Chapter 1

Images in Python

An image is usually represented as a three-dimensional array of 8-bit integers. NumPy arrays are the most commonly used library for this sort of data structure.

If you have an RGB image that is 480 pixels tall and 640 pixels wide, you will need a $480 \times 640 \times 3$ NumPy array.

There is a separate library (imageio) that:

- Reads an image file (like JPEG files) and creates a NumPy array.
- Writes a NumPy array to a file in standard image formats

Let's create a simply python program that creates a file containing an all-black image that is 640 pixels wide and 480 pixels tall. Create a file called `create_image.py`:

```
import NumPy as np
import imageio
```

```
import sys

# Check command-line arguments
if len(sys.argv) < 2:
    print(f"Usage {sys.argv[0]} <outfile>")
    sys.exit(1)

# Constants
IMAGE_WIDTH = 640
IMAGE_HEIGHT = 480

# Create an array of zeros
image = np.zeros((IMAGE_HEIGHT, IMAGE_WIDTH, 3), dtype=np.uint8)

# Write the array to the file
imageio.imwrite(sys.argv[1], image)
```

To run this, you will need to supply the name of the file you are trying to create. The extension (like .png or .jpeg) will tell imageio what format you want written. Run it now:

```
python3 create_image.py blackness.png
```

Open the image to confirm that it is 640 pixels wide, 480 pixels tall, and completely black.

1.1 Adding color

Now, let's walk through through the image, pixel-by-pixel, adding some red. We will gradually increase the red from 0 on the left to 255 on the right.

```
import NumPy as np
import imageio
import sys

# Check command-line arguments
if len(sys.argv) < 2:
    print(f"Usage sys.argv[0] <outfile>")
    sys.exit(1)

# Constants
IMAGE_WIDTH = 640
IMAGE_HEIGHT = 480
```

```
# Create an array of zeros
image = np.zeros((IMAGE_HEIGHT, IMAGE_WIDTH, 3), dtype=np.uint8)

for col in range(IMAGE_WIDTH):

    # Red goes from 0 to 255 (left to right)
    r = int(col * 255.0 / IMAGE_WIDTH)

    # Update all the pixels in that column
    for row in range(IMAGE_HEIGHT):
        # Set the red pixel
        image[row, col, 0] = r

# Write the array to the file
imageio.imwrite(sys.argv[1], image)
```

When you run the function to create a new image, it will be a fade from black to red as you move from left to right:

Now, inside the inner loop, update the blue channel so that it goes from zero at the top to 255 at the bottom:

```
    # Update all the pixels in that column
    for row in range(IMAGE_HEIGHT):

        # Update the red channel
        image[row,col,0] = r

        # Blue goes from 0 to 255 (top to bottom)
        b = int(row * 255.0 / IMAGE_HEIGHT)
        image[row,col,2] = b

imageio.imwrite(sys.argv[1], image)
```

When you run the program again, you will see the color fades from black to blue as you go down the left side. As you go down the right side, the color fades from red to magenta.

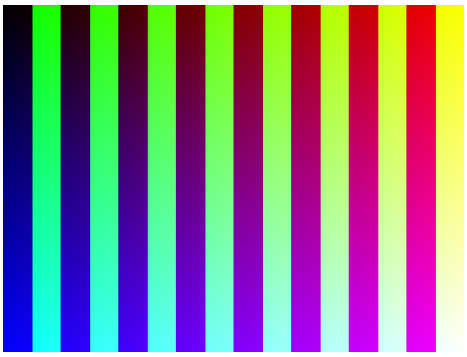
Notice that red and blue with no green looks magenta to your eye.

Now let's add some stripes of green:

```
import NumPy as np
import imageio
import sys

# Check command line arguments
if len(sys.argv) < 2:
    print(f"Usage sys.argv[0] <outfile>")
    sys.exit(1)

# Constants
IMAGE_WIDTH = 640
IMAGE_HEIGHT = 480
STRIPE_WIDTH = 40
pattern_width = STRIPE_WIDTH * 2

# Create an image of all zeros
image = np.zeros((IMAGE_HEIGHT, IMAGE_WIDTH, 3), dtype=np.uint8)

# Step from left to right
for col in range(IMAGE_WIDTH):

    # Red goes from 0 to 255 (left to right)
    r = int(col * 255.0 / IMAGE_WIDTH)

    # Should I add green to this column?
    should_green = col % pattern_width > STRIPE_WIDTH

    # Update all the pixels in that column
```

```
    for row in range(IMAGE_HEIGHT):

        # Update the red channel
        image[row,col,0] = r

        # Should I add green to this pixel?
        if should_green:
            image[row,col,1] = 255

        # Blue goes from 0 to 255 (top to bottom)
        b = int(row * 255.0 / IMAGE_HEIGHT)
        image[row,col,2] = b

imageio.imwrite(sys.argv[1], image)
```

When you run this version, you will see the previous image in half the stripes. In the other half, you will see that green fades to cyan down the left side and yellow fades to white down the right side.



1.2 Using an existing image

imageio can also be used to read in any common image file format. Let's read in an image and save each of the red, green, and blue channels out as its own image.

Create a new file called `separate_image.py`:

```
import imageio
import sys
import os

# Check command line arguments
if len(sys.argv) < 2:
```

```
    print(f"Usage {sys.argv[0]} <infile>")
    sys.exit(1)

# Read the image
path = sys.argv[1]
image = imageio.imread(path)

# What is the filename?
filename = os.path.basename(path)

# What is the shape of the array?
original_shape = image.shape

# Log it
print(f"Shape of {filename}:{original_shape}")

# Names of the colors for the filenames
colors = ['red','green','blue']

# Step through each of the colors
for i in range(3):

    # Create a new image
    newimage = np.zeros(original_shape, dtype=np.uint8)

    # Copy one channel
    newimage[:,:,i] = image[:,:,i]

    # Save to a file
    new_filename = f"{colors[i]}_{filename}"
    print(f"Writing {new_filename}")
    imageio.imwrite(new_filename, newimage)
```

Now you can run the program with any common RGB image type:

```
python3 separate_image.py dog.jpg
```

This will create three images: `red_dog.jpg`, `green_dog.jpg`, and `blue_dog.jpg`.

Appendix A

Answers to Exercises