Image Compression Using PCA

Brent White

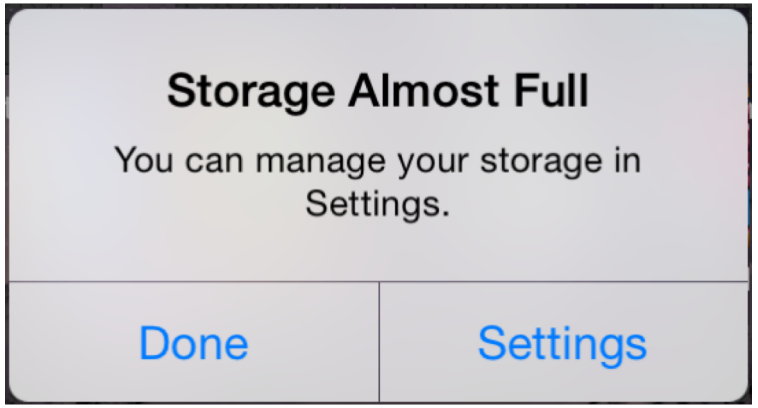
12-5-2018

Goals

- Use linear algebra to solve the matrix so we can extract main components
- Compress images of dogs using Principal Component Analysis to obtain the most visually appealing compressed image
- Compare the effectiveness of our program by comparing an image to its accompanying 50%, 90%, and 99% explained version

Why use PCA?

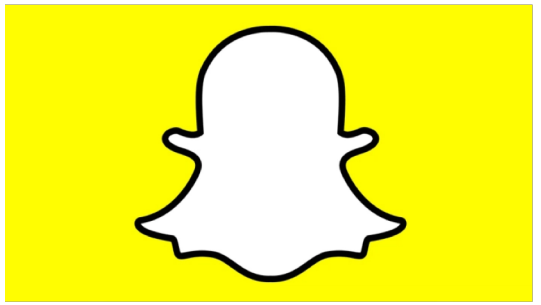
- PCA takes a realistic approach to compression matrix generation by gathering real images to use as data and attempts to find an underlying structure that is common amongst all pictures



- Uses orthogonal transformations to turn a potentially correlated set of data into a linearly uncorrelated set of data which contain principal components

PCA in Business Applications

- Save space in a database, and thus the company's revenue, which is a beneficial and efficient way to store image data
- When you have limited space, this is one of the best ways to store part of your data and revise it to images as close as possible to the original images
- When you just want to identify what's in the picture. Compressed data should be enough

