1. Introdução
   1. Motivação

Há muito o Munícipio do Rio de Janeiro está vivendo um grave problema de segurança pública. A Polícia Civil do Rio de Janeiro encontra-se sobrecarregada, afogada em papeladas e ferramentas, muitas vezes ultrapassadas e que diminuem sua produtividade. O aplicativo Desaparecidos (Vinicius, 2017) Delegacia de Descoberta de Paradeiros (DDPA) de descoberta de pessoas

O aplicativo fornece aos parceiros da DDPA uma maneira de consultar, apenas com o preenchimento de um formulário, o banco de dados de desaparecidos da própria polícia e, portanto, multiplicando as frentes capazes de reconhecer rapidamente uma pessoa sem identificação que tenha sido encontrada.

Apesar da enorme ajuda que o aplicativo introduz, devido ao grande número de pessoas desaparecidas cadastradas com características similares, é possível que o retorno de resultados possa ser muito maior do que o esperado e se torne difícil buscar com precisão. Diante dos fatos, noperíodo,

Contudo, antes da sua implementação, uma pergunta foi feita de modo que levantou dúvidas: o reconhecimento facial poderia realmente ser um método de busca eficaz para pacientes com uso de aparelhos respiratórios ou com alguma deformação? Não havia uma resposta correta.

Sendo assim, este trabalho se propõe a estudar a maneira mais eficiente de implantar a busca por reconhecimento facial no aplicativo Desaparecidos-RJ, com a finalidade aumentar a acurácia das buscas.

* 1. Objetivos

Este trabalho tem por objetivo analisar duas bibliotecas de reconhecimento facial, Face Recognition [1] e OpenFace [2], afim de verificar a precisão de seus resultados em caso de rostos parcialmente cobertos.

A avaliação será feita através do uso de uma base de dados especializada para estudos em oclusão parcial e os resultados de comparação levarão em consideração a rapidez e a acuidade de cada biblioteca.

Organização do texto

O texto está organizado em:

Introdução

Trabalhos relacionados

Reconhecimento facial

Tecnologias

Avaliação

Conclusão

1. Referencial Bibliográfico

Para conseguir reconhecer um rosto, a imagem deve passar por diversos processos até chegar ao resultado final. Este capítulo apresentará, em linhas gerais, parte da teoria por detrás do reconhecimento facial, também conhecida como Visão computacional, e do reconhecimento facial e seus diferentes aspectos.

* 1. Visão computacional

A visão computacional é a área da computação que busca simular o poder da visão humana em um computador. Segundo Marengoni et al [x], a visão computacional é quando o computador consegue interpretar como um todo, ou parcialmente, uma imagem de entrada dada pelo usuário.

O processamento de imagens é uma das partes dessa interpretação feita pelo computador e consiste basicamente em transformar a imagem em um conjunto de números para que a máquina consiga ser capaz de interpretar a foto de modo a chegar a uma conclusão o mais próximo possível ao que um ser humano seria capaz de chegar. Basicamente, processar uma imagem consiste em um processo de transformações sucessivas com o objetivo de extrair de forma eficiente a informação nela presente (ALBUQUERQUE; ALBUQUERQUE, 2002)

O processamento de imagens pode ser divido em três níveis: baixo, médio e alto.

No nível baixo, encontram-se tarefas voltadas a compactação, isto é, remoção de redundância de informações para reduzir a quantidade de dados que serão processados, e pequenas correções, como: escala de cores, retirada de bordas e filtros, por exemplo. No fim, este nível engloba grande parte das tarefas realizadas relativo ao pré-processamento de imagens. O pré-processamento de imagens vai desde correção de cores e seu redimensionamento à segmentação de suas partes importantes e remoção de ruídos, que nada mais são do que a remoção de condições que atrapalham a interpretação da imagem (marcas d’água, filtros, iluminação, etc).

O nível médio é responsável pelo reconhecimento de pequenos padrões. Essas características encontradas ajudam o mapeamento de formas geométricas que serão usadas como facilitadores para as tarefas do nível alto.

O nível alto, por outro lado, é onde acontece a classificação de um objeto, ou seja, onde é feito seu reconhecimento. É o nível mais próximo ao sistema de visão humano.

O Reconhecimento Facial, assunto abordado neste trabalho, passa por todos os níveis, tendo cada um função importante para o resultado final. No nível mais baixo, a imagem sofrerá o pré-processamento para reduzir a quantidade de redundâncias. Nela haverá o corte inicial do rosto e o descarte do restante da imagem. Em seguida, no nível médio, serão encontradas as características faciais, fazendo mais um recorte na imagem, selecionando apenas o que será analisado pelo algoritmo. Por fim, serão feitos o reconhecimento e a classificação do rosto.

Na seção 2.2, será abordado mais sobre o que é o reconhecimento facial e as principais técnicas utilizadas.

* 1. **Reconhecimento facial**
     1. **O que é?**

Com expansão do uso de tecnologias, as técnicas de biometria evoluem para garantir a segurança do enorme volume de informações privadas que são geradas todos os dias. A biometria é um estudo que se baseia em aspectos biológicos e únicos do ser humano para identificar um indivíduo. Graças ao seu poder de distinção de pessoas, atualmente ela vem ganhando espaço em diversas áreas, principalmente àquelas voltadas para segurança.

O reconhecimento facial é uma das técnicas biométricas que estão presentes em vários aspectos da vida moderna. Pode ser utilizada para algo simples como o desbloqueio da tela de um celular, assim como em questões mais críticas, como na República Popular da China, onde o reconhecimento facial está sendo utilizado para calcular pontos de cidadania que garantem direitos e privilégios aos chineses.

O governo utiliza a inteligência artificial para prestar serviços públicos e monitorar a cidade, e as pessoas que nela vivem. Já é possível que o cidadão chinês pague suas contas, saque dinheiro e acesse demais serviços públicos apenas com seu rosto (LOPES, João Fabiano Martucci; 2018. p. 92)

Tratando-se de técnicas, existem dois modos de abordar o tema reconhecimento facial: o reconhecimento facial com profundidade ou sem profundidade.

é Em modelos em duas dimensões, muitas análises são feitas primariamente contando com o posicionamento de certos elementos no rosto, como: olhos, nariz, boca, sobrancelha e etc.. Muitas técnicas de reconhecimento de rostos para imagens bidimensionais, ao tratar de imagens frontais, procura por esses elementos em posições previamente mapeadas. Deste modo, ele é capaz de localizar e afirmar que naquela imagem, existe um rosto que deve ser reconhecido. Porém, ao rotacionar um rosto em uma imagem bidimensional, como em uma foto em que apareça uma pessoa de perfil, o posicionamento destes elementos não será o mesmo uma vez que você terá a visão de apenas uma parte do rosto.

O reconhecimento facial com profundidade trata apenas rostos com três dimensões, ou seja, o dado de entrada é uma imagem tridimensional. Estes modelos 3D podem ser obtidos de alguns modos diferentes: através de um software com capacidade de transformar imagens bidimensionais em tridimensionais, ou através de escaneamento do rosto que está sendo modelado.

Utilizar esse tipo de dados de entrada possui vantagens em relação ao utilizado na análise bidimensional, podendo tratar com uma melhor precisão problemas de iluminação, posicionamento do rosto, reconhecimento de características, etc. Apesar de ainda sofrer com ruídos (movimentação, problemas no escaneamento, etc), os reconhecimentos feitos por imagens 3D possuem uma taxa de acerto maior quando comparados com as análises 2D.

Diferente de modelos em três dimensões por escaneamento, onde costumam ser armazenados sem a reprodução do ambiente em que a pessoa está inserida, as imagens 2D, podendo esta ser apenas uma etapa anterior a criação de um modelo 3D, possuem a preocupação de detectar um rosto em uma imagem, uma vez que a fotografia pode ter sido tirada em uma rua, em um quarto ou em um concerto musical, por exemplo. O primeiro passo, logo, deve ser garantir que o computador seja capaz de distinguir o que é um rosto na imagem que está sendo analisada, isso reduzirá o processamento a um segmento específico da foto.

A detecção de faces, porém, pode ser feita de diversos modos diferentes. Algumas delas são:

1. Método baseado em conhecimento: utiliza-se de regras pré-estabelecidas para identificar elementos que são comuns a todas as faces. Por exemplo, sabemos que todas as pessoas possuem um nariz, localizado no meio do rosto. O mesmo não pode ser dito de sobrancelhas, uma vez que é um elemento que pode estar ou não presentes em um rosto.
2. Método baseado em *template*: este método avalia a existência de um rosto comparando-o com um modelo pré-existente. Este modelo pode ser apenas uma forma geométrica, por exemplo.
3. Métodos baseados na aparência: diferente dos outros métodos apresentados, o método baseado em aparências não possui regras ou modelos pré-estabelecidos, toda e qualquer informação avaliada surge da própria imagem que está sendo analisada. São utilizadas metodologias de aprendizado e treinamento e existem diversos algoritmos e maneiras de abordar esse método.

Contudo, a detecção de um rosto pode apresentar erros ou problemas por causa de ruídos. Nesta fase, ruídos são conhecidos como oclusão e hoje representam um dos maiores desafios para os algoritmos de reconhecimento facial. A oclusão pode ser classificada como: inexistente, parcial ou completa.

* 1. A oclusão inexistente é um rosto frontal, bem iluminado e sem partes cobertas, como mostrado na figura XX.a. Quando não existe oclusão, o rosto está completamente exposto e pode ser facilmente detectado pelo algoritmo.
  2. A oclusão parcial possui uma grande variedade de tipos, podendo ser, mas não se limitando a: um acessório que cubra parte do rosto, como ilustrado na figura XX.b, uma foto de perfil com o rosto parcialmente virado ou uma iluminação ruim. A oclusão parcial pode gerar resultados positivos ou negativos, dependendo de como a detecção está feita.

3. A oclusão completa, não há reconhecimento de rosto, pois o mesmo se encontra totalmente encoberto, como na figura XX.c.

