Book Cover Identification System

Team Haplocoyponsis

George Hodgson        Douglas Krofcheck       Jeremy Leakakos
Elizabeth Leib        David McClelland

Project Sponsor
Pro Quo Books

Faculty Mentor
Robert Kuehl

Project Overview

Pro Quo Books is an online book distributor, reselling books on e-commerce sites such as Amazon and Half.com. Before a new book can be added to Pro Quo’s inventory, it must be identified in order to be priced and sold correctly. The Book Cover Identification System shall identify books based on their cover art and weight. The covert art images are to be taken when the book is passed along on a conveyor belt system. Using a scanbundle consisting of this cover art image and the weight, the system shall search through an existing database of book information and return the closest matching book as well as a confidence rating representing the correctness of the returned book.

As the system will be integrated into Pro Quo Book’s preexisting book sorting process, there is a criterion of a strict time limit of nine seconds from when the system receives the book information to when it must be identified. In light of the strict time constraints on the system, Pro Quo Books is considering upgrading their server to meet performance goals if needed.

Every book the system is able to correctly identify will increase the value of the product as that is one less book that will need to be manually identified by workers. Furthermore, when the system claims to identify a book, but is actually incorrect, the book will be advertised online under the wrong title. If sold, Pro Quo Books’ automated shipping system may mail this misidentified book to a customer. Delivering the wrong book to a customer increases costs and hurts Pro Quo Books' reputation with customers. As such, the system must try to minimize the number of false positives. Unlike many software projects, the goal of the project is not to implement a large amount of features, but to implement one feature with the best accuracy and speed as possible.

Basic Requirements

The Book Cover Identification System has three main features: image and database preprocessing, book identification, and the test application, which simulates the production environment. These three features will work in combination to allow Pro Quo Books to deploy and test any existing or new algorithms for book identification.

The book identification algorithms take scan bundles consisting of covert art and weight as inputs. Each identification algorithm attempts to generate a BookBreedID
representing the matching book as well as the confidence level demonstrating the quality of the possible match. Under specified conditions, an exception is thrown instead. These conditions occur when no possible IDs could be guessed, when the guesses all had too low of a confidence rating, or when there were multiple BookBreedID guesses with the same confidence rating. The test application can run the identification process on many scan bundles. It also outputs test metrics, including the number of true positives, the number of false positives, the failed identification rate, and the average time taken to identify a book. One can use the test application to test new algorithms or different combinations of existing algorithms.

**Constraints**

The books being identified by the system are laying on a moving conveyor belt. There is a physical point on the conveyor belt at which the book must be identified by. The system will have approximately nine seconds from the time the information is passed to the system until the book on the conveyor belt reaches the critical point. Even if the book is identified after nine seconds, the information would no longer be of use to the system as a whole as the book will have already traveled past the conveyor belt section where it had to be identified in order to be properly sorted.

When preprocessing the existing database of book cover images, metadata will be calculated and stored in the database alongside the cover images. This information adds to the size of the database. However, the information is necessary to reach the nine second performance goal as outlined above. The database server which is used by the identification system must be monitored to ensure adequate storage capacity is maintained.

A large amount of the processing done by the identification system takes the form of SQL queries. The overall speed of the identification system will be impacted greatly by the speed of the database server on which the queries are run. A dedicated database server is suggested, as concurrent queries and other database traffic may lead to slowdowns and missed performance goals.

The team used Visual Studio, C#, and MySQL since those were the technologies our sponsor, Pro Quo Books, is currently utilizing. Using the same technologies will make it easier for others to continue to work on the project if they choose to do so.

**Development Process**

The team used the Evolutionary Delivery lifecycle model for this project. The Evolutionary Delivery model supports delivery of portions of the software at different times. After the requirements elicitation and architecture design phases, we moved into a cyclical development process. After each short release, the team received sponsor feedback through weekly teleconference meetings and incorporated the results into the next increment.

This lifecycle model increases visibility of progress, encourages communication between the team and the sponsor, and supports more frequent product releases. This
process was chosen because the core functionality, book identification, was known and well understood, but uncertainty existed around the specific algorithms.

The team used incremental development, splitting up development time into multiple 2-3 week long increments. The first increment involved designing and implementing the test application used to test our algorithms. This was followed by three prototype increments where the team researched image identification techniques. The final increments were product increments. In these increments, the algorithms were polished and the test application was refactored for the final release.

Team roles were identified and filled based on the areas of expertise of each team member. Jeremy served as team leader and sponsor contact, running our weekly meetings. George was the notetaker, recording meeting minutes for the team’s reference. Jeremy and Doug were the computer vision experts, sharing their knowledge about the problem domain. Doug also served as the webmaster, designing, creating, and regularly updating the Team Haplocyonopsis website. Elizabeth was our C# expert, helping the team adopt to this new technology. Finally, Dave was the keeper of the timesheets, updating this important progress metric every week.

**Project Schedule: Planned and Actual**

The schedule for this project was designed around two different sets of deliverables/milestones: Pro Quo Deliverables and SE Deliverables. Pro Quo Deliverables were customer facing releases that demonstrated completed functionality for different parts of the project. This included an initial test application, a prototype of the libraries to be included, and the finished product. SE Deliverables were defined on the Software Engineering Senior Project community web site. These included a project plan, interim and final presentations, a poster presentation, and this technical report. The work for this project was separated into multiple, mostly two-week, increments. The end of these increments coincided with the different deliverables.

The actual schedule that the project ran on was almost identical to the planned schedule. One increment release slipped by about a week, but all others went according to plan. This slip was caused by a miscommunication about how the release would be handled. The team was able to compensate by getting the release out a little bit late, and continuing on with the next increment, and getting it out on time.

**System Design**
The system architecture is designed around the two system libraries: Identification and Preprocessing. Logic associated with identifying algorithms is encapsulated in the identification library. Likewise, any preprocessing logic is contained within the preprocessor library. Two more libraries exist. One being a 'common library' which contains classes shared by all the other libraries and the other being the entry point Pro Quo Books should use to identify a book. The entry point is not in the identification library, as one uses the test application to configure the identification sequence. The configuration has to be loaded from somewhere, so we thought a 'main library' would be better than duplicating the functionality in different identification libraries. The context in which the two main libraries are used falls into two categories: identification and system updating.

When a book is passed into the system from the conveyor belt, it is first sent to the main library. If this is the first time a book is being identified since the system's last restart, the configuration/settings files are read and the correct identification, sql filtering, and preprocessing algorithms are linked and loaded into memory. The main library converts the given ScanBundle into a Book format and sends it through a pipe-and-filter architecture.

First, the new book is preprocessed by the preprocessing library/algorithms. Such preprocessing tends to include cropping the book out from the conveyor belt. Second, the book is sent through the SQL filter algorithms. These take some statistics of the image and generate a database query to get similar books from the matching database. Now that the possible results are down to a much more manageable level, the book is sent on to the identification library.

At the identification library, the book is compared against all the possible matches returned from the above SQL query. Confidence values are generated and stored based
Upon how will the identification algorithms think the unknown book matches a book in the known dataset. The identification phase is over, the book with the confidence is normally returned.

Backing up to the SQL filters for a moment, a variety of coverart image statistics are pre-calculated and stored in the database. These statistics include weight, aspect ratio, weight bin, aspect ratio bin, pre-calculated pixel difference values, as well as RGB mean, median, and standard deviation. By storing these values, we can easily look up books that have similar values. Such books are much more likely to match than those without similar values. As Pro Quo Books adds more coverart images into the database, these statistics must be calculated as well, or the system will not function correctly.

When the updating routine calls into the preprocessing library, the system queries the database and locates all images for which preprocessed statistics that are not currently calculated and updates all values accordingly. Once all new data has been preprocessed, updating is complete and the updating script is complete.

Determining which coverart rows are missing preprocessed data requires several complex and time consuming SQL JOIN statements to execute. To allow the database server to dedicate maximum processing power to the intense queries required during identification, it was decided that these updates should be performed during off-hours in a nightly build instead of during working hours.

Below are more detailed descriptions of the identification and SQL filter algorithms.

**Binned weight filter**
- Books are separated into multiple bins based on their weight. Bins can be created with either a fixed width (an equal range across the entire set) or fixed size (a roughly equal number of books in each bin).

**Binned aspect ratio filter**
- Books are separated into multiple bins based on their aspect ratio. Bins can be created with either a fixed width (an equal range across the entire set) or fixed size (a roughly equal number of books in each bin).

**Weight filter**
- Only books with weights within a certain percentage threshold are returned from the SQL SELECT statement.

**Aspect ratio filter**
- Only books with aspect ratios within a certain percentage threshold are returned from the SQL SELECT statement.

**RGB Statistics Identifier**
- This algorithm uses AForge.NET to calculate the color histograms of each image, one for each channel, red, green, and blue. The algorithm then calculates the mean, median, and standard deviation for each of the three histograms. The differences between the RGB stats of the scanbundle image and each candidate
image are added together. The candidate image with the smallest RGB difference is selected as the correct book. This algorithm does not deal with gray scale cover arts, since such images do not have RGB statistics.

The confidence level of the identification is calculated in the same manner as the pixel difference identifier algorithm. A candidate image’s score is the sum of the difference in R stats plus the difference in G stats plus the difference in B stats. In image with zero differences is a perfect match with confidence of 100%. Otherwise, the lowest and second lowest scores become the best and runner up scores, respectively. The confidence level is based on how far away the runner up is from the best score.

\[
\text{rStats} = \text{redMeanDiff} + \text{redMedianDiff} + \text{redStdDiff} \\
\text{gStats} = \text{greenMeanDiff} + \text{greenMedianDiff} + \text{greenStdDiff} \\
\text{bStats} = \text{blueMeanDiff} + \text{blueMedianDiff} + \text{blueStdDiff} \\
\text{Score} = \text{rStats} + \text{gStats} + \text{bStats} \\
\text{Confidence} = (\text{runnerUpScore} - \text{bestResultScore}) / \text{bestResultScore}
\]
The algorithm examines some number, \( n \), sub-images evenly spaced horizontally and vertically about the source image. For each sub-image examined, the RGB values are averaged and stored as an averaged meta-pixel. This results in \( n \) meta-pixels which serve as a type of fingerprint for the source image.

Additional images are examined with the same \( n \) and the sub-images are extracted from the same relative (percentage-based) coordinates.

When comparing two fingerprints the difference between the fingerprints’ R, G, and B values for all meta-pixels is computed and then averaged. The result is a numerical score denoting how different the images are. The image with the smallest score (least different) is selected as being the match.

Computing the meta-pixel averages for all coverart in the dataset is an expensive operation. During prototyping the number, position, and size of sub-images will be adjusted to determine what gives the best accuracy/time tradeoff. Once these constants are determined, the coverart for all bookbreeds can be preprocessed and the average R, G, and B values for each meta-pixel stored in the database. At that point a production comparison will require only that the meta-pixel averages for the scanbundle image be computed and a database query issued to find the bookbreed whose coverart has the smallest difference.
The above diagram represents a quick and rough overview of the classes within the entire system. The complete class diagram is not shown as we won't be describing every class in detail here and it would only confuse the reader to include unneeded information. To get a complete class diagram of the system, use one of the many numerous code analyzer tools. Visual Studio has a class relationship exporter tool built-in too. The following is a list of the important packages and a description of what they hold or represent:

- **Common** – Contains the classes common to the other libraries in the system. This is packaged up as Common.dll.
  - Models.Interfaces – Contains the interfaces to the identifier, preprocessor, and SQL filter algorithms. Each interface can be packaged up into its own DLL or be included with other interfaces into the same DLL. As an example, the preprocessor library happens to be two IPreprocessor classes combined into a single DLL file.
Models.Book – Contains the classes which hold book data. A Book represents a BookBreedID from Pro Quo Book’s system. While not show in the above image, UnknownBook is used in the test application. A separate class was needed as the test application must also track the correct ID of each book. We decided sub-classing the existing book class was the easiest way to accomplish the ID tracking.

Models.Exception – Contains the exceptions that are throw under different specific conditions when attempting to identify a book.

- Identifier – Contains the identification and SQL filter algorithm classes. This is packaged up as Identifier.dll.
- Main – Contains DeployedEntryPoint, which is where the identification function lies. That identification function is the only call Pro Quo Books needs to make into the system in order to identify a scan bundle. This is packaged up as Main.exe, but it is meant to be used as a DLL, not as a stand alone executable.
- BookIDTestApp – Contains the classes used only for the test application. This is packaged up as BookIDTestApp.exe and is not required when the system is running on the conveyor belt.
- Preprocessor – Contains the preprocessing algorithm classes. This is packaged up as Preprocessor.dll.

**Process and Product Metrics**

The team tracked effort, progress, and productivity metrics as required by the department. Tracking these metrics helped the team, as well as outside observers, understand how the project was progressing.

The effort metrics include hours worked each week as well as the team's estimation accuracy. Each team member estimated and tracked his or her individual time spent on each project task. At the end of each week, the time sheets were aggregated to determine the team's estimation accuracy; each task's actual development time will be divided by that task’s estimated development time. Results showed the team overestimated development time during the first quarter, but improved estimation skills during the second quarter.

