Kyffin Williams: Digital Analysis of Paintings

Final Report for CS39440 Major Project

Author: Alexander David Brown (adb9@aber.ac.uk)
Supervisor: Hannah Dee (hmd1@aber.ac.uk)

22nd April 2012
Version: 0.1.1539 (Draft)

This report was submitted as partial fulfilment of a MEng degree in
Software Engineering (G601)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, UK

Declaration of originality

In signing below, I confirm that:

- This submission is my own work, except where clearly indicated.

- I understand that there are severe penalties for plagiarism and other unfair practice, which can lead to loss of marks or even the withholding of a degree.

- I have read the sections on unfair practice in the Students' Examinations Handbook and the relevant sections of the current Student Handbook of the Department of Computer Science.

- I understand and agree to abide by the University's regulations governing these issues.

Signature ....

Date ....

Consent to share this work

In signing below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Signature ....

Date ....

Acknowledgements

Abstract

Computer vision has addressed many problems in art, but has not yet looked in detail at the way artistic style can develop and evolve over the course of an artists career. The project takes a computational approach to modelling stylistic change in the body of work amassed by Sir John Kyffin Williams, a nationally renowned and prolific Welsh artist. Using images gathered from catalogues and online sources, a leave-one-out methodology is used to classify paintings by year; despite the variation in image source, size, and quality significant correlations between predicted year and actual year can be obtained, and the age of a painting is able to be classified within 15 years, for around 70% of the dataset. The incorporation of expert knowledge within this framework by consdering a subset of paintings chosen as exemplars by a scholar familiar with Williams' work has also been investigated.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF LISTINGS

Chapter 1

Background & Objectives

This section outlines the background of this project, mainly focusing on how the project came to be and why it is an interesting topic to research into, as well as discussing some of the interdisciplinary work performed with art historians.

Carrying on from this, further sections detail some of the existing work that has already been performed on digital images of artwork and other methods which are useful in the context of this project.

The main objectives and goals of this project are defined in the final sections and begin to outline some of the analysis and classification techniques which are used in the project.

1.1   Sir John Kyffin Williams

Sir John Kyffin Williams (1918-2006) was one of the predominant figures in Welsh art of the twentieth century. He was advised by a doctor in the British Army to take up something which would not tax his brain, such as painting, after failing a medical exam due to epilepsy. Kyffin – as he was almost universally known in Wales – studied at the Slade School of Art and worked as an art master at Highgate School, before returning to live on his native Anglesey in 1973. He was a prolific painter and once claimed to have painted "two pictures per week when in London, and three per week when in Wales.' [25, p.209] With a career spanning from the mid-1940s to approximately 2004, this rate amounts to a large body of work.

His style evolved from a very representational style to something more expressive, which retained representational qualities: computer scientists would say that the paintings became more *blocky*; whilst art historians say that his landscapes are almost constructed with swathes of textural paint. His was a style characterised by thick impasto paint, applied almost exclusively with palette knife, although the application technique appears to change over time. This development of style led to the question: "is it possible to date the pictures from images alone?"

Although also a portrait painter, Williams is primarily known for his landscape paintings of north west Wales and Anglesey. While his technique and style changed over the years, his landscapes in oil are instantly recognisable, often featuring bold chunks of colour, and various points during his career bold black outlines to figures and landscapes features. Greens, browns and greys often form the palette of his paintings of the Welsh landscape. These colour selections seem appropriate for the artists claim that melancholy, derived from the "dark hills, heavy clouds and enveloping sea mists", is a national characteristic of the Welsh. [25] This combination of colour selection and technique seems appropriate for the depiction of the areas where he painted. Many of his most successful paintings are said to have a "dark quality" in depicting "rain lashed hillsides,"

Figure 1.1: "*Snowdon, the Traeth and the Frightened Horse*", Sir John Kyffin Williams, 1948: note curved strokes, rather than blocky application

and it was this darkness which "makes his landscapes so distinctively Welsh." [13]

As his life progressed Williams' epilepsy grew steadily worse, especially when exposed to bright light. As a result most of his paintings are of overcast Welsh landscapes and appear to become visibly darker over time [16].

The aesthetic of Williams' Welsh landscapes is contrasted by the paintings he made following a trip to Patagonia to paint the landscape and people of the Welsh communities there in 1968 as part of a Winston Churchill Foundation scholarship. The colours and application of paint in pictures produced in following this journey (such as *Lle Cul* (Figure 1.3), *Henry Roberts*, *Bryngwyn Patagonia*, *Euros Hughes Irrigating his Fields*, all 1969, National Library of Wales) differ starkly from paintings of Welsh landscapes, incorporating pinks, purples and oranges. This contrast, combined with the fact that the Patagonian pictures were produced during a definite period of time has reinforced our interest in the analysis of the formal qualities of pictures from different collections remotely, using digital images.

Williams' work is well represented in public collections in Wales (particularly at the National Library of Wales, the National Museums and Galleries of Wales and Oriel Ynys Môn, Anglesey). His pictures, often depicting the landscape and people of north-west Wales were also tremendously popular with the art buying public. Of the 325 paintings by Williams in public collections in the UK listed on the BBC/Public Catalogue Foundation's "Your Paintings" website, 212 of them are in the collections of the National Library of Wales [3]. Many of these paintings were bequeathed to the Library as part of a larger bequest by the artist (including works on paper and other archival material). Many of the pictures which came to the library from the artist's studio had little in the way of meta-data, and as such have been catalogued with large date-ranges estimating the dates of production. This uncertainty in meta-data is another motivating force behind the current project.

1.1.1 Taking a digital humanities approach to art history

Digital humanities is an established area of research that brings together digital content, tools and methods in order to address and create new knowledge across the disciplines. Digital humanities approaches can be seen in two distinct types of inquiry. The first is to carry out *traditional* humanities research more effectively or efficiently, by applying computational methods or approaches to digitized humanities sources (originally text, image, or audio-visual content from archives or

Figure 1.2: "*Above Carneddi, No. 2*", Sir John Kyffin Williams 1985: note much blockier style and changed use of colour

libraries). Using John Unsworths definition of "scholarly primitives" [24] digital humanities scholarship customarily involves the use of digital tools and methods for discovering, annotating, comparing, referring, sampling, illustrating, or representing humanities data. A classic example of this sort of work would be the use of concordances and other computer-based analysis of digitized primary sources that have been processed by optical character recognition software to count, classify, or interpret digital texts (see, for example, the Historical Concordance of the Welsh language [1]). The second strand of digital humanities inquiry is the development of new research questions that can only be developed through the synthesis of digital content, tools and methods: work that would have otherwise been unimaginable [17]. This type of research is by necessity multi-disciplinary, drawing together expertise to be found across humanities, scientific and engineering disciplines, as well as involving content experts from libraries, archives and museums. However, in order to be truly transformative, this type of research must also be interdisciplinary.

The National Library of Wales now has a research programme in digital collections, which is a forum for investigation into the digital collections of Wales in collaboration with academics and students at universities in Wales and beyond, in order to develop new research based around the digital content created by the Library [2]. The research project described in this article is an example of a digital humanities collaborative venture, bringing together digital humanists, art historians, and computer scientists. The results of this research have value across all these groups. Arts historians are able to better investigate a large corpus of digital paintings through the application of computer science approaches to this content, and computer scientists are able to configure new approaches in imaging to working with a complex humanities dataset.

Figure 1.3: "*Lle Cul, Patagonia*", Sir John Kyffin Williams 1969

1.2 Interdisciplinary work with the National Library of Wales

This project originated through conversations between Hannah Dee and Gareth Lloyd Roderick on the topic of the digital image processing of art; this led to Hannah building a small application to show a Gaussian mixture model of Williams' work with the ultimate intention of being able to geographically determine the location of a picture, especially those with a well defined setting especially those painted at Crib Goch, a good example of which is shown in Figure 1.4.

As has been discussed already, it becomes very clear to the human eye that Williams' style changes dramatically as time progresses and, whilst it would be interesting to try and compare painted images to photographs of locations, the feasibility of being able to do so, especially given that artists tend not to remain brilliantly faithful to the subject of their painting, is just so low that it would have been very difficult to get any meaningful results. This makes temporal classification a much more appealing subject to research into.

To help with this Lloyd had produced a spreadsheet of all Williams' work known to the National Library of Wales (NLW), including several collections over the country, but with the majority coming from the NLW collection. It was a part of Williams' will that the contents of his workshop was to be donated to NLW on the condition that any destroyed or defaced work was not to be displayed publicly.

Therefore this spreadsheet is the most comprehensive documents on Williams' work. Unfortunately a lot of Williams' work does not have a lot of information on the dates on which it was painted, Lloyd had made some good attempts at predicting the year based on when the painting first appeared in exhibitions, so not all paintings have a single year associated with them. This spreadsheet does contain a lot of other useful meta-data as well as the year or year range in which the work was created, including canvas size and painting type. Not all of this meta-data was useful or relevant to temporal analysis, but is useful to the general study of Williams' work.

Figure 1.4: "*Crib Goch*", Sir John Kyffin Williams

1.2.1 Meetings with members of the National Library of Wales

There have been two meetings to discuss the interdisciplinary work with members the National Library of Wales as well as a tour around the stores to view the NLW collection of Williams' work.

1.2.1.1 Meeting of the 31st October, 2011

This meeting was between Lloyd, Hannah and I to discuss the state of the project particularly the aims Hannah and I had for the project and Lloyd's expectations and aspirations for the joint work. Notably both our aspirations included being able to publish a paper before the submission of the project, or, failing that, over the summer following the submission of the project.

This also provided an opportunity to discuss some of the artistic details with a more knowledgeable source which helped with the implementation of certain analysis techniques as well as discussing more elements of Williams' life.

1.2.1.2 Meeting of the 18th February, 2012

This meeting was between Lloyd, Hannah, Lorna M. Hughes[1] and I, again to discuss the state of the project, but was also used to discuss finer details of the potential of publishing a paper. Lloyd was also able to provide a cleaner version of his spreadsheet.

---

[1]Lloyd's PhD supervisor

More artistic details were discussed in the meeting, particular retaining to the application of expert knowledge to the temporal classification of Williams' work. It was shortly after this meeting that Dr. Paul Joyner was contacted to provide artistic exemplars for many years in Williams' career.

Some other interesting and useful points were brought up with regards to Williams' work; such as the artistic cut-off point between early and late in Williams' career is around 1973. The theory that the canvas size would be proportional to the year it was painted is due to the expense of paint on larger canvases.

It was also mentioned that the validity of some dates may be dubious, it might be the case that certain paintings were said to be painted later than they actually were to increase the sale price.

1.3 Existing Work

1.3.1 Edge-Orientated Gradients

As Williams' work is highly textural, looking at the edge orientation of the image is likely to be a valuable technique to use.

One technique recommended by Hannah was to look at Histogram of Orientated Gradients (HOG). The suggested paper outlined the use of grids of HOG descriptors to improve the feature set for robust visual object recognition [11]. As it significantly improves the feature set it seems sensible to try and implement it as a technique to experiment with a non-colour-based approach.

The approach involves quite a few separate steps, only some of which are relevant to the project:

1. Gamma and colour normalization. Grayscale, Red, Green, Blue (RGB) and L, a, b Colour Space (LAB) spaces were used. RGB and LAB give similar results. Grayscale reduced performance less that square root gamma compression, but not as much as log compression.

2. Compute gradients. Often the simplest are the better here; Gaussian smoothing followed by discrete derivative masks (e.g.: Figure 1.5, Figure 1.6), etc. For colour this was done for each channel, and take the one with the best gradient (the largest norm).

3. Spatial and Orientation binning:

    - Spatial binning is done by splitting the images into cells which can be rectangle or radial.

    - Orientation binning are spaced equally between either 0-180 "unsigned" or 0-360 "signed" bins.

4. Normalisation and Descriptor Blocks. Gradients vary over foreground, background, etc. Typically the blocks were overlapped so that each scalar response contains several components.

5. Pass a detector window across the image.

6. Run through a Linear Support Vector Machine (SVM) to classify the image.

Obviously, when applying this as an analysis technique to paintings, there are some points which are completely irrelevant. Passing a detector window and running through a Linear SVM are the obvious two. Normalisation is another unneeded step as time complexity isn't a problem

$$\begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

Figure 1.5: Example of a 1 by 2 Discrete Derivative Mask

$$\begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

Figure 1.6: Example of a 1 by 3 Discrete Derivative Mask

which needs to be thoroughly considered, as long as the analysis completes within a reasonable amount of time.

This leaves the act of computing the gradients (again, this can be done without Gaussian smoothing as that reduces accuracy) which should have been implemented by earlier techniques. Binning by Rectangular Histogram of Orientated Gradients (R-HOG) or Circular Histogram of Orientated Gradients (C-HOG) descriptors, which may prove to be one of the more difficult parts of implementing this technique.

R-HOG descriptors have similarities to Scale-Invariant Feature Transform (SIFT) descriptors, but are used quite differently; SIFT descriptors are optimised for sparse baseline matching whilst R-HOG descriptors are optimised for the dense and robust coding of a spatial form. The size of the descriptor affects performance when using R-HOG. For paintings it may turn out that a size relating to the original size of the painting is a good way of getting around this problem.

C-HOG descriptors become more complex still. They are similar to Shape Contexts [5], but differ in one key aspect: in C-HOG descriptors each spatial cell holds a stack of gradient-weighted orientation cells. This is instead of an orientation-independent edge-presence count which Shape Contexts use.

According to the author it is better to think of C-HOG descriptors as an advanced form of centre-surround coding, as small descriptors with very few radial bins gave the best results.

Local contrast normalisation can be performed to help against local variations in the illumination of foreground and background.

It would seem that both R-HOG and C-HOG descriptors are designed more for the detection window rather than analysis technique. This may make them less useful and result in an implemented technique being just a simple histogram of edge orientations.

1.3.2 Brush-stroke Analysis

Stroke analysis is one of the main goals for this project. It is quite apparent from looking at Williams' paintings that his brush-strokes 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 found relating to the analysis of brush-strokes involved moving a circular filter across the whole painting to find the ridges of strokes, then filling any unbroken areas. They then shrunk these areas to a single pixel line and fitted a nth order polynomial to this line [6]. This method seems fairly simplistic, but could be an interesting first step, but as it is more focused on authenticating paintings it may be of limited use.

Another method for stroke analysis has been published in the IEEE Transactions on Pattern Analysis and Machine Learning journal. This method is far more complex, but is able to extract

and label individual brush-strokes. An interesting part of their findings was the ability to date some of Van Gogh's paintings to a known period in his career [20].

This method involves performing edge detection of the painting followed by an edge linking algorithm which aims to remove small, noisy edges and to trace every edge. With this they then perform enclosing, as strokes may not be complete this stage also aims to fill in missing gaps of strokes and to fill these in within a certain tolerance.

The algorithm then decides if a stroke really is a painted stroke, if the stroke is completely enclosed, isolated from other non-edge pixels and forms a connected component then it is likely that it is a proper brush-stroke and is extracted. The edge pixels are used as the background and the non-edge pixels as the foreground, this is the process of labelling the brush-stroke.

For each of these labelled candidates, a heuristic function is used to threshold any brush-strokes that are either too long or too short, these strokes are discarded. These strokes are then considered to be candidates if they are not significantly branched, the stroke is not too wide (this may change for Williams as he used a pallet knife rather than a brush) and the brush-stroke is not too big or small.

Separately, the image is then segmented using k-means clustering by RGB values. This clustering algorithm is applied several times, lowering the tolerances for distance within a cluster. Connected components as a result of this clustering and have noise reduction performed upon them. Finally, the two types of brush-strokes are combined.

This technique may need some changing to account for Williams' use of a pallet knife, but the overall principals of this technique should work with Williams' paintings.

1.4   Analysis Objectives

Analysis is one of the biggest sections of this project and involves creating techniques which allow the comparison of paintings. This analysis will produce a result which can be used in a classification algorithm.

Typically it is expected that this produces some form of high-dimensional state space in which each painting is a point in the state space. From this state space the distance between one painting and another can be easily resolved using a distance measure. Common distance measure include Manhattan distance (1.1), euclidean distance (1.2) or a distance measure more specific to the state space should it be needed (e.g.: Chi Squared (χ^2) for histograms).

$$d_1(\mathbf{p}, \mathbf{q}) = \|\mathbf{p} - \mathbf{q}\|_1 = \sum_{i=0}^{n} |p_i - q_i| \tag{1.1}$$

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=0}^{n} (q_i - p_i)^2} \tag{1.2}$$

1.4.1   Colour-space Analysis

The simplest way of analysing a digital image is to look at the colours which it consists of. Doing this is relatively simple; each pixel has a set of values defining the colour of that point, getting something meaningful from this is less simple.

The simplest strategy is to perform some form of statistical analysis on each painting then use this for classification. Several good and computationally cheap options exist for this; mean

(1.3) and standard deviation (1.4), are some good examples which often come predefined in image processing and computer vision libraries.

$$\mu = \frac{1}{N} \sum_{i=1}^{N} x_i \tag{1.3}$$

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_i - \mu)^2} \tag{1.4}$$

The representation of colour is another important factor, an RGB representation will have all three values change if there are many changes in brightness of the colours whilst a Hue, Saturation, Value (HSV) representation will only have a single value change.

Therefore, an object of this section should be to explore different colour models and statistical methods which can be applied to them.

Another useful technique which should be investigated early into the project are image histograms. These histograms plot the distribution of colour across an image and are therefore a very powerful method of analysing an image, especially for comparison. As with statistical analysis, histograms will be largely effective by colour model.

1.4.2 Texture Analysis

As Williams' work is very textural, it follows that a main part of the analysis should focus around the texture of his paintings. Unfortunately for this section, it was not possible to get a hold of 3-dimension models of Williams' paintings. This would have given more detailed results but would have taken up more time than was available for the project. Instead this has been suggested as a topic for future research.

Instead it is more sensible to look at the orientation of edges in Williams' work. Some useful pre-existing techniques have already been discussed in section 1.3.1. Histograms of orientated gradients [11] seem to be a promising concept which may prove relatively simple to implement.

This section may also help with any work into brush-stroke analysis (see section 1.4.3).

1.4.3 Brush-stroke Analysis

With Williams' distinctive style and how obviously this style changes over time, the ultimate aim of this project is to be able to analyse the brush-strokes[2] in a painting.

From looking at the paintings it is very apparent that in his earlier work he made a lot more strokes than in his later work[3]. The strokes in his later work tend to have larger areas and span more of the canvas.

