BRNDTS Technical Challenge - User Guide

1 Content

There are 3 main methods available. This could have been all merged into a single method of generation, but I opted to keep them separate to better convey the processing pipeline.

- `advert_locations`: takes the destination RGB image as input and outputs two lists: one with the possible locations and one with the possible locations with a perspective correction.
- `segment_occlusions`: the input is the same as before, but now the output is a composite mask of the occlusions.
- `gen_advert`: final method that joins everything and returns the image with the ad. It takes the RGB image for the ad and the destination, the possible locations (raw and fixed for perspective) and the occlusions' mask. Additionally, there are several parameters that can be tuned for a better final result:
  - `down_scale` (float): Reduces the size of the image to account for the border.
  - `border_t` (float): Border thickness.
  - `border_color` (tuple (R, G, B)): Color of the border.
  - `warp (bool)`: If True, fixes the perspective.
  - `bordered (bool)`: If True, adds a border.
  - `color_correction` (str): Mode of color correction:
    - `'histogram'`: Matches the histogram of the advertisement with the background image.
    - `'color_transf'`: Adjusts the colors using the mean and standard deviation of the background image.
  - `smoothing` (float): Matches the ad apparent resolution to the source's and smoothes.
  - `mode` (str): Different mode of applications of the ad:
    - `'area'`: Looks for the location with the bigger area.
    - `'random'`: Looks for a random location of the list of locations.
    - `'total'`: Applies the ad to all the locations in the list.

For more information regarding the methods, check their docstrings.

All the needed methods and helper functions can be found inside the backbone `BRNDTS.py` class. I opted for the design choice of keeping the static methods inside the class instead of placing them in a utils module because, although they do not directly interact with an instance of the class, they are required by the class's methods.

Inside the utils folder there is a `utils.py` file that contains a helper function (load_image) that loads all the images and converts them from the default BGR to RGB. This function takes as input a single image or a directory of images and returns a generator that can be convinently placed in a loop for further processing.

2 The config.py file

This file contains all the parameters that are required by the program's methods. The default values are defined for the optimal results for this problem, but they can be easily overridden by passing a new value to the `gen_advert` function. To check what can be defined consult the method's docstring.

- MODEL_PATH = 'models/yolov8s-oiv7.pt'
- SEG_MODEL_PATH = 'models/yolov8s-seg.pt'
- DEVICE = 'gpu'
- SOURCE = 'source/image.png'
- ADVERT = 'source/advert.jpeg'
- ALPHA = 2.5
- BETA = 0
- SP = 15
- CR = 21
- T = 100
- SET_PX = 255
- CONF_SCORE = 0.001
- SEG_CONF_SCORE = 0.5
- DOWN_SCALE = 0.95
- BORDER_T = 0.03

- BORDER_COLOR = (0,0,0)
- WARP = False
- BORDERED = True
- COLOR_CORRECTION = 'histogram'
- SMOOTHING = 0.9
- MODE = 'area'
- SAVE = True

3 How to run the code

To run the code, first instantiate the `BRNDTS` class and load the advert using a `next()` call. To process, simply create a loop that iterates over the `load_image` generator and take its return to pass to the next methods. Then follow the pipeline sequence and call the respective functions: first the `advert_locations`, then the `segment_occlusions` and finally the `gen_advert`. An example can be found here:

```
import numpy as np
import cv2
import matplotlib.pyplot as plt

from BRNDTS import BRNDTS
from config import SOURCE, ADVERT
from utils.utils import load_image

ad_gen = BRNDTS()

advert = next(load_image(ADVERT)) #load_image creates a generator,
                                  #hence the need for the next() call

for image in load_image(SOURCE):
    cnt_list, warp_boxes = ad_gen.advert_locations(image) #Detection
    seg_occlusion_mask = ad_gen.segment_occlusions(image) #Occlusion removal
    final_image = ad_gen.gen_advert(advert, image, cnt_list,
                                    seg_occlusion_mask, warp_boxes,
                                    mode = 'area', color_correction = None,
                                    smoothing = None) #Advert insertion stage
```