ImageNet Object Recognition with Scale-Invariant Deep Convolutional Neural Network

Alireza Akbari
Department of Computer Engineering
Sharif University of Technology
Tehran, Iran
Email: akbary@ce.sharif.edu

***Abstract*—ImageNet is a large dataset including more than 1.2 million images with 1000 different classes. Statistical methods have shown poor results in image classification tasks on these large datasets. We present a deep convolutional neural network model, which has done an excellent job on this task. It contains five convolutional layers, followed by max-pooling layers producing some invariances, and three fully-connected layers which led to multinomial logistic regression at the end due to 1000 different classes. The model has more than 60 million parameters, so some techniques are designed to prevent our model from overfitting. Furthermore, images in the ImageNet dataset exist in different scales. Although deep neural networks have shown robustness to object locations and orientations, they are not invariant to the scale of objects. It is easy to introduce new parameters for each scale, but there are already many parameters in our model. Consequently, we have designed a simple layer called scale-invariant which appears instead of every convolutional layer and adds no new parameters to the model. Our model achieved a 37.5% top-1 error rate in LSVRC-2010 contest which is a huge improvement between existing methods.**

***Index Terms*—Convolutional Neural Networks, Deep Learning, Scale variation robustness, ImageNet**

I. Introduction

Image Classification is one of the most famous tasks in the Computer Vision Field. Several types of applications can be tackled with that. Machine Learning methods are used widely recently and they have shown good results on small datasets. For example Keysers et al. [9] have achieved an error rate of 0.54 on the MNIST dataset [12].But these small datasets are way too far from practical applications. Improvement of existing methods needs larger datasets and stronger models [16].

By emerging large datasets e.g. LabelMe [17] and ImageNet [2], machine learning methods acted poorly, since they were highly dependent to feature engineering. The desirable model has to have a high capacity for learning, due to the immense number of images and different classes. Convolutional Neural Networks (CNNs) are shown that have the capability of learning these kinds of hard problems [8], [11], [13], [14]. Their specific property which makes them successful is their layer by layer structure, which is inspired by similar brain hierarchical organization [7], allowing them extracting simple patterns to complex ones like other deep neural networks. Besides, they do not distort spatial location and two-dimensional structure by convolving filters all around the image.

However, there are still some problems with CNN. Their application on high-resolution images can be computationally expensive. Fortunately, current GPUs can handle sufficient computational resources, if they are equipped for an optimized implementation of convolving operation. Another problem is called scale variation. Objects in the real world come in different scales because of the third dimension and CNNs cannot find the same pattern which is appeared in different scales. The introduction of new filters for each scale to capture patterns in multiple scales leads to an increase in the number of parameters, which is already a serious constraint as mentioned. So, it is important to reach scale invariance without adding new parameters.

In this paper, we introduce our deep convolutional neural network having 60 million parameters, which has achieved the best error rate so far in ImageNet Large Scale Visual Recognition Challenge 2010 (ILSVRC-2010) [1], outperforming existing models. The dataset is a subset of original ImageNet [2] having 1.2 million training images, 50000 validation images, and 150000 test images all labeled (Images come from different resolutions, so we rescale each image to 256×256 pixels). Then we describe specific techniques and layers we used to solve different problems. For example, applying non-saturating nonlinearities to accelerate learning procedure, or we will describe scale-invariant layer (SI-layer) which has brought us robustness to scale variation without adding new parameters by sharing information between multiple scales by using just one filter.

With more than 60 million parameters, overfitting is a nightmare even if one has a large training set. Therefore, in the next section, we will describe the methods used to prevent overfitting. Then we explain the details of the learning procedure, especially for our new SI-layer. Finally, we present our results separately for SI-layer that it is more robust to unfamiliar scales against simple CNNs and then the results for the whole model in the contest.

II. Model Details

Our model architecture is displayed in Fig. 1 generally. The full details about layers of the model will be described after the explanation of methods used to solve different problems which we faced in process of learning.

