

Teaching Hands-On Digital Image Processing with `morph.py`: Methods and Comprehensive Results

Francisco de Assis Zampirolli
Federal University of ABC
ORCID: [0000-0002-7707-1793](https://orcid.org/0000-0002-7707-1793)
fzampirolli@ufabc.edu.br

João Marcelo Borovina Josko
Federal University of ABC
ORCID: [0000-0002-8654-5866](https://orcid.org/0000-0002-8654-5866)
marcelo.josko@ufabc.edu.br

Fernando Teubl
Federal University of ABC
ORCID: [0000-0002-2668-5568](https://orcid.org/0000-0002-2668-5568)
fernando.teubl@ufabc.edu.br

Celso Setsuo Kurashima
Federal University of ABC
ORCID: [0000-0002-3858-2814](https://orcid.org/0000-0002-3858-2814)
celso.kurashima@ufabc.edu.br

Renato de Avila Lopes
Federal University of ABC
ORCID: [0009-0002-8108-1326](https://orcid.org/0009-0002-8108-1326)
renato.avila@ufabc.edu.br

Abstract

Teaching Digital Image Processing (DIP) presents significant challenges due to mathematical and algorithmic complexities. Although the field has experienced recent growth, there is a lack of comprehensive resources to support effective DIP education. To address this gap, this paper introduces a practical course utilizing a Python library named `morph.py`, which is designed for beginners and accessible on Google Colab. This interactive course features illustrative examples and hands-on exercises to help learners grasp fundamental DIP concepts and operators. It starts with basic concepts (e.g., image representation) and gradually advances to more complex topics, including image transformations and feature extraction. This work demonstrates how to use `morph.py` in Colab for teaching materials to tackle different computer vision problems. It also discusses the integration with `MCTest` and Moodle for assessments with automatic grading, enhancing the reproducibility of the presented method. Additionally, the paper details the use of Safe Exam Browser (SEB) for securing exams. We conducted an exploratory case study in one group ($N = 15$) and gathered their perception through a voluntary survey. Our quantitative analysis provides strong support for the effectiveness of our teaching method based on the `morph.py` library, successfully addressing the difficulties of teaching DIP to beginners.

Keywords: Computing Education; Digital Image Processing; Colab; Automatic Correction; `MCTest`; Moodle.

1 Introduction

Digital Image Processing – DIP – is a computer science and engineering course that analyzes, manipulates, and interprets digital images. This field has experienced substantial growth in recent years, driven by the expanding accessibility of digital images and the rising demand for image analysis across diverse domains (e.g., medical imaging, remote sensing, and autonomous vehicles) (Gonzalez & Woods, 2009).

However, learning DIP can be challenging for beginners as it requires a solid understanding of several theoretical concepts and developing practical skills. Hence, isolating lectures and reading materials may not be the most effective teaching and learning approach for subjects like DIP. Research has shown that interactive and practical methods can better engage students and enhance their learning experience Rowe et al. (2018) and Yahya (2019).

The literature provides a few toolboxes to support DIP learning. Most toolboxes are based on a virtual notebook environment and provide a comprehensive set of functions for image analysis (Rowe et al., 2018; Silva et al., 2003). However, some works are outdated, such as the `mmorph` toolbox used in the book of Dougherty and Lotufo (2003). Additionally, none provide constructive feedback to students (Yahya, 2019) or address morphological operators (Rowe et al., 2018).

Furthermore, numerous studies focused on teaching DIP using commercial libraries (Yahya, 2019). These often come with complex syntax (García & Suárez, 2015), demand unavailable environments (Konstantinides & Rasure, 1994), or rely on outdated programming languages like Java, which offer limited DIP capabilities (Sage & Unser, 2001, 2003). Also, there is a lack of literature addressing the automated assessment of programming exercises, specifically in the context of DIP.

This paper presents an interactive DIP course utilizing a toolbox named `morph.py` developed especially in response to the identified gap. The course adopts a hands-on approach, teaching theoretical concepts through practical examples and exercises. This method helps foster the development of practical skills and problem-solving abilities in students. Additionally, the course includes an automatic correction environment that provides feedback on each student's submitted DIP exercises through the Moodle Learning Management System. These exercises are created using the open-source `MCTest` system (Zampirolli, 2023). The introductory lessons focus on fundamental concepts, such as image representation and sampling, while the advanced lessons include computer vision applications like feature extraction and object detection.

This paper also provides a set of notebooks in Google Colaboratory (Colab) to illustrate the method presented. However, due to copyright restrictions on the images used in the book by Gonzalez and Woods (2009), the complete didactic material used in the DIP course in Colab is not fully available in this article.

To evaluate the effectiveness of our DIP course approach, we conducted a voluntary survey with students ($N = 15$). Our quantitative analysis revealed that the course's interactive format positively contributed to students' DIP concept learning. The results suggest that this practical teaching approach is effective in overcoming the challenges of teaching complex technical topics to beginners.

This work is organized as follows: In Section 2, we provide background information on DIP and review relevant previous studies. Our pedagogical approach is described in Section 3.

The implementation of `morph.py` is presented in Section 4. Section 5 shows `morph.py` as a didactic approach, highlighting the Minkowski sum, distance transform, and geodesic distance in solving maze images. In Section 6, we detail the usage of the `morph.py` library in exams. In Section 7, we present the results and engage in corresponding discussions. Lastly, we conclude this work in Section 8. Details on `MCTest`, Moodle, and the SEB (Safe Exam Browser) for creating and securing an exam are included in Appendices 1, 2, and 3, respectively. With all the content provided, the teacher is expected to apply the method to new content and courses. Appendix 4 details the process of creating a dataset using Roboflow, allowing for a comparison of the geometric object detection methods with those described in Section 6.2.

2 Background

A digital image in two dimensions (2D) can be represented by a function $f(x,y)$ defined over a finite subset of the Cartesian plane (x,y) , where the codomain is the interval $[0,k]$. When $k = 1$, f represents a binary image. For $k = 255$, it corresponds to a grayscale image. If f assumes three values within this interval, it can be defined as a color image in RGB (Red, Green, and Blue) format (Gonzalez & Woods, 2009).

DIP is a field of computer science and engineering that focuses on analyzing, manipulating, and interpreting digital images, as seen in the preview. Python has become a popular programming language for DIP due to its user-friendly nature, expansive community, and access to powerful libraries like OpenCV (opencv.org), Pillow (pillow.readthedocs.io), Skimage (scikit-image.org), and TensorFlow (tensorflow.org). These libraries offer various methods for DIP applications, including those based on Mathematical Morphology (MM). Functional operators like dilation, erosion, opening, and closing are valuable for image enhancement, segmentation, and feature extraction tasks.

`MMach` is a DIP library designed for MM and was developed by Prof. Dr. Junior Barrera (Banon & Barrera, 1994). `MMach` had contributions from several researchers, including students at different academic levels, and different versions were generated as Khoros versions changed (Konstantinides & Rasure, 1994). In the mid-1990s, Prof. Dr. Roberto de Alencar Lotufo joined the team to make `MMach` platform-independent (Lotufo et al., 1997). The most recent version of `MMach`, known as `mmorph`, was developed using AdessoWiki (Machado et al., 2011). It is structured in XML and includes automatic code generation for C, Matlab, and Python languages, and automatic documentation generation in TXT, HTML, and \LaTeX formats. Although `mmorph` has been used in several publications, the latest version is no longer available online. One excellent example of its usage can be found in the book by Dougherty and Lotufo (2003).

This paper proposes using `morph.py`, an open-source Python library that provides methods for DIP. The library is available on github.com/fzampirolli/morph and allows for community contributions. `morph.py` has an easy-to-use interface and offers a wide range of morphological operators on images, which have been effectively applied in various DIP courses, including image segmentation, edge detection, and object recognition. The library includes multiple implementations of the same operator, allowing students to compare them with those available in libraries like OpenCV and Skimage. The goal of `morph.py` is to encapsulate functions from other libraries

(such as OpenCV) while offering a simplified interface that enables users to concentrate better on the core content. Furthermore, it aims to standardize the behavior of operations, which is essential for developing questions with predictable answers - a crucial factor in activities involving automated evaluations.

Related Works

This section provides an overview of scientific papers and currently available toolboxes for teaching DIP topics. Silva et al. (2003) developed a toolbox for teaching DIP courses using Python and Adesso (Machado et al., 2003). This toolbox was used in one of the earliest Python-based DIP courses (dca.fee.unicamp.br/ia636) at UNICAMP, consisting of 83 DIP methods with code and documentation. However, the toolbox has not been updated. A newer version of the ia636 course is now available at github.com/MICLab-Unicamp/ia636, but it only includes 78 Python-based DIP methods without documentation. Another toolbox from Prof. Lotufo is a more comprehensive version of the ia636 course, including five lessons and dozens of methods documented with Jupyter Notebooks, and is available at github.com/robertoalotufu/ia898. However, this toolbox provides no morphological operators nor support for adding new operators. Prof. Lotufo has developed another toolbox for DIP courses at github.com/robertoalotufu/ia870p3. This toolbox includes comprehensive documentation of the `mmorph` toolbox, with methods now implemented in Python and serves as supplementary material in the book by Dougherty and Lotufo (2003). These toolboxes inspired the development of `morph.py`, a more recent toolbox that includes morphological operators and allows contributions from the community.

Rowe et al. (2018) present a DIP teaching method using Jupyter Notebook, showing increased student comfort with Python and exposure to polar data. Yaniv et al. (2018) introduce the SimpleITK toolkit for reproducible research and image analysis workflows through Jupyter Notebook. Although their work focuses on providing a toolkit for image analysis, our approach is different as we introduce a library specifically designed to support the teaching of DIP, including the construction of new operators. Yahya (2019) propose using Matlab to teach DIP, utilizing visual, interactive, and experimental methods.

The Matlab Image Processing Toolbox is a popular tool for image analysis in scientific and industrial applications. It offers various functions and algorithms for image-processing tasks (MathWorks, 2024). However, its proprietary nature can be a disadvantage, as it may limit access for users without licenses and pose financial challenges for those with limited budgets. In contrast, with its open community and collaborative nature, the Python environment allows for constant improvements and innovations.

Among our review of related literature, only López et al. (2016) and Rowe et al. (2018) and our study included student feedback on teaching DIP. Additionally, we replicated some of the demonstrations from the ia870p3 course using `morph.py` in Colab and presented different implementations of the same operator. Notably, our pedagogical approach only requires the `morph.py` file, which can be easily inserted into a VPL (Virtual Programming Lab for Moodle) activity for automatic code correction (Rodríguez del Pino et al., 2010).

It is also possible to teach DIP using problem-solving approaches. In López et al. (2016), an experiment was described in a class of 37 students to solve eleven DIP problems using group work in the Python language. This group work approach is questionable because the paper did not

detail whether all students could effectively absorb the DIP skills and competencies.

Table 1 summarizes the similarities and differences between the toolboxes discussed in this section.

Table 1: Comparing toolboxes/papers for teaching DIP.

Toolbox/ Paper	Teaching Tool	Focus	Morphological Operators	Student Feedback
(Silva et al., 2003)	Python	Toolbox	No	No
ia898 (Lotufo, 2017)	Jupyter	Toolbox	No	No
ia870p3 (Lotufo, 2019)	Jupyter	Toolbox	Yes	No
(Yaniv et al., 2018)	Jupyter	Toolbox	No	No
(Rowe et al., 2018)	Jupyter	Polar image	No	Yes
(Yahya, 2019)	Matlab	Interactive visual	No	No
(López et al., 2016)	Python	Problem-solving	No	Yes
<code>morph.py</code>	Jupyter/Colab [†] /PC	Compare operators	Yes	Yes

[†] Use of special conditions for execution in the Colab environment.

3 Method

This section presents the context of the DIP course (Section 3.1) and our pedagogical intervention (Section 3.2), focusing on how we used the `morph.py` library as a motivational factor. Finally, we discuss the development environments (Section 3.3).

3.1 Our Context

DIP is an elective course available to students across multiple programs at our university, including the Bachelor of Science in Computer Science and various Engineering disciplines. The course runs for 12 weeks, with four hours of class per week. The professor delivers synchronous laboratory classes to students through the projector screen, presenting Colabs containing concepts, examples, and exercises within Moodle. In 2023, we offered the course to 25 students from February to May. Additionally, we provide recorded classes created during the COVID-19 pandemic.

3.2 Pedagogical Intervention

The course covers a comprehensive overview of DIP. The first six weeks focused on equipping students with the necessary skills to develop DIP operators, including thresholds, histograms, convolution, erosion, dilation, filters, watershed, labeled, and distance transforms, as outlined in the textbook by Gonzalez and Woods (2009). During week seven, students take Exam 1, while in the following weeks (eight to ten), they focus on applying these DIP operators to solve real-world computer vision problems. In summary, the first part of the course is designed to provide students with the skills necessary to develop DIP operators. In contrast, the second part builds on this foundation by applying these operators to solve various computer vision problems. Week eleven is reserved for review, and the course concludes with Exam 2 (a final exam) in week twelve.

The evaluation strategy for this course consists of four individualized and parameterized

assignments, accounting for 15% of the final grade. These assignments focus on the most recent topic covered in class and are provided to students before each session. They are automatically evaluated by integrating `MCTest`, Moodle, and VPL (Zampirolli et al., 2020, 2021). Further, Exam 1 (30% of the final grade) consists of similar exercises. For the final individual project (15% of the final grade), students must develop a Colab to solve a DIP problem. Exam 2, worth 40% of the final grade, will be similar in format to the project. Both exams are to be completed within a two-hour class period.

