Computer Vision 1: Exercise Sheet 2

Summary:

1. Read, display and write images using the packages `imageio` and `matplotlib`
2. Each image is represented by a multi-dimensional `numpy` array. Image manipulation is array manipulation.
3. Statistical features of images: mean, variance and histograms

Read and display an image

The image in Figure 1 comes from the MSCOCO dataset[1]. Download it from Moodle with the name `sample.jpg`.

Figure 1: A sample image (id=345434) from the MSCOCO dataset

Read an Image

The image can be read as follows:

```
import imageio
im = imageio.imread(uri='sample.jpg')
print(im.shape) # shape of the image
print(type(im)) # type of the variable that represents the image
print(im.dtype)
```

The output should be

```
(480, 640, 3)
<class 'imageio.core.util.Image'>
uint8
```

The shape of the array is $(480, 640, 3)$ which corresponds to `(Height, Width, Channels)`, with the three channels usually standing for red (R), green (G), and blue (B). The data type is `unsigned 8 bit integer` (what is the range of values for this data type?).

[1] http://cocodataset.org/

The `type(im)` command does not give `<type 'numpy.ndarray'>` as we normally see for numpy arrays, but the class `imageio.core.util.Image` really is numpy multi-dimensional array in disguise[2].

Display an Image

One of the most popular Python packages for displaying images is `matplotlib`. An image can be visualized as follows:

```
import matplotlib.pyplot as plt
plt.imshow(im)
plt.show()
```

Note: Using `plt.show()` in a script will block the executing of the script until the figures have been closed.

A new window will pop out with our image displayed in it and some control buttons at the bottom-left corner, something like this:

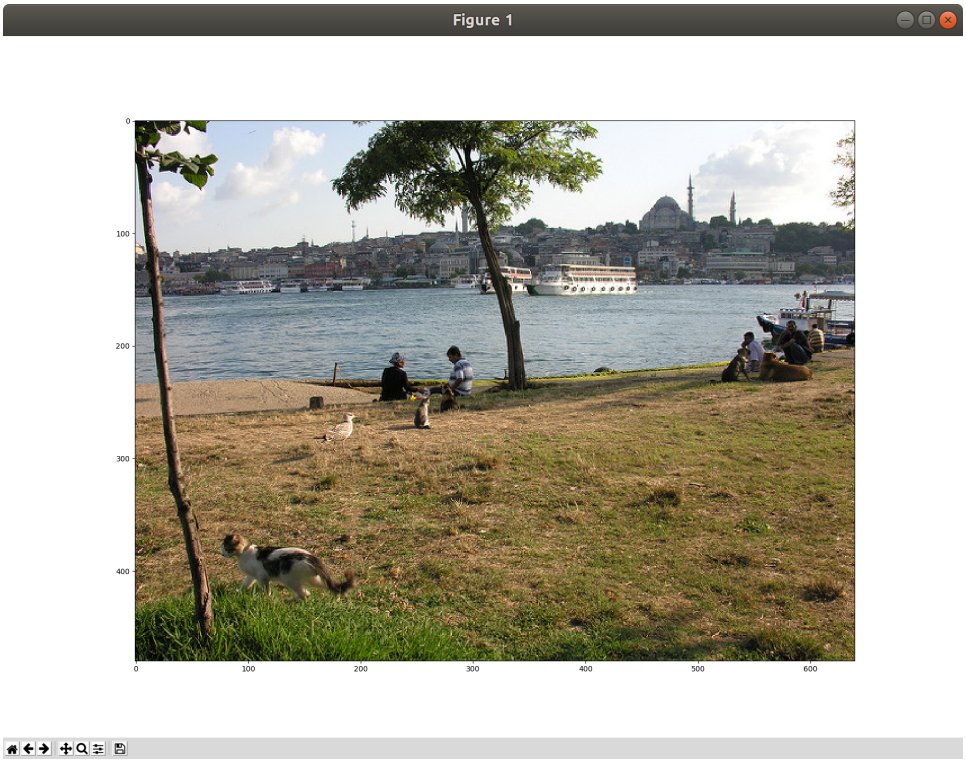

Figure 2: Display a RGB image using `matplotlib`

Note that you can also see the numbering on the axes and verify the size of the image matches what you saw earlier.

To display a grayscale image, add `cmap='gray'` to the `imshow` function. Download the grayscale image `woman.png` from Moodle and display it:

```
im_grayscale = imageio.imread(uri='woman.png')
print(im.shape)
plt.imshow(im_grayscale, cmap='gray')
plt.show()
```

Expected output:

[2] If you have doubts about this, check out the source code at `https://github.com/imageio/imageio/blob/master/imageio/core/util.py`, line 130 and 191.

Figure 3: Display a grayscale image using `matplotlib`

`Matplotlib` is a very powerful visualization tool, but in this exercise we only use it to view images. After the exercise, you may want to explore what else `matplotlib` can do by following the online tutorial at `https://matplotlib.org/tutorials/index.html`.

