Final Year Project

Recolouring Problematic Images for Readers with Colour Blindness

Abdo Moustafa

Student ID: 19495386

A thesis submitted in part fulfilment of the degree of

BSc. (Hons.) in Computer Science

Supervisor: Dr. Colm Ryan

UCD School of Computer Science

University College Dublin

April 25, 2024

Table of Contents

Abstract

In the realm of data visualization, colour selection serves as a critical tool for illustrating quantitative and qualitative information rather than for just mere aesthetics [1]. However, poor colour choices in data visualization can distort data interpretation, particularly for individuals with colour vision deficiencies. This distortion creates a barrier to effective communication and decision-making due to the increased difficulty in interpretation. [2]

This project couples machine learning techniques with computer vision to address the challenge of recolouring problematic plots to make them colourblind-friendly. With the use of object detection to identify the plot's legend and its colour distribution, we were able to create a system capable of extracting the colours from each legend. These colours were analyzed by simulating colourblindness to get an estimation of how the user with the colour vision deficiency views these colours, and from this information, we decided whether the image needs recolouring. In the case of recolouring, we select a colour palette we determined to be colourblind-safe for the user and apply it to the image. This, in turn, results in a plot that's easily understandable and interpretable for the user, especially when compared to its original counterpart.

Chapter 1: Project Specification

Edward Tufte, a renowned expert in information visualization, emphasized the importance of colour choice: 'Avoiding catastrophe becomes the first principle in bringing colour to information: Above all, do no harm.'[3] Effective colour choices can significantly enhance the clarity and comprehension of a presentation, while poor selections have the potential to obscure data and confuse the audience. The fundamental role of colour in data visualization is to enable the distinction between different elements or quantitative values, facilitating an intuitive understanding of the data presented. [2]

From a very broad view, the aim here is to create a script that can take in an image of a plot from a user, as well as their colour vision deficiency (CVD), and output a recoloured plot to accommodate for their deficiency. Thus, the goals for the project are as follows :

1.1 Core

- Create a script to recolour problematic plot images for the most common CVD, red-green colour blindness. [4]

- Devise a manner to validate the output to ensure that a specific plot has been correctly recoloured to accommodate the colour vision deficiency, and to validate if the image needs recolouring in the first place.

- Implement this script to work with at least one plot that plots continuous data (E.g. Heatmap) and at least one plot that plots discrete data (E.g. Bar Chart).

- Set up a system to detect the distribution of the colourmap of the inputted image (i.e. is it sequential, diverging, or qualitative?).

- Simulate colourblindness to be able to form a comparison between how the plot is perceived under normal vision vs under a CVD.

1.2 Advanced

- Extend the functionality of the script to accommodate the next most common CVD, blue-yellow colour blindness. [5]

- Create a very simple console user interface to show the potential of the script if it was scaled and deployed. This system would allow the user to input an image, select their deficiency, and get

a recoloured plot in return.

- Implement a manner to select the most representative colours from a colour map.

1.3 In-scope/Out-of-Scope

- Due to its flexibility across different operating systems and ease of use, Matplotlib is the most common data visualization tool in Python. [6]. "According to the 2020 Kaggle Machine Learning and Data Science survey, Matplotlib is the number one data visualization library among Kagglers, leading by a significant margin."[7]. For these reasons, this project will be optimized for images of Matplotlib plots. Although our script may still work for other plots derived from other tools, the main focus here will be Matplotlib.

- We aim to focus on plots where colour has meaning and great significance. We can say that colour is redundant in a plot if a colour legend isn't included. Take these plots below for example:

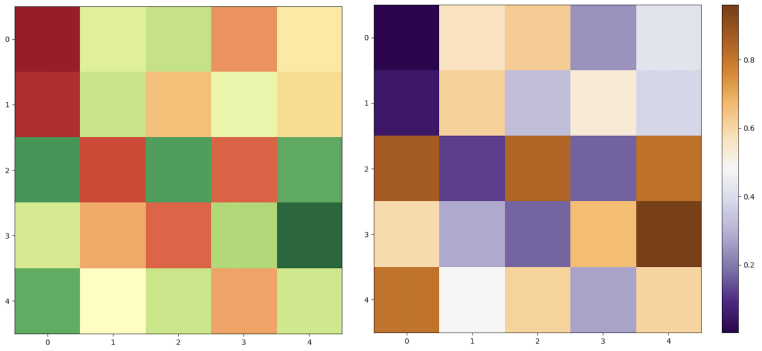


Figure 1.1: Matplotlib Generated Heatmaps

Here we observe an illustration of a heatmap, with reds and greens, and a heat map with purples and oranges. As we can see, for the heatmap with the absence of a colour legend, we have no idea what the significance of these colours is. We can't make out what the reds or greens mean at all. Conversely, for the second heatmap, we can tell that darker oranges tend to 1, and darker purples tend to 0. Colour has no meaning in the first plot, whereas in the second heatmap, it plays a critical role. Because of this, we will focus solely on plots that include a colour legend.

- Monochromacy is very rare, affecting 0.00001% of the population. [8]. For this reason, we will not focus on optimizing plots for this colour vision deficiency.

- White is usually always the best choice when it comes to setting the background colour for our plots. This is because it's the least distracting when it comes to data analysis, making it easier to interpret the data. [9]. For this reason, we will only focus on plots with a white background. We will also assume that the axes will always be black, as it provides great contrast to the white background. For heatmaps, we will maintain black and white for our axes and our background respectively, however, we will permit the use of shades of black and white to represent the data. In discrete plots, we will also maintain black and white for our axes and our background respectively, however, we will only permit the use of black to represent data, as white would not be visible on a white background.

- In 2023, the two most common image formats were PNG and JPEG [10]. For the scope of this project, we will optimize our implementation for these two formats.

Chapter 2: **Introduction**

Colour blindness is a condition that affects approximately 8% of men and 0.5% of women, according to Colour Blindness Awareness (CBA). It's important to note that colour blindness isn't a blindness to colour; rather, it's a deficiency in the ability to see colour differences [5]. Colourblindness is predominantly a genetic condition that affects how the cones in the retina respond to light, but it can also be caused by eye injuries, certain brain tumours, and radiation treatment [4]

Red-green colour vision deficiency is the most common type of colour vision deficiency, making it difficult to distinguish between red and green. There are 4 types of red-green deficiencies, with the most common being Deuteranomaly. Deuteranomaly is a deficiency that makes shades of green look more red. The other three are Protanomaly which makes shades of red look more green and less bright, and Protanopia and Deuteranopia, which make someone completely unable to tell the difference between red and green. [4]

A less common type of colour vision deficiency is blue-yellow colour vision deficiency, with two subcategories: Tritanomaly, which increases the difficulty in differentiating between blue and green and between yellow and red, and Tritanopia. "Tritanopia makes someone unable to tell the difference between blue and green, purple and red, and yellow and pink. It also makes colours look less bright."[4] The rarest form of colour blindness is Monochromacy, which is the inability to see colours [4]. Monochromacy is present in 0.00001% of the population. [8]

Colourblindness can lead to many disadvantages in life, such as a lack of accessibility and efficiency in education and work, as well as less personal enjoyment of digital media.

This project is built around the belief that data visualizations, and digital media as a whole, should adapt to the needs of the entire population so everyone can take advantage of the digital age we're in.

In the upcoming sections, we will take a deep dive into the specifics of realizing this vision, including a comprehensive analysis of related works, our methodologies, and our project work plan.

2.1 Colour Spaces

Throughout this report, I will make mention of the following colour spaces:

RGB (Red, Green, Blue): This is an additive colour model that adds the wavelengths of red, green and blue to create a broad range of colours. [11]

LAB / CIELAB*: This colourspace uses three axes: L* for lightness, a* for green-red colour difference, and b* for blue-yellow colour difference. Lab is designed to approximate human vision, and it covers the entire range of colours that the human eye can perceive. [12]

LCH (Lightness, Chroma, and Hue): LCH is a perceptually uniform colour space in which Lightness " describes the relative brightness of a colour compared to a similarly illuminated white." [13], Chroma describes the intensity of the colour and Hue represents the colour itself.[13]

LMS: This colour space models how the three types of cones in the eye respond to long (L), medium (M), and short (S) wavelengths of light to approximate human colour perception.

2.2 Colour Schemes / Distributions

For the scope of this project, we will focus on sequential (quantitative), diverging (quantitative), and qualitative colour schemes in order to keep the range of our implementation concise while maintaining diversity. Quantitative plots display numerical data, often showing trends, patterns, and relationships between variables. On the other hand, qualitative plots focus on non-numerical data that typically represent categories or discrete data. [2]

Sequential: Data in plots using sequential colour schemes are logically arranged from high to low (or vice versa). The sequence of data categories is characterized by sequential lightness steps. [2]

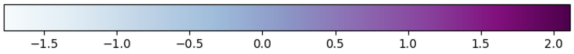


Figure 2.1: Sequential Colour Scheme

Qualitative: These colour schemes are made up of unordered hue steps [2]

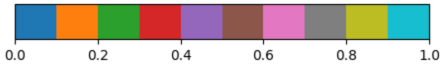


Figure 2.2: Qualitative Colour Scheme

Diverging: This scheme consists of two distinct hues, with a neutral middle. This is typically used to illustrate "progressions outward from a critical midpoint of the data range". [2]

Figure 2.3: Diverging Colour Scheme

2.3 Plotting Libraries

There are a variety of libraries used for plotting in the field of data visualisation such as Tableau, Altair, Matplotlib and ggplot.

As outlined in the **Project Specification** section, Matplotlib is a flexible Python plotting library, and it's by far the most popular data visualization tool. For this reason, we will focus on and optimise our implementation around Matplotlib. Although the focus is on Matplotlib, we will illustrate the range and flexibility of this implementation in the **Additional Testing & Evaluation** section by applying it to ggplot visualisations. Ggplot is an open-source data visualization package used in the R programming language [14].

Chapter 3: Related Work and Ideas

Work relating to specifically recolouring problematic plots is limited, however there exists a larger body of work regarding the recolouring of general images. Recolouring plots to enhance their distinguishability for a specific CVD requires a series of structured steps. Broadly, these steps include identifying the legend as it contains the range of colours used in the visualization, the extraction of these colours, the analysis of these colours to see how perception differs for the user with a CVD (similarity), and the recolouring process. For the duration of this chapter, I will focus on 5 key bodies of work that have shaped my implementation in applying these steps. Additionally, I will acknowledge other works that have contributed valuable insights.

Approach 1 - Reverse Engineering Encodings: [15] Although not directly related to the recolouring of images, the author here provides valuable insights on how one can extract the encodings from an image of a plot. I researched this approach with the initial intention of aiming to manipulate the colour encodings of a plot to recolour it. Although I decided against reverse engineering my plots, I adopted Optical Character Recoginition (OCR), which is capable of detecting text in images.

Approach 2 - Similarity Matrix Analysis: [16] The author here compares different similarity metrics to analyze which is most ideal when it comes to colour similarity. Along with this, they simulate colourblindness and employ a matrix of ratios in order to compare the distinguishability of colours under a CVD. The findings of this study played an instrumental role in choosing my similarity metric, adopting the idea of ratio matrices, as well as simulating colourblindness.

Approaches 3 & 4 - Recolouring Images: [17][18] These approaches focus on two differing methods and mechanisms used to recolour problematic images to make them colourblind-friendly. These two approaches are a subset of a larger body of works that outline the recolouring of general images in which the hues of the image are shifted to the visible spectrum [19][20][21][22]. In the context of recolouring plots in particular, this approach isn't ideal as I will discuss, however, these works gave me a great insight into the simulation of colour blindness to compare the perceived colours with those perceived by normal vision, as well as different ways of gathering a representative set of primary colours from an image.

Approach 5 - Colour Extraction from Plot Legend: [23] In this approach, we focus on how the author goes about isolating the legend of a plot, which is a critical element in efficiently recolouring a plot image since the entire range of colours lies on the legend.

3.1 Approach 1 - Reverse Engineering Encodings

When I was first confronted with the challenge of recolouring problematic plots, my first thought was to research how feasible it would be to reverse engineer the plot images to recover their visual encodings [15]. In this way, I could recover the encoding of a problematic plot and simply change its colour map to one that is colourblind-friendly.

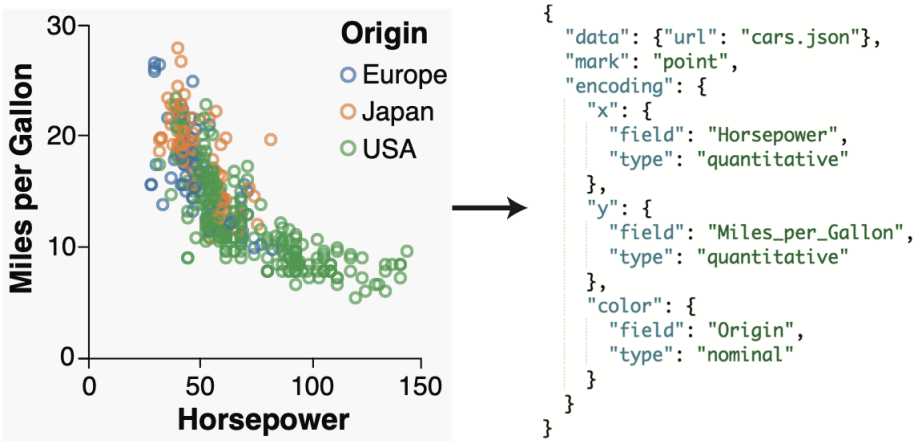


Figure 3.1: Extracting Visual Encoding From a Vega-Lite Illustration, image from [15]

From this approach, the encodings were recovered through a series of steps:

1. **Text Localization and Recognition** - This step involved the identification and interpretation of text within the plot, as well as determining their bounding boxes and the role of the text elements (e.g. title, axis label)

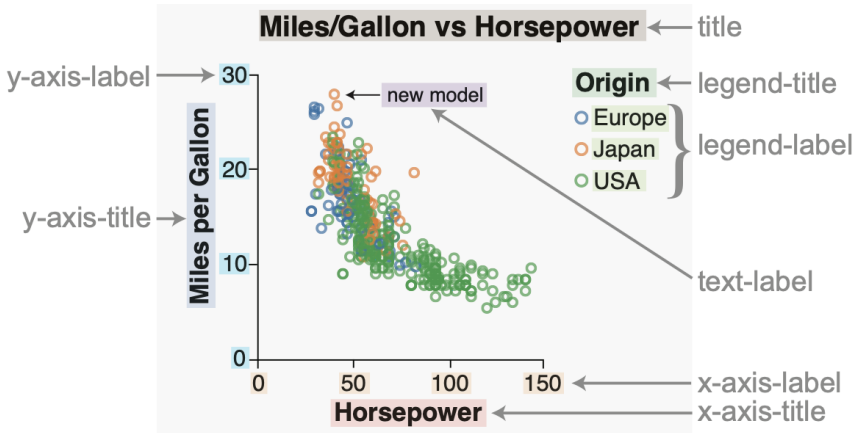


Figure 3.2: Recognition of Text Roles, image from [15]

2. **Mark Type Classification** - Using a Convolutional Neural Network (CNN), the mark of the illustration (e.g. bar, point) was recovered.

3. **Inferring Data Types, Axis Domain & Range** - Optical Character Recognition (OCR) was used to read and interpret the axis labels, as well as the tick marks. This was further extended to infer the domain, range, and data type of the illustration.

4. **Combining Recovered Text and Mark to Infer Visual Encodings** - When the text elements and their role have been recovered, and the mark has been classified, the script can infer the encoding of the illustration.

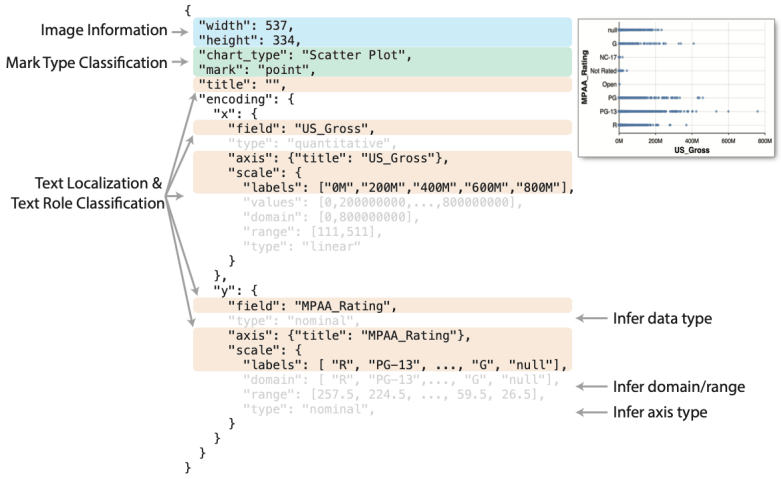


Figure 3.3: Visual encodings for a scatter plot highlighting how each subsection of the encoding is recovered, image from [15]

Conclusion

Colour recovery was not in the scope of this particular work, however, I believe that this wouldn't be the most ideal approach to take to recolour problematic images for the following reasons:

1. **Unnecessary Complexity**: For the scope of recolouring problematic plot images, all we need to extract is the colour and the legend from the visualization. The approach will become over-complicated if we incorporate more elements of the visualization, such as the text and mark. Furthermore, highly complex or unconventional plots could pose a challenge for accurate text localization.

