PLP-Lab-Aula 6 - Detecção de faces em imagens

Uma das grandes habilidades dos seres humanos é a capacidade de rapidamente identificar padrões em imagens. Isso sem dúvida foi crucial para a sobrevivência da humanidade até os dias de hoje. Busca-se desenvolver a mesma habilidade para os computadores através da visão computacional e várias técnicas foram criadas nos últimos anos visando este objetivo.

O que há de mais moderno (estado da arte) atualmente são as técnicas de “deep learning” ou em uma tradução livre “aprendizado profundo” que envolvem algoritmos de inteligência artificial e redes neurais para treinar identificadores.

Outra técnica bastante utilizada e muito importante são os “haar-like cascades features” que traduzindo seria algo como “características em cascata do tipo haar” já que a palavra “haar” não possui tradução pois o nome é derivado de “wavelets Haar” que foram usados no primeiro detector de rosto em tempo real. Essa técnica foi criada por Paul Viola e Michael J. Jones no artigo Rapid Object Detection using a Boosted Cascade of Simple Features de 2001. O trabalho foi melhorado por Rainer Lienhart e Jochen Maydt em 2002 no trabalho An Extended Set of Haar-like Features for Rapid Object Detection.

As duas referências seguem abaixo:

Paul Viola and Michael J. Jones. Rapid Object Detection using a Boosted Cascade of Simple Features. IEEE CVPR, 2001. The paper is available online at http://research.microsoft.com/en-us/um/people/viola/Pubs/Detect/violaJones_CVPR2001.pdf

Rainer Lienhart and Jochen Maydt. An Extended Set of Haar-like Features for Rapid Object Detection. IEEE ICIP 2002, Vol. 1, pp. 900-903, Sep. 2002. This paper, as well as the extended technical report, can be retrieved at http://www.multimedia-computing.de/mediawiki//images/5/52/MRL-TR-May02-revised-Dec02.pdf

A principal vantagem da técnica é a baixa necessidade de processamento para realizar a identificação dos objetos, o que se traduz em alta velocidade de detecção.
Historicamente os algoritmos sempre trabalharam apenas com a intensidades dos pixels da imagem. Contudo, uma publicação de Oren Papageorgio "A general framework for object detection" publicada em 1998 mostrou um recurso alternativo baseado em Haar wavelets em vez das intensidades de imagem. Viola e Jones então adaptaram a ideia de usar ondas Haar e desenvolveram as chamadas Haar-like features ou características Haar. Uma característica Haar considera as regiões retangulares adjacentes num local específico (janela de detecção) da imagem e então se processa a intensidades dos pixel em cada região e se calcula a diferença entre estas somas. Esta diferença é então usada para categorizar subseções de uma imagem.
Por exemplo, digamos que temos imagens com faces humanas. É uma característica

comum que entre todas as faces a região dos olhos é mais escura do que a região das bochechas. Portanto, uma característica Haar comum para a detecção de face é um conjunto de dois retângulos adjacentes que ficam na região dos olhos e acima da região das bochechas. A posição desses retângulos é definida em relação a uma janela de detecção que age como uma caixa delimitadora para o objeto alvo (a face, neste caso).
Na fase de detecção da estrutura de detecção de objetos Viola-Jones, uma janela do tamanho do alvo é movida sobre a imagem de entrada, e para cada subseção da imagem é calculada a característica do tipo Haar. Essa diferença é então comparada a um limiar aprendido que separa não-objetos de objetos. Como essa característica Haar é apenas um classificador fraco (sua qualidade de detecção é ligeiramente melhor que a suposição aleatória), um grande número de características semelhantes a Haar são necessárias para descrever um objeto com suficiente precisão. Na estrutura de detecção de objetos Viola-Jones, as características de tipo Haar são, portanto, organizadas em algo chamado cascata de classificadores para formar classificador forte. A principal vantagem de um recurso semelhante ao Haar sobre a maioria dos outros recursos é a velocidade de cálculo. Devido ao uso de imagens integrais, um recurso semelhante a Haar de qualquer tamanho pode ser calculado em tempo constante (aproximadamente 60 instruções de microprocessador para um recurso de 2 retângulos).
Uma característica Haar-like pode ser definida como a diferença da soma de pixels de áreas dentro do retângulo, que pode ser em qualquer posição e escala dentro da imagem original. Esse conjunto de características modificadas é chamado de características de 2 retângulos. Viola e Jones também definiram características de 3 retângulos e características de 4 retângulos. Cada tipo de recurso pode indicar a existência (ou ausência) de certos padrões na imagem, como bordas ou alterações na textura. Por exemplo, um recurso de 2 retângulos pode indicar onde a borda está entre uma região escura e uma região clara.

A OpenCV já possui o algoritmo pronto para detecção de Haar-like features, contudo, precisamos do arquivo XML que é a fonte dos padrões para identificação dos objetos. A OpenCV oferece arquivos prontos que identificam padrões como faces e olhos. Em github.com/opencv/opencv/tree/master/data/haarcascades é possível encontrar outros arquivos para identificar outros objetos.

