ImageCompressionProject

December 6, 2024

[8]:
```
import numpy as np
from PIL import Image
#Use your own finish_im or import this one
#from mth433 import finish_im
from matplotlib import pyplot as plt
```

[104]:
```
img1 = Image.open('first.jpg')
img2 = Image.open('bluer.jpg')
img2b = Image.open('redder.jpg')
img2c = Image.open('greener.jpg')
img3 = Image.open('whiter.jpg')
img4 = Image.open('text.jpg')
img5 = Image.open('face.jpg')
img6 = Image.open('colorful.jpg')
img7 = Image.open('contrast.jpg')
img8 = Image.open('idek.jpg')
```

[11]:
```
def calc_variance(S, k):
    ssv = S**2
    var_data = []
    total_var = np.sum(ssv)
    cum_sum = np.cumsum(ssv[:k])
    return (cum_sum / total_var) * 100
```

1 Part 1

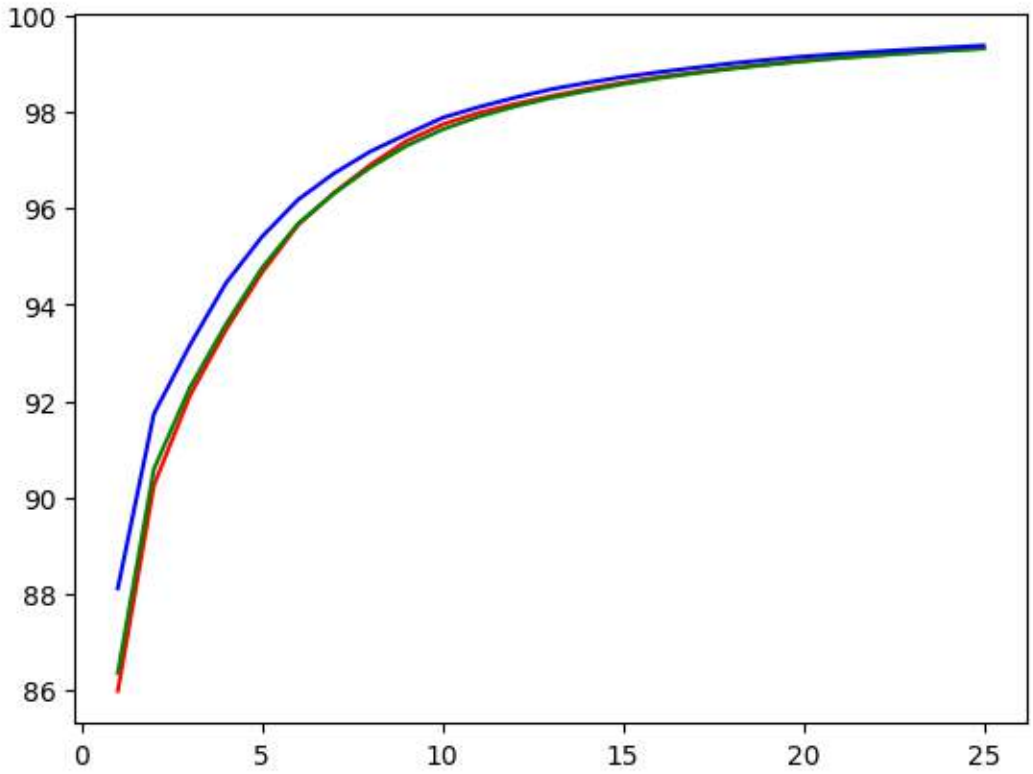
[33]:
```
def percent_variance(im, upper_rank):
    """Creates a plot of rank vs. percentage variance.
    On the x-axis are the integers k = 1,2,...,upper_rank.
    On the y-axis are three plots of the percentage variance
    captured by the first k singular values of each of the red, green and blue␣
 ↪channels.
    These should be appropriately colored as red, green and blue.

    Parameters
    ----------
```

```
    im : PIL Image
    upper_rank : int

    Returns
    -------
    None

    """
    imarr = np.array(im)
    _, S_red, _ = U, S, Vh = np.linalg.svd(imarr[:,:,0])
    _, S_green, _ = np.linalg.svd(imarr[:,:,1])
    _, S_blue, _ = np.linalg.svd(imarr[:,:,2])

    x = np.array(range(1, upper_rank+1))
    y_red = calc_variance(S_red, upper_rank)
    y_green = calc_variance(S_green, upper_rank)
    y_blue = calc_variance(S_blue, upper_rank)

    plt.plot(x, y_red, color='r')
    plt.plot(x, y_green, color='g')
    plt.plot(x, y_blue, color='b')
```

