Multi-scale recognition with DAG-CNNs

Songfan Yang
College of Electronics and Information Engineering,
Sichuan University, China
syang@scu.edu.cn

Deva Ramanan
Robotics Institute,
Carnegie Mellon University, USA
deva@cs.cmu.edu

Abstract

We explore multi-scale convolutional neural nets (CNNs) for image classification. Contemporary approaches extract features from a single output layer. By extracting features from multiple layers, one can simultaneously reason about high, mid, and low-level features during classification. The resulting multi-scale architecture can itself be seen as a feed-forward model that is structured as a directed acyclic graph (DAG-CNNs). We use DAG-CNNs to learn a set of multi-scale features that can be effectively shared between coarse and fine-grained classification tasks. While fine-tuning such models helps performance, we show that even "off-the-self" multi-scale features perform quite well. We present extensive analysis and demonstrate state-of-the-art classification performance on three standard scene benchmarks (SUN397, MIT67, and Scene15). In terms of the heavily benchmarked MIT67 and Scene15 datasets, our results reduce the lowest previously-reported error by **23.9%** *and* **9.5%***, respectively.*

1. Introduction

Deep convolutional neural nets (CNNs), pioneered by Lecun and collaborators [19], now produce state-of-the-art performance on many visual recognition tasks [17, 30, 33]. An attractive property is that it appear to serve as universal feature extractors, either as "off-the-shelf" features or through a small amount of "fine tuning". CNNs trained on particular tasks such as large-scale image classification [5] *transfer* extraordinarily well to other tasks such as object detection [11], scene recognition [40], image retrieval [12], etc [28].

Hierarchical chain models: CNNs are hierarchical feed-forward architectures that compute progressively invariant representations of the input image. However, the appropriate level of invariance might be task-dependent. Distinguishing people and dogs requires a representation that is robust to large spatial deformations, since people and dogs can articulate. However, fine-grained categorization of car models (or bird species) requires fine-scale features that capture subtle shape cues. We argue that a universal architecture capable of both tasks must employ some form of multi-scale features for output prediction.

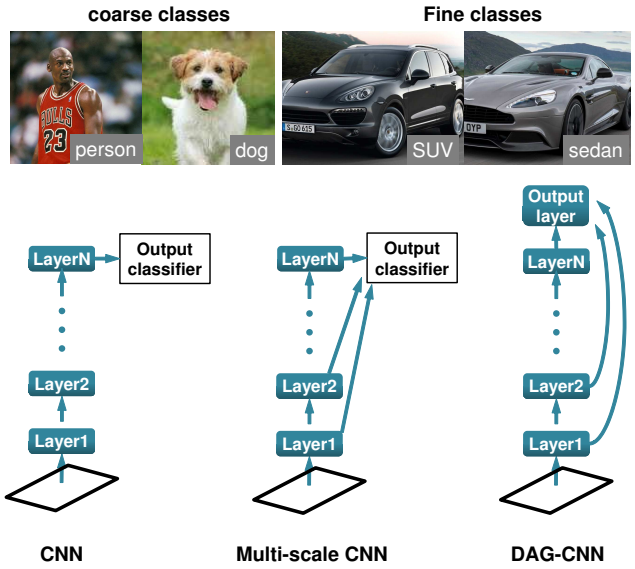


Figure 1. Recognition typically require features at multiple scales. Distinguishing a person versus dog requires highly invariant features robust to the deformation of each category. On the other hand, fine-grained recognition likely requires detailed shape cues to distinguish models of cars (**top**). We use these observations to revisit deep convolutional neural net (CNN) architectures. Typical approaches train a classifier using features from a single output layer (**left**). We extract multi-scale features from multiple layers to simultaneously distinguish coarse and fine classes. Such features come "for free" since they are already computed during the feed-forward pass (**middle**). Interestingly, the entire multi-scale predictor is still a feed-forward architecture that is no longer chain-structured, but a directed-acyclic graph (DAG) (**right**). We show that DAG-CNNs can be discriminatively trained in an end-to-end fashion, yielding state-of-the-art recognition results across various recognition benchmarks.

Multi-scale representations: Multi-scale representations are a classic concept in computer vision, dating back to image pyramids [4], scale-space theory [21], and multi-

(a) mid-level feature is preferred

(b) high-level feature is preferred

Figure 2. Retrieval results using L2 distance for both mid- and high-level features on MIT67 [26], computed from layer 11 and 20 of the Caffe model. *Green* (*Red*) box means correct (wrong) results, in terms of the scene category label. The correct label for wrong retrievals are provided. The retrieval results are displayed such that the left-most image has the closest distance to the query, and vice versa. Certain query images (or categories) produce better matches with high-level features, while others produce better results with mid-level features. This motivates our multi-scale approach.

resolution models [23]. Though somewhat fundamental notions, they have not been tightly integrated with contemporary feed-forward approaches for recognition. We introduce multi-scale CNN architectures that use features at multiple scales for output prediction (Fig. 1). From one perspective, our architectures are quite simple. Typical approaches train a output predictor (e.g., a linear SVM) using features extracted from a single output layer. Instead, one can train an output predictor using features extracted from *multiple* layers. Note that these features come "for free"; they are already computed in a standard feed-forward pass.

