Recoloring Images

Candidate Number: 1018098

May 6, 2017

1 Introduction

In this case study, we investigated a method that aims at completing the color information of an image where most of the pixels contain only greyscale information and very few pixels contain color information. The problem can be thought of in terms of an *image compression problem*, i.e. the memory on a computer required to store a given image shall be reduced, or, alternatively, it can be thought of in terms of an *image restoration problem*, i.e. for a given image, most of the color information was lost and shall be restored by mathematical and computational means. Both scenarios shall further be explored in the following two paragraphs.

Motivation: Image Compression We consider a situation in which one wants to send an image via the internet, e.g. via E-Mail. In order for this to work quickly, it is beneficial to compress the image before sending it, so that fewer information has to be transmitted and the time the process takes can be reduced. Within the signal processing community, many such approaches have been explored. A common idea is to try and find *clusters* within the image, such that one only has to store information about the clusters and which cluster each pixel belongs to. A common method that works in this manner is called *vector quantisation*, see [6]. Vector quantisation describes a general class of clustering methods that includes the famous *K-Means Algorithm*, see for instance [8]. A different idea for image compression would be to remove almost all of the color information before transmitting the image, and restoring it in some way when it arrives at the receiver. Our given problem can be viewed in this light.

Motivation: Image Restoration In the year 1452, the Italian painter Andrea Mantegna and possibly other painters created a large fresco in the Eremitani's Church in Padua [5]. The fresco was several hundred square meters large and constituted important testimonial of Italian renaissance art. Unfortunately, it was largely destroyed during World War II when allied planes dropped bombs on Padua. What remains today are fragments of the fresco that the surviving population collected after the bombing, as well as black and white photographs that were taken before the war. This leads to the question of how to restore the full fresco, given only some fragments and the black and white photographs. This is a very challenging problem as only around 7% of the fragments are still available and it is unknown where they belong [5]. Computational and mathematical methods to solve this problem have been developed recently, and this case study will introduce one such method. In Figure 1,

Figure 1: Shown is a computer based reconstruction of the fresco by Mantegna. The fragments contain color information while the rest of the picture relies on information from black and white photographs. Figure taken from [5].

a computer reconstruction of the fresco by Mantegna is shown which was created by Massimo Fornasier, see [5].

Setting the Scene: The Problem at Hand The main difference between the two scenarios described above is that in image compression, one can *chose* the pixels where one would like the color information to be, whereas in image restoration, this is impossible and one has to deal with whatever remains there are of the original image. In this case study, we will explore and discuss both scenarios. Assume there is a set of pixels Ω at which we know the corresponding greyscale information. Further assume there exists a subset of pixels $D \subset \Omega$ at which we know the full color information. The elements of D can be given (image restoration) or chosen (image compression). The problem we would like to address in this case study is how to approximate the full color information for *all* elements of Ω.

2 Theory

This section will start out by explaining how pictures are stored in MATLAB. Since this is the software that was used for all of our computations, the theory chapter will be focused around it. The section will then demonstrate how a color picture can be converted to a greyscale image. Once this is done, it will be possible to mathematically formulate the recolorisation problem and to reduce it to a linear system of equations.

2.1 Images

In MATLAB, images are stored in three dimensional arrays of size $m \times n \times 3$, where $m \times n$ is the number of pixels in the image, m is the number of rows and n is the number of columns. Indexing starts at the top left corner, which is defined to be the origin. In each of the three layers, the intensity of one of the three colors red, green and blue is stored (in this order). The intensity of each color is measured on a score from 0 to 255, corresponding to eight bits or one byte of memory. Define the set $I := \{j \in \mathbb{Z} | 1 \leq j \leq 255\}$. MATLAB refers to this kind of data as $uint8$. In this notation, the color white corresponds to (255, 255, 255), the color black corresponds to (0, 0, 0) and the color red corresponds to $c\,(1, 0, 0)$, where $c \in I$, $c > 0$ controls the intensity of red.

2.2 Greyscale Conversion

Let S be the MATLAB representation of an $m \times n$ image, so S is a three dimensional array. Let (r_j, g_j, b_j) denote the intensities of red, green and blue at pixel z_j $\forall j \in \{1, ..., N_p\}$ where $N_p := mn$ is the total number of pixels in the image. Define $\Omega := \{z_1, ..., z_{N_p}\}$ to be the set of all pixels in the image. In order to convert a color picture to a greyscale picture, one has to combine the three variables r_j, g_j and b_j into a single variable $\mathcal{G}_j$ that measures the intensity of grey at pixel z_j. Care has to be taken that $\mathcal{G}_j \in \{0, ..., 255\}\, \forall z_j \in \Omega$. There are multiple possibilities to do so, we decided to use

$$\mathcal{G}_j = 0.3 r_j + 0.59 g_j + 0.11 b_j\,. \tag{1}$$

We will assume in the following that $\mathcal{G}_j$ is rounded to the next integer value. At pixel $z_j \in \Omega$, we will now store the following triplet in image S:

$$(\mathcal{G}_j, \mathcal{G}_j, \mathcal{G}_j)\,. \tag{2}$$

This finalises the conversion of image S from color to greyscale.

2.3 Mathematical Formulation of the Problem

As above, let $\Omega := \{z_1, ..., z_{N_p}\}$ be the set of all pixels in a given image S. As outlined in the introduction, greyscale information is available at each pixel $z_j \in \Omega$. Consider a subset $D \subset \Omega$, $D := \{x_1, ..., x_{N_c}\}$ that contains all pixels at which color information is available. Here, N_c denotes the number of such pixels and it holds $N_c \ll N_p$.

