# Python - Automate Resizing of Images in a Folder to a Fixed Width and Height

Image resizing is essential for various applications, including social media posts, website optimization, and digital marketing.

[[](https://substackcdn.com/image/fetch/$s_!NlR_!,f_auto,q_auto:good,fl_progressive:steep/https%3A%2F%2Fsubstack-post-media.s3.amazonaws.com%2Fpublic%2Fimages%2Ffc6b324f-d9e6-4b42-9010-725d7042475d_1280x720.png)](https://substackcdn.com/image/fetch/$s_!NlR_!,f_auto,q_auto:good,fl_progressive:steep/https%3A%2F%2Fsubstack-post-media.s3.amazonaws.com%2Fpublic%2Fimages%2Ffc6b324f-d9e6-4b42-9010-725d7042475d_1280x720.png" \t "_blank)

**1. Problem Statement : Image Resizing**

**Scenario:** You need to resize multiple images to fit social media post requirements.

**Use Case:** Automate resizing of images in a folder to a fixed width and height.

**Tools & Modules:**

* PIL (from pillow to manipulate images)

**2. Why We Need This Use Case**

Image resizing is essential for various applications, including social media posts, website optimization, and digital marketing. Instead of resizing images manually, automating the process saves time, ensures uniformity, and minimizes errors.

**3. When We Need This Use Case**

* When preparing images for social media platforms (Instagram, Facebook, Twitter, etc.).
* When optimizing images for web pages to reduce load time.
* When converting a large set of high-resolution images to a specific format for mobile apps.
* When maintaining consistency in image sizes for branding and design purposes.
* When reducing image sizes for better storage management.

**4. Challenge Questions**

1. How can we ensure only image files are processed and other file types are ignored?
2. What would happen if the input folder contains non-image files?
3. How can we modify the script to maintain the aspect ratio while resizing?
4. What are the differences between Image.LANCZOS and other resizing filters?
5. How can we extend this script to support bulk image format conversion (e.g., PNG to JPG)?
6. How does the Image.resize() function work, and what does the Image.LANCZOS filter do?
7. What happens if the input folder is empty? How can you modify the script to handle this case?
8. How can we modify the script to allow users to input the image dimensions dynamically?
9. What happens if a file in the input folder is not an image (e.g., a .txt file)? How can we prevent errors?
10. How can we add a feature to display the original and new image sizes before and after resizing?
11. How can we modify the script to convert all images to a single format, such as .jpg, while resizing?
12. What are the differences between resizing PNG, JPG, and GIF images? How can we ensure transparency is preserved in PNG and GIF files?
13. How can we handle images that have different color modes (e.g., RGB vs. CMYK)?
14. How can we modify the script to maintain EXIF metadata (camera settings, GPS data) in the resized images?
15. What happens if an animated GIF is resized? How can we ensure that all frames of the GIF are resized properly?
16. How can we modify the script to maintain the original aspect ratio of the images instead of forcing them into a fixed size?
17. What is the difference between Image.LANCZOS, Image.BILINEAR, and Image.NEAREST filters? When should each be used?
18. How can we modify the script to allow users to specify a percentage-based resizing instead of fixed dimensions?
19. What happens if we resize an image to a very small size and then enlarge it again? How can we minimize quality loss?
20. How can we reduce file size without significantly affecting image quality?
21. How can we improve the performance of the script when processing thousands of images?
22. How can we implement parallel processing (using multiprocessing or threading) to speed up image resizing?
23. How can we add a progress bar (e.g., using tqdm or progressbar) to show the progress of the resizing operation?
24. What are some ways to optimize memory usage when working with large image files?
25. How can we resize images in-place without creating a separate output folder? What are the pros and cons of this approach?
26. How can we modify the script to log errors when an image fails to resize instead of stopping the entire script?
27. How can we handle cases where an image file is corrupted and cannot be opened?
28. How can we modify the script to retry an image resizing operation if it fails due to memory issues?
29. How can we add detailed logging (e.g., using Python’s logging module) to track processed images and errors?
30. How can we display a summary report at the end, showing the number of images processed, skipped, and failed?

**5. Prerequisites for the Lab**

* Basic Python knowledge: Understanding basic syntax and operations.
* Python 3 installed on the system.
* Familiarity with file handling and directory operations in Python.
* Understanding of the Pillow (PIL) library for image manipulation.
* A folder with images to test the script.

**6. Advantages and Disadvantages**

**Advantages**

✔️ Automates repetitive image resizing tasks.  
✔️ Ensures consistency in image dimensions.  
✔️ Saves time when handling large batches of images.  
✔️ Reduces manual errors and improves efficiency.  
✔️ Supports multiple image formats (PNG, JPG, JPEG, BMP, GIF).

**Disadvantages**

❌ The script requires images to be in a specific input folder.  
❌ May cause image quality loss if resized incorrectly.  
❌ Doesn’t maintain aspect ratio unless explicitly handled.  
❌ Requires Pillow (PIL) to be installed in Python.

**7. Step-by-Step Implementation Instructions**

This script automates resizing multiple images in a folder to a fixed width and height using the **Pillow (PIL) library** in Python.

Step 1: Import Necessary Modules

from PIL import Image

import os

Step 2: Define the resize\_images() Function

def resize\_images(input\_folder, output\_folder, width, height):

This function takes **four parameters**:

* input\_folder: Path of the folder containing original images.
* output\_folder: Path of the folder where resized images will be saved.
* width: Desired width of the resized images.
* height: Desired height of the resized images.

