VRVis Research Center

Implementation of Semantic Texton Forests for Image Categorization and Segmentation

Technical Report

Document Version: 2015-06-09

Document Authors: Attila Szabo, Stefan Maierhofer

Table of Contents

1. Introduction

During an internship at the *VRVis Zentrum für Virtual Reality und Visualisierung Forschungs-GmbH* from March to June 2015, an image classification and segmentation system developed by Shotton et al. [1] has been implemented and tested on internal and publicly available data sets. The goal of the internship was the creation of a framework to facilitate the according image classification and segmentation workflows. The resulting framework has been packaged into an open source library and is publicly available at *https://github.com/vrvis/aardvark.semantictextonforests* [5]. Additionally, during the internship *libsvm.clr*, a clean and fast C# wrapper for the popular Support Vector Machine library *libsvm* [3], has been created and is available at *https://www.nuget.org/packages/libsvm.clr.vs2015.x64/* [4] .

The performance of the framework is at least comparable to or better than the basic tools used internally at the time, either in terms of accuracy or computation time. It is modular and extensible, facilitating adaptation to specific problems and the implementation of possible future improvements (see Section 4). The operations of categorizing or segmenting an image are performed using the concept of Semantic Texton Forests. These Forests are extremely fast to evaluate and provide a very expressive feature representation of image data, particularly of photographs. A Forest represents a large number of very simple pixel features organized in a hierarchical structure, allowing it to capture not only the visual properties, but also a sense of the context and the semantics of image data.

The system can be trained using natural photographs, removing the need for expensive image filtering operations otherwise often associated with the term “Texton". However, the feature extraction operates on the intensity values of the pixels directly, which means that some ramifications have to be kept in mind. Insensitivity to illumination change can not be guaranteed for most features, but the user has the option to select only those features with this property (see Section 2.2). Also, the approach is conceptually not invariant to change in rotation. If necessary, it is a good idea to include rotated copies of the training data into the training set (for example 7 copies, each rotated by 45° compared to the previous one).

The scope of the internship focused on the implementation of the Image Categorization and Segmentation system described in the original paper [1] and the details learned from the according reference implementation. Beyond that, the internship necessitated the creation of our libsvm wrapper, which was not part of the original work (which used a different SVM library entirely). On the other hand, there are several combinations of evaluation methods described in the original work, but the internship only focused on the ones which were reported to produce the best results, those are described in this report. Furthermore, the original work cites two advanced use cases, the soft Categorization of images using the Classifier and unsupervised learning using the Texton Forest, both of which were not implemented during the internship.

In Section 2, details about the implementation of the framework are discussed. Section 2.1 goes into special detail about the generation and application of Semantic Texton Forests.

Section 2.2 describes the use of a Forest for the purpose of image classification. In Section 2.3, the method is extended for use in image segmentation. In Section 3, some examples are presented and discussed. It also includes a quick start guide for working with the library. Finally, Section 4 includes a summary and some still open topics for future work.

2. Implementation

In order to apply classification and segmentation operations on a set of images or image patches, appropriate data structures with the required expressive capabilities have been implemented. These data structures are used to represent data elements and the relationships between them in the algorithms for building and running the system.

The system uses lists of Class Labels to define all images' possible class memberships and classification outcomes within a work flow. One Class Label has an Index, which is its unique numeric identifier, and an optional friendly Name for better readability. Note that the Class Label list defines within the system every possible label that may occur, and may also contain Labels which are not included in the training data set (such as the “Unknown" label). Using one of these Class Labels each, a set of pictures is loaded into the system as Labeled Images, which combine the Label with a PixImage from the Aardvark library. By using data types from the Aardvark library, high performance and optimal memory management can be ensured. Currently, there is no generic method for reading images from disk and connecting them with a Label since this would be outside of the internship's scope. The user has to do this manually, for example by parsing the file name or path. Alternatively, the user can supply a Label Map, an image which specifies a Label for each pixel (for example by color-coding the pixels). After loading the images, they can be split into a training set, which is used to train the system parameters, and a test set, using which the classification and segmentation performance is evaluated.

