Lab4

July 30, 2020

0.0.1 Machine Learning Lab 4 - 29/07/2020 | Prasenjit Dey (1947114)

Demonstrate PCA for Dimensionality Reduction on Image

Principal Component Analysis PCA is a statistical procedure to convert observations of possibly correlated variables/features into 'Principal Components' that are – 1. Uncorrelated with/independent of each other 2. Constructed to capture maximum information/variance in the data 3. Linear combinations of the original variables

Using PCA for image compression An image can be treated as a matrix - a grid of pixels, with values being the pixel intensities. The basic steps to be followed:

1. Apply PCA on the image matrix to reduce the dimensionality to a smaller number of principal components (PCs).
2. This is lossy compression, as we are discarding some of the information.
3. To assess how much visual information we retained, we'll reconstruct the image from the limited number of PC.
4. We'll see how good the reconstructed images are for different number of selecte components.

[72]:
```
# Necessary Imports
# ======================================================

import numpy as np
from matplotlib.image import imread
import matplotlib.pyplot as plt
```

[73]:
```
from google.colab import drive
drive.mount("/content/drive")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[74]:
```
!unzip '/content/drive/My Drive/Birds-Test.zip'
```

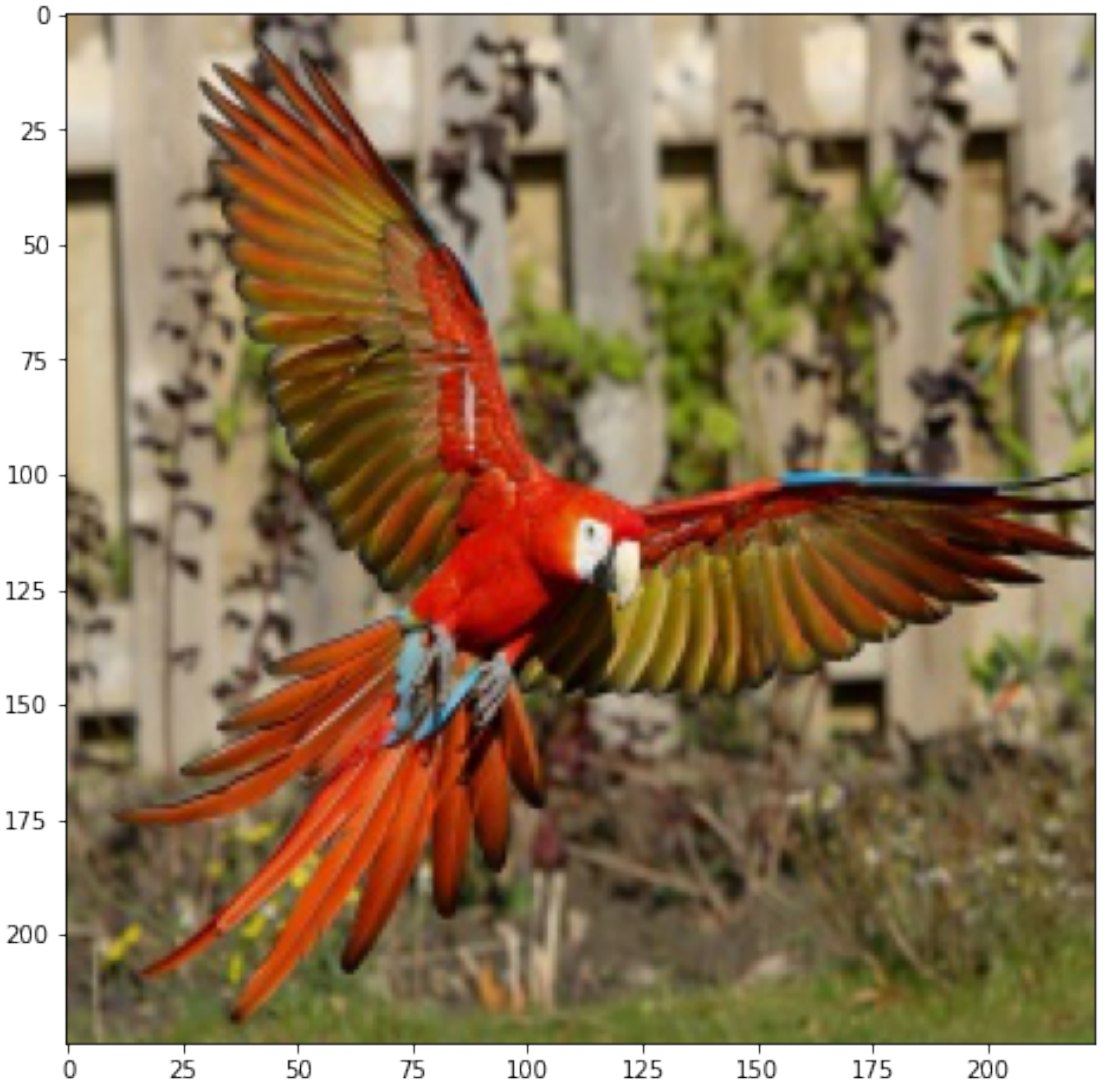
```
Archive:  /content/drive/My Drive/Birds-Test.zip
replace Birds-Test/001_3.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename: y
  inflating: Birds-Test/001_3.jpg
```

