On the Exploration of Convolutional Fusion Networks for Visual Recognition

Yu Liu, Yanming Guo, and Michael S. Lew

LIACS Media Lab, Leiden University, Leiden, The Netherlands
{y.liu, y.guo, m.s.lew}@liacs.leidenuniv.nl

Abstract. Despite recent advances in multi-scale deep representations, their limitations are attributed to expensive parameters and weak fusion modules. Hence, we propose an efficient approach to fuse multi-scale deep representations, called convolutional fusion networks (CFN). Owing to using $1 \times 1$ convolution and global average pooling, CFN can efficiently generate the side branches while adding few parameters. In addition, we present a locally-connected fusion module, which can learn adaptive weights for the side branches and form a discriminatively fused feature. CFN models trained on the CIFAR and ImageNet datasets demonstrate remarkable improvements over the plain CNNs. Furthermore, we generalize CFN to three new tasks, including scene recognition, fine-grained recognition and image retrieval. Our experiments show that it can obtain consistent improvements towards the transferring tasks.

Keywords: Multi-scale deep representations · Locally-connected fusion module · Transferring deep features · Visual recognition

1 Introduction

Since their repeated success in ImageNet classification [16,27,29,10], deep convolutional neural networks (CNNs) have contributed much to computer vision and the wider research community around it. CNN features can be used for many visual recognition tasks, and obtain top-tier performance [2]. Moreover, some works [1,21,34] begin capturing complementary features from intermediate layers. However, their methods mainly make use of one off-the-shelf model trained on the ImageNet dataset [26], but not to train a new network that can integrate intermediate layers. Instead, recent work [33] trains a multi-scale architecture for scene recognition at the expense of increasing algorithm complexity.

Hence, we propose to train an efficient fusion architecture to integrate intermediate layers for visual recognition. Our architecture is called convolutional fusion networks (CFN), which mainly consists of three characteristics: (1) Efficient side outputs: we add few parameters to generate new side branches due to using efficient $1 \times 1$ convolution and global average pooling [20]. (2) Early fusion and late prediction: in contrast to [33], we present an “early fusion and late prediction” strategy. It can not only reduce the number of parameters, but also
produce a richer image representation. (3) **Locally-connected fusion**: in the fusion module, we propose making use of a locally-connected layer to learn adaptive weights (importance) for the side outputs. To the best of our knowledge, this is the first attempt to apply a locally-connected layer to a fusion module.

In a nutshell, our contributions can be summarized as follows. First, an efficient fusion architecture is presented to provide promising insights into efficiently exploiting multi-scale deep features. Second, we train CFN on the CIFAR and ImageNet 2012 datasets, and evaluate its efficiency and effectiveness. Experimental results demonstrate the superiority of CFN over the plain CNN. Third, we generalize the CFN model to other new tasks, including scene recognition, fine-grained recognition and image retrieval. Results show that CFN can consistently achieve significant improvements on these transferring tasks.

### 2 Related work

In this section, we summarize existing approaches that focus on intermediate layers in the following three aspects.

**Employment of intermediate layers.** In CNNs, intermediate layers can capture complimentary information to the top-most layers. For example, Ng, et al. [35] employed features from different intermediate layers and encoded them with VLAD scheme. Similarly, Cimpoi, et al. [5] and Wei, et al. [31] made use of Fisher Vectors to encode intermediate activations. Moreover, Liu, et al. [21] and Babenko, et al. [3] aggregated several intermediate activations and generated a more discriminative and expensive image descriptor. Based on intermediate layers, these methods are able to achieve promising performance on their tasks, as compared to using the fully-connected layers.

**Intermediate supervision.** Considering the importance of intermediate layers, Lee, et al. [18] proposed the deeply supervised nets, which imposed additional supervision to guide the intermediate layers earlier, rather than the standard approach of only supervising the final prediction. Similarly, GoogLeNet [29] created two extra branches from the intermediate layers and supervised them jointly. However, these approaches do not explicitly fuse the outputs of intermediate layers.

