Scale-Invariant Convolutional Neural Network

Yichong Xu
Tsinghua University
xycking@163.com

Tianjun Xiao
Peking University
xiaotianjun@pku.edu.cn

Jiaxing Zhang
Microsoft Research Asia
jiaxz@microsoft.com

Kuiyuan Yang
Microsoft Research Asia
kuyang@microsoft.com

Zheng Zhang
NYU Shanghai
zz@nyu.edu

Abstract

Even though convolutional neural networks (CNN) has achieved near-human performance in various computer vision tasks, its ability to tolerate scale variations is limited. The popular practise is making the model bigger first, and then train it with data augmentation using extensive scale-jittering. In this paper, we propose a scale-invariant convolutional neural network (SiCNN), a model designed to incorporate multi-scale feature exaction and classification into the network structure. SiCNN uses a multi-column architecture, with each column focusing on a particular scale. Unlike previous multi-column strategies, these columns share the same set of filter parameters by a scale transformation among them. This design deals with scale variation without blowing up the model size. Experimental results show that SiCNN detects features at various scales, and the classification result exhibits strong robustness against object scale variations.

1. Introduction

Many classical computer vision tasks have enjoyed a great breakthrough, primarily due to the large amount of training data and the application of deep convolution neural networks (CNN) [8]. In the most recent ILSVRC 2014 competition [11], CNN-based solutions have achieved near-human accuracies in image classification, localization and detection tasks [14, 16].

Accompanying this progress are studies trying to understand what CNN has learnt internally and what contribute to its success [2, 13, 17]. By design, layers within the network have progressively larger receptive field sizes, allowing them to learn more complex features. Another key point is the shift-invariance property, that a pattern in the input can be recognized regardless of its position [9]. Pooling layers contribute resilience to slight deformation as well small scale change [12].

However, it is evident that CNN deals with shift-variance far better than scale-invariance [3]. Not dealing with scale-invariance well poses a direct conflict to the design philosophy of CNN, in that higher layers may see and thus captures features of certain plain patterns simply because they are larger at the input, not because they are more complex. In other words, there is no alignments between in the position of a filter and the complexity it captures. What is more, there are other invariance that CNN does not deal with internally. Examples include rotations and flips (since features of natural objects are mostly symmetric).

A brutal force solution would be to make the network larger by introducing more filters to cope with scale variations of the same feature, accompanied by scale-jittering the input images, often by order of magnitude. This is, in fact, the popular practice today [1, 8, 14]. It is true even for proposals that directly deal with this problem. For example, [3] drives the CNN with crops of different size and positions with three differnt scales, and then uses VLAD pooling to produce a feature summary of the patches.

We explore a radically different approach that is also simple. Observing that filters that detect the same pattern but with different scales bear strong relationship, we adopt a multi-column design and designate each column to specialize on certain scales. We call our system SiCNN (Scale-invariant CNN). Unlike a conventional multi-column CNN, filters in SiCNN are strongly regulated among columns. The goal is to make the network resilient to scale variance without blowing up number of free parameters, and thus reduce the need of jittering the input.

We performed detailed analysis and verified that SiCNN exhibits the desired behavior. For example, the column that deals with larger scale is indeed activated by input patterns with the larger scaling factor, and the system as a whole becomes less sensitive to scale variance. On unaugmented CIFAR-10 dataset [7], our method produces the best re-

sult among all previous works using a single CNN and a simple softmax classifier, and is complementary to other techniques that improve the performance. Our model increases training cost linear to number of columns, but we find that incremental refinement can dramatically reduce the cost without compromising the performance significantly.

The rest of the paper is organized as follows. Section 2 presents SiCNN, covering the high-level intuition, the mathematical foundation and the architecture. Detailed analysis and results are presented in Section 3, and we conclude in Section 4.

2. Model

Consider the case of classifying objects that have only one *canonical* scale and the only free parameter is their positions. A stack of convolution filters can progressively build more complex hidden representations. These hidden representations are all *invariant by shift*, meaning that the activations preserve the same pattern except that they are shifted. In other words, $\mathrm{conv}(\mathbf{Shift}(I), f) \equiv \mathbf{Shift}(\mathrm{conv}(I, f))$, for arbitrary image I and filter f, and this relationship is upheld layer to layer. This makes the job of the classifier easy.

In the existing CNN architecture, dealing with multiple scales is jointly achieved by the pooling layers and the convolution layers. The convolution layer not only needs to learn different features but also their scaled variants into multiple feature maps. Units in the pairing pooling layer generate scale-invariance within their receptive fields, which help to save feature maps. This multi-scale solution leads to bigger model and, since filters are independently learned, the need of more training data. The popular practice is scale-jittering [8].