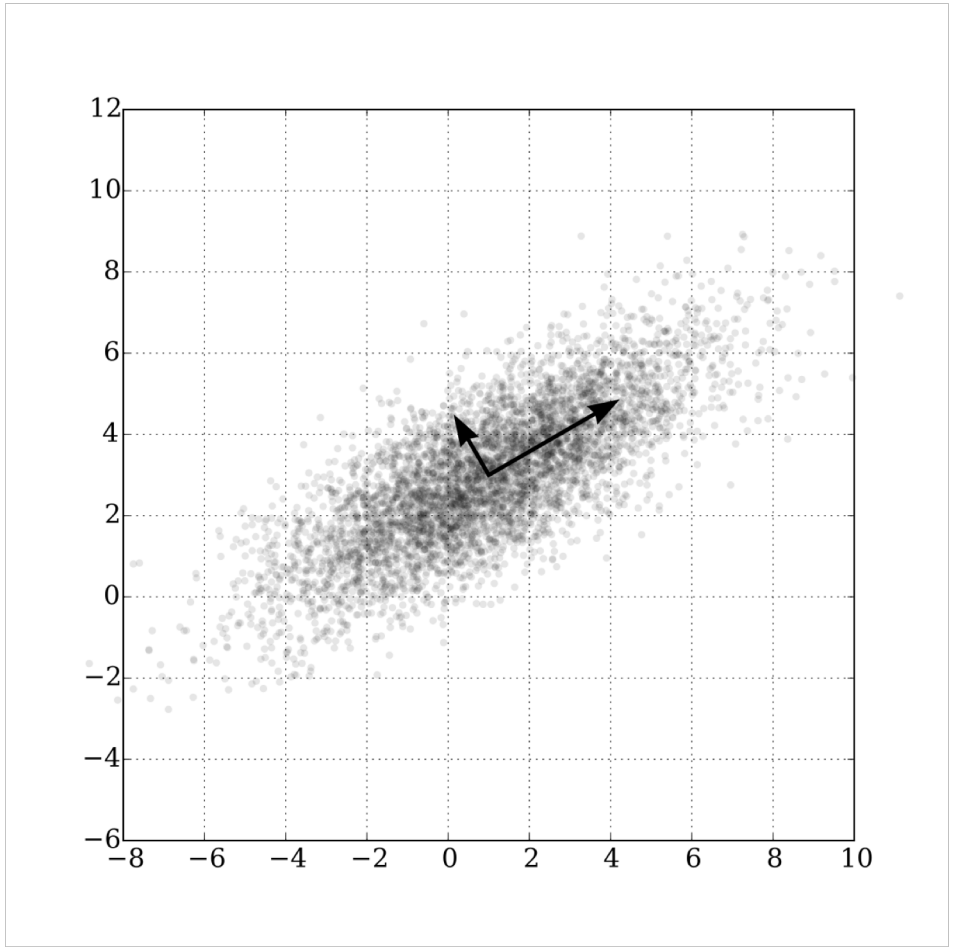


Instagram

So, what's the plan?

Steps:

- Transform image to matrix
- Calculate covariance matrix
- Find eigenvalues (to determine magnitude)
- Find eigenvectors (principal components)
- Multiply matrix with eigenvectors



Mathematical Background

- If A is symmetric (meaning $A^T = A$), then A is orthogonally diagonalizable and has only real eigenvalues. That is, there exist eigenvalues $\lambda 1, \ldots, \lambda n$ and eigenvectors $v_1, v_2 \ldots, v_n$ such that:

$$Av_i = \lambda_i v_i \text{ for each } i = 1, 2, \ldots, n.$$

- This is the spectral theorem: A^TA and AA^T share the same eigenvalues *and* these eigenvalues are non-negative

- So if we have have a matrix we can make it symmetric by calculating its covariance matrix.

Mathematical Application

- First we scale the data. This reduces the emphasis caused by the larger data points in the raw data
- We then compute the covariance matrix from our data thus:
  - $S = AA^T/(n - 1)$
- This covariance matrix should show the main values. Our eigenvalues are derived from the covariance matrix:

Sample covariance matrix: $\begin{pmatrix} 45 & 2 \\ 2 & 8 \end{pmatrix}$

- We then find the eigenvalues: λ_1, λ_2 , then the eigenvectors from these eigenvalues.
- We use these eigenvalues and eigenvectors to identify the dog images

Code Sample

Set File Paths for our image directory

In [2]:
```
base_path = Path().resolve()
path = os.path.join(base_path, 'dogs')
path_pre_transform = os.path.join(base_path, 'dogs_pre_transform')
path_compressed = os.path.join(base_path, 'dogs_compressed')
path_post_transform = os.path.join(base_path, 'dogs_post_transform')
path_figures = os.path.join(base_path, 'figures')
```

Create a dictionary of images and convert to YUV (Y component only) form

In [3]:
```
im_dict = {}
old_size_dict = {}
for f in os.listdir(path):
    old_size_dict.update({f: os.path.getsize(os.path.join(path, f))})
    img = cv2.imread(os.path.join(path,f))
    img_yuv = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)
    y, u, v = cv2.split(img_yuv)
    cv2.imwrite(os.path.join(path_pre_transform,  f) , y)
    im_dict.update({f : y})
```

