HIGHER-ORDER REPRESENTATIONS FOR VISUAL RECOGNITION

A Dissertation Presented

by

TSUNG-YU LIN

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

February 2020

College of Information and Computer Sciences

HIGHER-ORDER REPRESENTATIONS FOR VISUAL RECOGNITION

A Dissertation Presented

by

TSUNG-YU LIN

Approved as to style and content by:

Subhransu Maji, Chair

Erik Learned-Miller, Member

Daniel Sheldon, Member

Rosie Cowell, Member

James Allan, Chair of the Faculty
College of Information and Computer Sciences

ACKNOWLEDGMENTS

Studying PhD is a 'highly-risky' investment in one's life. Without a good advisor and collaborators, one may spend a huge amount of time but is doomed to failure. I am fortunate to have been advised by Subhransu Maji. Without him, I would have gotten lost in the big field of Computer Vision research and never completed my degree. As my advisor and a talented researcher, he showed me his professionalism, breadth and depth of his knowledge, and his enthusiasm in doing research. I am also grateful to Erik Learned-Miller for extending me an admission to UMass in the first place and the discussions of research problems over the years. Without them, my career would have gone to a completely different path.

I would like to thank the rest of my committee members, Daniel Sheldon and Rosie Cowell, for their thoughtful feedback on this work. I also want to thank Shang-Hong Lai, Tyng-Luh Liu, and Hwann-Tzong Chen from my previous institutes in Taiwan, who led me into the field of Computer vision and supported me in pursuing the degree abroad. I am also thankful to my collaborators, mentors and colleagues, no matter our works were published or not, for your assistance and guidance during the school and internships, including Piotr Koniusz, Alex Berg, Tamara Berg, R. Manmatha, Deva Ramanan, Bharath Hariharan, Omkar Parkhi, Jong-Chyi Su, Chenyun Wu, and Mikayla Timm. I would particularly like to thank Aruni RoyChowdhury for being an excellent co-author for my first paper and going over many of my otherwise terrible writing, including this thesis and acknowledgments.

I had wonderful five and a half years in Amherst, thanks to two groups of people: labmates in the Computer Vision Lab and other friends who are mostly from Taiwan. I would like to thank our lab members: Li Yang Ku, Pia Bideau, Hang Su, Huaizu

Jiang, Ashish Singh, Matheus Gadelha, Gopal Sharma, Chetan Manjesh, and Zezhou Cheng, for exploring all the possibilities together for research problems and parties. I would like to extend a special thank to SouYoung Jin, who entered the lab at the same time as me. It was wonderful to have you on this journey and hitting all the milestones together to complete the degree requirements. I would also like to thank Meng-Chieh Chiu, Iwin Liu, Jian Niu, Hisn-Fei Tu, Cheng-Xian Li, Stella Huang, and Patt Huang for being awesome friends, hanging out together, and helping me release my stress from doing research.

I would like to thank especially my partner, Yi-Ting Chuang, who has been tremendously supportive of my studies and managing hard the long-distance relationship over so many years. Thank you for your unwavering support and understanding and tolerating all the holidays I spent working rather than spending time with you. Lastly, I would like to express my gratitude to my family members: my dad, mom, and brother for their endless care, love, support and sacrifices. I wish I could have spent more time with them over the past few years.

To young students who are considering studying PhD, it will be a risky decision. You will sacrifice a lot of time that you can spend on many other things otherwise and struggle most of the time on the journey and easily fail for multiple reasons. Nevertheless, your persistence will pay off, and the reward will be tremendous.

ABSTRACT

HIGHER-ORDER REPRESENTATIONS FOR VISUAL RECOGNITION

FEBRUARY 2020

TSUNG-YU LIN

B.Sc., NATIONAL TSING HUA UNIVERSITY

M.Sc., NATIONAL TSING HUA UNIVERSITY

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Subhransu Maji

In this thesis, we present a simple and effective architecture called *Bilinear Convolutional Neural Networks* (B-CNNs). These networks represent an image as a pooled outer product of features derived from two CNNs and capture localized feature interactions in a translationally invariant manner. B-CNNs generalize classical orderless texture-based image models such as bag-of-visual-words and Fisher vector representations. However, unlike prior work, they can be trained in an end-to-end manner. In the experiments, we demonstrate that these representations generalize well to novel domains by fine-tuning and achieve excellent results on fine-grained, texture and scene recognition tasks. The visualization of fine-tuned convolutional filters shows that the models are able to capture highly localized attributes. We present a texture synthesis framework that allows us to visualize the pre-images of fine-grained categories and the invariances that are captured by these models.

In order to enhance the discriminative power of the B-CNN representations, we investigate normalization techniques for rescaling the importance of individual features during aggregation. Spectral normalization scales the spectrum of the covariance matrix obtained after bilinear pooling and offers a significant improvement. However, the computation involves singular value decomposition, which is not computationally efficient on modern GPUs. We present an iteration-based approximation of matrix square-root along with its gradients to speed up the computation and study its effect on fine-tuning deep neural networks. Another approach is democratic aggregation, which aims to equalize the contributions of individual feature vector into the final pooled image descriptor. This achieves a comparable improvement, and can be approximated in a low-dimensional embedding unlike the spectral normalization. Therefore, this approach is friendly to aggregating higher-dimensional features. We demonstrate that the two approaches are closely related, and we discuss their trade-off between performance and efficiency.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

OVERVIEW

Image classification is a fundamental problem in Computer Vision research. Classical approaches involve the following two steps: building image descriptors that extract semantic information from images and representing them as high-dimensional vectors. The second step involves learning classifiers to separate the images according to their categories in the feature space. When the classifier is a hyperplane in high-dimensional space, ideally, the images belonging to different categories will lie on different sides of the hyperplane. Traditionally these two steps are designed separately, and most of the research works had focused on designing better image representations based on prior knowledge of the classification tasks. This has resulted in several widely-used image representations such as deformable part models [41, 10, 39, 133], spatial pyramid representations [79], and Fisher vector embeddings [100, 101].

Modern approaches consolidate the two steps into a single 'learnable' machine learning model where image descriptors and classifiers are derived from data via end-to-end training. Convolutional neural networks (CNNs) are the most successful models of this kind. Krizhevsky et al. [74] demonstrated that convolutional neural networks trained on large-scale datasets result in significantly more accurate recognition systems than the approaches based on a hand-designed heuristic for image classification. A further study [131] had shown that the convolutional features learned from data could extract highly-semantic information from images.

Given the success of feature learning, researchers have been interested in incorporating the designs of classical models into the feature learning framework. For

example, recent works [27, 51] studied deformable models for convolutional neural networks; He *et al.* [53] proposed spatial pyramid pooling to aggregate convolutional features; Several works [23, 84, 113, 102, 3] encodes convolutional features with Fisher vector and higher-order statistics. In this thesis, we focus on learning higher-order representations – modeling multiplicative feature interactions – based on the features extracted from convolutional neural networks. In particular, we propose bilinear CNNs (B-CNN) to aggregate second-order statistics of convolutional feature activations resulting in an end-to-end trainable architecture. We demonstrate that these models are effective in fine-grained, texture, and indoor scene recognition tasks.

Our method is motivated by classical texture-based image representations such as bag-of-visual-words and Fisher vector representations. In Chapter 2, we provide an overview of image representations for classification. In Chapter 3, we present bilinear CNNs (B-CNN) model to aggregate second-order statistics of convolutional feature activations in a translationally invariant manner for image classification. We study how orderless aggregation achieves the invariance of translation comparing against data jittering by random cropping. In addition, we discuss the approaches to reducing the feature dimension of outer-product features.

The bag-of-visual-words framework suffers from the burstiness of visual features that refer to the repeated visual elements arising from large homogeneous texture patterns. In Chapter 4, we study the approaches to enhance the robustness of bilinear pooling to the bursty features. We show that power normalizations [101, 19] are effective in handling bursty features and improve recognition accuracy, especially using matrix power functions. The approach is closely related to using matrix logarithm to map the Riemannian manifold of covariance matrices to a Euclidean space that preserves the geodesic distance between elements in the underlying manifold. We present a technique inspired by iterative numerical methods to compute matrix square-root to speed up the computation. In order to derive gradients for training

neural networks, we study techniques to compute the gradients w.r.t. matrix square-root and their numerical stability. Apart from matrix normalization, another line of research normalizes the image representations based on the frequency of local descriptors with feature-reweighting using democratic aggregation [97]. We explore this approach in the context of second-order representations and establish the connection between democratic aggregation and matrix power normalization. In addition to the analysis, we discuss the tradeoff between them regarding the computational cost and memory usages.

In Chapter 5 we present a unified framework to visualize novel images based on B-CNN representations. We formulate the image generation as an optimization problem to maximize the prediction score of a given category and simultaneously minimize the reconstruction errors under texture and image content representations. The framework is closely related to inverting deep image representations [94], neural style transfer [45], and texture synthesis [44]. In a particular case when we maximize the prediction scores of given categories, it allows us to analyze how categories from several fine-grained and texture recognition benchmarks are described by their textural content.

Since the breakthrough of CNNs on the ImageNet [108] classification performance drew the attention of the Computer Vision community, deep neural networks have been widely adopted by researchers. Since then, the architectures of deep networks have been improved in several ways, resulting in even higher performance. State-of-the-arts CNN networks [117, 54, 58, 57] increasing the depth of the networks have shown improvement on ImageNet challenges as well as fine-grained benchmarks. These networks are constructed by stacking multiple layers of building blocks that are designed to enrich the representations. Inception modules [117] and dense blocks [58] concatenate convolutional features from multiple streams. With deeper architectures and multi-stream features, these models could implicitly capture higher-order feature

correlation. We present the experiments and show that aggregating state-of-the-art convolutional features with bilinear pooling can further improve recognition accuracy on several fine-grained tasks.

Other approaches augment CNNs to localize salient features with attention mechanisms that allow focused reasoning on regions of an image [96, 4]. Spatial transformer networks [61] localize the objects and extract the features from the selected regions and improve the backbone Inception networks on fine-grained classification. B-CNNs can be viewed as an implicit spatial attention model since the outer product modulates one feature based on the other, similar to the multiplicative feature interactions in attention mechanisms. In addition to fine-grained recognition, this attention mechanism has also been shown to be effective in video action recognition and visual question answering. Recent works [21, 127] predict attention maps to localize actions in videos and aggregate the CNN activations from the selected spatio-temporal windows for action recognition. The higher-order representations are also effective in modeling the correlation between features from multiple modalities. Multi-modal bilinear pooling [42, 130] model the interaction between features from natural language and vision to locate visual features based on question representations. Recent work [86] applies bilinear pooling to model the interactions between visual and acoustic features for video classification. The two-stream-networks approach [40] models visual and motion features for video action recognition. Throughout the thesis, we will go over the techniques for modeling higher-order feature interactions and how to compute them efficiently.

CHAPTER 2

IMAGE REPRESENTATIONS FOR CLASSIFICATION

The main challenge for image classification is to design image representations that are discriminative to different categories while invariant to nuisance factors such as background, geometry, and illumination. Classical image classification algorithms tackle this by aggregating local feature descriptors, which are designed to be invariant to some of these factors. Using bag-of-visual-words representations and its variants for image classification was motivated by modeling texture and had been the state-of-the-art approaches before convolutional neural networks were widely adopted. In this chapter, we start with the introduction of local feature descriptors followed by the discussion on the approaches to modeling textures based on image filter responses. We continue the discussion with an overview of bag-of-visual-words representations and their variants. Then we describe convolutional neural networks and its training procedure to learn the image representations. We then discuss the techniques to aggregate the convolutional features and motivate bilinear pooling.

2.1 Local feature descriptors

Local feature descriptors are designed to characterize image patches and represent them by vectors. Linear filters bank has been widely used to represent local features. It consists of a set of D linear filters that are able to extract the responses of local patches to the selective local structures, such as edges and blobs shown in Figure 2.1. To densely extract feature responses across pixel locations, we slide each filter over an image and compute the response at each pixel by the sum of the element-wise

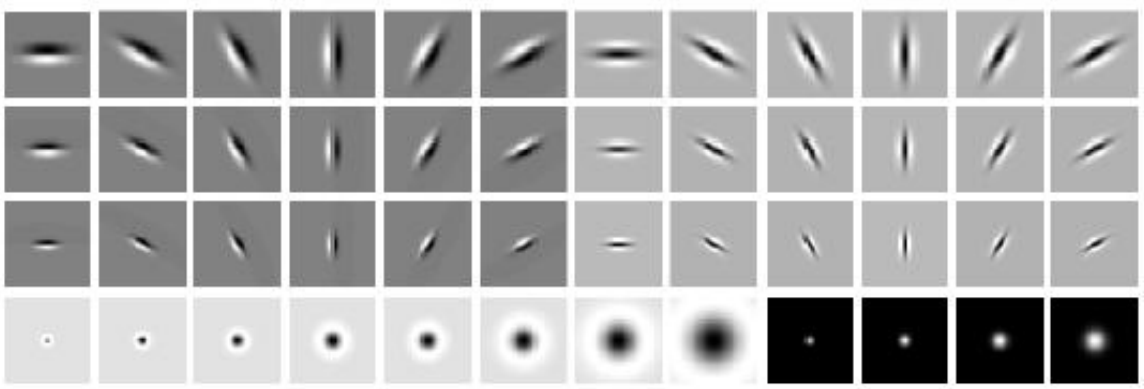


Figure 2.1. Leung-Malik filters bank. A set of linear filters proposed by Leung and Malik [80] consists of edge, blob and center-surrounding filters at multiple scales and orientations. This figure comes from: `http://www.robots.ox.ac.uk/~vgg/research/texclass/filters.html`.

products between the neighborhood of that pixel and the filter. The response to a particular filter is high when the image patch at that location is locally similar to the filter. For each image, this results in a set of N vectors, where N is the size of the image, and each of the N vectors is D-dimensional, where the element of each dimension corresponds to the filter response to one of the D filters. For image classification, the set of local feature vectors are aggregated to construct the image representations as described in Section 2.2.3.

Another widely used feature descriptor is SIFT [89]. Instead of calculating filter responses, SIFT descriptor computes the histogram of oriented gradients within a local neighborhood. The neighborhood is divided into 4×4 sub-regions and each sub-region is represented with an 8-bin histogram. This results in a128-dimensional vectors. For feature matching, the descriptor achieves the invariance to image scaling by calibrating the blob detection scores across the radii and achieves the invariance to image rotation by aligning histograms based on the dominating orientation. For image classification, SIFT descriptors are densely extracted across multiple scales at

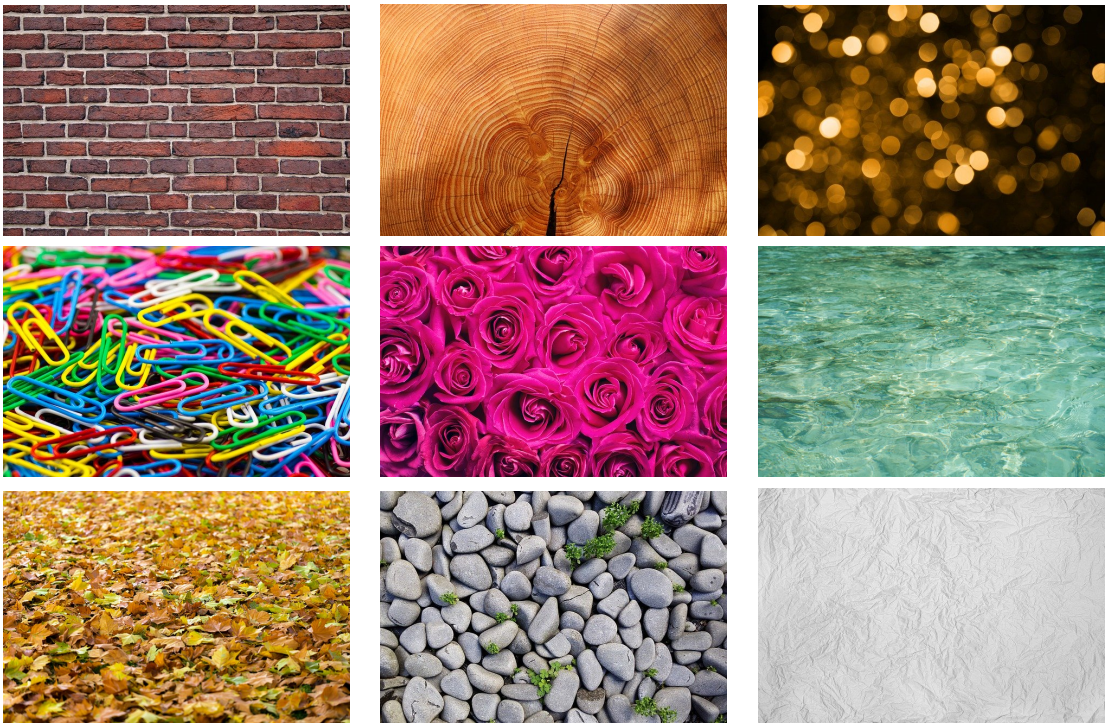


Figure 2.2. Examples of texture. Texture arises from different sources such as natural or man-made objects. The examples come from: `https://pixabay.com/images/search/texture/`.

each pixel locations and the densely extracted SIFT descriptors are then aggregated into image representations.

2.2 Texture representations

2.2.1 Representing texture with textons

Texture refers to the visually distinct repeated patterns arranged regularly or irregularly that characterize the appearance of an object's surface, as shown in Fig 2.2. It is a strong visual clue to infer the information of objects such as the materials, surface properties, and object categories. An early approach [80] to modeling texture is based on the histogram over *textons* - prototype local structured patches represented by feature vectors that hopefully capture the distinct patterns such as a leaf for foliage texture or a rectangular brick for brick texture. Texton was first named by Julesz [66]

to describe the putative units of pre-attentive human texture perception, such as line, terminators, crossings, and intersections. The term was later used by Leung and Malik [80] to describe the distinctive micro-structure of texture. The first step of this approach is to construct a set of textons by clustering the local descriptors obtained by filtering images with a linear filter bank (Fig. 2.1). Given a set of texture images, we convolve each image with image filters to extract the vectors of filter responses for each pixel. A universal set of K textons is obtained by clustering the filter responses into K clusters, where the vector representations of the cluster centers are used as textons. This leads to a quantization of the feature space into K disjoint regions based on the nearest cluster center. Given the quantization, a texture image is modeled as a distribution of filter responses using the histogram of K elements over textons, where each element is the number of local patches that are in the corresponding region normalized by the total number of patches in the image. The distance between two histograms can be used as the similarity between texture images and facilitate the applications in texture recognition and synthesis.

2.2.2 Region covariance matrices

Region covariance [118] is another widely-used texture representation. Given an image represented as an array of d-dimensional vectors obtained via some local descriptors, instead of modeling the distribution of feature responses, it computes the d x d covariance matrix where each element captures the co-occurrence between features. Unlike texton-based histogram representation, computing region covariance does not involve the clustering step, which requires a pre-defined number of clusters. Second-order pooling (O2P) computed the covariance matrices without mean-centering to model region of interest for semantic segmentation. As covariance matrices are not in Euclidean space, previous works [118, 119, 19] mapped the manifold to its tangent

space to preserve the geometry by Log-Euclidean mapping. The distance between two covariance matrices can be derived using the logarithm of joint eigenvalues.

2.2.3 Bag-of-visual-words representation and its variants

Image representations are obtained by aggregating the response to the local feature descriptors over locations. Similar to the construction of texton-based texture representations, a universal vocabulary of local descriptor responses are obtained by quantization, such as K-means clustering or Gaussian mixture models (GMM). An image is then represented by counting the appearance of visual words or computing the statistics on the deviation from the cluster centers. Bag-of-visual-words (BoVW) model was the earliest approach of such aggregation. Each feature $\mathbf{x}$ is then assigned to the closest cluster center (also called "hard assignment") and the image is represented as a histogram denoting frequencies of each visual word. Given a set of local descriptor responses $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\}$ and k cluster centers $\mathcal{M} = \{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \cdots, \boldsymbol{\mu}_k\}$, BoVW model results in an image representation $\mathbf{v} \in \mathbb{R}^k$ such that

$$v_j = \frac{1}{N} \sum_{\mathbf{x}_i \text{ such that } NN(\mathbf{x})=\boldsymbol{\mu}_j} 1$$

where $NN(\mathbf{x})$ maps feature vector $\mathbf{x}_i$ to its nearest visual word $\boldsymbol{\mu}_j$. The aggregation by summation discards the order of features $\mathcal{X}$ resulting in an *orderless* representation, which means the representation is invariant to the permutation of $\mathcal{X}$.

Although BoVW is simple and effective, the representation only considers the count statistics while ignores how the features distribute with respect to the visual words. Vector of locally aggregated descriptors (VLAD) [62] was proposed to address this issue by capturing the deviation of local descriptors from cluster centers in the aggregate. For each visual word $\boldsymbol{\mu}_j$, VLAD representation accumulates the residual

$\mathbf{x}_i - \boldsymbol{\mu}_j$ for all $\mathbf{x}_i$ such that $NN(\mathbf{x}_i) = \boldsymbol{\mu}_j$. Hence the component $\mathbf{v}_j$ associated with the visual word $\boldsymbol{\mu}_j$ is written as:

$$\mathbf{v_j} = \sum_{\mathbf{x}_i \text{ such that } NN(\mathbf{x})=\boldsymbol{\mu}_j} \mathbf{x}_i - \boldsymbol{\mu}_j$$

Finally, all vectors $\mathbf{v}_j$ are concatenated into a Dk-dimensional vector $\mathbf{v}$ followed by a L2-normalization ($\mathbf{v} \leftarrow \mathbf{v}/||\mathbf{v}||_2$).

Fisher vector (FV) representation [100] further enriches the representation by encoding the variance of the deviation. It is also related to the approximation of the Fisher kernel to compute the similarity for image classification. Given a GMM distribution with k components and parameters $\{w_j, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j, j = 1, \cdots, k\}$, FV representation computes the derivative of log-likelihood given $\mathcal{X}$ with respect to $\boldsymbol{\mu}_j$ and $\boldsymbol{\Sigma}_j$ resulting in the statistics:

$$\boldsymbol{\alpha_j} = \frac{1}{N\sqrt{w_j}} \sum_{i=1}^{N} \theta_j(\mathbf{x}_i) \boldsymbol{\Sigma}_j^{-\frac{1}{2}} (\mathbf{x}_i - \boldsymbol{\mu}_j)$$

$$\boldsymbol{\beta_j} = \frac{1}{N\sqrt{2w_j}} \sum_{i=1}^{N} \theta_j(\mathbf{x}_i) \boldsymbol{\Sigma}_j^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_j) \odot (\mathbf{x}_i - \boldsymbol{\mu}_j) - 1$$

where $\boldsymbol{\mu}_j$, $\boldsymbol{\Sigma}_j$ and w_j are the mean vector, covariance matrix and the mixture weight respectively of the *j-th* Gaussian component. The covariance matrices are usually assumed to be diagonal and the operation $\odot$ denotes element-wise multiplication. The statistics are computed by a weighted sum based on the GMM posteriors $\theta_j(\mathbf{x}_i)$. The final FV encoding is a concatenation of $\boldsymbol{\alpha}_j$ and $\boldsymbol{\beta}_j$ for all j and results in a $2Dk$-dimensional representation. The improved FV [101] further normalizes the representation by power normalization ($\mathbf{v} \leftarrow \text{sign}(\mathbf{v})|\mathbf{v}|^\gamma$) and L2 normalization to improve the performance.

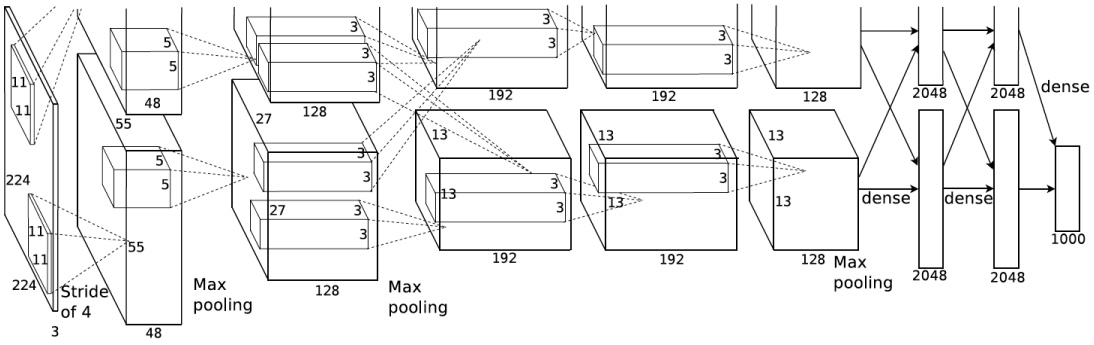


Figure 2.3. Architecture of AlexNet. At each layer, each block represents a three-dimensional tensor, which is convolved with the filters shown as a small block within it. The output from the last convolutional layer is fed into a two-layer network generating 4096-dimensional features to predict the class labels using a k-way softmax classifier. The network is separated into two paths each of which is processed by one GPU to fit the overall network given GPU memory constraint. This figure comes from [74].

2.3 Convolutional neural networks

The bag-of-visual-words and its variants were popular approaches for various image recognition tasks. However, the invariance of the representations to the local structure change relies on the design of local feature descriptors. These local descriptors are *handcrafted* and might not be optimal for the given tasks. Instead of engineering the local descriptors, feature learning approaches aim to learn the representations from the raw data to optimize the performance of the end tasks. Convolutional neural networks (CNNs) have demonstrated that end-to-end learning encourages low-level neurons to behave like feature detectors such as edge filters, while high-level neurons capture highly-semantic information.

A convolutional neural network is a sequence of layers, each of which is connected to the previous and the next layers. The output of the previous layer is input to the current layer and the results of the current layer are propagated to the next layer. As an example, the AlexNet architecture is shown in Figure. 2.3, which consists of

five convolutional layers and two fully-connected layers. Each convolutional layer is followed by a non-linearity and a pooling function. At the l-th layer, we consider a 3-dimensional input tensor $\mathbf{x}^{l-1} \in \mathbb{R}^{c^{l-1} \times h^{l-1} \times w^{l-1}}$ conceptualized as an image with c^{l-1} channels and $h^{l-1} \times w^{l-1}$ as spatial resolution. We define $\mathbf{x}^0$ as the input image. A convolution slides a filter over every location of input tensor and outputs the response for each location. At each location, the operation computes the sum of elementwise multiplication between the filter weights and the input values at the receptive field. This is done simultaneously with multiple filters and the responses are concatenated to produce a multiple-channel output. A ReLU nonlinearity follows the convolution and transforms the value $z' = \max(0, z)$. A pooling layer reduces the spatial dimension by aggregating the ReLU outputs over small neighborhoods.

The output of the last convolutional layer after ReLU activation is fed into a two-layer fully-connected network. Each neuron in a fully-connected layer is connected to all the neurons in the previous layer, *i.e.* the receptive field is the full image, and activated by the ReLU function. The output of the last fully-connected layer is fed into a k-way classification layer. The overall network is parameterized by the weights of convolutional filters and the fully-connect layers. These parameters are solved via optimization methods such as stochastic gradient descent to minimize the cross-entropy loss for multi-class image classification.

