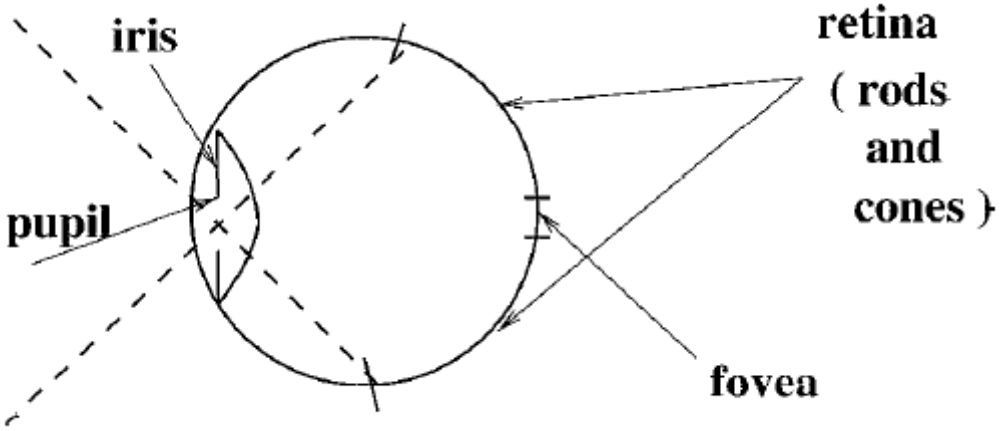
Module 3a: Image Compression

How can we represent the same amount of data while taking up less memory. Image Compression! Let's go through a simple example.

Let's begin with how our eye works.



"Crudely speaking, the human eyes is a spherical camera with a 20mm focal length...The iris controls the amount of light passing through the lens by controlling the size of the pupil...The retina is unevenly populated with sensor cells. An area near the center of the retina, called the fovea, has a very dense concentration of color receptors, called cones. Away from the center, the density of cones decreases while the density of black and white receptors, the rods, increases." (Computer Vision, Linda Shapiro, 2000)

In peripheral vision our visual system is interpreting color on top of the black and white information . The person sitting next to you might appear in color (you could have color blindness) but your eye is mainly capturing black and white (brightness) information. Since your peripheral vision has a limited amount of color receptors (cones), the color information needs to be shared among a large section of black and white (brightness) receptors (rods). Our brains combine the cone's color information with the neighboring rod's black and white information producing a color image.

Let's try to use our brains technique in an image compression algorithm

	0	1	2
0	55,100,200	74,124,100	89,210,10
1	124,74,191	174,43,34	201,142,60
2	191,50,10	215,111,84	245,139,81

How much memory is used to store one color image of size 200 pixels by 200 pixels? Which contains 40,000 individual pixels, each containing three 8 bit Integers.

40,000 pixels x 24 bits (RGB) = 960,000 bits = 120,000 bytes = 0.12 megabytes

Simple approach. Two seperate arrays.

The first contains the brightness values of all the pixels. This can be saved as one value, one 8 Bit Integer, stored in a 2D array ([]). The second contains the RGB pixel values of every other row. This can be saved in 3 values, three 8 bit Integers, stored in a 3D array [[[]]].

1) All the Brightness values. GreyScale 2) Every other row of color vales.

40,000 pixels x 8 bits = 320,000 bits = 40,000 bytes = 0.04 megabytes
20,000 pixels x 24 bits = 480,000 bits = 60,000 bytes = 0.06 megabytes

We saved 0.02 megabytes! A 17% savings! Not bad at all for such a simple approach.