[28]:
```
img1
```

[28]:

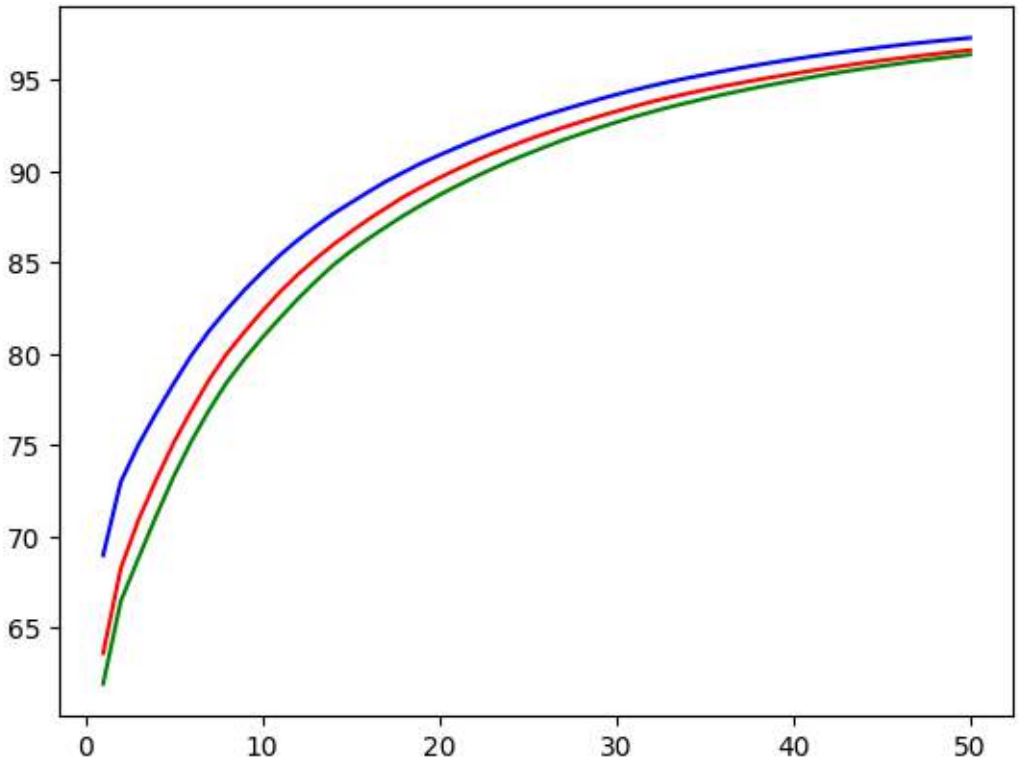
[54]:
```
percent_variance(img1, 25) # can see image mostly blues, browns
```



