FACULDADE IMPACTA TECNOLOGIA

Carolina Gabrielle Castro Vieira	**1900127**
Gabriela Rodrigues Oliveira Lima	**1903020**
Roberta Yumi Romero Takahashi	**1903220**

DETECÇÃO DE FACES EM IMAGENS

São Paulo

2023

1. INTRODUÇÃO

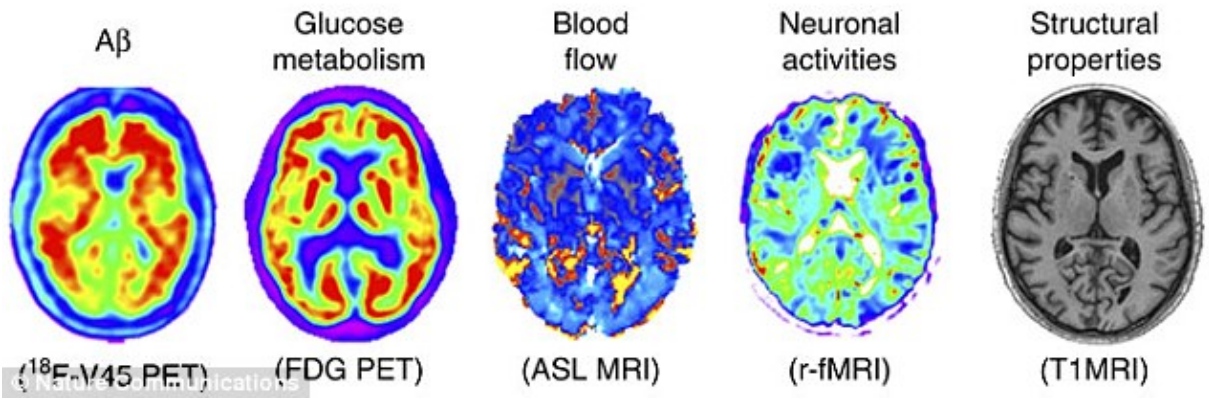
Nos últimos anos, o processamento de imagens vem se tornando uma área cada vez mais relevante na ciência da computação e em várias outras áreas. Com a evolução da tecnologia de câmeras digitais e dispositivos móveis, o número de imagens geradas e armazenadas em todo o mundo tem crescido exponencialmente, tornando a análise e processamento dessas imagens cada vez mais desafiadoras.

A capacidade de extrair informações úteis a partir de imagens é muito valiosa em muitos campos, como medicina, biologia, engenharia, indústria e entretenimento. No entanto, a análise manual de imagens é uma tarefa demorada e propensa a erros, especialmente quando se lida com grandes quantidades de dados. É aqui que entra o processamento de imagens computacional, que utiliza algoritmos de computação para automatizar a análise de imagens, tornando-a mais rápida, precisa e eficiente.

Uma das técnicas mais populares em processamento de imagens é a análise de características, que envolve a extração de recursos significativos das imagens para realizar uma variedade de tarefas, como classificação, segmentação, reconhecimento de objetos e detecção de anomalias. Esses recursos podem ser extraídos usando algoritmos de aprendizado de máquina, que treinam um modelo a reconhecer padrões nas imagens.

Um exemplo de aplicação de processamento de imagens na medicina é a análise de imagens médicas, que pode ser usada para auxiliar na detecção e diagnóstico de várias doenças. A análise de imagens pode ser usada para segmentar órgãos ou tecidos em imagens médicas e identificar anomalias ou áreas suspeitas. Além disso, a análise de imagens médicas pode ser usada para monitorar o progresso da doença e avaliar a eficácia do tratamento.

Figura 1 – Imagens processadas de possível primeiro sinal de Alzheimer.