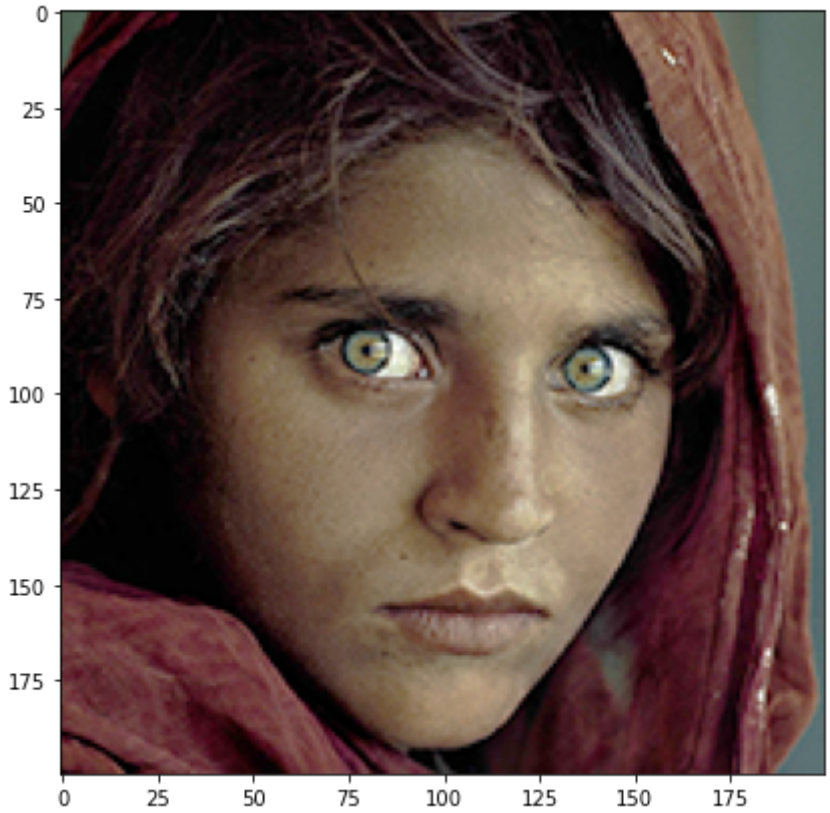
	0	1	2
0	226,226,226	226,226,226	226,226,226
1	124	124	124
2	40,40,40	40,40,40	40,40,40

In [ ]:

```
#Install Open CV and give tutorial on getting Jupyter notebook up and running.
# %pip install opencv-contrib-python
# %pip install matplotlib

#Import Libraries
import cv2
import numpy as np
import copy
import matplotlib.pyplot as plt

#Saving and displaying an image
img = cv2.imread('Graphics/face.png')

#OpenCV by defualt uses BGR instead of RGB.  First convert the color values to g
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

#cv2.imshow('Face',img)
#Use matplotlib while using a python notebook.  Let's make it bigger to see the
fig = plt.figure(figsize = (7,7))
plt.imshow(img)
```

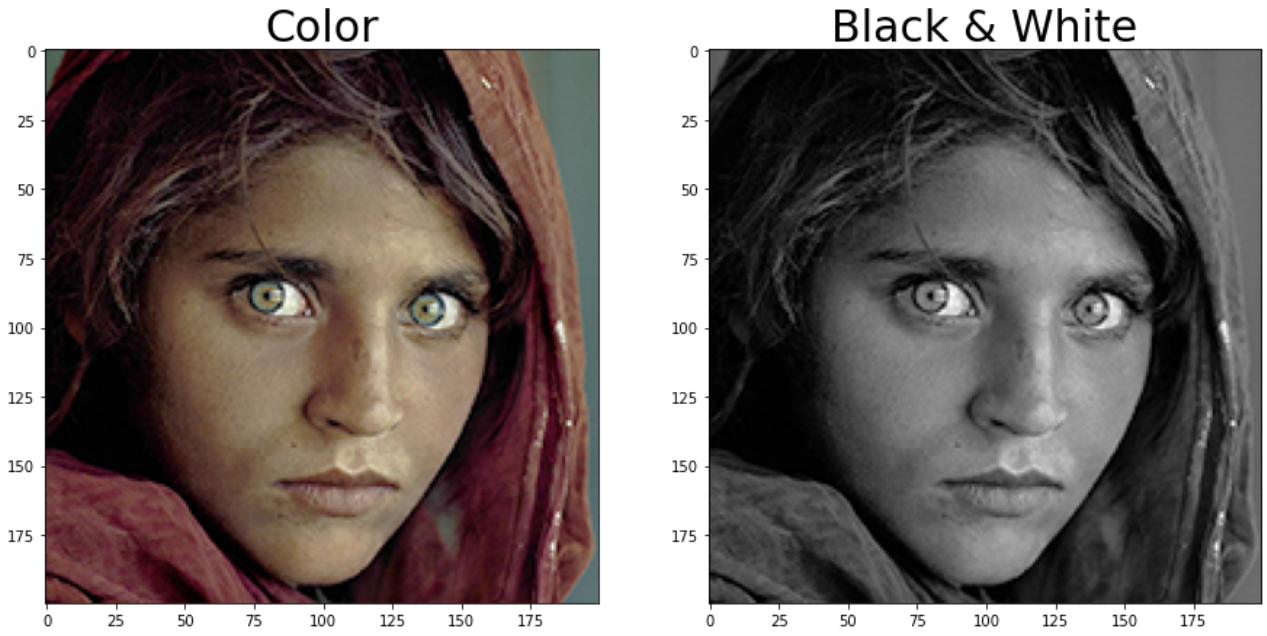
Out[ ]:

```
<matplotlib.image.AxesImage at 0x7fb98ebb9db0>
```



