DDETC – Área Departamental de Engenharia Eletrónica e Telecomunicações e de Computadores

MEIM - Mestrado Engenharia informática e multimédia

Visão Artificial e Realidade Mista

Projeto 1

Turma:

MEIM-21N

Trabalho realizado por:

Duarte Domingues Nº45140

Docente:

Pedro Mendes Jorge

Data: 02/05/2022

Índice

Índice de ilustrações

1. Introdução

Este projeto baseia-se no desenvolvimento de uma aplicação para deteção e recognição de faces que permite a inclusão de elementos virtuais alinhados com objetos reais.

A primeira fase do projeto assenta-se na deteção de faces em tempo real utilizando diferentes métodos. Inicialmente utilizou-se para esta tarefa o classificador Haar Cascade, um método baseado em um algoritmo de deteção de objetos utilizado para identificar faces em uma imagem ou em um video em tempo real. Utilizou-se também modelos de redes neuronais de aprendizagem profunda com redes neuronais convulocionais pre-treinadas para deteção de faces.

A segunda fase do projeto baseia-se na recognição de faces. A recognição de faces tem como objetivo identificar ou confirmar a identidade de um individuo através das sua face, esta técnologia pode ser utilizada para diversas funções, como por exemplo um sistema que permita o desbloqueio automático do telemovel de um individuo através da sua recognição facial.

O objetivo desta tarefa foi conseguir a partir de um conjunto de faces identificadas (conjunto de treino) e um conjunto de faces não identificadas (conjunto de teste), pertencentes às mesmas pessoas, identificar cada pessoa no conjunto de teste. Foram utilizados os métodos *EigenFaces* e *FisherFaces* para reduzir a dimensionalidade do conjunto de dados.

A ultima fase do projeto foi realizada a combinação de objetos virtuais e reais. Nesta fase o objetivo foi conseguir adicionar objectos virtuais à face de um individuo em tempo real, como por exemplo um chapéu.

Para conseguir realizar as diferentes fases do projeto como por exemplo a deteção de faces e o reconhecimento facial foi utilizada a bilioteca OpenCV do Python, uma biblioteca *open source* para Visão Computacional e aprendizagem automática.

2. Desenvolvimento

1. Deteção de faces

Nesta etapa realizou-se a deteção de faces em tempo real e imagens, identificado também os olhos. Foram utilizados algoritmos de aprendizagem automática para encontrar faces dentro de imagens ignorando outros objetos não faciais, como elementos do fundo da imagem, por exemplo edifícios e arvores.

1.1 Classificador *Haar Cascade*

O primeiro método utilizado para esta tarefa foi o classificador *Haar Cascade*, este método baseia-se em treinar um modelo para perceber o que é e o que não é uma face. Depois de ser treinado, o modelo extrai *features* especificas para conseguir detetar faces, que são gravadas em um ficheiro de forma às *features* de novas imagens possam ser comparadas com as *features* do modelo treinado.

Os modelos estão guardados em ficheiros XML e foram lidos com métodos do OpenCV, nomeadamente o método CV2.CascadeClassifier. Foi utilizado um modelo treinado para detetar as faces e outro para detetar os olhos.

Foram realizados programas para testar o desempenho deste tipo de classificador em imagens e em vídeo em tempo real.

Começou-se por detetar as faces, inicialmente encontram-se as faces, os seus tamanhos, desenham-se retângulos e guarda-se o ROI (região de interesse), através do modelo XML. De seguida são detetados os olhos dentro das coordenadas encontradas para a face, isto resolve o problema de encontrar objetos parecidos com olhos fora da face. Nas figuras seguintes podem ser observados exemplos da deteção realizada.