```
replace Birds-Test/005_2.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename: a
error:  invalid response [a]
replace Birds-Test/005_2.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename: A
  inflating: Birds-Test/005_2.jpg
  inflating: Birds-Test/007_4.jpg
  inflating: Birds-Test/020_1.jpg
  inflating: Birds-Test/023_4.jpg
  inflating: Birds-Test/024_4.jpg
  inflating: Birds-Test/027_2.jpg
  inflating: Birds-Test/028_1.jpg
  inflating: Birds-Test/030_3.jpg
  inflating: Birds-Test/036_1.jpg
  inflating: Birds-Test/041_2.jpg
  inflating: Birds-Test/044_2.jpg
  inflating: Birds-Test/049_3.jpg
  inflating: Birds-Test/050_4.jpg
  inflating: Birds-Test/052_3.jpg
  inflating: Birds-Test/053_1.jpg
```

[76]:
```
# Reading the Image
# ====================================================

image_raw = imread("/content/Birds-Test/001_3.jpg")
print(image_raw.shape)

# Displaying the image
# ====================================================

plt.figure(figsize=[12,8])
plt.imshow(image_raw)
```

```
(224, 224, 3)
```

[76]:
```
<matplotlib.image.AxesImage at 0x7f2e09ecd9e8>
```



About the image -

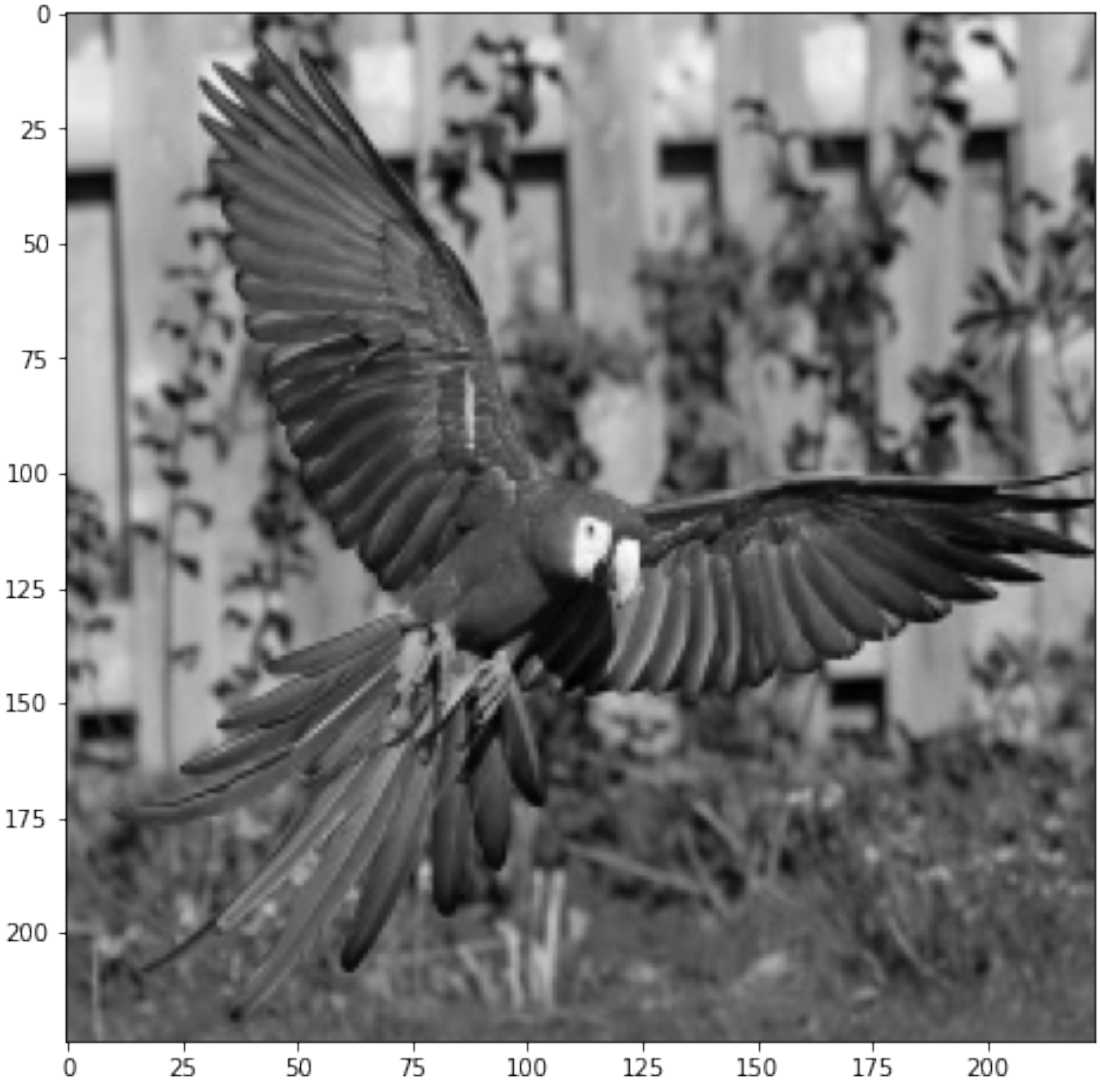
The image is a colour image i.e. has data in 3 channels - Red, Green, Blue. Hence the shape of the data - 224 x 224 x 3. It is essentially 224 x 224 matrix for each channel. We need to convert it to grayscale for the sake of simplicity

[77]:
```
# To Convert the image into Grayscale
# =====================================================
# Summing RGBs channel values for each pixel
# Capping values to 1
# =====================================================

image_sum = image_raw.sum(axis=2)
print(image_sum.shape)

image_bw = image_sum/image_sum.max()
```

```
print(image_bw.max())