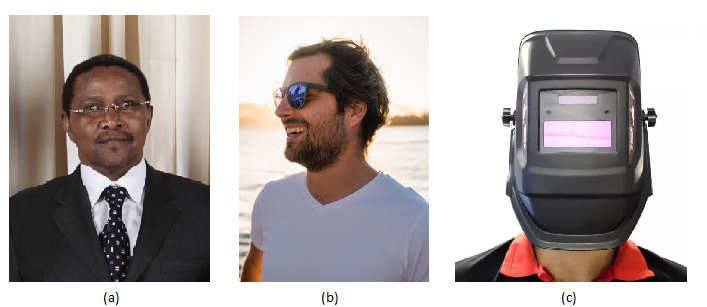


Figura XX – a. Oclusão inexistente (foto frontal), b. Oclusão parcial (foto perfil com acessórios) e c. Oclusão completa (foto de rosto completamente coberto pela máscara)

Atualmente um grande número de pesquisadores se dedicam ao estudo e aprimoramento de técnicas que sejam capazes de superar a barreira que a oclusão parcial representa. Entretanto, o *DeepFace* do Facebook, consegue reconhecer grande com sucesso uma alta quantidade de oclusões diferentes, possuindo uma precisão maior que 97%¹.

Esse trabalho focará apenas na análise de fotografias (2D), uma vez que está sendo analisado para ser incorporado a um aplicativo de celular, deixando de lado a avaliação de soluções 3D. A próxima seção, consiste em uma análise genérica sobre como funciona o processo de reconhecimento facial nesse tipo de imagem.

* 1. **Processo para o reconhecimento facial**

Antes de trazer as abordagens utilizadas para classificar os diversos algoritmos existentes é necessária uma breve explicação de como funciona um sistema de reconhecimento facial genérico, em outras palavras, saber como funciona o processo clássico de identificação. A explicação abaixo busca explicitar elementos em comum em qualquer técnica de reconhecimento facial para imagens bidimensionais.

O desenvolvimento de um *framework* de reconhecimento facial pode ser organizado em alguns passos, sendo eles: entrada de dados, localização do rosto, extração de características, classificação e decisão.

Decisão

Localização facial

Pré-processamento

Classificação

Imagem de entrada

Figura XX – Etapas de um processo de reconhecimento facial

Ao receber uma imagem de entrada, o algoritmo deve ser primeiramente capaz de reconhecer o que dentro daquela imagem é um rosto. Para tanto, é utilizado algum método de detecção de faces como dito na seção anterior. Conseguir identificar que existe um rosto em uma foto, indicará ao algoritmo em qual subseção da imagem ele deverá trabalhar, podendo descartar o restante que não será utilizado no processo de reconhecimento.

A normalização é o processo de padronização desta subseção. Sua intenção é transformar o primeiro recorte do rosto no mais próximo possível de uma imagem frontal, podendo: escalonar, girar e padronizar cores, se necessário. Após isso, será extraído apenas o que realmente importa, como visto na figura XX.

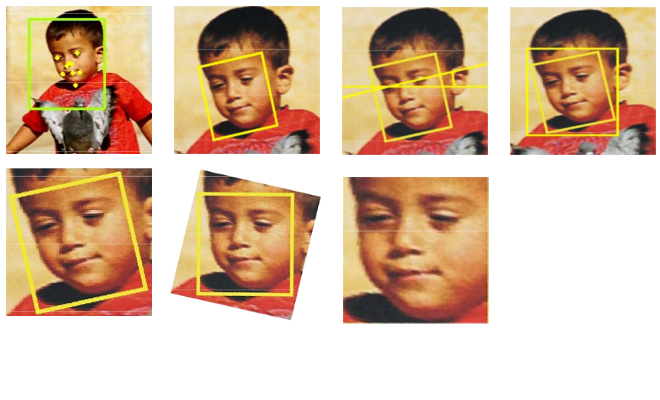


Figura XX – A detecção de um rosto e o processo de normalização de uma imagem

Nesta etapa, o algoritmo poderá redimensionar a imagem para reduzir o número de características encontradas e, por conseguinte, a carga computacional utilizada. E assim encontrar, ao extrair as características faciais, o conjunto ótimo que será capaz de reconhecer o rosto.

Na classificação, onde de fato haverá o reconhecimento, o algoritmo deverá retornar se as pessoas comparadas são, de fato, a mesma. A identificação correta sempre será feita com um modo comparativo entre duas imagens de cada vez, a que está sendo utilizada como dado de entrada e a que está no banco de dados. O método de identificação irá variar para cada tipo de abordagem realizada, podendo se utilizar de algoritmos de classificação ou métodos estatísticos ou matemáticos.

Na próxima seção será abordado a ideia por detrás dos métodos de reconhecimento facial utilizados hoje em dia.

* 1. **Métodos**

O reconhecimento facial pode ser feito de diversas formas, com ideias e enfoques diferentes. Na figura XX, é ilustrado 3 (três) abordagens que, atualmente, abrangem a área reconhecimento facial: abordagem holística, abordagem baseada em características e abordagem híbrida.

Abordagem holística

Reconhecimento facial

Abordagem baseada em características

Abordagem híbrida

Figura XX – Algumas abordagens de reconhecimento facial para imagens bidimensionais

**Abordagem holística**

Na abordagem holística o rosto é tratado como um objeto de análise único, não se preocupando em estudar separadamente cada uma das características individuais da face, ou seja, não tratando isoladamente os olhos, a boca, o nariz e etc.

O processamento da imagem baseia-se no reconhecimento de pequenos aspectos do rosto inteiro, tomando cuidado ao analisar cada pixel da imagem e calcular a variância entre eles para fazer a classificação.

Entre os métodos que utilizam essa abordagem estão as análises estatísticas e os que utilizam inteligência artificial.

**Abordagem baseada em características (ou métodos geométricos)**

A abordagem baseada em características consiste na ideia de mapear as características individuais do rosto como, olhos, boca e nariz, para fazer o reconhecimento. Em seguida, o processamento da imagem será feito através da relação geométrica entre os pontos criados.

Grafos são utilizados para fazer esse mapeamento e representar a relação entre as características, como na Figura XX.

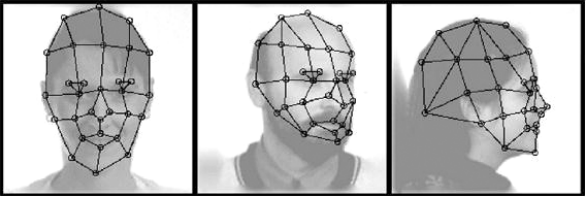


Figura XX – Grafos mapeando o método por características

**Abordagem híbrida**

Como o próprio nome já diz, a abordagem híbrida é uma mistura de técnica utilizadas na abordagem holística e na abordagem baseada em características. É o que mais se aproxima do conceito humano, uma vez que as pessoas são capazes de identificar pessoas por características individuais ou pelo rosto inteiro. A ideia é conseguir aproveitar o melhor das duas técnicas e criar novos algoritmos mais eficientes e robustos.

Após o entendimento de como funciona o reconhecimento facial, percebe-se que se trata de uma ferramenta poderosa que pode ir muito além da área de segurança, expandido para outros mercados, como marketing e automatização de tarefas. O próximo capítulo discute, com foco em segurança pública, sobre aplicações de reconhecimento facial no dia a dia.

1. Aplicações de Reconhecimento Facial

Este capítulo tem por objetivo introduzir algumas aplicações de reconhecimento facial, focando principalmente, porém não exclusivamente, em segurança pública. A abordagem estará separada em duas categorias diferentes, reconhecimento facial para segurança pública e reconhecimento facial para outros segmentos, e visa esclarecer as principais similaridades e diferenças entre elas.

* 1. **Reconhecimento facial para segurança pública**

Segundo a Secretaria Estadual de Desenvolvimento Social e Direitos Humanos do Rio de Janeiro (agosto/2019), o estado possui em torno de 15 (quinze) pessoas desaparecidas por dia, o que soma mais de 400 (quatrocentas) pessoas em um mês. No país inteiro, os valores chegam a 8 (oito) pessoas desaparecidas por hora.

Com números tão expressivos, iniciativas de aliar tecnologia à inteligência da polícia surgiram por todo o país, possibilitando a criação de aplicativos para celular, como as delegacias virtuais, e realizando pesquisas que envolvem reconhecimento facial para a consultar os bancos de dados da polícia, como o aplicativo criado por Vinícius Rodrigues, aluno da Universidade Federal do Estado do Rio de Janeiro (UNIRIO) [x].

**Reconhecimento facial por vídeo**

Atualmente existe uma grande quantidade de câmeras de segurança espalhadas por centros com grande circulação. Essa constante vigilância, aliada ao poder do reconhecimento facial pode se tornar um grande facilitador na hora da busca por pessoas. Soussa et al [x] apresenta a utilização dessas câmeras de segurança no processo de busca de pessoas desaparecidas e no auxílio de investigações criminais utilizando uma das primeiras técnicas bem sucedidas de reconhecimento facial, o *eigenfaces¹.* Em 2019, no Rio de Janeiro foi comprovada a eficiência dessa sugestão quando foram utilizadas com sucesso imagens de câmeras de segurança no processo de investigação, em casos voltados para a política de segurança pública.

Diferente de uma fotografia que pode ser tirada em condições ideias: uma imagem frontal, com o rosto livre de acessórios e com o fundo menos ruidoso possível, o reconhecimento facial em vídeo, não costuma dispor de uma imagem nessas condições.

Para ter uma visão melhor dos arredores, essas câmeras costumam estar localizadas em uma altura elevada, possuindo uma visão ligeiramente inclinada das pessoas. Além disso, questões como iluminação, qualidade de vídeo, posicionamento de face e vestimentas/acessórios podem impedir o algoritmo de reconhecer um rosto. Em casos de ruas muito movimentadas, o sistema deve ser capaz de reconhecer dezenas de transeuntes ao mesmo tempo.

O processo de reconhecimento em vídeo, porém, não é de todo muito diferente do reconhecimento de uma fotografia, uma vez que o algoritmo trabalha verificando cada quadro (*frame)* como se fosse uma imagem estática. Após o reconhecimento de uma pessoa, para que a marcação do identificador seja feita de maneira constante em um vídeo, como na Figura XX, o algoritmo aplicará o classificador em todos os frames seguintes até o rosto sair do alcance da câmera. Para tratar da análise de vários rostos ao mesmo tempo, é preciso que o algoritmo seja multi-instanciado de modo que consiga rodar paralelamente o algoritmo para cara rosto na imagem.



Figura XX – Pessoas na China tendo seus rostos reconhecidos por algum algoritmo de reconhecimento facial em tempo real (AUTOR/FONTE, ANO).

Um forte sistema de reconhecimento facial vem sendo implementado pelo governo chinês há anos, que atualmente consegue encontrar crianças desaparecidas em todo o país, através das câmeras de segurança. A base de dados conta com mais de 20 (vinte) redes sociais disponíveis e já foi capaz de encontrar mais de 6 (seis) mil crianças ao longo de três anos.

