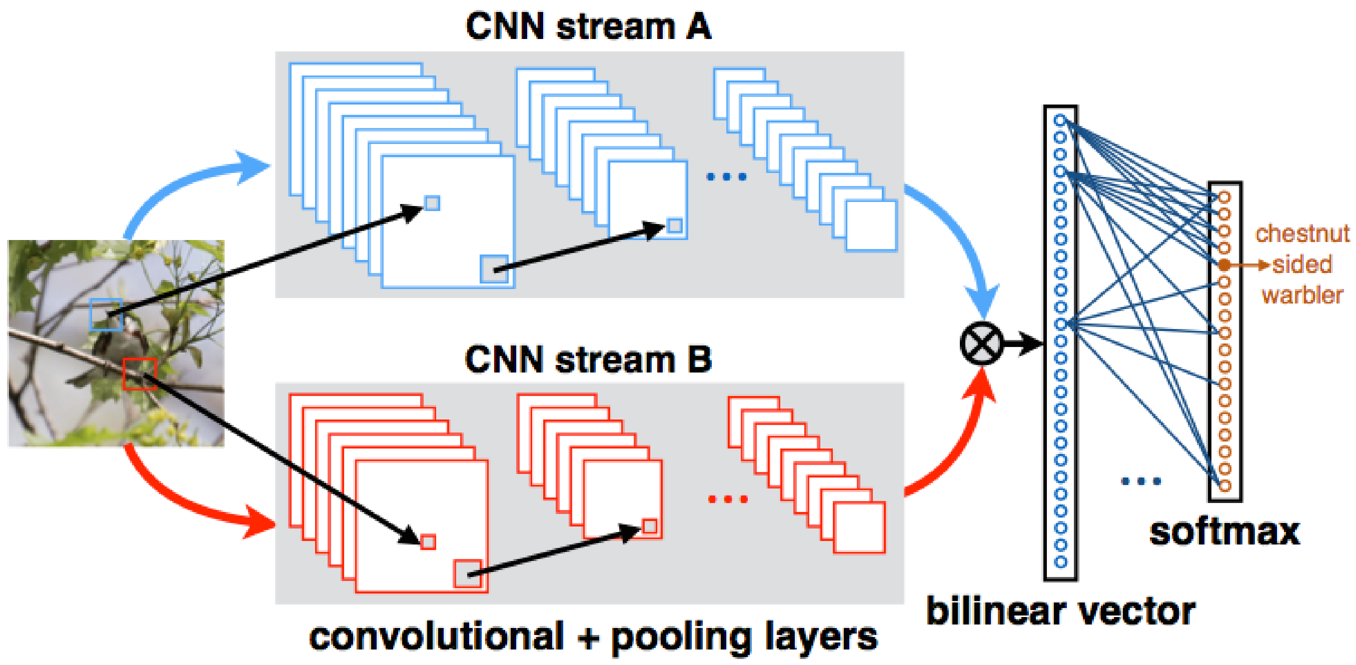
A Research Report on B-CNN & MatConvNet

By 苏金成 @ Hikvision, Mail: jcsu14@fudan.edu.cn
Date: 2016.11.14 ~ 2016.11.20



0 摘要

本文是针对 Tsung-Yu Lin 在 *ICCV2015* 上发表的 *Bilinear CNN Models for Fine-Grained Visual Recognition* 的一个调研报告. 本次调研主要完成的工作:

1. 对文章的技术细节, 涉及的数学公式进行了论证.
2. 在 `CUB-100-2011` 数据集上, 成功复现模型B-CNN[M,M], B-CNN[D,D] 和 B-CNN[M,D].
3. 评估了B-CNN模型在 `Hikvision car1000_data` 数据集上的性能(待结果).
4. 对 B-CNN 模型进行了多个维度的思考.
5. 对源代码进行了深入的剖析 (见附录).
6. 对 MatConvNet 深度学习框架进行了学习 (见附录).
7. 成功往 MatConvNet 中加入自定义网络层 (见附录)
8. 尝试对 B-CNN 模型进行了一些小改进, 效果不明显 (见附录).
9. 深入分析了论文中对称双线性模型(symmetric bilinear model) 和 方差的关系(见附录, 待实验结果).
10. 学习并总结了图像细分类领域的相关技术(BoVW, Fisher Vector, 和 VLAD) (见附录.

1 引言

1.1 背景

目前，应用于物体细分类任务的计算机视觉技术大概可以分为两大类: `part-based` 模型和 `holistic` (整体)模型.

`part-based` 模型通过定位物体的各个部分(localize parts)，并从中提取出特征来刻画图像(constructing image representation)，从而消除了位置、姿势和视角变化的影响，提高了分类的性能. 这些物体的部位通常是手动进行定位.

`holistic` 模型则直接刻画整幅图像. 这些方法包括 BoVW (bag-of-visual-word), FV (fisher vector) 和 VLAD (vector-of-locally-aggregated), 还有一些基于CNN的方法.

`part-based` 模型通常更准确，在数据有限的情况下泛化(generalize)能力更强， 缺点是需要手动进行 part 标注.

`holistic` 模型通常是先提取图像的SIFT特征 [5]，然后用 BoVW, FV 或者 VLAD 的方式对这些SIFT特征进行编码，不需要part标注. 由于 SIFT 特征对光照、位置、视角和姿势等的变化具有较强的抵御能力，所构造的图像表示(image representation)也具有较强的鲁棒性(Robustness). Cimpoi M. 等人将 SIFT 特征替换为从在**ImageNet**预训练好的深度网络的卷积层提取的特征，在多个分类人物上取得了state-of-the-art (2009). 然而这些模型的性能仍然没有 `part-based` 模型的好.

这篇文章提出了 bilinear CNN 模型，该模型利用在**ImageNet**预训练好的深度网络的卷积网络进行特征提取，然后进行双线性编码，取得了state-of-the-art(2015, 2016).

1.2 相关工作

从纵向看，双线性模型最初是被 Tanenbaum 和 Freeman [2]于2000年提出的，用来对两个因子之间的相互作用进行建模，比如字体的风格(style) 和内容(content)之间的相互作用. M. Alex O 等人对此进行扩展，于 2002 年提出了多线性模型[3]，用于对人脸进行建模，提出了 TensorFace. 基于双线性模型，H. Perronnin等人于2009年提出了双线性分类器[4]，该方法直接对SVM分类器的目标函数进行双线性改造，从而得到了双线性分类器.

从横向看，用于物体细分类任务的模型有 part-based [16, 17, 18, 19]， BoVW [6, 7, 8], FV [9, 10, 11, 12] 和 VLAD [13, 14]等.

1.3 参考文献

[1] [**B-CNN**] Lin T Y, Roychowdhury A, Maji S. Bilinear CNN Models for Fine-Grained Visual Recognition[C]. *IEEE International Conference on Computer Vision*. IEEE, 2015:1449-1457.
[2] [**Bilinear**] Tenenbaum J B, Freeman W T. Separating Style and Content with Bilinear Models[J]. *Neural Computation*, 2000, 12(6):1247-1283.
[3] [**Mutilinear**] Vasilescu M A O, Terzopoulos D. Multilinear Analysis of Image Ensembles: TensorFaces[J]. *Lecture Notes in Computer Science*, 2002, 2350:447—460.
[4] [**BilinearClassifier**] Pirsiavash H, Ramanan D, Fowlkes C. Bilinear classifiers for visual recognition[C]. In *NIPS* 2009:1482-1490.
[5] [**SIFT**] Lowe D G, Lowe D G. Distinctive Image Features from Scale-Invariant Keypoints[J]. International Journal of Computer Vision, 2004, 60(2):91-110.
[6] [**BoVW**] Csurka G, Dance C R, Fan L, et al. Visual categorization with bags of keypoints[J]. Workshop on Statistical Learning in Computer Vision Eccv, 2004:1—22.
[7] [**BoVW**] Farquhar J D R, Szedmak S, Meng H, et al. Improving “bag-of-keypoints” image categorisation: Generative Models and PDF-Kernels[J]. University of Southampton, 2005, 68(s 3–4):673-683.
[8] [**BoVW**] Perronnin F, Dance C, Csurka G, et al. Adapted vocabularies for generic visual categorization[C]// Computer Vision - ECCV 2006, European Conference on Computer Vision, Graz, Austria, May 7-13, 2006, Proceedings. 2006:464-475.
[9] [**FV**] Perronnin F, Dance C. Fisher Kernels on Visual Vocabularies for Image Categorization[J]. 2007:1-8.
[10] [**FV**] Perronnin F, Sánchez J, Mensink T. Improving the Fisher Kernel for Large-Scale Image Classification[C]// Computer Vision - ECCV 2010, European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings. 2010:119-133.
[11] [**FV**] Sanchez J, Perronnin F. High-dimensional signature compression for large-scale image classification[J]. 2011, 42(7):1665-1672.
[12] [**FV**]Sánchez J, Perronnin F, Mensink T, et al. Image Classification with the Fisher Vector: Theory and Practice[J]. International Journal of Computer Vision, 2013, 105(3):222-245.
[13] [**VLAD**] Jegou H, Douze M, Schmid C, et al. Aggregating local descriptors into a compact image representation[J]. 2010, 238(6):3304-3311.
[14] [**netVLAD**] Arandjelović R, Gronat P, Torii A, et al. NetVLAD: CNN architecture for weakly supervised place recognition[J]. Computer Science, 2016.
[15] [**DeepFilterBanks**] Cimpoi M. Deep filter banks for texture recognition and segmentation[J]. International Journal of Computer Vision, 2016, 118(1):65-94.
[16] [**part-based**] Bourdev L, Maji S, Malik J. Describing People: A Poselet-Based Approach to Attribute Classification ∗[C]. IEEE International Conference on Computer Vision. IEEE, 2011:1543-1550.
[17] [**part-based**] Zhang N, Farrell R, Darrell T. Pose pooling kernels for sub-category recognition[J]. 2012, 157(10):3665-3672.
[18] [**part-based**] Branson S, Horn G V, Belongie S, et al. Bird Species Categorization Using Pose Normalized Deep Convolutional Nets[J]. Eprint Arxiv, 2014.
[19] [**part-based**] Zhang N, Donahue J, Girshick R, et al. Part-Based R-CNNs for Fine-Grained Category Detection[J]. 2014, 8689:834-849.
[20] [**M-Net**] Chatfield K, Simonyan K, Vedaldi A, et al. Return of the Devil in the Details: Delving Deep into Convolutional Nets[J]. Computer Science, 2014.
[21] [**D-Net**] Simonyan K, Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition[J]. Computer Science, 2014.
[22] Gao Y, Beijbom O, Zhang N, et al. Compact Bilinear Pooling[J]. CVPR 2016.

1.4 术语

图像表示 (image representation)
鲁棒性 (Robustness)
双线性卷积网络 (Bilinear-CNN or B-CNN)
双线性模型 (Bilinear model)
双线性特征 (Bilinear feature)
双线性向量 (Bilinear vector)
双线性编码 (Bilinear encoding)
池化双线性 (Bilinear-pool)
特征提取器 (feature extractor)

平方根规范化 (square root normalization)
2-范数规范化 (l_2 normalization)
全共享 (fully-shared)
部分共享 (partially-shared)
四元组 (Quadruple)
特征图 (feature map)
细调 (fine-tuning)
截断卷积网 (truncated CNN)
下采样 (downsampling)

2 B-CNN 模型 [1]

2.1 B-CNN 概览

B-CNN (Bilinear-CNN) 模型将从两个特征提取器 (feature extractor) 提取所得的特征按位置作外积 (outer product at each location)， 然后将所有位置的外积累加起来，这个累加过程称为 sum-pooling 操作. sum-pooling 的结果被 reshape 成为一个向量，然后依次进行平方根规范化 (square root normalization) 和l_2 规范化 (l_2 normalization). 规范化后的向量称为双线性 (bilinear feature). 最后再针对这些双线性特征训练一个 softmax 分类器或者一组线性SVM分类器进行分类。因此一个 B-CNN 模型由三个部分组成：特征提取器、双线性编码和分类器。

两个特征提取器通常采用在 **ImageNet** 数据集上训练好的模型，并在最后一个卷积层之后截断并丢弃后面的结构。这两个模型可以是全部共享 (fully-shared), 部分共享 (partially-shared) 或者 无共享(no sharing).

B-CNN 模型**训练过程只需要类别标签 (category labels)**，在多个数据集(`CUB-200-2011 dataset` , `FGVC aircraft dataset` , 和 `Stanford cars dataset`) 上达到了 **state-of-the-art** (2015).

2.2 Bilinear 模型技术细节

一个用于图像分类的双线性模型被定义为一个四元组 $\mathcal{B}$:

$$\mathcal{B} = (f_A, f_B, \mathcal{P}, \mathcal{C}).$$

其中，

- f_A, f_B 是特征提取函数 (feature extractor function): $f: \mathcal{L} \times \mathcal{I} \rightarrow \mathcal{R}^{c \times D}$, 其中，
  - $\mathcal{I}$: 可以是输入图像,
  - $\mathcal{L}$: 可以是一组位置 (location),
  - $\mathcal{R}$: 可以是一组大小为 $c \times D$ 的特征,
- $\mathcal{P}$ 是池化函数 (pooling function),
- $\mathcal{C}$ is 是分类函数 (classification function).

将特征提取函数得到的输出在每一个位置上作外积得到双线性特征 (bilinear feature), 即特征 f_A 和 f_B 在位置 l 的双线性特征由下式给出：

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

其中，$f_A(l, \mathcal{I})$ 和 $f_B(l, \mathcal{I})$ 必须具有相同的特征维度c，所得 bilinear feature 的大小为 $c \times c$.

