Instituto Superior de Engenharia de Lisboa

Visão Artífical e Realidade Mista

Engenharia Informática e Multimédia

Deteção e Reconhecimento Facial para Realidade Aumentada

Engº Pedro Jorge

MM2N

Arman Freitas nº45414

1 de maio de 2022

Conteúdo

Lista de Figuras

1 Introdução e Enquadramento

O seguinte projeto visa a deteção e o reconhecimento facial para realidade aumentada. Realidade aumentada é a combinação entre objetos digitais com os sentidos humanos em tempo real, que causa o digital a estar anexado ao espaço físico (registo 3D).

Realidade aumentada deve conter as seguintes características:

- Combinar o real com o virtual.
- Interação em tempo real.
- Registar em 3D.

Será utilizada a linguagem de programação *Python* para o desenvolvimento da aplicação que deteta, reconhece e utiliza faces para realidade aumentada. De forma a auxiliar o desenvolvimento, será também utilizada a biblioteca *OpenCV*, uma biblioteca extremamente poderosa no que diz respeito a processamento de imagens em *Python*.

O produto final, será uma aplicação que deteta faces e, consoante a face detetada adiciona um objeto virtual à mesma.

É de notar que serão utilizados os algoritmos *Eigenface* e *Fischerface* para processar as faces e, um classificador KNN (*K-Nearest-Neighbour*), que serão aprofundados posteriormente.

Sendo assim, o projeto segue os seguintes pontos:

- Deteção Facial
- Normalização
- Algoritmos *Eigenface* e *Fischerface*

- Classificador KNN
- Realidade Aumentada

2 Desenvolvimento

2.0.1 Deteção

O primeiro passo para o desenvolvimento do projeto é de facto a deteção facial. Este passo prende-se principalmente com encontrar uma face numa imagem.

Foram dadas a conhecer duas formas de detetar faces pelo docente no decorrer das aulas.

- Classificador *Haarcascade*
- Módulo DNN (*Deep Neural Network*) do opencv

O classificador *Haarcascade* utiliza deteção de características dos contornos. Para ser treinado, este classificador necessita de bastantes imagens com faces, assim como muitas imagens sem faces.

Os dados do mesmo são guardados em ficheiros XML que, podem mais tarde ser utilizados para testar a deteção facial em imagens ou vídeos.

Já o módulo DNN como o nome sugere, trata-se de uma rede neuronal, mais precisamente uma rede neuronal convulcional. Uma rede neuronal convulcional é uma rede neuronal frequentemente utilizada para imagens, para que possam ser extraídas características das imagens, tornando o processo de treino da rede mais rápido e eficaz. Considera-se que este módulo seja bastante superior em deteção facial em imagens sendo que, foi testado ao longo do trabalho e, reconhece faces mesmo com diferentes condições de luz e oclusão da mesma.

Decidiu-se utilizar o módulo DNN para a deteção facial e, dentro desta, uma deteção de olhos por parte do classificador *Haarcascade*, visto que o DNN não possuí classificadores para identificar olhos.

E assim, com as coordenadas dos pontos da cara e dos olhos na mesma, pode-se proceder à normalização da face para que possa ser utilizada nos algoritmos *Eigenface* e *Fischerface*.

2.0.2 Normalização

A normalização segue um conjunto de normas que devem ser seguidas por todas as imagens dos algoritmos *Eigenface* e *Fischerface*.

Não só permite que possíveis classificações se foquem mais na face, como também permitem eliminar bastante informação da imagem desnecessária (fundo, por exemplo).

Assim, para uma imagem ser completamente normalizada:

- A sua altura e largura deve ser de 46 x 56px.
- Os olhos devem estar perfeitamente alinhados horizontalmente.
- O olho esquerdo e direto devem se situar nas posições (16, 24) e (31, 24), respetivamente.

De forma a seguir o *standard*, começa-se por alinhar os olhos na horizontal. A forma de se concretizar é a seguinte, com as coordenadas de cada olho (descobertas pelo *Haarcascade*, secção 2.0.1) aplicar a seguinte regra do cosseno:

$$C^2 = a^2 + b^2 - 2 \times a \times b \times cos\alpha$$

Correspondente ao triângulo da seguinte figura:

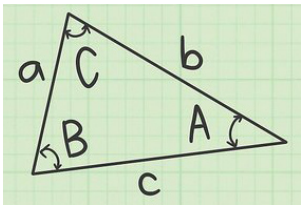


Figura 1: Regra do cosseno

Comparando as posições dos olhos, é possível encontrar o terceiro ponto do triângulo (retângulo) formado entre os olhos. Com três pontos no espaço que formam um triângulo basta aplicar a fórmula acima e, através do cosseno, chega-se ao ângulo entre os olhos.

