# **CVI620 – Assignment 1**

Summer 2025

| Total Mark: | 7.5 marks (7.5% of the total course grade) |
| --- | --- |
| Submission file(s): | * Assignment1.py or Assignment1.ipynb (you can upload multiple files) * Assignment1.docx (this document with your answers) |
| Deadline | * May 28th, 2025 |

If you are unable to complete the assignment on-time for any legit reason, please provide documentation explaining your absence (e.g., an appointment confirmation or a work letter).

Please be aware that the assignment is designed to make you **research** and develop.

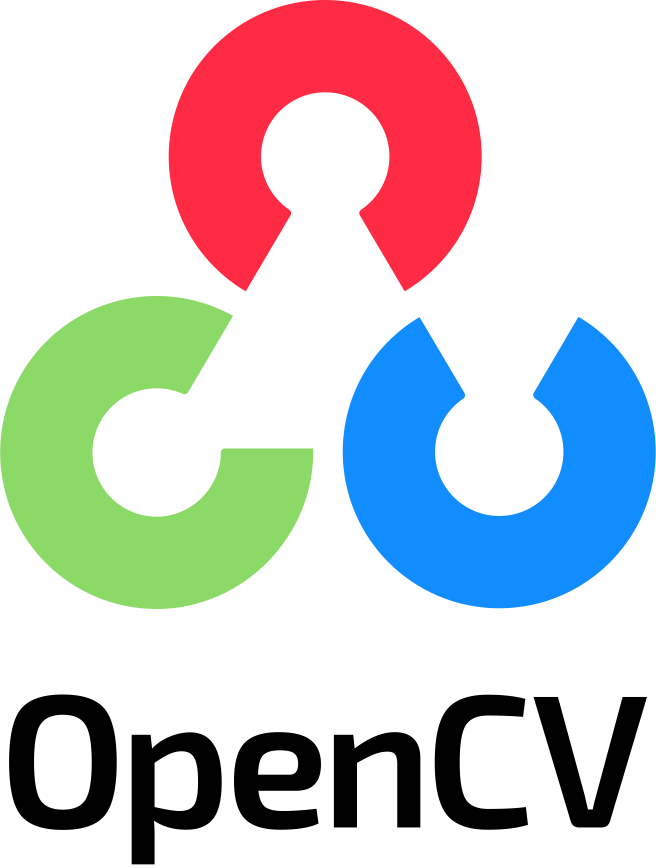
Please submit the submission file(s) through Learn@Seneca. Make sure to use GitHub and provide the link to your GitHub account for all your contributions in the box below:

|  |  |
| --- | --- |
| Project GitHub repository: | <https://github.com/Gaganot-Singh/cvi620/tree/main/ASSIGNMENT%201> |

**Please paste the resulting images and answers in this document.**

## **Part I: A photo booth application**

Recreate the OpenCV logo using OpenCV drawing functions only (e.g., cv2.circle, cv2.line, etc.).



You must draw:

* Three colored shapes (ellipse: Blue, Green, Red) arranged in a triangular pattern.
* Proper positioning and size of circles.
* The text "OpenCV" at the center/bottom.
* img = np.ones((800, 800, 3), dtype=np.uint8) \* 255
* radius = 100
* thickness = -1
* inner\_radius = 40
* center\_red = (400, 250)
* center\_green = (300, 450)
* center\_blue = (500, 450)
* color\_red = (64, 42, 255)
* color\_green = (87, 217, 126)
* color\_blue = (255, 144, 30)
* # Creating main circle of red, blue and green
* cv2.circle(img, center\_red, radius, color\_red, thickness)
* cv2.circle(img, center\_green, radius, color\_green, thickness)
* cv2.circle(img, center\_blue, radius, color\_blue, thickness)
* # creating white mini circles to show them as hollow
* cv2.circle(img, center\_red, inner\_radius, (255, 255, 255), thickness)
* cv2.circle(img, center\_green, inner\_radius, (255, 255, 255), thickness)
* cv2.circle(img, center\_blue, inner\_radius, (255, 255, 255), thickness)
* # creating gap
* cv2.ellipse(img, center\_red, (radius, radius), 0, 60, 120, (255, 255, 255), thickness)
* cv2.ellipse(img, center\_green, (radius, radius), 0, 300, 360, (255, 255, 255), thickness)
* cv2.ellipse(img, center\_blue, (radius, radius), 0, 240, 300, (255, 255, 255), thickness)
* font = cv2.FONT\_HERSHEY\_SIMPLEX
* text = "OpenCV"
* font\_scale = 2
* font\_thickness = 5
* text\_size, \_ = cv2.getTextSize(text, font, font\_scale, font\_thickness)
* text\_x = (img.shape[1] - text\_size[0]) // 2
* text\_y = 700
* cv2.putText(img, text, (text\_x, text\_y), font, font\_scale, (0, 0, 0), font\_thickness)
* cv2.imwrite("opencv\_logo.png", img)
* img\_rgb = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)
* plt.figure(figsize=(6, 6))
* plt.imshow(img\_rgb)
* plt.axis('off')
* plt.show()