2. **Text Dependence** - This approach is heavily reliant on the use of text elements for the interpretation of plots. Let's take this Matplotlib-generated illustration for example:

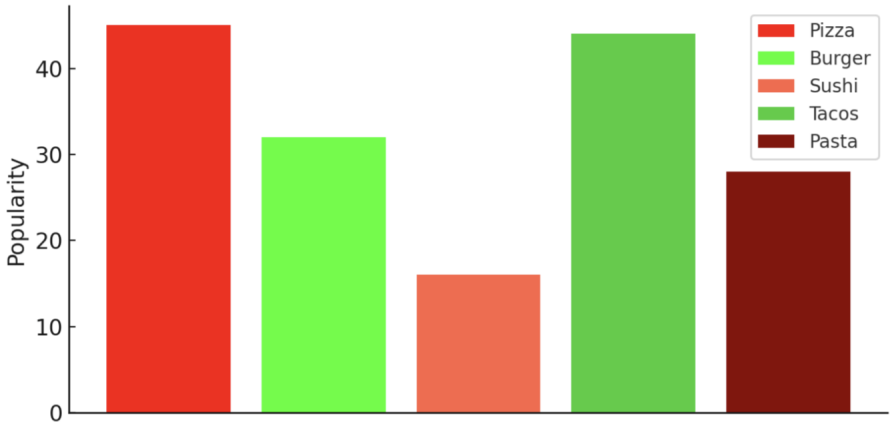


Figure 3.4: Matplotlib generated plot with a problematic colourmap

In the above plot, the text is used sparingly and quite frankly isn't descriptive enough. This lack of text here would greatly limit the system's ability to understand and categorize the bar chart's components as text is the primary element used for classification and interpretation.

3.2 Approach 2 - Similarity Matrix Analysis

This approach applies much more to the context of my project. Colours are first extracted from a heatmap. Following this, a similarity metric is used to evaluate how distinguishable the colours are for the human eye. This step involves simulating colour blindness on these sets of colours, which allows one to analyze how someone with a specific CVD perceives these colours. A matrix is then used to output the ratios of a pair of colour's distinguishability. This is essentially how distinguishable a set of colours is to someone who doesn't have the CVD vs someone who does have it. [16]

Colour Extraction: Twenty evenly spaced colours on the spectrum were selected, approximately every 5th percentile.

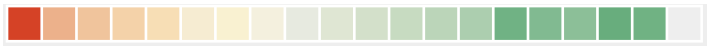


Figure 3.5: Twenty evenly spaced colours along the spectrum of the heatmap, image from [16]

Compute Similarity: Here, different similarity metrics are used to test which is most accurate.

	RGB	Lch	Lab	deltaE
	255	41.4	66.3	26.9
	255	40.0	58.0	34.7

Figure 3.6: Different similarity metrics, image from [16]

Colours along the spectrum where hues change rapidly (like red to blue) are likely more distinguishable than areas with gradual changes in hue (like yellow to green) [24]. As pink to blue reflects a more rapid change in hue when compared to yellow to green, we would generally expect that pink and blue are more distinguishable than yellow and green.

- **RGB (Euclidean Distance):** Here, both yellow/green and purple/blue are computed to be equidistant. This makes sense mathematically since it takes 255 steps from yellow (255,255,0) to green (0,255,0), and 255 steps from pink (255,0,255) to blue (0,0,255). This doesn't make sense visually as RGB does not account for the complexities of human vision.

- **Lch (Euclidean Distance):** In the Lch space, the pairs are nearly equidistant.

- **Lab (Euclidean Distance):** In the Lab space, the green/yellow pair has a larger distance, meaning it's more distinguishable than pink/blue, which shouldn't be the case.

- **deltaE:** The algorithm used here to compute colour differences is called deltaE or CMC l:c. DeltaE is based on the Lch colour model and it perceives these colour differences optimally. The one minor pitfall here is that it's not symmetrical, this means that the difference between green and yellow and yellow to green will slightly differ. To mitigate this issue, the mean of both distances is used:

$$dist(c_1, c_2) = \frac{deltaE(c_1, c_2) + deltaE(c_2, c_1)}{2}$$

Figure 3.7: deltaE algorithm, image from [16]

- **Compute colour distance matrix:** A matrix was composed to illustrate the distance between each pair of colours.

- **Simulating Colour Blindness:** "Colour blindness simulation is done by mapping colours from RGB to a reduced colour space. It means that you have a function that gets one colour as input and returns a new colour."[16]. This 'new colour' will illustrate how a person with a specific CVD perceives the original colour. A JavaScript package, NPM, was used to achieve this.

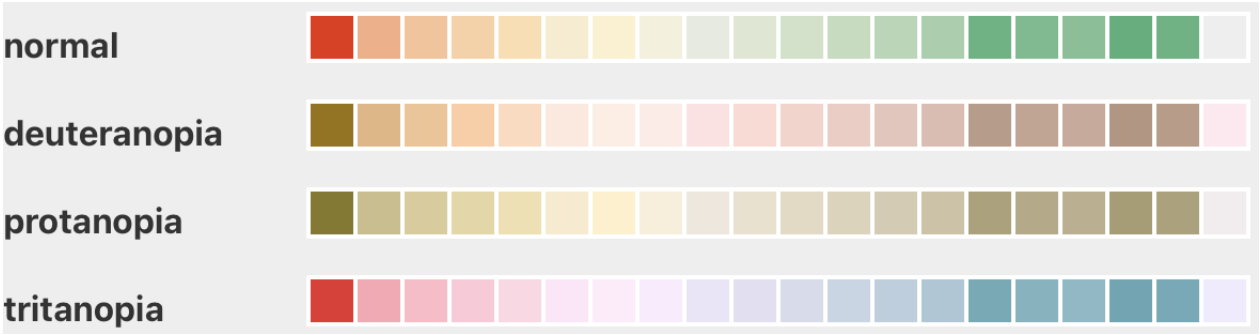


Figure 3.8: Red/green and blue/yellow colour blindness simulated to illustrate how they perceive the selected twenty colours, image from [16]

- **Difference ratios between normal and colourblind vision:** After simulating colourblindness on the twenty selected colours, another colour matrix is computed to illustrate the distance between each pair of colours as seen by someone with a specific CVD, in this case, red/green colourblindness. Following this, distances under normal vision are compared with the distances under a colour blindness simulation in the form of ratios, using the formula:

$$ratio(c_1, c_2) = \frac{dist_{normal}(c_1, c_2)}{dist_{colorblind}(c_1, c_2)}$$

Figure 3.9: A ratio of colour similarity to normal vision over colour similarity to simulated colour-blindness, image from [16]

Example: If we use the formula to compare normal vision with red/green colourblindness, a ratio of 13.2 would mean that the pair of colours is 13.2 times more differentiable with normal vision when compared to red/green colourblindness.

This approach wasn't extended to outline methods such as colour correction, validation, etc, however, a lot of useful information was derived here, such as:

DeltaE: From the comparison of Euclidean in the RGB, Lch, and Lab spaces, deltaE demonstrated superior performance. It was able to more accurately reflect human perception.

Colour Selection: Twenty evenly spaced colours along the spectrum were selected, approximately every 5th percentile. To analyze the entire colour palette for a specific CVD, this method gives an effective representation of the range of colours across the entire heatmap.

Colourblind Simulation: By simulating colourblindness, we can get a better grasp of how a user visualizes an image with a specific CVD. By simulating the similarity of colours under a CVD, we can use these metrics to aid us in selecting a colourblind safe palette for the user's CVD.

3.3 Approach 3 & 4 - Recolouring Images

3.3.1 Approach 3 - Recolouring with GMM Clustering

After realizing that extracting encodings directly from the plot image was not the optimal approach, I shifted my focus to an alternative method. My research led me to a technique originally developed for recolouring problematic images of all sorts, not just plots. Regardless of that, it still provided valuable insights that could be applied to our context.

This approach involved extracting the colours from the image, grouping similar colours through clustering, adjusting the mean colour of each cluster to make the colours more distinguishable for people with a specific CVD, and adjusting colours to ensure smooth transitions. [17]

Here's a more detailed explanation of the steps :

Colour Extraction: The colour of each pixel is extracted from the image in the CIELab colour space. This colour space is effective for this application as it is designed to approximate human vision [12].

Similarity Grouping: The perceptual differences between colours were measured by Euclidean distance in the CIELab colourspace. The process of clustering starts with a basic algorithm, K-means clustering, which will provide a basic grouping of similar colours. Gaussian Mixture Modelling (GMM), an advanced clustering method, refines these basic clusters through the Expectation-Maximization (E-M) algorithm. GMM is a "probabilistic and interpretable model for clustering", where the model computes the optimal manner to best group colours. [25]. The E-M algorithm iteratively refines and adjusts both the Gaussian distribution parameters and the probabilities of colour assignments to a cluster.

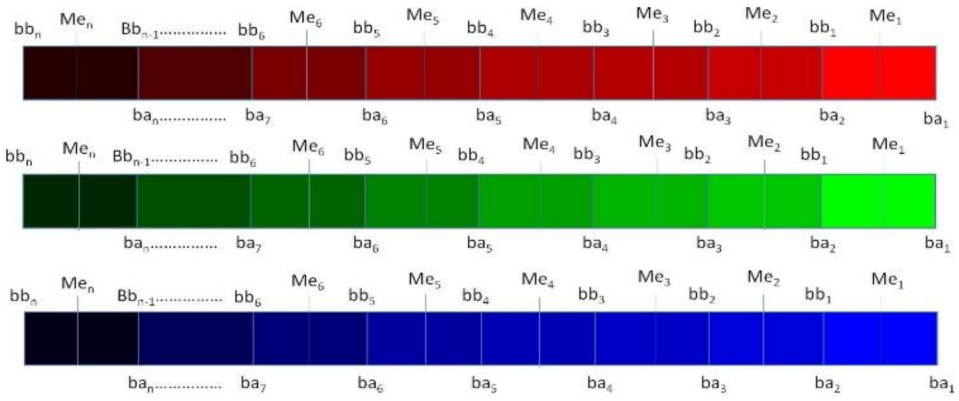


Figure 3.10: Formed colour clusters, image from [17]

Colour Correction: Once the colours have been grouped, the next task is to tweak the colours to increase legibility for users with the CVD. This is done by moving the mean vector of each colour group (each cluster) to make colours more distinguishable. The mean represents the central colour of a particular group. When we adjust this colour, we shift the entire cluster towards a more distinguishable hue. This is done to create more contrast between each colour group, thus

making them more legible. The chroma (intensity of colour) and the hue of the colours are then slightly adjusted to ensure a smooth natural transition between colours and to mitigate abrupt colour changes.

3.3.2 Approach 4 - Fuzzy Clustering Aided recolouring

In this approach, [18] proposes image recolouring, using fuzzy clusters to extract the key colours of each image. The illustration below highlights the basic structure of their recolouring process.

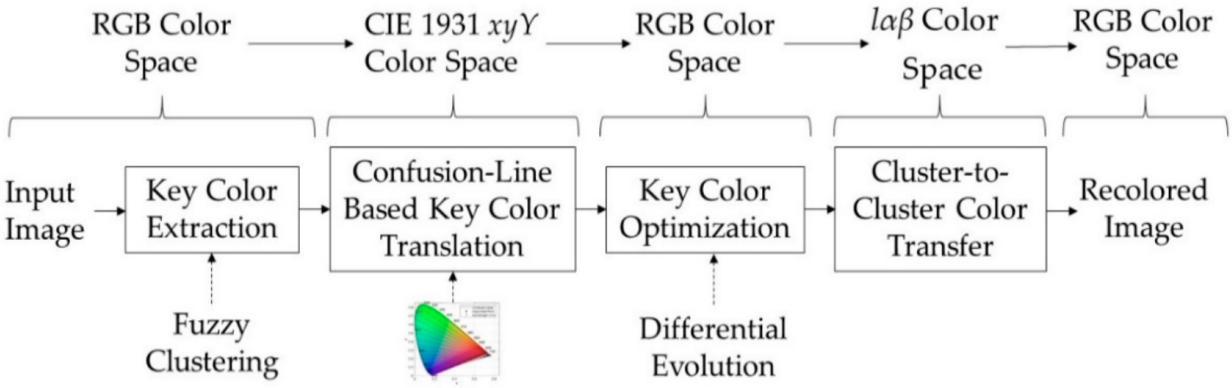


Figure 3.11: Methodology with Fuzzy Clustering, image from [18]

In fuzzy clustering, data points can belong to more than one cluster. Data points have a certain degree of membership, ranging from 0 (no membership) to 1 (full membership). [18]

Conclusion

No Unnecessary Complexity: Here, colour is the only encoding that we need to extract from the image, unlike **Approach 1**, where the use of text recognition and localization was critical.

CIELab*: In these approaches, the perceptual differences between colours were determined in the CIELab* colour space. CIELab uses three axes: L* for lightness, a* for green-red colour difference, and b* for blue-yellow colour difference. CIELab* is designed to approximate human vision and it covers the entire range of colours that the human eye can perceive. [12]. This means that colours with a short distance between each other in this colour space are less distinguishable by the human eye compared to a pair of colours with a larger distance.

That being said, these approaches come with their limitations when put into the context of our project, which is solely dealing with image plots:

Colour Correction (Moving Average): While effective in certain contexts, this colour correction technique may not be the most appropriate for plot images. Shifting the colour hues of a cluster can inadvertently alter the data representation in a plot. Plots are precision tools for data visualization and illustration, so even the slightest colour changes can lead to a misrepresentation of the underlying data. Where colour acts as a direct representation of data (i.e. heatmaps), this is particularly crucial. [26]

Pre-existing Colourblind-Safe Palettes: Tools like Matplotlib and OpenCV have colour palettes that are optimized for users with CVDs [27]. These colour palettes are designed to provide differentiation and contrast without detriment to the representation of the data. In our context, it would be much safer and more straightforward to make use of these pre-designed palettes to

ensure we preserve the integrity of our data representation.

Uncertainty of GMM: GMM is a probabilistic algorithm that clusters data based on their likelihood of belonging to a Gaussian distribution. GMM makes assumptions about the shape and spread of the data, which may not correlate with the distribution of the colours in the image. "Due to the uncertainty of GMM clustering, the error of recolouring may occur". [17]

3.4 Approach 5 - Colour Extraction from Plot Legend

Here, [23] isolates the legend from the image of the plot, extracts the colours from the legend, and recolours the image. The legends here are extracted by the user highlighting the region and providing it as an input. Following this, further operations are carried out: binarizing the image, flood fill to recover the colour gradient, and morphological erosion to separate text from legends.

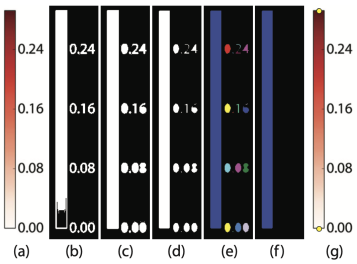


Figure 3.12: "Extracting colour from continuous colour legends. (a) Original legend image. (b) Binarized image. (c) Flood fill to recover colour gradient. (d) Morphological erosion to separate text from legends. (e) Connected components. (f) Colour gradient is the largest connected component. (g) Yellow circles indicate positions of minimum and maximum values." [23], image from [23]

The same is done for discrete legends:

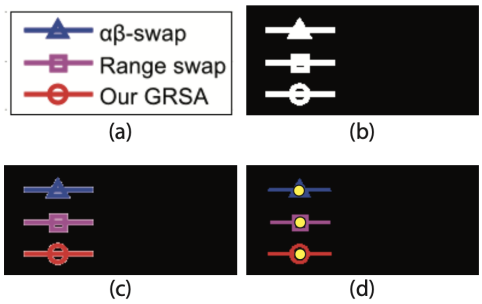


Figure 3.13: " Extracting colour from discrete colour legends. (a) Original legend image. (b) Mask combining background and grayscale masks. (c) colour pixels after applying the mask. (d) Yellow circles indicate pixels with representative colours found via cluster analysis." [23], image from [23]

After this, a Convolutional Neural Network (CNN) is used on the detected legend to determine whether it is discrete or continuous. Along with this, [23] attempted to extract text with Tesseract OCR. In the **Outline of Approach** section, I discuss my experience with this specific engine, compared with other OCR engines.

