Matthew Arnold

An investigation into selected segmentation-derived techniques for image quality assessment

Computer Science Tripos – Part II

Sidney Sussex College

May 18, 2017

Proforma

Name:	**Matthew Arnold**
College:	**Sidney Sussex College**
Project Title:	**An investigation into selected segmentation-derived techniques for image quality assessment**
Examination:	**Computer Science Tripos – Part II, July 2017**
Word Count:	**11,785**[1]
Project Originator:	Mr Matthew Arnold
Supervisor:	Mr Matthew Ireland

Original Aims of the Project

To create a GUI that sorted images by a weighted combination of hand-crafted extracted features (two high-level segmentation-derived features and two simple low-level features as **core**, with further features as extensions). Weights were to be chosen by the user. A segmentation algorithm was to be selected and implemented as a base for the high-level features, using saliency information for identifying subjects.

A numerical evaluation of my segmentation implementation was to be carried out, as well as a human evaluation. The principal aim of the user study was to compare the usefulness of the segmentation-derived features with the others.

Work Completed

All **core** success criteria from the Proposal were completed, as well as many extra goals.

After a thorough research phase, the segmentation algorithm was chosen and implemented. Eight image feature extractors (4 high-level and 4 low-level) were implemented, and feature values were combined with GUI user input using an efficient and optimised tool.

[1] This word count was computed using the TeXcount web service: `http://app.uio.no/ifi/texcount/online.php`

My segmentation implementation was measured against other algorithms using the Berkeley Segmentation Dataset, and the improvement gained from augmenting the segmentation with saliency information was quantified. A statistical, quantitative and qualitative evaluation of the tool and implemented features was also carried out, including a user study.

Special Difficulties

None.

Declaration

I, Matthew Arnold of Sidney Sussex College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed: Matthew Arnold

Date: May 18, 2017

Contents

List of Figures

List of Tables

Acknowledgements

I would like to express my gratitude to Matthew Ireland, my project supervisor, whose guidance and enthusiasm throughout was greatly appreciated.

Thanks are also due to those who gave up some of their time to participate in my user study for free.

Chapter 1

Introduction

I created and evaluated a tool that assists humans in sorting photographs using their personal subjective preferences. No objective quality metric is assumed; what one human judges as a “good” photograph may not appeal to another. The tool extracts a set of pre-defined features, and ranks the images by their weighted sum. The weights applied to each feature may be adjusted by the user. This personalises the ranking according to their own interpretation of what makes an aesthetically pleasing image.

The results showed with high statistical significance that the tool’s ordering correlated with subjective human rankings. Users preferred to sort images using my tool over the alternative manual approach, and the tool significantly sped up the task. The improvement in accuracy resulting from enabling the “high-level”, segmentation-derived features was not found to be statistically significant. Nevertheless, study participants indicated that such features were required in order to extract the essence of a “good” image.

1.1 Motivation

A tool for subjective image sorting has many use-cases:

- A **nature photographer** takes many hundreds of photographs, most of which are out-of-focus or poorly-framed. They wish to select their favourite few to submit for a competition.
- An **event photographer** needs to rapidly upload photographs to social media. Blurry or under-exposed photographs need to be quickly deselected. Such a tool could automatically select the best image as the "album cover".
- A **holiday maker** wishes to show highlights of their trip. The task of manually sorting their thousands of images is daunting; it would take too long.
- A **magazine editor** is tasked with finding an image that suits a particular article. For example, they might be looking for a bright photograph with a large subject.

Were any of these users able to generate a subjective sorting, their task would be reduced to selecting a top sub-list. Such a generation is infeasible without assistance from an automated tool: the human would require up to $\frac{1}{2}n(n-1)$ image comparisons[1] to rank n images. As this is quadratic in n, the task would take too long for a realistic dataset. Furthermore, the human will lose concentration or forget about images during the task, resulting in an inconsistent sorting.

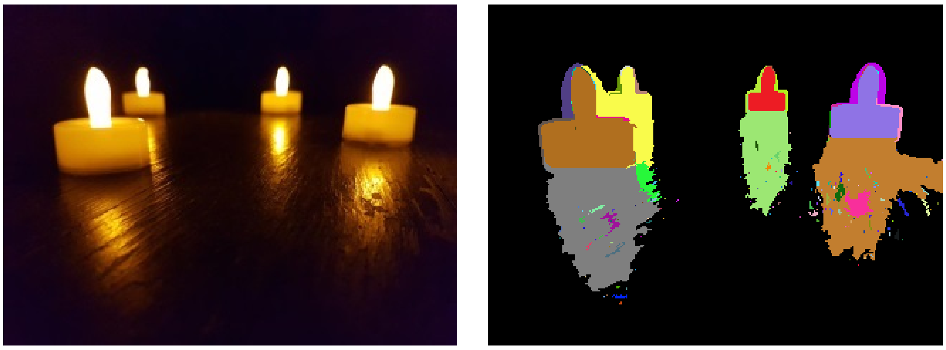
The proposed tool could determine the user's individual preferences without them having to do so much work. Once the tool has this information, it could automatically sort the images and then be re-used to sort further similar sets of images without requiring more input from the user. This project explores certain techniques for extracting such preference information efficiently enough to speed up the sorting process.

From my investigations, there exists no accessible open source tool to do this. The many websites and tools that claim to assist in photograph organisation only use the image metadata (tags such as GPS-location and time-of-capture). In contrast, this project extracts both global and compositional features from the raw pixel data to discriminate quality, then combines this with user preference input to sort the images aesthetically.

1.2 Background

This project has roots in the field of aesthetic image quality analysis, a subtopic of Computer Vision concerned with evaluating how beautiful images are to humans. The task addressed by the project is a specific one within this underlying field, with emphasis on leaving the subjectivity with the humans that create it.

[1] $n-1$ pairs from the first image, then $n-2$ from the second etc. This is a sum from 1 to $n-1$.



Original Image

Segmented Image

Figure 1.1: Example of segmenting an image.

Arguably the "holy grail" problem of Computer Vision is that of general object recognition. It is something that humans do with minimal effort and yet we have little cognitive penetrance of the task. We use past experiences to recognise or try to understand the salient objects and scenes, and use multiple contexts to form an aesthetic opinion. Therefore it makes sense to mimic this object recognition process when trying to automate the analysis of "image quality". However, since general object recognition is hypothesised to be *AI-complete*[2], it must be approximated. This project performs such an approximation by segmenting the image and uses the resulting segmentation to analyse properties of the image subjects.

Segmentation is a non-linear operation that extracts meaningful symbols corresponding to the structures and objects in an image. Figure 1.1 shows an example of this abstraction. Humans see an image rich in meaning, whereas computers have only raw data. Segmentation takes this raw data and automatically approximates the high-level symbols.

Throughout this dissertation there is a distinction between "high-level" and "low-level" image properties. High-level properties are those that incorporate top-down contextual information about the image. A feature which is derived from an image's segmentation is classed as high-level since it has access to approximations for the positions, sizes, and colours of different objects in the image. Conversely, low-level properties are relatively simple and operate on a pixel-by-pixel basis. They infer global characteristics of the image using linear transformations on the pixel data.

High-level features correspond to human analysis of the salient objects. They allow extraction of information about complicated things like scene composition. This is not to say that humans do not consider ideas similar to low-level features when judging image quality. For example, a human certainly could dislike an image because it is too bright. A combination of the two types should be needed to capture more about the aesthetic image quality.

[2] Currently can only be solved by a human or with the assistance of humans.

1.3 Related Work

Early work in aesthetic image analysis used biological properties of the human visual system to reason about how people react to certain image transformations [21]. More recently, the focus shifted to aesthetic judgement using perceptual factors, such as Aydin *et al.* in 2015 [4]. Ke *et al.* [12] and Luo & Tang [15] treat this as a binary classification problem, deciding between "professional" photographs and "snapshots".

Yeh *et al.* [22] use a semi-automated feature-based ranking approach. They proposed a set of features and learn associated weights for these features, before allowing the user to adjust these weights to rank the images based on their personal preference. Datta *et al.* [8] use a purely machine learning approach, using hand-crafted features as input to their classification algorithm.

In 2015, Lu *et al.* [14] moved away from hand-crafted features, instead using a deep convolutional neural network, which was made feasible by their 1.5 million-image dataset annotated by online rankings. It analysed both local and global stylistic and semantic image attributes. Although they achieve high accuracy, the system can only learn the preferences of an average human, ignoring subjective differences between individuals.

1.4 Context of the Work

This project uses the feature-based approach of Yeh *et al.* [22], a core paper that was the foundation for later work in the field. Their approach is sensible because they allow users to adjust feature weightings, leaving subjectivity with humans. My approach differs in that I rely on humans more by not using machine learning as a baseline for weight selection.

My project explores many of their assumptions in more detail, in particular the choice of segmentation algorithm and associated parameters. The project augments their idea with additional features such as Shape Convexity, inspired by Datta *et al.* [8].

I avoid a machine learning approach since a particular person's favourite photographs may not align with the average of the image ratings used to annotate the training dataset. These preferences will likely change over time for the same person.

1.5 Overview of the Dissertation

First, the choices made during the research phase are summarised in Chapter 2. Chapter 3 discusses the implementation of segmentation, feature extraction, feature processing, and the structure of the tool. The quantitative and qualitative results are presented in Chapter 4, and Chapter 5 gives the final conclusions.

Chapter 2

Preparation

2.1 Starting Point

I attended the Part IB Computer Graphics and Image Processing course, and took the Part II Computer Vision course this year (although this did not begin until Lent term). I had not used C# before, which was chosen as the programming language. I have not had previous experience of implementing image processing algorithms.

2.2 Summary of the Research Phase

2.2.1 Feature-based vs AI approach

I chose to use the hand-crafted feature-based approach of Yeh *et al.* [22].

The alternative (machine learning) approach uses a large set of annotated images to "learn" what makes certain photographs more aesthetically pleasing than others. The fundamental disadvantage of this approach is that such a system would only learn the preferences of the particular humans who annotated the set. In this project, I found that peoples' manual sortings of a set of images differed significantly.

Even if this group of people is large, individuals' preferences do not necessarily align with the average of the group. It is this average that is learned by the algorithm. This issue could be avoided by training the machine learning algorithm separately for every user, but that would require each user to first annotate a much larger set of images than those they were interested in sorting.

Furthermore, it is unclear how well the AI approach generalises to new kinds of image. Aesthetic quality factors may differ between different types of images, for example those with well-defined subjects, those with landscapes or those with people.

2.2.2 Segmentation Algorithm Choice

Image segmentation is required for the high-level, compositional features in order to approximate the object detection process in the human visual system. Ideally, each "object" in the image will be abstracted to its own segment, and one or more will correspond to the image "subject(s)". A careful choice of segmentation algorithm is essential, since it underlies all of the high-level features. Figure 2.1 compares a selection of possibilities.

The simplest segmentation algorithms use *Histogram Thresholding* techniques. These construct a histogram (with a bin for each possible pixel intensity) and determine a threshold intensity by which to partition the pixels. These were discounted due to the binary classification, the fact that they discard all spatial information (leading to non-contiguous segments), and because it is difficult to extend them to use more than one colour channel. Furthermore, they are extremely sensitive to noise, commonly resulting from smartphone cameras.

Colour information is more easily used by applying *clustering* algorithms to the task, for example *K-means*, which groups pixels into a *fixed* number of classes, k, based on their relative distances in some Euclidean space (Figure 2.1 uses RGB squared distance). K-means was discounted because the number of segments in each image is initially unknown and the cluster centres are randomly seeded, meaning results change between runs. Further down the pipeline, this could affect feature values and therefore image sortings between runs.

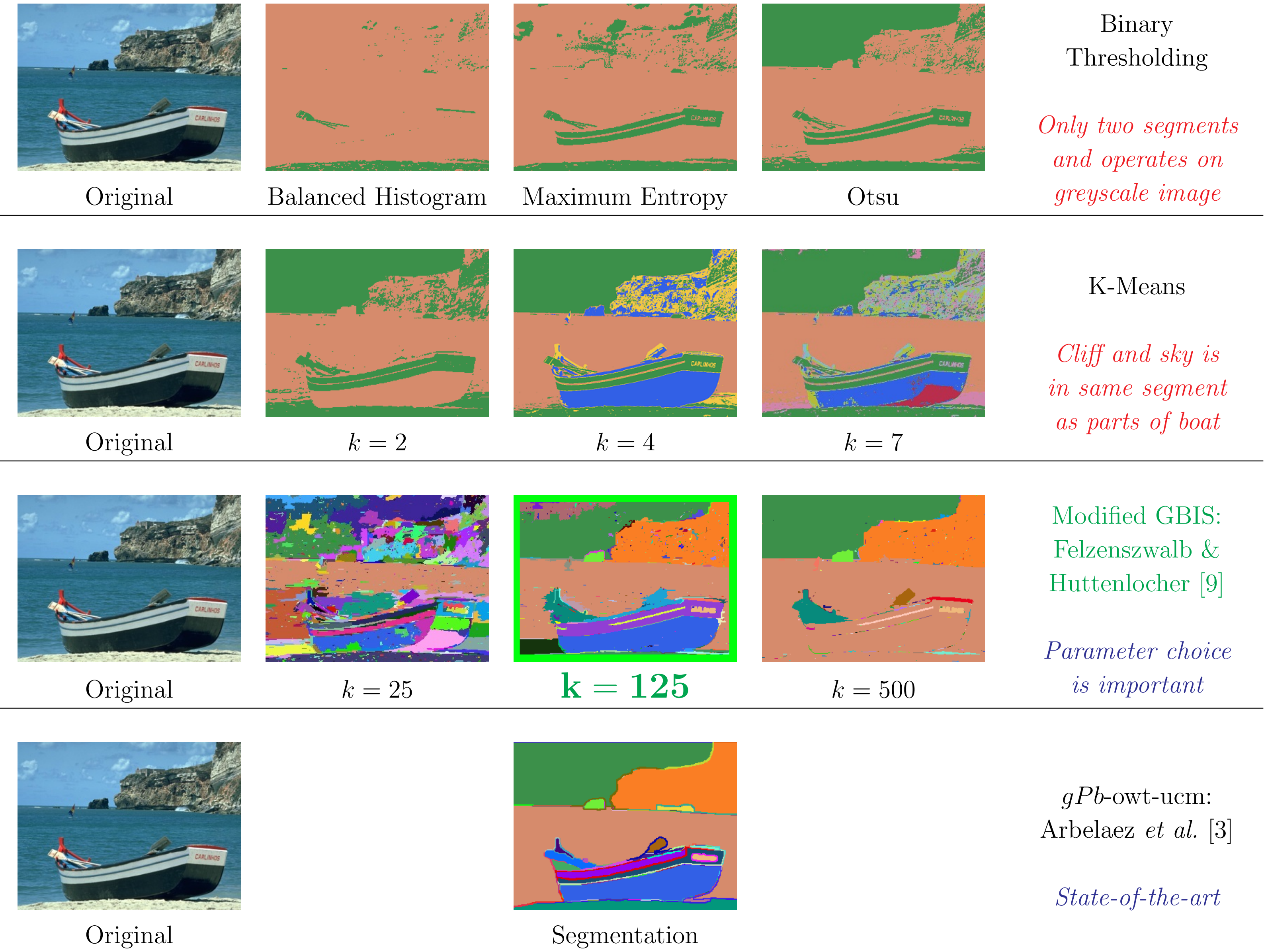
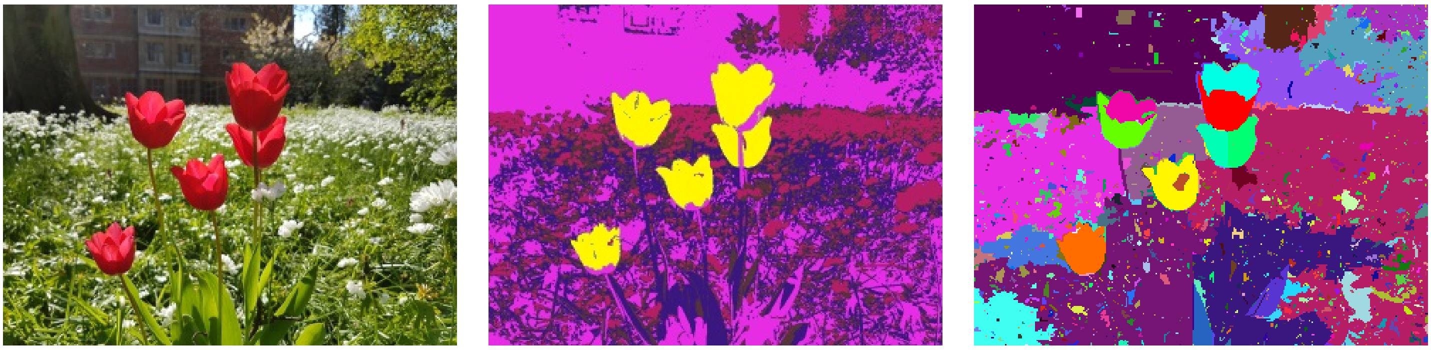

Original
Balanced Histogram
Maximum Entropy
Otsu
Binary Thresholding
Only two segments and operates on greyscale image
Original
$k = 2$
$k = 4$
$k = 7$
K-Means
Cliff and sky is in same segment as parts of boat
Original
$k = 25$
$\mathbf{k = 125}$
$k = 500$
Modified GBIS: Felzenszwalb & Huttenlocher [9]
Parameter choice is important
Original
Segmentation
gPb-owt-ucm: Arbelaez et al. [3]
State-of-the-art

Figure 2.1: Possible choices of segmentation algorithm. The chosen algorithm is highlighted in green.



Original K-Means ($k = 5$) Chosen Algorithm ($k = 125$)

Figure 2.2: K-Means assigns the flowers to the same segment. The chosen algorithm separates individual flowers.

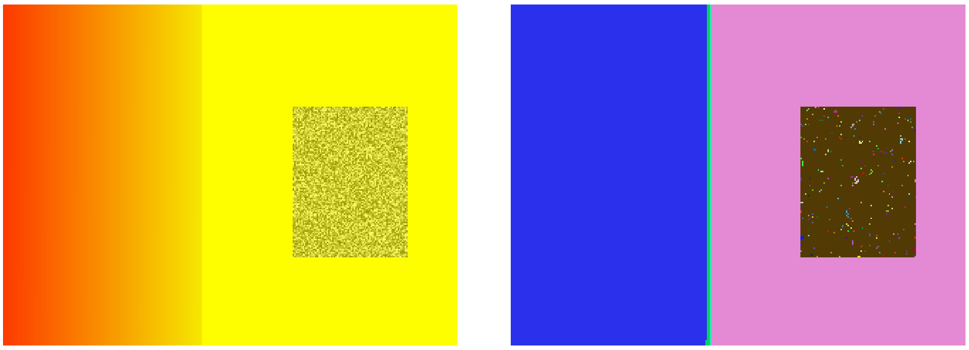
Thresholding and clustering have only a *global* image view, resulting in non-contiguous segments; Figure 2.2 gives an example. The high-level features need to reason about the positions and sizes of objects, so it is important to have contiguous segments.

Graph-based segmentation algorithms take an opposite, bottom-up approach, treating the image as a graph. There is a vertex for each pixel, and undirected edges form a mesh (called a *grid-graph*). Edge-weights correspond to pixel difference, and regions are grown from single pixels, with more pixels being added to a region if the difference at the boundary is small enough. Region-growing algorithms are less affected by *intensity inhomogeneity* than thresholding: a problem for MRI segmentation [20] which causes image objects to be split in half. It would make it difficult to reason about the sizes or positions of objects in the proposed tool. Region-growing is discounted, however, since it only has a *local* view of the data.

The best algorithms include a *global* view/criteria when deciding on a segmentation, in addition to looking at *local* spatial data. Shi & Malik [18] proposed "*Normalized Cuts*", which minimises pixel difference within segments, and simultaneously maximises difference between segments. However, this solution is *NP-complete*, and approximations to it are complicated, slow and inconsistent.

Amongst the state-of-the-art in image segmentation is gPb-owt-ucm by Arbelaez *et al.* [3]. This is a hierarchical combination of: the *Oriented Watershed Transform* for segmentation; the *Global Probability of Boundary* advanced contour detection; and the *Ultrametric Contour Map*. Although the algorithm produces excellent results, it is a complicated amalgamation of much work from the Berkeley Vision group, which I would not have had time to understand or implement myself in the planned time-scale. There is an available implementation, but it would be difficult to interface with my chosen programming language, and it runs slower without GPU parallelism.

The chosen algorithm, by Felzenszwalb & Huttenlocher [9], is an extension of Kruskal's greedy Minimum-Spanning-Tree-finding algorithm [13]. In Kruskal's, edges from the sorted edge list are considered in a non-decreasing order, and added to the MST if they do not introduce a cycle. Felzenszwalb & Huttenlocher add the adaptive *global* criterion



Original   Segmentation with chosen algorithm

Figure 2.3: These three perceptually distinct regions could not be isolated with only a *local* view. The chosen algorithm can pick out regions of high-variability, regions of no variability, and regions of constant variability.

that the edge is only added if the two segments it connects are "similar enough" compared to the variability within them, which merges those segments. Because of this criterion, Felzenszwalb & Huttenlocher proved that "although this algorithm makes greedy decisions it produces segmentations that satisfy global properties", which are illustrated in Figure 2.3. It is these global properties, combined with computational tractability, that make this my chosen algorithm. The result is not an MST, but a mutually-spanning forest of trees, each of which corresponding to a segment.

Apart from having been used in image quality analysis by Yeh *et al.* [22], the chosen algorithm has been used for many other applications, including Video Segmentation (Grundmann *et al.* [10]) and 3D Image Structure Learning (Saxena *et al.* [16]).

2.2.3 Saliency

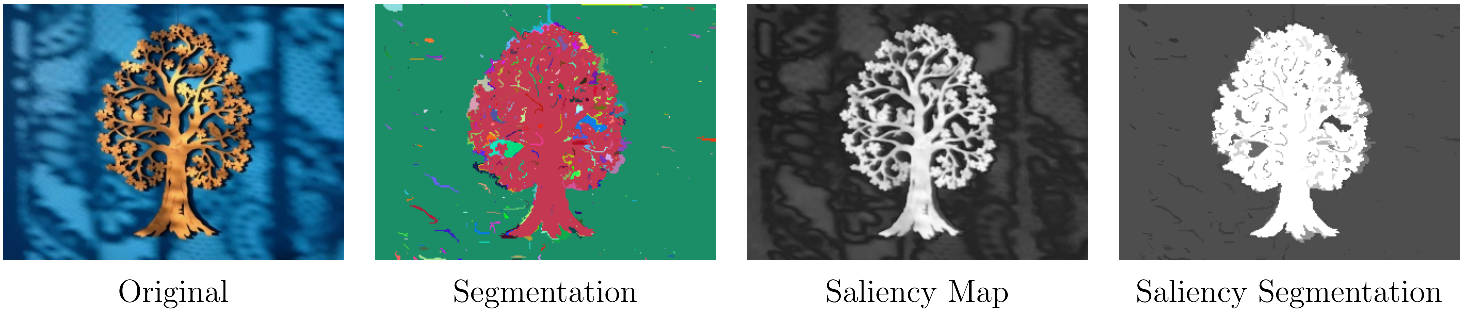
Having computed the segmentation, the subject(s) of the image need to be identified. When looking at an image, humans are immediately drawn to the regions that stand out the most. Saliency is a measure that attempts to identify these regions.

There is a vast body of research into saliency metrics for the purpose of object recognition and segmentation. AI techniques generally yield the best results, but are specific to a particular family of image. I instead opted for Achanta *et al.*'s *saliency map* [1] because it has the desired generality and is perceptually defined.

Each pixel is given a salience value according to the Euclidean distance

$$S(x, y) = ||\ I_\mu - I_\sigma(x, y)\ || \tag{2.1}$$

where I_μ is the average CIELAB colour of the image, and I_σ is the Gaussian-blurred input image. A pixel has a higher saliency if its colour is more different from the average. Since the CIELAB-space was derived using human perception experiments, this saliency map approximates the way humans identify regions that stand out to them the most.



Original Segmentation Saliency Map Saliency Segmentation

Figure 2.4: The segmentation algorithm can only partition the image, but the saliency segmentation approximates which regions correspond to image subjects.

Image subjects are found by looking at which regions from the segmentation have high average salience, as exemplified in Figure 2.4.

2.2.4 Chosen Features

Figure 2.5 gives examples of high and low feature-valued images for each of the chosen features.

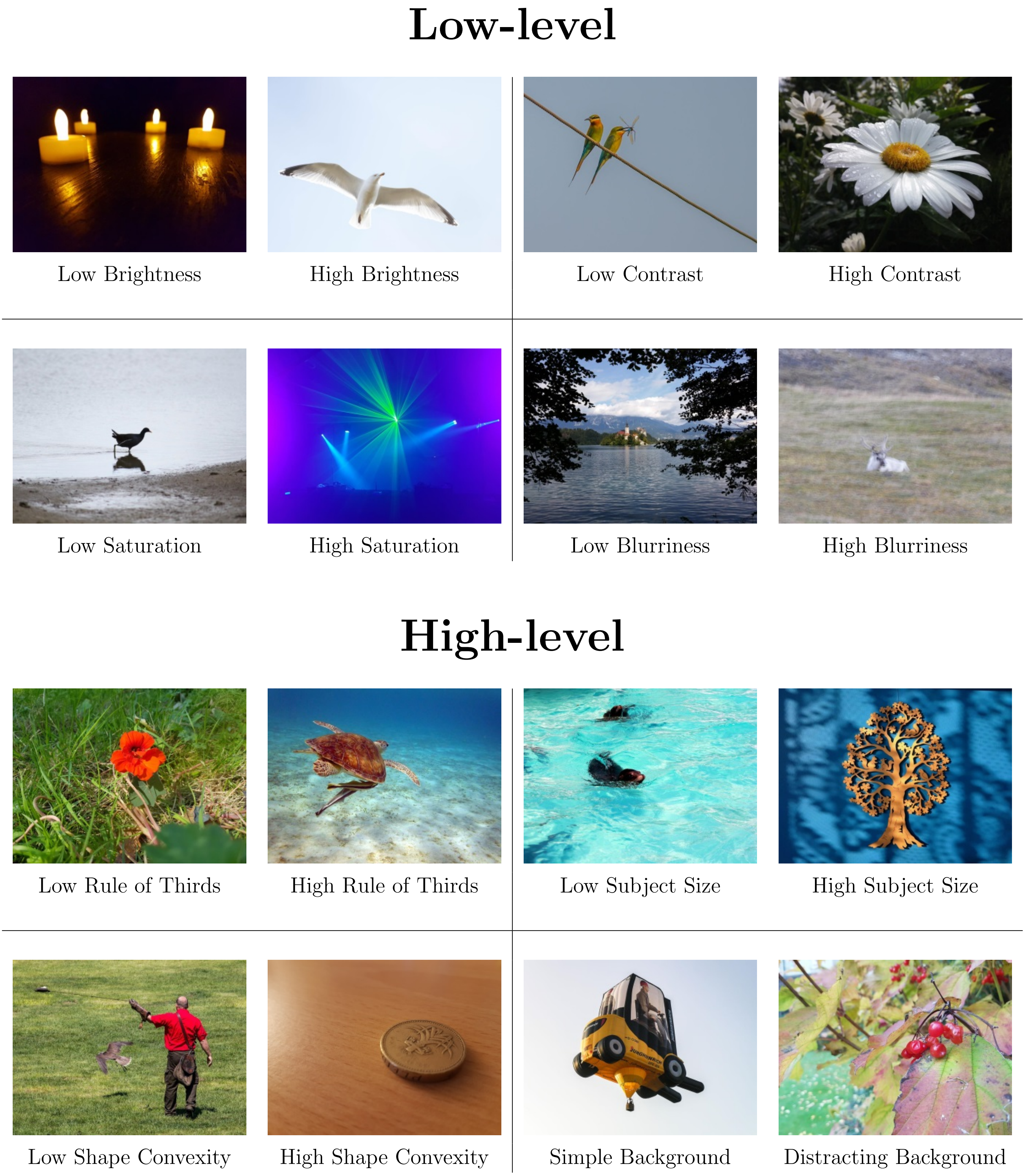
High-level features extract compositional information about the subject(s) of the image. The reason for choosing this particular set is that, between all of them, all pixels in the image are considered and analysed at a high level; the *size*, *position*, and *shape* of the subject(s) is quantified.

1. **Subject(s) Size (CORE TASK)**
   Yeh *et al.* [22] suggest that the size of the subject can discriminate image quality; some people prefer large subjects, whereas others prefer small ones. The feature draws bounding boxes around the subjects, and computes the fraction of the image taken up by them.

2. **Rule of Thirds (CORE TASK)**
   There is a photography rule-of-thumb which says that subjects should be close to the *intersections* of third-lines of the image, called *power points.* Modified from Yeh *et al.* [22], this feature computes a weighted sum of the segments, weighted by size, saliency, and distance to the closest power point.

3. **Shape Convexity (EXTENSION TASK)**
   Research by Vartanian *et al.* showed that "participants were more likely to judge spaces as beautiful if they were curvilinear than rectilinear" [19], and neuroscientists at Johns Hopkins University found people "like shapes with gentle curves as opposed to sharp points."[1]. Heavily modified from Datta *et al.* [8], this feature works out which of the subjects are like convex objects (no jagged edges or structure). It then quantifies how circular these salient, convex segments are.

[1] Study outlined at `http://www.smithsonianmag.com/science-nature/do-our-brain...`



Low-level

Low Brightness

High Brightness

Low Contrast

High Contrast

Low Saturation

High Saturation

Low Blurriness

High Blurriness

High-level

Low Rule of Thirds

High Rule of Thirds

Low Subject Size

High Subject Size

Low Shape Convexity

High Shape Convexity

Simple Background

Distracting Background

Figure 2.5: Images with low and high feature values

4. **Background Distraction** **(EXTENSION TASK)**
   Luo *et al.* [15] hypothesise that "simple" images are more appealing. This feature uses everything in the image *not* considered to be a subject: it computes how "busy" the background is by counting the number of different colours in it.

The low-level features are simpler, pixel-wise metrics, of which I chose four. These sorts of features commonly appear in the literature:

1. **Brightness** **(CORE TASK)**
   The average intensity of the image. Relating back to a potential application use-case, a magazine editor may need brightness to correctly set the atmosphere for their article image.
2. **Intensity Contrast** **(CORE TASK)**
   Also known as Weber contrast, this is the average absolute difference from the mean image intensity.
3. **Saturation** **(EXTENSION TASK)**
   The average HSV saturation of all pixels in the image. Saturation is a measure of how "colourful" a colour is.
4. **Blurriness** **(EXTENSION TASK)**
   An analysis of the frequencies present in the image. This uses the 2DFT.

2.3 Libraries and Tools

Throughout the project I made effective use of libraries and tools where appropriate:

- The standard libraries with the chosen programming language, C#.NET, make image/file handling simple, with no need to write or find low-level graphics packages.
- Windows Presentation Foundation (WPF) was used to easily create Windows GUIs using the MVVM pattern.
- Git for version control, which integrates well with Visual Studio. The *Feature Branch Workflow* was used: new features were created on separate branches, such as `feature/newFeature`. The `testing/newFeature` branch would be used for testing. After implementation and testing, changes were merged back to the `master` branch.
- The Berkeley Segmentation Dataset contains human segmentations of many images [2]. It was used to perform a numerical evaluation of my segmentation implementation.
- RGB ↔ CIELAB conversions and Gaussian blurring used the .NET Image Library[2]. Colour-space conversions are non-linear transformations which are highly prone to implementation error. It therefore made sense to use an existing implementation.

[2] `https://github.com/fschultz/NetImageLibrary`

- The *Blurriness* feature uses a Cooley-Tukey library[3] for the standard 2DFT.
- To compute convex hulls for the *Shape Convexity* feature, I used the `MIConvexHull` library[4], since it is an already solved problem with optimised solutions (Part IA Algorithms Course).
- For storing images in memory, .NET's `Bitmap` class has slow pixel read/write access due to locking memory. Since I never access the same image in multiple threads, I used the `DirectBitmap`[5] wrapper class to bypass locking/unlocking.

2.4 Requirements Analysis

As required by the Agile project strategy, I made a chronological list of implementation tasks to complete. It is summarised in Table 2.1, but the full list is in Appendix A.

	Low Risk	Moderate Risk	High Risk
Low Difficulty	– Saliency Augmentation – Subject(s) Size – Brightness – Intensity Contrast – Saturation		
Moderate Difficulty	– Background Distraction	– Rule of Thirds – Blurriness – Exif Metadata	
High Difficulty	– Segmentation – Efficient Interface		– Shape Convexity – Intuitive Interface

Table 2.1: Implementation tasks organised by difficulty and risk (**core** first, and **extensions** later, time-permitting)

2.5 Software Engineering Strategy

I adopted the Agile software engineering framework for my project: I had two-week sprints with regular supervisor meetings. Usually, end-of-sprint meetings involved checking a deliverable or milestone, for example the completion of a new feature's implementation and testing. During the meetings, my supervisor and I did code reviews of important modules (and research summaries during the research phase). The Agile approach suited my project well, since I could mostly break the workload down into separately implementable and testable modules, such as individual features, or phases of evaluation.

[3] `https://www.codeproject.com/kb/gdi/fft.aspx`

[4] `https://github.com/DesignEngrLab/MIConvexHull`

[5] `http://stackoverflow.com/questions/24701703/c-sharp-faster-altern...`

I have made my code open-source, so that others may use and/or contribute to it[6]. I have not included proprietary libraries, and have documented my code so that it may be easily understood by others. Furthermore, I have tried to keep to C#.NET code conventions. Initially, I did not know these, since the language was new to me. However, the Visual Studio plugin called ReSharper was used to warn when code conventions were broken. Not only did this make my code cleaner and consistent, but it taught me many conventions, which will be useful for future projects in the language.

2.6 Planned vs Actual Work Done

Figure 2.6 gives a detailed breakdown of actual work done compared with the planned work blocks in the Project Proposal. The green section over Christmas was for testing the implementation, creating an optimised feature computer structure that was multi-threaded, and saving computed image feature values in JPEG Exif metadata for near-instant re-use. This section was added during the research phase, since I had not realised its importance at the time of writing the proposal.

Figure 2.7 shows the `master` branch git commit history for the project. The first commit was on the 13th November 2016, and the latest was on the 5th March 2017.

