TransformVector

An automated raster to vector conversion platform for GIS (Geographical Information Systems)

Damodaran Dillon Thisaru Lakshman

Informatics Institute of Technology

In Collaboration With

University of Westminster, UK

Contents

1. Solution Proposed 4
   1.1 Project Overview 4
      1.1.1 Project Background 4
      1.1.2 Problem Domain 5
      1.1.3 Problem Justification 6
      1.1.4 Research Question 6
      1.1.5 Motivation 6
2. Project Aim 7
   2.1 In-Scope 7
   2.2 Out-Scope 7
3. Overview of the Solution 8
4. Selection of Libraries 9
   4.1 Scikit-learn (Image quantization) 9
   4.2 Scikit-Image (Structural Similarity Index) 9
   4.3 Tensorflow (Keras – Image classification CNN) 9
   4.4 OpenCV 10
   4.5 Python Flask API Framework 10
5. Design 11
   5.1 Process Overview 11
   5.2 Class Diagram 12
   5.3 Activity Diagram 13
   5.4 Sequence Diagram 14
6. Implementation of Components 16
   6.1 Parameter Trainer Component 16
      6.1.1 Image Resizing 16
      6.1.2 Image Colour Quantization 16
      6.1.3 Converting SVG into PNG 17
      6.1.4 SVG2PNG.bat 17
      6.1.5 Image Structural Similarity Index Calculator 17
      6.1.6 Variable Trainer Process 18
      6.1.7 Create Best Fit Parameters for category 20
      6.1.8 Cleanup 20

6.1.9 Full Training Process Executor 21
6.2 Image Classifier Component 21
6.2.1 Create and Prepare Training Data 21
6.2.2 Training Image Classification Model 22
6.2.3 Training and Saving Model 23
6.3 Backend API 23
6.3.1 Flask API Initialization 23
6.3.2 Flask API method signatures 24
6.3.3 Analyzer Class 24
6.3.4 Classify Image (Analyzer Class) 25
6.3.5 Colour Counter (Analyzer Class) 25
6.3.6 Determine Colour Clusters (Analyzer Class) 26
6.3.7 Get Best Parameters (Analyzer Class) 26
6.3.8 Determining Best parameters for user image (Analyzer Class) 27
6.3.9 Analyze (Analyzer Class) 29
6.3.10 Converter Class 29
6.4 Frontend 30
6.4.1 Initialization 30
7. Testing 32
7.1 Testing Criteria 32
7.2 Unit Testing 32
7.2.1 Unit Testing Tools 32
7.3 Integration Testing 33
7.4 Testing Image Classification Model Accuracy 34
7.5 Testing Image analyzing process Performance of Flask API Backend 35
7.6 Testing Image Conversion Performance of Flask API Backend 37
7.7 Testing structural similarity between input image and converted vector image. 39
References 40

1. Solution Proposed

platform that identifies the properties of a Raster image and converts it into a Vector file by using the best method of conversion using parameters which best fit the use case of the resulting vector image addressed currently in the research domain of GIS graphics

It can also be used for general purpose Raster to vector conversion

1.1 Project Overview

1.1.1 Project Background

Images

Human beings can be considered predominantly visual creatures as we use our sense of vision when available to understand our surroundings. Due to this reason we have created images to capture moments of this ever constantly moving world. Images have since then evolved to not only capturing real world moments but to digital drawings and other representations of visual still media. An image can be a single picture which represents a certain object, location or scenery. When image files are attempted to be classified, two main classifications can be identified. Vector images and Raster images. There are different pros and cons of using either a Vector image or a Raster image.

Raster Images

A raster image is built up of colour pixels which are arranged to form the necessary result image. Raster images are mostly suited for linear art images because they can better represent subtle chromatic gradients due to the fact that each pixel can change its value independently of other pixels to form the necessary image. These types of images are also called continuous tone images.

A raster image while being faster to process and display as there is no methodical processing or such involved will have a larger file size as information is electronically stored on a pixel-by-pixel basis (Winnemoeller *et al.*, 2018), the size of the image is directly proportional to the resolution of the image

Vector Images

Vector images on the other hand are created from points and lines and curves joining them. These are based off of mathematical formulas that create combinations of multiple true geometric primitives to create a final image (Seel-audom, Naiyapo and Chouvatut, 2017)..

Vector format based computer graphics tools have become very powerful tools allowing artists, designers etc. to mimic many artistic styles, exploit automated techniques, etc. and across different simulated physical media and digital media (SEVERENUK *et al.*, 2019). Similarly, in real world applications according to the factors that need to be considered, the image may be required in either Raster or in Vector format.

As stated above, as a vector file and raster file of the same image may have similar resulting image, when observed on a deeper level multiple differences can be identified between them.

1.1.2 Problem Domain

Geographic Information System (GIS) is the processes of managing, manipulating, analyzing, updating and presenting metadata according to its geographic location, to be effectively used in different aspects of life (Al-Bayari, 2018). The increasing popularity GIS technology has increased the usage of spatial data. Making maps is relatively easy even for those who do not have much cartographic training (Wong and Wu, 1996).

According to the analysis needed to be performed on a certain image obtained, the requirement for a Raster image or a Vector image may vary. There are several pros and cons when you consider each type of image. There is an old GIS adage stating that “Raster is Faster but Vector is Corrector”.(Berry, 1995).

Vector GIS results in the geometrization of the geographical world, and generalizing and reducing its theory into theories about relations between points, lines, polygons and areas. Such objects which are in a GIS can be counted, moved about, stacked, rotated labelled, cut etc. and be handled like a variety of other everyday solid objects that bear no particular relationship to geography (Couclelis, 1992).

Vector maps use simple geometric components such as points, lines and polygons in adjacencies, unions and inclusions to describe spatial information and Raster maps are based on pixel matrices and are richer and realistic than vector maps (Lin and Guo, 2011).

There are several advantages of using a Vector data format. These can be stated as the output being more aesthetically pleasing and zoom able to very close detail as it is made up of points and line segments connecting them and not using fixed number of pixels which might look pixelated and less clear when zoomed into more than its resolution allows. It also provides higher geographical accuracy due the same reason as it being not of a fixed resolution. There are other reasons as why vector images are used in GIS such as data integrity, and allowing network analysis and proximity operations as they both use vector data structures. As well as there are advantages are there are disadvantages to using vector images as well. As these images are a result of mathematical calculations it is often very processing intensive. Vector data structures are also poor performing when displaying continuous data, and needs to be generalized in some manner to display, which can result in loss of some information.

While vector data structures in GIS over determines the geographic world by forcing it into a geometric objects generalizing them, the raster data structure feigns maximal ignorance on the nature of things in the world. Yet Raster data structures provide an implicit view of the geographical world with measurable values discretized into pixel arrays (Couclelis, 1992).

Raster data can store unique values per each pixel without any generalization being required. Therefore, is a good option when continuous data is required to be displayed. Even though continuous data can be very accurately represented in a Raster image, because of the resolution. Raster graphics display devices are capable of reproducing very complex images (Sloan and Tanimoto, 1979). It struggles when representing linear features and can cause pixilation if the resolution of the image obtained is low or when zoomed to obtain a closer look. Raster datasets can also be very large file as when the resolution is increased to get a more accurate image with high detail the file size increases proportionally with it.

From the statements above, it can be identified that both these formats are equally important when considering the use of imagery in GI Systems. Therefore, there becomes necessary a method of conversion between these two data types.

1.1.3 Problem Justification

Automated conversion of engineering drawings and such similar content into Raster and vector data has been a very widely discussed topic. A critical step in this process can be considered as the conversion of these images into a vector format (Liu and Josep Lladós (Eds.), 2005). Many techniques for conversion of raster to vector have been proposed which has even led to development of commercial solutions to tackle this issue. The systems created all did provide quite acceptable results but each had their own drawbacks (Hilaire and Tombre, 2006). (Lacroix, 2009)

1.1.4 Research Question

How can the Raster to vector conversion process be improved to obtain better quality outputs by changing parameters that affect the conversion process?

1.1.5 Motivation

After researching on the basic concept of converting Raster images into Vector images for graphic design purposes, the author has come to find the importance of it but in a different domain which is in the field of GIS. Raster and Vector data structures are widely used in analysis in GIS and as both of these type of images are needed according to different situations. It has motivated the author to create this automated Raster to Vector conversion tool.

2. Project Aim

To investigate design and implement a Raster to Vector conversion platform that selects the best method of conversion using image processing techniques.

