Bilinear CNN Models for Fine-grained Visual Recognition

Tsung-Yu Lin Aruni RoyChowdhury Subhransu Maji
University of Massachusetts, Amherst
{tsungyulin,arunirc,smaji}@cs.umass.edu

Abstract

We propose bilinear models, a recognition architecture that consists of two feature extractors whose outputs are multiplied using outer product at each location of the image and pooled to obtain an image descriptor. This architecture can model local pairwise feature interactions in a translationally invariant manner which is particularly useful for fine-grained categorization. It also generalizes various orderless texture descriptors such as the Fisher vector, VLAD and O2P. We present experiments with bilinear models where the feature extractors are based on convolutional neural networks. The bilinear form simplifies gradient computation and allows end-to-end training of both networks using image labels only. Using networks initialized from the ImageNet dataset followed by domain specific fine-tuning we obtain 84.1% accuracy of the CUB-200-2011 dataset requiring only category labels at training time. We present experiments and visualizations that analyze the effects of fine-tuning and the choice two networks on the speed and accuracy of the models. Results show that the architecture compares favorably to the existing state of the art on a number of fine-grained datasets while being substantially simpler and easier to train. Moreover, our most accurate model is fairly efficient running at 8 frames/sec on a NVIDIA Tesla K40 GPU. The source code for the complete system will be made available at http://vis-www.cs.umass.edu/bcnn

1. Introduction

Fine-grained recognition tasks such as identifying the species of a bird, or the model of an aircraft, are quite challenging because the visual differences between the categories are small and can be easily overwhelmed by those caused by factors such as pose, viewpoint, or location of the object in the image. For example, the inter-category variation between "Ringed-beak gull" and a "California gull" due to the differences in the pattern on their beaks is significantly smaller than the inter-category variation on a popular fine-grained recognition dataset for birds [37].

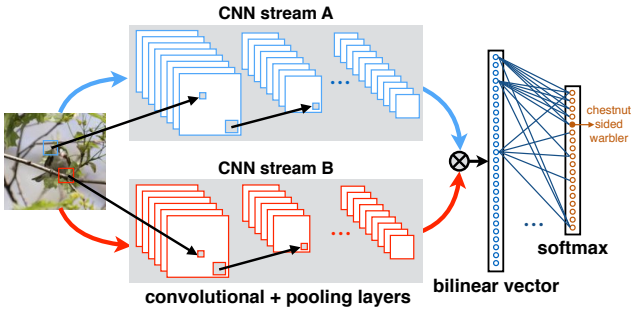

Figure 1. **A bilinear CNN model for image classification.** At test time an image is passed through two CNNs, A and B, and their outputs are multiplied using outer product at each location of the image and pooled to obtain the bilinear vector. This is passed through a classification layer to obtain predictions.

A common approach for robustness against these nuisance factors is to first localize various parts of the object and model the appearance conditioned on their detected locations. The parts are often defined manually and the part detectors are trained in a supervised manner. Recently variants of such models based on convolutional neural networks (CNNs) [2, 38] have been shown to significantly improve over earlier work that relied on hand-crafted features [1, 11, 39]. A drawback of these approaches is that annotating parts is significantly more challenging than collecting image labels. Morevoer, manually defined parts may not be optimal for the final recognition task.

Another approach is to use a robust image representation. Traditionally these included descriptors such as VLAD [20] or Fisher vector [28] with SIFT features [25]. By replacing SIFT by features extracted from convolutional layers of a deep network pre-trained on ImageNet [9], these models achieve state-of-the-art results on a number of recognition tasks [7]. These models capture local feature interactions in a translationally invariant manner which is particularly suitable for texture and fine-grained recognition tasks. Although these models are easily applicable as they don't rely on part annotations, their performance is below the best part-based models, especially when objects are small and appear in clutter. Moreover, the effect of end-to-end training of such architectures has not been fully studied.

Our main contribution is a recognition architecture that addresses several drawbacks of both part-based and texture models (Fig. 1 and Sect. 2). It consists of two feature extractors based on CNNs whose outputs are multiplied using the outer product at each location of the image and pooled across locations to obtain an image descriptor. The outer product captures pairwise correlations between the feature channels and can model part-feature interactions, *e.g.*, if one of the networks was a part detector and the other a local feature extractor. The bilinear model also generalizes several widely used orderless texture descriptors such as the Bag-of-Visual-Words [8], VLAD [20], Fisher vector [28], and second-order pooling (O2P) [3]. Moreover, the architecture can be easily trained end-to-end unlike these texture descriptions leading to significant improvements in performance. Although we don't explore this connection further, our architecture is related to the two stream hypothesis of visual processing in the human brain [15] where there are two main pathways, or "streams". The ventral stream (or, "what pathway") is involved with object identification and recognition. The dorsal stream (or, "where pathway") is involved with processing the object's spatial location relative to the viewer. Since our model is linear in the outputs of two CNNs we call our approach *bilinear CNNs*.

Experiments are presented on fine-grained datasets of birds, aircrafts, and cars (Sect. 3). We initialize various bilinear architectures using models trained on the ImageNet, in particular the "M-Net" of [5] and the "verydeep" network "D-Net" of [32]. Out of the box these networks do remarkably well, *e.g.*, features from the penultimate layer of these networks achieve 52.7% and 61.0% accuracy on the CUB-200-2011 dataset [37] respectively. Fine-tuning improves the performance further to 58.8% and 70.4%. In comparison a fine-tuned bilinear model consisting of a M-Net and a D-Net obtains 84.1% accuracy, outperforming a number of existing methods that additionally rely on object or part annotations (*e.g.*, 82.0% [21], or 75.7% [2]). We present experiments demonstrating the effect of fine-tuning on CNN based Fisher vector models [7], the computational and accuracy tradeoffs of various bilinear CNN architectures, and ways to break the symmetry in the bilinear models using low-dimensional projections. Finally, we present visualizations of the models in Sect. 4 and conclude in Sect. 5.