A logo with a red green and blue circle

AI-generated content may be incorrect.

## **Part II: Image Arithmetic**

Write a Python script to manually blend two images using NumPy operations.

Use the following formula to blend them:

blend = (1−α)⋅img1 + α⋅img2

* alpha must be a value between 0 and 1.
* Do not use cv2.addWeighted.
* Display and save the blended image as "manual\_blend.jpg"

alpha = 0.5

img1\_path = 'image1.jpg'

img2\_path = 'image2.jpg'

output\_path = 'manual\_blend.jpg'

img1 = cv2.imread(img1\_path)

img2 = cv2.imread(img2\_path)

if img1 is None or img2 is None:

raise FileNotFoundError(f"Could not load '{img1\_path}' or '{img2\_path}'")

if img1.shape[:2] != img2.shape[:2]:

img2 = cv2.resize(img2, (img1.shape[1], img1.shape[0]))

f1 = img1.astype(np.float32)

f2 = img2.astype(np.float32)

blend = (1.0 - alpha) \* f1 + alpha \* f2

blend = np.clip(blend, 0, 255).astype(np.uint8)

cv2.imwrite(output\_path, blend)

print(f"Blended image saved as '{output\_path}'")

img1\_rgb = cv2.cvtColor(img1, cv2.COLOR\_BGR2RGB)

img2\_rgb = cv2.cvtColor(img2, cv2.COLOR\_BGR2RGB)

blend\_rgb = cv2.cvtColor(blend, cv2.COLOR\_BGR2RGB)

plt.figure(figsize=(12,4))

plt.subplot(1,3,1)

plt.imshow(img1\_rgb)

plt.title('Image 1')

plt.axis('off')

plt.subplot(1,3,2)

plt.imshow(img2\_rgb)

plt.title('Image 2')

plt.axis('off')

plt.subplot(1,3,3)

plt.imshow(blend\_rgb)

plt.title(f'Blend (Alpha={alpha})')

plt.axis('off')

plt.tight\_layout()

plt.show()

A close up of a car

AI-generated content may be incorrect.

## **Part II: Image Arithmetic**

Design a modular photo editing application in Python using OpenCV and NumPy. The app should allow users to load an image and apply a sequence of processing steps interactively through a menu interface.

Required Functionalities:

* Load an image from file.
* Show the following menu:

==== Mini Photo Editor ====

1. Adjust Brightness

2. Adjust Contrast

3. Convert to Grayscale

4. Add Padding (choose border type)

5. Apply Thresholding (binary or inverse)

6. Blend with Another Image (manual alpha)

7. Undo Last Operation

8. View History of Operations

9. Save and Exit

* Brightness/Contrast
* Padding: Ask the user to specify the padding size and border type (constant, reflect, replicate, etc.). Additionally, include an option for the user to choose the padding proportion: Square, Rectangle, Custom Ratio (e.g., 4:5)

A cat lying on its back on a carpet

AI-generated content may be incorrect.

If the user selects a custom ratio like 4:5, your program should calculate and apply the padding so that the final image respects the chosen aspect ratio, regardless of the original size. The user should also be able to adjust the total padding size, and your code must maintain the proportion accordingly. (add smallest padding at the beginning to make it rectangle, then increase or decrease the padding size of the user wants)

* Thresholding: Let user choose between cv2.THRESH\_BINARY and cv2.THRESH\_BINARY\_INV.
* Blending: Ask for a second image path and alpha (0 to 1).
* Undo Feature: Keep a history stack of image states. Allow the user to revert to the previous state.
* History Log: Keep a list of all actions performed (e.g., "brightness +50", "padded 20px with reflect"). Display the list before exiting or when requested.

