Image Selection Based on Optimal Characteristic Analysis

DT228
BSc in Computer Science

Jameel Briones
Basel Magableh

School of Computing
Dublin Institute of Technology

April 06, 2016

Abstract

The goal of this project is to develop a mobile application that utilizes image processing techniques to automatically rank photos out of multiple similar images. This project can be separated into two main areas.

The first area is to select and categorize images based on their similarity using image feature extractions and comparison. The user is able to select images from their local folder to be uploaded to the server. These images will be checked for similarity and is grouped.

The second area is to determine the best images out of multiple similar images based on the analysis results of the images' properties. This will involve implementing various image processing techniques to analyse the image's quality. Basic image properties such as sharpness, noise level, exposure and contrast will be analysed to test for the image's quality. The photos will then be ranked based on the analysis results. The user will then be able to decide to share these images to a social media site.

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

Jameel Briones

____

C12456888

April 06, 2016

Acknowledgements

I would like to thank Dr. Basel for his guidance, enthusiasm and motivation throughout the extent of the project. Finally, I would like to thank my close friends and family for their support.

Table of Contents

Chapter 1: Introduction

Background

Photography has always been one of the primary medium for recording events since its invention. “There is hardly a person in the developed world who does not have photographs from his or her past” [1]. According to Tom de Castella [2], there is an increasing importance of photography in recent years due to digitalization during the digital era. The role of photography in society as well as society’s perception towards photography has changed drastically. Before the digital era, photography was treated as a luxury commodity and not everyone was able to afford it. Cameras and films were expensive. The usage of films resulted in limited photos taken very carefully. Poorly taken pictures would result to a wasted film which, for a photographer, would mean wasted money. The introduction of digital cameras and integrated digital cameras in mobile devices made photography available to almost everyone in society today. Photography in today’s society is “not only a hobby but can also be embedded in other activities” [1]. Online photo sharing has also become a regular activity with the increase on popularity of social media sites such as Facebook, Instagram and Twitter.

Photography has been made instant and convenient due to digitalization. Usage of films have become unnecessary, eliminating the limitation of photos taken at a time. People do not hesitate on taking multiple similar images with the same scene, just taken in different angles, as there is no longer a cost on films for each picture taken. During the analogue era, photographers would look first and then shoot. Today, with digital photography, the photographer would shoot first and then look at the resulting photo taken.

According to John Ingledew [3], today a photographer seems to be able to take unlimited numbers of pictures without being burdened by the costs of film. Large amounts of photos are taken and stored daily, causing difficulties on selecting which images would be best to use. Not every photo taken will be of good quality. Photo organization becomes an important task in order to discard redundant and poor quality images when a photo collection becomes larger. However, the selection of the best images out of a large amount of photos can take some time.

The purpose of this project is to make this selection process to be automated in order to make photo organization become a lot easier on mobile devices.

Objectives

The goal of this project is to develop a mobile application that utilizes image processing techniques to automatically rank photos out of multiple similar images. This project can be separated into three main objectives.

The first objective is to categorize the uploaded images based on their similarity using ORB, an image feature extraction and comparison algorithm.

The second objective is to create a set of techniques to rank the grouped images, returned by the image comparison feature, based on the analysis results of the image’s quality. The quality assessment of the image will include the analysis of multiple image properties such as exposure, brightness, contrast, saturation, sharpness, noise level and subject identification.

The third and final objective is to provide a fully functional mobile application that will allow the users to select and upload multiple images from the album of their mobile device and receive a response from the server which contains the ranked and grouped images. The mobile application will then be able to display these images according to the results and allow the user to share images of their choice to social media sites.

Chapter 2: Background Research

This chapter focuses on the background research undertaken for the development of this project. It will cover various topics in relation to image comparison and analysis, starting with a brief introduction to digital image processing. A section in this chapter focuses on the technologies and algorithms researched which would help in the implementation of the application. Similar solutions are also outlined in this chapter in order to get an idea of various features and evaluate key functional requirements that the application will have.

What is a digital image?

Digital images are represented as a multi-dimensional array. For example, a coloured image can be represented as a three dimensional array (X, Y, Z), where X and Y are the co-ordinates within that image and Z containing the colour dimension RGB and a monochromatic image can be represented as a two-dimensional array (X,Y). Each element in this array is called a pixel, the smallest unit that you could find in a digital image. Visually, each pixel is a small square of colour. Each pixel contains a value which may be a red (R), green (G) and Blue (B) value for a coloured image or a grey level value for a greyscale image.

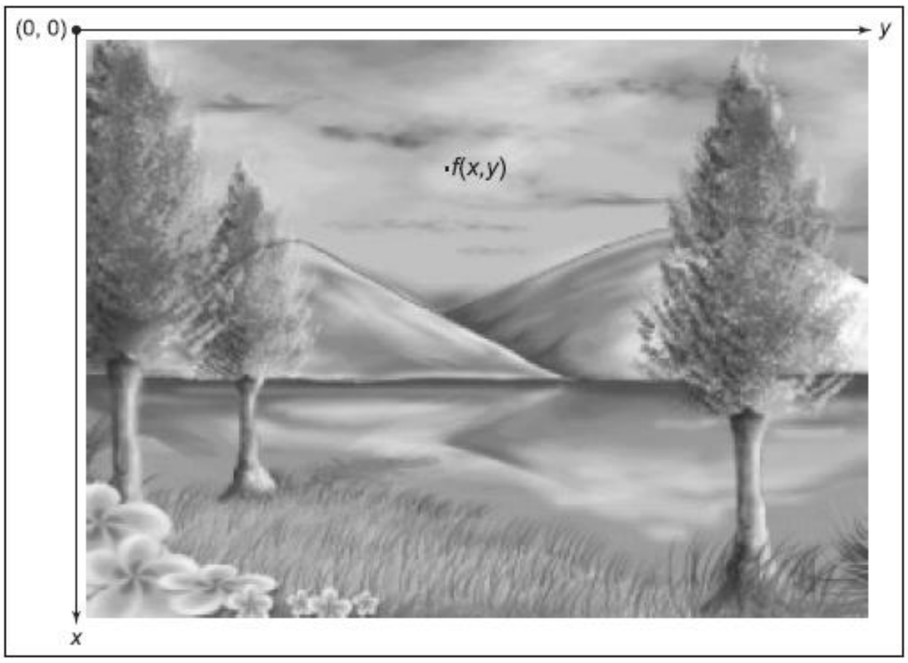


Figure 2.1: Shows how an image is represented [4]

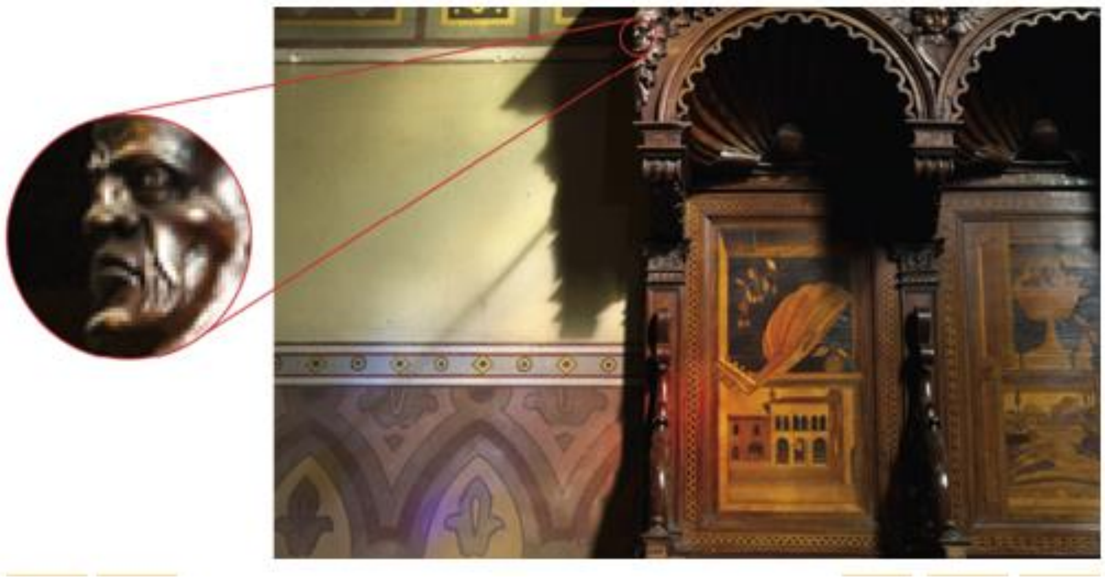


Figure 2.2: Pixels in an image [5]

Digital image processing

“During the last century, photography was based on chemical processes, but its future is now based in computation. A mobile device is a computer with lenses, capable of capturing images in digital form. A computer may have software which allows these images to be processed” [6].For example, these images could be enhanced and manipulated or useful information could be extracted from them.

“Digital image processing is the use of computer algorithms to perform image processing on digital images. In particular, digital image processing is the only practical technology for classification, feature extraction, pattern recognition, projection, and multi-scale signal analysis” [7]. Image processing techniques are used in many different applications such as image enhancing and computer vision. These techniques can also be applied to extract and describe features in an image which would allow us to match these features with another image for comparison. It will also allow us to analyse image properties such as contrast, exposure and sharpness.

Feature Detection

Key-points refers to spatial locations of points of interests which can be described and extracted from an image. According to the Weizmann Institute [8], these points of interests needs to be sensitive to changes where even a small difference can have a significant change in all directions, therefore out of all the different features in an image, a corner is best and mostly used in feature detection.

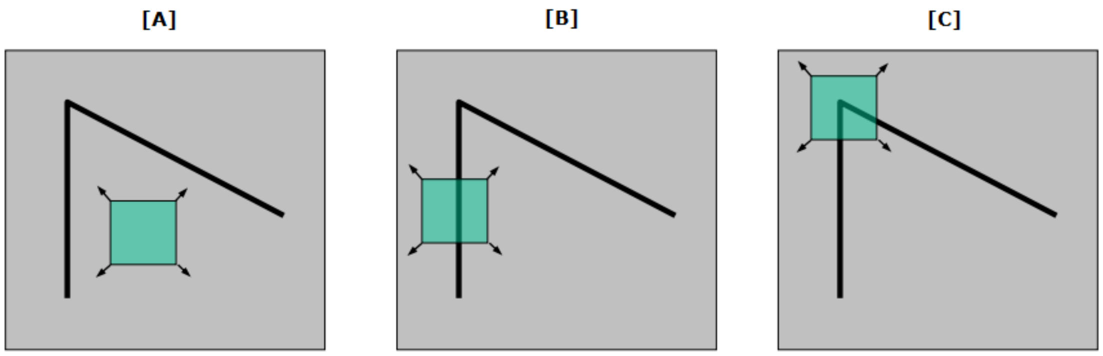


Figure 2.3 (A) Flat region: There is no change in all directions, (B) Edge: No change along the edge direction, (C) Corner: Significant change in all directions [8]

These key-points are rotation and scale invariant which can be detected whether an image is subjected to a certain degree of rotation, scale and distortion. According to Yingxu Wang [9], the extraction of key-point-based image features includes two steps:

1. **Key-point detection:** Key-point detectors are used to find these point of interests in an image.
2. **Key-point description:** Key-points descriptors are used to describe characteristics of these key-points detected by key-point detectors. Detecting key-points is simply not enough as this will only give us the position of each key-point. Key-point description is an important feature in order to allow comparison between images as descriptors differentiate each key-points from each other [10]. Characteristics of a key-point is described in a vector which can then be used to compare with a key-point in another image.

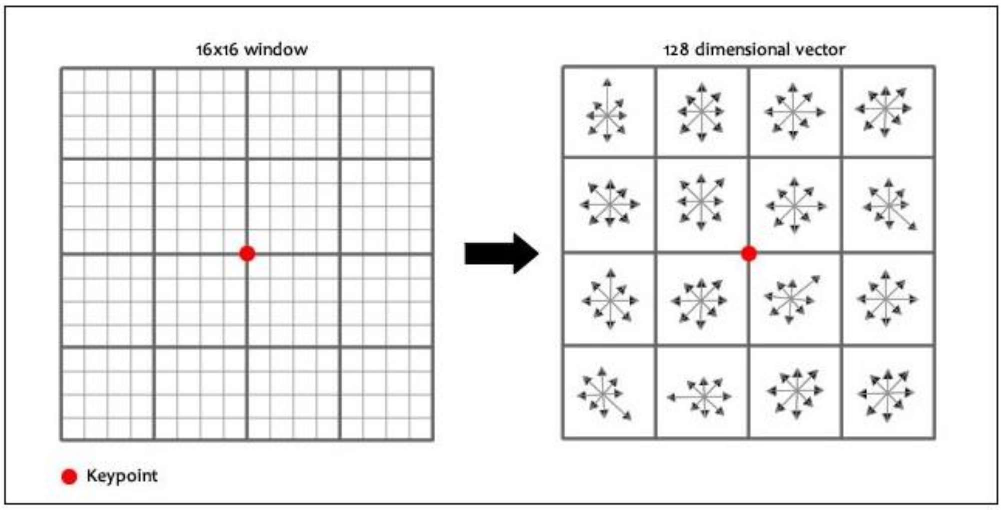


Figure 2.4 Example of a key-point and its description represented by a vector, calculating gradient magnitudes and orientations in each 4x4 sections. [10]

Image quality evaluation

According to Yiwen Luo and Xiaou Tang [11], the task of evaluating an image's quality is traditionally a human task and the idea of automating this assessment in consistent of human perception is a very challenging topic in computer vision. The human ability to determine the difference between a high-quality, professionally taken, image and a low-quality, amateur taken, image can be very difficult to implement in an application especially with the selection of the best photo due to its nature of being subjective. A photo picked as the best amongst a group of pictures by a certain person does not necessarily mean that it will be the same selection as another individual. However, there are several differences between high and low quality images due to the photography techniques applied to take the picture. The popularity of camera phones has allowed for a rapid development in the camera functionality integrated with the mobile phone. Though not as fully functional as a digital single-lens reflex (DSLR) camera, there is now more options to control the camera manually such as ISO, metering, exposure control and scene selection.

These photography techniques applied to take the picture can be seen on the image properties in effect. Basic properties such as contrast, exposure, white balance and saturation can be affected depending on how the photographer takes the image. The right application of these techniques can determine the factors that can make a photo look good in human perception. The evaluation of an image will be based according to the following basic properties that are used as a standard to assess image quality in photography.

Brightness & Exposure

"Lighting is the most important technical aspect of any photograph" [12]. Lighting can affect an image's variance on colour and tones and is an important factor that can make a huge difference between a high quality and low quality image. The difference of having a good lighting can be clearly seen in the following images below.

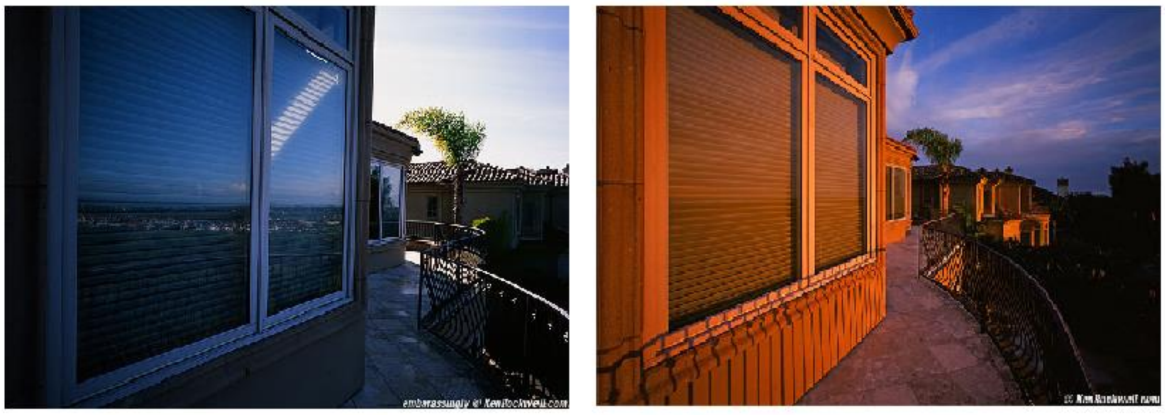


Figure 2.5 Left: Photo with bad lighting. Right: Similar photo with different lighting [12]

Photography techniques that controls the light coming into an image such as aperture and ISO also plays a huge part on making a high quality image. Two of the properties hugely affected by light is the brightness and exposure of an image. Brightness and exposure may appear to have similar effects and are closely related but they are very different. The brightness of an image is and affects the overall tone of the image but exposure affects the highlight tones of an image. For example, an image that is considered to be bright or dark can have a correct exposure.

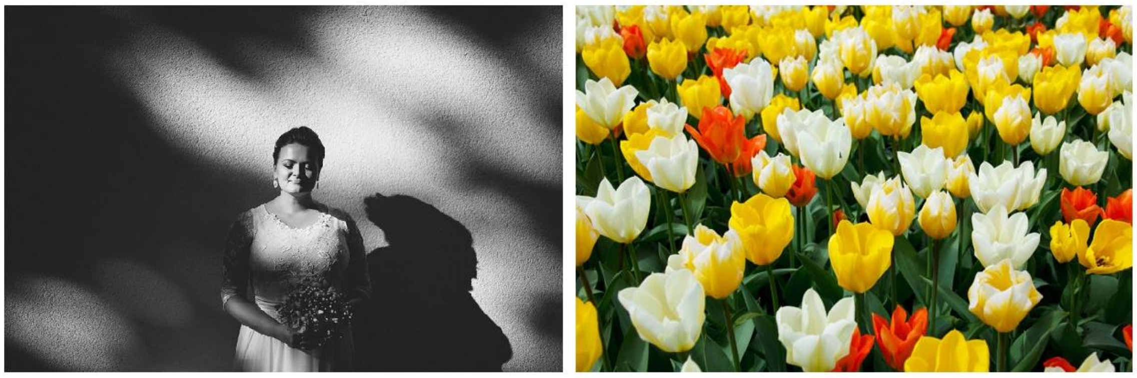


Figure 2.6 Left: Image considered to be a dark image with correct exposure. Right: Image considered to be a bright image with correct exposure

The brightness and exposure of an image can be visualised using a luminance histogram. Photographers using a DSLR camera can use an in-built light meter, which its result is displayed by an exposure meter, while taking a photo as a guide to see if the camera’s setting will lead to an evenly exposed image. Histograms are also useful for this evaluation and can be reviewed by photographers after taking the photo. Histograms can be used as a guide in order to see if an image has the correct exposure or if it is over or underexposed. However, these feature does not normally exist as part of the camera application on mobile phones, though there are a few mobile camera applications that offer this feature, thus this evaluation is often left to human perception.

