Assignment 1

Chosen photo (963x640) for PCA analysis:

Since the dimension of the photo is a bit high, we decided to reduce it so that width and height are only 30% of the original image in order to speed up PCA computation and inverse transformation. We ended with a reduced image of size 289 x 192.

Here is the comparison of the reduced images with different number of eigenvalues:

(first 2 eigenvalues) (first 10 eigenvalues) (11^{th}-100^{th} eigenvalues)

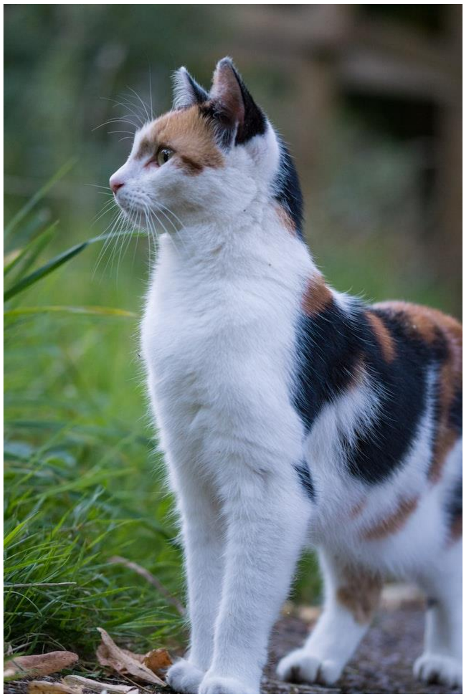
(first half (96) of the eigenvalues) (first 100 eigenvalues) (all of the eigenvalues)

We can see that when we have only 2 eigenvalues, the image reduction loses a lot of the necessary information. When there are 10 eigenvalues, we can see that the shape and color of the cat is preserved although the exact appearance of the cat is a bit unclear. Interestingly, when we consider 11^{th} through 100 eigenvalues, the shape of the cat is preserved, but the image loses body color of the cat almost completely. And we see that with images in the second row above, as the number of eigenvalues became close 100, the image became almost the same as the original image.

Appendix

Image Compression Using PCA

```
#!pip install opencv-python

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import cv2

img = cv2.cvtColor(cv2.imread('cat.jpg'), cv2.COLOR_BGR2RGB)
plt.imshow(img)
plt.show()
```



```
img.shape
```

```
(963, 640, 3)
```

```
# resize the image for faster computation of PCA components and inverse
tranformation
img = cv2.resize(img, None, fx = 0.3, fy = 0.3) # width and height 30 % of
the original values
plt.imshow(img)
plt.show()
```

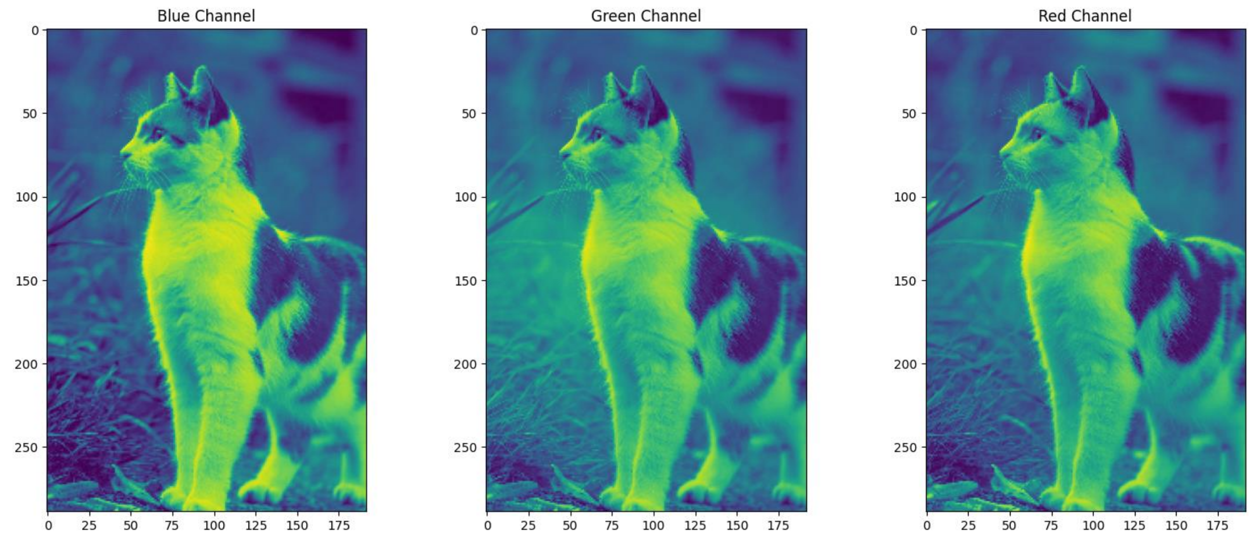
```
cv2.imwrite("resized_cat_image.png", cv2.cvtColor(img, cv2.COLOR_RGB2BGR))
```

```
True
```

```
img.shape
```

```
(289, 192, 3)
```

```
# Splitting into channels
red, green, blue = cv2.split(img)
# Plotting the images

fig = plt.figure(figsize = (18,7))

fig.add_subplot(131)
plt.title("Blue Channel")
plt.imshow(blue)

fig.add_subplot(132)
plt.title("Green Channel")
plt.imshow(green)

fig.add_subplot(133)
plt.title("Red Channel")
plt.imshow(red)

plt.show()
```