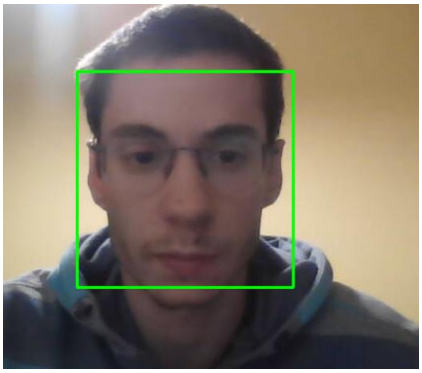

Figura 1 - deteção de faces em tempo real

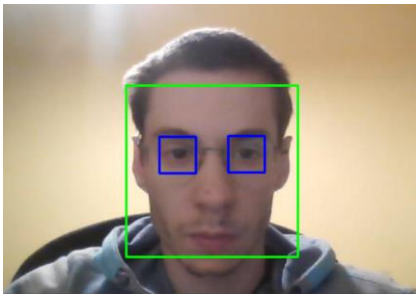

Figura 2 - Deteção de faces e olhos em tempo real

Este classificador tem a vantagem de ter um tempo de computação rápido, porém, teve alguns problemas. O modelo não funciona tão bem se a face estiver coberta com um objeto como uma máscara e um cachecol ou se a cara não estiver propriamente orientada, como se pode observar na figura seguinte.

Figura 3 - Falha ao detetar a face devido à orientação

1.2 *Deep Neural Network module* (dnn)

O segundo método utilizado para esta tarefa foi o *Deep Neural Network module* (dnn). O OpenCV tem um modelo de redes neuronais de aprendizagem profunda com redes neuronais convolucionais pre-treinadas para deteção de faces. Foi utilizado, entretanto, um modelo Caffe pre-treinado para detetar faces de diferentes ângulos.

Esta arquitetura é baseada na arquitetura ResNet com um *Single Shot Detector* (SSD) para a fase de deteção. A SSD precisa apenas de um shot para detetar múltiplos objetos dentro da imagem, tornando-se consideravelmente mais rápida.

Uma CNN é um tipo de redes neuronais que é especificamente direcionada para o processamento de *data* de pixéis.

Inicialmente foi necessário carregar o modelo Caffe, de seguida é necessário carregar a imagem de cada *frame* e criar um *blob*, de forma a detetar faces passa-se o *blob* para a rede obtendo assim as deteções.

O classificador extrai também um valor de confiança para a deteção das faces. A partir deste valor de confiança foi estabelecido um limiar de valor de confiança mínima de 50 %, de forma a filtrar para fora deteções fracas. São desenhados retângulos nas faces e é exposto o valor da confiança das deteções como pode ser observado na seguinte figura.

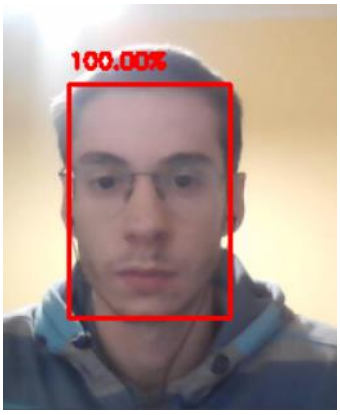

Figura 4 - Deteção de faces em tempo real com dnn

Este classificador teve um melhor desempenho no processo de deteção das faces do que o mencionado anteriormente. Este classificador tem a vantagem de conseguir detetar faces mesmo estando cobertas com objetos como mascaras ou mesmo se as caras não estiverem com uma orientação completamente direita. Este classificador lida também melhor com condições de luzes menos favoráveis, como por exemplo sombras a cobrirem parte da cara.

Figura 5 - Deteção de faces em tempo real com dnn sem orientação ideal

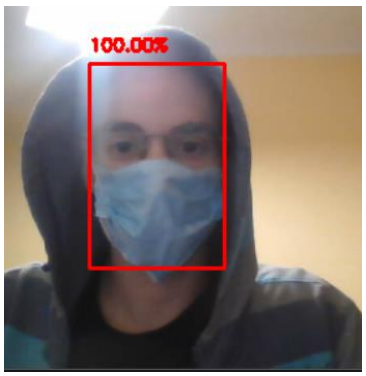

Figura 6 - Deteção de faces em tempo real com dnn com mascara

1.3 Observações

No ponto seguinte, de forma a conseguir detetar as faces para poder aplicar métodos de recognição de faces irá ser utilizado o classificador *Deep Neural Network module* (dnn). Porém de forma a obter as coordenadas dos olhos, para conseguir realizar as operações necessárias posteriormente na normalização, irá ser utilizado o Haar Cascade Classifier. A deteção de olhos é apenas realizada dentro das coordenadas das faces encontradas pelo classificador *Deep Neural Network module* (dnn).

