Machine Learning: Foundations **March 2013**

Empirical Data (Final) Project

Yaniv Bar

"Tackling the problem of automatic detection, recognition and segmentation of object classes in photographs"

based on

TextonBoost: Joint Appearance, Shape and Context Modeling for Multi-Class Object Recognition and Segmentation
J. Shotton, J. Winn, C. Rother, and A. Criminisi

Abstract:

The paper investigates the problem of achieving automatic detection, recognition and segmentation of object classes in photographs. Namely, given an image the system is expected to automatically partition the image into semantically meaningful areas each labeled with a specific object class.

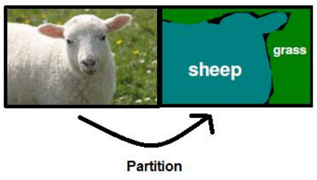


Figure: Image partitioning into semantically meaningful object classes

This problem is far from being an easy task. Just think on a local level - a window can be part of a car, a building or an aeroplane. Hence; there are ambiguities which the system must overcome.

It is important to this problem was avoided by almost all researchers in the field mainly due to the fact that object recognition and segmentation are considered as separate tasks.

The project provides a MATLAB limited but working implementation to the massive problem raised by the paper and achieves several goals such as:
- Tackling ("playing") with real research data
- Developing sophisticated feature extraction (Texton based shape filters)
- Developing auxiliary machine learning classification algorithm (fast K-means)
- Multi-class boosting algorithm implementation for recognition and segmentation (AdaBoost.Mh multiclass reduction to binary variation)
- And more…

Note that although the paper goal is to efficiently detect the object classes and give segmentation of an image into these classes, in practice the paper (and the limited MATLAB implementation) shows that it is a massive computational task. This being said to note that the more "classical" machine learning analysis based on training set size and common bag of algorithms is replaced in this paper (and project) to the novelty of the algorithms and accuracy level.

High Level Description and Execution Information:

High level algorithm of the system:

Learn a boosting classifier based on relative texture locations for each class. The classification is then refined.

Given an image and for each pixel:
- Texture-Layout features are calculated
- A boosting classifier gives the probability of the pixel belonging to each class
- The model combines the boosting output with shape, location, color and edge information; Pixel receives final label. Image receives final label.

The MATLAB implementation Code-Sequence of execution is described below and will be elaborated in following sections.

1. imagesTextonization.m	← extracts efficient (texture-layout) train images characterization
2. calcModelFeatures.m	← calculates the appearance (shape) potential context
3. trainModel.m	← builds the boosting multi-class classification model
4. testModel.m	← tests the classification model with test data

The directory structure for the MATLAB implementation is as followed:

1. The code and examples are packed within a self contained compressed file.
2. When unpacked, the project folder holds all MATLAB code (~30 code files), a few test images And the following directory structure:

- Unpacked folder\ClassificationAttempt\Images ← Holds Train images
- Unpacked folder\ClassificationAttempt\GroundTruth ← Holds Train GT images

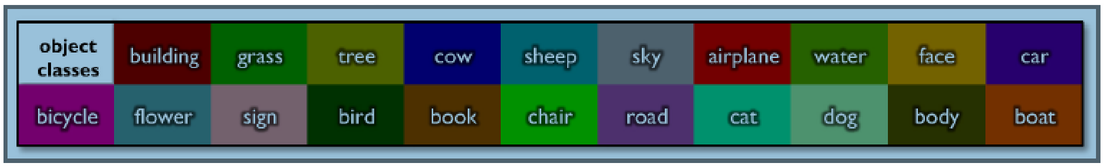
Following imagesTextonization.m (step #1) execution:
- Unpacked folder\ClassificationAttempt\TextonizedImages ← Holds Textonized Train images
- Unpacked folder\ClassificationAttempt\TextonizedImages\View

Following calcModelFeatures.m (step #2) execution and trainModel.m (step #3) execution:
- Unpacked folder\ClassificationAttempt\Model ← Holds Model and Model Preliminaries

Following testModel.m (step #4) execution:
- Unpacked folder\ClassificationAttempt\Test ← Holds Test image and GT image and GT output Image

In addition,
- Unpacked folder\ClassificationAttempt\attempt[1-3] ← A Few Classification Attempts With The same directory structure

