Name: Pujan Bhusal

Id: 2060274

Image Compression Using Principal Component Analysis

PCA is a technique for reducing the dimensions of an image, such as an image, by transforming the data into a new coordinate system that highlights the most important features of the data and captures the image's most important patterns.

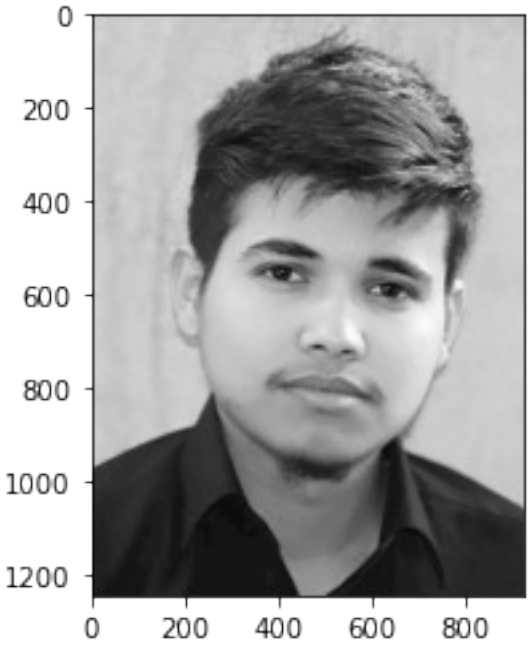
Load and Prepare the data

```
# importing liberary for computing, read image, visulaize
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

img1 = Image.open(r"1.jpg")
img1

# opening image
im = np.array(img1)
print('Pixel type:', im.dtype)
print('Number of dimensions:', im.ndim)
print('Image size:', im.shape)
```

```
Pixel type: uint8
Number of dimensions: 3
Image size: (1245, 927, 3)
```

```
#convert the image into gray scale i.e. black and white
gray_img = img1.convert("L")
plt.imshow(gray_img, cmap='gray')
data=np.array(gray_img)
print('Number of dimensions:', data.ndim)
```

```
Number of dimensions: 2
```



Standardize/Scale the data

```
# using epsilon avoid division by zero errors
epsilon = 1e-8
# standardizing the input data to have zero mean and unit variance
standardized_dataset = (data - np.average(data, axis=0)) /
(np.std(data, axis=0) + epsilon)
standardized_dataset
```

```
array([[ 0.5220618 ,  0.52457734,  0.52682477, ...,  0.19017349,
         0.18856936,  0.20625755],
       [ 0.5220618 ,  0.52457734,  0.52682477, ...,  0.20869109,
         0.20716785,  0.20625755],
       [ 0.5220618 ,  0.52457734,  0.52682477, ...,  0.20869109,
         0.22576635,  0.22488716],
       ...,
       [-1.975866  , -1.9747794 , -2.00204492, ..., -2.43932529,
        -2.43381895, -2.43914812],
       [-1.975866  , -1.9747794 , -2.00204492, ..., -2.43932529,
        -2.43381895, -2.43914812],
       [-1.975866  , -1.9747794 , -2.00204492, ..., -2.43932529,
        -2.43381895, -2.43914812]])
```

Calculate Covariance Matrix

```
#calculate_covariance that takes in a image X and returns the
covariance matrix of the image
def calculate_covariance(X):
    #stores the mean of each feature
  X_mean = np.mean(X, axis=0)
  n = X.shape[0]
```

```
  cov_mat = (X-X_mean).T.dot((X-X_mean))/(n - 1)
  return cov_mat
#cov_mat is the covariance matrix of the image
cov_mat = calculate_covariance(standardized_dataset)
cov_mat
```

```
array([[1.00080386, 1.00070523, 1.0004798 , ..., 0.84375597,
0.84294978,
        0.84190518],
       [1.00070523, 1.00080386, 1.00067954, ..., 0.84262161,
0.84181513,
        0.84076565],
       [1.0004798 , 1.00067954, 1.00080386, ..., 0.84216323,
0.84137638,
        0.84033615],
       ...,
       [0.84375597, 0.84262161, 0.84216323, ..., 1.00080386,
1.00067942,
        1.00040415],
       [0.84294978, 0.84181513, 0.84137638, ..., 1.00067942,
1.00080386,
        1.0006757 ],
       [0.84190518, 0.84076565, 0.84033615, ..., 1.00040415, 1.0006757
,
        1.00080386]])
```

```
# calculate Covarience
#calculate_covariance that takes in a numpy array X as an input
def calculate_covariance(X):
  X_mean = np.mean(X, axis=0)
  n = X.shape[0]
    #  calculates the covariance matrix of the input array using the
formula cov_mat = (X-X_mean).T.dot((X-X_mean))/(n - 1)
  cov_mat = (X-X_mean).T.dot((X-X_mean))/(n - 1)
  return cov_mat
cov_mat = calculate_covariance(standardized_dataset)
cov_mat
```

```
array([[1.00080386, 1.00070523, 1.0004798 , ..., 0.84375597,
0.84294978,
        0.84190518],
       [1.00070523, 1.00080386, 1.00067954, ..., 0.84262161,
0.84181513,
        0.84076565],
       [1.0004798 , 1.00067954, 1.00080386, ..., 0.84216323,
0.84137638,
        0.84033615],
       ...,
       [0.84375597, 0.84262161, 0.84216323, ..., 1.00080386,
1.00067942,
        1.00040415],
```

```
       [0.84294978, 0.84181513, 0.84137638, ..., 1.00067942,
1.00080386,
        1.0006757 ],
       [0.84190518, 0.84076565, 0.84033615, ..., 1.00040415, 1.0006757
,
        1.00080386]])
```