1.1. Related work

Bilinear models were proposed by Tanenbaum and Freeman [33] to model two-factor variations, such as "style" and "content", for images. While we also model two factor variations arising out of part location and appearance, our goal is prediction. Our work is also related to bilinear classifiers [29] that express the classifier as a product of two low-rank matrices. However, in our model the features are bilinear, while the classifier itself is linear. Our reduced dimensionality models (Sect. 3.3) can be interpreted as bilinear classifiers. "Two-stream" architectures have been used to analyze video where one networks models the temporal aspect, while the other models the spatial aspect [12, 31]. Ours is a two-steam architecture for image classification.

A number of recent techniques have proposed to use CNN features in an orderless pooling setting such as Fisher vector [7], or VLAD [14]. We compare against these methods. Two other contemporaneous works are of interest. The first is the "hypercolumns" of [17] that jointly considers the activations from all the convolutional layers of a CNN allowing finer grained resolution for localization tasks. However, they do not consider pairwise interactions between these features. The second is the "cross-layer pooling" method of [24] that considers pairwise interactions between features of adjacent layers of a single CNN. Our bilinear model can be seen as a generalization of this approach using separate CNNs simplifying gradient computation for domain specific fine-tuning.

2. Bilinear models for image classification

In this section we introduce a general formulation of a bilinear model for image classification and then describe a specific instantiation of the model using CNNs. We then show that various orderless pooling methods that are widely used in computer vision can be written as bilinear models.

A bilinear model $\mathcal{B}$ for image classification consists of a quadruple $\mathcal{B} = (f_A, f_B, \mathcal{P}, \mathcal{C})$. Here f_A and f_B are *feature functions*, $\mathcal{P}$ is a *pooling function* and $\mathcal{C}$ is a *classification function*. A feature function is a mapping $f : \mathcal{L} \times \mathcal{I} \rightarrow R^{c \times D}$ that takes an image $\mathcal{I}$ and a location $\mathcal{L}$ and outputs a feature of size $c \times D$. We refer to locations generally which can include position and scale. The feature outputs are combined at each location using the matrix outer product, *i.e.*, the *bilinear feature* combination of f_A and f_B at a location l is given by $\text{bilinear}(l, \mathcal{I}, f_A, f_B) = f_A(l, \mathcal{I})^T f_B(l, \mathcal{I})$.

Both f_A and f_B must have the feature dimension c to be compatible. The reason for $c > 1$ will become clear later when we show that various texture descriptors can be written as bilinear models. To obtain an image descriptor the pooling function $\mathcal{P}$ aggregates the bilinear feature across all locations in the image. One choice of pooling is to simply sum all the bilinear features, *i.e.*, $\phi(\mathcal{I}) = \sum_{l \in \mathcal{L}} \text{bilinear}(l, \mathcal{I}, f_A, f_B)$. An alternative is max-pooling. Both these ignore the location of the features and are hence *orderless*. If f_A and f_B extract features of size $C \times M$ and $C \times N$ respectively, then $\phi(\mathcal{I})$ is of size $M \times N$. The *bilinear vector* obtained by reshaping $\phi(\mathcal{I})$ to size $MN \times 1$ is a general purpose image descriptor that can be used with a classification function $\mathcal{C}$. Intuitively, the bilinear form allows the outputs of the feature exactors f_A and f_B to be conditioned on each other by considering all their pairwise interactions similar to a quadratic kernel expansion.

2.1. Bilinear CNN models

A natural candidate for the feature function f is a CNN consisting of a hierarchy of convolutional and pooling layers. In our experiments we use CNNs pre-trained on the ImageNet dataset [9] *truncated* at a convolutional layer including non-linearities as feature extractors. By pre-training we benefit from additional training data when domain specific data is scarce. This has been shown to be beneficial for a number of recognition tasks ranging from object detection, texture recognition, to fine-grained classification [6, 10, 13, 30]. Another advantage of using only the convolutional layers, is the resulting CNN can process images of an arbitrary size in a single forward-propagation step and produce outputs indexed by the location in the image and feature channel.

In all our experiments we use sum-pooling to aggregate the bilinear features across the image. The resulting bilinear vector $\mathbf{x} = \phi(\mathcal{I})$ is then passed through signed square-root step ($\mathbf{y} \leftarrow \text{sign}(\mathbf{x})\sqrt{|\mathbf{x}|}$), followed by ℓ_2 normalization ($\mathbf{z} \leftarrow \mathbf{y}/||\mathbf{y}||_2$) inspired by [28]. This improves performance in practice (see supplementary material for experiments evaluating the effect of these normalizations). For the classification function $\mathcal{C}$ we use logistic regression or linear SVM. This can be replaced with a multi-layer neural network if non-linearity is desirable.