Data Sets:
The classification models in the paper (and project) are learned from a set of labeled training images.
- The database is composed of photographs of 24 object classes such as grass, tree, cow etc. including void which acts as an unassigned object class
- The photographs were taken under general lighting conditions, camera viewpoint, scene geometry, object pose and articulation.
- The training images were hand-labeled with the assigned colors acting as indices into the list of object classes.
- The data set can be downloaded using the following Microsoft Image Understanding Research Data link:
  http://research.microsoft.com/en-us/projects/objectclassrecognition

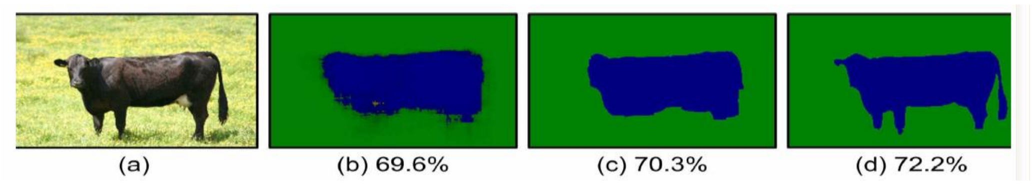


object classes	building	grass	tree	cow	sheep	sky	airplane	water	face	car
bicycle	flower	sign	bird	book	chair	road	cat	dog	body	boat

Figure: Object Classes

Building the Boosted Learning Model:

The most important part of the model is the Shape/Context Potential – it is significant for object recognition and very rough segmentation results. Other potential such as Edge and Color refine the segmentation results.



(a) Original image, (b) Shape, (c) (b)+edge, (d) (c)+color

Figure: Effect of different model potentials.

The Shape/Context potential part is based on a novel set of features which we call shape filters. These features are capable of capturing shape, texture and appearance context jointly.

Boosted Learning of Shape, Texture and Context:

Efficiency demands compact representations for the range of different appearances of an object. For this we utilize a concept called 'Textons' which have been proven effective in categorizing materials as well as generic object classes.

For e.g. in categorizing materials the task is to recognize surfaces made from different materials on the basis of their texture appearance. Now, different materials show different texture appearance And moreover, texture appearance of the same material changes dramatically due to different viewpoint/lighting settings (shadows, occlusions, etc.).

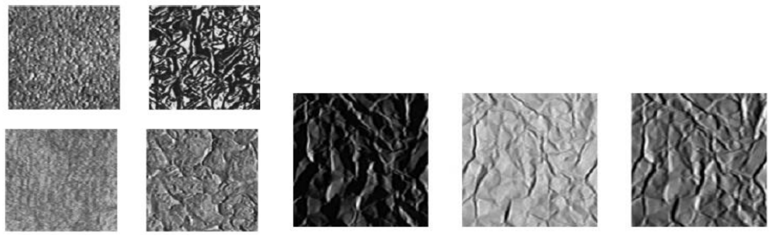


Figure: Effect of different model potentials.

Calculating Texture-Layout features:

A dictionary of textons is learned by convolving a 17-dimensional filter bank with all the training images and running K-means clustering on the filter responses. Finally, each pixel in each image is assigned to the nearest cluster center, thus providing the texton map.

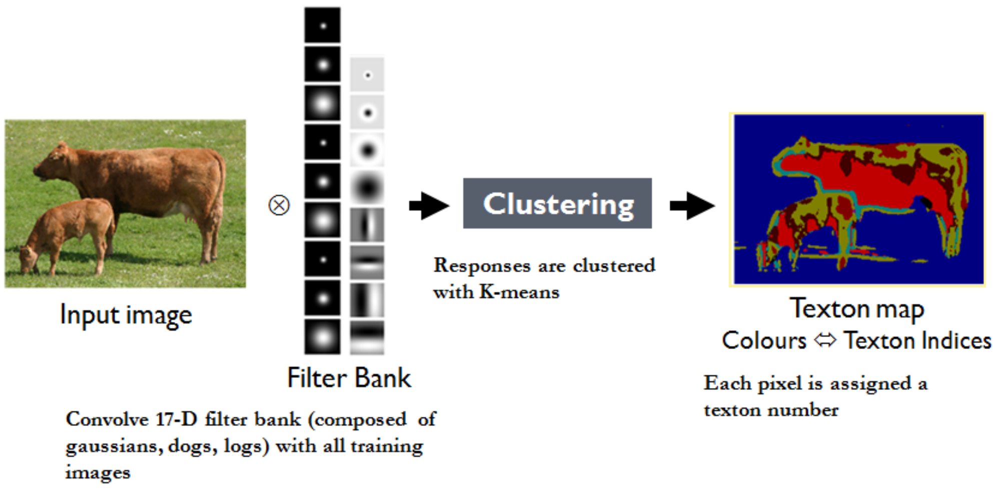

Input image
Filter Bank
Convolve 17-D filter bank (composed of gaussians, dogs, logs) with all training images
Clustering
Responses are clustered with K-means
Texton map
Colours ⇔ Texton Indices
Each pixel is assigned a texton number