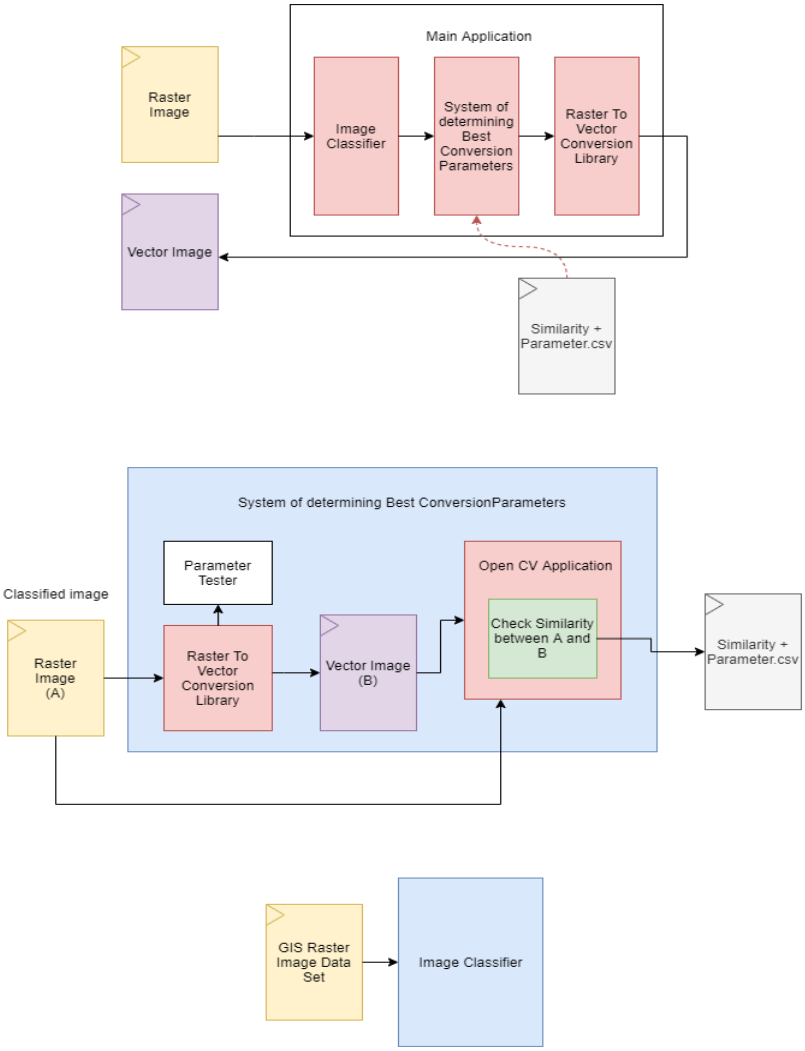
2.1 In-Scope

- Raster to Vector conversion tool is only developed geared towards GIS
- Training an image processing model to identify certain properties of images that affect conversion algorithms.
- Integrating Image processing for the identification special characteristics to identify the best conversion algorithm
- Considering of continuous tone images as well as line based images for the conversion process
- Set conversion method from either one of Accurate or Fast conversion

2.2 Out-Scope

- Conversion of Raster to Vector for other domains such as graphic design.
- OCR functionality out of image text is not considered, and will be represented in the converted images as graphical data and not textual data

3. Overview of the Solution



The basic component of the system can be divided into three main sub sections. These are the Main application which gets a raster image as an input, performs all acts of preprocessing on it and identifies the best fit conversion parameters for an output with high structural accuracy. The Image classifier that identifies what type of GIS image the main application is dealing with at the moment and the System that determines the best parameters for raster to vector conversion for a certain classification of images according to the structural accuracy comparison between input and output image.

The core functionality of all three of these components are implemented using the programming language Python. There are several libraries which are being employed to carry out specific tasks in each of these components and some are recurring libraries such as the python machine learning library that handles structural image comparison and the basic image processing libraries that are being employed

for reading images and conversion of them into matrices for further operation to be carried out on them.

The User Interface of the Raster to Vector converter is implemented using Electron JS. A UI frame work that employs the power of HTML, CSS and JS to create cross platform desktop applications top of Node.js and Chromium. This will be further elaborated as this chapter continues. Parameters calculated are stored for future access for processing in csv format and are accessed and manipulated using the core language of the application which is Python. The Backend of this application is implemented using python Flask API and will be further elaborated and justified as to its use in the sections below.

4. Selection of Libraries

4.1 Scikit-learn (Image quantization)

Scikit-learn is a python library that provides unsupervised machine learning algorithms. It is a library that is built upon other common python libraries such as matplotlib, NumPy and Pandas. Out of the many algorithms that Scikit learn provides, this project will be utilizing the K-Means Algorithm which is a Clustering algorithm which will be used to quantize the colours found in an image to ease in raster to vector conversion and execute other functionalities which ease processing overheads and therefore, reduce processing time of the system.

4.2 Scikit-Image (Structural Similarity Index)

Scikit-Image is a python Image processing library that is built on other common python libraries such as numPy and sciPy. We will be using the set of methods that are created for comparing structural similarity and obtaining and index between two images of the same dimensions for the purpose of this project.

4.3 Tensorflow (Keras – Image classification CNN)

Tensorflow is an open source end to end machine learning platform. It supports multiple programming languages including python which is the core programming language used in this project. Its API is also available in multiple versions; this project will be using version r2.1 (stable) to implement the image classification training model that will be deployed using Tensorflow. Tensorflow allows multiple levels of abstraction. Due to this, the project will be utilizing the high level Keras API provided by Tensorflow. The Keras API will be utilized to implement a simple Image processing model using the layers provided to create a Convoluted Neural Network (CNN) to classify GIS related image according to the training data sets obtained.

4.4 OpenCV

Open Computer Vision Library (OpenCV) is another Image processing library that supports the programming language python among other programming languages that is being used for image processing in this project.

4.5 Python Flask API Framework

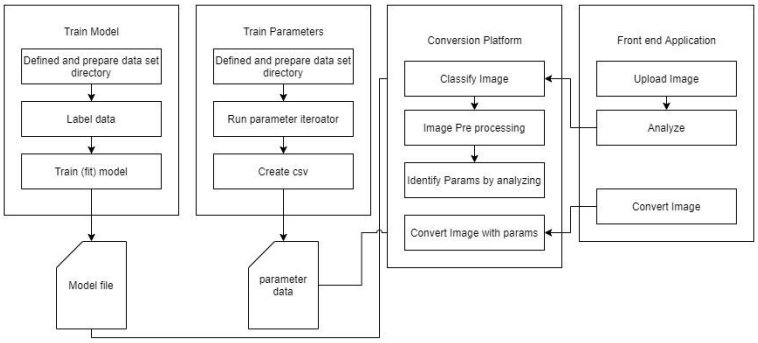
For the Application Programming Interface that is used to expose the backend functionality to the frontend desktop application, FLASK a popular python based API will be used. The advantages of using Flask as the selected API for the project are as follows.

When compared to popular web frameworks such as Django, Rails, Play and Laravel etc. Flask can be considered a significantly minimal web framework. This allow the programmer to implement flask applications in any design structure. In this case OOP based. As OOAD is selected as the preferred Software Design methodology as specified in the third chapter of this document. Flask has been chosen the Framework to be used to implement the backend exposing API in this project.