To generate a feature representation of an image, a Semantic Texton Forest is created. A Semantic Texton Forest is a collection of binary Trees, in which each node reaches a binary decision by evaluating a very simple local pixel feature. When an image region is passed through a Tree from root to leaf, the path it takes forms a hierarchical representation of the local textural structure. The expressive power and computational performance of a Forest depends on the completeness of the training set and is determined by several parameters, including:

- Number of Trees
- Maximum depth of Trees
- Training subset size per Tree
- Sampling method of the Images
- Feature calculation from Image samples

These parameters are often correlated and the quality of the result can heavily depend on the correct choice. Additionally, different parameters perform variously well for different problems. Implementing a general heuristic for finding good parameters would be possible, but would have been outside of the internship's scope. However, some generally acceptable default values, as well as a framework for testing parameter variations and displaying statistical results, are provided in the code to facilitate parameter exploration.

2.1 Forest Training

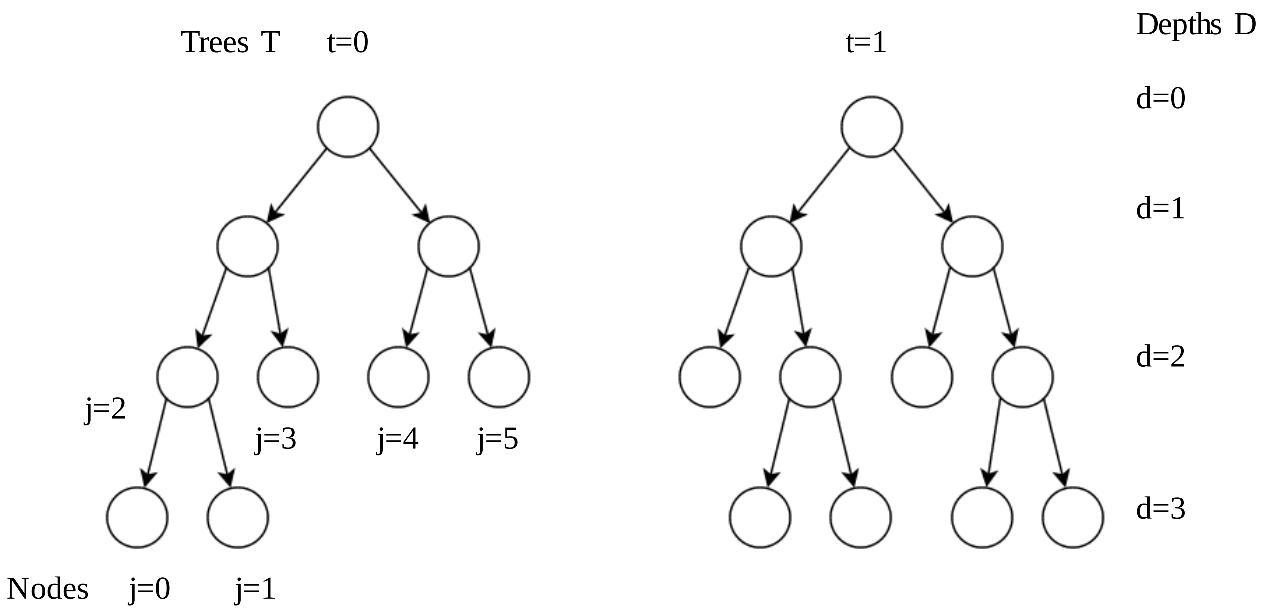

Figure 1: Example for a Semantic Texton Forest

During Forest training, each tree is trained independently. To improve generalization, only a random subset of the training images is presented to each Tree. The Tree possesses a Sampling Provider, which extracts a collection of Data Points from the images. This is done either by dense sampling or, to reduce computation time, regular grid sampling. Each Data Point consists of coordinates in pixel and color channel dimensions, and the Label of its source image. A Tree is initialized with a root Node and all Data Points are presented to it. The Node has a Feature Provider, which considers a small pixel window around a Data Point and calculates a numeric value using arbitrarily chosen pixels within that window:

- the intensity of a pixel,
- the sum of two pixels,
- the (absolute) difference of two pixels.

These Feature Values are directly compared to the Node's threshold value, effectively splitting the Data Point set into left and right subsets depending on whether the feature value is less than or greater than the threshold. To find an appropriate threshold for this split, a number of random candidates is generated. The split is performed with each of the candidates, and the quality of each split is assessed using a score formula based on the resulting sets' entropy:

$$S = -\frac{|l|}{|n|}E(l) - \frac{|r|}{|n|}E(r)$$

where n is the input set of Data Points and l and r are the left and right subsets generated by the split, and

$$E(n) = -\sum_{c} p(c)\log_2 p(c)$$

is a labeled set's entropy.

