Co-Occurrence Neural Network

Ira Shevlev
School of Electrical Engineering
Tel-Aviv University
Tel-Aviv, Israel
irenably@post.tau.ac.il

Shai Avidan
School of Electrical Engineering
Tel-Aviv University
Tel-Aviv, Israel
avidan@eng.tau.ac.il

Abstract

Convolutional Neural Networks (CNNs) became a very popular tool for image analysis. Convolutions are fast to compute and easy to store, but they also have some limitations. First, they are shift-invariant and, as a result, they do not adapt to different regions of the image. Second, they have a fixed spatial layout, so small geometric deformations in the layout of a patch will completely change the filter response. For these reasons, we need multiple filters to handle the different parts and variations in the input.

We augment the standard convolutional tools used in CNNs with a new filter that addresses both issues raised above. Our filter combines two terms, a spatial filter and a term that is based on the co-occurrence statistics of input values in the neighborhood. The proposed filter is differentiable and can therefore be packaged as a layer in CNN and trained using back-propagation.

We show how to train the filter as part of the network and report results on several data sets. In particular, we replace a convolutional layer with hundreds of thousands of parameters with a Co-occurrence Layer consisting of only a few hundred parameters with minimal impact on accuracy.

1. Introduction

A common solution to many computer vision problems is based on Convolutional Neural Networks (CNNs). CNNs gained popularity, in part, because they offer a flexible architecture that can be adapted to many different tasks. At their core, CNNs are based on simple primitives that include convolutions, non-linearity and pooling.

A convolutional layer is the most informative layer, because it stores the parameters of the network. But convolutions (i.e., filters) are shift invariant and one needs a large number of filters to deal with different regions of the input. Moreover, filters depend on the spatial layout of the input and are not suitable to deal directly with the distribution of the input values. As a result, a small geometric deformation in the input patch will be considered as a different pattern that requires additional filters to deal with it.

We propose a new filter that addresses these issues. The filter is based on the Co-occurrence Filter (CoF) [10]. CoF combines a spatial filter and a component that is based on the co-occurrence of pixel values. The co-occurrence component will mix (i.e., smooth) pixel values that co-occur frequently in the image plane, while pixel values that do not co-occur frequently will not mix. This makes CoF a boundary preserving filter that can smooth textured regions while preserving the boundaries between them. A unique property of the co-occurrence term is that it depends on pixel values and *not* on the spatial layout of pixels in the image plane.

We take this idea one step further. Instead of *collecting* co-occurrence statistics, as is done in CoF, we *learn* weights based on co-occurrence statistics that optimize the objective function of the network. So if values in the input co-occur often in the training set, we learn the best weight to take advantage of this fact and minimize the loss function of the network. This *deep* co-occurrence matrix differs from a standard co-occurrence matrix in two ways. First, the weights of a co-occurrence matrix are positive, because they describe distributions. In contrast, the weights of the deep co-occurrence matrix can be negative. Second, the co-occurrence matrix is symmetric by definition. This is not necessarily the case for the deep co-occurrence matrix.

The new filter is embedded in a Co-occurrence Layer (CoL) that can replace standard convolutional layers. CoL offers a number of useful properties. To begin with, CoL uses a small number of parameters to generate a large number of filters, depending on the values in a given neighborhood. In addition, the new layer is differentiable and can be trained using back-propagation without any pre- or post-processing. On top of that, CoL can handle different patterns of the input, as well as distributions (i.e., histograms) or re-arrangement of values, because of its co-occurrence term. We show how to train CoL using back-propagation and evaluate its performance on a number of data sets.

2. Background

Almost as soon as neural networks re-gained popularity there has been a surge of interest in ways to speed them up and reduce their memory footprint.

Some early work exploited the linear structure of convolutions for efficient evaluations. This includes, for example, works by Denton *et al.* [4] and Jaderberg *et al.* [8]. The key insight of these works is to approximate the 4D tensor that defines the network using various decomposition techniques.

In Han *et al.* [7] the authors propose a three stage pipeline to compress a network: Pruning, trained quantization and Huffman coding. This leads to impressive reduction in the network size by about $\times 35$ of $\times 49$ depending on the network. But the authors use standard filters to achieve the significant savings and do not offer new filters similar to the one proposed here.

Using new filters in Neural Networks can be found in the work of Wang *et al.* on non-local neural network [17]. In their work, the response at a pixel location is a weighted sum of the features at all locations in its neighborhood. While this approach helps improve the performance of the network, it does not offer a new filter as the one described in this proposal. Another work is that of Cohen *et al.* [3]. They propose a steerable CNN approach where the filter response at each pixel location is a linear combination of some base filters. The emphasis of the work is that transformed versions of the same image region should be treated the same by the network. Instead of taking an axiomatic approach, Weiler *et al.* [18] propose to learn filters that are rotation equivariant. Excellent results on the rotated MNIST dataset and the ISBI 2012 2D EM segmentation challenge are reported.