Apesar do grande desafio que é reconhecer um rosto tempo real, inúmeros algoritmos gratuitos disponibilizados na internet já são capazes de fazer reprodução em vídeo, como, por exemplo, o *Facial Recognition*, de Adam Geitgey, e *OpenFace*, ambos testados neste trabalho.

**Reconhecimento facial em aplicações (Imagens estáticas)**

Apesar de não haver uma diferença muito grande no processo de reconhecimento de um rosto, o reconhecimento facial de uma imagem estática costuma ser usado de outras maneiras.

A principal diferença entre a utilização da técnica entre o vídeo e uma imagem estática é apenas que o vídeo é um conjunto de fotos tiradas em uma frequência muito rápida, que conferem a ideia de movimento ao serem colocadas em sequência. Enquanto a imagem é apenas um elemento desse conjunto de dados.

Desse modo, no final não há nenhuma diferença relevante entre o algoritmo que deve ser aplicado, uma vez que tanto para vídeos, quanto para imagens, dado que este sempre processará uma imagem por vez. A diferença está no propósito de utilização.

Para o caso de busca de pessoas desaparecidas, Moisés et al [x] utiliza fotografias ao invés de vídeos, como proposto por Soussa et al. A utilização de fotos implica que a aplicação deva ser alimentada com uma imagem por vez, seja de forma manual ou automatizada, para que seja o banco seja percorrido fazendo as devidas comparações.

Assim como em vídeo, é possível reconhecer vários rostos ao mesmo tempo se o algoritmo estiver sendo executado em instâncias paralelas. Logo, se a foto comparada, por exemplo, é uma foto de família, é possível reconhecer todos os rostos de uma vez se o programa for capaz de ser instanciado para cada rosto detectado.

**Reconhecimento facial para outros segmentos**

Reconhecimento facial em tempo real pode ser utilizado para questões que vão além da temática de segurança, muitas vezes buscando melhorar ou automatizar alguma tarefa já existente.

O *PresentEye* (Mattos, 2017), por exemplo, é uma aplicação que se utiliza de câmeras de em salas de aula para verificação automática de presença de alunos em aula. Através do uso de reconhecimento facial, a aplicação consegue identificar todos os alunos de uma vez, acelerando o processo de checagem que antes era feito manualmente para cada aluno presente.

Muitas companhias hoje em dia se utilizam de ferramentas de detecção de rostos e reconhecimento facial para validação de transações financeiras, como a *MasterCard* com o seu sistema *Identity Check Mobile*. Através dele, o usuário dos cartões de crédito da bandeira, conseguem finalizar transações com a digital ou com uma foto do rosto tirada na hora, ao invés da utilização tradicional de senhas.

Profissionais da área de marketing também estão usufruindo da facilidade do reconhecimento facial para direcionarem melhor suas campanhas, por exemplo, em São Paulo, uma das linhas metrô passou a identificar as reações das pessoas à comerciais através de portas interativas instaladas nos trens.

Aliado a outros tipos de tecnologias, como por exemplo, a realidade aumentada, o reconhecimento facial começa a revolucionar a maneira de se fazer comércio. Existem estudos que procuram unir os óculos inteligentes à tecnologia. A ideia é utilizar o reconhecimento facial para identificar o cliente e automaticamente trazer à tela dos óculos o histórico de compras do consumidor, suas reclamações, etc, a fim de personalizar ao máximo a experiência do cliente com a empresa.

1. **Ferramentas e Tecnologias**

Com o objetivo de avaliar a viabilidade da utilização do reconhecimento facial, foram testados dois algoritmos e para comparar sua eficiência, algumas ferramentas precisaram ser utilizadas de forma direta ou indireta. As escolhas foram motivadas, pela linguagem de programação, que poderiam ser utilizadas e pelo licenciamento, dando preferência à ferramentas gratuitas. Avaliações do GitHub foram essenciais na decisão das bibliotecas também.

**Python**

Python é uma linguagem interpretada, conhecida pela sua sintaxe simples e amigável, o que lhe concedeu fama e rápido reconhecimento. É uma linguagem para trabalhar com *frameworks* para criaçãode servidores para páginas *web* e com a área de ciência de dados.

Esta linguagem foi escolhida para ser usada neste trabalho por ser a utilizada no desenvolvimento do servidor do aplicativo Desaparecidos-RJ, além de possuir uma variedade de bibliotecas de reconhecimento facial.

**PyCharm - Professional (versão licenciada para estudantes)**

PyCharm é um ambiente de desenvolvimento para a linguagem Python e seus *frameworks* desenvolvido pela empresa JetBrains. Este software foi escolhido para ser utilizado por causa da facilidade da instalação de pacotes e familiaridade com o funcionamento da ferramenta.

Por ser um *software* pago, em sua versão completa, foi requisitada a licença especial de estudantes. Não obstante, este trabalho também poderia ser desenvolvido utilizando a sua versão gratuita, *Community,* disponível na sua página *web*.

**Redes Neurais Convolucionais**

Redes Neurais são modelos de aprendizado de máquina que se propõe a simular o funcionamento de um sistema nervoso humano. Redes Neurais Convolucionais (CNNs) são provavelmente o modelo de rede *Deep Learning* mais conhecido e utilizado atualmente. (PONTI e COSTA, 2017, p.74).

Muito comum no tratamento de imagens, sua criação tem a intenção de se aproximar o máximo possível dos aspectos biológicos da visão. O funcionamento dos neurônios de uma rede neural convolucional é inspirado na parte do cérebro responsável pela visão humana, atribuindo pesos e valores para elementos de uma imagem. Segundo Vargas, Carvalho e Vasconcelos (2016, p.1): “uma CNN é capaz de aplicar filtros em dados visuais, mantendo à relação de vizinhança entre os pixels da imagem ao longo do processamento da rede”.

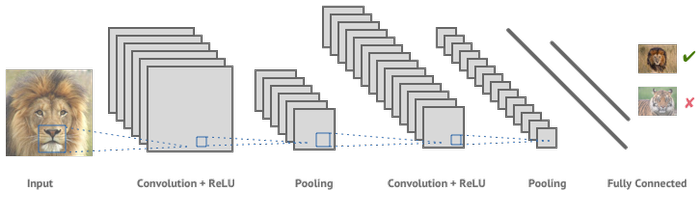


Figura XX - Exemplo de uma rede neural convolucional

A figura XX mostra o funcionamento de uma rede neural convolucional, dado uma imagem de entrada. Na primeira etapa, a de convolução, haverá uma série de neurônios responsáveis pela aplicação de filtros em pedaços específicos da imagem. Em sua essência, um filtro é um vetor ou uma matriz pequena que percorre a matriz de entrada, fazendo cálculos que transformam o dado de entrada em uma matriz menor, ou vetor. Existem diversos filtros diferentes, como mostra a figura XX.

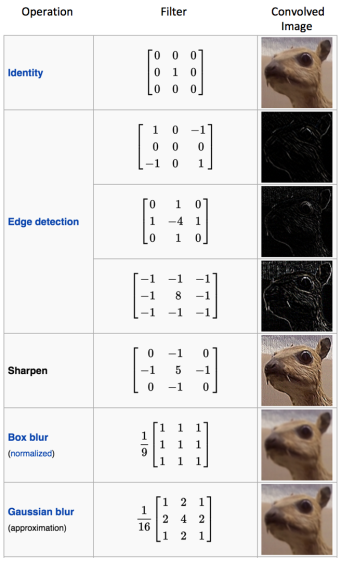


Figura XX – Tipos de filtros de imagem

Aplicados os filtros, a próxima etapa é a aplicação de uma função de ativação. As funções de ativação têm como objetivo retirar a linearidade dos resultados obtidos anteriormente, trazendo uma maior capacidade interpretativa a rede neural, uma vez que a realidade não segue um padrão linear. Existem diversas funções de ativação, tais como:  [Sigmoide](https://matheusfacure.github.io/2017/07/12/activ-func/" \l "sig), [Tangente Hiperbólica (TanH)](https://matheusfacure.github.io/2017/07/12/activ-func/#tanh) e [Unidade Linear Retificada (ReLU)](https://matheusfacure.github.io/2017/07/12/activ-func/#relu).

Após a fase de convolução, existe uma fase chamada *Pooling*, responsável pela extração de apenas características importantes e sua redução de tamanho. Tratando-se de matrizes, ele irá aplicar um filtro para subdividir a matriz ou vetor de dados vindo da fase anterior, podendo este filtro ser diferente em cada rede neural. Alguns exemplos de filtros são: maior valor, soma e média.

A Figura XX mostra o processo de *pooling* com filtro de maior valor:

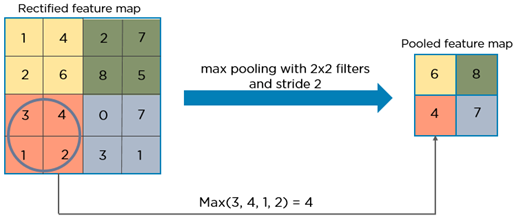


Figura XX – *Pooling* com filtro de maior valor

Essas etapas são extremamente importantes, pois diminuem a carga de processamento das redes neurais. Como visto na figura XX, a convolução e o *pooling* fazem parte da seção de extração de características e ambas ocorrem, no mínimo, duas vezes antes de ir para a seção de classificação. Isto garante a possibilidade de reaplicação de filtros para retirar ruídos restantes ou a aplicação de novos filtros em cima daquela mesma etapa.

A seção de classificação, seguinte a seção de extração de características, possui a fase conhecida como “Totalmente conectada”. Nesta fase se encontra a rede neural *perceptron*, responsável por conectar todos os neurônios de uma camada anterior, a todos da camada seguinte para produzir um resultado final. A cada camada é atribuído um peso que servirá como parâmetro de comparação e irá determinar o valor daquele neurônio ao passar para frente. Cada camada é responsável pela tomada de uma decisão importante para o resultado.

