Determining Bread Quality Using Machine Vision

Design III Project Report

Oliver Beaudin - 260376823
Bryan Wattie - 260313966

Presented to Dr. G. Clark
McGill University
BREE 490
April 19, 2013

TABLE OF CONTENTS

LIST OF FIGURES

EXECUTIVE SUMMARY

In this report we present the culmination of the work done as part of our final design project, in which we were tasked with developing a software program and imaging apparatus able to determine the visual quality of baguette crumb for Première Moisson. We will briefly revisit the problem definition, design objective and criteria defined in our project proposal report, as well as summarize the results of our preliminary design analysis. Next, we discuss and explain the design process of the three major components of the software program: image acquisition, image analysis and neural networks. This is followed by a discussion on the design considerations of the physical imaging apparatus, as well as the test evaluations of our prototype design. We will be explaining the specification, prototyping, testing and optimization phases of our project, but due to the fast iterative design process of our project, many of these phases overlapped. Finally, we finish with a brief look at the functionality of the user interface, which was worked on primarily as an independent design project by Sara Tawil. The final product has yet to be installed and fully validated, which will be done in the coming weeks, and will monitored to ensure functionality.

INTRODUCTION

The goal of our final design project was to design and develop an imaging apparatus and software program able to quantitatively categorize the visual quality of baguette crumb. The need for such a product was made apparent at Première Moisson, where the monitoring of various quality control factors is an integral part of their production line. How a baguette looks, and the visual texture of its inner crumb, are vital factors in dictating consumer appeal towards the product. Currently, visual quality is being inspected by a quality control expert, but there are a number of inherent issues with this method that we sought to fix and improve upon with our project. These were explained in our previous report as part of our problem definition, but they are repeated briefly here.

First, the visual quality of baguette crumb is very subjective, and can vary greatly from person to person based on their personal preferences. Because of this, it is very difficult to set up a consistent grading rubric that can be used among various people of differing opinions, meaning that it always has to be same person grading the baguettes for the grading criteria to remain constant. Furthermore, human subjectivity, even within the same individual, is not perfectly consistent or repeatable. The quality grade attributed to the same baguette might vary depending on factors such as time of day or fatigue, or the inclusion of other quality indicators, such as baguette smell, which can influence a grading decision. External influencers should be completely removed if the quality grade is to be exclusively based on visual criteria. As well, it is very hard for a human's perception and memory to track how quality changes over long time periods, or how precisely quality varies under different recipe or production parameters.

These considerations lead to our design problem definition: **the evaluation of the visual quality of baguette crumb must be done by a human expert, but it's time consuming, subjective, non-comparable and doesn't provide hard data**. Based on these issues, we set out to design a system that would be repeatable, consistent and scalable, as well as easy and fast to use. Our design objective was then to: **develop an imaging apparatus and software program able to autonomously and consistently evaluate the visual quality of baguette crumb**.

In our previous report we explained the important criteria we were designing for, the constraints we had to adhere to, and assumptions we made. The design criteria are reiterated here:

1) The program must be able to identify a baguette sample in an image and be able to distinguish it among other image objects and noise.
2) The program must be able to extract pertinent information from the baguette image, such as pore area, pore density, pore distribution, etc.
3) The program must be able to grade the baguette on a scale from 1 (worst) to 5 (best), using classifying algorithms able to emulate the grade given by a human expert.

4) The program must be flexible enough to allow for multiple baguette varieties or recipes to be analyzed.
5) The program must be interfaced with an accompanying physical imaging apparatus to facilitate the analysis of baguette samples.
6) The program must be equipped with an intuitive user interface.
7) The program must come equipped with database capabilities suitable to store and display necessary information for each variety.
8) The program must be able to run as a standalone executable application, without requiring an installed version of Matlab.

The primary focus of our previous design proposal report was to determine which classifying algorithms would be used to assign a quality grade based on the extracted feature information from the baguette image. We reviewed four possible solutions based on the categories shown in Fig. 1: user-weighted linear classifiers, multinomial logistic regression classifiers, expert systems and artificial neural networks. It was decided after careful analysis that artificial neural networks would be the best classifier to use for our given requirements. Neural networks were the most flexible of the four options, allowing the user to add new varieties easily to the database. Once enough training images have been collected, it is possible to train the network to emulate the same quality correlation that would be attributed by the expert. However, a large training set of data has to be accumulated before the correlations would be accurate. In order to implement neural networks into our program, we used the built in functions from the MATLAB® Neural Network Toolbox™ (MATLAB, 2011).

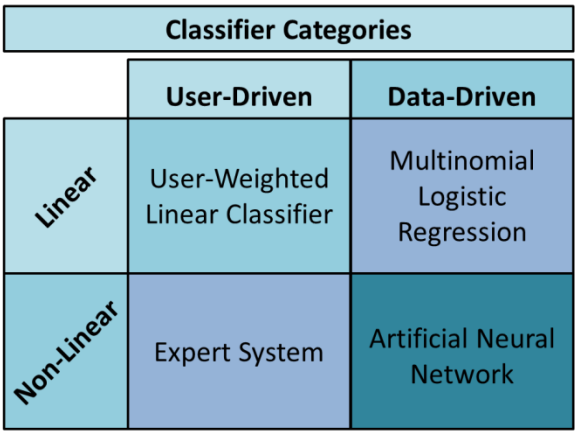

Fig. 1: Possible classifier categories.

Our design project falls into three separate categories: the image processing and analysis algorithms, the imaging apparatus, and the user interface. Each of these three components underwent their own specification, prototyping, optimization and testing phases. Only once each was in a functional state were they integrated with the other components. As such, we will be treating the design cycle

explanation of each component separately in this report. The main focus will be on the imaging program and apparatus, as the user interface was primarily worked on as an independent project in conjunction with ours by Sara Tawil.

IMAGE PROCESSING AND ANALYSIS

The image processing and analysis program was built entirely in MATLAB®. Due to MATLAB®'s workspace environment, it is very easy to perform fast specification, prototyping, testing and optimization. Because of this, there aren't very clearly defined distinctions between these four phases of the design cycle since each run of the program would effectively run through this cycle. For example, a few lines of code would be written to specify a certain action that is wished to be performed (specification). This would first be tested directly in the command window interface with some placeholder variables to see if the syntax is correct and the code is behaving as expected (prototyping). Next, this bit of code is then integrated with the rest of the program function file, changing the placeholder variables to the desired variables in the function. The entire function is run from the start to see how it performs, and determine where it breaks (testing). Finally, the bugs are detected and fixed, and the code simplified in order to speed up the function execution time (optimization).