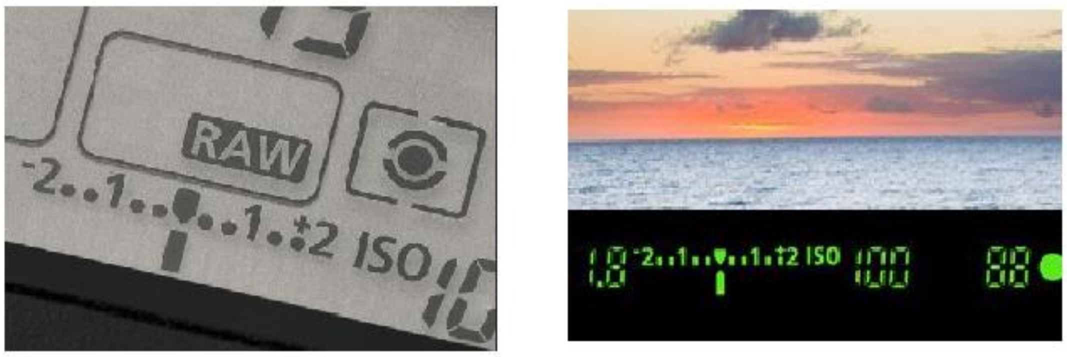


Figure 2.7 Left: Exposure meter display on LCD, Right: Exposure meter display in the viewfinder [13]

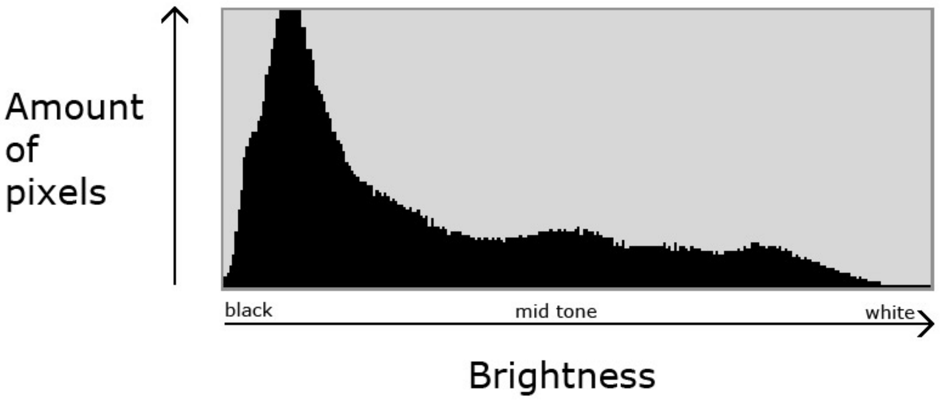


Figure 2.8: An example of a luminance histogram, black as shadows and white as highlights. [14]

According to Gibson [15], the luminance histogram is a guide which shows the tonal distribution of an image. For example, an image which contains a lot of dark tones would have a histogram that would lean heavily to the left and an image which contains a lot of bright tones will have a histogram that will lean heavily to the right.

A correctly exposed image is considered to have a tonal distribution that is spread across the histogram. An under exposed image is considered to have a tonal distribution that does not go all the way across to the right of the histogram, lack of bright highlights, and an over exposed image will have clipped highlights, graph climbing the wall on the right hand side of the histogram. Under-exposed images can cause loss of shadow details whereas over-exposed images can cause a loss of highlight details.

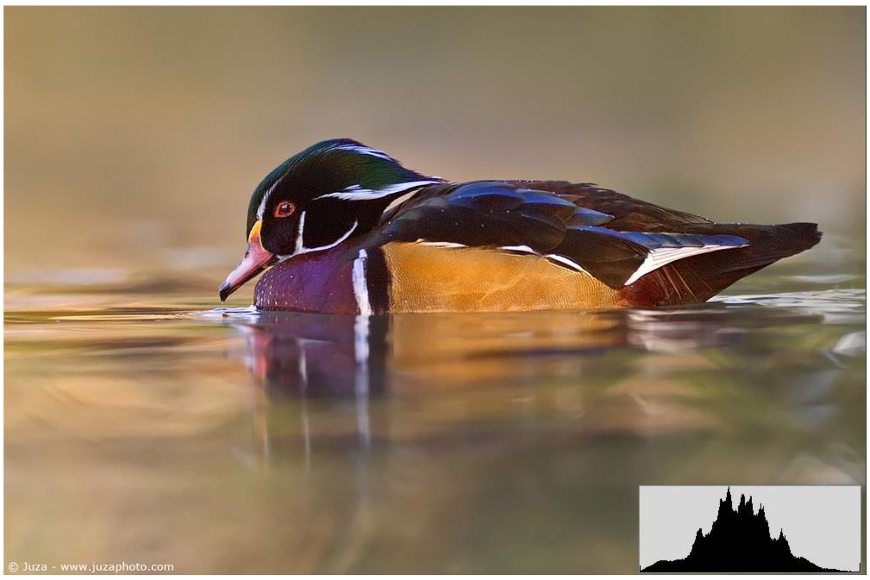


Figure 2.9 The image above has average brightness and is well exposed. The tonal distribution is spread across the histogram, going from black (0) to white (1). [14]

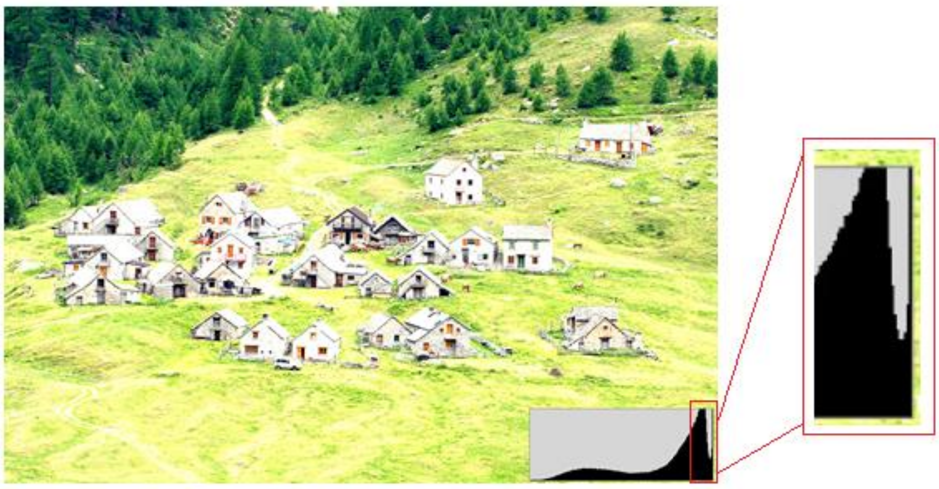


Figure 2.10: The image above shows an overexposed photo. Notice the clipping on the right side of the histogram, showing the loss of highlight details in the picture. [14]

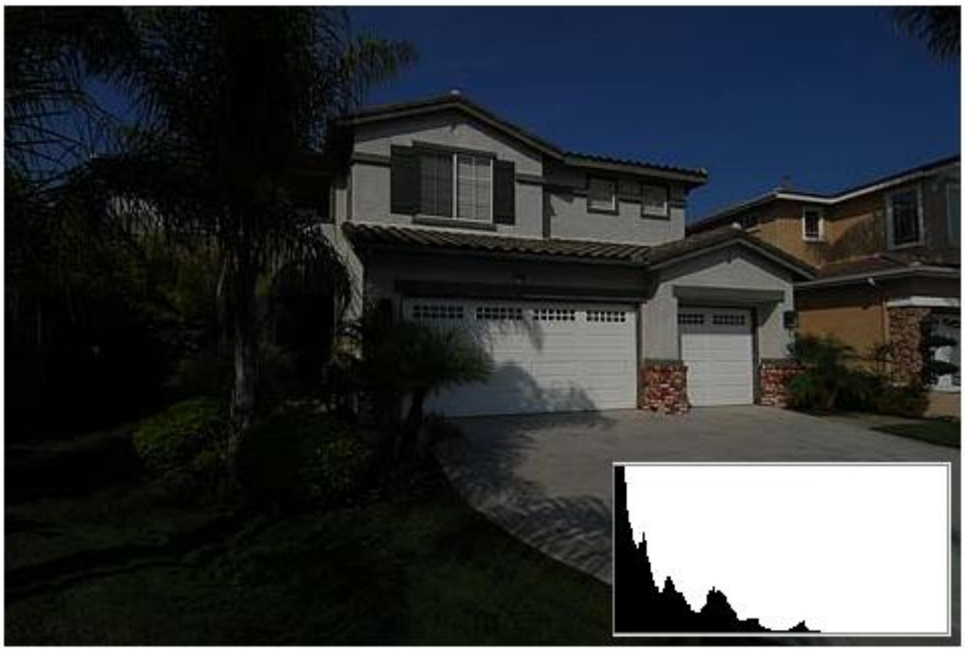
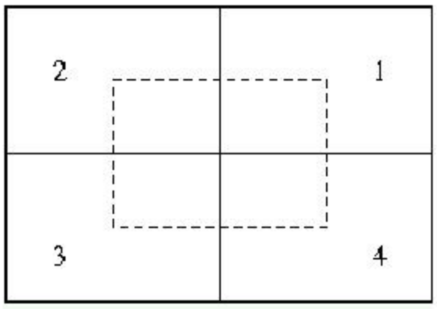


Figure 2.11: The image above shows an underexposed photo. Notice that the histogram stops abruptly near the middle without reaching the end. The highlights of the image are shown to have mid-tone values. Loss of details are also shown in the darkest part of the image. [16]

The overall brightness of an image can also be calculated using the luminance values visualized by the luminance histogram. The mean of the luminance values can be calculated for the overall brightness value of the image. According to Henry Chu, this may not be a precise measure as an image may not have an even luminance distribution. Thus, a geometric mean may be a more precise measure for this calculation. Five different local regions can be looked at and an average of luminance of these different regions may be calculated.



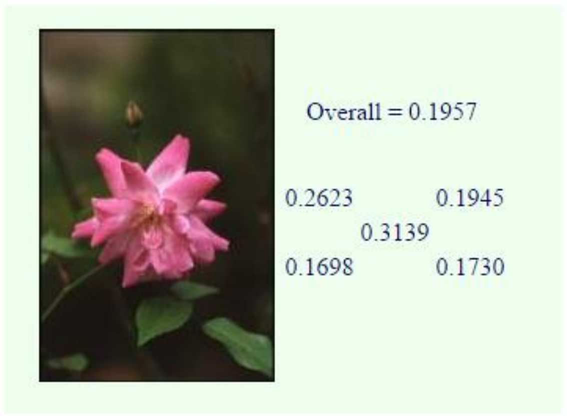


Figure 2.12: The above image shows an example of how an image can be separated into five different local regions of interest to calculate a geometric average of an image’s brightness. [17]

Contrast

The contrast of an image can be defined as “the transition of tones from dark to light or from the shadows to the highlights” [18]. An image with a low contrast has a limited range of tonal distribution, the histogram graph is not spread out over the entire range. On the other hand, an image with a good or high contrast will have a tonal distribution that is spread out across the entire range of the histogram. This is often applied to contrast enhancement techniques such as contrast stretching. This technique, often called normalization, ‘stretches’ the range of its luminance values in order to improve the contrast of an image.

According to Tom P. Ashe [18], Images with low contrast can appear to be lacking of depth thus it can look flat or very dull. However, images which have a contrast that is too high can have a negative effect on the image’s quality. This can lead to a lock of detail in both the highlight and the shadows of the image. Therefore, an image with a balanced tonal distribution are considered to be optimal.

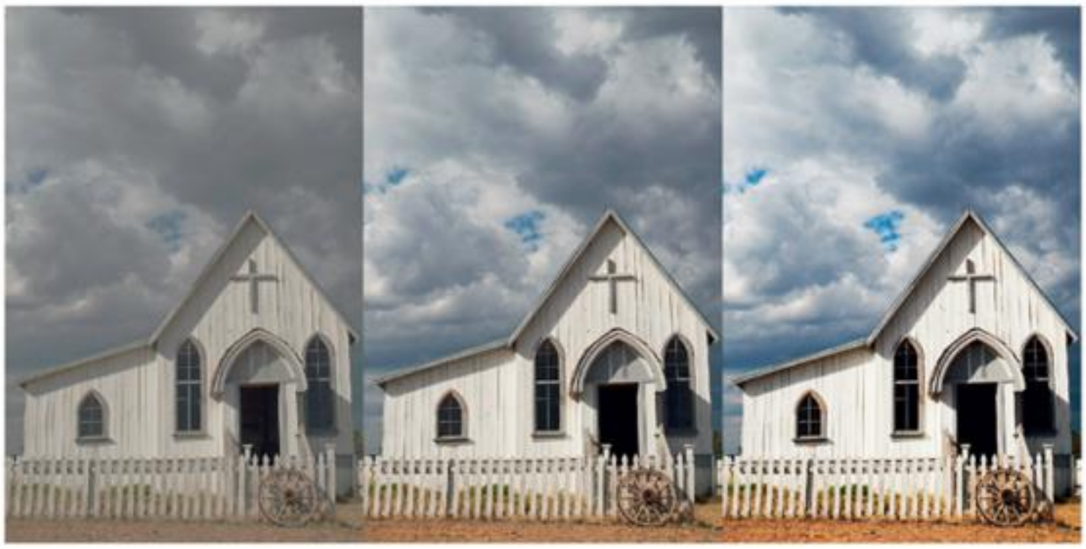


Figure 2.13: Differences between contrasts using the same image. Left: low contrast version of the original image. Middle: Original version. Right: high contrast version of the original image.

There are many contrast measures available and discussed in different studies and literatures such as the Weber, Michelson and root mean squared (RMS) contrast. According to Elizabeth Allen and Sophie Triantaphillidou [19], the Weber contrast measure is used for measuring local contrast with a uniform luminance against a uniform background. This will not be a good contrast measure for digital images as not all images will have such characteristics. The Michelson contrast measure on the other hand, measures the contrast of a spatially periodic pattern like single frequency sinusoidal grating. This measure, however, cannot be used for digital images with complex scene content, such as those with complexity in terms of spatial frequency content. However, a global contrast for a digital image can be measured using the RMS contrast. This measures the standard deviation of the luminance level pixel values without depending on the spatial frequency content of the image or the spatial distribution of the contrast in the image. Therefore, I believe this would be the best contrast measure suited for my application.

Focus

"Imaging devices, particularly those with lenses of long focal lengths, usually suffer from limited depth-of-field. Therefore, in the acquired images, some parts of the object are well-focused while the other parts are defocused with a degree of blur" [20]. On the other hand, the photo can simply be just blurry by the way it was taken which for example may be due to the photographer's movement as the picture was taken. Blur or distortion in images is an important factor for measuring image quality as this leads to the reduction of sharpness and focus which causes loss of details in the picture.

There are many different methods for measuring the focus or sharpness of an image proposed in studies and literature. I have researched on the following focus measures to find out which is the best measure suited for this application:

- **Local variance:** "A well-focused image is expected to have a high variation in grey levels" [21]. This measure calculates the global variance of the luminance level of an image based on its local variances. This calculation is shown in the following figure where $\overline{lv}$ is the mean luminance of the image.

$$VAR(I) = \frac{1}{NM}\sum_{m}^{M}\sum_{n}^{N}\left[lv(m,n) - \overline{lv}\right]^2$$

Figure 2.14: Local variance formula for calculating focus measure.

- **Tenengrand measure:** The Tenengrand measure is based on the gradient magnitude of an image. "A well-focused image is expected to have sharper edges" [21]. Because of this, gradient magnitude based methods uses the image's gradients to determine a reliable focus measure. With a given image gradient, the focus measure pools the data at each point as a unique value. This measure uses the Sobel operator with the convolution mask. The following figure shows how this focus measure is calculated where $S(m,n)$ represents the gradient magnitude.

$$TEN(I) = \sum_{m}^{M}\sum_{n}^{N}\left[S(m,n)\right]^2 \quad for\ S(m,n) > T$$

Figure 2.15: Tenengrand focus measure

- **Sobel and variance measure:** This is also a measure based on the gradient magnitude of an image. Instead of pooling gradient information, the local variance of gradient magnitude is calculated instead. Similar with the Tenengrand measure, it also uses the Sobel operator with the convolution mask. This can be calculated using the following formula shown in the figure below where $S(m,n)$ represents the gradient magnitude and $\bar{S}$ is the mean of the gradient magnitudes.

$$SOB\_VAR(I) = \sum_{m}^{M}\sum_{n}^{N}\left[S(m,n) - \overline{S}\right]^2 \quad for\ S(m,n) > T$$

Figure 2.16: Sobel and variance focus measure

- **Laplacian measure:** This measure is based on the use of a second derivative operator which it to pass the high spatial frequencies associated to sharp edges. This uses a Laplacian operator with the convolution mask. This measure can be calculated by adding up all of the absolute values of the calculated gradient measure, $L(m, n)$

$$LAP(I) = \sum_{m}^{M} \sum_{n}^{N} |L(m, n)|$$

Figure 2.17: Laplacian focus measure

J. L. Pech-Pacheco, G. Cristobal, J. Chamorro-Martinez and J. Fernandez-Valdivia's compared and assessed each of the different measures mentioned above based on sharpness, smoothness and noise sensitivity. According to their assessment [21], Tenengrand and Sobel and variance method has a much sharper focus response and a better noise tolerance than the other mentioned measures. These are important factors for to consider as this can affect the precision of the focus measure. The following figure shows the result of their tests of each measure and from this, I believe that the Sobel and variance measure would be the optimal focus measure due to its better tolerance on noise.

Sobel+var	Tenengrad	Laplacian	Laplacian+var
1.0411	1.0646	2.507	2.101

Figure 2.18: Table showing the results of Pech-Pacheco et al.'s assessment of the different focus measures.

Noise

Image noise a visual distortion that can be defined as "the unwanted fluctuation of light intensity over the area of the image" [19]. This can be seen in an image as "random speckles on an otherwise smooth surface" [22]. This can have a huge effect on the quality of an image as it can make the photo appear very grainy.

According to Sean McHugh [22], a degree of noise is always present and is unavoidable in any electronic device that transmits or receives a signal, light in the case of a digital camera. However, this noise can seem to be non-existent in an image if the image has a relatively small signal to noise ratio (SNR). An image with a high SNR has little to non-visible noise in the image whereas an image with a low SNR will have a high noise visibility resulting to a grainy image. The following figures illustrates this concept. Figure 2.19 shows an image with high SNR which clearly illustrates how the image information can be clearly separated from the noise which exists in the background. Figure 2.20 on the other hand shows an image that has a low SNR which clearly illustrates how it is harder to distinguish between the image information and the background noise.

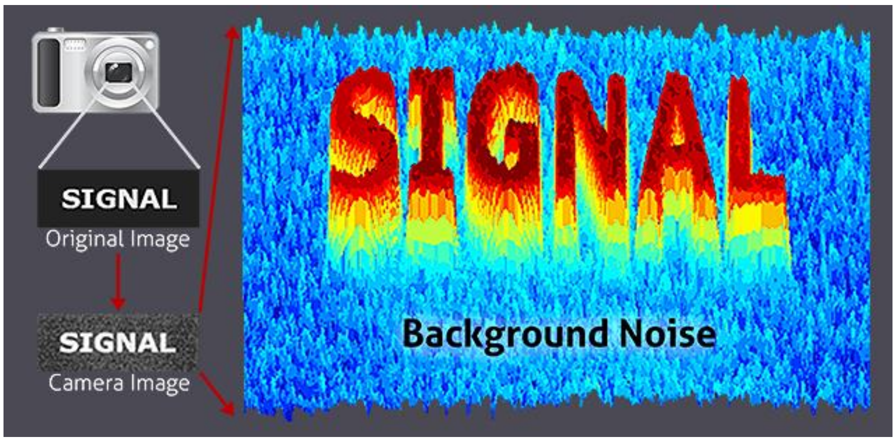


Figure 2.19: Image with a high signal to noise ratio.