[39]:
```
img2
```

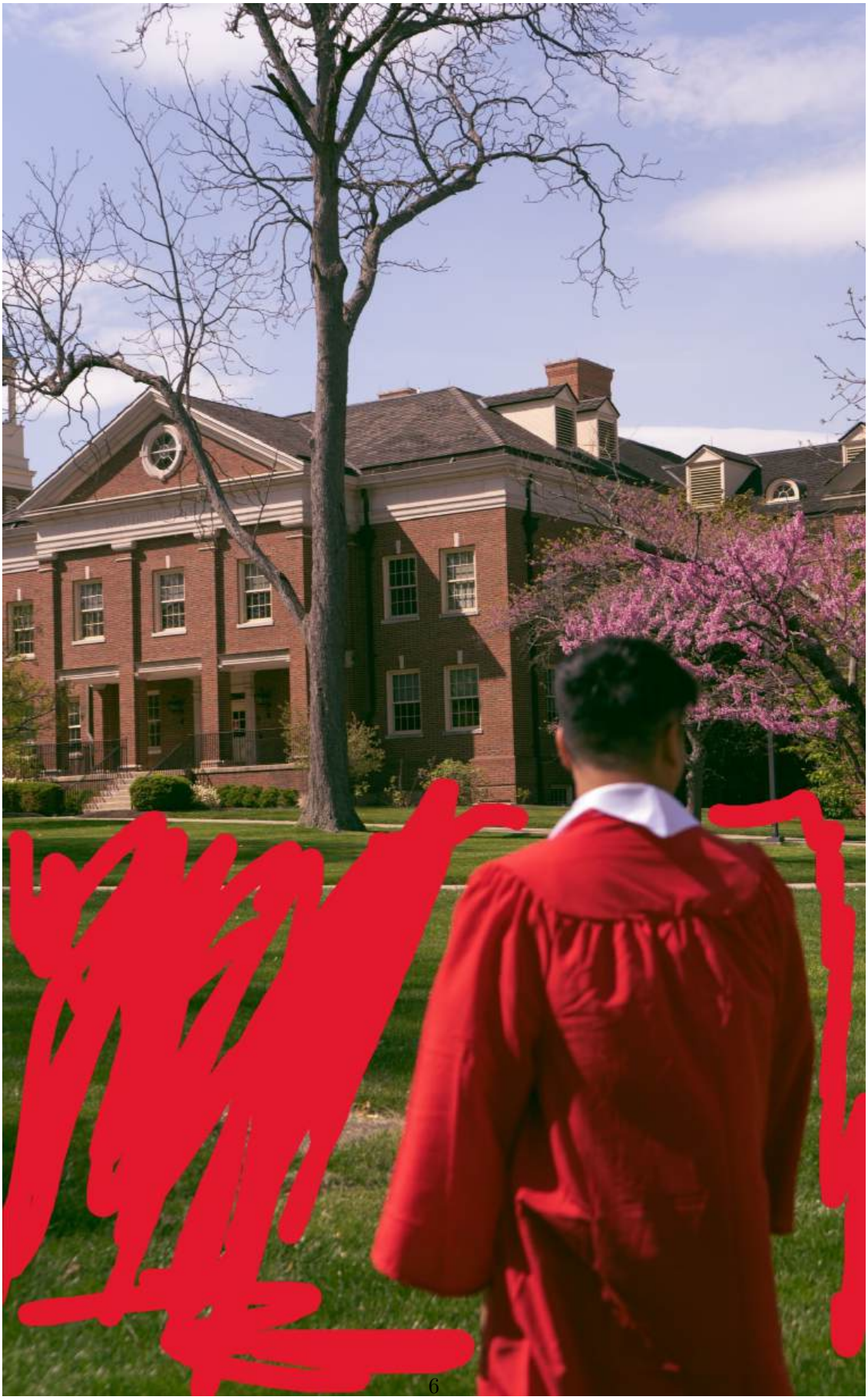
[39]:

[ ]:
```
percent_variance(img2, 50)
# blues clearly stand out
# variance btwn channels converge less strongly and at higher k compared to␣
  ↪first image
# so more color variance in image probably
```

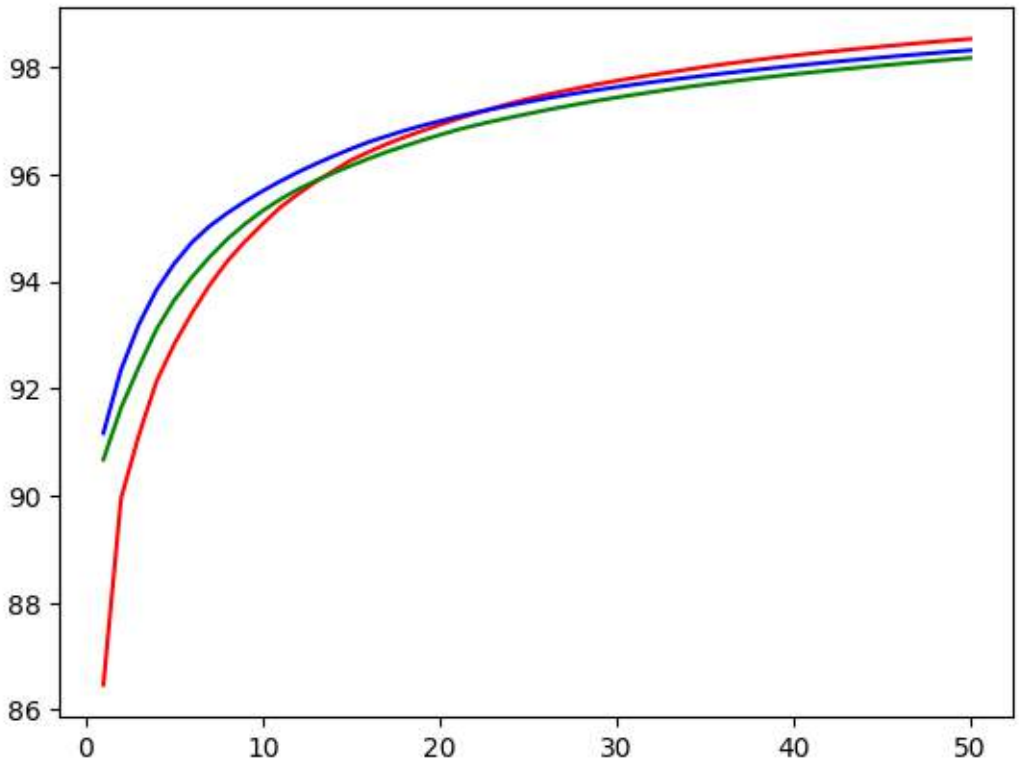


[95]:
```
img2b
```

[95]:



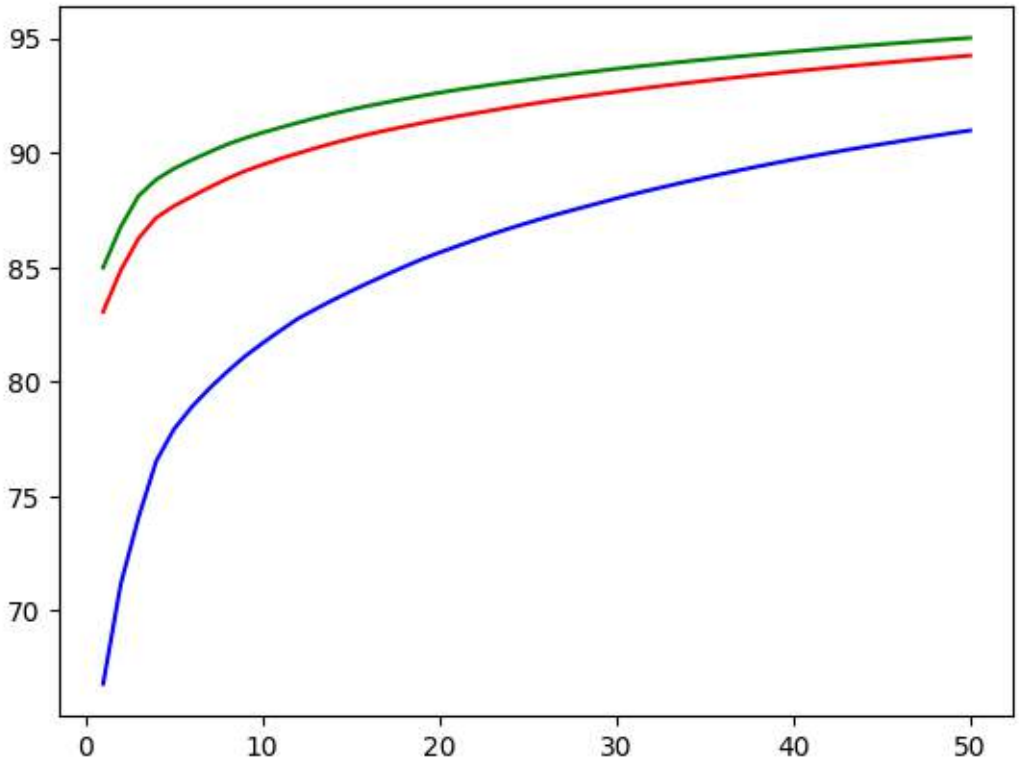
[ ]:
```
percent_variance(img2b, 50)
# more blues are captured at first
# i think it's because color patterns are uniform in the sky
# but it should be mostly red
```



[98]:
```
img2c
```

[98]:

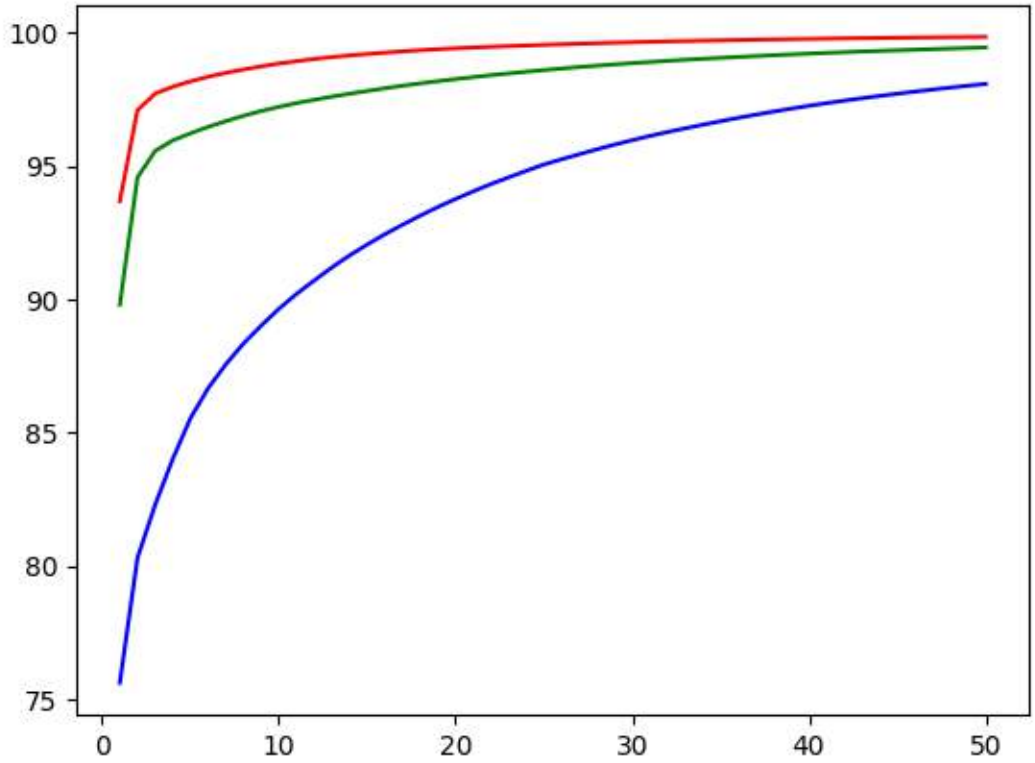
[97]:
```
percent_variance(img2c, 50) # mostly green
# fewer blues. red + green make brown bark on tree
```



