UNIVERSIDADE DE SOROCABA
PRÓ-REITORIA DE GRADUAÇÃO E ASSUNTOS ESTUDANTIS
CURSO DE ENGENHARIA DA COMPUTAÇÃO

Felipe Augusto de Almeida

SISTEMA DE RECONHECIMENTO DE EXPRESSÕES FACIAIS

Sorocaba/SP
2020

Felipe Augusto de Almeida

SISTEMA DE RECONHECIMENTO DE EXPRESSÕES FACIAIS

Trabalho de Conclusão de Curso apresentado como exigência parcial para obtenção do Diploma de Graduação em Engenharia da Computação, da Universidade de Sorocaba.

Orientador: Prof. Me. Bruno Aguilar da Cunha

Sorocaba/SP
2020

Felipe Augusto de Almeida

SISTEMA DE RECONHECIMENTO DE EXPRESSÕES FACIAIS

Trabalho de Conclusão de Curso apresentado como exigência parcial para obtenção do Diploma de Graduação em Engenharia da Computação, da Universidade de Sorocaba.

Aprovado em: ____/____/_____

BANCA EXAMINADORA:

Prof. Me. Bruno Aguilar da Cunha
Universidade de Sorocaba

Prof.(a) Nome Completo do(a) Examinador(a)
Instituição a que ele(a) pertence

Prof.(a) Dr.(a) Nome Completo do(a) Examinador(a)
Instituição a que ele(a) pertence

Dedico a conclusão deste trabalho aos meus familiares que se esforçaram tanto quanto eu para conclusão deste curso, ao meu orientador Bruno Aguilar da Cunha sem o qual não seria possível a finalização desta tarefa.

AGRADECIMENTOS

Agradeço primeiramente a minha família pelo apoio e incentivo incondicional.

Aos meus colegas de sala e professores por compartilharem seus conhecimentos.

Especial agradecimento a duas pessoas que me deram apoio nos momentos em que eu mais me sentia desamparado: minha irmã de consideração Júlia Cavalcante da Silva, que foi a base da minha força de vontade e o Dr. Deusvenir de Souza Carvalho que me incentivou desde a tomada de decisão da temática deste trabalho.

As pessoas supra citadas foram de suma importância para o decorrer deste projeto e, consequentemente, para o meu crescimento pessoal.

“This life is like a swimming pool. You dive into the water, but you can’t see how deep it is.” – Dennis Rodman

RESUMO

No decorrer dos anos a evolução na área da leitura e análise das expressões faciais ficou mais precisa e utilizada nas mais diversas áreas como jurídica, cinematográfica e computação devido a sua capacidade de compreender e se aproximar aos sentimentos humanos. Por meio do estudo das microexpressões foi utilizada neste trabalho uma implementação pronta e desenvolvida utilizando redes neurais artificiais (RNAs) para a realização de testes e verificação da capacidade de classificação das expressões faciais em situações diversas. A RNA convolucional utilizada na implementação foi treinada com o *dataset* FER2013 que contém o total de 35887 imagens de expressões faciais de pessoas, sendo elas divididas em 28709 imagens de treino e 7178 de validação. As imagens de treino foram divididas em 3995 de raiva, 436 de nojo, 4097 de medo, 7215 de felicidade, 4965 de neutralidade, 4830 de tristeza e 3171 de surpresa. As imagens de validação divididas em 958 de raiva, 111 de nojo, 1024 de medo, 1774 de felicidade, 1233 de neutralidade, 1247 de tristeza e 831 de surpresa. Os resultados obtidos foram promissores em ambiente bem iluminado e com um bom enquadramento do usuário, apesar das dificuldades de classificação em ambientes escuros e com o rosto do usuário obstruído ou com algum adereço. No geral a abordagem analisada foi capaz de identificar as expressões faciais com uma precisão de 61.09% com potencial de melhora, por meio de ajustes e a realização de mais testes.

Palavras-chave: Expressões. Leitura facial. Python. Keras. Tensorflow. OpenCV. Redes Neurais Convolucionais.

ABSTRACT

During years, the evolution in the field of reading and analysis of facial expressions get accurate and used in the most different areas like legal, cinematic and computation according to its capacity of understand and get closer to the human feelings. Through the study of micro expressions was used an artificial neural network (ANNs) to a fulfillment of test and verification of capacity from classification of the expressions in different situations. The convolutional ANN in this implementation was trained with the dataset FER2013 that include an amount of 35887 images from people facial expressions being them sorted by 28709 images from training and 7178 from validation. Being the images of training sorted in 3995 from angry, 436 from disgust, 4097 from fear, 7215 from happiness, 4965 from neutral, 4830 from sadness and 3171 from surprise, and the images of validation sorted in 958 from angry, 111 from disgust, 1027 from fear, 1774, from happiness, 1233 from neutral, 1247 from sadness and 831 from surprise. The obtained results were promising in a good lighting environment and with a well framed user, besides the difficulties of classification in dark environments and with the users face blocked or with some adornment. In general, the approach analyzed was capable to identify facial expressions with a accuracy of 61,09% with a potential of improvement, by adjustments and fulfillment of more tests.

Keywords: Expressions. Face reading. Python. Keras. Tensorflow. OpenCV. Artificial Neural Network.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Muito se debate sobre a forma mais eficaz de reconhecer as expressões faciais. Os especialistas em ciências comportamentais afirmam que o método mais efetivo é o reconhecimento em tempo real feito por um estudioso da área, já que há diferença entre a velocidade e precisão de reconhecimento sendo ele feito por máquina ou humano (MEYER, 2011). Dessa forma, a utilização de softwares para leitura de expressões faciais é de grande valia para os especialistas, já que pode dar grande suporte para trabalhos que necessitam de provas exatas para serem executados, como um processo judicial e interrogatório, por exemplo.

Com o avanço das tecnologias foi possível fazer com que softwares reconhecessem as expressões faciais de forma rápida. Sendo assim é possível com uma linguagem de programação fazer um dispositivo de captura reconhecer o movimento facial e categorizá-lo referente à expressão facial apresentada atualmente.

Para reconhecer uma expressão facial é necessário, primeiramente, entender de onde as manifestações de raiva, felicidade, nojo, tristeza, desprezo, surpresa e raiva que serão posteriormente avaliadas ao desenvolver desse projeto.

O desenvolvimento de um sistema de reconhecimento facial a partir de um sistema simples e um dispositivo de captura de imagens e vídeos é uma alternativa para um reconhecimento mais rápido e eficaz para organizações que necessitem de uma apresentação cientifica com dados exatos e precisem de portabilidade e não exija elevados custos para aplicação. Além disso, tal projeto também visa aprofundamento nos conhecimentos da linguagem Python tal como suas bibliotecas OpenCV, tensorflow e keras, o que será explanado nesse projeto.

Ou seja, o sistema apresentado nesse trabalho tem como objetivo implementar uma aplicação que possa reconhecer a face e suas expressões a partir de um código já existente, será utilizada a biblioteca TensorFlow juntamente com sua API Keras, responsável por redes neurais permitindo assim a utilização de *deep learning* para que a máquina leia e interprete um banco de dados de expressões faciais. A situação será testada e analisada implementando assim melhorias que permitam a verificação da capacidade do algoritmo de classificação. A utilização do Keras foi escolhida devido à praticidade e eficiência em compilar redes neurais combinando camadas de diferentes dimensões e funções de ativação (MODELO... 2018). Após a utilização do

Keras, usou-se o OpenCV para inicialização da câmera para leitura da expressão facial e indicação da mesma.

O objetivo principal deste trabalho é realizar estudos sobre expressões faciais para compreender os diferentes sentimentos que podem ser expressos na face de uma pessoa e fazer testes, análises e modificações em uma implementação a ser utilizada para classificar automaticamente expressões faciais com o uso de redes neurais artificiais (RNAs) convolucionais. Para isso, os objetivos específicos do projeto são o estudo e compreensão do protocolo Facial Action Coding System (FACS) para verificar o desempenho do sistema de reconhecimento de microexpressões, estudo, compreensão e utilização da linguagem Python por meio das bibliotecas TensorFlow, Keras e OpenCV para realizar modificações que sejam necessárias e busquem facilitar a análise dos resultados obtidos pela RNA e implementar melhorias pontuais que sejam necessárias e identificadas para a correta execução do código fonte.

Essa monografia está organizada por capítulos aos quais o leitor poderá encontrar: Introdução: breve explicação sobre a aplicação do sistema de reconhecimento facial, suas vantagens e tecnologias envolvidas. Juntamente com os objetivos e motivação do trabalho. Desenvolvimento: abordagem teórica dos tópicos apresentados anteriormente e explanação de etapas e conceitos. Implementação: descrição dos passos de programação e aplicação do sistema. Resultados: os resultados obtidos no capítulo 3 serão explanados neste., e por fim as conclusões obtidas neste projeto são apresentadas neste capítulo.

2 REFERENCIAL TEÓRICO

Para o desenvolvimento deste trabalho foram necessários conhecimentos de mais de uma área de conhecimento e estudo de tecnologias e suas peculiaridades. Nas seções a seguir, serão apresentados os conhecimentos adquiridos e fundamentos teóricos utilizados no decorrer deste trabalho para a análise e melhoramento do sistema de reconhecimento de expressões faciais.

2.1 Estudo das expressões faciais

Para iniciar os estudos das expressões faciais foram seguidos os princípios apresentados pelos autores Paul Ekman e Wallace Friesen que afirmam que “Os temas principais – as plantas azuis da expressão facial – são universais” (EKMAN E FRIESEN, 2003) sendo assim o estudo da base das expressões abrangeria toda a espécie humana independente da cultura, criação, raça ou gênero. Foi necessário também o estudo mais profundo sobre os principais problemas da leitura facial.

O maior deles é o fato que o rosto provê mais de um tipo de sinal para gerar uma expressão o que pode levar a um erro de classificação caso verifique a combinação de sinais de forma errada como é explicado por Paul Ekman e Wallace Friesen,

> A face provê mais de um sinal para transmitir mais de um tipo de mensagem. Na tentativa de seguir as mensagens das emoções, você pode olhar ao sinal errado. Ou talvez você não diferencie claramente a mensagem da emoção de outras mensagens transmitidas pelo rosto (EKMAN E FRIESEN, 2003).

Além das expressões faciais foi feito também um estudo sobre as microexpressões que será explanado posteriormente. Cabe reforçar que é necessário o conhecimento sobre as expressões faciais para entender como funciona a musculatura e as ações do rosto humano.

2.1.1 Microexpressões

Um estudo das microexpressões para melhor entendimento das musculaturas e Action Units (AUs) para demonstração de um sentimento foi realizado. As expressões faciais ocorrem por cerca de 1 segundo, enquanto as microexpressões acontecem em um período de 1/5 à 1/25 segundos (EKAMAN E FRIESEN, 2003).

Essas expressões têm suma importância para definir as emoções de um humano em tempo real por serem as expressões que transparecem os detalhes cruciais para a leitura e análise das emoções.

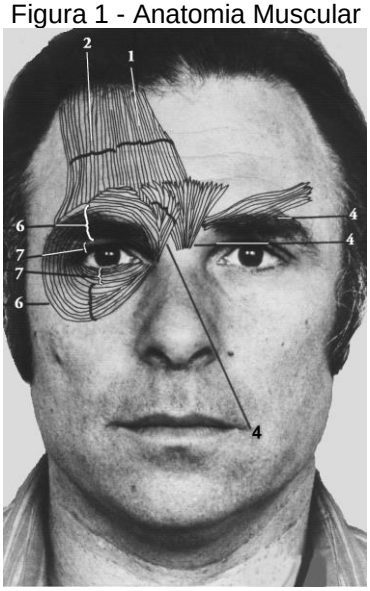
Um humano treinado consegue ver a mudança facial e ler as microexpressões, portanto, partindo deste princípio, a pesquisa tem como base analisar a capacidade do código de ler a emoções e verificar sua precisão. As microexpressões são expressões faciais de menor duração que ocorrem em pontos mapeados por protocolos como o *Facial Action Coding System* (FACS) (EKMAN et al., 2002).

2.1.2 Manual FACS

O manual FACS é o protocolo utilizado para mapear o rosto e sua musculatura nomeada como (AUs) que representam a atividade muscular do rosto no momento de uma microexpressão (EKMAN et al., 2002). Para a verificação das AUs é necessário saber a área de movimentação dessas musculaturas que, ao entrarem em funcionamento, traçam as Unidades de Ação que permitem a leitura facial.

O uso do FACS para o estudo das expressões permitiu que fosse feita e praticada a análise das mesmas e comparados os resultados do software com o do analista. Primeiramente foi feita a análise da parte muscular que demonstra a maior alteração notada durante uma expressão, Figura 1, tal área pode ser analisada na que consta os músculos da região dos olhos que se enrugam ou enrijecem durante uma expressão.

