Introduction

This report documents my 6 months internship as a research student at Kyushu University. I worked in the Sakamoto laboratory under the supervision of eponymous professor Sakamoto, part of the faculty of design on Ohashi campus in Fukuoka, Japan.

The subject of my research was the identification of animation character from color images, an image classification problem in which we consider images of characters from Japanese animation. The automated analysis of animation images is not a well studied field, although there are a few research papers in Japanese. The goal was therefore to improve on this state of the art, with the envisioned application domain being the automated annotation of images in web artist communities such as Pixiv[1] or deviantArt[2]. One of the expected milestones was the presentation of our work at the JCEEE Kyushu International Sessions in Kumamoto at the end of September.

Our solution to the animation character identification problem takes the shape of a computer vision system with preprocessing, segmentation and classification of images. A number of methods were considered for each of these steps, with a strong focus on classification. In this document, we present each of the algorithms considered, their underlying theoretical foundations, as well as their implementation in practice and we analyse their performance against a dataset of animation images.

Our methods for classification start by considering a graph structure from the set of segments obtained from the previous segmentation step. We first studied graph kernels inspired from works in string matching for the purposes of classifying the graphs as developed by Harchaoui and Bach [1]. Failing to capture the global features of the graph in an efficient manner, we then turned to methods from spectral graph theory to classify our segmentation graphs, borrowing from works by Wilson [1] [2] and inspired by developments in spectral methods for clustering [3], segmentation [4] and dimensionality reduction [5] [6]. Graph Laplacians did not capture enough information about the images, so we finally turned to a segment matching method based on a fuzzy control system and a simple algorithm to determine one-to-one relationships between segments of 2 images. This method is computationally efficient, and gives good results on our dataset, measuring a recognition rate of 59% using leave one out cross validation.

In this document, we also consider future extensions to solve the shortcomings of our method. Notably, our method fails to properly distinguish characters sharing similar color schemes. To solve this issue, we propose a semi-supervised embedding method to determine a non-linear color space ideally clustering the training data based on literature for non-linear dimensionality reduction [5] [6] and classification [7]. Furthermore we suggest that, although failing to properly characterize the images on their own, segmentation graphs could help distinguish these characters sharing similar color palettes. We also introduce some possible research directions for the purpose of background extraction.

[1] `www.pixiv.net` is a popular Japanese web artist community.
[2] `www.deviantart.com` is a popular English speaking web artist community.

Thanks

From Kyushu University, I would like to thank teacher Hiroyasu Sakamoto for his excellent supervision throughout my stay in the laboratory, and his help both in and outside laboratory work. I would also like to thank all the laboratory members for their help, bouncing ideas off of each other, and for making the effort of speaking English with me. Many thanks to the people at the support center and in particular Chitose Oka for her help with accommodations and the English school.

From UTBM, I would like to thank Japanese language teacher Keiko Jimbo, for being a great teacher and a tremendous help finding this internship in the first place. Yassine Ruichek and Cindy Capelle for teaching me nearly everything I know about image processing and machine learning. Mireille Jacquot for being very patient and understanding both during my internship search and during my stay in Japan.

In France, I would like to thank my family for its support, and trusting me in my choices and decisions. The French government and the province of Franche-Comté for their financial help. Special thanks goes to the group whose name changes every month for its support during difficult times.

In Japan, I would like to thank the very many friends I met there, and made my stay there an incredible experience. Special thanks go to Jelle Postma, Ketty Chung and Giorgia Sassi for helping provide plenty of Japanese animation material. Yuki Nakagawa, for being a great supporter and being there whenever I needed his help. Patxi Garcia Novo, for his help with the scholarship application and allowing me to crash on his couch for 2 weeks. Everyone from Ijiri dorm and all the Japanese friends I haven't mentioned, you guys are awesome and I enjoyed hanging out with you all.

Contents

Chapter 1

Presentation of Kyushu University

1.1 Kyushu University

Figure 1.1: Map of Kyushu University's campuses in Fukuoka. The 6^{th} campus is the Beppu campus, not included on this map. Map from [8] .

Kyushu University is a public university in Fukuoka, on the island of Kyushu in western Japan. It was founded in 1911 as the Kyushu Imperial University, and is currently the largest public university in Kyushu. It employs 2186 faculty members and has 18765 students, including more than 1700 international students [9]. Kyushu University is one of the best research universities

in Japan according to Thomson Reuters, and especially shines in Materials Science, Chemistry, Biology and Biochemistry, Immunology as well as Pharmacology and Toxicology [10]. It counts 12 undergraduate schools, 19 graduate schools, 16 graduate faculties and 8 research institutes [11], spread out on 6 campuses (Figure 1.1).

1.2 Faculty of design and the department of visual communication design

Figure 1.2: Ohashi Campus, home of the faculty of design. Photograph courtesy of the author.

Kyushu University's faculty of design is located in Ohashi campus alongside the graduate school of design and the school of design. It includes various departments teaching and researching environmental design, visual communication design, art and information, industrial design, acoustic design, and others. The department of visual communication design includes the section of Image Engineering, which includes computer vision and machine learning [12].

1.3 Sakamoto laboratory

The Sakamoto laboratory, part of the department of visual communication design, is supervised by eponymous teacher Hiroyasu Sakamoto. It specializes in computer vision, virtual reality and machine learning. It is constituted at the time of writing by 8 graduate students and 3 undergraduate students, including

4 foreign students. Domains currently researched in the laboratory include animation character image analysis, 3D mesh segmentation, face recognition and head-mounted devices (google glass like devices).

Chapter 2

A method for identifying animation characters from color images

2.1 Subject

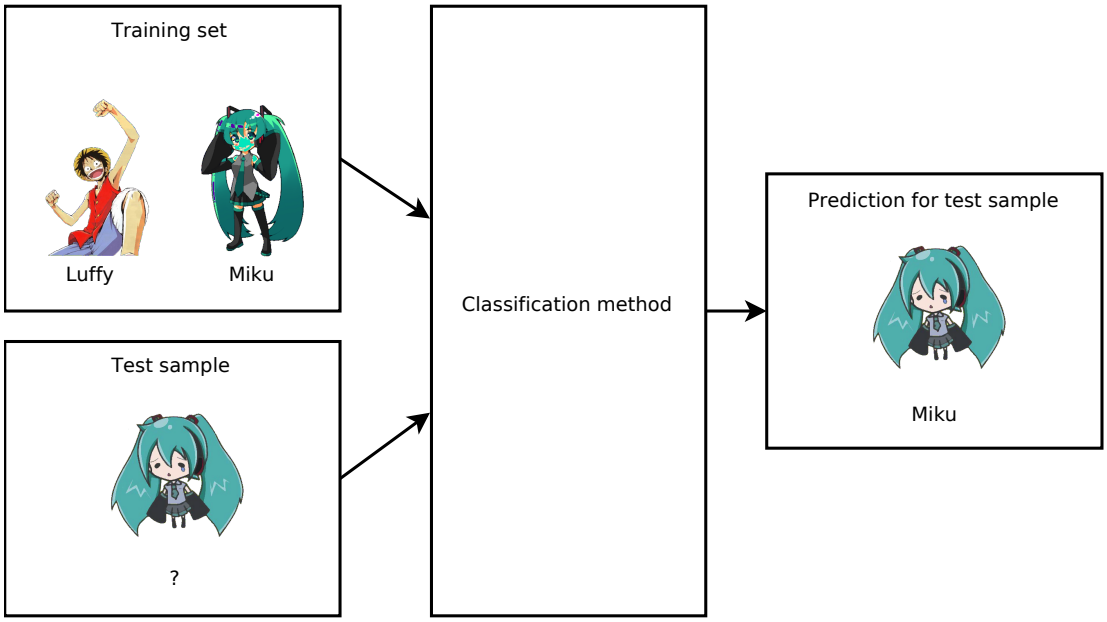


Figure 2.1: Diagram depicting how training and prediction work for the method.

The subject of my work here was to design a computer algorithm able to automatically identify characters from color images. It should take the shape of a supervised or semi-supervised classification method, meaning it should be able to identify an image by comparing it against a training set. Said train-

ing set may consist of labeled images - images which are known to depict a certain character - as well as unlabeled images in the case of semi-supervised classification (Figure 2.1).

2.2 Position in the laboratory

I worked in the Sakamoto laboratory as a research student.

