

**FAESA CENTRO UNIVERSITÁRIO  
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO**

**MARCELO LEITE LOPES**

**DETECÇÃO DE EXPRESSÕES FACIAS: ESTUDO E APLICAÇÃO DE UMA CNN**

**VITÓRIA  
2020**

**MARCELO LEITE LOPES**

**DETECÇÃO DE EXPRESSÕES FACIAS: ESTUDO E APLICAÇÃO DE UMA CNN**

Trabalho de Conclusão de Curso de Graduação em Ciência da Computação apresentado à FAESA Centro universitário, como requisito parcial para obtenção de título de bacharel em Ciência da Computação, sob orientação do Prof. Rober Marccone Rosi.

**VITÓRIA**  
**2020**

## RESUMO

Por muito tempo, as expressões faciais foram objeto de profunda análise realizada por pessoas pertencentes à área de estudo da Psicologia. Com a criação de técnicas automatizadas para análise de expressões faciais e o avanço da tecnologia, houve uma contínua expansão por parte dos observadores no estudo de expressões faciais, devido suas aplicabilidades relacionadas ao comportamento humano estarem fortemente ligadas a externar emoções. Desenvolver a percepção emocional dos computadores é uma tendência tecnológica. A expressão facial é uma maneira efetiva para reconhecer emoções, sobretudo por ser menos intrusiva na coleta de dados, quando comparada aos outros métodos, e pela facilidade de obter imagens da face diante da popularização das câmeras. Neste sentido, o desenvolvimento de algoritmos especializados em aprendizagem de máquina foi um fator de contribuição evidente. O objetivo deste trabalho é classificar, por meio da técnica de Aprendizado Profundo denominada Rede Neural Convolucional, imagens que compõem sete categorias de expressões faciais, apresentando a taxa de acerto (acurácia) na classificação. Utilizando-se uma base *open-source* de imagens faciais, foi realizado o treinamento de uma rede neural para a detecção de emoções através da análise de expressões faciais. A rede treinada apresentou acurácia de 63,97% e pode ser aplicada para a detecção de emoções em fotografias e imagens reais do cotidiano.

**Palavras-chave:** Inteligencia Artificial, Aprendizado Profundo, Redes Neurais Convolucionais, Expressões Faciais

## SUMÁRIO

<b>1 INTRODUÇÃO</b> .....	<b>5</b>
1.1 EMOÇÕES E EXPRESSÕES FACIAIS .....	5
1.2 FORMULAÇÃO DO PROBLEMA .....	6
1.3 HIPÓTESE .....	6
1.4 OBJETIVOS .....	6
<b>1.4.1 Objetivo geral</b> .....	<b>6</b>
<b>1.4.2 Objetivos específicos</b> .....	<b>6</b>
1.5 JUSTIFICATIVA .....	7
<b>2 REFERENCIAL TEÓRICO</b> .....	<b>8</b>
2.1 VISÃO COMPUTACIONAL .....	8
2.2 RECONHECIMENTO FACIAL E COMPUTAÇÃO AFETIVA.....	10
2.3 REDES NEURAIS .....	11
<b>2.3.1 Aplicações das RNA</b> .....	<b>13</b>
<b>2.3.2 Como as RNA funcionam</b> .....	<b>14</b>
2.4 TIPOS E FUNCIONAMENTO DAS RNA.....	15
<b>2.4.1 Redes autocodificadoras</b> .....	<b>15</b>
<b>2.4.2 Redes convolucionais</b> .....	<b>15</b>
2.4.2.1 Aplicações das CNN.....	20
<b>3 METODOLOGIA</b> .....	<b>21</b>
3.1 MATERIAL DE REFERÊNCIA .....	21

3.2 BASE DE DADOS .....	22
3.3 DEFINIÇÃO DA INFRA-ESTRUTURA .....	23
3.4 ARQUITETURA DA REDE NEURAL .....	24
3.5 TREINAMENTO, TESTE E VALIDAÇÃO .....	25
3.6 APLICAÇÃO DA REDE NEURAL .....	25
<b>4 TREINAMENTO DA REDE NEURAL .....</b>	<b>26</b>
4.1 AMBIENTE E BIBLIOTECAS NECESSÁRIAS.....	26
4.2 BASE E PRÉ-PROCESSAMENTO DAS IMAGENS .....	27
4.3 DIVISÃO EM CONJUNTOS DE TREINAMENTO, TESTE E VALIDAÇÃO.....	29
4.4 MONTAGEM DA ARQUITETURA DA REDE NEURAL .....	29
4.5 TREINAMENTO, TESTE E VALIDAÇÃO.....	31
<b>5 APLICAÇÃO DA REDE NEURAL TREINADA.....</b>	<b>35</b>
5.1 PRÉ-PROCESSAMENTO DA IMAGEM.....	35
5.2 DETECÇÃO DA FACE .....	36
5.3 UTILIZAÇÃO DA REDE NEURAL TREINADA.....	37
<b>6 CONCLUSÃO .....</b>	<b>39</b>
<b>REFERÊNCIAS BIBLIOGRÁFICAS.....</b>	<b>40</b>

## 1 INTRODUÇÃO

### 1.1 EMOÇÕES E EXPRESSÕES FACIAIS

A emoção representa um estado psicológico da mente humana. Pesquisas em diversas áreas defendem opiniões distintas sobre o processo de desenvolvimento da emoção (KONAR, CHAKRABORTY, 2015).

Alguns filósofos acreditam que a emoção é resultado de mudanças (positivas ou negativas) em situações pessoais ou no ambiente. Entretanto, alguns biólogos consideram os sistemas nervoso e hormonal como principais responsáveis pelo desenvolvimento das emoções.

Embora não haja um consenso sobre o que causa a emoção, é fato que a sua excitação é geralmente acompanhada de alguma manifestação em nossa aparência, como alterações na expressão facial, voz, gesto, postura e outras condições fisiológicas (DARWIN, 2013).

Há décadas a comunidade científica está interessada no reconhecimento de emoções. As diversas maneiras de expressar as emoções humanas têm sido investigadas, e as seguintes fontes de dados têm sido exploradas (CRUZ, 2019):

- Sinais fisiológicos;
- Textos;
- Envio de emoticons;
- Padrão de uso em dispositivos de entrada de dados (teclado e mouse);
- Voz; e
- Expressões faciais.

Expressões faciais fornecem informações sobre a resposta emocional e exercem um papel fundamental na interação humana e como forma de comunicação não verbal. Podem complementar a comunicação verbal ou, até mesmo, transmitir uma mensagem por si só. As emoções podem contribuir mais para o efeito da mensagem falada do que a entonação ou a própria mensagem.

A expressão facial é anunciada como um sistema de comunicação comum a todas as populações humanas e, portanto, geralmente é aceita como um comportamento universal, de base biológica. Felicidade, tristeza, medo, raiva, surpresa, neutralidade e nojo são sete emoções universalmente reconhecidas e produzidas, e a comunicação desses estados emocionais é considerada essencial para navegar no ambiente social (EKMAN,1970).

Como existe uma correlação entre o estado emocional e as expressões faciais, a análise automatizada dessas expressões faciais para identificar o estado emocional de uma pessoa é um importante campo de pesquisa e aplicação da computação.

## 1.2 FORMULAÇÃO DO PROBLEMA

Como implementar um método automatizado de reconhecimento e classificação das emoções dos indivíduos através da análise de suas expressões faciais?

## 1.3 HIPÓTESE

É possível treinar e utilizar uma rede neural convolucional para, a partir de fotografias de expressões faciais humanas, detectar, identificar e classificar as emoções conforme definidas por Ekman (1970): felicidade, tristeza, medo, raiva, surpresa, neutralidade e nojo.

## 1.4 OBJETIVOS

### 1.4.1 Objetivo geral

Reproduzir o treinamento, teste, validação e aplicação de um modelo de rede neural convolucional para classificar expressões faciais em imagens de faces humanas.

### 1.4.2 Objetivos específicos

- Obter e preparar um banco de imagens de faces humanas;
- Implementar um classificador de emoções utilizando redes neurais artificiais com a tipologia de Aprendizado Profundo ou Deep Learning;

- Avaliar a precisão com a qual a rede neural classificou corretamente as sete expressões faciais básicas e universais; e
- Utilizar técnicas de visão computacional para reconhecer rostos humanos em fotografias e, a partir da detecção dos rostos, aplicar a rede neural obtida para classificar as emoções.

## 1.5 JUSTIFICATIVA

Pesquisas em redes neurais profundas para aprendizagem de máquina vêm se tornando cada vez mais eficientes em aplicações em diversas áreas, por lidar com grande quantidade de dados não lineares em diversas abstrações através das camadas convolucionais (BENGIO, 2009).

A detecção e classificação de expressões é uma excelente oportunidade de empregar as redes neurais convolucionais (CNN), sendo um caminho alternativo em relação a outras técnicas já utilizadas, como por exemplo:

- Principal Component Analysis (PCA) e Independent Component Analysis (ICA) (DRAPER et al., 2003);
- Support Vector Machine (SVM) (HEISELE, POGGIO, 2001); e
- Hidden Markov Model (HMM) (RABINER, 1989).

Embora seja uma das tarefas rotineiras do homem, o processo de identificação das expressões faciais não é uma tarefa trivial para as máquinas, e foi historicamente subestimada no campo da Inteligência Artificial (IA). Segundo Baltrusaitis (2014), antes mesmo de abordagens com IA a detecção de expressões faciais já possuía importantes ramificações e aplicações em diversas áreas, como:

- Na área médica: tecnologias assistivas para detectar dor;
- Na educação: adaptação do ensino ao estado emocional do aluno;
- No comércio: detecção da afetividade ao produto; e
- Na aviação: sistemas de monitoração de cansaço para pilotos.

Todas essas áreas de aplicação abrem espaço para a abordagem de classificação de expressões faciais por meio da técnica de convolução inclusa à rede neural.