A imagem é rodada com este ângulo para que os olhos estejam sempre horizontalmente alinhados.

Para resolver a altura e largura da imagem, deu-se *resize* da imagem, para que a distância dos olhos seja a distância definida (15 pontos) e, cortou-se a imagem tal que os olhos pertencessem exatamente às coordenadas pedidas ((16, 24) e (31, 24)) e, a imagem fosse do tamanho 46 x 56px.

Após a aplicação em *Python* da normalização, uma das imagens retiradas, totalmente normalizada, é a seguinte:

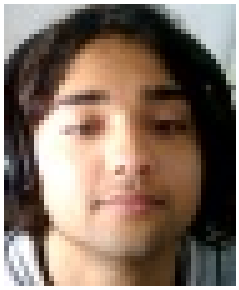


Figura 2: Face normalizada

É de notar que a normalização é realizada em tempo real, para que o processo de realidade aumentada seja mais fácil e, que as imagens devem estar em níveis de cizento, no entanto quando o *OpenCV* lê as imagens, é possível apenas ler todas as imagens em *grayscale* como é o caso.

2.0.3 *Eigenface* e *Fischerface*

***Eigenface*:** É um algoritmo que procura descrever faces por parte de vetores matemáticos. Primeiramente, começa-se por treinar o algoritmo com várias imagens de faces. É sempre necessário que as faces sejam sempre normalizadas com um mesmo *standard*, definido no segmento 2.0.2.

Através destas imagens são geradas as chamadas *eigenfaces* através de PCA (*Principal Component Analysis*). O PCA é um método de redução de dimensionalidade de dados, podendo ser utilizado em imagens para retirar características que não são necessárias à transmissão de informação.

São sempre gerada m *eigenfaces*, sendo que m, pode ser um valor inteiro com um máximo do número de imagens - 1. Quanto maior o m, maior a dimesionalidade das imagens vetorizadas, melhorando a qualidade de vetorização e reconstrução.

É através das *eigenfaces* que, se consegue vetorizar as faces e, mais tarde, caso necessário fazer a reconstrução das mesmas.

A seguinte figura (fig. 3) representa uma vista geral do processo de vetorização e reconstrução de uma face apartir do vetor e das *eigenfaces*:

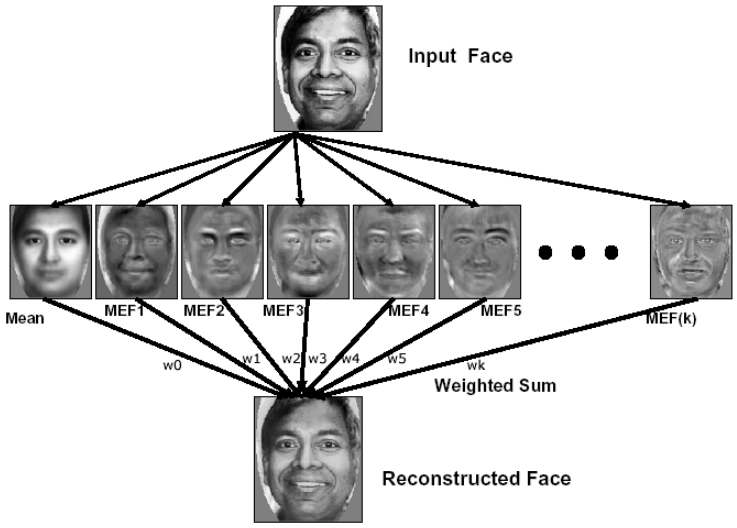


Figura 3: *Eigenface*

Uma forma de averiguar se o *Eigenface* implementado em *Python* esteja correto, é pela multiplicação da matriz W tranposta (matriz que contém as m *eigenfaces*) pela própria matriz W, o que deve retornar uma matriz identidade. A seguinte figura representa o resultado do produto destas duas matrizes que, como se pode confirmar, é de facto uma matriz identidade, verificando-se uma base ortogonal da matriz W.

```
Multiplicação de W transposto com W
[[ 1.00000000e+00 -4.64648394e-17  4.55093862e-17  2.88885669e-16]
 [-4.64648394e-17  1.00000000e+00  1.33709233e-16 -4.60623293e-16]
 [ 4.55093862e-17  1.33709233e-16  1.00000000e+00  1.06034972e-16]
 [ 2.88885669e-16 -4.60623293e-16  1.06034972e-16  1.00000000e+00]]
```

Figura 4: Verificação de base ortogonal da matriz W

***Fischerface*:** Diferente do *Eigenface* que nem possui o conceito de classe, este método tenta maximizar a distância dos vetores de cada classe. Isto resulta em projeções onde as distâncias entre as classes são muito maximizadas, enquanto que as distâncias entre vetores intra-classe são minimizadas, "agrupando" faces da mesma classe.

