16720J: Homework 2 - Bag-of-Words for Scene Classification

Luting Wang

October 8, 2015

3 Warming up with some theory (10pts)

Question 3.1 (3pts, 2-3 lines)

Given an $N \times N$ image, and an $h \times h$ Gaussian filter, we can convolve the image and the filter in $O(h)$ time by implementing two convolutions of size h. That's to say, for each row and column, we do convolution separately.

Question 3.2 (2pts, 2-3 lines)

No. Because in bag-of-words model, an image is represented as an orderless collection of local features. So it doesn't respect spatial information.

Question 3.3 (5 pts, 2-3 lines)

Because the use of filter responses rather than raw pixel values provides robustness against small shifts and changes in lighting. That is why we use the filter response rather than raw pixel data to extract features.

4 Representing the World with Visual Words (40pts)

Question 4.1 (5 pts, 3-4 lines) Theory:

There are 33 filters banks totally, which can be grouped into three categories. The first categories is Gaussian Filter, which includes No.1,No.2,No.3,No.12,No.13,No.14,No.23,No.24, No.25 filter banks. It is a low pass filter, which smooths the image by the same amount in all directions. The next categories is Laplacian Filter, which includes No.4,No.5,No.6,No.7,No.15, No.16,No.17,No.18,No.26,No.27,No.28,No.29 filter banks. It highlights regions of rapid intensity change and is therefore used for edge detection. The last categories is Oriented Gaussian Filters, which includes No.8,No.9,No.10,No.11,No.19,No.20,No.21,No.22,No.30,No.31,No.32, No.33 filter banks. They smooth the image in X or Y direction.

Question 4.2 (15 pts)

```
function [filterBank,dictionary] = getFilterBankAndDictionary(trainFiles)

alpha = 50;
K = 200; %the size of dictionary
filterBank = createFilterBank();
l = length(trainFiles);
total_response = zeros(alpha*l, length(filterBank)*3);
for i = 1:l
    im = imread(trainFiles{i});
    response = extractFilterResponses(im, filterBank);
    n = size(im, 1)*size(im,2);
    pixels = randperm(n);
    for j = 1:alpha
        total_response((i-1)*alpha+j,:) = response(pixels(j),:);
    end
end

[unused, dictionary] = kmeans(total_response, K, 'EmptyAction', 'drop');


end
```

Question 4.3 (5 pts, 3-4 lines) Theory

Dictionary size controls the trade-off between being distinctive and robust, and affects the efficiency of the algorithm which use bag of features. Basically, large dictionary size would be more distinctive, but less robust and efficient, while small dictionary size would be more robust and efficient, but less distinctive. Choosing a suitable size is a empirically process and it is problem specific. Both being too large or small won't be a good choice.

Question 4.4 (15 pts)

```
function [wordMap]=getVisualWords(I,filterBank,dictionary)


   wordMap = zeros(size(I, 1)* size(I, 2),1);
   filterResponses = extractFilterResponses(I, filterBank);

   for i = 1:size(filterResponses, 1);
       l = pdist2(filterResponses(i,:), dictionary);
       [unused, index] = min(l);
        wordMap(i, 1) = index;
   end
```

```
    wordMap = reshape(wordMap, size(I, 1), size(I, 2));

end
```

5 Building a Recognition System (95pts)

Question 5.1 (10 pts)

```
function [h] = getImageFeatures(wordMap,dictionarySize)

wordMap_1 = wordMap(:);
h = hist(wordMap_1, dictionarySize);
h = h./size(wordMap_1, 1);
h = h';

end
```

Question 5.2 Extra credit (5 pts 2-3 lines) Theory:

Because for different images, the total number of its features are different. So if we don't normalize its feature histogram, we can't compare one image histogram to another. Normalizing helps us to force the total number of features in all the images to be the same.

Question 5.3 Extra credit (5 pts 2-3 lines) Theory

For each image , it has multi-resolution(0, 1, .. l .. L-1), and for each resolution, the image is be cut into $2^l * 2^l$ cells. Two points matching means they fall into the same cell. From histogram intersection function, we know that the number of matches found in layer l includes the number of matches found in layer $l+1$. That's to say, matched found at higher layer should be weighted more. And we can also figure out that the weight in layer l is inversely proportional to its cell width.