Eigen Decomposition

purpose of doing this is to how information is captured by each principal component.

```
# Eigen Decomposition

# performs eigen decomposition on the standardized image using the
np.linalg.eig function to obtain
#the eigenvalues and eigenvectors of the covariance matrix of the
image

def find_eigens(X):
  cov_mat = calculate_covariance(X)
  eig_vals, eig_vecs = np.linalg.eig(cov_mat)

  return eig_vals, eig_vecs

eigen_values, eigen_vectors = find_eigens(standardized_dataset)

eigen_vectors
```

```
array([[-0.03688273, -0.02888294,  0.02565125, ...,  0.04101371,
        -0.029924  , -0.01758994],
       [-0.03690723, -0.02875268,  0.02574349, ..., -0.04497501,
         0.02588895, -0.04924173],
       [-0.03693231, -0.02863314,  0.0258897 , ...,  0.03270903,
        -0.02556479,  0.05165132],
       ...,
       [-0.03309024, -0.02668702,  0.03205498, ...,  0.03713627,
        -0.02150279, -0.02980716],
       [-0.03306481, -0.02666604,  0.03202541, ..., -0.01484485,
        -0.02198605,  0.03238297],
       [-0.03303961, -0.02664056,  0.03195296, ..., -0.01334538,
         0.0140651 , -0.03822627]])
```

eigenvectors are sorted in descending order of their corresponding eigenvalues

```
eigen_values
```

```
array([5.63731078e+02, 1.62562034e+02, 4.26022192e+01, 3.40050576e+01,
       2.46698783e+01, 2.09287866e+01, 1.42756541e+01, 7.69232169e+00,
```