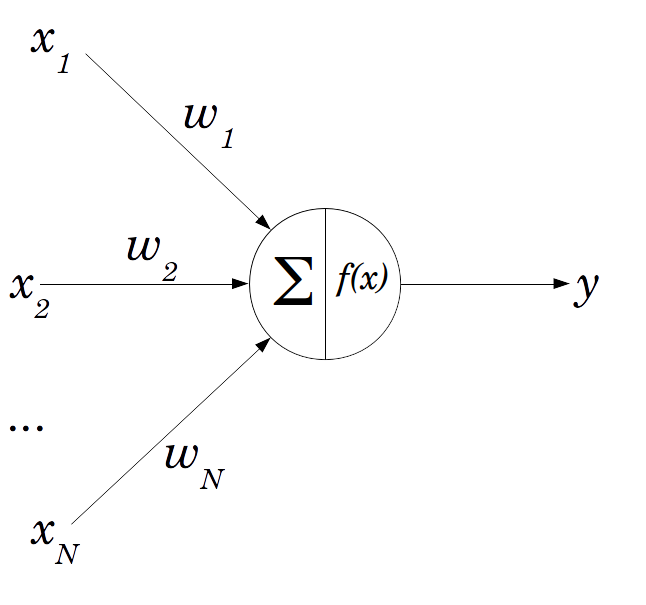


Figura XX – Exemplo de *perceptron*

A figura XX ilustra a ideia de um *perceptron*. As entradas x1, x2, ..., xN estão vindo de posições diferentes e possuem pesos diferentes (w1, w2, ..., wN). O neurônio que está recebendo essas informações, fará uma soma ponderada com os valores recebidos e irá comparar o resultado com o limiar estabelecido para aquela camada. Ele identificará qual foi o nível de sucesso esperado e irá retornar um resultado y que estará ligado ao(s) neurônio(s) da próxima camada. Isso se seguirá até que haja uma camada com um único neurônio que irá comparar o resultado da soma com o limiar e classificar a imagem de entrada.

As redes neurais convolucionais foram utilizadas pelas bibliotecas *OpenFace* para treinar o modelo que será usado para a detecção de rostos. Este modelo também é utilizado pela outra biblioteca testada neste trabalho, o *Face Recognition.*

**Base de dados**

Para se trabalhar com reconhecimento facial, é necessário que exista uma base de fotografias de indivíduos que possa ser dividida entre dois tipos de dado:

1. Banco de dados: Representa um subconjunto dos dados onde os elementos são identificáveis e serão usados para comparações com as imagens que estão sendo usadas como entrada.
2. Dados de entrada: É um subconjunto da base de dados original onde, em teoria, as pessoas não possuem identificação. Eles serão usados para simular a busca por pessoas sem identidade em um banco de dados de rostos conhecidos.

A base de dados utilizada foi criada pela Universidade de Milano Bicocca. É uma base gratuita, porém de acesso restrito a universidades e pesquisadores, voltada para estudos de reconhecimento facial com oclusão parcial. São 590 (quinhentos e noventa) imagens com oclusão parcial dos mais diversos tipos: cabelos, óculos de sol, chapéus, etc.

**Biblioteca Dlib**

Inicialmente idealizada por Davis King, hoje com mais de 100 (cem) colaboradores, Dlib é uma biblioteca gratuita, desenvolvida em C++ que objetiva servir como uma coleção de componentes de *framework* para serem usados em qualquer tipo de aplicação, fins comerciais ou acadêmicos. Atualmente, disponibiliza módulos para suporte a Álgebra Linear, Processamento de imagens, Otimização, entre outros, todos com uma extensa documentação.

O site oficial da biblioteca oferece exemplos de classes desenvolvidas para detecção de face, alinhamento, clusterização, etc. Para detecção de rostos, a Dlib utiliza Redes Neurais Convolucionais.

Esta biblioteca é usada pelos algoritmos escolhidos para serem testados e é responsável pela detecção de rostos em uma imagem.

**Face Recognition**

Criada por Adam Geitgey, face recognition é uma API de reconhecimento facial, baseado no estado da arte da biblioteca Dlib, desenvolvido em Python e disponibilizado gratuitamente no GitHub do próprio autor. Foi escolhida pela sua ótima recomendação no GitHub, com mais de 20.000 (vinte mil) estrelas, e sua simplicidade de acesso, sendo muito fácil sua utilização.

**FaceNet e a função *Tiplet Loss***

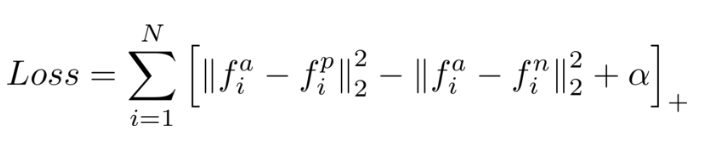
FaceNet é uma rede neural convolucional que mapeia rostos utilizando-se de espaços Euclidianos para encontrar similaridades dos rostos. Assim como a biblioteca Dlib, mostra uma enorme eficiência com bases de dados de grande porte, como as do YouTube, onde atingiu 95,12% (noventa e cinco vírgula doze porcento) de acurácia.

A função *triplet loss* do FaceNet agrupa resultados similares de modo que eles permaneçam próximos, e os diferentes de maneira que eles fiquem mais distantes. Para tanto, ele se utiliza da fórmula apresentada na figura XX e da codificação de face de três imagens que são classificadas como:

- Imagem âncora: Funciona como base comparativa. Ela é utilizada para criar um padrão de resultados.

- Imagem positiva: A imagem de uma mesma pessoa da imagem âncora. Os resultados serão bastante próximos do padrão esperado (o resultado da imagem âncora).

- Imagem negativa: Imagem de uma pessoa diferente. Os resultados vão ser distantes do determinado como base pela imagem âncora e, logo, serão agrupados juntos.



Equação XX – Fórmula da *Triplet Loss*

Esta função é utilizada para melhorar a performance do modelo, uma vez que irá agrupar as imagens das mesmas pessoas e separar em outros grupos as pessoas diferentes.

O FaceNet é aplicado em dois momentos na criação do modelo utilizado no OpenFace. Primeiramente, empregam o modelo pré-treinado do FaceNet para criar um vetor de 128 dimensões que representará um modelo genérico de rosto. Em seguida, utilizarão a função *triplet loss*  para a organização do modelo de modo que ele se torne mais performático.

**OpenCV**

A biblioteca OpenCV está dentre as mais famosas e utilizadas dentro da área de visão computacional. Sua intenção é servir como infraestrutura para o desenvolvimento de aplicativos voltados a visão computacional.

Também é utilizado pelo OpenFace, porém, na parte de normalização da imagem. Ele é responsável pela leitura da imagem e a padronização do esquema de cores.

**Torch**

Torch é um *framework*, desenvolvido em lua, de código aberto que busca prover uma estrutura para um melhor funcionamento de algoritmos que exijam grande poder computacional. É usado principalmente para fins científicos, trabalhando com otimização de numerais, álgebra linear, aprendizagem de máquina, etc. Ele possui suporte para GPU, tornando-se atrativo para desenvolvimento de códigos de visão computacional.

**OpenFace**

Open Face é uma biblioteca desenvolvida em Python com o propósito de ser utilizada em tecnologias móveis. Sua performance faz com que ela possa ser comparada com tecnologias e bibliotecas grandes, como a DeepFace do Facebook.

Utiliza-se de tecnologias como FaceNet, OpenCV e Torch, como mostra a Figura XX. Ela foi escolhida para este trabalho por causa de um número de recomendações no GitHub e ótima performance.

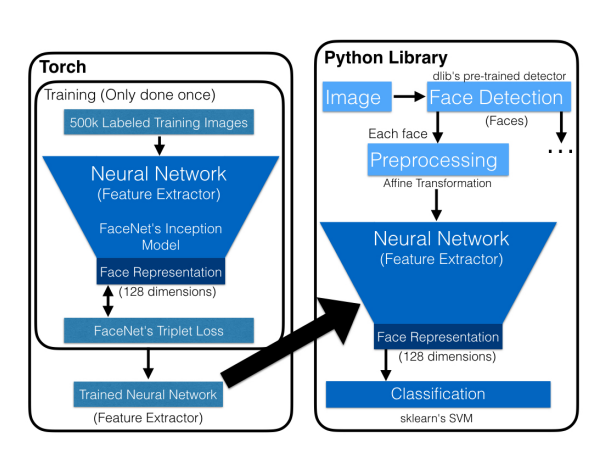


Figura XX – Arquitetura do algoritmo OpenFace

**Converseen**

Programa de edição de imagens gratuito para diversos sistemas operacionais. Este foi utilizado para converter as imagens da base de dados de seus formatos originais para que pudessem ser lidos propriamente pelos algoritmos utilizados nos testes.

Todas as imagens foram convertidas de X para .png para que a qualidade da imagem não fosse perdida no processo.

O próximo capítulo irá apresentar os testes realizados com as bibliotecas OpenFace e FaceRecognition apresentados nesta seção.

1. **Testes**

Os testes foram feitos utilizando 120 (cento e vinte) fotos da base de dados da Universidade de Milano-Bicocca criada especialmente para estudos de técnicas de reconhecimento facial com oclusão parcial. A base possui um total de 590 (quinhentos e noventa) imagens com oclusão variadas: cabelo, óculos, chapéus, cachecóis e etc.

Inicialmente foram selecionados 22 (vinte e dois) indivíduos, cada uma contendo entre 3 (três) e 12 (doze) imagens de uma mesma pessoa, enumeradas de forma ordenada, seguindo o seguinte padrão: X-Y.png, onde X é o número de identificação do indivíduo e Y, o número sequencial da imagem.

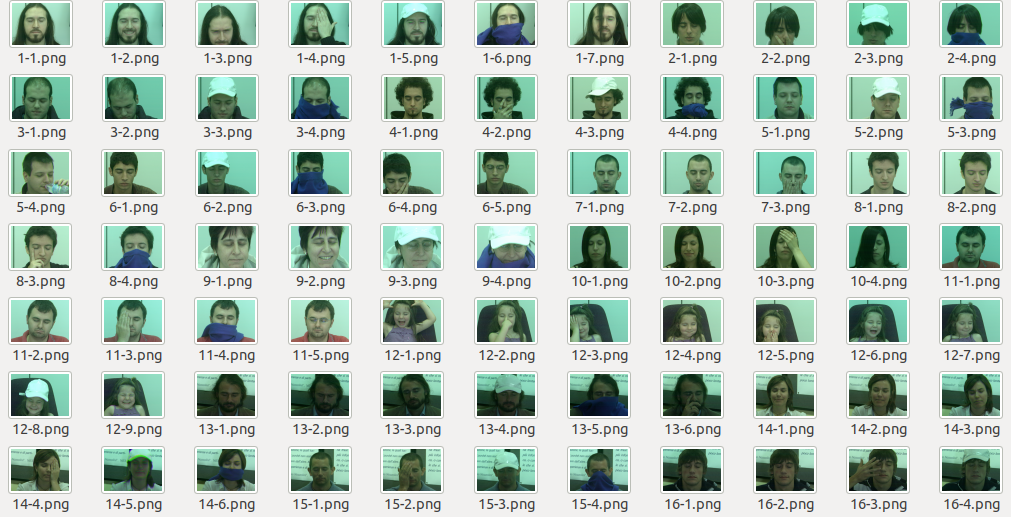


Figura XX – Organização da base de dados

Os testes feitos consistem na comparação entre uma imagem frontal de cada indivíduo, similar a 1-1.png mostrada na figura XX, e todas as imagens restantes do banco, analisando separadamente os seguintes resultados: imagens de um mesmo indivíduo ao comparar contra todas as imagens dele mesmo, as imagens de um indivíduo contra todo grupo em busca de falsos negativos e falsos positivos e tipos de oclusão com maior número de erros.