By training CNNs on a large scale image dataset, AlexNet [74] outperformed the previous state-of-the-art method using Fisher vector with handcrafted features by more than 8% accuracy on ImageNet challenge [108]. Recently, several variants of building block for convolutional neural networks have been proposed and improved the model performance. VGG networks [112] factored the large convolutional kernels into a sequence of 3×3 kernels to reduce the number of parameters. Inception network [117] deployed filters in multiple scales within an inception module and combined the information from the context of multiple sizes. Residual network [54] inserted skip

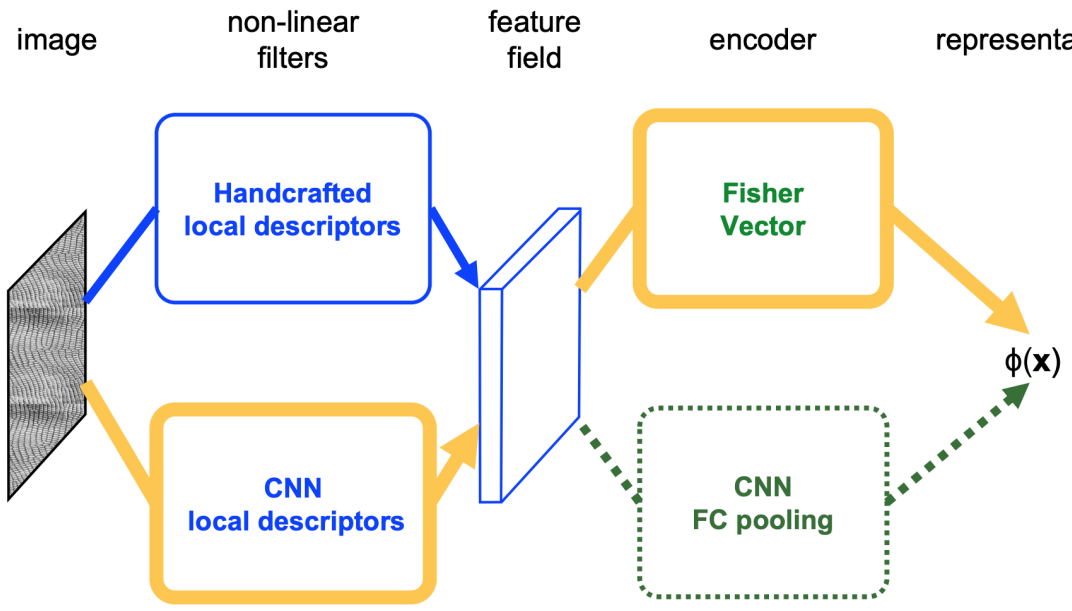


Figure 2.4. The procedure of computing image representations. Classical texture-based representations encode handcrafted local descriptors using a higher-order encoding such as Fisher vector. The representations obtained from AlexNet can be considered as convolutional features aggregated by a two-layer fully-connected network. Deep filter banks [23] attempted to aggregate convolutional features with Fisher vector. The figure comes from: `https://www.robots.ox.ac.uk/~vgg/publications/2015/Cimpoi15/presentation.pdf`.

connections to allow gradient flow to the bottom layers alleviating the problem of vanishing gradients. DenseNet [58] connected all layers within a dense block to further improve the recognition performance. SENet [57] proposed squeeze and excitation modules to reweight feature channels with global features. Designing better building blocks for CNNs models is an area of research.

2.4 Aggregating convolutional features

Conceptually we can consider the fully-connected network as an aggregation function, which accumulates the convolutional features. Theoretically, a two-layer feed-forward network given sufficient hidden units is a universal approximator of continuous functions. However, the number of hidden units required for a given task

is unclear. Moreover, these layers usually consist of millions of parameters, which make the learning prone to overfitting. The recent models proposed to aggregate convolutional features in a translationally invariant manner resulting in deep texture representation, as illustrated in Fig. 2.4. State-of-the-art CNN models for image classification [117, 54, 58, 57] replaced the two-layer fully-connected layer with global average pooling to aggregate the convolutional features. Deep filter banks [23] constructed Fisher Vector representation on top of CNN activations. NetVLAD [3] extended the VLAD representation and used soft cluster assignments to derive an end-to-end trainable VLAD representation.

In this thesis, We present *Bilinear CNNs (B-CNNs)* that generalize existing texture representations. Our key insight is that several widely-used texture representations can be written as a pooled outer product of two suitably designed features. The output is a fixed high-dimensional representation which can be combined with a fully-connected layer to predict class labels. The simplest bilinear layer is one where two *identical* features are combined with an outer product. This is closely related to the *Second-Order Pooling* approach of Carreira *et al.* [19] popularized for semantic image segmentation. When these features are extracted from convolutional neural networks, the resulting architecture can be trained in an end-to-end manner on large datasets, or domain-specific fine-tuning for transfer learning.

CHAPTER 3

BILINEAR CONVOLUTIONAL NEURAL NETWORKS

In this chapter, we introduce bilinear CNNs that represent an image as a pooled outer product of features derived from two CNNs and capture localized feature interactions in a translationally invariant manner. B-CNNs generalize various widely-used bag-of-visual-words representations, but unlike prior work, they can be trained in an end-to-end manner. We demonstrate the effectiveness of the models and report the recognition accuracy on various fine-grained, texture, and scene recognition tasks. The feature aggregation is orderless, and thus the representations are invariant to image translation. We analyze the effect of orderless feature aggregation in achieving translationally-invariant representations and compare it against the commonly-used data augmentation approach with spatial jittering. As the feature dimension of the outer product grows quadratically, representing the images with full bilinear features is not memory-efficient. We study the methods of approximating outer-product features with PCA or Tensor sketches for dimensionality reduction.

3.1 The B-CNN architecture

B-CNN 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* based on CNNs, $\mathcal{P}$ is a *pooling function*, and $\mathcal{C}$ is a *classification function*. A feature function is a mapping $f : \mathcal{L} \times \mathcal{I} \rightarrow \mathbb{R}^{K \times D}$, that takes an image $I \in \mathcal{I}$ and a location $l \in \mathcal{L}$ and outputs a feature of size $K \times D$. 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 combination of f_A and f_B at a location l is given by

$$\mathsf{bilinear}(l, I, f_A, f_B) = f_A(l, I)^T f_B(l, I). \tag{3.1}$$

Both f_A and f_B must have the same feature dimension K to be compatible. The value of K depends on the particular model. For example, $K = 1$ for BoVW model and equals the number of clusters in a FV model. The pooling function $\mathcal{P}$ aggregates the bilinear combination of features across all locations in the image to obtain a global image representation $\Phi(I)$. We use sum pooling in all our experiments, *i.e.*,

$$\Phi(I) = \sum_{l \in \mathcal{L}} \mathsf{bilinear}(l, I, f_A, f_B) = \sum_{l \in \mathcal{L}} f_A(l, I)^T f_B(l, I). \tag{3.2}$$

Since the location of features is ignored during pooling, the bilinear feature $\Phi(I)$ is an *orderless* representation. If f_A and f_B extract features of size $K \times M$ and $K \times N$ respectively, then $\Phi(I)$ is of size $M \times N$. The bilinear feature is a general-purpose image representation that can be used with a classifier $\mathcal{C}$ (Figure 3.1). Intuitively, the outer product conditions the outputs of features f_A and f_B on each other by considering their pairwise interactions, similar to the feature expansion in a quadratic kernel.

3.1.1 Feature functions

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 *truncated* at an intermediate layer as feature functions. By pre-training we benefit when domain-specific data is limited. This has been shown to be effective for a number of tasks ranging from object detection, texture recognition, to fine-grained classification [33, 50, 106, 22]. Another advantage of using CNNs is that

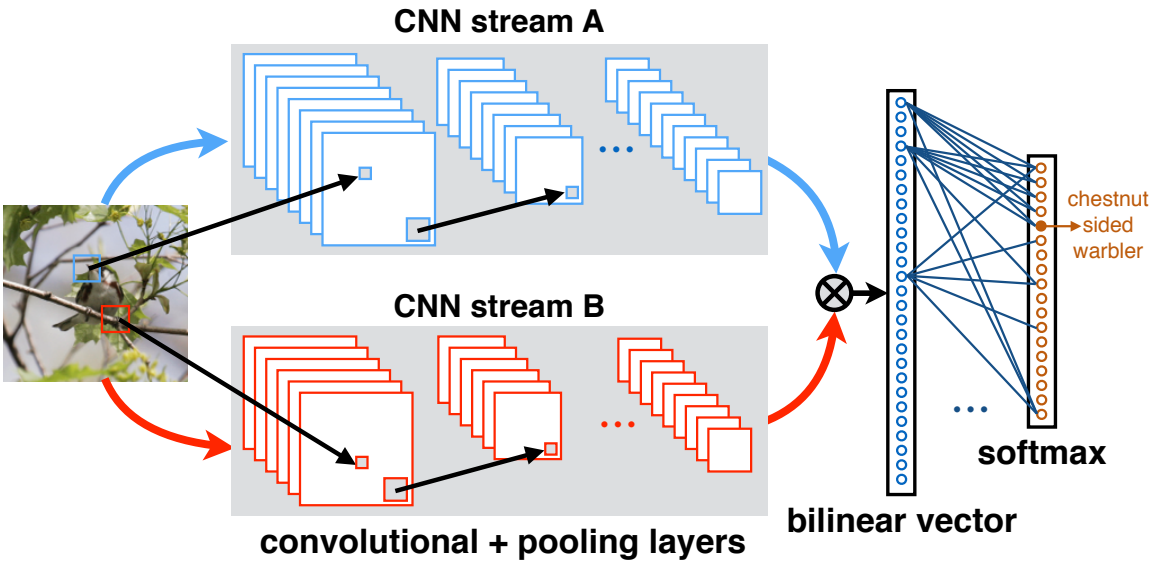


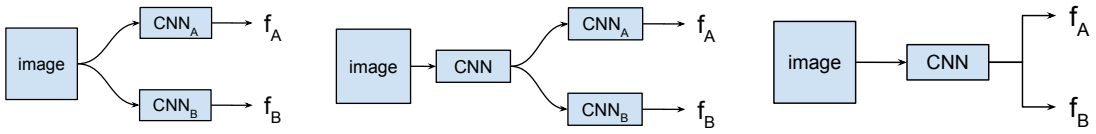
Figure 3.1. Image classification using a B-CNN. An image is passed through CNNs A and B, and their outputs at each location are combined using the matrix outer product and average pooled to obtain the bilinear feature representation. This is passed through a linear and softmax layer to obtain class predictions.

the resulting network can process images of an arbitrary size and produce outputs indexed by image location and feature channel.

The feature functions f_A and f_B can be *independent*, *partially shared*, or *fully shared* as shown in Figure 3.2. The independent model corresponds to the most-general model where f_A and f_B are two CNNs with different parameters. Second-order pooling which computes the outer product between the features with itself is considered as a fully shared B-CNN models. The feature functions used to approximate classical texture representations we present in Section 3.2, as well as the low-dimensional B-CNNs we present in Section 3.4 are considered as partially shared B-CNN models.

3.1.2 Normalization and classification

We perform additional normalization steps where the bilinear feature $\mathbf{x} = \Phi(I)$ is passed through a signed square-root ($\mathbf{y} \leftarrow \text{sign}(\mathbf{x})\sqrt{|\mathbf{x}|}$), followed by ℓ_2 normalization



(a) no sharing (b) partially shared (c) fully shared

Figure 3.2. Variants of B-CNNs architecture. Feature functions in B-CNNs can (a) share no computations (*e.g.*, B-CNN model based on VGG-M and VGG-D), (b) share computations partially (*e.g.*, NetVLAD, B-CNN PCA model described in Section 3.4), and (c) share all computations (*e.g.*, B-CNN model based on VGG-M).

($\mathbf{z} \leftarrow \mathbf{y}/||\mathbf{y}||_2$) inspired by [101]. This improves performance in practice (see our earlier work [83] for an evaluation of the effect of normalization). For classification we use logistic regression or linear SVM [109]. Although this can be replaced with an arbitrary multi-layer network, we found that linear models are effective on top of bilinear features.

3.1.3 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.*, cross-entropy). The *bilinear form* simplifies the gradient computations. If the outputs of the two networks are matrices A and B of size $L \times M$ and $L \times N$ respectively, then the 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 with respect to $\mathbf{x}$, then by chain rule of gradients we have:

$$\frac{d\ell}{dA} = B\left(\frac{d\ell}{d\mathbf{x}}\right)^T, \qquad \frac{d\ell}{dB} = A\left(\frac{d\ell}{d\mathbf{x}}\right). \tag{3.3}$$

As long as the gradients of the features A and B can be computed efficiently the entire model can be trained in an end-to-end manner. The scheme is illustrated in Figure 3.3.

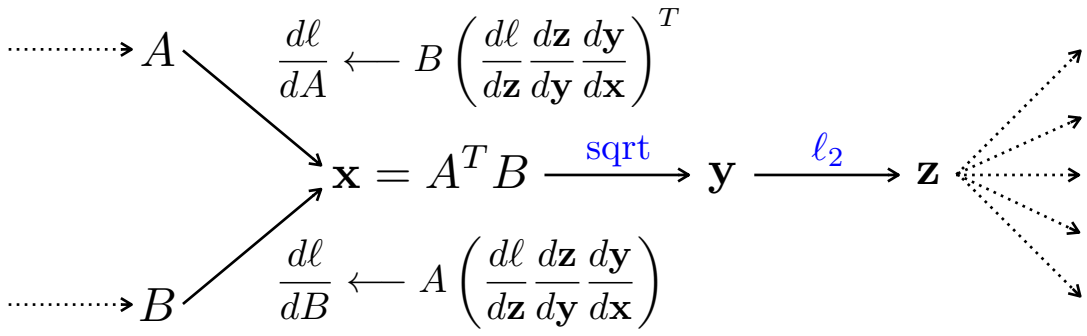


Figure 3.3. Flow of gradients in a B-CNN.

3.2 Relation to classical texture representations

In this section we show that various *orderless* texture descriptors we discussed in Chapter 2 can be written in the bilinear form and derive variants that are end-to-end trainable. Since the properties of texture are usually translationally invariant, most texture representations are based on orderless aggregation of local image features, *e.g.*, sum or max operation. A non-linear encoding is typically applied before aggregation of local features to improve their representation power. Additionally, a normalization of the aggregated feature (*e.g.*, power and ℓ_2) is done to increase invariance. Thus, texture representations can be defined by the choice of the *local features*, the *encoding function*, the *pooling function*, and the *normalization function*. To simplify further analysis, we decompose the feature function f as $f(l, \mathcal{I}) = g(h(l, \mathcal{I})) = g(\mathbf{x})$ to denote the explicit dependency on the image and the location of h and additional non-linearities g.

3.2.1 Classical texture representations as bilinear models

Bag-of-visual-words (BoVW) [25] obtains a set of visual words by clustering and assign each feature $\mathbf{x}$ to the closest cluster center (also called "hard assignment") and the image is represented as a histogram denoting frequencies of each visual word. If we denote $\eta(\mathbf{x})$ as the *one-hot encoding* that is 1 at the index of the closest center of

$\mathbf{x}$ and zero elsewhere, then BoVW can be written as a bilinear model with $g_A(\mathbf{x}) = 1$ and $g_B(\mathbf{x}) = \eta(\mathbf{x})$.

The VLAD representation[62] encodes a descriptor $\mathbf{x}$ as $(\mathbf{x}-\boldsymbol{\mu}_k)\otimes\eta(\mathbf{x})$, where $\otimes$ is the *kronecker product*, $\boldsymbol{\mu}_k$ is the closest center to $\mathbf{x}$, and $\eta(\mathbf{x})$ is the one-hot encoding of $\mathbf{x}$ as before. These encodings are aggregated across the image by sum pooling. Thus VLAD can be written as a bilinear model with $g_A(\mathbf{x}) = [\mathbf{x} - \boldsymbol{\mu}_1; \mathbf{x} - \boldsymbol{\mu}_2; \ldots; \mathbf{x} - \boldsymbol{\mu}_k]$. Here, g_A has k rows each corresponding to a center. And $g_B(\mathbf{x}) = \text{diag}(\eta(\mathbf{x}))$, a matrix with $\eta(\mathbf{x})$ in the diagonal and 0 elsewhere. Notice that the feature functions for VLAD output a matrix with $k > 1$ rows at each location.

The FV representation [101] computes both the first order $\boldsymbol{\alpha_i} = \boldsymbol{\Sigma}_i^{-\frac{1}{2}}(\mathbf{x} - \boldsymbol{\mu}_i)$ and second order $\boldsymbol{\beta}_i = \boldsymbol{\Sigma}_i^{-1}(\mathbf{x} - \boldsymbol{\mu}_i) \odot (\mathbf{x} - \boldsymbol{\mu}_i) - 1$ statistics, which are aggregated and weighted by the Gaussian mixture model (GMM) posteriors $\theta(\mathbf{x})$. Here $\odot$ denotes element-wise multiplication. Thus, FV can be written as a bilinear model with $g_A = [\boldsymbol{\alpha}_1\ \boldsymbol{\beta}_1; \boldsymbol{\alpha}_2\ \boldsymbol{\beta}_2; \ldots; \boldsymbol{\alpha}_k\ \boldsymbol{\beta}_k]$ and $g_B = \text{diag}(\theta(\mathbf{x}))$.

The O2P representation [19] computes the covariance statistics of SIFT features within a region, followed by log-Euclidean mapping and power normalization. Their approach was shown to be effective for semantic segmentation. O2P can can be written as a bilinear model with symmetric features, *i.e.*, $f_A = f_B = f_{\text{sift}}$, followed by pooling and non-linearities.

The appearance-based cluster centers learned by the encoder, $\eta(\mathbf{x})$ or $\theta(\mathbf{x})$, in the BoVW, VLAD and FV representations can be thought of as part detectors. Thus, by modeling the joint statistics of the encoder $\eta(\mathbf{x})$ or $\theta(\mathbf{x})$, and the appearance $\mathbf{x}$, the models can effectively describe appearance of parts regardless of where they appear in the image. This is particularly useful for fine-grained recognition where objects are not localized in the image.

	Exact formulation		End-to-end trainable formulation	
Model	$g_B(\mathbf{x})$	$g_A(\mathbf{x})$	$g_B(\mathbf{x})$	$g_A(\mathbf{x})$
VLAD	$\text{diag}(\eta(\mathbf{x}))$	$\mathbf{x}-\boldsymbol{\mu}$	$\text{diag}(\bar{\eta}(\mathbf{x}))$	$\mathbf{x}-\boldsymbol{\mu}$
FV	$\text{diag}(\theta(\mathbf{x}))$	$[\boldsymbol{\alpha},\boldsymbol{\beta}]$	$\text{diag}(\bar{\eta}(\mathbf{x}))$	$[\mathbf{x}-\boldsymbol{\mu},(\mathbf{x}-\boldsymbol{\mu})\odot(\mathbf{x}-\boldsymbol{\mu})]$
BoVW	$\eta(\mathbf{x})$	1	$\bar{\eta}(\mathbf{x})$	1
O2P	$\mathbf{x}$	$\mathbf{x}$	$\mathbf{x}$	$\mathbf{x}$

Table 3.1. Texture encoders such as VLAD, FV, BoVW and O2P can be written as outer products of the form ${g_A}^T g_B$. On the right are their end-to-end trainable formulations that simplify gradient computations by replacing "hard assignment" η with "soft assignment" $\bar{\eta}$, ignoring variance normalization for FV, *etc.* For the symmetric case (*i.e.*, when $f_A = f_B$) bilinear pooling is identical to O2P. See Section 3.2.2 for details.

3.2.2 End-to-end trainable formulations

Prior work on fine-grained recognition using texture encoders [23], [52] did not learn the features in an end-to-end manner. Below we describe end-to-end trainable approximation of VLAD proposed by [3] called NetVLAD, and present similar formulations for all texture representations described in the earlier section. The ability to directly fine-tune these models leads to significant improvements in accuracy across a variety of fine-grained datasets. Table 3.1 summarizes the end-to-end trainable approximations.

The first simplification was to replace the "hard assignment" $\eta(\mathbf{x})$ in g_B by a differentiable "soft assignment" $\bar{\eta}(\mathbf{x})$. Given the k-th cluster center μ_k, the k-th component of the soft assignment vector for an input $\mathbf{x}$ is given by,

$$\bar{\eta}_k(\mathbf{x}) = \frac{e^{-\gamma||\mathbf{x}-\mu_k||^2}}{\sum_{k'} e^{-\gamma||\mathbf{x}-\mu_{k'}||^2}} = \frac{e^{{\mathbf{w}_k}^T\mathbf{x}+b_k}}{\sum_{k'} e^{{\mathbf{w}_{k'}}^T\mathbf{x}+b_{k'}}} \tag{3.4}$$

where $\mathbf{w}_k = 2\gamma\mu_k$, $b_k = -\gamma||\mu_k||^2$ and γ is a parameter of the model. This is simply the `softmax` operation applied after a convolution layer with a bias term, and can be implemented using standard CNN building blocks. The function g_A remains unchanged $[\mathbf{x}-\mu_1;\mathbf{x}-\mu_2;\ldots;\mathbf{x}-\mu_k]$. The second simplification is to decouple the

dependence on μ of both the g_A and g_B during training which makes gradient computation easier. Thus in NetVLAD during training the weights $\mathbf{w}_k$, b_k and μ_k are independent parameters.

We extend the NetVLAD to NetFV by appending the second order statistics to the feature g_A, *i.e.*, $g_A = [\mathbf{x} - \mu_1, (\mathbf{x} - \mu_1)^2; \mathbf{x} - \mu_2, (\mathbf{x} - \mu_2)^2; \ldots; \mathbf{x} - \mu_k, (\mathbf{x} - \mu_k)^2]$. Here, the squaring is done in an element-wise manner, *i.e.*, $(\mathbf{x} - \mu_i)^2 = (\mathbf{x} - \mu_i) \odot (\mathbf{x} - \mu_i)$. The feature g_B is kept identical to NetVLAD. This simplification discards the covariances and priors present in the true GMM posterior used in the FV model. Similarly, the *NetBoVW* approximation to BoVW replaces the hard assignments by soft assignments $\bar{\eta}(\mathbf{x})$ computed in a manner similar to NetVLAD.

The O2P representation is identical to B-CNN when the feature functions f_A and f_B are identical. However, the O2P representation applies a Log-Euclidean (matrix logarithm) mapping to the pooled representation which is rather expensive to compute since it involves an Eigenvalue decomposition and currently does not have efficient implementation on GPUs. This significantly slows the forward and gradient computations of the entire network. In addition, back-propagating the gradients through matrix logarithm is not straightforward and recent work on DeepO2P proposed by Ionescu *et al.* [60] computing the gradients via back-propagation through SVD was suffering from the numerical instability. We skip Log-Euclidean to efficiently fine-tune the model right here. In Chapter 4 we will discuss an alternative normalizing O2P representation using matrix square-root which can be approximated efficiently and achieve better accuracy comparing to matrix logarithm.

3.3 Image Classification Experiments

We outline the models used in our experiments in Section 3.3.1. We then provide a comparison of various B-CNNs to prior work on fine-grained recognition in Section 3.3.2, and texture and scene recognition in Section 3.3.3.

3.3.1 Models and training setup

Below we describe various models used in our experiments:

FV with SIFT. We implemented a FV based using dense SIFT features [101] extracted using VLFEAT [123]. The image is first resized to 448×448 and SIFT features with a *bin size* 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.

CNN with fully-connected (FC) layers. This is a standard baseline where the features are extracted from the last FC layer, *i.e.*, before the `softmax` layer of a CNN. The input image is resized to 224×224 (the input size of the CNN) and mean-subtracted before propagating it though the CNN. We consider two different representations: 4096 dimensional *relu7* layer outputs of both the VGG-M network [20] and the 16-layer VGG-D network [112].

FV/NetFV with CNNs. This denotes the method of [23] that builds a descriptor using FV pooling of CNN filter bank responses with 64 GMM components. One modification over [23] 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*. We consider two representations based on the VGG-M and VGG-D network where the features are extracted from the *relu5* and *relu5_3* layers respectively.

VLAD/NetVLAD with CNNs. This approach builds VLAD descriptors on CNN filter banks responses. We use the same setting as FV and aggregate the CNN features with VLAD/NetVLAD pooling with 64 cluster centers.

BoVW/NetBoVW with CNNs. For BoVW we construct a vocabulary of 4096 words using k-means on top of the CNN features. For NetBoVW we use a 4096-way `softmax` layer as an approximation to the hard assignment. We use the same setting as FV and VLAD for feature extraction.

B-CNNs. These are models presented in Section 3.1 where features from two CNNs are pooled using an outer product. When the two CNNs are identical the model is a extension of O2P using deep features without the log-Euclidean normalization. We consider several B-CNNs – (i) one with two identical VGG-M networks truncated at the *relu5* layer, (ii) one with a VGG-D and VGG-M network truncated at the *relu5_3* and *relu5* layer respectively, and (iii) initialized with two identical VGG-D networks truncated at the *relu5_3* layer. The input images are resized to 448×448 and features are extracted using the two CNNs. The VGG-D network produces 28×28 output compared to 27×27 of the VGG-M network, so we downsample the VGG-D output by ignoring a row and column when combining it with the VGG-M output. The bilinear feature for all these models is of size 512×512. Note that the options (i) and (iii) result in symmetric models where the two networks share all parameters (which is also the case when they are fine-tuned due to symmetry), and hence have the same memory overhead and speed as a single network evaluation in practice.

3.3.1.1 Fine-tuning

For fine-tuning we add k-way linear + softmax layer where k is the number of classes in the fine-grained dataset. The parameters of the linear layer are initialized randomly. We adopt a two-step training procedure of [11] where we first train the linear layer using logistic regression 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. Although, the exact VLAD, FV, BoVW models cannot be directly fine-tuned, we report results using *indirect* fine-tuning where the local CNN features are extracted from the networks fine-tuned with FC layers. We found this improves accuracy; however, direct fine-tuning end-to-end trainable formulations such NetFV is significantly better.

3.3.1.2 SVM training and evaluation

In all our experiments before and after fine-tuning, training and validation sets are combined to train one-vs-all linear SVMs on the extracted features with 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 at test time average the predictions of the image and its flipped copy. SVM training provides 1-3% improvement over logistic regression with the VGG-M networks, but provides *negligible* improvement with the VGG-D networks. Test time flipping improves performance by 0.5% on average for the VGG-M networks, while has *negligible* impact on the accuracy for the VGG-D networks. Performance is measured as the percentage of correctly classified images for all datasets.

3.3.2 Fine-grained recognition

We evaluate methods on following fine-grained datasets and report the per-image accuracy in Table 3.3.2.1.

CUB-200-2011 [124] dataset contains 11,788 images of 200 bird species which are split into roughly equal train and test sets with detail annotation of parts and bounding boxes. Notice that in all our experiments, we only use image labels during training without any part or bounding box annotation. In the following sections, "birds" refers to the results on this dataset.

FGVC-aircraft dataset [95] 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.

Stanford cars dataset [73] 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.

NABirds [122] is larger than the CUB dataset consisting of 48,562 images of 555 spices of birds that include most that are found in North America. The work engaged citizen scientists to produce high-quality annotations in a cost-effective manner. This dataset also provides parts and bounding-box annotations, but we only use category labels for training our models.

3.3.2.1 Bird species classification

Comparison to baselines. Table 3.3.2.1 "bird" column shows results on the CUB-200-2011 dataset. The end-to-end approximations of texture representations (NetBoWV, NetVLAD, NetFV) improve significantly after fine-tuning. Exact models with indirect fine-tuning (*i.e.*, fine-tuned with FC layers) also improve, but the improvement is smaller (shown in gray italics in Table 3.3.2.1). With fine-tuning the single-scale FV models outperforms the multi-scale results reported in [23] – 49.9% using VGG-M and 66.7% using VGG-D network. B-CNNs offer the best accuracy across all models with the best performing model obtaining **84.1%** accuracy (VGG-M + VGG-D). The next best approach is the NetVLAD with 81.9% accuracy.