Our idea is simple, and is inspired by the *invariance-by-shift* property of the existing convolution layer. Just like CNN convolve a filter on different positions, we also "convolve" the filter on different scales. This is done by adding independent columns, each is a conventional CNN but "specialized" at detecting one scale. Crucially, the columns are strongly regularized such that the number of free parameters in the convolution layers stay the same. Thus, we inject scale-invariance into the model, requiring neither additional data augmentation nor increasing the model size.

In the followings, we first introduce our architecture, present the intuition and then give the concrete mathematical definition.

2.1. Scale-Invariance Architecture

SiCNN uses multiple columns of convolutional stack with varying filter size to capture objects with unknown scales in input images. The architecture is illustrated in Figure 1. From bottom up, the input image is fed into all the columns. Each column has several convolutional layers
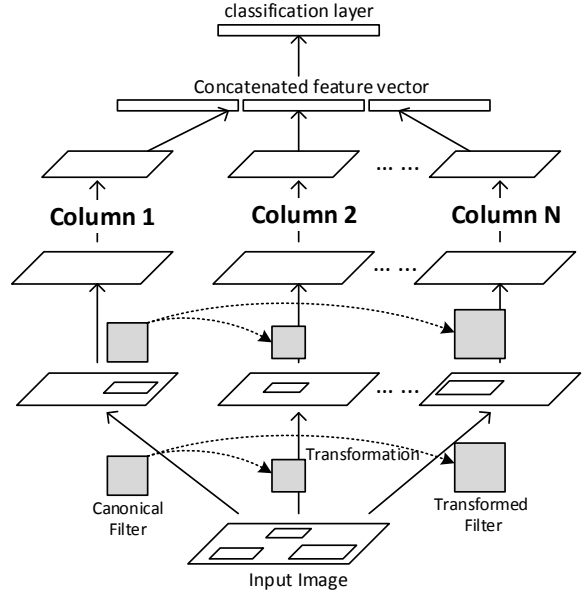

Figure 1. Architecture of SiCNN.

with max-pooling. The key difference from conventional multi-column CNN is that, although these columns use different filter size, they still share a set of common parameters among their filters. A canonical column (Column 1 in Figure 1) keeps canonical filters in each layer. Other columns, which we call *scale* columns, transform these canonical filters into their own filter. Collectively, a canonical filter and its transformed filters detect its pattern at different scales in multiple columns simultaneously. Therefore, a single pattern with different scales trigger one or more columns.

In our architecture, we simply concatenate the top-layer feature maps from all the columns into a feature vector. The final classification layers (a softmax layer in the simplest case) take this feature vector as input.

2.2. Filters in Multiple Scales

Filters that are transformed into different columns from the canonical filter capture the same pattern at different scales. We will discuss this transformation from canonical filters to other columns.

Consider a canonical filter f, which detects a pattern in image I by convolution $\mathrm{conv}(I, f)$ (Figure 2). When the image is scaled by a scaling operation S to $S(I)$, we expect another column k with transformed filter $T(f)$ to capture the same pattern instead. Thus, the column k generates another convolution result $\mathrm{conv}(S(I), T(f))$. Just as invariance-by-shift, we require this convolution be equivalent with scaling from the convolution result in the canonical column. That is

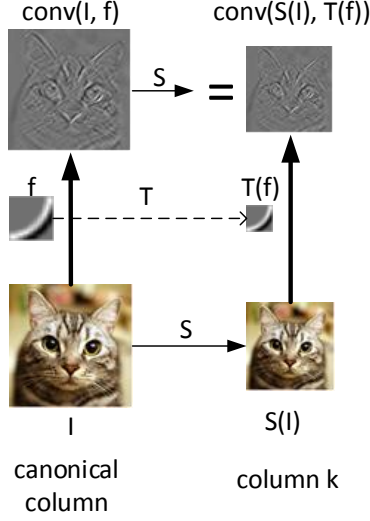$$\mathrm{conv}(S(I), T(f)) = S(\mathrm{conv}(I, f)). \qquad (1)$$

Figure 2. The transformation from a canonical filter to another column. Best viewed in electronic form.

We call this property of filter as *invariance-by-scaling*. Given a scaling S, we want to find the T that satisfies Equation 1 for any image I and filter f.
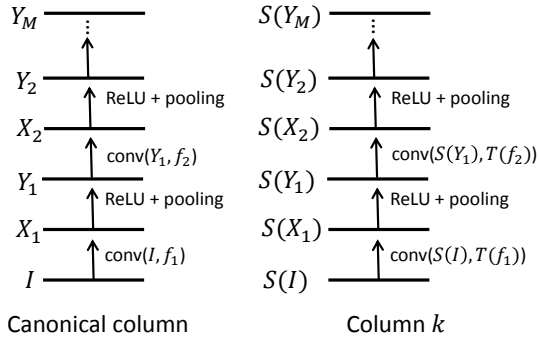

Figure 3. Columns with multiple layers to capture patterns in different scales.