**Multi-scale fusion (or skip connections).** To incorporate intermediate outputs explicitly during training, multi-scale fusion is presented to train multi-scale deep neural networks [33,22,32]. A similar work in [33] builded a DAG-CNNs model that summed up the multi-scale predictions from intermediate layers. However, DAG-CNNs required processing a large number of additional parameters. In addition, its fusion module (i.e. sum-pooling) failed to consider the importance of side branches. However, our CFN can learn adaptive weights for fusing side branches, while adding few parameters.

### 3 Proposed approach

In this section, we introduce the architecture of CFN and its training procedure.
Fig. 1: The general pipeline of convolutional fusion networks (best viewed in zoom in). The side branches start from the pooling layers and consist of a $1 \times 1$ convolution and global average pooling. All side outputs are then stacked together. A locally-connected layer is used to adaptively learn adaptive weights for the side outputs. Finally, the fusion feature is fed to the following fully-connected layer that is used to make a final prediction.

3.1 Architecture

Similar to [20, 29, 10], we use $1 \times 1$ convolutional layer and global average pooling at the top layers. To reduce the number of parameters in a plain CNN model. Based on a plain CNN, we develop our convolutional fusion networks, as illustrated in Fig. 1. Overall, our CFN mainly consists of three characteristics that will be described in the following.

Efficient side outputs. Instead of using the fully-connected layers, CFN efficiently generates the side branches from the intermediate layers while adding few parameters. First, the side branches are grown from the pooling layers by inserting $1 \times 1$ convolution layers like the main branch. All $1 \times 1$ convolutional layers must have the same number of channels so that they can be integrated together. Then, global average pooling is performed over the $1 \times 1$ convolutional maps to obtain one-dimensional feature vector, called GAP feature here. Notably, we can also consider the full depth main branch as a side branch.

Assume that there are $S$ of side branches in total and the last side branch (i.e. $S$-th) indicates the main branch. We notate $h_{i,j}^{(s)}$ as the input of $1 \times 1$ convolution in the $s$-th side branch, where $s = 1, 2, \ldots, S$ and $(i, j)$ is the spatial location across feature maps. As $1 \times 1$ convolution has $K$ of channels, its output associated with the $k$-th kernel, denoted as $f_{i,j,k}^{(s)}$, where $k = 1, \ldots, K$. Next, let $H^{(s)}$ and $W^{(s)}$ be the height and width of features maps derived from the $s$-th $1 \times 1$ convolution. Thereby, global average pooling performed over the feature map $f_{i,j,k}^{(s)}$ is calculated by

$$g_k^{(s)} = \frac{1}{H^{(s)}W^{(s)}} \sum_{i=1}^{H^{(s)}} \sum_{j=1}^{W^{(s)}} f_{i,j,k}^{(s)}, \quad (1)$$

Where $g_k^{(s)}$ is the $k$-th element in the $s$-th GAP feature vector. Thus, we can notate $g^{(s)} = [g_1^{(s)}, \ldots, g_K^{(s)}]$, a $1 \times K$ dimensional vector, as the whole GAP
Early fusion and late prediction. Considering when to fuse the side branches, related work [33,32] used an “early prediction and late fusion” (EPLF) strategy. In contrast to EPLF [33], in which a couple of FC layers are added, we present an opposite strategy called “early fusion and late prediction” (EFLP). EFLP can fuse the GAP features from the side outputs and obtain a fused feature. Then, one fully-connected layer following the fused feature is used to estimate the final prediction. Figure 2 shows the comparison between EPLF and EFLP. As compared to EPLF, EFLP consumes less parameters due to using only one fully-connected layer. We assume that each fully-connected layer has $C$ units that correspond to the number of object categories in the dataset. The fusion module has $W_{\text{fuse}}$ of parameters. Quantitatively, we can compare the parameters (i.e. weights and bias) between EFLP and EPLF by

$$W_{\text{EFLP}} = S(C + 1) + W_{\text{fuse}} < W_{\text{EPLF}} = SK(C + 1) + W_{\text{fuse}}.$$  

More importantly, the fused feature in EFLP can be extracted as a richer image representation, compared with the widely-used fc6 and fc7 [16,27]. The fused feature could be transferred from generic to specific vision recognition tasks. However, EPLF fails to specify which feature can serve as a good representation. Additionally, EFLP can achieve the same accuracy as EPLF, though EPLF consumes more parameters.