If it is possible to calculate a rough number and size of strokes made in a given painting, it should provide a reasonable set of data on which to classify this painting. As previously discussed in Section 1.3.2 there has already been a decent amount of research into determining brush-strokes in a painting.

It would be preferable to try and take one of the techniques discussed in that research and change it to suit the needs of the project. Rather than attempting to create a whole new method of brush-stroke recognition.

---

[2] A slight misnomer as Williams used a palette knife to paint with rather than a traditional brush

[3] Although this isn't quite true as the canvases he worked on in his later life tended to be larger

1.4.4 Ensemble Techniques

With some of the aforementioned analysis techniques it makes sense to combine two or more techniques together; a good example would be colour histograms and histograms of edge orientation.

This form of analysis is inspired by the concept of the same name in statistics and machine learning which tend to obtain better predictive performance. It may also be worth while trying to weight the results of techniques so that those which give the best performance affect the overall result of the ensemble more.

1.5 Classification Objectives

The overall objective of classification is to be able to label a painting by Williams as being painted in a given year based on analysis performed on all other paintings with known years. The ultimate aim of this is to be able to label a painting by Williams, where the date it was painted in is unknown, as being from a given year within a reasonable degree of certainty.

Evidently for paintings with an unknown year it is difficult to know how accurately the system has been, so, for the most part, these paintings have been ignored and those paintings with a known year have made up the training and validation set.

This can be used to evaluate the performance of the analysis technique and classification algorithm. Pearson's product-moment correlation coefficient (1.5) between actual year and classified year has been suggested to be a good performance measure for this project.

$$\rho_{X,Y} = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y} \tag{1.5}$$

1.5.1 Classification

One of the simplest methods of classification is k-Nearest Neighbour. This takes an average of the years from the k closest paintings in the data-set. This is assigned to be the year of the painting to classify.

Depending which form of average you take (mathematical mean, median or mode) will alter the result; although it should be noted that median is very unlikely to give a result on its own due to the sparseness of the data.

There are other techniques which could be applied to this problem. The benefits for implementing them are not likely to be outweighed by the time taken to implement these techniques. There is a workaround for this; there are several machine learning tool-kits which provide pre-implemented version of these techniques.

One of the most popular tool-kits available for general use is Weka [15], which is discussed in more detail in section 1.5.1.1.

Another technique suggested by Julie Greensmith is to use Learning Classifier Systems (LCS) [4], which has an implementation for Weka. This may prove to give very good results for the kind of analysis being performed on Williams' work.

1.5.1.1 Use of Weka

Weka is a machine learning toolkit which boasts an impressive number of learning techniques. As such is makes sense to use it to help classify the results of analysis techniques. This does mean adding additional logic to export data in a form Weka understands, but the results from Weka may

provide a new and detailed insight into our own analysis techniques which k-Nearest Neighbour cannot.

1.5.2 Exemplars

Another method of classification is to use expert knowledge, making use of the interdisciplinary work with members of the NLW, to label dates of paintings. This expert knowledge takes the form of a set of exemplar paintings for each year in Williams' career. Therefore, allowing these exemplars to be compared to the statistical centre of that year. This will allow the comparison of digital analysis to human experts.

Chapter 2

Development Process

This chapter defines the development processes which were used to aid the production this project and how they were changed to suit the needs. The decisions behind the choices of development process are also discussed and evaluated.

2.1 Introduction

This project was developed using a mixture of iterative development and rapid prototyping. As a research project it does not need a heavy-weight development process. Such a process would be too cumbersome to adapt to the changing nature of research. Agile methodologies would have also provided the adaptability required, but are designed for a team-based approach so would have needed a lot of modification to be applicable to a team of one.

Maintenance and testing are known to be the large majority of focus in the methodology chosen, experience in the field of software engineering typically focuses an individual on both these topics. However, as this was a research project it was unlikely to be used much after the end of the project. If it were to be used in further research the author or the supervisor would be available to provide assistance. It was decided that maintenance was not an issue for this project and could be disregarded to a reasonable extent.

Due to the research nature of this project, testing would prove to be difficult. Proving that a given technique would run was relatively straight-forward by leaning on the interpreter (or compiler in cases of any statically typed elements). However, proving a technique would do what it was expected to do would take so much time out of the implementation of other techniques, that it would be a pointless task. Especially when a lot of the techniques would just call external libraries to implement the parts that were likely to fail (e.g. applying filters through Open Computer Vision Library (OpenCV)).

With both these two issues being lesser than usual it was easy to discount a lot of the traditional and formal methods from the list of potential methodologies. Waterfall and Spiral models, for example, had too much of a focus on testing and maintenance, and on the management of risk.

Iterative development involves developing a system through repeated cycles, at each iteration design modifications are made and new functionality is added. Iterative development encourages modular design and implementation, suiting this project well as it will have a number of techniques to both analyse and classify data points.

Rapid Prototyping involves quickly producing a working prototype of an area of a system to get a working implementation of that area. This can then be evolved to improve the design and implementation as needed. This fits well as a method for implementing the various techniques in

the system, they can be rapidly prototyped to get a set of results, then improved to increase the performance and/or accuracy of the technique.

The requirements for the iterations were typically decided in the weekly project meetings, normally taking the form of completing a given technique before the next meeting. The technique was then implemented as a prototype and any adjustments would be made to the system in line with the iterative process, including improving existing techniques that might have been earlier prototyped in an evolutionary fashion.

Initially there were a set of topics which were aimed to be completed:

- Colour-space analysis

- k-Nearest Neighbour classification

- Histogram-based colour-space analysis

- Texture analysis

- Brush-stroke analysis

Each of the above items had several parts to them; colour-space analysis included different colour-modes: RGB and HSV. However as a research project new ideas were often suggested during meetings or as a part of research so this list of requirements grew during the project.

Python helped a lot with both methodologies, thanks to it's dynamic typing it is easy to deal with changing forms of data. It also has very readable syntax whilst still remaining compact in terms of lines of code, this makes locating areas which need changing very simple.

Version control was a strong requirement. Having used git for many personal projects the environment was familiar. The decentralised nature of git allowed for the remote working on any machine with an internet connection, so progress was not delayed with the switching of computers.

For hosting the GitHub service was used, GitHub is one of the most popular git hosting services available and offers free private repositories for students (GitHub is commonly used for Open Source software development so most repositories are public by default). It should be noted that this service is known to be very stable and reliable, but even during brief moments of downtime, the decentralised nature of git meant that work could continue regardless.

2.2 Modifications

The main modifications made to iterative development was to shift focus away from testing and into evolution and implementation. It was also changed to that each iteration was a rapid prototype, to allow merging of the two techniques. But the evolution step was focused on the whole system. A good example of this is when finding out about *cv2* and going back to change all the old techniques to use the latest version of *OpenCV*.

Rapid prototyping was modified very little. However, as there was no official client and there were only meetings with Hannah once a week, there was very little client focus. This means it was just used as a tool for rapidly implementing techniques in a way which could be improved and evolved later on. Although this isn't exactly in the spirit of rapid prototyping, it does fit fairly well with iterative development.

For a solo project, there wasn't any focus on team collaboration. Though iterative development isn't an official agile methodology, it does act quite similarly to one. Therefore it is quite tied to collaborating code as part of the iterative cycle. This project didn't take advantage of any of

the collaborative parts, nor any review of coding standards. The only review of code was of the author's own standards. Whilst these are very high, inexperience into writing Python code led to very messy code initially.

The evolutionary approach of the methodology allowed for this code to be cleaned up when required. Especially when modifying existing code later on in the project, when more knowledge into the language had been gained.

Chapter 3

Design

This chapter outlines an overview of the design of the project and why certain decisions were made regarding the design. To help with this, sections on the dataset and research methodology used have also been included to give a better view of the entire system.

3.1 The Image Dataset

The image dataset consists of 325 paintings, with associated metadata. Metadata includes title, year or year ranges (for those works where year is unknown but can be estimated by curators), genre, original painting size, painting materials and image size.

These photographs of paintings are challenging themselves: they are not colour calibrated; some suffer from reflections (towards the end of his life Kyffin painted using exceptionally thick and textural strokes, which gives specularities on the catalogue images); they are at varying resolutions; and come from a range of different cameras. Image size bears little relation to the original painting size, and some images are even optimised for the web. Table 3.1 below summarises the dataset.

Type	Number	Number (Known date)	Notes
Landscapes	247	64	
Portraits	52	35	
Seascapes	11	2	
Still lifes	4	1	
Other	8	0	Other or studies

Table 3.1: A summary of the Kyffin Williams painting dataset

3.2 Methodology

Within the database of 325 paintings, the actual year of painting is known for 102 artworks. To judge how well an analysis technique performs a leave-one-out cross validation methodology is applied to each technique, only working with these 102 artworks so that any amount of uncertainty is removed.

Leave-one-out cross validation involves taking a painting for which the year is known, then use the classifier to guess the year; this allows the result to be validated against the known year, thereby showing how well a given analysis technique works.

To simplify the classification stage a k-Nearest Neighbour classifier is used with the remaining 101 paintings for which the date is known. k-Nearest neighbour is a fast, non-parametric classifier. It has the advantage of being easy to implement and makes no assumptions about the underlying patterns in the data. Only the other paintings which are similarly located in the feature space are considered as part of this process. This functions on the assumption that paintings similarly located in feature space are also similarly located in time, this will be the case if an analysis technique is performing well. Figure 3.1 shows this process.
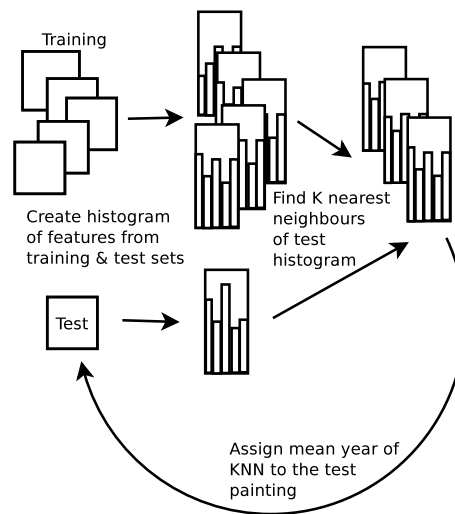
Figure 3.1: Overview of the Classification Methodology

3.3 Overall Architecture

The basic architecture for any system like this is to load the data in from a source, apply an analysis technique to each member of the dataset then pass this data into the classification system. The results of this system should return the classified and actual year for each member of the dataset. These results can then have cross validation performed on them. This architecture is summed up in Figure 3.2.

Building up from this it is apparent that to implement the analysis and classification steps that there is a need to implement the factory method design pattern [14, p. 107-117]. Reading from a data source should be a simple matter of reading from a file, and cross validation has already been decided upon: leave-one-out cross validation. Figure 3.3 shows the design after adding in the factory methods.

From this there is the need for two top-level interfaces `Analyser` and `Classifier`. The `Analyser` interface should have a single method which runs analysis on a painting and return some form of object which represents the analysed data.

The `Classifier` class should have a method which takes a single painting and a set of paintings, returning the classified year of the single painting based on the set of paintings.

At this point it is also required that there is a class to store the metadata of a painting. Figure 3.4 depicts the design after adding in these parts.
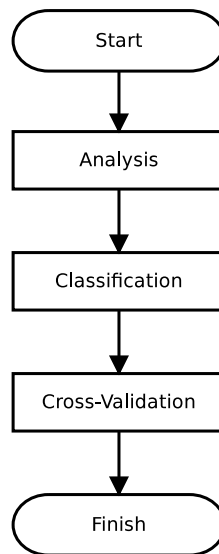
Figure 3.2: Basic Overall Architecture

From this the concrete techniques were implemented, at points it became necessary to add additional methods to the interfaces with some abstract implementation. This was particularly apparent during the change to OpenCV cv2 ("cv2") where all results from analysis techniques became histograms. The distance method in the `Analyser` interface was then implemented to call the OpenCV compare histogram method. This allowed the distance technique to always use the same measure of distance, allowing the analysis techniques to be compared more fairly.

With exemplars it was also useful to implement a centroid method as part of the `Analyser` interface. This was necessary to ensure the method for determining the centre point was the same. This highlights the usefulness of constantly evolving the design using the rapid prototyping methodology.

Until this point the inheritance had been unnecessary due to Python's duck typing. This inheritance had been purely to help the author understand the architecture. After this point the properties of object-orientation were being leveraged to help reduce the amount of repeated code.

3.3.1 GUI Elements

Initially a GUI Factory was implemented to display the results from the techniques graphically, using the same sort of method that had been used for the analysis and classification techniques. However, as the project continued the focus shifted away from this section due to the importance of other sections of work. Initially this was a part of the design and was included as a part of the progress report; shown in Figure 3.5. This also includes details of which methods each interface would include.

It should also be noted that of the time of writing the progress report some of the classes had different names, these were later refactored to better describe the functionality they perform. Figure 3.6 shows a better version of the class design with these changes in mind and with the GUI elements removed to keep the design simplistic.

These methods inevitably changed as the design evolved; mainly to add extra functionality whilst keeping the re-use of code down, as good object-orientated design should do. The evolutionary nature of the methodology allowed for this. Figure 3.7 shows the updates to the class design.
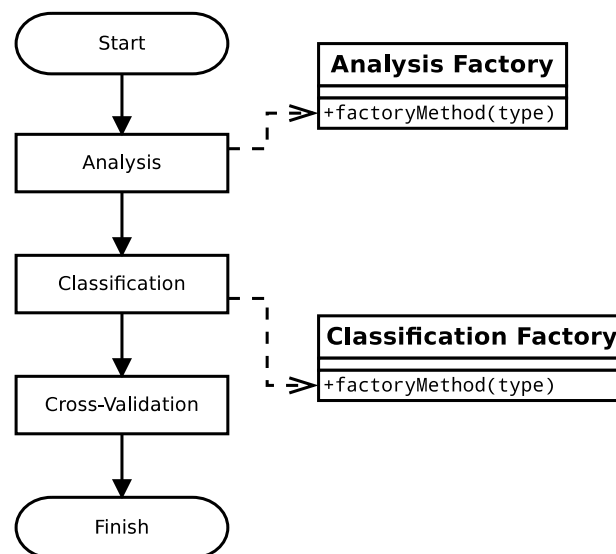
Figure 3.3: Overall Architecture with Factory Methods
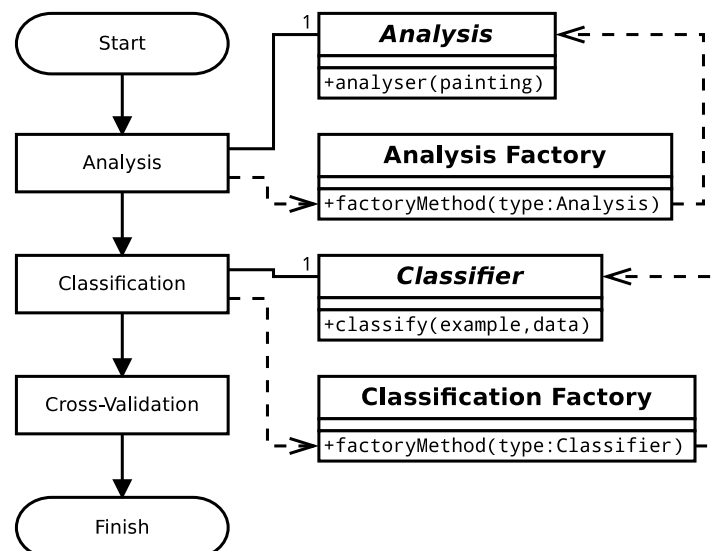
Figure 3.4: Overall architecture with Interfaces for Analysers and Classifiers
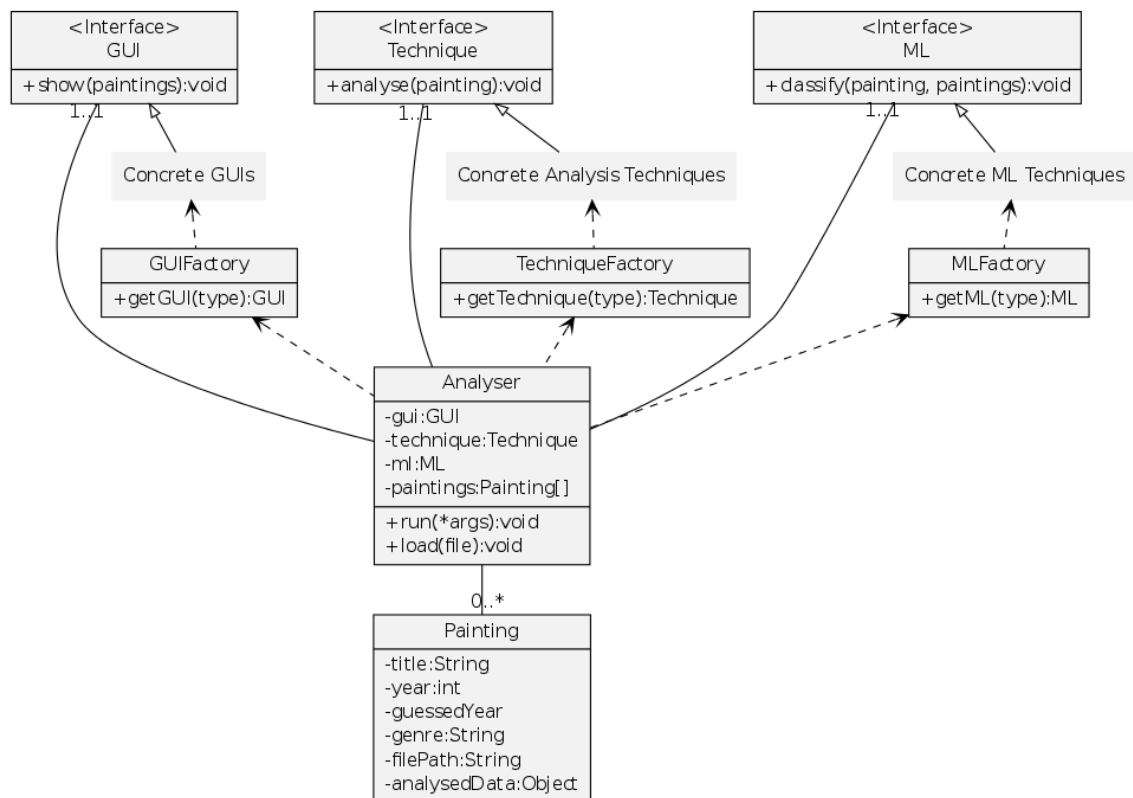
Figure 3.5: Initial Design from the Progress Report
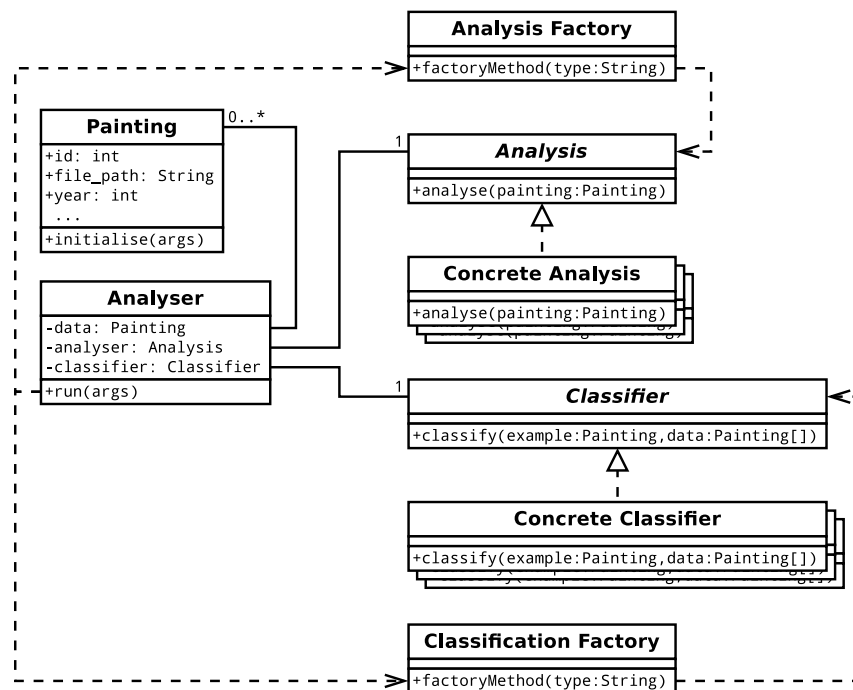
Figure 3.6: Overall class design
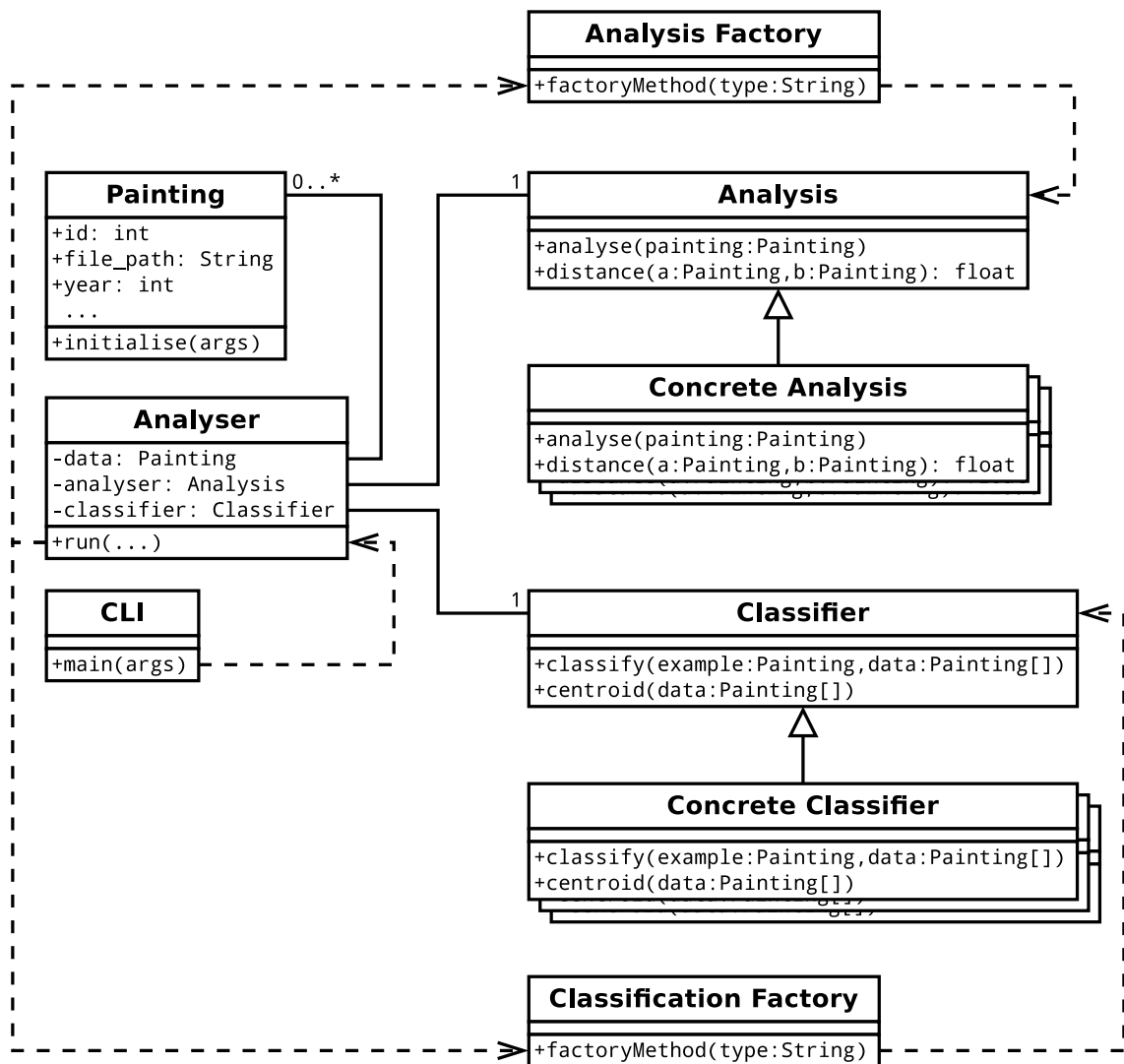
Figure 3.7: Final class design