Jampani *et al.* [9] embed bilateral filters within a neural network. They treat bilateral filters as filters in higher dimensional space and propose to learn the weights of these filters as part of the network training. Recently, SPLATNet [15] used this idea of sparse bilateral convolutional layers for direct processing of point clouds with impressive results.

Battaglia *et al.* [1] proposed an interactive network to learn interaction between objects. This is related to co-occurrences with a couple of important differences. First, we operate at the pixel level and not at the object level. Second, and more importantly, they use existing building blocks to capture the interaction between objects. We, on the other hand, propose a filter that directly captures co-occurrence statistics.

There has been a considerable amount of work at the intersection of edge-preserving filters and CNNs. However, most of this effort is focused on learning the parameters of the edge-preserving filter. They are not designed to introduce the edge-preserving filter into the network.

For example, Wu *et al.* [19] propose a very fast deep version of the guided image filter where the parameters of the filter are learned by the network. They use standard building blocks in their network.

Gharbi *et al.* [6] use pairs of input/output images to train a convolutional neural network to predict the coefficients of a locally-affine model in bilateral space. This lets them perform real-time image enhancement.

Our work is also inspired by the Network in Network approach [12] where the goal is to replace the linear convolution with a micro neural network. However, they use standard building blocks to build the micro neural network.

Cohen *et al.* [2] described an architecture that is driven by two operators: generalization of inner product and a long-mean-exp function. The proposed structure does not deal with statistical context of the input. Moreover, SimNet requires pre-process on the input data while CoL can be just defined as a part of the network without additional effort.

We, as opposed to previous work, propose a novel building block to be used in neural networks.

3. Method

A linear filter is of the form:

$$J_p = \sum_{q \in N_p} w_q I_q \tag{1}$$

where I is the input image, J is the output image, the subscript I_q denotes pixel q in image I, N_p denotes the neighborhood of pixel p and w_q is the weight assigned to pixel q.

In Convolutional Neural Networks (CNNs) the weights w_q of each layer are adjusted so as to optimize the objective function of the network. In this section we define a Co-occurrence Layer (CoL) that is based on a variation of the co-occurrence filter and show how to use it as part of a neural network.

3.1. The Co-occurrence filter

The Co-occurrence filter (CoF) [10] extends the Bilateral Filter (BF) [16] by replacing the range Gaussian G_r with a co-occurrence matrix M. Specifically, a BF is defined as:

$$J_p = \frac{1}{Z} \sum_{q \in N_p} G_s(p,q) G_r(I_p, I_q) I_q \tag{2}$$

where Z is a constant designed to ensure the weights sum to 1. The CoF is defined as:

$$J_p = \frac{1}{Z} \sum_{q \in N_p} G_s(p,q) M(I_p, I_q) I_q \tag{3}$$

where the co-occurrence matrix M is given by:

$$M(a,b) = \frac{C(a,b)}{h(a)h(b)} \tag{4}$$

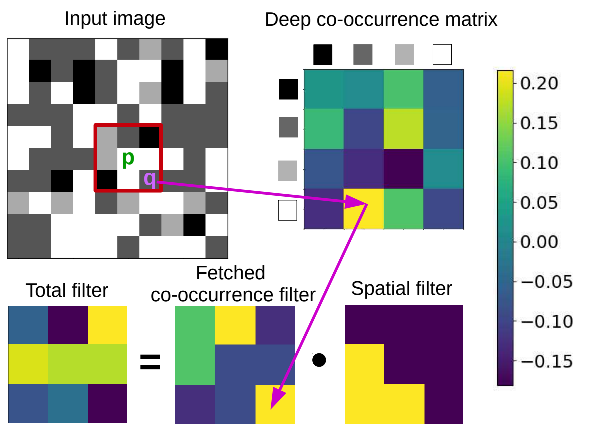


Figure 1. **Applying CoL:** The CoL is based on Equation 7. In the input image for a center pixel p and a neighboring pixel q, we use *deep co-occurrence* matrix entry $L([I_p],[I_q])$ to create a filter based on co-occurrence statistics. Multiplying this filter (element wise) with the spatial filter w leads to the filter that is actually applied at pixel location p. This way different filters are used at different regions of the image.

and

$$C(a,b) = \sum_{p,q} \exp - \frac{d(p,q)^2}{2\sigma^2} [I_p = a][I_q = b] \tag{5}$$

counts the number of times pairs of pixel values co-occur within a window, weighted by their distance. This count is normalized by $h(a), h(b)$ which are the histogram values for pixel values a and b, respectively. The σ is a user specified parameter and $[\cdot]$ is the indicator function.

Intuitively, the CoF mixes pixel values that co-occur frequently, while preserving pixel values that do not co-occur frequently.