[6] `https://github.com/matwx/Part2Project`

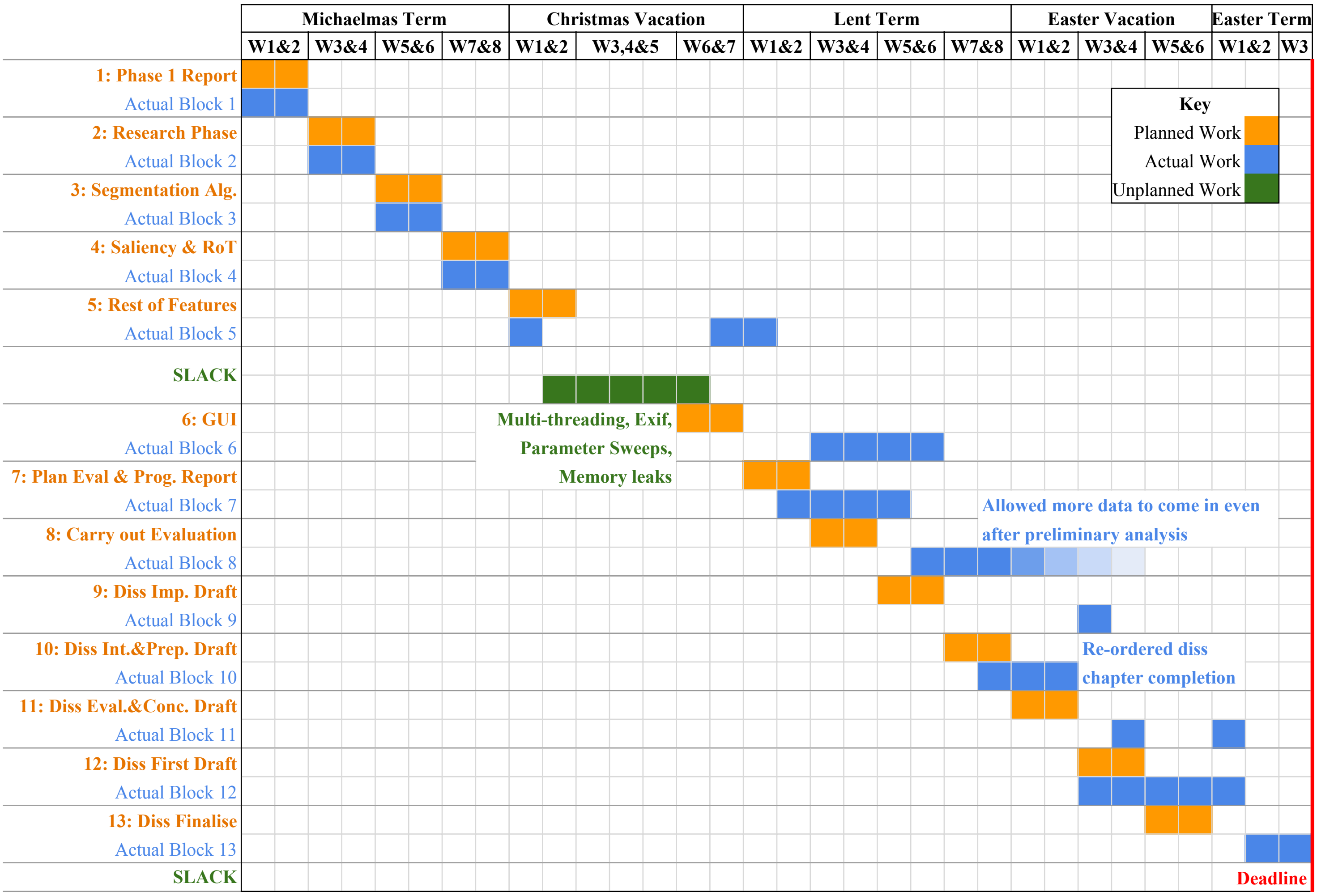


Figure 2.6: Gantt Chart showing planned vs actual work done

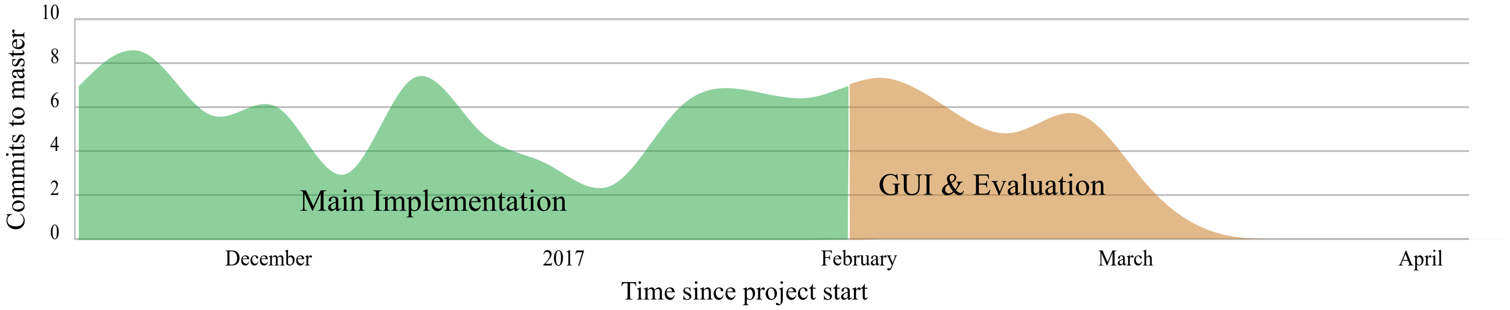


Figure 2.7: Git commits to `master` branch during the project

Chapter 3

Implementation

This chapter is guided by the diagrams on the next page (Figures 3.1-3.4). The first depicts the overall tool architecture, and each successive diagram zooms in on the most important part of the last.

The efficient, multi-threaded feature computer architecture is first described, followed by the saliency segmentation implementation, then the features themselves, and finally the user interface. This chapter ends with a discussion of optimisations, the feature testing strategy, and some interesting bugs.

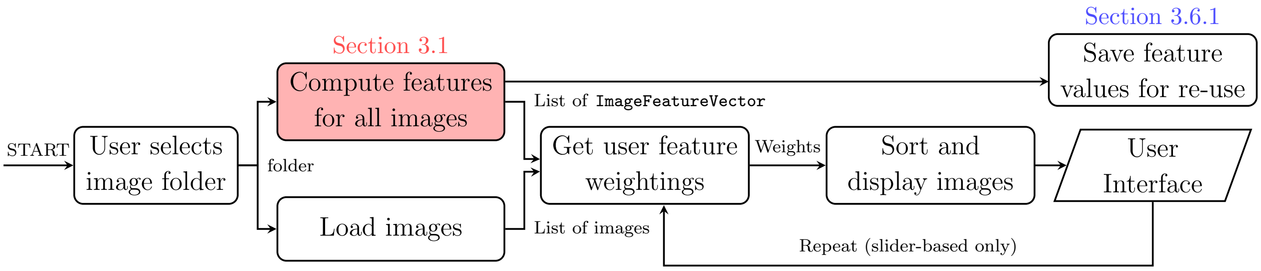

Section 3.1
Section 3.6.1
START
User selects image folder
folder
Compute features for all images
Load images
List of ImageFeatureVector
List of images
Save feature values for re-use
Get user feature weightings
Weights
Sort and display images
User Interface
Repeat (slider-based only)

Figure 3.1: User Interface

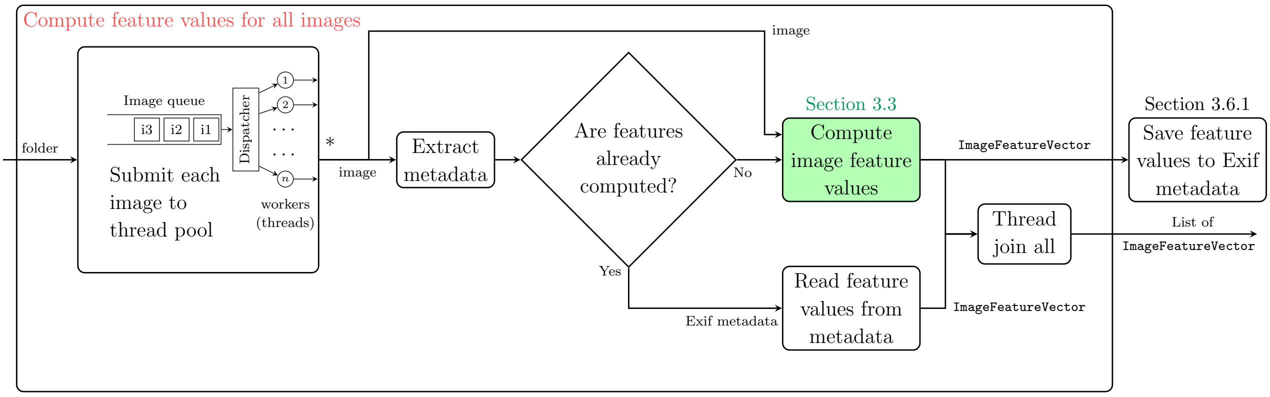

Compute feature values for all images
folder
Image queue
i3
i2
i1
Dispatcher
1
2
. . .
. . .
n
workers (threads)
Submit each image to thread pool
*
image
image
Extract metadata
Are features already computed?
No
Yes
Exif metadata
Section 3.3
Compute image feature values
Read feature values from metadata
ImageFeatureVector
ImageFeatureVector
Thread join all
Section 3.6.1
Save feature values to Exif metadata
List of ImageFeatureVector

Figure 3.2: Multi-threaded Feature Computer

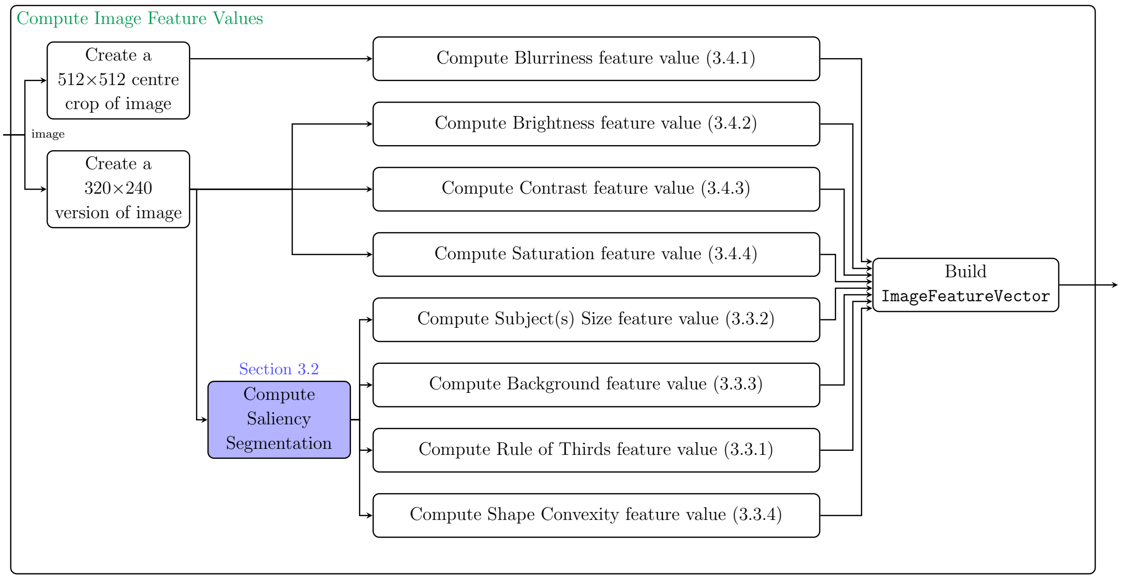

Compute Image Feature Values
image
Create a 512×512 centre crop of image
Create a 320×240 version of image
Section 3.2
Compute Saliency Segmentation
Compute Blurriness feature value (3.4.1)
Compute Brightness feature value (3.4.2)
Compute Contrast feature value (3.4.3)
Compute Saturation feature value (3.4.4)
Compute Subject(s) Size feature value (3.3.2)
Compute Background feature value (3.3.3)
Compute Rule of Thirds feature value (3.3.1)
Compute Shape Convexity feature value (3.3.4)
Build ImageFeatureVector

Figure 3.3: Computing Image Feature Values

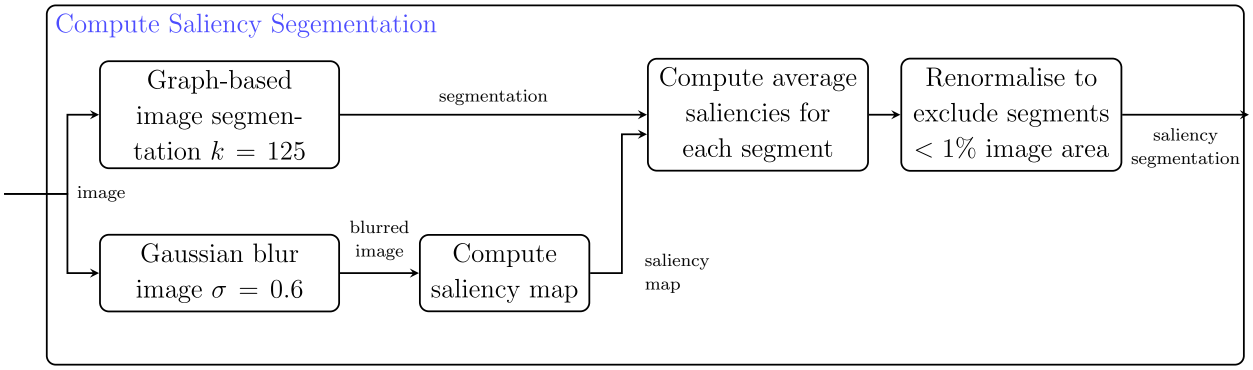

Compute Saliency Segementation
image
Graph-based image segmentation $k = 125$
segmentation
Gaussian blur image $\sigma = 0.6$
blurred image
Compute saliency map
saliency map
Compute average saliencies for each segment
Renormalise to exclude segments < 1% image area
saliency segmentation

Figure 3.4: Saliency Segmentation

3.1 Multi-threaded Feature Computer

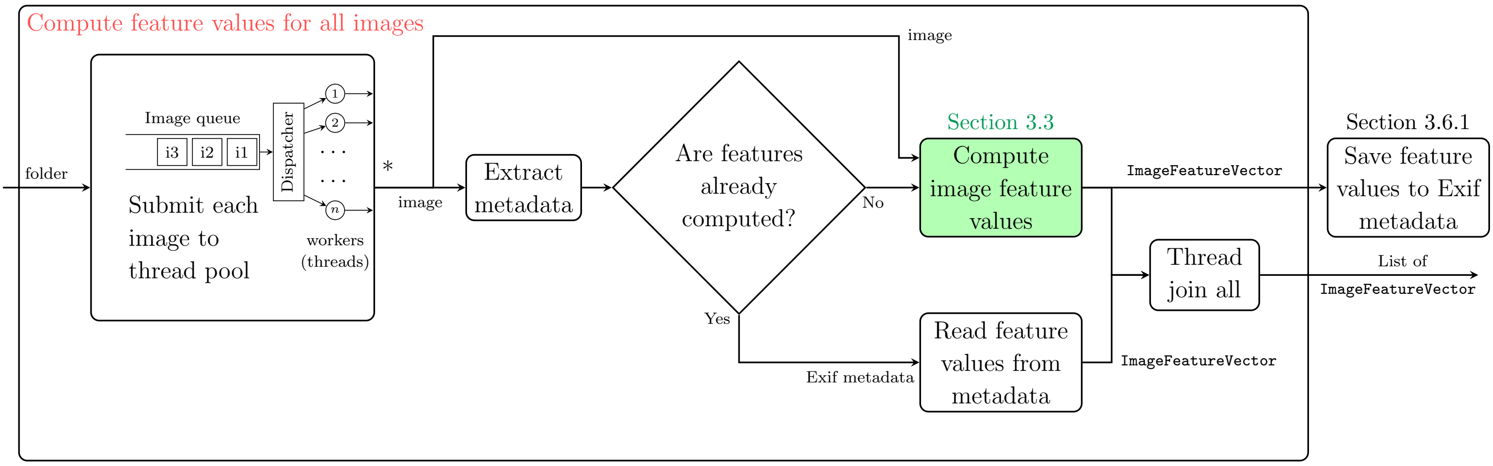

Compute feature values for all images
folder
Image queue
i3
i2
i1
Dispatcher
1
2
n
workers (threads)
Submit each image to thread pool
*
image
Extract metadata
Are features already computed?
No
Yes
image
Section 3.3
Compute image feature values
Exif metadata
Read feature values from metadata
ImageFeatureVector
ImageFeatureVector
Thread join all
Section 3.6.1
Save feature values to Exif metadata
List of ImageFeatureVector

Figure 3.5: Multi-threaded Feature Computer

The project's principal focus is investigating segmentation-derived features, so the computation of such feature values from images forms the main part of the program. It is important to have an efficient feature computer because people will not have the patience to wait a long time when using it. My program achieves a 2.75× speed-up by allowing multiple images to be processed in parallel.

C# `Tasks` are, according to Microsoft, "the preferred way to write multithreaded and parallel code"[1]. Computation of an image's features constitutes a `Task`. These are queued by the program, and scheduled and assigned to available logical threads by the *thread pool* (Figure 3.5). Since there are never multiple threads working on the same image, there is no risk of race conditions or deadlock. Each image `Task` returns a new `ImageFeatureVector` and, after all `Tasks` have joined/finished, a list of these vectors is constructed and returned.

Within the Agile framework, I prioritised the implementation of an early working single-threaded version to allow progress on dependent tasks. In later sprints, I refined and re-implemented this with the multi-threaded architecture described above. Figure 3.6 shows the speed-up achieved by the parallelised version.

[1] `https://msdn.microsoft.com/en-us/library/dd460717(v=vs.110).aspx`

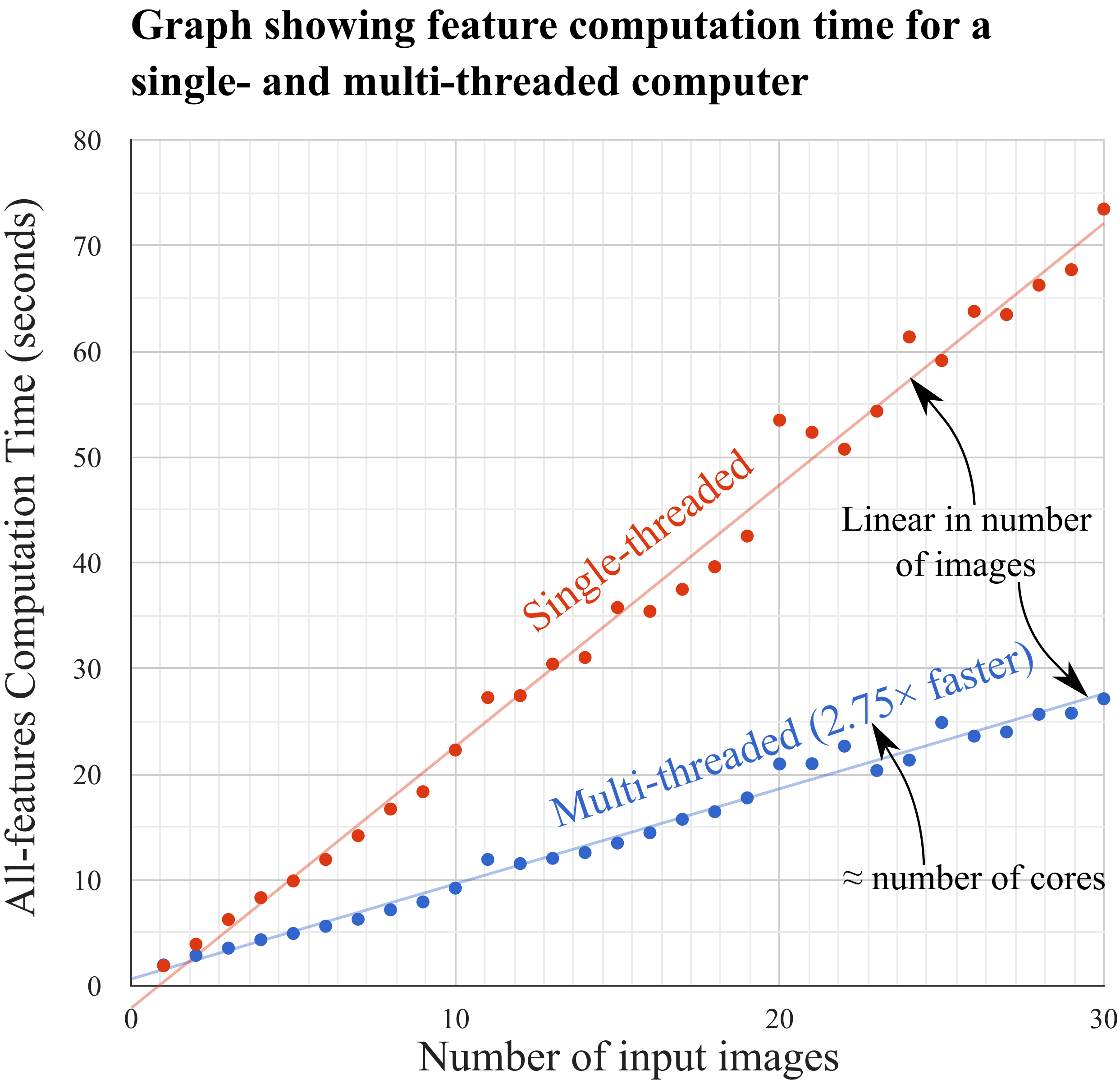


Figure 3.6: The exact timing results here are not particularly important, since they are dependent on my computer. What is of interest are the relative differences. The gradients of the trend-lines are about 2.474 for single-threaded and 0.900 for multi-threaded.

3.2 Segmentation and Saliency Segmentation

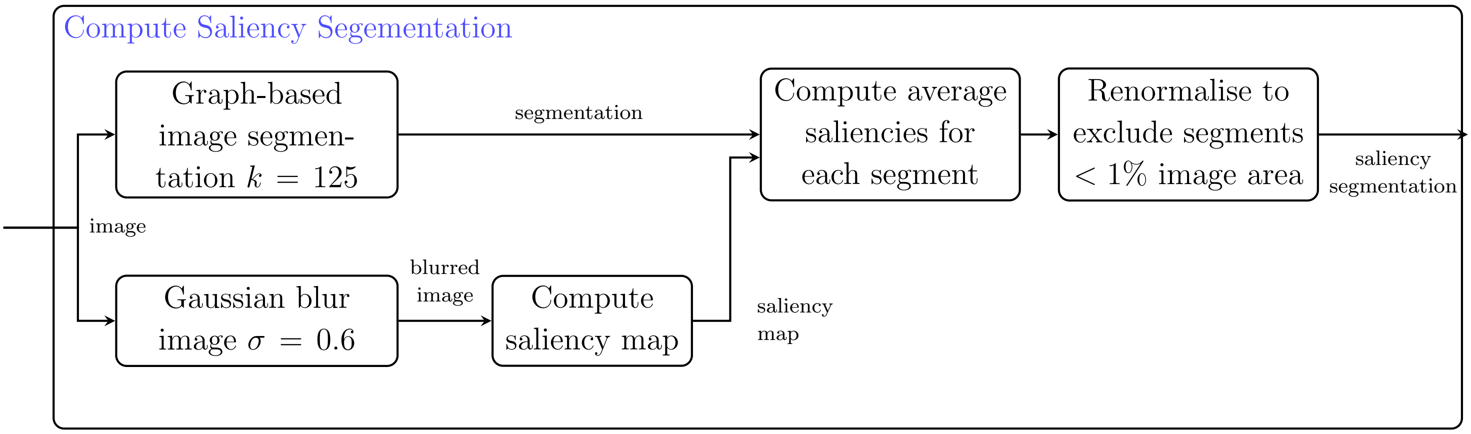

Compute Saliency Segementation
Graph-based image segmentation $k = 125$
segmentation
Compute average saliencies for each segment
Renormalise to exclude segments $< 1\%$ image area
saliency segmentation
image
Gaussian blur image $\sigma = 0.6$
blurred image
Compute saliency map
saliency map

Figure 3.7: Saliency Segmentation

Before high-level features are extracted, the image needs to be simplified into symbols by segmentation. This module runs the image segmentation algorithm, and then combines saliency information with the segmentation into a *saliency segmentation*.

3.2.1 Graph-based Image Segmentation

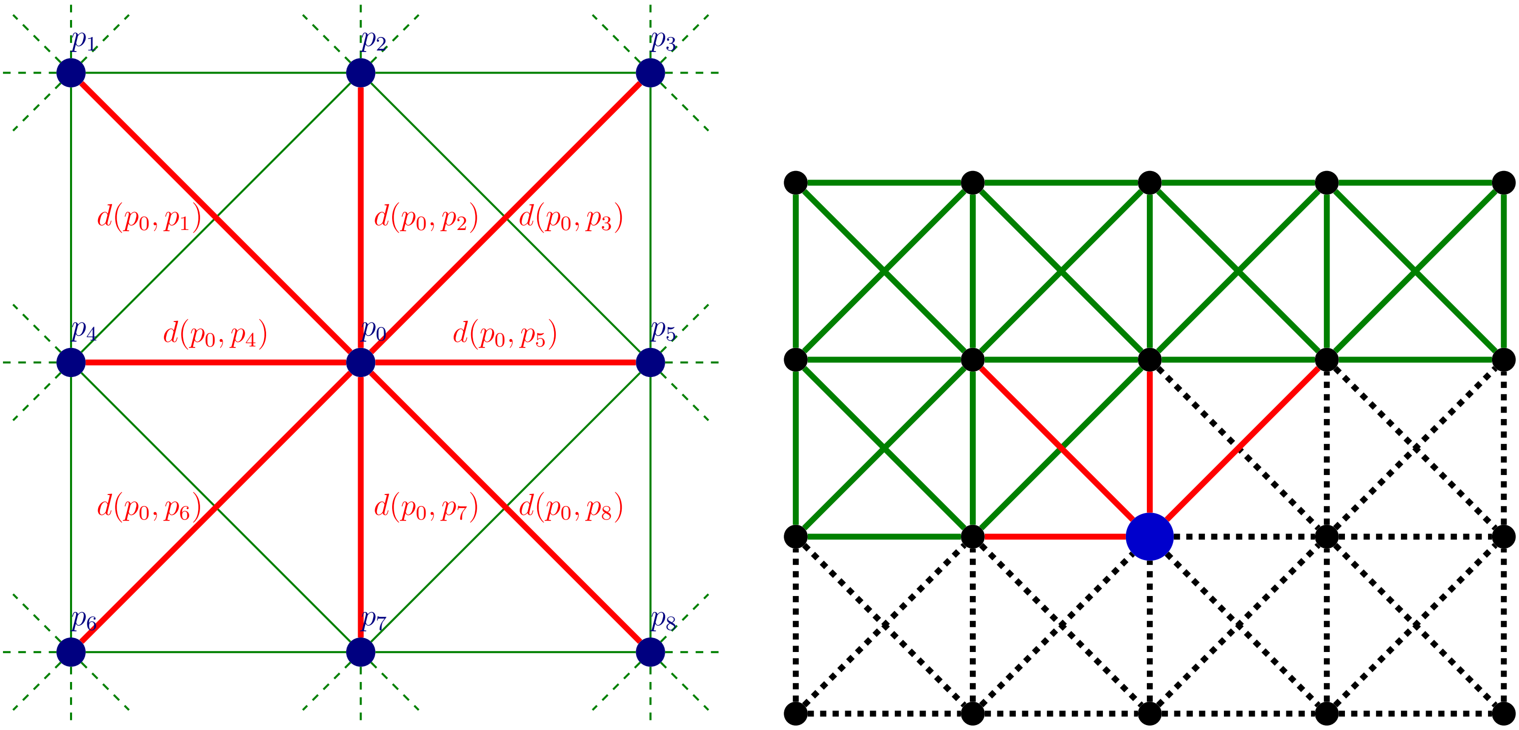
The image is segmented in order to obtain high-level symbols analogous to those obtained by cortical V1 neurons in the human brain, to approximate the object detection/recognition process that humans likely use when evaluating image quality. The chosen segmentation algorithm was proposed in Felzenszwalb & Huttenlocher's paper "*Efficient Graph-Based Image Segmentation*" [9].

Implementing the algorithm myself allowed me to: adjust the output format for easy augmentation with saliency information; investigate the way the algorithm measures pixel difference (Section 3.2.2); and use different parameter values from those in the original paper.

The output of the algorithm is a mapping from pixel position (x, y) to a segment index. Pixels mapping to the same segment index are in the same segment. My implementation also calculates segment sizes and average colour (in CIELAB space), which it associates with the segmentation.

The first and most computationally expensive step is to construct the undirected graph $G = (E, V)$ over which the algorithm operates. Figure 3.8 shows a section of the constructed graph, as well as the order in which edges are created such that each edge is added exactly once.

A *disjoint-set forest* data structure is maintained throughout the running of the algorithm. At any step, there is a forest of disjoint, but mutually spanning MSTs of vertices, each corresponding to a segment of the image. The core disjoint-set data structure algorithm, with the *union by rank* and *path compression* optimisations is in CLRS [7] (p571).



(a) A section of the constructed graph

(b) A snapshot of the graph creation process

Figure 3.8: (a) There is a vertex for each image pixel. Red edges are those connected to vertex p_0, and weights measure pixel-difference. All other edges are shown in green. (b) Vertices are considered in English reading order. The vertex considered at this step is blue, the edges to be added at this step are red, previously added edges are green, and edges still to be added are dotted.

```
MAKE-SET($x$)
$x.p = x$
$x.rank = 0$
```

```
MAKE-SET($x$, $X$, $Y$)
$x.p = x$
$x.rank = 0$
$x.size = 1$
$x.X = X$
$x.Y = Y$
$x.iDiff = 0$
$x.totalL = I(X,Y).L$
$x.totalA = I(X,Y).a$
$x.totalB = I(X,Y).b$
```

```
FIND-SET($x$)
if $x \neq x.p$ then
    $x.p =$ FIND-SET($x.p$)
end if
return $x.p$
```

```
FIND-SET($x$)
if $x \neq x.p$ then
    $x.p =$ FIND-SET($x.p$)
end if
return $x.p$
```

```
UNION($x$, $y$)
LINK(FIND-SET($x$),FIND-SET($y$))
```

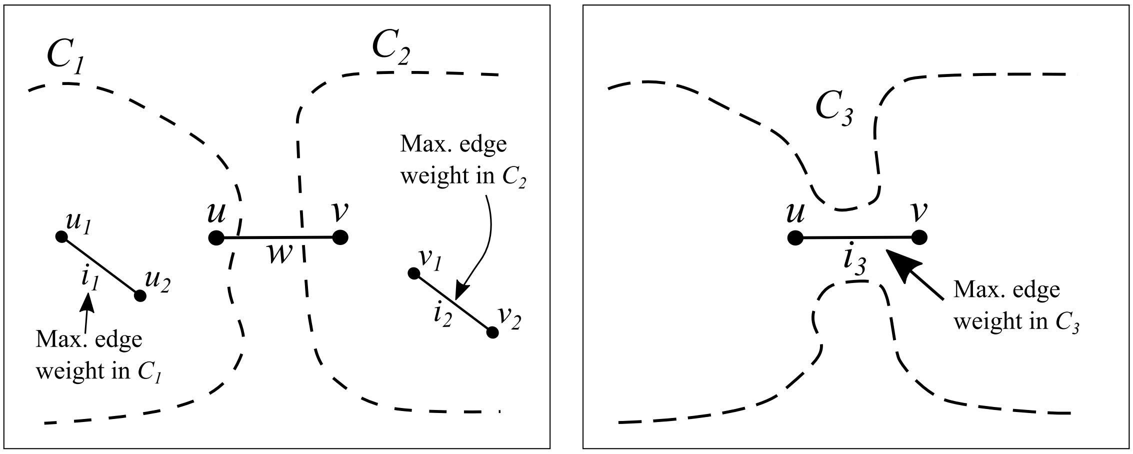
```
UNION($x$, $y$, $joiningEdgeWeight$)
LINK(FIND-SET($x$),FIND-SET($y$),
    $joiningEdgeWeight$)
```

```
LINK($x$, $y$)
if $x.rank > y.rank$ then
    $y.p = x$
else
    $x.p = y$
    if $x.rank == y.rank$ then
        $y.rank = y.rank + 1$
    end if
end if
```

```
LINK($x$, $y$, $joiningEdgeWeight$)
if $x.rank > y.rank$ then
    $y.p = x$
    $x.size += y.size$
    $x.iDiff = joiningEdgeWeight$
    $x.totalL += y.totalL$
    $x.totalA += y.totalA$
    $x.totalB += y.totalB$
else
    $x.p = y$
    if $x.rank == y.rank$ then
        $y.rank = y.rank + 1$
    end if
    $y.size += x.size$
    $y.iDiff = joiningEdgeWeight$
    $y.totalL += x.totalL$
    $y.totalA += x.totalA$
    $y.totalB += x.totalB$
end if
```

from "Introduction to Algorithms" (CLRS) | **Augmented for segmentation**

Figure 3.9: Disjoint-set augmented code


C_1
C_2
u_1
i_1
u_2
Max. edge weight in C_1
u
w
v
Max. edge weight in C_2
v_1
i_2
v_2
C_3
u
v
i_3
Max. edge weight in C_3

Before merge

After merge

Figure 3.10: The merge happens if $w \leq \min\left(i_1 + \frac{k}{|C_1|}, i_2 + \frac{k}{|C_2|}\right)$.

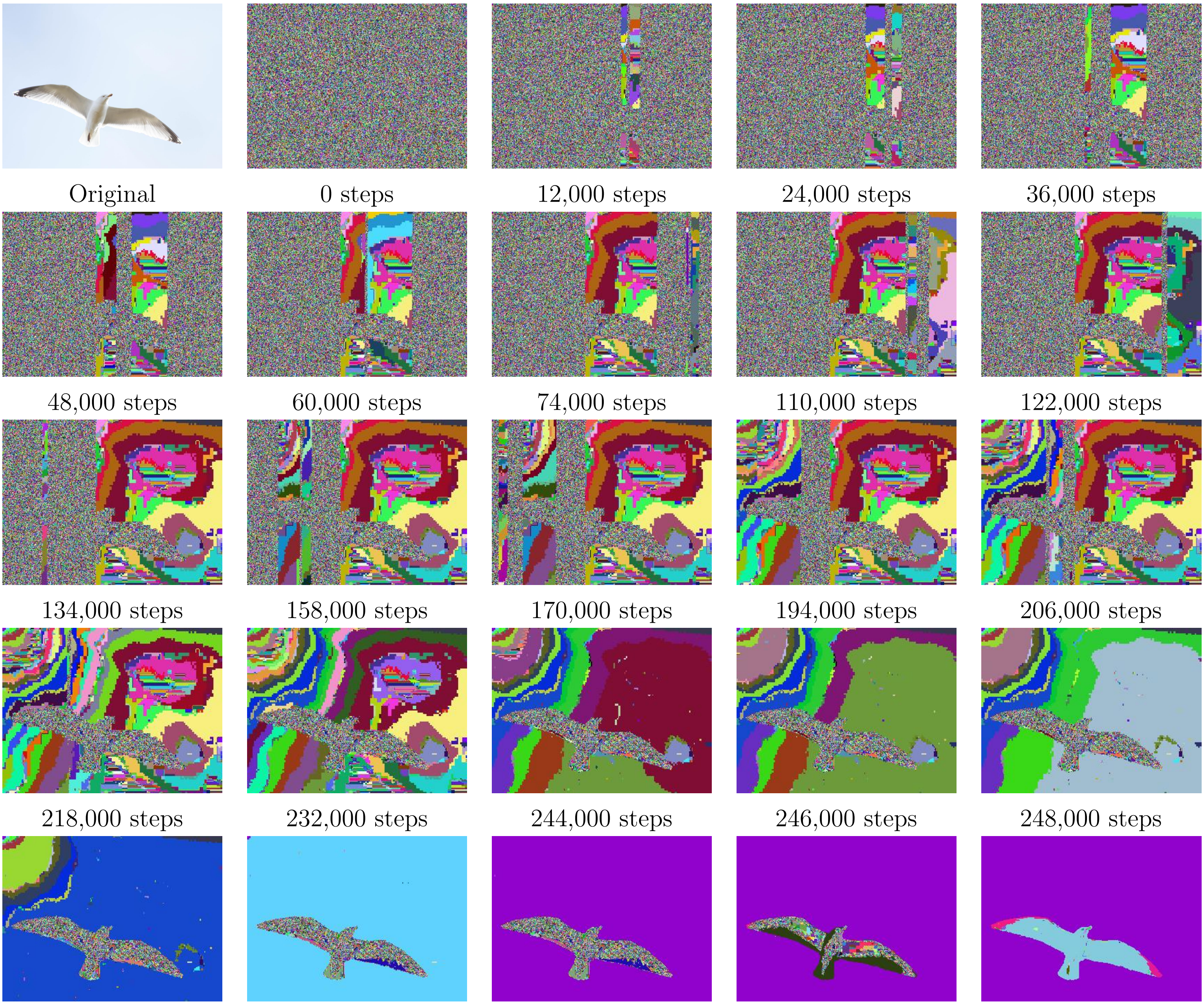
Since the structure needed to be augmented for use in the segmentation algorithm, I could not use an existing disjoint-set library. The augmentations are shown in Figure 3.9. Extra properties were added to each element, which are mostly only valid when the element is a set representative:

- *size*: the number of vertices in the element's set/segment.
- X and Y: x- and y-coordinates of the pixel that the element corresponds to.
- *iDiff*: The maximum edge weight between any two vertices within the set/segment (*internal difference*).
- *totalL*, *totalA*, and *totalB*: Totals of the element's set's CIELAB colour components, to be divided by size at the end to find the segment's average colour.