5. Design

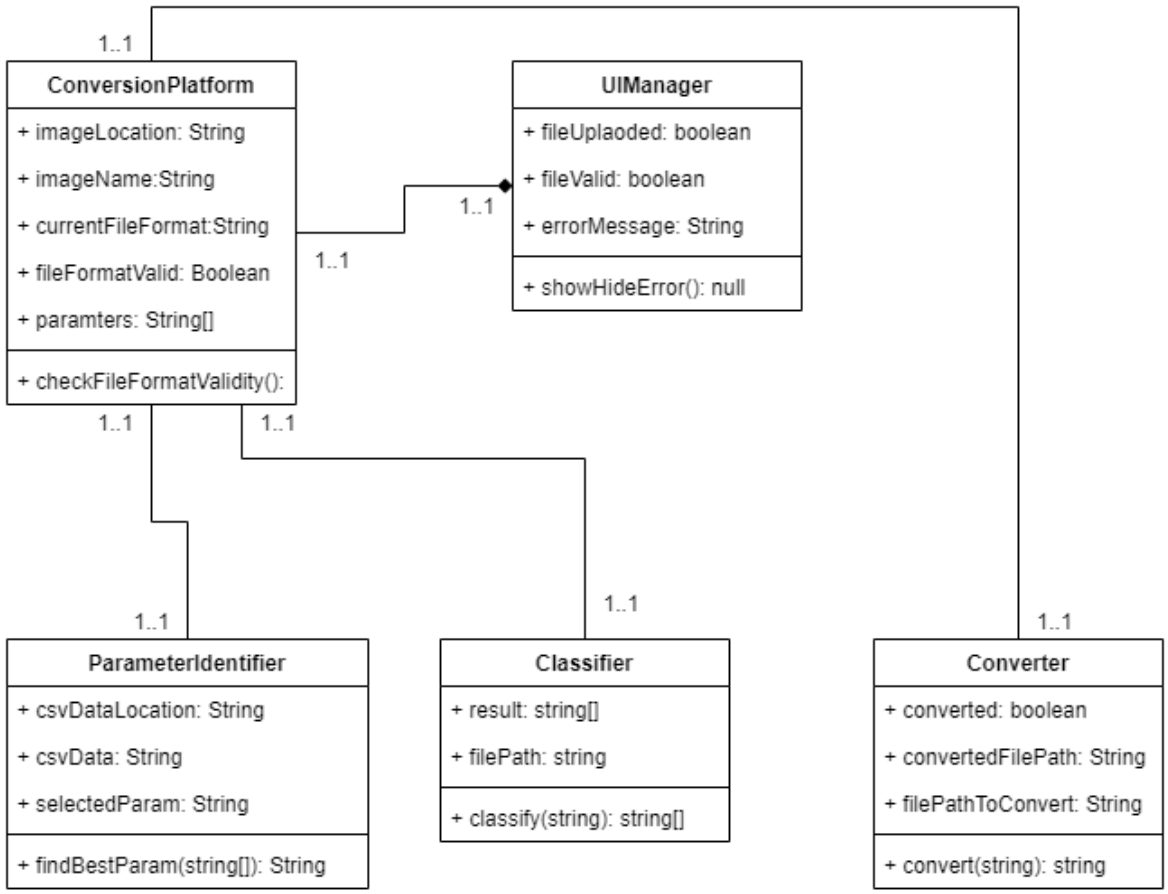
5.1 Process Overview

The objective that is to be achieved in the research is to develop a software platform that can analyze an image, identify its properties and convert an image from raster to vector. Several separate components will have to be designed to achieve the final goal of this system, that is the conversion process. A process flow must initially be defined to design the system. This process flow will be discussed below and further elaborated in the diagrams found in this chapter. The three main components that are identified on a high level are the conversion component, the image classification component and the parameter trainer and selector component. The image classification component will be a sub system where an image is classified into types of GIS imagery types. A data set will be prepared and an image classification model using CNN (convolution neural networks) will be used to train a model that will be used in the classification process. The parameter trainer and identifier will be an application which converts images using the raster to vector conversion library that has been selected using each parameter combination possible and create a data set where the best parameter set and Structural similarity between the input and the output is calculated. Finally, this generated parameter and SSIM value data is used to convert the image after it is converted analyzed, and all functions of pre-processing deemed suitable are executed on the image. Figure represent a high-level diagram of the proposed system.



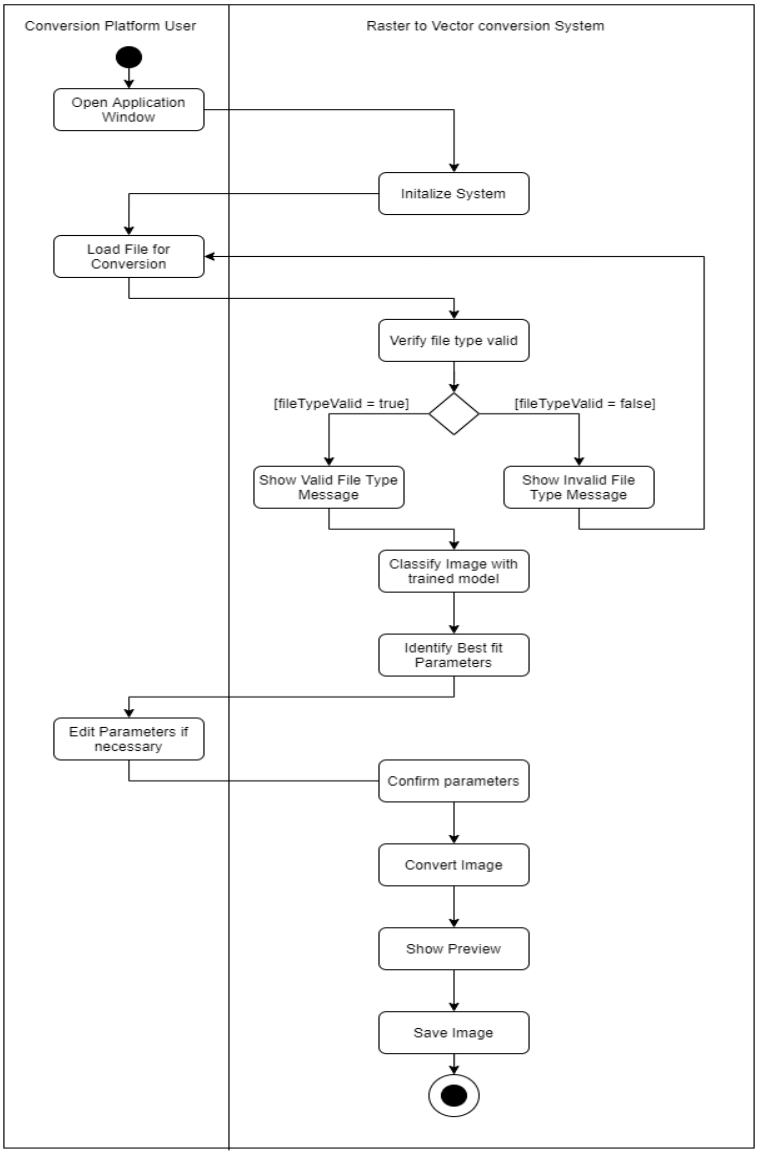
5.2 Class Diagram

Class diagram below represents the attributes and methods of each class in the proposed raster to vector conversion platform. Each class's relationship with each other is also shown and also aggregation/composition relationships.

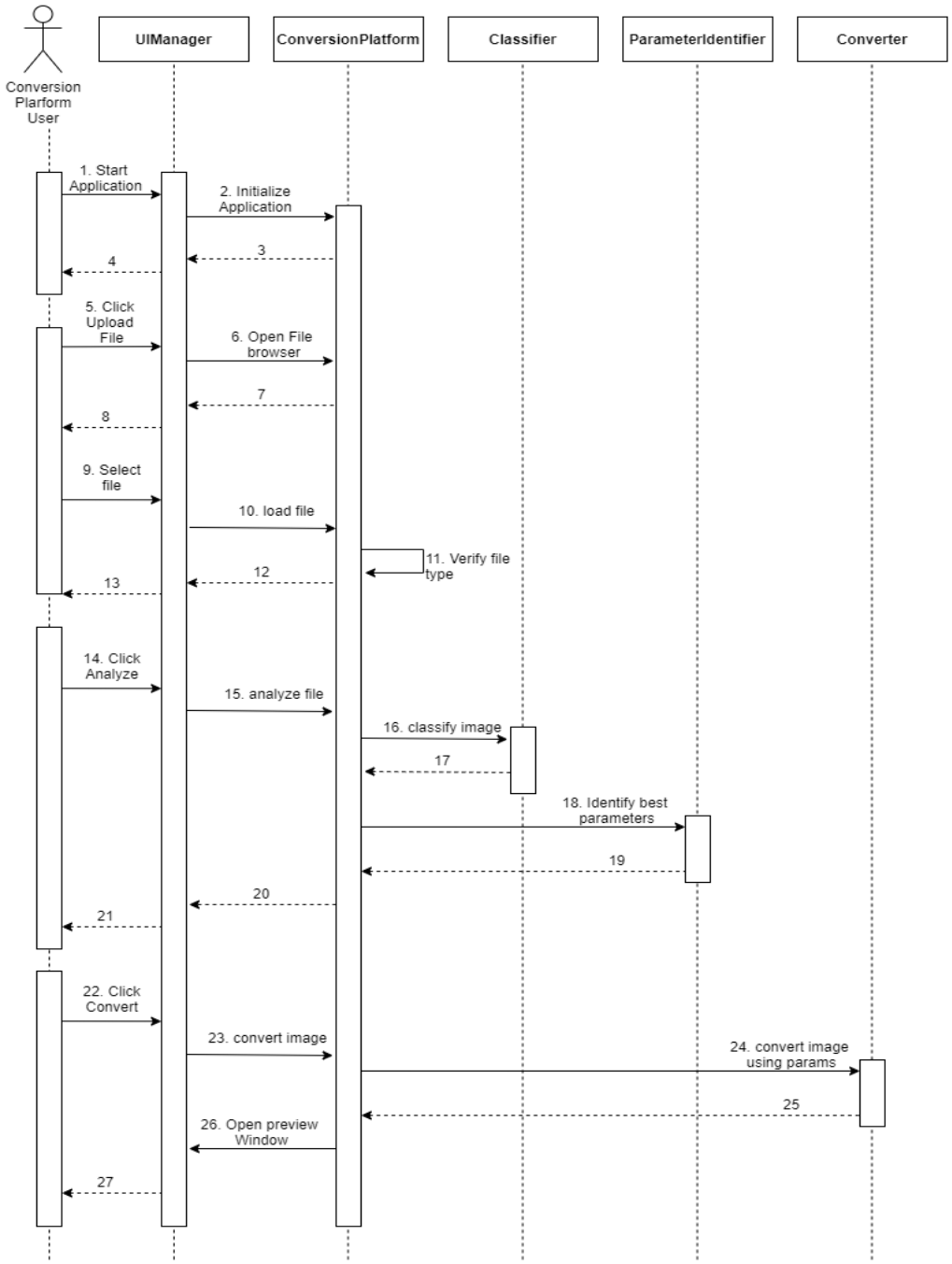


5.3 Activity Diagram

The flow of activities in the Raster to Vector conversion platform is shown by the activity diagram shown below. This diagram is created according to the flows that were described in the use case diagrams and descriptions. This can be identified as the successful flow of activities for converting an image from raster to vector.



5.4 Sequence Diagram



The diagram above represents how the user interacts with the raster to vector conversion platform. The user starts by opening up the application, which then initializes the conversion platform. Then the user can locate an image file that they wish to convert. Once the file is loaded. It will be verified if it is of the valid file types accepted by the system. If this is the case. The user will be allowed to analyze the file. The conversion platform then uses the image classification trained model to classify the image and get its accuracy as to which classification it belongs to. Using this information then the best fit parameters are identified. Then the user is allowed to click the convert button which then converts the image into a temporary location. The user is also showed a preview of the converted image. This can be considered the sequence followed for a successful image conversion.