Conclusion

[23]'s approach is interesting as it confirmed that a rule-based approach in automatically detecting the legend is not the optimal way to go. Later in the **Outline of Approach** section, I highlight my experience in attempting to implement a rule-based approach. For the scope of my implementation, I wanted to create a more automated and robust experience for the user by not having them highlight the legend manually. For that reason, I opted to go for the machine learning approach which is discussed in depth in the **Outline of Approach** section.

3.5 Research Conclusions:

My implementation was inspired and influenced by my background research. For example, both Approaches 1 and 5 utilize OCR which is used throughout my implementation. Approaches 2, 3, and 4 discuss the simulation of colourblindness, which is a central aspect of my system. Approach 2 and 6 outline colour extraction from the legend, which is a key aspect I adopted. In conclusion, I will revisit these approaches once again while outlining my implementation. As well as that, I will reference many other works related to specific mechanisms of my procedure. Along with that, I will also discuss, explore, and outline alternative libraries analogous to the ones opted for in the project.

Chapter 4: Data Considerations

With regards to the data used in this implementation, it is structured and static. We use Matplotlib-generated plot images to train our object detection model. Since these images were generated in-house through Matplotlib, this waives off any privacy or ethical issues with how the data is collected or used. That being said, all user-inputted images will be utilized under the assumption that the user has the right to use the image for this specific purpose. Given the structured nature of Matplotlib plots, data cleaning will be minimal. The rationale behind using Matplotlib is due to its popularity, this is justified in our **Project Specification** section.

Approximately 750 Matplotlib-generated plot images were annotated for the presence of a legend, its distribution (diverging, sequential, or qualitative), the nature of the legend (either a discrete legend or continuous legend), the presence and dimensions of a heatmap, and for the presence of gridlines (indicated as True or False). These plots were manually annotated via Roboflow. "Roboflow also allows access to labelling datasets, annotating images, preprocessing, augmentation process, and other beneficial functions to handle the dataset."[28]. I will discuss this further in the **Outline of Approach** section, however, I was responsible for precisely drawing the bounding box around aspects such as the legend and the heatmap. This took a considerable amount of time, however, it resulted in an object detection model capable of accurately detecting the legend, colour distribution, and heatmap dimensions on a diverse set of test images. The object detection model was tested on 1000 Matplotlib-generated plots with a continuous legend, and a further 1000 Matplotlib-generated plots with a discrete legend. I meticulously validated the results of these tests by manually comparing the input vs the output to calculate the accuracy of my model, which took a significant amount of time.

To create the "Matplotlib-generated plot images" used for the training and test set, I designed a Python script that creates plots by randomly selecting data patterns (such as uniform, normal, log, sin wave, and linear distributions), colourmaps and their distributions (diverging, sequential, qualitative), plot sizes, legend locations, legend dimensions & orientation, visual styles (such as grid lines, borderlines, etc), mark types, font sizes, number of data points, etc. in order employ a strategy that mimics a broad spectrum of real-world data visualizations. This approach allowed me to generate a comprehensive and diverse dataset which was ideal for testing the performance of my implementation.

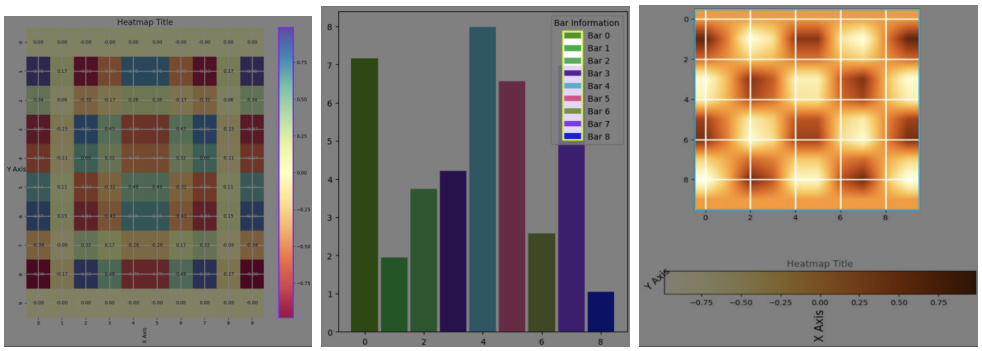


Figure 4.1: Annotating continuous legend, discrete legend and heatmap respectively with Roboflow by manually highlighting the region of interest

It is assumed that any images inputted by the user are either PNGs or JPEGs. Along with that, it's assumed that these inputted plots have a legend, white background, and black axes. These points are stated and justified above in the **Project Specification** section.

Chapter 5: Outline of Approach

My approach will consist of the following steps:

- **Legend Detection:** *This section outlines the attempted and proposed methods to accurately detect the legend from various plots.*
- **Colour Extraction & Distribution Analysis:** *This section describes the process of extracting colours from the legend and analyzing the distribution of the legend (diverging, sequential and qualitative)*
- **Representative Colour Selection:** *Here walk through how we extract a representative sample of colours that we will use for* ***Colour Similarity Analysis***
- **Colour Similarity Analysis:** This section compares the most effective similarity metrics and gives the logic behind our selection of metric.
- **Colourblind Simulation:** *This section covers the simulation of colour blindness to get an estimation of how the user with the CVD perceives the plot.*
- **Comparison:** *This section illustrates the matrix of ratios in order to give us a better view of how distinguishable the colour palette is under a CVD vs normal vision*
- **Pre-check & Validation:** *Here we determine if the input plot needs recolouring. As well as that, we outline our chosen colourblind safe palettes across the different colour distributions (diverging, sequential, qualitative).*
- **Colourmap Creation:** *This section covers the series of steps taken (such as interpolation) to create an ideal colourmap and addresses the issues brought forward by plots inputted with anti-aliasing.*
- **Applying Colourmap:** *This section discusses the application of the previously created colourmap to make the input image colourblind safe.*
- **Preserving Text:** *This section covers the steps taken to preserve the text/numbers in plots, which are often discoloured after applying the colour mapping.*
- **Preserving Gridlines:** *Similar to the previous section, this section outlines how we preserved gridlines, which are often discoloured after applying the colour mapping.*

The Gitlab repository for this implementation can be found here

5.1 Legend Detection:

Initially, I opted for a rule-based approach for legend detection. I converted the plot image into grayscale and began to identify contours with significant colour. For each of the contours, we calculated their position and their aspect ratio (width and height). A colour legend in a heatmap is typically found at the edge of the heatmap, with a longer width than height (or vice versa). Our method found a contour that fits this description and outputted it. If a plot has no legend, then colour has no useful meaning. For example, the contours for the heatmap below are the heatmap itself and the colour legend:

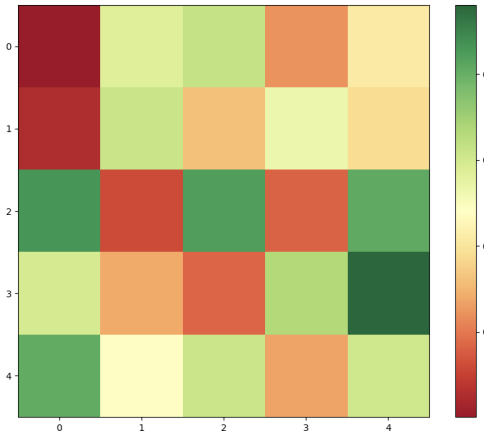


Figure 5.1: Matplotlib heat map with colourlegend

However, this rule-based approach proved to be ineffective and lacked robustness across a large number of examples. In many instances, part of the heatmap itself would be detected as the legend, and the continuous legend would be left undetected. Across qualitative plots with discrete legends, this approach proved to be even more ineffective than in heatmaps. For qualitative plots, I would aim to find regions of colour that lie on approximately the same x-coordinate and have a similar size.

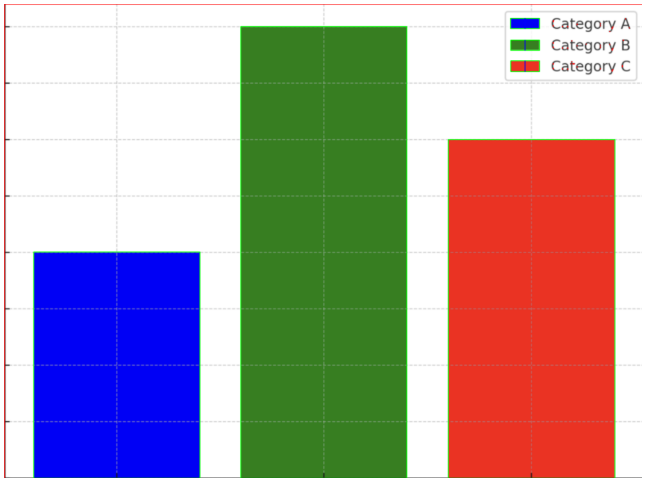


Figure 5.2: Rule-based approach highlighting the three legend colours in green, and striking a blue line through each one to denote they lie on the same x-axis and size

In the example above, we draw green bounding boxes around regions of colour, and we draw a blue line through the colour regions with a similar x value, as well as size. Here, the rule-based approach can correctly isolate the colours in the legend. That being said, the results were inconsistent:

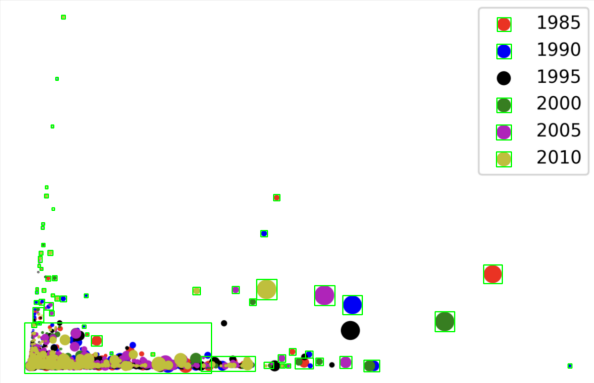


Figure 5.3: Inconsistencies of the Rule-based Approach. Original image is from [29]

For scatter plots, it's not rare that some points will have similar sizes and x-values, which led to a multitude of different regions being detected as a legend. Scatter plots exposed the issues regarding the rule-based approach, so my research led me to a different approach.

Machine Learning Approach

Instead of creating a fixed set of rules that each image must satisfy, I went with a machine-learning object detection model, YOLOv8 (You Only Look Once). YOLOv8 makes use of "the sigmoid function as the activation function for the objectness score, representing the probability that the bounding box contains an object."[30]. YOLOv8 is a powerful object detection model that strikes a successful balance between speed and accuracy, making it perfect for my implementation in detecting the legends from plot images. [31] compared YOLO with other two major object detection models, Single Shot Detection (SSD) and Faster Region-based Convolutional Neural Networks (Faster R-CNN). [31] concluded that YOLO provided the best overall performance and accuracy. This comparative analysis was carried out on an older version of YOLOv8 which was still the best performer. This, coupled with the plethora of documentation and ease of use of YOLOv8, led me to choose this model over the rest.

I carried out annotation through Roboflow due to the ease of use. "Roboflow also allows access to labelling datasets, annotating images, preprocessing, augmentation process, and other beneficial functions to handle the dataset."[28] This made the annotation process very simple. In creating my model, I manually annotated approximately 750 images in which I would highlight the region of interest (the legend), and I saw a high degree of accuracy in legend detection. On a test of 1000 randomly generated Matplotlib plots with a continuous legend, the model successfully detected the legend on 98% of occasions. On a test of 1000 randomly generated plots with a discrete legend, the model successfully detected the legend on 99% of occasions. These percentages can be improved upon even more by adding more images to the training set.

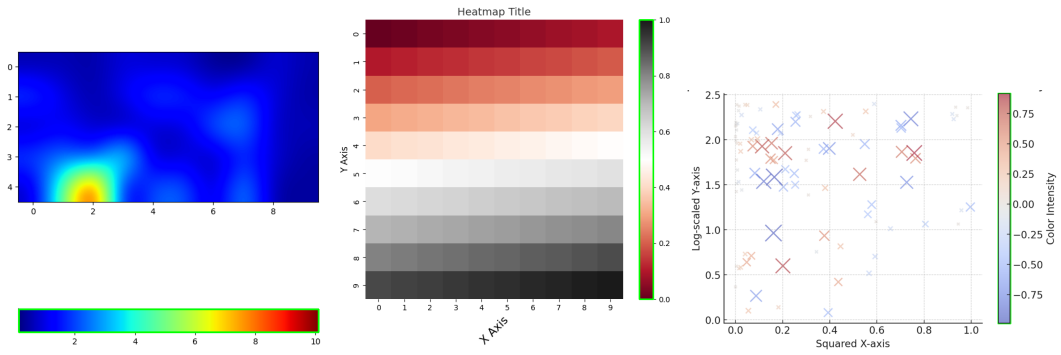


Figure 5.4: Example continuous legends detected through YOLOv8, highlighted in green

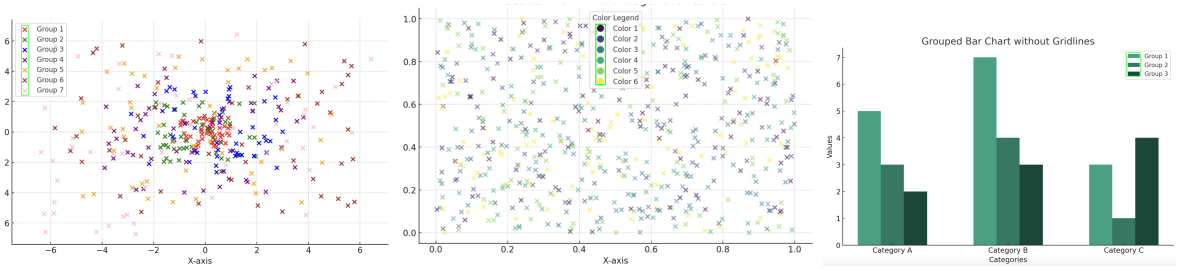


Figure 5.5: Example discrete legends detected through YOLOv8, highlighted in green

5.2 Colour Extraction & Distribution Analysis:

Once we've isolated our legend, we extract all the colours from it. In a continuous legend, the colours are extracted by using the Numpy library to retrieve the unique RGB values from a vertical/horizontal slice of the detected bounding box. Following this, we aim to determine whether our distribution is diverging, qualitative, or sequential. From [31] highlighting the power of YOLOv8, and [28] illustrating the simplicity of Roboflow, I labelled each continuous legend with their distribution (diverging, qualitative, or sequential) while annotating my model mentioned in the **Legend Detection** section.

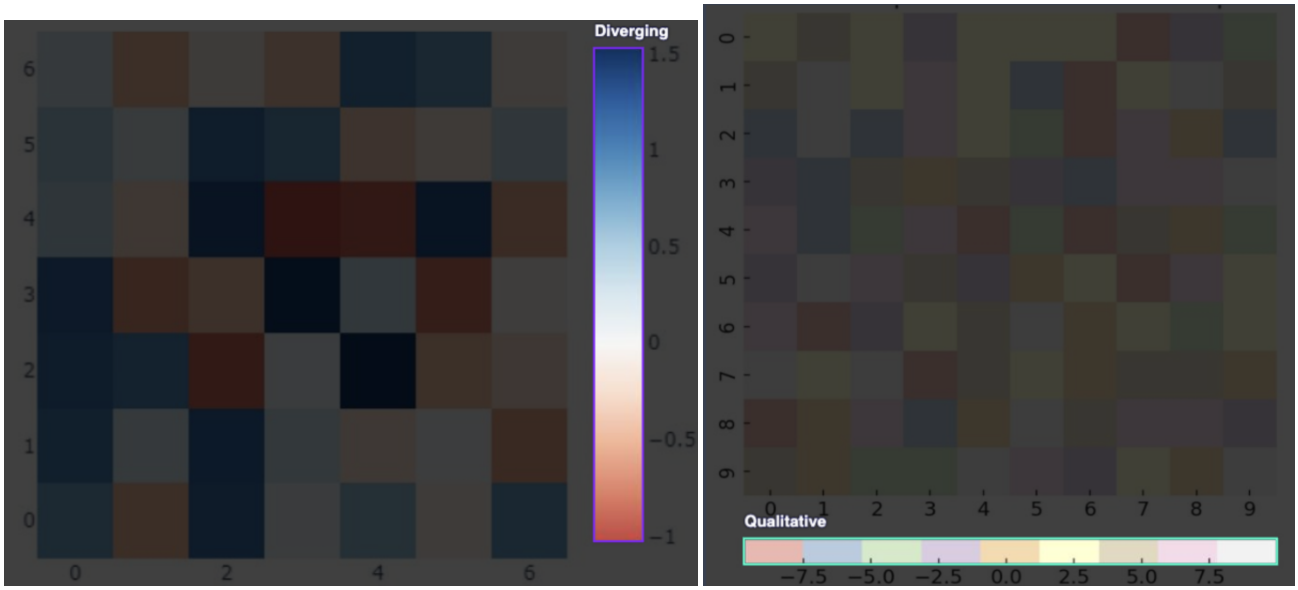


Figure 5.6: Annotating continuous legend distributions with Roboflow by manually highlighting the region containing the legend

Initially, I aimed to implement a rule-based approach. I defined a diverging colourmap to have a neutral hue in the middle, and two extremes on each end, sequential being a hue transition, and qualitative being an unordered block of hues. [32] I implemented logic to detect distributions based on these rules, however, the results were inconsistent. This was especially evident for miscellaneous colourmaps, such as gist and jet (rainbow colourmaps), which would often get misdetected to be qualitative due to their irregularity. Using YOLOv8 to detect colour distributions led to significant improvements in accuracy. On a test of 1000 randomly generated Matplotlib plots with a continuous legend, the model successfully detected the distribution on 96.5% of occasions. Once again, this rate can be significantly improved by adding more annotations to the training set.

