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

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

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.

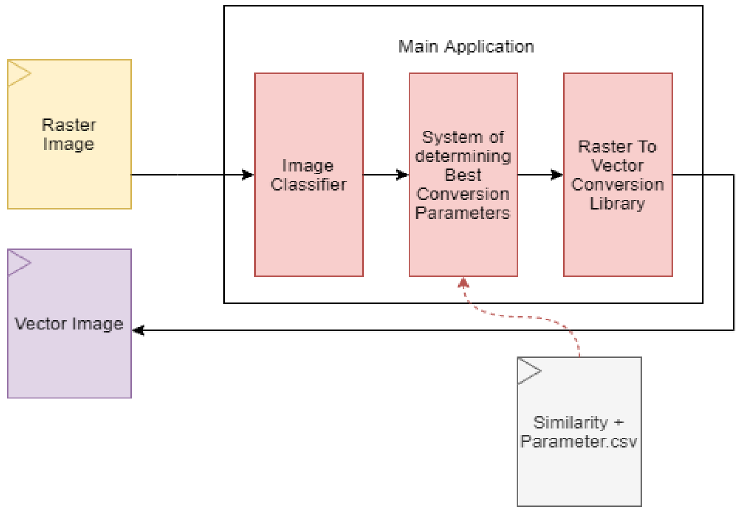
2.1 In-Scope

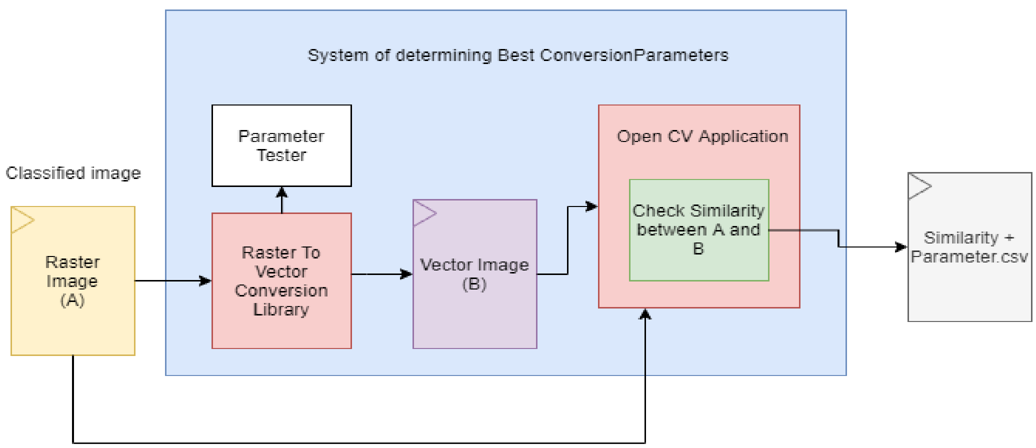
- Raster to Vector conversion tool is only developed geared towards GIS
- Training an image processing model to identify certain properties of images that affect conversion algorithms.
- Integrating Image processing for the identification special characteristics to identify the best conversion algorithm
- 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

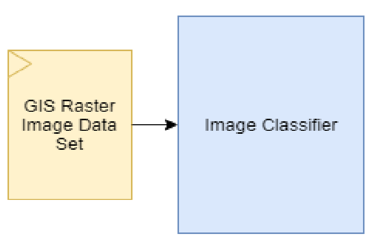
2.2 Out-Scope

- 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. Implementation of Components

5.1 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.

5.1.1 Image Resizing

```python
# resize Image
def image_resize(image_to_resize: object, set_width: int):
    width = int(image_to_resize.shape[1] * set_width / image_to_resize.shape[1])
    height = int(image_to_resize.shape[0] * set_width / image_to_resize.shape[1])
    dimensions = (width, height)
    return cv2.resize(image_to_resize, dimensions, interpolation=cv2.INTER_AREA)

```

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.

5.1.2 Image Colour Quantization

```python
# quantize image to reduce colour data for easy raster to vector conversion
def quantize_image(image: object, number_of_colors: int):
    (h, w) = image.shape[:2]
    image = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)
    image = image.reshape((image.shape[0] * image.shape[1], 3))
    clt = MiniBatchKMeans(n_clusters=int(number_of_colors))
    labels = clt.fit_predict(image)
    quantized_image = clt.cluster_centers_.astype("uint8")[labels]
    quantized_image = quantized_image.reshape((h, w, 3))
    quantized_image = cv2.cvtColor(quantized_image, cv2.COLOR_LAB2BGR)
    # return resized image
    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.

5.1.3 Converting SVG into PNG

```
def convert_to_png(in_path: str, out_path: str):
    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.

