Soft Computing project report 4 **Akilesh Badrinaaraayanan**
CS 5190 Roll no: CS13B1042
Prof. C. Krishna Mohan
TA: Debaditya Roy Due Date: 27/03/16

Problem Statement

Implementation of Bilinear CNN Models for fine-grained visual recognition [1]

Dataset details

- **Birds**: CUB-200-2011 dataset. Birds+box uses bounding boxes at training and test time. It has 200 categories of birds, 11788 images. It contains 15 part locations, 312 binary attributes and 1 bounding box.

Installation

The implementation is MATLAB based and depends on VLFEAT and MatConvNet. I installed matconvnet-1.0-beta8 and vlfeat-0.9.20. After installing, I have to configure MatConvNet. For this purpose, I made setup.m as shown below.

```
run ../vlfeat-0.9.20/toolbox/vl_setup
run ../matconvnet-1.0-beta8/matlab/vl_setupnn
addpath ../matconvnet-1.0-beta8/examples
clear mex ;
```

Fine-grained datasets

To run experiments download the datasets from various places and edit the *model_setup.m* file to point it to the location of each dataset. For instance, you can point to the birds dataset directory by setting opts.cubDir = 'data/cub'.

Implementation Details

The asymmetric B-CNN model can be implemented using two networks whose feature outputs are bilinearly combined followed by a shallow network for normalization and computing softmax loss. This implementation runs forward and backward passes through two networks separately.

When the same network is used to extract both features, the symmetric B-CNN model can be implemented as a single network architecture consisting of bilinearpool, sqrt, and l2 norm layers on the top of convolutional layers. This implementation is expected to be twice as fast and memory efficient than asymmetric implementation.

The implementation for B-CNN can be done by using the following MATLAB functions:

1. vl_bilinearnn(): This extends vl_simplenn() of the MatConvNet library to include the bilinear layers.
2. vl_nnbilinearpool(): Bilinear feature pooling of outer product with itself.

3. vl_nnbilinearclpool(): Bilinear feature pooling with outer product of two different features (same resolution of two feature outputs).
4. vl_nnsqrt(): Signed square-root normalization
5. vl_nnl2norm(): L2 normalization

BCNN Model

Here, f_a and f_b are decoupled by using separate feature functions.

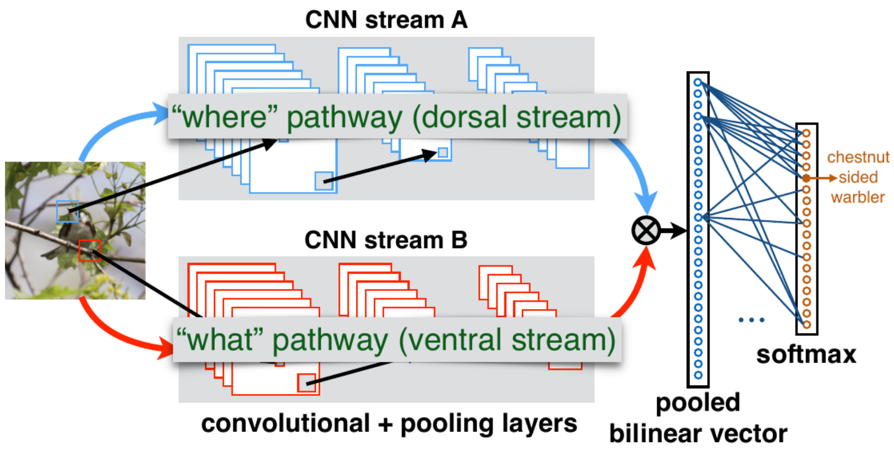


Figure 1: Bilinear CNN model

BCNN Model training

Back propagation through the bilinear layer is easy.

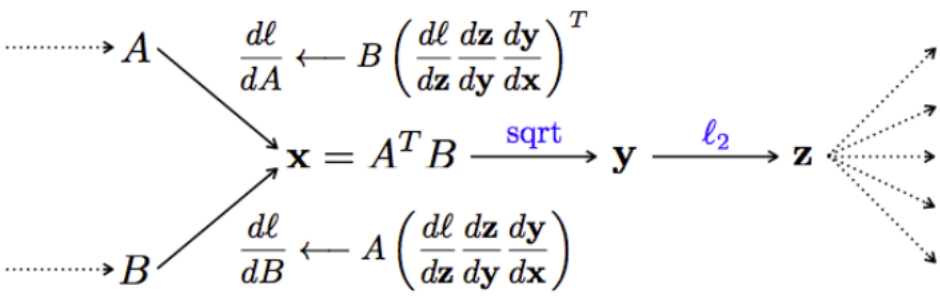


Figure 2: Back propagation

It allows end-to-end training.