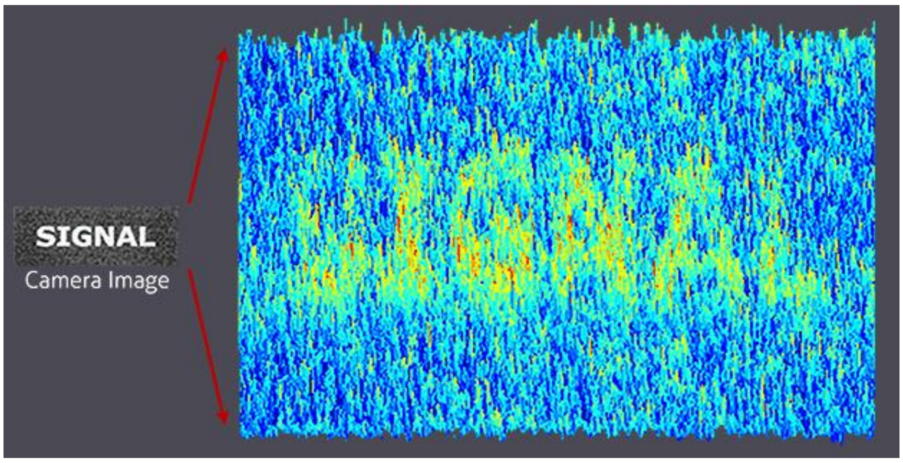


Figure 2.20: Image with a low signal to noise ratio.

Noise quantity can vary in digital images and there are several factors which can cause a significant decrease in the signal to noise ratio. Camera settings is one of the factors that can cause a decrease in SNR, mainly due to the ISO setting. “ISO is the level of sensitivity of the camera to available light” [23]. Increasing the ISO, amplifies the light the camera can capture, producing a stronger signal that the camera receives. Image information signal is not the only thing that gets amplified but the background noise gets amplified as well resulting in an increase of noise visible in an image. The camera sensor itself is also a factor as the greater the area of a pixel in the camera’s sensor enables a camera to have a greater light gathering ability thus also producing a stronger signal received by the camera.

Noise in an image is difficult to measure and is a challenging topic in computer vision. In order to measure noise, an estimate of a noise free image is often required. Even the signal to noise ratio can only be calculated if the standard deviation of the noise is known, which can only be “formally computed when the noise model and parameters are known”, otherwise it can be obtained as an empirical measurement [24].

According to Antoni Buades et al. [24], most de-noising methods depends on a filtering parameter h, which for most methods depends on the estimation of the noise variance of the image. “There are multiple different types of noises and often this noise is often assumed to be an additive Gaussian white noise (AWGN) but in reality this noise is unknown and non-additive” [25]. Therefore it is a difficult and challenging task to calculate an estimation of noise variance without knowing the noise characteristics present in the image.

I have researched the following image de-noising methods in order to measure the noise in an image.

- **Gaussian smoothing:** This de-noising method, convolves an image using a Gaussian kernel resulting in a blurred image with noise removed. However, this method depends on “the fact that the neighbourhood involved in the smoothing is large enough, so that the noise gets reduced by averaging” [24]. This method is optimal for the flat surfaces of an image but executes poorly no parts of the image which has edges or texture, often losing them as a result.
- **Bilateral filtering:** This method is an alternative way of adapting Gaussian filtering to preserve edges by taking both the space and range distances into account. “It averages pixels based on their spatial closeness and radiometric similarity” [26]. I believe that this is a much better method than the Gaussian method, therefore I will be using this as my de-noising method for this application.

Technologies Researched

There are multiple technologies and libraries available out there that is used in image processing applications. I have researched potential technologies and libraries available in the market today to find out which of these technologies are best suited for the development of this application.

Server side technologies

The following three programming languages were the candidate languages I have researched on to use in developing the server side of my application.

Java

“Java is a powerful, general-purpose programming environment. It is one of the most widely used programming languages in the world, and has been exceptionally successful in business and enterprise computing” [27]. It is an object oriented language and is platform independent making it powerful to use for application development in different platforms such as mobile development.

Python

“Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together” (python.org, 2015). Python is created by Guido Van Rossum and its implementation started during December 1989. It has since then became one of the leading programming languages used today. Python is portable and can be used on different multiple platforms. It is simple and easy to use and learn. According to Fabrizio Romano [28], “Python also has an extensive library, coming with an incredibly wide standard libraries. Python is used in a wide variety of applications such as web programming, gaming and robotics and rapid prototyping. Google, Dropbox and YouTube are three examples of the many companies who are using Python today".

MATLAB

MATLAB, also known as matrix laboratory, is a high-level language and interactive environment for technical computing used by millions of engineers and scientists worldwide (uk.mathworks.com, 2015). It was initially released in 1984, developed by MathWorks. It lets you explore and visualize ideas and collaborate across disciplines including signal and image processing, data analysis, exploration and visualization and modelling, simulation and prototyping (uk.mathworks.com, 2015). It provides an interface with programs that are developed in different languages such as Java and Python.

	Strength	**Weakness**
Java	1. Open Source 2. Familiarity 3. Ability to write server-side and client-side in java - allows sharing of load between the client and the server 4. Supports mobile development for native apps. 5. Platform independent - Allows development for multiple platforms 6. High performance	1. Low availability of good image processing libraries and resources 2. Slow development time - It is not suited for rapid prototyping
Python	1. Free and open source 2. Simple and coherent 3. Portability 4. Fast development time - "A python program is typically one-fifth or one-third of a java code" [28]. 5. Extensive library - Large collection of libraries and resource available for image processing including numpy, scikit-image and opencv. 6. Equally suited for both quick scripting and large development projects	1. Execution speed is slow 2. Unfamiliarity

MATLAB	1. Fast calculations and operations. 2. Simple 3. Large collection of available tools in different applications including image processing. 4. Large scientific community 5. Good documentation	1. Costly and not open source 2. Algorithms are proprietary - "Unable to see algorithm code. - Makes it difficult or impossible for third parties to extend its functionality" (pyzo.org, 2013). 3. Unfamiliarity

Conclusion

Each of these programming languages has their own strengths and weaknesses and I have taken this into consideration when selecting which programming language is best suited for my application. Development time, availability and library support are also three crucial selection criteria I have considered upon my decision. Due to these criteria, I have decided to use Python as my programming language for the server-side of my application. The wide availability of libraries and resources for image processing was one of the main criteria that made me chose Python. Even though it has a slow execution speed, "libraries used with python such as Numpy can achieve a C-like speed due to the way it is implemented. Python is also used in the backend of services such as Spotify and Instagram where performance is one of the main concerns" [28]. Thus, I believe that python is a suitable choice to use in the server side development of my application.

Python libraries for image processing

Python has a wide range of available libraries and resources in different applications including image processing. Here are a couple of python image processing libraries I have found and researched which could help me in the development of this application.

OpenCV

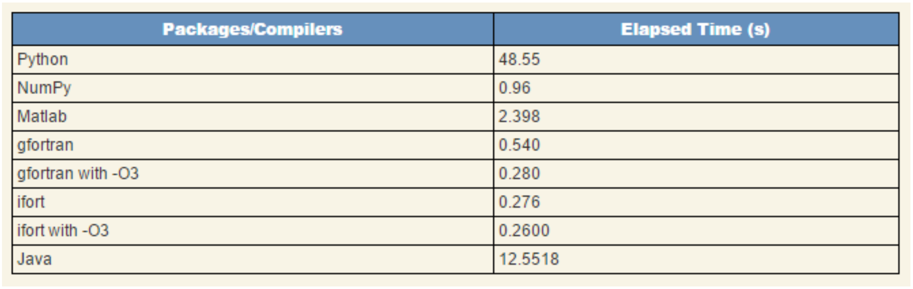
OpenCV, Open Source Computer Vision Library, is a free open source computer vision and machine learning library developed by Itseez in 1998. Since then, it has become a popular library in academic and commercial use for computer vision and image processing. "OpenCV was designed for computational efficiency with a strong focus on real-time applications" (opencv.org, 2015). Interfaces for C++, C, Python and java is available and it also supports different multiple platforms such as Windows, Linux, Mac, iOS and Android. OpenCV is written in optimized C/C++ enabling the library to take advantage of multi-core processing. OpenCV contains thousands of algorithms for computer vision, machine learning and image processing and is used in multiple applications. This includes face detection and recognition, image analysis, feature extraction and description and tracking of moving objects.

SciPy, Numpy and Scikit-image

Basic operations used in scientific programming includes multi-dimensional arrays, matrices, statics and more. However, python does not have any of these functionalities built in. It only contains basic mathematical operations that can deal with a variable. It cannot deal with arrays or matrices operations which are important in image processing. Python packages such as numpy and scipy enables python to be used efficiently for scientific purposes.

Numpy is a fundamental python package for scientific computing. It adds multi-dimensional arrays, element by element operations and core mathematical operations which are all important for image processing. Images are represented as a 2-dimensional or 3-dimensional array. For most calculations and operations of image processing, these n-dimensional arrays will be used. Numpy provides multi-dimensional array objects called ndarray which can only store the same type of element.

Python has a list object which can store nearly any type of python object as an element. It is very easy to construct and manipulate. It can store different types in one list. However, this means that only a few list operations can be carried out. Each iteration would require type checks and other python API bookkeeping. It is therefore not able to perform element by element operation such as addition and multiplication as efficiently as numpy. Numpy has faster operation speed in contrast to python lists due to this.



Packages/Compilers	Elapsed Time (s)
Python	48.55
NumPy	0.96
Matlab	2.398
gfortran	0.540
gfortran with -O3	0.280
ifort	0.276
ifort with -O3	0.2600
Java	12.5518

Figure 2.21: "The table above shows the elapsed time needed to do the array assignments" (modelingguru.nasa.gov, 2009)

"SciPy stack is a python-based ecosystem of open-source software for mathematics, science and engineering" (scipy.org, 2015). Its core packages contains open-source packages and libraries such as numpy, matplotlib and scipy library. The scipy library provides advanced and efficient mathematical functions. It has many uses in image processing such as reading input and displaying output images, basic image manipulations and many more. It also takes care of keeping array values in the right format when saving image files enabling more focus on processing the images. "Scikit-image is a free and open-source image processing toolkit for scipy" (scikit-image.org, 2015). It is written in python and was developed by the scipy community. Scikit-image provides algorithms used for image processing such as feature extraction and detection algorithms and corner detection.

Matlab

Matlab contains an image processing toolbox similar to scikit-image. "It provides a comprehensive set of reference-standard algorithms, functions and apps for image processing, analysis, visualization and algorithm development" (uk.mathworks.org, 2015). However, Matlab is not open-source and its licensing is very costly. Additionally, its source code for its functions and algorithms are proprietary preventing its users from learning from the code or modifying it for their specific needs.

I have decided to focus on using scikit-image due to its simplicity and its open source nature making it easier and faster to learn, allowing me to build and test prototypes quicker as well as enabling me to get familiar with python and the concept and techniques of image processing faster. However, using scikit-image also does not limit me to just use this library. It also allows me to integrate it with other libraries that uses numpy and python quite easily such as opencv, whereas this would be much harder to do with matlab. Due to its closed source nature, "it makes it difficult or impossible for third parties to extend its functionality" (pyzo.org, 2013).

Python Web Framework

Django

"Django is a free and open-source full stack web framework written in Python that follows a model view controller architecture" (djangoproject.com, 2015). Django has a large community support, great documentation and is a popular option as a python web framework amongst businesses today. It has a great admin interface, allowing the developer to interact with data models and business logic through a user interface from the start. Django, however, has a steep learning curve. Fitting pieces together is extremely challenging due to Django's class-based views. If the developer would like to change a subclass' behaviour slightly, he/she will have to look at a couple more subclasses as the subclasses itself is a hierarchy of subclasses.

Flask

"Flask is a free and open-source web micro-framework based on Werkzeug, a web server gateway interface utility library for Python, and Jinja2, a template engine for Python" (flask.pocoo.org, 2015). Even though it has a small community support, it is well documented and is enough to have a manageable learning curve as it is easy to understand. It is very simple and fast to set up and start, allowing for quick prototyping. It is easily extensible with many external extensions like flask-restful available for free. It also even has an integrated support for unit testing.
For my application, I chose to use Flask as my web framework. Unlike Django, flask is very easy to customize, the developer does not have to go through multiple classes in order to change one functions behaviour slightly. It is also simpler and easy to learn, allowing for a quick start up. Django offers a lot of functionalities and components but I believe it is too complex for my requirements of implementing a simple REST API server.

Image Comparison Algorithms

Mean Squared Error (MSE)

According to Zhou Wang and Alan Bovik [29], Mean squared error (MSE) can be used as an image similarity measure. "The goal of an image similarity measure is to compare two pixels by providing a quantitative score that describes the degree of similarity or the level of error between them" [29]. MSE measures the average of the difference between an image and the image that it is being compared to. This is done by calculating the differences between each pixels of the 2 images. A result of zero means that the image are perfectly the same. The closer the result is to zero, the more similar the image is to the one that it is being compared to. MSE is very simple to implement and is inexpensive to compute. However, it is inaccurate of its prediction of human perception on image similarity as these errors, or difference between each pixels, does not necessarily mean that the contents of the image are different. MSE can also only be applied to images that have the same dimension.

Structural similarity index measure (SSIM)
“Structural similarity index measure’s (SSIM) principle philosophy is that human visual system highly adapted to extract structural information from visual scenes, therefore the digital image structure should be an important ingredient for image similarity measurement” [29]. Local image patches, small groups of nearby pixels, are “taken from the same location of two images that are being compared. The similarity of these local image patches are measured according to three elements – contrast, brightness level and its structure which are combined together to form a local SSIM” [29]. SSIM perceives change in the structural information of an image and also taking its contrast and brightness level into account in contrast to MSE which just perceives the difference between two pixels. SSIM has been implemented by scikit-image as part of the algorithms that they provide. SSIM also requires that the images being compared have the same dimensions and that the images have to be represented in a 2d-array.

MSE and SSIM are simple to implement however they are simply not enough to use for comparing images that are taken from different angles or scaled differently due to its non-scale and non-rotation invariance. Therefore, the use of feature extraction and description techniques would be better suited for this functionality. There are several feature extraction and description algorithms available and implemented by different libraries.

Feature Extraction & Detection

SIFT
Scale-invariant feature transform (SIFT) is an algorithm published by David Lowe in 1999 which detects and describes local features in an image. According to Kenneth Dawsone-Howe [30], SIFT has four main stages for feature extraction and description:

1. **Scale-space extrema detection**
   This involves scaling an image in a number of octaves in order to make the detection of features scale invariant. “The image’s resolution is lowered by convolving it with a number of Gaussians” [30]. After convolution, the smallest recognizable detail in the image has become larger and more noticeable. The images are stacked with the original image at the bottom. The higher the image is in the stack, the lower its resolution. This stack of images are called Gaussian Scale-Space. This allows the selection of an image at a certain level of resolution.
   The difference of Gaussian (DoG) is calculated across the various scale spaces. “To find the extrema of a scale-space, the minimum and highest point in a scale-space, multiple DoG images and locate points are considered which are greater (or less) than all of their neighbours both in the current scale and in the neighbouring scales (higher and lower)” [30]. Each local extrema can be potential key-points that are best represented in that scale.

2. **Key-point localization**
   SIFT then checks for the accuracy of each key-points found. In order to do this, the data is modelled locally using a 3-d quadratic which makes it possible

to find the interpolated maximum and minimum. There are two test performed in this stage in order to check for reliable key-points. First, it checks the local contrast in the area around the key-point. Second, it checks the localisation of the key-point, for example whether it is on edge. All key-point features that did not satisfy these tests are discarded.

3. **Key-point orientation computation**
   Now that sift has handled scale invariance, it must also make sure that features detected are also rotation invariant. SIFT does this by assigning each feature a specific orientation. The scale of the key-point is used to select the Gaussian smoothed image closest to the scale and for each image sample, the gradient magnitude and orientation is computed.

4. **Key-point descriptor extraction**
   In this stage, the region is described around the key-point. This allows for comparison with other key-points in image that it is being compared to. Image that has the closest scale is used as well as sample points surrounding the key-point, computing their gradients and orientations. The image can be rotated using the key-point orientation for rotation invariance.

Once the features has been extracted and described, SIFT is then able to match the key-points to the key-points of the image that it is being compared to.

ORB

Oriented Fast and Rotated Brief (ORB) is another algorithm that is used for feature extraction and description. “It was developed at the OpenCV labs by Ethan Rublee, Vincent Rabaud, Kurt Konolige and Gary R. Bradski in 2011 as an efficient and viable alternative to SIFT and SURF” [10]. According to Thakkar and Kapur [10], the following methods are used in ORB for feature extraction and description:

1. **Oriented Fast Detection (oFAST)**
   Key-points are detected using oriented fast detection method and is the first stage of the algorithm. A parameter is used by FAST which is set by the user or uses the default value of nine which gives it a good performance. “This value is the threshold of the ring radius between the centre pixel and the circular ring surrounding it” [10]. Harris corner is used to overcome the issue of key-points being detected along the edges of the image. A threshold of N key-points can be set which depicts the total amount of key-points which we want it to find. It then orders the key-points according to its Harris corner measure. The top N key-points are then picked while the rest are discarded. FAST does not produce multiscale features, however, ORB employs a scale pyramid of the image to produce features at each level in the pyramid to overcome this scale invariance issue. To compensate for its rotation invariance, ORB uses a measure of corner orientation called intensity centroid.

2. **Rotated BRIEF**
   ORB uses BRIEF for feature description. However it performs poorly with rotation. ORB overcomes this issue by steering BRIEF according to the key-points’ orientation. A matrix is defined for a feature set which contains the coordinates of these pixels. This is then rotated using the orientation of the

key-point to construct a steered version of that matrix. "As long as the key-point orientation is consistent across views, the correct set of steered points will be used to compute the descriptor eliminating its issue of rotation invariance. BRIEF has an important property that each bit feature has a large variance and a mean near 0.5" [10]. However, this property is lost once it is oriented towards the key-point direction. ORB resolves this by running all possible binary tests to find each bit features that have both a large variance and a mean close to 0.5.

Once the features has been extracted and described, the resulting key-point features are matched with those in the image that it is being compared to. ORB is one of the algorithms that scikit-image has already implemented and provided. Scikit-image uses a hamming distance measure for matching these key-point features. This measures the minimum number of substitution that is required to make in order to transform one string to another. In this case, a key-point may hold the value of 0010 and the key-point that it is being matched to holds the value of 1010. The resulting distance in this case is 1, as 1 bit is required to change in order to match the other key-point value. SIFT has been a very successful key-point detector and descriptor. "However, it imposes a large computational burden, especially for real-time systems or for low-power devices such as phones" [31].

ORB is a good replacement for SIFT as it performs as well as SIFT but imposes a lesser computational burden and has an excellent speed of execution. ORB was tested and compared to SIFT by Rublee, Rabaud, Konolige and Bradski [31] and according to their findings, ORB performs faster and better than SIFT under certain circumstances. ORB is relatively immune to Gaussian image noise, unlike SIFT[31].

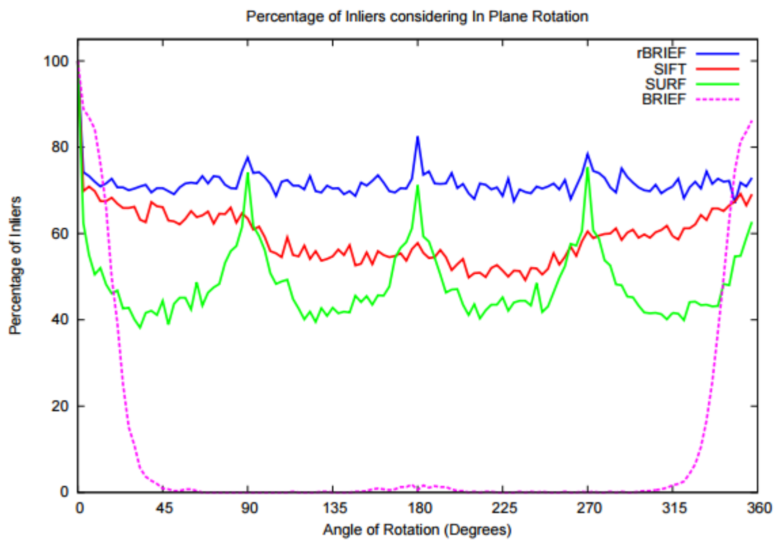


Figure 2.22: "Match performance of rBRIEF, SIFT, SURF and BRIEF under a synthetic rotation with a Gaussian noise of 10. rBRIEF results in a lower drop rate in comparison to SIFT." [31]

ORB also works better than SIFT and SURF on outdoor datasets which is shown in the following figure.