Question 5.4 (20 pts)

```
function [h] = getImageFeaturesSPM(layerNum, wordMap, dictionarySize)

    x = floor(size(wordMap, 1)/4);
    y = floor(size(wordMap, 2)/4);
    x = 4*x;
    y = 4*y;
    wordMap=wordMap(1:x, 1:y);

    k=dictionarySize;
    L=layerNum-1;
    h = zeros(k*(power(4, L+1)-1)/3, 1);
```

```
    l = L;
    numofcellperline = power(2,l);
    numofcells = numofcellperline * numofcellperline;
    sizeofcellsx = size(wordMap, 1)/numofcellperline;
    sizeofcellsy = size(wordMap, 2)/numofcellperline;
    mx = sizeofcellsx*ones(1, numofcellperline);
    my = sizeofcellsy*ones(1, numofcellperline);
    n = mat2cell(wordMap, mx, my);
    c = cell(numofcellperline, numofcellperline);
    d = cell(2, 2);

    for i = 1:numofcellperline
        for j = 1:numofcellperline
           c{i,j} = getImageFeatures(n{i,j},dictionarySize);
           h((((i-1)*4+j-1)*k + 1):(((i-1)*4+j)*k),1) \\
           = c{i,j}*power(2, l-L-1)/numofcells;
        end
    end

    d{1, 1} = (c{1,1} + c{1,2} +c{2, 1} +c{2,2})/4;
    d{1, 2} = (c{1,3} + c{1,4} +c{2, 3} +c{2,4})/4;
    d{2, 1} = (c{3,1} + c{3,2} +c{4, 1} +c{4,2})/4;
    d{2, 2} = (c{3,3} + c{3,4} +c{4, 3} +c{4,3})/4;

    h((16*k+1:17*k), 1) = d{1, 1}*power(2, -L)/4;
    h((17*k+1:18*k), 1) = d{1, 2}*power(2, -L)/4;
    h((18*k+1:19*k), 1) = d{2, 1}*power(2, -L)/4;
    h((19*k+1:20*k), 1) = d{2, 2}*power(2, -L)/4;

    h((20*k+1:21*k), 1) = ((d{1, 1} +  d{1, 2} +  d{2, 1} +  d{2, 2})/4)*power(2, -L);

end
```

Question 5.5 (10 pts)

```
function [histInter] = distanceToSet(wordHist, histograms)

    histInter = sum(bsxfun(@min, wordHist, histograms));

end
```

Question 5.6 (5 pts)

```
function outputHistograms = createHistograms(dictionarySize,imagePaths,wordMapDir)
```

```
    l = length(imagePaths);
    outputHistograms = zeros(dictionarySize*(4^3-1)/3, l);
    layerNum = 3;
    for i = 1 : l
    wordMapFileName = fullfile(wordMapDir,strrep(imagePaths{i},'.jpg','.mat'));
        wordMap = load(wordMapFileName);
        wordMap = wordMap.wordMap;
        outputHistograms(:, i) \\
        = getImageFeaturesSPM(layerNum, wordMap, dictionarySize);
    end

end
```

Question 5.7 (5 pts)

```
    trainSystem.m:

    % load('traintest.mat');
    imageDir = '../images'; %where all images are located
    targetDir = '../wordmap';%where we will store visual word outputs

    %compute filter responses and make dictionary
    fprintf('Computing dictionary ... ');
    computeDictionary(trainImagePaths,imageDir);
    load('dictionary.mat','dictionary','filterBank');
    fprintf('done\n');

    %now compute visual words for each image
    numCores = 12;
    fprintf('Computing visual words ... ');
    batchToVisualWords(trainImagePaths,classnames,filterBank,\\
        dictionary,imageDir,targetDir,numCores);
    fprintf('done\n');

    %now compute histograms for all training images using visual word files
    trainingHistogramsFile = fullfile(targetDir,'trainingHistograms.mat');
    dictionarySize = size(dictionary,1);
    fprintf('Computing histograms ... ');
    trainHistograms = createHistograms(dictionarySize,trainImagePaths,targetDir);
    fprintf('done\n');
    save(trainingHistogramsFile,'trainHistograms');
    load(trainingHistogramsFile,'trainHistograms');
```

```
    %the test code just needs to load trainOutput.mat, so lets store it
    save('trainOutput.mat','dictionary','filterBank',\\
        'trainHistograms','trainImageLabels');
```

