Improving second-order representations

“better, faster, stronger”

Subhransu Maji

College of Information and Computer Sciences

University of Massachusetts, Amherst

Talk outline

Motivation and background

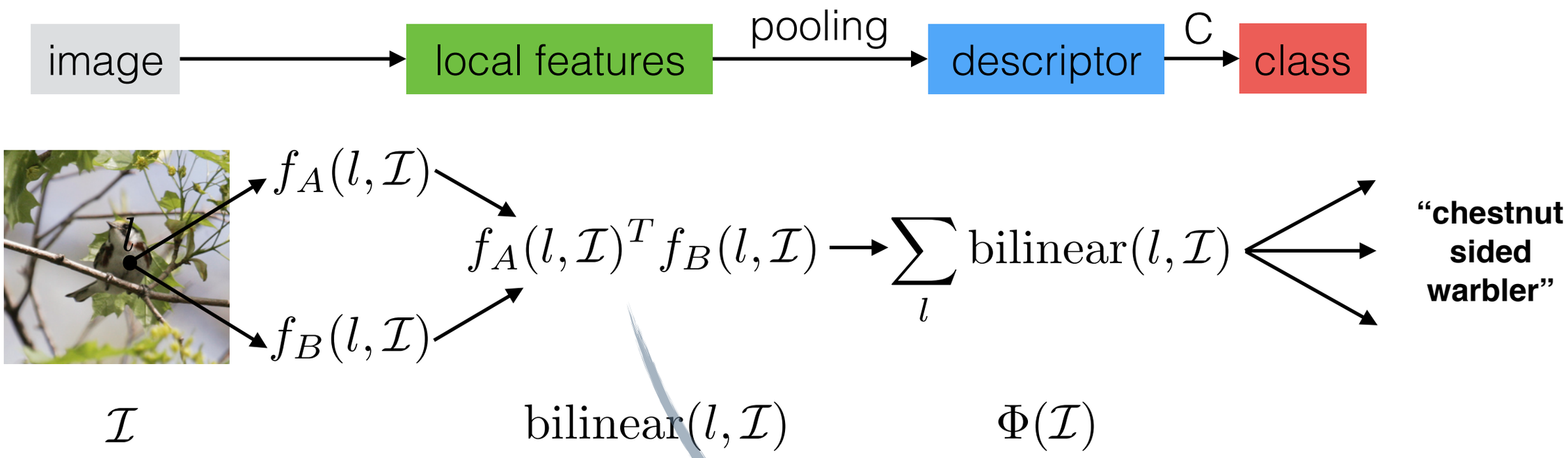
Improving their robustness and efficiency

- spectral normalization and democratic pooling

What do these models learn?

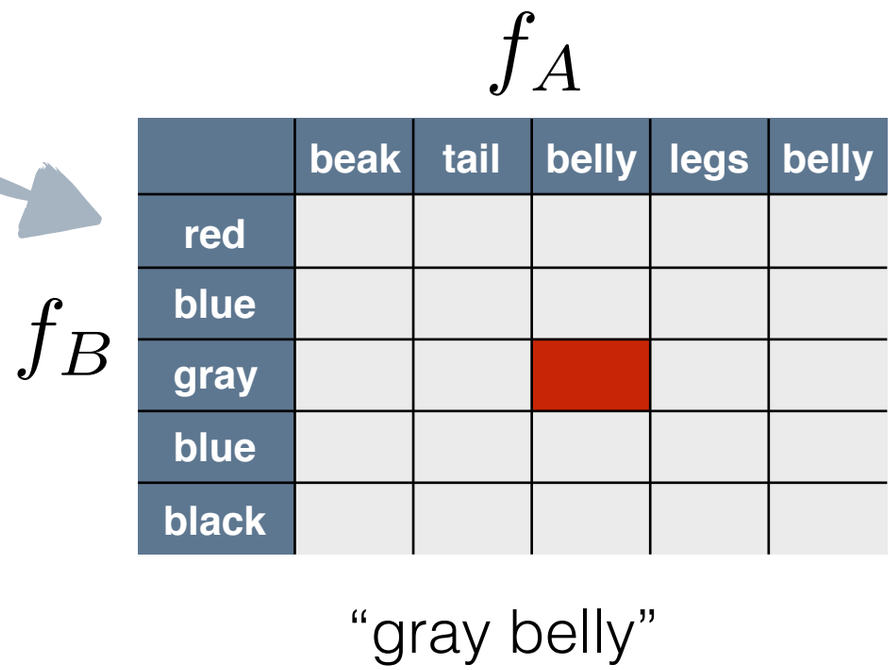
Bilinear (second-order) pooling

CNN activations pooled after outer-product encoding



Generalizes texture encoders

- Fisher vectors, Bag of Visual Words, VLAD
- Gram-matrix (when f_A=f_B)
- Excellent transfer from ImageNet to fine-grained domains (e.g., birds, cars, airplanes)



Lin et al., Bilinear CNNs for Fine-grained Visual Recognition, ICCV 15, PAMI 17

Bilinear (second-order) pooling

Classical texture encoders are bilinear $f_A \otimes f_B$

- **Bag of Visual Words (BoVW)** [Sivic and Zisserman 03, Csurka et al. 04]

$$f_A(\mathbf{x}) = \eta(\mathbf{x}) \qquad f_B(\mathbf{x}) = 1$$

"hard" or "soft" assignment

- **Vector of Locally Aggregated Descriptors (VLAD)** [Jegou et al. 10]

$$f_A(\mathbf{x}) = \eta(\mathbf{x}) \qquad f_B(\mathbf{x}) = [\mathbf{x} - \mu_1; \mathbf{x} - \mu_2; \ldots; \mathbf{x} - \mu_k]$$

"hard" assignment — first order residuals

- **Fisher vectors (FV)** [Perronnin et al. 10]

$$f_A(\mathbf{x}) = \eta(\mathbf{x}) \qquad f_B(\mathbf{x}) = [\alpha_1\beta_1; \alpha_2\beta_2; \ldots; \alpha_k\beta_k]$$

$$\alpha_i = \Sigma_i^{-\frac{1}{2}}(\mathbf{x} - \mu_i)$$
$$\beta_i = \Sigma_i^{-1}(\mathbf{x}-\mu_i) \odot (\mathbf{x}-\mu_i) - 1$$

"soft" assignment — first and second order residuals

Bilinear (second-order) pooling

Fine-grained classification (VGG-D + VGG-M networks)

CUB 200-2011
200 species, 11,788 images

FGVC Aircraft
100 variants, 10,000 images

Stanford cars
196 models, 16,185 images

Method	Birds	Aircraft	Cars
FC [D]	70.4	76.6	79.8
FV [D]	74.7	78.7	85.7
Bilinear [D+D]	**84.0**	**83.9**	**90.6**
Bilinear [D+M]	**84.1**	**84.5**	**91.3**
Previous work	**84.1** [1]	**80.7** [2]	**92.6** [3]

Method	NABirds
Inception-BN	**73.1** [4]
B-CNN [D+M]	**79.4**

48,562 images of 555 categories

[1] Spatial Transformer Networks, Jaderberg et al., NIPS 15
[2] Revisiting the Fisher vector for Fine-grained Classification, Gosselin et al., PR Letters 14
[3] Fine-Grained Rec. w/o Part Annotations, Krause et al., CVPR 15
[4] Batch-normalized Inception Architectures, Szegedy et al., CVPR 15

Talk outline

Motivation and background

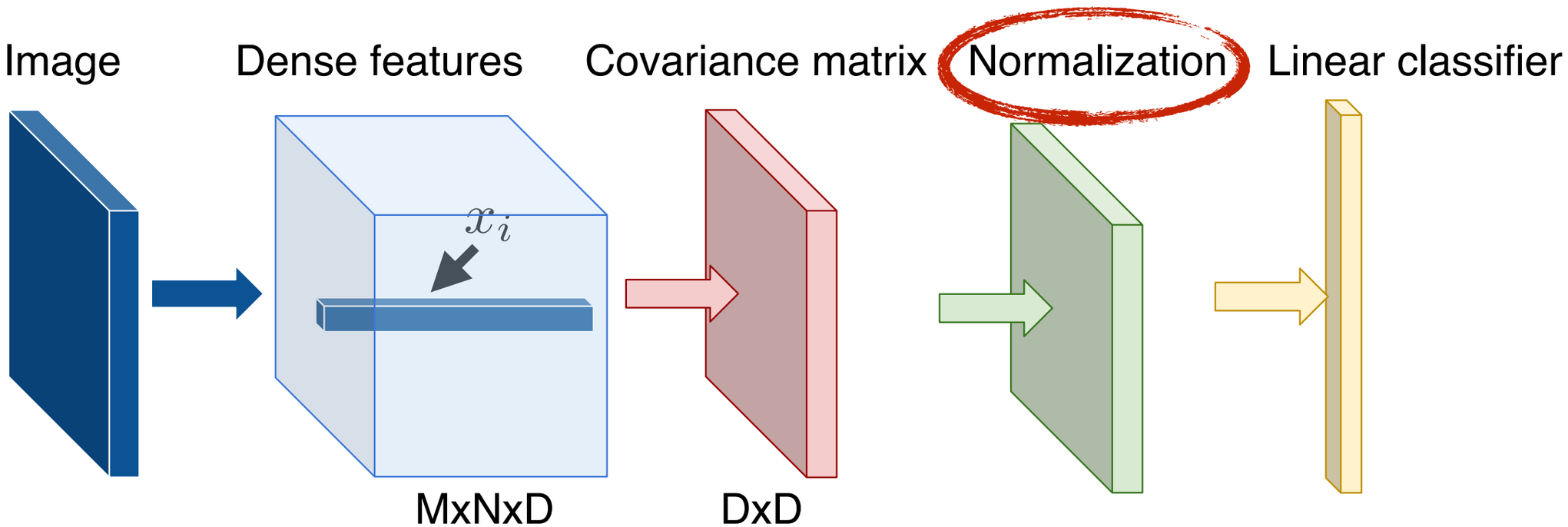
Improving their robustness and efficiency

- spectral normalization and democratic normalization

What do these models learn?