Mapping Pixels to Colours The greyscale information can be represented by a mapping

$$\gamma : \Omega \to I\,, \tag{3}$$

$$z_i \mapsto \gamma(z_i) := \mathcal{G}_i\,, \tag{4}$$

which maps each pixel in the set Ω to a value in I representing a certain type of grey. In a similar fashion, we can represent the color information by a mapping

$$f : D \to I^3\,, \tag{5}$$

$$x_j \mapsto \big(f^r(x_j), f^g(x_j), f^b(x_j)\big) := (r_j, g_j, b_j)\,, \tag{6}$$

where each $f^s : D \to I, s \in \{r, g, b\}$ maps a pixel in the smaller set D to the intensity of one of the three basic colors red, green or blue.

Basic Challenge We seek to find the unknown color information at the remaining pixels in the larger set Ω. Mathematically speaking, we seek a mapping of the from

$$F : \Omega \to I^3\,, \tag{7}$$

$$z_i \mapsto \big(F^r(z_i), F^g(z_i), F^b(z_i)\big) := (r_j, g_j, b_j)\,, \tag{8}$$

where each $F^s : \Omega \to I, s \in \{r, g, b\}$ maps a pixel in the larger set Ω to the intensity of one of the three basic colors red, green or blue. This relies upon the assumption that the three color mappings F^r, F^g and F^b are independent of each other. The basic challenge is to find a mapping F, such that for all pixels in the smaller set D, the mapping agrees approximately with the mapping f, $F|_D \approx f|_D$. Yet, the mapping F should not simply replicate the mapping f but rather *generalise* the information it learned from f on D to unseen data in ω.

Representing the Unknown Mapping F We have to settle on a form for F. We will represent F as the sum

$$F^s(x) = \sum_{j=1}^{N_c} K(x, x_j)\, a_j^s\,, \tag{9}$$

where $s \in \{r, g, b\}$. The coefficients a_j^s, $j \in \{1, ..., N_c\}, s \in \{r, g, b\}$ are unknown at this stage and will have to be determined. K is a *kernel* that will be specified further below.

Finding the coefficients a_j^s Define the vector $a^s \in \mathbb{R}^{N_c}$, $a^s = (a_1^s, ..., a_{N_c}^s)$. We can then find the coefficients by solving a minimisation problem:

$$a^s := \arg\min_{a^s} \frac{1}{N_c} \sum_{i=1}^{N_c} \left[F^s(x_i) - f^s(x_i)\right]^2 + \delta \, ||F^s||^2_{\mathcal{H}(\Omega)} \,, \tag{10}$$

where $||\cdot||_{\mathcal{H}(\Omega)}$ denotes a norm in a *reproducing kernel hilbert space*, δ is a *regularisation parameter* and the a^s dependence is hidden in $F^s(x_i)$. We define

$$||F^s||^2_{\mathcal{H}(\Omega)} := \sum_{i,j=1}^{N_c} K(x_i, x_j) a_i^s a_j^s \,. \tag{11}$$

The first term in Equation (10) can be seen as a sum of squared errors, which is a special instance of a *loss function*, a common concept in parameter inference. It quantifies how well our mapping F can approximate the original mapping f on all points in the smaller set D. In the machine learning literature, the set D is often referred to as a *training set*. The second term in Equation (10) is a regularisation term that will ensure that the resulting linear system is well conditioned trough convexification of the optimisation problem, see Subsection 2.4. Alternatively, one can understand the regularisation term as a way to avoid *overfitting* of the model by penalising numerous, large parameters a_j^s. In this context, F is our model that we are fitting to the data. The model has $3N_c$ parameters, which in the case of a typical image of size 600×600 pixels will amount to approximately $3 \cdot 300 = 900$ parameters, if we are allowed to chose N_c such that the method works well (image compression setting). This is a comparatively large number of parameters and therefore, it makes intuitively sense to regularise the optimisation problem. Before discussing how one can solve the optimisation problem, we will explore the kernel $K(x, y)$.

The Kernel $K(x, y)$ We define

$$K(x, y) := \phi\left(\frac{||x - y||_2}{\sigma_1}\right) \phi\left(\frac{|\gamma(x) - \gamma(y)|^p}{\sigma_2}\right) \,, \tag{12}$$

where (σ_1, σ_2, p) are model *hyperparameters* which depend on the type of image that is to be recoloured. See Subsection 2.5 for an explanation of the term "hyperparameters". Intuitively, σ_1 and σ_2 are weights for the *euclidian distance* between x

and y and for the greyscale difference between x and y, respectively. The practical significance of the parameter p will be investigated in Subsection 3.3. The function $\phi : \mathbb{R}^+ \to \mathbb{R}^+$ is a *radial basis function*. In this case study, we investigated two different choices of radial basis function:

$$\text{gaussian:} \qquad \phi(r) := e^{-r^2}\,, \tag{13}$$

$$\text{compactly supported (CS):} \qquad \phi(r) := (1-r)_+^4\,(4r+1)\,. \tag{14}$$

For the Gaussian basis function, it holds $\phi \in C^\infty$ whereas for the CS basis function, it holds $\phi \in C^2$. We have employed here the notation

$$(1-r)_+ := \max(1-r, 0)\,. \tag{15}$$

The support for the second alternative is given by the compact set $[0, 1]$. Both choices of radial basis function will work in principle, however, the recoloring procedure might work better on certain types of images depending on the choice of radial basis function. This will further be investigated in Subsection 3.2. Note that for either choice of basis function, the kernel $K$is symmetric in its arguments, $K(x, y) = K(y, x)$.

