16-720 Computer Vision: Homework 2
Bag-of-Words-based Object Classificaton

Instructor: Martial Hebert
TAs: David Fouhey, Heather Knight, Daniel Maturana

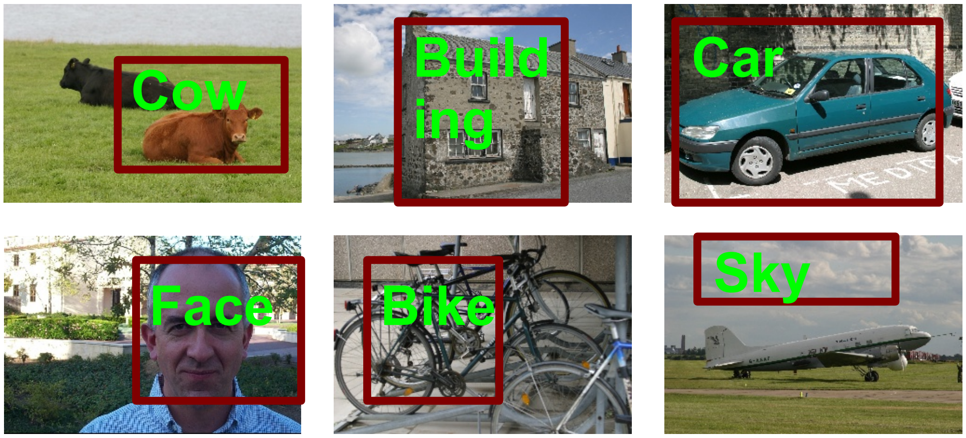


Figure 1: **Object Classification:** Given a bounding box, can we detect the object inside? In this homework, you will build a representation based on bags of visual words for classifying the contents of bounding boxes.

Instructions/Hints

1. **Please stick to headers, variable names, and file conventions provided.**
2. **Start early!** This will take longer than the last homework, and cannot be debugged as easily since there are multiple inter-connected components.
3. **(Corollary) Attempt to verify your implementation as you proceed:** If you don't verify that your implementation is correct on toy examples, you risk having a huge mess when you put everything together.

Overview

The bag-of-words (BoW) approach, which you learned about in class, has been applied to a myriad of problems in computer vision. Two classic papers on the topic[1] are:

[1] This homework aims at being largely self-contained; however, reading the listed papers (even without trying to truly understand them) is likely to be helpful.

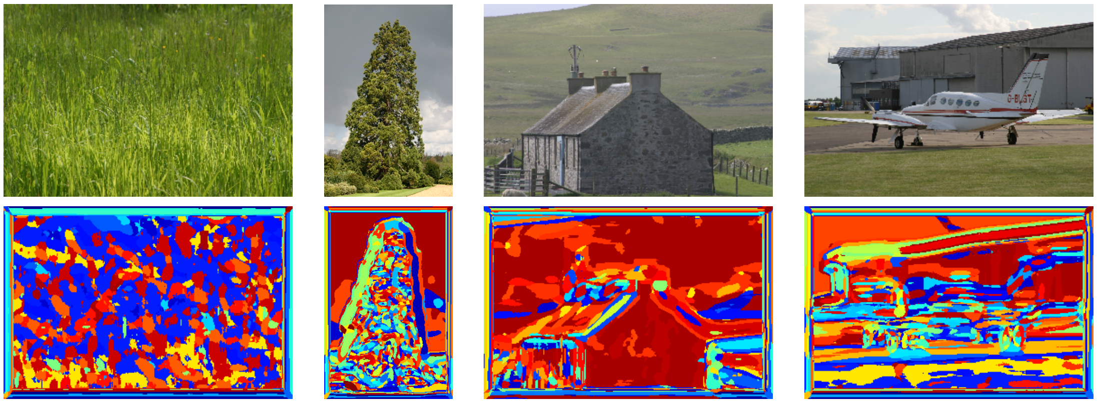


Figure 2: Visual words over the image. You will use the spatially un-ordered distribution of visual words in a region (a bag of visual words) as a feature for object classification.

