Software design and development year 12

Assessment task 2

Noah Hallows

Identification of the problem:

In the modern world there are a significant number of photos taken by people all the time. This poses significant privacy and possible safety concerns as anyone will be able to access information about your appearance, location history and friends or someone could use these pictures to make sexually explicit content of you using AI. However, searching through large numbers of photos to remove your face is not a viable solution. My solution is an automated system that uses open-source face recognition to search through large numbers of images and then deletes or edits the images with the users face to anonymize them.

Generation of ideas:

I have always been concerned with my digital privacy and since I have stopped being home schooled, I have had significantly more photos of me taken by other people. I found inspiration for my system when I was watching a Japanese cyberpunk anime one of the characters would hack computer systems to place a logo over their face to preserve their anonymity, which gave me the idea to do a similar thing with python.

Requirements of the solution

- Accurately identify faces of any demographic
- Accurately blur/remove these faces without destroying the whole image
- Only run on the specified images
- Have a low difficulty bar to use
- Run quickly on small numbers of images

Communication with others:

- Ana Williams (teacher)
- Brendon Godfrey
- Zeek Mattaronno
- John Hallows (father)
- Chris Web

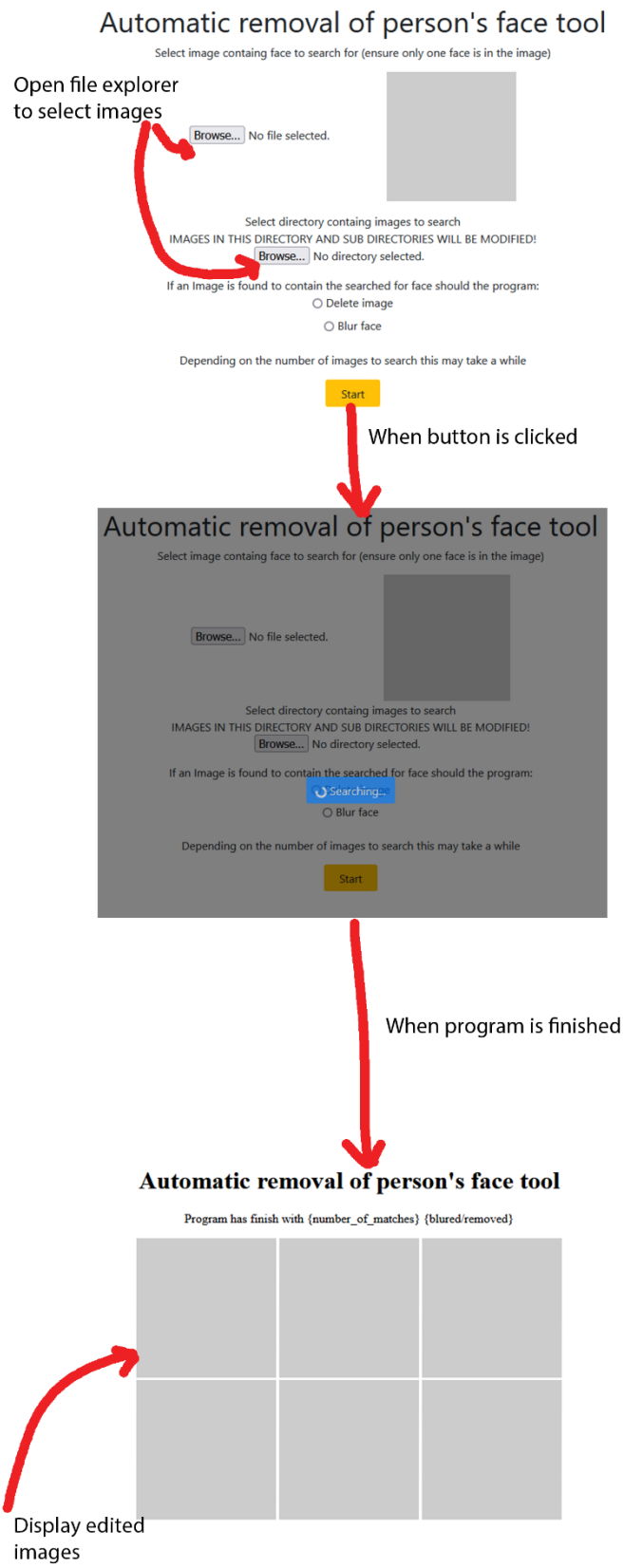
Selection of appropriate data types and data structure:

This software solution will use two main data types, strings to store the location of images and numpy arrays to store the processed images.

- Image containing face to search for – ARRAY

- Images to search – ARRAYS
- Location of image containing face to search for – STRING
- Location of images to search – String

Draft interface design: I used html, css and javascript to make a mockup interface design the screenshot doesn’t display the whole design.


Automatic removal of person's face tool
Select image containg face to search for (ensure only one face is in the image)
Open file explorer
to select images
Browse... No file selected.
Select directory containg images to search
IMAGES IN THIS DIRECTORY AND SUB DIRECTORIES WILL BE MODIFIED!
Browse... No directory selected.
If an Image is found to contain the searched for face should the program:
Delete image
Blur face
Depending on the number of images to search this may take a while
Start
When button is clicked
Automatic removal of person's face tool
Select image containg face to search for (ensure only one face is in the image)
Browse... No file selected.
Select directory containg images to search
IMAGES IN THIS DIRECTORY AND SUB DIRECTORIES WILL BE MODIFIED!
Browse... No directory selected.
If an Image is found to contain the searched for face should the program:
Searching...
Blur face
Depending on the number of images to search this may take a while
Start
When program is finished
Automatic removal of person's face tool
Program has finish with {number_of_matches} {blured/removed}
Display edited
images

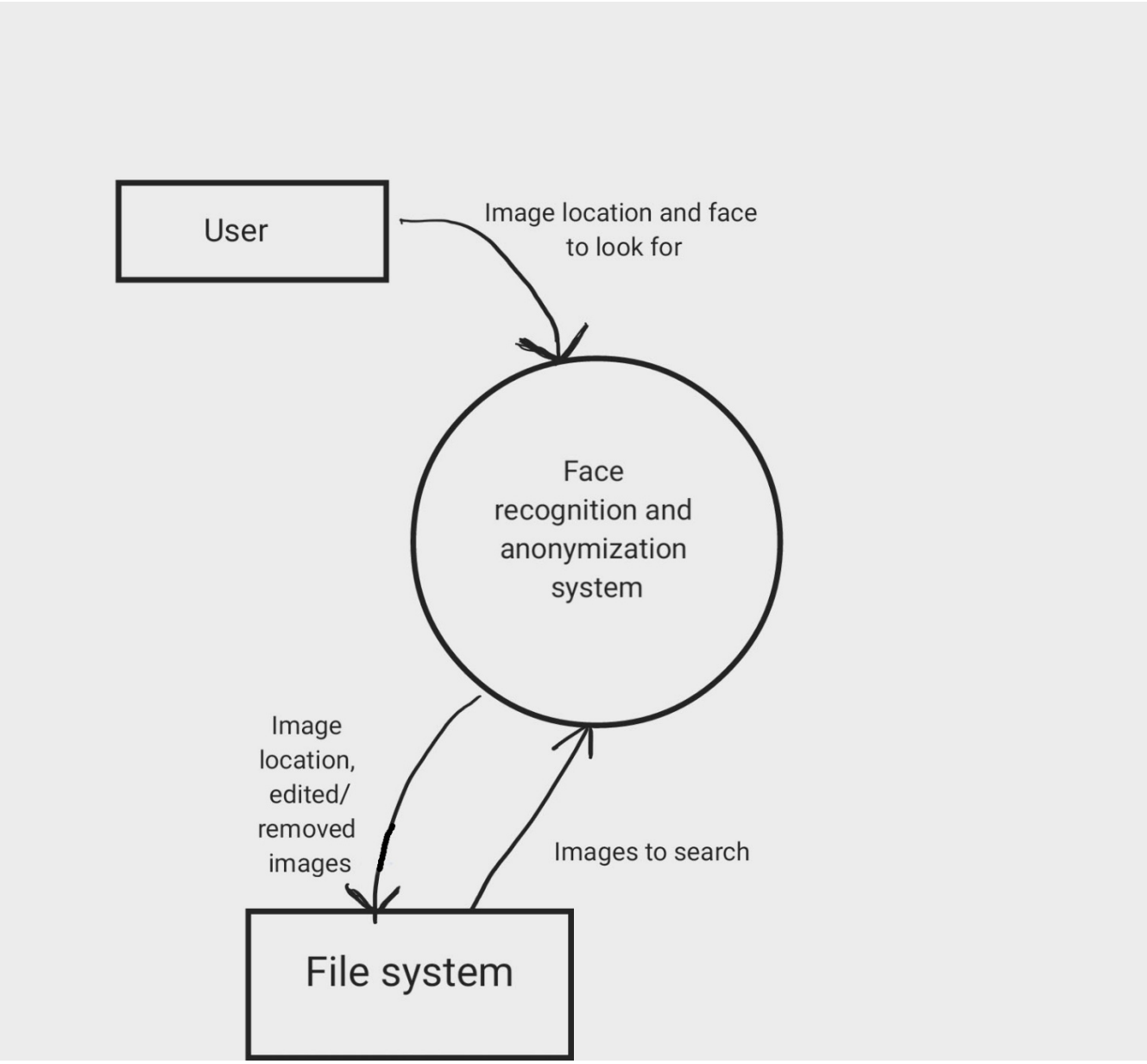
Social and Ethical issues:

