# Resizing

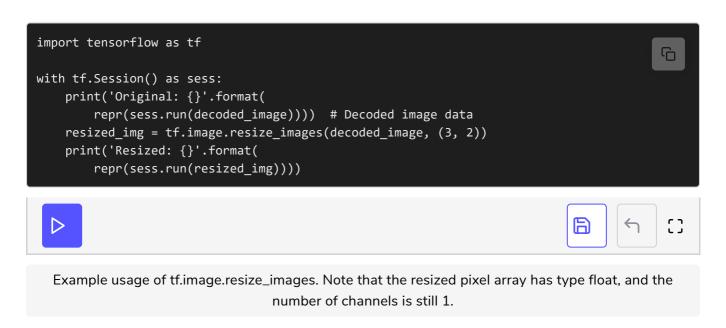
Use TensorFlow to resize images with a variety of different image scaling algorithms.

#### **Chapter Goals:**

- Be able to resize pixel data when required
- Understand how resizing works in TensorFlow

### A. Basic resizing

The function we use for resizing pixel data is <code>tf.image.resize\_images</code>. It takes in two required arguments: the original image's decoded data and the new size of the image, which is a tuple/list of two integers representing <code>new\_height</code> and <code>new\_width</code>, in that order.



The function compresses or expands the image (depending on the relationship between the new image dimensions and the old image dimensions) and then returns the pixel data for the resized image, with the same number of channels. Note that if the resized dimensions don't match the same aspect ratio as the original dimensions, the new image data will be distorted.

In the example above, the original image's dimensions were 3x3, while the resized dimensions were 3x2. This resulted in the image data becoming

aistortea.

#### B. Resizing methods

The TensorFlow tf.image.resize\_images function allows us to specify a keyword argument called method. The method argument represents the image scaling algorithm. There are 4 possible values for method:

- tf.image.ResizeMethod.BILINEAR
- tf.image.ResizeMethod.NEAREST\_NEIGHBOR
- tf.image.ResizeMethod.BICUBIC
- tf.image.ResizeMethod.AREA

The default value for method is tf.image.ResizeMethod.BILINEAR. A nice comparison of some of the methods can be found here. The comparison does not mention the AREA method, which is normally used for downsampling (resizing to a smaller size).

#### C. Unknown type

As mentioned in the previous chapter, a benefit to using <code>tf.image.decode\_image</code> is when we don't know the type of input image (e.g. PNG vs. JPEG). However, we can't use <code>tf.image.resize\_images</code> if the decoding function was <code>tf.image.decode\_image</code>. This is because the input data for <code>tf.image.resize\_images</code> needs to have a known number of dimensions, but the output of <code>tf.image.decode\_image</code> can have 3 or 4 dimensions depending on the image type.

If it is still necessary to resize an image of unknown type (non-GIF), we can use tf.image.resize\_image\_with\_crop\_or\_pad. This resizes pixel data by either padding the data with 0's (for a size increase) or cropping the pixel data (for a size decrease). Cropping the pixel data means removing certain pixels along each dimension that needs to be shrunk.

In contrast to tf.image.resize\_images, the output of

tf.image.resize\_image\_with\_crop\_or\_pad is the same type as the original image
data, since none of the individual pixels are transformed.





Example usage of tf.image.resize\_image\_with\_crop\_or\_pad. Note that the resized pixel array remains the same type.

In the example, we resize a 4x3 image (with 1 channel) to new dimensions of 5x2. The second argument of tf.image.resize\_image\_with\_crop\_or\_pad represents the new height, while the third argument represents the new width.

To decrease the width and increase the height, cropping is applied along the width dimension, while padding is applied along the height dimension.

## Time to Code!

In this chapter we'll be completing the <a href="decode\_image">decode\_image</a> function.

We need to check that there is a specified resize\_shape and the image\_type is valid.

Create an if code block outside the previous if...elif...else block. The if condition checks that both resize\_shape is not None and image\_type is either 'png' or 'jpeg'.

If the previous conditions are met, then we can resize the decoded image.

Inside the if block, set decoded\_image equal to the output of tf.image.resize\_images with first argument decoded\_image and second argument resize\_shape.

To finish the function we'll return <code>decoded\_image</code>, outside the scope of the <code>if</code> code block. This ensures that the pixel data is returned regardless of whether it is resized or not.

Return decoded\_image, outside the scope of the if block.

```
import tensorflow as tf

# Decode image data from a file in Tensorflow

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)
# CODE HERE
```