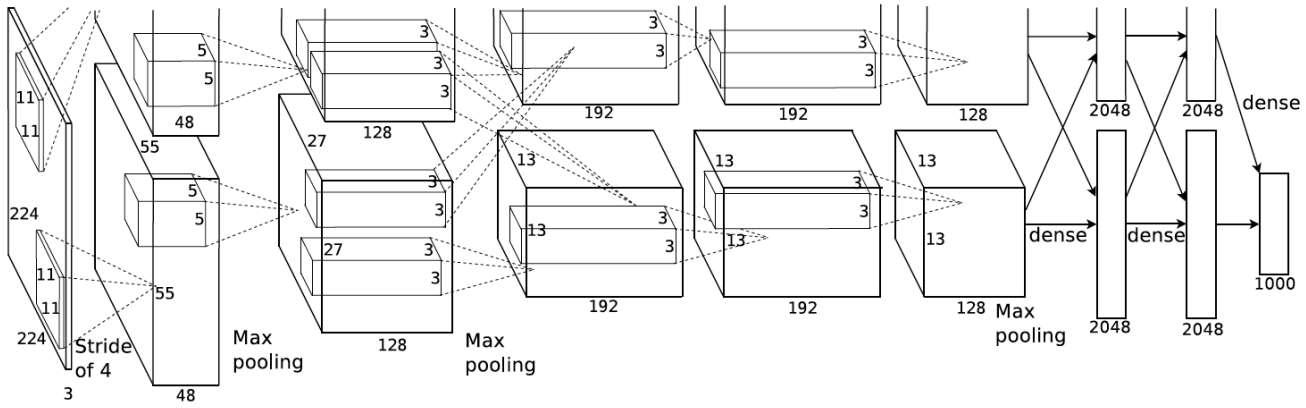


Fig. 1: The model general architecture. The two branch of the figure is because of the GPUs. The pattern of connection between layers in different GPUs is determined here.

A. Model Features

Some novel features used in our model are:

1) Non-Saturating Nonlinearities: Primary nonlinear activation functions used in neural networks were functions like tanh or sigmoid function $(\frac{1}{1+e^{-x}})$. The main problem with these kinds of nonlinearities is that they saturate which leads to a slow training procedure. However non-saturating functions like f(x) = max(0, x) called Rectified Linear Units (ReLUs) make deep CNNs train much faster [15]. This is illustrated in Fig. 2.

2) Scale Invariant Layer: To obtain scale-invariant feature detection, we have changed the traditional convolutional layer. It is demonstrated in Fig. 3. We convolve the exact same filter on multiple scales of input, which are obtained by different linear maps parametrized by a hyperparameter s. To apply pooling between multiple outputs we got, first we have to restore all the outputs to the same scale. Therefore, we use a corresponding inverse linear map for each scale to get all the outputs aligned to the same shape. Subsequently, in the end, we max-pool different firings of neurons at each specific spatial location. This special layer is called SI-layer. Fig. 4 is showing detection of "V" in different scaled by SI-layer. At each scale, the corresponding scale branch fired and others did not.

3) GPU Implementation: 1.2 million training images cannot reside on the memory of the GPU we used, altogether. So, we had to divide the deep neural network into two GPUs. An important thing to notice is that current GPUs can interact with other GPUs´ memory properly, so the performance would not be affected by using parallel GPUs. Therefore, on each GPU, there are half of the neurons. That is the reason in Fig. 1 that the model is divided into two parts.

In this situation, GPUs can interact with each other in different layers. For example, in Fig. 1, notice that layer 3 gives input from all neurons, but layer 2 gives only input from neurons staying on its same GPU. So, the pattern of this kind of connectivity between layers can affect the model clearly. The optimum pattern can be achieved by the cross-validation method.