Improved bilinear CNNs

Covariance (non-centered) matrix as an image representation



Symmetric Positive Definite

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

Normalization is critical for performance

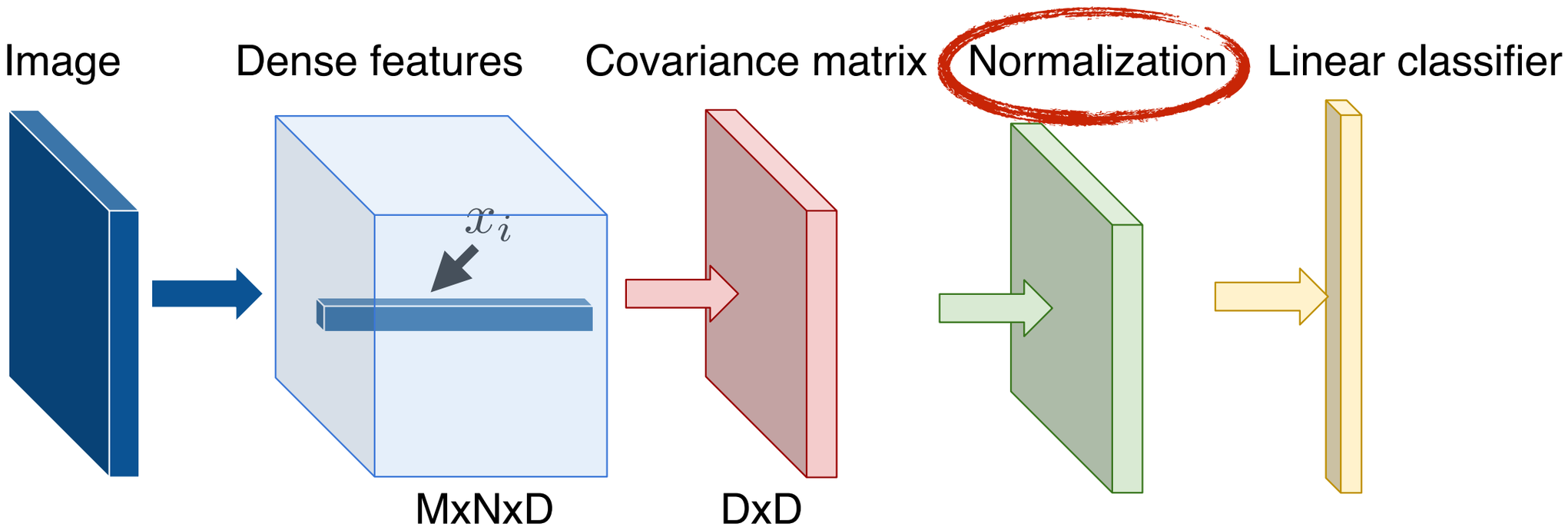
- **Element wise**: l2norm, signed square-root

2-4% improvement

Lin and Maji, Improved Bilinear Pooling with CNNs, BMVC 17

Improved bilinear CNNs

Covariance (non-centered) matrix as an image representation



Normalization is critical for performance

- **Element wise**: l2norm, signed square-root
- **Matrix normalization**:
  - Log-Euclidean [Carreira et al., 12]
  - Matrix power (0 < p < 1)
  - Whitening

Symmetric Positive Definite

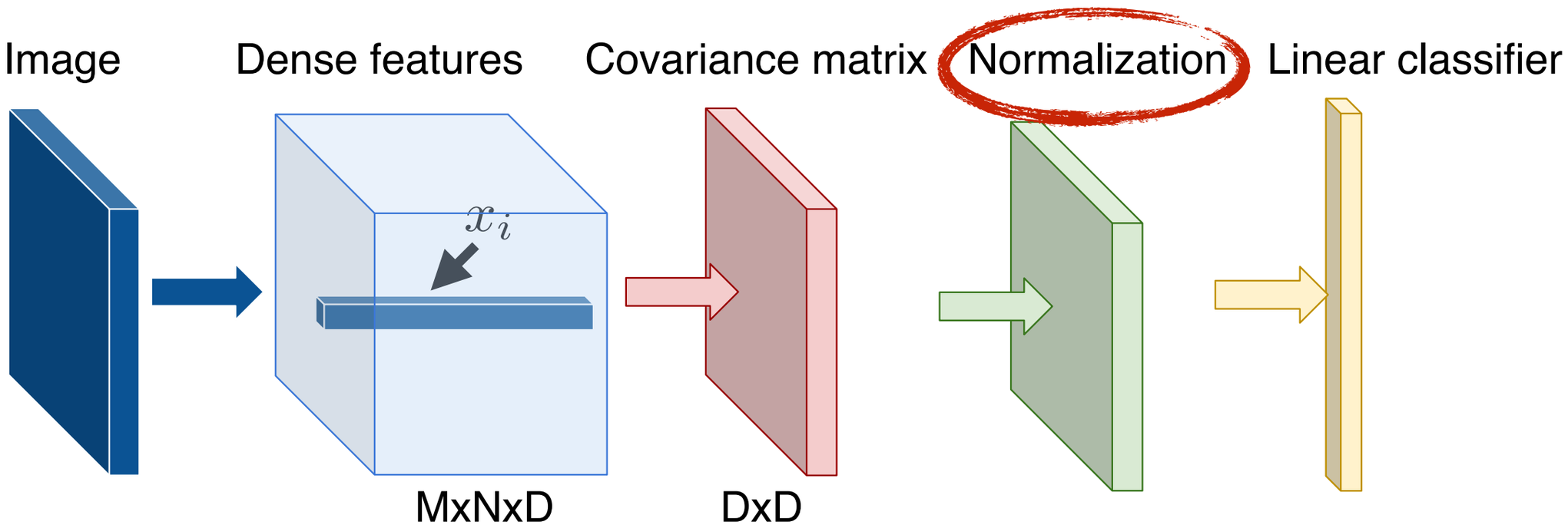
$$A = \frac{1}{n}\left(\sum_{i=1}^{n} x_i x_i^T\right) + \epsilon I$$

$$log(A) = U \begin{bmatrix} log(\sigma_1) & 0 & \cdots & 0 \\ 0 & log(\sigma_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & log(\sigma_d) \end{bmatrix} U^T$$

Lin and Maji, Improved Bilinear Pooling with CNNs, BMVC 17

Improved bilinear CNNs

Covariance (non-centered) matrix as an image representation



Normalization is critical for performance

- **Element wise**: l2norm, signed square-root
- **Matrix normalization**:
  - Log-Euclidean [Carreira et al., 12]
  - Matrix power (0 < p < 1)
  - Whitening

Symmetric Positive Definite

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

$$A^p = U \begin{bmatrix} \sigma_1^p & 0 & \cdots & 0 \\ 0 & \sigma_2^p & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_d^p \end{bmatrix} U^T$$

Lin and Maji, Improved Bilinear Pooling with CNNs, BMVC 17

Improved bilinear CNNs

On the burstiness of visual elements

Hervé Jégou Matthijs Douze Cordelia Schmid
INRIA Grenoble, LJK
firstname.lastname@inria.fr