Este método não utiliza algumas das componentes que são responsáveis pela intensidade da luz, sendo muito mais tolerável a ambientes de luz diferentes.

Com isto dito, é natural que, mesmo antes de realizar testes se veja melhores resultados no que diz respeito à classificação de faces pré-processadas pelo algoritmo *Fischerface*.

É também de notar que este algoritmo utiliza o próprio *Eigenface* para alguns dos seus cálculos e, reconhece o conceito de classes no campo de reconhecimento facial. O *Eigenface* não visa assim tanto ser utilizado para classificação, mas sim para obter as melhores formas de representar faces por parte de vetores.

2.0.4 Classificador

Tal como requerido no enunciado, foi utilizado um classificador KNN (*K-Nearest-Neighbours*) para classificar os vetores vindos dos algoritmos.

***K-Nearest-Neighbours*:** É um classificador que classifica novos vetores relativamente aos K vizinhos mais próximos. Uma instância é sempre classificada na classe a que pertencem mais dos K vizinhos mais próximos.

Para maior simplicidade e rapidez, resolveu-se utilizar o classificador KNN presente na biblioteca *Scikit-learn*.

O classificador é então treinado, utilizando as imagens normalizadas presentes em formato vetor (*Eigenface* ou *Fischerface*) e, as respetivas classes de cada uma delas.

Assim, é possível predizer a classe de futuras imagens, quando transformadas em vetores pelo respetivo algoritmo.

2.0.5 Realidade Aumentada

Após o desenvolvimento do classificador e dos algoritmos por este utilizados, é necessário utilizar imagens reais para reconhecer a cada face. Será colocado um objeto virtual na face da pessoa, consoante a sua classe.

O primeiro passo é normalizar objeto. Isto é, realizar transformações (*scale*) que o adaptem a face da pessoa detetada. Cada objeto é escalado no eixo do x e do y para ter porporções da cara.

Com o objeto normalizado, apenas se deteta os pontos onde se deseja colocar o objeto na face da pessoa. Todas as coordenadas e tamanhos dos objetos devem ser relativos à face e, não absolutos.

Para realmente se colocar o objeto na imagem, utiliza-se a técnica de *masking*.

Esta técnica, envolve a criação de uma máscara do objeto, de forma a remover o fundo da imagem do mesmo (fig. 5). A utilização da máscara inversa na imagem original nas coordenadas onde se pretende colocar o objeto cria a figura 6.

Figura 5: Remoção do fundo do objeto virtual

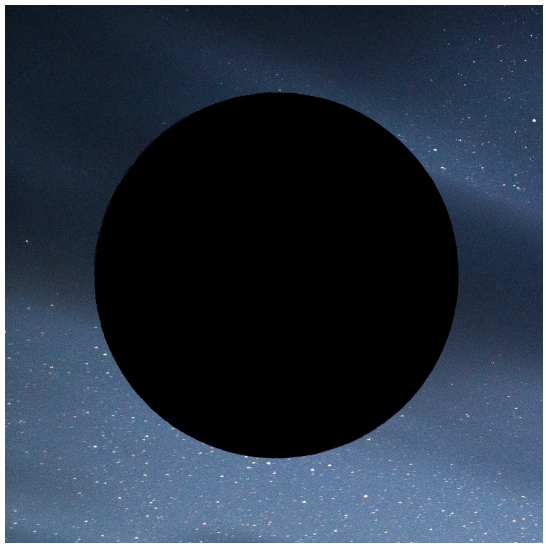


Figura 6: Adição do fundo da imagem principal

Com estas duas imagens, apenas se adicionam uma à outra:

Figura 7: Adição do fundo ao objeto virtual

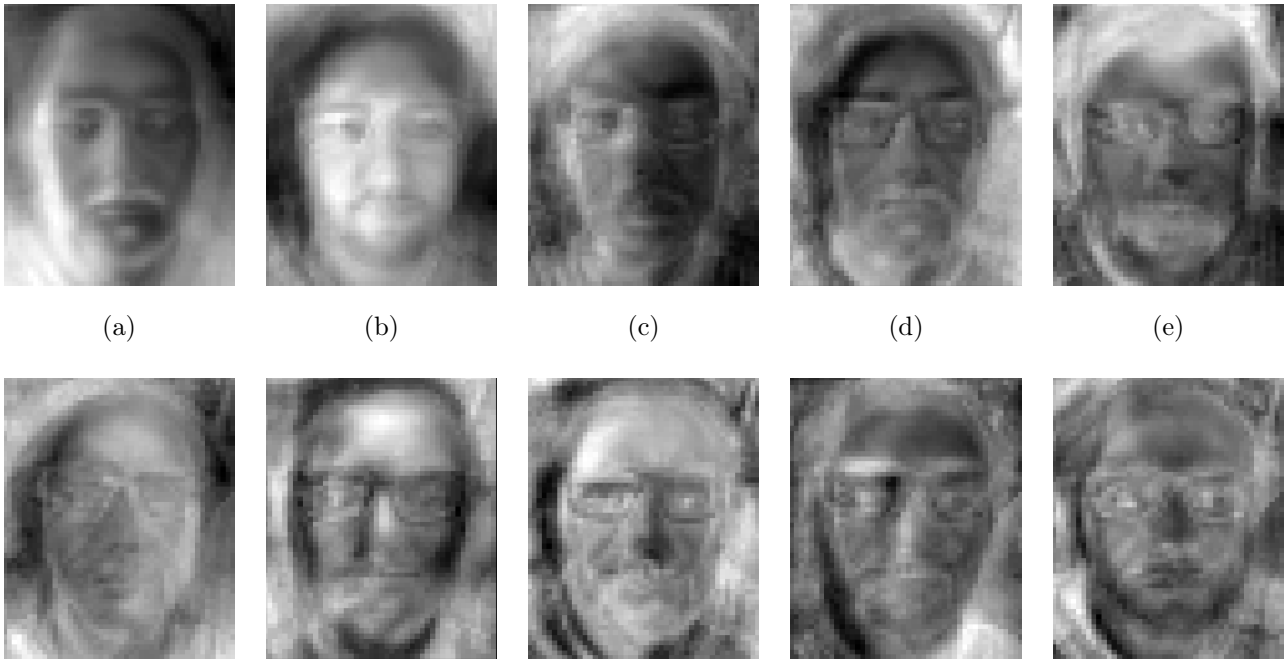
Por fim, adiciona-se à imagem original e, as imagens estão sobrepostas:

Figura 8: Imagem final com objeto sobreposto

2.1 Resultados

Este segmento pretende a demonstração dos resultados obtidos, no que diz respeito aos algoritmos, à classificação e à realidade aumentada.

***Eigenfaces*:** As seguintes figuras representam as *eigenfaces* para as imagens todas com um total de 41 faces.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

Figura 9: *Eigenfaces* com $m = 10$

Figura 10: *Mean face*

***Fischerfaces*:** A figuras em seguida representam as *mean faces* para as quatro classes do *dataset*:

Figura 11: *Mean face* - Classe 0

Figura 12: *Mean face* - Classe 1

Figura 13: *Mean face* - Classe 2

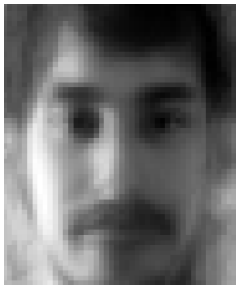


Figura 14: *Mean face* - Classe 3

É também criada uma *mean face* para as imagens todas que tem o mesmo aspeto que a figura 10.

Reconstrução de faces: Foram também utilizados o *Eigenface* e o *Fischerface* para reconstrução de faces. Para testar esta funcionalidade, reconstruiu-se a figura 15 em ambos os algoritmos.

Figura 15: Face original

As faces reconstruídas e os respetivos erros (subtração da face original com a reconstrução) são os seguintes:

Figura 16: Face reconstruída - *Eigenface*

Figura 17: Face reconstruída - *Fischerface*

Figura 18: Erro - *Eigenface*

Figura 19: Erro - *Fischerface*

Como se pode averiguar nas figuras acima, o erro apresenta-se bastante maior com o *Fischerface*, sendo a imagem reconstruída deste algoritmo bastante diferente da original. Isto deve-se ao *Fischer-*

face se focar na parte de classificação, tentando separar as classes umas das outras e comprimindo-as, não tendo sido criado com o propósito de reconstrução de faces.