As in Kruskal's, graph edges are considered in a non-decreasing order. For each edge, $e = (u, v)$, if it does not introduce a cycle in the current forest (i.e. Find-Set$(u) \neq$ Find-Set(v)) and its weight is below an adaptive threshold $|e| \leq T(u, v)$, it is added to the forest, merging the segments that contain u and v (Union$(u, v, |e|)$). The threshold is defined as an adaptive amount above the *minimum internal difference* of the two segments to be merged:

$$T(u,v) = \min\left(\textsc{Find-Set}(u).\mathit{iDiff} + \frac{k}{\textsc{Find-Set}(u).\mathit{size}},\right.$$
$$\left.\textsc{Find-Set}(v).\mathit{iDiff} + \frac{k}{\textsc{Find-Set}(v).\mathit{size}}\right)$$

where k is a parameter. In essence, two segments are merged if their internal variations, based on my pixel-difference metric, are similar to the joining edge's weight. The merge step is illustrated in Figure 3.10, Figure 3.11 shows how the segments evolve throughout the running of the algorithm, and Figure 3.12 gives an example of the graph after algorithm termination.



Original

0 steps

12,000 steps

24,000 steps

36,000 steps

48,000 steps

60,000 steps

74,000 steps

110,000 steps

122,000 steps

134,000 steps

158,000 steps

170,000 steps

194,000 steps

206,000 steps

218,000 steps

232,000 steps

244,000 steps

246,000 steps

248,000 steps

250,000 steps

270,000 steps

272,000 steps

286,000 steps

304,000 steps

Figure 3.11: Visualisation of the runtime of the segmentation algorithm. Segments are randomly coloured. This took approximately 1.8 s for an input resolution of 320×240. A GIF version can be found at `http://imgur.com/a/WLGOh`.

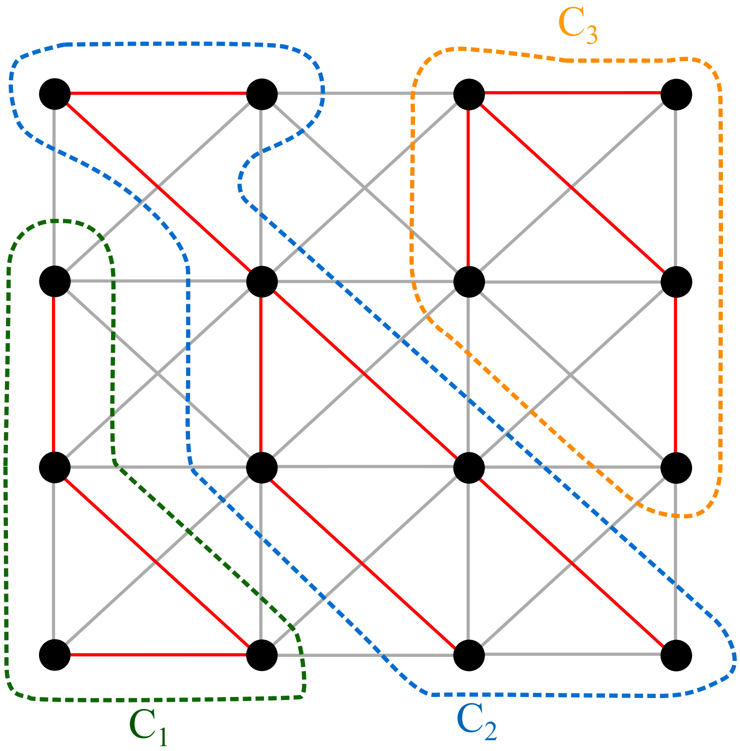


Figure 3.12: Segmentation final graph state

3.2.2 Pixel Difference Metric

The segmentation algorithm requires a function representing the *difference* between pixels p_i and p_j, which is used for the edge weights. I created a colour-difference metric for this purpose.

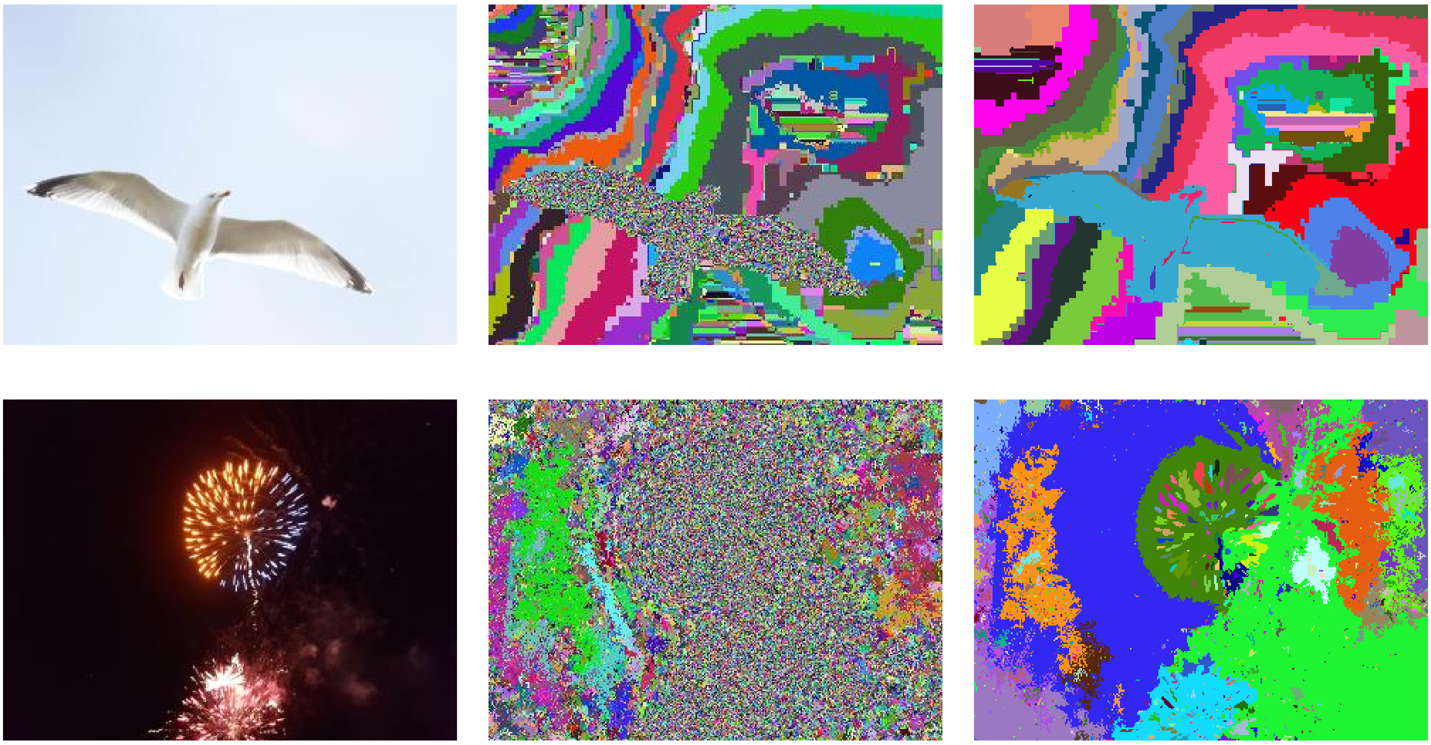
I explored and tested a selection of options from across the literature. The best was the CIEDE2000 [6] because it is a *perceptually uniform* colour-difference metric for the CIELAB colour space (Euclidean distances in the space are proportional to the colour difference perceived by an average human eye). I implemented CIEDE2000 with the help of Sharma *et al.*'s paper [17].

The reason for creating my own hybrid metric was that I sometimes noticed extra incorrect segments using CIEDE2000. I believe this to be in part due to image compression, which can create regions of constant colour, exemplified in Figure 3.13. Since the segmentation algorithm is more reluctant to merge large regions, they can become distinct segments in the result. I found this error most apparent in under- and over-exposed image regions. Therefore, I augmented the colour difference metric to act as CIEDE2000 by default and as absolute intensity difference for dark or bright pixels (< 20 or > 210).

The other options that I surveyed, from worst to best, were:

- **Absolute intensity difference**

  This operates on a greyscale version of the image. It works in the general case, but fails in many, such as in Figure 3.14, where all of the colours have the same



Original image Regions of constant colour Segmentation using CIEDE2000

Figure 3.13: Negative effect of image compression on segmentation

perceptual intensity. It was implemented first to get a working version early, so that I could test the segmentation algorithm, before being refined.

- **RGB Euclidean distance**

  This naïve measure of *colour* difference is defined as

  $$d = \sqrt{(R_1 - R_2)^2 + (G_1 - G_2)^2 + (B_1 - B_2)^2}. \tag{3.1}$$

  Felzenszwalb & Huttenlocher, when they proposed the segmentation algorithm, restricted their pixel difference to be based on RGB information. However, the RGB colour space is known to be not *perceptually uniform*.

- **CIELAB Euclidean distance**

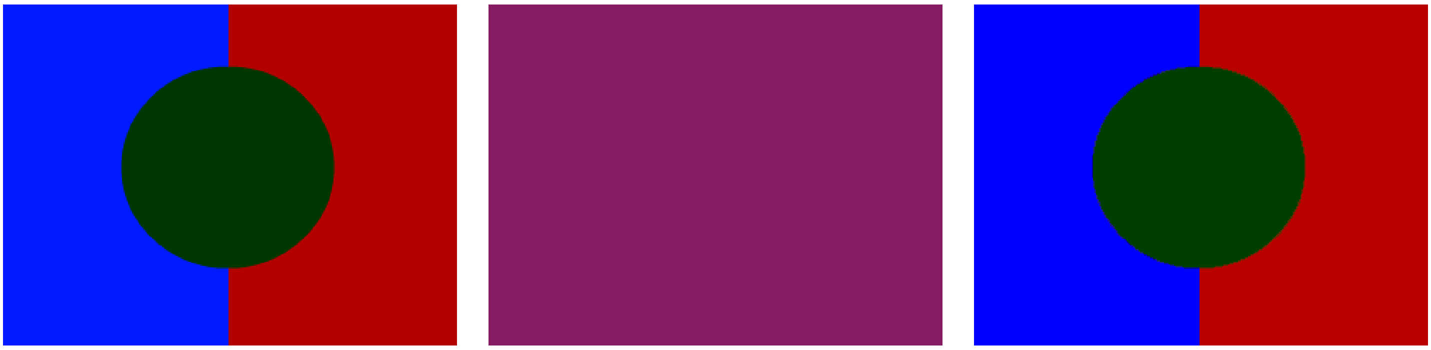
  The CIELAB colour space [6] is partially *perceptually uniform*. In 1976, the preferred colour-difference measure was L*a*b* Euclidean distance:

  $$d = \sqrt{(L_1^* - L_2^*)^2 + (a_1^* - a_2^*)^2 + (b_1^* - b_2^*)^2}. \tag{3.2}$$

  For the RGB → L*a*b* conversion in my program, I used the .NET Image Library developed by Fredrik Schultz and Contributors[2].

Figures 3.15 and 3.16 compare the different options and highlight where my measure helps.

[2] `https://github.com/fschultz/NetImageLibrary`



Original

Segmentation using intensity as pixel difference

Segmentation using colour as difference (Me)

Figure 3.14: By using intensity difference as the pixel difference metric in the segmentation algorithm, important information that humans use to detect objects is lost.

3.2.3 Adding Saliency Information

I combine the segmentation with saliency information as in the approach of Yeh *et al.*[22], making a *saliency segmentation.*

For extracting saliency information, I used Achanta *et al.*'s *saliency map* [1], using CIEDE2000 instead of Euclidean CIELAB distance. I tried my hybrid colour difference metric here but, because it would require multiple transformations from RGB ↔ LAB, floating point errors ruined the result.

The saliency map is computed as follows:

```
SALIENCY-MAP($I$)
B = GAUSSIAN-BLUR($I$, $\sigma : 0.6$)                      ▷ Blur the image
C = MEAN-CIELAB-COLOUR($I$)                    ▷ Find the average image colour
$s_{max} = 0$
for each $p_{xy}$ do
    $s_{xy}$ = CIEDE2000(RGB-TO-CIELAB($p_{xy}$), $C$)         ▷ Colour difference
    $s_{max} = \max(s_{max}, s_{xy})$
end for
for each $p_{xy}$ do                                   ▷ Normalise saliency map
    $s_{xy} = s_{xy}/s_{max}$
end for
return $S$
```

A segment's saliency is the average of the saliencies of its constituent pixels. The segment saliencies are normalised, ignoring all segments smaller than 1% of the image. Figures 3.17 and 3.18 exemplify the process.

The "ignoring segments < 1%" step was added because I discovered that feature implementations later in the pipeline could not correctly detect image subjects. I noticed a trend of small, high-salience regions not corresponding to actual image objects. After

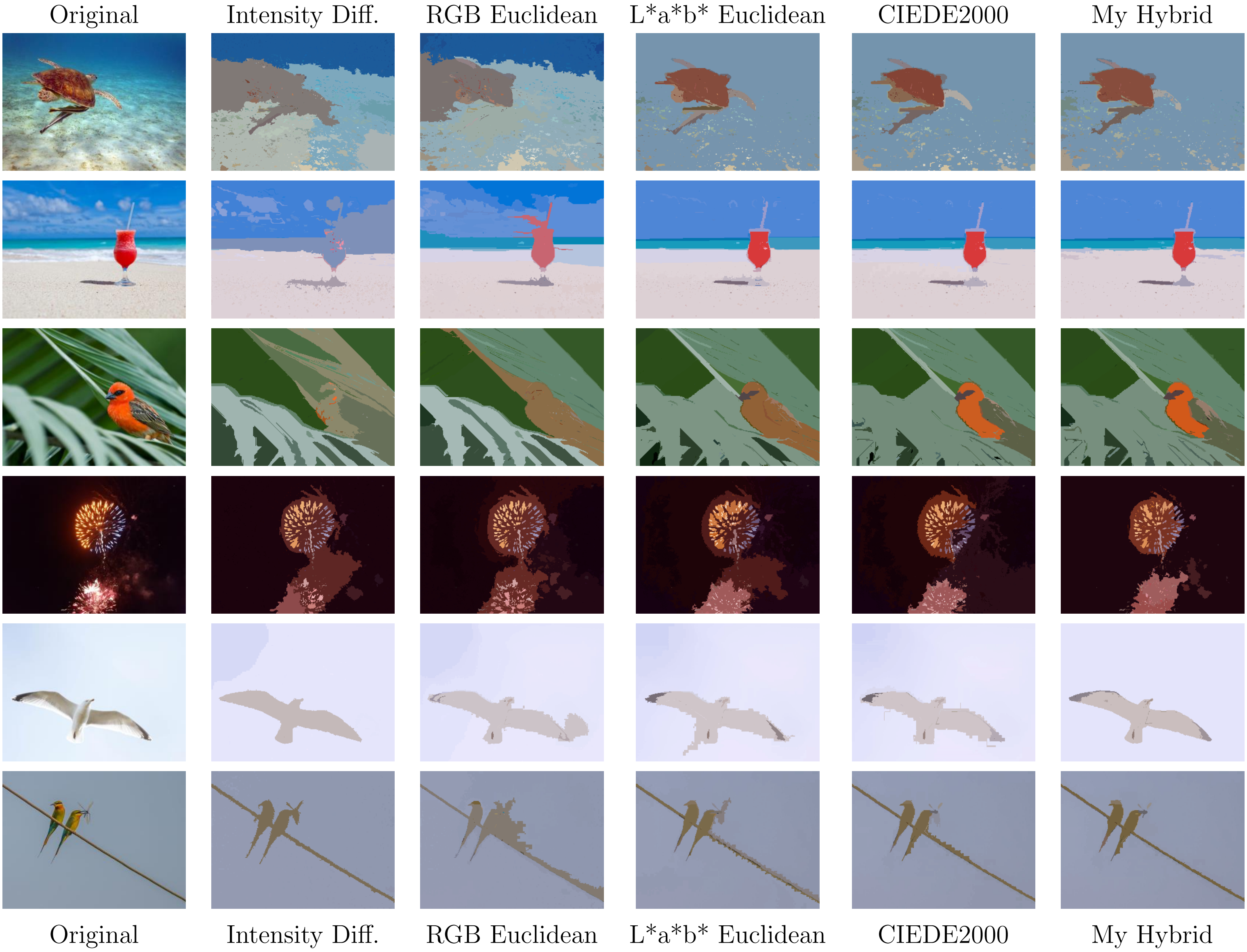


Figure 3.15: Comparison of pixel difference metrics (Average segment colouring). Segment spilling is less extreme in my hybrid, so important information is not “smudged”.

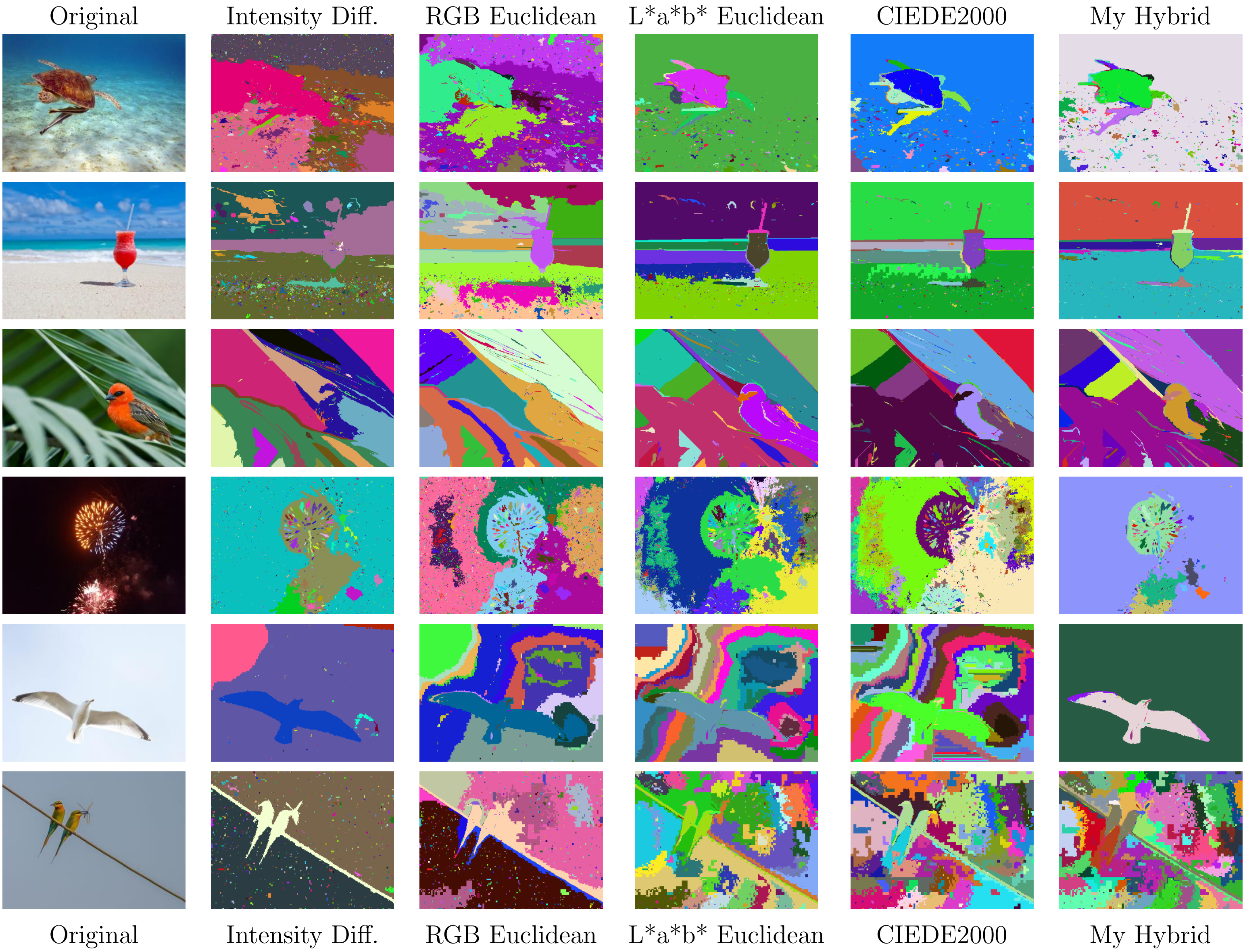
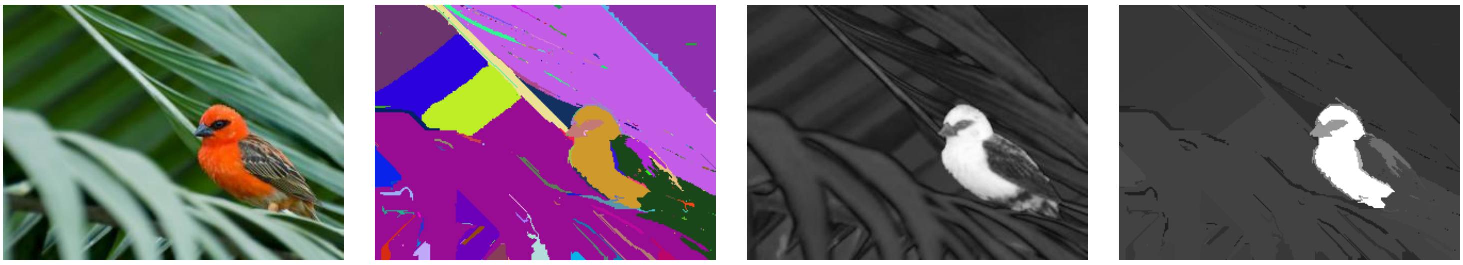


Figure 3.16: Same data as in Figure 3.15, but segments are coloured randomly, which gives a better picture of what the computer has to work with. Images 4 and 5 show how my hybrid metric successfully deals with the compression artefacts. However, image 6 shows that it does not always work; in this example, the compressed region is neither over- or under-exposed, so the hybrid inherits the error of CIEDE2000. To further assist with these artefacts, my implementation uses saliency information.



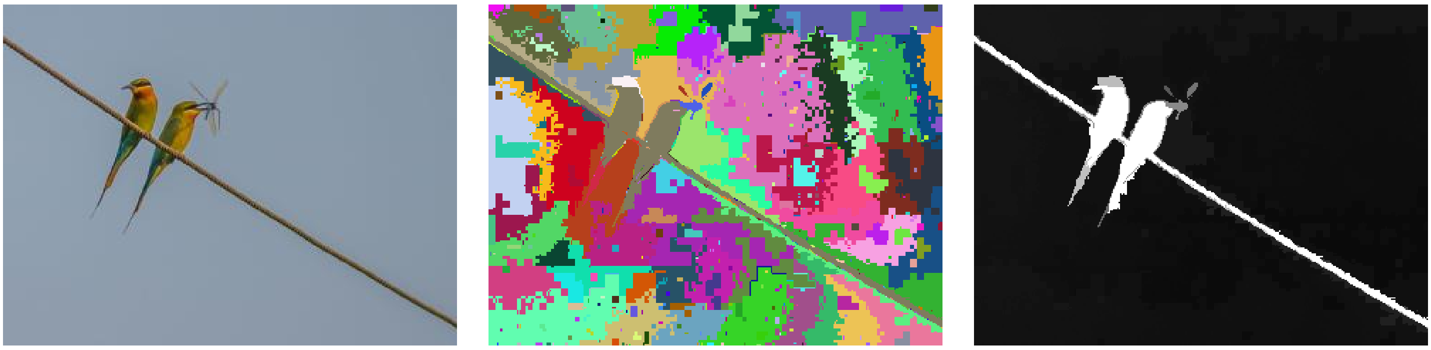
Original

Segmentation

Saliency Map

Saliency Segmentation

Figure 3.17: Segmentation can only partition the image. The saliency segmentation is used to try to find the subject(s). This is important for extracting features about their positions/sizes/shapes.

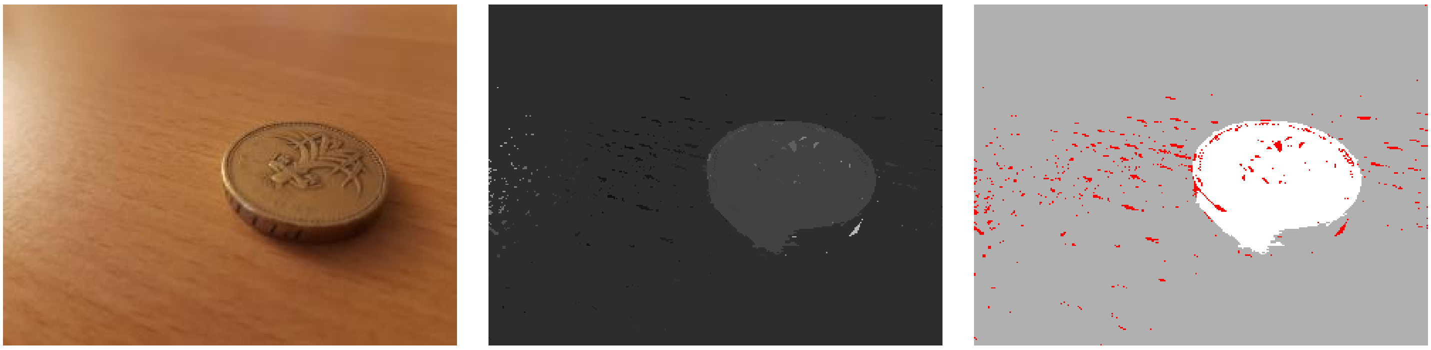


Original

Segmentation

Saliency Segmentation

Figure 3.18: The compression artefacts of image 6's segmentation in Figure 3.16 are mitigated by the saliency segmentation.

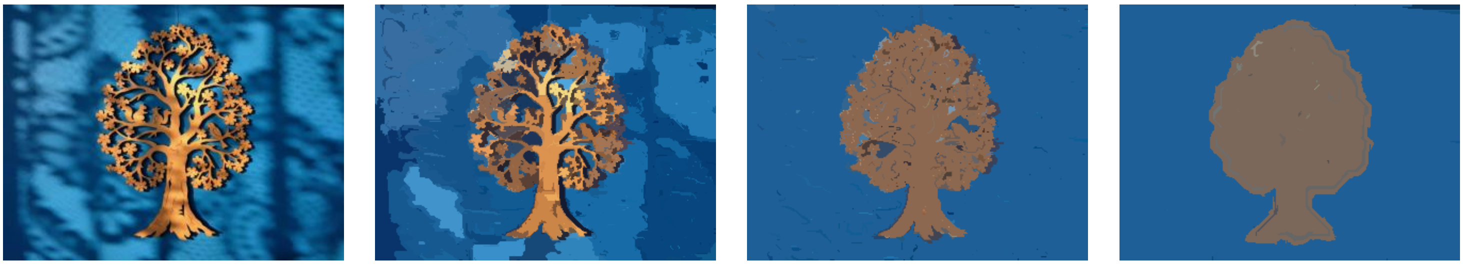


Original

Before ignoring small segments

After ignoring small segments (Ignored segments in red)

Figure 3.19: The original saliency segmentation gave maximum saliency scores to small, spurious segments.



Original

$k = 25$, $\sigma = 0$

$k = 125$, $\sigma = 0$

$k = 125$, $\sigma = 4$

Figure 3.20: Decreasing k causes more and smaller segments. Increasing σ blurs the image more before.

ignoring these, noise in the high-level features was significantly reduced (see Figure 3.19). Although I acknowledge that 1% is a somewhat arbitrary cut-off point for "too small" segments, the intuitive justification is that a person probably would not try to take a picture of something so small in the frame.

3.2.4 Choosing the Segmentation Parameters

The algorithm has two parameters

- k Related to the scale of observation. A larger value encourages larger segments;
- σ Also related to scale. A pre-processing Gaussian blur is performed. A larger value encourages segments to form for more general objects, rather than finer details within objects.

Figure 3.20 gives an example of changing the parameters.

In Felzenszwalb & Huttenlocher's paper, they state parameter values without much justification, so I performed a systematic parameter sweep for k and σ to determine reasonable values. The sweep tested segmentation-derived features on a set of 80 images for different

k, σ	Score	k, σ	Score	k, σ	Score	k, σ	Score
25, 1	51	75, 0	56	**100, 0**	61	150, 0	54
25, 1.4	46	74, 0.4	58	100, 0.4	56	150, 0.8	58
25, 2.5	52	75, 0.6	59	**100, 0.6**	61	175, 1.4	51
50, 0.4	48	**75, 0.8**	60	100, 1.8	58	200, 0	55
50, 0.6	58	**75, 1**	64	**125, 0**	61	250, 0.6	53
50, 0.8	57	75, 1.4	59	125, 0.4	53		
50, 1.4	62	75, 1.8	57	**126, 0.6**	61		
50, 1.8	60			125, 0.8	59		

Table 3.1: Segmentation parameter sweep initial results. Highlighted pairs advanced to further testing.

(k, σ) pairs. Table 3.1 gives the initial results. I then tested on more images for pairs scoring above 60/80 to finally decide on $k = 125$, $\sigma = 0.6$.

While doing the sweeps, I spotted a potential trend: it seemed that, as k increased, a suitable σ resulting in similar segmentations decreased. This seemed to be a linear relationship for $25 < k < 150$. Some interesting further work could be a more rigorous investigation of this hypothesis.

3.3 Computing High-level Image Features

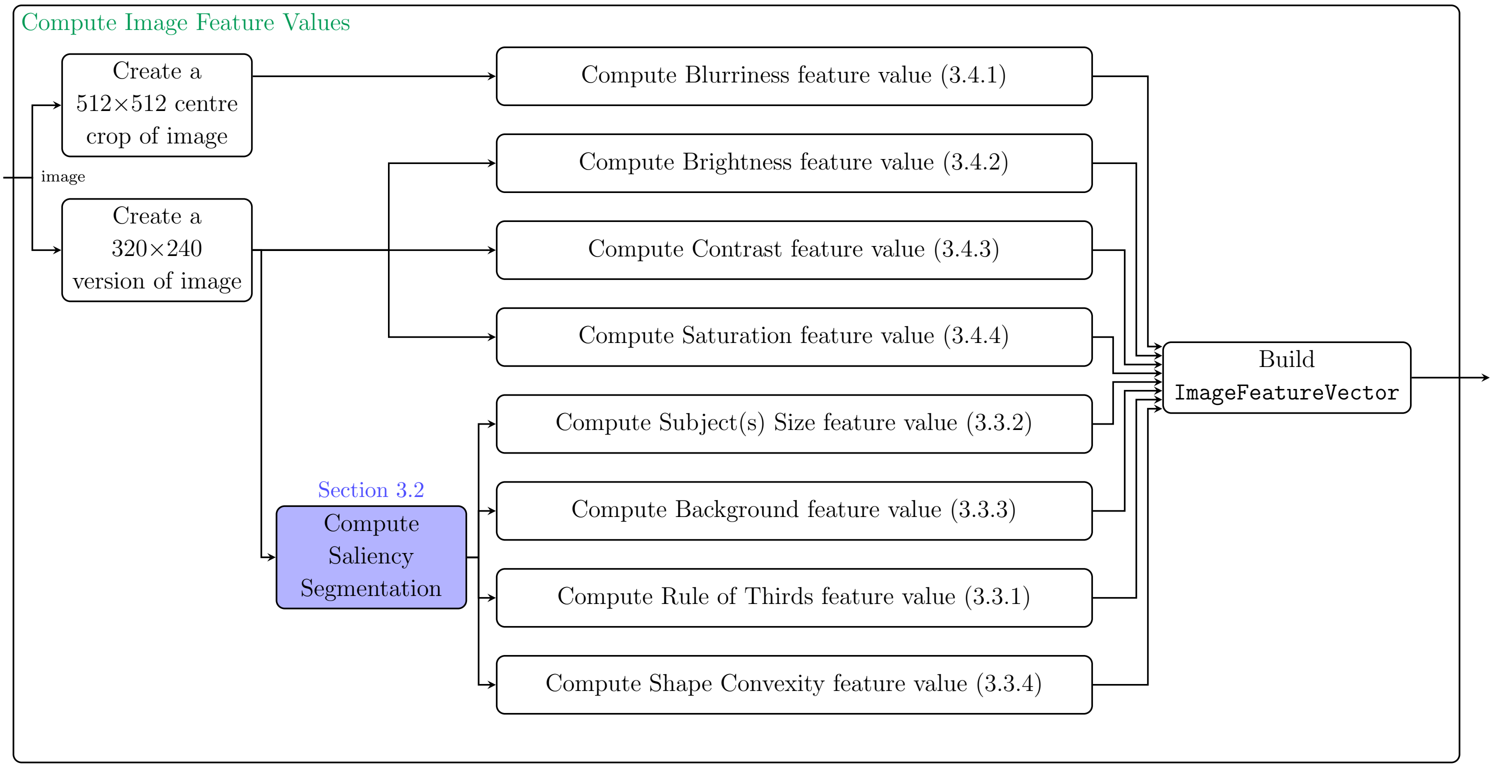

Compute Image Feature Values
image
Create a 512×512 centre crop of image
Create a 320×240 version of image
Section 3.2
Compute Saliency Segmentation
Compute Blurriness feature value (3.4.1)
Compute Brightness feature value (3.4.2)
Compute Contrast feature value (3.4.3)
Compute Saturation feature value (3.4.4)
Compute Subject(s) Size feature value (3.3.2)
Compute Background feature value (3.3.3)
Compute Rule of Thirds feature value (3.3.1)
Compute Shape Convexity feature value (3.3.4)
Build ImageFeatureVector

Figure 3.21: Computing Image Feature Values

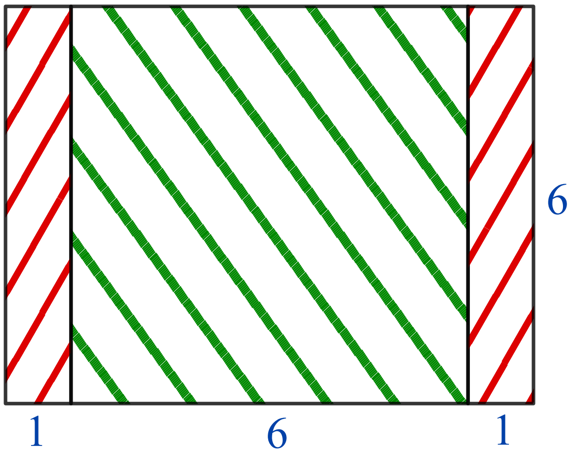


Figure 3.22: I minimise the amount of data cropped by taking a centre-crop. It is most likely that focal objects are near the centre.

The purpose of this module is to compute the 8 individual floating-point feature values, and package them up into an `ImageFeatureVector`.

The program takes input images of the 4:3 aspect ratio. There is a trade-off in image size between feature computation speed and retention of useful information. I chose 320×240 in line with Felzenszwalb & Huttenlocher [9]. However, the blurriness feature, with $O(nlogn)$ time complexity, needed a square, power of 2 input, so I chose 512×512 to avoid aliasing artefacts, shown in Figure 3.22.

In order to implement the features within the Agile framework, I needed a modular architecture that would facilitate the implementation and unit-testing of features in sprints. Therefore, each feature was considered as a separate work item.