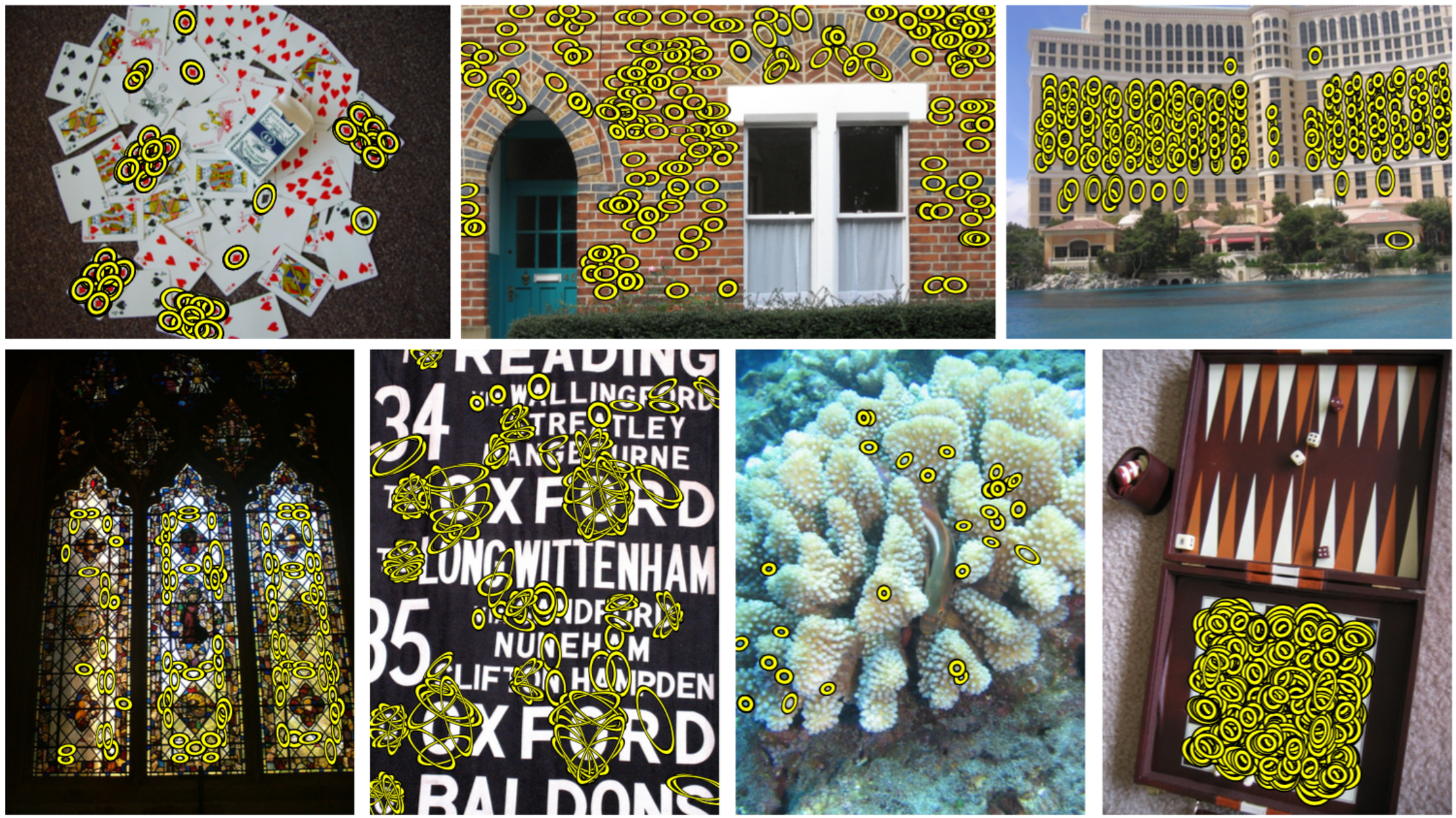
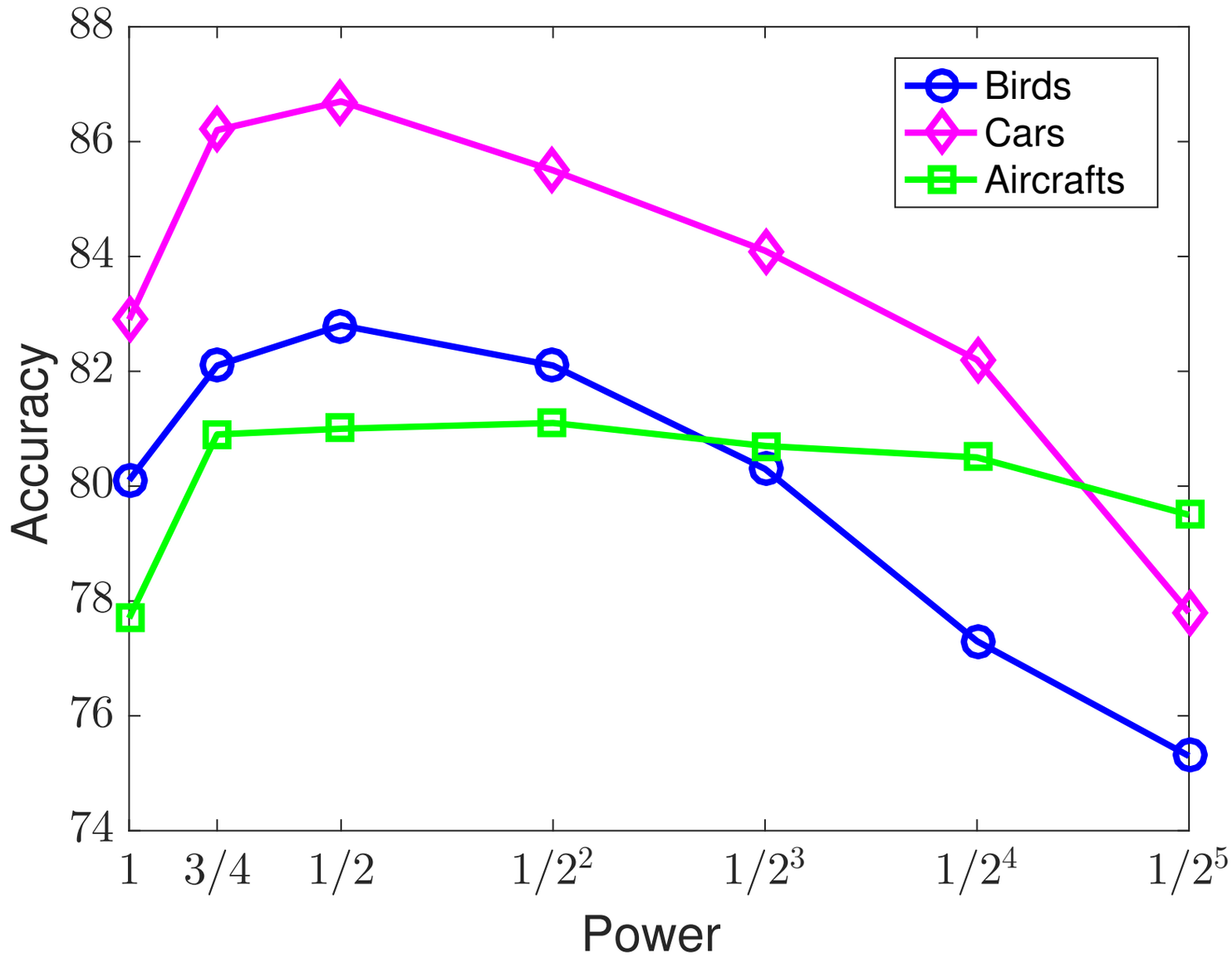


Figure 1. Illustration of burstiness. Features assigned to the most "bursty" visual word of each image are displayed.

Improved bilinear CNNs

Effect of the exponent in the power normalization



$$A \rightarrow A^p$$

p = 1/2 is nearly optimal

encodes scale invariance?

Improved bilinear CNNs

Relative effects of different normalizations

L2 normalization is added at the end for all methods

Base network	Normalizations			Accuracy		
	log(A)	$A^{1/2}$	*sgnsqrt(A)*	Birds	Aircrafts	Cars
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

Spectral normalization offers complementary benefits

Related work: Ionescu et al. 15, Li et al. 17, Huang and Van Gool 17

Improved bilinear CNNs

Why are matrix normalization layers not common in deep architectures?

- **Forward computation: matrix normalization via SVD is slow**

  Lacks an efficient batch-mode GPU implementation: SVD of a 512x512 matrix takes the same time as the rest of the VGG-16 network evaluation!

  CPU versions are sometimes faster, but incur copying overhead

- **Backward computation: gradients of matrix normalization layers**

  Non-trivial to compute manually

  Numerical precision can be an issue

Can we make forward and backward faster?

- **Yes, for the matrix square-root case!**

Improved bilinear CNNs

Matrix square-root on the GPU

Forward via Newton iterations ($X^2 - A = 0$)

- **Newton-Schulz iterations:** Initialize $Y_k=A$, $Z_k=I$ then Y_k quadratically to $A^{1/2}$ and $A^{-1/2}$ respectively (after scaling)

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

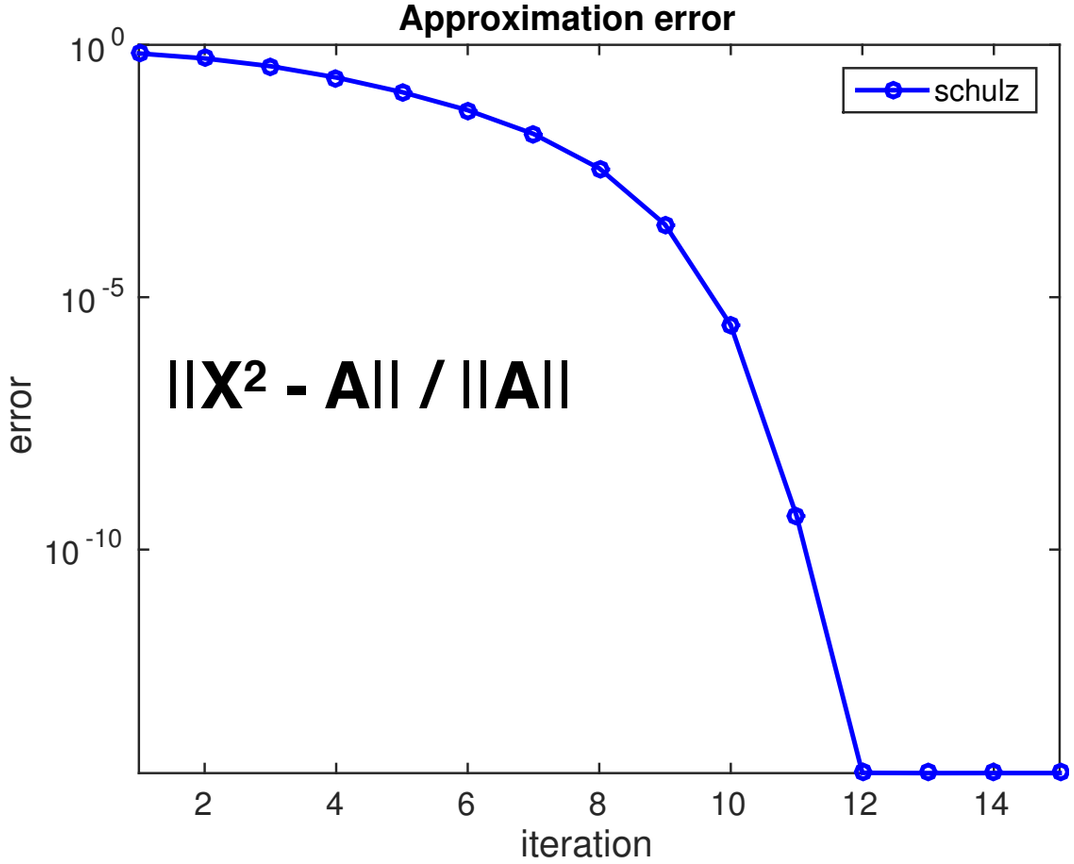
- Only matrix multiplications and additions