The above discussion is for the first layer. However, it is easy to see that if the filter transformation T in each layer satisfies Equation 1, then invariance-by-scaling property is preserved layer by layer till reaching the classification layer. In Figure 3, when input images I and $S(I)$ with different scale are fed into the canonical column and the column k separately, they generate X_1 and $S(X_1)$ respectively. The following ReLU nonlinear activation function and max-pooling keep this scale relationship and result in Y_1 and $S(Y_1)$. By recursively applying Equation 1, we know the top layers of these two columns also keep the same scale relationship: if canonical column generates Y_M on input image I, the column k generates $S(Y_M)$ on image $S(I)$.

If the object scale fits exactly one of the columns, then there is a perfect matching with the column outputting the highest responses. Otherwise, if the object scale just falls in between the scales of two neighboring columns, both the two columns will have relatively high responses. The concatenated feature vector at the end makes it possible for the classifier to do a linear combinations of responses from multiple columns to eliminate above variance.

2.3. Filter Transformation

By a vector representation of the image (concatenating all the rows or columns in matrix), scaling and convolution are all linear transformations. Given a canonical filter f_c represented by a vector, we can solve the following equation derived from Equation 1 to get the transformed filter f_t,

$$\forall I, \mathrm{conv}(S(I), f_t) = S(\mathrm{conv}(I, f_c)). \qquad (2)$$

Equation 2 is a system of linear equations for f_t. However, such a system doesn't always have a valid solution because it has too many constraints (linear equations). To address this problem, we reduce I to be of the same size as f_c, which makes the $\mathrm{conv}(I, f_c)$ produce only a single number. Then, Equation 2 turns into

$$\forall \mathbf{I}, (\mathbf{S} \cdot \mathbf{I})^T \cdot \mathbf{f}_t = \mathbf{I}^T \cdot \mathbf{f}_c, \qquad (3)$$

where $\mathbf{S}$ is the scaling matrix, $\mathbf{I}$ is the vector representation of the image, and similar are $\mathbf{f}_c$ and $\mathbf{f}_t$. It's easy to prove that Equation 3 is equivalent to

$$\mathbf{S}^T \cdot \mathbf{f}_t = \mathbf{f}_c, \qquad (4)$$

We can solve Equation 4 to obtain $\mathbf{f}_t$. However, in practice we can't always obtain an exact or unique $\mathbf{f}_t$ because $\mathbf{S}$ is not a square and invertible matrix. When $\mathbf{S}$ is a scaling-up matrix (#rows > #columns), the equation have infinite number of solutions; when $\mathbf{S}$ is a scaling-down matrix (#rows < #columns), the equation have no exact solutions.

For the first case with infinite solutions, we choose the solution with the minimum L2 norm,

$$\mathbf{f}_t = \arg\min_{\mathbf{f}_t} \|\mathbf{f}_t\|_2^2 \quad \text{subject to} \quad \mathbf{S}^T \cdot \mathbf{f}_t = \mathbf{f}_c. \qquad (5)$$

The reason that we choose a minimum-norm solution is similar of applying weight decay to the weights, i.e., to reduce over-fitting. A flat filter is likely to have more generalization to various cases. It is easy to get the solution of (5) by the generalized inverse of $\mathbf{S}^T$,

$$\mathbf{f}_t = \mathbf{S}(\mathbf{S}^T\mathbf{S})^{-1} \cdot \mathbf{f}_c. \qquad (6)$$

For the second case with no exact solution, we see the problem from a different angle: we take the scaled image $\tilde{\mathbf{I}} = \mathbf{S} \cdot \mathbf{I}$ as an input image, and proximate the original $\mathbf{I}$ with a scaling of $\tilde{\mathbf{I}}$, $\mathbf{I} \approx \tilde{\mathbf{S}}\tilde{\mathbf{I}}$. Here the $\tilde{\mathbf{S}}$ is a scaling-up matrix in the reverted direction of S. We turn the Equation 3 into

$$\forall \tilde{\mathbf{I}}, \ \tilde{\mathbf{I}}^T \cdot \mathbf{f}_t = (\tilde{\mathbf{S}} \cdot \tilde{\mathbf{I}})^T \cdot \mathbf{f}_c. \qquad (7)$$

Similar to Equation 4, we get

$$\mathbf{f}_t = \tilde{\mathbf{S}}^T \cdot \mathbf{f}_c. \tag{8}$$

In our implementation, we use bicubic interpolation [6] as the scaling method to transform filters. This method can produce nice scaling results without losing too much information of the original image.

