Aiden Molyneaux - 101162579
Patrick Kye Foley - 101162436
Dr. Matthew Holden
COMP3106
December 8th, 2023

Project Report

Front Matter

- Title of project: Aiden and Pat's BIG Accessibility Gauger (The BAG for short)
- Course code: COMP3106
- Names and student numbers of all group members:
    - Aiden Molyneaux, 101162579
    - Patrick Kye Foley, 101162436

Statement of Contributions

1. Did each group member make significant contributions?

   Yes, both of us made significant time contributions as well as significant code and documentation contributions - Aiden & Patrick

2. Did each group member make approximately equal contributions?

   Yes, we both agree that approximately equal contributions were made - Aiden & Patrick

3. Exactly which aspects of the assignment did each student complete?

   Aiden:
   - Heavily contributed to feature extraction methods
   - Heavily contributed to model selection, training, and testing functionality
   - Heavily contributed to data structure selection

   Patrick:
   - Heavily contributed to dataset research and creation
   - Heavily contributed to external programs such as the dataset creator and GUI
   - Heavily contributed to colorblind functionality

   Aiden & Patrick:
   - Equal contribution to Project Proposal, Demonstration, and Report
   - Equal contribution to all other aspects of the project

Introduction

1. Background and motivation for the project:

All images fall on an accessibility spectrum, ranging from inaccessible to accessible. Image accessibility in this case being the measure of how groups of people with differing vision health and vision disabilities can see and interpret the content of an image. This type of accessibility is becoming of increased importance, and having a program that can grade the accessibility of an image could enable communities to increase the accessibility and inclusivity of their content. In addition to an accessibility grade, our program constructs positive suggestions to return to the users such that they may know how to improve the accessibility of their image.

2. Related prior work:

We researched the internet prior to developing our program, and did not find any programs that functioned similar to the idea we had in mind. We found sites that recommend how to make content accessible, and programs that assess the contrast between colors, but no program that takes an input image and outputs an accessibility score or any suggestions. *Microsoft Accessibility Checker* is a program that assesses Word, Excel, and Powerpoint documents for accessibility issues, and does provide some useful tips in response, but this tool is limited to those who own a Microsoft Office subscription, and it can only be used on content made in these Microsoft applications. Our goal was more broad than this, as we aimed to develop a simple program that returns an accessibility grade and helpful suggestions for any image with text, whether it be of a road sign or a Word document.

3. Statement of objectives
    - Our main objective was to create a program that could take advantage of artificial intelligence to assess the accessibility of an image and subsequently assign a label of ‘Accessible’ or ‘Inaccessible’ to the image.
    - We also had the objective of producing a UI for the program that was both concise and easy to use.
    - An objective we identified and fulfilled was to provide feedback on areas of accessibility that are lacking in provided images.

Methods

1. Methods from artificial intelligence used:

The artificial intelligence method that we used is called Logistic Regression. We used this method because it is a popular classification technique and is commonly used for tasks that require the ability to classify a piece of unlabeled data as either belonging to the category of label A or label B. As Logistic Regression maps an input to a binary output, it is a binary classification technique, which is exactly what we needed in our case, as we just aimed to classify an image as either ‘Accessible’ or ‘Inaccessible’.

We also used the Python ‘Tesseract’ library. Tesseract is an optical character recognition (OCR) engine developed by Google. It is typically used to extract text from paper or PDF documents, as well as images. It worked well in our project for the purpose of extracting text from images as well as the box around the text (giving coordinates of where the text-box is found on the image). This made easy extracting recognized text, as well as computing font-size and colour contrast between text and the immediate background surrounding it.