紧接外积之后就是池化函数 $\mathcal{P}$，在双线性模型中，该池化函数定义为将所有位置$l \in \mathcal{L}$上的 bilinear feature 进行累加，即 sum-pooling:

$$\mathcal{P}: \ \ \phi(\mathcal{I}) = \sum_{l \in \mathcal{L}} bilinear(l, \mathcal{I}, f_A, f_B).$$

然后 $\phi(\mathcal{I})$ 被重塑为 (reshape) 一个大小为 $MN \times 1$的向量，称为双线性向量 (**bilinear vector**) $\phi(\mathcal{I})$:

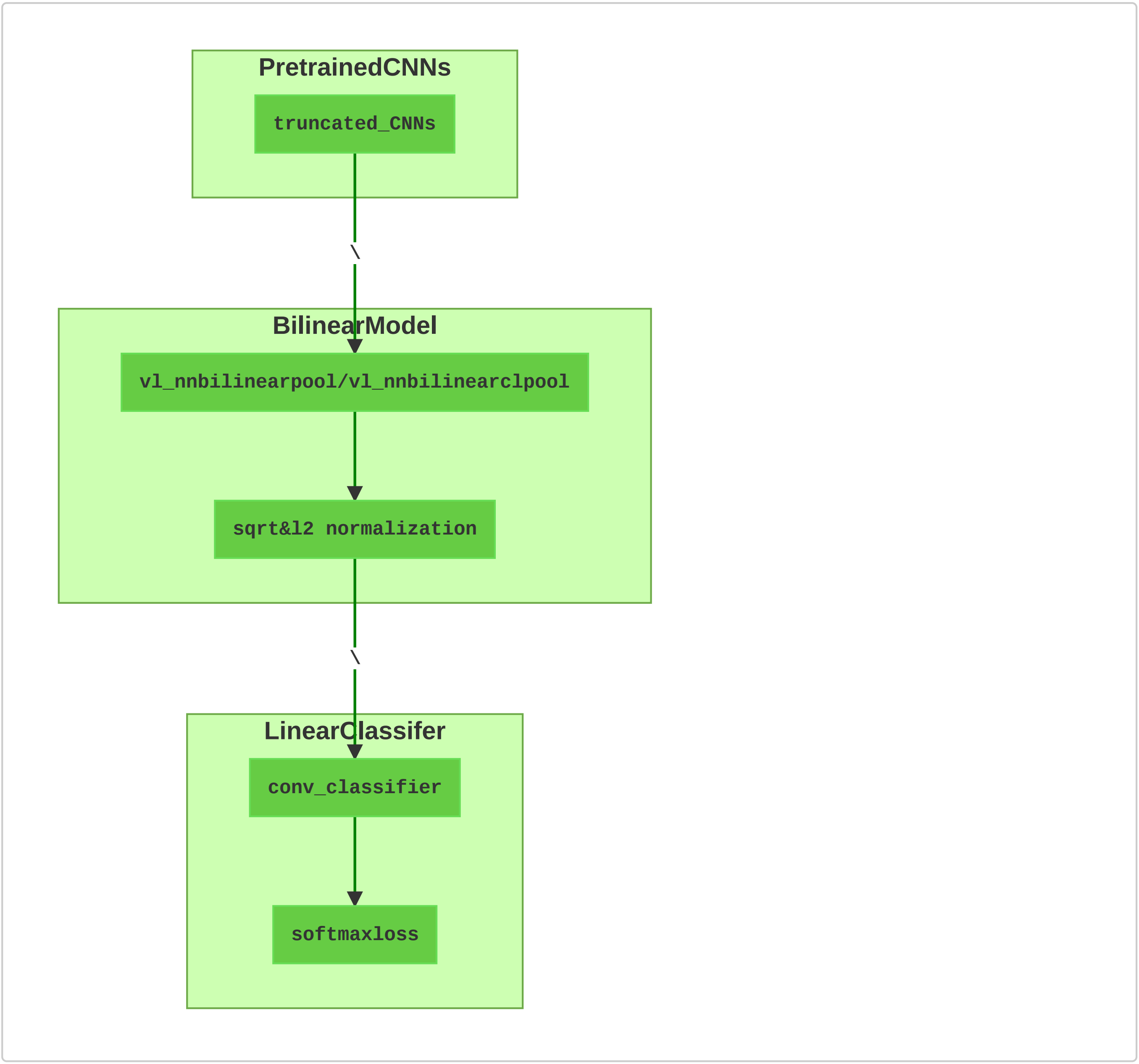
$$\phi(\mathcal{I}) = reshape(\phi(\mathcal{I}), [MN, 1]).$$

该过程看起来似乎有点复杂，实际上它可以通过简单的矩阵相乘来实现。假设现在从两个 CNN 的最后一个卷积层之后分别取得两组特征，一组为 p 个特征图 (feature maps), 每个特征图大小为$m \times m$, 另一组为 q 个特征图, 每个特征图大小

也为$m \times m$. 第一步, 分别将两组特征重塑 (reshape) 为 $(m*m)\times p$ 维和 $(m*m)\times q$ 维的两个矩阵, 分别记为 $X_{(m*m)\times p}$ 和 $Y_{(m*m)\times q}$. 第二步, 计算矩阵乘积X^TY 并 reshape 成一个 $n*n$ 维的向量。这个向量就是所谓的 **bilinear vector** $\phi(\mathcal{I})$.

对于全共享结构，只采用一个CNN网络，最后得到的特征自己跟自己进行矩阵相乘即 X^TX.

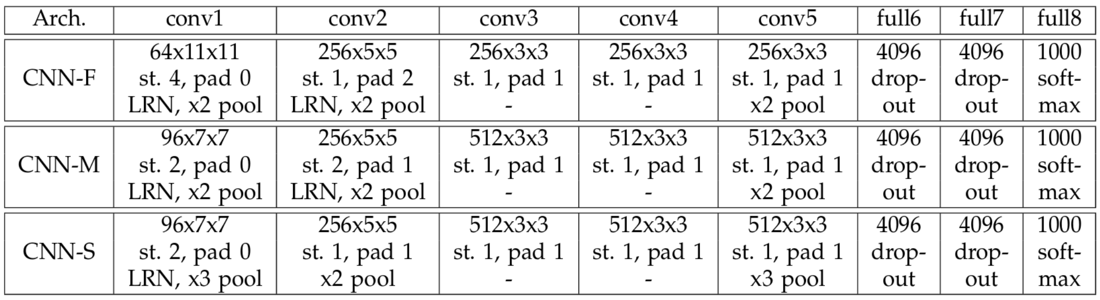
2.3 Bilinear-CNN 模型



1. 将两个在 **ImageNet** 数据集上预训练过的 CNN 模型砍掉最后一个卷积层之后的结构 (保留非线性函数 $relu$), 剩余的部分用作特征提取器。

   文章中采用了 `imagenet-vgg-m.mat` , 在第 14 层 $(conv_5 + relu)$之后截断，称为 `M-Net` , 以及 `imagenet-vgg-verydeep-16.mat` , 在第30层 $(conv_{5\_3} + relu)$ 之后截断, 称为 `D-Net` , 这两个作为特征提取器。

   - `M-Net` 即是下表中的 **CNN-M**

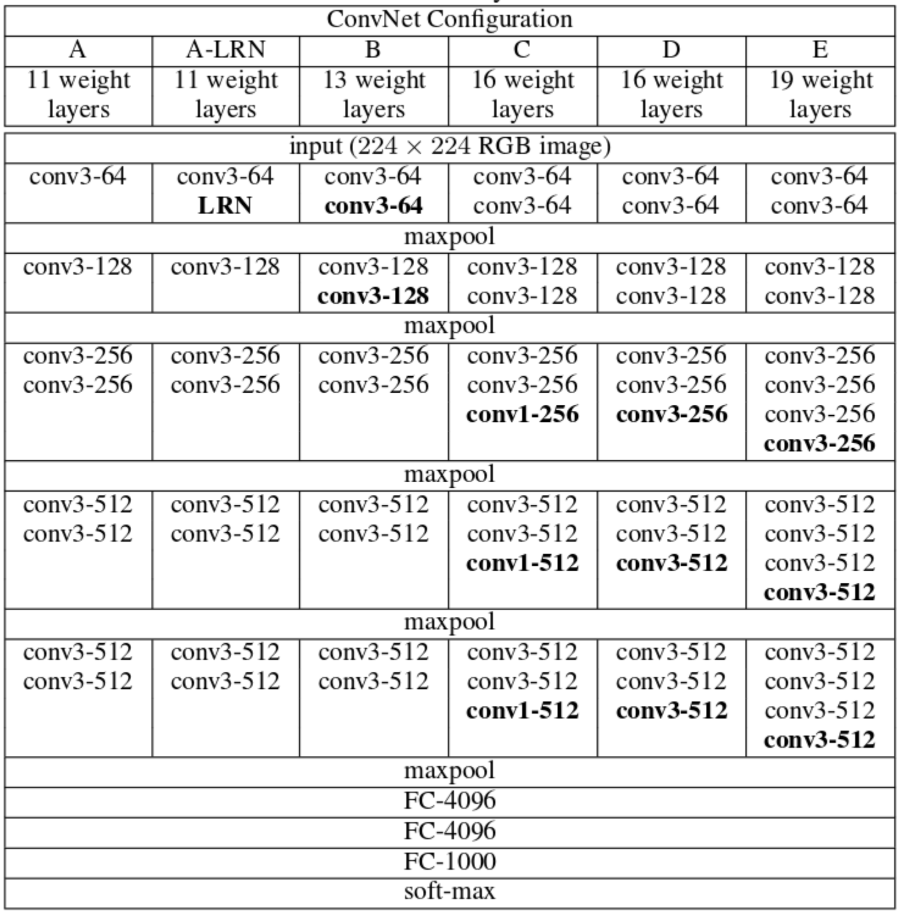


Arch.	conv1	conv2	conv3	conv4	conv5	full6	full7	full8
CNN-F	64x11x11 st. 4, pad 0 LRN, x2 pool	256x5x5 st. 1, pad 2 LRN, x2 pool	256x3x3 st. 1, pad 1 -	256x3x3 st. 1, pad 1 -	256x3x3 st. 1, pad 1 x2 pool	4096 drop- out	4096 drop- out	1000 soft- max
CNN-M	96x7x7 st. 2, pad 0 LRN, x2 pool	256x5x5 st. 2, pad 1 LRN, x2 pool	512x3x3 st. 1, pad 1 -	512x3x3 st. 1, pad 1 -	512x3x3 st. 1, pad 1 x2 pool	4096 drop- out	4096 drop- out	1000 soft- max
CNN-S	96x7x7 st. 2, pad 0 LRN, x3 pool	256x5x5 st. 1, pad 1 x2 pool	512x3x3 st. 1, pad 1 -	512x3x3 st. 1, pad 1 -	512x3x3 st. 1, pad 1 x3 pool	4096 drop- out	4096 drop- out	1000 soft- max

TABLE 1
CNN architectures. Each architecture contains 5 convolutional layers (conv 1–5) and three fully-connected layers (full 1–3). The details of each of the convolutional layers are given in three sub-rows: the first specifies the number of convolution filters and their receptive field size as "num x size x size"; the second indicates the convolution stride ("st.") and spatial padding ("pad"); the third indicates if Local Response Normalisation (LRN) [13] is applied, and the max-pooling downsampling factor. For full 1–3, we specify their dimensionality, which is the same for all three architectures. Full6 and full7 are regularised using dropout [13], while the last layer acts as a multi-way soft-max classifier. The activation function for all weight layers (except for full8) is the REctification Linear Unit (RELU) [13].

- `D-Net` 即是下表中的 **VGG-16**

Table 1: **ConvNet configurations** (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as "conv⟨receptive field size⟩-⟨number of channels⟩". The ReLU activation function is not shown for brevity.



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

2. 应用 *2.2* 节介绍的双线性模型，计算双线性向量 $x = \phi(\mathcal{I})$.
3. 将该向量依次进行平方根规范化和 2 —范数规范化：

$$y \leftarrow sign(x)\sqrt{|x|}$$

$$z \leftarrow \frac{y}{||y||_2}$$

4. 利用 softmax 或者 SVM 作为分类函数 $\mathcal{C}$. 文章中用了一组线性SVM作为最后的分类器.

2.4 梯度计算&编程细节

在训练网络时，通常每次处理一小批样本。为了方便，这里假设只有一个样本，x_1 和 x_2 为两个截断卷积网提取的特征。注意，当两个卷积网相同时，有$x_1 = x_2$.

1. 池化双线性 (Bilinear-pool): `y = vl_nnbilinearclpool(x1, x2, varargin)`
   输入: 两组特征图 x_1, x_2 , 大小分别为 $[h_1, w_1, ch_1]$ 和 $[h_2, w_2, ch_2]$ ，对应高度, 宽度和通道数. $\nabla_Y Z$ (`dzdy`) 表示损失函数 Z 对 y 的梯度.
   输出: 双线性向量 y ，大小为 $[ch1 * ch2, 1]$.

   - **正向传播**

     - 把 x_1, x_2 `reshape` 为同样的大小, i.e., 对其中较大的一个进行下采样 (downsampling) 使得 $h_1 * w_1 == h_2 * w_2$:

       ```
       resize_image(x1, x2)
       ```

     - Reshape x_1, x_2 to X_a, X_b with sizes $[h_1 * w_1, ch_1]$ and $[h_2 * w_2, ch_2]$ respectively:

       ```
       Xa = reshape(x1, [h1*w1, ch1])
       Xb = reshape(x2, [h2*w2, ch2])
       ```

     - Cacultate their outer product:

       $$Y = X_a^T X_b$$

       ```
         Y = Xa' * Xb
       ```

     - Reshape y to a vector with size $[ch_1 * ch_2, 1]$.

       ```
       y = reshape(Y, [ch1 * ch2, 1])
       ```

   - **反向传播**
     Since $Y = X_a^T X_b$, calculate $\nabla_{X_a} Z$ is easy (Z denote the cost function):

     $$\nabla_{X_a} Z = \nabla_{X_a} Y \cdot \nabla_Y Z = X_b \\ \cdot (\nabla_Y Z)^T, \\ \nabla_{X_b} Z = \nabla_{X_b} Y \cdot \nabla_Y Z = X_a \cdot (\nabla_Y Z).$$

     - Reshape x_1, x_2 to X_a, X_b with sizes $[h_1 * w_1, ch_1]$ and $[h_2 * w_2, ch_2]$ respectively:

       ```
         Xa = reshape(x1, [h1*w1, ch1])
         Xb = reshape(x2, [h2*w2, ch2])
       ```

     - Reshape input `dzdy` to size `[ch1, ch2]` .

       ```
         Delta = reshape(dzdy, [ch1, ch2]
       ```

     - Calculate `dzdxa` and `dzdxb`

       ```
         dzdxa = Xb * Delta'
         dzdxb = Xa * Delta
       ```

     - Reshape `dzdxa` and `dzdxb` to sizes as with `x1` and `x2` .

       ```
         dzdx1 = reshape(dzdxa, [h1, w1, ch1])
       ```

```
    dzdx2 = reshape(dzdxb, [h2, w2, ch2])
```

2. 平方根规范化: `y = vl_nnsqrt(x, param, varargin)`
   - **正向传播** (element-wise operation):

     $$y = sign(x)\ .*\ \sqrt{abs(x)}$$

     Note that y is with the same size as x.
   - **反向传播** (element-wise operation):

     $$\frac{dy}{dx} = 0.5\ .*\ \frac{1}{\sqrt{abs(x)\ .+ param}}$$

     $$\frac{dz}{dx} = \frac{dy}{dx}\ .*\ \frac{dz}{dy}$$

     where $\frac{dy}{dx} = (\frac{dy_1}{dx_1}, \dots, \frac{dy_n}{dx_n})^T$, and `param` is used for numeric stability.
3. 2-范数规范化: `y = vl_nnl2normalization(x, param, varargin)`
   - **正向传播**:

     $$y = \frac{x}{||x||_2}$$

     Note that y is with the same size as x.
   - **反向传播**:

     $$\frac{d}{dx_j}(\frac{x}{||x||_2}) = \frac{1}{||x||_2^3}(x_1^2 + \dots + x_{j-1}^2 + x_{j+1}^2 + \dots + x_n^2) = \frac{1}{||x||_2} - \frac{x_j^2}{||x||_2^3}$$

     Which gives the vectorize form:

     $$\nabla(\frac{x}{||x||_2}) = \frac{1}{||x||_2}\ .-\ \frac{x.^2}{||x||_2^3}$$

     where $x.^2 = (x_1^2, \dots, x_n^2)^T$.