J. Winn, A. Criminisi and T. Minka, Object Categorization by Learned Universal Visual Dictionary. ICCV 2005 `http://johnwinn.org/Publications/papers/Winn_Criminisi_Minka_VisualDictionary_ICCV2005.pdf`

L.W. Renninger and J. Malik, *When is scene recognition just texture recognition?*, Vision Research, 2004. `http://www.cs.berkeley.edu/~malik/papers/renningermalik.pdf`

However, the BoW representation has also been the subject of a great deal of study aimed at improving it, and you will see a large number of approaches that remain in the spirit of bag-of-words but improve upon the traditional approach (which you will implement here). Accordingly, these techniques often go by other names. One recent survey paper, which is worth glancing at to see the sheer volume of work in the vein of bag-of-words, is:

K. Chatfield, V. Lempitsky, A. Vedaldi and A. Zisserman. The devil is in the details: an evaluation of recent feature encoding methods. In BMVC 2011. `http://eprints.pascal-network.org/archive/00008315/02/chatfield11comparison.pdf`

One interesting paper on the topic of scene recognition that makes heavy use of the bag-of-words approach is:

J. Xiao, J. Hays, K. Ehinger, A. Oliva, A. Torralba. SUN Database: Large-scale Scene Recognition from Abbey to Zoo. In CVPR 2010. `http://web.mit.edu/jxiao/Public/publication/2010/CVPR_sun/paper.pdf`

If you are interested, look in the “Image Features and Kernels” section: HOG2x2, Dense SIFT, Sparse SIFT histograms, SSIM, Texton HIstograms, Color Histograms all use of the Bag-of-Words model.

What you will be doing: You will be implementing a bounding-box classification system that uses the bag-of-words approach, and applying it to the MSRC v1 dataset, which contains 240 images with 13 classes. In the course of the homework, you will do three things:

1. First, you will take responses of a filter bank on images, and build a dictionary of visual words.
2. Then, you will learn a model for the visual world based on a bag of visual words, and use nearest-neighbor to predict object classes in a test set. You will be specifically predicting the class or label (e.g., "car") of the object contained by a user-selected bounding-box (rectangle).
3. Finally, we ask you to try to improve the performance of your algorithm via a number of different ways.

A complete submission consists of a zip file with the following folders (please keep each system in a separate folder):

1. `baseline/`: the baseline system that you and everyone else implement through this homework
2. `custom/`: the custom system that you design as part of **Q3.1**
3. `segment/` (if you do the extra credit): a segmentation system that uses your recognition system
4. a writeup (pdf format).

When you submit, submit a copy of your system with the folder `data/` deleted, as well as any large temporary files that we did not ask you to create.

We provide you with a number of functions and scripts in the hopes of alleviating some tedious or error-prone sections of the implementation. You can find a list of files provided in Section 5. You can download this code and the MSRC dataset from:

Download the HW 2 package from:
`http://www.cs.cmu.edu/~dfouhey/hw2.zip`

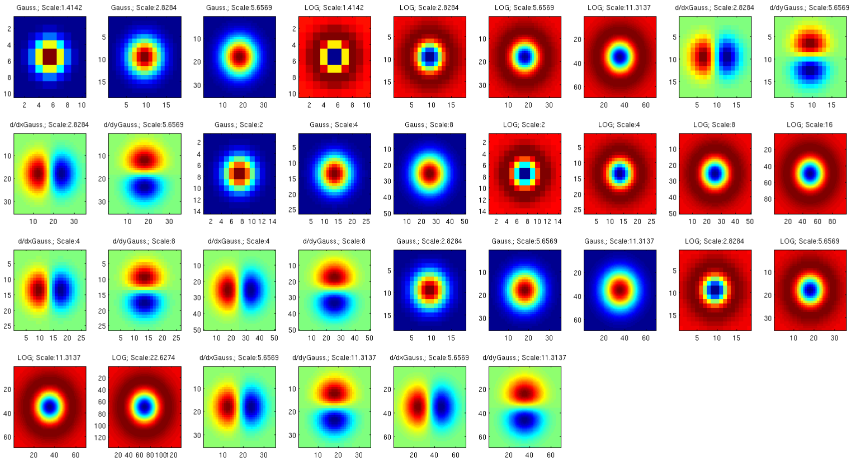


Figure 3: The multi-scale filter bank provided

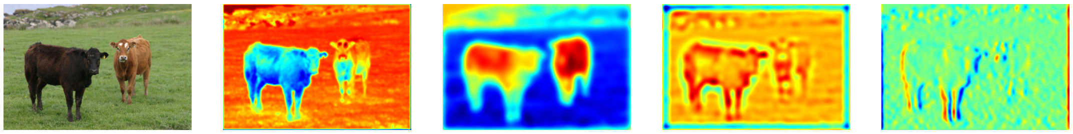


Figure 4: An input image, and four filter responses

1 Representing the World with Visual Words

We have provided you with a multi-scale filter bank that you will use to understand the visual world. You can get it with the following provided function:

```
[filterBank] = createFilterBank()
```

`filterBank` is a cell array[2] We have also provided you with a function to extract filter responses that takes a 3-channel RGB image and filter bank and returns the responses of the filters on the image.

```
[filterResponses] = extractFilterResponses(I, filterBank)
```

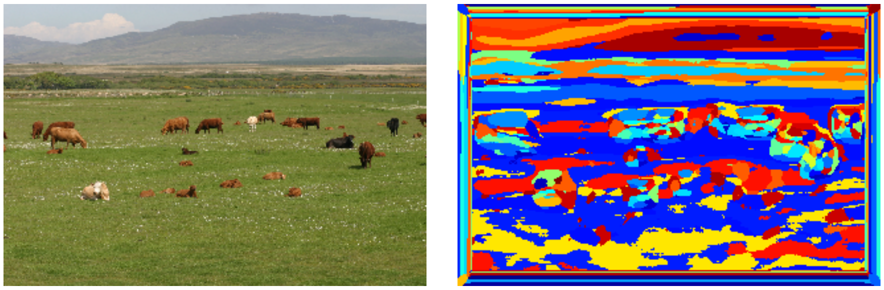
`filterResponses` is a $N \times M$ matrix, where N is the number of pixels in the input image, and M is the number of filter responses (three times the size of the filter bank, since you are applying it to a 3-channel image).

Q1.0 (5 points): What types of properties do each of the filter functions pick up? You should group the filters into broad categories (i.e., all the Gaussians). Answer in your writeup.

1.1 Creating Visual Words

You will now create a dictionary of visual words from the filter responses using k-means. After applying k-means, similar filter responses will be represented by the same visual word.

[2] Look at MATLAB's documentation for more details, but `filterBank{i}` is a 2D matrix, and `filterBank{i}` and `filterBank{j}` are not necessarily the same size.



Color input

Output map of visual words

Figure 5: A sample output, rendered with imagesc.

You will use a dictionary with fixed-size, as described in Section 3.1.3 of the paper by Leung and Malik. Instead of using all of the filter responses, you will use responses at α random pixels[3]. If there are T training images, then you should collect a matrix `filterResponses` over all the images that is $\alpha T \times N$, where N is the number of filter responses. Then, to generate a visual words dictionary with K words, you can cluster the responses with k-means using the built-in MATLAB function `kmeans` as follows:

```
[ignore,dictionary] = kmeans(filterResponses, K, 'EmptyAction','drop');
```

Q1.1 (15 points): You should write the following function to generate a dictionary given a list of images.

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

As an input, `getFilterBankAndDictionary` takes a cell array of strings containing the full path to an image. You can load each file by iterating from `1:numel(trainFiles)`, and doing `imread(trainFiles{i})`. Load the αT filter responses over the training files and call kmeans. Some sensible initial values to try for K and α are between 100 and 300, but you might want to play with these values.

Once you have written `getFilterBankAndDictionary`, call the provided script `computeDictionary`, which will pass in the filenames, and go get a coffee. If all goes well, you will have a `.mat` file named `dictionary.mat` that contains the filter bank as well as the dictionary of visual words. If all goes poorly, you will have an error message[4]. If it takes more than 10 minutes, reduce the number of clusters and samples. If you have debugging issues, try passing in a small number of training files manually.