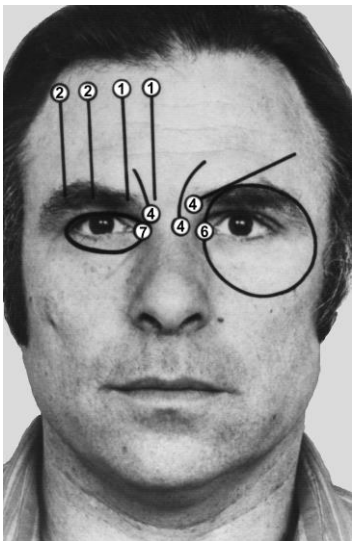
Figura 1 - Anatomia Muscular



Fonte: EKMAN, ET AL., 2002

Após o estudo dos músculos e nervos foram analisadas as áreas que se movimentam e seus vetores após uma emoção como pode ser visto na Figura 2. Na raiva, por exemplo, há uma tensão na testa e aproximação das sobrancelhas.

Figura 2 - Ação Muscular



Fonte: EKMAN, ET AL., 2002

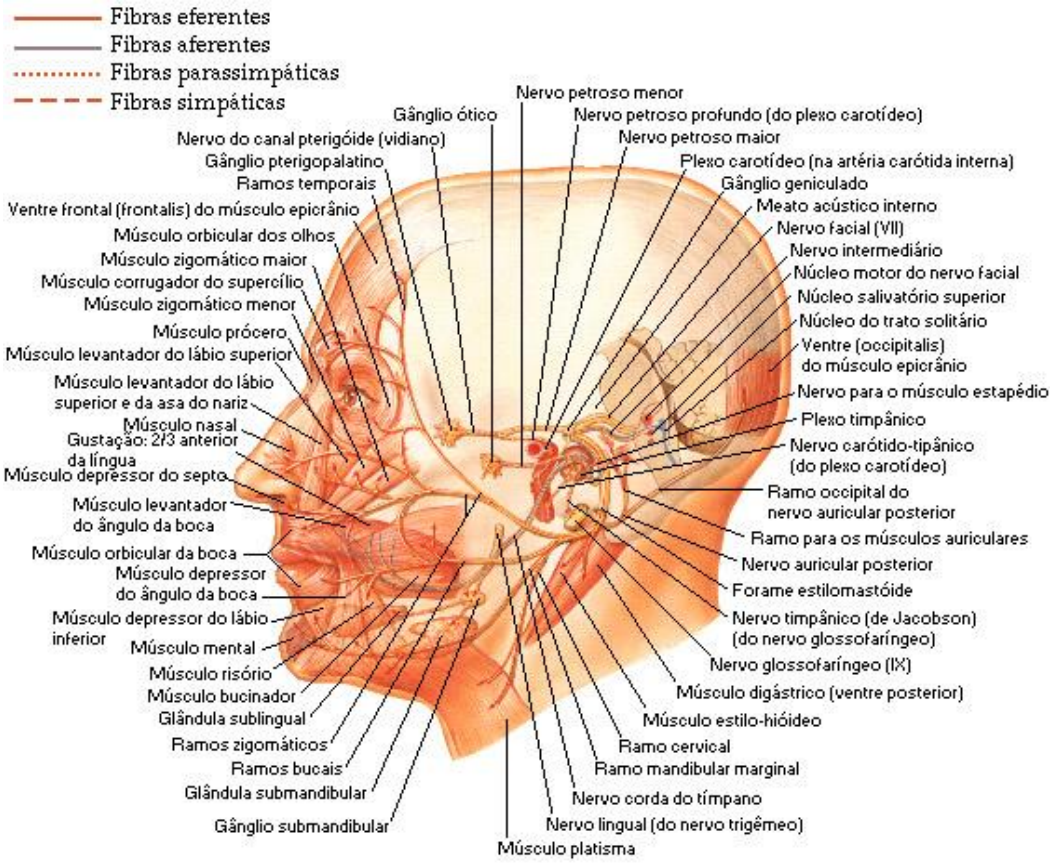
Os nervos e músculos agem de acordo com comandos enviados do nosso cérebro e o nervo responsável pelos músculos da cabeça e pescoço, o NC7 ou Nervo Facial.

2.1.3 NC7

O NC7(Nervo Craniano 7), também conhecido como Nervo Facial é o responsável por ramificar a musculatura facial. O conjunto de contrações na face com o tempo inferior a 2 segundos pode ser considerado uma micro expressão facial, a qual é categorizada dependendo da combinação de nervos e contrações faciais (Figura 3) notadas no momento da detecção.

As combinações especificas da ativação dos nervos e contrações das musculaturas dão origem a 7 emoções primárias: raiva, nojo, tristeza, desprezo, felicidade, surpresa e medo.

Figura 3 - Nervo Craniano 7 - Nervo Facial

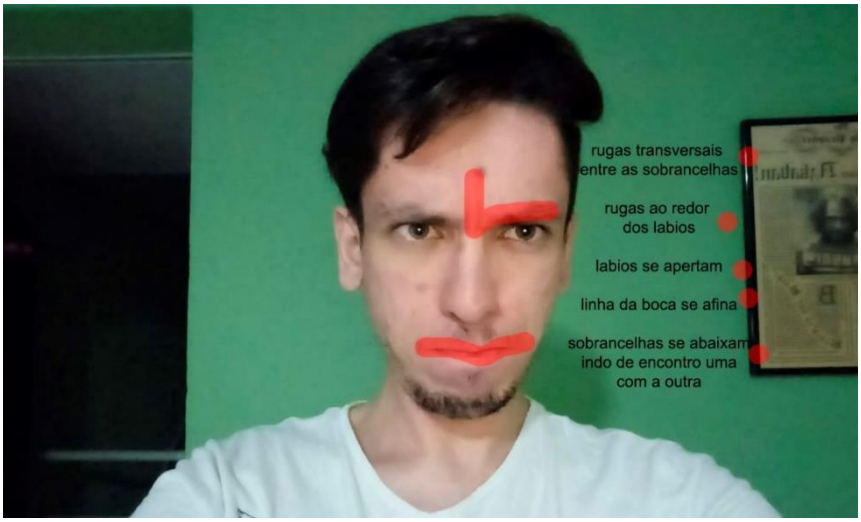


Fonte: NETTER, Frank H.. Atlas de Anatomia Humana. 2ed. Porto Alegre: Artmed, 2000.

2.1.4 Raiva

A raiva gera o impulso emocional quando sentimos que temos que ser agressivos para proteção própria ou de alguém próximo.

Figura 4 - Expressão de Raiva



Fonte: Elaboração própria..

2.1.5 Nojo

O sentimento e expressão de nojo tem como função preservar nosso organismo ante estímulos tóxicos que possam degradar nosso corpo ou mente. Ao sentir nojo de uma pessoa, o instinto inicial é desconsiderá-la como humana e sim como objeto podre ou tóxico que nos causaria danos psicológicos

Figura 5 - Expressão de Nojo

Fonte: Elaboração própria..

2.1.6 Tristeza

As expressões de tristeza, apesar do que se diz popularmente, não tem como função apenas demonstrar que está se sentindo mal devido à alguma situação ou ocorrência. A tristeza tem como função buscar amparo e aproximação social.

Figura 6 - Expressão de Tristeza

Fonte: Elaboração própria.

2.1.6 Desprezo

O desprezo é um sentimento extremamente perigoso para relações sociais pois está correlacionado a pessoas hostis e perigosas que se sentem no direito de tratar outros como objetos ou ferramentas para uso próprio. Nos categoriza como superior a algo ou alguém como inferior, seja material ou psicologicamente.

Figura 7 - Expressão de Desprezo

Fonte: Elaboração própria..

2.1.8 Felicidade

O sentimento de felicidade está relacionado a tudo que nos é prazeroso (êxtase, comer bem, conquistas, entre outros) visando tanto o prazer quanto o bem estar social.

Figura 8 - Expressão de Felicidade

Fonte: Elaboração própria..

2.1.9 Surpresa

A surpresa é a emoção de duração mais breve seguida sequencialmente de outra emoção que durará no máximo 1 segundo.

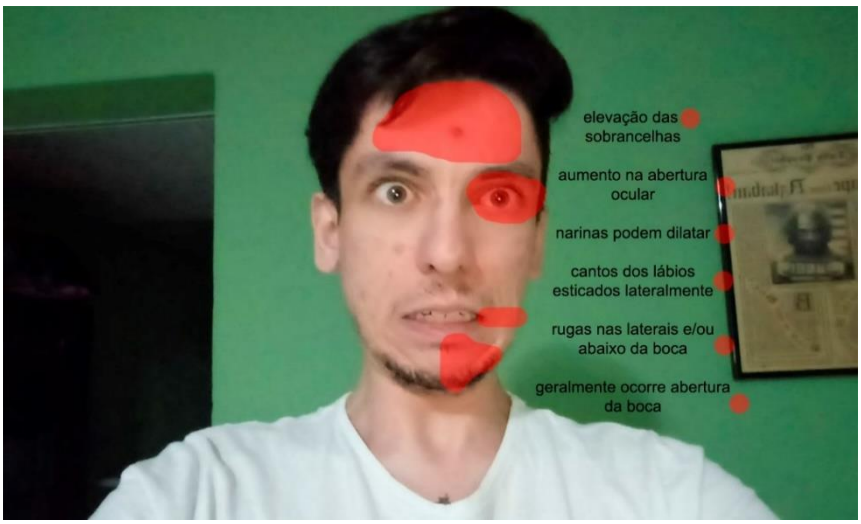
Figura 9 - Expressão de Surpresa

Fonte: Elaboração própria.

2.1.10 Medo

A emoção de medo é associada à preservação do nosso corpo perante danos iminentes. É a única emoção que tem três possibilidades de reação: Congelar, lutar ou fugir. Conhecido pelo termo FFF (Freeze, fight and flight).

Figura 10 - Expressão de Medo



Fonte: Elaboração própria.

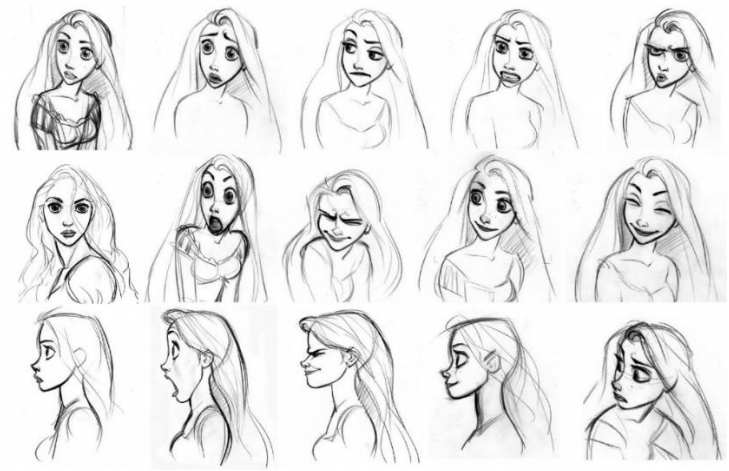
As mesmas emoções explanadas nos tópicos anteriores tiveram as imagens baseadas no trabalho de Paul Ekman.

Para transmitir tais emoções para um software foi necessária a escolha da linguagem de programação flexível, prática e com boa precisão. Sendo assim, a linguagem Python foi decidida a mais apropriada devido às bibliotecas que a permitem uma boa análise referente à face humana com a flexibilidade procurada.

2.1.11 Aplicações usando análise de sentimentos

Uma das aplicações citadas pelo autor Paul Ekman em seu *website* é a utilização dos estudos das emoções nas animações e desenhos das empresas Disney e Pixar (PAUL, 2004). O estudo das emoções e análise dos sentimentos permitem que as expressões dos personagens sejam mais reais e causem empatia no espectador (EXPRESSAMENTE, 2014), podemos ver na Figura 11 que após a revisão das emoções os personagens se tornam mais “humanos”.

Figura 11 - Emoções Disney



Fonte:EXPRESSAMENTE, 2014

Outra área que, a partir dos estudos de Detecção de Mentiras do Paul Ekman, se beneficiou foi a jurídica. Após o lançamento do livro Telling Lies, órgãos como CIA, FBI e Scotland Yard solicitaram treinamento para reforçar a eficiência de atuação com a implementação da análise de sentimentos e leitura facial (PAUL, 2004).

Houve também a compra de uma empresa por parte da Apple que fará uso das micro e macro expressões para aumentar a eficácia da assistente pessoal Siri que utilizaria as expressões faciais para identificar as emoções do usuário e aumentar a qualidade da experiência de socialização com o aplicativo de assistência (NOYES, 2016).