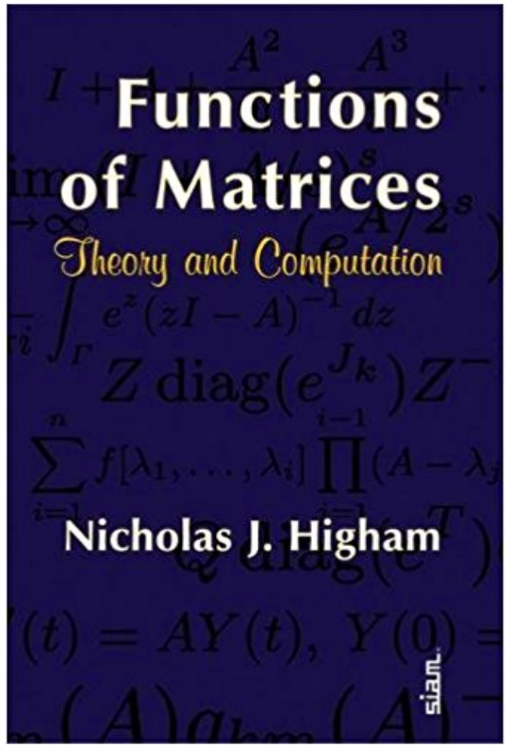
Babylonian method

$$x_0 \approx \sqrt{S},$$

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{S}{x_n}\right),$$

$$\sqrt{S} = \lim_{n\to\infty} x_n.$$





Denman-Beavers

$$Y_{k+1} = \frac{1}{2}(Y_k + Z_k^{-1}),$$

$$Z_{k+1} = \frac{1}{2}(Z_k + Y_k^{-1}).$$

Lin and Maji, Improved Bilinear Pooling with CNNs, BMVC 17

Improved bilinear CNNs

Matrix square-root on the GPU

Forward via Newton iterations ($X^2 - A = 0$)

- **Newton-Schulz iterations:** Initialize $Y_k=A$, $Z_k=I$ then Y_k and Z_k converge quadratically to $A^{1/2}$ and $A^{-1/2}$ respectively (after scaling)

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

- Only matrix multiplications and additions

Iteration	0	1	**5**	10	20	SVD
Birds	80.1	81.7	**83.0**	82.9	82.8	82.8
Cars	82.9	85.0	**87.0**	86.8	86.7	86.7
Aircrafts	77.7	79.5	**81.3**	80.9	80.9	80.9
Time	0ms	1ms	**4ms**	6ms	11ms	22ms

5x faster on Matlab, 30x faster on PyTorch

Improved bilinear CNNs

Matrix square-root on the GPU

Backward evaluation

- **Matrix function gradients via SVD** [Magnus and Neudecker 88, Ionescu et al. 15]

  Given a matrix A = U Σ V and function Z = f(A) = U g(Σ) V, the gradients satisfy

$$\frac{\partial L}{\partial U} = \left\{ \frac{\partial L}{\partial Z} + \left(\frac{\partial L}{\partial Z} \right)^T \right\} U g(\Sigma), \quad \frac{\partial L}{\partial \Sigma} = g'(\Sigma) U^T \frac{\partial L}{\partial Z} U$$

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

$$K_{i,j} = 1/(\sigma_i - \sigma_j) \mathbf{I}(i \neq j)$$

Poor numerical stability (adding a positive diagonal does not help!)

Improved bilinear CNNs

Matrix square-root on the GPU

Backward evaluation

- **Matrix function gradients via SVD** [Magnus and Neudecker 88, Ionescu et al. 15]

  Poor numerical stability (adding a positive diagonal does not help!)

- **Matrix function gradients via solving a Lyapunov equation**

  The gradient of a matrix square-root satisfies a Lyapunov equation

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

Numerical stability depends on $1/(\sigma_i+\sigma_j)$

$O(n^3)$ solvers via SVD, Bartels-Stewart algorithm

Improved bilinear CNNs

Matrix square-root on the GPU

SVD vs Lyapunov gradients

- **Lyapunov gradients lead to better accuracy (e.g., Aircrafts)**

network	gradient	normalization		accuracy		
		$A^{1/2}$	$sgnsqrt(A)$	birds	cars	aircrafts
VGG-D			✓	84.0	90.6	86.9
	Lyapunov	✓	✓	**85.8**	**92.0**	**88.5**
	SVD	✓	✓	85.5	91.8	86.8

- **What about matrix logarithm?**

  We couldn't get fine-tuning to converge with SVD gradients. Numerical stability is a major issue.

  Possible solution: iterative square-root and scaling $\log(A) = 2^k \log(A^{1/2^k})$

Improved bilinear CNNs

Gradients of the matrix square root layer

Automatic differentiation

- **Implement the iterations in a library that supports auto-diff (e.g., pytorch)**

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

- **Gradients are almost free!**

  Additional 2% of forward time for 512x512 matrices (10 iterations)

  100x faster than gradients via SVD or Bartels-Stewart

- **Memory overhead?**

  A naive implementation checkpoints **all** the intermediate values

  Solution: run it for a few iterations

Improved bilinear CNNs

Gradients of the matrix square root layer

Gradients via iterative Lyapunov solver

- **Iterative solver for the Lyapunov gradients**

initialize:
$$Z = A^{\frac{1}{2}}; \quad A_0 = Z; \quad Q_0 = \frac{dL}{dZ}$$

iterate:
$$Q_{k+1} = \frac{1}{2}\left(Q_k(3I - A_k^2) - A_k^T(A_k^T Q_k - Q_k A_k)\right)$$
$$A_{k+1} = \frac{1}{2}\left(A_k(3I - A_k A_k)\right)$$

return:
$$\frac{dL}{dA} = \frac{1}{2} Q_{k+1}$$

- **No memory overhead!**

In theory, this works if the forward was run to convergence.

Improved bilinear CNNs

Accuracy with spectral normalization layers

Method	Base network	Accuracy		
		Birds	Cars	Aircrafts
B-CNN [PAMI 16]	VGG-D	84.0	90.6	86.9
Improved B-CNN [BMVC 17]	VGG-D	**85.8**	**92.0**	**88.5**
CBP [CVPR 16]	VGG-D	84.0	-	-
LRBP [CVPR 17]	VGG-D	84.2	90.9	86.9
BoostCNN [BMVC 16]	B-CNN	**86.2**	92.1	**88.5**
Kernel Pooling [CVPR 17]	VGG-D	**86.2**	92.4	86.9
STN [NIPS 15]	Inception-BN	84.1	-	**-**
Krause et al. [CVPR 15]	VGG-D + box	-	**92.6**	**-**

Improved B-CNN matches the new state of the art (ImageNet1K pre-training but no additional training data except labelled images of the dataset)

- **CBP**: Compact bilinear pooling (10x reduction in the classifier size)
- **LRBP**: Low-rank bilinear pooling (10-100x reduction in the classifier size)
- **BoostCNN**: Boosts B-CNNs at several scales (10-50x slower)
- **Kernel Pooling**: Pooling higher-order interactions

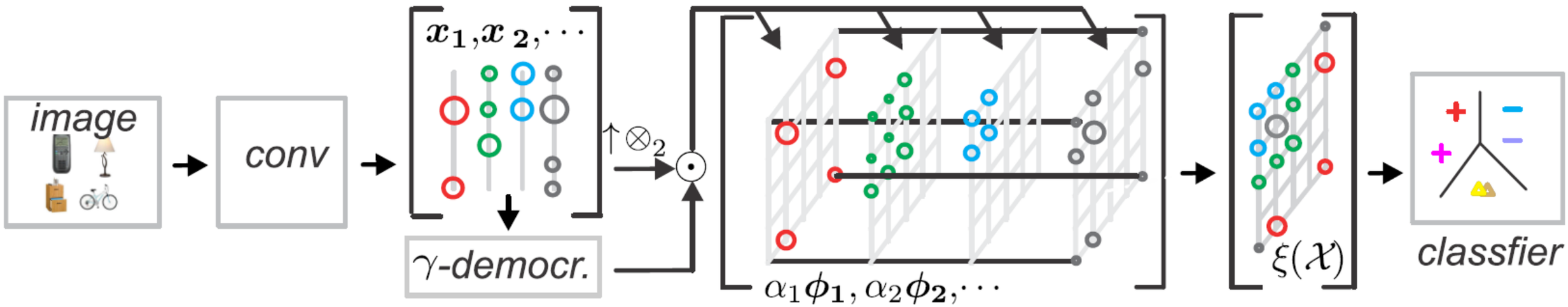
Improved bilinear CNNs

Using DenseNet-201 [Huang et al., CVPR 17]

Dataset	#images	#classes	DenseNet (regular)	DenseNet (bilinear)
FGVC Aircrafts	6,667	100	**92.3**	90.6
iNat Plantae	118,800	2,917	**69.6**	66.9
iNat Animalia	5,966	178	**80.5**	79.0
Stanford Cars	8,144	196	**93.2**	92.9
iNat Reptilia	22,754	284	**53.3**	51.6
iNat Amphibia	11,156	144	55.6	**56.9**
iNat Aves	143,950	1,258	60.1	**62.0**
iNat Mollusca	8,007	262	73.5	**75.1**
iNat Fungi	6,864	321	70.5	**72.8**
iNat Mammalia	20,104	234	60.1	**63.3**
iNat Arachnida	4,037	114	66.4	**71.3**
iNat Insecta	87,192	2,031	78.2	**82.0**
CU Birds	5994	200	84.9	**88.3**
iNat Actinopterygii	7,835	369	76.5	**82.7**

Second-order democratic aggregation

Democratic aggregation [Murray et al., PAMI 17]



Democratic aggregation [Murray et al.] computes a weighted combination of features:

$\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\}$ encoded by $\phi(\mathbf{x}) = \mathbf{x}\mathbf{x}^T$

$$\xi(\mathcal{X}) = \sum_{\mathbf{x} \in \mathcal{X}} \alpha(\mathbf{x})\phi(\mathbf{x})$$

such that $\alpha(\mathbf{x}) > 0$ and satisfy:

$$\alpha(\mathbf{x})\phi(\mathbf{x})^T \xi(\mathcal{X}) = C, \quad \forall \mathbf{x} \in \mathcal{X}$$

γ-Democratic aggregation $\quad \mathtt{diag}(\alpha)\mathbf{K}\mathtt{diag}(\alpha)\mathbf{1}_n = (\mathbf{K}\mathbf{1}_n)^\gamma$

Lin, Maji, Konuisz, Second-order democratic aggregation, ECCV 18

Matrix square-root is not linear!

