Álgebra Linear

Eigenfaces - Aplicando em imagens de Gatos e Cachorros

George Dutra e Zuilho Segundo

Escola de Matemática Aplicada
Fundação Getúlio Vargas
Brasil
Novembro 2022

Sumário

1 Introdução

O presente trabalho visa aplicar os conceitos aprendidos durante o semestre na disciplina de Álgebra Linear em algo prático. O tema escolhido pela dupla foi *EigenFaces*. Apesar de limitada, essa técnica permite a aplicação de conceitos como *eigenvectors*, *eigenvalues*, SVD e PCA no reconhecimento de imagens, um tema muito relevante nos dias de hoje.

Criada na em 1987 pelos pesquisadores Sirovich and Kirby e mais tarde utilizada por Matthew Turk e Alex Pentland no reconhecimento de rostos, a técnica consiste na análise das componentes principais para mais tarde identificar se a imagem é um rosto, e por fim, descobrir quem é o 'dono' do rosto.

Nosso objetivo com o presente trabalho é criar uma adaptação da aplicação do *EigenFaces*, para que ao invés de identificar rostos, sejamos capazes de criar um algoritmo de classificação de imagens de gatos e cachorros. Os detalhes serão explicados nas sessões posteriores.

2 EigenFaces

A ideia geral por trás dessa técnica consiste em descobrir o conjunto de vetores (os *eigenvectors*) que são capazes de descrever 'qualquer' imagem como uma combinação da face média, que é obtida de todas as imagens do conjunto de treinamento, com as *eigenfaces*, que são os *eigenvactors* calculados do conjunto de imagens.

Para começar, tendo um *dataset* com k imagens de $m \times n$ *pixels*, é preciso construir uma matriz onde cada imagem é convertida para escala de cinzas, e transformada em uma linha. Dessa forma, temos a matriz $A_{k \times mn}$ da qual serão extraídas as *eigenfaces*.

$$A_{k \times mn} = \begin{bmatrix} — & I_0 & — \\ & \vdots & \\ — & I_k & — \end{bmatrix}$$

O próximo passo é encontrar a face média, para que seja possível centralizar e realizar a análise dos componentes principais. Nessa análise, a face média é dada por:

$$f_m = \frac{1}{k} \sum_{i=1}^{k} a_i$$

Agora vamos calcular o SVD da matriz de coeficientes descontando a média:

$$\overline{A} = \begin{bmatrix} — & I_0 - f_m & — \\ & \vdots & \\ — & I_k - f_m & — \end{bmatrix} = USV^T$$

Essa operação nos retorna os *eigenvectors* da matriz de covariância com todas as imagens.

Agora que temos o a matriz de *eigenvectors*, que resultou da decomposição SVD (o *facespace*), precisamos projetar as imagens que queremos no espaço coluna da matriz, e por fim calcular o erro da projeção, que seria a norma do vetor p projetado. Determinando um *treshold* para esse norma, somos capazes de identificar se uma imagem está contida no *facespace*, e pela distancia dos elementos de alguma face, teríamos a identificação do rosto.

3 Aplicando em Imagens de Gatos e Cachorros

3.1 Criando os *EigenCats* e os *EigenDogs*

Com tudo isso esclarecido, seguimos para o desenvolvimento do projeto de identificar cachorros e gatos. Num primeiro momento, selecionamos uma base de dados do Kaggle de faces de gatos e

cachorros, como 771 imagens de gatos e 1264 images de cães para o treinamento. A primeira base de dados pode ser encontrada no seguinte link: cat2dog. Porém, enquanto desenvolvíamos, tivemos problemas com a dimensão dos espaços e a generalização do nosso modelo, por isso, foi necessário incluir mais algumas fotos no nosso *dataset*, por isso, para as imagens de gatos acrescentamos o *dataset* 'Cats faces 64x64' e o 'Cropping Dog Faces' para os cachorros.