Cada vez mais as empresas de tecnologia tentam deixar seus produtos mais adaptados para parecerem mais “humanizados”. O estudo das emoções foi um grande passo para a indústria tecnológica adicionar preocupação emocional aos seus produtos. Até mesmo aplicativos desenvolvidos apenas para diversão como o criador de avatares do Facebook utilizam da análise emocional para criar algo que envolva o usuário de forma mais profunda (ELGAN, 2017).

2.2 Python

A linguagem de programação utilizada para desenvolver a aplicação utilizada neste projeto foi o Python devido à sua versatilidade e objetividade na hora de se projetar.

Existem diversas bibliotecas para serem exploradas sendo algumas delas especificas para reconhecimento de imagens ou de Machine Learning (PYTHON, 2001) o que agiliza o processo de produção e diminui os custos.

A linguagem Python não tem área específica de aplicação pelo fato de poder ser aplicado de acordo com o propósito desejado, sendo assim, não existe uma tarefa específica que a linguagem deve suprir pois seu objetivo é simplesmente geral (SILVA, 2019).

Python pode ser caracterizada por ser uma linguagem simples voltada para o desenvolvimento rápido, seguindo certas metodologias como RAD (Rapid Application Development) a qual foca na diminuição de desperdícios, tendo o código assim uma qualidade maior, com menos tempo de produção e menor custo. A linguagem também utiliza da técnica DRY (Don’t Repeat Yourself) onde a parte de codificação não deve conter repetições ou ambiguidades. Segue também o princípio KISS (Keep It Simple, Stupid) onde o objetivo é manter o sistema simples pois quanto mais simples mais rápido e eficiente ele será (SILVA, 2019).

Sendo uma linguagem de alto nível muitos dos componentes são escritos com a própria palavra como os comandos lógicos E, OU, IGUAL ou DIFERENTE. Nas linguagens tradicionais como C, C#, Java e outras são usados os comandos “&&, | |, ==, !=”. Porém com o Python é diferente, os mesmos comandos são escritos com as próprias palavras, respectivamente, “and, or, is, is not” (SILVA, 2019).

Outras das características do “corpo” do código em Python é que o mesmo não utiliza o token “;” para finalizar uma linha de código pois o próprio é finalizado

utilizando, de forma implícita, o caractere “\0” (SILVA, 2019), o qual não necessita ser digitado pois é o caractere final de toda e qualquer linha de código.

O funcionamento da linguagem pode ser encontrado nos mais diversos lugares como relógios, celulares, TVs e outros aparelhos eletrônicos. A interpretação em Python funciona de tal forma: os arquivos com a extensão .py codificados pelo programador são compilados e geram um arquivo de formato .pyc, esses quais podem ser interpretados por uma Máquina Virtual Python se tornando assim, um código nativo de máquina (SILVA, 2019).

2.2.1 Tensorflow e Keras

Como visto anteriormente o manual FACS foi de grande ajuda para entender a musculatura e nervos do rosto e suas variações de movimentos ao exibir uma emoção.

A utilização do Tensorflow como biblioteca foi escolhida devido à sua praticidade com redes neurais artificiais que é o que mais se pareava com o sistema planejado, no qual se utilizaria da biblioteca e uma de suas APIs para utilizar um banco de dados de domínio público para criar uma máquina treinada para ler expressões faciais e identifique a emoção apresentada.

Ele cria modelos de machine learning com facilidade (TENSORFLOW, 2015), o que permitiu que o dataset fosse utilizado para criação da rede neural que aprendeu os padrões das imagens e suas emoções, treinou a máquina para verificar se as mesmas conferem com a expressão demonstrada pelo usuário e mostrá-la na interface provida pelo OpenCV.

Seu desempenho se dá pelo fato de utilizar XLA, um compilador de álgebra linear que faz com que a execução seja bem mais rápida, podendo rodar em CPUs, GPUs, TPUs e outros mais (TENSORFLOW, 2015).

Várias empresas utilizam o TensorFlow assim como a Intel, Uber, DropBox e Google, que usa e incentiva o uso do framework. Pode ser aplicado em diversas situações devido à sua abstração, permitindo ao programador poder focar mais na lógica geral do que com os detalhes básicos.

A flexibilidade obtida também possibilitou o desenvolvimento e a implementação de melhorias do código fonte utilizado nesse trabalho afinal as APIs dão suporte ao usuário para que o projeto possa usar o melhor de suas tecnologias. Nesta monografia veremos também o uso do Keras, uma das APIs dispostas pelo TensorFlow.

Keras é uma API para Python usada na biblioteca Tensorflow que foi escolhida devido à sua praticidade e eficiência ao criar Redes Neurais.

Como citado no site da própria biblioteca: Keras é uma API designada a seres humanos, não máquinas. Keras segue a melhor prática para redução de carga cognitiva: oferece uma consistente e simples API, minimiza o número de ações requeridas pelo usuário para casos de uso comum, e provê claro e acionáveis mensagens de erro (KERAS, 2015).

Uma das maiores vantagens de ter se utilizado essa API foi sua interface simples e otimizada, o que facilitou o uso do modelo de redes neurais disposto nesse trabalho.

Para fazer a análise das expressões é necessária uma quantidade considerável de combinações que são facilmente lidas pela API e interpretadas a ponto de que as expressões no banco de dados possam ser comparadas com a expressão lida em imagens ou frames analisados

A utilização do Keras foi útil para que o projeto pudesse rodar de forma rápida, porém eficiente fazendo assim com que a leitura e interpretação fosse feita em tempo real.

2.2.2 OpenCV

O uso do OpenCV na implementação analisada neste trabalho deu-se para a finalização do software para que ele ative a webcam que será o dispositivo responsável pela captura da face para análise apresentando as expressões identificadas claramente ao usuário.

O OpenCV é uma biblioteca disponível para Pyhton que tem como função prover uma infraestrutura comum para computadores com aplicações visuais (OPENCV, 1999), no caso, a interface e amostra do rosto analisado pela webcam e as expressões de forma explicita para total entendimento do processo de verificação das emoções no rosto.

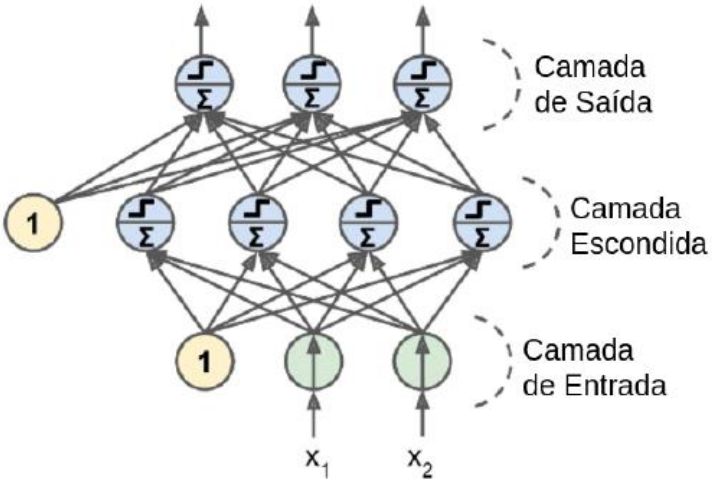
A biblioteca foi escolhida pensando na combinação das bibliotecas anteriores. O OpenCV foi uma peça fundamental que fez com que o funcionamento das outras bibliotecas pudessem ser visto pelo usuário.

2.3 Redes Neurais Artificiais

As Redes Neurais Artificiais (RNA) foram inspiradas no método de funcionamento do cérebro dos mamíferos. O foco das redes neurais é construir máquinas mais inteligentes (GÉRON, 2017).

O *perceptron* é o componente mais simples de uma rede neural, conhecido também como neurônio. Sendo inspirado no neurônio biológico, detêm conexões de entrada e saída para conectar-se com outros neurônios para, assim, enviar sinais e informações. Utilizando a função de ativação é possível determinar os parâmetros de sinal de entrada. Determinados os parâmetros e utilizada a ativação, é gerado um sinal de saída que é utilizado como sinal de entrada para outros neurônios (GÉRON, 2017).

Figura 12 - Rede Neural Artificial



Fonte: (GÉRON, 2017).

A Figura 12 ilustra uma rede neural multicamadas, tal rede neural possui uma camada de entrada, diversas camadas intermediárias e uma camada de saída. Por serem compostas por neurônios *perceptron,* a camada de entrada recebe os dados e os encaminha para a próxima camada. A cama escondida tem como propriedade ser inteiramente conectada, sendo assim todo e qualquer neurônio desta camada conecta-se como todos da camada anterior e da próxima. Por fim, a camada de saída é a responsável para fornecer o resultado da RNA, sendo assim a quantidade de neurônios da camada de saída é a mesma das classes do problema (GÉRON, 2017).

Sabendo disso, pode-se dizer que se o problema for o reconhecimento das emoções primárias, inclusa a neutra, a camada de saída teria o total de sete neurônios.

2.3.1 Rede Neural de Convolução

A Rede Neural de Convolução (RNC) é uma das principais técnicas utilizadas neste trabalho. Foi necessário, inicialmente, uma etapa de treinamento na qual as imagens são processadas com o propósito de extrair os padrões e relevâncias que diferenciam uma emoção da outra.

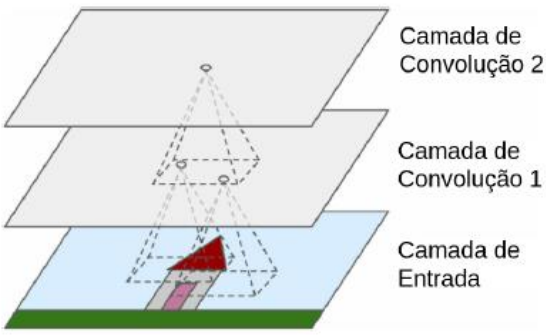
Após o treinamento é criado um modelo feito para processar expressões faciais capaz de ler uma imagem tridimensional e transformá-la em um vetor unidimensional contendo as informações que fazem dela uma imagem representativa da emoção específica.

O modelo contém uma camada final de decisão utilizada para classificação chamada Softmax. Tal camada é padrão em modelos de RNC, a mesma recebe o vetor unidimensional para fins de decisão, classificando assim a emoção da imagem lida. Categorizando assim, o modelo como como um classificador.

2.3.2 Camada de Convolução

A camada de Convolução é a parte mais importante de uma RNC. Convolução consiste em uma operação matemática cuja função faz um deslizamento sobre outra calculando a integral da soma do produto delas a partir da sobreposição adquirida à partir do deslizamento.

Figura 13 - Camada de Convolução



Fonte: (GÉRON, 2017)

Cada camada de convolução recebe informações em seu campo de visão a partir de uma imagem de entrada, como podemos ver na Figura 13

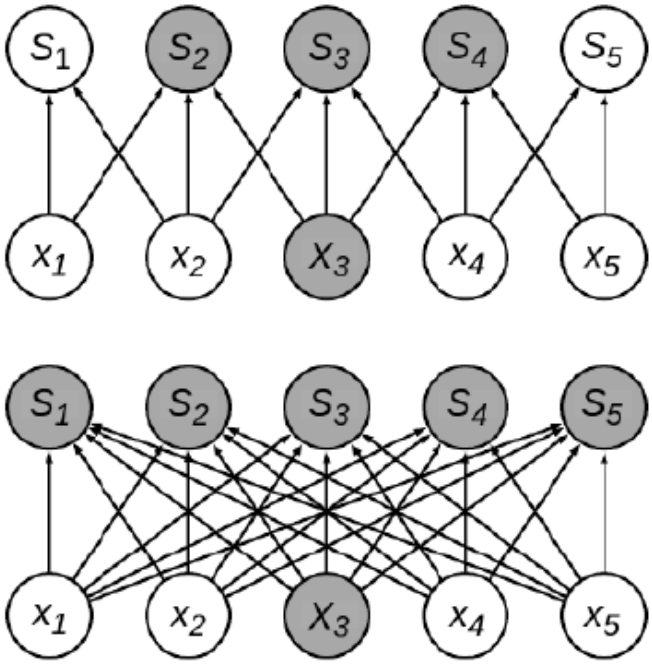
Diferente da metodologia regular de um neurônio, o mesmo da camada de convolução 1 é conectado a apenas alguns pixels da imagem, sendo assim, apenas uma sub-região da imagem é processada pelo neurônio da camada de convolução.

Os neurônios da camada de convolução 2 estão conectados com um pequeno campo de visual da camada 1 também diferente da *perceptron* (GÉRON, 2017).

