PIL Library

Use the PIL library to extract and modify data from an image.

Chapter Goals:

- Learn the basics of the `PIL` module
- Use `PIL` to load an image and return its resized pixel data

A. PIL module

While we can do large scale image processing in TensorFlow, the `PIL` module (from Pillow) allows us to do more fine-grained image processing. In this Lab we will demonstrate basic resizing and filtering as an introduction to `PIL`. However, the library also has many more utility functions for advanced image processing and analysis. Documentation and extended examples using the `PIL` module can be found here.

The `PIL` module has a submodule called `Image`, which is the main module used for image processing. With it, we can create an `Image` object from an image file and obtain the image's pixel data. The nice thing about PIL is that the methods are more user-friendly than the TensorFlow methods; for example, the file reading and decoding are done together and resizing can work with unknown image formats.

The `image_mode` variable refers to the interpretation of the pixels, e.g. Grayscale vs. RGB vs. RGBA. PIL provides us with many types of modes.

We'll use `image_mode` to convert our image into RGBA format so that we can resize it without worrying about what format it's in.

B. Image modification

Similar to TensorFlow, the `Image` module allows us to resize our image. Also like TensorFlow, the `Image` module's resizing function takes in an optional resizing method, which the `PIL` documentation refers to as *resampling filters*. Detailed descriptions, as well as a scaling quality and performance comparison for each resampling filter can be found here. In the code at the end of this chapter, we will use the Lanczos resampling filter for the best quality resizing.

`PIL` also allows us to apply *filtering* to an image, which is provided via the `ImageFilter` submodule. The `ImageFilter` module has a substantial list of predefined filters, as well as some more advanced filter classes.

Filtering allows us to perform tasks such as sharpening or blurring an image's features. Filters like these are actually the crux of image recognition, and they're discussed in more detail in the **CNN** section of this course.

Time to Code!

In this chapter we'll be completing the `pil_resize_image` function, which loads and resizes an image using `PIL`.

We'll first load the image using the `Image` module, and convert it to the specified `image_mode`.

Set `im` equal to `Image.open` applied to `image_path`.

Set `converted_im` equal to `im.convert` applied to `image_mode`.

After converting the image, we'll resize it using Lanczos filtering.

Set `resized_im` equal to `converted_im.resize` applied with first argument `resize_shape` and second argument `Image.LANCZOS`.

Before returning our image data, we'll check to see if an `image_filter` is specified.

Create an `if` code block which checks that `image_filter` is not `None`.

If an `image_filter` is specified, apply it to `resized_im`.

Inside the `if` block, set `resized_im` equal to `resized_im.filter` applied with argument `image_filter`.

Now we can return the image pixel data, converted to a NumPy array.

Outside the `if` block, set `im_data` equal to the output of

`resized_im.getdata` applied with no arguments.

Then return the output of `np.asarray` applied with `im_data` as the only argument.

```
import numpy as np
from PIL import Image, ImageFilter

# Load and resize an image using PIL, and return its pixel data
def pil_resize_image(image_path, resize_shape,
    image_mode='RGBA', image_filter=None):
    # CODE HERE
    pass
```