2.4 Reduction to a Linear System

Given the form of the kernel, we will now reduce the minimisation problem in Equation (10) to a linear system. Define the *objective function*

$$J(a^s) := \frac{1}{N_c}\sum_{i=1}^{N_c}\left[F^s(x_i) - f^s(x_i)\right]^2 + \delta\,||F^s||^2_{\mathcal{H}(\Omega)}\,, \tag{16}$$

$$= \frac{1}{N_c}\sum_{i=1}^{N_c}\left[\sum_{j=1}^{N_c} K(x_i, x_j)\,a_j^s - f^s(x_i)\right]^2 + \delta\sum_{i,j=1}^{N_c} K(x_i, x_j) a_i^s a_j^s\,. \tag{17}$$

Our aim is to find the global minimum of $J(a^s)$. We will search for *critical points* of the objective function, that is, points where $\frac{\partial J}{\partial a_k^s} = 0\,\forall k \in \{1, ..., N_c\}$. Since our objective function J is *convex*, every critical point will automatically be a local (and global) minimum of J. The function J is convex because it is a composition of convex

functions, see Subsection A.1 in the appendix for a proof. We evaluate

$$\begin{aligned}\frac{\partial J}{\partial a_k^s} &= \frac{2}{N_c}\sum_{i=1}^{N_c}\left\{\left[\sum_{j=1}^{N_c} K(x_i,x_j)\,a_j^s - f^s(x_i)\right]K(x_i,x_k)\right\} + \delta\sum_{i,j=1}^{N_c} K(x_i,x_j)\frac{\partial}{\partial a_k^s}(a_i^s a_j^s)\\ &= \frac{2}{N_c}\sum_{i,j=1}^{N_c} K(x_i,x_j)\,K(x_i,x_k)\,a_j^s - \frac{2}{N_c}\sum_{i=1}^{N_c} f^s(x_i)K(x_i,x_k)\\ &\quad + 2a_k^s\delta K(x_k,x_k) + 2\delta\sum_{i=1,\,i\neq j}^{N_c} K(x_i,x_k)a_i^s \overset{!}{=} 0\end{aligned}$$

$\forall k \in \{1,...,N_c\}$, which is equivalent to

$$\begin{aligned}0 &= \sum_{i,j=1}^{N_c} K(x_i,x_j)\,K(x_i,x_k)\,a_j^s - \sum_{i=1}^{N_c} f^s(x_i)K(x_i,x_k) + N_c\,\delta\sum_{i=1}^{N_c} K(x_i,x_k)a_i^s\\ &= \sum_{i=1}^{N_c} K(x_i,x_k)F^s(x_i) - \sum_{i=1}^{N_c} f^s(x_i)K(x_i,x_k) + N_c\,\delta\sum_{i=1}^{N_c} K(x_i,x_k)a_i^s\,.\end{aligned}$$

We will now define a matrix $K_D \in \mathbb{R}^{N_c\times N_c}$ with elements $K_{D_{i,j}} := K(x_i,x_j)$. Since the kernel K is symmetric in its arguments, the matrix K_D is symmetric. With this definition, we can write

$$0 = (K_D\cdot F^s)_k - (K_D\cdot f^s)_k + N_c\delta\,(K_D\cdot a^s)_k\,, \tag{18}$$

where we have used the fact that K_D is symmetric and introduced the vectors $F^s := (F^s(x_1),...,F^s(x_{N_c}))^\intercal \in \mathbb{R}^{N_c}$ and $f^s := (f^s(x_1),...,f^s(x_{N_c}))^\intercal \in \mathbb{R}^{N_c}$. Since Equation (18) must hold for all $k \in \{1,...,N_c\}$, we may conclude

$$K_D\,(F^s - f^s + \delta N_c a^s) = 0\,. \tag{19}$$

From the definition of the vector F^s, it is obvious that $F^s = K_D\,a^s$. If we use this relation and assume that K_D is invertible, i.e. a unique solution (a unique global minimum) exists, then we can rewrite Equation (19) to give

$$(K_D + \delta N_c I)\,a^s = f^s\,. \tag{20}$$

This equation now has the form $Ax = b$. It therefore represents a linear system that we can solve using standard techniques from (numerical) linear algebra. Note that the matrix $A := K_D + \delta N_c I \in \mathbb{R}^{N_c\times N_c}$ is symmetric, so one possibility would be to use the iterative MINRES algorithm for symmetric matrices that was developed in the 1970s by Paige and Saunders, see [10].

2.5 Summary: The Recolorisation Process

Based on the theory developed above, we can now outline the main steps in solving the image recoloration problem. First, we imported an image S into MATLAB. Second, the image was converted to greyscale except for few pixels which were set at random (image restoration setting). Third, the matrix K_D and the vectors f^s were created for $s \in \{r, g, b\}$. Fourth, the linear system from Equation (20) was set up and solved using build in MATLAB methods, separately for each color. Fifth and last, the image was recoloured using the now specified mapping F.

Recolouring the Image Solving the problem given by Equation (20) yields the coefficients a_j^s which uniquely determine the mappings F^s for all three basic colors. For each pixel $z_i \in \Omega$, the color s can then be found through evaluating

$$F^s(z_i) = \sum_{j=1}^{N_c} K(z_i, x_j)\, a_j^s \,. \tag{21}$$

We can write this in matrix form by defining $K_\Omega \in \mathbb{R}^{N_p \times N_c}$ with elements $K_{\Omega_{i,j}} := K(z_i, x_j)$. Note that K_Ω is not quadratic, it has in fact many more rows than columns. By overloading our previous definition of F^s, we now define a larger vector $F^s := \big(F^s(z_1), ..., F^s(z_{N_p})\big)^\intercal \in \mathbb{R}^{N_p}$ that stores all of the color information that was recovered for the image S. This enables us to write

$$F^s = K_\Omega\, a^s \,, \tag{22}$$

which computes the approximated color information at all pixels in the larger set Ω and stores it in the vector F^s.

Parameters and Hyperparameters Throughout the remainder of this work, the coefficients a_j^s will be referred to as the *parameters* of the model, whereas the quantities δ, p, σ_1 and σ_2 will be referred to as the *hyperparemeters* of the model. This distinction is commonly made in the machine learning setting when one wants to distinguish parameters that act on very different hierarchical levels of the model, see [9] for more explanations on this distinction.

