# **Workflow Raphael**

**Image Preprocessing**

**Raphael painting? (11\_authenticate.py)**

- Check whether and which painting is made by Raphael, and which don’t

- Use code as part of the 2023 article “Deep transfer learning for visual analysis and attribution of paintings by Raphael”

**Image size analysis (12\_painting\_phot\_sizes.py)**

* Extract information of each painting on pixel size, resolution dpi, size in cm
* Input: raw images (jpeg)
* Use external data (Dimensions Mismatch.xlsx) to correct image size, if possible
* TODO (Looking for best approach to deal with cropping, etc)
* Output: resized images, table (in excel) with information

**Animation (13\_animation.py)**

* See how there is rotation present in the copy compared to the original. Create an animation that smoothly transitions between two similar images, using image registration techniques and interpolation to align the outlines of the figures correctly. We consider using algorithms like RANSAC for robust alignment.
* Input: raw images (jpeg)
* Output: on screen

**Degree of rotation present between two images (14\_rotation.py)**

- Determine rotation between the central figures in two paintings relative to each other

- Input: raw images (jpeg)

- Output: heatmaps of degrees of rotation

**Image alignment and registration (image alignment.py)**

* Align all images relative to the original for accurate extraction and comparison of segmented figures
* Remove rotation to enable better comparison
* Input: raw images (jpeg)
* Output: Saved in data/interim/aligned
* <https://pyimagesearch.com/2020/08/31/image-alignment-and-registration-with-opencv/>

**2. Outline Extraction**

**Background removal (background removal.py)**

* Remove background so that all that remains are the main figure in the foreground
* Input: the aligned paintings
* Output: colored images, saved in data/interim/no\_background

**Foreground extraction (foreground extraction.py)**

* Further extraction from the main figures
* Input: the paintings without background
* Output: Foregrounded images saved in data/interim/foreground

**Face extraction (figure extraction.py)**

* haarhascade opencv
* bbox per figure
* explore alternatives HOG and COCO

https://www.digitalocean.com/community/tutorials/how-to-detect-and-extract-faces-from-an-image-with-opencv-and-python

https://stackoverflow.com/questions/44360643/simple-implemented-method-for-human-silhouette-extraction-python-opencv

https://pyimagesearch.com/2015/11/09/pedestrian-detection-opencv/

https://www.tutorialspoint.com/how-to-detect-humans-in-an-image-in-opencv-python

**Image segmentation (yolo.py)**

* create individual objects (i.e., outlined figures) for comparison
* <https://www.christopherminson.com/articles/extract.html>
* <https://github.com/cminson/public/blob/master/extract/extract.py>
* Use yolov8 for image segmentation
* Use image inpainting to remove foregrounded figure

https://stackoverflow.com/questions/65787046/carving-out-exactly-humans-from-image

https://stackoverflow.com/questions/44360643/simple-implemented-method-for-human-silhouette-extraction-python-opencv

https://pyimagesearch.com/2015/11/09/pedestrian-detection-opencv/

https://www.tutorialspoint.com/how-to-detect-humans-in-an-image-in-opencv-python

<https://stackoverflow.com/questions/65787046/carving-out-exactly-humans-from-image>

**Remove overlapping objects (inpainting.py)**

* <https://github.com/sujaykhandekar/Automated-objects-removal-inpainter>
* <https://medium.com/analytics-vidhya/removing-objects-from-pictures-with-deep-learning-5e7c35f3f0dd>

**3. Figure outline preprocessing**

**Preprocessing figures and deriving measures**

* adaptive thresholding
* histogram equalization
* erosion and dilation for morphological operations
* <https://www.geeksforgeeks.org/erosion-dilation-images-using-opencv-python/>
* bilateral filter (BF) to smooth the images, further aiding the detection of candidate keypoints using the SIFT algorithm
* Calculate aspect ratio and solidity using contour analysis
* Kmeans color reduction
* Morphological operations in image processing (gradient)

**Edge detection interactive (edge\_detection\_interactive.py)**

- Determining the best canny edge detection parameters for each individual painting to obtain the optimal edges from the aligned main figures. The edges and contours of the foreground figures are later input for SIFT/SURF/ORB comparison.

