INSTITUTO DE EDUCAÇÃO SUPERIOR DA PARAÍBA
BACHARELADO EM SISTEMAS DE INFORMAÇÃO
LUIZ AUGUSTO FERREIRA MONTEIRO

A Inteligência Artificial no Ambiente Acadêmico: Uso do Reconhecimento Facial para a verificação de presença do discente

CABEDELO
2019

LUIZ AUGUSTO FERREIRA MONTEIRO

A Inteligência Artificial no Ambiente Acadêmico: Uso do Reconhecimento Facial para a verificação de presença do discente

Monografia apresentada ao Curso de Sistemas de informação Instituto de Educação Superior da Paraíba – IESP como requisito para obtenção do título de bacharel em Sistema de Informação.

ORIENTADOR: Prof. Me. Danilo Rangel Arruda Leite

CABEDELO
2019

LUIZ AUGUSTO FERREIRA MONTEIRO

A Inteligência Artificial no Ambiente Acadêmico: Uso do Reconhecimento Facial para a verificação de presença do discente

Monografia apresentada ao Curso de Sistemas de informação Instituto de Educação Superior da Paraíba – IESP como requisito para obtenção do título de bacharel em Sistema de Informação.

Aprovada em: _____de_____de 2019.

BANCA EXAMINADORA

_____

Prof. Me. Hercilio de Medeiros Sousa (orientador)
Instituto de Educação Superior da Paraíba

_____

Prof. XXXXXX
Instituto de Educação Superior da Paraíba

_____

Prof. XXXXXX
Instituto de Educação Superior da Paraíba

AGRADECIMENTOS

À IESP - Instituto de Educação Superior da Paraíba -, pela oportunidade de fazer o curso de Sistemas de Informação e proporcionar o melhor ambiente e corpo docente para o aprimoramento de meus conhecimentos.

Agradeço a todos os professores pela dedicação e tempo desprendido para me manter focado e por me guiarem diversas vezes, e não somente por terem me ensinado, mas por terem me feito aprender. A palavra mestre, nunca fará justiça aos professores dedicados aos quais sem nominar terão os meus eternos agradecimentos.

Ao meu orientador, pelo empenho dedicado à elaboração deste trabalho. Suas ideias e dicas foram de extrema de importância.

Agradeço a minha mãe Maria Júlia, heroína que me deu apoio e incentivo nas horas difíceis, de desânimo e cansaço. Sempre me mostrando que o conhecimento é poder e sempre devemos nos aprimorar.

À minha namorada, Bárbara Elis, que ao longo deste trabalho sempre esteve ao meu lado, me ajudando com palavras de apoio e sem sua ajuda não teria conseguido.

A todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

RESUMO

Com uma abrangência mundial, o reconhecimento facial é utilizado em diferentes países por diferentes formas, e sempre visando à melhoria de vida da população. O presente estudo tem como objetivo buscar a facilitação dos discentes universitários ao acesso de determinadas áreas dentro do campus e, além disso, a substituição de registro de presença em sala de aula através da implantação de um sistema de reconhecimento facial. Sendo este objeto de estudo na área de reconhecimentos de padrões da inteligência artificial.

Palavras-chave: Inteligência Artificial. OpenCV. Python. Reconhecimento Facial. Detecção de Faces.

ABSTRACT

With a worldwide coverage, facial recognition is used in different countries in different ways, always aiming to improve the population's life. The present study aims to facilitate the facilitation of university students to access certain areas within the campus and, in addition, the substitution of classroom presence registration through the implementation of a facial recognition system. Being this object of study in the area of pattern recognition of artificial intelligence.
Keywords: Artificial intelligence. OpenCV. Python. Facial Recognition. Face Detection.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

INTRODUÇÃO

Atualmente a inteligência artificial está avançando a passos largos e diversas áreas do conhecimento humano encontram-se em processo de substituição por sistemas automáticos e artificiais. Desde o começo dos estudos sobre a psique humana nos preparamos para um processo de evolução que, em sua época, não se imaginava (SILVEIRA, p. 8-23, 2018)

A inteligência artificial é dividida em várias áreas, cada uma com sua característica e complexidade única, porém um único objetivo: facilitar a interação humana com máquinas (MACHADO, 2019).

Nesse projeto, iremos trabalhar com a área de reconhecimento facial usando o OpenCV (https://opencv.org/) e a linguagem Python (https://www.python.org/) para a melhoria do sistema acadêmico universitário de presença dos discentes em sala de aula.

O reconhecimento facial é autoexplicativo, é a área responsável pela identificação de pessoas através da captura de seus rostos usando dispositivos que estejam disponíveis, podendo ser: webcam, câmera de celular, circuito interno de uma determinada instituição, entre outros. Esse processo pode ser feito através de um banco de dados, buscando identificar por fotos ou usando um sistema que reconhece o indivíduo capturado pela câmera em utilização.

Pensando na melhoria do sistema acadêmico, o reconhecimento facial poderia ser utilizado para otimizar o tempo dos professores. Dessa forma, ao invés de ter uma chamada para uma sala lotada, o reconhecimento facial faria todo o trabalho; identificando alunos e passando a informação para o sistema acadêmico se o aluno está presente ou não.

Porém, a implementação de um sistema como esse apresentaria diversos problemas por ser algo complexo. Citando alguns dos problemas enfrentados, o hardware necessário está comprovando ser o maior problema. Pois precisaríamos de um computador bom suficiente para rodar o sistema, além da disponibilidade de uma câmera. Nesse projeto, foi utilizado um *Notebook* com as seguintes configurações: *Intel(R) Core (TM)* i5-4210U *CPU* de 1.70GHz, memória *RAM* de 8*GB* e um HD SATA de 1*TB*. É uma máquina com um poder de processamento razoável. Entretanto, há uma demora na resposta da *IDE* utilizada e, também, na execução de um código simples de detecção de faces e reconhecimento facial.

Nesse trabalho, nos empenharemos em demonstrar o funcionamento dessa ferramenta e sua importância através de um estudo mais aprofundado referente ao reconhecimento facial, com a finalidade de mudar o atual sistema de presença dos discentes.

1. OBJETIVOS

1.1. GERAL

Desenvolver um sistema que realize a verificação da frequência do aluno por meio de reconhecimento facial.

1.2 ESPECIFICOS

- Conceituar e analisar historicamente a Inteligência Artificial
- Compreender o Reconhecimento Facial
- Apontar quais ferramentas serão utilizadas
- Conhecer e analisar o uso do OpenCV
- Conceituar e analisar o uso do Python
- Desenvolver um sistema de Reconhecimento Facial
- Testar os componentes do sistema

2. Inteligência Natural

O estudo da inteligência artificial vem sendo utilizado há muito tempo, desde Aristóteles e sua teoria sobre o conjunto de leis que governam o pensamento, e ainda se considerarmos a construção de instrumentos como o ábaco e calculadora mecânica, datadas do século I. Porém, somente em 1956 temos a terminologia "inteligência artificial" sendo usada no meio acadêmico. (ARTERO, 2009).

Atualmente ainda existe uma grande diferença entre a inteligência artificial e a natural, ou seja, a nossa inteligência. A complexidade humana é uma barreira a ser superada em relação a esse campo, a fim de se ter uma melhor eficiência na construção de uma máquina autônoma. Para entendermos melhor sobre essa área devemos compreender como nossa inteligência funciona. Em seu conceito mais básico é a habilidade humana de aprender novos conhecimentos e aplicá-las de forma satisfatória ou, em alguns casos, de forma extraordinária.

> Quando falamos do intelecto ou, melhor, dos poderes intelectuais e desempenho de pessoas, estamos nos referindo principalmente a essa classe especial de operações que constituem a teorização. O objetivo dessas operações é o conhecimento de proposições ou fatos verdadeiros. Matemática e algumas ciências naturais estabelecidas são as realizações modelo dos intelectos humanos. Os primeiros teóricos naturalmente especularam sobre o que constituía as excelências peculiares das ciências e disciplinas teóricas, cujo crescimento eles testemunharam e ajudaram. Eles estavam predispostos a descobrir de que estava na capacidade de uma teoria rigorosa que coloca a superioridade dos homens sobre os animais, dos homens civilizados sobre os bárbaros e até da mente divina sobre mentes humanas. Eles legaram a ideia de que a capacidade de atingir o conhecimento das verdades era a propriedade definidora de uma mente. Os poderes de outro humano poderiam ser classificados como mentais apenas se pudessem ser demonstrados de alguma maneira ser pilotados pela compreensão intelectual de proposições verdadeiras. Ser racional era ser capaz de reconhecer verdades e as conexões entre elas. (RYLE, 2009, p. 15)

Dessa forma, Ryle (2009) relaciona o poder intelectual humano como forma de soberania sobre outras formas de vida. Esse conjunto de poderes intelectuais define a mente humana e, de certa forma, controla as ações de cada indivíduo, mas de forma única para cada um.

A mente humana é algo muito complexo, podendo, somente a parte responsável pelo armazenamento de memórias, chegar a uma quantidade enorme de *Petabytes* – equivalente a 1024 *Terabytes* (HADHAZY, 2015). Em uma rápida análise, percebemos o quanto seria difícil de criar, perfeitamente, todas as funcionalidades da mente humana. Pois além de tudo isso, ainda existe os sentimentos, movimentos, expressões e vários outros detalhes que nos torna humanos.

Em 1983, surgiu uma nova teoria sobre a inteligência humana, revolucionando a forma de compreensão de nossa mente. Nesse ano, Howard Gardner (1995), escreveu um livro sobre a teoria das inteligências múltiplas, chamado *Frames of Mind* (Estruturas da mente). Nele podemos perceber a grande diferença entre as teorias e da capacidade de processamento de nossa mente.

> (...) a teoria das inteligências múltiplas diverge dos pontos de vista tradicionais. Numa visão tradicional, a inteligência é definida operacionalmente como a capacidade de responder a itens em testes de inteligência. A inferência a partir dos resultados de testes, de alguma capacidade subjacente, é apoiada por técnicas estatísticas que comparam respostas de sujeitos em diferentes idades; a aparente correlação desses resultados de testes através das idades e através de diferentes testes corrobora a noção de que a faculdade geral da inteligência, g, não muda muito com a idade ou com treinamento ou experiência. Ela é um atributo ou faculdade inata do indivíduo. A teoria das inteligências múltiplas, por outro lado, pluraliza o conceito tradicional. Uma inteligência implica na capacidade de resolver problemas ou elaborar produtos que são importantes num determinado ambiente ou comunidade cultural. A capacidade de resolver problemas permite à pessoa abordar uma situação em que um objetivo deve ser atingido e localizar a rota adequada para esse objetivo. A criação de um produto cultural é crucial nessa função, na medida em que captura e transmite o conhecimento ou expressa as opiniões ou os sentimentos da pessoa. Os problemas a serem resolvidos variam desde teorias científicas até composições musicais para campanhas políticas de sucesso. (GARDNER, 1995, p.21)

Ainda hoje não existe um consenso que defina a inteligência humana. Ao passar dos anos foram feitos diversos simpósios a fim de definir, porém sempre terminava em indecisões. De acordo com a teoria de Gardner, na inteligência artificial poderia ser aplicada de forma que a máquina execute ações especificas de acordo com o algoritmo produzido. Essa seria a maior diferença entre humanos e máquinas, enquanto um é multitarefas o outro é um especialista. A inteligência artificial surgiu como uma necessidade para ajudar os humanos a melhor compreender informações e utilizá-las da melhor forma possível. Muito se ouve falar sobre o papel antagônico da inteligência artificial na vida profissional das pessoas, todavia temos potencial de analisar e determinar a capacidade dela de ajudar e não substituir a função do humano perante a sociedade.

3. Inteligência Artificial

Em 1950, Alan Turing – conhecido por ser o pai da computação – escreveu um livro na qual falava sobre a predisposição de uma máquina em torna-se inteligente. Para tanto, ele produziu um teste no intuito de afirmar se determinada máquina é inteligente ou não. O Teste de Turing, como foi chamado, consistia de uma série de perguntas e a interação da máquina

com o usuário. Se o mesmo não percebesse que estava interagindo com uma máquina, o teste determinava a máquina como sendo inteligente. E para passar nesse teste a máquina deve cumprir alguns requisitos, como por exemplo, o reconhecimento da linguagem natural. (SERRANO, 2012)

Em 1956 ocorreu em Hanover, cidade dos Estados Unidos, o *Dartmouth Summer Research Project*, que consistia no primeiro evento para a discussão e estudo da inteligência artificial. Localizado em uma universidade de mesmo nome, *Dartmouth College,* esse evento teve início com uma proposta feita por John McCarthy, Marvin Minsky, Nathaniel Rochester e Claude Shannon, datada de um ano antes de ser realizado. Os objetivos dos autores era criar um grupo de estudo para aprender, e sobretudo entender, como um computador pode tornar-se inteligente. Dessa forma programando a máquina para se comportar e até realizar ações comumente feitas pelos humanos. (MCCARTHY, 1955)

Na proposta do *Dartmouth Summer Research Project* existem vários tópicos sobre os assuntos a serem tratados, como: computadores automáticos, como um computador pode ser programado pra usar uma linguagem, redes neurais, teoria do tamanho do cálculo, auto aperfeiçoamento, abstrações e aleatoriedade e criatividade. Cada um deles fala especificamente como os autores pensavam como iria se suceder no futuro e isso foi um grande passo na inteligência artificial, tendo em vista que alguns desses tópicos se transformaram em uma subárea. ((MCCARTHY, 1955)

> Examinando a proposta do seminário de Dartmouth (McCarthy et al., 1955), podem os ver porque era necessário que a IA se tornasse um campo separado. Porque todo o trabalho feito na IA não podia ficar sob o nome de teoria de controle, pesquisa operacional ou teoria da decisão que, afinal de contas, têm objetivos semelhantes aos da IA? Ou, então, porque a IA não poderia ser um ramo da matemática? Primeiro, porque a IA abraçou desde o início a ideia de reproduzir faculdades humanas como criatividade, auto aperfeiçoamento e uso da linguagem, e nenhum dos outros campos tratava dessas questões. A segunda resposta é a metodologia. A IA é o único desses campos que claramente é um ramo da ciência da computação (embora a pesquisa operacional compartilhe uma ênfase em simulações por computador), e a IA é o único campo a tentar construir máquinas que funcionarão de forma autônoma em ambientes complexos e mutáveis. (RUSSEL, NORVIG, 2013, p.21)

A inteligência artificial pode ser datada de muito cedo, praticamente no início da era da informática, e durante essa época houve pouquíssima aplicação em relação a essa área, construindo-se apenas máquinas básicas. A limitação tecnologia era um impedimento para o seu grande potencial. Porém, os cientistas não desanimaram e durante os anos 60 surgiram

várias outras teorias, como a linguagem natural, a teoria de qualidade total de Feigenbaum, entre outros. E, atualmente, a inteligência artificial vem sendo estudada e melhor compreendida. Agora focando em problemas mais reais e concretos, como na indústria. (GALIPIENSO, 2003)

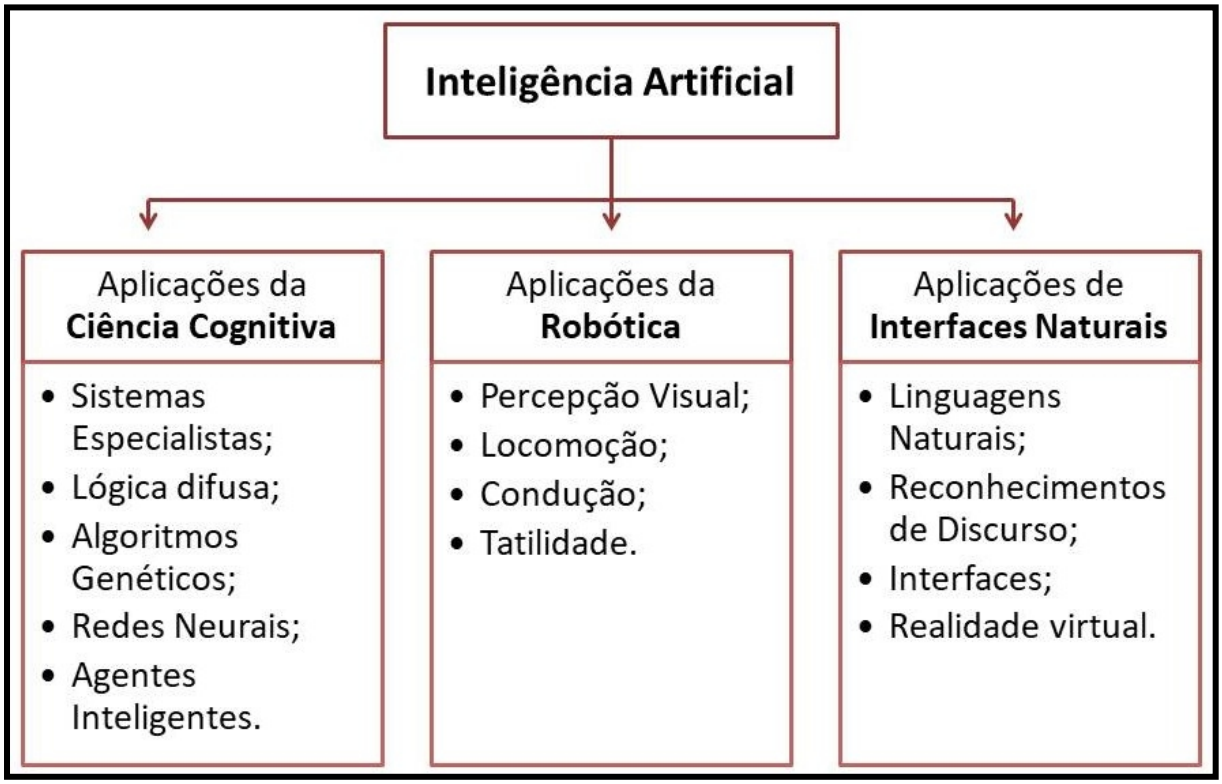
Nós podemos ver o avanço da inteligência artificial de forma mais nítida atualmente, o maior exemplo disso é o robô criado em 2016 pelo norte-americano David Hanson, chamada Sophia. Ela tem uma grande capacidade de processamento, e ainda usa a combinação de algumas áreas dentro da inteligência artificial, como o reconhecimento facial e de voz. Quando apareceu pela primeira vez na mídia, Sophia causou grande surpresa e admiração. Em vários âmbitos a inteligência artificial supera a capacidade humana e um grande exemplo disso foi a vitória da inteligência artificial do Google sobre um campeão de xadrez, apenas depois de quatro horas de treinamento. (GIBBS, 2017)

Muitas vezes podemos nos enganar excogitando a ideia da inteligência artificial se resumir a uma máquina simulando um ser humano, onde em quase todos os casos imaginamos um robô igual à Sophia. Entretanto, ideias como essa limita muito a inteligência artificial. Hoje ela é usada como uma ferramenta em várias empresas e de formas muito diferente do que vemos em Sophia. Alguns exemplos da utilização da inteligência artificial na qual se sobressaem são: o reconhecimento facial para detecção de pessoas perdidas (BASILIO, 2019), rastrear os movimentos migratórios de pássaros (CECCON, 2019), rede neural usando eletrocardiograma para fazer predições de problemas de saúde (CECCON, 2019), dentro tantos outros. A cada dia surgem ainda mais aplicações para essa área em ascensão.