     ```
       To prevent large values and ensure numeric stability, it is recommended to prep
     rocess ||x|| by adding a threshold:
     ```

     $$||x||_2 = ||x||_2 + threshold$$

2.5 训练方法

文章采用两步训练法:

- **第一步**. 先用 双线性特征 训练一个 softmax 分类器。这实际上是一个凸优化问题，利用 SGD 算法可以很快得到最优解. 文章中将学习速率设为 `learningRate=0.001` , 训练了300个 *Epoch*. 实际上，将学习速率设为 `learningRate=0.06` , 可以在 10 个 *Epoch* 之内达到最优解。
- **第二步**. 在原始数据集上训练整个B-CNN模型, 即细调 (fine-tuning). 文章用 `learningRate=0.001` 训练100个*Epoch*.

注意 文章在第二步训练完成后，便将 softmax 分类器砍掉，重新训练一组SVM分类器.

2.6 实验结果

1. **细分类(fine-grained)数据集**

    - 鸟类: CUB-200-2011 dataset 包含200类共11788张图片，其中接近一半的图片被划分为测试集. 具体参见该数据集下的 .txt 文件.
    - 飞机: FGVC aircraft dataset包含100类共10000张图片。
    - 汽车: Stanford cars dataset 包含196类共16185图片。
      其中飞机和汽车两个数据集是 **FGComp2013 challenge** 的一部分.

2. **参照组**

    文章中采用了 `imagenet-vgg-m.mat` , 在第 14 层 $(conv_5 + relu)$之后截断，称为 `M-Net` , 以及 `imagenet-vgg-verydeep-16.mat` , 在第30层 $(conv_{5\_4} + relu)$ 之后截断, 称为 `D-Net` , 这两个作为特征提取器。

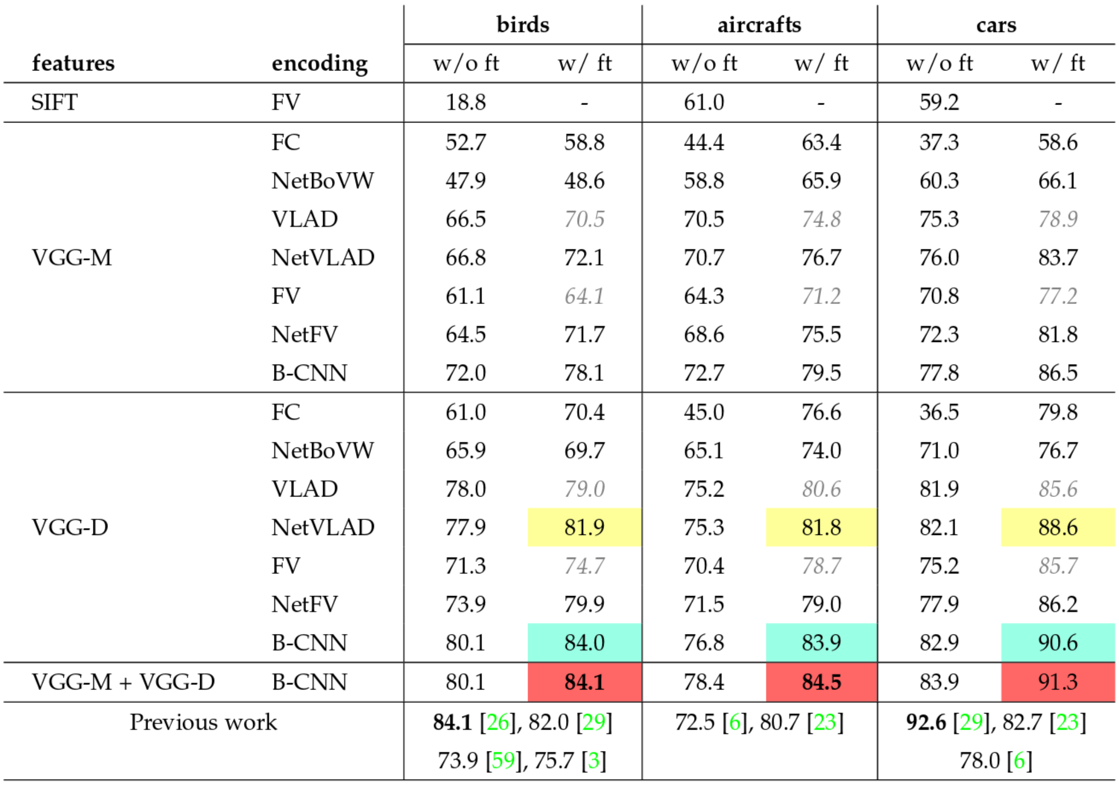
    - **Baseline**, FC-CNN (Full-Connected CNN): FC-CNN[M], FC-CNN[D]
    - 不同的特征提取器 + 不同的编码方式:
        - 特征提取器: SIFT, VGG-M (M-Net) 和 VGG-D (D-Net)
        - 编码方式: FC, NetBoVW, VLAD, NetVLAD, FV, NetFV 和 BCNN.
    - 以前的文章

3. **实验结果**

    实验结果见下图. 其中 `w/o ft` 表示 without fine-tuning， 即仅仅训练了第一步，而没有对整个B-CNN模型进行整体细调(fine-tuning)的结果. `w/ ft` 则表示细调之后的结果.

    - 对比 SIFT + FV 和 VGG-M/D + FV 的结果可以推断，这三个数据集上， CNN特征的性能明显比 SIFT 特征好.
    - 对于同样的特征提取器来说，不同的编码方式，最终得到的结果之间差距十分巨大; 而对于同样的编码方式，特征提取器的重要性不亚于编码方式. 可以看到，特征提取器和编码方式可能互为瓶颈，从另一个方面来说，它们也是相辅相成，互相增益的.

    - 对于多种编码方式，Bilinear-pool 的表现十分突出. 因此双线性编码似乎是目前表现最优的编码方式. 然而并没有理由表明, 双线性编码是最好的编码方式, 或许可以通过深入理解双线性编码来找到更好的编码方式.



		birds		aircrafts		cars	
features	encoding	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	FC	52.7	58.8	44.4	63.4	37.3	58.6
	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	FC	61.0	70.4	45.0	76.6	36.5	79.8
	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	76.8	83.9	82.9	90.6
VGG-M + VGG-D	B-CNN	80.1	**84.1**	78.4	**84.5**	83.9	**91.3**
Previous work		**84.1** [26], 82.0 [29] 73.9 [59], 75.7 [3]		72.5 [6], 80.7 [23]		**92.6** [29], 82.7 [23] 78.0 [6]	

3. B-CNN 模型复现&评估

3.1 B-CNN 模型复现情况

1. **数据集** `CUB-100-2011`
2. **复现模型** `B-CNN[M,M]`, `B-CNN[M,D]`, `B-CNN[D,D]`

3. **复现情况**

模型	文章准确率	复现准确率	速度(X TITAN)
B-CNN[M, M]	78.1%	77.56%	37.5 fps
B-CNN[D, D]	84.0%	83.90%	6.7 fps
B-CNN[M, D]	84.1%	83.83%	6.7 fps

3.2 B-CNN 模型在 `Hikvision car1000_data` 数据集上的表现

1. **数据集**. `car1000_data` 数据集包含1000类超过300,000张图片，测试集包含2946张图片，每张图片大小不一.

2. **评估方式**. 为了评估B-CNN模型在我们车类数据集上的表现，本次评估选取了3中模型进行比较: FC-CNN[M], B-CNN[M], 和 B-CNN[D]. 输入到FC-CNN的图片被resize为224×224大小， 输入到B-CNN[M/D]的图片则被resize为448×448的大小. VGG-M 提取的特征大小为 $27 \times 27 \times 512$, 而 VGG-D 提取的特征大小为 $28 \times 28 \times 512$. 最终编码后得到的双线性向量(bilinear vector)大小为 $(512 * 512 = 262,144) \times 1$.

3. **训练方式**. 训练的第一步(即先单独对softmax分类器进行训练)需要对所有的图片提取CNN特征并编码为262,144维的双线性向量并全部保存下来作为数据集，$300,000 \times 262,144 \times 32 = 300GB$, 也就是先构造一个300GB大小的数据集，并且在训练过程中，B-CNN 还会保存许多中间数据，这使得在我的工作站上无法进行训练(我的工作站/home目录当时只剩不到400GB).因此舍弃第一步。直接对softmax分类器的权重进行高斯随机初始化，然后执行训练的第二步.

1. **评估结果**.

模型	训练方式	准确率	速度(X TITAN)
F-CNN[M]	lr0.001+20, lr0.0005+10	92.27%	55 fps
B-CNN[M, M]	lr0.001+10, lr0.0005+10	xx.xx%	30 fps
B-CNN[D, D]	lr0.0005+10	xx.xx%	6 fps

4 反思

4.1 直观解释(intuition)

直觉上，双线性类似于二次核的展开$((x_1+x_2+\ldots+x_n)^2 = \sum_{i,j=1,\ldots,n} x_i x_j)$, 通过考虑所有因子两两之间的相互作用，使得特征提取器 f_A 和 f_B 的输出互为条件.
(Intuitively, the bilinear form allows the outputs of the feature extractors f_A and f_B to be conditioned on each other by considering all their paire-wise interactions similar to a quadratic kernel expansion.)

考虑性别识别任务. 一种方法是首先训练一个 gender-neutral 的人脸探测器, 随后用一个性别分类器进行分类. 然而，更好的方法也许是直接训练一个 gender-specific 的人脸探测器. 双线性模型就是这样一种方法. 通过特征提取器 f_A 和 f_B 的联合训练 (jointly training)， 双线性模型可以基于训练数据对这些特征的表达进行有效的折中.
(Considering the task of gender recognition. One approach is to first train a gender-neutral face detector and followed

by a gender classifier. However, it may be better to train a gender-specific face detector instead. By jointly training f_A and f_B the bilinear model can effectly trade-off the representation power of the features based on the data.)

4.2 从训练的过程看双线性模型

双线性模型对两个因子之间的相互作用进行建模, 即对于两个因子x_1 和 x_2, 考虑它们的乘子 x_1x_2, 给定其中一个因子, 该乘子对另一个因子呈线性关系.

在进行分类时, softmax分类器赋予每一个乘子 x_ix_j 一个权重 w_{ij} 并全部相加, 根据相加的结果决定最终的类别. 由于 x_i 和 x_j 的相互作用, 在训练阶段, w_{ij} 同时受这两个因子影响, 最终学习到一个折中值.

考虑单个神经元(neuron), 它接受输入 x_i 和 x_j, 并输出 $y = f(w_{ij}x_ix_j)$. 对应的梯度为 $\frac{dy}{dw_{ij}} = x_ix_jf'(w_{ij}x_ix_j)$. 在利用简单的SGD算法进行训练时, w_{ij}根据$w_{ij} := w_{ij} - \gamma\frac{dy}{dw_{ij}}$ 进行更新. 可见, x_i 和 x_j 中的任何一个发生变化, w_{ij} 都会及时随之进行调整，从而同时向它们学习.

分析到这里, **一个自然的问题是, 为什么是简单的线性相乘？ 是否可以用一个更高阶的非线性函数 $f(x_i, x_j)$对x_i和x_j$ 的相互作用进行建模**?

我们看到, 该双线性模型中计算了所有因子两两之间的内积, 在这一点上, **是否可以应用核技巧(kernel trick)**?

4.3 对称双线性模型(symmetric bilinear model)跟方差有什么关系？

对于对称双线性模型， 文章中的双线性模块不过是将经平方根规范化和2-范数规范化后的 X^TX 传递给后面的分类器. 注意到， 当矩阵 X 的行均值为零即 $\sum_j X_{ij} = 0$ 时，X^TX 实际上为矩阵 X^T 的方差矩阵. 且看 X 的方差矩阵公式:

$$\begin{aligned} Cov(X) = E(XX^T) - EX(EX)^T &= \frac{1}{n-1}(XX^T - nEX(EX)^T) \\ &= \frac{1}{n-1}(XX^T - \frac{1}{n}(Xe)(Xe)^T) \end{aligned}$$

其中， n是矩阵 X 的列数, $e \in \{1\}^n$. 上式的第二个等号利用了方差的定义式 $Cov(X) = E(X-EX)(X-EX)^T$ 以及均值定义式 $EX = \frac{1}{n}\sum_{k=1}^n X_{.k}$. 第三个等号只利用了均值定义式. $= \frac{1}{n-1}\sum_{k=1}^n (X_{.k} - EX)(X_{.k} - EX)^T$

可见当 $Xe = 0$, 也就是矩阵X 的行和为0时， X^TX 即为 X^T的方差矩阵.

联系到文章中，X 是由卷积网络提取的特征矩阵, X 的每一列是对应一个特征图(feature map) reshape后的列向量， X^T **的方差的统计意义即是各个特征图之间的相关性**.

那么问题来了,

- 卷积网提取的特征是否隐含了这样一个性质: **所有的特征图相加结果为0**？

亦或者,

- 对称双线性模型是否隐含了这样一个约束： **所有的特征图相加结果为0**？

另外,

- **如果把双线性编码方式直接替换为方差矩阵即 $\frac{1}{n-1}(XX^T - \frac{1}{n}(Xe)(Xe)^T)$, 性能会有什么变化呢**？

这两个问题可以通过分析细调之前的X和细调之后的X得到答案. 详见**附录-6 深入探究 symmetric bilinear 和 Covariance 之间的关系**.

4.4 双线性模型参数复杂度

文中的 bilinear 编码本身不引入任何参数. 对于 `B-CNN[M,M]` 模型, 待编码的是一个 729×512 的矩阵 X, 编码过程就是先计算 X^TX 得到一个 512×512 的矩阵, 然后 `reshape` 成一个262144维的双线性向量 x. 随后经过平方根规范化$y \leftarrow sign(x)\sqrt{|x|}$ 和 2-范数规范化$z \leftarrow \frac{y}{||y||_2}$. 最后交给 softmax分类器进行分类.

问题在于, 输入到 softmax 分类器的是一个 262144 维的向量, 对于鸟类数据集100类来说，将会是一个 262144×100 的权重矩阵，即 $26,214,400$ 个参数(26个million)!

这么多的参数必然存在着大量的冗余. 针对这一点, Y.Gao等人于CVPR2016[22]发表的文章提出了一个紧凑的双线性编码方法(compact bilinear pooling).

考虑非对称情形，双线性特征是两个特征 x 和 y 的外积. 本文作者在随后的工作中，尝试了三种降维策略:

- 先计算 x 和 y 的外积, 再把外积投射到低维空间.
- 先把 x 和 y 投射到低维空间, 再计算它们的外积.
- 只把 x 和 y 中的一个投射到低维空间，再计算它们的外积.

投射矩阵采用 PCA 方法求得. 第一种方法计算复杂度太高, 因为 PCA 需要计算一个很高维的方差矩阵 (对于B-CNN[M,M], 262144×262144维). 第二种方法会计算复杂度低很多, 但是会带来明显的性能损失. 所幸第三种方法不会带来明显的性能损失, 甚至还可能稍微提升性能. 具体参见本篇文章的最新版.

