Python GUI

Abstract

Developed a Graphical User interface (GUI) in python. The GUI can be used to select the raster file in TIFF (.tif) format. By selecting the type of image segmentation, Object Based Image Analysis can be done through k means clustering segmentation algorithm, Chessboard, QuickShift, QuickShift_ratio and slic. Currently, segmentation can be done through the GUI. The best means for OBIA implemented in the GUI is K means clustering. The show raster and show vector functions and the canvas was for the purpose of displaying geographical data, however it is not implemented currently. All other features work successfully. The main purpose of GUI was to perform Object Based Image segmentation that can be done through GUI in four different types of segmentation. The sample input file and output file can be found in corresponding folders within the GitHub repository that has been used for the test purpose. Using pyinstaller library there was an attempt to convert jupyter notebook file into exe file. Opening the executable file, the GUI works fine within the same user as it was created. However, transferring .exe file into other computer doesnot work without installing the dependencies. Therefore, for making the code standalone other means would be preferred e.g. Docker.

Requirements

Following are the packages used for creation of Graphical User Interface along with their purpose of use.

-tkinter

For Graphical User Interface

-opencv

For Kmeans Segmentation

-skimage

For other four types of segmentation

-gdal

Working with raster data

-numpy

For rasters to be processed as numpy arrays

-time

For calculating time on each process.

Literature Background:

GDAL

GDAL, also known as GDAL/OGR, is a library of tools used for manipulating geospatial data. GDAL works on both raster and vector data types, and is an incredible useful tool to be familiar with when working with geospatial data. While the GDAL library can be used programmatically, GDAL also includes a CLI (Command Line Interface).

A note about names

Traditionally, the name "GDAL" was used to refer to the raster-related half of the library, while "OGR" referred to the vector part. You may commonly hear people use both "GDAL" and "GDAL/OGR" to refer to this library - but in all cases, the toolset being referenced is the same.

Commonly-used commands

- ogrinfo - Get information about a vector dataset
- gdalinfo - Get information about a raster dataset
- ogr2ogr - Convert vector data between file formats
- gdal_translate - Convert raster data between file formats

Python provides different modules and libraries to make the projects interactive. One such attribute is the Graphical User Interface (GUI) which can be added to the projects using the Python libraries like Tkinter, PyQt5, Kivy and more. The Tkinter library is among the most common libraries for designing interactive GUI applications. It is lightweight and easy to use. We can operate this library on different platforms or Operating Systems like Windows, Linux, and macOS.

TKINTER PROGRAMMING

Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

Creating a GUI application using Tkinter is an easy task. All you need to do is perform the following steps:

- Import the Tkinter module.
- Create the GUI application main window.
- Add one or more of the above-mentioned widgets to the GUI application.
- Enter the main event loop to take action against each event triggered by the user.

Tkinter Widgets

Tkinter provides various controls, such as buttons, labels and text boxes used in a GUI application. These controls are commonly called widgets.

There are currently 15 types of widgets in Tkinter. We present these widgets as well as a brief description in the following table.

Sr.No.	Operator & Description
1	Button The Button widget is used to display buttons in your application.
2	Canvas The Canvas widget is used to draw shapes, such as lines, ovals, polygons and rectangles, in your application.
3	Checkbutton The Checkbutton widget is used to display a number of options as checkboxes. The user can select multiple options at a time.
4	Entry The Entry widget is used to display a single-line text field for accepting values from a user.
5	Frame The Frame widget is used as a container widget to organize other widgets.
6	Label The Label widget is used to provide a single-line caption for other widgets. It can also contain images.
7	Listbox The Listbox widget is used to provide a list of options to a user.
8	Menubutton The Menubutton widget is used to display menus in your application.
9	Menu The Menu widget is used to provide various commands to a user. These commands are contained inside Menubutton.
10	Message The Message widget is used to display multiline text fields for accepting values from a user.
11	Radiobutton The Radiobutton widget is used to display a number of options as radio buttons. The user can select only one option at a time.
12	Scale The Scale widget is used to provide a slider widget.
13	Scrollbar The Scrollbar widget is used to add scrolling capability to various widgets, such as list boxes.
14	Text The Text widget is used to display text in multiple lines.
15	Toplevel The Toplevel widget is used to provide a separate window container.
16	Spinbox The Spinbox widget is a variant of the standard Tkinter Entry widget, which can be used to select from a fixed number of values.