Chapter 4

Implementation

This chapter outlines the process taken to implement this projects, as well as any issues encountered during this process and how these issues were overcome. Initially a lot of research was performed to work out which libraries would suit this project best. A basic framework was then created following the design in the previous section.

Analysis techniques were then implemented, starting at colour-space analysis and moving up to complex texture analysis.

Finally this chapter details the techniques used to classify paintings, from a simple nearest neighbour algorithm to integrating expert knowledge into the system.

4.1   Research into Image Processing Libraries

Due to the number of libraries out there and lack of experience in the area of computer vision and image processing, it was beneficial to research into some of the more popular libraries out there to get a feel for each library and image processing in general. This was done by creating a simple application to perform Gaussian blur on an image, from this it was easy to gauge the use of the library and its documentation and could be extrapolated to see how difficult the library in question would be to use for more complex applications.

Table 4.1 shows an overview of all the libraries which were investigated. Listings can be found in Appendix C.

Library	Platform					Language(s)	Example
	Windows	Mac	Linux	Android	iOS		
OpenCV	✓	✓	✓			C, C++, Python	Listing C.5
OpenCV - cv2	✓	✓	✓	✓	✓	C, C++, Python, Java	Listing C.6
FIJI	✓	✓	✓			Java	Listing C.7
IVT	✓	✓	✓			C++	Listing C.8

Table 4.1: Comparison of image processing/computer vision libraries.

OpenCV appeared to be the most polished of all the libraries researched into, boasting a wide range of features with comprehensive documentation, especially for "cv2", which, as has previously been discussed, is the more recent version of OpenCV.

FIJI Is Just ImageJ (FIJI) has a good range of high-level features, especially through their GUI elements, however for use as a library it was rather unwieldy and difficult to find the classes which performed simple functionality. Combined with equally bad documentation it was already

unlikely to be used but when, after fifteen minutes of struggling with the API images could only be blurred into a grey-scale output, it was completely discounted.

It should be noted that most of the FIJI features weren't used at all and the actual code seemed to just use ImageJ libraries. FIJI is often descibed as a distribution of ImageJ [**?**], the analogy "As Ubuntu is a distribution of Linux, FIJI is a distribution of ImageJ" is commonly used to describe this further.

To continue that analogy, when programming with operating system libraries; it is common only to use the core libraries; not those libraries made to use the kernel more usable. The same logic applies with FIJI and ImageJ; whilst FIJI provides a cleaner interface in which to use the libraries, at its core it is still calling ImageJ methods. On reflection it would have been better to only consider ImageJ rather than the more cluttered FIJI.

Intergrated Vision Toolkit (IVT) was somewhat similar to FIJI in that it had a good range of high-level features, but was less impressive as a library. Even following the example code it was difficult to compile against IVT, despite using the makefiles provided in their own examples.

It was, therefore, an simple choice with only one library being workable for this project. Not only was OpenCV the top choice from the above research, it is also one of the most prevalent libraries for computer vision problems.

Later in the project, the transition was made to "cv2", therefore it has been added to the above research for completeness. When comparing OpenCV and "cv2" it is apparent that "cv2" is much easier to work with and also has much better documentation.

4.2   Basic Structure

4.2.1   Loading Data

One of the more key parts to implement before all others in this project is the ability to load in data from the initial spreadsheet. The first step of this was to convert it to a Comma-Separated Values (CSV) file-type, which is easier to read programatically.

Initially the spreadsheet has a different format from the version depicted below. In this version the file names tended to be much longer and were in sub-folders depending on their collection. Extra logic was needed to locate the file, though this was a simple matter of concatenating the collection to the file name as a directory.

Later in the project a new version was provided by Lloyd, the format of which is shown in Table 4.2, this was then converted to a CSV file format shown in Figure 4.1. This makes the spreadsheet data easier to parse digitally as there are many libraries available to parse CSV files whereas attempting to parse a specialised spreadsheet format. This was also a good excuse to clean up some of the initial code to use better Python programming practises; replacing messy loops with list comprehension where possible, using Key Word Arguments (`**kwargs`) and dictionaries instead of having utility methods, etc.

Filename	ID	Title	Catalogue entry BBC YP	Genre	Height	Width	Area	Materials	Collection	image width	image height	image height/image width
001.jpg	1	A Chapel in the Tyrol	1950-1960	Landscape	40.7	29.8	1212.86	oil on hardboard	NLW	687	944	1.3741

Table 4.2: Layout of the Painting Data Spreadsheet

```
001.jpg,1,A Chapel in the Tyrol,1950-1960,Landscape,40.7,29.8,1212.86,oil on hardboard, NLW,687,944,1.3741
```

Figure 4.1: Painting Data in CSV Format

This CSV file was then parsed using the Python 2.7 in-built `csv` module with relative ease. The example code (see listing C.1) helped to ensure correct resource management as well as showing how the library parsed files.

4.2.2 Top-Level Classes

This section focuses on implementing code based on the design shown in Figure 3.7. Being used to Java, the initial instinct was to set up interfaces for the majority of top-level classes (namely the `Analyser` and `Classifier` classes). However, Python does not have the concept of interfaces, so normal classes were used to represent these instead. These classes acted as abstract stub classes and just held place-holder method definitions which needed to be overridden in the concrete sub-classes. This was a slightly pointless exercise due to the duck typing, but was useful to depict the architecture in code. It also became more useful when these classes began to define a lot of the common methods.

A `Painting` class was also implemented and had a builder method to take an array (from the CSV file) and create a new instance of it from this. Later this was changed to make use Python `**kwargs` instead to make it easier to both read and process, especially in the exemplar work where mock objects were used.

4.2.3 Command Line Interface

As a research program with lots of different analysis and classification techniques the next step in setting up the basic architecture was to create a set of command-line arguments which would switch the functionality of the program. These arguments are depicted in Table 4.3.

Name	Short Flag	Long Flag	Description
Analyser	`-a`	`--analyser`	Switch the analysis technique
Machine Learning	`-m`	`--ml`	Switch the machine learning technique
Data	`-d`	`--csv`	The data file to use
GUI	`-g`	`--gui`	Switch the GUI visualisation
Binning	`-5`	`--bin-years`	Round paintings to the nearest 5th year
Export	`-e`	`--export`	Export analysed data
Classify	`-c`	`--classify`	Classify the specified image

Table 4.3: Command Line Arguments

The `argparse` library handles this nicely, the example code shown in listing C.2 shows the usage of this library.

These flags then hook into the factory methods to actually create the instances and return them to the central point which then called them where required. A façade [14, p.185-194] was used to hold and call these instances, as well as handling the calls to the factories.

4.3 Colour Space Analysis

Colour space analysis involves performing statistical analysis on different colour models (RGB, HSV, etc.). This gives a very simplistic view of the entire image, through creating a small histogram representing mean colour over the entire image.

OpenCV offers a method to perform the average across the image, however with a further look into the documentation there is also a method which performs both mean and standard deviation on an image.

The analysed data was just the tuple returned by this method. The distance measure was defined to be the sum of all elements in the tuple (in the case of an RGB colour model the mean red, green and blue and the standard deviation of red, green and blue).

4.3.1 Colour Models

There are many colour models to consider with digital image processing. RGB is one of the better know colour spaces as it is often how images are captured. It does have a feature in that all three values can change when the brightness changes.

As one of the main hypotheses of this project is that Williams' work darkened over time, it may follow that RGB may not be the best colour model to use.

To account for this it was decided to also use a HSV colour model to compare and contrast RGB. RGB is a colour-model in which a pixel of colour is represented by three 8-bit values, one for the intensity of each colour. HSV, in contrast, is a colour model which uses a range of $360°$ for the hue and is the only representation of colour in this model. The value defines the lightness of the colour and the saturation defines the degree to which the colour is present.

OpenCV handles colour spaces slightly oddly. Initially it uses a method to load the image, which uses an integer argument as a flag to define whether the image should be loaded in colour or grey-scale.

From this image you then can use a function to convert the colour model of an image, which uses an integer argument as a flag to define a number of different colour spaces. Once converted, all methods act exactly the same as they would on a RGB image.

4.3.2 Colour Histograms

Colour Histograms are a representation of the distribution of colour across an image; this makes them very powerful for analysing the colour space of an image. For digital images, histograms are the number of pixels which have a colour in a fixed range of colours. These ranges are typically referred to as the bins of a histogram. These bins are arranged so they span the range of colours in the entire colour space.

OpenCV provides a number of methods for both calculating and operating upon colour histograms. Here the example code (Figure C.4) from the OpenCV documentation was a useful reference as some of the details of creating histograms in OpenCV are not noted explicitly in the python documentation.

Initially this was performed as a 3D histogram to span three channels of the chosen colour model representation; for an RGB representation each dimension represents one of the three primary colours - red, green and blue. A low number of bins was also used to keep the memory limits down as this was a very large object if the whole range of colour in the model are represented. Later on, with the move to "cv2", this became a less expensive option and it was possible to create a 2D histogram for each channel in the colour model.

The distance measure between colour histograms is also handled by OpenCV using a compare histogram method. Under the covers this uses a χ^2 (Equation 4.1) method to compare to histograms of equal dimensions. χ^2 is not the only distance measure that can be applied to histograms; distance measures like Euclidean and Manhattan distance which have been previously

mentioned can also be applied to histograms, but do not include normalisation that χ^2 does so they are less useful to the histograms used in this project.

$$\chi^2(H_1, H_2) = \sum_I \frac{(H_1(I) - H_2(I))^2}{H_1(I)} \tag{4.1}$$

There are other methods of comparing histograms to one another, which OpenCV provides by default, but which are not very well suited to the application. Methods like histogram intersection where, for each bin, the smallest value is chosen between the two histograms; or calculating the correlation between two histograms, are obviously less useful than an actual measure of distance.

4.4 Texture Analysis

4.4.1 Edge Count

The simplest technique to analyse texture is to apply an edge detection algorithm over the image and produce a count of all the marked edges regardless of orientation.

The Canny edge detector [10] provided a good output for measuring this, operators like Sobel provided less useful results and thus provided a worse analysis technique.

From the generated image a histogram was produced with only 2 bins; one bin for the presence of an edge, the other for the absence of an edge. The second bin is just to help with the normalisation of distances as it is just the number of pixels in an image minus the number of pixels in bin one.

Open CV provides methods for applying the Canny edge detector to an image so this technique was simple to implement.

4.4.2 Histograms of Orientation Gradients

This technique is based on the HOG paper [11] and mirrors some of the implementation in a simplified fashion.

The main crux of the method was to generate the exact orientation of a given point of an image, apply this across the image and then bin the results into a histogram. The original paper then focused on using this for human recognition and aimed to keep the time complexity down through normalisation, applying it to this project only needed the generation of the histogram of exact orientations.

Calculating the edge orientation involves passing a filter over an image which is used to find the orientation of that gradient.

There are many forms of filter which can be used to do this. The simplest of which discrete derivative masks, steerable filters are a more adjustable implementation of this. There are also more complex filters, like Gabor filters, which provide better flexibility and matching.

Using discrete derivative masks it is fairly simple to work out to orientation of a gradient mathematically. First pass a mask of the form $\begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$, followed by one in the form $\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$. These give $\frac{\delta f}{\delta x}$ and $\frac{\delta f}{\delta y}$ respectively.

A gradient direction function, shown in Equation 4.2, can then be used on both these values to work out the actual direction of the gradient.

$$\theta = \text{atan2}\left(\frac{\delta f}{\delta x}, \frac{\delta f}{\delta y}\right) \tag{4.2}$$

4.4.2.1 Caching of Histogram of Orientated Gradient Results

Working out the exact orientation of a gradient is a costly operation and it became useful to *cache* (persist) the results to allow the technique to complete in a decent amount of time. This change was made after the change to "cv2" and was done using numpy methods for simplicity.

For speed of execution and in order to experiment with bin sizes it was easier to cache the results from the gradient direction function than to just cache the final histogram result. This allows for the ease of comparison between all forms of texture analysis.

4.4.3 Steerable Filters

Another way of analysing the distribution of edge orientations to change the orientation of the filter and bin the direction into certain cardinal directions; typically 0, $\frac{\pi}{4}$, $\frac{\pi}{2}$ and $\frac{3\pi}{4}$ (from π to 2π becomes a repeat of any of those directions, only in reverse and are, therefore, covered already).

To do this steerable filters $S(\theta)$ can be used (shown in Figure 4.2). The angle of the filter (θ) is manipulated to match the four cardinal directions, which will then give the degree to which a gradient matches that angle. A histogram is created with a bin for each value of θ, these bins are added to if the value of a given filtered pixel is above a threshold, helping to prevent the overlapping of orientations.

$$S\left(0\right) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad S\left(\frac{\pi}{4}\right) = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \quad S\left(\frac{\pi}{2}\right) = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad S\left(\frac{3\pi}{4}\right) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Figure 4.2: Steerable Filters

These have the advantage of being far less complex to compute than edge orientation; the maths and histogram creation on a floating point array are costly operations, to the point where canning the results for edge orientation is desirable.