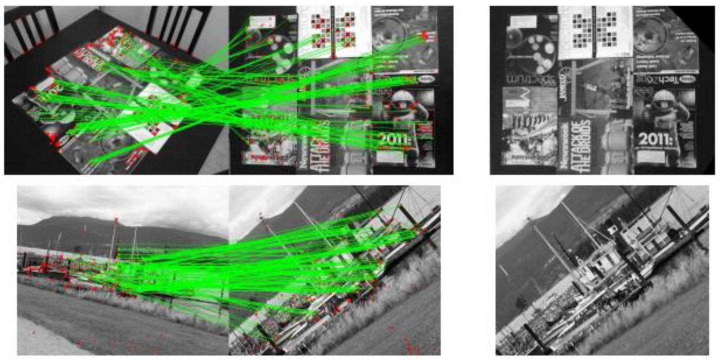
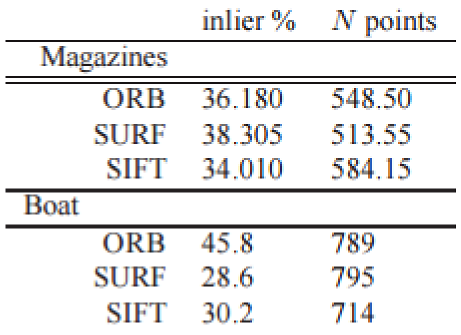


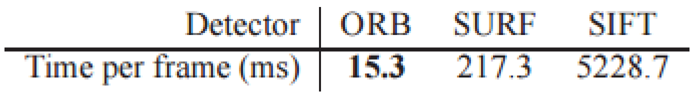
Figure 2.23: “Picture of magazines is taken indoors whereas the picture of the boat is taken outdoors.” [31]



	inlier %	N points
Magazines		
ORB	36.180	548.50
SURF	38.305	513.55
SIFT	34.010	584.15
Boat		
ORB	45.8	789
SURF	28.6	795
SIFT	30.2	714

Figure 2.24 “The resulting percentage of inlier matches are almost the same for the magazine which is taken indoors but the results of ORB for the outdoor picture is significantly higher than that of SURF and SIFT.” [31]

ORB is also able to detect and compute features faster than SIFT and SURF on the same data, for the same number of features and scales.



Detector	ORB	SURF	SIFT
Time per frame (ms)	**15.3**	217.3	5228.7

Figure 2.25: “Average times of 24 images (640x480) from the Pascal dataset.” [31]

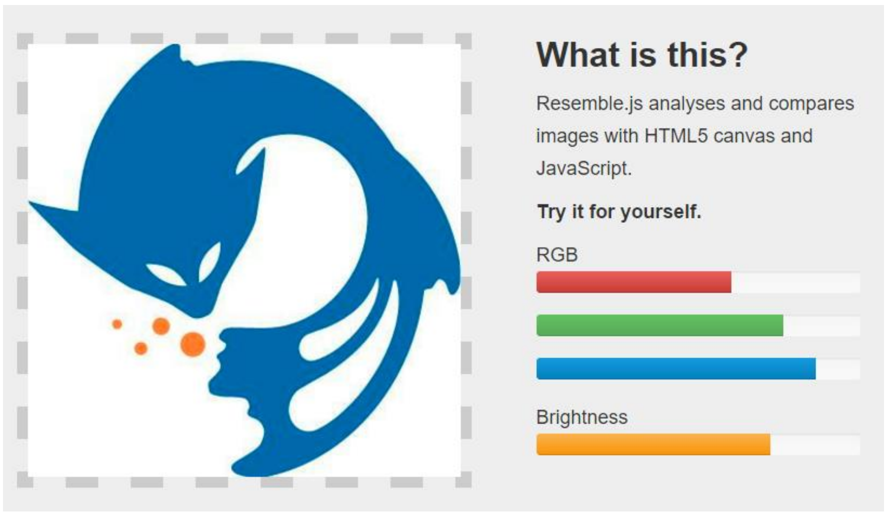
Due to these findings, I have decided to use ORB as a feature extractor and description for image comparison for this application.

Similar Image Analysis and Comparison Solutions

Resemble.js: Image analysis and comparison

Resemble.js is a JavaScript library created by James Cryer and the Huddle development team. This library can be used to compare and analyse images in browsers using HTML5 canvas and JavaScript. It also uses HTML5 file API to parse image data and canvas for rendering different types of images.

Resemble.js provides two features and these are both demonstrated on their demo page. First, this library enables to analyse images in browsers. The user can select an image in their directory to be analysed. In the demo, for example, the user can simply drag and drop an image from their directory that they would wish to be analysed into an input box. This allows resemble.js to read the image and analyse its RGB spectrum and brightness levels. The result of this analysis is displayed as a bar chart which shows the values of red, green, blue and brightness level of the image.



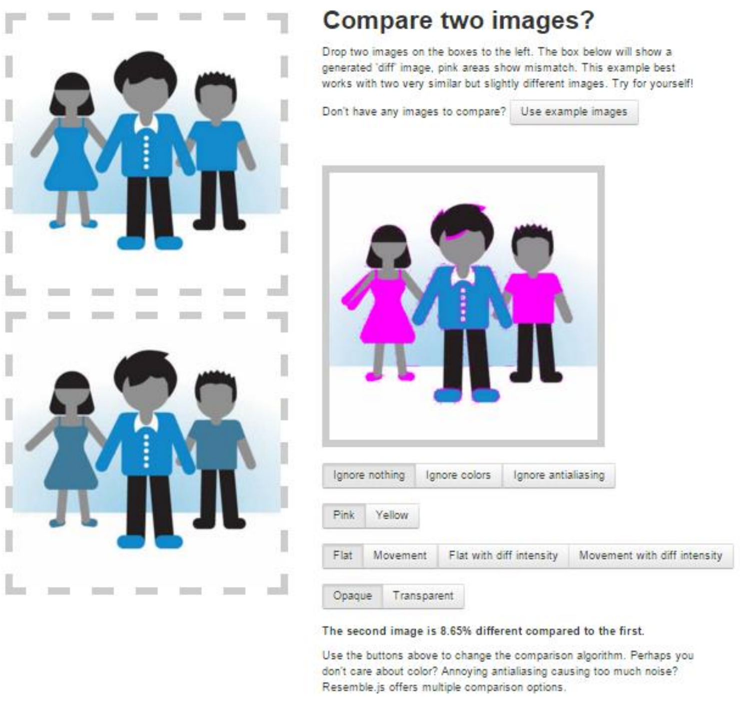
What is this?

Resemble.js analyses and compares images with HTML5 canvas and JavaScript.

Try it for yourself.

Figure 2.26: Resemble.js' image analysis demo

This library also provides image comparison. The user can select two images from their directory to be compared. In the demo, this has the same feature as the image analysis where the user can simply drag and drop their images into an input box. The two images compared are combined in one picture, displaying the results of the comparison. Each section where the image is deemed to be different is highlighted to show the difference between the two images. A percentage of how much the two images differ is also calculated and is shown with the results. There are also multiple comparison options available including a choice of ignoring the image's colour or anti-aliasing to increase the accuracy of the comparison between the two pictures.



Compare two images?

Drop two images on the boxes to the left. The box below will show a generated 'diff' image, pink areas show mismatch. This example best works with two very similar but slightly different images. Try for yourself!

Don't have any images to compare? Use example images

Ignore nothing | Ignore colors | Ignore antialiasing

Pink | Yellow

Flat | Movement | Flat with diff intensity | Movement with diff intensity

Opaque | Transparent

The second image is 8.65% different compared to the first.

Use the buttons above to change the comparison algorithm. Perhaps you don't care about color? Annoying antialiasing causing too much noise? Resemble.js offers multiple comparison options.

Figure 2.27: Resemble.js image comparison demo

Since Resemble.js is a library, anyone can implement this in their own work by just downloading and installing the library in their own workspace. Another benefit of this library is that it is written in JavaScript which makes it much easier to implement in browser and integrate it with other web development languages. Processing of the images is also done in browser without having to upload the image to a server. It also provides an image threshold which causes it to ignore pixels past the threshold to increase the speed of the comparison. Multiple comparison options are also available which can provide for more accurate results.

Despite of having many benefits, it also has a couple of disadvantages on its functionality. The image analysis only analyses an image's RGB spectrum and brightness level. It does not provide any analysis on the image's quality at all. Its comparison algorithm also only compares the images pixel by pixel. It checks to see if the colour and brightness level in each pixel is the same. Therefore, the images to be compared must be almost similar for it to be considered the same. It is not able to handle similar pictures in a different scale, angle or rotation.

Gallery Doctor

Gallery Doctor is an Android and iOS application developed by Flavyr Media and released as part of MyRoll products. "It is the only application released in the market which automatically identifies photos with bad lighting, blurry spots, similar photos and even photos that the user might find boring for mobile devices" (gallerydoctor.com, 2015).

Gallery Doctor first scans the user's image folders on their mobile device. The application scans all of the folders as a default but it also offers a selection of folders containing images on the user's device which can be selected to be included or excluded in the scan. The scan may take up to a couple of seconds to a few minutes depending on how many images are contained in the folders that is being scanned.

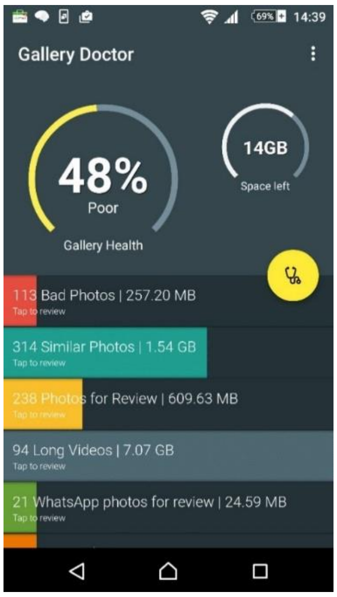


Figure 2.28: Shows the results of Gallery Doctor's scan used in an android device.

The results of the scan are shown in figure 2.4. It shows the percentage and status of the 'health' of the user's device. This percentage represents how many bad quality and similar photos are taking up space in the user's device's memory. It also shows the amount of space that is free in the user's device. At the bottom half of the screen, it groups each images according to its results in a user-friendly and easily readable format. The results does not just include images but it also includes videos. Each of these sections can be reviewed further by tapping on it.

For example, in the similar photos section, it groups each images with similar pictures in the album. Each set of pictures shows which photo is the best out of the set while the other photos who are deemed to have a lower quality is marked for deletion. Any of these photos can be marked or unmarked for deletion. It also has an option to clear each set of pictures for quicker deletion. Once the user has decided to delete all of the marked photos, the user can click on the delete button at the button of the screen, instantly deleting all of the marked photos. It also shows how much space will be recovered by deleting these photos.

This application is very easy to use and the scanning speed is quite fast. The layout and style of the application was done in a very colourful and user-friendly way. However, I found that the results of the selection of bad photos were not very accurate. The results included some good quality screenshots which some are not bright in colour. Since the application is testing for brightness level, it might have deemed it having a bad quality due to its darker range of colour. It also included images that are focused on a subject with a blurred background. This can be seen in the figure 2.5

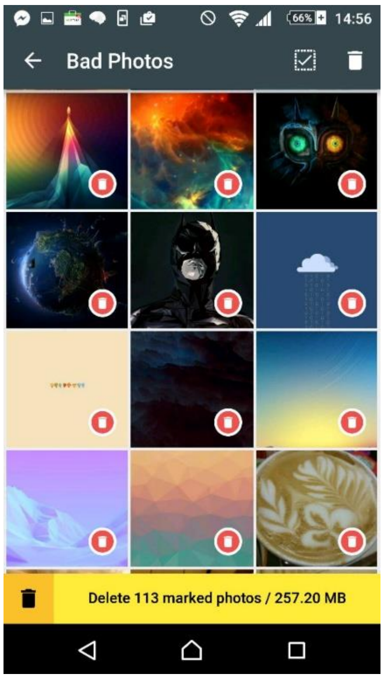


Figure 2.29: Shows resulting images that are deemed to have bad quality.

Chapter 3: Design & Methodology

Introduction

Chapter 3 will focus on the design methodology chosen for the development of this project as well as the design of the system architecture and of different project components. Any software development requires the use of a design methodology in order for the project to be a success. Its aim is to make sure that the project will be able to meet with the functional requirements discussed at the beginning of the project as well as enabling these functional requirements to be evaluated and tested. A clear design of different project components would make it easier to understand the proposed functional requirements.

Design Methodology

"Agile software development is a collection of software methods in which response to changing requirements; priority of requirement, adaptive planning, iterative and evolutionary development and early delivery of product" [32]. According to Amitoj Singh [32], agile development is a much better approach than traditional approaches such as the waterfall model as it is much more efficient when changing requirements throughout the development. This is very useful as evaluation of features is an important factor in development. Agile development also delivers tested working software frequently unlike traditional approaches where software is only delivered and tested at the very end of development. Since my project's requirements were often changing, some functionality or components may change according to evaluation, I believe that this is the best approach for my project development.

The agile methodology that I will be using for my project development is scrum. "Scrum is an agile project management framework first introduced in 1999 by Ken Schwaber" [33]. According to Tilo Linz [33], the main aim of using scrum is to enable a project team to react quickly, simply and appropriately. Schwaber" [33].

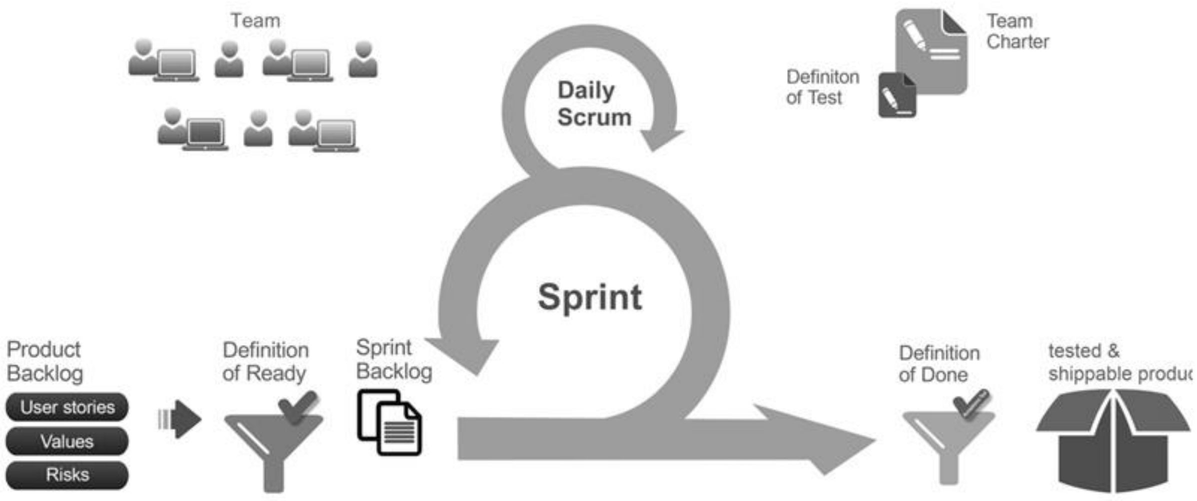


Figure 3.1 shows a diagram of scrum development. [33]

Scrum contains the following features in its development.

1. **Product Backlog:** “The product backlog contains all known requirements and results that have to be implemented or achieved to complete the project successfully” [33]. My product backlog will contain functional requirements such as image comparison, image analysis and assessment and image upload. It will also contain non-functional requirements such as usability and efficiency. These requirements may also be broken down further into sub-categories to understand them better.
2. **Sprint:** “Scrum divides a project into short, fixed-length iterations called Sprints” [33]. For my project, each sprint will be done in a short matter of time, lasting two to four weeks of development. This makes each iteration manageable and easier to plan. “Each sprint, the product is incremented, resulting in a working and potentially releasable product.” [33] At the end of each sprints, the requirements in the product backlog may be subjected to change upon its evaluation.
3. **Definition of Ready:** Requirements that has been completely understood is defined as ready to be put in the sprint backlog.
4. **Sprint Backlog:** This contains the requirements that are needed to be implemented during the sprint phase. Requirements with higher priority that has been deemed ready are taken first from the product backlog.
5. **Definition of Done:** Each product increment is assessed according to multiple criteria and tests to check if it is working and complete. Product increments that complies with these criteria and tests are deemed as done.

Design of Project Components

Algorithm Design

The selection and upload of images to the server by the client and the image comparison and analysis in the server are the three main components of my project therefore it is important to have a design and a structure of how these key functional requirements are to be implemented. This will make it easier to implement as well as making sure to have a clear understanding of these functional requirements.

Image Comparison

The image comparison of the uploaded images is one of the main functional requirements to be implemented in the server, as mentioned above. Upon the server’s acceptance of a client request, the images uploaded by the client are put into a list. This list of images are then compared against each other for categorization of images that are similar to each other. The following figure illustrates this workflow.