2.3 Previous studies

Previous work in the laboratory by graduate student Yuki Nakagawa on the animation character identification problem identified a segmentation method[13] and a classification method[1] for the problem. A dataset of animation character images was also previously constituted in the laboratory.

There is little previous literature on animation character image analysis, and all of it is in Japanese. There has been work on identifying artist information from images in [14], whose abstract has been translated to me by professor Sakamoto.

2.4 Objectives

Figure 2.2: Images for one character illustrating differences in posture, exaggerations, occlusion and artist differences.

The first objective was to design a method which would improve on the state of the art in animation character identification, in terms of recognition rate, clustering quality or run time performance. The envisioned application domain being automatic image annotation for web artist communities such as Pixiv or deviantArt, it should also be able to handle large datasets.

A secondary objective was to improve on the state of the art in the fields of computer vision and machine learning through the challenges specific to animation characters. There is for instance a large literature on human face recognition [15] [16], but such methods fail to account for meaningful color information

in animation images as well as variations introduced by character posture, exaggerations, occlusion of body parts and differences between artists (Figure 2.2). Many classification and clustering methods also require data to form convex clusters (for instance k-means clustering), which is rarely the case with animation images because of previously mentioned variations, so recent research involving non-linear embeddings of data were of particular interest [5] [6] .

2.5 Planning

No provisional planning was established upon my arrival. The only milestones were:

- Deadline for paper submission to the JCEEE Kyushu 2013 conference [17] at the end of August.
- Paper presentation at the the JCEEE Kyushu 2013 international sessions on the 24^{th} of September in Kumamoto.

2.6 Design of the method

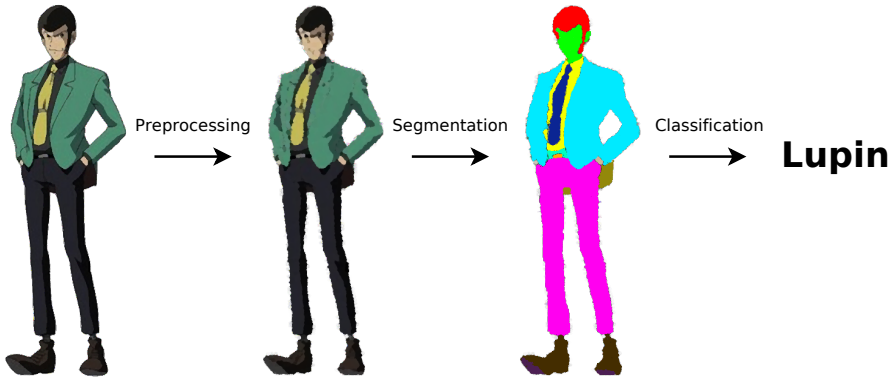


Figure 2.3: Diagram depicting how preprocessing, segmentation and classification interact.

The method takes the shape of a computer vision system with separate phases of preprocessing, segmentation and classification.

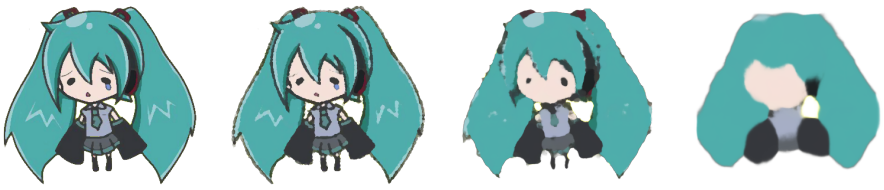
- The preprocessing step should operate transformations on the original image so segmentation can be applied efficiently and accurately. This includes filtering, color space transformations, histogram manipulations and geometric transformations. This results into a new preprocessed image.

- The segmentation method aims to automatically isolate parts of the character image as faithfully to human perception as possible. An ideal segmentation would for instance segment separately hair, face, clothes, hands and feet as distinct components. The end result of the segmentation is a partition of the set of pixels in the image.
- The goal of classification is to extract features of interest from the segmentation, and use these features to predict the character identity from the training information. The end result is the predicted name of the character.

Although the global structure of the method described above did not change during design, many techniques were experimented with for each of the steps. In this section each technique will be described. See section 3.1 in the appendices for the mathematical conventions and definitions used in this document, which although not departing from standard rely heavily on vocabulary from spectral graph theory, image processing and fuzzy logic.

2.6.1 Preprocessing

Kuwahara filter



(a) Before filtering. (b) Small window. (c) "good" window. (d) Large window.

Figure 2.4: Results of Kuwahara filtering with varying window size. The small window causes outlines to be accentuated, and the large window removed the tie, ribbon and legs.

One of the early issues encountered when attempting to segment animation character images was the presence of black outlines, which usually get segmented into many large components, but do not contain any information relevant to identification. We identified the Kuwahara filter [18] as a way to not only remove the black outlines, but also remove other unnecessary details and make areas of color more homogeneous and amenable to segmentation.

The method Consider a grayscale image I (we'll see later how it generalizes to color images). For each point (x, y) on I, we consider a square window of size $2h + 1$ centered on it. We then consider four overlapping square regions in this window as shown in Figure 2.5. For each region Q_i for $i \in \{1, ..., 4\}$ we compute the arithmetic mean $m_i(x, y)$ and standard deviation $\sigma_i(x, y)$ of pixel values inside Q_i. The pixel value at (x, y) in filtered image J is then defined as the arithmetic mean $m_i(x, y)$ corresponding to the smallest standard deviation $\sigma_i(x, y) = \min_{j\in\{1,...,4\}} \sigma_j(x, y)$. When dealing with borders of the

image, we simply do not consider the missing pixels while computing the means and variances.

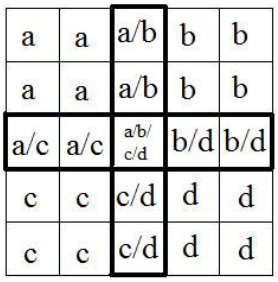


Figure 2.5: Window used by the Kuwahara filter with half-size $h = 2$ and square regions a, b, c and d.

Generalizing to color images, we still consider the arithmetic means of pixel colors, but this time we consider the brightness of the pixels (e.g. the V in HSV color space) for standard deviation.

Theoretical justification The algorithm was originally designed to improve segmentation in medical image processing. For the case of removing outlines, if one considers a window size "large enough" that outlines never fills most of it, then it will introduce a large standard deviation in the square region it is contained in - meaning it will be removed in the filtered image. It should be noted that if the window is not large enough, then outlines can actually be thickened by the Kuwahara filter, and if it is too large then relevant information may be lost (Figure 2.4). We found however that it is easy to empirically determine a window size in practice.

Implementation A naive implementation of the filter recomputes all means and variances for each pixel, runs in $O(nh^2)$ time where n is the number of pixels in the image, and h is the window half size.

Color space selection

As the goal is to obtain a segmentation as close to human perception as possible, the $L^*a^*b^*$ color space has been found to be most suitable. The HSV or HSL color spaces are also useful to extract hue information, (the H in either color space) which we use for histogram equalization (section 2.6.1) and as post-processing to segmentation (section 2.6.2).

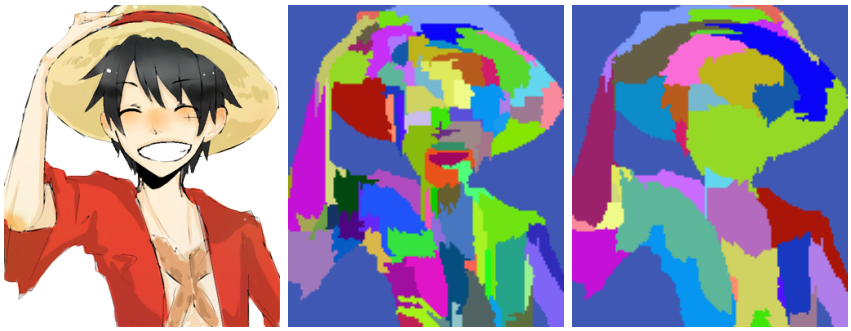
Although using $L^*a^*b^*$ color space gives good segmentations, it may not be ideal for classification as 2 different characters may share predominantly the same color palette. A possible extension of this would be to use training data to determine an abstract and possibly non-linear color space where members of the same class are close and members of different classes are far away, using - for instance - a (semi) supervised embedding method similar to the one used in [7]. See section 2.7 for more details.

Color histogram equalization by hue

Some characters are represented predominantly by a few colors, which makes segmentation more difficult. Equalizing the histogram by hue allows segmentation to distinguish between more closely related colors. However, irrelevant differences in color space are also accentuated, so histogram equalization was eventually removed from preprocessing as it had an overall negative impact on segmentation performance.