5.3 Representative Colour Selection:

When it comes to continuous legends, we will need to select a representative set of colours to compare the present colour palette in search of a more optimal one. My initial approach aligned with [16], where 20 colours were selected from the continuous legend at every 5th percentile to give a fair representation of the range of colours in the heatmap. Following this, I experimented with [25]'s approach to clustering via Gaussian Mixture Modelling (GMM) clustering to find the most representative colours. That being said, I decided to side with [16]'s approach of selecting colours every 5th percentile as that gave a better representation of the entire colourmap.

For discrete legends, this approach was made very simple through YOLOv8. After detecting the legend, I carried out morphological operations on the legend to ensure the colours were detected correctly. I applied a dilation mask to expand the detected colour region with a given kernel. Dilation masks help to close tiny holes detected within colour regions, this allows these regions to be more pronounced and cohesive [33]. I also implemented a small, yet reliable rule-based approach to extract colour regions in the detected legend with a similar x-coordinate. This was especially helpful in ignoring data points that were under-lapping the legend.

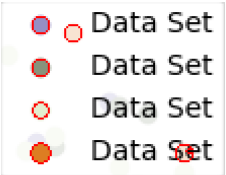


Figure 5.7: Avoiding underlapping data points

The image above shows that the 4 legend colours are correctly detected, along with 2 other points under-lapping the legend. Following this, we extract colours only from contours with approximately the same x value (+- 1). In cases where we would have 2 sets of data points with similar x values, we would select the group with a constant y value between each point. In the case where we only have 2 colours in our legend and 2 sets of data points with similar x values, we can't test for constant y, so we picked the pair of colours that had a higher average saturation, as under-lapped colours always tend to have lower saturation as they were overlapped by a white legend background [34]. This rule proved to be robust over many tests and allowed us to only extract the legend colours, which are our representative colours in this case. We extract the unique shades of colours from each one of our detected legend colours, which will prove to be important when colour mapping.

5.4 Colour Similarity Analysis:

From [16]'s similarity analysis across different colour spaces, using different metrics, it was clear that the use of deltaE in the lab space was most accurate.

Different forms of the deltaE formula provide "about 95% accuracy in correlation to human perception of colour differences" [35]. [36] carried out a survey on a group of observers to test which deltaE formula best aligns with human perception. The top performers were deltaE CMC(l:c), CIEDE2000 and CIE94. "Although CIE94 and CIEDE2000 have been shown to be superior colour difference equations when skilled observers are employed, the textile industry is concerned about the judgment of average observers who comprise the end consumer" [36]. Following on, "this research indicated that if untrained observers who have average discrimination, and no commercial bias, it is determined that CMC is the best equation to use for the evaluation"[36]. For the scope

of this project, the aim is to develop an implementation to recolour problematic images for colour blindness. From my interpretation, an implementation like this is better aimed at accommodating the average, untrained observer. Based on the research from [36], and personal testing conducted between the three deltaE formulas, I have decided that deltaE CMC(l:c) is most appropriate.

DeltaE CMC(l:c) was developed in 1984 to better quantify the difference between two colours in a way that aligns with human visual perception when compared to the earlier CIELAB E* formula. [37]

CMC(*l*:*c*)

$$\Delta E = \left[\left(\frac{\Delta L^*}{l S_L} \right)^2 + \left(\frac{\Delta C^*}{c S_C} \right)^2 + \left(\frac{\Delta H^*}{S_H} \right)^2 \right]^{1/2} \qquad (1)$$

where $S_L = \dfrac{0.04975 L_1^*}{1 + 0.01765 L_1^*}$ unless $L_1^* < 16$ when $S_L = 0.511$

$$S_C = \frac{0.0638 C_1^*}{1 + 0.0131 C_1^*} + 0.638$$

$$S_H = S_C \, (Tf + 1 - f)$$

$$f = \left[\frac{(C_1^*)^4}{(C_1^*)^4 + 1900} \right]^{1/2}$$

$T = 0.36 + |0.4\cos(h_1 + 35)|$ unless h_1 is between 164° and 345°,
when $T = 0.56 + |0.2\cos(h_1 + 168)|$

Figure 5.8: CMC(l:c) formula, image from [37]

After using deltaE CMC to compute the colour differences for each of the representative colours, I output my results to a matrix [16]. Below is a colour difference matrix of the representative colours from the heatmap in **Figure 5.1**.

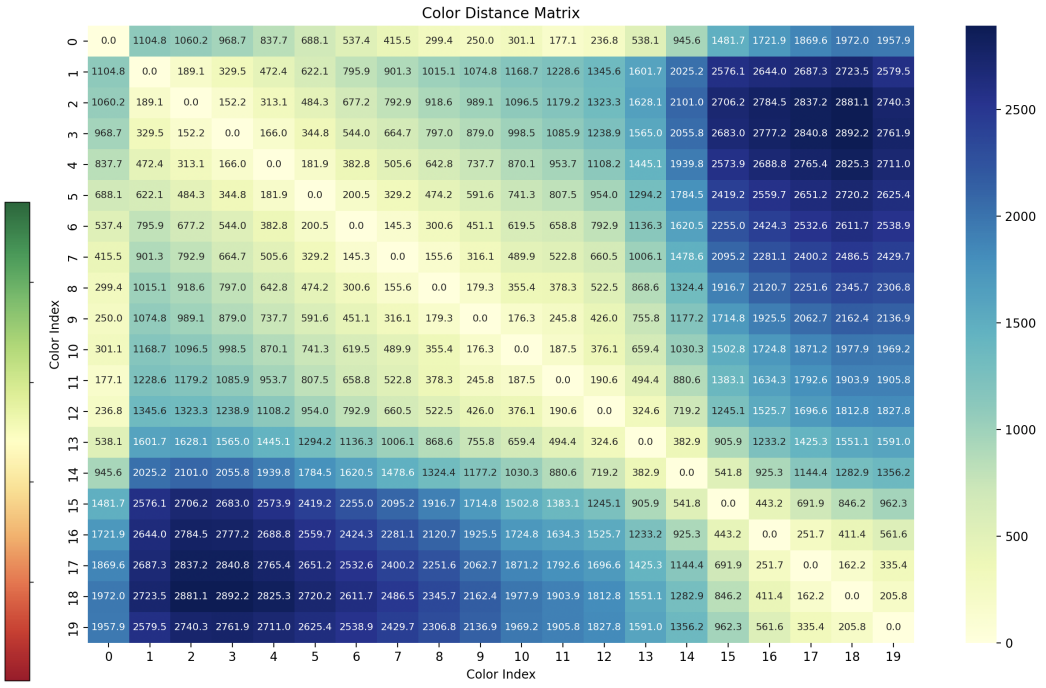


Figure 5.9: Colour distance matrix for our 20 representative colours

As we can see, a dark blue cell indicates a pair of colours that have a large distance between one another, and are distinguishable under normal vision. The lighter the colour gets, the less distinguishable these colours are.

5.5 Colourblind Simulation:

"Using colour blindness simulators reveals how images may appear to users with a variety of colour blindness conditions" [22]. By simulating colourblindness in Python, we can analyze how our similarity metric, deltaE CMC, varies between normal and colourblind vision. The list below highlights different colourblind simulators and their access medium (i.e. API, web app, etc) [22]:

1) **Color Oracle3** [38]:

-***Platform***: "A downloaded program that works offline for Windows, Mac, and Linux". [22]

2) **Vischeck** [39]:

- ***Platform***: A web-app and offline plugin for Adobe Photoshop and ImageJ [22]

3) **Coblis** [40]:

-***Platform***: A web-app.

4) **DaltonLens** [41]:

-***Platform***: A web-app and Python plugin for various different simulators.

For my implementation, the DaltonLens package was the most appropriate due to its ease of integration with Python. DaltonLens implements "several variants of the LMS colour vision model, which is based on the responses of the three types of cones in the human eye"[41]. Through testing, this package proved to be useful. Moreover, DaltonLens comes with a variety of different simulators, with the Brettel 1997 simulator being the most accurate and reliable across protanopia/protanomaly, deuteranopia/deuteranomaly, and tritanopia/tritanomaly [41]. The Brettel 1997 simulator projects colour stimuli onto a reduced stimulus surface in a three-dimensional LMS space. The accuracy of this simulator was validated by observers with colour vision deficiencies [42].

Using the plot in **Figure 5.1**, we simulate deuteranopia/deuteranomaly with maximum severity. We apply the maximum severity for a very good reason: If we can correct colourblindness for the most severe case of a particular colour vision deficiency, then it will also be corrected for lesser cases. Following this, we compute the similarity matrix for severe deuteranopia/deuteranomaly using deltaE. [16]

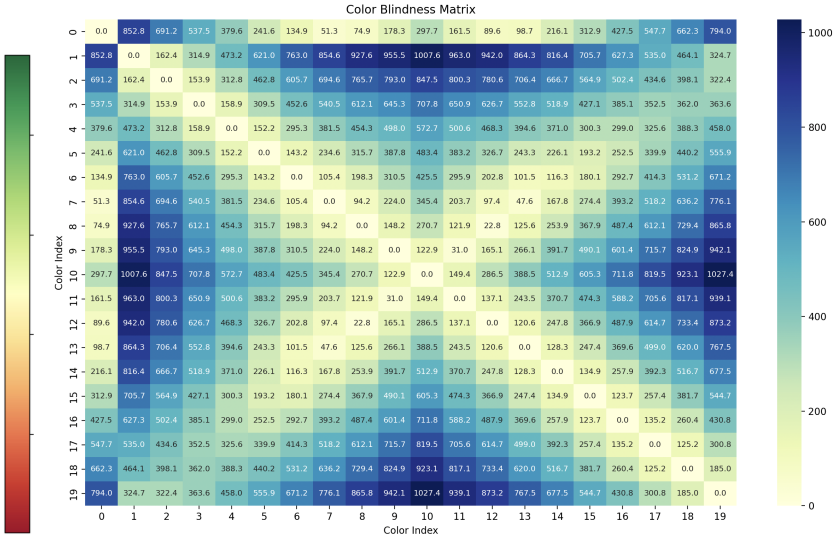


Figure 5.10: Colour distance matrix under Simulated Conditions

As we can see in **Figure 5.10**, colour distances have drastically decreased under simulated colour blindness. This is evident with the change of scale in the continuous legend from 2500 in **Figure 5.9** to 1000. The tighter scale gives a clear indication that colours in this heatmap are less distinguishable for severe deuteranopia/deuteranomaly.

5.6 Comparison:

Once we have our two matrices, we divide our values in the normal matrix by the matrix with simulated colourblindness. This gives us a matrix of ratios [16]. E.g. A ratio of 4 in a cell would mean that the specific colour pair is 4 times less differentiable under colourblind circumstances when compared to normal vision.

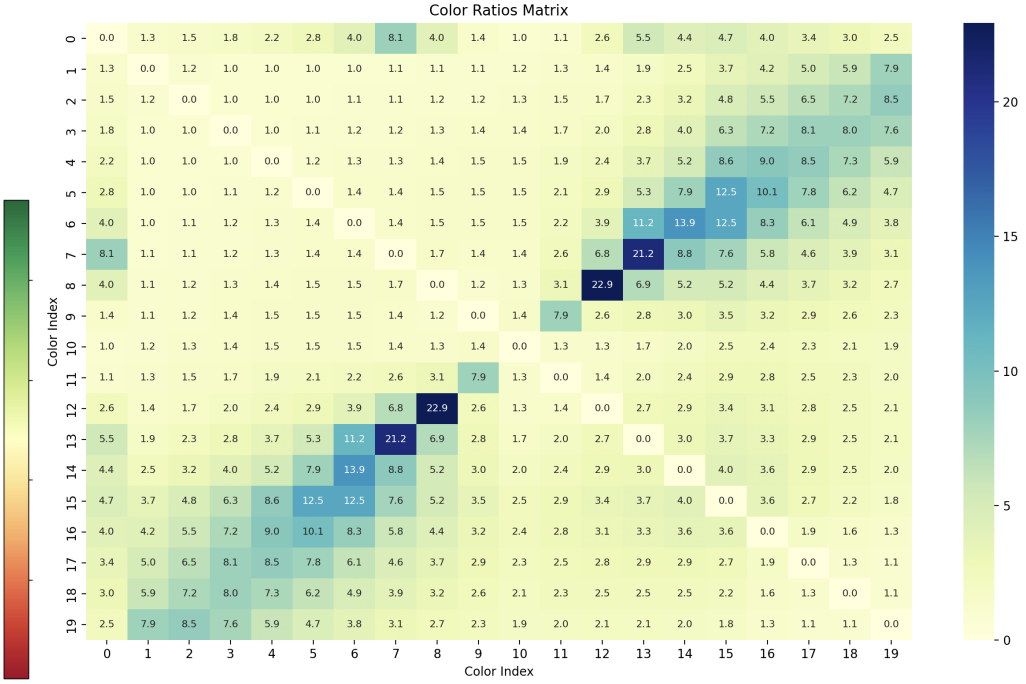


Figure 5.11: Matrix of Ratios

Now that we have our ratio matrix in Figure **5.11**, we can begin to compare this matrix with ratio matrices from other colour palettes. The goal here is to get the matrix with the lowest average ratios. This means that we're looking for the most distinguishable colour palette under the most severe case of deuteranopia/deuteranomaly.

5.7 Pre-check & Validation:

Before we recolour a plot, we must first determine if it needs recolouring in the first place. Is the current palette already colourblind safe?

To determine this, I analyzed the ratio matrix of several Matplotlib colourblind safe palettes for deuteranopia and protanopia such as PuOr, Viridus, Cividis, and Bone, as well as several Matplotlib colourblind safe palettes for tritanopia such as RdGy, Greens, Reds [27].Regarding the qualitative palettes I analyzed, [43] created a 6, 8, and 10 colour palette that was more distinguishable to people with colour vision deficiencies than popular and established safe palettes such as Okabe and Ito, an eight-colour colour palette that's designed to be colourblind-friendly for both red/green and blue/yellow colourblindness [44]. This was determined by measuring the similarity of colours in the lab space.

