

Web Content Mining

IKT 408 – Web Mining

Autumn 2006



<p><i>Authors</i></p> <p><i>Sigbjørn Tvedt</i></p> <p><i>Christian Kroken</i></p>	<p><i>Supervisor</i></p> <p><i>Morten Goodwin Olsen</i></p>
<p><i>Version:</i></p> <p><i>Status: Final</i></p>	<p><i>Pages: 19(including this page)</i></p>
<p><i>Keywords:</i></p> <p><i>Image classification, image analysis, text detection</i></p>	

Abstract

Our purpose is to classify an image into different categories, deciding whether it's an image, logo, a text or a mathematical formula. We convert the image and then run it through our interpreter made in ruby using the Camellia picture library. Most of our implementation is made in ruby supported by three other languages to get the functions we need.

Our main concern is to find out what classifies an image as being in a certain category and getting an acceptable rate of success. We also need to find effective ways of making the job easier for the interpreter, by dividing the image into smaller parts for example.

The benefits of being able to categorize images is that you can find out the content of the image. This improves accessibility by making it easier for people with poor eyesight to browse web pages for example.

Table of Contents

Abstract	2
1 Executive summary	5
2 Introduction	6
3 Background	7
4 Problem description	8
5 Requirements specification for analyzer	9
6 Design of the analyzer	10
6.1 Color detection	10
6.2 Line detection	10
6.3 A high level description of the classification method	11
7 Implementation	12
7.1 Languages used	12
7.1.1 Java	12
7.1.2 C#	12
7.1.3 Python	12
7.1.4 Ruby	12
7.2 How the software works	13
7.2.1 Decide the colors we want to scan for	13
7.2.2 Text pattern detection	13
7.2.3 Intersection detection	13
7.2.4 Classifying a image as a formula	13
7.2.5 Classifying a text as a logo	13
7.2.6 Labeling something as a image	14
7.3 Decreasing the amount of elements wrongfully classified as images	14
7.4 Increasing the efficiency	15
7.4.1 Maximum intersections	15
7.4.2 Example:	15
8 Evaluation and testing	16

<i>9 Discussion</i>	17
<i>9.1 Project outcomes</i>	17
<i>9.2 Evaluation of the overall results</i>	17
<i>10 Conclusion</i>	18
<i>References:</i>	19

1 Executive summary

Images are widely used on the internet today. They are usually given to illustrate a text, draw a logo containing special fonts or to display text that the creator of the page is unable to, or does not know how to represent by other means. As these images usually is inaccessible or difficult to access for people who are blind or has other difficulties with their eyesight we want to create a automatic classifier for these images so that it is possible to know if the content of a image

We have decided to base this project on the Camellia Image Processing & Computer Vision library (1). By using this and creating our own algorithms, we have created a fairly fast scanner that is able to sort a given image into one of 4 categories.

Some of the most difficult tasks have been to distinguish between the different types, as they are floating into each other. How artistic should we allow a text to be before we call it a logo, or when does a logo with lots of graphical elements turn into a picture?

We have been successful in creating a set of algorithms that are able to distinguish between the different categories most of the time. By adjusting some small parameters, it is also possible to tune the speed versus accuracy ratio. We have implemented this option to make the software more versatile and thus easier to use in different settings.

2 Introduction

The main purpose of our project is to classify an image into different categories, deciding whether it's an image, a logo, text or a mathematical formula. This will be done by running it through our interpreter made in ruby. By using this approach we can improve the accessibility a web page for both physically challenged people, or for use on mobile devices. The process itself consists of converting the image to a manageable format, and scanning for various colors and shades of colors.

As we want to be able to use the scanner in real-time, we are using the KISS method, also called "Keep It Simple, Stupid". By following this "guideline" we have created a fast analysis program that is able to classify most images given to it correctly.

By using camellia (1) and Ruby (2) we have created a set of algorithms that enables us to quickly analyze a large amount of images and classify them.

We have also given the scanner the option of performing a more extensive scanning in order to create more accurate results, and made it able to "cheat" during the analysis in order to increase the detection speed.