We also trained the B-CNN (VGG-M + VGG-D) model on the much larger *NABirds dataset*. For this model we skipped the SVM training step and report the accuracy using `softmax` layer predictions. This model achieves **79.4%** accuracy outperforming a fine-tuned VGG-D network that obtains 63.7% accuracy. Van Horn *et al.* [122] obtains **75%** accuracy using AlexNet and part annotations at test time, while

the "neural activation constellations" approach [111] obtains **76.3%** accuracy using a GoogLeNet architecture [117].

Comparison to other techniques. Table 3.3.2.1 shows other top-performing methods on this dataset. The dataset also provides bounding-box and part annotations and techniques differ based on what annotations are used at training and test time (also shown in the Table). Two early methods that performed well when bounding-boxes are not available at test time are 73.9% of the "part-based R-CNN" [133] and 75.7% of the "pose-normalized CNN" [11]. These methods are based on AlexNet [74] and can be improved with deeper and more accurate networks such as the VGG-D. For example, the SPDA-CNN [132] trains better part detectors and feature representations using the VGG-D network and report 84.6% accuracy. Krause *et al.* [72] report 82.0% accuracy using a weakly-supervised method to learn part detectors, followed by the part-based analysis of [133] using the VGG-D network. However, our approach is simpler and faster since it does not rely on training and evaluating part detectors. One of the top-performing approaches that does not rely on additional annotations is the Spatial Transformer Networks [61]. It obtains **84.1%** accuracy using the batch-normalized Inception network [59].

Effect of normalization Tab. 3.3 shows the results of the B-CNN (VGG-M + VGG-D) model without fine-tuning using various normalizations of the bilinear vector. The model with both square-root and ℓ_2 normalization achieves 80.1% accuracy. Only square-root normalization results in small drop in accuracy to 79.4%. Only ℓ_2 normalization causes a larger drop in accuracy to 77.3%. No normalization at all is significantly worse at 74.7% accuracy. This shows that both these normalizations are useful and square-root has a higher effect on the performance than ℓ_2.

Common mistakes. Figure 3.4 shows the top six pairs of classes that are confused by our fine-tuned B-CNN (VGG-M + VGG-D) model. The most confused pair of classes is "American crow" and "Common raven". The differences lie in the wing-

features	train	test	encoding	birds		aircrafts		cars	
				w/o ft	w/ ft	w/o ft	w/ ft	w/o ft	w/ ft
SIFT			FV	18.8	-	61.0	-	59.2	-
VGG-M (*relu5*)	*n/a*	*n/a*	FC	52.7	58.8	44.4	63.4	37.3	58.6
			BoVW	41.9	*43.8*	56.2	*60.1*	54.2	*58.4*
			NetBoVW	47.9	48.6	58.8	65.9	60.3	66.1
			VLAD	66.5	*70.5*	70.5	*74.8*	75.3	*78.9*
			NetVLAD	66.8	72.1	70.7	76.7	76.0	83.7
			FV	61.1	*64.1*	64.3	*71.2*	70.8	*77.2*
			NetFV	64.5	71.7	68.6	75.5	72.3	81.8
			B-CNN	72.0	78.1	72.7	79.5	77.8	86.5
VGG-D (*relu5_3*)	*n/a*	*n/a*	FC	61.0	70.4	45.0	76.6	36.5	79.8
			BoVW	56.6	*58.8*	61.9	*71.3*	62.7	*73.9*
			NetBoVW	65.9	69.7	65.1	74.0	71.0	76.7
			VLAD	78.0	*79.0*	75.2	*80.6*	81.9	*85.6*
			NetVLAD	77.9	81.9	75.3	81.8	82.1	88.6
			FV	71.3	*74.7*	70.4	*78.7*	75.2	*85.7*
			NetFV	73.9	79.9	71.5	79.0	77.9	86.2
			B-CNN	80.1	84.0	77.7	86.9	82.9	90.6
VGG-M+ VGG-D			B-CNN	80.1	84.1	78.0	86.6	83.9	91.3
Inception-BN	*n/a*	*n/a*	STNs [61]	84.1		-		-	
VGG-D	Box	*n/a*	Krause *et al.* [72]	82.0		-		**92.6**	
VGG-D	Box+P	Box	SPDA-CNN [132]	84.6		-		-	
VGG-D	Part	*n/a*	Zhang *et al.* [137]	**85.9**		-		-	
AlexNet	B+P	*n/a*	Part-based RCNN [133]	73.9		-		-	
AlexNet	B+P	*n/a*	Branson *et al.* [11]	75.7		-		-	
VGG-D	Box	Box	BoT [126]	-		**88.4**		92.5	
FV+SIFT	*n/a*	*n/a*	Gosselin *et al.* [52]	-		80.7		82.7	

Table 3.2. We compare various texture representations and prior work (separated by a double line) on the birds [124], aircrafts [95], and cars [73] datasets. The first column lists the features used in the encoding followed by the pooling strategy. FC pooling corresponds to fully-connected layers on top these intermediate layer features such that it corresponds to the penultimate layer of the original network. The second and third columns show additional annotations used during training and testing. Results are shown without and with domain-specific fine-tuning. Directly fine-tuning the approximate models leads to better performance than indirectly fine-tuning (shown in gray italics). B-CNN models achieve the best accuracy across texture representations. The first, second, and third best texture models are marked with red, blue and yellow colors respectively.

Figure 3.4. Top six pairs of classes that are most confused with each other on the CUB dataset. In each row we show the images in the test set that were most confidently classified as the class in the other column.

spans, habitat, and voice, none of which are easy to measure from the image. Other confused classes are also similar – various Shrikes, Terns, Flycatchers, Cormorants and Gulls. We note that the dataset has an estimated 4.4% label noise hence some of these errors may come from incorrect labeling [122].

3.3.2.2 Aircraft variant classification

The trends among the baselines are similar to those in birds with a few exceptions. The FV with SIFT is remarkably good (61.0%) and comparable to some of the CNN baselines. Compared to the birds, the effect of fine-tuning is significantly larger

normalization	accuracy	mAP
square-root + ℓ_2	80.1	81.3
square-root only	79.4	77.9
ℓ_2 only	77.3	79.6
none	74.7	70.9

Table 3.3. Effect of normalization on the B-CNN (VGG-M + VGG-D) model w/o fine-tuning on the CUB-200-2011 dataset ("birds" setting).

for models based on the VGG-D network suggesting a larger domain shift from the ImageNet dataset. As aircrafts appear mostly on the image center, cropping the central image improves the accuracy over resizing whole image. We resize the images into 512×512 and then crop the central 448×448 as input. This achieves the best performance of **86.9%** by the B-CNN (VGG-D) model.

3.3.2.3 Car model classification

FV with SIFT does well on this dataset achieving 59.2% accuracy. The effect of fine-tuning on cars is larger in comparison to birds and airplanes. Once again the B-CNNs outperform all the other baselines with the B-CNN (VGG-D + VGG-M) model achieving **91.3%** accuracy. The best accuracy on this dataset is by Krause *et al.* [72] which obtains 92.6% accuracy. This is closely matched by **92.5%** accuracy of the discriminative patch triplets [126].

3.3.3 Texture and scene recognition

We experiment on three texture datasets – the *Describable Texture Dataset* (DTD) [22], *Flickr Material Dataset* (FMD) [110], and *KTH-TISP2-b* (KTH-T2b) [18]. DTD consists of 5640 images labeled with 47 describable texture attributes. FMD consists of 10 material categories, each of which contains 100 images. Unlike DTD and FMD where images are collected from the Internet, KTH-T2b contains 4752 images of 11 materials captured under controlled scale, pose, and illumination. The KTH-T2b dataset

	FV			B-CNN		
dataset	$s = 1$	$s = 2$	ms	$s = 1$	$s = 2$	ms
DTD	67.8 ±0.9	70.6 ±0.9	73.6 ±1.0	69.6 ±0.7	71.5 ±0.8	72.9 ±0.8
FMD	75.1 ±2.3	79.0 ±1.4	80.8 ±1.7	77.8 ±1.9	80.7 ±1.5	81.6 ±1.7
KTH-T2b	74.8 ±2.6	75.9 ±2.4	77.9 ±2.0	75.1 ±2.8	76.4 ±3.5	77.9 ±3.1
MIT indoor	70.1	78.2	78.5	72.8	77.6	79.0

Table 3.4. Mean per-class accuracy on texture and scene datasets. Results for input images at different scales $s = 1$, $s = 2$ and ms correspond to a size of 224×224, 448×448 and multiple sizes respectively.

splits the images into four samples for each category. We follow the standard protocol by training on one sample and test on the remaining three. On DTD and FMD, we randomly divide the dataset into 10 splits and report the mean accuracy across splits. Besides these, we also evaluate our models on *MIT indoor scene* dataset [105]. Indoor scenes are weakly structured and orderless texture representations have been shown to be effective here. The dataset consists of 67 indoor categories and a defined training and test split.

We compare B-CNN to the prior state-of-the-art approach of FV pooling of CNN features [23] using the VGG-D network. These results are without fine-tuning. On the MIT indoor dataset fine-tuning B-CNNs leads to a small improvement 72.8% → 73.8% using *relu5_3* at $s = 1$, while on the other datasets the improvements were negligible, likely due to the relatively small size of these datasets. Table 3.4 shows the results obtained by features from a single scale and features from multiple scales $2^s, s \in \{1.5\text{:-}0.5\text{:-}3\}$ relative to the 224×224 image using B-CNN and FV representations. We discard scales for which the image is smaller than the size of the receptive fields of the filters, or larger than 1024^2 pixels for efficiency. Across all scales of the input image the performance of the two approaches are identical. Multiple scales consistently lead

to an improvement in accuracy. These results show that the B-CNNs are comparable to the FV pooling for texture recognition. One drawback is that the FV features with 64 GMM components has smaller in size (64×2×512) than the bilinear features (512×512). However, bilinear features are highly redundant and their dimensionality can be reduced by an order of magnitude without loss in performance (see Section 3.4).

3.4 Dimensionality reduction

The outer product of CNN features generates very high dimensional image descriptors, *e.g.*, 262K for the B-CNN models in Table 3.3.2.1. These features are highly redundant and their dimensionality can be reduced by an order of magnitude without loss in classification performance. Prior work [62] has also shown that in the context of SIFT-based FV and VLAD, highly compact representations can be obtained.

In this section we investigate the trade-off between accuracy and feature dimension for various texture models proposed in Section 3.2 for fine-grained recognition. For NetVLAD and NetFV the feature dimension can be varied by changing the number of cluster centers. For B-CNNs, consider the case where the outer product is computed among features $\mathbf{x}$ and $\mathbf{y}$. There are several strategies for reducing the feature dimension:

(1) Projecting the outer product into a lower dimensional space, *i.e.*, $\Phi(\mathbf{x}, \mathbf{y}) = \texttt{vec}(\mathbf{x}^T\mathbf{y})\mathbf{P}$, where $\mathbf{P}$ is a projection matrix and the `vec` operator reshapes the matrix into a vector.

(2) Projecting both the features into a lower-dimensional space and computing outer product, $\Phi(\mathbf{x}, \mathbf{y}) = (\mathbf{x}\mathbf{A})^T(\mathbf{y}\mathbf{B})$, where $\mathbf{A}$, $\mathbf{B}$ are projection matrices.

(3) Projecting one of the features into a lower-dimensional space and computing the outer product, *i.e.*, by setting $\mathbf{B}$ to an identity matrix in the previous approach.

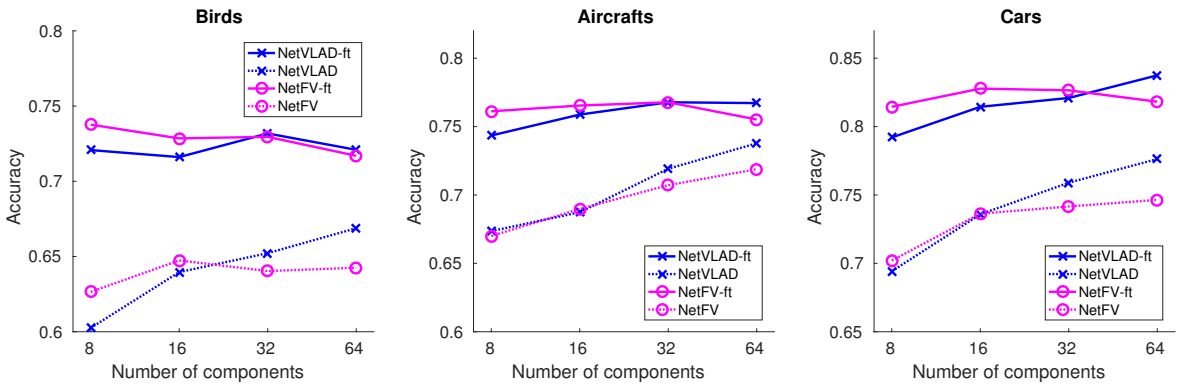


Figure 3.5. Performance of NetVLAD and NetFV models encoding VGG-M *relu5* features with different number of cluster centers on fine-grained datasets before (dashed lines) and after (solid lines) fine-tuning. Given the same number of cluster centers, the feature dimension of NetFV representation is twice as large as NetVLAD.

In each case, the projection matrices can be initialized using Principal Component Analysis (PCA). Although the first approach is straightforward, computing the PCA is computationally expensive due to the high dimensionality of the features (the covariance matrix of the outer product has d^4 entries for d-dimensional features). The second approach is computationally attractive but the outer product of two PCA projected features results in a significant reduction in accuracy as shown in our earlier work [83], and more recently in [43]. We believe this is because after the PCA rotation, the features are no longer correlated across dimensions. Remarkably, reducing the dimension of only one feature using PCA (third option) works well in practice. While the projection can be initialized using PCA, they can be trained jointly with the classification layers. In addition to reducing the feature dimension, It also breaks the symmetry of the features when identical networks are used and is an example of a partially shared feature pipeline (Figure 3.2b). It also resembles the computations of VLAD and FV representations where both f_A and f_B are based on the same underlying feature.

The accuracy as a function of feature dimension shown in Figure 3.5 for NetFV and NetVLAD. These results are obtained by varying the number of cluster centers.

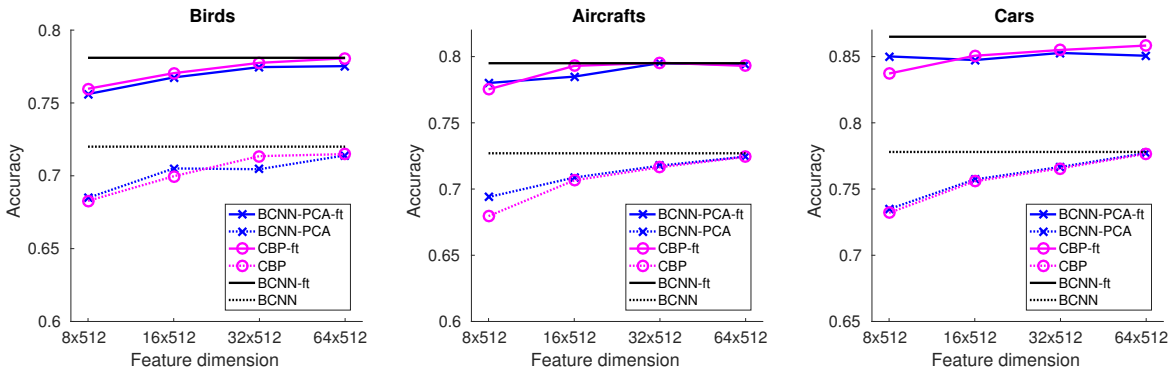


Figure 3.6. Performance of B-CNNs using VGG-M *relu5* features as function of feature dimension before (dashed lines) and after (solid lines) fine-tuning. One of the 512 dimensional feature is projected using PCA to k dimensions leading to a outer product of size $k \times 512$ (see Section 3.4 for details). The performance using Compact Bilinear Pooling (CBP) [43] and the full 512×512-dimensional model is shown in red and black respectively.

The results indicate the performance of NetVLAD and NetFV do not improve with more cluster centers beyond 32.

Figure 3.6 shows the same for B-CNN features and compares the PCA approach to Compact Bilinear Pooling (CBP) technique [43]. CBP approximates the outer product using a product of sparse linear projections of features with a Tensor Sketch [103]. The performance of the full model with 512×512 dimensions with and without fine-tuning is shown as a straight line. On birds and aircrafts the dimensionality can be reduced by 16× (*i.e.*, to 32×512) with a less than 1% loss in accuracy. In comparison, NetVLAD with the same feature size (*i.e.*, with 32 components) is about 3-4% less accurate. Overall, for a given budget of dimensions the projected B-CNNs outperform NetVLAD and NetFV representations. The PCA approach is slightly worse than CBP. However, one advantage is that PCA can be implemented as a dense matrix multiplication which is empirically 1.5× faster than CBP which involves computing Fourier transforms and their inverses.

	Bilinear pooling			FC pooling	
data aug.	f1	f5	f25	f1	f25
error@1	38.7	37.1	36.6	46.4	39.6
error@5	17.0	16.3	16.0	22.5	17.6

Table 3.5. Accuracy on the ILSVRC 2014 *validation* set using bilinear and FC pooling on top of *relu5* layer output of a VGG-M network. Both networks are trained from scratch on the ILSVRC 2012 training set with varying amounts of data augmentation.

3.5 Training B-CNNs on ImageNet LSVRC

We evaluate B-CNNs trained *from scratch* on the ImageNet LSRVC 2012 dataset [108]. In particular, we train a B-CNN with a *relu5* layer output of VGG-M network and compare it to the standard VGG-M network. This allows a direct comparison of bilinear pooling with FC pooling since the rest of the architecture is identical in both networks. Additionally, we compare the effect of implicit translational invariance obtained in CNNs by spatially "jittering" the data, as well a explicit translation invariance of B-CNNs due to the orderless pooling.

We train the two networks to classify 224×224 images with different amounts of spatial jittering – "f1" for flip, "f5" for flip + 5 translations, and "f25" for flip + 25 translations. Training is done with stochastic sampling where one of the jittered copies is randomly selected for each example. The parameters are randomly initialized and trained using stochastic gradient descent with momentum for a number of epochs. We start with a high learning rate and reduce it by a factor of 10 when the validation error stops decreasing, and continue till no further improvement is observed.

Table 3.5 shows the "top1" and "top5" validation errors for B-CNN and VGG-M. The validation error is reported on a single center cropped image. Note that we train all networks with neither PCA color jittering nor batch normalization and our baseline results are within 2% of the top1 errors reported in [20]. The VGG-M model achieves 46.4% top1 error with flip augmentation during training. The performance

improves significantly to 39.6% with f25 augmentation. B-CNN achieves 38.7% top1 error with f1 augmentation, outperforming VGG-M trained with f25 augmentation. The results show that B-CNN is discriminative, robust to translation and that explicit translation invariance is more effective.

This trend is also reflected in the design of the latest deep architectures such as Google Inception [117], Residual Networks [54] and DenseNet [58] that replace the fully-connected layers with global pooling layers. Although the experiment shows the improvement using B-CNN with VGG-M model training from scratch for large-scale image classification task, in this experiment we mainly intend to analyze the effect of explicitly modeling the invariance of translation comparing to achieving the invariance via data augmentation. We would like to note that the VGG models is relatively shallower comparing to state-of-the-art CNN models [117, 54, 58]. Recent work proposed by [81] presented the experiment and showed that training second-order features from scratch based on latest CNN backbones can further improve the recognition accuracy on large-scale image classification.

3.6 Discussion

One of the motivations for bilinear CNNs was to model the multiplicative feature interactions in a way similar to part-based models [10, 135, 11, 39, 133, 134], which were common approaches for fine-grained recognition. The multiplicative feature interactions implicitly model the spatial attention mechanism, and together with orderless pooling, the resultant representations are robust to object deformation. At the time when we published the models, they outperformed the previous state-of-the-art fine-grained recognition models by a significant margin and inspired subsequent works in this direction [26, 43, 137, 138, 139, 49, 116]. The particular case when the two feature streams share the same computation is related to using a second-order polynomial as the kernel function. Kernel pooling [26] extended the idea to model

higher-order feature interactions with compact approximation [43] and demonstrated how to approximate Gaussian RBF up to a given order. This further improved fine-grained recognition accuracy.

In Chapter 5, we visualize the CNN activations by showing the patches that highly activate to selected filters. We show that it is challenging for the models to achieve the part-appearance decomposition without supervision beyond category labels. This has motivated several works [137, 138] in attempts to explicitly model the attention to a pre-defined set of object parts with or without extra supervision. There were also other recent works [139, 49, 116] exploring attention models without using extra supervision to achieve better fine-grained recognition performance. The multiplicative feature interactions have also been shown to be effective in other applications such as aggregating spatio-temporal features for video action recognition [127, 21] and modeling multi-modal data such as appearance and motion [40], video and audio [130], and vision and language [42, 130].

In summary, we presented the B-CNN architecture that aggregates second-order statistics of CNN activations resulting in an orderless representation of an image. The architecture can be optimized end-to-end for the training from scratch on large datasets or domain-specific fine-tuning for transfer learning. We also compared B-CNNs to both exact and approximate variants of deep texture representations and studied the accuracy and memory trade-offs they offer. Moreover, these representations are redundant, and in most cases, their dimension can be reduced by order of magnitude without significant loss in accuracy. We also compared the effect of translational invariance achieved by orderless aggregation against the data augmentation approach with spatial jittering for general object classification on the ImageNet dataset.

CHAPTER 4

IMPROVING FEATURE AGGREGATION

Bursty features, as shown in Figure 4.1, refer to the visual elements that appear many times in an image. The phenomenon happens when a homogeneous texture occupies a large portion of the image. Bag-of-visual-words framework as image representations suffers from feature burstiness for image classification. The framework counts each appearance of individual features independently, ignoring the fact that the same feature is likely to appear multiple times. Feature normalization is an approach to mitigating this problem. In Chapter 3, we normalize the B-CNN representation with element-wise sign square-root, which scales feature co-occurrences by square root and reduce the values of frequent co-occurrences. In this chapter, we explore two other normalization techniques: (1) spectral normalization with matrix functions and (2) democratic aggregation. The first approach scales the spectrum of covariance matrix and is related to Log-Euclidean mapping to linearize the manifold for covariance matrices with matrix logarithm. Democratic aggregation measures the frequency of each outer product by the similarity to the aggregated image representation. The normalization is achieved by a weighted sum of outer products with the weighting coefficients inversely proportional to the frequency. We study the approaches to efficiently computing the two normalization techniques and discuss the tradeoff between recognition accuracy, speed, and memory usage.

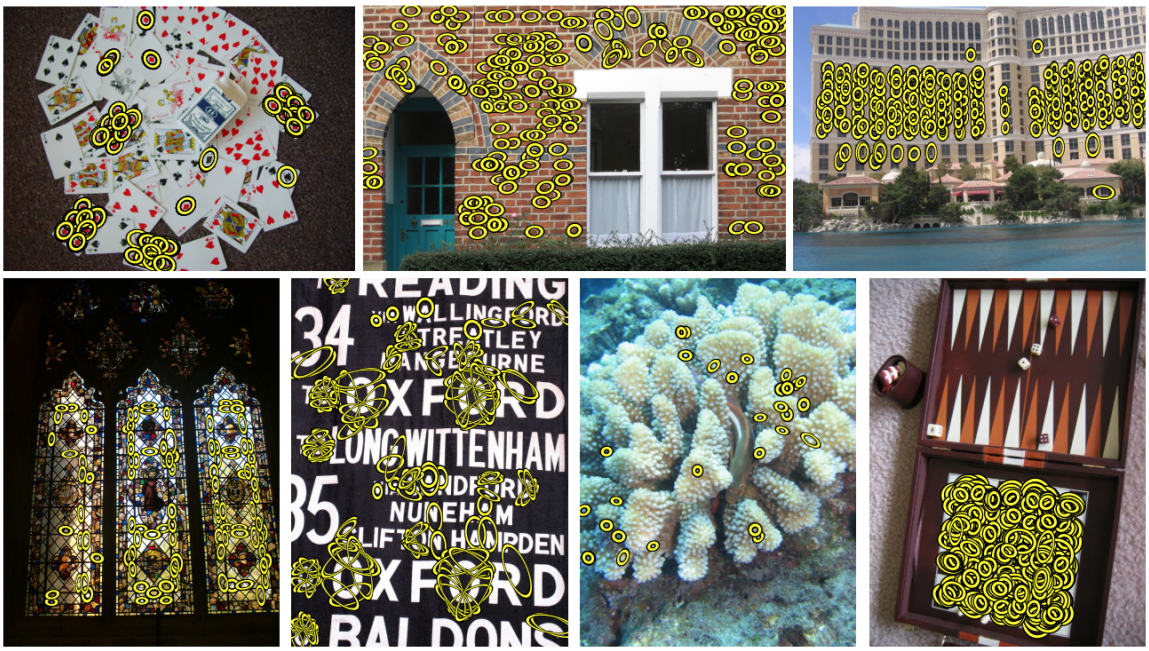


Figure 4.1. Burstiness of visual elements. Bursty visual elements shown in the yellow circles are extracted from the homogeneously-textured regions. The figure comes from [63].

4.1 Spectral normalization using matrix functions

In this section we analyze the bilinear features when the two features are identical which results in a symmetric positive semi-definite matrix. The symmetric B-CNNs are identical to the Second-Order Pooling (O2P) [19] popularized for semantic segmentation. The network architecture is illustrated in Figure 4.2. Given an image a CNN is used to extract a set of features $\mathbf{x}_i$ across locations $i = 1, 2, \ldots, n$. The bilinear pooling extracts the second-order statistics and adds a small positive value ϵ to the diagonal resulting in matrix A given by:

$$A = \frac{1}{n}\left(\sum_{i=1}^{n} \mathbf{x_i}\mathbf{x_i}^T\right) + \epsilon I \tag{4.1}$$

Given a feature $\mathbf{x_i}$ of d dimensions the matrix A is of size $d \times d$. We showed that normalization of the matrix A is critical for good performance in Table 3.3 in. In

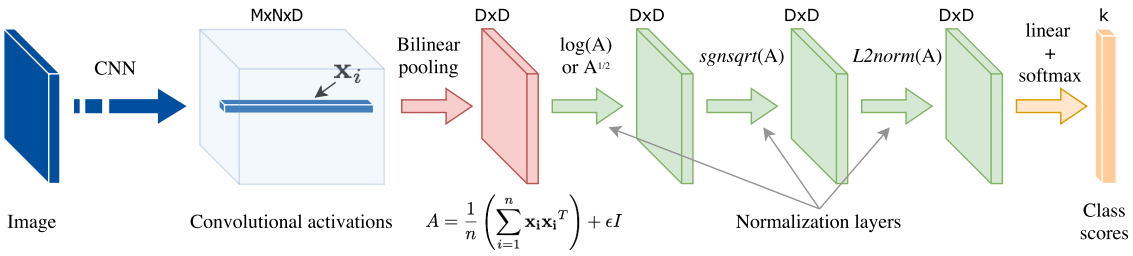


Figure 4.2. Improved B-CNN architecture with a $\log(A)$ or $A^{1/2}$, signed square-root, and ℓ_2 normalization layers added after the bilinear pooling of CNN activations.

particular, elementwise signed square-root ($x \leftarrow sign(x)\sqrt{|x|}$) and ℓ_2 normalization is applied to the matrix A before it is plugged into linear classifiers. Both the pooling and normalization steps are efficient and piecewise differentiable and hence the entire network can be trained in an end-to-end manner by back-propagating the gradients of the objective function (*e.g.*, cross-entropy loss for classification tasks).

The improved B-CNN architecture additionally applies a matrix function normalization to the matrix A after pooling (Figure 4.2). In particular we consider the matrix logarithm $\log(A)$ originally proposed in the O2P scheme [19] and the matrix power function A^p for fractional positive values of $0 < p < 1$. Of particular interest is when $p = 1/2$ which corresponds to the matrix square-root defined as a matrix Z such that $ZZ = A$. Unlike elementwise transformations, matrix-functions require computations that depend on the entire matrix. Approaches include variants of Newton iterations or via a Singular Value Decomposition (SVD) (described next).