Locally-connected fusion. Another significant component in CFN is its fusing the branches based on a locally-connected (LC) layer. Owing to its no-sharing filters over spatial dimensions, LC layer can learn different weights in each local field [9]. We aim to make use of a LC layer to learn adaptive weights (or importance) for the side branches, and to generate the fused feature. As we know, this is the first attempt to apply a locally-connected layer to a fusion module. The detail computation are introduced as follows.

At first, we stack GAP features together (from $g^{(1)}$ to $g^{(S)}$), and form a stack layer $G$ with size of $1 \times K \times S$, see Fig. 1. The $s$-th feature map of $G$ is $g^{(s)}$. Then, one LC layer which has $K$ of no-sharing filters is convolved over $G$. Each filter has $1 \times 1 \times S$ kernel size. As a result, LC can learn adaptive weights for different elements in the GAP features. Here, the fused feature convolved by LC also has

Fig. 2: Comparison between EPLF and EFLP. (a) The pipeline of EPLF strategy; (b) The pipeline of EFLP strategy.
Fig. 3: Comparison of three fusion modules (best viewed in color). Left: Sum-pooling fusion has no weights; Middle: Convolution fusion learns sharing weights over spatial positions, as drawn in the same color; Right: Locally-connected fusion learns no-sharing weights over spatial positions, as drawn in different colors. To learn element-wise weights, we use $1\times 1$ local field.

$1 \times K$ shape, denoted as $g^{(f)}$. Each element in $g^{(f)}$ can be computed via

$$g^{(f)}_i = \sigma \left( \sum_{j=1}^{S} W^{(f)}_{i,j} \cdot g^{(j)}_i + b^{(f)}_i \right), \quad (3)$$

where $i = 1, 2, \ldots, K$; $\sigma$ indicates the activation function (i.e. ReLU). $W^{(f)}_{i,j}$ and $b^{(f)}_i$ represent the weights and bias. The number of parameters in the LC fusion is $K \times (S + 1)$. These additional parameters benefit adaptive fusion while do not need any manual tuning.

To be clear, Figure 3 compares LC fusion with other simple fusion methods. In Fig. 3(a), the sum-pooling fusion simply sums up the side outputs without learning any weights. In Fig. 3(b), the convolution fusion can learn only one sharing filter over the whole spatial dimensions (as drawn in the same blue color). On the contrary, LC can learn independent weights over each local field (i.e. $1\times 1\times S$ size), as drawn in different colors in Fig. 3(c). Although LC fusion has a little more parameters than the sum-pooling and convolution fusion, these parameters are nearly negligible as compared to the whole network parameters.

### 3.2 Training

Since CFN has efficient forward propagation and backward propagation, it can maintain the ease of training as similar to CNN. Assume that $W$ indicates the set of all parameters learned in the CFN (including the LC fusion weights), and $\mathcal{L}$ is the total loss cost during training. To minimize the total loss, the partial derivative of the loss with respect to any weight will be recursively computed by the chain rule during the backward propagation [6]. Since the main components in our CFN model are the side branches, we will induce the detail computations of their partial derivatives. For notational simplicity, we consider each image independently in the following.