Aqui no relatório iremos apresentar apenas as partes mais importantes da implementação, porém, o código para a consulta se encontra no GitHub.

Começamos pela geração das *eigenfaces* dos gatos, que chamaremos de *eigencats*. A primeira coisa a fazer, foi redimensionar as imagens para que tivéssemos capacidade de realizar os cálculos em nossos computadores. Para exemplificar, quando tentamos rodas com as imagens originais (de 178x218 pixels) era necessário aproximadamente 12GiB de memória para a execução do *script*.

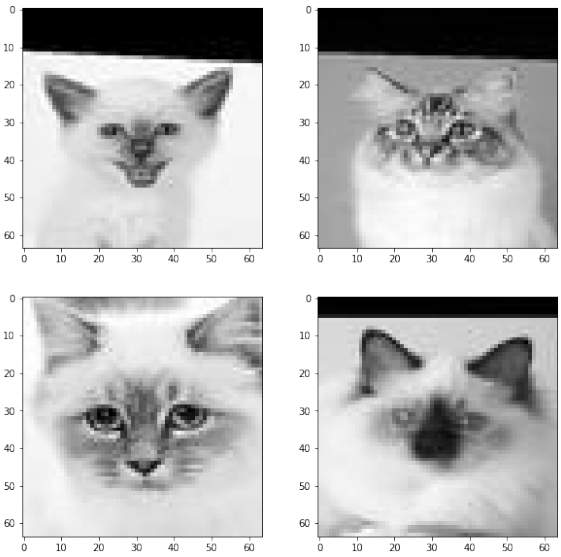

Figura 1: Exemplos do *Dataset* 64x64

Com as imagens redimensionadas, construímos a matriz A com as imagens e dela calculamos o SVD.

```
train_cat_np_matrix = train_cat_np.reshape(train_cat_np.shape[0],
↪  train_cat_np.shape[1]*train_cat_np.shape[2])
mean_train_cat = np.mean(train_cat_np_matrix, axis=0)
centered_train_cat = train_cat_np_matrix - mean_train_cat
U_CAT_TRAIN, S_CAT_TRAIN, V_CAT_TRAIN = np.linalg.svd(centered_train_cat,
↪  full_matrices=False)
```

Aqui podemos ver o que seria o 'gato médio' do nosso código:

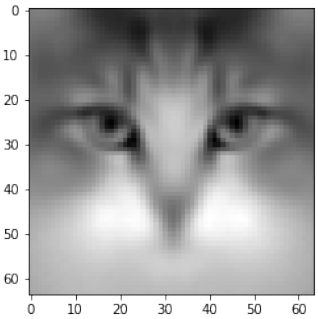

Figura 2: Gato Médio

E em seguida temos alguns *eigencats* extraídos da nossa matriz V^T:

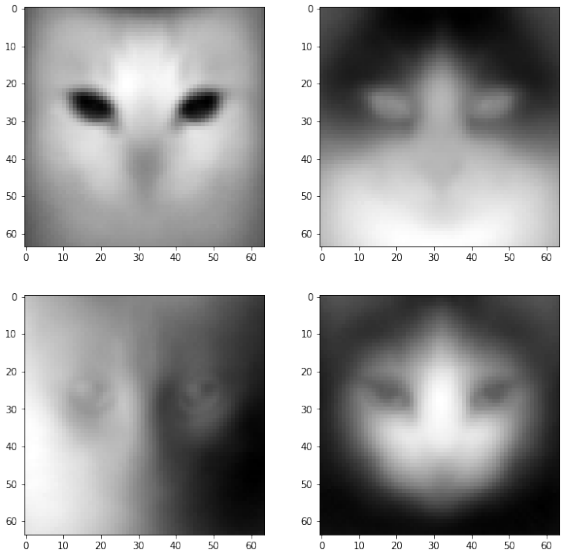

Figura 3: *Eigencats*

Com isso já somos capazes de fazer a função que identifica se uma imagem é um gato ou não. O limite de 5000 foi definido experimentalmente, rodando o código com algumas imagens e vendo como eram os resultados:

```
1 def is_a_cat(imagem, n_componentes):
2     img3 = imagem.reshape((4096,))
3     U_ = U_CAT_TRAIN[:n_componentes]
4     p = np.dot(np.identity(n_componentes) - np.dot(U_,U_.T), img3 -
      ↪  mean_train_cat)
5     d = np.linalg.norm(p)
6     lim = 5000
7     return d < lim, d, p
```