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.

Second-order democratic aggregation

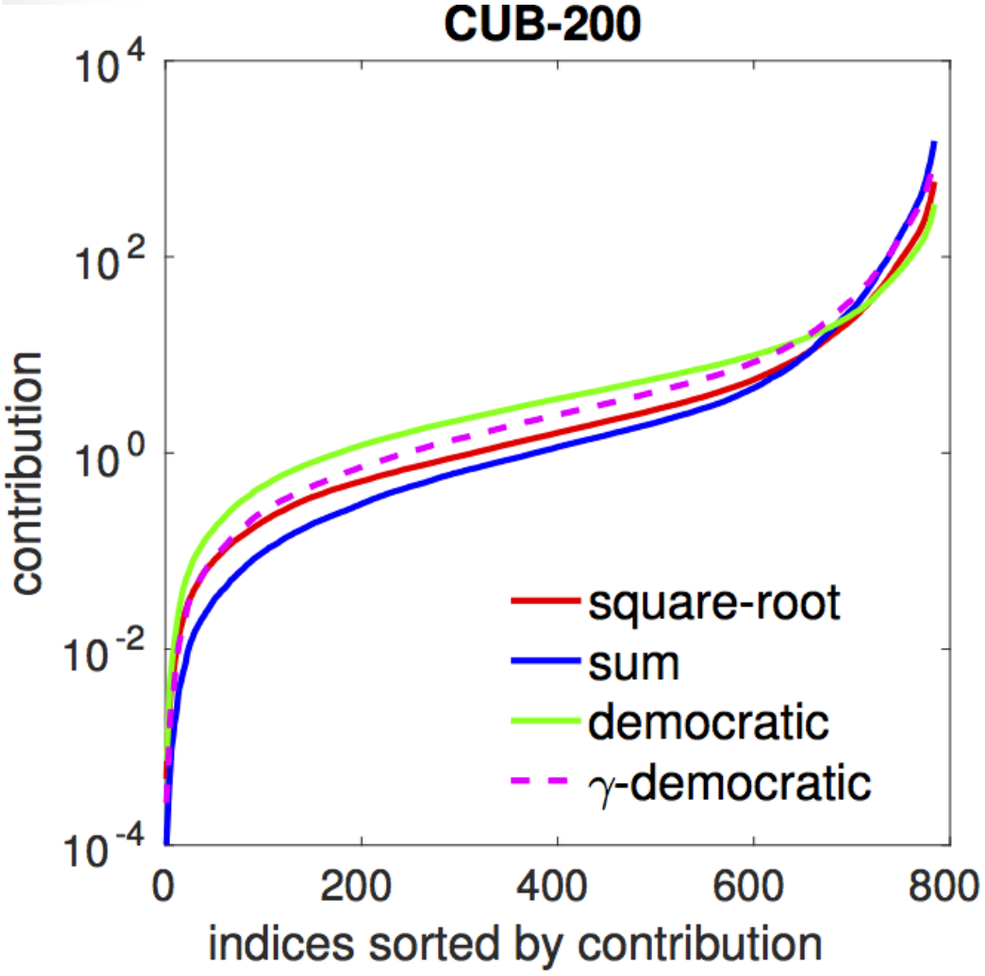
Relationship between spectral normalization and democratic pooling

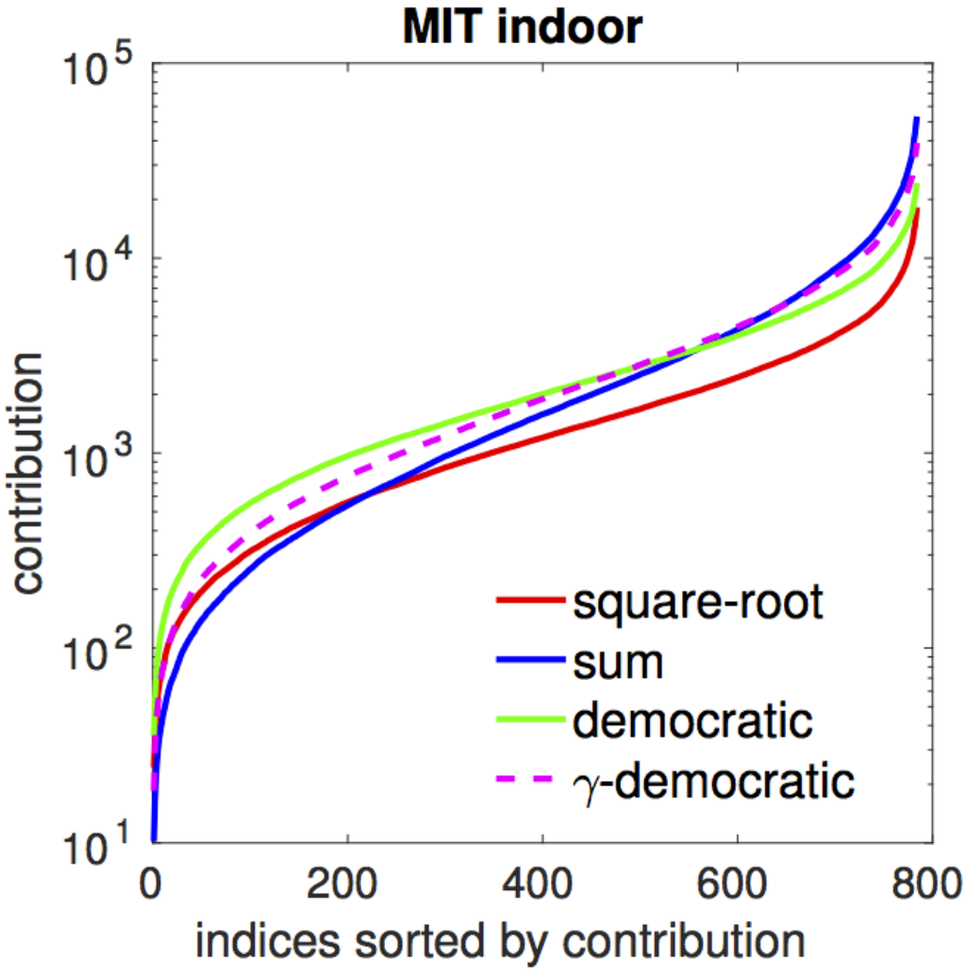
Equalizing feature contributions. While seemingly different, both democratic aggregation and power normalization aim at equalizing contributions among features. Formally lets define the contribution of a feature $\mathbf{x}$ as $C(\mathbf{x})$ given as

$$C(\mathbf{x}) = \phi(\mathbf{x})^T \xi(\mathcal{X}).$$

For democratic aggregation $C(\mathbf{x}) = 1/\alpha(\mathbf{x})$. In the paper we show that the variance of the contributions for the matrix normalization with a power p satisfies

$$\sigma^2 < \frac{r_{\max}^2 \lambda_1^{2p}}{4 \sum_i \lambda_i^{2p}},$$





Second-order democratic aggregation

Democratic vs. Spectral

Democratic aggregation is faster than spectral normalization

- **O(n^2)** vs. **O(n^3)** per iteration

Algorithm 1 Dampened Sinkhorn Algorithm

```
procedure SINKHORN(K, τ, T)
    α ← 1_n
    for t = 1 to T do
        σ = diag(α)K diag(α)1_n
        α ← α/σ^τ
    return α
```

But, it's performance is worse

Dataset	Sum	Democratic	Spectral
Caltech UCSD Birds	84.0	84.9	85.9
Stanford Cars	90.6	90.8	91.7
FGVC Aircrafts	85.7	86.7	87.6
DTD	71.2	72.3	72.9
FMD	84.6	84.6	85.0
MIT Indoor	79.5	80.4	80.9

Lin, Maji, Konuisz, Second-order democratic aggregation, ECCV 18

Talk outline

Motivation and background

Improving their robustness and efficiency

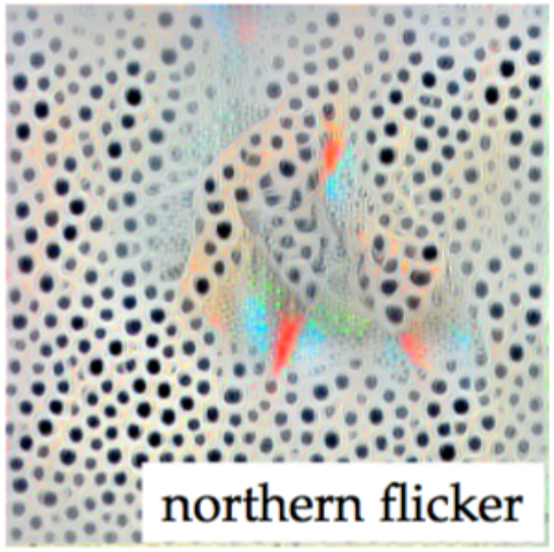
- spectral normalization and democratic normalization

What do these models learn?