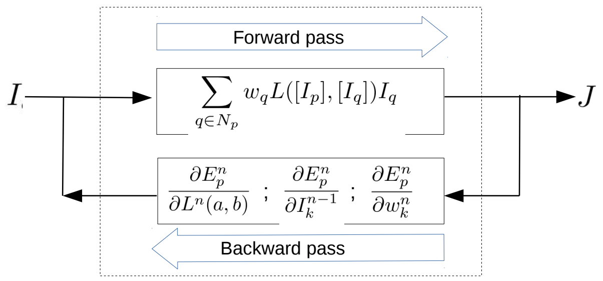
In the case of a gray scale image, the size of the matrix M is conveniently set to 256×256, but if the input is a color image then the co-occurrence matrix becomes a $256^3 \times 256^3$ matrix. This is clearly too large a matrix to work with and the solution proposed in [10] was to quantize the RGB values into, say, 256 color clusters using k-means. The co-occurrence filter can then be written as:

$$J_p = \frac{1}{Z} \sum_{p,q} G_s(p,q) M([I_p],[I_q]) I_q \tag{6}$$

where we denote by $[I_q]$ the index of the cluster to which I_q was assigned.

3.2. The Co-occurrence layer

The co-occurrence matrix M captures the joint probability of observing activation values a and b together. This does not have a direct link to the actual objective function



Forward pass
I
J
$\sum_{q \in N_p} w_q L([I_p],[I_q]) I_q$
$\frac{\partial E_p^n}{\partial L^n(a,b)}$; $\frac{\partial E_p^n}{\partial I_k^{n-1}}$; $\frac{\partial E_p^n}{\partial w_k^n}$
Backward pass

Figure 2. **Training CoL:** In the forward pass we use equation 7 to propagate pixel values from the input layer I, using current w and L. In the backward pass we update the weights of w and L using equations 13, 14 and propagating the total loss using equation 12.

of the network. To this end, we use a deep co-occurrence matrix L that is trained to learn weights that are based on co-occurring activation values. The entries of L are trained to optimize the objective function of the network. With slight abuse of notation, we will define a Co-occurrence Layer (CoL) function as:

$$J_p = CoL(I, w, L([I_p],[I_q])) = \sum_{q \in N_p} w_q L([I_p],[I_q]) I_q \tag{7}$$

where the N_p denotes the neighborhood of activation p.

In words, the value of activation J_p in the output layer is a weighted sum of activation values I_q from the input layer. The weights are defined both by the spatial filter w_q and a matrix $L([I_p],[I_q])$. The implementation details of CoL are illustrated in Figure 1. We quantize the pair of the input activations I_p, I_q and fetch the relevant entry from the *deep co-occurrence* matrix L. Then we multiply the fetched filter with the spatial filter w in order to calculate a total filter for each point. The final step is simply applying the total filter to the input layer at that particular location.

Because the input activation values can take any real value, we quantize them uniformly into k bins. Specifically, we normalize the values of every channel to be in the range $[0,1]$ and then for each $x \in [0,1]$, we define the index of x as $[x] = round(kx)$.

The difference between the co-occurrence matrix M used for CoF and the matrix L used for CoL is the core of this work. The co-occurrence matrix M counts the number of times pixel values co-occur in the input image. The matrix L, on the other hand, learns weights that are designed to optimize the performance of the network.

This difference is also evident in the way CoL and CoF are calculated and applied. In CoF, there is a collection stage in which we scan the image and collect co-occurrence statistics. Once the co-occurrence matrix M is collected, we apply it to the image using equation 6. The situation is completely different with CoL. In the CoL case we start with

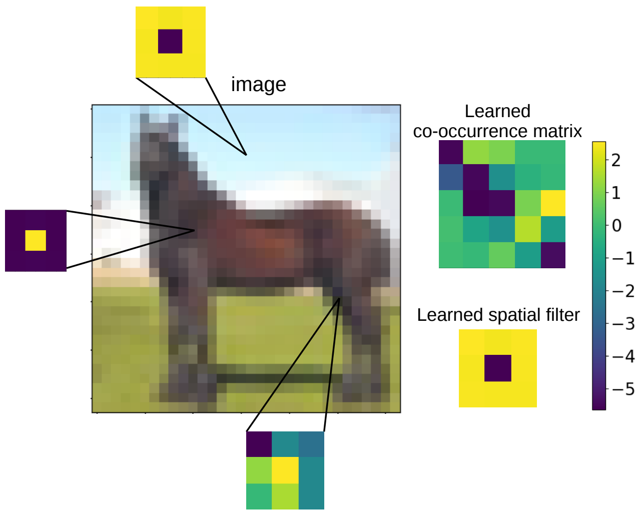


Figure 3. **Applying the CoL filter:** We apply the matrix L and filter w to the image using equation 7. For a center pixel p and a neighboring pixel q, we use matrix entry $L([I_p],[I_q])$ and the weight in the filter w at the corresponding spatial location to obtain the weight of pixel q. This leads to different filters at different regions of the image. In the sky region, the filter remains roughly the same. On the horse, the filter flips sign. Near the rear leg of the horse we have a completely different filter.

a randomly generated matrix L. The forward pass of the training algorithm applies it to the input using equation 7, while the back-prop pass *updates* the weights of the matrix L (see Figure 2). We give the details of the back-prop stage in section 3.3.

