Modern Book Catalog

The first Vector Database for image retrieval

Arpad Kiss, BSc 2014 @ Óbuda University

CONTENTS

1. ABSTRACT

My main goal in this project was to design and develop a system, which can provide a new, efficient way for library's visitors to discover the stored documents through a photo based search method. The system is applying the tools of image processing, machine learning and data mining.

This paper is deeply describing the preprocessing algorithms, which are used for the book cover recognition, multiple approaches to gain feature vectors, and as well the supervised and unsupervised machine learners. Explains the problems of storing and searching pictorial features, shows more method for speeding up the process. Presents the general approaches of optical character recognition, problems of post-processing and possible solutions.

The core of the system was written in C++, the network communication managed by a Node JS based REST API, which have an Objective C written mobile client application. The metadata of the books are stored in schema-less JSON format, the features are stored on the file system in XML format. The system is using Open CV for certain image processing algorithms, the optical character recognition based on Tesseract framework.

2. INTRODUCTION

2.1 Goal

The aim of the implemented application is to extend the traditional input-field-based search system in the library catalogue, with a more lifelike experience from the users' perspective. The application will not only provide an opportunity for regular browsing, but also making wish-lists, bookmarks or gathering more in-depth information about certain documents.

At the end of the project, users will be able to use the application on their phone, and they will be capable of searching in the system by the help of a book cover photo. My aim is to make the process easy and fast. The new approach operates on image processing, feature extraction, and machine learning fundamentals.

2.2 Digital Library Catalogue

Library catalogues [1] have been used since ancient times. However, the traditional card-based library catalogues have been replaced by their information and communication technology (ICT)-based counterparts. For the sake of an easy and fast searching process through the documents stored in the library's archives, these systems provide an interactive interface for users. Documents are ordered to main-, and subcategories according to a predetermined framework, that is called Universal Demical Classification (UDC). This internationally recognised framework is a numeric coding system. For instance, within the field of animal taxonomy, the identifier 599.742.13 signifies the dog, regardless of language. An international organisation is responsible for the maintenance of this numeric coding system. They ensure concept coherence and the hierarchical structure. The general applied classification of UDC is illustrated in the first point of the appendix.

UDC has lost its significance due to the spread of digital libraries, nevertheless libraries still digitally store and classify documents using this numeric coding system.

Formal exploration is an important step during the classification and insertion of documents into a database. This process consists of gathering basic data: title, subtitle, title in the original language, serial title, author, translator, illustrator, director, consultant, and the names of other contributors, date-, and place of publication, name of publisher, sentiment analysis data and technical data. Otherwise, the ISBN (International Standard Book/Serial Number) is commonly stored for each document. This internationally recognised unique identifier encodes both the country of origin and the publisher. The exact storing of scientific and other ranks at authorship attribution is infrequent because it can cause problems at the making of alphabetical order. However, this data is usually stored in a different field.

Content exploration is also an important step. This process happens during the insertion of documents because documents do not contain content features directly. Subject terms, also known as tags, are commonly used for these features. The number of these depends on the richness of the document's subject and the depth of the exploration work. This step is a crucial factor for later searches in the catalogue, as well as for uncovering potential associations among the documents.

2.3 Storing Image Features in Database

One key feature of the implemented system is that it allows users to access detailed information about a document based on a photo of the book cover they have taken. The identification process is carried out by the examination of the correlation between the so-called descriptors extracted from the book cover on the photo, and the descriptors stored in the database. These descriptors are typically represented as vectors of arbitrary dimensionality.

The identification process should be completed in the shortest possible time as it is a key factor in the system. If we store 1000 books, the cover of these books, and the corresponding descriptive vectors in the database. determining which stored book matches our search sample requires comparing every book in the database and then selecting the one with the most matches as a potential solution. This method's efficiency decreases exponentially with the amount of data stored, since each sample comparison adds complexity in O(N*K) time, where N is the number of dimensions in the vector being examined and K is the number of dimensions in the current sample.

2.4 Possible Issues

As with any image processing system dealing with information extracted from real-world spaces, significant issues can arise due to information loss from varying lighting conditions. Different lighting can also lead to such significant changes in the colours of objects in the image under examination that it impacts the pattern recognition step, causing a substantial portion of the extracted descriptors to be lost.

A second problem may occur from the orientation of our object. To extract reliable descriptors from a book situated in real-world space, the object needs to be transformed onto the image plane. This transformation requires a book-shaped model, with its X and Y components proportional to one another as it is typical for books, namely 1 to 1.456. The issue emerges when the transformed image's longer main axis maps onto the narrower axis of the model. The interpolation step during transformation leads to significant information loss, it potentially alters the positions of contours and thereby characteristic points. This makes it inevitable to determine the object's orientation before transformation, and allows us to modify the model used during mapping. This modification involves an affine transformation, specifically a 90-degree rotation. The

resulting image can then be used regardless of orientation in subsequent steps, with the quality of the image's proportions influencing the outcome.

The third problem – that may occur – is speed, which may cause the most trouble, since one part of my goal is to complete the identification step as quickly as possible. It is necessary to examine only those elements of our database, that may contain thousands of book covers, that show substantial correlation with the pre-processed sample. In cases we cannot use multi-threaded processing for the comparison, linear searching can be resource- and time-intensive. Therefore, the examination process requires high quality of selection and accuracy of the covers, and the filtering on the database side must happen quickly. In addition to this, we must pay attention to the framework applied during the examination.

3. RESEARCH

3. 1 Book Cover Recognition

To conduct further analysis on a book cover positioned in real space, it's necessary to handle the cover independently. For this step, the incoming image needs to be processed using various pre-processing algorithms to meet the required standards. This involves removing noise generated during the photographing process, identifying the segment of the image that contains the book cover, and then transforming the resulting object to the image plane for further processing.

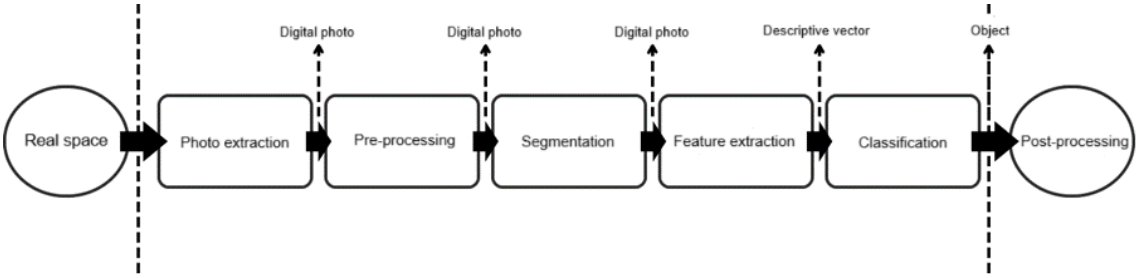


Figure 1: flowchart of traditional image processing programs

3.1.1 Pre-Processing

The phase of pre-processing can be divided into three parts: noise filtering, segmentation, and transforming the segment into space. The filtering is a window-based operation, where the values of specific pixels on the resulting image are derived from their respective neighbourhoods. During the filtering process, the resulting image's size does not change from the initial image, and the calculation is performed pixel by pixel, with the window represented by a matrix.

During noise filtering, we perform a filtering operation on our incoming image, which is capable of removing damaged pixels with outlier intensity values arising from sampling or correcting them by assigning a new value derived from their surroundings. Noise filtering can be done in both linear and nonlinear ways. In the linear case, we can apply an averaging filter, where the new pixel intensity value is the average of the intensity values of the surrounding pixels, or a Gaussian filter, where a weighting value is assigned to the pixels covered by the mask. In the Gaussian filter, the central pixel generally receives the highest weight, with weights decreasing progressively away from this point. This mask approximates the Gaussian function. In the nonlinear case, this can be done with a median filter.

The segmentation step involves the task of extracting the examined object from its context in the given image. Segmenting the image I (x,y) includes dividing it into

related sub-images. When a certain similarity condition is met, a unit is formed. This similarity condition can be intensity equality or if the difference in intensities within the region does not exceed a threshold value, as well as if the variance of intensity values within the region is small. Segmentation is considered successful if the regions are homogeneous, contain no gaps, and there is a significant difference between neighbouring regions.

Segmentation can be based on binarization, edges, regions, or fitting. In the case of binarization, we can use global, local, or dynamic thresholding, with the histogram formed from the pixel intensity values of the given image serving as the starting point. For edge-based segmentation, we start from the gradient image obtained during the differentiation of the image. In a gradient-based approach, areas with well-connected, closed boundaries form cohesive regions. In this case, noise present in the images can create so-called false edges, making noise filtering a crucial step before generating the gradient image. Edge detection not only provides information about the local gradient values but also their direction, which is useful for feature-point detection in later steps.

3.1.2 Orthographic Projection

In the next step, the segment representing the book cover needs to be transformed onto the image plane. Since the book cover can appear in virtually any orientation on the photo taken by the camera, we need to consider the camera's focal length during this operation, as it can significantly distort the result. Given that we are performing a 3D-to-2D projection, the solution might be to apply an orthographic projection to determine the transformation matrix for the book cover.

In orthographic projection, the rays arriving into the camera from the object are considered parallel, which approximates the focal length with infinity. Our object is represented as a well-defined, rectangular segment with four corner points. The projection of the point $[a_x, a_y, a_z]$ in three-dimensional space to the point $[b_x, b_y]$ is calculated as follows:

$$b_x = s_x a_x + c_x \tag{1}$$

$$b_y = s_z a_z + c_z$$

where s is the so-called scaling factor, and c is the offset. The parametric equations written in matrix form are as follows:

$$\begin{bmatrix} b_x \\ b_y \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & 0 & s_z \end{bmatrix} \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} + \begin{bmatrix} c_x \\ c_z \end{bmatrix} \quad (2)$$

3.1.2 Descriptive Extraction

After the perspective transformation, we get the spatially transformed projection of the book cover. Since our goal is to compare it with the samples in the database, it's necessary to gather some features from the image. We need to collect features that are invariant to various geometric and photometric transformations, such as translation, rotation, scaling, brightness, and exposure. A common approach is to gather local features, as these effectively characterise the given object. They are robust, and can be found in large quantities per image. The descriptors can include vectors derived from colour, shape, structure, texture, or feature-point-based approaches.

Good feature points are generally found in a small neighbourhood. We consider points to be feature points when they exhibit a strong response compared to their surroundings. Feature point detection starts from the previously mentioned gradient image, and points with a strong response are those found at the intersection of multiple edges, which means their eigenvalues are computed within a small neighbourhood and they show up as local maxima. When searching for feature points, it's important to use a detector that's invariant to translation, scaling, and affine transformations. Gaussian image pyramids can be generally applied to improve the results. The descriptors derived from these feature points typically contain information from a small neighbourhood around the point, so we also need to ensure that the descriptors are invariant.

One such descriptor extraction method is the Multiscale Oriented Patches [3] descriptor. The algorithm starts with a 40x40 pixel window around the feature points. For feature calculation, the window size is first reduced to 8x8 pixels by applying a pre-filter, then rotated horizontally. The rotation uses the angle calculated from the smoothed gradient direction of the point. In the final step, the normalised intensities of these 8x8 windows are used as a descriptor, which is first reduced by the mean, then divided by the standard deviation.

In a colour-based approach, the values derived from calculations performed on the specific pixel intensity values of the images are used as descriptors. These can include the average, minimum, and maximum values, as well as the variance of the pixel intensities of the given image.

This paper does not cover shape, structure, and texture descriptors, as we aim to compare clearly defined rectangular objects, and using shapes obtained from further segmentation of the transformed cover would significantly increase complexity.

3.2 Indexing of Images and Searching in the Database

The question of image retrievability is an extremely complex and challenging task. This complexity can arise from the size of the database, the difficulty of interpreting images by computers, the complexity of phrasing queries, and the intricacies of evaluating the results. There are several solutions to this problem, including the IBM QBIC[4] system, MIT Photobook [5], WebSEEK [6], CMU Informedia [7], and Stanford WBIIS [8], which emerged in the 1990s. The common feature of these systems is that they all aim to solve the problem of image searchability based on so-called signatures extracted from images.

Signatures typically originate from pixel-based descriptors and define predefined comparison rules for the system's operation. The elements of signatures are derived from the previously mentioned features. One advantage of using signatures instead of pixel intensity values is that the image representation becomes highly compressible, although the true reason for its application lies in the database partitioning that arises from recognizing the correlation between image representations and their semantics, resulting in significant inherent speed gains. Therefore, the primary task in determining signatures is the semantic analysis of the images' pixel representation. These types of systems are collectively referred to as Content Based Image Retrieval (CBIR) systems, and based on the approaches for extracting signatures, we can distinguish three categories: colour histogram-based, colour layout-based, and region-based searches. There are systems that combine these approaches.

After extracting the signatures, the next step is to define the rules for comparing the desired images, i.e., the similarity measures that can indicate the likelihood of correlation between different images. In most systems, a query also uses an image, and in some systems[9], it is possible to search based on a predefined region of this image.

In histogram-based search, the characteristics of an image are examined by analysing the distance between the curve derived from its pixel intensity distribution and the curve derived from the same distribution of database images. Many distance metrics are used to compare these curves, such as Euclidean distance, Histogram Intersection distance[10], and Histogram Quadratic distance, which is derived from the cross-correlation of the individual examined points of the histogram. The disadvantage of this approach is that we lack information about the position, shape, or texture of the examined object, and the colour histogram-based search is sensitive to variations arising from lighting, cropping, and intensity variance.

The colour layout-based approach attempts to eliminate problems arising from these errors by partitioning the images into blocks[4] and storing the average intensity values calculated in these blocks. In some systems[8], the coefficients of the Daubechies wavelet transform are stored as block descriptors. In this latter solution, increasing the

block size and the level of the wavelet transform allows the colour intensity values to be tuned to individual blocks, which can yield better search results. Another advantage of this approach is that at an appropriate resolution, it can retain the shape, position, and texture of our objects, though the search remains sensitive to translation, scaling, and rotation.

In the WALRUS [11] system, problems arising from translation and scaling were addressed by partitioning the images into blocks comprehensively. This partitioning was achieved using a moving kernel mask of various sizes, with the colour signature of each block calculated for each window. The image comparison was then performed by comparing the sub-images. The drawback of this system manifests in increased computational performance and time the search requires, and the shapes and textures of the objects remain ignored as features, with only the average colours of the sub-image blocks being used as features.

The third group consists of systems that employ region-based search. The goal of these systems is to implement search at the object level, thereby overcoming the issues associated with colour-based approaches. In these systems, images are first segmented, and these decompositions are then used in the searching process. This approach is designed to approximate the interpretation of image context in a manner similar to human perception, despite the fact that real images are merely 2D projections of the 3D world perceived by humans, and computers are not taught to perceive 3D space in the same way as humans.

In these systems, since the objects located in the images are treated individually, it becomes easier to compare them with objects in other images, regardless of the orientation and size of the specific object.

Two such systems are NeTra and Blobworld. A common feature of these systems is that they examine images through the comparison of individual regions. The number of regions found in each image is limited, and the search results are based on the aggregation of matches to these unique regions. For example, if an image contains three objects, the results will be a list of matches corresponding to each region, leaving the final decision to the user. It is evident that this approach provides greater control to the user, as it allows them to search by considering the texture and color of the regions. However, the representations of individual regions are overshadowed by the user's semantic understanding.

A further drawback of these two systems is that they do not place significant emphasis on defining measures of similarity, instead relying on information from individual regions. However, there is an approach that attempts to address this gap. In this approach, labels are assigned to each region based on their characteristics, and a descriptor made from these labels subsequently defines the regions. A so-called CRT

descriptor is created from the descriptor, which is intended to determine the regions based on the relative order of the labels. During analysis, these CRT descriptors are used for comparison. Unfortunately, this solution is not robust enough, as the order of the labels can change with shifts, scaling, and rotation, and the system's performance largely depends on the size of the dictionary containing the labels. The labels are derived from colour, shape, and texture characteristics, and with the increase in the range of values, the computational complexity grows exponentially, thus efficiency at large scales is not guaranteed.