Question 5.8 (10 pts)

```
function [predictedLabel] = knnClassify(wordHist,trainHistograms,trainImageLabels,k)

    distances = distanceToSet(wordHist, trainHistograms);
    [~, pos] = sort(distances, 'descend');
    index = trainImageLabels(pos(1:k));
    predictedLabel = mode(index);

end
```

Question 5.9 (10 pts)

evaluateRecognitionSystem.m:

```
%Loading the dictionary, filters and training data
numCores=12;
imageDir = '../images'; %where all images are located
targetDir = '../wordmap';%where we will store visual word outputs
layerNum = 3;
KNN = 20;
dictionarySize = size(dictionary, 1);
load('traintest.mat');
load('trainOutput.mat');

try
    fprintf('Closing any pools...\n');
    matlabpool close;
catch ME
    disp(ME.message);
end

fprintf('Will process %d files in parallel to test the systems ...\n',\\
    length(testImagePaths));
fprintf('Starting a pool of workers with %d cores\n', numCores);
matlabpool('local',numCores);

testDir = '../testimage_wordmap';
for i = 1:length(classnames)
    if ~exist(fullfile(testDir,classnames{i}),'dir')
        mkdir(fullfile(testDir,classnames{i}));
```

```
    end
end

l = length(testImagePaths);
testM = zeros(l, 1);
predictM = zeros(l, 1);
wordRepresentation = cell(l,1);
accuracy = zeros(1, KNN);
C = cell(1, KNN);

parfor i=1:l
fprintf('Converting the test images to visual words %s\n', testImagePaths{i});
image = imread(fullfile(imageDir, testImagePaths{i}));
wordMap = getVisualWords(image, filterBank, dictionary);
wordRepresentation{i} = getImageFeaturesSPM(layerNum, wordMap, dictionarySize);
end

fprintf('Dumping the files\n');
for i=1:l
wordMap = wordRepresentation{i};
save(fullfile(testDir, [strrep(testImagePaths{i},'.jpg','.mat')]),'wordMap');
end

for knn = 1:KNN
    parfor i=1:l
    predictedLabel = knnClassify(wordRepresentation{i},\\
        trainHistograms,trainImageLabels,knn)
    testM(i) = testImageLabels(i);
    predictM(i) = predictedLabel;
    end
    [c,~]=confusionmat(testM,predictM);
    accuracy(knn) = trace(c)/sum(c(:));
    C{knn} =  c;
    disp(c);
    fprintf('when k is%d, the accuracy is %d\n', knn, accuracy(knn));
end
save('evaluateRecognition.mat','C','accuracy','wordRepresentation');

fprintf('Closing the pool\n');
matlabpool close

confusionMatrix =
    25     0     0     1     1     0     1     0     1
     0    24     7     0     0     2     0     1     1
     0     7    13     0     0     4     0     7     0
```

```
    1     3     1    25     4     1     4     0     1
    1     0     1     1    40     0     1     0     0
    0     8     7     1     0     4     0     6     0
    0     1     0     3     8     1    20     0     1
    0     0     3     0     0     4     0    17     7
    0     2     9     0     0     2     0     7    31

when k is 1, the accuracy is 6.199377e-01
```

The console is shown below(confusionMatrix and accuracy when k is 8)