17	PanedWindow A PanedWindow is a container widget that may contain any number of panes, arranged horizontally or vertically.
18	LabelFrame A labelframe is a simple container widget. Its primary purpose is to act as a spacer or container for complex window layouts.
19	tkMessageBox This module is used to display message boxes in your applications.

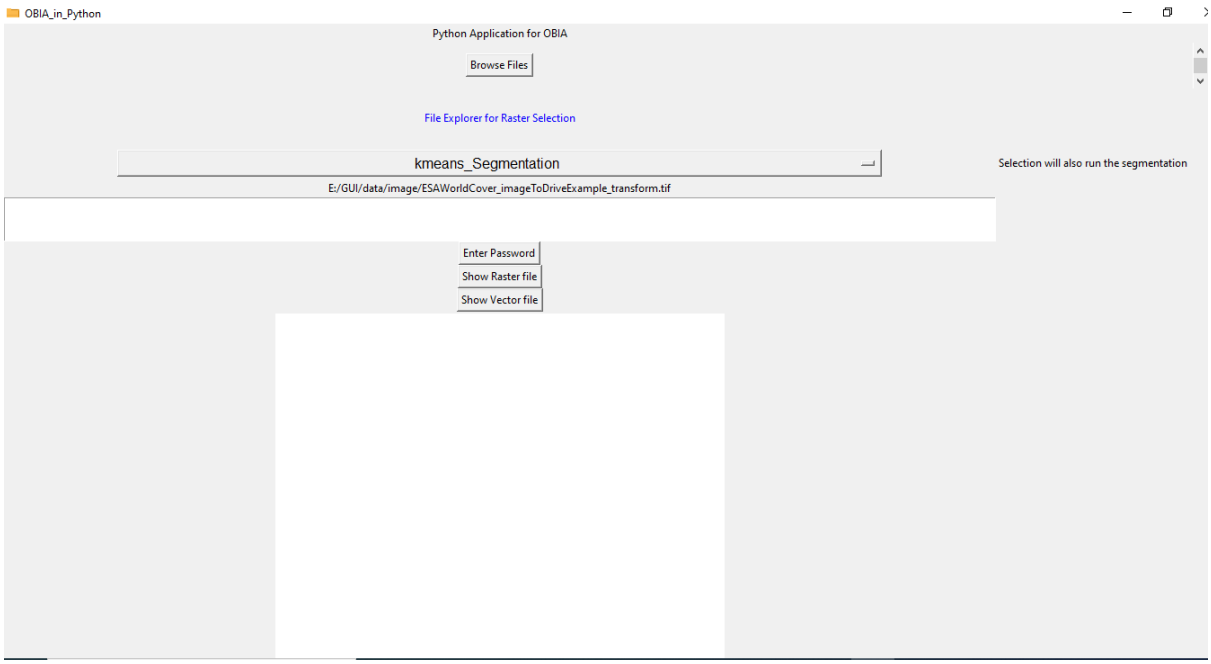
Geometry Management

All Tkinter widgets have access to specific geometry management methods, which have the purpose of organizing widgets throughout the parent widget area. Tkinter exposes the following geometry manager classes: pack, grid, and place.

- The pack() Method − This geometry manager organizes widgets in blocks before placing them in the parent widget.
- The grid() Method − This geometry manager organizes widgets in a table-like structure in the parent widget.
- The place() Method − This geometry manager organizes widgets by placing them in a specific position in the parent widget.

