UNIVERSIDADE DE CAXIAS DO SUL
ÁREA DO CONHECIMENTO DE CIÊNCIAS EXATAS E ENGENHARIAS

RODRIGO REUSE CHAVES

REDES NEURAIS DEEP LEARNING APLICADAS AO RECONHECIMENTO FACIAL

CAXIAS DO SUL
2018

RODRIGO REUSE CHAVES

REDES NEURAIS DEEP LEARNING APLICADAS AO RECONHECIMENTO FACIAL

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Ciência da Computação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Orientador:
Prof.ª Dra. Carine Geltrudes Webber

CAXIAS DO SUL
2018

RODRIGO REUSE CHAVES

REDES NEURAIS DEEP LEARNING APLICADAS AO RECONHECIMENTO FACIAL

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Ciência da Computação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Orientador:
Prof.ª Dra. Carine Geltrudes Webber

Aprovado em 13/12/2018

Banca Examinadora

_____
Prof. Carine Geltrudes Webber
Universidade de Caxias do Sul - UCS

_____
Prof. Maria de Fatima Webber do Prado Lima
Universidade de Caxias do Sul - UCS

_____
Prof. Andréa Cantarelli Morales
Universidade de Caxias do Sul - UCS

RESUMO

O reconhecimento facial por computadores ainda é uma tarefa desafiadora. Sistemas de redes neurais buscam soluções para que a tarefa de reconhecimento seja cada vez mais uma realidade. Hoje, as redes neurais convolucionais alcançam os resultados mais precisos se comparado a outros modelos de redes neurais. As redes neurais convolucionais são bem adaptadas para a classificação de imagens, porque se utilizam da estrutura espacial da imagem para realizar a análise. Por conta disso, redes neurais convolucionais e algumas variantes próximas são as mais utilizadas no reconhecimento de faces e imagens em geral. Técnicas de aprendizado de máquina chamadas de *deep learning* são aplicadas em conjunto com as redes neurais artificiais na tarefa de reconhecimento. O aprendizado de máquina realiza a busca de padrões nas imagens. O resultado da busca é aquisição de conhecimento, o que possibilita a adaptação necessária da rede neural para realizar o reconhecimento das imagens. A busca de padrões é feita através de treinamentos, que são realizados diversas vezes sobre uma base de dados com imagens de exemplo. Para o reconhecimento facial, o método de aprendizado mais utilizado é o aprendizado de máquina supervisionado. Nesse método a base de dados com imagens de exemplo já é previamente rotulada, indicando qual é a classe de cada imagem. A implementação de redes neurais artificiais e utilização de métodos de *deep learning* normalmente são feitos com o auxilio de *frameworks*. Existem *frameworks open source* voltados para a criação de redes neurais e aprendizado de máquina. Eles agilizam o processo de criação de sistemas de reconhecimento, já que criar um sistema de reconhecimento de imagens apartir do zero é algo complexo e demorado. Os *frameworks* possuem arquiteturas de redes neurais e métodos de aprendizado de máquina já implementados, deixando ao programador a tarefa de parametrização. No experimento apresentado nesse trabalho, foi realizado a identificação facial em tempo real de 10 pessoas. Utilizando 3 arquiteturas de rede neural diferentes, Inception V3, Inception V4 e Mobilenet V1 224. Todas as arquiteturas obtiveram as mesmas imagens de treinamento, 1149 fotos, que variou entre 74 e 160 fotos de cada um dos participantes. Na análise das imagens foi utilizado uma câmera de 13 Mpx, para a captura das imagens, e um equipamento Raspeberry para o processamento dos dados. A arquitetura Inception V4 não pode ser iniciada no equipamento Raspeberry, pois o mesmo não conseguiu carregá-la devido a pouco quantidade de memória RAM, 1 GB, e, portanto, os seus resultados não foram contabilizados. Quanto as demais, Inception V3 e Mobilenet V1 224, obtiveram resultados distintos em termos de acurácia e velocidade de processamento. A Inception V3 obteve uma maior acurácia, enquanto a Mobilenet V1 224 obteve a maior velocidade de processamento.

Palavras-chave: Aprendizado profundo. Aprendizado de máquina. Reconhecimento Facial. Inteligência Artificial. Redes Neurais.

LISTA DE FIGURAS

Figura 1 – Gradiente Descendente 11
Figura 2 – Gráfico de Aprendizado 11
Figura 3 – Neurônio Perceptron 16
Figura 4 – Arquitetura - *Single-Layer Feedforward Networks* 18
Figura 5 – Arquitetura MLP - Multilayer Perceptron 19
Figura 6 – Arquitetura -*Recurrent Networks* 20
Figura 7 – Arquitetura -*Recurrent Networks* com camada oculta 21
Figura 8 – CNN Local Receptive Field 23
Figura 9 – Estrutura de uma CNN 24
Figura 10 – Exemplo-1 do processo de trabalho de uma CNN 25
Figura 11 – Exemplo-2 do processo de trabalho de uma CNN 25
Figura 12 – Exemplo de reconhecimento facial com CNN 27
Figura 13 – Tempo de Treinamento dos Frameworks - Dataset CIFAR10 30
Figura 14 – Tempo de Treinamento dos Frameworks - Dataset MNIST 31
Figura 15 – Comparação de Interesse no GitHub 32
Figura 16 – Comparação de Contribuições no GitHub 33
Figura 17 – Sistema de scores do teste 36
Figura 18 – Infraestrutura 40
Figura 19 – Pré-processamento das imagens de treinamento 41
Figura 20 – Estrutura da Análise 46

LISTA DE TABELAS

Tabela 1 – Exemplo Q-Learning 13
Tabela 2 – Soma ponderada dos neurônios perceptron 16
Tabela 3 – Resultados da pesquisa 28
Tabela 4 – Comparação de Aplicações 34
Tabela 5 – Parâmetros da Rede de Teste MLP 34
Tabela 6 – Parâmetros da Rede de Teste CNN 35
Tabela 7 – Resultados do Teste TensorFlow for Poets 37

SUMÁRIO

1 INTRODUÇÃO **7**
1.1 OBJETIVO GERAL E ESPECíFICOS 8
1.2 ORGANIZAÇÃO DO DOCUMENTO 8

2 APRENDIZADO DE MÁQUINA (*MACHINE LEARNING*) **9**
2.1 APRENDIZADO SUPERVISIONADO (*SUPERVISED LEARNING*) 9
2.1.1 Aprendizado Supervisionado em Redes Neurais Artificiais 10
2.2 APRENDIZADO POR REFORÇO (*REINFORCEMENT LEARNING*) 12
2.3 APRENDIZADO NÃO SUPERVISIONADO (*UNSUPERVISED LEARNING*) . . . 14

3 REDES NEURAIS ARTIFICIAIS **15**
3.1 NEURÔNIOS PERCEPTRON 15
3.2 NEURÔNIOS SIGMÓIDES 17
3.3 ARQUITETURA DE UMA REDE NEURAL ARTIFICIAL 17
3.3.1 *Single-Layer Feedforward Networks* 17
3.3.2 *Multilayer Feedforward Networks* 18
3.3.3 *Recurrent Networks* 20
3.4 REDES NEURAIS PROFUNDAS 21
3.4.1 Redes Neurais Convolucionais (CNN) 22

4 RECONHECIMENTO FACIAL **26**
4.1 CARACTERIZAÇÃO DO PROBLEMA 26
4.2 TRABALHOS RELACIONADOS 27
4.2.1 Reconhecimento Facial com Luz Infravermelha 27
4.2.2 Reconhecimento de Imagens na Biomedicina 28
4.3 COMPARAÇÃO DAS APLICAÇÕES 29
4.3.1 Comparação dos Frameworks 30
4.3.2 Testes Realizados 34

5 IMPLEMENTAÇÃO DO PROJETO **38**
5.1 MÉTODO 38
5.2 APRESENTAÇÃO DO SOFTWARE 38
5.3 MATERIAIS E MODELOS UTILIZADOS 39
5.4 COLETA E PRÉ-PROCESSAMENTO DE IMAGENS 40
5.5 IMPLEMENTAÇÃO DA REDE NEURAL CONVOLUCIONAL 41
5.6 TREINAMENTO DA REDE NEURAL 42
5.7 EXECUÇÃO NOS DISPOSITIVOS FíSICOS 44
5.8 AVALIAÇÃO DOS RESULTADOS 45
5.8.1 Regras de Avaliação 45
5.8.2 Resultados Obtidos 47

6 CONSIDERAÇÕES FINAIS **48**
6.1 SíNTESE DO TRABALHO 48
6.2 TRABALHOS FUTUROS 48

REFERÊNCIAS **50**

1 INTRODUÇÃO

O termo aprendizado de máquina se refere ao conjunto de técnicas para construir sistemas capazes de adquirir conhecimento de forma automática (MONARD; BARANAUSKAS, 2003). Em um sentido amplo, o aprendizado de máquina pode ser categorizado em aprendizado supervisionado ou aprendizado não supervisionado (HAYKIN, 2008). A aquisição de conhecimento ocorre com a busca de padrões sobre uma base de dados, através de uma etapa conhecida como treinamento. O treinamento pode ocorrer diversas vezes sobre os mesmos dados até alcançar a melhor precisão. Métodos específicos são utilizados dependendo de cada situação, pois não há um método único que apresente o melhor resultado, seja velocidade de treinamento, seja velocidade de convergência, seja precisão. Cada caso deve ser analisado separadamente para que se possa buscar a melhor solução (MONARD; BARANAUSKAS, 2003).

