ABSTRACT
Our purpose in this research is to develop a methodology to automatically and efficiently classify web images as UML static diagrams, and to produce a computer tool that implements this function. The tool receives as input a bitmap file (in different formats) and tells whether the image corresponds to a diagram. The tool does not require that the images are explicitly or implicitly tagged as UML diagrams. The tool extracts graphical characteristics from each image (such as grayscale histogram, color histogram and elementary geometric forms) and uses a combination of rules to classify it. The rules are obtained with machine learning techniques (rule induction) from a sample of 19000 web images manually classified by experts. In this work we do not consider the textual contents of the images.

Keywords
UML diagram recognition; Image processing; Rule induction.

1. INTRODUCTION
Our purpose in this research is to develop a methodology to automatically and efficiently classify web images as UML static diagrams, and to produce a computer tool that implements this function. The tool receives as input a bitmap file (in different formats) and tells whether the image corresponds to a diagram.

This tool is useful as long as it improves the effectiveness of common image search engines. For example, Google currently offers the possibility of advanced search of images with a filter set to “line drawing”. Using this filter, a generic search argument such as “control system” provides nearly no results that can be recognized as software engineering diagrams. Simply adding “uml” to the search argument dramatically increases the success of the search in the first obtained results with Google Images (nearly 100%), but after the first dozens of results the precision drops to around 90%. Perhaps this is not that bad, but we can improve it. In fact, adding “uml” to the search argument is a lot more relevant to the results that using the line drawing filter.

However, adding “uml” (or a similar term) to the search argument will detect only those images that either have been manually labeled with this tag, or that are surrounded by text that contains it. Our tool, instead, does not require that the images are explicitly or implicitly tagged as UML diagrams, therefore offering the additional advantage that the search
argument can be more specific of the problem at hand: a search argument like “control system”, without “uml” or similar tags, will produce results where images that are not UML diagrams are filtered out, with a higher degree of precision.

The tool can be used, then, in several ways. First, it can be connected to the output of a generic search engine (such as Google). The initial result set is provided by the search engine, therefore using the tool will not improve its recall; however, the precision of the results will improve (as our experiments show), since more false positives will be discarded.

Second, the tool can be integrated in a specific UML-oriented search engine. In fact we are developing such a search engine, including a web crawler that indexes UML diagrams, with the potential to improve not only precision, but also recall. In this paper we are presenting the classification tool in isolation, but it is part of a larger information retrieval project.

The tool is useful also when employed to search private repositories in an organization. In general, the utility of such a tool is manifested in a context of software reuse, where the capability to correctly identify software engineering artifacts such as analysis or design diagrams is most beneficial. Once the artifact has been located, the next step in a reuse context would be extracting the information it contains in a format that is directly conformant to the UML metamodel, so that the output can be further processed to develop new software. Besides, other related artifacts that are linked to the identified diagram (such as code implementations) can also be retrieved and reused. However, in this paper we consider neither the problem of information extraction from the diagram, nor that of retrieving related artifacts, focusing only on the correct classification of images as diagrams. Of course, if the local repository is made up with files produced with modeling CASE tools, the classification, retrieval and extraction of relevant information will be a lot easier, and a tool to classify images and extract information from them will not be necessary. However, we think the local repository can include also bitmap images, or even diagrams produced with general drawing tools (maybe prior to the adoption of a specific CASE tool by the organization); in that case metadata will not be enough for effective retrieval and our tool will be most useful.

Our work is limited to bitmap images (or, more properly, pixmap images, i.e. spatially mapped arrays of colored pixels), either uncompressed bitmaps such as BMP, or compressed formats such as GIF, JPEG, PNG or TIFF [7]. We do not consider, instead, vector graphics. The work is also limited to UML static diagrams (classes, components) which are mainly made of straight lines and rectangles, leaving diagrams that use rounded forms (such as use case diagrams, activity diagrams and statecharts) for future research and development. Since the tool is tuned to detect static diagrams, other diagrams that use mainly straight lines and rectangles (such as sequence diagrams) will produce a certain number of false positives.

The methodology we have followed can be roughly described as follows. First, we obtained a sample of images from the web (nearly 19000 items) that resulted from queries in Google Images involving the terms “uml diagram”; then a team of experts manually classified the images as UML static diagrams (Yes/No). Second, we analyzed the main graphical characteristics of images that represent diagrams, such as grayscale histogram, color histogram, elementary geometric forms detected with image recognition techniques, and so on. Third, using machine learning techniques (rule induction), we selected a combination of characteristics that maximizes the effectiveness of the automatic classification. Fourth, we implemented the rules in a tool that is freely available for the public. The rest of this article explains in detail these steps, followed by final sections devoted to related work and concluding remarks.
2. OBTAINING A SAMPLE OF IMAGES AND MANUAL CLASSIFICATION OF THEM AS DIAGRAMS

In order to obtain a large sample of images from the web, we queried Google Images with the terms “uml diagram” and other search arguments from various application domains. We obtained nearly 19000 items, including UML diagrams, other kinds of diagrams, and non-diagrammatic pictures (see Table 1).