Observamos então, o vasto mundo no qual a inteligência artificial tem a nos oferecer. E com isso, possamos ser capazes de evoluir nosso estilo de vida a ponto de estarmos conectados, convivendo e usufruindo o melhor que a inteligência tem a oferecer. Porém, devemos ser cientes de que nem todas as áreas evoluem de maneira similar. Um grande exemplo disso é o progresso da aprendizagem de máquina em relação à robótica. Isso está relacionado diretamente ao poder aquisitivo das empresas e cientistas para evoluir cada uma das áreas. É muito mais fácil programar um sistema automático de apoio de decisão do que construir um robô para uma tarefa de uma determinada área.

Falando sobre essa vastidão de conhecimento que chamamos de inteligência artificial, conseguimos extrair diversas áreas de aplicação e cada uma com sua particularidade, assim como podemos ver na figura 1.

Figura 1: Aplicações da Inteligência Artificial.



FONTE: Próprio Autor (2019)

Tendo em vista a grande quantidade de aplicações, vamos elucidar alguns deles a fim de diferenciar e conhecer mais um pouco sobre o que cada área tem a oferecer.

Os sistemas especialistas podem concluir temas específicos, ainda assim deve ser direcionado e sustentado. Uma melhor forma de definir um sistema especialista é se pensarmos nele como um profissional responsável por uma determinada atividade. Para solucionar os problemas direcionados a estes sistemas, o mesmo precisa acessar um banco de informações relacionado ao domínio da aplicação. Seu funcionamento é simples, primeiro acessa mecanismos de raciocínio para solucionar o problema e logo após isso acessa outro mecanismo para mostrar o usuário qual ação foi feita. (GOMES, 2010)

> No início dos estudos sobre SE, o primeiro sistema a ser considerado foi o DENDRAL, em 1965, no entanto, sua aplicabilidade ficou restrita ao meio acadêmico. Em 1976 desenvolveu-se o SE mais conhecido, o MYCIN que também ficou restrito aos meios acadêmicos. Somente em 1982, com o desenvolvimento do XCON, os SE saíram das Universidades e ganharam o interesse das indústrias. (...) (KHOELER, 1998, p.36)

Segundo Raykin (2008), as Redes Neurais, em um melhor entendimento, se aproximam mais na simulação da mente humana. Sua programação refere-se à construção de uma rede neural, por algoritmos ou através de componentes eletrônicos, para a modelagem de uma determinada atividade ou função requerida. A complexidade dessa área é enorme, pois para ter um embasamento sobre o assunto seria recomendado estudos mais profundos sobre o entendimento da mente humana. Suas principais características são sua estrutura e sua capacidade de generalização (poder de tomar decisões não previamente vista no treinamento).

A robótica é uma das áreas mais abrangentes e usadas da inteligência artificial. Estando presente na maioria - se não em todas - as fabricantes de automóveis, os robôs são parte essencial do melhor aproveitamento e maior agilidade inerente à sua fabricação.

Nessa área podem ser encontrados três tipos de robôs, sendo eles: manipuladores – pode ser definidos como máquinas de braços fixos em uma estação de trabalho -, muito utilizados em montadoras de automóveis, um exemplo dele está na figura 2; robôs móveis com a principal característica de operar com pernas, rodas e outros tipos de mecanismos que facilitam o deslocamento, mostrado na figura 3; e o manipulador móvel, sendo esse a junção dos outros dois tipos, observado na figura 4.

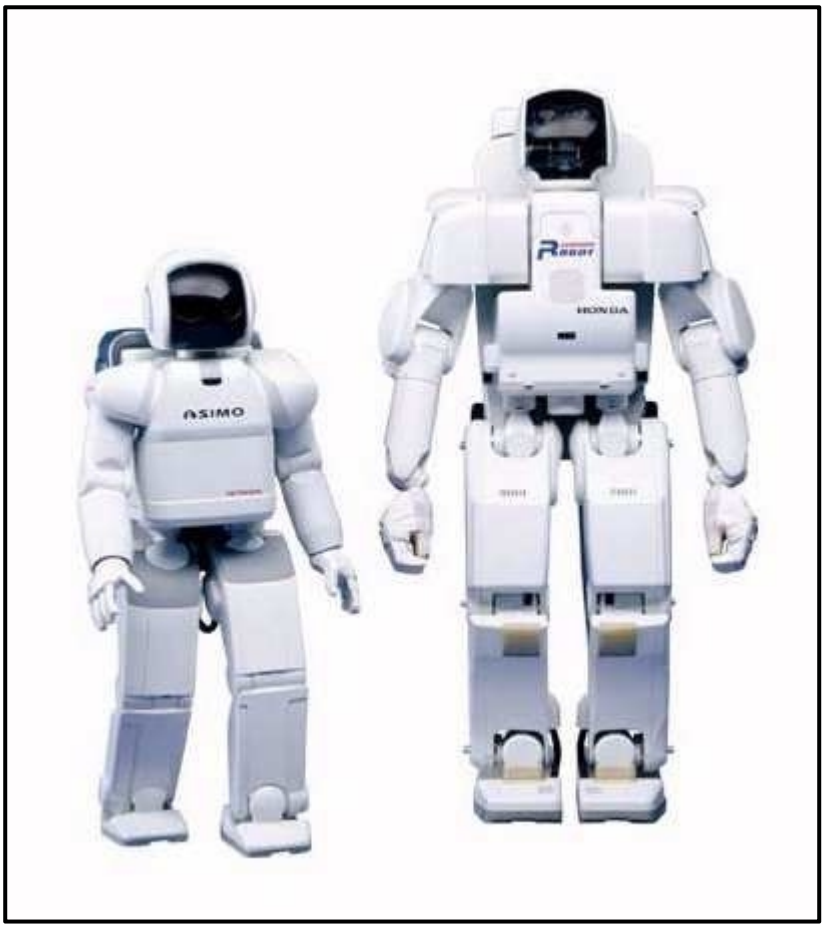
Figura 2: Braço manipulador.

Fonte: Carrara (2019)

Figura 3: Robô Sophia.

Fonte: Alves (2019)

Figura 4: Honda Asimo e P3.



Fonte: Dertien (2005)

A linguagem natural é um processo complexo por si só, onde gramáticos estudam a fundo todo conhecimento obtido através da linguagem falada e escrita. Toda essa complexidade é analisada através da universalidade dentre as várias línguas mundiais, além de suas estruturas e sobre a variação única existente em cada linguagem. (PASSOS, 2013)

De acordo com Passos (2013), o processamento de linguagem natural tenta simular exatamente essas características, transformando formas de linguagem como uma forma de entrada e gerando uma saída a estrutura linguística escolhida pelo usuário. E ainda existem três tipos de processamento: por comando, discreto e contínuo. Cada um com sua característica única e seu uso depende da necessidade do usuário.

E, além de todas essas aplicações da inteligência artificial, temos o reconhecimento de faces. Muito utilizado em câmeras fotográficas e em redes sociais, o reconhecimento facial pode ser utilizado de duas formas: para detecção de imagens estáticas e para o reconhecimento de faces e objetos.

4. Reconhecimento Facial

Para melhor entendermos sobre esse determinado assunto, é imprescindível o entendimento sobre visão computacional. Segundo Coldewey (2016), ela é basicamente uma simulação da visão humana através de circuitos e uma forma lógica. Porém, se analisarmos e uma forma mais detalhada, uma simples ação de pegar uma bola arremessada por outra pessoa, requer um grande processamento por parte de nosso cérebro. Envolvendo várias partes ao mesmo tempo, desde a captura da imagem do objeto arremessado até o cálculo de onde esse objeto cairá, para desse modo, ter-se uma chance de pegá-lo. Sabendo-se disso, notamos a dificuldade em replicar uma simples ação como essa em uma máquina. Por temos pouco conhecimento de como o funcionamento do cérebro humano funciona em detalhes.

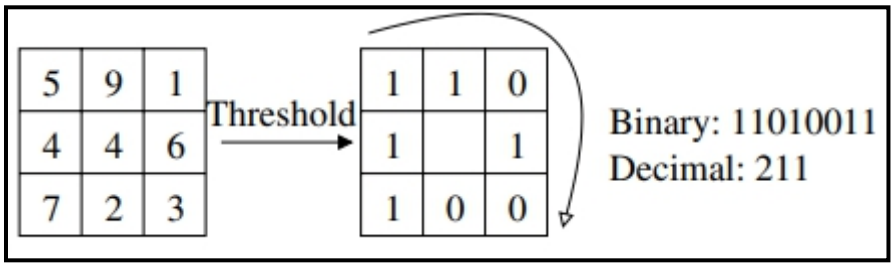
Existem três abordagens básicas para o reconhecimento da face. A primeira é baseada na extração de vetores características de partes básicas de uma face, tais como: olhos, nariz, boca e queixo. Mais conhecida como *Eigenfaces*, essa abordagem utiliza modelos deformáveis e análise matemática para extrair informação das componentes básicas da face e, em seguida, converter essa informação em um vetor característico. Para tal, ela utiliza a Análise de Componentes Principais, onde teoricamente o algoritmo lê uma grande quantidade de dados e retorna características únicas de cada face para diferenciá-los. (SOUZA, 2014)

A segunda abordagem, *Fisherfaces*, é baseada nos conceitos da teoria da informação. Nessa abordagem a informação que melhor descreve uma face é derivada a partir da imagem

da face toda. Utilizando-se da Análise de Discriminante Linear, o algoritmo separa as características únicas de determinadas faces em várias classes, dessa forma o reconhecimento de uma imagem desconhecida seria feito através dessas classes. (SOUZA, 2014)

E, ainda temos o *Local Binary Pattern Operator*. É uma ferramenta poderosa que separa os pixels da imagem e limitam-nos em uma matriz 3x3. Onde cada célula tem um valor pré-determinado pelo algoritmo e, logo após, é transformado em números binários, como vemos na figura 5. E, finalmente, o resultado dessa transformação é vinculado a um "historiograma", assim sendo utilizado como um descritor de textura. (AHONEN, 2006)

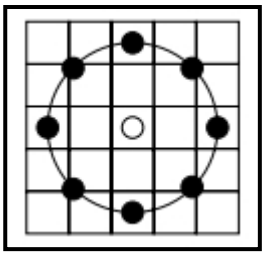
Figura 5: Transformação dos valores de matriz 3x3.



Fonte: Ahonen (2006)

Visando melhorar ainda mais a funcionalidade do *Local Binary Pattern Operato* (figura 6), foi adicionado um maior número de células por pixel. Dessa forma, permitindo quaisquer números de pixel ou valores que cada célula poderia ter. (AHONEN, 2006)

Figura 6: *Local Binary Pattern Operator* multivalorado.



Fonte: Ahonen (2006)

Enfim, podemos um pouco sobre o mundo chamado inteligência artificial e como ela tem ajudado a humanidade aos poucos a ter mais praticidade, agilidade e dentre outras tantas vantagens. Ainda existem diversas formas de aplicação aqui não estudadas, mostrando o quão diverso e profundo cada área pode ser.

4.1 Eigenfaces

O reconhecimento de faces através de códigos computacionais pode ser uma tarefa árdua, caso observarmos essa ação da perspectiva da máquina. Haja visto que para ter esse reconhecimento, devemos passar uma série de instruções em um nível mais alto de abstração e, ainda, transformar tais informações em nível de máquina; dessa forma, conseguindo se aproximar o máximo possível do reconhecimento facial igual à dos humanos. Porém, percebemos a dificuldade na implementação de tal algoritmo, visto que nós humanos falhamos nessa mesma tarefa, pois há entre nós aqueles cujo a visão não é perfeita e ainda aqueles onde possuem gêmeos idênticos.

Porém, obstáculos nunca pararam os cientistas de avançarem mais ao futuro, na verdade é exatamente o contrário. Temos uma predisposição em superar dificuldades e uma capacidade criativa gigantesca. Com isso um algoritmo poderoso foi criado com o intuito de melhor classificar o reconhecimento facial, L. Sirovich e M. Kirby criaram o *Eingenfaces*.

> A presente investigação refere-se às questões dos problemas gerais de caracterizar, identificar e distinguir padrões individuais extraídos de alguma classe bem definida de Padrões. Tanto a intuição quanto a aplicação dos métodos provêm do problema particular da identificação de faces. Por esse motivo, usamos uma terminologia específica para este caso, embora a generalidade da abordagem seja aparente. (...) com base em um método conhecida como expansão de Karhunen-Loeve no reconhecimento de padrões e como análise de fator ou componente principal na literatura estatística. (SIROVICH, KIRBY, 1987, p. 519-524)

O algoritmo apresentado por L. Sirovich e M. Kirby (1987) trata-se do reconhecimento de faces através das melhores coordenadas entre eles, chamados de *eigenpictures.* De uma forma geral, perceberam que uma face pode ser separada em várias escalas de cinza. Podendo, assim, com um mínimo de erro identificar uma face comparando todas as *eigenpictures* com suas escalas de cinza.

Aproveitando esse estudo, Turk e Pentland (1991), fizeram o uso do mesmo para a criação de um algoritmo capaz de reconhecer e detectar faces automaticamente com a ajuda dos computadores. O processo de reconhecimento, descrito por Turk e Pentland (1991), é definido pela identificação da cabeça do indivíduo e comparando as características de outros indivíduos previamente conhecidos.

> Nossa abordagem trata o reconhecimento de face como um problema de reconhecimento bidimensional, aproveitando o fato de que as faces normalmente estão na vertical e, portanto, podem ser descritas por um pequeno conjunto de vistas características 2D. As imagens de rosto são projetadas em um espaço de recurso ("*face space*") que melhor codifica a variação entre as imagens de rosto conhecidas. O espaço da face é definido pelos "*eigenfaces*", que são os *eigenvectors* do conjunto de faces; eles não correspondem necessariamente a características isoladas, como olhos, ouvidos e narizes. A estrutura fornece a capacidade de aprender a reconhecer novos rostos de maneira não supervisionada. (TURK, PENTLAND, 1991. p. 586-591.)

Para um melhor entendimento, o algoritmo separa várias imagens a partir de uma principal, chamado *eigenfaces*; fazendo uma combinação linear com as características principais desse individuo (olhos, nariz, boca e curvatura da cabeça), e transforma essas imagens em uma imagem média.

Nos estudos iniciais de reconhecimento facial, cientistas da área consideravam suficientes os aspectos pré-definidos de rostos conhecidos. No entanto, esse tipo de abordagem não tinha um retorno de reconhecimento aceitável. E a ideia principal na formulação do *eigenfaces* é a captura de diversas imagens, armazená-las em um banco de dados e fazer a captura e comparação de faces conhecidas através de características retiradas a partir das imagens bases. Matematicamente, o *eigenfaces* extrai os principais componentes de uma imagem, dividindo-a em seus *eigenvectors* – um conjunto recursos que juntos caracterizam a variação entre imagens de rosto.