plt.figure(figsize=[12,8])
plt.imshow(image_bw, cmap=plt.cm.gray)
```

```
(224, 224)
1.0
```

[77]: `<matplotlib.image.AxesImage at 0x7f2df9e85b70>`



0.0.2 Performing PCA on the image

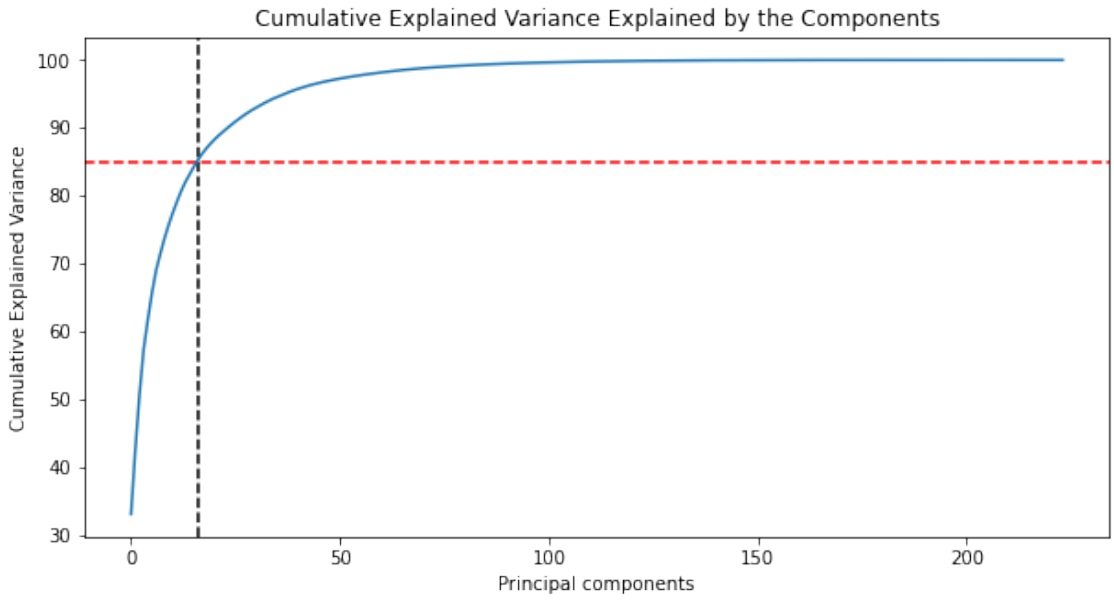
1. We'll perform PCA on the matrix with all the components

2. We'll then look at the scree-plot to assess how many components we could retain and how much cumulative variance they capture

3. We'll pick a suitable number of components to represent the image for compression

[81]:
```
from sklearn.decomposition import PCA, IncrementalPCA
pca = PCA()
pca.fit(image_bw)

# Getting the cumulative variance
# ================================================

var_cumu = np.cumsum(pca.explained_variance_ratio_)*100

# How many PCs explain 85% of the variance?
# ================================================

k = np.argmax(var_cumu>85)
print("Number of components explaining 85% Variance: "+ str(k))

plt.figure(figsize=[10,5])
plt.title('Cumulative Explained Variance Explained by the Components')
plt.ylabel('Cumulative Explained Variance')
plt.xlabel('Principal components')
plt.axvline(x=k, color="k", linestyle="--")
plt.axhline(y=85, color="r", linestyle="--")
ax = plt.plot(var_cumu)
```