2. Dataset used:

For this project we used 3 datasets, each being used to generate a different model. The first dataset, which we referred to as the “nature” set, was a collection of 224 real-world photographs taken by University of Essex professor Simon Lucas for use in the ICDAR 2003 Robust Reading Competitions. This dataset contained images in a plethora of different settings all including text on a variety of different surfaces including street signs, historical plaques, and storefronts. The second dataset we used was a custom dataset which we referred to as the “generated” set. This dataset contained 250 procedurally generated poster-like images, generated by a small Python script we developed, that utilizes the pygame library. Each image contains three sections of text, a header, body, and footer, all of random size and color, rendered on a randomly colored background. The purpose of the generated set was to have greater control and testing capabilities when working to determine contrast feature values. Our final dataset, which we called the “hybrid” set, contained a mixture of 125 images from the “nature” set and 125 images from the “generated” set.

In lieu of a dataset containing images which we deemed inaccessible to individuals with different colorblind disorders, we developed another Python script that can transform a given image to what it would be perceived as under different colorblind afflictions. The method for these transformations were outlined in a paper by Jinmi Lee and Wellington Pinheiro dos Santos from the University of Pernambuco, Brazil. Their method for generating these colorblind emulations involved converting an image from its RGB (red, green, blue) color model to the color model LMS (longwave, middlewave, shortwave), and then performing a linear transformation between these LMS matrices and a static matrix (a different matrix for each type of colorblind affliction) (Lee & dos Santos, 2017). What follows is the colorblind simulations for a given image, preceded by the original image itself.