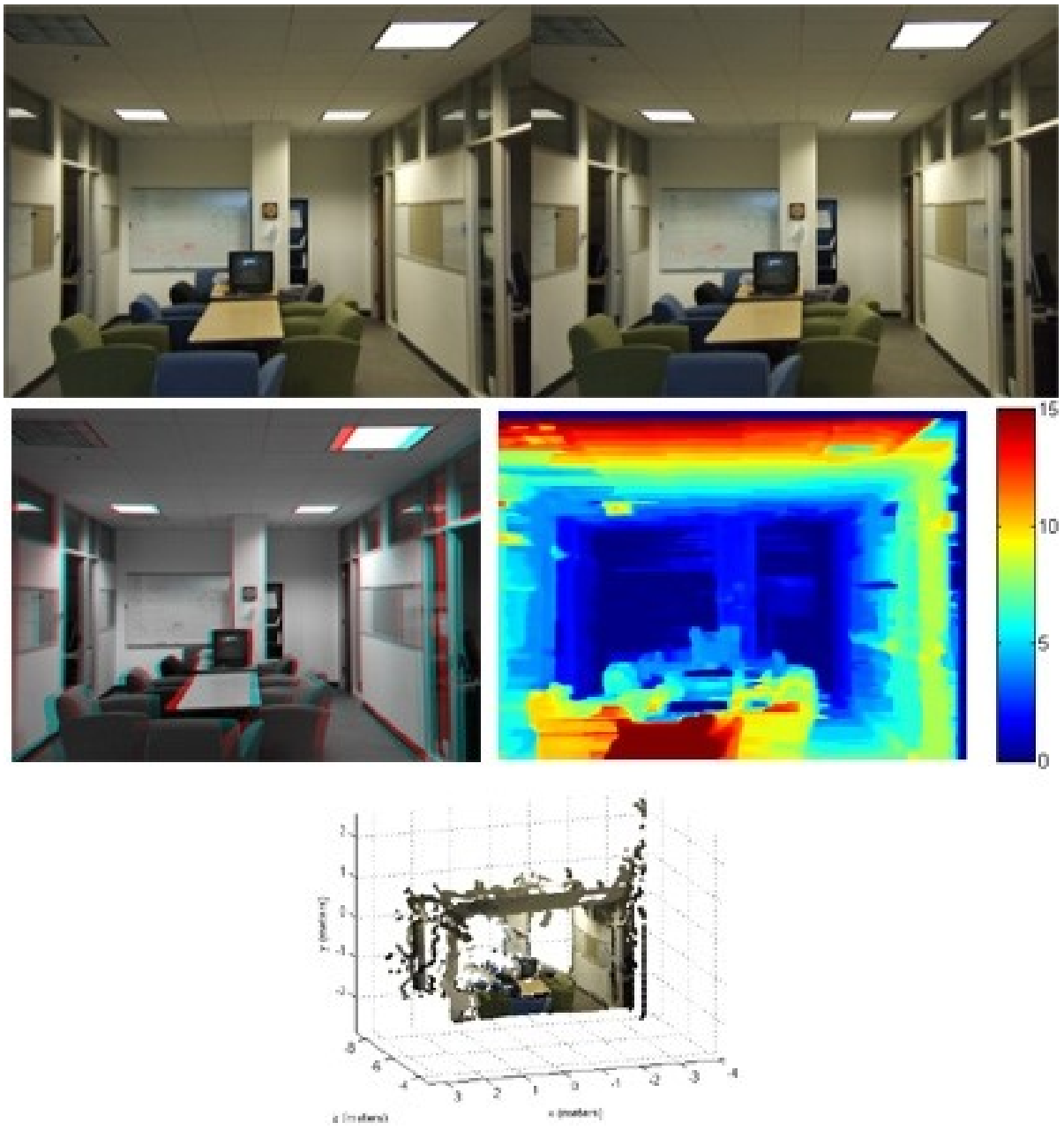


Fonte: Noticia Alternativa (https://noticiaalternativa.com.br/sinal-de-alzheimer/).

Outra aplicação importante do processamento de imagens é na indústria, onde pode ser usado para inspecionar a qualidade do produto e detectar defeitos. O processamento de imagens pode ser usado para inspecionar a aparência de peças e produtos, procurando por defeitos como rachaduras, riscos, manchas ou outras irregularidades. Isso pode ser particularmente útil em setores como fabricação de automóveis, eletrônicos e embalagens.