5 总结与展望

5.1 总结

本文是针对 Tsung-Yu Lin 在 *ICCV2015* 上发表的 *Bilinear CNN Models for Fine-Grained Visual Recognition* 的一个调研报告. 本次调研完成的主要工作:

1. 对文章的技术细节, 涉及的数学公式进行了论证.
2. 在 `CUB-100-2011` 数据集上, 成功复现模型B-CNN[M,M], B-CNN[D,D] 和 B-CNN[M,D].
3. 评估了B-CNN模型在 `Hikvision car1000_data` 数据集上的性能(待结果).
4. 对 B-CNN 模型进行了多个维度的思考.
5. 对源代码进行了深入的剖析 (见附录).
6. 对 MatConvNet 深度学习框架进行了学习 (见附录).
7. 成功往 MatConvNet 中加入自定义网络层 (见附录)
8. 尝试对 B-CNN 模型进行了一些小改进, 效果不明显 (见附录).
9. 深入分析了论文中对称双线性模型(symmetric bilinear model) 和 方差的关系(见附录, 待完成).
10. 学习并总结了图像细分类领域的相关技术(BoVW, Fisher Vector, 和 VLAD) (见附录).

5.2 展望

从文章中可以看出，细分类任务的技术思路是: 特征提取器(CNN, SIFT) + 编码器(Bilinear, VLAD, FV, BoVW) + 分类器(softmax, SVM).

相关文章表明 CNN 提取的特征可以优于 SIFT 特征. 未来的工作可以是:

- 寻找更好的特征提取器.
- 寻找更好的编码方式.
- 将 Bilinear 模型移植到 Caffe 上.

附录-1 MatConvNet 介绍

The B-CNN model is implemented based on *MatConvNet*.

MatConvNet is a MATLAB toolbox implementing *Convolutional Neural Networks* (CNNs) for computer vision applications. It is simple, efficient, and can run and learn state-of-the-art CNNs. Many pre-trained CNNs for image classification, segmentation, face recognition, and text detection are available.

- **Main functions provided**:

```
    vl_nnbilinearsampler
    vl_nnconcat
    vl_nnspnorm
    vl_nnpdist
    vl_nnconv
    vl_nnconvt
```

```
vl_nncrop
vl_nndropout
vl_nnloss
vl_nnnoffset
vl_nnnormalize
vl_nnpool
vl_nnrelu
vl_nnroipool
vl_nnsigmoid
vl_nnsoftmax
vl_nnsoftmaxloss
DagNN wrapper
vl_simplenn
vl_simplenn_tidy
vl_simplenn_diagnose
vl_simplenn_display
vl_simplenn_move
vl_argparse
vl_compilenn
vl_rootnn
vl_setupnn
vl_imreadjpeg
vl_taccum
```

- **Pretrained models**:
  - **Fast R-CNN** for object detection
  - **VGG-Face** for face recognition
  - **Fully-Convolutional Networks** (FCN), **BVLC FCN** and **Torr Vision Group FCN-8s** for semantic segmentation
  - **ResNet**, **GoogLeNet**, **VGG-VD**, **VGG-S/M/F**, **Caffe reference model**, and **AlexNet** for ImageNet ILSVRC classification.
- **Documentation**
  - manual
  - Function description
  - CNN practical

附录-2 B-CNN 实验平台搭建

1. download bcnn: `git clone https://bitbucket.org/tsungyu/bcnn.git`

   ```
   $ cd bcnn
   ```

   download bcnn-package: `git clone https://bitbucket.org/tsungyu/bcnn-package.git`
   download vlfeat: `git clone https://github.com/vlfeat/vlfeat.git`
   download matconvnet: `git clone https://github.com/vlfeat/matconvnet.git`
2. keep `bcnn-package`, `vlfeat`, and `matconvnet` under `bcnn` directory:

   ```
   --bcnn
       --bcnn-package
       --vlfeat
       --matconvnet
       ...
   ```

3. install `vlfeat`:

   ```
   $ cd /path/to/bcnn/vlfeat
   $ sudo make
   $ matlab
   > ./toolbox/vl_setup
   > vl_demo
   ```

4. install `matconvnet` (must already installed cuda):

   ```
   $ cd /path/to/bcnn/matconvnet
   $ matlab
   > vl_compilenn('enableGpu', true, 'cudaRoot', '/usr/local/cuda') %./matlab/vl_compilen
   n.m
   ```

   info:

   ```
   A lot of complaints that `gcc/g++-4.8.3` are not supported. However, it said 'compile
   succeeded'.
   ```

   setup `matconvnet`:

   ```
   > ./matlab/vl_setupnn
   ```

5. install `bcnn`:

   ```
   $ cd /path/to/bcnn
   $ matlab
   > ./setup
   ```

   download `bcnn-cub-dm.mat`, and put it under `/path/to/bcnn/data/ft_models`.
   download `svm-cub-vdm.mat`, and put it under `/path/to/bcnn/data/models`

   ```
   > ./bird_demo
   ```

附录-3 带 B-CNN 飞起来

1. 让 run_experiments_bcnn_train 跑起来

In summary, what should you do after installing the B-CNN package to run the training:

- download `imagenet-vgg-m.mat` and `imagenet-vgg-verydeep-16.mat`, and put it under `/path/to/bcnn_root/data/models`
- download datasets `CUB-200-2011`, and put it under `/path/to/bcnn_root/data` with a new name `cub`

- check the variables `setupNameList`, `encoderList`, and `datasetList` in `run_experiments_bcnn_train.m`, to use the `bcnnmm` model and `cub` dataset.

- **NOTE: The B-CNN source code seems to have a built-in bug**. To work around this bug, add one line to file `/path/to/bcnn/imdb_get_batch_bcnn.m`:

```
%/path/to/bcnn/imdb_get_batch_bcnn.m

    ...
```

```
    opts(i).prefetch = false;
    opts(i).scale = 1;
    opts(i).cropSize = 224/256; % adde this line
    opts(i) = vl_argparse(opts(i), {varargin{1}(i),varargin{2:end}});

    if(i==1)

    ...
```

OK, Let the function `run_experiments_bcnn_train` run!

2. 解决源代码自带 Bug 的过程

Assume the line `opts(i).cropsize = 224/256` is not added to file `/path/to/bcnn/imdb_get_batch_bcnn.m` and run `run_experiments_bcnn_train`!

```
 196:  196.House_Wren (train:    29, test:    30 total:    59)
 197:  197.Marsh_Wren (train:    30, test:    30 total:    60)
 198:   198.Rock_Wren (train:    30, test:    30 total:    60)
 199: 199.Winter_Wren (train:    30, test:    30 total:    60)
 200: 200.Common_Yellowthroat (train:    30, test:    30 total:    60)
 Inf:       **total** (train: 11788, test: 11788 total: 11788)
dataset: there are 11788 images (5994 trainval 5794 test)
collecting image stats: batch starting with image 2 ...Error using vl_argparse (line 101)
Unknown parameter 'cropSize'

Error in vl_argparse (line 80)
      opts = vl_argparse(opts, vertcat(params,values)) ;

Error in imdb_get_batch_bcnn (line 35)
    opts(i) = vl_argparse(opts(i), {varargin{1}(i),varargin{2:end}});

Error in getBatchSimpleNNWrapper>getBatchSimpleNN (line 10)
im = imdb_get_batch_bcnn(images, opts, ...

Error in getBatchSimpleNNWrapper>@(imdb,batch)getBatchSimpleNN(imdb,batch,opts) (line 4)
fn = @(imdb, batch) getBatchSimpleNN(imdb, batch, opts) ;

Error in imdb_bcnn_train_dag>getImageStats (line 213)
  temp = fn(imdb, batch) ;

Error in imdb_bcnn_train_dag (line 83)
  [averageImage, rgbMean, rgbCovariance] = getImageStats(imdb, bopts) ;

Error in run_experiments_bcnn_train (line 81)
          imdb_bcnn_train_dag(imdb, opts);
```

ERROR! …

Well, set some breakpoints, and locate the error (NOTE: although MATLAB gives lots of helpful error information to help us debug, chances are that the error location can be not that accurate. And it also helps a lot for understanding the code to trace back the execution stack.):

```
**function run_experiments_bcnn_train():**
    -> imdb_bcnn_train_dag(imdb, opts);
**function imdb_bcnn_train_dag(imdb, opts, varargin):**
    -> [averageImage, rgbMean, rgbCovariance] = getImageStats(imdb, bopts);
**function [averageImage, rgbMean, rgbCovariance] = getImageStats(imdb, opts):**
    -> fn = getBatchSimpleNNWrapper(opts); temp = fn(imdb, batch);
**function fn = getBatchSimpleNNWrapper(opts):**
    -> fn = @(imdb, batch) getBatchSimpleNN(imdb, batch, opts);
**function [im,labels] = getBatchSimpleNN(imdb, batch, opts):**
    -> im = imdb_get_batch_bcnn(images, opts, ...
                        'prefetch', nargout == 0);
**function imo = imdb_get_batch_bcnn(images, varargin):**
    -> opts(i) = vl_argparse(opts(i), {varargin{1}(i),varargin{2:end}});
**function [opts, args] = vl_argparse(opts, args, varargin):**
    ->  if nargout == 1
            opts = vl_argparse(opts, vertcat(params,values)) ;
**function [opts, args] = vl_argparse(opts, args, varargin):**
    ->  if nargout == 1
            error('Unknown parameter ''%s''', param) ;
```

What the hell is `cropSize`?

After some tracing during debugging, I find that `cropSize` is introduced by `net.meta.normalization (imdb_bcnn_train_dag.m: line 71)` to `bopts` which will be passed to function call `getImageStats(imdb, bopts)` and further be passed to function `imdb_get_batch_bcmm(images, varargin)` via `bopts -> opts -> varargin`, and finally be passed to function `vl_argparse(opts, args, varargin)`.

Inspect a few more lines around `opts = vl_argparse(opts, vertcat(params,values))` in `vl_argparse.m`:

```
% ./matconvnet/matlab/vl_argparse.m

if recursive && isstruct(args{ai})
    params = fieldnames(args{ai})' ;
    values = struct2cell(args{ai})' ;
    if nargout == 1
        opts = vl_argparse(opts, vertcat(params,values)) ;
    else
        [opts, rest] = vl_argparse(opts, reshape(vertcat(params,values), 1, [])) ;
        args{ai} = cell2struct(rest(2:2:end), rest(1:2:end), 2) ;
        keep(ai) = true ;
    end
    ai = ai + 1 ;
    continue ;
 end
```

where `recursive` is set to `true`, and `args` is passed in as a pack `{varargin{1}(i),varargin{2:end}}`:

```
**function imo = imdb_get_batch_bcnn(images, varargin):**
-> opts(i) = vl_argparse(opts(i), {varargin{1}(i),varargin{2:end}});
**function [opts, args] = vl_argparse(opts, args, varargin):**
```

Look one more wrapper outsider, the `varargin` in function `imdb_get_batch_bcnn()` is passed in via `opts,'prefetch', nargout == 0`:

```
**function [im,labels] = getBatchSimpleNN(imdb, batch, opts):**
    -> im = imdb_get_batch_bcnn(images, opts, ...
                    'prefetch', nargout == 0);
```

Bingo! `opts` is the first element of `args` in `vl_argparse`. Therefore, at the very beginning when `ai = 1`, `isstruct(args{ai})` will also be `true`. So the next sentence is going to proceed:

```
    opts = vl_argparse(opts, vertcat(params,values)) ;
```

What are `opts`, `params` and `values` when program execution reaches here?

```
opts =

         imageSize: [227 227]
            border: [0 0]
        keepAspect: 1
       numAugments: 1
    transformation: 'none'
      averageImage: []
       rgbVariance: [0x3 single]
     interpolation: 'bilinear'
        numThreads: 1
          prefetch: 0
             scale: 1

params =

    'imageSize'    'keepAspect'    'averageImage'    'border'        'cropSize'      'interp
olation'    'numThreads'

values =

    [1x4 double]    [          1]                []    [1x2 double]    [1x2 double]    'biline
ar'          [         12]
```

where `params` is the `fieldnames` of `args{1}`: `params = fieldnames(args{ai})` and `values` packs their correponding values: `values = struct2cell(args{ai})`.

So the function `vl_argparse(opts, args, varagin)` is executed for a second time with parameters the same `opts` and a new `args = vertcat(params,values)`.

Now none of the elements in `args` is a struct. The execution will then reach the code block, where the ERROR occurs:

```
% ./matconvnet/matlab/vl_argparse.m

  param = args{ai} ;
  value = args{ai+1} ;
  p = find(strcmpi(param, optNames)) ;
  if numel(p) ~= 1
    if nargout == 1
        error('Unkno parameter ''%s''', param) ;
    else
      keep([ai,ai+1]) = true ;
      ai = ai + 2 ;
      continue ;
    end
  end
```

This code block is in a `while` loop. It finds the elements of `params` that appear in `optNames` at each iteration of the loop. So what is the `optNames`? It is a pack of all the fieldnames of `opts`. Having a comparison:

```
params =

    'imageSize'     'keepAspect'     'averageImage'     'border'     'cropSize'      'interpolat
ion'     'numThreads'

optNames =

    'imageSize'     'border'     'keepAspect'     'numAugments'     'transformation'     'averageI
mage'     'rgbVariance'     'interpolation'     'numThreads'     'prefetch'     'scale'
```

All elements but `cropSize` in `params` appear in `optNames` !!

That's why this error occurs:

```
Error: Unkown parameter 'cropSize'
```

So the problem is two fold:

```
`cropSize` should not appear in `net.meta.normalization`, or
`cropSize` should be made appear in `opts` of `vl_argparse`.
```

Since `net.meta.normalization` seems to be the last thing I should touch, I would try to set `cropSize` to `opts` firstly. To do this, let's have a check where the `opts` of `vl_argparse` comes from, which means we should look into the execution stack to find out who calls the function `vl_argparse`.

It is `imo = imdb_get_batch_bcnn(images, varargin)`:

```
% /bcnn/imdb_get_batch_bcnn.m
function imo = imdb_get_batch_bcnn(images, varargin)
% imdb_get_batch_bcnn  Load, preprocess, and pack images for BCNN evaluation
% For asymmetric model, the function preprocesses the images twice for two networks
% separately.

% OUTPUT
% imo: a cell array where each element is a cell array of images.
%       For symmetric bcnn model, numel(imo) will be 1 and imo{1} will be a
%       cell array of images
%       For asymmetric bcnn, numel(imo) will be 2. imo{1} is a cell array containing the prep
rocessed images for network A
%       Similarly, imo{2} is a cell array containing the preprocessed images for network B

%
% Copyright (C) 2015 Tsung-Yu Lin, Aruni RoyChowdhury, Subhransu Maji.
% All rights reserved.
```

```
%
% This file is part of the BCNN and is made available under
% the terms of the BSD license (see the COPYING file).
%
% This file modified from CNN_IMAGENET_GET_BATCH of MatConvNet

for i=1:numel(varargin{1})
    opts(i).imageSize = [227, 227] ;
    opts(i).border = [0, 0] ;
    opts(i).keepAspect = true;
    opts(i).numAugments = 1 ;
    opts(i).transformation = 'none' ;
    opts(i).averageImage = [] ;
    opts(i).rgbVariance = zeros(0,3,'single') ;
    opts(i).interpolation = 'bilinear' ;
    opts(i).numThreads = 1 ;
    opts(i).prefetch = false;
    opts(i).scale = 1;
    opts(i) = vl_argparse(opts(i), {varargin{1}(i),varargin{2:end}});
```

Oh — My — God !!
…
Sir, the source code is poisonous!!
…

The code has built-in bugs!