2. Recognição de faces

2.1 Carregar a base de dados de caras

Nesta etapa foi necessário realizar uma base de dados com imagens de caras de alunos, de forma a estas imagens poderem ser posteriormente utilizadas no processo de classificação como o conjunto de dados.

2.2 Normalização das faces

De modo a conseguir utilizar as imagens carregadas, no processo de classificação, primeiro é necessário normalizar as imagens para estas estarem de acordo com as recomendações do MPEG-7.

Para cumprir este objetivo é necessário rodar, escalar e selecionar as imagens de forma a estas possuírem 256 níveis de cor, sendo assim imagens monocromáticas com apenas níveis de cinzento. As imagens têm que ter 56 linhas por 46 colunas, com ambos os olhos, esquerdo e direito, alinhados perfeitamente horizontalmente, e localizados na linha 24, coluna 16 e 31 respetivamente.

2.2.1 Rotação da imagem

A primeira parte realizada no processo de normalização foi verificar que ambos os olhos, esquerdo e direito, estão perfeitamente alinhados na horizontal. De forma a conseguir alinhar os olhos horizontalmente é necessário primeiro obter as coordenadas dos olhos e o pixel central entre os dois olhos. A partir deste pixel central é traçada uma reta entre os dois olhos, para que seja possível calcular o angulo entre o olho esquerdo e o olho direito, como se pode observar nas seguintes figuras:

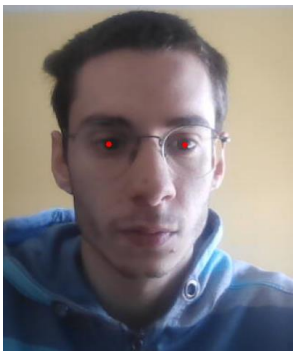

Figura 7 – Deteção do centro dos olhos

Figura 8 - Deteção da linha entre os centros dos olhos

Tendo o centro entre os dois olhos é possível calcular a inclinação da reta e a distância entre dois pontos. Os dois pontos dos olhos traçam entre si um triangulo retângulo, então é possível calcular a inclinação (hipotenusa) entre os dois olhos através do teorema de Pitágoras, como se pode observar na seguinte figura:

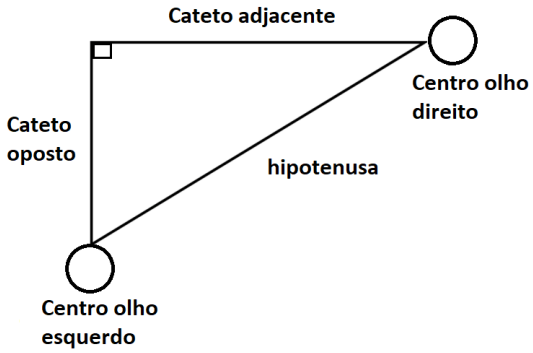

Figura 9 - Relação triângulo retangular entre o cento dos olhos

A fórmula utilizada para calcular a hipotenusa, segundo o teorema de Pitágoras foi a seguinte:

$$hipotenusa = \sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2}$$

De seguida foi calculada a distância entre os pontos na horizontal, a partir da seguinte fórmula:

$$\sin \alpha = \frac{cateto\ oposto}{hipotenusa}$$

Por fim foi calculado o ângulo entre os dois pontos, a partir da seguinte fórmula:

$$\hat{a}ngulo\_x\_y = sin^{-1}(\sin \alpha)$$

A partir do ângulo obtido entre os centros dos dois pontos é agora possível realizar a rotação da imagem mantendo sempre os olhos alinhados na horizontal. De forma a decidir se a rotação deve ser realizada para a esquerda ou para a direita, verifica-se se o ponto y do olho da esquerda é superior ou inferior ao ponto y do olho da direita, a rotação é realizada no sentido do ponto superior.