Figure: Computing a Texton map

The choice of filter-bank is somewhat arbitrary, as long as it is sufficiently representative (The paper/project uses Gaussians at scales k, 2k and 4k, x and y derivatives of Gaussians at scales 2k and 4k, and Laplacians of Gaussians at scales k, 2k, 4k and 8k. The Gaussians are applied to all three color channels, while the other filters are applied only to the luminance.

The MATLAB implementation of 'imagesTextonization.m' (Step #1 of the code execution sequence) calls 'imgTextonization.m' for every train image based on the filter bank which was described above, thus, generating a texton-map image. A fast version of K-means clustering algorithm was also developed within the scope of the project (a test script file is also provided).

The MATLAB implementation 'testFilters.m' depicts the Gaussian based filters generation used in the generation of the filter bank.

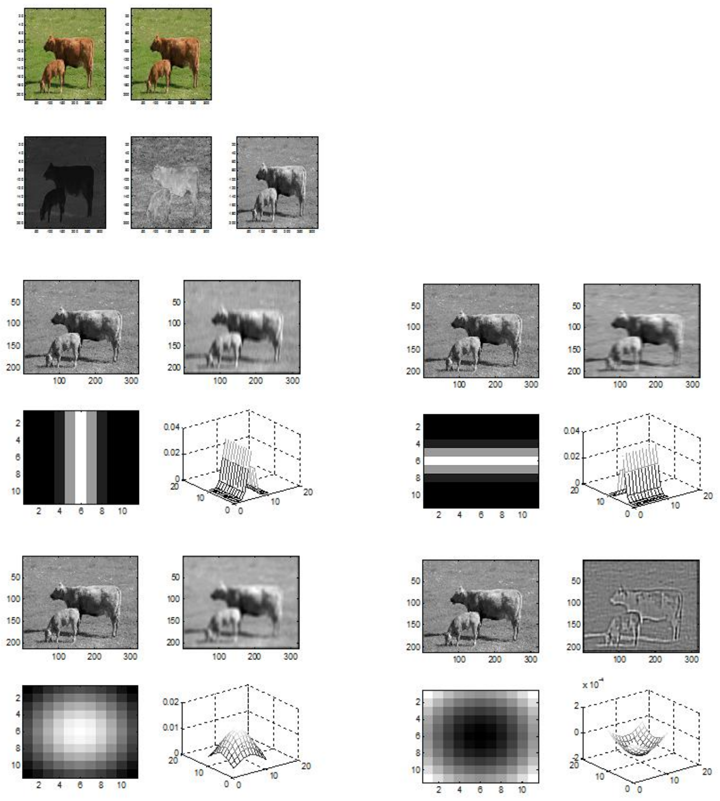


Figure: Gaussian based filters

The MATLAB implementation 'testImgTextonization.m' depicts the generation of a single texton-map given an image.