End-to-end training Since the overall architecture is a directed acyclic graph the parameters can be trained by back-propagating the gradients of the classification loss (*e.g.*, conditional log-likelihood). The bilinear form simplifies the gradients at the pooling layer. If the outputs of the two networks are matrices A and B of size $L \times M$ and $L \times N$ respectively, then the pooled bilinear feature is $\mathbf{x} = A^T B$ of size $M \times N$. Let $d\ell/d\mathbf{x}$ be the gradient of the loss function ℓ wrto. $\mathbf{x}$, then by chain rule of gradients we have:

$$\frac{d\ell}{dA} = B\left(\frac{d\ell}{d\mathbf{x}}\right)^T, \quad \frac{d\ell}{dB} = A\left(\frac{d\ell}{d\mathbf{x}}\right). \tag{1}$$

The gradient of the classification and normalization layer is straightforward, and the gradient of the layers below the pooling layer can be computed using the chain rule. The scheme is illustrated in Fig 2. We fine-tune our model using stochastic gradient descent with mini-batches with weight decay and momentum as described in Sect 3.1.

2.2. Relation to orderless texture descriptors

In this section we show that various orderless texture descriptors can be written as bilinear models. These methods typically extract local features such as SIFT densely from an image and pass them through a non-linear encoder η. A popular encoder is a Gaussian mixture model (GMM) that assigns features to the k centers, $\mathbf{C} = [\mu_1, \mu_2, \ldots, \mu_k]$,

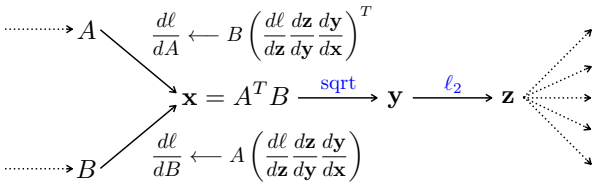

Figure 2. Computing gradients in the bilinear CNN model.

based on their GMM posterior. When these encoded descriptors are sum-pooled across the image we obtain the Bag-of-Visual-Words (BoVW) model [8]. Using the bilinear notation this can be written as $\mathcal{B} = (\eta(f_{\text{sift}}), 1, \mathcal{P}, \mathcal{C})$, *i.e.*, a bilinear model where the second feature extractor f_B simply returns 1 for all input.

The Vector of Locally Aggregated Descriptors (VLAD) descriptor [20] aggregates the first order statistics of the SIFT descriptors. Each descriptor $\mathbf{x}$ is encoded as $(\mathbf{x} - \mu_k) \otimes \eta(\mathbf{x})$, where $\otimes$ is the *kroneker product* and μ_k is the closest center to $\mathbf{x}$. In the VLAD model $\eta(\mathbf{x})$ is set to one for the closest center and zero elsewhere, also referred to as "hard assignment." These are aggregated across the image by sum pooling. Thus VLAD can be written as a bilinear model with $f_A = [\mathbf{x} - \mu_1; \mathbf{x} - \mu_2; \ldots; \mathbf{x} - \mu_k]$, *i.e.*, f_A has k rows each corresponding to each center, and $f_B = \text{diag}(\eta(\mathbf{x}))$, a matrix with $\eta(\mathbf{x})$ in the diagonal and zero elsewhere. Notice that the feature extractors for VLAD output a matrix with $k > 1$ rows.

The Fisher vector (FV) [28] computes both the first order $\alpha_i = \Sigma_i^{-\frac{1}{2}}(\mathbf{x} - \mu_i)$ and second order $\beta_i = \Sigma_i^{-1}(\mathbf{x} - \mu_i) \odot (\mathbf{x} - \mu_i) - 1$ statistics, which are aggregated weighted by $\eta(\mathbf{x})$. Here μ_i and Σ_i is the mean and covariance of the i^{th} GMM component respectively and $\odot$ denotes element-wise multiplication. This can be written as a bilinear model with $f_A = [\alpha_1\ \beta_1; \alpha_2\ \beta_2; \ldots; \alpha_k\ \beta_k]$ and $f_B = \text{diag}(\eta(\mathbf{x}))$.

In both VLAD and FV the encoding function η can be viewed as a part detector. Indeed it has been experimentally observed that the GMM centers tend to localize facial landmarks when trained on faces [27]. Thus, these models simultaneously localize parts and describe their appearance using joint statistics of the encoding $\eta(\mathbf{x})$ and feature $\mathbf{x}$ which might explain their effectiveness on fine-grained recognition tasks. Another successful method for semantic segmentation is the second-order pooling (O2P) method [3] that pools the covariance of SIFT features extracted locally followed by non-linearities. This is simply the bilinear model $\mathcal{B} = (f_{\text{sift}}, f_{\text{sift}}, \mathcal{P}, \mathcal{C})$.

In all these descriptors both f_A and f_B are based on the same underlying feature $\mathbf{x}$, *e.g.*, SIFT or CNN. One may want to use different features to detect parts and to describe their appearance. Furthermore, these methods typically do not learn the feature extractor functions and only the parameters of the encoder η and the classifier function $\mathcal{C}$ are learned on a new dataset. Even when CNN features are

pooled using FV method, training is usually not done end-to-end since it is cumbersome to compute the gradients of the network since f_A and f_B both depend on the $\mathbf{x}$. Our main insight is to decouple f_A and f_B which makes the gradient computation significantly easier (Eqn. 1), allowing us to fine-tune the feature extractors on specific domains. As our experiments show this significantly improves the accuracy. For Fisher vector CNN models we show that even when fine-tuning is done indirectly, *i.e.*, using a different pooling method, the overall performance improves.