Ambos os *thresholds* possuem o valor 0.6. Este valor foi escolhido por ser o padrão da biblioteca *Face Recognition*. Segundo o autor da biblioteca, este é o número mais balanceado, onde há uma maior chance de reconhecimento sem trazer um grande número de falsos positivos ou falsos negativos. A decisão de utilizar este mesmo valor para o *OpenFace* não quer dizer que seja o valor ótimo para resultados, apenas que este trabalho testou eles em condições similares. Destaca-se, porém, a importância do *threshold*, parâmetro de tolerância utilizado para comparação entre pontos, indicador de resultados verdadeiros ou falsos.

O método de reconhecimento facial utilizado nesses testes se dá através da captura de 128 (cento e vinte e oito) pontos de cada um dos rostos que estão sendo comparados. Os cálculos são feitos utilizando uma equação matemática, ilustrada pela Equação XX. Para tanto, primeiro diminui-se o ponto A da figura 1 do ponto A da figura 2, obtendo um resultado. Eleva-se esse resultado ao quadrado. Repete-se todo o processo até o fim do cálculo da diferença de todos os 128 (cento e vinte e oito) pontos das imagens. Ao fim, soma-se os valores encontrados e, em seguida, calcula-se a raiz quadrada do resultado final. O valor encontrado é o que será comparado ao *threshold* e decidirá se a pessoa na figura 1 (um) é a mesma pessoa da figura 2 (dois). Desse modo, o *threshold* poderá variar de valores para ter mais acurácia (baixando-o) ou ser mais abrangente (aumentando-o).

=

Equação XX – Distância Euclidiana

* 1. **Limitações**

Os modelos pré-treinados disponibilizados por ambos as bibliotecas utilizam bases de dados públicas, onde o conteúdo varia de dezenas a centenas de milhares de fotos em uma grande variedade de posições. Elas, porém, não são feitas para análises de reconhecimento facial com oclusão parcial, o qual é o foco deste trabalho.

Ainda assim, devido a limitações computacionais, os modelos padrões utilizados como *default* foram mantidos durante os testes. Deste modo, os resultados esperados serão avaliados levando-se em consideração que não houve treinamento adequado para reconhecer todos os tipos de oclusão apresentados pela base de dados utilizada neste trabalho.

* 1. **Face Recognition**

O algoritmo do Face Recognition possui pacote disponível para *download* no gerenciador de pacotes do Python, o pip. Após o *download*, é possível utilizá-lo dentro de qualquer código Python importando-o como um pacote normal.

Em seu repositório no GitHub, existe uma breve explicação de como utilizar os códigos disponíveis, contudo, o código apresentado da Figura XX foi o principal pedaço utilizado neste trabalho.

Primeiramente, ele importa a biblioteca *face\_recognition* no código. Em seguida, ele endereça a imagem de origem e a imagem comparada para as variáveis *known\_image* (imagem conhecida) e *unknown\_image* (imagem desconhecida), respectivamente. Essas variáveis serão utilizadas para gerar dois vetores de 128 (cento e vinte e oito) pontos, um para cada imagem, que serão usados para fazer comparações. É importante saber que esses pontos serão gerados apenas se o algoritmo for capaz de reconhecer um rosto.

import face\_recognition

known\_image = face\_recognition.load\_image\_file("biden.jpg")

unknown\_image = face\_recognition.load\_image\_file("unknown.jpg")

biden\_encoding = face\_recognition.face\_encodings(known\_image)[0]

unknown\_encoding = face\_recognition.face\_encodings(unknown\_image)[0]

results = face\_recognition.compare\_faces([biden\_encoding], unknown\_encoding)

**Figura XX - exemplo de como utilizar a biblioteca**

As variáveis *biden\_enconding* (em homenagem a Joe Biden, ex-vice-presidente dos Estados Unidos, utilizado como exemplo pelo autor) e *unknown\_encoding* armazenam os pontos gerados pela imagem conhecida e pela imagem desconhecida, respectivamente.

Ao fim, ele chama um método para comparar os dois vetores e estimar se há uma correspondência de resultados.

Note, porém, que o detalhe mais importante deste pequeno trecho de código é a geração do vetor.

Em um teste de reconhecimento facial com foco em imagens com oclusão é importante que o vetor de pontos seja testado, uma vez que o algoritmo pode não ser capaz de compreender se existe um rosto na imagem. Da mesma forma, é importante que ao descobrir um problema na geração de vetores seja mapeado se o problema foi originado pela imagem de origem ou pela imagem desconhecida, quando não por ambas.

Sendo assim, os resultados possíveis dos testes são:

* [True], quando houver uma correspondência.
* [False], quando não houve uma correspondência.
* “Problema na imagem comparada”, quando não houver geração de vetor na imagem desconhecida.
* “Problema na imagem de origem”, quando não houver geração de vetor na imagem conhecida.
* “Problema nas duas imagens”, quando não houver geração de vetor em nenhuma das duas imagens.

**O código**

Para a utilização deste código neste teste, foi desenvolvido um pequeno programa em *python* separado em 4 (quatro) métodos:

1. Método de importação de endereços

Responsável pela criação de um dicionário de endereços de onde as imagens estão localizadas.

**Entrada:** O endereço E da pasta onde estão localizadas as imagens da base de dados

Criar um novo dicionário para armazenamento do endereço de imagens de cada indivíduo.

**Para todos** **os arquivos em E faça:**

**Para cada** nome\_arquivo em arquivos **faça:**

arquivo := nome\_arquivo.separar(-)

**Se** **arquivo[0] dicionario.chaves então:**

Criar uma nova lista para armazenar o endereço daquele arquivo. Em seguida, adicionar lista de imagens a dicionario[arquivo[0]].

**Senão:**

Adicionar endereço de arquivo à lista de arquivos no dicionário[arquivo[0]].

**Saída:** Dicionário com endereço de todos as imagens, separadas para cada indivíduo

Figura XX – Pseudo-código do método de importação de endereços da base de dados

Este é o primeiro método chamado pelo programa. Sua principal função é percorrer a pasta onde estão localizadas as imagens da base de dados e adicioná-las ao dicionário de indivíduos.

A identificação de cada indivíduo no dicionário é feita através da chave, enquanto os endereços de cada imagem daquele indivíduo são armazenados numa lista encadeada que está relacionada a cada chave.

1. Método de comparação

Faz a comparação de um indivíduo com todas as imagens do banco de dados. Este procedimento é repetido em um laço até que todas as imagens tenham sido comparadas.

Este é o principal método do código, é através dele que você fará as comparações, chamará os métodos de resultado e os exportará para o documento CSV.

Para conseguir comparar todas as imagens e retornar todas as variáveis necessárias para sua avaliação de desempenho foi necessário percorrer o dicionário duas vezes para recuperar a lista de indivíduo\_origem e indivíduo\_comparado.

**Entrada:** Dicionário D de endereço de todas as imagens da base de dados

**Para toda chave\_origem, lista\_imagem\_origem em D faça:**

**Para toda chave\_comparada, lista\_imagem\_comparada em D faça:**

**Para cada imagem\_comparada em Pi, i={0, tamanho(lista\_imagem\_comparada)} faça:**

Iniciar cálculo de tempo de cada reconhecimento

Chamar funções apresentadas na figura XX para fazer a detecção de rostos, utilizando  
os endereços para a primeira imagem\_origem de cada indivíduo e das demais imagens para imagem\_comparada.

Finalizar cálculo de tempo de cada reconhecimento

Chamar a função de cálculo de resultados e a função de exportação de resultados.

**Saída:** Documento CSV com os resultados finais.

Figura XX – Pseudo-código do método de comparação de imagens

As listas são percorridas, preenchendo os endereços de cada imagem no código da Figura XX. Em seguida, o método de verificação de resultado é chamado para averiguar se não há erros de detecção de faces em nenhuma das duas imagens. Após isso, o método de exportação de resultados é chamado.

1. Método de verificação de rosto

Verifica se há algum erro de reconhecimento de rosto e geração de um vetor de pontos. Caso haja, ele retorna um dos resultados de acordo com o teste de condicional aplicado.

Este método recebe os vetores de 128 pontos das duas imagens sendo comparadas com a intenção de descobrir se o vetor está vazio ou não. Se estiver, significa que não foi possível detectar um rosto em pelo menos uma das imagens e retornará uma mensagem indicando o erro.

Caso contrário, ele fará o cálculo da distância euclidiana utilizando-se da fórmula apresentada na Figura XX e retornará um resultado *booleano*.

**Entrada:** Vetor V1 gerado pelo algoritmo ao detectar um rosto na imagem de origem e o vetor V2 também gerado pelo algoritmo ao detectar um rosto na imagem comparada.

**Se** **tamanho(V1) == 0 e tamanho(V2)==0 então:**

A variável resultado receberá o valor “Problema nas duas imagens”

**Senão se** **tamanho(V1) == 0 então:**

A variável resultado receberá o valor “Problema na imagem de origem”

**Senão se** **tamanho(V2) == 0** **então:**

A variável resultado receberá o valor “Problema na imagem comparada”

**Senão:**

A variável resultado receberá o valor de comparação entre os dois vetores, podendo assumir os valores [True] ou [False]

**Saída:** O resultado do reconhecimento facial

Figura XX – Pseudo-código do método de verificação de resultados

1. Método de exportação de resultados

Exporta o resultado em um arquivo .csv.

