Kyffin Williams: Digital Analysis of Paintings

Report Name	Progress Report
Author (User Id)	Alexander D. Brown (adb9)
Supervisor (User Id)	Hannah M. Dee (hmd1)
Module	CS39440
Date	November 12, 2012
Revision	0.1
Status	Draft
Word Count	?

Contents

List of Figures

List of Tables

1 Project Summary

Sir John "Kyffin" Williams was a landscape painter from Wales whose work was predominantly based in Wales and Patagonia. His work, with associated metadata collected by Gareth Lloyd Roderick and the National Library of Wales, allows for some interesting analysis; particularly that of temporal or geological data for a given painting.

Temporal analysis will be the focus of this project as it allows for a diverse range of techniques; from statistical analysis of RGB values of the paintings to looking at the length and style of paintbrush strokes.

Whilst it would be nice to be able to predict the age of a painting with no known year, it is far more interesting to try to guess the year of paintings where the date is known. This allows for cross-validation of the techniques employed to guess the year. This project will use leave-one-out cross-validation for simplicity as the data set is small enough to allow this computationally expensive technique.

2 Current Progress

2.1 Library and Language Research

The first step in the project was to research into the numerous image processing/computer vision libraries available for use with the aim to find the most suitable library to use constantly for the rest of the project. Because of this the library had to be easy to install and use, as well as having all the necessary features which would be used in this project.

The choice of library would also have a knock-on affect on the choice of language this project would be written in. To this end I downloaded each of the libraries in turn, created a simple application to perform blur on a simple image to test their use and documentation. From this I gained a good insight into how easy that library would be to perform more complex tasks and how confident I felt with the language(s) the library had bindings for.

Table 1 shows an overview of all the libraries I have currently considered and experimented with.

Library	**Platform**				**Language(s)**	**Example**
	Windows	Mac	Linux	Android		
OpenCV	✓	✓	✓	✓	C, C++, Python	Figure 1
FIJI	✓	✓	✓		Java	Figure 2
IVT	✓	✓	✓		C++	Figure 3
VXL	✓	✓	✓		C++	Figure 4

Table 1: Comparison of image processing/computer vision libraries.

Of these libraries OpenCV was the easiest to work with. OpenCV also boasts a wide range of features, all of which are well documented. FIJI provided a lot of high-level functionality, but for use as a library it quickly became unwieldy and was difficult to find the correct classes just for the simple task of blurring an image. On a side note, I only managed to get the blurring outputting a greyscale image in the short period of time I spent using FIJI.

IVT was somewhat similar to FIJI in that it had a good range of high-level features, but was less impressive as a library. Despite following the example code I struggled to compile my own example and eventually gave up trying to get a working binary due to time constraints.

Having decided to go with OpenCV I was then faced with a choice of languages. OpenCV runs natively with C++ and has bindings for Python. Of these two languages I am slightly more familiar with C++. I am also used to the syntax from programming Java continually for three years.

However, I wanted to take a rapid prototyping approach with this project; constantly adding new modules each week to build up a better and better system. C++ seemed too heavyweight for this approach, whilst Python seemed designed for it.

Another consideration I had was that the results of any analysis could take any form at all, from an array of numbers to a complex data structure returned by OpenCV. Trying to work around this with a statically typed language such as C++ could end up being difficult, especially when trying to keep a object-orientated approach.

With a dynamically typed language such as Python it's a lot easier to pass these sorts of objects so long as the functions you pass to are expecting the right sort of object. There may be some parts of the project which I might consider writing in C++ after prototyping in Python. An example of this would be if I wrote an algorithm which located strokes across the painting, as it may be used in other projects, not just my own.

2.2 Other Tools

Having used git for several personal projects I was keen to continue using it for my version control. GitHub was a convenient hosting company to go with as they provide students with free private repositories and have very high uptime. It also means I am able to easily work on any system without difficulty but with the assurance of security. This is preferable to hosting my own repositories and running the risk of losing a lot of work.