Intuitive Explanation Intuitively, the algorithm uses the given color information at the pixels where it is available together with the greyscale information at all other pixels to interpolate the missing color information. This interpolation can be adjusted

to be more sensitive to the euclidean distance or to the greyscale difference between points. The interpolation can be performed using two different sets of radial basis functions. The interpolation is not done exactly in a sense that we do not require our recoloured image to have the same color information as the original image on the N_c given color pixels in D. Rather, we think of the situation as a case of data fitting where we have an overdetermined problem and we aim to chose our model parameters a_j^s such that the best agreement between model and data is achieved at all given color pixels in D.

3 Results

For the computations, a *graphical user interface (GUI)* was written in MATLAB that allowed a comfortable usage of the recoloring code. The GUI is shown in Figure 2. Most images in this section were taken from the University of South California Signal and Image Processing Institute database (USC-SIPI database), see [2], and from the NASA Image and Video Library, see [1]. In all of the following results, the pixels used for approximately restoring the original color information were chosen at random.

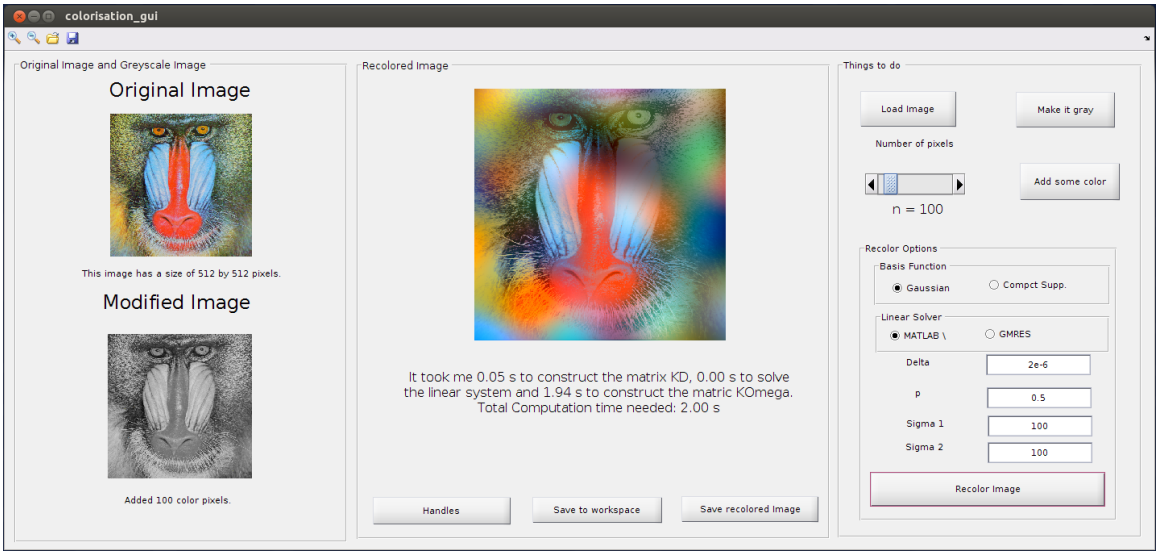


Figure 2: GUI written in Matlab. The GUI can load images from a specified location and save the recoloured images. It can further convert an image to greyscale, add in color pixels and recolour the image using the procedure outlined in Subsection 2.5. Both the basis functions as well as the hyperparameters δ, p, σ_1 and σ_2 can be chosen by the user. The image displayed in the GUI was taken from [2].

3.1 Exemplary Images

To start this section, some exemplary images shall be presented that demonstrate the functionality of the code.

Dependency on the Number N_c of Available Colour Pixels In Figure 3, the dependency of the recoloured image on the number N_c of color pixels in D is investigated. Unsurprisingly, the recoloured image becomes more and more similar to the original as the number of color pixels that the algorithm may use to approximate the full color information is increased. For the picture in Figure 3 which contains $N_p = 256 \times 256 = 65,536$ pixels, one needs approximately $N_c = 100$ pixels to obtain a recoloured image which is very similar to the original. In order to measure this similarity with the original image, it would be preferable to compute an error of the recoloured image, using for example a point wise sum of squares of the difference between the original color value and the restored color value for each color at each pixel. This was not done here, but will further be discussed from an optimisation point of view in Section 5.

Computational Time and Image Compression Another result relating to our algorithm is that the computational time it takes to restore approximate colour information increases with the number N_c of color pixels. Most of the time is consumed by the task of constructing the large matrix K_Ω that is used to construct the vector F_s of restored color information. As N_c increases, the number of parameters a_j^s increases and therewith the time it takes to construct K_Ω. It is surprising that with only 100 out of 65,536 color pixels, corresponding to approximately 0.15% of the original color information, it is possible to restore the vast majority of the original color information. Of course, the method presented here also takes into account the greyscale information at all 65,536 pixels. Based on this observation, it is interesting to ask whether this method can be used for image compression. This question will further be investigated in Section 4.

Performance on Large Images The image presented in the previous Figure 3 was a comparatively small image containing only 256×256 pixels. It is interesting to explore the question whether the method presented here also works on larger images. This is investigated in Figure 7 in Appendix A, where a 769×1024 pixel image showing a waterfall is recoloured using $N_c = 766$ color pixels which corresponds to approximately 0.1% of the original color information. The algorithm performed very

(a) Original picture, taken from [2].

(b) Recolored, using $N_c = 20$. $T \approx 0.09\,\text{s}$.

(c) Recolored, using $N_c = 30$. $T \approx 0.13\,\text{s}$.

(d) Recolored, using $N_c = 100$. $T \approx 0.41\,\text{s}$.