<table>
<thead>
<tr>
<th>Category</th>
<th>Sample size</th>
<th>Fraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive: UML static diagram</td>
<td>794</td>
<td>4.2%</td>
</tr>
<tr>
<td>Negative: other diagrams</td>
<td>6703</td>
<td>35.5%</td>
</tr>
<tr>
<td>Negative: not a diagram</td>
<td>11402</td>
<td>60.3%</td>
</tr>
<tr>
<td>TOTAL</td>
<td>18899</td>
<td>100.0%</td>
</tr>
</tbody>
</table>

The UML is very formally specified regarding its abstract syntax (its metamodel), but its concrete syntax (its graphical notation) admits a larger degree of freedom. Therefore, for pragmatic reasons, we have restricted ourselves to the simplest kinds of diagrams: computer-edited 2D representations of static diagrams, leaving the rest for future developments.

A team of experts was in charge of classifying the sample of images obtained from the web. They proceeded according to the following criteria:

Classify as **Positive** those images that:

1. Contain geometrical figures similar to those used in UML to represent classes, objects, notes, components, packages, ports, etc. (that is, rectangles or variations of rectangles).
2. The figures contain textual information inside.
3. The figures are linked to each other.

Classify as **Negative** those images that do not satisfy the preceding rule, more specifically:

1. The image satisfies the rule only partially, or only a certain portion of the image does.
2. The image is not the product of a tool, but a manual drawing.
3. The image is not a flat representation of a diagram (however, typical 3D effects or shadows to highlight flat figures are allowed).
4. The image contains overlapping elements, ornamental elements, or any other kind of ambiguous elements.

Dubious cases must be classified as Negative. See examples of the application of these criteria in Figures 1 through 5, and results of the classification of the full sample in Table 1.

The large fraction of UML-unrelated images (in fact including any kind of pictures) guarantees that the dataset is not biased towards UML diagrams and can be effectively used to train an automatic classifier.
Figure 1. Example of image classified as Positive

Figure 2. Example of image classified as Negative (no rectangles)

Figure 3. Example of image classified as Negative (only a portion of it is a diagram)

Figure 4. Example of image classified as Negative (manual drawing)
3. ANALYSIS OF GRAPHICAL CHARACTERISTICS OF DIAGRAMMATIC IMAGES

UML diagrams always contain textual information that is most relevant for their interpretation. However, analyzing the textual content of an image is computationally very time-consuming, especially when compared with graphical features that are a lot easier to compute and provide enough information to classify the image as a diagram. Therefore, in this work we do not consider the textual contents of the images.

Typical UML diagrams have a short number of different gray tones (or colors, when used), and they use elementary geometric forms such as rectangles (or combinations of rectangles), straight polylines (vertical, horizontal and oblique), rhombuses, triangles, circles, ellipses, rounded rectangles, and so on. Next we explain how these characteristics can be effectively and efficiently used to classify a given image as a diagram.

3.1 Grayscale histogram

A typical diagram (UML or not) has a low number of different gray tones (see Figure 6). Tone, or intensity, is defined as \( I = (R+G+B)/3 \). However, some UML diagrams can show usage even of all gray tones due to shadowing, a decorative element found very often (see Figure 7). Observe in this case that, even if the distribution is highly skewed, i.e. white pixels
having intensity near 255 are predominant\textsuperscript{1}, the diagram uses all different tones of gray. If we used only the number of different tones to discriminate (like “an image is a diagram if it has a number of different tones less than a given threshold”), we would get here a false negative.

Besides, some pictures, even of natural landscapes, may feature also a very few number of tones (see Figure 8). In this case we could get a false positive because the absolute number of gray tones is also low, comparable to the histogram of a typical diagram.

Summing up, the absolute number of different gray tones (obtained from RGB information encoded in the bitmap file) provides useful information and is very easy to compute, but this criterion alone is not enough to discriminate diagrams.