As vantagens de se usar a RNC são apresentadas a seguir:

i. Devido à conectividade esparsa, como demonstrado na Figura 14, a RNC possui menor quantidade de requer menos tempo e recursos devido a menor quantidade de parâmetros a serem treinados (GOODFELLOW et al., 2016);
ii. A RNC já é comumente utilizada no dia a dia sendo usada, por exemplo, pelo córtex visual. Devido a tal inspiração, a funcionalidade da RNC é bem propicia para o reconhecimento de imagens (GÉRON, 2017);
iii. O compartilhamento de parâmetros entre os neurônios resulta num aprendizado em conjunto (GOODFELLOW, 2016).

Figura 14 - Conectividade Esparsa



Fonte: (GOODFELLOW et al., 2016)

2.3.3 Funções de Ativação

Há também a possibilidade de, em um modelo de rede, ser incluída uma não-linearidade em sua saída, a mesma necessita ser reduzida. A função de ativação tem como objetivo representar o efeito que a entrada interna e o estado atual de ativação exercem na definição do próximo estado de ativação da unidade (FLECK, Leandro et. al., 2016). Existem diversos tipos de funções de ativação, sendo as mais populares (HAYKIN, 2001):

Função limiar, sendo essa a que normalmente restringe a saída da RNA em valores binários (0 e 1). A saída do neurônio é considerada 0 quando resultado negativo e 1 quando positivo, sendo assim representada pela equação 1.

$$f(u) = \begin{cases} 1 & se \quad u \geq 0 \\ 0 & se \quad u < 0 \end{cases} \tag{1}$$

Há também a função linear por partes, que pode ser considerada uma aproximação de um amplificador não-linear que pode ser representada pela equação 2.

$$f(u) = \begin{cases} 1 & se \quad u \geq +1/2 \\ u & se +1/2 > u > -1/2 \\ 0 & se \quad u \leq -1/2 \end{cases} \tag{2}$$

Na linear por partes, como é dito Leandro Fleck (2016),

> “As seguintes situações podem ser vistas como formas especiais da função linear por partes: a) se a região linear de operação é mantida sem entrar em saturação, surge um combinador linear; b) a função linear por partes se reduz à função limiar, se o fator de amplificação da região linear é infinitamente grande”.

Tem-se também a função sigmoidal, sendo essa a mais comum e definida por ser uma função crescente com um bom balanceamento referente ao comportamento linear e o não-linear, a mesma assume um intervalo de variação entre 0 e 1 (FLECK, Leandro et. al., 2016). A função sigmoidal pode ser representada pela equação 3.

$$f(\mu) = \frac{1}{1 + \exp(-a\mu)} \tag{3}$$

Sendo o *α* o parâmetro de inclinação da função sigmóide, ou seja, quanto maior o valor de *α,* maior a inclinação da curva (FLECK, Leandro et. al., 2016).

Por último, a função tangente hiperbólica é a função utilizada para, em diversos casos, substituir as funções logísticas pelo mesmo poder assumir valores positivos e negativos, enquanto as logísticas funcionam apenas no intervalo entre 0 e 1, para a representação da função tangente hiperbólica temos a equação 4.

$$f(u) = \tanh(u) \tag{4}$$

O fato de poder assumir valores negativos pode ser muito benéfico, analiticamente falando, em certos casos.

2.3.4 Aprendizado Supervisionado

Nesse método de aprendizado é fornecido um objetivo a ser alcançado pelo treinamento, sendo assim, o treinamento já tem o conhecimento do ambiente.

O treinamento consiste em conjuntos de exemplos com entradas e saídas. O algoritmo de aprendizado de máquina adquiri o conhecimento a partir desses dados passados pelos exemplos. O objetivo de tal aprendizado é que a máquina seja capaz de criar saídas para entradas ainda não apresentadas, ou seja, a partir do conhecimento apresentado à máquina a mesma se torna capaz de ler outras situações do mesmo caso (MITCHELL, 1997).

O erro de um aprendizado supervisionado pode ser calculado subtraindo o valor da saída desejada e saída gerada como é apresentado na equação 5 (MITCHELL, 1997).

$$e_k = d_k - y_k \tag{5}$$

Sendo que “k” é o estímulo, “e” representa sinal de erro, “d” é a saída desejada e “y” é a saída obtida após o estímulo de entrada.

2.3.5 Preparação de Dados

A preparação dos dados foi feita em duas etapas: treinamento e validação. Foi feita a base de treino para que os algoritmos pudessem ser treinados.

Normalmente, a base de treino tem maior quantidade de instâncias. Caso a base de treino seja formada de forma errada podem ocasionar duas situações que definem o modelo como inconfiável: *underfitting* e *overfiting*. O *underfitting* significa que o modelo não foi capaz de aprender a como resolver o problema, já o *overfitting* não conseguiu aprender a generalizar o problema ou só funciona com a base de treino, o que a caracteriza como super vício. Os modelos *underfitting* e *overfitting* detêm problemas de acurácia os tornando pouco confiáveis. Sendo assim a preparação dos dados é de suma importância para obtenção de resultados satisfatórios.

A base de validação é usada para validar os modelos passados pelo treinamento. Assim que é finalizada cada época (*epoch*) é calculada a função de perda e acurácia. Após o treinamento da arquitetura é feita a validação para pode calcular as métricas de desempenho para poder verificar a melhoria de resultado.

A base de dados utilizada no programa foi do *dataset* FER 13 (Facial Expression Recognition) apresentada por Goodfellow em uma conferência em Berlim, a qual contém 35887 imagens em *grayscale* de 48x48 pixels, sendo 80% delas usadas para treino e 20% usadas para testes. Ou seja, foram usadas 28709 imagens para treino e 7178 para testes.

2.3.5 Implementações utilizadas

Para a realização de testes e implementação deste trabalho foi utilizado um codificação disponibilizada pelo autor/usuário “simplesaad” no site” https://github.com/simplesaad/EmotionDetection_RealTime” no qual o autor disponibiliza o código que foi utilizado para fazer os testes de aplicação das redes neurais artificiais para leitura de expressões faciais e identificação de emoções o qual teve pontos modificados como a cor da moldura e das legendas e a tradução das mesmas e a implementação de algumas métricas para avaliação dos resultados.

Figura 15 - Emoções mostradas nas molduras

```
121
122     emotion_dict = {0: "Raiva", 1: "Nojo", 2: "Medo", 3: "Felicidade", 4: "Neutro", 5: "Tristeza", 6: "Surpresa"}
123     # emocoes que serão amostradas acima do frame
```

Fonte: Elaboração própria..

Figura 16 - Cores da moldura e texto

```
for (x, y, w, h) in faces:
    cv2.rectangle(frame, (x, y-50), (x+w, y+h+10), (0, 0, 255), 2)
    roi_gray = gray[y:y + h, x:x + w]
    cropped_img = np.expand_dims(np.expand_dims(cv2.resize(roi_gray, (48, 48)), -1), 0)
    prediction = model.predict(cropped_img)
    maxindex = int(np.argmax(prediction))
    cv2.putText(frame, emotion_dict[maxindex], (x+20, y-60), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2, cv2.LINE_AA)
```

Fonte: Elaboração própria..

A base de dados utilizada foi a FER2013, liberada para uso em um concurso de *machine learning* (GOODFELLOW, 2013). O *dataset* foi disponibilizado para download pelo mesmo autor do código, que contém fotos de pessoas emitindo as expressões faciais colocadas em *grayscale* num tamanho de 48x48 pixels.

Figura 17 - Código original

```
def plot_model_history(model_history):
    """
    Plot Accuracy and Loss curves given the model_history
    """
    fig, axs = plt.subplots(1,2,figsize=(15,5))
    # summarize history for accuracy
    axs[0].plot(range(1,len(model_history.history['acc'])+1),model_history.history['acc'])
    axs[0].plot(range(1,len(model_history.history['val_acc'])+1),model_history.history['val_acc'])
    axs[0].set_title('Model Accuracy')
    axs[0].set_ylabel('Accuracy')
    axs[0].set_xlabel('Epoch')
    axs[0].set_xticks(np.arange(1,len(model_history.history['acc'])+1),len(model_history.history['acc'])/10)
    axs[0].legend(['train', 'val'], loc='best')
    # summarize history for loss
    axs[1].plot(range(1,len(model_history.history['loss'])+1),model_history.history['loss'])
    axs[1].plot(range(1,len(model_history.history['val_loss'])+1),model_history.history['val_loss'])
    axs[1].set_title('Model Loss')
    axs[1].set_ylabel('Loss')
    axs[1].set_xlabel('Epoch')
    axs[1].set_xticks(np.arange(1,len(model_history.history['loss'])+1),len(model_history.history['loss'])/10)
    axs[1].legend(['train', 'val'], loc='best')
    fig.savefig('plot.png')
    plt.show()
```

Fonte: Elaboração própria..

Outra modificação feita no código foi a mudança do valor ‘acc’ para ‘accuracy’ pois o código não era capaz de plotar o treinamento da base com sucesso.

Figura 18 - Código alterado

```
def plot_model_history(model_history):
    # plots criados a partir de dados coletados do modelo pre-definido acima

    fig, axs = plt.subplots(1,2,figsize=(15,5))
    # plot de acuracia do treinamento e validacao dos epocas
    axs[0].plot(range(1,len(model_history.history['accuracy'])+1),model_history.history['accuracy'])
    axs[0].plot(range(1,len(model_history.history['val_accuracy'])+1),model_history.history['val_accuracy'])
    axs[0].set_title('Model Accuracy')
    axs[0].set_ylabel('Accuracy')
    axs[0].set_xlabel('Epoch')
    axs[0].set_xticks(np.arange(1,len(model_history.history['accuracy'])+1),len(model_history.history['accuracy'])/10)
    axs[0].legend(['train', 'val'], loc='best')
    # plot de perda do treinamento e validacao dos epocas
    axs[1].plot(range(1,len(model_history.history['loss'])+1),model_history.history['loss'])
    axs[1].plot(range(1,len(model_history.history['val_loss'])+1),model_history.history['val_loss'])
    axs[1].set_title('Model Loss')
    axs[1].set_ylabel('Loss')
    axs[1].set_xlabel('Epoch')
    axs[1].set_xticks(np.arange(1,len(model_history.history['loss'])+1),len(model_history.history['loss'])/10)
    axs[1].legend(['train', 'val'], loc='best')
    fig.savefig('plot.png')
    plt.show()
```

Fonte: Elaboração própria.

Após essa mudança foi feito o treinamento e nos foi devolvida a Figura 52 utilizada para analisar a precisão e taxa de perda do modelo da rede.

Foi necessária também a mudança do diretório que leria o *dataset* para a pasta em que estão as imagens de treino e validação.

Figura 19 - Diretórios

```
# caminhos do data set, pastas com imagens de treinamento e validacao respectivamente
train_dir = 'C:/Users/Felipe/College/TCC/New/data/train'
val_dir = 'C:/Users/Felipe/College/TCC/New/data/test'
```

Fonte: Elaboração própria.

Seguida das definições dos diretórios é necessário também a definição dos números de imagens de treino, validação, número de exemplos de treinamento e quantidade de épocas usadas no treino.

Figura 20 - Treino, validação, exemplos e épocas

```
num_train = 28709 #quantidade de imagens de treino
num_val = 7178 #quantidade de imagens de validacao
batch_size = 64 #numero de exemplos de treinamento
num_epoch = 50 #numero de vezes que o programa vai 'ler' o bd em um treinamento
```

Fonte: Elaboração própria.

Outra situação que teve uma análise especial foi a parte do código que indica o modelo e os detalhes da rede.