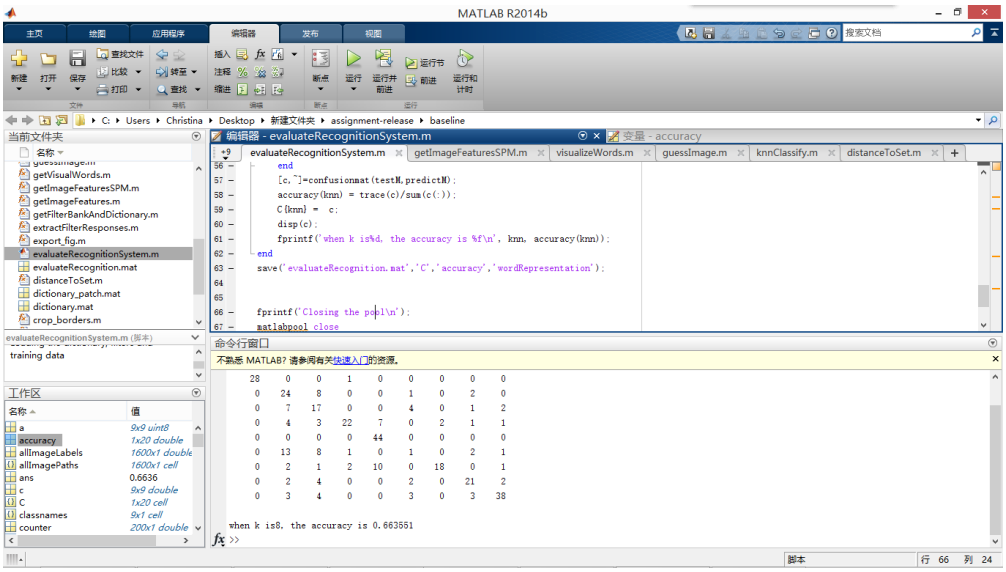


Figure 1: **ConfusionMatrix and Accuracy when k is 8**

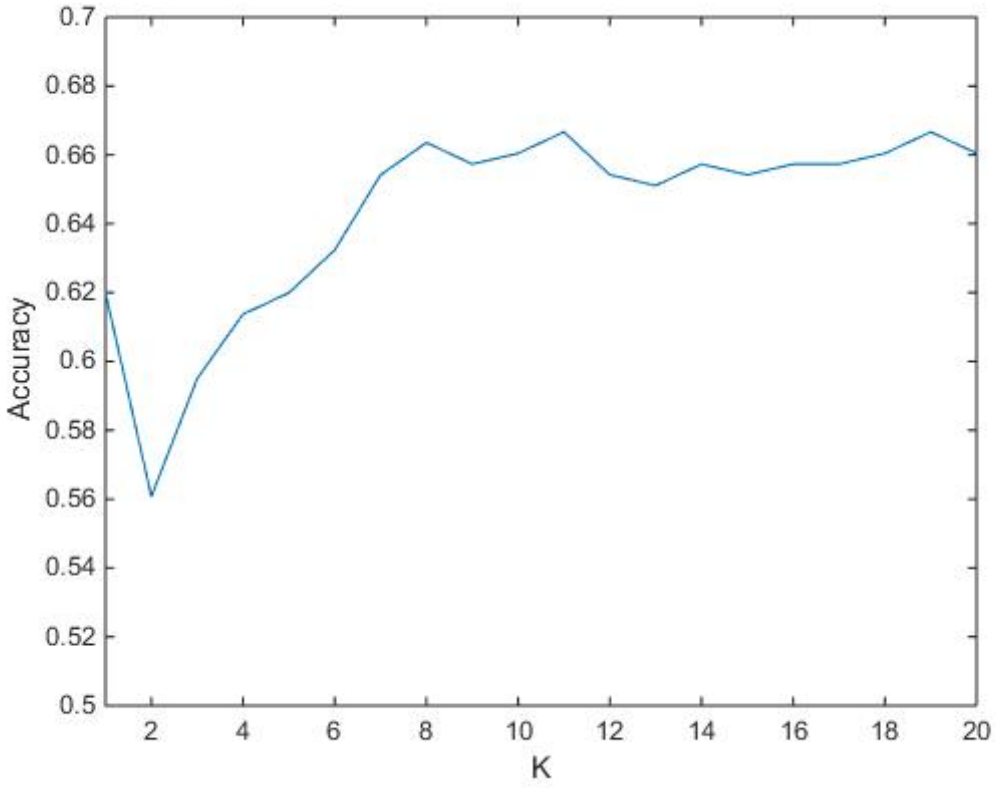
Question 5.10 (10 pts)

K	1	2	3	4	5	6	7	8	9	10
Accuracy	0.619	0.560	0.595	0.613	0.619	0.632	0.654	0.663	0.657	0.660

Table 1: **Accuracy versus k values.**

The accuracy is between 0.560 to 0.663 when k is from 1 to 10. And it tends to be constant as we continuing to increase k . When k is small, noise would have a higher influence on the result, so the accuracy is lower. And as k becomes larger, the result would become stable, but it is computationally expensive. And when k is larger, points that are near might have similar densities or classes, which would defect the basic philosophy behind KNN. A simple approach to select k is to set k = $\sqrt{n}$, n is the dictionary size. The graph of k vs the classification accuracy is shown in Figure2.

Figure 2: Accuracy versus k values.



Question 5.11 (5 pts, 2-3 lines): Theory

As discussed above, k is often set to be $\sqrt{n}$. So when we have a large database, finding the top k nearest neighbors of the query can be time-consuming. If the distance measure we use is computationally expensive(histogram intersection similarity), the problem would be even worse.

6 Final thoughts on visual words (40 pts)

Question 6.1 (20 pts)

visualizeWords.m:

```
load('traintest.mat');
imageDir = '../images'; %where all images are located
targetDir = '../wordmap';%where we will store visual word outputs
l = length(trainImagePaths);
dictionarySize = size(dictionary, 1);
total_patch = cell(dictionarySize, 1);
for i = 1:dictionarySize
```

```
    total_patch{i} = zeros(9,9,3);
end
counter = zeros(dictionarySize, 1);

for i = 1 : l
    wordMapFileName = fullfile(targetDir,strrep(trainImagePaths{i},'.jpg','.mat'));
    imageFileName = fullfile(imageDir,trainImagePaths{i});
    image = imread(imageFileName);
    wordMap = load(wordMapFileName);
    wordMap = wordMap.wordMap;
    numx = floor(size(wordMap, 1)/9);
    x = numx * 9;
    numy = floor(size(wordMap, 2)/9);
    y = numy * 9;
    wordMap = wordMap(1:x, 1:y);
    image = image(1:x, 1:y, :);
    mx = 9*ones(1, numx);
    my = 9*ones(1, numy);
    wordMap_patch = mat2cell(wordMap,mx, my);
    image_patch = mat2cell(image, mx, my,[1 1 1]);

    for j = 1:numx
        for k = 1:numy
        patch = wordMap_patch{j, k};
        index = patch(41);
        I = im2double(cell2mat(image_patch(j, k, :)));
        total_patch{index, 1} = total_patch{index, 1}+ I;
        counter(index, 1) = counter(index, 1)+1;
        end
    end
end

for i = 1:dictionarySize
    total_patch(i, 1) = mat2cell((cell2mat(total_patch(i, 1)) \\
         / counter(i, 1)), 9,9,3);
end
```

These 200 $9 * 9$ patches is roughly a visualization of our visual words in the dictionary, where 200 is the dictionary's size. That's to say, give a word in the dictionary, there is a corresponding patch that represents what the word looks like in a image. That' to say, if given a image, we can roughly tell its small patches's features(color), and compare them to the visualizing visual words and pick the word from the dictionary.

word in dictionary.png

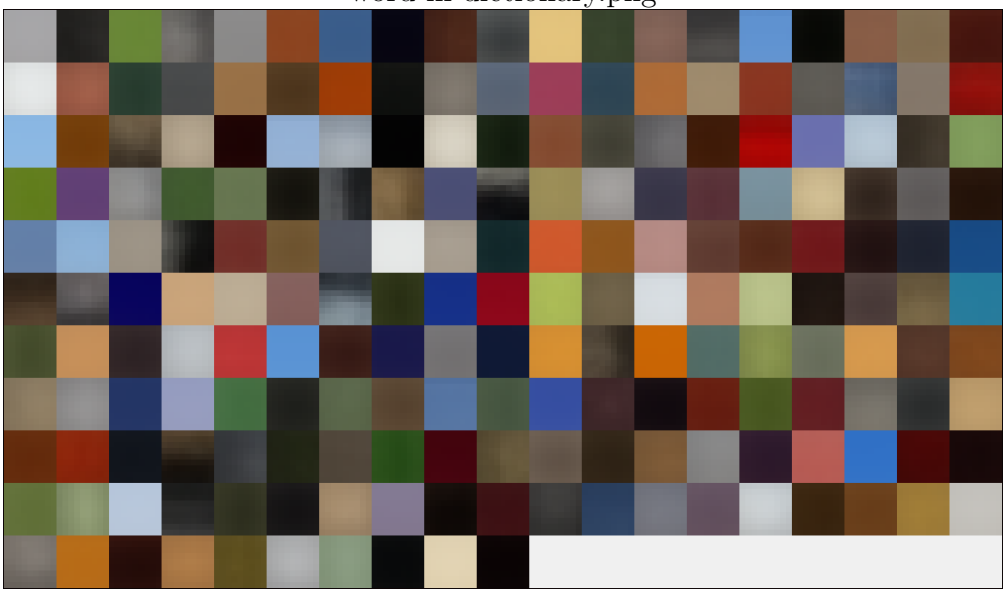


Figure 3: **Visualizing visual words**

Question 6.2 (10 pts)

As you can see, if we pick up two images from one class, and plot their top ten visual words, we can see there is a relation between these two images' patches. Their color is similar and can well represent what the images really look like. For example, in the class 'Bamboo', most of their visual words is green-related color, and the bamboo image is green. So it can really works well. You can refer to them below. (To be concise, I only show two images's visual words from class Bamboo, Basilica, Dam and Desert. And other 5 classes only has one image visualization of top 10 visual words.)

Question 6.3 (10 pts, 4-5 lines): Dataset Expansion

Classifiers would benefit greatly from more data. Generally, the larger the training set, the better the classifier. But flipping would not work under the bag of words representation, because bag of words representation doesn't represent spatial information. And for bag of words with spatial pyramid matching, it won't work too. For spatial pyramid matching, we use spatial histograms to represent images feature. Although the image has been flipped, the magnitude of different K is still the same. And the flipped image's spatial histogram which composeds of small subsets(each cell's histogram) just change the combination of these subsets from the original image, so we actually don't enlarge the data set.

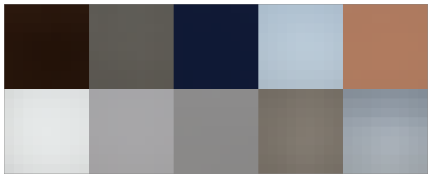
Bamboo1

Bamboo1 Top Words

Bamboo2

Bamboo2 Top Words

Basilica



Basilica Top Words

Basilica2

Basilica2 Top Words

Dam

Dam Top Words

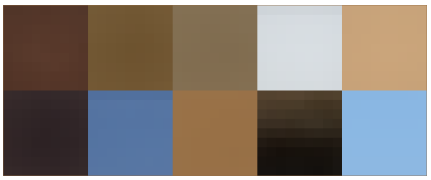
Dam2

Dam2 Top Words

Desert

Desert Top Words

Desert2



Desert2 Top Words

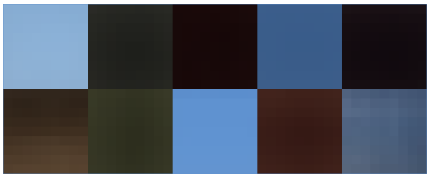
Kitchen

Kitchen Top Words

Railroad

Railroad Top Words

Sky

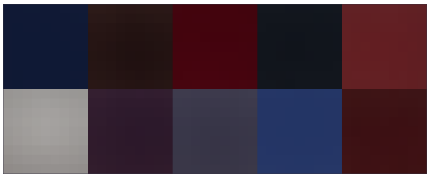


Sky Top Words

Supermarket

Supermarket Top Words

Theater



Theater Top Words

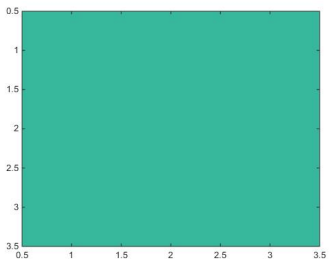


Figure 4: Average filter

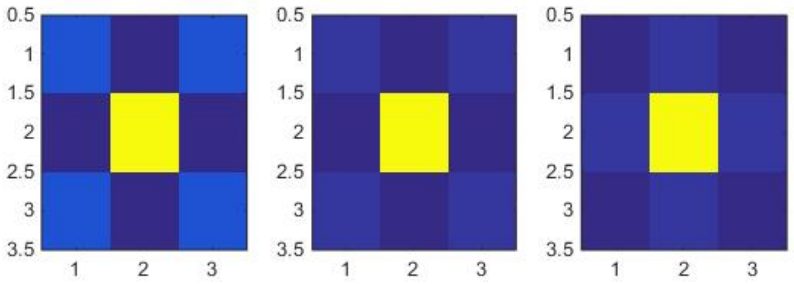


Figure 5: Unsharp filter

7 Extra Credit: Boosting performance (45 pts)

Question 7.1 Better filter bank (10 pts)

I have added two additional filters in my filter bank. They are Averaging filter and Unsharp Contrast Enhancement Filter. Averaging filter smooth the image evenly. For each pixel, its gray-scale value is replaced by the average value of the surrounding pixels. Unsharp Contrast Enhancement Filter is from the negative of the Laplacian filter. It improve the contrast ratio of the image. You can refer to them in Figure 4 and Figure 5.

Question 7.2 Better image similarity function (5 pts)

I tried Euclidean distance to measure similarity between two histograms.