\textbf{Figure 7.} Grayscale histogram of UML diagram in Figure 1

\textbf{Figure 8.} Grayscale histogram of an artistic landscape picture

3.2 Color histogram

UML diagrams are not necessarily black and white (with gray tones), they frequently appear in color. But, when colored, the number of different hues is also usually very small. Hue is the angular component in the polar representation of the HSI (hue-saturation-intensity) color

\textsuperscript{1} We show only the lower part of histograms, to better appreciate the differences. As we have explained, we are not interested in the absolute number of pixels per intensity value, but the number of different intensity values present in the image.
space, i.e. the angle in the familiar color wheel representation, from 0 to 359 degrees, with a precise mathematical relationship to RGB encoding [7]. As in the case of gray tones, we can find UML diagrams that use a big number of different hues; therefore they could be false negatives, if having few hues in the color histogram were used alone to discriminate (see Figure 9); we can also find pictures with a very low number of hues that could be false positives (see Figure 10). Therefore, the absolute number of different hues is also useful but insufficient to discriminate diagrams.

![Figure 9. Color histogram of a normal UML class diagram, featuring many different colors (hues)](image)

![Figure 10. Color histogram of picture in Figure 8](image)

### 3.3 Elementary geometric forms

Since this work is focused on static UML diagrams, we do not consider all possible geometric forms in the UML notation, in particular we do not consider rounded forms. Even if some of them can appear in class and component diagrams (e.g. circles for interfaces, or curved lines for relationships), their significance is low when the purpose is not information extraction, but only classification of the image as a diagram. Besides, we disregard also oblique lines for the sake of computational efficiency. This includes not only oblique relationship lines, but
also n-ary association symbols, arrowheads, and other kinds of relationship terminators. In other words, we consider only vertical and horizontal straight polylines (solid or dashed), a special case of which is the rectangle, by far the most common geometrical form in UML diagrams.

The first step is to identify the pixels that belong to a border, according to the following rule: a pixel belongs to a border (it is tagged as ‘black’) if it has a horizontally or vertically adjacent pixel (tagged as ‘white’) with a gray intensity at least 80 levels above (this is a heuristic value), in a scale from 0 to 255. Then we identify straight lines (borders) consisting of series of horizontally or vertically contiguous black pixels, allowing for short discontinuities in dashed lines. Each detected segment receives a unique identifier within the diagram. The basic image processing procedure is described with greater detail in [17]. Four cyclically connected segments form a rectangle, which receives also a unique identifier. Other figures composed of several superimposed or adjacent rectangles, such as components, ports, and packages, are conveniently processed.

These two values (number of vertical and horizontal segments, and number of rectangles) are rather easy to compute and they discriminate very well UML static diagrams, especially when combined with the analysis of histograms (see Figure 11), so that we can avoid the more time consuming identification of other features, such as text or more complex geometrical forms. However, the concrete values and combination of rules to discriminate diagrams is not so easily found by intuition. Therefore we used machine learning techniques to find an optimal combination, as explained in the next section.

![Image](image.png)

**Figure 11.** A picture with a very low number of different colors can be discarded as a diagram because it features very few straight segments and rectangles

### 4. SELECTION OF GRAPHICAL CHARACTERISTICS AND RULE INDUCTION BY MACHINE LEARNING TECHNIQUES

The discrimination of images representing UML static diagrams is achieved in this project by an automatic classifier that uses data mining techniques of supervised learning. Machine learning techniques are mechanisms to obtain patterns through the analysis of data. This is a subfield of computer science that evolved from the study of pattern recognition and computational learning theory in artificial intelligence. Machine learning explores the construction and study of algorithms that can learn from data, by building a model from
example inputs in order to make data-driven predictions or decisions, rather than following strictly static program instructions [1].

Machine learning techniques can be supervised or unsupervised [20] [22]:

- Supervised learning: the algorithm is presented with example inputs and their desired outputs, and the goal is to learn a general function that maps inputs to outputs.
- Unsupervised learning: the inputs to the learning algorithm are given no output labels, leaving the algorithm on its own to find structure in its input.

More specifically, supervised learning is the task of inferring a function from labeled training data. The training data consist of a set of training examples, each example consisting of an input object (in our case, an imaged downloaded from the web) and a desired output value (in our case, the manual classification as a UML static diagram, or not). A supervised learning algorithm analyzes the training data and infers a generalized function, which can be used for classifying new inputs.

Rule induction is a kind of supervised machine learning, where the inputs are described by a set of common attributes. The training examples include also the output attribute. In our case, the output attribute is the classification (i.e. the picture is, or is not, a UML static diagram) and the input attributes are:

1. Number of gray tones.
2. Number of color hues.
3. Number of vertical/horizontal segments.
4. Number of vertical/horizontal segments at least 30 pixels long.
5. Number of horizontal segments at least 30 pixels long.
6. Number of vertical segments at least 30 pixels long.
7. Number of rectangles.
8. Number of main rectangles (not included in other rectangles).