The use of this scoring formula aims to distribute the counts of Labels as equally as possible across the two resulting sets. The candidate which maximizes the score value is selected as final threshold for the current Node. Afterward, the Node training function is called recursively for two new Nodes, the left and right child of the current Node, using the left and right subsets as training input. The process is repeated for each child. Each Node created this way is designated as Inner Node. The recursion stops when either the maximum depth (distance from the root) is reached, or the score value of the best threshold is too close to zero, which means that the input data set is either empty or contains only elements with the same Label, and no more information gain is possible by splitting. These Nodes are called Leaves. The training of a Tree is finished when each of its branches ends in a Leaf. The Forest is trained when each of its Trees is trained. An example for a Forest can be seen in Figure 1.

Concerning the selection of appropriate Feature types, either every Node may use the same Feature, or each Node selects one at random. It is noteworthy that the “difference" Features are less sensitive to global illumination change than the others, which may prove advantageous for certain problems. However, letting the Forest work with arbitrary features allows it to learn the ones that provide the most information gain.

2.2 Classification

Due to the simplicity of Data Point Features, a trained Forest can be evaluated very quickly to create a very powerful and expressive feature representation of an image. This feature vector is called the Textonization of an image, and it consists of a histogram counting the occurrences of all of the Forest's Nodes. First, a set of Data Points is extracted from the image. This set is presented to each Tree of the Forest in turn. Starting from the root Node, a Data Point has its Feature value calculated, compared against the Node's threshold and passed on to the left or right child Node. This is done repeatedly until a Leaf is reached. Every time a Data Point arrives at a Node, the Node's associated histogram bin is incremented. The Textonization is complete after all Data Points have passed through all Trees. For purpose of increased access speed, each histogram bin stores not only the associated Node's global index, but also the index of its Tree and the depth level it is in.

To predict the Label of a texonized image, an appropriate nonlinear kernel function for a Support Vector Machine (SVM) has been implemented. The SVM Classifier is considered trained after being initialized with such a kernel. The kernel contains a value for every pair of training data elements representing a similarity measure between them.

Over the course of the internship, the popular open source library *libsvm* [3] has been used as implementation of the Support Vector Machine model. Libsvm includes many algorithmic and performance related improvements as well as support for pre-computed kernel functions, making it ideal for use as Classifier. However, since the library is written in C++ and there is a lack of publicly available well-optimized C# wrappers, a new C# wrapper library for libsvm, called *libsvm.clr* [4], was developed. Libsvm.clr is designed for having minimal impact on the comparatively fast performance of libsvm, while staying true to the original interfaces and usage patterns and utilizing the state-of-the-art programming paradigms of C# 6.

The kernel function is calculated as follows. Its values are stored in a matrix that has one row and one column for element of the training set. A matrix entry at the position (p,q) represents the similarity measure between the elements p and q. Given two Textonizations P and Q of two training elements, the similarity measure K(P,Q) is calculated by a modified Pyramid Match [2] metric:

$$K(P,Q) = \frac{1}{T}\sum_{t=0}^{T}\frac{\tilde{K}(P,Q)}{Z}$$

$$Z = \sqrt{\tilde{K}(P,P) * \tilde{K}(Q,Q)}$$

with

$$\tilde{K}(P,Q) = \sum_{d=0}^{D}\frac{1}{2^{D-d+1}} * (I_d - I_{d+1})$$

where I_d is the summed up histogram intersection at the depth *d*

$$I_d = \sum_{j \in d} \min(\ \mathrm{P[j]}, \mathrm{Q[j]}\ )$$

The histogram intersection used in I_d operates on the parts of the histograms at depth *d*, where *j* is the running variable for all Bins that have Depth *d*. It represents the amount of points that can be matched across both histograms at that depth. In terms of the Forest, it is the number of Data Points from the respective images P and Q that have reached the Leaves or passed through the Nodes at that depth. The higher this number, the more similar the two images are at that depth. The expression $I_d - I_{d+1}$ subtracts from the match count of one depth the count of the depth below. This means that at the current depth, only the matches are considered which are "new" at this depth and have not been already matched anywhere below the current one. The expression is divided by a scaling factor that has its largest value at the deepest level (the most "difficult" match) and progressively becomes smaller at higher levels closer to the root (where the matching becomes “easier"). This difference is summed up over all depths. The root is at depth d = 0 and I_{D+1} = 0. The resulting value is large for image pairs which are similar in the sense of the Forest, and small for dissimilar images. Through division by Z the value is normalized for images of different sizes and scaled to the range [0,1]. The average of all Trees' individual results is the final value for K(P,Q).