```
Number of components explaining 85% Variance: 16
```



It is found that "16" components, instead of 224 pixels, can explain 85% of the variance in the image

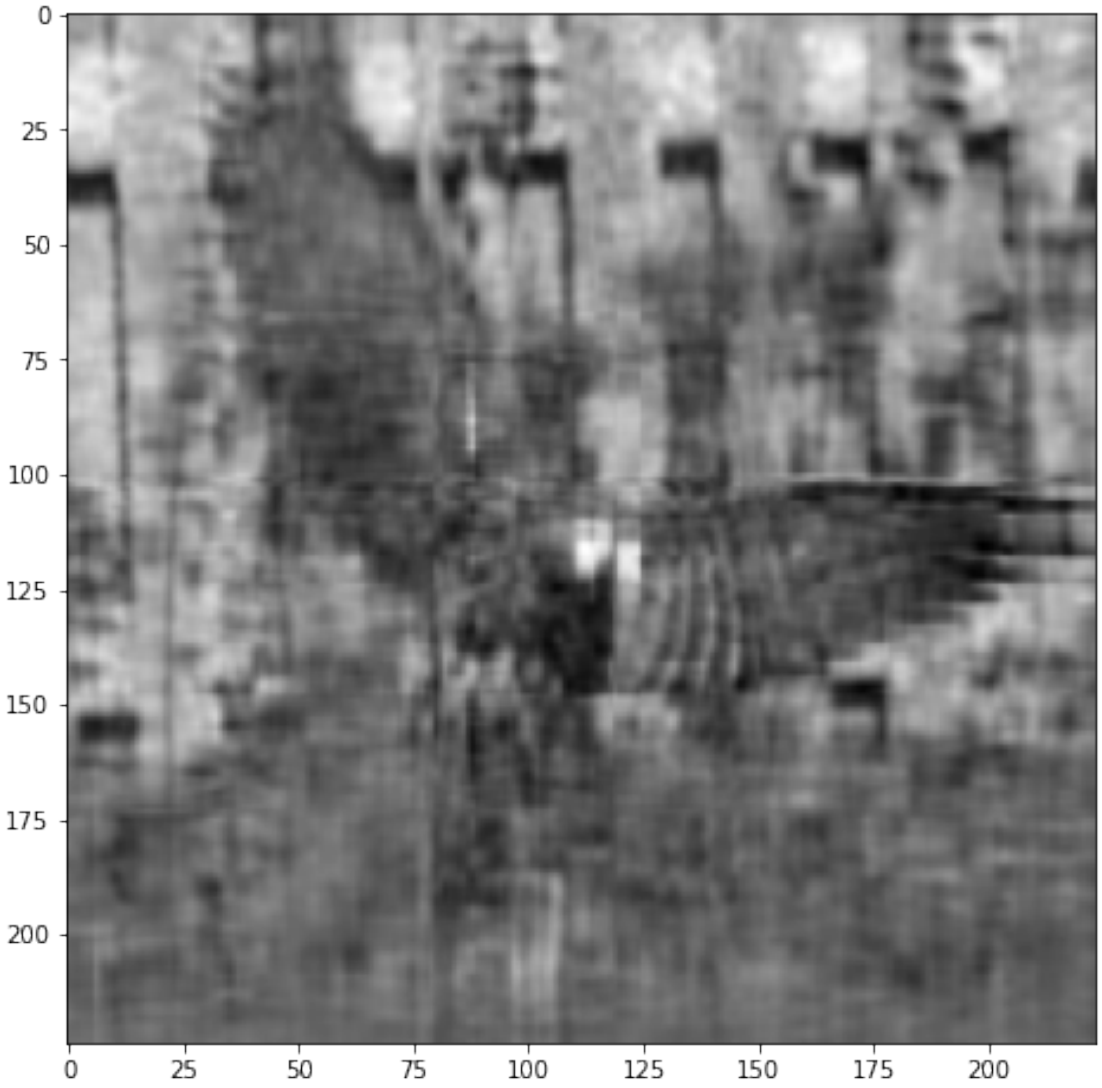
We can reconstruct the image using only 16 components and see if the reconstructed image is visually very different from the original.

0.0.3 Reconstructing the b/w image with the limited number of components i.e 16 -

1. We'll use the "fit_transform" method from the IncrementalPCA module
2. To first find the 16 PCs and transform and represent the data in those 16 new components/columns.
3. Next, we'll reconstruct the original matrix from these 16 components using the "inverse_transform".
4. Then plot the image to visually to check the quality of it.

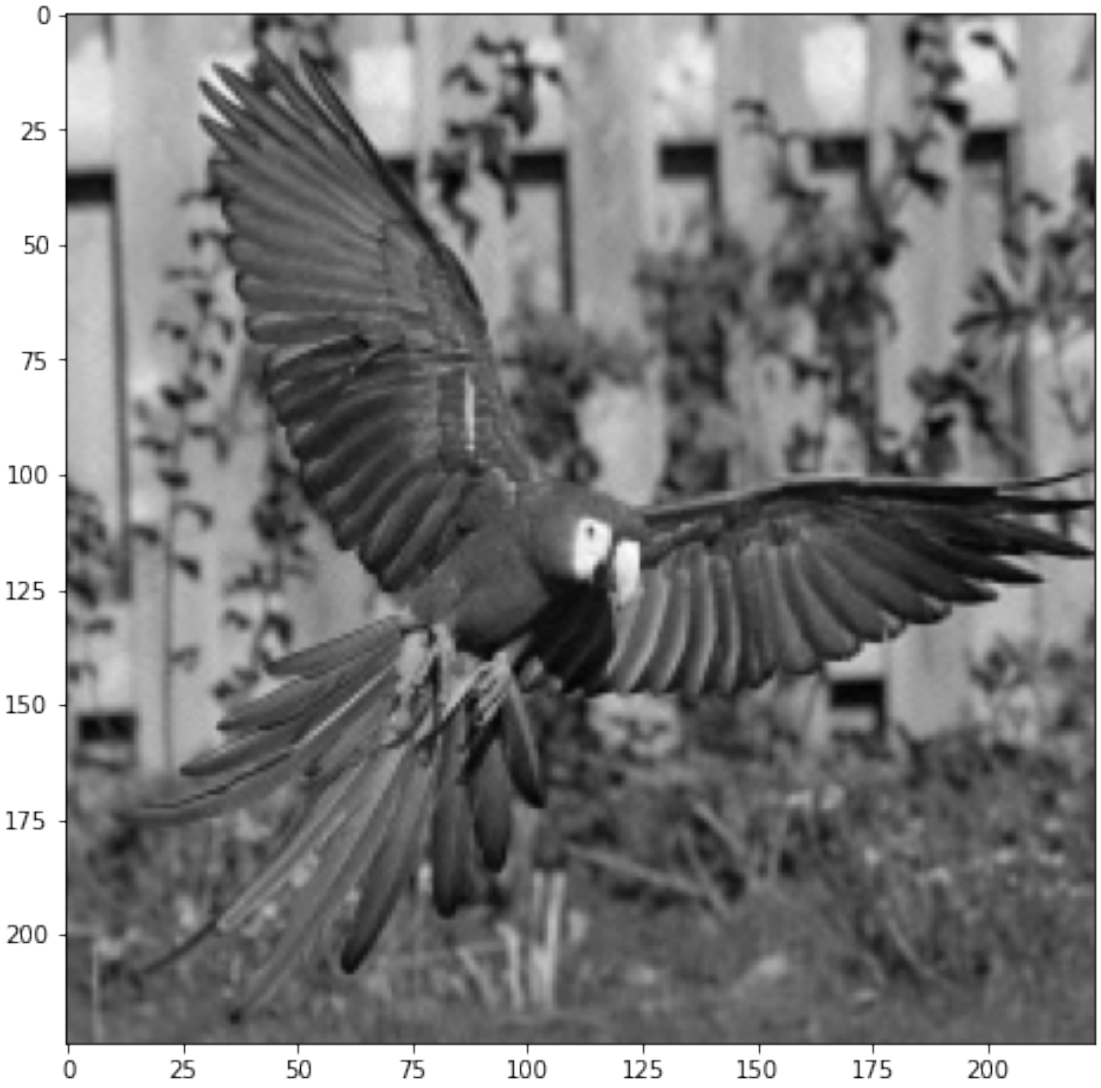
[80]:
```
ipca = IncrementalPCA(n_components=k)
image_recon = ipca.inverse_transform(ipca.fit_transform(image_bw))