**Entrada:** Individuo Ind\_Ori da imagem de origem, a imagem de origem Img\_Ori, o indivíduo Ind\_Com da imagem de comparação, sua imagem que está sendo comparada Img\_Com, o Resultado final da comparação e o tempo de reconhecimento.

**Abrir** arquivo “resultado-face\_recognition.csv”

**Acrescentar** à última linha cada variável na seguinte ordem: Ind\_Ori, Img\_Ori, Ind\_Com, Img\_Com, Resultado, tempo\_reconhecimento. Quebrar a linha no final.

**Fechar** arquivo.

**Saída:** Arquivo CSV acrescido do resultado da última comparação feita.

Figura XX – Pseudo-código do método de exportação de resultados

Este método é responsável pela criação, abertura, acréscimo de dados e fechamento do arquivo onde os resultados deste teste estão sendo documentados. Ele recebe cinco parâmetros que serão acrescentados ao final do documento, sendo eles: Indivíduo\_Origem, Imagem\_Origem, Indivíduo\_Comparado, Imagem\_Comparada, Resultado e Tempo.

* 1. **OpenFace**

Para a instalação do OpenFace é aconselhado seguir os tutoriais apresentados no site do laboratório desenvolvedor, cmusatyalab², onde será possível encontrar diferentes métodos de instalação.

Para esses testes, foi reaproveitado o modelo do código de demonstração disponibilizado no repositório do GitHub³ junto à biblioteca. O código é simples, possui apenas chamadas de carregamento de bibliotecas e modelos, uma função para o processo de normalização e a iteração onde são feitas as comparações entre as imagens de origem e as imagens comparadas.

Contudo, para que os resultados pudessem ser obtidos de modo satisfatório foram necessárias algumas modificações. Para tanto, foi criada uma classe chamada ReconhecimentoFacial() com 4 (quatro) métodos, responsáveis por todo o processo de reconhecimento:

1. Método de importação de endereços

Responsável pela criação de um dicionário de endereços de onde as imagens estão localizadas. Exatamente igual ao teste do *Face Recognition,* ver Figura XX.

1. Método utilizado para o processo de normalização

Esse método é responsável pela chamada das funções básicas do pré-processamento das imagens.

**Entrada:** Dicionário de imagens

Carregar a imagem E

**Se imagem não pôde ser carregada:**

Levantar exceção de “Imagem não carregada”

Alterar esquema de cores de imagem para RGB

Encontrar o maior rosto detectado na imagem carregada

**Se não pôde encontrar nenhum rosto:**

Levantar exceção de “Rosto não encontrado”

Transformar e alinhar rosto encontrado na imagem carregada

**Se não pôde alinhar o rosto:**

Levantar exceção de “Incapaz de alinhar o rosto”

Alimentar a rede neural com a imagem transformada para extrair suas características

**Saída:**Vetor de características extraídas pela rede neural.

Figura XX – Método de exportação de resultados

Esse método cuida de toda a parte principal de pré-processamento do código. Ele foi retirado da demonstração apresentada pelos criadores, com poucas modificações, deixando apenas o essencial para o funcionamento do código.

Ele faz correção de cores da imagem para padronização, encontra 68 (sessenta e oito) pontos que são utilizados para detectar um rosto e, por último, transforma a imagem e a alinha.

Ao fim, o resultado final das transformações passa pela rede neural para extração das características do rosto que serão transformadas num vetor de 128 (cento e vinte e oito) pontos.

1. Método de análise e extração de resultados:

O método apresentado na Figura XX é responsável pela comparação das imagens para extração de resultados.

**Entrada:** Dicionário D de endereço de todas as imagens da base de dados

**Para toda chave\_origem, lista\_imagem\_origem em D faça:**

**Para toda chave\_comparada, lista\_imagem\_comparada em D faça:**

**Para cada imagem\_comparada em Pi, i={0, tamanho(lista\_imagem\_comparada)} faça:**

Iniciar cálculo de tempo

Imagem\_origem = Endereço + indivíduo\_origem + “-1.png”

Imagem1 = False

Imagem2 = False

**Tentar:**

A variável auxiliar da imagem origem tenta receber o vetor gerado pelo método de normalização, V1.

**Exceção:**

Imagem1 = True

**Tentar:**

A variável auxiliar da imagem origem tenta receber o vetor gerado pelo método de normalização, V2

**Exceção:**

Imagem2 = True

Finalizar cálculo de tempo

**Se Imagem1 == False ou Imagem2== False, então:**

**Se Imagem1 == True e Imagem2==True, então:**

A variável resultado receberá o valor “Problema nas duas imagens”

**Senão se Imagem1 == True e Imagem2 ==False, então:**

A variável resultado receberá o valor “Problema imagem de origem”

**Senão se Imagem2 == True e Imagem1 ==False, então:**

A variável resultado receberá o valor “Problema imagem comparada”

**Senão:**

Faz-se a diferença entre os vetores encontrados, V1 - V2, encontrando o resultado R. Em seguida, aplica-se um método de multiplicação de matrizes sobre R para elevá-lo ao quadrado.

**Se R<=0.99:**

A variável resultado receberá o valor “True”

**Senão:**

A variável resultado receberá o valor “False”

**Saída:** Resultado da comparação das imagens.

Figura XX – Pseudo-código do método de comparação e extração de resultados

Primeiro, ele tenta fazer a normalização de cada uma das imagens que estão sendo comparadas, separadamente. A divisão foi criada, de modo que, através de variáveis booleanas, é possível fazer o controle e rastrear, em casos de falha, suas devidas origens. Para estes erros, os resultados seguem o mesmo padrão utilizado para os testes do *Facial Recognition*.

Em caso de sucesso na normalização das duas imagens, será aplicada a distância Euclidiana Quadrática, a qual segue a mesma ideia da Distância Euclidiana apresentada na Figura XX, porém, sem a aplicação da raiz quadrada.

Equação XX – Fórmula da Distância Euclidiana Quadrática

A razão de se não usar a Distância Euclidiana apresentada na Figura XX, deve-se ao fato de que ao não fazer a raiz quadrada, existe um ganho de tempo e, logo, uma maior eficiência nos cálculos de comparação.

1. Método de exportação de resultados

Exporta o resultado em um arquivo .csv. Exatamente igual ao teste do *Face Recognition*, ver Figura XX.

Para chamar a classe ReconhecimentoFacial() com sucesso, porém, existe a necessidade de atribuir algumas variáveis responsáveis pelo carregamento dos modelos e dados utilizados.

fileDir = os.path.dirname(os.path.realpath(\_\_file\_\_))

modelDir = os.path.join(fileDir, '..', 'models')

dlibModelDir = os.path.join(modelDir, 'dlib')

openfaceModelDir = os.path.join(modelDir, 'openface')

Figura XX – Código responsável pelo endereçamento das pastas

Inicialmente, são atribuídos caminhos padrões a variáveis. Esses endereços são utilizados para encontrar arquivos em caminhos *default*, ou seja, caso não haja alterações no código ou na organização das pastas, esses endereços serão usados como referência para recuperar os arquivos necessários para o funcionamento da classe.

parser = argparse.ArgumentParser()

parser.add\_argument('--dlibFacePredictor', type=str, help="Path to dlib's face predictor.",

default=os.path.join(dlibModelDir, "shape\_predictor\_68\_face\_landmarks.dat"))

parser.add\_argument('--networkModel', type=str, help="Path to Torch network model.",

default=os.path.join(openfaceModelDir, 'nn4.small2.v1.t7'))

parser.add\_argument('--imgDim', type=int,

help="Default image dimension.", default=96)

args = parser.parse\_args()

Figura XX – Código responsável pela parametrização de componentes utilizados na biblioteca

Após isso, são importados os arquivos *default* ou os passados por parâmetro através da linha de comando.

Esses arquivos são fundamentais para o funcionamento do código, pois tratam-se dos modelos treinados para detecção e reconhecimento de rostos. Também pode ser passado como parâmetro o tamanho, em pixels, que a imagem terá após o seu redimensionamento. Lembrando que quanto menor o tamanho da imagem, mais rápido é seu processamento.

align = openface.AlignDlib(args.dlibFacePredictor)

net = openface.TorchNeuralNet(args.networkModel, args.imgDim)

Figura XX – Código responsável pela inicialização das redes neurais utilizadas

Por fim, os modelos e dados importados são carregados e podem ser utilizados pelo método de normalização da classe ReconhecimentoFacial().

A próxima seção trará a análise dos resultados gerados pelos testes descritos nessa seção. Os códigos completos podem ser encontrados no Anexo Y ou no repositório do GitHub4.

1. **Resultados**

Os resultados deste trabalho foram analisados tendo em vista sua utilização prática em um aplicativo voltado para a segurança pública. Para tanto foram avaliados:

* Acertos – Imagens reconhecidas corretamente pelo algoritmo;
* Falsos Positivos – Imagens de indivíduos diferentes que foram reconhecidas;
* Falsos Negativos – Imagens de uma mesma pessoa que ao serem comparadas, retornaram resultados negativos, ou seja, não houve o reconhecimento;
* Tempo – O tempo médio para reconhecer cada rosto/levantar cada erro.
* Performance – Porcentagem de acertos dentro do universo de rostos reconhecidos como um indivíduo.
* Erros – Problemas de detecção de faces devido a alguma oclusão.

**Acertos**

Acertos em teste de reconhecimento facial significam o algoritmo foi capaz de reconhecer o indivíduo A como ele mesmo. Porém, como o foco desse trabalho é reconhecimento facial com oclusão, os acertos também significam que os algoritmos são capazes de reconhecer a pessoa independentemente do tipo de oclusão da imagem.

* Indivíduo – Número identificador do indivíduo no banco de dados
* Qtde. Imagens – Quantidade de imagens que aquele indivíduo possui na base de dados
* Face Recognition – Número de acertos obtidos no teste com a biblioteca Face Recognition
* OpenFace – Número de acertos obtidos no teste com a biblioteca OpenFace

|  |  |  |  |
| --- | --- | --- | --- |
| **Indivíduo** | **Qtde Imagens** | **Face Recognition** | **OpenFace** |
| 1 | 7 | 7 | 5 |
| 2 | 4 | 1 | 1 |
| 3 | 4 | 2 | 2 |
| 4 | 4 | 4 | 3 |
| 5 | 4 | 3 | 2 |
| 6 | 5 | 4 | 3 |
| 7 | 3 | 3 | 2 |
| 8 | 4 | 3 | 3 |
| 9 | 4 | 3 | 3 |
| 10 | 4 | 3 | 3 |
| 11 | 5 | 5 | 4 |
| 12 | 9 | 8 | 6 |
| 13 | 6 | 5 | 3 |
| 14 | 6 | 5 | 4 |
| 15 | 4 | 2 | 2 |
| 16 | 5 | 3 | 3 |
| 17 | 6 | 3 | 3 |
| 18 | 4 | 2 | 2 |
| 19 | 12 | 12 | 1 |
| 20 | 8 | 5 | 4 |
| 21 | 6 | 4 | 2 |
| 22 | 6 | 6 | 2 |
| **Total:** | **120** | **93** | **63** |

Tabela XX – Número de acertos dos algoritmos testados

De acordo com os resultados apresentados, o algoritmo Face Recognition possui uma margem de acerto bastante superior ao algoritmo do OpenFace. A razão para tal diferença pode estar em várias etapas do código. Alguns exemplos que poderiam causar isso: precisão do modelo, valor usado como *threshold*, etc.