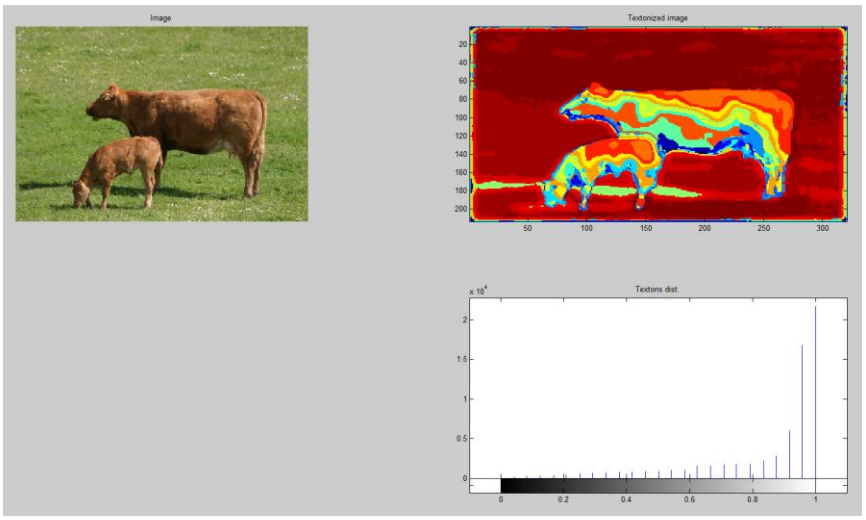
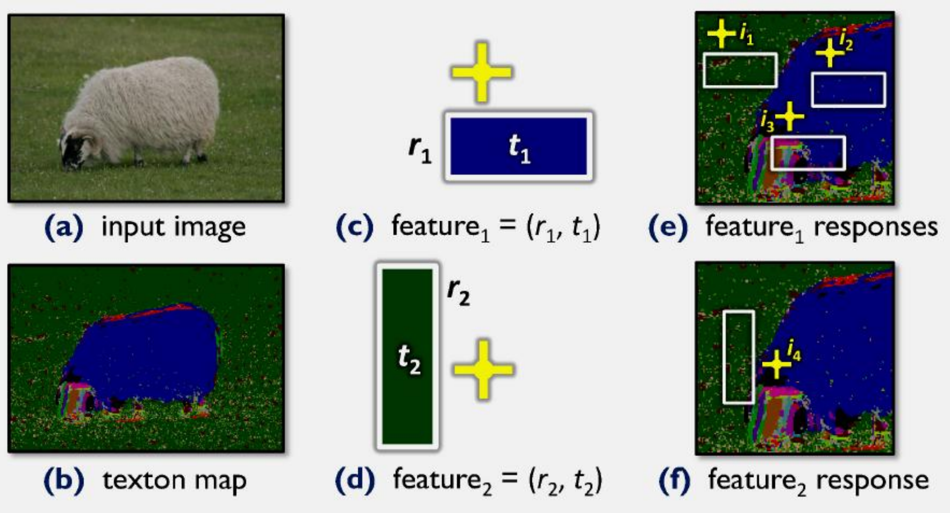


Figure: Computing a Texton map

<u>**Calculating The Shape Filters:**</u>

Shape filters consist of a set of rectangular regions whose four corners are chosen at random within a fixed bounding box. For a particular texton *t*, the feature response at location *i* is the count of instances of that texton under the offset rectangle mask.
The filter responses can be efficiently computed over a whole image with integral images:The texton map is separated into K channels (one for each texton) and then, for each channel, a separate integral image is calculated.

These features are sufficiently general to allow us to learn automatically shape and context information.



Calculating feature responses and capturing textural context. (a, b) An image and its corresponding texton map (colors map uniquely to texton indices). (c) Texture-layout filters are defined relative to the point i being classified (yellow cross). In this first example feature, rectangular region r_1 is paired with texton t_1 (mapping to the blue color). (d) A second feature where region r_2 is paired with texton t_2 (green). To improve readability, (c) and (d) are shown double size compared with (e) and (f). (e) The response $v_{[r_1,t_1]}(i)$ of the first feature is calculated at three positions in the texton map (magnified). In this example $v_{[r_1,t_1]}(i_1) \approx 0$, $v_{[r_1,t_1]}(i_2) \approx 1$, and $v_{[r_1,t_1]}(i_3) \approx \frac{1}{2}$. (f) For the second feature (r_2, t_2), where t_2 corresponds to 'grass', our algorithm can learn that points i (such as i_4) belonging to sheep regions tend to produce large values of $v_{[r_2,t_2]}(i)$, and hence can exploit the contextual information that sheep pixels tend to be surrounded by grass pixels.

Figure: Extracted from paper

The MATLAB implementation of 'calcModelFeatures.m' (Step #2 of the code execution sequence) calls 'calcImgFeatures.m' for every train texton image map, which in turn separates the texton map into multiple channels: The texton map of an image, containing *K* textons, is split into *K* channels. An integral image is built for each channel and used to quickly compute texture-layout filter responses. We've implemented vast configurations capabilities which allow us to select a fixed or random bounding box, variety number of rectangular regions and if its four corners are selected randomly or not.

In addition, the considerable memory and processing requirements make training on a per-pixel basis impractical. This is why for efficiency reasons Sub-sampling and random feature selection was also introduced in the project as an option in order to enable the reduction in computation expense.

The MATLAB implementation 'testCalcImgFeatures.m' allows you to select 3 pixel locations, *i*, using the mouse and to see the distribution of the filter responses (it calls 'calcImgFeatures.m').