In our model, we also consider a special scaling operation: horizontal flipping. We add some columns with flipped filters to capture the flipped patterns in input. The scaling matrix for flipping is a symmetric invertible matrix. It is very easy to solve Equation 4,

$$\mathbf{f}_t = \mathbf{S} \cdot \mathbf{f}_c. \tag{9}$$

2.4. Training Multiple Columns

We integrate all the columns with tied filters into a single model and train them together with back-propagation algorithm. Observing Equation 6, 8 and 9 in above cases, we find the transformation from canonical filter to any scale is always a linear transformation. That means, the filters in all the columns are tied to the canonical filter by a matrix multiplication,

$$\mathbf{f}_t = \mathbf{Q} \cdot \mathbf{f}_c,$$

where $\mathbf{Q}$ is some transformation matrix. Particularly, $\mathbf{Q}$ is an identity matrix for canonical column. This property makes back-propagation very convenient.

Suppose we have n columns, and the corresponding filters are

$$\mathbf{f}_t^i = \mathbf{Q}_i \cdot \mathbf{f}_c.$$

Define the cost function as E, which is a function of all the $\mathbf{f}_t^i$. By the chain rule of derivatives, we get

$$\frac{\partial E}{\partial \mathbf{f}_c} = \sum_{i=1}^{n} \left(\frac{\partial \mathbf{f}_t^i}{\partial \mathbf{f}_c} \right)^T \cdot \frac{\partial E}{\partial \mathbf{f}_t^i}$$
$$= \sum_{i=1}^{n} \mathbf{Q}_i^T \cdot \frac{\partial E}{\partial \mathbf{f}_t^i}.$$

In training, we first do the back-propagation in each column independently. Then, derivatives of the filters distributed on all columns are transformed and gathered as the canonical filters' derivatives. When the canonical filters are updated by these aggregated derivatives, all the filters on the scaled columns are recomputed by filter transformation from the new canonical filters.

3. Experiment Results

This section presents our experimental results. We begin with a detailed analysis on the scale invariance achieved within the network, followed by end-to-end performance on

CIFAR-10 dataset [7]. The baseline CNN is close to the Alex network [5], with 3 layers of convolution. Each layer uses 5×5 receptive size and a stride of 1, pooling of receptive size 3×3 and a stride of 2, followed by local normalization. The first convolution is paired with max pooling whereas the latter two is followed by average pooling. SiCNN extends it to 6 columns. The first three columns use filter size of 3, 5 and 7, with the last three columns being the flipped versions of the first three. All the weights are regularized and tied to the column with the 5×5 non-flipped column. We train these models on standard CIFAR-10 with the training method similar to that in [5]. We use the same hyper-parameters (learning rate, momentum, weight decay) as in [5]. We first train the whole net for 240 epochs, then reduce the learning rate by a factor of ten. We train for another 20 epochs, tune the learning rate again, and train for another 20 epochs to get the final result.

To exploit the invariance property of the model, we need to generate a new test dataset with a mixture of different scales. We crop the central 24×24 and 28×28 of the CIFAR-10 images and resize them to 32×32. This mixed dataset has 3 different scales: small, middle and large. We refer to this dataset as *scaled* CIFAR-10 later in this section.

Our experiment results are best viewed in electronic form.

3.1. Filter Transformation for Scale-Invariance

Consider an arbitrary image I and its scaled version $S(I)$. After applying a filter f and its transformation $T(f)$, their corresponding activations become $\texttt{conv}(I, f)$ and $\texttt{conv}(S(I), T(f))$, respectively. As described in Section 2.3, to achieve scale-invariant pattern matching, we expect the former after scaling is indistinguishable with the latter, i.e.,

$$S(\texttt{conv}(I, f)) \equiv \texttt{conv}(S(I), T(f)). \tag{10}$$

In section Section 2.3, we achieve this for small image patches; in this section, we verify this property for larger images.

Note that the left side of Equation 10, $S(\texttt{conv}(I, f))$ is the scaled activation of the canonical image, and is the design target of our transformation function. So we quantify with relative error using

$$\texttt{diff}(x, y) = \frac{\|x - y\|_2}{\|x\|_2}.$$

where $x = S(\texttt{conv}(I, f))$ and $y = \texttt{conv}(S(I), T(f))$.