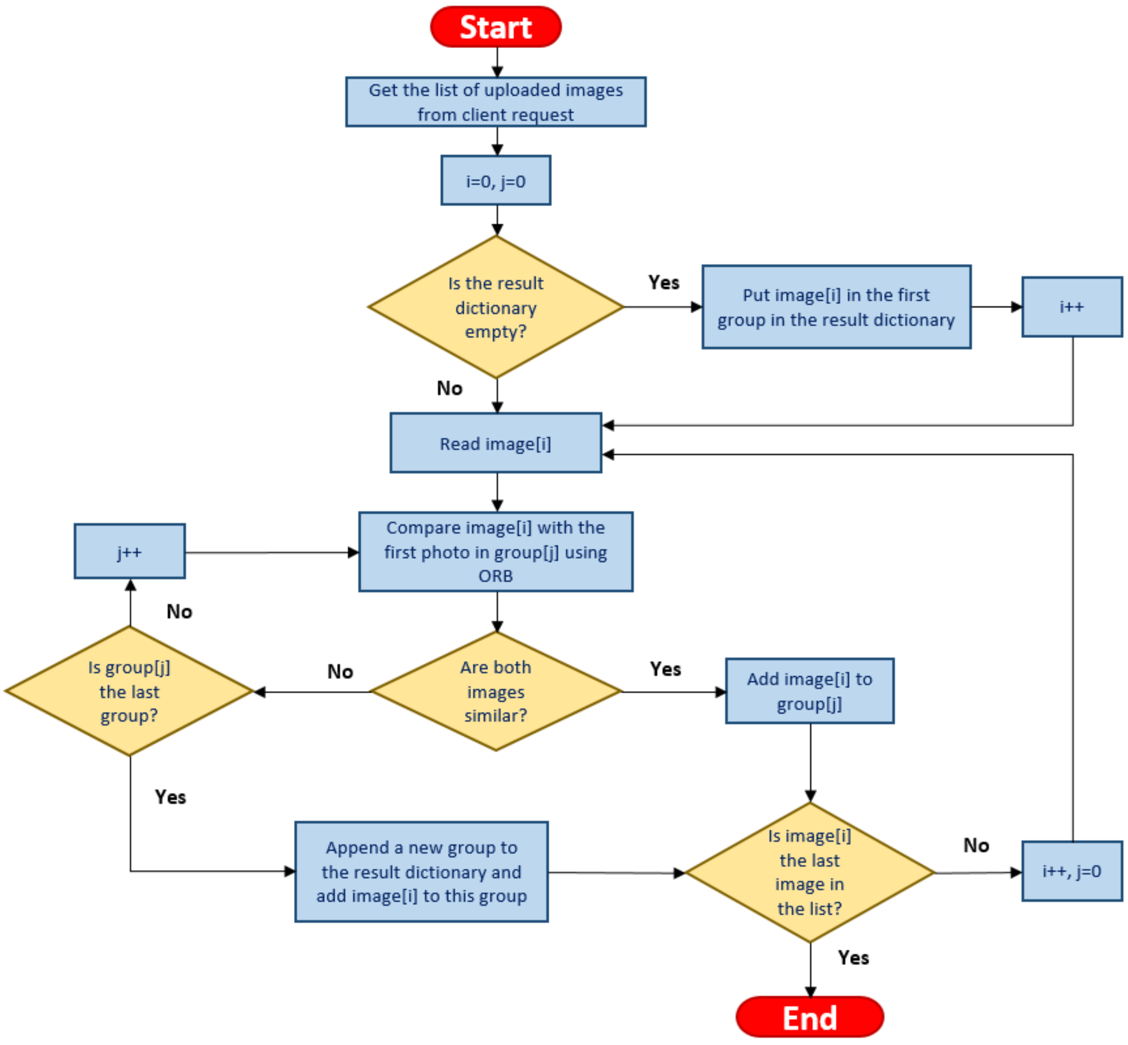

Start
Get the list of uploaded images from client request
i=0, j=0
Is the result dictionary empty?
Yes
Put image[i] in the first group in the result dictionary
i++
No
Read image[i]
Compare image[i] with the first photo in group[j] using ORB
j++
No
Is group[j] the last group?
No
Are both images similar?
Yes
Add image[i] to group[j]
Yes
Append a new group to the result dictionary and add image[i] to this group
Is image[i] the last image in the list?
No
i++, j=0
Yes
End

Figure 3.2: Image comparison workflow

The flow of this algorithm is as follows:

1. The workflow starts by extracting the uploaded images from the client request as a list.
2. Once the list has been created, go through the list starting from the first image. If the result dictionary is empty, add the current image as the first entry.
3. If the result dictionary is not empty, read the current image as a Numpy array.
4. Compare the current image with the first photo listed in the first group in the result dictionary using ORB.
5. If both of the images are deemed to be similar, add this image to the group of the photo it was compared to. If the current image was the last image in the list, the comparison ends and the result of the comparison can be then seen in the result dictionary. Otherwise, go onto the next image and repeat the steps from step 3.
6. On the other hand, if the current group is not the last group in the result dictionary, repeat the steps from step 4 with the next group listed in the result dictionary. However, if the current group was the last group in the result dictionary, create a new group in the dictionary and add the current image to that group. If the current image is then the last image in the list, the image

comparison ends, otherwise, the steps are repeated from step 3 with the next image on the list.

Image Selection and Upload

The last main functional requirement of this application is to allow the user to select images that they would like to be analysed and upload them to the server. The process is simple enough with the user triggering the start of the process by clicking the browse button in the application. The user selects image(s) of their choice. Once the user is finished their selection, these set of pictures are prepared for upload to the server. The application reads the image first and adds this to the form data. Once all of the images have been added to the form data successfully, the form data is then sent to the server using a HTTP POST request. This process is illustrated by figure 3.3.

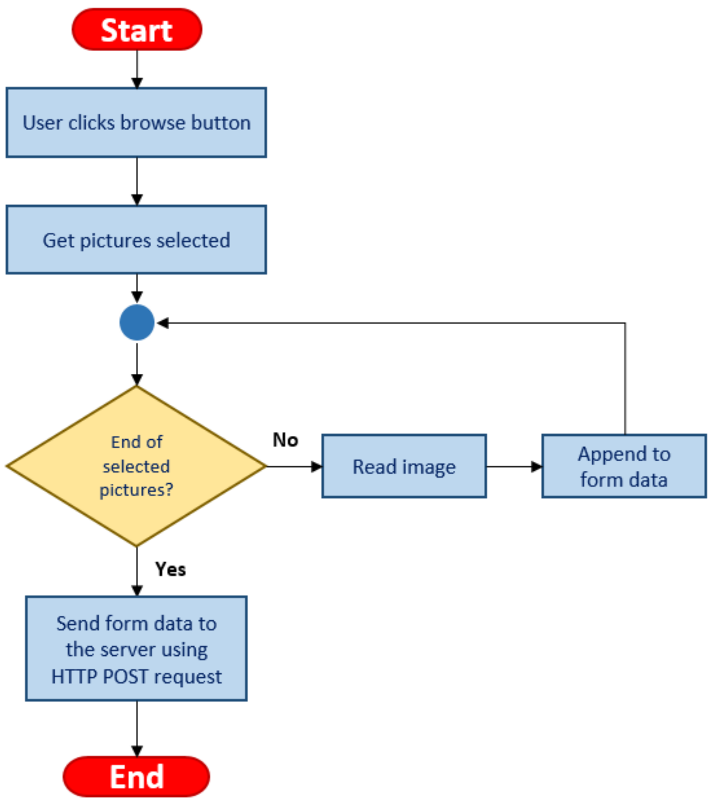

Start
User clicks browse button
Get pictures selected
End of selected pictures?
No
Read image
Append to form data
Yes
Send form data to the server using HTTP POST request
End

Figure 3.3: Illustrates the image selection and upload workflow.

Sample Sequence Diagrams

Client-Server Communication

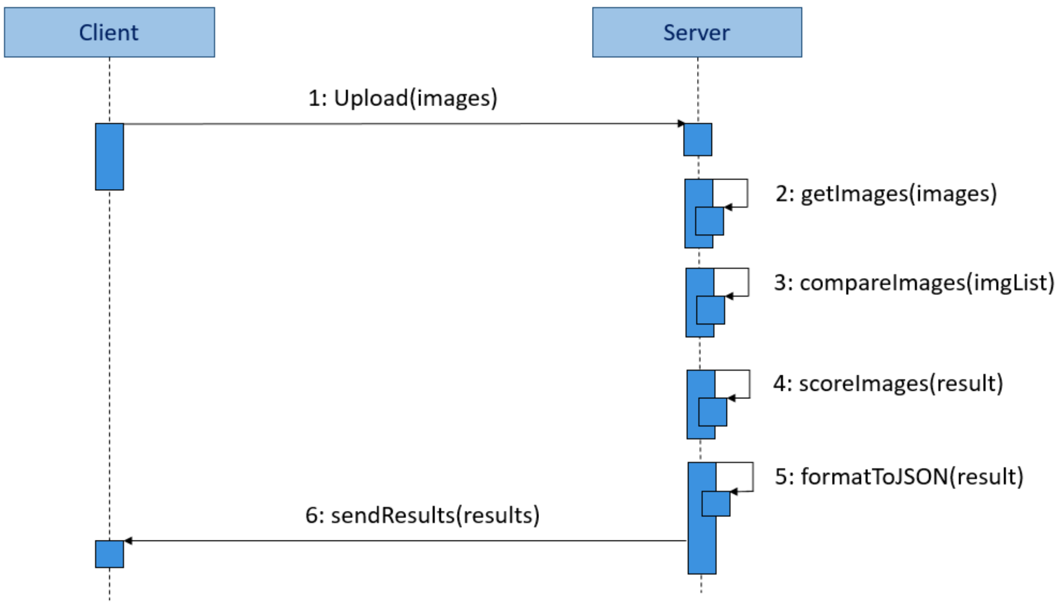


Figure 3.4: Sequence diagram which illustrates client and server communication and their behaviour.

The client starts the communication by sending a HTTP Post request to the server with the uploaded images as form data. The server receives this request and extracts the images uploaded from the request and put them into a list. The server then go through each of the images in the list and compares them with each other and groups them according to their similarity, recording the results in a dictionary. The results from the image comparison are then passed onto the next functionality for scoring the images based on their quality. The score for each image are added to the result dictionary. Once all the images have been scored, the result dictionary is then formatted into a JSON string which is then sent back to the client for displaying the results to the user.

Relevant Prototypes

This section consists of the Cordova application UI mock-ups used for the development of this application as a base design of what the final application may look like

Cordova Application UI Mock-up

"A UI mock-up consists of one or several static pictures that resemble the future UI of the application in as many aspects as possible" [34]. The following figure illustrates the mock-up screens for the Cordova application. This illustrates how the user can interact with the application, showing the process of how they can traverse from screen to screen. This includes the home page, which will be the starting point of the application, the image selection page and the result page. This mock-up also shows the main functional requirements to be implemented in the Cordova application. A live demo of this mock-up can be viewed here.

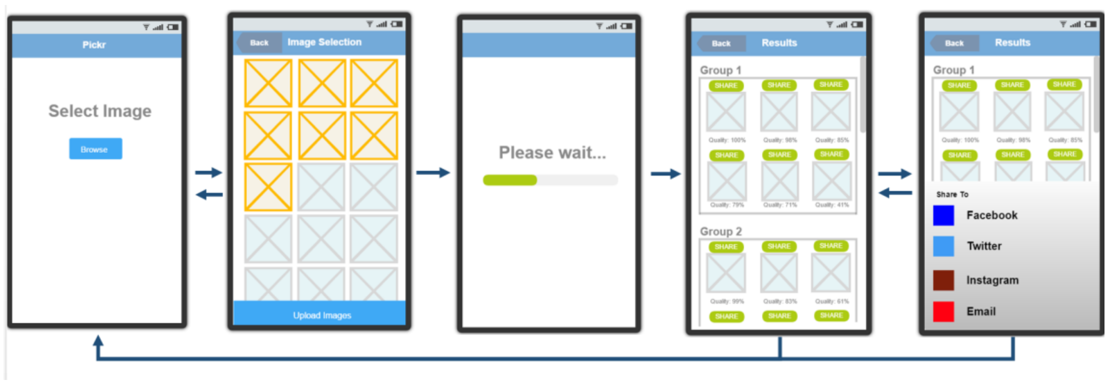


Figure 3.5 shows the mock-up screens of the user interface for the mobile application.

Home Screen: The home screen will be the first page the user will see once they start the application. The user can click on the browse button which will then take them to the image selection screen.

Image Selection Screen: In this page, the user can select images, which will then be highlighted. The user can then decide if they would like to upload these highlighted images for comparison and analysis, which will then take them to a loading screen until it receives the server's response. If the user decided to cancel, they can click the back button to go back to the home page.

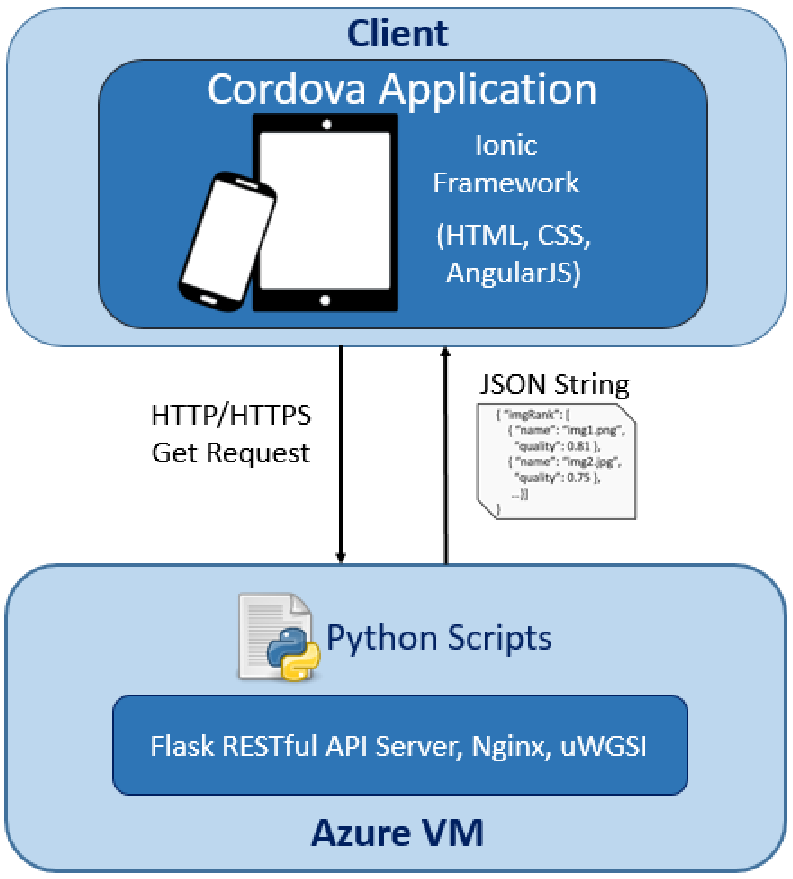
Result Screen: Once the application has received a response from the server, it is directed to the result page which displays the grouped images as well as their scores. The images are sorted in each of their own group according to their quality score. Above every image, there is a share button which the user can click in order to share their selected image via social media sites, email or messaging. When this share button is clicked, a native sharing window of the mobile device is shown, with the list of application which the image can be shared. When the user is finished reviewing the results, they can go back to the home page by clicking the back button, otherwise they can simply exit the application.

Chapter 4: Architecture & Development

Chapter 4 focuses on the architecture and the development of the system, discussing the implementation and integration of each system component as well as the various decisions and challenges faced throughout the development of the system.

System Overview

The system to be developed implements a two-tier architecture which consists of the client in the top-tier and the server in the bottom-tier. The system's overview is illustrated in the figure below.



```
{ "imgRank": [
   { "name": "img1.png",
    "quality": 0.81 },
   { "name": "img2.jpg",
    "quality": 0.75 },
   ...]
}
```

Figure 4.1: System overview

Client: The client consists of a Cordova application created using HTML, CSS and AngularJS and deployed using Ionic Framework. The Cordova application can send a HTTP/HTTPS POST request to the server, allowing images to be uploaded to the server for image comparison and analysis. The client will act as a user interface to allow the user to select images from their device's photo album. The application will also receive a response from the server, consisting of a JSON string containing the results of the image comparison and analysis. The results received shall be displayed in the client for the user to review.

Server: The server is hosted in an Azure virtual machine consisting of Python scripts which implements a RESTAPI. Requests sent by the client are served by Nginx, a free, open-source, high-performance HTTP server and are then passed to the uWSGI, a web server gate interface (WSGI) implementation, which will translate these requests into a format that the application can process. The images uploaded by the client are analysed here for comparison and assessment. The result of the finished analysis are recorded and sent back to the client as a JSON string.

Development Environment

Server

The server utilizes various technologies, mentioned in chapter 2, which are needed to be setup and integrated together in order to proceed in the development of this system component. The server is hosted in an Azure Linux virtual machine which already came with a variety of development tools I required pre-installed such as Python 2.7.6

and pip. This made it easier to set up the server environment by using pip to install the rest of the packages required in order to proceed with the development of this system.

The following table lists the tools and libraries that were installed in the server using pip along with their dependencies:

Tools/Library	Dependencies
Flask	Jinja2, Werkzeug
Flask-Cors	
Scikit-Image	Matplotlib, Numpy, Networkx, SciPy, Six, Pillow
uWSGI	
Nginx	

Figure 4.2: Tools and libraries installed along with their dependencies.

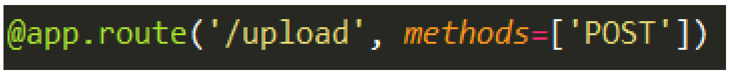
System Components

Server

The server is a core component of the system which handles requests received that are sent by the client. It consists of a flask application which implements a RESTful API which handles the image processing functionality in the system.

Routes

The **/upload** route is the only route available in this Flask application. This route is used by the client for uploading their selected images to the server by an HTTP POST method. Therefore this route is defined as shown in figure 4.3. This registers a view function for the /upload URL which is limited to the POST method.



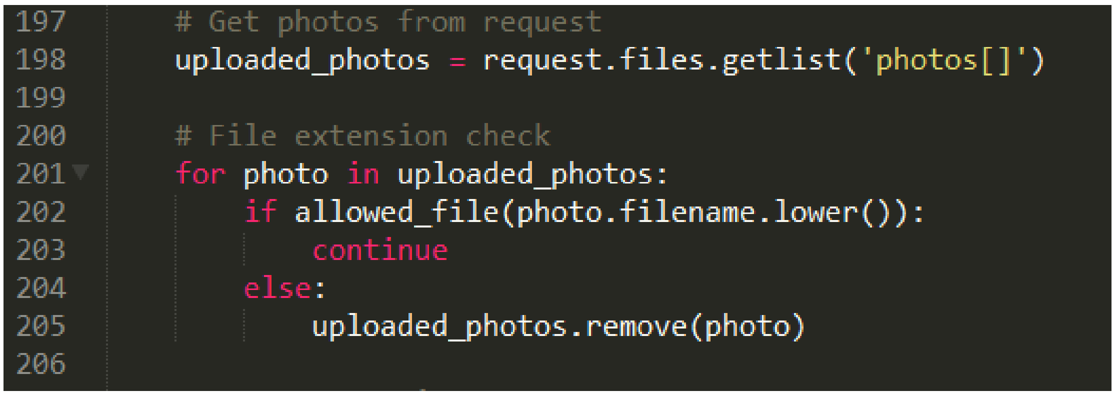
```
@app.route('/upload', methods=['POST'])
```

Figure 4.3: /upload route

This route consists several functions including the two of the core functional requirements of the system: the image comparison and quality assessment.

