# Dataset

Create a dataset of image files and convert them into pixel data.

Chapter Goals:

- Learn how to create a dataset of image files
- Convert a dataset of image files into decoded pixel data

## A. Image dataset

Normally when we do image related tasks we're dealing with a large amount of image data. In this case, it's best to use a TensorFlow dataset, i.e. `tf.data.Dataset`, to store all the images. We can create a dataset using the `from_tensor_slices` function.

The `Dataset` class makes it easier and more efficient to perform tasks with all the image files. If you want to learn more about using TensorFlow's `tf.data` API, check out the Industry Case Study course on Educative.

## B. Mapping

After we create a dataset with the image files, we will need to decode each file's contents into usable pixel data. Since the `decode_image` function works for single image files, we will need to use the dataset object's `map` function to apply `decode_image` to each image file in our dataset.

The output of the `map` function is a new dataset with each element now converted from the original image file to its corresponding pixel data. We use `map` rather than using a `for` loop to manually convert each image file because `map` does the image decoding in parallel across the files, making it a more efficient solution.

```
import tensorflow as tf

image_paths = ['img1.jpg', 'img2.jpg']
dataset = tf.data.Dataset.from_tensor_slices(image_paths)
def _map_fn(filename):
    # FUNCTION FROM PREVIOUS CHAPTERS
    return decode_image(...)
map_dataset = dataset.map(_map_fn)
```

Dataset of image files converted to a dataset of pixel data using the map function.

In the example above, `dataset` represents a dataset containing the names of each file. After we apply the `map` function, the output dataset (`map_dataset`) contains the pixel data from each image file.

## Time to Code!

In this chapter we'll be creating a function called `get_dataset`, which creates a dataset of image data from file paths.

We'll first convert the image file paths to a tensor of strings, so we can process it as a dataset.

**Set `filename_tensor` equal to `tf.constant` with argument `image_paths`.**

Now we can create our dataset from the tensor of image file paths.

**Set `dataset` equal to `tf.data.Dataset.from_tensor_slices` with argument `filename_tensor`.**

To apply our image decoding to each image file in the dataset, we need a wrapper function. Our wrapper function will be created **within the scope of** `get_dataset`.

**Right below the initialization of `dataset`, define a function called `_map_fn` with a single required argument, `filename`.**

**Inside `_map_fn`, return `decode_image` applied with required arguments `filename`, `image_type`, and `resize_shape`, as well as keyword argument `channels=channels`.**

We can now apply the mapping wrapper to our dataset, and return the output.

**Return `dataset.map` applied with argument `_map_fn`.**

```
import tensorflow as tf

def decode_image(filename, image_type, resize_shape, channels=0):
    value = tf.read_file(filename)
    if image_type == 'png':
        decoded_image = tf.image.decode_png(value, channels=channels)
    elif image_type == 'jpeg':
        decoded_image = tf.image.decode_jpeg(value, channels=channels)
    else:
        decoded_image = tf.image.decode_image(value, channels=channels)
    if resize_shape is not None and image_type in ['png', 'jpeg']:
        decoded_image = tf.image.resize_images(decoded_image, resize_shape)
    return decoded_image

# Return a dataset created from the image file paths
def get_dataset(image_paths, image_type, resize_shape, channels):
    # CODE HERE
    pass
```