As redes neurais artificiais foram motivadas desde seu início pela forma que o cérebro humano computa as informações. O cérebro é um computador altamente complexo, não linear e paralelo, que tem a capacidade de organizar seus constituintes estruturais, conhecidos como neurônios, de modo à realizar certos cálculos como: reconhecimento de padrões, percepção e controle motor, muitas vezes mais rápido que o melhor computador existente hoje. De forma geral, uma rede neural é uma máquina projetada para modelar o caminho em que o cérebro realiza para uma determinada tarefa (HAYKIN, 2008). As redes neurais artificiais são constituídas por neurônios artificias que processam e trocam informações entre si. O primeiro neurônio artificial foi desenvolvido por Frank Rosenblatt. Esse neurônio, que foi chamado de *perceptron*, possui a capacidade de receber informações - números reais -, processá-las, e após enviar um resultado booleano para a próxima camada de neurônios conectada a ele. Atualmente, o modelo de neurônio mais utilizado é o sigmoidal, que possui um funcionamento similar, com a diferença de produzir uma saída sigmoidal ao invés de uma saída booleana (NIELSEN, 2015).

Após o ano de 2006, técnicas de aprendizado profundo, chamadas de *deep learning*, e modelagem de redes neurais profundas conseguiram melhorar o desempenho em muitos problemas importantes de visão computacional (NIELSEN, 2015). Áreas como publicidade *online*, análise no mercado de ações, websites adaptativos, jogos de estratégia, detecção de fraude virtual, classificação de imagens, reconhecimento de fala, processamento de linguagem natural, utilizam aplicações como redes neurais profundas e *deep learning* em suas soluções.

O reconhecimento facial é um das tarefas computacionais que utilizam as redes neurais profundas e as técnicas de *deep learning* em suas soluções. O reconhecimento facial é um sistema biométrico de funcionamento similar aos já conhecidos sistemas de reconhecimento por impressão digital e reconhecimento pela retina. Isso porque, assim como a digital e a retina, cada pessoa possui um rosto único. A grande vantagem da utilização biométrica pelo reconhecimento facial é fato dela ser uma técnica natural, não intrusiva e de fácil utilização (LI; JAIN, 2005). Atualmente, essa técnica vem sendo cada vez mais utilizada comercialmente e militarmente, tornando-a um assunto cada vez mais popular. Entretanto, para ser confiável, os

sistemas de reconhecimento precisam trabalhar com uma alta taxa de precisão, algo que ainda não foi alcançado (ÇARıKçı; öZEN, 2011).

1.1 OBJETIVO GERAL E ESPECÍFICOS

O objetivo principal deste trabalho é avaliar um modelo de rede neural com aprendizado profundo para a tarefa de identificação de pessoas por meio de reconhecimento de imagens da face.

A fim de atingir o objetivo proposto, foram definidos os seguintes objetivos específicos:

- Identificar métodos computacionais utilizando modelos de redes neurais para reconhecimento de imagens da face e selecionar o método mais apropriado .

- Implementar, treinar e testar métodos e algoritmos de aprendizado profundo para fins de reconhecimento de imagens.

- Apresentar os resultados obtidos.

1.2 ORGANIZAÇÃO DO DOCUMENTO

O trabalho está organizado em 6 capítulos, sendo esse o primeiro, onde é passado uma breve introdução do assunto e os objetivos do trabalho.

No Capítulo 2, o aprendizado de máquina é abordado. Nele os métodos de aprendizado de máquina são classificados por categorias. A Seção 2.1 fala sobre os métodos de aprendizado de máquina supervisionado. Na Seção 2.2 o aprendizado por reforço e, finalizando, a Seção 2.3 fala sobre o aprendizado não supervisionado.

No Capítulo 3, as redes neurais artificiais são o assunto, iniciando pelo funcionamento dos neurônios perceptron, na Seção 3.1 e, após, os neurônios sigmoidais na Seção 3.2. A Seção 3.3 aborda o funcionamento dos neurônios em uma arquitetura de rede neural. A última Seção do capítulo, a Seção 3.4, entra no assunto das redes neurais profundas, onde é falado sobre as redes neurais convolucionais.

No Capítulo 4, o reconhecimento facial é discutido. A Seção 4.1 caracteriza a problema. A Seção 4.2 tem trabalhos relacionados a área de reconhecimento facial. E a Seção 4.3, mostra uma comparação dos principais *frameworks* utilizados na implementação de reconhecimento de imagens.

No Capítulo 5, é apresentado a forma em que o software foi desenvolvido.

Finalizando, o Capítulo 6 tem as considerações finais do trabalho.

2 APRENDIZADO DE MÁQUINA (*MACHINE LEARNING*)

Algoritmos de aprendizado de máquina buscam determinar padrões para resolver um determinado problema, adaptando o algoritmo para chegar o mais próximo possível da solução (REVISTABW, 2015). Pode-se dizer que o aprendizado de máquina é o conjunto de técnicas utilizadas para a aquisição de conhecimento pelo sistema (PEREZ, 2017). Algoritmos de aprendizado de máquina podem ser categorizados em: algoritmos de aprendizado supervisionado e algoritmos de aprendizado não supervisionado (PEREZ, 2017). Tem-se ainda, segundo (HAYKIN, 2008), os algoritmos de aprendizado por reforço como uma subcategoria do aprendizado não supervisionado.

2.1 APRENDIZADO SUPERVISIONADO (*SUPERVISED LEARNING*)

O aprendizado supervisionado é utilizado sobre conjuntos de dados já previamente rotulados. Ao associar esses conjuntos de dados ao algoritmo, o aprendizado de máquina deverá ter a capacidade de se adaptar para alcançar o resultado. A adaptação ocorre gradativamente conforme os dados vão sendo analisados. A análise acontece diversas vezes até que se chegue no resultado esperado. A sequência de análises sobre o mesmo conjunto de dados é uma etapa conhecida como treinamento. O treinamento é dividido em épocas, uma época começa no momento da análise do primeiro dado do conjunto, e termina no fim da análise do último dado do conjunto. A cada época de treinamento a aplicação vai aprendendo e se modificando, produzindo resultados cada vez mais próximos dos resultados esperados (REVISTABW, 2017). A aprendizagem supervisionada, portanto, utiliza dois vetores de dados, um com os dados de entrada e outro com os dados de saída desejados. Após cada época de treinamento, o vetor resultante da análise do treinamento é comparado com o vetor de saída desejado. A partir dessa comparação, o aprendizado de máquina altera os parâmetros da aplicação com objetivo de chegar o mais próximo possível da resposta (PEREZ, 2017). Essa técnica de aprendizado de máquina é a mais comum para o treinamento de redes neurais artificiais e árvores de decisão, além de ser a principal técnica utilizada para casos de classificação (REVISTABW, 2017).

Casos de classificação são casos em que os dados são usados para prever uma categoria. Por exemplo, uma aplicação que é utilizada para o reconhecimento de imagens em que o resultado de saída deve ser uma categoria como gato ou cachorro. Mais especificamente, nesses casos em que há apenas duas categorias possíveis, o caso de classificação é chamado de classificação binomial ou de classificação de duas classes. Nos demais casos, quando há mais de duas categorias possíveis, o caso de classificação é chamado classificação multi-classe (DOCS, 2017).

Esse documento tem como objetivo a modelagem de uma aplicação de classificação que é especificamente o reconhecimento facial. O aprendizado de máquina supervisionado é o principal método utilizado para esse fim. Portanto, apesar de existirem métodos de aprendizado

de máquina supervisionado para outras aplicações, aqui é abordado os métodos de aprendizado supervisionado que, em conjunto com redes neurais artificiais, são empregados em sistemas de reconhecimento facial.

2.1.1 Aprendizado Supervisionado em Redes Neurais Artificiais

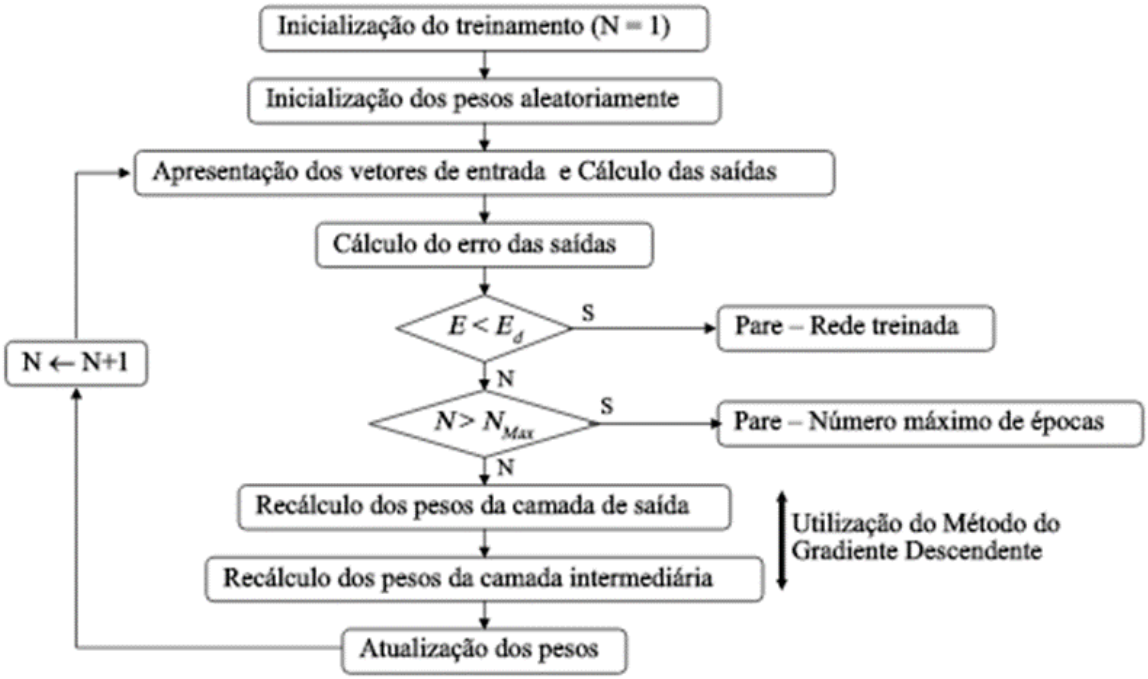
No aprendizado de máquina supervisionado que tem como objetivo o treinamento de redes neurais artificiais (RNAs), o algoritmo de treinamento realiza a comparação do vetor de saída do treinamento com o vetor de saída desejado e, após, realiza a adaptação da aplicação, como já dito anteriormente. A adaptação da aplicação, quando se trata de redes neurais, acontece com o ajuste dos pesos e vieses dos neurônios da rede (REVISTABW, 2015). No Capítulo 3, onde o assunto abordado são as redes neurais, é mostrado como os pesos e vieses dos neurônios funcionam. De momento, não é necessário a compreensão do funcionamento da rede neural para entender o funcionamento do método de aprendizado *backpropagation*, que será o tema da sequência dessa seção.