There are two normalization layers in the architecture

- Square-root normalization

$$y = sign(x)\sqrt{x} \tag{1}$$

- l2 normalization

$$z = \frac{y}{||y||} \tag{2}$$

Classification Demo

I wrote a MATLAB script (*bird_test_aki.m*) which loads the CUB-200-2011 dataset, takes a test image from this dataset, load the bilinear models provided by the authors and compute the B-CNN features (the authors have provided *get_bcnn_features* function which combines the feature extracted by the two CNNs and does sum-pooling to aggregate the bilinear features across the image) as described in the paper. Based on the features, I calculate scores to predict the top 5 closest class labels. If a GPU is installed on the machine, give *opts.useGpu = true* to speed up the computation.

```
function bird_demo(varargin)
setup;
% Default options
opts.modela = '../data/models/bcnn-cub-dm/bcnn-cub-dm-neta.mat';
opts.layera = 30;
opts.modelb = '../data/models/bcnn-cub-dm/bcnn-cub-dm-netb.mat';
opts.layerb = 14;
opts.cubDir = '../data/CUB_200_2011';
opts.useGpu = false;
opts.svmPath = fullfile('../data', 'models','svm_cub_vdm.mat');
opts.imgPath = 'test_image.jpg';
opts.regionBorder = 0.05 ;
opts.normalization = 'sqrt_L2';
opts.topK = 5; % Number of labels to display

% Parse user supplied options
opts = vl_argparse(opts, varargin);

% Load CUB database
tic;
imdb = cub_get_database(opts.cubDir, false, false);
fprintf('%.2fs to load imdb.\n', toc);

% Read image
origIm = imread(opts.imgPath);
im = single(origIm);

% Load classifier
tic;
classifier = load(opts.svmPath);

% Load the bilinear models and move to GPU if necessary
neta = load(opts.modela);
neta.layers = neta.layers(1:opts.layera);
netb = load(opts.modelb);
netb.layers = netb.layers(1:opts.layerb);
if opts.useGpu,
    neta = vl_simplenn_move(neta, 'gpu');
    netb = vl_simplenn_move(netb, 'gpu');
    neta.useGpu = true;
```

```
    netb.useGpu = true;
else
    neta = vl_simplenn_move(neta, 'cpu');
    netb = vl_simplenn_move(netb, 'cpu');
    neta.useGpu = false;
    netb.useGpu = false;
end
fprintf('%.2fs to load models into memory.\n', toc);

tic;
% Compute B-CNN feature for this image
code = get_bcnn_features(neta, netb, im, ...
        'regionBorder', opts.regionBorder, ...
        'normalization', opts.normalization);

% Make predictions
scores = classifier.w'*code{1} + classifier.b';
[~, pred] = sort(scores, 'descend');

% Predict class labels
pred_class = classifier.classes(pred(1:opts.topK));
fprintf('Top %d prediction for %s:\n', opts.topK, opts.imgPath);
fprintf('%s\n', pred_class{:});
fprintf('%.2fs to make predictions [GPU=%d]\n', toc, opts.useGpu);

% Display 4 other images from the training set from the top class
N = 4; w = 224; h = 224;
classId = pred(1);
imageInd = find(imdb.images.label == classId & imdb.images.set == 1);
imageInd = imageInd(randperm(length(imageInd)));
classImage = cell(4,1);
for j=1:N
    classImage{j} = imresize(imread(fullfile(imdb.imageDir, imdb.images.name{ima
    if(size(classImage{j}, 3) == 1) % Make color
        classImage{j} = repmat(classImage{j}, 1, 1, 3);
    end
end
montageImage = cat(1, classImage{:});
figure(1); clf;
subplot(4, 5, [1:4, 6:9, 11:14, 16:19]); image(origIm);
subplot(4, 5, 5*(1:4)); image(mat2gray(montageImage));
set(gcf,'NextPlot','add');axes;
h = title(pred_class{1}, 'interpret' ,'none');
set(gca,'Visible','off');set(h,'Visible','on');
```

Figure 3: Test image

```
>> bird_test_aki
0.11s to load imdb.
2.73s to load models into memory.
Top 5 prediction for test_image.jpg:
064.Ring_billed_Gull
059.California_Gull
147.Least_Tern
062.Herring_Gull
060.Glaucous_winged_Gull
2.21s to make predictions [GPU=0]
>>
```

Figure 4: Classification demo for test image

Fine tuning B-CNN models

The script *run_experiments_bcnn_train.m* is 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.