Six Colors					Eight Colors					Ten Colors				
	R	G	B	min ΔE_{cvd}		R	G	B	min ΔE_{cvd}		R	G	B	min ΔE_{cvd}
blue	87	144	252	100.0	blue	24	69	251	100.0	blue	63	144	218	100.0
orange	248	156	32	57.1	orange	255	94	2	66.9	orange	255	169	14	56.8
red	228	37	54	21.3	red	201	31	22	18.2	red	189	31	1	33.4
purple	150	74	139	21.3	purple	200	73	169	18.1	gray	148	164	162	22.3
gray	156	156	161	21.3	gray	173	173	125	18.1	purple	131	45	182	18.3
purple	122	33	221	20.5	light blue	134	200	221	18.1	brown	169	107	89	16.4
					blue	87	141	255	18.1	orange	231	99	0	16.3
					gray	101	99	100	18.1	tan	185	172	112	16.1
										gray	113	117	129	16.1
										light blue	146	218	221	16.1

Figure 5.12: 6, 8 and 10 colour palette developed by [43], image from [43]

This Work 8 $J' \in [40.2, 78.5]$			Okabe and Ito $J' \in [0.0, 90.7]$		
	min $\Delta E'$	min ΔE_{cvd}		min $\Delta E'$	min ΔE_{cvd}
	100.0	100.0		100.0	100.0
	78.0	66.9		80.8	77.2
	20.0	18.2		56.8	49.3
	20.0	18.1		31.5	13.8
	20.0	18.1		20.8	13.8
	20.0	18.1		20.8	13.1
	20.0	18.1		20.8	13.1
	20.0	18.1		20.8	**11.0**

Figure 5.13: [43]'s 8-colour palette compared with Okabe and Ito, showing the similarity of both palettes with normal vision and across deuteranopia, protanopia, and tritanopia, images from [43]

Both Okabe and Ito and the colourmaps from [43] aim to accommodate deuteranopia, protanopia, and tritanopia altogether. To increase distinguishability, I took inspiration from [43] to create two different colour palettes, one for deuteranopia and protanopia, and one for tritanopia. This allowed me to use a combination of blues which are distinguishable for deuteranopia, and protanopia, and combinations of reds and greens which are distinguishable for tritanopia.

Figure 5.14: Safe palette for deuteranopia and protanopia

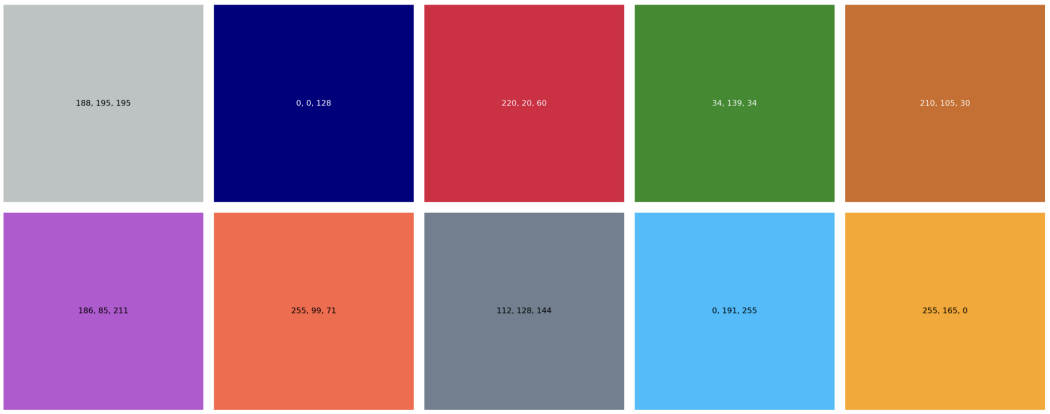


Figure 5.15: Safe palette for tritanopia

It's generally not recommended to make a palette with over 8 colours if you aim to make it colourblind-safe [45]. By separating these colour vision deficiencies, I believe I was able to stretch this to 10 colours, so therefore we have limited this implementation to a maximum of 10 qualitative colours. For deuteranopia and protanopia our colour palette achieves an average ratio matrix of 1.05, meaning when all 10 colours of the palette are used, a user with severe deuteranopia or protanopia will be only 5% less able to distinguish the colours than someone with normal vision. This percentage decreases as fewer colours are used. For tritanopia, our colour palette achieves an average ratio matrix of 1.1. These numbers align with ratio matrices achieved through analyzing the colourblind safe sequential and diverging maps from Matplotlib, where each colourmap achieved no more than a 1.1 average ratio matrix. Based on these findings, we concluded that any image with a ratio matrix of less than 1.1 is colourblind safe, and does not need recolouring. Since we have analyzed all of the possible palettes that can be mapped to an inputted image, this also completes our validation as we have pre-determined that each palette achieves less than a 1.1 average ratio matrix.

For deuteranopia and protanopia, after analyzing the ratio matrices using deltaE CMC in the lab colour space, the most optimal sequential colour palette was Cividis. The most optimal diverging colour palette was PuOr. For tritanopia, the most optimal sequential and diverging colourmaps are Reds and RdGy respectively.

Figure 5.16: Cividis and PuOr from Matplotlib, image from [27]

Figure 5.17: Reds and RdGy from Matplotlib, image from [27]

5.8 Colourmap Creation:

At this point, we've extracted our colours from the input, and we have our desired colour mapping. For continuous legends, we used polynomial interpolation in the lab colourspace to ensure a smooth gradient for our colourmap [17]. For this, I used Scipy's PchipInterpolator [46] in Python. I also tried Scipy's interp1d [46] for linear interpretation, and I didn't notice much difference compared to the polynomial interpolation, however, I opted for polynomial interpolation since it typically guarantees a smoother gradient. [47]

Initially for discrete plots, I just mapped each colour detected from the legend to a colour from the colourblind safe palette. This approach was problematic with anti-aliased plots as the edges of some data points were recoloured incorrectly. In the image below, we can see the edges of the output being discoloured. This is due to the edges of the data points being a slightly lighter colour.

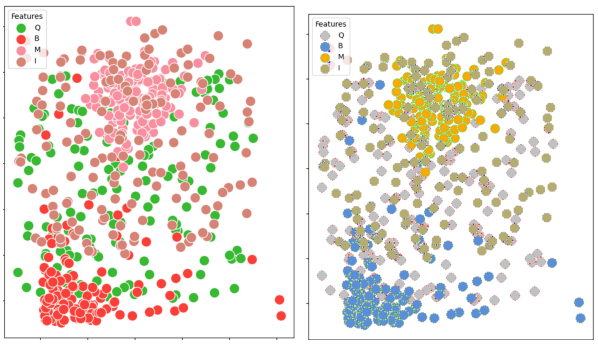


Figure 5.18: Input Vs Output: Misdetected Edges

To address this, I made some changes. From the legend in the input above in **Figure 5.18**, instead of just extracting the colours green, red, pink, and dark pink, I extracted every shade of these colours present in the legend. E.g instead of just extracting RGB(255,0,0) for the red data point, I would extract every shade of red present in each data point. Now, for each datapoint in the legend, the 'main colour' was the colour at the centre i.e. the purest shade of that colour. I interpolated lightness in the lab space for each colour in the data point around its main colour [17]. So now, instead of the colour mapping consisting of just the pure colours in the legend, it consists of every shade present, all mapping to darker/lighter colours based on their lightness i.e. if RGB(255,0,0) mapped to RGB(0,0,255), a colour close to RGB(255,0,0), RGB(255,50,50) would have initially mapped to RGB(0,0,255), now it maps to RGB(50,50,255). Below we can see the input vs output after interpolating on lightness. This increased the size of our mapping from being the length of the data points detected in the legend, to the length of every shade in each data point, which increased our accuracy and decreased the chance of miscolouring while mapping.

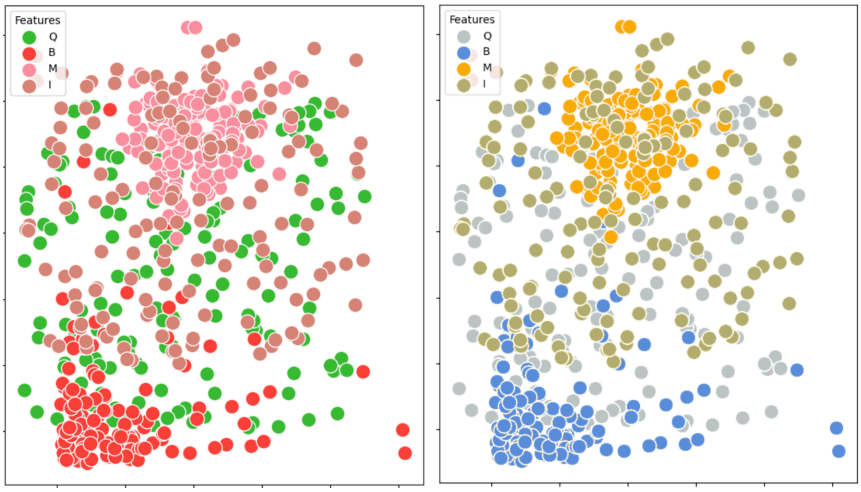


Figure 5.19: Input Vs Output After Interpolation

Prior to this solution, I experimented with Gaussian Mixture Modelling clustering, as well as k-means clustering to extract the representative colours and potentially fix the anti-aliasing issue [17]. as well as this, I used KNN with the motivation of recolouring problematic pixels by nearest neighbours. This approach wasn't ideal as it was very inefficient due to the number of times we had to check the nearest neighbours. As well as that, defining a problematic pixel wasn't always trivial. I assumed that problematic pixels would be edge pixels of data points as they are the ones most affected by anti-aliasing. I used variations of the Canny algorithm [48] to attempt to detect these edges, however, this wasn't very accurate for plots like scatter plots where we would have many data points, with lots of them overlapping.

5.9 Applying Colourmap:

To begin, we isolated coloured regions (excluding black and white). This was done by using a binary mask on a NumPy array of the image, which filtered out colours based on the colour regions detected earlier in the **Representative Colour Selection** section. The efficiency of NumPy arrays is pivotal here as they allow for fast and computationally efficient manipulation of large image data [49]. Initially, I was iterating over the pixels in the image itself using Open CV, which yielded long run times, especially for large images.

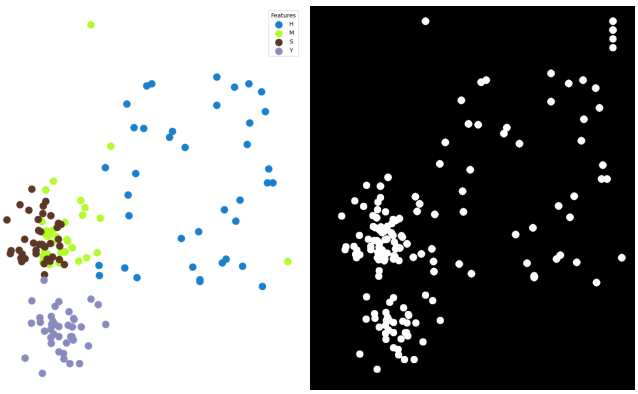


Figure 5.20: Visual representation of detected colour regions mask

After this, the unique colours were extracted from the regions detected by the mask. A k-dimensional tree (KD tree) was then set up using the keys of the colour mapping dictionary from the **Colourmap Creation** section. The colours detected from the legend of the inputted plot are the keys in this dictionary, whereas the values of the dictionary are the new colourblind safe colours to be mapped. I then converted these keys into the lab colour space before adding them to the KD tree. The KD tree is capable of fast nearest-neighbour searches, this optimized the recolouring process as we were able to quickly identify the closest colour matches [50]. To recolour, we used a two-way strategy. While iterating, if a detected colour is already present in the colour mapping, it is mapped to its new corresponding colour. If however, the colour is not in the mapping, we use the KD tree to find the nearest LAB colour match. Lastly, the new colours are assigned back to their original positions within the image. A soft Gaussian blur is then added to smoothen the image. [17].

5.10 Preserving Text:

After colour mapping, the preservation of text in a continuous plot (mainly heatmaps) was at the mercy of the colour palette. For continuous plots, we allowed for the use of black and white colours to illustrate data points. So in instances where we have black/white text on an inputted plot, if our new colour palette didn't include black/white, we would have discolouration.

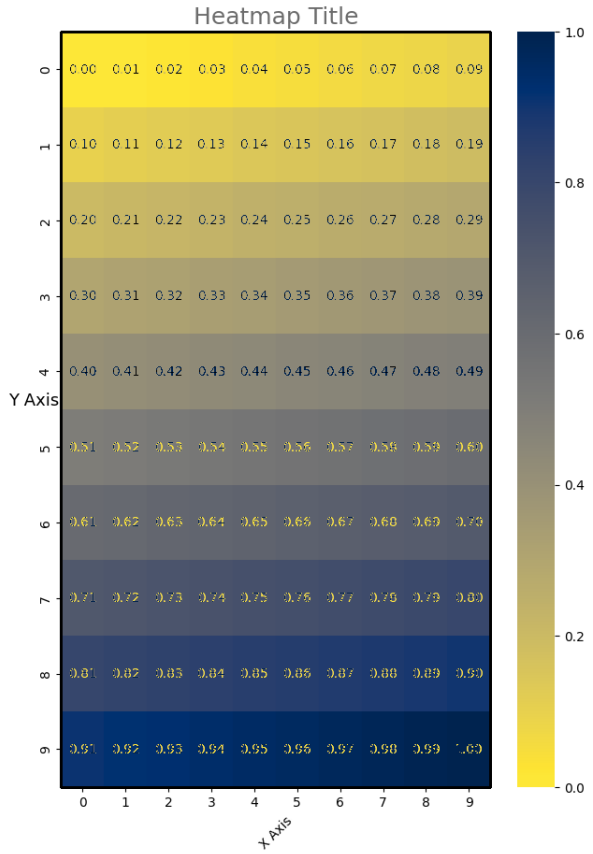


Figure 5.21: Discolouration after recolouring

To solve this, I used Optical Character Recognition (OCR) [15] to preserve text from the input image to our output. I experimented with many different OCR engines to begin with. [15] made use of Microsoft OCR, however, better accuracy has been found with Google Vision API. [51]

[52]. I also tried Pytesseract, Easy OCR, and Keras OCR with a mask to enhance text regions. In alignment with [52], these engines were less accurate when compared to Google Vision API.

Figure 5.22: Example of some text not being detected with Easy OCR

Using Google Vision API, I detected the bounding boxes of the numbers in the plot. I extracted these numbers, along with their bounding boxes. Following this, I directly removed the numbers from the plot by recolouring each column of bounding boxes by the nearest neighbour. After applying the colour map, I simply redraw the numbers back on using the same bounding box dimensions as the dimensions don't change. The surrounding area of the box is analyzed, if the bounding box is located in a dark colour region, we will redraw the text white with a black outline, in a lighter region we do the opposite. In the case where we have two numbers or more numbers detected within the same bounding box, we will evenly draw both numbers along the bounding box, which will draw them in their correct position.

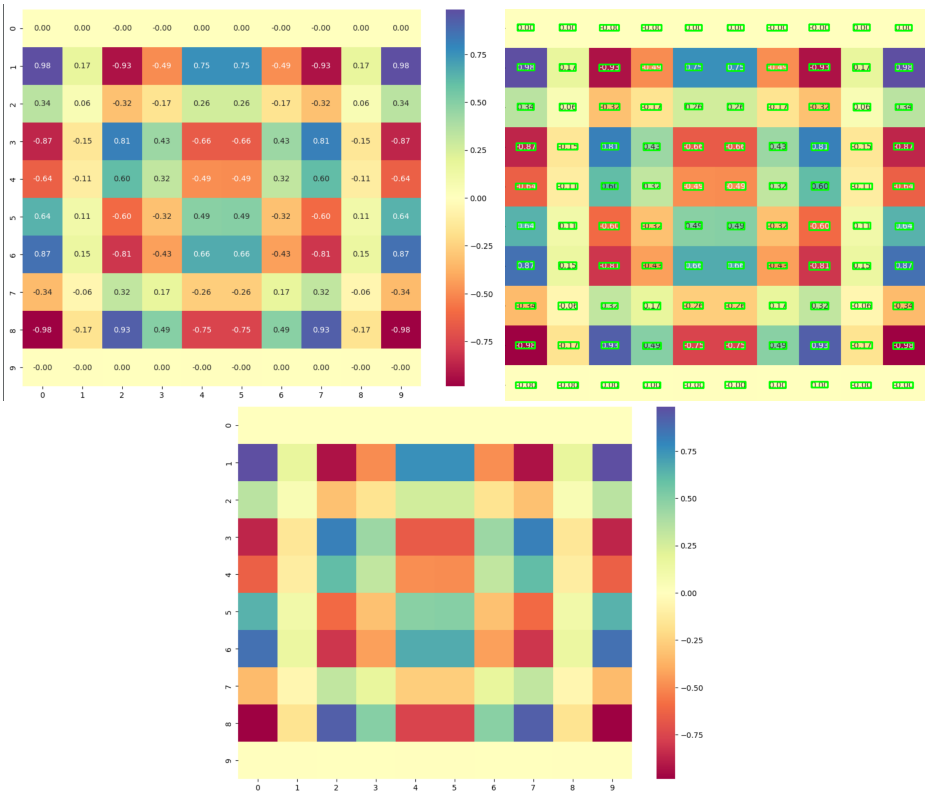


Figure 5.23: a) Input Image b) Detected Text c) After text removal

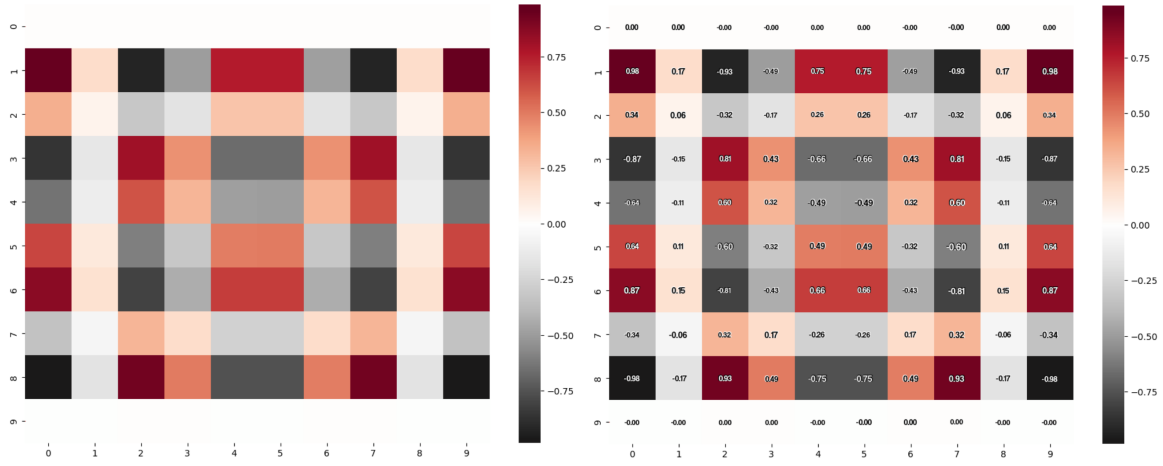


Figure 5.24: d) after recolouring d) redraw text

Limitations of Preserving Text: It is assumed that the image does not contain too much noise to the extent that it takes away from the legibility of the text [52]. We can't preserve text if Google Vision API can't detect it. Examples here include white numbers on a very light background, and when text and gridlines are the same colour. This clash makes it very difficult for our API to detect the text. If we can't detect text, or we only partially can detect text, we opt not to preserve text at all due to the aesthetic imbalance brought about by partially restored text.