The MATLAB implementation 'testFeatureResponse.m' depicts a few filter responses against a texton image map (similar in spirit to the figure above).

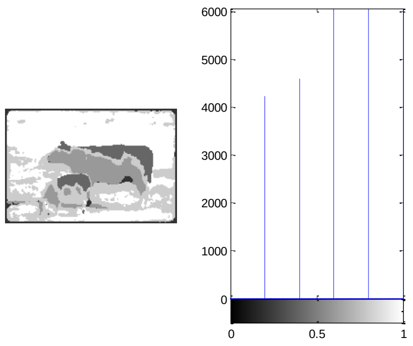


Figure: Texton-Map of an image (5 layers)

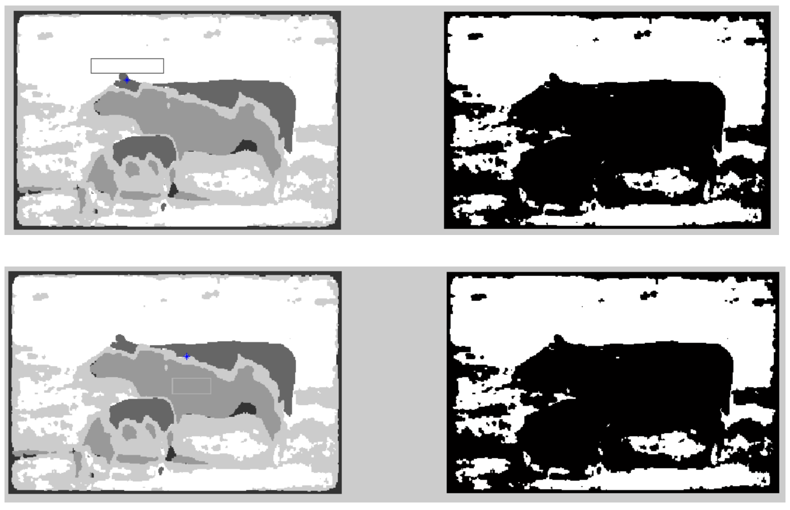


Figure: (right) Relevant texton layer (W is the layer). (top) Response is close to 1. (bottom) Response is 0.

```
textons =  0.2000, 0.4000, 0.6000, 0.8000, 1.0000

featureResponse = 991

featureResponsePerc = 0.9176

featureResponse = 0

featureResponsePerc = 0
```

The MATLAB implementation 'testFeatureExtraction_ToyExample.m' shows how shape filters are able to model appearance-based context by showing this against a toy example (in the spirit of the example in the paper where the rectangular region is a region located in the top-left location of the pixel). This toy example illustrates how shape filters capture relative positions of textons.

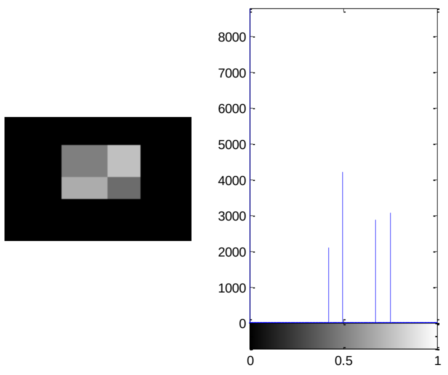


Figure: Toy example. Texton-map.

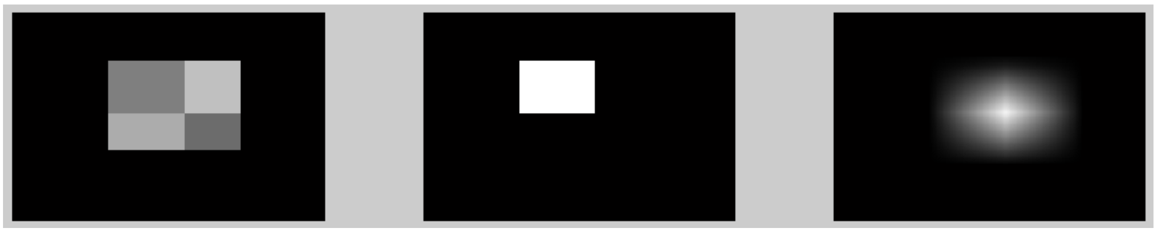


Figure: Toy example. (Middle) Relevant texton layer. (Right) Since the rectangular region is a region located in the top-left location of the pixel we see an offset to the right-bottom corner of the relevant texton map.