3.3 Development Environments

The DIP course uses the `morph.py` library as part of the teaching materials, with methods implemented in Python. This library can be used with Colab, `MCTest`, VPL activities, or in command lines. The assessment part is developed in `MCTest` to generate programming exercises, and students must submit their answers in VPL activities in Moodle. As detailed below, `morph.py` can facilitate the process in all these situations.

3.3.1 Course Material

Figure 1 provides an overview of the pedagogical intervention employed in the DIP course, utilizing the `morph.py` library, which encompasses 71 methods, including operations (such as minimum, maximum, and negation) and operators (such as erosion and dilation). While the ia870p3 course (refer to Section 2) covered some algorithms from the `mmorph` toolbox, neither that course nor our course covered the entire range of functionalities the toolbox offers. The course material consisted of six conceptual notebooks (files `class.0*` in Figure 1) covered during the initial six weeks. Additionally, eight documents focus on morphological operators (files `mm*` in Figure 1). These resources were used until Exam 1, marking a significant milestone in the course progression. After Exam 1, the course incorporates a set of 27 notebook demonstrations (files `*.mmd` in Figure 1) that showcase the practical application of computer vision in real-world scenarios, adapted from `mmorph`. These notebook documentations are available in `ipynb` format and can be accessed using literate programming tools such as Colab or Jupyter (Knuth, 1984). Although the `morph.py` library can also be used on a computer console, its main application is in programming exercises and didactic classes with Colab. These exercises are generated in `MCTest` and include automatic correction submitted by students in VPL activities in Moodle (Zampirolli et al., 2020, 2021). `MCTest` and VPL activities can also be used with `morph.py`.

3.3.2 Automated Evaluation

The automatic assessment process for this course comprises four customized tasks that assess the most recent topic discussed in class during the first part until Exam 1. These tasks are distributed to students before each class and are automatically evaluated using a combination of `MCTest`, Moodle, and VPL as reported by Zampirolli et al. (2020, 2021). Exam 1 also follows the same process. All methods implemented in `morph.py` are available in the VPL activity runtime files. Students must correctly call these methods or make adaptations to ensure the test cases execute correctly. For example, we can create a new morphological erosion question where the origin of a *structuring function* b must include an additional argument in the method. Note that this simple change has no solution in traditional libraries or in `morph.py`. Therefore, the students must develop their solutions. For each student, it is also possible to draw a different origin for

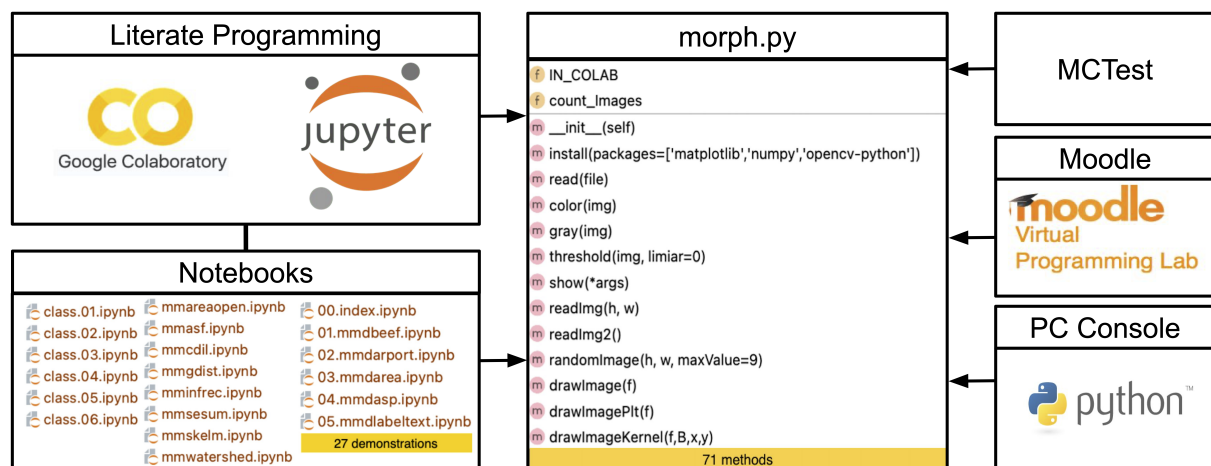


Figure 1: Overview of the pedagogical intervention used (adapted from Zampirolli et al. (2024)).

the structuring function b , thus making plagiarism more difficult. During Exam 1, the SEB (Safe Exam Browser – safeexambrowser.org) feature is used, preventing students from accessing the internet to find ready-made solutions. In the initial lists, even the student who consulted chatbots at the time could not find a correct solution to this question. These processes are detailed in Appendices 1, 2, and 3 at the end of this paper.

4 Learning and Developing DIP Through `morph.py`

We utilize `morph.py` to gradually introduce the fundamental concepts of DIP programming one by one, allowing students to understand how these concepts combine to create a complete DIP application for ten weeks without feeling overwhelmed. In this section, we will explain the specific details of this process, highlighting the importance of including code for various methods to enhance the reproducibility of this teaching approach. The codes can be accessed at github.com/fzampirolli/morph.

4.1 Initial Part

In week 1, we introduce the initial structure of the `morph.py` library, as illustrated in Code 1, with the `install` method. This method, with the default arguments, will install the latest versions of the Matplotlib (matplotlib.org), OpenCV (opencv.org), and Skimage (scikit-image.org) libraries if they are not already installed.

To use this method in Colab, simply download the `morph.py`, run the cell with the code `from morph import *` and `mm.install()`, as illustrated in Code 2.

4.2 Image Reading

The method shown in Code 3 demonstrates how to read images using different approaches. If a Google Drive ID is provided, starting with `'id='` (line 15), or a URL beginning with `'http'` is given (line 17), the image will be retrieved from that source. If neither is provided, the image

```

1 import matplotlib.pyplot as plt, numpy as np, cv2, requests, sys, subprocess
2 from PIL import Image; from skimage import io
3 class mm(object): """ A helper class for image processing tasks. """
4     count_images, IN_COLAB = 0, 'google.colab' in sys.modules # INITIALIZATION
5     def install(packages=['matplotlib', 'scikit-image', 'opencv-python']):
6         """ This method will install the packages
7         Input: <packages> list of packages.
8         Example:
9             mm.install(['matplotlib', 'scikit-image'])
10        """
11        for p in packages:
12            subprocess.check_call([sys.executable, "-m", "pip", "install", p])

```

Code 1: Initial part of the `morph.py` containing the method for package installation.

```

1 # download morph.py from GitHub
2 # !wget https://raw.githubusercontent.com/fzampirolli/morph/main/morph.py
3
4 from morph import *
5 mm.install()

```

Code 2: Code for downloading `morph.py` from GitHub.

will be read from the folder where the code cell is executed (line 20). To utilize this method in the latter case, use the following syntax: `img = mm.read('image.png')`.

```

1 def read(file):
2     """ Reads an image from a local file path or URL.
3     Input: <str> File path or URL (full or 'id=keyGoogleDrive').
4     Output: the read image.
5     Example:
6         img_local = mm.read('image.png')
7         img_url = mm.read('https://example.com/image.jpg')
8         img_gdrive = mm.read('id=keyGoogleDrive')
9     """
10    if file.startswith(('http', 'id=')):
11        url, pre = '', 'https://drive.google.com/file/d/'
12        if pre in file:
13            url = 'https://drive.google.com/uc?export=view&id='
14            url += file[len(pre):].split('/')[0]
15        elif file.startswith('id='):
16            url = 'https://drive.google.com/uc?export=view&id=' + file[3:]
17        else:
18            url = file
19        return io.imread(url)
20    else:
21        return cv2.imread(file)

```

Code 3: Method for reading an image from a local file path or URL.

4.3 Image Format Conversions

After reading an image, it is often necessary to convert it to a desired format. While many image formats are available, we recommend using the RGB format for color images and converting them

to this format using the `mm.color()` method in `morph.py` (Code 4). Additionally, we suggest using images with values between 0 and 255 of type 'uint8' whenever possible. To convert an image to grayscale, use the `mm.gray()` method (Code 5).

```

1 def color(img):
2     """ Converts an image to RGB color space.
3     Input: <numpy.ndarray> Image in BGR, grayscale, or RGBA format.
4     Output: RGB image in <numpy.ndarray> format.
5     Example:
6         img = mm.read('image.png')
7         img_rgb = mm.color(img)
8     """
9     if len(img.shape) == 2:
10        return cv2.cvtColor(img, cv2.COLOR_GRAY2RGB)
11    elif len(img.shape) == 3 and img.shape[2] == 4:
12        return cv2.cvtColor(img, cv2.COLOR_RGBA2RGB)
13    elif len(img.shape) == 3 and img.shape[2] == 3:
14        return cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
15    else:
16        raise ValueError("Unsupported image format.")

```

Code 4: Method to convert an image to the RGB color space.

```

1 def gray(img):
2     """ Converts a color image to grayscale.
3     Input: <numpy.ndarray> Input color image.
4     Output: grayscale image.
5     Example:
6         img = mm.read('image.png')
7         img_gray = mm.gray(img)
8     """
9     if len(img.shape) == 3 and img.shape[2] == 4:
10        return cv2.cvtColor(img, cv2.COLOR_BGRA2GRAY)
11    else:
12        return cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)

```

Code 5: Method to convert an image to the grayscale space.

Finally, to convert a grayscale image to binary, please refer to Code 6 for an example. When the second parameter `threshold>0` exists in this method, all pixels greater than `threshold` will receive 255. Furthermore, if this argument does not exist, as in the comment on line 7, the Otsu method will perform the thresholding, which calculates the value of the threshold automatically (Otsu, 1979).

Please take note that, for image conversion using this method, it is significantly easier for students to remember the command:

```
th = mm.threshold(img)
```

compared to the command mentioned in line 10 of Code 6:

```
value, th=cv2.threshold(img, 0, 255, cv2.THRESH_BINARY
+ cv2.THRESH_OTSU)
```

Furthermore, it's worth highlighting that this method can be further enhanced to accommodate the conversion of various image types and can be integrated with alternative libraries in

```

1 def threshold(img, threshold=0):
2     """ Thresholds an input image by a threshold value or using Otsu's method.
3     Input: <numpy.ndarray> Input image to be thresholded.
4     Output: <numpy.ndarray> Thresholded image.
5     Example:
6         img = mm.read('image.png')
7         th = mm.threshold(img)
8     """
9     if threshold == 0:
10        value,th=cv2.threshold(img,0,255,cv2.THRESH_BINARY + cv2.THRESH_OTSU)
11    else:
12        value,th=cv2.threshold(img,threshold,255,cv2.THRESH_BINARY)
13    return th

```

Code 6: Method for converting a grayscale image to binary.

addition to OpenCV. This allows for the encapsulation of implementation details, eliminating the need for students to memorize them to utilize the method effectively.

4.4 Displaying Multiple Images

The code 7, adapted from the `mmorph` toolbox, presents a straightforward method for displaying multiple images. To use this method, call `mm.show(imgRGB, img1, img2)`. In RGB format, the first image acts as the base, while the subsequent binary images are overlaid with colors (see lines 11 and 12). Even if the notebook is not executed in Colab (`mm.IN_COLAB=False`, as defined in Code 1), the code after line 18 will save the image to the local computer's local disk when used in Jupyter Notebook or a local computer's console.

```

1 def show(*args):
2     """ This method will draw images f
3     Input: <*args> set of images f_i, where i>0 is binary image
4     Output: image drawing
5     Example:
6         f1, f2 = np.zeros((100, 100,3)), np.zeros((100, 100))
7         f2[50:60, 50:60] = 1
8         mm.show(f1, f2)
9     """
10    f = args[0].copy()
11    colors = [[255,0,0], [0,255,0], [0,0,255], [255,0,255], [0,255,255],
12             [255,255,0], [255,50,50], [50,255,50]] # red, green, blue, cyan, ...
13    for i in range(1,len(args)):
14        if i >= len(colors):
15            break
16        f[args[i] > 0] = colors[i-1]
17    _ = plt.imshow(f, "gray")
18    if not mm.IN_COLAB:
19        plt.savefig('fig_' + str(mm.count_Images).zfill(4) + '.png')
20        mm.count_Images += 1

```

Code 7: Method for drawing multiple images, where the first image is RGB.

4.5 Morphological Erosion

In `morph.py`, we conventionally represent morphological operators in the following way. First, if we ignore the weights of neighboring pixels in the *structuring element* B of a morphological operator, such as in erosion (Banon & Barrera, 1994), we denote this operation as `mm.ero0(img)`.

Refer to Code 8 for an example. Morphological erosion calculates, for each pixel in the output image, the minimum value among the corresponding pixels in the input image, considering the neighborhood defined by *B*, usually with the center of *B* as the origin.

```

1 def ero0(f, B=np.ones((3, 3), dtype='uint8')):
2     """ Creates an erosion of the input image f by the structuring element B.
3     Input: f (ndarray): The input image; B (ndarray, optional): The str. elem.
4     Output: The result of the erosion.
5     Example:
6     mm.ero0(f, B=np.ones((5, 5), dtype='uint8'))
7     """
8     H, W, Bh, Bw, g = f.shape, B.shape, f.copy()
9     for y in range(H): # Loops over each pixel in the input image.
10        for x in range(W):
11            for by in range(Bh): # Loops over each neighbor of the current pixel.
12                for bx in range(Bw):
13                    neig_y, neig_x = int(y + by - Bh/2 + 0.5), int(x + bx - Bw/2 + 0.5)
14                    # Check if the neighbor is within the bounds of image and B.
15                    if B[by, bx] and 0 <= neig_y < H and 0 <= neig_x < W:
16                        # Update current pixel with the minimum value of its neighbors.
17                        if g[y, x] > f[neig_y, neig_x]:
18                            g[y, x] = f[neig_y, neig_x]
19    return g

```

Code 8: Morphological erosion without weights on neighboring pixels.