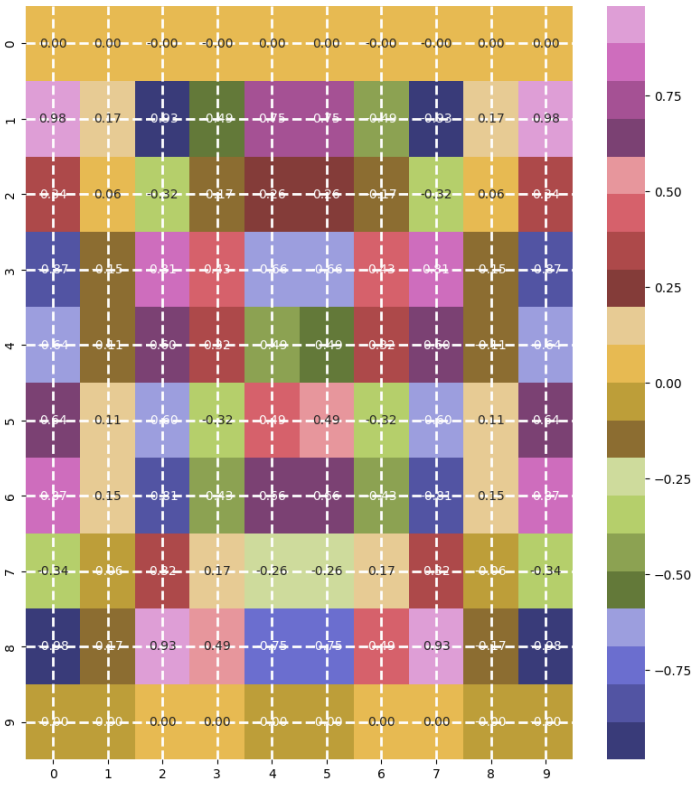


Figure 5.25: Text colour clashing with gridlines (Text won't be preserved)

5.11 Preserving Gridlines:

Similar to text, gridlines were prone to discolouration after recolouring heatmaps.

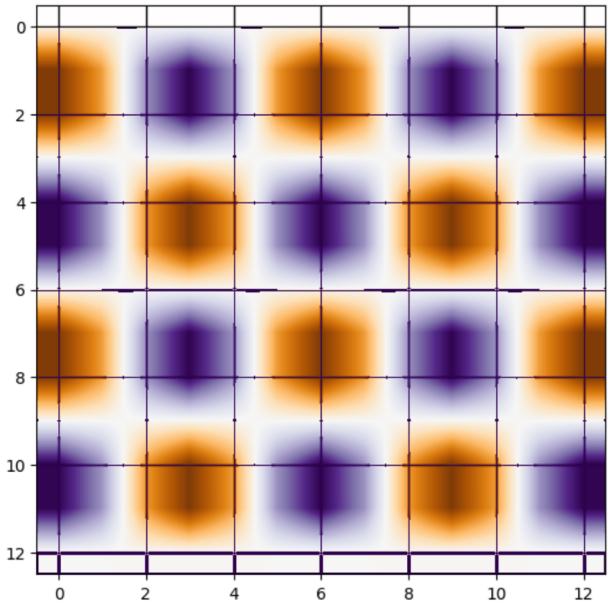


Figure 5.26: Discolouration of the grid

I originally experimented with using Hough Transform [53] to detect the gridlines, but there were many misdetections.

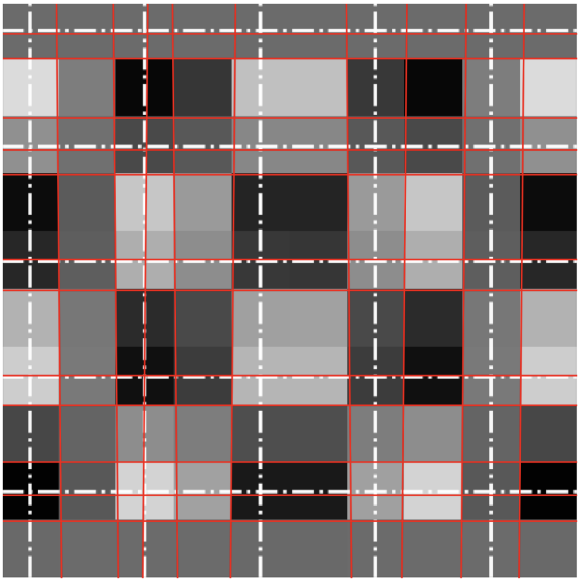


Figure 5.27: Misdetections with Hough Transform

To detect the gridlines, we must first make some assumptions. The assumptions are **a)** the axis must have tick marks **b)** the gridlines must be stemming from (coming from) the tick marks and **c)** the tick marks are facing outwards, not inwards. These three assumptions are based on the default plot setup of a typical Matplotlib plot [27]. Our object detection model is first used to detect if a grid is present, or not. If it is, we apply the following steps. Since the detected colour region for a typical heat map will be a square shape around the heatmap, we expand this bounding box on the left and the bottom (where the axis is located) until the only colours in a column for a vertical axis

are black and white, and the only colours in a row for a horizontal axis are black and white. Since we assume the tick marks face outwards, the 'white' corresponds to the background, and the black corresponds to each tick mark. From each tick mark, we draw a white line to represent the gridline. The immediate limitation here is that the pattern of a gridline isn't preserved i.e. if the gridline is dashed in the input, it will be a solid line in the output. I experimented with preserving the pattern of the gridline by only recolouring the regions along the gridline dimensions that are white or grey (assumed gridline colours) however the results were not aesthetic. I also tried incorporating interpolation in hopes of preserving the pattern, but the results were not favourable so I opted to stick with the solid line design.

Figure 5.28: Sample tick marks (black) from vertical axis, and white space. From this, we determine the position of each gridline. In this case, there would be 3 gridlines.

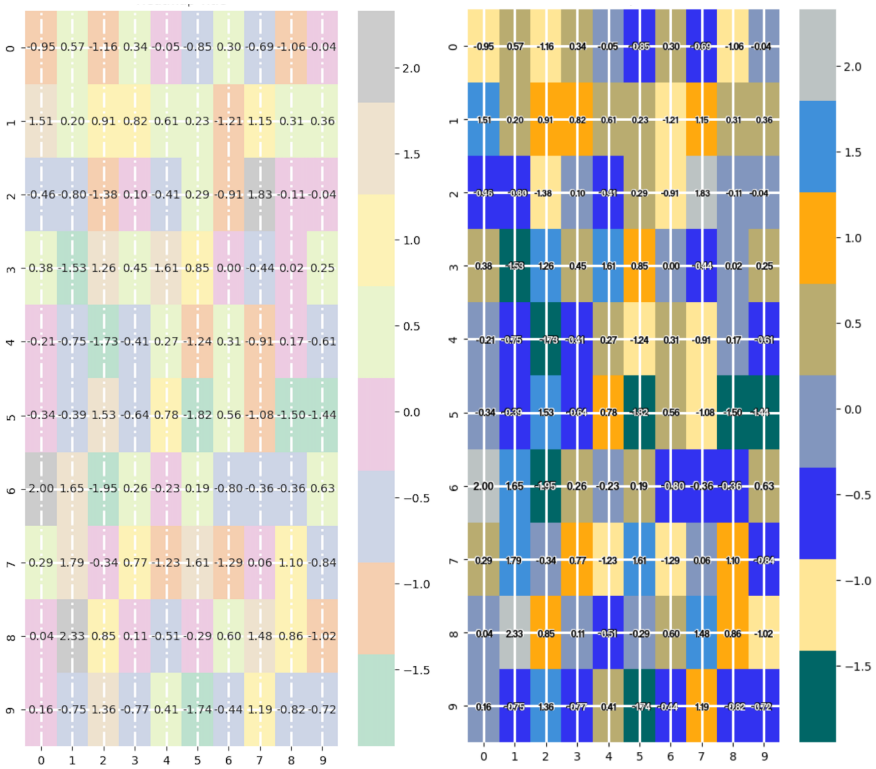


Figure 5.29: Input vs Output with gridline preservation

5.12 Console Interface

I created a simple, console interface, adding colour with the colorama library.

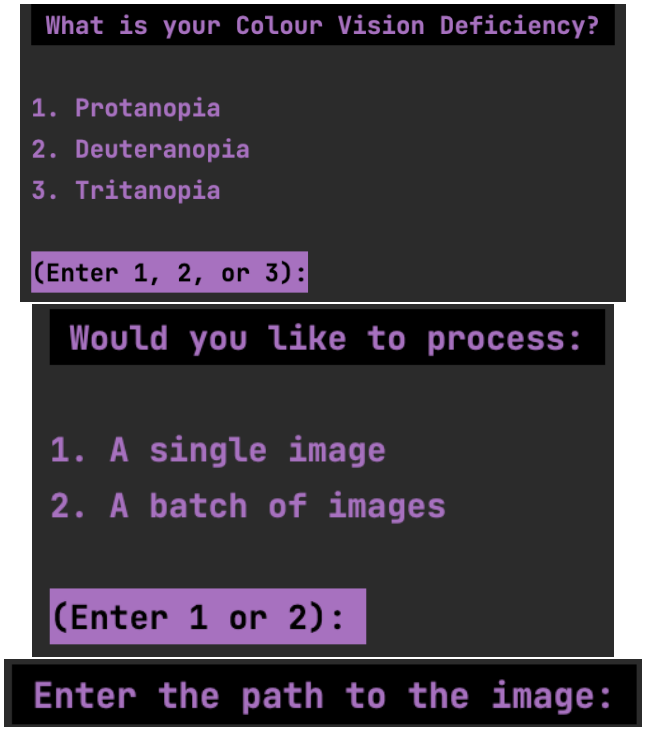


Figure 5.30: a) First, we ask for the user's CVD b) Option to upload an image or a folder of images c) enter the image path (or folder path if 'batch of images' was selected)

Figure 5.31: Console output for a successfully recoloured image

Figure 5.32: Console output for an inputted image that's already colourblind safe

5.13 Examples & Evaluation

In the below examples, we showcase various user inputs, as well as outputs where the image has been recoloured to accommodate red/green and blue/yellow colourblindness in order to evaluate and validate our implementation.

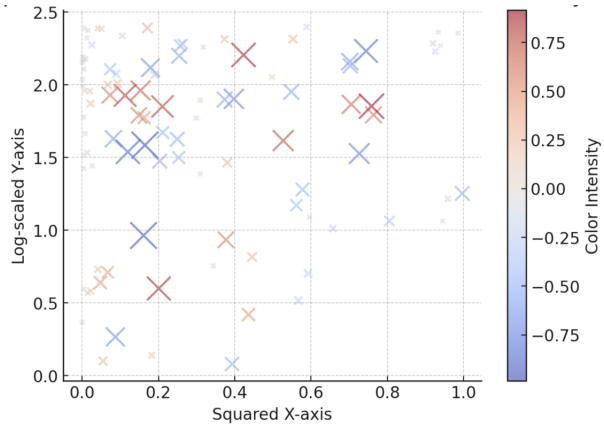


Figure 5.33: User Input

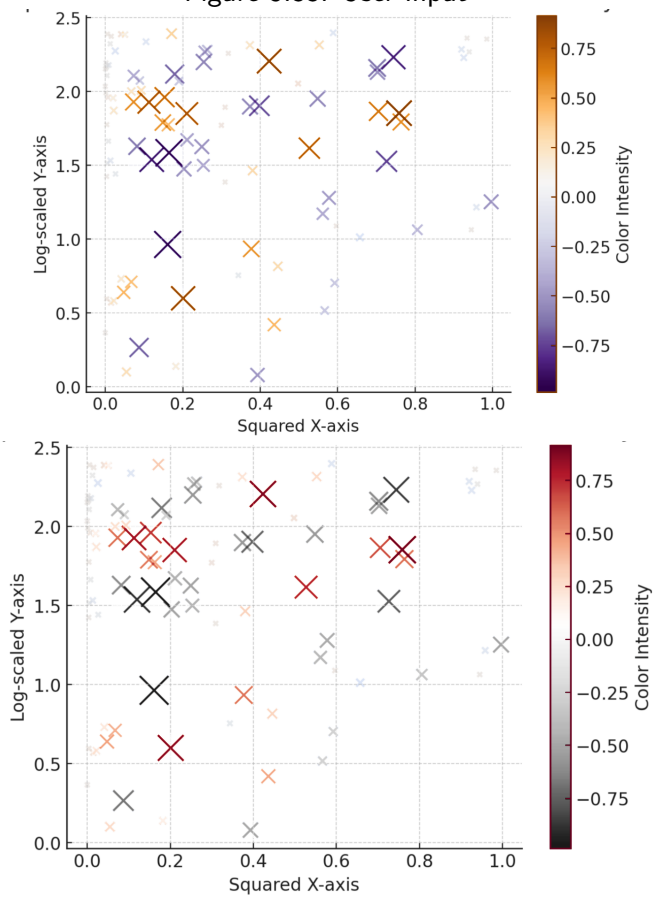


Figure 5.34: Output for protanopia/protanomaly & deuteranopia/deuteranomaly (Red/green colourblindness) vs tritanopia/tritanomaly (Blue/yellow colourblindness)

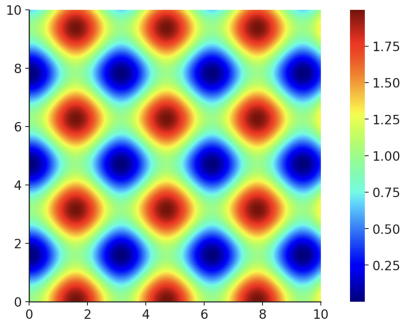


Figure 5.35: User Input

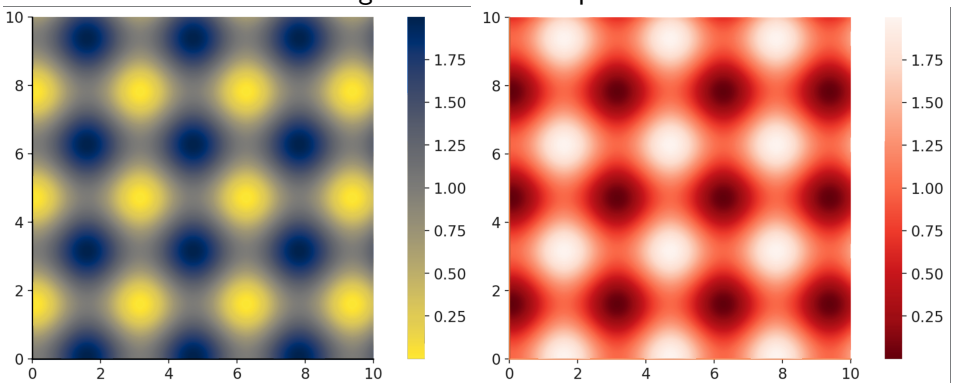
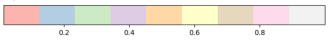