We chose this project because it seemed challenging and useful. It is also a good way to learn the ruby language and to be able to create some algorithms of our own.

3 Background

There have been made some attempts to classify images, and most of them have been fairly successful. Most attempts at image recognition have been made in fields unrelated to web-based problems. Image recognition have been used to find patterns for use in Artificial Intelligence and is often described as belonging to the field of Artificial Intelligence. On example of use is getting a robot through an environment by recognizing its surroundings. (6)

There has been two earlier projects here at HiA where the goal where to create an image analyzer and classifier. They have tried to approach the problem by using for instance statistics, pattern recognition and edge detection.

The 2004 project (3) chose to ignore edge detection and assumed use of png or gif for text and formulas. They used statistical methods to determine the nature of an image (Poisson, Entropy, standard deviation, line scan, Benford's law).

The 2005 project (4) used edge detection, making it easier to distinguish between objects and the background. They used three different algorithms (Entropy, standard deviation and pixel search. Also called line scan) to determine the type of the image.

4 Problem description

We want to be able to categorize images containing logos, pictures, formulas and text. The approach we want to use is that we want to use the different colors of an image to detect patterns in structure of the image instead of trying to create a OCR software that scans for predefined patterns. This way we should also be able to detect unknown characters and uncommon fonts as text.

The first problem we encountered where how we wanted to distinguish a logo with a text and an image. We had the same problem differentiating between formulas and text.

Example:

The formula $\cot x * \tan x = \sin x$ is readable as a straight forward string, and is therefore impossible to detect as a formula as long as we also want to detect strings like "this is a plain text" and classify them into different categories.

The CTRL+ALT+DEL logo contains artistic text on a graphical background. It is therefore extremely hard to detect the text in this image when comparing with, for instance, the EIAO webmining logo. It also contains some graphical elements, but they are still a part of the text. We have therefore decided that we are going to classify "artistic" logos as images while the EIAO logo could be called a text or a text logo.



Figure 1: The CTRL+ALT+DEL comic logo



Figure 2: The EIAO webmining logo

5 Requirements specification for analyzer

The image analyzer should be able to classify most images correctly. It should also be able to detect if an image is a text. We are not going to create software that does OCR.

We want to be able to divide the images given to the software into 4 different categories. These are pictures, formulas, text and logos containing text.

We also want to make it really fast, so we have to make sure the algorithm stays simple.

We would like our interpreter to have a success ratio of around 80%, thereby making it suitable for practical use.

The text pattern analyzer should only be able to detect horizontal lines as catering for all the different angles will demand more processing power and make the software much more complex. We do not want to be able to detect multiple lines of text in the same image.

6 Design of the analyzer

The image will first be processed to detect the colors that are used as background colors. These colors will not be used when detecting regions that are classified as interesting.

We then detect all interesting areas from the image and sort them. The next step is to analyze the interesting squares and then try to detect if they are aligned on a line. The last part of the analysis is to detect if the interesting areas are intersecting each other

6.1 Color detection

We have defined some colors that will should be able to detect most colors used in a image, but as we do not want to detect the background color, we have to remove these from the detection procedure. Since the background color varies, depending on the colors used by the fonts in the image, we have to rebuild the array of colors we want to detect for each image. We also have to keep in mind that some images have a small border around them, and still be able to detect the background color.

6.2 Line detection

In order to detect if we are working on an image containing a text we want to check if most of the interesting areas are in between two lines. This way we may detect text strings and logos containing mostly text.

6.3 A high level description of the classification method

When speed is more important than the percentage of correctly classified images, the results are interpreted as outlined by the diagram below

	<i>Image</i>	<i>Text</i>	<i>Text logo</i>	<i>Formula</i>
<i>Intersections</i>	Yes	No	Yes	No
<i>Line detection</i>	No	Yes	Yes	No

Table 1: interpretation of the scanner results

We have also implemented more extensive algorithms, like the DoubleScan method.

These are discussed in chapter 7.3 and 7.4

7 Implementation

7.1 Languages used

We have been using 4 different languages in this project. Each of them has different strengths and weaknesses, and our knowledge of them vary greatly. We have tried to develop most of the project in Ruby, while using the other languages to create helping functions that would be difficult to implement in ruby.