One hybrid system designed to address the issues of previous systems is SIMPLYcity, which is based on the premise that the semantic characteristics of relevant images are very similar. This system is grounded in a supervised learning approach, aiming to significantly reduce the time required for image search by classifying images into different categories. It performs a classification on both stored and queried images, and categorise them into semantic subcategories such as graphic, photograph, textured, non-textured, indoor, outdoor, urban, landscape, it contains people, and it does not conation people. Each semantic class uses a different database and conducts a unique classification, with unique features stored for each class. For textured images, for example, shape-related features are not used since texture-based energy features are much more significant here. The system determines regions by channel-based colour averaging and six features extracted during a high-frequency wavelet transformation. It initially divides images into 4x4 blocks, calculating features within these blocks. Regions are determined by the results of a K-means clustering calculated between the blocks, where the number of expected groups approaches a maximum value, and an empirically calculated threshold value for goodness is used as an exit condition. For colours, it uses the LUV colour space, where L is luminance and U and V represent chromaticity. This colour space is excellent for calculating colour distances. Wavelet transformation is computed using Haar transformation. The system developers have introduced an integrated region matching (IRM) distance measurement in their method, which is designed to correct errors arising from weak segmentation. After the preliminary classification and segmentation, depending on the semantic class, the system uses features based on texture, colour, and shape. Ultimately, this system offers an extremely robust solution to the problems listed previously, capable of producing accurate results even when the orientation, colour intensities, and surface characteristics of the searched pattern vary.

3.3 Descriptor Comparison and Classification

The analysis of the correlation between descriptors extracted from images and those stored in the database is a complex task in Content-Based Image Retrieval (CBIR) systems. In the case of colour-based descriptors, our task is to explore features that can significantly characterise our specific image, considering errors caused by colour

distortions due to lighting. The colour arrangement-based approach discussed previously may be the most effective method for examining these features, but it is essential to ensure that comparisons are made in a colour space where distances can be uniform across variations in channels. It is evident that this property does not hold in the commonly used RGB colour space, where changes in the intensity values of individual channels may not significantly alter the distance between the new point in RGB space and the original point, yet a very different colour perception is experienced by human perception. To address this problem, the XYZ and LAB colour spaces are suitable, among others. In these colour spaces, the Z and L components represent only the brightness of the colour, while the X-Y and A-B components encode its hue. The LAB colour space is essentially a non-linearly compressed variant of the XYZ space, designed to approximate human colour perception. In these colour spaces, distances are generally calculated using the Euclidean formula. This shift to LAB or XYZ colour spaces from RGB allows for more accurate and perceptually relevant colour comparisons. This is because these colour spaces are designed to more closely align with the ways humans perceive colour differences, making them more suitable for applications where colour fidelity and perceptual similarity are crucial.

Feature-based image matching relies on pairing characteristic points found in two images. These methods strive to optimise a measure of similarity, either by directly solving systems of equations or by following a direct strategy. It is crucial to identify the same feature points in both images, meaning the transformations between these point groups should be similar. To discover these transformations, a method that is tolerant to shifts, rotations, scaling, and affine transformations must be employed.

The direct strategy aims to find pairs of points, typically starting by minimising the distance between a given point and points in the other image. This method works even if the matching is ambiguous, but it is crucial to introduce some kind of goodness/utility function to handle incorrect matches. To evaluate goodness, a threshold value is usually set to regulate the distance between points. Mathematically, X represents the explanatory variables, Y the variables to be explained, and f(.) : X -> Y denotes the classification function. The goal is to maximise the expected value of E[U(Y, f(X))], where U(y, y') is the previously mentioned utility function, and y' is the approximation of y.

Algorithms such as K-Nearest Neighbour (KNN) and Support Vector Machine (SVM) are often used to solve the problem. Both methods are supervised learning techniques where the objective is to determine which class the input data belongs to, essentially assigning a label to it. The classification occurs within a Euclidean vector space. In general, the training data are represented by numerical values, and the dimensionality of these values determines the number of dimensions of the vector space in which the classification is performed. The hypothesis of the classification is that documents

belonging to the same class form a contiguous region within the vector space. Classifications are differentiated between partitioning and selecting classifications, where in the latter case, the classes are mutually exclusive.

During training, a process known as model building occurs, in which labels are assigned to each training data point. This is a supervised labelling, which means that during the training process, for each class, it is necessary to determine the separating hyperplanes between the set formed by the points contained in the class and the complementary sets. These hyperplanes become the boundaries that will later define whether a given object under examination can be classified into that particular class.

3.4 Further Acceleration of Search

In cases with a large set of values, classification can be extremely computationally demanding, as we must determine the placement of each of our objects in the previously estimated N-dimensional vector space. One way to accelerate this process is by performing this model building during the prediction step, so we only need to assess the probability of falling into a given class, eliminating the need to repeat the training step with each examination. Another acceleration option is to group the classes according to some criteria, a technique collectively referred to as Branch-and-bound. Here, we need to ensure that we organise the classes into groups in such a way that their distribution closely approximates the expected value distribution of the objects we want to classify. An additional speed-up can be achieved by sorting our samples by some criterion during the examination, thereby reducing the generally O(n) time-consuming linear search to a binary search, meaning ideally the search can be completed in just O(log n) time.

Furthermore, we can use KD-trees to accelerate the search process by dividing the decision space into hyperrectangles. During classification, we select the hyperrectangle that contains the given class, and the decision falls on the class closest to our examined point. However, if our examined point is closer to the boundary of such a hyperrectangle, we also include the class found in the neighbouring hyperrectangle in our set of values, thus refining the evaluation. When building a KD tree, we must ensure that the tree is balanced, with each branch and leaf containing the dimensions of a hyperrectangle. During construction, we follow a simple heuristic: we start with the classes that have the greatest variance when selecting axes, and for the dividing point, we take the median of the points in the given class.

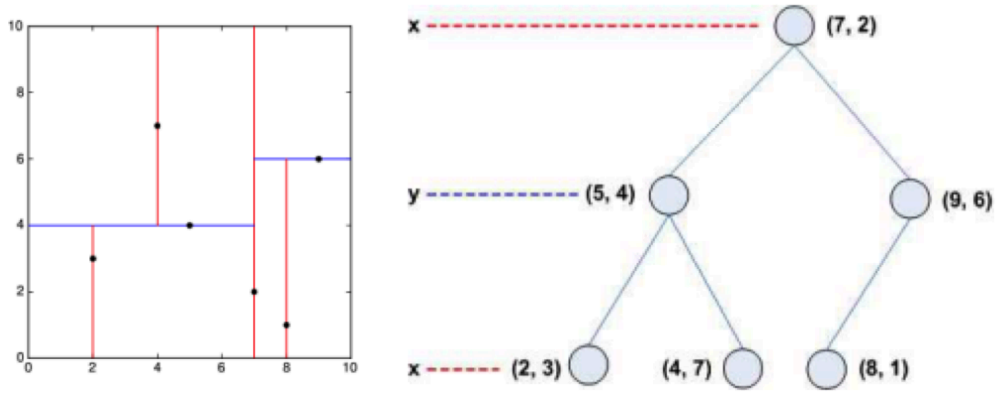


Figure 2, KD-tree hyperrectangles and tree representation

3.5 Optical Character Recognition

During image-based identification, problems can arise if the cover of a book is damaged, or if different editions of the book have different covers. Furthermore, during recognition, there may be cases where the book covers do not contain enough information that can be used as features, containing only the title and author of the book. To avoid incorrect classification resulting from this, it is necessary to enhance the search using textual information found on the cover, in addition to image features. Various approaches are used for character recognition, which always begins with a preprocessing phase. For the identification of individual characters, it is important that they can be treated as separate entities. A good approach is to perform binarization or edge-based segmentation on grayscale versions of the images, and then conduct further examinations on well-separated regions aligned in the same direction. The search for characters is carried out by matching these regions, typically based on shape-related features such as skeletons, shape numbers, moments, or features derived from vertices.

To segment letters, a commonly used algorithm is the Stroke Width Transform, which is based on the assumption that the width of characters typically correlates to a consistent value, changing only slightly within a small vicinity.

In the second phase, after feature extraction, the goal is to identify individual objects. Since numbers and punctuation marks are mutually exclusive, selective classification is generally used to recognize these characters.

The third phase involves post-processing, where multiple issues may be encountered. The first one is the incorrectly classified characters. To correct these errors, techniques such as locality-sensitive hashing, neighbourhood analysis, and/or Levenshtein distance are typically employed. Conducting these examinations generally requires the presence

of a dictionary, as the specific occurrence of characters varies by language, and better results can be achieved based on the combined occurrence of various character groups.

The Levenshtein distance is intended to provide a solution to the problem of determining how many permutations would make up the transformation of a target word into a given word when the two words are similar. Mathematically, the distance between two words, a and b, is defined by the following recursive formula:

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases} \quad (3)$$

As an example, consider the words "kitten" and "sitting", which have a distance of three, as the following three permutations are necessary to transform one word into the other and there is no shorter path to accomplish this:

- kitten → sitten (“k” instead of “s”)
- sitten → sittin (“i” instead of “e”)
- sittin → sitting (an additional “g” at the end of the word)

The Levenshtein distance defines five rules for the lower and upper bounds of each distance value:

- The distance is always at least the difference in length between the two words
- The maximum distance can be as much as the length of the longer word
- The distance is zero only if the two words are identical
- If the lengths of the two words are the same, the distance is equal to the Hamming distance between the two words
- The Levenshtein distance between two words can be at most as large as the sum of their distances from a third word

It is easy to see that using this concept of distance can greatly assist our investigations into individual word matching.

3.5.1 Tesseract by Google

This framework was developed and is maintained by Google as an open source project, allowing anyone to modify and make it suitable to specific tasks. Originally designed for processing Latin languages, considerable effort is made within Google to extend its applicability to Chinese, Japanese, Korean, Arabic, and Hindi. These languages present

unique challenges due to their vast existing character sets, the direction of text writing, and in the case of Arabic, the inseparability of certain characters. Thus, they strive to interpret these languages specifically. Clearly, creating an interpreter specific to a given language is easier than developing a globally applicable solution, as classifying a large set of values remains a challenging research area even today.

Initially, their system finds some global optimum and then creates a binarized version of the input image. Subsequently, it performs connected component analysis on this binary image to identify regions representing characters. As the framework was designed to also recognize text on both white-on-black and black-on-white backgrounds, it uses only the boundaries of these regions thereafter. Since it is very rare for Latin characters to contain more than two holes, this characteristic is well-utilised as a termination condition during pre-selection.

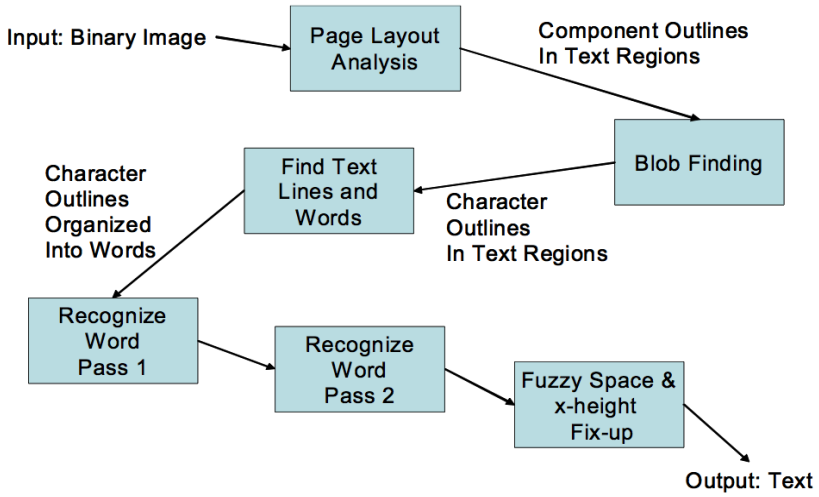


Figure 3, the block diagram of the Tesseract framework [18]

After determining which region outlines could represent characters, they identify the orientation of each line of text using the baseline of these regions. Once these baselines are found, a space detector examines each line to locate the spaces, and after these are defined, the lines are partitioned into words, meaning the space-delimited region groups are combined. The subsequent step involves analysing these grouped words, treating each word individually.

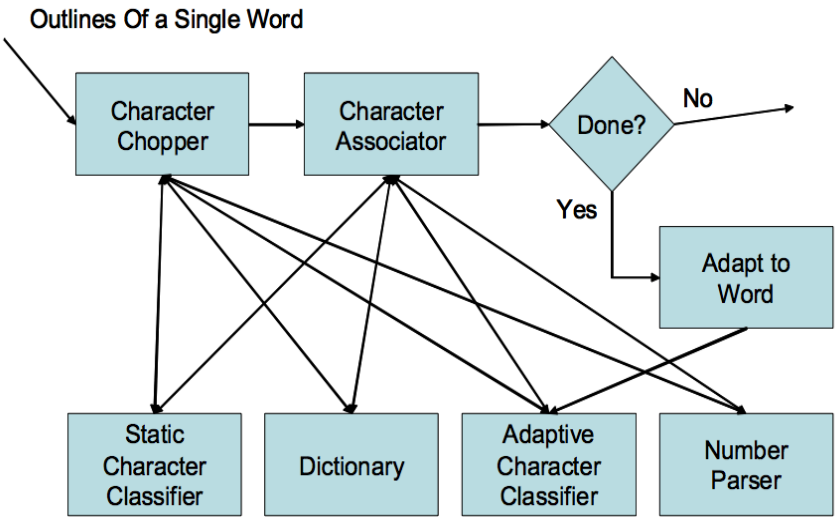


Figure 4, the block diagram of the word recognition process [18]

Typically, each region represents a specific character, so the word recognizer first classifies the potential characters individually. For the classification of characters, their shapes are initially decomposed using polygon approximation, and then a four-dimensional descriptor vector is generated for each segment from the midpoints' x, y coordinates, as well as the segment's direction and length values. These four-dimensional vectors are then clustered to form a prototype of the character. During recognition, polygons of varying dimensions are transformed to a uniform dimension to make them independent of the descriptor vector.

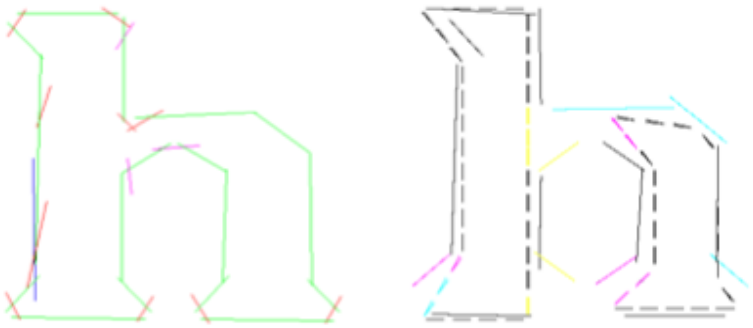


Figure 5, a) the prototype of the letter 'h' constructed from the Times New Roman font, b) a representation of a prototype as a match in relation to the given model [18]

In the a) image, the green lines indicate clustered features, which are considered significant in the comparison step. The blue color indicates that this cluster is formed from the merging of two significant clusters. The magenta color indicates that these segments are not examined because they are already part of an existing cluster. Red indicates that the given cluster is not significant, lacking sufficient samples, and therefore cannot be classified into any cluster.

In the b) image, the short dashed lines represent the characteristics of the match, and the longer lines are the characteristics of the prototype. The colours signify the quality of the match: black is good, magenta is acceptable, cyan is weak, and yellow indicates no match. Vertical matches are generally good, despite the character being significantly incomplete.

Character classification consists of two steps: first, the number of objects to be compared is reduced to 1-10 characters using locality-sensitive hashing, and then distance calculations are performed on this reduced set of values. During the distance examination, the Euclidean distance is calculated between clusters from the x, y, and length values, and this is adjusted with the angle stored in the cluster's fourth dimension, to which a weight is also applied:

$$d_f = d^2 + w\theta^2 \tag{4}$$

This classification calculates the distance from the ideal. To refine this, a so-called certainty measure has been introduced, which is:

$$E_f = \frac{1}{1 + kd_f^2} \tag{5}$$

Where k is a controlling factor for the distance. When a match occurs during feature examination, this E_f factor is transferred to the prototype E_p, since multiple features are compared during the matching process. The collection of good matches occurs independently, and the sum of the features and the certainty of the prototype may differ. The final distance is then normalised by the number of features and the number of prototypes L_p, and then used as a distance measure:

$$d_{final} = 1 - \frac{\sum_f E_f + \sum_p E_p}{N_f + \sum_p L_p} \tag{6}$$

The two-step classification allows third parties to expand our system with prototypes created using multiple fonts for each character. For numeric characters and punctuation, the system uses an encoded dictionary that cannot be expanded. Once each character is identified, a dictionary is used to verify the correctness of a word through context analysis, ensuring the detection was successful. The system allows for the expansion of

this dictionary and also employs adaptive dictionary expansion. The search occurs in a directed acyclic word graph, where each vertex is represented by 8-bit characters. Two such graphs can be distinguished in the system: one for certain words and another for uncertain words.