‘ Entretanto, em uma análise superficial dos resultados apresentados pelo algoritmo OpenFace, encontrados na tabela XX (anexo Y), ao comparar os valores de acertos com os valores falsos negativos apresentados pode-se dizer que existe uma possibilidade da necessidade de ajuste do valor de *threhsold*.

Como explicado no capítulo de testes, os *thresholds* são elementos essenciais na hora da classificação de um resultado. Ele é, em partes, responsável pela precisão dos acertos, uma vez que ao ser alterado pode reconhecer rostos demais, ou rosto de menos.

Para aumentar o número de acertos, neste caso, deve-se aumentar o valor do *threshold,* porém, com a ressalva de que isso pode aumentar também o número de falsos positivos do resultado.

**Falsos positivos**

Falsos positivos, como a própria expressão diz, são resultados positivos que não são verdadeiros. Em reconhecimento facial, um falso positivo é quando o algoritmo reconhece o indivíduo B, como indivíduo A, sendo A e B pessoas distintas. A tabela XX ilustra a quantidade de falsos positivos de ambos os algoritmos.

* Indivíduo – Número identificador do indivíduo no banco de dados
* Qtde. Imagens – Quantidade de imagens que aquele indivíduo possui na base de dados
* Face Recognition – Número de falsos positivos obtidos no teste com a biblioteca Face Recognition
* OpenFace – Número de falsos positivos obtidos no teste com a biblioteca OpenFace

|  |  |  |  |
| --- | --- | --- | --- |
| **Indivíduo** | **Qtde Imagens** | **Face Recognition** | **OpenFace** |
| 1 | 7 | 6 | - |
| 2 | 4 | 1 | - |
| 3 | 4 | - | 3 |
| 4 | 4 | 1 | 1 |
| 5 | 4 | - | 7 |
| 6 | 5 | 7 | 1 |
| 7 | 3 | 5 | 5 |
| 8 | 4 | - | 2 |
| 9 | 4 | 1 | - |
| 10 | 4 | 2 | 1 |
| 11 | 5 | 3 | 3 |
| 12 | 9 | - | - |
| 13 | 6 | 1 | 1 |
| 14 | 6 | - | 1 |
| 15 | 4 | 5 | - |
| 16 | 5 | 6 | 2 |
| 17 | 6 | 5 | 4 |
| 18 | 4 | 2 | 1 |
| 19 | 12 | 2 | 5 |
| 20 | 8 | 6 | - |
| 21 | 6 | - | - |
| 22 | 6 | 8 | - |
| **Total:** | **120** | **61** | **37** |

Tabela XX – Quantidade de falsos positivos dos algoritmos testados

Apesar de apresentar um número superior de acertos, como visto no tópico anterior, o Face Recognition também obteve um maior número de falsos positivos. Isto significa que ao buscar por uma pessoa, ele vai trazer outras pessoas diferentes do indivíduo sendo buscado. A causa disso está diretamente ligada ao valor do *threshold* de cada algoritmo.

Após calcular o resultado da distância Euclidiana dos rostos, é feita uma comparação com *threshold.* Se o valor resultado do cálculo feito for menor ou igual ao *threshold,* há grandes chances de serem o mesmo indivíduo. Portanto, quando o número de falsos positivos é alto, quer dizer que os resultados dos cálculos feitos estão dentro do intervalo de valor que classifica aquele rosto como verdadeiro, por isso, existe a necessidade de ajuste do valor de tolerância.

Entretanto, o valor do *threshold* utilizado para os testes foi escolhido, pois, o autor da biblioteca Face Recognition, em seus testes, o determinou como o parâmetro que obteve o melhor resultado. Quando levado em consideração, este fator demonstra que apesar da capacidade de reconhecimento ser melhor que a do OpenFace, é possível que o algoritmo, quando em comparação, tenha dificuldade reproduzir resultados mínimos.

Por causa de retornos irrelevantes em uma pesquisa, ajustar a quantidade de falsos positivos é bastante importante. É ideal que ele seja sempre o menor possível, sem comprometer o número de acertos.

**Falsos Negativos**

Os falsos negativos são as imagens de um indivíduo as quais o algoritmo falhou em reconhecer, ou seja, quando duas fotos comparadas são de uma mesma pessoa, mas o algoritmo diz que são pessoas diferentes.

* Indivíduo – Número identificador do indivíduo no banco de dados
* Qtde. Imagens – Quantidade de imagens que aquele indivíduo possui na base de dados
* Face Recognition – Número de falsos negativos obtidos no teste com a biblioteca Face Recognition
* OpenFace – Número de falsos positivos negativos no teste com a biblioteca OpenFace

|  |  |  |  |
| --- | --- | --- | --- |
| **Indivíduo** | **Qtde Imagens** | **Face Recognition** | **OpenFace** |
| 1 | 7 | - | 2 |
| 2 | 4 | - | - |
| 3 | 4 | - | - |
| 4 | 4 | - | 1 |
| 5 | 4 | - | 1 |
| 6 | 5 | - | 1 |
| 7 | 3 | - | 1 |
| 8 | 4 | - | - |
| 9 | 4 | - | - |
| 10 | 4 | - | - |
| 11 | 5 | - | 1 |
| 12 | 9 | - | 2 |
| 13 | 6 | - | 2 |
| 14 | 6 | - | 1 |
| 15 | 4 | - | - |
| 16 | 5 | - | - |
| 17 | 6 | - | - |
| 18 | 4 | - | - |
| 19 | 12 | - | 11 |
| 20 | 8 | 1 | 2 |
| 21 | 6 | - | 2 |
| 22 | 6 | - | 4 |
| **Total:** | **120** | **1** | **31** |

Tabela XX – Quantidade de imagens que o algoritmo não foi capaz de reconhecer corretamente

Os falsos negativos são importantes para medir a quantidade de imagens que não serão reconhecidas e retornadas em uma busca. Vendo a tabela XX é possível perceber que existe uma diferença de reconhecimento entre ambos os algoritmos.

Isso pode significar que o *threshold* do OpenFace precisa ser ajustado, enquanto o valor utilizado pelo Face Recognition já está no que seria considerado um “ponto ótimo”, ou seja, onde o número de reconhecimento é alto e o de falsos negativos, baixo.

Contudo, ao fazer uma análise dos resultados apresentados na tabela XX em uma comparação com os valores de falsos negativos ilustrados neste tópico, percebe-se que existe uma enorme quantidade de rostos reconhecidos de maneira errada nos resultado, o que contrabalanceia a ideia de que o valor de tolerância usado para classificação está bom. Portanto, mesmo que o algoritmo atualmente pareça com o *threshold* em um “ponto ótimo”, trata-se de uma falsa impressão.

**Tempo**

A métrica de tempo é bastante importante quando se trata de reconhecimento facial aplicado a um *software*, é preciso saber quanto tempo leva para percorrer a base de dados inteira, reconhecer um rosto ou levantar um erro. A tabela XX traz os resultados de tempo médio aproximado de ambos os testes para cada categoria.

* Face Recognition – Tempo utilizado em média para apurar cada categoria dos testes com a biblioteca Face Recognition.
* OpenFace – Tempo utilizado em média para apurar cada categoria dos testes com a biblioteca OpenFace.

|  |  |  |
| --- | --- | --- |
|  | **Face  Recognition** | **OpenFace** |
| **Acertos** | 00:00:01 | 00:00:01 |
| **Falsos Positivos** | 00:00:01 | 00:00:01 |
| **Falsos negativos** | 00:00:01 | 00:00:01 |
| **Erros** | 00:00,8 | 00:00,9 |
| **Percorrer a base** | 0:01:58 | 0:02:10 |
| **Tempo total  de teste** | 00:43:00 | 00:48:00 |

Tabela XX – Tabela de medição de tempo dos algoritmos

A diferença de rapidez entre os dois algoritmos é bastante pequena. Levando-se em consideração que a diferença de tempo, que uma foto de 1 (um) indivíduo demora para percorrer e classificar as 120 (cento e vinte) imagens da base, é de apenas 12 (doze) segundos ao comparar os dois algoritmos, pode-se dizer que ao utilizar bases pequenas, dificilmente o usuário final sentirá diferença de entrega de resultado.

**Performance**

Também foi feita uma análise performática do algoritmo, levando em conta dois dos atributos avaliados: acertos e falsos positivos. A precisão do algoritmo é medida pela quantidade de acertos sobre a quantidade de imagens reconhecidas para aquele indivíduo, ou seja, a soma entre acertos, falsos positivos e falsos negativos. A equação XX representa a fórmula da precisão.

Equação XX – Fórmula usada para o cálculo de precisão

P – Precisão

a – Quantidade de reconhecimentos corretos

QR – Quantidade de reconhecimentos feitos, incluindo acertos, falsos positivos e falsos negativos.

A performance foi medida para cada indivíduo em ambos os algoritmos afim de fazer uma análise comparativa.