This design process, on a miniature scale, has been repeated hundreds of times during the course of this project. Every new addition to the program requires that the program be tested to ensure that it works before moving on to the next step. As a result, this section will not be structured by explaining the work done for each design cycle stage. If we were to do so, it would be structured as follows:

Specification: the algorithms used in the program
Prototyping: testing the program without being interfaced with apparatus and user interface
Testing: testing program once interfaced with apparatus and user interface
Optimization: optimizing program based on results of testing

However, we feel that it makes more sense to offer a complete description of how the program works from start to finish. For instances where large changes were made, decided upon during the testing and optimization phases, we will describe what was changed and why these changes were decided upon.

Image Acquisition

The first step is to acquire an image of the baguette sample. Originally (during the "prototype" phase), this was just done by loading a sample image that had already been saved on the computer. When the program was interfaced with the imaging apparatus, the image would come directly from the webcam setup.

Next, it is required that the program identify the baguette sample in the image. This is done by applying a contrast threshold to separate the light and dark areas of the image. The baguette will be predominantly pale in comparison to the black surface behind it. Pixels above a certain grayscale threshold brightness are painted white, while those below the threshold are painted black, resulting in a binary (black and white) image (Fig. 2). The use of contrast thresholds are a fast and computationally cheap method to pick out certain objects in an image. Although not as robust as more complex methods

such as color, shape, edge or texture recognition, early prototyping showed that it would be adequate for our requirements.

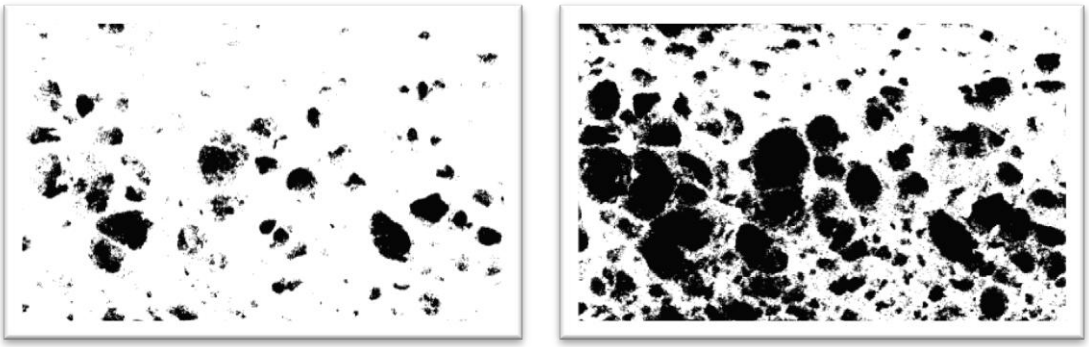

Fig. 2: Binary images with 0.4 and 0.6 contrast thresholds, respectively.

The resulting image will now have many unconnected groups of white pixels, which often arise from particulate matter on the surface (such as baguette crumbs), glare from the lighting, or areas where the camera picks up parts of the imaging apparatus other than the black surface. It is important that the program select the correct group of pixels (those belonging to the baguette) and ignore all the others. This is done by determining the area of all white objects and selecting the largest one, which will correspond to the baguette sample. In order to ensure this is accurate, the *imfill(holes)* command is first used to fill in all holes in an image object (MATLAB, 2011). For example, the pores in the baguette will show up as black and not be counted towards the area of the baguette, but these will become white once the *imfill(holes)* command is used.

Once the largest object is selected (the baguette) the *regionprops* command is used to extract certain properties about that image object (MATLAB, 2011). The important properties extracted are the orientation and bounding box. The orientation finds the angle of the object's major axis above the horizontal axis of the image. If the baguette was put into the apparatus crooked, this will straighten the image. The bounding box specifies the pixel coordinates that creates a box that will completely enclose the object, enabling the original image to be cropped to only include the baguette object. This greatly reduces computation time and memory requirements because the image size is minimized as much as possible. The cropped version of the original image is now passed onto the next function.

Image Analysis

Now that the baguette sample has been identified and the image re-oriented and cropped, it is possible to extract the important image feature information. This is once again done using the contrast thresholds. Originally a threshold value was specified manually in order to obtain the best results, but once we started testing with the imaging apparatus, we realized that there was too much variation in lighting for this value to work consistently. Instead, we decided to use the *graythresh* command to minimize the variation between the black and white pixels in an image (MATLAB, 2011). What this does

is find a contrast threshold value that best balances black and white for a given image. We had to manually adjust the multiplication factor until testing showed it gave consistent results.

The baguette surface now appears white, and pores appear black. This is working under the principle that since the pores are deeper than the surface, they will be darker due to the shadows. The lighting requirements and design considerations to emphasize these pore features will be further discussed in imaging apparatus design section. Originally we used several contrast thresholds in order to find an equivalent pore "depth" (Fig. 3). However, once we begun testing the system, we realized that our lighting doesn't adequately emphasize pores to enable the determination of depth. Furthermore, this greatly added computational requirements to process additional image layers. This was removed during a large optimization sweep and the program began running between 5-10 times faster.

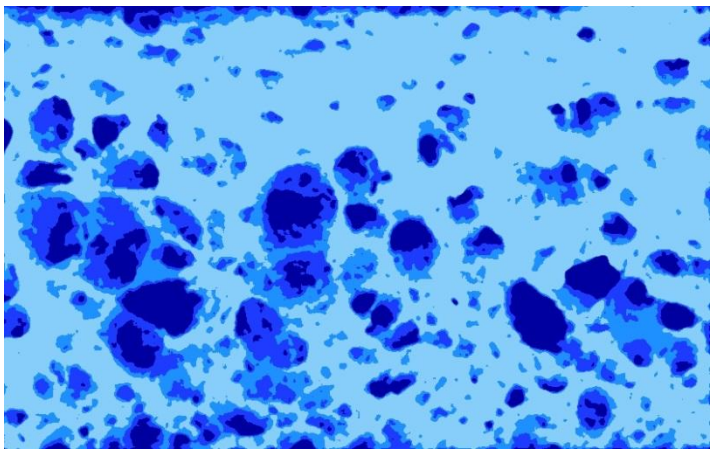

Fig. 3: Image with approximate pore depth.