Em modos gerais, podemos separar o reconhecimento de faces em cinco passos essenciais:

1. Inicialização: Adquirir o treinamento do conjunto de faces e calcular as *eigenfaces*; armazenando-as em uma *face space*.
2. Quando uma imagem nova é encontrada, calcula-se um conjunto de peso baseados na imagem de entrada.
3. Determinar se a imagem de entrada é realmente uma face, comparando-a com a *face space* e analisando se é aproximado a ela.
4. Se for uma face, classificar o peso padrão como sendo uma face conhecida ou não.
5. Finalmente, como opção, se uma imagem desconhecida é vista mais de uma vez, calcula-se o peso padrão de suas características e adiciona-a as faces conhecidas.

5. OpenCV

O *OpenCV, Open Source Computer Vision Library,* surgiu de um empreendimento da *Intel Research*, onde fazia-se estudos sobre aplicações avançadas de CPU-intensivas. Um dos autores visitava diversas universidades nos Estados Unidos à procura de tecnológicos que pudesse ser usada. Quando ele conheceu um grupo de alunos do *MIT* que usavam um padrão de infraestrutura de visão computacional. (BRADSKI; KAEHLER, 2008)

Com o pensamento de tornar uma infraestrutura parecida com aquela usada pelos estudantes, o *OpenCV* surgiu. Ele foi idealizado dentro do país, mas a programação dele foi concebida na Rússia, no qual estava situado o *Software Performance Libraries*. (BRADSKI; KAEHLER, 2008)

Vadim Pisarevsky, chefe dos membros russos da *Intel*, foi o maior colaborador para a construção do *OpenCV.* Sendo ele responsável por gerenciar, codificar e otimizar a maior parte da biblioteca. Ainda houve outros responsáveis pela implementação, e tão importantes quanto, como Victor Eruhimov e Valery Kuriakin. (BRADSKI; KAEHLER, 2008)

Durante a implementação existiu três objetivos primordiais que serviam como base da motivação de elaborar uma infraestrutura de visão computacional. Eram elas:

- Uma visão avançada da pesquisa para com isso ter uma infraestrutura aberta e otimizada. Desse jeito, programadores e interessados não precisassem "reinventar a roda".
- Divulgar o conhecimento da visão, assim desenvolvedores teriam fácil acesso a um código legível e transferível.
- Uma visão comercial voltada a uma otimização de código de forma livre

A principal motivação da *Intel* em deixar essa biblioteca gratuita para uso é a necessidade de processadores ainda mais poderosos para ter uma melhor qualidade em relação à utilização do OpenCV. (BRADSKI; KAEHLER, 2008)

Com uma variação enorme de algoritmos otimizados, o OpenCV fornece o suporte para a programação de diversas áreas, seja ela sobre visão computacional clássica ou sobre inteligência artificial em seu estado de arte. Esses algoritmos têm a predisposição de realizar diversos tipos de tarefas, como: detectar e reconhecer rostos, identificar objetos, classificar ações humanas em vídeos, rastrear movimentos da câmera, dentre outros. Podemos perceber que o OpenCV é, basicamente, uma ferramenta de apoio ao programador para tratar imagens, sendo elas estáticas ou não.

Essa biblioteca possui diversos módulos, como:

> O módulo opencv_core que contém as principais funcionalidades da biblioteca, em particular, as estruturas básicas de dados e as funções aritméticas. O módulo opencv_imgproc que contém as principais funções de processamento de imagem. O módulo opencv_highgui que contém a leitura de imagem e vídeo e funções de escrita, juntamente com outras funções da interface do usuário.
> O módulo opencv_features2d que contém os detectores de pontos de recurso e descritores e a estrutura de correspondência de pontos de recurso. O módulo opencv_calib3d que contém a calibração da câmera, geometria *two-view* e funções estéreo.O módulo opencv_video que contém a estimativa de movimento, rastreamento de recursos, e funções e classes de extração de primeiro plano.O módulo opencv_objdetect que contém as funções de detecção de objeto, como os detectores de rosto e pessoas. (LAGANIÈRE, 2014, p.10)

Esses módulos foram implementados desde a versão 2.2 do OpenCV. Atualmente na versão 4.1, temos uma maior variedade de módulos e funcionalidades ainda melhores.

5.1 Haarcascades

Para ter uma melhor fundamentação sobre como o processo de reconhecimento de imagem funciona, vamos começar pelos arquivos base de todo projeto: os Haarcascades.

Arquivos Haarcascade são arquivos *xml* necessários para o reconhecimento de pessoas ou parte de seus corpos – como, por exemplo, os olhos. E ainda objetos que serão capturados pelo dispositivo, como: banana, carros, etc. Porém esses arquivos são muito complexos, sendo difícil encontrar de forma gratuita – muitas empresas vendem esses arquivos -, mas ainda existe aqueles distribuídos de forma gratuita pelo *OpenCV*. Para sua utilização deve-se dar o *import* no projeto desejado, dessa forma, a captura daquilo desejado será feita com a ajuda desses xml.

A figura 7 nos mostra o início de um Haarcascade. Podemos observar o quão complexo é a produção, envolvendo muitos cálculos e diferentes tipos de uma mesma característica. Nesse caso é o Haarcascade de reconhecimento de olhos que vem acompanhado com o *OpenCV*. Para fazer esses arquivos devem-se comparar diversas imagens que podem ser chamadas de imagens negativas e positivas. As negativas estão relacionadas a tudo que não é da característica estudada; como no exemplo é um Haarcascade de olho, as imagens negativas dela seriam de imagens sem olhos. As positivas são exatamente as imagens que possuem olhos.

Figura 7: Linhas de código de um Haarcascade.

```
<opencv_storage>
<cascade type_id="opencv-cascade-classifier"><stageType>BOOST</stageType>
 <featureType>HAAR</featureType>
 <height>20</height>
 <width>20</width>
 <stageParams>
  <maxWeakCount>93</maxWeakCount></stageParams>
 <featureParams>
  <maxCatCount>0</maxCatCount></featureParams>
 <stageNum>24</stageNum>
 <stages>
  <_>
   <maxWeakCount>6</maxWeakCount>
   <stageThreshold>-1.4562760591506958e+00</stageThreshold>
   <weakClassifiers>
    <_>
     <internalNodes>
      0 -1 0 1.2963959574699402e-01</internalNodes>
     <leafValues>
      -7.7304208278656006e-01 6.8350148200988770e-01</leafValues></_>
    <_>
     <internalNodes>
      0 -1 1 -4.6326808631420135e-02</internalNodes>
     <leafValues>
      5.7352751493453979e-01 -4.9097689986228943e-01</leafValues></_>
    <_>
     <internalNodes>
      0 -1 2 -1.6173090785741806e-02</internalNodes>
     <leafValues>
      6.0254341363906860e-01 -3.1610709428787231e-01</leafValues></_>
    <_> (..)
```

Fonte: Próprio Autor (2019)

Deve-se ter uma grande variedade de imagens para se obter um melhor resultado. Pois o algoritmo irá rodar e achar as diferenças em cada uma das imagens. Para isso computadores com um grande processamento são utilizados, caso contrário duraria meses para processar as imagens.

Existem alguns passos para criar um Haarcascade: ter um servidor, baixar os *sources* do *OpenCV*, ter as imagens negativas e positivas e criar arquivos de texto com os caminhos das imagens. Isso seria somente uma ideia de como ser feito esses arquivos. Não vamos nos aprofundar no assunto, pois não é nosso objetivo.

A necessidade de conseguir um servidor é pelo que já foi dito antes. O processo de criação do arquivo é muito complexo e precisa de uma quantidade enorme de processamento. Um arquivo simples em um computador pessoal demoraria meses para processar, em contrapartida o servidor faria o mesmo serviço em semanas.

E os demais passos são todos relacionados a criação do arquivo. *Sources* são baixados diretamente do site do *OpenCV*, nele serão utilizados métodos importantes para esse processo. As imagens podem ser adquiridas em qualquer lugar, contanto que as mesmas sejam referentes ao objeto ou parte do corpo estudado. O último passo é a criação de 2 arquivos de texto com os caminhos das imagens negativas e positivas, nesse caso sendo um arquivo para cada. Utilizando um método chamado *opencv_create_samples* ele irá retornar um arquivo .vec e depois utilizar outro método para fazer o treinamento dessas imagens. O método mais utilizado é o *opencv_traincascade*. Além dele também existe o *opencv_haartraining*, esse é o mais antigo e por isso pouco utilizado. Depois de todo esse processo haverá a criação do xml.

6. Python

Essa linguagem começou a ser desenvolvida no começo dos anos 80 por Guido Von Hossum, onde nessa época programava em C em uma empresa situada na Holanda. Porém, ele percebeu a dificuldade de resolver alguns problemas usando essa linguagem. Então ele idealizou uma linguagem onde a programação fosse fácil e intuitiva, surgindo assim o Python.

Sendo uma das linguagens de programação mais utilizadas no mundo, o *python* também é usada para a codificação de reconhecimento de imagens e faces através da biblioteca *OpenCV*. Utilizaremos essa ferramenta no projeto aqui apresentado. Porém, existe outra também muito conhecida no mundo da inteligência artificial, a linguagem R. Mas como nosso foco não é essa linguagem, nos ateremos em mostrar um pouco mais sobre o *python*.

> Python é uma linguagem de programação poderosa e fácil de aprender. Possui estruturas de dados de alto nível eficientes e uma abordagem simples, porém eficaz, à programação orientada a objetos. A sintaxe elegante do Python e a digitação dinâmica, juntamente com sua natureza interpretada, o torna uma linguagem ideal para scripts e desenvolvimento rápido de aplicativos em muitas áreas na maioria das plataformas. (ÖZMEN, 2019, p. 2)

Özmen (2019) ainda fala sobre suas principais características, dentre elas: o fácil uso, por meio de uma sintaxe elegante, e uma maior facilidade para a programação de protótipos; Uma variedade imensa de bibliotecas, dando suporte à maioria das tarefas relacionadas à programação; Não usa tipos especiais de caracteres no término de cada linha do código em questão, como por exemplo “;” (ponto e vírgula) como é comum em outras linguagens; E ainda suporta diversos tipos de paradigmas de programação, como: orientação a objetos e a

aspectos, funcional, imperativo e reflexivo (RASHED, AHSAN, 2012). Além de tudo, é uma linguagem *open source* e não é exclusiva de um só sistema operacional.

6.1 Paradigmas de Programação

O paradigma serve como um padrão a ser seguido. No mundo da programação, ela serve para definir as principais características da linguagem estudada. Traz consigo a forma com qual a linguagem resolve e funciona para a resolução de diversos problemas em diferentes formas. É capaz de ter mais de um paradigma, como é o caso do *python*.

Um deles, e o mais usado, é o orientado a objetos ou POO. Ele se destaca por fazer uso de objetos previamente descritos através de modelos. Eles são caracterizados por possuir dados primitivos, escritos pelo programador ou pela própria linguagem utilizada. A diferente entre os dois está nas limitações que os módulos feitos pela linguagem têm referente aos feitos pelo programador. (SANTOS, 2003)

Criada em 1997, o paradigma de orientação a aspectos (POA), foi construída em razão dos programadores construírem seus algoritmos voltados mais à regra de negócios, ao invés de preocupar-se com outras camadas do sistema. Pensando dessa forma, percebermos que esse paradigma surgiu como um complemento para outras. Com isso, aumentando a produtividade e facilitando a vida das empresas. (FORTES, 2019)

Um dos mais antigos paradigmas desenvolvidos, a programação imperativa, é a base da arquitetura dos computadores atuais, além de oferecer. De uma maneira simplificada, esse paradigma está ligado diretamente a atribuições de tipos de dados e alocação de memória. E ele é muito usado em diversas linguagens de programação, com o C sendo um dos primeiros a implantar esse paradigma.

> Já que elas surgiram do modelo de von Neumann-Eckert, todas as linguagens imperativas incluem a atribuição como um elemento central. Além disso, elas suportam declarações de variáveis, expressões, comandos condicionais, laços e abstração procedural. Declarações atribuem nomes a locais de memória e associam tipos aos valores armazenados. As expressões são interpretadas por meio da recuperação dos valores correntes das variáveis com nomes a partir das suas respectivas localizações na memória e pelo cálculo de um resultado a partir desses valores. Dada uma referência a uma variável x, a memória retorna o valor corrente no local associado a x. (TUCKER, NOONAN, 2009, p.278)

Esse paradigma é a base da criação de laços de repetição, estruturas condicionais, e praticamente toda estrutura relacionada ao algoritmo. E programação imperativa tornou-se tão comum que hoje programamos usando-a e não percebemos isso.

Segundo Tucker e Norman (2009), o paradigma funcional surgiu na década de 60, com o intuito de apoiar os cientistas a desenvolverem a inteligência artificial e toda a subárea surgida através dela – levando em consideração a existência de somente algumas nessa época. Era notória a dificuldade trazida com as linguagens imperativa para o desenvolvimento, de maneira mais fácil, desse tipo de abordagem.

> (...)a característica essencial da programação funcional é que a computação é vista como uma função matemática mapeando entradas a saídas. Diferentemente da programação imperativa, não há uma noção de estado e, portanto, não há necessidade de uma instrução de atribuição. Assim, o efeito de um laço é obtido via repetição, pois não há uma maneira de incrementar ou decrementar o valor de uma variável no estado, já que não há variáveis. Em termos práticos, porém, muitas linguagens funcionais suportam as noções de variável, atribuição e laço. (TUCKER, NOONAN, 2009, p.362)

Nesse momento, vemos a evolução das linguagens de programação quanto a sua estrutura. A partir daqui o reuso do código e a facilidade no entendimento, e consequentemente em sua produção, está muito desenvolvido e muito bem encaminhado para o futuro.

E o último paradigma suportado pela linguagem *python*, é a programação reflexiva. Trata-se de softwares com a habilidade de resolver problemas por si próprios. Inicialmente surgiu com a inteligência artificial, onde surgiam obstáculos não previstos pelos programadores durante a codificação. Depois disso, começou a ser implementado, junto com a aplicação original do sistema, um dispositivo que permitia a reflexão.

6.2 Erros

Pode haver diferentes tipos de erros enquanto nos aventuramos no mundo da programação. E o *python*, como sendo uma linguagem dinâmica e fortemente tipada, relata esses erros sem permitir que o atual algoritmo escrito seja executado. Tucker e Noonan (2019 p.103) falam que em nível de engenharia de software, erros podem ter um custo muito alto para a empresa responsável se eles forem descobertos após toda a implementação.