O *backpropagation* é um dos métodos de aprendizado supervisionado mais utilizados para o treinamento de redes neurais artificiais. Ele propaga o erro entre o vetor de saída do treinamento e o vetor de saída desejado, inciando a propagação na camada de saída, passando camada por camada até chegar na camada de entrada. A correção de erros é feita com um método conhecido por descida gradiente (*gradient descendent*). A descida gradiente reduz o erro gerado, auxiliando na rapidez e na convergência, tornando o aprendizado mais eficiente e confiável (PEREZ, 2017). A Figura 1 ilustra o funcionamento do método Gradiente Descendente.

A convergência do vetor de saída do treinamento da rede neural geralmente ocorre numa função de superajuste. Essa função sofre variâncias, aumentando gradativamente a precisão com o passar das épocas, mas com algumas quedas de precisão no meio do caminho (DONG; ZHOU, 2008). Esse é um comportamento normal no processo de aprendizado. Parâmetros de aprendizado como quantidade de dados de treinamento, velocidade de aprendizado e quantidade de épocas de treinamento influenciam nesse comportamento. Porém, mesmo com os parâmetros mais otimizados as variâncias irão acontecer. A Figura 2 apresenta um gráfico do *framework* TensorFlow que mostra as variâncias durante o processo de aprendizado de uma rede neural.

A velocidade de aprendizado, parâmetro conhecido como passo de aprendizagem, informa para o algoritmo como as mudança nos pesos e vieses da rede neural devem ocorrer. Um passo grande promove mudanças bruscas nos valores dos neurônios da rede, enquanto que um passo pequeno promove mudanças mais sutis, o que influência diretamente na velocidade de convergência. Valores altos para esse parâmetro tendem a fazer a rede convergir mais rápido, mas aumentam as variâncias, podendo tornar a rede menos precisa até mesmo no final do

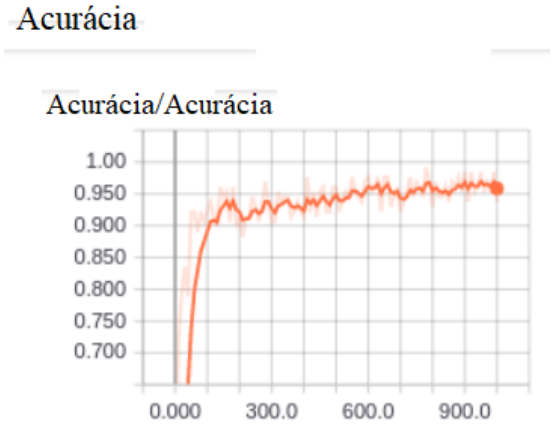
Figura 1 – Gradiente Descendente



Fonte: (PEREZ, 2017)

treinamento. Por outro lado, valores baixos fazem a rede convergir mais lentamente, com menos variâncias, podendo necessitar de muito tempo até chegar na precisão desejada (NIELSEN, 2015).

Figura 2 – Gráfico de Aprendizado



Fonte: (FACURE, 2017)

O excesso de épocas de treinamento (*overfitting*), ocorre quando a rede neural é treinada muitas vezes com a mesma base de dados de treinamento. O excesso pode piorar a precisão, ocasionando erros de classificação no momento em que a rede analisa dados que não apareceram no treinamento (MONARD; BARANAUSKAS, 2003). Em outras palavras, pode-se dizer que a rede acaba "decorando"e não aprendendo, perdendo a capacidade de aplicar os padrões aprendidos a novos exemplos.

Por outro lado, a baixa quantidade de épocas de treinamento, ou a pouca quantidade

de exemplos nos dados de treinamento, ocasionam o chamado *underfitting*. O *underfitting*, assim como o *overfitting*, causa problemas na precisão no momento da análise (MONARD; BARANAUSKAS, 2003).

Não existe uma fórmula para saber qual o valor ideal, nem para a quantidade de épocas de treinamento, nem para a quantidade de exemplos na base de treinamento, nem para o passo de aprendizagem. O valor ideal vai depender das variáveis de cada caso. Portanto, alcançar esse valor ideal não é uma tarefa fácil, e ele somente irá aparecer com testes (NIELSEN, 2015).

2.2 APRENDIZADO POR REFORÇO (*REINFORCEMENT LEARNING*)

A aprendizagem por reforço é realizada através da interação contínua com o ambiente, tendo como objetivo minimizar um índice escalar de desempenho (HAYKIN, 2008). Nesse modo de aprendizagem não há um vetor de dados de treinamento já previamente rotulado que a rede possa usar para aprender, mas a aplicação ainda precisa aprender como agir nas suas tarefas. Na ausência de dados de treinamento a rede aprende com suas próprias experiência, por meio de tentativa e erro (MAINI; SABRI, 2017).

O *Q-learning*, e algoritmos relacionados, é um método em que um agente tenta aprender a política ótima a partir de sua história de interação com o ambiente. Usando a ideia de recompensas e penalidades para as ações tomadas, o algoritmo consegue entender quando as ações realizadas são boas ou ruins. Uma história de interação do agente com o ambiente pode ser dada da seguinte forma: (s0,a0,r1,s1,a1,r2,s2,a2,r3,s3 ...). Essa história significa que o agente estava no estado s0 e fez a ação a0, o que resultou no recebimento da recompensa r1. Já no estado s1 ele fez uma nova ação, denominada como a1, recebendo a recompensa r2, chegando ao estado s2. Após, o agente realizou a ação a2, recebeu a recompensa r3 e acabou no estado s3, e assim por diante. Essas experiências serão utilizadas pelo agente para decidir o que fazer em novas situações (POOLE; MACKWORTH, 2017). Quando o agente se depara com uma nova situação, ele se utiliza das experiências obtidas e compara diferentes sequências de ações, comparando as recompensas que cada uma lhe dará. A maneira mais comum de se fazer a comparação é converter cada recompensa em um valor de maneira que quanto melhor é a recompensa mais alto o seu valor. Após, os valores de cada recompensa da sequência são somados, com isso o agente terá o conhecimento das recompensas imediatas e das recompensas futuras, podendo escolher se suas ações serão baseadas nas recompensas imediatas ou nas recompensas futuras (POOLE; MACKWORTH, 2010). A forma como o agente vai agir dependerá da aplicação. As vezes o caminho que proporciona recompensas imediatas não leva a maior recompensa futura. Se o algoritmo pensasse sempre da mesma forma, procurando sempre pela recompensa mais imediata, pode ser que ele nunca consiga chegar ao melhor resultado possível.

Um exemplo dado por (DAS, 2017) explicou o funcionamento do algoritmo da seguinte forma: tem-se um agente no início de um labirinto, que é definido como estado inicial. As possíveis ações do agente são ou andar para à esquerda, ou andar para à direita, ou andar

para frente, ou andar para baixo ou permanecer parado. Cada ação o leva para uma célula da grade da Tabela 1, que ilustra a situação descrita. No fim do labirinto há um tesouro, que é o objetivo, e no meio dele há alguns poços de cobras que devem ser evitados. O agente não sabe onde o tesouro está, tão pouco onde estão os poços com cobras. Então, para dar a ideia de poço de cobra e tesouro, é dado alguma recompensa para cada ação tomada. Quando o agente cai em um dos poços com cobras, ele recebe uma recompensa de -10 pontos, o que na verdade seria uma penalidade, e quando ele encontra o tesouro recebe uma recompensa de +10 pontos. Para que o agente ande pelo labirinto a procura do caminho mais curto, é dado uma penalidade de -1 ponto para cada passo. Dessa forma, o agente aprende que as cobras são prejudiciais e que o tesouro é benéfico e que ele deve tentar pegar o caminho mais curto possível.

Tabela 1 – Exemplo Q-Learning

Início			
-	Cobras	Cobras	
-	Cobras		
-	-	-	Tesouro

Fonte: (DAS, 2017)

É importante considerar quais as informações que estarão disponíveis para o agente no momento em que ele vai tomar a decisão. Para isso há duas variações, o processo de decisão de Markov totalmente observável e o processo de decisão de Markov parcialmente observável. No primeiro, o agente consegue observar o estado atual para decidir sua ação; no segundo, o agente tem acesso apenas ao histórico de observações e ações anteriores. Quanto a qual caminho o agente deve seguir, deve-se levar em conta se o agente vai preferir a recompensa total, a recompensa média ou a recompensa com desconto (POOLE; MACKWORTH, 2010).

A recompensa total é a soma de todas as recompensas parciais. Essa ação funciona bem quando há a garantia de que a soma das recompensas é finita. Caso contrário, o agente não terá como comparar qual sequência de recompensas é a melhor. A recompensa média é o valor da média das recompensas sobre um determinado período de tempo. Nesse caso, o importante é em que situação o agente ficará no final de um tempo. A recompensa com desconto usa o critério de que as recompensas futuras valem menos do que as atuais, o que fará o agente escolher sempre a recompensa mais imediata (POOLE; MACKWORTH, 2010).