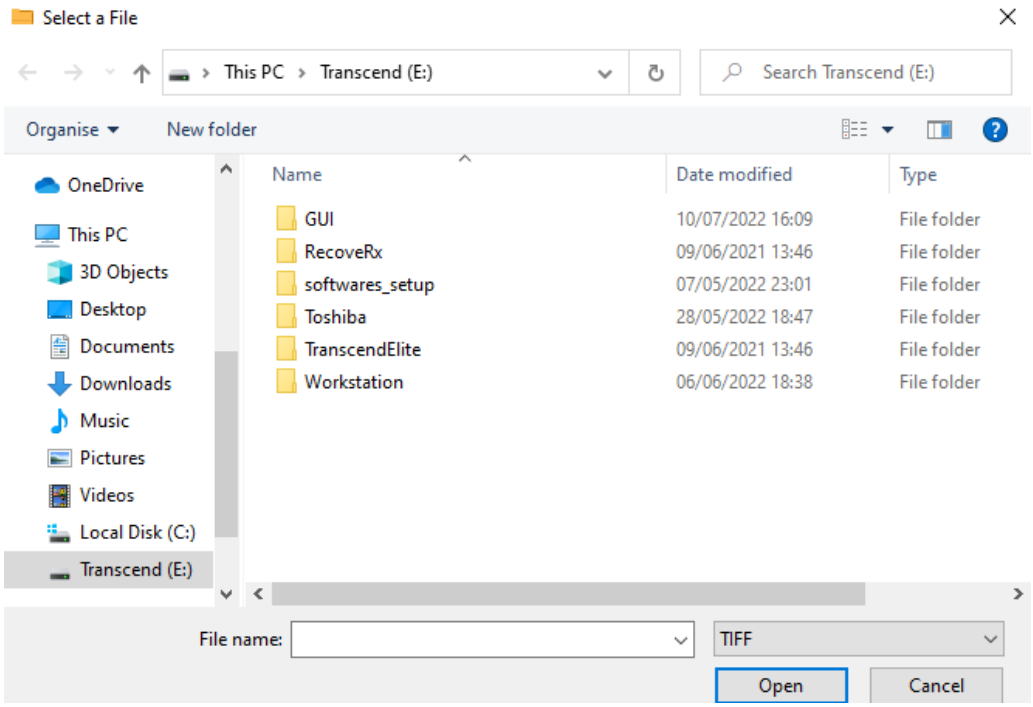
For the created GUI grid method was used for organizing the widget.

Result:



