PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
ESCOLA POLITÉCNICA
GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

RECONHECIMENTO FACIAL PARA DETECÇÃO DE EMOÇÕES UTILIZANDO REDES NEURAIS CONVOLUCIONAIS COM TENSORFLOW

MICHAEL HENRIQUE SOUZA FERREIRA

GOIÂNIA
2021

MICHAEL HENRIQUE SOUZA FERREIRA

RECONHECIMENTO FACIAL PARA DETECÇÃO DE EMOÇÕES UTILIZANDO REDES NEURAIS CONVOLUCIONAIS COM TENSORFLOW

Trabalho de Conclusão de Curso apresentado à Escola Politécnica, da Pontifícia Universidade Católica de Goiás, como parte dos requisitos para a obtenção do título de Bacharel em Engenharia de Computação.

Orientador(a): Prof. Me. Rafael Leal Martins

GOIÂNIA
2021

MICHAEL HENRIQUE SOUZA FERREIRA

RECONHECIMENTO FACIAL PARA DETECÇÃO DE EMOÇÕES UTILIZANDO REDES NEURAIS CONVOLUCIONAIS COM TENSORFLOW

Este Trabalho de Conclusão de Curso julgado adequado para obtenção do título de Bacharel em Engenharia de Computação, e aprovado em sua forma final pela Escola Politécnica, da Pontifícia Universidade Católica de Goiás, em _____/_____/_____.

_____

Prof. Ma.
Coordenador(a) de Trabalho de Conclusão de Curso

Banca examinadora:

_____

Orientador(a): Prof. Me. Rafael Leal Martins

_____

Prof. Me. Fernando Gonçalves Abadia

_____

Prof. Me. Gustavo Siqueira Vinhal

GOIÂNIA
2021

AGRADECIMENTOS

A Deus, pela minha vida, pelas oportunidades e conquistas, e por me ajudar a ultrapassar os obstáculos ao longo da realização do curso. Agradeço especialmente a meu orientador Prof. Me. Rafael Leal Martins, pela confiança, amizade e principalmente pelo apoio nos momentos complexos durante a elaboração do trabalho. Muitíssimo obrigado por entender e apoiar as minhas decisões e mudanças de pensamentos ao longo do desenvolvimento do projeto, além de todos os ensinamentos que me permitiram elaborar esse trabalho.

Agradeço a minha família pelo apoio e pelo amor incondicional em todos os momentos, principalmente nos momentos mais difíceis. Obrigado a Maria e a Rafaela, com vocês ao meu lado os problemas parecem menores. Um obrigado especial a Wesley, por todas as sessões de terapia, e por me ajudar a enxergar a vida de uma maneira mais leve.

Agradeço aos membros da banca pelas contribuições e incentivo para pesquisas futuras. Obrigado ao Welder, Gabriel e Henrique pelo companheirismo ao longo de toda a jornada acadêmica.

RESUMO

Esta pesquisa tem como objetivo levantar e validar metodologias para o desenvolvimento e criação de redes neurais convolucionais capazes de identificar emoções com o uso de imagens. O cenário adotado é o de identificação de expressões faciais em imagens e suas respectivas emoções. Neste contexto, são utilizados estudos de comportamento humano sobre a definição de emoções universais, com base na análise proposta por Paul Ekman. Para o reconhecimento das emoções, foram utilizadas diferentes técnicas de visão computacional e técnicas de aprendizagem de máquina com redes neurais convulsionais. Foram utilizadas ferramentas como o *TensorFlow* e a API do *Keras* para desenvolver de maneira diligente e precisa as redes neurais convolucionais. Pretende-se, com esse trabalho, verificar a capacidade das redes neurais em reconhecer emoções e quão precisos são as predições feitas por elas. Como resultados busca-se encontrar uma melhor arquitetura que possa reconhecer emoções, foi possível conseguir uma comparação satisfatório da acurácia apresentada por cada um dos modelos de redes neurais convolucionais adotados.

Palavras-chave: *expressões faciais, emoções, emoções universais, redes neurais convolucionais, TensorFlow.*

ABSTRACT

This research aims to raise and validate methodologies for the development and creation of convolutional neural networks capable of identifying emotions using images. The scenario adopted is the identification of facial expressions in images and their respective emotions. In this context, studies of human behavior on the definition of universal emotions are used, based on the analysis proposed by Paul Ekman. For the recognition of emotions, different computer vision techniques and machine learning techniques with convulsive neural networks were used. Tools such as TensorFlow and the Keras API were used to diligently and accurately develop convolutional neural networks. The aim of this work is to verify the ability of neural networks to recognize emotions and how accurate the predictions made by them are. As a result, an attempt is made to find a better architecture that can recognize emotions, and it was possible to obtain a satisfactory comparison of the accuracy presented by each of the adopted convolutional neural network models.

Keywords: facial expressions, emotions, universal emotions, convolutional neural networks, TensorFlow.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

As reações a determinadas situações variam conforme o contexto social em que o individuo está envolto. Porém, algumas expressões faciais podem ser consideradas como sendo universais (EKMAN, 2003), isto independentemente da etnia, idade ou gênero, até mesmos em tribos isoladas do contato de outras culturas. Embora sejam consideradas universais, existem vários matizes na demonstração dessas emoções, que vão desde uma resposta clara ao observador até movimentos menos evidentes dos músculos faciais. Essas microexpressões são facilmente observáveis por humanos, dado que podem durar apenas uma fração de segundo. Esse tipo de restrição não ocorre em uma análise realizado por um computador, porém a questão passa ser a forma que esse software foi desenvolvido.

Segundo (LIBRALON, 2014), “sistemas para reconhecimento e análise emocional por meio de expressões podem ser utilizados em segurança, na psiquiatria e ciências comportamentais humanas”, além de diversas outras áreas. Um bom exemplo de aplicação de análise de emoções na relação de interação homem-máquina (IHM) é o desenvolvimento de sistema de tutoria automatizada, onde um tutor virtual para funcionar de maneira eficaz precisaria saber o que usuário está considerando os exemplos interessantes e relevantes, podendo assim se adaptar de modo a obter uma atenção maior do usuário para o assunto. O uso de análise de expressões faciais pode ser aplicado para auxiliar no tratamento de doenças, como Alzheimer e depressão, doenças essas que afetam o reconhecimento de emoções das pessoas.

Uma das áreas onde pode-se aplicar o reconhecimento de imagens são os jogos, que utilizam interfaces de usuário tridimensionais para desenvolver e aprimorar os sensores de captura de imagem. Com isso, permitiram o desenvolvimento de aplicações que geram um mapeamento das feições humanas, além também do espaço em que elas estão inseridas. Possibilitando assim, o desenvolvimento de jogos que possuem uma jogabilidade adaptada para realizar movimentos através dos gestos feitos pelo usuário captados por um sensor (URIBE-QUEVEDO e REIS, 2015).

Redes neurais artificiais são algoritmos que se destacam no campo do aprendizado de máquina. Vê-se hoje cada vez mais interações humano-computador, por meio de softwares de detecção de spam, sistema de recomendação, reconhecimento de voz, escrita e de imagens. O reconhecimento das imagens atualmente é feito principalmente por meio de redes neurais artificiais. Diferentes arquiteturas são usadas a fim de lidar com os mais diferentes tipos e volumes de dados (NIELSEN, 2015).

O modelo de rede neural artificial fundamental é conhecido como rede neural convolucional (CNN), na qual é uma combinação entre PDI e RNA. O objetivo de uma rede CNN e aprender características de objetos para que posteriormente possa classificá-los em cenas diversas representadas através de fotos, vídeos ou até mesmo em tempo real. O processo de aprendizagem de uma CNN se resume em treinar a rede com imagens que contenham os objetos que ela precisa reconhecer. Para que a própria rede seja capaz de extrair as características que julga necessária para o reconhecimento do objeto, é fundamental que as imagens de treinamento contenham o objeto em diferentes situações, como ângulo, escala, foco, ambiente etc. Conforme a rede treina, as características dos objetos vão se tornando mais claras para ela. Até chegar em um ponto onde a rede já sabe quais as características ela deve buscar na imagem quando estiver classificando este objeto (NIELSEN, 2015).

Justifica-se estudar esse assunto devido à pouca quantidade de estudos comportamentais utilizando aprendizagem de máquina, visando estudar principalmente a eficiência no reconhecimento de emoções através de expressões faciais em imagens.

Diante deste contexto, este projeto de pesquisa visa responder a seguinte questão: - **É possível desenvolver uma aplicação capaz de reconhecer com perfeição as emoções apresentadas por um indivíduo, através de expressões faciais?**

1.1 Objetivos

1.1.1 *Objetivo geral*

- Pesquisar e testar métodos de classificação de emoções faciais.
- Analisar e reconhecer emoções a partir de reconhecimento facial.
- Buscar ferramentas fundamentais para treinamento de uma rede neural convolucional;
- Buscar ferramentas de auxílio para treinamento de uma rede neural convolucional;
- Analisar o funcionamento e aplicação do reconhecimento de emoções em vídeo.

1.1.2 *Objetivos específicos*

- Desenvolver uma aplicação utilizando a linguagem Python e a ferramenta OpenCV para detecção e rastreamento da face.
- Desenvolver uma aplicação utilizando ferramenta TensorFlow e a API do Keras para utilizando redes neurais convolucionais.
- Elaborar uma aplicação para reconhecer as principais expressões faciais humanas.
- Comparar os tipos de métodos de redes neurais utilizadas para o reconhecimento de

emoções em imagens.

2 REFERENCIAL TEÓRICO

2.1 O estudo das emoções

Não é fácil encontrar dentro da literatura uma definição única e invariante para o termo emoção. Diversos estudiosos e pesquisadores do tema tentaram definir o que seria emoção ao longo dos anos, buscando encontrar um ponto comum que pudesse mostrar a importância de se estudar as emoções e seus impactos nas relações de convívio humano (VIANA, 2014).

Um dos primeiros estudos sobre o comportamento humano em relação às emoções foi publicado por Charles Darwin em 1872 no livro "A expressão das emoções no homem e nos animais". Darwin defende a presença de emoções no cérebro de animais a partir dos princípios evolucionistas, de modo que as emoções se constituíram com objetivo de garantir melhores condições de sobrevivência e reprodução das espécies, tendo como base os eventos e mudanças do ambiente em que o indivíduo está inserido. Contudo, os seres humanos, possuem características decorrentes do seu amadurecimento individual, que desenvolvem emoções e expressões comportamentais sem função evidentes, que estarão presentes na vida adulta, sendo úteis ou não (DARWIN, 2009).

A linguagem verbal e não verbal se complementa, de modo a tornar a comunicação humana mais rica, compreensível e acessível. Os seres humanos utilizam o corpo, gestos e a voz para reforçar, completar ou negar palavras para transmitir certas mensagens. A comunicação verbal é feita conscientemente, já o processo de comunicação não-verbal é silencioso (BIRCK, 2008).

O homem utiliza diversas formas de comunicação verbal e não-verbal para expressar suas emoções. A comunicação não verbal é contínua e ininterrupta, isso revela quais emoções internas as pessoas estão querendo estão sentindo. Um dos mais importantes meios de comunicação são as expressões faciais (BIRCK, 2008).

A psicoterapia utiliza a leitura corporal para designar estudos e práticas que tratam a utilização de elementos para análise do indivíduo, de forma a avaliar além do que é expresso de forma vocalizada. As expressões corporais podem ser empregadas de forma reveladora nas relações de comunicação entre as pessoas, o corpo pode expor 'verdades', favorecer ou dificultar o entendimento do que se deseja comunicar ou até reforçar ideias. Ao decorrer de uma conversa é importante observar o olhar, o sorriso, e o posicionamento dos membros, já que estas posturas e expressões corporais carregam um significado, que podem ajudar na

leitura da comunicação interpessoal, intrapessoal e intergrupal (GOIS et al., 2011).

A comunicação não-verbal pode ser usada de forma consciente e estratégica, no entanto, normalmente esse tipo de comunicação acontece de forma involuntária. Esse tipo de comunicação pode auxiliar a decifrar a verdadeira intenção do locutor, através da avaliação de contradições entre as falas e os gestos. Já que normalmente as pessoas não têm consciência sobre sua postura. Os gestos de comunicação não-verbal são frequentemente utilizados como uma forma de sustentar uma conversa entre duas pessoas, onde sem estímulos e comportamentos de retorno, a conversa fatalmente acaba (BIRCK, 2008).

Alguns tipos de comunicação tendem a revelar as personalidades internas e as emoções do ser humano, mesmo que de forma indireta. Algumas emoções com medo, honestidade, nervosismo, alegria etc. podem ser reconhecidos de forma independente, sem levar em consideração a aparência por meio das roupas. Nem sempre existe uma clara distinção entre aparência e linguagem corporal, estes trazem diferentes tipos de informação, porém ambas são igualmente importantes (BIRCK, 2008).

2.2 Expressões faciais

As expressões faciais são uma das formas que o ser humano utiliza para externalizar suas emoções, elas estão ligadas diretamente ao que o corpo sente. Essas expressões podem ser definidas como voluntárias, que podem ou não serem artificiais, ou involuntárias de modo que o agente não tem controle sobre esse tipo de externalização, tem se que, esse tipo de emoção demonstra a verdadeira intenção de quem a produz (LANGER e BARBOSA, 2020).

A face é a região corporal com maiores recursos para expressão. O conjunto de músculos que compõem o rosto humano são essenciais para a externalização das expressões emocionais. Os músculos envolvidos no desenvolvimento das expressões faciais são os músculos das sobrancelhas, da testa, das pálpebras, do pescoço e principalmente os concentrados na região oral, que possuem uma movimentação, um grau de liberdade elevado (MIGUEL, 2015).

A teoria sobre emoções difundida por (EKMAN, 2011), busca construir um modelo de emoções básicas, as quais possuem as mesmas manifestações faciais em diferentes culturas, ou seja, são consideradas assim universais. Em seus estudos com diferentes povos, com uma ampla diversidade cultural e de localidades distintas do mundo, notou-se então uma mesma forma de exibir essas expressões, desde criança até idosos. Com isso foram definidos uma lista de sete conjuntos de emoções, que possuem expressões faciais universais, sendo essas:

alegria, surpresa, medo, aversão, raiva, tristeza e desprezo.

> Por mais preconceituoso que seu subconsciente possa ser, ele dificilmente elege apenas um único gesto. Ao contrário, alimenta-se de uma série complexa de mensagens, conhecida como conjuntos de sinais. Da próxima vez que se encontrar com algum desconhecido, tente lembrar o que você pensou sobre essa pessoa quando a viu pela primeira vez. Se achar mais fácil poderá fazer essa experiência usando uma foto de revista. A parte lógica do seu cérebro provavelmente lhe dirá que você estava com a mente totalmente livre de preconceitos. Se tivesse que fazer uma lista, ela provavelmente incluiria o contato visual e o olhar, a expressão facial, os gestos e a postura, o tom da voz, o comportamento espacial, o contato físico a aparência e o modo de se vestir. Tudo isso compõe a nossa linguagem corporal individual (JAMES, 2008, p. 11).

Diversos métodos de reconhecimento facial avaliam as características do rosto humano, como por exemplo, posição das sobrancelhas, curvatura da boca, ou presença de sorriso. Isso permite que o ser humano expresse diversas emoções, através de gestos e ações musculares que caracterizam e diferenciam cada uma das emoções. Ao reconhecer gestos que caracterizam determinadas emoções as máquinas podem agir de forma específica e interagir com o usuário, com isso decisões que o ser humano não seria capaz de perceber podem ser tomadas pelas máquinas para melhorar a condição de vida do usuário (URIBE-QUEVEDO e REIS, 2015).

Existem mais de uma técnica que pode ser utilizada para a detecção e reconhecimento de rostos a partir de imagens. Essas técnicas podem ser baseadas em “regras de identificação de um típico rosto humano”, “características que são invariantes no rosto, independente das variações como posição do rosto e luminosidade”, “padrões modelados utilizando características armazenadas”, “métodos baseados na aparência aprendida e reconhecida pelo modelo” (ALBUQUERQUE, SANTIAGO e MOREIRA, 2014).

2.3 Microexpressões faciais e FACS

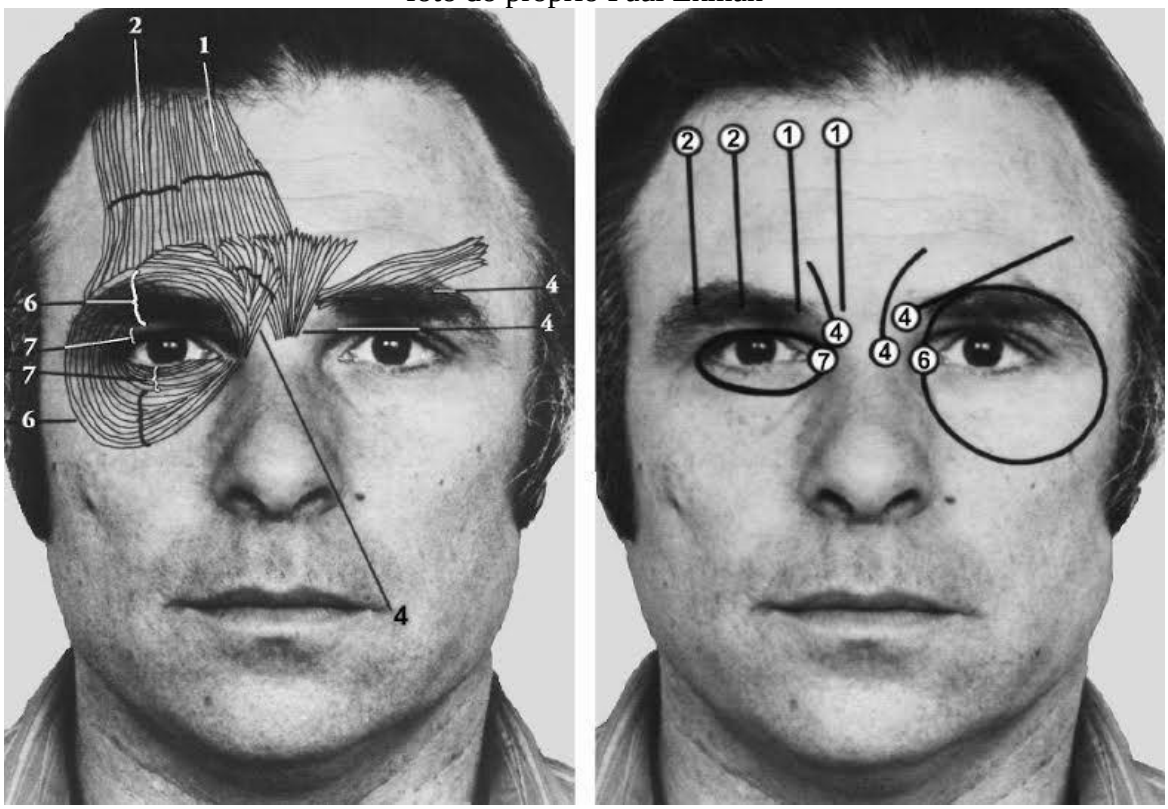
O estudo de microexpressões faciais como uma forma de interpretar e classificar emoções não é algo recente, desde a pesquisa realizada por Darwin, muitos autores discutiram inúmeras teses sobre esse tema. Muitos autores lançaram diferentes teses questionando a veracidade das microexpressões e suas características, mas foi somente com os estudos de Ekman que muitas dessas questões foram respondidas (GONÇALVES e PEPPI, 2019).

No entanto, Paul Ekman não foi o primeiro a estudar as microexpressões faciais, estas foram na verdade descobertas por Haggard e Issacs, em 1966. Essa descoberta foi feita digitalizando filmes cinematográficos de psicoterapia em busca de indícios de comunicação não verbal entre terapeuta e paciente. Na época a descoberta foi nomeada como “micro

momentâneas expressões faciais", essa descoberta foi apresentada ao mundo na publicação do artigo "*Micromomentary facial expressions as indicators of ego mechanisms in psychotherapy*" (GONÇALVES e PEPPI, 2019).

Ekman iniciou suas pesquisas em 1968, mas foi somente em 1978 que Ekman, juntamente com Wallace V. Friesen, desenvolveu a maior e mais importante ferramenta de identificação de microexpressões, que ficou conhecido como Facial Action Coding System (FACS). Esta metodologia científica permite identificar os músculos envolvidos na expressão e mensurar a intensidade do movimento muscular (GONÇALVES e PEPPI, 2019).

Figura 1 – Ilustração dos músculos e Unidades de Ação da *upper face*, ilustrada com a foto do próprio Paul Ekman



Fonte: "*Muscles underlying upperface Action Units".* Ekman, Friesen e Hager, 2002, p.15

Os movimentos musculares podem ser codificados através de algo que Ekman como *Action Unit* (Unidade de Ação) e *Action Descriptor* (Descritor de Ação). Cada *Action Unit* (AU) é relacionada com um movimento muscular específico. Sendo assim todo movimento facial pode ser codificado em forma de AU. É possível fazer inferência de sua atividade muscular e sua intensidade de ação (CARVALHO, 2017).

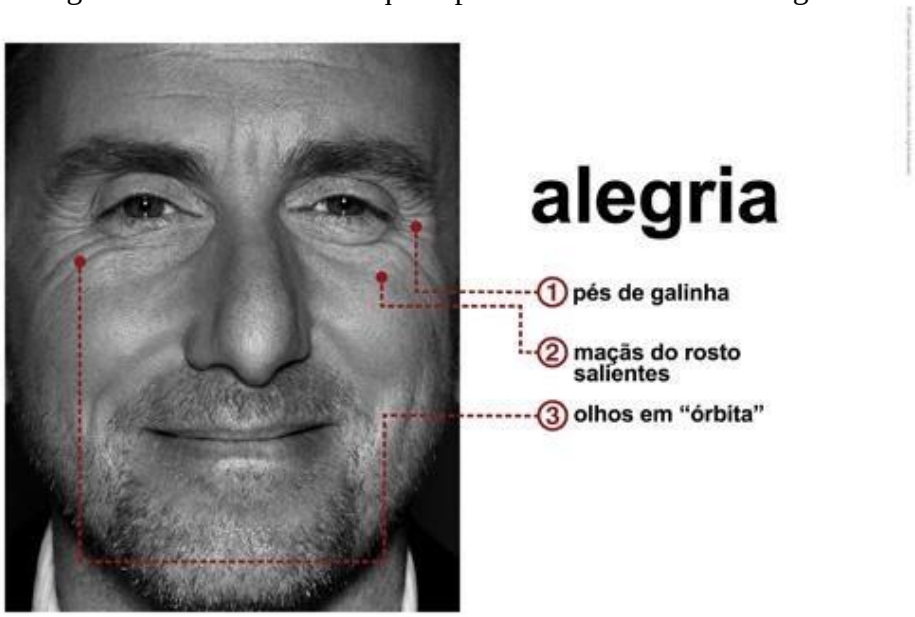
Tanto na pesquisa de Darwin, quanto na pesquisa de Ekman realizada na Nova Guiné, foram identificadas somente seis emoções universais: alegria, raiva, tristeza, medo, surpresa, aversão/nojo. Por volta do ano 2000 após pesquisas realizadas por Ekman foi adicionada uma

sétima emoção universal, o desprezo (GONÇALVES e PEPPI, 2019).

2.3.1 Alegria

A alegria é a emoção mais contagiante, ela é ativada por um estado global de satisfação, pela concretização de objetivos ou pela criação de conexões e interação com algo ou alguém. Essa expressão está relacionada diretamente com sensações de excitação. É uma emoção diretamente positiva, pelo fato da pessoa que a vivenciar gozar de efeitos biológicos positivos. A alegria verdadeira se caracteriza pelos olhos bem brilhantes com rugas nos seus extremos formadas pela contração dos músculos ao redor dos olhos, o *Orbicularis oculipar sorbitalis* e os extremos da boca se levantam formando assim, um sorriso na face (ROBERTO e LUIGI, 2017).

Figura 2 – Características que representam o estado de Alegria



Fonte: Fontoura, 2019

2.3.2 Surpresa

Essa é a emoção mais rápida de todas. Seu despertar é sempre que recebe um estímulo súbito ou inesperado, por isso ela só dura poucos segundos, caso ela se estenda por muito tempo, é indicativo que o indivíduo está mentindo, fingindo estar surpreso com algo não causou a emoção verdadeiramente. A surpresa é considerada uma emoção de transição, ou seja, onde o campo demora alguns segundos para entender o que está ocorrendo, absorver e processar o estimo, depois disso o corpo pode transacionar para outra emoção como, raiva, tristeza, alegria ou até mesmo nenhuma emoção. Em relação à expressão facial, a surpresa é caracterizada pelo relaxamento total do tônus muscular facial, os olhos ficam bem abertos, as sobrancelhas ficam levantadas, a mandíbula solta e a boca entreaberta (EKMAN, 2011).

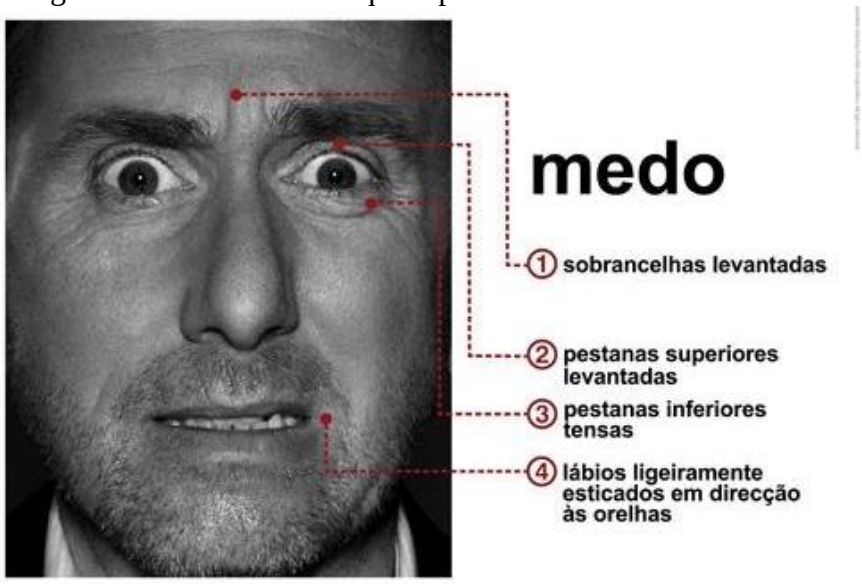
Figura 3 – Características que representam o estado de Surpresa

Fonte: Fontoura, 2019

2.3.3 Medo

Está emoção é focada diretamente na sobrevivência humana. Do ponto de vista da psicologia, o medo é um estado afetivo e emocional, necessário para os organismos se adaptar ao meio. É impulsionada por uma percepção de ameaça ao bem-estar físico ou psicológico do indivíduo. Suas características da microexpressão facial são os olhos abertos, com as pálpebras superiores erguidas, sobrancelhas tensas, levantadas e juntas, e pôr fim a mandíbula solta, com a boca esticada horizontalmente. Normalmente os indivíduos tentam esconder a emoção de medo, suprimindo a reação ou a mascarando com outra emoção (ROBERTO; LUIGI, 2017).

Figura 4 – Características que representam o estado de Medo

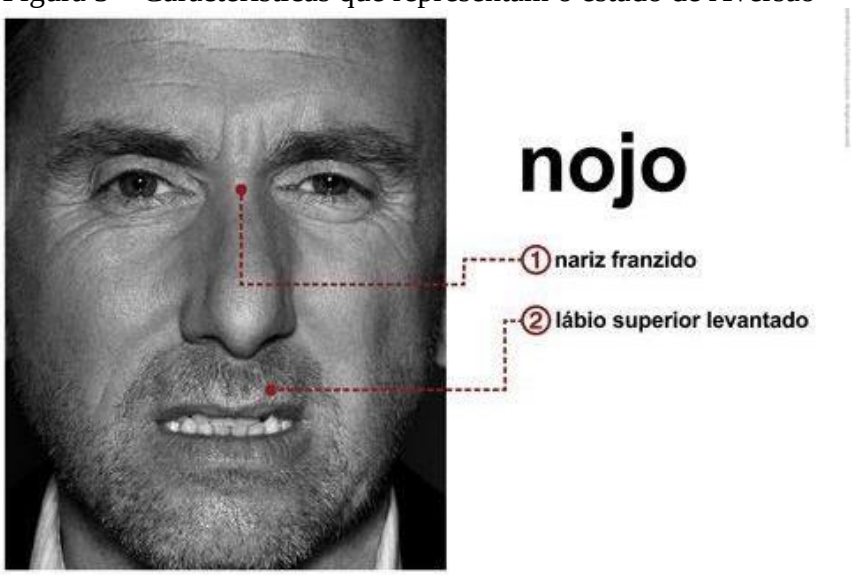


Fonte: Fontoura, 2019

2.3.4 Aversão

Esta emoção é produzida quando um dos cinco sentidos percebe algo repugnante, ofensivo ou imoral. Segundo Ekman (2011), as emoções de nojo e aversão apresentam o mesmo display facial e são consideradas uma única emoção pelo fato de causarem os mesmos efeitos no corpo de quem vivência a emoção. A emoção de nojo pode ser definida como uma repulsa por estímulos físicos, como cheiros desagradáveis, sujeira ou gostos, é possível também demostrar essa emoção sem o estímulo físico através apenas com a ideia, visão ou som deles a pessoa também experiência sensações de repugnância, característica do nojo. A diferença em relação à aversão é que a repugnância é causada por uma pessoa, seja por seu físico, por sua personalidade, traços de comportamento ou até mesmo determinada ações. As características principais que definem o display facial tanto para o nojo quanto para a aversão, são o nariz enrugado produzindo rugas transversais no mesmo e o lábio superior elevado (EKMAN, 2011).

Figura 5 – Características que representam o estado de Aversão

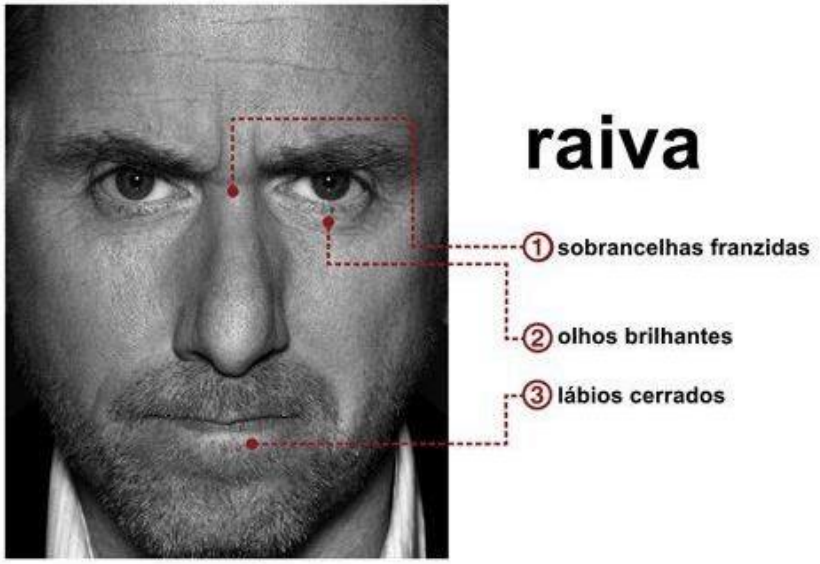


Fonte: Fontoura, 2019

2.3.5 Raiva

Está é a emoção mais perigosa e assustadora. Ela é desencadeada quando alguém age de forma injusta ou impede a concretização de um objetivo importante. Sua detecção se dá principalmente na parte superior da face, onde testa e sobrancelhas fazem o movimento de franzir, com um olhar fixo e as sobrancelhas ficam para baixo de encontro uma com a outra. Lábios então ficam tenso, pressionados um contra o outro ou abertos mostrando os dentes (rosnar), o queixo também pode se levantar com o intuito de demonstrar agressividade. Essa emoção possui o display facial diretamente ligado a agressão ou ataque (ROBERTO; LUIGI, 2017).

Figura 6 – Características que representam o estado de Raiva



Fonte: Fontoura, 2019

2.3.6 *Tristeza*

Esta é uma das emoções que possuem a duração mais prolongada, de acordo com a intensidade do fator externo que a provocou, pode gerar um display facial tão forte e potente que se assemelha à face de dor. As características principais podem ser definidas com a presença de principais do AU1, que representa a movimentação do musculo frontal – parte interna, responsável por erguer a parte interna das sobrancelhas, com isso as sobrancelhas ficam baixas e se juntam em direção ao centro, produzindo rugas horizontais no centro da testa. Juntamente com ele podem aparecer outros AUs, como o AU15, que é o musculo depressor do ângulo da boca, o que é responsável pela queda dos cantos da boca, resultando no arco labial para baixo (EKMAN, 2011).

Figura 7 – Características que representam o estado de Tristeza

Fonte: Fontoura, 2019

2.3.7 Desprezo

Diferente das outras emoções aqui apresentadas, o desprezo é uma emoção secundária sendo composta por um misto das emoções primarias nojo e raiva. Ao contrário das demais emoções, o desprezo é fruto da racionalidade humana. O desprezo é a única emoção assimétrica, parece um sorriso para si mesmo, pois só uma parte da boca é levantada. Isso não descreve um sorriso, e sim um sinal de que a pessoa está tendo pensamentos sobre a sua superioridade em relação a outras pessoas. Suas características podem ser notadas na parte inferior do rosto, onde um meio sorriso ou um lado da boca é levantado (ROBERTO; LUIGI, 2017).

Figura 8 – Características que representam o estado de Desprezo

Fonte: Fontoura, 2019

2.4 Redes Neurais Artificiais

As Redes Neurais Artificiais (RNAs) é um modelo computacional ligeiramente inspirado nos sistemas de redes neurais biológicas do cérebro humano e dos animais, que buscam reproduzir a forma que o cérebro processa informações. A própria rede neural não é um algoritmo, mas sim a estrutura de muitos algoritmos dede aprendizagem de máquina, utilizada para definir e processar a entrada de dados. Estes sistemas aprendem a realizar tarefas levando em consideração exemplos, sem serem programados com regras de tarefas especificas. Por exemplo, no reconhecimento de imagem, as redes neurais podem aprender a identificar imagens que contenham cachorros analisando exemplos que tenham sido rotulados como 'cachorro' ou 'não cachorro', ou seja divididos em classes de reconhecimento, em seguida, pode-se usar o resultado para identificar cachorros em outras imagens. As redes neurais fazem isso sem nenhum conhecimento prévio dos cachorros, por exemplo, de que eles

têm pelo, cauda, rosto de cachorro. Em vez disso, eles identificam automaticamente as características do material de aprendizado com os quais estão lidando, ou seja, esse método busca simular a funcionalidade do cérebro de aprendizagem por experiência (SILVA; SPATTI; FLAUZINO, 2016).

2.4.1 Neurônio Artificial

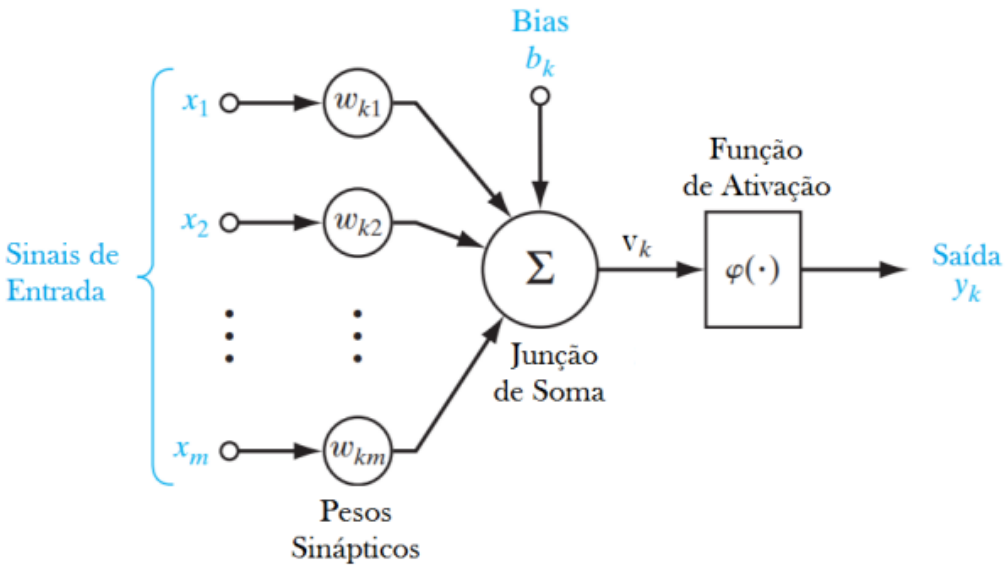
Devido ao fato de a estrutura das redes neurais artificiais ter sido desenvolvida a partir de modelos conhecidos de sistemas nervosos biológicos e do próprio cérebro humano, as unidades processadoras ou elementos computacionais, denominadas de neurônios artificiais, são modelos simplificados dos neurônios biológicos. Os neurônios artificiais utilizados nos modelos de RNAs são não-lineares, fornecem saídas tipicamente contínuas e realizam funções simples (SILVA; SPATTI; FLAUZINO, 2016).

O modelo de neurônio mais simples e que engloba de maneira simplificada os componentes e o funcionamento de um neurônio biológico, isto é, paralelismo e conectividade, foi proposto por McCulloch e Pitts em 1943, o modelo MCP, A Figura 9, exemplifica o modelo.

Neste modelo os sinais de entrada (X_m), que podem ser provenientes de outros neurônios, dentre os vários estímulos recebidos, alguns excitarão mais e outros menos o neurônio receptor, essa medida de quão excitatório é o estímulo é representado nesse modelo através dos pesos sinápticos, quanto maior o valor do peso, mais excitatório é o estímulo. Os pesos sinápticos são representados por W_{km}, onde k representa o índice do neurônio em questão e p se refere ao terminal de entrada da sinapse à qual o peso sináptico se refere (FURTADO, 2019).

A soma é representada por uma composição de dois módulos, o primeiro é o somatório dos estímulos multiplicado pelo seu fator excitatório, e posteriormente uma função de ativação, que definirá com base nas entradas e pesos sinápticos, qual será a saída do neurônio (Y_k). Assim como no modelo biológico, o estímulo pode ser excitatório ou inibitório, representado pelo peso sináptico positivo ou negativo respectivamente (BARCELOS; TOSHIMITSU, 2018).

Figura 9 – Modelo de Neurônio Artificial, segundo McCulloch-Pits



Fonte: Adaptada de (HAYKIN, 2008)

Em termos matemáticos o neurónio artificial pode ser representado como

$$u_k = \sum_{j=0}^{m} W_{kj} * x_j \quad (1)$$

onde m é o número de sinais de entrada incidentes no neurônio k e posteriormente a aplicação da função de ativação

$$y_k = \varphi(u_k + b_k) \quad (2)$$

Basicamente a função de ativação recebe o valor provido pela junção aditiva, denominado por (HAYKIN, 2000) como "campo local induzido", atribuindo 1, se o campo local induzido daquele neurônio for não-negativo e 0 caso for negativo. A função de ativação adotada no modelo é uma função degrau definida como

$$\varphi(\mathrm{u}) = \begin{cases} 1\,, & se\ u \geq 0 \\ 0\,, & se\ u < 0 \end{cases} \quad (3)$$

2.5 OpenCV

O OpenCV (Open Source Computer Vision Library) é biblioteca de software aberto, suas principais aplicações são em projetos de visão computacional e *machine learning*. Desenvolvida inicialmente pela Intel e patenteada pela BSD License, seu principal objetivo é o de tornar a visão computacional mais acessível para desenvolvedores, e toda a comunidade acadêmica. A versão mais recente desta biblioteca é a OpenCV 4, a qual será utilizada no desenvolvimento desse projeto (OPENCV TEAM, 2020; VILLÁN, 2019).

Sendo dividido em vários módulos, onde cada módulo busca solucionar um grupo de problemas de visão computacional. A biblioteca tem mais de 2.500 algoritmos otimizados, que englobam um conjunto amplo de algoritmos de visão computacional *machine learning*. Esses algoritmos podem ser utilizados para diversas aplicações como, por exemplo, classificar ações humanas, rastrear tráfego de carros, localizar rostos etc. (OPENCV TEAM, 2020; VILLÁN, 2019).

Por se tratar de um software livre o OpenCV possui uma grande quantidade de usuários, além de fóruns onde os usuários podem discutir problemas e procurar soluções para os problemas do dia a dia, compartilhando experiências entre os usuários. A biblioteca possui interfaces para diversas linguagens de programação C ++, Python, Java e MATLAB e suporta os principais sistemas operacionais Windows, Linux, Android e Mac OS (OPENCV TEAM, 2020).

Segundo Kaehler e Bradeski (2017), a referência da biblioteca OpenCV e dividida em várias seções, onde cada um pertence a um módulo na biblioteca. A lista de módulos que compõem a biblioteca evoluiu com o tempo. Cada função da biblioteca faz parte de um módulo. Deste modo, de acordo com a documentação do OpenCV os principais módulos que compõem a biblioteca são:

core

Este módulo contém todos os tipos básicos de objetos e suas operações básicas.

imgproc

O módulo de processamento de imagem, contém transformações básicas em imagens, incluindo filtros, operadores convolucionais semelhantes e diversas outras funções de processamento de imagens.

highgui

Este módulo foi dividido em imgcodecs, videoio e highgui a partir do OpenCV 3.0. De modo geral este módulo contém funções de interface de usuário, que podem ser usadas para exibir imagens ou obter entradas simples do usuário. O módulo imgcodecs contém funções para ler, codificar e salvar imagens. O módulo videoio contém funções de leitura e escrita de vídeo ou sequência de imagens, a partir de arquivos de vídeo, sequência de imagens, ou captura direta da câmera. Por fim o módulo highgui permite criar e manipular janelas que podem exibir imagens e “lembrar” seu conteúdo, sem precisar lidar com eventos de redesenho do sistema operacional, além disso, permite manipular eventos simples do mouse e comandos do teclado.

video

A biblioteca de vídeo contém funções de ler e gravar streams de vídeo.

calib3d

Este módulo contém implementações de algoritmos para calibrar câmeras, bem como matrizes estéreo ou multicâmeras.

features2d

Este módulo contém algoritmos para detectar, descrever e combinar recursos de pontos-chave.

objdetect

Este módulo contém algoritmos para detectar objetos específicos, como rostos. É possível treinar os detectores para detectar outros objetos também.

dnn

Esse módulo possui a funcionalidade projetada apenas para cálculos de aprovação, assim sendo, para teste de rede. Não há a princípio suporte para treinamento de rede. Este módulo contém uma API para criação de novas camadas, API para construir e modificar redes neurais abrangentes, entre outras funções.

ml

A biblioteca de Machine Learning ou, traduzindo para o português, aprendizado de máquina é, na verdade, uma biblioteca completa, e contém uma ampla variedade de algoritmos de aprendizado de máquina, implementados de forma a trabalhar com as estruturas de dados naturais do OpenCV.

flann

FLANN (Fast Library for Approximate Nearest Neighbors) é uma biblioteca que contém uma coleção de algoritmos otimizados para busca rápida do vizinho mais próximo em grandes conjuntos de dados e para recursos dimensionais elevados. Esta biblioteca contém métodos que provavelmente não serão usados diretamente, mas que são usados por outras funções em outros módulos, para fazer pesquisas pelos vizinhos mais próximos em grandes conjuntos de dados.

photo

Este é um módulo que contém ferramentas úteis para fotografia computacional.

stitching

Todo este módulo implementa um sofisticado canal de stitching de imagens. Essa classe possibilita configurar e/ou remover algumas etapas de execução de uma classe, ou seja, o

pipeline de stitiching de acordo com as necessidades particulares. Esta é uma nova funcionalidade na biblioteca, mas é uma área onde se espera um crescimento futuro.

gapi

Conhecido como OpenCV Graph API, ou seja, uma API Gráfica do OpenCV. Esse módulo é voltado para tornar o processamento regular de imagens rápido e portátil. Para isso se introduziu um novo modelo de execução baseado em gráficos. Este módulo atua como uma estrutura em vez de um algoritmo de visão computacional específico.

2.6 TensorFlow

Criado pela equipe do **Google Brain**, o TensorFlow é um framework de código aberto para *machine learning*, focada em computação numérica, sua biblioteca é compatível com Python e JavaScript. Ele possui um ecossistema abrangente e flexível de ferramentas, bibliotecas e recursos da comunidade que são utilizados por pesquisadores e desenvolvedores criar e implementar aplicativos utilizando *machine learning* (ABADI et al., 2015).

O TensorFlow reúne uma série de modelos e algoritmos de *machine learning* e *deep learning*, e os torna úteis por meio de uma metáfora comum. Ele utiliza o Python para fornecer uma API de front-end conveniente para criar aplicativos com a estrutura, enquanto executa esses aplicativos em C++ de alto desempenho (ARAÚJO et al., 2017).

O TensorFLow pode treinar e executar redes neurais profundas para diversas linhas de pesquisa, como por exemplo, reconhecimento de imagens, incorporação de palavras, modelos sequência a sequência para tradução automática. Ele é capaz de suportar previsões de produtos em escala, com os mesmos modelos usados para treinar. Ele pode ser executado utilizando diversas plataformas e arquiteturas, incluindo CPUs, GPUs e as até TPUs. Uma das principais plataformas do mercado para criação de redes neurais e *deep learning.* Com ele, é possível agilizar e facilitar o processo de obtenção de dados, treinar modelos, realizar predições e refinar resultados futuros (ARAÚJO et al., 2017).

2.6.1 Arquitetura do TensorFlow

O TensorFlow utiliza uma API rica em linguagem Python, que busca um desenvolvimento simplificado para o programador final. Sua execução se passa sobre uma aplicação de alta performance escrita em C/C++. Sua arquitetura principal pode ser dividida em três partes principais, sendo elas (ABADI et al., 2016):

- Pré-processamento dos dados;
- Construção dos modelos;

● Treinamento e estimativas do modelo criado.

> O Tensor flow fornece uma abstração de programação baseada em fluxo de dados simples que permite aos usuários implantar aplicativos em clusters distribuídos, estações de trabalho locais, dispositivos móveis e aceleradores personalizados. Uma interface de script de alto nível envolve a construção de gráficos de fluxo de dados e permite que os usuários experimentem diferentes arquiteturas de modelo e algoritmos de otimização sem modificar o sistema central (ABADI et al., 2016).

Os dados de entrada de uma aplicação utilizando o TensorFlow é um array multidimensional, chamado de *tensors*. Utilizando o array de entrada são construídos fluxogramas das operações que serão realizadas. Sendo assim, os dados de entrada entram em uma extremidade, fluem por um número determinado de operações e saem na outra extremidade. Esse comportamento caracteriza o nome TensorFlow do framework (ABADI et al., 2015).

2.6.2 Componentes do TensorFlow

Os principais componentes em um sistema TensorFlow são o cliente, que usa a interface de sessão para se comunicar com o mestre, que por sua vez executa um ou mais processos de trabalho, com cada processo de trabalho responsável por arbitrar o acesso a um ou mais dispositivos computacionais e para executar nós de gráfico nesses dispositivos conforme as instruções do mestre. O TensorFlow possui duas arquiteturas principais de execução, sendo elas implementações locais com um único processo ou distribuídas com mais de um processo. A implementação local é usada quando o cliente, o mestre e o trabalhador são executados em uma única máquina no contexto de um único processo do sistema operacional. A implementação distribuída compartilha a maior parte do código com a implementação local, mas o estende com suporte para um ambiente onde o cliente, o mestre e os trabalhadores podem estar em processos diferentes em máquinas diferentes (ABADI et al., 2015).

2.6.2.1 Dispositivos

Os dispositivos são o coração computacional do TensorFlow. Cada dispositivo possui um tipo e um nome. A interface do TensorFlow possui implementações de dispositivos como núcleos de CPUs ou placas GPUs, além disso novas implementações para outros tipos de dispositivos podem ser fornecidas por meio de um mecanismo de registro. Cada objeto do dispositivo é responsável por gerenciar a alocação e desalocação da memória do dispositivo e por organizar a execução de quaisquer kernels solicitados por níveis superiores na implementação do TensorFlow (ABADI et al., 2015).

2.6.2.2 Tensor

O nome TensorFlow é derivado diretamente da sua ideia central: o Tensor. No

framework, todas as operações de computação envolvem *tensors*. Um tensor é um array multidimensional tipado. Existe uma variedade de tipos de elementos tensores, incluindo inteiros assinados e não assinados variando em tamanho de 8 bits a 64 bits, IEEE float e tipos duplos, um tipo de número complexo e um tipo de string (uma matriz de bytes arbitrária). Um tensor é um vetor ou matriz de N dimensões que representa todos os tipos de dados. Um tensor pode ser originado a partir de uma entrada de dados ou o resultado de uma operação (ABADI et al., 2015).

2.6.2.3 Diagramas

TensorFlow faz uso de um framework específico para gerenciar os diagramas de execução. Esses diagramas reúnem um conjunto de operações realizadas de forma sequencial, agregando e descrevendo todos os cálculos relizados durante o treinamento. Algumas das principais vantagens de utilizar esses diagramas são, a possibilidade de executar as aplicações em diversas arquiteturas dentre elas, CPUs, GPUs e até mesmo em sistemas operacionais mobile; possuir portabilidade, ou seja, é possível preservar os cálculos para uso imediato ou posterior. O gráfico pode ser salvo para ser executado no futuro; todos os cálculos no diagrama são feitos conectando os tensors em conjunto.

2.6.2.4 Keras API

De acordo com a documentação oficial do TensorFlow, “Keras é uma API de alto-nível que é utilizada para construir e treinar modelos de aprendizado profundo. É usada para prototipagem rápida, pesquisa de ponta e produção, com 3 vantagens principais: Fácil de usar, Os modelos modulares e compostos, Fácil de estender.” O Keras foi inicialmente desenvolvido para pesquisa, com o objetivo de permitir uma rápida experimentação aplicações de aprendizagem profunda. O Keras prioriza a experiência do desenvolvedor por isso possui uma interface simples, projetada para ser de uso fácil e comum. Possui um mecanismo de feedback de erros, facilitando o desenvolvimento por parte do programador final (CHOLLET, 2020).

O uso da ferramenta não possui uma única maneira "verdadeira" de construir e treinar modelos. Em vez disso, é permito que o usuário utilize uma ampla gama de fluxos de trabalho diferentes, do nível muito alto ao nível muito baixo, correspondendo a diferentes perfis de usuário. Os modelos são constituídos interligando blocos configuráveis, com baixa restrição, permitindo criar modelos complexos sem que isso se torne um grande conglomerado de código para o desenvolvedor. A API fornece várias funções de ativação e indicadores, que podem ser usados para otimização de redes neurais de aprendizado profundo, além de permitir

que seja feito medições quanto a performance e precisão dos modelos criados (CHOLLET, 2020).

2.7 Redes Neurais Convolucionais

O termo Redes Neurais Convolucionais vem do inglês Convolutional Neural Network (CNNs) que se trata de Redes Neurais com camadas, e foi projetada especialmente para uso de dados bidimensionais, tais como imagens e vídeos. CNNs são redes neurais que utilizam convolução em pelo menos uma camada (GOODFELLOW; BENGIO; COURVILLE, 2016).

Segundo Dumoulin e Visin (2016), independente da dimensionalidade da entrada ou do tipo dos dados, podendo ser eles uma imagem, um som, ou uma coleção desordenada de recursos, a representação pode sempre ser achatada em um vetor. E por mais que possuam diferenças peculiares, essas entradas compartilham propriedades importantes tais como: são armazenadas como matrizes multidimensionais; apresentam um ou mais eixos para cada atributo, como por exemplo, imagens possuem atributos como eixo de altura e eixo de largura.

Um eixo de canal é usado para acessar diferentes aspectos dos dados. Por exemplo, em uma imagem colorida, existem canais de cores, como vermelho, verde e azul (RGB), já em uma trilha de áudio estéreo existem canais da direita e esquerda. No entanto, as características específicas que definem cada um desses atributos não são exploradas quando uma transformação é aplicada. Isso ocorre porque todos os eixos são processados da mesma maneira e a informação topológica não é levada em consideração. Mesmo assim é possível aproveitar a estrutura implícita dos dados, para resolver tarefas de visão computacional e reconhecimento de voz, acarretando o uso das convoluções discretas. A convolucão discreta é uma transformação linear que preserva as noções de ordenação. É seletivo, pois apenas algumas unidades de entrada contribuem para a saída, e reutiliza parâmetros, ou seja, os mesmos pesos são aplicados a vários locais na entrada (DUMOULIN; VISIN, 2016).

2.7.1 Camada Convolucional

A camada de uma rede neural convolucional é uma rede que compartilha os seus parâmetros por toda camada. A primeira camada em uma rede neural convolucional é a camada de convolução, tratando a entrada dos dados. No caso de imagens, cada exemplo possui uma largura, uma altura e uma profundidade que é representada pelos canais de cor (vermelho, verde e azul). Uma convolução consiste em coletar um trecho da imagem de exemplo e aplicar sobre esta lista um *kernel* ou um filtro de dimensão igual ou menor que a imagem. Isso é feito deslizando a partir do canto superior esquerdo, deslocando para a direita

na imagem, sem alterar os pesos e montando as saídas verticalmente em uma coluna de profundidade definida partir da dimensão do filtro. No final será montada uma nova imagem de largura, altura e profundidade diferente. Essa imagem é um conjunto de mapas de características da imagem original (NIELSEN, 2015).

2.7.2 Camada de Pooling

Geralmente, após a acamada convolucional tem-se a camada de *Pooling*, também conhecida como como camada de agrupamento. Seu principal objetivo é reduzir gradualmente a dimensão espacial de entrada, consequentemente a redução diminui o custo computacional da rede e evita *overfitting*. Essa função ajuda a fazer com que a representação se torne invariante para pequenas traduções da entrada. Uma função *pooling* substitui a saída da rede em um determinado local através de um resumo estatístico das saídas mais próximas, geradas pelas camadas convolucionais (GOODFELLOW; BENGIO; COURVILLE, 2016).

A forma mais comum de funcionamento da camada de *pooling* consiste em substituir os valores de uma região pelo maior valor dentro de um conjunto de valores presentes em uma região. Essa operação é conhecida como *max pooling*, ela é útil para eliminar valores irrelevantes, reduzindo a dimensão da representação dos dados e acelerando a computação necessária para as próximas camadas, além de criar uma invariância a pequenas mudanças e distorções locais. Isso facilita com que a rede encontre alguma característica ao invés de saber onde essa característica está (GOODFELLOW; BENGIO; COURVILLE, 2016).

2.7.3 Camada Totalmente Conectada

A saída das camadas convolucionais e de *pooling* representam as características extraídas dos dados de entrada. O objetivo das camadas totalmente conectadas é utilizar essas características para classificar os dados de entrada de acordo com as classes pré-definidas. Essas camadas são iguais a uma rede neural artificial convencional que usa a função de ativação *softmax* na camada final. Essas camadas são formadas neurônios artificiais, e o termo “totalmente conectado” significa que todos os neurônios da camada anterior estão conectados a todos os neurônios da camada seguinte (KRIZHEVSKY; SUTSKEVER; HINTON, 2017).

Como mencionado anteriormente, a última camada da rede utiliza *softmax* como função de ativação. Essa função recebe um vetor de valores como entrada e produz a distribuição de probabilidade de a imagem de entrada pertencer a cada umas das classes na qual a rede foi treinada. A soma de todas as probabilidades é igual a 1 (HAYKIN, 2008).

Uma técnica que também é bastante utilizada entre as camadas totalmente conectadas, com o objetivo de reduzir o tempo de treinamento e evitar *overfitting*, é conhecida como

dropout. Essa técnica remove, aleatoriamente a cada iteração de treinamento, uma determinada porcentagem dos neurônios de uma camada, adicionando-os na iteração seguinte. Essa técnica também confere à rede a habilidade de aprender atributos mais robustos, uma vez que um neurônio não pode depender da presença específica de outros neurônios (GOODFELLOW; BENGIO; COURVILLE, 2016).

2.7.4 Backpropagation

O *backpropation* é um dos algoritmos-chave de uma rede neural, ele é utilizado para ajustar pesos e *biases* para o treinamento de redes. Sem ele o processo de treinamento seria uma tarefa bastante complicada dada a disponibilidade computacional que existe hoje. Esse algoritmo foi desenvolvido para otimizar o custo computacional de treinamento das redes neurais. Ele basicamente retropropaga os valores das derivadas usando Programação Dinâmica a fim de calcular os gradientes em cada passo e evitando a complicação do cálculo de muitas derivadas (HAYKIN, 2008).

2.7.5 Bias

O bias, também conhecido como "viés", é um parâmetro constante da rede neural que é inserido junto às entradas. Sua principal função é orientar o aprendizado da rede neural e evitar que ela transmita valores zerados, fazendo com que a rede neural caia em mínimos locais e não consiga aprender corretamente. Ele atua aumentando ou diminuindo a entrada líquida da função de ativação, dependendo se é positiva ou negativa, respectivamente. Ele é tratado como uma entrada e possui pesos que podem ser atualizados pelo algortimo de atualização de pesos que estiver sendo utilizado. Na Figura 9, mostrada anteriormente, pode-se notar a representação do bias atuando sobre um único neurônio (HAYKIN, 2008).

2.7.6 Funções de Ativação

A principal finalidade das funções de ativação é converter um sinal de entrada de um neurônio, em uma rede neural artificial, para um sinal de saída. Essa saída será usada na próxima camada. Essas funções possibilitam que pequenas mudanças nos pesos e biases cause apenas uma pequena mudança na saída.

2.7.6.1 Step function

A primeira função de ativação que foi usada para redes neurais artificiais é chamada de *step function*, ou conhecida em portugues como, função degrau, foi criada por Oliver Heaviside (1850–1925). Ela recebe esse nomo porque funciona basicamente como se fosse um degrau, possuindo apenas dois estados. A *step function* é baseada em um *threshold*, se for maior que um valor definido ela é ativada, caso contrário, ela fica desativada, como mostrado

na equação 3. Um dos problemas da utilização dessa função em redes neurais é a mudança brusca de estado. Uma vez que uma pequena alteração na rede neural causará um efeito muito maior do que o esperado, já que a função trabalha apenas com 0 e 1.

2.7.6.2 Função Sigmóide

A curva da Função Sigmóide se parece com uma forma em S. Essa função foi desenvolvida principalmente para corrigir o problema da função degrau, onde se tem uma mudança brusca de estado. Deste modo, o principal benefício de utilizar a função sigmóide é a obtenção de uma certa suavidade nas alterações dos valores da rede neural. Não existe um resultado abrupto, já que ela trabalha em um intervalo entre 0 e 1, e não absolutamente 0 ou 1 (HAYKIN, 2000). Esses resultados são gerados a partir de uma função, descrita como:

$$\varphi(z) = \frac{1}{1+e^{-z}} \quad (4)$$

2.7.6.3 Função ReLU

ReLU é a sigla para Rectified Linear Unit, sendo a função de ativação mais usada em modelos de aprendizagem profunda. Seu funcionamento ocorre da seguinte forma, a função retorna 0 se receber qualquer entrada negativa, mas retorna um valor para qualquer entrada positiva. Então, pode ser escrito como

$$f(x) = max(0, x) \quad (5)$$

A função ReLU é convencionalmente usada como uma função de ativação para as camadas ocultas em uma rede neural profunda e não na camada de saída, pois ela cresce os valores infinitamente (KRIZHEVSKY; SUTSKEVER; HINTON, 2017).

2.7.7 Adam Optimizer Algorithm

Adam é um algoritmo de otimização que pode ser utilizado em vez do procedimento de descida de gradiente estocástico clássico. Na descida de gradiente estocástico mantém uma a taxa de aprendizado para todas as atualizações e a taxa de aprendizado não muda durante o treinamento, além disso a aprendizagem é mantida de para cada peso da rede e adapta conforme o aprendizado. O método utilizado pelo algoritmo Adam calcula as taxas de aprendizado de forma adaptativa, de maneira individual para diferentes parâmetros de estimativa (KINGMA; BA, 2014).

Os pesos da rede com a utilização do método Adam são atualizados de forma iterativa com base nos dados de treinamento, isso faz com que os pesos de atualização variem de acordo com os dados de treinamento da rede neural. As principais vantagens de se utilizar este método são: O método é fácil de implementar, é computacionalmente eficiente, tem poucos requisitos de memória, é invariável para redimensionamento diagonal dos gradientes , é

adequado para problemas que são grandes em termos de dados e / ou parâmetros, também é apropriado para objetivos não estacionários e problemas com gradientes muito ruidosos e / ou esparsos, além disso os hiperparâmetros têm interpretações intuitivas e normalmente requerem pouco ajuste (KINGMA; BA, 2014).

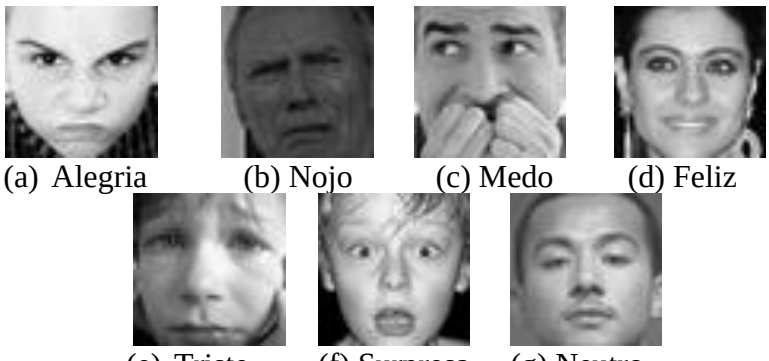
3 MATERIAIS E MÉTODOS

3.1 Banco de Imagens

A base de dados utilizada é conhecida como FER2013, essa base vem de um desafio de aprendizagem, especificamente, o desafio de reconhecimento de expressão facial a partir de imagens realizado pelo Kaggle. Os dados consistem em imagens de faces em tons de cinza, com dimensão de 48×48 pixels. O conjunto de imagens possui 28709 imagens de treinamento, 3589 imagens de teste público e outros 3589 exemplos para teste privado. Essas imagens de expressão facial são rotuladas com sete classes emocionais, ou seja, raiva, nojo, medo, feliz, triste, surpresa e neutro. A Figura 10 apresenta exemplos de cada classe de expressão do conjunto de dados FER2013 (CARRIER; COURVILLE, 2013).

Esta classificação é baseada nos primeiros estudos de Ekman, pode-se notar que não há no banco de dados a sétima emoção considerada posteriormente por Ekman como universal, rotulado como desprezo. Este desafio do *Kaggle* considerou somente seis classes emocionais, juntamente com uma classe rotulada como neutro, para compor seu dataset. As faces que compõem as imagens foram registradas automaticamente para que fiquem mais ou menos centralizadas e ocupem aproximadamente a mesma quantidade de espaço em cada imagem (CARRIER; COURVILLE, 2013).

Figura 10 – Exemplos de imagens do dataset do FER2013



(a) Alegria (b) Nojo (c) Medo (d) Feliz

(e) Triste (f) Surpresa (g) Neutro

Fonte: Dataset FER2013.

A quantidade de imagens presentes em cada uma das classes de classificação pode ser vista na Figura 11, pode-se perceber que a quantidade de imagens que representam a emoção

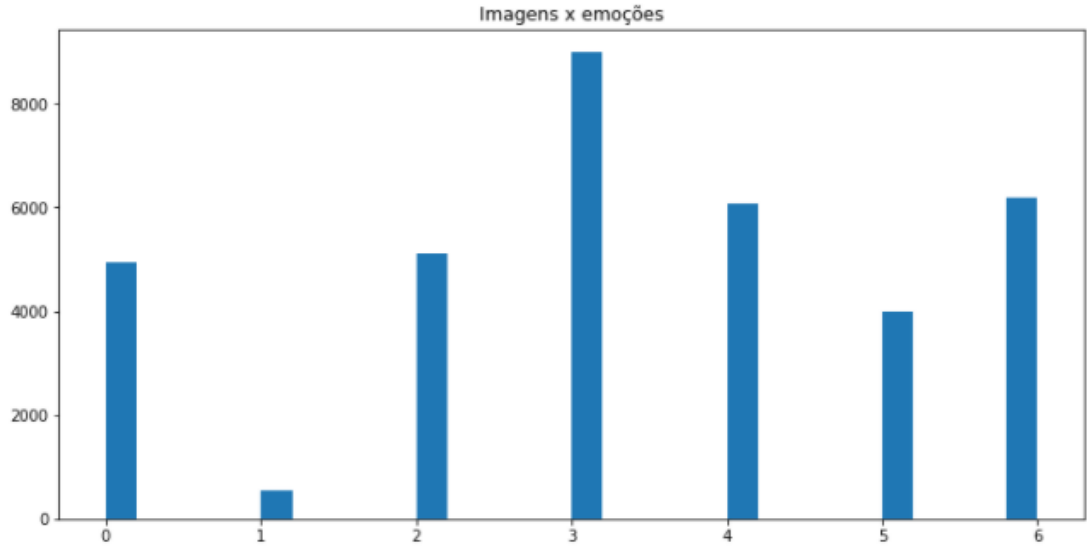
de ‘Feliz’ possui maior quantidade de imagens, em contrapartida a quantidade de imagens que representa a emoção de ‘Nojo’ possui bem menos imagens quando comparada as outras emoções. Isso deve ser levado em consideração já que a quantidade de imagens de uma emoção irá influenciar diretamente na capacidade de reconhecimento da mesma pela rede neural.

Figura 11 – Plot do gráfico com a quantidade de imagens para cada classe

```
plt.figure(figsize=(12,6))
plt.hist(data['emotion'], bins = 30)
plt.title('Imagens x emoções')

# Classes: ['Angry', 'Disgust', 'Fear', 'Happy', 'Sad', 'Surprise', 'Neutral']
```

```
Text(0.5, 1.0, 'Imagens x emoções')
```



Fonte: Autoria própria.

3.2 *Jupyter notebooks*

Os *Jupyter notebooks* são amplamente usados nas comunidades de ciência de dados aprendizagem de máquina, um notebook ou caderno em português, é um arquivo gerado pelo *Jupyter Notebook* (jupyter.org). Esses ‘cadernos’ combinam a capacidade de executar códigos, desenvolvidos utilizando a linguagem de programação Python, com recursos de edição de texto para realização de anotações. Dentro de um ‘caderno’ tem-se a possibilidade se dividir experimentos longos em pedaços, blocos, que podem ser executados de forma independente, isso torna o desenvolvimento interativo, ou seja, caso ocorra um erro, não é necessário executar novamente todo o código, mas somente os blocos que apresentarem problemas ou alterações.

3.3 *Google Colab*

O *Google Colaboratory*, ou *Google Colab* em sua nomenclatura mais curta, é um serviço com versão gratuita, para executar *Jupyter notebooks*, que não requer instalação e

funciona na nuvem. Sua versão gratuita fornece ao usuário acesso a um tempo de execução utilizando arquiteturas GPU ou TPU, porém essa disponibilidade é limitada. O Google Colab possui mais duas versões, Pro e Pro+, na versão Pro o usuário tem acesso a GPUs e TPUS mais rápidas, mais memória RAM e mais discos para maior armazenamento de código e variáveis, além de um tempo de execução mais longo e menos limites de tempo de inatividade. A versão Pro+ permite que as aplicações sejam executadas em segundo plano, mesmo após o fechamento do navegador, GPUs mais rápidas, mais memórias e tempos de execução ainda mais longos. Devido esses motivos o *Google Colab Pro* foi utilizado no desenvolvimento desse trabalho (CHOLLET, 2020).

3.4 Desenvolvimento

Primeiramente deve-se é importar as dependências, como mostrado na Figura 12. As bibliotecas utilizadas são, a biblioteca do *TensorFlow* e o modulo do *Keras*. A biblioteca "*tensorflow*", nomeada como "tf", provê acesso às funções de manipulação e treinamento da rede neural. A modulo Keras permite a criação dos modelos e o uso da base de dados. Também serão importadas outras bibliotecas importantes para manipulação de imagens, como a biblioteca do "OpenCV", importada como "cv2", que permite o uso de *haarcascade* prontos para localização de faces dentro da imagem. A versão do *TensorFlow* utilizada para o desenvolvimento dessa aplicação foi a versão *2.0.0*, já a versão do *OpenCV* foi a versão *3.4.3*.

Figura 12 – Bibliotecas usadas no desenvolvimento

```
# Bibliotecas gerais para manipulação de imagens, funções numéricas, gráficos, arquivos, entre outros
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import zipfile
import tensorflow

# Patch de correção de funcionamento da função 'imshow' do OpenCV para o google colab
from google.colab.patches import cv2_imshow
# Importação de biblioteca para a manipulação do Google Drive dentro do Colab
from google.colab import drive

# Função utilizada para dividir a base de dados
from sklearn.model_selection import train_test_split

# Funções do módulo Keras a serem uilizados no desenvolvimento do projeto
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D, BatchNormalization
from tensorflow.keras.losses import categorical_crossentropy
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping, ModelCheckpoint
from tensorflow.keras.models import load_model
from tensorflow.keras.models import model_from_json
```

Fonte: Autoria própria.

Após importar as bibliotecas, é necessário buscar e tratar as imagens a serem utilizadas de treinamento, validação e teste da CNN. Para isso, é utilizado um arquivo em formato ".csv" contendo os valores de pixels da imagem em escala de cinza, e um atributo de emoção, que representa a qual classe essa imagem pertence. A Figura 13 mostra a importação do arquivo e a Figura 14 a manipulação das imagens.

Figura 13 – Importação do arquivo com os dados de entrada

```
# Leitura da base de dados
# O arquivo já está com os pixeis da imagem extraídos em forma de tabela
# Os rótulos vinculando as imagens a uma emoção já estão definidos
data = pd.read_csv("fer2013/fer2013.csv")
```

Fonte: Autoria própria.

Figura 14 – Manipulação das imagens de entrada

```
# Transformar a coluna de pixels do arquivo em uma array
pixels = data['pixels'].tolist()

# Definir as especificações de dimensões da imagem
largura, altura = 48 , 48

faces = []

for pixel_sequence in pixels:
    face = [int(pixel) for pixel in pixel_sequence.split(' ')]
    # Converter os valores dos pixels da imagem para o formato de numpy array
    # Essa conversão será de acordo com as dimensões da imagem
    face = np.asarray(face).reshape(largura, altura)
    faces.append(face)

# Converter a lista de 'faces' para um numpy array
faces = np.asarray(faces)

# Adicionar mais um atributo a variável 'faces'
faces = np.expand_dims(faces, -1)
# Neste caso, esse atributo irá especificar a quantidade de canais de cores presentes na imagem
```

Fonte: Autoria própria.

Para o TensorFlow, valores dos dados de entrada são números entre 0 e 1, sendo a saída um número entre 0 e 1. As funções de ativação irão sempre normalizar os valores para esse intervalo pequeno, evitando overflows por conta de números muito grandes. Na Figura 15, é feito uma normalização dos valores dos pixels das imagens do dataset para que fiquem dentro do intervalo requerido.

Figura 15 – Normalização dos valores dos pixels da imagem

```
# Definir função de normalização dos valores
# Os valores resultante dessa função estão localizados no intervalo entre 0 e 1
def normalizar(x):
    x = x.astype('float32')
    x = x / 255.0
    return x

# Normalizar os valores do pixels da imagem
faces = normalizar(faces)
```

Fonte: Autoria própria.

A etapa mais simples é listar os rótulos das imagens. Para a rede neural, tudo são números e funções matemáticas. A ideia é que a rede neural retorne qual é a emoção presente na imagem de acordo com a categorização da entrada informada e, a partir disso, é buscado os rótulos no array apresentado na Figura 16.

Figura 16 – Rótulos de classificação das emoções

```
# Converter os valores de cada rótulo de emoções para numpy array
# Essa array irá mostrar o valor de saída final da rede neural
emocoes = pd.get_dummies(data['emotion']).values

# emocoes[0] - array([1, 0, 0, 0, 0, 0, 0], dtype=uint8)
# Classes: ['Angry', 'Disgust', 'Fear', 'Happy', 'Sad', 'Surprise', 'Neutral']
```

Fonte: Autoria própria.

Após o tratamento dos dados de entrada e definição dos rótulos de saída, é necessário dividir os dados da base de dados para a realização do treinamento, validação e teste da CNN. Para isso, é utilizado o método *train_test_split()* do modulo de *model_selection* da biblioteca *sklearn*, que se trata de uma biblioteca de machine learning de código aberto, com suporte para classificação, regressão e agrupamento de dados, dentre outras manipulações. A Figura 17 exemplifica como é feito essa divisão dos dados de entrada.

Para gerar as variáveis de teste e de validação deve-se fornecer para a função alguns atributos, os atributos utilizados nessa aplicação foram:

- Base de dados – Os dados que serão utilizados para seleção de teste, validação e treinamento;
- As classes de classificação – O vetor com a as classes de classificação de cada imagem;
- Porcentagem da base de dados – A porcentagem de imagens que será selecionada para teste ou validação, neste caso foram utilizados 10%;
- Proporção da base de dados – A quantidade de imagens randômicas que serão

selecionadas.

Figura 17 – Divisão dos dados para treinamento da rede neural

```
X_train, X_test, y_train, y_test = train_test_split(faces, emocoes, test_size = 0.1, random_state = 42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size = 0.1, random_state = 41)

# print('Número de imagens no conjunto de treinamento:', len(X_train))
# print('Número de imagens no conjunto de teste:', len(X_test))
# print('Número de imagens no conjunto de validação:', len(X_val))

# Número de imagens no conjunto de treinamento: 29068
# Número de imagens no conjunto de teste: 3589
# Número de imagens no conjunto de validação: 3230
```

Fonte: Autoria própria.

Todos os passos anteriores serão os mesmos realizados em cada um dos treinamentos com um modelo de CNN. A seguir está a definição de algumas variáveis importantes que serão utilizadas no primeiro modelo de CNN que será testado, essas variáveis e valores estão definidos na Figura 18.

Figura 18 – Parâmetros a serem utilizados pelo primeiro modelo da rede neural

```
# Número de filtros da rede neural convolucional
num_features = 64
# Quantidade de classes de resposta da rede neural
num_labels = 7
# Quantidade de registros a serem verificados antes de fazer a troca dos pesos da rede neural
batch_size = 64
# Número máximo de épocas que esse algoritmo irá executar
epochs = 100
# Altura e largura das imagens
width, height = 48, 48
```

Fonte: Autoria própria.

Finalmente, uma das etapas mais importantes é a definição do modelo que será utilizado. Neste caso, devem ser considerados alguns conceitos sobre os tipos de camadas que a API do Keras disponibiliza e a forma de montar o modelo. O primeiro item, mostrado na Figura 19, é o método “Sequential”. Segundo a documentação oficial do Keras, o método *Sequential* cria uma estrutura linear e em pilha das camadas da rede neural. Isso definirá a forma da rede neural. Em relação às camadas, nota-se que iremos adicionar as camadas utilizando o método de *model.add*.

A primeira camada adicionada é uma camada convolucional de entrada, ou seja, por se tratar da primeira camada ela irá receber diretamente a base de dados, deve-se declarar os atributos utilizados nessa camada, sendo eles:

- Número de filtros – define a quantidade de filtros de saída na convolução;
- Tamanho da janela – especifica a altura e largura da janela de convolução 2D;
- Função de ativação – função de ativação a ser usada;

- Dimensões da entrada – dimensões dos dados nas entradas;
- Formato da entrada – ordem das dimensões nas entradas;
- Regularização do kernel – função *regularizer* aplicada ao vetor de polarização;

A segunda camada adicionada também é uma camada convolucional, no entanto, será necessário definir menos parâmetros comparado a primeira camada. Deve-se definir somente um novo parâmetro, o *padding*, que nesse caso será usado o método '*same'* que resulta no preenchimento com zeros uniformemente à esquerda/direita ou para cima/para baixo da entrada, de forma que a saída tenha a mesma dimensão de altura/largura da entrada. A terceira camada é a "*BatchNormalization*", ou seja, é uma camada de normalização, que irá atuar em cima das camadas escondidas normalizando os valores para que fiquem entre 0 e 1.

A quarta camada será uma calada de "*MaxPooling",* desta forma o resultado dessa camada será o maior valor dentro de uma determinada região, para a execução da camada deve-se definir primeiro o tamanho da região a ser analisada, ou seja, o tamanho da matriz de *pooling*, além do tamanho dos *strides* que especifica até que ponto a janela de *pool* se move para cada das etapas de *pool*. A quinta camada será uma camada de "*Dropout"*, essa camada será utilizada para zerar uma fração dos neurônios de maneira aleatória diminuindo assim a entrada da próxima camada, neste caso será zerado 50% dos neurônios, isso é feito com o intuito de remover *overfitting*.

As camadas seguintes irão se repetir três vezes seguindo o padrão: uma camada de convolução, uma camada de normalização, outra camada de convolução, outra camada de normalização, uma camada de *pooling* e uma camada de *dropout*. A única diferença nessas camadas será a quantidade de filtros na camada de convolução. Em seguida será adicionado uma camada de "*Flatten*" que irá converter o resultado das camadas anteriores em um vetor, que será utilizado como entrada das camadas seguintes.

Agora será adicionado uma sequência de camadas intercalando uma camada densa e uma camada de "*Dropout*", é necessário atribuir as camadas densas a quantidade de neurônios em cada camada e uma função de ativação. Por fim será adicionada uma camada densa com a para gerar a saída da rede neural, tendo como atributos a quantidade de saídas, ou seja, aos rótulos de classificação e a função de ativação, a função '*softmax'* é utilizada quando a aplicação possui mais de uma classe de classificação.

Figura 19 – Definição do primeiro modelo da rede neural utilizando o Keras API

```
model = Sequential()

model.add(Conv2D(num_features, kernel_size=(3,3), activation='relu',
                 input_shape=(width, height, 1), data_format = 'channels_last',
                 kernel_regularizer = l2(0.01)))
model.add(Conv2D(num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.5))

model.add(Conv2D(2*num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(2*num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.5))

model.add(Conv2D(2*2*num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(2*2*num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.5))

model.add(Conv2D(2*2*2*num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(2*2*2*num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.5))

model.add(Flatten())

model.add(Dense(2*2*2*num_features, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(2*2*num_features, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(2*num_features, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(num_labels, activation = 'softmax'))
```

Fonte: Autoria própria.

É possível utilizar as mesmas dados de entrada para treinamento de outros modelos de CNN. A seguir está a definição de algumas variáveis importantes que serão utilizadas no segundo modelo de CNN que será testado, essas variáveis e valores estão definidos na Figura 20. A única variável diferente das utilizadas no primeiro modelo, será a quantidade de filtros da rede neural.

Figura 20 – Parâmetros a serem utilizados pelo segundo modelo da rede neural

```
# Número de filtros da rede neural convolucional
num_features = 32
# Quantidade de classes de resposta da rede neural
num_labels = 7
# Quantidade de registros a serem verificados antes de fazer a troca dos pesos da rede neural
batch_size = 16
# Número máximo de épocas que esse algoritmo irá executar
epochs = 100
# Altura e largura das imagens
width, height = 48, 48
```

Fonte: Autoria própria.

As principais diferenças desse segundo modelo de CNN é a função de ativação, neste modelo de rede neural será utilizado uma camada de ativação chamado '*elu*' que é uma função similar a função '*relu*', a principal diferença entre elas é o problema de unidades 'mortas' apresentado pelas funções '*relus*', esse problema faz com que os neurônios produzam apenas zeros. Outra diferença é a função de inicialização do kernel neste modelo será utilizado a função '*he_normal*', essa função extrai amostras de uma distribuição normal truncada centrada em 0 com $stddev = sqrt\,(2 \,/\, fan\_in)$ onde fan_in é o número de unidades de entrada no tensor de peso. Por fim a quantidade de neurônios que serão zerados no fim do processamento das camadas de "*MaxPooling*" também é menor do que o modelo anterior.

Figura 21 – Definição do segundo modelo da rede neural utilizando o Keras API

```
model = Sequential()

model.add(Conv2D(num_features, kernel_size=(3,3), padding = 'same', kernel_initializer="he_normal",
                 input_shape = (width, height, 1)))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Conv2D(num_features, kernel_size=(3,3), padding = "same", kernel_initializer="he_normal",
                 input_shape = (width, height, 1)))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(2*num_features, kernel_size=(3,3), padding="same", kernel_initializer="he_normal"))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Conv2D(2*num_features, kernel_size=(3,3), padding="same", kernel_initializer="he_normal"))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(2*2*num_features, kernel_size=(3,3), padding="same", kernel_initializer="he_normal"))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Conv2D(2*2*num_features, kernel_size=(3,3), padding="same", kernel_initializer="he_normal"))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
```

```
model.add(Conv2D(2*2*2*num_features, kernel_size=(3,3), padding="same", kernel_initializer="he_normal"))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Conv2D(2*2*2*num_features, kernel_size=(3,3), padding="same", kernel_initializer="he_normal"))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(2*num_features, kernel_initializer="he_normal"))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(2*num_features, kernel_initializer="he_normal"))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(num_labels, kernel_initializer="he_normal"))
model.add(Activation("softmax"))
```

Fonte: Autoria própria.

Para esse terceiro modelo de CNN será feita uma otimização utilizando o algoritmo *Adam*, além de executar uma pré-compilação do modelo tendo como métrica a acurácia de treinamento do modelo, como pode ser visto na Figura 22. As variáveis utilizadas neste modelo possuem o mesmo valor das utilizadas no primeiro modelo, apresentadas anteriormente.

Figura 22 – Definição do terceiro modelo da rede neural utilizando o Keras API

```
model = Sequential()

model.add(Conv2D(num_features, kernel_size=(3,3), activation='relu',
                 input_shape=(width, height, 1), data_format = 'channels_last',
                 kernel_regularizer = l2(0.01)))
model.add(Conv2D(num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.5))

model.add(Conv2D(2*num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(2*num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(2*num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.5))

model.add(Conv2D(2*2*num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(2*2*num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(2*2*num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.5))
```

```
model.add(Conv2D(2*2*2*num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(2*2*2*num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(2*2*2*num_features, kernel_size=(3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.5))

model.add(Flatten())

model.add(Dense(2*2*2*num_features, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(2*2*num_features, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(2*num_features, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_features, activation='relu'))
model.add(Dropout(0.5))


model.add(Dense(num_labels, activation = 'softmax'))

adam = optimizers.Adam(lr = 0.001)

model.compile(optimizer = adam, loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

Fonte: Autoria própria.

O quarto e último modelo que será testado neste projeto será relativamente mais simples que os modelos anteriores, a quantidade de registros a serem verificados antes de trocar os pesos da rede neural é maio do que a dos modelos anteriores como pode-se notar na Figura 23, número de filtros da rede neural também irá seguir um padrão diferente do dos modelos anteriores como se pode ver na Figura 24. O modelo será composto por, primeiramente duas camada convolucionais, onde a primeira camada irá receber os dados de entrada, neste caso as imagens, em seguida tem-se uma camada de *pooling* e só então uma camada de normalização, por fim será realizado um '*Dropout'* nos valores. A quantidade de camadas densas do modelo também será menor se comparado aos outros modelos, como pode-se notar na Figura 24.

Figura 23 – Parâmetros a serem utilizados pelo quarto modelo da rede neural

```
# Quantidade de classes de resposta da rede neural
num_labels = 7
# Quantidade de registros a serem verificados antes de fazer a troca dos pesos da rede neural
batch_size = 256
# Número máximo de épocas que esse algoritmo irá executar
epochs = 100
# Altura e largura das imagens
width, height = 48, 48
```

Fonte: Autoria própria.

Figura 24 – Definição do quarto modelo da rede neural utilizando o Keras API

```
num_labels = 7
batch_size = 256
epochs = 100
width, height = 48, 48

model = Sequential()

model.add(Conv2D(20, kernel_size=(3,3), padding='same', activation='relu', input_shape=(width, height, 1)))
model.add(Conv2D(30, kernel_size=(3,3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Conv2D(40, kernel_size=(3,3), padding='same', activation='relu'))
model.add(Conv2D(50, kernel_size=(3,3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Conv2D(60, kernel_size=(3,3), padding='same', activation='relu'))
model.add(Conv2D(70, kernel_size=(3,3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(80, kernel_size=(3,3), padding='same', activation='relu'))
model.add(Conv2D(90, kernel_size=(3,3), padding='same', activation='relu'))

model.add(Flatten())

model.add(Dense(1000, activation='relu'))
model.add(Dense(512, activation='relu'))

model.add(Dense(num_labels, activation='softmax'))

model.summary()
```

Fonte: Autoria própria.

Não basta ter a estrutura, modelos e dados carregados, é necessário compilar a rede neural, passando um algoritmo de otimização. Neste projeto, foi definida uma função para cálculo de perda, chamada "*Categorical Crossentropy*", que irá informar o quanto ainda há para aprender sobre a base de dados. Para otimização do modelo foi utilizado o *Adam Optimizer Algorithm*, um algoritmo de otimização estocástico, baseado em gradiente. Também é nesta etapa que são definidas as métricas para avaliar o desempenho da rede neural. Essa estimativa define se é necessário continuar o treinamento ou se a rede já aprendeu o máximo possível com os dados processados. A métrica utilizada é a "*Accuracy*", como objetivo de validar o quanto a rede neural está sendo assertiva nos resultados. Na Figura 25 é ilustrado como foi feita a compilação da rede neural.

Figura 25 – Compilando, otimizando e definindo métricas do modelo

```
# Compilação do modelo de rede neural convolucional
model.compile(loss = 'categorical_crossentropy',
              optimizer = Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-8),
              metrics = ['accuracy'])
```

Fonte: Autoria própria.

Para utilizar o modelo treinado em aplicações futuras é necessário salvar o modelo treinado, além do modelo da arquitetura utilizada. Para isso, deve definir um local para salvar esses arquivos, o arquivo no formato '.h5' será utilizado para realizar as predições da rede neural, já o modelo no formato '.json' irá armazenar a arquitetura do modelo, contendo quais são as camadas da rede neural, além de valores atribuídos a cada uma delas. Será necessário definir as funções de retorno que irão verificar se a rede neural está aprendendo durante o treinamento, além de realizar as intervenções cabíveis para melhorar o aprendizado da rede neural, como mostrado na Figura 26.

A primeira função é '*ReduceLROnPlateau'* essa função reduz a taxa de aprendizagem quando uma métrica para de melhorar, os atributos para execução da função que será utilizada, são:

- *monitor* – métrica a ser monitorada pela função;
- *factor* – fator pelo qual a taxa de aprendizagem será reduzida;
- *patience* – número de épocas sem melhora, após as quais a taxa de aprendizagem será reduzida;
- *verbose* – mostrar mensagens de execução da função.

A segunda função é '*EarlyStopping'*, essa função para o treinamento quando uma métrica para de melhorar, os atributos para execução da função que será utilizada, são:

- *monitor* – métrica a ser monitorada pela função;
- *min_delta* - mudança mínima na quantidade monitorada para qualificar como uma melhoria, ou seja, uma mudança absoluta menor que min_delta, não contará como uma melhoria;
- *patience* – número de épocas sem melhora, após as quais o treinamento será interrompido;
- *verbose* – mostrar mensagens de execução da função.
- *mode* – essa variável defini se o treinamento será interrompido se o valor da métrica de monitoramento parar de aumentar ou diminuir. Nesta aplicação, quando se utiliza o *mode = "auto"* a direção é inferida automaticamente a partir d nome da métrica monitorada.

A última função '*ModelCheckpint'* é utilizada para salvar um modelo ou pesos para que possam ser carregados posteriormente para continuar o treinamento a partir do estado salvo. Para isso, deve-se definir o caminho para salvar o arquivo de modelo, o nome da métrica a ser monitorada e quando o modelo será salvo, neste caso será considerado que somente o modelo

considerado como o ‘melhor’ será salvo, esse atributo também impede que o último modelo considerado melhor, de acordo com a métrica monitorada, seja sobescrito.

Figura 26 – Caminho de salvamento dos arquivos e funções de retorno

```
# Arquivos de saída do modelo - em formato em h5 e json
arquivo_modelo = "gdrive/MyDrive/modelo_01_expressoes.h5"
arquivo_modelo_json = "gdrive/MyDrive/modelo_01_expressoes.json"

# Definir funções de callbacks que serão aplicadas no treinamento do modelo
lr_reducer = ReduceLROnPlateau(monitor='val_loss', factor=0.9, patience=3, verbose=1)
early_stopper = EarlyStopping(monitor='val_loss', min_delta=0, patience=8, verbose=1, mode='auto')
checkpointer = ModelCheckpoint(arquivo_modelo, monitor='val_loss', verbose=1, save_best_only=True)
```

Fonte: Autoria própria.

Na Figura 27, é apresentado o trecho de código necessário para realizar o treinamento da rede neural. O *TensorFlow* utiliza o método *model.fit()* para executar esse treinamento. Esse método é responsável por realizar o treinamento dos modelos criados nos passos anteriores. São passados os *datasets* de treinamento e as *labels* para a função, deve-se lembrar que o *TensorFlow* somente recebe dados do tipo *array* do *Numpy* ou tensores do próprio *tesorflow*. É passada o número de amostrar por atualização de gradiente, e a quantidade de máxima de épocas que o algoritmo deverá treinar.

Também serão passados os dados de validação do modelo, para validar se ele está realmente melhorando seu aprendizado. O atributo *shuffle* define se os dados de treinamento devem ser embaralhados antes de cada época. E por fim é necessário definir quais são os retornos da chamada a serem aplicados durante o treinamento.

Figura 27 – Treinamento do modelo

```
# Treinamento do modelo
history = model.fit(np.array(X_train), np.array(y_train),
                    batch_size = batch_size,
                    epochs = epochs,
                    verbose = 1,
                    validation_data = (np.array(X_val), np.array(y_val)),
                    shuffle = True,
                    callbacks = [lr_reducer, early_stopper, checkpointer])
```

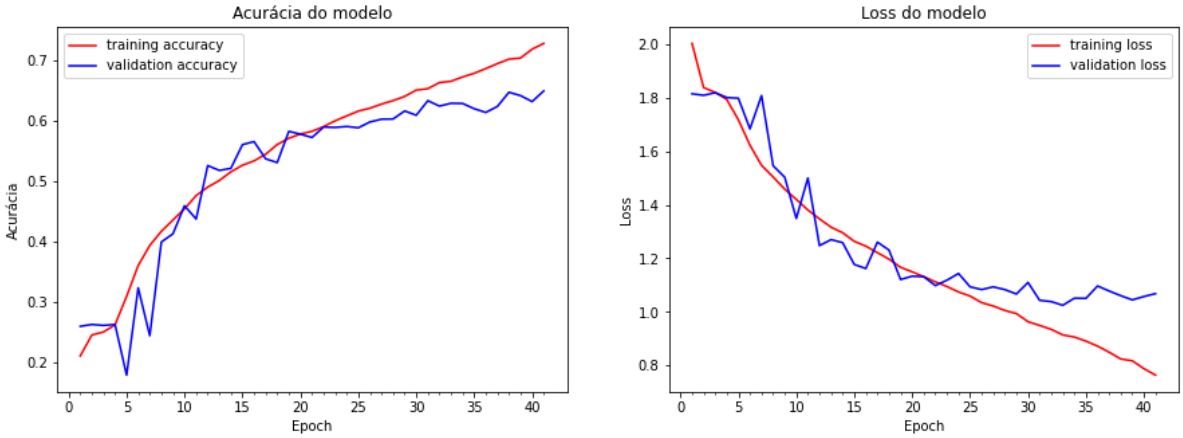
Fonte: Autoria própria.

4 RESULTADO E DISCUSSÃO

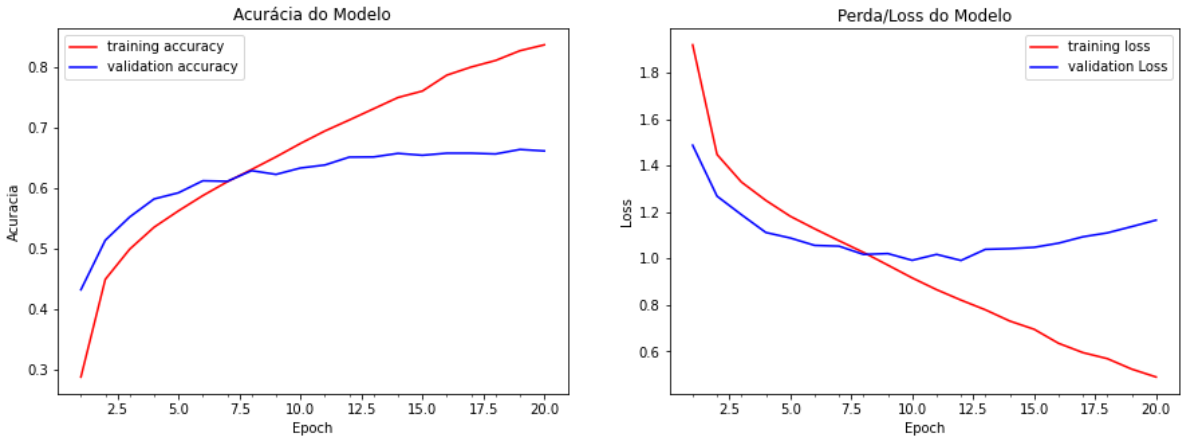
Após o treinamento, a rede neural estará pronta para fazer suas classificações e predições. A Figura 28 mostra os gráficos de aprendizagem dos modelos, neles é possível perceber que alguns modelos possuem um grau de aprendizado mais rápido e param de melhorar também rapidamente. Como a métrica que estava sendo monitorada era a de nome ‘*val_loss*’, que corresponde a perda dos testes de validação do modelo, quando essa variável para de diminuir o treinamento é encerado.

É possível avaliar pelo gráfico a partir de qual época o modelo começa a estagnar e decair a aprendizagem, no entanto o treinamento ainda ocorre devido as funções de retorno que buscam uma correção no treinamento para que ele possa melhorar ainda mais. Nenhum dos modelos precisou ser treinado pelo número máximo de épocas definidas na aplicação.

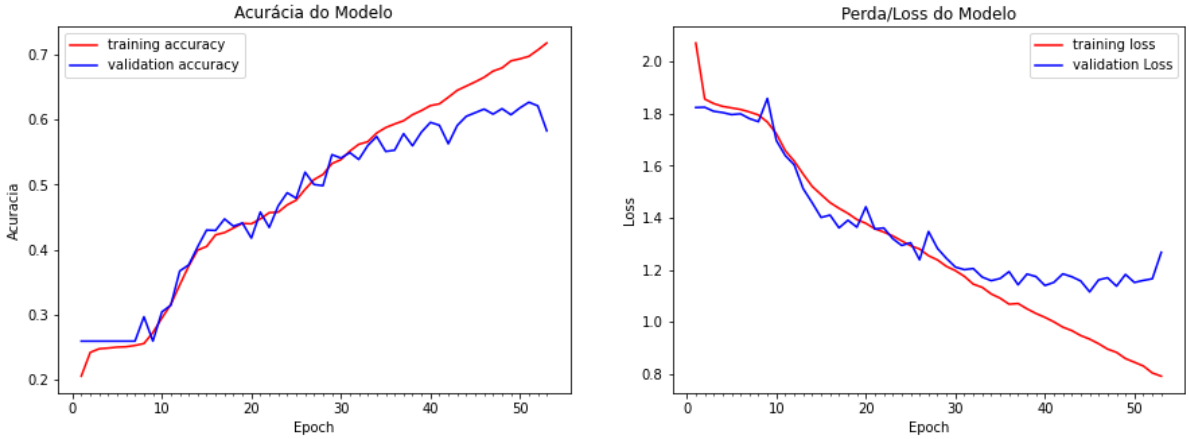
Figura 28 – Gráficos da acurácia e da perda dos modelos



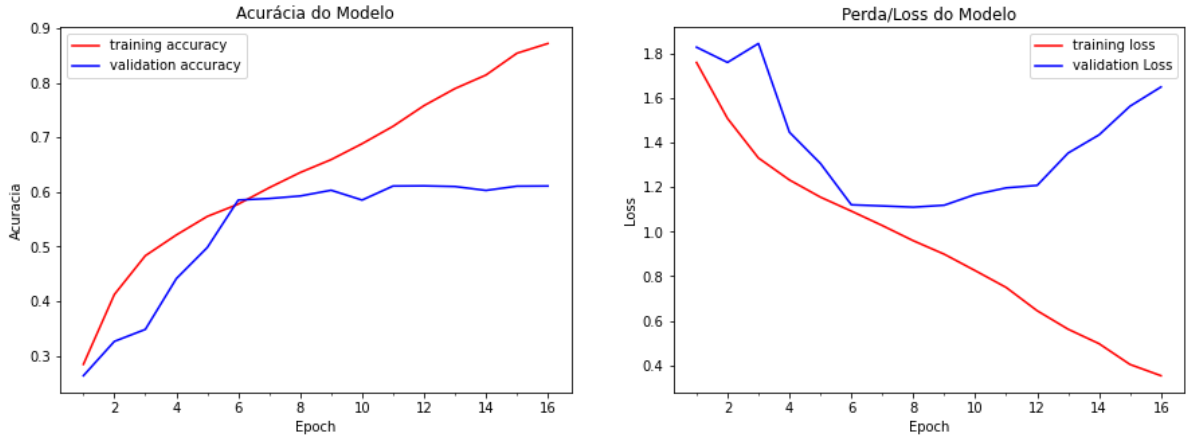
(a) Gráficos de treinamento do modelo 1



(b) Gráficos de treinamento do modelo 2



(c) Gráficos de treinamento do modelo 3



(d) Gráficos de treinamento do modelo 4

Fonte: Autoria própria.

Para validar se o modelo realmente é capaz de realizar predições de maneira correta será utilizado os dados que não foram utilizados nem no treinamento nem na avaliação do modelo, para definir a acurácia e a perda/erro do modelo em um conjunto de teste, como mostrado na Tabela 1. Pode-se notar que o Modelo 2 apresentou uma maior acurácia, no entanto o Modelo 1 é quem possui o menor valor erro/perda. Para validar a acurácia real do valor de teste é será feita uma comparação entre os valores reais das emoções, qual emoção esta retratada na imagem, e as predições do modelo. Esses acertos são somados e depois dividido pela quantidade de imagens presentes na base de dados de teste, neste caso é composta por 3589 imagens, assim obtém-se os valores apresentados na Tabela 2.

Tabela 1 – Validação da acurácia e perda dos modelos

	Acurácia	**Perda**
Modelo 1	0.618278	1.135641
Modelo 2	0.657286	1.203296
Modelo 3	0.560880	1.340449
Modelo 4	0.592922	1.739553

Fonte: Autoria própria.

Tabela 2 – Acurácia no conjunto de testes

	Acurácia
Modelo 1	61.38%
Modelo 2	63.80%
Modelo 3	59.77%
Modelo 4	57.62%

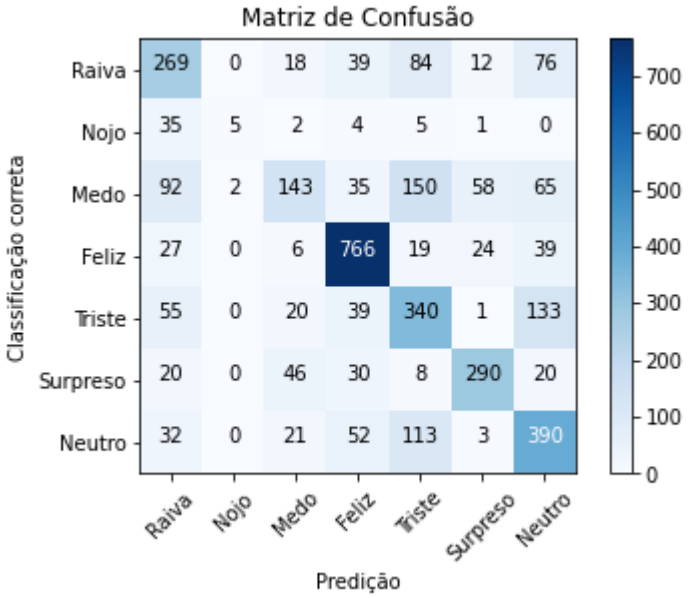
Fonte: Autoria própria.

Utilizando as respostas reais e as respostas de predição do modelo é possível gerar uma matriz de confusão para mostras a predições feitas pelo modelo, matriz pode-se a quantidade de acerto dos modelos para esse banco de testes. Pode-se perceber que a classe em que os modelos tiveram uma maior porcentagem de predições erradas, é a classe correspondente a

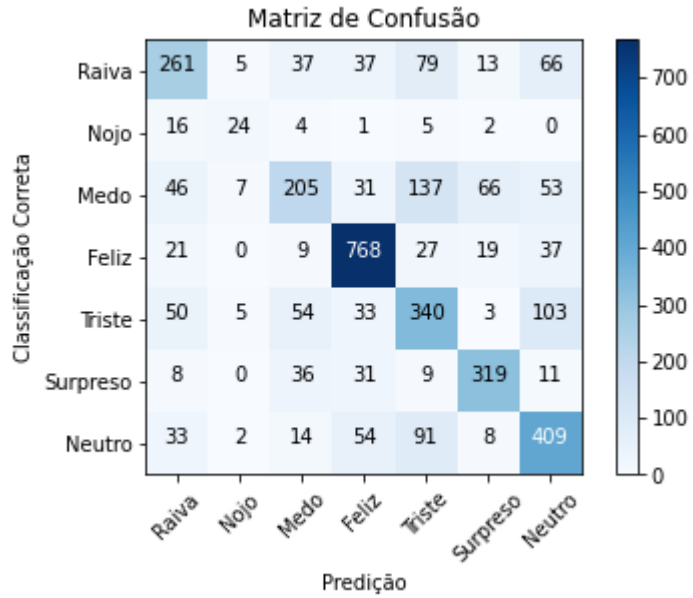
emoção de ‘Nojo’, no caso do Modelo 3, por exemplo, o modelo errou todas as predições para essa emoção, isso pode ser atribuído ao fato de que essa emoção possuía a menor quantidade de imagens do banco de dados. Já a classe que possui o maior número de acerto foi justamente a emoção que possuía o maior número de imagens do *dataset*, no caso, a emoção ‘Feliz’.

A emoção do ‘Medo’ possui uma quantidade relevante de predições como sendo ‘Surpreso’, isso pode ser atribuído as características microexpressões faciais que definem essas emoções, já que analise destas emoções possuem muitos pontos em comum. As demais emoções possuem uma taxa de acerto mediana.

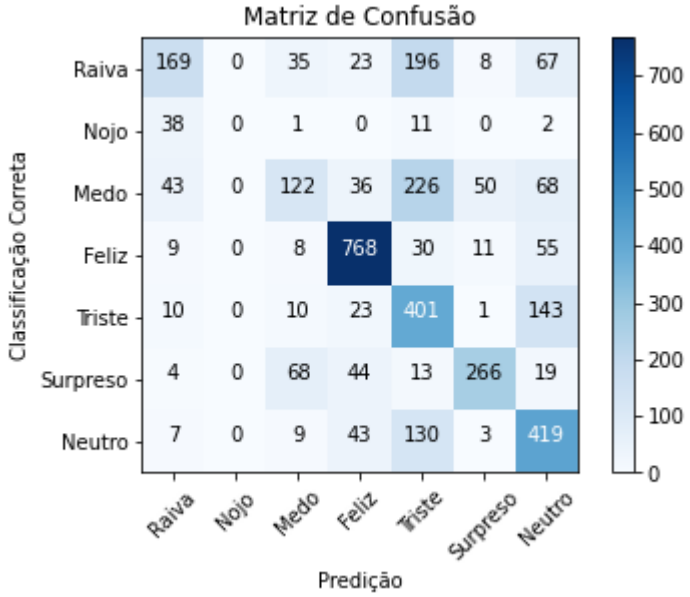
Figura 29 – Matrizes de confusão com a classificação dos dados de teste



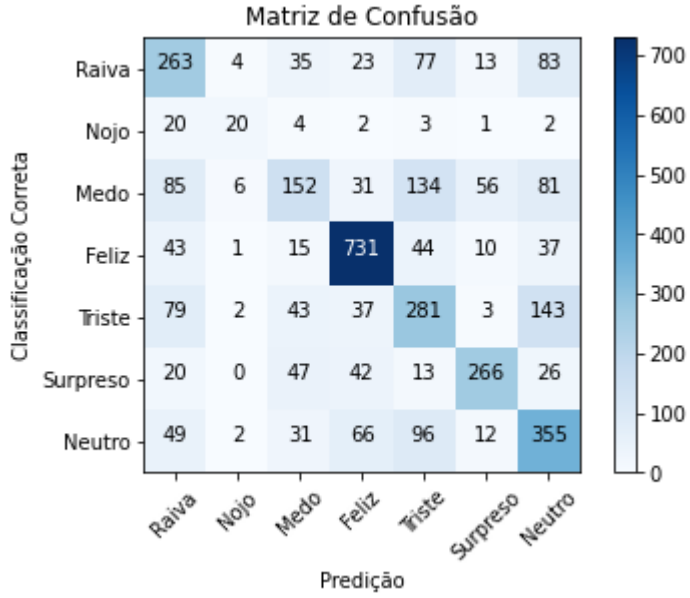
(a) Matriz de confusão do modelo 1



(b) Matriz de confusão do modelo 2



(c) Matriz de confusão do modelo 3



(d) Matriz de confusão do modelo 4

Fonte: Autoria própria.

Para realizar o reconhecimento de emoções em vídeo é necessário dividir o vídeo em *frames*, já que os *frames* funcionam como imagens para modelo de classificação, como mostra a Figura 30. Será utilizado o segundo modelo, descrito acima, já que este teve a maior acurácia comparado com os demais. Ainda que o modelo possua uma acurácia baixa, ele é capaz de identificar as emoções em cada um dos *frames* do vídeo. O reconhecimento da emoção usa um *haarcascade* para localizar as faces presente na imagem, por isso em alguns momentos as faces podem não ser detectadas devido aos parâmetros para localização da face estarem incorretos ou não otimizados.

Figura 30 – Reconhecimento de emoções em vídeo

Fonte: Autoria própria.

5 CONCLUSÕES E TRABALHOS FUTUROS

As Redes Neurais de Aprendizado Profundo abrem um leque de possibilidades para solucionar problemas que antes não eram possíveis. Cada vez mais aplicações são desenvolvidas com o intuído de melhorar a interação homem máquina, buscando uma qualidade de vida melhor com o uso das máquinas. Busca-se automatizar cada vez mais serviços com o auxílio de algoritmos capazes de realizar predições e adaptações, que antes só eram possíveis com a intervenção humana. Atualmente cada vez mais a I.A. está presente no dia a dia do ser humano. A Inteligência Artificial é utilizada em vários cenários, como já citado neste artigo, em carros autônomos, medicina, indústria e experiência do usuário.

As redes neurais possem uma gama de aplicações e se mostram cada vez mais eficientes em aplicações que necessitam de predições e adaptações para determinados cenários. Já que esse modelo se aproxima mais do cérebro humano. O surgimento de novas ferramentas

auxilia cada vez no desenvolvimento de aplicações mais precisas e robustas, como é o caso *framework* do *TensorFlow*, que juntamente com a API do *Keras* permite com que sejam desenvolvidas redes neurais convolucionais de maneira cada vez mais simples.

O surgimento de serviços como o *Google Colab* permite com que cada vez mais pessoas estudem e desenvolvam aplicações que antes não eram possíveis por limitações de hardware. Já não é mais necessário que se tenha a disponibilidade fisicamente de um hardware robusto para se desenvolver aplicações de *deep learning* ou *machine learning*, com o surgimento de serviços que possibilitam rodar aplicações em nuvem utilizando especificações de hardwares mais elevada, como o uso de GPUs ou uma quantidade maior de memória RAM.

O reconhecimento e distinção de emoções é uma interação essencial para o convívio humano. O desenvolvimento de aplicações que buscam identificar e reconhecer características comportamentais, pode ser um passo adiante para melhor a interação homem-máquina. Ainda a muito o que melhorar na questão do reconhecimento de emoções utilizando imagens, mas já temos aplicações e arquiteturas capazes de reconhecer com um nível alto de acurácia emoções humanas.

Deve-se levar em consideração como essas aplicações serão utilizadas. Futuramente em novos estudos seria possível verificar como as emoções influência o ser humano em suas tomadas de decisões, e como uma máquina capaz de reconhecer as emoções humanas pode tirar vantagem sobre as interações que ela realiza com o usuário. Será verificado uma abordagem de como essas predições de emoções podem ser utilizadas para auxiliar em casos criminais ou de saúde.

REFERÊNCIAS

ABADI, Martín et al. **TensorFlow**: Large-Scale Machine Learning on Heterogeneous Distributed Systems. Software available from tensorflow.org, 2015. Disponível em: https://www.tensorflow.org/. Acesso em: 23 out. 2021.

ABADI, Martín et al. **TensorFlow**: A System for Large-Scale Machine Learning. 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), Savannah, GA, p. 265-283, 2016. Disponível em: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi. Acesso em: 23 out. 2021.

ALBUQUERQUE, César T. de; SANTIAGO, Rafael de; MOREIRA, Benjamin G. **Switch**

KVM Através de Visão Computacional. Revista Eletrônica Argentina-Brasil de Tecnologias da Informação e da Comunicação, Três de Maio, v. 1, ed. 1, 2014. Disponível em: http://doi.org/10.5281/zenodo.59427. Acesso em: 27 mar. 2020.

ARAÚJO, Flávio H. D. et al. **Redes Neurais Convolucionais com Tensorflow**: Teoria e Prática. III Escola Regional de Informática do Piauí. Livro Anais - Artigos e Minicursos, v. 1, n. 1, p. 382-406, jun, 2017.

AREL, I.; ROSE, D.; KARNOWSKI, T. **Deep machine learning-a new frontier in artificial intelligence research**. IEEE Computational Intelligence Magazine, v. 5, n. 4, p. 13–18, 2010. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-77958488310&partnerID=40&md5=621fd1749901fd579e898db77d159e18>.

BARCELOS, Arlei Fonseca; TOSHIMITSU, Hugo Shokychi. **Modelo matemático neural artificial para cálculo de rendimento metálico teórico de um laminador de tiras a frio**. In: XV Simpósio De Excelência Em Gestão E Tecnologia – SEGET, 2018, Resende/RJ. Revista Eletrônica do XV SEGeT [...]. [S. l.: s. n.], 2018. Disponível em: https://www.aedb.br/seget/arquivos/artigos18/36626398.pdf. Acesso em: 21 ago. 2021.

BIRCK, Vera Regina. **A voz do corpo: A Comunicação Não-Verbal e as Relações Interpessoais.** In. XXXI Congresso Brasileiro de Ciências da Comunicação. Jornada de Inovações Midiáticas e Alternativas Experimentais. Curitiba, 2008.

FER2013: Learn facial expressions from an image. In: CARRIER, Pierre-Luc; COURVILLE, Aaron. **Challenges in Representation Learning**: Facial Expression Recognition Challenge. 2013. Disponível em: https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge. Acesso em: 28 ago. 2021.

CARVALHO, Anderson. **FACS, o que é e para que serve**. In: Academia Internacional de Linguagem Corporal. 26 dez. 2017. Disponível em: https://cursodelinguagemcorporal .com/facs-o-que-e-e-para-que-serve/. Acesso em: 24 abr. 2021.

CHOLLET, François. **Deep Learning with Python**. 2. ed. Manning, 2020. ISBN 9781617296864.

DARWIN, Charles. **A expressão das emoções no homem e nos animais**. Tradução: Leon de Souza Lobo Garcia. Companhia de Bolso, 2009.

DUMOULIN, V.; VISIN, F. A guide to convolution arithmetic for deep learning. ArXiv eprints, mar. 2016.

EKMAN, Paul. **A linguagem das emoções**: Revolucione sua comunicação e seus relacionamentos reconhecendo todas as expressões das pessoas ao redor. Tradução: Carlos Szlak. São Paulo: Lua de Papel, 2011.

EKMAN, P. **Emotions Revealed**: Recognizing Faces and Feelings to Improve Communication and Emotional Life. Henry Holt and Company, 2003. ISBN 9780805072754. Disponível em: https://books.google.com.br/books?id=JJ9xHXo7hPgC

FURTADO, Maria Inês Vasconcellos. **Redes Neurais Artificiais:** Uma Abordagem Para Sala de Aula. Ponta Grossa (PR): Atena Editora, 2019. DOI 10.22533/at.ed.262191504. Disponível em: https://www.atenaeditora.com.br/wp-content/uploads/2019/05/e-book-Redes-Neurais-Artificiais-uma-Abordagem-para-Sala-de-Aula.pdf. Acesso em: 21 ago. 2021.

GOIS, Aline Katia de et al. A Linguagem do corpo e a Comunicação nas Organizações. **Revista Anagrama**: Revista Científica Interdisciplinar da Graduação, São Paulo, ano 4, ed. 4, Junho/Agosto 2011.

GONÇALVES, Antonio Baptista; PEPPI, Fabiani Mrosinski. **MICROEXPRESSÕES FACIAIS: LENDA OU REALIDADE?**. Revista da Seção Judiciária do Rio de Janeiro, [S.l.], v. 23, n. 45, p. 25-60, jul. 2019. ISSN 2177-8337. Disponível em: http://lexcultccjf.trf2.jus.br/index.php/revistasjrj/article/view/217. Acesso em: 25 março 2021. doi: https://doi.org/10.30749/2177-8337.v23n45p25-60.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep learning.** Book in preparation for MIT Press. 2016. Dispon´ıvel em: http://www.deeplearningbook.org

HAYKIN, Simon. **Redes Neurais**: Princípios e Prática. 2ª. ed. Bookman, 2000.

HAYKIN, Simon. **Neural Networks and Learning Machines**. 3ª. ed. Prentice Hall, 2008.

JAMES, Judi. **Linguagem corporal no trabalho**. Rio de Janeiro: Best Seller, 2008.

KAEHLER, Adrian; BRADSKI, Gary. **Learning OpenCV 3**: Computer Vision in C++ with the OpenCV Library. Sebastopol: O'Reilly Media, Inc., 2017.

KINGMA, Diederik P.; BA, Jimmy. **Adam**: A Method for Stochastic Optimization. 2014. Disponível em: https://arxiv.org/abs/1412.6980v9. Acesso em: 6 nov. 2021.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. I**mageNet Classification with Deep Convolutional Neural Networks**, 2017. Disponível em: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neuralnetworks.pdf. Acesso em: 5 nov. 2021.

LANGER, Bernardo; BARBOSA, Fernando de Alvarenga. Análise das memórias falsas e microexpressões faciais em depoimentos judiciais. **Revista Jus Navigandi**, ISSN 1518-4862, Teresina, ano 25, n. 6313, 13 out. 2020. Disponível em: https://jus.com.br/artigos/84254. Acesso em: 15 mar. 2021.

LIBRALON, Giampaolo Luiz. **Modelagem computacional para reconhecimento de emoções baseada na análise facial**. 2014. Tese (Doutorado em Ciências de Computação e Matemática Computacional) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2014. Disponível em: https://teses.usp.br/teses/disponiveis/55/55134/tde-10042015-104538/pt-br.php. Acesso em: 27 mar. 2020.

MIGUEL, Fabiano Koich. Psicologia das emoções: uma proposta integrativa para compreender a expressão emocional. **Psico-USF**, Itatiba , v. 20, n. 1, p. 153-162, Apr. 2015. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S1413-8271201500

0100015&lng=en&nrm=iso>. Acesso em: 15 mar. 2021. https://doi.org/10.1590/1413-827120 15200114.

NIELSEN, Michael A. **Neural Networks and Deep Learning**. Determination Press, 2015. Disponível em: http://neuralnetworksanddeeplearning.com. Acesso em: 30 out. 2021.

OPENCV TEAM. **OpenCV**: about. About. 2020. Disponível em: https://opencv.org. Acesso em: 26 out. 2020.

ROBERTO, Marcos; LUIGI, Thiago. **Curso de Micro expressões Faciais**: apostila técnica. São Paulo: IMELCO, 2017.

SILVA, Ivan Nunes da; SPATTI, Danilo Hernane; FLAUZINO, Rogério Andrade. **Redes Neurais Artificiais**: para engenharia e ciências aplicadas. 2ª edição. ed. rev. e aum. São Paulo: Artliber Editora Ltda., 2016.

URIBE-QUEVEDO, Alvaro; REIS, Silas Alves dos. **Rastreamento de rosto como ferramenta interativa e de monitoramento do estado emocional do usuário**. **Revista Científica General José María Córdova**, Colômbia: Bogotá, v. 13, n. 15, p.245-255, jan. 2015.

VIANA, Isabel. **Comunicação não verbal e expressão facial das emoções básicas**. Quinta de Prados/PT: Revista de Letras, n°13, 2014

VILLÁN, Alberto Fernández. **Mastering OpenCV 4 with Python**: a practical guide covering topics from image processing, augmented reality to deep learning with opencv 4 and python 3.7. Birmingham, Uk: Packt Publishing Ltd., 2019.