Another option to consider is the incorporation of weights on neighboring pixels, which is achieved through a *structuring function* *b*. For an illustration, refer to Code 9. This modification can be observed in lines 15, 17, and 18 of Codes 8 and 9. In the latter case, we have the difference $- b[by, bx]$. It is crucial to draw the students' attention to this point, as this difference may be less than zero in some instances, which is often undesirable. In such cases, additional post-processing may be necessary to clamp it to zero.

```

1 def erol(f, b=np.ones((3, 3), dtype='uint8')):
2     """ Creates an erosion of the input image f by the structuring function b.
3     Input: f (ndarray): The input image; b (ndarray, optional): The str. func.
4     Output: The result of the erosion.
5     Example:
6     mm.erol(f, b=np.ones((5, 5), dtype='uint8'))
7     """
8     H, W, bh, bw, g = f.shape, b.shape, f.copy()
9     for y in range(H): # Loops over each pixel in the input image.
10        for x in range(W):
11            for by in range(bh): # Loops over each neighbor of the current pixel.
12                for bx in range(bw):
13                    neig_y, neig_x = int(y + by - bh/2 + 0.5), int(x + bx - bw/2 + 0.5)
14                    # Check if the neighbor is within the bounds of the image.
15                    if 0 <= neig_y < H and 0 <= neig_x < W: # HERE IS DIFFERENT
16                        # Update the current pixel with the minimum value of b.
17                        if g[y, x] > f[neig_y, neig_x] - b[by, bx]: # HERE IS DIFFERENT
18                            g[y, x] = f[neig_y, neig_x] - b[by, bx] # HERE IS DIFFERENT
19    return g

```

Code 9: Morphological erosion with weights on neighboring pixels.

When comparing the OpenCV erosion usage, as shown in Code 10, with these three implementations, it's important to note that they do not yield the same results. The student should

understand the differences between them. One potential for a more efficient implementation is to increase the image dimensions, which can exclude the border pixels from the neighborhood calculations. This approach offers a trade-off between efficiency and accuracy. Another strategy is to develop a highly efficient implementation in ANSI C and then create a Python method that invokes this implementation, such as the `mmorph` toolbox, produced by AdessoWiki (Machado et al., 2011). Moreover, erosion can be implemented on a GPU for even better performance (Zampirolli & Filipe, 2017).

```
1 def ero(f, b=np.ones((3, 3), dtype='uint8')):
2     """ Creates an erosion of the input image f by the structuring function b.
3     Input: f (ndarray): The input image; b (ndarray, optional): The str. func.
4     Output: The result of the erosion.
5     Example:
6         mm.ero(f, b=np.ones((5, 5), dtype='uint8'))
7     """
8     return cv2.erode(f, b)
```

Code 10: Morphological erosion of OpenCV.

4.6 A Simple Example of Using the Library `morph.py`

The code for displaying the image (b) of Figure 2 is provided in Code 11. In line 9, the command `np.ones((7, 7), dtype='uint8')` defines the structure element and can be replaced with the `mm.sebox(2)` method, which is also available in the `morph.py` library. If the code cell in Code 2 has already been executed in the Jupyter Notebook or Colab, there is no need to execute lines 1 to 5 in Code 11 again.

```
1 # download morph.py from GitHub
2 # !wget https://raw.githubusercontent.com/fzampirolli/morph/main/morph.py
3
4 from morph import *
5 mm.install()
6 img = mm.read('https://www.dropbox.com/s/ekjbzp14jt90bfq/00004.jpg?dl=1')
7 img = img[25:120, 30:285] # cropping the image - Figure 2-(a)
8 th = mm.threshold(mm.gray(img), 30)
9 mm.show(img, th-mm.ero(th, np.ones((7, 7), dtype='uint8'))) # Figure 2-(b)
```

Code 11: Code for downloading `morph.py` from a GitHub key, reading an image from a Dropbox URL, and displaying the image.

5 Using `morph.py` for DIP in a Didactic Approach

As discussed in the previous section, using the `morph.py` library to highlight the edges of objects is straightforward, provided that these objects are set against a uniform background (as seen in Figure 2, where the black background has a value of zero). This section will introduce additional operators and examples from the first six weeks of the course.

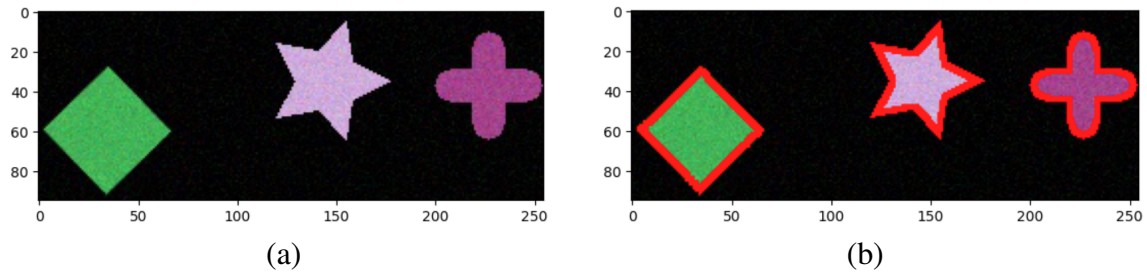


Figure 2: Image (a) shows the cropped portion of the image read in lines 6 and 7 of Code 11. Image (b) provides an example of using `mm.threshold()` and `mm.show()` in lines 8 and 9 of Code 11.

5.1 Operator `mm.sesum()`

A first example is the operator `mm.sesum()`, adapted from the `mmorph` toolbox. Unlike the morphological operators of erosion and dilation, the `mm.sesum()` operator increases the dimensions of the resulting image. While the erosion operator, outlined in the previous section, and the dilation operator (which acts as the complement of erosion by determining the maximum value among neighboring pixels) produce output images that maintain the exact dimensions as the original, the `mm.sesum()` operator functions differently by expanding the dimensions of the resulting image. It creates a structuring function nb by the Minkowski sum of b , defined as (Banon & Barrera, 1994; Lotufo, 2019):

$$nb = \underbrace{((b \oplus b) \oplus \dots \oplus b)}_n, \quad n \geq 0.$$

Consider $0b = \text{mm.sesum}(b, 0)$ as the identity and $1b = b \oplus b = \text{mm.sesum}(b, 1)$, which equals the dilation of b by b , resulting in a 5×5 image. Also, consider that b has odd dimensions and the origin is the center.

To exemplify the use of `mm.sesum()` in `morph.py`, we will use the cross structuring function b of size 3×3 , defined as `b = np.ones((3, 3), dtype='uint8')` with zeros at the corners: `b[0, 0] = b[0, 2] = b[2, 0] = b[2, 2] = 0`. The difference between a structuring element and a structuring function is that in the first case, only non-zero values are considered neighbors without considering these values in the operator, as exemplified in Code 8. In `morph.py`, you can simply use `b = mm.secross()`. If you want a structuring element/function of size 5×5 , just write `mm.secross(1)`, which utilizes `mm.sesum(b, 1)`. If you want a 7×7 element, use `mm.sesum(b, 2)`, which will use the result of the Minkowski sum of the 5×5 element by the 3×3 element, and so on. Figure 3 illustrates these three cross-structuring elements.

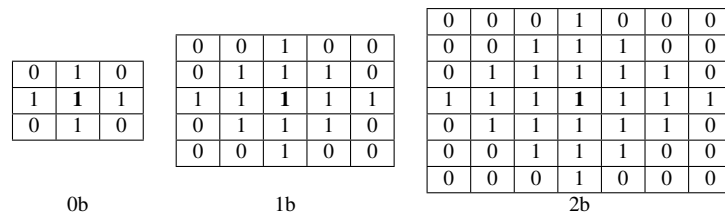


Figure 3: Comparison of structuring elements cross $0b$, $1b$, and $2b$, with the center in bold.

5.2 Operator `mm.dist()`

A widely used operator in image segmentation is the Distance Transform, which analyzes the topology of each object and can be implemented in various ways. The Distance Transform calculates the distance from each object pixel (> 0) to the nearest background pixel ($= 0$) using a specified metric. The following presents a straightforward and didactic approach using successive erosions (Huang & Mitchell, 1994), as illustrated in Section 6.1.3. Code 12 demonstrates that the image f is computed using a cross-shaped structuring element, scaled by a factor of 9 (this value equals the greatest possible distance). The structuring function b_0 is a 3×3 matrix where all elements are -1 except for the center, which is 0. The image g is obtained by applying erosion three times to f using b_0 . The images f , g , and the structuring function b_0 are displayed in Figure 4, and the result g is shown in Figure 5 to enhance visualization using the `mm.drawImagePlt()` method of `morph.py`, as demonstrated in line 5 of Code 12.

```
1 f = mm.secross(3) * 9
2 b0 = mm.sebox() * -1
3 b0[1,1] = 0
4 g = mm.erol(mm.erol(mm.erol(f,b0),b0),b0)
5 mm.drawImagePlt(g)
```

Code 12: Code example of the distance transform by erosions.

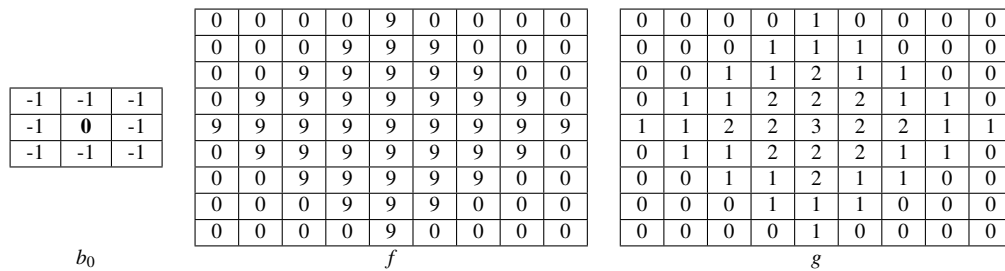


Figure 4: Distance transform of the image f using the structuring function b_0 , resulting in the image g .

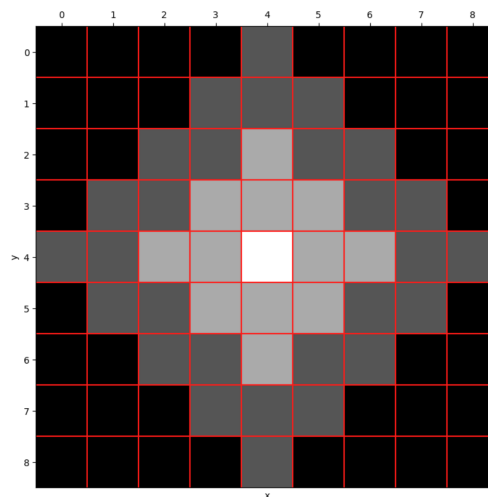


Figure 5: Distance transforms g as image.

As presented in this section, the iterative erosion method for distance transform is computationally inefficient. Instead, the `morph.py` implementation uses the OpenCV library with the `cv2.distanceTransform()` method efficiently calculates the Euclidean distance from each pixel to the nearest background pixel using a specified neighborhood size.

5.3 Solving Problems: Maze

A variation of the Distance Transform is the geodesic distance (Lantuejoul & Beucher, 1981), implemented in `morph.py` with the method `mm.gdist(f, g, b)`. This method calculates the geodesic distance by iteratively eroding the negation of g intersected with f , using the structuring function b . In Figure 6, the input image (a) defines two markers at the entrance (m_1) and exit (m_2) of the maze. Images (b) and (c) show the `mm.gdist(f, m1)` and `mm.gdist(f, m2)` results, respectively, with the default b equal to `mm.sebox()`. These two resulting images are summed, and the minimum is calculated, highlighting the path in red in image (d). The minimum distance calculated was 47 in line 19 of Code 13.

```
1 url='https://drive.google.com/file/d/1jRzJ6he1sFc-AH7tGRXWYqxbASu3CUdy/view?usp=sharing'
2 img = mm.read(url)
3 img = cv2.resize(img, (35, 35))
4 imgRGB = mm.color(img)
5 img1 = mm.gray(imgRGB)
6 f = (mm.threshold(img1)/255).astype('uint16')
7 plt.figure(figsize=(7, 7))
8 mm.show(f) # image (a)
9
10 m1 = np.zeros_like(f).astype('uint8')
11 m2 = np.zeros_like(f).astype('uint8')
12 m1[1, 17] = 1
13 m2[-2, 17] = 1
14
15 b = mm.gdist(f, m1) # image (b)
16 c = mm.gdist(f, m2) # image (c)
17
18 d = b + c
19 min = np.amin(d[d != np.amin(d)]) # minimum
20 print(min)
21 mm.show(imgRGB, d==min) # image (d)
```

Code 13: Code example of the geodesic distance.