3. Experiments

3.1. Methods

In addition to SIFT, we consider two CNNs for extracting features in the bilinear models – the M-Net of [5] and the verydeep network D-Net of [32] consisting of 16 convolutional and pooling layers. The D-Net is more accurate but is about 7× slower on a Tesla K40 GPU. In both cases we consider the outputs of the last convolutional layer with non-linearities as feature extractors, *i.e.*, layer 14 (conv_5+relu) for the M-net and layer 30 ($\text{conv}_{5\text{-}4}$+relu) for the D-Net. Remarkably, this represents less than 10% of the total number of parameters in the CNNs. Both these networks produce 1×512 dimensional features at each location. In addition to previous work, we evaluate the following methods keeping the training and evaluation setup identical for a detailed comparison.

I. CNN with fully-connected layers (FC-CNN) This is based on the features extracted from the last fully-connected layer before the softmax layer of the CNN. The input image is resized to 224×224 (the input size of the CNN) and mean-subtracted before propagating it though the CNN. For fine-tuning we replace the 1000-way classification layer trained on ImageNet dataset with a k-way softmax layer where k is the number of classes in the fine-grained dataset. The parameters of the softmax layer are initialized randomly and we continue training the network on the dataset for several epochs at a smaller learning rate while monitoring the validation error. Once the networks are trained, the layer before the softmax layer is used to extract features.

II. Fisher vector with CNN features (FV-CNN) This denotes the method of [7] that builds a descriptor using FV pooling of CNN filter bank responses with 64 GMM components. One modification is that we first resize the image to 448×448 pixels, *i.e.*, twice the resolution the CNNs were trained on and pool features from a *single-scale*. This leads to a slight reduction in performance, but we choose the single-scale setting because (i) multi-scale is likely to improve results for all methods, and (ii) this keeps the feature extraction in FV-CNN and B-CNN identical making comparisons easier. Fine-tuned FV-CNN results are reported using the fine-tuned FC-CNN models since direct fine-tuning is non-trivial. Surprisingly we found that this indirect training improves accuracy outperforming the non fine-tuned but multi-scale results (Sect 3.2.1) .

III. Fisher vector with SIFT (FV-SIFT) We implemented a FV baseline using dense SIFT features [28] extracted using VLFEAT [35]. Keeping the settings identical to FV-CNN, the input image is first resized to 448×448 before SIFT features with *binsize* of 8 pixels are computed densely across the image with a *stride* of 4 pixels. The features are PCA projected to 80 dimensions before learning a GMM with 256 components.

IV. Bilinear CNN model (B-CNN) We consider several bilinear CNN models – (i) initialized with two M-nets denoted by B-CNN [M,M], (ii) initialized with a D-Net and an M-Net denoted by B-CNN [D,M], and (iii) initialized with two D-nets denoted by B-CNN [D,D]. Identical to the setting in FV-CNN, the input images are first resized to 448×448 and features are extracted using the two networks before bilinear combination, sum-pooling, and normalization. The D-Net produces a slightly larger output 28×28 compared to 27×27 of the M-Net. We simply downsample the output of the D-Net by ignoring a row and column. The pooled bilinear feature is of size 512×512, which comparable to that of FV-CNN (512×128) and FV-SIFT (80 × 512). For fine-tuning we add a k-way softmax layer. We adopt the two step training procedure of [2] where we first train the last layer using logistic regression, a convex optimization problem, followed by fine-tuning the entire model using back-propagation for several epochs (about 45 – 100 depending on the dataset and model) at a relatively small learning rate ($\eta = 0.001$). Across the datasets we found the hyperparameters for fine-tuning were fairly consistent.

Classifier training In all our experiments once fine-tuning is done, training and validation sets are combined and one-vs-all linear SVMs on the extracted features are trained by setting the learning hyperparameter $C_{\text{svm}} = 1$. Since our features are ℓ_2 normalized the optimal of C_{svm} is likely to be independent of the dataset. The trained classifiers are calibrated by scaling the weight vector such that the median scores of positive and negative training examples are at $+1$ and -1 respectively. For each dataset we double the training data by flipping images and and at test time we average the predictions of the image and its flipped copy and assign the class with the highest score. Directly using the softmax predictions results in a slight drop in accuracy compared to linear SVMs. Performance is measured as the fraction of correct image predictions for all datasets.

3.2. Datasets and results

We report results on three fine-grained recognition datasets – birds [37], aircrafts [26], and cars [22]. Birds are smaller in the image compared to aircrafts stressing the role of part localization. Cars and birds also appear in more clutter compared to aircrafts. Fig. 3 shows some examples from these datasets. Approximate feature extraction speeds of our MatConvNet [36] based implementation and per-image accuracies for various methods are shown in Tab. 1.

3.2.1 Bird species classification

The CUB-200-2011 [37] dataset contains 11,788 images of 200 bird species. We evaluate our methods in two protocols – "birds" where the object bounding-box is *not* provided both at training and test time, and "birds + box" where the bounding-box is provided *both* at training and test time. For this dataset we crop a central square patch and resize it to 448×448 instead of resizing the image, which performed slightly better.

Several methods report results requiring varying degrees of supervision such as part annotation or bounding-boxes at training and test time. We refer readers to [2] that has a comprehensive discussion of results on this dataset. A more up-to-date set of results can be found in [21] who recently reported excellent performance using on this dataset leveraging more accurate CNN models with a method to train part detectors in a weakly supervised manner.

