptive field sizes 5x5, 5x5 and 3x3 (respectively); max pooling after layer 1, average pooling after layer 2, in both cases pooling windows are 3x3 in size with stride 2 between them. We trained the network with basic data augmentation, and regularized using multiplicative Gaussian noise[6] in conv$\rightarrow \ell_p$-sim layers. We did not make use of ensembles ([6]) or aggressive data augmentation that includes rescaling images ([14]). These practices are known to improve accuracy, but are orthogonal to the SimNet vs. ConvNet distinction. We did not include them in our study in order to facilitate a simpler comparison between the two architectures. Table 2 draws a comparison between the test accuracy reached by the SimNet and reported state of the art results that did not make use of ensembles or aggressive data augmentation. As the table shows, SimNets compare to state of the art ConvNets, even in the over-specified setting.

As a final sanity check, we compared extremely compact versions of our three layer SimNet and Network in Network (NiN, [26]) [7]. Specifically, we changed the number of channels in all layers of both networks to 10, and removed dropout (NiN) and multiplicative Gaussian noise (SimNet), leaving all other hyper-parameters intact. The resulting networks had only 5K parameters each, and required just 3.5M FLOPs to classify an image. With such limited resources we expect the SimNet to benefit from its inherent expressiveness, and indeed, it outperformed NiN significantly, providing 76.8% accuracy compared to 72.3% reached by NiN.

Method	**Acc.** (%)
Network in Network ([26])	91.19
Deeply Supervised Nets ([25])	92.03
Highway Network ([34])	92.4
ALL-CNN ([32])	92.75
Three layer SimNet	92.18

Table 2. Three layer SimNet vs. state of the art ConvNets on CIFAR-10 (ensemble and aggressive data augmentation methods excluded) – comparison of test accuracies.

7. Conclusion

We presented a deep layered architecture called SimNets that generalizes convolutional neural networks. The architecture is driven by two operators: (i) the similarity operator, which is a generalization of the inner-product operator on which ConvNets are based, and (ii) the MEX operator, that can realize non-linear activation and pooling, but has additional capabilities that make SimNets a powerful generalization of ConvNets. An interesting property 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 (sec. 3). This also holds for the more elaborate image processing SimNet incorporating locality, sharing and pooling (sec. 4.1).

The feature spaces realized by SimNets depend on the choice of similarity type: linear or ℓ_p with/without weights. We have shown that the simplest setting using linear similarity (corresponding to regular convolution) realizes the feature space of the Exponential kernel, while ℓ_p settings realize feature spaces of more powerful kernels (Generalized Gaussian, which includes as special cases RBF and Laplacian), or even dynamically learned feature spaces (Generalized Multiple Kernel Learning). These observations suggest that SimNets, when equipped with ℓ_p similarity, have higher abstraction level than ConvNets, which correspond to linear similarity.

We argue that a higher abstraction level for the basic network building blocks carries with it the advantage of obtaining higher accuracies with small networks, an important trait for mobile and real-time applications. Through a detailed set of experiments we validated the conjecture of higher accuracy for small networks, and we have also shown that SimNets can achieve state of the art accuracy in large-scale settings where computational efficiency is not a concern (and thus the higher abstraction per given network size is not an advantage).

Finally, the SimNet architecture is endowed with a natural pre-training scheme based on unlabeled data. Besides its aid in training, the scheme also has the potential of determining the number of channels in hidden layers based on statistical analysis of patterns generated in previous layers. This implies that the structure of SimNets can potentially be determined automatically based on (unlabeled) training data. Future work includes a study of this capability, and more generally, further analysis of probabilistic properties of SimNets and unsupervised/supervised algorithms derived thereof.

Acknowledgments

We thank Ronen Tamari for his dedicated contribution to the experiments. The work is partly funded by Intel grant ICRI-CI 9-2012-6133 and ISF grant 1790/12. Nadav Cohen is supported by a Google Fellowship in Machine Learning.

[6] This regularization technique was shown to be more effective than dropout ([33]), and better suits the nature of SimNets (zeroing out an input coordinate does not neutralize its effect on ℓ_p similarity).

[7] We chose to work against NiN since it bears an architectural resemblance to our SimNet, thus it was clear how both networks can be made compact in an analogous way.