4.1.1 Matrix functions and their gradients via SVD

Ionescu *et al.* [60] explore matrix back-propagation for training CNNs and using techniques for computing the derivative of matrix functions (*e.g.*, [93]). Given matrix A with a SVD given by $A = U\Sigma U^T$, where the matrix $\Sigma = diag(\sigma_1, \sigma_2, \ldots, \sigma_n)$, the matrix function f can be written as $Z = f(A) = Ug(\Sigma)U^T$, where g is applied to the

elements in the diagonal of Σ. Given the gradient of a scalar loss L with respect to Z, the gradient of L with respect to A can be computed as:

$$\frac{\partial L}{\partial A} = U\left\{\left(K^T \odot \left(U^T \frac{\partial L}{\partial U}\right)\right) + \left(\frac{\partial L}{\partial \Sigma}\right)_{diag}\right\}U^T. \tag{4.2}$$

Here $\odot$ denotes element-wise matrix multiplication. The matrix K is a skew-symmetric matrix given by $K_{i,j} = 1/(\sigma_i - \sigma_j)\mathbf{I}(i \neq j)$, where $\mathbf{I}(\cdot)$ is the indicator function. The gradients of L with respect to U and Σ are:

$$\frac{\partial L}{\partial U} = \left\{\frac{\partial L}{\partial Z} + \left(\frac{\partial L}{\partial Z}\right)^T\right\}Ug(\Sigma), \qquad \frac{\partial L}{\partial \Sigma} = g'(\Sigma)U^T\frac{\partial L}{\partial Z}U. \tag{4.3}$$

Here $g'(\Sigma)$ is the gradient of the g with respect to Σ. Since g is applied in an elementwise manner the gradients can be computed easily. For example for the matrix square-root

$$g'(\Sigma) = diag\left(\frac{1}{2\sqrt{\sigma_1}}, \frac{1}{2\sqrt{\sigma_2}}, \ldots, \frac{1}{2\sqrt{\sigma_n}}\right). \tag{4.4}$$

4.1.2 Effect of the exponent p in the matrix power normalization A^p.

Figure 4.3 shows that accuracy of non-fine-tuned B-CNNs with the VGG-D network using power normalization A^p for different values of the power p. The value $p = 1$ corresponds to the baseline B-CNN accuracy [83] where only elementwise signed square-root and ℓ_2 normalization are applied. The plots show that the matrix square-root ($p = 1/2$) works the best and outperforms the baseline B-CNN accuracy by a considerable margin.

4.1.3 Effect of combining various normalization schemes.

Table 4.1 shows the accuracy of non-fine-tuned B-CNNs using various normalization schemes. The baseline model is shown as the row with only the $sgnsqrt(A)$ column checked. Matrix square-root or the matrix logarithm normalization alone does not always improve over elementwise signed square-root normalization. However,

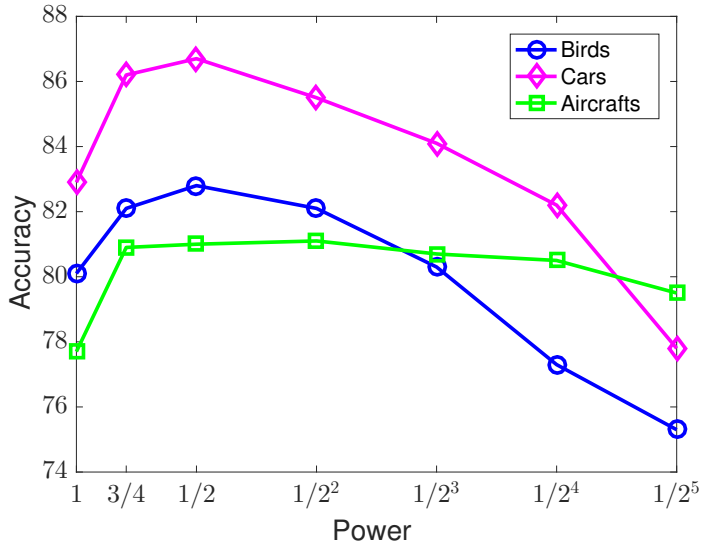


Figure 4.3. Accuracy *vs.* the exponent p.

when combined, the improvements are significant suggesting that the two normalization schemes are complementary. Overall the combination of matrix square-root normalization is better than the matrix logarithm normalization.

4.2 Matrix square-root and its gradients

4.2.1 Newton's method for computing the matrix square-root

A drawback of SVD based computations of matrix functions is that the computation on GPU is currently poorly supported and sometimes slower than CPU computations. In practice for the networks we consider the time taken for the SVD is comparable to the rest of the network evaluation. For smaller networks this step can become the bottleneck. Instead of computing the matrix square-root accurately, one can instead run a few iterations of a Netwon's method for root finding to the equation $F(Z) = Z^2 - A = 0$. Higham [56] describes a number of variants and analyzes their stability and convergence properties. One such is the Denman-Beavers iterations [30]. Given $Y_0 = A$ and $Z_0 = I$, where I is the identity matrix, the iteration is defined by

$$Y_{k+1} = \frac{1}{2}(Y_k + Z_k^{-1}), \quad Z_{k+1} = \frac{1}{2}(Z_k + Y_k^{-1}). \tag{4.5}$$

Network	Normalization			Accuracy on dataset		
	$\log(A)$	$A^{1/2}$	$sgnsqrt(A)$	Birds	Cars	Aircrafts
VGG-M			✓	72.0	77.8	74.7
	✓			70.8	77.4	77.1
		✓		70.3	76.8	75.0
	✓		✓	72.7	81.2	**81.0**
		✓	✓	**76.3**	**83.4**	80.7
VGG-D			✓	80.1	82.9	77.7
	✓			77.9	79.8	78.7
		✓		80.6	82.3	78.7
	✓		✓	81.1	85.1	**81.4**
		✓	✓	**82.8**	**86.7**	80.9

Table 4.1. Accuracy of B-CNNs with different normalization schemes with non fine-tuned networks. The best results are obtained with matrix-square root followed by element-wise signed square-root normalization. Notably the matrix square-root is better than the matrix logarithm normalization on most datasets for both networks.

The matrices Y_k and Z_k converge quadratically to $A^{1/2}$ and $A^{-1/2}$ respectively. In practice about 20 iterations are sufficient. However, these iterations are not GPU friendly either since they require computing matrix inverses which also lack efficient GPU implementations. A slight modification of this equation obtained by replacing the inverses using a single iteration of Newton's method for computing inverses [7] results in a different update rule called Newton-Schulz iterations given by:

$$Y_{k+1} = \frac{1}{2}Y_k(3I - Z_kY_k), \quad Z_{k+1} = \frac{1}{2}(3I - Z_kY_k)Z_k. \tag{4.6}$$

However, unlike the Denman-Beavers iterations the above iterations are only locally convergent, *i.e.*, they converge if $||A - I||_2 < 1$. In practice one can scale the matrix $A = \alpha A$ to satisfy this condition [8]. These iterations only involve matrix multiplications and are an order-of-magnitude faster than SVD computations on the GPU. Moreover, one can run these iterations for a small number of iterations (even one!) to trade-off accuracy and speed. Another advantage given by the above Newton

iterations is that the iterations could be unrolled as layers of computations whose derivatives could be derived 'for free' via automatic differentiation with latest deep learning packages.

4.2.2 Gradients of matrix-square root by solving a Lyapunov equation

We have discussed two options to compute the gradients for matrix square-root: 1) back-propagation via SVD and 2) automatic differentiation via Newton iterations. However, there are drawbacks to both approaches. The SVD-based approach is numerically unstable, as described in Section 4.2.3. Newton iterations, along with automatic differentiation, provide a straightforward approach for backward computation. However, this requires to cache the intermediate outputs from the iterations during forward computations. When the matrices are high-dimensional with a large number of iterations, the automatic differentiation approach is memory-demanding. The alternative approach is to derive the gradients via solving the Lyapunov equation.

Given a symmetric PSD matrix A, with $Z = A^{1/2}$, and a small change dA to the matrix A, the change dZ to the matrix Z satisfies the equation:

$$A^{1/2}dZ + dZA^{1/2} = dA. \tag{4.7}$$

This can be derived by applying the product rule of derivatives to the equation $ZZ = A$. From this one can derive the corresponding chain rule for the loss L as:

$$A^{1/2}\left(\frac{\partial L}{\partial A}\right) + \left(\frac{\partial L}{\partial A}\right)A^{1/2} = \frac{\partial L}{\partial Z}. \tag{4.8}$$

The above is a Lyapunov equation [90] which has a closed-form solution given by:

$$vec\left(\frac{\partial L}{\partial A}\right) = \left(A^{1/2} \otimes I + I \otimes A^{1/2}\right)^{-1} vec\left(\frac{\partial L}{\partial Z}\right). \tag{4.9}$$

Here $\otimes$ is the Kronecker product and the $vec(\cdot)$ operator unrolls a matrix to a vector. However, the Lyapunov equation can be solved more efficiently using the Bartels-Stewart algorithm [5] or the iterative solver proposed by [107]. The iterative solver is in favor due to its efficiency and simple implementation.

We can rewrite the Lyapunov equation in the following matrix form:

$$X = \begin{bmatrix} A^{1/2} & -\frac{\partial L}{\partial Z} \\ 0 & -A^{1/2} \end{bmatrix} = \begin{bmatrix} I & \frac{\partial L}{\partial A} \\ 0 & I \end{bmatrix} \begin{bmatrix} A^{1/2} & 0 \\ 0 & -A^{1/2} \end{bmatrix} \begin{bmatrix} I & -\frac{\partial L}{\partial A} \\ 0 & I \end{bmatrix} \tag{4.10}$$

from which we can derive the following equation as detailed in [56],

$$\text{sign}\left(\begin{bmatrix} A^{1/2} & -\frac{\partial L}{\partial Z} \\ 0 & -A^{1/2} \end{bmatrix} \right) = \begin{bmatrix} I & -2\frac{\partial L}{\partial A} \\ 0 & I \end{bmatrix} \tag{4.11}$$

where $\text{sign}(X)$ is matrix sign function which satisfies the property that $\text{sign}(X) = X(X^2)^{-1/2}$. The gradient of the loss with respect to the matrix A can be read off from the top-right block of the right matrix in equation 4.11. Newton-Schulz iterations for computing the matrix sign result in the following iterations for solving the Lyapunov equation:

$$Y_{k+1} = \frac{1}{2} Y_k (3I + Y_k^2), \quad Z_{k+1} = \frac{1}{2} Z_k (3I - Y_k^2) - \frac{1}{2} Y_k (Y_k Z_k + Z_k Y_k)) \tag{4.12}$$

where Y_0 and Z_0 are initialized as $A^{1/2}$ and $\frac{\partial L}{\partial Z}$ respectively, and Z_k converges to $2\frac{\partial L}{\partial A}$. The iterations involve one-step Newton iteration for matrix inverse, and hence the matrix X needs to be scaled to satisfy $||X - I||_2 < 1$ for the convergence. The iterations can be solved as fast as the automatic differentiation of the iterative matrix square-root without caching the intermediate outputs during the forward computation. One can check the stopping criterion to decide if the approximation converges for terminating the iterations. In practical, we found that five iterations are sufficient for the training.

4.2.3 Numerical stability and truncated SVD gradients

While the SVD can be used to compute arbitrary matrix functions in the forward step, computing the gradients using the Equation 4.2 is problematic when the matrix A has eigenvalues that are close to each other. This stems from the fact that SVD is ill-conditioned in this situation. Adding a small ϵ to the diagonal of A does not solve this problem either. In practice a truncated SVD gradient where the matrices K and Σ are set to zero for indices corresponding to eigenvalues that falls below a threshold τ works wells. However, even with the truncated SVD we found that the gradient computation results in numerical exceptions. Simply ignoring these cases worked well for fine-tuning when the learning rates were small but their impact on training networks from scratch remains unclear. Lyapunov gradients on the other hand are numerically stable because the inverse in Equation 4.9 depends on $1/\sigma_{\min}$ where $\sigma_{\min}$ is the smallest eigenvalue of the matrix $A^{1/2}$. The iterative solver is favorable as it only involves matrix-matrix multiplication and thus it is more efficient than the SVD-based approach.

4.2.4 Effect of network fine-tuning with matrix square-root normalization

We perform fine-tuning of the network using the matrix square-root layer in combination with elementwise square-root layer. Table 4.2 shows the results with fine-tuning B-CNNs with the VGG-M and VGG-D networks. For these experiments the gradients were computed using the Lyapunov technique described in Section 4.1.1. Training with SVD-based gradients led to a slightly worse performance, details of which are described in the next section. The matrix square-root normalization remains useful after fine-tuning and results in a **2-3%** improvement on average across the fine-grained datasets for both networks. The improvements are especially large for the VGG-M network.

Network	Normalization		Accuracy on dataset					
	$A^{1/2}$	$sgnsqrt(A)$	Birds		Cars		Aircrafts	
VGG-M		✓	72.0	78.1	77.8	86.5	74.7	81.3
	✓	✓	76.3	**81.3**	83.4	**88.5**	80.7	**84.0**
VGG-D		✓	80.1	84.0	82.9	90.6	77.7	86.9
	✓	✓	82.8	**85.8**	86.7	**92.0**	80.9	**88.5**

Table 4.2. For each dataset the accuracy before and after end-to-end fine-tuning of the networks are shown on the left and right column respectively. Matrix square-root normalization provides consistent improvements in accuracy over the baseline across all datasets.

4.2.5 Analysis on the precision of computations

Despite the improvement it offers a drawback of the matrix square-root is that computing the SVD is relatively slow and lacks batch-mode implementations. For the VGG-D network the computing the SVD of a 512x512 matrix takes about 22 milliseconds on a NVIDIA Titan X GPU, which is comparable to the rest of the network evaluation. Instead of computing the matrix square-root accurately using SVD, one can compute it approximately using a few iterations of the modified Denman-Beavers iterations described in Section 4.1.1. Table 4.3 shows that accuracy on the final classification task and time taken as a function of number of iterations. As few as 5 iterations are sufficient for matching the accuracy of the SVD method while being 5× faster. Surprisingly, even a *single* iteration provides non-trivial improvements over the baseline model (0 iterations) and takes less than 1 millisecond to evaluate. Although we didn't implement it, the method can be made faster using a batch-mode version of these iterations. With these iterative methods matrix-normalization layers are no longer the bottleneck in network evaluation.

Table 4.3 also shows the accuracy of fine-tuning with various gradient schemes for the matrix square-root. The time taken for backward computations (both LYAP

	Forward						Backward		
Iterations	0	1	5	10	20	SVD	LYAP	SVD	Faster
Birds	80.1	81.7	83.0	82.9	82.8	82.8	85.8	85.5	85.3
Cars	82.9	85.0	87.0	86.8	86.7	86.7	92.0	91.8	91.4
Aircrafts	77.7	79.5	81.3	81.1	80.9	80.9	88.5	87.8	86.8
Time	0	1ms	4ms	6ms	11ms	22ms	-	-	-

Table 4.3. On the left is the effect of number of Newton iterations for computing the matrix square-root on the speed and accuracy of the network. On the right is the accuracy obtained using various gradient computation techniques in the backward step. The VGG-D network is used for this comparison.

and SVD) are negligible given the SVD decomposition computed in the forward step and hence are not shown in the table. A faster scheme where the matrix square-root layer is ignored during fine-tuning is worse, but in most cases outperforms the fine-tuned baseline B-CNN model (Table 4.2). Although we found that SVD gradients are orders of magnitude less precise than Lyapunov gradients, the loss in accuracy after fine-tuning is negligible.

Our attempts at fine-tuning the network with matrix-logarithm using SVD-based gradients were not very successful, even with double precision arithmetic. On the other hand, all the experiments with matrix square-root were done with single precision arithmetic. This suggests that the numerical issues are partly due to the logarithm scaling of the eigenvalues.

4.3 Second-Order Democratic Aggregation

Although matrix function normalization provides a significant improvement on second-order features, it does not apply to combining with dimensionality reduction techniques mentioned in Section 3.4. In this section, we explore the technique called democratic aggregation [64, 97] that was proposed to reweight the first-order feature vectors prior to their aggregation [97] in order to balance their contributions to the final image representations. The aggregate is formulated as a linear combination of lo-

cal features, where the linear coefficients are solved efficiently by a modified Sinkhorn algorithm [71]. In this section, we study democratic aggregation in the context of second-order feature descriptors and show that this feature descriptor has favorable properties when combined with the democratic aggregator, which was applied originally to the first-order descriptors.

4.3.1 Democratic aggregation

Given a sequence of features $\mathcal{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$, where $\mathbf{x}_i \in \mathbb{R}^d$, a global descriptor $\xi(\mathcal{X})$ computes an *orderless aggregation* of the sequence. A common approach is to encode each feature using a non-linear function $\phi(\mathbf{x})$ before aggregation via a simple symmetric function such as sum or max. For example, the global descriptor using sum pooling can be written as:

$$\xi(\mathcal{X}) = \sum_{\mathbf{x} \in \mathcal{X}} \phi(\mathbf{x}). \tag{4.13}$$

The symmetric B-CNN representation uses outer-product encoders, *i.e.* $\phi(\mathbf{x}) = \text{vec}(\mathbf{x}\mathbf{x}^T)$, where $\mathbf{x}^T$ denotes the transpose and $\text{vec}(\cdot)$ is the vectorization operator. Thus, if $\mathbf{x}$ is d dimensional then $\phi(\mathbf{x})$ is d^2 dimensional.

The democratic aggregation approach was proposed in [97] to minimize interference or equalize contributions of each element in the sequence. The contribution of a feature is measured as the similarity of the feature to the overall descriptor. In the case of sum pooling, the contribution $C(\mathbf{x})$ of a feature $\mathbf{x}$ is given by:

$$C(\mathbf{x}) = \phi(\mathbf{x})^T \sum_{\mathbf{x}' \in \mathcal{X}} \phi(\mathbf{x}'). \tag{4.14}$$

For sum pooling, the contributions $C(\mathbf{x})$ may not be equal for all features $\mathbf{x}$. In particular, the contribution is affected by both the norm and frequency of the feature. Democratic aggregation is a scheme that weights each feature by a scalar $\alpha(\mathbf{x})$

that depends on both $\mathbf{x}$ and the overall set of features in $\mathcal{X}$ such that the weighted aggregation $\xi(\mathcal{X})$ satisfies:

$$\alpha(\mathbf{x})\phi(\mathbf{x})^T\xi(\mathcal{X}) = \alpha(\mathbf{x})\phi(\mathbf{x})^T \sum_{\mathbf{x}' \in \mathcal{X}} \alpha(\mathbf{x}')\phi(\mathbf{x}') = C, \ \forall \mathbf{x} \in \mathcal{X}, \tag{4.15}$$

under the constraint that $\forall \mathbf{x} \in \mathcal{X}$, $\alpha(\mathbf{x}) > 0$. The above equation only depends on the dot product between the elements since:

$$\alpha(\mathbf{x}) \sum_{\mathbf{x}' \in \mathcal{X}} \alpha(\mathbf{x}')\phi(\mathbf{x})^T\phi(\mathbf{x}') = \alpha(\mathbf{x}) \sum_{\mathbf{x}' \in \mathcal{X}} \alpha(\mathbf{x}')k(\mathbf{x}, \mathbf{x}'), \tag{4.16}$$

where $k(\mathbf{x}, \mathbf{x}')$ denotes the dot product between the two vectors $\phi(\mathbf{x})$ and $\phi(\mathbf{x}')$. Following the notation in [97], if we denote $\mathbf{K}_{\mathcal{X}}$ to be the kernel matrix of the set $\mathcal{X}$, the above constraint is equivalent to finding a vector of weights $\boldsymbol{\alpha}$ such that:

$$\texttt{diag}(\boldsymbol{\alpha})\mathbf{K}\texttt{diag}(\boldsymbol{\alpha})\mathbf{1}_n = C\mathbf{1}_n, \tag{4.17}$$

where `diag` is the diagonalization operator and $\mathbf{1}_n$ is an n dimensional vector of ones. In practice, the aggregated features $\xi(\mathcal{X})$ are ℓ_2 normalized hence the constant C does not matter and can be set to 1.

The authors [97] noted that the above equation can be efficiently solved by a dampened Sinkhorn algorithm [71]. The algorithm returns a unique solution as long as certain conditions are met, namely the entries in $\mathbf{K}$ are non-negative and the matrix is not fully decomposable. In practice, these conditions are not satisfied since the dot product between two features can be negative. A solution proposed in [97] is to compute $\boldsymbol{\alpha}$ by setting the negative entries in $\mathbf{K}$ to zero.

For completeness, the dampened Sinkhorn algorithm is included in Algorithm 1. Given n features of d dimensions, computing the kernel matrix takes $\mathcal{O}(n^2d)$, whereas each Sinkhorn iteration takes $\mathcal{O}(n^2)$ time. In practice, 10 iterations are sufficient to

Algorithm 1 Dampened Sinkhorn Algorithm

procedure SINKHORN($\mathbf{K}, \tau, \mathrm{T}$)
  $\boldsymbol{\alpha} \leftarrow \mathbf{1}_n$
  for $t = 1$ to T **do**
    $\boldsymbol{\sigma} = \mathtt{diag}(\boldsymbol{\alpha})\mathbf{K}\mathtt{diag}(\boldsymbol{\alpha})\mathbf{1}_n$
    $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha}/\boldsymbol{\sigma}^{\tau}$
  return $\boldsymbol{\alpha}$

find a good solution. The damping factor $\tau = 0.5$ is typically used. This slows the convergence rate but avoids oscillations and other numerical issues associated with the undampened version ($\tau = 1$).

4.3.2 γ-democratic aggregation.

We propose a parametrized family of democratic aggregation functions that interpolate between sum pooling and fully democratic pooling. Given a parameter $0 \leq \gamma \leq 1$, the γ-democratic aggregation is obtained by solving for a vector of weights $\boldsymbol{\alpha}$ such that:

$$\mathtt{diag}(\boldsymbol{\alpha})\mathbf{K}\mathtt{diag}(\boldsymbol{\alpha})\mathbf{1}_n = (\mathbf{K}\mathbf{1}_n)^{\gamma}. \tag{4.18}$$

When $\gamma = 0$, this corresponds to the democratic aggregation, and when $\gamma = 1$, this corresponds to sum aggregation since $\boldsymbol{\alpha} = \mathbf{1}_n$ satisfies the above equation. The above equation can be solved by modifying the update rule for computing σ in the Sinkhorn iterations to:

$$\sigma = \mathtt{diag}(\boldsymbol{\alpha})\mathbf{K}\mathtt{diag}(\boldsymbol{\alpha})\mathbf{1}_n/(\mathbf{K}\mathbf{1}_n)^{\gamma}, \tag{4.19}$$

in Algorithm 1, where / denotes element-wise division. Thus, the solution can be equally efficient for any value of γ. Intermediate values of γ allow the contributions $C(\mathbf{x})$ of each feature $\mathbf{x}$ within the set to vary and, in our experiments, we find this can lead to better results than the extremes (*i.e.*, $\gamma = 1$).

4.3.3 Democratic aggregation on second-order features

In practice, features extracted using deep convolutional neural networks can be high-dimensional. For example, an input image I is passed through layers of a convolutional neural networks to obtain a feature map $\mathbf{\Phi}(I)$ of size $W \times H \times D$. Here $d = D$ corresponds to the number of filters in the convolutional layer and $n = W \times H$ corresponds to the spatial resolution of the feature. For state-of-the-art CNNs from which features are typically extracted, the values of n and d are comparable and in the range of a few hundred to a thousand. Thus, explicitly realizing the outer products can be expensive. Below we show several properties of democratic aggregation with outer-product encoders. Some of these properties allow aggregation in a computationally and memory efficient manner.

Proposition 1. *For outer-product encoders, the solution to the γ-democratic kernels exists for all values of γ as long as $||\mathbf{x}|| > 0,\ \forall \mathbf{x} \in \mathcal{X}$.*

Proof. For the outer-product encoder we have:

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}') = \operatorname{vec}(\mathbf{x}\mathbf{x}^T)^T \operatorname{vec}(\mathbf{x}'\mathbf{x}'^T) = (\mathbf{x}^T\mathbf{x}')^2 \geq 0.$$

Thus, all the entries of the kernel matrix are non-negative and the kernel matrix is strictly positive definite when $||\mathbf{x}|| > 0,\ \forall \mathbf{x} \in \mathcal{X}$. This is a sufficient condition for the solution to exist [71]. Note that the kernel matrix of the outer product encoders is positive even when $\mathbf{x}^T\mathbf{x}' < 0$. □

Proposition 2. *For outer-product encoders, the solution $\boldsymbol{\alpha}$ to the γ-democratic kernels can be computed in $\mathcal{O}(n^2 d)$ time and $\mathcal{O}(n^2 + nd)$ space.*

Proof. The running time of the Sinkhorn algorithm is dominated by the time to compute the kernel matrix $\mathbf{K}$. Naively computing the kernel matrix for d^2 dimensional

features would take $\mathcal{O}(n^2d^2)$ time and $\mathcal{O}(n^2 + nd^2)$ space. However, since the kernel entries of the outer products are just the square of the kernel entries of the features before the encoding step, one can compute the kernel $\mathbf{K}$ by simply squaring the kernel of the raw features, which can be computed in $\mathcal{O}(n^2d)$ time and $\mathcal{O}(n^2 + nd)$ space. Thus the weights $\boldsymbol{\alpha}$ for the second-order features can also be computed in $\mathcal{O}(n^2d)$ time and $\mathcal{O}(n^2 + nd)$ space. □

Proposition 3. *For outer-product encoders, γ-democratic aggregation $\xi(\mathcal{X})$ can be computed with low-memory overhead using Tensor Sketching.*

Proof. Let θ be a low-dimensional embedding that approximates the inner product between two outer-products, *i.e.*,

$$\theta(\mathbf{x})^T\theta(\mathbf{x}') \sim \operatorname{vec}(\mathbf{x}\mathbf{x}^T)^T \operatorname{vec}(\mathbf{x}'\mathbf{x}'^T), \tag{4.20}$$

and $\theta(\mathbf{x}) \in \mathbb{R}^k$ with $k << d^2$. Since the γ-democratic aggregation of $\mathcal{X}$ is a linear combination of the outer-products, the overall feature $\xi(\mathcal{X})$ can be written as:

$$\xi(\mathcal{X}) = \sum_{\mathbf{x}\in\mathcal{X}} \alpha(\mathbf{x})\mathbf{x}\mathbf{x}^T \sim \sum_{\mathbf{x}\in\mathcal{X}} \alpha(\mathbf{x})\theta(\mathbf{x}). \tag{4.21}$$

□

Thus, instead of realizing the overall feature $\xi(\mathcal{X})$ of size d^2, one can use the embedding θ to obtain a feature of size k as a democratic aggregation of the approximate outer-products. One example of an approximate outer-product embedding is the Tensor Sketching (TS) approach of Pham and Pagh [103]. Tensor sketching has been used to approximate second-order sum pooling [43] resulting in an order-of-magnitude savings in space at a marginal loss in performance on classification tasks. Our experiments show that sketching also performs well in the context of democratic aggregation.