Comparison to baselines Without object bounding-boxes the fine-tuned FC-CNN [M] and FC-CNN [D] achieve accuracy of 58.8% and 70.4% respectively. Even without fine-tuning the FV models achieve better results than the corresponding fine-tuned FC models – FV-CNN [M] 61.1%, and FV-CNN [D] 71.3%. We evaluated FV models with the fine-tuned FC models and surprisingly found that this *improves* performance, *e.g.*, FV-CNN [D] improves to 74.7%. This shows that domain specific fine-tuning can be useful even when early convolutional layers of a CNN are used as features. Moreover, if FV-CNN fine-tuning was done to directly optimize its performance, results may further improve. However, as we discussed earlier such direct training is hard due to the difficultly in computing the gradients. We also note that the FV-CNN results with indirect fine-tuning outperforms the multi-scale results reported in [7] – 49.9% using M-Net and 66.7% using D-Net. The bilinear CNN models are substantially more accurate than the corresponding FC and FV models. Without fine-tuning B-CNN [M,M] achieves 72.0%, B-CNN [D,M] achieves 80.1%, while B-CNN [D,D] achieves 80.1% accuracy, even outperforming the fine-tuned FC and FV models. Fine-tuning improves performance of these models by about 4-6% to 78.1%, 84.1% and 84.0% respectively.

Figure 3. Examples from (left) birds dataset [37], (center) aircraft dataset [26], and (right) cars dataset [22] used in our experiments.

The trends when bounding-boxes are used at training and test times are similar. All the methods benefit from the added supervision. The performance of the FC and FV models improves significantly – roughly 10% for the FC and FV models with the M-Net and 6% for those with the D-Net. However, the most accurate B-CNN model benefits less than 1% suggesting a greater invariance to the location of parts in the image.

Comparison to previous work Two methods that perform well on this dataset when bounding-boxes are not available at test time are 73.9% of the "part-based R-CNN" [38] and 75.7% of the "pose-normalized CNN" [2]. Although the notion of parts differ, both these methods are based on a two step process of part detection followed by CNN based classifier. They also rely on part annotation during training. Our method outperforms these methods by a significant margin without relying on part or bounding-box annotations. Moreover, it is significantly simpler and faster – the bilinear feature computation using B-CNN [M,M] runs at 87 frames/sec, while B-CNN [D,M] runs at 8 frames/sec. Compared to the part detection step which requires thousands of network evaluations on region proposals [13] our method effectively requires only two evaluations and hence is significantly faster. We note that the accuracy of these methods can be improved by replacing the underlying AlexNet CNN [23] with the more accurate but significantly slower D-Net. Recently [21] reported 82.0% accuracy using a weakly supervised method to learn part detectors followed by the part-based analysis of [38] using a D-Net. However, this method relies on object bounding-boxes for training. Another recent approach called the "spatial transformer networks" reports 84.1% accuracy [19] using the Inception CNN architecture with batch normalization [18]. This approach also does not require object or part bounding-boxes at training time.

When bounding-boxes are used at test time all methods improve. The results of [38] improves to 76.4%. Another recently proposed method that reports strong results on this setting is the "cross-layer pooling" method of [24] that considers pairwise features extracted from two different layers of a CNN. Using AlexNet they report an accuracy of 73.5%. Our B-CNN model with two M-Nets method achieves 80.4% outperforming this by a significant margin.

method	birds		birds + box		aircrafts		cars		FPS
	w/o ft	w/ ft	w/o ft	w/ ft	w/o ft	w/ ft	w/o ft	w/ ft	
FV-SIFT	18.8	-	22.4	-	61.0	-	59.2	-	10†
FC-CNN [M]	52.7	58.8	58.0	65.7	44.4	57.3	37.3	58.6	124
FC-CNN [D]	61.0	70.4	65.3	76.4	45.0	74.1	36.5	79.8	43
FV-CNN [M]	61.1	64.1	67.2	69.6	64.3	70.1	70.8	77.2	23
FV-CNN [D]	71.3	74.7	74.4	77.5	70.4	77.6	75.2	85.7	8
B-CNN [M,M]	72.0	78.1	74.2	80.4	72.7	77.9	77.8	86.5	87
B-CNN [D,M]	80.1	**84.1**	81.3	**85.1**	78.4	83.9	83.9	**91.3**	8
B-CNN [D,D]	80.1	84.0	80.1	84.8	76.8	**84.1**	82.9	90.6	10
Previous work	**84.1** [19], 82.0 [21] 73.9 [38], 75.7 [2]		82.8 [21], 73.5 [24] 73.0 [7], 76.4 [38]		72.5 [4], 80.7 [16]		**92.6** [21], 82.7 [16] 78.0 [4]		†on a cpu

Table 1. **Classification results.** We report per-image accuracy on the CUB-200-2011 dataset [37] without (birds) and with bounding-boxes (birds + box), aircrafts dataset [26] and cars dataset [22]. FV-SIFT is the Fisher vector representation with SIFT features, FC-CNN uses features from the last fully connected layer of a CNN, and FV-CNN uses FV pooling of CNN filter banks [7]. B-CNN is the bilinear model consisting of two CNNs shown in brackets. For each model results are shown without and with domain specific fine-tuning. For FV-CNN fine-tuned results are reported using FC-CNN fine-tuned models. We report results using the M-Net [5] and D-Net [32] for various approaches. The feature extraction speeds (frames/sec) on a Tesla K40 GPU for various methods using our MatConvNet/VLFEAT based implementation are shown on the rightmost column. See Sect. 3 for details of the methods and a discussion of results.

Common mistakes Fig. 4 shows the top six pairs of classes that are confused by our fine-tuned B-CNN [D,M] model. The most confused pair of classes is "American crow" and "Common raven", which look remarkably similar. A quick search on the web reveals that the differences lie in the wing-spans, habitat, and voice, none of which are easy to measure from the image. Other commonly confused classes are also visually similar – various Shrikes, Terns, Flycatchers, Cormorants, etc. We note that the dataset has an estimated 4.4% label noise hence some of these errors may be incorrect [34].