We compare three different kinds of filter transformation methods. $\texttt{trans}^I$ is an identity transformation, with which we apply the original filter f onto the scaled image $S(I)$. $\texttt{trans}^T$ is the filter transformation described in Section 2.3. $\texttt{trans}^S$ is a comparison method, where we directly use simple image sampling to scale the filter. We also

normalize the transformed filter to the same L1 norm as the original filter; we find this normalization performs the best compared to other alternatives. $\mathtt{trans}^S$ takes such a transformation on filter f,

$$T(f) = \frac{\|f\|_1}{\|S(f)\|_1} S(f).$$

(a) Scaling up

Filter	$\mathtt{trans}^I$	$\mathtt{trans}^S$	$\mathtt{trans}^T$	pooling
random	11.64%	1.71%	1.67%	
CNN	59.63%	9.10%	9.19%	11.20%
SiCNN	66.11%	12.88%	10.94%	

(b) Scaling down

Filter	$\mathtt{trans}^I$	$\mathtt{trans}^S$	$\mathtt{trans}^T$	pooling
random	14.14%	1.81%	1.56%	
CNN	85.13%	20.50%	15.85%	9.22%
SiCNN	107.19%	29.23%	17.10%	

Table 1. Invariance-by-scaling regarding different filter transformation methods. The smaller the values listed in the table, the better Invariance-by-scaling.

We report the filter invariance-by-scaling by measuring

$$\mathrm{diff}(S(\mathtt{conv}(I, f)), \mathtt{conv}(S(I), T(f)))$$

in Table 1. We take 100 random images from the test set of CIFAR-10 and take their averaged result. Three canonical filters with size of 5×5 are considered: random filter (the first row), filters learnt in the baseline CNN model (the second row), and filters learnt in SiCNN (the third row). As comparison, non-overlapped 2×2 max pooling, which is usually considered powerful for scale-invariance, is taken as a non-parametric filter applied to image I and scaled $S(I)$. In Table 1(a), we use a scaling-up S that doubles the image size from 32×32 to 64×64. Accordingly, $\mathtt{trans}^T$ and $\mathtt{trans}^S$ scale the filter size from 5×5 to 9×9. In Table 1(b), S is a scaling-down. Image size is halved from 32×32 to 16×16, and the filter size is scaled down from 5×5 to 3×3.

From Table 1, it is clear that convolution with the same filter without any transformation is very sensitive to the image scale (column $\mathtt{trans}^I$). Our filter transformation method (column $\mathtt{trans}^T$) and the sampling-based method (column $\mathtt{trans}^S$) are much more robust to the image scale. $\mathtt{trans}^T$ is almost always better than the simple-minded $\mathtt{trans}^S$, especially when the image is scaled down and we need a more precise filter with a very small size. Considering the fact that $\mathtt{trans}^S$ is hard for back-propagation because of the normalization, our method becomes an apparent choice for transforming filters. Also, when the image is scaled up, our filter transformation is even better than
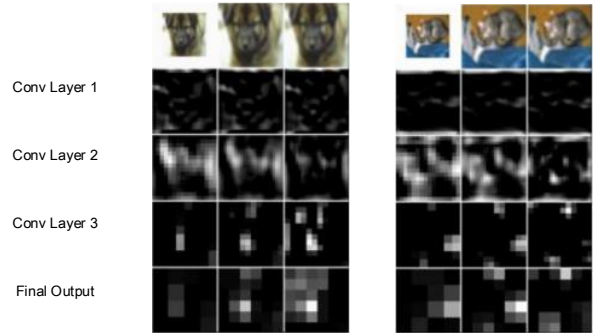

Figure 4. Visualization of activations in each layer. In each of the two examples, the left column shows the original image of size 32×32 and its activations in each layer using a 5×5 filter; the middle and right columns show enlarged image of size 48×48 and its activations in each layer using the 7×7 transformed filter and the same 5×5 filter, respectively. The filter is chosen randomly from the SiCNN model we trained.

pooling. Considering pooling doesn't need to detect any patterns, it's interesting to see that our method achieves robust invariance-by-scaling. When the image is scaled down, our method is still comparable against pooling. From random filter to SiCNN trained one, these filters adapt to some specific scale more and more. Consequently, the invariance-by-scaling of these filters also gets worse (from first row to the last row) as expected.

To give a more concrete feeling of our approach, we inspect the feature maps generated by images of different scales. Fig. 4 shows two particular examples, visualizing a feature map in each of the three convolution layers and the final result after pooling and normalization. In each example, the left column is the activations from the original image of 32×32, and the other two columns are from scaled image of 48×48. The left and middle column are results of using filter of the size 5×5 and its transformed 7×7 filter. All the feature maps have been scaled to the same size for ease of comparison. From the relatively small difference in each layer, it is clear that applying the transformed filters on the scaled image preserves the essential characteristics of the original. The rightmost column is the result of applying the original 5×5 filter to the scaled image. It is clear that the fixed filter generates the activations that diverge from that of the original image (the leftmost column) significantly.