Because of the different ways M and L are calculated, they have different properties. The entries of M must be positive because they form a probability distribution function. The entries of L can, and often do, have negative values. Also, by definition, M is symmetric but L is not necessarily so.

The neighborhood $N_p = [N_{x_0}, N_{y_0}, N_{z_0}]$ of CoL can be 2D or 3D depending on N_{z_0}. If $N_{z_0} = 1$ then the spatial support is $2D$ which means that each input channel is processed separately. If $N_{z_0} > 1$ the support is a $3D$ cube that spans several input channels at once.

For k bins and a neighborhood of size $N_p = [N_{x_0}, N_{y_0}, N_{z_0}]$ the total number of parameters in CoL is $k \cdot k + N_{x_0} \cdot N_{y_0} \cdot N_{z_0}$. Such a CoL has the potential to create a $k^{(N_{x_0} \cdot N_{y_0} \cdot N_{z_0})}$ filters, depending on input content. For example, for $k = 5$ bins and a filter of size $3 \times 3 \times 3$ there are a total of 5^{27} different filters that can be generated with as few as 52 parameters, which is the number of parameters needed to store L and w. This, we believe, is one of the reasons that a single CoL equation can replace a typical convolution layer with tens or even hundreds of regular, shift-invariant filters.

Figure 3 illustrates the properties of CoL that were detailed above. We trained a network on CIFAR-10[1] with CoL (with filter sizes $L[5 \times 5]$ and $w[3 \times 3]$) as a first layer, that works directly on raw pixel values. As can be seen, the filter looks different at different locations in the image. Since the spatial filter is shift invariant the differences in the filters are because of the co-occurrence term. For example, the pixel in the sky of the image has a filter that is very similar to w. The reason for this is the values of neighboring pixels are close to the value of the central pixel and quantized to the same bin, so the fetched co-occurrence filter is constant. The sample on the horse also captures a fairly uniform region, only this time the corresponding entry in L has a negative value and, as a result, the w filter is flipped. This example demonstrates that indeed our algorithm can learn negative weights. Near the rear leg of the horse the pixel values are much more diverse, leading to a completely different filter. Finally, as shown in Figure 3, observe that L is not symmetric.

3.3. Backpropagation

Equation 7 is used in the forward pass of the network. We now define its derivatives as required for backpropagation.

The general formulation of the co-occurrence layer is a function of the input layers, convolution, and co-occurrence filter. Specifically, let E_p^n denote the error function of the n-th layer for pixel p, and let J_p^n denote the output of the n-th CoL at pixel p, then:

$$J_p^n = CoL(I^{n-1}, w^n, L^n([I_p^{n-1}],[I_q^{n-1}])) \tag{8}$$

is the forward pass equation. Applying the chain rule we have:

$$\frac{\partial E_p^n}{\partial I_k^{n-1}} = \frac{\partial E_p^n}{\partial J_p^n} \frac{\partial CoL(I^{n-1}, w^n, L^n([I_p^{n-1}],[I_q^{n-1}]))}{\partial I_k^{n-1}} \tag{9}$$

$$\frac{\partial E_p^n}{\partial w_k^n} = \frac{\partial E_p^n}{\partial J_p^n} \frac{\partial CoL(I^{n-1}, w^n, L^n([I_p^{n-1}],[I_q^{n-1}]))}{\partial w_k^n} \tag{10}$$

$$\frac{\partial E_p^n}{\partial L^n(a,b)} = \frac{\partial E_p^n}{\partial J_p^n} \frac{\partial CoL(I^{n-1}, w^n, L^n([I_p^{n-1}],[I_q^{n-1}])}{\partial L^n(a,b)} \tag{11}$$

By substituting the CoL definition, the derivatives are:

$$\frac{\partial E_p^n}{\partial I_k^{n-1}} = \frac{\partial E_p^n}{\partial J_p^n} \cdot w_k^n \cdot L^n([I_p^{n-1}],[I_k^{n-1}]) \tag{12}$$

$$\frac{\partial E_p^n}{\partial w_k^n} = \frac{\partial E_p^n}{\partial J_p^n} \cdot L^n([I_p^{n-1}],[I_k^{n-1}]) \cdot I_k^{n-1} \tag{13}$$

[1]Taken from https://github.com/seansoleyman/cifar10-resnet.

$$\frac{\partial E_p^n}{\partial L_{(a,b)}^n} = \frac{\partial E_p^n}{\partial J_p^n} \cdot \sum_{q \in N_p} w_q^n \cdot I_q^{n-1} \cdot \delta(a,b) \quad (14)$$

Equation 12 expresses the back-propagated error function. Equations 13 and 14 show the derivatives of the spatial filter and the co-occurrence matrix, respectively. The delta function

$$\delta(a,b) = \delta(a = [I_p^{n-1}], b = [I_q^{n-1}]) \quad (15)$$

in equation 14 verifies if the input values are quantized to the relevant indices of the deep co-occurrence matrix L.