There is the obvious trade off that steerable filters can only work in four directions which does affect the performance quite severely.

4.4.4 Gabor Filters

To increase the orientations matched from steerable filters without the need for increasingly large kernels, Gabor filters [12] were implemented. Gabor filters allow for an almost-infinite range of orientations without affecting the size of the filter. To do this they use a sinusoidal wave; which matches gradients, wrapped in a Gaussian envelope. Figure 4.3 shows an example Gabor filter and Equation 4.3 shows the mathematics used to generate a Gabor Filter.

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right)\cos\left(2\pi\frac{x'}{\lambda} + \psi\right) \tag{4.3}$$

where:

$$x' = x\cos\theta + y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$
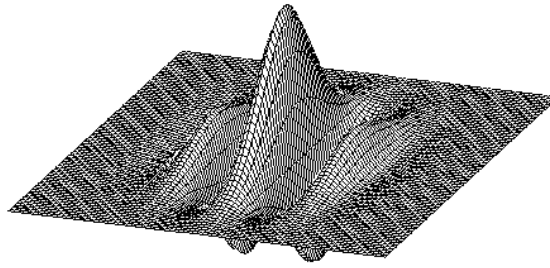
Figure 4.3: The real part of the impulse response of a Gabor filter [23]

As with steerable filters the convolution of an image with a set of Gabor filters were accumulated in a histogram. Initially Gabor filters were implemented with the same 4 orientations as steerable filters (0, $\frac{\pi}{4}$, $\frac{\pi}{2}$ and $\frac{3\pi}{4}$), however this has the problem that the variation in orientations is not being utilised.

To solve this issue the way Gabor filters were initialised was changed by adding in a parameter which defined the number of orientations that instance of the Gabor filter would have, the logic of which is defined in Figure 4.4, note that the range is $1 \ldots b$, as $\theta = 0$ and $\theta = \pi$ will produce the same result (both matching vertical lines best). This helps to prevent certain mathematical issues, particularly those involving passing θ in as 0.

function GABOR.INIT(b) $\triangleright$ b is the number of orientations
    for $i \in 1 \ldots b$ **do**
        $\theta_i \leftarrow \frac{i\pi}{b}$ $\triangleright$ θ is the list of orientations to produce filters for
    end for
end function

Figure 4.4: Psuedocode for Gabor Filter initialisation including number of orientations to produce filters for

The actual Gabor filters are then produced for all values in θ at runtime and applied to the images in turn.

An example section of MATLAB code which implements Gabor filters shown in Listings C.3 [26] was used as a reference to write the code for the Gabor filters. This code was improved to make use of numpy array manipulation functionality as well as to be more readable.

4.5 Brush-Stroke Analysis

Due to time constraints it was not viable to look at brush-stroke analysis, especially with the shift of focus to exemplars after texture analysis.

It would have been nice to explore this area due to Williams' style, but a lack of high-resolution images would have also made this a difficult area to get right.

There were several good ideas thought of for this area. One was to train a machine learning technique to recognise brush-strokes through the manual marking of strokes. This would have been a very interesting method to try, especially being able to apply existing techniques to aid and improve the classification of brush-strokes.

Other techniques involved passing filters over the image looking for the edges of brush-strokes [7], then applying other operations to complete the stroke area. This is a much simpler technique which will likely give bad results due to the low resolution and non-uniform scaling of the images as it seems to be very dependant on the filter.

One final technique suggested by Li et al. [20] involves a combination of edge analysis and clustering in colour space with a number of heuristics involving branching, stroke-width modelling and gap filling to refine the original brush-stroke estimates.

The difficulty that would be associated with this section of work is that existing techniques may not be applicable to Williams' work due to his use of the palette knife. Whilst there are very obviously clear strokes in his work, comparing these strokes to the likes of Van Gogh are unlikely to pay off due to the *blockier* strokes of the palette knife this is another reason this analysis technique was side-lined in favour of exemplars.

4.6   Ensemble Methods

Ensemble methods are commonly used in statistical and machine learning to improve the performance, the main concept of an ensemble method is to combine two or more models to produce better results than would be gained from a single model.

The simplest form of ensemble methods just involve running all models in the ensemble, combining them into one large histogram and then performs normal distance measure on them. However, due to some problems with the "cv2" histogram calculation method, it was a lot easier to flatten (to collapse a multi-dimensional array into a one-dimensional array) the results from each analysis technique making up the ensemble method.

This does affect the performance a little, but it is better than the other solution of adding resulting distance from each model together as the distances are not normalised. This was my initial solution to the problem and hugely affected the results as the distance returned from statistical colour-space methods is a lot less than those returned from colour histograms and so on.

Of course it would also be possible to normalise each distance measure of each technique, but this would be difficult as the mean and variance of the distance for an analysis technique is not tracked. It may also cause problems when new examples are added (during validation, for example) which affect both these values.

This method is commonly known as Bootstrap Aggregating (Bagging); where each model has a "vote" with equal weight.

One part I considered adding to this, but didn't have the time to complete, would be to weight the vote of each technique (likely via a machine learning technique) in the ensemble so that the better performing techniques affect the results more.

Another method commonly used in machine learning is Boosting, where the model is built incrementally, with the focus being on examples which were mis-classified by the previous models. This method does improve accuracy, but is known to over-fit the dataset.

4.7   Classification and Validation

With the results of analysis techniques there's also a need to be able to classify paintings based on these results and also to validate the results of both so that different methods of analysis and classification can be compared for performance.

4.7.1 k-**Nearest Neighbour**

The simplest method of classification is to work out which painting in the dataset is closest in feature space to the painting which needs to be classified and classify the year of this painting with the year of the nearest painting.

This is a Nearest Neighbour algorithm with $k = 1$, Figure 4.5 shows the psuedocode for this algorithm

> **function** 1-NN(p, $data$)
> $n = \arg\min_{d \in data} \text{DISTANCE}(p, d)$
>   **return** n_{year}
> **end function**

Figure 4.5: Psuedocode for 1-Nearest Neighbour

From 1-Nearest Neighbour it is a simple matter to change this to a k-Nearest Neighbour algorithm as Figure 4.6 shows.

> **function** K-NN(p, $data$, k)
>   **for all** k **do**
> $n = \arg\min_{d \in data} \text{DISTANCE}(p, d)$
> $a \leftarrow a + n_{year}$
> $data \leftarrow data \setminus n$
>   **end for**
>   **return** AVERAGE(a)
> **end function**

Figure 4.6: Psuedocode for k-Nearest Neighbour

With this, the method of averaging the year can be any statistical method of taking the average (mean, median or mode). Mean is the typical case for k-Nearest Neighbour as it is simple to implement. Median is less common as it is slightly more complex to implement. Mode is more difficult to implement programmatically, especially given the sparseness of the data.

However, using mean raises an interesting problem. Mean returns a point, in this case a year, which may not be a year of any painting in the dataset. Median and mode will nearly always return a year in the dataset. With mean there is an implicit assumption that Williams painted every year without a gap whilst our dataset does have these gaps.

4.7.2   **Leave-One-Out Cross Validation**

To validate the classified years from k-Nearest Neighbour, this project uses Leave-one-out Cross Validation; a technique which involves removing each point from the known dataset, applying the classification technique to this point against the remaining dataset. This produces a list of classified years against the actual years.

To measure the goodness of fit the Pearson's product-moment correlation coefficient was calculated on these orderings; this provides us with a performance measure of each classifier. It is also possible to test Pearson's R for statistical significance.

Correlation is the measure of statistical relationship between two sets of data, a correlation of 1 is a perfect match where one set of data is exactly equal to the other. A correlation of 0 means there is no identifiable match between the datasets and a correlation of -1 is where the datasets

are opposite to one another. Because the aim is to have the classified exactly equal to the actual year, a correlation closer to 1 means that the classified years are closer as a whole to the actual years.

Another measure applied alongside correlation is statistical significance. This is the degree to which a statistical result could have occurred by chance given a null hypothesis. This can be applied to a correlation to calculate the extent to which the correlation could have been arrived at by chance.

Overall it is best to try and maximise the correlation between two data-sets whilst keeping the significance low. Pearson's product-moment correlation is able to generate the correlation between two data-sets and also defines a method of generating the statistical significance on top of this. numpy's implementation of this provides both the correlation and p-value (significance) as part of the return value.

Both of these are easy to implement - leave-one-out cross validation just involves iterating over the entire list of known paintings, popping the current painting from the list and running the classifier on it, then pushing it back into the list at its original position.

A plotting library, matplotlib, was used to generate scatter plots of actual year against classified year to represent this data visually. An example of this is shown in Figure 4.7
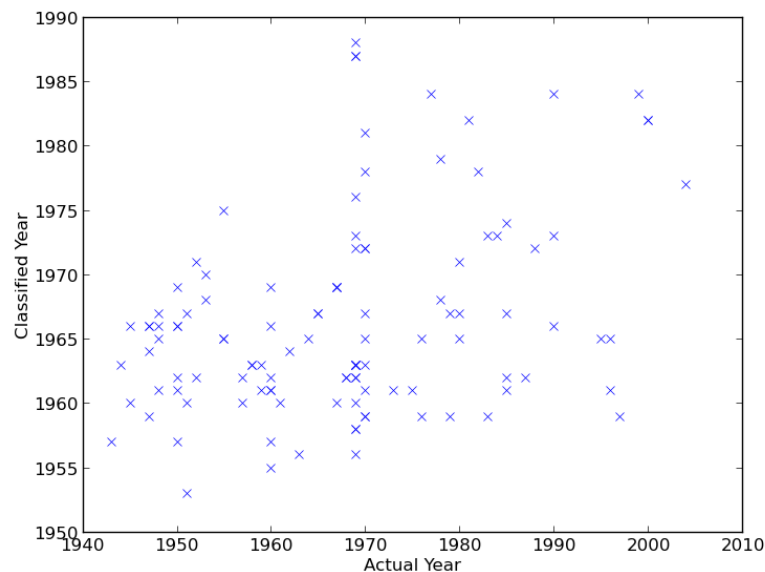
Figure 4.7: Scatter Plot of Actual Year against Classified Year for Gabor Filters and k-Nearest Neighbour ($k = 7$)

4.7.3 Weka 3

Weka is a collection of machine learning algorithms [15], used predominately for data mining. Due to the number of machine learning techniques it provides, it was interesting to input data from the analysis techniques into Weka.

The first classification technique applied through Weka was linear regression [22, p.717-727]: the act of fitting a linear function to the training data-set then classifying new examples using this function. The function can either be univariate (a straight line function) or multivariate, where

each example is an n-element vector. For our data multivariate is more applicable but may require a reduction in the complexity of the function to avoid the overfiting of data (where the classifier is to specific to the training data-set and cannot generalise to new examples well).

Another set of classifiers applied to the data-set through Weka were artificial neural networks [22, p.727-737]. Neural networks are biologically inspired classifiers which use layers of nodes or units, which are connected with directed links. The simplest form of neural network, a single-layer feed-forward neural network (preceptron), where all input nodes are connected directly to outputs was used. As well as a more complex neural network with a set number of hidden layers - layers of nodes which do not directly correspond to the inputs or outputs, but which perform some intermediary function.

Unfortunately, it seems that the data provided to Weka is not well suited to most of the methods in the tool-kit. The data was, perhaps unsurprisingly, ill suited to regression techniques; each parameter has such a variable set of values that it is difficult to perform regression on these parameters.

Similarly, Neural Networks also perform badly, again it seems that so many of the parameters can take on such widely different values, even in the same year group, that it is difficult to build a model to suit the data.

In the end Weka proved to be more time consuming than it was worth and, though it did provide a few interesting results, the variation in data proved just too much for it.

k-Nearest Neighbour does not suffer from this flaw as it does not need to build a model based on the data. It purely classifies on the distance between points in feature space. This does make it simple (and usually too much so) for machine learning techniques, but in this case, it turns out to be one of the best choices.

4.7.3.1   Attribute-Relation File Format (ARFF)

Part of inputting data into Weka is to produce Attribute-Relation File Format (ARFF) from the analysis techniques. ARFF is the main format supported by Weka, so it is one of the easiest ways of importing the data into Weka.

A few libraries exist for ARFF conversion in Python, however they all have their own quirks. liac-arff was one of the easiest to use, but doesn't support the date attribute. This isn't a problem for our current project as we only use year which can easily be represented as an integer instead. I did intend to contribute a working patch for dates for the library, but it would have taken too much time out of the project to get it working correctly.

To be able to export in ARFF file some translation from OpenCV structures to raw data is needed. To begin with (whilst still not using "cv2") this was easiest done by exporting complex OpenCV to a file using the functions provided by OpenCV, then read back from this file in for each painting.

To do this the Python operating system library was used to create a named temporary file, then to pass the name of this file to OpenCV (which only takes a file name and not a file descriptor or any other form of pointer).

The output from OpenCV export methods is a YAML Ain't Markup Language (YAML) - a human-readable data serialisation format - which is well supported by Python. This data can then be read back in a non-OpenCV data structure and then re-exported using the ARFF method described above.

The other option would be to find a parser which converts the YAML format to an ARFF one. However no such parser currently exists. Whilst this would be a nice tool to write and contribute back to Python, the time it would take to do so would not outweigh the benefits.

4.7.4 Exemplars

Another method of classification is to incorporate expert knowledge within the framework. For each year represented in the collection Dr Paul Joyner, of the National Library of Wales was asked to choose the one painting which best represented Williams' work for that year. Dr. Joyner is a member of the Trustees of the Kyffin Williams Estate and he has written widely on Welsh Art and Kyffin Williams.

These chosen paintings are considered to be connoisseurially or artistically selected exemplars (*artistic exemplars*, for short), which can be used as a representation of that particular year.

4.7.4.1 Nearest Exemplar Classification

The most obvious use for artistic exemplars is to classify a given example using the nearest artistic exemplar, rather than other members of the dataset, as depicted in Figure 4.8.

function NEARESTEXEMPLAR(p)
    $n \leftarrow \arg\min_{e \in exemplars}$ DISTANCE(p,e)
    return n_{year}
end function

Figure 4.8: Psuedocode for Nearest Exemplar Classification

To implement Nearest Exemplar Classification was a fairly easy task: a secondary spreadsheet was provided which contained all the necessary information of exemplar by year (see Table D.1 for the full document).

The spreadsheet was arranged in the format described in Table 4.4, from there it was a simple matter of saving the spreadsheet as a CSV file and taking some of the existing code for parsing CSV files. This caused a slight problem in that the parsed data didn't have enough information to create a full `Painting` object, yet all the analysis techniques worked from these objects.

Filename	ID	Title	Catalogue Entry	Year
154.jpg	154	Landscape at Llanaelhaearn	1947	1947
etc.				

Table 4.4: Layout of the Exemplar Spreadsheet

This was solved easily thanks to Python's dynamic typing. A simple class which implemented all the necessary elements of `Painting` could be passed to the analysis techniques without any complaints. With a statically typed language this would have been harder to complete, but there would have been ways around using sub-classes and so on.

With the exemplars loaded and analysed, the program could continue as normal, until the classification step.

The idea of Nearest Exemplar Classification is to classify the unknown example using the nearest exemplar to that example in the feature space. This acts as a k-Nearest Neighbour with $k = 1$ and the space of neighbours only including the exemplars, rather than every other example. The psuedocode for this is shown in Figure 4.9.

Initially this was implemented so that the examples that were exemplars were also classified, but this is a pointless exercise which only skews the results. Additional logic to remove any exemplar which matched the current example was added as paintings were often classified as begin nearest to themselves.

function NEARESTEXEMPLARCLASSIFICATION(p)
     $n \leftarrow \arg\min_{e \in exemplars \setminus p} \text{DISTANCE}(p, e)$
     return n_{year}
end function

Figure 4.9: Psuedocode for Nearest Exemplar Classification with Logic to Remove p from *exemplars*

This proved to give slightly worse correlation per technique than k-Nearest Neighbour. This result is to be expected; for a start an artistically classified exemplar is unlikely to be the same as a statistically classified exemplar (see Section 4.7.5). Also, with fewer examples to classify against, any variance in the dataset (of which there is a lot) will likely be magnified.

Lastly, a painting may be picked as an exemplar by an expert for different reasons than any analysis technique that currently exists can give; emotional connections and knowledge of the artists history can be very subjective and may not relate to anything put down in paint, or captured in the feature spaces we investigate.

4.7.5 Statistically Classified Exemplars

Another approach to exemplars is to work out a theoretical exemplar for a given period; the centroid of paintings within the given feature space for a single year, for example. Figure 4.10 shows how a centroid is generated for three points in a 2D feature space. However, for a high-dimensional feature space this method needs to be simplified.
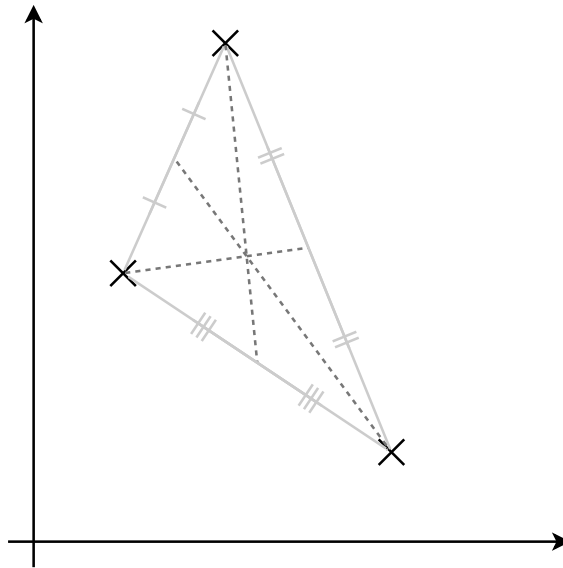
Figure 4.10: Centroid for three points in a 2 dimensional feature space

Equation 4.4 shows how the centroid (C) can be generated in a high-dimensional feature space (of D dimensions), this works by taking the mean (Equation 1.3) of each feature in the set of paintings ($x_1, x_2, \ldots, x_k$) for a single year, this will give the point in feature space that is most central. This is the same technique used to generate centroids in most clustering algorithms (k-Means Clustering, for example).

$$\forall_{d \in D} \; C_d = \frac{1}{k} \sum_{i=1}^{k} x_{i_d} \tag{4.4}$$

This may become a long operation depending on how many dimensions the feature space has. A technique like Principal Component Analysis (PCA) [18] may be useful to help cut down the number of dimensions needed that this algorithm uses.