Face recognition is the topic of significant debate about its potential for misuse. As with most projects containing face recognition my project has the potential for abuse however I have taken steps to minimize this through only identifying the face of the individual searched for, meaning this can't be used for surveillance, and using open-source software to ensure non authorized people aren’t using my project for nefarious purposes. Additionally, there have been issues with face recognition software not recognizing people who are of different demographics to the developers, I will run extensive testing to reduce these issues as much as possible in my software. This program could accidentally remove the wrong person from pictures which could cause harm, this could be mitigated by adding a manual confirmation system to the program. I am also only using open-source software in accordance with its licensing.

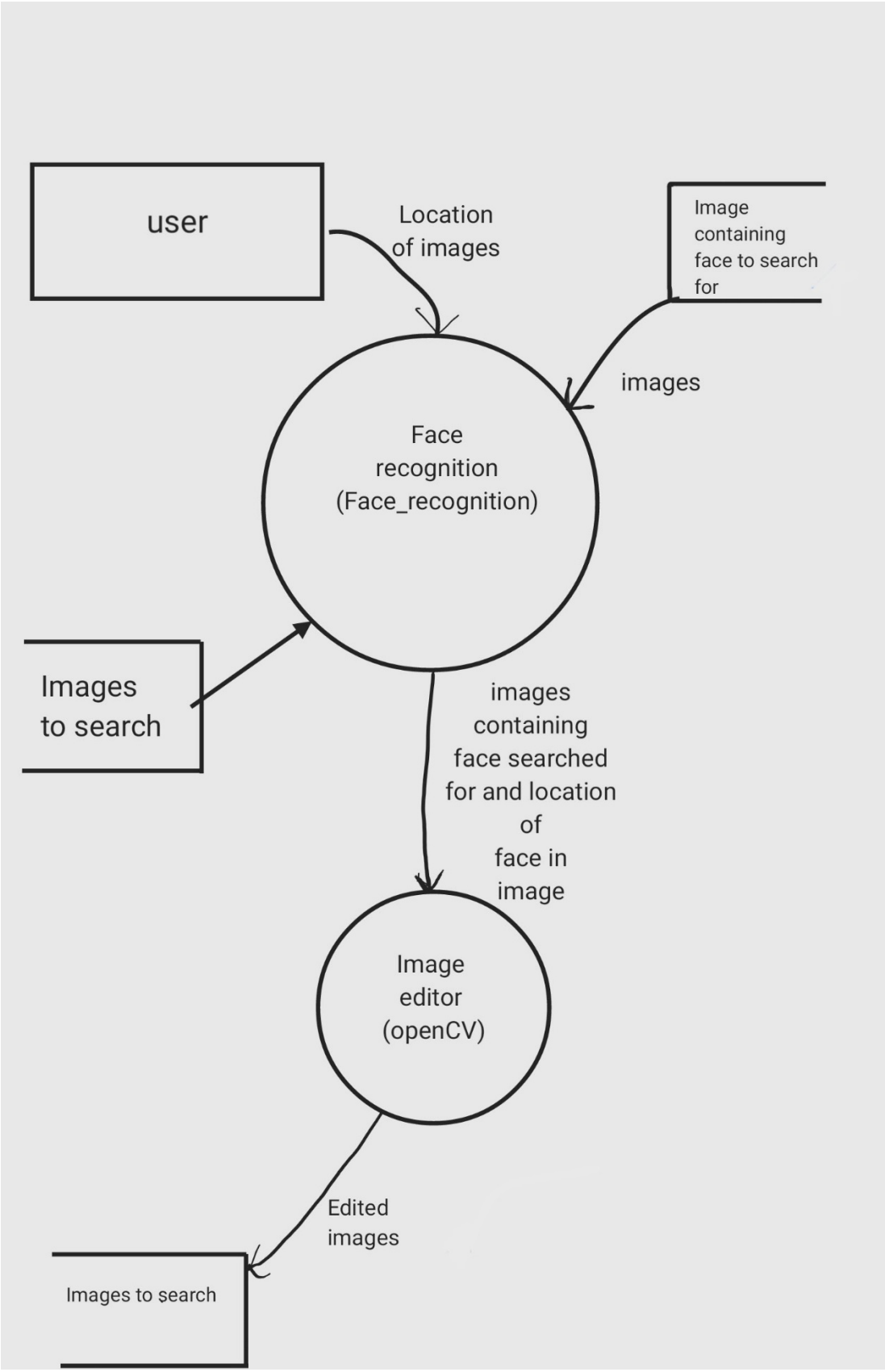
IPO diagram

Input	Processing	Output
Image of person to find	Run face recognition on image	
Blur or delete images containing face searched for		
Location of images to search	If image contains the person the program is looking for edit the image according to the previous input	Edited images