Figure 5.36: Output for Red/green colourblindness vs Blue/yellow colourblindness



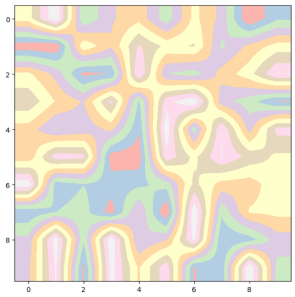


Figure 5.37: User Input

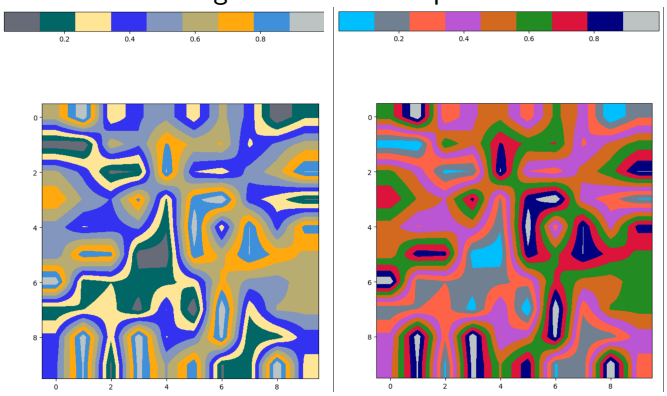


Figure 5.38: Output for Red/green colourblindness vs Blue/yellow colourblindness

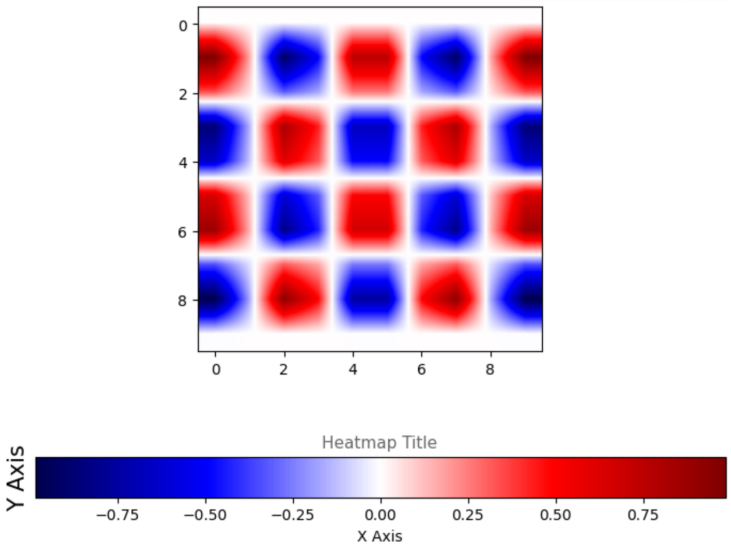


Figure 5.39: User Input

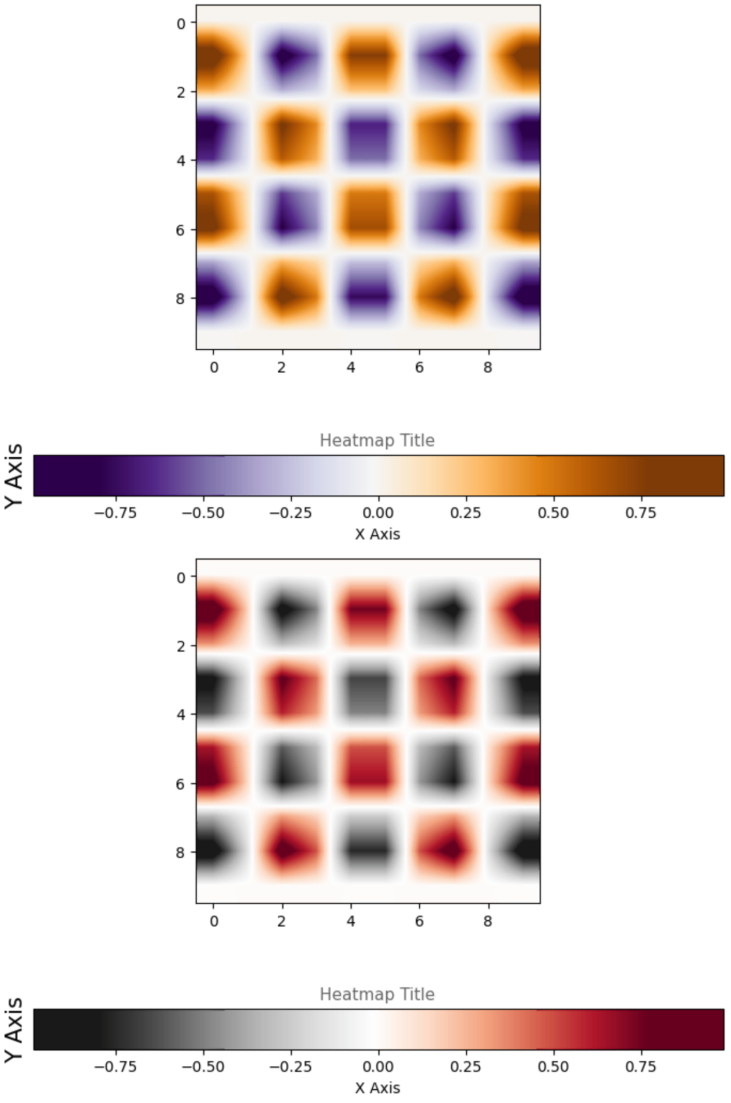


Figure 5.40: Output for Red/green colourblindness vs Blue/yellow colourblindness

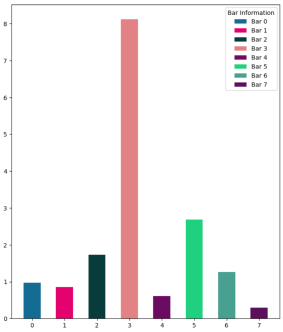


Figure 5.41: User Input

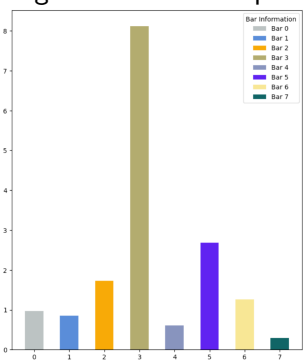


Figure 5.42: Output for Red/green colourblindness (input is safe for blue/yellow colourblindness)

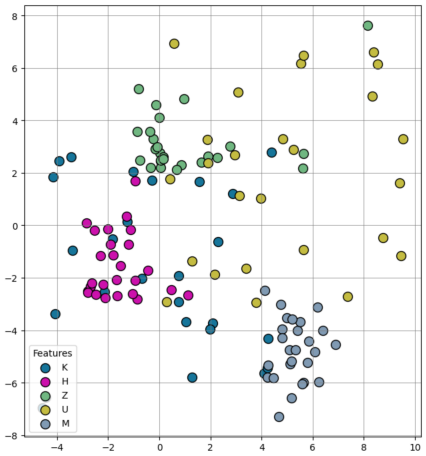


Figure 5.43: User Input

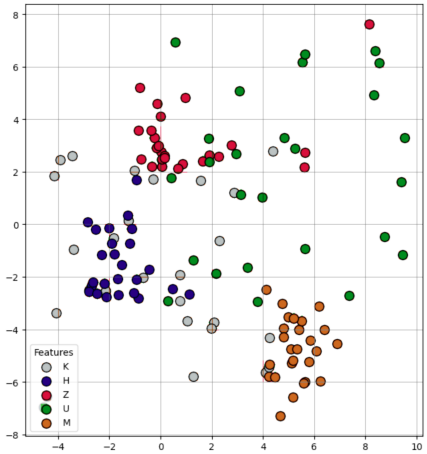


Figure 5.44: Output blue/yellow colourblindness (safe for red/green)

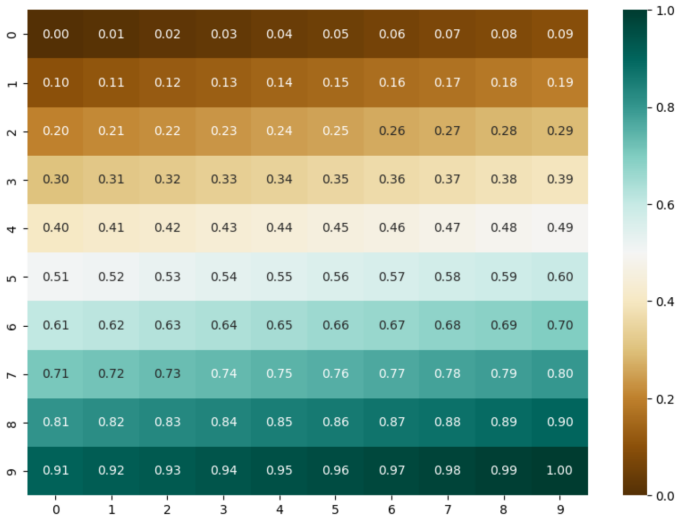


Figure 5.45: User Input

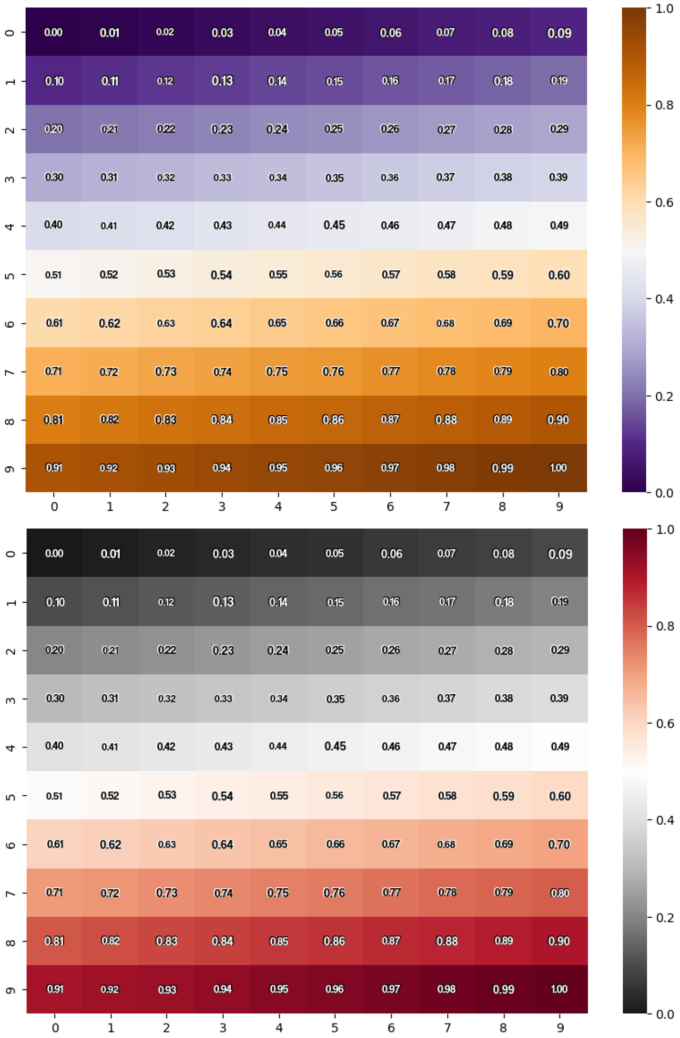


Figure 5.46: Output for red/green vs blue/yellow colourblindness

Chapter 6: Additional Testing & Evaluation

Although Matplotlib plots were the sole focus for the scope of this implementation, we also see success with ggplot.

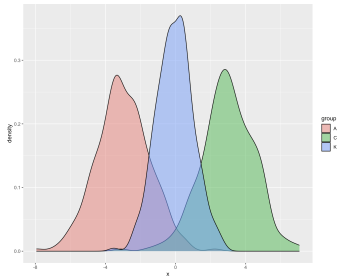


Figure 6.1: User Input

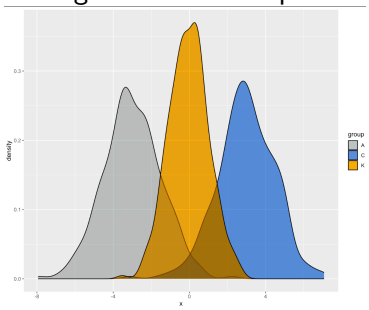


Figure 6.2: Output for Red/green colourblindness (Safe for blue/yellow)

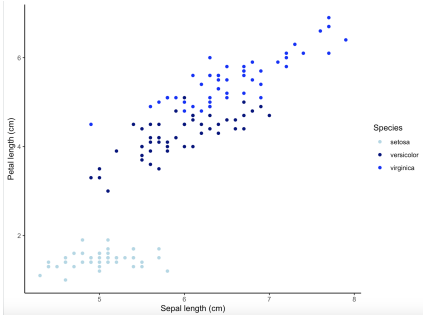


Figure 6.3: User Input

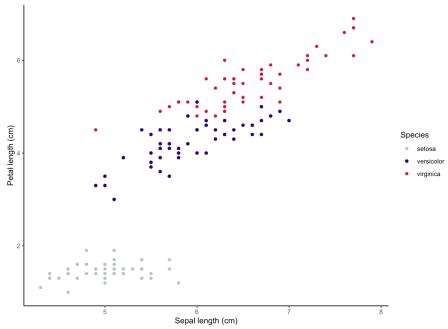


Figure 6.4: Output for blue/yellow colourblindness (Safe for red/green)

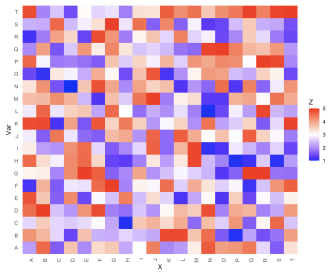


Figure 6.5: User Input

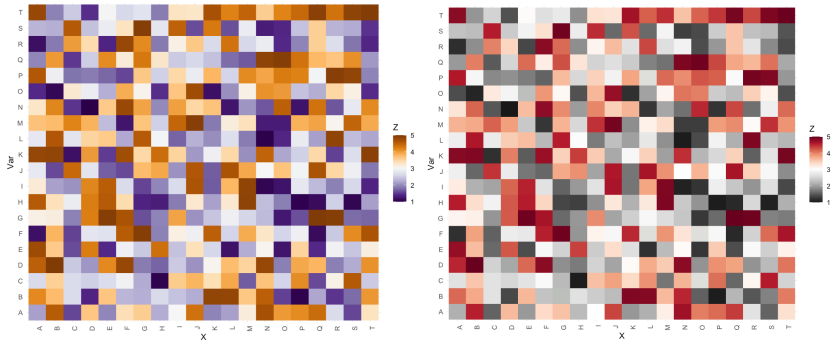


Figure 6.6: Output for red/green vs blue/yellow colourblindness

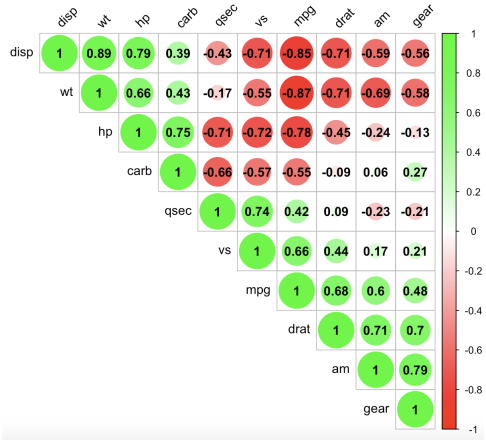


Figure 6.7: User Input

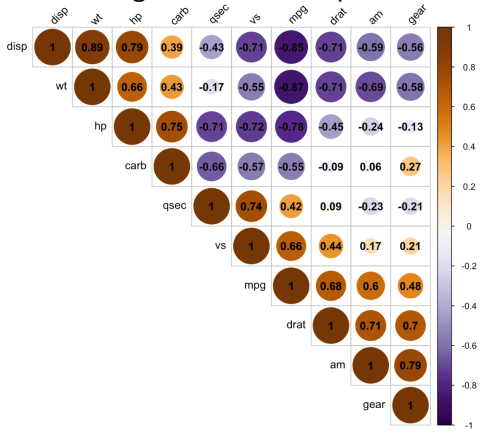


Figure 6.8: Output for red/green colourblindness (Safe for blue/yellow)