A rotação é realizada a partir da criação de uma matriz de transformação M que vai ser utilizada para rodar a imagem, onde é passado o centro e o angulo da rotação. De seguida é realizada uma transformação afim onde se preserva as linhas e o paralelismo.

Como a imagem foi rodada é necessário voltar a encontrar as coordenadas do centro dos olhos, de forma a conseguir manter uma linha reta entre o olho esquerdo e direito, isto foi realizado através da função np.matmul do Numpy, que permite realizar o produto matricial entre o *array* da matriz de transformação M e as coordenadas centrais de cada olho.

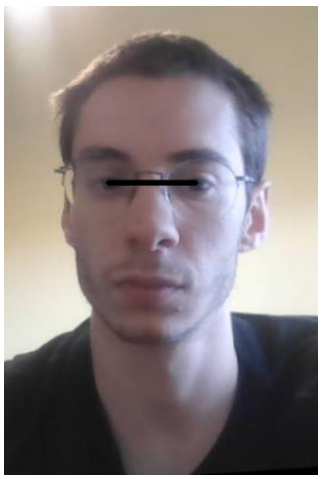

Figura 10 - imagem rodada com os olhos alinhados

2.2.2 Reescalonamento da imagem

De seguida é necessário realizar o reescalonamento da imagem, com o intuito da imagem ter as dimensões 56 x 46. Tendo as coordenadas atualizadas dos olhos é necessário reduzir drasticamente as dimensões das imagens. Realiza-se então a divisão das dimensões da imagem por 16. Na seguinte figura pode-se observar a imagem reescalonada.

Figura 11 - Imagem reescalada

2.2.3 Filtrar a face na imagem

É necessário agora realizar uma filtração de forma à imagem conter apenas a face encontrada. A partir da nova escala da imagem é realizado um reescalonamento das dimensões dos olhos. Foi necessário alinhar os olhos perfeitamente na linha 24, na coluna 16 e 31. A partir das coordenadas dos olhos são definidos os pixéis da imagem até se formar uma imagem 56 x 46. Por fim foi necessário converter a imagem para uma imagem monocromática.

Figura 12 - Imagem com formato adequado final

2.2.4 Criação da base de dados de imagens

A partir do processo de normalização são finalmente criadas 9 imagens todas com o mesmo formato que irão servir como o conjunto de dados.

2.3 *EigenFaces*

De forma a conseguir reduzir a dimensionalidade das faces foram utilizados os métodos *EigenFaces* e *FisherFaces.* Estes métodos utilizam técnicas de redução de dimensionalidade através da combinação de características pela projeção num subespaço.

O EigenFaces calcula os melhores discriminantes sem ter conhecimento de classes, sendo assim um método não supervisionado.

O método *EigenFaces* utiliza *EigenVectores* e *EigenValues* para reduzir a dimensionalidade e projetar um conjunto de treino em um espaço de *features*.

Inicialmente as imagens do conjunto de dados encontram-se com as resoluções 56 x 46. É necessário primeiro realizar um *reshape* nas imagens para estas ficarem em um vetor de 2576 (56 x 46) por 1.

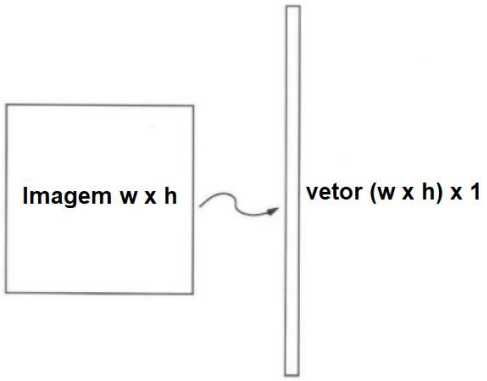

Figura 13 - *reshape* das imagens

De seguida é necessário realizar a média de todas as imagens e subtrair por cada imagem, criando assim a matriz A (2576, N), sendo N o número de imagens.