Linguagem dinâmica está relacionada com o tipo da variável declarada pelo programador. Normalmente em outras linguagens é obrigatória a declaração do tipo de variável, assim como mostra a figura 8. Porém, em *python*, não é necessário fazer essa declaração, pois a própria linguagem reconhece o tipo de acordo com o valor escolhido, como vemos na figura 9. (ORTIN, 2010)

Figura 8: Declaração de variáveis em Java.

```
public static void main(string args){
//Declaração de variáveis

int a = 10;
int b = 5;

system.out.println ("A soma dos dois é: " + (a + b));
```

Fonte: Próprio Autor (2019)

Figura 9: Declaração de variáveis em Python.

```
i, j = 10, 20
print("O resultado é: ", i + j)
```

Fonte: Correia (2016)

Figura 10: Erro de declaração de variáveis.

```
i, j = 10, "Rafael"
print(i)
print(j)
print("O resultado é: ", i + j)
```

Fonte: Correia (2016)

Figura 11: Erro após a compilação do código.

```
$ python3 teste.py
10
Rafael
Traceback (most recent call last):
  File "teste", line 4, in <module>
    print("O resultado é: ", i + j)
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Fonte: Correia (2016)

O *python* também é uma linguagem fortemente tipada. Dessa forma, uma variável que receber um tipo inteiro, não pode receber qualquer outro tipo. A figura 10 nos revela um erro em relação a isso, onde variáveis de tipo inteiro recebem um valor *String*, retornando a exceção mostrada na figura 11.

Com essa linguagem sendo fortemente tipada e dinâmica, temos uma grande vantagem em relação ao reuso do código e ainda facilidade na declaração de variáveis.

O *python* pode mostrar dois tipos de erros ao usuário: o de sintaxe, como já vimos, e as exceções. Essa segunda refere-se a erros ocorridos durante a execução do programa, mesmo o algoritmo se apresente totalmente correto sintaticamente. Esses tipos de erros parecem ser fatais, mas não são, pois dentro da própria linguagem há um caminho para resolver esse tipo de questão, são os chamados tratamentos de exceções.

7. Estudo de Caso

Para este trabalho utilizaremos o *Microsoft Windows* 10 como Sistema Operacional (SO), por isso iremos nos ater ao funcionamento do *OpenCV* nesse ambiente. No início tentamos rodar o programa no *Microsoft Windows* 7, porem tivemos um grande problema em relação a compatibilidade do *OpenCV* em relação a esse SO – tendo em vista que não mais existe suporte. Não optamos pelo uso de uma máquina virtual porque o computador utilizado para a implementação do *software* tem algumas limitações relacionadas a hardware, dessa maneira deixando a divisão de recursos impraticável.

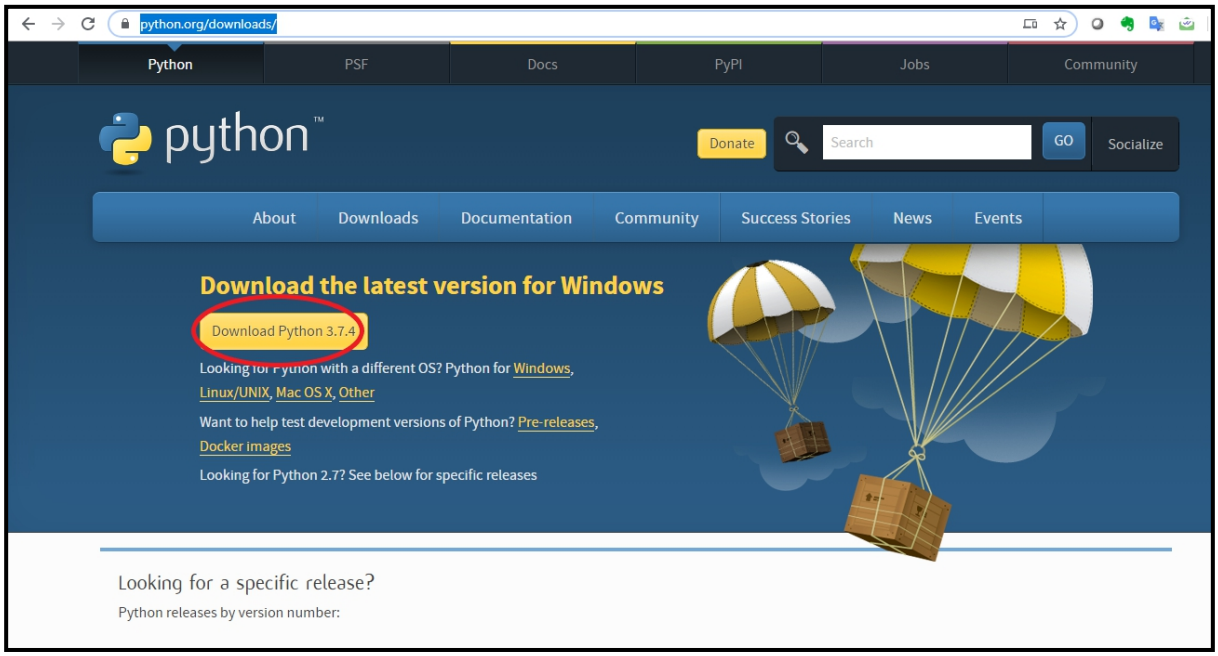
A partir daqui iremos demonstrar a construção de um protótipo de um sistema de reconhecimento facial com a finalidade de identificar discentes de uma instituição de ensino para computar a presença, ou não, do mesmo dentro da sala de aula. Inicialmente teremos uma pequena amostragem, envolvendo poucas pessoas fora do contexto para demonstrar o funcionamento do algoritmo. Porém com a finalidade de aplicação dentro da instituição de ensino superior IESP (Instituto de Educação Superior da Paraíba), que em 2017 contava com aproximadamente cinco mil alunos.

7.1 Configuração do Ambiente de Estudo

Aqui iremos demonstrar o caminho feito para a preparação do ambiente para melhor elucidar os leitores sobre as dificuldades enfrentadas e facilitar futuras configurações para aqueles pretendem utilizar as ferramentas aqui utilizadas, que serão: *OpenCV*, *Python* 3.7.4 e a *IDE PyCharm*.

Primeiro fizemos a instalação do *Python* 3.7.4 através do site de origem, https://www.python.org/downloads/, como mostra a figura 12 e, logo após o *download,* seguimos com o assistente até a finalização do processo.

Figura 12: Site Oficial do Python.



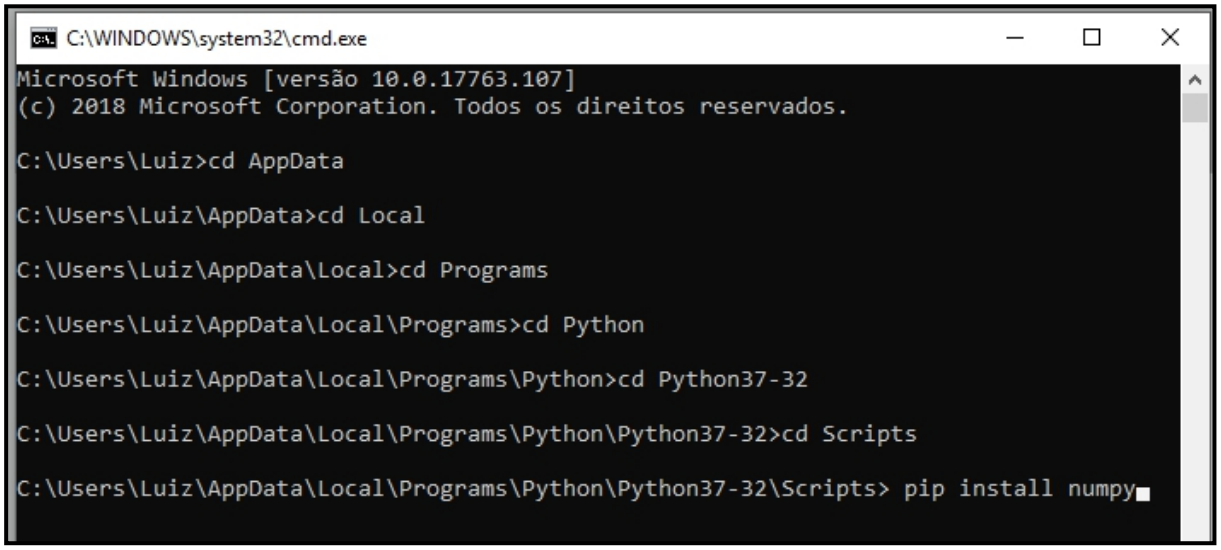
Fonte: Próprio Autor (2019)

O próximo passo será a instalação do *OpenCV*, a parte mais importante durante esse processo. Dependendo do ambiente onde será instalada (como no caso do *Microsoft Windows* 7), essa biblioteca pode trazer desafios para aqueles que pretendem utiliza-lo. Porém, essa dificuldade só irá aparecer mais para frente, quando for iniciar a programação em si. Como foi dito, no início do projeto foi usado o *Microsoft Windows* 7. Todo o processo de instalação foi completado com sucesso, entretanto no momento de "importar" o OpenCV surgiu um erro de reconhecimento da biblioteca. Procuramos soluções por toda a comunidade, havia algumas, mas optamos pela troca de SO por acharmos ser a melhor entre todas.

Devemos acessar o *CMD* (nessa parte não precisa de um conhecimento profundo sobre o assunto) e, de acordo com a figura 13, ir até o caminho C:\Users\Luiz\AppData\Local\Programs\Python\Python37-32\Scripts. Dessa forma podemos instalar o primeiro componente para poder utilizar o reconhecimento facial com *OpenCV,* o *numpy*.

Numpy é um pacote essencial para a computação científica. Existem muitas formas de uso em relação a ele, como: funções básicas de álgebra linear, sofisticadas funções de transmissão, entre outros. Além de todas essas características, o *numpy* pode ser utilizado como um container multidimensional de dados genéricos. Sendo assim, um pacote necessário para complementar o funcionamento do *OpenCV*, visto que o mesmo utiliza matrizes para identificação de características faciais e de objetos.

Figura 13: Instalação do *numpy.*



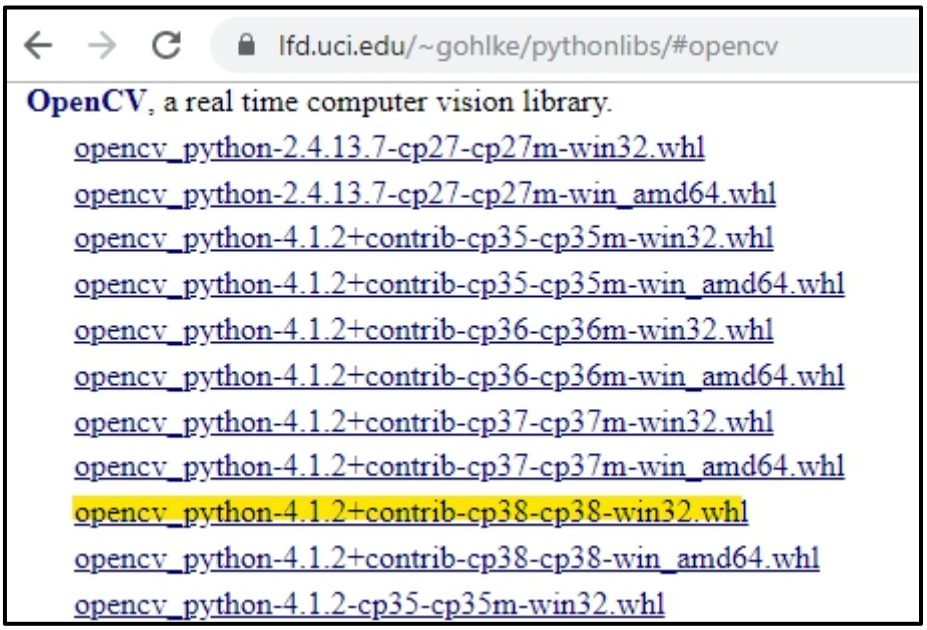
Fonte: Próprio Autor(2019)

No mesmo caminho, mostrado na figura 13, iremos fazer a instalação do *OpenCV*. Essa biblioteca pode ser encontrada através do endereço ttps://www.lfd.uci.edu/~gohlke/pythonlibs/. Aqui iremos encontrar diversos binários de muitas extensões *open-source* científicos para a distribuição oficial do *CPython* da linguagem de programação *Python*. Todos os arquivos presentes nesse endereço têm como finalidade a utilização para testes e avaliações, tornando-os perfeitos para o nosso objetivo.

Portanto, basta procurarmos pelo arquivo especifico do projeto atual, no caso o OpenCV, e poderemos encontrar diferentes versões do mesmo. Todavia iremos implementar a versão mais recente – especificamente o OpenCV 4.1, onde o link para o download desse

arquivo será o opencv_python- 4.1.2+contrib- cp38- cp38- win32.whl, como está destacado na figura 14. A versão instalada, 32 *bits*, está de acordo com a versão atual do *python* instalado na máquina do usuário, não estando correlacionado com a versão do SO utilizado. Para fazer a instalação desse arquivo, ele deverá estar na pasta "*Scripts*" (Figura 13).

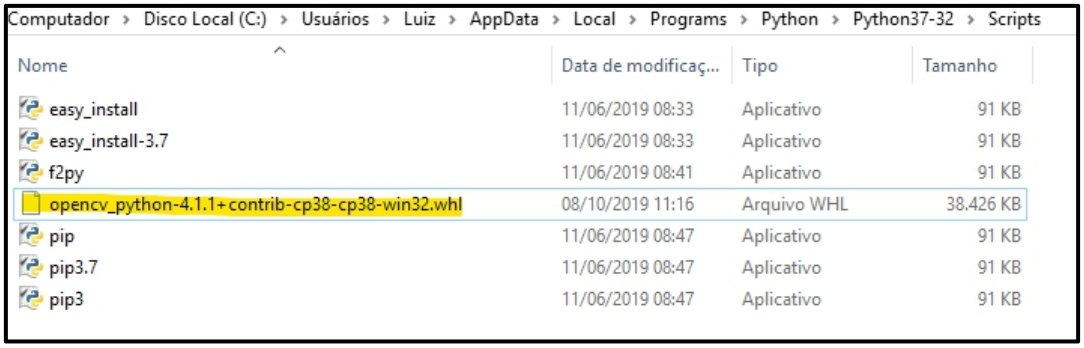
Figura 14: Versão do OpenCV.



Fonte: Próprio Autor (2019)

Uma observação importante é o usuário fazer o download das versões que contenha o pacote *contrib,* caso o objetivo seja o estudo do reconhecimento facial e de objetos. Pois é somente nessas versões que se encontram as configurações necessárias para a utilização do *OpenCV* com essa finalidade. Feito o download desse arquivo, é necessário copiá-lo para a pasta *Scripts*, como mostra a figura 15.

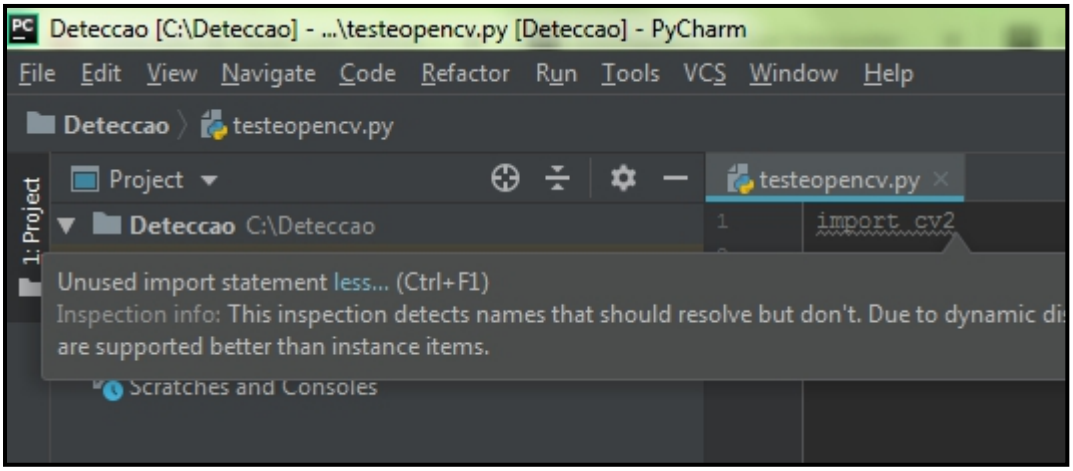
Figura 15: Caminho de instalação do OpenCV.



Fonte: Próprio Autor (2019)

O próximo passo para a configuração do ambiente será a instalação da IDE utilizada para a implementação desse projeto. Para tal utilizamos o *PyCharm Community Edition*, ferramenta de programação distribuída pela *Jetbrains,* apresentando-nos um ambiente satisfatório para a construção do projeto. Essa versão do *PyCharm* tem como características: editor inteligente de *python*, teste e *debbug* gráfico, navegação e refatoração, inspeções de código e suporte VCS. A página para o download dessa ferramenta se encontra no endereço: https://www.jetbrains.com/pycharm/.

Figura 16: Erro de configuração.



Fonte: Próprio Autor (2019)

Para finalizar, escrevemos nosso primeiro código utilizando o *OpenCV* como biblioteca e iniciando os primeiros passos no mundo do reconhecimento facial. Inicialmente tivemos um pequeno problema com a instalação da biblioteca no ambiente de estudo. A *IDE* não estava reconhecendo a importação, assim como vemos na figura 16. Resolvemos esse problema instalamos o *OpenCV* diretamente pelo *PyCharm*. Com isso conseguimos configurar corretamente o ambiente e continuar com a implementação do projeto.

7.2 Reconhecimento Facial utilizando o OpenCV

Depois de explanarmos todo o conteúdo referente ao objeto de estudo e configurado todo o ambiente de programação, podemos seguir em frente e explicar o funcionamento do projeto de reconhecimento de faces. A meta inicial era usar uma amostragem maior, classificando uma determinada turma da instituição de ensino superior IESP - Instituto de

Educação Superior da Paraíba. Entretanto, ao longo do desenvolvimento isso mostrou ser inviável por motivos de locomoção e disponibilidade. Então, o atual projeto foi simplificado ao reconhecimento de duas pessoas.

Na etapa de codificação do projeto foram enfrentados alguns obstáculos, como: problemas na captura das imagens e no reconhecimento. Esses casos serão discutidos em capitulo próximo, nesse iremos focar na parte funcional do projeto.

Incialmente fizemos o básico em relação ao reconhecimento de faces através da webcam do computador utilizado. A figura 17 nos mostra o algoritmo.

Figura 17: Algoritmo básico para reconhecimento.