Figura 21 - Arquitetura da RNA convolucional utilizada

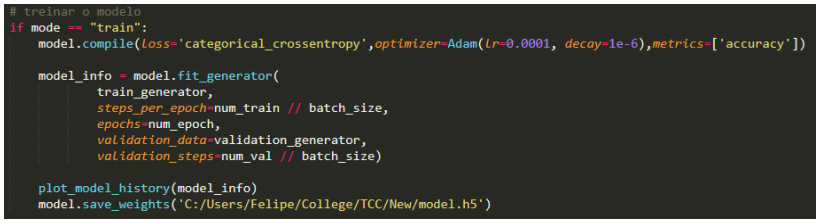
```
model = Sequential()

model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(48,48,1)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

Fonte: Elaboração própria.

Os dados apresentados pela Figura 21 *indicam que a rede utilizada foi como rede neural artificial de convolução 2D com um* kernel *de 3x3 e adicionada também uma camada de* max pooling *2D 2x2.*

Logo em seguida o código apresenta as duas funções disponíveis para iniciá-la no promp de comando.

Figura 22 - Modo treino



```
# treinar o modelo
if mode == "train":
    model.compile(loss='categorical_crossentropy',optimizer=Adam(lr=0.0001, decay=1e-6),metrics=['accuracy'])

    model_info = model.fit_generator(
            train_generator,
            steps_per_epoch=num_train // batch_size,
            epochs=num_epoch,
            validation_data=validation_generator,
            validation_steps=num_val // batch_size)

    plot_model_history(model_info)
    model.save_weights('C:/Users/Felipe/College/TCC/New/model.h5')
```

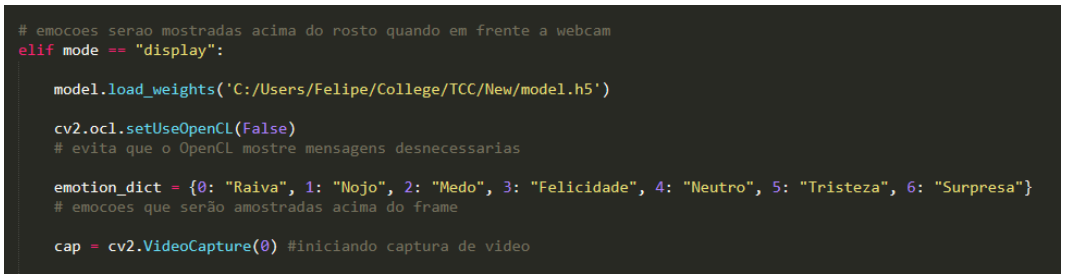
Fonte: Elaboração própria.

Nesta parte do código é definido o modo de treino na rede o qual utiliza o número de imagens de treino por época e validação das mesmas.

Após o treino foi plotada a Figura 52 *definindo gráficos de acurácia e perda, e seguindo a apresentação desse gráfico os dados do treino são salvos no "model.h5" apontado no diretório inserido no código.*

Seguindo o treino, o código apresenta a parte que inicia a captura de vídeo.

Figura 23 - Código captura de vídeo



```
# emocoes serao mostradas acima do rosto quando em frente a webcam
elif mode == "display":

    model.load_weights('C:/Users/Felipe/College/TCC/New/model.h5')

    cv2.ocl.setUseOpenCL(False)
    # evita que o OpenCL mostre mensagens desnecessarias

    emotion_dict = {0: "Raiva", 1: "Nojo", 2: "Medo", 3: "Felicidade", 4: "Neutro", 5: "Tristeza", 6: "Surpresa"}
    # emocoes que serão amostradas acima do frame