Step 3: Create Output Folder if It Doesn’t Exist

if not os.path.exists(output\_folder):

os.makedirs(output\_folder)

* **Checks if the output folder exists.**
* If not, it **creates** the output folder.

Step 4: Loop Through All Images in the Input Folder

for filename in os.listdir(input\_folder):

* os.listdir(input\_folder) gets all file names in the input\_folder.
* The **for loop** iterates over each file.

Step 5: Check If the File is an Image

if filename.lower().endswith(('png', 'jpg', 'jpeg', 'bmp', 'gif')):

* This ensures that only **image files** (PNG, JPG, JPEG, BMP, GIF) are processed.
* lower() ensures case insensitivity (e.g., JPG and jpg are treated the same).

Step 6: Open the Image File

img\_path = os.path.join(input\_folder, filename)

img = Image.open(img\_path)

* os.path.join(input\_folder, filename): Combines folder path and filename to get the **full path** of the image.
* Image.open(img\_path): Opens the image.

Step 7: Resize the Image

img\_resized = img.resize((width, height), Image.LANCZOS)

* resize((width, height), Image.LANCZOS):
* **Resizes** the image to the specified width and height.
* **Uses**Image.LANCZOS**for high-quality resizing.** (Earlier, Image.ANTIALIAS was used, but it was removed in Pillow 10.0.)

Step 8: Save the Resized Image

output\_path = os.path.join(output\_folder, filename)

img\_resized.save(output\_path)

print(f'Resized {filename} and saved to {output\_path}')

Step 9: Call the Function with Example Parameters

input\_folder = 'input\_images' # Folder containing original images

output\_folder = 'output\_images' # Folder to save resized images

width, height = 800, 800 # Set required width and height

resize\_images(input\_folder, output\_folder, width, height)

* Defines the **input and output folder paths**.
* Sets the **desired image dimensions (800x800 pixels)**.
* Calls the resize\_images() function to **start processing the images**.

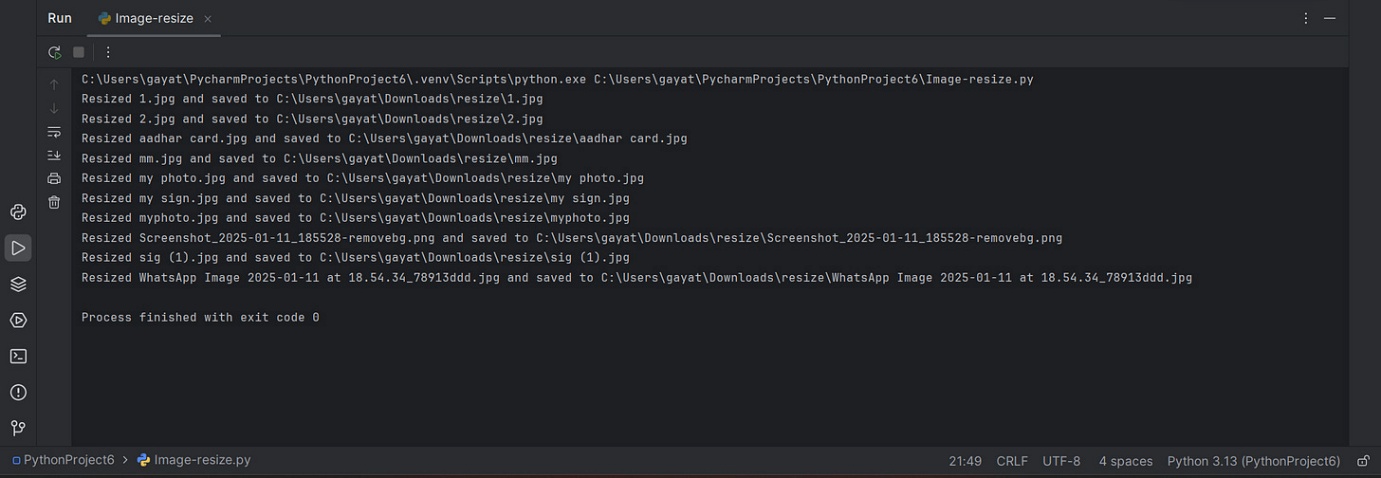
How to Use the Script?

1. Install Pillow (if not installed):

pip install pillow

1. **Create a folder named**input\_images and add some images inside it.
2. **Run the script.**
   * Resized images will be saved in a new folder named output\_images.
3. **Check the**output\_images**folder** for resized images.

Expected Output (Terminal)

[[](https://substackcdn.com/image/fetch/$s_!VWzT!,f_auto,q_auto:good,fl_progressive:steep/https%3A%2F%2Fsubstack-post-media.s3.amazonaws.com%2Fpublic%2Fimages%2F23f6b39d-4c04-4e3c-8ca1-e00b23a317f1_1914x662.png)](https://substackcdn.com/image/fetch/$s_!VWzT!,f_auto,q_auto:good,fl_progressive:steep/https%3A%2F%2Fsubstack-post-media.s3.amazonaws.com%2Fpublic%2Fimages%2F23f6b39d-4c04-4e3c-8ca1-e00b23a317f1_1914x662.png" \t "_blank)

**8. Conclusion**

Automating image resizing with Python and Pillow helps streamline workflows for web developers, designers, and content creators. By modifying the script, users can enhance its capabilities, such as preserving aspect ratios, adding watermarks, or converting image formats.