In [ ]:

```
# Save a copy of the image in color and grayscale.

colorData = img
greyData = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)


# Show images.
fig = plt.figure(figsize = (15,30))
columns = 2
rows = 1
fig.add_subplot(rows,columns, 1)
plt.imshow(colorData)
plt.title('Color', fontdict={'fontsize': 30})
fig.add_subplot(rows,columns, 2)
plt.imshow(greyData, cmap="gray")
plt.title('Black & White', fontdict={'fontsize': 30})
plt.show()
```



What is the size of the color and black & white array.

In [ ]:
```
print(colorData.shape)
print(greyData.shape)
```

```
(200, 200, 3)
(200, 200)
```

Compression (Encoder)

In [ ]:
```
# Create a blank array to size our compressed image.

blended = np.zeros((200,200,3), dtype=np.uint8)

#Save the height, width, and depth
h = img.shape[0]
w = img.shape[1]


for y in range(0, h):
    for x in range(0,w):
        # Use the modulo operator as a way to get odd and even rows.
        if(y % 2 == 0):
            # Save the color data in even rows.
            blended[y,x] = colorData[y,x]
        else:
            # Save the black and white data in odd rows
            blended[y,x] = greyData[y,x]
```

In [ ]:
```
fig = plt.figure(figsize = (15,15))
plt.imshow(blended)
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7fb98e16d570>
```

Convert to YCrCb color system.

In the YCbCr color model, the Y represents the luminance or brightness of the image, while the Cb and Cr represent the chrominance or color information. The Y channel is the grayscale or black-and-white part of the image, and the Cb and Cr channels represent the color information.

In the YCbCr color model, the red-difference (Cr) channel and the blue-difference (Cb) channel represent the color information that is separated from the luminance or brightness information.

The Cr channel represents the difference between the red component and the luminance value (Y), which can be calculated as:

Cr = R - Y

Similarly, the Cb channel represents the difference between the blue component and the luminance value (Y), which can be calculated as:

Cb = B - Y

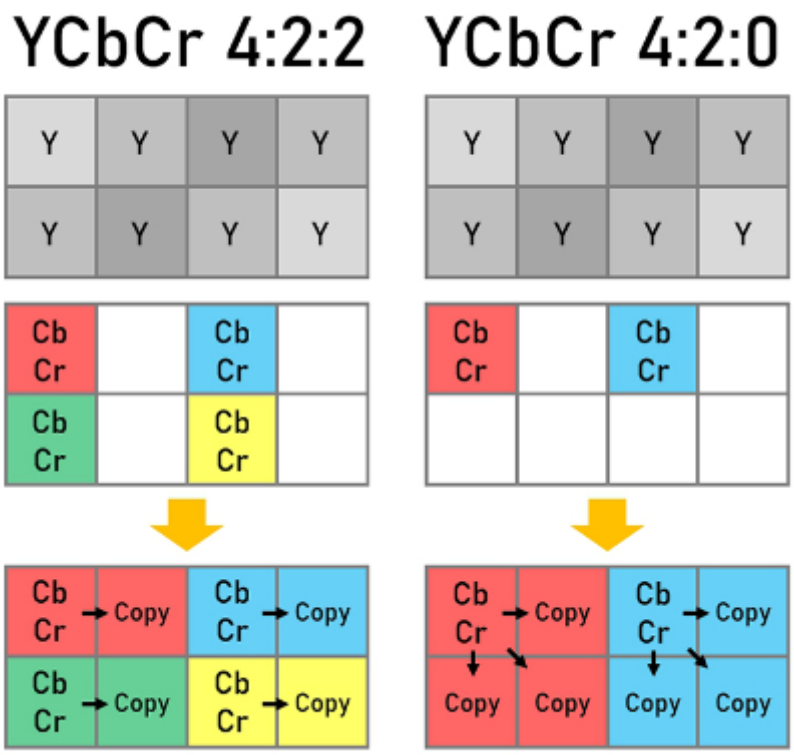
The Cr and Cb channels contain information about the color and hue of the image, which can be used for various image processing and compression techniques. By separating the color information from the luminance information, the YCbCr color model allows for more efficient compression of digital images and videos while maintaining good image quality.