6. Implementation of Components

6.1 Parameter Trainer Component

This component handles the pre training of the system to determine the best conversion parameters for each classification of GIS image determined by the image classification model. It contains the methods for various image pre-processing functions that are used throughout the system as well.

6.1.1 Image Resizing

```
13      # resize Image
14      def image_resize(image_to_resize: object, set_width: int):
15          width = int(image_to_resize.shape[1] * set_width / image_to_resize.shape[1])
16          height = int(image_to_resize.shape[0] * set_width / image_to_resize.shape[1])
17          dimensions = (width, height)
18          return cv2.resize(image_to_resize, dimensions, interpolation=cv2.INTER_AREA)
19
```

The image pre-processing functions that ready the image for the training and ssim functions are a vital part of the whole system. Every time an image is to be imported and converted using open cv. After it is imported it has to be resized to certain dimension for operation such as quantization and structural analysis to be run on them. As some images are too large. They may cause the system to run out of memory. Hence for operations such as image classification model training and Structural similarity index to be calculated on the image data sets it should be scaled down to a constant size. This also helps with training of the image classification model as images with similar dimensions train better with less inconsistencies.

This function gets an image and a width and scales the image height, maintaining its aspect ratio to the newly defined width and returns it.

6.1.2 Image Colour Quantization

```
21      # quantize image to reduce colour data for easy raster to vector conversion
22      def quantize_image(image: object, number_of_colors: int):
23          (h, w) = image.shape[:2]
24          image = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)
25          image = image.reshape((image.shape[0] * image.shape[1], 3))
26          clt = MiniBatchKMeans(n_clusters=int(number_of_colors))
27          labels = clt.fit_predict(image)
28          quantized_image = clt.cluster_centers_.astype("uint8")[labels]
29          quantized_image = quantized_image.reshape((h, w, 3))
30          quantized_image = cv2.cvtColor(quantized_image, cv2.COLOR_LAB2BGR)
31          # return resized image
32          return quantized_image
```

Image quantization is also an important pre-processing functionality that is being employed to reduce the computational power required and time of running of the system when dealing with high quality images with high number of colours that also might make the conversion software run out of memory due there being really high polygon count when raster to vector conversion is to be done. Colour quantization in method is done by using the clustering algorithm found in k-means and passing the

number of clusters to be used when quantization of the image. The image should be converted to the LAB colour format before being used to in the clustering algorithm the image after quantization is converted back into the BGR format the default colour space configuration for Open CV. The quantized image is then returned.

6.1.3 Converting SVG into PNG

```
39  def convert_to_png(in_path: str, out_path: str):
40      subprocess.call(["svg2png.bat", in_path, out_path])
```

As there is no python library to convert SVG images into PNG format, a library known as inkscape is being used to do this process. The requirement for an SVG image to be converted to PNG comes when the result conversion has to be compared to the original image to identify its Structural similarity as that is the index used to measure the quality of the output in this project. This code calls a sub process call and runs the svg2png batch file which accepts 2 arguments which are the path of the image to be converted and the output path where the converted image will be saved to. Sub process is an inbuilt module in python that allows the spawning of new processes and obtain their return codes.

6.1.4 SVG2PNG.bat

```
1  cd libraries/inkscape/bin
2  set arg1=%1%
3  set arg2=%2%
4
5  set path1=%arg1%
6  set path2=%arg2%
7
8  inkscape  %path1% -o %path2%
```

Inkscape is an open source Vector graphics editor. It also has a command line version of its tool and also supports functionalities such as conversion vector data formats to PNG or jpeg formats. This batch file takes the power of the inksacpe command line tool and converts the image that it obtains as the first augment and saves it in the location provided in the second argument. As mentioned above this batch file is invoked though python using sub processes.

6.1.5 Image Structural Similarity Index Calculator

```
43  def check_similarity(path_1: str, path_2: str, image_size):
44      img1 = cv2.imread(path_1)
45      img2 = image_resize(cv2.imread(path_2), image_size)
46
47      w1, h1 = img2.shape[:-1]
48      img1 = cv2.resize(img1, (h1, w1))
49
50      s = ssim(img1, img2, multichannel=True)
51      return str(s * 100)
```

Structural similarity index measure (SSIM) calculates the mean squared error (MSE). Structural similarity takes texture of an image to account when comparing the original and to be compared to image. Two images are passed in as parameters where the later image is resized to be the same size as the original image, then using Scikit-images SSIM algorithm the similarity between these two images are calculated and returned as a percentage.

6.1.6 Variable Trainer Process

```
54  def variable_full_process(category):
55      image_size = 720
56      end_file_type = ".svg"
57
58      # Initialize directories for category
59      try:
60          os.mkdir("images/png/" + category)
61          os.mkdir("images/tempimages/" + category)
62          os.mkdir("images/converted/" + category)
63          os.mkdir("csv/" + category)
64      except OSError as e:
65          print(e)
66
67      # Load necessary paths for the reading and writing
68      data_path = "images/data/" + category
69      temp_path = "images/tempimages/" + category
70      out_svg_base_path = "images/converted/" + category
71      png_base_path = "images/png/" + category
72
73      csv_path = 'csv/' + category
74
75      for img in os.listdir(data_path):
76
77          image = cv2.imread(os.path.join(data_path, img))
78          image = image_resize(image, image_size)
79          image = quantize_image(image, 16)
80
81          image_name = img.split(".")[0]
82          image_extension = img.split(".")[1]
83
84          try:
85              os.mkdir(os.path.join(temp_path, image_name))
86              os.mkdir(os.path.join(out_svg_base_path, image_name))
87              os.mkdir(os.path.join(png_base_path, image_name))
88          except OSError as e:
89              print(e)
90
91          out_svg_specific_path = os.path.join(out_svg_base_path, image_name)
92
93          sample_path = os.path.join(temp_path, image_name, "sample." + image_extension)
94          save_image_to_path(sample_path, image)
95
96          index = 0
```

```
97
98          with open(csv_path + "/" +image_name + '.csv', 'w', newline='') as f:
99              thewriter = csv.writer(f)
100
101             thewriter.writerow(["index", "ltres", "qtres", "pathomit", "file_path", "similarity"])
102
103             for ltres in range(0, 9, 2):
104                 for qtres in range(0, 9, 2):
105                     for pathomit in range(0, 101):
106                         if pathomit == 1 or pathomit == 10 or pathomit == 100:
107                             print("progress:" + str(int(index / 75 * 100)) + "%")
108
109                             index = index + 1
110
111                             subprocess.call([
112                                 'java', '-jar', 'libraries/imageTrace.jar', sample_path,
113                                 'outfilename', out_svg_specific_path + "/" + str(index) + end_file_type,
114                                 'pathomit', str(1 / pathomit),
115                                 'ltres', str(ltres),
116                                 'qtres', str(qtres),
117                                 'colorsampling', str(0),
118                                 'colorquantcycles', str(16)
119                             ])
120
121                             png_out_path = "images/png/" + category + "/" + image_name + "/" + str(index) + ".png"
122
123                             convert_to_png("../../../" + out_svg_specific_path + "/" + str(index) + end_file_type,
124                                            "../../../" + png_out_path)
125
126                             similarity_val = check_similarity(sample_path, png_out_path, image_size)
127
128                             thewriter.writerow(
129                                 [str(index), str(ltres), str(qtres), str(pathomit), str(index) + end_file_type,
130                                  str(similarity_val)])
131
132     print("completed!")
```

The full process of identifying best fit parameters of each classification of image as classified by the trained image classification model starts by creating and initializing the directories required by the function to store and read images to and from. Once all directories are initialized and the paths to the data set directories are all validated A for loop loops through all the data for the necessary category (images). The image is then resized and quantized using the methods described above. The processed image is then stored in a temporary location which gets overwritten every time the for loop moves into a new iteration. A csv writer then is opened and a csv of the name of the image is created to store the parameters, and similarity of each conversion. Then the code enters a series of nested for loops that iterate each through a single parameters possible values. The temporary image is then converted using these parameters each time. Converted to PNG using the method described above and stored and then the converted PNG and the temp image is compared to identify similarity between them using the method described above. The values are then stored in the csv file and after all the parameter values are iterated through the initial for loops moves on to the next image. After all images in the directory have been iterated through the function is ended.

6.1.7 Create Best Fit Parameters for category

```
135  def read_csv(category: str):
136      base_file_path = 'csv'
137
138      with open('final_csv/' + category + '.csv', 'w', newline='') as f:
139          thewriter = csv.writer(f)
140
141          thewriter.writerow(["index", "ltres", "qtres", "pathomit", "file_path", "similarity"])
142
143          for csv_path in os.listdir(os.path.join(base_file_path, category)):
144              df = pd.read_csv(os.path.join(base_file_path, category, csv_path), engine='python')
145              df = df.sort_values('similarity', ascending=False)
146              df.to_csv(os.path.join(base_file_path, category, "converted_" + csv_path), index=False)
147
148          for csv_path in os.listdir(os.path.join(base_file_path, category)):
149              if (csv_path.startswith("converted_")):
150                  input_file = csv.DictReader(open(os.path.join(base_file_path, category, csv_path)))
151
152                  # print highest 10 accracy values
153                  index = 0
154
155                  for row in input_file:
156                      if index < 1:
157                          thewriter.writerow(
158                              [str(row['index']), str(row['ltres']), str(row['qtres']), str(row['pathomit']),
159                               str(row['index']) + row['file_path'],
160                               str(row['similarity'])])
161
162                      index += 1
```

The parameter and ssim csv files are read by a for loop where each is arranged in descending order of their ssim values and the highest ssim value of each parameter file is taken and noted down in a csv file. This csv file then contains the best parameters that were used to get the highest ssim value in all of the images that were used as the data set. In the previous method where the parameter iteration and ssim is carried out. This csv is then saved.

