

★ Get unlimited access to the best of Medium for less than \$1/week. [Become a member](#)



The Complete Guide to Image Preprocessing Techniques in Python



Maahi Patel · [Follow](#)

11 min read · Oct 23, 2023



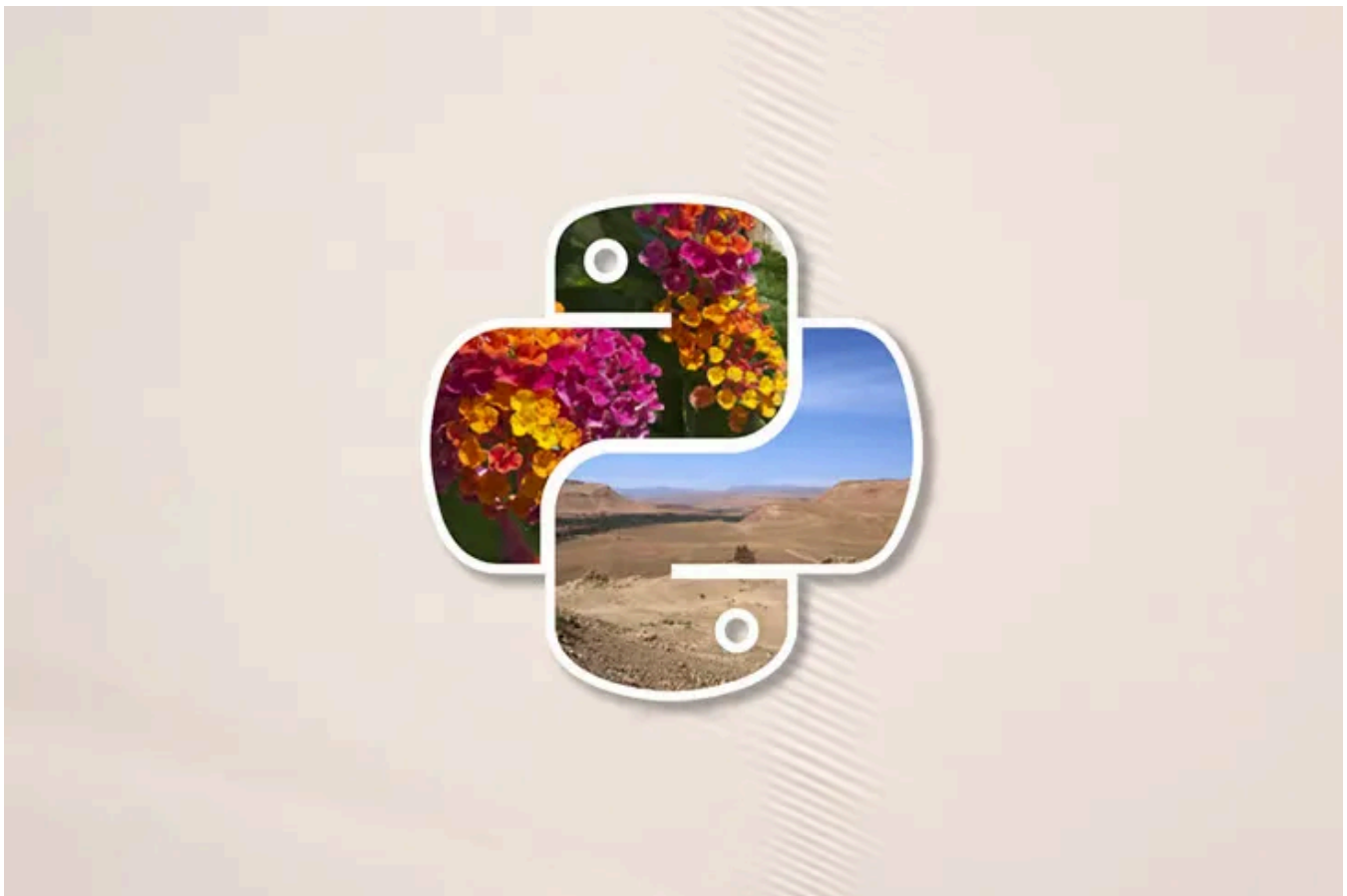
Listen



Share



More



Have you ever struggled with poor quality images in your machine learning or computer vision projects? Images are the lifeblood of many AI systems today, but not all images are created equal. Before you can train a model or run an algorithm, you often need to do some preprocessing on your images to get the best results. Image preprocessing in Python is your new best friend.

