Definition

Project Overview

During several years, I've worked in the 3D content creation field. One of the most challenging stages I faced in this area was the creation of seamlessly repetitive textures. In the majority of the cases, the textures created were used to represent walls and floors made of blocks or tiles

To create the textures, I received several pieces of information which included the next:

- Dimensions of the blocks/tiles
- The layout to be created. In some cases, I had to create it.
- Pictures of the individual blocks/tiles

The workflow of this stage was pretty straightforward. In summary, the process was the following:

- Create the layout when it was not provided.
- Clean the individual blocks. That means removing the background of the images, so just the block was visible.
- Use the cleaned blocks to fill the layout.
- Use the repetitive pattern as texture

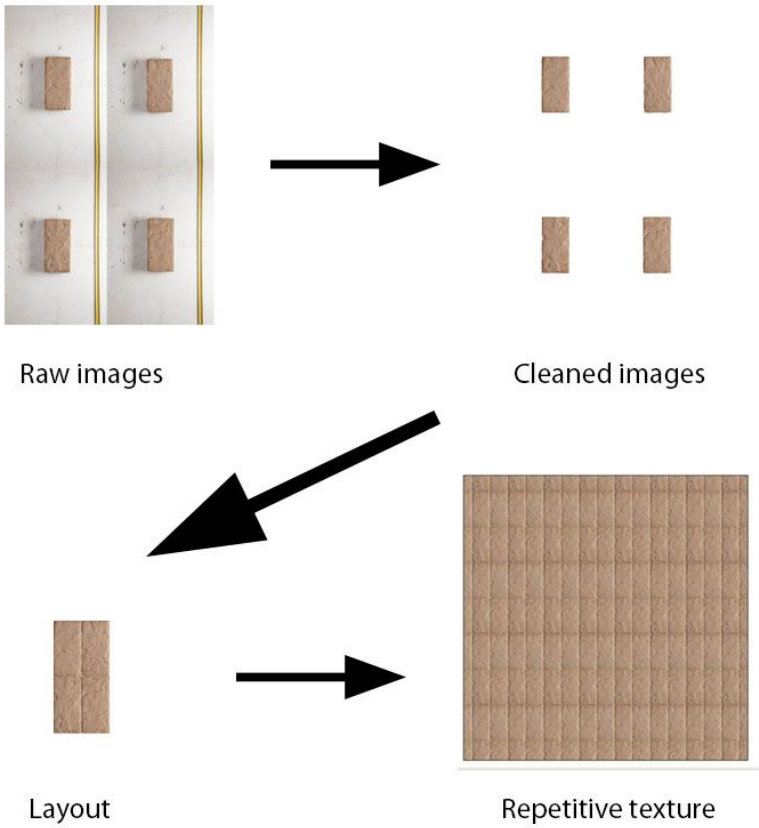

Image 1. Texture creation process

Using the tools provided by the image edition software I've used (Photoshop), I was able to automate most of the tasks in the workflow, therefore improving my performance and delivering results as fast as possible.

However, there were other task that were not possible to automate and required additional work. One of those tasks was cleaning the individual blocks.

This task was by far the most time consuming in the whole process, I'd say that 60% of the time was consumed in this step. It was a real problem because the tools provided by the software were not enough to speed this process up.

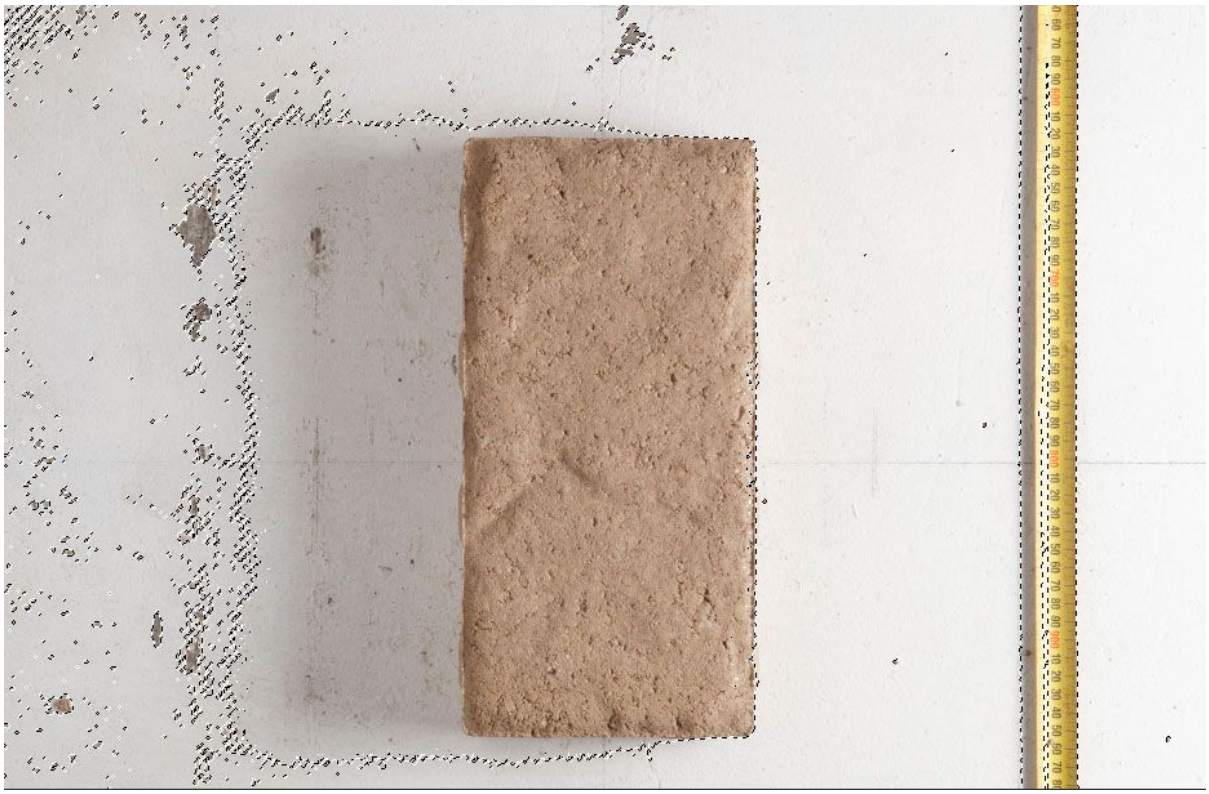

Image 2. Background selection with Magic wand tool (Photoshop)

Image 2 shows an attempt to select the background of the image with the Magic wand tool. The dashed lines show the sections of the image that are selected. As seen the tool is not able to select the background properly. For example, there is a big area in the left that should be selected, but it is not.

Based on that, I had to do it the hard way. Basically, I had to define the boundaries of the block by myself (this part was really tedious) and from that remove the background, as shown in Image 3.