1.2 Computing Visual Words

Q1.2 (15 points): We want to map each pixel in the image to its closest word in the dictionary. Create the following function to do this:

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

`wordMap` is a matrix with the same width and height as I, where each pixel in `wordMap` is assigned the closest visual word of the filter response at the respective pixel in `I`. We will use

[3] Try using `randperm`.

[4] Don't worry about "did-not-converge" errors.

the standard euclidean distance to do this; to do this efficiently, use the MATLAB function `pdist2`. A result is shown in Fig. 5.

Since this can be slow, we have provided a function `batchToVisualWords(numberOfCores)` that will apply your implementation of the function `getVisualWords` to every image in the training and testing set. This function will automatically[5] use as many cores as you tell it to use. For every image "`X.png`" in `data/`, there will be a corresponding file named "`X.mat`" in `dataProcessed/` containing the variable `wordMap`. You are *highly* encouraged to visualize a few of the resulting wordmaps; they should look similar to the ones in Figs. 2, 5.

2 Building a Recognition System

We have formed a convenient representation for recognition. We will now produce a basic recognition system. The goal of the system is presented in Fig. 1: given a bounding box, classify (colloquially, "name") the object contained inside. This is a common way of formulating the object recognition problem in computer vision, and many state-of-the-art systems use this approach. Typically, this is referred to as a "sliding window" approach, since when detecting objects, the computer looks at a window in the scene (i.e., the interior of some bounding box) and sees whether it recognizes anything. As output, the computer then lists where it thinks cows, sheep, and cars are. In this assignment, you will be doing something slightly different: you are given a particular window, and you are answering (i.e., classifying) what object is inside.

Traditional classification problems follow two phases: training and testing. During training time, the computer is given a pile of formatted data (i.e., a collection of feature vectors) with corresponding labels (e.g., "cat", "cow") and then builds a model of how the data relates to the labels: "if fuzzy and brown, then a cow"". At test time, the computer takes features and uses these rules to infer the label: e.g., "this is fuzzy and brown, so therefore it is a cow".

In this assignment, we will use the simplest classification model: nearest neighbor. At test-time, we will simply look at the query's nearest neighbor in the training set and transfer that label. In this example, you will be looking at the query bounding box and looking up its nearest neighbor in a collection of tens of thousands of bounding boxes. This approach works surprisingly well, and can do nearly as well as very complicated methods in a large number of domains. One very cool graphics application worth looking at is:

> James Hayes and Alexei Efros. *Scene Completion Using Millions of Photographs.* In SIGGRAPH 2007. `http://graphics.cs.cmu.edu/projects/scene-completion/scene-completion.pdf`.

The components of any nearest-neighbor system are: features (how do you represent your instances?) and similarity (how do you compare instances in the feature space?). You will implement each.

2.1 Extracting Features

We will represent our windows with a bag of words approach. In each window, we simply look at how often each word appears.

[5] Interested parties should investigate `batchToVisualWords.m` and the MATLAB commands `matlabpool` and `parfor`.

Q2.1 (10 points): Create a function `getWindowFeatures` that extracts the histogram[6] of visual words within the given window (i.e., the bag of visual words).

```
[h] = getWindowFeatures(bbox, wordMap, dictionarySize)
```

As inputs, the function will take:

- `bbox = [minX, minY, maxX, maxY]`
- `wordMap` is a `H` × `W` image containing the ids of the visual words
- `dictionarySize` is the maximum visual word id (i.e., the number of visual words)

As output, the function will return `h`, a `dictionarySize` × `1` histogram that is L_1 normalized, (i.e., $\sum h_i = 1$). You may wish to load a single visual word map, visualize it, and verify that your function is working correctly before proceeding.

2.2 Comparing Windows

We will need a way of comparing windows to find the "nearest" instance in the training data. In this assignment, we'll use the histogram intersection similarity. The histogram intersection similarity between two histograms $x_{1:n}$ and $y_{1:n}$ is defined as $\sum_{i=i}^{K} \min(x_i, y_i)$, or `sum(min(x,y))` in MATLAB. Note that since this is a similarity, you want the *largest* value to find the "nearest" instance.