Figure 4. Top six pairs of classes that are most confused with each other. In each row we show the images in the test set that were most confidently classified as the class in the other column.

3.2.2 Aircraft variant classification

The FGVC-aircraft dataset [26] consists of 10,000 images of 100 aircraft variants, and was introduced as a part of the FGComp 2013 challenge. The task involves discriminating variants such as the Boeing 737-300 from Boeing 737-400. The differences are subtle, *e.g.*, one may be able to distinguish them by counting the number of windows in the model. Unlike birds, airplanes tend to occupy a significantly larger portion of the image and appear in relatively clear background. Airplanes also have a smaller representation in the ImageNet dataset compared to birds.

Comparison to baselines The trends among the baselines are similar to those in birds with a few exceptions. The FV-SIFT baseline is remarkably good (61.0%) outperforming some of the fine-tuned FC-CNN baselines. Compared to the birds, the effect of fine-tuning FC-CNN [D] is significantly larger (45.0% $\rightarrow$ 74.1%) perhaps due to a larger domain shift from the ImageNet dataset. The fine-tuned FV-CNN models are also significantly better than the FC-CNN models in this dataset. Once again indirect fine-tuning of the FV-CNN models via fine-tuning FC-CNN helps by 5-7%. The best performance of 84.1% is achieved by the B-CNN [D,D] model. Fine-tuning leads to 7% improvement in its accuracy.

Comparison to previous work This dataset does not come with part annotations hence several top performing methods for the birds dataset are not applicable here. We also compare against the results for "track 2", *i.e.*, w/o bounding-boxes, at the FGComp 2013 challenge website [1].

[1] https://sites.google.com/site/fgcomp2013/results

The best performing method [16] is a heavily engineered FV-SIFT which achieves 80.7% accuracy. Notable differences between our baseline FV-SIFT and theirs are (i) larger dictionary (256 $\rightarrow$ 1024), (ii) Spatial pyramid pooling (1×1 $\rightarrow$ 1×1 + 3×1), (iii) multiple SIFT variants, and (iv) multi-scale SIFT. The next best method is the "symbiotic segmentation" approach of [4] that achieves 72.5% accuracy. However, this method requires bounding-box annotations at training time to learn a detector which is refined to a foreground mask. The B-CNN models outperform these methods by a significant margin. The results on this dataset show that orderless pooling methods are still of considerable importance – they can be easily applied to new datasets as they only need image labels for training.

3.2.3 Car model classification

The cars dataset [22] contains 16,185 images of 196 classes. Categories are typically at the level of Make, Model, Year, *e.g.*, "2012 Tesla Model S" or '2012 BMW M3 coupe." Compared to aircrafts, cars are smaller and appear in a more cluttered background. Thus object and part localization may play a more significant role here. This dataset was also part of the FGComp 2013 challenge.

Comparison to baselines FV-SIFT once again does well on this dataset achieving 59.2% accuracy. Fine-tuning significantly improves performance of the FC-CNN models, *e.g.*, 36.5% $\rightarrow$ 79.8% for FC-CNN [D], suggesting that the domain shift is larger here. The fine-tuned FV-CNN models do significantly better, especially with the D-Net which obtains 85.7% accuracy. Once again the bilinear CNN models outperform all the other baselines with the B-CNN [D, M] model achieving 91.3% accuracy. Fine-tuning improves results by 7-8% for the B-CNN models.

Comparison to previous work The best accuracy on this dataset is 92.6% obtained by the recently proposed method [21]. We also compare against the winning methods from the FGComp 2013 challenge. The SIFT ensemble [16] won this category (during the challenge) achieving a remarkable 82.7% accuracy. The symbiotic segmentation achieved 78.0% accuracy. The fine-tuned B-CNN [D,M] obtains 91.3% significantly outperforming the SIFT ensemble, and nearly matching [21] which requires bounding-boxes during training. The results when bounding-boxes are available at test time can be seen in "track 1" of the FGComp 2013 challenge and are also summarized in [16]. The SIFT ensemble improves significantly with the addition of bounding-boxes (82.7% $\rightarrow$ 87.9%) in the cars dataset compared to aircraft dataset where it improves marginally (80.7% $\rightarrow$ 81.5%). This shows that localization in the cars dataset is more important than in aircrafts. Our bilinear models have a clear advantage over FV models in this setting since it can learn to ignore the background clutter.

3.3. Low dimensional bilinear CNN models

The bilinear CNN models that are symmetrically initialized will remain symmetric after fine-tuning since the gradients for the two networks are identical. Although this is good for efficiency since the model can be implemented with just a single CNN evaluation, this may be suboptimal since the model doesn't explore the space of solutions that can arise from different CNNs. We experimented with several ways to break the symmetry between the two feature extractors. The first is "dropout" [23] where during training a random subset of outputs in each layer are set to zero which will cause gradients of the CNN to differ. However, we found that this led to 1% loss in performance on birds. We also experimented with a structured variant of dropout where we randomly zero out the rows and columns of the the pooled bilinear feature ($A^T B$). Unfortunately, this also performed 1% worse. We hypothesize that the model is stuck at a local minima as there isn't enough training data during fine-tuning. On larger datasets such schemes may be more important.

Our second idea is to project one of the CNN outputs to a lower dimension breaking the symmetry. This can be implemented by adding another layer of the CNN with a convolutional filter of size $1\times1\times$N$\times$D where N is the number of channels in the output of the CNN and D is the projected dimension. We initialize the parameters using PCA, projecting the 512 dimensional output of the M-Net to 64. Centering is absorbed into a bias term for each projection.