O aprendizado por reforço é utilizado em problemas complexos em que não parece haver uma solução óbvia ou facilmente programável. Um exemplo são jogos de computadores, situação onde há várias variáveis a serem analisadas para a tomada de uma decisão. Outro exemplo, são problemas de controle como agendamento de elevador. Nesses casos, os agentes podem ser deixados para aprender em um ambiente simulado e, eventualmente, obterão boas políticas de controle. Algumas vantagens de se usar o aprendizado por reforço para problemas de controle é que um agente pode ser treinado com facilidade para se adaptar às mudanças do

ambiente (EDEN; KNITTEL; VAN UFFELEN, 2002).

2.3 APRENDIZADO NÃO SUPERVISIONADO (*UNSUPERVISED LEARNING*)

O aprendizado não supervisionado, como o próprio nome sugere, e da mesma forma que o aprendizado por reforço, não há um vetor de dados rotulado. Na falta desse vetor de dados rotulado, o algoritmo faz uma medida da qualidade de representação que ele deve aprender. Para isso, os parâmetros livres da aplicação são otimizados em relação a essa medida. Uma vez que a aplicação esteja sintonizada com as regularidades estatísticas dos dados de entrada, ela desenvolve a capacidade de formar representações internas para codificar os recursos da entrada e, assim, criar novas classes automaticamente (HAYKIN, 2008).

Os algoritmos de *clustering*, que pode ser traduzido para o português como agrupamento, são algoritmos de aprendizado não supervisionado. Um caso de uso bem interessante dessa técnica é o sistema de armazenamento em *cluster* do fornecedor de dados da Acxiom, o Personicx. Esse serviço segmenta as residências dos Estados Unidos em 70 *clusters* distintos em 21 grupos, que são usados por anunciantes ao segmentar anúncios do Facebook, exibir propagandas, campanhas de mala direta entre outros. O objetivo principal dessa técnica é criar grupos de dados conforme suas semelhanças. (MAINI; SABRI, 2017).

Um exemplo de algoritmo de *clustering* é o *K-means clustering*. Dado um conjunto de dados o *K-means clustering* tem por objetivo realizar a separação dos dados em K grupos. O valor de K é variável, um K alto indica uma quantidade maior de grupos, como consequência, os grupos serão menores e com mais granularidade. Por outro lado, um K menor indica uma menor quantidade de grupos, acarretando em grupos maiores e com menos granularidade. A saída do algoritmo gera um conjunto de rótulos que são atribuídos aos dados. Os grupos são definidos a partir da criação de um dado central chamado de *centroide*. Os *centroides* são criados para auxiliar na busca de similaridades em cada grupo (MAINI; SABRI, 2017).

Segundo (MAINI; SABRI, 2017), os passos seguidos pelo algoritmo *K-means clustering* são os seguintes:

1. Definição dos *centroides* dos K grupos, que podem ser definidos aleatoriamente, ou com a ajuda de algoritmos sofisticados, que os definem com base em regras específicas, tornando a convergência dos dados mais rápida.

2. Procura dos *centroides* mais próximos atualizando as atribuições do *cluster*. Os pontos similares ao *centroide* de um grupo K, são atribuídos a esse grupo.

3. Os *centroides* de cada *cluster* são movidos para o centro, que é definido como a posição média de todos os pontos contidos no *cluster*.

4. As operações 2 e 3 se repetem até que o *centroide* pare de se mover, ou se mova muito pouco. Quando isso acontecer significa que o algoritmo convergiu.

3 REDES NEURAIS ARTIFICIAIS

As redes neurais artificiais (RNAs) são um modelo computacional que tem o seu funcionamento inspirado no cérebro humano (HAYKIN, 2008). Elas possuem, portanto, a capacidade de aprender por meio de exemplos. As redes neurais têm contribuído muito no desenvolvimento de sistemas de reconhecimento e classificação de padrões (XIAO et al., 1999). O primeiro neurônio artificial, base do funcionamento das redes neurais, foi o *perceptron*, gerador de saídas rígidas, desenvolvido nas décadas de cinquenta e sessenta pelo cientista Frank Rosenblatt. Atualmente o modelo de neurônio artificial mais utilizado é o *sigmoid neuron* que possui um funcionamento similar ao *perceptron*, com a diferença de gerar saídas sigmoidais (NIELSEN, 2015).

Os neurônios biológicos tem como sua principal função disseminar sinais elétricos, sendo a rede formada por eles a responsável pela capacidade de processamento do cérebro humano (RUSSELL; NORVIG, 2003). Os neurônios são divididos em corpo celular, dendritos e axônio, cada um exercendo uma função, porém complementares entre si. Os dendritos recebem as informações na forma de impulsos elétricos, que são oriundos de suas ligações com outros neurônios. As informações são enviadas até o corpo celular, onde ocorre o processamento da informação e após esse processamento a informação é enviada, por meio do axônio, para outro neurônio conectado a ele, repetindo o processo.

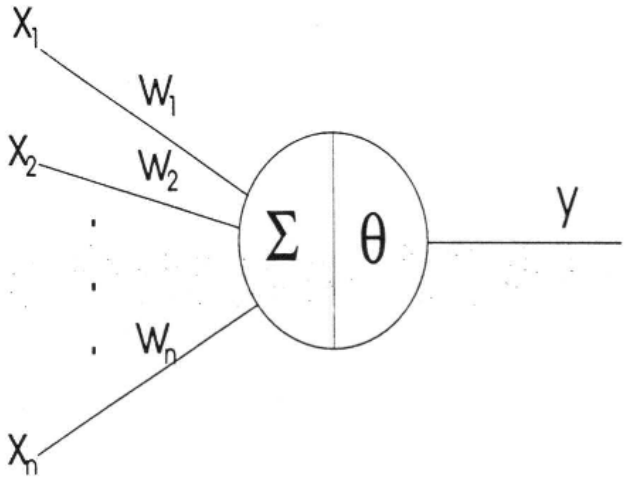
As conexões entre o axônio de um neurônio com o dendrito de outro são chamadas de sinapses (BRAGA; CARVALHO; LUDERMIR, 2007). As sinapses cerebrais são tão poderosas que nem mesmo o melhor computador da atualidade é capaz de realizar com precisão e agilidade algumas tarefas realizadas pelo cérebro, como por exemplo, controle motor, reconhecimento de padrões e percepção (XIMENES, 2011).

3.1 NEURÔNIOS PERCEPTRON

Os *perceptrons* funcionam como um modelo simplificado do neurônio biológico, possuindo diversas entradas binárias de dados nas camadas internas, que seriam os equivalentes aos axônios dos neurônios biológicos, e apenas uma saída binária, que seria o equivalente ao dendríto, como pode ser visto na Figura 3. Entretanto, os neurônios da primeira camada, denominada camada de entrada, possuem apenas uma entrada (NIELSEN, 2015), a 3 mostra um neurônio de uma camada interna, em que há varias conexões de entrada.

Cada uma das conexões entre os neurônios possui um peso (um número real) que expressa o quão importante essa conexão é para o processamento da informação, quanto maior o peso mais importante é a conexão. As entradas possuem um valor binário, um valor 1, indica que a entrada deve ser ativada, um valor 0 indica que a entrada não deve ser ativada. A Figura 3 apresenta um neurônio artificial em que os pesos estão sendo são expressos por W1, W2 ... Wn,

Figura 3 – Neurônio Perceptron



Fonte: (BRAGA; CARVALHO; LUDERMIR, 2007)

enquanto as entradas estão sendo expressas por X1, X2 ... Xn. O processamento da informação ocorre no centro do neurônio. Nos neurônios artificiais a informação é processada da seguinte forma: os pesos das conexões são colocados em um vetor de dados e os valores das entradas são colocados em outro vetor de dados. Com esses dois vetores, o neurônio realiza o produto escalar entre vetor das entradas e o vetor coluna dos pesos (W * X). Essa operação também pode ser expressa como a soma dos resultados das multiplicações de cada entrada com o seu respectivo peso ((W1.X1)+(W2.X2)+ ... +(Wn.Xn)), ou, em outras palavras, a soma ponderada dos pesos e entradas. Ambas as formas de processar a informação chegam ao mesmo resultado, a única diferença é a forma de calcular. É importante observar que mesmo uma conexão tendo um peso alto, ela não será contabilizada na soma ponderada se a sua entrada possuir o valor binário 0, já que a multiplicação de qualquer número por 0 é 0. O resultado do cálculo é verificado para saber se ele ficou abaixo ou acima do limiar setado no neurônio. O limiar (*threshold value*), ou ainda mais popularmente conhecido como viés (*bias*), denotado na Figura 3 como θ, é utilizado para saber qual será o resultado binário da saída, ou seja, se será 0, caso o resultado da soma ponderada seja inferior ou igual ao limiar, ou se será 1, caso o resultado da soma ponderada seja superior ao limiar. Os resultados 0 e 1 produzidos pela saída do neurônio indicam se esse estará desativado ou ativado, ou ainda, indicar um não ou um sim. Em termos algébricos, pode-se descrever a operação realizada pelo neurônio na Tabela 2, em que j é a conexão do neurônio, w é o valor do peso dessa conexão e x é o valor binário da entrada dessa conexão (NIELSEN, 2015).

Tabela 2 – Soma ponderada dos neurônios perceptron

Saída igual a 0, se $\sum_j w_j x_j \leq$ Limiar
Saída igual a 1, se $\sum_j w_j x_j >$ Limiar

Fonte: (NIELSEN, 2015)

O grande problema dos perceptrons é o aprendizado. Pequenas mudanças nos pesos ou no viés podem fazer com que a saída dele seja completamente invertida, pelo fato de poderem produzir apenas saídas binárias. Esse fato dificulta possíveis mudanças que talvez sejam necessárias para que a rede neural se comporte da maneira desejada (NIELSEN, 2015).

3.2 NEURÔNIOS SIGMÓIDES

Os neurônios sigmóides (*sigmoid neuron*) contornam o problema de aprendizado dos perceptrons, pois seu modelo permite que as mudanças nos pesos e vieses sejam mais sutis para o resultado de saída. Assim, fica mais fácil realizar o aprendizado de máquina.