Figure 3: Recolouring a 256×256 pixel image taken from [2]. As before, N_c denotes the number of color pixels used. We define T to be the computational time consumed by the algorithm to complete the task of approximating the original color information. In all three recoloured images, Gaussian basis functions were used. Hyperparameters were held constant at $\delta = 2 \cdot 10^{-6}, p = 0.5, \sigma_1 = 100$ and $\sigma_2 = 100$. As N_c increases, the recoloured image becomes more and more similar to the original.

well, producing a recoloured image that is very similar to the original image. The computational time it took the algorithm to do so increased greatly to approximately 100 s. Using MATLAB's backslash method to solve the resulting linear system proved to be extremely efficient, even on this very large image it took less than 0.3 s to solve the linear system. The vast majority of the time was again consumed by constructing K_Ω.

3.2 Investigation of the Effect of the Basis Functions

In a further step, we investigated the effect of using either Gaussian basis functions or CS basis functions when constructing the kernel $K(x, y)$. Two different experiments were undertaken, investigating the influence of the basis functions in one setting where many color pixels were available (see Figure 4) and in another setting where only very few color pixels were available (see Figure 8 in Appendix A).

Using a Large Number N_c of Colour Pixels In Figure 4, enough color pixels are available for both variants of the method to effectively restore the original color information (approximately 0.18% of the original colour information). Slight differences remain, but is impossible to call one choice of basis functions better than the other based on this figure. There is, however, a difference in terms of computational time the two algorithms take: while it takes $T \approx 0.5$ s to construct the recoloured image using Gaussian basis functions, it takes $T \approx 1.3$ s to complete the same task using CS basis functions. This is most likely due to an inefficient implementation of the CS basis functions as the algorithm is calculating lots of zeros outside the support of ϕ.

Using a Small Number N_c of Colour Pixels In Figure 8 in Appendix A, only very few color pixels are available (approximately 0.0038 % of the original colour information). None of the variants of our method is able to produce an output that is similar in color to the original image. However, the two choices of basis functions do produce different outputs in this setting. It is debatable which one is the superior result. In order to answer this question, one would need a measure for the error the two variants of our method make, see Subsection 3.1.

Adjusting the Hyperparameters Note that in Figures 4 and 8, the hyperparameter values were adjusted to produce the "best possible" results, judging solely by eye

and without a formal mathematical optimisation procedure. Incorporating such an optimisation on the hyperparameters δ, p, σ_1 and σ_2 is outside the scope of this work, however, it is briefly discussed in Section 5. In general, it appears that the CS basis functions require greater values for the two hyperparameters σ_1 and σ_2.

(a) Original picture from [2]. (b) Gaussian basis functions. (c) CS basis functions.

Figure 4: Recolouring a 256×256 pixel image taken from [2], using $N_c = 120$ color pixels, demonstrating the two different choices of basis functions. Both produce similar results. The hyperparameters expect for $\delta = 2 \cdot 10^{-6}$ were slightly adjusted between the two cases: for the Gaussian basis functions, we used $p = 0.5, \sigma_1 = 100$ and $\sigma_2 = 100$, as before. However, for the CS basis functions, we used $p = 1, \sigma_1 = 400$ and $\sigma_2 = 400$.

3.3 Investigation of the Effect of the Hyperparameters

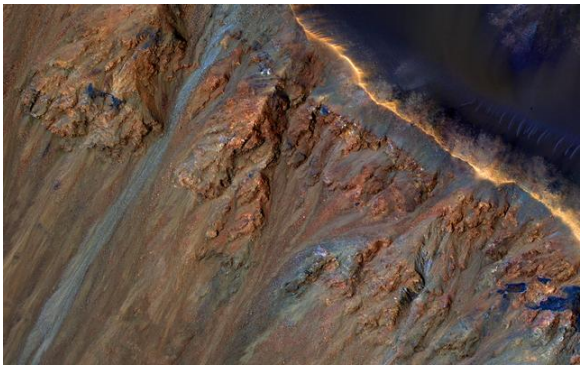
In order to determine what effects the four hyperparameters have on the recoloured image, we varied them one by one, keeping the other hyperparameters constant. We decided to chose the number N_c of color pixels in each case in a manner such that the observed change was the greatest.

Investigating the Effect of δ Recall that the hyperparameter δ is a regularisation parameter that can be though of in two ways, either as a means of preventing the matrix K_D from becoming ill conditioned or as a way to control overfitting of the model. Intuitively, as δ decreases, the minimum of the regularised objective function moves towards the minimum of the unregularised objective function which could make the system harder to solve numerically. However, at the same time, the resulting image should match the original color information better and better because the actual minimum of the loss function J (without regularisation) is attained asymptotically. This

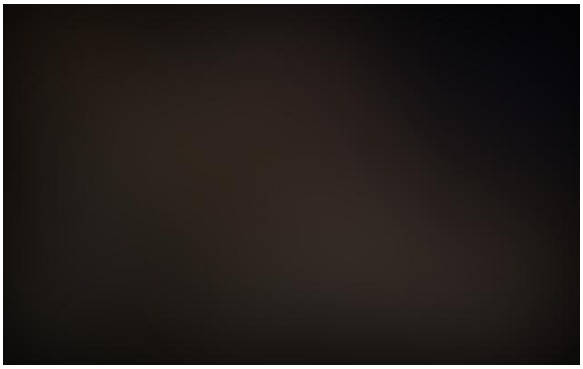
effect is demonstrated in Figure 5 in this section for a larger image and in Figure 9 in Appendix A. It was found that the optimum value of δ depends on the resolution of the image and should be smaller for greater images. In the two cases investigated here, the optimum value for δ ranged from 10^{-8} to 10^{-6}.