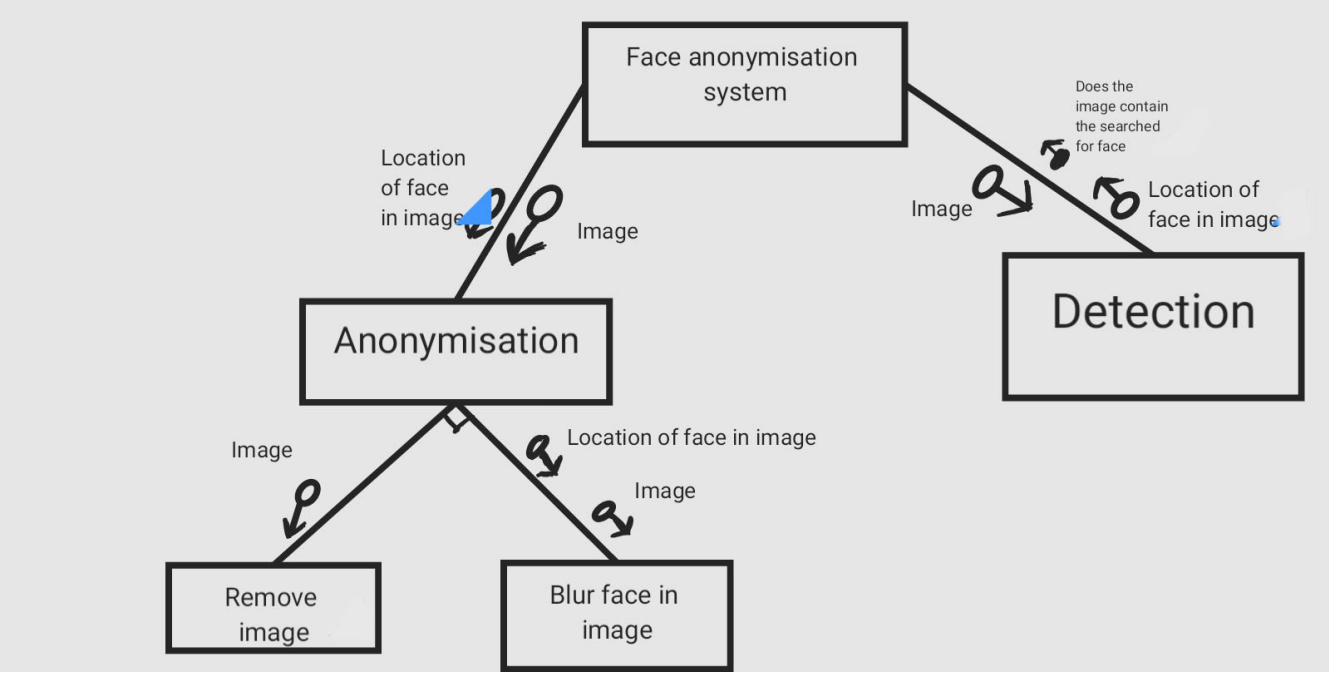
Context diagram



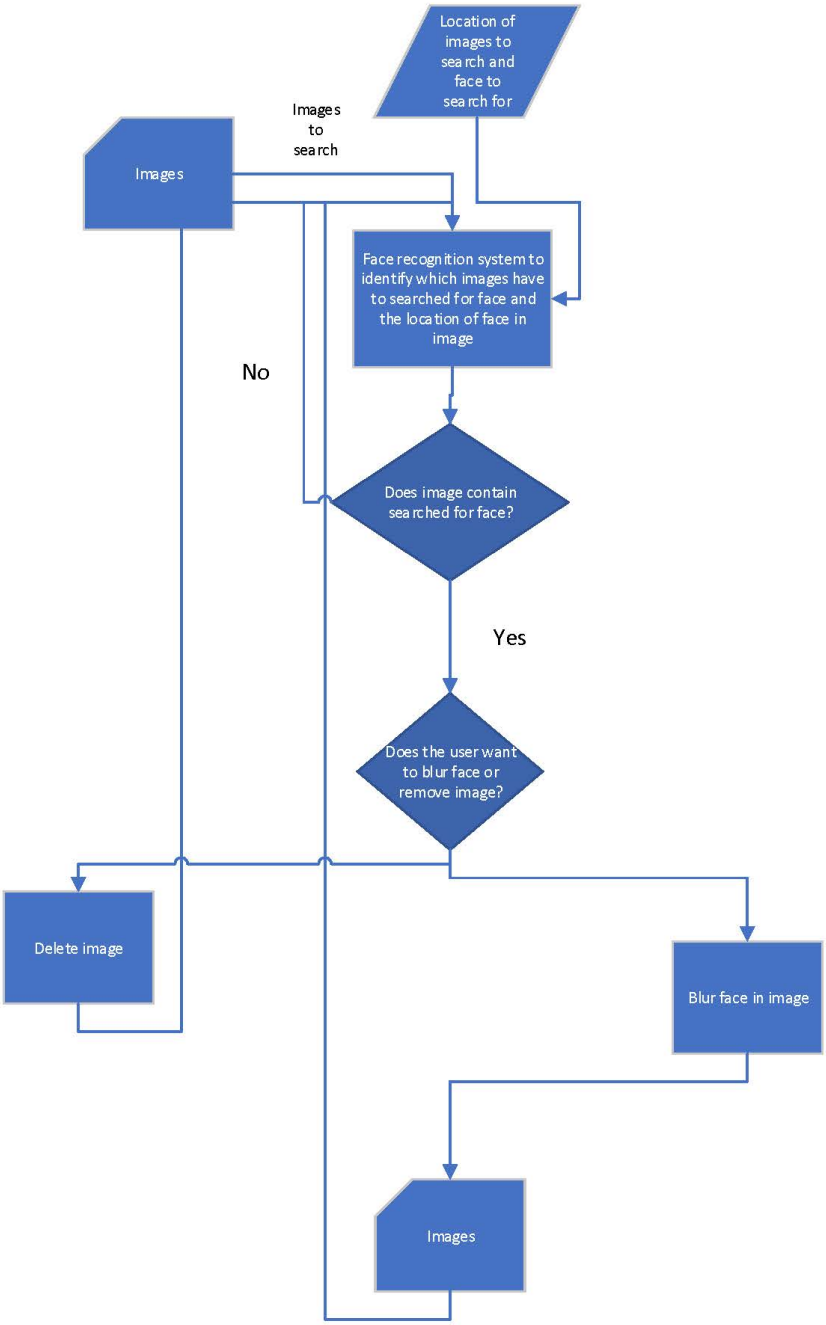
Data flow diagram



Structure chart



System flow chart



Data Dictionary

item	Data type	Description	validation
face_to_search_for	string	String that stores the location of the image containing the face to search for	File exists, valid file directory format
images_to_search	string	String that stores the directory of images to search	Directory exists, valid file directory format
face_to_search_for	Image(eg .png)	Image containing face to search for	Checks the face contains an image, checks it’s a valid image file
Image_to_search	Image(eg .png)	Image to search	Check it’s a valid image file
edited_image	Image(eg .png)	Edited version of “image_to_search” to remove face searched for	Check it’s a valid image file
blur	boolean	True/false value to specify if the program should delete the image or blur the face	NA