In this guide, you'll learn all the tips and tricks for preparing your images for analysis using Python. We'll cover everything from resizing and cropping to reducing noise and normalizing. By the time you're done, your images will be ready for their closeup. With the help of libraries like OpenCV, Pillow, and scikit-image, you'll be enhancing images in no time. So get ready to roll up your sleeves — it's time to dive into the complete guide to image preprocessing techniques in Python!

What Is Image Preprocessing and Why Is It Important?

Image preprocessing is the process of manipulating raw image data into a usable and meaningful format. It allows you to eliminate unwanted distortions and enhance specific qualities essential for computer vision applications. Preprocessing is a crucial first step to prepare your image data before feeding it into machine learning models.

There are several techniques used in image preprocessing:

- **Resizing:** Resizing images to a uniform size is important for machine learning algorithms to function properly. We can use OpenCV's `resize()` method to resize images.
- **Grayscale:** Converting color images to grayscale can simplify your image data and reduce computational needs for some algorithms. The `cvtColor()` method can be used to convert RGB to grayscale.
- **Noise reduction:** Smoothing, blurring, and filtering techniques can be applied to remove unwanted noise from images. The `GaussianBlur()` and `medianBlur()` methods are commonly used for this.
- **Normalization:** Normalization adjusts the intensity values of pixels to a desired range, often between 0 to 1. This can improve the performance of machine learning models. `Normalize()` from scikit-image can be used for this.
- **Binarization:** Binarization converts grayscale images to black and white by thresholding. The `threshold()` method is used to binarize images in OpenCV.
- **Contrast enhancement:** The contrast of images can be adjusted using histogram equalization. The `equalizeHist()` method enhances the contrast of images.

With the right combination of these techniques, you can significantly improve your image data and build better computer vision applications. Image preprocessing allows you to refine raw images into a format suitable for the problem you want to solve.

Loading and Converting Images With Python Libraries

To get started with image processing in Python, you'll need to load and convert your images into a format the libraries can work with. The two most popular options for this are OpenCV and Pillow.

Loading images with OpenCV: OpenCV can load images in formats like PNG, JPG, TIFF, and BMP. You can load an image with:

```
import cv2
image = cv2.imread(path/to/image.jpg')
```

This will load the image as a NumPy array. The image is in the BGR color space, so you may want to convert it to RGB.

Loading images with Pillow: Pillow is a friendly PIL (Python Image Library) fork. It supports even more formats than OpenCV, including PSD, ICO, and WEBP. You can load an image with:

```
from PIL import Image
image = Image.open('path/to/image.jpg')
```

The image will be in RGB color space.

Converting between color spaces: You may need to convert images between color spaces like RGB, BGR, HSV, and Grayscale. This can be done with OpenCV or Pillow. For example, to convert BGR to Grayscale in OpenCV, use:

```
gray = cv2.cvtColor (image, cv2.COLOR_BGR2GRAY)
```

Or to convert RGB to HSV in Pillow:

```
image = image.convert('HSV')
```

With these foundational skills, you'll be ready to move on to more advanced techniques like resizing, filtering, edge detection, and beyond. The possibilities are endless! What image processing project will you build?

Resizing and Cropping Images to Standard Dimensions

Resizing and cropping your images is an important first step in image preprocessing.

Images come in all shapes and sizes, but machine learning algorithms typically require a standard size. You'll want to resize and crop your images to square dimensions, often 224x224 or 256x256 pixels.

In Python, you can use the OpenCV or Pillow library for resizing and cropping. With OpenCV, use the `resize()` function. For example:

```
import cv2
img = cv2.imread('original.jpg')
resized = cv2.resize(img, (224, 224))
```

This will resize the image to 224x224 pixels.