[42]:
```
img6
```

[42]:

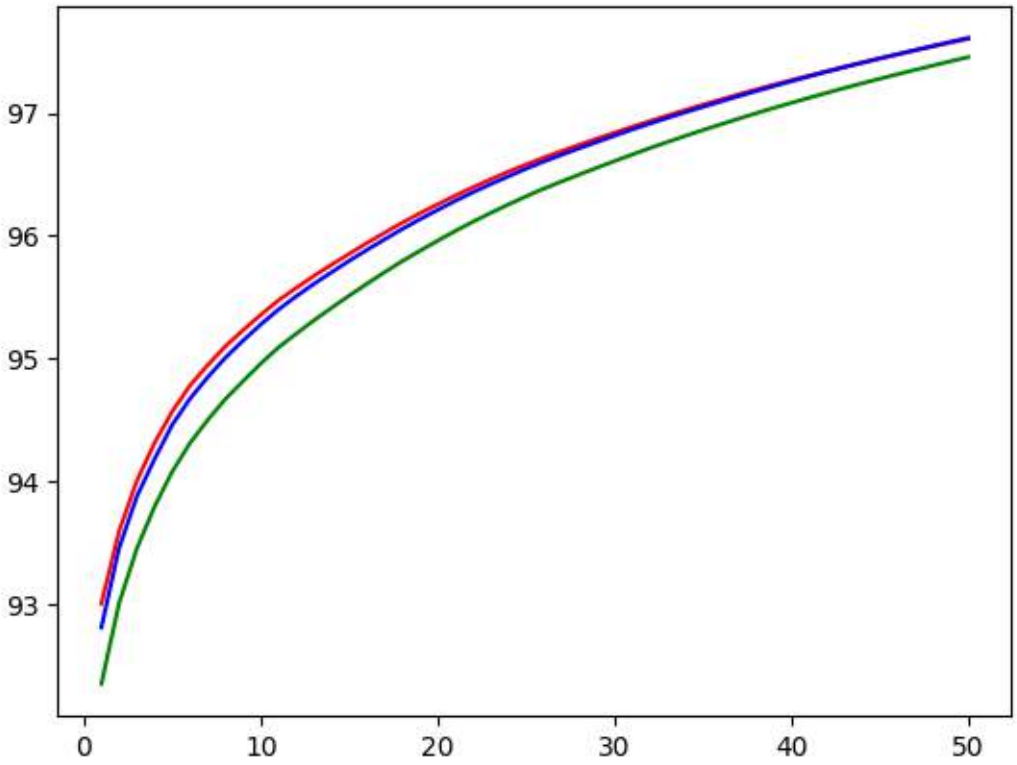
[ ]:
```
percent_variance(img6, 50) # colorful image, strong hues
# for lower k: most reds, greens captured, less blue
```



[47]:
```
img7
```

[47]:

[99]:
```
percent_variance(img3, 50) # contrasting image
```



2 Part 2

2.0.1 Problem 1

[ ]:
```
# helper function for im_approx
def channel_approx(channel, k):
    U, S, Vt = np.linalg.svd(channel)
    S_split = np.zeros_like(S)
    S_split[:k] = S[:k]

    # truncate each component with respect to upper rank specified by␣
 ↪parameter, then return result
    reduced_ch = U[:,:k] @ np.diag(S_split[:k]) @ Vt[:k,:]
    reduced_ch = np.clip(reduced_ch, 0, 255).astype(np.uint8)
    return reduced_ch
```