7.1.1 Java

In order to download images automatically, we had to use a web crawler. As the objective of this project is to analyze images, not to create a way to download them, we are going to use WebLeech (7)

7.1.2 C#

We have used C# from Microsoft in order to create a image converter. This small software converts all images into BMP files that Camellia is able to work on.

7.1.3 Python

Due to the lack of image processing libraries in ruby, we had to create a small script that could split the images when using the "Double scan" option. This is done by using the Python Image Library (PIL) (5)

7.1.4 Ruby

The main implementation has been done in ruby by using the Camellia image library and all image classification is done here.

7.2 How the software works

As the software we have created depends heavily on the Camellia Image Processing and Computer Vision library, most of these methods consist of passing data to Camellia and analyzing the results. Below is an outline of the most important methods we have created and how they are used to classify the images.

7.2.1 Decide the colors we want to scan for

The software will scan an image for blobs using different colors. All colors that return a blob that has the same width or height as the image we are working on are then discarded since they probably are the background color. If all colors are discarded, we try to scan using all the discarded colors.

7.2.2 Text pattern detection

By analyzing the blocks we are using the average top and bottom value to define where we expect to locate a text. When performing this scan, we are using a minimum pixel height and width in order to avoid that small blocks like punctuations or an area of an image with lots of different colors, are able to confuse the detector. The way we have implemented the detector, results in it being unable to detect lines of text that are not on, or close to, a horizontal line.

7.2.3 Intersection detection

By scanning for areas where the blocks are crossing each other, we are able to detect if elements in a image are intersecting each other. When used together with the text pattern detection, we are able to detect formulas like $f = \lim_{n \rightarrow \infty} \left(1 + \frac{1}{n}\right)^n$. The formula $f = \sin x + \cos 2x$ will however be detected as a line of text since most of the characters have the same size

7.2.4 Classifying a image as a formula

As common properties of the formulas we want to detect are that its height is larger than the used font *
2. Examples of formulas satisfying these requirements are $\tan \theta = \frac{\sin \theta}{\cos \theta}$ and $\frac{y}{4} = x^2$. The formulas that contain ordinary characters and are fitted on one line, like $f = \sin x + \cot y$ will be detected as a ordinary line of text.

The properties we are searching for are therefore few intersecting areas while the detected areas are scattered in a way that makes the text pattern detection algorithm classify the given image as not text.

7.2.5 Classifying a text as a logo

When classifying an image as a text logo, we want to be able to classify the image as a image and a text. This is done by using the results from the text pattern detection and the intersection detection.

If the image is classified as a text, but the amount of intersecting areas are above a certain threshold, we believe this to be evidence of textual elements and artistic graphics inside the same image.

7.2.6 Labeling something as a image

If a image is not classified as a image, it probably fits here. Common properties of an image are that the text pattern detection algorithm is unable to locate a text pattern, while many of the detected blocks intersect with each other.

7.3 Decreasing the amount of elements wrongfully classified as images

After performing the normal scan, all pictures that area classified as images, will be divided into two parts and then rescanned. The scanner will use the same method as the normal scan, but the result will be two values. It is therefore necessary to combine these results. This combination is given by the table 2.

	Image	Text	Text logo	Formula
Image	Image	Text Logo	Text Logo	Formula
Text	Text Logo	Text	Text Logo	Text
Text Logo	Text Logo	Text Logo	Text Logo	Text Logo
Formula	Formula	Text	Text Logo	Formula

Table 2: Combining the results from the DoubleScan

7.4 Increasing the efficiency

7.4.1 Maximum intersections

As all photos will be detected with a lot of intersections, we are offering the option to define a number of intersections that will classify a picture as a image. The higher the value given, the more precise the detection will be, but a high number also reduces the advantage of using this method.