To crop an image to a square, you can calculate the center square crop size and use `crop()` in OpenCV with the center coordinates. For example:

```
height, width = img.shape[:2]
size = min(height, width)
x = (width - size) // 2
y = (height - size) // 2
cropped = img[y:y+size, x:x+size]
```

With Pillow, you can use the `Image.open()` and `resize()` functions. For example:

```
from PIL import Image
img = Image.open('original.jpg')
resized = img.resize((224, 224))
```

To crop the image, use `img.crop()`. For example:

```
width, height = img.size
size = min(width, height)
left = (width - size) / 2
top = (height - size) / 2
right = (width + size) / 2
bottom = (height + size) / 2
cropped = img.crop((left, top, right, bottom))
```

Resizing and cropping your images to a standard size is a crucial first step. It will allow your machine learning model to process the images efficiently and improve the accuracy of your results. Take the time to resize and crop your images carefully, and your model will thank you!

Normalising Pixel Values for Consistent Brightness

When working with image data, it's important to normalize the pixel values to have a consistent brightness and improve contrast. This makes the images more suitable for analysis and allows machine learning models to learn patterns independent of lighting conditions.

Rescaling Pixel Values: The most common normalization technique is rescaling the pixel values to range from 0 to 1. This is done by dividing all pixels by the maximum pixel value (typically 255 for RGB images). For example:

```
import cv2
Img = cv2.imread('image.jpg')
normalized = img / 255.0
```

This will scale all pixels between 0 and 1, with 0 being black and 1 being white.

Histogram Equalization: Another useful technique is histogram equalization. This spreads out pixel intensities over the whole range to improve contrast. It can be applied with OpenCV using:

```
eq_img = cv2.equalizeHist(img)
```

This works well for images with low contrast where pixel values are concentrated in a narrow range.

For some algorithms, normalizing to have zero mean and unit variance is useful. This can be done by subtracting the mean and scaling to unit variance:

```
mean, std = cv2.meanStdDev (img)
std_img = (img - mean) / std
```

This will center the image around zero with a standard deviation of 1.

There are a few other more complex normalization techniques, but these three methods—rescaling to the 0–1 range, histogram equalization, and standardization — cover the basics and will prepare your image data for most machine learning applications. Be sure to apply the same normalization to both your training and testing data for the best results.

Applying Filters to Reduce Noise and Sharpen Images

Once you have your images loaded in Python, it's time to start enhancing them. Image filters are used to reduce noise, sharpen details, and overall improve the quality of your images before analysis. Here are some of the main filters you'll want to know about:

Gaussian Blur:

The Gaussian blur filter reduces detail and noise in an image. It “blurs” the image by applying a Gaussian function to each pixel and its surrounding pixels. This can help smooth edges and details in preparation for edge detection or other processing techniques.

Median Blur:

The median blur filter is useful for removing salt and pepper noise from an image. It works by replacing each pixel with the median value of its neighboring pixels. This can help smooth out isolated noisy pixels while preserving edges.

Laplacian Filter:

The Laplacian filter is used to detect edges in an image. It works by detecting areas of rapid intensity change. The output will be an image with edges highlighted, which can then be used for edge detection. This helps identify and extract features in an image.

Unsharp Masking:

Unsharp masking is a technique used to sharpen details and enhance edges in an image. It works by subtracting a blurred version of the image from the original image. This amplifies edges and details, making the image appear sharper. Unsharp masking can be used to sharpen details before feature extraction or object detection.

Bilateral Filter:

The bilateral filter smooths images while preserving edges. It does this by considering both the spatial closeness and color similarity of pixels. Pixels that are close together spatially and similar in color are smoothed together. Pixels that are distant or very different in color are not smoothed. This results in a smoothed image with sharp edges.

The bilateral filter can be useful for noise reduction before edge detection.

By applying these filters, you'll have high-quality, enhanced images ready for in-depth analysis and computer vision tasks. Give them a try and see how they improve your image processing results!

Detecting and Removing Backgrounds With Segmentation

Detecting and removing backgrounds from images is an important preprocessing step for many computer vision tasks. Segmentation separates the foreground subject from the background, leaving you with a clean image containing just the subject.

There are a few common ways to perform image segmentation in Python using OpenCV and scikit-image:

Thresholding:

Thresholding converts a grayscale image into a binary image (black and white) by choosing a threshold value. Pixels darker than the threshold become black, and pixels lighter become white. This works well for images with high contrast and uniform lighting. You can use OpenCV's `threshold()` method to apply thresholding.

