## Data Set Preparation

Solve the following exercises and upload your solutions to Moodle (unless specified otherwise) until the specified due date. Make sure to use the *exact filenames* that are specified for each individual exercise. Unless explicitly stated otherwise, you can assume correct user input and correct arguments. You are allowed to write additional functions, classes, etc. to improve readability and code quality.

### Exercise 1 – Submission: `a2_ex1.py` 40 Points

Write a function `to_grayscale(pil_image: np.ndarray) -> np.ndarray` that converts a `pil_image` (expected to be the raw data of an image loaded with Pillow) to grayscale using the colorimetric conversion. The conversion works on an RGB input image where all values must have been normalized to the range $[0, 1]$ before the (also normalized) grayscale output $Y$ is calculated as follows:

$$C_{\text{linear}} = \begin{cases} \frac{C}{12.92} & \text{if } C \leq 0.04045, \\ \left(\frac{C+0.055}{1.055}\right)^{2.4} & \text{otherwise.} \end{cases} \qquad \text{with } C \in \{R, G, B\}$$

$$Y_{\text{linear}} = 0.2126 \cdot R_{\text{linear}} + 0.7152 \cdot G_{\text{linear}} + 0.0722 \cdot B_{\text{linear}}$$

$$Y = \begin{cases} 12.92 \cdot Y_{\text{linear}} & \text{if } Y_{\text{linear}} \leq 0.0031308, \\ 1.055 \cdot Y_{\text{linear}}^{\frac{1}{2.4}} - 0.055 & \text{otherwise.} \end{cases}$$

The function must return the denormalized grayscale-converted image *including* a dedicated channel for the brightness information in the following 3D shape: `(1, H, W)`, where `H` is the height and `W` the width of the image. The specified `pil_image` must be handled as follows:

- If the image has a 2D shape, it is assumed to already be a grayscale image with shape `(H, W)`, and a copy with shape `(1, H, W)` is returned.
- If the image has a 3D shape, it is assumed to be `(H, W, 3)`, i.e., the third dimension represents the RGB channels (in the order R, G and B, which can also be assumed to be true). If the third dimension does not have a size of exactly three, a `ValueError` must be raised.
- If the image has a different shape, a `ValueError` must be raised.
- You can assume that the images passed to the function will only have values within the range $[0, 255]$. This enables an easy normalization by dividing all image values by 255. However, the original input image must not be changed, i.e., the normalization and grayscale conversion must be done on a copy. Do not forget about the denormalization of $Y$ before you return the grayscale-converted image.
- The data type of the image can be arbitrary. If it is an `np.integer` (use `np.issubdtype`), then rounding (to zero decimals) must be applied before returning the grayscale-converted image. The returned grayscale-converted image must have the same data type as the input image, so make sure to cast appropriately.

**Hints:**

- The function `np.where` might be useful when implementing the mathematical formulae above.
- Avoid manual loops, utilize broadcasting instead.
- Do not forget about the additional brightness channel before returning the result (both when the input is already a grayscale image and when it is an RGB image).

## Exercise 2 – Submission: `a2_ex2.py` 60 Points

Write a function

```
prepare_image(
    image: np.ndarray,
    width: int,
    height: int,
        x: int,
        y: int,
        size: int
) -> tuple[np.ndarray, np.ndarray]
```

that prepares an `image` for the classification machine learning project. Its main task is to resize the given image by padding and cropping it, and also to return a subarea that is specified via the parameters `x`, `y` and `size`. All parameters are explained in the following:

- `image` is a 3D NumPy array with shape `(1, H, W)` that contains a grayscale image. This image must be resized and cropped according to the remaining parameters (cf. Figure 1 and Figure 2).
- `width` specifies the width of the resulting image(cf. Figure 1).
- `height` specifies the height of the resulting image (cf. Figure 1).
- `x` specifies the x-coordinate within `resized_image` where the subarea should start(cf. Figure 2).
- `y` specifies the y-coordinate within `resized_image` where the subarea should start (cf. Figure 2).
- `size` specifies the size in both dimensions of the cropped subarea (cf. Figure 2).

