P-Set #1

David Sileman AMATH 797 01/27/20

1. We want to project $X \in \mathbb{R}^{p \times n}$ onto a lower dimensional subspace $V \in \mathbb{R}^{p \times d}$. The optimal projection is the orthogonal projection (Gram-Schmidt) given by $V^T X$. Therefore, this implies $\boxed{\beta_k = V^T x_k}$

$$\min_{\substack{\mu, V, \beta \\ V^TV = I}} \sum_{k=1}^{n} \|x_k - \mu + V\beta_k\|_2^2 = \sum_{k=1}^{n} \|x_k - \mu + V(V^T x_k)\|_2^2$$

$$= \sum_{k=1}^{n} \|x_k\|^2 + \|\mu + VV^T x_k\|^2 - 2\langle x_k, \mu + VV^T x_k \rangle$$

$$= \sum_{k=1}^{n} \|x_k\|^2 + \|\mu + VV^T x_k\|^2 - 2\langle x_k, \mu \rangle - 2\langle x_k, VV^T x_k \rangle$$

$$= \sum_{k=1}^{n} \|x_k\|^2 + \|\mu\|^2 + \|VV^T x_k\|^2 + 2\langle \mu, VV^T x_k \rangle - 2\langle x_k, \mu \rangle - 2\langle x_k, VV^T x_k \rangle$$

Take partial to optimize

$$= \sum_{k=1}^{n} \frac{\partial}{\partial \mu}\left(\|\mu\|^2 + 2\langle \mu, VV^T x_k \rangle - 2\langle x_k, \mu \rangle + \|x_k\|^2 + \|VV^T x_k\|^2 - 2\langle x_k, VV^T x_k \rangle \right) = 0$$

$$= \sum_{k=1}^{n} 2\mu + 2VV^T x_k - 2x_k = 0$$

$$= \sum_{k=1}^{n} \mu = \sum_{k=1}^{n} (I - VV^T) x_k$$

$$= n\mu = (I - VV^T) \sum_{k=1}^{n} x_k$$

$$\Rightarrow \boxed{\mu = \frac{1}{n} \sum_{k=1}^{n} x_k}$$

; <u>Lemma</u>: $(I - VV^T)(I - VV^T) = I - 2VV^T + VV^TVV^T = I - VV^T$ $\therefore$ Idempotent $\Rightarrow$ psd $\Rightarrow$ convex

Thus choosing μ to be the sample mean is a valid solution, and due to convexity, all solutions are equivalent minima

$$\min_{\substack{\mu, V, \beta \\ V^TV = I}} \sum_{k=1}^{n} \|x_k - \left(\tfrac{1}{n}\sum_{k=1}^{n} x_k\right) + VV^T x_k\|_2^2 = \sum_{k=1}^{n} \|(x_k - \mu_n) + VV^T x_k\|_2^2$$

$$= \sum_{k=1}^{n} \|x_k - \mu_n\|^2 + \|VV^T x_k\|^2 - 2\langle x_k - \mu_n, VV^T x_k \rangle = \|x_k - \mu_n\| + (VV^T x_k)^T (VV^T x_k) - 2(x_k - \mu_n)^T (VV^T x_k)$$

$$= \sum_{k=1}^{n} \|x_k - \mu_n\|^2 - (x_k - \mu_n)^T VV^T (x_k - \mu_n) = \|x_k - \mu_n\|^2 - \mathrm{trace}\left(V^T (x_k - \mu_n)(x_k - \mu_n)^T V\right)$$

Minimize Quantity: Constant Maximizing this trace is equivalent to PCA.

(2) 

Clearly, the mean of the data scattered in the first quadrant is not the origin. Therefore, without being centered at the origin and having expectation 0, the covariance matrix of X:

$(X-\mathbb{E}X)(X-\mathbb{E}X)^T = XX^T$ iff $\mathbb{E}X = 0$

Thus, finding the eigenvectors of XX^T for the left U matrix of SVD is <u>NOT</u> equivalent to finding the eigenvectors of X's covariance matrix as is done in PCA.