Planning and designing

Algorithm

Pseudocode

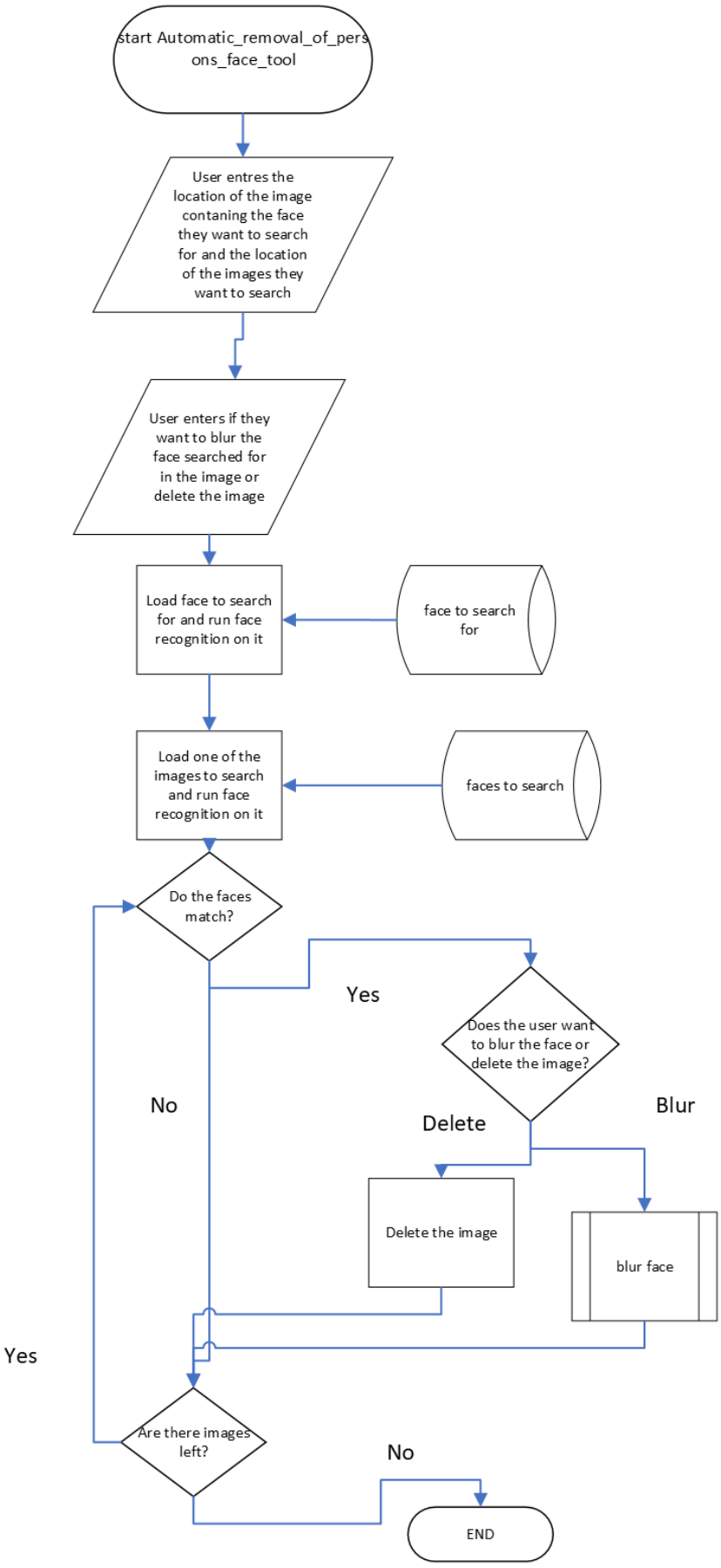
`Frontend:`

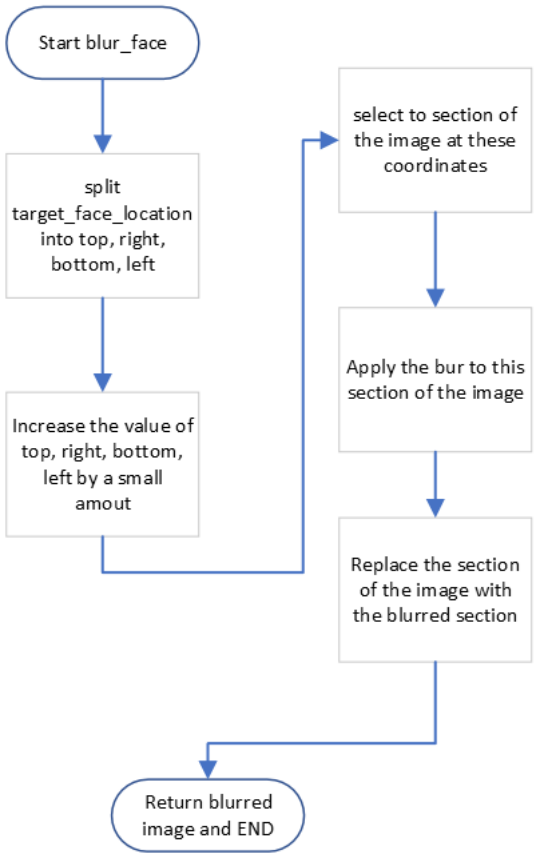
```
BEGIN Automatic_removal_of_face_frontend

  DISPLAY("Select the image containing the face to search for: ")
  face_to_search_for_location <- INPUT()
  DISPLAY("Select the directory containing the images to search: ")
  images_to_search_location <- INPUT()
  DISPLAY(“What sould the program do to matching faces: “)
  action <- INPUT.RADIOBUTTON{
        "Blur matching faces"
        "Replace matching faces"
        "Delete images containg matching faces"
  }
  DISPLAY(“Finished”)
```

END Automatic_removal_of_face_frontend

Algorithm flow chart


start Automatic_removal_of_persons_face_tool
User entres the location of the image contaning the face they want to search for and the location of the images they want to search
User enters if they want to blur the face searched for in the image or delete the image
Load face to search for and run face recognition on it
face to search for
Load one of the images to search and run face recognition on it
faces to search
Do the faces match?
Yes
No
Does the user want to blur the face or delete the image?
Delete
Blur
Delete the image
blur face
Yes
Are there images left?
No
END



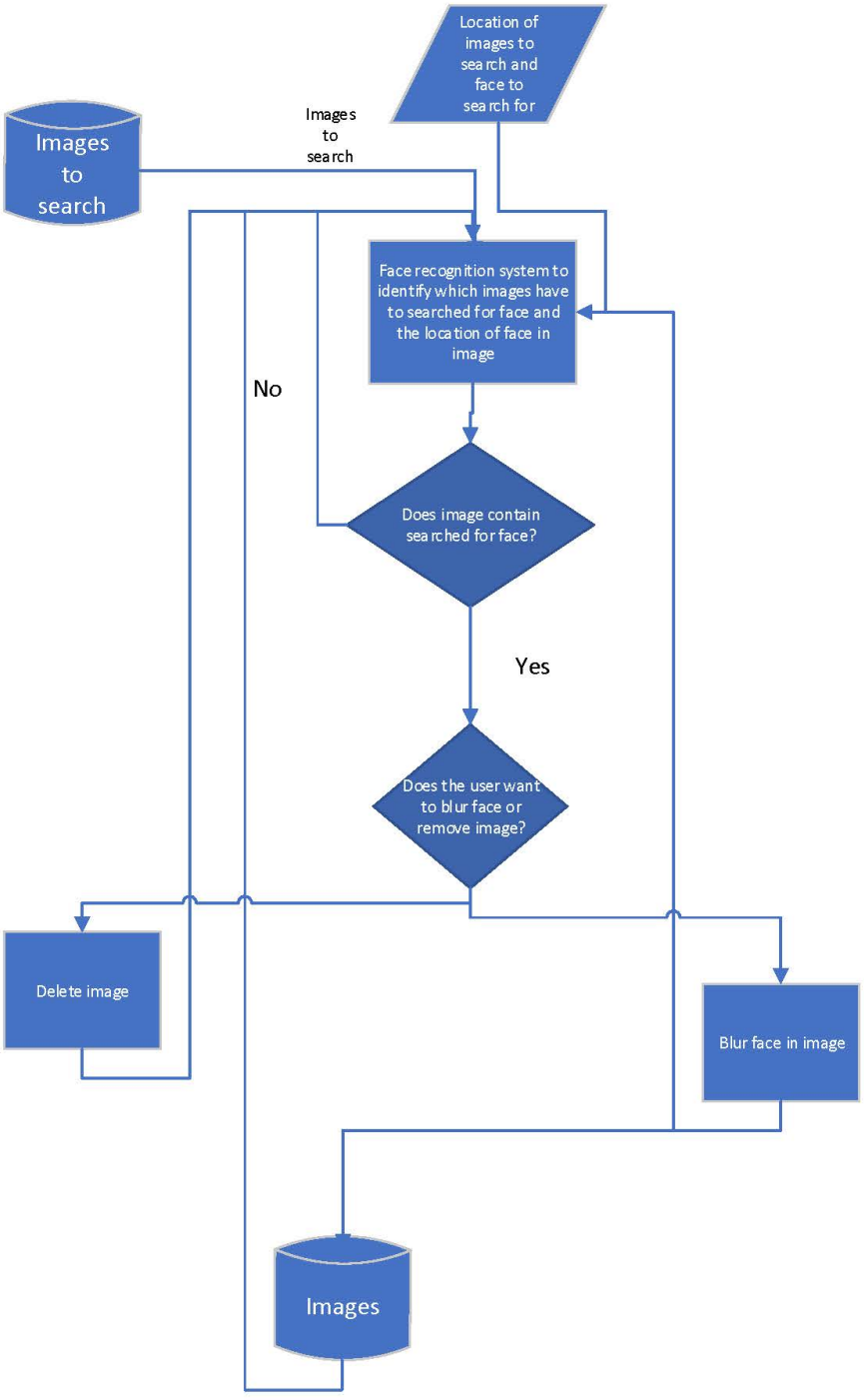
Refined system modeling tools

Data Dictionary

item	Data type	Description	validation
face_to_search_for_location	string	String that stores the location of the image containing the face to search for	File exist, valid file directory format
images_to_search_location	string	String that stores the directory of images to search	Directory exist, valid file directory format
n	int	Int to keep track of what image is being used	NA
face_to_search_for	numpy array	Face to search for loaded into face_recognition	NA
image	numpy array	Image to be search loaded into face_recognition	NA
face_to_search_for_encoding	numpy array	Variable to store the processed face to search for	NA

image_encoding	numpy array	Variable to store the processed image to search	NA
blur	Boolean	True/false value to specify if the program should delete the image or blur the face	Checks it is a valid Boolean

System Flow chart



Software environment

As this program will be doing advanced AI processing and image manipulation this makes many lower-level languages such as C or COBOL unsuited for this purpose. Out of high-level languages the best suited in my opinion is Python as it has support for packages and is the most popular for machine learning, allowing me to utilize per-existing packages to reduce the need for me to write extremely complex code. Python also has other benefits such as being an interpretation language, so I don't need to compile code when I am writing and debugging my software solution and its cross-platform support. The next best language for this project is C# because like python it is a high-level language that has machine learning capabilities; however, I didn't choose it because it doesn't have as good cross platform support which would make development much harder for me as a will be writing this software on a Linux machine and it would likely be run on Windows computers. Additionally, because it was created by Microsoft, I expect I'll have issues getting it to work on Linux.