The training kernel is obtained by evaluating K for all pairs of elements in the training set. The resulting kernel matrix is symmetric (K(P,Q) = K(Q,P)) and has ones in the diagonal (K(P,P) = 1). The SVM Classifier is considered trained after being initialized with this kernel matrix together with the appropriate Labels and indices (for the precise memory layout, please refer to the libsvm manual).

To predict the Label of an image or a set of images ("test" images), a second kernel is presented to the Classifier. For each of these test images, the similarity value K is established to all of the training images used for training in the previous step and stored in the test kernel matrix. The SVM Classifier compares this matrix with the trained model and evaluates the membership of the test data to the class Labels. The resulting output is the Label that most likely belongs to the input image.

2.3 Segmentation

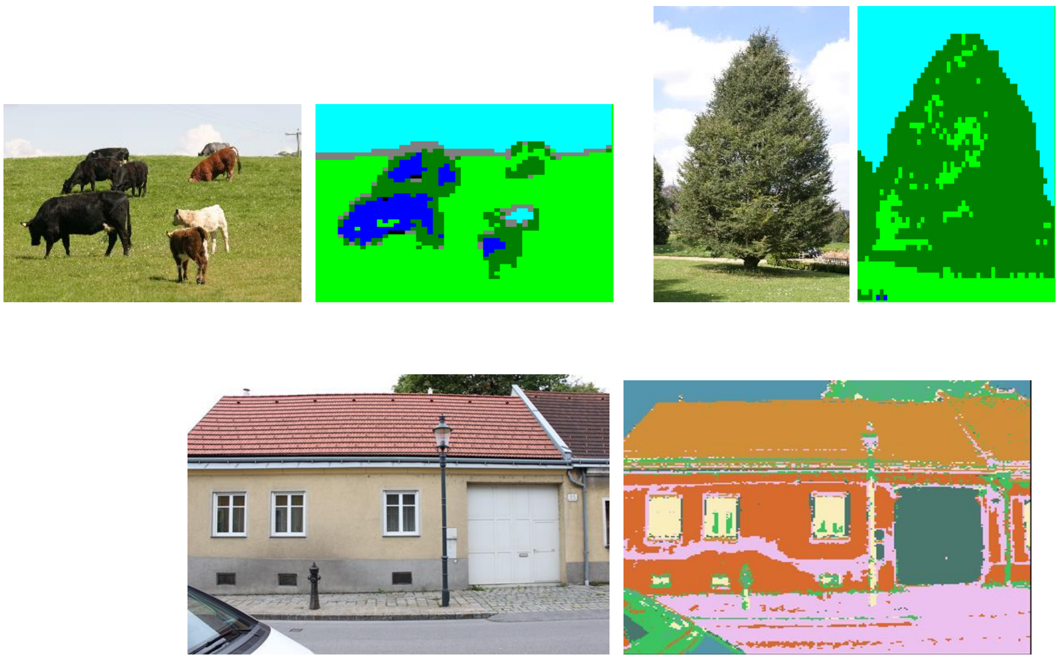

Figure 2: Some examples for segmentation results. (Colors in the visualizations are not consistent.)

In principle, image Segmentation is performed by feeding image Patches, which are small rectangular subregions of an image, into the Classification system defined in the previous Section. This Segmentation can and should be done independently of the image-level Classification, since it generally requires a Forest trained with different Parameters and a different data set. Commonly, the Forest used for Patches will be flatter and have more trees than the one used for Images. Also, it is worth repeating that the training Patches, which are textural representations of their objects' appearances (trees, grass, sky, …), may include rotated copies of themselves where appropriate to improve accuracy. Additionally, the Patches must have a Label for each pixel they cover. Should it not be uniformly the same Label for all pixels, the Forest training step is refined by selecting the actual Label for the currently considered Data Point to represent that part of a Patch. When using photographs with ground truth maps, the size of the Data Point sampling window can be increased to allow the feature representation to take the semantics, that is the context of different objects around the current one, into consideration where appropriate.

