Image Types

Learn how to decode raw image data into pixel data using TensorFlow.

Chapter Goals:

- Decode raw byte data into usable pixel data
- Understand the relationship between channels and pixels

A. Decoding

We'll now decode the raw byte data into usable pixel data. The decoding function that we use depends on the format of the image. If the input is a PNG image then we use `tf.image.decode_png`, and if the input is a JPEG image we use `tf.image.decode_jpeg`. For generic decoding (i.e. decoding any image format), we use `tf.image.decode_image`.

Since `tf.image.decode_image` can decode any type of image, you might be wondering why we even bother with the other two decoding functions. One reason is that you may want to only use specific image formats, in which case it's more efficient and better for code clarity to just use the format-specific decoding function.

Another reason is that `tf.image.decode_image` supports GIF decoding, which results in an output shape of `(num_frames, height, width, channels)`. Since the function can return data with different shapes, we can't use `tf.image.decode_image` when we also need to resize the image with `tf.image.resize_images` (see next chapter).

B. Channels

In the previous chapter we discussed image interpretations. If we only pass in `value` as the required argument for one of the decoding functions, we're using the interpretation specified in the raw image data. Normally it's fine to do this, but sometimes we want to use a specific format for the pixels.

We can change the pixel format of the decoded image via the `channels`

keyword argument.

```
import tensorflow as tf
value = tf.read_file('image3.jpg')
with tf.Session() as sess:
    arr = sess.run(tf.image.decode_jpeg(value, channels=1))
    print(arr.shape)
    print(repr(arr))
```

As you can see, the `channels` argument represents the number of integers per pixel. The default value for `channels` is `0`, which means the decoding function uses the interpretation specified from the raw data. Setting `channels` to `1` specifies a grayscale image, while setting `channels` to `3` specifies an RGB image. For PNG images we're also allowed to set `channels` to `4`, corresponding to RGBA images. Setting `channels` to `2` is invalid.

Time to Code!

In this chapter we'll continue to work on the `decode_image` function from the previous chapter. The previous chapter's code is already filled in the function.

We decode `value` based on the image type. There are two image types, PNG and JPEG, that we will specifically check for.

Create an `if...elif...else` code block, where the `if` condition checks that `image_type` equals `'png'` and the `elif` condition checks that `image_type` equals `'jpeg'`.

If the image type is PNG or JPEG we'll use the specific decoding function for the type. For all other image types (e.g. gif) or if the image type is unknown (`image_type=None`), we'll use the generic image decoding function.

Inside the `if` block, set `decoded_image` equal to the output of `tf.image.decode_png` with required argument `value` and keyword argument `channels=channels`.

Inside the `elif` block, set `decoded_image` equal to the output of `tf.image.decode_jpeg` with required argument `value` and keyword argument `channels=channels`.

Inside the `else` block, set `decoded_image` equal to the output of `tf.image.decode_image` with required argument `value` and keyword argument `channels=channels`.

```
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)
    # CODE HERE
```