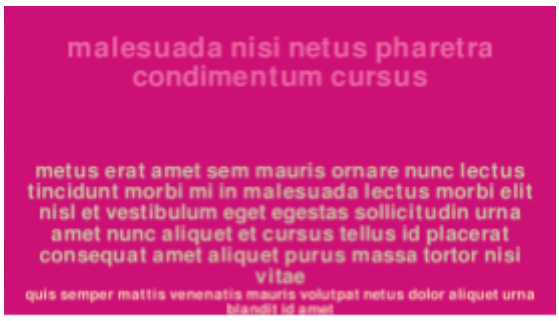

Figure 1: A generated image under normal vision

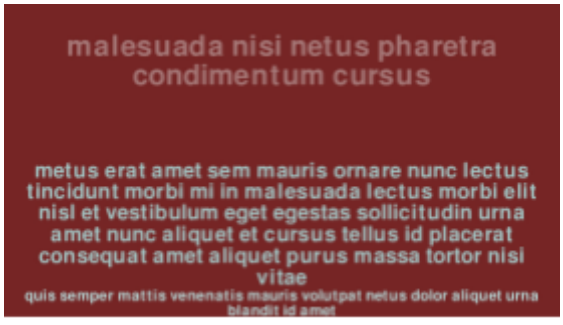

Figure 2: Protanopia rendering of Figure 1

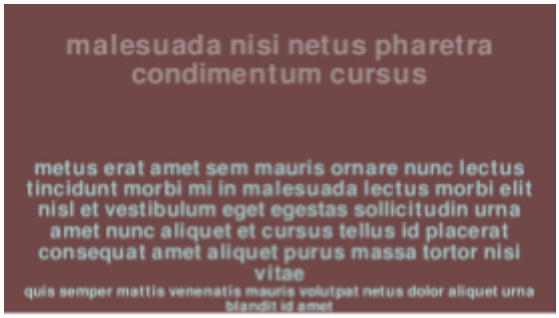

Figure 3: Deuteranopia rendering of Figure 1

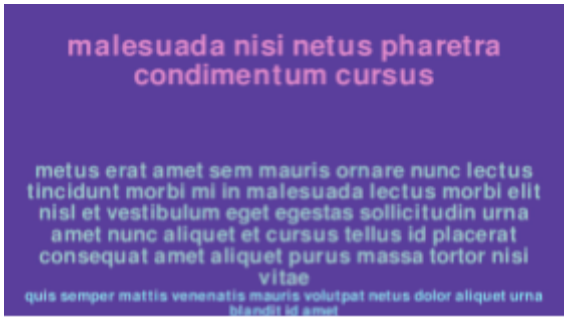

Figure 4: Tritanopia rendering of Figure 1

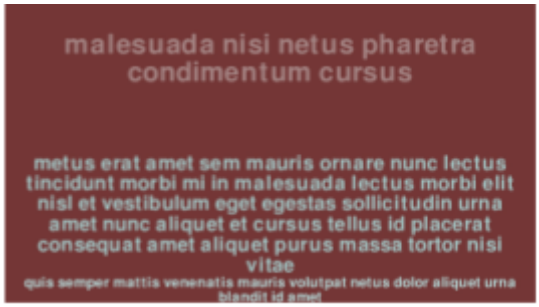

Figure 5: Hybrid protanopia-deuteranopia rendering of Figure 1

3. Validation strategy or experiments:

Each model trained was immediately tested against a distinct testing dataset of images that had not been seen by the model yet. This test set was of length 176, and contained “nature” and “generated” images.

We initially only had the “nature” dataset and one model that was trained on it, but we were unsure that this dataset was sufficient, so we went about creating another Python script that could generate a dataset of more consistently diverse images. With this new “generated” dataset we trained our second model. Then it seemed intuitive to create a hybrid dataset with images from both the “nature” and “generated” datasets, so we trained our third model on this third set. For the results of this experimentation, proceed to read the “Results” section below.

In addition to the experiments described above, we conducted another experiment after finding that many images in the “nature” dataset would not have their text properly read by Tesseract, resulting in these images not contributing to the model’s learning while training. We implemented “retry” logic such that if an image has no text found, then resize the image to either be bigger or smaller, depending on if it was already big or small. This was a shot in the dark, but we knew that Tesseract specifies that the text in the image should not be “too big or too small”. The result of this experiment was actually a 12% decrease in images with no recognized text, a non-nominal change.

Results

1. Qualitative results:

Our program classifies images as ‘Accessible’ or ‘Inaccessible’ fairly accurately. Much of the time that the user provides an image they deem accessible or inaccessible, the program returns the correct classification. On occasion however, especially with natural images like photos, Tesseract has increased difficulty identifying whole words. This can lead to some unreliability as natural images that appear to be accessible to the human eye may be labeled inaccessible because in truth, our program (or our use case of Tesseract) just can’t find the words. This is a slight fault in our implementation, as by needing Tesseract to identify the words, it is akin to first checking if the image is accessible for a machine. In our project, this is considered as one valid measure of accessibility, but overall our goal is not to identify if an image is machine accessible, but if an image is human accessible, which are obviously two very different things. However, Tesseract is very good at identifying text in digital images such as text documents and the “generated” posters in our “generated” dataset. The reliability for these types of images is much higher.

2. Quantitative results:

Our first model, which was trained on the “nature” dataset, then tested on a mixed set of “nature” and “generated” images, had an accuracy of 86%.

Our second model, which was trained on the “generated” dataset, then tested on the same mixed testing set, had an accuracy of 88%.

Our third model, which was trained on the hybrid dataset, then tested on the same mixed testing set, had an accuracy of 91%. Since this is our best model we have also computed the precision, recall, and F1-score:

- Number of correct predictions: 160
- Number of incorrect predictions: 16
- Confusion Matrix:

	Predicted Positive	Predicted Negative
Actual Positive	True Positive: 96	False Negative: 5
Actual Negative	False Positive: 11	True Negative: 64

- Precision = TP / (TP + FP)
  = 96 / (96 + 11)
  = 0.897

- Recall = TP / (TP + FN)
  = 96 / (96 + 5)
  = 0.951

- F1-score = 2 * ((Precision * Recall) / (Precision + Recall))
  = 2 * ((0.897 * 0.951) / (0.897 + 0.951))
  = 0.923

As mentioned in the “Validation strategy or experiments” section, we conducted an experiment to test if the hybrid model would be more accurate then its predecessors, and the fruit of this experiment was a model that performed with 91% accuracy.