Define a helper function to scale features and perform PCA on an image then return its reconstructed form

In [4]:
```
def transform_images(f, X, var_frac):
    pipeline = Pipeline([('scaler',StandardScaler()), ('pca', PCA(var_frac))])
    compressed_img = pipeline.fit_transform(X)
    num_features = compressed_img.shape[1]
    return f, compressed_img, pipeline.inverse_transform(compressed_img), num_features, v
ar_frac
```

In [5]:
```
pool = mp.Pool(mp.cpu_count())
```

Perform the PCA across several targets for variance explained

In [6]:
```
arg_list = []
for var_frac in [0.75, 0.9, 0.95, 0.99]:
    for f, X in im_dict.items():
        arg_list += [(f, X, var_frac)]
results = pool.starmap(transform_images, arg_list)
```

Write the results to a file

In [7]:
```
size_dict = []
for f, compressed_img, img, n, v in results:
    cv2.imwrite(os.path.join(path_compressed, str(v)+'_' +str(n)+'_'+f) , compressed_img)
    cv2.imwrite(os.path.join(path_post_transform, str(v)+'_'+str(n)+'_'+f), img)
    new_size = os.path.getsize(os.path.join(path_compressed, str(v)+'_'+str(n)+'_'+f))
    size_dict.append({'name' : f,
                      'num features' : n,
                      'var explained' : v,
                      'old size' : old_size_dict[f],
                      'new size' : new_size})
```

In [8]:
```
sizes_df = pd.DataFrame(size_dict).sort_values(['name', 'var explained'])
sizes_df['size ratio'] = sizes_df['new size']/sizes_df['old size']
```

Step 1: Load data

Step 2: Convert RGB (color jpg files) to YUV and strip the Y component

Step 3: Define a function that fits sklearn PCA to an image and returns a reconstructed image using the fitted model

Step 4: Create a pool of workers and distribute application of the PCA to each worker for a variety of target variances