Initially, as I wasn't producing histogram results from analysis techniques, this was a little complex to do; each analysis technique had to have its own method of generating the centroid and returning the data for this centroid. The change to histogram results made everything a lot easier and, combined with the mean method from numpy, which can handle the mean along different axes, rather than just over a flattened array made this process a lot easier to compute, especially as histograms from "cv2" are returned as numpy arrays and the data passed into the centroid method is now an array of these histograms. Calculating the mean across the first axis has the effect of calculating the centroid. Equation 4.5 shows this mathematically on 2D Matrices where $\bar{x}_n$ is the mathematical mean of x along axis n.

$$x = \left[\begin{bmatrix} 1.0 & 1.0 & 0.0 \\ 0.0 & 1.0 & 1.0 \\ 0.0 & 0.4 & 0.0 \end{bmatrix} \begin{bmatrix} 0.0 & 1.0 & 0.0 \\ 1.0 & 1.0 & 1.0 \\ 0.6 & 0.0 & 1.0 \end{bmatrix} \right]$$

$$\bar{x}_1 = \begin{bmatrix} 0.5 & 1.0 & 0.0 \\ 0.5 & 1.0 & 1.0 \\ 0.3 & 0.2 & 0.5 \end{bmatrix} \tag{4.5}$$

Whilst the mean histogram could be a useful centroid, it doesn't fit with the idea of an artistic exemplar, it is quite possible that the art historians would be able to imagine a theoretical painting which suits that year but they were limited to real paintings so it should follow that the statistical methods are also be constrained in the same fashion.

A note on terminology; an *artistic exemplar* is an exemplar which the art historians view to be the most representative painting. A *centroid* is the mean of a set of analysis over a particular year and is a point in feature space, it is not an exemplar as it does not exists as a real painting. A *statistical exemplar* is the nearest real painting to a centroid.

Statistical exemplars are generating by applying a nearest neighbour algorithm to all paintings where the only neighbour is the centroid. This could be seen as a form of generalisation and may help prevent overfitting of a classification technique. It may also show which years in which the artistic exemplar commonly matches the statistical exemplars and where they do not. This gives a vision of the difference between artistic and statistic exemplars and, perhaps, where the artistic exemplars are the outliers of a particular year.

By re-using methods from the nearest-exemplar classifier (which uses the artistic exemplars only), the option to use inheritance to cut down on code repetition was made available by extending this existing classifier and overriding the method which loaded the statistical exemplars from the CSV file to generate the centroids and statistical exemplars instead. Here the use of super calls in Python was confusing to a certain extent; listing 4.1 shows one correct solution to this[1].

Listing 4.1: Super method calls in Python

```
# Note Base has to extend 'object' for super calls to work
```

---

[1] There are, of course, other correct methods, but the one listed is one of the better solutions

```
class Base(object):
    def call(self, params):
        # ...

class Sub(Base):
    def call(self, params):
        # Before Base.call
        super(self, Sub).call(params)
        # After Base.call
```

4.8   Filtering

One area that was experimented with was the ability to filter paintings by certain pieces of metadata for that painting. Typically with would allow the filtering of just a single type of painting (e.g. filtering out any painting which isn't a landscape to remove potential outliers).

Unfortunately, with most techniques this produces worst Correlation than not filtering by type, this may be due to the limited size of the data-set. Removing the non-landscape paintings leaves such a gap in the years that k-Nearest Neighbour cannot classify the painting correctly. Before filtering the data-set only contained 325 paintings, after filtering down to just landscapes this was further reduced to 247 paintings.

A potential way of helping with this (and effectively filtering the data in a slightly different fashion) is to round the year (y) of a painting to the nearest 5 years (y_5). The maths to do this is shown in Equation 4.6.

$$y_5 = 5 \left\lfloor \frac{y}{5} \right\rfloor \tag{4.6}$$

This does improve the correlation by a little as it helps to counteract the sparseness of the data in the dataset. Whilst the improvements in correlation are nice, the information lost by rounding is not really gained back by this increase in correlation.

Chapter 5

Testing

This chapter details the approach to testing and how cross-validation was performed to evaluate different analysis and classification techniques.

5.1   Overall Approach to Testing

As this is a research-based project, it is difficult to perform any unit, functional or requirements based-testing. Most testing is based on validating the results of analysis and classification techniques.

There is, however, a sense in which the validation of analysis techniques is a form of testing. Higher correlations would suggest that a given analysis technique is both implemented properly and that it represents the data well.

There is also a sense in which testing that a technique does what it says it will do, is both difficult and somewhat unnecessary. One would expect to see the correlation increase as more complex analysis techniques are implemented, so if a recently implemented technique gives a lower correlation that a previous technique, then this might be considered as a failing test.

If the correlation is improved by the analysis then, even if the technique isn't working completely as advertised, it is still doing something right. It was common for the analysis to be "tinkered" with to try and improve the performance of the analysis, so commonly any errors were picked up at this point.

All analysis techniques were performed across two machines and typically acted similarly. The only discrepancies noticed were on techniques which heavily relied on floating point numbers. This took the form of a few of the correlations being slightly different between the machines. This is believed to be because one system has a 32-bit operating system and the other is a 64-bit. This would mean that floating point numbers would be slightly more accurate on one machine over the other.

5.2   Validation

Cross-validation is a statistical concept used to assess how the results of statistical analysis generalise to an independent dataset. Cross-validation is typically applied to machine learning classifiers to test if a technique is overfitting the data. That is how specific the technique is to the original data set and how it can cope with new, unseen, examples. For example, a rote learner cannot classify any unseen examples, as the classification is performed based solely on the training set.

As the overall goal is to be able to classify any Williams painting, it is important that the program is able to handle new examples, not just those for which the year is known.

There are many forms for calculating cross-validation, usually involving taking sub-samples of the original dataset and removing them from the training set to create a validation set.

k-fold cross-validation is a common method for performing this. It involves splitting the dataset into k sections or *folds*. Each fold is removed from the dataset, the classifier is trained on the remaining data, then the fold is classified using this trained data.

The other common form of cross-validation is to take random sub-samples from the dataset and are removed from the training set and then classified using it. This process is performed repeatedly until a good number of samples have been taken. This has advantages and disadvantages over k-fold cross-validation, it has the problem that not all data points are considered during validation, but doesn't depend on the size of the data set.

5.2.1 Leave-One-Out Cross-Validation

Leave-one-out cross-validation is a form of k-fold cross-validation, in which the number of folds is equal to the size of the data set. This leads to the behaviour of leaving a single data point each time and classifying it against the rest of the data set. Because data sets in machine learning are typically very large this usually isn't an option due to time complexity.

However, because of the use of k-Nearest Neighbour-based classifiers (which need no re-training when data points are added or removed) and a small number of known data points, it is one of the best method of cross-validation for this project.

Leave-one-out cross-validation provides us with a classified year for each data point. Because only data points with known years are used, the correlation between actual and classified years can be calculated to get a measure of performance.

To produce this value Pearson's product-moment correlation was used (Equation 1.5). The best techniques would produce a correlation value closer to 1 than other techniques. Another part of Pearson's is that it is easy to generate statistical significance (the p-value). This describes the extent to which the observations could appear by chance in a given single null hypothesis.

The results of the correlation between actual and classified year is defined to be r (formally: $r = \rho_{classified,actual}$). The p-value is defined to be $P(r)$. The ultimate goal being to maximise r whilst minimising $P(r)$.

Of course, an exact correlation of $r = 1$ would be unrealistic to achieve, especially for such a humanities-based project. A good technique might reach correlations of $r \geq 0.5$ and a great technique might reach correlations of $r \geq 0.75$.

Chapter 6

Results

This chapter shows the results from running the research on the dataset. These results focus on correlation coefficients and the differences between artistic and statistic exemplars.

6.1 Correlation Coefficient Results

One parameter for our classifier is the choice of k for k-Nearest Neighbour. Simply setting $k = 1$ has the effect of assigning the year of the nearest painting in feature space to the current test painting. Whilst setting $k = 102$ has the effect of giving the painting the mean year of the entire data set. Obviously, a point between the two is likely to be the best.

From Figure 6.1 it is apparent that for many of the feature spaces we have an optimum value of k of around 7 or 8. Table 6.1 shows the actual results for each feature space for a k of 7, where r is the correlation coefficient given by Pearson's, $P(r)$ is the p-value significance of r.

$C(n)$ is the percentage of paintings correctly classified within n years of the actual year (where $E(X) - n \leq X \leq E(X) + n$).

Technique	r	$P(r)$	$C(15)$
Edge Strength	0.0107	0.910	60%
HSV	0.112	0.237	64%
RGB	0.118	0.214	63%
HSV Histograms	0.146	0.123	64%
RGB Histograms	0.270	0.004	62%
HOG (Discrete Derivatives): 4 orientations	0.307	0.001	65%
Steerable filters: 4 orientations	0.312	0.001	68%
HOG (Discrete Derivatives): 8 orientations	0.346	<0.001	65%
HOG (Discrete Derivatives): 16 orientations	0.367	<0.001	64%
Gabor filters: 16 orientations	0.370	<0.001	67%
Gabor filters: 8 orientations	0.438	<0.001	70%
Gabor filters: 4 orientations	0.441	<0.001	71%

Table 6.1: Correlation Coefficients, ordered by strength for $k = 7$
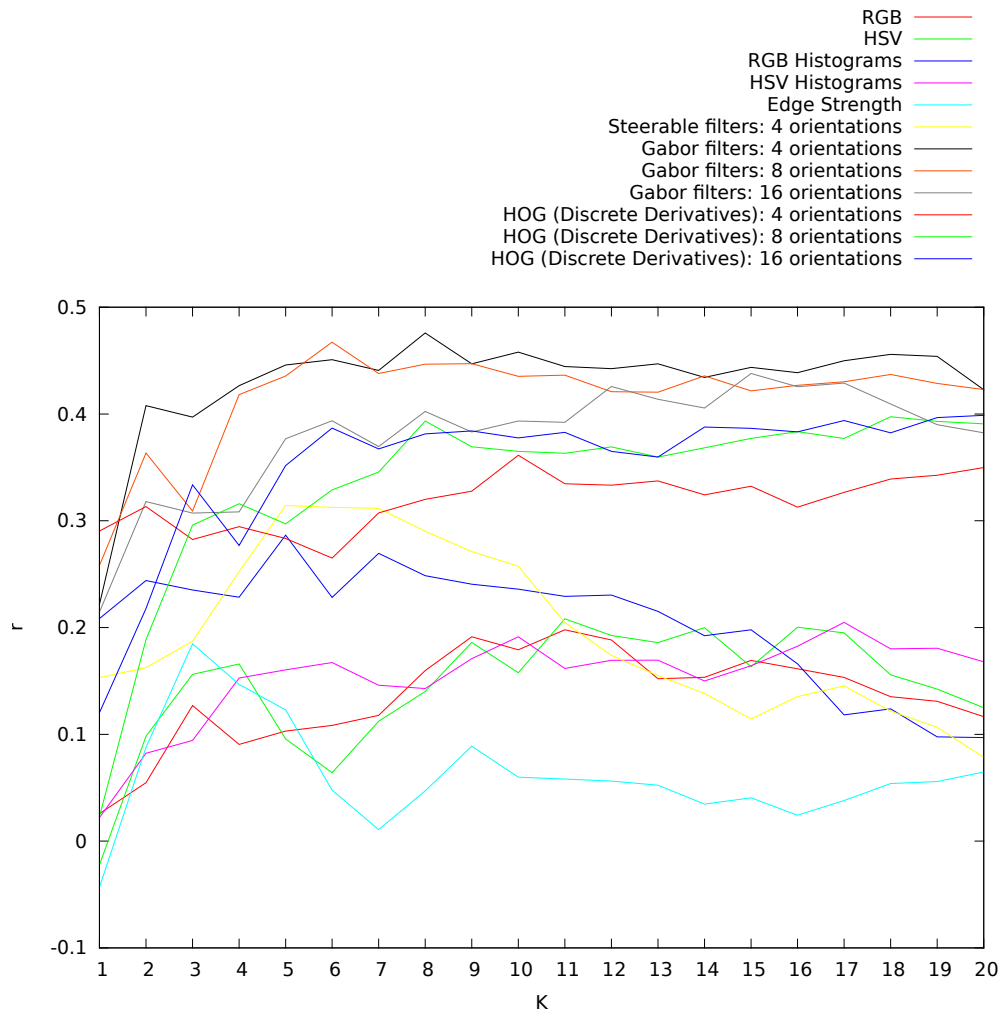
Figure 6.1: Correlation Coefficients r against k values for k-Nearest Neighbour

6.2 Ensemble Results

Using ensemble methods can both improve and weaken the correlation coefficients depending on the techniques used. The inclusion of colour histogram analysis and HSV colour-space statistical analysis typically weakens the results. Whilst combining HOG and Gabor filters typically improves the correlation coefficients beyond each technique individually. Table 6.2 shows some examples of this.

This does show that ensemble methods can help to improve the strength of analysis techniques, as expected.

Method	r	$P(r)$
RGB	0.118	0.214
Ensemble RGB Histogram and HOG (16)	0.256	0.006
RGB Histograms	0.269	0.004
HOG (Discrete Derivatives): 16 orientations	0.367	<0.001
Ensemble RGB and HOG (16)	0.372	<0.001
Gabor filters: 4 orientations	0.441	<0.001
Ensemble RGB and Gabor (4)	0.443	<0.001
Ensemble RGB, Gabor (4) and HOG (16)	0.464	<0.001

Table 6.2: Correlation Coefficients comparing individual and ensemble methods. Note how the combination of RGB Histograms and HOG has decreased performance, whilst the combination of HOG and Gabor filters has improved performance.

6.3   Exemplar Results

The intuition – that using artistically chosen exemplars could help to exploit knowledge about the way the paintings change over time – turned out to be incorrect. Results for Gabor filters with 4 orientations (the best performing method in the previous experiment) are shown in Table 6.3. Results for the other feature spaces show a similar pattern, the same distance measure (χ^2) has been used throughout.

Technique	r	$P(r)$	$C(15)$
Artistic Exemplars	0.328	<0.001	57%
Statistical Exemplars	0.383	<0.001	61%
Centroid	0.403	<0.001	64%

Table 6.3: Correlation coefficients, ordered by strength, for Exemplars

These exemplars give an interesting insight into the feature space. The paintings shown in Figures 1.1 and 1.2 are both artistic exemplars, however the earlier painting "*Snowdon, the Traeth and the Frightened Horse*", from 1948, is far from the feature space centroid for that year, whereas the later painting is very close to the feature space centroid for 1985. A visualisation of artistic exemplars and their corresponding statistical representations is given in Figure 6.2. From this you can see that artistic information does not necessarily correspond well to the feature space(s) we use. Note that whilst Figure 6.2 uses the feature space defined by Gabor filters with 4 orientations, our best performing: the pattern is similar for all other feature spaces.
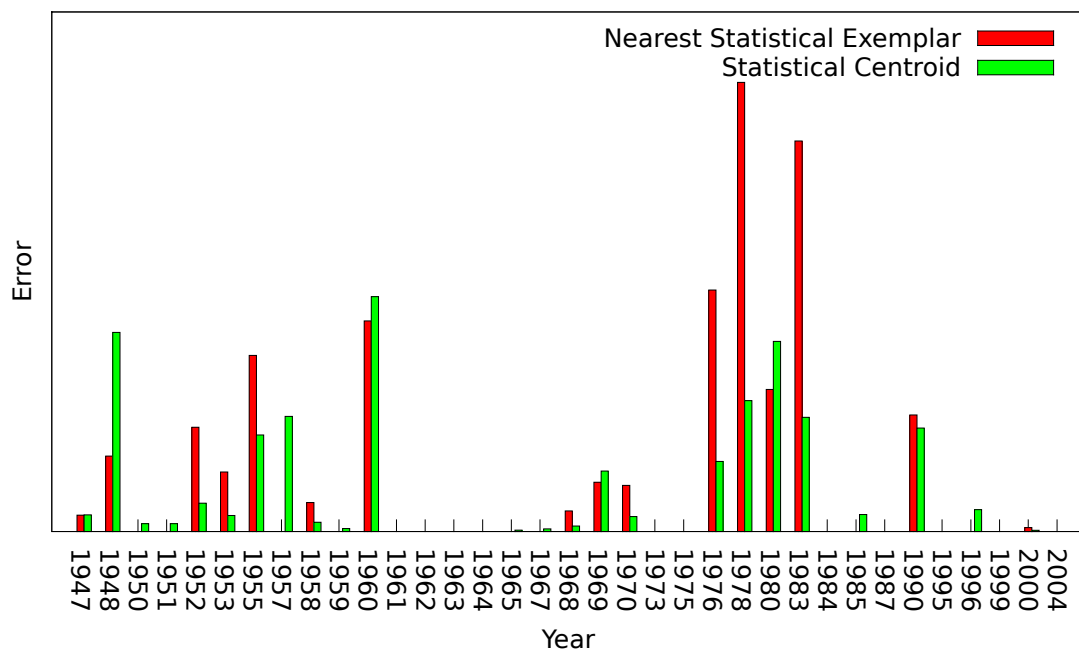
Figure 6.2: Distance in feature space from artistic to statistic exemplars (red); distance from artistic exemplar to centroid (green). Lower values indicate that the artistic exemplar is near to the mean painting for a particular year, higher values that an artistic exemplar painting is an outlier for this particular feature space

Chapter 7

Evaluation

The chapter reflects back on the project to critically evaluate the main choices made. The three main areas which this focuses on are the identification of requirements, the design decisions that were made and the use of third party tools and libraries.

This chapter also discusses the conclusions of this project and some potential future work that could come from it.

7.1   Evaluation of Requirements

Due to this being a research project, the requirements were not well defined other than the end result of being able to classify paintings. Other requirements were added during meetings as new problem areas were encountered but were never formally defined. This would have been a problem if the methodology used wasn't able to keep up with changing requirements.

Most of the aims that were defined to begin with were accomplished; in fact all but brush-stroke analysis were completed effectively. However, on further investigation the completion of brush-stroke analysis was not feasible for this project, the area is rather large and complex. Also, with various time constraints, it turned out that exemplars were a more revealing, less time-consuming area of investigation.

7.2   Evaluation of Design

The initial design evolved very little over the course of the project, the main changes made were to actually make use of inheritance to cut down on code-reuse. The design actively encouraged this as it was set up to allow this activity.

The design wasn't perfect: until most of the way through the project, objects in parameters were manipulated to have properties set during the operation, instead of using return values from functions. Eventually this began to cause problems and was changed to correctly use return values instead. Listing 7.1 shows the changes made.