4.4 Democratic aggregation versus spectral normalization

4.4.1 Spectral normalization as equalizing contributions

In Section 4.1, we introduced the approach using matrix functions to normalize the second-order representations obtained by sum pooling. Let the matrix $\mathbf{A} = \sum_{\mathbf{x}\in\mathcal{X}} \mathbf{x}\mathbf{x}^T$. is normalized using matrix power function $\mathbf{A}^p$ with $0 < p < 1$. Matrix function can be computed using the Singular Value Decomposition (SVD). Given matrix $\mathbf{A}$ with a SVD given by $\mathbf{A} = \mathbf{U}\Lambda\mathbf{U}^T$, where the matrix $\Lambda = \texttt{diag}(\lambda_1, \lambda_2, ..., \lambda_d)$, with $\lambda_i \geq \lambda_{i+1}$, the matrix function f can be written as $\mathbf{Z} = f(\mathbf{A}) = \mathbf{U}g(\Lambda)\mathbf{U}^T$, where g is applied to the elements in the diagonal of Λ. Thus, the matrix power can be computed as $\mathbf{A}^p = \mathbf{U}\Lambda^p\mathbf{U}^T = \mathbf{U}\texttt{diag}(\lambda_1^p, \lambda_2^p, ..., \lambda_d^p)\mathbf{U}^T$. The following establishes a connection between the spectral normalization techniques and democratic pooling.

Let $\hat{\mathbf{A}}^p$ be the ℓ_2 normalized version of $\mathbf{A}^p$ and $r_{\max}$ and $r_{\min}$ be the maximum and minimum squared radii of the data $\mathbf{x} \in \mathcal{X}$ defined as:

$$r_{\max} = \max_{\mathbf{x}\in\mathcal{X}} ||\mathbf{x}||^2, \; r_{\min} = \min_{\mathbf{x}\in\mathcal{X}} ||\mathbf{x}||^2. \tag{4.22}$$

As earlier, let $C(\mathbf{x})$ be the contribution of the vector $\mathbf{x}$ to the aggregated representation defined as:

$$C(\mathbf{x}) = \text{vec}(\mathbf{x}\mathbf{x}^T)^T \, \text{vec}(\hat{\mathbf{A}}^p). \tag{4.23}$$

Proposition 4. *The following properties hold true:*

1. *The ℓ_2 norm of* $\text{vec}(\mathbf{A}^p)$ *is* $\rho(\mathbf{A}^p) = ||\,\text{vec}(\mathbf{A}^p)|| = \left(\sum_i \lambda_i^{2p}\right)^{1/2}$.

2. $\sum_{\mathbf{x}\in\mathcal{X}} C(\mathbf{x}) = \texttt{Trace}(\mathbf{A}^{1+p}/||\mathbf{A}^p||) = \left(\sum_i \lambda_i^{1+p}\right)/\rho(\mathbf{A}^p)$.

3. *The maximum value* $M = \max_{\mathbf{x}\in\mathcal{X}} C(\mathbf{x}) \leq r_{\max}\lambda_1^p/\rho(\mathbf{A}^p)$.

4. *The minimum value* $m = \min_{\mathbf{x}\in\mathcal{X}} C(\mathbf{x}) \geq r_{\min}\lambda_d^p/\rho(\mathbf{A}^p)$.

Proof. The proof is left in Appendix. □

Proposition 5. *The variance σ^2 of the contributions $C(\mathbf{x})$ satisfies*

$$\sigma^2 \leq (M-\mu)(\mu-m) \leq \frac{(M-m)^2}{4} \leq \frac{r_{\max}^2 \lambda_1^{2p}}{4\rho(\mathbf{A}^p)^2}, \tag{4.24}$$

where M and m are the maximum and minimum values defined above and μ is the mean of $C(\mathbf{x})$ given by $\sum_{\mathbf{x}\in\mathcal{X}} C(\mathbf{x})/n$ where n is the cardinality of $\mathcal{X}$. All of the above quantities can be computed from the spectrum of the matrix $\mathbf{A}$.

Proof. The proof can be obtained by a straightforward application of Popoviciu's inequality on variances [104] and a tighter variant by Bhatia and Davis [9]. The last inequality is obtained by setting $m = 0$. □

The above shows that smaller values p reduce an upper-bound on the variance of the contributions thereby equalizing their contributions. The upper bound is a monotonic function of the exponent p and is minimized when $p = 0$ reducing all the spectrum to an identity matrix. This corresponds to whitening of the matrix $\mathbf{A}$. However, complete whitening often leads to poor results while intermediate values such as $p = 1/2$ can be significantly better than $p = 1$

Proposition 6. *For exponents $0 < p < 1$, the matrix power $\mathbf{A}^p$ may not lie in the linear span of the outer-products of the features $\mathbf{x} \in \mathcal{X}$.*

The proof of Proposition 6 is left in Appendix. A consequence of this is that the matrix power cannot be easily computed in the low-dimensional embedding space of outer-products encoding such as Tensor Sketch. It does however lie in the linear span of the outer-products of the eigenvectors. However, computing eigenvectors can be significantly slower than computing weighted aggregates. We describe the computation and memory trade-offs between computing the matrix powers and democratic pooling in Section 4.4.5.

4.4.2 The distribution of the spectrum and feature contributions

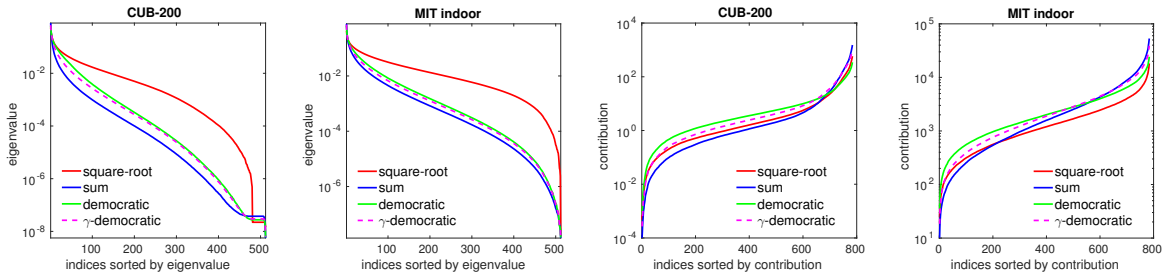
In this section, we analyze how democratic pooling and matrix normalization effect the spectrum (set of eigenvalues) of the aggregated representation, as well as how the contributions of individual features are distributed as a function of γ for the democratic pooling and p of the matrix power normalization.

We randomly sampled 50 images from CUB and MIT indoor datasets each and plotted the spectrum (normalized to unit length) and the feature vector contributions $C(\mathbf{x})$ (Eq. (4.23)) in Figure 4.4. In this experiment, we use the matrix power $p = 0.5$ and $\gamma = 0.5$. Figure 4.4(a) shows that the square root yields a flatter spectrum in comparison to the sum aggregation. Democratic aggregation distributes the energy away from the top eigenvalues but has considerably sharper spectrum in comparison to the square root. The γ-democratic pooling interpolates between sum and fully democratic pooling.

Figure 4.4(b) shows the contributions of each feature $\mathbf{x}$ to the aggregate for different pooling techniques (Eq. (4.23)). The contributions are more evenly distributed for the matrix square root in comparison to sum pooling. Democratic pooling flattens the individual contributions the most – we note that it is explicitly designed to have this effect. These two plots show that democratic aggregation and power normalization both achieve equalization of feature contributions.

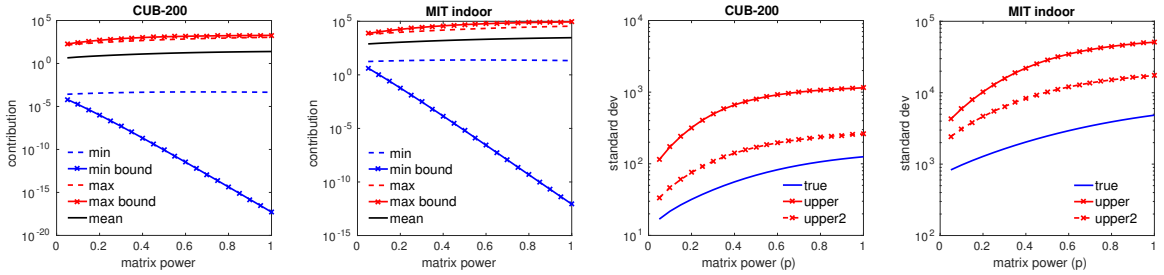
Figure 4.5 shows the variances of the contributions $C(\mathbf{x})$ to the aggregation $\hat{\mathbf{A}}^p$ using the VGG-16 features for different values of the exponent p. Figure 4.5(a) shows the true minimum, maximum, mean as well as the bounds of these quantities expressed in Proposition 4. The upper bound on the maximum contribution, *i.e.*, $r_{\max}\lambda_1^p/\rho(\mathbf{A}^p)$, is tight on both datasets, as can be seen in the overlapping red lines, while the lower bound is significantly less tight.

Figure 4.5(b) shows the true deviation and two different upper bounds on the variance of the contributions as expressed in Proposition 5 and Eq. (4.24). The tighter



(a) spectrum (eigenvalues) (b) contributions $C(\mathbf{x})$

Figure 4.4. (a) The spectrum (eigenvalues) for various feature aggregators on CUB-200 and MIT indoor datasets. (b) The individual feature vector contributions $C(\mathbf{x})$.



(a) bounds on contribution (b) bounds on variance

Figure 4.5. (a) The upper (red solid) and lower bounds (blue solid) on the contributions to the set similarity versus the exponent of matrix power normalization on Birds and MIT indoor datasets. Maximum and minimum values are shown in dashed lines and the the mean is shown in black solid lines. (b) The upper bounds to the variance of feature contributions $C(\mathbf{x})$.

bound shown by the dashed red line corresponds to the version with the mean μ in Eq. (4.24). The plot shows that the matrix power normalization implicitly reduces the variance in feature contributions similar to equalizing the feature vector contributions $C(\mathbf{x})$ in democratic aggregation. These plots are averaged over 50 examples from the CUB-200 and MIT indoor datasets.

Dataset	γ-democratic			$\mathbf{A}^p$
	Democratic γ=0	Optimal γ	Sum $\gamma = 1$	
Caltech UCSD Birds	84.7	84.9 (0.5)	84.0	85.9 (0.3)
Stanford Cars	89.7	90.8 (0.5)	90.6	91.7 (0.5)
FGVC Aircrafts	86.7	86.7 (0.0)	85.7	87.6 (0.3)
DTD	72.2	72.3 (0.3)	71.2	72.9 (0.6)
FMD	82.8	84.8 (0.8)	84.6	85.0 (0.7)
MIT indoor	79.6	80.4 (0.3)	79.5	80.9 (0.6)

Table 4.4. The accuracy of aggregating second-order features w.r.t. various aggregators using fine-tuned VGG-16 on fine-grained recognition (top) and using ImageNet pretrained VGG-16 on other (bottom) datasets. From left to right, we vary γ values and compare democratic pooling, γ-democratic pooling and average pooling with the matrix power aggregation. The optimal values of γ and p are indicated in parentheses.

4.4.3 Experiments comparing spectral normalization and democratic aggregation

Table 4.4 shows the performance as a function of γ for the γ-democratic pooling and p for the matrix normalization on the VGG-16 network. For DTD dataset, we report results on the first split. For FMD dataset, we randomly sample half of the data in each category for training and use the rest for testing. We use the standard training and testing splits on remaining datasets. We augment the training set by flipping its images and train k one-vs-all linear SVM classifiers with hyperparameter $C = 1$. At the test time, we average predictions from an image and its flipped copy. Optimal γ and the matrix power p are also reported.

The results on sum pooling correspond to the symmetric BCNN models. Fully democratic pooling (γ=0) improves the performance over sum pooling by 0.7-1%. However, equalizing feature contributions hurts performance on Stanford Cars and FMD dataset. Table 4.4 shows that reducing the contributions by adjusting $0 < \gamma < 1$ helps outperform sum pooling and fully democratic pooling. Matrix power normalization outperforms γ-democratic pooling by 0.2-1%.

4.4.4 Democratic Aggregation with Tensor Sketch

One of the main advantages of the democratic pooling approaches over matrix power normalization techniques is that the embeddings can be computed in a low-dimensional space using tensor sketching. To demonstrate this advantage, we compute the second-order democratic pooling combined with tensor sketching on 2048 dimensional ResNet-101 features. Direct construction of second-order features yields ∼4M dimensional features which are impractical to manipulate on GPU/CPU. Therefore, we apply the Tensor Sketch [103] to approximate the outer product using 8192 dimensional features, which is far lower than 2048^2 of the full outer product. The features are aggregated using γ-democratic approach with $\gamma = 0.5$.

Table 4.5 reports the accuracy on MIT indoor. The baseline model approximating second-order features with tensor sketch followed by sum pooling achieves 82.8% accuracy. With democratic pooling, our model achieves state-of-the-art accuracy of 84.3% which is 1.5% more than the baseline. Moreover, Table 4.4 shows that we outperform the matrix power normalization using VGG-16 network by 3.4%. Note that (i) matrix power normalization is impractical for ResNet101 features, (ii) it cannot be computed by sketching due to Proposition 6. We also outperform FASON [28] by 2.6%. FASON fuses the first- and second-order features from *conv*4_4 and *conv*5_4 layers of the VGG-19 networks given 448×448 image size and scores 81.7% accuracy. Recent work on Spectral Features [69] achieves the same accuracy as our best model with democratic pooling. However, approach [69] uses more data augmentations (rotation, shifts, *etc.*) during training and pretrains the VGG-19 network on the large-scale Places205 dataset. In contrast, our networks are pretrained on ImageNet which arguably has a larger domain shift from the MIT indoor dataset than Places205.

Method		accuracy
Places-205	[125]	80.9
Deep Filter Banks	[23]	81.0
Spectral Features	[69]	84.3
FASON	[28]	81.7
ResNet101 + TS + sum pooling	*(baseline)*	82.8
ResNet101 + TS + γ-democratic	*(ours)*	**84.3**

Table 4.5. Evaluations and comparisons to the state-of-the-art on MIT indoor dataset.

4.4.5 Discussion on efficiency

While matrix power normalization achieves marginally better performance, it requires SVD which is computationally expensive and not GPU friendly *e.g.*, the CUDA BLAS cannot perform SVD for large matrices. Even in the case of matrix square root, which can be approximated via Newton's iterations, the iterations involve matrix-matrix multiplication of $\mathcal{O}(n^3)$ complexity. In contrast, solving democratic pooling via the Sinkhorn algorithm (Algorithm 1) involves only matrix-vector multiplication, which is $\mathcal{O}(n^2)$. Empirically, we find that solving Sinkhorn iterations is an order of magnitude faster than solving the matrix square root on an NVIDIA Titan X GPU. Moreover, the complexity of Sinkhorn iteration depends only on the kernel matrix – it is independent of the feature vector size. In contrast, the memory required by a covariance matrix grows with $\mathcal{O}(n^2)$, which becomes prohibitive for feature vectors greater than 512 dimensions. Second-order democratic pooling with tensor sketching yields comparable results and reduces the memory usage by two orders of magnitude over the matrix power normalization.

4.5 Bilinear models with state-of-the-art CNNs

Bilinear pooling has been shown to be effective in aggregating the higher-order statistics on convolutional features obtained by VGG networks and improves the recognition performance over the vanilla VGG models by a significant margin. Since

we published bilinear CNNs in 2015, several techniques have been proposed to stabilize the training of convolutional neural networks and allowed the training of deeper architectures with higher model capacity. For example, batch normalization [59] reduced the internal covariate shift to regularize network training; ResNet [54] introduced residual connections to facilitate the gradient propagation toward bottom layers; DenseNet [58] densely connect convolutional layers to reuse the features obtained from previous layers. With multiple interconnections between layers, feature activations from deeper layers may implicitly model the higher-order feature statistics. To verify if state-of-the-art CNN models can still benefit from bilinear pooling, we compare the bilinear CNNs with DenseNet against vanilla DenseNet, which aggregate the convolutional features with average pooling, on CUB, Cars, FGVC-Aircrafts, and iNaturalist [2] datasets. We split the iNaturalist dataset into several classification tasks according to the super-category labels. The details for each task are shown in Table 4.6. For both methods, we start with ImageNet pretrained DenseNet-201 with input size 448 x 448 and aggregate the ReLU feature activations from the penultimate layers and fine-tune the representations on the target datasets.

The features dimension for the DenseNet penultimate layer is 1,092 for which it is too big to compute the full outer product. We reduce the feature dimension to 128 by adding a 1x1 convolutional layers followed by the outer-product encoding. The weights for the dimensionality reduction layer are initialized randomly and fine-tuned end-to-end. The bilinear features are sequentially normalized by matrix square-root, element-wise signed square-root and, ℓ_2 normalization. The normalization with matrix square-root is computed via Newton-Schultz iterations, and its gradients are computed via the iterative solver for the Lyapunov equation. We set the number of iterations to 5 for both forward and backward computations.

Table 4.6 shows the results with DenseNet models. Comparing to the results in Table 4.2, vanilla DenseNet with global average pooling outperforms B-CNN with

Dataset	#classes	#images	DenseNet	
			Average pooling	Bilinear pooling
Caltech UCSD Birds	200	5,994	84.9	**87.5**
Stanford Cars	196	8,144	**93.2**	92.9
FGVC Aircrafts	100	6,667	**92.3**	90.6
Plantae	2,917	118,800	**69.6**	66.9
Animalia	178	5,996	**80.5**	79.0
Reptilia	284	22,754	**53.3**	51.6
Amphibia	144	11,156	55.6	**56.9**
Aves	1,258	143,950	60.1	**62.0**
Mollusca	262	8,007	73.5	**75.1**
Fungi	321	6,864	70.5	**72.8**
Mammalia	234	20,104	60.1	**63.3**
Archanida	114	4,037	66.4	**71.3**
Insecta	2,031	87,192	78.2	**82.0**
Actinopterygii	369	7,835	76.5	**82.7**
iNaturalist mean acc.	-	-	67.7	**69.4**

Table 4.6. The accuracy of bilinear pooling against average pooling using DenseNet features. The first column lists the datasets along with the number of classes and images in the second and third columns. The bottom of the table shows the results for the tasks created from the dataset obtained from iNaturalist 2018 fine-grained challenges. Mean per-task accuracies for the iNaturalist dataset are reported in the last row.

VGG models on Cars (93.2 vs. 92.0) and Aircraft (92.3 vs. 88.5) while achieving comparable results on CUB (84.9 vs. 85.8). This demonstrates that constructing deeper neural networks is effective in improving recognition accuracy for fine-grained tasks. Aggregating DenseNet features with bilinear pooling further improves the accuracy on most datasets, while the gaps become smaller comparing to the improvement obtained on VGG models. The bilinear pooling achieves lower performance on Cars, Aircrafts, Plantae, Animalia, and Reptilia datasets. There are multiple confounding factors such as domain shifts from ImageNet, size of training data, and the distribution of size and location of objects, which could complicate the training. Deeper neural network architectures could have sufficient model capacity to learn a better

representation and implicitly capture higher-order feature correlations. Nevertheless, aggregating state-of-the-art CNN features via second-order pooling is still effective. However, more controlled experiments are needed to to understand the cases where the higher-order features are more discriminative than their first-order counterparts.

4.6 Summary

In this chapter, we present two techniques to normalize the aggregate of second-order features according to feature frequency. Matrix-normalization offers complementary benefits to elementwise normalization layers and leads consistent improvements in accuracy over the vanilla B-CNN models. The matrix square-root normalization outperforms the matrix logarithm normalization when combined with elementwise square-root normalization for most of our experiments. Moreover, the matrix square-root can be computed efficiently on a GPU using a few Newton iterations and allows accurate gradient computations via a Lyapunov equation. The second-order aggregation method referred to as γ-democratic pooling that interpolates between sum (γ=1) and democratic pooling (γ=0) and outperforms both extremes. We demonstrated that γ-democratic pooling enjoys low computational complexity compared to the matrix square root approximations via Newton's iterations; but with a smaller improvement compared to using matrix square root. This leads to different design choices based on the given computational budget.

CHAPTER 5

VISUALIZING DEEP TEXTURE REPRESENTATIONS

In the previous chapter, we have introduced bilinear CNN models and their connections to classical texture-based image representations and showed their effectiveness on image recognition. In this chapter, we will focus on the techniques using the representations to synthesize novel images. We present a parametric image synthesis framework that jointly optimizes the following objectives given source images and target attributes: 1) reconstruction of source B-CNN representations, 2) reconstruction of source CNN representations, and 3) maximizing the prediction score of target attributes. The framework is a generalization of the related works in texture synthesis [44], style transfer [45], and inversion of deep representations [94, 34]. By generating the novel images by reconstructing B-CNN representations, we are able to visualize the invariance of the representations. We visualize the pre-images of the categories from fine-grained and texture recognition benchmarks based on B-CNN representations to understand the properties of the given categories.

5.1 Methodology

We describe our framework for parametric texture synthesis, inversion, and attribute-based manipulation using CNNs. For an image $\mathcal{I}$ one can compute the activations of the CNN at a given layer r_i to obtain a set of features $F_{r_i} = \{f_j\}$ indexed by their location j. The bilinear feature $B_{r_i}(\mathcal{I})$ of F_{r_i} is obtained by computing the

outer product of each feature f_j with itself and aggregating them across locations by averaging, *i.e.*,

$$B_{r_i}(\mathcal{I}) = \frac{1}{N}\sum_{j=1}^{N} f_j f_j^T. \tag{5.1}$$

This is equivalent to Gram matrix representation, the terminology used in [44, 45].

Let $r_i, i = 1, \ldots, n$, be the index of the i^{th} layer with the bilinear feature representation B_{r_i}. Gatys *et al.* [44] propose a method for texture synthesis from an input image $\mathcal{I}$ by obtaining an image $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ that matches the bilinear features at various layers by solving the following optimization:

$$\min_{\mathbf{x}} \sum_{i=1}^{n} \alpha_i L_1 \left(B_{r_i}, \hat{B}_{r_i} \right) + \gamma \Gamma(\mathbf{x}). \tag{5.2}$$

Here, $\hat{B}_{r_i} = B_{r_i}(\mathcal{I})$, α_i is the weight of the i^{th} layer, $\Gamma(\mathbf{x})$ is a natural image prior such as the total variation norm (TV norm), and γ is the weight on the prior. Note that we have dropped the implicit dependence of B_{r_i} on $\mathbf{x}$ for brevity. Using the squared loss-function $L_1(x, y) = \sum (x_i - y_i)^2$ and starting from a random image where each pixel initialized with a *i.i.d* zero mean Gaussian, a local optimum is reached through gradient descent. The authors employ L-BFGS, but any other optimization method can be used (*e.g.*, Mahendran and Vedaldi [94] use stochastic gradient descent with momentum).

Prior work on minimizing the reconstruction error with respect to the "un-pooled" features F_{r_i} has shown that the *content* of the image in terms of the color and spatial structure is well-preserved even in the higher convolutional layers. Recently, Gatys *et al.*in a separate work [45] synthesize images that match the style and content of two different images $\mathcal{I}$ and $\mathcal{I}'$ respectively by minimizing a weighted sum of the texture and content reconstruction errors:

$$\min_{\mathbf{x}} \lambda L_1 \left(F_s, \hat{F}_s \right) + \sum_{i=1}^{n} \alpha_i L_1 \left(B_{r_i}, \hat{B}_{r_i} \right) + \gamma \Gamma(\mathbf{x}). \tag{5.3}$$

Here $\hat{F}_s = F_s(\mathcal{I}')$ are features from a layer s from which the target content features are computed for an image $\mathcal{I}'$.

The bilinear features can also be used for predicting attributes by first normalizing the features (signed square-root and ℓ_2) and training a linear classifier in a supervised manner as described in Chapter 2. Let $l_i : i = 1, \ldots, m$ be the index of the i^{th} layer from which we obtain attribute prediction probabilities C_{l_i}. The prediction layers may be different from those used for texture synthesis. Given a target attribute $\hat{C}$ we can obtain an image that matches the target label and is similar to the texture of a given image $\mathcal{I}$ by solving the following optimization:

$$\min_{\mathbf{x}} \sum_{i=1}^{n} \alpha_i L_1 \left(B_{r_i}, \hat{B}_{r_i} \right) + \beta \sum_{i=1}^{m} L_2 \left(C_{l_i}, \hat{C} \right) + \gamma \Gamma(\mathbf{x}). \tag{5.4}$$

Here, L_2 is a loss function such as the *negative log-likelihood* of the label $\hat{C}$ and β is a tradeoff parameter. If multiple targets $\hat{C}_j$ are available then the losses can be blended with weights β_j resulting in the following optimization:

$$\min_{\mathbf{x}} \sum_{i=1}^{n} \alpha_i L_1 \left(B_{r_i}, \hat{B}_{r_i} \right) + \beta_j \sum_{i=1,j}^{m} L_2 \left(C_{l_i}, \hat{C}_j \right) + \gamma \Gamma(\mathbf{x}). \tag{5.5}$$

Synthesized images are obtained by solving the optimization. Using this framework we: (i) study the role of initialization in the convergence for texture synthesis, (ii) investigate the nature of invariances of these features by visualizing the images obtained by reconstructing texture representations (iii) provide insights into the learned models by inverting them, and (iv) show results for modifying the content of an image with texture attributes.

Experiments setting We use the 16-layer VGG network [112] trained on ImageNet for all our experiments. For the image prior $\Gamma(\mathbf{x})$ we use the TV_β norm with $\beta = 2$:

$$\Gamma(\mathbf{x}) = \sum_{i,j} \left((x_{i,j+1} - x_{i,j})^2 + (x_{i+1,j} - x_{i,j})^2 \right)^{\frac{\beta}{2}}. \tag{5.6}$$

The exponent $\beta = 2$ was empirically found to lead to better reconstructions in [**?**] as it leads to fewer "spike" artifacts than $\beta = 1$. In all our experiments, given an input image we resize it to 224×224 pixels before computing the target bilinear features and solve for $\mathbf{x} \in \mathbb{R}^{224\times224\times3}$. This is primarily for speed since the size of the bilinear features are independent of the size of the image. Hence, an output of any size can be obtained by minimizing Eqn. 5.5. We use L-BFGS for optimization and compute the gradients of the objective with respect to $\mathbf{x}$ using back-propagation. One detail we found to be critical for good reconstructions is that we ℓ_1 normalize the gradients with respect to each of the L_1 loss terms to balance the losses during optimization. Mahendran and Vedaldi [94] suggest normalizing each L_1 loss term by the ℓ_2 norm of the target feature $\hat{B}_{r_i}$. Without some form of normalization the losses from different layers are of vastly different scales leading to numerical stability issues during optimization.