6.1.8 Cleanup

```
165  def cleanup(base_file_path: object, category: object) -> object:
166      for csv_path in os.listdir(os.path.join(base_file_path, category)):
167          if csv_path.startswith("converted_"):
168              os.unlink(os.path.join(base_file_path, category, csv_path))
```

Deletes all the temporary csv files created in the process of sorting and creating the final category best fit parameter csv file.

6.1.9 Full Training Process Executor

```
171  def finalize_params():
172      category = 'sat'
173
174      variable_full_process(category)
175      read_csv(category)
176      cleanup('csv', category)
177
```

This method runs all of the methods described above in their proper sequence. This process is fully auto mated and the sample space for the best fir parameters can be increased simply by increasing the dataset of images per category. And as this is an automated process. There is no monitoring required by the user of the application to create the best fit parameter files.

6.2 Image Classifier Component

This component handles the classification of various types of GIS data into three main categories. Satellite Imagery, Land classification Images and Scanned maps. This component has three main subcomponents. One that prepares the data and creates the training data set, the Model training component and the component that creates the neural network model with its layers and other parameters.

6.2.1 Create and Prepare Training Data

```
12  def create_training_data(categories, data_dir, img_size, training_data):
13      for category in categories:
14          print("Image Loading from " + category + "...")
15          path = os.path.join(data_dir, category)
16          class_num = categories.index(category)
17          for img in os.listdir(path):
18              try:
19                  img_array = cv2.imread(os.path.join(path, img), cv2.IMREAD_GRAYSCALE)
20                  resized_array = cv2.resize(img_array, (img_size, img_size))
21                  training_data.append([resized_array, class_num])
22              except Exception as e:
23                  print("Image Failed to Load")
24                  pass
25      random.shuffle(training_data)
```

The images from a directory are loaded onto the application and stored in an numpy image array. The images here are loaded in grayscale (2 channels) and are indexed with the category, which is their folder name in which they reside in the data directory. The data is then shuffled to get better test result when training the system as it gets trained to each category randomly without being trained one after the other. This can help with the performance of the trained models accuracy.

6.2.2 Training Image Classification Model

```
28  def train_save_model(training_data, img_size):
29      print("starting to train data...")
30
31      X = []  # images
32      y = []  # labels
33
34      for features, label in training_data:
35          X.append(features)
36          y.append(label)
37
38      X = np.array(X).reshape(-1, img_size, img_size, 1)
39      y = np.array(y)
40
41      X = tf.keras.utils.normalize(X, axis=1)
42
43      model = Sequential()
44
45      model.add(Flatten(input_shape=X.shape[1:]))
46      model.add(Dense(64, activation='relu'))
47      model.add(Dense(3, activation='softmax'))
48
49      model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
50      model.fit(X, y, batch_size=32, epochs=20, validation_split=0.1)
51      print(model.summary())
52      return model
```

Once the prepared training data and image sized that is being used is passed as arguments into this method. It creates two arrays, one with the features (the image) X and the other with their corresponding label y. The X array of images is converted to a numpy array and reshaped using the image size passed as an argument. The values in X are then normalized to obtain better accuracy when training. The model is then created using the **layers** library that is provided by Keras by Tensorflow. In this model, a flatten layers then two dense layers with 64 and 3 nodes respectively are added sequentially the activation function for the first dense layer is **relu** which is a linear activation function that ramps up for any value that is not negative. The second dense layer has a **softmax** activation function and as we have only three categories it has 3 nodes. This model is then compiled using the *optimized* **adam**, *loss function* **sparse categorical crossentropy,** and to obtain metrics of accuracy. The model is then fitted with the data set. Which trains it to classify images between those three categories. And finally returns this model.

6.2.3 Training and Saving Model

```
def train():
    data_dir = "D:\FYP\DataSets\DataCategorised"
    categories = ["LandClassificationMaps", "Satellite", "Scanned"]

    img_size = 250
    training_data = []

    print("starting...")
    create_training_data(categories, data_dir, img_size, training_data)

    print("data loaded!")
    model = train_save_model(training_data, img_size)

    model.save('classification_model/gis_classify.h5')
```

This function calls the above two described methods of this component by passing in the necessary parameters for each function such as the data directory for the data sets, and categories array which is an array of the folder name of the categorized data set which will be used for labelling the images. The image size is also then defined. After the data set is created and the model is trained the model is saved locally to a file of file type h5. It is a Hierarchical Data Format that contain multi-dimensional array of scientific data.

6.3 Backend API

This component exposes the backend functionalities to the front end application through an application programming interface. This API uses the python Flash API framework.

6.3.1 Flask API Initialization

```
from flask import Flask, jsonify, render_template, request
from flask_cors import CORS

import subprocess
import imghdr
import csv
import os

import numpy as np
import pandas as pd
import cv2

from tensorflow.keras.models import Sequential, load_model
from sklearn.cluster import MiniBatchKMeans
from skimage.metrics import structural_similarity as ssim

app = Flask(__name__)
CORS(app)
```

```
if __name__ == '__main__':
    app.run()
```

This segment of code initializes the Flask application, enables CORS for cross origin access as both these backend and frontend to communicate without the browser blocking it. And starts the application.

6.3.2 Flask API method signatures

```
converter = Converter()
analyzer = Analyzer()


@app.route('/convert')
def perform_conversion():
    return converter.convert()


@app.route('/analyze')
def perform_classification():
    return analyzer.analyze()


if __name__ == '__main__':
    app.run()

```

These two methods are the methods of the API that are exposed through the routes defined and can be called through an http request from the front end. The methods which are being executed here will be further explained below.

6.3.3 Analyzer Class

```
class Analyzer:
    def prepare_image(path: str):
        img_size = 250
        img_array = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
```

This is a python class that contains the method necessary for the image to be analysed and also for various pre-processing functions to be carried out before analysis, and finally identify the best fit parameters for the input image uploaded by the user. In this section we will not discuss on the methods that are used for quantization, resize and reshaping as it has been discussed precious and the same concept is implemented in this class as well. Instead the unique functions of the analyzer class will be described.

6.3.4 Classify Image (Analyzer Class)

```
50      def classify(path: str):
51          my_model = load_model('classification_model/gis_classify.h5')
52          prediction = my_model.predict(Analyzer.prepare_image(path))
53
54          # ["LandClassificationMaps", "Satellite", "Scanned"]
55
56          values = prediction[0]
57          type = "unidentified"
58          category= "unidentified"
59
60          if values[0] == 1:
61              type = "Land Classification Maps"
62              category = "landclass"
63          elif values[1] == 1:
64              type = "Satellite Imagery"
65              category = "sat"
66          elif values[2] == 1:
67              type = "Scanned Map"
68              category = "scanned"
69
70          return [type, category]
```

This method uses the h5 file of the model during the classification model training process and classifies an image that is passed into it. And return the type of the image and the category which is assigned by nested if statements that reads the prediction result and decides the classification of the image.

6.3.5 Colour Counter (Analyzer Class)

```
45      def count_colours(src: str):
46          image = cv2.imread(src)
47          unique, counts = np.unique(image.reshape(-1, image.shape[-1]), axis=0, return_counts=True)
48          return counts.size
```

This method calculates the number of unique colours in the image by converting it into a numpy array and counting the number of unique values in each pixel. It is a rough value of the number of colours in the image but can be helpful to understand how to quantize the image without losing a lot of quality.

6.3.6 Determine Colour Clusters (Analyzer Class)

```
72      def identify_quntize_cluster(colors: int):
73          if colors < (2 ** 4):
74              return 4
75          elif colors < (2 ** 8):
76              return 8
77          elif colors < (2 ** 12):
78              return 12
79          elif colors < (2 ** 16):
80              return 16
81          elif colors < (2 ** 20):
82              return 20
83          elif colors < (2 ** 24):
84              return 24
85          elif colors < (2 ** 28):
86              return 28
87          else:
88              return 32
```

It is a simple nested If loop to determine the number of colour clusters required for quantization. It calculates this value using the colour counter value that is returned.