3.2. Multi-Column Features

To give an idea of what features the different columns learn, we scan over the activations on the last pooling layer caused by 30,000 test images from the scaled CIFAR-10.
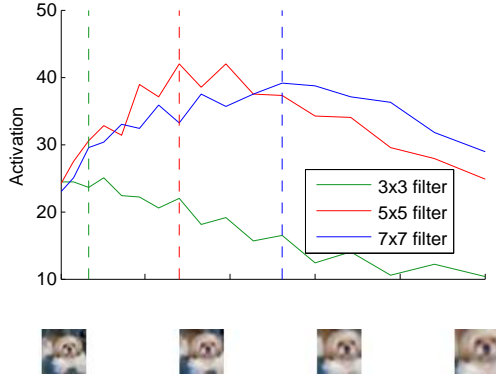
Figure 6. Activation versus object size in three columns. Peaks of each curve are illustrated by dash lines.

We randomly pick up a filter in the last layer and visualize the top 16 images that cause the largest outputs of this filter, in each of the 6 columns individually (Fig. 5). This method is similar to that used in [17]. It can be seen that each column in our model focuses on a particular scale and orientation: the images which causes largest activations get larger from left to right, and the automobiles in the two rows face opposite directions.

In addition to the visual inspection, we try to quantify how sensitive filters of different columns are to the scales. We take the top 100 images that this feature of a given column gets activated the most, then break them down according to which scale they belong to in the data set: small, middle or large. This statics is reported at the bottom of the images in Fig. 5. It is clear that columns with small filters "picks" the small-scale images more, whereas the columns with larger filters does the opposite.

When an object for recognition is scaled from small to large, the columns in SiCNN will also work in turn to capture this object. We illustrate that in Figure 7. Using method similar as in Fig. 5, we first select a feature map that detects dogs in the last layer. Then we pick a dog image from CIFAR-10 and scale the object into different sizes (2x larger at most). The max activation value in the feature map are plotted as a function of the object size for each scale column. In Figure 7, it is clear that when the object is small, the column with 3×3 filter first captures it and gives a big response. When the object gets larger, activations on this 3×3-filter column drop. The 5×5 and 7×7 columns gradually reach their peak responses in turn. The peaks of the three columns locate with the same interval along the object size, because of the equal-interval filter sizes of 3, 5 and 7. Comparing activation values among different columns is meaningless, because they are eventually summed up with different weights for classification. However, this study clearly shows that by tracing which column is activated more, we can detect a object as well as its scale.

3.3. Scale-Invariant Classification

Model	Standard CIFAR-10	Scaled CIFAR-10	Performance drop
CNN	17.33%	24.82%	43.22%
SiCNN	14.22%	18.83%	32.42%
Improvement	17.94%	24.13%	

Table 2. Classification error rate, tested on standard CIFAR-10 and scaled CIFAR-10. The last row shows the classification improvement by SiCNN. The last column shows the performance drop due to test dataset with more scales.

Table 2 compare the results of the baseline CNN and SiCNN, both of which are trained on standard CIFAR-10 dataset. SiCNN achieves statistically significant gain on standard CIFAR-10. Its full advantage is more apparent with the scaled CIFAR-10, where CNN has a performance drop of more than 43.22%, and SiCNN drops by 32.42%. We manually examine the error cases, and find that the simple central-crop-resize has cut off many significant features in the scaled CIFAR-10. We speculate SiCNN will work better on higher quality multi-scale datasets.

To verify the above hypothesis, we pick 5 random images in which the object is at the center, and scale them to different sizes; the largest one is the central 16×16 area of the image resized to 32×32. We put these images into both CNN and SiCNN, and compare the probability of the correct class. The results are shown in Fig. 7. It can be seen that as the scale of the image goes up, the performance of CNN drops whereas SiCNN is stable. The only exception among these samples is the one of horse; its scaled up versions start to lose vital features.

3.4. Training results on CIFAR-10

Method	Testing error
CNN [5]	16.6%
CNN+ dropout [5]	15.6%
CNN+Spearmint [15]	14.98%
SiCNN	**14.22**%
CNN+Maxout [4]	11.68%
CNN+Maxout [4] + **SiCNN** (voting)	**11.35**%
Maxout-**SiCNN** (2-column)	**11.33**%
Network in Network [10]	10.41%

Table 3. Comparison of error rate.

In Table 3, we compare the classification error rate of SiCNN with other previous approaches on CIFAR-10 [7]. We achieve an error rate of 14.22% on unaugmented data, an improvement of more than 2% absolute gain over the baseline CNN in [5]. SiCNN also exceeds other improvement on CNN, such as dropout [5] and Spearmint [15], but is insufficient to catch up with the maxout [4] and network-