Seu funcionamento, como já dito anteriormente, é similar ao perceptron. Ele possui a mesma ideia de entrada e pesos. Porém, o valor das entradas e da saída não são mais binárias, e sim qualquer valor real entre 0 e 1. De forma algébrica, a saída de um neurônio sigmóide em que , x é a entrada, w é o peso e b é o viés, pode ser expressa conforme (NIELSEN, 2015):

$$\frac{1}{1 + exp(-\sum_j w_j x_j - b)} \tag{3.1}$$

Como a saída agora é um número real, a interpretação da saída mudará dependendo da situação. Em um caso da rede neural estar sendo utilizada para uma análise de imagem, a saída pode representar a intensidade média dos píxeis da imagem de entrada. Por outro lado, se o objetivo for definir o resultado como sim ou não, pode-se convencionar que um resulta abaixo do viés seja interpretado como não, e um resultado acima do viés seja interpretado como sim (NIELSEN, 2015).

3.3 ARQUITETURA DE UMA REDE NEURAL ARTIFICIAL

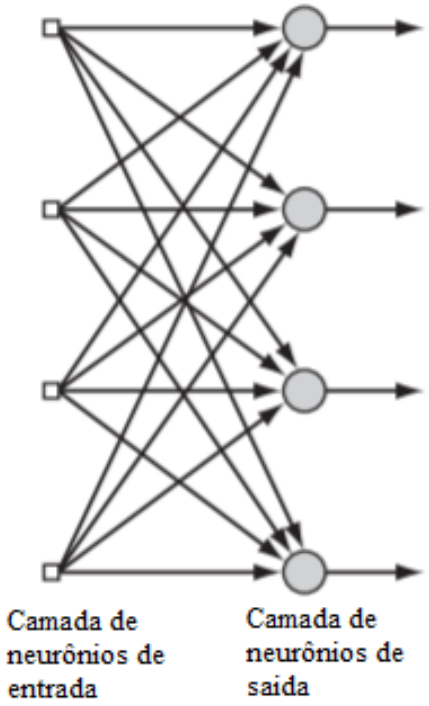
Uma vez que é sabido o funcionamento de um neurônio artificial, é necessário entender como eles funcionam dentro de uma arquitetura de rede neural. Fundamentalmente, as arquiteturas de rede neural podem ser divididas em três tipos: *Single-Layer Feedforward Networks*, *Multilayer Feedforward Networks*, *Recurrent Networks* (HAYKIN, 2008). As três se utilizam de uma organização em camadas de neurônios, em que as saídas dos neurônios de uma camada são conectadas nas entradas dos neurônios da camada seguinte (BRAGA; CARVALHO; LUDERMIR, 2007).

3.3.1 *Single-Layer Feedforward Networks*

As *Single-Layer Feedforward Networks* são a forma mais simples de redes em camadas, tendo apenas duas, a camada de entrada e a camada de saída. A camada de entrada projeta sua saída diretamente para a camada de saída, e a informação segue apenas esse fluxo, ou seja,

a informação é sempre enviada para frente. Vale a pena ressaltar que mesmo havendo duas camadas, as redes neurais *Single-Layer Feedforward Networks* são chamadas de redes neurais de camada única. Isso porque não é contabilizada a camada de entrada, é apenas contabilizado a camada de saída. Não se contabiliza a camada de entrada, pois nenhuma computação é executada nos seus neurônios. A Figura 4 apresenta uma rede *Single-Layer Feedforward Networks* com quatro neurônios de entrada e quatro neurônios de saída, assim como as conexões entre os neurônios da camada de entrada e a camada de saída.

Figura 4 – Arquitetura - *Single-Layer Feedforward Networks*



Fonte: (HAYKIN, 2008)

3.3.2 *Multilayer Feedforward Networks*

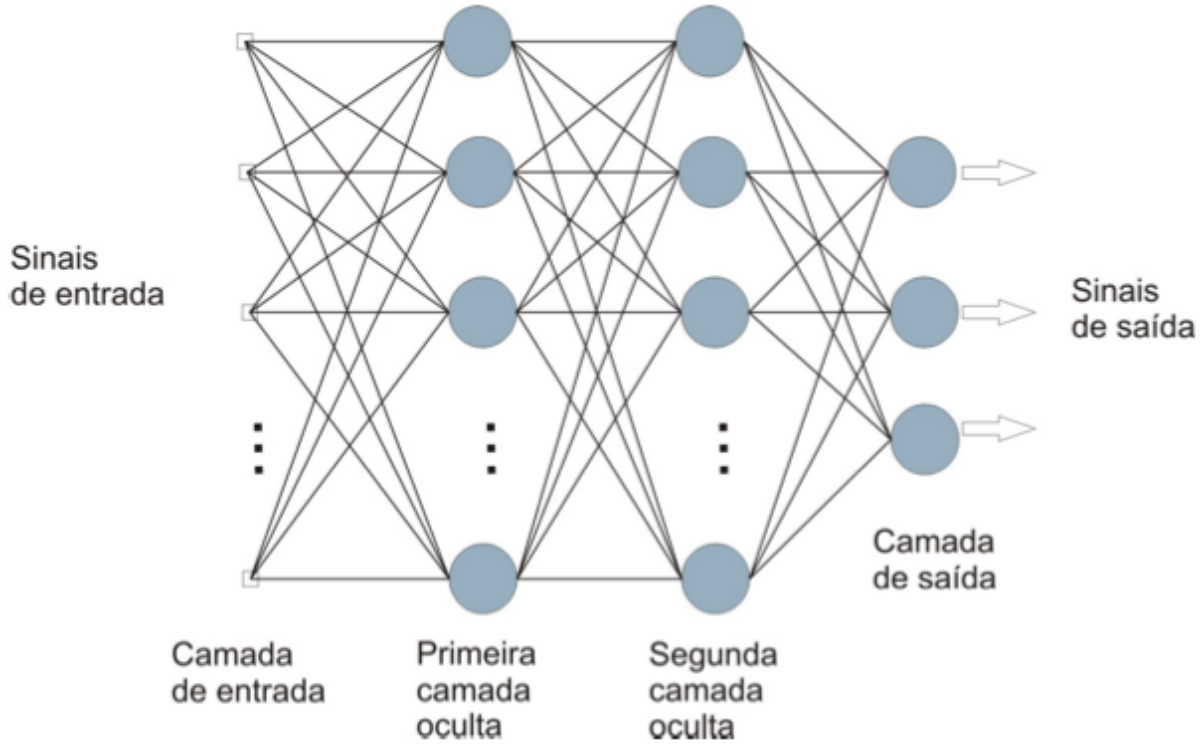
As *Multilayer Feedforward Networks* se diferem das *Single-Layer Feedforward Networks* pelo fato de terem mais de uma camada. Nesse modelo de rede neural, as saídas dos neurônios de uma camada são conectadas nas entradas dos neurônios da camada seguinte, que, por sua vez, conectam suas saídas nos neurônios da próxima camada e assim por diante. Dessa forma, pode-se dizer que há uma camada de entrada e uma camada de saída, assim como nas *Single-Layer Feedforward Networks*, porém com a adição de outras camadas entre elas. Essas camadas adicionais são chamadas de camadas ocultas. O termo "oculto" aparece pelo fato de que esta parte da rede neural não é vista diretamente da entrada ou da saída da rede. Ao adicionar uma ou mais camadas ocultas a rede está habilitada para extrair estatísticas de ordem superior da sua entrada (HAYKIN, 2008). Tendo, portanto, a função de transformar os dados de entrada em uma representação tratável para a camada de saída. A adição das camadas ocultas

pode tornar um problema que não seria linearmente separável em um problema linearmente separável (HAYKIN, 2001). Problemas que são linearmente separáveis são mais fáceis de se analisar, o que aumenta as chances de acerto. A camada de saída, ao receber as informações da última camada oculta, fornece a resposta geral da rede (HAYKIN, 2008).

Outro fator que as *Multilayer Feedforward Networks* tem em comum com as *Single-Layer Feedforward Networks*, é o fato de que os pesos e as entradas são calculados pelos neurônios e os resultados são sempre passados adiante, por essa razão, o modelo também possui a denominação *feedforward* em seu nome (HAYKIN, 2001).

Esse modelo de rede é também conhecido como *multilayer perceptron* ou (MLP) (BRAGA; CARVALHO; LUDERMIR, 2007) que, apesar da denominação ser *multilayer perceptron*, a implementação de redes com essa arquitetura é realizada com neurônios sigmóides (NIELSEN, 2015), como pode ser visto na Figura 5, que ilustra a arquitetura MLP.