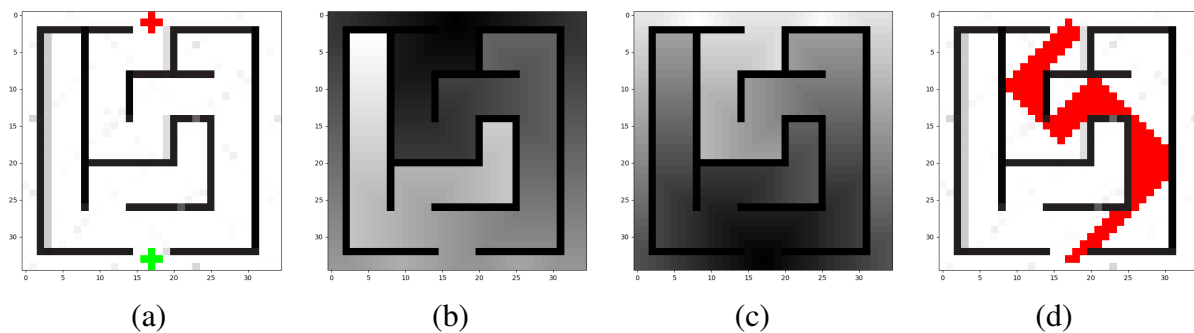


Figure 6: Example of using geodesic distance transforms: (a) input image with two markers; (b) and (c) geodesic distances of the first and second markers; (d) output image.

Will the solution implemented in `mm.gdist()` solve the mazes created by Singh et al. (2024)? See Figure 7 for the solution using Code 13. The problem is that a very long time is required to generate the two geodesic transforms. It was necessary to reduce the size of the image presented in the article and limit the operator `mm.gdist()` to 2000 iterations, which consumed 314 seconds per execution on a MacBook Pro with 16 GB of RAM and a 2.5 GHz Intel Core i7 Quad-Core processor. The minimum distance calculated was 622 in line 19 of Code 13 using the structuring function `mm.secross()` in lines 15 and 16 of this code. The codes are available at github.com/fzampirolli/morph, in folder Expanded/Maze.

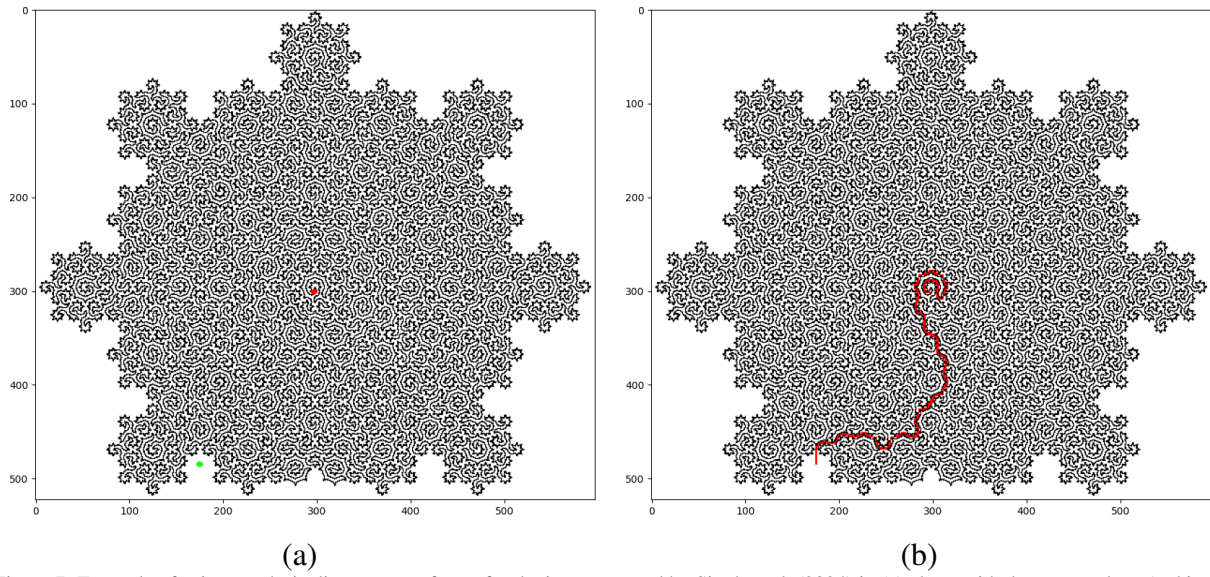


Figure 7: Example of using geodesic distance transforms for the image created by Singh et al. (2024) in (a) along with the two markers (red in the center and green in the bottom left corner). In (b), the result is presented.

Geodesic Distance Equation

The geodesic distance transform y of an image f with marker g , both binary images in $\{0, 1\}$, using a structuring function b , is calculated through the iterative process given by $g_{\text{neg}} = y_0 = 1 - g$ and $f_{\text{neg}} = \max - f * \max$ (where \max is the maximum possible distance in the image). For each iteration $n > 0$, the update is given by:

$$y_{n+1} = \text{logical_xor}(g_{\text{neg}}, f_{\text{neg}}) \cap (y_n \cup \text{mm.cero}(g_{\text{neg}}, f_{\text{neg}}, b, n)).$$

The process continues until $y_{n+1} = y_n$. The operator `mm.cero()` used in this equation is defined in Code 14. It performs conditional erosion of a marker image g using an input image f , a structuring function b , and a specified number of iterations n . Initially, the method computes the maximum between f and g . Each iteration applies erosion to this maximum result and updates it by taking the maximum between the eroded image and g . The final output results from performing n iterations of this conditional erosion process. It is important to note that all operations ($-$, $*$, \cap , \cup) are performed pixel-by-pixel. For example, \cup corresponds to `np.maximum()` in Code 14.

```

1 def cero(f,g,b=np.zeros((3,3),dtype='uint8'),n=1):
2     """This operator will be erosion g with maximum f, n times
3     input:
4         - f: input image
5         - g: mark image
6         - b: neighbors
7         - n: number of iterations
8     output:
9         - y: result of condictional erodes
10    """
11    y = np.maximum(f,g)
12    for i in range(n):
13        d = cv2.erode(y,b)
14        y = np.maximum(d,g)
15    return y

```

Code 14: Code from conditional erode.

6 Using morph.py for DIP in Exams

In this section, we will explain how the `morph.py` library was utilized in the two exams conducted for the DIP course. Building on the pedagogical intervention discussed in Section 3.2 and the development environments described in Section 3.3, we will also explore how `morph.py` serves as a resource that is resilient to Generative AI for a diverse array of assessment tasks, including structured, semi-structured, and more open-ended formats.

6.1 Exam 1

Exam 1 was designed parametrically, with each student randomly selecting 3 questions, as detailed in this section.

6.1.1 Question 1

In Zampirolli et al. (2019), the process of creating parametric questions was presented, along with a detailed example of a question designed for a programming logic course (CS1) focusing on matrices. This question was adapted for DIP, as illustrated in Figure 8. Additionally, the process of creating a parametric question is summarized in Appendix 1 for a question used in List 1.

1. An entry $a_0 = (i, j)$ of a matrix is called **North lesser** if its value is **lesser** than the neighbouring ones positioned at a_8, a_1, a_2 , as indicated in the table. See in figure an example matrix/image f with dimensions 9×17 and its corresponding **North lesser**, where “-” means “-1”, without considering the border elements.

$a_8 = \text{Northwest}$	$a_1 = \text{North}$	$a_2 = \text{Northeast}$
$a_7 = \text{West}$	$a_0 = (i, j)$	$a_3 = \text{East}$
$a_6 = \text{Southwest}$	$a_5 = \text{South}$	$a_4 = \text{Southeast}$

It is possible to solve this problem using the dilation or erosion operators defined below:

$$\text{dil}(f, B)(x) = (f \oplus B)(x) = \delta_B(f)(x) = \max\{f(y) : y \in \mathbb{B}_x \cap \mathbb{E}\}$$

$$\text{ero}(f, B)(x) = (f \ominus B)(x) = \varepsilon_B(f)(x) = \min\{f(y) : y \in \mathbb{B}_x \cap \mathbb{E}\}$$

$\forall x \in \mathbb{E}, f \in K^{\mathbb{E}}$ or $f \in [0, k]^{\mathbb{E}}$, k is a positive integer representing the maximum number of grayscale levels in the set $[0, k]$ for the digital image f defined over the domain \mathbb{E} . Also, consider $\mathbb{B} \subset \mathbb{Z} \times \mathbb{Z}$ as the structuring element (neighborhood/kernel), with its origin at its center.

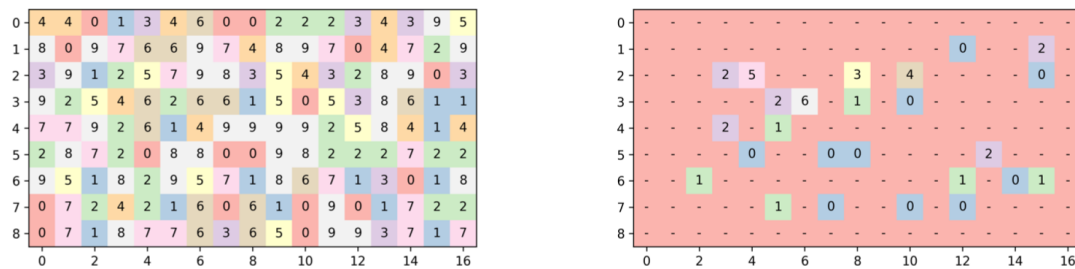


Figure 8: Question 1 on Exam 1 involves using adapted erosion or dilation.

The text highlighted in blue emphasizes the parametric component, which varies for each generated exam. This portion has been removed from the PDF version, which will be printed and delivered individually to each student. Students are required to solve this question in a Moodle VPL activity. This question can be easily adapted for DIP and was used in Exam 1. Stu-

dents should apply erosion or dilation operators using a 3×3 neighborhood in this case. For the **North lesser** case, students must utilize the structuring element `B=np.array([[1,1,1],[0,1,0],[0,0,0]])` together with the function `ero0(f,B)` (adaptation code is shown in Code 8, excluding border elements). The input matrix/image, shown in Figure 8-(left), is provided as input data in the test cases, while the output is displayed on the right.

To read this input image, there is also specific code in `morph.py` library, `readImg2()`, which can be used when the dimensions of the image are not known, as shown in Code 15. When the image dimensions are known, the `readImg(width, height)` method is also available. This code reads input lines containing pixel values, splits each line into individual pixel values, converts them to integers, and stores them in a list. This process continues until no more input (the last line is empty). Finally, it converts the list into a NumPy array with an unsigned 8-bit integer data type and returns the resulting array.

```

1 def readImg2():
2     """ This method reads an image of varying size from standard input.
3     Example:
4         f = mm.readImg2()
5         255  0  255
6         128 64  192
7         0   192 128
8
9     """
10    f = []
11    read_row = input()
12    while read_row: # Read each line of the input until there is no more input
13        # Split the line into individual pixel values and convert them to int.
14        row = [int(i) for i in read_row.split() if I]
15        f.append(row) # Add the row to the list of rows.
16        read_row = input()
17    return np.array(f).astype('uint8')

```

Code 15: Read a variable size image.

Even if this question is presented as an activity requiring internet research, the student will not find a ready-made solution. For example, when inquiring about generative models like ChatGPT (chat.openai.com), the answers will not be readily available. Additionally, this question has 16 variations, which include different values and dimensions for the input image. There are eight variations for the cardinal directions, with two variations each for greater or lesser. Furthermore, it is possible to create even more significant variations by using a neighborhood of 5×5 or 7×7 . For a reflection on generative models used in teaching, see Section 6.3.

6.1.2 Question 2

The second question of Exam 1 instructed the student to read an input image `f` and apply a filter from the OpenCV library. Such a parametric question was created for each of the filters:

1. `cv2.medianBlur(f, bw),`
2. `cv2.filter2D(f, -1, bw),` and
3. `cv2.GaussianBlur(f, (bw, bh), 0).`

Each student received a random question. For the filter `cv2.medianBlur(f, bw)`, we randomly assigned each student a neighborhood `bw` of size 3, 5, or 7. We generated a random input image, as Figure 9 illustrates.

1. The convolution of an image f with a kernel b ($bw \times bh$) is defined as follows:

$$(f * b)[x, y] = \sum_{m=0}^{bw-1} \sum_{n=0}^{bh-1} f[x + m - bw//2, y + n - bh//2] b[m, n]$$

Calculate the convolution using `cv2.medianBlur(f, 5)` (b has dimensions 5×5), as shown in the example below:

Note: The 2D dimensions of f change in each test case. The left side shows the input image, and the right side shows the output image.

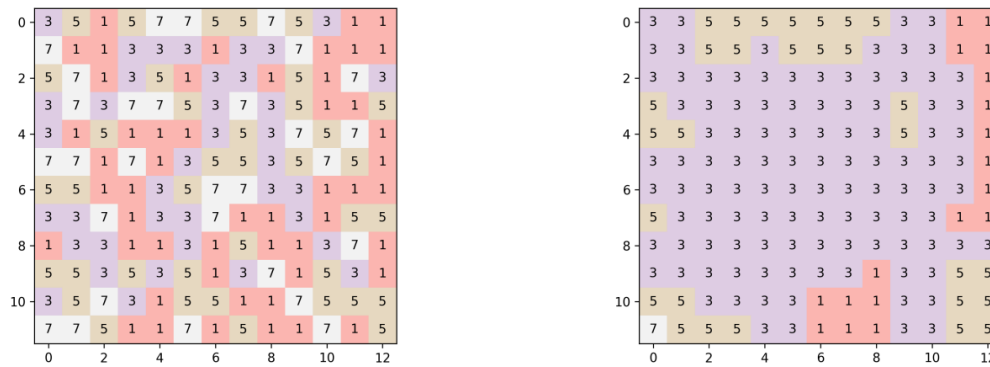


Figure 9: Question 2 on Exam 1 involves using the `cv2.medianBlur()` filter.