Investigating the Effect of p In contrast to the hyperparameter δ which controlled the conditioning of the linear system, it is more difficult to formulate an expectation for what the hyperparameter p controls, or which value should be picked. In order to analyse the significance of p, two very different images were recoloured, keeping all hyperparameters except for p constant. One picture was a landscape in Tuscany with smooth color gradients while the other one was an artwork by Pablo Picasso which showed very rapid changes in color. Each image was recoloured using two values of p: one value that was found to be optimal for the image and one that was found to be optimal for the other image, respectively. The results are shown in Figure 10 in Appendix A. It appears that for images such as artworks that show abrupt changes in color, a larger value for p should be used than for images such as photographs of landscapes, where colors gradually fade into one another. In both cases, the optimal value for p found in the experiment was closed to 1.

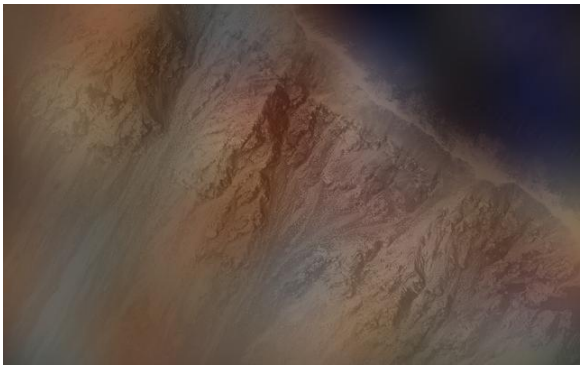
Investigating the Effect of σ_1 and σ_2 The two hyperparameters σ_1 and σ_2 appear in the definition of the kernel K and control how much relative weight is given to the euclidean distance between two points and the greyscale difference between two points, respectively. The default setting in the MATLAB GUI which we programmed is $\sigma_1 = \sigma_2 = 100$ for the Gaussian basis functions and $\sigma_1 = \sigma_2 = 400$ for the CS basis functions. Note that not only the ratio $\frac{\sigma_1}{\sigma_2}$ matters for the recolouring process, but also the actual magnitude of the two hyperparameters, as can be seen from the definition of the kernel K, see Equation (12). Intuitively, as both σ_1 and σ_2 are varied, the influence of pixels further away from the current pixel on the interpolated color is varied. The default setting usually performs well on most images, however, as shown in Figure 6, some images require different settings. The image used in Figure 6 is mostly white with a colourful object in the centre. The greyscale information outside this object is almost uniform. Therefore, it occurs that it is advantageous to increase the weight of the euclidean distance between pixels, which in this context is more meaningful than the difference in greyscale.



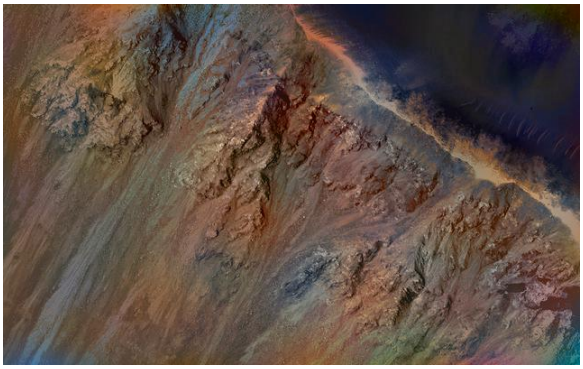
(a) Original picture.



(b) Recoloured, using $\delta = 2 \cdot 10^{-1}$. $T \approx 7.6\,\mathrm{s}$.



(c) Recolored, using $\delta = 2 \cdot 10^{-4}$. $T \approx 7.2\,\mathrm{s}$.



(d) Recolored, using $\delta = 2 \cdot 10^{-8}$. $T \approx 7.0\,\mathrm{s}$.

Figure 5: Recolouring a 400 $\times$ 640 pixel image from `https://images.nasa.gov/#/details-PIA21605.html`, showing "Colorful Equatorial Gullies in Krupac Crater" [1], investigating the effect of the regularisation parameter δ. We used $N_c = 400$ color pixels with Gaussian basis functions. The hyperparameters except for δ were held constant at $p = 0.5, \sigma_1 = 100$ and $\sigma_2 = 100$. The computational time needed was decreased slightly with decreasing δ from $T \approx 7.6\,\mathrm{s}$ for $\delta = 2 \cdot 10^{-1}$ to $T \approx 7.0\,\mathrm{s}$ for $\delta = 2 \cdot 10^{-8}$. As the regularisation parameter decreases, the recoloured images become more and more similar to the original image. Values of δ smaller that $2 \cdot 10^{-8}$ were tested, but no significant change was observed.

(a) Original picture, taken from [2].

(b) Recoloured, using $\sigma_1 = 100, \sigma_2 = 100$.

(c) Recolored, using $\sigma_1 = 200, \sigma_2 = 100$.

(d) Recolored, using $\sigma_1 = 100, \sigma_2 = 200$.

Figure 6: Recolouring a 512 $\times$ 512 pixel image taken from [2]. We used $N_c = 100$ color pixels width Gaussian basis functions. The hyperparameters except for σ_1 and σ_2 were held constant at $\delta = 2 \cdot 10^{-6}$ and $p = 0.5$. The best result in this case is achieved in c), where we chose $\sigma_1 = 200$ and $\sigma_2 = 100$.

4 Extension: Image Compression

The previous section demonstrated how our algorithm can be used to recolour an image where only very little color information is available. As mentioned in the introduction, this is interesting from an image compression point of view as it means that the information needed to store a given image is reduced by approximately a factor three. However, standard compression formats such as JPG in most situations achieve a compression by factor 10 without notable less in image quality [7]. Therefore, the idea developed to combine our algorithm with a basic image compression method in order to achieve a similar compression rate as JPG does.

4.1 The Compression Method

The new method consists of the following steps:

1. compress the image by a factor 4,
2. remove almost all color information,
3. approximate missing color information,
4. decompress the image.