It is also important to ensure that no junk data is included in our analysis. For example, it is possible for the program to confuse parts of the background surrounding the baguette as being pore features. This is because during the analysis, any black object is considered a pore. However, this is solved by disregarding any object touching the edges of the image. What this does is ensure that only black objects (pores) *within* the baguette object are considered. Furthermore, the area of each pore was determined, and pores that fell below a certain minimum size (ex: 20 pixels) were excluded from the analysis. The reason for this is that it is possible to get a lot of image noise which doesn't add any valuable information to the analysis, but will significantly increase the computational time, as each image object, no matter how small, would have to be analyzed. This had previously been done (during the prototype phase) by applying smoothing filters to the image to eliminate noise. It was later determined that these filters were actually eroding the image, and valuable data such as the exact pore area were being altered. Once these filters were removed and replaced with the minimum-size screen, the program begun running faster.

The information for the location and size of each pore is determined using the *regionprops* command (MATLAB, 2011). With this information it is now possible to determine the parameter values

that will be used in the neural network. We had to decide on parameters that would be able to meaningfully describe the features of the baguette surface and are somewhat independent from each other. We decided on the following:

Pore density
Pore area / total area
Average pore area
Pore area coefficient of variance
Pore distribution

As can be seen, the first four parameters are very similar, although they each describe a slightly different feature consideration. *Pore density* is an indicator of how many pores there are on the surface. Typically, high pore density will be caused by baguettes with many small pores, while low pore density will be caused by fewer larger pores. However, it is also possible to have baguette with only a few small pores and vice versa. This factor is accounted for by the *pore area / total area* parameter which is a ratio of the area consisting of pores over the total baguette area. This allows the neural network to differentiate between small and large pores, even if the pore density is the same between two different samples.

Average pore area gives an approximate estimate of the pore sizes, while the *pore area coefficient of variance* gives a good indicator of the distribution of pore sizes. For example, it might be desirable for the pores to be as similar in size as possible if the average is below a certain size. Finally, *pore distribution* is a measure of how the pores are distributed with respect to the centerline. For example, a value of 0.5 means that the pores are evenly distributed over the surface of the baguette, while a value of 0.25 means the pores are concentrated near the middle, and a value of 0.75 means they are concentrated near the edges. This was calculated by taking the average distance of all pores to the centerline, divided by half the width of the baguette to obtain a value that ranges from 0 to 1.

Despite there only being five quasi-independent parameters, it is still possible to obtain a very large set of different potential combinations. The neural network will be able to correlate these different combinations with the quality value associated with the sample during the training phase, and correctly assign newly analyzed samples.

Neural Networks

As was previously mentioned, we are using the built-in neural network commands from the MATLAB® Neural Network Toolbox™. As such, we aren't concerned with the internal workings of the neural networks (we assume they have been validated by MathWorks®), only how to properly set them up and use them. A neural network has three main "visible" components: the *input array*, the *target array* and the *net*. When creating a new neural network, it is assumed that the training data set has already been defined. The *input array* consists of all the samples (columns) that make up that variety and the parameters (rows) assigned to each sample, as shown in Fig. 4.

The *target array* consists of all the samples (columns) within that variety, and the target category (row) assigned to each sample. The target category will be assigned a 1 while all others are assigned a 0, as shown in Fig. 5. Each category is assigned a particular outcome. In our case, this is simply the quality grade, ranging from 1 to 5. If the fourth row has a 1, for example, then that sample was graded a quality value of 4. However, since neural networks use categorical classification, the target array simply signifies which category that sample falls within. It's therefore possible to assign string names to particular categories, as is often done when neural networks are used as means to diagnose patients in medical cases.

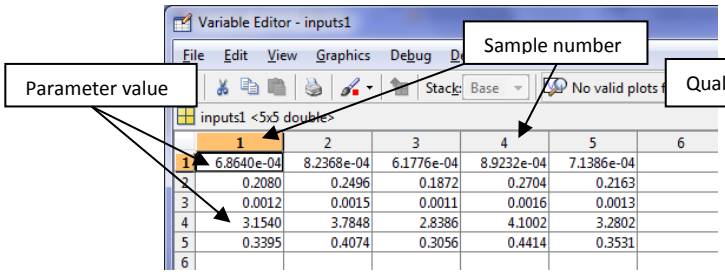

Fig. 4: Sample neural network input array.

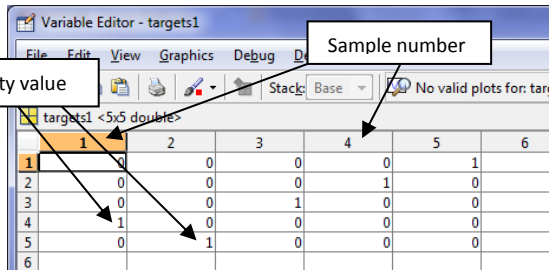

Fig. 5: Sample neural network target array.

Finally, the *net* is the neural network function that will correlate the inputs with the targets. An empty net is first defined and created. The input and target arrays from the training data set are then inputted into the *train* function, and the neural network is trained using 70% of the training set parameters. 15% is used for validation of the network, and the remaining 15% is used as independent testing of the network. Once the training function has converged on a network that satisfies the validation and testing requirements, the *net* variable is saved and stored along with the training data for that variety.

When a sample of a specific variety is to be analyzed, the program loads the neural network associated with that variety. Once the image is taken and processed, the image features are sent as an input array into the neural network. This then returns a target array based on the correlations, and the quality value of the sample can then be displayed.

So far we have been able to test the system and ensure that the neural networks are being trained, saved, accessed and used properly for each variety of baguette. However, we have not yet been able to validate the classifying performance of the neural networks because that would require inputting a very large training data set which we've not yet had the time to do. This validation step will take place once we install the system at Première Moisson.