2. Edit opts.cubDir=CUBROOT in *model_setup.m*, CUBROOT is the location of CUB dataset.

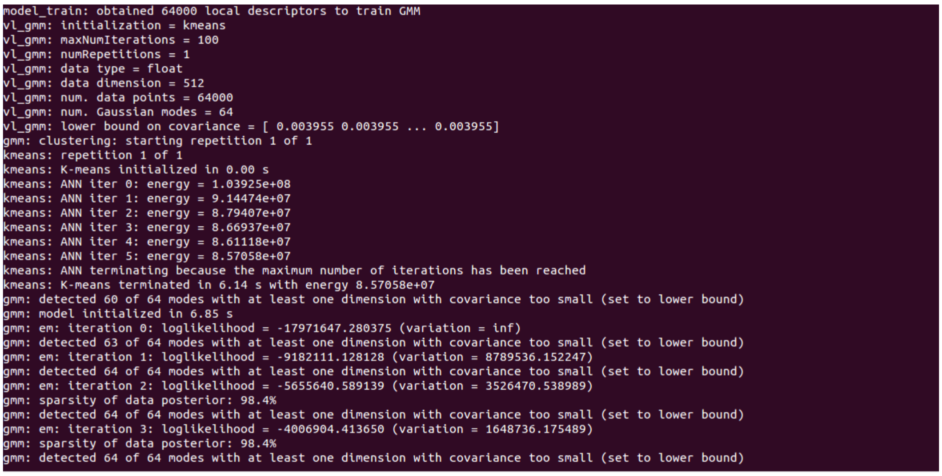
3. Download imagenet-vgg-m model.

4. Set the path of the model in *run_experiments_bcnn_train.m*. For example, set PRETRAINMODEL='dat vgg-m.mat', to use the Oxford's VGG-M model trained on ImageNet LSVRC 2012 dataset.

5. The option shareWeight=true in bcnnmm.opts 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.

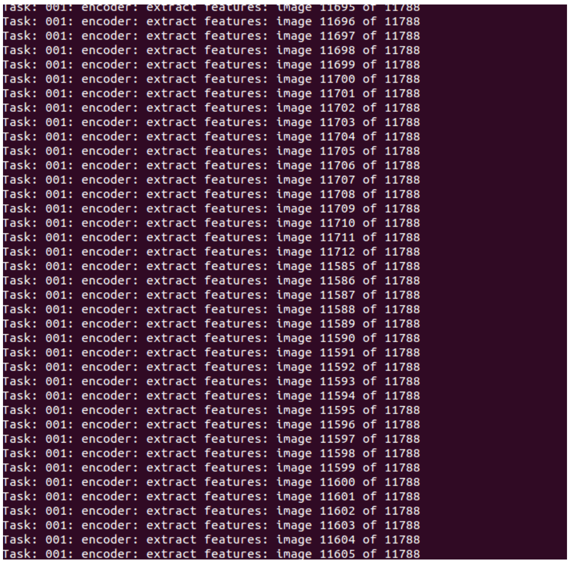
6. Once the fine-tuning is complete, you can train a linear SVM on the extracted features to evaluate the model. The script *run_experiments.m* for training/testing using SVMs. You can simply set the MODELPATH to the location of the fine-tuned model by setting MODELPATH='data/ft-models/bcnn-cub-mm.mat'

7. Finally run the script *run_experiments.m* from the MATLAB command line. The results with be saved in the opts.resultPath.



```
model_train: obtained 64000 local descriptors to train GMM
vl_gmm: initialization = kmeans
vl_gmm: maxNumIterations = 100
vl_gmm: numRepetitions = 1
vl_gmm: data type = float
vl_gmm: data dimension = 512
vl_gmm: num. data points = 64000
vl_gmm: num. Gaussian modes = 64
vl_gmm: lower bound on covariance = [ 0.003955 0.003955 ... 0.003955]
gmm: clustering: starting repetition 1 of 1
kmeans: repetition 1 of 1
kmeans: K-means initialized in 0.00 s
kmeans: ANN iter 0: energy = 1.03925e+08
kmeans: ANN iter 1: energy = 9.14474e+07
kmeans: ANN iter 2: energy = 8.79407e+07
kmeans: ANN iter 3: energy = 8.66937e+07
kmeans: ANN iter 4: energy = 8.61118e+07
kmeans: ANN iter 5: energy = 8.57058e+07
kmeans: ANN terminating because the maximum number of iterations has been reached
kmeans: K-means terminated in 6.14 s with energy 8.57058e+07
gmm: detected 60 of 64 modes with at least one dimension with covariance too small (set to lower bound)
gmm: model initialized in 6.85 s
gmm: em: iteration 0: loglikelihood = -17971647.280375 (variation = inf)
gmm: detected 63 of 64 modes with at least one dimension with covariance too small (set to lower bound)
gmm: em: iteration 1: loglikelihood = -9182111.128128 (variation = 8789536.152247)
gmm: detected 64 of 64 modes with at least one dimension with covariance too small (set to lower bound)
gmm: em: iteration 2: loglikelihood = -5655640.589139 (variation = 3526470.538989)
gmm: sparsity of data posterior: 98.4%
gmm: detected 64 of 64 modes with at least one dimension with covariance too small (set to lower bound)
gmm: em: iteration 3: loglikelihood = -4006904.413650 (variation = 1648736.175489)
gmm: sparsity of data posterior: 98.4%
gmm: detected 64 of 64 modes with at least one dimension with covariance too small (set to lower bound)
```