6.52152763e+00, 5.90486612e+00, 4.16511519e+00, 3.54012686e+00,
2.95130537e+00, 2.60421445e+00, 2.26711311e+00, 2.00181306e+00,
1.79463152e+00, 1.73618347e+00, 1.53817745e+00, 1.38940221e+00,
1.30555861e+00, 1.24466950e+00, 1.10715406e+00, 9.50725628e-01,
8.41623700e-01, 8.12318527e-01, 7.20819677e-01, 6.81041022e-01,
6.55303727e-01, 6.19079429e-01, 5.36451844e-01, 5.11687865e-01,
4.90554050e-01, 4.61597990e-01, 4.17297696e-01, 4.06250372e-01,
3.89070689e-01, 3.72441841e-01, 3.30530730e-01, 3.02583679e-01,
2.97003119e-01, 2.81997501e-01, 2.77059180e-01, 2.58010795e-01,
2.46028480e-01, 2.31791006e-01, 2.27517381e-01, 2.17706982e-01,
2.04314257e-01, 1.95329755e-01, 1.89167713e-01, 1.84205474e-01,
1.75006558e-01, 1.71688407e-01, 1.60205293e-01, 1.56017552e-01,
1.50775430e-01, 1.48838702e-01, 1.41870267e-01, 1.38318935e-01,
1.28278589e-01, 1.19929103e-01, 1.16014536e-01, 1.13504860e-01,
1.07445145e-01, 1.03015420e-01, 9.88677001e-02, 9.61382215e-02,
9.19224629e-02, 8.92590006e-02, 8.44727279e-02, 8.25239627e-02,
7.66442069e-02, 7.60299762e-02, 7.45183657e-02, 6.96019779e-02,
6.80289445e-02, 6.44570076e-02, 6.29108692e-02, 6.06350769e-02,
5.74479180e-02, 5.62171409e-02, 5.19320930e-02, 5.13361098e-02,
5.00203077e-02, 4.84503658e-02, 4.67438891e-02, 4.56971512e-02,
4.51142063e-02, 4.32097993e-02, 4.07014878e-02, 3.90824142e-02,
3.85222990e-02, 3.66968597e-02, 3.57614229e-02, 3.56037201e-02,
3.42239938e-02, 3.29713003e-02, 3.22630631e-02, 2.94023344e-02,
2.88652409e-02, 2.85365061e-02, 2.75427474e-02, 2.52817991e-02,
2.47642945e-02, 2.43463498e-02, 2.35941585e-02, 2.31273302e-02,
2.22227112e-02, 2.18269303e-02, 2.10183001e-02, 2.02655201e-02,
1.98625884e-02, 1.94639298e-02, 1.90284417e-02, 1.84881581e-02,
1.76761385e-02, 1.75515386e-02, 1.64826726e-02, 1.58686076e-02,
1.54169516e-02, 1.53270260e-02, 1.49049367e-02, 1.48287006e-02,
1.43349429e-02, 1.35007793e-02, 1.31455684e-02, 1.26104140e-02,
1.20730960e-02, 1.17650616e-02, 1.11824633e-02, 1.10575112e-02,
1.08516052e-02, 1.04349582e-02, 1.01024155e-02, 9.85107342e-03,
9.24486526e-03, 9.15048650e-03, 8.95446383e-03, 8.68002199e-03,
8.49233904e-03, 8.34848063e-03, 8.22968736e-03, 7.97952964e-03,
7.82731371e-03, 7.72713049e-03, 7.29068276e-03, 7.11732641e-03,
6.78284798e-03, 6.46731069e-03, 6.22437749e-03, 6.19875690e-03,
6.02562642e-03, 5.76819536e-03, 5.52463849e-03, 5.44628329e-03,
5.26325391e-03, 5.19436456e-03, 5.05316038e-03, 4.90982701e-03,
4.80649599e-03, 4.70118029e-03, 4.52522433e-03, 4.51427840e-03,
4.37982607e-03, 4.23549194e-03, 4.26003120e-03, 3.96808276e-03,
3.85993406e-03, 3.77999548e-03, 3.70613052e-03, 3.47069929e-03,
3.49859416e-03, 3.42816963e-03, 3.37111997e-03, 3.21417566e-03,
3.16964857e-03, 3.02463172e-03, 2.91692320e-03, 2.85304784e-03,
2.76420049e-03, 2.73616725e-03, 2.65747294e-03, 2.59877186e-03,
2.58319388e-03, 2.48468193e-03, 2.34361492e-03, 2.28504673e-03,
2.24840501e-03, 2.20748446e-03, 2.12698855e-03, 2.09042770e-03,
2.03992636e-03, 1.96772087e-03, 1.94290268e-03, 1.93094760e-03,
1.83952973e-03, 1.81514001e-03, 1.77965103e-03, 1.71237008e-03,
1.68526706e-03, 1.66058626e-03, 1.60251891e-03, 1.59093261e-03,
1.51638578e-03, 1.45856452e-03, 1.44999797e-03, 1.40443642e-03,