Listing 7.1: Using return values instead of manipulating parameters

```
# Manipulating parameters
def analyse(self, painting):
    img = imread(painting.file_path)
    painting.data = # specific analysis on img
```

```
# Using return correctly
def analyse(self, painting):
    img = imread(painting.file_path)
    return # specific analysis on img
```

Though this seems like a trivial change, it does affect how the code is called, especially in the exemplar classification where affecting the painting data is not wanted.

7.3 Evaluation of Tools

This section is broken down into the evaluation of the choice of programming language and the image processing library used.

7.3.1 Programming Language

Python is a dynamic programming language which offers clear, reliable syntax; object orientation including multiple inheritance; full modularity; exception-based error handling; and a good range of dynamic data types.

Python was a good choice of programming language; it's dynamically typed nature allows a lot fewer restrictions and though on the initial design, fitting well with the choice of methodology.

Python was also very useful for built-in features like list comprehension (see listing 7.2 for an example), keeping the amount of code needed to build list- and dictionary- based elements down to a minimum. As well as other built-in operations on these data types, for example zip on two arrays to help graph results.

Listing 7.2: Example of List Comprehension in Python

```
year = [painting.year for painting in self.paintings
        if is_absolute(painting.year)]
```

Python also boasts a good number of libraries by default; libraries like the CSV parsing library used to read in data. These libraries provide a lot of extended functionality to the language. There is also the Python Packaging Index (PyPI) which allows the easy installation of additional packages.

A lack of experience in Python was a problem to begin with at first and a lot of the earlier code didn't utilise a lot of the useful features Python provides. However, after spending time contributing to an open source project, also written in Python, these skills quickly developed through feedback from commits and being able to read source code.

Java was a commonly suggested language, in part due to the wealth of experience there is within the department. However, on investigation into image processing libraries, it was found that most of these libraries were either difficult to work with or were not as complete as other available options.

A recent beta version of OpenCV 2.4.4 provides support for the Java programming language for desktop users. However, this is vastly more complex to include as part of a project, requiring additional tools like the Apache Ant build system. The use of a beta library would have also run the risk of running into bugs in the software and relying on the project to fix them. Due to constrained time it was the right decision not to use OpenCV with Java.

Java is also very constrained in the nature of typing and lacks a lot of the mathematical features Python offers (list comprehension, lambda functions, etc.), which helps keep the code-base clean.

This combined with the verbose nature of Java would have increased the size of the project to be less manageable.

Newer Java Virtual Machine (JVM) based languages, Scala for example, could have helped with this problem. But most of the advantages gained by choosing Java as a language - especially knowledge - would have been lost in doing so. These languages also still lack the mathematical features of Python.

As OpenCV is natively written in C++, it might have made sense to use either C or C++ to write this project in, and both were strongly considered at the start of the project.

C lacks native object-orientation[1] and is generally difficult to write "correctly". Other issues like cross-platform support were also factors in the choice against C.

C++ has similar problems to C, though it does have more advanced features, including object-orientation, which do make it easier to work with. It can still be difficult to produce programs without problems. Both C and C++ have a decent set of compilers, from the standard GNU C Compiler (GCC), which provide very basic error helping features, to LLVM based compilers like clang which ease the process of finding bugs.

It was decided that, if the project ever managed to produce anything worthy of making into a library, it would be ported to C++ and investigation would be made into contributing it back into OpenCV. This would have likely been related to the brush-stroke section of work in the project. The only other reason for using C++ or C over Python for this project, is if the processing needed to be completed within a short period of time. As this is a research project it matters very little how fast each technique completes in[2].

Newer, more popular languages like Ruby, don't have the support for image processing libraries and the language features of Ruby don't particularly beat Python's. Many other languages suffer the same flaws of not having good support for image processing libraries.

Python is not without its flaws, the lack of strictness, especially before running code, does mean a lot of errors are encountered at runtime. This is a disadvantage of the dynamically typed and interpreted nature of the language. Statically typed languages are a lot easier to pick up on typing errors, whilst Python takes a step further than this and uses what is commonly known as duck typing: "If it looks like a duck and quacks like a duck, it must be a duck." [21], this means there's no direct need for the inheritance hierarchy.

Not having a compiler to lean on for picking up syntax errors before runtime is another disadvantage which Python suffers from. However, Python is generally quick enough to run that hitting these errors at runtime isn't a huge issue. There are also tools available to help pick up on these errors. PyLint was used during the project to find syntax errors and clean up code to typical Python standards. This was useful as most of the code was difficult to test cleanly so runtime errors could be fairly common. The interpreter also has some checking for invalid syntax before runtime so it was useful to lean on that on occasion.

If the author were to take on this project again, Python still would have been the choice of language as the strengths detailed above far outweigh any of the weakness of the language.

7.3.1.1   Dependency Management

setuptools is a Python package designed to allow the easy downloading, building, installation, upgrading and uninstallation of Python packages from PyPI. It can be used to manage the dependencies of a Python project through the use of a set-up file specifying the packages required for a

---

[1]Using structures and functions points it can mimic the behaviour to a certain extent, but still lacks inheritance

[2]And, if a technique does take too long there are solutions to this problem through the canning of results, as was performed on HOG analysis (see Section 4.4.2.1)

project to run. On build of a project, all required packages will be downloaded and installed, along with any packages they in turn require.

Again this is a bonus of using Python. Java has very bad dependency management, the only tool which is widely used to do this is Apache Maven, and is not widely adopted, therefore it is not all that useful. Open Services Gateway Initiative (OSGi) is another technology which makes managing dependencies easier, but in itself doesn't handle the download and installation of packages.

Languages like C and C++ tend to use packages through the operating system or user-based compilation. This can lead to a lot of dependency issues if the user is not well versed in the installation of libraries from source. This is especially an issue on less common packages which are not provided through a package manager[3].

Newer languages, again using Ruby as an example, do have better dependency management and tend to be more tied to the language itself, rather than being a tool like setuptools. There aren't many advantages to this as PyPI is widely known.

7.3.2   Image Processing/Computer Vision Libraries

There are numerous image processing and/or computer vision libraries available for use, each with their own flavours and supported programming languages. OpenCV was the eventual choice after a lot of research into these libraries (as discussed in Section 4.1). OpenCV is one of the leading computer vision libraries freely available (released under the BSD License) and boasts an impressive number of methods. It has the benefit of being highly optimised and cross-platform.

In comparison to most other computer vision libraries OpenCV feels well thought out and polished; with good, well written, documentation for all interfaces (C, C++ and Python). Some of the Python documentation was a little difficult to use before discovering "cv2", but this was partly because it was no longer being maintained.

Other libraries, notably FIJI, lacked comprehensive, well laid out documentation. Method calls were either not obviously named or buried in so many namespaces that it was impossible to find the correct call without documentation.

7.3.2.1   OpenCV cv2

Knowing about "cv2" from the start would have been a useful piece of information, the updates to the library made it a lot easier to use. For example; calling the calculate histogram method in OpenCV is shown in Listings 7.3, whilst the same operation in "cv2" is shown in Listings 7.4. This is evidently a lot easier to call and work with.

Listing 7.3: Creating a Histogram in OpenCV

```python
import cv

image = cv.LoadImageM("file.png")

planes = [cv.CreateMat(image.rows, image.cols, cv.CV_8UC1) for i
    in xrange(3)]
planes = planes + [None]

cv.Split(image, planes)
```

---

[3]Or on Windows, which has no standardised package manager

```
hist = cv.CreateHist([255,255,255], cv.CV_HIST_ARRAY,
    [[0,255],[0,255],[0,255]], 1)
cv.CalcHist([cv.GetImage(i) for i in planes], hist)
```

Listing 7.4: Creating a Histogram in OpenCV cv2

```
import cv2

image = cv2.imread("file.png")
hist = cv2.calcHist([plane for plane in cv2.split(image)], [0],
    None, [255,255,255], [0,255,0,255,0,255])
```

Another advantage "cv2" has is its tight integration with numpy - a numeric library for Python and part of the scipy library. This integration allows many more complex operations to be performed on analysis results; operations which OpenCV does not provide or provides in ways only specific to image processing. A notable example of this is being able to take the mean along an axis to generate the centroid in statistical exemplar classification.

numpy is also useful for non-image processing use; many complex mathematical functions are pre-defined. These are useful for generating the statistical results for a given set of analysis and classification techniques, after leave-one-out cross validation is performed. The main use was for Pearson's product-moment correlation coefficient and p-value significance.

Other libraries exist for these functions, but could not reasonably be considered due to the integration with "cv2".

7.4   Conclusions

To the best of the authors knowledge this is the first work that attempts to date work by an artist by year. Similarly, it is believed that this is the first attempt to try and perform digital analysis of paintings from a range of catalogue and web images.

The results presented here show that computer vision *can* help with the job of dating art within an artist's body of work. It has been shown that strong, statistically significant correlations between a method's allocation of year and the actual year of painting; as well as the ability to classify 70% of paintings to within their actual year of painting (within a dataset that spans 6 decades). These results are not yet of great use to art historians, but it is hoped that future work will be able to improve upon this. Several statistical avenues remain to be explored: looking into feature combination and selection, and also investigate the potential for treating the year classification problem not as a nearest neighbour problem, but as an ordinal regression problem.

Future directions will also involve testing the methods presented here on the works of other artists who have shown great stylistic variation over the course of their career: one plan was to build a dataset of, for example, David Hockney works. Whilst this test has not yet been performed it is hoped it would be a success: by avoiding brushstroke detection (which we expect to be artist specific) we hope to have developed techniques with application across a broader range of artistic styles, and by building techniques which work on catalogue images rather than those captured in controlled conditions, there is a better openness to working with paintings from a wider range of artists.

There are several larger projects that could be considered for continuation from this project. Being able to display a graphical representation of the analysis performed, for each painting in a publicly-accessible format. This would take the form of a website with HTML5 elements and would be used as an aid for teaching digital image processing.

Another venture was to create a 3D model of Williams' work; due to the highly textural nature of his work this could provide more interesting analysis techniques. This is also a novel way of being able to showcase an artist's work for those who do not have access to the physical work. Again this would ideally use HTML5 elements to make the work available to the general public.

As this project never considered brush-stroke analysis it would be interesting to see how generalised some of the brush-stroke detection algorithms mentioned in Section 1.3.2 would work on an artist who primarily used a palette knife. Perhaps more interestingly, how these techniques could be modified to apply them too the palette knife strokes. A further work from this would be the ability to differentiate between different painting implements.

Unfortunately, there is a sticking point when it comes to this project. Using painting size to classify the year of a painting produces the best correlation of all other analysis techniques. There may be a mundane reason for this: cost.

At the beginning of his career Williams would have only been able to afford small canvases, especially with the amount of paint he used. As his career progressed he would have been able to afford increasingly large canvases and the paint associated. However, as this project is wanted to be generalised to other artists this may not hold true for all catalogues of differing artists.

Appendices

Appendix A

Paper Submission for ISPA 2013

The following paper entitled *Can we date an artist's work from catalogue photos?* [9] has been submitted for the 8th International Symposium on Image and Signal Processing and Analysis.

    Not all content from this paper was written by the author, however the code this paper is based upon is all the author's own work, except where explicitly stated.

Can we date an artist's work from catalogue photographs?

Alexander David Brown
Computer Science,
Aberystwyth University,
Penglais,
Aberystwyth,
Ceredigion,
Wales SY23 3DB
adb9@aber.ac.uk

Gareth Lloyd Roderick
School of Art,
Aberystwyth University,
Buarth Mawr,
Aberystwyth,
Wales SY23 1NG
glr7@aber.ac.uk

Hannah M. Dee
Computer Science,
Aberystwyth University,
Penglais,
Aberystwyth,
Ceredigion,
Wales SY23 3DB
hmd1@aber.ac.uk

Lorna M. Hughes
The National Library of Wales,
Aberystwyth,
Ceredigion,
Wales SY23 3BU
lorna.hughes@llgc.org.uk

Abstract—Computer vision has addressed many problems in art, but has not yet looked in detail at the way artistic style can develop and evolve over the course of an artist's career. In this paper we take a computational approach to modelling stylistic change in the body of work amassed by Sir John "Kyffin" Williams, a nationally renowned and prolific Welsh artist. Using images gathered from catalogues and online sources, we use a leave-one-out methodology to classify paintings by year; despite the variation in image source, size, and quality we are able to obtain significant correlations between predicted year and actual year, and we are able to guess the age of the painting within 15 years, for around 70% of our dataset. We also investigate the incorporation of expert knowledge within this framework by consdering a subset of paintings chosen as *exemplars* by a scholar familiar with Williams' work.

I. INTRODUCTION

This paper presents a interdisciplinary computational study into the modelling of artistic style, and how this style changes over time. Sir John Kyffin Williams (1918-2006) was one of the predominant figures in Welsh art of the twentieth century. Kyffin – as he was almost universally known in Wales – studied at the Slade School of Art and worked as an art master at Highgate School, before returning to live on his native Anglesey in 1973. He was a prolific painter and once claimed to have painted "two pictures per week when in London, and three per week when in Wales.'[1, p.209] With a career spanning from the mid-1940s to approximately 2004, this rate amounts to a large body of work.

His technique evolved from a very representational style to something more expressive, which retained representational qualities: the computer scientists on our team would say that the paintings became more *blocky*; the art historians that his landscapes are almost constructed with swathes of textural paint. His was a style characterised by thick impasto paint, applied almost exclusively with palette knife, although the application technique appears to change over time. This development of style led us to wonder: is it possible to date the pictures from images alone?

Through a collection of digital photographs of oil paintings, collected from museum websites, catalogues and other sources, we first investigate whether it is possible to date a painting based upon image features. We show that using a K-nearest neighbour classifier, tested using a leave-one-out methodology, we can obtain a strong correlation between image feature descriptors and year of painting. We go on to investigate whether exemplar based methods are able to improve on this, using what we call *artistic exemplars* (paintings selected by an expert as being typical for a particular year) and *statistic exemplars* (paintings which are near the centre of year-based clusters in feature space).

II. BACKGROUND

Although also a portrait painter, Williams is primarily known for his landscape paintings of north west Wales and Anglesey. While his technique and style changed over the years, his landscapes in oil are instantly recognisable, often featuring bold chunks of colour, and various points during his career bold black outlines to figures and landscapes features. Greens, browns and greys often form the palette of his paintings of the Welsh landscape. These colour selections seem appropriate for the artist's claim that melancholy, derived from the "dark hills, heavy clouds and enveloping sea mists", is a national characteristic of the Welsh [1]. This combination of colour selection and technique seems appropriate for the depiction of the areas where he painted. Many of his most successful paintings are said to have a "dark quality" in depicting "rain lashed hillsides," and it was this darkness which "makes his landscapes so distinctively Welsh" [2]. Figure 1 shows an early Williams painting, complete with rain lashed hillsides.

The aesthetic of Williams's Welsh landscapes is contrasted by the paintings he made following a trip to Patagonia to paint the landscape and people of the Welsh communities there in 1968 as part of a Winston Churchill Foundation scholarship. The colours and application of paint in pictures produced in following this journey (such as *Lle Cul*, *Henry Roberts*, *Bryngwyn Patagonia*, *Euros Hughes Irrigating his Fields*, all 1969, National Library of Wales) differ starkly from paintings of Welsh landscapes, incorporating pinks, purples and oranges. This contrast, combined with the fact that the Patagonian pictures were produced during a definite period of time has reinforced our interest in the analysis of the formal qualities of pictures from different collections remotely, using digital images.

Fig. 1. "*Snowdon, the Traeth and the Frightened Horse*", Sir John Kyffin Williams, 1948: note curved strokes, rather than blocky application

Williams's work is well represented in public collections in Wales (particularly at the National Library of Wales, the National Museums and Galleries of Wales and Oriel Ynys Môn, Anglesey). His pictures, often depicting the landscape and people of north-west Wales were also tremendously popular with the art buying public. Of the 325 paintings by Williams in public collections in the UK listed on the BBC/Public Catalogue Foundation's "Your Paintings" website, 212 of them are in the collections of the National Library of Wales [3]. Many of these paintings were bequeathed to the Library as part of a larger bequest by the artist (including works on paper and other archival material). Many of the pictures which came to the library from the artists studio had little in the way of metadata, and as such have been catalogued with large date-ranges estimating the dates of production. This uncertainty in metadata is another motivating force behind the current project.

A. Taking a digital humanities approach to art history

Digital humanities is an established area of research that brings together digital content, tools and methods in order to address and create new knowledge across the disciplines. Digital humanities approaches can be seen in two distinct types of inquiry. The first is to carry out *traditional* humanities research more effectively or efficiently, by applying computational methods or approaches to digitized humanities sources (originally text, image, or audio-visual content from archives or libraries). Using John Unsworths definition of "scholarly primitives" [4] digital humanities scholarship customarily involves the use of digital tools and methods for discovering, annotating, comparing, referring, sampling, illustrating, or representing humanities data. A classic example of this sort of work would be the use of concordances and other computer-based analysis of digitized primary sources that have been processed by optical character recognition software to count, classify, or interpret digital texts (see, for example, the Historical Concordance of the Welsh language [5]). The second strand of digital humanities inquiry is the development of new research questions that can only be developed

through the synthesis of digital content, tools and methods: work that would have otherwise been unimaginable [6]. This type of research is by necessity multi-disciplinary, drawing together expertise to be found across humanities, scientific and engineering disciplines, as well as involving content experts from libraries, archives and museums. However, in order to be truly transformative, this type of research must also be interdisciplinary.

The National Library of Wales now has a research programme in digital collections, which is a forum for investigation into the digital collections of Wales in collaboration with academics and students at universities in Wales and beyond, in order to develop new research based around the digital content created by the Library [7]. The research project described in this article is an example of a digital humanities collaborative venture, bringing together digital humanists, art historians, and computer scientists. The results of this research have value across all these groups. Arts historians are able to better investigate a large corpus of digital paintings through the application of computer science approaches to this content, and computer scientists are able to configure new approaches in imaging to working with a complex humanities data set.

B. Computer vision and the analysis of paintings

Stork, in his 2009 review paper, presents an overview of the field of digital painting analysis [8]. Leaving aside structural aspects of painting analysis (for example, there is a rich seam of work looking at the geometry of figurative art, for example [9]) most work in the area of style analysis is aimed at authentication. With the problem of authentication, one tries to build a two class classifier for a painter where the classes in question are "painted by artist X" or "not painted by X" (e.g. Irfan and Stork's feature based classifier for authenticating Jackson Pollock artworks, [10]).

When we consider computer vision-based analysis of painterly style we find that the vast majority of work concentrates on brush stroke detection and analysis. For example,

Berezhnoy and colleagues in [11] detect brush-strokes by moving a circular filter across the whole painting to find the ridges of strokes, then filling any unbroken areas. They then shrunk these areas to a single pixel line and fitted a nth order polynomial to this line.