Classificação: As imagens foram vetorizadas das seguintes formas nos algoritmos *Eigenface* e *Fischerface*, respetivamente:

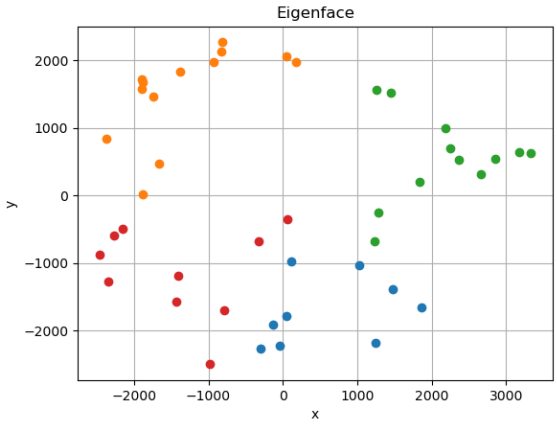


Figura 20: *Dataset - Eigenface*

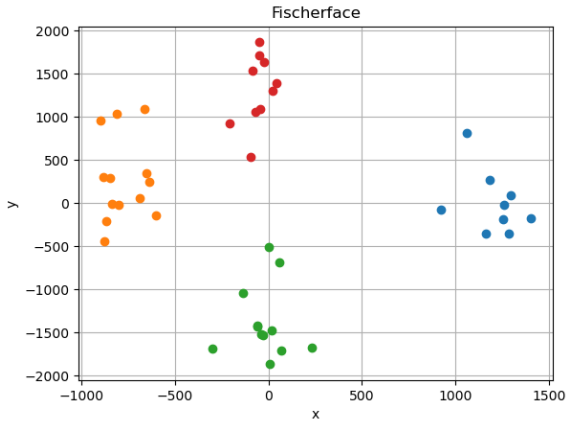


Figura 21: *Dataset - Fischerface*

Os vetores contêm m dimensões porém, apenas são demonstradas as primeiras duas para melhor compreesão. Mas relembra-se que se perdeu de facto alguma informação no *plot* dos gráficos.

Como se pode verificar, os vetores gerados pelo *Eigenface* são deveras mais dispersos que o *Fischerface*. Isto deve-se ao algoritmo *Fischerface* fazer uso de diferentes classes e, tentar maximizar a distância entre as classes, assim como uma compactação dos vetores da mesma classe, para que estes fiquem mais perto uns dos outros.

Realidade Aumentada: Para cada classe foram escolhidos objetos virtais, dependendo da clasificação. À classe 0 pertence a uma coroa, à classe 1 a um nariz de porco e , às classes 2 e 3 a uma tatuagem na cara. Este objetos virtuais foram testados todos, com os seguintes resultados:

Figura 22: Realidade aumentada - classe 0

Figura 23: Realidade aumentada - classe 1

Figura 24: Realidade aumentada - classes 2 e 3

3 Conclusões

O presente projeto permitiu, deveras, um maior entendimento sobre o conceito de realidade aumentada e reconhecimento facial. O nível de interesse realmente cresceu bastante no que diz respeito a esta área. Foram utilizados os algoritmos *Eigenface* e *Fischerface* que conseguem vetorizar uma face em pequenas dimensões que, na sua natureza é bastante impressionante.

Em geral, o desenvolvimento do projeto foi fluído, com pequenas adversidades ao longo do caminho que, com um pouco de pesquisa se resolveram.

Alguns conceitos surgiram ao longo do trabalho, nomeadamente, no segmento de classificação onde se poderia extender mais e, com a ajuda de aprendizagem supervisionada verificar o comportamento do classificador perante dados e algoritmos diferentes.

Desta forma, seria possível caracterizar cada classificador com métricas como o *score*, *accuracy*, vizualização de matrizes de confusão, curvas de roc, etc. Logo após reconhecer as métricas seria possível recalibrar o classificador para melhores resultados, dependendo do algoritmo utilizado (*Eigenface* ou *Fischerface*). No entanto, seria necessário relativamente mais imagens de faces por cada classe, para a divisão das instâncias entre treino e teste.

Outro aspeto que se poderia tomar era, pesquisar diferentes classificadores ou mesmo o uso de redes neuronais para tentar classificar as faces, todavia, mais uma vez seria necessário um maior leque de imagens das faces das classes presentes.

Em boa verdade, considera-se que o projeto tenha tido um grande aproveitamento, conseguindo ter atingido todos os resultados pretendidos pelo enunciado.

4 Bibliografia

- *Slides* fornecidos pelo docente
- Artigo *Medium* - https://shorturl.at/drsOX
- *Comparison between face recognition algorithms* - https://shorturl.at/aouIR