# Plotting the reconstructed image
plt.figure(figsize=[12,8])
plt.imshow(image_recon,cmap = plt.cm.gray)
```

[80]:
```
<matplotlib.image.AxesImage at 0x7f2e09b3fb70>
```



All the major details are captured but we can see the lack of clarity. Let's try out a different value of k, say k = 32 components

[82]:
```
# Function to reconstruct and plot image for a given number of components

def plot_at_k(k):
    ipca = IncrementalPCA(n_components=k)
    image_recon = ipca.inverse_transform(ipca.fit_transform(image_bw))
    plt.imshow(image_recon,cmap = plt.cm.gray)


k = 32
plt.figure(figsize=[12,8])
plot_at_k(100)
```



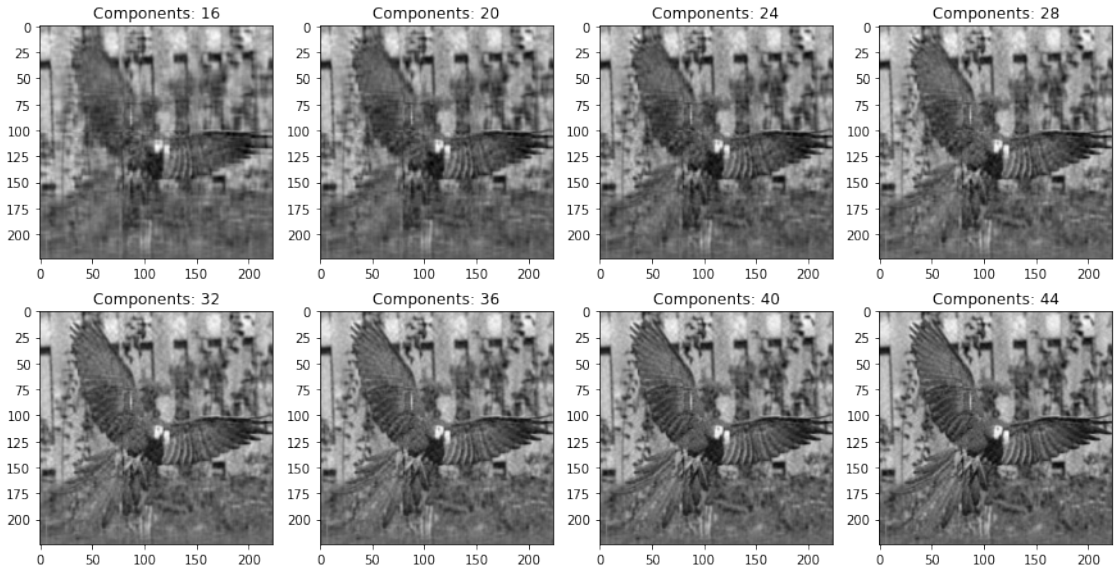
We get better result than previous but still a bit grainy image.

0.0.4 Reconstructing and plotting for different number of components

1. We'll try out different number of components, begining from 16, ending at 44
2. we'll reconstruct the image at each k and plot the images

[84]:
```
ks = [16, 20, 24, 28, 32, 36, 40, 44]

plt.figure(figsize=[15,7.5])

for i in range(8):
    plt.subplot(2,4,i+1)
    plot_at_k(ks[i])
```

```
    plt.title("Components: "+str(ks[i]))