Figura 5 – Arquitetura MLP - Multilayer Perceptron



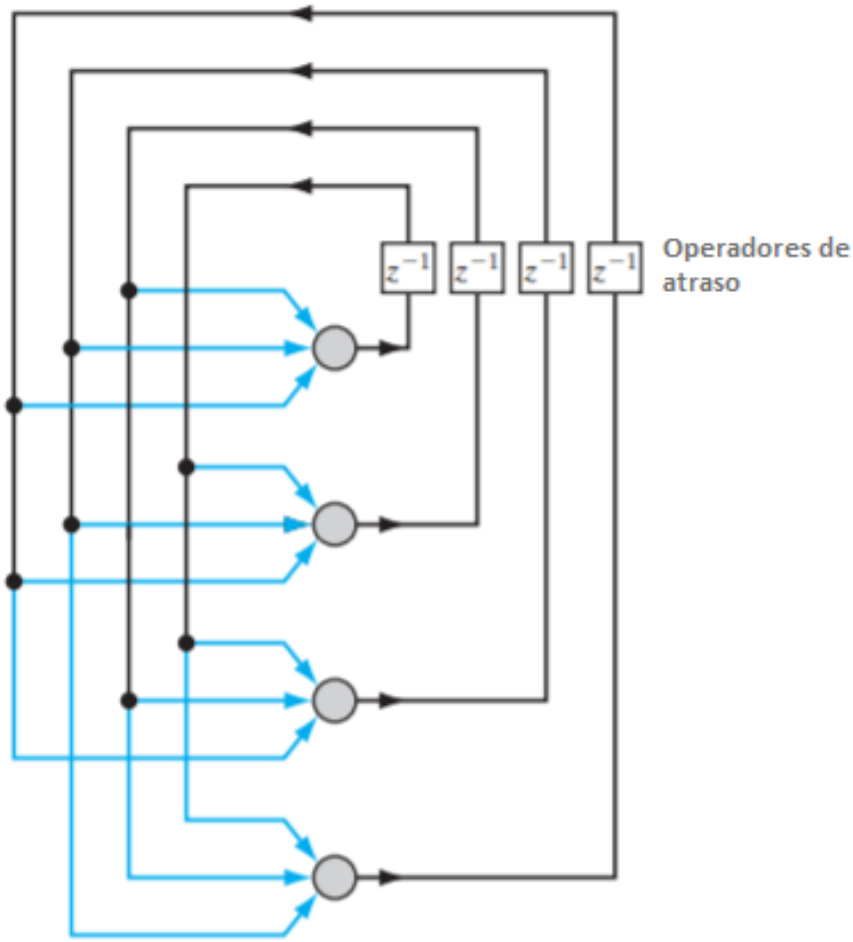
Fonte: (HAYKIN, 2001)

A rede neural ilustrada na Figura 5 é chamada de rede totalmente conectada (*fully connected*). Ela é uma *multilayer perceptron*, mas é também uma rede totalmente conectada. A denominação *fully connected* apenas indica que cada nó de cada camada da rede está ligado a todos os outros nós adjacentes da camada seguinte. Se, no entanto, não houvesse conexão entre todos os nós, a rede seria chamada de rede parcialmente conectada (*partially connected*) (HAYKIN, 2008).

3.3.3 *Recurrent Networks*

As redes neurais recorrentes (*Recurrent Networks*) distinguem-se das demais pelo fato de sempre haver ao menos um ciclo de retorno da informação (*feedback*). Isso significa que não é sempre que a informação é passada adiante, ocorrem *loops* de retorno em que o sinal de saída do neurônio volta para as entradas dos outros neurônios, ou, até mesmo, para a sua própria entrada, no caso de um *auto-feedback* (HAYKIN, 2008). De acordo com (NIELSEN, 2015), redes neurais recorrentes estão mais próximas de simular o funcionamento do cérebro humano do que as outras arquiteturas de rede neural. Isso porque no cérebro humano também ocorrem *loops* de retorno. Apesar das redes neurais recorrentes simularem melhor o funcionamento do cérebro humano, elas são menos utilizadas, pelo menos até o momento, por obterem resultados menos satisfatórios. Na Figura 6 pode ser visto o retorno da informação em uma estrutura de rede neural recorrente sem camadas ocultas.

Figura 6 – Arquitetura -*Recurrent Networks*

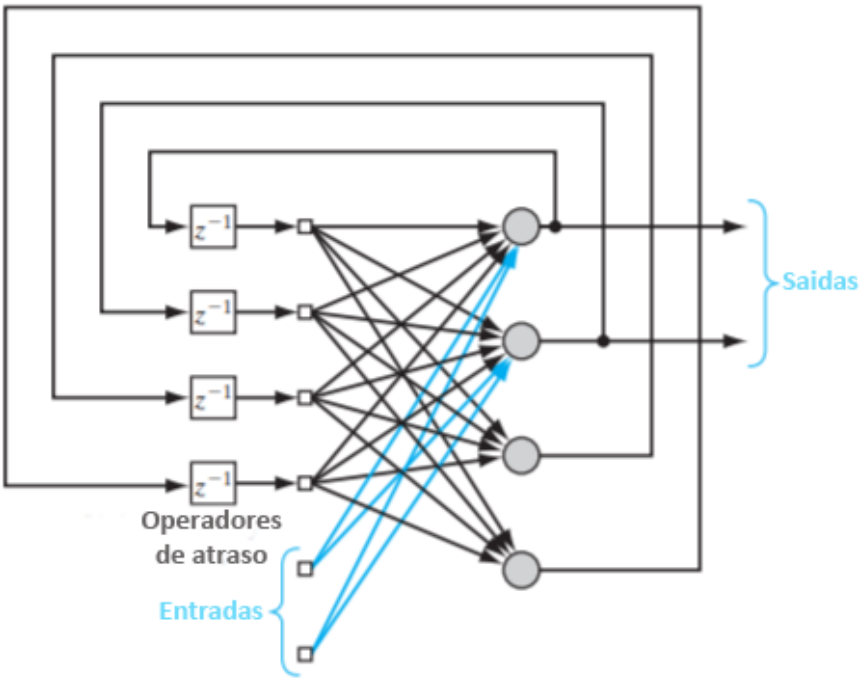


Fonte: (HAYKIN, 2008)

Na Figura 7, pode-se ver uma rede neural recorrente com camada oculta. Os *loops* de realimentação possuem um impacto profundo na capacidade de aprendizagem da rede e no

seu desempenho. Além disso, os laços de feedback envolvem o uso de ramos específicos compostos de elementos de atraso de tempo unitário (denotado por z-1), que resultam em um comportamento dinâmico não-linear, assumindo que a rede neural contém unidades não-lineares (HAYKIN, 2008).

Figura 7 – Arquitetura -*Recurrent Networks* com camada oculta



Fonte: (HAYKIN, 2008)

3.4 REDES NEURAIS PROFUNDAS

Redes neurais profundas (*DNN - deep neural networks*) tornaram-se uma ferramenta indispensável para uma ampla gama de aplicações como classificação de imagens, reconhecimento de fala e processamento de linguagem natural. Essas técnicas alcançaram uma precisão preditiva extremamente alta, em muitos casos a par com o desempenho humano. Sua arquitetura não é muito diferente das MLPs. Uma rede neural profunda é uma coleção de neurônios organizados em uma sequência de múltiplas camadas, onde os neurônios recebem como entrada as ativações neurais da camada anterior, e realizam um cálculo simples. Os neurônios da rede implementam em conjunto um mapeamento não linear complexo da entrada para a saída (MONTAVON; SAMEK; MüLLER, 2017).

As primeiras camadas das DNNs respondem a perguntas simples e muito específicas sobre os dados de entrada. Posteriormente, as próximas camadas montam uma hierarquia de conceitos cada vez mais complexa e abstrata. Os algoritmos de treinamento para tais redes são baseados nos métodos de descida gradiente (*gradient descendent*) e no método da retropropagação *backpropagation*, uma vez que esses métodos nas suas formas padrão são muito lentos

para relizar o treinamento nessa arquitetura de rede neural (NIELSEN, 2015). Na construção de algoritmos mais precisos geralmente a taxa de transferência das máquinas é o fator limitante. Por conta disso, o potencial das DNNs foram desbloqueado somente a partir da evolução das unidades de processamento gráfico (GPUs) e a criação de uma grande escala de base de dados (CAMPOS et al., 2017).

3.4.1 Redes Neurais Convolucionais (CNN)

Redes Neurais Convolucionais são bem adaptadas para a classificação de imagens, porque, ao contrário das arquiteturas mostradas anteriormente, as CNNs se utilizam da estrutura espacial da imagem. Hoje, redes convolucionais profundas ou algumas variantes próximas são usadas na maioria das redes neurais para reconhecimento de imagens (NIELSEN, 2015). O nome Rede Neural Convolucional foi dado pelo fato da rede aplicar uma operação matemática denominada convolução, que é um tipo especializado de operação linear. Essas redes se utilizam da convolução no lugar da multiplicação geral de matrizes em pelo menos uma de suas camadas (COSTA, 2016).