5.1.4 SVG2PNG.bat

```
cd libraries/inkscape/bin
set arg1=%1%
set arg2=%2%

set path1=%arg1%
set path2=%arg2%

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.

5.1.5 Image Structural Similarity Index Calculator

```
def check_similarity(path_1: str, path_2: str, image_size):
    img1 = cv2.imread(path_1)
    img2 = image_resize(cv2.imread(path_2), image_size)

    w1, h1 = img2.shape[:-1]
    img1 = cv2.resize(img1, (h1, w1))

    s = ssim(img1, img2, multichannel=True)
    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.

5.1.6 Variable Trainer Process

```
def variable_full_process(category):
    image_size = 720
    end_file_type = ".svg"

    # Initialize directories for category
    try:
        os.mkdir("images/png/" + category)
        os.mkdir("images/tempimages/" + category)
        os.mkdir("images/converted/" + category)
        os.mkdir("csv/" + category)
    except OSError as e:
        print(e)

    # Load necessary paths for the reading and writing
    data_path = "images/data/" + category
    temp_path = "images/tempimages/" + category
    out_svg_base_path = "images/converted/" + category
    png_base_path = "images/png/" + category

    csv_path = 'csv/' + category

    for img in os.listdir(data_path):

        image = cv2.imread(os.path.join(data_path, img))
        image = image_resize(image, image_size)
        image = quantize_image(image, 16)

        image_name = img.split(".")[0]
        image_extension = img.split(".")[1]

        try:
            os.mkdir(os.path.join(temp_path, image_name))
            os.mkdir(os.path.join(out_svg_base_path, image_name))
            os.mkdir(os.path.join(png_base_path, image_name))
        except OSError as e:
            print(e)

        out_svg_specific_path = os.path.join(out_svg_base_path, image_name)

        sample_path = os.path.join(temp_path, image_name, "sample." + image_extension)
        save_image_to_path(sample_path, image)

        index = 0
```

```python

    with open(csv_path + "/" +image_name + '.csv', 'w', newline='') as f:
        thewriter = csv.writer(f)

        thewriter.writerow(["index", "ltres", "qtres", "pathomit", "file_path", "similarity"])

        for ltres in range(0, 9, 2):
            for qtres in range(0, 9, 2):
                for pathomit in range(0, 101):
                    if pathomit == 1 or pathomit == 10 or pathomit == 100:
                        print("progress:" + str(int(index / 75 * 100)) + "%")

                        index = index + 1

                        subprocess.call([
                            'java', '-jar', 'libraries/imageTrace.jar', sample_path,
                            'outfilename', out_svg_specific_path + "/" + str(index) + end_file_type,
                            'pathomit', str(1 / pathomit),
                            'ltres', str(ltres),
                            'qtres', str(qtres),
                            'colorsampling', str(0),
                            'colorquantcycles', str(16)
                        ])

                        png_out_path = "images/png/" + category + "/" + image_name + "/" + str(index) + ".png"

                        convert_to_png("../../../" + out_svg_specific_path + "/" + str(index) + end_file_type,
                                       "../../../" + png_out_path)

                        similarity_val = check_similarity(sample_path, png_out_path, image_size)

                        thewriter.writerow(
                            [str(index), str(ltres), str(qtres), str(pathomit), str(index) + end_file_type,
                             str(similarity_val)])

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.

5.1.7 Create Best Fit Parameters for category

```python
def read_csv(category: str):
    base_file_path = 'csv'

    with open('final_csv/' + category + '.csv', 'w', newline='') as f:
        thewriter = csv.writer(f)

        thewriter.writerow(["index", "ltres", "qtres", "pathomit", "file_path", "similarity"])

        for csv_path in os.listdir(os.path.join(base_file_path, category)):
            df = pd.read_csv(os.path.join(base_file_path, category, csv_path), engine='python')
            df = df.sort_values('similarity', ascending=False)
            df.to_csv(os.path.join(base_file_path, category, "converted_" + csv_path), index=False)

        for csv_path in os.listdir(os.path.join(base_file_path, category)):
            if (csv_path.startswith("converted_")):
                input_file = csv.DictReader(open(os.path.join(base_file_path, category, csv_path)))

                # print highest 10 accracy values
                index = 0

                for row in input_file:
                    if index < 1:
                        thewriter.writerow(
                            [str(row['index']), str(row['ltres']), str(row['qtres']), str(row['pathomit']),
                             str(row['index']) + row['file_path'],
                             str(row['similarity'])])

                    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.