To ensure the reliability of a newly entered word, two rules are formulated: the shape recognizer must clearly favour one variant of the word over others, meaning it must have significantly greater certainty than the second possible variant. The second rule is that there must not be a best choice for the word's form already existing in the dictionary of certain words.

The use of two graphs significantly increases speed because, after dictionary expansion, a specific word can be quickly found in the dictionary of certain words, eliminating the need for further investigation.

3.5.2 Comparison of Identified Texts with the Samples Stored in Database

When comparing our extracted texts with samples stored in the database, one issue may arise from treating identified and categorizable groups of characters/words as author names, titles, or subtitles. The separation into categories is defined as belonging to a particular line. Since authors' names generally consist of 2 to 4 words (assuming a single author), it is advisable to start the search with such word groups. Additionally, we can assume that document titles usually have larger font sizes, which can also be considered to accelerate the search. If neither characteristic is met, we must perform a match for all the above-listed properties and any continuous text segments found in the image.

The search can be further accelerated by lemmatizing or stemming the words stored in each attribute, and then storing the resulting new words as independent labels. During the search, books containing the most matches should undergo further investigation. In lemmatization, suffixes are removed from the ends of words; in stemming, words are replaced with their stems.

In English, the most commonly used stemmer is the Porter algorithm [20]. The algorithm executes phases sequentially, each phase consisting of groups of operations. The goal is to select, for each phase, those operations that most significantly reduce the length of the word.

It's important to highlight that stemmers are typically language-specific. For example, the Snowball stemmer, which is a modification of the Porter algorithm, includes support for Hungarian. Due to the differing grammatical rule systems in each language, it's essential to use a stemmer that is well-suited to the language of the text being analyzed. To select the appropriate stemmer, we must first determine the language of the text under study.

There are various methods for determining a language. One of the most frequently used is N-gram analysis [22], where words are divided into overlapping segments of length n, and the occurrence probabilities of these segments are then studied. A word consisting of k characters has k + 1 bi-grams, k + 2 tri-grams, and k + 3 4-grams. The probabilities of individual N-grams are determined using a hash table on a tokenized text, with a suggested input of a sample text comprising several tens of thousands of words. The most frequent 300 N-grams correlate well with the respective language, so it is sufficient to store only these. If an N-gram appears in the hash tables of multiple languages, the one with the higher probability is considered the correct result.

Short word analysis can also be applied, using words of 4 or fewer characters for recognition. For classifying extracted keywords/N-grams, one might use methods such as Naive Bayes classification [23], which is based on Bayes' theorem of conditional probability. In this case, the occurrence probability of each word is determined. Another approach might involve using Hidden Markov Models for language recognition.

In a previous university study [24], statistical features were used for language recognition, including the frequencies of word-ending characters, average word length, the ratios of A/E and S/T characters, and the frequency of letter transitions up to changes of 1-2 characters. The method could identify the language of a text with an accuracy of about 95% for texts longer than 1500 characters, and it was significantly faster than N-gram classification. During the measurements, languages such as English, German, French, Spanish, and Hungarian were classified based on their statistics, examining a subset of the English alphabet empirically determined for word-ending characters.

3.6 Presentation of Similar Systems

3.6.1 Book Cover Recognition Project

This system [25] was developed under the supervision of a PhD student at the University of California San Diego. The system uses the SIFT [26] algorithm for extracting descriptor vectors. The algorithm first applies a Gaussian convolution filter to smooth the image, then builds a multi-level pyramid from the resulting image. Difference levels (DoG) are created from each level of the pyramid by subtracting the current Gaussian pyramid level from the levels below it in the following manner:

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) = L(x, y, \sigma) - L(x, y, \sigma) \quad (6)$$

Where σ is the standard deviation of the applied Gaussian function, k is the level being examined, I(x, y) is the input image, *a represents the convolution operator, and D(x, y,σ) is the convolution mask used to create the difference levels.

In the next step, the function D searches for local extrema in the space of difference levels it generates; these points are selected as feature points. The search for these local extrema is conducted at every level, and interpolation is used to calculate between levels. To reduce the number of key points, a Laplacian operation is applied to the collected points as a thresholding measure to remove weak edge points.

The final step involves determining the feature vectors. The algorithm calculates an edge-orientation histogram for each feature point based on the local gradient characteristics. The gradient magnitude and direction are determined as follows:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2} \quad (7)$$

$$\theta(x,y) = arctg\,((L(x,y+1) - L(x,y-1))\,/\,(L(x+1,y) - L(x-1,y))) \quad (8)$$

Where *m(x,y)* is the gradient magnitude at point *x, y,* and θ is its direction. The calculations are performed in a 16x16 pixel neighbourhood around the points, which is further divided into 4x4 pixel windows. For every point in these windows, their gradient values in 8 directions are calculated and from this, a global histogram is created for each window. The descriptor is composed of a 128-dimensional vector from these calculations. The method's advantage is that it is extremely robust, tolerating rotations up to 60 degrees, and since it is based on edge map computations, it also relatively well tolerates issues with lighting. The algorithm is also suitable for real-time operation. There exists a variation of the algorithm based on principal component analysis, which reduces the descriptor vector to 36 dimensions, significantly speeding up the matching of descriptor vectors, though the cost of extracting the descriptor increases.

For recognizing books, the author uses K-Means [27] for unsupervised learning, clustering, or in other words, categorization. In general, clustering attempts to organise given objects into groups using a similarity function, with membership in each group determined by a membership function. We distinguish between hierarchical and partitioning clustering; in the former, groups can contain subgroups. Additionally, clustering can be either top-down, starting from the entire set and dividing some groups at each step, or bottom-up, starting from individual clusters and merging them step by step. The examination of group similarities uses a weighted dendrogram, with the goal to merge those groups that are most similar. This merging process repeats until a so-called termination count is reached, which is a predetermined number of clusters. After each merging, the algorithm recalculates the weight associated with each cluster.

In K-Means, the assumption is that the centroid of each cluster well represents that group. Group classification during evaluation is based on the distance from these centroids. The algorithm first randomly determines K initial elements, then iterates to assign the nearest points found in the vicinity of these K elements. At each step, one point is assigned to a specific group, and after the assignment, it recalculates the centroids of the newly formed groups to modify the similarity function. The algorithm continues iteration until a termination condition is met, such as when all points belong to a group, a specific number of iterations is reached, the partitions no longer change, or the centroids' positions remain stable. The final result will be K well-separated clusters.

This algorithm is excellently suited for pattern matching and can also be used for segmenting images based on colour intensity if a good similarity function can be defined among the colours of the image being examined. Among others, the SIMPLYcity [14] system also uses this algorithm during image segmentation, employing the LUV colour space in the process.

3.6.2 Book Cover Recognition project by Wang

This system [28] was developed by a student at Columbia University. In this system, during preprocessing, the creator applies bilateral filtering, Canny edge detection [39], Hough transform, and threshold-based cropping. The descriptors are extracted from feature points using the previously mentioned SIFT algorithm, and for recognition, it employs a linear Support Vector Machine (SVM) [29]. The database stores 15 samples for each book and uses 5-fold cross-validation as a parameter. The creator claims that the system operates with an efficiency of around 99%.

The SVM, or support vector machine, strives to maximize the margin between the separating hyperplanes for the different classes. The decision function is determined by a subset of the training data, known as support vectors. This is one of the most effective classification methods; during classification, it only considers those training data that are near the decision boundaries. If the algorithm cannot find a clearly definable separating hyperplane, it expands the space through a nonlinear transformation, introduces a new dimension, and then establishes new class boundaries. However, it is important to note that SVM can be unstable with small training data sets in terms of finding optimal hyperplanes.

3.6.3 BookSpineOCR - Hybrid Approach to Mobile Book Spine Recognition

This system [30] combines image features and optical character recognition to achieve higher accuracy during evaluation. It is primarily developed for recognizing book spines, using Maximally Stable Extremal Regions [31] (MSER) and Stroke Width Transform [32] (SWT) for localizing letters. Segmentation is edge-based, using a Canny edge detector to crop the letters found by MSER, and then the Tesseract framework is

used to evaluate the results. The dictionary required by Tesseract is prepared from the texts of documents stored in the database.

To accelerate the search, a database *(W)* is constructed from the words in the dictionary, which stores identifiers of documents that contain these words in their metadata. Thus, for each word $w_i \in W$, this $L(w_i)$ indicator is determined. During a search, for the recognized word $q_j \in Q$, the word m_j in the dictionary that matches fully or is the nearest neighbour is selected. The quality of each k-th match is determined as follows:

$$s_t(k) = \sum_j I(k \in L(m_j)), \tag{9}$$

where $I(\,)$ is the indicator function, valued at 1 if $L(m_j)$ includes document *k* and zero otherwise. The quality of each document is then weighted by term frequency and inverse document frequency. The term frequency is calculated by the number of occurrences of the word in the document divided by the total number of words in the document, calculated using the following formula:

$$tf(t,\ d) = 0.5 + \frac{0.5 \times f(t, d)}{max\{f(w, d) : w \in d\}} \tag{10}$$

where $f(t,\ d)$ indicates the number of occurrences of word *t* in document *d*. The inverse document frequency (idf) can be calculated based on the rarity of the term across the entire corpus, using the following method:

$$idf_i = 1 / log(n / df_i) \tag{11}$$

where df_i is the number of documents that contain the term *i*. The weight is then given by the scalar product of these two values, term frequency (tf) and inverse document frequency (idf).

The colour-based features of book spines are extracted using the SURF [33] algorithm, which builds upon the SIFT algorithm's approach but promises better accuracy and robustness. To accelerate the search, a Vocabulary Tree [34] is used.

3.7 Rating

In this section, I have outlined the general approaches to book cover recognition, including the tasks of the preparation phase, various methods for extracting descriptors,

and the issues related to matching extracted features. I have explained the general approaches to building image databases, multiple methods to accelerate searches, problems encountered during the search, and potential solutions. I have detailed the tasks of optical character recognition, its possible applications, and related problems that need to be solved. I presented three similar systems and the algorithms used in them.

Since speed is an important aspect of the system's operation, I intend to perform book cover classification in two steps, similar to SIMPLYcity [14]. In the system, I plan to perform a color-layout based pre-selection in the LAB color space. Since the L channel in the LAB color space only controls the brightness of the color defined by the A-B channels, it can be omitted from the examination, thus reducing the dimensionality of the vectors to be examined during pre-selection.

As the system is not only intended to display matches based on similarity, as in the color histogram example, but also exact matches, the latter method is not feasible due to its computational complexity. Instead, I plan to use a feature point-based descriptor, specifically a SURF descriptor based on Haar wavelet responses, which is very similar to SIFT.

For classifying descriptor vectors in this case, SVM would not offer an efficient solution due to its computational demand and the amount of training data. Since I only intend to store one specific book cover per book, the K-Nearest Neighbor [35] classification seems most appropriate. This algorithm allows me to perform the training phase at runtime, significantly reducing memory usage.

To correct misclassifications, I plan to use the Tesseract framework, for which I intend to build a dictionary by decomposing the metadata of each book. To evaluate the quality of the matches, I plan to implement a solution similar to the one used in BookSpineOCR.

4. System Design

As previously explained, the system is fundamentally a catalog application based on web technologies, focusing on the identification process based on information extracted from the real world. The system consists of a client developed for the iOS platform, a REST API built on NodeJS and communicating over HTTP, and a subsystem written in C++ for extracting the information. The database is a custom implementation, and data is stored in a schema-less JSON format.

4.1 Architecture

The system architecture is divided into four well-defined parts: iOS client, API, Identification and Recognition subsystem, and Database. I based the design on the Three-tier architectural pattern. The descriptors are stored in the filesystem, with only their identifiers and related information stored in the database. The Identification and Recognition subsystem uses the filesystem to load descriptors, and the descriptors extracted from uploaded images are also stored on the filesystem. Beyond handling basic queries, the API also acts as an intermediary layer in the identification process.

The structure of the architecture is illustrated by the following diagram, where arrows represent the communication channels between the subsystems:

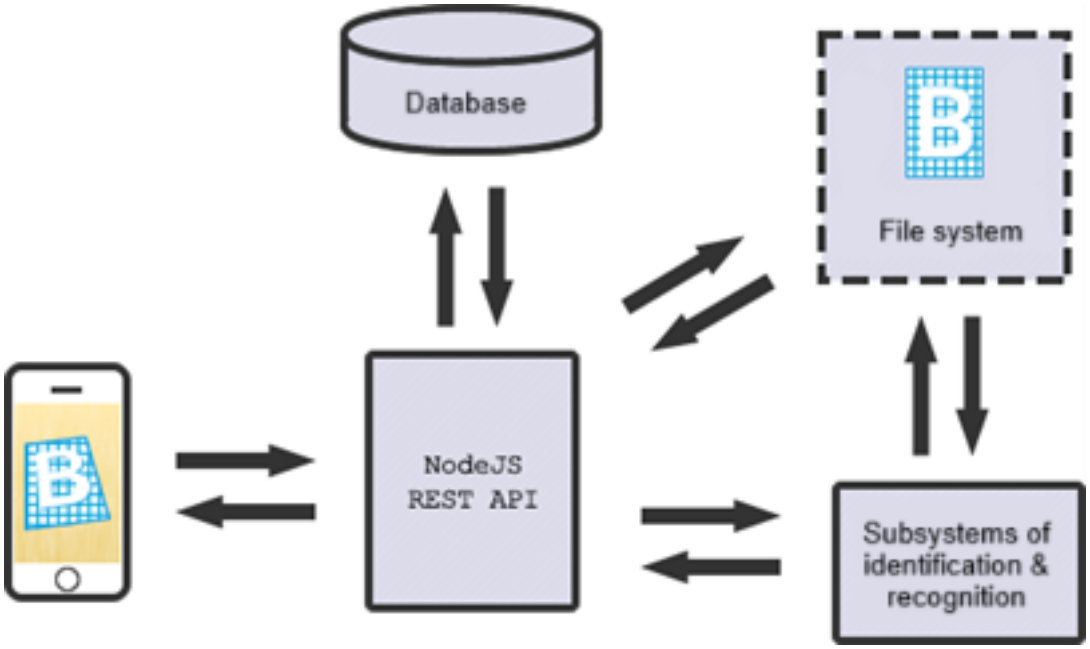


Figure 6, system architecture

4.2 Subsystem of Identification

The task of the identification subsystem is to process the photo taken by the user, extracting descriptors necessary for object identification. The input is an RGB image, and the output is an N-dimensional vector containing M descriptor vectors per dimension.

4.2.1 Pre-processing

The task of preprocessing is to transform the book cover located in real space into a processable dataset that enables the extraction of high-quality descriptors. Preprocessing consists of several parts: noise removal, edge detection, segmentation, and plane transformation.

4.2.1.1 Median Filtering

Median filtering is a type of nonlinear filtering. In mathematics, operations sequences are called linear if they satisfy the mathematical properties of concatenation and conjunction.

During the filtering process, a 3x3 mask is moved across the image. The mask takes on the intensity values of the specific pixel and its surroundings, and the new pixel intensity is assigned the middle element of these values after they are sorted. Median filters are exceptionally good at removing noise with outlier intensity values, as these values are positioned at the beginning or end of the sorted sequence. The performance of the algorithm can be enhanced by modifying the sorting algorithm used so that the sorting only continues until the fifth element of the result set is found.

4.2.1.2 Morphological Filtering

Mathematical morphology is an operation defined on any subset of a d-dimensional Euclidean vector space, where X and Y are subsets of the set V, with V = Z2 in the case of two-dimensional images. Two fundamental operations [36] are distinguished: dilation and erosion. From these two basic operations, additional operations such as opening, closing, and morphological filtering can be described. These operations are generally performed on binary images using a structuring element. During erosion, the edges of the image segments are reduced, consuming the image. As a result, the size of the segments decreases, and points previously appearing as noise disappear. During dilation, the segments are expanded, which decreases or possibly eliminates the size of holes within the segments. Morphological opening is achieved by performing erosion followed by dilation, while closing is the reverse sequence. The operations of opening and closing are idempotent; repeating them consecutively does not alter the image. Morphological filtering means performing opening and closing successively, which results in the removal of noise and holes, and smooths the contours of the segments.