Additional Requirements:

* Code must be modular. Each operation must be a separate function.
* Allow the user to apply multiple transformations in a row.
* When exiting, ask whether the user wants to save the final image, and under what filename.
* Use matplotlib to show side-by-side [original | preview] after every transformation.
* Include some of the results in this document.
* import cv2
* import numpy as np
* import matplotlib.pyplot as plt
* def show\_pair(orig, modified, title="Preview"):
* """
* Display original and modified images side by side using Matplotlib.
* """
* orig\_rgb = cv2.cvtColor(orig, cv2.COLOR\_BGR2RGB)
* mod\_rgb = cv2.cvtColor(modified, cv2.COLOR\_BGR2RGB)
* plt.figure(figsize=(8, 4))
* plt.subplot(1, 2, 1)
* plt.imshow(orig\_rgb)
* plt.title('Original')
* plt.axis('off')
* plt.subplot(1, 2, 2)
* plt.imshow(mod\_rgb)
* plt.title(title)
* plt.axis('off')
* plt.tight\_layout()
* plt.show()
* def adjust\_brightness(img, value):
* """
* Increase or decrease brightness by adding or subtracting a constant using cv2.add or cv2.subtract.
* """
* delta = np.ones(img.shape, dtype='uint8') \* abs(int(value))
* if value >= 0:
* return cv2.add(img, delta)
* else:
* return cv2.subtract(img, delta)
* def adjust\_contrast(img, alpha):
* """
* Adjust contrast by scaling pixel values by alpha using cv2.convertScaleAbs.
* """
* return cv2.convertScaleAbs(img, alpha=float(alpha), beta=0)
* def convert\_grayscale(img):
* """
* Convert to grayscale and back to BGR for consistency.
* """
* gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)
* return cv2.cvtColor(gray, cv2.COLOR\_GRAY2BGR)
* def add\_padding(img):
* """
* Add padding to meet a desired aspect ratio and extra size, then choose border type.
* """
* h, w = img.shape[:2]
* print("Choose aspect ratio mode:")
* print(" 1) Square (1:1)")
* print(" 2) Custom ratio (e.g., 4:5)")
* mode = input("Enter choice [1-2]: ")
* if mode == '1':
* rw, rh = 1, 1
* else:
* ratio = input("Enter ratio as w:h (e.g., 4:5): ")
* rw, rh = map(int, ratio.split(':'))
* target\_w = int(np.ceil(h \* rw / rh))
* target\_h = int(np.ceil(w \* rh / rw))
* new\_w = max(w, target\_w)
* new\_h = max(h, target\_h)
* # initial padding to meet aspect ratio
* pad\_w = new\_w - w
* pad\_h = new\_h - h
* left = pad\_w // 2
* right = pad\_w - left
* top = pad\_h // 2
* bottom = pad\_h - top
* # extra uniform padding
* extra = int(input("Enter extra padding (px) to add on each side: "))
* left += extra; right += extra; top += extra; bottom += extra
* # border type selection
* print("Choose border type:")
* print(" 1) CONSTANT 2) REFLECT 3) REPLICATE")
* bt = input("Choice [1-3]: ")
* types = {'1': cv2.BORDER\_CONSTANT,
* '2': cv2.BORDER\_REFLECT,
* '3': cv2.BORDER\_REPLICATE}
* border = types.get(bt, cv2.BORDER\_CONSTANT)
* if border == cv2.BORDER\_CONSTANT:
* while True:
* bcol = input("Enter BGR color for constant border as B,G,R: ")
* parts = bcol.split(',')
* if len(parts) != 3:
* print("Invalid format. Use B,G,R with three integers.")
* continue
* try:
* val = [int(p) for p in parts]
* break
* except ValueError:
* print("Invalid values. Please enter integers between 0 and 255.")
* else:
* val = 0
* padded = cv2.copyMakeBorder(img, top, bottom, left, right, border, value=val)
* msg = f"padded top={top},bottom={bottom},left={left},right={right} type={bt}"
* return padded, msg
* def apply\_threshold(img):
* """
* Apply binary or inverse binary thresholding.
* """
* gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)
* tval = int(input("Enter threshold value [0-255]: "))
* mode = input("Type [1] binary or [2] inverse: ")