5.1.8 Cleanup

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

5.1.9 Full Training Process Executor

```
def finalize_params():
    category = 'sat'

    variable_full_process(category)
    read_csv(category)
    cleanup('csv', category)

```

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.

5.2 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.

5.2.1 Create and Prepare Training Data

```
def create_training_data(categories, data_dir, img_size, training_data):
    for category in categories:
        print("Image Loading from " + category + "...")
        path = os.path.join(data_dir, category)
        class_num = categories.index(category)
        for img in os.listdir(path):
            try:
                img_array = cv2.imread(os.path.join(path, img), cv2.IMREAD_GRAYSCALE)
                resized_array = cv2.resize(img_array, (img_size, img_size))
                training_data.append([resized_array, class_num])
            except Exception as e:
                print("Image Failed to Load")
                pass
    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.

5.2.2 Training Image Classification Model

```python
def train_save_model(training_data, img_size):
    print("starting to train data...")

    X = []  # images
    y = []  # labels

    for features, label in training_data:
        X.append(features)
        y.append(label)

    X = np.array(X).reshape(-1, img_size, img_size, 1)
    y = np.array(y)

    X = tf.keras.utils.normalize(X, axis=1)

    model = Sequential()

    model.add(Flatten(input_shape=X.shape[1:]))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(3, activation='softmax'))

    model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    model.fit(X, y, batch_size=32, epochs=20, validation_split=0.1)
    print(model.summary())
    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.

5.2.3 Training and Saving Model

```python
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.

5.3 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.

5.3.1 Flask API Initialization

```python
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)
```

```python
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.

5.3.2 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.

5.3.3 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.

5.3.4 Classify Image (Analyzer Class)

```python
def classify(path: str):
    my_model = load_model('classification_model/gis_classify.h5')
    prediction = my_model.predict(Analyzer.prepare_image(path))

    # ["LandClassificationMaps", "Satellite", "Scanned"]

    values = prediction[0]
    type = "unidentified"
    category= "unidentified"

    if values[0] == 1:
        type = "Land Classification Maps"
        category = "landclass"
    elif values[1] == 1:
        type = "Satellite Imagery"
        category = "sat"
    elif values[2] == 1:
        type = "Scanned Map"
        category = "scanned"

    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.

5.3.5 Colour Counter (Analyzer Class)

```python
def count_colours(src: str):
    image = cv2.imread(src)
    unique, counts = np.unique(image.reshape(-1, image.shape[-1]), axis=0, return_counts=True)
    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.

5.3.6 Determine Colour Clusters (Analyzer Class)

```python
def identify_quntize_cluster(colors: int):
    if colors < (2 ** 4):
        return 4
    elif colors < (2 ** 8):
        return 8
    elif colors < (2 ** 12):
        return 12
    elif colors < (2 ** 16):
        return 16
    elif colors < (2 ** 20):
        return 20
    elif colors < (2 ** 24):
        return 24
    elif colors < (2 ** 28):
        return 28
    else:
        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.

5.3.7 Get Best Parameters (Analyzer Class)

```python
def get_best_param_range(self, category):
    path = 'final_csv/'+category+'.csv'

    ltres_low = Analyzer.find_lowest(path, "ltres")
    ltres_high = Analyzer.find_highest(path, "ltres")

    qtres_low = Analyzer.find_lowest(path, "qtres")
    qtres_high = Analyzer.find_highest(path, "qtres")

    pathomit_low = Analyzer.find_lowest(path, "pathomit")
    pathomit_high = Analyzer.find_highest(path, "pathomit")

    return [('ltres_low', ltres_low), ('ltres_high', ltres_high), ('qtres_low', qtres_low),
            ('qtres_high', qtres_high),
            ('pathomit_low', pathomit_low), ('pathomit_high', pathomit_high)]

def find_lowest(path, prop_val):
    input_file = csv.DictReader(open(path))
    lowest = 0
    index = 0

    for row in input_file:
        if index == 0:
            lowest = row[prop_val]
        else:
            if row[prop_val] < lowest:
                lowest = row[prop_val]

        index += 1

    return lowest
```

```python
def find_highest(path, prop_val):
    input_file = csv.DictReader(open(path))
    highest = 0
    index = 0

    for row in input_file:
        if index == 0:
            highest = row[prop_val]
        else:
            if row[prop_val] > highest:
                highest = row[prop_val]

        index += 1

    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.

