AI & SUSTAINABILITY IN VET EDUCATION

Name of the Author/s: Sucessos Criativos, Lda

Title of WP: WP3 – AI-SAVES@ProjectHub

Activity Reference: 3.2 – Project template for AI-SAVES@ProjectHub

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Table of Contents

1. Project Title: 4

2. Project Category: 4

3. Project Description: 4

4. Keywords: 4

5. Target Audience: 4

6. Project Objectives: 5

7. Project Timeline: 6

8. Resources Provided: 6

9. Tools and Technologies Used: 6

10. Step-by-Step Implementation Plan: 6

Step 1: Import Libraries and Setup: 7

Step 2: Load and Display a Color Image: 7

Step 3:Convert Color Format (BGR to RGB) for Correct Display: 11

Step 4: Check Image Properties (Dimensions and Channels): 12

11. Evaluation Criteria: 17

12. Languages Available: 17

13. Contributors: 17

14. Feedback and Continuous Improvement: 18

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

1. Project Title:

Image Processing with OpenCV in Python

---

2. Project Category:

AI in Education

---

3. Project Description:

This micro-course introduces the fundamentals of digital image processing using OpenCV in Python. Learners will explore how computers store and manipulate images, starting from simple tasks like loading and displaying an image to more advanced operations such as shape detection. The project is designed as a hands-on tutorial: participants will write Python code to transform images and extract useful information from them. By the end of the course, learners will understand how to handle color images versus grayscale, apply geometric transformations, adjust image brightness/contrast, convert images to binary via thresholding, detect object contours, and create image masks. This practical approach aims to build skills in computer vision, preparing learners for more advanced AI and image analysis projects in a VET context.

---

4. Keywords:

Image Processing, OpenCV, Computer Vision, Python, Education

---

5. Target Audience:

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Educators, learners, and makers with or without prior experience who are beginning their journey in AI.

---

6. Project Objectives:

Understand Image Representation: Learn how images are stored as pixel arrays (grayscale vs. RGB color channels) and interpret image dimensions (width, height, channels).

Perform Basic Image Operations: Be able to load images from disk, display them with correct color rendering, and convert between color spaces (BGR to RGB, RGB to grayscale).

Apply Geometric Transformations: Practice resizing and cropping images, as well as rotating and flipping images to change orientation or viewpoint.

Adjust Image Brightness and Contrast: Modify pixel intensity values to brighten or darken images and understand how contrast scaling works.

Use Thresholding for Segmentation: Convert grayscale images to binary (black and white) images using threshold techniques to isolate features of interest.

Contour Detection and Shape Recognition: Detect object contours in binary images and determine basic shapes (e.g., distinguishing circles, rectangles, triangles) from those contours.

Implement Masking: Create and use binary masks to focus on specific regions of an image (e.g., isolate an object or filter out background) using bitwise operations.

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

7. Project Timeline:

The estimated duration is approximately **2.5 to 3 hours**.

8. Resources Provided:

- For these tasks you can use either a computer or your raspberry pi.

9. Tools and Technologies Used:

- **Jupyter Notebook (Colab):** The project uses a Jupyter Notebook *Image Processing Basics.ipynb* (run on Google Colab for ease of setup). This notebook contains step-by-step code and explanations, and can be saved to your Google Drive for editing.
- **OpenCV Library (cv2):** Open-source computer vision library used for image reading, processing, and transformations.
- **Python & NumPy:** Python is the programming language for the project, and NumPy is used for numerical operations on image arrays.
- **Matplotlib:** Used for displaying images within the notebook. Matplotlib expects images in RGB format for correct color display.

10. Step-by-Step Implementation Plan:

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Step 1: Import Libraries and Setup:

Begin by importing the necessary libraries. This includes OpenCV (cv2), NumPy (np), and Matplotlib for plotting. In a Colab environment, you should also mount your Google Drive to access image files (e.g., drive.mount('/content/drive')).

```
import cv2                  # OpenCV for image processing
import numpy as np          # NumPy for array operations
import matplotlib.pyplot as plt  # Matplotlib for displaying images
from google.colab import drive
drive.mount('/content/drive')   # (Colab only) Mount Google Drive to access data
```

Explanation: These imports set up the programming environment. Mounting the drive (if using Colab) gives access to the folders where our images are stored (e.g., TrainingGreeceCV/data/). After this step, we can read and write files as needed.