7.4.2 Example:

44 files	Time used with DoubleScan Total/average	Errors with DoubleScan	Time used without DoubleScan Total/average	Errors without DoubleScan
Without maximum intersections	23.469s 0.574s	3 Errors 3 not classified	14.125s 0.353s	4 Errors 4 not classified
With maximum intersections set to 100	12.469s 0.304s	2 errors 3 not classified	5.344s 0.134s	3 Errors 4 not classified

Table 3: Time used to analyze 44 files with different content

As we are able to see from this table, it is possible to greatly reduce the time the analyzer are using by setting a maximum amount of intersections needed before the picture in question is classified to be an image. It does however introduce some more errors.

The combination of splitting images into two and analyzing each part will in cases where there are many images be faster than analyzing the large image as one single unit. It also gives more correctly labeled images as long as the images are large. The smaller a image is, the more inaccurate the scanner becomes

8 Evaluation and testing

We have created a set of images that we wanted to test. This set has been based on the definitions given in chapter 7, and limitations from chapter 5, in order to make sure that we are not counting images, which we should not be able to categorize, as errors.

As we expected, we have been able to categorize all images, texts and formulas correctly, while the logos are the most difficult to put into the right category.

Input	Result	Image	Text	Logo	Formula
Images (13)		100%	0	0	0
Texts (6)		0	100%	0	0
Logos with text (16)		19%	50%	31%	0
Formulas(7)		0	0	0	100%

Table 4: Percentage of analyzed images correctly identified

We have, as table 4 shows, been able to detect most of the images correctly. We do however have some problems detecting the logos correctly. This is because of the diversity of the logos that we are analyzing.

9 Discussion

9.1 Project outcomes

We are able to categorize most of the images correctly, but some errors still exist. The analyzer is able to quickly scan a large amount of images while categorizing them in a satisfactory way. As we predicted, there are some problems when trying to categorize logos, but we are satisfied with the result we have gained.

In order to classify these graphical elements correctly, we need to separate the image into two pieces, which is the processed. This is a more computer intensive approach, but it usually gives better results than the standard mode as long as the size of the original image is not too small.

Further expansions could be to implement histogram analysis in order to separate photos from drawn images. Another expansion should be to make the image cropping more dynamic, thus enabling it to crop the image into parts based on the amount of information stored in each area of the image. We would also recommend that an algorithm for cropping the images based on horizontal lines is implemented. This would enable the software to scan images and detect if they only contains text, like most spam images, or if they contain graphical elements like a photo.

9.2 Evaluation of the overall results

The combination of pattern detection and color intersection has been highly successful, and we are really satisfied with the performance of the software.

As we have been unable to locate similar approaches to the process of image classification, we feel that we have found a good new method for the computer classification of images.

The decision to create the software in Ruby where done since the Camellia Image Processing & Computer Vision library is written in C, but is available for use with Ruby. As we wanted to do a “proof of concept” rather than a highly optimized software, we settled for Ruby as the main language. If this is to implemented as a large system, we would however recommend that the ruby code is translated to C in order to reduce the image processing time.

10 Conclusion

As long as content creators are unable to display the content they want in a way that is understandable by most large web browsers, they are going to create content as images, thus making it inaccessible for people using text only browsers.

Since there are a lot of different ways to display text and images, a method that fit all will probably not be possible to create without using large amounts of computing power.

Our implementation relays heavily on the work done by the Camellia project, and due to their high speed image analysis, we have been able to create a high speed classification program for images while being able to keep the percentage of correct images on a high level.

Due to the implementation of the two different scanning levels, we are able to give the user the choice between a fast classification or a more accurate, thus slower, analysis. Possible areas where this could be used is together with a OCR software and a text based browser. We could then avoid that the OCR software tries to detect text in a image that we have found that contains only images.

References:

1: <http://camellia.sourceforge.net/>

2: <http://www.ruby-lang.org/en/>

3: http://www.eiao.net/webmining/previousprojects/ikt407_deliveries/gruppe3/ProjectReport.pdf

4: <http://www.eiao.net/webmining/previousprojects/reportgroup5.pdf>

5: <http://www.pythonware.com/products/pil/>

6: http://en.wikipedia.org/wiki/Image_recognition

7: <http://weblech.sourceforge.net/>