Anyway just add a line `opts(i).cropSize = 227/256;` and give it another shot.

```
opts(i).scale = 1;
opts(i).cropSize = 227/256; % added by jcsu, 2016/11/16
opts(i) = vl_argparse(opts(i), {varargin{1}(i),varargin{2:end}});
```

OK, Let's remove output directory `checkgpu` and see what will happen! ‘

```
>> run_experiments_bcnn_train
```

Matlab standard outputs:

```
dataset: classes: 200 in use. These are:
   1: 001.Black_footed_Albatross (train:    30, test:    30 total:    60)
   2: 002.Laysan_Albatross (train:    30, test:    30 total:    60)
   3: 003.Sooty_Albatross (train:    30, test:    28 total:    58)
   4: 004.Groove_billed_Ani (train:    30, test:    30 total:    60)
   ...
```

```
199: 199.Winter_Wren (train:    30, test:    30 total:    60)
 200: 200.Common_Yellowthroat (train:    30, test:    30 total:    60)
 Inf:       **total** (train: 11788, test: 11788 total: 11788)
dataset: there are 11788 images (5994 trainval 5794 test)
Initialization: extracting bcnn feature of batch 1/185
Initialization: extracting bcnn feature of batch 2/185
...
```

```
Initialization: extracting bcnn feature of batch 184/185
Initialization: extracting bcnn feature of batch 185/185
     layer|      0|         1|      2|
      type|  input|      conv|softmxl|
      name|    n/a|classifier|   loss|
----------|-------|----------|-------|
   support|    n/a|         1|      1|
  filt dim|    n/a|    262144|    n/a|
filt dilat|    n/a|         1|    n/a|
 num filts|    n/a|       200|    n/a|
    stride|    n/a|         1|      1|
       pad|    n/a|         0|      0|
----------|-------|----------|-------|
   rf size|    n/a|         1|      1|
 rf offset|    n/a|         1|      1|
 rf stride|    n/a|         1|      1|
----------|-------|----------|-------|
 data size|NaNxNaN|   NaNxNaN|NaNxNaN|
data depth|    NaN|       200|      1|
  data num|    256|       256|      1|
----------|-------|----------|-------|
  data mem|    NaN|       NaN|    NaN|
 param mem|    n/a|     200MB|     0B|

parameter memory|200MB (5.2e+07 parameters)|
     data memory|  NaN (for batch size 256)|
```

It works! But wait… Where is the `vl_nnbilinearpool`, `vl_nnsqrt` and `vl_nnl2norm` layers? Well, it is the first stage of training, i.e., training a softmax classifier.

附录-4 深入 B-CNN 源码

1. Having a quick overview.

`run_experiments_bcnn_train.m` under `/path/to/bcnn/` is a top-level function that performs the **two-step** training of a `B-CNN` model. Having a look into this function can give us a quick overview of the source code.

- **Overview of `run_experiments_bcnn_train.m`**:

```
            Entry :`run_experiments_bcnn_train()`
                      |
    Set Pretrained CNN(s): `bcnnmm`, `bcnnvdm`, or `bcnnvdvd` or all three
                      |
   Set Model Parameters and dataset: `[opts, imdb] = model_setup(/*parameter pairs*/)
                      |
    Begin to train a bcnn model: `imdb_bcnn_train_dag(imdb, opts)`
                      |
                     End
```

The first job of this function is to set the `B-CNN` model (`B-CNN[M,M]`, `B-CNN[D,D]`, or `B-CNN[M,D]`) to be trained and set the dataset to be trained on via struct/variables `bcnnxx.opts`, `setupNameList`, `encoderList` and `datasetList`:

```
  % run_experiments_bcnn_train.m

  bcnnmm.name = 'bcnnmm' ;
  bcnnmm.opts = {...
  'type', 'bcnn', ...
  'modela', 'data/models/imagenet-vgg-m.mat', ...     % intialize network A with pre-tr
ained model
  'layera', 14,...                                    % specify the output of certain l
ayer of network A to be bilinearly combined
  'modelb', 'data/models/imagenet-vgg-m.mat', ...     % intialize network B with pre-tr
ained model
  'layerb', 14,...                                    % specify the output of certain l
ayer of network B to be bilinearly combined
  'shareWeight', true,...                             % true: symmetric implementation
where two networks are identical
  } ;

  bcnnvdm.name = 'bcnnvdm' ;
  bcnnvdm.opts = {...
  'type', 'bcnn', ...
  'modela', 'data/models/imagenet-vgg-verydeep-16.mat', ...
  'layera', 30,...
  'modelb', 'data/models/imagenet-vgg-m.mat', ...
  'layerb', 14,...
  'shareWeight', false,...                              % false: asymmetric implementatio
n where two networks are distinct
  } ;

  bcnnvdvd.name = 'bcnnvdvd' ;
  bcnnvdvd.opts = {...
  'type', 'bcnn', ...
  'modela', 'data/models/imagenet-vgg-verydeep-16.mat', ...
  'layera', 30,...
  'modelb', 'data/models/imagenet-vgg-verydeep-16.mat', ...
  'layerb', 30,...
  'shareWeight', true,...
  };

  setupNameList = {'bcnnmm'};
  encoderList = {{bcnnmm}};
  % setupNameList = {'bcnnvdvd'};
  % encoderList = {{bcnnvdvd}};
  datasetList = {{'cub', 1}}
```

Then it calls function `model_setup` to set up hyper-parameters and prepare the dataset. Finally it calls function `imdb_bcnn_train_dag` to perform the *two-step* training.

- **Overview of `model_setup(...)`:**

```
        Entry:  `[opts, imdb] = model_setup(varargin)`
          |
Setup data structure `opts`: `batchSize`, `numEpochs`, `learningRate`, ...
          |
Setup data structure `opts.encoders`: what the hell is encoders?
          |
    Load dataset: `cub`, `cars`, or `aircraft`, or others
          |
         End: save imdb
```

Hyper-parameters for training is set default inside the `model_setup` function. We can change it directly inside `model_setup` or by passing a different value when the function `model_steup` is called by `run_experiments_bcnn_train`.

Then it loads the dataset specified in `datasetList`:

```
% /path/to/bcnn/model_setup.m

    switch opts.dataset
        case 'cubcrop'
            imdb = cub_get_database(opts.cubDir, true, false);
        case 'cub'
            imdb = cub_get_database(opts.cubDir, false, opts.useVal);
        case 'aircraft-variant'
            imdb = aircraft_get_database(opts.aircraftDir, 'variant');
        case 'cars'
            imdb = cars_get_database(opts.carsDir, false, opts.useVal);
        case 'imagenet'
            imdb = cnn_imagenet_setup_data('dataDir', opts.ilsvrcDir);
        case 'imagenet-224'
            imdb = cnn_imagenet_setup_data('dataDir', opts.ilsvrcDir_224);
        otherwise
            error('Unknown dataset %s', opts.dataset) ;
    end
```

- Overview of `imdb_bcnn_train_dag(imdb, opts)`:

```
              Entry: `imdb_bcnn_train_dag(imdb, opts)`
                |
     Setup some training params: `opts.train....`
                |
   Build up and Initialize network:
       `initializeNetworkSharedWeights` or ` initializeNetworkTwoStreams`
                |
   Applying SGD algorithm to train the bcnn
                |
               End: save networks
```

The *two-step* training process is performed inside `imdb_bcnn_train_dag.m`. It calls `initializeNetworkSharedWeights` or `initializeNetworkTwoStreams` to perform training step 1 and `bcnn_train_simplenn` or `bcnn_train_dag` to perform traning step 2. Some hyper-parameters of step 1 are also set inside this function.

2. Analysizing some typical functions in `bcnn-package` which implement layers of a `B-CNN` model.

- `function y = vl_nnbilinearpool(x, varargin)`

```
% /path/to/bcnn/bcnn-package/vl_nnbilinearpool.m

function y = vl_nnbilinearpool(x, varargin)
% ------------------------------------------------------------------------------------
% functionality:
%   implementation of the feed forward pass and backward pass of 'bilinearpool', whic
h
%   is a layer connected next to a pretrained CNN model (or two same model)
%
% $x$: input feature of size [height, width, channels, batch size].
% $varargin$: $dzdy$ when in backward pass.
% $y$:
%   forward pass:
%       Self outer product of $x$.
%       For each image with size `[height, width, channels]`, firstly
%       reshape it into size `[height * width, channels]`, and then
%       compute the output
%                   $$y = \frac{1}{height * width} x^T x$$
%       which gives $y$ the size `[channels, channels]`, reshape it again to a vector
.
%   backward pass:
%       gradient of $y$ w.r.t $x$.
%       $y$ is the same size as $x$, i.e., `[height, width, channels, batch_size]`.
%       For each image, reshape $dzdy$ to size `[channels, channels]`
%                       reshape $x$ to size `[height * width, channels]`.
%       $dydx$ is caculated as:
%                   $$y = \frac{1}{height * width} x * dzdy$$
%       which gives $y$ the size `[height * width, channels]`,
%       Reshape $y$ to `[height, width, channels]` as output.
% ------------------------------------------------------------------------------------

% if the number of elements in `varargin` > 0 and the first element is not a string
backMode = numel(varargin) > 0 && ~isstr(varargin{1})
```

```
% if in backward mode, take out the `dzdy` in `varargin`
if backMode dzdy = varargin{1}; end

% if `x` is a `gpuArray`, it is in `gpuMode`
gpuMode = isa(x, 'gpuArray');

% unpack the size of x into height, width, number of channel, and batch size
[h, w, ch, bs] = size(x);

% backward mode
if backMode
    if gpuMode
        y = gpuArray(zeros(size(x), 'single'));
    else
        y = zeros(size(x), 'single'));
    end
    % for each image / for each feature map with `ch` channels
    for b = 1:bs
        dzdy_b = reshape(dzdy(1, 1, :, b), [ch, ch]);
        a = reshape (x(:, :, :, b), [h*w, ch]);
        % caculate dydx
        y(:, :, :, b) = reshape(a * dzdy_b, [h, w, ch]) / (h * w);
    end
else
    if gpuMode
        y = gpuArray(zeros([1, 1, ch * ch, bs], 'single'));
    else
        y = zeros([1, 1, ch * ch, bs], 'single');
    for b = 1:bs
        a = reshape(x(:, :, :, b), [h * w, ch]);
        % caculate output
        y(1, 1, :, b) = reshape(a'*a, [1, ch*ch]) / (h * w);
    end
end
```

- `function y = vl_nnbilinearclpool(x1, x2, varargin)`

```
% /path/to/bcnn/bcnn-package/vl_nnbilinearclpool.m

function y = vl_nnbilinearclpool(x1, x2, varargin)
% --------------------------------------------------------------------------------------
% functionality:
%   implementation of the feed forward pass and backward pass of 'bilinearclpool', wh
ich
%   is a layer connected next to a pretrained CNN model (or two same model)
%
% $x$: input feature of size [height, width, channels, batch size].
% $varargin$: $dzdy$ when in backward pass.
% $y$:
%   forward pass:
%       Self outer product of $x$.
%       For each image with size `[height, width, channels]`, firstly
%       reshape it into size `[height * width, channels]`, and then
%       compute the output
%                   $$y = \frac{1}{height * width} x^T x$$
%       which gives $y$ the size `[channels, channels]`, reshape it again to a vector
.
%   backward pass:
%       gradient of $y$ w.r.t $x$.
```

```matlab
%       $y$ is the same size as $x$, i.e., `[height, width, channels, batch_size]`.
%       For each image, reshape $dzdy$ to size `[channels, channels]`
%                       reshape $x$ to size `[height * width, channels]`.
%       $dydx$ is caculated as:
%                   $$y = \frac{1}{height * width} x * dzdy$$
%       which gives $y$ the size `[height * width, channels]`,
%       Reshape $y$ to `[height, width, channels]` as output.
% ----------------------------------------------------------------------------------

% if the number of elements in `varargin` > 0 and the first element is not a string
backMode = numel(varargin) > 0 && ~isstr(varargin{1})

% if in backward mode, take out the `dzdy` in `varargin`
if backMode dzdy = varargin{1}; end

% if `x` is a `gpuArray`, it is in `gpuMode`
gpuMode = isa(x1, 'gpuArray');

% unpack the size of x into height, width, number of channel, and batch size
[h1, w1, ch1, bs] = size(x1);
[h2, w2, ch2, ~] = size(x2);

% resize the CNN output to the same size
if w1 * h1 <= w2 * h2
    % downsample feature 2
    x2 = array_resize(x2, w1, h1);
else
    % downsample feature 1
    x1 = array_resize(x1, w2, h2);
end
h = size(x1, 1); w = size(x1, 2);

% backward mode
if backMode
    if gpuMode
        y = gpuArray(zeros(size(x), 'single'));
    else
        y = zeros(size(x), 'single'));
    end
    % for each image / for each feature map with `ch` channels
    for b = 1:bs
        dzdy_b = reshape(dzdy(1, 1, :, b), [ch1, ch2]);
        A = reshape (x1(:, :, :, b), [h*w, ch1]);
        B = reshape (x2(:, :, :, b), [h*w, ch2]);
        dB = reshape(A * dzdy_b, [h, w, ch2]);
        dA = reshape(B * dzdy_b', [h, w, ch1]); %'
        if w1 * h1 <= w2 * h2
            % B is downsampled
        else
            % A is downsampled
    end
% feed forward pass
else
    if gpuMode
        y = gpuArray(zeros([1, 1, ch1 * ch2, bs], 'single'));
    else
        y = zeros([1, 1, ch1 * ch2, bs], 'single');
    for b = 1:bs
        xa = reshape(x1(:, :, :, b), [h * w, ch1]);
        xb = reshape(x2(:, :, :, b), [h * w, ch2]);
```

```
            y(1, 1, :, b) = reshape(xa'*xb, [1, ch1*ch2]); % why not '/(h *w)'?
        end
    end

    function Ar = array_resize(A, w, h)
    %-----------------------------------------
    % downsample A with size `[w, h]`
    %-----------------------------------------

    ...
```

- `function y = vl_nnsqrt(x, param, varagin)`

```
    % /path/to/bcnn/bcnn-package/vl_nnsqrt.m

    function y = vl_nnsqrt(x, param, varagin)
    % -------------------------------------------------------------------
    % functionality: perform square root normalization for the input features
    %                at each location
    %
    % x: the input features of size [height, width, channels, batch_size]
    % param: the threshold to prevent large value when close to 0
    % varargin: dzdy, only needed in backward pass
    % y:
    %     forward pass:
    %         y = sign(x) .* sqrt(|x|)
    %     backward pass:
    %         dydx = 0.5 ./ sqrt(|x| + param)
    %         y = dydx .* dzdy % the chain rule
    % -------------------------------------------------------------------
```