Step 2: Load and Display a Color Image:

Digital images, whether black-and-white or color, are represented as arrays of numerical values in a computer. Each of these values, known as pixel values, defines the intensity of a particular spot within the image.

2.1- Grayscale Images: A Simpler Representation

In a grayscale image, each pixel has a single intensity value, which describes the brightness of that pixel. These values range from 0 to 255:

• 0 represents black;

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

• 255 represents white;

• Values between 0 and 255 represent various shades of gray.

Consider a grayscale image of the number "8" as follows:



Image retrived from https://www.analyticsvidhya.com/blog/2021/03/grayscale-and-rgb-format-for-storing-images/

In this example, the image consists of small squares called pixels.

If the image has 24 pixels in height and 16 in width, we would say it has dimensions of 24 x 16.

When zoomed in, you can see these pixels as individual units, each contributing to the larger image.

Computers store grayscale images as a matrix where each element represents the intensity of a pixel, forming a 2D array.

This matrix is also called a channel because it holds intensity information across all pixels in the image.

Grayscale images are stored as a single 2D matrix of intensity values, where pixel values represent brightness on a scale from 0 (black) to 255 (white), and only one channel is used:

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

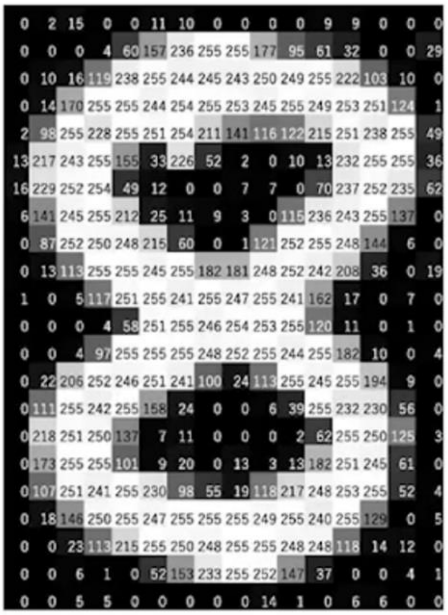


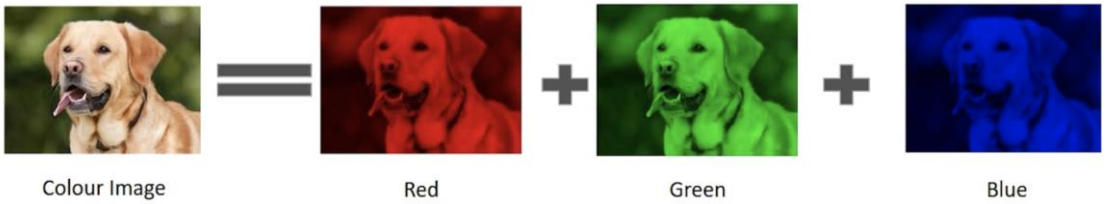
Image retrived from https://www.analyticsvidhya.com/blog/2021/03/grayscale-and-rgb-format-for-storing-images/

2.2 Colored Images: The Role of RGB Channels

In colored images, pixels represent not just brightness but also color information. Most digital images use the RGB color model, where colors are created through a combination of three primary color channels:

1. Red Channel: Contains intensity values for red shades
2. Green Channel: Contains intensity values for green shades
3. Blue Channel: Contains intensity values for blue shades

Each of these channels is stored as a matrix, similar to grayscale images, but with one channel per color. When the three matrices are combined, they form a complete color image:



Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Image retrived from https://www.analyticsvidhya.com/blog/2021/03/grayscale-and-rgb-format-for-storing-images/