[60]:
```
def im_approx(im, k):
    """Returns the best rank k approximation of an image using
       the svd.

       More specifically, k is a list of 3 integers and im_approx returns
       the best k[0],k[1],k[2] approximations of the red,green,blue
```

```
    channels.

    Parameters
    ----------
    im : PIL Image
    k : list of 3 integers (e.g. [25,19,100]).

    Returns
    -------
    PIL Image
    """

    imarr = np.array(im)

    # for each color channel
    red_approx = channel_approx(imarr[:,:,0], k)
    green_approx = channel_approx(imarr[:,:,1], k)
    blue_approx = channel_approx(imarr[:,:,2], k)

    # initialize reconstruction array
    approximated_image = np.zeros_like(imarr)

    # replacing each color channel with approximations
    approximated_image[:,:,0] = red_approx
    approximated_image[:,:,1] = green_approx
    approximated_image[:,:,2] = blue_approx

    # return reconstructed image
    return Image.fromarray(np.uint8(approximated_image))
```

[64]:
```
im_approx(img1, 100)
```

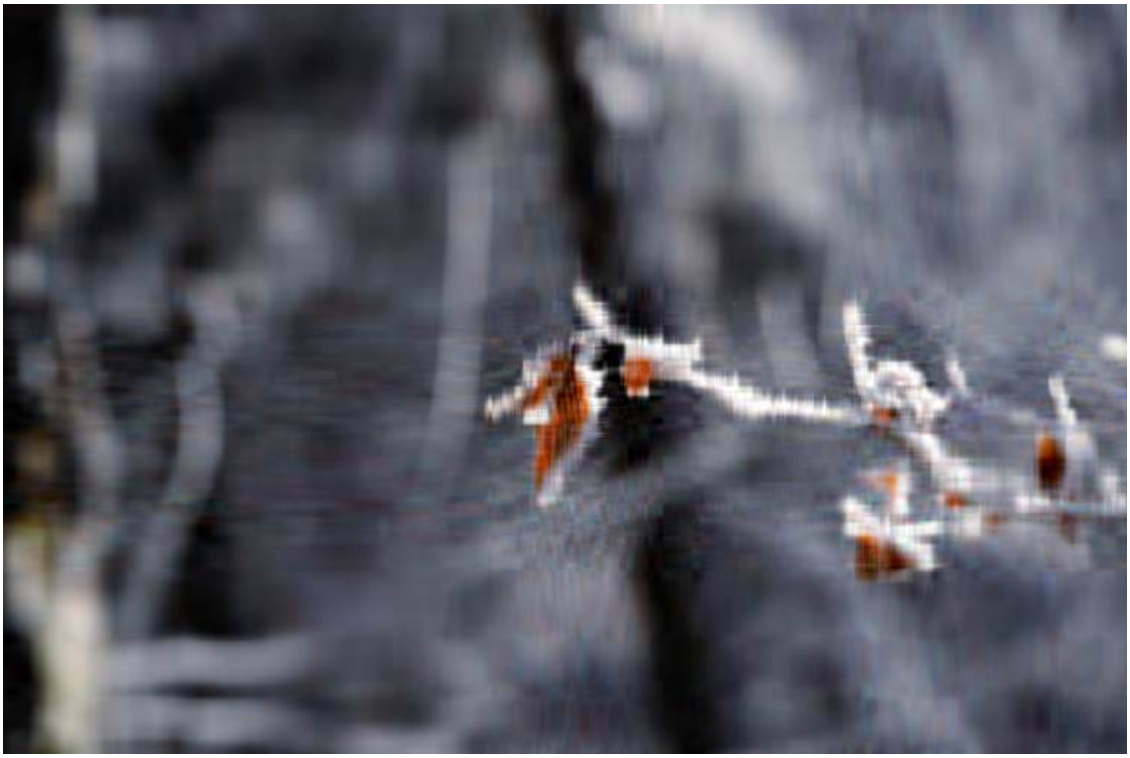
[64]:

[67]:
```
def im_reducer(im,percent):
    """Returns the best rank k approximation of an image where k/rank(im) =␣
 ↪percent.
    Calls the function im_approx

    Parameters
    ----------
    im : PIL Image
    percent : list of 3 float values in the range [0,1]

    Returns
    -------
    PIL Image

    """
    rank_approx = []
    imarr = np.array(im)
    rank = np.min(imarr.shape) # rank is smallest of n, m
    new_image = im_approx(im, np.ceil(percent*rank)) # rounds upper rank up
    return new_image
```