First, we compute the gradient of the loss cost with respect to the outputs of the side branches. As an example of the $s$-th side branch, the gradient of $\mathcal{L}$...
with respect to the side output \( g^{(s)} \) can be formulated as below

\[
\frac{\partial L}{\partial g^{(s)}} = \frac{\partial L}{\partial g^{(f)}} \cdot \frac{\partial g^{(f)}}{\partial g^{(s)}}, s = 1, 2, \ldots, S. \tag{4}
\]

Second, we formulate the gradient of \( L \) with respect to the inputs of the side branches. We notate \( a^{(s)} \) as the input of the \( s \)-th side branch. As depicted in Fig. 1, \( a^{(s)} \) represents the pooling layer. Note that the input of the main branch, denoted as \( a^{(S)} \), refers to the last convolutional layer (i.e. conv \( S \)). We can observe that the gradient of \( a^{(s)} \) depends on several related branches. For example in Fig. 1, the gradient of \( a^{(1)} \) is influenced by \( S \) of branches; the gradient of \( a^{(2)} \) need to consider the results from the 2-th to \( S \)-th branch; but the gradient of \( a^{(3)} \) is updated by only the main branch. Mathematically, the gradient of \( L \) with respect to the side input \( a^{(s)} \) can be computed as follows:

\[
\frac{\partial L}{\partial a^{(s)}} = \sum_{i=s}^{S} \frac{\partial L}{\partial g^{(i)}} \cdot \frac{\partial g^{(i)}}{\partial a^{(i)}}, \tag{5}
\]

where \( i \) indexes the related branch that contributes to the gradient of \( a^{(s)} \). We then need to sum up the gradients from these related branches. Like [16], we employ standard stochastic gradient descent (SGD) algorithm with mini-batch to train the whole network.

### 3.3 Discussion

To present more insights into CFN, we compare CFN with other related models.

**Relationship with CNN.** Normally, a plain CNN only estimates a final prediction based on the topmost layer, as a result, the effects of intermediate layers on the prediction are implicit and indirect. In contrast, CFN can connect the intermediate layers using side branches, and deliver their effects on the final prediction jointly. Hence, CFN can take advantage of intermediate layers explicitly and directly.

**Relationship with DSN.** DSN [18] adds extra supervision to intermediate layers for earlier guidance. However, CFN that still uses one supervision towards the final prediction aims to generate a fused and richer feature. In a nutshell, DSN focuses on “loss fusion”, but CFN instead focuses on “feature fusion”.

**Relationship with ResNet.** ResNet [10] addresses the vanishing gradient problem by adding “linear” shortcut connections. CFN has three main differences as compared to ResNet: (1) The side branches in CFN are not shortcut connections. They start from a pooling layer and end in a fusion module together. (2) In contrast to adding a “linear” branch, we still use ReLU in each side branch. (3) The output of a fusion module is fed to the final prediction. As mentioned in the ResNet work, when the network is not overly deep (e.g. 11 or 18 layers), ResNet may obtain little improvement over a plain CNN. However, CFN can obtain some considerable improvements as compared to CNN. In contrast to increasing the depth, CFN can serve as an alternative to improving the discriminative capacity of not-very-deep models. This explains the usefulness and effectiveness of CFN.
4 Experiments

First, we trained CFN models on the CIFAR-10/100 [15] and ImageNet 2012 [26]. Then, we transferred the trained ImageNet model to three new tasks, including scene recognition, fine-grained recognition and image retrieval. We conducted all experiments using the Caffe framework [13] with a NVIDIA TITAN X card.

4.1 CIFAR dataset

Both CIFAR-10 [15] and CIFAR-100 [15] consist of 50,000 training images and 10,000 testing images. But they define 10 and 100 object categories, respectively. We preprocessed their RGB images by global contrast normalization [7]. We built a plain CNN that consists of seven convolutional layers and one fully-connected layer. The first six convolutional layers have 3×3 kernel size, but the seventh one is 1×1 convolution. Global average pooling locates between the last convolutional layer and the fully-connected layer. Based on the plain CNN, we developed the CFN counterpart as illustrated in Fig. 4.

Overall, we use the same hyper-parameters to train CNN and CFN. We use a weight decay of 0.0001, a momentum of 0.9, and a mini-batch size of 100. The learning rate is initialized with 0.1 and is divided by 10 after 10×10^4 iterations. The whole training will be terminated after 12×10^4 iterations. As for CFN, the initialized weights in LC fusion is set with 1/S (S is the number of branches).