Abaixo veja um exemplo de código que utiliza a OpenCV com Python para identificar faces. O código abaixo foi escrito no ambiente Google Colab e a imagem utilizada no exemplo, empresa.jpg, pode ser encontrada no material disponibilizado para esta aula.

```
# Importa a biblioteca openCV
from google.colab.patches import cv2_imshow
import cv2

# Carrega a imagem "empresa.jpg" na variável img e mostra a imagem
img = cv2.imread("empresa.jpg")
cv2_imshow(img)

# Transforma a imagem em tons de cinza, o que facilita a aplicação
do classificador
imgPB = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2_imshow(imgPB)

# Cria um objeto do classificador usando seu construtor
# O parâmetro recebido pelo construtor é o endereço do arquivo xml
que guarda os padrões de identificação
```

```
df = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')

# Utiliza função de detecção a partir do objeto df
faces = df.detectMultiScale(imgPB, scaleFactor=1.05,
minNeighbors=7,minSize=(30,30), flags = cv2.CASCADE_SCALE_IMAGE)

# Desenha retângulos amarelos nas posições em que foram encontradas
faces
for(x,y,w,h) in faces:
  cv2.rectangle(img, (x,y), (x+w,y+h), (0,255,255), 7)

# Mostra a imagem atualizada com a marcação dos locais onde foram
encontradas faces
cv2_imshow(img)
```

Os argumentos que precisam ser informados para o método de detecção além do arquivo XML que contém a descrição do objeto seguem abaixo:

- ScaleFactor: Quanto o tamanho da imagem é reduzido para cada face encontrada. Isso é necessário porque o tamanho da imagem pode influenciar nos padrões de detecção do arquivo .xml. Um valor de 1,05 indica que a imagem será reduzida em 5% de cada vez.

- minNeighbors: Quantos vizinhos (padrões de detecção) cada janela deve ter para a área ser considerada um rosto. O classificador em cascata detectará vários padrões ao redor da face e este parâmetro controla quantos são necessárias para considerar aquela área como um rosto válido.

- minSize: Uma tupla de largura e altura (em pixels) indicando o tamanho mínimo da janela para que caixas menores sejam ignoradas.

Detecção de faces em vídeos

O processo de detecção de objetos em vídeos é muito similar a detecção em imagens. Um vídeo nada mais é do que um fluxo contínuo de imagens que são enviadas a partir de uma fonte, como uma webcam ou ainda um arquivo de vídeo mp4.

Um looping é necessário para processar o fluxo contínuo de imagens do vídeo. É importante apontar também que o Google Colab não possui acesso à webcam do computador, então não poderemos realizar este exemplo utilizando o Google Colab. Para este exemplo, utilizaremos o VS Code. Em uma IDE como o VS Code, a instalação do pacote OpenCV deve ser feita de forma manual.

Primeiro criamos uma pasta de trabalho no VS Code e em seguida abrimos um terminal nesta pasta, clicando no menu “Terminal” e na opção “Novo Terminal”. Em seguida, executamos o comando **pip install opencv-python** e aguardamos o final da instalação.

Em seguida criamos um arquivo **main.py** e digitamos o código abaixo, para exemplificar a detecção de faces em vídeo. Note que ao criarmos um objeto de câmera passando o parâmetro 0 para a função “.VideoCapture(0)”, estamos acessando a câmera do próprio computador e não um arquivo de vídeo. O arquivo “vídeo.mp4” utilizado no exemplo foi disponibilizado no material desta aula e deve estar na pasta de trabalho do VS Code, juntamente com o arquivo main.py. Para executar o código abaixo, utilize o comando **python main.py** no terminal.

```
import cv2

# Utiliza o construtor CascadeClassifier para criar o objeto df,
que guarda as funções de um algoritmo de classificação em cascata
df = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')

# Cria o objeto de camera considerando um arquivo de vídeo
camera = cv2.VideoCapture("video.mp4")

# Cria o objeto camera considerando a webcam do computador
#camera = cv2.VideoCapture(0)

while True:
    # A função .read() retorna dois valores: 1- Um valor booleano
identificando se um frame foi encontrado;
    # 2- Uma matriz de pixels representando o frame do vídeo que
foi encontrado
    (sucesso, frame) = camera.read()

    if not sucesso: #final do video
        break

    # Converte o frame para tons de cinza
    frame_pb = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)

    # Detecta as faces no frame
    faces = df.detectMultiScale(frame_pb, scaleFactor = 1.1,
minNeighbors=15, minSize=(30,30), flags=cv2.CASCADE_SCALE_IMAGE)

    # Faz uma cópia do frame
    frame_temp = frame.copy()

    # Desenha retângulos amarelos, na imagem frame_temp, nas
posições onde foram encontradas faces
    for (x, y, lar, alt) in faces:
        cv2.rectangle(frame_temp, (x, y), (x + lar, y + alt), (0,
255, 255), 2)
```

```
    cv2.imshow("Encontrando faces...", frame_temp)

    # Espera que a tecla 's' seja pressionada para sair
    if cv2.waitKey(1) & 0xFF == ord("s"):
        break

# Desaloca a memória do objeto camera e fecha todas as janelas
abertas pela biblioteca OpenCV
camera.release()
cv2.destroyAllWindows()
```