This projection also reduces the number of parameters in the model. For the B-CNN [M,M] model with k classes there are $512\times512\times k$ parameters in the classification layer. With the projection there are only $512\times64\times k$ parameters in the classification layer, plus 512×64 parameters in the projection layer. Thus, the resulting classification function $\mathcal{C}$ can also be viewed as a "bilinear classifier" [29] – a product of two low-rank matrices.

However, PCA projection alone worsens performance. Fig. 5 shows the average precision-recall curves across the 200 classes for various models. On birds the mean average precision (mAP) of the non fine-tuned model w/o PCA is 72.5% which drops to 72.0% w/ PCA. Since the projection is just another layer in the CNN, it can be jointly trained with the rest of the parameters in the bilinear model. This improves mAP to 80.1% even outperforming the original fine-tuned model that achieves 79.8%. Moreover the projected model is also slightly faster. Finally, we note that when PCA was applied to both the networks the results were significantly worse even with fine-tuning suggesting that sparse outputs are preferable when pooling.

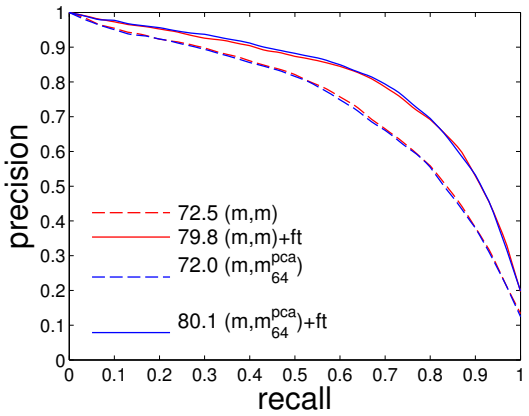

Figure 5. Low dimensional B-CNN (M,M) models.

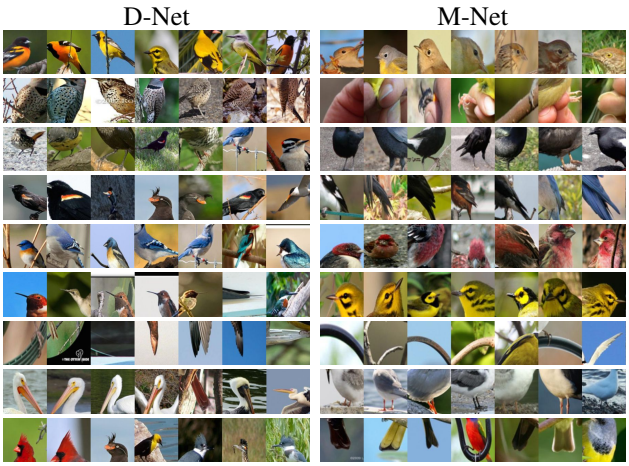

Figure 6. Patches with the highest activations for several filters of the fine-tuned B-CNN (D, M) model on CUB-200-2011 dataset.

4. Discussion

One of the motivations for the bilinear model was the modular separation of factors that affect the overall appearance. But do the networks specialize into roles of localization ("where") and appearance modeling ("what") when initialized asymmetrically and fine-tuned? Fig. 6 shows the top activations of several filters in the D-Net and M-Net of the fine-tuned B-CNN [D, M] model. These visualizations suggest that the roles of the two networks are not clearly separated. Both these networks tend to activate strongly on highly specific semantic parts. For example, the last row of D-Net detects "tufted heads", which can be seen as either part or a feature (visualizations on other datasets can be found in the supplementary material).

The above visualizations also suggests that the role of features and parts in fine-grained recognition tasks can be traded. For instance, consider the task of gender recognition. One approach is to first train a gender-neutral face detector and followed by a gender classifier. However, it may be better to train a gender-specific face detector instead. By jointly training f_A and f_B the bilinear model can effectively trade-off the representation power of the features based on the data. Thus, manually defined parts not only requires significant annotation effort but also is likely to be sub-optimal when enough training data is available.

Our bilinear CNN models had two feature extractors whose processing pathways separated early, but some of the early processing in the CNNs may be shared. Thus one can design a more efficient architecture where the feature extractors share the first few stages of their processing and then bifurcate to specialize in their own tasks. As long as the structure of the network is a directed acyclic graph standard back-propagation training applies. Our architecture is also modular. For example, one could append additional feature channels, either hand-crafted or CNNs, to the either f_A or f_B only update the trainable parameters during fine-tuning. Thus, one could train models with desired semantics, *e.g.*, color, describable textures [6], or parts, for predicting attributes or sentences. Finally, one could extend the bilinear model to a trilinear model to factor out another source of variation. This could be applied for action recognition over time where a third network could look at optical flow.

5. Conclusion

We presented bilinear CNN models and demonstrated their effectiveness on various fine-grained recognition datasets. Remarkably, the performance is comparable to methods that use the similar CNNs and additionally rely on part or bounding-box annotations for training. Our hypothesis is that our intuition of features that can be extracted from CNNs are poor and manually defined parts can be sub-optimal in a pipelined architecture. The proposed models can be fine-tuned end-to-end using image labels which results in significant improvements over other orderless texture descriptors based on CNNs such as the FV-CNN.