Building the Classifier:

The paper employs an adapted version of the multiclass boost algorithm which joins the different unary potentials such as shape, context, edges and colors. As discussed above, the most important part of the model is the Shape/Context unary Potential which produces a good object recognition results with vague segmentation accuracy where the added unary potential refines the perceived segmentation accuracy. In the project we have focused only on the shape and context unary potential and for this reason we've have adapted a multi-class adaboost algorithm, 'AdaBoost.Mh' which is in essence a multiclass reduction to binary adaptation.

The multi-class classiffier is learned iteratively builds a strong classiffier as a sum of `weak classifiers', by selecting discriminative shape and context features. Each weak classiffier is a decision stump based on a thresholded feature response allowing a single feature to help classify several classes at once.

The MATLAB implementation of 'trainModel.m' (Step #3 of the code execution sequence) calls the multi-class boosting classifier on the random shape and context extracted feature which were extracted in Step #2 of the code execution sequence.

Testing the Classifier:

The MATLAB implementation of 'testModel.m' (Step #4 of the code execution sequence) tests the object class recognition and segmentation accuracy level of the classifier given a test image.

The testing of the classifier is done by extracting the test image shape and context random features, "feeding" it to the classifier model resulting in a object class segmented image. The accuracy level is obtained by comparing this result against the true object class segmented ground truth image.

The classifier model generation consumes a massive computational power. The paper notes training phase which takes 42 hours and up to 14000 hours without random feature selection.
We have implemented random feature selection and sub-sampling to improve the training efficiency however we avoided full training phase and focused on relatively very fast generated classifier model generation (done using a few training images) focusing on the accuracy level.

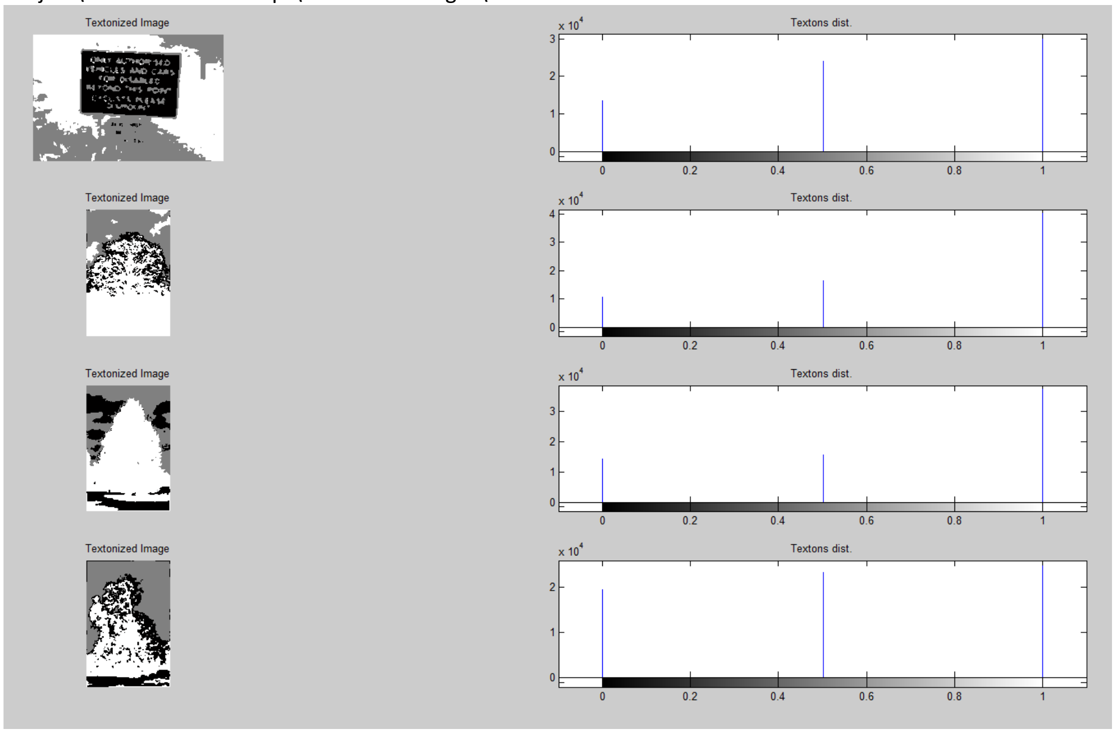
Execution Walk-Through and Environment Notes:

The entire project was written in Matlab R2012a.