2.6.2 Segmentation

Felzenszwalb's method



(a) Before preprocessing. (b) With $k = 100$. (c) With $k = 1000$.

Figure 2.6: Segmentation results with Felzenszwalb's method, with a small k and a larger k.

The segmentation method by P. F. Felzenszwalb and D. P. Huttenlocher, described in detail in [13], is an efficient graph-based segmentation method for color images.

The method Let I be a color image. The algorithm considers a graph $G = (V, E)$ defined on the set of pixels of I with a weight function w measuring dissimilarity between pixels. Let n be the number of vertices and m the number of edges of G. The pseudo code of the algorithm is given in algorithm 1, where $MInt : S \times S \to \mathbb{R}^+$ for some segmentation S is defined by:

$$MInt(C_1, C_2) = \min(Int(C_1) + \tau(C_1), Int(C_2) + \tau(C_2))$$

$\tau(C) = \frac{k}{|C|}$ is a threshold function for a scale parameter k, and $Int(C)$ measure the internal difference of a component C, defined by the largest weight of the minimum spanning tree of C:

$$Int(C) = \max_{\{u,v\} \in MST(C,E)} w(u, v)$$

Note that in our case, we consider a slightly different threshold function to allow segments of varying size provided they are homogeneous enough by using graph-theoretic volume instead of cardinality: $\tau(C) = \frac{k}{vol(C)}$ where $vol(C) = \sum_{v \in C} d_v$.

Algorithm 1 Felzenszwalb's segmentation algorithm

```
1: function FELZENSZWALB(G = (V, E))
2:     Sort E into π = (o_1, ..., o_m) by ascending edge weights
3:     Initialize S_0 with each vertex in its own component
4:     for q ← 1 to m do
5:         Let {u, v} = o_q the q^th edge in the ordering
6:         Let C_u and C_v the components containing u and v in S_{q-1}
7:         if C_u ≠ C_v and w(u, v) ≤ MInt(C_u, C_v) then
8:             S_q is obtained by merging C_u and C_v in S_{q-1}
9:         else
10:            S_q is just S_{q-1}
       return S_m
```

Theoretical justification The authors show that this algorithm results in a segmentation which is neither too coarse nor too fine, according to a precise notion of boundary between segments. This is interesting in the case of animation character images, which are characterized by large areas of homogeneous color, which this algorithm captures reasonably well.

Implementation The algorithm can be implemented to run very efficiently. By using a disjoint set forest data structure for the segmentation, determining the component of a vertex and merging components to be performed in amortized $O(\alpha(n))$ time where α is the inverse of the Ackermann function [1] [19]. Computing $MInt$ between each iteration can be done in constant $O(\alpha(n))$ time - the largest weight of the minimum spanning tree is the one we used for the merge, and one can keep the cardinality of segments in the disjoint set data structure to find it with a single find operation. With an adjacency list data structure for the graph G, it is easy to list all edges in $O(m)$ time. This makes clear that the time behavior of the algorithm is dominated by the initial sorting of edges, which can be computed in $O(m \log(m))$ time using an optimal comparison sort like merge sort.

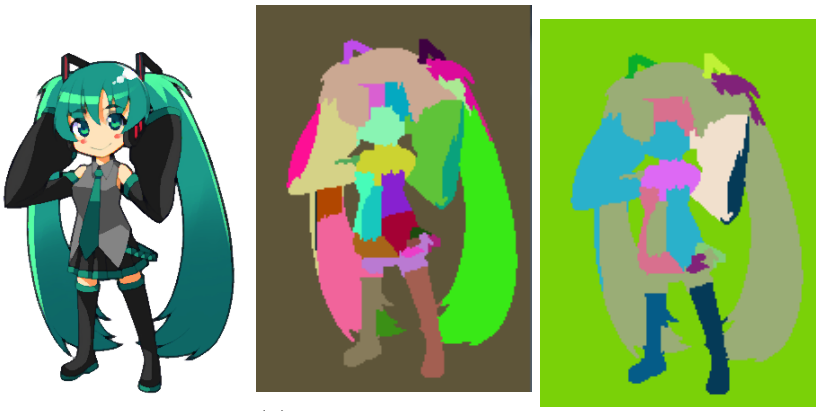
The authors study the case where G is an 8-connected graph and where it is a K-nearest-neighbor graph in (x, y, r, g, b) feature space. Since both classes of graphs are sparse - the number of edges is within a constant factor of the number of vertices - this makes the time complexity of the algorithm $O(n \log(n))$.

Results and analysis Although this algorithm is efficient, it suffers from a few drawbacks. The method as described above tends to create many very small components, which can be solved by a post-processing step greedily merging components smaller than a threshold. It also tends to produce segments of similar size, which in the case of animation character image is not desirable. It produces either an oversegmentation where large segments are separated in many small ones (Figure 2.6b), or an undersegmentation where smaller segments are incorrectly merged into larger ones (Figure 2.6c). To solve this issue,

[1] The Ackermann function is a very fast growing function. Its inverse is therefore extremely slow growing, so it can be considered equivalent to a small constant for all practical purposes. See `http://en.wikipedia.org/wiki/Ackermann_function` for an introduction.

we introduced a post-processing step which we present in the next section (section 2.6.2).

Hue-based segment merging



(a) Original image. (b) Result of Felzenszwalb's method. (c) After merging.

Figure 2.7: Example of merging by hue. In the initial segmentation, the hair is separated in 3 large segments, which are fused into one after merging - even though the arms separate these 3 segments.

Felzenszwalb's method, using a small enough scale parameter, tends to produce over segmentations of the image (relevant areas in the image are divided in multiple segments), with segments of similar area. Furthermore, using a 4-connected graph, it cannot produce spatially disconnected segments, although such cases arise due to changes in posture or occlusion by other objects. To solve this issue, we introduced a post-processing step merging segments whose average hue are "close enough" in some sense, to produce a suitable segmentation with varying segment sizes and disconnected segments.

The method We consider a segmentation S for a color image I, our algorithm outputs a new segmentation for I. It proceeds by considering the complete graph on the set of segments of the image, with edges weighted by absolute hue difference. We then consider a small fraction of these edges in ascending order of weight, and fuse the corresponding segments. The pseudo code of the algorithm is given in algorithm 2.

Theoretical justification This is mostly a heuristic method which relies on greedy assumptions for efficiency. For instance we assume that the difference in average hue between the merged segment is small enough that the average hue of the new segment does not need recomputing. As we merge them from smallest hue difference to largest, this assumption holds to some extent. Although computing the new hue can be done efficiently by associativity of the barycenter, it would require resorting the data structure for E to account for the changes - which propagate inside the graph in a non-trivial way.

Algorithm 2 Hue-base segment merging algorithm

function HUEMERGING(I, $S = \{S_1, ..., S_q\}$, $k \in \mathbb{N} - \{0\}$)
    $E \leftarrow \emptyset$
    for each unordered pair $\{S_i, S_j\}$ in S **do**
        Let h_i and h_j be the average hues of S_i and S_j in I respectively.
        $E \leftarrow E \cup \{(i, j, |h_i - h_j|)\}$
    for $i \leftarrow 1$ to $\lfloor \frac{q}{k} \rfloor$ **do**
        Remove edge (i, j, w) with smallest weight w from E.
        Merge segments containing S_i and S_j in S.
    return S

Implementation Computing the average hue for each segment can be done in $O(n\alpha(n))$ steps using a disjoint set forest data structure for S where α is the inverse of the Ackermann function and n is the number of pixels in the image - assuming the hue of each pixels was precomputed as part of a preprocessing step. Using a binary heap data structure for E makes computing the edge weights run in $O(q^2)$ time, and the second loop then runs in $O(\lfloor \frac{q}{k} \rfloor \log(q))$. As there is, to the author's knowledge, no direct relationship between q and n (besides $q \leq n$) allowing us to decide which of the first 2 steps dominates the other, we will leave the overall time complexity as $O(n\alpha(n) + q^2)$.

Results and analysis As a post-processing step to Felzenszwalb's method, this algorithm solves the issues mentioned earlier - segments of varying size, connecting areas which are spatially disconnected. This yields segmentations which correspond better to human perception, and are overall more consistent across images of the same character, which is essential to classification.