In terms of time to train and time to test:

- The “nature” model trained significantly quicker, as there are typically less words to be found in each image, as well as many images are immediately skipped because no words were found in the image. Although, in cases where no words are found, the program runs the “retry” logic mentioned previously which does contribute to a higher time complexity. This model took about 15 minutes to train.
- The “generated” model trained the slowest, as there are consistently many more words to be found on these poster-like images. There are also 25 more images in this dataset. Remember that for each word found, a normal contrast and a contrast for each colorblind affliction is computed, which is a heavy computation, contributing heavily to the time complexity. This model took about 45 minutes to train.

Discussion

1. Limitations of the work and directions for future work:

Throughout the implementation of this project we identified a number of limitations with our approach to gauging the accessibility of images. For one, colorblind considerations are not taken into account when determining the accessibility scores of an image. While we think this was mitigated through the inclusion of these considerations as suggestions to the user, we would have liked it to be optional for the user to indicate whether these considerations should cause the image to get a failing grade on a per-image basis. Another limitation of The BAG was that accessibility scores are gauged using average feature values across all text located on an image. This means that scores can be skewed easily, significantly negating the impact of very small or low-contrast text, should they be low in number compared to other more-accessible text. To mitigate this we included flags to be raised if any text was found in the image was of extremely low font size or contrast for each colorblind rendering, such that these warnings could be communicated to the user in the GUI. Additionally, we found that by the time that all features were implemented The BAG ran slower than we would have liked. This is especially the case for images with large amounts of text on them, as each and every word is evaluated to its full extent. This could have been mitigated to a degree by skipping the computation for a few words if they appear to be similar to, or close to, another word which has already been evaluated. In the end however, we decided that it would be best to fully evaluate every word located on the image, especially with Tesseract's occasional inability to locate all words.

Throughout implementing The BAG, a few future features jumped out at us which we didn’t have time to consider for this project. Firstly, we would have liked to be able to gauge the blurriness of text as a feature value, as it is entirely possible to have apparently large text that is still blurry, and therefore difficult to read. A couple of attempts were made at this, but ultimately

dismissed when resulting scores on images didn't correspond to how we would expect them to be, and at times seemed to be nearly random for "nature" images. Most importantly, providing suggestions to the user is nice, but directly altering an image in order to increase its accessibility would have been even better. We consider this to be within the realm of possibility for correcting low-contrasting text, as it would be a simple operation to find and replace all pixels of a certain color with a higher-contrasting alternative. However, increasing font size in such a way that it wouldn't overlap or interfere with surrounding elements would be a more difficult operation, albeit not impossible.

2. Implications of the work:

While we do not believe that The BAG is a production-level application of artificial intelligence, we certainly see it having a positive impact in its practical applications, as well as its ethical implications. A system similar to The BAG would be incredibly useful in a handful of real world scenarios. For instance, the goal of a graphic designer is to develop ocular goods that are pleasing to the eye of the masses, and therefore needs to consider their product through the eyes of each person who might see it. Considering how an image would appear through the lens of someone with colorblind afflictions would be an incredibly hard task for even someone with lots of design experience, and is a good opportunity for the aid of computer assistance. With this, The BAG has a positive impact for consideration and inclusion of people with visual disabilities.

As with the development of any artificial intelligence system, there are risks associated with the use of systems like The BAG. All systems are not foolproof, and human users should ultimately place the final authority in themselves when consulting these systems for advice. In addition to this, there are a plethora of other factors that contribute to accessibility, with more and more research and understanding being formed surrounding these issues constantly. The BAG offers a simple way to approach a couple of these understandings as we see them now, and hopefully increase the quality of support that we offer to people who are afflicted.

References

Lee, J., & dos Santos, W. P. (2017). An Adaptive Fuzzy-Based System to Simulate, Quantify and Compensate Color Blindness. University of Pernambuco.

Lucas, S. (2011, June 28). ICDAR 2003 robust reading competitions. TC11. http://www.iapr-tc11.org/mediawiki/index.php/ICDAR_2003_Robust_Reading_Competitions

nikki2398. (2023, September 5). Extract dominant colors of an image using Python. GeeksforGeeks. https://www.geeksforgeeks.org/extract-dominant-colors-of-an-image-using-python/