Visualizing deep networks

“inverse” images for bilinear CNNs

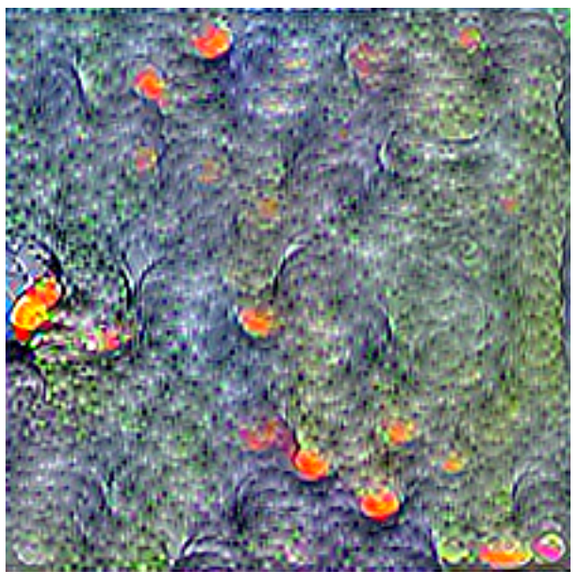
Maximal images: $\arg\max_{\mathcal{I}} \log P(c|\mathcal{I}, \mathbf{W}) + \log \Gamma(\mathcal{I})$



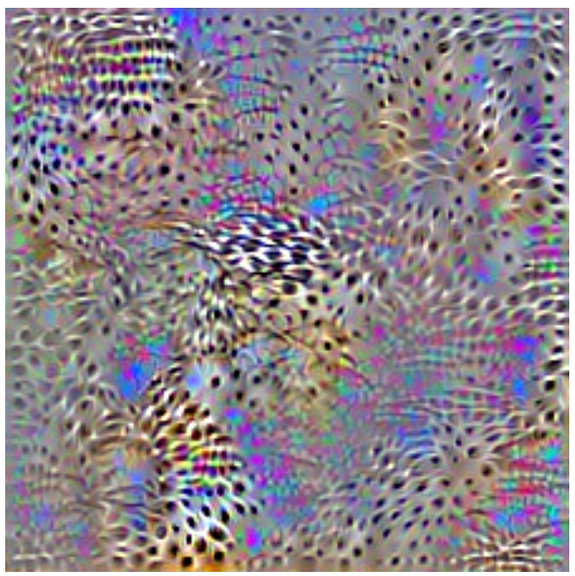
Lin and Maji, Visualizing and Understanding Deep Texture Representations, CVPR 16

Visualizing deep networks

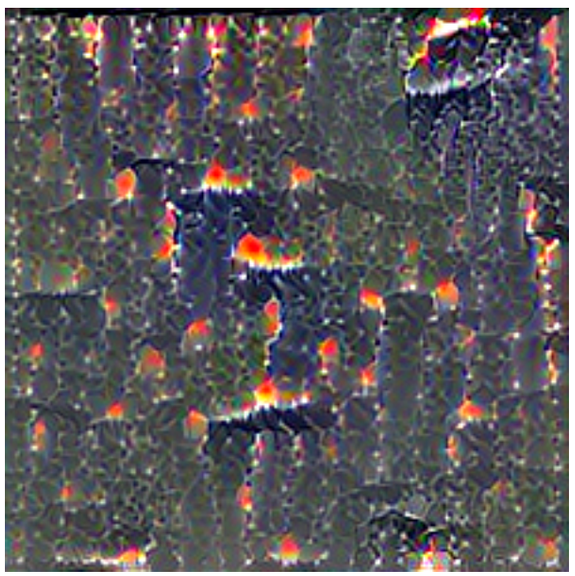
What texture are birds?