2.6.3 Classification

Tree-walk kernel

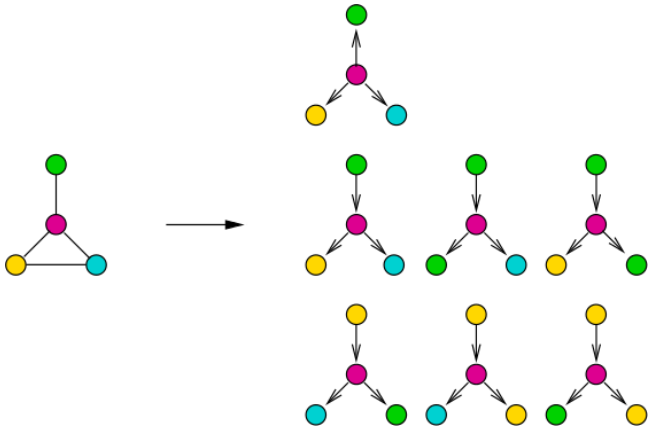


Figure 2.8: Examples of tree walks in a graph, from the original paper [1].

Our initial idea for classifying the animation characters is to consider the

region adjacency graph on the segments of the image. The intuition is that although changes in posture, scale change the overall image significantly, the arrangement of segments in the image relative to each other should roughly stay the same. Classifying images by their region adjacency graph has been studied by Z. Harchaoui and F. Bach in [1], in which they a introduce a number of kernels to compute similarity between such graphs. We chose to study and implement the tree-walk kernel presented in this paper.

The method Let I be a color image, $S = \{S_1, ..., S_q\}$ a segmentation of I and a 4-connected notion of adjacency between pixels. The region adjacency graph G of I is defined as the graph where vertices are segments and there is an edge between segments S_i and S_j if and only if there is a pair of adjacent pixels $(p_i, p_j) \in S_i \times S_j$.

Let G and H be region adjacency graphs with labelling functions $l_g : V(G) \to L$ and $l_h : V(H) \to L$, where L is a segment label set. The method considers a kernel function $k : L \times L \to \mathbb{R}^+ \cup \{0\}$ which measures similarity between segment labels. The authors first define the p-th order walk kernel $k^p_{\mathcal{W}}(G, H)$ between G and H as:

$$k^p_{\mathcal{W}}(G, H) = \sum_{\substack{(r_1,....,r_p)\in\mathcal{W}^p_G \\ (s_1,....,s_p)\in\mathcal{W}^p_H}} \prod_{i=1}^{p} k(l_G(r_i), l_H(s_i))$$

where $\mathcal{W}^p_G$ (resp. $\mathcal{W}^p_H$) denotes the set of walks of length at most p in G (resp. H).

The authors then go on to generalize this kernel not just to walks in the graph, but to what they define as a *tree walk*. An $\alpha - ary$ *tree-walk* in a graph G is any rooted tree whose vertices are vertices of G, and neighboring vertices in the tree must be neighbors in G. This differs slightly from the definition of a subtree, as this definition allows vertices to be repeated in the tree-walk as they would be in a walk - although 2 adjacent vertices in the tree walk must be distinct.

Although we could generalize the previous formula for tree walk kernels easily enough, it would be highly inefficient to evaluate as the number of tree walks of a graph is prohibitively large. In order to make this more efficient, we restrict the set of possible children of a given node in the tree walk to intervals in the cyclic order given by the "planar" embedding of the graph given by the segments of the image.

Remark. At this point, it is necessary to clarify what is meant by the "planar" embedding of the graph, as it is not described in detail in the original paper. It should be noted that the segmentation method in our program does not necessarily yield a planar embedding of the graph using the gravity center of segments, as the segments may be non convex. In fact the graph may not be planar at all using 8-connectivity for Felzenszwalb's method. Indeed, the 8-connected graph is not planar for grids larger than 5 by 5 pixels by Euler's formula on planar graphs (see Figure 2.10), and the segmentation is a minor of this graph, which could very well make it non-planar by Wagner's theorem [20] - such counterexamples were found in practice using Boyer and Myrvold's planarity testing algorithm [21]. The hue-based merging post-processing step

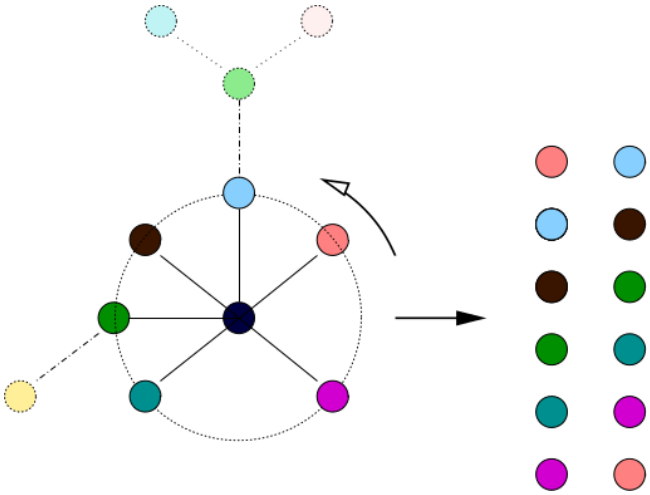


Figure 2.9: Neighbor intervals of size 2, from the original paper [1].

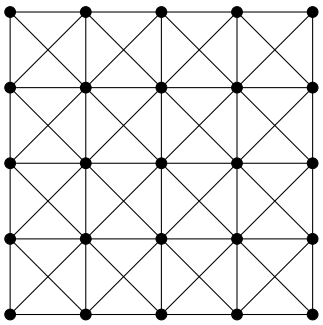


Figure 2.10: The 8-connected graph on a 5 by 5 image. It has $v = 25$ vertices and $e = 72$ edges, so Euler's planarity criteria $e \leq 3v - 6$ does not hold. Therefore any larger 8-connected graph must be non planar.

complicates things further, as it ignores the graph structure entirely. If by any chance the resulting graph is planar, then one can use Boyer and Myrvold's planarity testing algorithm to efficiently determine a planar embedding for the graph, but such an embedding may not be representative of the arrangement of segments in the image, and it may not be unique. So in the following we will refer to the "planar" embedding of the graph induced by the segmentation as the embedding given by gravity center of segments, although it may not be planar in the usual definition - hence the quotes around "planar".

Applying this, we can define the *α-ary tree walk kernel* of depth p between graphs G and H, denoted $k_{\mathcal{T}}^{p,\alpha}$, recursively as follows:

$$k_{\mathcal{T}}^{p,\alpha}(G,H,r,s) = k(l_G(r), l_H(s)) \sum_{\substack{I \in \mathcal{I}_G^p(r) \\ J \in \mathcal{I}_H^p(s) \\ |I|=|J|}} \prod_{\substack{r' \in I \\ s' \in J}} k_{\mathcal{T}}^{p-1,\alpha}(G,H,r',s')$$

Where $k_{\mathcal{T}}^{1,\alpha}(G,H,r,s) = k(l_G(r), l_H(s))$, and $\mathcal{I}_G^p(r)$ (resp. $\mathcal{I}_H^p(s)$) denotes

the set of neighbor intervals of length at most p of vertex r in G (resp. s in H). The final kernel between graphs G and H is then given by summing the recursive kernel for each pairs of root vertices:

$$k_{\mathcal{T}}^{p,\alpha}(G,H) = \sum_{\substack{r \in V(G) \\ s \in V(H)}} k_{\mathcal{T}}^{p,\alpha}(G,H,r,s)$$

Using this kernel as a similarity measure between segmentation graphs, we then classify them using the nearest neighbor rule. Although the original paper suggests using color histograms as segment labels, for animation image we found average $L*a*b*$ color to perform as well and to be more efficient. We therefore used a simple Gaussian kernel weighted for segment areas:

$$k(l_1, l_2) = A_1 A_2 e^{-\frac{||l_1 - l_2||}{\sigma^2}}$$

Where A_1 and A_2 are the areas of segment labelled by l_1 and l_2 respectively.

Theoretical justification The idea of comparing graphs for similarity leads us to the problems of graph matching, inexact graph matching and subgraph matching. However these are impractical to solve, and are anyway undesirable - we want to compare graph in a looser fashion. The authors of the original paper therefore combine approaches inspired from text classification and kernel methods to derive the tree walk kernel, and go further to show how the framework of multiple kernel learning can be used to determine the best parameters.