Matrix Representation

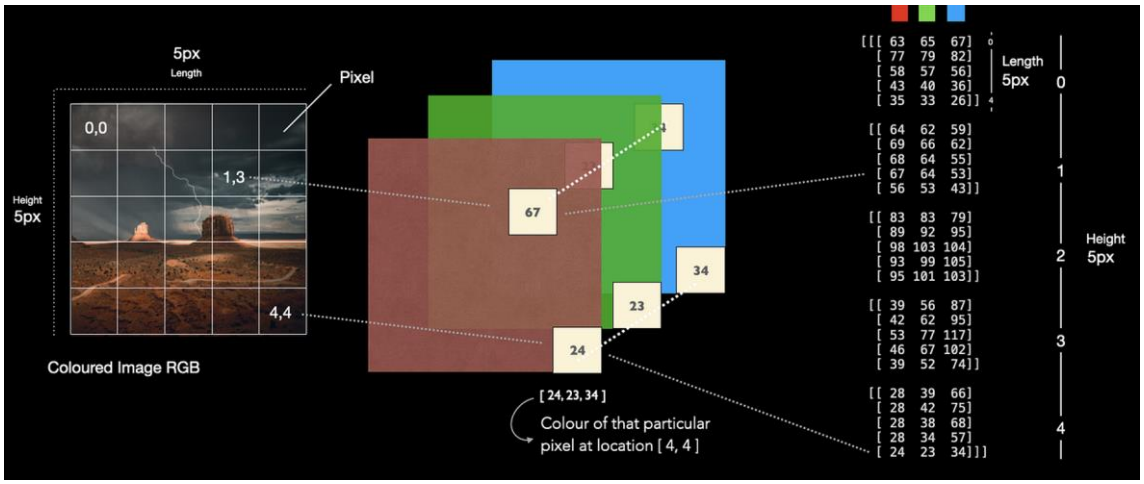
A colored image is represented by a 3D array or matrix. The three dimensions represent:

• Height (number of pixels in rows)

• Width (number of pixels in columns)

• Color channels (Red, Green, Blue)

For example, an image with dimensions 6 x 5 (height x width) and 3 channels (RGB) would be stored as a matrix of shape 6 x 5 x 3 . This array format allows the computer to store complex color information for each pixel.



Using OpenCV, load an image from the provided dataset. For example, load a tiger image by specifying its path. Then attempt to display it using Matplotlib**:**

img = cv2.imread("/content/drive/My Drive/TrainingGreeceCV/data/tiger.jpg")

plt.imshow(img) # Attempt to display the image

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
plt.title("Loaded Image (BGR)"); plt.axis('off')

plt.show()
```

Explanation: The cv2.imread() function reads the image file into a NumPy array (img). Here, img is in BGR format by default. We then use plt.imshow() to show the image. The plt.title and plt.axis('off') are used to add a title and hide axis ticks for clarity. At this stage, you might notice the colors look strange (the tiger's colors may appear distorted). This is because of the BGR vs. RGB format mismatch. Matplotlib assumes the image is in RGB, but OpenCV gave it to us in BGR, so the colors are flipped.

Step 3:Convert Color Format (BGR to RGB) for Correct Display:

To fix the color issue, convert the BGR image to RGB using OpenCV's color conversion function, then display again:

```
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Convert BGR -> RGB

plt.imshow(img_rgb)

plt.title("Image in RGB format"); plt.axis('off')

plt.show()
```

Explanation: cv2.cvtColor(img, cv2.COLOR_BGR2RGB) converts the image from BGR color space to RGB. Now the image colors will appear correct because Matplotlib is receiving an RGB-formatted image. OpenCV's default BGR ordering differs from the more common RGB format – without conversion, colors can appear incorrect. After conversion, the tiger image should look natural. This step highlights the importance of color format when exchanging images between libraries.

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Step 4: Check Image Properties (Dimensions and Channels):

Now that the image is loaded, examine its properties. You can retrieve the image's shape to get its height, width, and number of color channels:

```
height, width, channels = img.shape

print(f"Image dimensions: {width} x {height} pixels")

print(f"Number of channels: {channels}")
```

Explanation: The `shape` attribute of the image array gives us `(height, width, channels)`. For the tiger image, for example, you might see output indicating something like `Image dimensions: 1280 x 853 pixels` and `Number of channels: 3`. Three channels correspond to Red, Green, and Blue in a color image. Understanding these properties is fundamental – it tells us the resolution of the image and whether it's color or grayscale (grayscale images would have just one channel).

Step 5: Convert to Grayscale:

Many image processing tasks require grayscale images. Convert the loaded color image to grayscale and display it:

```
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  # Convert color to grayscale

print(f"Grayscale image shape: {img_gray.shape}")

plt.imshow(img_gray, cmap='gray')

plt.title("Grayscale Image"); plt.axis('off')

plt.show()
```

Explanation: Using `cv2.COLOR_BGR2GRAY` converts the original BGR image to a single-channel grayscale image (`img_gray`). Printing the shape now will show only

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

two values (height and width) because there is just one channel. For instance, `Grayscale image shape: (853, 1280)` and no channel number. We use Matplotlib with a grayscale colormap (`cmap='gray'`) to display the image in shades of gray. The grayscale image highlights intensity (brightness) of pixels only, without color. This is a common preprocessing step for many algorithms that don't require color information.

Step 6: Resize the Image

Resizing is used to scale images to a new resolution. Let's resize the original color image to a smaller size (for example, 400×400 pixels) and display it:

img_resized = cv2.resize(img_rgb, (400, 400)) # Resize to 400x400

print(f"Resized image shape: {img_resized.shape}")

plt.imshow(img_resized)

plt.title("Resized Image (400x400)"); plt.axis('off')

plt.show()

Explanation: `cv2.resize` takes the image and a tuple (new_width, new_height). The example above resizes the image to 400 by 400 pixels. The aspect ratio might not be preserved if the original wasn't square, but for learning purposes we explicitly set both dimensions equal. The printout of shape will confirm the new size. Resizing is often used to standardize image input sizes (for instance, before feeding into a neural network) or to reduce data for faster processing. Keep in mind that resizing can distort the image if the aspect ratio changes.

Step 7: Crop a Region of Interest

Cropping allows us to cut out a portion of the image. We can extract a sub-image by slicing the NumPy array of the image. For example, crop a region from the tiger image:

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
# Define region of interest coordinates

x_start, y_start = 300, 100   # top-left corner (x, y)

x_end, y_end     = 400, 200   # bottom-right corner (x, y)

img_crop = img_rgb[y_start:y_end, x_start:x_end]  # Note indexing: img[y, x]

plt.imshow(img_crop)

plt.title("Cropped Region"); plt.axis('off')

plt.show()
```

Explanation: In image arrays, indexing is `[row, column]` which corresponds to `[y, x]`. Here we take y from 100 to 200 and x from 300 to 400. This will cut out a 100×100 pixel section of the image (perhaps focusing on a specific part of the tiger). The result `img_crop` is a smaller image containing just the selected region. Cropping is useful to isolate regions of interest within an image – for example, focusing on a single object or area for analysis.

Step 8: Adjust Brightness and Contrast

Brightness can be adjusted by adding or subtracting a constant value to all pixels, and contrast can be adjusted by scaling (multiplying) pixel values. We will create a brighter and a darker version of the image, as well as versions with lower and higher contrast:

```
# Increase brightness by adding 50 (ensure values stay in [0,255] range)

img_brighter = cv2.add(img, np.ones(img.shape, dtype='uint8') * 50)

# Decrease brightness by subtracting 50

img_darker   = cv2.add(img, np.ones(img.shape, dtype='uint8') * -50)

# Decrease contrast (scale pixel values down)

img_low_contrast  = cv2.multiply(img, np.array([0.3]))

# Increase contrast (scale pixel values up, with clipping)
```

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
img_high_contrast = cv2.multiply(img, np.array([3.0]))
```

Explanation: OpenCV's `cv2.add` and `cv2.multiply` functions handle pixel-wise addition and multiplication. We use an array of ones (`np.ones`) scaled by 50 to add to the image, making it brighter, and -50 to make it darker. (OpenCV ensures the result stays within valid range 0–255 for pixel values.) For contrast, multiplying by 0.3 reduces contrast (pixels intensities are closer together, image looks grayer), while multiplying by 3.0 increases contrast (making dark regions darker and bright regions brighter). After each operation, you could display `img_brighter`, `img_darker`, etc., to see the effect. Brightness manipulation helps improve visibility under various lighting, and contrast adjustment can enhance or reduce differences in intensity. These techniques are often necessary for preparing images taken in suboptimal lighting conditions.

Step 9: Rotate the Image

Rotation transforms an image by pivoting it around a center point. Let's rotate the image by 45 degrees around its center:

```
(h, w) = img.shape[:2]

center = (w//2, h//2)  # center of the image (x, y)

M = cv2.getRotationMatrix2D(center, 45, 1.0)     # rotation matrix for 45 degrees

img_rotated = cv2.warpAffine(img, M, (w, h))

plt.imshow(cv2.cvtColor(img_rotated, cv2.COLOR_BGR2RGB))

plt.title("Rotated Image (45°)"); plt.axis('off')

plt.show()
```

Explanation: We first determine the image center coordinates. `cv2.getRotationMatrix2D(center, angle, scale)` generates a matrix to rotate the image by the given angle (45° here) with the specified scale (1.0 means no scaling). `cv2.warpAffine` applies this matrix to produce the rotated image. We use the original image's width and height as the output size to keep the whole image. The result may have black corners where no pixel data lands, due to rotation.

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Displaying the rotated image (after converting to RGB for proper color display) shows the tiger rotated. Rotation is useful for aligning images or as a form of data augmentation so that models learn to handle different orientations

---

Step 10 :Flip the Image (Mirroring)

Flipping creates a mirror image. We can flip horizontally (left-right) or vertically (upside-down). Here's how to do both:

img_flipped_h = cv2.flip(img, 1) # Horizontal flip (1 = flip around y-axis)

img_flipped_v = cv2.flip(img, 0) # Vertical flip (0 = flip around x-axis)

plt.imshow(cv2.cvtColor(img_flipped_h, cv2.COLOR_BGR2RGB))

plt.title("Horizontally Flipped"); plt.axis('off')

plt.show()

Explanation: `cv2.flip` with flag `1` produces a horizontal mirror image, flipping the image left-to-right. A flag of `0` flips it vertically (upside down), and using `-1` would flip both axes. Flipping is a straightforward way to augment your dataset (e.g., to get mirror images of your data) and to account for symmetry in imagesfile-x1zapa7w5kurnvmsrjyzxv. For example, a horizontally flipped tiger image will have the tiger facing the opposite direction. Vertical flipping is less common in natural images unless needed for specific effects or data augmentation.

Step 10 :Flip the Image (Mirroring)

Flipping creates a mirror image. We can flip horizontally (left-right) or vertically (upside-down). Here's how to do both:

img_flipped_h = cv2.flip(img, 1) # Horizontal flip (1 = flip around y-axis)

img_flipped_v = cv2.flip(img, 0) # Vertical flip (0 = flip around x-axis)

plt.imshow(cv2.cvtColor(img_flipped_h, cv2.COLOR_BGR2RGB))

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

plt.title("Horizontally Flipped"); plt.axis('off')

plt.show()

Explanation: `cv2.flip` with flag `1` produces a horizontal mirror image, flipping the image left-to-right. A flag of `0` flips it vertically (upside down), and using `-1` would flip both axes. Flipping is a straightforward way to augment your dataset (e.g., to get mirror images of your data) and to account for symmetry in imagesfile-x1zapa7w5kurnvmsrjyzxv. For example, a horizontally flipped tiger image will have the tiger facing the opposite direction. Vertical flipping is less common in natural images unless needed for specific effects or data augmentation.

---

11. Evaluation Criteria:

- **Correct Execution of Tasks:** The participant can successfully load images and perform all outlined operations without errors. For example, their code correctly converts BGR images to RGB for display and produces intended outputs (resized images, cropped regions, etc.).

- **Understanding of Concepts:** The learner can explain in their own words the difference between color and grayscale images, the effect of each transformation, and why certain steps are necessary. This might be assessed through short written answers or a verbal discussion.

- **Practical Application:** Given a new image, the learner can replicate the process to analyze it.

---

12. Languages Available:

English, **Portuguese** (translations in progress)

---

13. Contributors:

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Fátima Pais, Luís Soares – Project Designers and Facilitators

---

14. Feedback and Continuous Improvement:

- **Surveys**: At the end of this milestone, students and educators will complete feedback surveys to assess the effectiveness of the project.
- **Project Reviews**: After project completion, a final review meeting will be held to discuss potential improvements for future AI-based projects.
- **Data Analysis**: Feedback data will be analyzed to identify strengths and areas for improvement, informing future projects.

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.