Figure 5: Classification demo for test image



```
Task: 001: encoder: extract features: image 11695 of 11788
Task: 001: encoder: extract features: image 11696 of 11788
Task: 001: encoder: extract features: image 11697 of 11788
Task: 001: encoder: extract features: image 11698 of 11788
Task: 001: encoder: extract features: image 11699 of 11788
Task: 001: encoder: extract features: image 11700 of 11788
Task: 001: encoder: extract features: image 11701 of 11788
Task: 001: encoder: extract features: image 11702 of 11788
Task: 001: encoder: extract features: image 11703 of 11788
Task: 001: encoder: extract features: image 11704 of 11788
Task: 001: encoder: extract features: image 11705 of 11788
Task: 001: encoder: extract features: image 11706 of 11788
Task: 001: encoder: extract features: image 11707 of 11788
Task: 001: encoder: extract features: image 11708 of 11788
Task: 001: encoder: extract features: image 11709 of 11788
Task: 001: encoder: extract features: image 11710 of 11788
Task: 001: encoder: extract features: image 11711 of 11788
Task: 001: encoder: extract features: image 11712 of 11788
Task: 001: encoder: extract features: image 11585 of 11788
Task: 001: encoder: extract features: image 11586 of 11788
Task: 001: encoder: extract features: image 11587 of 11788
Task: 001: encoder: extract features: image 11588 of 11788
Task: 001: encoder: extract features: image 11589 of 11788
Task: 001: encoder: extract features: image 11590 of 11788
Task: 001: encoder: extract features: image 11591 of 11788
Task: 001: encoder: extract features: image 11592 of 11788
Task: 001: encoder: extract features: image 11593 of 11788
Task: 001: encoder: extract features: image 11594 of 11788
Task: 001: encoder: extract features: image 11595 of 11788
Task: 001: encoder: extract features: image 11596 of 11788
Task: 001: encoder: extract features: image 11597 of 11788
Task: 001: encoder: extract features: image 11598 of 11788
Task: 001: encoder: extract features: image 11599 of 11788
Task: 001: encoder: extract features: image 11600 of 11788
Task: 001: encoder: extract features: image 11601 of 11788
Task: 001: encoder: extract features: image 11602 of 11788
Task: 001: encoder: extract features: image 11603 of 11788
Task: 001: encoder: extract features: image 11604 of 11788
Task: 001: encoder: extract features: image 11605 of 11788
```

Figure 6: Classification demo for test image

Method	Birds	Birds with bounding box
B-CNN [M,M]	79.9	80.2
B-CNN [D,M]	83.3	84.8
B-CNN [D,D]	83.2	83.9

Table 1: Fine-grained classification results

Extensions to the current work

In the current work, the authors propose bilinear CNN models for fine-grained visual recognition and apply it to birds, aircrafts and cars dataset. In their extension work, they propose One-to-many face recognition using Bilinear CNNs. They perform the experiments on Face-Scrub and IJB-A Train dataset. They use a linear SVM classifier learned for each person in gallery and max-pooling features or classifier scores to aggregate multiple media. Finally, they demonstrate how a standard CNN pre-trained on a large face database (say VGG-Face model) can be converted into a B-CNN without any additional feature training.

Conclusion and difficulties faced

The bilinear CNN models are quite effective on various fine-grained recognition datasets. The proposed models when fine-tuned using image labels result in significant improvements over orderless texture descriptors. I was not able to use the GPU for training due to compatibility issues. The CUDA version installed in our GPU is 6.5 and corresponding compatible MATLAB version should be R2015a but the one installed in our GPU is R2013a.

Current work

I'm currently working on Action recognition and Temporal Action Detection as suggested by my TA (Mr. Debaditya Roy). I'm coordinating with him, finishing the tasks assigned by him, from time to time. The goal is to recognize and localize a large number of human action classes from open source videos in a realistic setting. Our system shall output a real-valed score indicating the confidence of the predicted presence. The untrimmed nature of the videos is what makes the action recognition in such videos challenging. In other words, a significant part of a test video may not include any particular action, and multiple instances may occur at different timestamps within the video.

Acknowledgements

My sincere thanks and gratitude to Mr. Debaditya Roy for his advice on the implementation and for his valuable time spent, clearing my doubts (even if they were silly).

References

[1] Tsung-Yu Lin, Aruni RoyChowdhury, and Subhransu Maji: Bilinear CNNs for Fine-grained Visual Recognition, ICCV 2015