Aqui temos o exemplo de como fica a projeção de uma imagem de um gato no espaço coluna da matriz de imagens:

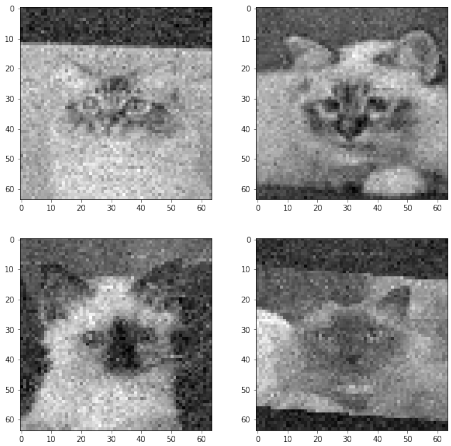

Figura 4: *Projeção de Gatos*

A mesma ideia foi seguida para encontrar os 'cachorros médios', *eigendogs* e as projeções e para evitar repetições, vamos mostrar apenas alguns resultados:

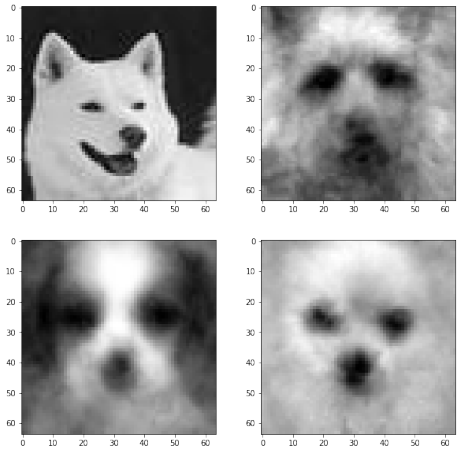

Figura 5: Exemplos do *Dataset* 64x64

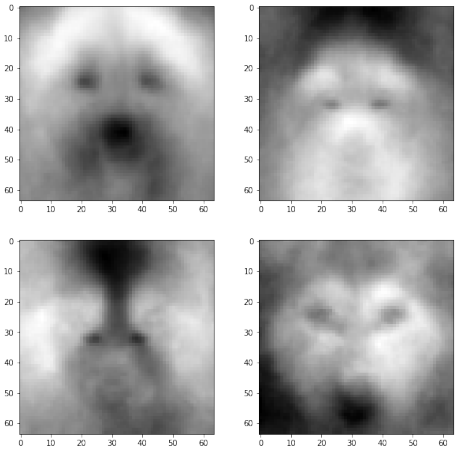

Figura 6: Eigendogs

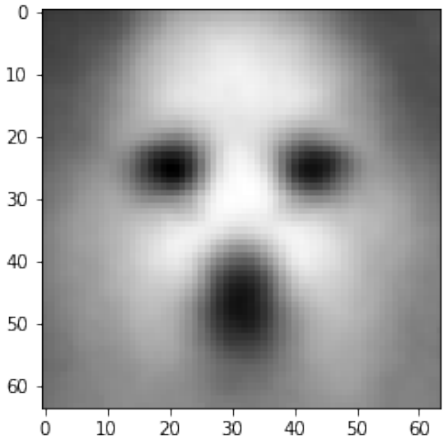

Figura 7: Cachorro Médio

Figura 8: Projeção

```python
def is_a_dog(imagem, n_componentes):
    img3 = imagem.reshape((4096,))
    U_ = U_DOG_TRAIN[:n_componentes]
    p = np.dot(np.identity(n_componentes) - np.dot(U_,U_.T), img3 -
    ↪  mean_train_dog)
    d = np.linalg.norm(p)
    lim = 4500
    return d < lim, d,
```