1.38398311e-03, 1.35719823e-03, 1.32616953e-03, 1.26450389e-03,
1.24246253e-03, 1.22381424e-03, 1.20139649e-03, 1.15595718e-03,
1.09387699e-03, 1.06924532e-03, 1.04464421e-03, 1.03574347e-03,
1.02523043e-03, 9.92931979e-04, 9.48293246e-04, 9.41135103e-04,
9.08443660e-04, 9.06898670e-04, 8.75907817e-04, 8.95810411e-04,
8.57794164e-04, 8.36010025e-04, 8.13388470e-04, 7.95957908e-04,
7.76387126e-04, 7.53692411e-04, 7.54619596e-04, 7.25356846e-04,
7.08786306e-04, 7.04763464e-04, 6.81799890e-04, 6.73757369e-04,
6.58218500e-04, 6.29381066e-04, 6.10690131e-04, 5.88926891e-04,
5.76852055e-04, 5.62827610e-04, 5.51842737e-04, 5.41693201e-04,
5.35475306e-04, 5.22476231e-04, 5.06705907e-04, 5.02964992e-04,
4.92872313e-04, 4.98959208e-04, 4.81115564e-04, 4.62368666e-04,
4.66925574e-04, 4.49893352e-04, 4.33195231e-04, 4.31938420e-04,
4.22354191e-04, 4.19233593e-04, 4.07918770e-04, 3.99486604e-04,
3.86958064e-04, 3.82411293e-04, 3.71141447e-04, 3.66606599e-04,
3.58925697e-04, 3.53754837e-04, 3.45166118e-04, 3.34496173e-04,
3.33542929e-04, 3.28022373e-04, 3.23050966e-04, 3.20056254e-04,
3.15080883e-04, 3.14284881e-04, 2.95313044e-04, 2.90798336e-04,
2.86993059e-04, 2.82826285e-04, 2.75260581e-04, 2.66998001e-04,
2.64247315e-04, 2.59535952e-04, 2.54674865e-04, 2.49035745e-04,
2.46831996e-04, 2.42853048e-04, 2.41120281e-04, 2.35960392e-04,
2.34502439e-04, 2.30930149e-04, 2.26540556e-04, 2.20599144e-04,
2.16836097e-04, 2.11114980e-04, 2.13613433e-04, 2.14141520e-04,
2.04992998e-04, 2.03766857e-04, 2.01309791e-04, 1.97741465e-04,
1.95823658e-04, 1.93925882e-04, 1.91965352e-04, 1.89767974e-04,
1.87417072e-04, 1.84627783e-04, 1.82135544e-04, 1.79922523e-04,
1.78183971e-04, 1.75112384e-04, 1.73885018e-04, 1.72257330e-04,
1.67546547e-04, 1.68895452e-04, 1.69281100e-04, 1.70142630e-04,
1.63606132e-04, 1.61977175e-04, 1.60749700e-04, 1.59802601e-04,
1.58498695e-04, 1.53988164e-04, 1.55230780e-04, 1.56189210e-04,
1.56038109e-04, 1.51547630e-04, 1.49056262e-04, 1.47785206e-04,
1.44082758e-04, 1.45425100e-04, 1.45216618e-04, 1.42596665e-04,
1.41359560e-04, 1.40703135e-04, 1.38961706e-04, 1.37611293e-04,
1.36435091e-04, 1.35375662e-04, 1.33993166e-04, 1.34658900e-04,
1.32300780e-04, 1.31561986e-04, 1.30711731e-04, 1.30219228e-04,
1.29170653e-04, 1.28262051e-04, 1.27986975e-04, 1.25812615e-04,
1.24546828e-04, 1.23737425e-04, 1.21871278e-04, 1.20183168e-04,
1.21049056e-04, 1.18923455e-04, 1.17810189e-04, 1.17496016e-04,
1.16275884e-04, 1.14887577e-04, 1.14658876e-04, 1.13867638e-04,
1.10043441e-04, 1.12783226e-04, 1.12098422e-04, 1.11799364e-04,
1.09256070e-04, 1.08461795e-04, 1.09141471e-04, 1.07509977e-04,
1.06369954e-04, 1.05906561e-04, 1.05222603e-04, 1.04461277e-04,
1.03716460e-04, 1.03029413e-04, 1.00711025e-04, 9.95034736e-05,
1.00260908e-04, 1.01670053e-04, 1.01664183e-04, 8.99165808e-05,
9.09884514e-05, 9.01744522e-05, 9.81419122e-05, 9.77460018e-05,
9.74020764e-05, 9.62678633e-05, 9.67736332e-05, 9.41795907e-05,
9.19236467e-05, 9.49446632e-05, 9.29092767e-05, 9.26803575e-05,
9.33553777e-05, 9.36070533e-05, 9.53781469e-05, 8.79556798e-05,
8.84227509e-05, 8.90053019e-05, 8.92187556e-05, 8.94604376e-05,
8.69207257e-05, 8.61032036e-05, 8.55841731e-05, 8.58321749e-05,