Steps 1 and 2 of the procedure form the *compression* part, steps 3 and 4 form the *decompression* part. If one wanted to send an image over a bandwidth limited connection, one would first compress it, then send it, and decompress it once it has been received. The total compression rate c of this method is $c = \frac{1}{4} \cdot \frac{1}{3} = \frac{1}{12}$. The image will be compressed to a similar extent as JPG compresses in many situations. Steps 1 and 4 of the algorithm will be explained in the following paragraphs. Steps 2 and 3 remained as they were before.

Step 1: Compression A basic compression method was implemented which averages the color information of 4 neighbouring pixels and saves this averaged color information. In other words, for every 4 original pixels, only one pixel was kept, which was assigned the average color of the 4 original pixels. Averaging was carried out separately for each color and the results were rounded to the next integer to ensure MATLAB can interpret the color information, see Subsection 2.1.

Step 4: Decompression An image of the original size was obtained by using two dimensional function interpolation. In between every two pixels in the compressed image in each dimension, a new pixel was inserted which was given linearly interpolated color information, separately for red, green and blue. This yielded an image which contained color information for the original number of pixels.

4.2 Results

The new routine was tried on various images. As a first result, it was observed that compressing the image before carrying out the recolouring process process substantially decreased the computational time needed. Using the 769×1014 pixel image from Figure 7 showing a waterfall, the total time needed to carry out all 4 steps of the new methods was $T \approx 9\,\mathrm{s} + 10\,\mathrm{s} = 19\,\mathrm{s}$, where $9\,\mathrm{s}$ were needed to compress the image and $10\,\mathrm{s}$ were needed to decompress and recolour the image. Recall that previously, $98\,\mathrm{s}$ were needed to recolour this image, using the same number N_c of color pixels. This reduction in complexity by a factor 5 is intuitively understandable, as most of the time was previously needed to construct the large matrix K_Ω. The size of this matrix (rows times columns) has now been reduced by a factor 4 because the image is first recoloured and then decompressed. The decompressed waterfall image and the original are shown in Figure 11. The decompressed image has a lower quality than the original image, notably because it is more "pixely" and less sharp. This can also be observed in Figure 12 in Appendix A.

5 Summary

In this case study, we implemented a method that allows one to approximate the full color information of an image where only around 0.1% of the pixels contain original color information. The method can be useful from two perspectives: first, it can be used to recolour images where most of the original color information was lost, and second, it can be employed as part of an image compression algorithm. We showed that using a combination of a basic image compression technique in conjunction with our method for recolouring allowed us to compress an image to $\frac{1}{12}$ of its original size. Many extensions to the methods presented here are possible, two of which shall be discussed in the following paragraphs.

Optimising the Hyperparameters So far, the hyperparameters δ, p, σ_1 and σ_2 were either set to some default values that seemed to work well for most images, or it was attempted to pick some "optimal" values by a trial and error strategy. It would of course be much better if a formal optimisation was to be carried out which would at least find a local optimum for the four hyperparameters. However, this is difficult as every single function evaluation requires the whole recolorisation process to be carried out. Even for small images, this is a computationally very expensive task. Possible solutions to this problem include the usage of derivative free methods (see e.g. [4]) or the usage of approximate Bayesian methods (see e.g. [11]).

Clustering Pixels In the extension, a basic image compression method was introduced that simply averages over 4 adjacent pixels. The method does not take into account whether those pixels are actually similar to each other in their color values. It would be much better to first use an algorithm which finds *color clusters* of varying size. Given the clusters, one could average over their color information and just keep one pixel per cluster. This is likely to yield a better result as the averaging would be carried out over pixels that are more similar to each other in terms of their color, therefore, less information is lost. An extended method which would optimise the hyperparameters and cluster pixels before compression is likely to yield a superior image quality after decompression.

References

[1] NASA image and video library. `https://images.nasa.gov/#/`. Accessed: 2017-05-04.

[2] The USC-SIPI image database. `http://sipi.usc.edu/database/database.php`. Accessed: 2017-05-03.

[3] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[4] A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to derivative-free optimization*. SIAM, 2009.

[5] M. Fornasier. Nonlinear projection recovery in digital inpainting for color image restoration. *Journal of Mathematical Imaging and Vision*, 24(3):359–373, 2006.

[6] R. Gray. Vector quantization. *IEEE Assp Magazine*, 1(2):4–29, 1984.

[7] R. F Haines and S. L. Chuang. The effects of video compression on acceptability of images for monitoring life sciences experiments. *National Aeronautics and Space Administration*, 1992.

[8] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967.

[9] K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[10] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM journal on numerical analysis*, 12(4):617–629, 1975.

[11] T. Toni, D. Welch, N. Strelkowa, A. Ipsen, and M. P. H. Stumpf. Approximate bayesian computation scheme for parameter inference and model selection in dynamical systems. *Journal of the Royal Society Interface*, 6(31):187–202, 2009.

[12] S. Vajda. *Theory of linear and non-linear programming*. Longmans, 1974.

[13] Z. Wang, A. C. Bovik, and B. L. Evan. Blind measurement of blocking artifacts in images. In *Image Processing, 2000. Proceedings. 2000 International Conference on*, volume 3, pages 981–984. IEEE, 2000.

A Appendix

A.1 Convexity of the Objective J

The objective function J had the form

$$J(a^s) = \frac{1}{N_c} \sum_{i=1}^{N_c} \left[F^s(x_i) - f^s(x_i)\right]^2 + \delta \, ||F^s||^2_{\mathcal{H}(\Omega)} \, . \tag{23}$$

We would like to show that J is a *convex* function of the variables a^s_j. A function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if it holds

$$f\left(\lambda x + (1-\lambda)y\right) \leq \lambda f(x) + (1-\lambda) f(y) \, , \tag{24}$$

for all $x, y \in \mathbb{R}^n$ and for all $\lambda \in [0, 1]$. If the inequality is replaced with a strict inequality, then this defines a *strictly convex* function. Consider the definition of the mapping $F^s(x)$ for fixed x as a function of the coefficients a^s_j,

$$F^s(a^s_1, ..., a^s_{N_c}|x) = \sum_{j=1}^{N_c} K(x, x_j) \, a^s_j \, . \tag{25}$$

As this is linear in $a^s_j \; \forall j \in \{1, ..., N_c\}$, we conclude that the mapping F^s is a convex function of the variables a^s_j since linear functions are convex (but not strictly convex), see [12]. Furthermore, the sum over the squares of convex functions (minus an offset) is convex, because the function $f(x) = x^2$ is (strictly) convex and convexity is invariant under affine transformations (e.g. linear transformations), see e.g. [3]. We conclude that the first term in Equation (23) is convex. For the second term in Equation (23), we simply use the fact that every norm on $\mathbb{R}^n$ is convex, see [3]. In conclusion, $J : \mathbb{R}^{N_c} \to \mathbb{R}$ is convex as a function of the coefficients a^s_j.