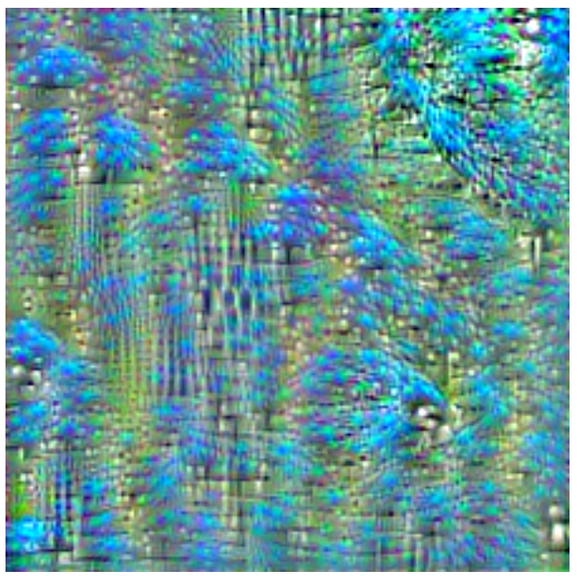
Crested aucket



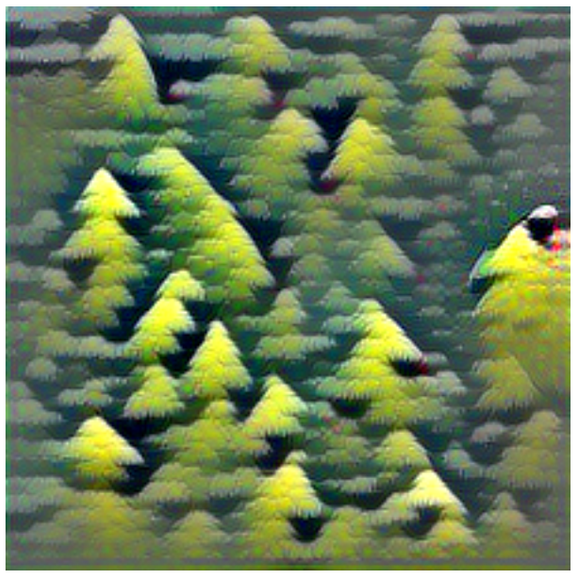
Cactus wren



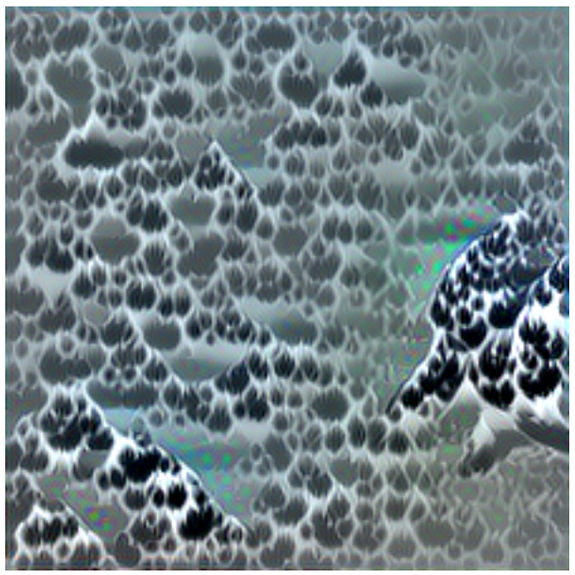
Red winged blackbird



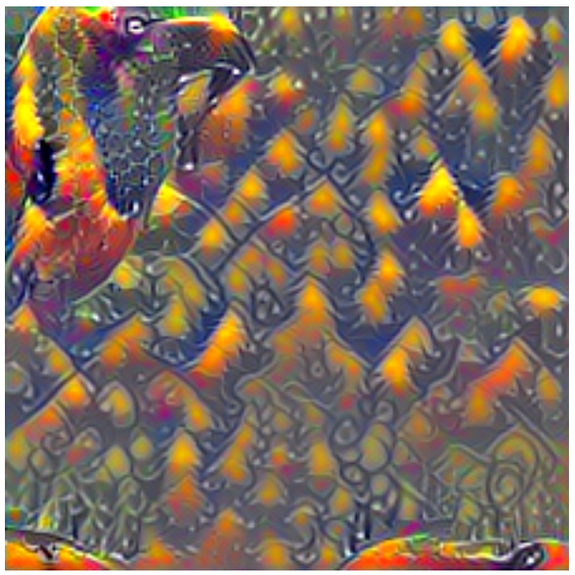
Indigo bunting



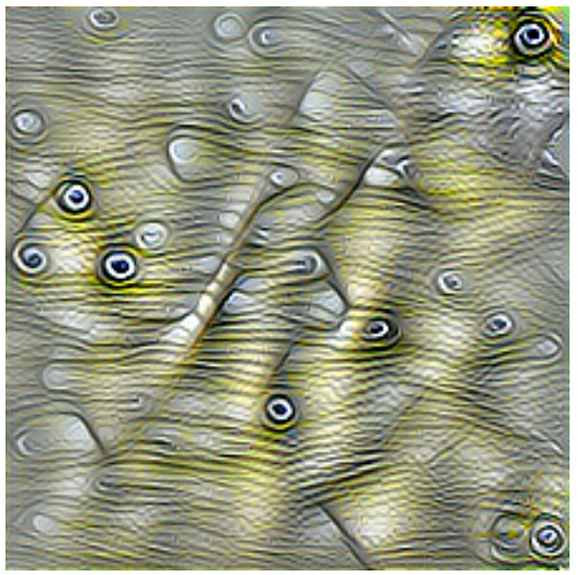
American goldfinch



Pied kingfisher



Hooded oriole



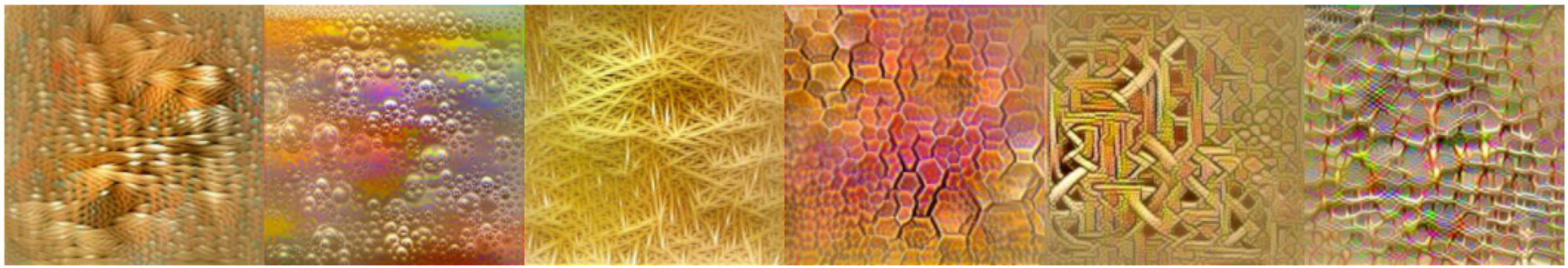
White eyed vireo

Lin and Maji, Visualizing and Understanding Deep Texture Representations, CVPR 16

Visualizing deep networks

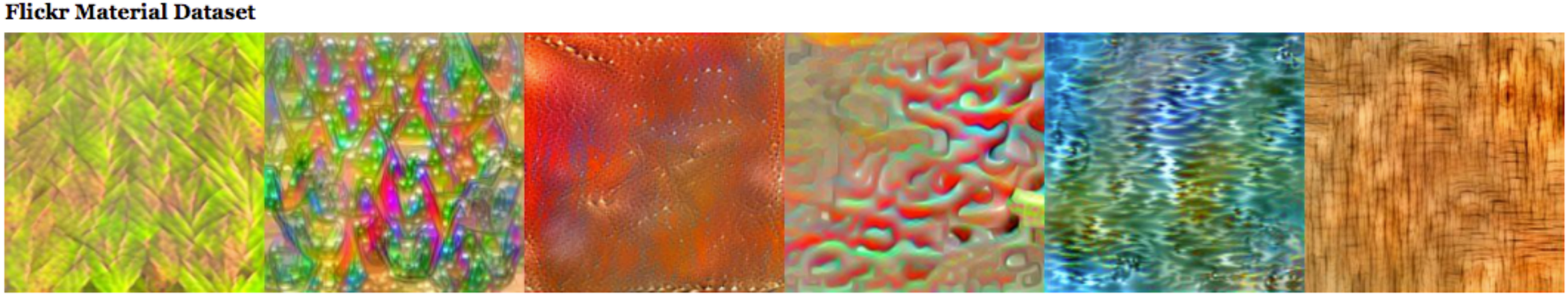
What texture are bookstores?

Describable Texture Datatset



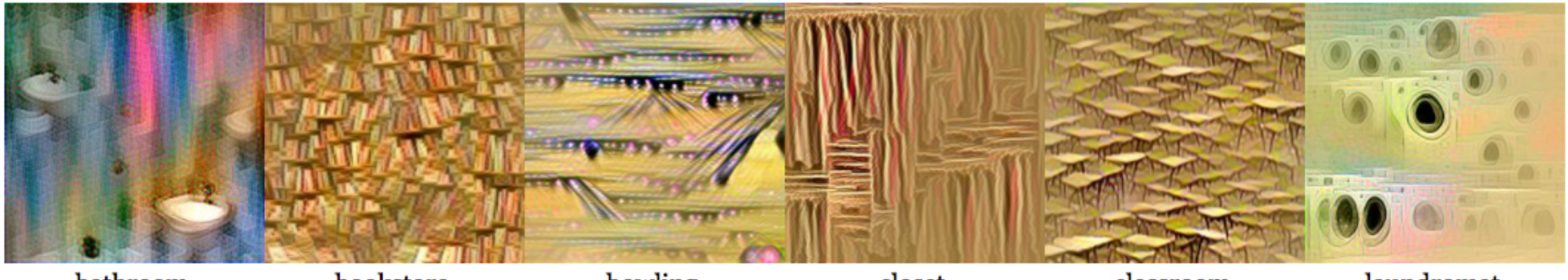
braided | bubbly | fibrous | honeycombed | interlaced | meshed