Apôs isto, foram calculados os *Eigenvalues* e *Eigenvectors*, a partir da função *linalg.eig* do NumPy, da seguinte matriz R:

$$R = A^T A$$

São calculados agora os "m" maiores *Eigenvectors*, sendo "m" no máximo N-1, obtendo assim uma matriz V.

Calcula-se agora uma matriz W a partir do produto interno entre a matriz A e a matriz V. A matriz W tem a seguinte fórmula:

$$W = AV$$

Foi necessário também normalizar os valores de W. Realizando o produto interno ente W e o W transposto, o resultado é uma matriz identidade, que prova que os vetores formam uma base ortogonal.

Por fim é necessário obter os vetores das *features* das projeções no subespaço de faces, a partir da seguinte fórmula:

$$\mathbf{y} = W^T(\mathbf{x}\text{-}\boldsymbol{\mu})$$

De seguida para realizar a projeção de uma das faces do conjunto de treino e observar a sua reconstrução utilizou-se a seguinte fórmula:

$$\text{imagem reconstruída} = Wy + \mu$$

O resultado da reconstrução para uma imagem do conjunto de dados foi o seguinte:

Figura 14 - Imagem original

Figura 15 - Imagem reconstruida

2.4 *FisherFaces*

No método *FisherFaces* é necessário inicialmente obter as *labels* das diferentes classes do conjunto de treino.
O *EigenFaces* calcula os melhores discriminantes com conhecimento prévio de classes, sendo assim um método supervisionado.

Inicialmente como foi realizado anteriormente é feito um *reshape* das imagens para estas ficarem em um vetor de 2576 (56 x 46) por 1.

De seguir foi calculada a média para todas as faces do conjunto de dados e a média para as faces por classe. Tendo as médias das classes foram criadas as matrizes S_b e S_w.

Na matriz S_w é necessário percorrer cada imagem da classe realizando um somatório, subtraindo cada imagem pela média da sua classe, de seguida é realizado um produto interno entre essa subtração e a subtração transposta sendo esse produto interno somado para cada classe percorrida.

Na matriz S_b é necessário percorrer cada *label* de classe no conjunto de dados, realizando uma subtração entre a média das faces e a média da *label*. De seguida é realizado um produto interno entre essa subtração e essa subtração transposta, sendo esse produto interno somado para cada classe percorrida.

Apôs o cálculo das matrizes, é calculada a matriz Wpca através do método utilizado no EigenFaces para calcular o W.

De seguida são calculadas as matrizes $\hat{S}_b$ e $\hat{S}_w$ através da seguinte fórmula:

$$\breve{S}_b = W_{pca}^T S_b W_{pca}$$
$$\breve{S}_w = W_{pca}^T S_w W_{pca}$$

São determinados os c-1 "maior *eigenvectores*" a partir das matrizes e por fim é calculado o W como no método anterior.

Por fim é necessário obter também os vetores das *features* das projeções no subespaço de faces, a partir da seguinte fórmula:

$$\mathbf{y} = W^T(\mathbf{x}\text{-}\boldsymbol{\mu})$$

2.5 Classificação

No processo de classificação foi desenvolvido um classificador *Nearest-Neighbor* (NN) com o número de vizinhos igual a 3 por predefinição. O classificador baseia-se nos vetores de *features y* resultantes dos métodos *EigenFaces* e *FisherFaces*, para realizar o processo de classificação.

Foi realizada uma classe para classificar com o método *EigenFaces* e outra com o método *FisherFaces*. Ambas as classes têm um método que permite realizar o "*fit"* dos dados e outro método que permite fazer "*predict"* de um conjunto de imagens, retornando um conjunto de *labels* preditas.

Os classificadores foram treinados com um conjunto de imagens, com faces de 4 pessoas diferentes, que são no total um conjunto de 36 imagens.