4.2.1.3 Kuwahara-Nagao filtering

The Kuwahara-Nagao [37] filter is also a type of nonlinear filtering. The algorithm operates with a 3x3 mask, within which four overlapping 2x2 windows are defined. The

result is that the average intensity value of the window with the minimal variance, i.e., the most homogeneous window, becomes the new intensity value of the central pixel.

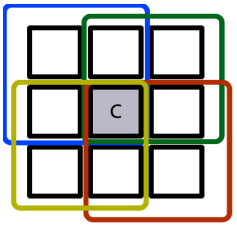


Figure 7, the filter mask of the Kuwahara-Nagao algorithm with overlapping windows

The algorithm is primarily recommended for processing color images, and the resulting image takes on characteristics similar to those seen with bilateral filtering, where homogeneous regions blend together, yet edges remain well connected. Compared to bilateral filtering, significant differences are observed in terms of speed and noise tolerance. [38]

Figure 8, a) original image b) Kuwahara-Nagao c) bilateral filtering [38]

Figure 9, a) original image b) Kuwahara-Nagao c) bilateral filtering [38]

Overall, this filtering alone can replace the steps of median and morphological filtering.

4.2.1.4 Symmetric Nearest Neighbour

To be able to use book-shaped objects located in real space for further processing, it is necessary to be able to cut them out as contiguous segments from our image. One possible approach to determining these segments is edge detection, where the images

are derived in a discrete domain, in other words, their gradient image is produced. The Symmetric Nearest Neighbour [37] algorithm, also known as differential edge detection, serves as a solution for this task and, like the median filter, is a type of nonlinear filtering. The algorithm operates with a 3x3 mask, where the maximum pixel intensity difference of the diagonal components determines the new pixel intensity value. The advantage of the algorithm is that it produces well-connected edges everywhere and eliminates errors in homogeneous regions.

The mask:	The new pixel intensity value
P1 P2 P3 P8 *X* P4 P7 P6 P5	$X = \max(\lvert P1-P5\rvert, \lvert P2-P6\rvert, \lvert P3-P7\rvert, \lvert P4-P8\rvert)$ (Chebyshev distance)

However, the disadvantage of the algorithm is that some edges appear thickened if the gradient magnitude of the surrounding edge pixels also exceeds the threshold value, which is referred to as a multi-pixel response.

4.2.1.5 Canny edge detection

Canny edge detection [39] attempts to address the issue of multi-pixel responses encountered in the previous method. A multi-pixel response occurs when the intensity differences at certain points in the image also appear in neighboring pixels, resulting in the boundary lines of objects appearing thick. This algorithm strives for more precise localization of edges by selecting the local maxima at each gradient point. It first prepares the image with Gaussian filtering, then after localizing the gradient points, it eliminates the resulting noise with hysteresis thresholding. The final step is based on the assumption that significant edges are continuously located, and their average intensity values converge. Due to its high computational demand, it is not always applicable in real-time.

4.2.1.6 Hough transform

The Hough Transform [40] starts from the equation of a line:

$$y = mx + c \tag{12}$$

where y is the vertical component of the line, m is the slope, x is the horizontal component, and c is the constant by which the line is shifted vertically. By transforming this equation, we obtain the following relationship:

$$c = (-x)m + y \tag{13}$$

This results in the spatial representation of the line in the *M*-*C* space. This equation is nothing but a transformation of a point in *X*-*Y* space to a representation as a line in *M*-*C*

space. Where these lines intersect in *M-C* space (*M', C'*), there is a corresponding line in *X-Y* space. The mapping is illustrated in the following diagram:

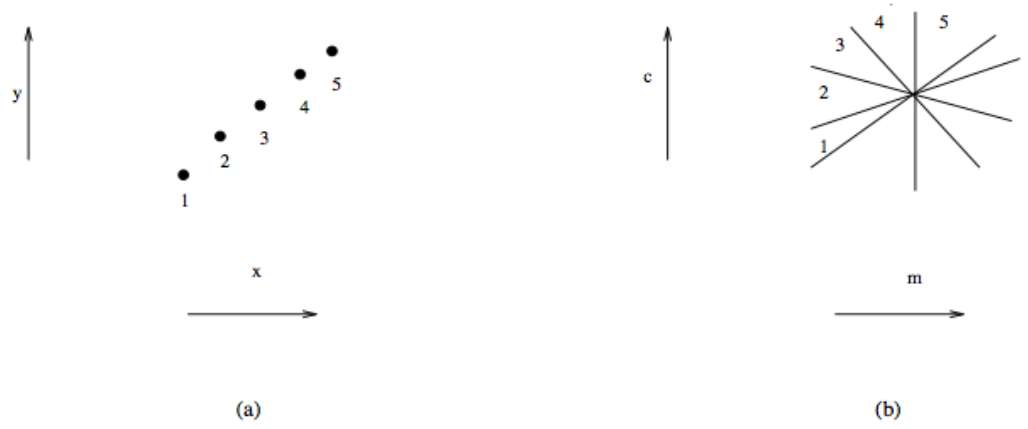


Figure 10, a) the points of a line in X-Y space, b) the representation of these points as lines in M-C space, with the actual line in X-Y space determined by the intersection of these projections [36]

The problem arises only where we project the points of a line onto the M-C parameter space when the original line is parallel to the Y-axis, because in this case we are talking about an infinite slope. To avoid this error, the following modified equation can be used:

$$p = x \cos\theta + y \sin\theta \qquad (14)$$

Where θ is the angle that the line p makes with the x-axis. In this relationship, the initial values of p and θ are infinite, but θ can be directly calculated from the gradient directions produced during edge detection.

In the algorithm, the first step is to establish the parameter space with initial values set to zero. In the second step, for every edge point of the pre-detected image, we need to determine the p and θ values from the previous equation, which allows us to update the parameter space: P[θ,p]=P[θ,p]+1. The final step is to collect the local maxima of the parameter space, which will represent our lines in the XY space.

4.2.1.7 Otsu's binarization

An excellent algorithm for cutting out book-shaped objects located in real space as contiguous segments is Otsu's binarization [41], especially when a significant intensity difference between the object under examination and the background can be detected. The algorithm is based on the premise that the variance within homogeneous regions is small. In the first step, the algorithm generates an intensity histogram from our input grayscale image. In the second step, it normalises this vector, i.e., it produces a new histogram containing the probability of occurrence for each pixel intensity value. In the

third step, iterating through this new histogram, it determines the threshold that maximises the variance between the foreground and the background.

4.2.1.8 Correction of Otsu's binarization

The Otsu algorithm does not always provide satisfactory solutions; the result depends on the average intensity differences between the foreground and the background. In this case, the foreground represents our object. A possible improvement could involve using a cropped sub-image from the centre of our image to determine its average intensity values and variance, channel by channel. Since it is required that the detectable book cover must be located in the central region of the image, this provides us with a result vector that can characterise the specific book. In the next step, by using a threshold value, we can examine these values. If the variance in any channel exceeds our threshold, using that channel as grayscale can improve our segmentation result from the Otsu algorithm, as that channel better characterises the object to be cut out than a grayscale image derived from the average of the three channels.

4.2.2 Estimating axes with principal component analysis

Principal component analysis [42] can be defined as a type of regression algorithm within classification methods. The goal of regression is to model the underlying characteristics in the data. During principal component analysis, the number of dimensions of our sample is reduced through a transformation in space.

The first step in the algorithm involves normalising the input vectors, which is done by subtracting the mean value of each dimension from the points in that dimension. The next step is to determine a covariance matrix among the normalised dimension vectors. Covariance measures the dependency between two different vectors, indicating their joint variability. For a given set of N-dimensional vectors, the number of covariance matrices to be calculated is (N * (N-1) / 2). The method of calculation is as follows:

$$\mathrm{Cov}(X,Y) = \frac{1}{N-1}\sum_{i=1}^{N}(X_i - \overline{X})(Y_i - \overline{Y}) \tag{15}$$

where N is the length of the given vectors, it is easily understood that the covariance of a vector X with itself yields the variance of the vector. The construction of the covariance matrix occurs as follows:

$$C^{n\times n} = (c_{i,j},\ c_{i,j} = Cov(Dim_i,\ Dim_j)) \tag{16}$$

Where n is the number of dimensions of the input vectors.

The next step involves calculating the eigenvalues and eigenvectors of the resultant covariance matrix, as well as sorting these eigenvectors according to their eigenvalues. The list of components thus formed characterises our input dataset well, and the eigenvalues associated with these components determine their significance. By discarding vectors below a certain threshold in this sorted list, we can reduce the number of descriptive dimensions derived from our input dataset. While this involves some loss of data, considering the significance ensures that the resulting dataset with reduced dimensions closely approximates the original and may also lead to significant computational performance gains in certain cases. This algorithm is widely used, from pattern recognition to compression. In this task, its property of using the first and second principal components calculated from an input set of points containing image coordinates is utilised. This determines the two axes of the point set, allowing - depending on the orientation of the object - modifications to the model used during the mapping before the object is cropped.

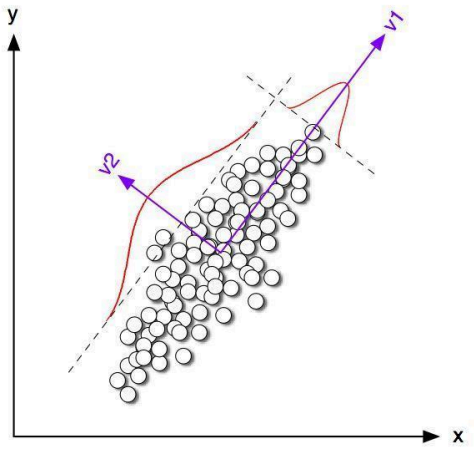


Figure 11, results of principal component analysis for two-dimensional input vectors [47]

4.2.3 Feature point detection

In the system, the descriptors for book covers are calculated using the feature points detected on them. There are numerous algorithms available for finding these points, one of which is the Harris [48] operator.

In the case of the Harris [48] operator, the first step is to approximate the derivatives of the image at every point, essentially performing edge detection on the image. It is advisable to first smooth the image with a Gaussian or averaging filter to minimise environmental noise. The next step involves calculating a gradient covariance matrix in a specified 2n+1 by 2n+1 neighbourhood around each point using the following formula:

$$M_H = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \tag{17}$$

where I_x is the gradient magnitude in the x-direction and I_y is the gradient magnitude in the y-direction. The eigenvectors of the covariance matrix indicate the direction of edges, and its eigenvalues indicate the magnitude of edges. If both eigenvalues are sufficiently large, we store it as a potential corner point. It is advisable to use a sorted list for storage.

In the next step, the algorithm iterates through the curated list and systematically eliminates those points that are proximate to a more prominently detected point. In essence, this involves selecting the local maxima within the vicinity of detected points to ensure the retention of the most significant features.The final step in the methodology involves the application of a thresholding procedure to the image, which encompasses the retained points.

4.2.4 Descriptor Extraction

During the extraction of descriptors, it is crucial for both the feature detector and the descriptor to be invariant to rotation, translation, and scaling. Most detectors are designed based on these principles; however, the Harris detector encounters issues with scaling. To mitigate this, it is advisable to perform point detection across the Gaussian pyramids of the image, as more modern methods automatically identify the scaling factor. It is essential that the descriptors themselves remain invariant, since the matching process relies on the analysis of these descriptors.

4.2.4.1 SURF Descriptor

In the system, I employ the SURF [33] descriptor, which offers greater accuracy, robustness, and repeatability compared to the SIFT descriptor. This descriptor is invariant to rotation, scaling, and translation. Feature point detection is based on the Hessian matrix, with the search conducted using second-order partial derivatives calculated on the integrated image. Unlike SIFT, which detects features across Gaussian pyramids, here a mask of increasing size is moved across the integrated image. This approach yields results similar to those of SIFT in each iteration, but since a two-dimensional lookup table can be used with the integrated image, it is highly parallelizable. The mask sizes grow from 9x9 to 15x15, 21x21, and up to 27x27. Before computing the descriptor, the point's orientation is determined using a Haar wavelet transformation within a 6s radius environment, where 's' is the scale factor at which the

point is located. The descriptor is then calculated within a 20s radius around the point, which is further divided into 4x4 blocks, and the x and y directional Haar wavelet responses are computed for each block. Each block's descriptor vector is as follows:

$$\mathbf{v} = (\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|) \tag{18}$$

Where dx is the wavelet response in the x-direction, and dy is in the y-direction. Combining these responses ultimately yields a 64-dimensional vector. As the wavelet transformations are invariant to lighting and the descriptor is represented as a unit vector, contrast variations are well managed, and the method also exhibits better noise resilience.

Overall, it can be stated that, compared to SIFT, this method allows for significantly faster detection and feature matching, produces robust features, and reduces the descriptor dimensionality by half.

4.3 Subsystem of Database

Given that the database may contain several thousand book covers and their associated metadata, it is necessary to perform some form of pre-selection. In the system, I utilize a color-layout based approach to reduce the size of the value set needed for classification, thereby dividing the classification problem into two parts: pre-selection and recognition.

4.3.1 Clustering Based on LAB Colour Distance, with Adaptive Sampling

For pre-selection, I compare colour-based features. To do this, each stored book cover is divided into six segments, and then I calculate the average LAB colour values for these segments. I apply a 40-pixel inset from the edge of every book cover to minimise the distortion at the edges of the samples during the transformations performed at the end of preprocessing. To convert RGB colours to LAB, they must first be converted to the XYZ colour space as follows:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \frac{1}{b_{21}} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} = \frac{1}{0.17697} \begin{bmatrix} 0.49 & 0.31 & 0.20 \\ 0.17697 & 0.81240 & 0.01063 \\ 0.00 & 0.01 & 0.99 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{19}$$

Following this, the conversion to the LAB space takes place, which involves normalising the colour space using two constants [43].

Since the L channel in the LAB colour space only stores the brightness of the colour, it can be ignored during analysis. Therefore, I only use six two-dimensional vectors to

reduce the value set. In the system, the ten books that are vectorially closest are selected and then forwarded to the recognition subsystem. The mask used in the process is illustrated in the following figure:

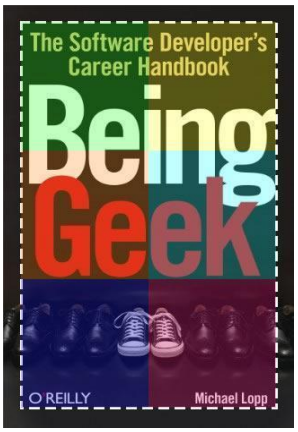


Figure 12, the coloured rectangles represent each region examined

4.4 Subsystem of Recognition

In this subsystem, the descriptor vector extracted is matched with the descriptor vectors stored in the database. The number of books to be matched is only a subset of the total number of books stored in the database, as a result of the clustering performed in the previous step. During the matching process, a linear search is conducted, and the sample with the best fit is selected as a match, provided it is deemed acceptable according to a predefined threshold.

4.4.1 K-nearest Neighbour Classification

The K-nearest Neighbour (KNN) is known as a lazy learning classifier, where the model building process is omitted. The algorithm is based on the assumption that objects with similar properties are alike, and it typically uses Euclidean distance for numerical data or Levenshtein distance for text data to evaluate this similarity. Since it skips the model building phase, there is no learning step involved; it merely requires the storage of our descriptors.

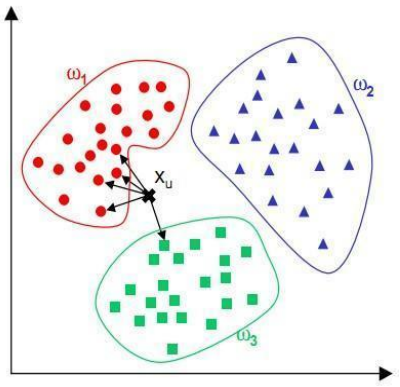


Figure 13, kNN, X_u is in the red category [46]

In the prediction phase, the algorithm finds the k nearest neighbours and makes a selection by majority voting. The winning label is the one that receives the most votes, i.e., it considers the set of the k closest documents to document d (Nk), examines their classes, and selects the class with the most documents belonging to it. This algorithm can also be used for regression, where it provides the average of the samples, and for classification, where it gives the relative frequency. Because of this characteristic, it is also called a universal approximator: with an infinite number of training points, it can approximate any function arbitrarily well.