The resizing process works as follows:

- If the input image is smaller in width or height than `width` or `height` respectively, pad the image to the desired shape by equally adding pixels to the start and end of that dimension. The pad-pixel used should be the same as the previous border. If an unequal amount of padding needs to be added, add the additional pixel to the end.
- If the input image is larger in width or height than `width` or `height` respectively, crop the image to the desired shape by cutting out an equal amount of the bordering pixels in both dimensions. If an unequal amount of pixels have to be cut, break the tie by first cutting pixels at the start of the dimension.

In addition, a smaller, square subarea of the resized image should be returned. `x` and `y` specify the start of the subarea with `size` specifying the length in both dimensions.

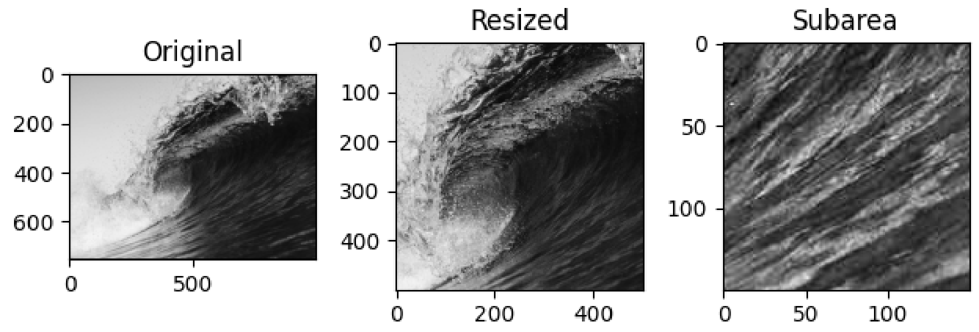

Figure 1: Expected output for `prepare_image(im, 500, 500, 300, 300, 150)`

The function must return the 2-tuple `(resized_image, subarea)`:

- `resized_image` is the 3D NumPy array that represents the resized version of the input image. The array must be a copy, i.e., the original input image must not be changed. It must have the same data type as the input image and shape `(1, height, width)`.
- `subarea` is a 3D NumPy array that represents a subarea of pixels according to the other parameters. It must have the same data type as the input image and shape `(1, size, size)`

The function must also handle various error cases. Specifically, it must raise a `ValueError` in the following cases:

- The shape of `image` is not 3D or the channel size is not exactly 1 (i.e., shape `(1, H, W)`).
- Either `width`, `height` or `size` are smaller than 32.
- `x` is smaller than 0 or `x + size` is larger than the width of the resized image, i.e., the subarea would exceed the resized image width.
- `y` is smaller than 0 or `y + size` is larger than the height of the resized image, i.e., the subarea would exceed the resized image height.

**Hints:**

- You are not allowed to use the torch or torchvision packages. `np.pad` might be useful
- Avoid loops and use slicing.

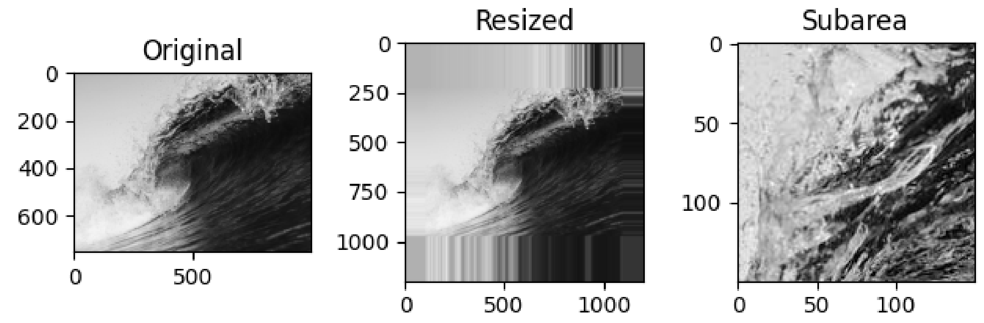

Figure 2: Expected output for `prepare_image(im, 1200, 1200, 400, 400, 150)`

- Make sure to compute the subarea on the resized image.