IMAGING APPARATUS

In this section of the report we will review the process that we went through in developing the physical apparatus for image acquisition. We have three 3D models made using Google SketchUp, and we built two physical prototypes. We will go through each model and its respective prototype in a chronological order. Although the design cycle was used, some phases occurred simultaneously. Our main goal as aforementioned was to acquire images in a controlled, consistent and repeatable fashion.

In the first design drawing (Fig. 6) we identified the basic functions that we needed to include, and used the drawings to communicate with our industry partner Claude Seanosky at Première Moisson, and receive feedback from Scott Manktelow on material choice, and construction techniques. 3D modeling allowed us to continuously return to the synthesis step of the design cycle, adding or changing aspects of our design, such as the lighting equipment. These discussions and research are analogous to the synthesis and evaluation components of the design cycle.

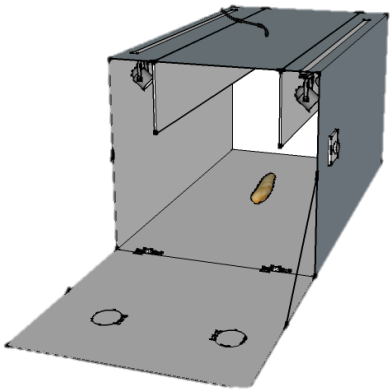

Fig. 6: Preliminary imaging apparatus design.

The first prototype is where we first addressed the specification and prototyping aspects of the design process. There were two main aspects of our design that needed specification. We needed the height of the apparatus so as to capture the whole baguette in the field of view of the camera, and the lighting of the sample. The height was determined experimentally as we built the prototype, and the lighting was tested by trial and error. Then we gathered many images for testing.

After reviewing the test results we made the necessary changes to our model and built a new prototype. For our processing algorithms to be robust, we needed to acquire images in a controlled, and repeatable fashion. Thus our goal was to design and build an apparatus that would house a camera in a fixed position, deliver the same amount of light at the same angle, and placed the bread in the same position and orientation. Moreover, we wanted the apparatus to be easy to load and reload.

In developing our first design we used Google SketchUp to produce a 3D model to identify the necessary functions, discuss equipment choices, and their placement. We knew there needed to be a lighting source, and some form of deflection to prevent direct illumination. To acquire the necessary

information from the bread surface, adequate contrast is need to find the differences between the two phases in the image (Hall & Bracchini, 1997). To enhance the contrast we needed a diffuse light source, with as little direct light on the surface the bread. In the first design drawing the panels that extend from the top piece of the apparatus are to deflect any light that might shine directly on the sample. To control the intensity of light within the apparatus we included a dimmer switch, this would allow even more user control over the imaging parameters. We needed a surface for the bread that clearly delineated the bread contour, and did not reflect light and create interference. At this point in our design process we had little notion as to the actual dimensions of the apparatus as we did not yet know the standard baguette length, or how high our camera would be placed.

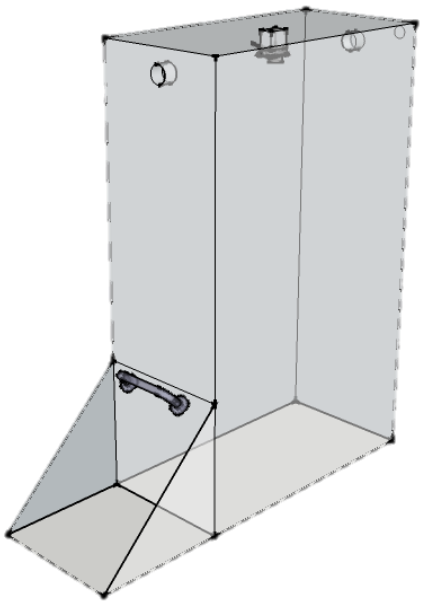

Fig. 7: Design of first imaging apparatus prototype.

Dimensions

Preceding the construction of our first prototype following the first model, we needed to determine our apparatus dimensions. The length of the apparatus was determined by consulting with our industry partner. Their baguette product has very little variation in length, as their process is highly automated and precise. Given a baguette length of 55 cm we added an additional 5 cm of length to the apparatus to ensure that there was ample space for the baguette to be placed. Once we determined the length of our imaging area we were also able to determine the height. The only constraint on the height was that the camera needed to capture the whole length in its field of view. For loading and unloading of the sample we needed a door. In the construction of our first prototype we added an angled extension to the walls so that the door could rest without the need for a latch.

Lighting

Two lights were placed at each end of the apparatus at a perpendicular angle to the imaging surface. A third light was placed on the roof beside the camera facing the imaging surface. We used three 20W halogen lamps that were encased in a screw mountable shell. These lights far outweighed other options. They were inexpensive at 10$ a unit. The set of three we used came with an electrical plug eliminating the need for any circuit work, they simply plug into a standard 120V outlet. However, these lights did not have the dimming capability that we had included in our first design drawings. Other options with this capability were much more expensive, and while controlling the intensity of light is desirable as a perfect intensity could be obtained, it can also be a liability. If the user has control over light intensity we would have needed to add controls to our image-processing algorithm that adjusted for light intensity, and include sensors further raising the cost of the project. Otherwise the process would not have been consistent in its conditions affecting the comparison from sample to sample.

Fig. 8: 20 W halogen lamps used for lighting.

Materials

To build our apparatus we chose 5/8” medium density fiberboard, as it is inexpensive, readily available and easy to work with. Its finished surface suited our project well, as it is ideal for paint, and provided a smooth finish with little disruption to the lighting for image acquisition. Assembly was also fairly quick; we used wood glue instead of screws to fasten the pieces together. The main drawback of this material choice was that our prototypes were quite large making them heavy and difficult for one person to move.

We used a matte black paint to paint the entire interior surface of the first prototype. Matte black provides a distinct contrast with the colour of the bread, and causes little reflection or glare, thus improving image quality.

Design Evaluation

Our first prototype was then paired with a preliminary user interface and sent to Première Moisson to collect our first set of images. After a week of use they had amassed several photos of each quality category. Unfortunately, when we applied our image processing algorithms to these photos we

found that the quality was very poor, and much of the detail we needed in the bread crumb was being bleached out by our lighting arrangement.

Upon visual inspection of the image and the processed result, it is clear that there is too much incident light shared by the pore and the surface giving them very similar color properties. Thus our processing algorithm cannot distinguish between the two. The pores are highly eroded, their calculated areas are much smaller than their actual areas, and small pores are being missed entirely.

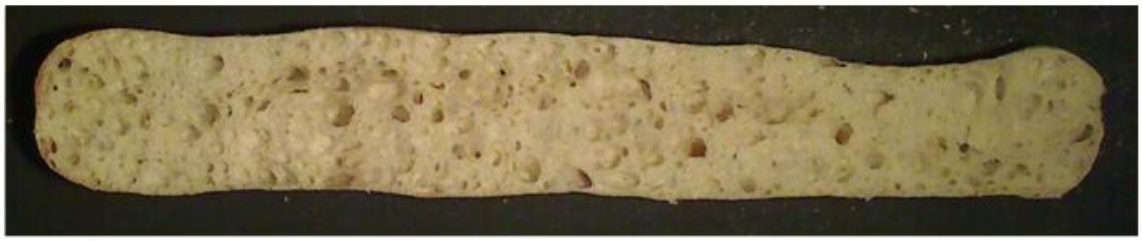

Fig. 9: First prototype, original image.

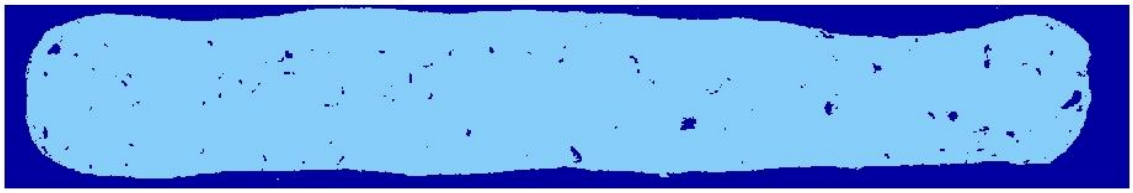

Fig. 10: First prototype, processed image.

These problems were very critical and needed to be addressed, thus leading us to our second design and prototype.

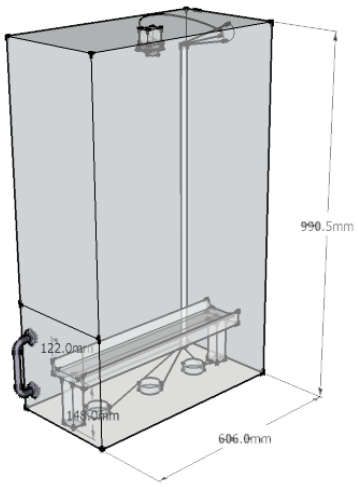

Fig. 11: Design of second imaging apparatus prototype.

From the results of our first prototype we knew that our main issue was the lighting of the sample. The details of the bread crumb pore structure are most easily extracted when there is a large contrast in color from the surface to the pore. In our previous prototype the contrast was not sufficient. To enhance the contrast we made one key change, and a few improvements.

To improve the contrast we moved the lighting underneath the sample and allowed it to reflect through the box. To enhance the reflection we chose not to paint the entire interior surface with matte black paint. In this prototype we only painted the surface of the trays and a small rectangle on the door and back wall to make sure that if that sample was placed Furthermore we needed to increase the height of the apparatus to be able to place a tray for the sample over the lights while not blocking the light too much. The tray also included beveled edges to ease placement and ensure the bread was fully surrounded by a black background.

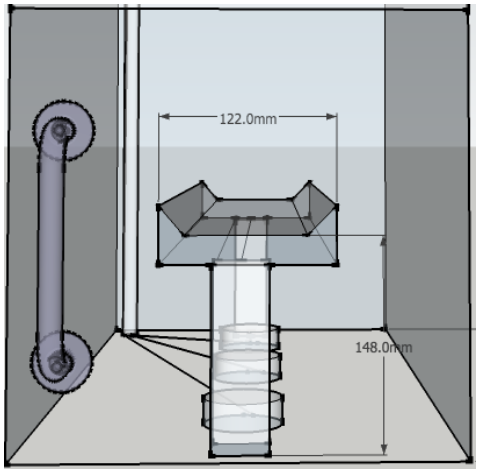

Fig. 12: Front view of second prototype.

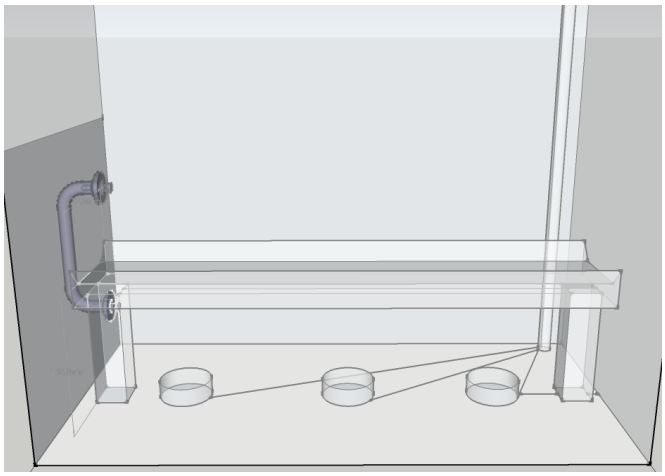

Fig. 13: Side view of second prototype.

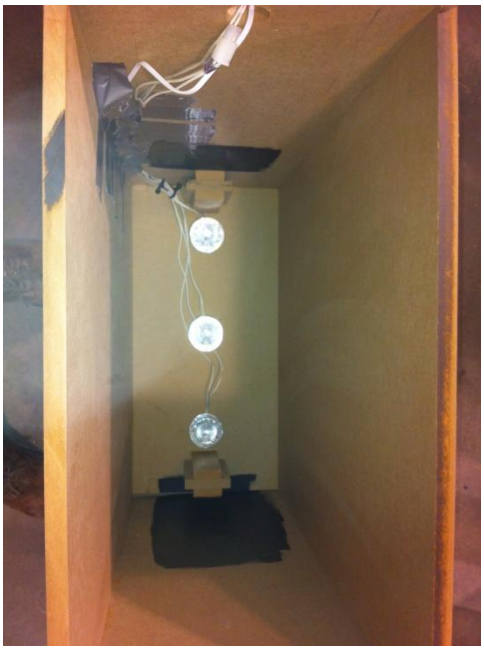

Fig. 14: Top view of lighting arrangement.