Even more so than with entire images, the number of Patches can quickly become very high within a Segmentation workflow, which causes the kernel computation to take a very long time. Some functionality is provided in order to combat this problem. The number of Patches per Tree can be limited as well as the number of Data Points per training step using a parameter as hard constraint. These parameters also improve generalizability of a model. Also, it is a good idea to select only a small subset of the training data and coarsely test out appropriate parameters using the testing framework before committing to a training run with the full data set. Furthermore, some helper functions are included to facilitate working with Patches. Since the ground truth Labels for Patches are often encoded in the color channels of a Label map, functions are included to map a color onto a Label and vice versa. Finally, using these mapping functions, a method is included to create a visualization of a Segmentation output by generating a picture the same size as the input image and coloring the patches according to their Labels. Some examples for Segmentation results can be seen in Figure 2.

An extension to this segmentation method [1], which uses a "second-level" Segmentation Forest that operates on the output of the original Forest in order to perform segmentation, has been implemented. Using this second Forest instead of the Classifier decouples the semantic context consideration from the original Classification workflow by admitting the use of a different set of parameters in an extra step, while also drastically reducing the computation time by not having to calculate a kernel. However, the quality of the segmentation heavily depends on the quality of the training set and on the parameters, increasing the degrees of freedom in the system.

The Segmentation Forest is trained on a set of Distribution Images created by the Texton Forest (Section 2.1). A Distribution Image is map of Label Distributions predicted by the Texton Forest for each individual pixel of an image. The Distribution Image has the same width and height as the image, but the number of "color" channels equal to the number of

Labels. It is generated by presenting each pixel (faster: regular sampling) to the Texton Forest and following its path to a Leaf. The pixel's Label distribution is the distribution of Labels which arrived at that Leaf during training. Each Label in this distribution is weighted by the inverse of its frequency in the training set to remove the underlying bias towards the more frequent Labels.

The Segmentation Forest is trained in a fashion similar to the Texton Forest, and its resulting structure is also mostly similar. However, the Data Points are extracted from Distribution Images instead of actual photographs and are chosen as a regular grid spanning the entire image. A Node's split function differs in that it has an offset value which may be very large (up to half the image size). By considering regions far away from the actual patch, the semantic information (the "surrounding") is incorporated into the model. The split function randomly selects one of the "color" channels in the Distribution Image and compares the value to its threshold directly. In case of a pixel region, the average of all pixels is computed. This computation uses the efficient matrix algebra of Aardvark and implements Summed Area Tables in order to maximize performance.

After training is complete, the Label of an image can be predicted by transforming the image into a Distribution Image and presenting the Distribution Image to the Forest. The Forest splits it into patches similar as during training and returns a new Distribution Image which contains the prediction of Labels for each pixel. The resulting segmentation can be visualized using the same helper function as in the previously described in the Classifier Segmentation approach. Some examples can be seen in Figure 2.

3. Usage

The following C# code includes an example application using Semantic Texton Forests.

```
string workingDirectory = @"C:\Temp";

// Parameters:
var parameters = new TrainingParams(6,   // Number of Trees
                 18,                      // Maximum depth of Trees
                 15,                      // Sampling window size
                 5,                       // Sampling Frequency
                 Labels);                 // Array of all Labels

// (0) Read and Prepare Data

LabeledImage[] images = HelperFunctions.GetLabeledImages(Path);

LabeledImage[] train;
LabeledImage[] test;

images.Split(out train, out test);

// (1) Train Forest

var forest = new Forest(ForestName, parameters);
```

```
forest.Train(train, parameters);

// (2) Textonize Data

var trainTextons = train.Textonize(forest, parameters);
var testTextons = test.Textonize(forest, parameters);

// (3) Train Classifier

var classifier = new Classifier(workingDirectory);

classifier.Train(trainTextons, parameters);

// (4) Classify

Label[] prediction = classifier.PredictLabel(testTextons, parameters);
```

The `parameters` object specifies many aspects of the Classifier system. By initializing it as shown here, sensible default values are chosen. First, the image set is read from disk. `HelperFunctions` contains some example functions for this task which can be copied and modified. Afterwards, the Forest is trained using a part of the image set. Then, both parts of the image set are transformed into their respective Textonizations. Finally, the classifier is trained using those Textonizations. The Labels of texonized images can be predicted using that classifier.

The syntax for Classifier Segmentation looks identical, except for using `LabeledPatch` instead of `LabeledImage`. Similarly, the syntax for the Segmentation Forest looks the same, but uses

```
var trainDists = train.Distributionize(forest, parameters);

var segForest = new SegmentationForest(ForestName, segParameters);

segForest.Train(trainDists, segParameters);
```

to compute Distribution Images and perform training.