3.2 Classificação

Agora com as matrizes de *eigenvectors* contruídas, temos o código que classifica a imagem entre cachorros, gatos e nenhum dos dois:

```
def cat_or_dog(image):
    is_dog = is_a_dog(image, n_comp)
    is_cat = is_a_cat(image, n_comp)
    if is_dog[1] < is_cat[1] and is_dog[0] == True:
        return "Dog", is_dog[1], is_cat[1]
    elif is_dog[1] > is_cat[1] and is_cat[0] == True:
        return "Cat", is_cat[1], is_dog[1]
    else:
        return "None", is_cat[1], is_dog[1]
```

Para validar a eficiência do nosso algoritmo realizamos um teste para verificar a acurácia do nosso modelo, utilizando um conjunto de imagens de gatos e cachorros.

```
test_cat = "cat2dog/testA"
accuracy_test = []
for image in (os.listdir(test_cat)):
    path = os.path.join(test_cat, image)
    img = plt.imread(path)
    img1 = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img2 = cv2.resize(img1, dsize=(64,64), interpolation=cv2.INTER_CUBIC)
    img3 = img2.reshape((4096,))
    accuracy_test.append(cat_or_dog(img3)[0] == "Cat")
accuracy_test = np.array(accuracy_test)
print("Accuracy:", np.sum(accuracy_test)/len(accuracy_test))
>>> Accuracy: 0.35
```

```
test_dog = "cat2dog/testB"
accuracy_test_dog = []
for image in (os.listdir(test_dog)):
    path = os.path.join(test_dog, image)
    img = plt.imread(path)
    img1 = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img2 = cv2.resize(img1, dsize=(64,64), interpolation=cv2.INTER_CUBIC)
    img3 = img2.reshape((4096,))
    accuracy_test_dog.append(cat_or_dog(img3)[0] == "Dog")
accuracy_test_dog = np.array(accuracy_test_dog)
print("Accuracy:", np.sum(accuracy_test_dog)/len(accuracy_test_dog))
>>> Accuracy: 0.64
```

Como podemos reparar, as imagens de teste de gato foram bem mal, se comparadas as imagens de cachorros. Tentando explicar isso, percebemos um dos principais problemas do método de *Eigenfaces*, que é a dificuldade de generalizar e lidar com mudanças de angulos, distância e iluminação. O conjunto de teste que temos é de antes de adicionarmos a segunda base de dados. Como podemos perceber olhando para a Figura 9, a distância das fotos desses gatos para câmera é bem maior do que a obtida pelo 'gato médio', o que explica o fato de eles não terem sido classificados como gatos. O mesmo não acontece com os cachorros, estão bem representados mesmo com a adição desse novo *dataset*.

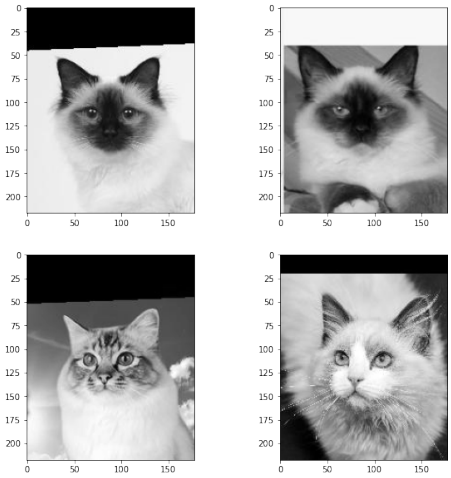

Figura 9: *Gatos de Teste*

Aqui temos exemplos de algumas imagens que foram utilizadas para realizar testes isolados. Como já falado, qualquer pequena variação gera resultados estranhos, pois os componentes que o computador olha utilizando o *eigenfaces* é diferente do que um cérebro humano identifica.