Fig. 15: Lighting with bread sample.

Another change we made was to the door. We decided to remove the gravity rest and make a sideways swinging door. However, fiberboard made hinge placement for the doors challenging as it is prone to splitting. To place screws on the sides we used very small screws as the door is small and the screws did not have to bear a large load. The screws were also placed far enough from the edges so that splitting would not reach the corners. When splitting did occur the cracks were filled with wood glue and clamped to prevent further splitting and secure the hinges.

Fig. 16: Second prototype, original image.

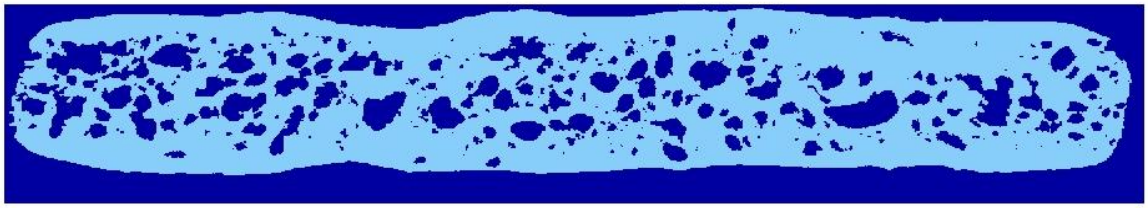

Fig. 17: Second prototype, processed image.

The image quality given by our second prototype was distinctly improved. Now even smaller pores are being detected. There are still some small issues with the results. For example in Figure 9 we can see that the top left corner has been interpreted as one big pore when in fact there are a few pore within a larger depression. This is likely an issue with our processing algorithm and it should be easily fixed, however should it be unresolvable it may not be a prohibitive error as we are using artificial neural networks and if the error is constantly repeated perhaps this flaw in the data will not sway the quality result too much. While we continue to improve our algorithms, we now have sufficient image quality given the physical parameters, and have met our original design goals for the physical apparatus.

USER INTERFACE

Although the baguette analysis program can be run directly from the MATLAB®'s work environment, this can be a very tedious, slow and complicated method of running the program. The need for an intuitive user interface is essential if the program is to be used by people unfamiliar with MATLAB®. The user interface was primarily worked on by Sara Tawil as part of her BREE 497 Independent Design Project. This section will only provide a brief overview of the features and functionality of the user interface, but will forgo a description of its development cycle and inner programming.

The interface consists of various panels and tabs, as can be seen in Fig. 18 & 19. The panel in the top left corner, titled "à Remplir", is used to input the batch ID number, the bread variety (or recipe), and other useful analysis information such as the length, width, height and weight of the bread sample. The panel in the lower left corner, titled "Ajouter/Supprimer", is used to either add a new variety (or recipe) to the system, or delete an existing one.

The rest of the interface consists of the tabbed panels "Video/Image", "Résultats" and "Database" (which will be translated to "Base de Donnés"). In the video/image tab there is a direct video feed from the webcam installed inside the imaging apparatus. When an image is captured ("Prise d'image") a cropped image appears. The user can keep adjusting the sample and taking pictures until they are pleased. Next, the user chooses between either analyzing the image or adding it to the database.

If the user has created a new variety, it is required that they add images to the database to build up the training data set. When adding to the database, the user is prompted to enter a quality value, from 1 to 5, for the baguette sample. When this is done, the image is sent through the analysis program to determine the baguette features, and this data is saved in an input array for that variety. As long as samples are being added to the database, the input and target arrays for that variety are continuously updated. Once enough samples have been entered to constitute an adequate training data sample size, the user clicks on the "Train" button in the lower right corner. This will create a new neural network for the chosen variety based on the input and target arrays for that variety. The user is able to continuously add more data to the database, and retrain the neural networks each time.

If a database (and neural network) for a variety already exists, then the user can simply click on the image analysis button. When this is done, the image is sent through the analysis program and the feature parameters are then sent through the neural network for that variety. The interface then automatically switches to the results tab and displays the quality value for the baguette. This information is not saved to the database, but the user can choose to do so by returning to the video/image tab and adding it. The results tab will display both the original cropped image, as well as the processed binary image so that the user can get a visual indication of how the program performed.

There is still some work that has to be done before the user interface is completely ready to go. The results section will also display a histogram to the right of the processed image displaying how this

sample's quality compares to the others of this variety. Furthermore, some statistical information from the image analysis will be displayed. This will include: the total pore area in mm^2, the pore area to total area ratio in percentage, the centreline distribution ratio, the total number of pores, and the distribution of area, divided into bins (ex: 10-20 mm^2, 20-30 mm^2, etc.).

In addition to this, we will also be including a logbook features that will write the results of the image analysis to an excel file, along with tracking information such as time and date, batch ID, variety, etc. This information can then be extracted from the excel file and used by Première Moisson to conduct more in depth analysis of baguette quality, either over time or over different varieties and recipes.

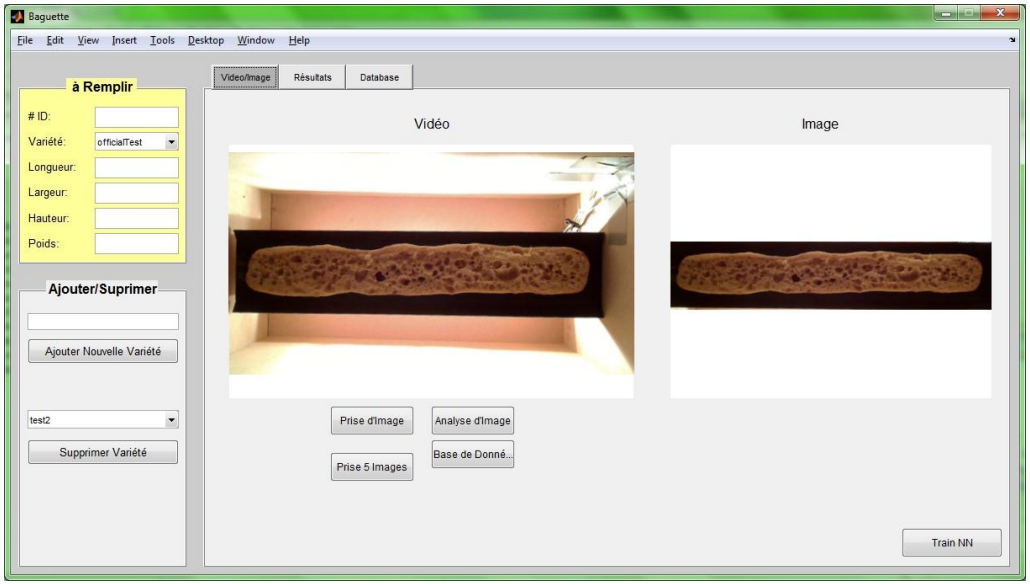

Fig. 18: User interface with video/image tab.