Hardware requirements

This software will run on most x86_64 computers from the last 15 years. However, it may not run quickly. I would recommend using at least a quad core 3rd gen intel core series or newer to get decent performance. A GPU is not required, however running the face recognition on a GPU will make it quicker. For GPU acceleration, a Nvidia GPU with CUDA is required.

Data files

This program will need the .py file to run or an equivalent file containing the software's code. The only other files needed are the image file the user input into the program. Additionally, as my solution uses existing libraries the files containing these libraries are also required.

Data structure

My solution will access the images from the file system. It will load these images into numpy arrays to allow the program to process them. Then once the processing is done, faces detected and blurred, the arrays are converted back to images and written to the file system in place of the original images. There are also some other variables that the program uses to store options, for example a Boolean to store if the program should blur the face searched for or remove the image and a string containing the location of the images.

Identifying Relevant Standard or Common Modules or Subroutines

This program is already reasonable modular due to the use of per-existing libraries, however the code to blur or remove the image could be split off to allow easier addition of different actions the program could take such as placing a new image over the identified face.

Social and ethical issues & Communication with others

In order to keep this document as ordered and logical as possible I have integrated these sections with the previous section on social and ethical issues and communication with others respectively.

User feedback

Depending on the final form of the software, it may be a web app or run locally, different systems will have to be in place to interoperate user feedback and updates. For both there will be links to a website and an email address likely at the bottom of the screen for the user to provide feedback via. If it is locally run there will have to be an additional module to check for and download updates if available to implement any changes suggested.

Documentation of design

I will be using Libre Office writer to maintain a logbook of my work and Libre Office cal (excel alternative) to manage a Gannt chart to plan my time. I will additionally be using GitHub for management of files and version control. My GitHub repository is at https://github.com/NoahHallows/software_mp This is in addition to updating parts of this document such as the data dictionary as required.

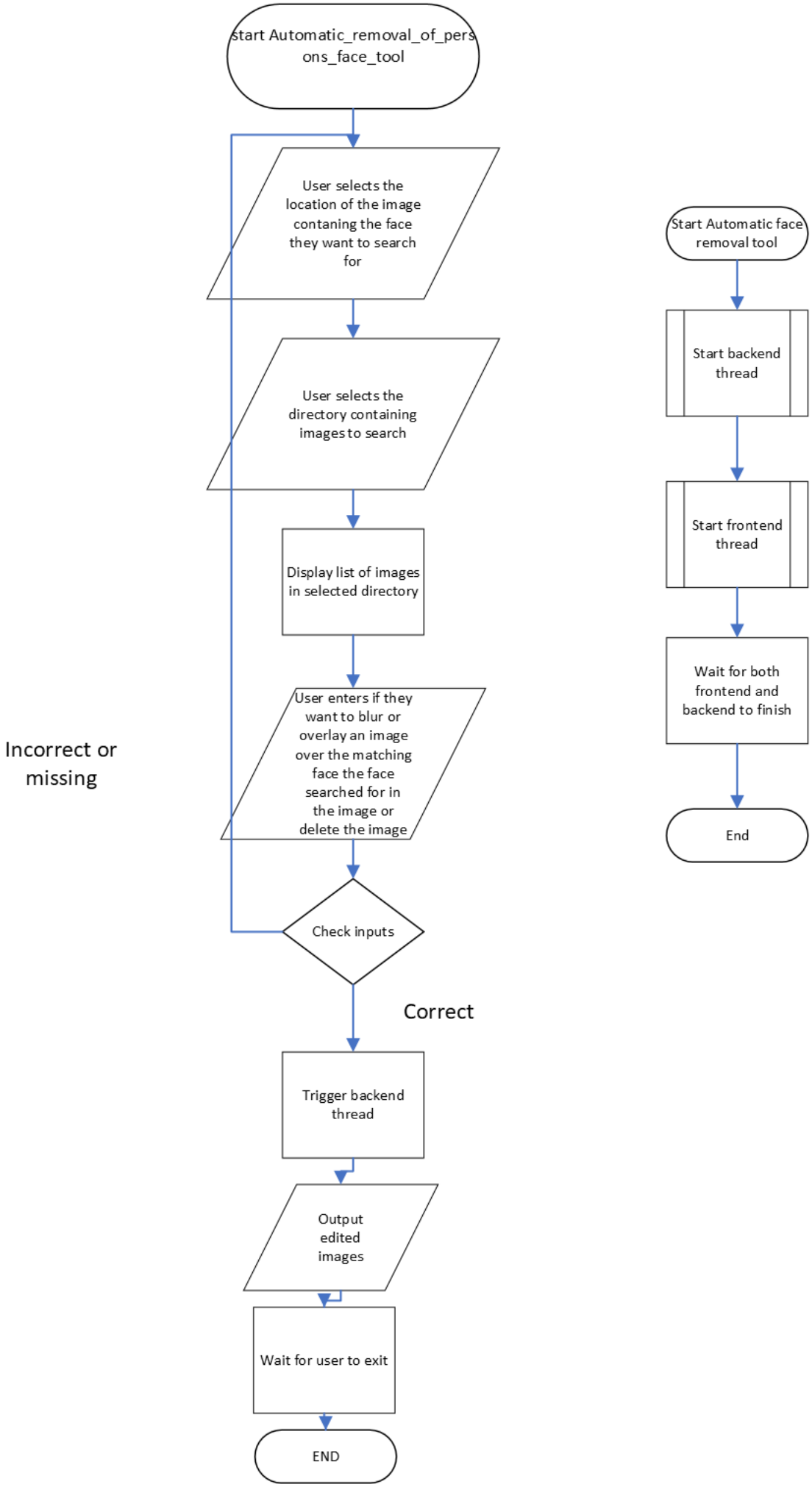
Implementing:

Issues with initial design:

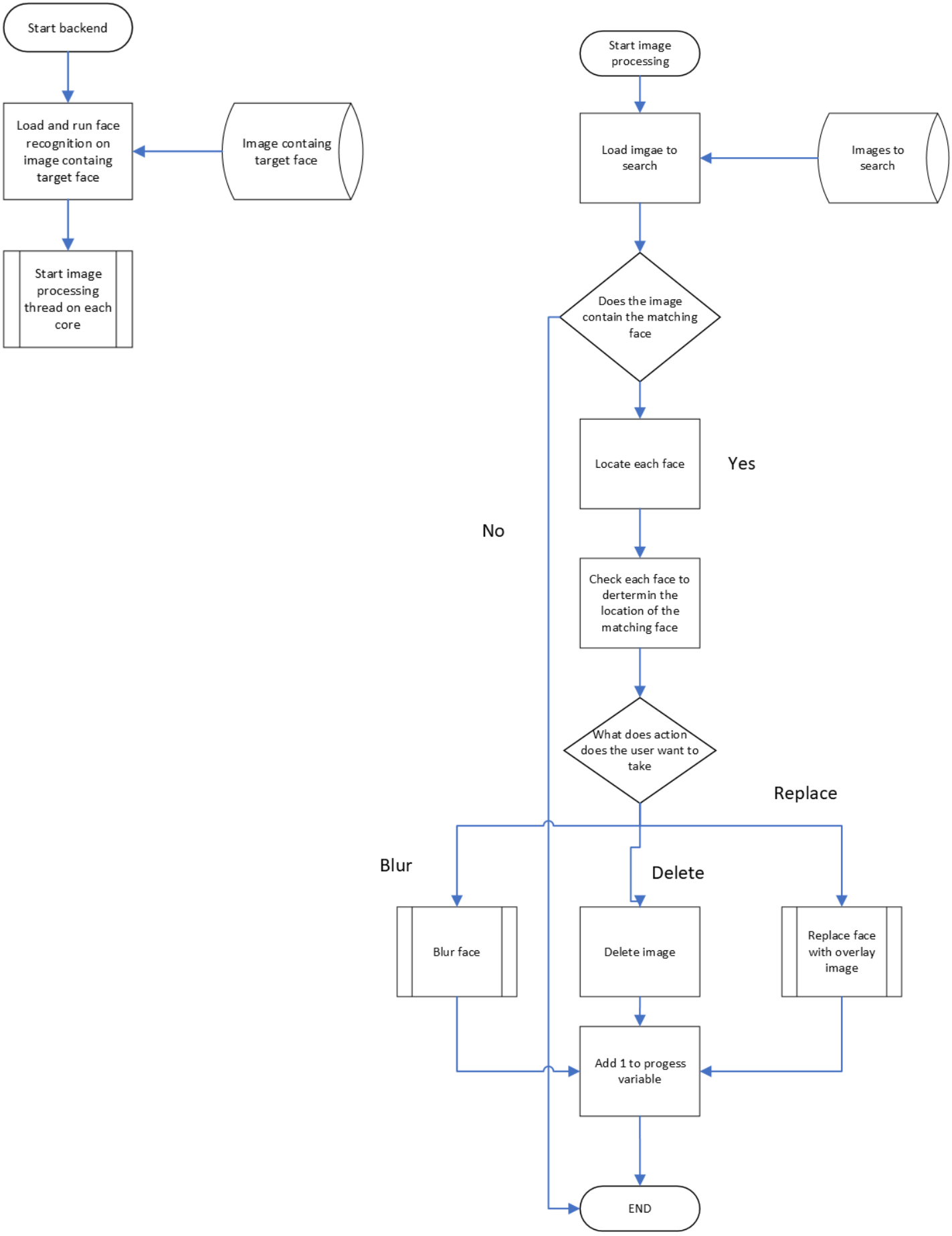
When attempting to implement my original design I encountered several issues with the first being the slow speed of the solution. By far the slowest part of my code was the actual face recognition which was run on the image as a whole and then the individual faces to identify the location of the matching face. This was a difficult issue to address as I didn't possess the time and knowledge to rewrite the face recognition library I was using, leaving me with two options. I could switch to a different, hopefully faster, library however, this had no guarantee to actually improve performance and would require rewriting a significant portion of my code, or I could use multiple cpu cores, running the processing in parallel. I ultimately decided to implement multithreading which has to bonus of fixing my next major issue, however I may switch to a different more modern face recognition library in the future. My next major issue is implementing a progress bar for my solution. On the surface this seems to be a simple task, have a variable tracking the number of images processed and use that to work out a percentage which you display to the user. However this simple approach requires the program to run asynchronously with a frontend and backend thread. Unfortunately the framework I originally intended to use, PySimpleGUI, was giving an error related to X11 when I tried to implement this feature so I decided to switch to PySide6 instead. One advantage of this multithreaded approach is the user is still able to interact with the ui while the program is running the image processing.

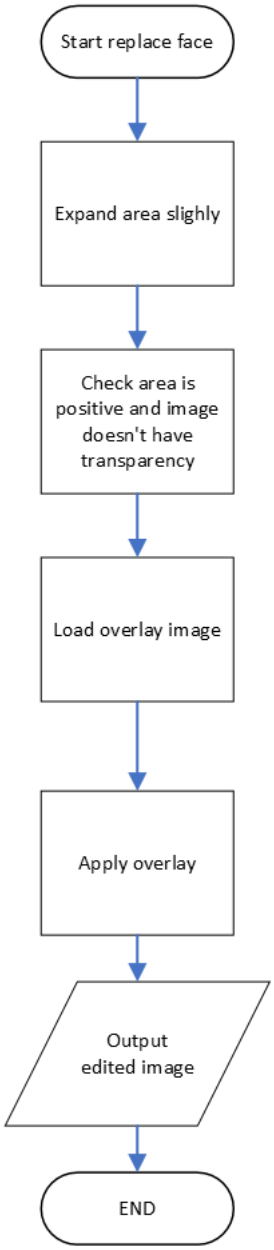
Changes to design and updated documentation:

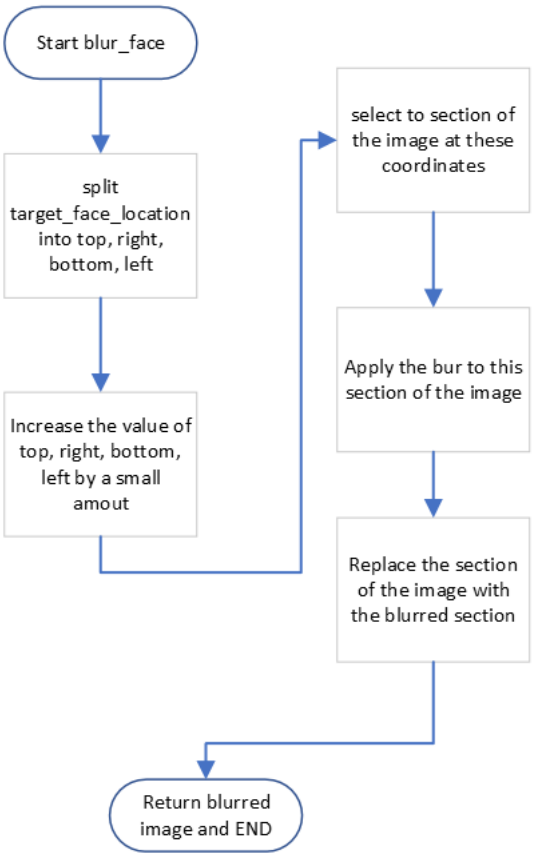
I had to split the program into several sub-programs to implement the multithreading. This had the bonus of making the program more modular.


start Automatic_removal_of_persons_face_tool
User selects the location of the image containg the face they want to search for
User selects the directory containing images to search
Display list of images in selected directory
User enters if they want to blur or overlay an image over the matching face the face searched for in the image or delete the image
Check inputs
Incorrect or missing
Correct
Trigger backend thread
Output edited images
Wait for user to exit
END
Start Automatic face removal tool
Start backend thread
Start frontend thread
Wait for both frontend and backend to finish
End

Algorithm flow charts: Frontend







Backend:

Data dictionary:

item	Data type	Description	validation
face_to_search_for	string	String that stores the location of the image containing the face to search for	File exists, valid file directory format
images_to_search_locati	string	String that stores the	Directory exists, valid

on		directory of images to search	file directory format
face_to_search_for	Image array(eg .png)	Image containing face to search for	Checks the face contains an image, checks it's a valid image file
Image_name	Image array(eg .png)	Image to currently check	Check it's a valid image file
new_image	Image array(eg .png)	Edited version of "image_to_search" to remove face searched for	Check it's a valid image file
action	int	Integer value to specify if the program should delete the image or blur the face	NA
Target_face_location	string	String that stores the location of the image containing the face to search for	Checks it's a valid directory and file exists
Overlay_image_location	string	String that stores the location of the image containing the overlay image	Checks it's a valid directory and file exists
progress	Multiprocessing Shared memory value	Stores the number of images checked	NA
image	Mostly opencv image array	Stores the image relevant to the current context	Checks it's a valid image
percentage	float	Percentage of images processed	NA
Image_bgr	Opencv image array	Convert image to bgr for opencv	NA
Image_encodings	Face encoding	Encoding for image currently being searched	Image contains a face
Images_to_search	list	List of images to search	Checks list is not empty
results	list	Output of face_recog function, list of images searched and if the image matches ect	NA
data_queue	Multiprocessing thread safe queue	Collection of variables including action, target_face_location,	NA

		images_to_search and overlay image location	
match	int	Int to store if the image currently being searched matches	NA

IPO diagram:

Input	Processing	Output
Image of person to search	Run face recognition on image	
Should the program blur, replace or delete images with matching face		
Overlay image location (if applicable)		
Location of images to search	If image contains the person the program is looking for edit the image according to the previous input	Edited images