```
1   import cv2
2
3   video = cv2.VideoCapture(0)
4   classicadorFace = cv2.CascadeClassifier('cascades\\haarcascade_frontalface_default.xml')
5
6   while True:
7       conectado, frame = video.read()
8
9       frameCinza = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
10      faceDetectadas = classicadorFace.detectMultiScale(frameCinza, minSize=(70, 70))
11      for (x, y, l, a) in faceDetectadas:
12          cv2.rectangle(frame, (x, y), (x + l, y + a), (0, 0, 255), 2)
13
14      cv2.imshow("Video", frame)
15
16      if cv2.waitKey(1) == ord('q'):
17          break
18
19  video.release()
20  cv2.destroyAllWindows()
```

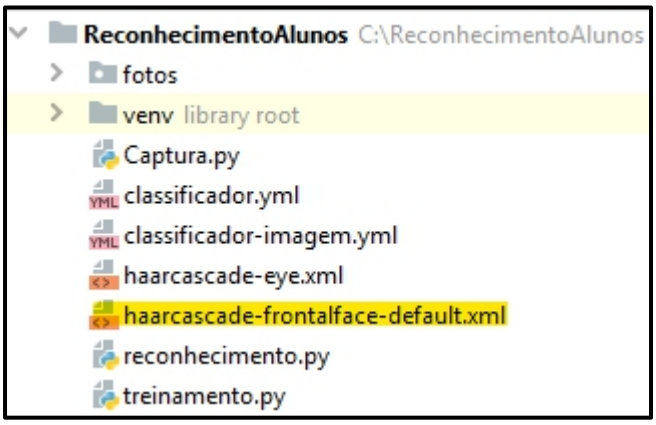
Fonte: Próprio Autor (2019)

Começamos dando um *import* da biblioteca OpenCV, sendo esse o principal passo para a codificação do reconhecimento facial; não somente de faces, como de objetos e outros tipos de imagens personalizadas, mas esse não é foco aqui.

Logo na linha 3, temos a declaração da variável responsável pela identificação de qual câmera será utilizada. O parâmetro passado nesse método está ligado com a quantidade de câmeras ativas no momento. Como o projeto foi feito em um *notebook,* o valor 0 (zero) é o padrão para referenciar quando tem uma única câmera.

Em seguida vemos a declaração do *haarcascade* utilizado nesse projeto. Esse arquivo tem como função o armazenamento de várias matrizes para identificação de rostos, como já citado nesse trabalho. Podendo ser encontrado em qualquer repositório na *internet,* basta copiá-lo diretamente dentro do projeto no qual se estar trabalhando, como mostra a figura 18.

Figura 18: Localização do *Haarcascade.*



Fonte: Próprio Autor (2019)

Ainda de acordo com a figura 17, na linha 6 é o loop utilizado para o vídeo ficar conectado e fazer o reconhecimento sem nenhuma interrupção, além do próprio utilizador. Para isso, temos a declaração de duas variáveis: *conectado* e *frame.* A função das duas está correlacionada com a conexão da câmera atual, onde *conectado* recebe o valor *true* ou *false* identificando se a câmera está ligada ou não; e o *frame,* incumbido de guardar a imagem reconhecida.

Em seguida, o algoritmo deve transformar a imagem capturada em escalas de cinza - pois, dessa forma, o algoritmo *eigenfaces* pode fazer a comparação entre imagens para a realização do treinamento - usando o método cvtColor. Passando-se como parâmetros a variável da imagem capturada e outro método da biblioteca *OpenCV* chamado *COLOR_BGR2GRAY.*

Na linha 10, na figura 17, usamos o método *detectMultiScale* para definir qual o tamanho da imagem a ser capturada. Para isso, existem alguns parâmetros usados: *minSize*, *minNeighbors* e *scaleFactor*.

- *minSize*: Refere-se a quantidade mínima de pixel capturada pela câmera.
- *minNeighbors*: Refere-se a quantidade mínima de vizinhos identificados.
- *scaleFactor*: Refere-se a abrangência da detecção em relação a imagem capturada.

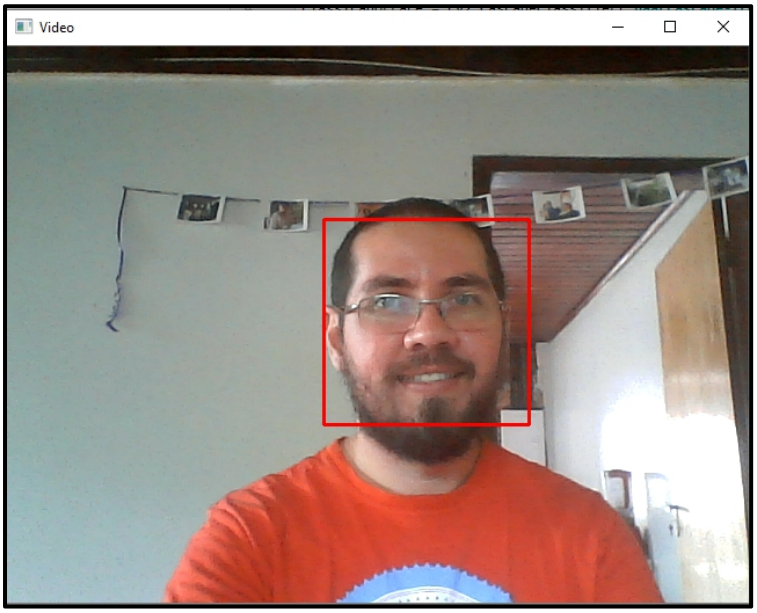
Em relação ao valor desses parâmetros, eles devem ser definidos pelo programador. Quanto menor o número, maior será o nível de detalhamento. À vista disso, deve-se ter uma maior atenção na definição desses valores, uma vez que estes influenciam a velocidade de processamento da máquina utilizada.

Ainda de acordo com a figura 17, temos a aplicação do método *rectangle* presente na biblioteca *OpenCV.* E tem como funcionalidade a demarcação da área de reconhecimento feita pelo algoritmo. Mas, para tal, devemos primeiro fazer uma estrutura de repetição passando os pontos da matriz referente à imagem, e também, a altura e a largura. Com isso, os parâmetros passados no método farão a identificação da área no qual o programador deseja marcar. Na linha 12 do algoritmo percebemos o uso desses parâmetros, começando pelo arquivo escolhido para ser reconhecido. Depois identificamos o início ((x + y)) e o fim ((x + l, y + a)). Os dois últimos são a cor escolhida e a espessura do retângulo produzido, respectivamente.

No laço de repetição *while,* vemos o uso de mais dois métodos: *imshow* e *waitkey.* O primeiro trata-se de qual imagem o programador quer abrir. Nesse caso, passamos uma *string* com o valor "Vídeo" (onde será mostrado no topo da janela aberta) e a variável da imagem trabalhada durante todo o processo. O outro método, *waitkey,* é necessário para o encerramento do vídeo de reconhecimento. Pois, podemos ver, caso não houvesse esse método o vídeo entraria em um loop infinito - e essa é a intenção, porque queremos que o algoritmo fique reconhecendo a imagem de maneira ininterrupta.

Logo ao fim das linhas de código da figura 17, temos mais dois métodos responsáveis pelo encerramento desse exemplo básico de reconhecimento facial: *release* e *destroyAllWindows.* Este tem a finalidade de fechar toda e qualquer janela na qual foi aberta depois da execução do algoritmo. Enquanto o *release* tem como função desligar a câmera utilizada durante o processo. O resultado final é mostrado na figura 19.

Figura 19: Execução do exemplo de algoritmo de reconhecimento facial.



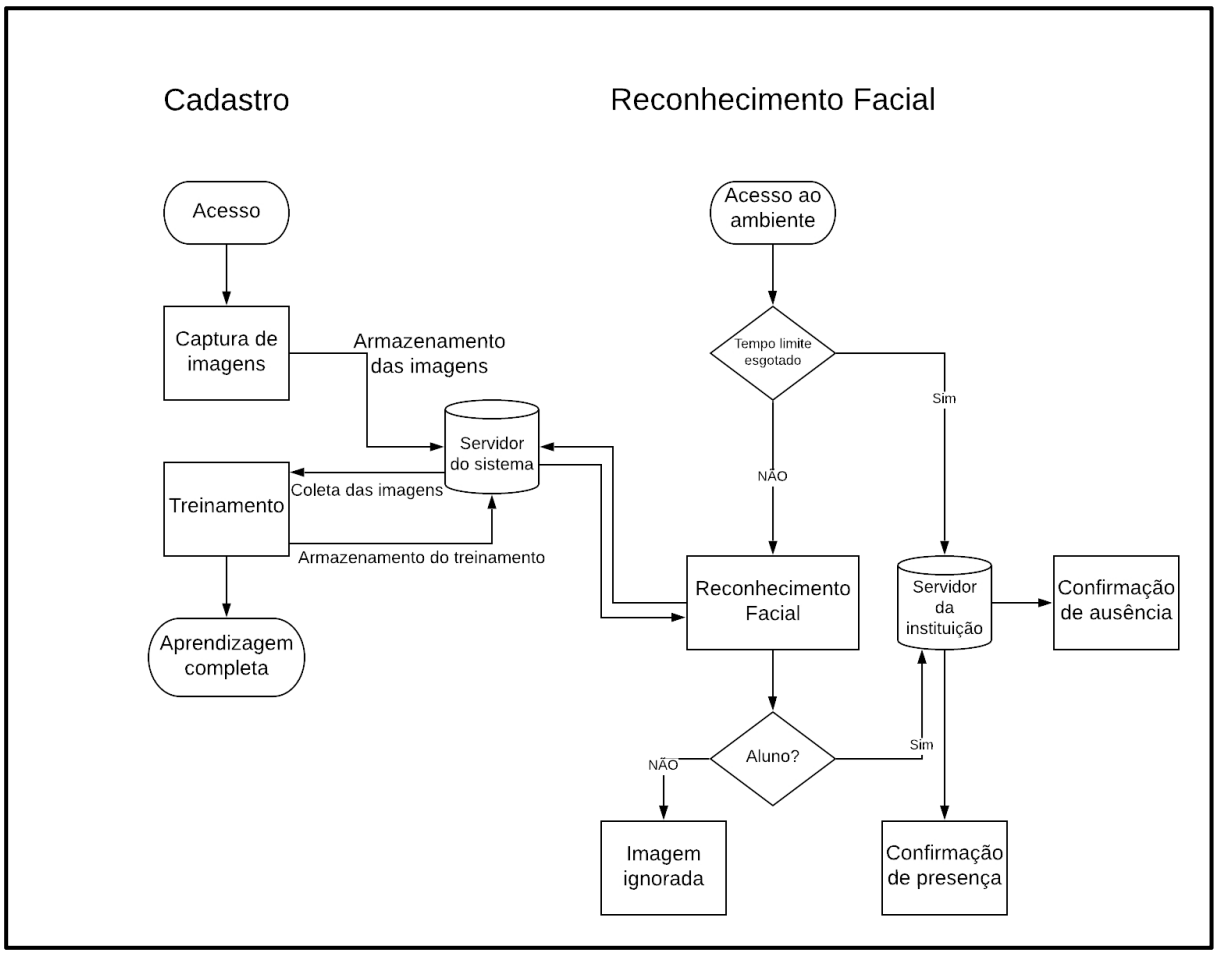
Fonte: Próprio Autor (2019)

7.2.1 Protótipo de Reconhecimento Facial de Discentes

Neste capitulo, iremos nos aprofundar mais em relação ao projeto no qual foi apresentado ao longo deste trabalho. Incialmente será exposto uma ideia de como seria o funcionamento do projeto. Não pudemos ir mais ao fundo e completá-lo de forma a realizar o que foi proposto. Portanto iremos demonstrar um protótipo desse projeto, no intuito de futuramente construí-lo em sua totalidade.

O fluxograma da figura 20 expõe a ideia central de como o sistema irá funcionar. Incialmente temos o cadastro dos alunos no sistema, com o propósito de capturar imagens do discente para o processo de treinamento de imagens formulado pelo algoritmo do sistema de reconhecimento. Feito isso, o aluno cadastrado tem agora sua face reconhecida gravada dentro da base de dados do sistema, podendo ser acessada a qualquer momento.

Figura 20: Fluxograma do sistema



Fonte: Próprio Autor (2019)

O fluxograma também apresenta o processo de reconhecimento de imagens após o cadastro de todos os discentes. Essa parte ainda está em estudo, porém demonstramos uma ideia geral do funcionamento dela. Pensamos em um limite de tempo para o acesso à sala de aula como a forma ideal do gerenciamento da ata de verificação de presença de discentes - claro, reforçando, ainda não temos o conhecimento suficiente para confirmar esse método como sendo viável. Dito isso, definimos dois casos possíveis: tempo limite esgotado ou não. Caso tenha passado desse limite proposto, o sistema irá acessar a base de dados da instituição e constatar como ausentes aqueles que não foram identificados pelo algoritmo. Agora, se ainda dentro do limite, o discente acessar o ambiente, a câmera do local irá efetuar o reconhecimento facial e identificar como sendo um discente cadastrado ou não. Estando devidamente cadastrado, o sistema irá acessar a base de dados da instituição e confirmar a presença do discente reconhecido. Caso contrário, o sistema irá simplesmente ignorar o indivíduo que entrou no ambiente.

Esses são os pontos principais identificados até esse momento. Em futuros trabalhos o sistema poderá ser melhorado, surgindo novas ideias e implantando funcionalidades novas e consolidando as antigas.

7.2.2 Algoritmo de Captura de Imagens

A partir deste momento iremos abordar sobre todos os aspectos do sistema no qual conseguimos realizar a programação. Construímos três arquivos diferentes de *python:* um para captura (abordaremos aqui essa classe), outro para o treinamento das imagens e mais um para o reconhecimento facial. Cada um deles tem suas características e obstáculos enfrentados durante o processo de codificação.

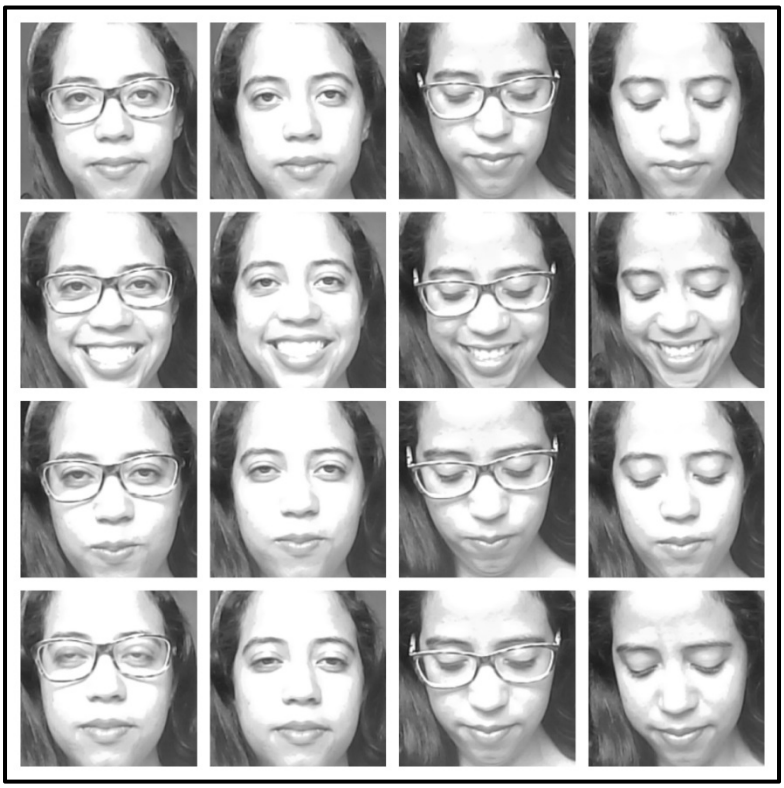
A classe de captura de imagem é baseada no exemplo visto na figura 15, onde o algoritmo faz uma simples captura de imagem identificando a face do usuário. Entretanto, modificamos alguns componentes no intuito de melhorar a identificação do usuário.

O processo de captura foi feito usando duas pessoas. Para isso, antes realizamos um roteiro de quais expressões seriam usadas e quais acessórios, sendo eles: sério, sorrindo, com raiva e triste e, com ou sem óculos, respectivamente. Além disso, foram capturados cinco tipos de fotos diferentes relacionado à direção, como: para cima, para baixo e para os lados, demonstrado na figura 21 e 22. Escolhemos fazer as capturas das imagens dessa maneira para termos maior quantidade e qualidade de fotos para o próximo processo, o de treinamento.

Figura 21: Fotos capturadas pelo algoritmo de reconhecimento facial. Primeira pessoa.

Fonte: Próprio Autor (2019)

Figura 22: Fotos capturadas pelo algoritmo de reconhecimento facial. Segunda pessoa.



Fonte: Próprio Autor (2019)

A maior dificuldade na captura das imagens está relacionada com o nível de iluminação do ambiente, pois essa é a característica mais importante ligada ao treinamento das imagens. Para o *OpenCV* executar o reconhecimento de faces de maneira correta, a variação média da iluminação do ambiente deve estar acima de 110, calculada pelo método *average,* encontrado no pacote *numpy* do *Python,* como vemos na figura 23.

Figura 23: Classe de captura de imagens usando *Python* e *OpenCV.*

```python
import cv2
import numpy as np

video = cv2.VideoCapture(0)

classicadorFace = cv2.CascadeClassifier("haarcascade-frontalface-default.xml")

amostra = 1
numeroAmostra = 40
id = input('Digite seu identificador: ')
largura, altura = 220, 220
print("Iniciando a captura das fotos...")