8.48187979e-05, 8.45613366e-05, 8.42439146e-05, 8.34489112e-05,
8.28704046e-05, 8.18684311e-05, 8.14684555e-05, 8.10991871e-05,
8.06500576e-05, 7.97295302e-05, 8.01497328e-05, 7.90155105e-05,
7.83168825e-05, 7.79450218e-05, 7.75316874e-05, 7.61426794e-05,
7.64378338e-05, 7.68672758e-05, 7.69185340e-05, 7.51828563e-05,
7.48800876e-05, 7.44780476e-05, 7.43490271e-05, 7.34142791e-05,
7.32249627e-05, 7.24860013e-05, 7.23211046e-05, 7.17789905e-05,
6.31570753e-05, 6.36945085e-05, 7.11216618e-05, 7.06744899e-05,
6.95092774e-05, 7.03189810e-05, 7.04376591e-05, 6.43173108e-05,
6.92448078e-05, 6.46760802e-05, 6.37445729e-05, 6.64354734e-05,
6.60196667e-05, 6.84823539e-05, 6.69692550e-05, 6.76029231e-05,
6.86082120e-05, 6.78074077e-05, 6.50627504e-05, 6.53144716e-05,
6.61033457e-05, 6.73766581e-05, 6.88368671e-05, 6.26475539e-05,
6.25499400e-05, 6.21316480e-05, 6.04918700e-05, 6.12546835e-05,
6.18379506e-05, 6.16598565e-05, 6.00357351e-05, 6.02216474e-05,
5.97419209e-05, 5.90415099e-05, 5.85934614e-05, 5.82917550e-05,
5.74798569e-05, 5.78738418e-05, 5.76881790e-05, 5.69996760e-05,
5.68705246e-05, 5.64688871e-05, 5.62917363e-05, 5.56928304e-05,
5.55825123e-05, 5.52663651e-05, 5.50570455e-05, 5.48382674e-05,
5.43218760e-05, 4.88804067e-05, 4.90549670e-05, 4.92794332e-05,
5.39862921e-05, 5.36905069e-05, 5.34423152e-05, 5.31909125e-05,
4.97051706e-05, 5.19631479e-05, 5.21648211e-05, 5.23594663e-05,
5.01147282e-05, 5.26217325e-05, 4.98576763e-05, 5.29321039e-05,
5.12124612e-05, 5.08485147e-05, 5.07766759e-05, 5.05258844e-05,
5.11346712e-05, 5.14850357e-05, 1.32495706e-06, 1.39776754e-06,
4.84850471e-05, 4.82204419e-05, 4.70758638e-05, 4.74291090e-05,
4.79580410e-05, 4.76860213e-05, 4.77716632e-05, 4.65805805e-05,
4.68598961e-05, 4.61718480e-05, 4.59677850e-05, 4.55609209e-05,
4.53542201e-05, 4.53909295e-05, 4.49010715e-05, 4.38332333e-05,
4.41870247e-05, 4.43682563e-05, 4.44431175e-05, 4.34695778e-05,
4.33886045e-05, 4.32384428e-05, 4.26463207e-05, 4.28824089e-05,
4.22528461e-05, 4.20143301e-05, 4.21294095e-05, 4.21017008e-05,
4.13254766e-05, 4.09311439e-05, 4.10790219e-05, 4.07249347e-05,
4.04642122e-05, 3.96843747e-05, 3.99429047e-05, 4.00554070e-05,
4.06403092e-05, 3.94379921e-05, 3.92218663e-05, 3.89097830e-05,
3.83103720e-05, 3.87592524e-05, 3.85318446e-05, 3.91610913e-05,
3.78848957e-05, 3.77479846e-05, 3.76426055e-05, 3.73144233e-05,
3.71064319e-05, 3.69642200e-05, 3.67357316e-05, 3.66088425e-05,
3.64784281e-05, 3.57098680e-05, 3.65275112e-05, 3.59546355e-05,
3.55733283e-05, 3.51162434e-05, 3.49069253e-05, 3.61028415e-05,
3.46794960e-05, 3.42504506e-05, 3.43835274e-05, 3.45170888e-05,
3.39010313e-05, 2.57955638e-05, 2.58838800e-05, 3.39283855e-05,
3.34210922e-05, 2.60999454e-05, 2.61814131e-05, 3.36297782e-05,
2.63165001e-05, 3.31838067e-05, 3.30951141e-05, 3.26429143e-05,
3.29918280e-05, 2.65551782e-05, 2.66828244e-05, 2.67708092e-05,
2.69523348e-05, 3.24821500e-05, 3.24668187e-05, 3.20826592e-05,
2.72110356e-05, 2.72761225e-05, 2.75214258e-05, 2.76289271e-05,
3.17716158e-05, 3.20315196e-05, 2.77743127e-05, 2.79837270e-05,
2.81788425e-05, 3.15975448e-05, 3.14174958e-05, 3.12858293e-05,
2.83983330e-05, 2.85062310e-05, 3.10321738e-05, 3.11448191e-05,