6.3.7 Get Best Parameters (Analyzer Class)

```
90      def get_best_param_range(self, category):
91          path = 'final_csv/'+category+'.csv'
92
93          ltres_low = Analyzer.find_lowest(path, "ltres")
94          ltres_high = Analyzer.find_highest(path, "ltres")
95
96          qtres_low = Analyzer.find_lowest(path, "qtres")
97          qtres_high = Analyzer.find_highest(path, "qtres")
98
99          pathomit_low = Analyzer.find_lowest(path, "pathomit")
100         pathomit_high = Analyzer.find_highest(path, "pathomit")
101
102         return [('ltres_low', ltres_low), ('ltres_high', ltres_high), ('qtres_low', qtres_low),
103                 ('qtres_high', qtres_high),
104                 ('pathomit_low', pathomit_low), ('pathomit_high', pathomit_high)]
105
106     def find_lowest(path, prop_val):
107         input_file = csv.DictReader(open(path))
108         lowest = 0
109         index = 0
110
111         for row in input_file:
112             if index == 0:
113                 lowest = row[prop_val]
114             else:
115                 if row[prop_val] < lowest:
116                     lowest = row[prop_val]
117
118             index += 1
119
120         return lowest
```

```
122     def find_highest(path, prop_val):
123         input_file = csv.DictReader(open(path))
124         highest = 0
125         index = 0
126 
127         for row in input_file:
128             if index == 0:
129                 highest = row[prop_val]
130             else:
131                 if row[prop_val] > highest:
132                     highest = row[prop_val]
133 
134             index += 1
135 
136         return highest
```

This method is responsible for determining the best fit parameters range for the category of the image that has been uploaded by the user. This category is a parameter passed into the method and is the classification category determined when the image is classified using the model. The method then reads the best parameter csv regarding that category and determines the highest and lowest values for each of the parameters and determines a range for the best fit parameters to exist in and returns this as an array of tuples.

6.3.8 Determining Best parameters for user image (Analyzer Class)

```
174     def get_params(image_path, category):
175         try:
176             os.rmdir('temp/png')
177         except OSError as e:
178             print("does not exist")
179 
180         try:
181             os.rmdir('temp/svg')
182         except OSError as e:
183             print("does not exist")
184 
185         try:
186             os.mkdir('temp/svg')
187         except OSError as e:
188             print("does not exist")
189 
190         try:
191             os.mkdir('temp/png')
192         except OSError as e:
193             print("does not exist")
194 
195         end_file_type = ".svg"
196         param_range = Analyzer().get_best_param_range(category)
197         image_size = 720
198 
199         image = cv2.imread(image_path)
200         image = Analyzer.image_resize(image, image_size)
201         image = Analyzer.quantize_image(image, 16)
202         image_extension = imghdr.what(image_path)
203         temp_image_path = 'temp/sample.' + image_extension
204         Analyzer.save_image_to_path(temp_image_path, image)
205 
206         index = 0
```

```
207
208         with open('temp/temp.csv', 'w', newline='') as f:
209             thewriter = csv.writer(f)
210
211             thewriter.writerow(["index", "ltres", "qtres", "pathomit", "file_path", "similarity"])
212
213             for ltres in range(int(param_range[0][1]), int(param_range[1][1]) + 1, 1):
214                 for qtres in range(int(param_range[2][1]), int(param_range[3][1]) + 1, 1):
215                     for pathomit in range(int(param_range[4][1]), int(param_range[5][1])):
216                         if pathomit == 1 or pathomit == 10 or pathomit == 100:
217                             index = index + 1
218
219                             svg_out_path = "temp/svg/" + str(index) + end_file_type
220
221                             subprocess.call([
222                                 'java', '-jar', 'imageTrace.jar', temp_image_path,
223                                 'outfilename', svg_out_path,
224                                 'pathomit', str(1 / pathomit),
225                                 'ltres', str(ltres),
226                                 'qtres', str(qtres),
227                                 'colorsampling', str(0),
228                                 'colorquantcycles', str(16)
229                             ])
230
231                             png_out_path = "temp/png/" + str(index) + ".png"
232
233                             Analyzer.convert_to_png("../../../" + svg_out_path, "../../../" + png_out_path)
```

```
235                             similarity_val = Analyzer.check_similarity(temp_image_path, png_out_path, image_size)
236
237                             thewriter.writerow(
238                                 [str(index), str(ltres), str(qtres), str(pathomit), str(index) + end_file_type,
239                                  str(similarity_val)])
240
241         df = pd.read_csv('temp/temp.csv', engine='python')
242         df = df.sort_values('similarity', ascending=False)
243         df.to_csv(os.path.join('temp/temp_sorted.csv'), index=False)
244
245         input_file = csv.DictReader(open('temp/temp_sorted.csv'))
246
247         # get highest accuracy values
248         index = 0
249
250         highest_row = []
251
252         for row in input_file:
253             if index < 1:
254                 highest_row = row
255
256             index += 1
257
258         return highest_row
```

This method is the main function of the Analyzer class which determines the best fit parameters for the image selected by the user. This method too gets the category as a parameter for its operations. It selects the csv file of best parameters for its category. And then after obtaining the best parameter range from the method previous defined it runs conversion on them similar to the parameter trainer component and finally returns the set of parameters with highest Structural similarity index.

6.3.9 Analyze (Analyzer Class)

```
261     def analyze(self):
262         file_path = request.args.get('file_path')
263 
264         classification = Analyzer.classify(file_path)
265         color_count = Analyzer.count_colours(file_path)
266         quant_val = Analyzer.identify_quntize_cluster(color_count)
267 
268         row = Analyzer.get_params(file_path, 'landclass')# category[1]
269 
270         return jsonify(file_path=file_path, classification=str(classification[0]), color_count=color_count
271                        , quant_val=quant_val, row=row)
```

This method combines all the methods defined above and creates the final JSON that will be passed to the front end with all of the parameters and some additional data that will be shown to the user before conversion.

6.3.10 Converter Class

```
21  class Converter:
22      def convert(self):
23          file_path = request.args.get('file_path')
24          file_name = request.args.get('file_name')
25          qtres = request.args.get('qtres')
26          ltres = request.args.get('ltres')
27          pathomit = request.args.get('pathomit')
28          k = request.args.get('k')
29          quant_bool = request.args.get('quantBool')
30          end_file_type = ".svg"
31 
32          if quant_bool:
33              Analyzer.quantize_image(cv2.imread(file_path))
34 
35          subprocess.call([
36              'java', '-jar', 'imageTrace.jar', file_path,
37              'outfilename', file_name + end_file_type,
38              'colorsampling', str(0),
39              'colorquantcycles', str(k),
40              'ltres', str(ltres),
41              'qtres', str(qtres),
42              'pathomit', str(pathomit)
43          ])
44 
45          return jsonify(progress="completed", file_name=file_name, file_path=file_path, out_path=file_name + end_file_type)
46 
```

This component is the class that handles the conversion of the image from all the parameters that have been identified using the analyser phase of the applications flow. It obtains the parameters from the URL params of the request and converts the image using the imagetrace library and output the image to the specified location. Once the operation is complete returns a JSON with the success flag to update the UI.

6.4 Frontend

6.4.1 Initialization

```
// Modules to control application life and create native brow
const {app, BrowserWindow} = require('electron')
const path = require('path')

function createWindow () {
  // Create the browser window.
  const mainWindow = new BrowserWindow({
    width: 1000,
    height: 600,
    frame: false,
    webPreferences: {
      preload: path.join(__dirname, 'preload.js'),
      nodeIntegration: true
    }
  })

  // and load the index.html of the app.
  mainWindow.loadFile('index.html')

  // Open the DevTools.
  // mainWindow.webContents.openDevTools()
}

// This method will be called when Electron has finished
// initialization and is ready to create browser windows.
// Some APIs can only be used after this event occurs.
app.whenReady().then(createWindow)
```

Electron JS application initialization configuration. The configuration defined here will be used when creating the window for the frontend application which will be initialized.

```
function analyze_file() {
    $.ajax({
        url: "http://127.0.0.1:5000/analyze?file_path=" + document.getElementById("convert-file").files[0].path,
        dataType: "json",
        success: function (data) {
            console.log(data);

            $('.step-1').hide();
            $('.step-2').show();

            $(".on-hover").css("pointer-events", "auto");

            preview_analysis_data(data);
        },
        error: function (data) {
            $('#analyze-error').show()

            $('#before-analyzing').show()
            $('#after-analyzing').hide()

            $(".on-hover").css("pointer-events", "auto");
            $('#analyze-btn').attr("disabled", false);
            $('#analyze-progress').hide();
        }

    })
}

function preview_analysis_data(data){
    document.getElementById("prediction-val").innerText = data.classification;
    document.getElementById("color-val").innerText = data.color_count;
    document.getElementById("quant-val").innerText = data.quant_val;

    document.getElementById("quant-range").value = data.quant_val;

    document.getElementById("ltres-val").value = data.row.ltres;
    document.getElementById("qtres-val").value = data.row.qtres;
    document.getElementById("pathomit-val").value = data.row.pathomit;
}
```

This function sends a request to the backend Flask API and gets the parameters for the file defined in the file path URL parameter. And after a successful response updates the UI with these values preparing the application for the conversion process.

```
$("#converter").click(function () {
    $.ajax({
        url: "http://127.0.0.1:5000/convert?" +
        "file_path="+ document.getElementById("convert-file").files[0].path+
        "ltres="+ document.getElementById("ltres-val").innerText+
        "qtres="+ document.getElementById("qtres-val").innerText+
        "pathomit="+ document.getElementById("pathomit-val").innerText+
        "k="+ document.getElementById("quant-range").value+
        "quantBool="+ true,
        dataType: "json",
        success: function (data) {
            console.log(data)
        }
    })
})
```

And after converts the image from raster to vector using the parameters found. When triggered by the user.