Pseudocode:

```
import sys
from PySide6.QtWidgets import QApplication, QDialog, QDialogButtonBox, QLineEdit, QVBoxLayout,
QFileDialog, QProgressBar, QRadioButton, QPushButton, QLabel, QGridLayout, QListWidget, QMessageBox
from PySide6.QtCore import Slot
from PySide6.QtGui import QPixmap
import re
import cv2 as cv
import os
from time import sleep
from multiprocessing import cpu_count, Queue, Pool, Value, Process, Event, freeze_support
import pathlib
import face_recognition
import concurrent.futures
progress <- Value('i', 0)
images_to_search <- []
event <- Event()
data_queue <- Queue()

CLASS Window(QDialog):
    # Logic for getting radio button value
    FUNCTION radio(self):
        button <-  sender()
         action <- button.option
    # For end screen
    ENDFUNCTION
```

```
    FUNCTION end_screen(self, results):
        end_grid <- QGridLayout()
        FOR image_name IN results:
            image <- cv.imread(image_name)
            # Get the dimensions of the image (height, width, number_of_channels)
            height, width, channels <- image.shape
            scale_factor <- 200/height
            new_width <- int(round(scale_factor*width, 0))
            resized_image <- cv.resize(image, (new_width, 200), interpolation=cv.INTER_AREA)
            imgbytes <- cv.imencode(".png", resized_image)[1].tobytes()
            image_display <-  QLabel(self)
            pixmap <- QPixmap(resized_image)
            image_display.setPixmap(pixmap)
            end_grid.addWidget(image_display, 0,0)
        ENDFOR
    # Logic for selecting target image
    ENDFUNCTION

    @Slot()
    FUNCTION select_target_face(self):
        target_face_location_untrimmed <- QFileDialog.getOpenFileName(self, ("Open image"), "", ("Image
(*.png *.jpg *.bmp)"))
        IF target_face_location_untrimmed:
            target_image_text_box.clear()
            # Use regular expression to find the file path
            # THIS STRING PROCESSING IS A STARDARD ALGORITHM
            target_face_location <- Extract from 2nd character until character == """ from
target_face_location_untrimmed into target_face_location
            target_image_text_box.setText( target_face_location)
        ENDIF
    # Logic for progress bar
    ENDFUNCTION

    FUNCTION progress_bar_update(self, images_to_search):
        percentage <- 0
        WHILE percentage <= 100:
            percentage <- progress.value * (100/len(images_to_search))
            progress_bar.setValue(percentage)
            sleep(0.5)
        ENDWHILE
    # For standard buttons
    ENDFUNCTION

    @Slot()
    FUNCTION accept(self):
        # Input checking
        IF  images_to_search_location != '' AND  action != 0 AND  target_face_location != '':
            # Setup worker to update progress bar
            progress_worker <- concurrent.futures.ThreadPoolExecutor(max_workers=1)
            progress_worker.submit( progress_bar_update,  images_to_search)
            images_to_search <- get_files( images_to_search_location)
            # Use of queues to transfer data from frontend to backend
```

```
        data_queue.put([ action,  target_face_location,  images_to_search,  overlay_image_location])
        # Tell the backend to run
        event.set()
      ELSE:
        msgBox <- QMessageBox()
        msgBox.exec()
      ENDIF
    ENDFUNCTION

@Slot()
FUNCTION reject(self):
    QUIT

    FUNCTION __init__(self):
        # Set overlay_image_location to none in case it's not applicable
        overlay_image_location <- None
        super().__init__(parent=None)
        setWindowTitle("Face removal tool v0.8")
        dialogLayout <- QVBoxLayout()
        #formLayout <- QFormLayout()
        gridLayout <- QGridLayout()
        # Selecting target face image
        target_image_button <- QPushButton("Browse")
        target_image_text_box <- QLineEdit()
        target_image_text_box.setReadOnly(True)
        gridLayout.addWidget(QLabel("Select the image containing the face to search for:"), 0, 0)
        gridLayout.addWidget( target_image_text_box, 0, 1)
        gridLayout.addWidget(target_image_button, 0, 2)
        target_image_button.clicked.connect(select_target_face)
        #Selecting image directory
        select_image_directory_button <- QPushButton("Browse")
        select_image_directory_text_box <- QLineEdit()
        select_image_directory_text_box.setReadOnly(True)
        gridLayout.addWidget(QLabel("Select the directory contining images to search:"), 1, 0)
        gridLayout.addWidget(select_image_directory_text_box, 1, 1)
        gridLayout.addWidget(select_image_directory_button, 1, 2)
        select_image_directory_button.clicked.connect(select_image_directory)
        # List images
        image_list_box <- QListWidget()
        gridLayout.addWidget(image_list_box, 2, 0, 1, 2)
        # Select action option
        blur_button <- QRadioButton("Blur matching faces", self)
        blur_button.option <- 1
        replace_button <- QRadioButton("Replace matching faces", self)
        replace_button.option <- 2
        delete_button <- QRadioButton("Delete images containg matching faces", self)
        delete_button.option <- 3
        # Add radio buttons to layout
        gridLayout.addWidget(QLabel("What should the program do to matching faces:"), 3, 0, 1, 2)
        gridLayout.addWidget(blur_button, 4, 0)
        gridLayout.addWidget(replace_button, 5, 0)
        gridLayout.addWidget(delete_button, 6, 0)
        # Connect radio buttons to fuction
```

```
        blur_button.clicked.connect(radio)
        replace_button.clicked.connect(radio)
        delete_button.clicked.connect(radio)
        # Select overlay image IF applicable
        select_overlay_button <- QPushButton("Browse")
        select_overlay_text_box <- QLineEdit()
        select_overlay_text_box.setReadOnly(True)
        gridLayout.addWidget(QLabel("Select the overlay image:"), 7, 0)
        gridLayout.addWidget(select_overlay_text_box, 7, 1)
        gridLayout.addWidget(select_overlay_button, 7, 2)
        select_overlay_button.clicked.connect(select_overlay_image)
        # Add formLayout to dialogLayout
        dialogLayout.addLayout(gridLayout)
        # Progress bar
        progress_bar <- QProgressBar()
        dialogLayout.addWidget( progress_bar)
        # Add standard buttons
        standard_buttons <- QDialogButtonBox(QDialogButtonBox.Ok
                        = QDialogButtonBox.Cancel)
        dialogLayout.addWidget(standard_buttons)
        # Connect the accepted AND rejected signals to respective methods
        standard_buttons.accepted.connect(accept)
        standard_buttons.rejected.connect(reject)
        # Set dialogLayout to be the layout of the window
        SetLayout(dialogLayout)
    # Logic for selecting overlay image
    ENDFUNCTION

    @Slot()
    FUNCTION select_overlay_image(self):
        overlay_image_location_untrimmed <- QFileDialog.getOpenFileName(self, ("Open image"), "", ("Image
(*.png *.jpg *.bmp)"))
        IF overlay_image_location_untrimmed:
            select_overlay_text_box.clear()
            # Use regular expression to find the file path
            Extract from 2nd character until character == """ from overlay_image_location_untrimmed into
overlay_image_location
            select_overlay_text_box.setText(overlay_image_location)
        ENDIF
    # Logic for selecting image directory
    ENDFUNCTION

    @Slot()
    FUNCTION select_image_directory(self):
        images_to_search_location <- QFileDialog.getExistingDirectory(self, ("Open folder"))
        IF  images_to_search_location:
            select_image_directory_text_box.clear()
            select_image_directory_text_box.setText(str( images_to_search_location))
            # Display images in directory
            images_to_search <- get_files( images_to_search_location)
            FOR image IN images_to_search:
                image_list_box.addItem(image)
            ENDFOR
```

```
    ENDIF
  ENDFUNCTION

ENDCLASS

CLASS backend:
  FUNCTION __init__(self):
    # Create backend windows for popups
    app <- QApplication([])
    window <- Window()
    # Wait for trigger
    event.wait()
    # Get input data
    input_data <- data_queue.get()
    action <- input_data[0]
    target_face_location <- input_data[1]
    images_to_search <- input_data[2]
    overlay_image_location <- input_data[3]
    # Call function to spawn processing threads
    start(images_to_search,  action)
  ENDFUNCTION

  FUNCTION start(self, images_to_search, action):
    progress.value <- 0
    # Process the image contaning the face to search fOR
    TRY:
      face_to_search_for <- face_recognition.load_image_file( target_face_location)
       face_to_search_for_encoding <- face_recognition.face_encodings(face_to_search_for)[0]
    EXCEPT Exception as e:
      OUTPUT f"Error {e}"
      msgBox <- QMessageBox()
      msgBox.setText(f"No face found in image containing target face\n{e}")
      msgBox.exec()
    ENDTRY
    IF action = 2:
      #if selected access the overlay image
       overlay <- cv.imread( overlay_image_location)
    ENDIF
    WITH Pool(processes=cpu_count()) as pool:
      # Map the image processing function over the images
      results <- pool.map(face_recog, images_to_search)
    results <- [item FOR item in results IF item is not None]
    msgBox <- QMessageBox()
    msgBox.setText("The images have been searched")
    msgBox.exec()
  ENDFUNCTION

  FUNCTION face_recog(self, image_name):
    TRY:
      # Load AND run face recognition on the image to search
      image <- face_recognition.load_image_file(image_name)
      image_encodings <- face_recognition.face_encodings(image)
      IF image_encodings:
```

```
            image_encoding <- image_encodings[0]
            # Compare faces
            results <- face_recognition.compare_faces([ face_to_search_for_encoding], image_encoding)
            # Convert image to BGR for OpenCV
            image_bgr <- cv.cvtColor(image, cv.COLOR_RGB2BGR)
            IF results[0]:
               # Get location of faces in image
               face_locations <- face_recognition.face_locations(image)
               FOR face_location in face_locations:
                  # See IF the face is a match for the known face
                  face_encoding <- face_recognition.face_encodings(image, [face_location])[0]
                  match <- face_recognition.compare_faces([ face_to_search_for_encoding], face_encoding)
                  # If it's a match, blur the face
                  IF match[0]:
                     IF action = 1:
                        # Call editing image CLASS blur function
                        new_image <- editing_image.blur(image_bgr, face_location, 1)
                     ELSEIF action = 2:
                        new_image <- editing_image.replace(image_bgr,  overlay, face_location, 1)
                     ELSEIF action = 3:
                        os.remove(image_name)
                     ENDIF
                     cv.imwrite(image_name, new_image)
                     progress.value += 1
                  ENDIF
                  RETURN image_name
               ENDFOR
            ELSE:
               # Put progress update to the queue
               progress.value += 1
               RETURN f"Image {image_name} doesn't match"
            ENDIF
         ELSE:
            # Put progress update to the queue
            progress.value += 1
            RETURN f"No faces found in image {image_name}"
         ENDIF
      EXCEPT Exception as e:
         progress.value += 1
         RETURN f"An error occurred with image {image_name}: {e}"
      ENDTRY
   ENDFUNCTION

ENDCLASS

FUNCTION get_files(images_to_search_location):
   FOR file in pathlib.Path(images_to_search_location).rglob("*.jpg"):
      images_to_search.append(str(file))
   ENDFOR
   FOR file in pathlib.Path(images_to_search_location).rglob("*.JPG"):
      images_to_search.append(str(file))
   ENDFOR
   FOR file in pathlib.Path(images_to_search_location).rglob("*.png"):
```

```
        images_to_search.append(str(file))
    ENDFOR
    RETURN list(images_to_search)
ENDFUNCTION

CLASS editing_image():
    #EDITING IMAGES
    FUNCTION blur(image_bgr, target_face_location, scale_factor):
        # Unpack the location
        top, right, bottom, left <- target_face_location
        # Scale face_location coordinates up to the original image size
        top <- int(top / scale_factor)
        right <- int(right / scale_factor)
        bottom <- int(bottom / scale_factor)
        left <- int(left / scale_factor)
        # Calculate the width AND height of the bounding box
        width <- right - left
        height <- bottom - top
        # Increase the region slightly to make sure the entire face is covered
        top <- max(top - 10, 0)
        right <- min(right + 10, image_bgr.shape[1])
        bottom <- min(bottom + 10, image_bgr.shape[0])
        left <- max(left - 10, 0)
        # Select the region of interest (ROI) where the face is located
        face_roi <- image_bgr[top:bottom, left:right]
        # Apply a Gaussian blur to the face region
        blurred_face <- cv.GaussianBlur(face_roi, (999, 999), 0)
        # Replace the original image region with the blurred face
        image_bgr[top:bottom, left:right] <- blurred_face
        RETURN image_bgr
    ENDFUNCTION

    FUNCTION replace(background, overlay, target_face_location, scale_factor):
        # Unpack the location
        top, right, bottom, left <- target_face_location
        # Scale face_location coordinates up to the original image size
        top <- int(top / scale_factor)
        right <- int(right / scale_factor)
        bottom <- int(bottom / scale_factor)
        left <- int(left / scale_factor)
        # Calculate the width AND height of the bounding box
        width <- right - left
        height <- botton - top
        # Expand the bounding box by the constant value
        top <- max(0, top - 10)
        bottom <- min(background.shape[0], bottom + 10)
        left <- max(0, left - 10)
        right <- min(background.shape[1], right + 10)
        width <- width + 20
        height <- height + 20
        # Ensure the width AND height are positive before resizing
        IF width > 0 AND height > 0:
            overlay_resized <- cv.resize(overlay, (width, height))
```

```
        ELSE:
            # Handle the invalid size case, e.g., by skipping the resizing OR setting a default size
            OUTPUT f"Invalid size for resize operation: width={width}, height={height}"
            RETURN background
             ENDIF
        # Check IF overlay image has an alpha channel (transparency)
        IF overlay_resized.shape[2] = 4:
            # Split overlay into color AND alpha channels
            overlay_color <- overlay_resized[:, :, :3]
            alpha_mask <- overlay_resized[:, :, 3] / 255.0
            # Get the ROI from the background AND blend using the alpha mask
            roi <- background[top:bottom, left:right]
            roi <- cv.addWeighted(overlay_color, alpha_mask, roi, 1 - alpha_mask, 0, roi)
            # Put the blended ROI back into the background
            background[top:bottom, left:right] <- roi
        ELSE:
            # If no alpha channel, just replace the ROI with the resized overlay
            background[top:bottom, left:right] <- overlay_resized
        ENDIF
        RETURN background
    ENDFUNCTION

ENDCLASS

FUNCTION ui_start():
    app <- QApplication([])
    window <- Window()
    window.show()
    sys.exit(app.exec())
ENDFUNCTION

FUNCTION main():
    # For pyinstaller
    freeze_support()
    # creating processes
    frontend_thread <- Process(target=ui_start)
    backend_thread <- Process(target=backend)
    frontend_thread.start()
    backend_thread.start()
    frontend_thread.join()
    backend_thread.join()
ENDFUNCTION

IF __name__ = "__main__":
    main()
ENDIF
```