2.3 Design

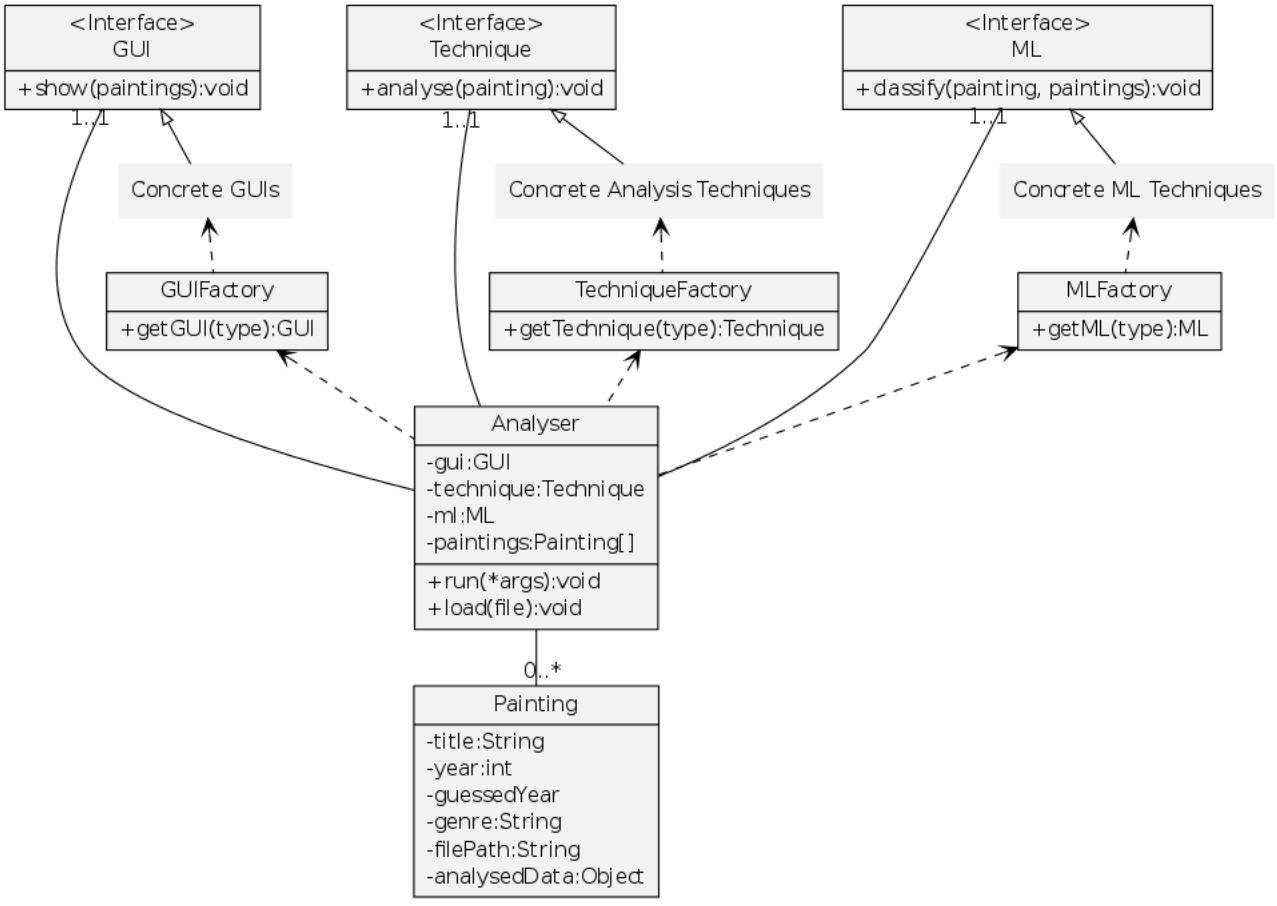

Figure 5: Initial Design

```
import cv

im = cv.LoadImageM("tux.png")
blur = cv.CreateMat(im.rows, im.cols, im.type)
cv.Smooth(im, blur, cv.CV_BLUR, 9)
cv.SaveImage("tux_blurred_opencv.png", blur)
```

Figure 1: Using OpenCV to blur an image

```
package fiji;

import ij.IJ;
import ij.io.FileSaver;
import imagescience.feature.Smoother;
import imagescience.image.ColorImage;
import imagescience.image.Image;

public final class Blur {
        public static final String IMG_PATH = "tux.png";
        public static final String OUTPUT_PATH = "tux_fiji.png";

        public static void main(String[] args) {
                final Image image = new ColorImage(IJ.openImage(IMG_PATH));
                final Smoother s = new Smoother();
                final Image blurred = s.gauss(image, 9f);
                final FileSaver fs = new FileSaver(blurred.imageplus());
                fs.saveAsPng(OUTPUT_PATH);
        }
}
```

Figure 2: Using FIJI to blur an image

```
#include <stdio.h>
#include "Image/ByteImage.h"
#include "Image/ImageProcessor.h"

int main(int argc, char **argv) {
        if(argc != 2) {
                fprintf(stderr, "Usage: ivt input output\n");
                return 1;
        }

        CByteImage in;
        CByteImage out;

        if(!in.LoadFromFile(argv[0])) {
                fprintf(stderr, "Unable to load file %s\n", argv[0]);
                return 2;
        }

        out = CByteImage(in);
        ImageProcessor::GaussianSmooth(&in, &out, 1.0f, 3);
        if(!out.SaveToFile(argv[1])) {
                fprintf(stderr, "Unable to save to file %s\n", argv[1]);
        }
}
```

Figure 3: Using IVT to blur an image

Figure 4: Using VXT to blur an image

An initial UML class diagram is depicted by figure 5. This design will allow additional techniques, both analysis and machine learning based, to be added easily. Command line arguments, or later a GUI front-end, will be used to specify which techniques should be used.