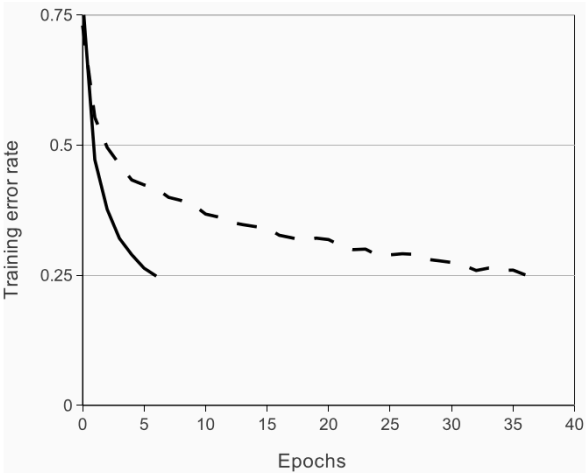
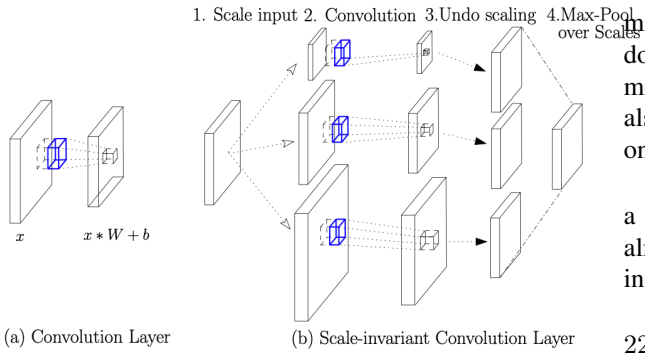


Fig. 2: Comparison between two convolutional neural networks with 4 layers. The solid line is related to the model in which neurons have ReLU nonlinearity and the dashed line is about the model with tanh neurons. As it is seen in the figure, the model with ReLU gets to 25% of error six times faster than the other one.

4) Local Response Normalization: Lateral Inhibition is the capability of a neuron to reduce its neighbors' firing rates while firing. This leads to a contrast between the firing neuron and its neighbors which results in amplified perception [4]. In a convolutional layer, there are multiple kernels each extracting specific features from the input, which means in a specific spatial location, there are multiple neurons from different



(a) Convolution Layer (b) Scale-invariant Convolution Layer

Fig. 3: (a) A natural convolutional layer (b) A scale-invariant layer. The exact same **W** used in (a) is also used here shared by all branches. The input image is scaled into multiple different scales and then rescaled into the first scale and finally, the maximum activation is computed between all branches. Notice that both in (a) and (b) the input and output are dimensionally the same.

kernels. Using a kind of normalization called Local Response Normalization between these neurons results somehow similar to lateral inhibition. This behavior added a normalization layer at the end of convolutional layers, although we used ReLU nonlinearities in our model. The expression of this normalization is

$$b^i_{x,y} = \frac{a^i_{x,y}}{\left(k + \alpha \sum_{j=max(0,i-\frac{n}{2})}^{min(N-1,i+\frac{n}{2})} (a^j_{x,y})^2\right)^\beta} \quad (1)$$

where $a^i_{x,y}$ is the former activity of neuron at position (x,y) in i-th kernel and $b^i_{x,y}$ is the normalized one. Moreover, N is the total number of kernels in that specific convolutional layer and the sum is along the n adjacent kernel neurons. There are also multiple hyperparameters, k, α, n, and β which can be obtained by cross-validation method. Values we used are $k = 2$, $\alpha = 10^{-4}$, $n = 5$ and $\beta = 0.75$

B. Preventing Overfitting

1) Dropout: Deep neural networks can use a strange combination of neurons to learn complex functions which can possibly lead to overfitting of the model. Dropout is a method enabling the prevention of complex co-adaptation of neurons in the model [6]. In this method, each neuron fires with a probability of 0.5 while forward step and backward propagation. So, each time there is a new input for our model, our model has a different formation. Therefore, neurons' co-adaptation becomes more simple preventing overfitting from our model.

In test time, we should consider the expected value of activation of the layer merged with dropout. A good approximation could be half of the activation of that layer since each neuron has contributed with a probability of 0.5 in training time.

2) Data Augmentation: Another technique widely used in overfitting methods is augmenting the dataset. The idea is to make transformations to images in the dataset in a way that doesnt change the label of that image [19]. As a result, the model will not only learn that specific pattern of the image but also will learn more general patterns instead of some special ones.

It is of great importance to implement augmentation with a very low computational cost. Actually, it is implemented almost computationally free. The new images are generating in CPU while the previous batch is running on GPU.