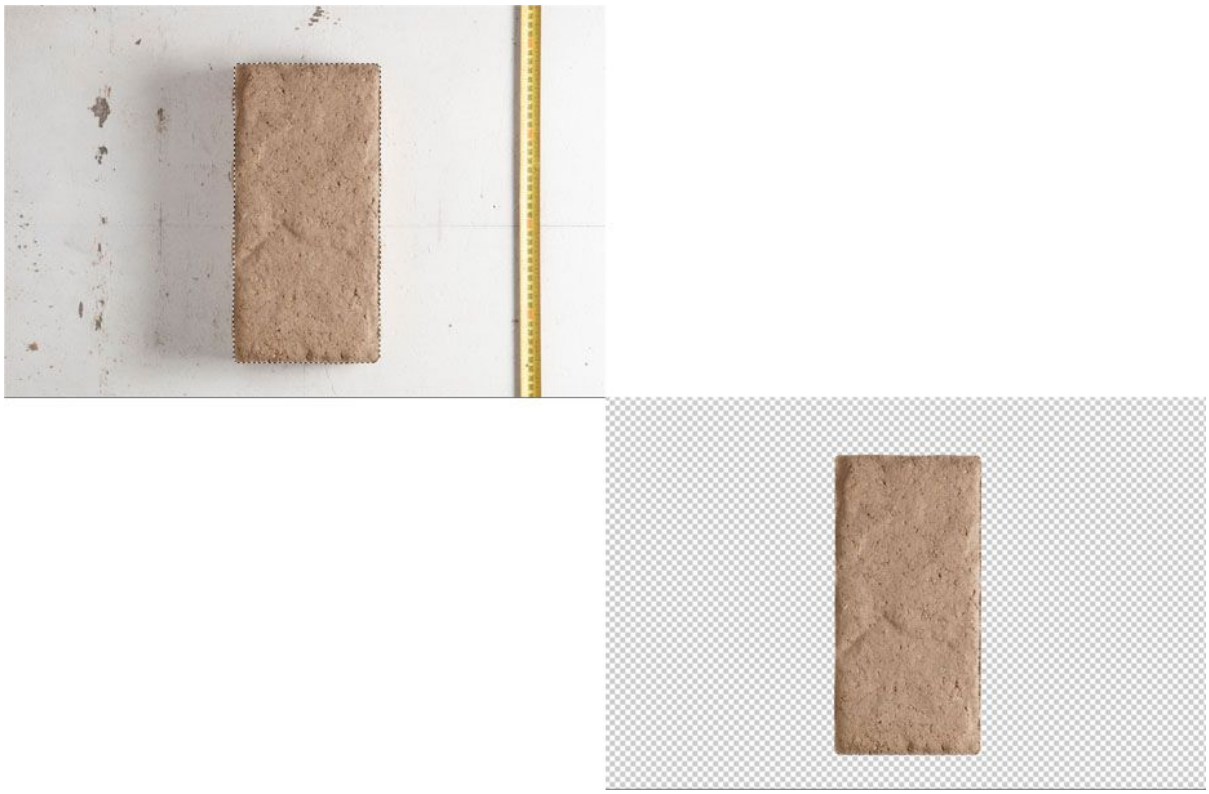

Image 3. Manual selection and background removal

I tried several ways to make that task easier with the tools provided by the software, but none of them worked as expected.

Problem Statement

In this project, I'm going to deal with pictures of blocks. Those photos were taken specifically to create the layouts mentioned above. However, the images need to be processed before use them in the layout pattern. This processing stage consists in cleaning the images by removing their backgrounds, as shown in Image 3. Given that, the objective of the project is defined as the following:

Classify the pixels of an image as background or foreground

In order to accomplish the objective, I need to use a supervised learning method to categorize the pixels of the image. As stated in the objective, I will have two categories of pixels:

- Background
- Foreground

Since I have the images cleaned up manually by myself, I could use them in the classification process to train the model. The cleaned up images shows the background as a

transparency, so in this case, I would use this information to define which pixels in the raw image belong to the Foreground category, and the rest would be categorized as Background

However, in the majority of cases, I won't have that advantage (the result image), so I have to define another strategy to classify the pixels. In this case, the best approach will be to manually define areas within the raw image that corresponds to Background and Foreground categories and from that point classify the rest of the pixels in the image.

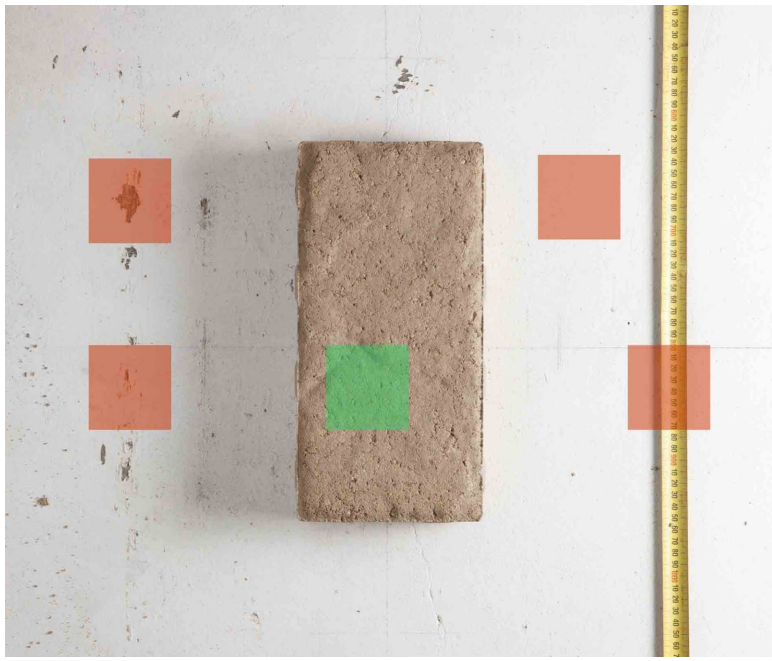

Image 4. Manual selection and background removal

Image 4 shows the base spots that will be used to classify the pixels in the entire picture. Green spot will define the Foreground category, red ones will define the Background category. Given that the background of the image is heterogeneous (there are lighter areas, shadows, and even a measuring tape), it will be necessary to select more than one spot (red ones) to define the Background category.

Metrics

Given the objective of the project, a classification metric is needed. The photos provided were taken specifically to identify the block in it, so there's a clear visual difference between the background and the foreground, for this reason is important that the pixels are correctly labeled, therefore I will use accuracy as the metric of performance.

Accuracy is a metric that may hide some inconsistencies in the classification because it relies only in the values correctly classified as positive or negative (foreground and background in the current project).

For example, in a set with 1000 pixels with 900 of them as background ones and the rest as foreground ones; 890 of the former ones and 50 of the latter ones are classified correctly. In

percentage terms, that is 98.89% and 50% (not really good for foreground pixels), however, the accuracy is 94%.

As seen, accuracy may throw values that hide issues in the process. Also, there are other metrics that can do a better job by evaluating the elements incorrectly classified (F1 score, precision, recall). Nonetheless, analyzing possible sources of errors, I have encountered that the definition of the background and foreground spots is key to make a good classification process, therefore having a reliable accuracy score.

When a pixel can be classified incorrectly? Given the features to be analyzed (explained in the next section), this can happen when there is no a clear definition of the background and foreground spots, which depends completely on the user. Next image shows a very simple classification scenario where the two categories are defined as ranges. As seen, the ranges are overlapped, therefore leading to problems when classifying pixels in that section.