3×3
column 1
5×5
column 2
7×7
column 3

Non-Flipped

column 4
column 5
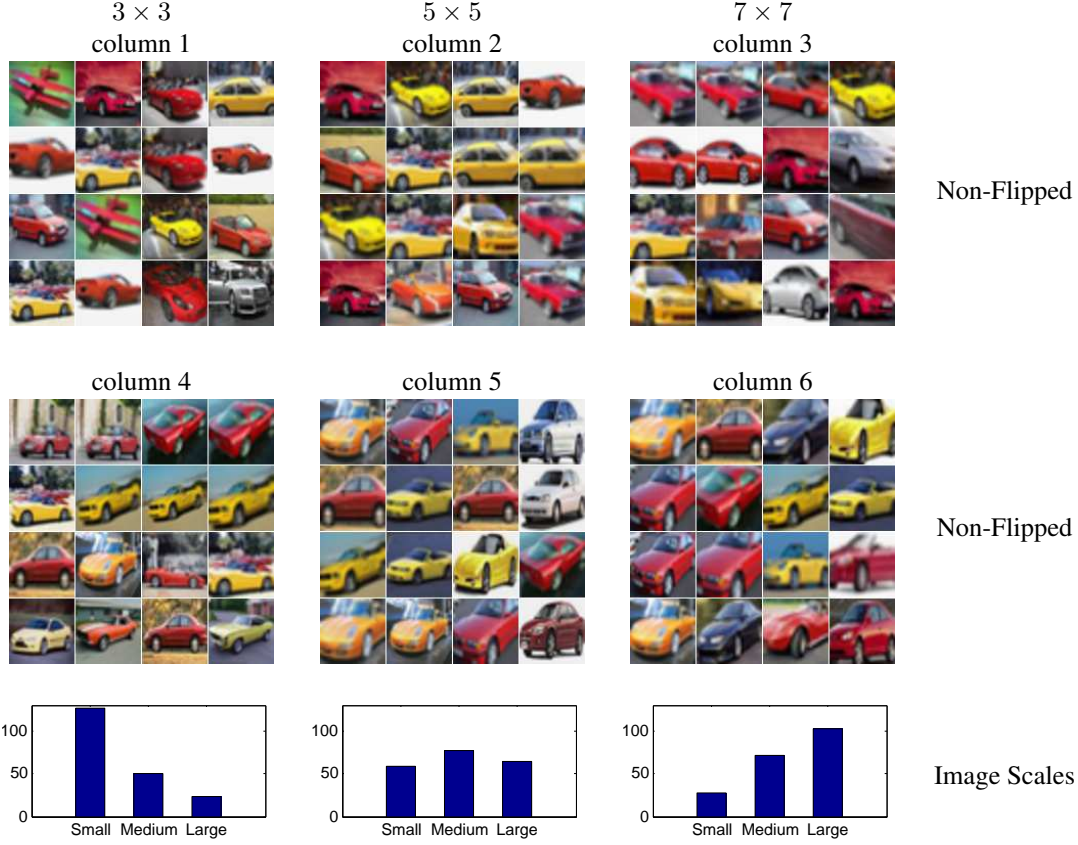column 6

Non-Flipped

Image Scales

Figure 5. Visualization a filter with respect to the 6 columns. In each column, the top 16 images that causes the largest activation of the filter are shown. The breakdown analysis of top 100 images are shown below each column scale (we combine each two flipped columns together).

(a) Corresponding Images
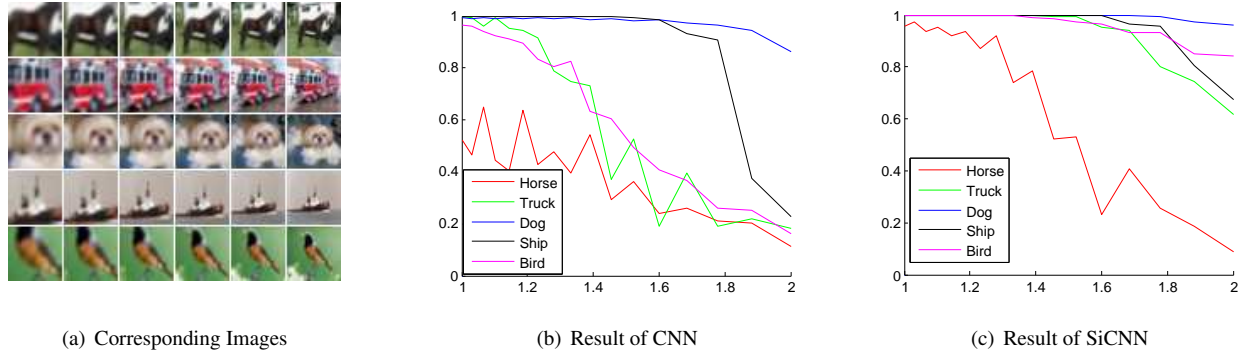
(b) Result of CNN

(c) Result of SiCNN

Figure 7. Probability of the correct class.

in-network [10], which are the current state of the art. Nevertheless, our method can be combined with these techniques as SiCNN is addressing scale-invariance problem, which is a different goal from others. For example, by using an average voting with SiCNN, we drop the error rate of maxout model from 11.68% to 11.35%. Moreover, by simply adding a extra flipped column to maxout model, we reach an error rate of 11.33% with a single 2-column maxout-SiCNN model. We find these results encouraging, and expect that SiCNN will work better on benchmarks that exhibit higher scale variations, as the results in Section 3.3 suggest. Replicating SICNN on larger and more complex dataset such as ImageNet is ongoing work.