The Classifier has been tested using two datasets. The first one was the MSRC Dataset (which can be downloaded at [6]), which contains outdoor photographs with 21 Labels and about 30 images each. This dataset poses a difficult problem for classification system because the images are varied in quality and lighting, and there aren't particularly many samples available per Label. Still, during testing, the Semantic Texton Forest classifier was able to realistically achieve upwards of 60% precision. Figure 3 shows an example run, in which the precision varies with different numbers of Trees. In this example, the best overall accuracy was achieved at 5 Trees, more than that lead to increased overfitting. A second second internal data set was also used for testing, which had two Labels and about 1000 examples each. The Classifier reached up to 80% precision.

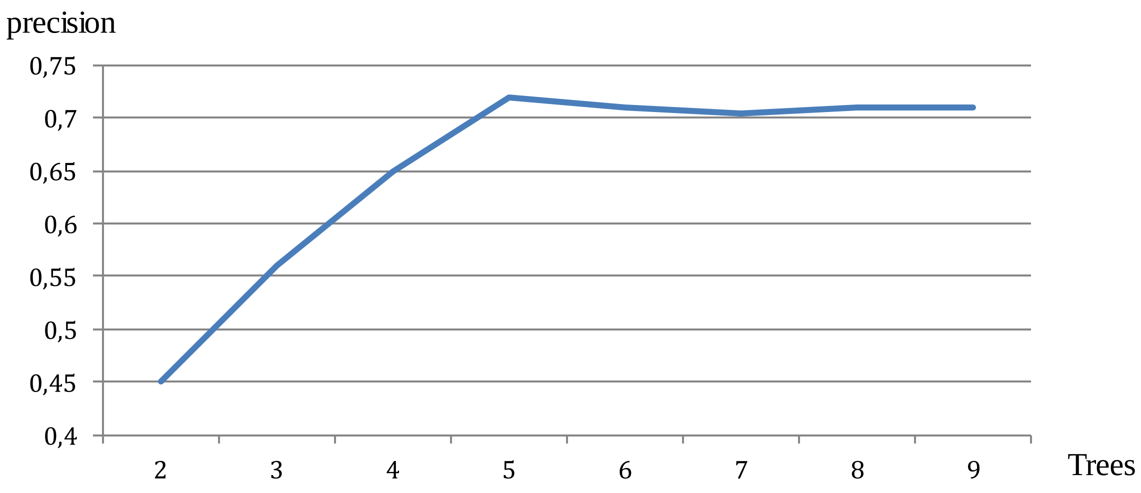

Figure 3: Precision versus the number of Trees.

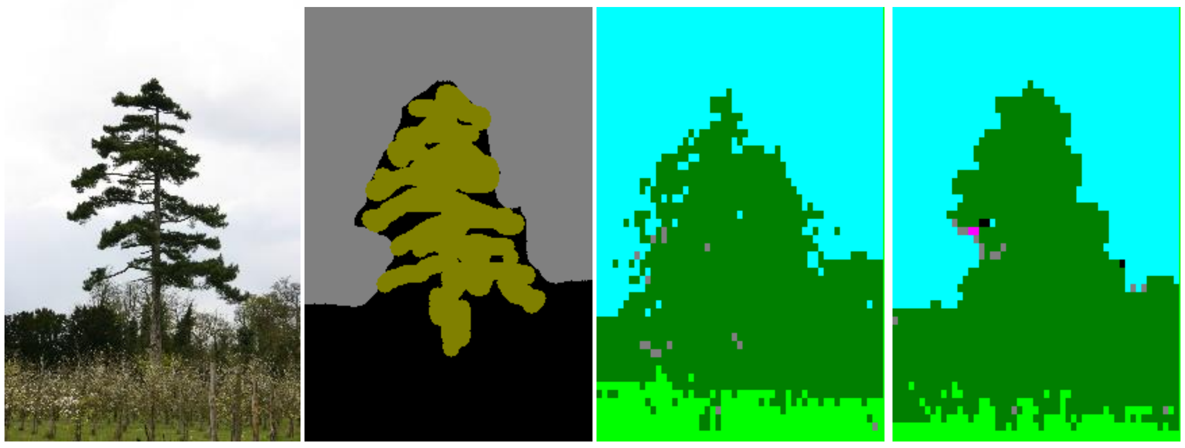

Figure 4: From left to right: Image, ground truth, Classifier segmentation, Segmentation Forest

Figure 4 shows some examples for Segmentation, and in particular, the difference between segmentation using the Classifier, which only considers local features, and the Segmentation Forest, which incorporates semantic information. In general, the local segmentation leads to a higher precision per patch, but doesn't preserve the shape of segmented objects. Conversely, the Segmentation Forest leads to smooth shapes and rounded outlines, but is more likely introduce artifacts due to the higher degree of freedom in the model and potentially bad training parameters.