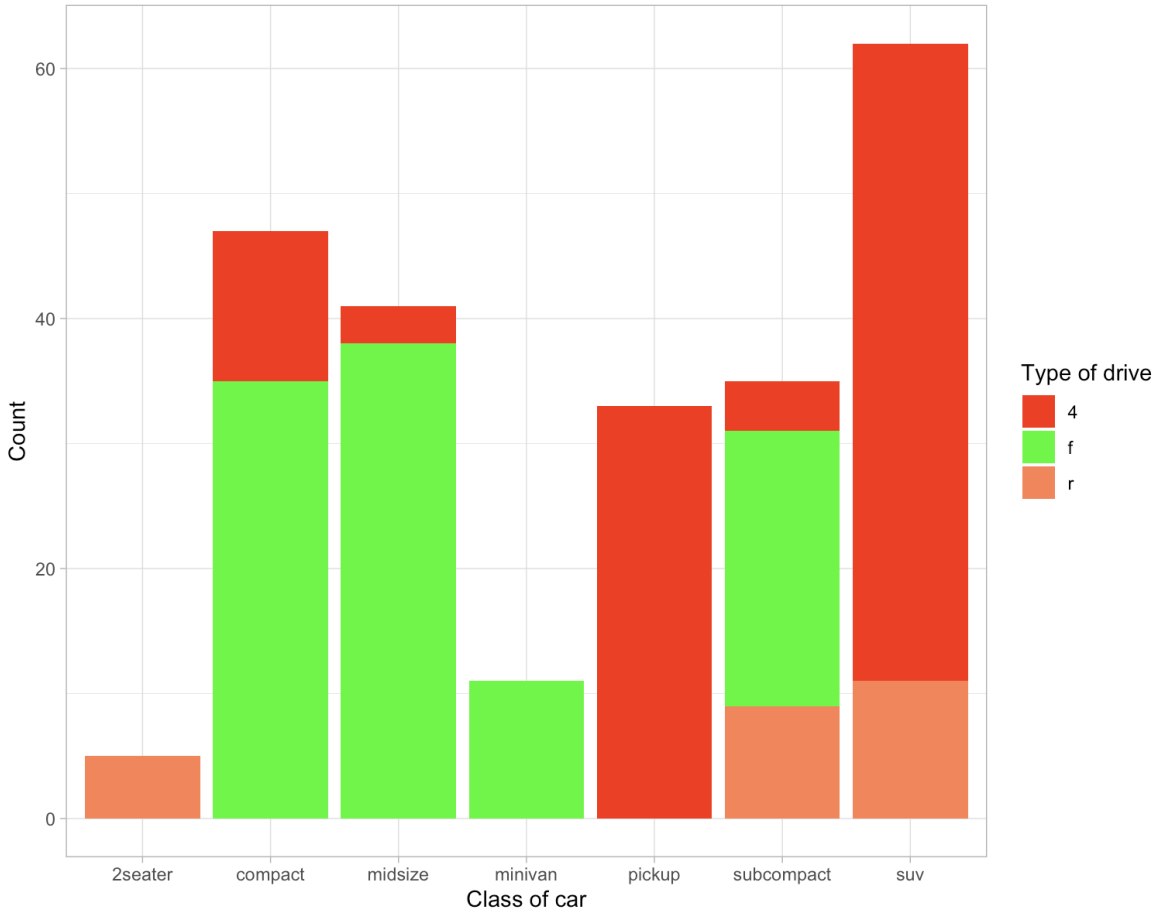


Figure 6.9: User Input

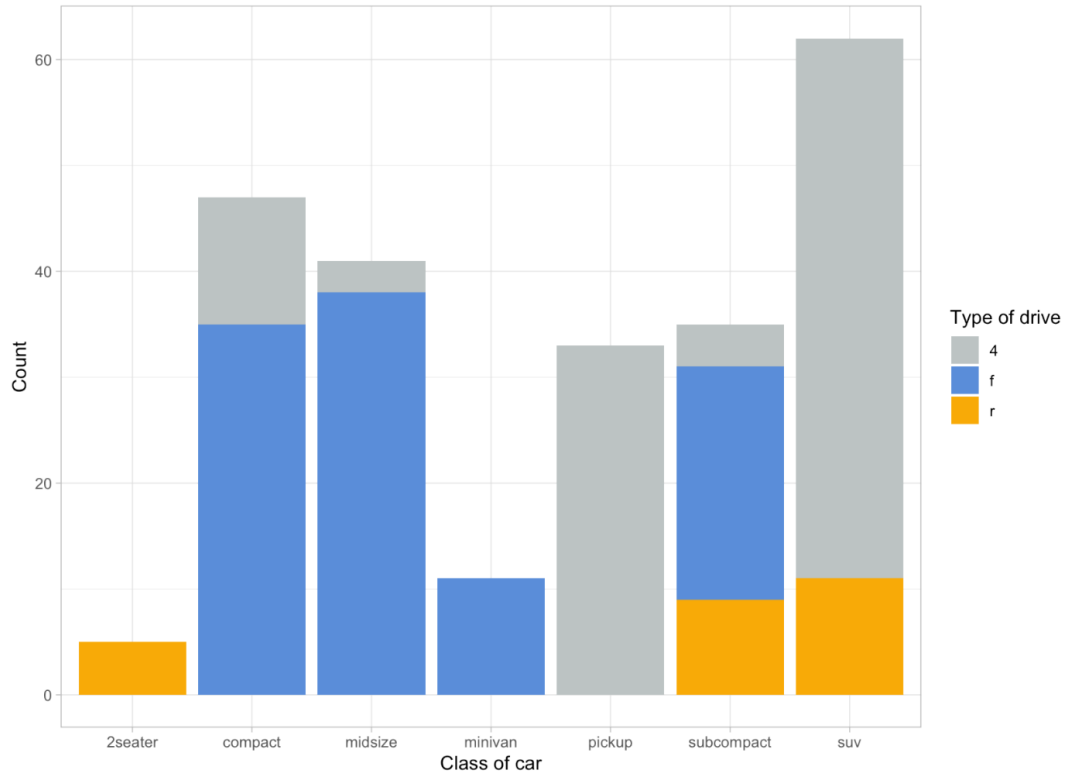


Figure 6.10: Output for red/green colourblindness (Safe for blue/yellow)

Chapter 7: Project Workplan

Below is an outline of my work plan for the next module. My first two blocks in the Gannt chart are dedicated to finishing the implementation of my project for:

- Discrete plots
- Continuous plots

On completion, I will spend the bulk of my time improving my algorithms by:

- Experimenting with different similarity metrics
- Trialing different ways of extracting representative colours from plots
- Exploring object detection models over my current rule-based approach for legend detection
- Increasing the efficiency of my program overall
- Addressing any and all bugs that arise along the way

I will then implement a simple console UI and begin to evaluate my work using test sets. The test sets will be designed to test the script's ability to detect colour blindness and correct it in various unique scenarios. I will also leave plenty of time at the end for writing up the final report (final block of the Gannt chart).

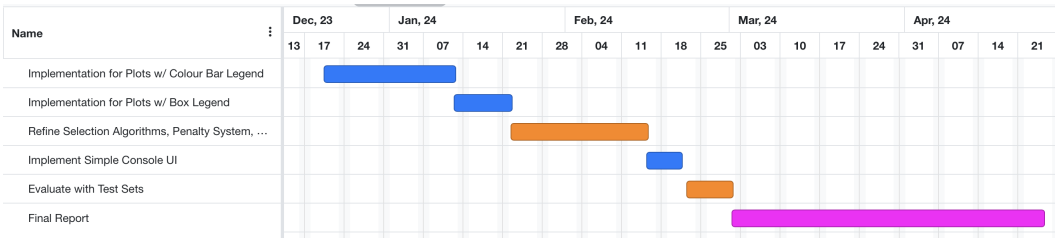


Figure 7.1: Gannt Chart of Project Workplan

Reflection: Having now completed my final year project, a reflective glance at my Gannt Chart reveals that I finished the project ahead of schedule. I invested a surplus of effort during the Christmas break before the spring trimester, which placed me in an advantageous position to finish ahead of time before my schedule got packed with assignments and exams.

Chapter 8: Summary and Conclusions

This report laid out a detailed, step-by-step approach to the recolouring of problematic images for colourblindness. Upon completion, this project implemented and realized the core and advanced goals specified in the **Project Specifications** section. Across this project, we capitalized on the tremendous advancements made in machine learning and artificial intelligence over the past decade. This was done by utilizing modern object detection models which reigned supreme over the rule-based approaches that were attempted.

In conclusion, the groundwork laid by this study, together with all other studies in this field, paves the way for subsequent research, with the potential to make a substantial impact on the inclusivity of visual information presentation in the digital era.

8.1 Limitations & Future Works

The results of this implementation have been exceptionally rewarding. Not only did we meet our core and advanced goals, but we also exceeded expectations by integrating this implementation with plotting libraries like ggplot, and expanding it to work with all discrete and continuous plots with a legend, not just bar charts and heatmaps. That being said, the limitations highlighted below pave an avenue for future work to build upon these findings and expand the limits of the current implementation.

Implementation Limited to White Background: We intentionally narrowed down the focus to Matplotlib-generated images with white backgrounds. This decision was based on the prevalence of these specifications in scholarly publications. That being said, this narrowed-down approach opens up an avenue for future work to broaden the range of accessible images for those with colourblindness, thus, creating a more inclusive digital environment. The background colour is likely the most dominant colour across the image, this logic can be used to identify the background colour for future works.

Gridline Patterns Were Not Preserved: Regardless of whether the input image had dashed, dotted, or continuous gridlines, the output converted all gridlines to a continuous pattern. The intention here was to employ the object detection model to detect the pattern of the gridlines, instead of just the presence of gridlines in general. That way we could preserve and redraw each gridline using the pattern from the input plot. However this would require the annotation of hundreds, and potentially thousands of images in order for the model to be accurate, and due to the time constraints of this project, this was not feasible.

Universal Colourblind Safety: I interpreted the implementation of this project as one to be used for the personal use of the target user. This way, the user can select their CVD, and they will be outputted a colourblind-friendly image for that CVD. Although not a limitation, this could be extended to accommodate multiple/all CVDs. As mentioned in the **Outline of Approach** section, I didn't do this as it would inherently make plots less distinguishable when compared to just focusing on CVD as we don't have an infinite spectrum of colours to choose from. The more CVDs we aim to accommodate, the more colour selection restrictions we have.

Expand Plotting Libraries: For conciseness, We centred the focus of our project on Matplotlib,

the most popular plotting library, with extended testing using ggplot to show the diversity of implementation. By adding more plots from different plotting libraries to the training set of the object detection model, future implementations can expand this to a multitude of plotting libraries by allowing the model to detect legends and heatmaps of varying formats from different libraries. The current training set is only comprised of Matplotlib images, however, we see success with ggplot due to the two libraries sharing many characteristics.

Improve Detection Accuracy: Similarly to the previous point, the legend and heatmap detection accuracy can be increased by adding more images to the training set.

Acknowledgements

This work was supported and guided by my supervisor, Dr. Colm Ryan.

Bibliography

1. Simmons, R. *Use of Color in Data Visualization* (Earth Observatory, NASA, n.d.).
2. Brewer, C. A. *Creating Color-Blind Accessible Figures* (San Diego State University, 1994).
3. Stone, M. *Choosing Colors for Data Visualization* (Perceptual Edge, 2006).
4. *Types of Color Vision Deficiency* (Nation Eye Institute, 2023).
5. Tina. *How Common Is Color Blindness:Facts And Solutions* (COVISN, 2022).
6. Duli, S. *Data Visualizations in Python with Matplotlib* (University "Luigj Gurakuqi", n.d.).
7. Tuychiev, B. *Mastering data visualization in Python with Matplotlib* (Log Rocket, 2021).
8. *Color Blindness by Inheritance* (Colour Blindness, n.d.).
9. *Setting plot background colors* (Packt, n.d.).
10. Hoffman, T. *Usage statistics of image file formats on websites in 2023* (Scanse, 2023).
11. Zelazko, A. *RGB colour model* (Britannica, n.d.).
12. Smith, E. *Evaluating Display Color Capability* (SID, 2020).
13. Vojtěch Vidra, O. P. *LCH is the best color space!* (Atmos, 2022).
14. *ggplot2* https://ggplot2.tidyverse.org/ (ggplot2, n.d.).
15. Poco, J. & Heer, J. *Reverse-Engineering Visualizations: Recovering Visual Encodings from Chart Images* (University of Washington, 2017).
16. Aisch, G. *I wrote some code that automatically checks visualizations for non-colorblind safe colors. Here's how it works* (vis4, 2018).
17. Choudary, S. A. H. A. *Live Video and Image Recolouring for Colour Vision Deficient Patients* (University of Windsor, 2021).
18. Et al., G. E. T. *A Novel Approach to Image Recoloring for Color Vision Deficiency* (ASCB, 2021).
19. Et al., Z. Z. *Personalized Image Recoloring for Color Vision Deficiency Compensation* (IEEE, 2012).
20. Et al., H. C. *Palette-based Photo Recoloring* (PIXL, 2015).
21. Et al., J.-B. H. *Image recolorization for the colorblind* (IEEE, 2010).
22. Elrefaei., L. A. *Smartphone Based Image Color Correction for Color Blindness* (iJIM, 2018).
23. Et al., J. P. *Extracting and Retargeting Color Mappings from Bitmap Images of Visualizations* (IEEE, 2018).
24. Nathaniel Pasmanter, S. M. *Physiology, Color Perception* (National Library of Medicine, 2022).
25. Rajan, S. R. K. V. *Avoiding inferior clusterings with misspecified Gaussian mixture models* (Nature, 2023).
26. Fabio Crameri Grace E. Shephard, P. J. H. *The misuse of colour in science communication* (Nature, 2020).
27. *Matplotlib* https://matplotlib.org/ (Matplotlib).
28. Et al., A. H. N. H. *Disease Detection of Solanaceous Crops Using Deep Learning for Robot Vision* (Journal of Robotics and Control, 2022).

29. *Stack Overflow* www.stackoverflow.com (Stack Overflow, n.d.).
30. Et al., J. T. *A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS* (MDPI, 2023).
31. Et al., S. S. *Comparative analysis of deep learning image detection algorithms* (Springer Open, 2021).
32. Alan M., D. F. T. *Visualisation in Modern Cartography* (Pergamon, 1994).
33. Et al., C. O. *An Analysis of Extended and Dilated Filters in Sharpening Algorithms* (IEEE, 2021).
34. Luong, Q.-T. *Handbook Of Pattern Recognition And Computer Vision* (World Scientific Publishing Company, 1993).
35. *Lab Colour Space and Delta E Measurements* (Lumen Learning, n.d.).
36. Et al., H. M. *A Preliminary Comparison of CIEColor Differences to Textile Color Acceptability Using Average Observers* (Wiley, 2005).
37. D Heggie, R. H. W. & Luo, M. R. *Lab Colour Space and Delta E Measurements* (Wiley, 1996).
38. *Color Oracle* www.colororacle.org (Color Oracle, n.d.).
39. *Vischeck* www.vischeck.com (Vischeck, n.d.).
40. *Coblis* www.color-blindness.com/coblis-color-blindness-simulator/ (Coblis, n.d.).
41. *Review of Open Source Color Blindness Simulations* (Dalton Lens, 2021).
42. Et al., B. *Computerized simulation of color appearance for dichromats* (Optica, 1997).
43. Petroff, M. A. *Accessible Color Sequences for Data Visualization* (Arxiv, 2021).
44. Houston, N. *Creating Color-Blind Accessible Figures* (Reed, 2015).
45. Goedhart, J. *Data Visualization with Flying Colors* (The Node, 2019).
46. *Scipy* https://docs.scipy.org/ (Scipy, n.d.).
47. K, J. *Polynomial Interpolation* (Data Science, 2021).
48. Li Xuan, Z. H. *An Improved Canny Edge Detection Algorithm* (IEEE, 2018).
49. McBride, M. *Numpy efficiency* (PythonInformer, 2022).
50. Wheeless, B. *A look into K-Dimensional Trees* (smucs, 2021).
51. Et al., Ö. B. *Systems Modelling and Management* (First International Conference, 2020).
52. Et al., D. V. *Cloud based Text extraction using Google Cloud Vison for Visually Impaired applications* (IEEE, 2011).
53. Et al., N. A. *Line detection in images through regularized hough transform* (IEEE, 2008).