3.06302511e-05, 2.92934823e-05, 2.99591548e-05, 3.03333624e-05,
2.92151538e-05, 2.86866380e-05, 3.08460254e-05, 3.01125640e-05,
3.08760474e-05, 2.89636729e-05, 2.90145623e-05, 2.87709005e-05,
2.96553151e-05, 3.01914832e-05, 1.18696201e-06, 2.40132285e-05,
2.41841547e-05, 2.42701083e-05, 2.44830721e-05, 2.46053130e-05,
2.48361245e-05, 2.47809421e-05, 2.50574600e-05, 2.51254479e-05,
2.54857859e-05, 2.55246294e-05, 2.51744417e-05, 2.18399296e-05,
2.20207830e-05, 2.21169484e-05, 2.24017154e-05, 2.28306786e-05,
2.29866672e-05, 2.34295976e-05, 2.32899741e-05, 2.26217110e-05,
2.36610392e-05, 2.37952585e-05, 2.31078167e-05, 2.21644340e-05,
2.25421188e-05, 2.26527027e-05, 2.40337924e-05, 1.09629972e-06,
1.15178024e-06, 1.25813145e-06, 1.28558807e-06, 1.40559575e-06,
2.15779762e-05, 2.15318564e-05, 2.14285989e-05, 2.10637207e-05,
2.12130661e-05, 2.12667787e-05, 2.08895725e-05, 2.01369279e-05,
1.93925314e-05, 2.06660598e-05, 2.06155858e-05, 1.99874407e-05,
1.92062578e-05, 1.94842742e-05, 1.97341324e-05, 2.03569399e-05,
2.06501380e-05, 1.98839684e-05, 1.92262201e-05, 1.98137205e-05,
1.49862811e-06, 1.50715204e-06, 1.55027387e-06, 1.61715101e-06,
1.65334530e-06, 1.73221727e-06, 1.76017597e-06, 1.91348260e-05,
1.78850323e-06, 1.81438653e-06, 1.89427303e-05, 1.90397425e-05,
1.88590294e-05, 1.88670137e-06, 1.86930880e-05, 1.87500660e-05,
1.94502560e-06, 1.97885421e-06, 1.83834657e-05, 1.83461127e-05,
2.01076669e-06, 2.02747185e-06, 2.06646027e-06, 2.10647925e-06,
2.18789746e-06, 2.30761164e-06, 1.81091587e-05, 1.80109736e-05,
1.80513804e-05, 1.76936915e-05, 2.27798075e-06, 1.79169616e-05,
1.78042117e-05, 2.34991019e-06, 2.14553245e-06, 1.74991435e-05,
1.74358596e-05, 1.75720471e-05, 1.72284810e-05, 1.70413375e-05,
1.69621393e-05, 1.68632195e-05, 1.66581881e-05, 1.68387389e-05,
1.67977456e-05, 1.65646383e-05, 1.65878354e-05, 1.63801384e-05,
1.46002381e-05, 1.46900066e-05, 1.61756062e-05, 1.62410527e-05,
1.59877031e-05, 1.45896551e-05, 1.48818820e-05, 1.58380237e-05,
1.58080206e-05, 1.57315971e-05, 1.50633567e-05, 1.49976519e-05,
1.54697954e-05, 1.56329054e-05, 1.55451376e-05, 1.51893504e-05,
1.53920890e-05, 1.52163874e-05, 2.37858770e-06, 2.44352113e-06,
2.48801458e-06, 2.52360541e-06, 2.40036893e-06, 2.58386947e-06,
2.59708895e-06, 2.61242131e-06, 2.73362149e-06, 2.66142508e-06,
2.83390033e-06, 2.86981190e-06, 2.89857370e-06, 3.01963198e-06,
2.99478690e-06, 3.58970780e-06, 3.29979368e-06, 3.32428074e-06,
3.56210721e-06, 3.43007482e-06, 3.54532499e-06, 2.94573061e-06,
3.13796678e-06, 3.53057615e-06, 3.21481902e-06, 3.25921908e-06,
3.06752741e-06, 3.48212377e-06, 3.10921600e-06, 1.39697706e-05,
1.40498601e-05, 1.41210758e-05, 1.42166297e-05, 1.43441154e-05,
1.43739455e-05, 3.72577558e-06, 3.64622255e-06, 3.76406325e-06,
3.76079259e-06, 1.38342911e-05, 1.37650308e-05, 1.33793364e-05,
1.36242191e-05, 3.85956821e-06, 3.90390047e-06, 4.00567370e-06,
1.35722905e-05, 1.34249383e-05, 1.33276951e-05, 3.98079779e-06,
4.11046124e-06, 4.04824972e-06, 1.37287762e-05, 1.31465329e-05,
1.30730764e-05, 1.29804318e-05, 1.28317631e-05, 1.28020527e-05,
4.14435736e-06, 1.26735411e-05, 1.18347247e-05, 1.22475825e-05,
1.25605751e-05, 1.23575708e-05, 1.26205428e-05, 1.20362561e-05,

```
       4.22245755e-06, 4.26524300e-06, 4.30186824e-06, 1.19803997e-05,
       4.38478880e-06, 4.64395659e-06, 1.20032830e-05, 1.24231504e-05,
       4.50578230e-06, 4.40814302e-06, 4.55059246e-06, 4.70492075e-06,
       4.57453711e-06, 1.24422602e-05, 4.46435059e-06, 1.17708190e-05,
       1.17078175e-05, 1.16117677e-05, 1.14073067e-05, 1.13143595e-05,
       1.12509306e-05, 1.11742938e-05, 1.11377862e-05, 1.10204455e-05,
       4.77829086e-06, 1.09444483e-05, 1.08911903e-05, 1.07631615e-05,
       1.07029325e-05, 4.83427471e-06, 4.98354119e-06, 4.91153380e-06,
       4.93795906e-06, 5.02778627e-06, 1.05455609e-05, 5.07635904e-06,
       5.10933171e-06, 1.08498511e-05, 5.20194856e-06, 1.04792563e-05,
       5.24560058e-06, 5.28031283e-06, 5.55614780e-06, 5.46965646e-06,
       5.45062099e-06, 5.70991481e-06, 5.63131114e-06, 5.31674211e-06,
       5.38832686e-06, 5.65476346e-06, 5.95456879e-06, 5.98286572e-06,
       5.88343232e-06, 5.83706387e-06, 1.04299654e-05, 1.03126504e-05,
       1.02830229e-05, 1.01347999e-05, 1.01041716e-05, 9.99540000e-06,
       9.96041973e-06, 9.79048345e-06, 9.74503620e-06, 9.83852165e-06,
       6.12433626e-06, 6.10556039e-06, 9.66132684e-06, 9.53233604e-06,
       6.24720020e-06, 6.36803349e-06, 6.30319873e-06, 6.31833835e-06,
       9.49706040e-06, 9.37150156e-06, 6.44561711e-06, 6.45482712e-06,
       6.49644137e-06, 6.61384751e-06, 9.29713739e-06, 9.27251039e-06,
       9.23189972e-06, 9.09425518e-06, 6.68029596e-06, 9.02263591e-06,
       6.72156889e-06, 6.78291573e-06, 8.85134755e-06, 8.98177345e-06,
       8.96098492e-06, 8.74169979e-06, 6.88478504e-06, 6.92519355e-06,
       8.63899170e-06, 8.60093443e-06, 7.03010265e-06, 7.06426313e-06,
       7.10408439e-06, 8.48900238e-06, 8.38951759e-06, 8.28181591e-06,
       8.45693775e-06, 7.17475480e-06, 7.48211834e-06, 7.41170848e-06,
       7.24768131e-06, 7.30470191e-06, 7.20993069e-06, 7.57241187e-06,
       8.20916998e-06, 8.06592481e-06, 8.17633883e-06, 8.11327916e-06,
       7.98314731e-06, 7.62727158e-06, 7.67985478e-06, 7.72147255e-06,
       7.83700785e-06, 7.81876780e-06, 7.85931564e-06])
```