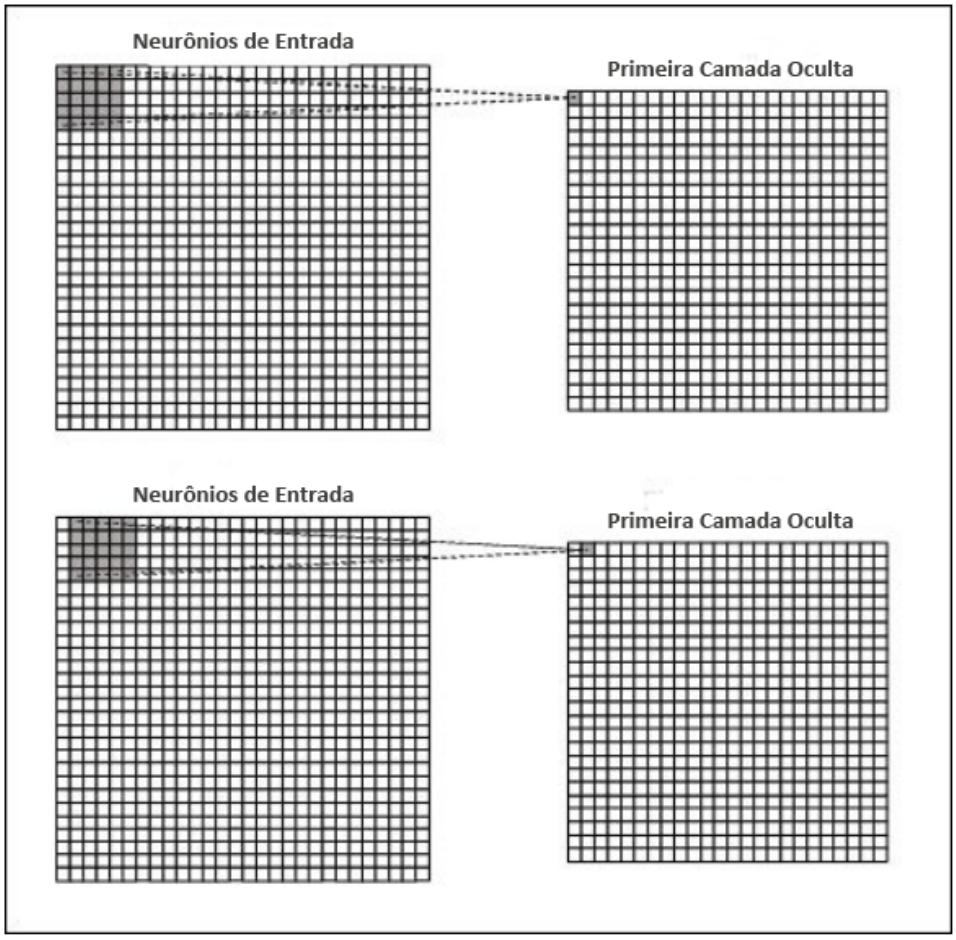
Nas redes neurais convolucionais, os neurônios da primeira camada da rede recebem os píxeis da imagem de entrada, um píxel por neurônio, assim como acontece nas MLPs. Porém esses neurônios não são conectados diretamente na camada seguinte, ao invés disso, é criado um campo receptivo local (*local receptive field*), que nada mais é que o agrupamento de uma região dos neurônios da primeira camada. O agrupamento pode possuir diversos tamanhos, por exemplo, 5x5 neurônios, o que corresponde a 25 neurônios da camada de entrada. Cada um dos neurônios agrupados é conectado a um neurônio da segunda camada utilizando os mesmos pesos e viéses, que são conhecidos como pesos e viéses compartilhados, ou kernel, ou filtro. Em termos algébricos, para o neurônio oculto, em que σ é a função de ativação neural, b é o valor compartilhado do viés, $w_{l,m}$ é a matriz dos pesos compartilhados do agrupamento, $a_{j+l,k+m}$ é a ativação de entrada na posição $a_{x,y}$, a saída produzida é dada pela equação (NIELSEN, 2015):

$$\sigma(b + \sum_{l=0}^{4} \sum_{m=0}^{4} w_{l,m} a_{j+l,k+m}) \tag{3.2}$$

O processo de agrupamento é repetido em toda imagem, deslizando a janela de agrupamento para cima, para baixo e para os lados. O valor que define o quanto esse agrupamento se move é chamado de passo (*stride length*) (MARTíNEZ, 2017). O processo de agrupamento promove um aumento na precisão de análise. Isso porque píxeis que estão próximos uns dos outros são mais propensos a estarem fortemente relacionados se comparado com píxeis que estão longe um do outro. Com o uso dessa técnica, o conjunto de píxeis que estão dentro do mesmo agrupamento são analisados juntos e com o mesmo peso, ao contrário do que acontecia nas arquiteturas citadas anteriormente. Nas redes neurais não convolucionais os píxeis ficavam ligados diretamente aos neurônios da camada seguinte, passando a informação sem

realizar qualquer tipo de agrupamento. Isso ocasionava uma maior carga computacional, tornando a rede menos precisa. Portanto, as redes neurais convolucionais tornam o processamento de imagens computacionalmente gerenciável através da filtragem das conexões por proximidade (REIS; RAVINDRA, 2017). A Figura 8 ilustra o processo descrito acima com o valor do passo valendo 1.

Figura 8 – CNN Local Receptive Field



Fonte: (NIELSEN, 2015)

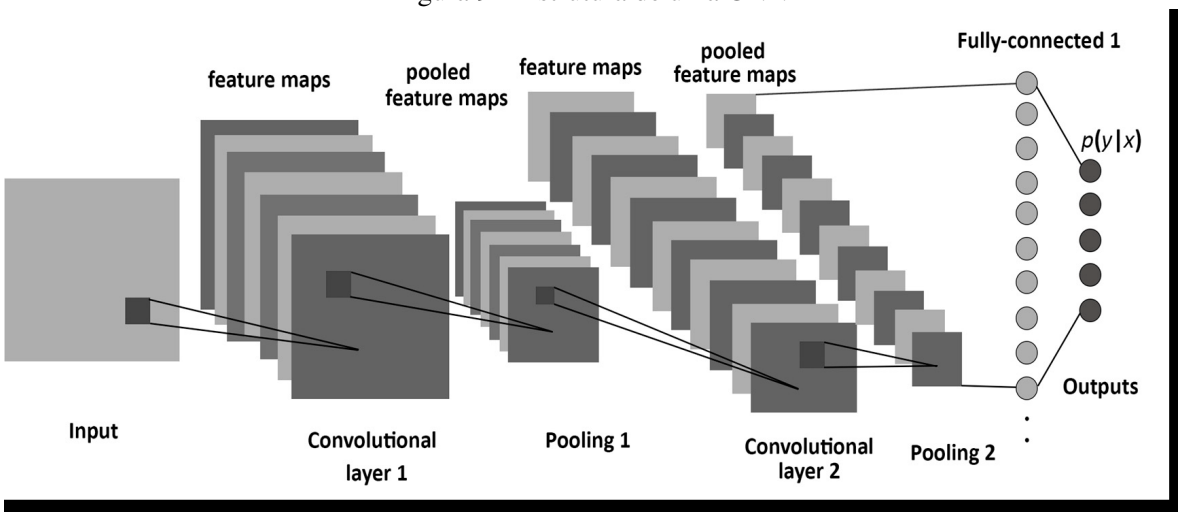
A segunda camada, ou seja, a camada de neurônios que recebe os campos receptivos locais é chamada de mapas de recursos (*feature maps*). O tamanho do mapa de recursos é baseado no tamanho do agrupamento e no valor do passo. Portanto, se a imagem de entrada possui um tamanho (HxH), o agrupamento um tamanho (FxF) com um passo valendo (S), o mapa de recursos possuirá um tamanho (WxW), pode ser definido por (MARTíNEZ, 2017):

$$W = [\frac{H-F}{S}] + 1 \tag{3.3}$$

Na prática, redes neurais convolucionais utilizam mais de um mapa de recursos, cada um com pesos e viéses compartilhados diferente. Isso é realizado com o objetivo de detectar diferentes tipos de padrões nas imagens. Uma das primeiras redes convolucionais, a LeNet-5, utilizou 6 mapas de recursos, cada um associado a um campo receptivo local de 5x5. Essa rede foi implementada para reconhecer dígitos do *dataset* MNIST (NIELSEN, 2015).

A camada seguinte de uma rede convolucional é a camada de *pooling* ou *down-sampling*. Sua função é reduzir a resolução dos mapas de recursos que estão ligados a ela. O *pooling* produz invariância a uma pequena transformação e/ou distorção, dividindo as entradas em regiões disjuntas com um tamanho de (RxR) para produzir uma saída de cada região. Em seguida, a camada de *pooling* pode enviar as informações para a camada mais externa, a camada *softmax classifier*, ou ainda, enviar para uma nova camada convolucional com um novo mapa de recursos, dependendo da arquitetura em que a rede neural foi implementada. Chegando na camada mais externa, na camada *softmax classifier*, a estimativa da probabilidade de cada rótulo de classe sobre K classes é processada. Em outras palavras, na camada *softmax classifier* é onde a rede neural calcula a probabilidade da imagem se encaixar em um dos padrões aprendidos, indicando qual é a sua classificação. Na Figura 9, pode-se observar a estrutura de uma CNN (MARTíNEZ, 2017).

Figura 9 – Estrutura de uma CNN

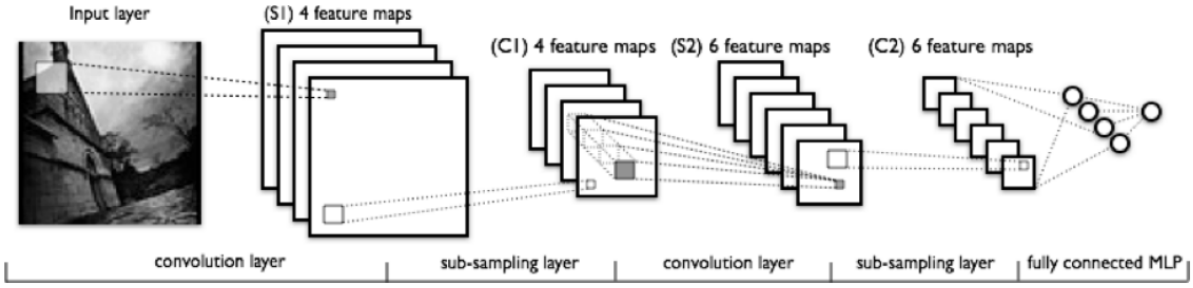


Fonte: (MARTíNEZ, 2017)

Na Figura 10, o objetivo é apresentar o processo de uma rede neural convolucional na análise de uma imagem. Nesse exemplo, o agrupamento está sendo passado por cima da imagem - ilustrado pelo retângulo branco - e, em seguida, esse agrupamento está enviando as informações para os mapas de recursos. Na camada seguinte, segunda camada, é onde os mapas de recursos estão dispostos em pilha e estão nomeados na imagem como (SI), cada um dos mapas de recursos é associado a um filtro diferente. Posteriormente, na terceira camada - camada (CI) - está a camada de *pooling*, onde os mapas de recursos são condensados. Em (S2) há um novo grupo de mapas de recursos, que foram gerados por uma novo processo de agrupamento sobre a imagem condensada. A camada (C2) é uma nova camada de *pooling*, que

realiza uma nova condensação da imagem. Finalizando, a camada totalmente conectada designa a saída da análise da rede (REIS; RAVINDRA, 2017).