5.3.8 Determining Best parameters for user image (Analyzer Class)

```python
    def get_params(image_path, category):
        try:
            os.rmdir('temp/png')
        except OSError as e:
            print("does not exist")

        try:
            os.rmdir('temp/svg')
        except OSError as e:
            print("does not exist")

        try:
            os.mkdir('temp/svg')
        except OSError as e:
            print("does not exist")

        try:
            os.mkdir('temp/png')
        except OSError as e:
            print("does not exist")

        end_file_type = ".svg"
        param_range = Analyzer().get_best_param_range(category)
        image_size = 720

        image = cv2.imread(image_path)
        image = Analyzer.image_resize(image, image_size)
        image = Analyzer.quantize_image(image, 16)
        image_extension = imghdr.what(image_path)
        temp_image_path = 'temp/sample.' + image_extension
        Analyzer.save_image_to_path(temp_image_path, image)

        index = 0
```

```python

    with open('temp/temp.csv', 'w', newline='') as f:
        thewriter = csv.writer(f)

        thewriter.writerow(["index", "ltres", "qtres", "pathomit", "file_path", "similarity"])

        for ltres in range(int(param_range[0][1]), int(param_range[1][1]) + 1, 1):
            for qtres in range(int(param_range[2][1]), int(param_range[3][1]) + 1, 1):
                for pathomit in range(int(param_range[4][1]), int(param_range[5][1])):
                    if pathomit == 1 or pathomit == 10 or pathomit == 100:
                        index = index + 1

                        svg_out_path = "temp/svg/" + str(index) + end_file_type

                        subprocess.call([
                            'java', '-jar', 'imageTrace.jar', temp_image_path,
                            'outfilename', svg_out_path,
                            'pathomit', str(1 / pathomit),
                            'ltres', str(ltres),
                            'qtres', str(qtres),
                            'colorsampling', str(0),
                            'colorquantcycles', str(16)
                        ])

                        png_out_path = "temp/png/" + str(index) + ".png"

                        Analyzer.convert_to_png("../../../" + svg_out_path, "../../../" + png_out_path)
```

```python
                        similarity_val = Analyzer.check_similarity(temp_image_path, png_out_path, image_size)

                        thewriter.writerow(
                            [str(index), str(ltres), str(qtres), str(pathomit), str(index) + end_file_type,
                             str(similarity_val)])

    df = pd.read_csv('temp/temp.csv', engine='python')
    df = df.sort_values('similarity', ascending=False)
    df.to_csv(os.path.join('temp/temp_sorted.csv'), index=False)

    input_file = csv.DictReader(open('temp/temp_sorted.csv'))

    # get highest accuracy values
    index = 0

    highest_row = []

    for row in input_file:
        if index < 1:
            highest_row = row

        index += 1

    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.

5.3.9 Analyze (Analyzer Class)

```python
    def analyze(self):
        file_path = request.args.get('file_path')

        classification = Analyzer.classify(file_path)
        color_count = Analyzer.count_colours(file_path)
        quant_val = Analyzer.identify_quntize_cluster(color_count)

        row = Analyzer.get_params(file_path, 'landclass')# category[1]

        return jsonify(file_path=file_path, classification=str(classification[0]), color_count=color_count
                       , 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.

5.3.10 Converter Class

```python
class Converter:
    def convert(self):
        file_path = request.args.get('file_path')
        file_name = request.args.get('file_name')
        qtres = request.args.get('qtres')
        ltres = request.args.get('ltres')
        pathomit = request.args.get('pathomit')
        k = request.args.get('k')
        quant_bool = request.args.get('quantBool')
        end_file_type = ".svg"

        if quant_bool:
            Analyzer.quantize_image(cv2.imread(file_path))

        subprocess.call([
            'java', '-jar', 'imageTrace.jar', file_path,
            'outfilename', file_name + end_file_type,
            'colorsampling', str(0),
            'colorquantcycles', str(k),
            'ltres', str(ltres),
            'qtres', str(qtres),
            'pathomit', str(pathomit)
        ])

        return jsonify(progress="completed", file_name=file_name, file_path=file_path, out_path=file_name + end_file_type)
```

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.

5.4 Frontend

5.4.1 Initialization

```javascript
// 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.

```javascript
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.