RGB YCrCb JPEG (or YCC)

where

Y, Cr, and Cb cover the whole value range.

4:4:4, 4:2:2, & 4:2:0 color



In [ ]:

```
# Convert our blended compressed image to YCrCb
blended_YCrCb = cv2.cvtColor(blended, cv2.COLOR_RGB2YCrCb)

# Save each channel. Brightness, Difference in Red, Difference in blue)
y = blended_YCrCb[:,:,0]
```

```
Cr = blended_YCrCb[:,:,1]
Cb = blended_YCrCb[:,:,2]

output = [blended, y, Cr, Cb]
title = ['Image','Luminance (brightness)','Red Color Difference','Blue Color Dif
fig = plt.figure(figsize = (20,20))
for i in range(4):
    plt.subplot(2,2,i+1)
    plt.title(title[i], fontdict={'fontsize': 20})
    if i == 0:
        plt.imshow(output[i])
    else:
        plt.imshow(output[i], cmap = 'gray')

plt.show()
```

Let's steal the Red Difference and the Blue Difference from the row above but keep the Luminance (brightness)

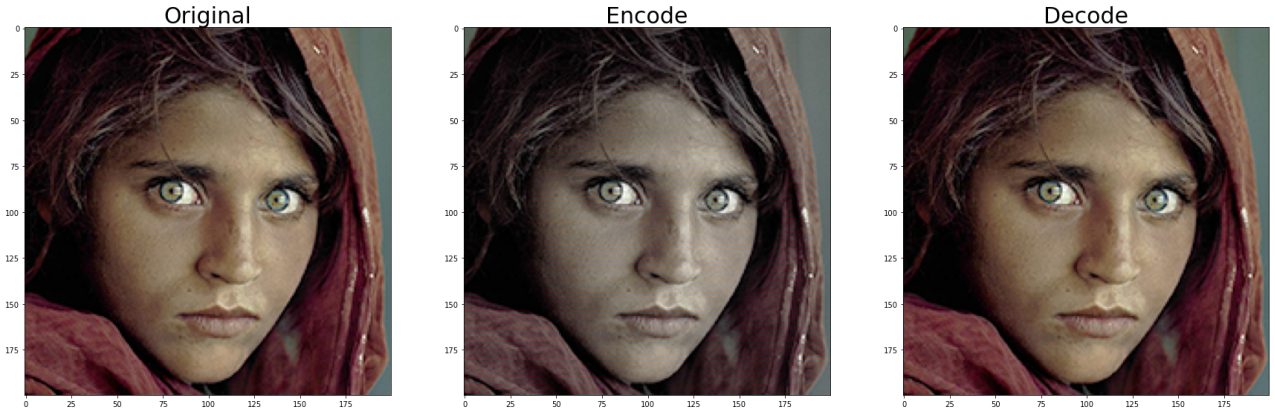
from the odd rows.

Decoder

In [ ]:
```
# Create a blank array to size our compressed image.

decode_blended = np.zeros((200,200,3), dtype=np.uint8)

#Save the height, width, and depth
h = blended.shape[0]
w = blended.shape[1]


for y in range(0, h):
    for x in range(0,w):
        # Use the modulo operator as a way to get odd and even rows.
        if(y % 2 == 0):
            # Save the color data in even rows.
            decode_blended[y,x] = blended_YCrCb[y,x]
        else:
            # Save the brightness value and take the change in red/blue from the
            decode_blended[y,x,0] = blended_YCrCb[y,x,0]
            decode_blended[y,x,1] = blended_YCrCb[y-1,x,1]
            decode_blended[y,x,2] = blended_YCrCb[y-1,x,2]
```

In [ ]:
```
decode_blended = cv2.cvtColor(decode_blended, cv2.COLOR_YCrCb2RGB)
```

In [ ]:
```
# Show images.
fig = plt.figure(figsize = (30,50))
columns = 3
rows = 1
fig.add_subplot(rows,columns, 1)
plt.imshow(img)
plt.title('Original', fontdict={'fontsize': 30})
fig.add_subplot(rows,columns, 2)
plt.imshow(blended)
plt.title('Encode', fontdict={'fontsize': 30})
fig.add_subplot(rows,columns, 3)
plt.imshow(decode_blended)
plt.title('Decode', fontdict={'fontsize': 30})
plt.show()
```



Exercises

(1) We implemented 4:2:2 image compression in class. Can you Implement 4:2:0 image compression?