5.2 Texture synthesis and image style transfer

Gatys *et al.* proposed the optimization-based texture synthesis approach formulated as equation 5.3. Given a source texture image. we extract the bilinear feature representations $\hat{B}_{r_i}$. To synthesize a new texture image, we set the weight λ to 0 ignoring the reconstruction of image content. Same approach is used for image style transfer where given a source style image and the a content image, we extract the bilinear features for the style image and the un-pooled CNN features for the content image. The optimization aims to find the output image such that the corresponding content and style are preserved. Although the approach works well, it is not practical since it requires several hundreds of CNN evaluations, which takes several minutes on a high-end GPU. In this thesis, we explore non-parametric patch-based approaches as initialization to speed up the process. We note that more efficient texture synthesis frameworks were proposed in recent work [121, 120, 65]. They further avoid

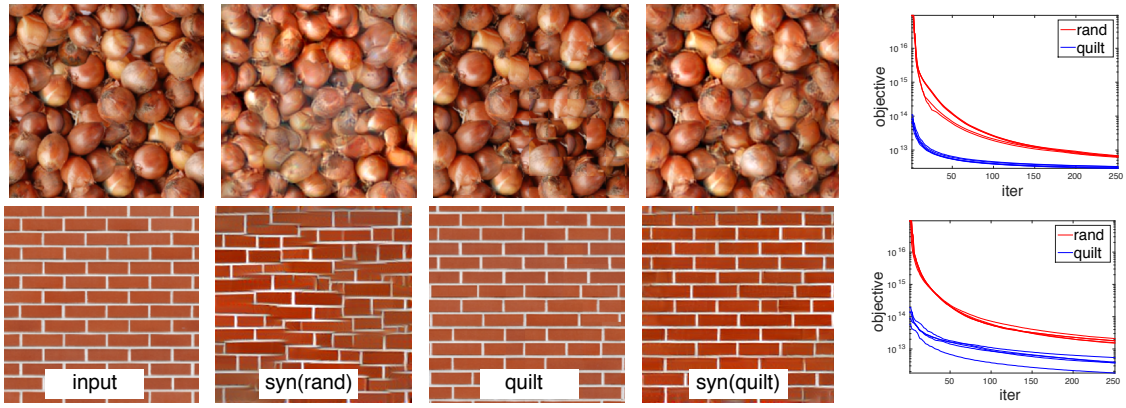


Figure 5.1. Effect of initialization on texture synthesis. Given an input image, the solution reached by the L-BFGS after 250 iterations starting from a random image: *syn(rand)*, and image quilting: *syn(quilt)*. The results using image quilting [35] are shown as *quilt*. On the right is the objective function for the optimization for 5 random initializations. Quilting-based initialization starts at a lower objective value and matches the final objective of the random initialization in far fewer iterations. Moreover, many artifacts of quilting are removed in the final solution (*e.g.*, the top row). *Best viewed with digital zoom.* Images are obtained from `http://www.textures.com`.

the optimization process and synthesize new texture images by feed-forward neural networks. This can be achieved by training neural networks with perceptual loss and instance normalization. We refer readers to their papers for details.

In comparison, non-parametric patch-based approaches such as *image quilting* [35] are orders of magnitude faster than optimization-based method. Quilting introduces artifacts when adjacent patches do not align with each other. The original paper proposed an approach where a one-dimensional cut is found that minimizes artifacts. However, this can fail since local adjustments cannot remove large structural errors in the synthesis. We instead investigate the use of quilting to initialize the gradient-based synthesis approach.

Fig. 5.1 shows the objective through iterations of L-BFGS starting from a random and quilting-based initialization. Quilting starts at a lower objective and reaches the final objective of the random initialization significantly faster. Moreover, the

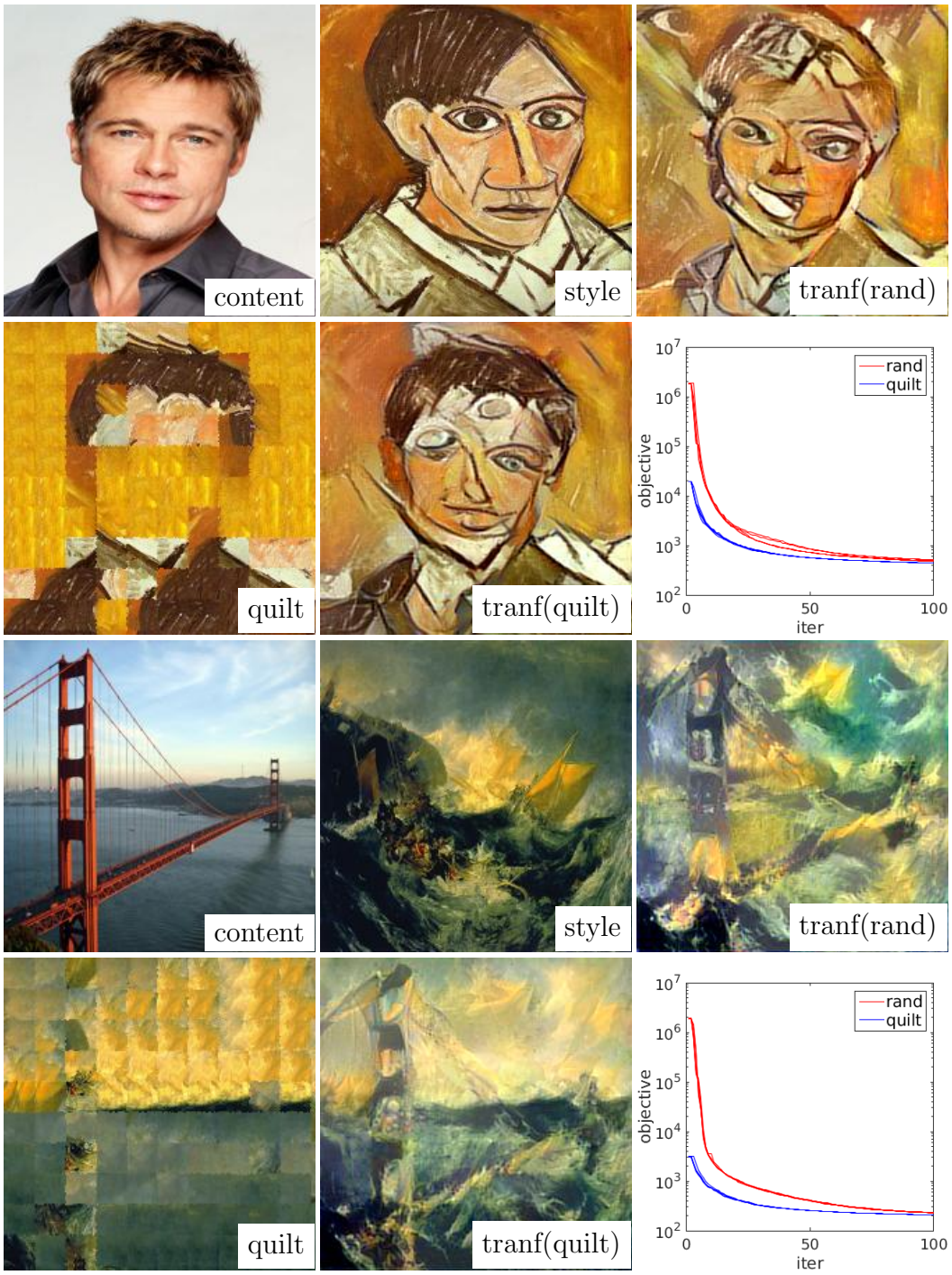


Figure 5.2. Effect of initialization on style transfer. Given a content and a style image the style transfer reached using L-BFGS after 100 iterations starting from a random image: *tranf(rand)*, and image quilting: *tranf(quilt)*. The results using image quilting [35] are shown as *quilt*. On the bottom right is the objective function for the optimization for 5 random initializations.

global adjustments of the image through gradient descent remove many artifacts that quilting introduces (digitally zoom in to the *onion image* to see this). Fig. 5.2 show the results using image quilting as initialization for style transfer [45]. Here two images are given as input, one for content measured as the *conv4_2* layer output, and one for style measured as the bilinear features. Similar to texture synthesis, the quilting-based initialization starts from lower objective value and the optimization converges faster. These experiments suggest that patch-based and parametric approaches for texture synthesis are complementary and can be combined effectively.

5.3 Visualizing invariances

In this section we aim to understand B-CNN texture representation by synthesizing *invariant images*, i.e. images that are nearly identical to a given image according to the bilinear features, and *inverse images* for a given category. Inverse images visualize the images that maximize the probability of the target categories. This help us gain the insights on the properties of categories that are captured by the recognition models. In addition to visualizing inverse images, we also visualize the learned convolutional filters by showing the top activating patches.

5.3.1 Visualizing invariant images for objects

We use *relu1_1, relu2_1, relu3_1, relu4_1, relu5_1* layers for texture representation. Fig. 5.3 shows several invariant images to the image on the top left, *i.e.* these images are virtually identical as far as the bilinear features for these layers are concerned. Translational invariance manifests as shuffling of patches but important local structure is preserved within the images. These images were obtained using $\gamma = 1e-6$ and $\alpha_i = 1\ \forall i$ in Eqn. 5.5. We found that as long as some higher and lower layers are used together the synthesized textures look reasonable, similar to the observations of Gatys *et al.*.

Figure 5.3. Invariant inputs. These six images are virtually identical when compared using the bilinear features of layers *relu1_1, relu2_1, relu3_1, relu4_1, relu5_1* of the VGG network [112].

5.3.2 Visualizing texture categories

We learn linear classifiers to predict categories using bilinear features from *relu2_2, relu3_3, relu4_3, relu5_3* layers of the CNN on various datasets and visualize images that produce high prediction scores for each class. We achieve this by solving the optimization described in equation 5.4 with α_i equal to 0. Fig. 5.4 shows some example inverse images for various categories for the DTD, FMD and MIT indoor datasets. These images were obtained by setting $\beta = 100$, $\gamma = 1e-6$, and $\hat{C}$ to various class labels in Eqn. 5.5. These images reveal how the model represents texture and scene categories. For instance, the *dotted* category of DTD contains images of various colors and dot sizes and the inverse image is composed of multi-scale multi-colored dots. The inverse images of *water* and *wood* from FMD are highly representative of these

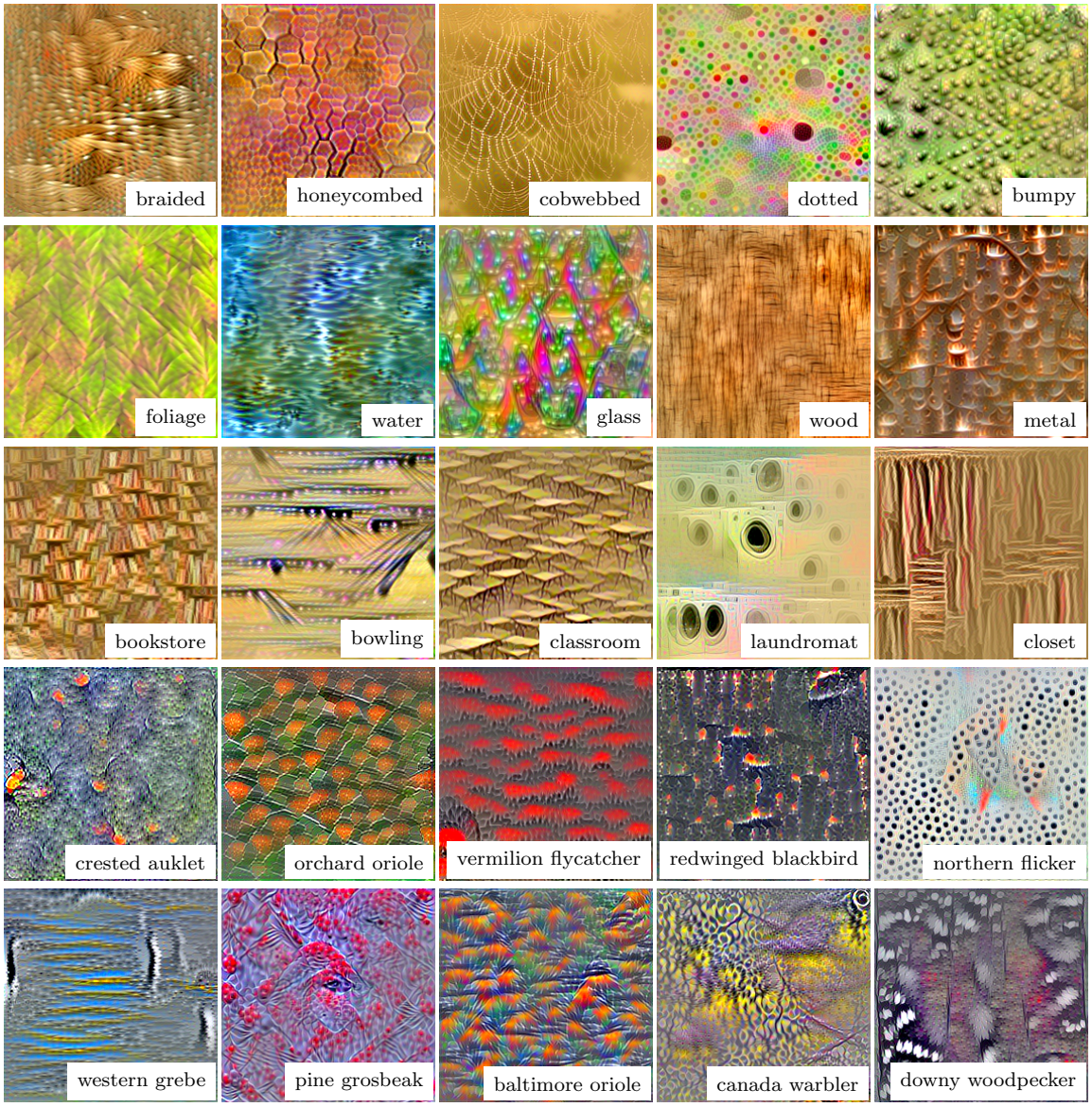


Figure 5.4. Visualizing various categories by inverting the B-CNN based on VGG-D network trained on DTD [22], FMD [110], MIT Indoor dataset [105] (first three rows from top to bottom), and the CUB dataset [124] (last two rows).

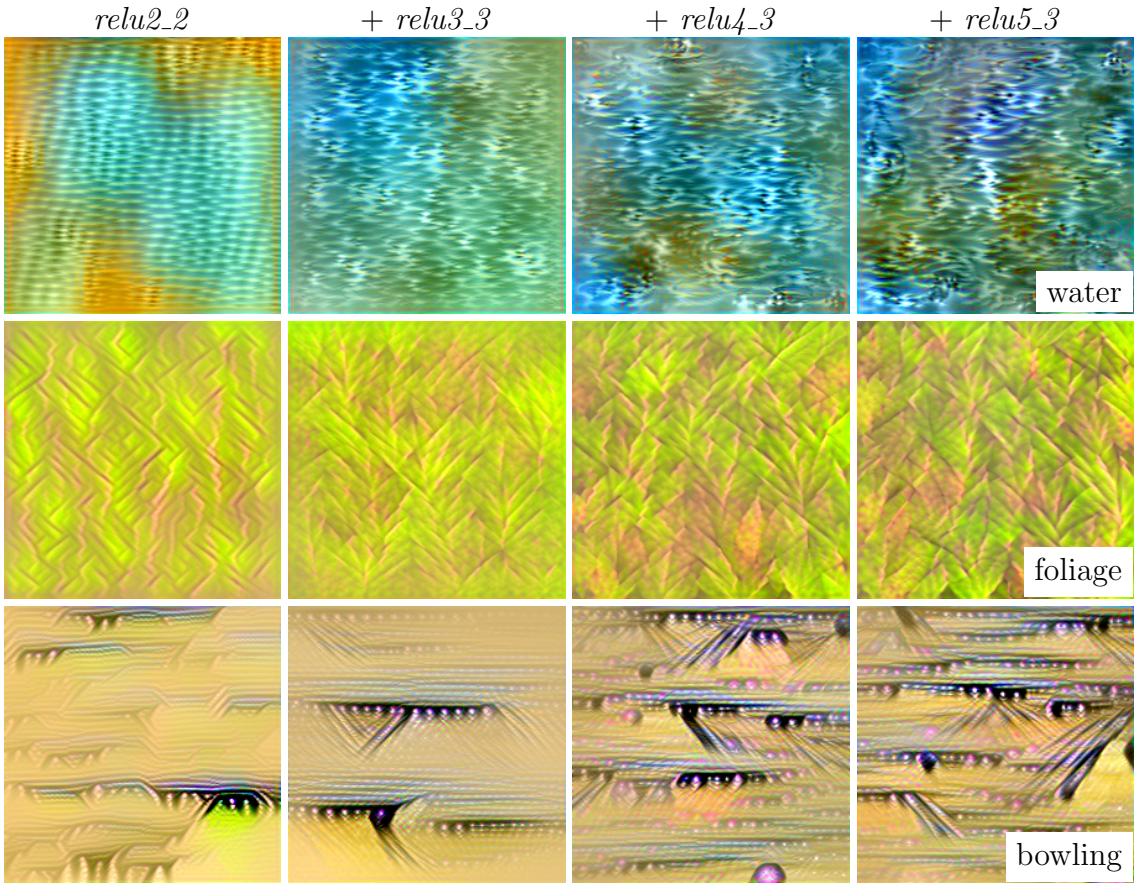


Figure 5.5. Effect of layers on inversion. Pre-images obtained by inverting class labels using different layers. The leftmost column shows inverses using predictions of *relu2_2* only. In the following columns we add layers *relu3_3*, *relu4_3*, and *relu5_3* one by one.

categories. Note that these images cannot be obtained by simply averaging instances within a category which is likely to produce a blurry image. The orderless nature of the texture descriptor is essential to produce such sharp images. The inverse scene images from the MIT indoor dataset reveal key properties that the model learns – a *bookstore* is visualized as racks of books while a *laundromat* has laundry machines at various scales and locations. In Fig. 5.5 we visualize reconstructions by incrementally adding layers in the texture representation. Lower layers preserve color and small-scale structure and combining all the layers leads to better reconstructions. Even though the *relu5_3* layer provides the best recognition accuracy, simply using that

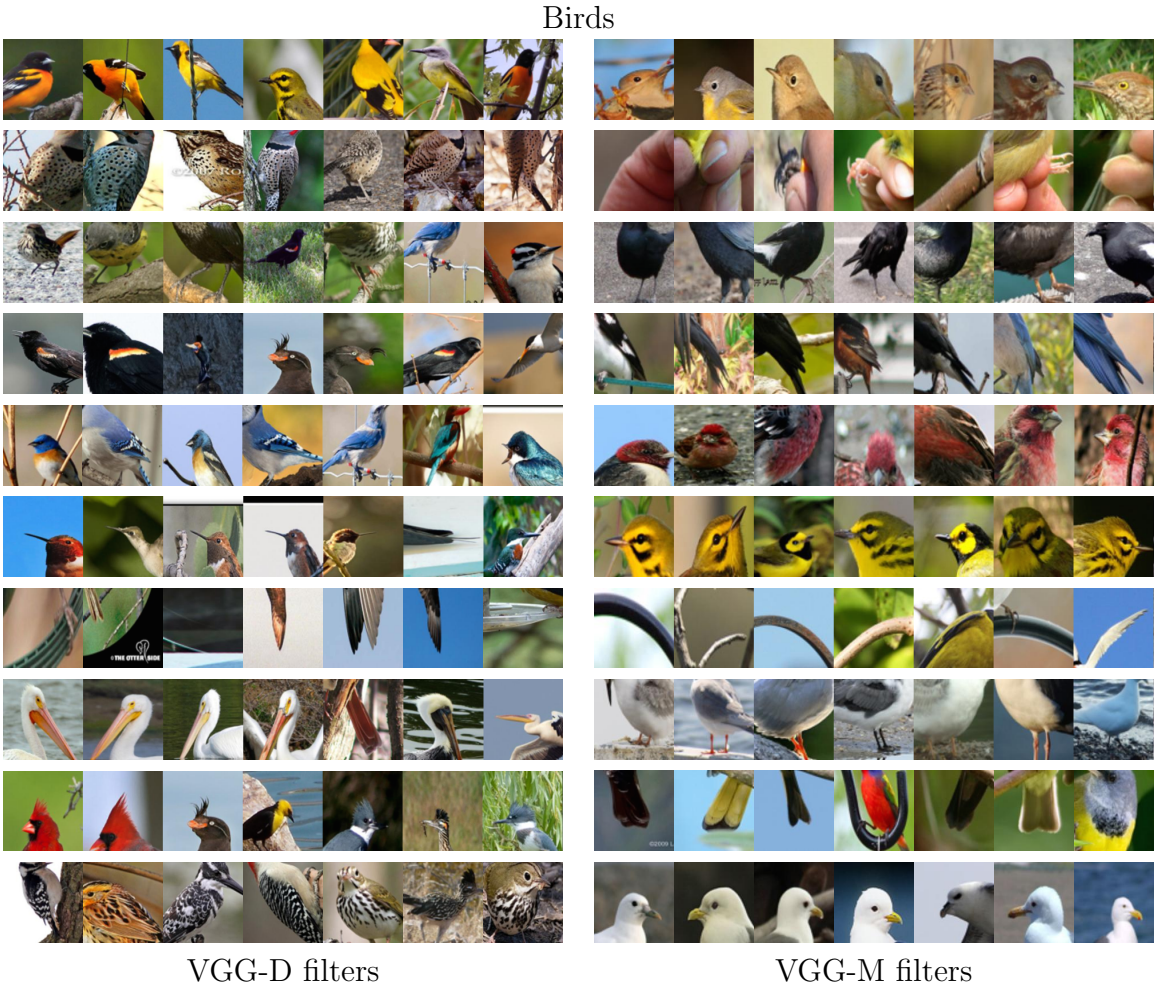


Figure 5.6. Patches with the highest activation for several filters of the fine-tuned B-CNN (VGG-D + VGG-M) model on birds dataset. These visualizations indicate that network units activate on highly localized attributes of objects that capture color, texture, and shape patterns.

layer did not produce good inverse images (not shown). Notably, color information is discarded in the upper layers. Fig. 5.4 shows visualizations of some other categories across datasets.

5.3.3 Retrieving top activations of neural units

The inverse images visualize the categories as texture images and show the invariances of the categories that are captured by the classification models. In this

Figure 5.7. Patches with the highest activation for several filters of the fine-tuned B-CNN (VGG-D + VGG-M) model on aircrafts dataset. These visualizations indicate that network units activate on highly localized attributes of objects that capture color, texture, and shape patterns.

experiments we visualize the learned convolutional filters to grain insights on the visual patterns that are discriminative for the classification. One of the motivations for the bilinear model was the modular separation of factors that affect the overall appearance. We are curious if the two networks specialize into different roles when initialized asymmetrically and fine-tuned. We shows the top activations of several filters form *relu5_3* layer for the D-Net and *relu5* layer for M-Net of the fine-tuned B-CNN [D, M] model for birds (Figure 5.6), aircrafts (Figure 5.7) and cars (Fig-

Figure 5.8. Patches with the highest activation for several filters of the fine-tuned B-CNN (VGG-D + VGG-M) model on cars dataset. These visualizations indicate that network units activate on highly localized attributes of objects that capture color, texture, and shape patterns.

ure 5.8) classification. Both these networks contain units that activate strongly on highly localized features. For example, in Figure 5.6 the last row of VGG-D detects "tufted heads", while the fourth row in the same column detects a "red-yellow stripe" on birds. Similarly for airplanes, the units localize different types of windows, noses, vertical stabilizers, with some specializing in detecting particular airliner logos. For cars, units activate on different kinds of head/tail lights, wheels, *etc.*. The visualization suggests that the roles of the two networks are not clearly separated. Several

work [136, 138] has explored interpretable models that attend to the local features at pre-defined object parts. Ideally, the fine-tuning should be able to figure out the optimal feature factorization given sufficient amount of training data. Whether using the priors on pre-defined object parts leads to a better model remains unclear.

5.4 Manipulating images with texture attributes

Our framework can be used to edit images with texture attributes. For instance, we can make a texture or the content of an image more honeycombed or swirly. Fig. 5.9 shows some examples where we have modified images with various attributes. The top two rows of images were obtained by setting $\alpha_i = 1\ \forall i$, $\beta = 1000$ and $\gamma = 1e-6$ and varying $\hat{C}$ to represent the target class. The bottom row is obtained by setting $\alpha_i = 0\ \forall i$, and using the *relu4_2* layer for content reconstruction with weight $\lambda = 5e-8$.

The difference between the two is that in the content reconstruction the overall structure of the image is preserved. The approach is similar to the neural style approach [45], but instead of providing a style image we adjust the image with attributes. This leads to interesting results. For instance, when the face image is adjusted with the interlaced attribute (Fig. 5.9 bottom row) the result matches the scale and orientation of the underlying image. No single image in the DTD dataset has all these variations but the categorical representation does. The approach can be used to modify an image with other high-level attributes such as artistic styles by learning style classifiers.

We can also blend texture categories using weights β_j of the targets $\hat{C}_j$. Fig. 5.10 shows some examples. On the left is the first category, on the right is the second category, and in the middle is where a transition occurs (selected manually).

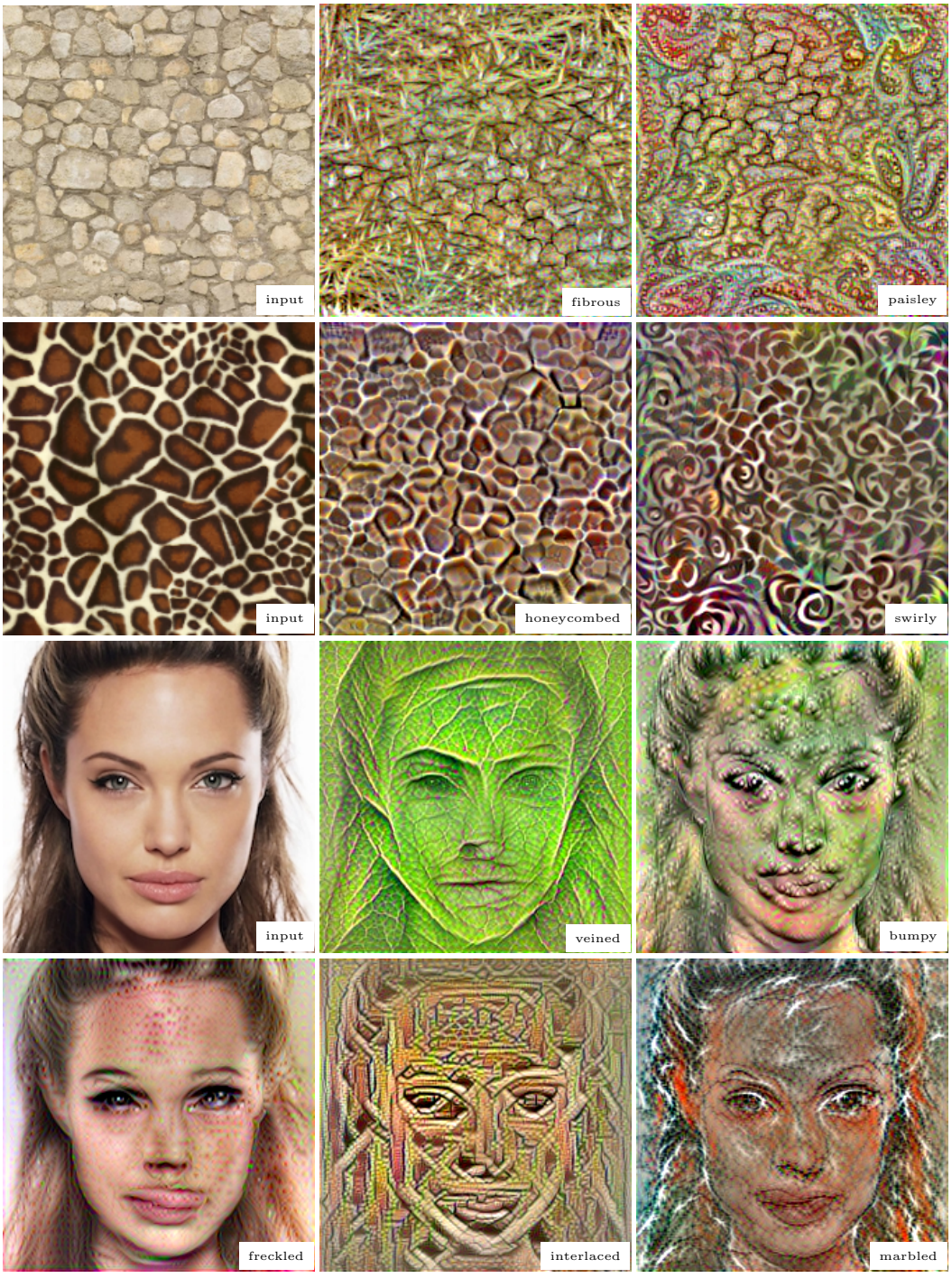


Figure 5.9. Manipulating images with attributes. Given an image we synthesize a new image that matches its texture (*top two rows*) or its content (*bottom two rows*) according to a given attribute (*shown in the image*).