The GUI elements depicted are used for visualising the analysed data or results of the machine learning algorithms. An example of this is a graphical representation of the colour space averages (see figure 7).

2.4 Colour Space Statistical Analysis

With the design solidified, I begun to work on the statistical analysis of digital images. The easiest of these techniques is to read the image pixel by pixel, taking the various intensities at that pixel, then averaging them out over the whole image.

The most common colour space used is RGB (Red, Green, Blue), where each pixel holds three different intensities, one for each colour with a value of 0 to 255.

Using this analysis I plotted a graph of all paintings with a known year against the different intensities (see figure 6). As can be seen from this graph, the RGB values tend to fluctuate without much correlation between year.

Another popular colour space used in image processing is HSV (Hue, Saturation, Value) as it shows variations in colour, saturation and brightness separately (unlike RGB where all three values are affected by changes in brightness). It was a simple matter of changing the existing code for RGB so that it would analyse HSV values instead.

I also decided it would be an improvement to average all values in the same year together to help show changes as time goes on (see figure 7).

2.5 Histogram Analysis

With colour-space analysis complete, the next sensible step was to start generating colour histograms from Kyffin Williams paintings.

A histogram is a nice way of displaying the distribution of colour in a given image and is therefore a more useful representation of a painting than colour-space averages.

As with before, histograms can be taken from different colour spaces, currently only the program can only generate RGB histograms, but it is trivial to include HSV histograms too.

2.6 Nearest Neighbour Classification

Having finished the analysis techniques, then next part was to create a simple classification algorithm. The simplest of which is 1-Nearest Neighbour, that is; take the "Nearest" training example to the example to be classified and classify the year of this example with the year of the nearest example (see figure 8).

This simple algorithm can be expanded to be a K-Nearest Neighbour algorithm without too much effort (see figure 9).

2.7 Distance Measures

Working out the "Nearest" example to a given painting requires some way of measuring distance between the two.

One very simple way of doing this is Manhattan distance (figure 10). This distance measure can be improved further to euclidean distance (figure 11) without too much effort. There are numerous other methods of measuring the distance of points in space, including those which focus on the distance between colour histograms.

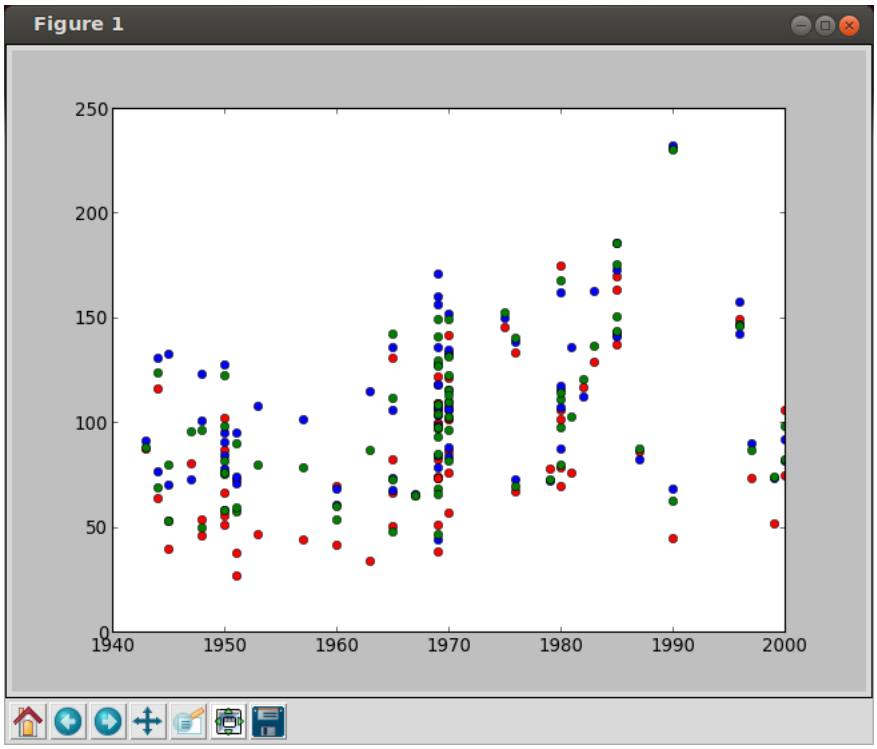

Figure 6: Mean RGB intensities by year.