Implementation Using methods from dynamic programming, we can derive a polynomial time implementation which runs fast in practice. The authors show that, given graphs G and H with maximum degrees d_G and d_H and n_G, n_H number of vertices respectively, the algorithm can be implemented to run in $O(p\alpha^2 d_G d_H n_G n_H)$ time to compute all α-ary tree walk kernels of depth at most p.

Results and analysis The method performed poorly for our purposes, partly because it relies on many parameters which are very difficult to choose.

Further, it seems that the segmentation graphs for animation images have a lot of variations in the smaller substructures of the graph, which makes the relevance of small tree-walks of the graph doubtful for measuring similarity in more global features of the graph. Indeed, computing tree walks of large depth and arity to capture the larger scale features of the graph was found to be impractical. The run time of the algorithm becomes quickly unacceptable for most applications, and is explained by the polynomial complexity of the tree walk kernel computation.

However, it should be noted that the segment matching solution (see section 2.6.3), which performed best on our datasets, resembles in many ways the tree walk kernel method, using segment labels and similarity measures between segments to compute an overall similarity for the image, weighing results by segment area. While our method performs better in practice, it lacks a sound theoretical grounding, which could be provided by studying how these two methods relate.

Furthermore, study of the multiple kernel learning method described in the original paper and other works by Bach [22] may give a solution to the problem of choosing the right parameters for the algorithm.

Segmentation graph spectral classification method

The tree walk kernels method performed poorly in part because it relies overly on the smaller variations of the region adjacency graph of the image. In an effort to match graph structure more loosely, we considered the results from spectral graph theory which attempt to analyse graphs through the eigenvalues and eigenvectors of its Laplacian.

The method For each segmentation S, we consider m features $(f_i : S \to \mathbb{R}_i^q)_{1 \leq i \leq m}$. For instance, we used average $L^*a^*b^*$ color, gravity center and area of the segment as features. For each feature f_i, we compute the K-nearest neighbor graph G_i on S with edges weighted by the Gaussian kernel $w(S_u, S_v) = e^{-\frac{||f_i(S_u) - f_i(S_v)||^2}{\sigma_i^2}}$, and its corresponding combinatorial Laplacian $L_i \in \mathbb{R}^{n \times n}$ where n is the number of segments (see subsection 3.1.1 for definitions).

We then use the method from Wilson, Hancock and Luo [2] to determine a pattern vector for the graph. To do so, we consider the k eigenvectors $(e_1, ..., e_k)$ corresponding to the k smallest non-zero eigenvalues $(\lambda_1, ..., \lambda_k)$ of L_i.

This method considers the *spectral matrix* $\Phi \in \mathbb{R}^{n \times k}$ which contains the eigenvectors as columns, scaled by the square roots of the eigenvalues.

$$\Phi = (\sqrt{\lambda_1} e_1 ... \sqrt{\lambda_k} e_k)$$

Note that the original paper considers all the eigenvectors of the Laplacian. We chose to only use the k first for efficiency and looser graph comparison. Φ is well defined as L_i is positive semi definite. Furthermore, 0 is an eigenvalue of the combinatorial Laplacian for any graph with multiplicity the number of connected components, so this makes clear why we chose to ignore the 0 eigenvalues - they will only result in a 0 column for any graph. See [23] for proofs of the elementary properties of the combinatorial Laplacian (and the closely related normalized Laplacian).

The spectral matrix, although it encodes all the information of the k eigenvectors and eigenvalues, cannot directly be used as vector for pattern matching because it is not invariant to vertex permutations on the graph - although the eigenvalues are invariant to such permutations, individual components of the eigenvectors are not. Therefore the authors introduce the *elementary symmetric polynomials* $P = (P_1, ..., P_n)$ to compute n permutation invariant features out of each scaled eigenvectors:

$$
\begin{aligned}
P_1(x_1, ..., x_n) &= \sum_{i=1}^{n} x_i \\
P_2(x_1, ..., x_n) &= \sum_{i=1}^{n} \sum_{j=i+1}^{n} x_i x_j \\
&... \\
P_r(x_1, ..., x_n) &= \sum_{0<i_1<...<i_r<n} \prod_{j=1}^{r} x_{i_r} \\
&... \\
P_n(x_1, ..., x_n) &= \prod_{i=1}^{n} x_i
\end{aligned}
$$

So, for each column $\Phi_i = \sqrt{\lambda_i} e_i$ of Φ, we apply each symmetric polynomial to its components to obtain a new vector $E_i \in \mathbb{R}^n$ defined as:

$$
E_i = \begin{pmatrix} signum(P_1(\Phi_i)) \ln(1 + |P_1(\Phi_i)|) \\ ... \\ signum(P_n(\Phi_i)) \ln(1 + |P_n(\Phi_i)|) \end{pmatrix}
$$

The logarithmic scaling is added to adjust the distribution of terms in the polynomials, as suggested in the original paper. Concatenating these vectors for each of the k columns of the spectral matrix, we get a single feature vector $B \in \mathbb{R}^{nk}$. Applying this method to each graph G_i then gives us m feature vectors $(B_i)_{1 \leq i \leq m}$ which we concatenate into the final feature vector $F \in \mathbb{R}^{nmk}$ for the image. We then classify the images by these feature vectors using support vector machines.

We deal with graphs of different size - when the images have different number of segments, which is almost always the case with Felzenszwalb's method - by padding the Laplacian matrix with zeros to match the largest graph in the dataset, as suggested in the original paper.

Theoretical justification Works by Wilson, Zhu, Hancock and Luo showed that such spectral methods could be used for graph pattern matching, including image classification through shock graphs [2][24]. Furthermore, works in graph cut algorithms and application to clustering [3] and segmentation[4][25] showed that the eigenvectors of the Laplacian corresponding to smaller eigenvalue encode to some extent sense the global structure of the graph, while ignoring smaller details. We hoped this would allow the method to be robust to variations in animation character images, and encode the information about the global structure of the image.

Implementation Computing our 3 features of interest - average color, gravity center and area of the segments - can be done in one pass over the image and runs in $O(q)$ time where q is the number of pixels in the image. Computing the K-nearest neighbor graph for each feature can then be done using an algorithm solving the all nearest neighbor problem, which can be done in $O(Kn \log(n))$

where n is the number of segments in the image [26]. The resulting graphs are sparse, as the number of edges is at most K times the number of vertices.

For each graph, we can then compute its Laplacian using a sparse matrix data structure in $O(n)$ time by sparsity of the graph and using an adjacency list data structure. We can compute the multiplicity l of the eigenvalue 0 by running a connected components detection algorithm on the graph, which runs in $O(n)$ time using a simple depth first search procedure. We can then efficiently compute the k eigenvectors of the Laplacian matrix corresponding to the smallest non zero eigenvalues by computing the $k+l$ smallest eigenvectors using the Lanczos method for sparse symmetric matrices, which can be tuned to run in $O((k+l)n^2)$ time [2].

Evaluating all the symmetric polynomials for a single column of Φ can be implemented in $O(n^2)$ time using the *power symmetric functions* $Q = (Q_1, ..., Q_n)$:

$$\forall r \in \{1, ..., n\} \quad Q_r(x_1, ..., x_n) = \sum_{i=1}^{n} x_i^r$$

And the Newton-Girard formula:

$$P_r = \frac{(-1)^{r+1}}{r} \sum_{i=1}^{r} (-1)^{i+r} Q_i P_{r-i}$$

Where P_0 denotes the constant 1 polynomial. One can first compute all the power symmetric functions in $O(n^2)$ time by accumulating the exponentiation of the terms of the sum from one polynomial to the next. We can then compute all the elementary symmetric polynomials by accumulating the inner sum of the Newton-Girard formula in $O(n^2)$ for a total $O(n^2)$ running time. So evaluating the feature vector for all columns of Φ takes $O(kn^2)$ run time.

So the procedure to compute the pattern vector for a single image runs in $O(q + (k+l)n^2)$ time.

Results and analysis Recognition rates measured using leave one out cross validation on a dataset of 12 characters results in a 10% recognition rate, which is barely better than random. Other methods based on the eigenvalues and eigenvectors of various Laplacians of graphs on the set of segments were considered, with similar results.

The main issue of this method is that the Laplacian only encodes information about edges in the graph, therefore only contains information about segments relative to other segments in the image. The information about the segments themselves, and more importantly between segments of different images, is largely lost. For animation images, segment labelling information turns out to be more important than the structure of segments in the image.