3.4. A Toy Example

We illustrate the power of CoL on a toy example and compare its performance with two other popular layers: fully-connected and convolutional. We generated a synthetic set of 6000 training images and 1000 test images of size 10×10 and 4 pixel values that were sampled i.i.d from two different distributions (i.e., histograms). The pixel values of the first distribution came from the histogram $[0.4, 0.1, 0.1, 0.4]$, and the pixel values of the second came from the histogram $[0.1, 0.4, 0.4, 0.1]$. The mean pixel value of both histograms is the same. (See Figure 4(a)). We evaluated a number of network architectures:

1. conv($1 \times 1 \times 9$) $\rightarrow$ avg(9×9) $\rightarrow$ fc(36×2)
2. conv($3 \times 3 \times 2$) $\rightarrow$ avg(7×7) $\rightarrow$ fc(32×2)
3. conv($3 \times 3 \times 9$) $\rightarrow$ avg(9×9) $\rightarrow$ fc(36×2)
4. fc(100×36) $\rightarrow$ fc(36×2)
5. CoL(4×4) $\rightarrow$ avg(5×5) $\rightarrow$ fc(36×2)

All networks use a stride step of 1. The CoL in network 5 only uses a co-occurrence term of size $L[4 \times 4]$ and a spatial filter that is set to one. In the first three architectures, the size of convolutions and average pooling was designed to reduce the image. In all cases, the last fully connected layer is trained to output an estimated distribution of the two classes as the output. See details in Figure 4(b). The average pooling layer is used to decrease the power of the final fully connected layer and still preserve enough information about the input. The CoL network (item (4) above) consists of 16 parameters and the spatial filter is constant 1 of size (10×10).

Figure 4(b) shows the different instances of networks and Figure 4(c) shows the results of this experiment. As can be seen, the CoL-based network, which uses far less parameters compare to a standard convolutional layer , achieves a far lower loss, and does so in a lower number of iterations. This shows that CoL can capture complex visual patterns in a very compact form.

This experiments highlights a number of properties of the CoL. First, it does not depend on the spatial layout of pixel values and can handle distributions of pixel values.

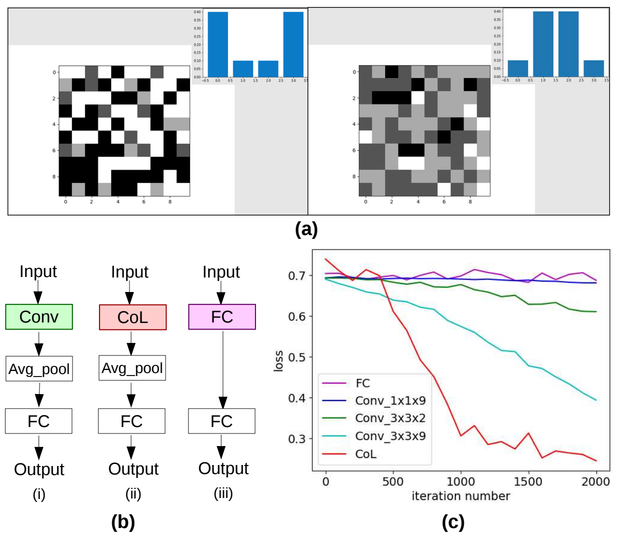


Figure 4. **Toy example: (a)** Two examples of images where the pixels are sampled i.i.d from two different 4-bin histograms. **(b)** Three toy network architectures with different initial layers: (i) the first layer is Conv with different sizes of filter (ii) the first layer is CoL (iii) the network contains a fully-connected layer. **(c)** Graphs of the loss function of the different networks. As can be seen, the CoL architecture converges faster and to a much lower loss. See more details about the different architectures in the text 3.4.

Second, the number of parameters of the co-occurrence matrix does not depend on the spatial support, and a $[4 \times 4]$ deep co-occurrence matrix can be used with a filter with a spatial support of $[10 \times 10]$.

4. Experiments and Results

The code was implemented in TensorFlow and tested on a GPU TITAN X (Pascal) with 12G memory. We tested the proposed layer on two datasets: CIFAR-10 [11] that is used for image classification, and the ADE20k dataset [20] that is used for semantic pixel labeling. In each case, we used an existing, publicly available code.