- `function y = vl_nnl2norm(x, param, varagin)`

```
    % /path/to/bcnn/bcnn-package/vl_nnl2norm.m

    function y = vl_nnl2norm(x, param, varagin)
    % -------------------------------------------------------------------
    % functionality: perform square root normalization for the input features
    %                at each location
    %
    % x: the input features of size [height, width, channels, batch_size]
    % param: the threshold to prevent large value when the norm is close to 0
    % varargin: dzdy, only needed in backward pass
    % y:
    %     forward pass:
    %         y = x ./ ||x||  % note: ||x|| is l-2 norm
    %     backward pass:
    %         \frac{d}{dx_j}(x./||x||) = \frack{1}{||x||^3}(x_1^2 + ... + x_{j-1}^2 + x_{
  j+1}^2 + ... + x_n^2)
    %         (d/dx_j)(x./||x||) = 1 / ||x|| - x_j^2 / ||x|| ^ 3
    %         gradient(x./||x||) = 1 / ||x|| - x.^2 / ||x|| ^ 3
    % -------------------------------------------------------------------
```

3. Looking into the code of preparing dataset.

Before using the birds dataset `cub`, have a look into its function `imdb = cub_get_database(cubDir, useCropped, useVal)`:

```
% cub_get_database.m
% the directory where the real images reside
if useCropped
    imdb.imageDir = fullfile(cubDir, 'images_cropped') ;
else
    imdb.imageDir = fullfile(cubDir, 'images');
end

...

% read the class names, image names, bounding boxes, lables...

[~, classNames] = textread(fullfile(cubDir, 'classes.txt'), '%d %s');
imdb.classes.name = horzcat(classNames(:));

% Image names
[~, imageNames] = textread(fullfile(cubDir, 'images.txt'), '%d %s');
imdb.images.name = imageNames;
imdb.images.id = (1:numel(imdb.images.name));

...

% if use validation, set 1/3 to validation set
if useVal
    rng(0)
    trainSize = numel(find(imageSet==1));

    trainIdx = find(imageSet==1);

    % set 1/3 of train set to validation
    valIdx = trainIdx(randperm(trainSize, round(trainSize/3)));
    imdb.images.set(valIdx) = 2;
end

...
```

Download the CUB-200-2011 dataset and unzip and find that all the required files are contained in the package, all we need to do is put the directory under `/path/to/bcnn_root/data` with a new name `cub`.

- Dataset details:
  - Birds: CUB-200-2011 dataset.
  - Aircrafts: FGVC aircraft dataset
  - Cars: Stanford cars dataset
- These results are with domain specific fine-tuning. For more details see the updated B-CNN tech report.

The job of `cub_get_database.m` is to build up a structure `imdb`:

```
imdb =

    imageDir: 'data/cub/images'
     maskDir: 'data/cub/masks'
        sets: {'train'  'val'  'test'}
     classes: [1x1 struct]
      images: [1x1 struct]
        meta: [1x1 struct]

imdb.images =

         name: {11788x1 cell}
           id: [1x11788 double]
        label: [1x11788 double]
       bounds: [4x11788 double]
          set: [1x11788 double]
    difficult: [1x11788 logical]
```

This structure seems quite clear. Only to note the member `imdb.images.set`, others are trivial.

```
% /path/to/bcnn_root/cub_get_database.m
    ...
    imdb.images.set(imageSet == 1) = 1; % 1  for training
    imdb.images.set(imageSet == 0) = 3; % 3  for test

    if useVal
        ...
        % set 1/3 of train set to validation
        valIdx = trainIdx(randperm(trainSize, round(trainSize/3)));
        imdb.images.set(valIdx) = 2; % 2 for validation
    end
    ...
```

That `imdb.images.set` contains a set of `set-lables`, one for each image. `1` lables the image for using in training, `2` for validation, and `3` for testing.

Let's trace the `imdb.iamges.set` to see how it is used in other files:

```
    $ grep imdb.images.set ./*.m
```

```
    ./bcnn_train_dag.m:if isempty(opts.train), opts.train = find(imdb.images.set==1) ; end
    ./bcnn_train_dag.m:if isempty(opts.val), opts.val = find(imdb.images.set==2) ; end
    ./bcnn_train_simplenn.m:if isempty(opts.train), opts.train = find(imdb.images.set==1) ; e
nd
    ./bcnn_train_simplenn.m:if isempty(opts.val), opts.val = find(imdb.images.set==2) ; end
    ./bird_demo.m:imageInd = find(imdb.images.label == classId & imdb.images.set == 1);
    ./imdb_bcnn_train_dag.m:train = find(imdb.images.set == 1) ;
    ./imdb_cnn_train.m:      train = find(imdb.images.set == 1) ;
    ./initializeNetworkSharedWeights.m:    train = find(imdb.images.set==1|imdb.images.set==2
);
    ./initializeNetworkTwoStreams.m:        train = find(ismember(imdb.images.set, [1 2]));
    ./model_train.m:        train = find(ismember(imdb.images.set, [1 2])) ;
    ./model_train.m:    train = ismember(imdb.images.set, [1 2]) ;
    ./model_train.m:    test = ismember(imdb.images.set, 3) ;
    ./print_dataset_info.m:train = ismember(imdb.images.set, [1 2]) ;
    ./print_dataset_info.m:test = ismember(imdb.images.set, [3]) ;
    ./run_experiments_bcnn_train.m:          imdb.images.set(imdb.images.set==3) = 2;
```

The lines with `./bcnn_train_dag.m` and `./bcnn_train_simplenn.m` are doing the jobs of extracting images with label `1` as training set and images with `2` as validation set. But none of the lines are related to label `3`, i.e., the test set, except three lines:

```
    ./model_train.m:    test = ismember(imdb.images.set, 3) ;
    ./print_dataset_info.m:train = ismember(imdb.images.set, [1 2]) ;
    ./print_dataset_info.m:test = ismember(imdb.images.set, [3]) ;
    ./run_experiments_bcnn_train.m:          imdb.images.set(imdb.images.set==3) = 2;
```

`model_train` is called by `run_experiments.m` and has no bussiness with `run_experiments_bcnn_train.m`. `print_dataset_info.m` prints out information about the dataset like:

```
    >> print_dataset_info(imdb)
        dataset: classes: 200 in use. These are:
           1: 001.Black_footed_Albatross (train:    60, test:     0 total:    60)
           2: 002.Laysan_Albatross (train:    60, test:     0 total:    60)
           3: 003.Sooty_Albatross (train:    58, test:     0 total:    58)
           4: 004.Groove_billed_Ani (train:    60, test:     0 total:    60)
```

What left is the last line:

```
./run_experiments_bcnn_train.m:          imdb.images.set(imdb.images.set==3) = 2;
```

which sets the test set to validation set!

Therefore, to leave out a test set, it is required to comment this line. In order to separate a validation set from training set, simply set `useVal` to `true` via params list in `model_setup`.

4. Inspecting the code of building up a B-CNN model.

The code of building up the actual `B-CNN` model lies in file `initializeNetworkSharedWeights.m` or `initializeNetworkTwoStreams.m`. Have a look into `initializeNetworkSharedWeights.m`:

```
% /path/to/bcnn/initializeNetworkSharedWeights.m

% ...

% Load the model
```

```
net = load(encoderOpts.modela);
net.meta.normalization.keepAspect = opts.keepAspect;

% truncate the network
maxLayer = max(encoderOpts.layera, encoderOpts.layerb);
net.layers = net.layers(1:maxLayer);

% ...

% stack bilinearpool layer
if(encoderOpts.layera==encoderOpts.layerb)
    net.layers{end+1} = struct('type', 'bilinearpool', 'name', 'blp');
else
    net.layers{end+1} = struct('type', 'bilinearclpool', 'layer1', encoderOpts.layera, 'layer
2', encoderOpts.layerb, 'name', 'blcp');
end

% stack normalization
net.layers{end+1} = struct('type', 'sqrt', 'name', 'sqrt_norm');
net.layers{end+1} = struct('type', 'l2norm', 'name', 'l2_norm');

 net.layers{end+1} = struct('type', 'relu', 'name', 'relu_wbcnn');

% build a linear classifier netc
initialW = 0.001/scal * randn(1,1,mapSize1*mapSize2,numClass,'single');
initialBias = init_bias.*ones(1, numClass, 'single');
netc.layers = {};
netc.layers{end+1} = struct('type', 'conv', 'name', 'classifier', ...
    'weights', {{initialW, initialBias}}, ...
    'stride', 1, ...
    'pad', 0, ...
    'learningRate', [1000 1000], ...
    'weightDecay', [0 0]) ;
netc.layers{end+1} = struct('type', 'softmaxloss', 'name', 'loss') ;
netc = vl_simplenn_tidy(netc) ;

% ...
```

In this code block, the pretrained CNN is loaded from path `encoderOpts.modela` and truncated at layer `encoderOpts.layera`. Then a `bilinearpool` layer followed by a `sqrt` normalization and `l2norm` normalization layers are stacked upon the truncated CNN building up a `Bilinear feature` extractor `net`.

A linear classifier `netc` is also created for classification. The classifier consists in a full-connected layer of size `[mapSize1*mapSize2, numClass]` followed by a `softmaxloss` layer, building up a `softmax` classifier.

The training step 1 is to train this classifier `netc`. Once the training completes, `netc` will be stacked upon `net` to build up a complete `B-CNN` model available for traning step 2.

5. Analysizing the code of training step 1.

The traning step 1 is performed by `initializeNetworkSharedWeights.m`.

```
% initializeNetworkSharedWeights.m

    ...

    % get bcnn feature for train and val sets
    train = find(imdb.images.set==1|imdb.images.set==2);

        ...

        % compute and cache the bilinear cnn features
        for t=1:batchSize:numel(train)

            ...

            batch = train(t:min(numel(train), t+batchSize-1));
            [im, labels] = getBatchFn(imdb, batch) ;

            netInit = net;

            ...

            net.layers{end}.class = labels ;

            res = [] ;
```

```
                res = vl_bilinearnn(netInit, im, [], res, ...
                    'accumulate', false, ...
                    'mode', 'test', ...
                    'conserveMemory', true, ...
                    'sync', true, ...
                    'cudnn', opts.cudnn) ;
                codeb = squeeze(gather(res(end).x));
                for i=1:numel(batch)
                    code = codeb(:,i);
                    savefast(fullfile(opts.nonftbcnnDir, ['bcnn_nonft_', num2str(batch(i), '%05d'
)]), 'code');
                end
            end
        end

    ...

            bcnndb = imdb;
            tempStr = sprintf('%05d\t', train);
            tempStr = textscan(tempStr, '%s', 'delimiter', '\t');
            bcnndb.images.name = strcat('bcnn_nonft_', tempStr{1}');
            bcnndb.images.id = bcnndb.images.id(train);
            bcnndb.images.label = bcnndb.images.label(train);
            bcnndb.images.set = bcnndb.images.set(train);
            bcnndb.imageDir = opts.nonftbcnnDir;

            %train logistic regression
            [netc, info] = cnn_train(netc, bcnndb, @getBatch_bcnn_fromdisk, opts.inittrain, ...
                'conserveMemory', true);

        end

    ...
```

For each image in training and validation set, this code block extracts the bilinear feature from it using network `net`. Given a batch of images, the `vl_bilinearnn` here extracts a bilinear feature for each of the images repectively and saves to disk.

The dataset `bcnndb` used for training network `netc` is then built up from these bilinear features. The training is performed by function `cnn_train`.

6. The code of training step 2.

Training step 2 is performed by `bcnn_train_simplenn` or `bcnn_train_dag` as indicated by the following code block:

```
% /path/to/bcnn/imdb_bcnn_train_dag.m

...

if simplenn
    fn_train = getBatchSimpleNNWrapper(train_bopts) ;
    fn_val = getBatchSimpleNNWrapper(val_bopts) ;
    [net,info] = bcnn_train_simplenn(net, imdb, fn_train, fn_val, opts.train, 'conserveMemory
', true) ;
    net = net_deploy(net) ;
    saveNetwork(fullfile(opts.expDir, 'fine-tuned-model', 'final-model.mat'), net, info);
else
    fn_train = getBatchDagNNWrapper(train_bopts, useGpu) ;
    fn_val = getBatchDagNNWrapper(val_bopts, useGpu) ;
    opts.train = rmfield(opts.train, {'sync', 'cudnn'}) ;
    [net, info] = bcnn_train_dag(net, imdb, fn_train, fn_val, opts.train) ;
    net = net_deploy(net) ;
    save(fullfile(opts.expDir, 'fine-tuned-model', 'final-model.mat'), 'net', 'info', '-v7.3'
);
end

...
```

The function `bcnn_train_simplenn` is modified from `cnn_train` of **MatConvNet**, which implementes the SGD algorithm for training a simple network.

The function `bcnn_train_dag` is modified from `cnn_train_dag` of **MatConvNet**, which implementes the SGD algorithm for training a complex network.

7. Additional training step.

Once training step 2 is done, the fine-tuning is said to be done. However, there should be one additional training step to train the whole network combining the training and validation dataset.
In this step, authors of the paper chop off the softmax classifier, i.e., `netc` in `initializeNetworkSharedWeights`, and replace it with a set of linear SVMs, which should call for one more traning step.

After the additional training step, the test set had been left out is used for evaluating the accuracy of the final model.

Note that training SVMs in this additional step consumes more than 32GB main memory of CPU.

8. Miscellaneous

- `vl_bilinearnn` is modified from `vl_simplenn` of **MatConvNet** and implements both the forward pass and backward pass of a network.
- `imdb_get_batch` is used for fetching a batch of examples along with labels from dataset `*.imdb`. The output batch is ready to be passed to `vl_bilinearnn` for training or validation.
- `run_experiments` is used for training SVMs after training step 2.

附录-5 修改 B-CNN，尝试改进

5.1 Motivation.

Given two feature extractors f_A and f_b, denote their output features as two matrices X_a and X_b respectively (they must have the same number of rows), where each column of them is a reshaped feature map. The B-CNN model proposed by Tsung-Yu Lin simply takes their matrix product $X_a^T X_b$ as the bilinear-pooled form. We observe that it can also be written as $X_a^T I X_b$, where I is an *identity matrix*, which brings up the question: does I is the optimal matrix? Holding the view that *identity matrix* is general to any matrix, i.e., contains no particular information and changes nothing, we would like to ask, can we train some matrices that contain some particular information for a specific dataset? It seems convincing that the identity matrix should not be the optimal and there should be some matrices that bring information about a given dataset to be optimal.

Holding this belief, we replace the identity matrix with a weight matrix and let the training algorithm learn a specific matrix for a given dataset.