while True:
    conectado, frame = video.read()
    frameCinza = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    faceDetectadas = classicadorFace.detectMultiScale(frameCinza, minSize=(100, 100), scaleFactor=1.5)

    for (x, y, l, a) in faceDetectadas:
        cv2.rectangle(frame, (x, y), (x + l, y + a), (0, 0, 255), 2)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            if np.average(frameCinza)> 110:
                imagemFace = cv2.resize(frameCinza[y:y +a, x:x +l], (largura, altura))
                cv2.imwrite("fotos/pessoa" + "." + str(id) + "." + str(amostra) + ".jpg", imagemFace)
                print("Foto " + str(amostra) + " capturada com sucesso!")
                amostra += 1

    cv2.imshow("Video", frame)
    if cv2.waitKey(1) == ord('s'):
        break
    if (amostra >= numeroAmostra + 1):
        break
print("Faces capturadas com sucesso")
video.release()
```

Fonte: Próprio Autor (2019)

A figura 23 demonstra o algoritmo responsável pela captura das imagens que, por sua vez, irão para a parte de treinamento. Podemos perceber também a semelhança desse código com o da figura 17, pois o usamos como base. Olhando de forma mais detalhada, temos

algumas diferenças. Por isso, iremos explicar o funcionamento desse algoritmo para um melhor entendimento.

No início do código definimos a quantidade de amostras e de fotos utilizadas para esse processo. Optamos por uma amostra mínima, pois ainda estamos em fase de aprimoramento do sistema e, dessa forma, nos ajudaram a não ter de repetir o processo várias vezes para as duas pessoas; dividimos em duas partes: a primeira rodada de fotos com a primeira pessoa e, logo após, com a segunda. Ainda assim, por motivos de iluminação fraca e reconhecimento errado, tivemos que repetir o processo por diversas vezes.

Logo após as variáveis: *id, largura* e *altura* são definidos. Sendo o *id* responsável pela identificação de qual amostra está sendo feita e, quanto à *largura* e *altura,* estão relacionadas com o tamanho da imagem após as capturas.

Na figura 23, a partir da linha 20 até aos 28, é onde acontece o tratamento da imagem no qual está sendo capturada. O método *average* define o valor mínimo aceitável de luminosidade do ambiente. Logo após o algoritmo começa a edição da foto capturada; primeiro com o *resize*, editando a imagem para um tamanho mais recomendável para o treinamento; *imwrite*, determinando o nome do arquivo final e, finalmente, a incrementação do número de amostras atuais.

Enfim, de uma maneira resumida, a câmera do *notebook* utilizado nesse projeto é ligada e começa o reconhecimento facial. Com o atalho criado no *if*, linha 23 da figura 23, através da transformação do hexadecimal no "q", começa a captura das fotos com posições pré-determinadas, sendo 40 ao total. Feito isso, ao final do processo teremos, em uma pasta chamada "fotos", todas as imagens capturadas nesse processo inicial.

7.2.3 Treinamento das Imagens

Concluída a etapa de captura de imagens, iremos para outro processo muito importante no reconhecimento facial e a parte que exige um maior processamento em relação a máquina, o treinamento das imagens capturadas.

Nessa etapa do reconhecimento poderíamos três algoritmos, são eles: *eigenfaces, fisherfaces e LBPH.* Existem semelhanças e diferenças entre eles, mas a principal característica de todos é em relação com a luminosidade. Anteriormente já detalhamos sobre cada um.

A escolha do algoritmo ideal para este trabalho será feita através de alguns testes manuais, passando parâmetros e mudando de um para outro até atingirmos o melhor

reconhecimento de acordo com os materiais disponíveis. Porém, faremos isso é uma outra parte deste trabalho. A seguir analisaremos o código responsável pelo treinamento das imagens, usando o *eigenfaces* como exemplo.

Figura 24: Classe de treinamento das imagens.

```
import cv2
import os
import numpy as np

eigenfaces = cv2.face.EigenFaceRecognizer_create()

def getImagemComId():
    caminhos = [os.path.join('fotos', f) for f in os.listdir('fotos')]
    faces = []
    ids = []
    for caminhoImagem in caminhos:
        imagemFace = cv2.cvtColor(cv2.imread(caminhoImagem), cv2.COLOR_BGR2GRAY)
        id = int (os.path.split(caminhoImagem)[-1].split('.')[1])
        ids.append(id)
        faces.append(imagemFace)
    return np.array(ids), faces

ids, faces = getImagemComId()

print("Treinamento em andamento...")
eigenfaces.train(faces, ids)
eigenfaces.write('classificador.yml')

print("Treinamento completo.")

```

Fonte: Próprio Autor (2019)

Sendo o primeiro algoritmo projetado para o reconhecimento facial e de objeto, o *eigenfaces* pode ser considerado o mais “simples” dentre todos. Sua porcentagem de acerto no reconhecimento facial é a mais alta, pelo fato do nível de proximidade dentre as *vectorfaces* serem as mais distantes. Essa distância refere-se ao grau de similaridade entre as imagens conseguidas através do algoritmo de captura, visto na figura 23, e as *vectorfaces* - conhecidas também por imagens fantasmas - geradas a partir do algoritmo de treinamento de imagens usando *eigenfaces* (figura 24).

Desse ponto em diante, com a finalidade de um melhor entendimento sobre o código da figura 24, iremos explicar detalhadamente cada método aplicado e, ao final, resumir de modo geral seu funcionamento.

Com o objetivo de termos acesso a pasta destino das imagens capturadas a partir do primeiro processo (visto na figura 23), realizamos o *import* da biblioteca *os.* Segundo o site oficial do *python* (encontrado em https://docs.python.org/3/library/os.html), o módulo *os,* ou interfaces diversas do sistema operacional, tem como objetivo o uso de funcionalidades inerentes ao sistema operacional de uma maneira portátil. Nesse módulo, podemos abrir um arquivo, manipular caminhos e fazer a leitura de todas as linhas de todos os arquivos. Fazemos o uso de uma dessas funcionalidades na linha 8 e 13, com o objetivo de ter acesso às imagens utilizando o método *join* e editando os nomes das imagens com o método *Split*, respectivamente.

Na linha 5, declaramos a variável responsável por realizar o treinamento. Com o método *face* da biblioteca *OpenCV*, fazemos a chamada do algoritmo escolhido para a realização do treinamento, o *eigenfaces.* Logo após isso, definimos uma função incumbida de realizar o tratamento das imagens. Nas próximas linhas declaramos mais três variáveis; caminhos, onde o algoritmo irá ter acesso à pasta das imagens; faces, vetor onde serão guardadas as imagens encontradas; ids, vetor que irá armazenar os ids referentes às imagens.

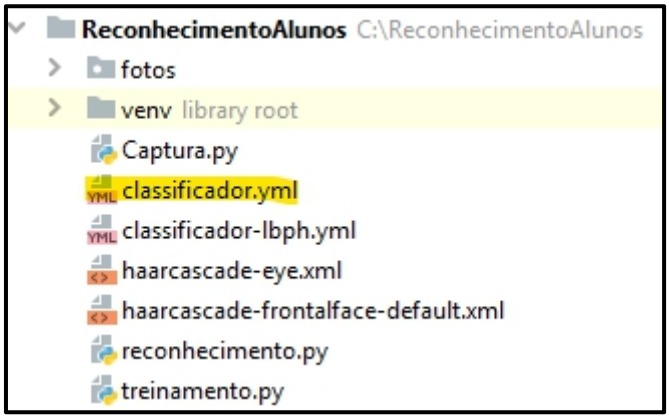
O laço de repetição *for* (figura 24) *é* responsável pela identificação das fotos, edição dos nomes e armazenamento desses valores em dois vetores: faces e ids. Primeiro, deve-se ter a transformação das imagens recolhidas da pasta destino do processo de captura em escala de cinza, apesar das mesmas já estarem dessa forma. Isto está relacionado com a criação dos *eigenvectors*. Depois realizamos a declaração do método para a edição dos nomes das imagens, assim o resultado final será somente o número da amostra.

A seguir, nas linhas 14 e 15, como vemos na figura 24, é alocada todos os valores coletados a partir das variáveis “imagemFace” (alocado no vetor faces) e “id” (alocado no vetor ids) para os vetores criados a partir do método *append.* Logo após, com o uso do método *array* do *numpy*, fazemos a conversão das duas variáveis em uma lista.

Ao final do algoritmo chamamos o método responsável por fazer o treinamento das imagens tratadas durante todo o processo (chamado *train*) e como parâmetros passamos as duas listas criadas anteriormente. E, finalizando o algoritmo, passamos todas as informações coletadas do treinamento para um arquivo, utilizando o método *write*. Esses dois métodos fazem parte do algoritmo de reconhecimento de faces e objetos *eigenfaces*.

Essa parte do reconhecimento facial terminar com o método *write*. Este responsável por armazenar todas as informações geradas pelo treinamento de faces. Então, após a execução do algoritmo irá aparecer no projeto um arquivo chamado “classificador.yml” (figura 25). Ele servirá para a próxima, e última, etapa do sistema: o reconhecimento facial.

Figura 25: Arquivo de treinamento.



Fonte: Próprio Autor (2019)

7.2.4 Reconhecimento Facial

Sendo a última etapa de todo o processo, o algoritmo que será apresentado a seguir tem a capacidade de reconhecer os indivíduos estudados na etapa de captura das imagens. Assim como fizemos em todas as etapas, iremos explicar detalhadamente os métodos utilizados no código.

Ao final da execução desse algoritmo, poderemos identificar as duas pessoas voluntárias desse trabalho juntamente com seu nome. Com isso, poderíamos ter uma ideia de como seria a construção do restante do trabalho idealizado no começo. Poderíamos acessar o banco de dados da instituição e comparar com os nomes cadastrados no sistema de reconhecimento ou formar um sistema próprio de confirmação de presença ou ausência do discente. De qualquer forma, é um conhecimento que ainda não temos e que deverá ser estudado em trabalhos futuros. Apresentamos uma “metade do caminho”, demonstrando a viabilidade do sistema. Dito isso, daqui em diante focaremos na explicação deste último algoritmo.

Como vemos na figura 26, o arquivo de reconhecimento de faces é muito parecido com o que vimos na figura 23, onde tratamos sobre a captura de imagens. A diferença aqui é que teremos o reconhecimento facial completo com o número de proximidade dos *eigenvectors* e com o nome dos respectivos indivíduos que participaram voluntariamente do projeto.

Figura 26: Classe de Reconhecimento Facial.