(3) <u>Mean</u>: $\frac{1}{n}\sum_k^n \beta_i = \frac{1}{n}\sum_k^n U_i^T(X-\mu_s 1^T) = U_i^T \frac{1}{n}\sum_k^n \begin{bmatrix} | & & | \\ (x_1-\mu_s) & \cdots & (x_n-\mu_s) \\ | & & | \end{bmatrix}$

$= U_i^T \frac{1}{n}\begin{bmatrix} \sum_k^n (x_{1k}-\mu_s) \\ \vdots \\ \sum_k^n (x_{dk}-\mu_s)\end{bmatrix}$, By definition of sample mean $= U_i^T \frac{1}{n}\begin{bmatrix} - & 0_1 & - \\ & \vdots & \\ - & 0_d & - \end{bmatrix} = 0$

<u>Uncorrelated</u>: $\beta\beta^T = (U^T(X-\mu_s 1^T))(U^T(X-\mu_s 1^T))^T = U^T(X-\mu_s 1^T)(X-\mu_s 1^T)^T U$

$= U^T \Sigma_s U$

$= \frac{S^2}{n-1}$

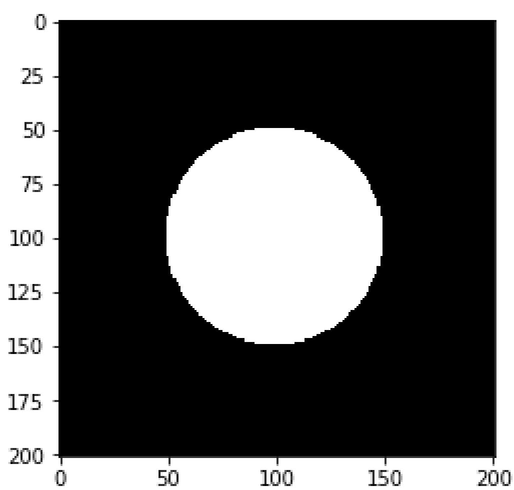
(I'm assuming U are principal directions, not principal components, or dimensions break)

and we know that the principal directions which are the eigenvectors of the covariance matrix diagonalize Σ. $\therefore$ the off-diagonals of the covariance matrix for β are all 0, so any pair of meta-features are uncorrelated.

In [1]:
```python
import numpy as np
import os
from numpy import linalg
from numpy.linalg import norm
import math
from PIL import Image
from sklearn.feature_extraction.image import extract_patches_2d
from sklearn.decomposition import PCA

import scipy
import scipy.io as sio
from scipy.io import loadmat

import matplotlib.pyplot as plt
from matplotlib import offsetbox
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.mplot3d import proj3d
%matplotlib inline

os.chdir(os.path.expanduser(os.sep.join(["~","Desktop","Homework Scans","2020S_AMATH797"
])))
np.random.seed(0)
```

In [2]:
```python
circle = np.array(Image.open("circle.png"))
#circle = circle.astype('float')/255
plt.imshow(circle, cmap = 'gray', vmin=0, vmax=255)
```

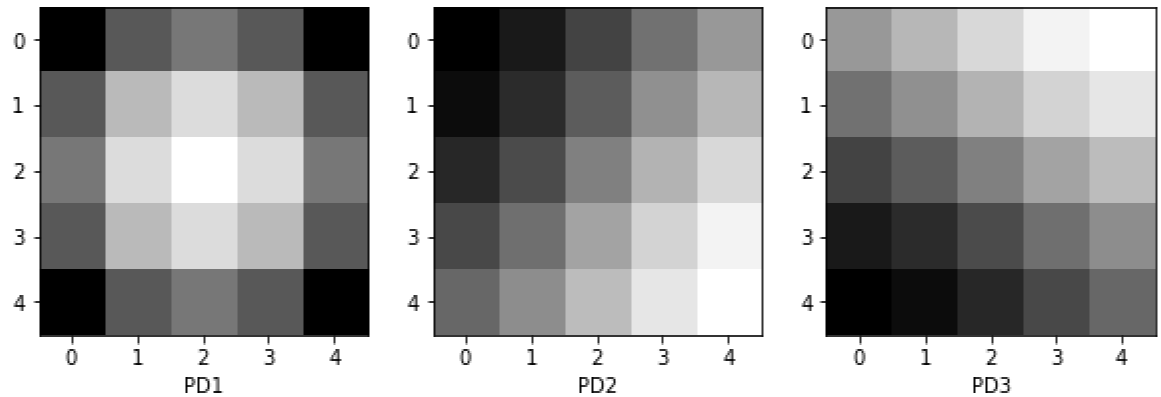
Out[2]: <matplotlib.image.AxesImage at 0x15cda3ec9c8>