5.2 Revised bilinear model.

$$Y = X_a^T W X_b,$$

where

$$X_a \in R^{P_a \times CH_a}, \ \ X_b \in R^{P_b \times CH_b}, \ \ W \\ \in R^{P_a \times P_b}, \ \ Y \in R^{CH_a \times CH_b}$$

The gradients w.r.t X_a, X_b, and W are

- $\nabla_{X_a} Z = W X_b \cdot (\nabla_Y Z)^T, \quad$ (size: $P_a \times P_b \cdot P_b \times CH_b \cdot CH_b \times CH_a$) $= P_a \times CH_a$
- $\nabla_{X_b} Z = W^T X_a \cdot (\nabla_Y Z), \quad$ (size: $P_b \times P_a \cdot P_a \times CH_a \cdot CH_a \times CH_b$) $= P_b \times CH_b$
- $\nabla_W Z = X_a \cdot (\nabla_Y Z) \cdot X_b^T. \quad$ (size: $P_a \times CH_a \cdot CH_a \times CH_b \cdot CH_b \times P_b$) $= P_a \times P_b$

5.3 Application

Considering two sets of feature maps output from some CNNs with sizes $W_a \times H_a \times CH_a$ and $W_b \times H_b \times CH_b$ respectively. The model can be applied easily by reshaping them to $(W_a * H_a) \times CH_a$ and $(W_b * H_b) \times CH_b$.

5.4 Add a self-defined layer `vl_weightedbilinearpool` to `matconvnet`

1. **Define the layer `vl_weightedbilinearpool.m`**

   The first step to add a self-defined layer to `matconvnet` is to define a function implementing the layer's functionalities of *feed forward* and *back propagation* with given inputs.

   The following is what the `vl_weightedbilinearpool.m` may look like.

```
% /path/to/bcnn/bcnn-package/vl_weightedbilinearpool.m

function [y, varargout] = vl_weightedbilinearpool(x1, x2, W, varargin)
% VL_WEIGHTEDBILINEARPOOL implementes the revised bilinear model with a weights matrix
%
% Copyright (C) 2016 Jincheng Su @ Hikvision.
% All rights reserved.
%
% * **Feed forward**
%   * Input: `x1` and `x2`, with shapes `[h1, w1, ch1, bs]` and `[h2, w2, ch2, bs]`.
%            `W`, the weights
%   * Output: `y`, with shape `[ch1*ch2, bs]`.
%
% * **Back propagation**
%   * Input: `x1`, `x2` and `W` are the same as in forward pass,
%            dzdy = varargin{1}, is the derivative of loss `z` w.r.t `y`.
%   * Output: y, the derivative of loss `z` w.r.t `x1`, i.e., dzdx1.
%             varargout{1} = y2, the derivative of loss `z` w.r.t `x2`, i.e., dzdx2.
%             varargout{2} = dw, the derivative of loss `z` w.r.t `W`, i.e., dzdW.
%
% -----------------------
% The revised B-CNN model
% -----------------------
%
%  $$ Y = X_a^TWX_b,$$
%
% where $I$ is an identity matrix and
%    $$X_a \in R^{P_a\times CH_a},~~X_b \in R^{P_b\times CH_b}, ~~W\in R^{P_a\times P
_b}, ~~Y\in R^{CH_a\times CH_b}$$
%
% The derivatives w.r.t $X_a$, $X_b$, and $W$ are
% * $\nabla_{X_a}Z = WX_b\cdot(\nabla_YZ)^T$, $~~~~$(size: $P_a\times P_b \cdot P_b\ti
mes CH_b\cdot CH_b\times CH_a = P_a\times CH_a$)
% * $\nabla_{X_b}Z = W^TX_a\cdot(\nabla_YZ)$,  $~~~~$(size: $P_b\times P_a \cdot P_a\t
imes CH_a\cdot CH_a\times CH_b = P_b\times CH_b$)
% * $\nabla_{W}Z = X_a\cdot(\nabla_YZ)\cdot X_b^T$.  $~~~$(size: $P_a\times CH_a \cdot
 CH_a\times CH_b\cdot CH_b\times P_b = P_a\times P_b$)
%
% ---------------------------------------------------------------------------------------
------------------

% flag for doing backward pass
isBackward = numel(varargin) > 0 && ~isstr(varargin{1});
if isBackward
    dzdy = varargin{1};
end

% if GPU is used
gpuMode = isa(x1, 'gpuArray');

% [height, widht, channels, batchsize]
```

```
[h1, w1, ch1, bs] = size(x1);
[h2, w2, ch2, ~ ] = size(x2);

if ~isBackward
    % forward pass
    if gpuMode
        y = gpuArray(zeros([1, 1, ch1 * ch2, bs], 'single'));
    else
        y = zeros([1, 1, ch1 * ch2, bs], 'single');
    end

    for b = 1: bs
        Xa = reshape(x1(:,:,:,b), [h1 * w1, ch1]);
        Xb = reshape(x2(:,:,:,b), [h2 * w2, ch2]);

        y(1, 1, :, b) = reshape(Xa'*W*Xb, [1, ch1 * ch2]); %'
    end
else
    % backward pass
    if gpuMode
        y1 = gpuArray(zeros(h1, w1, ch1, bs, 'single'));
        y2 = gpuArray(zeros(h2, w2, ch2, bs, 'single'));
        dw = gpuArray(zeros(h1*w1, h2*w2,'single'));
    else
        y1 = (zeros(h1, w1, ch1, bs, 'single'));
        y2 = (zeros(h2, w2, ch2, bs, 'single'));
        dw = (zeros(h1*w1, h2*w2,'single'));
    end

    for b = 1: bs
        dZdY = reshape(dzdy(1, 1, :, b), [ch1, ch2]);
        Xa = reshape(x1(:,:,:,b), [h1 * w1, ch1]);
        Xb = reshape(x2(:,:,:,b), [h2 * w2, ch2]);
        dZdXa = reshape(W*Xb*dZdY', [h1, w1, ch1]);
        dZdXb = reshape(W'*Xa*dZdY, [h2, w2, ch2]);
        dZdW = Xa*dZdY*Xb'; %'

        y1(:, :, ;, b) = dZdXa;
        y2(:, :, :, b) = dZdXb;
        dw = dw + dZdW;
    end
    y = y1;
    varargout{1} = y2;
    varargout{2} = dw / bs;
end
```

2. **Register the layer to `vl_simplenn.m`**

Since I always hold the belief of never changing the source code easily, I would leave `vl_simplenn.m` untouched but rather revise its alternative `vl_bilinearnn.m`.

```
$ cp /path/to/bcnn/bcnn-package/vl_bilinearnn.m /path/to/bcnn/bcnn-package/vl_bilinear
nn.m_bak
$ gvim /path/to/bcnn/bcnn-package/vl_bilinearnn.m
```

```
% vl_bilinearnn.m

    ...
```

```
     % forward pass
         case 'pdist'
           res(i+1) = vl_nnpdist(res(i).x, l.p, 'noRoot', l.noRoot, 'epsilon', l.epsilo
n) ;
         case 'bilinearpool'
           res(i+1).x = vl_nnbilinearpool(res(i).x);
         case 'bilinearclpool'
           x1 = res(l.layer1+1).x;
           x2 = res(l.layer2+1).x;
           res(i+1).x = vl_nnbilinearclpool(x1, x2);

         % this is my layer. As a first try, I simply assume the input `x1` and `x2` ar
e the same
         case 'weightedbilinearpool'
           res(i+1).x = vl_weightedbilinearpool(res(i).x, res(i).x, l.weights{1});

         case 'sqrt'
           res(i+1).x = vl_nnsqrt(res(i).x, 1e-8);
         case 'l2norm'
           res(i+1).x = vl_nnl2norm(res(i).x, 1e-10);
         case 'custom'

     ...

     % backward pass
         case 'bilinearclpool'
             x1 = res(l.layer1+1).x;
             x2 = res(l.layer2+1).x;
             [y1, y2] = vl_nnbilinearclpool(x1, x2, res(i+1).dzdx);
             res(l.layer1+1).dzdx = updateGradient(res(l.layer1+1).dzdx, y1);
             res(l.layer2+1).dzdx = updateGradient(res(l.layer2+1).dzdx, y2);

         % this is my layer. As a first try, I simply assume the input `x1` and `x2` ar
e the same
         case 'weightedbilinearpool'
             [y1, y2, dzdw{1}] = vl_weightedbilinearpool(res(i).x, res(i).x, l.weights{
1}, res(i+1).dzdx);
             res(i).dzdx = updateGradient(res(i).dzdx, y1 + y2);
             clear y1 y2

         case 'sqrt'
             backprop = vl_nnsqrt(res(i).x, 1e-8, res(i+1).dzdx);
             res(i).dzdx = updateGradient(res(i).dzdx, backprop);
             clear backprop
     ...

     % Add our type.
     switch l.type
         case {'conv', 'convt', 'bnorm', 'weightedbilinearpool'}
             if ~opts.accumulate
                 res(i).dzdw = dzdw ;
             else
                 for j=1:numel(dzdw)
                     res(i).dzdw{j} = res(i).dzdw{j} + dzdw{j} ;
                 end
             end
             dzdw = [] ;
     end
```

3. **Our work of adding a layer to `matconvnet` is almost done**. However, one another small revision is

indispensable for enabling our layer to run on a GPU unbuggly when using `vl_bilinearnn.m`. That is, add our layer to the `vl_simplenn_move.m`.

```
% /path/to/bcnn/matconvnet/matlab/simplenn/vl_simplenn_move.m
for l=1:numel(net.layers)
    switch net.layers{l}.type
        case {'conv', 'convt', 'bnorm', 'weightedbilinearpool'}
            for f = {'filters', 'biases', 'filtersMomentum', 'biasesMomentum'}
```

5.5 Build our `WB-CNN` model for training.

Now that we have added a self-defined layer with type `weightedbilinearpool` to `matconvnet`, we need to build a network to test if it works correctly or if it ever works. I'll do it by revised the `B-CNN` model.

As a first try, let's implements a simple symmetric `WB-CNN`, which can be built easily by replacing the `bilinearpool` layer in a symmetric `B-CNN` with our `weightedbilinearpool` layer.

1. **Modify `initializeNetworkSharedWeights.m`**

```
 $ cp /path/to/bcnn/initializeNetworkSharedWeights.m /path/to/bcnn/initializeNetworkWei
ghtedBcnn.m
 $ gvim /path/to/bcnn/initializeNetworkWeightedBcnn.m
```

```
 % initializeNetworkWeightedBcnn.m

     ...

     % build a linear classifier netc
     netc.layers = {};

     h1 = 27;
     w1 = 27;
     bilinearW = 0.001/scal * randn(h1*w1, h1*w1, 'single');
     % stack weighted bilinearpool layer
     netc.layers{end+1} = struct('type', 'weightedbilinearpool', 'name', 'wblp', ...
         'weights',{{bilinearW}} , ...
         'learningRate', [1000], ...
         'weightDecay', [0.9]);

     % stack a dropout layer, `rate` is defined as a probability of a varaiable not to
be zeroed.
     netc.layers{end+1} = struct('type', 'dropout', 'name', 'dropout_wbcnn',...
         'rate', 0.3);

     % stack a relu layer
      netc.layers{end+1} = struct('type', 'relu', 'name', 'relu6');

     % stack normalization
     netc.layers{end+1} = struct('type', 'sqrt', 'name', 'sqrt_norm');
     netc.layers{end+1} = struct('type', 'l2norm', 'name', 'l2_norm');

     % stack classifier layer
     initialW = 0.001/scal * randn(1,1,ch1*ch2,numClass,'single');
     initialBias = init_bias.*ones(1, numClass, 'single');
     netc.layers{end+1} = struct('type', 'conv', 'name', 'classifier', ...
         'weights', {{initialW, initialBias}}, ...
         'stride', 1, ...
         'pad', 0, ...
```

```
        'learningRate', [1000 1000], ...
        'weightDecay', [0 0]) ;
    netc.layers{end+1} = struct('type', 'softmaxloss', 'name', 'loss') ;
    netc = vl_simplenn_tidy(netc) ;

    ...

                codeb = squeeze(gather(res(end).x));
                for i=1:numel(batch)
                    %
                    code = codeb(:, :, :, i);
    %               code = reshape(codeb(:, :, :, i), size(codeb, 1)*size(codeb, 2),
size(codeb, 3));
                    savefast(fullfile(opts.nonftbcnnDir, ['bcnn_nonft_', num2str(batch
(i), '%05d')]), 'code');
                end
            end
        end

    ...

    function [im,labels] = getBatch_bcnn_fromdisk(imdb, batch)
    % -------------------------------------------------------------------------

    imtmp = cell(1, numel(batch));
    for i=1:numel(batch)
        load(fullfile(imdb.imageDir, imdb.images.name{batch(i)}));
        imtmp{i} = code;
    end
    h = size(imtmp{1}, 1);
    w = size(imtmp{1}, 2);
    ch = size(imtmp{1}, 3);
    im = zeros(h, w, ch, numel(batch));
    for i = 1:numel(batch)
        im(:, :, :, i) = imtmp{i};
    end
    clear imtmp
    %im = cat(2, im{:});
    %im = reshape(im, size(im, 1), size(im, 2), size(im,3), size(im, 4));
    labels = imdb.images.label(batch) ;
```

2. **Open `/path/to/bcnn/matconvnet/examples/cnn_train.m` and replace the `vl_simplenn` with `vl_bilinearnn`.**

```
 %      res = vl_simplenn(net, im, dzdy, res, ...
 %                        'accumulate', s ~= 1, ...
 %                        'mode', evalMode, ...
 %                        'conserveMemory', params.conserveMemory, ...
 %                        'backPropDepth', params.backPropDepth, ...
 %                        'sync', params.sync, ...
 %                        'cudnn', params.cudnn, ...
 %                        'parameterServer', parserv, ...
 %                        'holdOn', s < params.numSubBatches) ;

      res = vl_bilinearnn(net, im, dzdy, res, ...
                        'accumulate', s ~= 1, ...
                        'mode', evalMode, ...
                        'conserveMemory', params.conserveMemory, ...
                        'backPropDepth', params.backPropDepth, ...
                        'sync', params.sync, ...
                        'cudnn', params.cudnn) ;
```

5.6 Traning Method and Experimental results and Discussion.

We use the same training method as that in Tsung-Yu's paper. As a first try, we use the simple model **WB-CNN[M,M]** and train it on **birds** dataset for comparison.

Since **training step 1** is in fact solving a convex optimization problem, we believe that once it reaches a local optima it reaches the global optima. After some experiments, we find that with the learning rate setting to 0.06, weight decay to 0.9, it reaches a local optima quickly in 10 epochs.

Therefore we run the traning step 1 for 15 epochs using learning rate as 0.06, which gives us the best error rate smaller than 0.340. As a comparison, with the same learning parameters, the un-weighted bilinear model in Tsung-Yu's paper only reaches error rate larger than 0.39.

On **traning step 2**, we reduce the learning rate to 0.006, and train for 100 epochs, the best error rate on this epoch is no larger than 0.250. As a comparison, with learning rate setting to 0.001 and traning for 100 epochs, the un-weighted bilinear model only reaches error rate larger than 0.280.