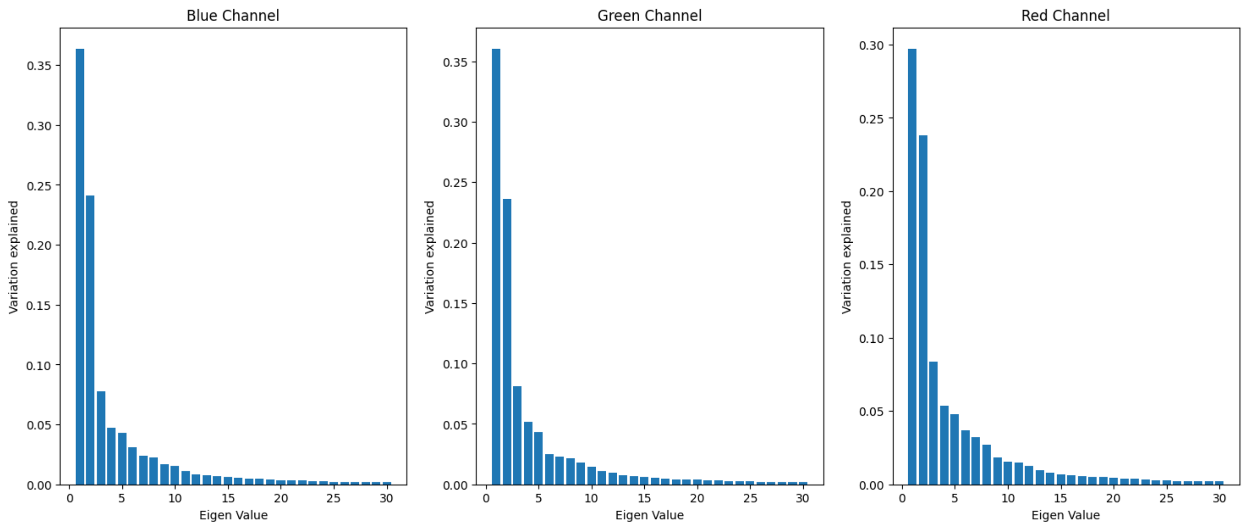



```
df_blue = blue / 255
df_green = green / 255
df_red = red / 255

pca_b = PCA()
pca_b.fit(df_blue)
trans_pca_b = pca_b.transform(df_blue)

pca_g = PCA()
pca_g.fit(df_green)
trans_pca_g = pca_g.transform(df_green)

pca_r = PCA()
pca_r.fit(df_red)
trans_pca_r = pca_r.transform(df_red)

sum(pca_b.explained_variance_ratio_[:10]) # variance explained by the first 10 components
```

```
0.8809956369501953
```

```
fig = plt.figure(figsize = (18, 7))
index_first_30 = list(range(1, 31))
fig.add_subplot(131)
plt.title("Blue Channel")
plt.ylabel('Variation explained')
plt.xlabel('Eigen Value')
plt.bar(index_first_30,pca_b.explained_variance_ratio_[:30])

fig.add_subplot(132)
plt.title("Green Channel")
plt.ylabel('Variation explained')
plt.xlabel('Eigen Value')
plt.bar(index_first_30,pca_g.explained_variance_ratio_[:30])

fig.add_subplot(133)
plt.title("Red Channel")
plt.ylabel('Variation explained')
```

```python
plt.xlabel('Eigen Value')
plt.bar(index_first_30, pca_r.explained_variance_ratio_[:30])
plt.show()
```




```python
print(f"Blue Channel : {sum(pca_b.explained_variance_ratio_[:10])}")
print(f"Green Channel: {sum(pca_g.explained_variance_ratio_[:10])}")
print(f"Red Channel   : {sum(pca_r.explained_variance_ratio_[:10])}")
```

```
Blue Channel : 0.8809956369501953
Green Channel: 0.8748197985751675
Red Channel   : 0.8502821548697636
```

```python
def reconstruct_image(num_pca_components, interval_end = None):

    transformed = [trans_pca_b,trans_pca_g, trans_pca_r]
    copy_transformed = []
    for channel_data in transformed:
        channel_data_copy = channel_data.copy()
        if interval_end is None:
            channel_data_copy[:, num_pca_components:] = 0
        else:
            channel_data_copy[:, :num_pca_components-1] = 0
            channel_data_copy[:, interval_end:] = 0
        copy_transformed.append(channel_data_copy)

    b_arr = pca_b.inverse_transform(copy_transformed[0])
    g_arr = pca_g.inverse_transform(copy_transformed[1])
    r_arr = pca_r.inverse_transform(copy_transformed[2])

    img_reduced= (cv2.merge((b_arr, g_arr, r_arr)))
    img_reduced *= 255 # return to original scale
    #plt.imshow(img_reduced)
    if interval_end is None:
cv2.imwrite(f"reduced_images/img_created_with_{num_pca_components}_pca_compon
ents.png", img_reduced)
    else:
        cv2.imwrite(f"reduced_images/img_created_with_{interval_end -
num_pca_components + 1}_pca_components.png", img_reduced)
```

```python
reconstruct_image(2) # two largest eigenvalues

reconstruct_image(10) # ten largest eigenvalues

reconstruct_image(11, 100) # 11th through 100th eigenvalues

reconstruct_image(100) # 100 largest eigenvalues

reconstruct_image(pca_b.components_.shape[0] // 2) # first half of the
eigenvalues

reconstruct_image(pca_b.components_.shape[0]) # all of the eigenvalues
```