[2] There is little literature on the run time complexity of the Lanczos algorithm. However, being an iterative method, we found in practice that limiting the number of iterations to n yields good enough results. Each iteration being dominated by a sparse matrix by dense vector product which runs in $O(n)$ time, and repeating the process for each of the $k+l$ eigenvectors, it is easy to deduce this run time complexity. However, the specifics of the implementation used - here from the Fortran library Arpack - may introduce more run time for the sake of numerical stability, and optimizations to compute multiple eigenvectors at once are not accounted for, so this may not be a very informative result.

Further, the theory behind the algorithm is not completely sound. The paper by Wilson, Hancock and Luo does indicate that the vectors obtained are not necessarily very useful for pattern matching, and should go through some form of dimensionality reduction to be embedded in a pattern space [2]. Indeed, this may solve the problem of graphs of different size to some extent, as padding the Laplacian with zeros effectively adds zeros to the final pattern vector.

Segment matching method

One of the big issues with the previous methods comes from the graphs on the set of segments of a single image not encoding enough information from the image for classification. In an effort to extract more information from the image and to compare the segment information between images directly, we tried to consider how one could match segments between 2 segmentations, so matched segments could correspond to the same part of the character. This resulted in the following method.

The method We once again consider some features on the set of segments of the image, namely average $L^*a^*b^*$ color, gravity center and area of the segment. Let us consider two color images I and J with segmentations $S = \{S_1, ..., S_n\}$ and $T = \{T_1, ..., T_m\}$ respectively.

We compute similarity between the segments of the 2 images using a fuzzy control system taking the similarity in features as inputs, and returning an overall similarity of the 2 segments (see Figure 2.11). The rules of the fuzzy control system were defined so closeness of the segments reinforces similarity when the other features coincide already, but does not decrease similarity much when the positions do not coincide (see Figure 2.12). The intent being to introduce robustness to variations in posture and occlusion of body parts by the frame of the image, which make segment position an unreliable indicator.

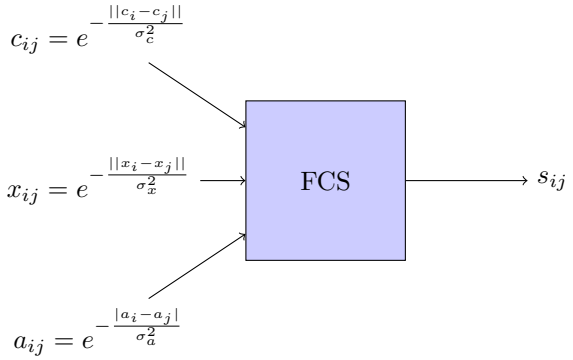
We used the defaults of the fuzzylite library [27] as choices of t-norms, s-norms and defuzzyfiers. Each scale parameter σ_c, σ_x and σ_a is automatically determined by computing the variance of the euclidean distances in feature space between all pairs of segments from the 2 images. Given feature $f_k : S \cup T \to \mathbb{R}^p$, the corresponding scale parameter σ_k is computed by the following formula:

$$\sigma_k^2 = (\frac{1}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} ||f_k(S_i) - f_k(T_j)||^2) - \mu_k^2$$

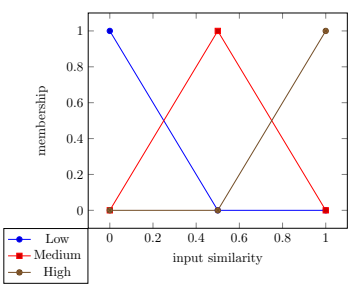
Where μ_k is the mean for feature f_k:

$$\mu_k = \frac{1}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} ||f_k(S_i) - f_k(T_j)||$$

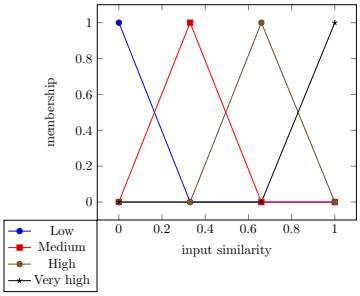
Using the similarity measures $(s_{ij})_{\substack{1 \le i \le n \\ 1 \le j \le m}}$ provided by the fuzzy control system, we now determine a one-to-one relation between two segmentations S and T such that 2 similar segments are related to each other. For this purpose, we designed a simple algorithm to determine such a relation (3). An example relation is shown in Figure 2.13.



(a) Inputs and output.



(b) Inputs membership functions.



(c) Output membership functions.

Figure 2.11: Fuzzy control system (FCS) for segment similarity. For a segment i, c_i denotes its average color, x_i the coordinates of its gravity center, a_i its area in number of pixels. s_{ij} denotes the computed similarity between segments S_i and T_j.

From this one to one relation R and the corresponding similarity measures s_{ij}, we measure overall similarity between segmentations S and T using a sum of the segment-wise similarities, weighted by segment size:

$$sim(S,T) = \sum_{(S_i,T_j)\in R} (|S_i| + |T_j|)s_{ij}$$

We then classify images according to the nearest neighbor rule by this measure of similarity.

Theoretical justification This method was designed to correct the flaws of the previous methods. As such, it focuses on comparing segment data directly between 2 images instead of trying to extract features from a single image, and classify them according to this. Although this is fruitful in practice (see the results and analysis paragraph), this approach lacks a solid theoretical foundation - although there are certainly interesting invariants to prove about the algorithm which provides the one-to-one relation, much work remains to be done.

Most standard classifiers (support vector machines, normal bayes classifier, artificial neural networks) expect feature vectors for each sample, which our method doesn't provide, hence why we consider simply the nearest neighbor for

```
string ruleStrings[] = {
        "if Color is HIGH and Area is HIGH and Position
            is HIGH then Similarity is VERYHIGH",
        "if Color is HIGH and Area is HIGH and Position
            is MEDIUM then Similarity is HIGH",
        "if Color is HIGH and Area is HIGH and Position
            is LOW then Similarity is HIGH",
        "if Color is MEDIUM and Area is HIGH then
            Similarity is MEDIUM",
        "if Area is MEDIUM and Color is HIGH then
            Similarity is MEDIUM",
        "if Area is MEDIUM and Color is MEDIUM then
            Similarity is LOW",
        "if Area is LOW or Color is LOW then Similarity
            is LOW",
        ""
};
```

Figure 2.12: Rules of the fuzzy control system in the syntax of the fuzzylite C++ library [27].

Figure 2.13: Original images (left) and corresponding relation (right). Segments with the same color are matched together.

classification. There are however a number of dimensionality reduction [5] [6], clustering [28] and classification [7] methods considering an arbitrary similarity matrix on the data like the one we have here, or can easily be extended to support this kind of input. Further theoretical work could borrow from these methods.

Implementation Consider once again 2 color images I and J, and corresponding segmentations $S = \{S_1, ..., S_n\}$ and $T = \{T_1, ..., T_m\}$.

Assuming features have been precomputed, computing s_{ij} for a given pair of segments can be done in constant $O(1)$ time, given that the fuzzy control system has a constant number of inputs, outputs and rules, so computing all the s_{ij} takes $O(nm)$ time. Sorting the similarities can then be done in $O(nm \log(nm))$ time using an optimal comparison sort such as merge sort. Producing the rela-

Algorithm 3 Segment matching algorithm.

```
1: function SegmentMatching(S = {S_1, ..., S_n}, T = {T_1, ..., T_m})
2:     Compute (s_ij)_{1≤i≤n, 1≤j≤m} using the fuzzy control system.
3:     Sort the s_ij by descending similarity into (e_1, ..., e_nm).
4:     R ← ∅
5:     for k ← 1 to nm do
6:         Let (S_i, T_j) ∈ S × T be the segments corresponding to similarity e_k.
7:         if neither S_i nor T_j are related to another segment by R then
8:             R ← R ∪ {(S_i, T_j)}
       return R
```

tion then takes $O(nm)$ time, using a list of pairs for the relation with a boolean array indicating segment membership in the relation, which takes $O(nm)$ space. Segment sizes can be retrieved in $O(\alpha(n) + \alpha(m))$ time, where α is the inverse of the Ackermann function, using a disjoint set forest data structure for the segmentations, and the relation has at worst $\min(n, m)$ members as it is one to one by construction so $sim(S, T)$ can be computed in $O(\min(n, m)(\alpha(n) + \alpha(m)))$ time from R. So overall, computing the similarity between 2 segmentations whose features have already been computed takes $O(nm \log(nm))$ time. Computing the nearest neighbor then simply requires computing this similarity between the test sample and the training samples, whose features can be precomputed as part of the training procedure.