O modelo foi treinado com todas as 36 imagens, de seguida pretende-se classificar em tempo real utilizando a imagem que retorna da normalização previamente realizada.

A partir da *label* predita é possível classificar em tempo real a pessoa, como pode ser observado na seguinte figura:

Figura 16 - Classificação em tempo real

3. Combinação de objetos reais e virtuais

A ultima fase do projeto foi realizada a combinação de objetos virtuais e reais. Nesta fase o objetivo foi conseguir adicionar objectos virtuais à face de um individuo em tempo real, neste caso chapéus. Consoante a classificação obtida a partir do classificador são atribuidos diferentes chapéus à pessoa. Tem-se no total 4 objetos diferentes devido ao conjunto de classes utilizado para o classificador ter tamanho de 4.

3.1 Normalização do objeto virtual

O primeiro passo foi a normalização do objeto virtual, realizando um escalamento do objeto consoante o tamanho da face.

A partir das coordenadas encontradas para a face, é defenida a altura e a largura do objeto virtual. A altura do chapéu é defenido como o comprimento da face a dividir por 1.55, um valor que permitiu obter bons resultados. O objeto no eixo X é centrado na cara, de forma a manter-se sempre no individuo. Foram realizados outros ajustes de forma a obter boas dimensões quer em largura e altura. Foi também necessário verificar que o objeto não sai para fora das bordas da imagem.

A partir das coordenadas obtidas obtem-se uma porção da face onde vai ser adicionada a mascára para se poder adicionar o objeto virtual.

3.1 Adição da máscara

De forma a conseguir adicionar o objeto virtual é necessário adicionar uma máscara. O primeiro processo foi converter a imagem para preto e branco, definindo um valor para o limiar em que o pixel passe para branco ou para preto. Foi também realizado um "*bitwise_and"* filtrando apenas a área de interesse e mantendo o resto com fundo preto. É também aplicada uma máscara inversa para definir os pixéis como preto na área onde vai ser colocado o objeto, finalmente é adicionada a imagem do objeto no local de interesse. Na figura seguinte pode-se observar a imagem resultante em tempo real com o objeto virtual.

Figura 17 - Imagem resultante com objeto virtual

3. Conclusões

A partir da realização deste trabalho prático foi possivél aplicar e estudar diferentes métodos e conceitos relacionados com deteção e recochimento facial. No trabalho prático foi realizado com sucesso uma aplicação que junta deteção e reconhecimento facial permitindo a inclusão de objetos virtuais com base na posição da face de uma pessoa.

Na area de deteção facial foram abordados diferentes métodos. O método do classificador *Haar Cascade* ofereceu a vantagem de ter um tempo de computação rápido e teve um desempenho decente apesar de ser um método mais antigo. O outro método estudado, o *Deep Neural Network module* (dnn) teve um melhor desempenho na deteção das faces, tendo entretanto um tempo de computação mais lento.

Na área de reconhecimento facial foi consolidada a necessidade de normalizar as imagens e ajustar as imagens de forma a estas estarem de acordo com um padrão predefinido, para poder ser realizadas operações como classificação corretamente sobre estas imagens.

No projeto foi abordado a necessidade de métodos que permitem a redução da dimensionalidade do espaço de dados como o *EigenFaces* e o *FisherFaces*. Foram consolidados tópicos de classificação com base em um conjunto de classes, tendo um conjunto de dados composto por imagens de diferentes alunos da turma.

Um aspeto muito importante para a realização do projeto foi a necessidade da normalização de um objeto virtual para este poder estar alinhado com um objeto real, sendo utilizado máscaras para adicionar uma imagem a outra imagem.

4. Bibliografia

Arnaldo Abrantes, Visão Artíficial e Realidade Mista, Face Recognition,

Pedro Mendes Jorge, Visão Artíficial e Realidade Mista, Introduction to Augmented Reality

ML | Face Recognition Using Eigenfaces (PCA Algorithm), em:

https://www.geeksforgeeks.org/ml-face-recognition-using-eigenfaces-pca-algorithm/