6.1.3 Question 3

The third question on Exam 1 required the student to calculate the distance transform (DT – the shortest distance to a pixel with a value of zero) using successive erosions, as illustrated in Figure 10. Each student is assigned a different image f and neighborhood b , which can vary significantly. Additionally, multiple test cases are generated for each student, with the randomly assigned neighborhood b determining the type of distance measure used. For the 2D case, various distance types are available, including Euclidean, City-Block, and Chessboard (Cuisenaire, 1999). In some variations of this question, students may also be asked to report the number of erosions performed until convergence to the DT.

These three questions presented in Exam 1 assess students' abilities and competencies in creating operators for DIP. Although students have access to the `morph.py` library during the assessment, they will need to make adaptations to ensure their solutions pass various test cases generated automatically for the Moodle VPL activity, following the method defined in Zampirolli et al. (2021), and summarised in Appendices 1, 2, and 3.

6.2 Exam 2

Following Exam 1, students are expected to learn how to use the methods offered by `morph.py` library and other DIP libraries. As a result, the second part of the course includes several practical examples of problem-solving using DIP, building upon concepts introduced in previous sections and evaluated in the final exam.

1. Implement the **1D Distance Transform (DT)** using successive erosions, considering erosion defined as:

$$ero(f, b)(x) = (f \ominus b)(x) = \varepsilon_b(f)(x) = \min\{f(y) - b(x - y) : y \in \mathbb{B}_x \cap \mathbb{E}\}$$

where $x \in \mathbb{E} \subset \mathbb{Z}$, $f \in K^{\mathbb{E}}$ or $f \in [0, k]^{\mathbb{E}}$, k is a positive integer representing the grayscale levels of the digital image with domain \mathbb{E} . Also, consider $b \in \mathbb{Z}^{\mathbb{B}}$ and $\mathbb{B} \subset \mathbb{Z}$, as the structuring function (neighborhood/kernel), with its center being the origin (in this example, the image f has 51 pixels). Implement the following DT algorithm, as shown in the example below:

$$\begin{aligned} DT : K^{\mathbb{E}} \times \mathbb{Z}^B &\rightarrow K^{\mathbb{E}} \\ DT(f, b) &= g \end{aligned}$$
$$\begin{array}{l} g = f; \\ i = 0; \\ \forall g \neq \varepsilon_b(g) \\ \quad g = \varepsilon_b(g); \\ \quad i ++; \end{array}$$

Example inputs (b and f):

[illegible]

Example of the corresponding output $(DT(f, b))$:

3 2 0 0 2 3 4 6 7 8 7 6 4 3 2 0 2 3 4 6 7 8 7 6 4 3 2 0 2 3 4 6 7 8 8 7 6 4 3 2 0 2 3 4 4 3 2 0 2 3 4

Figure 10: Question 3 on Exam 1 for 1D Distance Transform (DT) using successive erosions.

In Exam 2, students were tasked with detecting nine classes, encompassing 27 randomly generated geometric objects categorized by their color, size, and rotation. Figure 2-(a) illustrates a selection of three objects. The following URL was presented in a student’s question prompt:

<https://dropbox.com/s/ekjbzp14jt90bfq/00004.jpg>

These were the nine classes of random objects:

```
objects=['Triangle', 'Square', 'Pentagon', 'Hexagon',  
        'Heptagon', 'Circle', 'Ellipse', 'Star', 'Cross']
```

In addition to this image, a TXT file containing each object’s class and bounding box on separate lines was provided. Here’s an example:

<https://dropbox.com/s/s5z2muo0bx4calx/00004.txt>

The following is a TXT file snippet showing classes 0, 1, and 2, which correspond to Triangle, Square, and Pentagon, respectively. In the first column is the class, and in the subsequent columns are the bounding box values represented as a percentage of width (columns 2 and 4) and height (columns 3 and 5):

0	0.2895	0.2007	0.3750	0.2862
0	0.0938	0.6678	0.1595	0.7336
0	0.4934	0.3076	0.5954	0.4095
1	0.0543	0.0872	0.1595	0.1924
1	0.3421	0.7911	0.4079	0.8569
1	0.2664	0.4424	0.3322	0.5082
2	0.4211	0.6661	0.5263	0.7714
2	0.5757	0.4688	0.6678	0.5609
2	0.0938	0.8355	0.2122	0.9539

Each student received a unique image and a corresponding TXT file and was tasked with assessing the accuracy of their detection method. See all images and TXT files used in Exam2 in folder `Expanded/Exam2/image` at github.com/fzampirolli/morph. The question was designed with automatic corrections in the VPL activity, including various test cases, which means several pairs of images and corresponding TXT files containing bounding boxes. During the classes, it was mentioned that the datasets used to train convolutional neural networks often consist of pairs of images and accompanying TXT files, as seen in popular frameworks like YOLO (Hussain, 2023). Appendix 4 explains how to use Roboflow (roboflow.com) to detect the geometric objects presented in this section using transfer learning in YOLO.

6.3 `morph.py` and Generative AI

A decade ago, the use of Artificial Intelligence (AI) in education was limited due to its building complexity. Nevertheless, compelled by the fast-changing digital world, generative AI has emerged as a leading tool in education, experiencing a significant surge in usage over just a few years that surpassed that of social media platforms.

Generative AI (GenAI) has been a controversial topic in the educational context. On one hand, it offers new or enhanced possibilities, such as personalized learning experiences and multimodal learning resources. On the other hand, it raises several concerns, including increased dependency on technology, which may lead to the erosion of problem-solving skills, superficial understanding of content, and assignment cheating (Giannakos et al., 2024).

Realizing GenAI's benefits in education demands evolving curricula, teaching approaches, and assessment strategies. Regarding the former, instead of relying only on traditional exams, educators should design continuous tasks that use AI capabilities to improve human skills throughout the term. For example, these tasks must incorporate real-world problems, collaboration, system thinking, and GenAI-resilient opportunities for self-reflection. The `morph.py` library supports educators in designing tasks in various formats (from very structured to freer), allowing students to engage and learn DIP without interference from AI.

In a simple test, we used several GenAI platforms to evaluate our resilience in solving Question 3 from Exam 1 (Section 6.1.3), including Copilot, Poe, Perplexity, and ChatGPT. Although they all generated code solutions (see example Code 16), the initial attempts did not work and required more prompt interaction. The input was a copy-paste of all PDF content of Figure 10. Moreover, we note that the erosion method generated by all GenAI was very dissimilar from the one used by `morph.py`. In other words, GenAI erosion's method could not correctly calculate the distance from a central point.

7 Results and Discussion

We conducted one voluntary survey to gather data from students in our DIP course. The survey, administered after Exam 2, included 5 Likert-based questions to gather feedback on the course content and strategies.

For the students who completed the course, the survey received 15 responses ($\approx 60\%$) for the Likert questions (Table 2). We analyzed these responses and found that questions 22 and 23 resemble a normal distribution. This outcome was determined by considering the skewness

```

1 # chatGPT
2 import numpy as np
3
4 def erosion(f, b):
5     """Performs erosion of f with the structuring function b"""
6     f_len = len(f)
7     b_len = len(b)
8     result = np.full_like(f, np.inf) # Initialize result with infinity
9
10    # Perform erosion by computing the minimum value of f(y) - b(x - y)
11    for x in range(f_len):
12        min_val = np.inf
13        for y in range(max(0, x - b_len + 1), min(f_len, x + b_len)):
14            min_val = min(min_val, f[y] - b[x - y + b_len // 2])
15        result[x] = min_val
16
17    return result
18
19 def distance_transform(f, b):
20     """Performs the distance transform using successive erosions"""
21     g = f.copy() # Initialize g as the input f
22     i = 0
23
24     # Keep applying erosion until the result does not change
25     while True:
26         new_g = erosion(g, b)
27         if np.array_equal(new_g, g): # Stop if erosion does not change the result
28             break
29         g = new_g
30         i += 1
31
32     return g, i
33
34 # Example inputs
35 f = np.array([9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 0, 9, 9, 9, 9, 9, 9, 9, 9, 9, 0, 9, 9, 9, \
36             9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 0, 9, 9])
37 b = np.array([-2, -1, 0, -1, -2])
38
39 # Perform the distance transform
40 g, iterations = distance_transform(f, b)
41
42 # Output result
43 print("DT(f,b):", g)
44 print("Iterations:", iterations)

```

Code 16: Incorrect solution for Question 3 generated by ChatGPT.

or kurtosis z – values falling between -2.575 and 2.575 for 99% confidence level (CL) (Ryan, 2013). Therefore, we used a parametric method (t – test) for these two questions and an equivalent non-parametric method (*Wilcoxon Signed – Rank*) for the remaining. Additionally, we applied Cronbach’s Alpha to assess the internal consistency of the Likert data, and the obtained value was .90. Finally, we tested the hypothesis below.

H_0 : the morph.py library had a neutral effect on students’ learning, $mdn \leq 3$.

H_1 : the morph.py library positively affected students’ learning, $mdn > 3$.

Table 3 presents the results of assessing students’ perception of our library’s value in their DIP learning process. The findings demonstrate significant differences between all the questions

Table 2: Post-class Likert questions.

Id	Question
1	Does the <code>morph.py</code> library contribute to your DIP's learning process by allowing you to focus on the problem to be solved?
2	Did the <code>morph.py</code> library motivate you to learn DIP?
3	Have you used <code>morph.py</code> through all course tasks?
4	Do you suggest more implementations into <code>morph.py</code> ?
5	Do you think that <code>morph.py</code> was easier to use and learn than openCV?

Table 3: Wilcoxon Signed – Rank analysis for questions 21, 24 and 25 .

\mathcal{Q}	T	Median	$z - score$	Effect Size(ES)	Power(CL=99%)
21	120	5	$< .01$.93	.82
24	96	5	$< .01$.74	.61
25	118	5	$< .01$.87	.76

and the null hypothesis (H_0). For example, the students' positive perception of our library (21) indicates that the alternative hypothesis (H_1) holds statistical significance based on the one-group Wilcoxon Signed – Rank test, $N = 15$, $T=120$, $p < 0.01$, ES=large (.93). This result suggests that our library facilitated students to explore the different pre-implemented algorithms without implementing them from scratch. Consequently, they could focus more on understanding the underlying concepts and techniques behind the algorithms, thereby reducing the initial learning curve. This finding aligns with previous studies (Sage & Unser, 2003; Yaniv et al., 2018).

The results of a one-group t-test, measuring students' utilization of our library throughout the course, are presented in Table 4. Notably, the students' motivation scores (22) indicate that the alternative hypothesis (H_1) holds statistical significance, with a median score of 5, $t(14) = 7.67$, $p < 0.01$, ES=large (1.92). This result aligns with the findings about 21 and 23, indicating that the availability of high-level functions provided by `morph.py` significantly simplified the implementation of complex image processing operations. Consequently, students could concentrate more on the practical aspects of image processing, as was also suggested by (Rowe et al., 2018; Sage & Unser, 2003).

Finally, out of the 25 students who completed the course, only four did not achieve approval, resulting in an approval rate of 84%. We conducted this work's experiment at the beginning of 2023. Our institution does not have precise data on the pass rates of students who attended the course, excluding early dropouts, as shown below, which makes it challenging to compare this cohort with other classes.

Threats to validity

Our study faces challenges regarding its validity. Given that DIP is an elective course in our university, primarily selected by students nearing graduation, our results are inconclusive, highlighting the need for further experimentation. Many students enroll in multiple classes, often leaning toward those requiring less effort, contributing to a lower course completion rate.

Table 4: *t* – test analysis for all questions for questions 22 and 23 .

<i>Q</i>	<i>t</i>	<i>df</i>	<i>p</i> – <i>value</i>	Std. Deviation
	Mean	Median	Effect Size (ES)	Power(CL=99%)
22	7.67	14	< .01	.81
	4.56	5	1.92	.99
23	10.48	14	< .01	.61
	4.62	5	2.62	1

Moreover, conducting experiments involving both test and control groups necessitates approval from the ethics committee. Additionally, it requires the establishment of multiple classes, all taught by the same professor, with consistent learning conditions, scheduling within the same academic term, and a commitment to ensuring comparable levels of dedication. This presents logistical challenges at our university, where only one class is typically offered annually. Since our university's founding in 2006, the DIP has been presented in only ten classes, mainly after 2011.

Consequently, this article thoroughly presents our teaching, learning, and evaluation methods, laying the groundwork for future research to explore new avenues for experimentation.

8 Conclusions and Future Works

This paper presents an interactive course instructing novice learners in Digital Image Processing (DIP) using Python's `morph.py` library. The course is delivered through Google Colab and VPL activities on Moodle, offering practical examples and exercises to reinforce key concepts and operators. It covers essential topics, including image representation, sampling, quantization, and perception, as well as advanced subjects like transformations, enhancements, restoration, segmentation, feature extraction, object detection, and machine learning for computer vision.

The course has successfully addressed the challenges of instructing beginners in DIP. Students have consistently appreciated the course's practical and interactive nature. Students have successfully bridged the gap between theoretical knowledge with practical application by utilizing the `morph.py` library and engaging in hands-on exercises, leading to a deeper and more comprehensive understanding of DIP principles.