- Input: segmented figures

- Output: Outlined figures preprocessed (gray, optimal edges and contours), saved separately in data/interim/canny\_edges

**Contour analysis interactive (contour\_interactive.py)**

* Outer contour extraction
* Inner contour extraction
* Save outer contour as mold

**4. Template outline comparison**

* Sift features
* Proportion comparison (akin to Rick Johnson’s approach)
* Surface area comparison (oppervlakte vergelijking, rekening houdend met image size)
* Invariant template matching (invariant to size and rotation of template; focusing on similarity of the template *as a whole*)
* Similarity measures
* Brushstroke comparison (focusing on similarity *within* the template)

**SIFT feature comparison (sift\_bf\_heatmap.py)**

- Detect SIFT features and compute descriptors on the Canny edge images.

- Filter keypoints to keep only those located on the edges.

- The number of matching feature points can be used as a measurement of similarity between pairs of figures.

- Use bruteForce, FLANN matching. Match keypoints using FLANN and PROSAC:

- Adopt the fast library for approximate nearest neighbors (FLANNs) algorithm to search the matching points in the high-dimensional space.

- In order to improve the matching accuracy, the progressive sample consensus (PROSAC) algorithm is utilized to exclude erroneous matching points.

- Input: the preprocessed, foregrounded figures in canny edge format

- Output: heatmap

- TODO See main script

**Proportion comparison (main\_script3.py)**

- Calculate the proportions of the main figures based on the extracted SIFT features

-> use good matches

- We use the pairwise distances between all keypoints within the main figures

- Combine distances into one measure (e.g., average) and normalize by the diagonal length of the bounding box

- TODO quantify the similarity between the proportions of the main figures

- See main\_script 3

- compare with rick Johnson approach

**Surface area comparison (object size.py)**

- https://pyimagesearch.com/2016/03/28/measuring-size-of-objects-in-an-image-with-opencv/

**Invariant template matching (template matching.py)**

- Use the individual figures of the original as a template to match on the copies

- <https://stackoverflow.com/questions/58920750/how-to-perform-scale-and-rotation-invariant-templatefeature-matching-and-objec>

- <https://github.com/cozheyuanzhangde/Invariant-TemplateMatching>

- <https://github.com/brkyzdmr/TemplateMatcher/blob/main/template_matcher.py>

- <https://pyimagesearch.com/2015/01/26/multi-scale-template-matching-using-python-opencv/>

- <https://www.geeksforgeeks.org/template-matching-using-opencv-in-python/>

**Smilarity measures (Image similarity measures.py)**

- Comparing images using ORB similarity, structural similarity index, cosine similarity, and several other measures -> Quantitative assessment of figure similarity across paintings

- Input: preprocessed main figures, canny edge and/or contours without colors

- Output: Results in heatmap; heatmap for each measure

- TODO integrate image correlation.py

- <https://copyprogramming.com/howto/checking-images-for-similarity-with-opencv>

**Brushstroke comparison (brushstroke comparison.py)**

* The brushstrokes within the segmented outline carry the essence of an artist’s technique, and is often manifested as edges in an image. Therefore, capturing these subtle cues through edge detection can provide insights into the similarity between paintings.
* Combined edge detection techniques, a cmprehensive weighted representation of each painting’s low level visual features (original and copies). This is a combination of:
  + Canny edge
  + Sobel operator
  + Laplacian of Gaussian
  + Scharr Operator
* Standard deviation to quantify the variability and intensity of edge features in the image
* Sd comparison: sd’s similar -> similar brushstrokes -> more identical templates

**Deep image search (deep image search.py)**

Use more advanced tooling to rank the similarities between the paintings

- Input: the preprocessed figures as templates (in color)

- Output: ranking of similar figures

**5. Deployment**

**Streamlit app (app.py)**

Make an app that showcases similar images based on a selected image

- Adapt/customize to make it flexible in terms of

- image

- focus on complete image versus outline/ figure comparison

- similarity measure

- pipeline, with image segmentation, contour analysis etc in one go