Q2.2 (10 points): Create the function `distanceToSet`

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

where `wordHist` is a $K \times 1$ histogram with K histogram bins and `histograms` is a $K \times T$ matrix containing T histograms from T training samples concatenated along the columns. This function returns the histogram intersection similarity between `wordHist` and each training sample as a $1 \times T$ vector. Since this is called everytime you want to look up a classification, you want this to be fast, and doing a for-loop over tens of thousands of histograms is a very bad idea. Try `repmat` or (even faster) `bsxfun`[7].

2.3 Building A Model of the Visual World

Now that we've made all of the components, you want to put everything that you will need at test-time together.

You will need to load the training file names from `traintest.mat` and the filter bank and visual word dictionary from `dictionary.mat`. You will save everything to a matfile named `vision.mat`. Included will be:

1. `filterBank`: your filterbank.
2. `dictionary`: your visual word dictionary.

[6] Look into `hist` in MATLAB

[7] As a recommendation: unless you're experienced with MATLAB or confident, make sure your optimization works before moving on. Either use a few hand-made examples that you can manually verify or subtract the distances produced by the unoptimized and optimized examples.

3. `trainingInstances`: a $K \times N$ matrix containing all of the histograms of the N training windows in the dataset. I have a dictionary with 150 words and my `trainingInstances` is 150×71853.
4. `trainingLabels`: a $1 \times N$ vector containing the labels of each of the windows. (i.e., so that `trainingInstances(:,i)` has label `trainingLabels(i)`).

We have provided bounding boxes for each image in the folder `bboxes/` along with their labels ("car", "sheep", etc.). To get you comfortable with loading different types of data into MATLAB, we have provided the files in an alternate format than the usual `.mat` files. For every file "X.png" in the dataset, there is a corresponding pair of files in `bboxes/`, "X.bboxes.txt" and "X.ids.txt". The bboxes file contains a $B \times 4$ matrix of bounding boxes stored as minX, minY, maxX, maxY, and the ids file contains a $B \times 1$ giving the label of each bounding box. Use `dlmread` to read the files. It will work without any special syntax (i.e., `M = dlmread(myFileName);`). We have provided you with the names of the training and testing images in `traintest.mat`. You want to use the cell array of files `trainingFiles` for training, and the cell array of files `testingFiles` for testing. *You cannot use the testing images for training.*

From here, you want to take the bounding boxes and build your collection of training instances and their labels.

Q2.3 (15 points): Write a script named `buildRecognitionSystem.m` that produces `vision.mat`, and submit it as well as any helper functions you write.

Seeing things! (No points - just for fun/debugging)

We have provided an interactive demo program that will let you click and get predictions on a new image. This will give you a visual sanity check that you have implemented things correctly. Use the program as follows:

`demo2(absolutePathToImage)`,   (e.g., `demo2('fun/three.jpg')`)

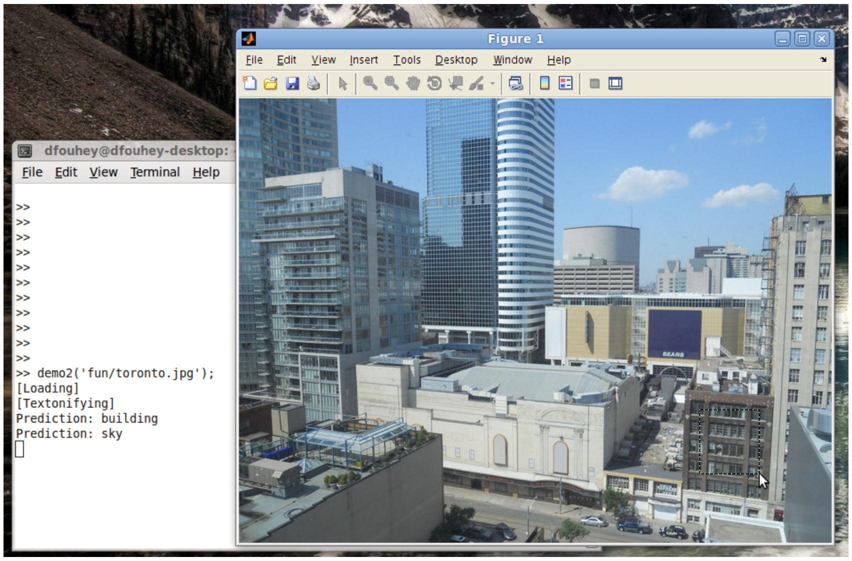
The program (see Fig. 6) will load the image, represent it with visual words, pop up the image in a window and let you click and drag a rectangle around the image to get a prediction based on the histogram. The predictions will appear inside your matlab session as text. Don't worry if you get a fair amount ($\approx 50-75\%$) of wrong answers. Do worry if the program crashes while calling your code or if you get zero correct answers, especially if it's the same wrong answer (e.g., "face" again and again). If you are getting 0% performance, go back and verify (visually) that each of your components produces correct output.