A.2 Performance on Larger Images

(a) Original picture. (b) Recolored picture.

Figure 7: Recolouring a 769 × 1024 pixel image of a waterfall that was taken from `http://www.wallpaperama.com/wallpapers/peaceful-water-fall.html`, using $N_c = 766$ color pixels and Gaussian basis functions. This demonstrates that the algorithm also works for larger images. Hyperparameters were $\delta = 2 \cdot 10^{-6}, p = 0.5, \sigma_1 = 100$ and $\sigma_2 = 100$, as before. The computational time needed was $T \approx 98\,\mathrm{s}$, out of which the vast majority (approximately 97 s) were used to construct the matrix K_Ω.

A.3 Investigating the Effect of the Basis Functions

(a) Original image. (b) Gaussian basis functions. (c) CS basis functions.

Figure 8: Recolouring a 512×512 pixel image showing peppers of various shapes and colors from `http://www.imageprocessingplace.com/root_files_V3/image_databases.htm`, using only $N_c = 10$ color pixels. The aim of this figure is to demonstrate how the two choices of basis functions cope with very few color pixels. The hyperparameters expect for $\delta = 2 \cdot 10^{-6}$ were adjusted between the two cases, to produce the best possible result with either choice of basis function, as far as this was possible without a formal mathematical optimisation. For the Gaussian basis functions, we used $p = 1, \sigma_1 = 500$ and $\sigma_2 = 400$. For the CS basis functions, we used $p = 1.05, \sigma_1 = 1000, \sigma_2 = 1200$. The computational time needed was $T \approx 0.18\,\mathrm{s}$ for Gaussian basis functions and $T \approx 0.65\,\mathrm{s}$ for the CS basis functions.

A.4 Investigating the Effect of δ

(a) Original picture, taken from [2].

(b) Recoloured, using $\delta = 2 \cdot 10^{-1}$. $T \approx 0.27\,\text{s}$.

(c) Recolored, using $\delta = 2 \cdot 10^{-3}$. $T \approx 0.24\,\text{s}$.

(d) Recolored, using $\delta = 2 \cdot 10^{-6}$. $T \approx 0.25\,\text{s}$.

Figure 9: Recolouring a 256×256 pixel image taken from [2], investigating the effect of the regularisation parameter δ. We used $N_c = 60$ color pixels and Gaussian basis functions. The hyperparameters except for δ were held constant at $p = 0.5, \sigma_1 = 100$ and $\sigma_2 = 100$. The computational time needed was approximately the same in all three cases with $T \approx 0.25\,\text{s}$. As the regularisation parameter decreases, the recoloured images become more and more similar to the original image. Values of δ smaller that $2 \cdot 10^{-6}$ were tested, but no significant change was observed.

A.5 Investigating the Effect of p

(a) Landscape photography by Norbert Nagel, $p = 0.9$.

(b) Landscape photography by Norbert Nagel, $p = 1.1$.

(c) Artwork by Pablo Picasso, $p = 1.1$.

(d) Artwork by Pablo Picasso, $p = 0.9$.

Figure 10: Investigating the effect of the hyperparameters. The upper picture, taken by Norbert Nagel and published on Wikimedia Commons under the license CC BY-SA 3.0, shows a landscape in Tuscany. There are no abrupt changes in color but rather smooth color gradients. The lower picture, an artwork by Pablo Picasso from 1937 titled "Weeping Woman", shows sharp contrasts and no smooth color gradients. Gaussian basis functions were used and $N_c = 100$ color pixels were employed for recolouring. Hyperparameter values apart from p were held constant at $\delta = 2 \cdot 10^{-6}, \sigma_1 = 100$ and $\sigma_2 = 100$. On the left, optimal choices for p are presented for both images. On the right, the p values are interchanged between the two images. In both cases, this alternative choice for p is worse than the optimal choice on the left.

A.6 Extension: Image Compression

(a) Original picture. (b) Recolored and decompressed picture.

Figure 11: Testing recolouring and decompression using the 769 × 1024 pixel image of a waterfall from `http://www.wallpaperama.com/wallpapers/peaceful-water-fall.html`, using $N_c = 766$ color pixels and Gaussian basis functions. Hyperparameters were $\delta = 2 \cdot 10^{-6}, p = 0.5, \sigma_1 = 100$ and $\sigma_2 = 100$, as before. The total computational time needed was $T \approx 19\,\mathrm{s}$. A difference in the image quality is notable, especially when looking at both images in full size. The image on the right is recovered from an image which requires 12 times less storage than the image on the left.

(a) Original picture, taken from [2]. (b) Recolored and decompressed picture.

Figure 12: Testing recolouring and decompression using a 512×512 pixel image taken from [2], using $N_c = 200$ color pixels and Gaussian basis functions. Hyperparameters were $\delta = 2 \cdot 10^{-6}, p = 0.5, \sigma_1 = 100$ and $\sigma_2 = 100$. This image, often referred to in the literature as "Lena", is one of the most frequently used images in the field of signal processing, see e.g. [13].