plt.subplots_adjust(wspace=0.2, hspace=0.2)
plt.show()
```



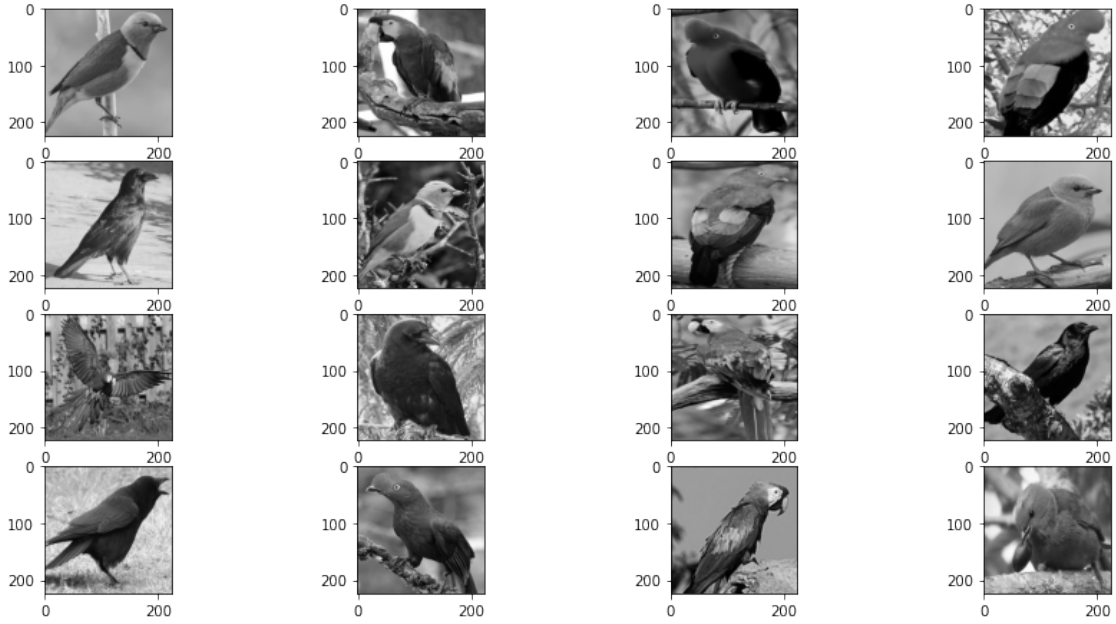
We'll looks like after 32, most of the reconstructed images are indistinguishable from the original greyscale. Even 32 components instead of 224 columns is significant compression.

Hence, the dimensionality reduction has taken place with the help of PCA

0.0.5 Creating PCA for a dataset of bird images

1. PCA tried for 95% and 99% variance capture
2. Se

[13]:
```
from glob import iglob
from PIL import Image
plt.figure(figsize=[15,7.5])
i=1
for path in iglob('/content/Birds-Test/*.jpg'):
 img = Image.open(path).convert('LA')
 f_name=path.split('/')[-1].split('.')[0]
 img.save('/content/Birds-Test/'+f_name+'.png')
 plt.subplot(4,4,i)
 plt.imshow(img)
 i+=1
```



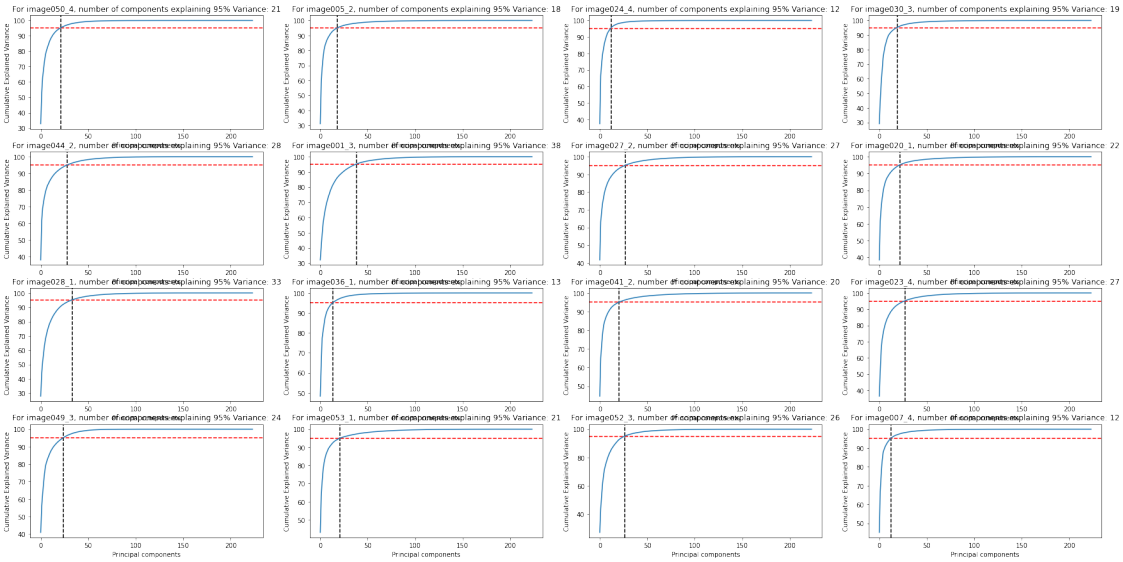
[62]:
```
from glob import iglob
import pandas as pd
birds = pd.DataFrame([])
for path in iglob('/content/Birds-Test/*.png'):
 img=imread(path)
 img=img[:,:,0:3]
 img1=img.sum(axis=2)
 img = img1 /img1.max()
 bird = pd.Series(img.flatten(),name=path)
 birds = birds.append(bird)
 print(img.shape)

fig, axes = plt.subplots(4,4,figsize=(9,9),
 subplot_kw={'xticks':[], 'yticks':[]},
 gridspec_kw=dict(hspace=0.01, wspace=0.01))
for i, ax in enumerate(axes.flat):
 ax.imshow(birds.iloc[i].values.reshape(224,224),cmap="gray")
```

```
(224, 224)
(224, 224)
(224, 224)
(224, 224)
(224, 224)
(224, 224)
(224, 224)
(224, 224)
(224, 224)
```

```
(224, 224)
(224, 224)
(224, 224)
(224, 224)
(224, 224)
(224, 224)
(224, 224)
```

[63]:
```
from sklearn.decomposition import PCA, IncrementalPCA
from glob import iglob
i=1
plt.figure(figsize=[30,15])
comps=[]
```

```
for path in iglob('/content/Birds-Test/*.png'):
  img=imread(path)
  img=img[:,:,0:3]
  img1=img.sum(axis=2)
  img = img1 /img1.max()
  f_name=path.split('/')[-1].split('.')[0]
  pca = PCA()
  pca.fit(img)