We have provided you with some "fun" images in `fun/`. We have provided you with some images from the testing set of MSRC in `MSRCtest/`.

2.4 Quantitative Evaluation

Qualitative evaluation is all well and good (and very important for diagnosing performance gains and losses), but we want some hard numbers.

Load the corresponding test boxes and their labels (stored in the exact same way as the training boxes), and compute the predicted labels of each loaded bounding box. To quantify


dfouhey@dfouhey-desktop:
File Edit View Terminal Help
Figure 1
File Edit View Insert Tools Desktop Window Help

```
>>
>>
>>
>>
>>
>>
>>
>>
>>
>>
>>
>> demo2('fun/toronto.jpg');
[Loading]
[Textonifying]
Prediction: building
Prediction: sky
```

Figure 6: Interactive demo program

the accuracy, you will compute a confusion matrix `C`: given a classification problem, the entry `C(i,j)` of a confusion matrix counts the number of instances of class `i` that were predicted as class `j`. When things are going well, the elements on the diagonal of `C` are large, and the off-diagonal elements are small. Since there are 13 classes, `C` will be 13×13. The accuracy, or percent of correctly classified windows, is given by `trace(C) / sum(C(:))`.

Q2.4 (15 points): Write a script named `evaluateRecognitionSystem.m` that tests the system and outputs the confusion matrix, and submit it as well as any helper functions you write. Report the confusion matrix for your results in your writeup. This does not have to be formatted prettily: if you are using LaTeX, you can simply copy/paste it from MATLAB into a `verbatim` environment. Additionally, do not worry if your accuracy is low: with 13 classes, chance is 7.7%.

3 Improving Performance

We want you to experiment with a variety of ways to possibly improve results. Please place all of your data / experiments on improving performance in a new folder named `custom/`. Use the same structure for training and testing (in fact, please copy from `baseline`) and submit all scripts and helper functions. Restrict yourself to MATLAB and any of its common toolboxes.

Q3.1 (15 points): Now you can bump up your accuracy on the validation set. Here are a list of things you can do to boost your performance. Pick some subset of the following adding up to at least 15 points (no extra credit). For each addition you make (i.e., improved filter bank), report the results in a confusion matrix, and **explain why you think the results changed (or didn't) the approach worked.** It is ok if you are not completely positive, but do give a plausible explanation **(at least three-five full sentences of actual explanation)**: gains in understanding are much more important than gains in performance.

Important: it is fine if performance does not change or decreases in terms of overall accuracy. Please report these results as well; they count as much as a performance gain.

There are other ways to improve performance (e.g., using SVMs or Random Forests for classification of windows); however, these require little thought on your part and just knowing the right toolbox to use, and therefore will not get you credit.

- **Better filter bank (5 points):** add filters to your filter bank. Explain in your writeup what each filter (or class of filters) is supposed to pick up on. Add at least two families of filters. Show your filter bank as we did in Fig. 3[8] and include this in the writeup.

- **Better window similarity function (5 points):** histogram intersection is nice, but there might be different distance and similarity metrics you can use: e.g., χ^2, Hellinger, Euclidean. Try at least one.