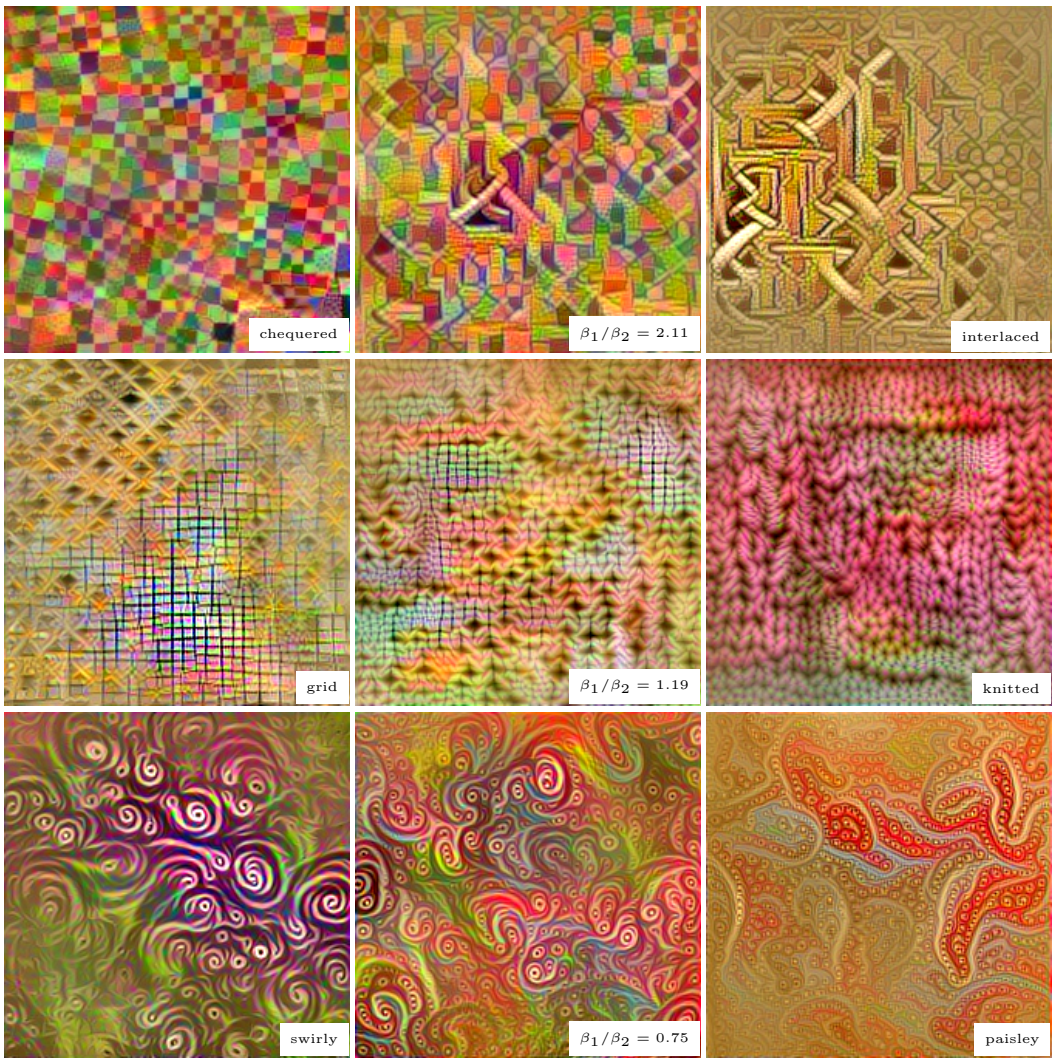


Figure 5.10. Hybrid textures obtained by blending the texture on the left and right according to weights β_1 and β_2.

5.5 Summary

In this chapter, we presented a systematic framework for synthesizing texture images and studied the invariances of B-CNN models by inverting them. The resulting models provide a parametric approach for texture synthesis and manipulation of the content of images using texture attributes. The critical challenge is that the approach is computationally expensive, and we present an initialization scheme based on image quilting that significantly speeds up the convergence and also removes many struc-

tural artifacts that quilting introduces. A visualization of B-CNNs showed that these models effectively represent objects as *texture* and their units are correlated with localized attributes useful for fine-grained recognition.

CHAPTER 6

OTHER CONTRIBUTIONS: MISTNET – MEASURING HISTORICAL BIRD MIGRATION WITH CNNS

In addition to my main research topic in learning higher-order representations, I am also interested in using computer vision techniques to analyze large-scale data and solve real-world problems. I collaborated with Prof. Sheldon and Prof. Maji working on Dark Ecology project [1] and developed MistNet, a deep learning model for segmenting biological targets from radar data. The model automates the pre-screening process that required substantial manual efforts prior to the analysis and allows ornithologists to conduct large-scale studies on bird migration. I will divert from higher-order representations to give a brief introduction of MistNet in this chapter. More detail can be found in MistNet paper [85].

6.1 Introduction

Researchers discovered more than 70 years ago that radars, originally designed for military purposes, can also detect bird movements [13, 76]. With the advent of large networks of weather radars, the possibility arose to use radar as a distributed instrument to quantify whole migration systems [46, 14, 47, 48, 32, 6, 99]. The US WSR-88D[1] weather radar network [24] stands out as one of the most comprehensive instruments for studying migration due to its size, uniformity, and historical data archive.

[1] **W**eather **S**urveillance **R**adar, 19**88**, **D**oppler

Weather radar data can answer a wide range of important migration ecology questions. Previous studies have used weather radar data to understand patterns and determinants of nocturnal migration [48, 68, 75, 38], identify critical stopover habitat [16, 15], locate on-the-ground roosting sites of birds [129, 17, 78, 77, 12], and many others. However, it required substantial manual effort to enable large-scale studies, primarily to screen radar images for precipitation and other unwanted targets prior to analysis [47, 48]. Human interpretation of images has become a substantial barrier to very large-scale research with WSR-88D data.

Recent advances have led to the first fully automated methods to extract biological information from weather radar data. In 2012–2013, the WSR-88D network was upgraded to dual polarization technology, which makes it significantly easier to separate biology from precipitation in modern data [115], but leaves open the problem of extracting biological information from historical data. One of the work during my PhD was to develop MistNet, a deep convolutional neural network (CNN) to separate precipitation from biology at a fine spatial resolution in historical WSR-88D data. Radar images contain clear visual patterns that allow humans to discriminate precipitation from biology. We trained deep learning models to automatically recognize these patterns.

6.2 MISTNET architecture and training

Historical radar data produces 3 measurements at each spatial locations. We collected the measurements from 5 elevations resulting in the input of dimension $15 \times 600 \times 600$. Radar geometry is show in Fig. 6.1 and more details can be found in [1]. The output $\mathbf{z}$ of MistNet has dimension $3 \times 5 \times 600 \times 600$, which corresponds to the class probability for each of 3 classes (precipitation, biology, background) at each position in five 600×600 images, one for each elevation angle. At prediction time, the class with the highest probability is predicted. Let $f(\mathbf{x}; \boldsymbol{\theta})$ be the function

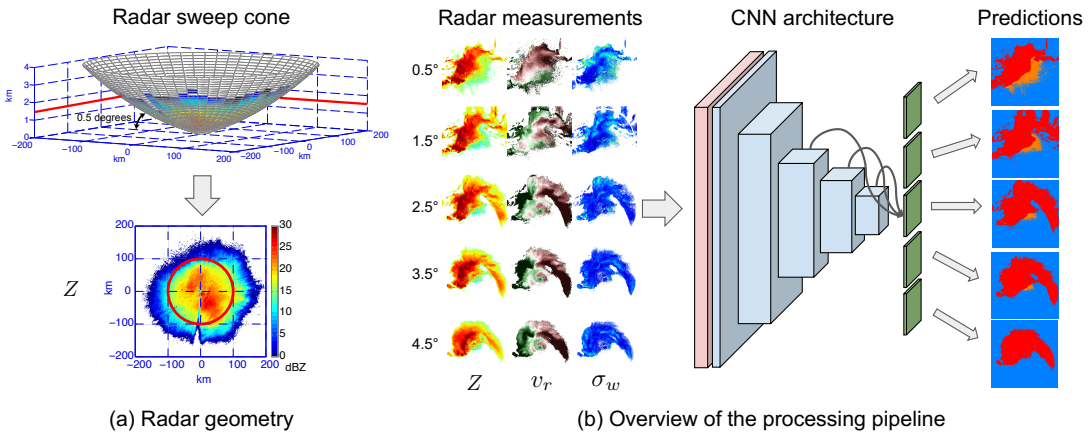


Figure 6.1. (a) Radar geometry. A sweep, shown here at an elevation of 0.5 degrees, traces out an approximately conical surface and is usually rendered as top-down image or "plan-position indicator" (i.e., PPI). **(b) Overview of processing pipeline.** Radar measurements are collected on a three-dimensional polar grid (3 products x 5 elevations) and rendered as a 15-channel "image" in Cartesian coordinates. An adapter network maps 15 channels to 3 channels to match a conventional RGB image. The CNN processes the image and outputs five segmentation masks, one for each elevation. Each segmentation mask delimits areas containing biology and weather (red: rain, orange: biology, blue: background). The inputs, intermediate activations, and outputs of the CNN are three-dimensional arrays arranged in layers and depicted as boxes (pink: input, light blue: intermediate, green: output). The activations in output branches (green boxes) are functions of several earlier layers, shown for one branch with black curved arrows.

describing the entire mapping from the CNN's input to its output, so that $\mathbf{z} = f(\mathbf{x}; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ contains the parameters of all layers.

MistNet is based on the FCN8 (fully convolutional network with predictions at a spatial granularity of 8 pixels) architecture from [87] with an ImageNet pre-trained VGG-16 "backbone" [114]. See also Figure 6.1. We added a linear adapter network to map the input data from 15 to 3 channels at each spatial location for compatibility with the input dimensions of the VGG-16 network, and trained the parameters of the linear adapter. Unlike the standard FCN8 network, which predicts one value per spatial location, MistNet makes five predictions, one per elevation. This is accomplished by creating five separate branches that take as input the activations

of several preceding convolutional layers and output class probabilities (the curved arrows in Figure 6.1 represent one of these branches).

Post-processing predictions The standard prediction rule is to classify a pixel as precipitation if the predicted class probability for precipitation exceeds 0.5. In preliminary experiments we observed that MistNet underpredicted precipitation at the boundaries of rain storms and sometimes missed rain mixed with biology at low elevations. We developed the following postprocessing rules to improve these cases: we predict a pixel as rain if the class probability for rain exceeds 0.45 *or* if the average class probability for rain across the five elevations at that spatial location exceeds 0.45. We further compute a "fringe" of 8 pixels surrounding any rain pixel and classify the fringe as rain, with the goal of conservatively removing as much rain as possible due to its possible adverse impacts of biological analysis.

Weak training labels Because we do not have a large data set of radar images with pixel-level labels, we conducted transfer learning from image classification models trained on the ImageNet dataset [29]. We initialized MistNet's model parameters using those models, and then adapted the parameters by training with weak annotations obtained from a simple rule to discriminate precipitation from biology using dual-pol data.

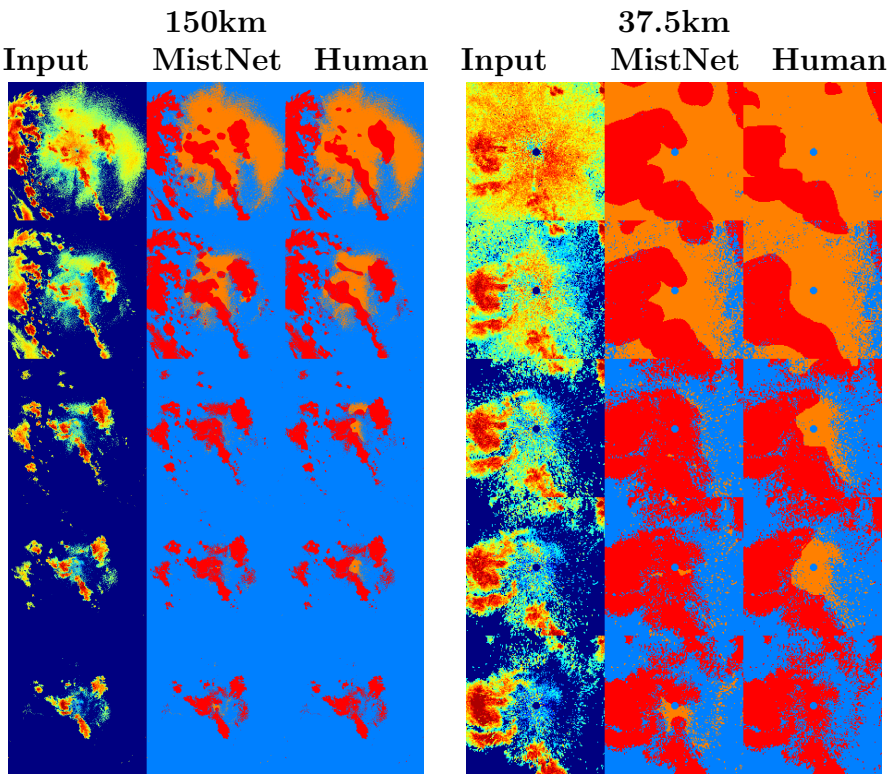
Biological scatterers tend to have a much lower correlation coefficient than hydrometeors because their orientation, position, and shape are much more variable in time [115, 70]. It has become common practice among radar biology practitioners to use a threshold of $\rho_{\text{HV}} \leq 0.95$ to identify biological scatterers [31]. Although weather events such as mixed precipitation can also produce ρ_{HV} values this low [82], this rule is believed to have reasonable accuracy in general, and has been validated through comparisons with a colocated bird radar [32, 98]. Little is known about the best threshold value or pixel-level accuracy of this method.

We used the following rules to generate training labels for MistNet. For each pixel, if the reflectivity factor Z is reported as "no data" (below signal-to-noise threshold) then we set the label to "background". Otherwise, if $\rho_{\mathrm{HV}} > 0.95$ we set the label to "precipitation". All remaining labels are "biology". We included the background class during training to avoid semantic confusion resulting from forcing the model to predict background pixels as either weather or biology. At prediction time, it is known whether or not a pixel belongs to the background class, and predictions are only made on non-background pixels.

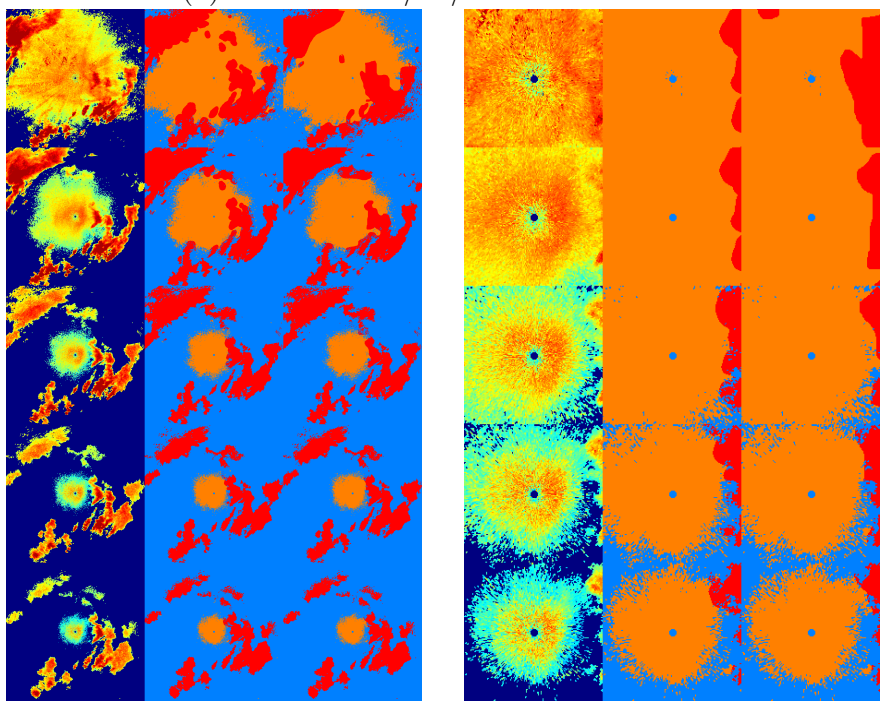
Evaluation data We collected two separate evaluation data sets of human-labeled ground truth data: a geographically representative *contemporary* set, and a historically representative *historical* set. Detail description of the datasets can be found in [1]. All evaluations were performed for the region within 37.5 km ground range from the radar. To measure pixel-level performance we first computed a confusion matrix in which each pixel was weighted by reflectivity factor on a linear scale ($\mathrm{mm}^6\,\mathrm{mm}^{-3}$) after first capping values at 35 dBZ to limit the effect of extremely high values that are typically discarded in biological analyses. From the confusion matrix, we computed the standard metrics of *precision* (fraction of predicted biology that is actually biology), *recall* (fraction of true biology that is predicted to be biology), and *F-score* (harmonic mean of precision and recall).

6.3 Results

Overall performance Figure 6.2 shows several examples of predictions made by MistNet compared to the ground-truth human annotations. Table **??** gives the overall performance measurements of MistNet on the historical and contemporary evaluation sets. The overall prevalance of precipitation is higher in historical data (81.4 % vs. 47.6 %).



(a) KBGM 2014/10/01 02:15:53 GMT



(b) KMOB 2007/09/01 03:10:00 GMT

Figure 6.2. MistNet Segmentation Results. Segmentation results (red: rain, orange: biology, blue: background) predicted by MistNet are shown along with the human annotations in the ranges of 150km and 37.5km. Each example is shown as a stack of five rows from top to bottom corresponding to the elevation angles from 0.5 to 4.5 degrees.

Data set	Post-processing?	Precision	Recall	F-score
Historical (all)	no	93.5	99.0	96.2
	yes	98.7	95.9	97.3
Historical (weather)	no	72.6	96.1	82.7
	yes	92.7	82.8	87.5
Contemporary	no	96.4	99.1	97.7
	yes	99.1	96.7	97.9

Table 6.1. Performance of MistNet with and without post-processing.

6.4 Summary

MistNet provides a fully automated method for extracting biological signals from historical WSR-88D weather radar data and opens the entirety of the more-than-25-year archive of US weather radar data for large-scale biological studies. Our results show that deep learning is an effective tool for discriminating rain from biology in radar data. Key ingredients to MistNet's success are a large enough training set, which is enabled by gathering labels automatically from dual polarization data, and an architecture that is able to use all available information—from all products across multiple elevations—while making predictions. An interesting technical aspect of MistNet's architecture is the fact that information is compressed down from 15 to 3 channels at the first layer, but MistNet is later able to make predictions at 5 separate elevations. The exact mechanisms by which the model compresses and retrieves information from these channels is an interesting topic of future research.

There are several promising research directions for future applications of deep learning to radar tasks. One direction is to improve performance by tracking recent progress in deep learning for images, for example, to adopt architectures such as residual networks [55] instead of the VGG-16 architecture used in MistNet. A more substantial change would be to explore novel architectures that are completely customized for radar data, which would necessitate training models from scratch.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

7.1 Summary of contributions

In this thesis, we explored texture-based representations in the context of aggregating convolutional features for end-to-end feature learning. We demonstrated that our B-CNN models are effective in modeling distinctive local patterns and achieve state-of-the-art results on fine-grained and texture recognition. This is achieved by modeling multiplicative feature interactions between two CNN activations and using feature normalizations to handle bursty features. To address the concerns of efficiency, we studied techniques to speed up the forward and backward computations for matrix square-root normalization. To address the concerns of memory usage, we studied techniques to reduce the feature dimension.

To understand the invariance of the bilinear models, we presented techniques to synthesize texture images that visualize the invariances under a given B-CNN representation or a specified target category. The visualization showed that the property of translational invariance allows B-CNN representations to capture distinct local appearance while the spatial arrangement of patches is ignored. The visualization of pre-images that maximize the prediction scores of given categories provided insight into the properties that are captured by the classification models.

In future research, we suggest exploring models that allow a detailed understanding of fine-grained objects beyond object categorization and can generalize to extreme conditions such as few-shot or long-tail settings. We describe some of the challenges in the following sections.

7.2 Comprehensive understanding of fine-grained objects

Although B-CNN models are able to do a reasonable job in recognizing fine-grained categories, the current recognition models still cannot provide a comprehensive understanding of deformable objects regarding their poses, viewpoints, and background. This is challenging because state-of-the-art machine learning algorithms rely on a massive amount of human annotations for supervised training, and providing detailed annotations beyond category labels at scale is prohibitive.

One of the research directions to achieve better fine-grained object understanding is to learn disentangled representations. This line of research attempts to recover underlying variables that are accountable for generating the image of objects, and hopefully, these variables emerge in some interpretable ways; for example, a set of variables are related to pose, a set of variables are related to appearance, and another set of variables are related to background. Several attempts [37, 91, 92, 88, 36] have been made in this direction, but they mainly focused on human poses for which there is massive amount of training data, rich annotations, and strong supervision. Without a massive amount of annotations, modeling object deformation, in general, is challenging in 2D due to occlusion and perspective projection. Recent works [67, 140] have shown some success in modeling object shapes and deformations in 3D via the reconstruction of 2D images. This might be a promising direction as the occlusion and perspective projections can be handled by camera projection.

7.3 Learning interpretable feature decomposition

We demonstrated that B-CNNs model two-factor decomposition and they are effective in fine-grained recognition; however, the decomposition is not interpretable, as shown in the visualization provided in Section 5.3.3. A particular desired decomposition motivated by classical part-based models [134, 39] is the separation of object parts localization and local appearance. Regularizing the training procedure to achieve the

feature decomposition without extra annotations beyond category labels is still challenging. Recent works on part-based attention models [136, 138] attempted to model fine-grained objects explicitly by the part-appearance decomposition. Although these models showed marginal improvement over B-CNN, the part-appearance decomposition might be beneficial in transfer learning settings, especially when we have limited training instances. We expect a better generalization of object part localization to novel categories and an easier learning problem to model local appearance conditioned on object parts.

7.4 Generalization to tail categories

Higher-order representations increase the feature dimension quadratically and are prone to overfitting when the data are scarce. Intuitively, when training data is not sufficient, the models are likely to pick up accidental feature correlations. Few-shot learning is an active research topic in machine learning. Specifically, for fine-grained recognition, the objects are naturally distributed as a long-tail distribution rather than a few-shot setting. Recent work [128] trained prototypical networks using B-CNN representations on heavy-tailed fine-grained classification and demonstrated that second-order features outperform their first-order counterpart. However, in the paper, the experiments showed that the improvement arose from head categories while the high feature dimensionality seemed to weaken the performance on the few-shot categories when data is limited. As fine-grained objects share similar structures among categories, we argue that the intra-category variances may be captured on the head categories and transferred to the tail. This requires a conditional model that allows the manipulation of data in a structured way to help the recognition systems learn the right invariance of categories. How to model these variances and leverage them to improve the recognition accuracy when data is scarce is still a very challenging problem.

APPENDIX

PROOF FOR PROPOSITIONS

A.1 Proofs of Proposition 4

1. The ℓ_2 norm of $\text{vec}(\mathbf{A}^p)$ is $\rho(\mathbf{A}^p) = ||\,\text{vec}(\mathbf{A}^p)|| = \left(\sum_i \lambda_i^{2p}\right)^{1/2}$.

   Proof. We have:

$$\begin{aligned} ||\,\text{vec}(\mathbf{A}^p)||^2 &= \text{vec}(\mathbf{A}^p)^T\,\text{vec}(\mathbf{A}^p) \\ &= \texttt{Trace}((\mathbf{A}^p)^T\mathbf{A}^p) \\ &= \texttt{Trace}(\mathbf{U}\Lambda^{2p}\mathbf{U}^T) \\ &= \sum_i \lambda_i^{2p}. \end{aligned}$$

   Thus the ℓ_2 norm: $\rho(\mathbf{A}^p) = ||\,\text{vec}(\mathbf{A}^p)|| = \left(\sum_i \lambda_i^{2p}\right)^{1/2}$ □

2. $\sum_{\mathbf{x}\in\mathcal{X}} C(\mathbf{x}) = \texttt{Trace}(\mathbf{A}^{1+p}/||\mathbf{A}^p||) = \left(\sum_i \lambda_i^{1+p}\right)/\rho(\mathbf{A}^p)$.

   Proof. We have:

$$\begin{aligned} \sum_{\mathbf{x}\in\mathcal{X}} C(\mathbf{x}) &= \sum_{\mathbf{x}\in\mathcal{X}} \text{vec}(\mathbf{x}\mathbf{x}^T)^T\,\text{vec}(\hat{\mathbf{A}}^p) \\ &= \text{vec}(\mathbf{A})^T\,\text{vec}(\hat{\mathbf{A}}^p) \\ &= \texttt{Trace}(\mathbf{A}^T\hat{\mathbf{A}}^p) \\ &= \texttt{Trace}(\mathbf{A}^T\mathbf{A}^p)/\rho(\mathbf{A}^p) \\ &= \left(\sum_i \lambda_i^{1+p}\right)/\rho(\mathbf{A}^p) \end{aligned}$$

   □

3. The maximum value $M = \max_{\mathbf{x} \in \mathcal{X}} C(\mathbf{x}) \leq r_{\max} \lambda_1^p / \rho(\mathbf{A}^p)$.

Proof. We have

$$\begin{aligned} C(\mathbf{x}) &= \text{vec}(\mathbf{x}\mathbf{x}^T)^T \, \text{vec}(\hat{\mathbf{A}}^p) \\ &= \texttt{Trace}((\mathbf{x}\mathbf{x}^T)^T \hat{\mathbf{A}}^p) \\ &= \texttt{Trace}(\mathbf{x}^T \mathbf{A}^p \mathbf{x}) / \rho(\mathbf{A}^p) \\ &\leq ||\mathbf{x}||^2 \lambda_1^p / \rho(\mathbf{A}^p) \\ &\leq r_{\max} \lambda_1^p / \rho(\mathbf{A}^p) \end{aligned}$$

□

4. The minimum value $m = \min_{\mathbf{x} \in \mathcal{X}} C(\mathbf{x}) \geq r_{\min} \lambda_d^p / \rho(\mathbf{A}^p)$.

Proof. We have:

$$\begin{aligned} C(\mathbf{x}) &= \texttt{Trace}(\mathbf{x}^T \mathbf{A}^p \mathbf{x}) / \rho(\mathbf{A}^p) \\ &\geq ||\mathbf{x}||^2 \lambda_d^p / \rho(\mathbf{A}^p) \\ &\geq r_{\min} \lambda_d^p / \rho(\mathbf{A}^p) \end{aligned}$$

□

A.2 Proof of Proposition 6

Proof. Here is an example where the matrix power $\mathbf{A}^p$ does not lie in the linear span of the outer-products of the features $\mathbf{x} \in \mathcal{X}$. Consider two vectors $\mathbf{x}_1 = [1 \; 0]^T$ and $\mathbf{x}_2 = [1 \; 1]^T$. The covariance matrix $\mathbf{A}$ formed by the two is

$$
\begin{aligned}
\mathbf{A} &= \mathbf{x}_1\mathbf{x}_1^T + \mathbf{x}_2\mathbf{x}_2^T \\
&= \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \\
&= \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}
\end{aligned}
$$

The square root of the matrix $\mathbf{A}$ is:

$$
\mathbf{A}^{1/2} = \begin{bmatrix} 1.3416 & 0.4472 \\ 0.4472 & 0.8944 \end{bmatrix}
$$

It is easy to see that $\mathbf{A}^{1/2}$ cannot be written as a linear combination of $\mathbf{x}_1\mathbf{x}_1^T$ and $\mathbf{x}_2\mathbf{x}_2^T$ since any linear combination will have all equal values for all the entries except possibly the top left value.

