# Process image data

This guide shows specific methods for processing image datasets. Learn how to:

- Use map() with image dataset.
- Apply data augmentations to a dataset with set_transform().

For a guide on how to process any type of dataset, take a look at the general process guide.

## Map

The map() function can apply transforms over an entire dataset.

For example, create a basic `Resize` function:

```
>>> def transforms(examples):
...     examples["pixel_values"] = [image.convert("RGB").resize((100,100)) for image in examples["
...     return examples
```

Now use the map() function to resize the entire dataset, and set `batched=True` to speed up the process by accepting batches of examples. The transform returns `pixel_values` as a cacheable `PIL.Image` object:

```
>>> dataset = dataset.map(transforms, remove_columns=["image"], batched=True)
>>> dataset[0]
{'label': 6,
 'pixel_values': <PIL.PngImagePlugin.PngImageFile image mode=RGB size=100x100 at 0x7F058237BB10>}
```

The cache file saves time because you don't have to execute the same transform twice. The map() function is best for operations you only run once per training - like resizing an image - instead of using it for operations executed for each epoch, like data augmentations.

map() takes up some memory, but you can reduce its memory requirements with the following parameters:

- `batch_size` determines the number of examples that are processed in one call to the

transform function.
- `writer_batch_size` determines the number of processed examples that are kept in memory before they are stored away.

Both parameter values default to 1000, which can be expensive if you are storing images. Lower these values to use less memory when you use map().

## Apply transforms

🤗 Datasets applies data augmentations from any library or package to your dataset. Transforms can be applied on-the-fly on batches of data with set_transform(), which consumes less disk space.

> [!TIP]
> The following example uses torchvision, but feel free to use other data augmentation libraries like Albumentations, Kornia, and imgaug.

For example, if you'd like to change the color properties of an image randomly:

```
>>> from torchvision.transforms import Compose, ColorJitter, ToTensor

>>> jitter = Compose(
...     [
...          ColorJitter(brightness=0.25, contrast=0.25, saturation=0.25, hue=0.7),
...          ToTensor(),
...     ]
... )
```

Create a function to apply the `ColorJitter` transform:

```
>>> def transforms(examples):
...     examples["pixel_values"] = [jitter(image.convert("RGB")) for image in examples["image"]]
...     return examples
```

Apply the transform with the set_transform() function:

```
>>> dataset.set_transform(transforms)
```