- **Multi-resolution (10 points):** bag of words ignores the spatial structure of the window, which might be suboptimal. Divide the window into a small number of cells, and concatenate the histogram of these cells to the histogram of the original window. One classic paper that uses this general idea is:
  S. Lazebnik, C. Schmid, and J. Ponce, *Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories*, CVPR 2006. `http://www.di.ens.fr/willow/pdfs/cvpr06b.pdf`

- **Different Features (15 points):** the bag of words representation is not tied to the particular featuers we're using: implement a new feature, and see what results you get. One option is local binary patterns; a classic work on the topic is:
  Ahonen, T., Hadid, A. and Pietikainen, M., Face Description with Local Binary Patterns: Application to Face Recognition. IEEE Trans. Pattern Analysis and Machine Intelligence 28(12):2037-2041, 2006. `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1717463&tag=1`

4 Extra Credit - Semantic Segmentation

QX1 (up to 20 points): Throughout this assignment, we've been focusing on classifying bounding boxes. However, this requires the user to point and indicate what objects they want identified. Ideally, we'd like to just label each pixel in the image as belonging to one of the classes. This problem is generally known as semantic segmentation, or the segmenting of the image into semantically meaningful regions (i.e., "chair", "cow"). One very successful approach to semantic segmentation is:

> J. Shotton, M. Johnson, and R. Cipolla. Semantic Texton Forests for Image Categorization and Segmentation. CVPR. 2008. `http://jamie.shotton.org/work/publications/cvpr08.pdf`

A less successful approach, built on the window-classification approach that you just implemented, is illustrated in Fig. 7.

Your goal is to take an image and the recognition system that you've built, and produce a semantic segmentation of the scene. To constrain the problem: you are permitted to use

[8] Hint: `subplot`

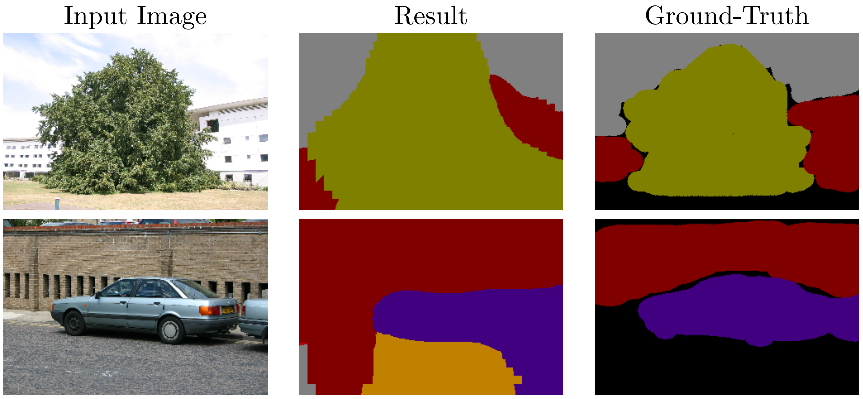


Figure 7: Extra-credit: results from a very basic approach and the ground-truth from MSRC

any tweaks from your custom implementation; however, the result must be fundamentally built upon the output of your nearest-neighbor window-classifier. You must also only use tools found in MATLAB.

Place any extra-credit work in the folder named `segment/`, and include a brief write-up of your system with 2 of your best results, as well as 2 failure cases, and some thoughts on them. Note that failures are often more illustrative and informative than successes. Use the provided function `MSRCvis` to visualize your results.

5 HW2 Distribution Checklist

After unpacking `hw2.zip`, you should have a folder `hw2` containing one folder for each system you might implement (`baseline`, `custom`, `segment`). In the `baseline` folder, where you will primarily work, you will find:

- `bboxes/`: a directory containing 480 text files giving the bounding boxes of training and testing windows.
- `data/`: a directory containing 240 `.png` images.
- `fun/`: a dataset containing some images that might be fun to play with.
- `MSRCTest/` a dataset with images in the test set that might be useful for diagnosing issues.
- `batchToVisualWords.m`: a provided script that will run your code to convert all the images to visual word maps.
- `computeDictionary.m`: a provided script that will provide input for your visual word dictionary computation.
- `createFilterBank.m`: a provided function that returns a cell array of filters.
- `demo2.m`: a provided script that will let you do the click and drag demo.
- `extractFilterResponses.m:` a provided function that takes a 3-channel RGB image and filterbank and provides the filter responses at each pixel.
- `MSRCvis.m`: a helper function for the extra credit problem.
- `RGB2Lab.m`: a provided helper function.
- `traintest.mat`: a `.mat` file with the filenames of the training and testing set.