Note we are using the number of vertical/horizontal segments in four different ways, in order to increase their discriminatory power.

Rule induction techniques process input training data and produce a set of IF-THEN rules used to classify the new examples [4] [9]. Two main strategies are commonly used:

- Produce a decision tree and then extract its rules.
- Generate the rules covering all the examples in a given class; exclude the covered examples and proceed with the next given class, until all classes are covered.

The first strategy is implemented in the C4.5 system [19], which extends the previous ID3 [18]. Other algorithms such as PRISM [3] are based only on covering, whilst PART [8] combines both strategies.

These strategies present the following advantages to minimize the impact of errors in the manual classification of the images used as training examples:

- Robustness against noise due to errors, omissions or insufficient data.
- Identification of irrelevant attributes.
- Detection of absent attributes or knowledge gaps.
- Extraction of expressive and easy to understand rules.
- Possibility to interpret or modify the produced rules with aid of expert knowledge, or even to incorporate new rules inferred by the experts themselves [16].
In order to improve the effectiveness of the individual classifiers obtained by means of rule induction, ensemble methods construct a set of classifiers and then classify new instances by taking a (weighted) vote of their decisions [5]. The technique has two main variants:

- Homogeneous classifiers are generated with the same learning algorithm [6]. The main methods are Bagging [2] and Boosting [21].
- Heterogeneous classifiers, instead, are generated with different learning algorithms. The most used method is Stacking (Stacked Generalization) [24].

We finally chose the Bagging method of homogeneous classifiers on the basis of the rule induction PART algorithm [8].

5. EXPERIMENTS AND RESULTS OBTAINED

As we have explained, we have applied the rule induction PART algorithm combined with the bagging approach of homogeneous classifiers. The experiments were implemented in the popular Weka suite [23], keeping its standard default parameter configuration. We used as training examples the set of 18899 images downloaded from the web, manually classified by the experts as explained in Section 2. From these initial dataset, a combination of rules was extracted that can be extrapolated to classify new images.

Some examples of the individual rules obtained are:

- **IF** (number-of-main-rectangles <= 0) AND (number-of-segments <= 11)
  **THEN** Negative (it is not a diagram)
  This rule classified 9445 of our 18899 examples as non-diagrams, with 55 false negatives (0.6 %).

- **IF** (number-of-main-rectangles > 2) AND (number-of-hues <= 6)
  **THEN** Positive (it is a diagram)
  This rule classified 176 examples out of 18899 as diagrams, with 7 false positives (4 %).

As we have explained, the algorithm provides not only individual rules, but a function that combines the rules and maximizes the overall results.

We have performed 10 experiments with the available dataset. In each experiment we have used the 794 positive instances (manually classified as UML static diagrams, see Table 1) together with other 794 negative instances (other diagrams and non-diagrammatic pictures), randomly chosen, without replacement, from the total set of negative instances. We have performed the usual 10-fold cross validation [15], obtaining the classifiers shown in Table 2.

6. THE i2m CLASSIFICATION TOOL

We have developed the experimental i2m (image-to-model) tool according to the principles exposed in this research. The current version of the tool\(^2\) accepts as input a bitmap file (in different formats) and tells in less than 1 second whether the image corresponds to a UML static diagram. The tool is freely accessible on the web.

\[^2\] i2m v1.0, image-to-model tool: [http://i2m.kr.inf.uc3m.es](http://i2m.kr.inf.uc3m.es)
Table 2. Summary of the results of the classifiers obtained with the 10 experiments: precision, recall and percentage of agreement in classifying images as UML static diagrams\(^3\)

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Precision</th>
<th>Recall</th>
<th>Agreement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.956</td>
<td>0.928</td>
<td>94.3 %</td>
</tr>
<tr>
<td>2</td>
<td>0.950</td>
<td>0.933</td>
<td>94.2 %</td>
</tr>
<tr>
<td>3</td>
<td>0.945</td>
<td>0.943</td>
<td>94.4 %</td>
</tr>
<tr>
<td>4</td>
<td>0.965</td>
<td>0.936</td>
<td>95.1 %</td>
</tr>
<tr>
<td>5</td>
<td>0.963</td>
<td>0.942</td>
<td>95.3 %</td>
</tr>
<tr>
<td>6</td>
<td>0.956</td>
<td>0.931</td>
<td>94.4 %</td>
</tr>
<tr>
<td>7</td>
<td>0.936</td>
<td>0.923</td>
<td>93.0 %</td>
</tr>
<tr>
<td>8</td>
<td>0.955</td>
<td>0.940</td>
<td>94.8 %</td>
</tr>
<tr>
<td>9</td>
<td>0.952</td>
<td>0.923</td>
<td>93.8 %</td>
</tr>
<tr>
<td>10</td>
<td>0.953</td>
<td>0.929</td>
<td>94.2 %</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>0.953</strong></td>
<td><strong>0.933</strong></td>
<td><strong>94.4 %</strong></td>
</tr>
</tbody>
</table>

Reminder: *Precision* is the ratio of true positives to all positive detections (the complement is the ratio of false positives); *Recall* is the ratio of true positives to all existing positives in the data set (the complement is the ratio of false negatives); and *Agreement* is the ratio of true positives plus true negatives to all existing instances.