In practice this is quite fast, as the number of segments in the images is quite small for a large enough scale parameter k to Felzenszwalb's method.

Results and analysis With a dataset containing 12 different characters with 15 sample images for each, the method has a 59% recognition rate measured by leave one out cross-validation.

Furthermore, the recognition rate scales well with the size of the dataset. Indeed, increasing the number of sample images per character from 5 to 15 increased recognition rate from 35% to 59% (see Figure 2.14). This indicates a certain degree of robustness to noise in the data - which was collected manually from the internet. More importantly, web artist communities such as Pixiv or deviantArt have thousands of images for each character, a number quickly increasing over time, which make this property very relevant.

The method does have trouble with characters sharing a similar color palette - for instance, when character have both the same hair color and the same clothes color, or otherwise large segments of similar color. Furthermore, it does not have a well defined theoretical grounding which would allow us to prove interesting properties about the algorithm.

Figure 2.14: Recognition rate of the segment matching method by number of images per character.

2.7 Possible improvements and extensions

2.7.1 Using training data to determine an ideal color space

The segment matching method, although it performs best in practice compared to the other methods, has trouble distinguishing characters drawing from a similar color palette.

Right now, the method uses the $L^*a^*b^*$ color space to compare colors against each other during prediction. The rationale behind this choice is that the $L^*a^*b^*$ color system aims to be perceptually uniform, and therefore is close in some sense to human perception. However, this may not be a desirable feature for classification, as different characters may share perceptually similar color palettes.

Using this reasoning, we could use training data to derive an "abstract" color space for ideal clustering of the training data. In such a color space, humanly imperceptible nuances of color between characters could be significant, while perceptually large differences between samples for the same character could be smoothed out.

Let us consider a color image I, and its color histogram H_I with k bins per channels. If we take the average color of each of the l most represented bins, we get some sort of color palette for the image. If we consider n labelled (or possibly unlabelled) training images in c classes, we can consider these colors as our new training samples labelled with the label of the original image. Embedding this data in some higher-dimensional space where it would be linearly separable - using manifold learning methods with a kernel function taking into account training data such as the one in [7] for instance - could give us a color subspace on which to project new test samples before classification.

It is however unclear how one would avoid the curse of dimensionality in this case - we would need a cl-dimensional space to make sure each color of each character is linearly separable. As the segment matching method only relies on

euclid distance in feature space, one could easily apply the kernel trick to the algorithm to tackle the issue. How such a kernel would incorporate training information remains to be solved, however. Further study of the literature on manifold learning, including methods for dimensionality reduction [5] [6] [29], spectral clustering [4] [30] [3] [25] and semi-supervised classification [7] could provide an answer to these questions.

2.7.2 Further study of segmentation graph based methods

A straightforward generalization of the segment matching method would have consider not just one-to-one relations on the sets of segments, but k-to-k for small k. Enumerating all such possible combinations may however be intractable. Drawing inspiration from the tree walk kernels method [1], one could use a graph structure on the set of segment to reduce the size of the space to explore - for instance a K-nearest neighbor graph. This leads us to consider sequences of distinct, adjacent segments on this graph, e.g. paths on length k on the graph. Using reasoning similar to the ones in [1], one might be able to derive efficient dynamic programming implementations to compute similarities between all such paths - as all 3 features we consider can be computed efficiently for sets of segments from the features of the individual segments.

Such a generalization would make the hue-based segment merging step of segmentation, which is mostly heuristic and greedy, mostly redundant using a proper graph structure. Further, although such graphs have proven insufficient on their own for animation image classification, perhaps this graph structure on the images can be beneficial to distinguishing characters with otherwise similar features.

This reasoning also makes clear that the segment matching method is related in some sense to the tree walk kernels method. Although our method lacks a sound theoretical grounding, the tree walk kernel doesn't and studying how they relate might provide important information about our method. Namely, and as has been mentioned in the previous section, our algorithm could easily benefit from the kernel trick.

2.7.3 Background extraction

Right now, our method expects animation character images whose background has already been removed - manually in the case of our dataset. In practice however, and as can be easily seen in web artist communities, animation images typically have a background, which rarely ever contains information relevant to identification. Background extraction or removal would therefore be a necessary step for our method to be useful in practice.

In the case of human images, the state of the art to identify the human when there is background present is to detect the face of the person using a method such as the Viola-Jones algorithm [31], and then to use a state of the art face recognition method such as the Eigenfaces or Fisherfaces algorithm [15] [16].

This technique cannot be directly applied to animation characters, as the face information is almost never relevant to identification - there is both too much variations by drawing style and too much uniformity in the way animation character faces are drawn for this. We would like to consider ideally the full body of the character, or at least a large part of the body.

We could still apply the Viola-Jones algorithm to detect the important parts of the character for identification - for instance hair, face, clothes, arms and legs - by using proper training data. Using some kind of abstract skeleton describing how these parts should be structured in the image, perhaps for different types of postures, one could then reasonably extract the entire body from the background. Our segment matching algorithm - or an improvement over it - can then identify the character.

There are however still many hurdles on the way to solve this problem, and many experiments to perform to test our hypothesises.

Conclusion

My work as a research student in Kyushu University led to significant development in animation character identification. Indeed, while there wasn't much state of the art prior to this work, our current method achieves a 59% recognition rate on a dataset with 12 characters, which is - at the very least - much better than random, which would give a roughly 8% recognition rate. Furthermore, we show experimentally that this number scales well with the size of the dataset. Although our method still suffers from issue related to characters sharing similar color schemes, we provide in this document the outline of a method which could solve these issues and we give general guidelines for further extensions. My work led to the acceptance of a paper at the Kyushu JCEEE conference, which I successfully presented at the international sessions at the end of August in Kumamoto.

During my work here, I learnt a lot through the study of numerous research papers, most notably in the fields of image processing, machine learning and more precisely applications of spectral graph theory in dimensionality reduction, clustering, (semi) supervised classification and graph pattern matching, the principles of kernel methods and their many applications.

I can therefore say that my work at Kyushu University has been very fruitful, and I plan to come back for a Ph.D. under the supervision of professor Sakamoto to extend our work on animation images analysis.

Bibliography

[1] Zaïd Harchaoui and Francis Bach. "Image classification with segmentation graph kernels". In: *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*. IEEE. 2007, pp. 1–8.

[2] Richard C Wilson, Edwin R Hancock, and Bin Luo. "Pattern vectors from algebraic graph theory". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 27.7 (2005), pp. 1112–1124.

[3] Andrew Y Ng, Michael I Jordan, Yair Weiss, et al. "On spectral clustering: Analysis and an algorithm". In: *Advances in neural information processing systems* 2 (2002), pp. 849–856.

[4] Jianbo Shi and Jitendra Malik. "Normalized cuts and image segmentation". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 22.8 (2000), pp. 888–905.

[5] Sam T Roweis and Lawrence K Saul. "Nonlinear dimensionality reduction by locally linear embedding". In: *Science* 290.5500 (2000), pp. 2323–2326.

[6] Mikhail Belkin and Partha Niyogi. "Laplacian eigenmaps for dimensionality reduction and data representation". In: *Neural computation* 15.6 (2003), pp. 1373–1396.

[7] Kiichi URAHAMA. "Semi-supervised classification with spectral projection of multiplicatively modulated similarity data". In: *IEICE transactions on information and systems* 90.9 (2007), pp. 1456–1459.

[8] *Website of the advanced graduate program in global strategy for green asia.* `http://www.tj.kyushu-u.ac.jp/leading/en/ga11-en.html`. Accessed: 2013-09-12.

[9] *History of Kyushu University on Kyushu University's website.* `http://www.kyushu-u.ac.jp/english/university/history/index.php`. Accessed: 2013-09-12.

[10] *Thomson Reuters ranking of research institutions in Japan.* `http://ip-science.thomsonreuters.jp/press/release/2011/esi2011/ranking/`. Accessed: 2013-09-12, in Japanese.

[11] *Kyushu University's Graduate school/Graduate Faculty system on Kyushu University's website.* `http://www.kyushu-u.ac.jp/english/university/information/fucultysystem.php`. Accessed: 2013-09-12.

[12] *Kyushu University's faculty of design website.* `http://www.design.kyushu-u.ac.jp/kyushu-u/english/`. Accessed: 2013-09-12.