Li et al [12] use a combination of edge analysis and clustering in colour space to determine strokes; a number of heuristics involving branching, stroke-width modeling, and gap filling are then used to refine the original brush stroke estimates. One interesting element of this work, from our perspective, is the ability to date some of Van Gogh's paintings to a known period in his career. To the best of our knowledge this is the only other work which aims, like us, to date work: that is, to automatically place an artwork in the context of the artists' own body of work.

Techniques based upon stroke analysis, whilst applicable to the work of some artists, are not applicable to all. In particular, Kyffin Williams painted with a palette knife and whilst there are clear *strokes* identifiable in his style, these vary widely in size and shape, so the morphological techniques which can detect strokes in Van Gogh's work are unlikely to pay off when considering the blockier paintings in the Williams oeuvre. Another difference of note is that much work on computerised painting analysis (including [12], [11]) is based upon high resolution scans acquired in controlled conditions, whereas the current paper deals instead with a collection of photographs from catalogues, websites, and other disparate sources.

III. THE IMAGE DATASET

Our image dataset consists of 325 paintings, with associated metadata. Metadata includes title, year or year ranges (for those works where year is unknown but can be estimated by curators), genre, original painting size, painting materials and image size.

These photographs of paintings are challenging in and of themselves: they are not colour calibrated; some suffer from reflections (towards the end of his life Kyffin painted using exceptionally thick and textural strokes, which gives specularities on the catalogue images); they are at varying resolutions; and come from a range of different cameras. Image size bears little relation to the original painting size, and some images are even optimised for the web. Table I below summarises the dataset; Figure 2 shows a late Williams painting.

Type	Number	Number (Known date)	Notes
Landscapes	247	64	
Portraits	52	35	
Seascapes	11	2	
Still lifes	4	1	
Other	8	0	Other or studies

TABLE I.   A SUMMARY OF OUR KYFFIN WILLIAMS PAINTING DATASET

IV. METHODOLOGY

Within our database of 325 paintings, we know the actual year of painting for 102 artworks. In order to determine the accuracy of our results, rather than work with the full dataset (and work with images with uncertain metadata in the form

Fig. 2.   *"Above Carneddi, No. 2"*, Sir John Kyffin Williams 1985: note much blockier style and changed use of colour
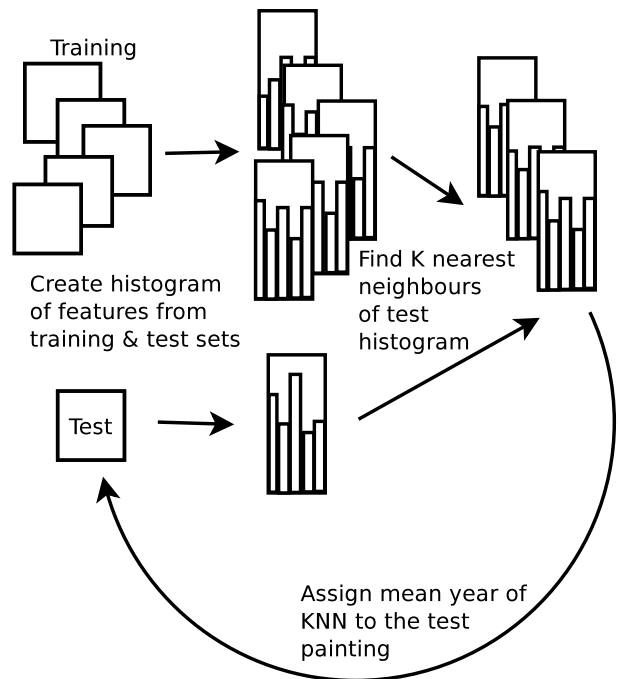
Fig. 3.   Overview of the classification methodology

of date ranges), we have used a leave-one-out cross validation methodology. This involves us taking a painting for which we know the year, and then using our classifier to guess that year; thus we are able to tell whether we are right. We are also able, if we are wrong, to determine exactly how wrong we are.

To simplify the classification stage we use a K-Nearest Neighbour (KNN) classifier with the other 101 paintings for which we know the date. KNN is a fast, non-parametric classifier which makes no assumptions about the underlying patterns in the data, merely that paintings from around the same time will be similarly located in our feature space(s). Whilst we suspect that there may be some broader underlying trend in the change of style, for this work have concentrated on features for classification rather than the question of classification or regression itself.

Thus for each feature set, we take all paintings for which we know the year of creation; select one painting, and find its nearest neighbours within that feature space. The year assigned by our classifier to that painting is the mean of the K neighbours; we found this provided better results than both median and mode. Figure 3 provides an overview of this classification methodology.

We also know that painting's actual year, and we can plot actual against predicted year for all known-year paintings. To measure goodness of fit, the Pearson's product-moment correlation coefficient was calculated on these orderings; this provides us with a performance measure of each classifier. It is also possible to test Pearson's r for statistical significance; thus significance levels are reported alongside r in this paper. With all of the feature spaces we consider, it is possible treat the painting descriptors as histograms. This allows us to use a single distance measure, namely chi-squared, in our K-nearest neighbour classification.

V. AN EXPLORATION OF COLOUR AND TEXTURE FEATURES

The digital analysis of paintings is a broad reseach area. Within the methodology we have selected, there are many feature spaces which could be useful: from simple analysis of the way in which colour changes over time, through edge detection, to texture analysis. We have concentrated on lower level image features – colours, textures, and edges – rather than attempt to extract brush strokes. As mentioned earlier, Williams painted with a pallette knife rather than a brush, and his work is characterised by angularity rather than identifiable "strokes". Our motivation for this is not only due to these issues with painterly style, but also because of the variation in image quality. By concentrating on simpler features we hope to retain some robustness to variation in image capture and quality. In this section we describe the various feature sets and feature spaces we have explored; results for each of these are presented in Section VI below.

There is a clear (to the eye) trend in colour usage, as the paintings get "gloomier" over time. Thus, we started with simple colour-space analysis: taking the mean RGB for each painting and using this with our KNN classfier; we also tested other colour spaces, such as HSV. Promisingly this provided us with a positive correlation. Remaining with the colour variation theme, we then used colour histograms, which provide a more precise representation of the way Williams used colour. These histograms were developed by counting the number of pixels within a particular colour range for each painting, and then building a normalised histogram representing the colour usage.

As a lot of Kyffin Williams' paintings are highly textural, edge detection and texture analysis were also thought to be a good avenue to explore. Firstly, we investigated simple *edginess*; as a rough estimate of the the edge properties of the artworks we apply a Canny [13] edge detector to the paintings, and then use a count of edge pixels as our feature.

Texture analysis can be thought of as a continuation of edge detection. Instead of taking simply the strength and number of edges, we create a histogram of orientated gradients as in [14]. In this way we begin to build up a richer representation of the texture of a painting. Given the change in style of Kyffin

Williams' work, moving away from figurative representations with curved lines towards more blocky rectilinear "brush" strokes, we expect these edge orientation frequencies to change over time. To this end we used simple steerable filters S, applied to the image at 0, $\frac{\pi}{4}$, $\frac{\pi}{2}$ and $\frac{3\pi}{4}$.

$$S\left(\frac{\pi}{2}\right) = \left(\begin{array}{ccc} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{array} \right) \qquad (1)$$

Equation 1 shows a sample steerable filter, in this case $S(\frac{\pi}{2})$, the filter which gives the highest response when presented with horizontal lines. By convolving each image with filters tuned to different orientations, we can build a histogram recording the frequency of lines at each orientation. Figures 4 and 5 show a painting, and the resulting edge orientation histogram; it is clear from this that there are more horizontal edges in this particular image due to the clear peak at $\frac{\pi}{2}$.

Fig. 4. "*Coastal Sunset*", Sir John Kyffin Williams. Date unknown, thought to be in the range 1990-2006
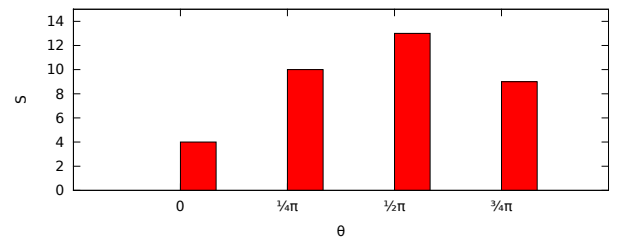
Fig. 5. Steerable filter strength $S(\theta)$ on the example image in figure 4

Gabor filters are linear filters which can be tuned to a greater range of angles and frequencies than simple steerable filters, which in turn results in a more accurate representation of the texture of the painting. The general equation for a Gabor filter is given in Equation 2.

$$g_e(x) = \frac{1}{2\pi\sigma_x\sigma_y} e^{-\frac{1}{2}\left(\frac{x^2}{\sigma_x} + \frac{y^2}{\sigma_y}\right)} \cos(2\pi\omega_{x_0}x + 2\pi\omega_{y_0}y) \quad (2)$$

Where $(\omega_{x_0}, \omega_{y_0})$ defines the centre frequency, and (σ_x, σ_y) the spread of the Gaussian window [15]. In this

work we use Gabor filters tuned to equally spaced orientations to build a histogram representing line orientations in each painting, and present results below for histograms built from the output of 4, 8 and 16 filter orientations.

The final method for producing histograms we consider involves the application of two discrete derivative masks to the image to get the gradient of x and y, and then to work out the gradient direction at each point. These gradient directions are then summarised in a histogram of oriented gradients, providing a yet richer representation of the texture of the image. This is similar to the method described in [14]; again we present results on the output of 4, 8 and 16 orientations.

VI. YEAR CLASSIFICATION RESULTS

The one parameter of our classifier is the choice of K in K-nearest neighbour. Simply setting $K = 1$ has the effect of assigning the year of the nearest painting in feature space to the current test painting, whereas setting $K = 102$ has the effect of giving each painting the mean value of the entire dataset. Clearly a point between these two extremes would be best; from Figure 6 we can see that for many of the feature spaces we consider, the optimum K value is around 7 or 8. Pearson's correlation coefficients r for K=7, alongside $P(r)$ are presented in Table II. A further measure of classification accuracy is also presented: this is the percentage of paintings for which our classifier manages to date the artwork in question within 15 years of actual painting date. This measure, $C(n)$, provides an easy to understand measure of classification accuracy.
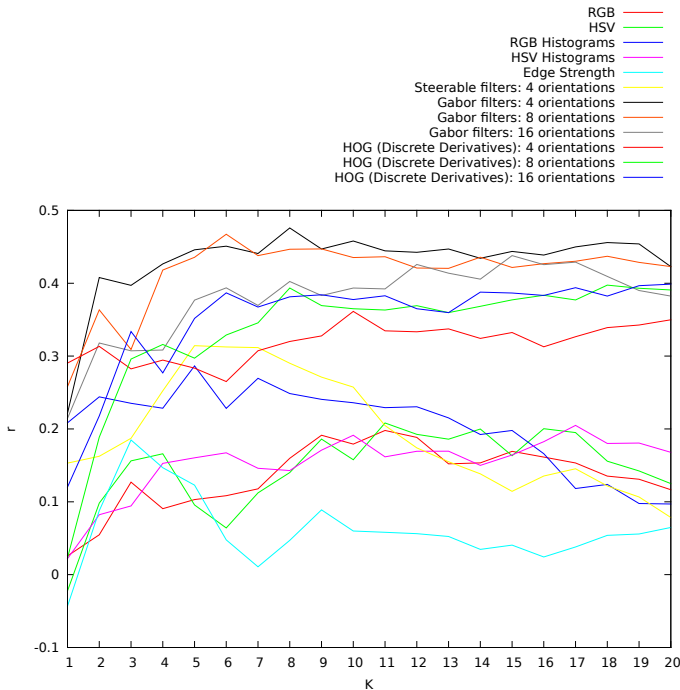

Fig. 6. Correlation Coefficients r against K values for K-Nearest Neighbour

From Table VI it is clear to see that whilst the dates predicted by our methods correlated fairly well with the actual painting dates, and these correlations are mostly significant,

Technique	r	$P(r)$	$C(15)$
Edge Strength	0.0107	0.910	60%
HSV	0.112	0.237	64%
RGB	0.118	0.214	63%
HSV Histograms	0.146	0.123	64%
RGB Histograms	0.270	0.004	62%
HOG (Discrete Derivatives): 4 orientations	0.307	0.001	65%
Steerable filters: 4 orientations	0.312	0.001	68%
HOG (Discrete Derivatives): 8 orientations	0.346	<0.001	65%
HOG (Discrete Derivatives): 16 orientations	0.367	<0.001	64%
Gabor filters: 16 orientations	0.370	<0.001	67%
Gabor filters: 8 orientations	0.438	<0.001	70%
Gabor filters: 4 orientations	0.441	<0.001	71%

TABLE II.   CORRELATION COEFFICIENTS, ORDERED BY STRENGTH, FOR K=7

we are able to date paintings within 15 years in 71% of cases. Whilst this result is not yet good enough to be of use to the art history world, it is promising.

VII. EXEMPLARS: CAN WE IMPROVE RESULTS BY INCORPORATING EXPERT KNOWLEDGE?

We have also investigated the utility of incorporating expert knowledge within our framework. For each year represented in our collection we asked Dr Paul Joyner, of the National Library of Wales, to choose the one painting which best represents the artist's work for that year. Dr Joyner is a member of the Trustees of the Kyffin Williams Estate and he has written widely on Welsh Art and Kyffin Williams. These chosen paintings we consider to be connoisseurially/artistically selected exemplars (*artistic exemplars*, for short), which we can then use as a representation of that particular year.

The data on artistic exemplars opens up the options for different methods of classification. Rather than using K-nearest neighbour to classify each point in the feature space, we take the year of the nearest exemplar, and assign that year to the painting in question. If these exemplars are indeed indicative of the artists' output for that year, they should prove to be useful *anchors* in our feature spaces. To compare with this, we can also determine *statistical exemplars* either by using the centroid in feature space for a particular year (which provides us with a point in feature space which will not correspond to an actual painting), or the nearest actual painting to the feature space centroid for a year. The former technique does not, strictly speaking, give us an exemplar; the latter chooses as exemplar the painting which best represents a particular year according to a particular feature space.

Our intuition – that using artistically chosen exemplars could help us to exploit knowledge about the way the paintings change over time – turned out to be incorrect; results for Gabor filters with 4 orientations (the best performing method in our previous experiment) are shown in Table III. Results for the other feature spaces show a similar pattern, the same distance measure (χ^2) has been used throughout.

These exemplars give an interesting insight into the feature space. The paintings shown in Figures 1 and 2 are both artistic exemplars, however the earlier painting "*Snowdon, the Traeth and the Frightened Horse*", from 1948, is far from the feature space centroid for that year, whereas the later painting is very close to the feature space centroid for 1985. A visualisation

Technique	r	$P(r)$	$C(15)$
Artistic Exemplars	0.328	<0.001	57%
Statistical Exemplars	0.383	<0.001	61%
Centroid	0.403	<0.001	64%

TABLE III.   CORRELATION COEFFICIENTS, ORDERED BY STRENGTH, FOR EXEMPLARS

of artistic exemplars and their corresponding statistical representations is given in Figure 7. From this you can see that artistic information does not necessarily correspond well to the feature space(s) we use. Note that whilst Figure 7 uses the feature space defined by Gabor filters with 4 orientations, our best performing: the pattern is similar for all other feature spaces.
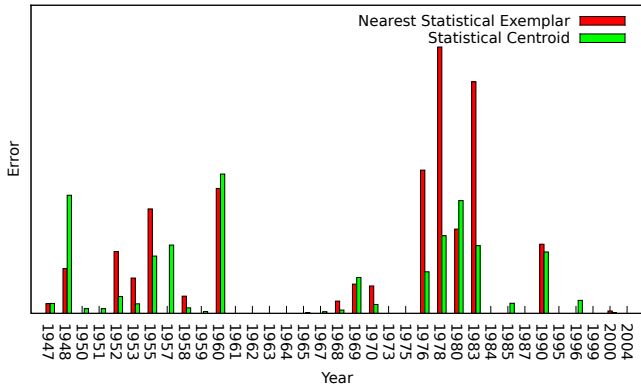

Fig. 7. Distance in feature space from artistic to statistic exemplars (red); distance from artistic exemplar to centroid (green). Lower values indicate that the artistic exemplar is near to the mean painting for a particular year, higher values that an artistic exemplar painting is an outlier for this particular feature space

VIII. CONCLUSIONS AND FUTURE DIRECTIONS

To the best of our knowledge this is the first work that attempts to date work by an artist by year. Similarly, we believe we are the first to try and perform digital analysis of paintings from a range of catalogue and web images. The use of expertly chosen artistic exemplar paintings is also novel, and whilst this does not perform well within the classification context it does provide an interesting insight into the feature spaces we explore.

The results presented here show that computer vision *can* help with the job of dating art within an artist's body of work. We are able to show strong, statistically significant correlations between our method's allocation of year and the actual year of painting; we are also able to classify 70% of paintings to within 15 years of their actual year of painting (within a dataset that spans 6 decades). These results are not yet of great use to art historians, but we are hopeful that future work will be able to improve upon this. Several statistical avenues remain to be explored: we will look into feature combination and selection, and also investigate the potential for treating the year classification problem not as a nearest neighbour problem, but as an ordinal regression problem.

Future directions will also involve testing the methods presented here on the works of other artists who have shown great stylistic variation over the course of their career: we

would like to build a dataset of, for example, David Hockney works. Whilst we have not yet performed this test we are hopeful of success: by avoiding brushstroke detection (which we expect to be artist specific) we hope to have developed techniques with application across a broader range of artistic styles, and by building techniques which work on catalogue images rather than those captured in controlled conditions, we are open to working with paintings from a wider range of artists.

REFERENCES

[1] K. Williams, *Across the Straits: An Autobiography*. Llandysul, Wales: Gomer, 1993.

[2] J. Davies, *100 Welsh heroes*. Aberystwyth, Wales: Culturenet Cymru, 2004.

[3] "Your Paintings - Kyffin Williams," Mar. 2013. [Online]. Available: http://www.bbc.co.uk/arts/yourpaintings/paintings/search/painted_by/kyffin-williams_artists

[4] J. Unsworth, "Scholarly primitives: what methods do humanities researchers have in common, and how might our tools reflect this?" King's College London, 2000. [Online]. Available: http://www3.isrl.illinois.edu/ unsworth/Kings.5-00/primitives.html

[5] "Corpws Hanesyddol yr Iaith Gymraeg 1500-1850/A Historical Corpus of the Welsh Language 1500-1850," 2004, http://people.ds.cam.ac.uk/dwew2/hcwl/menu.htm.