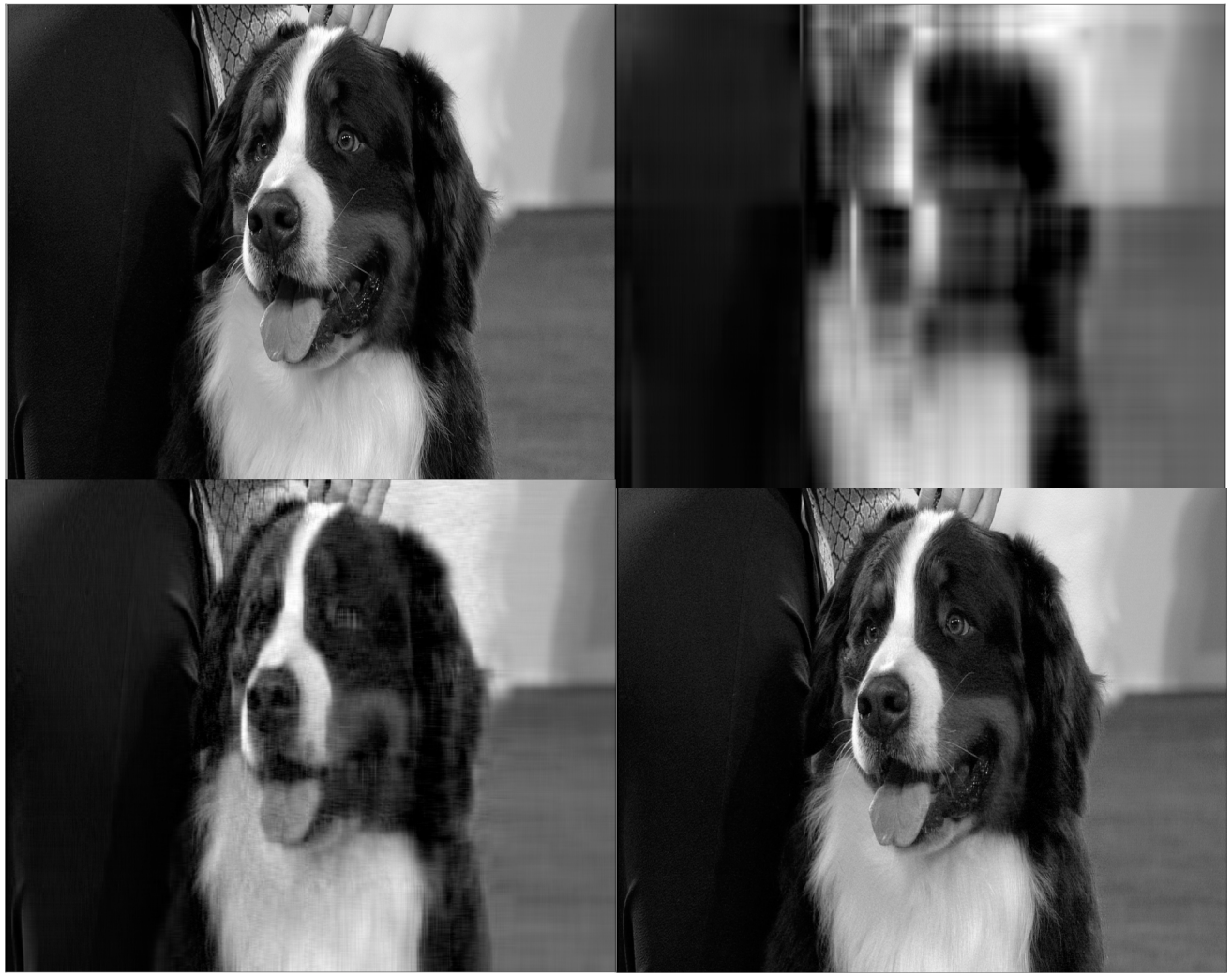
Step 5: Store results as .jpg files for comparison with originals

Image #1