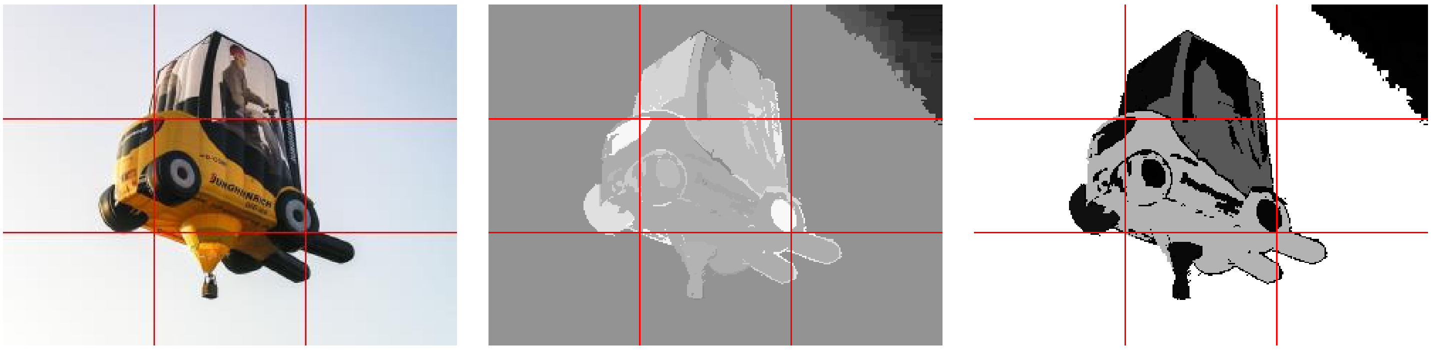
3.3.1 Rule of Thirds

This feature is based on a photography rule-of-thumb: the image is better if the subject(s) are close to any of the 4 intersections of the third lines of the image, *power points*. I compute a weighted sum of the segment distances from power points, weighted by salience and size.

To get an early working version, I implemented Yeh *et al.*'s [22] definition of the feature, before refining it later:

$$f_{\text{ROTO}} = \frac{1}{\sum_i A_i S_i} \sum_i A_i S_i e^{-D_i^2/2\sigma} \tag{3.3}$$

where S_i is salience, A_i is size, D_i is distance from segment centre to the closest *power point*, and σ is a parameter. As given, it favoured large segments too much (see Figure 3.23). Therefore, I tested some variations on the definition, each of which puts less emphasis on large segments in a different way. For each, I ignore segments smaller than 1% of the image:



Original

Power point distance Map

RoT heat-map using f_{ROT0}

Figure 3.23: The heat-map, which illustrates segment contributions to the feature value, shows that the background was incorrectly considered important.

1.

$$f_{\text{ROT1}} = \frac{1}{\sum_i S_i \ln A_i} \sum_i S_i \ln A_i e^{-D_i^2/2\sigma}. \tag{3.4}$$

Large segments are still favoured, but much less because of the logarithm.

2.

$$f_{\text{ROT2}} = \frac{1}{\sum_i S_i} \sum_i S_i e^{-D_i^2/2\sigma}. \tag{3.5}$$

Segment size is ignored entirely.

3.

$$f_{\text{ROT3}} = \frac{1}{\sum_i S_i/V_i} \sum_i \frac{S_i}{V_i} e^{-D_i^2/(2\sigma)} \tag{3.6}$$

where

$$V_i = \max\left(\max_{x_1,x_2 \in C_i}\left(\frac{x_2 - x_1}{Width}\right), \max_{y_1,y_2 \in C_i}\left(\frac{y_2 - y_1}{Height}\right)\right) \tag{3.7}$$

is the segment "spread", illustrated in Figure 3.24.

Figure 3.25 compares the implemented options. I chose f_{ROT3}; using spread instead of number of pixels accommodates for segments that cover a large area of the image, but are fairly thin (for example, a tree with holes between branches). These are still large objects to humans.

Figure 3.26 shows the steps in computing the chosen feature value.

I performed a parameter sweep, similar to that in Section 3.2.4, to choose $\sigma = 0.17$.

The different iterations made up different sprints, each with code reviews at the end for monitoring progress. I employed this strategy for the other features and the rest of the implementation.

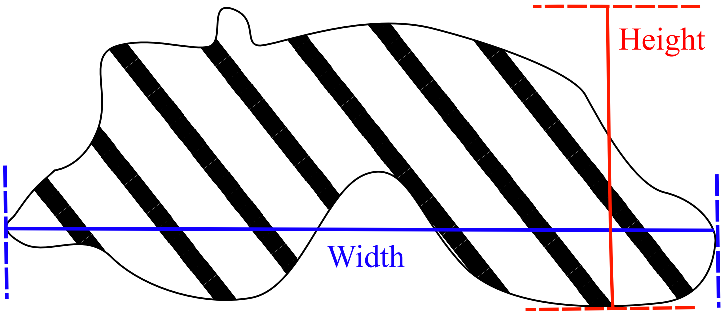


Figure 3.24: Segment spread is the longest of the width or height of the segment's bounding box.

3.3.2 Subject(s) Size

Inspired by Yeh *et al.*'s paper [22], in which it is called "ROI Size", this feature approximates the proportion of the image taken up by subjects.

First, the saliency segmentation is thresholded:

$$B(i) = \begin{cases} 1, & \text{if } S_i > \alpha \\ 0, & \text{otherwise} \end{cases} \tag{3.8}$$

where S_i are the normalised segment saliencies. The bounding box for each '1' segment in B is computed, and the feature value is the proportion of the image taken up by these bounding boxes. In other words, it is the rough proportion of the image taken up by subjects. Figure 3.27 illustrates the steps.

I performed a parameter sweep, similar to that in Section 3.2.4, to determine $\alpha = 0.73$. This is different from the value of $\alpha_{\text{Yeh}} = 0.67$ used by Yeh *et al.*

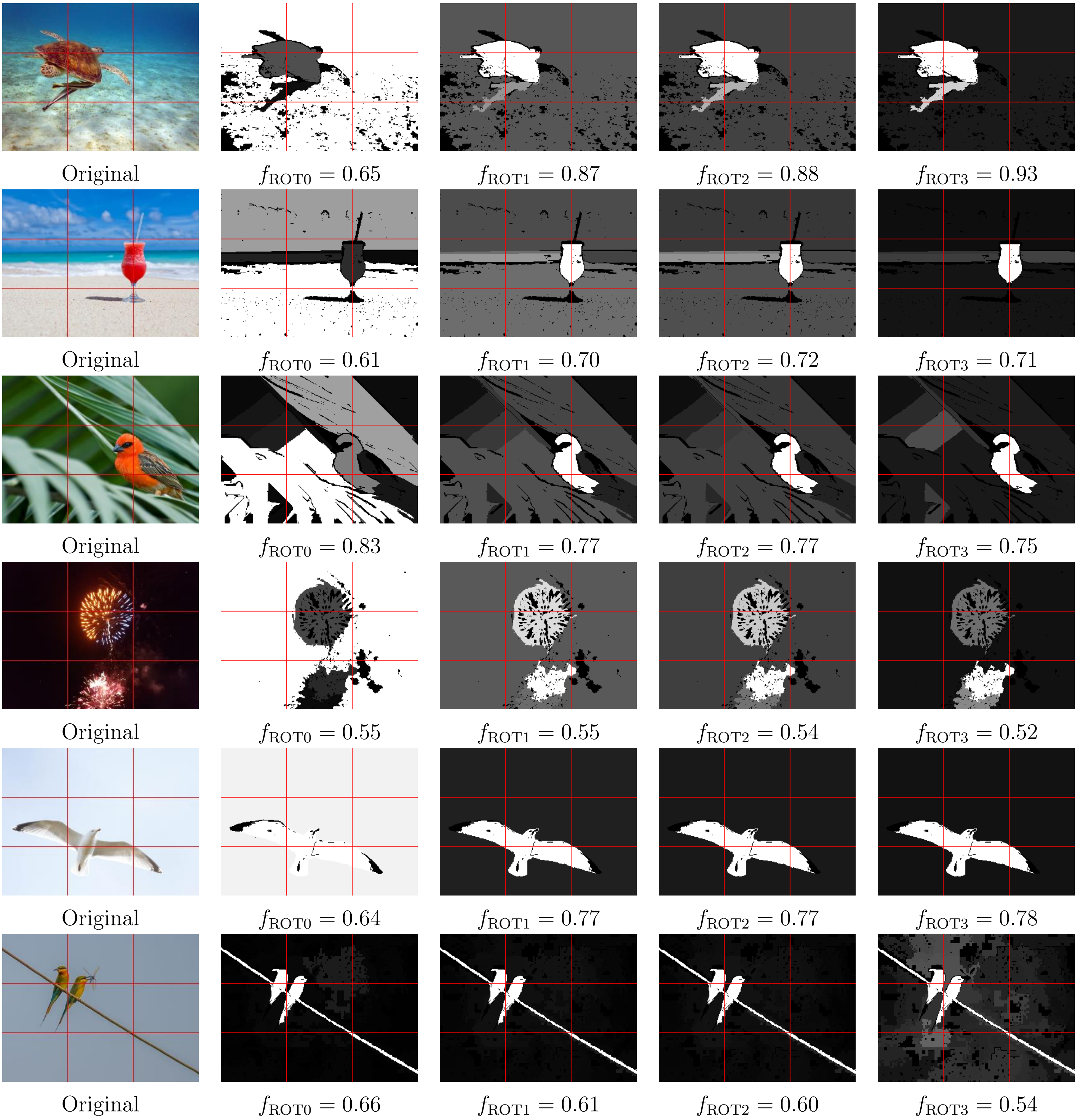
3.3.3 Background Distraction

This feature is from Luo *et al.*'s paper [15], in which it is called "Simplicity". It quantifies how "distracting" the background is. Luo *et al.* said that a distracting background contains many different colours.

First, the bounding box map from *Subject(s) Size* is inverted, leaving the "background". The remaining pixels are quantised to a bit-depth of 12 (rather than 24). Figure 3.28 shows all 4096 possible colours after quantisation.

The quantised background is used to populate a height-normalised histogram which has a bin for each colour. The feature value is obtained by computing the proportion of bins with heights above 0.01. This is an approximation of the background's colour variation. Bins lower than 0.01 are ignored to reduce the effects of noise.

From testing, I found the feature value only gets up to around 0.1. Therefore, I added an approximate normalisation factor of $\frac{1}{0.11}$.



Original $f_{\mathrm{ROT0}} = 0.65$ $f_{\mathrm{ROT1}} = 0.87$ $f_{\mathrm{ROT2}} = 0.88$ $f_{\mathrm{ROT3}} = 0.93$

Original $f_{\mathrm{ROT0}} = 0.61$ $f_{\mathrm{ROT1}} = 0.70$ $f_{\mathrm{ROT2}} = 0.72$ $f_{\mathrm{ROT3}} = 0.71$

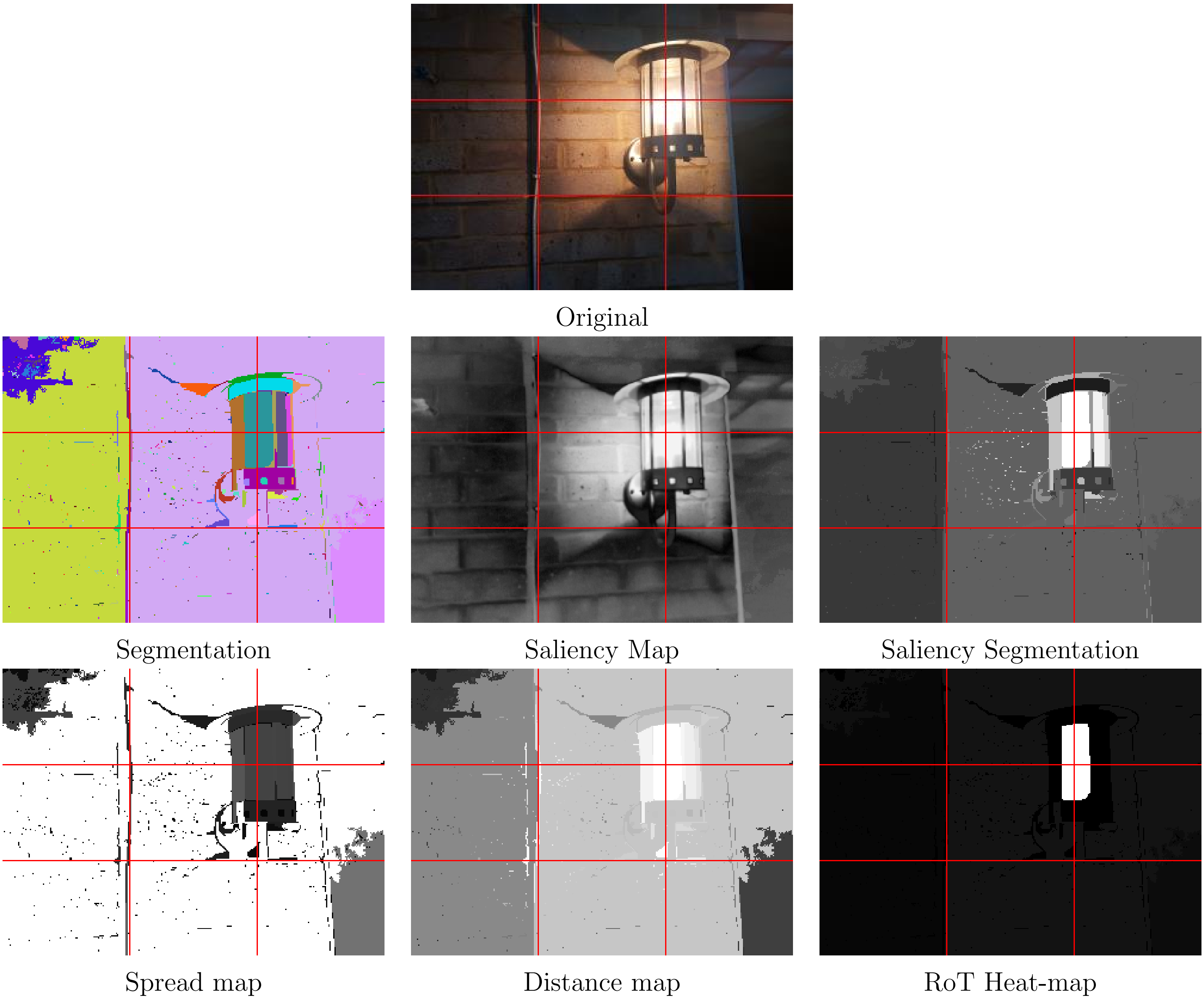
Original $f_{\mathrm{ROT0}} = 0.83$ $f_{\mathrm{ROT1}} = 0.77$ $f_{\mathrm{ROT2}} = 0.77$ $f_{\mathrm{ROT3}} = 0.75$

Original $f_{\mathrm{ROT0}} = 0.55$ $f_{\mathrm{ROT1}} = 0.55$ $f_{\mathrm{ROT2}} = 0.54$ $f_{\mathrm{ROT3}} = 0.52$

Original $f_{\mathrm{ROT0}} = 0.64$ $f_{\mathrm{ROT1}} = 0.77$ $f_{\mathrm{ROT2}} = 0.77$ $f_{\mathrm{ROT3}} = 0.78$

Original $f_{\mathrm{ROT0}} = 0.66$ $f_{\mathrm{ROT1}} = 0.61$ $f_{\mathrm{ROT2}} = 0.60$ $f_{\mathrm{ROT3}} = 0.54$

Figure 3.25: Heat-maps and feature values for the implemented options. In general, the chosen version (f_{ROT3}) is the best, because it correctly identifies where the subject is. However, it does not work so well on the last picture, whose 'subject' is thin and diagonal, giving it a large spread.



Original

Segmentation

Saliency Map

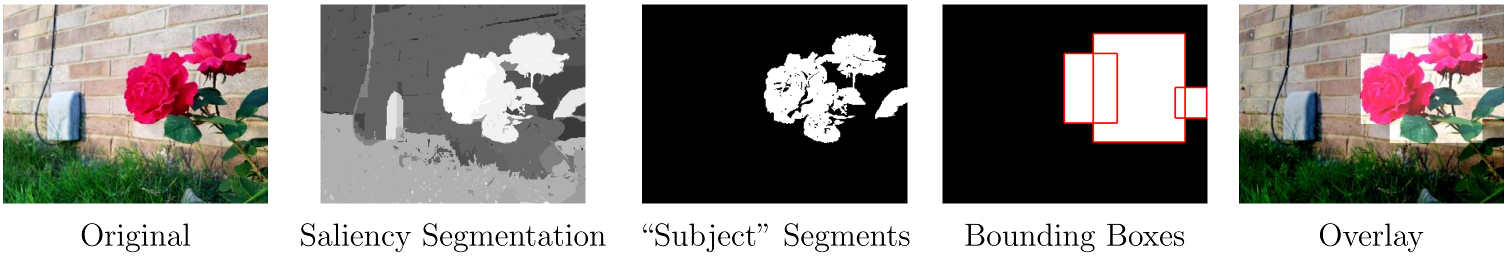
Saliency Segmentation

Spread map

Distance map

RoT Heat-map

Figure 3.26: $f_{\mathrm{ROT}} = 0.73$. The segments which contribute most to the feature value are those which are bright in the saliency segmentation, bright in the distance map, but dark in the spread map.



Original

Saliency Segmentation

“Subject” Segments

Bounding Boxes

Overlay

Figure 3.27: $f_{\mathrm{SubjectSize}} = 0.24$.

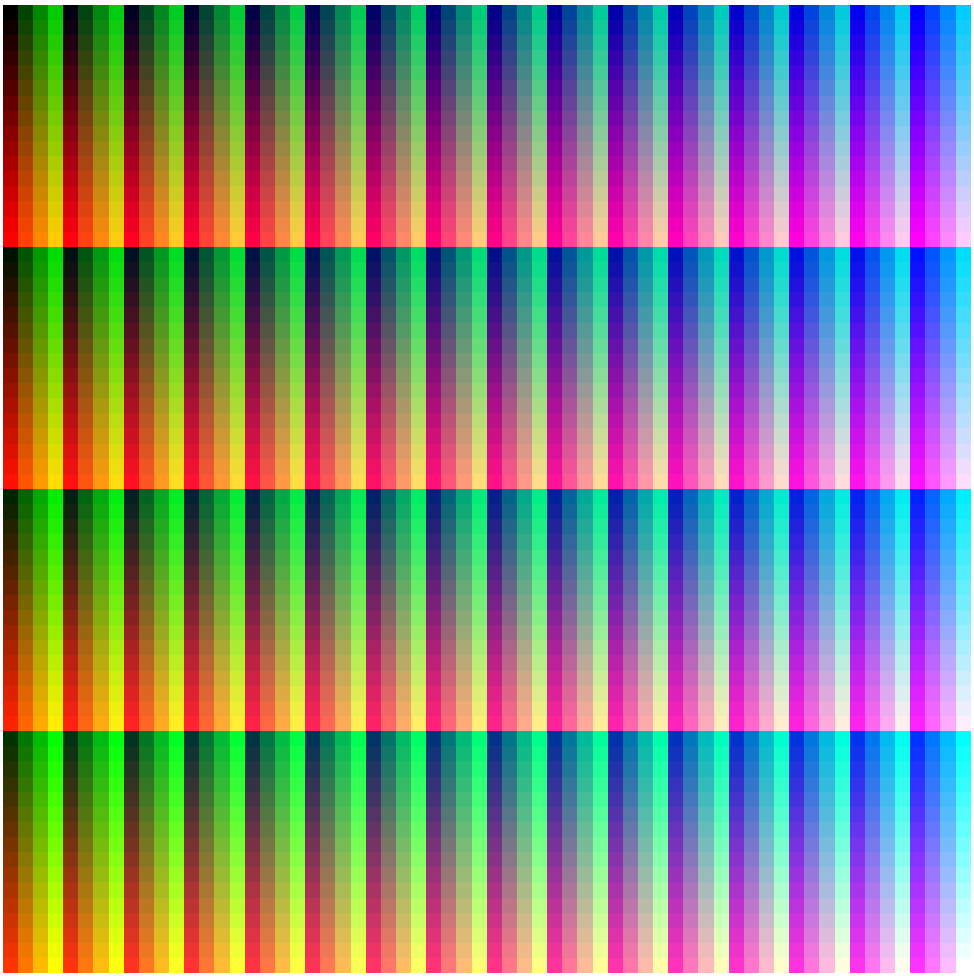
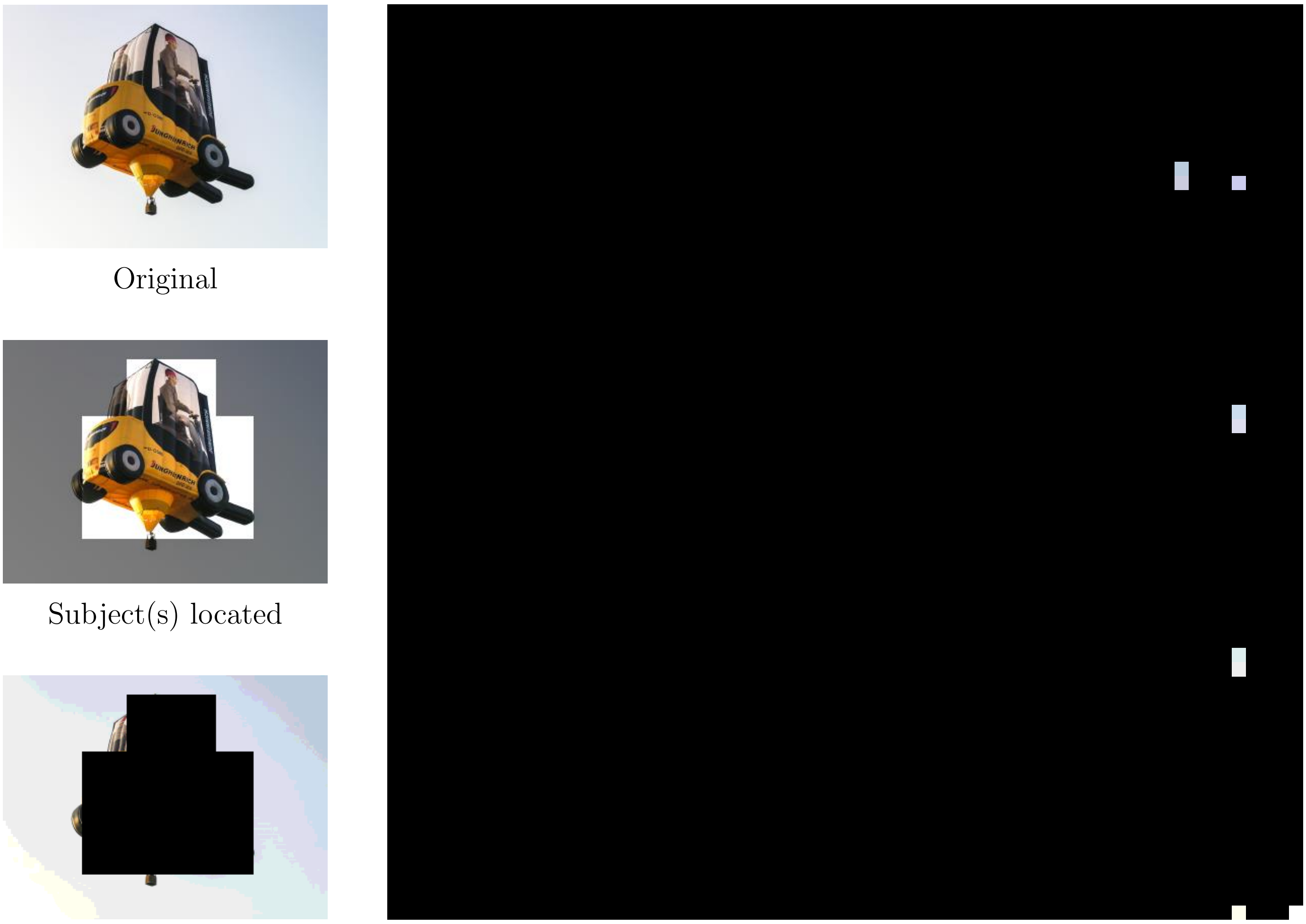


Figure 3.28: 4096 possible colors after the quantisation step



Original

Subject(s) located

Quantised Background

Colour histogram (9 noticeable colours present), $f_{\text{BackDistract}} = 0.02$

Figure 3.29: Only 9 colours are present in the quantised background. Note that this diagram shows a subset of the full 4096-colour histogram in Figure 3.28.

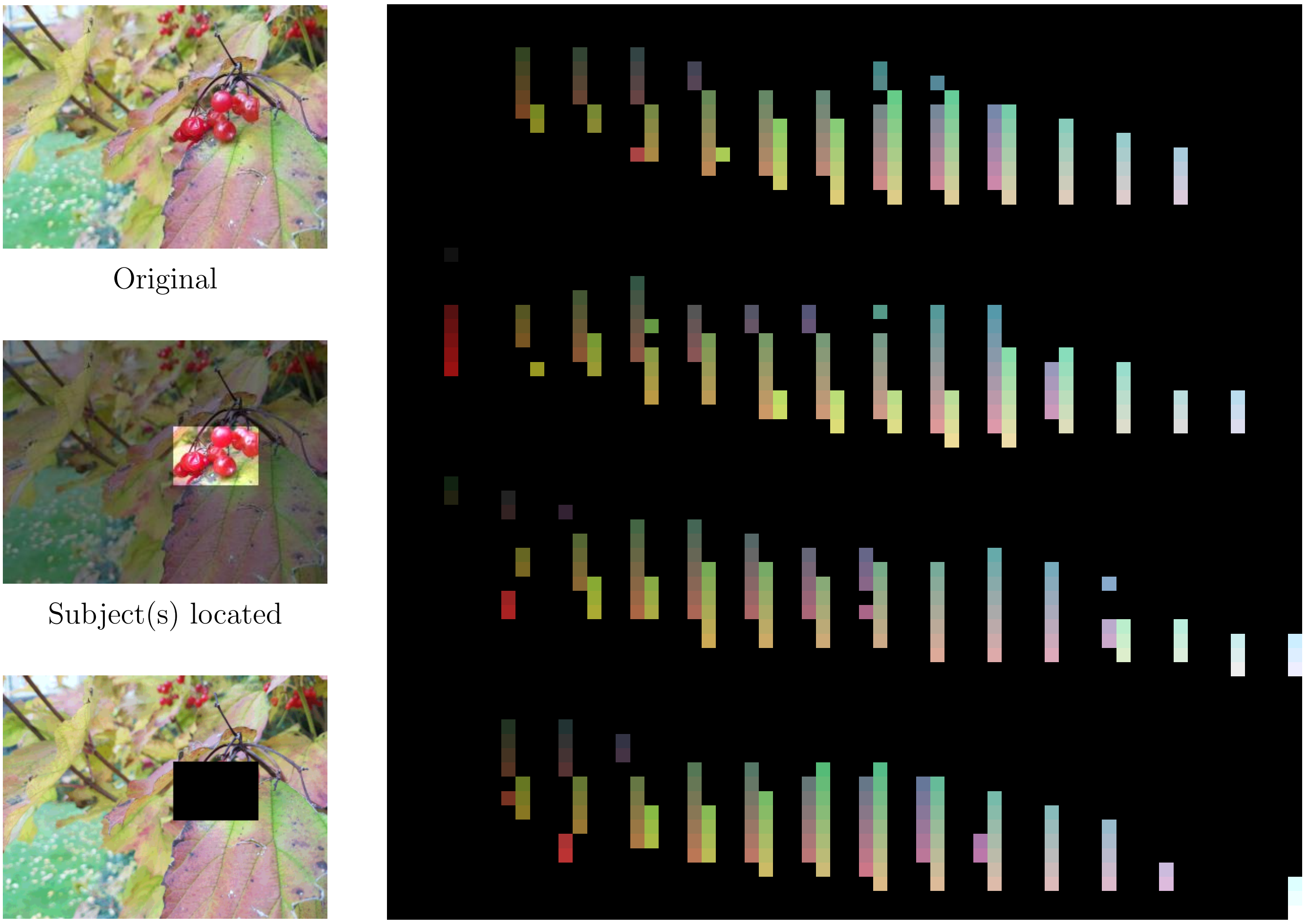
Figures 3.29 and 3.30 give two examples of this feature's extraction.

3.3.4 Shape Convexity

This feature proposes modifications to address several flaws in Datta *et al.*'s Shape Convexity [8]. Their feature identifies the segments that fill at least 80% of their convex hulls (*sufficiently convex* segments) and counts their areas:

$$f_{\text{DattaConvex}} = \frac{\sum \textit{sufficiently convex segment size}}{\sum \textit{segment size}}. \tag{3.9}$$

A problem with this is that it considers all segments equally. There is no point in having convex regions that peoples' eyes are not attracted to. Given my saliency segmentation, I refined it to only consider "subjects" (segments above 70% saliency). So, the feature



Original

Subject(s) located

Quantised Background

Colour histogram (461 noticeable colours present), $f_{\text{BackDistract}} = 1.02$

Figure 3.30: An image with a distracting background

value should be of the form

$$\frac{\sum_{\text{subjects}} \textit{sufficiently convex segment size}}{\sum_{\text{subjects}} \textit{segment size}}. \tag{3.10}$$

In Section 2.2.4, I say that people tend to prefer circular, curvilinear objects to those with jagged corners. Therefore, the second modification rewards circular convex shapes. It does this using the fact that the convex polygons of more circular shapes generally consist of more points. So, if n is the average number of hull points in *sufficiently convex* subjects, I propose the feature value to be

$$f_{\text{Convex}} = \underbrace{\frac{\sum_{\text{subjects}} \textit{sufficiently convex segment size}}{\sum_{\text{subjects}} \textit{segment size}}}_{\text{How much convexity there is in the salient parts of the image}} \times \underbrace{\left(1 - e^{-\beta(n-3)}\right)}_{\text{The average degree of convexity (circularity)}} \tag{3.11}$$

where $\beta > 0$ is a parameter. $(1 - e^{-\beta(n-3)})$ is a function that ranges from 0 and tends towards 1 as $n \to \infty$. The term $(n-3)$ is used, because 3 is the smallest possible number of points in a convex polygon, a triangle. I chose $\beta = 0.11$ using empirical results on some example images.

I used the `MIConvexHull` library[3] for computing convex hulls.

The computation of this feature is illustrated in Figure 3.31.

3.4 Computing Low-level Features

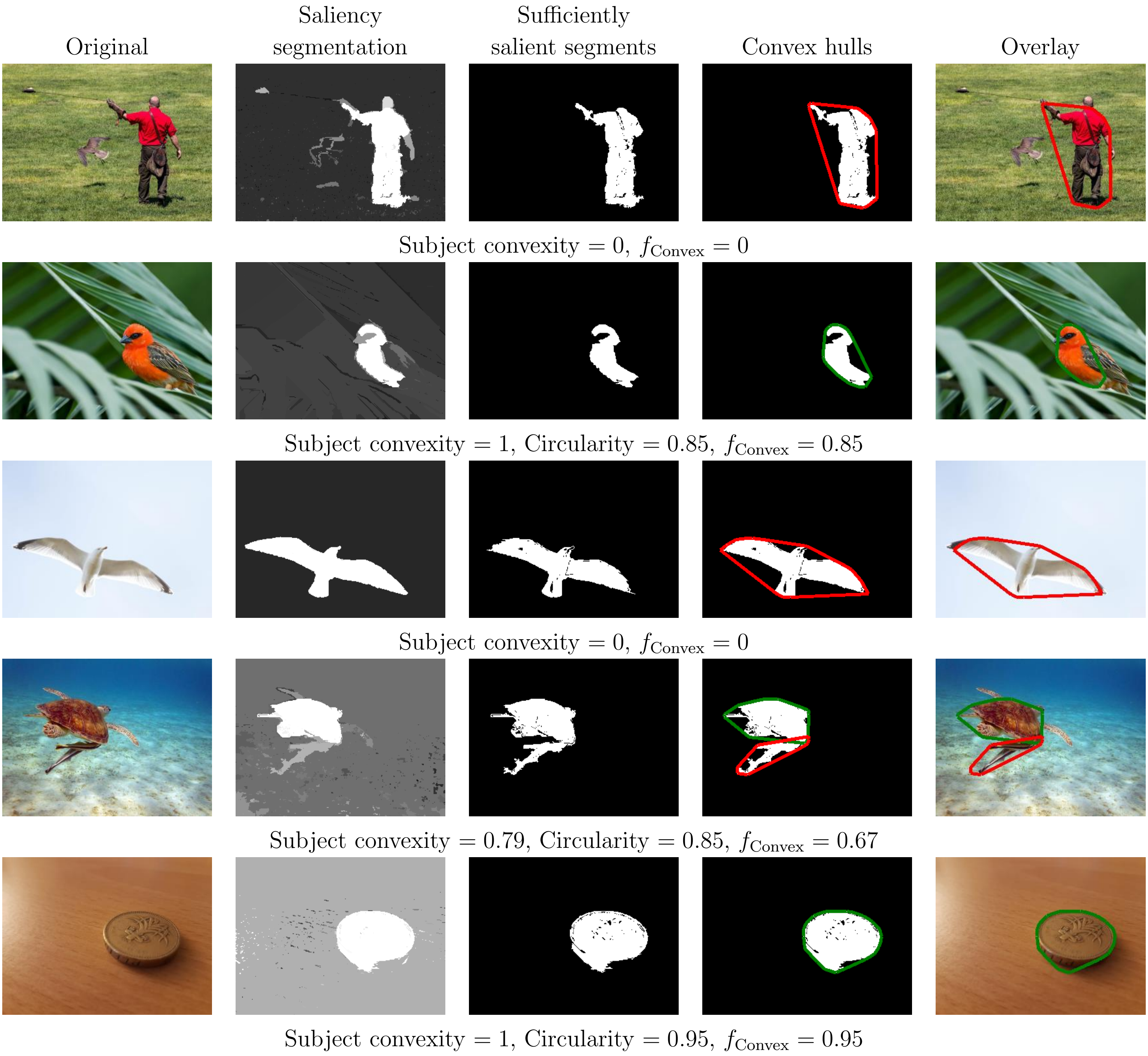
3.4.1 Blurriness

The 2D Discrete Fourier Transform (2DFT) is used to quantify how blurry the image is as a whole.

A *magnitude plot* illustrates the Fourier coefficients. Each pixel's intensity represents the modulus of a coefficient for a different two-dimensional complex exponential: a plane wave (see Figure 3.32). The original image can be represented as a superposition of these plane waves, weighted by the coefficients. In non-blurry images, higher frequencies are required to create sharp changes in pixel intensity. In blurry images, the lower frequency waves are sufficient to represent the image, so there is little activity in the outer regions of the plot. Since images are real-valued input arrays, symmetry can be exploited to only require analysis of one half of the plot.