In the future, the course has the potential to expand into more advanced topics in DIP, equipping students with a broader skill set by incorporating more libraries and frameworks. Using real-life case studies and projects could further enrich the learning experience, enhancing students' problem-solving abilities and comprehension of practical DIP applications.

Data Availability Statement

The data supporting the findings of this study are available at: github.com/fzampirolli/morph, and are open-source under the MIT License.

References

- Banon, G. J. F., & Barrera, J. (1994). *Bases da morfologia matemática para a análise de imagens binárias*. UFPE-DI. [GS Search].
- Cuisenaire, O. (1999). *Distance transformations: Fast algorithms and applications to medical image processing* (tech. rep.). Université catholique de Louvain (UCL), Louvain-la-Neuve, Bélgica. [GS Search].
- Dougherty, E. R., & Lotufo, R. A. (2003). *Hands-on morphological image processing* (Vol. 59). SPIE press. <https://doi.org/10.1117/3.501104> [GS Search].
- García, G. B., & Suárez, O. D. (2015). *Learning image processing with opencv*. Packt Publishing. [GS Search].
- Giannakos, M., Azevedo, R., Brusilovsky, P., Cukurova, M., Dimitriadis, Y., Hernandez-Leo, D., Järvelä, S., Mavrikis, M., & Rienties, B. (2024). The promise and challenges of generative ai in education. *Behaviour & Information Technology*, 1–27. [GS Search].
- Gonzalez, R. C., & Woods, R. C. (2009). *Processamento digital de imagens* (3rd). Pearson Educação. [GS Search].
- Huang, C., & Mitchell, O. (1994). A euclidean distance transform using grayscale morphology decomposition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(4), 443–448. <https://doi.org/10.1109/34.277600> [GS Search].
- Hussain, M. (2023). Yolo-v1 to yolo-v8, the rise of yolo and its complementary nature toward digital manufacturing and industrial defect detection. *Machines*, 11(7), 677. <https://doi.org/10.3390/machines11070677> [GS Search].
- Jocher, G., Chaurasia, A., & Qiu, J. (2023). Ultralytics yolov8. <https://github.com/ultralytics/ultralytics> [GS Search].
- Knuth, D. E. (1984). Literate programming. *The Computer Journal*, 27(2), 97–111. <https://doi.org/10.1093/comjnl/27.2.97> [GS Search].
- Konstantinides, K., & Rasure, J. R. (1994). The khoros software development environment for image and signal processing. *IEEE Transactions on Image Processing*, 3(3), 243–252. <https://doi.org/10.1109/83.287018> [GS Search].
- Lantuejoul, C., & Beucher, S. (1981). On the use of the geodesic metric in image analysis. *Journal of Microscopy*, 121(1), 39–49. <https://doi.org/10.1111/j.1365-2818.1981.tb01197.x> [GS Search].
- Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., & Zitnick, C. L. (2014). Microsoft COCO: Common objects in context. *European Conference on Computer Vision*, 740–755. https://doi.org/10.1007/978-3-319-10602-1_48 [GS Search].
- Lopes, R. d. A., Kitani, E., Zampirolli, F. A., Yoshioka, L., Celiberto, L. A., & Ibusuki, U. (2023). Revisão sistemática de visão computacional para veículos autônomos agrícolas. *2023 15th IEEE International Conference on Industry Applications (INDUSCON)*, 485–492. <https://doi.org/10.1109/INDUSCON58041.2023.10374618> [GS Search].
- López, A. F. J., Pelayo, M. C. P., & Forero, Á. R. (2016). Teaching image processing in engineering using python. *IEEE Revista Iberoamericana de Tecnologías del Aprendizaje*, 11(3), 129–136. <https://doi.org/10.1109/RITA.2016.2589479> [GS Search].
- Lotufo, R. A. (2017). Set of functions used in the course IA898 - Digital Image Processing. <https://github.com/robertoalotufo/ia898> [GS Search].

- Lotufo, R. A. (2019). Set of functions used in the course IA870 - Mathematical Morphology. <https://github.com/robertoalotufo/ia870p3> [GS Search].
- Lotufo, R. A., Zampirolli, F. A., Junior, R. H., & Barrera, J. (1997). MMachLib functions and MMach operators. *Proceedings of the Brazilian Workshop'97 on Mathematical Morphology*. [GS Search].
- Machado, R. C., Lotufo, R. A., Silva, A. G., & Saúde, A. V. (2003). Adesso: Scientific software development environment. *Journal of Computer Science and Technology*, 3(1), 1–6. <https://journal.info.unlp.edu.ar/JCST/article/view/944> [GS Search].
- Machado, R. C., Rittner, L., & Lotufo, R. A. (2011). Adessowiki: Collaborative platform for writing executable papers. *Procedia Computer Science*, 4, 759–767. <https://doi.org/10.1016/j.procs.2011.04.080> [GS Search].
- MathWorks. (2024). *Mathworks image processing toolbox* [Acessado em 8 de janeiro de 2024]. <https://www.mathworks.com/products/image.html> [GS Search].
- Menegola, A., Fornaciali, M., Pires, R., Bittencourt, F. V., Avila, S., & Valle, E. (2017). Knowledge transfer for melanoma screening with deep learning. *2017 IEEE 14th International Symposium on Biomedical Imaging (ISBI 2017)*, 297–300. <https://doi.org/10.1109/ISBI.2017.7950523> [GS Search].
- Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1), 62–66. <https://doi.org/10.1109/TSMC.1979.4310076> [GS Search].
- Ritter, G., & Sussner, P. (1996). An introduction to morphological neural networks. *Proceedings of 13th International Conference on Pattern Recognition*, 4, 709–717 vol.4. <https://doi.org/10.1109/ICPR.1996.547657> [GS Search].
- Rodríguez del Pino, J. C., Rubio Royo, E., & Hernández Figueroa, Z. J. (2010). VPL: Laboratorio virtual de programación para moodle. *Jornadas de Enseñanza Universitaria de la Informática*, 429–435. <http://hdl.handle.net/10045/127218> [GS Search].
- Rowe, P. M., Cheng, H., Fortmann, L., Wright, A., & Neshyba, S. (2018). Teaching image processing in an upper level CS undergraduate class using computational guided inquiry and polar data. *Journal of Computing Sciences in Colleges*, 34(1), 171–179. <https://dl.acm.org/doi/10.5555/3280489.3280517> [GS Search].
- Ryan, T. P. (2013). *Sample size determination and power*. John Wiley & Sons. <https://doi.org/10.1002/9781118439241> [GS Search].
- Sage, D., & Unser, M. (2001). Easy java programming for teaching image-processing. *Proceedings of the 2001 International Conference on Image Processing (Cat. No. 01CH37205)*, 3, 298–301. <https://doi.org/10.1109/ICIP.2001.958110> [GS Search].
- Sage, D., & Unser, M. (2003). Teaching image-processing programming in java. *IEEE Signal Processing Magazine*, 20(6), 43–52. <https://doi.org/10.1109/MSP.2003.1253553> [GS Search].
- Silva, A. G., Lotufo, R. A., Machado, R. C., & Saude, A. V. (2003). Toolbox of image processing using the python language. *2003 International Conference on Image Processing (Cat. No.03CH37429)*, 3, III–1049. <https://doi.org/10.1109/ICIP.2003.1247428> [GS Search].
- Singh, S., Lloyd, J., & Flicker, F. (2024). Hamiltonian cycles on ammann-beenker tilings. *Physical Review X*, 14(3), 031005. <https://doi.org/10.1103/PhysRevX.14.031005> [GS Search].

- Yahya, A. A. (2019). Teaching digital image processing topics via matlab techniques. *International Journal of Information and Education Technology*, 9(10), 729–734. <https://doi.org/10.18178/ijiet.2019.9.10.1294> [GS Search].
- Yaniv, Z., Lowekamp, B. C., Johnson, H. J., & Beare, R. (2018). Simpleitk image-analysis notebooks: A collaborative environment for education and reproducible research. *Journal of Digital Imaging*, 31(3), 290–303. <https://doi.org/10.1007/s10278-017-0037-8> [GS Search].
- Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). How transferable are features in deep neural networks? *Advances in Neural Information Processing Systems (NeurIPS 2014)*, 3320–3328. <https://proceedings.neurips.cc/paper/5347-how-transferable-are-features-in-deep-neural-networks.pdf> [GS Search].
- Zampirolli, F. A. (2023). *MCTest: Como criar e corrigir exames parametrizados automaticamente* [1a Edição]. Edição do Autor. [GS Search].
- Zampirolli, F. A., Borovina Josko, J. M., Teubl, F., & Kurashima, C. S. (2024). A practical digital image processing course with morph.py. *Anais do IV Simpósio Brasileiro de Educação em Computação (EduComp)*, 304–313. <https://doi.org/10.5753/educomp.2024.237274> [GS Search].
- Zampirolli, F. A., & Filipe, L. (2017). A fast CUDA-based implementation for the euclidean distance transform. *Proceedings of the International Conference on High Performance Computing & Simulation (HPCS)*, 815–818. <https://doi.org/10.1109/HPCS.2017.123> [GS Search].
- Zampirolli, F. A., Pisani, P. H., Borovina Josko, J. M., Kobayashi, G., Fraga, F., Goya, D., & Savegnago, H. R. (2020). Parameterized and automated assessment on an introductory programming course. *Anais do XXXI Simpósio Brasileiro de Informática na Educação (SBIE)*, 1573–1582. <https://sol.sbc.org.br/index.php/sbie/article/view/12913> [GS Search].
- Zampirolli, F. A., Sato, C. M., Savegnago, H. R., Batista, V. R., & Kobayashi, G. (2021). Automated assessment of parametric programming in a large-scale course. *2021 XVI Latin American Conference on Learning Technologies (LACLO)*, 357–363. <https://doi.org/10.1109/LACLO54177.2021.00079> [GS Search].
- Zampirolli, F. A., Teubl, F., & Batista, V. R. (2019). Online generator and corrector of parametric questions in hard copy useful for the elaboration of thousands of individualized exams. *Proceedings of the 11th International Conference on Computer Supported Education (CSEDU)*, 352–359. <https://doi.org/10.5220/0007672603520359> [GS Search].

Appendix 1 – About MCTest

The MCTest has been published in previous papers and organized in the book by Zampirolli (2023). This appendix summarizes the steps to create exams with automatic correction in the VPL plugin on Moodle, starting with installing MCTest and creating exams and parametric questions. All content in this paper relates to the second edition of the book, which is still under construction and available at github.com/fzampirolli/mctest/tree/master/book/2ed-br. The version of MCTest used in this paper is 5.3. Both editions share the same chapters, but the second edition includes new sections and appendices not present in the first edition. These additions are detailed in the preface of the second edition.

How to Install MCTest

To install MCTest, download and install VirtualBox, then set up Ubuntu 22.04 within it. Open the Ubuntu terminal and execute the following commands as root:

```
wget https://raw.githubusercontent.com/fzampirolli/mctest/master/_setup-all.sh
```

Replace `yourLogin` with your username and run the following lines. After a few minutes, MCTest will be configured.

```
sed -i 's/\/home\/fz\/\\\/home\/yourLogin\/\\g' _setup-all.sh
source _setup-all.sh
pip install mysqlclient
```

To run MCTest, use:

```
source /home/yourLogin/PycharmProjects/runDjango.sh
```

Access it via `http://127.0.0.1:8000` in your browser. For further details and configuration options, visit the book's website above. Both the book and the software MCTest are available for free at github.com/fzampirolli/mctest.

Creating an Exam on MCTest

The book by Zampirolli (2023) details the entire process of creating and grading assessments (Exams) with multiple-choice or essay questions. These essay questions can also be configured so students can submit programming exercises in VPL activities on Moodle. In MCTest, all assessments are configured on an Exam screen containing Classes, Topics (part of a Discipline, e.g., DIP), and Questions. Each Question is in a unique Topic. Each Discipline (subject) belongs to one or more Courses (for example, Bachelor's in Computer Science and Computer Engineering). In addition to these MySQL Database entities, an Exam has several attributes that need to be configured, such as the number of questions and variations. For example, each exercise list has four questions and 50 variations. It is possible to choose about 10 questions, and four questions are randomly selected for each variation. Creating variations occurs by pressing the "Create-Variations" button on the Exam screen, with the "Json" box selected. If the questions were created as VPL activities, the teacher would receive an email with a `*linker.json` file containing all the test cases of the 50 variations. Finally, by clicking the "Create-PDF" button, MCTest will also send the teacher an email with a PDF file per class, containing the randomly selected exams for each student, as well as a `*students_variations.csv` file containing the name, email, and selected variation for each student. These two buttons are located in the first part of the Exam screen, shown in Figure 11. The other parts of this screen are detailed in the book by Zampirolli (2023, Chapter 6).

Creating a Parametric Question on MCTest

This section summarizes Zampirolli (2023, Section 5.4.3), explaining how a parametric question used in List 1 was created in MCTest. Figure 12 shows the initial part for creating a parametric question with VPL integration. First, the question belongs to the DIP discipline's topic `im1-Introduction`. Next, a short description `list1-chess` is defined. The most important part is the "Description", which contains the question content in \LaTeX and Python. This section includes random variables between `[[code: and]]`. In this example, the variables are `height` for the height of an image, `caso0_inp`, and `caso0_out` for input and output examples of the question. The content between `comment` indicates that the question integrates with

Exam Update

Attention: Each time you click Create-Variations, a different exam will be generated! After printing the PDF with the exam, SAVE IT TO YOUR COMPUTER and DO NOT CHANGE the attributes of this page; Otherwise automatic correction will not be possible on scanned exams in Upload-PDF.