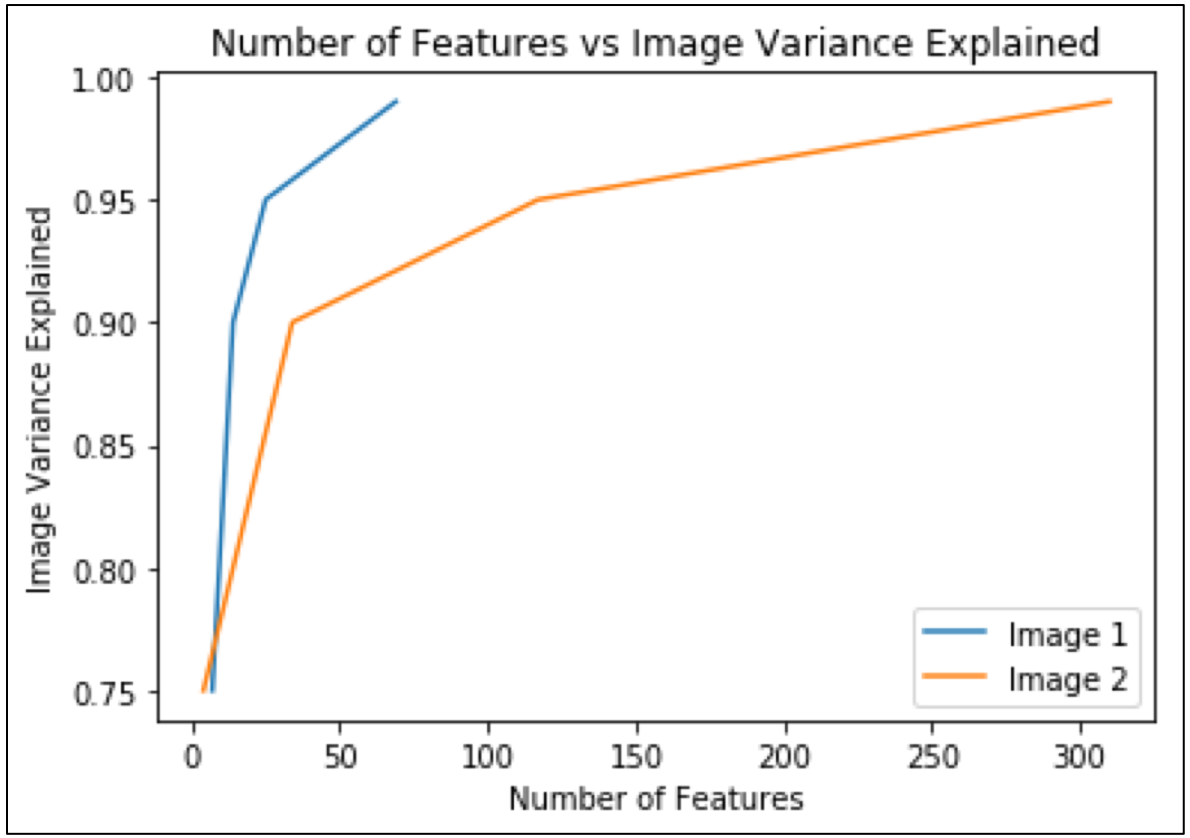
- Original
- 75% variance explained
- 90% variance explained
- 99% variance explained