Results. Table 1 shows the results on CIFAR-10 test set. We can analyze the results from the following aspects: (1) Compared with the plain CNN, CFN achieves about 1.01% and 1.21% improvement on CIFAR-10 and CIFAR-100, respectively. (2) In order to demonstrate the advantage of LC fusion, we also implement the sum-pooling fusion and convolution fusion, denoted as CNN-Sum and CNN-Conv. We can see that LC fusion used in CFN outperforms both CNN-Sum and CNN-Conv. (3) We compute the number of parameters in each

| Model       | #parameters          | CIFAR-10      | CIFAR-100     |
|-------------|----------------------|---------------|---------------|
| CNN         | 1.287M (basic)       | 9.28          | 31.89         |
| CNN-Sum     | 1.287M + 0.074M (extra branches) + 0 (fusion) | 8.84          | 31.42         |
| CNN-Conv    | 1.287M + 0.074M (extra branches) + 4 (fusion)  | 8.68          | 31.16         |
| CFN         | 1.287M + 0.074M (extra branches) + 768 (fusion) | **8.27**      | **30.68**     |
model. Importantly, the additional parameters for extra side branches and LC fusion are significantly fewer than the number of basic parameters. Although LC fusion uses a little more parameters for fusing branches, these parameters are nearly negligible for a deep network. To reflect the efficiency, we also compare the training time between CNN and CFN. For example on CIFAR-10, CNN and CFN consumes 1.67 and 2.08 hours, respectively.

In Fig. 5, we visualize and compare the learned feature maps in CNN and CFN. We select ten images from CIFAR-10 dataset. We extract the feature maps in the $1 \times 1$ convolutional layer and visualize the top-4 maps (we rank the feature maps by averaging their own activations). One can observe that CFN can learn complementary clues in the side branches to the full depth main branch. For example, the side-output 1 mainly learns the boundaries or shapes around the objects. The side-output 2 focuses on some semantic “parts” that fire strong near the objects. Furthermore, Figure 6(a) shows the adaptive weights learned
Table 2: Test error on CIFAR-10/100 to compare CFN with recent state-of-the-art. A superscripted * indicates to use the standard data augmentation [18].

| Method                | CIFAR-10  | CIFAR-10* | CIFAR-100 |
|-----------------------|-----------|-----------|-----------|
| Maxout Networks [7]   | 11.68%    | 9.38%     | 38.57%    |
| NIN [20]              | 10.41%    | 8.81%     | 35.68%    |
| DSN [18]              | 9.69%     | 7.97%     | 34.54%    |
| ALL-CNN [28]          | 9.08%     | 7.25%     | 33.71%    |
| R-CNN [19]            | 8.69%     | 7.09%     | 31.75%    |
| NIN + SReLU [14]      | 8.41%     | 6.98%     | 31.10%    |
| CNN (baseline)        | 9.28%     | 7.34%     | 31.89%    |
| CFN (ours)            | 8.27%     | 6.77%     | 30.68%    |

in the LC fusion. The side branch 3 (main branch) plays a core role, but other side branches are also complementary to the full depth main branch.

Comparison with the state-of-the-art. Table 2 compares the results on CIFAR datasets. Overall, CFN can obtain comparative results and outperform recent not-very-deep state-of-the-art models. It is worth mentioning that some work intends to push the results using much deeper networks [10] and large data augmentation [8]. In contrast to purely pushing the results, our aim is to demonstrate the advantage of fusing intermediate layers. Thus we only use a not-overly deep model and standard data augmentation [18]. We believe that adapting CFN to a very deep model will be an interesting future work.

4.2 ImageNet 2012

We developed a basic 11-layer plain CNN (i.e. CNN-11) whose channels of feature maps range from 64 to 1024. Based on this CNN, we built its CFN counterpart (i.e. CFN-11) as illustrated in Fig. 7. We create three extra side branches from the pooling layers (excluding the first pooling layer). Following existing literature [16,27,29,10], we use a weight decay of 0.0001, and a momentum of 0.9 and a mini-batch size of 64. Batch normalization (BN) [11] is used after each convolution. The learning rate starts from 0.01 and decreases to 0.001 at $10 \times 10^4$ iterations, and to 0.0001 at $15 \times 10^4$ iterations. The whole training will be terminated after $20 \times 10^4$ iterations. LC weights are initialized with 0.25 due to four side branches in total.