The feature value is defined as the average of the coefficients in the highest 90% of frequencies in the right half of the plot (see Figure 3.33). In sharp images, there are more

[3] `https://github.com/DesignEngrLab/MIConvexHull`



Subject convexity $= 0$, $f_{\text{Convex}} = 0$

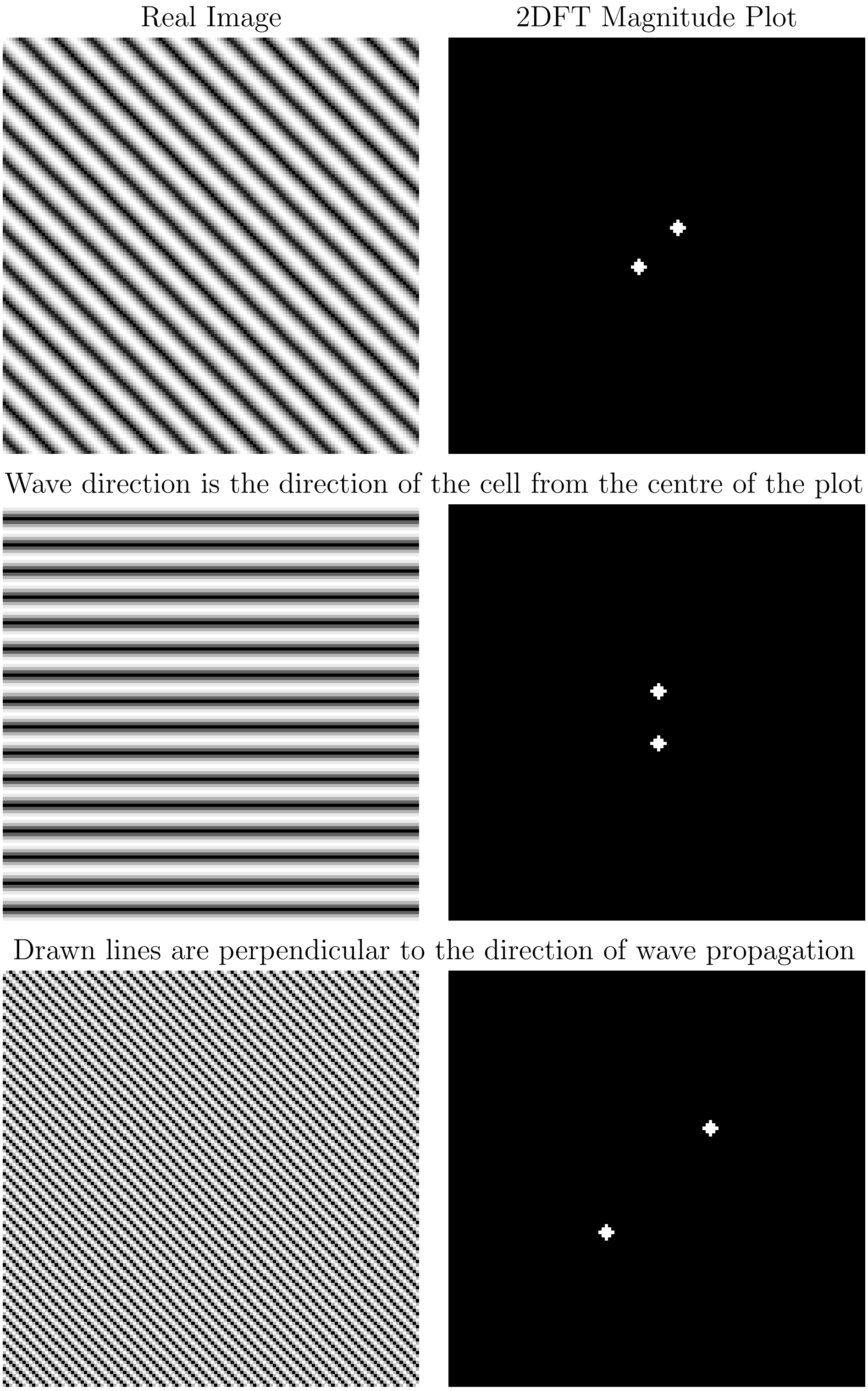
Subject convexity $= 1$, Circularity $= 0.85$, $f_{\text{Convex}} = 0.85$

Subject convexity $= 0$, $f_{\text{Convex}} = 0$

Subject convexity $= 0.79$, Circularity $= 0.85$, $f_{\text{Convex}} = 0.67$

Subject convexity $= 1$, Circularity $= 0.95$, $f_{\text{Convex}} = 0.95$

Figure 3.31: *Sufficiently convex* subjects have green convex hulls. Other subjects have red hulls.



Real Image

2DFT Magnitude Plot

Wave direction is the direction of the cell from the centre of the plot

Drawn lines are perpendicular to the direction of wave propagation

Wave frequency is the distance of the cell from the centre of the plot

Figure 3.32: 2DFT plane waves

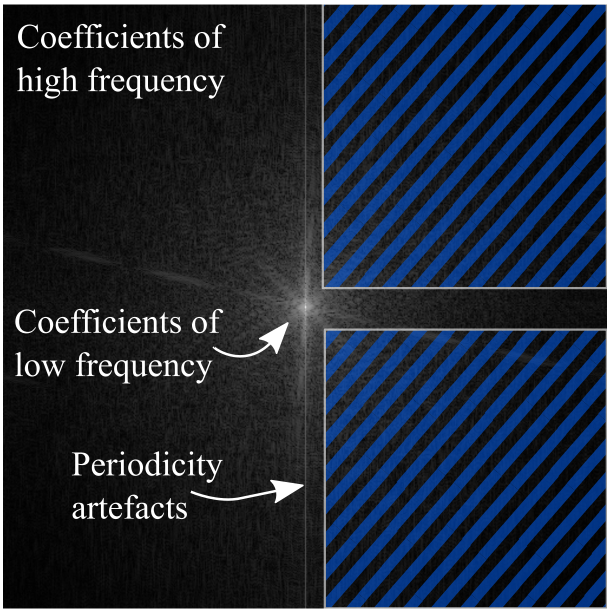


Figure 3.33: The feature value is the average of the cells in the blue-hashed region of the magnitude plot.

bright cells, resulting in a higher feature value (and vice versa for blurry images). I finally added a normalising factor of 5.5.

The reason for discarding the lowest 10% of frequencies is due to *periodicity artefacts* in the plot. The 2DFT assumes that the finite image is infinitely repeated in space, as shown in Figure 3.34. As a result, many images' plots have strong vertical and/or horizontal lines corresponding to lines between the image and its repetitions, *not* in the image itself. In general, a straight line on the plot means contribution from many plane waves with the same direction, but differing frequencies. In the image, this corresponds to sharp, straight, perpendicular lines. It is important to note that the image positions of these lines are lost in the magnitude plot: they are captured by coefficient *phase*.

Figures 3.35–3.37 give examples of the 2DFT for a few images. Figure 3.35 shows a sharp image, where there are bright cells far away from the centre. To contrast, Figure 3.37 shows a blurry image, whose 2DFT is constrained more closely to the centre.

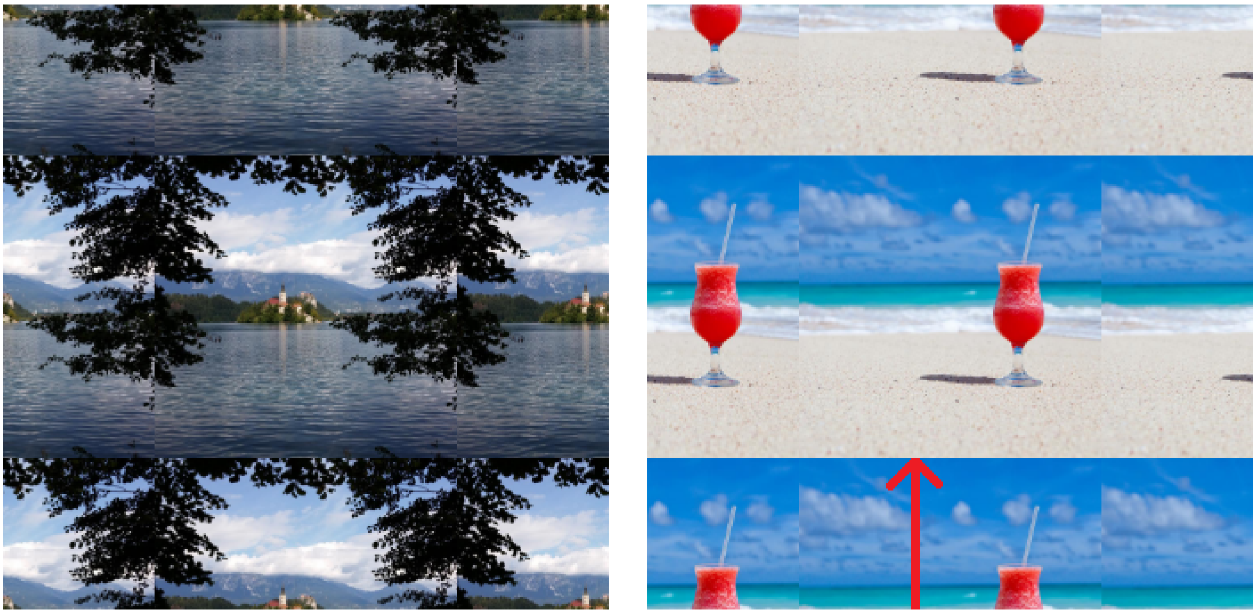
For the 2FFT I used an existing Cooley-Tukey implementation[4].

3.4.2 Brightness

This ostensibly simple feature converts the image to greyscale, and computes the average pixel intensity:

$$f_{\text{Bright}} = \frac{\sum_x \sum_y I(x,y)}{Width \times Height} \tag{3.12}$$

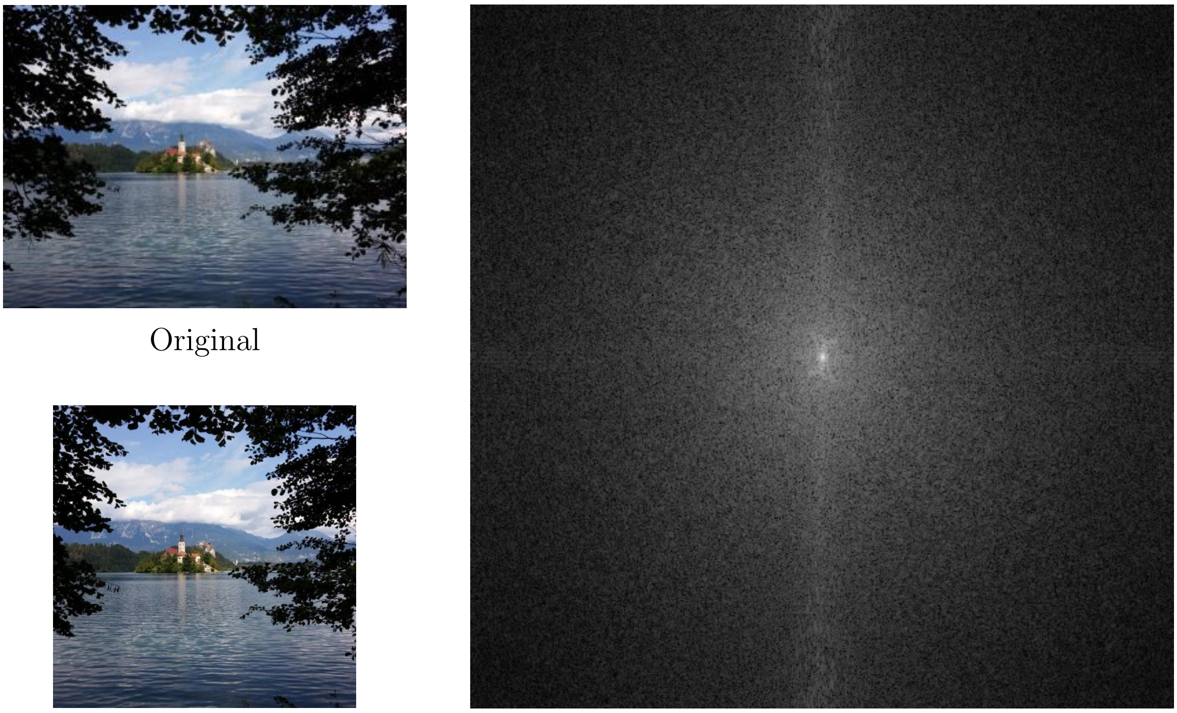
[4] `https://www.codeproject.com/kb/gdi/fft.aspx`



From Figure 3.35

From Figure 3.36 – Horizontal discontinuities lead to vertical artefacts in the 2DFT

Figure 3.34: Assumed 2DFT periodicity

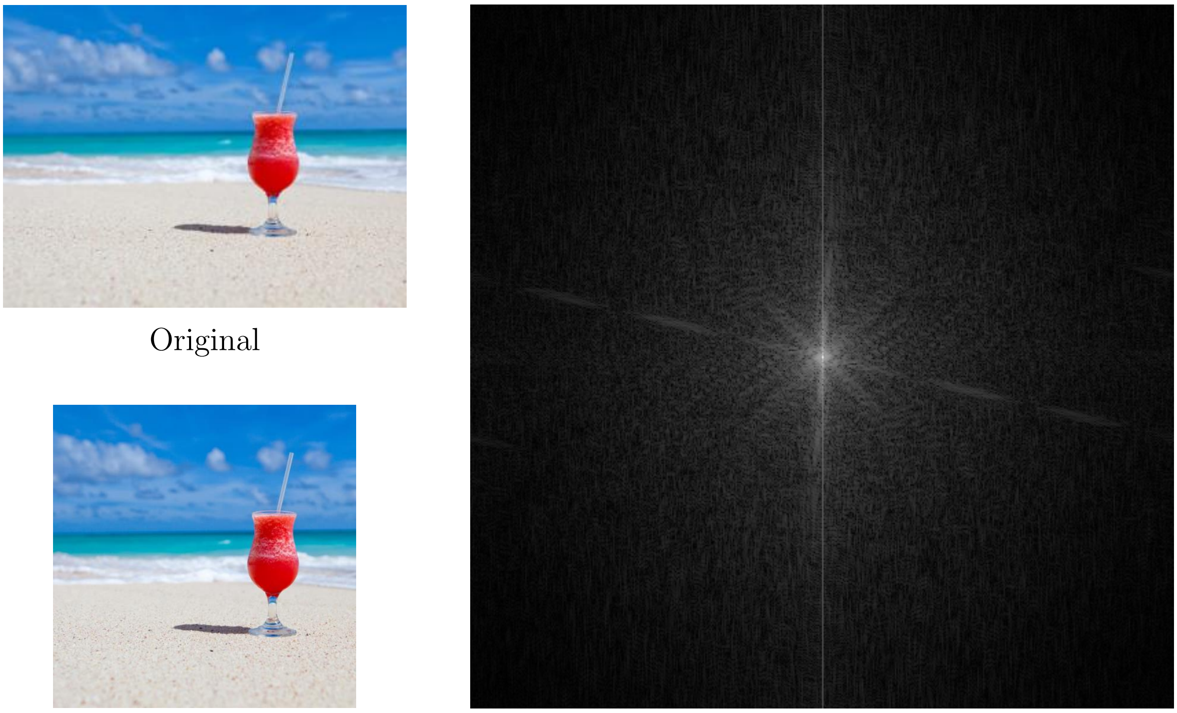


Original

Centre crop

2D Discrete Fourier Transform (Magnitude plot)

Figure 3.35: $f_{\text{blur}} = 0.92$

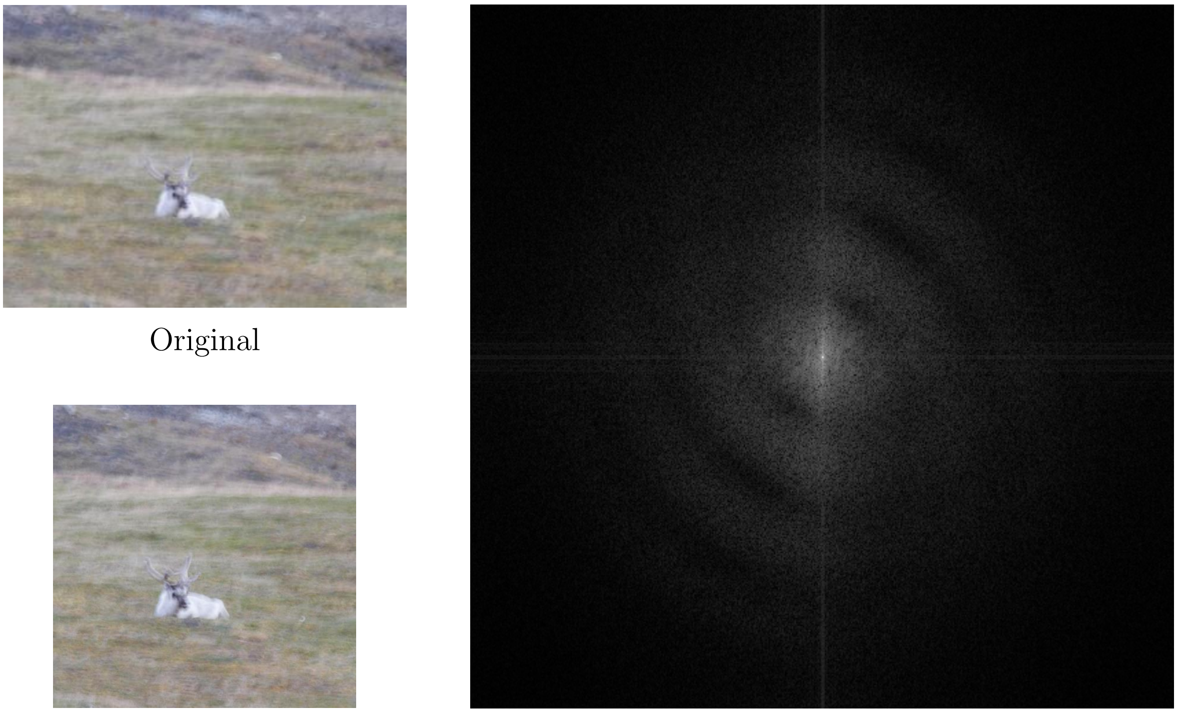


Original

Centre crop

2D Discrete Fourier Transform (Magnitude plot)

Figure 3.36: $f_{\text{blur}} = 0.40$

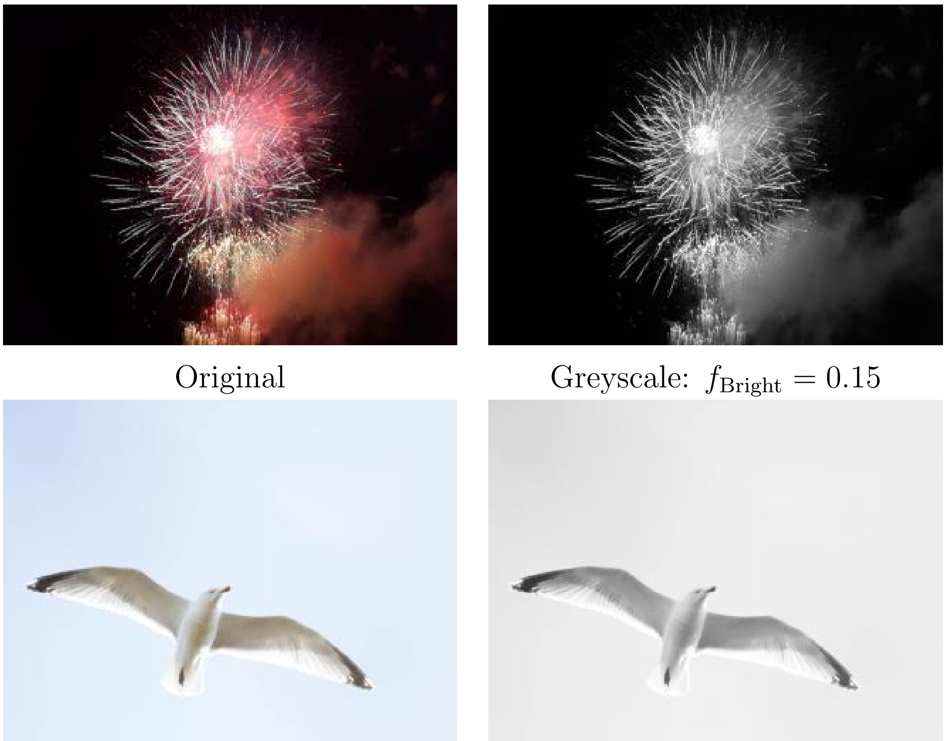


Original

Centre crop

2D Discrete Fourier Transform (Magnitude plot)

Figure 3.37: $f_{\text{blur}} = 0.29$



Original

Greyscale: $f_{\text{Bright}} = 0.15$

Original

Greyscale: $f_{\text{Bright}} = 0.90$

Figure 3.38: Converting to greyscale

Care must be taken with greyscale conversion. Naïvely computing the average of the RGB components would produce a convincing black-and-white image but it would not be perceptually accurate, since the human retina has different sensitivities to different colours. It seems unclear which conversion method is "best", but generally people perceive green as brightest, then red, and blue as dimmest. My program uses *relative luminance*[5], since it is concerned with comparing intensities rather than seeking a physical energy value:

$$I(x, y) = 0.2126 \times r + 0.7152 \times g + 0.0722 \times b \tag{3.13}$$

Figure 3.38 shows two examples of the greyscale image produced.

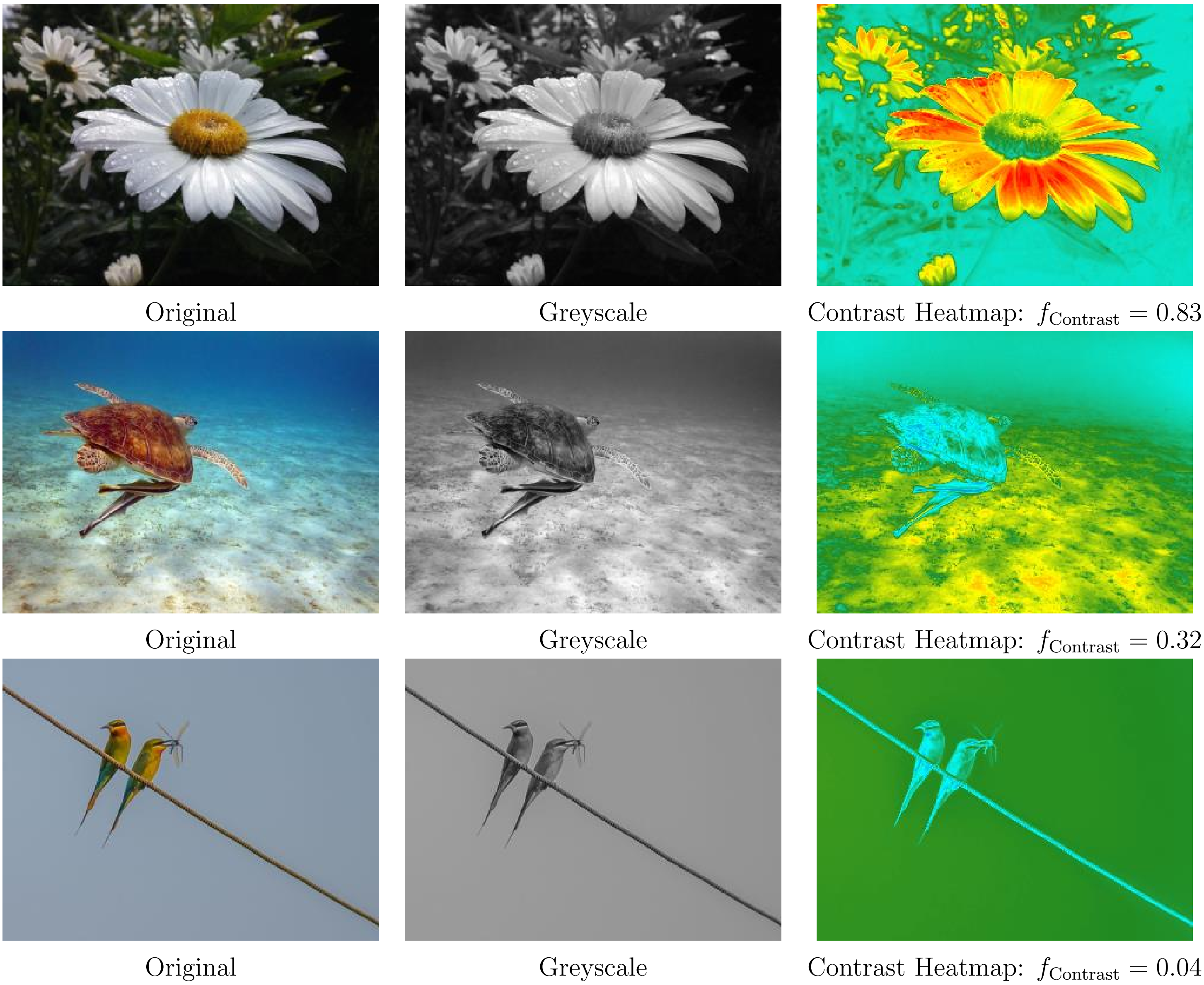
3.4.3 Intensity Contrast

High-contrast images are often thought to be better because they make use all possible intensity values, and draw attention to subjects. Known as *Weber Contrast*, this feature computes the average absolute difference from the mean intensity of the image:

$$f_{\text{Contrast}} = \frac{1}{\textit{Width} \times \textit{Height}} \sum_{x} \sum_{y} \frac{|I(x, y) - I_{\text{av}}|}{I_{\text{av}}} \tag{3.14}$$

where I_{av} is the average intensity of the image, and $I(x, y)$ is the *relative luminance*, defined in Equation (3.13).

[5] `https://www.w3.org/Graphics/Color/sRGB` (Figure 1.8)



Original

Greyscale

Contrast Heatmap: $f_{\text{Contrast}} = 0.83$

Original

Greyscale

Contrast Heatmap: $f_{\text{Contrast}} = 0.32$

Original

Greyscale

Contrast Heatmap: $f_{\text{Contrast}} = 0.04$

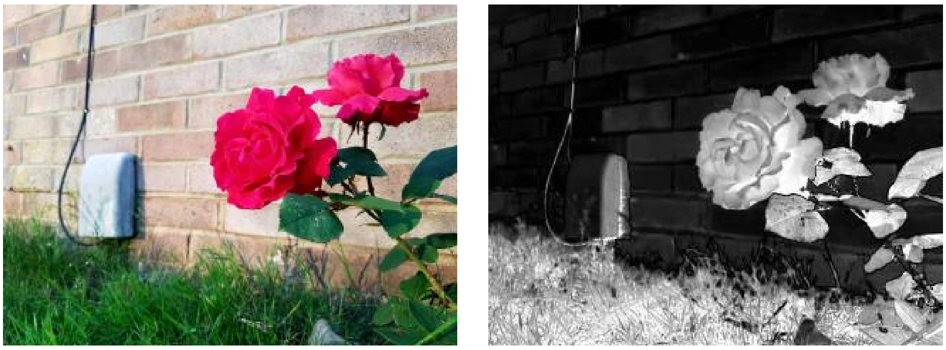
Figure 3.39: In the heatmaps, red pixels are much brighter than the average, blue are much darker, and green are close to average.

This is mostly the same as Yeh *et al.*'s definition [22]. However, they do not use the absolute difference, so just have

$$f_{\text{YehContrast}} = \frac{1}{Width \times Height} \sum_{x} \sum_{y} \frac{I(x,y) - I_{\text{av}}}{I_{\text{av}}}. \tag{3.15}$$

This would not work, because it is always equal to 0.

There is a final normalisation factor of $\frac{1}{1.3}$. Figure 3.39 compares intensity heatmaps.



Original

Saturation Heatmap: $f_{\text{Sat}} = 0.31$

Figure 3.40: Computing Saturation

3.4.4 Saturation

The final low-level feature computes the average pixel *saturation*, which is a measure of colourfulness. Colour is important for aesthetic image quality (it is used as the basis for the segmentation algorithm), so it makes sense to have a feature extracting it. There are many definitions of saturation, but my program uses the original 1978 definition from when Joblove & Greenberg [11] introduced the HSV colour space:

$$\text{Sat}(x, y) = \frac{\max(r_{x,y}, g_{x,y}, b_{x,y}) - \min(r_{x,y}, g_{x,y}, b_{x,y})}{\max(r_{x,y}, g_{x,y}, b_{x,y})} \tag{3.16}$$

I used HSV because it performs well despite its simplicity. Figure 3.40 gives an example.

3.5 User Interface

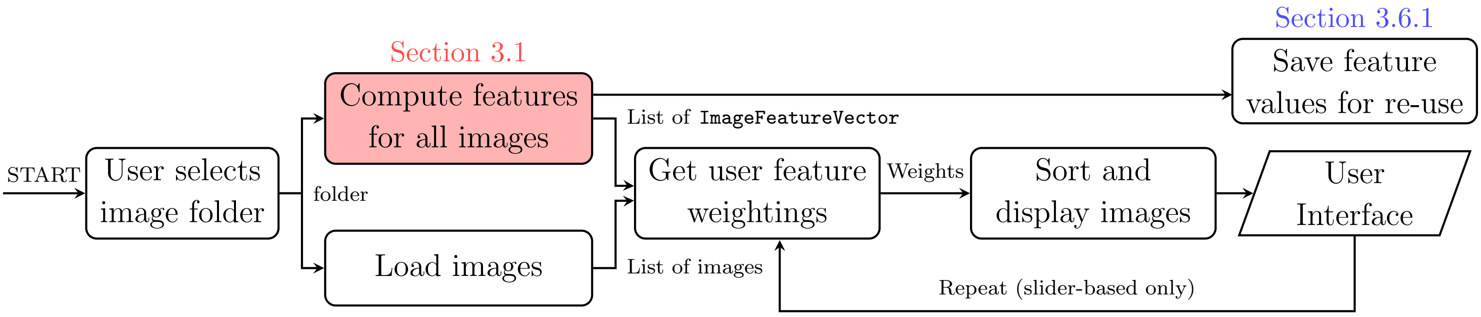
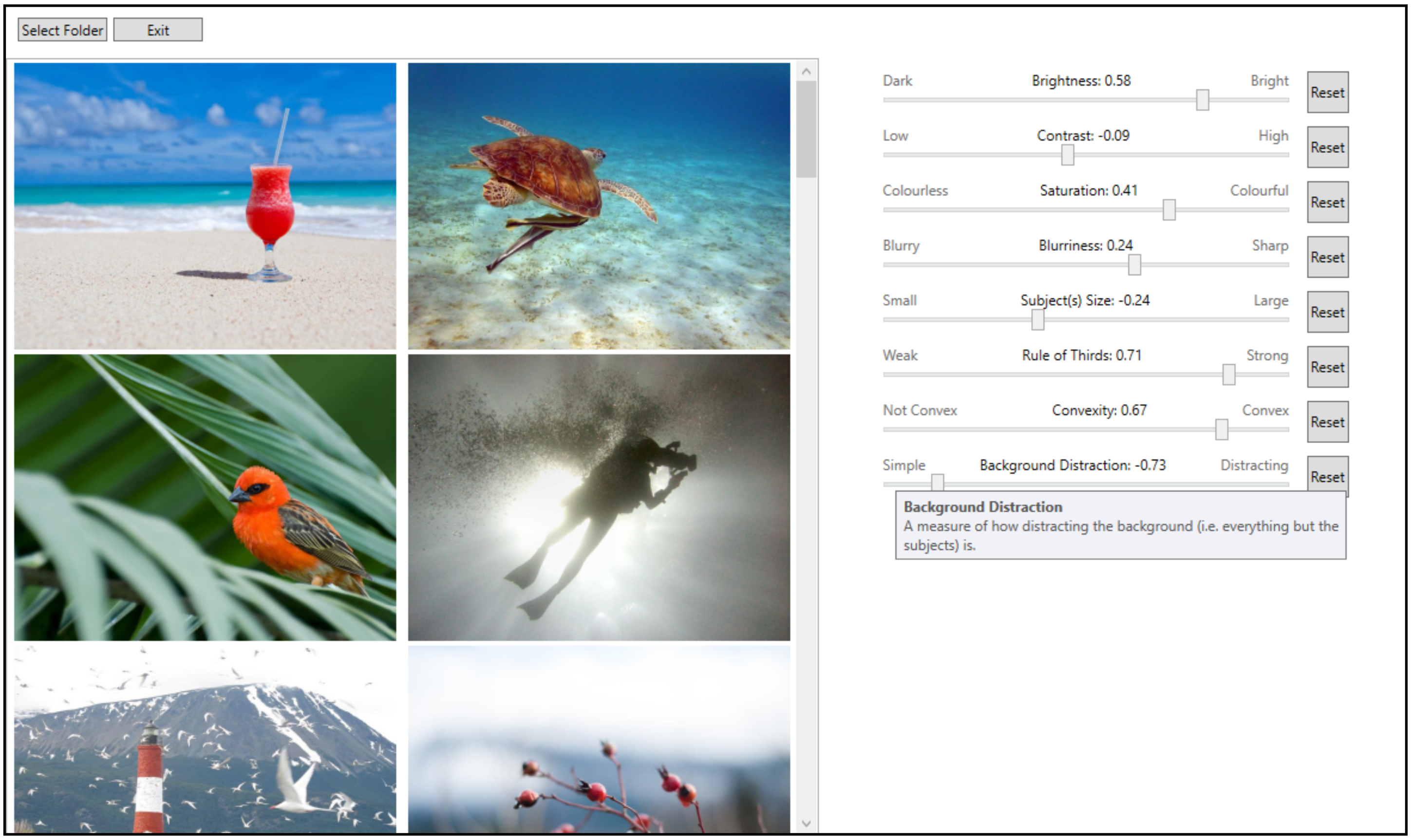

Section 3.1
Section 3.6.1
START
User selects image folder
folder
Compute features for all images
Load images
List of ImageFeatureVector
List of images
Get user feature weightings
Weights
Sort and display images
User Interface
Save feature values for re-use
Repeat (slider-based only)

Figure 3.41: User Interface

The purpose of the GUI is to obtain a weight for each feature from the user, score and rank the image set using the weighted combination of computed feature values, and display the sorted images. It was necessary for the project's human evaluation. In Human Computer Interaction (HCI) theory, there is a trade-off between *Intuitive* (easy to use, learn and remember) interfaces, and *Efficient* (powerful and flexible) interfaces[6]. Therefore, I made two interfaces, with the aim of comparing them in the evaluation.

[6] Part II HCI course:
`http://www.cl.cam.ac.uk/teaching/1617/HCI/HCI-notes-michaelmas2016.pdf`



Dark Brightness: 0.58 Bright

Low Contrast: -0.09 High

Colourless Saturation: 0.41 Colourful

Blurry Blurriness: 0.24 Sharp

Small Subject(s) Size: -0.24 Large

Weak Rule of Thirds: 0.71 Strong

Not Convex Convexity: 0.67 Convex

Simple Background Distraction: -0.73 Distracting

Background Distraction
A measure of how distracting the background (i.e. everything but the subjects) is.