\(^3\)Figure 12. The i2m (image-to-model) tool
We plan to develop future versions of the tool that will implement the following features:

- Classify a set of images, instead of one-by-one images.
- Classify other kinds of diagrams (use case diagrams, state transition diagrams, and so on).
- Desktop version with the capability to explore private repositories.
- Use the output of a generic search engine (such as Google) as input to the tool, so that the tool is used to further filter the output of the generic search engine and improve its precision.
- Integrate the image classification tool in a specific UML-oriented search engine, including a web crawler that indexes UML diagrams, with the potential to improve not only precision, but also recall.
- Extract textual information from the diagram with the aid of OCR technologies. Combine with the previous feature to search the web for specific diagrams containing a given text.
- Interpret the diagram according to the UML metamodel, producing its output as a model in XMI format.
- Generate a graphical UML representation of the extracted model.

All these ongoing developments, each one with different degree of completion, form part of a larger UML-oriented information retrieval project.

7. RELATED WORK
Karasneh and Chaudron have published research on the same problem [12], although they do not use an automatic learning approach, but a fixed set of classification criteria. Their Img2UML tool [13] extracts UML class models from pixmap images and exports them into XMI files that can be read by a commercial CASE tool (StarUML). They have created a private repository with images collected from the internet together with their corresponding XMI extraction [14]. In this sense, their work covers nearly all the steps we envision for future versions of our tool. However, their tool does not presently solve all the problems they have indentified (in particular, OCR recognition of text, recognition of relationships, and lack of XMI standardization in tools). Besides, they use a small number of images to validate the tool (respectively 10 and 200 images in [12] and [13]).

Ho-Quang et al. [10], based on a previous work by Hjaltason and Samúelsson [11], and also in collaboration with Karasneh, further investigate image features that can be effectively used to classify images as class diagrams. Using a training set of 1300 images, and with a success rate lower but similar to ours (90%-95%), they need instead a much higher average processing time to classify each image, nearly 6 seconds (our tool requires less than 1 second). Therefore, our approach demonstrates that using less attributes produces similar results but in a sensible lower processing time.

8. CONCLUSIONS AND FUTURE WORK
We have developed a methodology and a tool to classify web images in different formats (BMP, GIF, JPEG, PNG and TIFF) as UML static diagrams, without requiring that the images are explicitly or implicitly tagged as UML diagrams. The tool uses a combination of rules acquired through machine learning techniques on a dataset of nearly 19000 instances; the rules consider graphical characteristics of images, such as grayscale histogram, color histogram, number of straight lines and rectangles, and so on.
With nearly 95% of agreement with manually classified instances, this tool is useful to improve the effectiveness of common web image search engines like Google Images. The tool is useful also to search private repositories in an organization, when metadata are not available.

This research project is focused on the development of a UML-oriented information retrieval tool. However, once the technology has been fully developed, it can be easily expanded to be applied in different and promising domains, such as building plans, electrical circuitry, and so on.

9. REFERENCES
10. SOURCES OF IMAGES
Fig. 1: http://tweakers.net/nieuws/48673/microsoft-uml-gaat-niet-ver-genoeg.html
Fig. 2: http://docs.oracle.com/cd/E13197_01/rfid/edge_server/docs20/prog/read_tags_api.html
Fig. 3: https://technology.amis.nl/2004/11/04/working-with-omondos-eclipseuml-free-edition/
Fig. 4: http://www.agilemodeling.com/artifacts/classDiagram.htm
Fig. 5: https://www.edrawsoft.com/images/network/3dnetworksample.png
Fig. 6: http://articulo.mercadolibre.com.mx/MLM-18487899-cursos-completos-manuales-de-c-ejemplos-envio-gratis-_JM
Fig. 8: http://thadallender.photoshelter.com/image/I0000X1lt1LV9h0
Fig. 9: https://sites.google.com/site/analisisdiazirving/unidad-ii/resumen-3
Fig. 11: https://www.flickr.com/groups/flickr_aid/