    cap = cv2.VideoCapture(0) #iniciando captura de video
```

Fonte: Elaboração própria.

Seguidas as modificações e análises, foi feita a execução do código e análises dos resultados em diferentes situações vistas nas figuras 20, 21, 22 e 23.

2.4 Testes

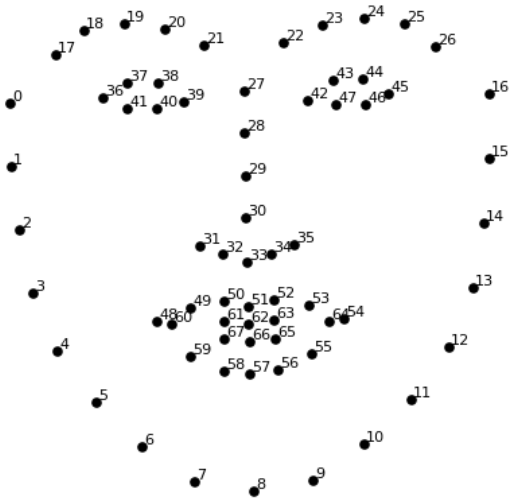
Neste ponto, apresenta-se as abordagens utilizadas para verificar o funcionamento esperado por essa pesquisa. Dentre as muitas abordagens, *LandMarks*, busca por demais bibliotecas, banco de dados e erros, são discutidas nas seções a seguir.

2.4.1 *Landmarks*

Inicialmente o software foi pensado como um leitor das 68 *landmarks* dispostas através do rosto pelas bibliotecas Dlib e OpenCV.

Foi-se utilizado as *landmarks* pois a mesma é capaz de explorar os pontos dos rostos como cantos dos olhos, ponta do nariz, sobrancelhas e boca. O objetivo é encontrar os 68 pontos dispostos na face. O algoritmo é treinado para encontrar os pontos citados para análise da face e da imagem (RECONHECENDO..., 2017).

Figura 24 - 68 landmarks



Fonte: (RECONHECENDO *LANDMARKS* EM FACES COM DLIB + PYTHON, 2017).

Foi possível realizar tal feito, porém não é suficiente para prosseguir à leitura das emoções. Devido a isso a programação do software teve de ser realizada utilizando a linguagem de programação Python.

2.4.2 Busca por bibliotecas diversas

Tendo em mente que o uso do Dlib foi um bom começo, porém não o suficiente para dar continuidade ao objetivo do projeto, foi necessária a utilização de outras bibliotecas que pudessem dar andamento ao software.

O OpenCV foi utilizado em todo o processo devido ao seu ótimo desenvolvimento em análise de imagens e vídeos, exatamente o que o software visa.

Considerando as bibliotecas úteis e adequadas ao problema, a biblioteca Keras, utilizada para *deep learning* e redes neurais foi escolhida por oferecer uma

ótima tratabilidade computacional ao problema da classificação com o uso de redes neurais convolucionais. Porém, apenas a biblioteca não era o suficiente.

2.4.3 Banco de dados

Após a escolha da biblioteca era necessário um banco de dados vasto para uso de *Deep Learning* do Keras.

O banco de dados escolhido foi o FER-2013, com 35887 imagens de 48x48 pixels em *grayscale,* sendo esse tipo de imagem fotos em diversos tons de cinza sem cor aparente com emoções diferentes, como apresentadas nas figuras 25 a 31, separadas em pastas e numeradas para que o software possa fazer a rede neural a partir do banco disposto.

Figura 25 - FER2013 raiva

Fonte: Elaboração própria.

Figura 26 - FER2013 nojo

Fonte: Elaboração própria.

Figura 27 - FER2013 felicidade

Fonte: Elaboração própria.

Figura 28 - FER2013 medo

Fonte: Elaboração própria.

Figura 29 - FER2013 surpresa

Fonte: Elaboração própria.

Figura 30 - FER2013 neutralidade

Fonte: Elaboração própria.

Figura 31 - FER2013 tristeza

Fonte: Elaboração própria.

2.4.4 Erros durante o processo

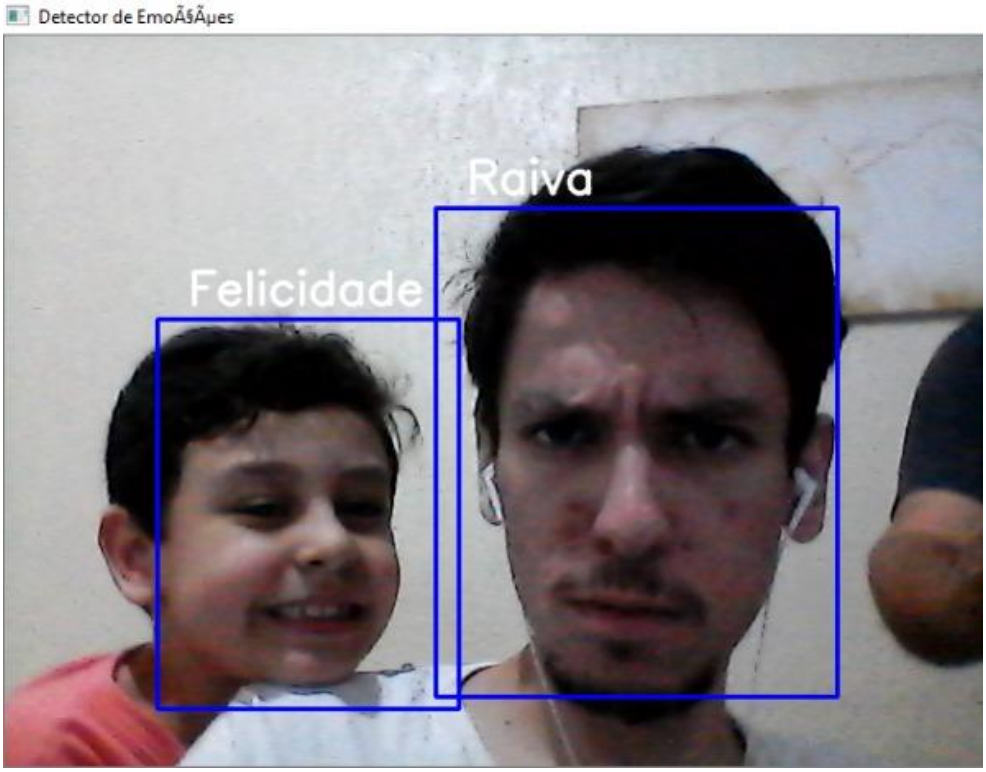
Conforme a implementação utilizada era testada e modificada houve alguns erros que foram resolvidos com pesquisas e estudos na área de programação em Pyhton e suas bibliotecas.

Um dos erros que mais causou retraso foi o “ImportError: Could not find 'nvcuda.dll'” que teria de ser resolvido fazendo o download da dll e de um executável no site da nvidia. Após pesquisa foi facilmente resolvido e pode-se prosseguir com a programação.

3 RESULTADOS

O software foi capaz de categorizar as expressões faciais citando as emoções acima da moldura em volta do rosto apresentado na câmera, como apresentado na Figura 32 em um ambiente e situação perfeita. Esse detector de emoções só funciona após rede neural artificial convolucional ter sido treinada e ajustada. É importante frisar que um bom desempenho do detector de emoções, pode representar que a RNA possui uma capacidade de generalização, afinal, os rostos identificados na aplicação certamente serão rostos de pessoas diferentes das existentes nas fotos contidas no *dataset* utilizado para treinamento.

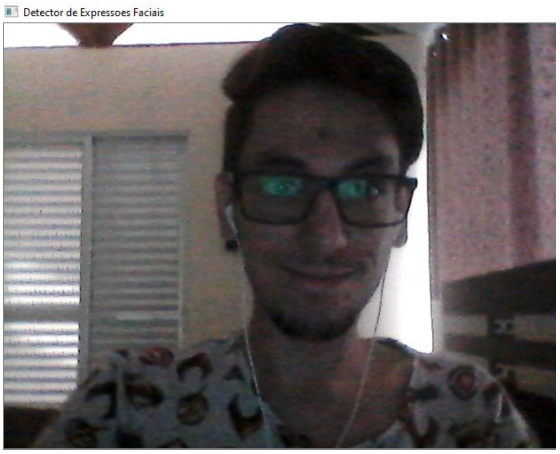
Figura 32 - Detector de Emoções



Fonte: Elaboração própria.

Fora de um ambiente perfeito e com detalhes adicionais os resultados não foram os mesmos e com uma precisão menor como demonstrado nas situações seguintes: A Figura 33 demonstra o resultado do detector de emoções ao identificar uma expressão de felicidade em um ambiente escuro no qual uma pessoa se encontra também com um óculos de grau. Conforme pode ser notado, a rede não conseguiu realizar a classificação e o detector não mostrou o marcador que indica a emoção expressa no rosto.

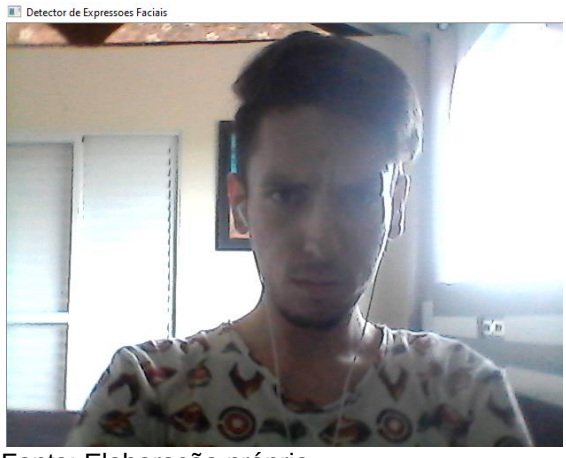
Figura 33 - Felicidade em ambiente escuro com óculos



Fonte: Elaboração própria..

A Figura 34 demonstra o resultado do detector de emoções em uma simulação na qual a expressão facial apresentada representa raiva em um ambiente muito iluminado por luz natural. Mais uma vez o detector não conseguiu identificar a face e consequentemente não houve classificação da emoção apresentada. É importante destacar que em aplicações de reconhecimento facial, as condições de iluminação do ambiente e também o dispositivo de captura utilizado representam grandes dificuldades ao processo de classificação.

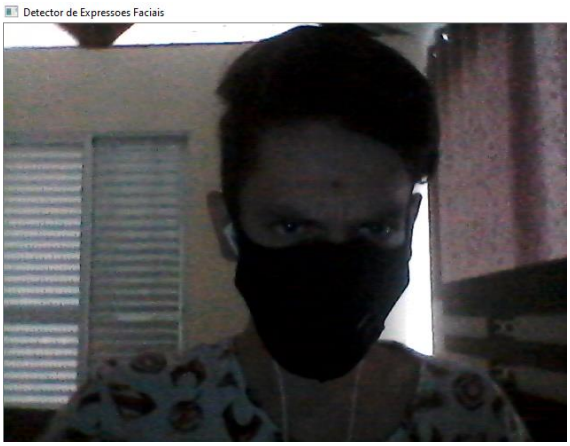
Figura 34 - Raiva em ambiente muito iluminado



Fonte: Elaboração própria.

A Figura 35 representa a simulação de uma situação em que uma pessoa pode estar com a expressão facial correspondente a raiva e utilizando máscara de proteção. Conforme pode ser notado, o detector também teve dificuldades na classificação, o que é compreensível, considerando que no *dataset* utilizado para treinamento não foram fornecidas imagens que representem situações semelhantes.

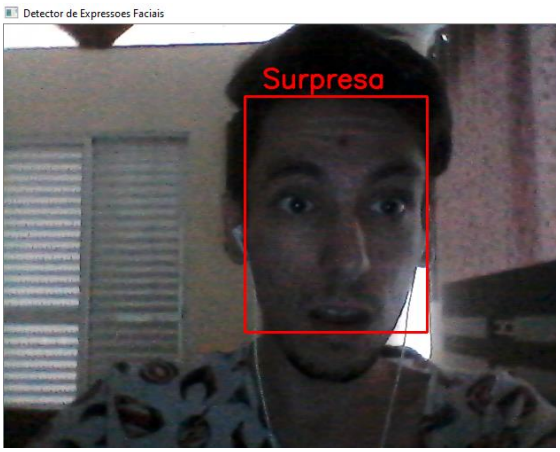
Figura 35 - Raiva com máscara



Fonte: Elaboração própria.

A Figura 36 ilustra a simulação de uma pessoa que esteja com a expressão facial de surpresa e também esteja em um ambiente mal iluminado. É possível notar que o detector conseguiu identificar o rosto e também classificar corretamente a emoção apresentada.

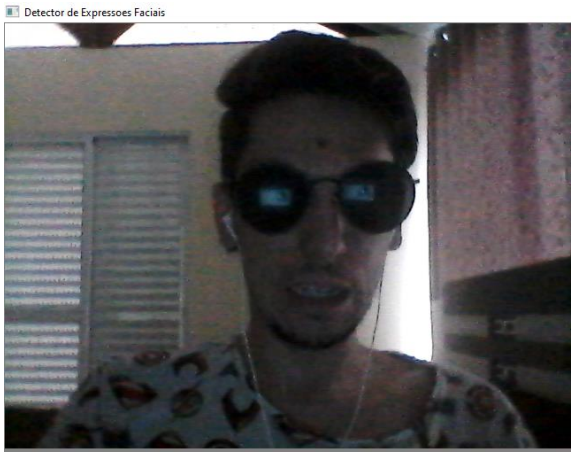
Figura 36 - Surpresa em ambiente mal iluminado



Fonte: Elaboração própria.

A Figura 37 representa a simulação de uma situação em que uma pessoa esteja com a expressão facial de medo e esteja também utilizando óculos escuros. O detector não conseguiu identificar o rosto e consequentemente não houve a classificação a emoção apresentada.

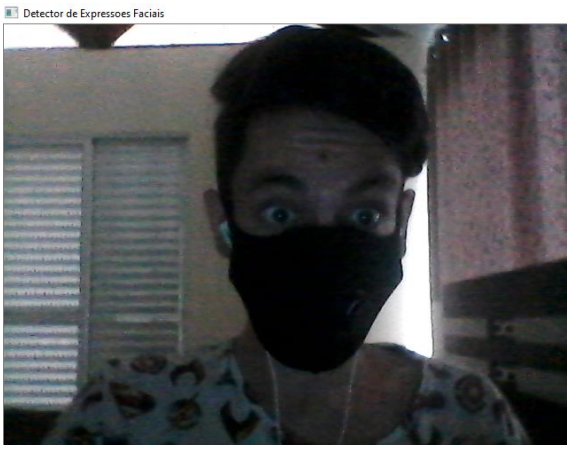
Figura 37 - Medo com óculos escuros



Fonte: Elaboração própria.

A Figura 38 representa a simulação de uma situação em que uma pessoa esteja com a expressão facial de surpresa e esteja também utilizando máscara de proteção. O detector também não conseguiu identificar o rosto e consequentemente não houve a classificação a emoção apresentada.

