2016/11/17, Thirsday, Rainy & Cloudy

bcnnvdm, useVal = false.

```
train: epoch 128:  22/ 24: 375.3 (373.1) Hz objective: 0.493 top1err: 0.012 top5err: 0.001
train: epoch 128:  23/ 24: 375.3 (375.5) Hz objective: 0.493 top1err: 0.012 top5err: 0.001
train: epoch 128:  24/ 24: 376.3 (441.7) Hz objective: 0.493 top1err: 0.012 top5err: 0.001
val: epoch 128:   1/ 23: 532.4 (532.4) Hz objective: 1.285 top1err: 0.199 top5err: 0.031
val: epoch 128:   2/ 23: 534.9 (537.3) Hz objective: 1.338 top1err: 0.225 top5err: 0.051
val: epoch 128:   3/ 23: 530.7 (522.5) Hz objective: 1.372 top1err: 0.228 top5err: 0.056
val: epoch 128:   4/ 23: 528.5 (522.2) Hz objective: 1.362 top1err: 0.229 top5err: 0.060

by Jincheng Su @ Hikvision, mail: jcsu14@fudan.edu.cn
```

bcnnvdvd, useVal = true.

Questions remain:

```
    1. Has the `bcnn` network been correctly constructed?
    2. The training outputs a final model, is it the final results that can reproduce the pap
er's experimental results?
    3. How to fine-tune the hyper-parameters?
    4. How to use the final model?
    5. How to transplant it to Caffe framework?
    6. Installation of `Caffe` ...
    7. ...
```

To answer those questions. It helps to review the `bcnn` paper.

Research: Bilinear CNN Models for Fine-grained Visual Recognition

```
ICCV2015, Tsung-Yu Lin et al. University of Massachusetts, Amherst
```

- **Overview**

  ```
    An architecture consists of two feature extractors whose outputs are multiplied using
   outer product at each location of the image and pooled to obtain an image descriptor i
  n a translationally invariant manner which is particular useful for fine-grained catego
  rization.
    The two feature extractors are based on convolutional neural networks (CNNs). The pre
  trained CNNs are truncated at a convolutional layer including non-linearities. Outputs
  of the two truncated CNNs are multiplied using outer product.
    By pre-training the model can benefit from additional training data when domain speci
  fic data is scarce. By using only the convolutional layers, another advantage is that t
  he resulting CNN can process images of an arbitrary size in a single forward-propagatio
  nstep and produce outputs indexed by the location in the image and the feature model.
    By using networks pretrained from the ImageNet dataset followed by `domain specific f
  ine-tuning`, the model achieve 84.1% accuracy on the `CUB-200-2011` dataset requiring o
  nly category labels at training time, which is state-of-the-art (2015).
  ```

- **Bilinear Model**

  A bilinear model $\mathcal{B}$ for image classification can be expressed by a quadruple:

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

  Where

  - f_A, f_B are *feature extractor functions*: $f : \mathcal{L} \times \mathcal{I} \rightarrow \mathcal{R}^{c \times D}$, where
    - $\mathcal{I}$: be an input image,
    - $\mathcal{L}$: be a location,
    - $\mathcal{R}$: be an feature of size $c \times D$,
  - $\mathcal{P}$ is a *pooling function*,
  - $\mathcal{C}$ is a *classification function*.

  The feature outputs are combined at each location using the matrix outer product, i.e., the *bilinear feature* combination of f_A and f_B at location l is given by:

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

  Where both $f_A(l, \mathcal{I})$ and $f_B(l, \mathcal{I})$ must have the same feature dimension c to be compatible.

  Followed by the outer product is the pooling function $\mathcal{P}$ that simply sum all the bilinear features:

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

  If $f_A(l, \mathcal{I})$ is with size $C \times M$, $f_B(l, \mathcal{I})$ with $C \times N$, the $\phi(\mathcal{I})$ would have size $M \times N$.

  Then $\phi(\mathcal{I})$ is reshape to a vector with size $MN \times 1$:

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

  That is all what the `vl_nnbilinearpool or vl_nnbilinearclpool` function in `bcnn` about.

- **Bilinear CNN model**

  1. Use two CNNs pretrained on the *ImageNet* dataset truncated at a convolutional layer including non-linearities as feature extractors f_A and f_B.
     `imagenet-vgg-m.mat` truncate at layer 14 ($conv_5 + relu$) referred as `M-Net` and `imagenet-vgg-verydeep-16.mat` truncated at layer 30 ($conv_{5\_4} + relu$) referred as `D-Net` are used in the paper.
  2. Apply bilinear model presented above to get $x = \phi(\mathcal{I})$ vector.
  3. Passed through a signed square-root normalization followed by a l_2 normalization:

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

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

  4. Use logistic regression or linear SVM as *classification function* $\mathcal{C}$. A linear SVM is used in this paper.

- **Implementation details**

  1. `y = vl_nnbilinearclpool(x1, x2, varargin)`
     Input: two different features x_1, x_2 with size $[h_1, w_1, ch_1]$ and $[h_2, w_2, ch_2]$ for height, width and number of channels and $\nabla_Y Z$ (`dzdy`) which is the gradient of const function Z w.r.t y.
     Output: A bilinear vector y with size $[ch1 * ch2, 1]$.

