Transfer Learning with Sparse Associative Memories

Quentin Jodelet¹,², Vincent Gripon¹, and Masafumi Hagiwara²

¹ IMT Atlantique, Technopole Brest Iroise, 29238 Brest, France
² Keio University, Yagami Campus, 223-8522 Yokohama, Japan

Abstract. In this paper, we introduce a novel layer designed to be used as the output of pre-trained neural networks in the context of classification. Based on Associative Memories, this layer can help design Deep Neural Networks which support incremental learning and that can be (partially) trained in real time on embedded devices. Experiments on the ImageNet dataset and other different domain specific datasets show that it is possible to design more flexible and faster-to-train Neural Networks at the cost of a slight decrease in accuracy.

Keywords: Neural Networks · Associative Memories · Self-organizing Maps · Deep Learning · Transfer Learning · Computer Vision.

1 Introduction

During the past decade, Deep Neural Networks, and more specifically Deep Convolutional Neural Networks have been established as the state-of-the-art solution for various problems of Computer Vision such as object recognition [17,30,12,31], image segmentation [21,6,27,11] and object tracking [32,2].

A standard Deep Neural Network relies on millions of trained parameters and thus requires millions of floating point operations in order to compute the output corresponding to a given input. Consequently, the use of Deep Neural Networks for inference in real time tasks requires massive computing power and large amounts of memory. However embedded devices have important limitations in terms of computing power, and memory and battery usage, so that Deep Neural Networks are difficult to implement. Many researches have been carried out in order to produce faster Deep Neural Networks for run-time using pruning [19,10], quantization [3,13,79], binarization [3,25,38] and new specific architectures have also been developed specifically for real-time execution [26] or embedded systems [13,28].

Even more complex is the training procedure, which requires going through a large dataset multiple times. As of today, this procedure is generally performed offline using specific hardware such as GPUs or TPUs. Moreover, the training procedure is not incremental, so that adding new elements to the dataset (or new classes) can be handled only by restarting the training from scratch [7]. In order to benefit from the accuracy of deep neural networks without having to train
them, a common solution is to rely on transfer learning [23]. In the context of computer vision, transfer learning consists in using Deep Neural Networks pre-trained on a large dataset such as ImageNet [4], Microsoft COCO [20] or Google OpenImages [18], in order to obtain a generic image representation for other tasks. It is then possible to address new classification tasks on a different dataset by using pre-trained models as feature extractors, which may then be fine-tuned and combined with a simple, sometimes incremental, classifier [5,29,36,1]

In this paper, by using Self-Organizing Maps and Associative Memories, we propose a new Neural Network model meant to be used for classification tasks using transfer learning with pre-trained Deep Neural Networks. We show it can help design flexible Deep Neural Network supporting incremental learning that can be trained in real-time on embedded devices.

2 Self-Organizing Maps and Sparse Associative Memories

2.1 Self-Organizing Maps

Presentation A Self-Organizing Map (SOM) [16] is a fully connected layer of $N$ neurons that associates a $d$-dimensional input vector $\mathbf{x}$ with an $N$-dimensional output vector $\mathbf{q}(\mathbf{x})$. These neurons are organized on a 2D-grid of $q$ by $r$ units in a way that each neuron but those on the edges has 4 direct neighbors. All these neurons are entirely connected with the $d$ neurons of the previous layer. The weights corresponding to a given neuron $i$ are denoted $\mathbf{w}_i$. Figure 1 depicts an example of such an input layer and a map layer.

![Fig. 1. Depiction of a SOM layer.](image-url)
Transfer Learning with Sparse Associative Memories

Inference

When an input vector \( x \) is presented to the map layer, the corresponding output is computed as a vector \( q(x) \in \{0, 1\}^N \) containing a single 1. The coordinate \( i^* \) which value is 1 is defined as \( i^* = \arg \min_{i=1}^{N} \text{dist}(w_i, x) \). This \( i^* \)-th neuron is referred to as the Best-Matching Unit (BMU). Note that when the vector \( x \) and the vectors \( w_i \) all have a unit norm, the dynamics of SOMs is equivalent to:

\[
q(x) = h(W \cdot x)
\]

where \( h \) is a Winner-Takes-All (WTA) operator (all values are put to 0 except for the maximal one, which is put to 1) and \( W \) is the matrix which lines are the vectors \( w_i \).