[6] L. M. Hughes, *Evaluating and Measuring the Value, Use and Impact of Digital Collections*. London: Facet, 2011.

[7] http://www.llgc.org.uk/research.

[8] D. G. Stork, "Computer vision and computer graphics analysis of paintings and drawings: An introduction to the literature," 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.

[9] A. Criminisi, M. Kemp, and A. Zisserman, "Bringing pictorial space to life: computer techniques for the analysis of paintings," in *Proc. Computers and the History of Art (CHArt)*, 2002.

[10] M. Irfan and D. G. Stork, "Multiple visual features for the computer authentication of jackson pollock's drip paintings: beyond box counting and fractals," in *Proc. SPIE 7251, Image Processing: Machine Vision Applications II*, 2009.

[11] I. E. Berezhnoy, E. O. Postma, and H. J. van den Herik, "Automatic extraction of brushstroke orientation from paintings," *Machine Vision and Applications*, vol. 20, no. 1, pp. 1–9, 2009.

[12] 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, Jun. 2012.

[13] J. F. Canny, "A computational approach to edge detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. PAMI-8, no. 6, pp. 679–698, Nov. 1986.

[14] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, ser. CVPR '05, vol. 1. Washington, DC, USA: IEEE, Jun. 2005, pp. 886–893 vol. 1.

[15] A. K. Jain and F. Farrokhnia, "Unsupervised texture segmentation using gabor filters," *Pattern Recognition*, vol. 24(12), pp. 1167–1186, 1991.

Appendix B

3rd Party Libraries and Tools

2.1 Python 2.7

`http://www.python.org`

2.1.1 setuptools

`http://pypi.python.org/pypi/setuptools`

2.1.2 scipy

`http://www.scipy.org/` [19]

2.1.3 numpy

`http://www.numpy.org/` [19]

2.1.4 matplotlib

`http://matplotlib.org/`

2.1.5 liac-arff

`https://github.com/renatopp/liac-arff`

2.2 OpenCV

`http://opencv.org/` [8]

2.2.1 OpenCV Python

`http://opencv.willowgarage.com/wiki/PythonInterface` [8]

2.3 Weka 3

`http://www.cs.waikato.ac.nz/ml/weka/` [15]

2.4   git

```
http://git-scm.com/
```

2.4.1   github

```
https://github.com/
```

Appendix C

Code Samples

3.1 Example CSV Parsing Code

Listing C.1: Example CSV Parsing Code from http://docs.python.org/2/library/csv.html

```
>>> import csv
>>> with open('eggs.csv', 'rb') as csvfile:
...     spamreader = csv.reader(csvfile, delimiter=' ',
    quotechar='|')
...         for row in spamreader:
...             print ', '.join(row)
Spam, Spam, Spam, Spam, Spam, Baked Beans
Spam, Lovely Spam, Wonderful Spam
```

3.2 argparse Example Code

Listing C.2: Example argparse Code from http://docs.python.org/2/library/argparse.html

```
import argparse

parser = argparse.ArgumentParser(description='Process some
    integers.')
parser.add_argument('integers',
                    metavar='N',
                    type=int,
                    nargs='+',
                    help='an integer for the accumulator')
parser.add_argument('--sum',
                    dest='accumulate',
                    action='store_const',
                    const=sum, default=max,
                    help='sum the integers (default: find the max
                        )')

args = parser.parse_args()
```

```
print args.accumulate(args.integers)
```

3.3   Gabor Filter Example Implementation

Listing C.3: Example implementation of a Gabor Filter in MATLAB [26]

```matlab
function gb=gabor_fn(bw,gamma,psi,lambda,theta)
% bw     = bandwidth, (1)
% gamma = aspect ratio, (0.5)
% psi    = phase shift, (0)
% lambda= wave length, (>=2)
% theta = angle in rad, [0 pi)

sigma = lambda/pi*sqrt(log(2)/2)*(2^bw+1)/(2^bw-1);
sigma_x = sigma;
sigma_y = sigma/gamma;

sz=fix(8*max(sigma_y,sigma_x));
if mod(sz,2)==0, sz=sz+1;end

% alternatively, use a fixed size
% sz = 60;

[x y]=meshgrid(-fix(sz/2):fix(sz/2),fix(sz/2):-1:fix(-sz/2));
% x (right +)
% y (up +)

% Rotation
x_theta=x*cos(theta)+y*sin(theta);
y_theta=-x*sin(theta)+y*cos(theta);

gb=exp(-0.5*(x_theta.^2/sigma_x^2+y_theta.^2/sigma_y^2)).*cos(2*
    pi/lambda*x_theta+psi);
```

3.4   OpenCV Histogram Example Code

Listing C.4: Example Histogram calculation and displaying code from OpenCV [8].

```python
# Taken from: http://opencv.willowgarage.com/documentation/
    python/imgproc_histograms.html#calchist
# Calculating and displaying 2D Hue-Saturation histogram of a
    color image

import sys
import cv
```

```python
def hs_histogram(src):
    # Convert to HSV
    hsv = cv.CreateImage(cv.GetSize(src), 8, 3)
    cv.CvtColor(src, hsv, cv.CV_BGR2HSV)

    # Extract the H and S planes
    h_plane = cv.CreateMat(src.rows, src.cols, cv.CV_8UC1)
    s_plane = cv.CreateMat(src.rows, src.cols, cv.CV_8UC1)
    cv.Split(hsv, h_plane, s_plane, None, None)
    planes = [h_plane, s_plane]

    h_bins = 30
    s_bins = 32
    hist_size = [h_bins, s_bins]
    # hue varies from 0 (~0 deg red) to 180 (~360 deg red again
        */
    h_ranges = [0, 180]
    # saturation varies from 0 (black-gray-white) to
    # 255 (pure spectrum color)
    s_ranges = [0, 255]
    ranges = [h_ranges, s_ranges]
    scale = 10
    hist = cv.CreateHist([h_bins, s_bins], cv.CV_HIST_ARRAY,
        ranges, 1)
    cv.CalcHist([cv.GetImage(i) for i in planes], hist)
    (_, max_value, _, _) = cv.GetMinMaxHistValue(hist)

    hist_img = cv.CreateImage((h_bins*scale, s_bins*scale), 8,
        3)

    for h in range(h_bins):
        for s in range(s_bins):
            bin_val = cv.QueryHistValue_2D(hist, h, s)
            intensity = cv.Round(bin_val * 255 / max_value)
            cv.Rectangle(hist_img,
                        (h*scale, s*scale),
                        ((h+1)*scale - 1, (s+1)*scale - 1),
                        cv.RGB(intensity, intensity, intensity)
                        ,
                        cv.CV_FILLED)
    return hist_img

if __name__ == '__main__':
    src = cv.LoadImageM(sys.argv[1])
    cv.NamedWindow("Source", 1)
    cv.ShowImage("Source", src)

    cv.NamedWindow("H-S_Histogram", 1)
```

```
cv.ShowImage("H-S_Histogram", hs_histogram(src))

cv.WaitKey(0)]
```

3.5   Computer Vision Research Code

3.5.1   OpenCV

Listing C.5: Using OpenCV to blur an image

```
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)
```

3.5.2   cv2

Listing C.6: Using cv2 to blur an image

```
import cv2

im = cv2.imread("tux.png")
blur = cv2.blur(im, 9)
cv2.imwrite("tux_blurred_cv2.png", blur)
```

3.5.3   FIJI

Listing C.7: Using FIJI 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);
    }
}
```

3.5.4 IVT

Listing C.8: Using IVT to blur an image

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

int main(int argc, char **argv) {
        CByteImage in;
        CByteImage out;
        char* from = "tux.png";
        char* to = "tux_ivt.png";

        in.LoadFromFile(from)
        out = CByteImage(in);
        ImageProcessor::GaussianSmooth(&in, &out, 1.0f, 3);
        out.SaveToFile(to)
}
```

Appendix D

Spreadsheet Data

Filename	ID	Title	Catalogue entry BBC YP	exemplar year
154.jpg	154	Landscape at Llanaelhaearn	1947	1947
258.jpg	258	Snowdon, the Traeth and the Frightened Horse	1948	1948
155.jpg	155	Landscape with Cattle	c.1950	1950
286.jpg	286	The Dark Lake	1951	1951
288.jpg	288	The Moelwyns from Aberglasyn	c.1952	1952
238.jpg	238	Self Portrait	c.1953	1953
051.jpg	51	Cottages, Llanddona	c.1955	1955
185.jpg	185	Mountain Landscape	1957	1957
023.jpg	23	Capel Carmel	c.1958	1958
246.jpg	246	Snow above Beddgelert	c.1959	1959
206.jpg	206	Nant Ffrancon from Llandegfan	c.1960	1960
260.jpg	260	Snowstorm off Penmon	1961	1961
053.jpg	53	Cottages, Mynydd Bodafon	c.1962	1962
120.jpg	120	German Girl	c.1963	1963
243.jpg	243	Sir David Hughes Parry	1964	1964
242.jpg	242	Sir Charles Evans	c.1965	1965
087.jpg	87	Farm below Crib Goch	1967	1967
248.jpg	248	Snow on Siabod	c.1968	1968
158.jpg	158	Lle Cul, Patagonia	1969	1969
073.jpg	73	Deserted Farm, Llanrhuddlad	c.1970	1970
275.jpg	275	Sun and Cloud on Lliwedd	1973	1973
093.jpg	93	Farm, Llanfairynghornwy	c.1975	1975
018.jpg	18	Blaen Nant	1976	1976
237.jpg	237	Sea at Trearddur	c.1976	1976
017.jpg	17	Blaen Ffrancon No.1	c.1978	1978
107.jpg	107	Farmers on the Carneddau	c.1980	1980
105.jpg	105	Farmer below the Ridge	c.1983	1983
202.jpg	202	Mrs Rowlands	c.1984	1984
003.jpg	3	Above Carneddi, No.2	c.1985	1985
264.jpg	264	Storm at Trearddur	1987	1987
165.jpg	165	Llyn-y-Cau, Cader Idris	c.1990	1990
267.jpg	267	Storm, Porth Cwyfan	1995	1995
268.jpg	268	Storm, Trearddur	1996	1996
214.jpg	214	Pengwyryd	c.1999	1999
020.jpg	20	Bryn Cader Faner	2000	2000
278.jpg	278	Sunset, Anglesey	2004	2004

Table D.1: Exemplar Spreadsheet Data

Annotated Bibliography

[1] "Corpws hanesyddol yr iaith gymraeg 1500-1850/A historical corpus of the welsh language 1500-1850," 2004. [Online]. Available: http://people.ds.cam.ac.uk/dwew2/hcwl/menu.htm

    An example of another digital humanities project.

[2] "Research programme in digital collections at NLW," Sept. 2012. [Online]. Available: http://www.llgc.org.uk/research

    Information into the National Library of Wales research scheme into digital collections.

[3] "Your paintings - kyffin williams," Mar. 2013. [Online]. Available: http://www.bbc.co.uk/arts/yourpaintings/paintings/search/painted_by/kyffin-williams_artists

    The digital collection of Sir John Kyffin Williams' work.

[4] J. Bacardit and X. Llorà, "Large-scale data mining using genetics-based machine learning," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 3, no. 1, pp. 37–61, Jan. 2013. [Online]. Available: http://dx.doi.org/10.1002/widm.1078

    A paper recommended by Julie Greensmith for information on a Learning Classifier System (LCS) which Julie believes will yield good results with the Kyffin Williams project.

[5] S. J. Belongie, J. Malik, and J. Puzicha, "Matching shapes," in *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, vol. 1. IEEE, 2001, pp. 454–461 vol.1. [Online]. Available: http://dx.doi.org/10.1109/iccv.2001.937552

    Defines the concept of Shape Contexts, which have similarities to Circular Histogram of Orientated Gradient Descriptors.

[6] 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

    Defines a method of analysing brushstrokes by applying a circular filter across a digital image to pick up the ridges of a brushstroke. This can then be used to pick out individual brushstrokes in order to be able to fit a nth order polynomial to them. Though this paper focuses on authenticating Van Gogh's paintings, it could easily be applied to the work of Kyffin Williams and may allow for some interesting analysis.

[7] I. Berezhnoy, E. Postma, and Herik, "Automatic extraction of brushstroke orientation from paintings," vol. 20, no. 1, pp. 1–9, 2009. [Online]. Available: http://dx.doi.org/10.1007/s00138-007-0098-7

> Defines a method of analysing brushstrokes by applying a circular filter across a digital image to pick up the ridges of a brushstroke. This can then be used to pick out individual brushstrokes in order to be able to fit a nth order polynomial to them. Though this paper focuses on authenticating Van Gogh's paintings, it could easily be applied to the work of Kyffin Williams and may allow for some interesting analysis.

[8] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

> Used Python (http://opencv.willowgarage.com/documentation/python) and C++ (http://opencv.willowgarage.com/documentation) documentation for library reference and some learning on image processing/computer vision. Used since 11 October 2012.

[9] A. D. Brown, H. M. Dee, G. L. Roderick, and L. M. Hughes, "Can we date an artist's work from catalogue photographs?" Apr. 2013.

> The paper submitted to ISPA as a part of this project.

[10] J. F. Canny, "A computational approach to edge detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. PAMI-8, no. 6, pp. 679–698, Nov. 1986. [Online]. Available: http://dx.doi.org/10.1109/tpami.1986.4767851

> Reference for the Canny Edge detection technique.

[11] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, ser. CVPR '05, vol. 1. Washington, DC, USA: IEEE, June 2005, pp. 886–893 vol. 1. [Online]. Available: http://dx.doi.org/10.1109/cvpr.2005.177

> Describes a method of producing histograms of edge orientations which may prove to be a useful analysis technique for Kyffin Williams' art. The most interesting part of this paper is the use of segmentation and binning of gradients which seems like it could be useful to differentiate different parts of the image which may be painted in different styles.

[12] J. G. Daugman, "Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters," *Journal of the Optical Society of America A*, vol. 2, no. 7, pp. 1160+, July 1985. [Online]. Available: http://dx.doi.org/10.1364/josaa.2.001160

> Defines a method of creating a filter to match a large range of orientations over two dimensions. These filters (named Gabor filters) are commonly applied to image processing problems due to the range of orientations it can match (rather than a restricted range of orientations as most other filters offer).

[13] J. Davies, *100 Welsh heroes*. Culturenet Cymru, 2004. [Online]. Available: http://www.worldcat.org/oclc/753176001

Reference for Sir John Kyffin Williams

[14] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, ser. Addison-Wesley professional computing series. Addison-Wesley, 1996. [Online]. Available: http://www.worldcat.org/isbn/9780201633610 9780201633610.

Design Pattern reference for the architecture of the program.

[15] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, Nov. 2009. [Online]. Available: http://dx.doi.org/10.1145/1656274.1656278

Citation for the Weka data mining software. Weka is a Java based tool which can be used to run a lot of classifiers to a dataset, making it a very useful tool to apply to the Kyffin Williams project. Weka allows the application of complex machine learning techniques without having to spend a lot of time learning, understand and implementing said techniques.

[16] R. Harris, "How rolf learnt to paint like sir kyffin williams," BBC Broadcast, Feb. 2011. [Online]. Available: http://www.bbc.co.uk/programmes/p00f6nyt

A video on the BBC by Rolf Harris about some of Kyffin Williams' life and about his interesting style of painting.

[17] L. M. Hughes. (2011) Evaluating and measuring the value: use and impact of digital collections. [Online]. Available: http://www.worldcat.org/isbn/9781856047203

Having extensively used a digital collection as part of this project it is interesting to look back on other projects which have made use of digital collections to be able to compare and contrast this project use of digital collections to these other uses.

[18] I. T. Jolliffe. (2002) Principal component analysis. [Online]. Available: http://www.worldcat.org/isbn/9780387954424

Defines a mathematical procedure for converting a set of observations, possibly correlated, into a set of principal components. This is done such that the first principal component has the largest possible variance. Each succeeding component in turn has the next highest variance possible, under the constraint that it has to be uncorrelated to the proceeding component. When applied to this project it would allow the dimensions of a high dimensional state space to be put in some order of importance and might allow for the removal of less significant dimensions.

[19] E. Jones, T. Oliphant, P. Peterson, *et al.*, "SciPy: Open source scientific tools for python," 2001.

Reference for SciPy and NumPy; two useful libraries which implement complex scientific and mathematical functions. NumPy is also heavily tied to OpenCV version 2.4 so it is a necessity to include as part of this project.

[20] 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

> Defines a complex method for analysing individual brush strokes which has been used to classify the period of two paintings by Van Gogh. This technique could be very powerful when applied to Kyffin Williams' work. This could be one of the most important techniques for the whole of the Kyffin Williams project.

[21] "Glossary – python v2.7.4 documentation," Python Software Foundation, Apr. 2013. [Online]. Available: http://docs.python.org/2/glossary.html#term-duck-typing

> Documentation reference for Python's definition of "duck typing"; a useful programming style which allows flexibility without the need for strict object hierarchies.

[22] S. J. Russell, P. Norvig, and E. Davis, *Artificial intelligence : a modern approach*, 3rd ed. Prentice Hall, Dec. 2010. [Online]. Available: http://www.worldcat.org/isbn/0136042597 9780132071482.

> Textbook reference for many common artificial intelligence and machine learning techniques that were applied to this project.

[23] R. Trapp, "Gabor filters," Nov. 1998. [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/TRAPP1/filter.html

> Reference for Gabor filters, which included as nice image to help the visualisation of the filter.

[24] J. Unsworth, "Scholarly primitives: what methods do humanities researchers have in common, and how might our tools reflect this?" May 2000. [Online]. Available: http://people.lis.illinois.edu/~unsworth//Kings.5-00/primitives.html

> Defines the idea of a scholarly primitives ("a finite list of self-understood terms") in a digital humanities.

[25] K. Williams, *Across the straits : an autobiography*. Gomer, 1993. [Online]. Available: http://www.worldcat.org/isbn/0863839940 0863839940.

> Sir John Kyffin Williams' autobiography which is a very useful reference for periods of his life.

[26] G. Yang, "Gabor filter - file exchange - MATLAB central," Nov. 2010. [Online]. Available: http://www.mathworks.co.uk/matlabcentral/fileexchange/23253-gabor-filter/content/Gabor%20Filter/gabor_fn.m

> Defines an example algorithm for Gabor filters in MATLAB. This was converted to Python for the generation of Gabor filters to avoid mathematical errors.