Flickr Material Dataset



foliage | glass | leather | plastic | water | wood

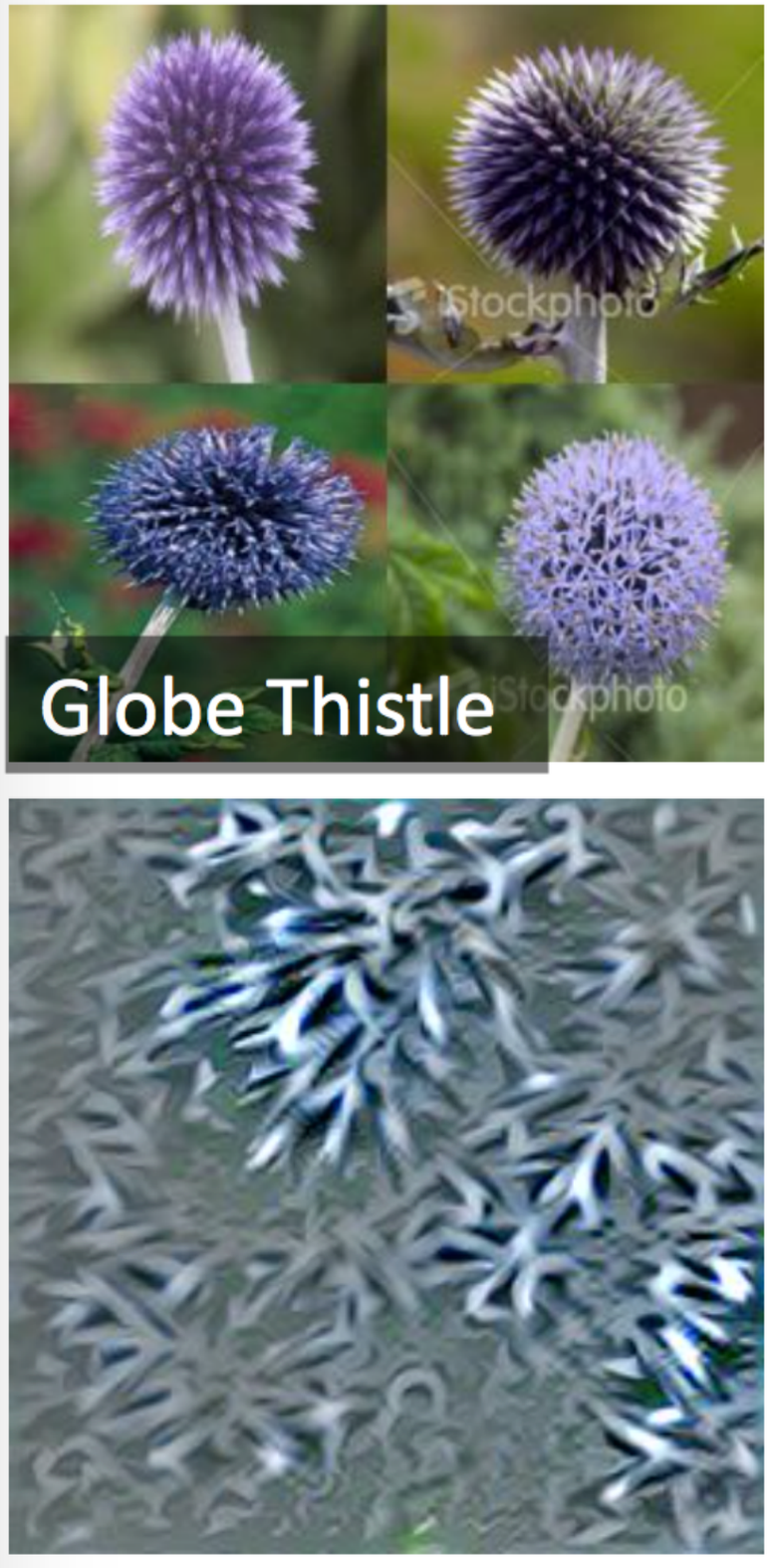
MIT Indoor

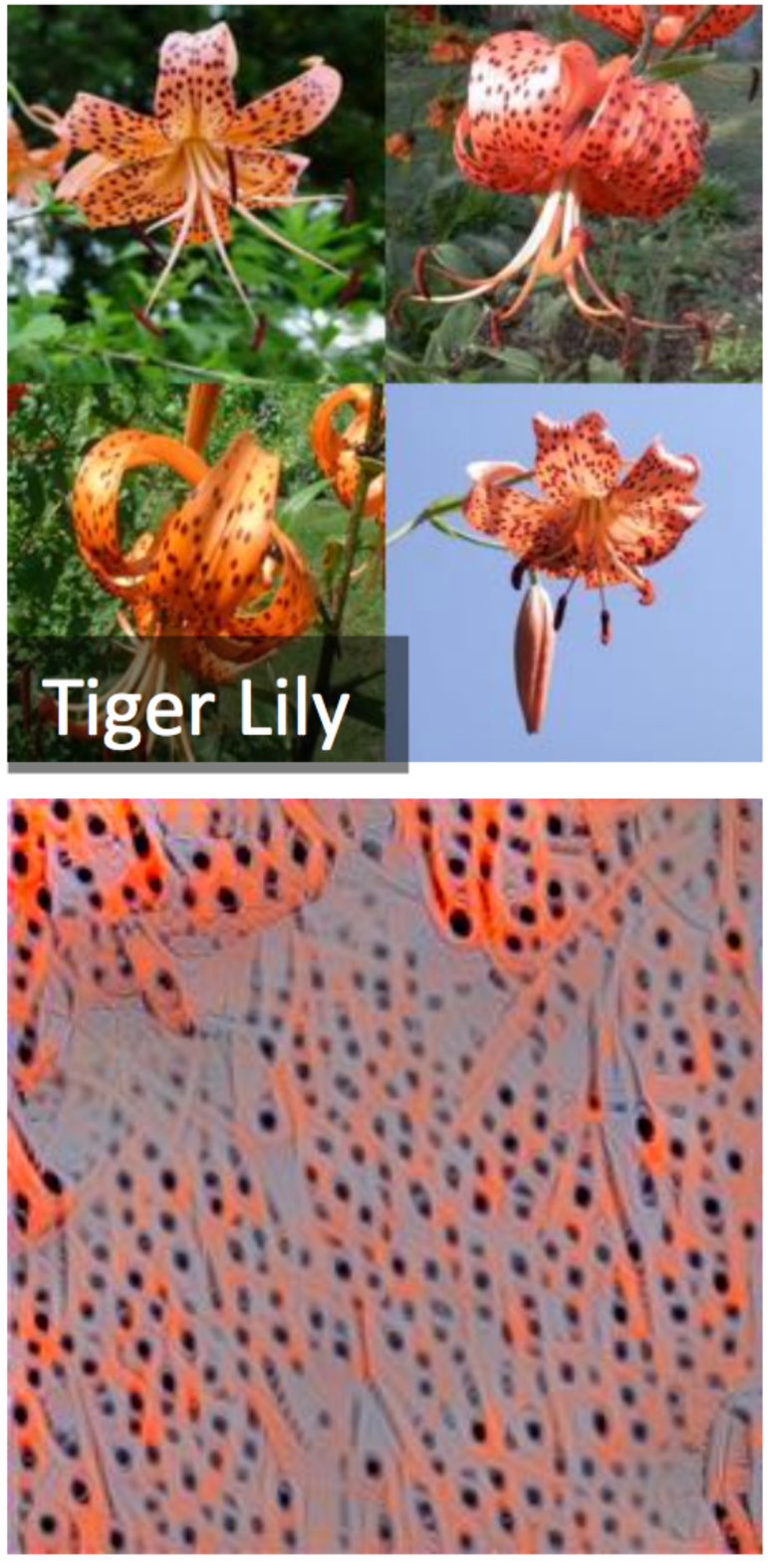


bathroom | bookstore | bowling | closet | classroom | laundromat

Visualizing deep networks

Oxford flowers



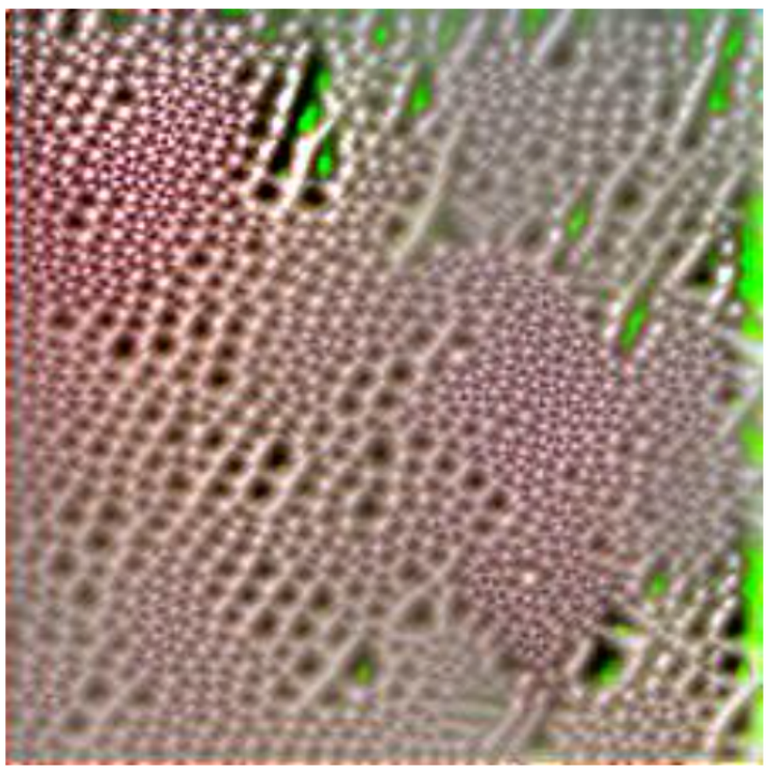


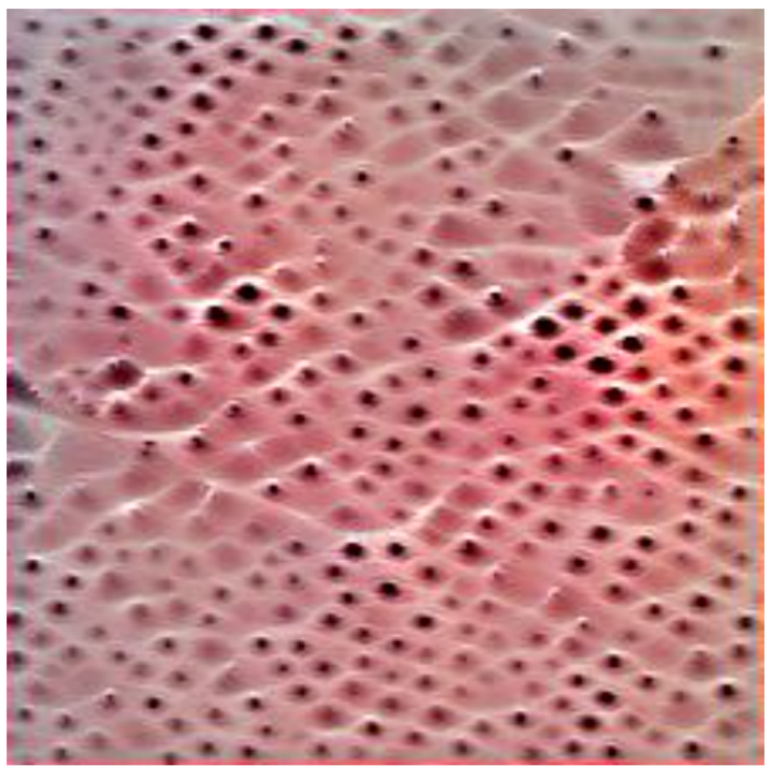
Visualizing deep networks

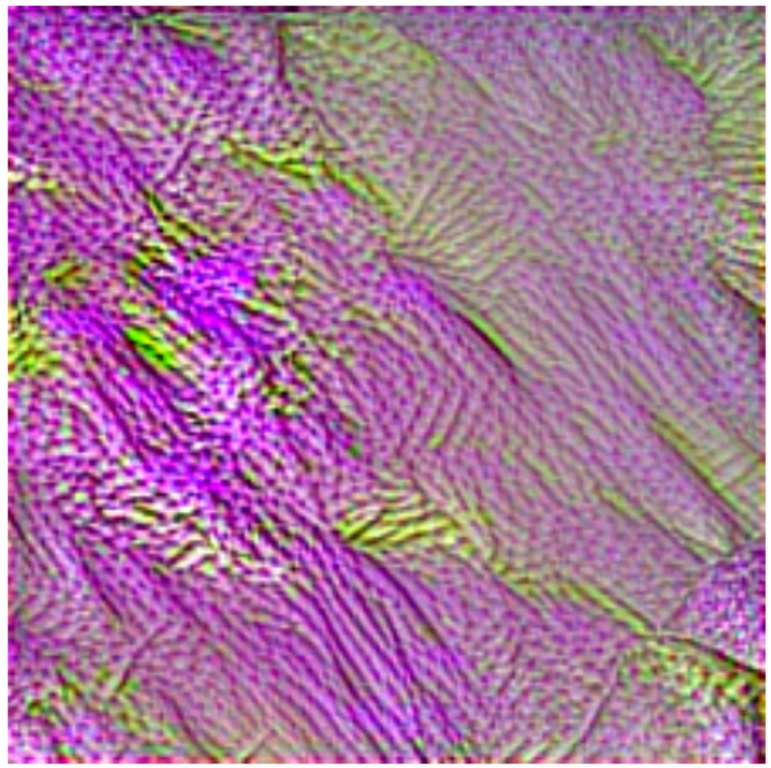
FGVC butterflies and moths

Visualizing deep networks

FGVC fungi







Summary

Second-order / bilinear pooling benefits:

- Excellent performance across diverse domains
- Works efficiently with image regions
- Multiplicative interactions have found their use in visual question answering, video understanding, few-shot learning, etc.

Open questions

- Evaluating matrix functions in the compact domain (cf., Tensor sketch)

Project pages

- http://vis-www.cs.umass.edu/bcnn
- https://people.cs.umass.edu/~smaji/projects/matrix-sqrt/
- http://vis-www.cs.umass.edu/o2dp/

Collaborators: Tsung-Yu Lin, Aruni RoyChowdhury, Mikayla Timm, Chenyun Wu, Piotr Konuisz