A sufficient condition for $\mathbf{A}^p$ to be in the linear span of outer products is that the vectors $\mathbf{x} \in \mathcal{X}$ which are used in constructing $\mathbf{A}$ be orthogonal to each other. This however, is not true in general for features extracted from convolutional layers. $\square$

BIBLIOGRAPHY

[1] Dark ecology. `http://darkecology.cs.umass.edu`.

[2] inaturalist 2018 competition. `https://github.com/visipedia/inat_comp`.

[3] Arandjelović, R., Gronat, P., Torii, A., Pajdla, T., and Sivic, J. NetVLAD: CNN architecture for weakly supervised place recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016).

[4] Ba, Jimmy, Mnih, Volodymyr, and Kavukcuoglu, Koray. Multiple object recognition with visual attention. In *International Conference on Learning Representations* (2015).

[5] Bartels, Richard H., and Stewart, GW. Solution of the matrix equation AX+XB=C. *Communications of the ACM 15*, 9 (1972), 820–826.

[6] Bauer, Silke, Chapman, Jason W., Reynolds, Don R., Alves, José A., Dokter, Adriaan M., Menz, Myles M. H., Sapir, Nir, Ciach, Michał, Pettersson, Lars B., Kelly, Jeffrey F., Leijnse, Hidde, and Shamoun-Baranes, Judy. From agricultural benefits to aviation safety: Realizing the potential of continent-wide radar networks. *BioScience 67*, 10 (2017), 912–918.

[7] Ben-Israel, Adi. An iterative method for computing the generalized inverse of an arbitrary matrix. *Mathematics of Computation* (1965), 452–455.

[8] Ben-Israel, Adi. A note on an iterative method for generalized inversion of matrices. *Mathematics of Computation 20*, 95 (1966), 439–440.

[9] Bhatia, Rajendra, and Davis, Chandler. A better bound on the variance. *The American Mathematical Monthly 107*, 4 (2000), 353–357.

[10] Bourdev, L., Maji, S., and Malik, J. Describing people: A poselet-based approach to attribute classification. In *IEEE International Conference on Computer Vision (ICCV)* (2011).

[11] Branson, Steve, Horn, Grant Van, Belongie, Serge, and Perona, Pietro. Bird species categorization using pose normalized deep convolutional nets. In *British Machine Vision Conference (BMVC)* (2014).

[12] Bridge, Eli S., Pletschet, Sandra M., Fagin, Todd, Chilson, Phillip B., Horton, Kyle G., Broadfoot, Kyle R., and Kelly, Jeffrey F. Persistence and habitat associations of Purple Martin roosts quantified via weather surveillance radar. *Landscape Ecology 31*, 1 (2016), 43–53.

[13] Brooks, Maurice. Electronics as a possible aid in the study of bird flight and migration. *Science 101*, 2622 (1945), 329.

[14] Bruderer, Bruno. The study of bird migration by radar part 1: The technical basis. *Naturwissenschaften 84*, 1 (1997), 1–8.

[15] Buler, Jeffrey J., and Dawson, Deanna K. Radar analysis of fall bird migration stopover sites in the northeastern US. *The Condor 116*, 3 (2014), 357–370.

[16] Buler, Jeffrey J., and Diehl, Robert H. Quantifying bird density during migratory stopover using weather surveillance radar. *IEEE Transactions on Geoscience and Remote Sensing 47*, 8 (2009), 2741–2751.

[17] Buler, Jeffrey J., Randall, Lori A., Fleskes, Joseph P., Barrow, Jr., Wylie C., Bogart, Tianna, and Kluver, Daria. Mapping wintering waterfowl distributions using weather surveillance radar. *PloS one 7*, 7 (2012), e41571.

[18] Caputo, B., Hayman, E., and Mallikarjuna, P. Class-specific material categorisation. In *IEEE International Conference on Computer Vision (ICCV)* (2005).

[19] Carreira, Joao, Caseiro, Rui, Batista, Jorge, and Sminchisescu, Cristian. Semantic segmentation with second-order pooling. In *European Conference on Computer Vision (ECCV)*. 2012.

[20] Chatfield, K., Simonyan, K., Vedaldi, A., and Zisserman, A. Return of the devil in the details: Delving deep into convolutional nets. In *British Machine Vision Conference (BMVC)* (2014).

[21] Chen, Yunpeng, Kalantidis, Yannis, Li, Jianshu, Yan, Shuicheng, and Feng, Jiashi. Aˆ 2-nets: Double attention networks. In *Advances in Neural Information Processing Systems* (2018), pp. 352–361.

[22] Cimpoi, M., Maji, S., Kokkinos, I., Mohamed, S., and Vedaldi, A. Describing textures in the wild. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2014).

[23] Cimpoi, Mircea, Maji, Subhransu, Kokkinos, Iasonas, and Vedaldi, Andrea. Deep filter banks for texture recognition, description, and segmentation. *International Journal of Computer Vision 118*, 1 (2016), 65–94.

[24] Crum, Timothy D., and Alberty, Ron L. The WSR-88D and the WSR-88D operational support facility. *Bulletin of the American Meteorological Society 74*, 9 (1993), 1669–1687.

[25] Csurka, G., Dance, C. R., Dan, L., Willamowski, J., and Bray, C. Visual categorization with bags of keypoints. In *ECCV Workshop on Statistical Learning in Computer Vision* (2004).

[26] Cui, Yin, Zhou, Feng, Wang, Jiang, Liu, Xiao, Lin, Yuanqing, and Belongie, Serge. Kernel pooling for convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017).

[27] Dai, Jifeng, Qi, Haozhi, Xiong, Yuwen, Li, Yi, Zhang, Guodong, Hu, Han, and Wei, Yichen. Deformable convolutional networks. In *Proceedings of the IEEE international conference on computer vision* (2017), pp. 764–773.

[28] Dai, Xiyang, Yue-Hei Ng, Joe, and Davis, Larry S. FASON: First and Second Order Information Fusion Network for Texture Recognition. In *CVPR* (2017).

[29] Deng, Jia, Dong, Wei, Socher, Richard, Li, Li-Jia, Li, Kai, and Fei-Fei, Li. ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2009), IEEE, pp. 248–255.

[30] Denman, Eugene D, and Beavers, Alex N. The matrix sign function and computations in systems. *Applied mathematics and Computation 2*, 1 (1976), 63–94.

[31] Dokter, Adriaan M., Desmet, Peter, Spaaks, Jurriaan H., van Hoey, Stijn, Veen, Lourens, Verlinden, Liesbeth, Nilsson, Cecilia, Haase, Günther, Leijnse, Hidde, Farnsworth, Andrew, Bouten, Willem, and Shamoun-Baranes, Judy. bioRad: biological analysis and visualization of weather radar data. *Ecography* (2018).

[32] Dokter, Adriaan M., Liechti, Felix, Stark, Herbert, Delobbe, Laurent, Tabary, Pierre, and Holleman, Iwan. Bird migration flight altitudes studied by a network of operational weather radars. *Journal of The Royal Society Interface 8*, 54 (2011), 30–43.

[33] Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, Ning, Tzeng, E., and Darrell, T. Decaf: A deep convolutional activation feature for generic visual recognition. In *International Conference on Machine Learning* (2013).

[34] Dosovitskiy, Alexey, and Brox, Thomas. Inverting visual representations with convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 4829–4837.

[35] Efros, Alexei A, and Freeman, William T. Image quilting for texture synthesis and transfer. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), ACM, pp. 341–346.

[36] Esser, Patrick, Haux, Johannes, and Ommer, Bjorn. Unsupervised robust disentangling of latent characteristics for image synthesis. In *Proceedings of the IEEE International Conference on Computer Vision* (2019), pp. 2699–2709.

[37] Esser, Patrick, Sutter, Ekaterina, and Ommer, Björn. A variational u-net for conditional appearance and shape generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), pp. 8857–8866.

[38] Farnsworth, Andrew, Van Doren, Benjamin M., Hochachka, Wesley M., Sheldon, Daniel, Winner, Kevin, Irvine, Jed, Geevarghese, Jeffrey, and Kelling, Steve. A characterization of autumn nocturnal migration detected by weather surveillance radars in the northeastern USA. *Ecological Applications 26*, 3 (2016), 752–770.

[39] Farrell, Ryan, Oza, Om, Zhang, Ning, Morariu, Vlad I, Darrell, Trevor, and Davis, Larry S. Birdlets: Subordinate categorization using volumetric primitives and pose-normalized appearance. In *IEEE International Conference on Computer Vision (ICCV)* (2011).

[40] Feichtenhofer, C., Pinz, A., and Zisserman, A. Convolutional two-stream network fusion for video action recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016).

[41] Felzenszwalb, Pedro F, Girshick, Ross B, McAllester, David, and Ramanan, Deva. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence 32*, 9 (2009), 1627–1645.

[42] Fukui, Akira, Park, Dong Huk, Yang, Daylen, Rohrbach, Anna, Darrell, Trevor, and Rohrbach, Marcus. Multimodal compact bilinear pooling for visual question answering and visual grounding. *CoRR abs/1606.01847* (2016).

[43] Gao, Yang, Beijbom, Oscar, Zhang, Ning, and Darrell, Trevor. Compact bilinear pooling. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016).

[44] Gatys, L. A., Ecker, A. S., and Bethge, M. Texture synthesis using convolutional neural networks. In *Advances in Neural Information Processing Systems* (2015).

[45] Gatys, Leon A., Ecker, Alexander S., and Bethge, Matthias. Image style transfer using convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016).

[46] Gauthreaux, Jr., Sidney A. Weather radar quantification of bird migration. *BioScience 20*, 1 (1970), 17–19.

[47] Gauthreaux, Jr., Sidney A., and Belser, Carroll G. Displays of bird movements on the WSR-88D: patterns and quantification. *Weather and Forecasting 13*, 2 (1998), 453–464.

[48] Gauthreaux, Jr., Sidney A., Belser, Carroll G., and Van Blaricom, Donald. Using a network of WSR-88D weather surveillance radars to define patterns of bird migration at large spatial scales. In *Avian migration*. Springer, 2003, pp. 335–346.

[49] Ge, Weifeng, Lin, Xiangru, and Yu, Yizhou. Weakly supervised complementary parts models for fine-grained image classification from the bottom up. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2019).

[50] Girshick, R. B., Donahue, J., Darrell, T., and Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2014).

[51] Girshick, Ross, Iandola, Forrest, Darrell, Trevor, and Malik, Jitendra. Deformable part models are convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition* (2015), pp. 437–446.

[52] Gosselin, Philippe-Henri, Murray, Naila, Jégou, Hervé, and Perronnin, Florent. Revisiting the fisher vector for fine-grained classification. *Pattern Recognition Letters 49* (2014), 92–98.

[53] He, K., Zhang, X., Ren, S., and Sun, J. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *European Conference on Computer Vision (ECCV)* (2014).

[54] He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016).

[55] He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 770–778.

[56] Higham, Nicholas J. Stable iterations for the matrix square root. *Numerical Algorithms 15*, 2 (1997), 227–242.

[57] Hu, Jie, Shen, Li, and Sun, Gang. Squeeze-and-excitation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), pp. 7132–7141.

[58] Huang, Gao, Liu, Zhuang, Van Der Maaten, Laurens, and Weinberger, Kilian Q. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2017), pp. 4700–4708.

[59] Ioffe, Sergey, and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning* (2015).

[60] Ionescu, Catalin, Vantzos, Orestis, and Sminchisescu, Cristian. Matrix backpropagation for deep networks with structured layers. In *IEEE International Conference on Computer Vision (ICCV)* (2015).

[61] Jaderberg, Max, Simonyan, Karen, Zisserman, Andrew, and Kavukcuoglu, Koray. Spatial transformer networks. In *Advances in Neural Information Processing Systems*. 2015.

[62] Jégou, H., Douze, M., Schmid, C., and Pérez, P. Aggregating local descriptors into a compact image representation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2010).

[63] Jégou, Hervé, Douze, Matthijs, and Schmid, Cordelia. On the burstiness of visual elements. In *2009 IEEE Conference on Computer Vision and Pattern Recognition* (2009), IEEE, pp. 1169–1176.

[64] Jégou, Hervé, and Zisserman, A. Triangulation embedding and democratic aggregation for image search. In *CVPR* (2014).

[65] Johnson, Justin, Alahi, Alexandre, and Fei-Fei, Li. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision* (2016), Springer, pp. 694–711.

[66] Julesz, Bela. Textons, the elements of texture perception, and their interactions. *Nature 290(5802)* (march 1981), 91–97.

[67] Kanazawa, Angjoo, Tulsiani, Shubham, Efros, Alexei A., and Malik, Jitendra. Learning category-specific mesh reconstruction from image collections. In *ECCV* (2018).

[68] Kemp, Michael U., ShamounBaranes, Judy, Dokter, Adriaan M., van Loon, Emiel, and Bouten, Willem. The influence of weather on the flight altitude of nocturnal migrants in midlatitudes. *Ibis 155*, 4 (2013), 734–749.

[69] Khan, Salman H, Hayat, Munawar, and Porikli, Fatih. Scene Categorization with Spectral Features. In *ICCV* (2017).

[70] Kilambi, Alamelu, Fabry, Frédéric, and Meunier, Véronique. A simple and effective method for separating meteorological from nonmeteorological targets using dual-polarization data. *Journal of Atmospheric and Oceanic Technology 35*, 7 (2018), 1415–1424.

[71] Knight, Philip A. The sinkhorn-knopp algorithm: Convergence and applications. *SIAM J. Matrix Anal. Appl. 30*, 1 (Mar. 2008), 261–275.

[72] Krause, Jonathan, Jin, Hailin, Yang, Jianchao, and Fei-Fei, Li. Fine-grained recognition without part annotations. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015).

[73] Krause, Jonathan, Stark, Michael, Deng, Jia, and Fei-Fei, Li. 3d object representations for fine-grained categorization. In *3D Representation and Recognition Workshop, at ICCV* (2013).

[74] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems* (2012).

[75] La Sorte, Frank A., Hochachka, Wesley M., Farnsworth, Andrew, Sheldon, Daniel, Fink, Daniel, Geevarghese, Jeffrey, Winner, Kevin, Van Doren, Benjamin M., and Kelling, Steve. Migration timing and its determinants for nocturnal migratory birds during autumn migration. *Journal of Animal Ecology 84*, 5 (2015), 1202–1212.

[76] Lack, David, and Varley, G. C. Detection of birds by radar. *Nature 156*, 3963 (1945), 446.

[77] Laughlin, Andrew J., Sheldon, Daniel R., Winkler, David W., and Taylor, Caz M. Quantifying nonbreeding season occupancy patterns and the timing and drivers of autumn migration for a migratory songbird using Doppler radar. *Ecography 39*, 10 (10 2016), 1017–1024.

[78] Laughlin, Andrew J., Taylor, Caz M., Bradley, David W., Leclair, Dayna, Clark, Robert G., Dawson, Russell D., Dunn, Peter O., Horn, Andrew, Leonard, Marty, Sheldon, Daniel, Shutler, Dave, Whittingham, Linda A., Winkler, David W., and Norris, D. Ryan. Integrating information from geolocators, weather radar and citizen science to uncover a key stopover area for an aerial insectivore. *The Auk 130*, 2 (2013), 230–239.

[79] Lazebnik, Svetlana, Schmid, Cordelia, and Ponce, Jean. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2006).

[80] Leung, T., and Malik, J. Representing and recognizing the visual appearance of materials using three-dimensional textons. *International Journal of Computer Vision 43*, 1 (2001), 29–44.

[81] Li, Peihua, Xie, Jiangtao, Wang, Qilong, and Gao, Zilin. Towards faster training of global covariance pooling networks by iterative matrix square root normalization. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2018).

[82] Lim, Sanghun, Chandrasekar, Venkatachalam, and Bringi, Viswanathan N. Hydrometeor classification system using dual-polarization radar measurements: Model improvements and in situ verification. *IEEE transactions on geoscience and remote sensing 43*, 4 (2005), 792–801.

[83] Lin, Tsung-Yu, RoyChowdhury, Aruni, and Maji, Subhransu. Bilinear CNN Models for Fine-grained Visual Recognition. In *IEEE International Conference on Computer Vision (ICCV)* (2015).

[84] Lin, Tsung-Yu, RoyChowdhury, Aruni, and Maji, Subhransu. Bilinear convolutional neural networks for fine-grained visual recognition. *IEEE transactions on pattern analysis and machine intelligence 40*, 6 (2017), 1309–1322.

[85] Lin, Tsung-Yu, Winner, Kevin, Bernstein, Garrett, Mittal, Abhay, Dokter, Adriaan M, Horton, Kyle G, Nilsson, Cecilia, Van Doren, Benjamin M, Farnsworth, Andrew, La Sorte, Frank A, et al. Mistnet: Measuring historical bird migration in the us using archived weather radar data and convolutional neural networks. *Methods in Ecology and Evolution*.

[86] Liu, Jinlai, Yuan, Zehuan, and Wang, Changhu. Towards good practices for multi-modal fusion in large-scale video classification. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2018), pp. 0–0.

[87] Long, Jonathan, Shelhamer, Evan, and Darrell, Trevor. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)* (2015), pp. 3431–3440.

[88] Lorenz, Dominik, Bereska, Leonard, Milbich, Timo, and Ommer, Björn. Unsupervised part-based disentangling of object shape and appearance. *arXiv preprint arXiv:1903.06946* (2019).

[89] Lowe, D. G. Object recognition from local scale-invariant features. In *IEEE International Conference on Computer Vision (ICCV)* (1999).

[90] Lyapunov, Aleksandr Mikhailovich. The general problem of the stability of motion. *International Journal of Control 55*, 3 (1992), 531–534.

[91] Ma, Liqian, Jia, Xu, Sun, Qianru, Schiele, Bernt, Tuytelaars, Tinne, and Van Gool, Luc. Pose guided person image generation. In *Advances in Neural Information Processing Systems* (2017), pp. 406–416.

[92] Ma, Liqian, Sun, Qianru, Georgoulis, Stamatios, Van Gool, Luc, Schiele, Bernt, and Fritz, Mario. Disentangled person image generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), pp. 99–108.

[93] Magnus, Jan R, and Neudecker, Heinz. Matrix differential calculus with applications in statistics and econometrics. *Wiley series in probability and mathematical statistics* (1988).

[94] Mahendran, A., and Vedaldi, A. Visualizing deep convolutional neural networks using natural pre-images. *International Journal of Computer Vision (IJCV)* (2016).

[95] Maji, Subhransu, Rahtu, Esa, Kannala, Juho, Blaschko, Matthew, and Vedaldi, Andrea. Fine-grained visual classification of aircraft. *arXiv preprint arXiv:1306.5151* (2013).

[96] Mnih, Volodymyr, Heess, Nicolas, Graves, Alex, and Kavukcuoglu, Koray. Recurrent models of visual attention. In *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. 2014.

[97] Murray, Naila, Jégou, Hervé, Perronnin, Florent, and Zisserman, Andrew. Interferences in match kernels. *IEEE TPAMI 39*, 9 (2017), 1797–1810.

[98] Nilsson, Cecilia, Dokter, Adriaan M., Schmid, Baptiste, Scacco, Martina, Verlinden, Liesbeth, Bäckman, Johan, Haase, Günther, Dell'Omo, Giacomo, Chapman, Jason W., Leijnse, Hidde, and Liechti, Felix. Field validation of radar systems for monitoring bird migration. *Journal of Applied Ecology 55*, 6 (2018), 2552–2564.

[99] Nilsson, Cecilia, Dokter, Adriaan M., Verlinden, Liesbeth, Shamoun-Baranes, Judy, Schmid, Baptiste, Desmet, Peter, Bauer, Silke, Chapman, Jason, Alves, Jose A., Stepanian, Phillip M., Sapir, Nir, Wainwright, Charlotte, Boos, Mathieu, Górska, Anna, Menz, Myles H. M., Rodrigues, Pedro, Leijnse, Hidde, Zehtindjiev, Pavel, Brabant, Robin, Haase, Günther, Weisshaupt, Nadja, Ciach, Michał, and Liechti, Felix. Revealing patterns of nocturnal migration using the European weather radar network. *Ecography* (2018).

[100] Perronnin, F., and Dance, C. R. Fisher kernels on visual vocabularies for image categorization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2007).

[101] Perronnin, F., Sánchez, J., and Mensink, T. Improving the Fisher kernel for large-scale image classification. In *European Conference on Computer Vision (ECCV)* (2010).

[102] Perronnin, Florent, and Larlus, Diane. Fisher vectors meet neural networks: A hybrid classification architecture. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2015), pp. 3743–3752.

[103] Pham, Ninh, and Pagh, Rasmus. Fast and scalable polynomial kernels via explicit feature maps. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining* (2013), ACM, pp. 239–247.

[104] Popoviciu, Tiberiu. Sur les équations algébriques ayant toutes leurs racines réelles. *Mathematica 9* (1935), 129–145.

[105] Quattoni, A., and Torralba, A. Recognizing indoor scenes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2009).

[106] Razavin, A. Sharif, Azizpour, H., Sullivan, J., and Carlsson, S. CNN features off-the-shelf: An astounding baseline for recognition. In *DeepVision workshop* (2014).

[107] Roberts, John Douglas. Linear model reduction and solution of the algebraic riccati equation by use of the sign function. *International Journal of Control 32*, 4 (1980), 677–687.

[108] Russakovsky, Olga, Deng, Jia, Su, Hao, Krause, Jonathan, Satheesh, Sanjeev, Ma, Sean, Huang, Zhiheng, Karpathy, Andrej, Khosla, Aditya, Bernstein, Michael, Berg, Alexander C., and Fei-Fei, Li. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV) 115*, 3 (2015), 211–252.

[109] Scholkopf, Bernhard, and Smola, Alexander J. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

[110] Sharan, L., Rosenholtz, R., and Adelson, E. H. Material perceprion: What can you see in a brief glance? *Journal of Vision 9:784*, 8 (2009).

[111] Simon, Marcel, and Rodner, Erik. Neural activation constellations: Unsupervised part model discovery with convolutional networks. In *IEEE International Conference on Computer Vision (ICCV)* (2015).

[112] Simonyan, K., and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations* (2015).

[113] Simonyan, Karen, Vedaldi, Andrea, and Zisserman, Andrew. Deep fisher networks for large-scale image classification. In *Advances in neural information processing systems* (2013), pp. 163–171.

[114] Simonyan, Karen, and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)* (2015).

[115] Stepanian, Phillip M., Horton, Kyle G., Melnikov, Valery M., Zrnić, Dušan S., and Gauthreaux, Jr., Sidney A. Dual-polarization radar products for biological applications. *Ecosphere 7*, 11 (2016).

[116] Sun, Ming, Yuan, Yuchen, Zhou, Feng, and Ding, Errui. Multi-attention multi-class constraint for fine-grained image recognition. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2018), pp. 805–821.

[117] Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott, Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, and Rabinovich, Andrew. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015).

[118] Tuzel, Oncel, Porikli, Fatih, and Meer, Peter. Region covariance: A fast descriptor for detection and classification. In *European conference on computer vision* (2006), Springer, pp. 589–600.

[119] Tuzel, Oncel, Porikli, Fatih, and Meer, Peter. Pedestrian detection via classification on riemannian manifolds. *IEEE transactions on pattern analysis and machine intelligence 30*, 10 (2008), 1713–1727.

[120] Ulyanov, Dmitry, Lebedev, Vadim, Vedaldi, Andrea, and Lempitsky, Victor S. Texture networks: Feed-forward synthesis of textures and stylized images. In *ICML* (2016), vol. 1, p. 4.

[121] Ulyanov, Dmitry, Vedaldi, Andrea, and Lempitsky, Victor. Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2017), pp. 6924–6932.

[122] Van Horn, Grant, Branson, Steve, Farrell, Ryan, Haber, Scott, Barry, Jessie, Ipeirotis, Panos, Perona, Pietro, and Belongie, Serge. Building a bird recognition app and large scale dataset with citizen scientists: The fine print in fine-grained dataset collection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015).

[123] Vedaldi, Andrea, and Fulkerson, Brian. VLFeat: an open and portable library of computer vision algorithms. In *ACM International Conference on Multimedia* (2010).

[124] Wah, C., Branson, S., Welinder, P., Perona, P., and Belongie, S. The Caltech-UCSD Birds-200-2011 Dataset. Tech. Rep. CNS-TR-2011-001, CalTech, 2011.

[125] Wang, L., Guo, S., Huang, W., and Qiao, Y. Places205-VGGnet models for scene recognition. *CoRR abs/1508.01667* (2015).

[126] Wang, Yaming, Choi, Jonghyun, Morariu, Vlad, and Davis, Larry S. Mining discriminative triplets of patches for fine-grained classification. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016).

[127] Wang, Yang, Tran, Vinh, Bertasius, Gedas, Torresani, Lorenzo, and Hoai, Minh. Attentive action and context factorization. *arXiv preprint arXiv:1904.05410* (2019).

[128] Wertheimer, Davis, and Hariharan, Bharath. Few-shot learning with localization in realistic settings. In *Computer Vision and Pattern Recognition (CVPR)* (2019).

[129] Winkler, David W. Roosts and migrations of swallows. *Hornero 21*, 2 (2006), 85–97.

[130] Yu, Zhou, Yu, Jun, Fan, Jianping, and Tao, Dacheng. Multi-modal factorized bilinear pooling with co-attention learning for visual question answering. In *Proceedings of the IEEE international conference on computer vision* (2017), pp. 1821–1830.

[131] Zeiler, Matthew D, and Fergus, Rob. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision* (2014).

[132] Zhang, Han, Xu, Tao, Elhoseiny, Mohamed, Huang, Xiaolei, Zhang, Shaoting, Elgammal, Ahmed, and Metaxas, Dimitris. SPDA-CNN: unifying semantic part detection and abstraction for fine-grained recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016).

[133] Zhang, N., Donahue, J., Girshick, R., and Darrell, T. Part-based R-CNNs for fine-grained category detection. In *European Conference on Computer Vision (ECCV)* (2014).

[134] Zhang, Ning, Farrell, Ryan, and Darrell, Trevor. Pose pooling kernels for subcategory recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2012).

[135] Zhang, Ning, Paluri, Manohar, Rantazo, Marc'Aurelio, Darrell, Trevor, and Bourdev, Lubomir. Panda: Pose aligned networks for deep attribute modeling. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2014).

[136] Zhang, Ning, Shelhamer, Evan, Gao, Yang, and Darrell, Trevor. Fine-grained pose prediction, normalization, and recognition. *arXiv preprint arXiv:1511.07063* (2015).

[137] Zhang, Ning, Shelhamer, Evan, Gao, Yang, and Darrell, Trevor. Fine-grained pose prediction, normalization, and recognition. In *Workshop at International Conference on Learning Representations* (2016).

[138] Zheng, Heliang, Fu, Jianlong, Mei, Tao, and Luo, Jiebo. Learning multi-attention convolutional neural network for fine-grained image recognition. In *The IEEE International Conference on Computer Vision (ICCV)* (Oct 2017).

[139] Zheng, Heliang, Fu, Jianlong, Zha, Zheng-Jun, and Luo, Jiebo. Looking for the devil in the details: Learning trilinear attention sampling network for fine-grained image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2019).

[140] Zuffi, Silvia, Kanazawa, Angjoo, Berger-Wolf, Tanya, and Black, Michael J. Three-d safari: Learning to estimate zebra pose, shape, and texture from images "in the wild". In *The IEEE International Conference on Computer Vision (ICCV)* (October 2019).