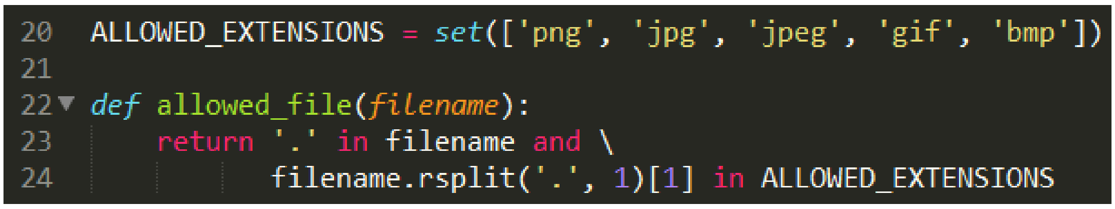
File extension check

This function is applied after the photos has been extracted from the request. Each of the images in the list are checked if their file extension is included in the allowed file list which contains a variety of different image file types such as ‘jpg’, ‘jpeg’ and ‘gif’. This is done to make sure that no invalid files proceed to the image processing functions as this will cause an error if the application attempts on reading a non-image file as an image.



```
    # Get photos from request
    uploaded_photos = request.files.getlist('photos[]')

    # File extension check
    for photo in uploaded_photos:
        if allowed_file(photo.filename.lower()):
            continue
        else:
            uploaded_photos.remove(photo)

```

Figure 4.4: An extract of upload.py file which shows photo extraction from the client request and the file extension check which function is defined as shown in figure 4.5



```
ALLOWED_EXTENSIONS = set(['png', 'jpg', 'jpeg', 'gif', 'bmp'])

def allowed_file(filename):
    return '.' in filename and \
        filename.rsplit('.', 1)[1] in ALLOWED_EXTENSIONS
```

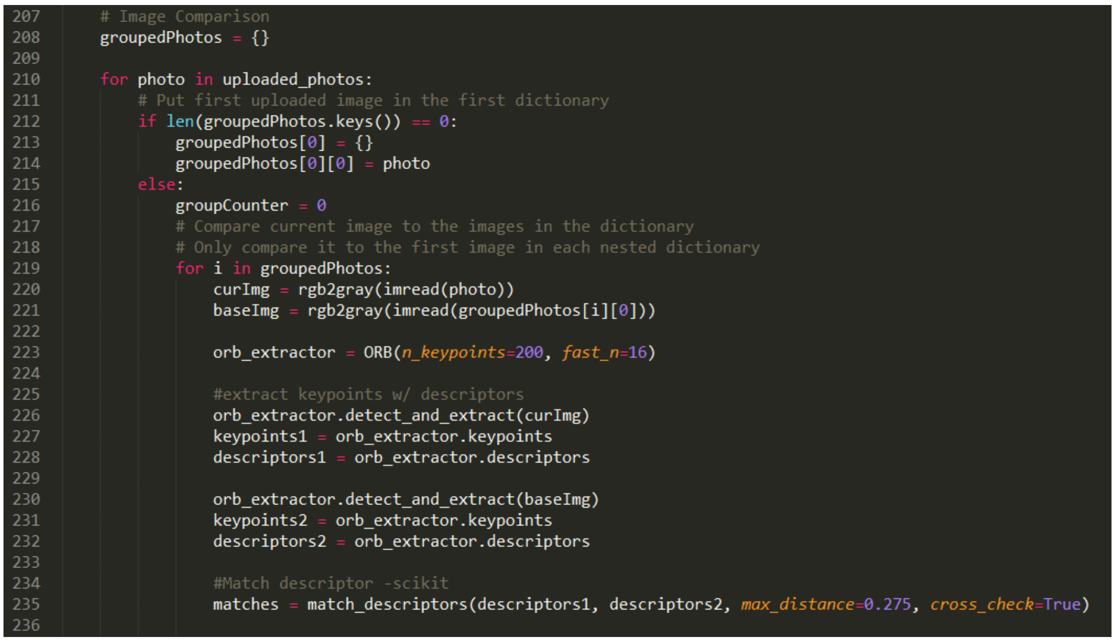
Figure 4.5: An extract of upload.py file which shows how the allowed_file() function is defined as well as the list of allowed file extensions the check is based on.

Image Comparison

Image comparison is one of my core functional requirements that was needed to be implemented in the server. The feature matching algorithm used in this process is ORB, as mentioned in chapter 2. First, an empty dictionary, *groupedPhotos*, is initialized which will later on contain the result of the image comparison. The next step is to go through the list of uploaded images, *uploaded_photos*, one by one using a for loop. It first checks if the image is the first photo in the list of uploaded images. If so, it is added as the first entry in the dictionary as there is no image it would be compared to. Otherwise, it is compared to the first image in each nested dictionary of the *groupedPhotos*, iterating through it.

A variable called *groupCounter* is initialized to 0 in each iteration. This serves as a counter for each nested dictionary in *groupedPhotos*, representing a group containing images that are deemed to be similar by ORB. The current and base image, first photo in the group, are read using the scikit-image's imread function which returns a Numpy three-dimensional array (RGB). This is then converted to a grey scale image using scikit-image's rgb2gray function, normalizing its values to a range of 0 to 1. ORB is then defined with the parameters *n_keypoints* with a value of 200 and *fast_n* with a value of 16. The parameter n_keypoints defines the amount of key points returned by the algorithm "according to the Harris corner response if more than the defined amount is detected" [35]. The fast_n parameter defines "the minimum number of consecutive pixels out of 16 pixels on the circle that should all be either brighter or darker in regards to the current pixel" [35]. Both of these parameters are optional and are defined in order for the result to be more accurate.

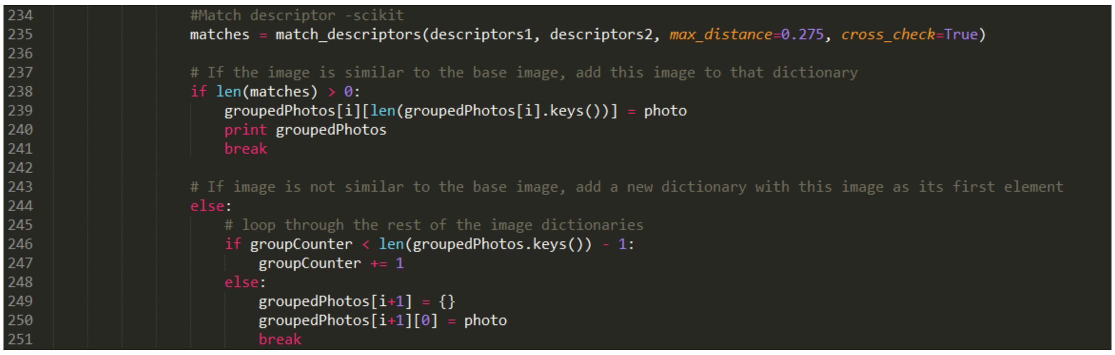
Key points and descriptors in each of the images are detected and extracted using scikit-image's *detect_and_extract* function and matched using scikit-image's *match_descriptors* function. *Match_descriptor* is a brute force matching algorithm implanted by scikit for matching descriptors, finding the closest descriptor in the second set of descriptors. It has two optional parameters, *max_distance* and *cross_check* defined. The *max_distance* parameter is the maximum value allowed for the distance between the two descriptors being compared in order to be regarded as a match. This value is chosen as it was the optimal value in regards to accuracy of the match. The *cross_check* parameter activates cross checking between matched key points if it is defined as true. This has been activated in order to receive a more accurate comparison result.



```
    # Image Comparison
    groupedPhotos = {}

    for photo in uploaded_photos:
        # Put first uploaded image in the first dictionary
        if len(groupedPhotos.keys()) == 0:
            groupedPhotos[0] = {}
            groupedPhotos[0][0] = photo
        else:
            groupCounter = 0
            # Compare current image to the images in the dictionary
            # Only compare it to the first image in each nested dictionary
            for i in groupedPhotos:
                curImg = rgb2gray(imread(photo))
                baseImg = rgb2gray(imread(groupedPhotos[i][0]))

                orb_extractor = ORB(n_keypoints=200, fast_n=16)

                #extract keypoints w/ descriptors
                orb_extractor.detect_and_extract(curImg)
                keypoints1 = orb_extractor.keypoints
                descriptors1 = orb_extractor.descriptors

                orb_extractor.detect_and_extract(baseImg)
                keypoints2 = orb_extractor.keypoints
                descriptors2 = orb_extractor.descriptors

                #Match descriptor -scikit
                matches = match_descriptors(descriptors1, descriptors2, max_distance=0.275, cross_check=True)

```

Figure 4.6: An extract from upload.py which shows the image comparison implementation, continued in figure 4.6

The results of the matches are then put against a threshold in order to determine if the images match or not. If any match is detected at all, it is considered to be similar, otherwise the images are considered to be different. If the two images matches, the current image is then added to the group the base image belonged to, otherwise it checks whether the group was the last one in the dictionary. If it was not, it moves onto the next group and repeats the same process, otherwise, the current image is added to a new group in the dictionary.



```
                #Match descriptor -scikit
                matches = match_descriptors(descriptors1, descriptors2, max_distance=0.275, cross_check=True)

                # If the image is similar to the base image, add this image to that dictionary
                if len(matches) > 0:
                    groupedPhotos[i][len(groupedPhotos[i].keys())] = photo
                    print groupedPhotos
                    break

                # If image is not similar to the base image, add a new dictionary with this image as its first element
                else:
                    # loop through the rest of the image dictionaries
                    if groupCounter < len(groupedPhotos.keys()) - 1:
                        groupCounter += 1
                    else:
                        groupedPhotos[i+1] = {}
                        groupedPhotos[i+1][0] = photo
                        break
```

Figure 4.7 Continuation of the image comparison implementation

The average execution time of the ORB comparison is 2.0618 seconds with ten comparisons of images with different sizes. The larger the image, the longer this comparison will take. Because of this, comparing each image with every single one of the images in one group can cause a very long execution time, especially if a group consisted of several images. Therefore, I have decided to have one base image representing each group, which will be the first image in that group, that the current image will be compared to. However, by implementing it this way, it increases the inaccuracy of the end result with the loss of the ability to double check whether the image do belong in that group by comparing it with the other images listed in that group, which may result in a few outliers in the grouped photos in the end. But, the length of the execution time poses a higher risk of timing out the client request, therefore I have decided to implement it this way.

Image Analysis

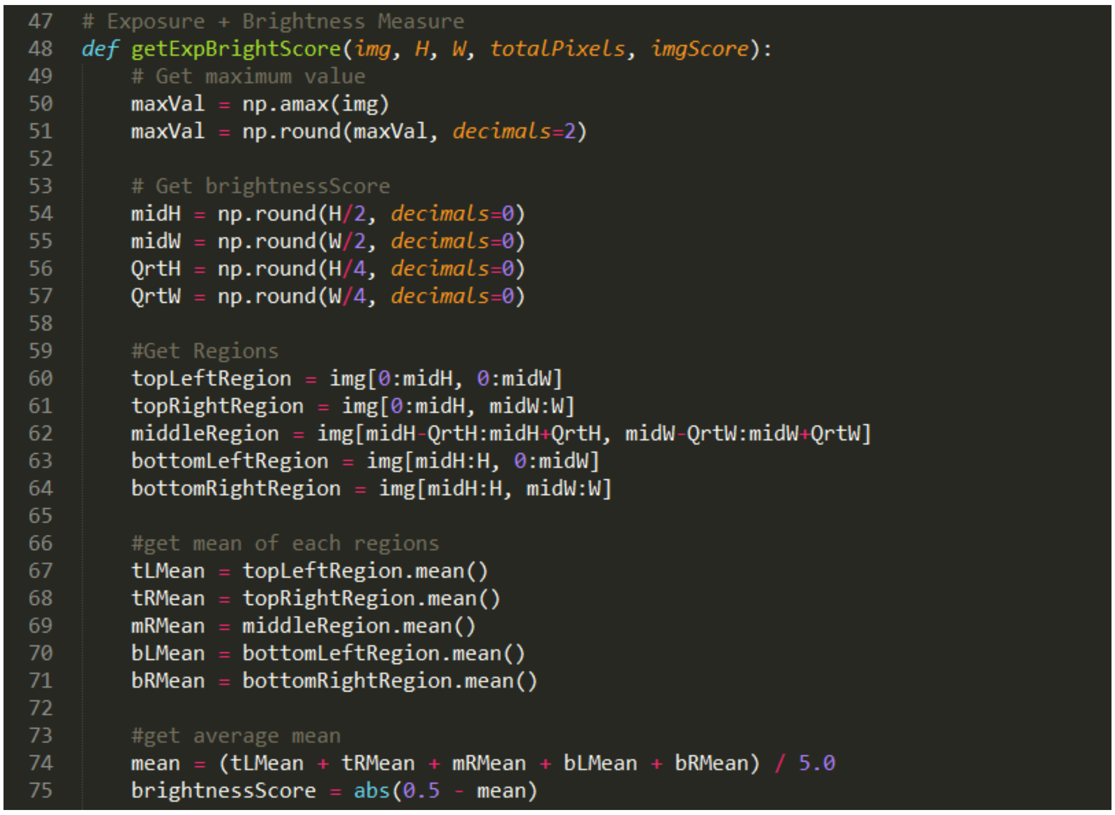
Image analysis of the different image properties is another core functional requirement of my system. This can be separated into four different components: the measure for exposure and brightness, contrast, focus and noise.

- **Exposure and Brightness Measure:** As discussed in chapter 2, exposure and brightness can be measured by analysing the luminance histogram of the image. This function takes a grayscale image (*img*), the height (*H*), width (*W*) and total pixels (*totalPixels*) of the image as well as an object called image score (*imgScore*) as parameters. A grayscale image is used for this measure because it contains the luminance value for each pixel which is required to measure brightness and exposure.

  First, the brightness score of an image is calculated. This is done by calculating the geometric mean of the image to be analysed. Before doing this, the image must first be divided into five different sections: the top left, top right, bottom left, bottom right and the centre regions of the image.

  Each of these regions can be found by first getting the middle and quarter values of the height, *midH* and *QrtH*, and the width, *midW*, *QrtW*, of the image. The extraction of these regions from the original image can be seen in figure 4.8 starting from line 60 to 64.

  The brightness measure is calculated by getting the average of each of the regions followed by calculating the overall average of the calculated region mean. This can be seen in figure 4.8, from line 67 to 74. Brightness can be assessed based on having the optimal value to be 0.5, the middle value of the normalized image luminance, ranging from 0 to 1. This value is chosen as the optimal value for image brightness as brightness that is too high or low can cause loss of details, thus a balanced tonal distribution is often sought after.

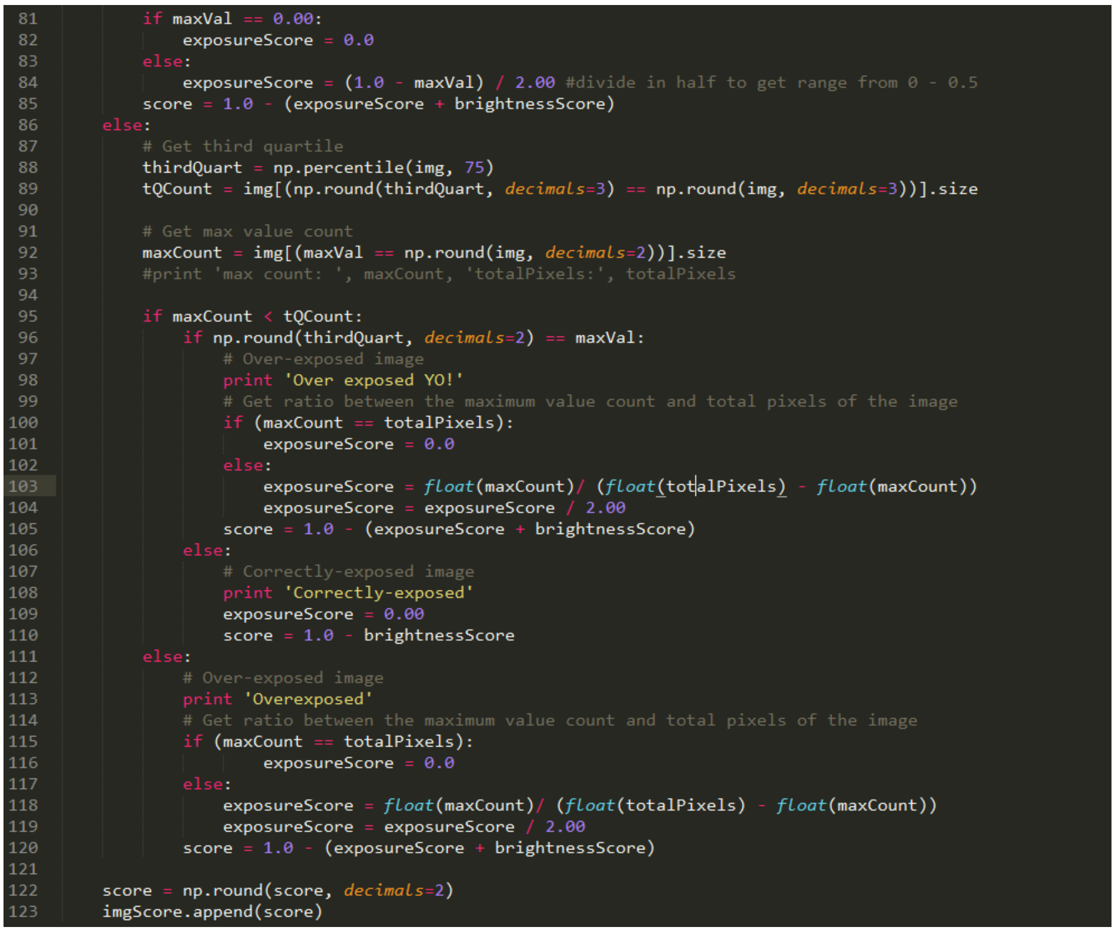


```python
# Exposure + Brightness Measure
def getExpBrightScore(img, H, W, totalPixels, imgScore):
    # Get maximum value
    maxVal = np.amax(img)
    maxVal = np.round(maxVal, decimals=2)

    # Get brightnessScore
    midH = np.round(H/2, decimals=0)
    midW = np.round(W/2, decimals=0)
    QrtH = np.round(H/4, decimals=0)
    QrtW = np.round(W/4, decimals=0)

    #Get Regions
    topLeftRegion = img[0:midH, 0:midW]
    topRightRegion = img[0:midH, midW:W]
    middleRegion = img[midH-QrtH:midH+QrtH, midW-QrtW:midW+QrtW]
    bottomLeftRegion = img[midH:H, 0:midW]
    bottomRightRegion = img[midH:H, midW:W]

    #get mean of each regions
    tLMean = topLeftRegion.mean()
    tRMean = topRightRegion.mean()
    mRMean = middleRegion.mean()
    bLMean = bottomLeftRegion.mean()
    bRMean = bottomRightRegion.mean()

    #get average mean
    mean = (tLMean + tRMean + mRMean + bLMean + bRMean) / 5.0
    brightnessScore = abs(0.5 - mean)
```

Figure 4.8: Extract from *upload.py* showing the implementation for the brightness measure.

The exposure measure is implemented as shown in figure 4.9. An underexposed image can be defined as an image with a tonal distribution that does not reach the histogram’s maximum value, which in this case is 1. Therefore, if the image’s maximum value does not equals to 1, the image is considered to be under exposed. The exposure measure for this can be calculated by taking the maximum value of the image from 1. On the other hand, if the maximum value of the image is 1, it can be considered as either a correctly exposed image or an over exposed image. In order to differentiate the two, we must figure out the third quartile quantity value of the image. This percentile is chosen as it is situated where the start of the highlight tones should be located in a histogram with a balanced tonal distribution. For an image to be over exposed, a clipping must appear at the right hand side of the histogram, which in this case would be a high quantity of luminance of 1. But how shall the luminance of 1 be thresholded? How much does the quantity of it should be in order for it to be considered over exposed? Sometimes, clipping on the right hand side of the histogram can be tolerated especially for images that are considered to be bright, or has a white background as shown in figure 4.10. To differentiate between a correctly exposed and over exposed image, the pixel quantity of the third quartile and the pixel quantity of luminance 1 is compared. If the quantity of the third quartile is larger than the quantity of luminance 1, then it is considered to be correctly exposed, unless the value of the third quartile is 1 which is then considered to be over exposed. On the other hand, if the quantity of luminance 1 is greater than the quantity of the third quartile, it is then considered to be over exposed. Correctly exposed image receives an exposure score of 0, whereas an overexposed image’s score is the ratio between the quantity of the clipped pixels and the rest of the pixels in the image.