Create-PDF

video

Escolher arquivo

Nenhum arquivo escolhido

Upload-PDF

Escolher arquivo

Nenhum arquivo escolhido

Send-Feedback-Students Text

Adjust PDF font size: 11pt

Adaptive Exam

Type of statistic:

WPC

MLE

SAT

Number of Exams with QM: ∞

Choose PDF file for correction of exams

☐ If you choose to provide feedback to the students, also send the questions and the answer keys.

Caution: (1) Before scanning, make sure all circles have been filled correctly. If you use correction fluid and erase part of the outline of the circle, the correction may not work. (2) Scan with a resolution of 150dpi (if you can not decode QRCode, use 200dpi), **gray levels**, just the front of the sheet and one PDF per class. (3) The 4 black disks can not be defective. (4) If you chose only answers on the screen of this exam, the first page of the PDF should contain the template and all questions will be disregarded. (5) If in this Exam screen the option to Return to Students was chosen, when enrolling the students in the class, you must also include the student's e-mail. If this has been done, you can follow the step to the right side Send-Return-Students, for each student receive the correction of your examination by email >>>

Caution: (1) If you used exams with written questions and in non-ecological format, then one question was generated per sheet, right? If you want to submit manually corrections to each student's in PDFs, you will need to follow a few steps: (1) in the previous column step, when uploading all exams in PDF, a zip file with a folder for each question was generated (for example, Download.zip). Within this folder a pdf was generated for each student in the format _e1_c2_q3_p001_5.pdf, where 1 is the exam ID, 2 is the classroom ID, 3 is the question ID, 001 is the pdf page, and 5 is the student ID. The teacher can correct by making annotations in the PDF itself OR also can scan with annotations made in the pen. (2) manually change the file names of each question in the format _A;e1,e2,...;_e1_c2_q3_p001_5.pdf, where A is a concept or a note and ei are error code numbers (optional); in the question folder, it should have a _e1_c2_q3.txt file (**one for each classroom**), with a message to be sent to each student. (3) compress the folder of each question for all students (for example _e1_q3.zip) and press the button above. Each student will receive the PDF with the corrections and will also return a CSV file with the concepts of each student.

Send-Feedback-Students

This option is to send feedback ONLY from multiple choice exams after successfully completing the step from the previous column.

Before creating the exams in the button above, first create the variations. It is necessary to create new variations of the exam each time you change the questions and the number of variations in the attributes below. The options marked below will be sent to your email.

Create-Variations

ID: 73213

☐ Json

☐ Template

☐ Aiken

☐ XML

☐ LaTeX+PDF

Name

DIP-list1

Figure 11: First part of the Exam screen in MCTest.

VPL in Moodle and contains test cases in the variable `moodle_cases`. These variables are defined in the Python code block between `[[def: and]]` and are detailed in Codes 17 and 18.

Code 17 begins by importing the `json` and `numpy` libraries. The `height` parameter is set to a random integer between 5 and 15 to ensure each student receives a random question. The `chess` method creates a 2D NumPy array of size $h \times w$ representing a chessboard, where h and w are the number of rows and columns. The array is filled with zeros and ones to alternate between black and white squares.

```

1 [[def:
2 import json
3 import numpy as np
4
5 # PARAMETERS USED IN QUESTION DESCRIPTION
6 #>>> BEGIN
7 # use random to have a different question for each student
8 height = np.random.randint(5,15)
9 #<<< END
10
11 def chess(h,w):
12     import numpy as np
13     m = np.zeros((h,w), dtype='int')
14     for i in range(h):
15         for j in range(w):
16             if (i+j)%2:
17                 m[i][j]=1
18     return m

```

Code 17: Second part for creating a question in MCTest.

Code 18 continues from the previous code by initializing empty lists `inp_list` and `out_list`, and setting the variable `test_cases` to 4, which represents the number of test

Question Update

Create-PDF	Compile-Colab	Save-Json
----------------------------	-------------------------------	---------------------------

See this question in PDF format [Copy-Paste the description of question for test in Colab Google](#) [It will save all your questions to a file in json format](#)

Choose Topic [PDI]<im1-Introduction> ▼

Short Description list1-chess

Group Only one question per group will be sorted for each exam

Description

```

Create a method to generate an image (matrix) of dimensions \(\ H \times W \) with values 0 (black) and 1 (white), resembling a chessboard pattern. Consider \(\ H = \) [[code:height]] \) and the dimension \(\ W \) should be read as an integer variable. See example:

\textbf{Input Example;}
\begin{verbatim}
[[code:caso0_inp]]
\end{verbatim}

\textbf{Output Example;}
\begin{verbatim}
[[code:caso0_out]]
\end{verbatim}

% needed to generate test cases in moodle
\begin{comment}
[[code:moodle_cases]]
\end{comment}

[[def:

```

Figure 12: First part of the Question screen in MCTest.

cases. For each test case, it generates a random `width` value between 5 and 15 and uses this value to create a chessboard of dimensions `height` \times `width`. The `height` value is generated in Code 17. The chessboard pattern is converted to a string format using the `drawImage` method, see Code 19. The `inp` string (representing the width) and the `out` string (representing the chessboard image) are appended to their respective lists. After processing all test cases, the code creates a dictionary `cases` containing the input and output lists, convert it to JSON format, and assigns it to `moodle_cases`. Additionally, it extracts the first test case input and output for use as examples in the question description.

In line 8 of Code 18, the `mm.drawImage` method of `morph.py` is called, which contains the method shown in Code 19. This code converts an input image, a numpy array, into a string suitable for printing. It calculates the digit format based on the minimum and maximum values of the image. If the image contains negative values, it ensures the format accounts for the sign. It then iterates through each array element, constructing a string representation row by row. Finally, it returns this string representing the image.

Figure 13 shows an output (variation) of the question to draw a chessboard after clicking the “Create-PDF” button in Figure 12. Each time this button is pressed, a new question variation is generated. The initial part of the figure presents attributes of the question, and the green part #1653 represents the question number in the database. Thus, all questions applied in the course, in lists and exams, follow the same process of creating parametric questions.

```

1 inp_list, out_list, test_cases = [], [], 4 # n. test cases - CHANGE
2 for i in range(test_cases):
3
4     #>>> BEGIN THE SOLUTION OF THE QUESTION
5     # In this example, the value must be read as an entry to be assessed in Moodle
6     width = int(np.random.randint(500,1500)/100) # use random to have different test cases
7     inp = str(width)+'\n'
8     out = mm.drawImage(chess(height, width))
9     #<<<< END OF QUESTION SOLUTION
10
11 # list of test cases
12 inp_list.append(inp)
13 out_list.append(out)
14
15 cases = {}
16 cases['input'] = np.array(inp_list).tolist()
17 cases['output'] = np.array(out_list).tolist()
18 moodle_cases = json.dumps(cases)
19
20 # to show an example in the question description
21 caso0_inp = cases['input'][0]
22 caso0_out = cases['output'][0]
23 ]]
```

Code 18: Third part for creating a question in MCTest.

```

1 def drawImage(f):
2     """ Converts the input image f into a string representation suitable for printing.
3     Args: f (ndarray): The input image.
4     Returns: A string representing the input image.
5     Example:
6         string_representation = mm.drawImage(f)
7         print(string_representation)
8     """
9     h, w = f.shape
10    if np.min(f) < 0:
11        digits = '%' + str(1 + len(str(np.max(f)))) + 'd '
12    else:
13        digits = '%' + str(len(str(np.max(f)))) + 'd '
14    # print('"' + digits + '"')
15    string_representation = ''
16    for i in range(h):
17        for j in range(w):
18            string_representation += digits % f[i][j]
19        string_representation += '\n'
20    return string_representation
```

Code 19: Method drawImage of morph.py.

Appendix 2 – About Moodle

This section provides an overview of configuring a VPL activity on Moodle, specifically using version 4.1.

Configuring a VPL Activity on Moodle

As presented in the previous appendix, creating an exam in MCTest with integration into VPL activities in Moodle generates files in JSON and CSV formats, which should be renamed to `linker.json` and `students_variations.csv` respectively. These two files, together with `morph.py`, should be placed in the “Execution Files” of the VPL activity, along with all

MCTest

Topic: im1-Introduction
Group:
Short Description: list1-chess
Type: QT
Difficulty: 1
Bloom taxonomy: remember
Last update: 2021-01-07
Who created: fzampirolli@ufabc.edu.br
Parametric: YES
Integration: Moodle+VPL

#1653 1. Create a method to generate an image (matrix) of dimensions $H \times W$ with values 0 (black) and 1 (white), resembling a chessboard pattern. Consider $H = 6$ and the dimension W should be read as an integer variable. See example:

Input Example:

14

Output Example:

```

0 1 0 1 0 1 0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0 1 0 1 0 1 0

```

Figure 13: A variation of the question after pressing the “Create-PDF” button in Figure 12.

the files available in the folder `VPL_modification/V13-Moodle_v4.1` from github.com/fzampirolli/mctest. All these files should be marked as “Files to keep when running”. The book by Zampirolli (2023, Chapter 10) details this entire process.

Figure 14 shows a screenshot of the Moodle interface for the VPL activity List1, highlighting the third question (`Q3.py`) on the left side. On the right side, it shows the feedback from the corrections, indicating success in correcting the four questions. It is necessary to import the `morph.py` library on line 5 to call the `drawImage` method on line 20.

The screenshot displays the Moodle VPL activity interface. At the top, there are tabs for 'Description', 'Submissions list', 'Similarity', and 'Test activity'. Below these, there are buttons for 'Submission', 'Edit', 'Submission view', 'Grade', and 'Previous submissions list'. The main area is divided into two panels. The left panel shows a code editor with a Python script for generating a chessboard pattern. The script defines a function `chess(h,w)` that takes dimensions `h` and `w` as input and returns a matrix `m` of size `h` by `w`. The matrix is filled with 0s and 1s in a checkerboard pattern. The script then reads an input value for `w` and calls the `drawImage(m)` method from the `mm` module. The right panel shows the feedback for the submission. It indicates a 'Proposed grade: 100 / 100' and 'Avaliação: 1.000/1 (100.00%)'. It also shows a 'Summary of tests' section with the message '4 tests run/ 4 tests passed!'. The bottom right corner of the interface features a green circular button with a white upward-pointing arrow.

Figure 14: Screenshot of a VPL activity in Moodle showing a solution for Figure 13.

Appendix 3 – About Safe Exam Browser

To restrict access to a VPL activity in Moodle, SEB (Safe Exam Browser) was used. The SEB configuration file is available for Windows. Windows 10 was used, and the SEB version used in this paper was 3.7.1.704 (safeexambrowser.org). This version must be installed on all machines in the laboratory. After generating the SEB configuration file and configuring the VPL activity as described, double-click this file, and SEB will automatically run, directing to the specific VPL activity.

Configuring a SEB Activity

In the “General” tab, include the URL of the Moodle VPL activity in the “Start URL” field. In the “Network” tab, include an alternative URL for the activity, as shown in Figure 15. Pay close attention to the *filter expression* at the end of the URL /vpl/*?id=12481*, where this number is the key for the VPL activity in Moodle. In the “Exam” tab, check “Use Browser Exam Key and Configuration Key” and then copy the key in “Browser Exam Key”. In the “Config File” tab, click “Save Settings As...” to save a file in SEB format ¹. The *filter expression* was created by Prof. Paulo Henrique Pisani, from UFABC, to whom the authors are grateful.

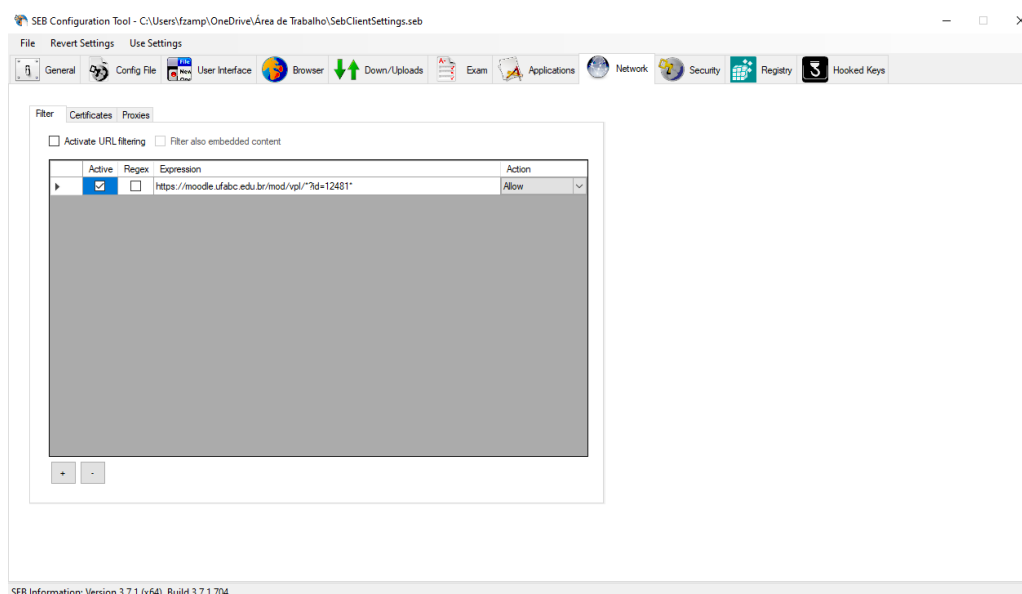


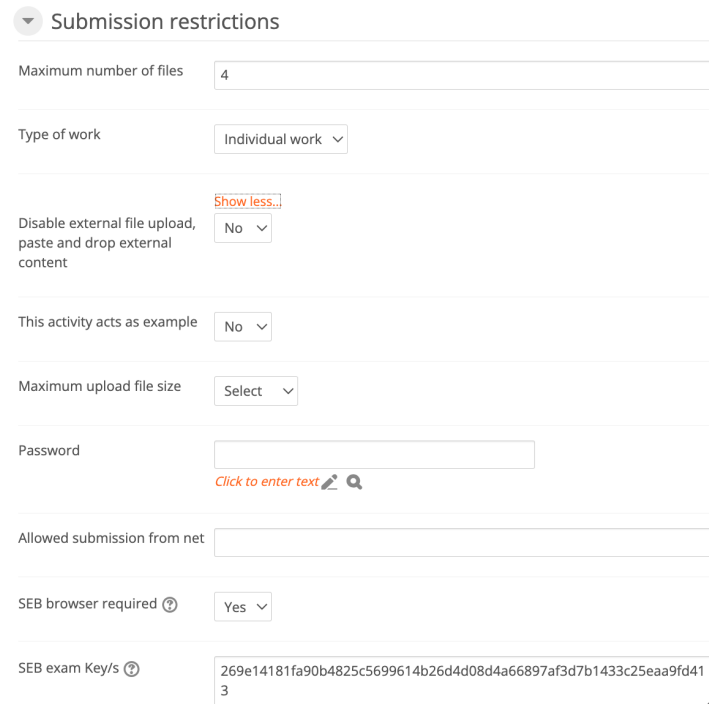
Figure 15: Screenshot of SEB showing the “Network” tab with access restrictions.