A Tabela XX contém as porcentagens de acertos para cada pessoa estudada:

* Indivíduo – Identificação do indivíduo na base de dados
* Face Recognition – Porcentagem de acertos dos testes feitos utilizando a biblioteca Face Recognition
* OpenFace – Porcentagem de acertos dos testes feitos utilizando a biblioteca OpenFace

|  |  |  |
| --- | --- | --- |
| **Indivíduo** | **Face Recognition** | **OpenFace** |
| **Precisão** | |
| 1 | 0,538462 | 0,714286 |
| 2 | 0,5 | 1 |
| 3 | 1 | 0,4 |
| 4 | 0,8 | 0,6 |
| 5 | 1 | 0,2 |
| 6 | 0,363636 | 0,6 |
| 7 | 0,375 | 0,25 |
| 8 | 1 | 0,6 |
| 9 | 0,75 | 1 |
| 10 | 0,6 | 0,75 |
| 11 | 0,625 | 0,5 |
| 12 | 1 | 0,75 |
| 13 | 0,833333 | 0,5 |
| 14 | 1 | 0,666667 |
| 15 | 0,285714 | 1 |
| 16 | 0,333333 | 0,6 |
| 17 | 0,375 | 0,428571 |
| 18 | 0,5 | 0,666667 |
| 19 | 0,857143 | 0,058824 |
| 20 | 0,416667 | 0,666667 |
| 21 | 1 | 0,5 |
| 22 | 0,428571 | 0,333333 |
| **Média** | **0,662812** | **0,581137** |

Tabela XX – Resultados de precisão de cada algoritmo

Em uma primeira análise na tabela XX, o primeiro dado importante é a média de precisão dos algoritmos. Essa média é o resultado geral da performance de cada um é um dos principais fatores de avaliação na hora de decidir qual algoritmo deve ser utilizado. Entretanto, não devem ser esquecidas as análises individuais de cada atributo para fazer uma melhor avaliação.

Trazendo para esta análise o que foi avaliado anteriormente, nota-se que o grande número de falsos negativos trouxe a média de análise do OpenFace para baixo. Do mesmo modo que, com o Face Recognition, os falsos positivos tiveram o mesmo efeito.

**Erros**

Os erros acontecem quando o algoritmo não é capaz de detectar um rosto na imagem, isso é causado por algum tipo de oclusão. Nenhum dos algoritmos testados tinham modelos voltados para a oclusão, então era esperado que alguns tipos fossem causar problemas em detecção de rostos.

* Indivíduo – Identificação do indivíduo na base de dados
* Face Recognition – Número sequencial que identifica qual foto daquele indivíduo está sendo comparado.
* OpenFace – Número sequencial que identifica qual foto daquele indivíduo está sendo comparado.

|  |  |  |
| --- | --- | --- |
| **Indivíduo** | **Face Recognition** | **OpenFace** |
| 2 | 2 | 2 |
| 3 | 3 |
| 4 | 4 |
| 3 | 2 | 2 |
| 4 | 4 |
| 5 | 3 | 3 |
| 6 | 3 | 3 |
| 8 | 4 | 4 |
| 9 | 4 | 4 |
| 10 | 4 | 4 |
| 12 | 3 | 3 |
| 13 | 5 | 5 |
| 14 | 6 | 6 |
| 15 | 2 | 2 |
| 4 | 4 |
| 16 | 3 | 3 |
| 5 | 5 |
| 17 | 4 | 4 |
| 5 | 5 |
| 6 | 6 |
| 18 | 2 | 2 |
| 4 | 4 |
| 20 | 5 | 5 |
| 7 | 7 |
| 21 | 4 | 4 |
| 6 | 6 |
| **Total** | **26** | **26** |

Tabela XX – Imagens com problemas por indivíduo

Por utilizarem o mesmo modelo, não é nenhuma surpresa que os erros encontrados sejam os mesmo em ambos os algoritmos. Porém, cabe uma análise sobre quais tipos de erro foram mais proeminentes durante os testes.

* Tipos de oclusão – Todos os tipos de oclusão encontrados na base de dados
* Quantidade – Quantidade de ocorrências de cada erro.

|  |  |
| --- | --- |
| **Tipos de oclusão** | **Quantidade** |
| Chapéu | 2 |
| Máscara | 11 |
| Mãos | 4 |
| Ângulo | 1 |
| Cabelo | 1 |
| Careta | 0 |
| Óculos | 0 |
| Objetos | 0 |
| 2 ou mais tipos | 7 |
| **Total** | **26** |

Tabela XX – Tipos de oclusão e quais erros foram encontrados

A análise foi feita separando as oclusões individualmente, ou seja, se um indivíduo possuísse mais de um tipo de cobertura, como chapéu e máscara, por exemplo, ele não estaria incluso na categoria chapéu e máscara individualmente, mas na categoria ‘2 ou mais tipos’.

Conforme visto na tabela XX a maior ocorrência de erros veio de usuários utilizando máscaras. Esse tipo de oclusão possui cobertura praticamente completa da parte inferior do rosto, cobrindo a boca, bochechas, nariz e parte do queixo, como visto na figura XX.

Figura XX – Rostos cobertos por máscara que não foram reconhecidos

Observando cada erro pela sua quantidade individual, o segundo maior tipo foi causado pela oclusão utilizando as mãos, cobrindo completamente, ou parcialmente, a lateral do rosto.

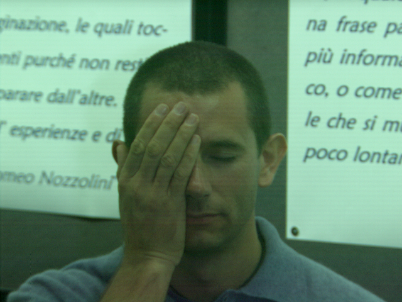
  

Figura XX – Erros encontrados com oclusão de mão

Observando a Figura XX percebe-se que não é necessário que haja cobertura completa da lateral do rosto para que haja problemas de reconhecimento. A adição de um elemento “estranho” confunde o algoritmo, fazendo-o reconhecer a mão como se fosse um elemento que pertencesse naturalmente àquela posição. Ao comparar com o modelo, ele não verá semelhanças, uma vez que para ele se tratam objetos diferentes, como dois moldes que não se encaixam.

Os outros tipos de erro possuem uma incidência muito pequena ou são inconclusivos, uma vez que é difícil compreender as razões que causaram a oclusão. As imagens com os erros restantes estarão no Anexo Y.

Observando o conjunto de resultados obtidos nos testes, que a performance dos algoritmos não possui uma diferença muito grande entre elas. Em uma análise ideal, deseja-se que haja uma maior taxa de certos, poucos ou nenhum resultado falso positivo, falso negativo e/ou erros. As tabelas completas de cada algoritmo estão disponíveis no Anexo Y, para fins de verificação. Na próxima seção, serão feitas as considerações finais em cima dos resultados.

1. **Conclusão**

A diferença de ambos não foi tão grande quanto o esperado. É possível que a falta de calibragem do *threshold* do OpenFace tenha causado algum impacto nos números.

No código demonstrativo, no qual os testes do OpenFace feitos neste trabalho foram baseados, o valor determinado como limiar foi 0.99. Em seus casos de teste, divulgados de maneira ilustrativa, a maior distância com sucesso encontrada pelos casos de teste apresentados no *website* foi superior ao limitante de 0.6 determinado para este trabalho. Neste caso, com o valor de *threshold* utilizado, este trabalho não seria capaz de reconhecer um dos casos de teste apresentados.

Considerando estes resultados, é esperado que haja uma quantidade de falsos negativos bastante alta, como de fato aconteceu.

O *Face Recognition* possui a melhor performance nos valores testados, podendo se tornar mais restrita se diminuir o valor de tolerância. Isso poderia reduzir o número de falsos positivos e aumentar a média de performance do algoritmo. Entretanto, também poderia causa a diminuição da quantidade de rostos reconhecidos e aumentar a quantidade de falsos negativos.

Considera-se que este trabalho possui a intenção de testar algoritmos para serem implementados como mecanismos de busca. Portanto, a quantidade de falsos negativos possui uma relevância maior em análise, pois, significa que, em uma busca, aquele rosto não seria reconhecido como o indivíduo procurado.

Uma pequena parcela de falsos positivos é ideal, mas não é crítica se os números retornados forem baixos. Eles significam que apesar de retornar o rosto que você está procurando, ele também trará de pessoas diferentes, as quais ele considerou que fossem a mesma procurada. Os testes feitos mostram que, o maior indivíduo com falsos positivos, entre os dois algoritmos, possui apenas 8 (oito) reconhecimentos errados, enquanto todas as suas imagens foram reconhecidas corretamente. Assumindo que apenas uma das imagens corretas retornará em uma busca, o resultado será de 9 (nove) rostos, o que é considerado um número baixo.

Por utilizarem o mesmo modelo, não foi uma surpresa que houvesse uma quantidade igual de erros, ou até mesmo erros parecidos.

Portanto, por possuir um número maior de acertos e menor de falsos negativos, o *Face Recognition* foi considerado o melhor dentre os dois algoritmos testados, nas condições apresentadas.

Entretanto, para melhorar a performance, o ideal seria a criação de um modelo capaz de reconhecer rostos com oclusão para evitar a repetição dos erros encontrados

**REFERÊNCIAS**

[Gary B. Huang](http://vis-www.cs.umass.edu/~gbhuang), Manu Ramesh, [Tamara Berg](http://research.yahoo.com/bouncer_user/83), and [Erik Learned-Miller](http://www.cs.umass.edu/~elm).  
**Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments.**  
*University of Massachusetts, Amherst, Technical Report 07-49*, October, 2007.

A. Ponti, M. and B. Paranhos da Costa, G. (2017). Como funciona o Deep Learning. In: V. Vieira, H. L. Razente and M. N. Barioni, ed., *TÓPICOS EM GERENCIAMENTO DE DADOS E INFORMAÇÕES 2017*, 1st ed. Uberlândia: Sociedade Brasileira de Computação – SBC, pp.63-88.

LECUN, Y.; KAVUKCUOGLU, K.; FARABET, C. Convolutional networks and applications in vision. In: Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on. [S.l.: s.n.], 2010. p. 253–256.

DEFREITAS JURASZEK, G. (2014). *Reconhecimento De Produtos Por Imagem Utilizando Palavras Visuais E Redes Neurais Convolucionais*. Mestre. Universidade Do Estado De Santa Catarina – UDESC.

*Davis E. King. Dlib-ml: A machine learning toolkit. JMLR, 10, 2009.*

De Lemos Ferreira Casimiro da Costa, B. (2018). *Practical Face Recognition: Building a Complete System Able to Identify Subjects in Real Time*. Mestre. Universidade do Porto.

B. Amos, B. Ludwiczuk, M. Satyanarayanan, "Openface: A general-purpose face recognition library with mobile applications," CMU-CS-16-118, CMU School of Computer Science, Tech. Rep., 2016.

A. Colombo, C. Cusano, and R. Schettini, “UMB-DB: A Database of Partially Occluded 3D Faces,” in in Proc. ICCV 2011 Workshops, pp. 2113-2119, 2011.