```
1   import cv2
2
3   detectorFace = cv2.CascadeClassifier("haarcascade-frontalface-default.xml")
4   reconhecedor = cv2.face.EigenFaceRecognizer_create()
5   reconhecedor.read("classificador.yml")
6   largura, altura = 220, 220
7   font = cv2.FONT_HERSHEY_COMPLEX_SMALL
8   camera = cv2.VideoCapture(0)
9
10  while (True):
11      conectado, frame = camera.read()
12      imagemCinza = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
13      faceDetectadas = detectorFace.detectMultiScale(imagemCinza, scaleFactor=1.5, minSize=(70, 70))
14
15      for (x, y, l, a) in faceDetectadas:
16          imagemFace = cv2.resize(imagemCinza[y:y + a, x:x + l], (largura, altura))
17          cv2.rectangle(frame, (x, y), (x + l, y + a), (0,0,255), 2)
18          id = reconhecedor.predict(imagemFace)
19          nome = ""
20          if id == 1:
21              nome = 'Luiz Augusto'
22          elif id == 2:
23              nome = 'Barbara'
24          cv2.putText(frame, nome, (x,y +(a+30)), font, 2, (0,0,255))
25
26      cv2.imshow("RFacial", frame)
27      if cv2.waitKey(1) == ord('q'):
28          break
29
30  camera.release()
31  cv2.destroyAllWindows()
```

Fonte: Próprio Autor (2019)

No início do algoritmo há a declaração de duas variáveis (detectorFace e reconhecedor), encarregados, respectivamente, de “chamar” o arquivo *Haarcascade* com as informações de detecção de rostos e o algoritmo de treinamento de imagens. Este incumbido de acessar o arquivo gerado pela etapa de treinamento.

Após isso, definimos o tamanho da imagem a ser reconhecida para um tamanho que seja praticável, pois quanto maior a imagem, maior será a quantidade de pixel analisada e, consequentemente, maior será o processamento exigido à máquina em uso. Em seguida, declaramos uma variável para a fonte utilizada na definição dos nomes.

Nas linhas subsequentes da figura 26, repetimos os passos feitos na figura 23; declaração da variável para selecionar a câmera que será utilizada no sistema (o valor zero significando a presença de apenas um dispositivo), usando a biblioteca *OpenCV*; Inicio do laço de repetição *while*, declarando as variáveis para determinar se a câmera está funcionando (linha 11), depois transformar a imagem detectada em escala de cinza (linha 12) e definir as dimensões da área de detecção de faces (linha 13). Este último, passamos parâmetros focando

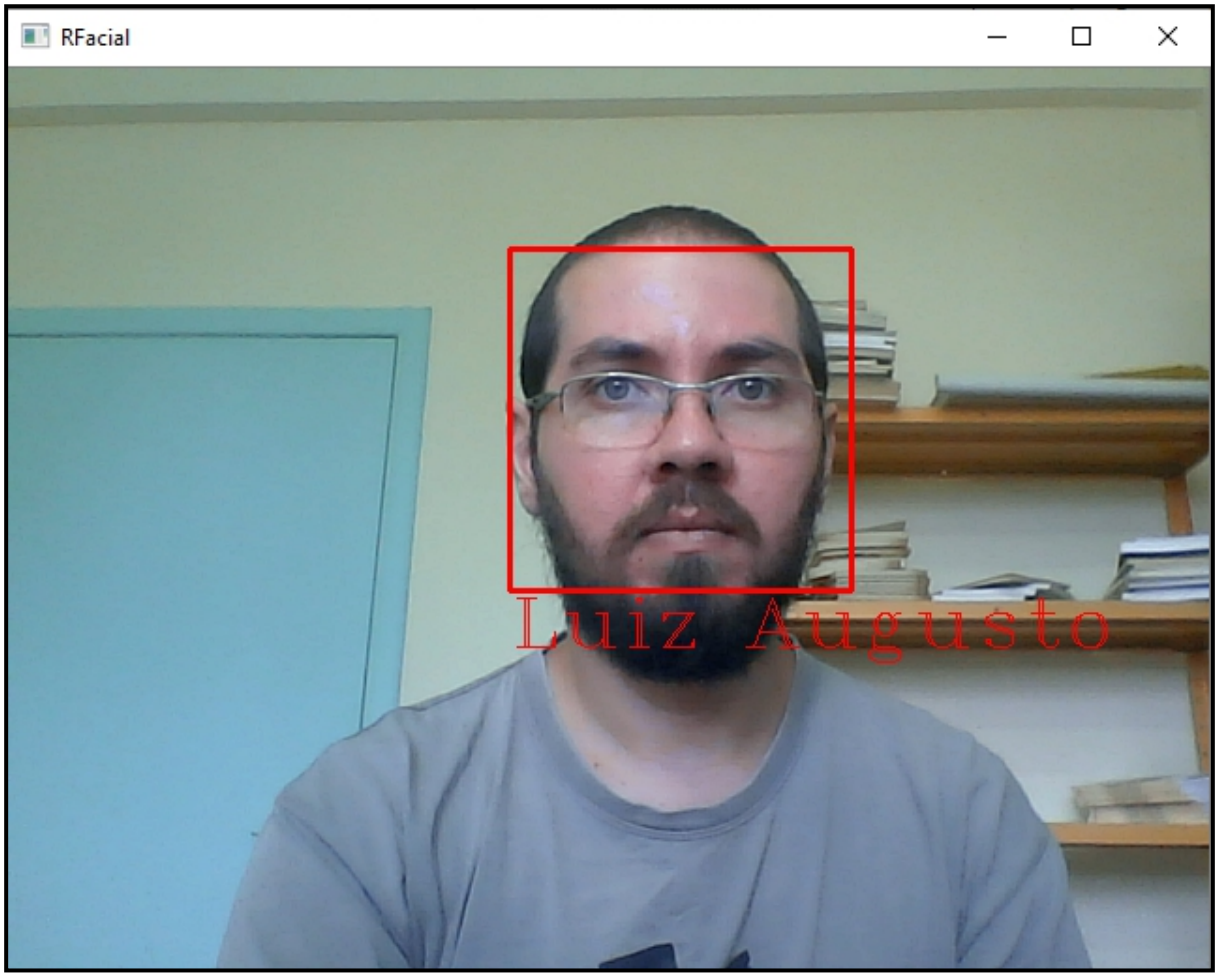
no dispositivo disponível no momento - câmera de um *notebook.* Visto que devemos modificar estes parâmetros de acordo com os dispositivos escolhidos.

Na linha 15 (figura 26) marca o início do laço de repetição responsável pelo reconhecimento facial. Nesse *loop* será definido o tamanho da imagem, as configurações do marcador de faces e de qual forma serão demarcados os nomes reconhecidos pelo algoritmo. A variável "imagemFace" redimensiona a imagem passando como parâmetros os valores pré-definidos anteriormente na linha 6. Após isso, definimos o tamanho do retângulo que irá aparecer na imagem, marcando a área reconhecida; Passamos como parâmetros: o início e o fim do ponto de marcação, a cor e a espessura da linha.

Nas próximas linhas do algoritmo (18 a 24 da figura 26), fazemos a comparação do arquivo com as imagens treinadas com a imagem que está sendo passada pelo dispositivo conectado (linha 18), para isso utilizamos o método *predict* do *FaceRecognizer*; e, ainda, temos a definição das amostras coletadas - identificando o número do id correspondente de cada indivíduo. Onde declaramos uma variável nome com a valor vazio, e em seguida, fazemos um *if* para que haja uma comparação com a imagem atualmente capturada e identifique qual pessoa está sendo reconhecida, dessa maneira aparecendo na tela seu nome. Fizemos a escolha do *if* por esse projeto ter um número pequeno de amostras. Porém, caso seja maior a quantidade de pessoas, poderíamos optar por usar o *switch*.

Terminando o laço *for*, da linha 15, vemos o uso do método *putText.* Onde poderemos editar o texto que aparecerá no vídeo, nesse caso seria o nome do indivíduo. Os parâmetros passados aqui são referentes à, respectivamente: variável da imagem atual capturada, variável do nome reconhecido, espaçamento entre pontos, fonte escolhida, espessura e cor. Depois disso tudo, há a finalização do algoritmo.

Figura 27: Reconhecimento Facial com identificação de nome.



Fonte: Próprio Autor (2019)

8. Metodologia

A metodologia utilizada foi uma revisão sistemática da literatura, tendo como ferramenta norteadora, material já publicado sobre o tema; livros, artigos científicos, publicações em periódicos e materiais disponíveis na internet. Além disso, como base de conhecimento sobre o assunto de reconhecimento facial, foram feitos curso através da plataforma digita de ensino *Udemy*. Os cursos feitos foram todos ministrados pelo professor Jones Granatyr, são eles: Reconhecimento Facial com Python e OpenCV e Detecção de Faces com Python e OpenCV.

Como objetivo final de entender mais sobre a inteligência artificial e sua área de reconhecimentos de padrões como apoio ao entendimento do reconhecimento facial como um instrumento facilitador para diversas áreas de estudo e, claro, para o atual tema aqui discutido.

9. Análise dos Resultados

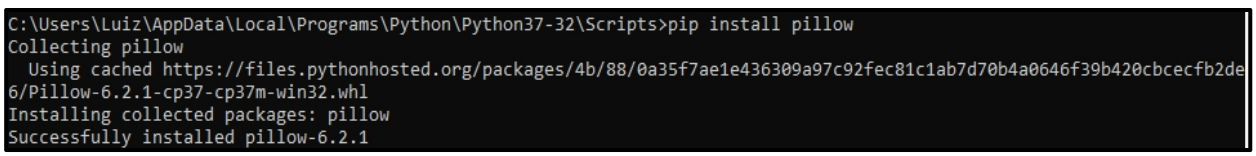
Assim como falamos em capítulos anteriores, existem três tipos de algoritmos para o reconhecimento facial: *Eigenfaces*, *Fisherfaces* e *Local Binary Patterns Histograms* (*LBPH*). Para definir qual será utilizado para o projeto, tivemos que executar testes com cada um deles de forma "manual". Ou seja, para cada um mudamos os parâmetros de seus respectivos métodos responsáveis pelo treinamento e detecção de imagens. Depois disso separamos os resultados e comparamos todos, escolhendo a partir disso o melhor algoritmo.

Inicialmente, como não tínhamos o conhecimento desses testes, optamos por usar o *Eigenfaces,* mas para demonstrar a capacidade de cada algoritmo realizamos diversos testes. Porém, esses testes foram realizados pensando no ambiente atual escolhido e amostragem disponível no momento. Portanto, para a ideia inicial de implementar em uma instituição de nível superior possivelmente não iria funcionar por ser um ambiente totalmente diferente.

Para o teste foi utilizado a *Yale Face Database,* que consiste em um banco de dados gratuito com inúmeras imagens de pessoas, cada uma com diversas expressões faciais e iluminação diferentes. Esta base contém 165 imagens em escala de cinza e ainda conta com a participação de 15 pessoas diferentes. O uso desse banco de dados é comum para testar projetos que exigem o reconhecimento facial, seja via *webcam* ou de forma estática através de fotos. Ela pode ser encontrada através do site: http://vision.ucsd.edu/content/yale-face-database. (A.S.GEORGHIADES, 2001).

Também foi utilizada a biblioteca do *Python* chamada *pillow.* Ela é responsável por carregar as imagens escolhidas para o teste. Possivelmente sem essa biblioteca haveria um erro quando as imagens fossem usadas no projeto. O *pillow* deve ser instalado usando o mesmo processo feito para a instalação do *numpy* e *OpenCV,* acessando o caminho C:\Users\Luiz\AppData\Local\Programs\Python\Python37-32\Scripts através do *CMD* e, logo após isso, digitar o comando "*pip install pillow"*, como nos mostra a figura 28.

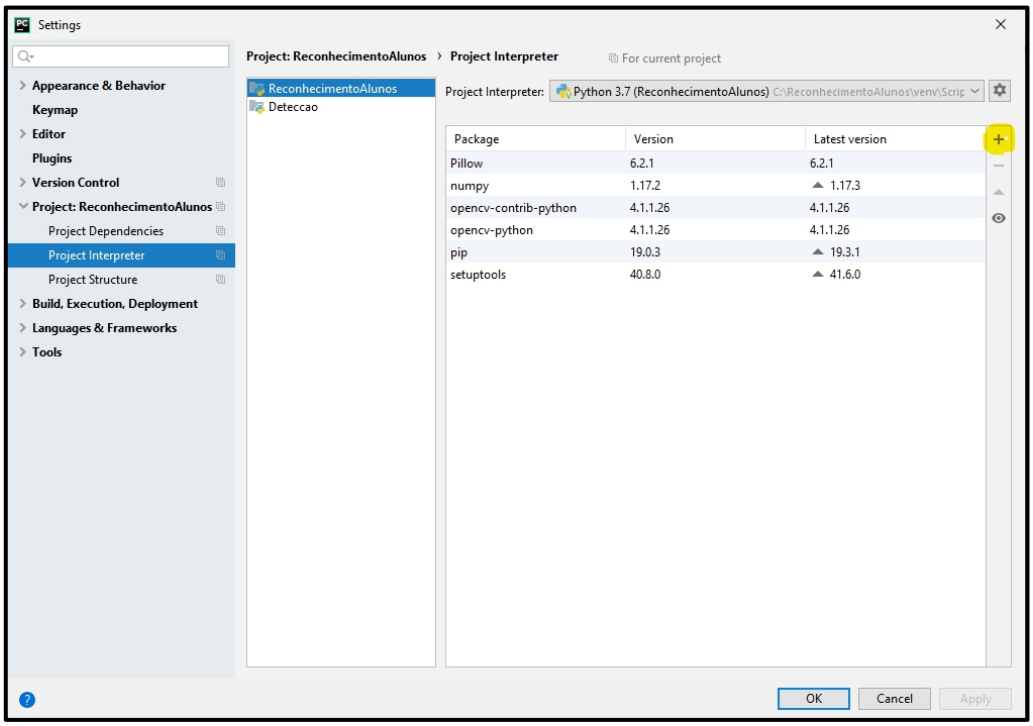
Figura 28: Instalação do *pillow.*



Fonte: Próprio Autor (2019)

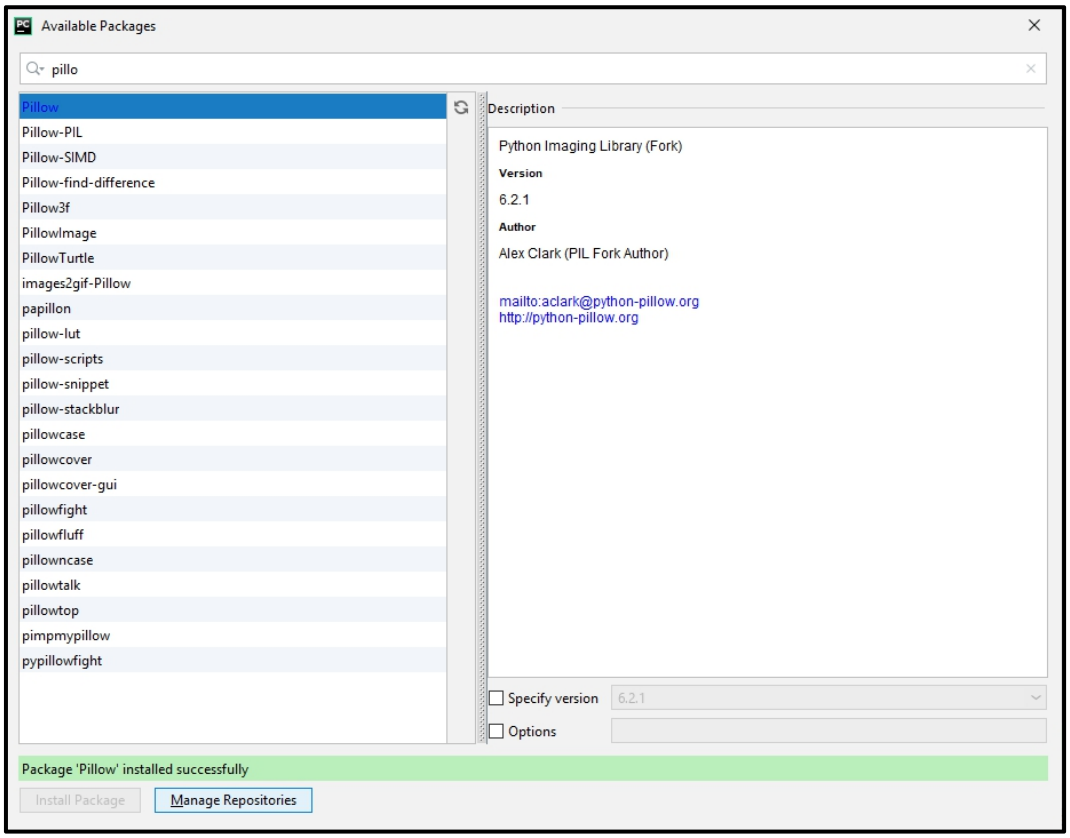
Depois de ter feito todo esse processo, ainda tivemos que fazer o download do *pillow* manualmente através do repositório do *PyCharm*. Isso acontece porque o projeto atual ainda não tinha sido atualizado. Para fazer esse download devemos acessar *file* > *settings* > *Project:ReconhecimentoAlunos* > *Project interpreter* e selecionamos a opção "+", como nos mostra a figura 29. Em seguida, procuramos por *pillow* e fazemos a instalação dele (figura 30).

Figura 29: *Project Interpreter.*



Fonte: Próprio Autor (2019)

Figura 30: Instalando o *pillow* através do repositório do *Pycharm.*



Fonte: Próprio Autor (2019)

Para não modificarmos nenhuma das classes feitas anteriormente para treinamento e reconhecimento facial, criamos duas novas (figura 31 e 32) para fazermos esse teste. Essas classes consistem basicamente na criação de arquivos com os três algoritmos disponíveis e para o teste em si envolvendo a mudança de parâmetros de acordo com a taxa de luminosidade, a distância dos *Eigenfaces* e outros fatores.

Na figura 31, podemos observar algumas mudanças fundamentais em relação ao algoritmo. Objetivando um melhor aproveitamento, optamos por fazer o processamento dos três algoritmos na mesma classe. Dessa forma, da linha 6 a 8, tem a declaração das três variáveis que recebem os algoritmos de reconhecimento. Na linha 6, declaramos o *Eigenfaces* com dois parâmetros: número de componentes e valor de *threshold.* Da mesma maneira fizemos com o *Fisherface*, tendo em vista a proximidade entre esses dois algoritmos, eles recebem os mesmos tipos de parâmetros. Na linha 8, vemos o reconhecimento facial feito pelo *Local Binary Patterns Histograms.* Onde observamos uma diferença entre os parâmetros, sendo eles respectivamente: *radius, neighbors, gri_x, grid_y e threshold.*

Figura 31: Classe Teste de Treinamento.

```
import ...

eigenface = cv2.face.EigenFaceRecognizer_create(40, 8000)
fisherface = cv2.face.FisherFaceRecognizer_create(3, 2000)
lbph = cv2.face.LBPHFaceRecognizer_create(2, 2, 7, 7, 50)

def getImagemComId():
    caminhos = [os.path.join('yalefaces/treinamento', f) for f in os.listdir('yalefaces/treinamento')]
    faces = []
    ids = []
    for caminhoImagem in caminhos:
        imagemFace = Image.open(caminhoImagem).convert('L')
        imagemNP = np.array(imagemFace, 'uint8')
        id = int(os.path.split(caminhoImagem)[1].split(".")[0].replace("subject", ""))
        ids.append(id)
        faces.append(imagemNP)

    return np.array(ids), faces

ids, faces = getImagemComId()

print("Treinando...")
eigenface.train(faces, ids)
eigenface.write('classificadorEigenYale.yml')

fisherface.train(faces, ids)
fisherface.write('classificadorFisherYale.yml')

lbph.train(faces, ids)
lbph.write('classificadorLBPHYale.yml')

print("Treinamento realizado")
```

Fonte: Próprio Autor (2019)

Radius: O raio usado para construir o padrão binário local circular. Quanto maior o raio, mais suave será a imagem, mas haverá mais informações espaciais.
Neighbors: O número de pontos de amostra necessários para construir um padrão binário local circular. Um valor apropriado é usar 8 pontos de amostra. Quanto mais pontos, maior será o custo computacional.
Grid_x e *Grid_y:* Número de células na horizontal e vertical, respectivamente.
Threshold: O limite aplicado na previsão. Se a distância até o vizinho mais próximo for maior que o limite, esse método retornará -1.

Possuindo o conhecimento do funcionamento dos parâmetros no momento da criação das variáveis, realizamos teste modificando cada um de acordo com a capacidade de processamento da máquina utilizada. Os parâmetros iniciais passados na figura 31 foram escolhidos aleatoriamente e como um ponto de início, pois não existe um padrão pré-estabelecido para a realização dos testes ou a modificação desses parâmetros.

Na linha 16, transformamos as imagens capturadas em um *array*, pois caso isso não acontece ocorrerá um erro e as imagens não serão processadas. Assim, passamos como parâmetros a imagem, obtida na pasta da base de dados *yale*, e *'unit8'*, transformando os valores em inteiros não assinados, ou seja, inteiros que guardam apenas valores positivos.

No restante do algoritmo, somos capazes de constatar a semelhança com a classe de treinamento anteriormente estudada, exposta na figura 24. Ao fim da execução, o algoritmo irá acessar a pasta das imagens da base *yale*, realizar o treinamento com os três algoritmos e gerar três arquivos finais com as informações de todas as imagens treinadas.

A partir disso, acessamos esses arquivos com a segunda classe feita (figura 32). Onde esse algoritmo faz o reconhecimento de outro pacote de imagens - originadas a partir da mesma base de dados - com os mesmos indivíduos, porém com outras expressões faciais. Nessa classe, para maior agilidade do processo de teste, também repetimos o mesmo processo para todos os algoritmos. Deixando em comentário aqueles não utilizados no momento.

Figura 32: Classe Teste de Reconhecimento Facial.