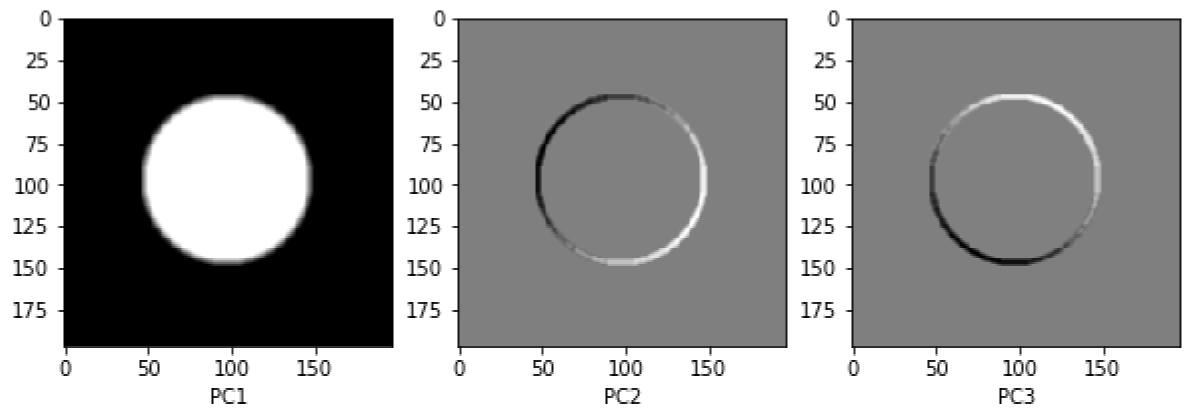
In [3]:
```python
patches = extract_patches_2d(circle, (5, 5))
patches = patches.reshape((38809, 25))
#patches_centered = patches - np.mean(patches[:], axis=0)

circle_pca = PCA(n_components = 3)
circle_pca.fit(patches)
print(circle_pca.explained_variance_ratio_)
```

```
[0.95940322 0.01202822 0.01202822]
```

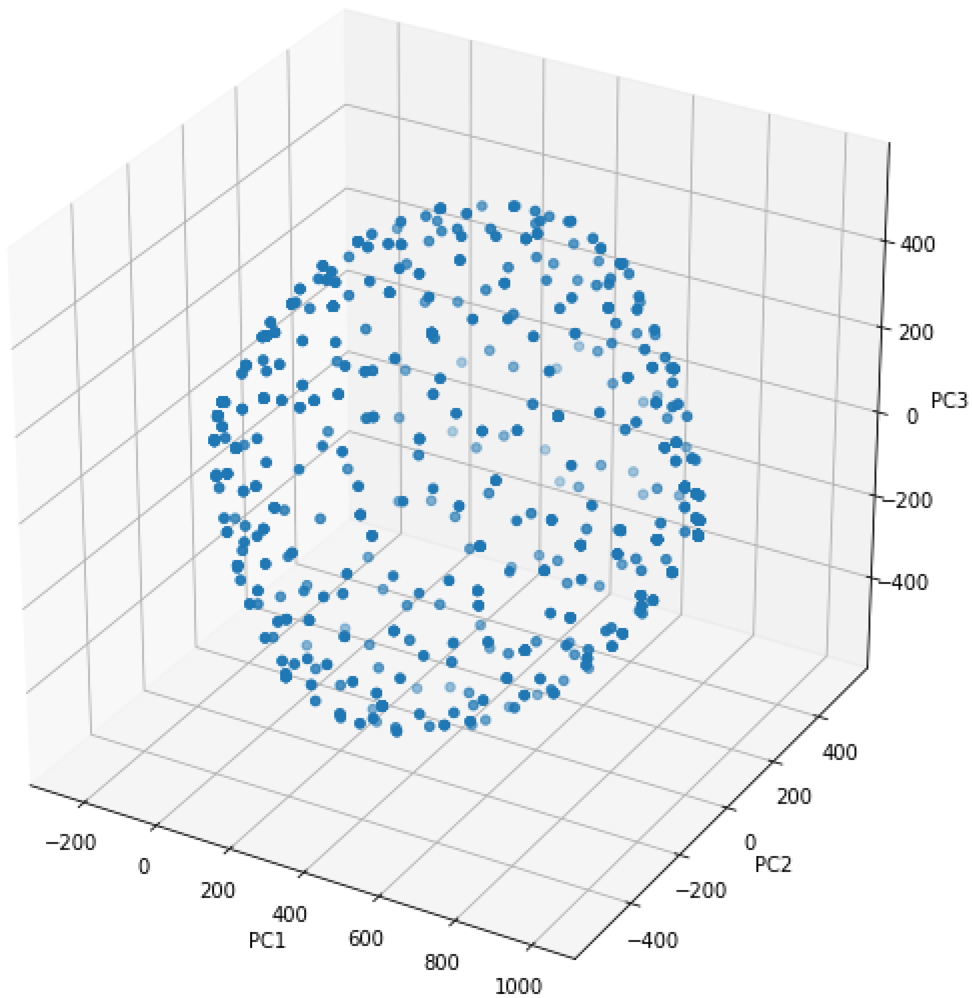
In [4]:
```python
PD = [np.reshape(circle_pca.components_[i], (5,5)) for i in range(3)]

fig = plt.figure(figsize=(10, 10))
for i in range(3):
    fig.add_subplot(1, 3, i + 1)
    plt.xlabel('PD' + str(i + 1))
    plt.imshow(PD[i], cmap = 'gray')
```