Edge Detection:

Edge detection finds the edges of objects in an image. By connecting edges, you can isolate the foreground subject. The Canny edge detector is a popular algorithm implemented in `scikit-image`'s `canny()` method. Adjust the `low_threshold` and `high_threshold` parameters to detect edges.

Region Growing:

Region growing starts with a group of seed points and grows outward to detect contiguous regions in an image. You provide the seed points, and the algorithm examines neighboring pixels to determine if they should be added to the region. This continues until no more pixels can be added. The `skimage.segmentation.region_growing()` method implements this technique.

Watershed:

The watershed algorithm treats an image like a topographic map, with high intensity pixels representing peaks and valleys representing borders between regions. It starts at the peaks and floods down, creating barriers when different regions meet. The `skimage.segmentation.watershed()` method performs watershed segmentation.

By experimenting with these techniques, you can isolate subjects from the background in your images. Segmentation is a key first step, allowing you to focus your computer vision models on the most important part of the image-the foreground subject.

Using Data Augmentation to Expand Your Dataset

Data augmentation is a technique used to artificially expand the size of your dataset by generating new images from existing ones. This helps reduce overfitting and improves the generalization of your model. Some common augmentation techniques for image data include:

Flipping and rotating:

Simply flipping (horizontally or vertically) or rotating (90, 180, 270 degrees) images

can generate new data points. For example, if you have 1,000 images of cats, flipping and rotating them can give you 4,000 total images (1,000 original + 1,000 flipped horizontally + 1,000 flipped vertically + 1,000 rotated 90 degrees).

Cropping:

Cropping images to different sizes and ratios creates new images from the same original. This exposes your model to different framings and compositions of the same content. You can create random crops of varying size, or target more specific crop ratios like squares.

Color manipulation:

Adjusting brightness, contrast, hue, and saturation are easy ways to create new augmented images. For example, you can randomly adjust the brightness and contrast of images by up to 30% to generate new data points. Be careful not to distort the images too much, or you risk confusing your model.

Image overlays:

Overlaying transparent images, textures or noise onto existing images is another simple augmentation technique. Adding things like watermarks, logos, dirt/scratches or Gaussian noise can create realistic variations of your original data. Start with subtle overlays and see how your model responds.

Combining techniques:

For the biggest increase in data, you can combine multiple augmentation techniques on the same images. For example, you can flip, rotate, crop and adjust the color of images, generating many new data points from a single original image. But be careful not to overaugment, or you risk distorting the images beyond recognition!

Using data augmentation, you can easily multiply the size of your image dataset by 4x, 10x or more, all without collecting any new images. This helps combat overfitting and improves model accuracy, all while keeping training time and cost the same.

Choosing the Right Preprocessing Steps for Your Application

Choosing the right preprocessing techniques for your image analysis project depends on your data and goals. Some common steps include:

Resizing:

Resizing images to a consistent size is important for machine learning algorithms to work properly. You'll want all your images to be the same height and width, usually a small size like 28x28 or 64x64 pixels. The `resize()` method in OpenCV or Pillow libraries make this easy to do programmatically.

Color conversion:

Converting images to grayscale or black and white can simplify your analysis and reduce noise. The `cvtColor()` method in OpenCV converts images from RGB to grayscale. For black and white, use thresholding.

Noise reduction:

Techniques like Gaussian blurring, median blurring, and bilateral filtering can reduce noise and smooth images. OpenCV's `GaussianBlur()`, `medianBlur()`, and `bilateralFilter()` methods apply these filters.

Normalization

Normalizing pixel values to a standard range like 0 to 1 or -1 to 1 helps algorithms work better. You can normalize images with the `normalize()` method in scikit-image.

Contrast enhancement:

For low contrast images, histogram equalization improves contrast. The `equalizeHist()` method in OpenCV performs this task.

Edge detection:

Finding the edges or contours in an image is useful for many computer vision tasks. The Canny edge detector in OpenCV's `Canny()` method is a popular choice.

The key is choosing techniques that will prepare your images to suit your particular needs. Start with basic steps like resizing, then try different methods to improve quality and see which ones optimize your results. With some experimenting, you'll find an ideal preprocessing workflow.

Image Preprocessing Techniques FAQs

Now that you have a good grasp of the various image preprocessing techniques in Python, you probably have a few lingering questions. Here are some of the most frequently asked questions about image preprocessing and their answers:

What image formats does Python support?