However, the algorithm is sensitive to independent attributes, so care should be taken to classify these separately. Another issue is its sensitivity to different scales, meaning the algorithm is not scale invariant; thus, normalisation or importance-weighted scaling should be applied. Choosing the number k can also be problematic; if too low or too high a number is chosen, it can affect the quality of the classification.

Therefore, it can be said that the classification time for kNN scales linearly with the size of the training data, but it scales well as it does not depend on the number of classes and does not require prior training for each class. With many training data points, feature selection is not necessary, though the classes can influence each other. Since I am performing classification among SURF descriptors, neither problem is present in this case.

4.4.2 OCR-based retrieval

When sufficient image descriptors are available, or to handle inaccuracies resulting from misclassification, I intend to implement an optical character recognition-based correction. This requires the presence of a hash table that stores individual keywords and the set of documents containing the keyword. I plan to weigh each keyword using

term frequency and inverse document frequency, which are commonly used in the BookSpineOCR system and other search systems in general.

It is important to note that the OCR step does not always yield accurate results, thus it is necessary to conduct a word match analysis that can handle errors resulting from 1-2 misclassified characters. An ideal solution is to use Levenshtein distance to filter out identified words, allowing for 2-3 permutations.

4.5 API

This subsystem is responsible for interfacing the communication between the client application, the database, and the image processing subsystems written in native languages. One of the major advantages of NodeJS is that its Google V8 JavaScript engine's Just In Time compiler provides speeds nearly as fast as if the module were written in a native language, while also offering platform independence, easy scalability, and the benefits of dynamic languages. Additionally, it provides the advantages of non-blocking I/O and the possibility of asynchronous programming.

The sole task of the API is to process incoming images and control the search in the database. The process used during the search is illustrated in Appendix 1.

4.6 Client

The client is an application with two tabs and two menu items: Favourites and Search. Upon starting the application, the user accesses the Search interface. Books saved to favourites are stored in a persistent SQLite database managed by the iOS SDK’s Core Data Object Graph Manager framework.

4.6.1 Search

On this interface, the user can take a photo of a specific book and submit it to the API for recognition. If the recognition is successful, the user is directed to the "book view" screen, where they can see the book’s cover, title, author, publisher, and have the option to add the book to their favourites.

4.6.2 Favourites

This is a Table View list where the user can view a list of previously detected and saved favourite books. After selecting an entry, the user is taken to the "book view screen" used during the Search, where they can view the stored information about the book, remove it from the bookmarks, and manually expand the list. When a new document is uploaded, the API takes care of storing its metadata, extracting the LAB cluster vector, and saving the cover descriptors to the file system.

5. Implementation

The system blends dynamic and native languages, with communication happening over HTTP protocol and standard output. The modules are broken down into distinct components. For the API, this means four components; for the identification and recognition subsystems, there are three classes, one execution unit, and one separate program module.

During the implementation, I used a version control system called GIT. The native modules were developed in the Xcode IDE environment, while the API was created in Sublime Text 2, a modular text editor.

5.1 Subsystems of identification and recognition

These two subsystems were implemented in a C++ module. Within the module, I differentiate four basic functions, which can be accessed by calling the program with the appropriate parameters. The parameterized calls for these four functions are as follows:

1. -ocr image-path
2. -cluster image-path
3. -recognize image-path [[possible-book-identifier] ...]
4. -store image-path new-identifier

Additionally, within the subsystem, I placed a program implementing Stroke Width Transformation in the swt library, which expects a processed image as the first parameter and the output image path as the second.

5.1.1 Optical character recognition

For this module to perform proper quality character recognition, it takes an input image that has already undergone Stroke Width Transformation. The Wrapper implemented in the API follows this principle, passing the cropped book cover to the program performing the SWT, and then performing character recognition afterwards. Finally, the character recognition is carried out by the Tesseract framework.

5.1.2 Clustering

The first step of this function is to locate the region representing the cover in the image, crop it, and then transform it spatially. Following this, the cropped book cover is divided into regions, converting the RGB colours of each region to the LAB colour space. Finally, it forms a 3*2 set of two-dimensional vectors from the average A-B values and output them to the standard output. The implementation of this algorithm can be found in the Preprocessor class's implementation file. The cluster vectors stored in the system are normalised to a range of 0 to 255 per dimension, and stored in floating-point format.

5.1.3 Recognition

For recognition, I use the previously cropped covers, performing point detection and descriptor calculation on these covers. The matching of images is conducted with a labelled, distance-thresholded kNN classification, taking the two nearest descriptor points for each descriptor. The acceptability threshold for matched descriptors is given by the ratio of the distances to the first and second nearest points, which is 0.7 in the system. If the ratio is met, a given match is added to the list of good matches. When the function is called, starting from the third parameter, identifiers of covers previously judged as proximate are passed, and using these identifiers, the previously saved descriptors associated with the covers are read.

5.1.4 Storage

After insertion into the database, this function's task is to extract the feature points and descriptors from the image passed as the second parameter, then create the corresponding XML file for the identifier and save the data. After saving, it calculates the vector associated with the cover using the procedure applied in clustering and outputs it to the standard output.

5.2 API

The API's role is to control the identification process and manage communication between the database operations and the native subsystems. The API is built on the Express framework. This framework implements what is known as a middleware design pattern, organising incoming requests and responses into a pipeline, thus allowing for data transformations or condition checks before and after routing. The middleware components are very similar to the decorators found in Python.

I use a middleware that augments the response object with a function named sendJSON. This function converts the provided object into JSON format and sets the content type in the response header to 'application/json'.

I handle the endpoints in a separate component, each with a dedicated function responsible for processing request data and assembling the response content. In the API, I distinguish three endpoints:

1. /books
2. /books/{book-id} (integer)
3. /books/recognize

5.2.1 Query and Insertion

The first endpoint's task, upon receiving a GET request, is to display a list of books stored in the database. In the case of a POST request, it expands the database based on

incoming parameters. The POST request expects Multipart content, with the cover parameter containing the image of the book cover, and the title, keywords, author, and publisher parameters expected to be of string type.

When inserting a new book, the uploaded book cover image is first moved to the public/images directory within the API library, followed by the compilation of a list of keywords associated with the book through metadata aggregation. If such a keyword does not exist in the globally applied hash table, it is inserted; otherwise, the new book's identifier is added to the list of book identifiers associated with that keyword.

Subsequently, the extraction of the LAB cluster vector related to the image is performed through an image-processing wrapper serving the native layer, followed by the storage of these vectors in the record representing the book. During saving, feature point detection and descriptor vector calculations are also performed, and these are stored in the api/storage library in XML format.

5.2.2 Search

In the case of book identification, the API performs clustering of the book by invoking the appropriate function of the native layer. It then performs a pre-selection by calculating the vectorial distance between the obtained vector and the vectors stored in the database. Subsequent to this pre-selection, the native layer's identification function is called with the identifiers of the books deemed good.

If the identification is unsuccessful, i.e., the confidence level of the sample considered as a match does not meet the globally applied threshold, a further search is conducted with the words identified on the document. The characters to be recognized are first collected from the image using Stroke Width Transformation [32], and the resulting image, containing grey characters on a white background, is forwarded to the function implementing character detection.

For each book stored in the database, I maintain a keyword vector, which is compiled using the book's metadata and other words found on the cover. Additionally, I store a hash table for every keyword, which contains the identifiers of documents that include the specific keyword.

For a given book, the identification of words is performed by comparing them with the contents of this hash table, and any detection errors are corrected using Levenshtein distance. For each book, I calculate a cumulative goodness score using the inverse document frequency and term frequency of the identified keywords. As an enhancement, I employ a “stop word list” to ensure that a keyword is only counted once in the goodness variable representing a given book.

5.3 Database

In the database, data is stored in JSON format, kept in memory during runtime, and otherwise on the file system. I wrote the database engine in JavaScript, as handling JSON objects seemed most straightforward this way. The database essentially employs a weak schema; the array forming the keywords at each book's record can be dynamically expanded, and I do not use strong relations. Given that the data for a book is compiled in two steps (metadata and cluster vector), this approach is ideal. In the database, I store both book objects and key-value pairs related to keywords. Consequently, the engine has five functions:

- Insert
- Key-insert
- Update
- Next-Id
- Commit
- Rollback

The commit function specifically handles saving the currently in-memory state to the disk, and the next-id function returns the current value of the sequence applied to book identifiers.

5.4 iOS Client

In the client implementation, fundamentally, I distinguished two groups according to the menu items. In the application implementation, the first entry point is a TabBarController, which contains two additional NavigationControllers. These two NavigationControllers converge in a common DetailViewController, which is responsible for displaying the metadata and cover of each book.

To obscure the data connection and the persistent storage framework CoreData, I implemented two singleton classes, APIAgent and StorageManager. The taking of photos was modified using an open-source component named YCameraViewController, and its display occurs modally. The resulting UIImage object is returned to the class displaying it, SearchViewController, via a delegated method, which then sends this to the API and initiates the recognition process.

To facilitate development, I used three additional libraries, one of which is AFNetworking. This library provides a more developer-friendly layer on top of the base HTTP communication classes of the iOS SDK. It includes functions named AFJSONRequestSerializer and AFJSONResponseSerializer, which greatly facilitate handling data returned from the API by performing an automatic conversion to NSDictionary-type objects.

Another library is MagicalRecords, which layers on top of the core functionality of CoreData and also serves to ease the developer's workload. Using it significantly reduces the creation and saving of NSManagedObject instances representing individual entities to the persistent storage. The StorageManager class uses this for setting up the CoreData stack, as well as for saving data to the persistent storage when the application terminates or goes into the background.

Depending on which branch leads to the DetailViewController, a new button appears that offers the user the option to save the recognized book to their favorites. To do this, I place the entity created as an instance of the Book class into what is known as a child context, which will temporarily hold the book. If the user decides to add the created object to the list of favourites, it is then written to the parent context set as the main thread-running context.

In the favourites view, there is also the possibility to expand the database with a new book. For this, I implemented a class named AddNewBookViewController, whose display occurs modally, and the data entered in the form fields are returned to the displaying MasterViewController through a delegated method. In this view, I implemented the UIImagePickerControllerDelegate, which allows the user to browse for the cover image for the book. After addition, the MasterViewController performs the book upload through a request sent via the APIAgent.

The screens of the application and their usage are detailed in section 9.5 of the appendix.

6. TESTING

I conducted the tests on a virtual server equipped with Ubuntu 12.04 operating system (Linux kernel version: 3.2.24). The testing occurred in three steps. In the first step, I tested the clustering results on a set of 10 well-segmentable images that I created myself. In the second step, I performed SWT transformation, optical character recognition, and then conducted searches using the extracted keywords on the same sample. In the third step, I used a database containing 100 book covers [44] and 100 photos taken with an iPhone 3GS camera to test the robustness of the image descriptors and the quality of the classification. Some results of the early prototype tests can be seen in section 2 of the appendix.

For evaluation, I calculated IR [45] metrics (precision/recall) for each book cover sample.

The results are summarised in a 2x2 table per measurement, where the first row contains the results accepted as hits during the measurement, and the second row contains the discarded hits, in the following format:

	True	False
Hit	Number of correct hits (TP)	Number of incorrect hits (FP)
Miss	Missing correct hits (FN)	Missing incorrect hits (TN)

Recall is given by the following formula:

$$P = \frac{TP}{TP+FN}$$

Where TP is the true positive, FN are the false negatives. Recall, on the other hand is given by:

$$R = \frac{TP}{TP+FP}$$

Where FP is the false positive. Typically, these two metrics move inversely to each other.

6.1 Results

6.1.1 Clustering in LAB colour space

During the measurement, I distinguished between two types of distance metrics:

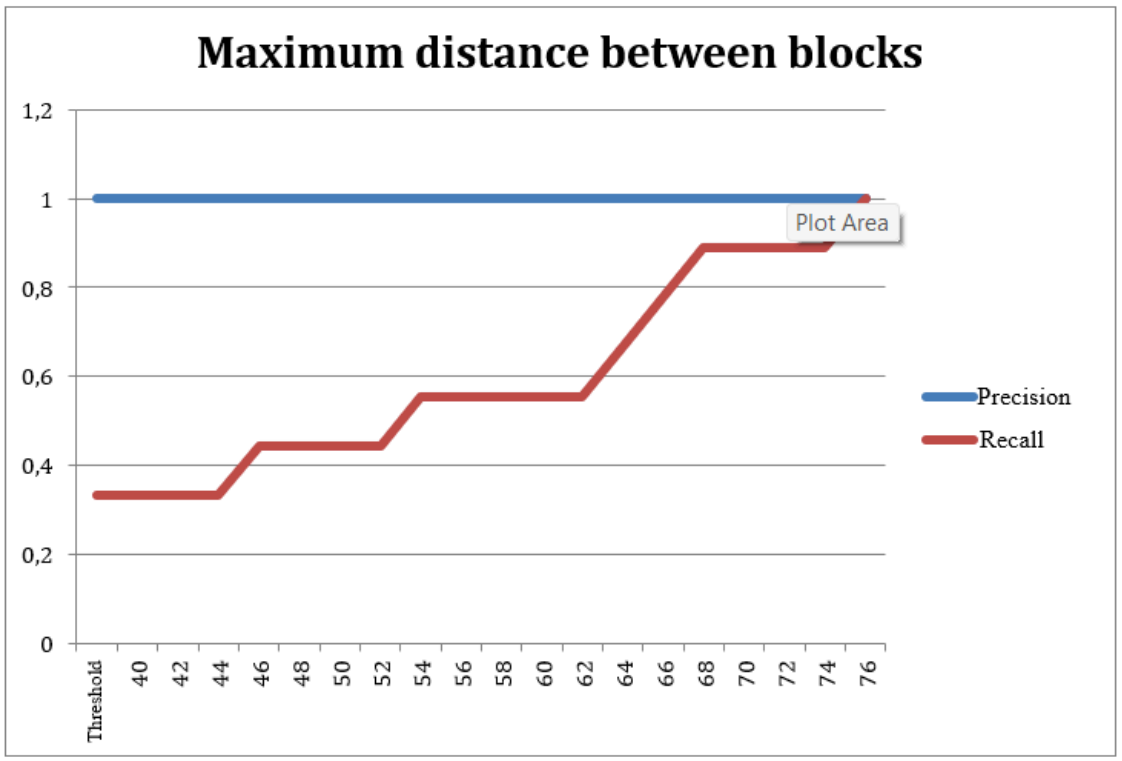
$$D_{Chebyshev}(b,\ s) = \ \ max\{ f(b_i, s_i) : i \in \{1,\ 2,\ 3,\ 4,\ 5,\ 6\}\}$$

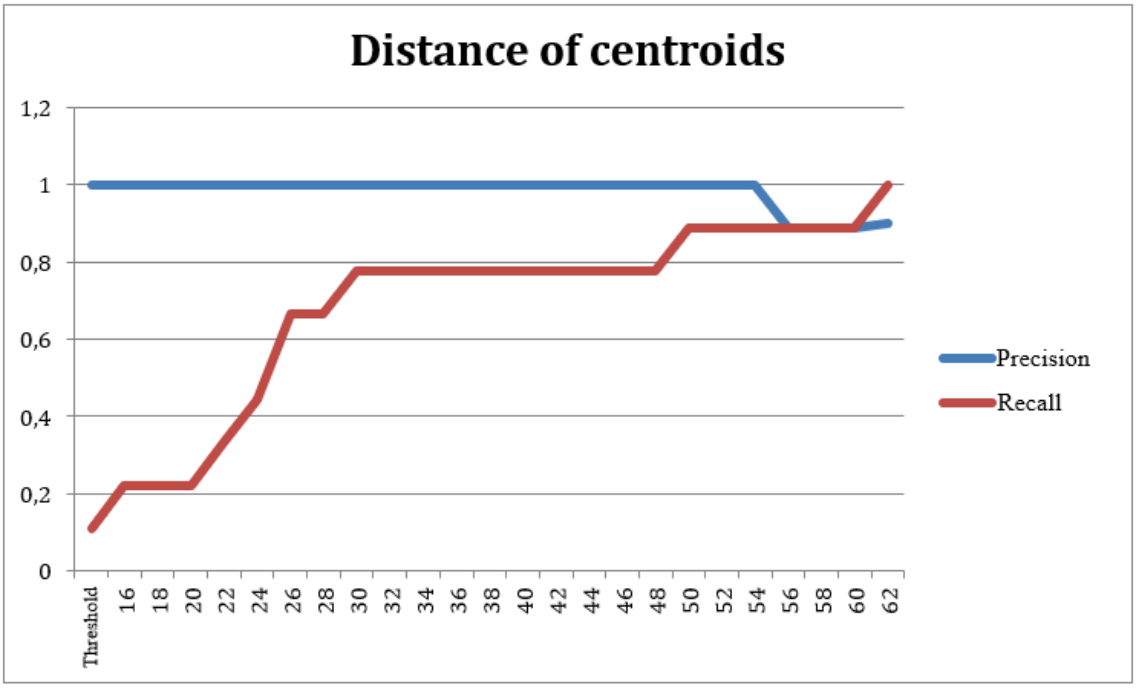
Where b_i is the value of the i-th block of the extracted cluster vector, s_i is the value of the i-th block of the stored cluster vector, and f is the vectorial distance between these points.

The second distance metric is the distance between centroids calculated from the values of the extracted and stored blocks:

$$D_{centroid}(b,\ s) = f\left(\sum_{i=1}^{6}\frac{b_i}{6},\ \sum_{i=1}^{6}\frac{s_i}{6}\right)$$

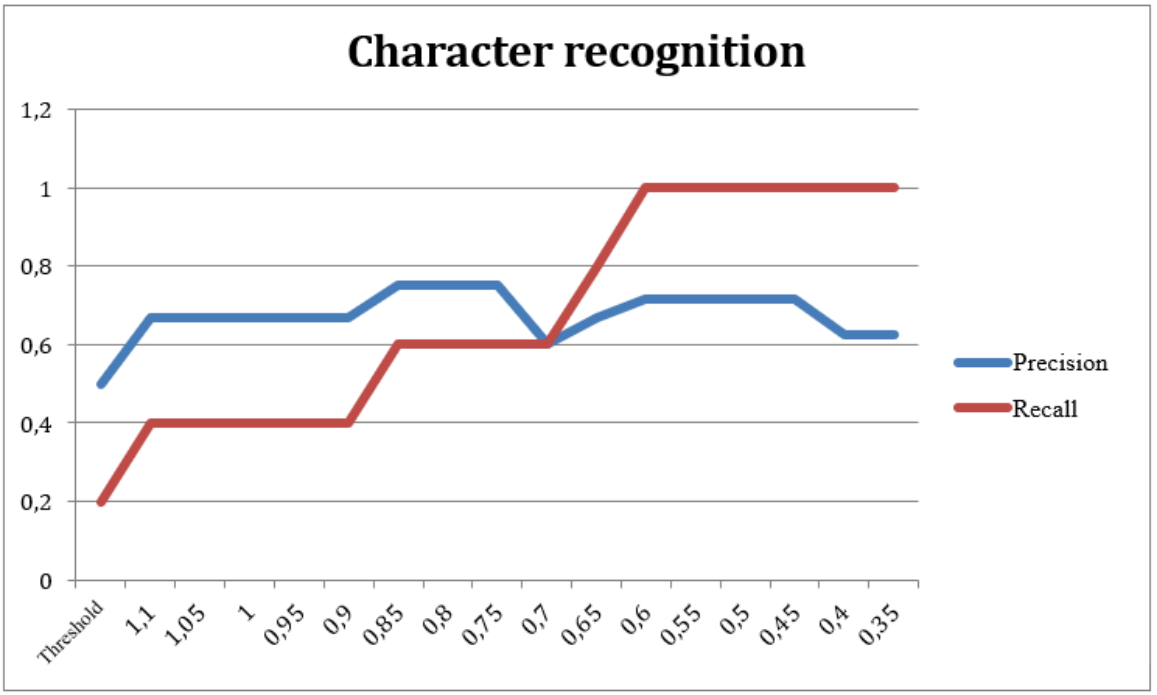
The results are the following:





6.1.2 Optical character recognition

In the case of search using optical character recognition, for each document, I determine a score that is the sum of the weighted values of the keywords that can be accepted as matches, weighted by their idf (inverse document frequency) and tf (term frequency).



6.1.3 Examination of image features

The montage of the database used for examining image features is illustrated in Figure 21 of the appendix. In the system, to evaluate the results, I use a ratio that is defined by the number of vector pairings produced during the matching of the descriptor vectors of the detected book cover and the descriptor vectors of the cover currently being examined, and the number of these that can be accepted as good matches. For assessing acceptability, I apply an additional distance threshold, which I compare to the ratio determined by the distance between the two nearest descriptor vectors to the examined descriptor vector, with a global value of 0.7.

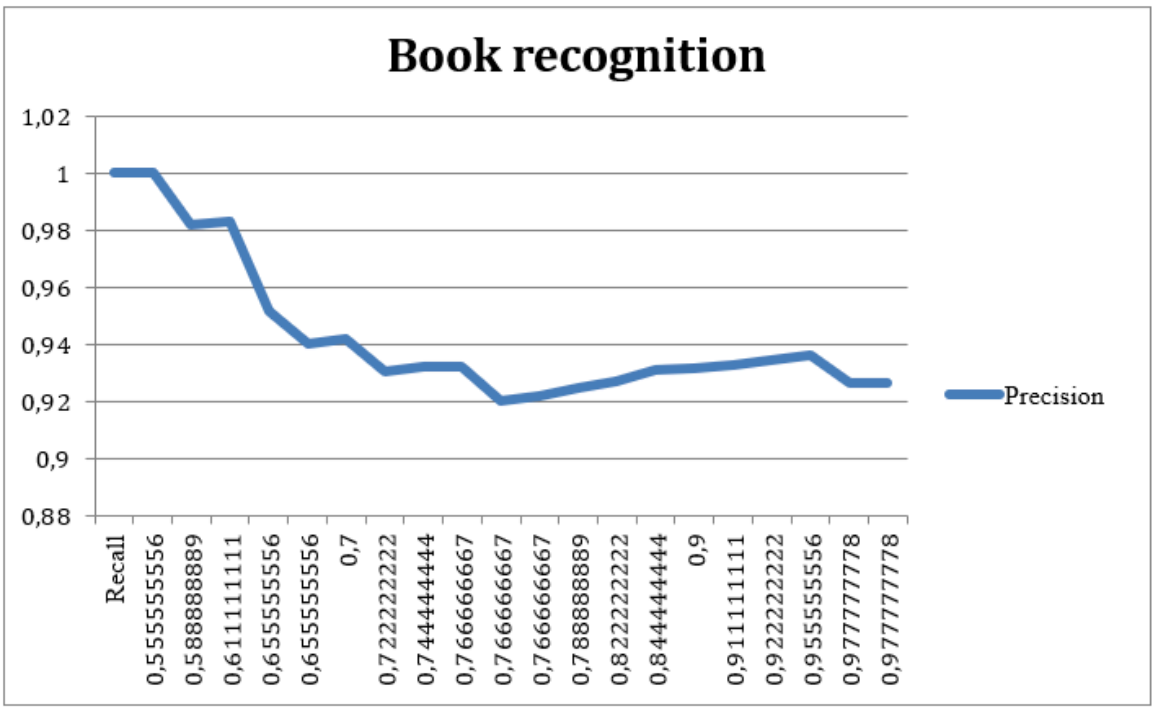
When analysing the results, my goal was to determine the optimal ratio where both recall and precision are maximised. To do this, I looked for the intersection of the curve formed by two values relative to this ratio.

Partial results at a ratio of 0.22:

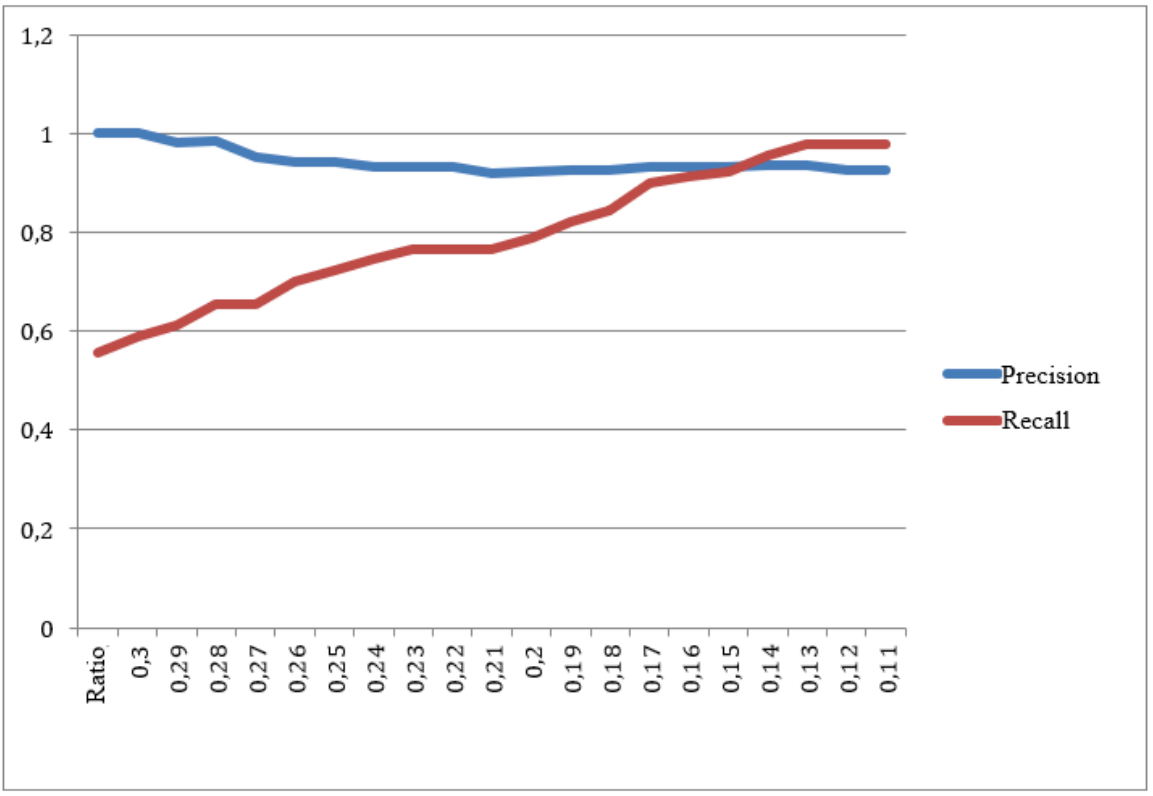
	True	False
Hit	69	5
Miss	21	5

Reduced values at a ratio of 0.16:

	True	False
Hit	81	6
Miss	9	4



The following diagram illustrates the precision and recall ratio obtained by varying the threshold between 0.3 and 0.1.



6.2 Evaluation

The first two measurements were carried out on a small dataset. During the pre-selection phase, I used two types of distance metrics for clustering: one is the maximum distance found between individual blocks, and the other is the distance between the centroids of points formed by the average intensities of the blocks. It is observable that in the first case, the system's recall increases almost linearly, with the harmonic mean at the intersection of the two metrics located around a value of 100. It is easy to see that this approach requires lower computational demand, as in certain cases, the termination condition may be met during the examination of a single block. It is also apparent that this approach does not lead to an optimum, as practically a quarter of the vector space around the points would need to be considered for accurate results. However, in the second case, a steep increase can be observed between radii of 16 and 40. This indicates that even colour distortions in individual blocks, appearing as noise, do not alter the centre of the cluster formed from the points to such an extent that it would not form within a small vicinity of the stored sample's centroid.

In the case of Optical Character Recognition, I believe that I have not yet achieved the desired results. Although the system is capable of handling incorrect detections caused by wrongly detected words in nearly half of the cases, and the idf and tf-based weighting performs as expected, it is necessary to conduct further work in this regard. The graph clearly shows, although recall shows a monotonic increase, precision fluctuates within the 50-80% range, indicating uncertain performance.

It is important to note regarding the threshold ratio changed during the measurement of image features' quality that its value is significantly influenced by the distance threshold used for determining the quality of hits. Generally, for an accepted book match, this ratio significantly stands out compared to the 1-3% range deemed good for vector pairs in other books.

The first diagram shows that the curve representing precision as a function of recall has a local minimum at around 0.76, above which precision shows a monotonic increase up to 0.93, where it begins to decrease again. In the second diagram, the curve formed by recall and precision intersects at a ratio of 0.15, after which recall does not show an increase and precision begins to decline.

Consequently, the new threshold ratio must have a value of at least 0.13 and at most 0.22, corresponding to the recall value of 0.76, with the global optimum at 0.15.

With the application of the global optimum, it can be established that the system can identify individual books from photos of their covers with 93% accuracy based solely on image features. With further development of optical character recognition, this result can be enhanced.

6.3 Opportunities for further development

In the clustering step, it might be interesting to use a subdivision into more blocks than the originally applied 2x3 block arrangement. The measurements revealed that this approach could be robust, and further improvements might be achieved by modifying the circle used to analyse the environment of the colour points into an ellipsoidal shape. This modification could only be achieved by applying different weightings to the a and b channels, considering the shape of the LAB colour space.

Another opportunity for development could be to replace or combine the Stroke Width Transformation used in the system with some other solution. Based on my experience, this approach does not always yield correct results; in some cases, a thresholding solution could lead to greater accuracy. From a development perspective, introducing some additional post-processing step to handle the merging of fragmented keywords detected could be important.

One possibility could be to integrate the system with Amazon's online store, thereby expanding user interactions with shopping. Various social interaction features could also be integrated into the application, such as sharing a book on different channels, commenting, and writing reviews.

Extending the system to the web would also be advisable by developing a web application. This application could use the existing API subsystem, eliminating the need for a new implementation. Document management could also be handled via a web interface to simplify data entry.

A much more complex development could be the implementation and integration of a recommendation system capable of uncovering hidden associations among documents, thereby enhancing the user experience. These documents could be displayed in a related works section at the bottom of the application's document-view screen

7 Conclusion

In the thesis, I have detailed the challenges of book cover-based search in digital library catalogues. I have described the general approaches of CBIR (Content-Based Image Retrieval) systems and the methods used therein. I elaborated on the steps and algorithms necessary for highlighting book covers from context. I illustrated several approaches to book cover recognition, as well as multiple methods for accelerating the search process. I have thoroughly described three similar systems and most of the algorithms used in them.

I outlined the issues arising from optical character recognition when examining book covers, and the approach that can be constructed using dictionary building, language recognition, and stemming to narrow the decision space. I explained the weighting methods based on inverse document frequency and term frequency, as well as the improvements possible through the use of Levenshtein distance.

Using the results of literature research and supplementing with my own ideas, I designed and implemented a system based on a Three-tier architecture to solve the task. Throughout the implementation, I incorporated several of my own solutions and strived to present their functionality in detail. The narrowing of the decision space was performed with clustering in the LAB color space.

From an architectural perspective, the implemented system can be considered easily scalable. The system's implementation has been completed, and testing and measurements have been carried out. The results of my measurement using image features showed 93% accuracy, which proves the robustness of the SURF descriptors and the effectiveness of my classification approach. With further development of Optical Character Recognition, the system's accuracy could potentially be further enhanced.

8 References

[1]Könyvtári katalógusok

(http://www.bdf.hu/konyvtar/informaciokereses/katalogus.htm),
Last visited: 2014. árpilis 13.

[2] 3D Projection (http://en.wikipedia.org/wiki/3D_projection),
Last visited: 2014. árpilis 13.

[3] M. Brown, R. Szeliski, and S. Winder, “Multi-Image Matching using Multi-scale Oriented Patches,” *Technical Report, Microsoft Research*, 2004.

[4] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom et al. “Query by Image and Video Content: The QBIC System,” *IEEE Computer*, vol. 28, no. 9, 1995.

[5] A. Pentland, R.W. Picard, and S. Sclaroff, “Photobook: Tools for Content-Based Manipulation of Image Databases,” *Proc. SPIE*, vol. 2185, pp. 34-47, Feb. 1994.

[6] J.R. Smith and S.-F. Chang, “VisualSEEk: A Fully Automated Content-Based Image Query System,” *Proc. ACM Multimedia*, pp. 87-98, Nov. 1996.