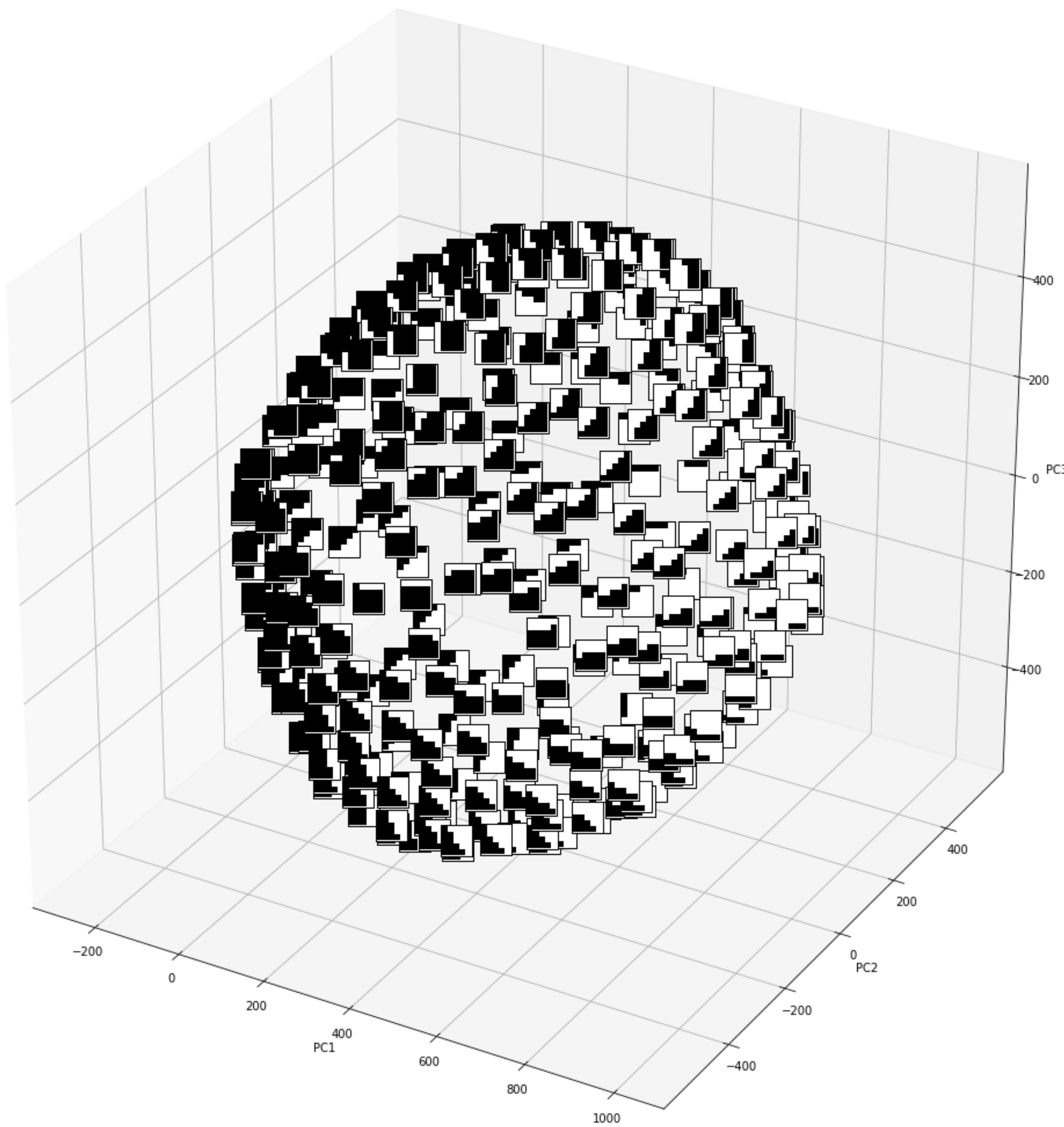
In [5]:
```python
PC = circle_pca.fit_transform(patches)

fig = plt.figure(figsize=(10, 10))
for i in range(3):
    fig.add_subplot(1, 3, i + 1)
    plt.xlabel('PC' + str(i + 1))
    plt.imshow(np.reshape(PC[:, i], ((197, 197))), cmap = 'gray')
```