The progress metrics include the time taken for a given algorithm to correctly identify a processed book. This metric illustrates how fast the product performs the task of identifying a book. There is a requirement that states the product must identify a book within nine seconds. The results of our progress metrics show the system meets the stated performance goal when testing on the small data set of images provided by the project sponsor. The first book to be identified tends to have a higher identification time as the identification libraries are loaded from the DLLs when they are first used. However once they have been loaded, they stay loaded unless the test application changes its stored values.

The productivity metrics include the percentage of matching books verses the development time for the corresponding algorithm. This metric allows one to see if the
time spent developing an algorithm is actually improving or harming the algorithm. The team spent many hours researching possible solutions, such as optical character recognition (OCR), which ended up not being used.

The test application reports the number of true positives, false positives, and failed identifications, as well as the average time taken to identify each book. An important project goal is to limit the number of false positives. There can be no more than one in two hundred books that are matched and misidentified. For each book that is misidentified, there may be a significant cost to the customer.

Our final results show we met our performance and accuracy goals with the small scale test data we were given. Using around 200,000 “perfect” cover images from Amazon, the test application reported a 99.9% true positive rate. When using about 100 actual book cover images provided by the sponsor, the success rate was 81.6%. Due to cropping and lighting issues, accuracy is harder to achieve with actual production data, but any books correctly identified provides business value to the sponsor and simplifies their process. The best algorithm seemed to be the RGB identifier and the best SQL filter appeared to be the binning weight filter.

**Product State at Time of Delivery**

**Test Application**

Initial mocks of the test application provided a very comprehensive set of features. These features were aimed at enabling developers (both on the senior project and Pro Quo Books teams) to debug and test algorithms that were under development. As the senior project team developed and used the test application, several of these features were found to be of less importance than originally thought.

The test application was originally planned to provide the user an option of specifying some integer number, and that many threads would be started and that number of concurrent identification queries would be launched. This functionality was not implemented, as it served little purpose to the developer beyond being a benchmark of the database and network connection.

The test application was originally planned to have an option where identification tests would be run using all permutations of algorithms. The results from these tests would be output and the best combination would be shared with the user. This functionality proved to be impractical. As the number of algorithms increased the number of tests skyrocketed and the time to complete such a comprehensive test would have been prohibitive. The test application currently runs any number of tests one at a time.

**Identification Library**

Some discussions were had late in the project regarding the possibility of the identification library attempting to identify the image passed to it, and then also rotating the image 180 degrees to attempt identification again in the event that the book was placed on the conveyor belt rotated. This functionality was not included in the
identification library. The team felt that this would have limited the reuse of the library and slowed down the overall identification rate. If Pro Quo Books desires this functionality, it would be simple to implement using the ‘vanilla’ identification library that was provided.

**Preprocessing Library**

The preprocessing library has many methods which generate all metadata necessary for book identification. As algorithms are added by future teams, the corresponding preprocessing code must be added to this library. All necessary preprocessing is execute via a call to PreprocessingLibrary.PreprocessDatabase().

**Project Reflection**

This project required significant research to learn the problem domain of image processing. The research portion of the project went well. A few of our team members had taken or were taking RIT's Intro to Computer Vision course to further their knowledge in this area. It was a good learning experience for all of us.

The team elicited requirements during weekly phone conferences with the sponsor. Meetings were effective and, combined with prototypes demonstrated to Pro Quo Books, greatly assisted the team in the requirements gathering process. Weekly agendas kept the team on track with our schedule.

Our initial elicitation on the design of the test application's user interface seemed to provide good feedback from our first two contacts at Pro Quo Books. Later when the contacts changed, we heard that it could have been done in a more clear manor.

The team prepared high quality documentation, especially code comments. However, we could have done a better job documenting architecture and design as well as how to use the test application. Documentation was regularly updated on the team's website, although website updates started to slip near the end of the quarter. This was partly due to focusing on completing the important work first and putting updating the public face of the project as a secondary concern. The team would have benefited from more time spent on initial design, especially of the test application, since it had to be revamped significantly during Spring quarter. The evolutionary delivery lifecycle model served us well, but having a stricter process could have helped with our design issues.

Implementation was steady throughout the project. Our only setback was lack of test images, but we were able to test our identification algorithms and filters on images gathered from Amazon.com in order to keep the project moving. The team met most project milestones, with one milestone being delayed by about one week. By the end of the project, we were able to deliver functioning software that met the sponsor's accuracy and time limit goals for small-scale batches of images.