Figure 3.42: Efficient Interface

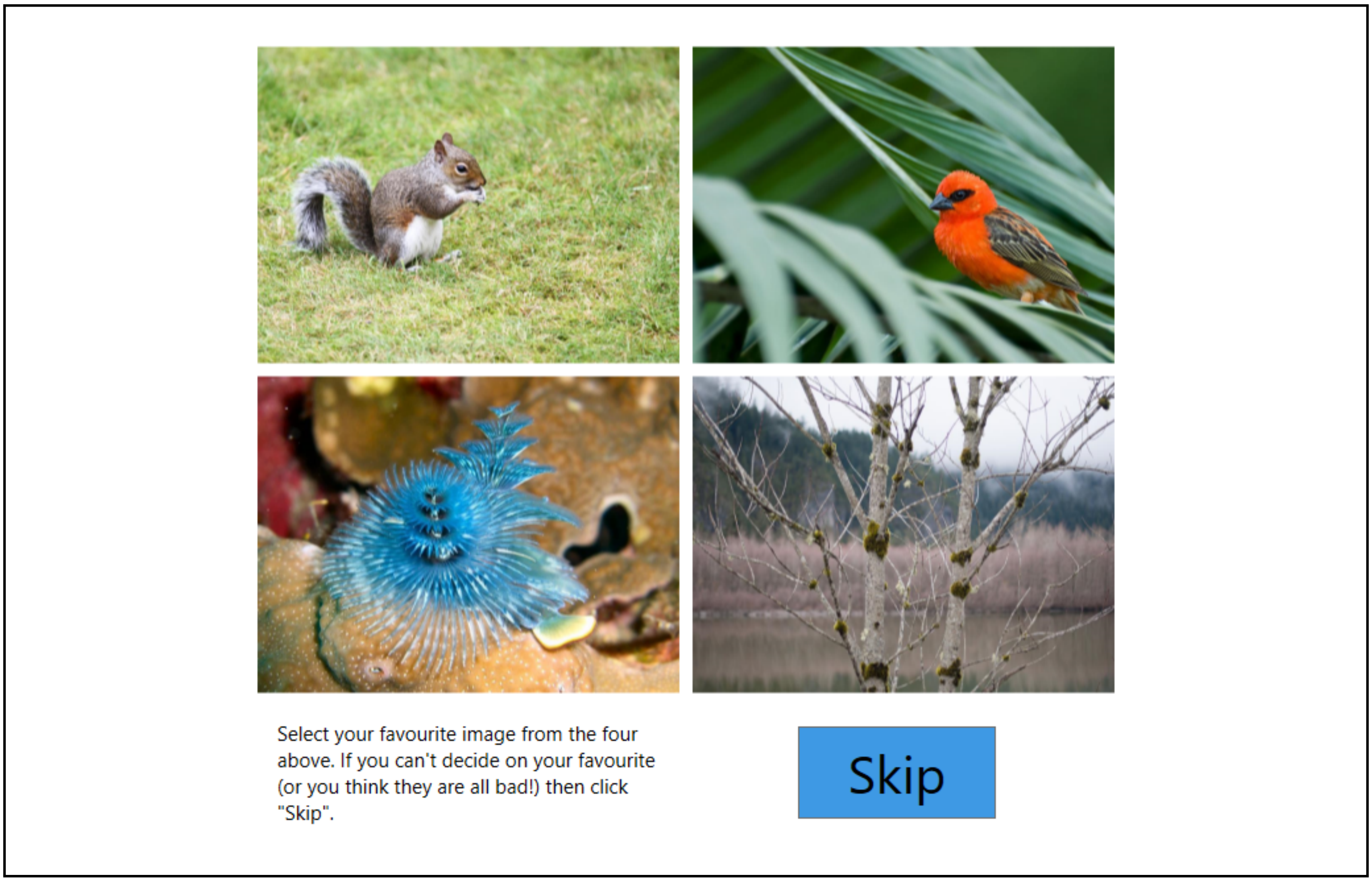
3.5.1 Efficient Interface

Although it is harder to use, and takes time to learn, the *efficient* interface allows the user greater flexibility by manually choosing the feature weights using sliders. The images are re-sorted in real-time as the sliders are moved. This instant user feedback helps them to learn exactly how their actions affect the result. The sliders themselves have helper prompts at either end, and mouse-hover tooltips for extra information. Figure 3.42 shows a screenshot of this UI.

3.5.2 Intuitive Interface

For the *intuitive* interface, the features and weights are not exposed to the user. Instead, they are inferred from a sequence of picture-based questions. Each question asks the user to compare 4 images (see Figure 3.43). The weights for each feature are calculated using the average of the feature values of the chosen images. Initially, I used only these averages for the weights. However, since different features' values are distributed differently, I modified to use the following equation to compensate:

$$w = \begin{cases} \frac{\overline{x}-\mu}{\mu-a} & \text{if } \overline{x} < \mu \\ \frac{\overline{x}-\mu}{b-\mu} & \text{if } \overline{x} > \mu \end{cases} \tag{3.17}$$



Select your favourite image from the four above. If you can't decide on your favourite (or you think they are all bad!) then click "Skip".

Figure 3.43: Intuitive Interface

where $\overline{x}$ is the average feature value of selected images, μ is the average feature value for all images, and a and b are the min. and max. feature values in the set. This codes the weights to properly range from -1 to 1, and be roughly normalised across the features.

The number of questions asked is adaptive based on how quickly the inferred weights converge to within a Normal 95% confidence interval.

3.5.3 Model-View-ViewModel

MVVM is an interface design pattern that efficiently splits the main program logic, Model, from the GUI, View (See Figure 3.44). The ViewModel is the interface between the two, which contains logic for a suitable View. Crucially, the View and Model have no knowledge of each other.

I used Windows Presentation Foundation (WPF) for the GUIs, since it is designed to make the MVVM pattern easy to implement and maintain. Views are in an XAML format, and they use *data binding* to subscribe to properties and commands in the ViewModel code. Since WPF is for C#.NET, it interacted seamlessly with my Model (the feature computer).

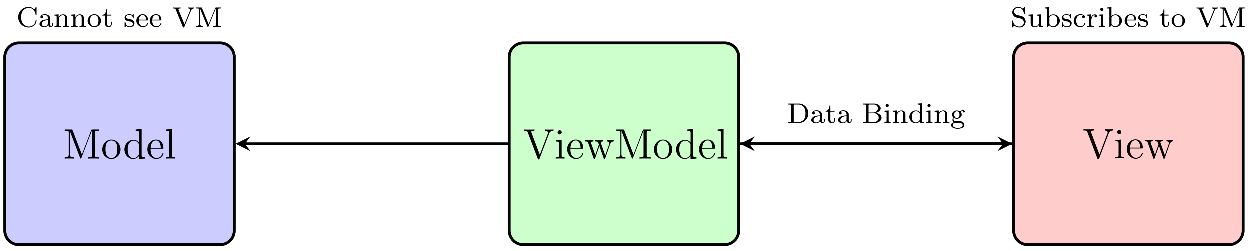


Figure 3.44: The arrow means "sees". The Model can be developed separately. A ViewModel is made for each screen in the program such that hot-swappable Views can be bound to it.

The Efficient interface has a single View-ViewModel, but the Intuitive interface uses a stack-based View-ViewModel screen approach.

3.6 Optimisations

3.6.1 Storing feature values for re-use

This optimisation stores computed feature values inside image metadata so that they may be quickly read if used by the program again. It was necessary for the human evaluation, to not require a minute of computation time for each participant. It is an example of the Agile strategy's ability to adapt to changing requirements, since I had not considered it at the point of writing the Project Proposal.

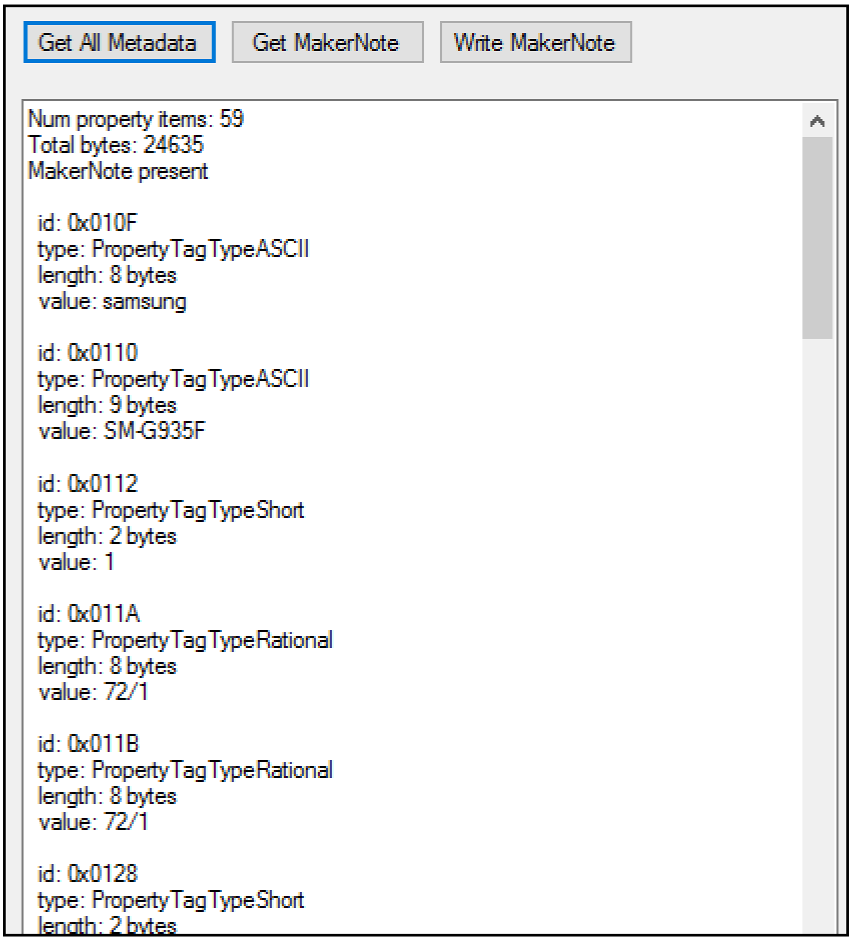
My program uses the `MakerNote` field in JPEG Exif metadata. For research and testing, I created an Exif viewing tool (Figure 3.45). I tested editing/reading/adding the metadata, which did not cause any Windows or Android viewing problems.

Data stored using Exif is invisible to the user, making the solution clean and seamless. Furthermore, metadata information is bound to the image, so it travels with it across networks and file-systems. The only drawback of the Exif method is that it only works on JPEG images (other image formats require re-computation each time). However, this project is intended for photographs, for which JPEG is the dominant compression standard.

Before discovering Exif, I planned on using a text/XML file in the chosen folder to store all image features. However, this solution is clunky and fails if images are renamed, removed or added. Even saving the file in a global user folder does not get around the problem.

Integrating this into the multi-threaded infrastructure gave a speed-up of about 95%, as graphed in Figure 3.46.

Feature values are stored as a byte-array and the `ImageFeatureVector` class allows conversion to and from this representation, shown in Figure 3.47. To check that saved metadata is from my program, the `Version` and `NumberOfFeatures` fields must be as



```
Num property items: 59
Total bytes: 24635
MakerNote present

 id: 0x010F
 type: PropertyTagTypeASCII
 length: 8 bytes
 value: samsung

 id: 0x0110
 type: PropertyTagTypeASCII
 length: 9 bytes
 value: SM-G935F

 id: 0x0112
 type: PropertyTagTypeShort
 length: 2 bytes
 value: 1

 id: 0x011A
 type: PropertyTagTypeRational
 length: 8 bytes
 value: 72/1

 id: 0x011B
 type: PropertyTagTypeRational
 length: 8 bytes
 value: 72/1

 id: 0x0128
 type: PropertyTagTypeShort
 length: 2 bytes
```

Figure 3.45: Exif viewer snippet

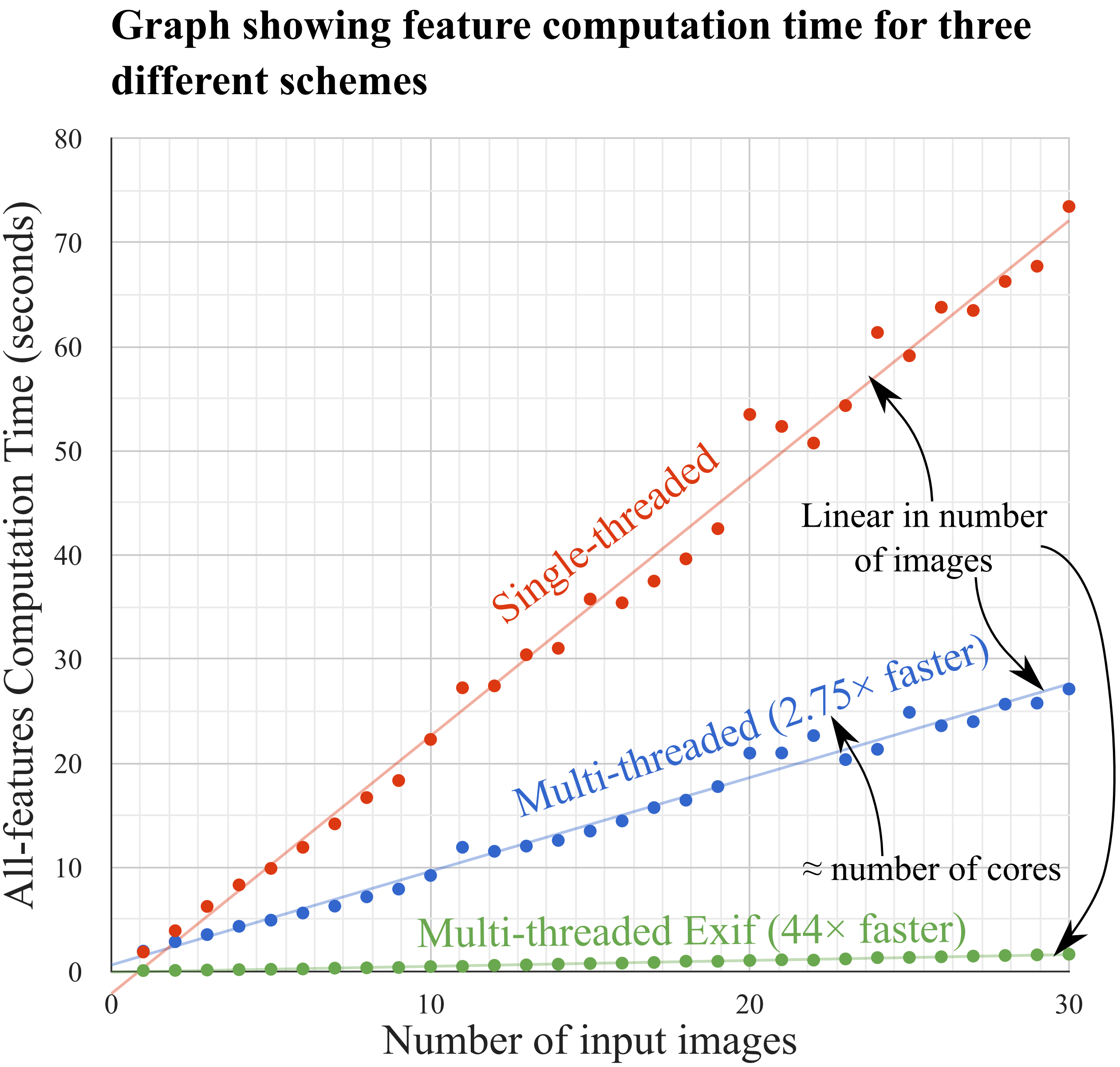


Figure 3.46: Pre-computing features gives a further 16.1× reduction in running time compared to multi-threaded, or a 44× speed-up compared to single-threaded.

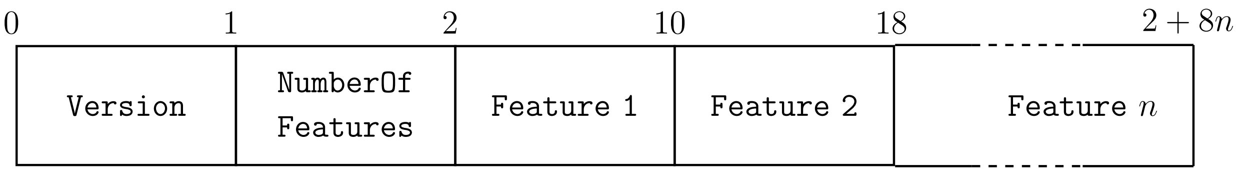


Figure 3.47: `ImageFeatureVector` byte-array representation. All feature values are double-precision floating point numbers.

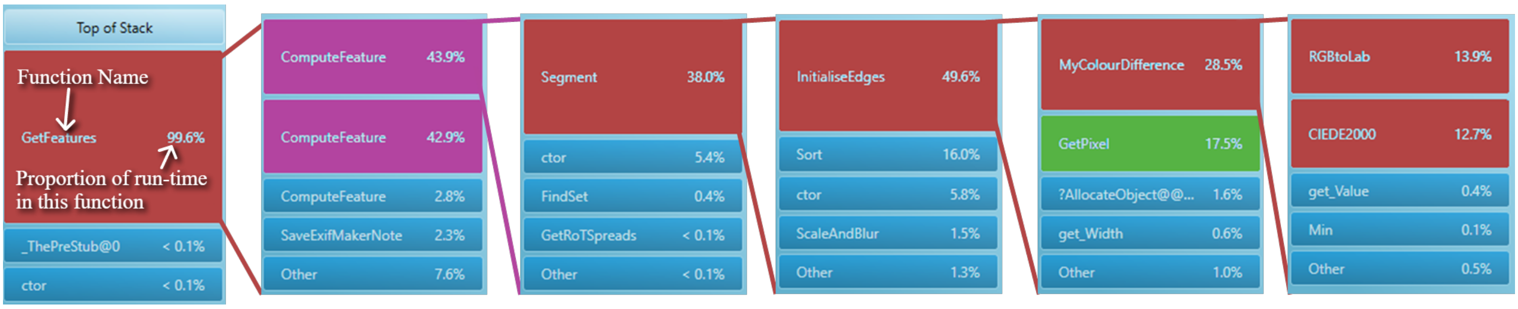


Figure 3.48: Performance analysis. Calls to `System.Drawing.Bitmap.GetPixel(x,y)` were hot, and the Segmentation algorithm was run again for each feature.

expected. Also, the length of the byte array must be consistent with the expected number of features ($= 2 + 8 \times \mathit{NumFeatures}$). If the feature implementations are upgraded, the current expected version is incremented. Future work could be improving this check, perhaps using a checksum or hash of the pixel data.

3.6.2 Detecting slow code in the segmentation implementation

By using a lower-level extension of C#.NET's `Bitmap` image class, I achieved approximately a 25% speed-up.

After implementing the segmentation algorithm, I used a Performance Analyser to find out which parts of the code were hottest. Figure 3.48 shows that this was finding pixel difference. I could not optimise the CIEDE2000 implementation, and the colour space conversion was part of a library. However, I could improve the tens of thousands of calls to `Bitmap.GetPixel(x,y)`. The speed-up was achieved using a `DirectBitmap` extension, which treats the image as an `Int32` array. This gets around having to lock and unlock the image each time a pixel value is read.

I achieved a further 4× speed-up by pre-computing the saliency segmentation before extracting the high-level features. Initially, each high-level feature was self contained by taking the image in as an input, and outputting the feature value. Although this made for tidy, modular code, it meant that the image segmentation algorithm was run for each of the high-level features.

3.7 Testing and Debugging

The feature extractor implementations were unit tested using constructed image examples, with tests being sprint objectives. For the low-level features, I used an image editor to increase/decrease the brightness/contrast/saturation/blur in a set of images, and tested to see if the feature values changed correctly (see Figure 3.49). I also tested edge-cases using extreme artificial examples. This helped to determine the theoretical range of the feature values. For example, it was important that a completely white image has a brightness value of 1.

For the high-level features, more specific test data was required (see Figure 3.50 for examples):

- **Rule of Thirds**

  Images with circles exactly on "power-points" should receive values of 1, and the value should reduce as the circles move farther away.

- **Subject(s) Size**

  The feature value should be between 0 and 1, increasing as the circle radius increases.

- **Shape Convexity**

  Images with higher proportions of convex shapes should receiver higher feature values.

- **Background Distraction**

  The white square is the 'subject', and increasing levels of Perlin noise is introduced behind it.

When performing parameter sweeps for the segmentation algorithm, the program needed to be running for ≈ 30 minutes at once. It was here when I first noticed the program running out of memory after a few minutes of feature computation. I used the dotMemory[7] profiler to find the source of the leaks to be with uses of the optimising `DirectBitmap` image class, which bypasses the Garbage Collector. This means that instances need to be properly `Dispose`d of after use. The C# `using(){...}` block, which guarantees the freeing of unmanaged memory even in the case of unchecked exceptions, was useful. Since I had left slack-time in the project schedule, I could insert a sprint for investigating and fixing the memory leaks. The modularity of my Agile sprints helped here.

Another bug in segmentation was that I accidentally bypassed the Gaussian blurring pre-processing step. For some reason, ReSharper's dead-code highlighting did not flag this up. However, it did not matter because $(k = 125, \sigma = 0)$ was about as good a set of parameters as $(k = 125, \sigma = 0.6)$.

[7] `https://www.jetbrains.com/dotmemory/`

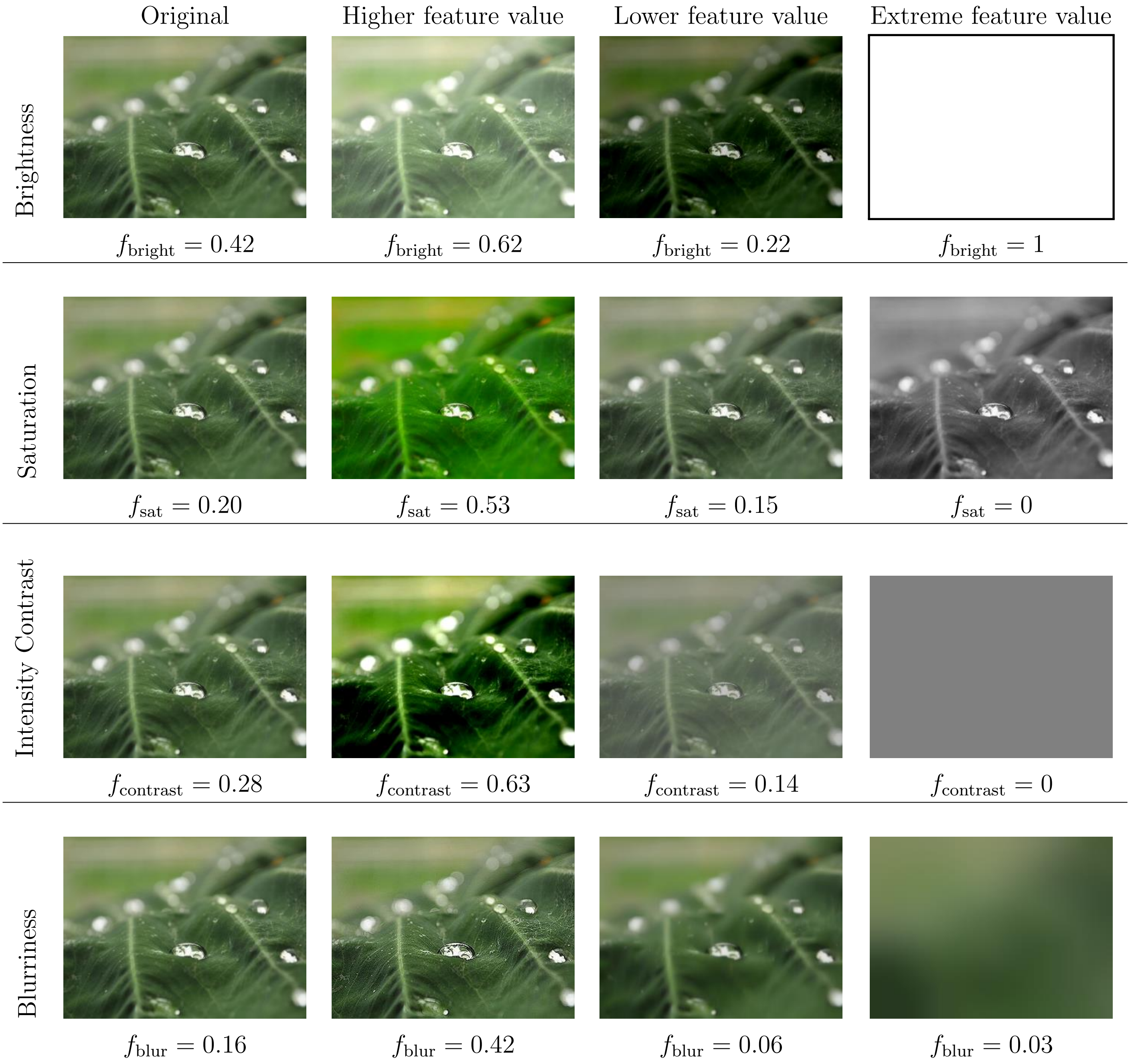


Figure 3.49: Low-level Feature Testing

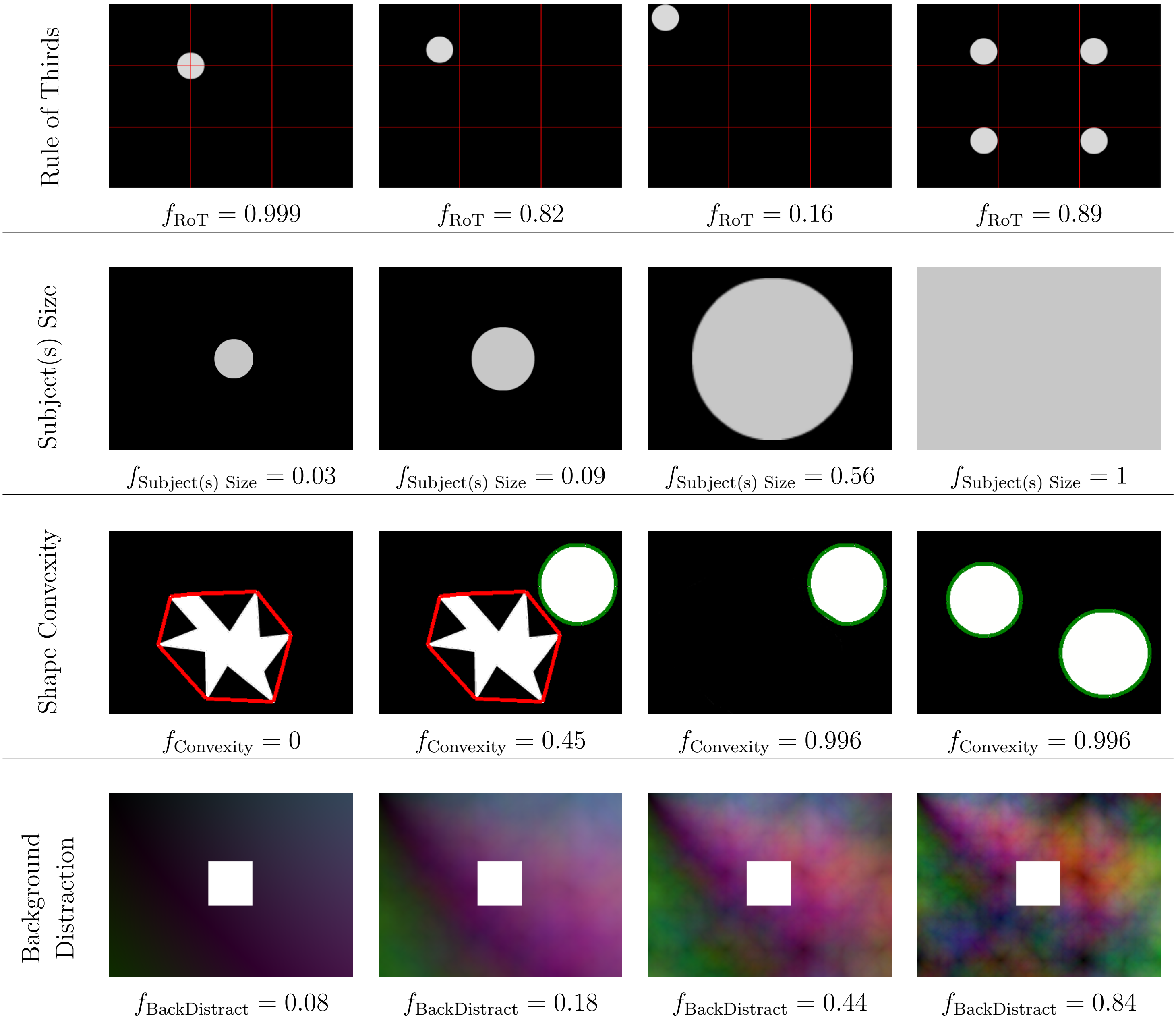

Rule of Thirds
$f_{\text{RoT}} = 0.999$
$f_{\text{RoT}} = 0.82$
$f_{\text{RoT}} = 0.16$
$f_{\text{RoT}} = 0.89$
Subject(s) Size
$f_{\text{Subject(s) Size}} = 0.03$
$f_{\text{Subject(s) Size}} = 0.09$
$f_{\text{Subject(s) Size}} = 0.56$
$f_{\text{Subject(s) Size}} = 1$
Shape Convexity
$f_{\text{Convexity}} = 0$
$f_{\text{Convexity}} = 0.45$
$f_{\text{Convexity}} = 0.996$
$f_{\text{Convexity}} = 0.996$
Background Distraction
$f_{\text{BackDistract}} = 0.08$
$f_{\text{BackDistract}} = 0.18$
$f_{\text{BackDistract}} = 0.44$
$f_{\text{BackDistract}} = 0.84$

Figure 3.50: High-level Feature Testing

Chapter 4

Evaluation

The project successfully meets the core success criteria laid out in the Proposal.

To evaluate the project I carried out a quantitative evaluation of the saliency segmentation algorithm, and a human evaluation of the feature-based program. I used summative, empirical methods.

The quantitative segmentation evaluation used the Berkeley Segmentation Dataset of manually-segmented images [2]. This allowed me to compare my computer-generated segmentations against the "ground-truth" of humans.

For the human evaluation, participants manually sorted a set of photographs and later used my program on the *same* images. I compared the performance of the tool with different combinations of features enabled.

Furthermore, I examined the correlation between feature values themselves, to determine which features contained the most independent information.

This chapter ends with a discussion of the strengths and weaknesses of the approach, and feedback from the study participants.

4.1 List of Objectives

The purpose of the evaluation was to answer the following questions:

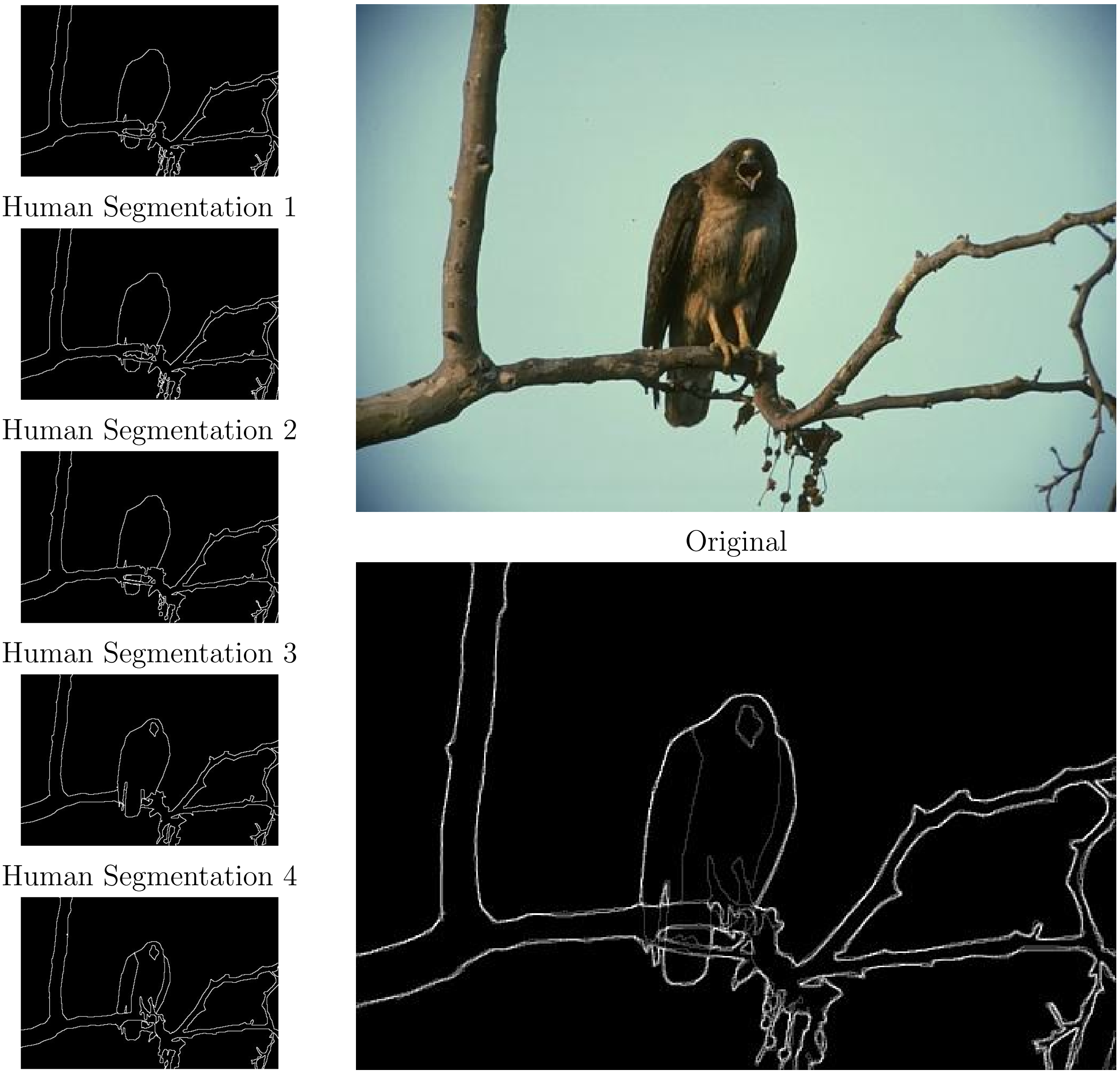
1. **Quantitatively, how much does augmenting the segmentation with saliency information help?** **(CORE)**
2. **Does the tool extract information that can discriminate aesthetic image quality?** Is the tool result better than a random guess of the user's preferred ordering? **(CORE)**
3. **Do the segmentation-derived features help?** Are the tool sortings more accurate when the high-level segmentation-derived features are enabled, compared to just the low-level features? **(CORE)**
4. **Is aesthetic image quality subjective?** (IMPORTANT)
5. **Does the tool speed up the task of image sorting?** **(EXTENSION)**
6. **Are humans biased by the initial order of images?** **(EXTENSION)**
7. **Do peoples' image quality preferences change over time?** **(EXTENSION)**
8. Also, if the preferences do change, **Does using the tool bias peoples' image quality preferences?** (EXTENSION)
9. **How satisfied are people with the tool's results?** **(EXTENSION)**
10. **How useful and popular is each feature?** **(EXTENSION)**

4.2 Segmentation Evaluation

I used the Berkeley Segmentation Dataset (BSDS) of images manually segmented by humans to evaluate my implementation [2]. Figure 4.1 gives an example from the dataset. I wrote my own tool for evaluating my results against other algorithms on the dataset.

First, the saliency segmentation is converted into a format compatible with the dataset, as shown in Figure 4.2). The generated edge-map G is defined as: $\forall\ i, j$

> If pixel (i, j) is in the same segment as its 8 neighbours, $G_{ij} = 0$ (black)
>
> Otherwise G_{ij} is part of an edge. Its value (brightness) is the square of the maximum segment saliency in the 3x3 neighbourhood. Segments smaller than 1% of the image are ignored.