Training

In contrast with classic fully-connected layers used in deep neural networks, SOMs are built so that neighbor units contain strong inner dependencies, as explained below. The learning algorithm is performed for a specific number of epoch \( E \) and a specific batch size. The parameters \( w_i \) are first initialized at randoms. The learning procedure is then performed by iterating the following operations for \( t \) from 0 to \( E \):

1. The training set \( \mathcal{X} \) is randomly shuffled.
2. For each input vector \( x \) in \( \mathcal{X} \), the corresponding output vector \( y \) is computed.
   Denoting \( i^* \) as the neuron where \( y(i^*) = 1 \), we perform the following update of all weights:

\[
\forall i, w_i \leftarrow w_i + (x - w_i)A(t)\Theta(t, i^*, i)
\]

where \( A \) is the learning decay function expressed as \( A(t) = \alpha T(t) \) with \( \alpha \) is the learning rate and \( T \) is a function which decreases with \( t \) that can be expressed as \( T(t) = 1 - \frac{t}{T} \) or \( T(t) = e^{-\frac{t}{T}} \); \( \Theta \) the neighborhood function which decreases with \( t \) and the distance in the grid between neurons \( i \) and \( i' \) defined as \( \Theta(t, i, i') = e^{-\frac{d_{i,i'}^2}{2T(t)^2}} \) where \( T \) is the decreasing function defined above and \( d_{i,i'} \) is the distance between the \( i \)-th and the \( i' \)-th neurons in the grid independently of their associated vectors \( w_i \) and \( w_j \).

Quantizing with multiple SOMs

A popular way to quantize a vector is to use Product Quantization [14] (PQ). PQ consists in the following: a) splitting the input vectors into \( k \) distinct subparts and b) quantizing each part individually and independently from the others. The term “product” comes from the fact that the anchor vectors in the initial space are the Cartesian product of the anchor vectors in each subspace.

In this section we propose to use multiple SOMs in order to perform PQ, using one for each subspace. The study is restricted to the case where vectors in each subspace all have the same dimension and where each SOM contains the same number \( N \) of neurons, such that the number of anchor vectors in the product (initial) space is \( N^k \).
Concretely, let us consider $dk$-dimensional input vectors. Our methodology is summarized as follows:

- **Training:**
  1. Initialize $k$ SOMs with input dimension $d$ (indexed from 1 to $k$). They each contain $N$ neurons,
  2. Split training vectors regularly into $k d$-dimensional subvectors each. The first subvector of each train vector are used to train the first SOM, the second subvector of each train vector to train the second SOM etc.

- **Quantizing:**
  1. Split the input vector $x$ into the $k$ corresponding subvectors,
  2. For each subvector, obtain the corresponding output subvector using the associated SOM,
  3. Concatenate the output subvectors to obtain a $kN$-dimensional binary vector containing exactly $k$ 1s, denoted $Q(x)$.

### 2.2 Sparse Associative Memories

Sparse Associative Memories (SAMs) are Neural Networks able to store and retrieve sparse patterns from incomplete inputs. They consist of a Neural Network made of $p$ distinct groups composed of variable numbers of units bound by binary connections. SAMs are able to store patterns which have exactly one active neuron in each group. Although not optimized for this problem, SAMs are also able to retrieve “close” patterns, meaning that some initially active neurons are changed during the retrieving procedure to find a stored pattern.

The learning procedure is as follows: The connections between the neurons are initialized empty; then for each pattern to store, the corresponding connections are added to the network. Since connections are binary (they either exist or not), they are not reinforced if shared by more than a single pattern.

The retrieving procedure starts from a partial pattern, meaning that some of its active neurons are initially not activated. Then, an iterative procedure is started. This procedure consists in finding in each group of neurons, the neuron (or the neurons) that has the maximum number of connections with the active neurons. Hopefully, after a few iterations, the network stabilizes to the stored pattern.

### 2.3 Proposed model: combining SOMs and SAMs

**Presentation** We propose a new model by combining SAMs with multiple SOMs. The proposed model is composed of $p = k + 1$ groups, where the first $k$ groups correspond to $k$ SOMs composed of $N$ neurons each, and the last group is the output layer containing $M$ neurons. Since we are only interested in finding the active neuron in the last group, connections between the first $k$ groups are ignored and the retrieving procedure is not iterated.

The proposed model has two hyper-parameters: $k$ the number of SOMs and $N$ the number of neurons on each SOM (as stated before, we assume that each SOM has the same number of neurons). For reasons of clarity, we denote $kxN$-AL the proposed model composed of $k$ SOMs with $N$ neurons on each.
**Training** The proposed model is expressed as $k$ weight matrix $W$ corresponding to each SOM and 1 sparse matrix $\Omega$ representing the connections between the $k$ SOMs and the output layer. We denote $X$ the training set containing pairs $(x, y)$, where $x$ is a $dk$-dimensional vector and $y$ is an integer value between 1 and $M$ corresponding to the label. The learning procedure consists in two distinct steps:

1. Training the multiple SOMs as described in the section 2.1 on the training dataset $X$ in order to compute the weight matrix $W$ associated with each SOM.
2. $\Omega$ the binary matrix made of $M$ lines and $kN$ columns, initially containing only 0s. Then for each couple $(x, y)$ in the training set $X$, the following equation is obtained:

$$\Omega = \max_{(x, y) \in X} e_y \cdot Q(x)$$

where $\top$ is the transpose operator, the max is applied componentwise, $e_\ell$ is the vector of size $M$ containing only 0s except for a 1 at the $\ell$-th coordinate and $Q$ is the quantization function which uses the $k$ SOMs of the model as defined in the section 2.1.

**Inference** The retrieving procedure is also twofold. By considering an input vector $x'$, the following prediction can be obtained by:

$$h(\Omega \cdot Q(x'))$$

where $h$ is an activation function, the Winner-Takes-All (WTA) function is the most commonly used.

When used this way, the sparse associative memory basically emulates a majority vote among the multiple SOMs.

### 3  Associative Memories used as classifier

#### 3.1  Presentation

The model composed of multiple SOMs and one SAM proposed in the previous section has been designed to be used as a new neural network classifying layer of a conventional Deep Neural Network.

The proposed layer has to be trained independently from the previous layers of the Neural Network. As it has been conceived to rely on Transfer Learning, it has to be used as the output layer of a Deep Neural Network pre-trained on a universal dataset, such as ImageNet for image classification, whose last layer has been removed.
3.2 Training and inference

We denote $U$ the universal training dataset containing pairs $(x, y)$, where $x$ is a training vector and $y$ is the corresponding label, and $U'$ the dataset containing pairs $(x', y)$ where $x'$ is a k-dimensional vector obtained as the output when $x$ is passed as the input to the pre-trained Deep Neural Network which last layer has been removed. Similarly, we denote $T$ the specific training set and $T'$.

As described in section 2.3, the first step, which must be done ahead, is to train the SOMs of the proposed model on the universal dataset $U'$, then the second step consists in updating the matrix $\Omega$ using the training set $T'$. One of the strengths of this learning procedure is that it is simple: once the SOMs have been trained on the universal dataset, learning a new element is limited to two matrix products and non-linear functions if the input of the layer is a unit vector. The learning of each element is independent, and thus it is possible to learn new elements in parallel and incrementally, without restarting the learning procedure from the initial state. This makes this training algorithm particularly suitable for being used on the edge on devices with limited computation power.

The inference of the proposed model, as explained in section 2.3, takes advantage of the high sparsity of the matrix $\Omega$ and the fact that both $\Omega$ and $Q(x)$ are binary variables, implying that the computation of the product of both can be highly optimized during implementation.

3.3 Improvement

It is important to notice that a lot of information is lost due to the process of the quantization and the binary representation of elements inside the matrix $\Omega$.

We propose a solution, denoted $k\times N$-IAL, to mitigate the loss related to the second point. The idea is to no longer restrain the matrix $\Omega$ to binary values but to use integer values instead. In this case, the learning procedure is expressed as:

$$\Omega = \sum_{(x,y) \in \mathcal{X}} e_y \cdot Q(x)^\top$$

However, the inference procedure remains unchanged. Although it is no longer possible to take advantage of binary operations to optimize the implementation, integer operations remain faster than floating-point operations.

4 Experiments

4.1 Protocol

In order to evaluate our method, the proposed model has been combined with a VGG-16 model trained on ImageNet whose dense classification layer has been removed. In all experiments below, SOMs composing the proposed model has been trained during 50 epochs using linear decreasing learning decay function and, in order to be able to express the complete layer as matrix products, input
of the layer are normed: this does not significantly impact the accuracy of the model in this case.

In the following subsection, the proposed model is evaluated on the large-scale dataset ImageNet and on several small domain-specific datasets.

### 4.2 Experiments on ImageNet

The first series of experiments consist in comparing the accuracy of the proposed model on the classification of the large-scale dataset ImageNet depending on the different hyper-parameters of the model.

![Figure 2](image2.png)

**Fig. 2.** Comparison of the Top-5 accuracy of the proposed layer on classification task on ImageNet depending on the number of SOMs composing the model.

![Figure 3](image3.png)

**Fig. 3.** Comparison of the Top-5 accuracy of the proposed layer on classification task on ImageNet depending on the number of neurons on each SOM composing the model.

Figure 2 and the Figure 3 compare the Top-5 accuracy of different versions of the proposed model on the classification of ImageNet. Figure 2 compares versions of the proposed model which each has a different number of SOMs and Figure 3 compares versions of the proposed model which each has a different number of neurons in each SOM. It appears that, both the increase of the number of SOMs composing the model and the increase of the number of neurons in each SOM will increase the accuracy of the model. This is tied to the fact that they will reduce the loss of information induced by the quantization. However, both parameters are directly correlated to the computation and memory costs of both the training procedure and the inference procedure of the layer. Thus, increasing one of the two will also inevitably increase the cost of its use along with the accuracy.

The computational cost of both the training procedure and the inference procedure of the proposed model is correlated to the total number of neurons in the layer. This total number of neurons is simply the sum of the number of
neurons in each SOM composing the proposed model. However, increasing the number of neurons is not necessarily synonymous with increasing the accuracy of the model. As shown in Figure 4, the 32x256-AL is outperformed by the 64x100-AL even though the latter has 28% more neurons than the first one. In general, for a fixed number of neurons, the model using the largest number of sub-layers should be preferred.

Figure 5 compares the original proposed layer with the improved version which uses integer values instead of binary values. As expected, the integer model is more accurate than the binary one in this case. In fact the integer model is more efficient for dataset containing a lot of training elements and a lot of classes such as ImageNet; while the binary model is more efficient on smaller datasets containing less training elements and classes.

Figure 6 summarizes the experiments on the large-scale dataset ImageNet by comparing the proposed layer with VGG-16. The selected version of the proposed layer is the integer values model composed of 128 sub-layers of 100 neurons each. On the full classification problem (1000 classes, 1.2 millions learned elements), the proposed layer reaches 78.9% accuracy Top-5 and 53.1% accuracy Top-1 while VGG-16 reaches 90.0% and 70.5%, respectively. This decrease in accuracy is traded for more flexible Neural networks which support incremental learning and which use a faster learning algorithm.

4.3 Experiments on domain specific datasets

In the second serie of experiments, the proposed model has been evaluated on several domain specific dataset: 102 Category Flower Dataset [22] (denoted by Flower102), the Indoor Scene Recognition Dataset [24] (denoted by Indoor67), the Caltech-UCSD Birds 200 dataset [33] (denoted by CUB200), the Stanford
Dogs Dataset [15] (denoted by Dog120) and the Stanford 40 Actions Dataset [35] (denoted by Stanford40).

In order to evaluate our method, for each dataset, the proposed model is compared with both a simple deep neural network classifier and a SVM. The C-SVM has been used and it has been trained using a One-vs-Rest policy. The neural networks have been trained using stochastic gradient descent. In order to compare the three solutions, it has been decided to not use augmentation on training data or fine-tuning of previous layers of the pre-trained VGG-16 model. Using these techniques will improve the performance of every method. The results of this experiment are shown in Table 1.

Table 1. Comparison of the accuracy of 128x100-AL with different transfer learning methods on classification task on different domain specific datasets.

| Dataset       | NN classifier Top-1 | SVM Top-1     | Proposed model Top-1 |
|---------------|---------------------|---------------|----------------------|
| Flower102     | 68.52 %             | 73.80 %       | 66.92 %              |
|               | 88.36 %             | 90.94 %       | 87.48 %              |
| Indoor67      | 65.07 %             | 67.24 %       | 56.42 %              |
|               | 90.30 %             | 90.90 %       | 87.54 %              |
| CUB200        | 35.97 %             | 38.02 %       | 41.51 %              |
|               | 63.01 %             | 67.66 %       | 76.18 %              |
| Dog120        | 74.69 %             | 76.35 %       | 71.91 %              |
|               | 95.07 %             | 95.90 %       | 95.34 %              |
| Stanford40    | 68.11 %             | 69.54 %       | 59.06 %              |
|               | 90.71 %             | 91.68 %       | 87.02 %              |

The proposed model is slightly inferior to the others methods in terms of performance on the tested datasets except for the CUB200 dataset. However,
Table 2. Comparison of the training time of 128x100-AL with different classifiers on different domain specific datasets. Training time duration are expressed relatively to the one of the proposed model. The lower the value, the better it is.

| Dataset  | NN classifier | SVM | Proposed model |
|----------|---------------|-----|----------------|
| Flower102| 33            | 73  | 1              |
| Indoor67 | 17            | 99  | 1              |
| CUB200   | 27            | 79  | 1              |
| Dog120   | 11            | 193 | 1              |
| Stanford40 | 18         | 70  | 1              |

due to the learning algorithm used, it is considerably faster to train as shown in Table 2. These time performance measures have been done on a single core general-purpose CPU. In order to moderate these results, it may be possible to decrease a bit the training time of the other models by using a more aggressive early stop function but the order of magnitude will remains similar: generally the proposed model is 10 time faster to train than the others. These results suggest that the training algorithms of the proposed model could be executed on the edge on embedded device with limited resources.

5 Conclusion

We introduced a new classifier model primarily composed of Self-Organizing Maps and Sparse Associative Memories. By combining the proposed layer with a pre-trained Deep Neural Network, it is possible to design flexible Deep Neural Networks with a faster learning algorithm and the support of incremental learning at the cost of a slight decrease of the accuracy. Using the proposed model could help the development and the deployment of new intelligent embedded devices which can learn new elements in real time and adapt themselves to their environment without the help of an external agent responsible for the execution of the training algorithm.