Implementation details: We use a single CoL filter for all the input data. That is, all input channels contribute to a single co-occurrence term. We found that this gives the best trade-off in terms of computer time and memory requirements. We do believe that further research is required on this topic. To speed up computation, we follow the approach used by Durand and Dorsey [5] to speed up the bilateral filter. Specifically, we loop over the bin values, and for each bin perform a regular shift-invariant convolution, which is fast to compute. As a result, our algorithm is k times slower than a regular CNN both in training and testing (where k is the number of bins we use). We use $k = 5$ in all the experiments reported here. The initializer that was

# stacks	# blocks	layer	size
first convolutional layer			$[3 \times 3 \times 3 \times 32]$
stack(1)	block(1)	conv1	$[3 \times 3 \times 32 \times 32]$
		conv2	$[3 \times 3 \times 32 \times 32]$
	block(2)	conv1	$[3 \times 3 \times 32 \times 32]$
		conv2	$[3 \times 3 \times 32 \times 32]$
stack(2)	block(1)	conv1	$[3 \times 3 \times 32 \times 64]$
		conv2	$[3 \times 3 \times 64 \times 64]$
	block(2)	conv1	$[3 \times 3 \times 64 \times 64]$
		conv2	$[3 \times 3 \times 64 \times 64]$
stack(3)	block(1)	conv1	$[3 \times 3 \times 64 \times 128]$
		conv2	$[3 \times 3 \times 128 \times 128]$
	block(2)	conv1	$[3 \times 3 \times 128 \times 128]$
		conv2	$[3 \times 3 \times 128 \times 128]$

Table 1. the original ResNet that used in CIFAR10 experiments. The table describe all the convolutional layers with sizes: the first convolutional layer changes the number of channels other has the same size per stack.

used in all our experiments is a common truncated normal distribution with a standard deviation of 0.1.

Experiments on CIFAR-10: For the CIFAR-10 dataset, we used the ResNet architecture[2]. The architecture consists of three stacks, each consisting of two blocks. Every block contains batch-normalization - convolution - batch-normalization - convolution. The first convolution maps the input channels to a different number of output channels and, as was explained in section 3, CoL requires that the number of input and output channels be equal. See Table 1.

The results are reported in Figure 5. We compared the performance of networks where we removed all instances of conv2 from a stack (see Table 1) completely or replaced them with CoL. Since there are three stacks we have tried all possible combinations of stacks. The graph shows the test error as a function of compression ratio and emphasizes the gap in test error between different types of blocks. The original block, which is marked with "X" in Figure 5, has the lowest test error, but also the largest number of parameters (its compression ratio is 1 and every other network is compared to it). As can be seen, CoL does not help performance at the early stages of the network and, in some cases, slightly hurts performance. However, using CoL in the third stack consistently reduces the error by about 1%, while cutting the total number of parameters by almost a half.

Ablation test on CoL components: We conducted an ablation test to measure the contribution of the spatial term and the co-occurrence term to the performance of CoL. Specifically, we replaced both conv2 layers of the third stack of the ResNet architecture. Then, we evaluated two

[2]Taken from https://github.com/seansoleyman/cifar10-resnet.

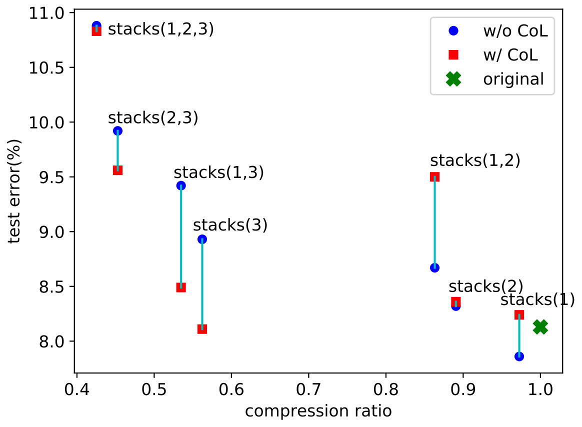


Figure 5. **CIFAR10 test results:** The x-axis denotes the compression ratio of the number parameters and the y-axis denotes the test error. Each point corresponds to a different configuration of the network with and without CoL. "w/o CoL" means dropping the conv2 layer in that stack and "w/ CoL" means replacing the conv2 layer with a CoL. The text indicates which stacks were modified (i.e., "stacks(2,3)" means that we modified the second and third stacks). The result of the original network is marked by 'X'. As can be seen, the modifying of the last stack (i.e., stack number 3) leads to a decrease of between 0.5% to 1% in the error, while *cutting* the number of parameters by approximately half.

variants: in the first, CoL contains only the $L[5 \times 5]$ term, and in the second, CoL only uses $w[3 \times 3 \times 3]$. The test error in both cases was 8.53. Only when we trained with both the spatial and co-occurrence terms did the test error drop to 8.13, which is similar to the error of the original network.