In [6]:
```
fig = plt.figure(figsize=(10, 10))
ax = plt.axes(projection = '3d')
ax.scatter3D(PC[:, 0], PC[:, 1], PC[:, 2])
ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
ax.set_zlabel('PC3')
plt.show()
```



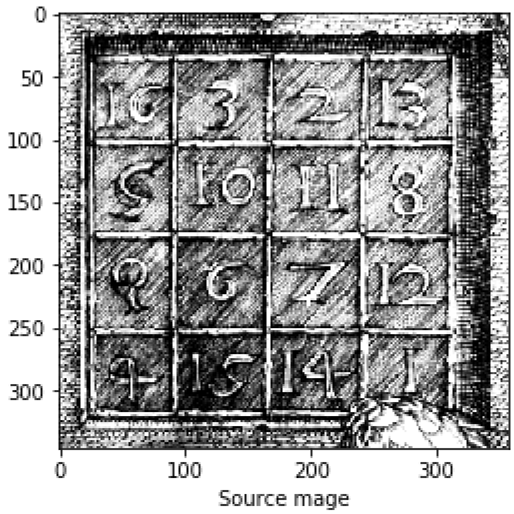
In [7]:
```
xs = PC[:, 0].flatten()
ys = PC[:, 1].flatten()
zs = PC[:, 2].flatten()

fig = plt.figure(figsize=(20, 20))
ax = fig.add_subplot(111, projection=Axes3D.name)
ax.scatter(xs, ys, zs)

ax2 = fig.add_subplot(111, frame_on=False)
ax2.axis("off")
ax2.axis([0,1,0,1])


def proj(X, ax1, ax2):
    """ From a 3D point in axes ax1,
        calculate position in 2D in ax2 """
    x,y,z = X
    x2, y2, _ = proj3d.proj_transform(x,y,z, ax1.get_proj())
    return ax2.transData.inverted().transform(ax1.transData.transform((x2, y2)))

def image(ax,arr,xy):
    """ Place an image (arr) as annotation at position xy """
    im = offsetbox.OffsetImage(arr, zoom = 5, cmap='gray', norm=plt.Normalize(0,255))
    im.image.axes = ax
    ab = offsetbox.AnnotationBbox(im, xy, xycoords = 'data', frameon = True, pad = 0.1)
    ax.add_artist(ab)


i = 0
for s in zip(xs,ys,zs):
    x,y = proj(s, ax, ax2)
    image(ax2, np.reshape(patches[i], ((5, 5))), [x,y])
    i += 1

ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
ax.set_zlabel('PC3')
plt.show()
```



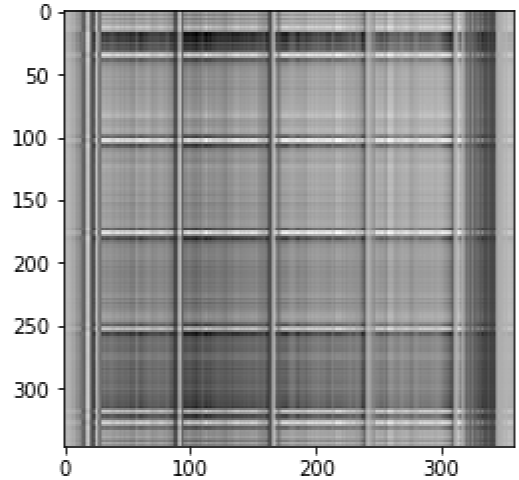
We observe that principal component 1 which captures roughly ~96% of the variance of the data seems to differentiate patches based on their color. Principal components 2 and 3, seem to capture directions for moving vertically and horiztonally (respectively), which are axises of symmetry for the circle. These results are exactly those which we would expect.