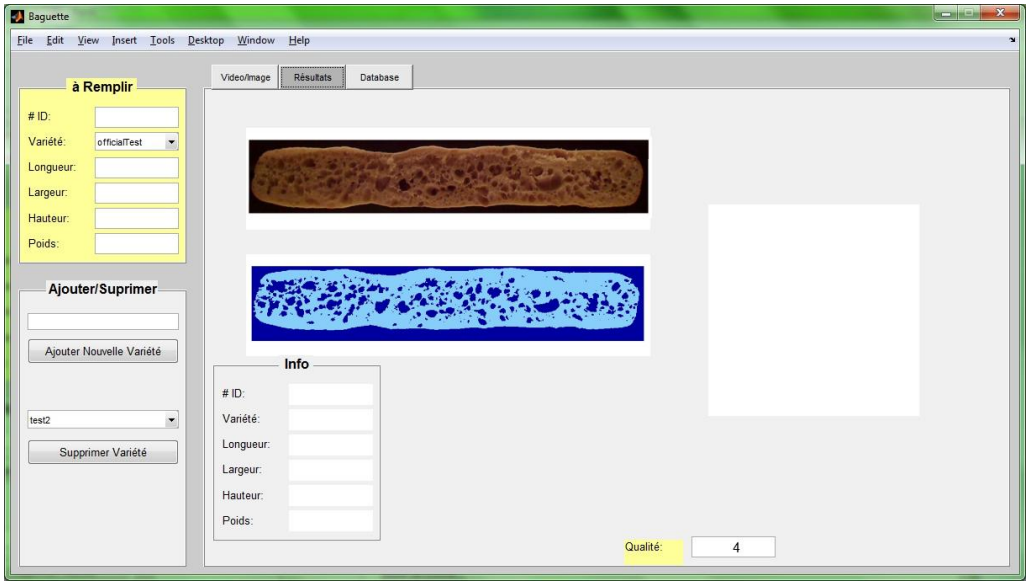

Fig. 19: User interface with results tab.

CONCLUSION

The scale and computer software oriented nature of our project gave us the opportunity to produce rapid iterations of the design cycle. As we move forward there will always be improvements to be made and features to be added. Our design is not entirely validated, as we still need to have it implemented and monitored at Première Moisson. Once in full use with a large data set it will be important to verify that the imaging apparatus is taking consistent images with repeated use, and that the artificial neural network has sufficient training data to make predictions that are accurate.

From this point forward, we are also currently working on including different functionality in the user interface, including a histogram of past data for comparison, a log book to record use, and the ability to process 5 different images and return an average score. Throughout the summer one of our group members, Bryan Wattie, will work with Première Moisson to integrate this design with their quality assurance protocol.

ACKNOWLEDGEMENTS

The members of our group would like to thank Dr. Grant Clark for his guidance, support and constructive criticism. As well as his continued enthusiasm! A big thank you to Claude Seanosky at Première Moisson for his patience, expert knowledge, helping us to run tests, and providing us with plenty of baguettes! Scott Manktelow has also been an invaluable source of shop help and advice. Thank you!

Finally, a big thanks to Sara Tawil, without whom the final project implementation might not have been possible.

REFERENCES

Bertrand, D., Le Guernevé , C., Marion, D., Devaux, M. F., & Robert,P. (1992). Description of the textural appearance of bread crumb by video image analysis. Cereal Chemistry , 69, 257–261.

Coles, G. D., & Wang, J. (1997). Objective determination of bread crumb visual texture by image analysis. In J. L. Steele, & O.K.Chung(Eds.),Proceedings of the international wheat quality conference (pp. 153–157). Manhattan, KS: Grain Industry Alliance.

Kvaal, K., Wold, J. P., Indahl, U. G., Baardseth, P., & Næs, T. (1998). Multivariate feature extraction from extraction from textural images of bread. Chemometrics and Intelligent Laboratory Systems, 42, 141–158.

Lassoued, Nejla, et al. (2007). "Granulometry of bread crumb grain: Contributions of 2D and 3D image analysis at different scale." Food Research International 40.8: 1087-1097.

MATLAB. (2011). *Help Documentation*. MATLAB version 7.12.0 R2011a. The MathWorks Inc., Natick, Massachusetts.