Pruning experiment: In another experiment, we compared CoL with two pruning techniques to see how much can a convolutional layer be compressed using existing techniques. Specifically, we evaluated two methods. The first is a magnitude-based method: prune weights with magnitude less than a chosen threshold. The second method calculates an SVD decomposition of the weights, set to zero a fixed number of eigenvalues and reconstructs the weights with the truncated eigenvalues. This experiment was conducted on the third stack of the network. In both methods we pruned the network to be of size as similar as possible to the size of the network with CoL in the third stack. Results are reported in Table 2. As can be seen, both magnitude and SVD-based pruning give inferior results to CoL.

Distribution of filters: CoL can encode a large number of different filters in a compact representation. We suspect that at the deeper layers of a network the number of different filters captured by CoL is higher than in the early layers, and this might lead to increased representation power that leads to improved performance. To quantify this, we collected the filters used by CoL, clustered them into prototypes us-

Method	Test error(%)
Original network	8.11
Magnitude-based pruning	9.57
SVD-based pruning	8.75
CoL (ours)	**8.13**

Table 2. **Pruning evaluation:** We compare two pruning techniques to CoL. As can be seen, both magnitude and SVD-based pruning give inferior results to CoL. Compression ratio for all methods is about 55%.

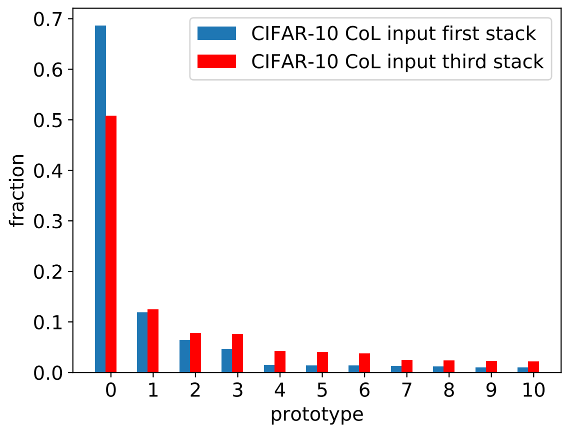


Figure 6. Distribution of filters by prototype (i.e., clusters), in the CIFAR-10 experiment, for the first and third stacks. As can be seen, the first prototype of the first stack accounts for almost 70% of the filters generated by CoL. In contrast, the first prototype of the third stack captures only 50%. See text for more details.

ing k-means, and calculated a histogram of filters. That is, we count, for each prototype (i.e., cluster center) filter, how many similar filters were generated by the CoL layer.

Figure 6 shows the distribution of filter prototypes for the CIFAR-10 experiments. In particular, we collected statistics from the first and third stacks. As can be seen, the first prototype of the first stack accounts for almost 70% of the filters generated by CoL. In contrast, the first prototype of the third stack accounts for only 50%. This shows a correlation between the distribution of filters generated by CoL and its performance. The more uniform the distribution, the higher the performance of CoL.

Experiments on ADE20k: We tested CoL on the MIT Scene Parsing Benchmark that is based on the ADE20k dataset [20]. This dataset contains 22k scene-centric images with 150 semantic categories. The training set consists of 20k images and we report results on the validation set that consists of 2k images. Our network is largely based on the Fully Connected Network (FCN) of [13][3]. The input to the

	Accuracy	Total IoU	Score
1^{st} conv(7×7) (orig.)	62.69	0.1362	0.3815
1^{st} conv(3×3)	62.02	0.1374	0.3788
1^{st} conv(3×3)/w CoL	**64.41**	**0.1618**	**0.4029**
without 2^{nd} conv	62.09	0.1218	0.3713
CoL as 2^{nd} layer	**63.27**	**0.1332**	**0.3829**

Table 3. **ADE20k result:** Results (on validation set) of semantic segmentation network with and without CoL. Reducing the filter size from the original size of (7×7) to (3×3) cuts the number of parameters of this layer by half but hurts performance slightly. Adding a single CoL layer increases performance by about 1%. A similar behaviour is observed for the second layer. See details of different architectures in Table 4.

network is the result of a preprocessing stage that takes each image from the dataset and passes it through VGG19 [14].

We evaluated a number of different architectures, as shown in Table 4. In the first experiment the first convolutinal layer of size $[7 \times 7 \times 512 \times 4096]$ is replaced by a layer with a smaller spatial support of size $[3 \times 3 \times 512 \times 4096]$, followed by a CoL of size $L[5 \times 5]$ and $w[7 \times 7 \times 13]$. As shown in Table 3 we have reduced the number of parameters by a factor of $\frac{9}{49}$ while *increasing* accuracy by 2.4%. In the second experiment we replaced the second convolution with CoL. In this case replacing a convolutional layer of size $[1 \times 1 \times 4096 \times 4096]$ with a CoL of size $L[5 \times 5]$ and $w[7 \times 7 \times 13]$ increased accuracy by 1%.