Python supports a wide range of image formats through libraries like OpenCV and

Pillow.

Some of the major formats include:

- JPEG — Common lossy image format
- PNG — Lossless image format good for images with transparency
- TIFF — Lossless image format good for high color depth images
- BMP — Uncompressed raster image format

When should I resize an image?

You should resize an image when:

- The image is too large to process efficiently. Reducing size can speed up processing.
- The image needs to match the input size of a machine learning model.
- The image needs to be displayed on a screen or webpage at a specific size.

What are some common noise reduction techniques?

Some popular noise reduction techniques include:

- Gaussian blur — Uses a Gaussian filter to blur the image and reduce high frequency noise.
- Median blur — Replaces each pixel with the median of neighboring pixels.

Effective at removing salt and pepper noise.

- Bilateral filter — Blurs images while preserving edges. It can remove noise while retaining sharp edges.

What color spaces are supported in OpenCV and how do I convert between them?

OpenCV supports RGB, HSV, LAB, and Grayscale color spaces. You can convert between color spaces using the `cvtColor` function. For example:

Convert RGB to Grayscale:

```
gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
```

Convert RGB to HSV:

```
hsv = cv2.cvtColor (img, cv2.COLOR_RGB2HSV)
```

Convert RGB to LAB:

```
lab = cv2.cvtColor (img, cv2.COLOR_RGB2LAB)
```

Converting to different color spaces is useful for certain computer vision tasks like thresholding, edge detection, and object tracking.

Conclusion

So there you have it, a complete guide to getting your images ready for analysis in Python. With the power of OpenCV and other libraries, you now have all the tools you need to resize, enhance, filter, and transform your images. Go ahead and play around with the different techniques, tweak the parameters, and find what works best for your specific dataset and computer vision task. Image preprocessing may not be the sexiest part of building an AI system, but it's absolutely critical to get right. Put in the work upfront, and you'll have clean, optimized images ready to feed into your machine learning models. Your computer vision system will thank you, and you'll achieve better results faster. Happy image processing!

[Machine Learning](#)[Image Processing](#)[Computer Vision](#)[Follow](#)

Written by Maahi Patel

43 Followers · 3 Following

Responses (6)



What are your thoughts?

Respond



Hiya Parekh

Jun 14, 2024 (edited)



Such a clear and concise article on image processing! Very well explained 😊



2

[Reply](#)



Data Quotient

May 7, 2024



it would be helpful to have before and after image for each code snippet.



3

[Reply](#)



Victorroza

Mar 12, 2024



The most complete and useful publication for those who work with image processing! congratulations



5

[Reply](#)

See all responses

Recommended from Medium



Avinash Maheshwari

Image Processing Tool with Python

In this article, we'll walk through the creation of an intuitive image processing and real time viewer tool using Python, Tkinter, and...

🌟 Jan 12 🖱️ 56 💬 1



Abisha

Image Feature Extraction using Python - Part I

Basics of Image feature extraction techniques using python

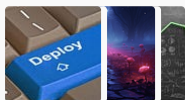
Open in app ↗

Medium

 Search

k

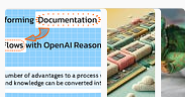
Lists

**Predictive Modeling w/ Python**

20 stories · 1793 saves

**Practical Guides to Machine Learning**

10 stories · 2173 saves

**Natural Language Processing**

1894 stories · 1555 saves

**The New Chatbots: ChatGPT, Bard, and Beyond**

12 stories · 545 saves



In The Deep Hub by Jorgecardete

The Art and Science of Interpolation

Exploring the pillars of image processing

Feb 9, 2024



622



4





 Elmar H.

Image Preprocessing for Computer Vision: Data Augmentation

What is Data Augmentation?

★ Sep 10, 2024




 In AI Innovator From PrismAI by Abhijat Sarari

License Plate Recognition with OpenCV and Tesseract OCR

License Plate Recognition (LPR) is a powerful tool in computer vision, used in applications like automated toll collection, traffic...

★ Nov 4, 2024 🖱 54



 Eran Feit

3D Photo Magic | Convert Any Picture to 3D with Python

Hi,

★ Sep 11, 2024 🖱 6



See more recommendations