APPENDIX – OPTIMIZED CROPPING AND FEATURE EXTRACTION CODE

```
%%----------------------------------------%%
SAMPLE IDENTIFACTION AND CROPPING PROGRAM
%%----------------------------------------%%


function [pic13, maxi] = bread_cropper(pic1);
% Takes the original image and crops it to only have the bread sample.

% determines the thresh value needed to identify the bread sample
thresh = graythresh(pic1)*.8;
pic2=im2bw(pic1,thresh);

pic3=imfill(pic2,'holes');

%find areas and orientation of each object
cc1=bwconncomp(pic3);
rr1=regionprops(cc1,'Area','Orientation');
%loop to extract the area info for the objects
rr1size = size(rr1);
for k = 1:rr1size(1);
    rrr(k,1) = rr1(k).Area;
end
%figure out which object is the largest
[maxi,ind]=max(rrr);

pic4=imrotate(pic3,-rr1(ind).Orientation);

cc2=bwconncomp(pic4);
rr2=regionprops(cc2,'Area','Centroid','BoundingBox','MajorAxisLength','MinorAxisLength','Orientat
ion');
% BoundingBox = [x-topleft, y-topleft, x-width, y-width]
% Orientation = angle between x-axis and major axis of bread

%find max for new rr
rr1size = size(rr1);
for k = 1:rr1size(1);
    rrr(k,1) = rr1(k).Area;
end
%figure out which object is the largest
[maxi,ind]=max(rrr);

%bounding box coordinates
bounding_box(1,:) = [rr2(ind).BoundingBox(1),rr2(ind).BoundingBox(2)]; %top-left corner
bounding_box(2,:) = [rr2(ind).BoundingBox(1)+rr2(ind).BoundingBox(3),rr2(ind).BoundingBox(2)];
%top-right corner
bounding_box(3,:) =
[rr2(ind).BoundingBox(1)+rr2(ind).BoundingBox(3),rr2(ind).BoundingBox(2)+rr2(ind).BoundingBox(4)]
; %bottom-right corner
bounding_box(4,:) = [rr2(ind).BoundingBox(1),rr2(ind).BoundingBox(2)+rr2(ind).BoundingBox(4)];
%bottom-left corner

pic12 = imrotate(pic1,-rr1(ind).Orientation);
pic13 = imcrop(pic12,rr2(ind).BoundingBox); % the original image cropped
end
```

```
%%-----------------------------------------%%
SAMPLE FEATURE EXTRACTION PROGRAM
%%-----------------------------------------%%


function [pic5,input_matrix] = porefinder_clean(pic1,total_area)
% This is the main function that process the image, connected to bread_cropper

format short g

imsize = size(pic1);
x_length = imsize(2);
y_length = imsize(1);

% reads the cropped image and displays it
imshow(pic1), title('Original image, cropped')

%%% MAKE THE THRESHOLD MULTIPLIER A VARIABLE THAT CAN BE ADJUSTED
% determines good threshold amount
thresh = graythresh(pic1);
thresh1=thresh*.8;

%converts to binary image
pic2=im2bw(pic1,thresh1);

%inverts the black/white of the image
pic3=zeros(imsize(1),imsize(2));
black1=find(pic2==0);
pic3(black1)=1;

% fills in the wholes of the objects, determines connectivity
pic4=imfill(pic3,'holes');
cc1=bwconncomp(pic4);
rr1=regionprops(cc1,'area','Centroid');

% turns composite image into a color-mapped image
pic5=ones(imsize(1),imsize(2),3);
numb=imsize(1)*imsize(2);
tier0=find(pic4==0);
tier1=find(pic4~=0);

% assigns dark blue to pores
pic5(tier1)=0/255;
pic5(tier1+numb)=0/255;
pic5(tier1+2*numb)=160/255;
% assigns light blue to non-pores
pic5(tier0)=135/255;
pic5(tier0+numb)=206/255;
pic5(tier0+2*numb)=250/255;

% converts structure array to normal array
rr1Length = length(rr1);
% pulls out info from the rr1 structure arrays
% makes it easier to pass on data
% columns: area, x-centroid, y-centroid
for k = 1:rr1Length
    r1(k,1) = rr1(k).Area;
    r1(k,2:3) = rr1(k).Centroid;
    r1 = round(r1);

end

%%% FIND OBJECTS TOUCHING EDGES OF IMAGE, TO BE REMOVED
pixlist = regionprops(cc1,'pixellist');
jj = 1;
for k = 1:rr1Length
    pix_index = pixlist(k).PixelList;

    x1 = find(pix_index(:,1) == 1); %1 or last x
```

```
    empt1 = isempty(x1);

    x2 = find(pix_index(:,1) == x_length); %1 or last x
    empt2 = isempty(x2);

    y1 = find(pix_index(:,2) == 1); %1 or last y
    empt3 = isempty(y1);

    y2 = find(pix_index(:,2) == y_length); %1 or last y
    empt4 = isempty(y2);

        if empt1 == 0
        to_remove(jj) = k;
        jj = jj+1;

        elseif empt2 == 0
        to_remove(jj) = k;
        jj = jj+1;

        elseif empt3 == 0
        to_remove(jj) = k;
        jj = jj+1;

        elseif empt4 == 0
        to_remove(jj) = k;
        jj = jj+1;

        end
end

% removes objects touching edges (makes area 0, so that it is filtered out
% in next step, and not passed on
for k = 1:length(to_remove)
    edge_detected = find(r1 == to_remove(k));
    is_empt = isempty(edge_detected);
    if is_empt == 0
        r1(to_remove(k),1) = 0;
    end
end

%%% FILTER TOO SMALL PORES (REMOVE THE PREVIOUS FILTERING?)
% removes max and min area values (pores too big or small)
% WARNING: make sure to exclude area of 0
t1max = 6000;
t1min = 20;
% finds where in the array the values fall between the max and min
% includes this into new array that only has the values of interest
r1f = find(r1(:,1) >= t1min & r1(:,1) <= t1max);
tier1_pore_info = r1(r1f,:);    % pore info to be used in neural network



%%% INPUT VALUES FOR THE NEURAL NETWORK

% PORE NUMBER/DENSITY (number/area - normalized)
tier1_pore_number = length(tier1_pore_info(:,1))/total_area;

% PORE AREA / TOTAL AREA (indirect pore density)
tier1_pore_density = sum(tier1_pore_info(:,1))/total_area;

% AVERAGE PORE AREA (as proportion of total area)
tier1_pore_area_avg = mean(tier1_pore_info(:,1))/total_area;

% PORE AREA COEFFICIENT OF VARIANCE
tier1_pore_covar = std(tier1_pore_info(:,1))/mean(r1(:,1));

% PORE DISTRIBUTION (absolute y-position to center)
% disp('PORE DISTRIBUTION:')
% not based on pore size
```

```
% ~0.5 evenly distributed
% ~0.25 concentrated in middle
% ~0.75 concentrated near edges
% imsize = size(pic1);
height = imsize(1);
centerline = height/2;
tier1_pore_dist = mean(abs(tier1_pore_info(:,3) - centerline))/centerline;

% puts the values into a matrix to saved with the neural network matrix
input_matrix = [tier1_pore_number;
    tier1_pore_density;
    tier1_pore_area_avg;
    tier1_pore_covar;
    tier1_pore_dist];

end
```