5. Conclusions

We proposed a new filter that augments regular filters with a term that is based on co-occurrence statistics. The resulting filter is differentiable and can be trained using back-propagation.

The filter is not shift-invariant and as such can adapt to different regions of the input image. In addition, the co-occurrence term lets the filter handle local geometric deformations in the image plane. The filter is defined by a small number of parameters yet can generate many different regular filters, based on input content. This property lets us replace many regular filters with a single co-occurrence based filter.

We defined a new layer, termed Co-Occurrence Layer (CoL), that encapsulates this filter and have shown that it can be used in different places of a network. Finally, experiments on two data sets demonstrate the advantages of our method.

Acknowledgments: Parts of this research were supported by ISF grant 1917/15.

[3]We use the TensorFlow implementation found at https://github.com/shekkizh/FCN.tensorflow.

Original network		
Layer Type	Filter Shape	Input Size
Conv1	$7 \times 7 \times 4096$	$7 \times 7 \times 512$
Conv2	$1 \times 1 \times 4096$	$7 \times 7 \times 4096$
Conv3	$1 \times 1 \times 150$	$7 \times 7 \times 150$

Modified Conv1		
Layer Type	Filter Shape	Input Size
Conv1	$3 \times 3 \times 4096$	$7 \times 7 \times 512$
Conv2	$1 \times 1 \times 4096$	$7 \times 7 \times 4096$
Conv3	$1 \times 1 \times 150$	$7 \times 7 \times 150$

Modified Conv1 + CoL		
Layer Type	Filter Shape	Input Size
Conv1	$3 \times 3 \times 4096$	$7 \times 7 \times 512$
CoL	$L[5 \times 5], w[7 \times 7 \times 13]$	$7 \times 7 \times 4096$
Conv2	$1 \times 1 \times 4096$	$7 \times 7 \times 4096$
Conv3	$1 \times 1 \times 150$	$7 \times 7 \times 150$

W/O Conv2		
Layer Type	Filter Shape	Input Size
Conv1	$3 \times 3 \times 4096$	$7 \times 7 \times 512$
Conv3	$1 \times 1 \times 150$	$7 \times 7 \times 4096$

CoL instead of Conv2		
Layer Type	Filter Shape	Input Size
Conv1	$3 \times 3 \times 4096$	$7 \times 7 \times 512$
CoL	$L[5 \times 5], w[7 \times 7 \times 3]$	$7 \times 7 \times 4096$
Conv3	$1 \times 1 \times 150$	$7 \times 7 \times 4096$

Table 4. **FCN:** The table describes the five different architectures evaluated on ADE20k. See results of different architectures in Table 3.

References

[1] P. Battaglia, R. Pascanu, M. Lai, D. Jimenez Rezende, and k. kavukcuoglu. Interaction networks for learning about objects, relations and physics. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4502–4510. Curran Associates, Inc., 2016.

[2] N. Cohen, O. Sharir, and A. Shashua. Deep simnets. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 4782–4791, 2016.

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

[4] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 1269–1277, 2014.

[5] F. Durand and J. Dorsey. Fast bilateral filtering for the display of high-dynamic-range images. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 257–266. ACM, 2002.

[6] M. Gharbi, J. Chen, J. T. Barron, S. W. Hasinoff, and F. Durand. Deep bilateral learning for real-time image enhancement. *ACM Trans. Graph.*, 36(4):118:1–118:12, 2017.

[7] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149, 2015.

[8] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. In *British Machine Vision Conference, BMVC 2014, Nottingham, UK, September 1-5, 2014*, 2014.

[9] V. Jampani, M. Kiefel, and P. V. Gehler. Learning sparse high dimensional filters: Image filtering, dense crfs and bilateral neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[10] R. J. Jevnisek and S. Avidan. Co-occurrence filter. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 3816–3824, 2017.

[11] A. Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[12] M. Lin, Q. Chen, and S. Yan. Network in network. *CoRR*, abs/1312.4400, 2013.

[13] E. Shelhamer, J. Long, and T. Darrell. Fully convolutional networks for semantic segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(4):640–651, Apr. 2017.

[14] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[15] H. Su, V. Jampani, D. Sun, S. Maji, E. Kalogerakis, M.-H. Yang, and J. Kautz. Splatnet: Sparse lattice networks for point cloud processing. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[16] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Proceedings of the Sixth International Conference on Computer Vision*, ICCV '98, pages 839–, Washington, DC, USA, 1998. IEEE Computer Society.

[17] X. Wang, R. Girshick, A. Gupta, and K. He. Non-local neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[18] M. Weiler, F. A. Hamprecht, and M. Storath. Learning steerable filters for rotation equivariant cnns. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[19] H. Wu, S. Zheng, J. Zhang, and K. Huang. Fast end-to-end trainable guided filter. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[20] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.