Graphical User Interface

Browse the image to be segmented through Browse File button

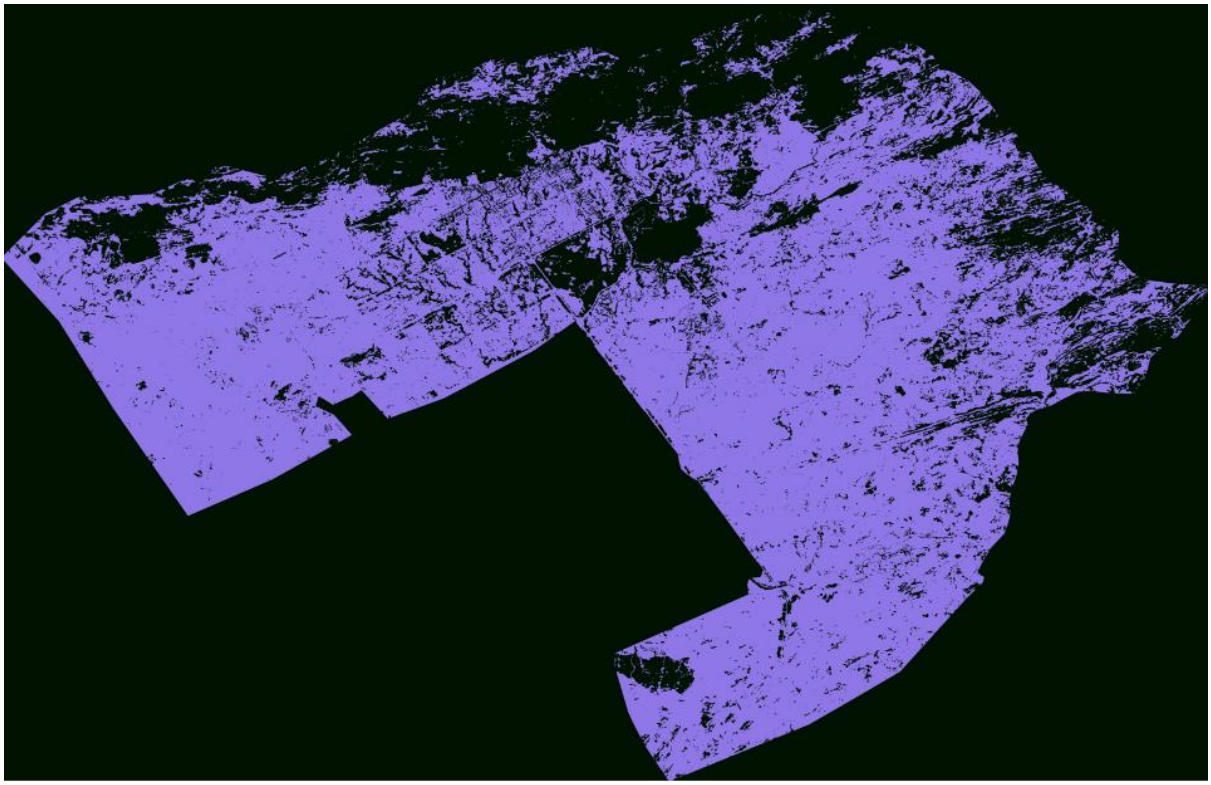


Click on the type of segmentation you want to run through option list, dropdown menu.

Currently, kmeans_Segmentation was selected.

After the selection, output folder window appears where one can select through file explorer. When the folder is selected, segmentation run's at the backend automatically.

Kmeans Segmentation output (generated through the GUI)



The segmentation can be viewed through python third party software such as Photos or QGIS.