* th\_type = cv2.THRESH\_BINARY if mode == '1' else cv2.THRESH\_BINARY\_INV
* \_, th = cv2.threshold(gray, tval, 255, th\_type)
* out = cv2.cvtColor(th, cv2.COLOR\_GRAY2BGR)
* msg = f"threshold {'binary' if mode=='1' else 'inverse'}@{tval}"
* return out, msg
* def blend\_image(img):
* """
* Blend two images using cv2.addWeighted with formula:
* """
* path = input("Enter second image path: ")
* other = cv2.imread(path)
* if other is None:
* print("Failed to load second image.")
* return img, "blend failed"
* h, w = img.shape[:2]
* other = cv2.resize(other, (w, h))
* alpha = float(input("Enter alpha weight for original image [0.0-1.0]: "))
* beta = 1.0 - alpha
* gamma = 0.0
* blended = cv2.addWeighted(img, alpha, other, beta, gamma)
* msg = f"blend alpha={alpha}, beta={beta}, file={path}"
* return blended, msg
* def main():
* history = []
* ops = []
* path = input("Enter image path: ")
* img\_orig = cv2.imread(path)
* if img\_orig is None:
* print("Failed to load image.")
* return
* current = img\_orig.copy()
* while True:
* print("\n==== Mini Photo Editor ====")
* print("1. Adjust Brightness")
* print("2. Adjust Contrast")
* print("3. Convert to Grayscale")
* print("4. Add Padding")
* print("5. Apply Thresholding")
* print("6. Blend with Another Image")
* print("7. Undo Last Operation")
* print("8. View History of Operations")
* print("9. Save and Exit")
* choice = input("Choose [1-9]: ")
* if choice == '1':
* v = int(input("Brightness change [-100 to 100]: "))
* history.append(current.copy())
* current = adjust\_brightness(current, v)
* ops.append(f"brightness {v}")
* show\_pair(img\_orig, current, f"Brightness {v}")
* elif choice == '2':
* a = float(input("Contrast alpha [0.1-3.0]: "))
* history.append(current.copy())
* current = adjust\_contrast(current, a)
* ops.append(f"contrast {a}")
* show\_pair(img\_orig, current, f"Contrast {a}")
* elif choice == '3':
* history.append(current.copy())
* current = convert\_grayscale(current)
* ops.append("grayscale")
* show\_pair(img\_orig, current, "Grayscale")
* elif choice == '4':
* history.append(current.copy())
* current, msg = add\_padding(current)
* ops.append(msg)
* show\_pair(img\_orig, current, "Padded")
* elif choice == '5':
* history.append(current.copy())
* current, msg = apply\_threshold(current)
* ops.append(msg)
* show\_pair(img\_orig, current, "Threshold")
* elif choice == '6':
* history.append(current.copy())
* current, msg = blend\_image(current)
* ops.append(msg)
* show\_pair(img\_orig, current, "Blended")
* elif choice == '7':
* if history:
* current = history.pop()
* last\_op = ops.pop()
* print(f"Undid: {last\_op}")
* show\_pair(img\_orig, current, "After Undo")
* else:
* print("Nothing to undo.")
* elif choice == '8':
* print("\nHistory of operations:")
* for i, op in enumerate(ops, 1):
* print(f" {i}. {op}")
* elif choice == '9':
* save = input("Save final image? (y/n): ")
* if save.lower() == 'y':
* fname = input("Enter filename to save (e.g. out.jpg): ")
* cv2.imwrite(fname, current)
* print(f"Saved as {fname}")
* print("Exiting.")
* break
* else:
* print("Invalid choice. Try again.")
* if \_\_name\_\_ == '\_\_main\_\_':
* main()

A screenshot of a white car

AI-generated content may be incorrect.

A screenshot of a car

AI-generated content may be incorrect.

A screenshot of a car

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

## **Part IV:**

[Academic integrity Policy](https://www.senecapolytechnic.ca/about/policies/academic-integrity-policy.html)

I, Gaganjot Singh, declare that I have read and understood the Academic Integrity Policy.

GOOD LUCK!