The eigenvalue corresponding to the first eigenvector indicates the amount of variance in the data explained by that eigenvector, the eigenvalue corresponding to the second eigenvector after accounting for the first eigenvector, and so on.

```
# calculates the percentage of explained variance for each eigenvalue,
and prints out the first ten values.
X = np.array(gray_img)

eigenvalues_sorted = np.sort(eigen_values)[::-1]

explained_variance = eigenvalues_sorted / eigenvalues_sorted.sum()
*100

print(explained_variance[:10])
```

```
[60.76356871 17.52227208  4.59201733  3.66534459  2.65912224
2.25587662
  1.53874733  0.82914165  0.70294385  0.636475  ]
```

the first value (60.76) represents the proportion of total variance in the original data that is explained by the first principal component. The second value (17.52) represents the proportion of total variance that is explained by the second principal component, and so on.

Eigenvectors represent the directions in which the data vary the most, and the corresponding eigenvalues represent the amount of variation in those directions

Identify Principal Components and Reconstruction of the image

calculating the total explained variance of the image for different numbers of principal components

n_components, it selects the first n eigenvectors and projects the standardized image onto the vectors to obtain a lower-dimensional representation of the data. Finally, reconstructs the original dataset from the projected data using the selected eigenvectors

```
n_components = [1, 20, 60]
for n in n_components:
    # variance explained by each principal component in descending
order
  explained_variance_n = explained_variance[:n]
  print("Number of components:", n)
  print("Explained variance:", explained_variance_n.sum())
```

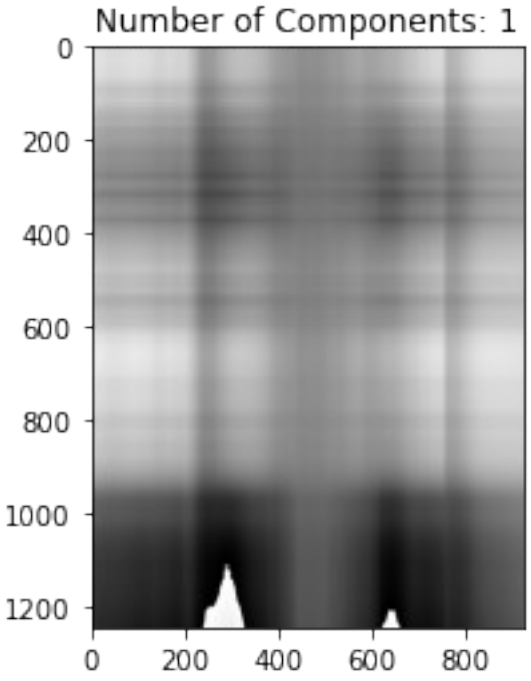
```
Number of components: 1
Explained variance: 60.76356870588485
Number of components: 20
Explained variance: 97.75114209765056
Number of components: 60
Explained variance: 99.60613567315633
```