## 2 REFERENCIAL TEÓRICO

### 2.1 VISÃO COMPUTACIONAL

Numerosos seres vivos têm no seu sistema de visão o elemento sensorial mais importante para a sua sobrevivência e para as suas condições de vida. A importância do sistema de visão prende-se com a riqueza de informação que este facultta, não só em termos quantitativos, mas também qualitativos. Tais informações permitem, por exemplo, a detecção e o seguimento de certos alvos (predadores, alimentos e outros), a determinação de obstáculos na sua trajetória, em suma, informações sobre o ambiente que rodeia cada ser (TAVARES, 2000).

Neste contexto, não é surpreendente que a comunidade científica tenha, nos últimos tempos, realizado intensos esforços no sentido de prover sistemas automáticos, isto é sistemas computadorizados, que sejam capazes de executar funções do sistema de visão que são normalmente encontradas nos sistemas equivalentes dos seres vivos, e em especial no sistema visual humano. A tentativa de implementar certas funções do sistema de visão humana em sistemas automáticos pode ser realizada quer ao nível de software, quer ao nível de hardware. Surge, assim, uma área de desenvolvimento científico que é designada por processamento de imagem e visão por computador ou visão artificial (NEVES, 2012).

O processamento de imagem e a visão por computador são normalmente divididos em quatro áreas de atuação:

- **Melhoramento ou realce de imagens:** consiste basicamente na tentativa de melhorar e realçar subjetivamente certas características de uma dada imagem (por exemplo, acentuar contraste, reduzir ruído etc.);
- **Restauração de imagens:** consiste basicamente na tentativa de restaurar imagens que tenham sido degradadas na sua qualidade por um qualquer processo, como por exemplo distorção geométrica, movimento etc.;
- **Compressão de imagens:** consiste basicamente na tentativa de representar uma imagem original de forma mais simples e, portanto, mais leve, sem perder informação; e

- **Análise de imagens:** consiste basicamente em descrever ou interpretar uma dada imagem ou sequência de imagens; isto é, na tentativa de medir, reconhecer, classificar uma imagem ou conjunto de imagens.

As três primeiras áreas costumam ser agrupadas na designação de “processamento de imagem”; a última está mais ligada à visão por computador e aparece por vezes associada à inteligência artificial.

Naturalmente surgem situações em visão por computador em que todas as áreas anteriores aparecem perfeitamente combinadas e integradas. Um exemplo desta combinação pode ser um sistema que procure analisar o movimento de certos objetos a partir de uma sequência de imagens (TAVARES, 1995b). Sistemas assim devem incluir, quase obrigatoriamente, funções de melhoramento das imagens originais (compensação de iluminação, remoção de ruído), de restauração das imagens degradadas geometricamente, de análise das imagens e, porventura, pode ser utilizada compressão (para efeitos de arquivo ou de transmissão).

Na atualidade, surgem cada vez mais aplicações do processamento de imagem e da visão por computador. Como exemplos de tais aplicações podem ser referidos os seguintes (TAVARES, 1995a):

- **Inspeção industrial:** a qualidade do produto tem cada vez mais um papel de importância primordial. Como, geralmente, as funções de inspeção visual humana são bastante rotineiras, cansativas, morosas, e conseqüentemente sujeitas a falhas e erros, surge a necessidade de as automatizar por sistemas computadorizados (utilizando, por exemplo, robôs e manipuladores em tais tarefas). É evidentemente necessário prover estes sistemas de “visão”. Surge, assim, uma área importante da visão por computador em que o objetivo é o controle dimensional de componentes, o controle da sua qualidade superficial ou a verificação da integridade;
- **Compressão de imagens:** quando se pretende armazenar um elevado número de imagens, torna-se essencial diminuir o volume da respectiva informação. Tal redução pode também ser necessária na transmissão de imagens, em que a largura de banda é inevitavelmente reduzida. Técnicas de compressão de imagem desempenham um papel fulcral em inúmeros sistemas

de arquivos e comunicação de imagens, em especial com o advento e proliferação de sistemas multimídia;

- **Aplicações médicas:** na medicina existem bastante imagens de diagnóstico obtidas por diferentes processos e técnicas (como por exemplo por raios-X, ecografia, endoscopia etc.). Tais imagens necessitam de ser processadas no sentido de remover ruído, melhorar algumas características e analisá-las. A análise não é geralmente pretendida com um sentido perfeitamente autônomo, mas como um auxiliar importante ao diagnóstico efetuado pelos especialistas. Não é, assim, surpreendente encontrar um elevado número de aplicações de processamento de imagem e de visão por computador em medicina;
- **Recuperação de imagens degradadas:** certas imagens são obtidas com uma inevitável deterioração; tal pode ser devido às más condições de iluminação, influência de campos eléctricos e/ou magnéticos, às elevadas distâncias de transmissão etc. Nestas situações, é necessário realizar uma melhoria da qualidade das imagens. Por curiosidade, refira-se que uma das primeiras aplicações do processamento de imagem se refere à recuperação das imagens enviadas para a Terra por uma sonda espacial em 1960. A deterioração ficava-se a dever a elevadas restrições acerca do peso do sistema de visão (LIM, 1990), implicando assim que o sistema de imagem a bordo da sonda fosse de reduzida qualidade;
- **Na meteorologia:** pela análise do movimento das nuvens, sistemas de visão por computador podem auxiliar em estudos de previsão do estado do tempo;
- **Em sistemas de tráfego automóvel:** cada vez mais pretende-se dotar os sistemas de gestão de tráfego automóvel atualmente existentes com sistemas de visão por computador. Tal incorporação tem como objetivo tornar a gestão mais flexível, eficiente e rápida; e
- **Na agricultura:** na análise do crescimento e grau de maturação das plantações, a visão por computador, baseada em imagens de detecção remota, surge cada vez mais como um sistema bastante útil para análise e controle.

## 2.2 RECONHECIMENTO FACIAL E COMPUTAÇÃO AFETIVA

Atualmente, imagens de faces são cada vez mais utilizadas como forma de reconhecimento de emoções. Existe uma grande variedade de aplicações tais como

sistemas de reconhecimento de faces humanas, sistemas de vigilância e sistemas de vídeo conferência, que têm como pré-requisito a localização da face e a extração de suas características. Por isso, a comunidade científica tem dedicado esforços para ampliar os estudos e encontrar melhores técnicas para o problema de localizar uma face e extrair suas características.

De acordo com Wong (2001), detectar faces humanas e extrair as características faciais de uma imagem sem restrição é um grande desafio. Uma série de fatores dificultam a detecção, tais como as cores da pele, uso de óculos, barba ou bigode e, em especial, as expressões faciais. Isso faz com que a extração de características faciais se torne difícil e desafiadora.

Várias pesquisas sobre técnicas de reconhecimento de expressões faciais estão sendo desenvolvidas com o propósito de obter soluções para esses problemas. Essas técnicas são muito úteis, pois não exigem a interação do usuário ou o conhecimento dele.

As expressões faciais são geradas a partir de contrações de músculos faciais, que resultam na deformação de características faciais, tais como pálpebras, sobrancelhas, nariz e boca, e resultam em mudanças nas suas posições relativas. A partir destas deformações, modelos de representações podem ser definidos, onde imagens com deformações semelhantes podem pertencer a um determinado modelo. Este processo caracteriza-se como reconhecimento de expressão facial.

A forma para representar uma determinada expressão facial é analisar as suas distinções ou variações entre a imagem da expressão e a sua imagem correspondente sob uma expressão normal. Portanto, alguns métodos de reconhecimento de expressão facial são baseados em uma sequência de imagens ou imagens de um vídeo. Contudo, apenas imagens estáticas estão disponíveis para alguns tipos de aplicações.

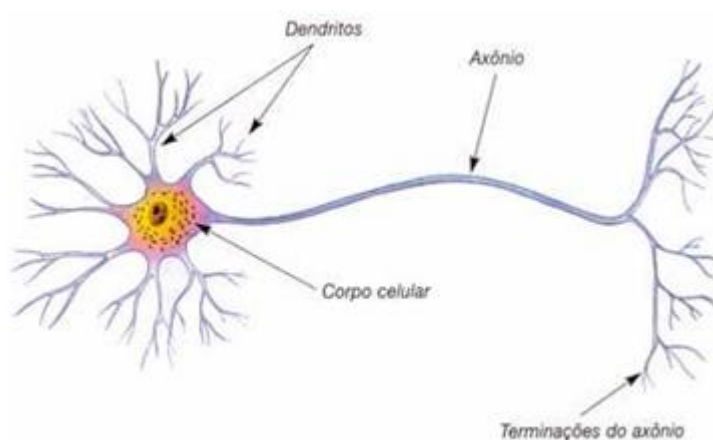
### 2.3 REDES NEURAIIS

Redes Neurais Artificiais (RNA) seguem uma linha de pesquisa da inteligência artificial chamada Conexionismo, o qual se propõe estudar a simulação de comportamentos

inteligentes através de modelos baseados na estrutura do cérebro humano (BITTENCOURT, 1998).

Os componentes estruturais do cérebro são os neurônios, que segundo Machado (1993), são células altamente excitáveis que se comunicam entre si ou com outras células efetadoras (células musculares e secretoras), sendo compostos, em sua maioria, por três componentes principais: dendritos, axônios e corpo celular. Como visto na Figura 1.

Figura 1: Neurônio



FONTE: retirado de MACHADO (1993).

Os dendritos são pequenas estruturas altamente ramificadas e de superfície irregular (HAYKIN, 2001), responsáveis por receber os estímulos externos ou os sinais dos outros neurônios. Geralmente são curtos e ramificam-se profusamente, à maneira de galhos de uma árvore, em ângulo agudo, originando dendritos de menor diâmetro. Apresentam contorno irregular e são especializados em receber estímulos, traduzindo-os em alterações do potencial de repouso da membrana. Tais alterações envolvem entrada ou saída de determinados íons e podem expressar-se por pequena despolarização ou hiperpolarização (MACHADO, 1993).

O corpo celular contém o núcleo e o citoplasma com suas organelas, sendo responsável pela manutenção fisiológica do neurônio (MACHADO, 1993).

Já o axônio é um prolongamento longo e fino que se origina no corpo celular, cuja função é receber o sinal vindo do corpo e transmiti-lo para o próximo neurônio ou célula efetadora.

Os neurônios, principalmente através das suas terminações axônicas, entram em contato com outros neurônios, passando-lhes informações. Os locais de tais contatos são denominados sinapses, ou, mais precisamente, sinapses inter-neuronais.

Existem vários tipos de sinapses, a mais comum é a sinapse química, onde um processo pré-sináptico transforma o sinal elétrico em uma substância neurotransmissora saindo do axônio, que se difunde através da junção sináptica chegando ao dendrito ou célula atuadora, que por sua vez, efetua um processo pós-sináptico transformando os neurotransmissores novamente em sinais elétricos (HAYKIN, 2001).

As sinapses podem ser excitatórias, as quais aumentamos o peso sináptico obtendo mais chances de o sinal ser propagado, ou inibitórias, que fazemos contrário. O número de moléculas de neurotransmissores que se combinam com a membrana receptora dos dendritos é chamado de peso sináptico (BRANDÃO, 2004).

Segundo Bittencourt (1998), os pioneiros no estudo das RNA foram o neurofisiologista, filósofo e poeta norte-americano Warren MacCulloch, e o lógico Walter Pitts, os quais desenvolveram o primeiro modelo matemático de um neurônio.

### **2.3.1 Aplicações das RNA**

Uma aplicação importante das RNA é a classificação de imagens. Ranzato (2012) realizou uma série de experimentos que envolvem a aplicação de várias técnicas de paralelismo e de processamento assíncrono para treinar redes neurais em modelos com 1 bilhão de conexões, a partir de um conjunto de dados com 10 milhões de imagens, cada uma das quais com 200 x 200 pixels.

Robôs podem ser treinados para navegar por um ambiente recebendo como estímulo apenas pixels de imagens desse ambiente. Segundo Zhang (2015), essa área promissora é também chamada de aprendizagem profunda por reforço (Deep Reinforcement Learning).

Agentes de software também foram treinados para realizar diversas tarefas complexas: jogar Go no nível de um mestre mundial, um desafio antigo da IA apenas recentemente sobrepujado (SILVER et al., 2016); aprender a jogar um jogo clássico

(“arcade”) por meio da análise de imagens capturadas desse jogo e do resultado de ações (MNIH et al., 2015), a princípio, aleatórias.

Na área chamada aprendizado multimodal, as redes são treinadas com texto mais imagem, ou áudio mais vídeo etc. Dois trabalhos representativos dessa área são os de Ngiam et al. (2011) e Karpathy et al. (2014). Apresentam um modelo que gera descrições de linguagem natural das imagens e das suas regiões. Essa abordagem utiliza conjuntos de dados de imagens e suas descrições de sentença para aprender sobre as correspondências intermodais entre linguagem e dados visuais.

As RNA possuem uma ampla área de aplicação como previsão de risco de crédito (STEINER e ALBERT, 2007), medicina (BLAZADONAKIS e ZERVAKIS, 2008) e em polímeros (CONTANT et al., 2004).

### **2.3.2 Como as RNA funcionam**

A transmissão do sinal de um neurônio a outro no cérebro é um processo químico complexo, no qual substâncias específicas são liberadas pelo neurônio transmissor. O efeito é um aumento ou uma queda no potencial elétrico no corpo da célula receptora. Se este potencial alcançar o limite de ativação da célula, um pulso ou uma ação de potência e duração fixa é enviado para outros neurônios. Diz-se então que o neurônio está ativo.

De forma semelhante à sua contrapartida biológica, uma RNA possui um sistema de neurônios artificiais (também conhecidas como unidades de processamento ou simplesmente unidades). Cada unidade realiza uma computação baseada nas demais unidades com as quais está conectada (BEZERRA, 2016).

Os neurônios de uma RNA são organizados em camadas, com conexões entre neurônios de camadas consecutivas. As conexões entre unidades são ponderadas por valores reais denominados pesos. A RNA mais simples possível contém uma única camada, composta por um único neurônio. Redes dessa natureza são bastante limitadas, porque resolvem apenas processos de decisão binários (nos quais há apenas duas saídas possíveis) e linearmente separáveis (funções booleanas como AND e OR).

Por outro lado, é possível construir redes mais complexas (capazes de modelar processos de decisão não linearmente separáveis) por meio de um procedimento de composição de blocos de computação mais simples organizados em camadas (BEZERRA, 2016).

## 2.4 TIPOS E FUNCIONAMENTO DAS RNA

### 2.4.1 Redes autocodificadoras

Conhecidas como autocodificadoras ou redes autoassociativas, são classes de redes neurais que podem ser treinadas para aprender de forma não supervisionada por características (features) a partir de um conjunto de dados.

Estas características são úteis para uso posterior em tarefas de aprendizado supervisionado, tais como reconhecimento de objetos e outras tarefas relacionadas à visão computacional.

Curiosamente, uma rede neural autocodificadora é normalmente treinada, não para prever alguma classe, mas para reproduzir na sua saída a própria entrada. Essa família de redes permite o aprendizado de representações mais concisas de um conjunto de dados (BEZERRA, 2016).

### 2.4.2 Redes convolucionais

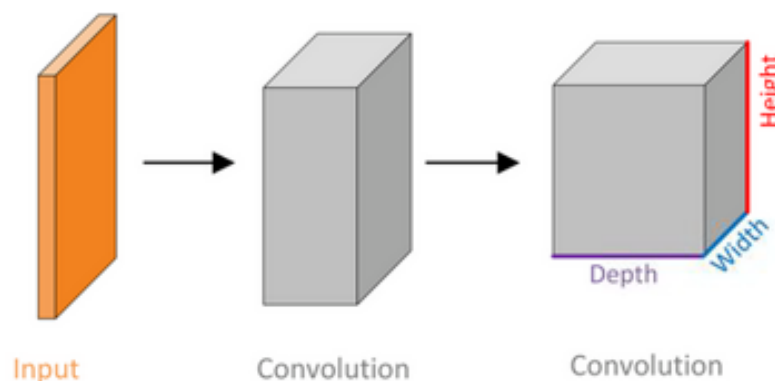
Redes neurais convolucionais (*convolutional neural networks*, CNN) se inspiram no funcionamento do córtex visual (LECUN et al., 1998; ZEILER, 2012) e se baseiam em algumas ideias básicas, a saber: campos receptivos locais (*local receptive fields*), compartilhamento de pesos (*shared weights*), convolução (*convolution*) e subamostragem (*subsampling* ou *pooling*). Neste sentido, o termo convolucional adota um significado que lembra campos receptivos em sistemas nervosos biológicos reais.

As CNN têm uma arquitetura distinta, projetada para imitar a maneira pela qual os cérebros de animais reais são organizados para funcionar: ao invés de cada neurônio de cada camada se conectar a todos os neurônios da camada seguinte (*perceptron* multicamada), os neurônios são organizados em uma estrutura tridimensional (ver



Figura 2) de modo a levar em conta as relações espaciais entre os diferentes neurônios em relação ao dados originais.

Figura 2: Camadas CNN arranjadas em três dimensões



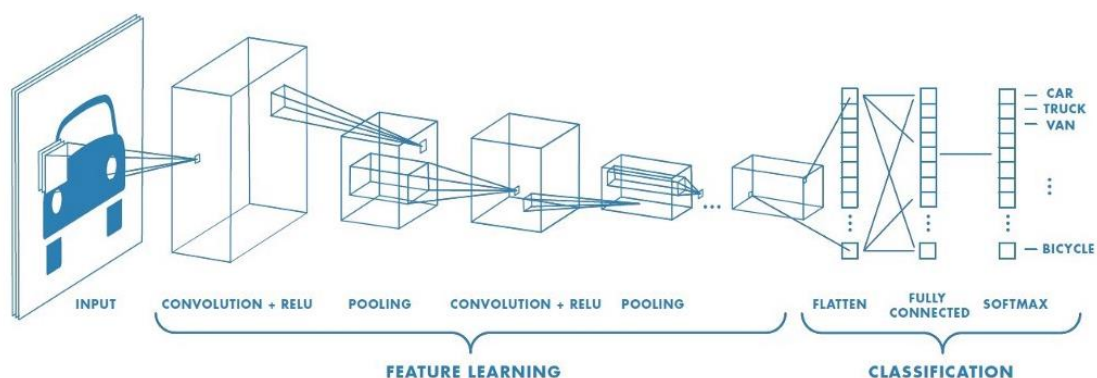
FONTE: adaptado de SAS Inc. (2020)

Como as CNN são usadas principalmente no campo da visão computacional, os dados que os neurônios representam são tipicamente uma imagem: cada neurônio de entrada representa um pixel da imagem original.

A primeira camada de neurônios é composta de todos os neurônios de entrada; neurônios na próxima camada receberão conexões de alguns dos neurônios de entrada (pixels), mas não todos, como seria o caso em um MLP (Perceptron Multicamadas) e em outras redes neurais tradicionais. Assim, em vez de cada neurônio receber conexões de todos os neurônios da camada anterior, as CNN usam um layout semelhante a um campo receptivo, no qual cada neurônio recebe conexões apenas de um subconjunto de neurônios na camada anterior (inferior).

O campo receptivo de um neurônio em uma das camadas inferiores engloba apenas uma pequena área da imagem, enquanto o campo receptivo de um neurônio em camadas subsequentes envolve uma combinação de campos receptivos de vários (mas não todos) neurônios no campo. Desta forma, cada camada sucessiva é capaz de aprender características cada vez mais abstratas da imagem original (Figura 3).

Figura 3: Exemplo de rede neural convolucional



FONTE: retirado de MathWorks Inc. (s.d.)

Pensa-se que o uso de campos receptivos dessa maneira dê às CNN uma vantagem no reconhecimento de padrões visuais quando comparados a outros tipos de redes neurais.

É provável que um determinado detector de alguma característica elementar seja útil em diferentes regiões da imagem de entrada. Para levar isso em consideração, em uma CNN, as unidades de uma determinada camada são organizadas em conjuntos disjuntos, cada um dos quais é denominado um mapa de característica (*feature map*), também conhecido como filtro.

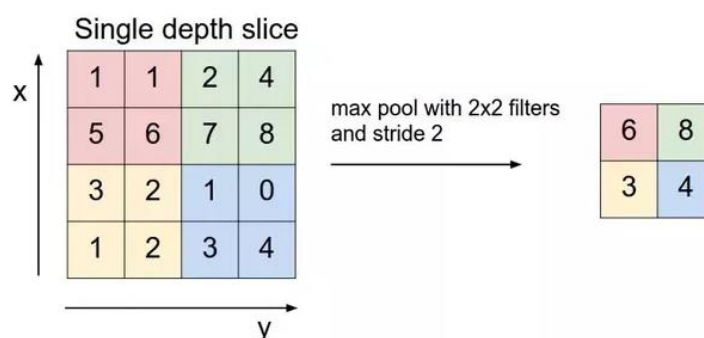
As unidades contidas em um mapa de características são únicas na medida em que cada uma delas está ligada a um conjunto de unidades (isto é, ao seu campo receptivo) diferente na camada anterior. Além disso, todas as unidades de um mapa compartilham os mesmos parâmetros (a mesma matriz de pesos e viés).

O resultado disso é que essas unidades dentro de um mapa servem como detectores de uma mesma característica, mas cada uma delas está conectada a uma região diferente da imagem. Portanto, em uma CNN, uma camada oculta é segmentada em diversos mapas de características em que cada unidade de um mapa tem o objetivo realizar a mesma operação sobre a imagem de entrada, com cada unidade aplicando essa operação em uma região específica dessa imagem.

Outra operação importante utilizada em uma CNN é a subamostragem. Em processamento de imagens, a subamostragem de uma imagem envolve reduzir a sua resolução sem alterar significativamente o seu aspecto. No contexto de uma CNN, a subamostragem reduz a dimensionalidade de um mapa de característica fornecido como entrada e produz outro mapa de característica, uma espécie de resumo do primeiro.

Há várias formas de subamostragem aplicáveis a um mapa de característica: selecionar o valor máximo (*max pooling*), a média (*average pooling*) ou a norma do conjunto (*L2 pooling*), entre outras. Como exemplo, partindo de um mapa de característica de tamanho 4 x 4, a Figura 4 apresenta outro mapa de característica, resultante da operação de subamostragem com o uso de filtros de tamanho igual a 2 x 2 e tamanho do passo igual.

Figura 4: processo de discretização baseado em amostragem



FONTE: retirado de Prabhu (2018)

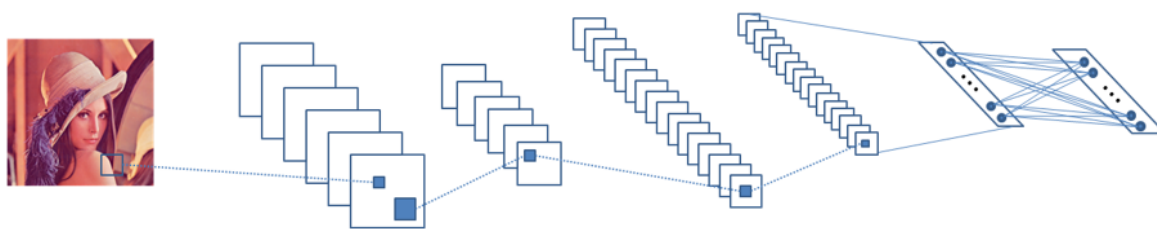
Em geral, uma CNN possui diversos tipos de camadas: camadas de convolução, camadas de subamostragem, camadas de normalização de contraste e camadas completamente conectadas.

Na forma mais comum de arquitetar uma CNN, a rede é organizada em estágios. Cada estágio é composto por uma ou mais camadas de convolução em sequência, seguidas por uma camada de subamostragem, que por sua vez é seguida (opcionalmente) por uma camada de normalização.

Uma CNN pode conter vários estágios empilhados após a camada de entrada (que corresponde à imagem). Após o estágio final da rede, são adicionadas uma ou mais camadas completamente conectadas.

A Figura 5 é um exemplo esquemático de uma CNN na qual, após a camada de entrada (que corresponde aos pixels da imagem), temos uma camada de convolução composta de 6 mapas de características (representados como planos na figura), seguida de uma camada de subamostragem, completando o primeiro estágio. Essa figura ainda ilustra um segundo estágio antes das duas camadas completamente conectadas.

Figura 5: arquitetura típica de um CNN



FONTE: adaptado de Lecun et al. (1998)

Redes neurais convolucionais também podem conter as denominadas camadas de normalização de contraste local (*Local Contrast Normalization*, LCN) (JARRETT et al., 2009). Quando utilizada, essa camada é posicionada na saída da camada de subamostragem. Uma camada LCN normaliza o contraste de uma imagem de forma não linear. Em vez de realizar uma normalização global (considerando a imagem como um todo), a camada LCN aplica a normalização sobre regiões locais da imagem, considerando cada pixel por vez.

A normalização pode corresponder a subtrair a média da vizinhança de um pixel em particular ou dividir pela variância dos valores de pixel dessa vizinhança. Esta transformação equipa a CNN com invariância de brilho, propriedade útil no contexto de reconhecimento de imagens.

O outro tipo de camada em uma CNN é a completamente conectada. Em arquiteturas de CNN modernas, é comum encontrar uma ou duas camadas desse tipo antes da camada de saída. Juntamente com as camadas de convolução e subamostragem, as camadas totalmente conectadas geram descritores de características da imagem que podem ser mais facilmente classificados pela camada de saída. É importante notar que os pesos dos mapas de características em cada camada de convolução são aprendidos durante o treinamento.

Por conta de as operações de convolução e subamostragem serem diferenciáveis, uma CNN pode ser também treinada por meio do algoritmo de retropropagação, com a diferença de que os pesos são atualizados considerando a média dos gradientes dos pesos compartilhados. Durante esse treinamento, é comum aplicar a técnica de desligamento às camadas completamente conectadas para evitar o sobreajuste.

#### **2.4.2.1 Aplicações das CNN**

Por meio das redes neurais convolucionais, é possível realizar a detecção de padrões em imagens de forma mais adequada, no sentido de que características dos dados dessa natureza podem ser exploradas para obter melhores resultados.

Desde sua origem, esse tipo de rede tem se mostrado adequado em tarefas que envolvem reconhecimento visual, tais como reconhecimento de caracteres manuscritos e detecção de faces (LECUN,1989; LECUN et al.,1998).

Com o sucesso resultante do uso de arquiteturas profundas para o processamento visual e o surgimento de bases de dados de imagem com milhões de exemplos rotulados (por exemplo, ImageNet, Places), o estado da arte em visão computacional por meio de CNN tem avançado rapidamente.

Essas redes ganharam popularidade em 2012, quando foram usadas para reduzir substancialmente a taxa de erro em uma conhecida competição de reconhecimento de objetos, a ImageNet (KRIZHEVSKY et al., 2012).

O objetivo de cada mapa de característica em uma CNN é extrair características da imagem fornecida. Portanto, uma CNN também desempenha o papel de extrator automático de características da imagem. De fato, desde 2012, a área de Visão Computacional tem sido invadida por abordagens que usam CNN para tarefas de classificação e detecção de objetos em imagens, em substituição às abordagens anteriores que envolviam a extração manual das características das imagens para posteriormente aplicar algum método de classificação, como por exemplo as máquinas de vetores suporte (*support vector machines*, SVM).

Aplicações mais recentes de CNN envolvem a detecção de pedestres (SERMANET, 2013) e de placas de sinais de trânsito (SERMANET, 2011).

### 3 METODOLOGIA

Para alcançar o objetivo de reproduzir o treinamento, teste, validação e aplicação de um modelo de rede neural convolucional para classificar expressões faciais em imagens de faces humanas, as seguintes etapas metodológicas foram estabelecidas:

- Busca de material de referência no treinamento de redes neurais para detecção de expressões faciais que possa ser utilizado como modelo a ser reproduzido, testado e aplicado;
- Obtenção de uma base adequada de imagens, incluindo o processamento necessário;
- Definição da infra-estrutura necessária ao trabalho (linguagem de programação, ambiente, hardware e software necessário, mecanismos de controle de versão etc.);
- Definição da arquitetura da CNN a ser utilizada;
- Treinamento, validação e teste da CNN, incluindo a definição da acurácia do modelo em identificar corretamente as expressões faciais; e
- Aplicação do modelo treinado em outra fotografia, utilizando técnicas de reconhecimento facial e detecção da expressão facial.

Essas etapas são descritas em detalhes a seguir.

#### 3.1 MATERIAL DE REFERÊNCIA

Dentre os inúmeros materiais e referências a respeito do tema, o trabalho escolhido como base a ser reproduzido e testado foi o “**Deep Learning Lab: fer2013**”, publicado por Kinli (2018). O trabalho de Kinli se mostrou particularmente útil e adequado pois:

- Pode ser aplicado a diversas bases de imagens;
- Utiliza código relativamente simples para treinar a rede neural;
- A arquitetura de rede neural proposta é simples, mas promissora; e
- Está disponível na internet para qualquer interessado.

### 3.2 BASE DE DADOS

Utilizamos uma base de imagens faciais open-source, a **fer2013**, criada por Pierre-Luc Carrier e Aaron Courville. Essa base de imagens foi compartilhada publicamente em uma competição Kaggle em 2013 e permanece à disposição na internet para qualquer interessado (Moorsy, 2018).

A base fer2013 contém 35.887 imagens faciais em tons de cinza, todas rotuladas com as sete emoções básicas e universais conforme a tabela abaixo:

Tabela 1: Quantidade de imagens por emoção, fer2013

Código	Emoção	Qtd. de Imagens
0	Raiva/Irritado	4.593
1	Nojo	547
2	Medo	5.121
3	Feliz	8.989
4	Triste	6.077
5	Surpresa	4.002
6	Neutro	6.198

FONTE: preparado pelo autor

Alguns exemplos das imagens da fer2013 podem ser vistos na Figura 6, abaixo. A base contém figuras bem diversificadas (em termos de cor da pele, uso de óculos, barba, cabelos e acessórios como microfones, bonés e faixas) que representam diversas situações reais do cotidiano.

Figura 6: Exemplos de imagens da base fer2013



FONTE: Retirado de Kinli (2018)

As imagens foram processadas para padronizar o tamanho em 48 x 48 pixels, e normalizar os tons de cinza.

### 3.3 DEFINIÇÃO DA INFRA-ESTRUTURA

Como o treinamento de uma rede neural, mesmo com um número relativamente pequeno de imagens como na base do fer2013, exige grande poder computacional – testes realizados em um desktop com processador Intel Core i5 com 8 GB de RAM mostraram que o tempo de treinamento de cada época da rede neural estava variando entre 2 e 4 horas – optou-se por utilizar a infraestrutura do Google Colaboratory (Google Inc., s.d.).

Através do Google Colab foi possível escrever e executar código Python diretamente no browser, sem a necessidade de configuração prévia de hardware e software. Além disso foi possível ter acesso gratuito a uma GPU Tesla T4 da NVIDIA (com 320 Tensor Cores, 2.560 CUDA Cores, 16 GB RAM e 8,1 TFLOPS), o que aumentou consideravelmente a performance do treinamento de cada época (20 segundos em média). As especificações da GPU utilizada estão na Figura 7, abaixo.

Figura 7: GPU utilizada no treinamento da rede neural

```

!nvidia-smi

Fri Jul 10 02:20:47 2020
+-----+
| NVIDIA-SMI 450.36.06      Driver Version: 418.67      CUDA Version: 10.1   |
+-----+
| GPU   Name           Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf    Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
|                                           | MIG M.         |
+-----+-----+
|  0   Tesla T4              Off      | 00000000:00:04:0 Off |                    |
| N/A   75C    P0              34W /  70W |  219MiB / 15079MiB |             0%      Default |
|                                           |                 ERR! |
+-----+-----+

+-----+
| Processes:                                                       GPU Memory |
|  GPU   GI    CI        PID   Type   Process name                      Usage    |
|-----+-----+
| No running processes found
+-----+

```

FONTE: captura de tela realizada pelo autor

Foram utilizadas bibliotecas Python adequadas (numpy, pandas, matplotlib, sklearn etc.) incluindo o TensorFlow, que é uma biblioteca de código aberto do Google para computação numérica e que contém várias funções e procedimentos para treinamento de redes neurais.

Uma vantagem extra do uso do Google Colab foi a possibilidade de incluir texto em Markdown. Isso possibilitou documentar o código à medida em que era escrito.



### 3.4 ARQUITETURA DA REDE NEURAL

A arquitetura da rede neural proposta por Kinli (2018), reutilizada neste trabalho, pode ser resumida, da entrada até a saída, no seguinte:

- (2 x CONV (3x3)) – MAXP (2x2) – DROPOUT (0.5)
- (2 x CONV (3x3)) – MAXP (2x2) – DROPOUT (0.5)
- (2 x CONV (3x3)) – MAXP (2x2) – DROPOUT (0.5)
- (2 x CONV (3x3)) – MAXP (2x2) – DROPOUT (0.5)
- DENSE (512) – DROPOUT (0.5)
- DENSE (256) – DROPOUT (0.5)
- DENSE (128) – DROPOUT (0.5)

Além da arquitetura global acima, na primeira camada de convolução foi aplicada regularização L2 e, em todas as camadas de convolução exceto a primeira, foi realizada normalização em lote.

A função “RELU” foi escolhida como a função de ativação em todas as camadas e a função “SOFTMAX” foi estimou as probabilidades de cada emoção ao final.

Uma decisão importante foi o número de épocas de treinamento, ou seja, quantas passagens completas do conjunto de dados (épocas) devem ser usadas. Se usarmos poucas épocas poderemos ter problemas de *underfitting*, não aprender tudo o que for possível com os dados de treinamento. Se usarmos muitas épocas poderemos ter problemas de *overfitting*, aprender “demais” e ajustar o ruído nos dados treinamento, além das características importantes.

Seguindo o recomendado por Kinli (2018), usamos 100 épocas de treinamento, associadas a uma função de “parada” que analisa o resultado à medida que o treinamento é realizado e interrompe o processo se não houver mais progresso na aprendizagem.

Para o treinamento, a rede neural será alimentada com lotes de imagens, cada lote com 64 imagens, dimensionadas para 48 x 48 pixels, normalizadas e pré-processadas.

### 3.5 TREINAMENTO, TESTE E VALIDAÇÃO

A base de imagens fer2013 será dividida em conjuntos de treinamento, teste e validação, conforme o preconizado por Kinli (2018) e o treinamento do modelo será feito com o conjunto de treinamento. Esse treinamento será testado e validado com os outros conjuntos de dados.

A precisão da rede neural, isto é, a capacidade da rede de classificar corretamente a emoção de uma pessoa de acordo com sua expressão facial, será avaliada através da acurácia global: a proporção total de acertos da rede neural.

Será gerada uma matriz de confusão para ilustrar os acertos e erros do modelo.

### 3.6 APLICAÇÃO DA REDE NEURAL

Após o treinamento, teste e validação da rede neural, utilizaremos o modelo treinado pronto para identificar a expressão facial em uma fotografia e verificaremos se o modelo final pode ser utilizado em aplicações reais para detecção de emoções humanas.

## 4 TREINAMENTO DA REDE NEURAL

Como a utilização direta do Python no Google Colab permitiu que o código ficasse bastante conciso e documentado, optamos por apresentar nesta seção os códigos utilizados juntamente com os resultados obtidos ao invés de colocar o código separado em um anexo. Demonstraremos assim todo o processo realizado e os resultados obtidos em uma seqüência lógica de fácil compreensão.

### 4.1 AMBIENTE E BIBLIOTECAS NECESSÁRIAS

Inicialmente o ambiente do Google Colab foi configurado para usar uma GPU Tesla T4 da NVIDIA, através do painel de configuração web:

Figura 8: Configuração para uso de GPU



FONTE: captura de tela realizada pelo autor

Depois todas as bibliotecas necessárias foram importadas com o seguinte código:

Figura 9: Carregamento de bibliotecas

```
[1] # Carrega bibliotecas necessárias
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from google.colab.patches import cv2_imshow
from google.colab import drive
%tensorflow_version 2.x
import tensorflow
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: captura de tela realizada pelo autor

## 4.2 BASE E PRÉ-PROCESSAMENTO DAS IMAGENS

A base de dados fer2013, em formato CSV, foi salva em um diretório do Google Drive, permitindo assim o acesso direto aos dados de dentro do Google Colab.

Figura 10: Montagem do Google Drive para acesso aos dados

```
[2] # Monta o Google Drive para acesso à base de imagens
drive.mount('/content/drive')

Go to this URL in a browser: https://accounts.google.com/

Enter your authorization code:
.....
Mounted at /content/drive
```

FONTE: captura de tela realizada pelo autor

O acesso à base de dados de imagens foi feito lendo-se o arquivo CSV e colocando os dados em um *data frame* chamado “data”. Também verificamos se os dados estão corretos:

Figura 11: Leitura dos dados para um *data frame*

```
# Lê o banco de imagens:
data = pd.read_csv('/content/drive/My Drive/Colab Notebooks/fer2013.csv')

# Verifica se o data frame está ok:
data.tail()
```

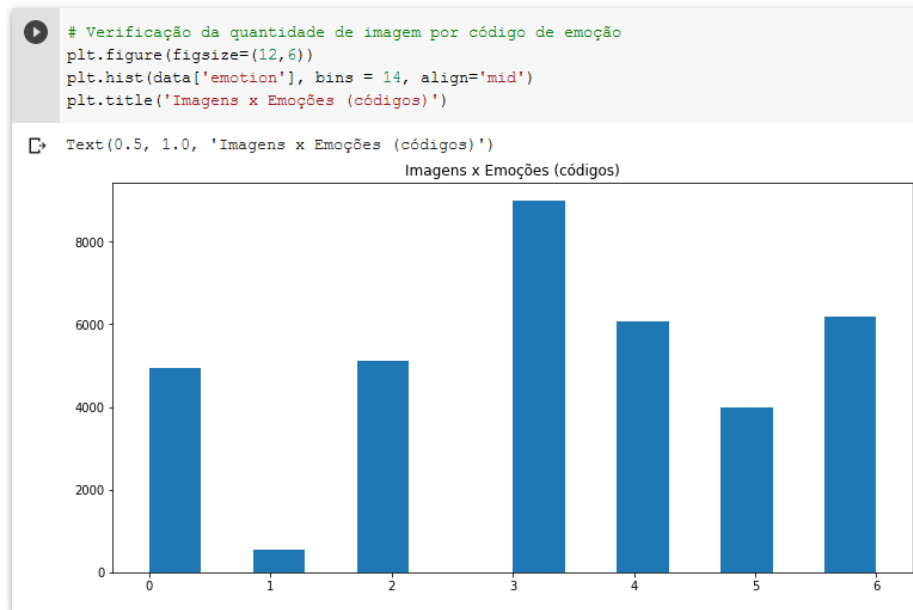
	emotion	pixels	Usage
35882	6	50 36 17 22 23 29 33 39 34 37 37 39 43 48 5...	PrivateTest
35883	3	178 174 172 173 181 188 191 194 196 199 200 20...	PrivateTest
35884	0	17 17 16 23 28 22 19 17 25 26 20 24 31 19 27 9...	PrivateTest
35885	3	30 28 28 29 31 30 42 68 79 81 77 67 67 71 63 6...	PrivateTest
35886	2	19 13 14 12 13 16 21 33 50 57 71 84 97 108 122...	PrivateTest

FONTE: captura de tela realizada pelo autor

Como a quantidade de imagens de cada emoção é diferente, geramos um gráfico para facilitar a compreensão dessa diferença.

A Figura 12, na próxima página, demonstra que a emoção “nojo” (código 1) tem um número muito pequeno de imagens em comparação com as outras (que têm, no mínimo, 4000 imagens). A emoção “nojo” conta apenas com 547 imagens.

Figura 12: Distribuição da quantidade de imagens, por emoção



FONTE: captura de tela realizada pelo autor

Para que as imagens estejam em condições adequadas para o treinamento da rede neural, executamos diversas ações de pré-processamento que incluíram obter a lista dos pixels das imagens, criar uma lista com as faces, normalizar as imagens e obter uma matriz com os *labels* (as emoções corretas de cada imagem).

Figura 13: Pré-processamento das imagens

```

# Pré-processamento das imagens

# Obtém lista de pixels
pixels = data['pixels'].tolist()

# Cria e popula lista de faces
faces = []
for pixel_sequence in pixels:
    face = [int(pixel) for pixel in pixel_sequence.split(' ')]
    face = np.asarray(face).reshape(48, 48)
    faces.append(face)

# Transforma lista em array e ajusta dimensão
faces = np.asarray(faces)
faces = np.expand_dims(faces, -1)

# Cria função para normalizar:
def normalizar(x):
    x = x.astype('float32')
    x = x / 255.0
    return x

# Normaliza as faces
faces = normalizar(faces)

# Obtém labels:
emotions = pd.get_dummies(data['emotion']).values

```

FONTE: captura de tela realizada pelo autor

### 4.3 DIVISÃO EM CONJUNTOS DE TREINAMENTO, TESTE E VALIDAÇÃO

Antes de definir a arquitetura da rede neural, dividimos a base de imagem em conjuntos de treinamento, teste e validação conforme o preconizado por Kinli (2018):

Figura 14: Divisão dos conjuntos de imagens

```
# Divide os conjuntos
X_train, X_test, y_train, y_test = train_test_split(faces, emotions, 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)

# Salva
np.save('mod_xtest', X_test)
np.save('mod_ytest', y_test)

# Verifica quantidades:
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: captura de tela realizada pelo autor

### 4.4 MONTAGEM DA ARQUITETURA DA REDE NEURAL

Para implementar a arquitetura definida na Seção 3.4, inicialmente, criamos algumas variáveis importantes que serão passadas como parâmetros aos procedimentos de treinamento:

Figura 15: Parâmetros para a rede neural

```
# Parâmetros da rede neural
num_features = 64
num_labels = 7
batch_size = 64
epochs = 100
width, height = 48, 48
```

FONTE: captura de tela realizada pelo autor

Esses parâmetros definem a quantidade de características que serão aprendidas, a quantidade de emoções diferentes, o tamanho do lote de imagens, a quantidade de épocas de treinamento e o tamanho em pixels de cada imagem.

O próximo passo agora é definir a arquitetura propriamente dita da rede neural, criando as camadas de convolução, informando as funções de ativação, como será feito a amostragem e demais configurações. Para isso usamos a seguinte configuração:

Figura 16: Arquitetura da rede neural

```

# Arquitetura da rede neural
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: captura de tela realizada pelo autor

A configuração da rede neural pode ser verificada facilmente:

Figura 17: Resumo da rede neural (output integral omitido)

```

# Resumo e configuração da rede neural
model.summary()

```

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 46, 46, 64)	640
conv2d_3 (Conv2D)	(None, 46, 46, 64)	36928
batch_normalization_1 (Batch Normalization)	(None, 46, 46, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 23, 23, 64)	0

FONTE: captura de tela realizada pelo autor

Para finalizar a configuração da rede neural, definimos arquivos de output (para salvar o modelo a ser treinado), definimos funções auxiliares (para redução de platô, checkpointer para salvar o melhor modelo e para interromper o treinamento se não houver mais melhora) e compilamos o modelo:

Figura 18: Finalização da arquitetura da rede neural

```
# Arquivos de output
arquivo_modelo = 'modelo_final.h5'
arquivo_modelo_json = 'modelo_final.json'

# Função auxiliar para escapar platôs
lr_reducer = ReduceLRonPlateau(monitor='val_loss', factor=0.9, patience=3, verbose=1)

# Função auxiliar para parar o treinamento e não houver melhora:
early_stopper = EarlyStopping(monitor='val_loss', min_delta=0, patience=8, verbose=1, mode='auto')

# Função auxiliar para salvar o modelo no arquivo de output
checkpointer = ModelCheckpoint(arquivo_modelo, monitor='val_loss', verbose=1, save_best_only=True)

# Compilando o modelo
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-7),
              metrics=['accuracy'])
```

FONTE: captura de tela realizada pelo autor

#### 4.5 TREINAMENTO, TESTE E VALIDAÇÃO

Com tudo pronto, basta agora iniciarmos o treinamento da rede neural. A figura abaixo mostra como foi realizado o treinamento (o output detalhado, pela sua extensão, foi omitido):

Figura 19: Treinamento, teste e validação da rede neural

```
# Salva resultados no JSON:
model_json = model.to_json()
with open(arquivo_modelo_json, 'w') as json_file:
    json_file.write(model_json)

# Inicia o treinamento, teste e validação da rede neural
modelo = 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: captura de tela realizada pelo autor

A rede neural realizou 45 épocas de treinamento quando a função auxiliar de parada antecipada foi ativada. O trecho final do output do treinamento pode ser visto na figura a seguir:



Figura 20: Modelo treinado com 45 épocas

```

Epoch 00043: val_loss did not improve from 0.99502
455/455 [=====] - 20s 45ms/step - loss: 0.6931 - accuracy: 0.7549 - val_loss: 1.0716 - val_accuracy: 0.6356 - lr: 7.2900e-04
Epoch 44/100
454/455 [=====>.] - ETA: 0s - loss: 0.6753 - accuracy: 0.7603
Epoch 00044: val_loss did not improve from 0.99502
455/455 [=====] - 20s 45ms/step - loss: 0.6752 - accuracy: 0.7604 - val_loss: 1.0502 - val_accuracy: 0.6582 - lr: 6.5610e-04
Epoch 45/100
455/455 [=====] - ETA: 0s - loss: 0.6541 - accuracy: 0.7628
Epoch 00045: val_loss did not improve from 0.99502
455/455 [=====] - 20s 45ms/step - loss: 0.6541 - accuracy: 0.7628 - val_loss: 1.1443 - val_accuracy: 0.6477 - lr: 6.5610e-04
Epoch 00045: early stopping

```

FONTE: captura de tela realizada pelo autor

Fizemos então uma avaliação gráfica da evolução da acurácia do treinamento de acordo com o número de épocas:

Figura 21: Avaliação gráfica da acurácia

```

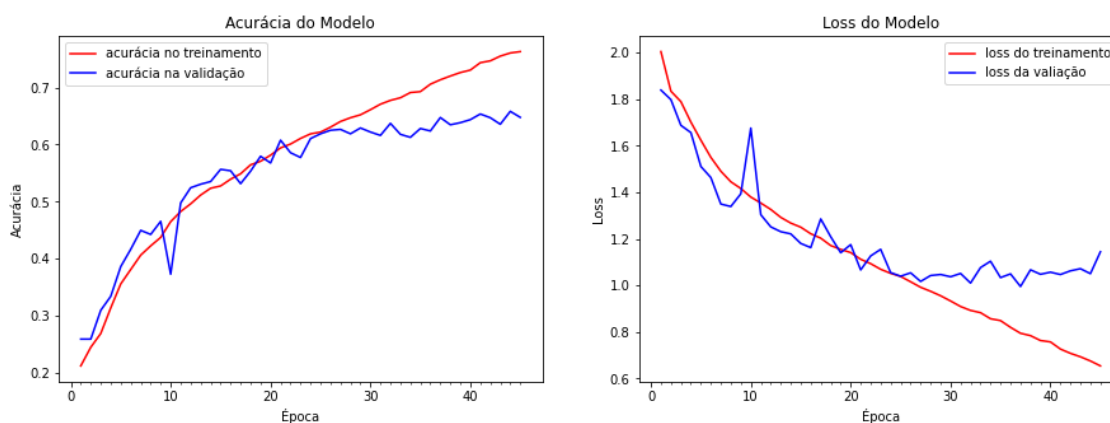
# Função para gerar os gráficos
def plota_historico_modelo(modelo):
    fig, axs = plt.subplots(1, 2, figsize=(15,5))
    axs[0].plot(range(1, len(modelo.history['accuracy']) + 1), modelo.history['accuracy'], 'r')
    axs[0].plot(range(1, len(modelo.history['val_accuracy']) + 1), modelo.history['val_accuracy'], 'b')
    axs[0].set_title('Acurácia do Modelo')
    axs[0].set_ylabel('Acurácia')
    axs[0].set_xlabel('Época')
    axs[0].set_xticks(np.arange(1, len(modelo.history['accuracy']) + 1,
                                len(modelo.history['accuracy']) / 10))
    axs[0].legend(['acurácia no treinamento', 'acurácia na validação'], loc = 'best')

    axs[1].plot(range(1, len(modelo.history['loss']) + 1), modelo.history['loss'], 'r')
    axs[1].plot(range(1, len(modelo.history['val_loss']) + 1), modelo.history['val_loss'], 'b')
    axs[1].set_title('Loss do Modelo')
    axs[1].set_ylabel('Loss')
    axs[1].set_xlabel('Época')
    axs[1].set_xticks(np.arange(1, len(modelo.history['loss']) + 1,
                                len(modelo.history['loss']) / 10))
    axs[1].legend(['loss do treinamento', 'loss da validação'], loc = 'best')

    fig.savefig('graficos_modelo_final.png')

# Gera gráficos:
plota_historico_modelo(modelo)

```



FONTE: captura de tela realizada pelo autor

Podemos perceber que a acurácia no conjunto de treinamento continuou melhorando, mas, no conjunto de validação, houve uma estagnação. Esse fato levou à interrupção do treinamento na época 45, já que não havia mais melhora.

A acurácia final do modelo foi de 63,97%:

Figura 22: Acurácia final do modelo

```
# Acurácia e erro de nossa rede neural
scores = model.evaluate(np.array(X_test), np.array(y_test), batch_size = batch_size)
print('Acurácia: ' + str(round(scores[1]*100, 2)) + '%')
```

57/57 [=====] - 1s 14ms/step - loss: 1.2040 - accuracy: 0.6397  
Acurácia: 63.97%

FONTE: captura de tela realizada pelo autor

Já sabemos que a acurácia global do modelo foi de praticamente 64%. Mas como foi a performance do modelo na predição de cada emoção? Isso pode ser visto em uma matriz de confusão.

Preparação dos dados para gerar a matriz de confusão:

Figura 23: Preparação dos dados para gerar matriz de confusão

```
[35] # Preparação para a geração da matriz de confusão:
true_y = []
pred_y = []
x = np.load('mod_xtest.npy')
y = np.load('mod_ytest.npy')
json_file = open(arquivo_modelo_json, 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
loaded_model.load_weights(arquivo_modelo)
y_pred = loaded_model.predict(x)
yp = y_pred.tolist()
yt = y.tolist()
count = 0
for i in range(len(y)):
    yy = max(yp[i])
    yyt = max(yt[i])
    pred_y.append(yp[i].index(yy))
    true_y.append(yt[i].index(yyt))
    if (yp[i].index(yy) == yt[i].index(yyt)):
        count += 1
acc = (count / len(y)) * 100
np.save('truey_mod01', true_y)
np.save('predy_mod01', pred_y)
```

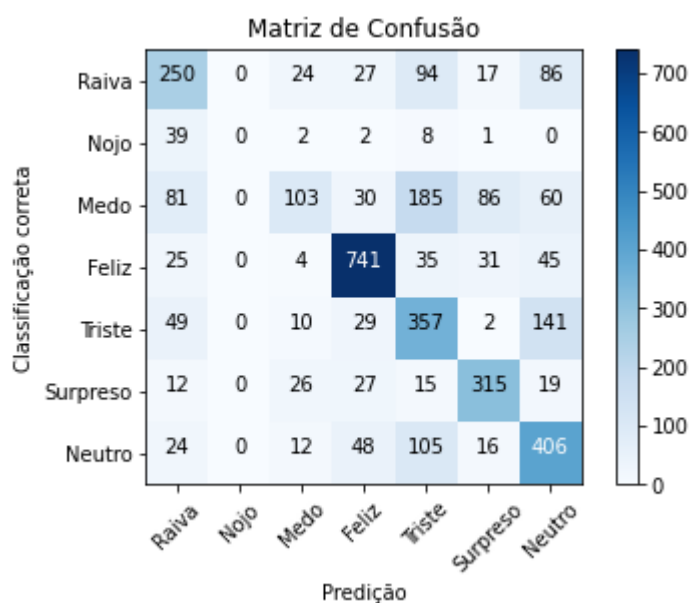
FONTE: captura de tela realizada pelo autor

Com os dados preparados, geramos a matriz de confusão:

Figura 24: Matriz de confusão

```
# Geração da matriz de confusão:
from sklearn.metrics import confusion_matrix
y_true = np.load('truey_mod01.npy')
y_pred = np.load('predy_mod01.npy')
cm = confusion_matrix(y_true, y_pred)
expressoes = ['Raiva', 'Nojo', 'Medo', 'Feliz', 'Triste', 'Surpreso', 'Neutro']
titulo = 'Matriz de Confusão'
import itertools
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title(titulo)
plt.colorbar()
tick_marks = np.arange(len(expressoes))
plt.xticks(tick_marks, expressoes, rotation = 45)
plt.yticks(tick_marks, expressoes)
fmt = 'd'
thresh = cm.max() / 2.
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, format(cm[i, j], fmt), horizontalalignment='center', color='white' if cm[i, j] > thresh else 'black')

plt.ylabel('Classificação correta')
plt.xlabel('Predição')
plt.savefig('matriz_confusao_mod01.png')
```



FONTE: captura de tela realizada pelo autor

A diagonal principal da matriz de confusão mostra as emoções corretamente classificadas pela rede neural. Pode-se perceber, por exemplo, que a emoção “feliz” foi classificada corretamente 741 vezes. É digno de nota que a rede neural não acertou nenhuma predição para a emoção “nojo”, e todas as 52 imagens cuja classificação correta era de “nojo” foram classificadas erroneamente pela rede neural.

## 5 APLICAÇÃO DA REDE NEURAL TREINADA

Agora que temos um modelo de rede neural treinado para a identificação de expressões faciais com acurácia de quase 64%, precisamos aplicar esse modelo para reconhecer expressões em uma imagem real, do cotidiano, que não seja previamente preparada. A imagem de teste escolhida encontra-se a seguir.

Figura 25: Imagem para teste de aplicação da rede neural



FONTE: Arquivo pessoal do autor (foto feita em 1974)

### 5.1 PRÉ-PROCESSAMENTO DA IMAGEM

A imagem foi enviada para o Google Drive em um local que pode ser acessado pelo Google Colab. A leitura da imagem foi feita da seguinte forma:

Figura 26: Leitura inicial da fotografia

```
# Leitura da fotografia:  
imagem = cv2.imread('/content/drive/My Drive/Colab Notebooks/amocim.png')
```

FONTE: captura de tela realizada pelo autor

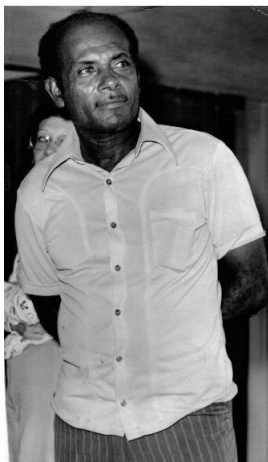
A seguir a imagem foi convertida para tons de cinza (ver Figura 27 e Figura 28, a seguir):

Figura 27: Conversão para tons de cinza

```
# Pré-processamento  
cinza = cv2.cvtColor(imagem, cv2.COLOR_BGR2GRAY)  
cv2_imshow(cinza)
```

FONTE: captura de tela realizada pelo autor

Figura 28: Imagem processada



FONTE: captura de tela realizada pelo autor

## 5.2 DETECÇÃO DA FACE

Ao contrário das imagens da base fer2013, que só mostravam as faces das pessoas, a imagem de teste utilizada aqui mostra uma pessoa da cintura para cima, com um outra pessoa ao fundo aparecendo parcialmente.

É necessário então, antes de aplicar a rede neural, identificar na imagem a região de interesse (roi), ou seja: identificar a região ocupada pela face.

Isso será feito utilizando-se um classificador já treinado para a detecção de faces, o Haar Cascade (KUMAR, 2019). Um arquivo XML com o Haar Cascade foi importado e utilizado para detectar a roi:

Figura 29: Detecção da face na imagem

```
# Detecção da face com Haar Cascade:
cascade_faces = "/content/drive/My Drive/Colab Notebooks/haarcascade_frontalface_default.xml"
face_detection = cv2.CascadeClassifier(cascade_faces)
faces = face_detection.detectMultiScale(cinza, scaleFactor = 1.1,
                                       minNeighbors = 3, minSize = (20,20))

faces


array([[ 68,  34, 114, 114],
       [ 20, 136,  79,  79]], dtype=int32)
```

FONTE: captura de tela realizada pelo autor

Agora que temos as informações da região de interesse, basta extrair essa região específica:

Figura 30: Face identificada e isolada

```
# Extrai a face
roi = cinza[34:34 + 128, 79:79 + 128]
cv2_imshow(roi)
```




FONTE: captura de tela realizada pelo autor

Para finalizar essa etapa, redimensionamos e ajustamos a imagem da face:

Figura 31: Ajustes finais da região da face

```
# Ajustes na roi:
from tensorflow.keras.preprocessing.image import img_to_array
roi = roi.astype('float')
roi = cv2.resize(roi, (48, 48))
cv2_imshow(roi)
roi = roi / 255
roi = img_to_array(roi)
roi = np.expand_dims(roi, axis = 0)
```



FONTE: captura de tela realizada pelo autor

### 5.3 UTILIZAÇÃO DA REDE NEURAL TREINADA

Conforme demonstrado na Seção 4.5, o modelo treinado da rede neural foi salvo para uso posterior. Inicialmente é necessário carregá-lo:

Figura 32: Carrega a rede neural treinada

```
# Carrega o modelo treinada da rede neural e a lista de emoções
from tensorflow.keras.models import load_model
caminho_modelo = "/content/modelo_final.h5"
classificador_emocoes = load_model(caminho_modelo, compile = False)
expressoes = ["Raiva", "Nojo", "Medo", "Feliz", "Triste", "Surpreso", "Neutro"]
```

FONTE: captura de tela realizada pelo autor

Com o modelo carregado, basta aplicá-lo à região de interesse para classificarmos a expressão facial:

Figura 33: Probabilidades para cada tipo de expressão facial

```
# Identifica a expressão facial:
preds = classificador_emocoes.predict(roi)[0]
for (i, (emotion, prob)) in enumerate(zip(expressoes, preds)):
    print(i, emotion, str(round(prob * 100, 2)) + '%')
```

```
0 Raiva 4.93%
1 Nojo 0.0%
2 Medo 2.92%
3 Feliz 12.07%
4 Triste 6.85%
5 Surpreso 0.41%
6 Neutro 72.81%
```

FONTE: captura de tela realizada pelo autor

Conforme demonstrado na Figura 33, a rede neural classificou a expressão facial como “Neutra” (com probabilidade de 72,81%) e “Feliz” (com probabilidade de 12,07%). Comparando esses resultados com a imagem original na Figura 25, podemos ver que a classificação foi condizente com uma avaliação humana e subjetiva da expressão facial na imagem original.

## 6 CONCLUSÃO

Este trabalho reproduziu o treinamento, teste, validação e aplicação de uma rede neural para a detecção de emoções humanas através da análise de expressões faciais.