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] Gilles Celeux and Gilda Soromenho. An entropy criterion for assessing the number of clusters in a mixture model. *Journal of classification*, 13(2):195–212, 1996.

[3] Wenlin Chen, James T Wilson, Stephen Tyree, Kilian Q Weinberger, and Yixin Chen. Compressing Neural Networks with the Hashing Trick. In *International Conference on Machine Learning*, 2015.

[4] Wenlin Chen, James T Wilson, Stephen Tyree, Kilian Q Weinberger, and Yixin Chen. Compressing Convolutional Neural Networks. *CoRR abs/1506.04449*, 2015.

[5] Yu Cheng, Felix X Yu, Rogerio S Feris, Sanjiv Kumar, Alok Choudhary, and Shih-Fu Chang. An exploration of parameter redundancy in deep networks with circulant projections. In *International Conference on Computer Vision*, 2015.

[6] Dan Ciresan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649. IEEE, 2012.

[7] 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.

[8] Nadav Cohen and Amnon Shashua. Simnets: A generalization of convolutional networks. *NIPS 2014 Deep Learning Workshop*, 2014.

[9] Maxwell D Collins and Pushmeet Kohli. Memory Bounded Deep Convolutional Networks. *CoRR abs/1412.1442*, 2014.

[10] Emily Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation. In *Advances in Neural Information Processing Systems*, 2014.

[11] Michael Figurnov, Dmitry Vetrov, and Pushmeet Kohli. PerforatedCNNs: Acceleration through Elimination of Redundant Convolutions. *CoRR abs/1202.2745*, cs.CV, 2015.

[12] Chelsea Finn, Lisa Anne Hendricks, and Trevor Darrell. Learning compact convolutional neural networks with nested dropout. *arXiv preprint arXiv:1412.7155*, 2014.

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

[14] Benjamin Graham. Fractional max-pooling. *arXiv preprint arXiv:1412.6071*, 2014.

[15] 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.

[16] Song Han, Jeff Pool, John Tran, and William J Dally. Learning both Weights and Connections for Efficient Neural Networks. *CoRR abs/1202.2745*, cs.NE, 2015.

[17] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. Deepspeech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.

[18] Geoffrey E Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. In *NIPS 2014 Deep Learning Workshop*, 2014.

[19] Aapo Hyvärinen, Juha Karhunen, and Erkki Oja. *Independent component analysis*, volume 46. John Wiley & Sons, 2004.

[20] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up Convolutional Neural Networks with Low Rank Expansions. In *British Machine Vision Conference*, 2014.

[21] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

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

[23] 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.

[24] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan V Oseledets, and Victor S Lempitsky. Speeding-up Convolutional Neural Networks Using Fine-tuned CP-Decomposition. In *International Conference on Learning Representations*, 2015.

[25] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. *arXiv preprint arXiv:1409.5185*, 2014.

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

[27] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 5. Granada, Spain, 2011.

[28] Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, and Dmitry Vetrov. Tensorizing Neural Networks. In *Advances in Neural Information Processing Systems*, 2015.

[29] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. FitNets: Hints for Thin Deep Nets. *CoRR abs/1412.6550*, 2014.

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

[31] Abd-Krim Seghouane and Andrzej Cichocki. Bayesian estimation of the number of principal components. *Signal Processing*, 87(3):562–568, 2007.

[32] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[33] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[34] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training Very Deep Networks. In *Advances in Neural Information Processing Systems*, 2015.

[35] 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.

[36] 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.

[37] Manik Varma and Bodla Rakesh Babu. More generality in efficient multiple kernel learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1065–1072. ACM, 2009.

[38] Zichao Yang, Marcin Moczulski, Misha Denil, Nando de Freitas, Alexander J Smola, Le Song, and Ziyu Wang. Deep Fried Convnets. In *International Conference on Computer Vision*, 2015.

[39] Xiangyu Zhang, Jianhua Zou, Xiang Ming, Kaiming He, and Jian Sun. Efficient and accurate approximations of nonlinear convolutional networks. In *Conference on Computer Vision and Pattern Recognition*. IEEE, 2015.

[40] Zejia Zheng, Zhu Li, Abhishek Nagar, and Woosung Kang. Compact deep convolutional neural networks for image classification.