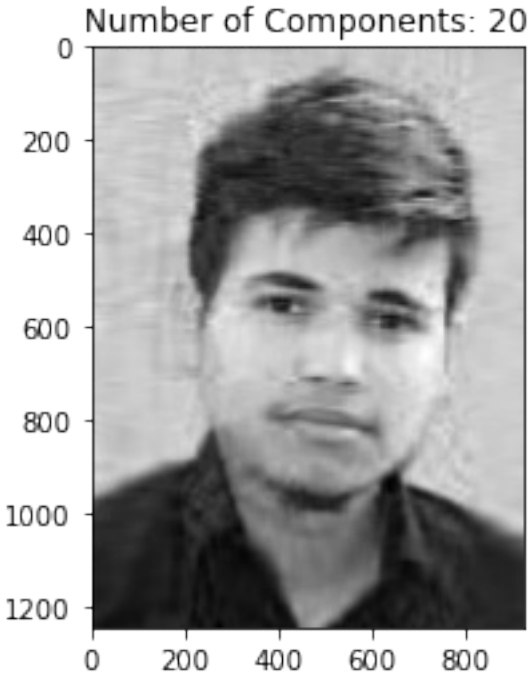
- It is important to accurately calculate the explained variance because it gives insight into how much information is retained by reducing the dimensionality of the data. Choosing the appropriate number of components can help avoid overfitting or underfitting the data, which can affect the performancance
- in our model, our PCA 1 explain around 60 % varience
- PCA 60 explain around 99.6 % varience

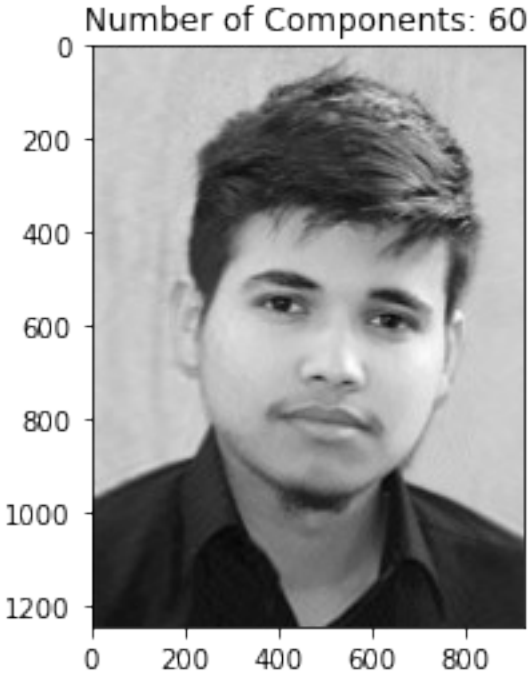
These values can be used to determine how many principal components to keep for a given level of variance retention. with principal components with the highest explained variances until their cumulative sum. we are going to use 3 principal components used.

```
# Identify priciple Component

for n in n_components:
    # Get the first n eigen vectors
    eigen_vectors_subset = eigen_vectors[:, :n]
```

```
    # Project standardized dataset onto the first n eigen vectors
    projected_data = standardized_dataset.dot(eigen_vectors_subset)

    # Reconstruct the original dataset from the projected data
    reconstructed_data = projected_data.dot(eigen_vectors_subset.T)

    # Rescale the data back to its original range
    reconstructed_data_rescaled = reconstructed_data * (np.std(data,
axis=0) + epsilon) + np.average(data, axis=0)

    # Convert reconstructed data to image
    reconstructed_data_rescaled = np.real(reconstructed_data_rescaled)
    reconstructed_img =
Image.fromarray(reconstructed_data_rescaled.astype(np.uint8),
mode='L')

    # Display the reconstructed image
    plt.figure()
    plt.imshow(reconstructed_img, cmap='gray')
    plt.title("Number of Components: {}".format(n))
    plt.show()
```







- the compressed data is generated by projecting the original data onto a lower-dimensional subspace, and the compressed data does not lie in the same range as the original data
- rescaling the data, it is converted back to an image using the Image.fromarray() function

- np.real() function is used to take the real part of the reconstructed data, which is necessary because the compression process can sometimes result in small imaginary components.
- reconstructed image is displayed using matplotlib.pyplot.imshow() function

here principal components 1 have less more directions of maximum variance so that it is blury with increase with principal components we get similar to the original image

It is important to get accurate results because the accuracy of the reconstructed image depends on the number of principal components used for compression. The more principal components used, the more accurate the reconstructed image will be. If the number of principal components used is too low, the reconstructed image will be too blurry and will lose important details. Conversely, if too many principal components are used, the reconstructed image will be too similar to the original image and there will be little compression achieved. Therefore, it is important to find the right balance between compression and accuracy.