The brightness and exposure score is then combined together and taken away from 1. The higher the score of an image, the closer it is to optimal brightness and exposure.



```
            if maxVal == 0.00:
                exposureScore = 0.0
            else:
                exposureScore = (1.0 - maxVal) / 2.00 #divide in half to get range from 0 - 0.5
            score = 1.0 - (exposureScore + brightnessScore)
        else:
            # Get third quartile
            thirdQuart = np.percentile(img, 75)
            tQCount = img[(np.round(thirdQuart, decimals=3) == np.round(img, decimals=3))].size

            # Get max value count
            maxCount = img[(maxVal == np.round(img, decimals=2))].size
            #print 'max count: ', maxCount, 'totalPixels:', totalPixels

            if maxCount < tQCount:
                if np.round(thirdQuart, decimals=2) == maxVal:
                    # Over-exposed image
                    print 'Over exposed YO!'
                    # Get ratio between the maximum value count and total pixels of the image
                    if (maxCount == totalPixels):
                        exposureScore = 0.0
                    else:
                        exposureScore = float(maxCount)/ (float(totalPixels) - float(maxCount))
                        exposureScore = exposureScore / 2.00
                    score = 1.0 - (exposureScore + brightnessScore)
                else:
                    # Correctly-exposed image
                    print 'Correctly-exposed'
                    exposureScore = 0.00
                    score = 1.0 - brightnessScore
            else:
                # Over-exposed image
                print 'Overexposed'
                # Get ratio between the maximum value count and total pixels of the image
                if (maxCount == totalPixels):
                        exposureScore = 0.0
                else:
                    exposureScore = float(maxCount)/ (float(totalPixels) - float(maxCount))
                    exposureScore = exposureScore / 2.00
                score = 1.0 - (exposureScore + brightnessScore)

        score = np.round(score, decimals=2)
        imgScore.append(score)
```

Figure 4.9: Continuation of the brightness and exposure measure function, focusing on the exposure measure of an image.

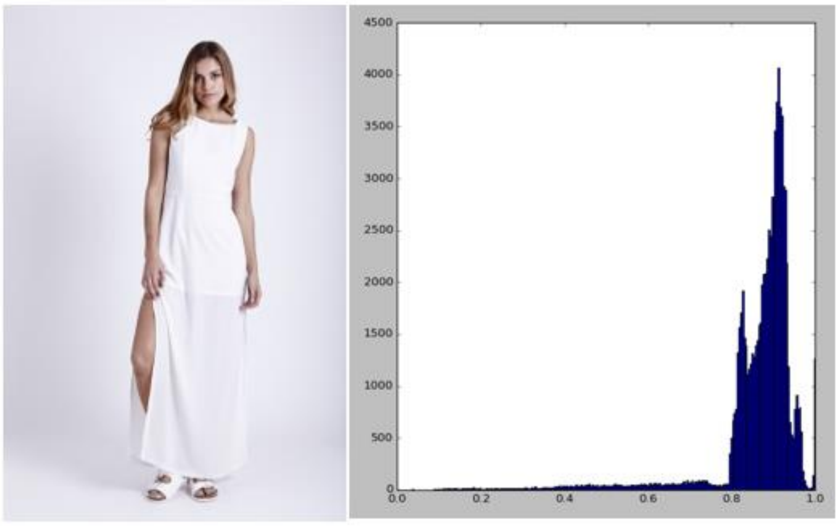


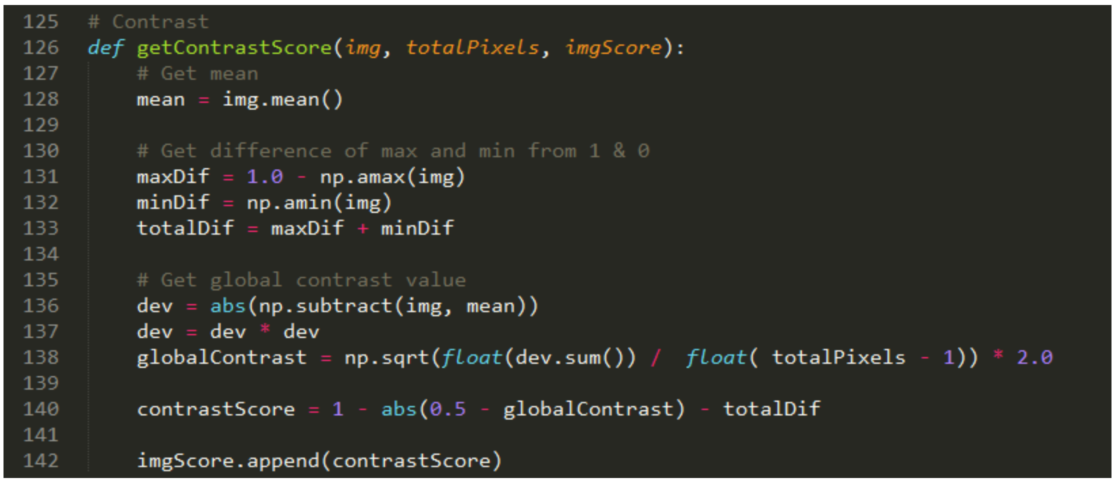
Figure 4.10: A bright correctly exposed image with a lot of white tones. Notice the clipping on the right hand side of the histogram.

- **Contrast Measure:** As mentioned in chapter 2, a global contrast measure of an image can acquired by measuring the root mean square (RMS) contrast. This is measure is defined as shown in the figure below, where x_i is a normalized grey level value with the range of 0 to 1 and where $\bar{x}$ is the mean luminance of an image. The score measure is then taken away from 0.5, which is the optimal value for contrast. Contrast that is too high can lead to a loss of details and can make a picture seem unrealistic whereas a contrast that is too low can make an image seem very dull.

$$\mathrm{rms} = \left[\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2\right]^{1/2}$$

Figure 4.11 RMS contrast measure

Contrast can also be defined by an image's tonal distribution. An image that has a limited tonal distribution, its luminance values not spread across a histogram, can be considered to have a low contrast. On the other hand, an image that has a tonal distribution that is spread out across the histogram, reaching the maximum and minimum value, are considered to have a good contrast. Therefore, I have also done another measure according to this concept. First, the image's maximum and minimum luminance value is figured out then it is taken away from the true maximum and minimum luminance value which is 1 and 0. The total difference can then be calculated. The contrast of the image can then be scored by adding the two measures together and take it away from 1.



```
# Contrast
def getContrastScore(img, totalPixels, imgScore):
    # Get mean
    mean = img.mean()

    # Get difference of max and min from 1 & 0
    maxDif = 1.0 - np.amax(img)
    minDif = np.amin(img)
    totalDif = maxDif + minDif

    # Get global contrast value
    dev = abs(np.subtract(img, mean))
    dev = dev * dev
    globalContrast = np.sqrt(float(dev.sum()) /  float( totalPixels - 1)) * 2.0

    contrastScore = 1 - abs(0.5 - globalContrast) - totalDif

    imgScore.append(contrastScore)
```

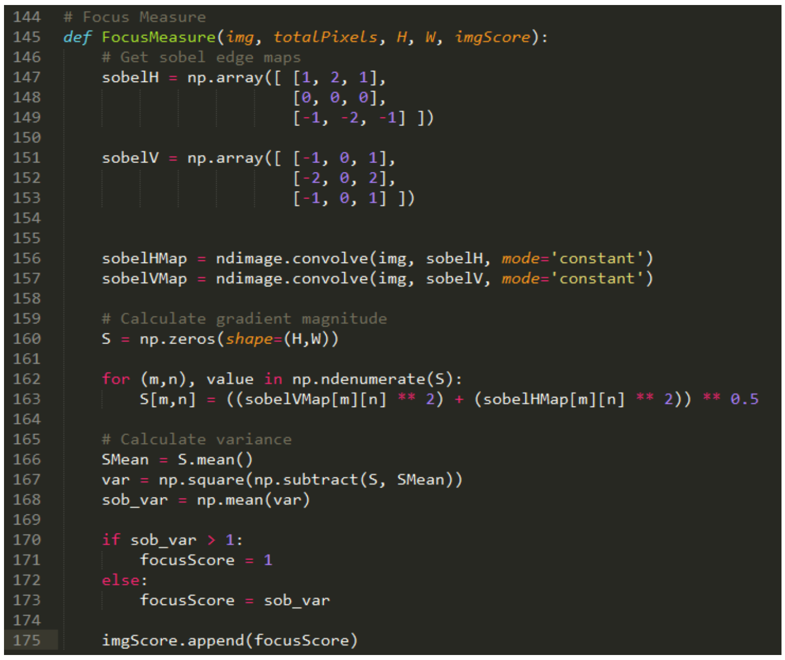
Figure 4.12: Extract from upload.py showing the implementation of the contrast measure.

- **Focus Measure:** In order to calculate the focus measure, the image must be first convolved with a vertical and horizontal Sobel edge filter which is defined in line 125 to 131 in figure 4.13. The image can be convolved using the convolve function of SciPy's ndimage module. The gradient magnitude can then be calculated according to the following equation. Where G_x and G_Y are the horizontal and vertical convolved image respectively.

$$S(m,n) = \sqrt{[G_x(m,n)]^2 + [G_y(m,n)]^2}$$

Figure 4.13 Gradient magnitude equation [21]

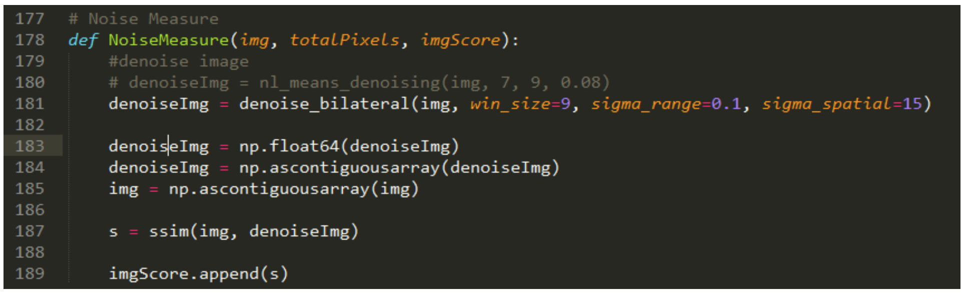
The Sobel variance can then be can then calculated by getting the mean of *S* and subtracting this from every pixel in *S.* The result of the subtraction is then squared and the mean of this can be calculated in order to get the variance. If the variance is greater than 1, indicating a sharp image, the focus measure is set to 1. Otherwise, it is set to the calculated variance value.



```python
# Focus Measure
def FocusMeasure(img, totalPixels, H, W, imgScore):
    # Get sobel edge maps
    sobelH = np.array([ [1, 2, 1],
                        [0, 0, 0],
                        [-1, -2, -1] ])

    sobelV = np.array([ [-1, 0, 1],
                        [-2, 0, 2],
                        [-1, 0, 1] ])


    sobelHMap = ndimage.convolve(img, sobelH, mode='constant')
    sobelVMap = ndimage.convolve(img, sobelV, mode='constant')

    # Calculate gradient magnitude
    S = np.zeros(shape=(H,W))

    for (m,n), value in np.ndenumerate(S):
        S[m,n] = ((sobelVMap[m][n] ** 2) + (sobelHMap[m][n] ** 2)) ** 0.5

    # Calculate variance
    SMean = S.mean()
    var = np.square(np.subtract(S, SMean))
    sob_var = np.mean(var)

    if sob_var > 1:
        focusScore = 1
    else:
        focusScore = sob_var

    imgScore.append(focusScore)
```

Figure 4.14: Extract from upload.py showing the implementation of the focus measure method.

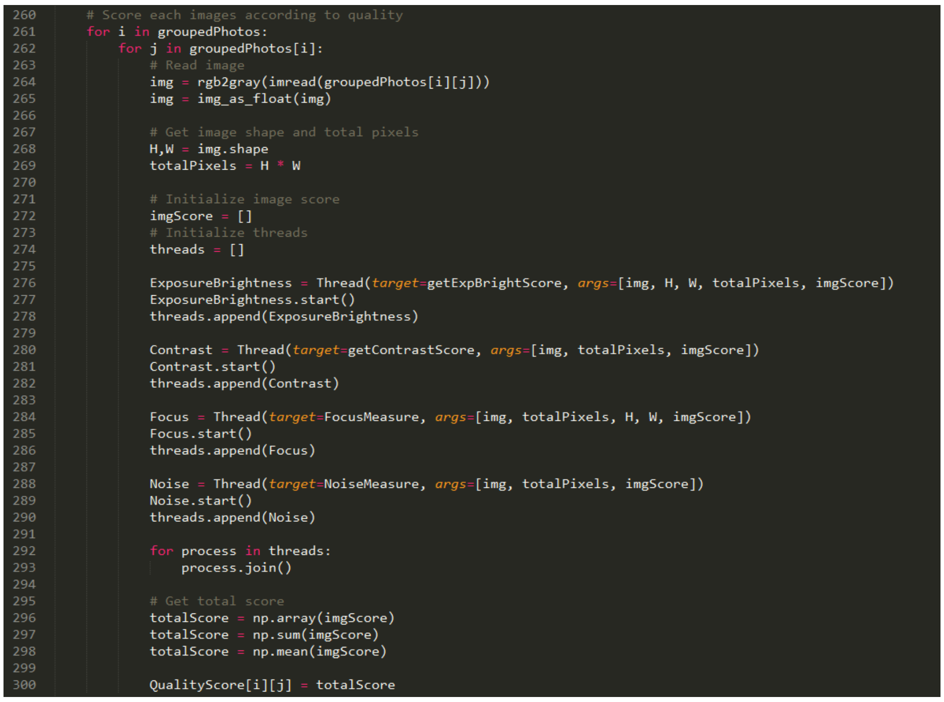
- **Noise Measure:** Noise is hard to measure especially without the original de-noised image. Thus, this can only be estimated. For this measure, I have used the bilateral filtering de-noising method which has already been implemented by scikit-image. With this, I can acquire an estimated de-noised image of the image to be analysed. The next step is to compare this acquired image with the image to be analysed. Two comparative quality measure can be used here, the mean squared error and the structural similarity index measure (SSIM) which are both mentioned in chapter 2 as part of the image comparison section. For this measure, I have decided to use the structural similarity index measure as this gives a more accurate measure than the mean squared error. SSIM has already been implemented by scikit-image and can therefore be called to compare the two images.



```python
# Noise Measure
def NoiseMeasure(img, totalPixels, imgScore):
    #denoise image
    # denoiseImg = nl_means_denoising(img, 7, 9, 0.08)
    denoiseImg = denoise_bilateral(img, win_size=9, sigma_range=0.1, sigma_spatial=15)

    denoiseImg = np.float64(denoiseImg)
    denoiseImg = np.ascontiguousarray(denoiseImg)
    img = np.ascontiguousarray(img)

    s = ssim(img, denoiseImg)

    imgScore.append(s)
```

Figure 4.15: Extract from upload.py showing the implementation of the noise measure.

All these functions are deployed as threads for each image so they can run in parallel to each other in order to decrease the execution time. The total score received from each of the functions are then averaged which will be the final quality score of the image.

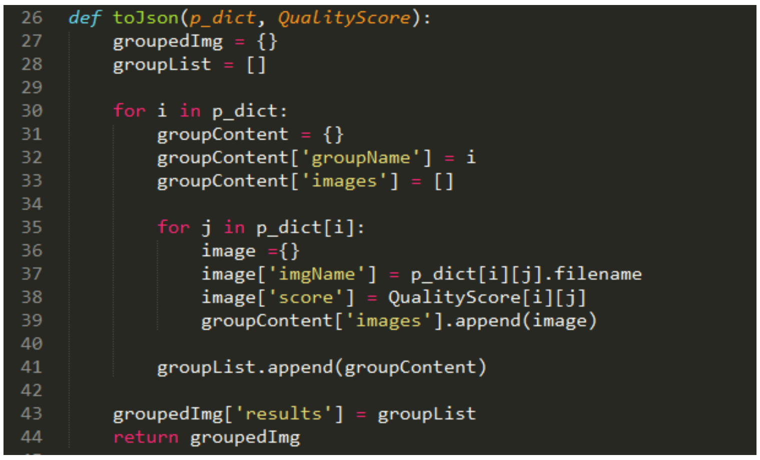


```python
    # Score each images according to quality
    for i in groupedPhotos:
        for j in groupedPhotos[i]:
            # Read image
            img = rgb2gray(imread(groupedPhotos[i][j]))
            img = img_as_float(img)

            # Get image shape and total pixels
            H,W = img.shape
            totalPixels = H * W

            # Initialize image score
            imgScore = []
            # Initialize threads
            threads = []

            ExposureBrightness = Thread(target=getExpBrightScore, args=[img, H, W, totalPixels, imgScore])
            ExposureBrightness.start()
            threads.append(ExposureBrightness)

            Contrast = Thread(target=getContrastScore, args=[img, totalPixels, imgScore])
            Contrast.start()
            threads.append(Contrast)

            Focus = Thread(target=FocusMeasure, args=[img, totalPixels, H, W, imgScore])
            Focus.start()
            threads.append(Focus)

            Noise = Thread(target=NoiseMeasure, args=[img, totalPixels, imgScore])
            Noise.start()
            threads.append(Noise)

            for process in threads:
                process.join()

            # Get total score
            totalScore = np.array(imgScore)
            totalScore = np.sum(imgScore)
            totalScore = np.mean(imgScore)

            QualityScore[i][j] = totalScore
```

Figure 4.16: Extract from upload.py which shows how all of the functions are started as threads to run in parallel to each other.

Return results to client

Before the result of the analysis is returned as a response, first, the result dictionary has to be converted to a JSON string format that the client will be able to read. This is achieved by calling the *toJson()* function which takes the result dictionary from the comparison along with the score dictionary from the image quality assessment. After the conversion, the formatted dictionary can then be converted to a JSON string using Python’s jsonify.



```python
def toJson(p_dict, QualityScore):
    groupedImg = {}
    groupList = []

    for i in p_dict:
        groupContent = {}
        groupContent['groupName'] = i
        groupContent['images'] = []

        for j in p_dict[i]:
            image ={}
            image['imgName'] = p_dict[i][j].filename
            image['score'] = QualityScore[i][j]
            groupContent['images'].append(image)

        groupList.append(groupContent)

    groupedImg['results'] = groupList
    return groupedImg
```

Figure 4.17: Extract from upload.py which shows how the two dictionaries from the image comparison and assessment is converted to a JSON format.

Client

The client consists of a Cordova application which is deployed by Ionic Framework. It consists of two main pages, the home and the results page with an index page as a holder for both of these views. This application can be deployed on an android phone.

Controller

The controller contains the image selection, upload and share function. This application has two controllers, one for each of the pages.

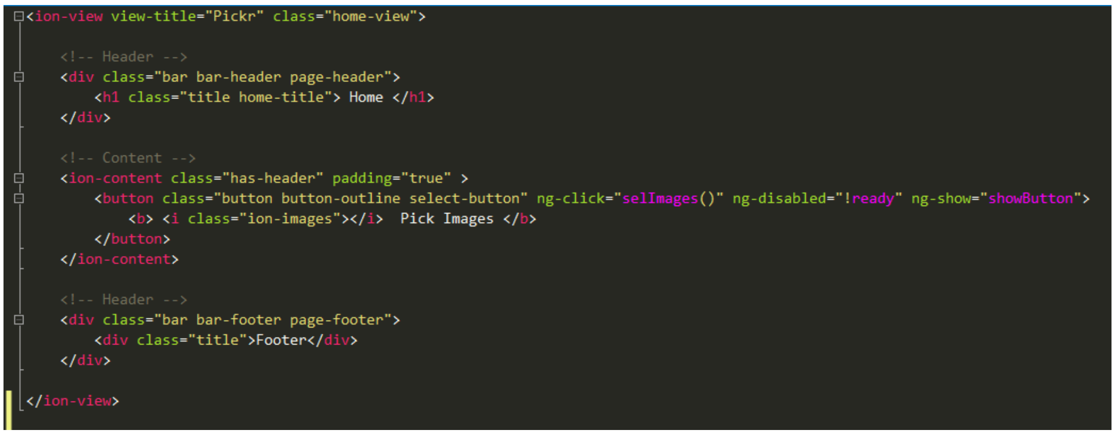
- **HomeCtrl:** This is the controller for the home page. It contains the image selection function as well as the upload function required for the application. The image selection is implemented by Cordova's image picker plugin. This plugin allows the user to select multiple images at once. It takes a temporary screenshot of the image selected and stores it temporarily in the application's cache folder. Because of this, the image's original meta-data is lost. There were other image selection methods but all of them were only allowing an image to be selected at once. Due to this, I believe that this was best suited for my requirement of uploading multiple images at the same time.