Results. Table 3 compares the results on the validation set. First, CNN-11 can achieve competitive results as compared to AlexNet [16], however, it consumes much fewer parameters (~6.3 millions) than Alexnet (~60 millions).

Fig. 7: The architecture of CFN built for ImageNet classification.
Fig. 8: Illustration of the feature activations of images of the ImageNet dataset.

Table 3: Error rates (%) on the ImageNet 2012 validation set.

| Method       | AlexNet | CNN-11 | DSN-11 | ResNet-11 | CFN-11 | CNN-19 | CFN-19 |
|--------------|---------|--------|--------|-----------|--------|--------|--------|
| Top-1        | 42.90   | 43.11  | 42.24  | 43.02     | 41.96  | 36.99  | 35.47  |
| Top-5        | 19.80   | 19.91  | 19.24  | 19.85     | 19.09  | 14.74  | 13.93  |

Second, CFN-11 obtains about 1% improvement over CNN-11, while adding few parameters (∼0.5 millions). It verifies the efficiency of fusing multi-scale deep representations. Furthermore, we reproduce the DSN [18] and ResNet [10] models based on the plain CNN-11. As a result, CFN-11 can achieve better accuracy than DSN-11 and ResNet-11. For such a not-overly deep network, CFN can serve as an alternative to improving the discriminative capacity of CNNs, instead of increasing the depth like ResNet. Moreover, to test the generalization of CFN to deeper networks, we build a 19-layer model following the principle of 11-layer model. Likewise, CFN-19 outperforms CNN-19 by a consistent improvement as seen in Table 3.

Similar to CIFAR-10, Figure 6(b) shows the adaptive weights learned in the LC fusion. We can see that the top branches (i.e. 3 and 4) have larger weights than the bottom branches (i.e. 1 and 2). In Fig. 8, we illustrate and compare the feature maps in the side branches.

4.3 Transferring fused feature to new tasks

To evaluate the generalization of CFN, we transferred the trained ImageNet model to three new tasks: scene recognition, fine-grained recognition and image retrieval. Each task is evaluated on two datasets: Scene-15 [17] and Indoor-67 [25], Flower [23] and Bird [30], and Holidays [12] and UKB [24]. For AlexNet,
the fc7 layer is used as a baseline; For CNN-11, we extract the result of global average pooling as another baseline; For CFN-11, the fused feature is extracted to represent images. For scene and fine-grained recognition, we use linear SVM [4] to compute the classification accuracy. For image retrieval, we use KNN to compute the mAP on Holidays and N-S score on UKB. Table 4 reports the evaluation results on six datasets. We can see that CFN-11 obtains consistent improvement performance on all datasets. Interestingly, their gains are more remarkable than those in ImageNet. It reveals that learning multi-scale deep representations are beneficial for diverse vision recognition problems. In addition, fine-tuning the model on the target datasets will further improve the results.

5 Conclusions

We proposed efficient convolutional fusion networks (CFN) by adding few parameters. It can serve as an alternative to improving recognition accuracy instead of increasing the depth. Experiments on the CIFAR and ImageNet datasets demonstrate the superiority of CFN over the plain CNN. Additionally, CFN outperforms not-very-deep state-of-the-art models by considerable gains. Moreover, we verified its significant generalization while transferring CFN to three new tasks. In future work, we will evaluate CFN with much deeper neural networks.

Acknowledgments
This work was supported mainly by the LIACS Media Lab at Leiden University and in part by the China Scholarship Council. We would like to thank NVIDIA for the donation of GPU cards.