Code:

```
import tkinter as tk
from tkinter import *
from tkinter import filedialog
from tkinter.filedialog import askdirectory

import cv2
import matplotlib as plt
import numpy as np



import gdal
from skimage import exposure
from skimage.segmentation import quickshift
import time



window = Tk()
window.title('OBIA_in_Python')
window.iconbitmap(r'ic.ico')
window.geometry("1000x1000")



"'Creating a label Widget'"
myHeading = Label(window, text='Python Application for OBIA')
"'Print on GUI such that its always in middle'"
myHeading.grid()

def myClick(var1):
    hello = "Hello "+ var1.get()
    myLabel = Label(window, text=hello)
    var1.delete(0,'end')
    myLabel.grid(column = 1, row = 1)



# Function for opening the
```

```
# file explorer window
def browseFiles():
    global filename
    filename = filedialog.askopenfilename(initialdir = "/",
                                          title = "Select a File",
                                          filetypes = (("TIFF","*.tif*"),
                                                       ("all files","*.*")))

    # Change label contents
    label_file_explorer.configure(text="File Opened: "+filename)



def showShapefile():
    #import geopandas
    #df_gpd = gpd.read_file(plot_centroid_path)
    #df_gpd.plot()
    print("Shapefile_Function")




label_file_explorer = Label(window, text = "File Explorer for Raster Selection",
                            width = 100, height = 4,
                            fg = "blue")

button_explore = Button(window, text = "Browse Files", command = browseFiles)






var1 = Entry(window, width=50, font =('Helvetica',30), show='*')
var1.grid(column = 0, row = 14)

myButton = Button(window, text="Enter Password", command=myClick)
myButton.grid(column = 0, row = 15)
```

```
label_file_explorer.grid(column = 0, row = 2)

button_explore.grid(column = 0, row = 1)



try:
    labelfilename = Label(window, text=filename) #shows as text in the window
    labelfilename.grid(column = 0, row = 6)
    print(filename)

except:
    print("Not now")




myButton2 = Button(window, text="Show Vector file", command=showShapefile)
myButton2.grid(column = 0, row = 18)

myButton3 = Button(window, text="Show Raster file", command=showShapefile)
myButton3.grid(column = 0, row = 16)



"""For Drop Down Menu"""
hello = "Selection will also run the segmentation "
myLabel = Label(window, text=hello)
myLabel.grid(column = 1, row = 5)

def seg(choice):
    global outputfilename
    outputfoldername = filedialog.askdirectory()
    print (outputfoldername)

    if choice=="kmeans_Segmentation":
        path = filename
        img = cv2.imread(path)
```

```
    img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
    twoDimage = img.reshape((-1,3))
    twoDimage = np.float32(twoDimage)
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
    K = 2
    attempts=10
    ret,label,center=cv2.kmeans(twoDimage,K,None,criteria,attempts,cv2.KMEANS_PP_CENTERS)
    center = np.uint8(center)
    res = center[label.flatten()]
    result_image = res.reshape((img.shape))
    cv2.imwrite(str(outputfoldername)+ '/segmented_result_Kmeans.tif',result_image)



elif choice=="Chessboard":
    myLabel = Label(window, text=choice)
    myLabel.grid(column = 1, row = 16)

    print(button_explore)

elif choice=="QuickShift" or "QuickShift_ratio" or "slic":
    myLabel = Label(window, text=choice)
    myLabel.grid(column = 1, row = 16)

    naip_fn = filename
    driverTiff = gdal.GetDriverByName('GTiff')
    naip_ds = gdal.Open(naip_fn)
    nbands = naip_ds.RasterCount
    band_data = []
    print('bands', naip_ds.RasterCount, 'rows', naip_ds.RasterYSize, 'columns',
          naip_ds.RasterXSize)
    for i in range(1, nbands+1):
        band = naip_ds.GetRasterBand(i).ReadAsArray()
        band_data.append(band)
    band_data = np.dstack(band_data)
    # scale image values from 0.0 - 1.0
```

```
        img = exposure.rescale_intensity(band_data)

        # do segmentation multiple options with quickshift and slic
        seg_start = time.time()
        if choice=="Quickshift":
            segments = quickshift(img, convert2lab=False)

        elif choice=="Quickshift_ratio":
            segments = quickshift(img, ratio=0.8, convert2lab=False)

        elif choice=="slic":
            segments = slic(img, n_segments=100000, compactness=0.1)

        #segments = slic(img, n_segments=500000, compactness=0.1)
        print('segments complete', time.time() - seg_start)
        # save segments to raster
        segments_fn = str(outputfoldername)+ '/segmented_result_'+str(choice)+'.tif'
        segments_ds = driverTiff.Create(segments_fn, naip_ds.RasterXSize, naip_ds.RasterYSize, 1, gdal.GDT_Float32)
        segments_ds.SetGeoTransform(naip_ds.GetGeoTransform())
        segments_ds.SetProjection(naip_ds.GetProjectionRef())
        segments_ds.GetRasterBand(1).WriteArray(segments)
        #Flush Cache here
        segments_ds = None
        del(segments_ds)
        #segments_ds_fn.SyncToDisk()

    print(choice)
OptionList = ["kmeans_Segmentation","Chessboard","QuickShift", "QuickShift_ratio", "slic"]
variable = tk.StringVar(window)
variable.set(OptionList[0])
opt = tk.OptionMenu(window, variable, *OptionList, command=seg)
opt.config(width=90, font=('Helvetica', 12))
opt.grid(column = 0, row = 5)
print(opt)
"Show Canvas"
myCanvas = Canvas(window, width=500,height=500, background="White")
```

```
myCanvas.grid(column = 0, row = 19)
scroll_bar = Scrollbar(window)
scroll_bar.grid(column = 2, row=1)
window.mainloop()
```