A acurácia obtida pela rede neural foi de 63,97%. Esse resultado é bom? Uma maneira de julgar se nosso modelo obteve uma boa acurácia é compará-lo com a acurácia obtida pelos modelos vencedores da competição Kaggle que utilizou a base de imagens fer2013 (Kaggle, 2013). Os quatro primeiros colocados foram:

- Yichuan Tang: 71,13%
- Yinbo Zhou: 69,27%
- Maxim Milakov: 68,82%
- Radu Ionescu: 67,48%

A arquitetura da rede neural treinada neste trabalho foi relativamente simples e, mesmo assim, ficou 3,51 pontos percentuais abaixo da acurácia do 4º colocado na competição Kaggle, resultado que consideramos muito bom.

O principal problema com o modelo treinado foi a incapacidade de detectar as expressões faciais de “nojo”, o que pode ter ocorrido devido ao reduzido número de imagens desse tipo na base fer2013 (apenas 547 imagens contra mais de 4000 das outras expressões).

Alcançamos os objetivos de obter e processar as imagens faciais, treinar, validar e aplicar uma rede neural para detectar expressões faciais com grau razoavelmente elevado de acurácia (63,97%).

O modelo final da rede neural treinada pode ser utilizado em softwares e aplicativos que necessitem da identificação de emoções humanas através da análise de expressões faciais.



## REFERÊNCIAS BIBLIOGRÁFICAS

- BALTRUSAITIS, T.; IMBRASAITĖ, V.; ROBINSON, P. **CCNF for continuous emotion tracking in music: comparison with CCRF and relative feature representation**. Cambridge: University of Cambridge, 2014. Online: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.650.4627&rep=rep1&type=pdf> [Acesso em maio/2020].
- BENGIO, Y. **Learning Deep Architectures for AI**. *Foundation and Trends in Machine Learning*, 2(1), p. 1-127, 2009.
- BITTENCOURT, G. **Inteligência Artificial: ferramentas e teorias**. Santa Catarina: EDUFSC, 1998.
- BLAZADONAKIS, M. E.; ZERVAKIS, M. **Support Vector Machines and Neural Networks as Marker Selectors in Cancer Gene Analysis**. In: *Intelligent Techniques and Tools for Novel System Architectures*. Studies in Computational Intelligence, vol 109. Springer, Berlin, Heidelberg, 2008.
- BEZERRA, E. **Introdução à Aprendizagem Profunda**. In: *Tópicos em Gerenciamento de Dados e Informações*. SBC, p. 57-86, 2016.
- BRANDÃO, M. L. **As bases biológicas do comportamento: introdução à neurociência**. In: *Revista do Instituto de Medicina Tropical de São Paulo*. 47(3), 2004.
- CONTANT, Sheila; LONA, Liliane M. F.; CALADO, Verônica M. A. **Predição do comportamento térmico de tubos compósitos através de redes neurais**. *Polímeros*, 14(5), p. 295-300, 2004.
- CRUZ, A. A. **Abordagem para reconhecimento de emoção por expressão facial baseada em redes neurais de convolução**. 2019. Dissertação (Mestrado em Informática) – Instituto de Computação, Universidade Federal do Amazonas, 2019.
- DARWIN, C. **The Expression of the Emotions in Man and Animals**. Cambridge: Cambridge University, 2013.
- DRAPER, B. A.; BAEK, K.; BARTLETT, M. S.; BEVERIDGE, J. R. **Recognizing faces with PCA and ICA**. *Computer Vision and Image Understanding*, 91, p. 115-137, 2003.

EKMAN, P. **Universal facial expressions of emotion**. California Mental Health Research Digest, 8(4), p. 151–158, 1970.

Google Inc. **Welcome To Colaboratory**. Online. Disponível na internet no endereço <https://colab.research.google.com/notebooks/intro.ipynb> [acesso em julho/2020].

HAYKIN, S. **Redes Neurais: Princípios e Práticas**. 2. ed. Hamilton: Bookman, 2001.

HEISELE, P.; POGGIO, T. **Face recognition with support vector machines: global versus componente-based approach**. In: *Proceedings of the Eight International Conference on Computer Vision*, p. 688-694, 2001.

JARRETT, K.; KAVUKCUOGLU, K.; RANZATO, M.; LECUN, Y. **What is the best multi-stage architecture for object recognition?** In: *IEEE 12th International Conference on Computer Vision*, Kyoto, p. 2146-2153, 2009.

KAGGLE. **Challenges in Representation Learning: Facial Expression Recognition Challenge**. Online. Kaggle, 2013. Disponível na internet em: <https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/> [acesso em julho/2020].

KARPATHY, A; JOULIN A.; FEI-FEI, L. F. **Deep Fragment Embeddings for Bidirectional Image Sentence Mapping**. In: *Advances in Neural Information Processing Systems*. NIPS Proceedings, 27, 2014.

KINLI F. **Deep Learning Lab: fer2013**. Medium, 2018. Disponível online em: [https://medium.com/@birdortyedi\\_23820/deep-learning-lab-episode-3-fer2013-c38f2e052280](https://medium.com/@birdortyedi_23820/deep-learning-lab-episode-3-fer2013-c38f2e052280) [acesso em março/2020].

KONAR, A; CHAKRABORTY, A. **Emotion Recognition: A Pattern Analysis Approach**. New Jersey: Willey, 2015.

KRIZHEVSKY, A.; SUTSKEVER, I; HINTON, G. E. **ImageNet Classification with Deep Convolutional Neural Networks**. In: *Advances in Neural Information Processing Systems*. NIPS Proceedings, 25, 2012.

KUMAR, S. **Face Identification Using Haar Cascade Classifier**. Online. Medium, 2019. Disponível em: <https://medium.com/analytics-vidhya/haar-cascade-face-identification-aa4b8bc79478> [acesso em julho/2020].

LECUN, Y. **Generalization and Network Design Strategies**. Toronto: University of Toronto, 1989.

LECUN, Y.; BOTTOU, L.; BENGIO, Y; HAFFNER, P. **Gradient-based learning applied to document recognition**. In *Proceedings of the IEEE*, 86(11), p. 2278-2324, 1998.

LIM, J. S. **Two-Dimensional Signal and Image Processing**. Englewood: Prentice-Hall, 1990.

MACHADO, A. **Neuroanatomia Funcional**. São Paulo: Atheneu, 1993.

MATHWORKS Inc. **Convolutional Neural Network: 3 things you need to know**. Disponível online no site da MathWorks: <https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html>. [Acesso em janeiro/2020].

MNIH, V.; et al. **Asynchronous Methods for Deep Reinforcement Learning**. In: *Proceedings of The 33rd International Conference on Machine Learning*, PMLR 48:1928-1937, 2015.

MOORSY, A. **Facial Expression Dataset**. Online. Disponível online na internet em <https://www.kaggle.com/ahmedmoorsy/facial-expression> [acesso em julho/2020].

NEVES, J; OMATU, S; RODRIGUEZ, J. M. C; SANTANA, J. F. P; et al. **Distributed Computing and Artificial Intelligence**. Springer, 2012.

NGIAM, J.; et al. **Multimodal Deep Learning**. In: *Proceeding of the 28th International Conference on Machine Learning*. Velleve, USA, 2011.

PRABHU, R. **Understanding of Convolutional Neural Network (CNN) – Deep Learning**. Online. Medium, 2018. Disponível na internet, no endereço: <https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148>. [Acesso em junho/2020]

RABINER, L. R. **A tutorial on hidden Markov models and selected applications in speech recognition**. In: *Proceedings of the IEEE*, 77(2), p. 257-286, 1989.

RANZATO M., et al. **Large Scale Distributed Deep Networks**. In: *Advances in Neural Information Processing Systems*. NIPS Proceedings 25, 2012.

SAS Inc. **SAS Visual Data Mining and Machine Learning 8.3: Deep Learning Programming Guide**. Online. SAS Inc, 2020. Disponível online em: <https://documentation.sas.com/?docsetId=casdlpg&docsetTarget=p0994iow7i7ogwn1vsk7xv2w4dz1.htm&docsetVersion=8.5&locale=en> [acesso em julho/2020].

SERMANET, P.; LECUN, Y. **Traffic Sign Recognition with Multi-Scale Convolutional Networks**. New York: New York University, 2011. Disponível online em: <http://yann.lecun.com/exdb/publis/pdf/sermanet-ijcnn-11.pdf> [acesso em maio/2020].

SERMANET, P. et al. **OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks**. Cornell University Library. Disponível online no arXiv: <https://arxiv.org/abs/1312.6229>, 2013. [Acesso em março/2020].

SILVER, D.; HUANG, A.; MADDISON, C.; et al. **Mastering the game of Go with deep neural networks and tree search**. *Nature* 529, 484–489, 2016.

STEINER, C. M.; ALBERT, D. **Personalising Learning through Prerequisite Structures Derived from Concept Maps**. In: *Advances in Web Based Learning. Lecture Notes in Computer Science*, Berlin, Springer, v. 4823, 2007.

TAVARES, J. M. R. S. **Algumas Ferramentas para Visão Tridimensional por Computador**. Dissertação (Mestrado em Engenharia Electrotécnica e de Computadores) – Departamento de Engenharia Electrotécnica, Universidade do Porto, 1995a.

TAVARES, J. M. R. S.; JORGE PADILHA, A. **Matching Lines in Image Sequences using Geometric Constraints**. In: 7th Portuguese Conference on Pattern Recognition. Aveiro, Portugal: 1995b.

TAVARES, J. M. R. S. **NiGAVaPS – Outbreeding in Genetic Algorithms**. In: **Symposium on Applied Computing**. Como, Itália, p. 477-482, 2000.

WONG, M. L. D.; NANDI, A. K. **Automatic digital modulation recognition using spectral and statistical features with multi-layer perceptrons**. In: *Proceedings Of The Sixth International Symposium On Signal Processing And Its Applications*. Kuala Lumpur: Malaysia, p. 390-393, 2001.

ZEILER, M. D. **ADADELTA: An Adaptive Learning Rate Method**. Cornell University Library. Disponível online no arXiv: <https://arxiv.org/abs/1212.5701>, 2012. [Acesso em março/2020].

ZHANG, F. et al. **A Simple Multivariate Filter for Estimating Potential Output**. IMF Working Paper, 2015.