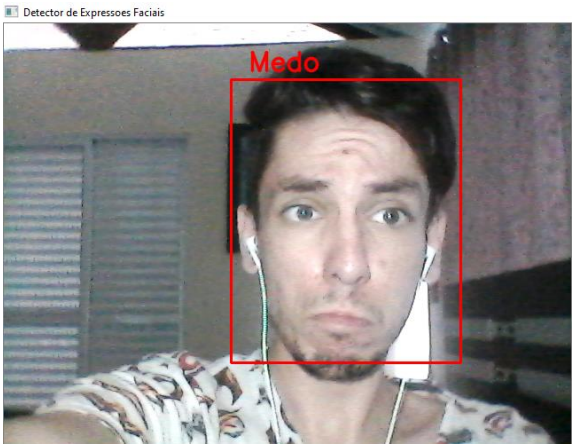
Figura 38 - Surpresa com máscara



Fonte: Elaboração própria.

A Figura 39 representa a simulação de uma situação em que uma pessoa esteja com a expressão facial de tristeza, contudo, para compreender a importância da iluminação da face, foi colocada uma iluminação artificial direcionada para o rosto para verificar se há melhora na identificação do rosto e consequentemente na classificação correta da emoção expressa. O detector conseguiu identificar o rosto com mais facilidade devido a iluminação, contudo, classificou de forma errada a expressão apresentada.

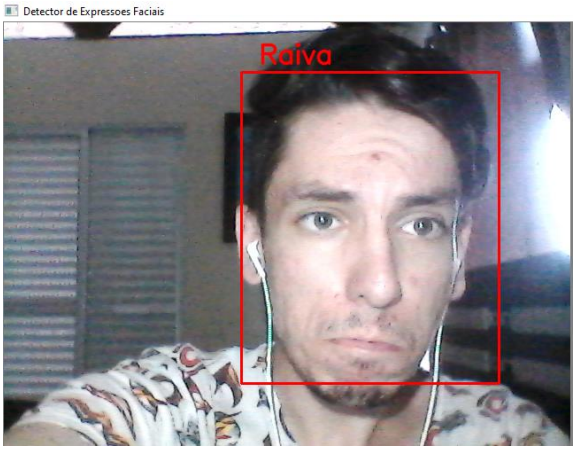
Figura 39 - Tristeza com iluminação artificial 1



Fonte: Elaboração própria.

As Figura 40 e Figura 41 tiveram o mesmo objetivo da Figura 39, ou seja, testar o desempenho do detector com a presença de iluminação, mudando levemente a expressão facial para representação da tristeza. Mesmo assim, o detector classificou de forma errada a expressão facial existente.

Figura 40 - Tristeza com iluminação artificial 2



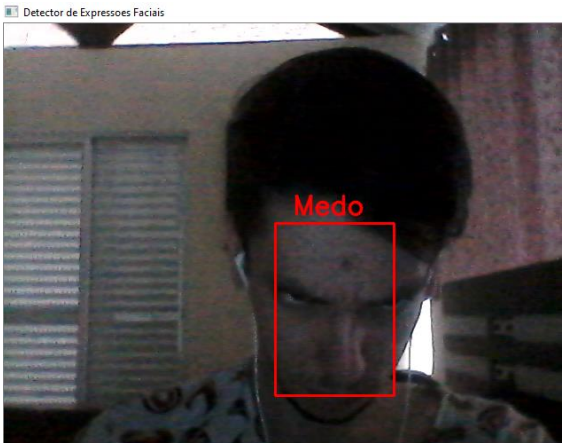
Fonte: Elaboração própria.

Figura 41 - Tristeza com iluminação artificial 3

Fonte: Elaboração própria.

A Figura 42 demonstra uma simulação realizada para verificar o desempenho do detector em situações em que o rosto tenha mudanças de posicionamento perante à câmera utilizada para captura. A identificação do rosto foi realizada, contudo, a classificação da expressão facial foi equívocada, classificando o sentimento como medo, quando deveria ser raiva.

Figura 42 - Raiva com rosto apontado pra baixo



Fonte: Elaboração própria.

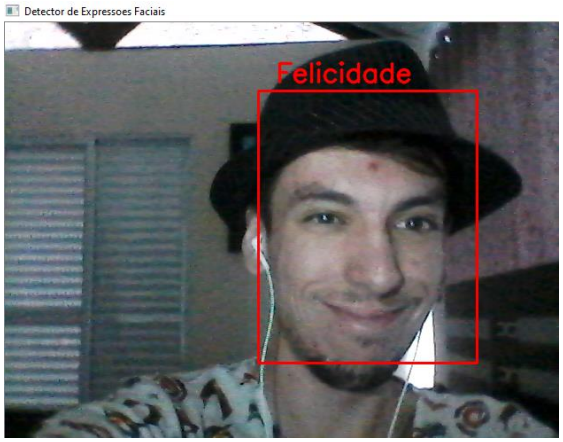
As Figura 43 e

Figura 44 demonstram simulações realizadas para verificar o desempenho do detector em situações em que exista objetos próximo ao rosto ou uso de adereços como é o caso de um chapéu. A identificação do rosto foi realizada normalmente e corretamente quando a expressão facial foi para esboçar surpresa e felicidade.

Figura 43 - Surpresa com chapéu

Fonte: Elaboração própria.

Figura 44 - Felicidade com chapéu

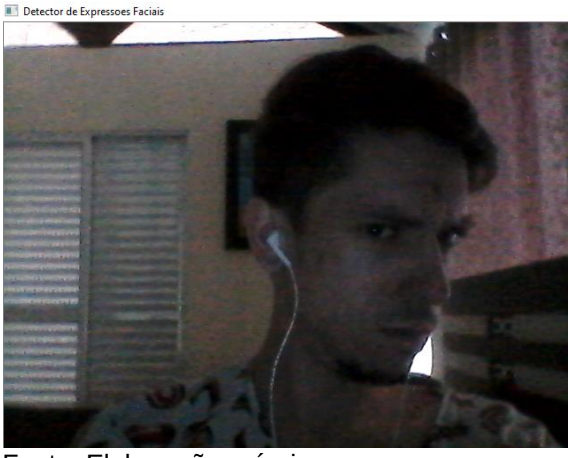


Fonte: Elaboração própria..

As Figura 45 e Figura 46 demonstram simulações realizadas para verificar o desempenho do detector em situações em o rosto esteja sendo exibido de perfil ou esteja em movimento. A identificação do rosto de perfil não ocorreu, portanto, não houve classificação da emoção expressa. Na situação do rosto que estava em movimento, a identificação da face foi realizada, mas a expressão facial representando medo foi classificada como raiva.

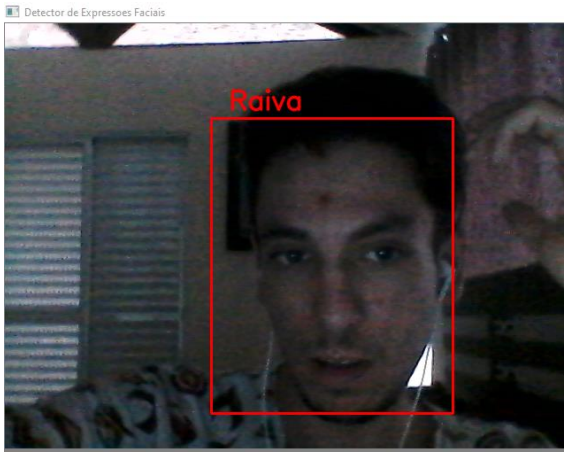
No caso das Figura 47 Figura 48 o perceptível foi a variação entre duas emoções no momento da leitura em que foi realizado testes para verificar o impacto na classificação do detector ao colocar a mão na lateral do rosto com uma expressão facial de raiva. A oscilação ocorreu devido a classificação inicial da expressão facial ser correta (raiva) e posteriormente apresentar uma classificação falha (medo).

Figura 45 - Raiva com o rosto apontado lateralmente



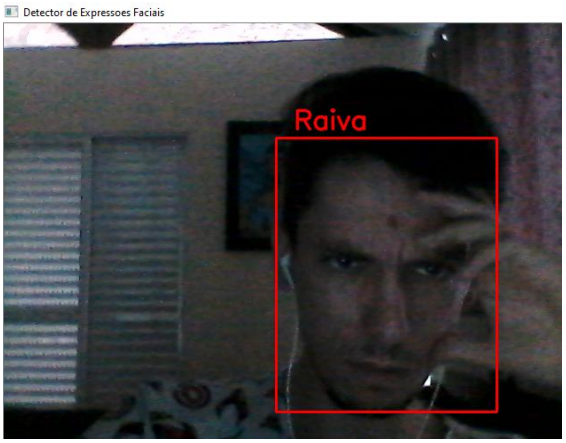
Fonte: Elaboração própria.

Figura 46 - Medo com rosto em movimento



Fonte: Elaboração própria.

Figura 47 - Raiva com a mão na lateral do rosto 1



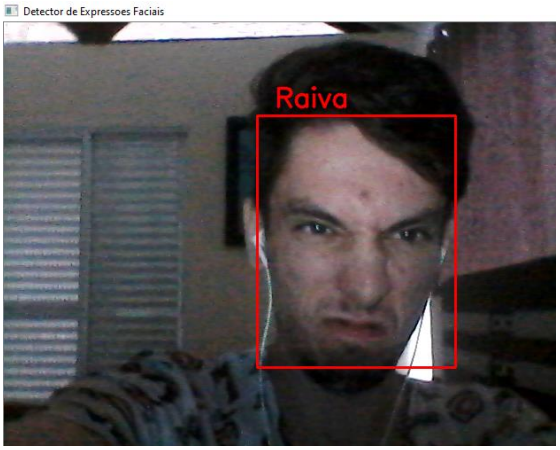
Fonte: Elaboração própria.

Figura 48 - Raiva com a mão na lateral do rosto 2

Fonte: Elaboração própria.

As Figura 49 e Figura 50representam testes que foram realizados simulando expressões faciais de nojo para verificar a capacidade do detector nessas situações. Em ambas as tentativas a identificação do rosto foi possível, mas a classificação não acertou a emoção apresentada.

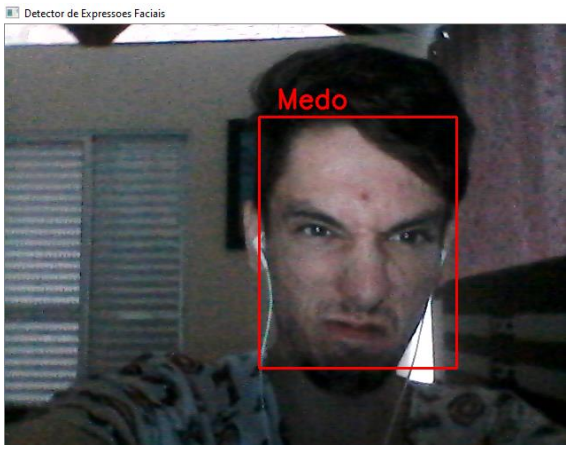
Figura 49 - Nojo 1



Fonte: Elaboração própria.

Em relação à expressão de nojo, em nenhum dos testes feitos ela foi apresentada e quando emitida pelo usuário foi definida como outra emoção. Isso pode ser explicado por ser uma classe com poucos exemplos no *dataset* utilizado para o treinamento da RNA. O número de exemplos em cada classe a ser utilizada para classificação precisa ser grande o suficiente para que a RNA busque aprender os diferentes comportamentos e variações da face.

Figura 50 - Nojo 2



Fonte: Elaboração própria..

Sendo assim, o que podemos concluir com os testes é que a variação de ambiente, obstrução do rosto e variação de posição interferem na leitura da emoção devido aos exemplos disponibilizados pelo *dataset* FER2013.

O uso de acessórios impossibilitou ou dificultou a leitura da emoção apresentada. Já a variação de ambiente e a mudança de angulação do rosto mudou completamente a leitura e a fez variar entre duas ou três emoções.

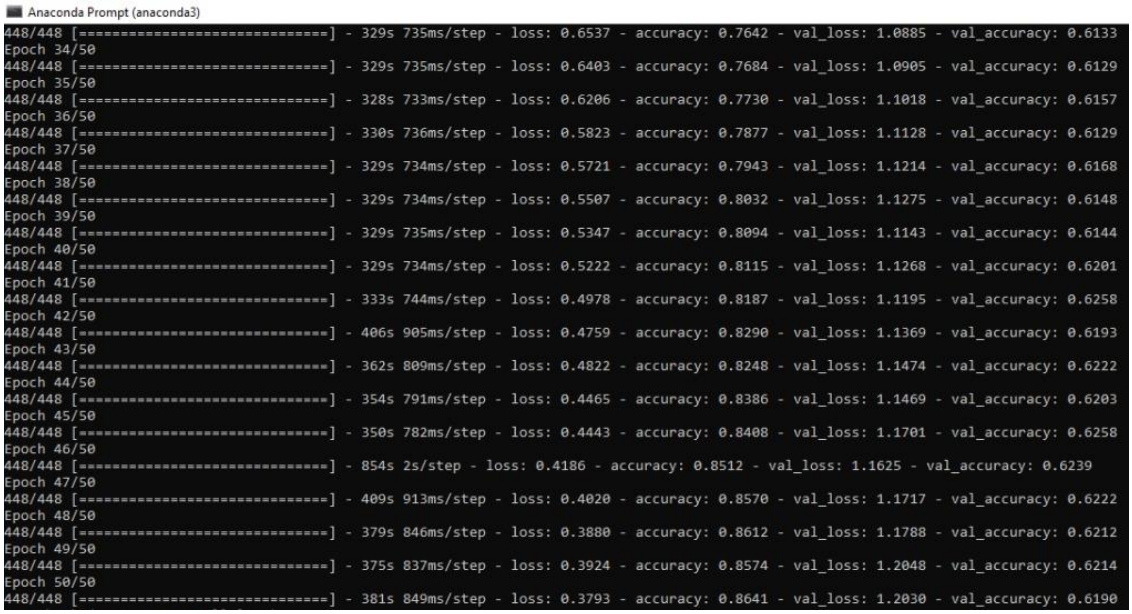
Após todo o estudo sobre as expressões e microexpressões faciais, musculaturas e programação de leitura de faces, notou-se que o software foi capaz de ler as expressões faciais e suas emoções e as reconheceu fazendo assim a análise do rosto em movimento e em tempo real.

O software criou a rede neural e a treinou para analisar as expressões faciais dispostas no banco de dados. Após a finalização de tal processo, foi feita a leitura da expressão facial e reproduzido o resultado no display criado pelo OpenCV.

O resultado foi um software capaz de ler as expressões faciais com uma precisão de 61.09% apresentada na Figura 51 que apresenta a acurácia do *dataset* já treinado, porcentagem próxima à citada no artigo do *dataset* FER2013 (GOODFELLOW, 2013). Apesar de uma porcentagem maior que a metade o resultado continua com a precisão menor do que a de um especialista devido à programação que gerou um possível *underfitting* e a fatores externos, ambientais e humanos que fazem com que a precisão de um especialista chegue a 80% dependendo do tempo de estudo à área do estudo das expressões faciais (MEYER, 2011).

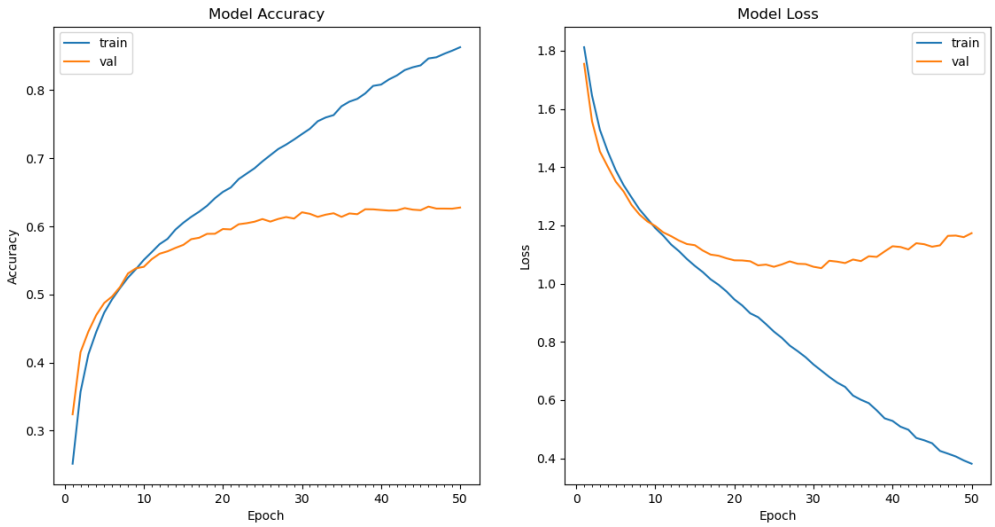
Os modelos de precisão/acurácia e perda apresentados na Figura 52 mostram que a diferença entre o treino e a validação tem uma variação considerável devido a um possível *underfitting.*

Figura 51 - Processo de treinamento e os valores de perda e acurácia



Fonte: Elaboração própria..

Figura 52 - Modelos de Acurácia e Perda



Fonte: Elaboração própria..

Como apresentado pela matriz de confusão das Figura 53Figura 54, a rede treinada é relativamente boa em detectar as emoções felicidade, neutra e tristeza, enquanto relação às emoções nojo, surpresa e raiva ela apresentou um desempenho ruim.

Figura 53 - Matriz de confusão e as medidas precision, recall, f1-score e support