7. Testing

This testing phase is carried out to ensure that the system implemented as described in Chapter 7 of this document meet the functional and non-functional requirements that were descried during the requirement specification stage of this project. This testing phase makes sure the following goals are achieved.

- Identifying bugs and fixing them.
- Validate and verify if all the functional requirements have been met by the system.
- Validate and verify if all the non-functional requirements have been met by the system.

7.1 Testing Criteria

As the testing criteria of this testing phase, the following two criteria which are defined below will be used to identify and reduce the gap between the expectation and reality of the implementation.

Functional Testing – To test the functional requirements defined

Non-Functional Testing – To test non-functional requirements of the system. But elevate the overall experience of using the system.

7.2 Unit Testing

The system implemented in this research, comprises of many unit components that through integration complete the overall process of the system. As these unit functionalities have to perform and generate the expected outputs for the full system to work. The validity of each component may affect the next component that it is to interact with. If this is not validated and rectified where anomalies occur, a waterfall affect may occur and affect the accuracy and performance of the system as a whole. Unit testing is thus used to test the individual units of the system to validate that the component being tested performs as expected.

7.2.1 Unit Testing Tools

PyTest has been selected as the unit testing tool to be used in the testing phase of this projects' unit tests. It is a simple testing framework that is open source and is well documented. It allows for compact test scenarios. **PyTest** allows storing of values from test cases, the ones which were asserted successfully and the ones which weren't. Therefore, it is much easier than going through logs and using debuggers to identify test result.

The Following table describes the unit test cases and their results that were carried out during the unit testing phase.

ID	Test case	FR ID	Priority	Result
UTC01	Load data set for variable and SSIM comparison csv creation	3	High	Passed
UTC02	Quantize loaded image	1	High	Passed
UTC03	Write parameters and SSIM to csv per each image	3	High	Passed
UTC04	Sub process call to Convert to Vector	3	High	Passed
UTC05	Bat file execution for SVG to PNG conversion	3	High	Passed
UTC06	Obtain Similarity Index for input image and output image comparison	3	High	Passed
UTC07	Read csv and order in ascending order of given column	3	High	Passed
UTC08	Find lower range of column in csv	3	High	Passed
UTC09	Find Higher range of column in csv	3	High	Passed
UTC10	Find best fit parameters from given csv	3	High	Passed
UTC11	Classify image using image classification model file which was fitted and saved.	2	High	Passed

Table 7.2:1 Unit test case results

7.3 Integration Testing

Integration testing is the testing of the unit modules after being logically integrated. This system comprises of multiple unit modules that in turn must communicate with each other to function as a complete system and produce the expected output. In this section integration test cases and their results will be depicted.

Test case	Component Name	Actual Result	Expected Result	Test Result
Train Parameters for Image category when data set provided	Python Parameter Trainer Module	(3*5*5) = 75 rows produced with ssim values in csv for each parameter iteration	75 rows created in csv for each parameter iteration with ssim value	Pass
Load Image for Analyzing	UI (Frontend)	Load image when selected from file browser, and reject all unsupported file formats	Load image file types only when selected from file browser	Pass
Analyze Image and return parameter JSON	Flask API	Return best row parameter obtained by reading csv	Return best row parameter	Pass
Send parameters and options defined by user to API through ajax call	UI (Frontend)	Send parameters through *url params* that were updated in the UI	Send parameters through *url params* to the backend	Pass

Convert Image with parameters obtained through request parameters	Flask API	Run conversion library and convert image	Run conversion library through subprocess and convert image	Pass

Table 7.3:1 Integration testing results

7.4 Testing Image Classification Model Accuracy

The following are the testing result and training result for the image classification model that was created to classify between the identified three types of common GIS imagery. The model was trained on 1418 samples and validated on 158 samples. 20 epoch were run to train the model and below are the results for the trainin accruacy and loss and validation accuracy and loss in each epoch during the training process.

Figure 7.4:1 Accuracy of each epoch

Figure 7.4:2 Loss of each epoch

The above statistics have been logged using Tensor board. A tool for providing the measuring and visualizing machine learning workflows. It can be seen that the loss at the end of the training of the 20 epochs was 0.2669 and loss of validation was 0.2378, the accuracy for training was 0.8886 and validation was 0.9114 (all normalized between 1 and 0) And as this has high accuracy and low loss it can be considered this model has a good classification strength.

7.5 Testing Image analyzing process Performance of Flask API Backend

This testing method analyzes the time taken for an image to be analyzed according to its image size. The data is collected over several iteration of the process by checking duration taken to analyze images of various image sizes. The images were prepared for this task using an image editing tool. *Note: the images used were chosen at random*

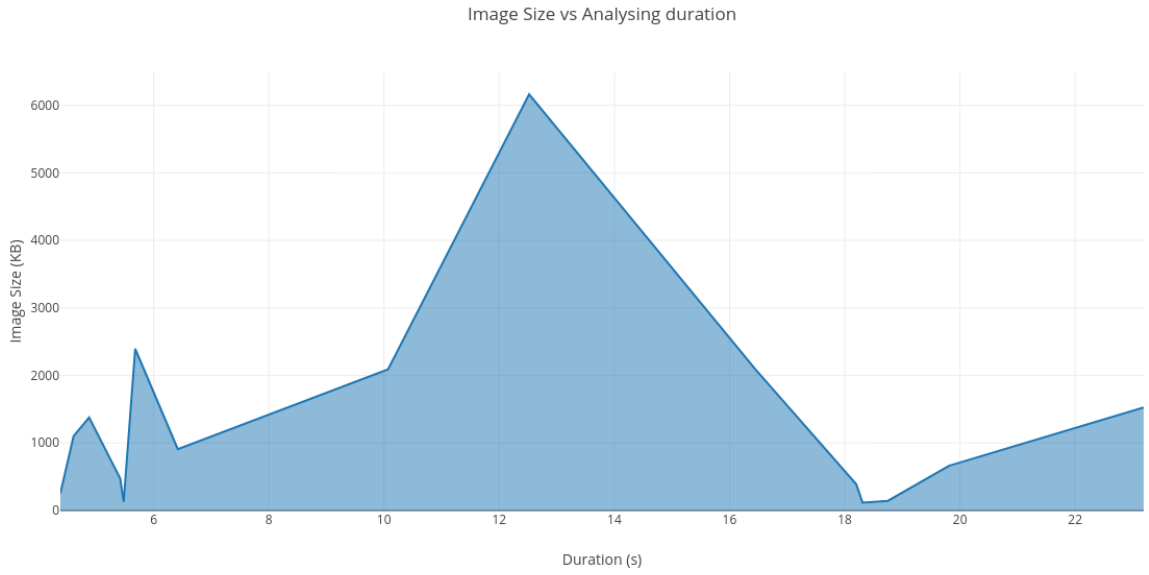


Figure 7.5:1 Image size vs analysis duration

Image 1: Land classification map

File size	Dimensions	Duration (s)
2080 kb	3720 x 2480	16.46
2092 kb	2500 x 1667	10.07
909 kb	1280 x 854	6.42
474 kb	300 x 534	5.42
129 kb	360 x 240	5.48

Table 7.5:1 Land classification map result for analysis duration testing

Image 2: Satellite Imagery

File size	Dimensions	Duration (s)
1378 kb	1024 x 1024	4.88
6165 kb	2500 x 2500	12.52
2395 kb	1280 x 1280	5.68
1012 kb	800 x 800	4.61
253 kb	360 x 360	4.38

Table 7.5:2 Satellite Image result for analysis duration testing

Image 1: Scanned Maps

File size	Dimensions	Duration (s)
116 kb	354 x 340	18.31
1527 kb	2500 x 2119	23.19
663 kb	1280 x 1085	19.81
389 kb	800 x 868	18.2
144 kb	360 x 305	18.75

Table 7.5:3 Scanned maps result for analysis duration testing

When going through the results obtained by analyzing the test image set, it can be observed that the image size and dimension plays a large role in the duration of analysis the longest duration of 23.19 s was recorded analyzing the scanned maps image of 1527 kb size. When Comparing the durations of analysis between classes of images, scanned maps clearly do take longer to analyze. The reason for this being that the suitable parameter range for scanned maps are much larger than the parameter ranges available in the other 2 categories therefor all parameters must be fitted to, to get the best parameters to convert the image.

Therefore, the conclusion can be made that, while image size and dimensions of an image matter when the analysis is being performed on it for its process duration. The suitable parameter range for that particular image class can also affect the image analysis time.

7.6 Testing Image Conversion Performance of Flask API Backend

This testing method analyzes the time taken for an image to be converted from raster to vector with the parameters auto selected by the platform according to its image size. The data is collected over several iteration of the process by checking duration taken to convert images of various image dimensions. The images were prepared for this task using an image editing tool.