O processamento de imagens também tem sido amplamente utilizado no campo da robótica, onde pode ser usado para permitir que robôs vejam e entendam o mundo ao seu redor. A visão computacional pode ser usada para ajudar robôs a detectar objetos, reconhecer pessoas e seguir caminhos. Além disso, a visão computacional é importante para robôs autônomos, que devem ser capazes de tomar decisões com base em informações visuais.

Figura 2 – Visão Robótica estuda a reconstrução de cenários 3D a partir de imagens.



Fonte: Universidade da Tecnologia (https://universidadedatecnologia.com.br/computacao-grafica-processamento-de-imagens-visao-computacional-cia/).

Uma das áreas mais promissoras de aplicação do processamento de imagens é o reconhecimento de objetos, que envolve a identificação de objetos em imagens ou vídeos. O reconhecimento de objetos tem uma ampla variedade de aplicações, desde a detecção de rostos em fotos até a classificação de veículos em imagens de satélite. É uma técnica fundamental para a construção de sistemas de vigilância por vídeo e é frequentemente usada em aplicações de segurança e monitoramento.

Além disso, o processamento de imagens também tem um papel importante no campo da realidade virtual e aumentada. O processamento de imagens pode ser usado para criar ambientes virtuais e aumentados mais realistas, onde as imagens são combinadas com elementos virtuais para criar uma experiência mais imersiva. A realidade virtual e aumentada tem um grande potencial em áreas como jogos, treinamento, educação e marketing.

Outra aplicação interessante do processamento de imagens é na área de arte e design. O processamento de imagens pode ser usado para criar efeitos visuais impressionantes em filmes, animações e videogames. Também pode ser usado em design gráfico para criar imagens estilizadas e efeitos visuais interessantes.

Apesar de todas as aplicações potenciais do processamento de imagens, ainda há muitos desafios a serem superados. Um dos maiores desafios é a necessidade de grandes quantidades de dados para treinar modelos de aprendizado de máquina. Além disso, as imagens podem ser muito complexas e variáveis, tornando difícil para os algoritmos de processamento de imagem capturar todas as nuances e informações presentes nas imagens.

Em resumo, o processamento de imagens é uma área em rápida evolução que tem uma ampla variedade de aplicações em muitas áreas, incluindo medicina, indústria, robótica, entretenimento, entre outros. O processamento de imagens é fundamental para a análise automatizada de imagens, permitindo que os pesquisadores extraiam informações úteis de grandes quantidades de dados de imagem. Apesar dos desafios, o processamento de imagens tem um grande potencial para transformar muitos campos e continuar a moldar o mundo em que vivemos.

OBJETIVO

O objetivo deste trabalho é desenvolver um sistema de visão computacional capaz de detectar a presença de rostos em um vídeo. Para isso, serão utilizadas técnicas de processamento de imagens e aprendizado de máquina para identificar regiões de interesse que contenham rostos humanos e realizar a segmentação dessas regiões. Os objetivos específicos incluem: (1) revisar a literatura existente sobre visão computacional e técnicas de detecção de rostos; (2) selecionar um vídeo para utilização do sistema; (3) desenvolver um modelo de processamento de imagem capaz de detectar faces em vídeos; (4) avaliar o desempenho do sistema.

Ao final do trabalho, espera-se obter um sistema eficiente e preciso para a detecção de rostos em vídeos, a partir de ajustes no código do modelo.

DESCRIÇÃO

O projeto foi desenvolvido utilizando o método Haar Cascade para classificação, e pra isso, foi preciso fazer o download de uma dataframe em planilha, disponibilizado no github pelos desenvolvedores da biblioteca OpenCV. Você pode encontrar essa e outras dataframes de classificação (como por exemplo: deteccção de corpo inteiro, sorriso, olhos, face de gatos e etc) em: https://github.com/opencv/opencv/tree/master/data/haarcascades. O modelo utilizado para este projeto foi o **haarcascade_frontalface_default.xml**.

Para dar início ao desenvolvimento do algortimo, já dentro do arquivo de código python, é preciso importar a biblioteca com o comando:

```
import cv2
```

Com estes preparativos prontos, cria-se por fim algumas funções que facilitem e evidenciem algumas rotinas de códigos que serão necessarias ao longo do projeto. A primeira delas é a de redimensionamento de imagem, como você pode ver no seguinte trecho de código:

```
# ---------------------------------------------------------------------
# Função para redimensionar uma imagem
def redim(img, largura):
    alt = int(img.shape[0]/img.shape[1]*largura)
    img = cv2.resize(img, (largura, alt), interpolation=cv2.INTER_AREA)
    return img

# ---------------------------------------------------------------------
```

A segunda é feita para a escrita nas imagens sendo processadas:

```
# ------------------------------------------------------------------------
#Função para facilitar a escrita nas imagem
def escreve(img, texto, cor=(255,0,0)):
    fonte = cv2.FONT_HERSHEY_SIMPLEX
    cv2.putText(img, texto, (12,23), fonte, 0.8, cor, 0, cv2.LINE_AA)

# ------------------------------------------------------------------------
```

Agora, é necessário carregar ambos o dataframe mencionado anteriormente, quanto o vídeo do qual se deseja utilizar como objeto de estudo para o projeto. Para isso, basta utilizar os seguintes comandos:

```
# Cria o detector de faces baseado no XML
df = cv2.CascadeClassifier('frontalface.xml')
```

A função CascadeClassifier cria um objeto classificador em cascata que pode ser usado para detectar objetos específicos em imagens ou vídeos. Ele é baseado em algoritmos de aprendizado de máquina, onde um conjunto de características é usado para treinar um classificador capaz de distinguir entre objetos positivos e negativos. A detecção de rostos pelo Cascade Classifier ocorre em várias etapas, cada uma aplicando um conjunto de filtros e características em cascata para determinar se um objeto é um rosto ou não. Passando por suas etapas, temos:

- Preparação da imagem:

  A imagem de entrada é convertida em tons de cinza, pois a detecção de rosto não depende de cores.
- Classificadores em cascata:

  O classificador em cascata consiste em várias etapas, chamadas de estágios, onde cada estágio contém múltiplos classificadores baseados em características. Cada estágio é projetado para ser rápido, descartando rapidamente as regiões não pertencentes a rostos e focando apenas nas regiões promissoras. O classificador em cascata é construído de forma que estágios subsequentes sejam mais complexos e exigentes, filtrando ainda mais as regiões de interesse.

- Características Haar:

  As características Haar são usadas para descrever as regiões de interesse em uma imagem. Essas características são calculadas em diferentes tamanhos e posições, e podem representar padrões simples, como bordas, linhas ou retângulos, que são comuns em rostos. As características são representadas por diferenças de intensidade entre regiões retangulares adjacentes.
- AdaBoost:

  AdaBoost é um algoritmo de aprendizado de máquina usado para treinar cada classificador em cascata. Cada classificador é treinado com um conjunto de características Haar em uma etapa específica. O AdaBoost atribui pesos às características com base em sua eficácia na distinção entre exemplos positivos e negativos. Os classificadores mais eficazes têm pesos maiores.
- Classificação em cascata:

  Durante a detecção, a imagem é varrida por diferentes janelas de detecção em tamanhos e posições variáveis. Em cada janela, as características Haar são calculadas e o classificador em cascata avalia se a janela contém um rosto ou não. Se uma janela falhar em qualquer estágio do classificador em cascata, ela é descartada. Somente as janelas que passam por todos os estágios são consideradas como regiões contendo rostos. O arquivo XML carregado no CascadeClassifier contém as informações necessárias para definir a estrutura do classificador em cascata treinado. Ele armazena os detalhes sobre o número de estágios, o número de classificadores em cada estágio e os pesos e informações de características associadas a cada classificador.

Ao carregar o arquivo XML no CascadeClassifier, essas informações são usadas para criar o classificador em cascata treinado correspondente, que é então usado para a detecção de rostos.

```
# Abre um vídeo gravado em disco
camera = cv2.VideoCapture('video.mp4')
```

O método VideoCapture( ) pode carregar tanto um arquivo de vídeo, quanto uma sequência de arquivos de imagem ou um dispositivo de captura de vídeo. No caso deste código, é carregado um arquivo de vídeo armazenado na máquina, com o nome “video”, e extensão “mp4”. Internamente, este arquivo de video é dividido em vários arquivos de imagens, que chama-se

de frames. Um frame é básicamente uma imagem estática que compõe a “animação” dos vídeos, e cada vídeo é composto por centenas de imagens.

Também foi criado uma variável para contagem de acurácia acumuladas em cada frame, e seu valor é inicialmente zerado. Suas funcionalidades serão explicadas mais tardiamente.

```
# Cria uma contagem para acerto em cada frame
acertos_frame = 0
```

Por fim, é feito um loop infinito, pois com o vídeo será processado frame por frame, é preciso repetir o método read( ), que será responsável por fazer a leitura dos arquivos de imagem gerados através do vídeo, só assim o próximo frame será carregado na variável “frame”. Assim então, inicia-se o loop e é feito a leitura do decorrente frame.

```
while True:
    # A função read() retorna se houve sucesso e o próprio frame
    (sucesso, frame) = camera.read()
```

A seguir, é preciso de métodos que identifiquem quando parar o loop infinito, para que seu código não permaneça rodando eternamente. Pra isso, foram feitos dois métodos; O primeiro identifica quando o vídeo chega ao final, de acordo com a variável “sucesso” recebida no método read( ), que indica se o carregamento da imagem foi bem-sucedida ou não.

```
# ----------------------------------------------------
# Métodos que possibilitam a quebra do loop:

# 1. Identifica o final do vídeo
if not sucesso:
    break
```

O segundo método, é uma pausa forçada que pode ser triggada a qualquer momento pelo usuários, bastando apenas pressionar a tecla “s”.

```
# 2. Espera que a tecla 's' seja pressionada para sair
if cv2.waitKey(3) & 0xFF == ord("s"):
    break
# ----------------------------------------------------
```

Agora, é necessário fazer ajustes nas dimensões do frame sendo processado, tanto quanto a aplicação de filtros em tons de cinza que facilitam o processamento da imagem.

```
# Reduz tamanho do frame para acelerar processamento
frame = redim(frame, 600)

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

Com isso feito, é hora de finalmente utilizar o método detectMultiScale( ), que fará propriamente a deteccção de faces no frame processado, a partir do dataframe que foi referenciado na variável “df”, e armazenará cada instancia em uma variável chamada “faces”.

```
# Detecta as faces no frame
faces = df.detectMultiScale(frame_pb, scaleFactor=1.1, minNeighbors=6,
                            minSize=(62, 64), maxSize=(110, 115),
                            flags=cv2.CASCADE_SCALE_IMAGE)
```

A função detectMultiScale é usada em conjunto com o CascadeClassifier do OpenCV para detectar objetos em uma imagem. Ela possui vários parâmetros que afetam o desempenho e a precisão da detecção. Vou explicar os principais parâmetros da função detectMultiScale:

- image: Este parâmetro especifica a imagem de entrada na qual deseja-se realizar a detecção de objetos. É uma matriz NumPy que representa a imagem.
- scaleFactor: Este parâmetro controla a taxa de escala da imagem durante a detecção. O valor padrão é 1.1, o que significa que a cada iteração, a imagem é reduzida em 10% do seu tamanho original. Um valor menor aumenta a sensibilidade, mas também aumenta o tempo de processamento.
- minNeighbors: Este parâmetro define o número mínimo de vizinhos que um retângulo candidato deve ter para ser considerado como um objeto válido. Um valor alto garante detecções mais robustas, pois filtra regiões falsas, mas também pode levar a uma menor taxa de detecção. Um valor baixo pode resultar em mais detecções, mas também pode incluir regiões falsas.
- minSize e maxSize: Esses parâmetros definem a faixa de tamanhos de objetos que você deseja detectar. minSize especifica o tamanho mínimo do objeto a ser detectado, enquanto maxSize especifica o tamanho máximo. Você pode usar esses parâmetros para restringir a detecção a objetos de um determinado intervalo de tamanhos.

- flags: Este parâmetro controla vários comportamentos da função detectMultiScale. Você pode fornecer uma combinação de sinalizadores usando operadores de bits (por exemplo, cv2.CASCADE_SCALE_IMAGE cv2.CASCADE_DO_ROUGH_SEARCH). Os sinalizadores permitem personalizar o comportamento da detecção, como o uso de escalas lineares em vez de logarítmicas.

Após a execução da função detectMultiScale, ela retorna uma lista de retângulos delimitadores (x, y, largura, altura) que representam as posições dos objetos detectados na imagem. É importante ajustar esses parâmetros de acordo com o seu cenário específico para obter um bom equilíbrio entre a taxa de detecção e a precisão. Experimentar diferentes valores e observar os resultados ajudará a otimizar a detecção de objetos na sua aplicação.

Foi adicionado também ao projeto um método básico que estima aproximadamente a acurácia do algoritmo de detecção de faces. Pra isso, começamos incrementando a variável "acertos_frame" instanciada anteriormente da seguinte forma:

```
# Faz a contagem da acuracia de detecção no atual frame sendo processado
if len(faces) == 3:
    acertos_frame+=1
elif len(faces) == 2:
    acertos_frame+=0.66
elif len(faces) == 1:
    acertos_frame+=0.33
```

Caso no decorrente frame tenha sido identificado 3 faces, é marcado a pontuação de 1 ao "acertos_frame", pois é a quantidade total de rostos contida no vídeo usado para caso de estudo deste projeto. Caso tenha sido identificado apenas 2 faces, é marcado a pontuação de 0.66, e para apenas 1 face, 0.33.

Agora, é feito uma cópia do atual frame. Isso vai nos permitir desenhar e escrever em cima do frame, sem que o original sofra qualquer alteração.

```
# Faz um cópia temporaria do frame pra não afetar o frame original
frame_temp = frame.copy()
```

Para que seja possível vizualizar os rostos encontrados na detecção de faces com mais facilidade, pode-se criar uma função simples que desenhe no frame copiado um retângulo na área identificada como face pelo algoritmo.

```
# Desenha retangulos amarelos no frame temporario (colorido)
for (x, y, lar, alt) in faces:
    cv2.rectangle(frame_temp, (x, y), (x + lar, y + alt), (0, 255, 255), 2)
```

Aqui, damos continuidade ao método para calcular a acurácia do algoritmo. O vídeo utilizado neste projeto possui um total de 925 frames. Considerando que 925 frames são 100% do vídeo, de acordo com os cálculos realizados, 1 frame equivale a 0.10810810810810811% do vídeo. Assim então, é necessário apenas pegar o valor contido em "acertos_frame" e multiplicar por 0.10810810810810811, assim obtendo o valor em porcentagem da acurácia decorrente do vídeo. Este cálculo é refeito a cada frame, tendo o valor acumulado dos frames anteriores para chegar a um valor x% de 100% do vídeo ao alcançar o seu final.

```
# Escreve e exibe, frame a frame, a atual precisão da detecccção de faces no video
escreve(frame_temp, "Acuracia: " + str("{:.1f}".format(0.10810810810810811 * acertos_frame)) + "%")
# O video possui 925 frames no total
# 1 frame é 0.10810810810810811% de 100% do video
```

O valor calculado é então escrito no canto superior esquerdo do frame copiado para que possa ser vizualizado a cada atualização.

Agora, para que o frame temporario com as alterações de escrita e desenho das áreas de face seja mostrado na tela da máquina, utilizamos

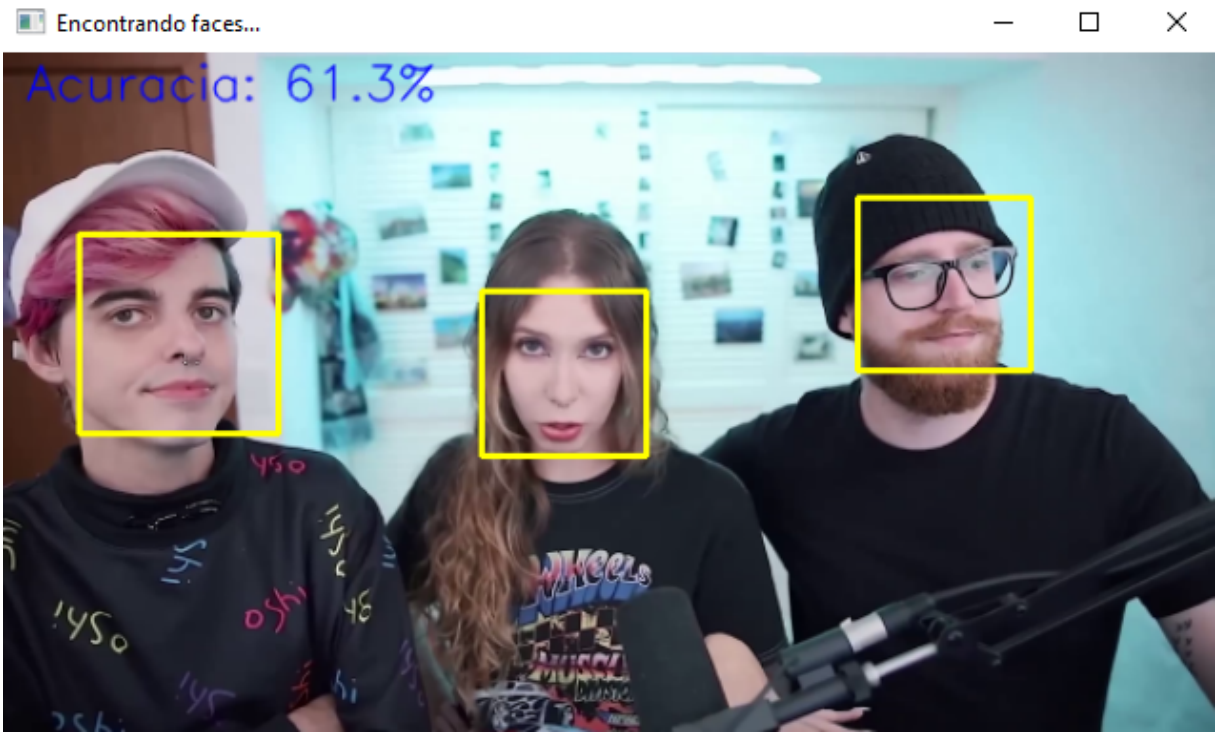
```
# Exibe um frame redimensionado (com perca de qualidade)
cv2.imshow("Encontrando faces...", redim(frame_temp, 640))
```

Por fim, quando todo este processo é finalizado, é necessário utilizar um método que "solte" a câmera utilizada, e então feche as janelas.

```
# --------------------------------------------------------------------------
# Fecha streaming
camera.release()
cv2.destroyAllWindows()
```

RESULTADOS E DISCUSSÃO

Por fim, rodando o código, podemos ter este resultado:



Ao final do vídeo, foi calculado cerca de 74% de acurácia na detecção de faces.

Para as considerações finais, digo que a maior dificuldade que tive no projeto foi ajustar os parâmetros da função “detectMultiScale” para diminuir a margem de falsos positivos ao detectar uma face nos frames, pois isso é um fator que atrapalha inclusive a função feita para calcular a acurácia do algoritmo. Chego a conclusão de que é muito difícil determinar quais são de fato os falsos e verdadeiros positivos que o algoritmo detecta em vídeos, pois para isso, seria necessário que um humano olhasse frame por frame para classificá-los, o que é inviável, pelo trabalho que daria. Ainda assim, o algoritmo se mostrou muito eficaz e, ajustando os parâmetros de funções corretamente, demonstra uma margem de erro relativamente baixa.

REFERÊNCIAS

GONZALEZ, Rafael C.; WOODS, Richard E. **Processamento de imagens digitais. 3ª edição**. Pearson Brasil, 2018.

SZELISKI, Richard. **Computer vision: algorithms and applications**. Springer Science & Business Media, 2010.