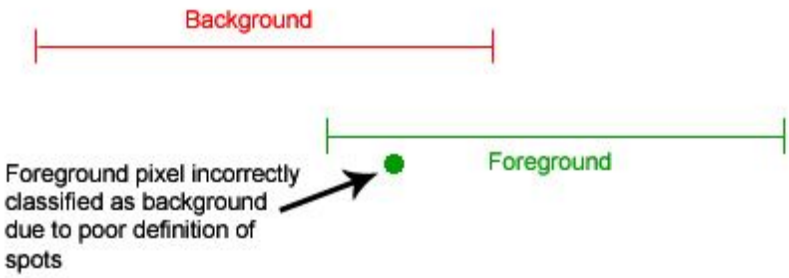

Image 5. Misclassification problem due to poor definitions of spots

Because of the explained in the previous paragraphs, accuracy can lead to unreliable values. However, as mentioned, the main source of errors is the human interaction, in such case any performance metric may fail.

Analysis

Data Exploration

For the casual user, an image is defined by its dimensions: width and height. These dimensions define the number of pixels that compose the image, so an image can be seen as a two-dimensional matrix of pixels, being the values for width and height the numbers of columns and rows respectively.

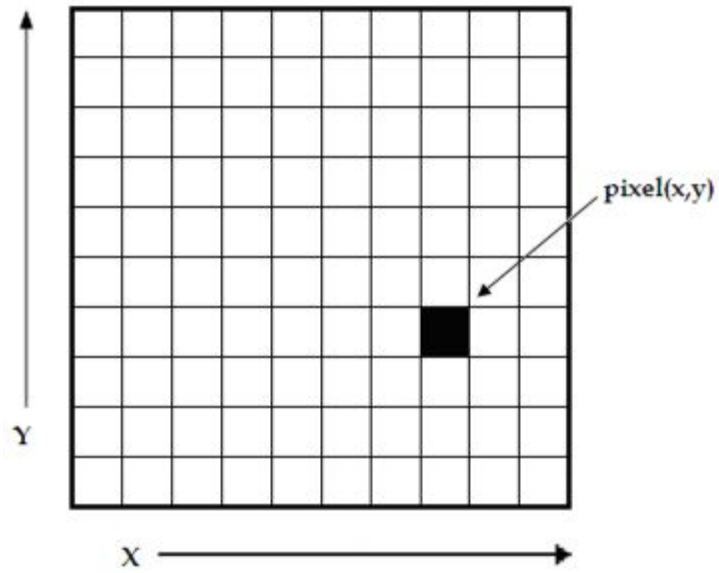

Image 6. Matrix of pixels (Image from https://www.researchgate.net)

Every pixel is a vector composed of three values which represent the primary colors used by computers to define the rest: Red, Green, Blue. In some cases, there is a four component that represents the opacity of the pixel, which is present in some image formats as PNG, however, that element is out of the scope of this project

Each component of a pixel is a numerical value which can be of several types. One of this types is the float format, in which case the values range from 0 to 1. However, I will use the integer format; with this data type, the range of possible values is from 0 to 255. The reason to choose integer over float data type is because of the amount of space needed to represent the former (8 bits) versus the latter (32 bits).

Based on the objective of the project, every pixel of the image has to be labeled as background or foreground. The information available to accomplish that is the RGB values of every pixel, so these values will act as the features of the dataset.

The input for the model will be the regions defined by the user which will be used to train the model and categorize the pixels.

Exploratory visualization

In order to classify the pixels of an image, their features have to analyzed. The features of the pixels of a certain class have to have certain similarity. Recalling that a pixel is made of three components (R, G and B), I will check those values to categorize the pixels.

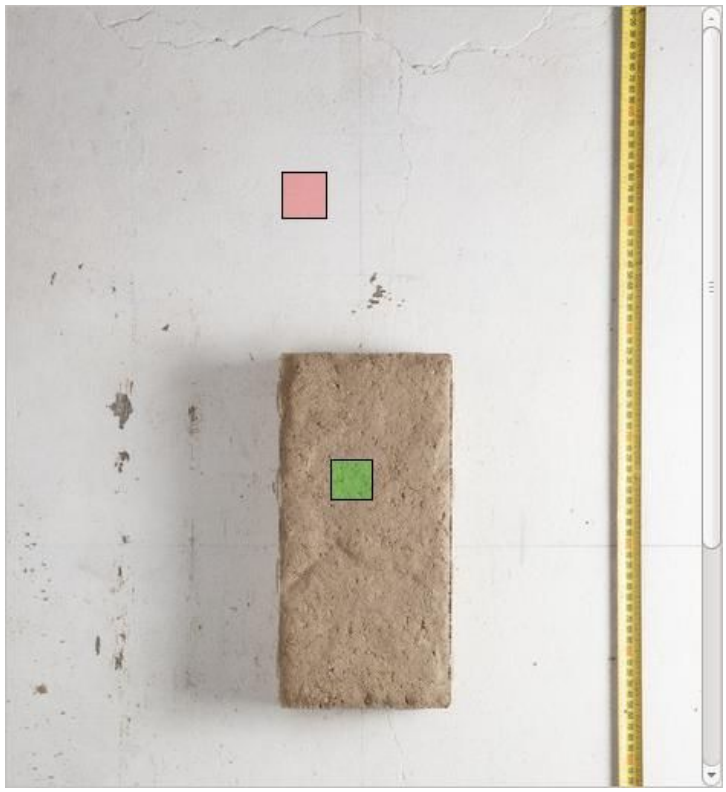

Image 7. Foreground and Background spots defined

Previous image shows an example of the spots that define the background (red) and foreground (green) regions that will be used in the model the categorize the rest of the pixels.

Analyzing the information provided by the pixels enclosed in those regions, I get the following:

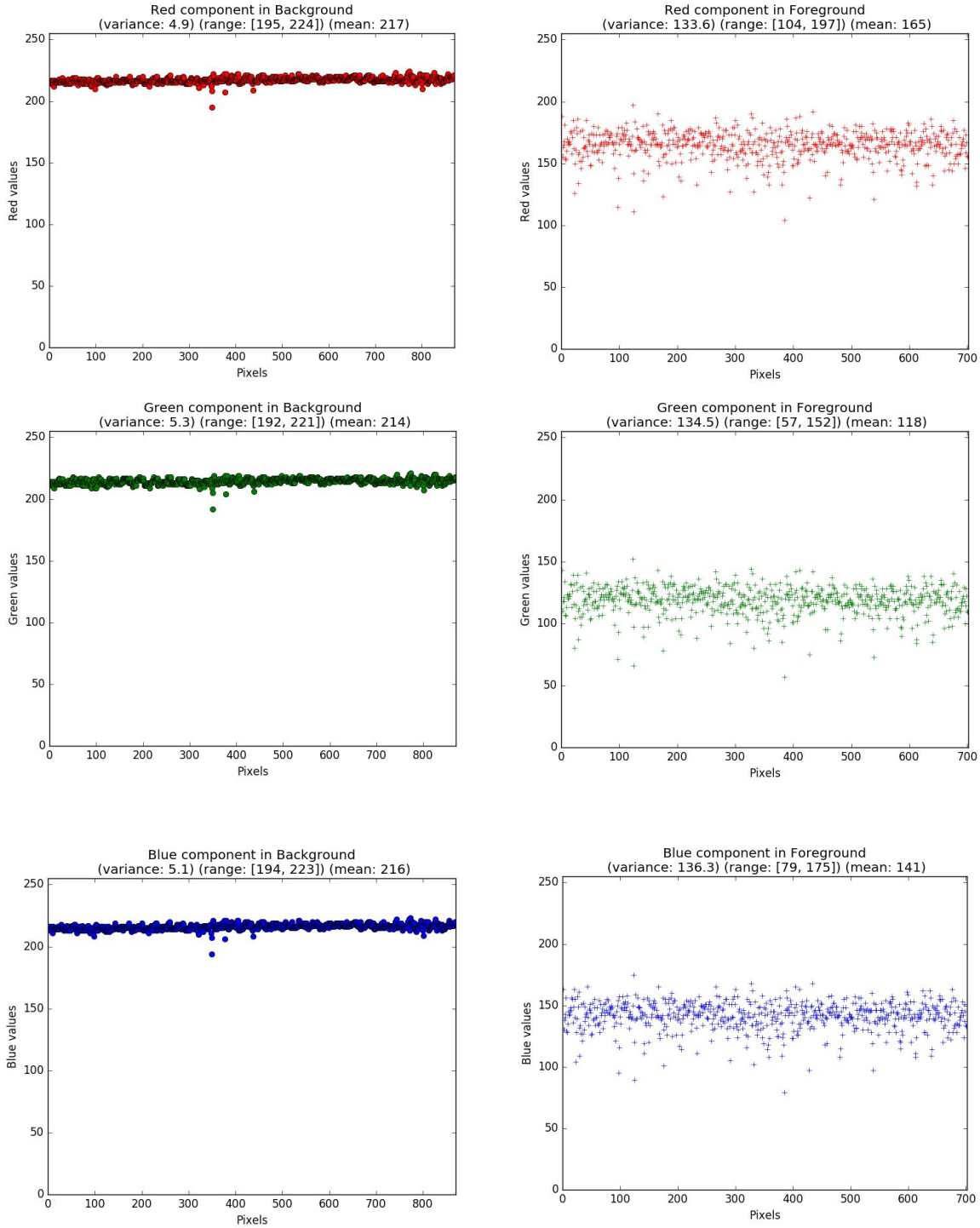

Image 8. Values of RGB components in foreground and background spots

From the previous image, there are several things to notice. First, the values of every component of the pixels vary within a defined range in background and foreground spots. In the case of the background, the ranges of the three components are almost the same: Red [195, 224], Green [192, 221], Blue [194, 223]. This makes senses since the background of the picture is a white floor, and pure white is represented by (255, 255, 255)

Looking again at the ranges, in the case of the background they are much more narrowed than the foreground ones. That is visually noticeable, and confirmed by the variance values.

However, a further analysis is required due the presence of the measuring type which pixels will need to be categorized as background too.

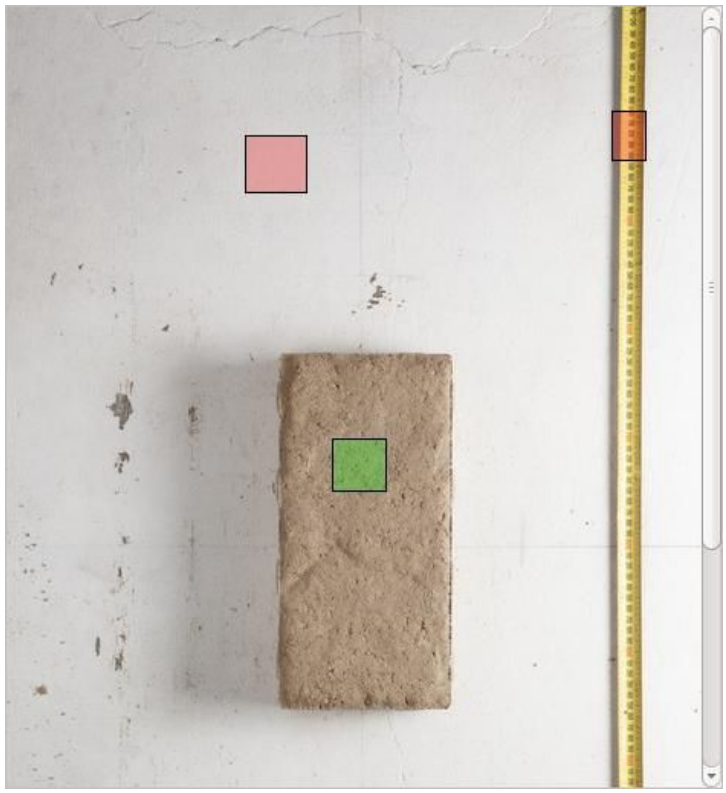

Image 9. Foreground and Background spots defined (including measuring tape)

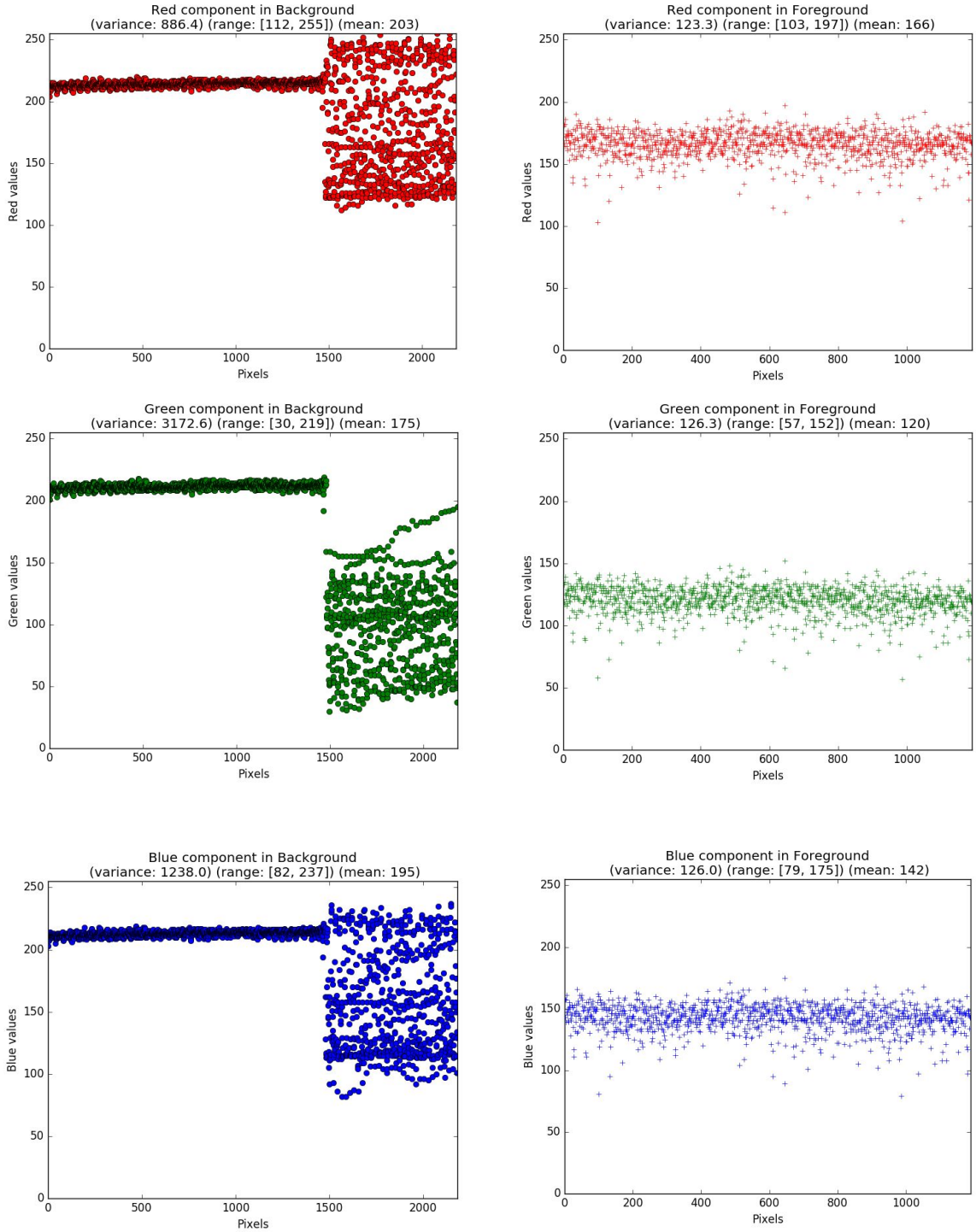

Image 10. Values of RGB components in foreground and background spots (including measuring tape)

In this case, we have similar results, except that wider ranges are defined for the background components. These expansions may cause problems because now the ranges for the background and foreground components are overlapped. However due the fact that there is a clear separation between the pixels that increase the ranges in the background (those coming from the measuring tape) and the pixels that define a narrowed range (those coming from the white floor), the classifier can still categorize the pixels properly.

Algorithms and Techniques

I'm going to use a Decision Tree classifier. This method is easy to understand and performs well with large datasets which is the current case. For instance, the images I will use have dimensions of 468 x 702 which means 328536 pixels to classify.

One downside of Decision Trees is that its accuracy can be misleading in unbalancing data sets, however since the input data for the algorithm is defined by the user, it is recommended that the background and foreground spots have similar sizes.

Benchmark

Since, the blocks that are going to be used have their corresponding edited pairs I created previously, I will use them to compare the performance of the solution.

Methodology

Data Preprocessing

There is no data preprocessing needed. The photos to be processed were taken specifically to identify the blocks, there is a white background with some imperfections (gray spots) that may be an issue when classifying the pixels, but those can be handled when processing the image.

Implementation

The project is implemented in python using the next third-party libraries:

- **PyQt4** to create the user interface
- **Numpy** to do array operations and conversions in the data
- **Sklearn** to classify the data
- **Skimage** and **Scipy** to process the images for refinement
- **Matplotlib** to visualize characteristics of the data

The first thing in the implementation was the design of the UI. This interface was implemented to aim simplicity and ease of use. The layout was created in the default designer provided by Qt. Next image shows the interface of the solution and descriptions of its components.

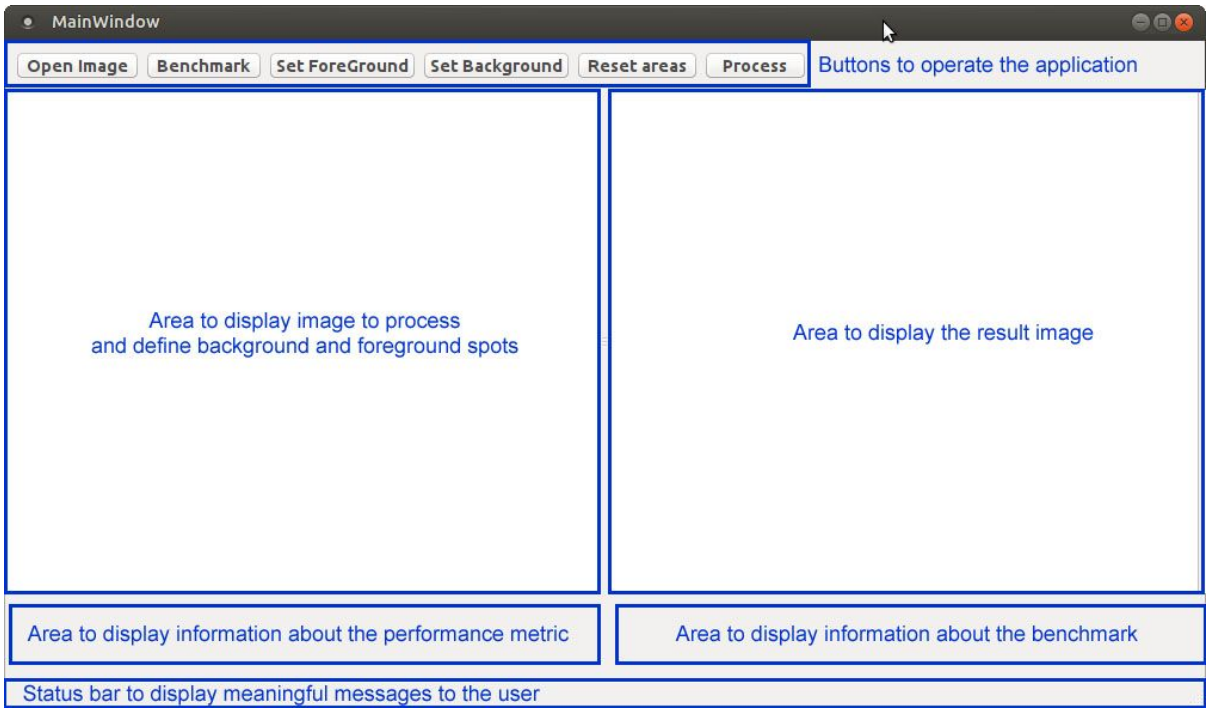

Image 11. User interface of the solution

The first thing to do is provide an image to process by clicking in “Open image” button. Optionally a benchmark can be provided by clicking in “Benchmark” button. Once this is done, the foreground and background spots that will define the input of the classifier model need to be set. This is done by clicking on the corresponding buttons to enable that action, then dragging and dropping the mouse pointer in the left canvas will define the areas, as shown in the following image.

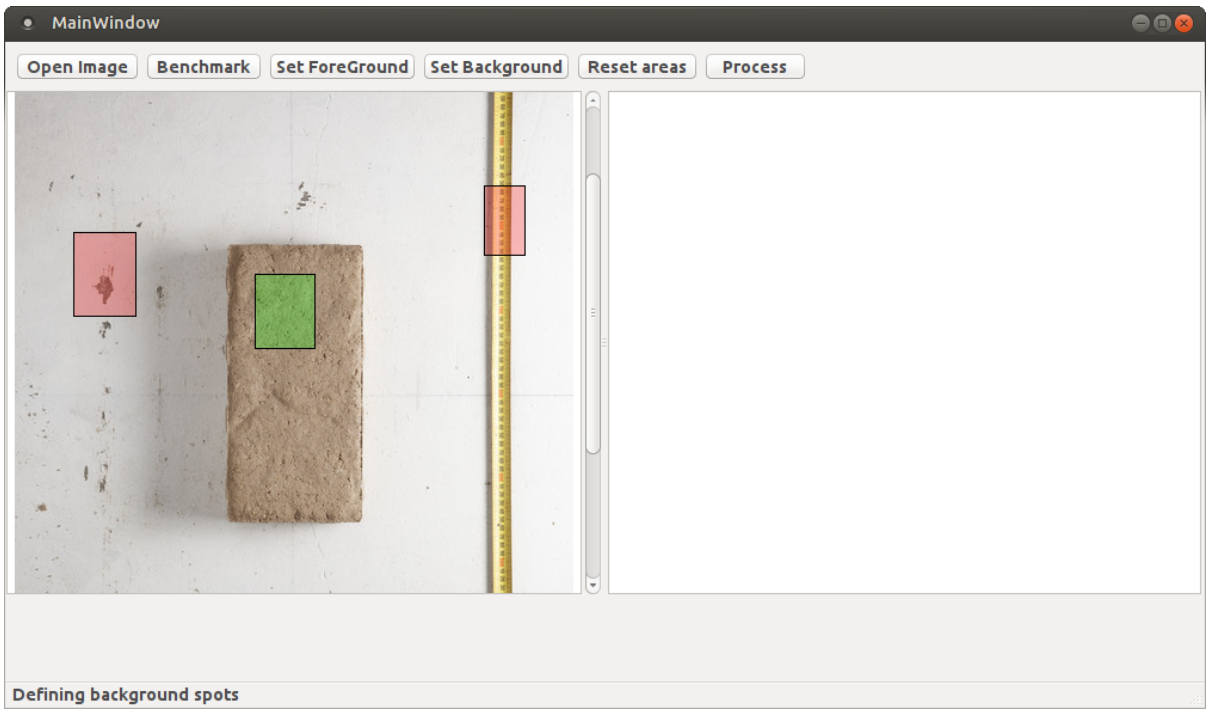

Image 12. Defining background and foreground spots to train the model

Now, the application is ready to process the image. When the “Process” button is clicked the next workflow occurs:

1. The data to train and test the model is defined by grabbing the pixels enclosed in the corresponding areas and assigning proper labels: -1 for background pixels, and 1 for foreground pixels.
2. The data is randomly split to define the training and testing set. This is especially important since in the previous step the background pixels were put at the beginning of the dataset and the foreground pixels at the end. Worth to mention that 30% of the pixels will be used for testing.
3. The next thing is to define the classifier. As mentioned, in this case I am using a Decision Tree classifier. In this step Grid Search is used to find the most suitable parameters for the classifier. Two parameters are used: max_depth and min_samples_leaf. The first one defines how depth in the tree the classifier will go to classify the data. The second one defines the minimum number of samples in a node to split it.
4. The classifier is trained using the training set.
5. The performance score is calculated for training and testing sets, its values are displayed to the user in the GUI.
6. All the pixels in the image are classified. In this stage pixels classified as background are set to black.
7. Refinement is done. This is explained in the next section
8. If a benchmark is provided, a comparison is done using the performance metric. The result is displayed to the user in the GUI
9. Finally, the result image is displayed in the right canvas. Images of every process are saved in the folder where the image was selected. Plots and a log are saved in the folder where the application is executed

The implementation has been done so that possible errors due to incorrect usage are handled accordingly.

Refinement

Some refinements are done in order to get a better result once the classifying is done. This is better explained with an example.

Image 13 shows the result after classification is done. As seen, the classifier is doing its work based on the spots defined by the user. However, it is noticeable how some pixels within the overall foreground region are defined as background (black pixels within the block).

To overcome this problem, I use the convex hull technique, which basically sets to the same category pixels enclosed by an convex area. This is possible to do in this case because the object of interest is a convex one, with other kind of objects, the results are not the expected ones. The result of this process is seen in Image 14

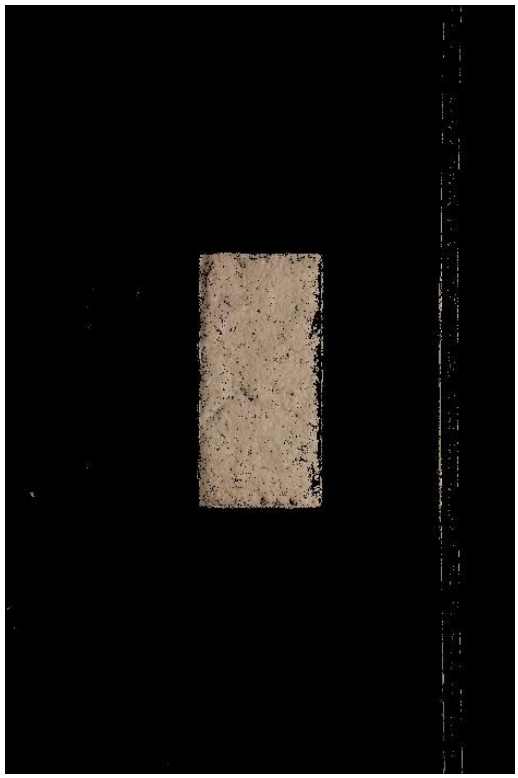

Image 13. Image after classification

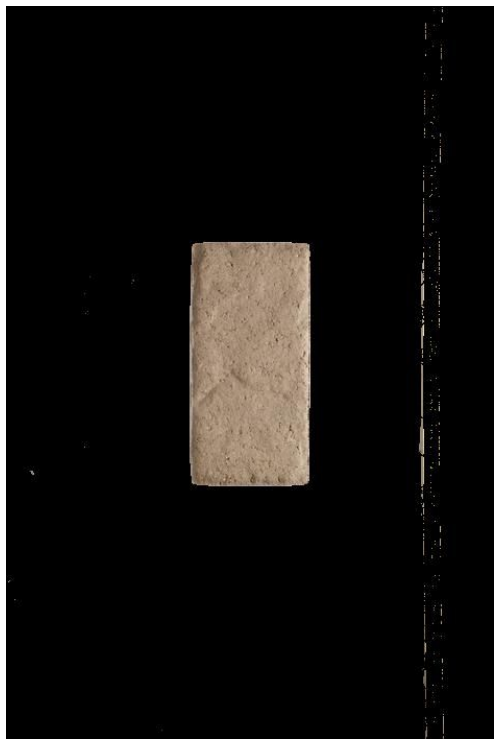

Image 14. Image after convex hull refinement

After the previous refinement, it is noticeable how some small spots are scattered in the background. To get rid of those small spots, the objects in the image are grabbed. An object

is defined as an agglomeration of continuous not black pixels. Once this is done,I removed the objects which are smaller than a given threshold. The result is shown in Image 15.

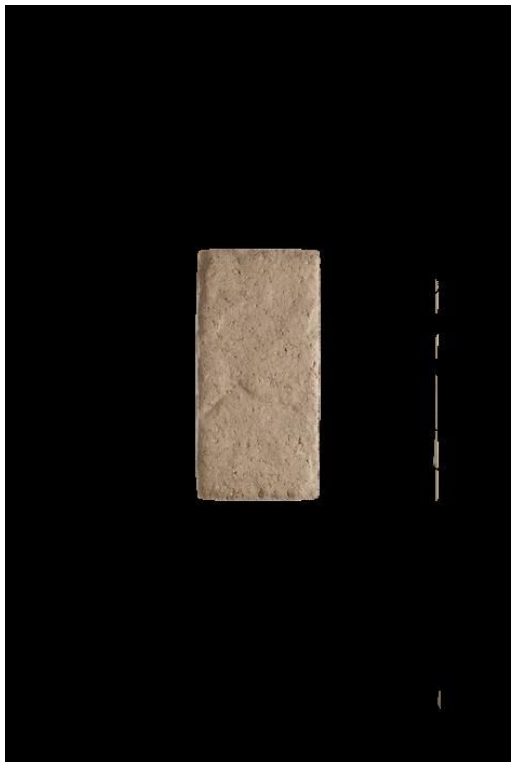

Image 15. Image after removing small spots

Results

Model Evaluation and Validation

As mentioned, the model used to classify the is a Decision Tree Classifier because of its simplicity and speediness in large datasets. I tried to use other classifiers for this project, but they don't behave well in this context.

SVM, took too much time to classify the pixels in the data, which is expected since one of its weakness is that it does not behaves well in large datasets. Also, the accuracy was not as good as Decision Tree's

KNN is a good alternative too, but similar to SVM, it takes more time to do the job, not as much as the previous one, but more than the selected model. This model did a good job classifying the pixels, but the speed was relevant to not choose it.

As mentioned as set two parameters for the model: max_depth and min_samples_leaf. I used the technique of Grid Search to let the application choose the most suitable values based on the given input, so based on that, those values vary accordingly.

Since this is a problem that was addressed using a supervising learning method, the spots for foreground defined by the user are critical to get good results. The following image shows some results for the same image based on the defined spots.

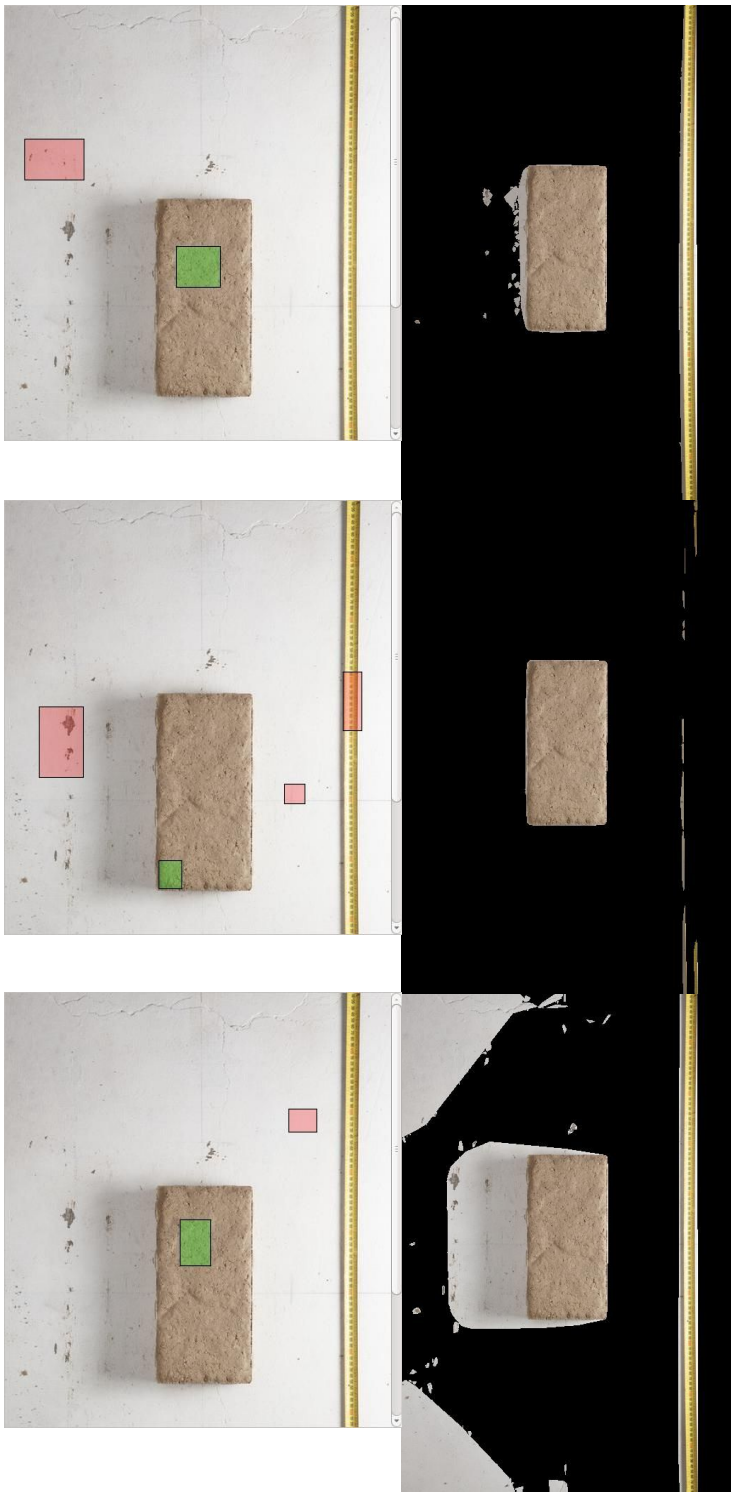

Image 16. Results from different background and foreground spots

Why is there such variability in the results if all previous spots seem to be valid? First, both categories have to be defined. If the problem is reduced to a simpler one, where there are

defined ranges of single values for each class, as shown in Image 17, it is easier to see that the classifier has to face the next situations:

- In overlapped sections, classify pixels that fall into the “shared” area.
- In not overlapped sections, classify pixels that do not fall in either.

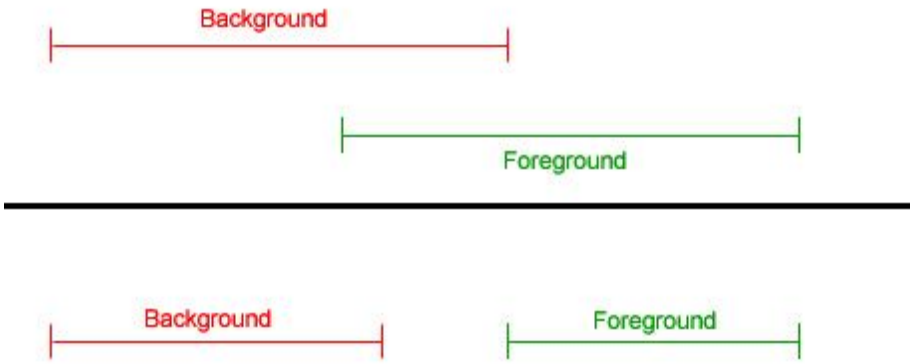

Image 17. Problematic situations to classify pixels

Those situations are difficult to handle for the classifier, in both cases, the information provided by the features is not enough to do a good classification.

One alternative to overcome the previous situations is to define the spot of just one category. That way pixels within the range of the category will be classified as such and the rest in the other type. However, this can lead to a similar problem when the spot defined includes pixels that it shouldn’t, therefore creating a range that is not good to classify the elements properly.

Another aspect to keep in mind is the refinement process, where the objective is to have a better definition of the background and the foreground. However, the nature of this process makes that some pixels correctly classified have to be moved to the other category.

Image 18. Refinement process issues

Previous image shows the result after the classification is done (left) and the result after refinement process (right). In that particular case the amount of pixels moved from the background to the foreground category is noticeable big.

Based on the previous paragraphs, the solution is good enough as long as the user defines the spots properly, which is an uncertainty. However, when human interaction is “acceptable”, some good results can be reached from unseen pictures, as seen in the next image.

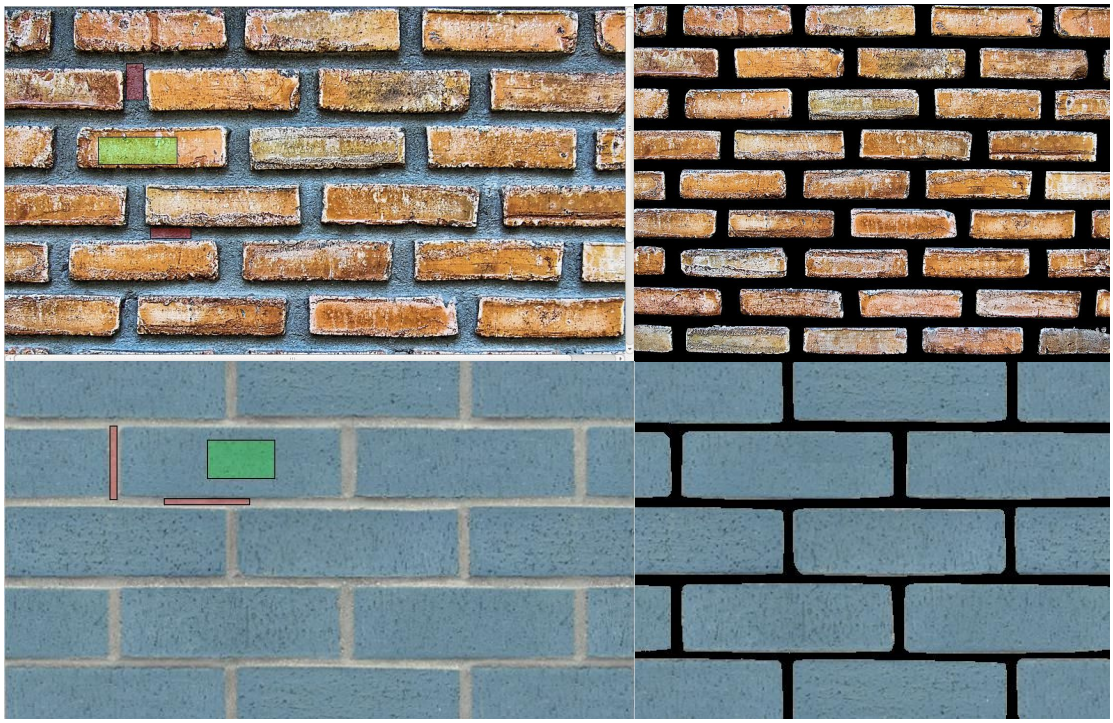

Image 19. Results obtained using external images
https://pixabay.com/static/uploads/photo/2015/08/01/12/26/wall-870221_960_720.jpg
https://image.freepik.com/free-photo/tileable-wall-of-brick-texture-with-15-colors_354-2147488430.jpg

Justification

Image 20 shows a processed picture compared against its benchmark. It is noticeable that the solution has identified and removed most of the background. However, some spots corresponding to the measuring have remained. In this case, the accuracy is 0.9092 which is a good value, and a good starting point.

One key to understand why the final solution for this particular example shows some unexpected spots is the process of refinement. The spots are removed based on a defined size. I set this value to 25, so spots smaller than that value are removed.

Checking the picture before that step (Image 21), the result of the convex hull process, it is clear that the small spots are removed. So the ideal solution should calculate the mentioned value programmatically to get rid of unwanted spots .

However this value is not easy to calculate because of the randomness in the spots size. Some alternatives I've tried, like calculating the mean size, don't remove enough spots. Others, like the percentile (> 95), removed all the objects which results in a completely black image. So, ultimately, is the refinement process which needs a further work.

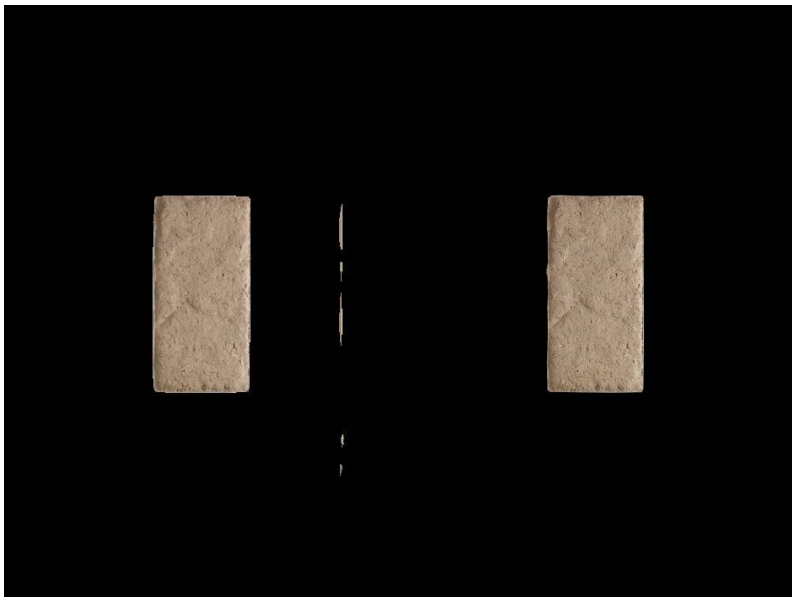

Image 20. Processed image compared to its benchmark

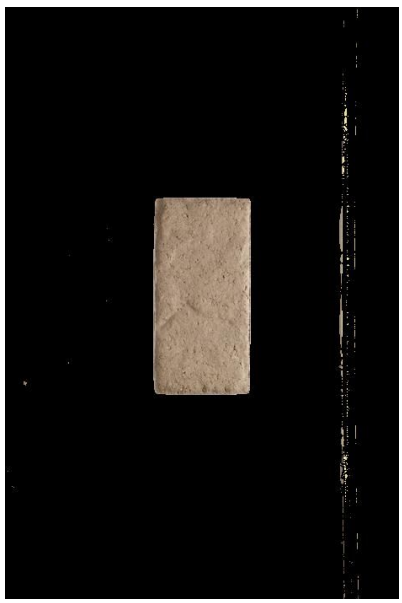

Image 21. Processed image after convex hull

Conclusion

Free-Form Visualization

Despite the fact that in the example discussed previously, the solution present some small issues compared to its benchmark, there are some good results obtained from other sources.

As shown in Image 19, the solution gets good results when the background is somehow homogeneous, that is when there no extra objects that should be classified as background.

Reflection

Recalling the objective of the project, **Classify the pixels of an image as background or foreground**, the solution presented here relies on two key factors:

1. Definition of the background and foreground areas for the training of the model
2. Refinement to get rid of unwanted spots

Factor number 1 depends on the human interaction. As seen in image 16, different definitions of the background and foreground areas give different results. After all, the classifier works based on what the user gives it.

Given that the user can identify which areas need to be removed and which don't, it is possible to trust that these regions will be set according to the objective. Nonetheless, human interactions is always a source of uncertainty.

Factor number 2 is something where more control can be gotten by using other alternatives to refine the result, as mentioned in the Justification section.

Improvement

There are several ways to improve the solution. First, a better classifier can be chosen. However, here there is a factor that can play an important role: speedness. There is not enough that the model does a good job if the entire process will take too much time.

As mentioned in Project Overview, my motivation for this project was the amount of time consumed when cleaning the blocks up. In my opinion, it is ok that the solution delivers results with some imperfections. As a person that have worked in the image edition field, I can say that those imperfections can be handled easily.

One question that may arise is "Why not to use a unsupervised learning method?" This is something that I thought for the project. However, I found that, even though there are good

algorithms that can classify objects in an image, the key point in this project is being able to define the background and foreground to retrieve the object of interest. As seen in Image 7, the background may be composed of several objects, that's where the human interaction is needed, to define the corresponding spots. When this is done properly, the results are good enough.

Note: I developed and executed the solution in a linux distribution (Ubuntu-Mate). The resources used in this project are provided with the code