- Repeat the steps above for another image in the MSCOCO dataset from `http://cocodataset.org/#explore`. Either load the image to disk before reading it, or directly read from the URL provided on the dataset webpage.

Image Manipulation and writing images

Import `numpy` and set the seed of the random number generator (RNG) as follows:

```
1 import numpy as np
2 np.random.seed(0)
```

If you set a seed for the RNG, you will obtain the same sequence of pseudo-random numbers every time you run the next commands. This is useful for ensuring repeatability of experiments.

Adding noise

Use `np.random.normal` to create normally distributed noise. The documentation can be found at `https://docs.scipy.org/doc/numpy-1.15.0/reference/generated/numpy.random.normal.html`. Add the created noise array to the image array, and display the image. Note the following:

- The shape of the noise array must match the shape of the image array.
- The created noise has a data type of `float64`. When you add it to an image with type `uint8`, the result will have the type `float64`. Before displaying the image, you must convert the result back to `uint8` by using Numpy's `astype` method. To avoid overflow

and underflow in the conversion, use the `clip` method to ensure the values are in the appropriate range for `uint8` (what are the minimum and maximum value representable as an unsigned 8-bit integer?).

- Try different inputs for the mean and standard deviation of the Gaussian, these are represented by the input parameters `loc` and `scale` of `np.random.normal`. How do they visually affect the image?

An example output with `loc=0.0` and `scale=20.0` is shown in Figure . Can you see how it differs from Figure 1?

Figure 4: Original image.

Figure 5: Noisy image.

Flipping and rotating an image

- Use `np.fliplr` and `np.flipud` to flip the image, and display the result.
- Use `np.rot90` to rotate the image and display the result.

Other less basic operations such as resizing, cropping and rotating with arbitrary angles are not very well supported by `numpy`. You can use other modules and libraries such as OpenCV and Skimage to do this.

Write an image to file

Write any modified image from above to a file using `imageio`. You should use the function `imwrite`, the documentation can be found at `https://imageio.readthedocs.io/en/stable/userapi.html#imageio.imwrite` – you should only need the `uri` and `im` input arguments. Check that you can find the image on disk, and that it looks the same as your visualization with `matplotlib`.

Statistical features: Mean, variance and histogram

Numpy provides built-in functions for computing statistical image features such as the mean, variance, and histogram.

- Find the documentation of `np.mean`, `np.var`, and `np.histogram`.

- Calculate the mean and variance of the R,G,B channels of an image using just one call to `np.mean` and `np.var`. The expected outputs should have a shape `(3,)`. Hint: use the `axis` argument to calculate the mean and variance along the width and height dimensions only.

- Compute and display the histograms of each of the three channels in the image: red, green, and blue. Follow the examples on the documentation page of `np.histogram`. How does the result change if you use a different number of bins? What if you manually specify which bins to use?

  Hints:

  - `np.histogram` flattens the input array, so only give the pixel values in one channel (R, G, or B) at a time. Do you remember from the previous exercise how to index Numpy arrays?
  - Take two output arguments from `np.histogram`:

    ```
    hist, bin_edges = np.histogram(...)
    ```

    Then calculate the display width and create a Numpy array for the histogram bin centers as follows:

    ```
    width = 0.8 * (bin_edges[1] - bin_edges[0])
    bin_centers = (bin_edges[:-1] + bin_edges[1:]) / 2
    ```

    Then you can draw the histogram using `plt.bar`:

    ```
    plt.bar(bin_centers, hist, width=width)
    ```

    If you want, you can add a title to a plot, e.g., by `plt.title('Red')`. Use `plt.xlabel()` and `plt.ylabel()` for axis labels.

Histogram in a bounding box

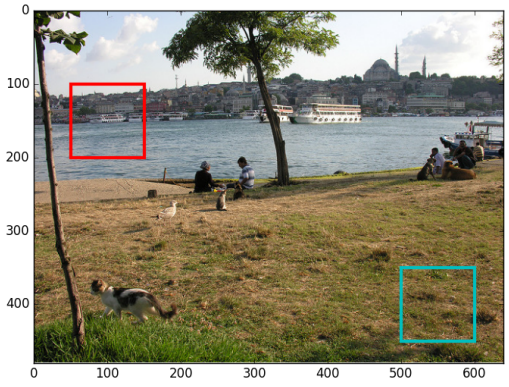

Figure 6: Figure with bounding boxes.

Figure 6 shows the image from the exercise with two bounding boxes, one red and one cyan. The upper left corner of the red bounding box is at $(50, 100)$, and the upper left corner of the cyan bounding box is at $(500, 350)$. Both bounding boxes have a size of 100-by-100 pixels. Load the sample image as `uint8`.

- For the pixels inside the red bounding box, calculate and draw the R, G, and B channel histograms for the using 20 equally spaced bins in the range between 0 and 255.
  - Use `numpy.linspace` to create an equally spaced vector to use as bins.
- Repeat the same for the pixels inside the cyan bounding box.
- Calculate the average RGB value inside each of the two bounding boxes.