SiCNN takes the form of multi-column CNN without blowing up number of free parameters. As a more direct comparison, we train a 6-column CNN where the filters are

Model	Error on CIFAR-10	Error on scaled CIFAR-10	cost
CNN	17.33%	24.82%	1
SiCNN	14.22%	18.83%	6
SiCNN, `inc-1`	14.71%	20.10%	3.5
SiCNN, `inc-2`	16.06%	23.24%	$1 + \epsilon$

Table 4. Classification error rate and cost for incremental training. The last two rows, `inc-1` and `inc-2`, correspond to the two incremental training methods. All the training costs are normalized to CNN's cost. ϵ is a very small value. Here $\epsilon = 0.015$.

independent. Under the same training condition this network suffers severe overfit, the testing error hovers around 19% while training error already reaches zero.

3.5. Incremental Training

Improving scale invariance does not come for free. In the current configuration, training costs increase linearly with number of columns. Clearly, we can first train one single column, transform its filters to other columns and finally refine the model. In this ideal setting, it is reasonable to expect that the additional training cost is insignificant.

We explored two incremental training methods. In the first choice (named `inc-1`), we first train a baseline CNN with about half the epochs of a full training, and we build a 6-column SiCNN based on the current filters. Then we begin to refine the entire model with the left half of the epochs. In the second choice (named `inc-2`), we continue from a fully trained baseline CNN, and use its filters to build the 6-column SiCNN. Then, with all the filter parameters frozen, we only refine the parameters in classifier. As we use a single softmax layer as classifier, the `inc-2` method has a very small extra cost.

Results for incremental learning are summarized in Table 4. Compared with SiCNN trained from scratch (the second row), `inc-1` training (the third row) takes nearly half cost, with the model achieving a comparable performance. With the `inc-2` training (the fourth row), although the extra training cost is very small (1.5%), we still get a model with better performance than baseline CNN. Also by combination with the maxout units [4], we're able to reach an error rate of 11.33% which is higher than the previous result. Incremental learning does help to balance the performance gain and training efficiency in SiCNN.

4. Conclusion

In this paper, we propose a new generalization of CNN, SiCNN, where we incorporate scale and flip invariance into the model. This model improves the results of traditional CNN, and complements other optimization techniques. Our results clearly indicate that the model learns the feature in different scales in different columns. The idea is generaliz-

able, can be applied in all aspects where CNN is employed, including supervised and unsupervised learning, recognition, detection, and localization tasks. Our preliminary study also implies a nice trade-off between performance and training cost.

Several open problems remain. For example, we can use a different way of summarizing all the columns (instead of concatenation), and different connectivity structure among column (e.g., pair-wise between columns instead of all-to-one against the canonical column). We plan to apply SiCNN to larger and more complex datasets such as Imagenet.

References

[1] D. Ciresan, U. Meier, and J. 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. 1

[2] D. Erhan, Y. Bengio, A. Courville, and P. Vincent. Visualizing higher-layer features of a deep network. *Dept. IRO, Université de Montréal, Tech. Rep*, 2009. 1

[3] Y. Gong, L. Wang, R. Guo, and S. Lazebnik. Multi-scale orderless pooling of deep convolutional activation features. *arXiv preprint arXiv:1403.1840*, 2014. 1

[4] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013. 6, 7, 8

[5] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012. 4, 6, 7

[6] R. Keys. Cubic convolution interpolation for digital image processing. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 29(6):1153–1160, 1981. 4

[7] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Computer Science Department, University of Toronto, Tech. Rep*, 2009. 1, 4, 7

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 1, 2

[9] Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361, 1995. 1

[10] M. Lin, Q. Chen, and S. Yan. Network in network. *CoRR*, abs/1312.4400, 2013. 6, 7

[11] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *arXiv preprint arXiv:1409.0575*, 2014. 1

[12] D. Scherer, A. Müller, and S. Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *Artificial Neural Networks–ICANN 2010*, pages 92–101. Springer, 2010. 1

[13] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification

models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013. 1

[14] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1

[15] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959, 2012. 6, 7

[16] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabi-novich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014. 1

[17] M. D. Zeiler and R. Fergus. Visualizing and under-standing convolutional neural networks. *arXiv preprint arXiv:1311.2901*, 2013. 1, 5