The model is also efficient requiring only two CNN evaluations on a 448×448 image. Our MatConvNet [36] based implementation of the asymmetric B-CNN [D,M] runs at 8 frames/sec on a Tesla K40 GPU for the feature extraction step, only a small constant factor slower than a single D-Net and significantly faster than methods that rely on object or part detections. The symmetric models are faster since they can be implemented with just a single CNN evaluation, *e.g.*, B-CNN [M,M] runs at 87 frames/sec, while the B-CNN [D,D] runs at 10 frames/sec. The source code for the complete system will be made available at http://vis-www.cs.umass.edu/bcnn

Acknowledgement This research was supported in part by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA) under contract number 2014-14071600010. The GPUs used in this research were generously donated by NVIDIA.

References

[1] L. Bourdev, S. Maji, and J. Malik. Describing people: A poselet-based approach to attribute classification. In *ICCV*, 2011. 1

[2] S. Branson, G. V. Horn, S. Belongie, and P. Perona. Bird species categorization using pose normalized deep convolutional nets. In *BMVC*, 2014. 1, 2, 4, 5, 6

[3] J. Carreira, R. Caseiro, J. Batista, and C. Sminchisescu. Semantic segmentation with second-order pooling. In *ECCV*. 2012. 2, 3

[4] Y. Chai, V. Lempitsky, and A. Zisserman. Symbiotic segmentation and part localization for fine-grained categorization. In *ICCV*, 2013. 6, 7

[5] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *BMVC*, 2014. 2, 4, 6

[6] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi. Describing textures in the wild. In *CVPR*, 2014. 3, 8

[7] M. Cimpoi, S. Maji, and A. Vedaldi. Deep filter banks for texture recognition and description. In *CVPR*, 2015. 1, 2, 4, 5, 6

[8] G. Csurka, C. R. Dance, L. Dan, J. Willamowski, and C. Bray. Visual categorization with bags of keypoints. In *ECCV Workshop on Stat. Learn. in Comp. Vision*, 2004. 2, 3

[9] 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

[10] 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*, 2013. 3

[11] R. Farrell, O. Oza, N. Zhang, V. I. Morariu, T. Darrell, and L. S. Davis. Birdlets: Subordinate categorization using volumetric primitives and pose-normalized appearance. In *ICCV*, 2011. 1

[12] K. Fragkiadaki, P. Arbeláez, P. Felsen, and J. Malik. Learning to segment moving objects in videos. In *CVPR*, 2015. 2

[13] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014. 3, 5

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

[15] M. A. Goodale and A. D. Milner. Separate visual pathways for perception and action. *Trends in neurosciences*, 15(1):20–25, 1992. 2

[16] P.-H. Gosselin, N. Murray, H. Jégou, and F. Perronnin. Revisiting the fisher vector for fine-grained classification. *Pattern Recognition Letters*, 49:92–98, 2014. 6, 7

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

[18] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 5

[19] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial transformer networks. *CoRR*, abs/1506.02025, 2015. 5, 6

[20] H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating local descriptors into a compact image representation. In *CVPR*, 2010. 1, 2, 3

[21] J. Krause, H. Jin, J. Yang, and L. Fei-Fei. Fine-grained recognition without part annotations. In *CVPR*, 2015. 2, 5, 6, 7

[22] J. Krause, M. Stark, J. Deng, and L. Fei-Fei. 3d object representations for fine-grained categorization. In *3D Representation and Recognition Workshop, at ICCV*, 2013. 5, 6, 7

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 5, 7

[24] L. Liu, C. Shen, and A. van den Hengel. The treasure beneath convolutional layers: Cross-convolutional-layer pooling for image classification. In *CVPR*, 2015. 2, 5, 6

[25] D. G. Lowe. Object recognition from local scale-invariant features. In *ICCV*, 1999. 1

[26] S. Maji, E. Rahtu, J. Kannala, M. Blaschko, and A. Vedaldi. Fine-grained visual classification of aircraft. *arXiv preprint arXiv:1306.5151*, 2013. 5, 6

[27] O. M. Parkhi, K. Simonyan, A. Vedaldi, and A. Zisserman. A compact and discriminative face track descriptor. In *CVPR*, 2014. 3

[28] F. Perronnin, J. Sánchez, and T. Mensink. Improving the Fisher kernel for large-scale image classification. In *ECCV*, 2010. 1, 2, 3, 4

[29] H. Pirsiavash, D. Ramanan, and C. C. Fowlkes. Bilinear classifiers for visual recognition. In *NIPS*. 2009. 2, 7

[30] A. S. Razavin, H. Azizpour, J. Sullivan, and S. Carlsson. Cnn features off-the-shelf: An astounding baseline for recognition. In *DeepVision workshop*, 2014. 3

[31] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. In *NIPS*, 2014. 2

[32] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 2, 4, 6

[33] J. B. Tenenbaum and W. T. Freeman. Separating style and content with bilinear models. *Neural computation*, 12(6):1247–1283, 2000. 2

[34] G. Van Horn, S. Branson, R. Farrell, S. Haber, J. Barry, P. Ipeirotis, P. Perona, and S. Belongie. Building a bird recognition app and large scale dataset with citizen scientists: The fine print in fine-grained dataset collection. In *CVPR*, 2015. 6

[35] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms. http://www.vlfeat.org/, 2008. 4

[36] A. Vedaldi and K. Lenc. MatConvNet – Convolutional Neural Networks for MATLAB. In *ACMMM*, 2015. 5, 8

[37] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, CalTech, 2011. 1, 2, 5, 6

[38] N. Zhang, J. Donahue, R. Girshickr, and T. Darrell. Part-based R-CNNs for fine-grained category detection. In *ECCV*, 2014. 1, 5, 6

[39] N. Zhang, R. Farrell, and T. Darrell. Pose pooling kernels for sub-category recognition. In *CVPR*, 2012. 1