For augmentation of a single 256×256 image, we take 224×224 pieces from the images (This is the reason that the input layer in Fig. 1 gives 224×224 image). Accordingly, this leads to an increase in the number of training images up to 1024 $((256 - 224) \times (256 - 224))$. We also reflected each of these images horizontally and used them as new samples as well. Hence, the samples increased by a factor of 2048 (1024×2).

C. Architecture Details

Some of the details of the architecture implied, like the connectivity between layers staying on the same GPUs or connectivity between all which are obvious on Fig. 1 too. But in general, the architecture contains five SI-layer (which is a scale-invariant version of a convolutional layer) and three fully-connected layers. ReLU is used for all layers as their activation functions. The dropout method is used just on the first two fully-connected layers. Although there are 5 SI-layer in the model, we only used local response normalization and max-pooling on the first two layers. And finally, the loss function is the softmax function.

Also, the kernels and filters (weights) used in each layer is as follows:

Layer	Weights
Conv1	96 kernels of size $11 \times 11 \times 3$(stride 4)
Conv2	256 kernels of size $5 \times 5 \times 48$
Conv3	384 kernels of size $3 \times 3 \times 256$
Conv4	384 kernels of size $3 \times 3 \times 192$
Conv5	256 kernels of size $3 \times 3 \times 192$
Fully-Connected Layers	4096 neurons each

TABLE I: Weights used in each layer details

III. Learning Procedure

A. Forward Propagation in SI-layer

In a pure convolutional layer, the final output of that layer is computed as follows

$$h = \sigma((W * x) + b) \quad (2)$$

where σ is layer nonlinearity applied. $*$ is the convolutional operator, which makes filter parameter (weights) W convolve all around the two-dimensional input x. Also, b is the bias parameter that change the mean of firing rates of each neuron.

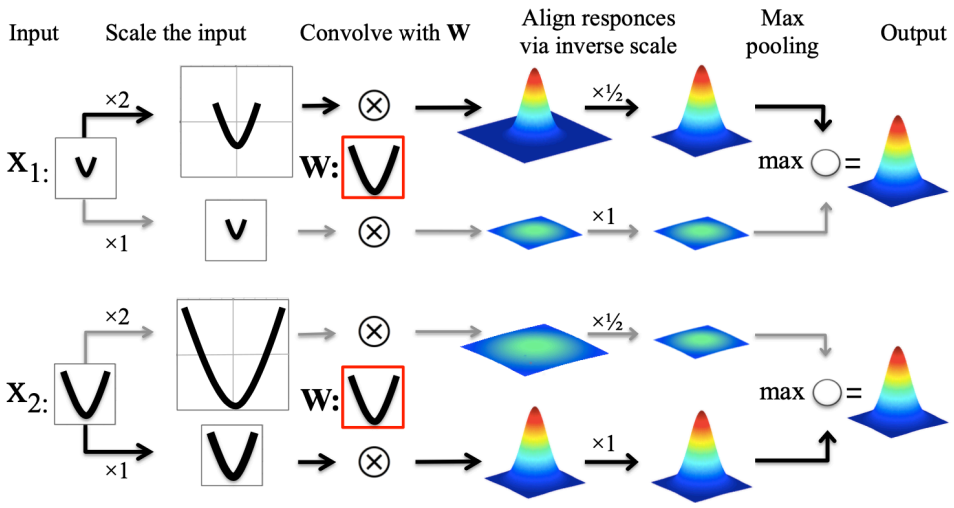


Fig. 4: Demonstration of how the SI-layer detects objects regardless of scale variation. In each image, the bolded path specifies the way that the layer extracts the adequate feature.

But in SI-layer the process gets different. As it was mentioned in Scale Invariant Layer part, we transform an image to multiple different scales. For a transformation T_i we have

$$\hat{z}_i = (W * T_i(x)) + b \tag{3}$$

$$z_i = T_i^{-1}(\hat{z}) \tag{4}$$

where $\hat{z}_i$ is the convolution output on that specific scale. And z_i is the output restored to the early scale. So, for $T_1, T_2, \cdots T_n$ transformations we have

$$h = \sigma\left(\max_{i\in\{1,2,\cdots n\}}[z_i]\right) \tag{5}$$

which is the final output activation computed when all of different scales output aligned in the same scale.