Note: the images used were chosen at random

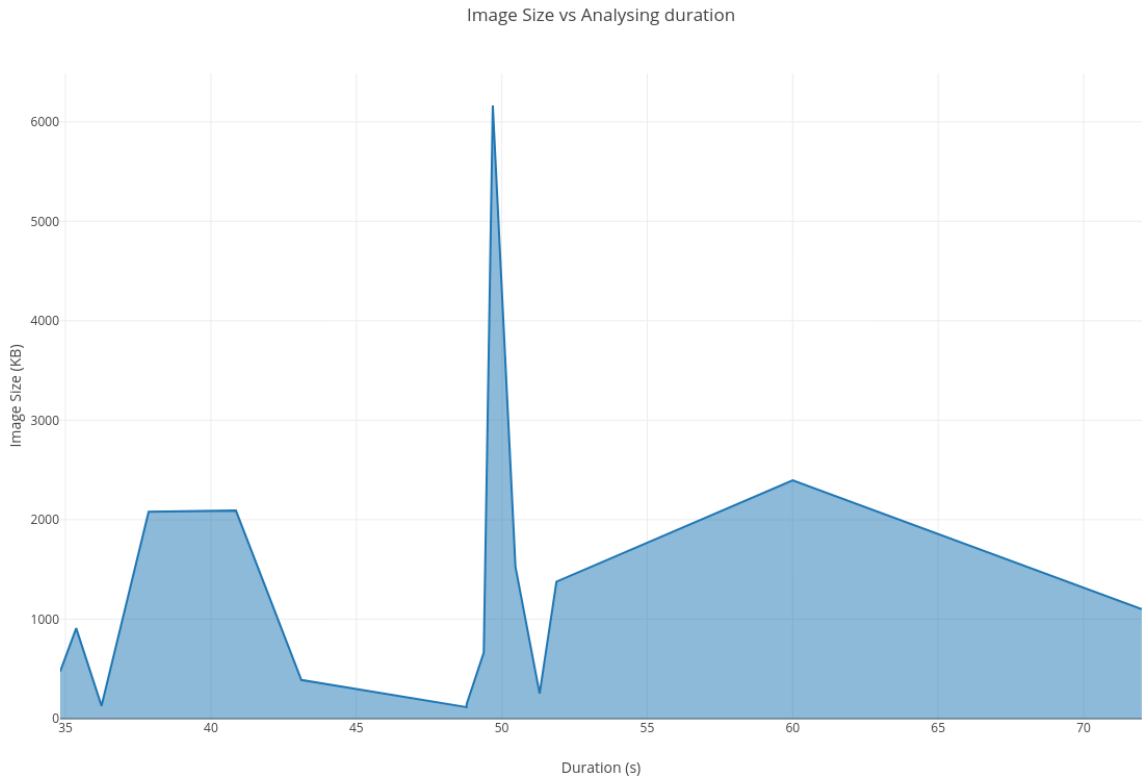


Figure 7.6:1 Image size vs conversion duration

Image 1: Land classification map

File size	Dimensions	Duration
2080 kb	3720 x 2480	37.86
2092 kb	2500 x 1667	40.86
909 kb	1280 x 854	35.37
474 kb	300 x 534	34.82
129 kb	360 x 240	36.24

Table 7.6:1 Land classification results for conversion duration

Image 2: Satellite Imagery

File size	Dimensions	Duration
1378 kb	1024 x 1024	51.88
6165 kb	2500 x 2500	49.69
2395 kb	1280 x 1280	60
1012 kb	800 x 800	72
253 kb	360 x 360	51.3

Table 7.6:2 Satellite image results for conversion duration

Image 1: Scanned Maps

File size	Dimensions	Duration
116 kb	354 x 340	48.79
1527 kb	2500 x 2119	50.47
663 kb	1280 x 1085	49.38
389 kb	800 x 868	43.11
144 kb	360 x 305	48.79

Table 7.6:3 Scanned map results for conversion duration

As it can be observed in the test results noted down above for each image and the time taken for it to be converted using the auto generated parameters of the system. It can be identified that roughly around 40-50 second averages can be seen in the duration of conversion of image of any dimension. This shows that the dimensions and file size of the image does not affect the conversion duration. There are several anomalies where there is60 and 72 seconds recorded as the time taken to convert. Therefore, further testing under different conditions will have to be carried out to understand why these occur.

7.7 Testing structural similarity between input image and converted vector image.

This phase will carry out a test to determine the structural similarity index between the original image that was uploaded by the user for conversion and the vectorized image after conversion. The image will be converted through the use of the platform using the auto selected best conversion parameters and no parameters will be changed. The testing will be done with de noising and without de-noising separately.

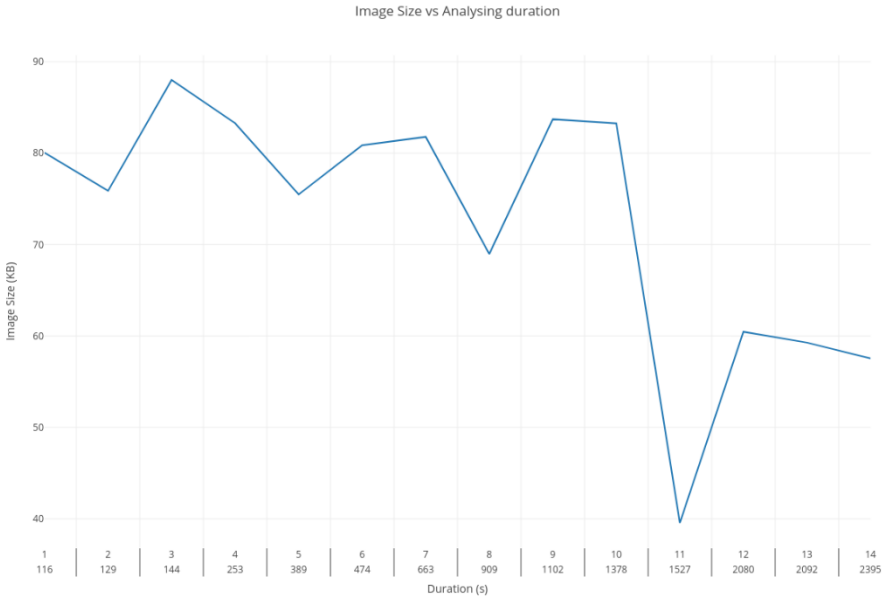


Table 7.7:1 Structural similarity between input image and converted vector image

When observing the results of the graph drawn against SSIM (similarity index) and the file size and index of conversion using the platform it can be seen that an average of 68% to 88% is maintained in the conversion process between the input image and the output. There were several anomalies that occurred during this process. This was due upscaling of the original image which was of very low quality to a higher resolution for testing purposes with the image editing tool. These upscaled images did not produce good results, while its original low resolution image produced 83% Similarity. Therefore, it can be said that the anomalies can occur in rare cases where an image is upscaled and converted when the initial image size is very low. But a better conversion can be obtained when just converting the original image resolution with its details.

References

Al-Bayari, O. (2018) 'GIS cloud computing methodology', *CITS 2018 - 2018 International Conference on Computer, Information and Telecommunication Systems*. IEEE, pp. 1–5. doi: 10.1109/CITS.2018.8440176.

Berry, J. K. (1995) 'Raster is Faster, But Vector is Corrector', *GIS World*, 8(6), p. 35. Available at: https://www.esri.com.

Couclelis, H. (1992) 'People Manipulate Objects but Cultivate Fields', *Proceedings from International Conference on GIS*, pp. 65–76. Available at: http://clamsitel.pbworks.com/w/file/fetch/49252349/Couclelis 1992 Objects Fields.pdf.

Hilaire, X. and Tombre, K. (2006) 'Robust and accurate vectorization of line drawings', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(6), pp. 890–904. doi: 10.1109/TPAMI.2006.127.

Lacroix, V. (2009) 'Raster-to-vector conversion: problems and tools towards a solution a map segmentation application', *Proceedings of the 7th International Conference on Advances in Pattern Recognition, ICAPR 2009*, 1(c), pp. 318–321. doi: 10.1109/ICAPR.2009.96.

Lin, F. and Guo, C. (2011) 'Raster-vector integration based on SVG on mobile GIS platform', *Proceedings - 2011 6th International Conference on Pervasive Computing and Applications, ICPCA 2011*, pp. 378–383. doi: 10.1109/ICPCA.2011.6106534.

Liu, W. and Josep Lladós (Eds.) (2005) *Graphics Recognition: Ten Years Review and Future Perspectives*, *6th International Workshop, GREC 2005 Hong Kong, China, August 25-26, 2005 Revised Selected Papers*. doi: 10.1007/11767978_5.

Seel-audom, C., Naiyapo, W. and Chouvatut, V. (2017) 'A search for Geometric shapes in a Vector Image', *9th International Conference on Knowledge and Smart Technology (KST)*, pp. 305–310.

SEVERENUK, T. *et al.* (2019) 'VECTOR GRAPHICS BASED LIVE SKETCHING METHODS AND SYSTEMS'.

Sloan, K. R. and Tanimoto, S. L. (1979) 'Progressive Refinement of Raster Images', *IEEE Transactions on Computers*, C–28(11), pp. 871–874. doi: 10.1109/TC.1979.1675269.

Winnemoeller, H. *et al.* (2018) 'ENHANCED VECTORIZATION OF RASTER IMAGES'.

Wong, D. W. S. and Wu, C. V. (1996) 'Spatial metadata and GIS for decision support', *Proceedings of the Annual Hawaii International Conference on System Sciences*, 3, pp. 557–566. doi: 10.1109/HICSS.1996.493251.