GUI:

I ended up making the gui using a python version of the popular gui framework QT6 called PySide6 with various version of QT being used in several major projects such as KDE plasma. I had originally planed to use a different library called PySimpleGUI however as described above there were issues with this framework. The screenshot below is the current state of the UI. There will be an end screen

that will display information about the images that have been edited.

Sources:

https://doc.qt.io/qtforpython-6/api.html
https://www.geeksforgeeks.org/multiprocessing-python-set-1/
https://www.geeksforgeeks.org/multiprocessing-python-set-2/
https://docs.python.org/3/library/multiprocessing.html
https://github.com/ageitgey/face_recognition
https://docs.python.org/3/library/threading.html
https://docs.opencv.org/4.x/d1/dfb/intro.html
https://realpython.com/face-recognition-with-python/
https://realpython.com/pysimplegui-python/
https://virtualenv.pypa.io/en/latest/user_guide.html

I also used some other sources for minor things such as getting visual studio code to work with virtual environment but I havn't included them because they were only indirectly related to this project and are no longer necessary

Software year 12 task 2 logbook

The Logbook is in a separate file called “Software y12 task1 log book.pdf”

Gannt chart:

The gannt chart is in a separate file called “Software task 2 gannt chart.ods”

Test data

Test data is in a separate directory called “test data”