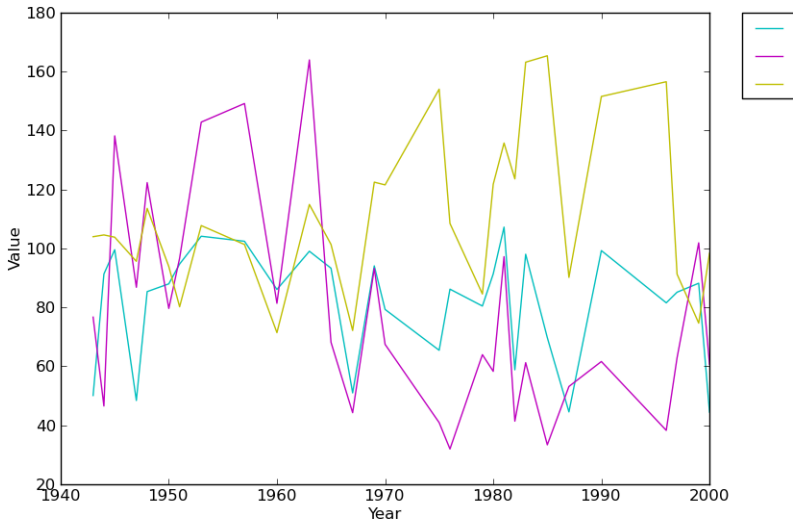

Figure 7: Mean HSV intensities by year.

$bestDistance \leftarrow \infty$
for $x = 1 \rightarrow X$ **do**
  if $distance(a, x) < bestDistance$ **then**
    $bestDistance \leftarrow distance(a, x)$
  end if
end for

Where:
a is the example to classify.
X is the list of all training examples.

Figure 8: 1-Nearest Neighbour

$kBest \leftarrow []$
for $x = 1 \rightarrow X$ **do**
  $cur \leftarrow distance(a, x)$
  for $k = 1 \rightarrow K$ **do**
    if $cur < kBest[k]$ **then**
      $temp \leftarrow kBest[k]$
      $kBest[k] \leftarrow cur$
      $cur \leftarrow temp$
    end if
  end for
end for

Where:
a is the example to classify.
X is the list of all training examples. K is the number of nearest neighbours to check.

Figure 9: K-Nearest Neighbour

2.8 Meetings with Gareth Lloyd Roderick

At current I have had one meeting with both Lloyd and Hannah to discuss the current progress of the project and where we see the project progressing to. Lloyd was impressed by the current state of the project and was looking forward to see where it was leading to.

We also discussed the potential of getting a paper published out of this project.

2.9 Research into Stroke Analysis

Stroke analysis is one of the main goals for this project. It is quite apparent from looking at Kyffin Williams' paintings that his brushstrokes change over time, his early work having lots of smaller strokes over the canvas to large bold strokes in his later work.

The first paper I found relating to the analysis of brushstrokes involved moving a circular filter across the whole painting to find the ridges of strokes, then to fill any unbroken areas. They then shrunk these areas to a single pixel line and fitted a n^{th} order polynomial to this line. [1]

3 Planning

Your Bibliography - REMOVE for final version

$$d = \sum_{x=0}^{X} |a_x - b_x|$$

X: All dimensions present in both a and b.
a: The first point.
b: The second point.

Figure 10: Manhattan Distance

$$d = \sqrt{\sum_{x=0}^{X} (a_x - b_x)^2}$$

X: All dimensions present in both a and b.
a: The first point.
b: The second point.

Figure 11: Euclidean Distance

Annotated Bibliography

[1] I. E. Berezhnoy, E. O. Postma, and H. J. van den Herik, "Authentic: Computerized brushstroke analysis," in *Multimedia and Expo, 2005. ICME 2005. IEEE International Conference on*. IEEE, July 2005, pp. 1586–1588. [Online]. Available: http://dx.doi.org/10.1109/ICME.2005.1521739

[2] J. Li, L. Yao, E. Hendriks, and J. Z. Wang, "Rhythmic brushstrokes distinguish van gogh from his contemporaries: Findings via automated brushstroke extraction," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 34, no. 6, pp. 1159–1176, June 2012. [Online]. Available: http://dx.doi.org/10.1109/TPAMI.2011.203

[3] S. J. D. Prince, *Computer vision : models, learning, and inference*. Cambridge University Press, 2012. [Online]. Available: http://www.computervisionmodels.com/ 9781107011793.

[4] D. Stork and M. Duarte, "Computer vision, image analysis, and master art: Part 3," *IEEE Multimedia*, vol. 14, no. 1, pp. 14–18, 2007. [Online]. Available: http://dx.doi.org/10.1109/MMUL.2007.6

[5] D. G. Stork, "Computer vision and computer graphics analysis of paintings and drawings: An introduction to the literature computer analysis of images and patterns," ser. Lecture Notes in Computer Science, X. Jiang and N. Petkov, Eds. Berlin, Heidelberg: Springer Berlin / Heidelberg, 2009, vol. 5702, ch. 2, pp. 9–24. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-03767-2_2