[13] Pedro F Felzenszwalb and Daniel P Huttenlocher. “Efficient graph-based image segmentation”. In: *International Journal of Computer Vision* 59.2 (2004), pp. 167–181.

[14] Takayuki Itamochi. “An identification algorithm of illustration artist”. In: *Proceedings of the* 74^{th} *national convention of IPSJ (Information Processing Society of Japan)*. Vol. 2. 2012, pp. 209–210.

[15] Matthew Turk and Alex Pentland. “Eigenfaces for recognition”. In: *Journal of cognitive neuroscience* 3.1 (1991), pp. 71–86.

[16] Peter N. Belhumeur, João P Hespanha, and David J. Kriegman. “Eigenfaces vs. fisherfaces: Recognition using class specific linear projection”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 19.7 (1997), pp. 711–720.

[17] *JCEEE Kyushu website*. Accessed: 2013-10-04. 2013. URL: `http://www.jceee-kyushu.jp/`.

[18] M Kuwahara et al. “Processing of RI-angiocardiographic images”. In: *Digital processing of biomedical images*. Springer, 1976, pp. 187–202.

[19] Robert Endre Tarjan. “Efficiency of a good but not linear set union algorithm”. In: *Journal of the ACM (JACM)* 22.2 (1975), pp. 215–225.

[20] Klaus Wagner. “Über eine Eigenschaft der ebenen Komplexe”. In: *Mathematische Annalen* 114.1 (1937), pp. 570–590.

[21] John M Boyer and Wendy J Myrvold. “On the Cutting Edge: Simplified O (n) Planarity by Edge Addition.” In: *J. Graph Algorithms Appl.* 8.2 (2004), pp. 241–273.

[22] Francis R Bach, Gert RG Lanckriet, and Michael I Jordan. “Multiple kernel learning, conic duality, and the SMO algorithm”. In: *Proceedings of the twenty-first international conference on Machine learning*. ACM. 2004, p. 6.

[23] Fan RK Chung. *Spectral graph theory*. Vol. 92. AMS Bookstore, 1997.

[24] Richard C Wilson and Ping Zhu. “A study of graph spectra for comparing graphs and trees”. In: *Pattern Recognition* 41.9 (2008), pp. 2833–2841.

[25] Marina Meila and Jianbo Shi. “A random walks view of spectral segmentation”. In: (2001).

[26] Kenneth L Clarkson. “Fast algorithms for the all nearest neighbors problem”. In: *FOCS*. Vol. 83. 1983, pp. 226–232.

[27] Juan Rada-Vilela. *fuzzylite: a fuzzy logic control library in C++*. 2013. URL: `http://www.fuzzylite.com`.

[28] Ulrike Von Luxburg. “A tutorial on spectral clustering”. In: *Statistics and computing* 17.4 (2007), pp. 395–416.

[29] Lawrence K Saul et al. “Spectral methods for dimensionality reduction”. In: *Semisupervised learning* (2006), pp. 293–308.

[30] Leo Grady and Eric L Schwartz. “Isoperimetric graph partitioning for image segmentation”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 28.3 (2006), pp. 469–475.

[31] Paul Viola and Michael J Jones. “Robust real-time face detection”. In: *International journal of computer vision* 57.2 (2004), pp. 137–154.

Chapter 3

Appendices

3.1 Conventions

This section establishes the mathematical conventions and definitions used in this document. They are provided to make this document self-contained, please see "Spectral Graph Theory" by Fan R. K. Chung [23] (from which these conventions are borrowed) for a more thorough treatment of spectral graph theory.

3.1.1 Spectral graph theory

In this document, a(n undirected) *graph* $G = (V, E)$ consists of a *vertex set* V and an *edge set* $E \subseteq \{\{u, v\} | u \in V \text{ and } v \in V\}$. We say that vertices u and v are *adjacent*, denoted $u \sim v$, if and only if $\{u, v\}$ is an edge. A *weighted graph* consist of a graph $G = (V, E)$ with an associated *weight function* $w : V \times V \to \mathbb{R}^+ \cup \{0\}$ where w is symmetric and there is no edge $\{u, v\}$ if and only if $w(u, v) = w(v, u) = 0$. A graph is called *simple* if it has no loops (e.g. no edge $\{u, v\}$ where $u = v$). A graph is called *finite* if its vertex set is finite. Unless indicated otherwise, we assume graphs to be undirected simple finite weighted graphs. We note that unweighted graphs can be seen as weighted graphs with weight function w defined by:

$$w(u, v) = \begin{cases} 1 & \text{if } u \sim v \\ 0 & \text{otherwise} \end{cases}$$

Common graphs used in this document include:

- the *K-nearest-neighbor graphs*, which given feature $f : V \to \mathbb{R}^q$ for $q \in \mathbb{N} - \{0\}$ connects a pixel to its K nearest neighbors by euclid distance in feature space. Since the K-nearest-neighbor membership relation is not symmetric, we distinguish between 2 cases [28]:
  - The graph where there is an edge between u and v if and only if v is one of u's K nearest neighbors *or* u is one of v's K nearest neighbors, which we simply call *K-nearest-neighbor graph.*
  - The graph where there is an edge between u and v if and only if v is one of u's K nearest neighbors *and* u is one of v's K nearest neighbors, which we simply call *mutual K-nearest-neighbor graph.*

- the *complete graph* in which all pairs of distinct vertices have an edge between them.

Let $G = (V, E)$ be a graph. For notational convenience, and since we assume graphs to be finite, we will interchangeably use the vertex and its index in the $\{1, ..., |V|\}$ set. We define the *adjacency matrix* $A \in \mathbb{R}^{n \times n}$ of G where $n = |V|$ by $A_{uv} = w(u, v)$, and the *degree* d_u of vertex u by $d_u = \sum_{u \sim v} w(u, v)$. Let $D = diag(d_1, ..., d_n)$ be the *degree matrix* of G.

We define the *combinatorial Laplacian* $L \in \mathbb{R}^{n \times n}$ of G by the following:

$$L = D - A \quad \Leftrightarrow \quad L_{uv} = \begin{cases} d_u & \text{if } u = v \\ -w(u, v) & \text{if } u \sim v \\ 0 & \text{otherwise} \end{cases}$$

the *normalized Laplacian*[1] $\mathcal{L} \in \mathbb{R}^{n \times n}$ by:

$$\mathcal{L} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \quad \Leftrightarrow \quad \mathcal{L}_{uv} = \begin{cases} 1 & \text{if } u = v \text{ and } d_u \neq 0 \\ -\frac{w(u,v)}{\sqrt{d_u d_v}} & \text{if } u \sim v \\ 0 & \text{otherwise} \end{cases}$$

where

$$D^{-\frac{1}{2}}_{u,v} = \begin{cases} \frac{1}{\sqrt{d_u}} & \text{if } u = v \text{ and } d_u \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

and the *random walk Laplacian*[2] $L_{rw} \in \mathbb{R}^{n \times n}$ of G by:

$$L_{rw} = D^{-1} L = I - D^{-1} A \quad \Leftrightarrow \quad (L_{rw})_{uv} = \begin{cases} 1 & \text{if } u = v \text{ and } d_u \neq 0 \\ -\frac{w(u,v)}{d_u} & \text{if } u \sim v \\ 0 & \text{otherwise} \end{cases}$$

and D^{-1} is defined similarly to $D^{-\frac{1}{2}}$.

3.1.2 Image processing

In this document, an *image* I is a function $I : \{1, ..., h\} \times \{1, ..., w\} \to \{0, ..., 2^d - 1\}^c$ where h is the *height* of the image, w its *width*, d its *depth* and c its *number of channels*. I is called a *color image* if and only if $c = 3$, and a *grayscale* image if and only if $c = 1$.

Let I be an image, and $G = (V, E)$ be a graph where $V = dom(I)$ is the set of pixel coordinates of I. A *segmentation* $S = \{S_1, ..., S_m\}$ is a partition of V where each S_i is a connected component in G. Common graphs considered for segmentation are:

- the 4-*connected graph*, which connects pixels directly under, over, left and right of each pixel.

[1] Both the combinatorial and normalized Laplacian are commonly refered to as "the Laplacian of G" in the literature. To avoid confusion, in this document we will use these names as they were defined in [28].

[2] The name comes from the fact that this matrix is closely related to the transition probability matrix of a random walk defined on G [28].

- the 8-*connected graph*, which is the 4-connected graph with additional diagonal edges.
- the K-nearest-neighbors graphs for some pixel features - usually a mix of color and coordinates information.