B. Update Rules Details

The update rule for weights following from stochastic gradient descent paradigm is as follows

$$v_{i+1} := 0.9 \cdot v_i - \epsilon \cdot \left\langle \frac{\partial L}{\partial w}|_{w_i} \right\rangle_{B_i} - 0.0005 \cdot \epsilon \cdot w_i \tag{6}$$

where 0.9 and 0.0005 are momentum and weight decay values. ϵ is learning rate which was initialized at 0.01 and divided by 10 when the error rate became stable. B_i is the i-th batch used in updating process which is a set of 128 examples and i is the iteration. Therefore, $\left\langle \frac{\partial L}{\partial w}|_{w_i} \right\rangle_{B_i}$ means the average of partial derivative of loss function with respect to weights over the i-th batch, measured at w_i. By calculating v_{i+1} which is momentum variable, we can update weights

$$w_{i+1} := w_i + v_{i+1} \tag{7}$$

Initialization of parameters is written in table II

Parameter	Initialization
Weights	from $\mathcal{N}(0, 0.01^2)$
Biases in 2nd,4th and 5th SI-layer	1
Biases in remaining layers	0

TABLE II: Initialization of Parameters in Learning

IV. Experiments

Our experiments and result consist two main parts. At first, we made a simple model with SI-layers and measured effect of this kind of layer in performance. Then we compared the results to current methods trying to be scale invariant and basic CNNs. The second part is related to our results in ILSVRC-2010 using the model we described.

A. Scale Invariant Results

1) MNIST-Scale Benchmark: A good benchmark dataset which has images in multiple scales is MNIST-scale which is a modified version of MNIST in which images are rescaled into multiple scales [20]. So, we took advantage of this dataset to measure SI-layer-based CNNs. We used a simple model with two SI-layers having 36 and 64 filters of size 7×7 and 5×5. Then a fully-connected layer with 150 weight parameters leads to a softmax classification layer. As we mentioned in the Scale-Invariant Layer part, this layer needs hyperparameter s to scale input images. We used six scales from 0.6 to 2. All the other networks that we compared to our model, have exactly the same architecture we used.

Our results are shown in table III. The models that we compared our method to are presented there. Our model obtains the best result among all the other methods.

There are some facts seen in this result. First of all, we see that the scale-invariant version of RBM resulted in almost 10%

Method	Test Error(%) averaged on 6 train/test fold
Restricted Boltzman Machine (RBM) [20]	6.1
Scale-invariant RBM [20]	5.5
Hierarchical CNNs [3]	3.58 ± 0.17
CNN [21]	3.48 ± 0.23
SI-layer CNNs	**3.17 ± 0.19**

TABLE III: Different Methods Performance on MNIST-Scale Dataset

improvement in comparison to the natural one. In addition to that, our model also achieved a similar improvement, which shows that our technique for reaching scale-invariant robustness in supervised learning is as good as the technique used in unsupervised learning methods.

Another thing to notice is that about hierarchical CNNs. Although they have many more parameters than natural CNNs, they achieved poorer performance. The reason possibly is about overfitting of this model, which shows the importance of not increasing the number of parameters.

2) Invariance Measurement: Furthermore, we measure the invariance of our technique against the scale of data. We used the criterion introduced in Goodfellow et al []. Each neuron gets an invariance score which is computed by $\frac{L_i}{G_i}$. If multiple transformations are applied to the inputs, the number of times that a neuron has fired to transformed inputs is called L_i. And $G_i = \frac{\sum |h_i(x) > t_i|}{N}$, where h_i is firing rate of a neuron and the meaning of $h_i(x) > t_i$ is that neuron is firing. And t_i is chosen in a way that G_i which is an average of firing between N input becomes more than 0.01.

Our results at the end of each layer of the model are in Fig. 5. We compared it to the natural CNN model. Our transformation parameter s in this experiment is chosen from 0.3 to 1.2. We see that our model feature detection at each layer is more robust to scale variation.