2.0.2 Problem 2

[65]:
```
im_approx(img1, 25)
```

[65]:



[68]:
```
im_reducer(img3, 50)
```

[68]:

[74]:
```
im_approx(img3, 100)
```

[74]:

[ ]:
```
img4 # image of text
```

[ ]:

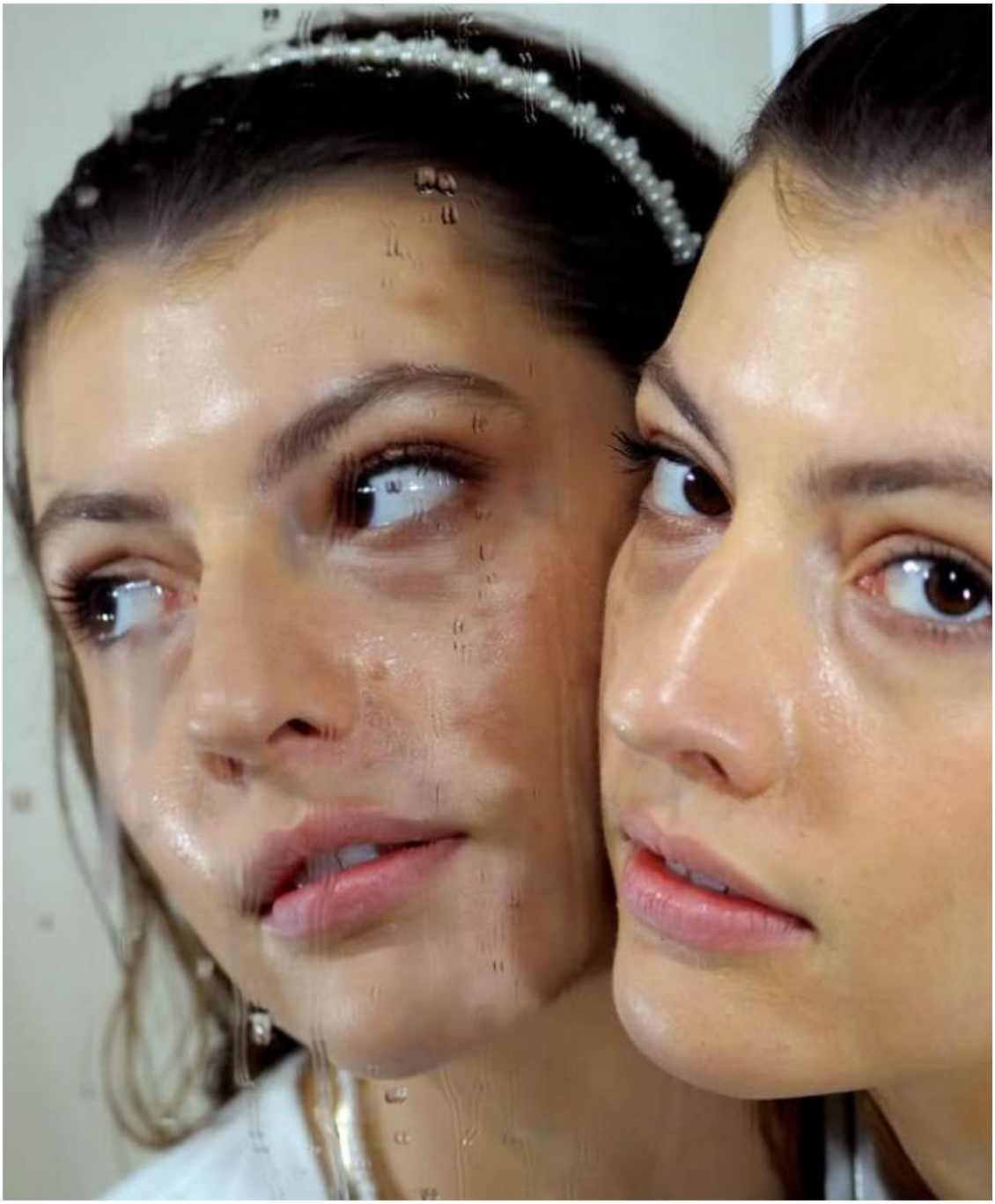
[ ]:
```
im_reducer(img4, 8) # can see numbers / text at lower k, but pixels start␣
↪blending and it's hard to tell
```

[ ]:

[ ]:
```
im_reducer(img4, 2)
```
[ ]:

[ ]:
```
img5 # original image
```

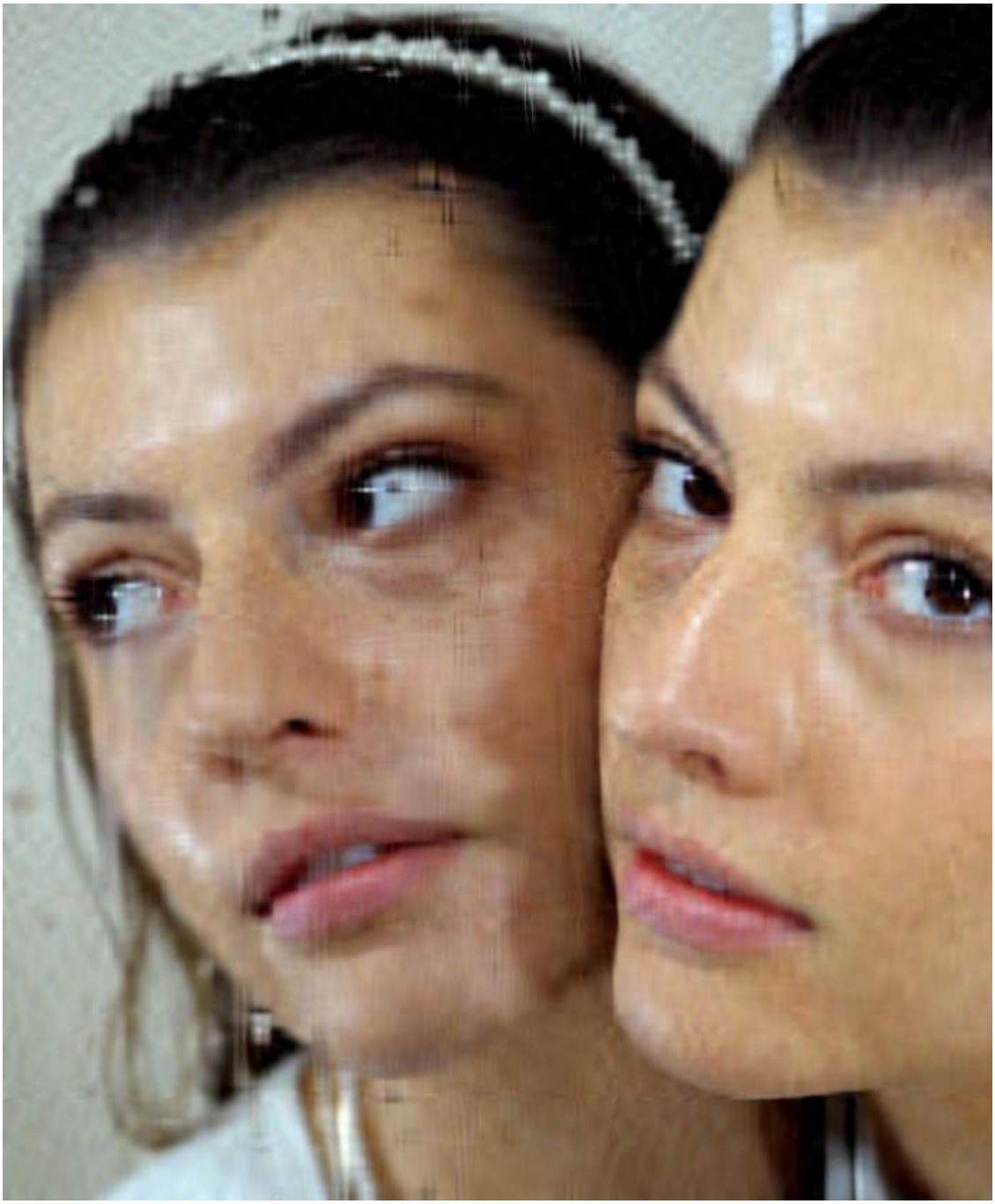
[ ]:



[86]:
```
im_reducer(img5, 5)
```

[86]:

[87]:
```
im_reducer(img5, 15)
```
[87]:



[88]:
```
im_reducer(img5, 25)
```
[88]:

[89]:
```
im_reducer(img5, 45)
```
[89]:

[91]:
```
im_reducer(img5, 75)
```

[91]:

[103]:
```
im_reducer(img5, 50)
```

[103]:

[106]:
```
im_reducer(img8, 50)
```

[106]:

[ ]:
```
im_reducer(img3, 10) # image loses most details, fewer similarities to original
```

[ ]:

[ ]:
```
im_reducer(img2, 10) # same thing
```

[ ]:

[115]:
```
im_reducer(img6, 10) # this one is still pretty easy to tell though
```

[115]: