Reconhecimento facial utilizando OpenCV na segurança Pública

Anastácio Kemerich[1], Mirkos Ortiz Martins[2]

[1] Acadêmico do Curso de Ciência da Computação – Universidade Franciscana Santa Maria – RS – Brasil.

[2] Professor do Curso de Ciência da Computação - Universidade Franciscana – Santa Maria – RS – Brasil.

{anastacio.kemerich,mirkos}@ufn.edu.br

Abstract. The use of computer vision in the monitoring of crowds has been increasingly evident in several computer solutions around the world. This work is aimed at the application of facial recognition in populations, for the discovery of fugitives from justice, through the use of OpenCV and Haarcascade implemented in the Python Language. For the work, a dataset of 4000 images was used for system training, simulating known faces of fugitives, to be located and notified to the police force. The success rate was 94,56 %, with accuracy of 44,52% for still images and 94.55 % for vídeos.

Resumo. A utilização de visão computacional no acompanhamento de multidões tem sido cada vez mais evidente, em diversas soluções computacionais pelo mundo. Esse trabalho se destina na aplicação do reconhecimento facial em populações, para a descoberta de foragidos da justiça, através do uso de OpenCV e Haarcascade implementados na Linguagem Python. Para o trabalho foi utilizado um dataset de 4000 imagens para treinamento do sistema, simulando rostos conhecidos de foragidos, para serem localizados e avisados à força policial. O índice de acerto foi de 94,58%, com acurácia de 44,52% para imagens estáticas e de 94,55% para vídeos.

1. Introdução

No ano de 2015, engenheiros chineses, em coparticipação com a Universidade Tsinghua (Pequim) e a empresa de segurança *Tzekwan Technology*, criaram um protótipo de caixa eletrônico com reconhecimento facial, cujo tinha a finalidade de redução nos crimes envolvendo o uso de serviços bancários [da SILVA and. da SILVA]. Um outro exemplo de tecnologia facial foi utilizada em Santa Maria - RS, onde, para evitar fraudes no uso das passagens de ônibus, foi instalado um sistema de reconhecimento facial, a fim de certificar que a carteira de passagem está sendo usada pela pessoa certa [Globo 2010].

Em 2018, um caso mostra o uso da tecnologia a favor da segurança, no norte do Brasil, quando um indivíduo foragido há 21 anos era acusado da morte de um publicitário, na cidade de Natal – RN. O mesmo foi localizado após as filmagens de um estabelecimento comercial sendo elas analisadas por um *software* do Laboratório Forense Computacional do (Gaeco). [Terra 1993]

Câmeras instaladas em óculos de policiais chineses foram capazes de identificar todos os rostos de pessoas em uma rua e esta informação foi comparada com uma base

de dados que busca coincidências. As respostas para essa busca foram visualizadas em um dispositivo móvel semelhante a um *tablet*. Com a ajuda desta tecnologia sete pessoas foram presas por diversos crimes, e vinte e seis passageiros foram autuados por portar documentação falsa. Esse sistema foi testado com sucesso, na província de *Henan - China*, na estação do trem-bala [Carneti 2015].

Baseado nesse cenário, esse trabalho tem como objetivo geral desenvolver um software com o intuito auxiliar órgãos públicos na identificação de pessoas, que podem estar com situação de investigação. E para o trabalho, foram determinados os seguintes objetivos específicos:

- Estudar e protótipo uma tecnologia que avalie os padrões de formatos geométricos que formam uma face capturada em imagem;
- Usar uma base de padrões geométricos reconhecidos e classificados, utilizando técnica de visão computacional;
- Testar a capacidade de identificação do reconhecimento facial da ferramenta desenvolvida;
- Desenvolver um *software* capaz de extrair imagens estáticas de vídeos, para posterior teste de reconhecimento facial;
- Comparar vídeos ao vivo (e gravados), *frame* a *frame*, em relação ao reconhecimento de face;
- Avaliar a acurácia de *software* em relação ao reconhecimento;
- Construir uma funcionalidade de alerta, quando houver identificação positiva de face no sistema, aos agentes de segurança pública.

2. Trabalhos Correlatos

Os trabalhos, em questão, utilizam diversas abordagens para realizar a captação, identificação e segmentação por meio de técnicas de processamento de imagem, e diversos outros que sustentaram este estudo.

2.1. Uma Análise do Processo Reconhecimento Facial

Neste projeto [Queiroz de Santana et al. 2014] faz um relato comparativo de reconhecimento da face humana por um ser humano e através de sistemas computacionais. O processo de reconhecimento resume-se na execução de vários passos, como aquisição de imagens, detecção de faces, a segmentação, a extração de características e somente após estas etapas faz a classificação da face. Foi utilizado, também, o método de classificação *K-NN* ou popularmente conhecido como K-vizinhos onde existem diversas métricas para calcular a distância entre dois pontos, como distancia *Mahalanobis*, *Euclidiana*, entre outras.

Para verificação dos resultados foi analisado um banco de dados com duzentos e trinta imagens, com dez imagens de diferentes posições como o tamanho de 92x112 de vinte e três indivíduos, distancia *Mahalanobis* e *Euclidiana* foi aplicada onde obtiveram um índice médio nas duas classes de 76,42%. Segundo [Queiroz de Santana et al. 2014] é um índice aceitável pela estrutura de dados coletados.

2.2. Um Método de Aquisição de Imagens para Reconhecimento de Rosto e implementação de um atendimento automático Sistema de Eventos

Em [Lung et al. 2019] propõe um método de aquisição de imagem para criar uma base de dados chamada *Smart Event Faces* que contem *frames* de vídeo para analisar as faces nelas encontradas. O sistema possui duas pastas, a primeira é a *event-aces* e a segunda é a *smart-aces*, sendo divididas em autorizadas e não autorizadas, para um total de cinquenta e duas pessoas.

Os vídeos dos *smartphones* são para treinamento e os vídeos do *Raspberry* Pi são para o teste, após localizado o rosto, as fotos são recortadas e redimensionadas para 140x140. Todos eram estudantes de graduação, que tinham entre 18 e 30 anos, ambos os sexos, estando cada pessoa vinculada nas duas pastas e um total da base de dados *Smart Event Faces* continha cinco mil e duzentas fotos.

2.3. Um sistema integrado de detecção e reconhecimento de faces

[Koh et al. 1999] mostrou em seu trabalho que com uso do *haarcascade* juntamente com *Python, OpemCv* na analise de *selfie* de usuários da rede social *Instagram*, onde o autor utilizou os marcadores chamados *hashtag*, da rede social *Instagram,* na Indonésia Ásia-Pacífico. Com isso o projeto consiste em detectar se em *Selfies* marcadas com as respectivas *hashtags*: #selfie, #selfiee, #selfies. Para a extração de imagem foi utilizada uma biblioteca chamada *Beautifulsoup,* dela é obtido um arquivo *JSON*[1], o método de *Haar. E*sse processo consiste na soma da intensidade dos *pixels* de regiões brancas das características da pele, subtraído da soma da intensidade do restante cinza da imagem, este resultado é uma hipótese de onde está a face para detectar rostos humanos, a digitalização é feita da parte superior esquerda e termina na inferior direita. Conforme as siglas que seguem, TP (verdadeiros positivos) indica a quantidade de faces detectadas corretas (faces detectadas como faces), TN (verdadeiros negativos) indica a quantidade de faces detectadas corretas (sem faces detectadas como não faces), FP (falsos positivos) indica a quantidade de rostos detectados incorretamente (rostos detectados como não-rostos) e FN (falsos negativos) indica a quantidade de rostos detectados incorretamente (não-rosto detectado como rostos).

$$Precisão = (TP + TN) / (TP + TN + FP + FN)$$

Com este método deu-se uma precisão de 71,48% e nota-se um elevado índice de falsos positivos (FP). Com isso foi observado o valor preditivo positivo definido (PPV).

$$PPV = TP / (TP + FP) \; x100\%$$

A questão foi que este valor obtido com o PPV foi ainda menor que o anterior, tendo como resultado 64,65%.

3. Revisão Bibliográfica

Este capítulo tem como objetivo descrever as etapas do processo de reconhecimento facial, que podem ser dividas nos seguintes grupos: detecção de faces, extração de características, representação da face, reconhecimento e verificação. Um sistema automatizado é capaz de receber como entrada uma imagem ou vídeo, identificar as faces presentes, caracteriza-las matematicamente, compara-las com outras previamente cadastradas em um banco de dados e, caso haja alguma correspondência, o reconhecimento facial informa qual face do banco de dados condiz com a imagem de entrada. A Figura 1 descreve um sistema de reconhecimento facial.

[1] *JSON* e um acrónimo para ”Java Script Object Notation”, é um formato leve para intercâmbio de dados computacionais.

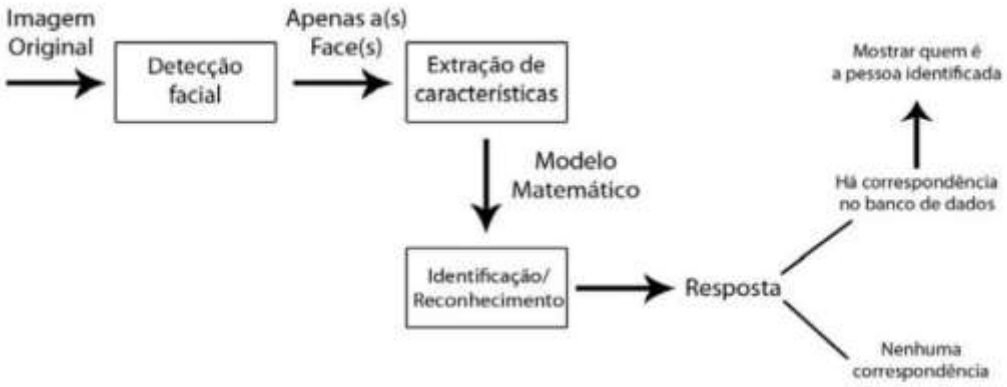

Figura 1. Processo de Reconhecimento

3.1. Detecção facial

A localização de uma face, em uma imagem, é muito importe nesta fase se o algoritmo encontra uma ou mais faces, dentre as imagens localizadas, duas medidas são muito importantes pra a qualidade do algoritmo, são elas: quantidade de faces incorretas identificadas, classificadas como (falso Positivo) e a quantidade de que não foram identificadas, classificadas como (falso negativo). Muito importante caso a entrada do sistema seja uma sequência ao vivo ou em vídeo, pois o algoritmo necessita ser rápido, uma vez que a análise será em tempo de execução.

3.2. Algoritmo de Viola-Jones

Abordagem para detecção de objetos em imagens que se baseia em três conceitos: integral de imagem, treinamento de classificadores usando *boosting* e o uso de classificadores em cascata [Viola et al. 2001]. Embora o algoritmo possa ser treinado para reconhecer qualquer objeto, a motivação principal da abordagem de Viola e Jones foi o reconhecimento facial, o ponto forte deste algoritmo é a rapidez com que é executado.

As *feature haar*, são mascaras retangulares, nas quais os valores dos *pixels* de uma determinada parte da imagem são subtraídas da outra parte, partes estas representadas pela diferença da intensidade de iluminação entre as áreas da imagem. Para cada área do rosto existe uma determinada *feature haar* como, por exemplo, visto na Figura 2 onde vemos um *feature haar* para localizar a área em destaque.

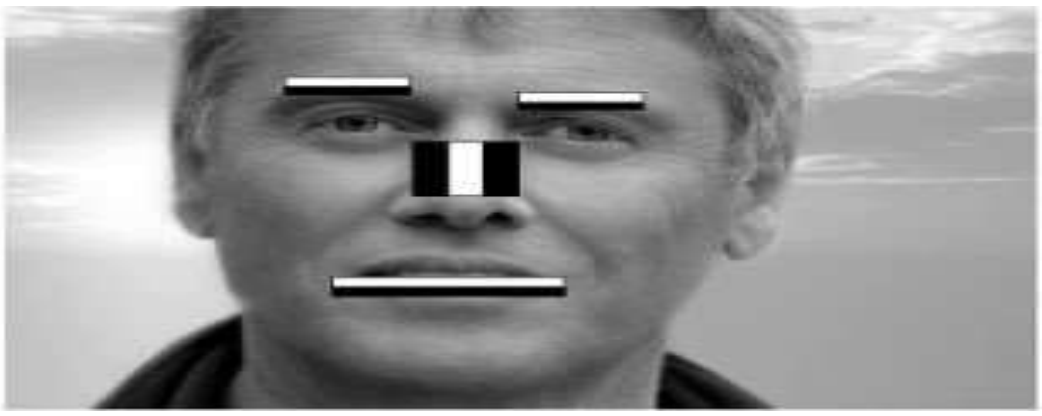

Figura 2. Haar Feature Seleção

3.3. Extração das Características

Nesta fase pode ser realizada a extração das características propriamente ditas, ou analisa as respostas obtidas, e com base nisto ajusta o reconhecedor com apoio de novos requisitos, com isso se tem a certeza que está obtendo informações consideráveis e que as características que serão analisadas são do objeto em questão.

3.4. Reconhecimento

A última fase do reconhecedor é confrontar os dados de entrada, que são as imagens vindas de uma câmera ou vídeo, com as que anteriormente foram cadastradas na base de dados. Isso irá mostra quão consciente reconhecedor está, e se a coleta e extração das características do objeto em desejo foi aplicada corretamente. O importante é escolher uma métrica confiável para estimar a similaridade entre as faces. Empregam-nos métodos, classificação de distância baseada no vizinho mais próximo, que usam distância, em geral *euclidiana*, para estimar a proximidade das imagens no subespaço da face. Métodos mais sofisticados se baseiam em densidades de probabilidade e redes neurais.

3.5. Eigenface

O algoritmo *Eigenface* é um método matemático para a decomposição de imagens em pequenos conjuntos de sub imagens, denominados *eigenfaces*, com o intuito de codificar os traços importantes de um grupo de faces, auxiliando na diferenciação entre elas [Silveira and de Sa 2018] agindo com um filtro.

Partindo das imagens em análise, em uma escala de *pixel* onde 0 é totalmente preto e 255 brancos, pode se esquematizar quaisquer imagens, pois transforma-se a referida em um vetor coluna, com base nos *pixeis*, ou seja, *IMG*(a1, a2, a3... an), considerando que a matriz *M* é composta dos autovetores e autovalores da matriz quadrada, as imagens do conjunto de treinamento são projetadas no espaço de faces, efetuando-se a operação de transformação em relação a vetor IMG (a1, a2, a3... an), ou seja.

$$E1 = U^t(a_n \rightarrow)$$

E para confronto do conhecido com o desconhecido na fase de reconhecimento, aplicam-se os passos iniciais e encontra-se uma face de entrada, desta subtraı-se das faces média da base, gerando a matriz de transformação da face, e comparando a distância em relação às imagens cadastradas, o que atribui à imagem de entrada com a face correspondente do reconhecimento.

4. Ferramentas Computacionais Utilizadas

Para desenvolver o algoritmo serão necessárias algumas ferramentas computacionais como o *Python*, uma linguagem dinâmica, interpretada, robusta, multi-plataforma, multi-paradigma (orientada a objetos, funcional, refletiva e imperativa) e está preparada para rodar em *JVM*[2] e *.NET Framework*[3]. Lançada em 1991, por Guido van Rossum, é uma linguagem livre (até para projetos comerciais) e hoje pode-se programar para *desktops*, *web* e *mobile* [Borges 2014].

[2] 2 A *JVM* e responsável pelo gerenciamento dos aplicativos,´a medida que são executados

[3] O.*NET Framework* e uma iniciativa da empresa *Microsoft*, que visa uma plataforma única para desenvolvimento e execução de sistemas e aplicações. Todo e qualquer código gerado para. NET pode ser´ executado em qualquer dispositivo que possua um *framework* de tal plataforma

OpenCV foi originalmente desenvolvida pela Intel, em 2000, é uma biblioteca multiplataforma, totalmente livre ao uso acadêmico e comercial, para o desenvolvimento de aplicativos na área de Visão Computacional, bastando seguir o modelo de licença da *BSD* Intel. A *OpenCV* possui módulos de Processamento de Imagens e Vídeos I/O, Estrutura de dados, Álgebra Linear, GUI (*Software* Gráfica do Usuário) básica com sistema de janelas independentes, controle de mouse e teclado, além de mais de 350 algoritmos de Visão Computacional como: filtros de imagem, calibração de câmera, reconhecimento de objetos, analise estrutural e outros. [Mordvintsev and Abid 2014].

Redes Neurais Artificiais – RNA, as redes neurais artificiais são modelos matemáticos inspirados no modelo do neurônio biológico com o objetivo de emular o processo cognitivo do cérebro humano.

Machine Learning (aprendizado de Máquina) é uma tecnologia poderosa no auxílio da Inteligência Artificial (IA), mas não existe um único algoritmo que apresente o melhor desempenho para todos os problemas [Dietterich 1997]. Dessa forma, e importante compreender o poder e suas limitações de diversos algoritmos existentes que permitam compara-los.

Reconhecimento de Padrão requer algum conhecimento prévio e algum tipo de armazenamento do conhecimento sobre o objeto, esta é a parte onde os sistemas de visão possuem uma intersecção com a inteligência artificial. Para fazer o reconhecimento, um sistema de visão necessita uma base de conhecimento dos objetos a ser reconhecidos, esta base de conhecimento, ou seja, são as amostras dos objetos a serem reconhecidos, os quais utilizam técnicas de aprendizado de máquina para auxiliar na tomada de decisões estratégicas em tempo execução [Queiroz and Pinto 2014]. O reconhecimento de objetos é uma das principais funções da área de visão computacional, um objeto pode ser definido por mais de um padrão (textura, forma, cor, dimensões, etc.) e o reconhecimento individual de cada um destes padrões podem facilitar o reconhecimento.

Graphical Processing Units(GPU) um processador de função fixa, construído sobre um *pipeline* gráfico com função apenas no processamento gráfico em três dimensões. Atualmente, estes processadores evoluíram e têm funções específicas para uma alta capacidade de cálculos massivos paralelos e com uma total programabilidade, tendo como resultado um *hardware* de imensa habilidade aritmética e não mais limitado às operações gráficas.

4.1. Visão Computacional

Conforme [Bradski et al. 2005] visão computacional é a transformação de dados de imagens estáticas ou em vídeos. Um exemplo de decisão seria verificar se há um carro em certa imagem ou identificar quantos carros existem na imagem, estas técnicas são puramente matemáticas, como geometria e análise, assim [Sanderson and Fisher 1994] discute a relação da visão computacional e a inteligência artificial, mostrando que ambas partilham de mesmos propósitos. Outra definição bem aceita pelos pesquisadores da área é a de [Marr and Poggio 1979], que afirma: “ visão é o processo que produz, a partir de imagens do mundo externo, uma descrição que é útil ao usuário e que não é repleta de informações irrelevantes” [Marr and Poggio 1979]. Analisando do ponto de vista da engenharia, esse é o processo de automatizar as tarefas que o sistema visual humano desempenha, assim como um médico pode identificar um tumor em uma

tomografia ou uma pessoa poderia reconhecer o próprio rosto em uma fotografia tais sistemas poderiam fazer o mesmo [Szeliski 2010]. Dentre algumas aplicações podem-se destacar as propostas na seção trabalhos correlatos deste artigo.

5. Metodologia

No que tange a metodologia de desenvolvimento do sistema proposto, ela está baseada na eXtreme Programmin (XP), pelo fato de atender as necessidades do trabalho e possuir características que permitem adaptar o projeto conforme a execução do mesmo careça.

5.1. eXtreme Programmin (XP)

XP como é conhecido, segundo [Beck et al. 2001], manifesto é [Nunes 2017] a implantação de uma metodologia leve, para times de tamanhos pequeno a médio, a equipe XP é composta de dois a dez desenvolvedores, chegando ao máximo em 12, se este for maior pede-se que seja dividida em duas equipes, pois a comunicação começa a ser reduzida. XP desenvolve s*oftwares* na presença de requisitos vagos que se modificam rapidamente, sendo uma metodologia de *software* ágil, utilizado na busca pela qualidade para que se atenda às necessidades do cliente e têm cinco valores fundamentos, os quais são: [Beck et al. 2001].

- Feedback - Nesta etapa a comunicação com os agentes da segurança pública passa positividade;
- Comunicação - Nesta etapa a comunicação direta, pontuando tudo que aconteceu no decorrer do projeto e na codificação, uma boa comunicação foi o ponto crucial;
- Simplicidade - Nesta fase buscamos fazer o projeto da forma mais simples possível, o que funcionou, pois ganhamos agilidade e entrega frequente;
- Coragem - Nesta fase buscamos deixar livre para mudanças, sejam elas quaisquer.

6. Visão Geral do Software

As informações de procurados pela Polícia Civil (PC) são sigilosas, logo a proposta do trabalho foi validada com imagens de livre acesso, obtidas pela *internet* ou mesmo geradas pelo próprio autor, consistindo em uma base de dados de treinamento para o algoritmo de inteligência artificial, na área de visão computacional. Representação dos passos e componentes da solução proposta na Figura 3.

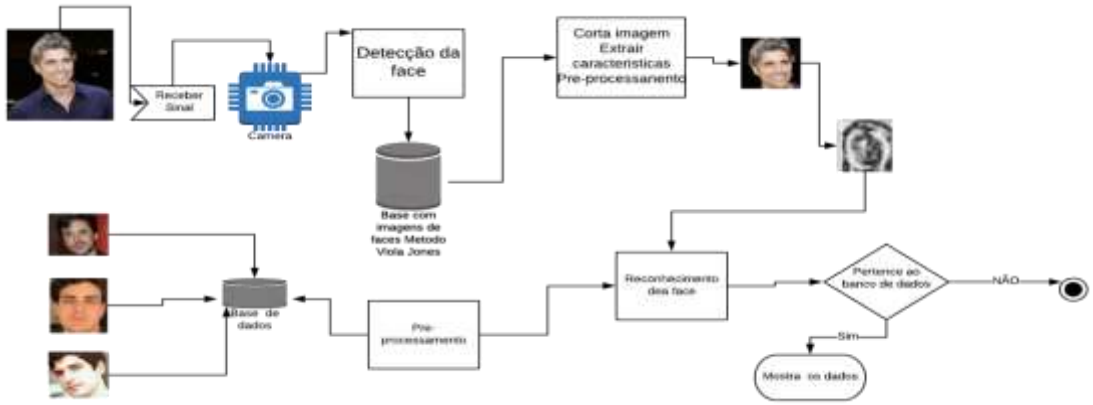

Figura 3. Diagrama e uma representação visual estruturada e simplificada

7. Implementação

Análise dos resultados experimentais mostrou que o algoritmo citado no artigo Viola-Jones [Viola et al. 2001] e utilizado por [Penteado 2009], o algoritmo usado para a detecção de faces em vídeo é o Viola-Jones. Para o treinamento foi coletada uma base de dados com 24 personagens e foram usados diversos vídeos e imagens retiradas da internet, vídeos de livre acesso, como as plataformas *Google*, que tem cerca de 3,3 bilhões de buscas, as quais são realizadas diariamente [Alves 2018].

Também utilizamos o maior portal de vídeos da internet, o *You-Tube.* [Puhl and Araujo 2012] refere-se como ferramenta de construção da memória coletiva no suporte digital, mídias estas que foram convertidas para a escala de cinza para que algoritmo *HaarCascade* possa localizar, recortar as faces presentes, redimensionar em 110 x 110, e gravar no *dataSet* as faces com extensão da imagem. *jpg*. A Figura 4 demostra como foi planejado o *dataSet*, diretório de armazenamento das imagens das faces cadastradas e o arquivo onde localiza-se os nomes dos cadastrados. Coletamos 50 unidades de imagens, a menor coleta, e, a maior com 500 unidades de imagens, onde houve variação da luz, fundo, esquerda da face, a direita da face, uso de óculos, sem óculos, expressões de feliz, triste, sonolento, com piscada, isso simula uma condição onde não temos controle do ambiente, onde iremos analisar dados, visto que este será o objeto de treino do *software.*

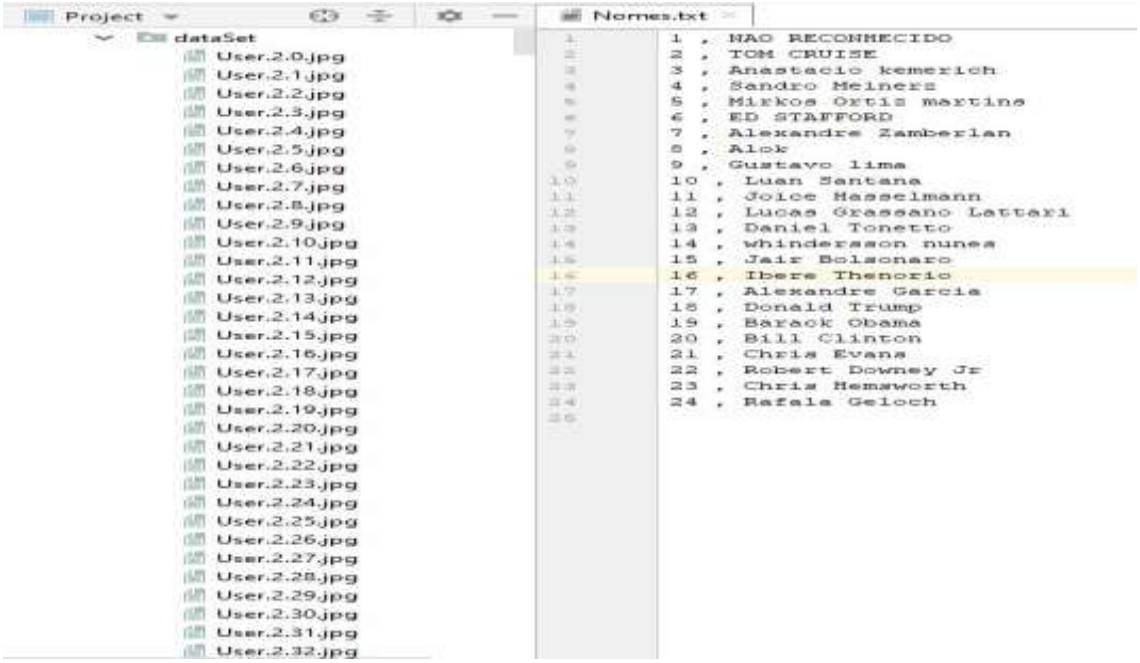

Figura 4. Base de Dados com o dataSet e Arquivo de Nomes

- Inicializar uma imagem;
- Tratar a Imagem;
- Detecção Facial na Imagem;
- Isolamento e Recorte da Face.

7.1 Inicialização da Imagem

Nesta seção mostra como inicializa uma imagem na linguagem *Python.* Iniciamos uma variável que irá receber o chamado do método de captura da câmera, ou imagem dependendo do parâmetro, O (Zero) uma imagem fixa e 1 (Um) para câmera de vídeo.

$$Cap = cv2.vıdeoCapture(0)$$

7.2 Tratar a Imagem

Nesta seção trataremos da conversão da imagem anteriormente capturada em formato colorido *RGB* para *BGR*, para facilitar o entendimento podemos pensar em uma planilha eletrônica, com linhas e colunas, portanto, uma matriz de 2 dimensões. Cada célula dessa matriz é um *pixel*, que no caso de imagens preto e brancas possuem um valor de 0 a 255, sendo 0 para preto e 255 para branco. Logo, cada célula contém um inteiro de 8 bits (sem sinal) que em Python e definido por "uint8" que é um *unsigned integer* de 8 bits. Figura 5

```
21      gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
22      if np.average(gray) > 110:
```

Figura 5. Trecho do código da conversão

E se a média do brilho da conversão for maior que 110 passa para a etapa de localizar o face e a posição, neste caso um critério para dar o *start* para a coleta das imagens no padrão desejado.

7.3. Detecção Facial na Imagem

O *Haarcascade* foi utilizado como algoritmo de detecção, e assim tornando o reconhecimento mais instantâneo, dando agilidade na busca pelo rosto. Os seguintes códigos da Figura 6, nas linhas de código 12 e 13 as duas variáveis que serão responsáveis por reconhecer a parte frontal da face e os olhos, respectivamente.

Na linha 24 do código a variável *faces* recebera a posição do rosto encontrado após a imagem ser totalmente analisada.

```
12  face_cascade = cv2.CascadeClassifier('Haar/haarcascade_frontalcatface.xml')
13  eye_cascade = cv2.CascadeClassifier('Haar/haarcascade_eye.xml')

24          faces = face_cascade.detectMultiScale(gray, 1.4, 5)
```

Figura 6. Utilização do *Haarcascade*

7.4. Captura da face

Com a posição do rosto na imagem daremos início à aquisição das imagens anteriormente trabalhadas, e as gravando em nossa Base de Dados própria, imagens estas que ficaram armazenadas de forma crescente, onde o nome *User* e o identificador

de todas as imagens acompanhado da identificação (ID) da imagem é um numeral crescente e sua extensão (.jpg), no código na Figura 7 e na Figura 8 observamos o armazenamento desta imagem.

```
        for (x, y, w, h) in faces:
            FaceImage = gray[y - int(h / 2): y + int(h * 1.5), x - int(x / 2):
                                                               x + int(w * 1.5)]
                                                        # Face é isolada e recortada
            #print(FaceImage)
            Img = (NomeFi.Detectaolho(FaceImage))
            cv2.putText(gray, "FACE DETECTADA", (x+int((w/2)), y-5),
                        cv2.FONT_HERSHEY_DUPLEX, .4, WHITE)
            if Img is not None:
                frame = Img
            else:
                frame = gray[y: y+h, x: x+w]
            cv2.imwrite("dataSet/User." + str(ID) + "." + str(Count) + ".jpg", frame)
            cv2.waitKey(50)
            cv2.imshow("FOTOGRAFIAS CAPTURADAS", frame)
            Count = Count + 1
```

Figura 7. Trecho de Código do Armazenamento das Imagens

Figura 8. Descrição de Armazenamento

Para o armazenamento do nome das pessoas cadastradas em nossa base de dados denominada *dataSet*, um arquivo de texto padrão com extensão *.txt* foi criado e nomeado de *Nomes.txt*, neste, quando vamos analisar uma imagem e dela retirar faces, e gravamos o nome que será dado à pessoa, no arquivo *Nome.txt a* primeira informação gravada é o que dará o início do reconhecedor, pois o *software* por si só não reconhece, sendo assim com ID 1 gravamos o texto NAO RECONHECE e os posteriores já como nome referenciando a imagem, como verá na Figura 9 e a cada nova imagem uma linha é criada Figura 10.

```
1 , NAO RECONHECIDO
2 , TOM CRUISE
3 , Anastacio kemerich
4 , Sandro Meinerz
```

Figura 9. Arquivo de Nomes

```
#     ------------------- ESTA FUNÇÃO LEIA O ARQUIVO E ADICIONA O NOME AO FIM DO ARQUIVO
def AddNome():
    Nome = input('Entre com o Nome:  ')
    Info = open("Nomes.txt", "r+")
    ID = ((sum(1 for line in Info)) + 1)
    Info.write(str(ID) + " " + "," + " " + Nome + "\n")
    print("Nome armazenado em:  " + str(ID))
    Info.close()
    return ID
```

Figura 10. Função Ler o Arquivo e Adiciona o Nome ao Fim do Arquivo

7.5. Treinamento

O algoritmo de aprendizado recebe um conjunto de exemplos para o treino, os quais os rótulos das classes associadas são conhecidos, os exemplos são descritos por um vetor de Valores (atributos) e pelo rótulo da classe associada.

Com o treinamento obtivemos o objetivo de construir um reconhecedor que possa determinar novas imagens ainda não rotuladas. É um método de aprendizagem supervisionado que constrói árvores de classificação a partir de exemplos [Quinlan 1986], os métodos baseados em árvores para classificação dividem o espaço de entrada em regiões disjuntas para construir uma fronteira de decisão. As regiões são escolhidas baseadas em uma otimização heurística onde a cada passo os algoritmos selecionam a variável que prove a melhor separação de classes, de acordo alguma função custo.

No *software* proposto tomamos uma abordagem de treino da seguinte forma, acessa o *dataSet*, contendo as imagens coletadas, divididas heuristicamente para um treino supervisionado, criamos um pacote de imagens, após abrimos as imagens e convertemos para *grayScale*. Como a base de dados contém imagens com tamanhos aleatórios há a necessidade de um padrão, para isso o tamanho 110x110 foi adotado.

```
path = 'dataSet'                                    #caminho para as fotos

def pegaIDimagem (path):
    imagepacote = [os.path.join(path, f) for f in os.listdir(path)]
    Facelista = []
    IDs = []
    for imagePath in imagepacote:
        faceImage = Image.open(imagePath).convert('L')  # Abrir imagem e converter para cinza
        faceImage = faceImage.resize((110,110))         # redimensionar a imagem para que o reconhecedor EIGEN possa ser treinado
        faceNP = np.array(faceImage, 'uint8')           # converter a imagem em matriz Numpy
        ID = int(os.path.split(imagePath)[-1].split('.')[1]) # Recriar novamente o ID da matriz
        Facelista.append(faceNP)                            # Append matriz Numpy a lista
        IDs.append(ID)                                      # Append o ID para a lista de IDs
        cv2.imshow('Conjunto de Treinamento', faceNP)       # Mostrar as imagens na lista
        cv2.waitKey(1)
    return np.array(IDs), Facelista                     # Os IDs são convertidos em uma matriz Numpy
IDs, Facelista = pegaIDimagem(path)
```

Figura 11. Fragmentado do Código

Após este processo criamos uma matriz com as características da imagem, Figura 11. Este processo é essencial para o processo de aprendizado, pois estes dados, após o fim do treinamento, origina um arquivo.*XML* que será a memória de nosso sistema, pois uma vez gravadas estas informações já somos capazes de reconhecer. Em sistemas de verificação [Mansfield and Roethenbaugh 1998] as características de um indivíduo desconhecido são comparadas com as características de um indivíduo em particular, as características biométricas do indivíduo desconhecido são comparadas com as características dos indivíduos conhecidos pertencentes a uma base de dados e nos resulta em uma confiança entre as imagens da *dataSet* e a imagem desconhecida.

8. Resultados

Nesta seção mostraremos os resultados obtidos durante os testes que validaram esta proposta, [Beck et al. 2001] o TDD (*Test Driven Development*), como a metodologia sugere, é uma técnica para construção de *software* que guia seu desenvolvimento por meio da escrita de testes, o objetivo é unicamente a aprovação nos testes. Neste

momento não é necessário se preocupar com as boas práticas, design patterns ou coisas do tipo, o código só deve funcionar e passar pelo teste sem quebrar outros testes.

Esta abordagem tende a ser mais coesa, pois o teste do código é parte íntima da codificação, e não um processo independente. Além disso, reduz o acoplamento dos componentes do *software.* Com isso, torna-se possível fazer decisões de projeto em cada estágio do desenvolvimento. Com o uso desta técnica é possível reduzir a complexidade do *software*, aumentando a manutenibilidade do mesmo. Falhas são facilmente identificadas ainda na etapa de desenvolvimento, graças ao contínuo feedback dado ao programador. Para a base, dados com 3000 imagens divididas em 24 identificadores (ID), com diferentes quantidades de imagens de cada ID obtiveram os seguintes resultados.

8.1. Teste de vídeo capturado ao vivo

Os testes de validação foram realizados utilizando OpenCV para Python e uma captura de imagem obtida durante um dos testes de detecção pode ser visualizada na Figura 12.

Figura 12. Reconhecimento pela webcam

Um rosto foi identificado contendo o padrão treinado, o qual foi utilizado nos testes, uma câmera de resolução congurável de (320 x 240) ou (640 x 480), nas quais o comportamento do detector foi avaliado. A câmera utilizada para capturar a imagem da Figura 12 foi um webcam. Os efeitos de distorções de lentes não foram considerados no processo.

Neste teste temos um índice de acerto de 94,5% de acurácia, para uma pessoa apenas em frente da webcam, dispensando fundo, iluminação entre outros, tal resultado vemos na Figura 13 do próprio autor.

Figura 13. Reconhecimento Pela Webcam

8.2. Teste com vídeo gravado

Para análise de um vídeo e verificar se as faces presentes seriam identificadas corretamente pelo *software* para um vídeo. Para este teste foi analisado um vídeo com apenas duas pessoas.

Para o reconhecimento foi calculado um índice confiança, este que está estipulado pela Figura15, abaixo, onde recebemos uma imagem na Figura14 e a partir dela identifica-se existem dois olhos, calcula-se a diferença entre a altura dos olhos e a largura e com isso calculamos a coordenada que vai formar um angulo entre os dois olhos, convertendo as posições dos olhos de radianos para graus se obtém o angulo de identificação da imagem.

```
171  def DetectEyes(Imagen):
172      Coordenadas = 0
```

Figura 14. Função DetectEyes

```
189          if (DX != 0.0) and (DY != 0.0):
190              # Certifique-se de que a alteração ocorra apenas se houver um ângulo
191              Coordenadas = math.degrees(math.atan(round(float(DY) / float(DX), 2)))  # Encontre o ângulo
192                  #math.degrees( X ) Converta o ângulo x de radianos em graus.
193                  #math.atanh( X ) Retorne a tangente hiperbólica inversa de x .
194              print("Dados recolhidos são:   " + str(Coordenadas))
195
```

Figura 15. Fragmento de código geração das coordenadas

- Para Imagens Estáticas

$$\frac{1087}{484} * \frac{100}{x}$$

$$-48,400 * 1087$$

$$x = \frac{48,400}{1087}$$

= 44,52%

- Para Imagens vídeo e Ao Vivo

$$\frac{1022}{945,80} * \frac{100}{x}$$

= 94,581 * 1022

$$x = \frac{945,80}{1022}$$

= 92,55%

Observou-se que o resultado da detecção é significantemente mais assertivo para vídeos e ao vivo via câmera, pois na Figura 16, analisamos as entradas na coluna 1, que mostra se é uma imagem ou um vídeo, logo, na coluna 2 se houve um reconhecimento ou não, coluna 3 quantas faces havia na imagem ou vídeo da coluna 1, na coluna 4 quantas faces o reconhecedor localizou e na coluna 8 se destas faces localizadas na coluna 4 quantas foram reconhecidas como acerto.

Coluna1	Coluna2	Coluna3	Coluna4	Coluna5	Coluna6	Coluna7	Coluna8	Coluna9
Imagem	Localizou	Qtd Faces	Qtd identificadas	Video	Foto	Reconhecimento	Qtd Reconhecimento - ok	%
IMG1	sim	7	3		sim	Não	0	-
IMG2	sim	1	1		sim	sim	1	100,00
IMG3	sim	30	4		sim	Não	0	-
IMG4	sim	30	4		sim	Não	0	-
IMG5	sim	1	1		sim	Não	0	-
IMG6	sim	4	3		sim	Não	0	-
IMG7	sim	60	31		sim	sim	5	16,13
IMG8	sim	5	5		sim	sim	2	40,00
IMG9	sim	20	2		sim	Não	0	-
IMG10	sim	17	1		sim	Não	0	-
IMG11	sim	4	4		sim	sim	1	25,00
IMG12	sim	2	2		sim	sim	1	50,00
IMG13	sim	1	1		sim	sim	1	100,00
IMG14	sim	2	1		sim	sim	1	100,00
IMG15	sim	1	1		sim	sim	0	-
IMG16	sim	1	1		sim	sim	1	100,00
Video1	sim	1	408	sim		sim	205	50,25
Video2	sim	1	93	sim		sim	86	92,47
WebCam	sim	1	212	sim		sim	161	75,94
WebCam	sim	2	309	sim		sim	19	6,15
Total		191	1087				484	

IMG	Coluna1	Coluna2	Coluna1	Coluna2	Coluna3	Coluna4	Coluna5	Coluna6
	Indice de Acurácia	44,53	5	1022	471			945,8104799
Video	Coluna1	Coluna2						
	Indice de Acurácia	92,55						

Figura 16. Tabela de Testes

9. Conclusões

Este projeto apresentou a proposta de uma interface para auxiliar o reconhecimento de faces humanas, abordamos assuntos como processamento de imagens digitais, redes neurais e reconhecimento de padrões. Dentro do contexto abordado proposto, é possível destacar algumas vantagens, tendo como principal objetivo disponibilizar para a comunidade junto à segurança pública no auxílio e na agilidade do reconhecimento e buscas de indivíduos, inseridos no meio da população.

Utilizando das tecnologias do texto, julgamos como válido nossos resultados, pois devido ao pouco tempo de estudo e conhecimento das tecnologias utilizadas e com o cenário vivido em nosso país, devido à pandemia do COVID19, mesmo obtendo resultados inesperados, percebe-se que o estudo realizado tem grande possibilidade de sucesso, com mais alguns refinamentos no levantamento das características da face antes do treinamento e um processamento mais rápido no treinamento da base de dados, pode ser usado processamento na *GPU*.

As *GPU* se destacam em cargas de trabalho paralelas e aceleram as redes neurais em até 10 a 20 vezes. [Uetz and Behnke 2009] em seu trabalho, relata um reconhecedor de objetos de larga escala, com processamento paralelo na *GPU* conseguindo alcançar uma velocidade de 82 vezes em relação ao *CPU* com um núcleo, isso mostra que a atualização paralela reduz o tempo de uma determinada interação de treinamento de dados, parte esta mais custosa, devido aos elevados índices de processamento.

Ao final da construção desse projeto, foi possível programar um sistema de reconhecimento de faces utilizando *Python* e a biblioteca *OpenCV* como base de um algoritmo de visão computacional. Para o reconhecimento propriamente dito, foi utilizada a arquitetura *Haarcascade*, a qual demonstrou um índice de 95,5% de acurária na determinação das faces, treinadas no modelo.

Dos objetivos listados nesse projeto, podemos concluir que foi possível estudar e protótipo uma tecnologia que determinou os padrões de formatos geométricos que formam uma face. Com base nessa tecnologia, foi possível gerar uma base de padrões para treinar o modelo de reconhecimento, na qual foi testada a capacidade de identificação do reconhecimento facial da ferramenta desenvolvida. Ainda nesse

projeto, foi desenvolvido um *software* capaz de extrair imagens estáticas de vídeos (frames), para posterior uso no reconhecimento facial. Foram comparados vídeos capturados ao vivo em relação àqueles previamente gravados e avaliada a acurácia de *software*, em relação ao reconhecimento.

Infelizmente, por motivo de distanciamento social, devido ao COVID-19, não foi possível construir uma funcionalidade de alerta aos agentes de segurança pública, pois as reuniões planejadas para definição de escopo, *features*, testes de avaliação e uso com o cercamento eletrônico da cidade de Santa Maria não ocorreram.

A funcionalidade em si foi simulada, porém o seu emprego real foi prejudicado pela pandemia que assola a humanidade nesse primeiro semestre de 2020.

Baseado nos resultados obtidos, conclui-se que o uso de linguagem de programação Python e as extrações das características com auxilio da biblioteca OpenCV em coparticipação do algoritmo Haarcascade na busca de foragidos em um sistema de câmeras é totalmente adequado no cenário tecnológico encontrado.

Referências

Alves, P. (2018). Google em números: veja curiosidades sobre os 20 anos da empresa. *Obtendo de https://www.techtudo.com.br/noticias/2018/09/google-em-numeros-vejacuriosidades-sobre-os-20-anos-da-empresa.ghtml*.

Beck, K., Beedle, M., Bennekum, A. v., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., et al. (2001). Manifesto para desenvolvimento agil de software. *Retirado em*, 20. Acessado em 24 sets 2019.

Borges, L. E. (2014). *Python para desenvolvedores: aborda Python 3.3*. Novatec Editora.

Bradski, G., Kaehler, A., and Pisarevsky, V. (2005). Learning-based computer vision with intel's open source computer vision library. *Intel technology journal*, 9(2).

Carneti, K. (2015). Pol´ıcia chinesa usa oculos com reconhecimento facial para identificar suspeitos disponível=https://exame.abril.com.br/tecnologia/china desenvolve caixa eletrônico com reconhecimento facial. Acesso 30 ago.2019.

da SILVA, I. D. and da SILVA, J. A. Reconhecendo as práticas de reconhecimento facial: comunicação das coisas, quarta revolução industrial e novas tecnologias digitais1. Dietterich, T. G. (1997). Machine-learning research. *AI magazine*, 18(4):97–97.

Globo, G. (2010). *Para evitar fraudes, ônibus de Santa Maria têm reconhecimento facial*. G1 Globo.

Koh, L. H., Ranganath, S., Lee, M. W., and Venkatesth, Y. (1999). An integrated face detection and recognition system. In *Proceedings 10th International Conference on Image Analysis and Processing*, pages 532–537. IEEE.

Lung, L. F., Barua, M. N., and Juarez, P. S. (2019). An image acquisition method for face recognition and implementation of an automatic attendance system for events. acessado em 20 Out 2019.

Mansfield, T. and Roethenbaugh, G. (1998). Glossary of biometric terms. *J. Association for Biometrics & International Computer Security Association*.

Marr, D. and Poggio, T. (1979). A computational theory of human stereo vision. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 204(1156):301– 328.

Mordvintsev, A. and Abid, K. (2014). Opencv-python tutorials documentation. *Obtenido de https://media. readthedocs. org/pdf/opencv-python-tutroals/latest/opencv-pythontutroals. pdf*.

Nunes, R. D. (2017). A implantação das metodologias ágeis de desenvolvimento de software scrum e extreme programming (xp): uma alternativa para pequenas empresas do setor de tecnologia da informação. *ForScience*, 4(2).

Penteado, B. E. (2009). Autenticação biométrica de usuários em sistemas de e-learning baseada em reconhecimento de faces a partir de vídeo.

Puhl, P. R. and Araujo, W. F. (2012). Youtube como espaço de construçã o da memó ria em rede: possibilidades e desafios. *Revista FAMECOS: mídia, cultura e tecnologia*, 19(3):705–722.

Queiroz, S. S. F. and Pinto, K. L. N. (2014). Extração de características e reconhecimento de padrões e objetos. *VETOR-Revista de Ciências Exatas e Engenharias*, 24(2):2–13.

Queiroz de Santana, L. M., Rocha, F., and Santos, T. (2014). Uma análise do processo reconhecimento facial. *Cadernos de Graduação: ciências exatas e tecnológicas*, 2:49 58.

Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, 1(1):81–106.

Sanderson, P. M. and Fisher, C. (1994). Introduction to this special issue on exploratory sequential data analysis. *Human–Computer Interaction*, 9(3-4):247–250.

Silveira, J. S. and de Sa, A. A. R. (2018). Reconhecimento facial utilizando o algoritmo eigenface da biblioteca open cv. *Revista Eletrônica Argentina-Brasil de Tecnologias da Informação e da Comunicação*, 1(9).

Szeliski, R. (2010). *Computer vision: algorithms and applications*. Springer Science & Business Media.

Terra, G. J. S. (1993). Com técnica de reconhecimento facial, acusado de matar publicitaria em natal e preso 21 anos após o crime. *Anuário do Instituto de Geociências*, 16:108–109.

Uetz, R. and Behnke, S. (2009). Large-scale object recognition with cuda-accelerated hierarchical neurais networks. In *2009 IEEE international conference on intelligent computing and intelligent systems*, volume 1, pages 536–541. IEEE.

Viola, P., Jones, M., et al. (2001). Rapid object detection using a boosted cascade of simple features. *CVPR (1)*, 1(511-518):3.