In [8]:
```
numbers = scipy.io.loadmat('numbers.mat')['mat']
numbers_mean = np.mean(numbers[:], axis=0)
plt.imshow(numbers, cmap = 'gray')
plt.xlabel('Source image')
numbers.shape
```

Out[8]: (346, 358)

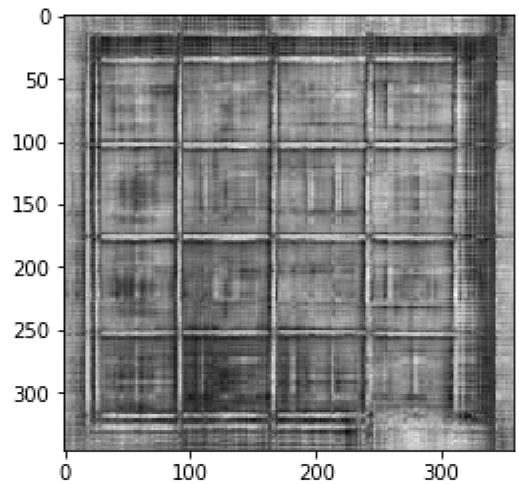


In [9]:
```
for i in np.array([1, 10, 20, 100]):
    numbers_pca = PCA(n_components = i)
    numbers_PC = numbers_pca.fit_transform(numbers)
    numbers_reconstruct = numbers_pca.inverse_transform(numbers_PC)
    plt.imshow(numbers_reconstruct, cmap = 'gray')
    plt.xlabel('Dimensions = ' + str(i))
    plt.show()
    print("Component-wise Percentage Total Variance:", numbers_pca.explained_variance_ra
tio_[:10])
    print("Reconstruction Error:", ((numbers - numbers_reconstruct) ** 2).sum())
    print("Compression Rate:", (346 * 358) / (346 * i + i + i * 358))
```



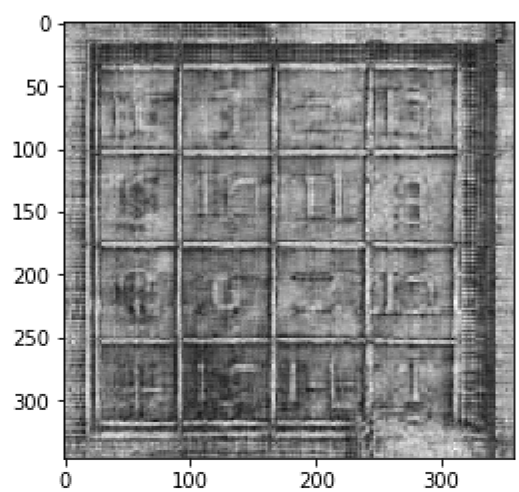
Dimensions = 1

```
Component-wise Percentage Total Variance: [0.22591042]
Reconstruction Error: 8690.909202650431
Compression Rate: 175.69929078014184
```



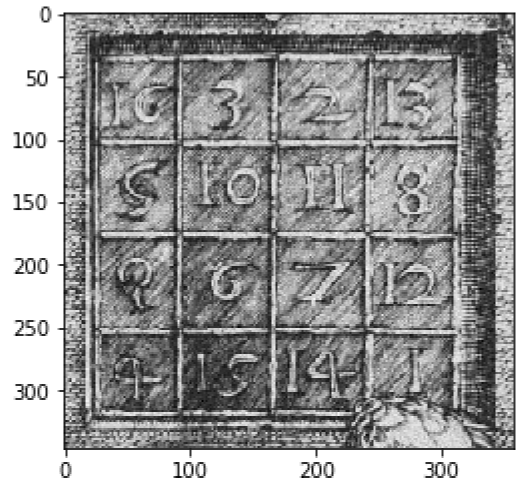
Dimensions = 10

```
Component-wise Percentage Total Variance: [0.22591042 0.0599177  0.04909906 0.03056869
0.02494228 0.02450543
 0.01909028 0.01847125 0.01619839 0.01573313]
Reconstruction Error: 5788.366782829698
Compression Rate: 17.569929078014184
```



Dimensions = 20

```
Component-wise Percentage Total Variance: [0.22591042 0.0599177  0.04909906 0.03056869
0.02494228 0.02450543
 0.01909028 0.01847125 0.01619839 0.01573313]
Reconstruction Error: 4501.927216110859
Compression Rate: 8.784964539007092
```



Dimensions = 100

```
Component-wise Percentage Total Variance: [0.22591042 0.0599177  0.04909906 0.03056869
0.02494228 0.02450543
 0.01909028 0.01847125 0.01619839 0.01573313]
Reconstruction Error: 962.002224382447
Compression Rate: 1.7569929078014184
```