Spatial pooling: One difficulty with multi-scale approaches is feature dimensionality - the total number of features across all layers can easily reach hundreds of thousands. This makes training even linear models difficult and prone to over-fitting. Instead, we use marginal activations computed from sum (or max) pooling across spatial locations in a given activation layer. From this perspective, our models are similar to those that compute multi-scale features with spatial pooling, including multi-scale templates [10], orderless models [12], spatial pyramids [18], and bag-of-words [32]. Our approach is most related to [12], who also use spatially pooled CNN features for scene classification. They do so by pooling together multiple CNN descriptors (re)computed on various-sized patches within an image. Instead, we perform a single CNN encoding of the entire image, extracting multi-scale features "for free".

End-to-end training: Our multi-scale model differs from such past work in another notable aspect. Our entire model is still a feed-forward CNN that is no longer chain-structured, but a directed-acyclic graph (DAG). DAG-structured CNNs can still be discriminatively trained in an end-to-end fashion, allowing us to directly learn multi-scale representations. DAG structures are relatively straightforward to implement given the flexibility of many deep learning toolboxes [35, 15]. Our primary contribution is the demonstration that structures can capture multi-scale features, which in turn allow for transfer learning between coarse and fine-grained classification tasks.

DAG Neural Networks: DAG-structured neural nets were explored earlier in the context of recurrent neural nets [1, 13]. Recurrent neural nets use feedback to capture dynamic states, and so typically cannot be processed with feed-forward computations. Recent networks have explored the use of "skip" connections between layers [27, 33, 29], similar to our multi-scale connections. [27] show that such connections are useful for a single binary classification task, but we motivate multi-scale connections through multi-task learning: different visual classification tasks require features at different image scales. [33] use skip connections for training, but not at test-time (implying the final model not a DAG). Finally, our work aligns with approaches that predict local pixel labels using features extracted from multiple CNN layers [14, 22]. We show that such features also improve global image classification.

Overview: We motivate our multi-scale DAG-CNN model in Sec. 2, describe the full architecture in Sec. 3, and conclude with numerous benchmark results in Sec. 4. We evaluate multi-scale DAG-structured variants of existing CNN architectures (*e.g.*, Caffe [15], Deep19 [30]) on a variety of scene recognition benchmarks including SUN397 [39], MIT67 [26], Scene15 [9]. We observe a consistent improvement regardless of the underlying CNN architecture, producing state-of-the-art results on all 3 datasets.

2. Motivation

In this section, we motivate our multi-scale architecture with a series of empirical analysis. We carry out an analysis on existing CNN architectures, namely Caffe and Deep19.

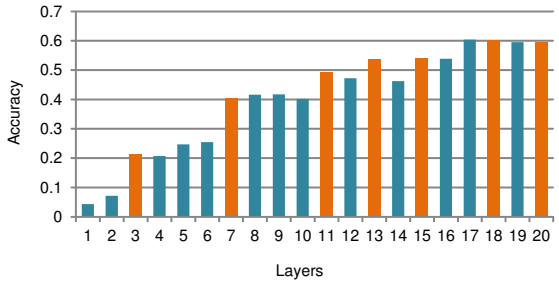


Figure 3. The classification accuracy on MIT67 [26] using activations from each layer. We use an orange color fill representing the output of a ReLU layer, where there are 7 in total for the Caffe model. We tend to see a performance jump at each successive ReLU layer, particularly earlier on in the model.

Caffe [15] is a broadly used CNN toolbox. It includes a pre-trained model "AlexNet" [17] model, learned with millions of images from the ImageNet dataset [5]. We denote the output of each convolution, rectification, normalization, pooling, and fully-connected inner product operation as a unique layer. Under this definition, the Caffe AlexNet model has 20 layers, while the state-of-the-art Deep19 model [30] has a total of 43 layers. To develop our motivation, we analyze the behavior of the "off-the-shelf" Caffe model on the heavily benchmarked MIT Indoor Scene (MIT67) dataset [26], using 10-fold cross-validation.

2.1. Single-scale models

Image retrieval: Recent work has explored sparse reconstruction techniques for visualizing and analyzing features [36]. Inspired by such techniques, we use image retrieval to begin our exploration. We attempt to "reconstruct" a query image by finding $M = 7$ closest images in terms of L2-distance, when computed with mean-pooled layer-specific activations. Results are shown for two query images and two Caffe layers in Fig. 2. The `florist` query image tends to produces better matches when using mid-level features that appear to capture *objects* and *parts*. On the other hand, the `church-inside` query image tends to produce better matches when using high-level features that appear to capture more global *scene* statistics.

Single-scale classification: Following past work [28], we train a linear SVM classifier using features extracted from a particular layer. We specifically train $K = 67$ 1-vs-all linear classifiers. We plot the performance of single-layer classifiers in Fig. 3. The detailed parameter options for both Caffe model are described later in Sec. 4. As past work has pointed out, we see a general increase in performance as we use higher-level (more invariant) features. We do see a slight improvement at each nonlinear activation (ReLU) layer. This makes sense as this layer introduces a nonlinear rectification operation $\max(0, x)$, while other layers (such an convolutional or sum-pooling) are linear operations that can be learned by a linear predictor.

Scale-varying classification: The above experiment required training $K \times N$ 1-vs-all classifiers, where K is the number of classes and N is the number of layers. We can treat each of the KN classifiers as binary predictors, and score each with the number of correct detections of its target class. We plot these scores as a matrix in Fig. 4. We tend to see groups of classes that operate best with features computed from particular high-level or mid-level layers. Most categories tend to do well with high-level features, but a significant fraction (over a third) do better with mid-level features.

Spatial pooling: In the next section, we will explore multi-scale features. One practical hurdle to including all features from all layers is the massive increase in dimensionality. Here, we explore strategies for reducing dimensionality through pooled features. We consider various pooling strategies (average versus max), pooling neighborhoods, and normalization post-processing (with and without L2 normalization). We saw good results with average pooling over all spatial locations, followed by L2 normalization (though we will re-examine these issues further in the next section). Specifically, assume a particular layer is of size $H \times W \times F$, where H is the height, W is the width, and F is the number of filter channels. We compute a $1 \times 1 \times F$ feature by averaging across spatial dimensions. We then normalize this feature to have unit norm.

2.2. Multi-scale models

Multi-scale classification: We now explore multi-scale predictors that process pooled features extracted from multiple layers. As before, we analyze "off-the-shelf" pre-trained models. We evaluate performance as we iteratively add more layers by feature concatenation. Fig. 3 suggests that the last ReLU layer is a good starting point due to its strong single-scale performance. Fig 5(a) plots performance as we add previous layers to the classifier feature set. Performance increases as we add intermediate layers, while lower layers prove less helpful (and may even hurt performance, likely do to over-fitting). Our observations suggest that high and mid-level features (*i.e.*, *parts* and *objects*) are more useful than low-features based on *edges* or *textures* in scene classification.

Multi-scale selection: The previous results show that adding all layers may actually hurt performance. We verified that this was an over-fitting phenomena; additional layers always improved training performance, but could decrease test performance due to over-fitting. This appears especially true for multi-scale analysis, where nearby layers may encoded redundant or correlated information (that is susceptible to over-fitting). Ideally, we would like to search for the "optimal" combination of ReLU layers that maximize performance on validation data. Since there exists

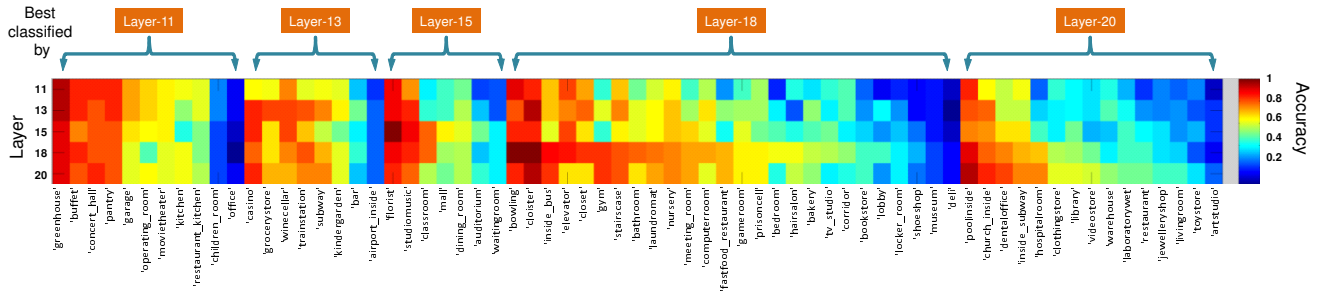
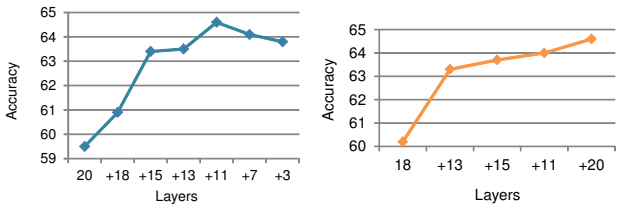


Figure 4. The "per-class" performance using features extracted from particular layers. We group classes with identical best-performing layers. The last layer (**20**) is optimal for only 15 classes, while the second-to-last layer (**18**) proves most discriminative for 26 classes. The third-most effective layer (**11**) captures significantly lower-level level features. These results validate our underlying hypothesis; different classes require different amounts of invariance. This suggests that a feature extractor shared across such classes will be more effective when multi-scale.



(a) Multi-scale (b) Forward Selection

Figure 5. (a) The performance of a multi-scale classifier as we add more layer-specific features. We start with the last ReLU layer, and iteratively add the previous ReLU layer. The "+" sign means the recent-most added layer. Adding additional layers help, but performance saturates and even slightly decreases when adding lower-layer features. This suggests it may be helpful to search for the "optimal" combination of layers. (b) The performance trend when using forward selection to incorporate the ReLU layers. Note that layers are *not* selected in high-to-low order. Specifically, it begin with the second-to-last ReLU layer, and skip one or more previous layers when adding the next layer. This suggests that layers encode some redundant or correlated information. Overall, we see a significant 6% improvement.

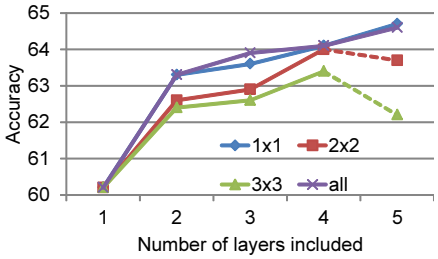


Figure 6. We evaluate spatial pooling strategies to see if retaining spatial information (by pooling over smaller windows) helps multi-scale performance. The answer is essentially 'no': global pooling (using a 1×1 window covering the entire layer) does as well as or better than local windows, either of a fixed-size or adaptive-size dynamically selected during the greedy search (**all**). We posit two reasons: (1) finer spatial cues may already be captured by multiscale features extracted from neighboring layers and (2) additional training data might be needed to realize the benefit of local pooling windows since they generate larger descriptors. Dotted line means the layer will not be selected by greedy procedure. For reference, we also experimented with max-pooling strategies, but saw consistently worse results.

an exponential number of combinations (2^N for N ReLU layers), we find an approximate solution with a greedy forward-selection strategy. We greedily select the next-best layer (among all remaining layers) to add, until we observe no further performance improvement. As seen in Fig. 5(b), the optimal results of this greedy approach rejects the low-level features. This is congruent with the previous results in Fig. 5(a).

Multi-scale pooling: We use our greedy scale-selection strategy to re-examine pooling strategies. Specifically, we divide the spatial features from a particular layer into $M \times M$ non-overlapping windows, where M varies from 1 to 3. We average-pool features within each window and concatenate the M^2 pooled features together into a final L2-normalized descriptor. When greedily searching over layers to add, we also greedily search over the 3 possible pooling windows for each layer. Interestingly, when evaluating pooling windows for *single-scale* classification, smaller pooling windows sometimes improved performance for certain layers. But in the multi-scale setting, there is essentially no improvement over global pooling (Fig. 6), perhaps because finer spatial cues may already be captured in multi-scale features extracted from neighboring layers. We posit that with more training data, locally-pooled features may perform better. In our subsequent analysis we consider only globally-pooled features, as they are simpler to implement and generate smaller descriptors.

Our analysis strongly suggest the importance (and ease) of incorporating multi-scale features for classification tasks. For our subsequent experiments, we use scales selected by the forward selection algorithm on MIT67 data (shown in Fig. 5(b)). Note that we use them for all our experimental benchmarks, demonstrating a degree of cross-dataset generalization in our approach.

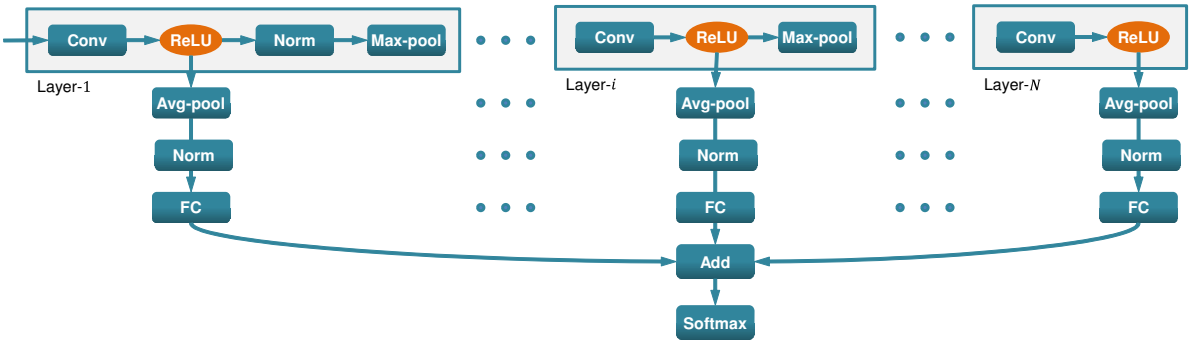


Figure 7. Our multi-scale DAG-CNN architecture is constructed by adding multi-scale output connections to an underlying *chain backbone* from the original CNN. Specifically, for each scale, we spatially (average) pool activations, normalize them to have unit-norm, compute an inner product with a fully-connected (FC) layer with K outputs, and add the scores across all layers to predictions for K output classes (that are finally soft-maxed together).

3. Approach

In this section, we show that the multi-scale model examined in Fig. 5(b) can be written as a DAG-structured, feed-forward CNN. Importantly, this allows for end-to-end gradient-based learning. To do so, we use standard calculus constructions – specifically the chain rule and partial derivatives – to generalize back-propagation to layers that have multiple "parents" or inputs. Though such DAG structures have been previously introduced by prior work, we have not seen derivations for the corresponding gradient computations. We include them here for completeness, pointing out several opportunities for speedups given our particular structure.

Model: The run-time behavior of our multi-scale predictor from the previous section is equivalent to feed-forward processing of the DAG-structured architecture in Fig. 5(b). Note that we have swapped out a margin-based hinge-loss (corresponding to a SVM) with a softmax function, as the latter is more amenable to training with current toolboxes. Specifically, typical CNNs are grouped into collections of four layers, *i.e.*, Conv., ReLU, contrast normalization (Norm), pooling layers (with the Norm and pooling layers being optional). The final layer is usually a K-way softmax function that predicts one of K outputs. We visualize these layers as a chain-structured "backbone" in Fig. 7. Our DAG-CNN simply links each ReLU layer to an average-pooling layer, followed by a L2 normalization layer, which feeds to a fully-connected (FC) layer that produces K outputs (represented formally as a $1 \times 1 \times K$ matrix). These outputs are element-wise added together across all layers, and the resulting K numbers are fed into the final softmax function. The weights of the FC layers are equivalent to the weights of the final multi-scale K-way predictor (which is a softmax predictor for a softmax loss output, and a SVM for a hinge-loss output). Note that all the required operations are standard modules except for the *Add*.

Training: Let $\mathbf{w}_1, ...\mathbf{w}_K$ be the CNN model parameters at $1, .., K$-th layer, training data be $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$, where $\mathbf{x}^{(i)}$ is the i-th input image and $\mathbf{y}^{(i)}$ is the indicator vector of the class of $\mathbf{x}^{(i)}$. Then we intend to solve the following optimization problem

$$\arg\min_{\mathbf{w}_1,...\mathbf{w}_K} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(f(\mathbf{x}^{(i)}; \mathbf{w}_1, ..., \mathbf{w}_K), \mathbf{y}^{(i)}) \qquad (1)$$

As is now commonplace, we make use of stochastic gradient descent to minimize the objective function. For a traditional *chain* model, the partial derivative of the output with respect to any one weight can be recursively computed by the chain rule, as described in the back-prop algorithm.

Multi-output layers (ReLU): Our DAG-model is structurally different at the ReLU layers (since they have multiple outputs) and the *Add* layer (since it has multiple inputs). We can still compute partial derivatives by recursively applying the chain rule, but care needs to be taken at these points. Let us consider the i-th ReLU layer in Fig. 8. Let α_i be its input, $\beta_i^{(j)}$ be the output for its j-th output branch (its j^{th} child in the DAG), and let z is the final output of the softmax layer. The gradient of z with respect to the input of the i-th ReLU layer can be computed as

$$\frac{\partial z}{\partial \alpha_i} = \sum_{j=1}^{C} \frac{\partial z}{\partial \beta_i^{(j)}} \frac{\partial \beta_i^{(j)}}{\partial \alpha_i} \quad \text{(in general)} \qquad (2)$$

where $C = 2$ for the example in Fig. 8. One can recover standard back-propagation equations from the above by setting $C = 1$: a single back-prop signal $\frac{\partial z}{\partial \beta_i^{(1)}}$ arrives at ReLU unit i, is multiplied by the local gradient $\frac{\partial \beta_i^{(1)}}{\partial \alpha_i^{(1)}}$, and is passed on down to the next layer below. In our DAG, *multiple* back-prop signals arrive $\frac{\partial z}{\partial \beta_i^{(j)}}$ from each branch j, each is multiplied by an branch-specific gradient $\frac{\partial \beta_i^{(j)}}{\partial \alpha_i}$, and their total sum is passed on down to the next layer.

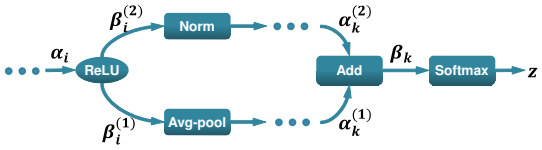


Figure 8. Visualization of the parameter setup at i-th ReLU.

Multi-input layers (Add): Let $\beta_k = g(\alpha_k^{(1)}, \cdots, \alpha_k^{(N)})$ represents the output of a layer with multiple inputs. We can compute the gradient along the layer by applying the chain rule as follows:

$$\frac{\partial z}{\partial \alpha_i} = \frac{\partial z}{\partial \beta_k}\frac{\partial \beta_k}{\partial \alpha_i} = \frac{\partial z}{\partial \beta_k}\sum_{j=1}^{C}\frac{\partial \beta_k}{\partial \alpha_k^{(j)}}\frac{\partial \alpha_k^{(j)}}{\partial \alpha_i} \quad \text{(in general)} \qquad (3)$$

One can similarly arrive at the standard back-propagation by setting $C = 1$.

Special case (ReLU): Our particular DAG architecture can further simplify the above equations. Firstly, it may be common for multiple-output layers to duplicate the same output for each child branch. This is true of our ReLU units; they pass the same values to the next layer in the chain and the current-layer pooling operation. This means the output-specific gradients are identical for those outputs $\forall j, \frac{\partial \beta_i^{(j)}}{\partial \alpha_i} = \frac{\partial \beta_i^{(1)}}{\partial \alpha_i}$, which simplifies (2) to

$$\frac{\partial z}{\partial \alpha_i} = \frac{\partial \beta_i^{(1)}}{\partial \alpha_i}\sum_{j=1}^{C}\frac{\partial z}{\partial \beta_i^{(j)}} \quad \text{(for duplicate outputs)} \qquad (4)$$

This allows us to add together multiple back-prop signals before scaling them by the local gradient, reducing the number of multiplications by C. We make use of this speed up to train our ReLU layers .

Special case (Add): Similarly, our multi-input `Add` layer reuses the same partial gradient for each input $\forall j, \frac{\partial \beta_k}{\partial \alpha_k^{(j)}} = \frac{\partial \beta_k}{\partial \alpha_k^{(1)}}$ which simplifies even further in our case to 1. The resulting back-prop equations that simplify (3) are given by

$$\frac{\partial z}{\partial \alpha_i} = \frac{\partial z}{\partial \beta_k}\frac{\partial \beta_k}{\partial \alpha_k^{(1)}}\sum_{j=1}^{C}\frac{\partial \alpha_k^{(j)}}{\partial \alpha_i} \quad \text{(for duplicate gradients)} \qquad (5)$$

implying that one can similarly save C multiplications. The above equations have an intuitive implementation; the standard chain-structured back-propagation signal is simply replicated along each of the parents of the `Add` layer.

Implementation: We use the excellent MatConNet codebase to implement our modifications [35]. We implemented a custom `Add` layer and a custom DAG data-structure to denote layer connectivity. Training and testing is essentially as fast as the chain model.

Vanishing gradients: We point out an interesting property of our multi-scale models that make them easier to train. Vanishing gradients [2] refers to the phenomena that gradient magnitudes decrease as they are propogated through layers, implying that lower-layers can be difficult to learn because they recieve too small a learning signal. In our DAG-CNNs, lower layers are *directly* connected to the output layer through multi-scale connections, ensuring they receive a strong gradient signal during learning. Fig. 9 experimentally verifies this claim.

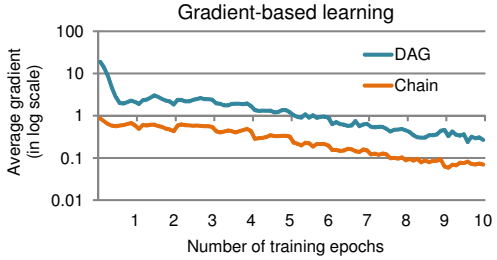


Figure 9. The average gradient from Layer-1 (Conv.) during training, plotted in log-scale. Gradients from the DAG are consistently $10\times$ larger, implying that they receive a stronger supervised signal from the target label during gradient-based learning.

4. Experimental Results

We explore DAG-structured variants of two popular deep models, Caffe [15] and Deep19 [30]. We refer to these models as Caffe-DAG and Deep19-DAG. We evaluate these models on three benchmark scene datasets: SUN397 [39], MIT67 [26], and Scene15 [9]. In absolute terms, we achieve the best performance ever reported on all three benchmarks, sometimes by a significant margin.

Feature dimensionality: Most existing methods that use CNNs as feature extractors work with the last layer (or the last fully connected layer), yielding a feature vector of size 4096. Forward feature selection on Caffe-DAG selects layers $(11, 13, 15, 18, 20)$, making the final multi-scale feature 9216-dimensional. Deep19-DAG selects layers$(26, 28, 31, 33, 30)$, for a final size of 6144. We perform feature selection by cross-validating on MIT67, and use the same multi-scale structure for all other datasets. Dataset-dependant feature selection may further improve performance. Our final multi-scale DAG features are *only 2X larger* than their single-scale counter part, making them practically easy to use and store.

Training: We follow the standard image pre-processing steps of fixing the input image size to 224×224 by scaling and cropping, and subtracting out the mean RGB value (computed on ImageNet). We initialize filters and biases to their pre-trained values (tuned on ImageNet) and initialize multi-scale fully-connected (FC) weights to small normally-distributed numbers. We perform 10 epochs of learning.

SUN397

Approach	Accuracy(%)
Deep19-DAG	**56.2**
Deep19 [30]	51.9
Caffe-DAG	46.6
Caffe [15]	43.5
Places [40] (Caffe)	54.3
MOP-CNN [12] (Caffe)	52.0
FV [34]	47.2
DeCaf [7]	40.9
Baseline-overfeat [38]	40.3
Baseline [39]	38.0

MIT67

Approach	Accuracy(%)
Deep19-DAG	**77.5**
Deep19 [30]	70.8
Caffe-DAG	64.6
Caffe [15]	59.5
MOP-CNN [12] (Caffe)	68.9
Places [40] (Caffe)	68.2
Mid-level [6]	64.0
FV+BoP [16]	63.2
Disc. Patch [31]	49.4
SPM [18]	34.4

Scene15

Approach	Accuracy(%)
Deep19-DAG	**92.9**
Deep19 [30]	90.8
Caffe-DAG	89.7
Caffe [15]	86.8
Place [40] (Caffe)	91.6
CENTRIST [37]	84.8
Hybrid [3]	83.7
Spatial pyramid [18]	81.4
Object bank [20]	80.9
Reconfigurable model [25]	78.6
Spatial Envelop [24]	74.1
Baseline [9]	65.2

Table 1. Classification results on SUN397, MIT67, and Scene15 datasets. We explicitly denote those methods that make use of the same Caffe model, either applied with a different (multi-scale) post-processing stage [12] or learned with a massively-large custom dataset [40]. Please see text for an additional discussion.

Baselines: We compare our DAG models to published results, including two additional baselines. We evaluate the best single-scale "off-the-shelf" model, using both Caffe and Deep19. We pass L2-normalized single-scale features to Liblinear [8] to train K-way one-vs-all classifiers with default settings. Finally, Sec. 4.1 concludes with a detailed diagnostic analysis comparing off-the-shelf and fine-tuned versions of chain and DAG structures.

SUN397: We tabulate results for all our benchmark datasets in Table 1, and discuss each in turn. SUN397 [39] is a large scene recognition dataset with 100K images spanning 397 categories, provided with standard train-test splits. Our DAG models outperform their single-scale counterparts. In particular, Deep19-DAG achieves the highest 56.2% accuracy. Our results are particularly impressive given that the next-best method of [40] (with a score of 54.3) makes use of a ImageNet-trained CNN and a custom-trained CNN using a new 7-million image dataset with 400 scene categories.

MIT67: MIT67 consists of 15K images spanning 67 indoor scene classes [26], provided with standard train/test splits. Indoor scenes are interesting for our analysis because some scenes are well characterized by high-level spatial geometry (*e.g.* `church` and `cloister`), while others are better described by mid-level objects (*e.g.* `wine celler` and `operating room`) in various spatial configurations. We show qualitative results in Fig. 10. Deep19-DAG produces a classification accuracy of 77.5%, reducing the best-previously reported error [12] by **23.9%**. Interestingly [12] also uses multi-scale CNN features, but do so by first extracting various-sized patches from an image, rescaling each to canonical size. Single-scale CNN features extracted from these patches are then vector-quantized into a large-vocabulary codebook, followed by a projection step to reduce dimensionality. Our multi-scale representation, while similar in spirit, is an end-to-end trainable model that is computed "for free" from a single (DAG) CNN.

Scene15: The Scene15 [9] includes both indoor scene (*e.g.*, store and kitchen) and outdoor scene (*e.g.*, mountain and street). It is a relatively small dataset by contemporary standards (2985 test images), but we include here for completeness. Performance is consistent with the results above. Our multi-scale DAG model, specifically Deep19-DAG, outperforms all prior work. For reference, the next-best method of [40] uses a new custom 7-million image scene dataset for training.

4.1. Diagnostics

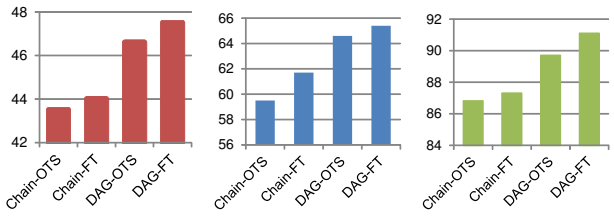
In this section, we analyze "off-the-shelf" (OTS) and "fine-tuned" (FT) versions of both single-scale chain and multi-scale DAG models. We focus on the Caffe model, as it is faster and easier for diagnostic analysis.

Chain: Chain-OTS uses single-scale features extracted from CNNs pre-trained on ImageNet. These are the baseline Caffe results presented in the previous subsections. Chain-FT trains a model on the target dataset, using the pre-trained model as an initialization. This can be done with standard software packages [35]. To ensure consistency of analysis, in both cases features are passed to a K-way multi-class SVM to learn the final predictor.

DAG: DAG-OTS is obtained by fixing all internal filters and biases to their pre-trained values, and only learning the multi-scale fully-connected (FC) weights. Because this final stage learning is a convex problem, this can be done by simply passing off-the-shelf multi-scale features to a convex linear classification package (e.g., SVM). We compare this model to a fine-tuned version that is trained end-to-end, making use of the modified backprop equation from Sec. 3.

Comparison: Fig. 11 compares off-the-shelf and fine-tune variants of chain and DAG models. We see two dominant trends. First, as perhaps expected, fine-tuned (FT) models consistently outperform their off-the-shelf (OTS) countparts. Even more striking is the large improvement from chain to DAG models, indicating the power of multi-scale feature encodings.

Figure 10. Deep19-DAG results on MIT67. The category label is shown on the left and the label for false-positives (in **red**) are also provided. We use the multi-scale analysis of Fig. 4 to compare categories that perform better with mid-level features (**top 2 rows**) versus high-level features (**bottom 2 rows**). Mid-level features appear to emphasize *objects* such as operating equipment for `operating room` scenes and circular grid for `wine celler`, while high-level features appear to focus on global *spatial statistics* for `inside bus` and `elevator` scenes.



(a) SUN397 (b) MIT67 (c) Scene15

Figure 11. Off-the-shelf vs. Fine-tuning models on both Chain and DAG model for Caffe backbone. Please see the text for a discussion.

DAG-OTS: Perhaps most impressive is the strong performance of DAG-OTS. From a theoretical perspective, this validates our underyling hypothesis that multi-scale features allow for better transfer between recognition tasks – in this case, ImageNet and scene classification. An interesting question is whether multi-scale features, when trained with gradient-based DAG-learning on ImageNet, will allow for even more transfer. We are currently exploring this. However even with current CNN architectures, our results suggest that *any system making use of off-the-shelf CNN features should explore multi-scale variants as a "cheap" baseline.* Compared to their single-scale counterpart, multi-scale features require no additional time to compute, are only a factor of 2 larger to store, and consistently provide a noticeable improvement.