[7] S. Stevens, M. Christel, and H. Wactlar, “Informedia: Improving Access to Digital Video,” *Interactions*, vol. 1, no. 4, pp. 67-71, 1994.

[8] J.Z. Wang, G. Wiederhold, O. Firschein, and X.W. Sha, “Content-Based Image Indexing and Searching Using Daubechies' Wave-lets,” *Int'l J. Digital Libraries*, vol. 1, no. 4, pp. 311-328, 1998.

[9] W.Y. Ma and B. Manjunath, “NaTra: A Toolbox for Navigating Large Image Databases,” *Proc. IEEE Int'l Conf. Image Processing*, pp. 568-571, 1997.

[10] S. Jeong, “Histogram-Based Color Image Retrieval”, *Psych221/EE362 Project Report*, 2001

[11] A. Natsev, R. Rastogi, and K. Shim, “WALRUS: A Similarity Retrieval Algorithm for Image Databases,” *SIGMOD Record*, vol. 28, no. 2, pp. 395-406, 1999

[12] C. Carson, M. Thomas, S. Belongie, J.M. Hellerstein, and J. Malik, “Blobworld: A System for Region-Based Image Indexing and Retrieval,” *Proc. Visual Information Systems*, pp. 509-516, June 1999.

[13] J.R. Smith and C.S. Li, “Image Classification and Querying Using Composite Region Templates,” *Int'l J. Computer Vision and Image Understanding*, vol. 75, nos. 1-2, pp. 165-174, 1999

[14] J. Z. Wang, J. Li and G. Wiederhold, “SIMPLIcity: Semantics-Sensitive Integrated Matching for Picture Libraries,” *IEEE Transactions on Pattern Analytis and Machine Intelligence*, vol 23, no 9, September 2001

[15] K.Velmurugan, Lt.Dr.S. Santhosh Baboo, “Content-Based Image Retrieval using SURF and Colour Moments” *Global Journal of Computer Science and Technology*, vol 11, May 2011

[16] P. Indyk and R. Motwani, “Approximate Nearest Neighbours: Towards Removing the Curse of Dimensionality” *Proceedings of the 30th Symposium on Theory of Computing*, pp. 604-613, 1998

[17] Levenshtein, Vladimir I., “Binary codes capable of correcting deletions, insertions, and reversals,” *Soviet Physics Doklady*, vol 10 (8), pp. 707–710, February 1966

[18] R. Smith, D. Antonova and Dar-Shyang Lee, “Adapting the Tesseract Open Source OCR Engine for Multilingual OCR,” *Proceedings of the International Workshop on Multilingual OCR*, 2009

[19] G. Nagy, “Chinese character recognition: a twenty-five-year perspective,” *9th Int. Conf. on Pattern Recognition*, pp. 163-167, November 1988

[20] Porter, Martin F., “An Algorithm for Suffix Stripping,” *Program*, 14(3): 130–137, 1980

[21] Snowball (http://snowball.tartarus.org/index.php),
Last visited: 2014. árpilis 13.

[22] W. B. Cavnar , J. M. Trenkle, “N-grambased text categorization,” *Proc. of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, 1994

[23] T. Gottron and N. Lipka, “A Comparison of Language Identification Approaches on Short, Query-Style Texts”,

[24] G Windisch, L. Csink, “Language Identification Using Global Statistics of Natural Languages”, *Proceedings of the 2nd Romanian-Hungarian Joint Symposium on Applied Computational Intelligence (SACI)*, pp. 243-255, 2005

[25] C. Galleguillos, “Book Cover Recognition Project,” *University of California San Diego* (http://cseweb.ucsd.edu/classes/wi06/cse190a/proposals/cgallegu.pdf), Last visited: 2014 április 14.

[26] Lowe, David G., “Object recognition from local scale-invariant features,” *Proceedings of the International Conference on Computer Vision 2*., pp. 1150–1157, 1999

[27] MacQueen, J. B., “Some Methods for classification and Analysis of Multivariate Observations,” *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability 1. University of California Press*. pp. 281–297, 1967

[28] Shao-Chuan Wang, “Book Cover Recognition”, Columbia University (http://www.slideshare.net/shaochuan/book-cover-recognition), 2010

[29] C. Cortes, V. Vapnik, “Support-vector networks,” Machine Learning 20 (3): 273., 1995

[30] S. S. Tsai, D. Chen , H. Chen , Cheng-Hsin Hsu , Kyu-Han Kim , J.P. Singh , B. Girod, “Combining image and text features: a hybrid approach to mobile book spine recognition,” *Proceedings of the 19th ACM international conference on Multimedia*, 2011

[31] J. Matas, O. Chum, M. Urban, and T. Pajdla., “Robust wide baseline stereo from maximally stable extremal regions,” *Proc. of British Machine Vision Conference*, pp 384-396, 2002

[32] B. Epshtein, Y. Wexler, and E. Ofek, “Stroke Width Transform,” *IEEE International Conference on Computer Vision and Pattern Recognition,* 2010

[33] H. Bay, A. Ess, T. Tuytelaars, L. Van Gool, “SURF: Speeded Up Robust Features,” *Computer Vision and Image Understanding (CVIU)*, vol. 110, no. 3, pp. 346-359, 2008

[34] D. Nister and H. Stewenius, “Scalable recognition with a vocabulary tree,” *CVPR*, 2006

[35] Altman, N. S., “An introduction to kernel and nearest-neighbor nonparametric regression,” *The American Statistician 46*, pp 175–185, 1992

[36] Palágyi K., “Képfeldolgozás haladóknak,” *Szegedi Tudományegyetem*, 2011

[37] V. Hong, H. Palus, D. Paulus, “Edge Preserving Filters on Color Images,” *Universität Koblenz-Landau*, 2004

[38] Jan Eric Kyprianidis, “Image and video abstraction by multi-scale anisotropic Kuwahara filtering,” *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering*, August 2011

[39] R. C. Gonzales, R. E. Woods, “Digital Image Processing. 2nd,” *Prentice Hall*, 2001

[40] P. V. C. Hough, “A Method and Means for Recognizing Complex Patterns,” *US Patent 3,069,654*, 1962.

[41] N. Otsu, “A threshold selection method from grey-level histograms,” *IEEE Transactions on Systems, Man, and Cybernetics*, pp. 62-66, 1979

[42] Pearson, K., “On Lines and Planes of Closest Fit to Systems of Points in Space,” *Philosophical Magazine 2*, pp. 559–572, 1901

[43] XYZ to LAB (http://www.brucelindbloom.com/index.html?Eqn_XYZ_to_Lab.html), Last visited: 2014 április 14.

[44] D. Chen, S. Tsai, C.-H. Hsu, K.-Y. Kim, J. P. Singh, and B. Girod, “Building book inventories using smartphones,” *ACM Multimedia (MM)*, October 2010

[45] D. Tikk, “Szövegbányászat,” *Typotex*, 2007

[46] Chapter 11 Non-Parametric Techniques (https://www.byclb.com/TR/Tutorials/neural_networks/ch11_1.htm), Last visited: 2014 április 14.

[47] CHAPTER 5 Learning Music Signals (http://web.media.mit.edu/~tristan/phd/dissertation/chapter5.html), Last visited: 2014 április 15.

[48] Z. Vámossy, “Vonalak, görbék, sarokpontok, ”, Budapest Tech, 2004

9 Appendix

9.1 Main Classes of UDC

General works

1. Philosophy
2. Region
3. Social sciences
4. (currently not classified)
5. Natural sciences

    51. Mathematics
        - 511. Number theory
        - 512. Algebra
        - 513. Geometry
        - 514. Analysis
        - 515. Combinatorics
    52. Astronomy, astrophysics
    53. Physics
    54. Chemistry, mineralogy
    55. Earth science, geológia
    56. Palaeontology
    57. Biology
    58. Botany
    59. Zoology

6. Applied Sciences
7. Arts, sports
8. Language and literature
9. History and geography

9.2 Identification process

The identification process is illustrated by the following flow diagram:

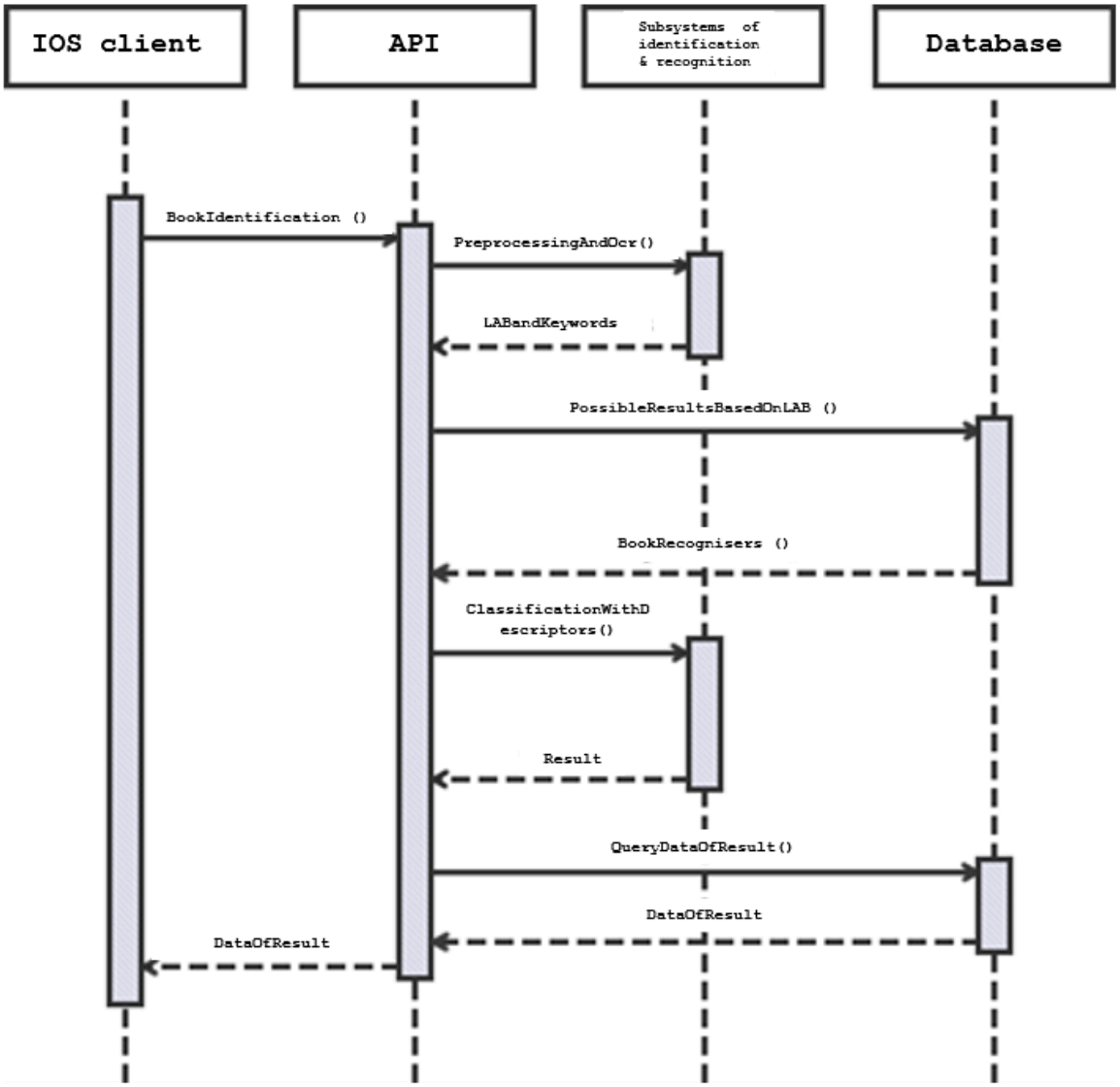


Figure 14, flow diagram of identification

9.3 Previous Test Results from the Prototype

In the background is the photo used for recognition, on the left is the result of the cropping and transformation into space, while on the right the result can be seen.

Figure 15, early test result

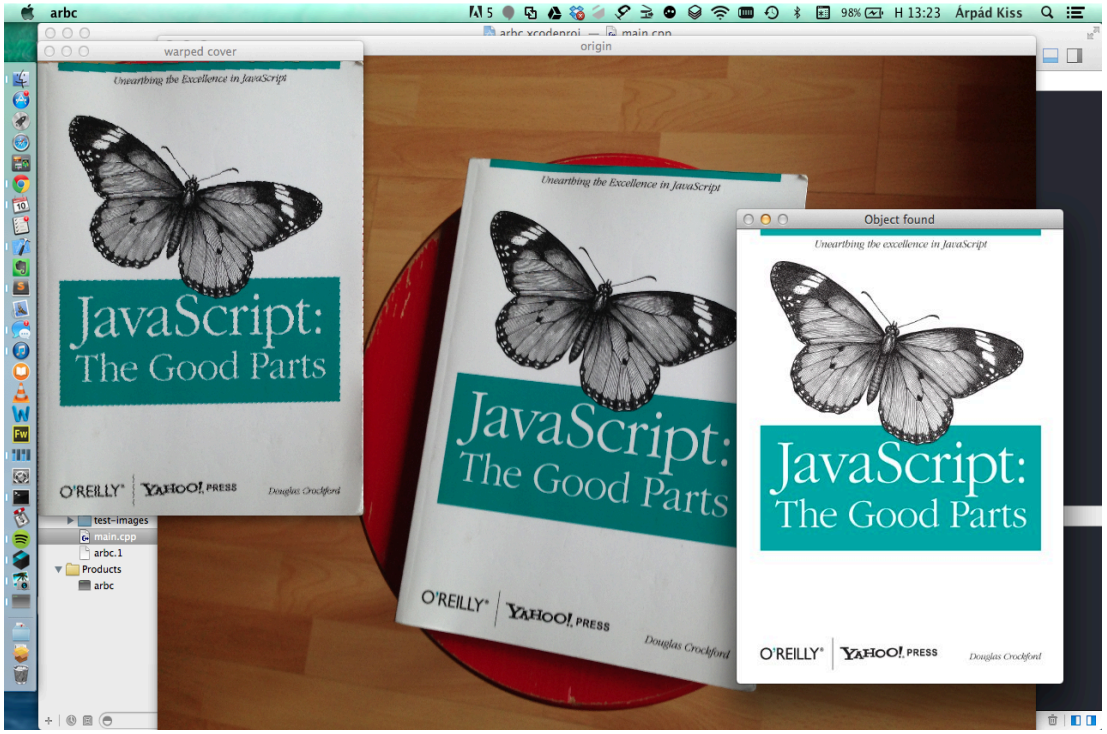


Figure 16, early test result

Figure 17, early test result

Figure 18, early test result

In Figure 19, the distortion in the model mapping resulting from the lack of the object's principal axis determination is clearly observable. Nonetheless, the system still delivers a correct result.

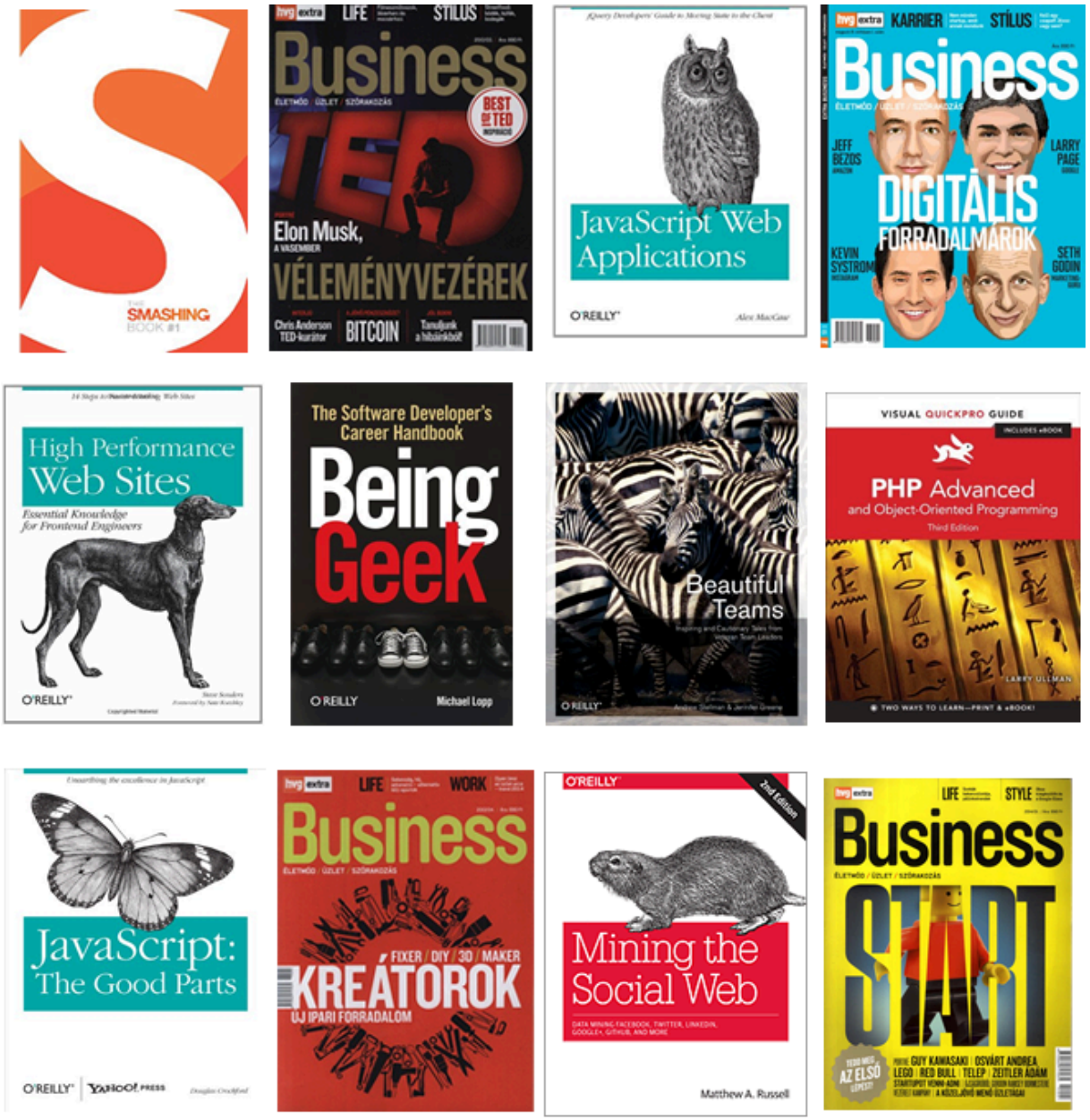


Figure 19, the training data of the early prototype

9.4 The Database Used for Testing

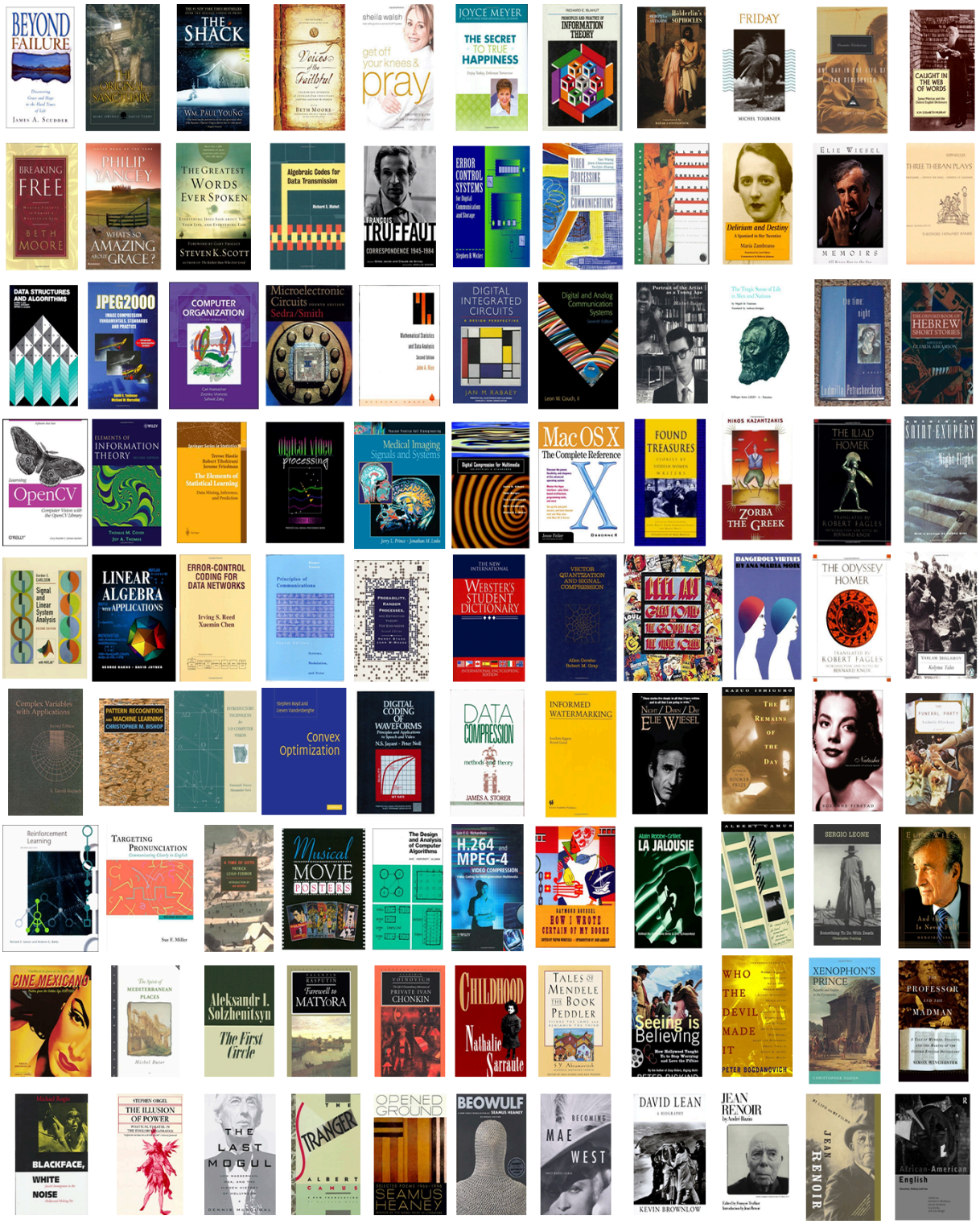
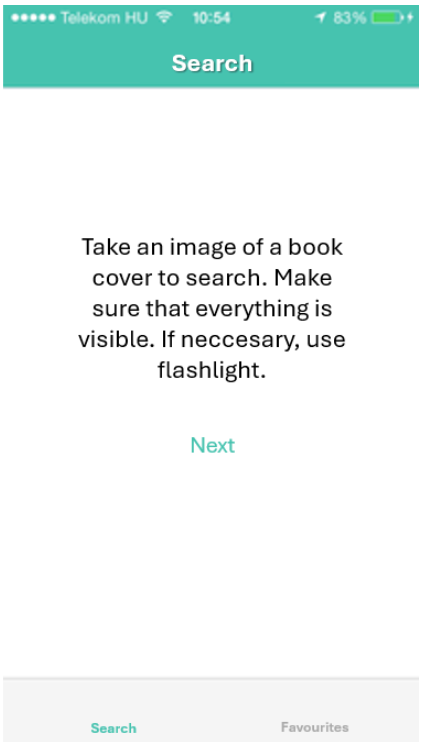


Figure 20, montage of the database used during testing

9.5 User Manual

9.5.1 Home Screen



When the user launches the application, they arrive at this screen. The two menu items found in the bottom bar allow access to the two sections of the application.

The *Continue* button located in the centre of the screen enables the user to proceed to the search interface.

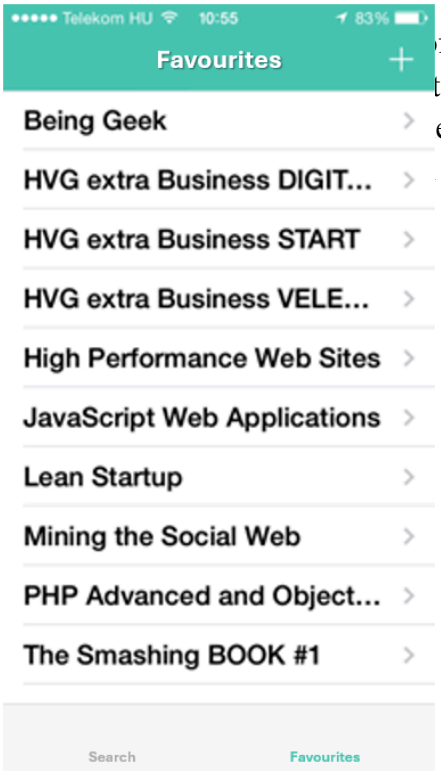
9.5.2 Capturing an Image

Upon reaching this screen, the user can take a photo of the book cover. The action buttons located in the top bar allow for the flash to be turned on and off, the grid to be enabled and disabled, and switching between the front and rear cameras. Using the icon with two squares at the bottom of the screen, the user can browse through previously taken photos, select one, and start a search with it. The photo is taken by pressing the large cyan circle.

9.5.3 Result

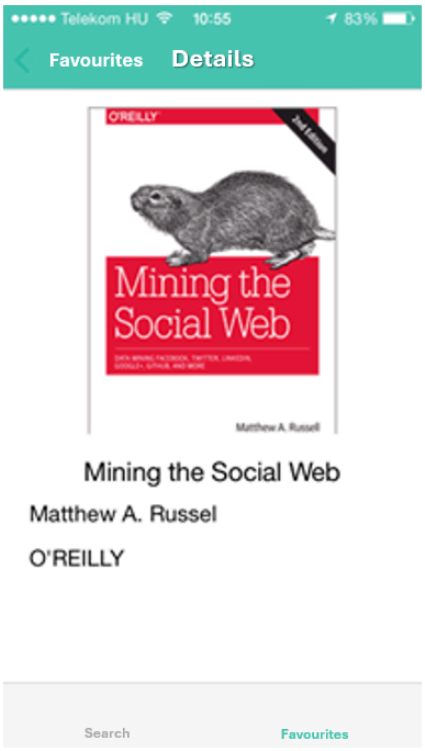
If the identification of the book cover in the photo sent by the user is successful, the user will reach this screen. They have the option to save the book displayed as a result to their favourites using the *Save to Favourites* button located at the bottom.

9.5.4 Favourites



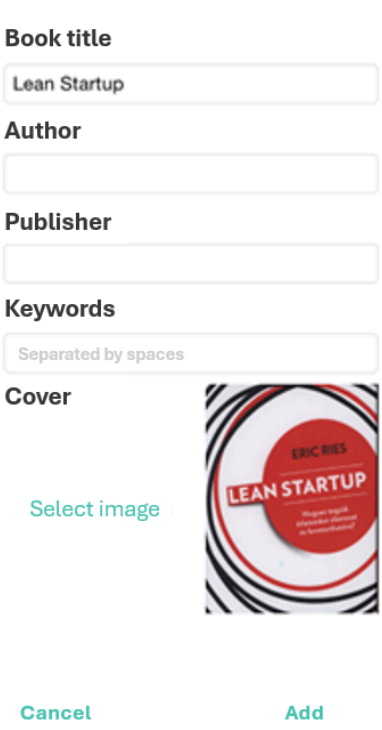
f books that the user has saved and added. By swiping a row te a book from their favorites list. Using the + button located e user can navigate to the "Add New Book" screen. By access the details page for that particular book.

9.5.4 Viewing favourites



This screen is essentially identical to the search results screen displayed after a search, with the only difference being the absence of the Save button.

9.5.5 Adding a New Book



This screen is designed for entering new books. Here, the user can provide the metadata associated with the book they wish to upload, as well as other keywords featured on the cover. Additionally, the user has the option to crop the cover image.

9.6 Installation Guide

9.6.1 System Requirements

- Operation system Ubuntu 12.04 vagy annál újabb, (3.2.24-es kernel legalább)
- Memory: 256 MB
- HDD space: 512 MB (1 GB javasolt)
- Processor: 1 mag

9.6.2 Downloading the Source Code

The system is located in two separate GIT repositories, which are encompassed by a third repository. To download the source code, only this third repository needs to be recursively cloned; the subrepositories will automatically be unpacked into the appropriate locations:

```
git clone --recursive https://github.com/arpad1337/ARBC.git
```

The source code of the iOS client is also saved in a GIT repository:

```
git clone https://github.com/arpad1337/ARBC-APP.git
```

9.6.3 Installation of Dependencies

The system has the following dependencies:

- OpenCV 2.4.9
- Leptonica 1.7
- Tesseract 3.0.2
- NPM 1.4.3
- Node 0.10.26

The installation steps below have been tested on Ubuntu 12.04 version; commands may vary for other distributions.

9.6.3.1 OpenCV 2.4.9

```
cd

sudo apt-get install build-essential

sudo apt-get install cmake git libgtk2-dev pkg-config
libavcodec-dev libavformat-dev libswscale-dev

sudo apt-get install python-dev python-numpy libtbb2
libtbb-dev libjpeg-dev libpng-dev libtiff-dev libjasper-dev
libdc1394-22-dev
```

```
mkdir opencv-working-dir

cd opencv-wokring-dir

git clone https://github.com/Itseez/opencv.git

cd opencv

mkdir release

cd release

cmake -D CMAKE_BUILD_TYPE=RELEASE -D
CMAKE_INSTALL_PREFIX=/usr/local -D WITH_TBB=ON -D
BUILD_NEW_PYTHON_SUPPORT=ON -D WITH_V4L=ON -D
INSTALL_C_EXAMPLES=ON -D INSTALL_PYTHON_EXAMPLES=ON -D
BUILD_EXAMPLES=ON -D WITH_QT=ON -D WITH_OPENGL=ON
BUILD_opencv_nonfree=ON ..

make -j2

sudo make install

sudo sh -c 'echo "/usr/local/lib" >
/etc/ld.so.conf.d/opencv.conf'

sudo ldconfig

cd ../../
```

9.6.3.2 Leptonica 1.70

```
cd

mkdir leptonica

cd leptonica

wget http://www.leptonica.com/source/leptonica-1.70.tar.gz

tar –zxvf leptonica-1.70.tar.gz

cd leptonica-1.70

./configure --prefix=/usr/local && make && make install

sudo ldconfig

cd ../../
```

9.6.3.3 Tesseract 3.02.02

```
cd

sudo apt-get install libpng-dev libjpeg-dev libtiff-dev
zlib1g-dev

sudo apt-get install gcc g++

sudo apt-get install autoconf automake libtools
checkinstall

mkdir tesseract

cd tesseract

wget
https://tesseract-ocr.googlecode.com/files/tesseract-ocr-3.
02.02.tar.gz

tar -zxvf tesseract-ocr-3.02.02.tar.gz

cd tesseract-ocr

./autogen.sh

./configure

make

sudo make install

sudo ldconfig

cd ../

wget
https://tesseract-ocr.googlecode.com/files/tesseract-ocr-3.
02.eng.tar.gz

tar -zxvf tesseract-ocr-3.02.eng.tar.gz

mkdir /usr/local/share/tessdata

cp ./tesseract-ocr/tessdata/* /usr/local/share/tessdata/

cd ..
```

9.6.3.4 Node and NPM

```
cd

sudo apt-get install python-software-properties

sudo apt-add-repository ppa:chris-lea/node.js

sudo apt-get update

sudo apt-get install nodejs

sudo apt-get install npm
```

9.6.3.5 Boost 1.55

```
cd

wget -O boost_1_55_0.tar.gz
http://sourceforge.net/projects/boost/files/boost/1.55.0/bo
ost_1_55_0.tar.gz/download

tar xzvf boost_1_55_0.tar.gz

cd boost_1_55_0/

wget https://dl.dropbox.com/u/88131281/install_boost.sh

chmod +x install_boost.sh

./install_boost.sh

cd ../../
```

9.6.4 Building of Image-Processing Module

A compiler script named COMPILE has been created for this purpose. Running this script compiles the image-processing module and the program that performs SWT (Stroke Width Transform).

```
cd image-processing/arbc

chmod +x COMPILE

./COMPILE
```

9.6.5 Installation and running of API

To launch the API, first resolve its dependencies, which are listed in the package.json file. Installation:

```
cd api

npm install
```

Running:

```
export NODE_PORT=1337; node app.js
```

The API will be accessible at http://localhost:1337 after starting.

9.6.6 Switching the Application to Your Own API Instance

To enable the application to communicate with the new API address, simply change the URL in the *init* function found in the ARBC/APIAgent.m file.