```
6   detectorFace = cv2.CascadeClassifier("haarcascade-frontalface-default.xml")
7   #reconhecedor = cv2.face.EigenFaceRecognizer_create()
8   #reconhecedor.read("classificadorEigenYale.yml")
9   #reconhecedor = cv2.face.FisherFaceRecognizer_create()
10  #reconhecedor.read("classificadorFisherYale.yml")
11  reconhecedor = cv2.face.LBPHFaceRecognizer_create()
12  reconhecedor.read("classificadorLBPHYale.yml")
13
14  totalAcertos = 0
15  percentualAcerto = 0.0
16  totalConfianca = 0.0
17
18  caminhos = [os.path.join('yalefaces/teste', f) for f in os.listdir('yalefaces/teste')]
19  for caminhoImagem in caminhos:
20      imagemFace = Image.open(caminhoImagem).convert('L')
21      imagemFaceNP = np.array(imagemFace, 'uint8')
22      facesDetectadas = detectorFace.detectMultiScale(imagemFaceNP)
23      for (x, y, l, a) in facesDetectadas:
24          idprevisto, confianca = reconhecedor.predict(imagemFaceNP)
25          idatual = int(os.path.split(caminhoImagem)[1].split(".")[0].replace("subject", ""))
26          print(str(idatual) + " foi classificado como " + str(idprevisto) + " - " + str(confianca))
27          if idprevisto == idatual:
28              totalAcertos += 1
29              totalConfianca += confianca
30          #cv2.rectangle(imagemFaceNP, (x, y), (x + l, y + a), (0, 0, 255), 2)
31          #cv2.imshow("Face", imagemFaceNP)
32          #cv2.waitKey(1000)
33  percentualAcerto = (totalAcertos / 30) * 100
34  totalConfianca = totalConfianca / totalAcertos
35  print("----------------------------------------")
36  print("Total de acertos: " + str(totalAcertos))
37  print("Percentual de acerto: " + str(percentualAcerto))
38  print("Total confiança: " + str(totalConfianca))
```

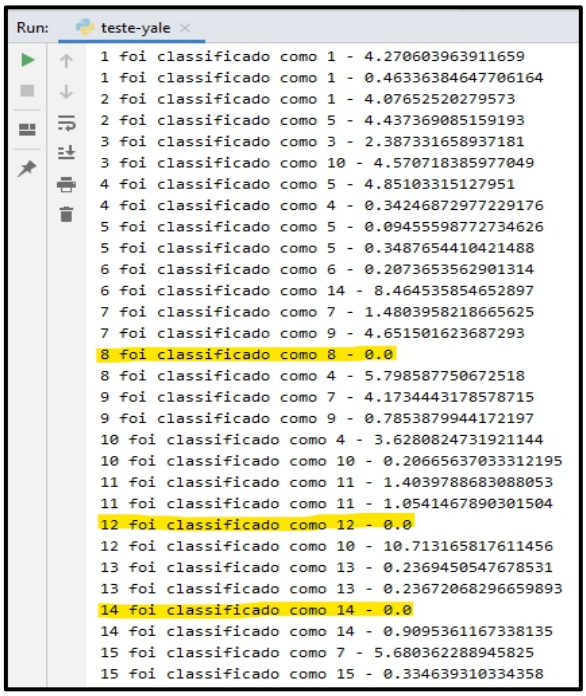
Fonte: Próprio Autor (2019)

O algoritmo visto na figura 32 também é muito semelhante com aquele visto na figura 26. Visto que o usamos como base, esse algoritmo tem como função calcular a média de confiança entre a imagem que está sendo capturada no momento e àquelas originadas a partir do treinamento. Além disso, teremos o total de acertos e sua porcentagem, dessa maneira escolheremos o melhor algoritmo e os melhores parâmetros de acordo com nossas necessidades.

Nas linhas 30 a 32, essa parte do código comentada é referente a um pequeno teste que realizamos para saber se as faces estavam sendo reconhecidas.

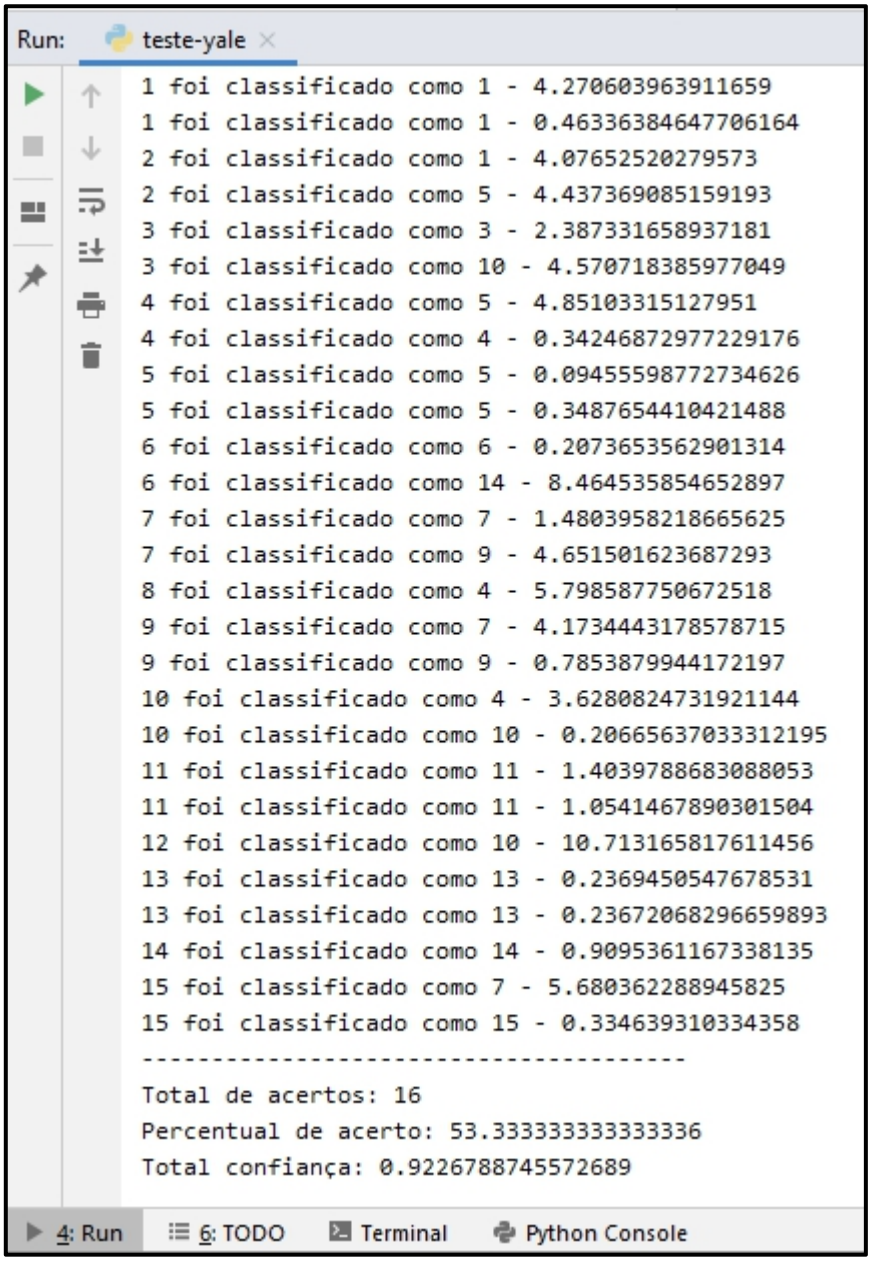
Após a execução dessa classe, tivemos o retorno de cada imagem ao qual o algoritmo escolhido (inicialmente, *Local Binary Patterns Histograms*) fez o reconhecimento e, além disso, dos resultados previstos. Veja na figura 33. Porém, esse processo nos trouxe alguns erros, como vemos nas linhas destacadas. O algoritmo reconheceu, com uma taxa de *0* (zero) erro, algumas imagens. E isso não pode acontecer, haja visto que caso isso ocorra é como se tivesse reconhecido a pessoa exatamente como nas imagens que passaram pelo treinamento. Isso é praticamente impossível. O erro estava na pasta utilizada para o reconhecimento – linha 18 da figura 32. Elas estavam idênticas com a da pasta de treinamento. Identificamos quais eram e eliminamo-las, para que não houvesse interferência nos resultados. Finalmente, conseguindo os resultados esperados (Figura 34).

Figura 33: Erro no reconhecimento facial. Imagens duplicadas.



Fonte: Próprio Autor (2019)

Figura 34: Resultados após conserto dos erros.



Fonte: Próprio Autor (2019)

Após todo esse processo de codificação de teste, montamos uma planilha simples no *Microsoft Office Excel 2010* mostrando uma pequena amostra dos resultados obtidos através de alguns testes. E com base nessas informações definimos o melhor algoritmo a ser usado e quais os melhores parâmetros.

Para cada rodada de teste, executamos as duas classes (Figura 31 e 32) novamente. Desse jeito, geramos arquivos de treinamento com os parâmetros novos e a classe de teste irá mostrar novos resultados. Colocamos como parâmetro padrão os vistos na figura 31, linhas 6, 7 e 8.

Tabela 1: Comparação de resultados do *Eigenfaces.*

Eigenfaces				
Nº do Teste	Total de Acertos	Porcentagem de Acertos	Média da Confiança	Parâmetros
1	15	50	3.571,88	Padrão
2	10	33,3	2.633,37	80, 4.000
3	11	36,66	1.154,18	10, 3.000
4	12	40	1.934,46	20, 5.000
5	11	36,66	2.278,70	40, 6.000

Fonte: Elaborado pelo Autor (2019)

Tabela 2: Comparação de resultados do *Fisherfaces.*

Fisherfaces				
Nº do Teste	Total de Acertos	Porcentagem de Acertos	Média da Confiança	Parâmetros
1	21	70	509,08	Padrão
2	24	80	1.297,31	10, 3.000
3	19	63,33	1.592,78	80, 4.000
4	19	63,33	1.592,78	30, 3.500
5	19	63,33	1.592,78	20, 6.000

Fonte: Elaborado pelo Autor (2019)

Tabela 3: Comparação de resultados do *Local Binary Patterns Histograms*.

Local Binary Patterns Histograms				
Nº do Teste	Total de Acertos	Porcentagem de Acertos	Média da Confiança	Parâmetros
1	16	53.33	0,92	Padrão
2	17	56,66	13,36	4, 4, 10, 10, 100
3	18	60	56,44	6, 6, 12, 12, 150
4	7	23,33	137,87	10, 10, 16, 16, 200
5	16	53,33	0,1	1, 1, 5, 5, 50

Fonte: Elaborado pelo Autor (2019)

Ao final dos testes, escolhemos o algoritmo de reconhecimento facial *Fisherfaces* com os parâmetros: número de componentes em 10 e o *threshold* em 3.000. A escolha foi feita a partir da porcentagem de acerto e a proximidade da imagem capturada atualmente com a do treinamento (Tabela 2). Pois, estes seriam os melhores parâmetros para o objetivo que queremos alcançar.

10. Conclusão

Em síntese, não pudemos demonstrar toda a capacidade do projeto por alguns obstáculos enfrentados no caminho. Todavia, podemos ter uma ideia de quão valioso poderia ser a implementação de um sistema, assim como foi estudado ao longo deste trabalho, em uma instituição de ensino superior. E, pensando mais ao longe, podendo ser utilizado em mais diversas áreas, como: reconhecimento de placas, de residentes de um condomínio, funcionários públicos e tantas outras.

Com o uso do reconhecimento facial, os discentes poderiam ter um melhor aproveitamento em relação às aulas. E, além do mais, com uma expansão do projeto em um momento futuro, poderiam ter um cadastro único dentro da universidade; podendo ter acesso a todas as áreas em comum, como: biblioteca, secretarias, dentre outras. Assim, aumentando o controle e segurança dentro do campus da universidade.

Podemos ter certa dificuldade para implementar esse tipo de sistema, como: registro de entrada e saída do discente, o limite de tempo do discente em sala de aula para registrar a presença e os equipamentos utilizados para implementação do sistema e para o reconhecimento facial (computadores e câmeras). Problemas esses que podem ser contornados com mais estudos nesse determinado tema e com uma maior disponibilidade de máquinas.

REFERÊNCIAS:

ARTERO, Almir Olivette. **Inteligência Artificial**: teórica e prática. São Paulo: Editora Livraria da Física, 2009.

ALVES, Ariane. **Simpatia é chave para lidar com humanos, diz criador de robô com cidadania:** Criações recentes valorizam abordagem amigável para acostumar humanos às máquinas. Disponível em: <https://exame.abril.com.br/tecnologia/simpatia-e-chave-para-lidar-com-humanos-diz-criador-de-robo-com-cidadania/>. Acesso em: 19 set. 2019.

AHONEN, Timo; HADID, Abdenour; PIETIKAINEN, Matti. Face description with local binary patterns: Application to face recognition. **IEEE Transactions on Pattern Analysis & Machine Intelligence**, n. 12, p. 2037-2041, 2006.

A.S.GEORGHIADES. **The Yale Face Database B.** 2001. Disponível em: <http://vision.ucsd.edu/~iskwak/ExtYaleDatabase/Yale%20Face%20Database.htm>. Acesso em: 4 nov. 2019.

BRADSKI, Gary; KAEHLER, Adrian. **Learning OpenCV: Computer vision with the OpenCV library**. " O'Reilly Media, Inc.", 2008.

BASILIO, PatrÍcia. **Aplicativo da Microsoft usa IA para localizar pessoas desaparecidas:** Ferramenta, lançada em parceria com a ONG paulistana Mães da Sé, utiliza reconhecimento facial para identificar pessoas em situação de suspeita de abandono. 2019. Disponível em: <https://epocanegocios.globo.com/Tecnologia/noticia/2019/05/aplicativo-da-microsoft-usa-ia-para-localizar-pessoas-desaparecidas.html>. Acesso em: 21 maio 2019.

CECCON, Denny. **Rede neural usa eletrocardiogramas para fazer predições inéditas de problemas de saúde.** 2019. Disponível em: <https://iaexpert.com.br/index.php/2019/08/30/rede-neural-usa-eletrocardiogramas-para-fazer-predicoes-ineditas-de-problemas-de-saude/?fbclid=IwAR1hjm2S4GFyAadFf4rzyQVhUSzsR6dvDrK9xy0NXsC_SLSIOVHli37C6EQ>. Acesso em: 30 ago. 2019.

CECCON, Denny. **Deep learning é usado para rastrear movimentos migratórios de pássaros.** 2019. Disponível em: <https://iaexpert.com.br/index.php/2019/09/02/deep-learning-e-usado-para-rastrear-movimentos-migratorios-de-passaros/?fbclid=IwAR0r97bERsgkXP4X0NGdy-qqNrG3ZopA41CDLc6g7KZ7znKucoCnvb1Fp5U>. Acesso em: 2 set. 2019.

COLDEWEY, Devin. **WTF is computer vision?** 2016. Disponível em: <https://techcrunch.com/2016/11/13/wtf-is-computer-vision/>. Acesso em: 2 set. 2019.

CARRARA, Valdemir. **Apostila de Robótica.** Disponível em: <http://www.logis.uff.br/~artur/AP/apostila_robotica.pdf>. Acesso em: 19 set. 2019.

DERTIEN, Edwin. **Realisation of an energy-efficient walking robot**. 2005. Dissertação de Mestrado. University of Twente.

FORTES, Charles Wellington de Oliveira. **Programação Orientada a Aspectos Aplicada.** Disponível em: <https://www.eclipse.org/lists/ajdt-dev/pdfEOArVJfjTm.pdf>. Acesso em: 23 set. 2019.

GIBBS, Samuel. **AlphaZero AI beats champion chess program after teaching itself in four hours:** Google's artificial intelligence sibling DeepMind repurposes Go-playing AI to conquer chess and shogi without aid of human knowledge. 2017. Disponível em: <https://www.theguardian.com/technology/2017/dec/07/alphazero-google-deepmind-ai-beats-champion-program-teaching-itself-to-play-four-hours>. Acesso em: 7 dez. 2017.

GOMES, Dennis dos Santos. Inteligência artificial: conceitos e aplicações. **Olhar Científico**, v. 1, n. 2, p. 234-246, 2011.

GALIPIENSO, María Isabel Alfonso et al. **Inteligencia artificial: modelos, técnicas y áreas de aplicación**. Editorial Paraninfo, 2003.

HADHAZY, Adam. **Até onde vai nossa capacidade de memória?** 2015. Disponível em: <https://www.bbc.com/portuguese/noticias/2015/04/150408_vert_fut_capacidade_cerebro_ml>. Acesso em: 8 abr. 2015.

HAYKIN, Simon. **Redes Neurais:** Princípios e práticas. 2. ed. Porto Alegre: Artmed Editora S.a., 1999. 898 p.

LAGANIÈRE, Robert. **OpenCV Computer Vision Application Programming Cookbook Second Edition**. Packt Publishing Ltd, 2014.

MACHADO, Vinicius Ponte. **Inteligência Artificial.** Disponível em: <https://www.ime.usp.br › ~slago › IA-introducao>. Acesso em: 11 nov. 2019.

MCCARTHY, John et al. A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955. **AI magazine**, v. 27, n. 4, p. 12-12, 2006.

ORTIN, Francisco et al. Including both static and dynamic typing in the same programming language. **IET software**, v. 4, n. 4, p. 268-282, 2010.

PASSOS, Alexandre Tachard et al. Combinatorial algorithms and linear programming for inference in natural language processing= Algoritmos combinatórios e de programação linear para inferência em processamento de linguagem natural. 2013.

RYLE, Gilbert. **The concept of mind.** Abingdon: Routledge, 2009. 314 p.

RASHED, Md Golam; AHSAN, Raquib. Python in computational science: applications and possibilities. **International Journal of Computer Applications**, v. 46, n. 20, p. 26-30, 2012.

SOUZA, Flávio Lima de. **Classificador Fisherface Fuzzy para o Reconhecimento de Faces.** 2014. 106 f. Dissertação (Mestrado) - Curso de Matemática, Instituto de Biociências, Letras e Ciências Exatas, Universidade Estadual Paulista, S˜ao Jos´e do Rio Preto, 2014. Cap. 2.

SANTOS, Rafael. Introdução à programação orientada a objetos usando Java. Elsevier, 2003.

SILVEIRA, José Atílio Pires da et al. **Inteligência artificial: um perguntar pelo homem?**. 2018.

SIROVICH, Lawrence; KIRBY, Michael. **Low-dimensional procedure for the characterization of human faces**. Josa a, v. 4, n. 3, p. 519-524, 1987.

TUCKER, Allen; NOONAN, Robert. **Linguagens de Programação-: Princípios e Paradigmas**. AMGH Editora, 2009.

TURK, Matthew A.; PENTLAND, Alex P. Face recognition using eigenfaces. In: **Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition**. IEEE, 1991. p. 586-591.