References

1. Azizpour, H., Razavian, A.S., Sullivan, J., Maki, A., Carlsson, S.: Factors of transferability for a generic convnet representation. IEEE transactions on pattern analysis and machine intelligence 38(9), 1790–1802 (2016)
2. Bertinetto, L., Valmadre, J., Henriques, J.F., Vedaldi, A., Torr, P.H.: Fully-convolutional siamese networks for object tracking. In: European conference on computer vision. pp. 850–865. Springer (2016)
3. Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. arXiv preprint arXiv:1602.02830 (2016)
4. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition. pp. 248–255. Iee (2009)
5. Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., Darrell, T.: Decaf: A deep convolutional activation feature for generic visual recognition. In: International conference on machine learning. pp. 647–655 (2014)
6. Girshick, R.: Fast r-cnn. In: Proceedings of the IEEE international conference on computer vision. pp. 1440–1448 (2015)
7. Goodfellow, I.J., Mirza, M., Xiao, D., Courville, A., Bengio, Y.: An empirical investigation of catastrophic forgetting in gradient-based neural networks. arXiv preprint arXiv:1312.6211 (2013)
8. Gripon, V., Berrou, C.: Sparse neural networks with large learning diversity. IEEE transactions on neural networks 22(7), 1087–1096 (2011)
9. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149 (2015)
10. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Advances in neural information processing systems. pp. 1135–1143 (2015)
11. He, K., Gkioxari, G., Dollár, P., Girshick, R.: Mask r-cnn. In: Computer Vision (ICCV), 2017 IEEE International Conference on. pp. 2980–2988. IEEE (2017)
12. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
13. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017)
14. Jégou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. IEEE Transactions on Pattern Analysis and Machine Intelligence 33(1), 117–128 (Jan 2011). https://doi.org/10.1109/TPAMI.2010.57
15. Khosla, A., Jayadevaprakash, N., Yao, B., Fei-Fei, L.: Novel dataset for fine-grained image categorization. In: First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition. Colorado Springs, CO (June 2011)
16. Kohonen, T.: The self-organizing map. Proceedings of the IEEE 78(9), 1464–1480 (Sept 1990). https://doi.org/10.1109/5.588325
17. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems 25, pp. 1097–1105. Curran Associates, Inc. (2012)
18. Kuznetsova, A., Rom, H., Aldrin, N., Uijlings, J., Krasin, I., Pont-Tuset, J., Kamali, S., Popov, S., Malladi, M., Duerig, T., Ferrari, V.: The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. arXiv:1811.00982 (2018)
19. LeCun, Y., Denker, J.S., Solla, S.A.: Optimal brain damage. In: Touretzky, D.S. (ed.) Advances in Neural Information Processing Systems 2, pp. 598–605. Morgan-Kaufmann (1990)
20. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft coco: Common objects in context. In: European conference on computer vision. pp. 740–755. Springer (2014)
21. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 3431–3440 (2015)
22. Nilsback, M.E., Zisserman, A.: Automated flower classification over a large number of classes. In: Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing (Dec 2008)
23. Pan, S.J., Yang, Q.: A survey on transfer learning. IEEE Trans. on Knowl. and Data Eng. 22(10), 1345–1359 (Oct 2010). https://doi.org/10.1109/TKDE.2009.191
24. Quattoni, A., Torralba, A.: Recognizing indoor scenes. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition. pp. 413–420. IEEE (2009)
25. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: Xnor-net: Imagenet classification using binary convolutional neural networks. In: European Conference on Computer Vision. pp. 525–542. Springer (2016)
26. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 779–788 (2016)
27. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. In: Advances in neural information processing systems. pp. 91–99 (2015)
28. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4510–4520 (2018)
29. Sharif Razavian, A., Azizpour, H., Sullivan, J., Carlsson, S.: Cnn features off-the-shelf: an astounding baseline for recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition workshops. pp. 806–813 (2014)
30. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
31. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2818–2826 (2016)
32. Wang, L., Ouyang, W., Wang, X., Lu, H.: Visual tracking with fully convolutional networks. In: 2015 IEEE International Conference on Computer Vision (ICCV). pp. 3119–3127 (Dec 2015). https://doi.org/10.1109/ICCV.2015.357
33. Welinder, P., Branson, S., Mita, T., Wah, C., Schroff, F., Belongie, S., Perona, P.: Caltech-UCSD Birds 200. Tech. Rep. CNS-TR-2010-001, California Institute of Technology (2010)
34. Wu, J., Leng, C., Wang, Y., Hu, Q., Cheng, J.: Quantized convolutional neural networks for mobile devices. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 4820–4828 (June 2016). https://doi.org/10.1109/CVPR.2016.521
35. Yao, B., Jiang, X., Khoela, A., Lin, A.L., Guibas, L., Fei-Fei, L.: Human action recognition by learning bases of action attributes and parts. In: 2011 International Conference on Computer Vision. pp. 1331–1338. IEEE (2011)
36. Yosinski, J., Clune, J., Bengio, Y., Lipson, H.: How transferable are features in deep neural networks? In: Advances in neural information processing systems. pp. 3320–3328 (2014)
37. Zhou, A., Yao, A., Guo, Y., Xu, L., Chen, Y.: Incremental network quantization: Towards lossless cnns with low-precision weights. arXiv preprint arXiv:1702.03844 (2017)
38. Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., Zou, Y.: Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv preprint arXiv:1606.06160 (2016)