  # Getting the cumulative variance
  # ================================================

  var_cumu = np.cumsum(pca.explained_variance_ratio_)*100

  # How many PCs explain 95% of the variance?
  # ================================================

  k = np.argmax(var_cumu>95)
  #print("Number of components explaining 95% Variance: "+ str(k))
  plt.subplot(4,4,i)
  plt.title("For image"+f_name+", number of components explaining 95% Variance:␣
↪"+ str(k))
  plt.ylabel('Cumulative Explained Variance')
  plt.xlabel('Principal components')
  plt.axvline(x=k, color="k", linestyle="--")
  plt.axhline(y=95, color="r", linestyle="--")
  ax = plt.plot(var_cumu)
  i+=1
  comps.append(k)
```

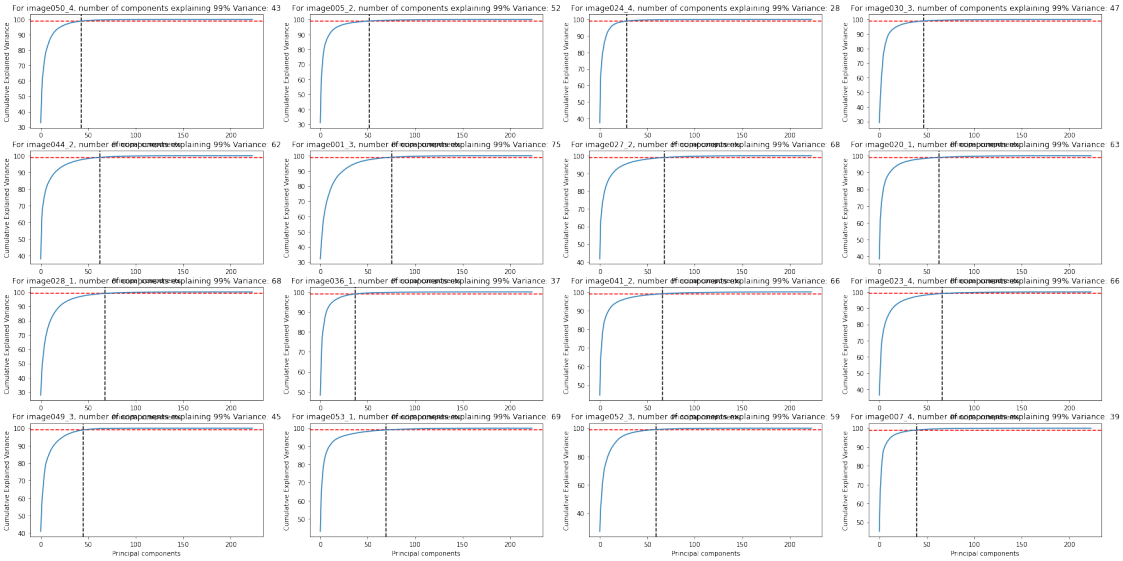


[65]:
```
from sklearn.decomposition import PCA, IncrementalPCA
from glob import iglob
i=1
plt.figure(figsize=[30,15])

for path in iglob('/content/Birds-Test/*.png'):
  img=imread(path)
  img=img[:,:,0:3]
  img1=img.sum(axis=2)
  img = img1 /img1.max()
  f_name=path.split('/')[-1].split('.')[0]
  ipca = IncrementalPCA(n_components=comps[i-1])
  image_recon = ipca.inverse_transform(ipca.fit_transform(img))

  plt.subplot(4,4,i)
  plt.title("For image"+f_name)
  plt.imshow(image_recon,cmap = plt.cm.gray)
  i+=1

v_comp=sum(comps)
print("Ratio of compression: ",str((16*224)/v_comp)[:6])
```

```
Ratio of compression:  9.9279
```

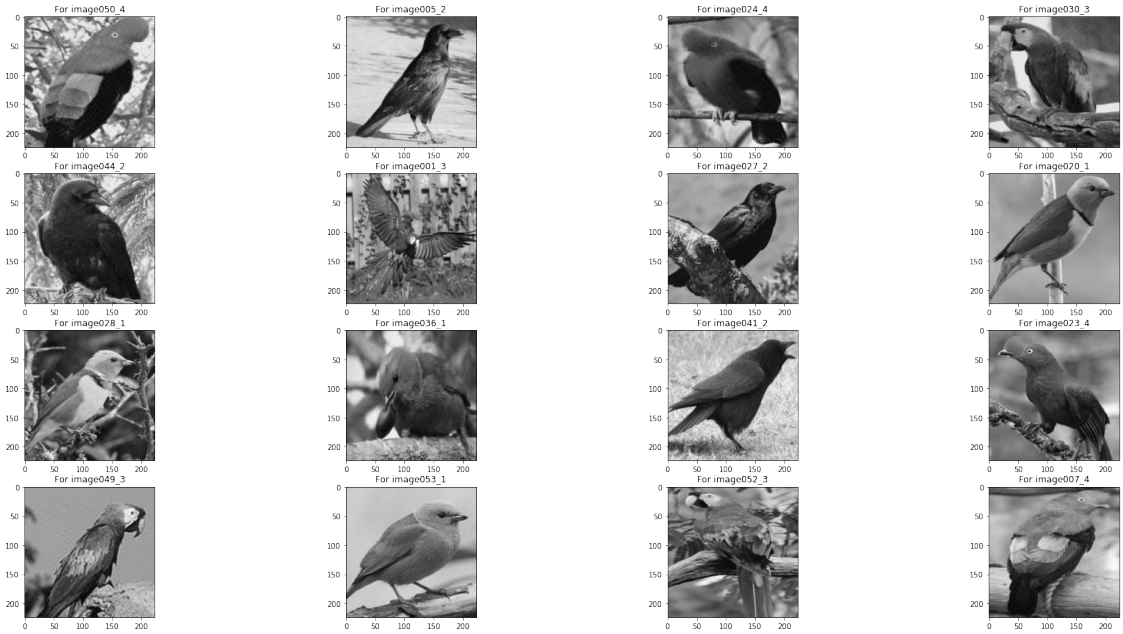
[71]:
```
from sklearn.decomposition import PCA, IncrementalPCA
from glob import iglob
```

```
i=1
plt.figure(figsize=[30,15])
comps1=[]

for path in iglob('/content/Birds-Test/*.png'):
  img=imread(path)
  img=img[:,:,0:3]
  img1=img.sum(axis=2)
  img = img1 /img1.max()
  f_name=path.split('/')[-1].split('.')[0]
  pca = PCA()
  pca.fit(img)