Human Segmentation 1

Human Segmentation 2

Human Segmentation 3

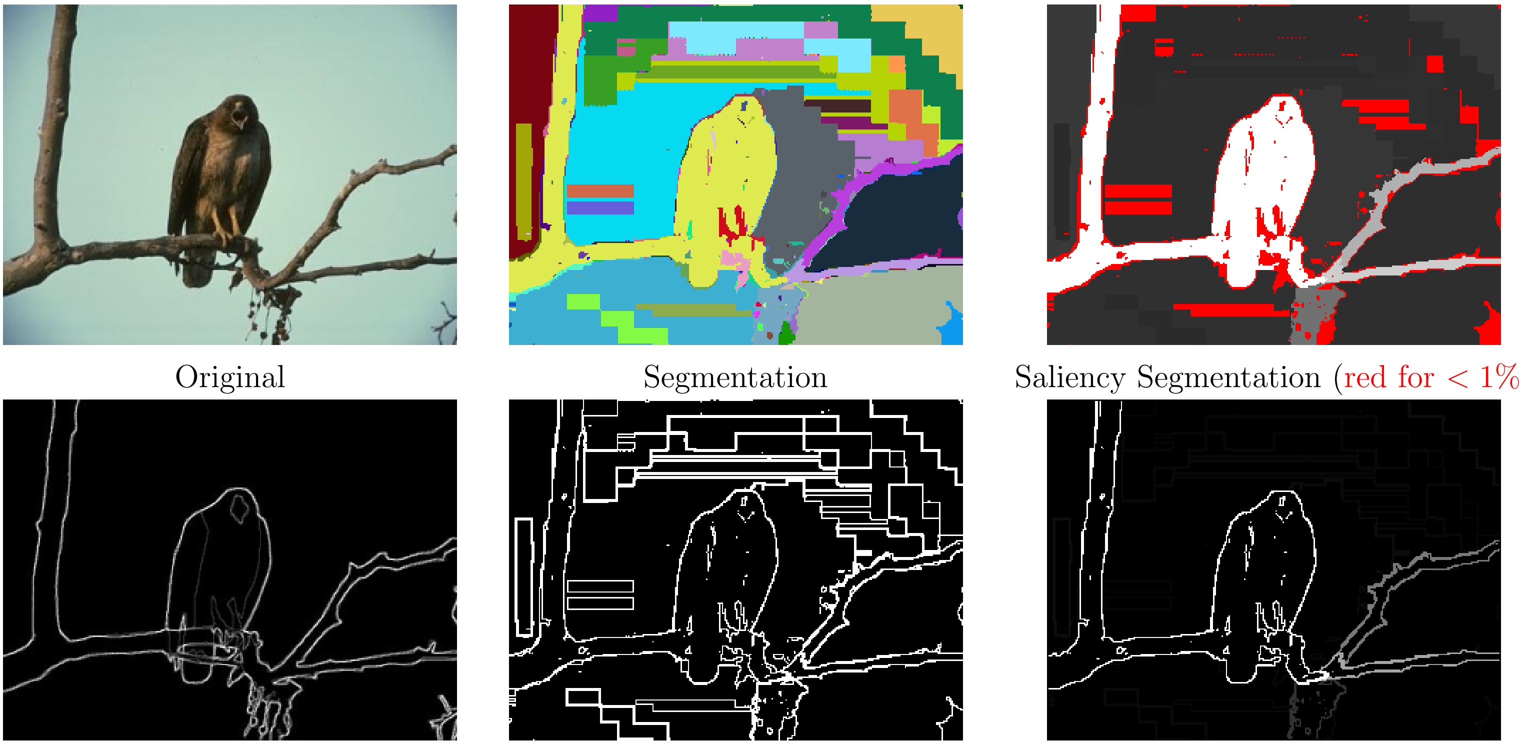
Human Segmentation 4

Human Segmentation 5

Original

Combined Segmentation (average of the five humans)

Figure 4.1: Berkeley Segmentation Dataset Example



Original

Segmentation

Saliency Segmentation (red for $< 1\%$)

Human Edges

Generated Edges (without Saliency)

Generated Edges (with Saliency)

Figure 4.2: Generated Edge Map Example

After processing to account for slight boundary offsets, I compute the accuracy as a measure of difference between the two edge maps,

$$A(G,T) = 1 - \frac{\overbrace{\sum_i \sum_j (G_{ij} - T_{ij})^2}^{\text{Squared differences of edge maps}}}{\underbrace{\sum_i \sum_j G_{ij}^2 + \sum_i \sum_j T_{ij}^2}_{\text{Squared differences if the edge maps did not overlap anywhere}}}.$$

Intuitively, if the generated map were to exactly match the true map, the numerator would be 0, giving an accuracy of 1. If the generated map did not match up at all with the true map, then the numerator would equal the denominator and the accuracy would be 0.

The results of the segmentation evaluation are shown in Table 4.1. As expected, my algorithm performed about half-way between K-Means and the state-of-the-art, *gPb*-ucm-color [3]. This is consistent with my initial research, shown in Figure 2.1. A more detailed analysis is given in Figure 4.3, which shows the distributions of accuracies of images across the dataset for different algorithms.

My results can be trusted because the ordering is consistent with the results of Arbelaez *et al.* [3], the creators of the BSDS.

Algorithm	Mean Accuracy	Standard Deviation
gPb-ucm-color	0.75	0.12
Brightness/Texture Gradients	0.59	0.12
My Saliency Segmentation	**0.55**	**0.14**
My Segmentation (no saliency)	0.44	0.14
Color Gradient	0.40	0.12
Otsu Binary Thresholding	0.24	0.14
K-Means ($k = 2$)	0.24	0.14
K-Means ($k = 3$)	0.18	0.11
K-Means ($k = 4$)	0.15	0.10
Random	0.12	0.04

Table 4.1: Average accuracies of different segmentation algorithms.

The results show an improvement in accuracy of 25% is achieved by using saliency information in my implementation.

4.3 Human Evaluation

The human evaluation had two stages: first, I asked participants to manually sort a set of 40 photographs and then, some time later, to use two separate interfaces to my tool (Efficient and Intuitive, see Section 3.5) to sort the *same* images. The tool aimed to assist sorting by using the extracted image features. If the tool successfully extracted useful aesthetic information, the resulting data would be correlated with the user's manual ranking. Participants were divided into two groups: one with only low-level features enabled, and one with both high- and low-level features enabled. There were 4 different image datasets, each with different themes to evaluate the tool in a different setting.

4.3.1 Evaluation Structure

Several considerations needed to be made in planning the evaluation:

- Participant demographic. I aimed to recruit a representative number of people with differing levels of photography experience (Table 4.2). Whilst most participants were students, I made sure to include some from other age groups and backgrounds where possible.
- To capture individuals' subjective preferences, I asked people to sort the same set of images in both stages. Comparing their tool sorting with another participant's manual sorting would not evaluate how well the program captures one person's unique preferences.

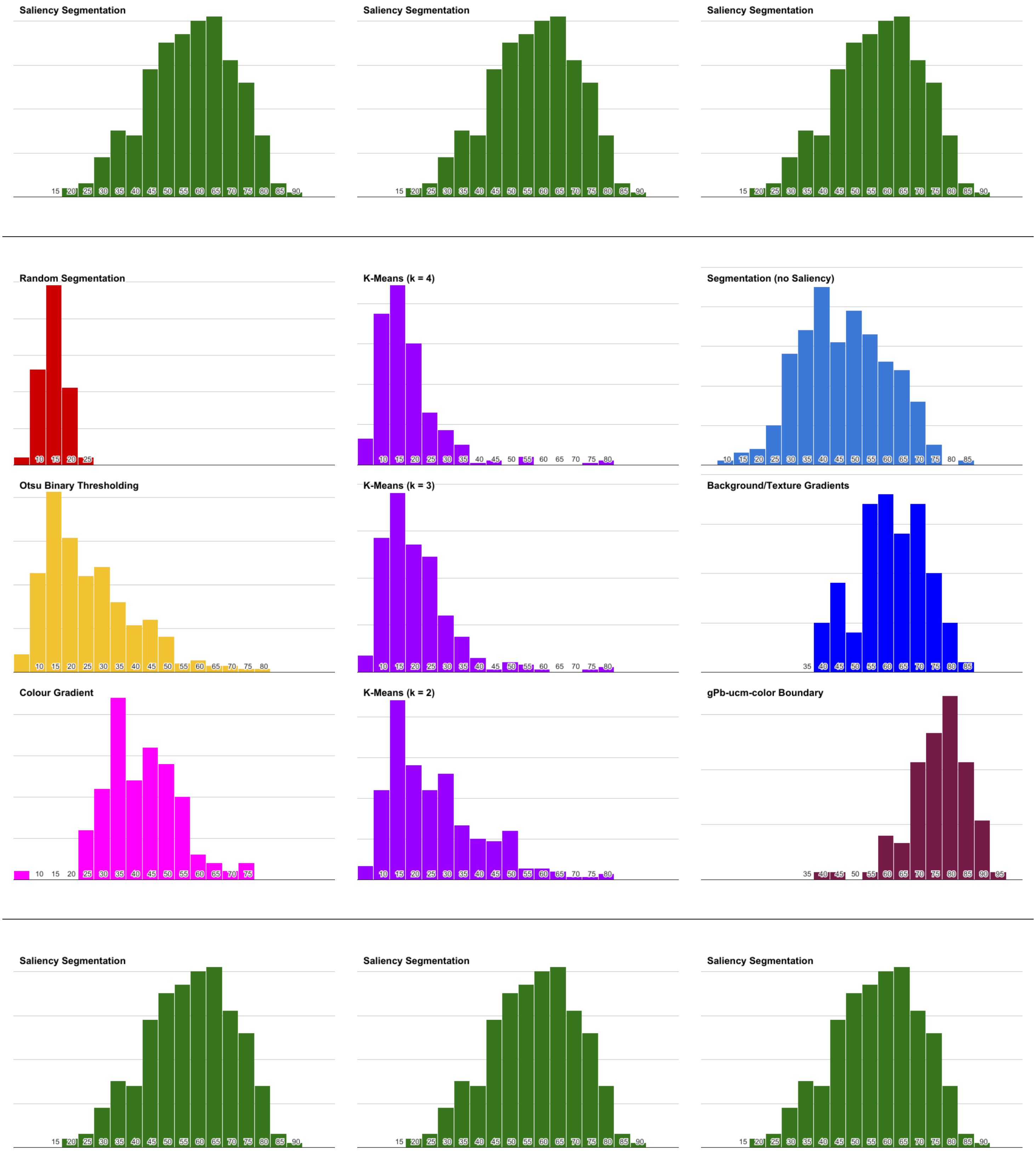


Figure 4.3: Normalised segmentation histograms. x-axis is the algorithm's segmentation accuracy against ground truth, as a percentage. y-axis is the count of images falling in an accuracy bin.

Experience	**Count**	**%**
Almost never use a camera	4	17%
Occasionally take photographs	14	61%
Photography is a hobby	5	22%

Table 4.2: Participant Photography Experience

- To mitigate bias of the manual sorting on the subsequent tool sorting, a week's gap was left between the two stages. The assumption (discussed in the next section) was that people agree with their own sorting at a later time.
- To eliminate any potential bias from the initial order of images, I randomised the filenames for each user, so the images would appear in a random order on their computer.

I constructed four 40-image datasets for the evaluation, with different themes:

Dataset 1: (to evaluate the application in the use-case of a professional photographer) Images containing well-defined subjects, from professional photographers.

Dataset 2: (to evaluate discrimination by subject matter) Also containing well-defined subjects, but split into 4 groups of 10, each of which was from a single nature shoot: Birds, Butterflies, Goats and Reindeer.

Dataset 3: Same theme as Dataset 1, but for amateur photographers.

Dataset 4: (to evaluate performance with less accurate segmentations) Images where a subject is difficult to define, such as landscapes and textures.

4.3.2 Speed of Sorting

Figure 4.4 shows that both tools are faster than manually sorting, and that the Intuitive interface is much faster to use than the slider-based one. This is partly because the participants had no training time to get used to the slider-based approach. The Efficient interface seems to take longer when high-level features are enabled, which makes sense because there are twice as many sliders to worry about.

4.3.3 Hypothesis Testing

For evaluating the tool, only data from sets 1 and 3 are used. This is because the high-level features are not designed for the other two sets, which do not contain well-defined subjects. I compare the results between datasets in Section 4.5.3. Apart from hypothesis test 3, I only use data from participants who had segmentation-derived features enabled. This evaluates the full power of the tool.

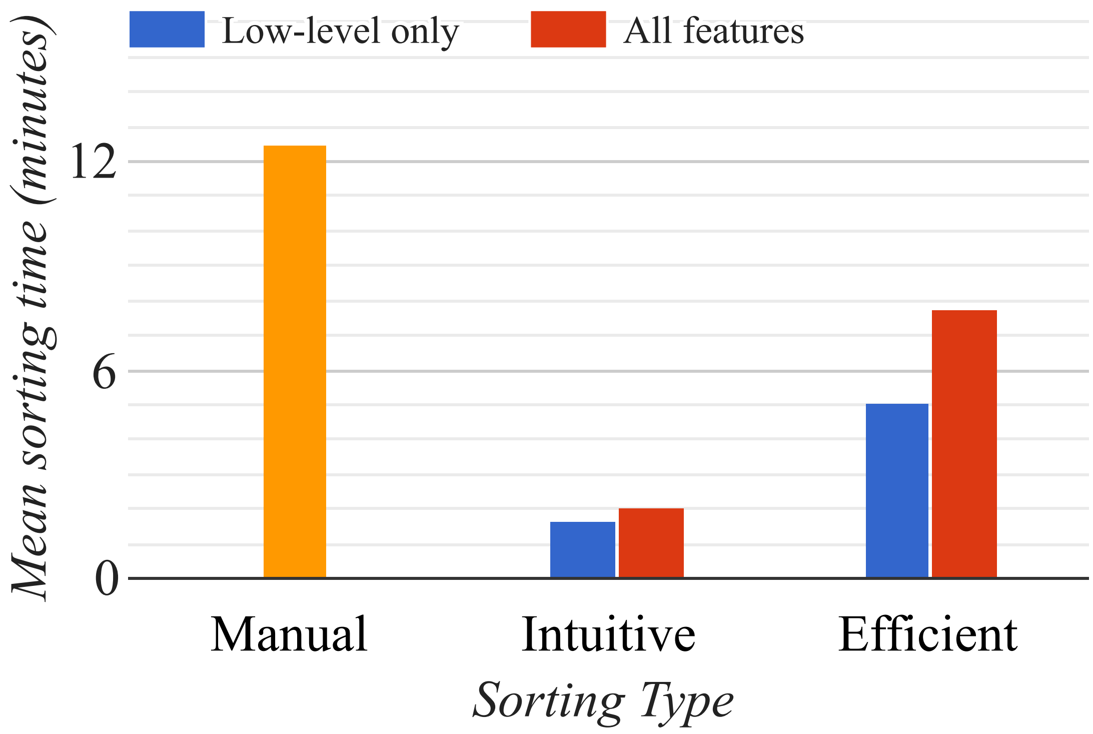


Figure 4.4: Speed of Sorting

For each of the following hypothesis tests, I summarise the results and give the *effect size* (correlation difference). Cohen [5] suggests that, in the social sciences, a correlation $|r| = 0.10$ is 'small', $|r| = 0.30$ is 'medium', and $|r| = 0.50$ is 'large'. The full statistical reasoning for each test can be found in Appendix B.

1. Does the Efficient tool extract some information that can discriminate aesthetic image quality? Yes (with 99.96% significance)

Effect size for tool accuracy (average correlation between tool sorting and manual sorting): $r = 0.24$.

The Efficient tool was accurate enough in my study to suggest that it successfully extracts personal preference information. This means that its results are better than a random guess.

2. Does the Intuitive tool extract some information that can discriminate aesthetic image quality? Yes (with 99.999% significance)

Effect size for tool accuracy: $r = 0.28$.

The Intuitive interface also passes.

3. Do the segmentation-derived features help? Not enough data to conclude

Effect size for accuracy gain when high-level features are enabled: $r = 0.08$.

There is not sufficient evidence to suggest that the segmentation derived features help. The test would only pass with less than 82% significance. This is likely to be due to insufficient data/participants, since user feedback suggested they did actually help.

4. Is aesthetic image quality subjective? Yes (with 99.9999% significance)

Effect size for difference between different participants' manual sortings: $r = 0.68$.

People do not sort images in the same way. That is, they have their own individual image preferences, rather than a collective view of a "good image". This is perhaps obvious, but is crucial for the motivation of my approach for the tool, which does not try to learn an average of many humans' image preferences (as a machine learning approach might). Instead, individual subjectivity is left with each user.

5. Are people biased by the initial ordering of images? Not with 95% significance

Effect size for average correlation between initial order and manual ranking: $r = 0.04$.

On average, people were not biased by the original image order, which means that it was probably not necessary to randomise the filenames.

6. Do peoples' image quality preferences change over time? Yes (with 99.99% significance)

I asked some participants to perform a manual sorting of their images a second time after doing the main study.

Effect size for change in manual sorting over time: $r = 0.26$.

Peoples' image preferences do change over time. This further motives my approach, but reduces the reliability of the study because I assumed peoples' manual sortings to be ground truth data. In fact, preferences may have changed by the time participants re-sorted the image set using the tool.

7. Does using the tool bias peoples' image quality preferences? Not at 95% significance

Effect size for correlation change of manual sortings with the tool sorting: $r = 0.01$.

People were not biased by the tool, and so it is unlikely to be the cause of the change found in test 6.

	Brightness	Intensity Contrast	Saturation	Blurriness	Subject(s) Size	Rule of Thirds	Shape Convexity	Background Distraction	
Brightness		-0.73	-0.42	-0.21	0.13	-0.07	-0.08	0.05	Low Level
Intensity Contrast	-0.73		0.36	0.18	-0.05	0.04	0.29	-0.06	
Saturation	-0.42	0.36		0.21	-0.16	0.17	0.22	0.06	
Blurriness	-0.21	0.18	0.21		-0.02	-0.10	0.05	0.17	
Subject(s) Size	0.13	-0.05	-0.16	-0.02		0.04	-0.20	-0.40	High Level
Rule of Thirds	-0.07	0.04	0.17	-0.10	0.04		-0.11	-0.10	
Shape Convexity	-0.08	0.29	0.22	0.05	-0.20	-0.11		0.06	
Background Distraction	0.05	-0.06	0.06	0.17	-0.40	-0.10	0.06		
	Low Level				High Level				

Table 4.3: There is not as much correlation between low- and high-level features as there is within the two categories.

Feature	Σ |**Correlation**|	
Rule of Thirds	0.63	**Most independent**
Background Distraction	0.90	
Blurriness	0.93	
Subject(s) Size	1.00	
Shape Convexity	1.01	
Saturation	1.59	
Brightness	1.69	
Intensity Contrast	1.71	**Most correlated**

Table 4.4: The second column is the sums of rows in Table 4.3. *Rule of Thirds* correlates the least with the other features, and *Intensity Contrast* and *Brightness* correlate the most.

4.4 Individual Features Evaluation

I sorted 80 images by the features separately and computed the correlations for each of the 28 pairs of features, shown in Table 4.3. The most correlated feature pairs are:

- *Brightness* and *Intensity Contrast* (negatively correlated) – Night-time images have high contrast. The average intensity is brought down by the darkness, but there is a large variance in intensities (from black to normal-brightness subjects).
- *Subject(s) Size* and *Background Distraction* (negatively correlated) – *Background Distraction* uses the *Subject(s) Sizes* information to decide what the background is. If the "subject" is large, then the "background" is small, which limits the number of possible colours inside it (and vice versa).
- *Brightness* and *Saturation* (positively correlated) – This could be due to the specific *Saturation* implementation, which sees very dark colours as highly saturated.

Furthermore, there is not as much correlation between low- and high-level features as there is within the two categories. This makes sense, because the saliency segmentation process is long and non-linear, which is unlikely to bear any resemblance to the relatively simple low-level feature extractors.

Table 4.4 summarises the data to show that *Rule of Thirds* captures the most unique image information that the other features do not, so is most crucial for allowing different image sortings. *Intensity Contrast* or *Brightness* would be "missed least" if they were deleted from the feature set. However, it is important to note that this analysis does not say anything about which features are best for discriminating image quality. For example, it could be the case that all of the "unique" information that *Rule of Thirds* contains is useless for the application.

Although it would be nice to have a minimal set of features which are like some form of *Principal Components* of the input image set, a trade-off must be made for my approach so that the features can be: (a) constant across different image sets, so that they do not need to be redefined depending on the exact set of images; and (b) meaningful to a human, so that useful insight may be gained about what high-level properties discriminate image quality best, and so that slider-based sorting is possible.

4.5 Discussion

4.5.1 Which features are best?

To get an idea of which features are most useful for discriminating image quality, I examined the feature weights. The Intuitive interface infers weights, and the Efficient interface lets users freely choose them. Table 4.5 shows "feature popularity" for the different interfaces and participant groups.

Feature	**Feature Popularity (Average absolute feature weight)**			
	Intuitive **All Features**	**Efficient** **All Features**	**Intuitive** **Low-level-only**	**Efficient** **Low-level-only**
Brightness	0.42	0.36	0.38	0.23
Intensity Contrast	0.23	0.32	0.73	0.52
Saturation	0.77	0.53	0.68	0.40
Blurriness	0.41	0.40	0.52	0.42
Subject(s) Size	0.46	0.41	-	-
Rule of Thirds	0.43	0.61	-	-
Shape Convexity	0.51	0.48	-	-
Background Distraction	0.49	0.58	-	-

Table 4.5: A larger popularity means that people, on average, relied upon the feature more to discriminate image quality.

Brightness seems to be the least used feature overall and, since it is one of the most redundant features (see Section 4.4), it would probably be missed the least if deleted from the feature set. *Saturation*, *Background Distraction* and *Rule of Thirds* appear to be the highly-used features.

Comparing the two interfaces, *Saturation* was highly-used in the Intuitive interface. This suggests that people truly use this feature the most (out of the 8). However, it was less popular with the Efficient interface, where participants could decide for themselves how important they thought it was.

Blurriness, *Subject(s) Size* and *Shape Convexity* are features whose popularity remains roughly constant between the interfaces. This implies that participants were good at deciding how useful these features actually were to them.

4.5.2 User Feedback

I asked participants to rate the correctness of the tool's sortings on a Likert-style scale between -3 and 3. Table 4.6 shows the results. These satisfaction scores are higher than the Spearman's correlation results from Section 4.3.3 (computing a weighted combination of the all-features happiness[1] gives 0.62). This could suggest that people evaluated the rankings in a more lenient way than Spearman's does.

Yeh *et al.* [22] evaluated their tool in a similar way, giving users the options "very good", "good", "bad", and "very bad". Their results were (8.3%, 91.7%, 0%, 0%) respectively. Computing a weighted combination of these[2] gives 0.54. This suggests that my modified approach may have been better than theirs, although this kind of human data is often noisy.

[1] $1 \times 17\% + \frac{2}{3} \times 67\% + 0 \times 17\% = 0.61\dot{6}$

[2] $1 \times 8.3\% + \frac{1}{2} \times 91.7\% = 0.5415$

Efficient Happiness	**All Features**	**Low-level only**	**Average**
(-3) - Nearly completely sorted the wrong way	0%	0%	0%
(-2) - Mostly sorted the wrong way	0%	0%	0%
(-1) - Slightly sorted the wrong way	0%	17%	8%
(0) - Apparently random	17%	0%	8%
(+1) - Slightly sorted correctly	0%	17%	8%
(+2) - Mostly sorted correctly	67%	67%	67%
(+3) - Nearly completely sorted correctly	17%	0%	8%

Table 4.6: Average participant satisfaction with the Efficient tool's results

Preference	**All Features**	**Low-level only**	**Average**
Manual Sorting	16.7%	16.7%	16.7%
Intuitive Tool	16.7%	16.7%	16.7%
Efficient Tool	66.7%	66.7%	66.7%

Table 4.7: Average participant sorting method preference

Table 4.7 suggest that the Efficient tool was most popular. There were, however, still some participants who preferred manual sorting.

Some participants gave written feedback after the study. One person wrote

> *“Once I had set the sliders I could easily see that some pictures that I may have manually sorted lower were in fact potentially of higher standard than originally thought (*[the efficient tool] *was kinda changing my mind as to what I originally thought was a good or bad picture(in a good way))”,*

which shows that preferences can change over time, and that happiness in the results is relative.

One participant, who considered photography as a hobby and had high-level features *disabled*, said that an important part of judging photo quality was

> *“the composition of the photo which is important but didn't have a bearing on a slider”.*

This is exactly what the high-level features aim to capture.

Other feedback included the image thumbnails being too small in the tools, and 40 images being too many to manually sort.

4.5.3 Subjects vs Landscapes

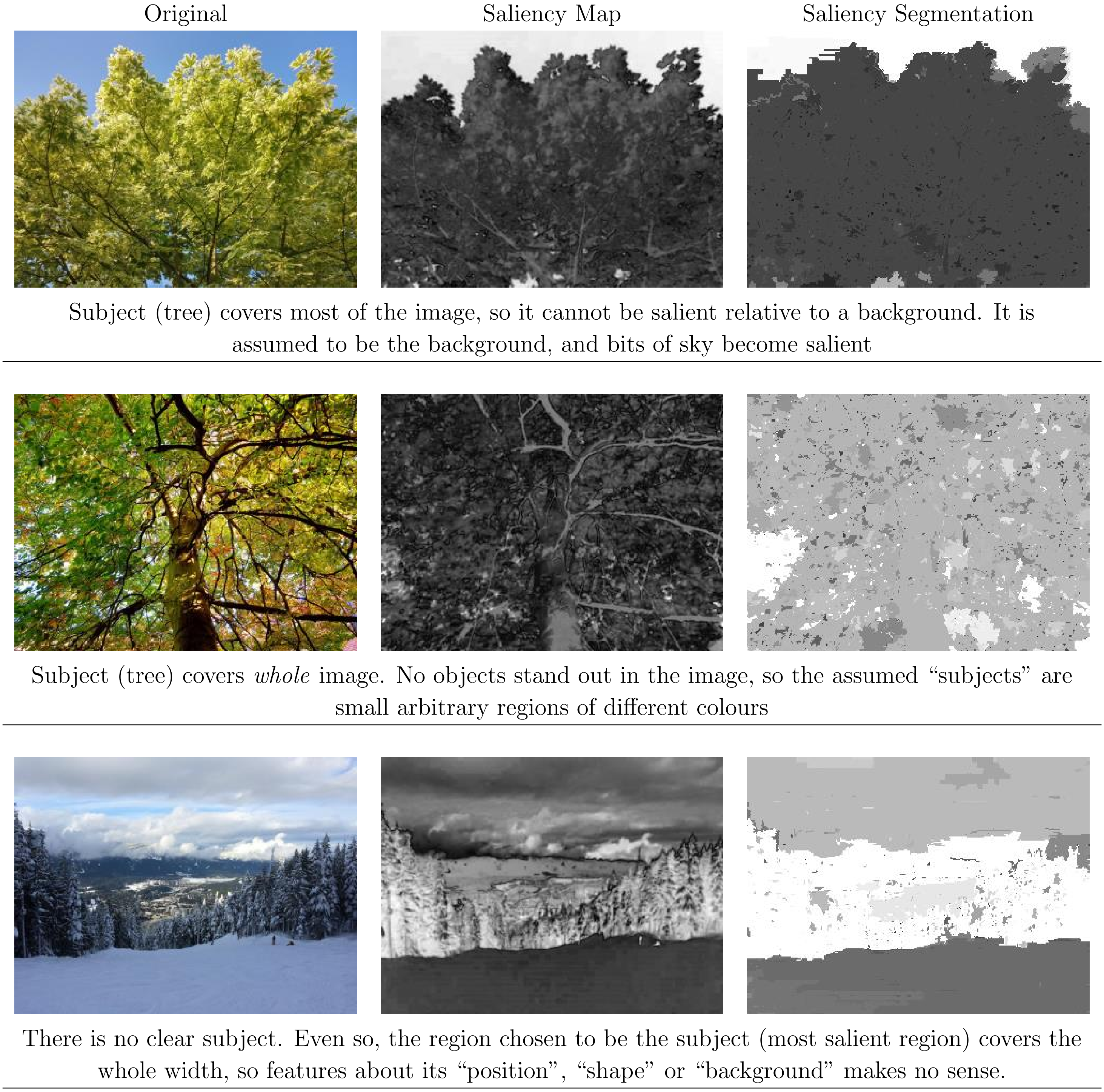
Since the high-level features are fundamentally defined in terms of the “subject(s)” of the image, they do not make much sense on images that do not have well defined subjects, such as the examples from Dataset 4 in Figure 4.5.

My tool also cannot discriminate by subject matter. This was particularly apparent in Dataset 2, which contained birds, butterflies, goats and reindeer. This is the dataset that the tool performed worse on, with an average correlation of 0.02 ± 0.16. One participant commented that they ranked the reindeer images lower than others because of finding reindeer ugly. *Reindeer ugliness* is not one of my features (nor should it be, since others find reindeer cute). There is an underlying problem here that humans use past experiences to judge images, which cannot be captured and assimilated by an automatic tool.

4.5.4 What’s missing?

Figure 4.6 shows that some images were consistently ranked incorrectly by the tool. The majority of these outliers are caused by segmentation errors, some examples of which are shown in Figure 4.7. However, the tool still consistently fails to correctly score some other images despite a sound segmentation, as shown in Figure 4.8. This suggests there is some information used by humans to discriminate image quality not captured by the implemented features.

This could be due to not having the optimal set of features or it could be due to external context that humans learn over the course of their lives. My high-level features make a step towards reasoning about objects in the image, but they cannot solve the AI-complete general object recognition task that would be necessary to rank a bird photograph higher because a particular user likes birds more than they like butterflies.

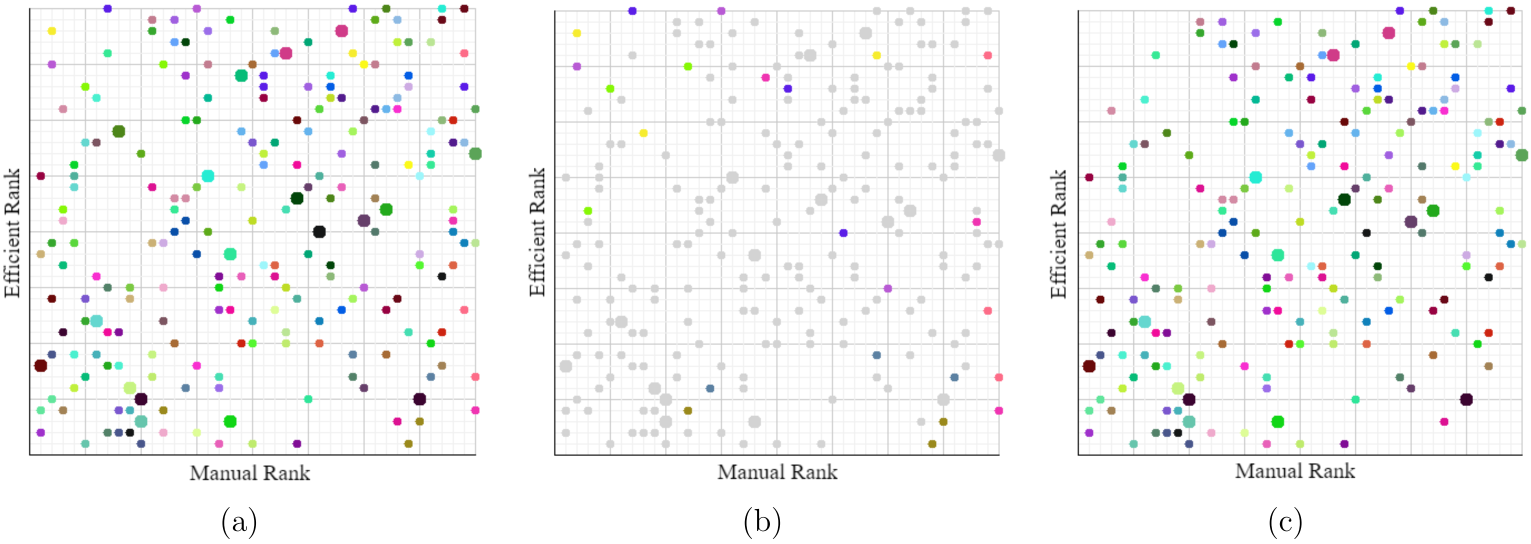


Subject (tree) covers most of the image, so it cannot be salient relative to a background. It is assumed to be the background, and bits of sky become salient

Subject (tree) covers *whole* image. No objects stand out in the image, so the assumed “subjects” are small arbitrary regions of different colours

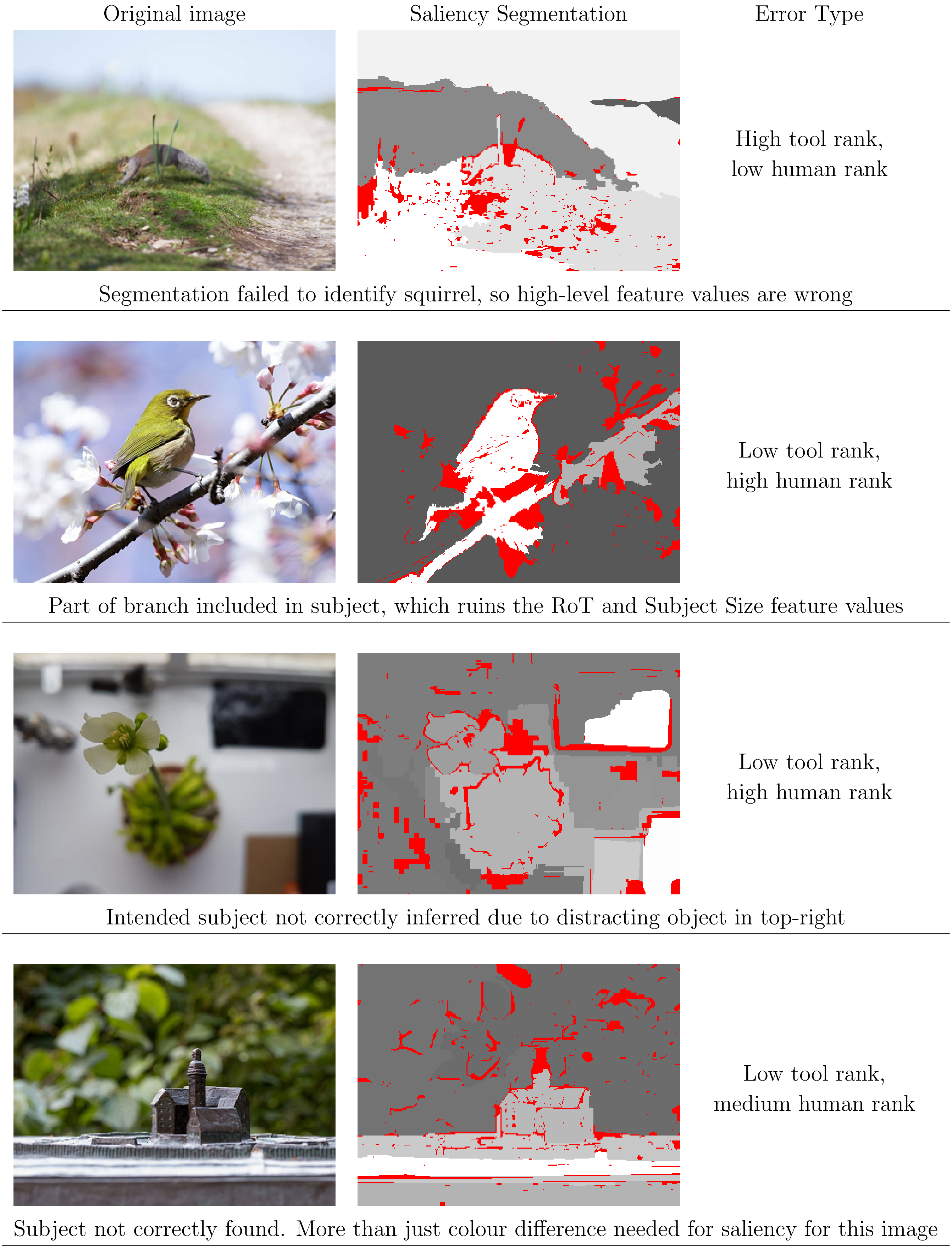
There is no clear subject. Even so, the region chosen to be the subject (most salient region) covers the whole width, so features about its “position”, “shape” or “background” makes no sense.

Figure 4.5: Images with poorly defined subjects have less meaningful saliency segmentations.