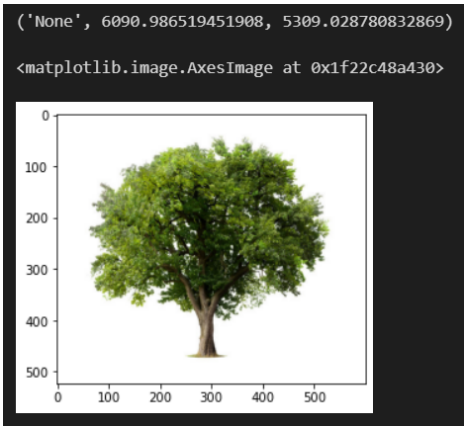

Figura 10: Árvore

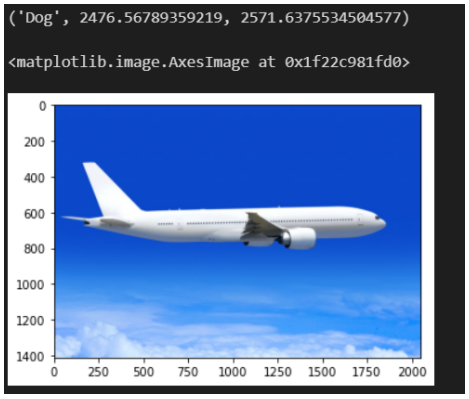

Figura 11: Avião

Figura 12: Cachorro 1

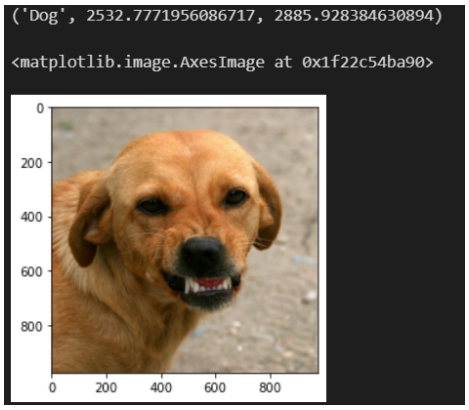

Figura 13: Cachorro 2

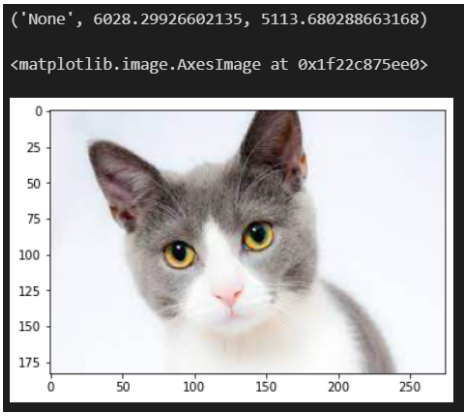

Figura 14: Gato 1

Figura 15: Gato 2

4 Conclusão

Na atualidade é pouco provável que um sistema eficiente vá se utilizar de um método de reconhecimento e classificação utilizando *eigenfaces* devido aos problemas já mosrados, como a dificuldade de genralização para lidar com variações de distância de iluminação, porém é importante perceber como coisas simples da Álgebra Linear são poderosas o suficiente para permitir o surgimento coisas como um sistema de reconhecimento.

5 Referências

- https://towardsdatascience.com/eigenfaces-recovering-humans-from-ghosts-17606c328184
- https://en.wikipedia.org/wiki/Eigenface
- https://www.lcg.ufrj.br/marroquim/courses/cos756/trabalhos/2013/abel-nascimento/abel-nascimento-report.pdf
- https://pyimagesearch.com/2021/05/10/opencv-eigenfaces-for-face-recognition/