  # Getting the cumulative variance
  # ================================================

  var_cumu = np.cumsum(pca.explained_variance_ratio_)*100

  # How many PCs explain 99% of the variance?
  # ================================================

  k = np.argmax(var_cumu>99)
  #print("Number of components explaining 99% Variance: "+ str(k))
  plt.subplot(4,4,i)
  plt.title("For image"+f_name+", number of components explaining 99% Variance:␣
 ↪"+ str(k))
  plt.ylabel('Cumulative Explained Variance')
  plt.xlabel('Principal components')
  plt.axvline(x=k, color="k", linestyle="--")
  plt.axhline(y=99, color="r", linestyle="--")
  ax = plt.plot(var_cumu)
  i+=1
  comps1.append(k)
```

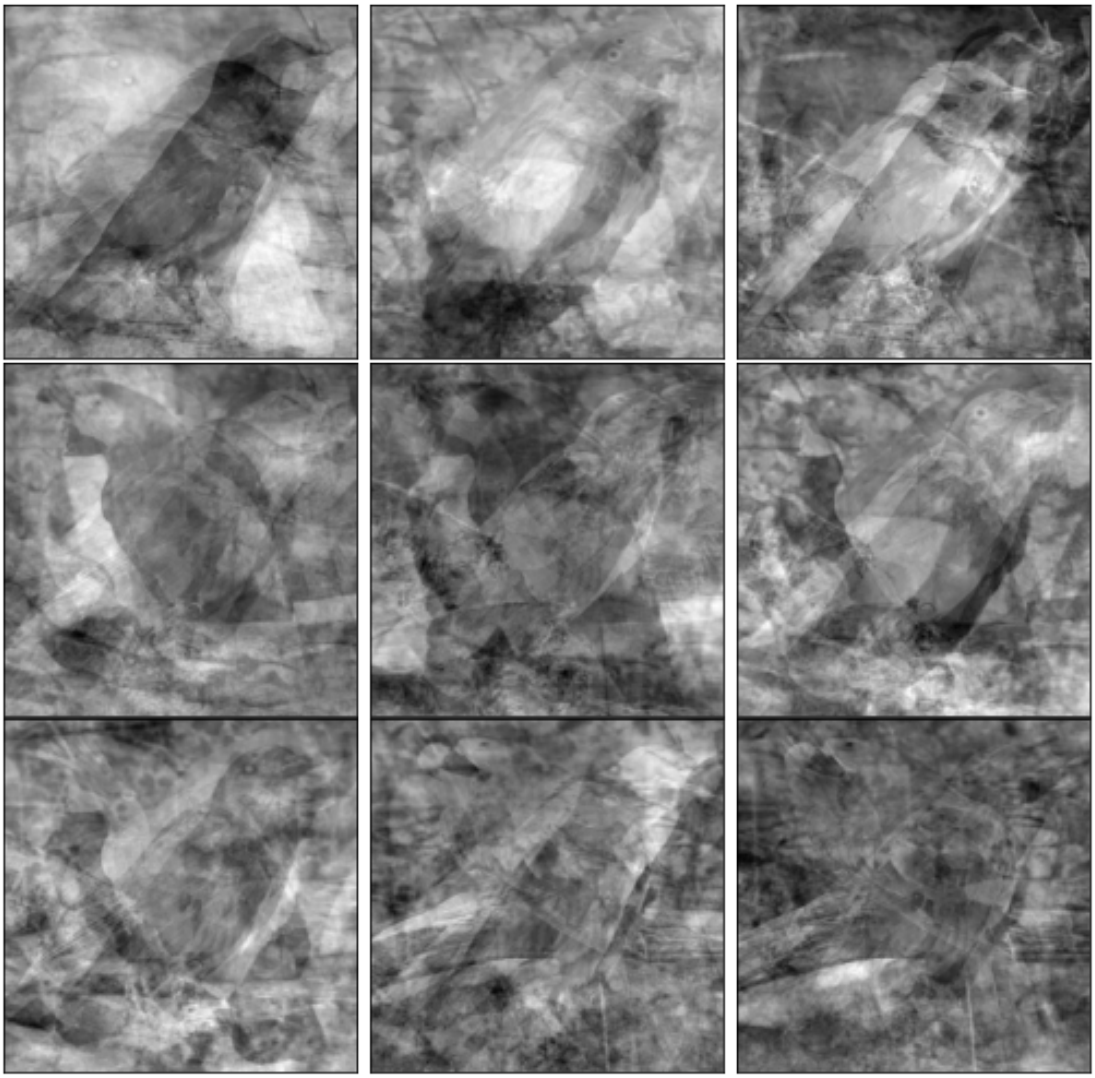


[67]:
```
from sklearn.decomposition import PCA, IncrementalPCA
from glob import iglob
i=1
plt.figure(figsize=[30,15])

for path in iglob('/content/Birds-Test/*.png'):
  img=imread(path)
  img=img[:,:,0:3]
  img1=img.sum(axis=2)
  img = img1 /img1.max()
  f_name=path.split('/')[-1].split('.')[0]
  ipca = IncrementalPCA(n_components=comps1[i-1])
  image_recon = ipca.inverse_transform(ipca.fit_transform(img))

  plt.subplot(4,4,i)
  plt.title("For image"+f_name)
  plt.imshow(image_recon,cmap = plt.cm.gray)
  i+=1

v_comp1=sum(comps1)
print("Ratio of compression: ",str((16*224)/v_comp1)[:6])
```

```
Ratio of compression:  4.0405
```



[69]:
```
from sklearn.decomposition import PCA
#n_components=0.80 means it will return the Eigenvectors that have the 80% of␣
 ↪the variation in the dataset
birds_pca = PCA(n_components=.8)
birds_pca.fit(birds)
fig, axes = plt.subplots(3,3,figsize=(9,9),
 subplot_kw={'xticks':[], 'yticks':[]},
 gridspec_kw=dict(hspace=0.01, wspace=0.01))
for i, ax in enumerate(axes.flat):
 ax.imshow(birds_pca.components_[i].reshape(224,224),cmap="gray")
```



[70]:
```
components = birds_pca.transform(birds)
projected = birds_pca.inverse_transform(components)
fig, axes = plt.subplots(4,4,figsize=(9,9), subplot_kw={'xticks':[], 'yticks':
 ↪[]},
            gridspec_kw=dict(hspace=0.01, wspace=0.01))
for i, ax in enumerate(axes.flat):
    ax.imshow(projected[i].reshape(224,224),cmap="gray")
```