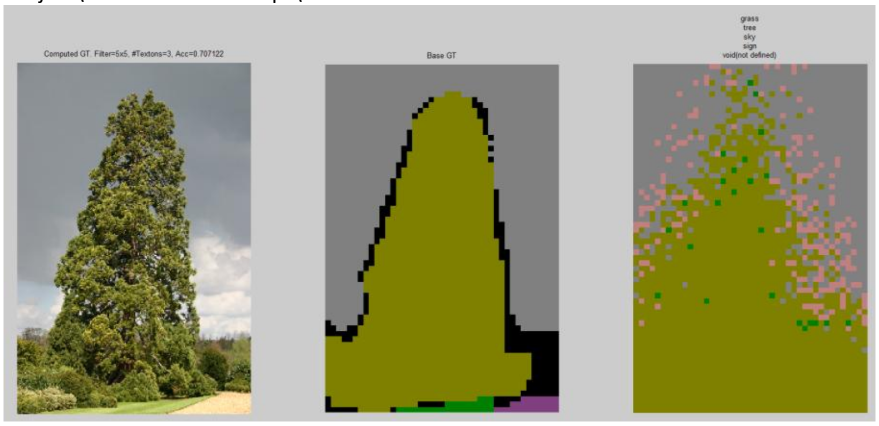
The following notes describes a Walk-Through for executing the code assuming that the project was packed in a given folder which will be denoted as 'Project' folder. Note that for your convenience the necessary directory structure is already set-up with appropriate files.

Walk-Through:

1. Place train images in 'Project\ClassificationAttempt\Images'.
2. Place corresponding GT train images in 'Project\ClassificationAttempt\GroundTruth'.
   Remark: The 2 folders currently holds 4 train and GT train images describing 5 object classes (grass, tree, sky, sign and unclassified object).
3. Execute 'imagesTextonization.m' (step #1) script.
   This will generate textonized train images and adapted textonized train images for viewing purposes in 'Project\ClassificationAttempt\TextonizedImages' and 'Project\ClassificationAttempt\TextonizedImages\View' correspondingly.
   In addition, the textonization distribution figure by the name of 'distFig.fig' will be saved in 'Project\ClassificationAttempt\TextonizedImages\View' folder.



4. Execute 'calcModelFeatures.m' (step #2) script.
   This will generate classifier model preliminaries (mainly shape and context information) of the textonized images in 'Project\ClassificationAttempt\Model'.
5. Execute 'trainModel.m' (step #3) script.
   This will generate classifier model in 'Project\ClassificationAttempt\Model'.
6. Place a test image in 'Project\ClassificationAttempt\Test'.
7. Place a corresponding base GT test image in 'Project\ClassificationAttempt\Test'.
   Remark: The folders currently holds a test and GT test image.
8. Execute 'testModel.m' (step #4) script.
   This will generate the textonized image, sub-sample base GT image and the sub-sample output object class segmentation image in 'Project\ClassificationAttempt\Test'. It will also calculate the accuracy level against the sub-sample base GT test image which was placed in the previous step.
   Along with the resulting image, a figure depicting the accuracy level result will be saved in 'Project\ClassificationAttempt\Test' folder.



Experimental Results and Analysis:

Several attempts for classification are located in 'Project\ClassificationAttempt\attempt[1-3]'.

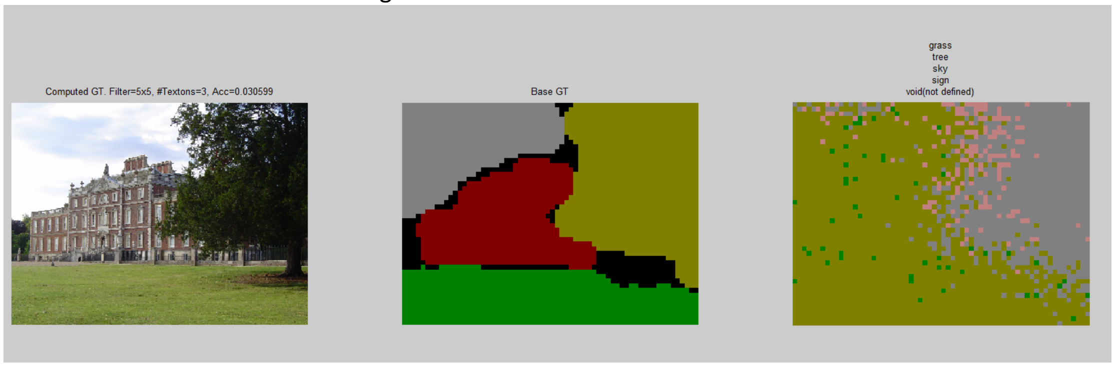
Applying our algorithm to test images we've managed to get an accuracy level of more than 70% in several multi-class attempts which is similar to results obtained in the paper. This was done only with the boosting classifier alone without any additional refinement. If we also reduce the sub-sampling effect (which costs in computational time) we even increase the results slightly. The computational time needed for testing an image against the classifier is relatively fast and takes about 1 minute.