Image #2

- Original
- 75% variance explained
- 90% variance explained
- 99% variance explained



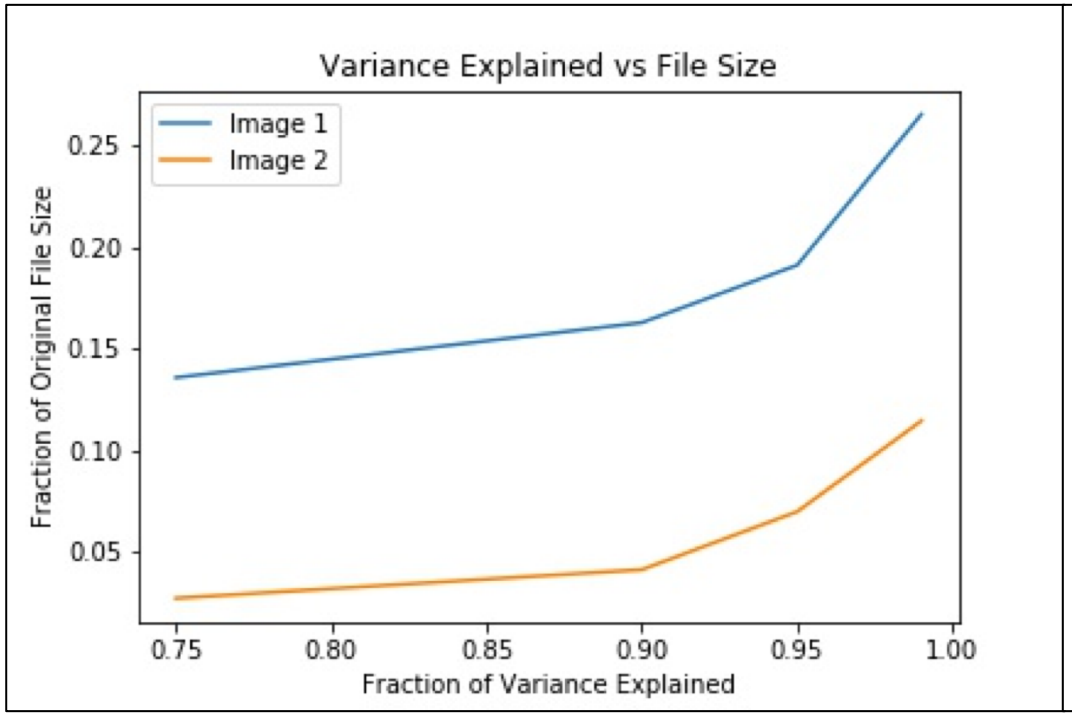
Results and Discussion

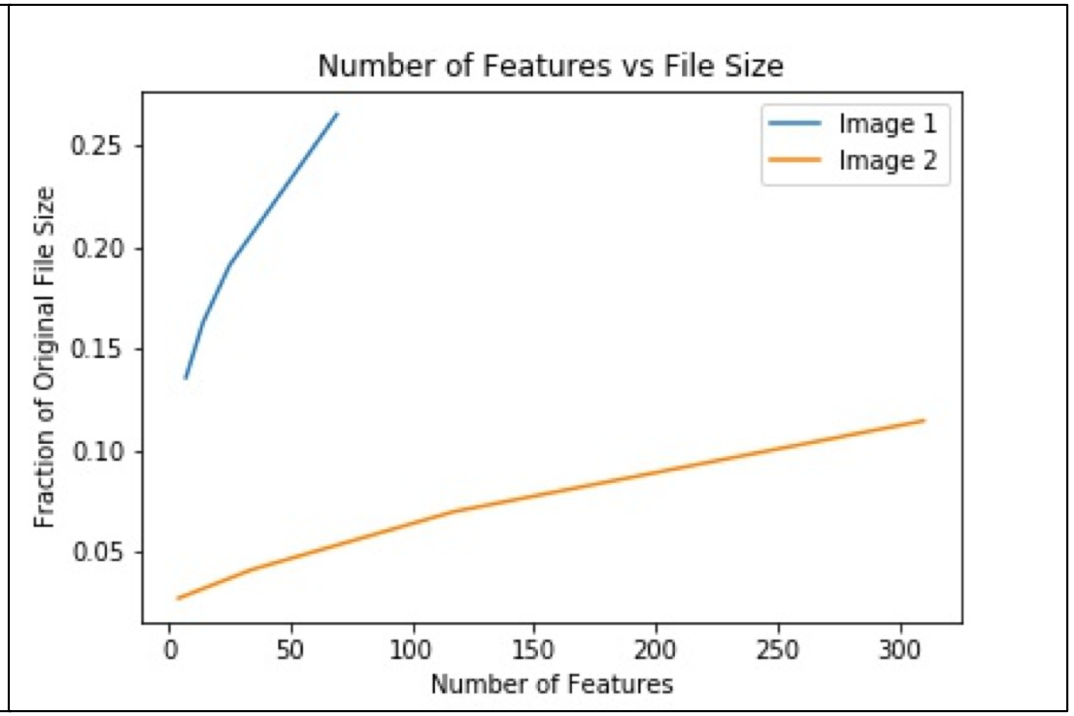


- A small number of features (relative to the original) are able to explain the majority variance in both images

- If we include more data on the amount of variance explained, the curve for Image 1 should look similar to that of Image 2

Results and Discussion





- Even at 99% variance, compressed image size is greatly decreased
- Looking at file size vs the number of features included in the compression, it is apparent that Image 1's features are more rich

Conclusions & Future Work

- The quantity of principal components (eigenvectors) used in the compression influences the recovery of the original image from the final (compacted) image.
- The number of eigenvectors to keep depends on business need: to identify a pattern or just for storage
- We will try to figure out what's the smallest number of eigenvalues to keep so that machine can tell the difference between cats and dogs
- Determine the best way to store PCA models

Sources

- Jauregui Jeff, Principal Component Analysis with Linear Algebra, August 31 2012
- http://www.scielo.br/scielo.php?script=sci_arttext&pid=S1679-45082012000200004
- https://sebastianraschka.com/Articles/2014_pca_step_by_step.html#drop_labels
- https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html