Conclusion: We have introduced multi-scale CNNs for image classification. Such models encode scale-specific features that can be effectively shared across both coarse and fine-grained classification tasks. Importantly, such models can be viewed as DAG-structured feedforward predictors, allowing for end-to-end training. While fine-tuning helps performance, we empirically demonstrate that even "off-the-self" multi-scale features perform quite well. We present extensive analysis and demonstrate state-of-the-art classification performance on three standard scene benchmarks, sometimes improving upon prior art by a significant margin.

Acknowledgement. This work was supported by NSF Grant 0954083 and ONR-MURI Grant N00014-10-1-0933. This research is based upon work supported in part by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via IARPA R & D Contract No. 2014-14071600012. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

References

[1] P. Baldi and G. Pollastri. The principled design of large-scale recursive neural network architectures–dag-rnns and the protein structure prediction problem. *JMLR*, 2003. 2

[2] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 1994. 6

[3] A. Bosch, A. Zisserman, and X. Muoz. Scene classification using a hybrid generative/discriminative approach. *PAMI*, 2008. 7

[4] P. J. Burt and E. H. Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, 31(4):532–540, 1983. 1

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 1, 3

[6] C. Doersch, A. Gupta, and A. A. Efros. Mid-level visual element discovery as discriminative mode seeking. In *ECCV*. 2013. 7

[7] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *ICML*, 2014. 7

[8] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *JMLR*, 2008. 7

[9] L. Fei-Fei and P. Perona. A bayesian hierarchical model for learning natural scene categories. In *CVPR*, 2005. 2, 6, 7

[10] P. Felzenszwalb, D. McAllester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. In *CVPR*, 2008. 2

[11] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014. 1

[12] Y. Gong, L. Wang, R. Guo, and S. Lazebnik. Multi-scale orderless pooling of deep convolutional activation features. In *ECCV*, 2014. 1, 2, 7

[13] A. Graves and J. Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. In *NIPS*, 2009. 2

[14] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik. Hypercolumns for object segmentation and fine-grained localization. In *CVPR*, 2015. 2

[15] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014. 2, 3, 6, 7

[16] M. Juneja, A. Vedaldi, C. Jawahar, and A. Zisserman. Blocks that shout: Distinctive parts for scene classification. In *CVPR*, 2013. 7

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*. 2012. 1, 3

[18] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, 2006. 2, 7

[19] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, (11), 1998. 1

[20] L.-J. Li, H. Su, E. P. Xing, and F.-F. Li. Object bank: A high-level image representation for scene classification & semantic feature sparsification. In *NIPS*, 2010. 7

[21] T. Lindeberg. *Scale-space theory in computer vision*. Springer Science & Business Media, 1993. 1

[22] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015. 2

[23] S. Mallat. *A wavelet tour of signal processing*. Academic press, 1999. 2

[24] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *IJCV*, 2001. 7

[25] S. Parizi, J. Oberlin, and P. Felzenszwalb. Reconfigurable models for scene recognition. In *CVPR*, 2012. 7

[26] A. Quattoni and A. Torralba. Recognizing indoor scenes. In *CVPR*, 2009. 2, 3, 6, 7

[27] T. Raiko, H. Valpola, and Y. LeCun. Deep learning made easier by linear transformations in perceptrons. In *AISTATS*, volume 22, pages 924–932, 2012. 2

[28] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. CNN features off-the-shelf: an astounding baseline for recognition. In *CVPR*, 2014. 1, 3

[29] P. Sermanet, K. Kavukcuoglu, S. Chintala, and Y. LeCun. Pedestrian detection with unsupervised multi-stage feature learning. In *CVPR*. IEEE, 2013. 2

[30] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 1, 2, 3, 6, 7

[31] S. Singh, A. Gupta, and A. A. Efros. Unsupervised discovery of mid-level discriminative patches. In *ECCV*, 2012. 7

[32] J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *ICCV*, 2003. 2

[33] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015. 1, 2

[34] J. Snchez, F. Perronnin, T. Mensink, and J. Verbeek. Image classification with the fisher vector: Theory and practice. *IJCV*, 2013. 7

[35] A. Vedaldi and K. Lenc. Matconvnet user manual. 2, 6, 7

[36] C. Vondrick, A. Khosla, T. Malisiewicz, and A. Torralba. Hoggles: Visualizing object detection features. In *ICCV*, 2013. 3

[37] J. Wu and J. Rehg. Centrist: A visual descriptor for scene categorization. *PAMI*, 2011. 7

[38] J. Xiao, K. A. Ehinger, J. Hays, A. Torralba, and A. Oliva. Sun database: Exploring a large collection of scene categories. *IJCV*, 2001. 7

[39] J. Xiao, J. Hays, K. Ehinger, A. Oliva, and A. Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *CVPR*, 2010. 2, 6, 7

[40] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. Learning deep features for scene recognition using places database. In *NIPS*, 2014. 1, 7