  This controller also contains the implementation of the upload functionality. Cordova's file transfer plugin was one way of doing this, however it only allowed for one photo to be uploaded at a time. Therefore, I've decided to use an HTTP POST method instead. First, the image file has to be read. This is achieved by using Cordova's readAsArrayBuffer plugin. The image read is then appended to the form data. Once all of the images have been put into the form data, it then beings its upload to the server. Once a response have been received, the page is directed to the results page for the display of the results.

- **ResultCtrl:** The response object from the previous page is returned by a factory service. This is then bind to a local scope. The share function is implemented using Cordova's social sharing plugin. First, the image source must be found as the response object from the server only contains the image's filename. This is achieved by adding the path of the application's cache directory, which was the place the images was saved to after image selection. The image must be then be read as a base64 image. This is achieved by using Cordova's file reader. If the file read of the image is a success, the image selected can then be shared to a social media site selected by the user.

Home page

The home page is denoted by the route **/home** and it is the first page that the user will see when they start the application. It contains a browse button which the user can click. This brings up a new window containing their images in their photo album which they can select to upload to the server.

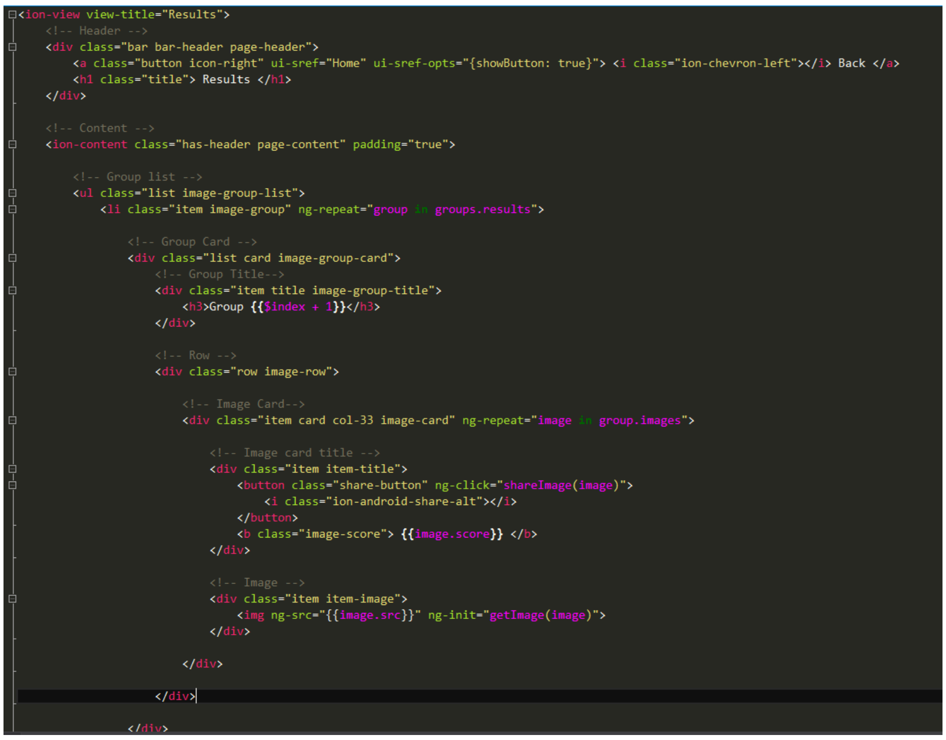


```html
<ion-view view-title="Pickr" class="home-view">

    <!-- Header -->
    <div class="bar bar-header page-header">
        <h1 class="title home-title"> Home </h1>
    </div>

    <!-- Content -->
    <ion-content class="has-header" padding="true" >
        <button class="button button-outline select-button" ng-click="selImages()" ng-disabled="!ready" ng-show="showButton">
            <b> <i class="ion-images"></i>  Pick Images </b>
        </button>
    </ion-content>

    <!-- Header -->
    <div class="bar bar-footer page-footer">
        <div class="title">Footer</div>
    </div>

</ion-view>
```

Figure 4.18: home.html

Result page

Once the client receives a response from the server, it directs to the result page denoted by the route **/results**. This page displays the results of the comparison and analysis. The similar images are grouped together in separate cards. Each of these cards contains the individual images showing the score of the image as well as a share button above. If the user would like to share an image, they can click this share button and a window will appear listing the various methods the photo can be shared such as Facebook, Twitter and Email.



```html
<ion-view view-title="Results">
    <!-- Header -->
    <div class="bar bar-header page-header">
        <a class="button icon-right" ui-sref="Home" ui-sref-opts="{showButton: true}"> <i class="ion-chevron-left"></i> Back </a>
        <h1 class="title"> Results </h1>
    </div>

    <!-- Content -->
    <ion-content class="has-header page-content" padding="true">

        <!-- Group list -->
        <ul class="list image-group-list">
            <li class="item image-group" ng-repeat="group in groups.results">

                <!-- Group Card -->
                <div class="list card image-group-card">
                    <!-- Group Title-->
                    <div class="item title image-group-title">
                        <h3>Group {{$index + 1}}</h3>
                    </div>

                    <!-- Row -->
                    <div class="row image-row">

                        <!-- Image Card-->
                        <div class="item card col-33 image-card" ng-repeat="image in group.images">

                            <!-- Image card title -->
                            <div class="item item-title">
                                <button class="share-button" ng-click="shareImage(image)">
                                    <i class="ion-android-share-alt"></i>
                                </button>
                                <b class="image-score"> {{image.score}} </b>
                            </div>

                            <!-- Image -->
                            <div class="item item-image">
                                <img ng-src="{{image.src}}" ng-init="getImage(image)">
                            </div>

                        </div>

                    </div>

                </div>
```

Figure 4.19: Extract from results.html

Chapter 5: System Validation

System validation is done to ensure that the application satisfies its intended functional requirements. This chapter focuses on the different methods of testing that has been done in order to ensure that the system executes the way it's supposed to perform. The various methods of testing which will be discussed in this chapter which includes unit testing and integration testing.

Unit Testing

Unit testing involves testing of various units, which in this case functions, within the code. “This gives the ability to verify that the functions work as expected” [36]. Unit testing is done for every product modification and increment in a sprint. Each of the server components are created as separated units which are integrated later on in development. These separated units are tested individually for validation and evaluation.

Image Comparison

A unit was created to test the image comparison function which is one of the core functional components of the system. This was tested in every increment of the function. The following table shows the results of these tests.

No.	Feature	Expectation	Result
1	Read image from file	Show image in a figure	PASS
2	Convert image to grayscale	Show grayscale image in figure	PASS
3	Compare image using ORB	Show two images side by side with plotted matches.	PASS
4	Threshold matches according to slope	More accurate comparison	FAIL
5	Define match_descriptors with different parameters	More accurate comparison	PASS
6	Threshold matches by 0	Categorize matches as a success or fail	PASS

Image Quality Assessment

Image quality assessment can be considered as four different functions: Exposure and brightness measure, contrast, noise and focus measure. Each of these measure are tested using a constant set of pictures for each different measure.

Exposure and Brightness Measure

No.	Feature	Expectation	Result	Reason
1	Extract region	Display an extracted region from the original image.	FAIL	Out of bound index
2	Extract region	Display an extracted region from the original image.	PASS	
3	Get region mean	Display the result of each region mean. Must match pre-calculated measure.	PASS	
4	Get global mean	Display the global mean. Must match pre-calculated measure	PASS	

5	Get max and min of luminance value	Display the maximum and minimum luminance value.	PASS	
6	Distinguish overexposed and underexposed image	Display image and print out if image is over or under exposed	FAIL	Wrong threshold
7	Distinguish overexposed and underexposed image (changed to third quartile)	Display image and print out if image is over or under exposed	PASS	
8	Distinguish between over and correctly exposed image	Display image and print out if image is over or under exposed	PASS	
9	Combine two measures	Display result in range of 0 - 1	FAIL	Wrong calculation
10.	Combine two measures	Display result in range of 0 - 1	PASS	

Contrast Measure

No.	Feature	Expectation	Result	Reason
1	Get image mean	Print image mean	PASS	
2	Get total difference of max and min from 1 and 0	Display result which must match pre-calculated answer.	PASS	
3	Get global contrast value using RMS	Display the global contrast. Must be accurate in regards to image tested	FAIL	Unmatched datatype
4	Get global contrast value using RMS	Display the global contrast. Must be accurate in regards to image tested	FAIL	Wrong calculation
5	Get global contrast value using RMS	Display the global contrast. Must be accurate in regards to image tested	PASS	
6	Combined measure	Display contrast score. Must be accurate in regards to image tested	PASS	

Focus Measure

No.	Feature	Expectation	Result	Reason

1	Convolve image using Sobel filters	Display convolved images.	FAIL	Wrong convolution method
2	Convolve image using Sobel filters	Display convolved images.	PASS	
3	Calculate gradient magnitude	Display result for gradient magnitude	FAIL	Original image has been changed in an attempt to copy the variable to another.
4	Calculate gradient magnitude	Display result for gradient magnitude	PASS	
5	Calculate Sobel variance	Display Sobel variance	PASS	
6	Threshold Sobel Variance	Display normalized focus score	PASS	

Noise Measure

No.	Feature	Expectation	Result	Reason
1	De-noise Image using bilateral filter	Display de-noised image	PASS	
2	Compare image using SSIM	Display the SSIM value	FAIL	Warning of non-contiguous array
3	Compare image using SSIM	Display the SSIM value	PASS	

Integration Testing

System is not just made up of one single components but is made up of multiple components. “Integration testing makes sure that, all of these multiple components are able to communicate and work together as planned” [33].

Flask Application

No.	Feature	Expectation	Result	Reason
1	Basic Flask Application	Return a message upon the access of http://localhost:5000/test	PASS	
2	Basic file upload using a web browser	Send the uploaded image back as a response	FAIL	Wrong file request extraction method
3	Basic file upload using a web browser	Send the uploaded image back as a response	PASS	

4	Integration of image comparison	Compare two images uploaded by client and display result in command line	PASS	
5	Return a JSON response	From the image comparison results, format the result to a JSON string and send back to server	FAIL	Wrong result dictionary structure
6	Return a JSON response	From the image comparison results, format the result to a JSON string and send back to server	PASS	
7	Integration of Exposure and Brightness Measure	Display result in command line	PASS	
8	Integration of Contrast Measure	Display result in command line	PASS	
9	Integration of Noise Measure	Display result in command line	PASS	
10	Multithread measure calculation	Multithread each measure calculation for every image.	FAIL	Wrong method.
11	Multithread measure calculation	Multithread each measure calculation for every image.	PASS	
12	Score Image	Score added to the response	FAIL	Attempt to copy the result dictionary failed.
13	Score Image	Score added to the response	PASS	

Chapter 6: Evaluation

This chapter focuses on the challenges encountered during the development of the system as well as the strengths and weaknesses of the developed system.

Challenges and Solutions

There are many challenges and obstacles that I faced during the development of this project and the risks that it put on the project's overall completion. The following discussion outlines these challenges and the solutions created to solve the problems.

The first challenge encountered during the duration of this project development is to research and implement a good image comparison algorithm. I have implemented and tested three different algorithms which were MSE, SSIM and ORB as mentioned in chapter 3. The algorithm that was picked for this component was ORB. ORB was a good alternative to SIFT and SURF algorithms, however, I found that it can be lacking accuracy depending on the parameter max_distance. I have tried to make this adaptive to the image in several ways but it was not working. Instead, I have tested different measures for this parameters with different images and came to a conclusion that the value it is now was the optimal value according to my tests. Even with this, there are several outliers that can be seen grouped with images that are not similar from it.

The second challenge was a good assessment measure for different image properties that can affect image quality. For exposure, there was no proper and simple measure for this image property and the concept only existed in theory. By studying the histogram, and testing out different methods of calculating a measure for this property, I found that the best way to calculate this is by analysing the characteristics of its histogram. Each of these measures can also be very slow in execution, but with the use of multithreading, this enables the application to improve its execution time by running the functions in parallel.

The last challenge of this project was to implement the image selection and upload to the server. The best way to implement the image selection is to use image picker, a Cordova plugin as it allows to select multiple image at one time. However, this removes the meta-data of the image as it only takes a temporary screenshot of it upon selection, removing the ability to delete pictures as a functionality.

Strengths

There will always be strengths and weaknesses in most, or all of software products to be identified during evaluation. The project may not have been as accurate and as efficient as first desired and the comparison may not always match completely, but it still has produced decent success rates. The strength of this project lies on the image quality assessment. The tests performed has produced a higher success rate than the image comparison, often matching human’s assessment of image quality. The image assessment is also made more efficient due to multithreading. It also provides a share functionality in the application.

Weaknesses

Since the start of the development of the project, there has always been risks on the accuracy and efficiency of the application. One of the project’s weaknesses lies on the image comparison. It may be accurate in a few samples, but it can also have a few outliers resulting from its brute force matching of its descriptors. Because of this, it often leads to an inaccurate matches of the images. It can also be quite slow at times, depending on the image size and quantity to be compared. This can often lead to timeout issues with the client request. The Cordova application also has limited functionalities which only includes the upload and selection of the images as well as a share functionality.

References

[1] Risto Sarvas, David M. Frohlich (2011) *From Snapshots to Social Media - The Changing Picture of Domestic Photography*, Springer.

[2] Castella, T. (2012). *Five ways the digital camera changed us,* Available at:*http://www.bbc.com/news/magazine-16483509* (Accessed: 18 November 2015).

[3] Ingledew, J. (2013) *Photography*, 2nd edition, London: Laurence King Publishing Ltd.

[4] S. Annadurai, R. Shanmugalakshmi (n.d.) *Fundamentals of Digital Image Processing*, Delhi: Pearson.

[5] Stephen Laskevitch (2014) *Photoshop CC and Lightroom*, : Rocky Nook.

[6] Dondero, R., Wayne, K., Sedgewick, R. (2015) *Introduction to Programming in Python: An Interdisciplinary Approach*, United States of America: Addison-Wesley Professional.

[7] Sarfraz, M. (2013) *Intelligent Computer Vision and Image Processing*, United States of America: IGI Global

[8] UW CSE vision faculty () *Features and Image Matching,* Available at:*https://courses.cs.washington.edu/courses/cse455/09wi/Lects/lect6.pdf* (Accessed: 4th February 2016).

[9] Wang, Y. (2012) *Advances in Abstract Intelligence and Soft Computing*, United States of America: IGI Global.

[10] Thakkar, N., Kapur, S., Zhang, V. (2015) *Mastering Opencv Android Application Programming*, United Kingdom: Packt Publishing.

[11] David Forsyth, Philip Torr, Andrew Zisserman (2008) *Computer Vision - ECCV 2008: 10th European Conference on Computer Vision, Marseille, France, October 12-18, 2008 : Proceedings*, : Springer Science & Business Media.

[12] Ken Rockewell *Lighting,* Available at: *http://www.kenrockwell.com/tech/lighting.htm*(Accessed: March 05 2016).

[13] Neil Creek *Photography 101.8 – The Light Meter,* Available at: *http://digital-photography-school.com/photography-1018-meter/* (Accessed: March 06 2016).

[14] *Exposure and histogram,* Available at: *http://www.juzaphoto.com/article.php?l=en&article=2* (Accessed: May 02, 2016).

[15] Andrew S. Gibson (2014) *Exposure and Understanding the Histogram*, 2nd edn., Peachpit Press.

[16] Ken Rockwell *How to Use Histograms,* Available at:*http://www.kenrockwell.com/tech/histograms.htm* (Accessed: March 27 2016).

[17] Henry Chu *Luminance of images,* Available at:*http://www.cacs.louisiana.edu/~cice/lal/* (Accessed: March 27 2016).

[18] Tom Ashe (2014) *Color Management & Quality Output*, Focal Press.

[19] Elizabeth Allen, Sophie Triantaphillidou (2012) *The Manual of Photography and Digital Imaging*, 10th edn., Focal Press.

[20] Information Association (2013) *Image Processing*, : IGI Global.

[21] J. L. Pech-Pacheco, G. Cristobal, J. Chamorro-Martnez, J. Fernandez-Valdivia (n.d.) *Diatom autofocusing in brighteld microscopy: a comparative study*, : .

[22] SEAN MCHUGH *DIGITAL CAMERA IMAGE NOISE - PART 1,* Available at:*http://www.cambridgeincolour.com/tutorials/image-noise.htm* (Accessed: October 23 2015).

[23] NASIM MANSUROV (2009) *Understanding ISO – A Beginner's Guide,* Available at:*https://photographylife.com/what-is-iso-in-photography* (Accessed: April 01 2016).

[24] Antoni Buades, Bartomeu Coll, Jean-Michel Morel. *A review of image denoising algorithms, with a new one. SIAM Journal on Multiscale Modeling and Simulation: A SIAM Interdisciplinary Journal*, 2005, 4 (2), pp.490-530.

[25] Ce Liu, Richard Szeliski, Sing Bing Kang, C. Lawrence Zitnick, William T. Freeman (n.d.) *Automatic Estimation and Removal of Noise from a Single Image*,

[26] Available at: *http://scikit-image.org/docs/dev/api/skimage.restoration.html#skimage.restoration.denoise_bilateral*(Accessed: February 26 2016).

[27] Evans, B. J., Flanagan, D. (2014) *Java in a Nutshell*, 6th edition, United States of America: O'Reilly Media, Inc.

[28] Romano, F. (2015) *Learning Python*, United Kingdom: Packt Publishing.

[29] Wang, Z., Bovik, A. C. (2009) 'Mean Squared Error: Love It or Leave It?', *IEEE SIGNAL PROCESSING MAGAZINE*, pp. 94.

[30] Dawson-Howe, K (2014). *A Practical Introduction to Computer Vision with OpenCV*, United Kingdom: John Wiley & Sons.

[31] Rublee, E. Rabaud, V., Konolige, K., Bradski, G. () *ORB: an efficient alternative to SIFT or SURF,* Available at:*https://www.willowgarage.com/sites/default/files/orb_final.pdf* (Accessed: 2 December 2015).

[32] Singh, A. (2015) *Achieving Enterprise Agility through Innovative Software Development*, United States of America: IGI Global.

[33] Linz, T. (2014) *Testing in Scrum*, CA: Rocky Nook.

[34] Hanna Kieser *,* Available at: *https://experience.sap.com/basics/what-is-a-mock-up/*(Accessed: March 13 2016).

[35] Available at: *http://scikit-image.org/docs/dev/api/skimage.feature.html#orb* (Accessed: Match 14 2016).

[36] Tom McFarlin (2012) *The Beginner's Guide to Unit Testing: What Is Unit Testing?,*Available at: *http://code.tutsplus.com/articles/the-beginners-guide-to-unit-testing-what-is-unit-testing--wp-25728* (Accessed: February 4 2016).