```
Confusion Matrix
[[133  15 119 247 178 161 105]
 [ 15   1  10  34  20  20  11]
 [117  16 128 293 172 178 120]
 [245  23 200 478 317 310 201]
 [158  16 147 345 218 210 139]
 [174  14 123 371 233 215 117]
 [ 96   8 109 211 149 163  95]]
Classification Report
              precision    recall  f1-score   support

       raiva       0.14      0.14      0.14       958
        nojo       0.01      0.01      0.01       111
        medo       0.15      0.12      0.14      1024
  felicidade       0.24      0.27      0.25      1774
      neutro       0.17      0.18      0.17      1233
    tristeza       0.17      0.17      0.17      1247
    surpresa       0.12      0.11      0.12       831

    accuracy                           0.18      7178
   macro avg       0.14      0.14      0.14      7178
weighted avg       0.17      0.18      0.17      7178
```

Fonte: Elaboração própria.

Em relação à emoção “medo” houve um desempenho mediano em relação às outras. Sendo assim a hipótese considerada é que a diferença de quantidade de imagens de referência no *dataset* interferiu no desempenho da rede pois ela teve uma

quantidade diferente de exemplos para se basear na hora de ler os sentimentos, a quantidade de imagens de felicidade é bem maior que a de nojo, sendo assim o desempenho da classificação de expressões faciais de felicidade foi bem melhor.

Figura 54 - Matriz de confusão com os rótulos das classes previstas e esperadas

CLASSE ESPERADA \ CLASSE PREVISTA	RAIVA	NOJO	MEDO	FELICIDADE	NEUTRO	TRISTEZA	SURPRESA
RAIVA	113	15	119	247	178	161	105
NOJO	15	1	10	34	20	20	11
MEDO	117	16	128	293	172	178	120
FELICIDADE	245	23	200	478	317	310	201
NEUTRO	158	16	147	345	218	210	139
TRISTEZA	174	14	123	371	233	215	117
SURPRESA	96	8	109	211	149	163	95

Fonte: Elaboração própria.

Uma solução para se resolver tal problema seria prover maiores imagens das emoções nojo, raiva e surpresa para balancear o *dataset* e gerar uma leitura/classificação mais precisa.

É importante destacar que o *dataset* apresenta um total de 35887 imagens sendo elas divididas em 28709 imagens de treino e 7178 imagens de validação, sendo assim, cerca de 80% dos dados são separados para treino e 20% para validação.

Abaixo foi apresentada uma tabela para melhor ilustrar a separação de cada emoção:

Figura 55 - Tabela de treino e validação

	Raiva	Nojo	Medo	Felicidade	Neutro	Tristeza	Surpresa
Treino	3995	436	4097	7215	4965	4830	3171
Validação	958	111	1024	1774	1233	1247	831

Fonte: Elaboração própria.

Tanto a Figura 54 quanto a Figura 55 demonstram a grande diferença da quantidade de imagens do *dataset* e esse desbalanceamento certamente influência no resultado, contudo, não é possível afirmar que seja o fator preponderante.

4 CONCLUSÃO

Com o desenvolvimento deste trabalho foi possível cumprir o objetivo de compreender os diferentes sentimentos que podem ser expressos na face de uma pessoa e verificar a capacidade de uma rede neural artificial convolucional em aprender, classificar e generalizar esse tipo de problema de identificação facial.

O estudo e entendimento da linguagem python e das bibliotecas utilizadas permitiram fazer modificações importantes no código utilizado como o cálculo das medidas *precision, recall, f1-score* e *recall* e também a geração da matriz de confusão que é indispensável para a melhor compreensão dos resultados da RNA utilizada.

Com relação aos resultados, pode-se chegar à conclusão que é necessário um banco de dados vasto e balanceado para um resultado satisfatório, pois mesmo as imagens atendendo a todos os requisitos de preparo (*grayscale* e tamanho 48x48) a quantidade de imagens utilizadas em cada classe/sentimento para treinamento foi crucial para o bom desempenho da rede neural que acabou reconhecendo mais algumas emoções do que outras.

O desempenho da classificação é mediano, mas não é o suficiente para se basear apenas nelas para uma leitura emocional. A rede teve um aprendizado relativamente rápido para a máquina utilizada e apesar das emoções ocasionalmente lidas erroneamente a câmera e o software detectaram bem um número de dois rostos.

A abordagem utilizada tem potencial de aumentar a porcentagem de acertos e ter uma melhor classificação, contudo, para isso é preciso realizar e verificar diferentes tipos de ajustes e mudanças na arquitetura que provavelmente irá demandar o uso de uma GPU para que o processo seja mais rápido e mais eficiente, o que não havia a disposição para a execução deste trabalho.

Após o desenvolvimento deste trabalho foi verificado alguns pontos de melhorias que sugerem possíveis trabalhos futuros a serem realizados e serão citados a seguir:

- Realizar treinamentos com a RNA utilizada com um dataset que tenha uma quantidade equivalente de imagens entre as diferentes classes que se deseja classificar;
- Criar um dataset com imagens em diferentes condições de captura com rostos em orientações distintas, assim como também considerando a utilização de adereços, máscaras de proteção, óculos, entre outras

situações que são corriqueiras no dia a dia e que podem ser importantes em um sistema de reconhecimento de expressões faciais;

- Testar diferentes arquiteturas da RNA utilizada e também modificar seus parâmetros para buscar obter melhores resultados;
- Incluir uma etapa para realizar o processamento de imagens utilizadas para o treinamento e para a validação do método de classificação utilizado com o objetivo de buscar melhorar e equilibrar a iluminação das imagens para facilitar a identificação da face e consequentemente melhorar a classificação;
- Testar o detector de expressões faciais com outros equipamentos de captura que possuam melhor resolução.

REFERÊNCIAS

ABOUT KERAS. 2015. **Disponível em**: https://keras.io/about/. Acesso em: 31 ago. 2020

EKMAN, Paul et al. **Paul Ekmans Group**: facial action coding system. Facial Action Coding System. 2002. Disponível em: https://www.paulekman.com/facial-action-coding-system/. Acesso em: 02 jun. 2020.

EKMAN, Paul; FRIESEN, Wallace V.. **Unmasking the Face**: a guide to recognizing emotions from facial expressions. Cambridge: Malor Books, 2003. 232 p.

ELGAN, Mike. **Computerworld**: Apple's iPhone X proves it: Silicon Valley is getting emotional. 2017. **Disponivel em**: https://www.computerworld.com/article/3235424/apples-iphone-x-proves-it-silicon-valley-is-getting-emotional.html. Acesso em: 12 nov. 2020.

EXPRESSAMENTE. 2014. Disponível **em:** http://expressamente.com/a-ciencia-por-tras-dos-filmes-da-disney/. Acesso em: 12 nov. 2020.

FLECK, Leandro; TAVARES, Maria; EYNG, Eduardo; HELMANN, Andrieli; ANDRADE, Minéia. **Redes Neurais Artificiais:** Princípios Básicos. Revista Eletrônica Científica Inovação e Tecnologia. Universidade Tecnológica Federal do Paraná, Medianeira, Paraná. V.1, n.13, p. 47-57, jan./jun. 2016.

GÉRON, A.. **Hands-on machine learning with Scikit-Learn and TensorFlow:** concepts, tools, and techniques to build intelligent systems. "O'Reilly Media, Inc." (2017).

GOODFELLOW, I., BENGIO, Y., E COURVILLE, A.. **Deep learning**. MIT press. (2016)

GOODFELLOW, I.J., ERHAN, D., CARRIER, P.L., COURVILLE, A., MIRZA, M., HAMNER, B., ZHOU, Y.: **Challenges in representation learning**: a report on three machine learning contests. In: Lee, M., Hirose, A., Hou, Z. G., Kil R.M. (eds.) Neural Information Processing, ICONIP 2013. Lecture Notes in Computer Science, vol. 8228, Springer, Berlin, Heidelberg (2013).

HAYKIN, S. **Redes Neurais- Princípios e Práticas**. BOOKMAN, São Paulo, 2ª ed. 900 p.(2001).

ICML – INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 30., 2013, Atlanta. **Anais** [...]. Atlanta: ICML, 2013. Disponível em: https://www.researchgate.net/publication/243964130_Challenges_in_Representation_Learning_A_Report_on_Three_Machine_Learning_Contests. Acesso em: 28 set. 2020.

KERAS. 2015. **Disponível em**: https://keras.io/. Acesso em: 04 out. 2020.

MEYER, Pamela. **Liespotting**: proven techniques to detect deception. New York: St. Martin's Griffin, 2011. 256 p.

MITCHELL, Tom M. **Machine Learning.** New York: McGraw-Hill, 1997. 414 p.

MODELO SEQUENCIAL DO KERAS. 2018. **Disponível em**: https://www.monolitonimbus.com.br/modelo-sequencial-do-keras. Acesso em: 31 ago. 2020

NETTER, Frank H.. Atlas de Anatomia Humana. 2ed. Porto Alegre: Artmed, 2000.

NOYES, Katherine. **Computerworld:** An iPhone that feels your pain? Apple's Emollient buy could make it happen. 2016. Disponível em: https://www.computerworld.com/article/3020159/an-iphone-that-feels-your-pain-apples-emollient-buy-could-make-it-happen.html. Acesso em: 12 nov. 2020.

OPENCV. 1999. **Disponível em**: https://opencv.org/. Acesso em: 12 maio 2020.

PAUL EKMAN GROUP. 2004. **Disponivel em**: https://www.paulekman.com. Acesso em: 11 nov 2020.

PYTHON. 2001. **Disponível em**: https://www.python.org/about/apps/. Acesso em: 24 mar. 2020.

RECONHECENDO LANDMARKS EM FACES COM DLIB + PYTHON. 2017. **Disponível em**: https://medium.com/@suzana.svm/reconhecendo-landmarks-em-faces-com-dlib-python-7bfb094e1bb4. Acesso em: 22 out. 2020.

SILVA, Igor; SILVA, Rogério. Linguagem de Programação Python. **Revista Tecnologias em Projeção**, v10, nº1, ano 2019. p.55-71.

TENSORFLOW. 2015. **Disponivel em:** https://www.tensorflow.org/overview. Acesso em: 31 ago. 2020.

ANEXO A

```
import numpy as np
import argparse
import cv2
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Flatten
from keras.layers.convolutional import Conv2D
from keras.optimizers import Adam
from keras.layers.pooling import MaxPooling2D
from keras.preprocessing.image import ImageDataGenerator
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix

import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import matplotlib as mpl
mpl.use('TkAgg')
import matplotlib.pyplot as plt

# linhas de comando que serao usados para incializar o programa
ap = argparse.ArgumentParser()
ap.add_argument("--mode",help="train/display")
a = ap.parse_args()
mode = a.mode

def plot_model_history(model_history):
    # plots criados a partir de dados coletados do modelo pre-definido acima

    fig, axs = plt.subplots(1,2,figsize=(15,5))
    # plot de acuracia do treinamento e validacao dos epocas

axs[0].plot(range(1,len(model_history.history['accuracy'])+1),model_history.history['ac
curacy'])

axs[0].plot(range(1,len(model_history.history['val_accuracy'])+1),model_history.histor
y['val_accuracy'])
    axs[0].set_title('Model Accuracy')
    axs[0].set_ylabel('Accuracy')
    axs[0].set_xlabel('Epoch')

axs[0].set_xticks(np.arange(1,len(model_history.history['accuracy'])+1),len(model_his
tory.history['accuracy'])/10)
    axs[0].legend(['train', 'val'], loc='best')
    # plot de perda do treinamento e validacao dos epocas

axs[1].plot(range(1,len(model_history.history['loss'])+1),model_history.history['loss'])
```

```
axs[1].plot(range(1,len(model_history.history['val_loss'])+1),model_history.history['val
_loss'])
    axs[1].set_title('Model Loss')
    axs[1].set_ylabel('Loss')
    axs[1].set_xlabel('Epoch')

axs[1].set_xticks(np.arange(1,len(model_history.history['loss'])+1),len(model_history.
history['loss'])/10)
    axs[1].legend(['train', 'val'], loc='best')
    fig.savefig('plot.png')
    plt.show()

# caminhos do data set, pastas com imagens de treinamento e validacao
respectivamente
train_dir = 'C:/Users/Felipe/College/TCC/New/data/train'
val_dir = 'C:/Users/Felipe/College/TCC/New/data/test'

num_train = 28709
num_val = 7178
batch_size = 64 #numero de exemplos de treinamento #64
num_epoch = 50 #numero de vezes que o programa vai 'ler' o bd em um treinamento

train_datagen = ImageDataGenerator(rescale=1./255) #treinamento do dataset feito
pelo Keras
val_datagen = ImageDataGenerator(rescale=1./255) #validacao do dataaset feito
pelo Keras

# parte do codigo que fara o treinamento a partir do bd
train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(48,48),
        batch_size=batch_size,
        color_mode="grayscale",
        class_mode='categorical')

# parte do codigo que validara as imagens do bd
validation_generator = val_datagen.flow_from_directory(
        val_dir,
        target_size=(48,48),
        batch_size=batch_size,
        color_mode="grayscale",
        class_mode='categorical')

# criação do modelo
model = Sequential()

model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(48,48,1)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
model.add(Dropout(0.25))

model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(7, activation='softmax'))

#treinamento do modelo FER2013
if mode == "train":
    model.compile(loss='categorical_crossentropy',optimizer=Adam(lr=0.0001,
decay=1e-6),metrics=['accuracy'])

    model_info = model.fit_generator(
            train_generator,
            steps_per_epoch=num_train // batch_size,
            epochs=num_epoch,
            validation_data=validation_generator,
            validation_steps=num_val // batch_size)

    plot_model_history(model_info)
    model.save_weights('C:/Users/Felipe/College/TCC/New/model.h5')

# emocoes serao mostradas acima do rosto quando em frente a webcam
elif mode == "display":

    model.load_weights('C:/Users/Felipe/College/TCC/New/model.h5')


    cv2.ocl.setUseOpenCL(False)
    # evita que o OpenCL mostre mensagens desnecessarias

    emotion_dict = {0: "Raiva", 1: "Nojo", 2: "Medo", 3: "Felicidade", 4: "Neutro", 5:
"Tristeza", 6: "Surpresa"}
    # emocoes que serão amostradas acima do frame

    cap = cv2.VideoCapture(0) #iniciando captura de video




    while True:

        ret, frame = cap.read()
```

```
        facecasc =
cv2.CascadeClassifier('C:/Users/Felipe/College/TCC/New/haarcascade_frontalface_
default.xml')
        #haar cascade ferramenta  do opencv usada para identificar objetos
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) #conversao da imagem
capturada em video para grayscale
        faces = facecasc.detectMultiScale(gray,scaleFactor=1.3, minNeighbors=5)
#detecta imagens de diferentes tamanhos em grayscale

        for (x, y, w, h) in faces:
            cv2.rectangle(frame, (x, y-50), (x+w, y+h+10), (0, 0, 255), 2)
            roi_gray = gray[y:y + h, x:x + w]
            cropped_img = np.expand_dims(np.expand_dims(cv2.resize(roi_gray, (48,
48)), -1), 0)
            prediction = model.predict(cropped_img)
            maxindex = int(np.argmax(prediction))
            cv2.putText(frame, emotion_dict[maxindex], (x+20, y-60),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2, cv2.LINE_AA)


        cv2.imshow("Detector de Expressoes Faciais", frame) #chama o opencv para
mostrar a tela de apresentacao
        key = cv2.waitKey(1) & 0xFF



        if key == ord("q"): #se pressionada a tecla 'q' é interrompido o loop
            break


    cap.release()
    cv2.destroyAllWindows()
```