(a) (b) (c)

Figure 4.6: Correlation of tool rank against manual rank. The colour of each point indicates the image being ranked. (a) With all data points, correlation $r = 0.24$. (b) The program assigned incorrect scores to some images. (c) With worst 8 images removed, correlation $r = 0.36$



Original image | Saliency Segmentation | Error Type

High tool rank, low human rank

Segmentation failed to identify squirrel, so high-level feature values are wrong

Low tool rank, high human rank

Part of branch included in subject, which ruins the RoT and Subject Size feature values

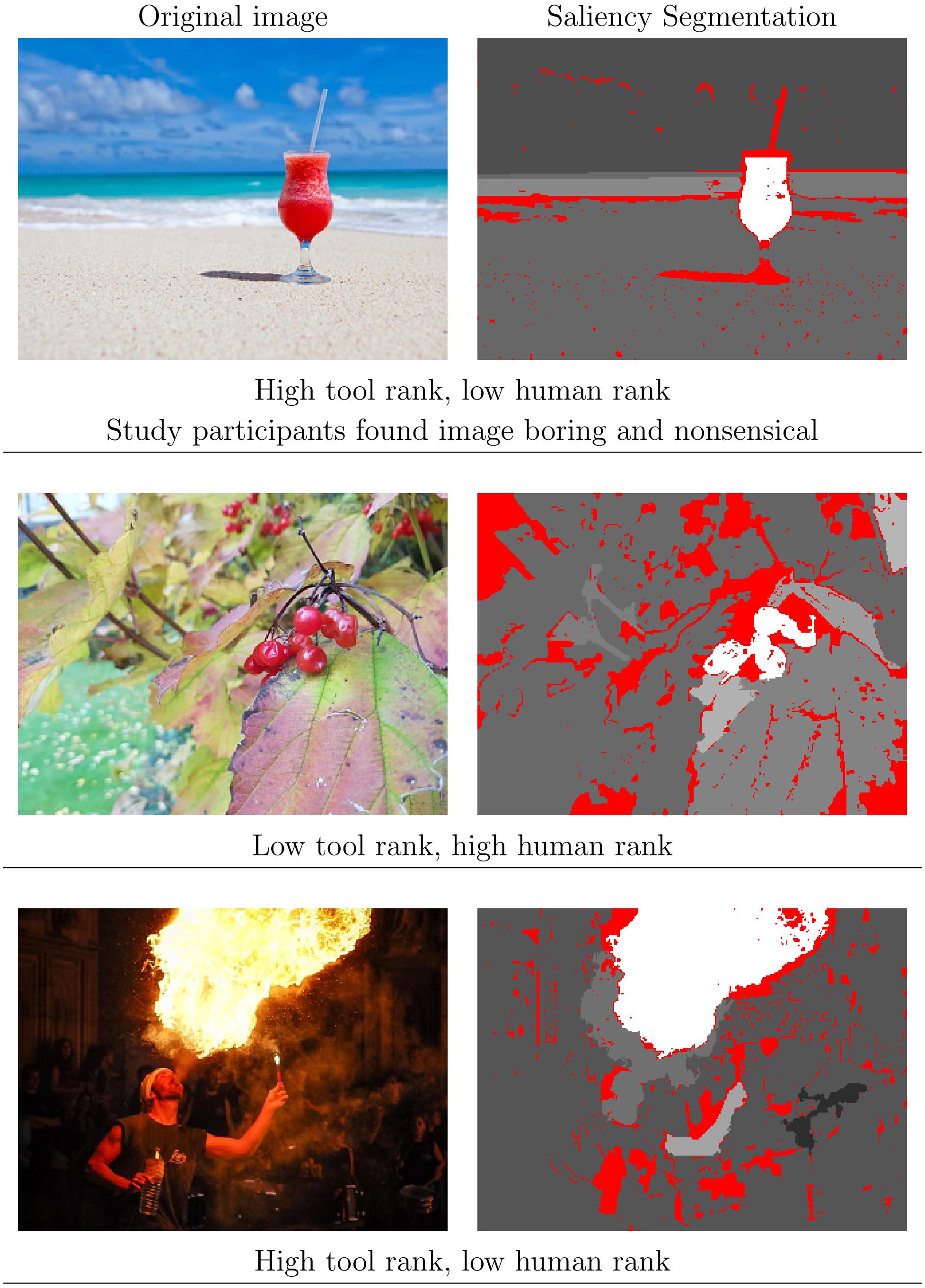
Low tool rank, high human rank

Intended subject not correctly inferred due to distracting object in top-right

Low tool rank, medium human rank

Subject not correctly found. More than just colour difference needed for saliency for this image

Figure 4.7: Segmentation Error Outliers



High tool rank, low human rank
Study participants found image boring and nonsensical

Low tool rank, high human rank

High tool rank, low human rank

Figure 4.8: Missing Information Outliers

Chapter 5

Conclusions

The project successfully met the core criteria established in the Proposal, as well as many extra goals along the way. I created a feature-based tool for assisting people in the task of preferentially sorting images, with the aim of leaving the inherent subjectivity with the humans that create it.

The user trial showed that my tool successfully extracted useful aesthetic information, and that it is faster and more popular than manual sorting. The results obtained were similar to that of Yeh *et al.* [22], on which the project is based. Although my evaluation of whether high-level, segmentation-derived features improve the accuracy was inconclusive, the study produced many other interesting results about the problem and my approach to tackling it. The quantitative results from the evaluation of the segmentation algorithm were consistent with expectations, and showed that saliency information helped.

5.1 Reflections

The chosen programming language, C#.NET, made GUI creation easy and was easier to learn than anticipated, in part due to the excellent code analysis tools available within the IDE. The Agile software engineering strategy encouraged me to create testable, modular work items that could fit into sprints in between supervisor meetings.

There are some things that I could have done differently, in hindsight:

- Data from only half of the participants was used for most of the tool's evaluation (datasets 1 and 3). I should have assigned more people to these sets than the other two, which were less important in evaluating the general behaviour of the tool.
- Furthermore, the segmentation algorithm failed on some images in sets 1 and 3, which gave an underrepresentation of the tool's full power. I should have put more thought into the construction of those datasets.

5.2 Future Work

- The *Blurriness* feature could be applied to the subject only, changing it from a low- to a high-level feature. This could be implemented using a *Discrete Wavelet Transform* with a wavelet basis such as the Gabor Wavelets. It would then discriminate between motion blur and depth-of-field.
- Investigate non-linear combinations of feature values, such as conditionals and thresholds.
- Some features could include content detection classifiers based on deep convolutional neural networks. For example, there could be a *Reindeer* slider with an associated classifier for detecting reindeer. People who like reindeer would then be able to assign this feature a high weight.
- State-of-the-art segmentation algorithms could be integrated with the same framework; more accurate segmentation-derived features could give more accurate sortings.

5.3 Final words

This project was a great opportunity to learn about computer vision, image processing and how humans judge image quality. Although I have not been able to statistically show that segmentation-derived features help with aesthetic image sorting, I justified and evaluated the feature-based approach to the problem and have shown that my tool successfully extracts user-specific information to produce “mostly correct” rankings. I hope that my program would be found useful by real users, to speed up a task common to professional, amateur and non-photographers alike.

Bibliography

[1] Radhakrishna Achanta et al. “Frequency-tuned salient region detection”. In: *Computer vision and pattern recognition, 2009. cvpr 2009. ieee conference on*. IEEE. 2009, pp. 1597–1604.

[2] Pablo Arbelaez, Charless Fowlkes, and David Martin. “The berkeley segmentation dataset and benchmark”. In: *see http://www. eecs. berkeley. edu/Research/Projects/CS/vision/bsds* (2007).

[3] Pablo Arbelaez et al. “Contour detection and hierarchical image segmentation”. In: *IEEE transactions on pattern analysis and machine intelligence* 33.5 (2011), pp. 898–916.

[4] Tunç Ozan Aydın, Aljoscha Smolic, and Markus Gross. “Automated aesthetic analysis of photographic images”. In: *IEEE transactions on visualization and computer graphics* 21.1 (2015), pp. 31–42.

[5] Jacob Cohen. “A power primer.” In: *Psychological bulletin* 112.1 (1992), p. 155.

[6] CIE Colorimetry. “CIE Publication”. In: *CIE, Paris* (1971), p. 15.

[7] Thomas H Cormen et al. *Introduction to algorithms*. MIT press Cambridge, 2009.

[8] Ritendra Datta et al. “Studying aesthetics in photographic images using a computational approach”. In: *European Conference on Computer Vision*. Springer. 2006, pp. 288–301.

[9] Pedro F Felzenszwalb and Daniel P Huttenlocher. “Efficient graph-based image segmentation”. In: *International journal of computer vision* 59.2 (2004), pp. 167–181.

[10] Matthias Grundmann et al. “Efficient hierarchical graph-based video segmentation”. In: *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE. 2010, pp. 2141–2148.

[11] George H Joblove and Donald Greenberg. “Color spaces for computer graphics”. In: *ACM siggraph computer graphics*. Vol. 12. 3. ACM. 1978, pp. 20–25.

[12] Yan Ke, Xiaoou Tang, and Feng Jing. “The design of high-level features for photo quality assessment”. In: *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*. Vol. 1. IEEE. 2006, pp. 419–426.

[13] Joseph B Kruskal. “On the shortest spanning subtree of a graph and the traveling salesman problem”. In: *Proceedings of the American Mathematical society* 7.1 (1956), pp. 48–50.

[14] Xin Lu et al. “Rating image aesthetics using deep learning”. In: *IEEE Transactions on Multimedia* 17.11 (2015), pp. 2021–2034.

[15] Yiwen Luo and Xiaoou Tang. “Photo and video quality evaluation: Focusing on the subject”. In: *European Conference on Computer Vision*. Springer. 2008, pp. 386–399.

[16] Ashutosh Saxena, Min Sun, and Andrew Y Ng. “Make3d: Learning 3d scene structure from a single still image”. In: *IEEE transactions on pattern analysis and machine intelligence* 31.5 (2009), pp. 824–840.

[17] Gaurav Sharma, Wencheng Wu, and Edul N Dalal. “The CIEDE2000 color-difference formula: Implementation notes, supplementary test data, and mathematical observations”. In: *Color Research & Application* 30.1 (2005), pp. 21–30.

[18] Jianbo Shi and Jitendra Malik. “Normalized cuts and image segmentation”. In: *IEEE Transactions on pattern analysis and machine intelligence* 22.8 (2000), pp. 888–905.

[19] Oshin Vartanian et al. “Impact of contour on aesthetic judgments and approach-avoidance decisions in architecture”. In: *Proceedings of the National Academy of Sciences* 110.Supplement 2 (2013), pp. 10446–10453.

[20] Uro Vovk, Franjo Pernus, and Botjan Likar. “A review of methods for correction of intensity inhomogeneity in MRI”. In: *IEEE transactions on medical imaging* 26.3 (2007), pp. 405–421.

[21] Zhou Wang et al. “Image quality assessment: from error visibility to structural similarity”. In: *IEEE transactions on image processing* 13.4 (2004), pp. 600–612.

[22] Che-Hua Yeh et al. “Personalized photograph ranking and selection system”. In: *Proceedings of the 18th ACM international conference on Multimedia*. ACM. 2010, pp. 211–220.

Appendix A

Preparation Task List

As required by the Agile project strategy, I made a chronologically ordered list of implementation tasks to complete. It is organised in Table A.1 by risk and difficulty.

1. Segmentation algorithm
2. Saliency augmentation to the segmentation
3. Implementation of feature extractors
   - (a) Rule of Thirds
   - (b) Subject(s) Size
   - (c) Brightness
   - (d) Intensity Contrast
   - (e) Saturation
   - (f) Blurriness
   - (g) Shape Convexity
   - (h) Background Distraction
4. Multi-threaded framework for computing feature vector of all images in a folder
5. Saving feature values in JPEG Exif meta-data for re-use
6. Windows GUI, using MVVM pattern, before the human evaluation
   - (a) Efficient Interface, which allows users to manually select feature values using slider
   - (b) Intuitive Interface, which should somehow predict user's feature weights

	Low Risk	Moderate Risk	High Risk
Low Difficulty	– Saliency Augmentation – Subject(s) Size – Brightness – Intensity Contrast – Saturation		
Moderate Difficulty	– Background Distraction	– Rule of Thirds – Blurriness – Exif Metadata	
High Difficulty	– Segmentation – Efficient Interface		– Shape Convexity – Intuitive Interface

Table A.1: Implementation tasks organised by difficulty and risk (**core** first, and **extensions** later, time-permitting)

Appendix B

Hypothesis Testing

For evaluating the tool, I use images only from datasets 1 and 3, which contain well-defined subjects. This is because the high-level features are defined based on subjects, so are not designed for the other two sets. Apart from the comparison between segmentation enabled and disabled, I use only data from participants who had segmentation-derived features enabled. This tests the full power of the tool.

1. Does the Efficient tool extract some information that can discriminate aesthetic image quality? Yes

If the tool extracted no preference information, then it would be no better than randomly guessing the user's preferential order.

> Let X_1 be the random variable for the Spearman's Rank correlation of a person's Efficient tool sorting with their manual sorting, with population mean $\mathbb{E}(X_1) = \rho_1$, and μ_1 is the expected correlation of a manual sorting with a random sorting ($\mu_1 = 0$).
>
> The *null hypothesis*, that the tool is no better than random is
>
> $H_0 : \rho_1 = \mu_1.$
>
> The *alternative hypothesis*, that the tool is better than random is
>
> $H_1 : \rho_1 > \mu_1.$
>
> The sample mean correlation is $\overline{X_1} = 0.2358348968...$, with sample standard deviation $s_1 = 0.1725153108....$ We can assume the correlations to be approximately Normally distributed ($X_1 \sim N(\rho_1, \sigma_1)$). Therefore
>
> $$Z \approx \frac{X_1 - \rho_1}{\sigma_1}. \tag{B.1}$$
>
> Since we have a sample mean, $\overline{X_1} \sim N(\rho_1, \frac{\sigma_1^2}{n})$
>
> $$Z \approx \frac{\overline{X_1} - \rho_1}{\sigma_1/\sqrt{n}} \tag{B.2}$$

and we use s_1^2 as an approximation for σ_1^2. So

$$z_1 \approx \frac{\overline{X_1} - \mu_1}{s_1/\sqrt{n}} = 3.3485443... \tag{B.3}$$

is approximately from a standard Normal for H_0.

For a 95% significance using a Standard Normal table, the z-value such that $\mathbb{P}(Z < z_\alpha) = 0.95$ is $z_\alpha = 1.64$. Since $z_1 > z_\alpha$, the correlation is strong enough to reject H_0.

For a 95% significance, the Efficient tool was accurate enough in my study to suggest that it successfully extracts personal preference information. In fact, using the Normal table, it would have passed with a 99.96% significance.

2. Does the Intuitive tool extract some information that can discriminate aesthetic image quality? Yes

Using the same procedure as statistical test 1, but with the results for the Intuitive tool:

$\overline{X_2} = 0.2757348343...$
$s_2 = 0.1414554353...$

So $z_2 = 4.774716835...$. Since $z_2 > z_\alpha$, H_0 can be rejected. This means that, if we assume the null hypothesis to be true ($\rho_2 = 0$), then the probability that the sample mean is in this distribution is less than 0.05.

For the Intuitive tool (and a 95% significance), the average correlation is high enough to suggest that its resulting sortings are better than random guesses. In fact, this test would pass with 99.999% significance.

3. Do the segmentation-derived features help? Not enough data to conclude

In other words, is the tool more accurate with high-level features enabled?

In this unpaired mean-difference test, let X_{3S} be the random variable for the Spearman's Rank correlation of a person's Intuitive tool sorting with their manual sorting, given they had all features enabled, and let X_{3NS} be the same for low-level-only participants. If $\mathbb{E}(X_{3S}) = \rho_{3S}$ and $\mathbb{E}(X_{3NS}) = \rho_{3NS}$, then:

The *null hypothesis*, that the high-level features do not increase tool accuracy is

$$H_0 : \rho_{3S} = \rho_{3NS} \text{ or } \rho_{3S} - \rho_{3NS} = 0.$$

The *alternative hypothesis*, that the segmentation-derived features help is

$$H_1 : \rho_{3S} > \rho_{3NS} \text{ or } \rho_{3S} - \rho_{3NS} > 0.$$

The test data is:
$\overline{X_{3S}} = 0.2757348343$, $s_{3S} = 0.1414554353$
$\overline{X_{3NS}} = 0.1975609756$, $s_{3NS} = 0.143360754$

We can approximate to the standard Normal for H_0 using

$$Z \approx \frac{\overline{X_{3S}} - \overline{X_{3NS}}}{\sqrt{\mathrm{Var}(\overline{X_{3S}}) + \mathrm{Var}(\overline{X_{3NS}})}} \tag{B.4}$$

$$= \frac{\overline{X_{3S}} - \overline{X_{3NS}}}{\sqrt{s_{3S}^2/n + s_{3NS}^2/n}} \tag{B.5}$$

So, for this test, $z_3 = 0.9507750654....$

Since $z_3 < z_\alpha$, the difference is not large enough to reject H_0.

There is not enough evidence to suggest that the segmentation derived features help. The test would only pass with less than 82% significance. This is likely to be due to insufficient data/participants, since there was actually a difference. With enough data, the variance would have reduced, and any difference would be more significant.

4. Is aesthetic image quality subjective? Yes

Do people share a collective view of a ‘good’ or ‘bad’ image, or are there subjective differences?

Let X_4 be the random variable for the Spearman's Rank correlation between the manual sortings of two *different* people on the *same* set of images. If $\mathbb{E}(X_4) = \rho_4$, then:

The *null hypothesis*, that there is no subjectivity, is

$H_0 : \rho_4 = 1$ or $1 - \rho_4 = 0$.

The *alternative hypothesis*, that people do not necessarily agree with each other, is

$H_1 : \rho_4 < 1$ or $1 - \rho_4 > 0$.

The test data is:
$\overline{X_4} = 0.3242239468...$
$s_4 = 0.2237562923...$

We can approximate the standard Normal for H_0 using

$$z_4 \approx \frac{1 - \overline{X_4}}{s_4/\sqrt{n}} = 20.03336387... \tag{B.6}$$

Since $z_4 > z_\alpha$, the difference is far large enough to reject H_0.

There is enough evidence to suggest that (with 95% significance) people do not sort images in the same way. That is, they have their own subjective aesthetic image preferences. In fact, this test would have passed with a 99.9999% significance level. This is perhaps obvious, but is crucial for the motivation of my approach for the tool, which does not try to learn an average of many humans' image preferences.

5. Are people biased by the initial ordering of images? No

For the study I randomised the initial order of images for each participant, in the hope that any bias would be eliminated through averaging. Here, I test whether this bias existed in the first place.

> Let X_5 be the random variable for the Spearman's Rank correlation of a person's manual sorting with their original, random ordering. If $\mathbb{E}(X_5) = \rho_5$, then:
>
> The *null hypothesis*, that there is no bias, is
>
> $H_0 : \rho_5 = 0.$
>
> The *alternative hypothesis*, that people are biased by the original order, is
>
> $H_1 : \rho_5 \neq 0.$
>
> The test data is:
> $\overline{X_5} = 0.0404221388...$
> $s_5 = 0.1897202127...$
>
> We can approximate the standard Normal for H_0 using
>
> $$z_5 \approx \frac{\overline{X_5} - 0}{s_5/\sqrt{n}} = 0.9528415435... \tag{B.7}$$
>
> Since $z_5 < z_{\alpha/2}$ ($z_5 < 1.96$) for this two-tailed test, the sample bias is far too small to be significant to reject H_0.

These results suggest that people were not biased by the original image order, which means that it was probably not necessary to randomise them. However, I did not know this before the study was carried out.

6. Do peoples' image quality preferences change over time? Yes

Perhaps not only do people disagree with each other over image quality, but with themselves. I asked some participants to perform a manual sorting of their images a second time after doing the main study.

> Let X_6 be the random variable for the Spearmans' Rank correlation of a person's manual sorting with their second manual sorting of the *same* images, at a later time. If $\mathbb{E}(X_6) = \rho_6$, then:
>
> The *null hypothesis*, that there is no change, is
>
> $H_0 : \rho_6 = 1$, or $1 - \rho_6 = 0.$
>
> The *alternative hypothesis*, that peoples' preferences change, is
>
> $H_1 : \rho_6 < 1$, or $1 - \rho_6 > 0.$

The test data is:
$\overline{X_6} = 0.7434646654...$
$s_6 = 0.1118344195...$

We can approximate the standard Normal for H_0 using

$$z_6 \approx \frac{1 - \overline{X_6} - 0}{s_6/\sqrt{n}} = 3.973125943... \tag{B.8}$$

Since $z_6 > z_\alpha$, the correlation is small enough to reject H_0.

Even with 99.99% significance, there is sufficient evidence to suggest that peoples' image preferences do change over time. This further motives my approach, but reduces the reliability of the study because I assumed peoples' manual sortings to be ground truth data. In fact, preferences may have changed by the time participants used the tool.

7. Does using the tool bias peoples' image quality preferences? No

Following on from the previous test, I wanted to test whether it was perhaps the tool that biased the change in preference.

For this paired difference test, let $X_7 = R_{7b} - R_{7a}$ be the random variable for the difference between the Spearman's Rank correlations of a person's tool sorting and their second manual sorting, and of the tool sorting and their first manual sorting. If $\mathbb{E}(X_7) = \mu_7$, then:

The *null hypothesis*, that there is no bias, is

$H_0 : \mu_7 = 0.$

The *alternative hypothesis*, that preferences change towards the tool sorting, is

$H_1 : \mu_7 \neq 0.$

The test data is:
$\overline{X_7} = 0.01475922452...$
$s_7 = 0.09425509766...$

We can approximate the standard Normal for H_0 using

$$z_7 \approx \frac{\overline{X_7} - 0}{s_7/\sqrt{n}} = 0.2712185058... \tag{B.9}$$

Since $z_7 < z_{\alpha/2}$, the difference is too small to reject H_0.

The results suggest that (with 95% significance) the differences are too small to suggest that the tool biased peoples' preferences. The change shown in test 6 must have been caused by something else.

Appendix C

Project Proposal

Computer Science Tripos – Part II – Project Proposal

An investigation into selected segmentation-derived techniques for image quality assessment

Matthew Arnold (mpa29), Sidney Sussex College

Originator: Matthew Arnold

20 October 2016

Project Supervisor: Matthew Ireland (mti20)

Director of Studies: Dr John Fawcett (jkf21)

Project Overseers: Jean Bacon & Ross Anderson

C.1 Introduction

The aim of the project is to make a tool that allows humans to sort a set of images preferentially, according to some precomputed features. The program will compute a number of metrics related to general aesthetic image quality, which I categorise as either "high-level" or "low-level". The high-level features will be calculated using a segmentation-derived feature-extraction approach based on the one taken by Yeh et al. [22]:

1. Identify the position and size of the most salient region, and
2. Relate these quantities to generally accepted aesthetic 'rules'. Some examples are:

- Rule of Thirds – A popular rule-of-thumb in photography. The subject of the image should lie on one of the four intersections of lines breaking the image into three in both axes, 'power-points'.
- Region of Interest (ROI) size – How much of the image area is consumed by the subject.
- Simplicity – How distracting the background is from the ROI.
- Shape Convexity – How much the subject segment fills its convex hull.

The low-level features do not use this image segmentation approach but include overall brightness or intensity contrast. The tool will allow the user to adjust weightings for all of the features, and the combination of features and weights will give a way to sort the images.

This project has roots in the underlying field of aesthetic image quality analysis, which is concerned with evaluating how beautiful images are to humans, by analysing the pixel data. The task addressed by the project is a very specific one within this wider field, with emphasis on leaving the subjectivity with the humans that create it. This project is aimed at the application of trying to pick the best image(s) from a set taken of a particular subject. Note that all of the photographs in the set would be of the same object or scene. It will be useful in assisting a nature photographer, for example, who has taken many photographs of birds, perhaps with different camera settings, lighting, and compositions. He or she will want to sort through these, probably to pick the best few to look at in more detail, for framing, sale on their website or submission to a competition. Manually comparing all the photographs with each other would be time consuming, and inaccurate with larger datasets.

Machine learning is not feasible for this application, because the photographer would need a new set of training data each time he or she shoots a new subject or in a new environment. The evaluation will attempt to determine which of the high-level, or low-level features are more useful metrics for preferentially sorting the images, and whether such a tool is more useful than the existing method of manual sorting.

When humans look at an image, they use past experiences to recognise different objects in the image and use multiple contexts to form an aesthetic opinion. Since absolute general object recognition is hypothesised to be AI-complete and it's infeasible to assimilate all of the context required, it is currently impossible to write a program that can exactly determine whether or not someone will think a photo is good with 100% accuracy. Furthermore, aesthetic preferences are inherently subjective, with image quality being no exception, which makes the problem even harder to solve. We therefore aim to leave the subjectivity with the user in this project, by allowing the user to set the feature weightings rather than artificially inferring them.

Despite these limitations, a significant body of successful work has been done, and continues to be done, in the wider field. Early work tended to look at detecting types of errors in images based on characteristics of the human visual system [21]. However, more recently, there has been a lot of work on finding image features that follow general trends of human aesthetic judgement [4].

The output of the project will be a tool to help humans sort a set of photographs using metrics that are more meaningful than those that are conventionally available such as file name, sequence number, or date captured, as well as an evaluation of the features proposed. It will do this by sorting the images according to an overall 'quality' metric, computed as a weighted sum of the extracted metrics, in such a way that the weightings applied to each feature may be adjusted by the user. This will personalise the ranking according to their own interpretation of what makes an aesthetically pleasing image. The user will then be able to use this ranking as they wish, for example choosing the 'best' one or more, or discarding the 'worst'. From my investigations, there doesn't currently appear to exist an accessible open source tool to do this.

C.2 Starting point

I have read some introductory material from the field of aesthetic image quality analysis, have limited experience of amateur photography, but have no experience of image segmentation and processing algorithms/programming.

C.3 Substance and Structure of the Project

The project will look at different options for image segmentation algorithms, then use one alongside a saliency map to extract some high-level features from images, then produce a GUI to sort images using these features, and finally use the created GUI to evaluate whether or not several specific high-level image features, contrasted against a selection of simpler ones, are good discriminators of aesthetic image quality. The goal is to determine whether such features could be used to aid humans in preferentially sorting their images. Note that it is the usefulness of the features that's being evaluated, not the GUI.

The main work units will be:

1. Research image segmentation algorithms, in order to decide which one to implement. Write a document summarising my findings: which algorithm I chose, and why (and why I rejected others).

2. Familiarisation with C#.NET and the Visual Studio IDE, especially the tools available for handling images, and creating GUIs.

3. Implementation of the chosen segmentation algorithm.

4. Write code to compute a saliency map [1] to assist in identifying the main segment of the image.

5. Implementation of functions that extract at least two high-level, segmentation-derived features.
6. Implementation of functions that extract at least two low-level features: features that don't make use of the segmentation, including average brightness, contrast (for intensity and colour), blurriness, and average saturation.
7. Design and creation of a GUI (as a Windows Forms Application) which allows the input of a set of images, either calculates or loads previously computed metrics based on the implemented features, and sorts the images by a weighted sum of these values. The weightings for each feature should be able to be set and adjusted by the user (perhaps using sliders).
8. Plan the human evaluation. I aim to investigate whether or not the tool is useful for the application, and whether or not the segmentation-derived features help.
9. Carry out the human evaluation. From the human evaluation, I want to achieve scientific results about the effectiveness of the feature-based approach for image sorting. I will need to make sure that the participant sample size is large enough to obtain meaningful results. Also, the number of images used, in total, and per evaluation set, will need to be large enough in order to notice differences between results for different types or qualities of image sets. The users should be given instructions and the opportunity to practise using the GUI before evaluation proper, so that the results aren't perhaps skewed by the users learning to use a new tool.
10. Perform a numerical evaluation of my segmentation implementation. The numerical evaluation will make use of the Berkeley Segmentation Dataset [2], which contains human-annotated segmentations of images, such as below.

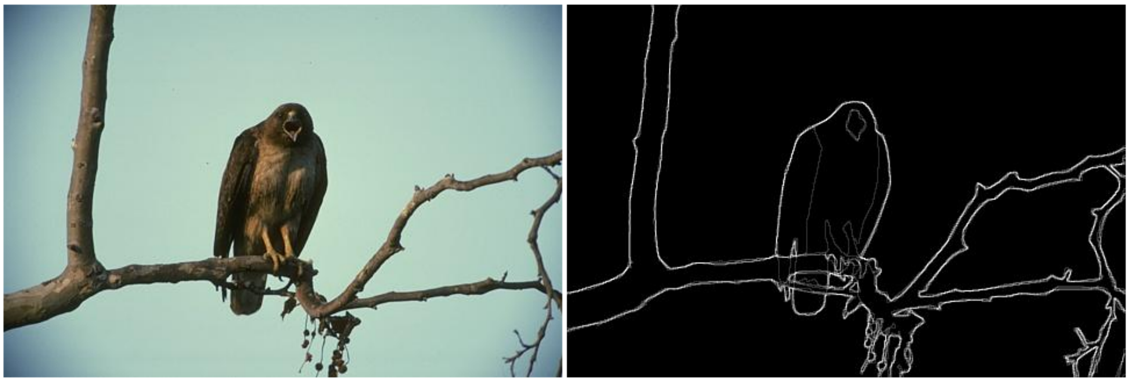


Figure C.1: An example from the Berkeley Segmentation Dataset. The original image is on the left, and on the right is an average of five human-annotated segmentations of the image.

11. Write the dissertation.

C.4 Success citeria

C.4.1 Core

The project will be considered a success if I:

- Implement an image segmentation algorithm. The actual algorithm to be implemented will be chosen in the research phase. My implementation will be tested by producing a visually similar result on five test images from the Berkeley Segmentation Dataset.
- Implement functions to extract values for two high-level features using the segmentation: Rule of Thirds, and Region-Of-Interest size. [TBC in research phase]
- Implement functions to extract values for two low-level features not derived from the segmentation: Intensity contrast, and Brightness. [TBC in research phase]
- Create a Windows GUI that can be used to sort an input set of images by a weighted combination of the implemented features, with each feature weight chosen by the user with some widget (such as a slider).
- Carry out a numerical evaluation of my image segmentation implementation.
- Carry out a human evaluation, comparing the usefulness of the segmentation-derived features to the others for sorting images.

C.4.2 Extensions

- Implement more functions to extract different features, both those using and not using the segmentation of the image. This includes Shape Convexity, Simplicity, Colour contrast, blurriness and average saturation. [TBC in research phase]
- Make the GUI such that the set of metrics is relatively easy to extend.
- Add an option that crops/rotates images so that the ROI obeys the Rule of Thirds better. This represents a major change in the scope of the project, so should only be attempted if there is a lot of spare time.

C.5 Plan of Work (Timetable and Milestones)

Below is my plan of work, broken up into fortnight time chunks from now until the dissertation hand-in deadline. I have put the most difficult implementation tasks first to minimise risk (e.g. extracting high-level features before low-level ones), and added slack blocks for the same reason.

06/10/16 – 19/10/16 (Mich W1&2)
Submission of Phase 1 Report form. Discussion with project supervisor and overseers. Arrange a schedule of regular meetings with my supervisor. Preliminary reading and writing of project proposal. If/once informal approval from overseers is gained, begin getting to grips with the C# language, and download an appropriate IDE.

Milestone: Send Phase 1 100-word report to overseers (Monday 10th Oct 3pm) [COMPLETED]
Milestone: Send Draft Project Proposal to overseers (Friday 14th Oct Noon).
Milestone: Hand in Final Project Proposal to Student Admin, signed by supervisor and DoS (Friday 21st Oct Noon).

20/10/16 – 02/11/16 (Mich W3&4)
Research phase. Practising C#.NET, including its image manipulation tools. Also look at editing and saving images (for making graphics for the dissertation). Read relevant papers and other material describing and comparing different image segmentation algorithms. Then decide which one to implement myself for the project.

Milestone: IDE installed and test program (e.g. sorting a list of numbers) compiling. (26/10)
Milestone: Write a test program to show I can input an image, and extract pixel data e.g. write a Gaussian blur program (02/11).
Milestone: Give a talk to fellow undergraduates about my image segmentation research. Also discuss research document with supervisor. (02/11)
Milestone: Image segmentation algorithm chosen. (02/11)

03/11/16 – 16/11/16 (Mich W5&6)
Implement the chosen image segmentation algorithm.

Milestone: Show supervisor a working example of an image segmentation. Test by producing a visually similar result on five test images from the Berkeley Segmentation Dataset (16/11)

17/11/16 – 30/11/16 (Mich W7&8)
Implement functions to extract the salient region of interest, and then the Rule of Thirds feature. Test the saliency map by comparing my program's results to those of Achanta et al. [1] on images that they use.

Milestone: Produce a small number of graphics showing original images, their segmentations, and salient regions (30/11).
Milestone: Have human evaluation form completed and sent off with supervisor (30/11).

01/12/16 – 14/12/16 (XmasVac W1&2)
Implement functions to extract remaining segmentation-based and non-segmentation-based features (at least two of each, defined in the Success Criteria). Test the low-level features by comparing values produced by my program on the same image after the (e.g. contrast or brightness) has been altered using an image editing application. Test the high level features by taking photographs of the same subject, but moving the subject

in the frame towards or away from a "power-point". If I'm ahead at this point, consider implementing more than two of each category of feature, as described in the Extensions section. Apply feature extraction to chosen evaluation and testing images, and save the results to persistent storage.

Milestone: Produce a small number of graphics showing original images, their segmentations and salient regions, and values for extracted features. Compare similar images (14/12).

15/12/16 – 04/01/17 (XmasVac W3,4&5)
SLACK or extensions.

05/01/16 – 18/01/17 (XmasVac W6&7)
Design and create the GUI as a Windows Forms Application. It should contain a list of images to be sorted, as well as sliders for each of the features. Extension: Make the GUI such that the set of extracted metrics is relatively easy to extend.

Milestone: Show supervisor the GUI working in the first project meeting after the holidays.

19/01/17 – 01/02/17 (Lent W1&2)
Plan the human evaluation, as well as writing the progress report.

Milestone: Hand in Progress Report (due Friday 3rd Feb at Noon).

02/02/17 – 15/02/17 (Lent W3&4)
Carry out the human and numerical segmentation evaluation. The exact structure of this block will be dependent on how and when the human evaluation will be distributed or carried out (which is to be decided in the previous block). Segmentation evaluation can essentially be done in the gaps.

Milestone: Carry out numerical image segmentation evaluation using the Berkeley segmentation dataset. (08/02)
Milestone: Carry out the human evaluation planned in the previous block. (15/02)
Milestone: End of all implementation and evaluation (i.e. all core implementation goals achieved, and a sufficient amount of data gathered from the human evaluation. "Sufficient" can be decided in the evaluation planning block) (15/02).

16/02/17 – 01/03/17 (Lent W5&6)
Start writing the dissertation. Draft the implementation section and send to supervisor.

Milestone: Hand in first draft of the Implementation section to supervisor for feedback. (01/03)

02/03/17 – 15/03/17 (Lent W7&8)
Integrate supervisor's feedback and draft the Introduction and Preparation sections of the dissertation.

Milestone: Hand in first draft of the Introduction and Preparation sections to supervisor for feedback. (15/03)

16/03/17 – 29/03/17 (EasterVac W1&2)
Integrate supervisor's feedback and draft the Evaluation and Conclusion sections of the dissertation.

Milestone: Hand in first draft of the Evaluation and Conclusion sections to supervisor for feedback. (29/03)

30/03/17 – 12/04/17 (EasterVac W3&4)
Integrate supervisor's feedback and finish first draft of the dissertation.

Milestone: Hand in first draft of the dissertation to supervisor (12/04).

13/04/17 – 26/04/17 (EasterVac W5&6)
Wait for supervisor's feedback and revise. Integrate final supervisor comments.

Milestone: Send dissertation to supervisor for final approval.

27/04/17 – 10/05/17 (Easter W1&2)
SLACK and revision for exams.

11/05/17 – 17/05/17 (Easter W3)
SLACK and revision for exams.

Milestone: Hand in dissertation (deadline noon on Friday 19th May at Noon).
Milestone: Make sure supervisor's form is handed in by Wed 24th May (4pm).

C.6 Resource Declaration

In order to test my implementation, and to carry out the evaluation, I shall need a set of images. These will be sourced from my project supervisor's and my own personal collections. No images containing peoples' faces will be used, unless I gain consent from those people.

I plan to use my personal laptop for writing code and storing images. A version control system will be used and cloud storage will be used for backups, and regular backups will be taken on an external hard drive. To ensure that the backups are working, I will do a test-recovery after one of the earlier backups is taken. I accept full responsibility for my machine and I have contingency plans to protect myself against hardware and/or software failure. In the unlikely and unfortunate case of my laptop dying, I would switch to MCS, using the backups, until I can buy a replacement computer.