This key should be copied to the Moodle VPL activity. In the gear icon, go to “Configuration”, then “Submission restrictions”, “Show more”, set “SEB browser required” to “YES”, and paste the key in “SEB exam Key/s”, as shown in Figure 16.

If the lab’s IPs are in a consecutive range, it is possible to restrict access to the VPL activity to only the lab machines by including the range in the “Allowed submission from net” field (see Figure 16). For example, the range 111.11.11.1–62 allows all IPs between 111.11.11.1 and 111.11.11.62. Otherwise, it is necessary to include each IP separated by a comma ².

¹For more details about SEB, see https://safeexambrowser.org/windows/win_usermanual_en.html.

²For more details about the configuration of IPs on VPL activity, see the documentation at vpl.dis.ulpgc.es.



Submission restrictions

Maximum number of files: 4

Type of work: Individual work

Disable external file upload, paste and drop external content: No

This activity acts as example: No

Maximum upload file size: Select

Password: [Click to enter text](#)

Allowed submission from net:

SEB browser required: Yes

SEB exam Key/s: 269e14181fa90b4825c5699614b26d4d08d4a66897af3d7b1433c25eaa9fd413

Figure 16: Screenshot of a VPL activity in Moodle showing the key of SEB.

Appendix 4 – About using YOLO by Transfer Learning

This appendix explains how to create a dataset using Roboflow (roboflow.com), a productive environment for object labeling. The reader can compare the geometric object detection process presented in Section 6.2 with the one presented here. Many recent works have used neural networks for object detection/segmentation. For example, Lopes et al. (2023) presents a systematic review and finds that state of the art computer vision applications in autonomous vehicles highlight that the most recent works use convolutional neural networks, including YOLO. It is expected that reading this appendix will enable the reader to create similar datasets to detect other objects not covered by pre-trained networks like YOLO.

Using Roboflow to Detect Geometric Objects with Transfer Learning

This text will show how to adapt a pre-trained YOLOv8 network (Jocher et al., 2023) to detect and classify the following geometric shapes: circle, cross, ellipse, heptagon, hexagon, pentagon, square, star, and triangle. It will detail creating and labeling the image dataset and training using the Ultralytics library. Additionally, it will include the use of an algorithm for inference/detection.

In the context of this problem, brief explanations about artificial neural networks, convolutional neural networks, and transfer learning are presented below. Artificial Neural Networks (ANNs) are computational models inspired by the human brain's neural networks. They consist of interconnected nodes or "neurons" that process data in layers, enabling pattern recognition. Convolutional Neural Networks (CNNs) are a specialized type of ANN designed to process structured grid data, such as images and the neighborhoods of their pixels. They use convolutional

layers to automatically and adaptively learn spatial hierarchies of features from input images. In other words, they learn, for example, the weights used in the neighborhoods of pixels, which is the structuring function used in this work. Transfer Learning is a machine learning technique where a pre-trained model is used as a starting point for a different but related task, significantly reducing the time and resources required for training. To reduce the time needed to train large deep learning models from scratch and overcome the scarcity of large datasets to train these models, one solution was to leverage knowledge from models previously trained with large volumes of data (Menegola et al., 2017). The knowledge transfer process, known as Transfer Learning, is widely used in image classification. It takes advantage of feature vectors from a previously trained deep neural network, with its synaptic weights already configured to recognize and extract features from a specific data set. These vectors become the descriptors for each image in the new database. Once these feature vectors are generated for the new set to be classified, they are used as input to a new classifier. This new classifier can be trained with these vectors or used as input to the fully connected classification layer of the architecture used for feature extraction (Yosinski et al., 2014).

To create a database of geometric shapes on [Roboflow](#), start by creating an account or logging in on the website. The next step is to create a new object detection project and define the classes of interest: “circle”, “cross”, “ellipse”, “heptagon”, “hexagon”, “pentagon”, “square”, “star”, and “triangle”. Next, you must upload the corresponding images of the geometric shapes created and store them locally on your computer. After the upload, begin manual labeling by drawing bounding boxes around the geometric shapes and assigning the correct class. If available, use automatic labeling tools to speed up the process.

During preprocessing, Roboflow provides several options to enhance and augment your dataset. You can resize images, normalize pixel values, and apply augmentations such as rotation, flipping, and color adjustments to increase the diversity of the training data. These steps help improve the robustness and generalization ability of the model. In this specific case, data augmentation techniques such as rotations, flips, brightness, and saturation adjustments were employed. To train the model, a dataset of 96 images was labeled with bounding box annotations, each depicting multiple geometric objects. Figure 17 shows the number of geometric objects labeled by class in the dataset during training, along with the data augmentation techniques.

After labeling and pre-processing are completed and the database is generated, the next step is to go to the “Export” section and select the “YOLOv8” format for export. By clicking ‘Download dataset’, you can get the labeled files in YOLOv8 format. The folder structure for training should be organized as follows: a root folder containing train, valid, and test subfolders for images and corresponding labels subfolders for annotations. This hierarchy ensures that the YOLOv8 model can properly locate and use the training, validation, and test images and their annotations. The created image base is publicly available at this URL:

<https://universe.roboflow.com/visao/geometric-shapes-5hwwu>

The necessary code for training is generated automatically, adapted for implementation in PyTorch ([pytorch.org](#)), including instructions for downloading the processed data and configuring the YOLOv8 model. The model’s specific configuration, such as image size and number of classes, is also done on the platform. Training can be done on-site for a fee. After creating the dataset using the site, download the dataset, which contains the folders and configuration files. To carry out weight training on the YoloV8 network, the Ultralytics library must be installed. The following

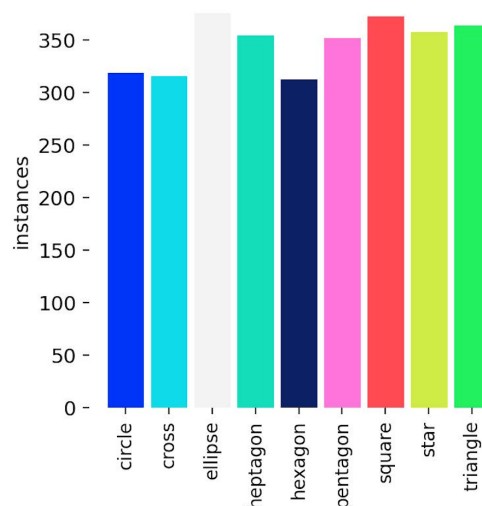


Figure 17: Number of geometric objects labeled by class used during training.

command accomplishes this task: `pip install ultralytics`.

A YAML configuration file (e.g., “data.yaml”) is generated in the process (YAML is a human-readable data serialization format often used for configuration files). This file contains the folder paths for the images and labels, `nc` for several classes, and `names` for the names of classes. For this example, the following file was created:

```
train: ../train/images
val:   ../valid/images
test:  ../test/images
nc: 9
names: ['circle', 'cross', 'ellipse', 'heptagon', 'hexagon', 'pentagon', 'square', 'star', 'triangle']
```

After installing the Ultralytics library and creating the folder structure downloaded from the Roboflow website, the next step is to perform the training by executing the appropriate command. Below is an example of the command used to run the training:

```
yolo task=detect mode=train model=yolov8n.pt imgsz=640 data=data.yaml epochs=320 plots=True
```

Where the `task` parameter describes the type of task to be performed on the dataset, which can be `detect`, `segment`, or `classify`. `mode` refers to the mode in which the program is operating. In this context, it specifies whether the model is in training mode (`train`), validation (`val`), or prediction (`predict`). `model` specifies the model’s architecture to be used. In this case, it mentions the “YOLOv8 Nano” (`yolov8n.pt`) model, pre-trained on the COCO dataset (Lin et al., 2014). The image size is defined next, where the default value is 640×640 pixels. `data` indicates the dataset’s YAML file path. The number of epochs is defined in the `epochs` parameter. An epoch is a complete pass through the entire training dataset. `plots=True` indicates that statistics generated during training will be saved, along with the trained model in the `best.pt` file.

During training, YOLOv8 adjusts the neural network weights to optimize object detection in the custom dataset. Performance metrics, including detection precision and mean Intersection over Union (IoU), are generated to facilitate model evaluation and further optimization. After training, the model can be used for inference. The Colab notebook demonstrates how to perform inference with the trained YOLOv8 model, available at the following URL:

https://drive.google.com/drive/folders/19cGYCLPriqGqOU8tcsUPvGs77BVHJH_5

The notebook loads the trained YOLOv8 model to detect geometric objects in images, displaying the results with bounding boxes and class labels.

The notebook for training produced the following report, indicating excellent results, with recognition of almost all instances in seven validation images, as shown in the confusion matrix (Figure 18). Interestingly, the matrix reveals that one instance, a background, was incorrectly classified as a heptagon.

320 epochs completed in 43 minutes.

Optimizer stripped from runs/detect/train/weights/best.pt, 6.3MB

Ultralytics YOLOv8.2.98 | Python-3.10.12 torch-2.4.1+cu121 CUDA:0 (Tesla T4)

Summary: 168 layers, 3,007,403 parameters, 0 gradients, 8.1 GFLOPs

Class	Images	Instances	Box (P	R	mAP50	mAP50-95)
all	7	63	0.991	1	0.995	0.995
circle	7	7	0.992	1	0.995	0.995
cross	7	7	1	1	0.995	0.995
ellipse	7	7	0.991	1	0.995	0.995
heptagon	7	7	0.969	1	0.995	0.995
hexagon	7	7	0.996	1	0.995	0.995
pentagon	7	7	0.993	1	0.995	0.995
square	7	7	0.991	1	0.995	0.995
star	7	7	0.992	1	0.995	0.995
triangle	7	7	0.993	1	0.995	0.995

Speed: 0.2ms preprocess, 2.8ms inference, 0.0ms loss, 1.2ms postprocess

Learn more at <https://docs.ultralytics.com/modes/train>

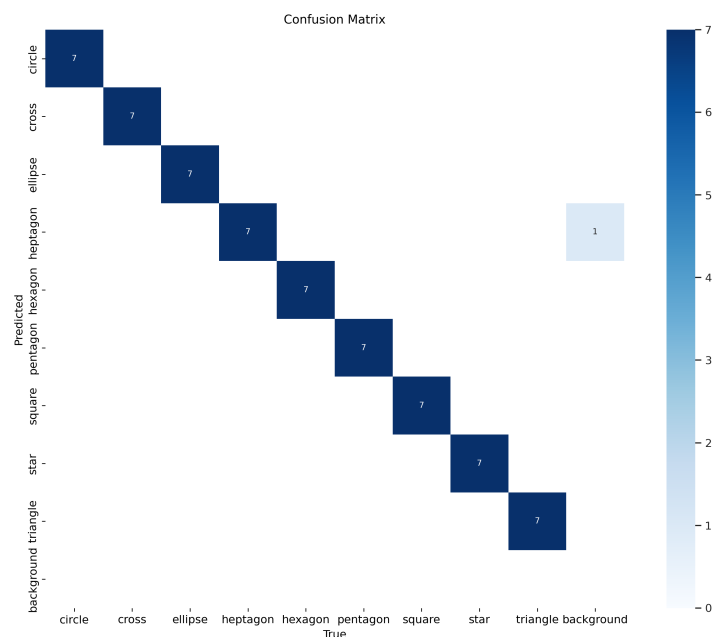


Figure 18: Confusion Matrix.

Finally, this appendix summarizes the state of the art in object detection using CNNs, as a proposal for future work in teaching image processing using this rapidly growing area of research over the last 10 years (Lopes et al., 2023). For example, despite discussions on replacing convolutional layers with morphological operators beginning with the work of Ritter and Sussner (1996), few applications utilize dilation or erosion layers, leaving room for significant scientific contributions.