     - **Feed forward.**

       - Resize x_1, x_2 to the same size, i.e., downsampling one of them to ensure $h_1 * w_1 == h_2 * w_2$:

```
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])
```

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

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

- 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. `y = vl_nnsqrt(x, param, varargin)`

- **Feed forward** (element-wise operation):

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

  Note that y is with the same size as x.

- **Back propagation** (element-wise operation):

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

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

where $\frac{dy}{dx} = (\frac{dy_1}{dx_1}, \ldots, \frac{dy_n}{dx_n})^T$.

3. `y = vl_nnl2normalization(x, param, varargin)`

   - **Feed forward**:

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

     Note that y is with the same size as x.

   - **Back propagation**:

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

     Which gives the vectorize form:

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

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

     > **To prevent large values, it is recommended to preprocess ||x|| by adding a threshold:**

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

- **Training the B-CNN model.**

  1. Train a B-CNN model (without classification function $\mathcal{C}$?):

     > **truncated_CNNs -> vl_nnbilinearpool -> sqrt_normalization -> l2_normalization -> conv_classifier -> softmaxloss**

     Three B_CNN fine-trained models are provided in the paper: `B-CNN[M, M]` with two `M-Net` , `B-CNN[D, D]` with two `D-Net` , and `B-CNN[M, D]` .

  2. After fine-training a `B-CNN` model, a linear SVM is trained on the `bcnn` features extracted by the `B-CNN` model.

  3. `run_experiments.m` extracts `B-CNN` features and trains a SVM classifier on fine-grained categories.

     3.1. Symmetric B-CNN: extracts the self outer-product of features at *'layera'*.

     ```
     bcnn.opts = {..
        'type', 'bcnn', ...
        'modela', PRETRAINMODEL, ...
        'layera', 14,...
        'modelb', [], ...
        'layerb', [],...
     } ;
     ```

     3.2. Cross layer B-CNN: extracts the outer-product between features at *'layera'* and *'layerb'* using the

same CNN.

```
bcnn.opts = {..
   'type', 'bcnn', ...
   'modela', PRETRAINMODEL, ...
   'layera', 14,...
   'modelb', [], ...
   'layerb', 12,...
} ;
```

3.3. Asymmetric B-CNN: extracts the outer-product between features from CNN *'modela'* at *'layera'* and CNN *'modelb'* at *'layerb'*.

```
bcnn.opts = {..
   'type', 'bcnn', ...
   'modela', PRETRAINMODEL_A, ...
   'layera', 30,...
   'modelb', PRETRAINMODEL_B, ...
   'layerb', 14,...
} ;
```

3.4. Fine-tuned B-CNN: If you fine-tune a B-CNN network (see next section), you can evaluate the model using:

```
bcnn.opts = {..
   'type', 'bcnn', ...
   'modela', FINE-TUNED_MODEL, ...
   'layera', [],...
   'modelb', [], ...
   'layerb', [],...
} ;
```

4. Fine-tuning B-CNN models
   See `run_experiments_bcnn_train.m` for fine-tuning a B-CNN model. Note that this code caches all the intermediate results during fine-tuning which takes about 200GB disk space.

   Here are the steps to fine-tuning a B-CNN [M,M] model on the CUB dataset:

   1. Download `CUB-200-2011` dataset (see link above)
   2. Edit `opts.cubDir=CUBROOT` in `model_setup.m` , CUBROOT is the location of CUB dataset.
   3. Download `imagenet-vgg-m` model (see link above)
   4. Set the path of the model in `run_experiments_bcnn_train.m` . For example, set `PRETRAINMODEL='data/model/imagenet-vgg-m.mat'` , to use the Oxford's VGG-M model trained on ImageNet LSVRC 2012 dataset. You also have to set the `bcnnmm.opts` to:

      ```
      bcnnmm.opts = {..
         'type', 'bcnn', ...
         'modela', PRETRAINMODEL, ...
         'layera', 14,...
         'modelb', PRETRAINMODEL, ...
         'layerb', 14,...
         'shareWeight', true,...
      } ;
      ```

      The option `shareWeight=true` implies that the blinear model uses the same CNN to extract both features resulting in a symmetric model. For assymetric models set `shareWeight=false` . Note that this roughly doubles the GPU memory requirement. The `cnn_train()` provided from

MatConvNet requires the setup of validation set. You need to prepare a validation set for the datasets without pre-defined validation set.

5. Once the fine-tuning is complete, you can train a linear SVM on the extracted features to evaluate the model. See `run_experiments.m` for training/testing using SVMs. You can simply set the MODELPATH to the location of the fine-tuned model by setting `MODELPATH='data/ft-models/bcnn-cub-mm.mat'` and the `bcnnmm.opts` to:

```
bcnnmm.opts = {..
   'type', 'bcnn', ...
   'modela', MODELPATH, ...
   'layera', [],...
   'modelb', [], ...
   'layerb', [],...
} ;
```

6. And type `>> run_experiments()` on the MATLAB command line. The results with be saved in the `opts.resultPath .`

- **Experimental methodology**
- **Introduction & Related work**