References

1. Agrawal, P., Girshick, R., Malik, J.: Analyzing the performance of multilayer neural networks for object recognition. In: ECCV (2014)
2. Azizpour, H., Razavian, A.S., Sullivan, J., Maki, A., Carlsson, S.: From generic to specific deep representation for visual recognition. In: CVPR, DeepVision workshop (2015)
3. Babenko, A., Lempitsky, V.S.: Aggregating deep convolutional features for image retrieval. In: ICCV (2015)
4. Chang, C.C., Lin, C.J.: LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology 2, 27:1–27:27 (2011)
5. Cimpoi, M., Maji, S., Vedaldi, A.: Deep filter banks for texture recognition and segmentation. In: CVPR (2015)
6. Cun, L., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Handwritten digit recognition with a back-propagation network. In: NIPS (1990)
7. Goodfellow, I.J., Warde-Farley, D., Mirza, M., Courville, A.C., Bengio, Y.: Maxout networks. In: ICML (2013)
8. Graham, B.: Fractional max-pooling. CoRR abs/1412.6071 (2014)
9. Gregor, K., LeCun, Y.: Emergence of complex-like cells in a temporal product network with local receptive fields. CoRR abs/1006.0448 (2010)
10. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR (2016)
11. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: ICML (2015)
12. Jegou, H., Douze, M., Schmid, C.: Hamming embedding and weak geometric consistency for large scale image search. In: ECCV (2008)
13. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe: Convolutional Architecture for Fast Feature Embedding. In: ACM Multimedia (2014)
14. Jin, X., Xu, C., Feng, J., Wei, Y., Xiong, J., Yan, S.: Deep learning with s-shaped rectified linear activation units. In: AAAI (2016)
15. Krizhevsky, A.: Learning multiple layers of features from tiny images. Master’s thesis, Department of Computer Science, University of Toronto. (2009)
16. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: NIPS (2012)
17. Lazebnik, S., Schmid, C., Ponce, J.: Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In: CVPR (2006)
18. Lee, C., Xie, S., Gallagher, P., Zhang, Z., Tu, Z.: Deeply-supervised nets. In: AISTATS (2015)
19. Liang, M., Hu, X.: Recurrent convolutional neural network for object recognition. In: CVPR (2015)
20. Lin, M., Chen, Q., Yan, S.: Network in network. In: ICLR (2014)
21. Liu, L., Shen, C., van den Hengel, A.: The treasure beneath convolutional layers: cross convolutional layer pooling for image classification. In: CVPR (2015)
22. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: CVPR (2015)
23. Nilsback, M.E., Zisserman, A.: Automated flower classification over a large number of classes. In: Indian Conference on Computer Vision, Graphics and Image Processing (2008)
24. Nister, D., Stewenius, H.: Scalable recognition with a vocabulary tree. In: CVPR (2006)
25. Quattoni, A., Torralba, A.: Recognizing indoor scenes. In: CVPR (2009)
26. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge. IJCV pp. 1–42 (2015)
27. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: ICLR (2015)
28. Springenberg, J.T., Dosovitskiy, A., Brox, T., Riedmiller, M.: Striving for simplicity: The all convolutional net. In: ICLR (2015)
29. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: CVPR (2015)
30. Wah, C., Branson, S., Welinder, P., Perona, P., Belongie, S.: The Caltech-UCSD Birds-200-2011 Dataset. Tech. Rep. CNS-TR-2011-001 (2011)
31. Wei, X.S., Gao, B.B., Wu, J.: Deep spatial pyramid ensemble for cultural event recognition. In: ICCV Workshops (2015)
32. Xie, S., Tu, Z.: Holistically-nested edge detection. In: ICCV (2015)
33. Yang, S., Ramanan, D.: Multi-scale recognition with DAG-CNNs. In: ICCV (2015)
34. Yoo, D., Park, S., Lee, J.Y., Kweon, I.S.: Multi-scale pyramid pooling for deep convolutional representation. In: CVPR, Deep Vision workshop (2015)
35. Yue-Hei Ng, J., Yang, F., Davis, L.S.: Exploiting local features from deep networks for image retrieval. In: CVPR, Deep Vision workshops (June 2015)