On **traning step 3**, we combine the traning and validation set and use the same learning parameters, and train it for another 100 epochs. It gives the best performce of 78.70% in terms of per-image accuracy. As a comparison, the best performance of our reproduced work of the original work is 77.56%.

One thing to note is that since the identity matrix gives the original paper excellent performance, it inspires us to **initialize our matrix W with an identity matrix**. By letting it begin with an identity matrix, we would like the training algorithm to find if there is a better matrix and if not, we hope the algorithm find its way back to the identity matrix. In fact, we also have experimented with a random matrix generated by a scaled gaussian distribution and it gives a poor result of accuracy smaller than 76%.

5.7 Another variation of revision

Another attemptation we made is to constrain the matrix to be symmetric by formulating it as $Y = X_a^T(W^T + W)X_b$. The gradient is correspondingly $\nabla_W Y = X_a(\nabla_Y Z + (\nabla_Y Z)^T X_b$. The observation is that by decomposing $X_a = (a_1, a_2, \ldots, a_m)^T, X_b$ and rewritting Y as

$= (b_1, b_2, \ldots, b_n)^T$

$$Y = (a_1, a_2, \ldots, a_m)W(b_1, b_2, \ldots, \\ b_n)^T = \sum_{i,j} w_{ij} a_i b_j$$

Note that when X_a equals X_b which is the condition in our simple try, $a_i b_j$ equals $a_j b_i$, which means w_{ij} should equal to w_{ji}, i.e., W should be symmetric.

However, experiments show that this attemptation does not help.

What's worse, experiments with the **WB-CNN[D,D]** do not show any improvement over the orignal un-weighted model.

Still one another attemptation is that we formualte it as $Y = X_a(W + I)X_b$, and initialize W randomly, however this dosen't show any improvement as well. We look into the trained matrix W and find that it learns very small weights, all with order 1e-3, which seems to indicate that the identity matrix is actually the best…

However, experimental results with step one seem to indicate that our revised models is superior to the original model. However, the benefit of our revised models is eliminated by the feature extractors. Why?

Observing that our weights matrix W is 729×729 for the WB-CNN[M,M] model, which is a lot of parameters, maybe it is because it overfits easily since our birds dataset is too small? How about try some dropout?

However, experiments with different dropout rates (0.7, 0.5, 0.3, 0.1) do not show any help and some even hurt.

Anyway the experiments with traning step 1 and 2 demonstrate that our revised models are superior to the original model at least in some aspects. As to what aspects they remain unknown.

5.7 Conclusion

In the revised model, we tried different formulations of bilinear models such as $Y = X_a^T W X_b$,

$Y = X_a^T (W + I) X_b$ or $Y = X_a^T (W^T + W) X_b$ and try with WB-CNN[M,M] and WB-CNN[D,D] models on the birds dataset. Experimental results show that the revised models improve the B-CNN[M,M] model slightly while do not help the B-CNN[D,D] model on the birds dataset.

附录-6 深入探究 Bilinear 与 Covariance 的关系

附录-7 BoVW, Fisher Vector 和 VLAD 简介

7.1 BoVW (bag-of-visual-word [6])

术语:

- 视字 (*visual word*)
- 关键点 (*keypoint*)
- 视字本 (*vocabulary of visual word*)
- 关键点包 (*bag of keypoints*)
- 矢量量化 (*vector quantization*)

主要想法:

BoVW 方法受文本分类启发而提出, 用于图像分类.

考虑简单的邮件二分类任务, 需要将一封邮件分类为垃圾 (spam) 或 非垃圾邮件. 一个方法是选取一个单词本 $v = (v_1, \ldots, v_i, \ldots, v_n)$, 其中 v_i 表示不同的单词, n 是总单词数. 对于一封待分类的邮件, 统计单词本中的各个单词在该邮件中出现的次数, 得到一个统计量 $x = (x_1, \ldots, x_i, \ldots, x_n)$, i指代单词本中的单词 v_i , x_i 表示了这个单词在邮件中出现的次数. 将这个统计量x作为这封邮件的特征向量, 则可以训练一个多分类器对其进行分类. 用于文本分类的朴素贝叶斯方法就是这样做的.

将这种方法应用于图像分类首先要解决的问题是单词本里面的单词是什么， 如何选取一个合适的单词本. BoVW 方法的做法是, 从训练集中提取 SIFT 特征, 然后利用矢量量化 (vector quantization) 算法对这些特征进行聚类, 然后将聚类中心作为单词, 称为视字 (visual word) 或关键点 (keypoints), 所有聚类中心的组合即是单词本, 称为视字本 (vocabulary of visual word). 对于一个待分类图像, 首先从中提取 SIFT 特征, 然后将其分配到各个聚类中心即关键点上, 统计各个关键点分配到的 SIFT 特征数得到一个直方图, 称为关键点包 (bag of keypoints). 这个关键点包被作为这个图片的特征向量. 然后可以用一个训练好的多分类器对其进行分类.

分类步骤:

1. 对于一张图片, 提取它的 SIFT 特征集, 记为 $x = (x_1, \ldots, x_i, \ldots, x_s)$, 其中 $x_i \in R^{128}$ 是一个 SIFT 特征, s 表示 SIFT 特征数.
2. 把各个 SIFT 特征分配到预训练好的视字本中距离最近的条目上去. 该距离可以是欧拉距离, 该视字本通过矢量量化算法得到.
3. 构造一个关键点包. 一个关键点包是一个统计视字本上各个视字分配到的 SIFT 特征数的直方图 $b = (b_1, \ldots, b_i, \ldots, b_n)$, b_i 表示被分配到视字上的 SIFT 特征数, n 表示视字本条目数.
4. 将该关键点包作为特征向量, 应用多分类器 (朴素贝叶斯或者SVM) 进行分类.

视字本的训练过程通常是首先从训练集的图片中提取 SIFT 特征, 然后将所有的 SIFT 特征利用 *k-means* 算法进行矢量量化得到 n 个聚类中心, 即为视字本 (vocabulary of visual word). 后来的文章将 *k-means* 硬聚类换成了如 *GMM* 等软聚类.

7.2 FV (Fisher Vector [12])

7.2.1 Fisher Vector

记 $X = \{x_1, \ldots, x_i, \ldots, x_n\}$ 为从一张图片中提取的 n 个局部特征, 比如可以是 SIFT 特征. 假设这些特征相互独立, 虽然这个假设是不准确的, 特别是有些 SIFT 特征对应的原图像区域可能会有重叠.

设用 $p(X; \theta)$ 对这些特征的概率密度函数进行建模, 其中 $\theta \in R^m$ 是该函数的 m 个参数组成的向量.

1. **评分函数 (score function)**

   评分函数 (sc function) 即是对数似然函数对参数的梯度:

$$S_\theta^X = \nabla_\theta log\, p(X;\theta).$$

S_θ^X 描述了如何对概率密度函数模型参数进行调整从而更好地拟合数据. 注意 $S_\theta^X \in R^m$, 其维度由模型参数个数决定.

2. **Fisher Information Matrix (FIM)**

   Fisher Information Matrix (FIM) $F_\theta \in R^{m\times m}$ 定义在评分函数之上:

$$F_\theta = E_{X\sim p(X;\theta)}[S_\theta^X (S_\theta^X)^T].$$

3. **Fisher Kernel (FK)**

   于是可以用 Fisher Kernel (FK) 来衡量两个样本 Y 和 Z 之间的相似性:

$$K_{FK}(Y,Z) = (S_\theta^Y)^T F_\theta^{-1}(S_\theta^Z).$$

4. **Fisher Vector (FV)**

   由于 F_θ 是对称半正定矩阵, 它的逆矩阵也对称半正定. 于是可以对之应用 *Cholesky* 分解： $F_\theta^{-1} = L_\theta^T L_\theta$.

   于是， $K_{KF}(Y,Z)$ 可以重写成:

$$K_{KF}(Y,Z) = (\mathcal{S}_\theta^Y)^T \mathcal{S}_\theta^Z.$$

   其中,

$$\mathcal{S}_\theta^Y = L_\theta S_\theta^Y = L_\theta \nabla_\theta\, log\, p(Y;\theta).$$

   该规范化的梯度向量即是 Y 的 **Fisher Vector** (FV).

7.2.2 深入理解 Fisher Information Matrix (FIM)

1. **FIM 是评分函数的方差**

   我们看看评分函数的期望(以下推导需要一定的条件):

$$\begin{aligned} E[S_\theta^X] &= E[\nabla_\theta\, log\, p(X;\theta)] \\ &= E\left[\frac{\nabla_\theta\, p(X;\theta)}{p(X;\theta)}\right] \\ &= \int_x \frac{\nabla_\theta\, p(x;\theta)}{p(x;\theta)}\, p(x;\theta)\, dx = \\ &\int_x \nabla_\theta\, p(x;\theta)\, dx \\ &= \nabla_\theta \int_x p(x;\theta)\, dx = \nabla_\theta\, 1\, dx = 0. \end{aligned}$$

   因此有,

$$\begin{gathered} E[(S_\theta^X)(S_\theta^X)^T] \\ = E[(S_\theta^X - E[S_\theta^X])(S_\theta^X - E[S_\theta^X])^T]. \end{gathered}$$

   可见 **FIM 即是评分函数的方差**. 注意 FIM 并非某次观察结果的函数, 而是其期望. 评分函数的绝对值越大, 方差越大, 于是 FIM 携带的信息就越多.

2. **FIM 是评分函数的一阶导的负期望, 是对数似然函数二阶导的负期望**

   从另一方面看, 由于

$$
\begin{aligned}
\nabla_\theta S_\theta^X &= \nabla_\theta^2 \, log\, p(X;\theta) \\
&= \nabla_\theta \left(\frac{\nabla_\theta \, p(X;\theta)}{p(X;\theta)} \right) \\
&= \frac{\nabla_\theta^2 \, p(X;\theta)}{p(X;\theta)} \\
&- \left(\frac{\nabla_\theta \, p(X;\theta)}{p(X;\theta)} \right) \left(\frac{\nabla_\theta \, p(X;\theta)}{p(X;\theta)} \right)^T \\
&= \frac{\nabla_\theta^2 \, p(X;\theta)}{p(X;\theta)} - \left(S_\theta^X\right) \left(S_\theta^X\right)^T
\end{aligned}
$$

在一定条件下，容易证明 $E\left[\frac{\nabla_\theta^2 \, p(X;\theta)}{p(X;\theta)}\right] = 0$, 因此有:

$$
E[\nabla_\theta S_\theta^X] = E[\nabla_\theta^2 \, log\, p(X;\theta)] = \\
-E\left[\left(S_\theta^X\right)\left(S_\theta^X\right)^T\right]
$$

可见**FIM 是评分函数的一阶导的负期望, 是对数似然函数二阶导的负期望**. 因此, FIM 可以看作是在对数似然函数在参数θ附近的曲率估计. 该曲率越高窄, 说明参数θ拟合得越好.

7.2.3 Fisher Vector 应用于图像分类

由于高斯混合模型 (GMM) 可以以任意精度近似任何连续分布, 在图像分类中, 通常用作为上面提到的概率密度函数 $p(X;\theta)$. 于是将 FV 应用于图像分类的主要步骤通常为:

1. 从训练集中提取所有图片的 SIFT 特征, 每个特征是一个128维的向量.
2. 假设给定一张图片的各个 SIFT 特征相互独立. 在从训练集中提取的所有 SIFT 特征上，利用 EM 算法训练一个方差为对角阵的高斯混合模型(GMM)

   $$p(x;w,\mu,\Sigma) = \sum_{j=1}^{k} w_j p(x;\mu_j,\Sigma_j),$$

   其中 $w = (w_1,\ldots,w_k)$ 是 k 个高斯函数的权重. $\mu = (\mu_1,\ldots,\mu_k)$ 和 $\Sigma = (\Sigma_1,\ldots,\Sigma_k)$ 是各个高斯函数的均值和方差矩阵.

   假设这些方差矩阵为对角阵是一个标准的假设.
3. 对于一幅图片, 首先提取其 SIFT 特征

   $$X = \{x_1,\ldots,x_i,\ldots,x_n\},$$

   计算 GMM 模型

   $$p(x_i;w,\mu,\Sigma) = \sum_{j=1}^{k} w_j p(x_i;\mu_j,\Sigma_j),$$

   然后求 X 的评分函数 (假设一幅图片的各个 SIFT 特征相互独立, 虽然该假设有偏颇)

   $$
   S_\theta^X = \nabla_\theta \, log\, p(X;w,\mu,\Sigma) = \\
   \sum_{i=1}^{n} \nabla_\theta \, log\, p(x_i;w,\mu,\Sigma)
   $$

   即X的对数似然函数对参数的梯度 ∇_w , ∇_μ , 和 ∇_Σ . 用 $1/\sqrt{w} = (1/\sqrt{w_1},\ldots,1/\sqrt{w_k})^T$ 对评分函数进行标准化(normalization). 注意, 在一定的假设下, FIM 成为一个对角阵, 该标准化操作意味着FIM $F_\theta = diag(w)$.

   最终得到这样一个FV向量 $FV = \frac{1}{\sqrt{w}}(\nabla_w \, , \nabla_\mu \, , \nabla_\Sigma \,)$. 假设高斯混合模型由$k$ 个高斯函数加权混合而成, 每

个高斯函数的均值和方差均为 D 维, 那么最终得到的FV向量为 $(2D+1)k$ 维. 注意，方差为对角阵, 可以用一向量表示.

4. 该 FV 向量便可以作为一幅图片的特征向量, 用多分类器对其进行分类.

7.3 VLAD (vector of locally aggregated descriptor [13])

VLAD 像 BoVW 一样, 先用 $k-means$ 算法学习一个包含 k 个字的视字本 (vocabulary of visual word) $\mathcal{C}=(c_1,\ldots,c_k)$.

图片的每一个局部特征 x 都会被分配到与它距离最近的视字 (visual word) $c_i = NN(x)$.

VLAD 的关键想法是 将分配到视字 c_i 的所有局部特征 x 到该视字的距离 $v_i = \sum_{x \ s.t. \ NN(x)=c_i} x - c_i$ 累加起来.

假设局部特征 x 具有 d 维, 视字本有 k 个字, 那么整个图像的全局特征就是 $k \times d$ 维的.

如果用下标 j 索引向量 v_i, x, c_i 的各个分量, 那么

$$v_{i,j} = \sum_{x \ s.t. \ NN(x)=c_i} x_j - c_{i,j}, \ j=1, ..\\ ., d.$$

最终得到的向量 $v=(v_1, \ldots, v_k)$ 即是整个图像的全局特征向量.

7.4 BoV, FV 和 VLAD 的关系

FV 可以看作是 BoV 的推广, VLAD 则是 FV 的简化.

对于同样大小的视字本 (visual vocabulary) 来说, FV 由于包含了对均值和方差的梯度而显著地比 BoV 携带了更多关于图像的信息. 特别是, BoV 特征只是一个 k 维的向量, 而 FV 特征具有 $(2D+1)*k$ 维. 实验表明, 对于同样维度的特征来说, FV 的性能通常不低于甚至有时明显好于 BoV; 并且由于 FV 的 k 值更小, 从而在运算速度上比 BoV 快得多.