3.1 Performance

All of our tests were performed on a relatively modern machine with an Intel Core i5 CPU and 16GB of memory. Still, it can be observed that all procedures relying on a Forest are very quick to evaluate, and, depending on the training parameters, do not take very long to train either. A data set with 50 pictures at 400 x 300 pixels each and regular sampling with a square sampling window 5 pixels in size produces around one million data points. Most of the test configurations we used resulted in around this number, and it was also a sensible hard limit to use for training with larger data sets. Training both a Texton Forest and the Segmentation Forest with 8 Trees and a maximum depth of 20 takes about ten minutes with a training set this large.

These parameters are what we found to be acceptable for most of our problems, anything (much) higher than that took (significantly) longer to train, but did not increase the output quality by any substantial amount. The resulting Forests contain about 5000 to 10 000 nodes each and the processing speed is roughly equivalent to 4000 data points per second. In comparison, for the same set-up, a Classifier approach is slower by about a factor of ten. Calculation of the Support Vector Machine kernel took up to thirty minutes and longer. Even worse, the size of the kernel and number of calculations grow exponentially with a larger training set, which usually makes the Forest-based approaches preferable for large problems.

Evaluation of the Forest only takes about one second per picture, which corresponds to about 100 000 data points processed per second. The actual performance bottleneck in our system lies in the loading and processing of images, and the reading of pixel data and calculating pixel features. These operations take several times as long as the actual Forest operations. Our system uses the naive approach of performing these actions on demand, but a future refactoring could be imagined in which all required pixel values and features are calculated as a pre-processing step and only need to be read from memory during segmentation.

Our processing times are roughly half of the numbers reported in the original paper [1]. This stems largely from the fact that we use parallelization and a quad-core CPU to effectively multiply our performance, while the original implementation is single-threaded. We also use a thread-safe caching system for faster access to image data, and our fast libsvm wrapper library. However, the original implementation includes several algorithmic improvements which we do not have. Therefore, speaking comparatively, our implementation can not quite reach the performance of the original.

4. Conclusion and Future Outlook

Semantic Texton Forests provide a tool for image classification and segmentation. They distinguish themselves by being extremely fast to evaluate and the ability to be tuned to incorporate semantic information about data. On the other hand, they may suffer from being badly configured and there is no easy method of finding proper parameters, apart from trial and error.

Semantic Texton Forests were implemented in the course of the internship. The implementation aims for openness, speed and ease of use and includes many parameters to configure the system. Also included are tools to facilitate the image classification and segmentation workflows.

Conversely, the system has not been optimized for memory efficiency. There are some algorithmic changes and re-arrangements to be done in order to reduce the memory overhead and fragmentation. Implementing these was outside of the internship's scope. In addition, the system lacks procedures to find optimal parameters. Some concepts to exhaustively search for, or heuristics to quickly find such parameters could be imagined.

5. References

[1] Shotton, Jamie, Matthew Johnson, and Roberto Cipolla. "Semantic texton forests for image categorization and segmentation." *Computer vision and pattern recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE, 2008.

[2] Grauman, Kristen, and Trevor Darrell. "The pyramid match kernel: Discriminative classification with sets of image features." *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*. Vol. 2. IEEE, 2005.

[3] Chang, Chih-Chung, and Chih-Jen Lin. "LIBSVM: A library for support vector machines." *ACM Transactions on Intelligent Systems and Technology (TIST)*2.3 (2011): 27.

[4] *https://www.nuget.org/packages/libsvm.clr.vs2015.x64/ (accessed: 09-06-2015)*

[5] *https://github.com/vrvis/aardvark.semantictextonforests (accessed: 09-06-2015)*

[6] http://research.microsoft.com/en-us/um/people/antcrim/data_objrec/msrc_objcategimagedatabase_v2.zip
. *(accessed: 09-06-2015)*