For example, in the test discussed above the accuracy level was 70.7% with the following TP/FN table results which reveals a good object recognition and segmentation for Tree and Sky object class with very poor Grass and Sign object class (although the poor detection does not affect the overall results dramatically as one can see from the relatively few incorrect classified pixels).

	TP Rate	FN Rate
Grass	0	1
Tree	0.8597	0.2786
Sky	0.7015	0.1616
Sign	0	1

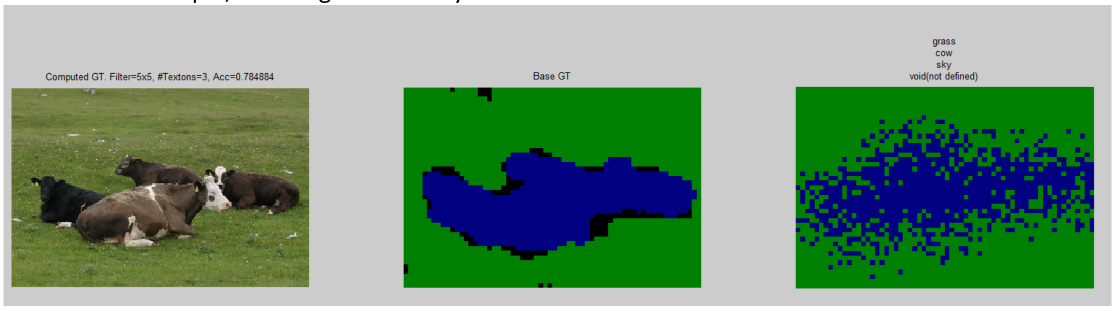
The "power" of the system is handy when we test the system on a counter example which was not trained by the boosting classifier. Here we get the expected very poor results of about 5% accuracy level. This can be explained both from the lack of training for specific object classes and the intentionally damage which was introduced to the shape-context filter (generating improper contextual features which is learned by the classifier).

The results are shown in the next figure and table.



	TP Rate	FN Rate
Grass	0.0156	0.7885
Tree	0.0628	0.9696
Sky	0.0427	0.9738
Sign	0	1

Yet another example, reaching an accuracy level of 78%.



Countless attempts were performed to reveal meaningful inferred conclusions when changing the filter bank kernels or the number of textons. In brief, increasing the number of textons to a large value damages the shape-context filter ability to create strong links (introducing noise) and reduces the accuracy level from about 60% to about 40%. Increasing the filter bank kernel size can benefit the results in those situations due the nature of the Gaussian filter to reduce the noise influence.
A relatively large attempt for classification is located in 'Project\ClassificationAttempt\attempt3' with several result figures.

<u>A Short Conclusion:</u>

The Good:

- Provides reasonable recognition + segmentation for many classes
- Combines several good ideas from various subject matters such as machine learning and machine vision.
- Most of previous works didn't tackle the problem as a whole – rather, problems were treated separately.

The Bad:

- Does not beat past work (in terms of quantitative recognition results)
- Difficult to program
- Computationally expensive (in practice)

Project Website and Files:

The project website is at:
http://www.cs.tau.ac.il/~yanivba1/ML2012_project.html

The website refers you to the followings:
- The final paper accompanied with a short presentation
- A compressed '.rar' file containing all project files (data, code and classification attempts)
- Full data set reference
- Papers references

In addition, these files are available under TAU server, at:
~yanivba1/ML2012/FP

References:

[1] TextonBoost: Joint Appearance, Shape and Context Modeling for Multi-Class Object Recognition and Segmentation
[2] TextonBoost for Image Understanding: Multi-Class Object Recognition and Segmentation by Jointly Modeling Texture, Layout, and Context