B. ILSVRC-2010 Results

Finally, we announce our performance with model seen in Fig. 1 in ILSVRC-2010 challenge. Our model obtained a huge improvement on this challenge among the current state-of-the-art methods. The results are in table IV

Model	Top-1	Top-5
Sparse coding [2]	47.1%	28.2%
SIFT + FVs [18]	45.7%	25.7%
CNN	**37.5%**	**17%**

TABLE IV: Our result and comparison to other methods' results in ILSVRC-2010

V. Conclusion

We presented a new deep convolutional neural network which showed excellent improvement on hard classification tasks. The deep structure helps to extract more complex features as we go through the depth. The model shows that one of its main limitations is GPUs memory and it can tackle the problem even better with the improvement of computational power. We also introduced a new layer called Scale-Invariant which makes the model robust to scale variation of images. The important advantage of this layer is that it adds no new parameters to the model.

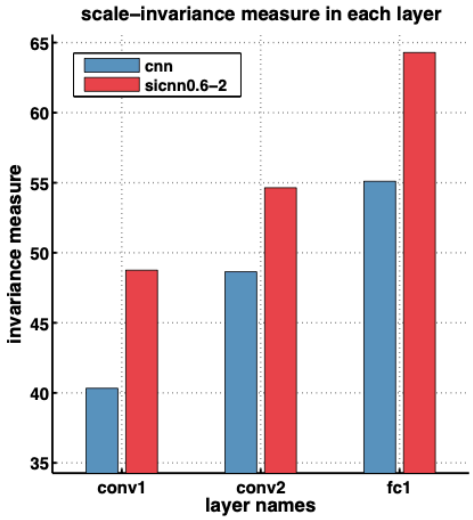


Fig. 5: Scale-Invariance Measurement and Comparison

Future approaches can extend this model and make it even more robust and powerful. For example, weight initialization can be changed to the method introduced by He et al. which considers specifically ReLU nonlinearity [5]. Another example is using Adam optimizer which has shown a fascinating learning process [10]. Additionally, they can use deep convolutional neural networks on data of videos which there is much information that doesnt exist in images.

References

[1] A. Berg, J. Deng, and L. Fei-Fei. Imagenet large scale visual recognition challenge. *www.image- net.org/challenges*, 2010.

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[3] Clement Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning hierarchical features for scene labeling. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1915–1929, 2012.

[4] E Bruce Goldstein. *Sensation and perception*. Cengage Learning, 2009.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

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

[7] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106–154, 1962.

[8] Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th international conference on computer vision*, pages 2146–2153. IEEE, 2009.

[9] Daniel Keysers, Thomas Deselaers, Christian Gollan, and Hermann Ney. Deformation models for image recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(8):1422–1435, 2007.

[10] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[11] Alex Krizhevsky and Geoff Hinton. Convolutional deep belief networks on cifar-10. *Unpublished manuscript*, 40(7):1–9, 2010.

[12] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[13] Yann LeCun, Fu Jie Huang, Leon Bottou, et al. Learning methods for generic object recognition with invariance to pose and lighting. In *CVPR (2)*, pages 97–104. Citeseer, 2004.

[14] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th annual international conference on machine learning*, pages 609–616. ACM, 2009.

[15] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[16] Nicolas Pinto, David D Cox, and James J DiCarlo. Why is real-world visual object recognition hard? *PLoS computational biology*, 4(1):e27, 2008.

[17] BryanC. Russell, Antonio Torralba, KevinP. Murphy, and WilliamT. Freeman. Labelme: A database and web-based tool for image annotation. *International Journal of Computer Vision*, 77(1-3):157–173, 2008.

[18] Jorge Sánchez and Florent Perronnin. High-dimensional signature compression for large-scale image classification. In *CVPR 2011*, pages 1665–1672. IEEE, 2011.

[19] Patrice Y Simard, David Steinkraus, John C Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *Icdar*, volume 3, 2003.

[20] Kihyuk Sohn and Honglak Lee. Learning invariant representations with local transformations. *arXiv preprint arXiv:1206.6418*, 2012.

[21] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International conference on machine learning*, pages 1058–1066, 2013.