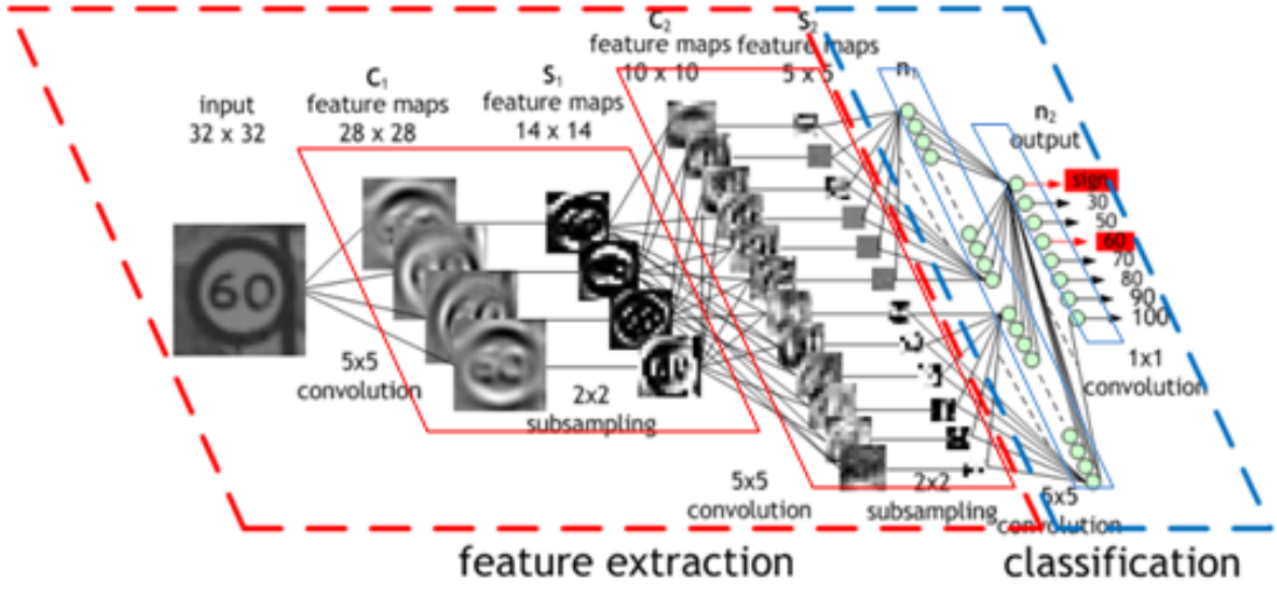
Figura 10 – Exemplo-1 do processo de trabalho de uma CNN



Fonte: (REIS; RAVINDRA, 2017)

A Figura 11 é um novo exemplo do processo de análise de imagem de uma rede neural convolucional. Nesse novo exemplo é apresentado as mudanças que a imagem sofre na passagem de uma camada para a outra. A estrutura da rede nessa abordagem é bem similar a do exemplo anterior, com apenas alterações na quantidade de *feature maps*. Portanto, apesar da pequena mudança na arquitetura da rede, o processo de análise segue o mesmo, assim como a função de cada camada.

Figura 11 – Exemplo-2 do processo de trabalho de uma CNN



Fonte: (COSTA, 2016)

4 RECONHECIMENTO FACIAL

O cérebro humano faz o reconhecimento de imagens de forma fácil e rápida, dando a impressão que essa é uma tarefa. De fato, isso não verdade, essa falsa impressão só ocorre porque o cérebro humano é extremamente bom em reconhecer imagens, o que já não é uma tarefa simples para um computador. Entretanto, o aprendizado de máquina realizou diversos avanços na área de reconhecimento, principalmente no modelo de rede neural convolucional profunda, alcançando com essa técnica resultados até mesmo superiores ao cérebro humano em algumas áreas (BRAIN, 2018a).

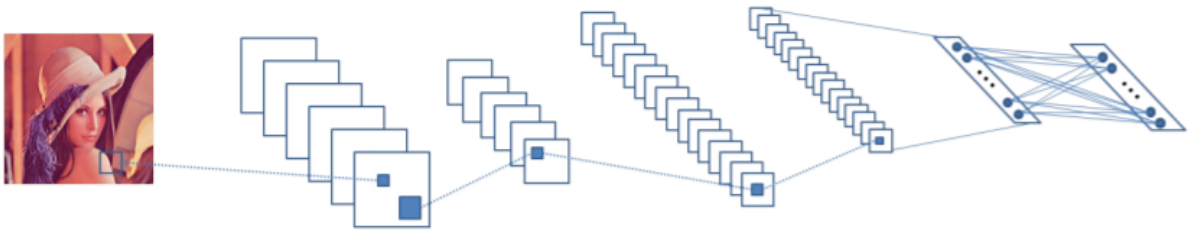
O reconhecimento facial vem sendo estudado desde a década de 60, década em que Woodrow Wilson Bledsoe desenvolveu o primeiro sistema para classificar faces. Apesar da limitação tecnológica da época e da necessidade de se incluir as coordenadas manualmente dos elementos do rosto como nariz, boca, orelhas, olhos, cabelo, esse sistema foi o primeiro passo para provar que o reconhecimento facial era algo possível. Entretanto, foi somente em 1991 que o reconhecimento automático de rostos dentro de uma imagem foi possível, quando Turk e Pentland expandiram o algoritmo Eigenface, algoritmo esse desenvolvido em 1988 por Sirovich e Kirby. Assim como Woodrow Wilson Bledsoe, Turk e Pentland também barraram nas limitações tecnológicas da época, o que os impediram de realizar um avanço ainda maior. Contudo, seu trabalho teve grande importância ao comprovar a viabilidade do reconhecimento facial automático (WEST, 2017). Ainda no início da década de 90 as redes convolucionais apresentaram um bom desempenho nas tarefas de classificação de manuscritos e detecção de rosto (ZEILER; FERGUS, 2014). Desde 2012 quando o vencedor da competição *ImageNet Competition*, Alex Krizhevsky, teve sucesso no reconhecimento de objetos com sua rede neural convolucional chamada de *AlexNet*, houve um estímulo maior nos estudos desse tipo de arquitetura. Apartir de 2014, a qualidade das arquiteturas melhorou significativamente, a VGGNet e a GoogleNet renderam desempenho similarmente altos no desafio de classificação *Large Scale Visual Recognition Challenge 2014* (SZEGEDY et al., 2015a).

Para realizar o reconhecimento de imagens, as redes neurais convolucionais são as mais utilizadas atualmente, como já dito no capítulo anterior, e com o reconhecimento facial não é diferente (NIELSEN, 2015). As redes neurais convolucionais realizam o mesmo processo descrito no capítulo anterior para o reconhecimento de faces. A Figura 12 mostra um diagrama de uma (CNN) realizando o reconhecimento de uma face.

4.1 CARACTERIZAÇÃO DO PROBLEMA

A tarefa de reconhecimento facial, apesar dos diversos avanços, ainda enfrenta alguns desafios. Pode-se dizer que os principais são que os rostos não são estáticos, há variações na

Figura 12 – Exemplo de reconhecimento facial com CNN



Fonte: (BEZERRA, 2018)

expressão, diferentes intensidades de iluminação e diferentes ângulos em que a imagem pode ser capturada (GUO; WU; XU, 2017).

4.2 TRABALHOS RELACIONADOS

Nessa seção são apresentados trabalhos relacionados a redes neurais profundas para o reconhecimento de imagens. O primeiro, apresentado por (GUO; WU; XU, 2017) no artigo *Face recognition using both visible light image and near-infrared image and a deep network*, possui uma grande relevância para a área de reconhecimento facial por se tratar de um método totalmente novo que aumenta consideravelmente a acurácia da rede neural convolucional. O método se baseia no uso de luz infravermelha e luz natural para a coleta de dados de treinamento.

O segundo, apresentado por (GIBSON et al., 2018) no artigo *NiftyNet: a deep-learning platform for medical imaging*, é voltado a área de reconhecimento de imagens na biomedicina, utilizando a plataforma NiftyNet. O NiftyNet é uma plataforma *open source* que utiliza o software TensorFlow para a implementação da rede neural e métodos de aprendizagem de máquina. Isso mostra que o TensorFlow é uma ferramenta muito poderosa e que pode ser utilizada comercialmente em redes neurais de grande porte, indo ao encontro do trabalho proposto nesse documento.

4.2.1 Reconhecimento Facial com Luz Infravermelha

No trabalho apresentado por (GUO; WU; XU, 2017), foi proposto uma rede neural convolucional profunda utilizando não somente dados de treinamento coletados sobre luz visível, como também dados coletados utilizando câmera de infra-vermelho. Segundo ele imagens em infravermelho são menos sensíveis a mudanças de iluminação, que é um dos fatores que mais dificultam a detecção de faces. Em contra partida, imagens de luz visíveis revelam detalhes importantes que não podem ser ignorados. Portanto, a proposta foi utilizar os dois tipos de imagens para realizar a classificação.

Inicialmente, foi usado dados de faces de luz visível de domínio público para o treinar o modelo. Após, a rede foi retreinada com imagens de infra-vermelho. Com isso, o modelo final foi usado aplicando uma estratégia que une a pontuação da classificação dos dois tipos de imagens em uma pontuação final, que é a pontuação utilizada para definir a classificação.

Os testes foram feitos utilizando duas bases de dados, a Sun Win Face e a HIT Lab2. A primeira conta com 4.000 imagens faciais, sendo 2.000 em luz visível e outras 2.000 com luz infravermelha. Cada pessoa foi fotografada 20 vezes com luz visível, sendo 10 com luz em condições normais e as outras 10 fotos em condições de luz diferenciadas, não ideais. Nas imagens de infravermelho o processo foi o mesmo, porém com uma câmera de infravermelho. Portanto, a câmera de infravermelho capturou 20 fotos, sendo 10 com luz em condições normais e as outras 10 fotos em condições de luz diferenciadas. As fotos também se apresentam com diferentes expressões faciais entre outras mudanças. A segunda base tem 2.000 fotos de 50 voluntários, e as imagens foram coletadas em luz natural, luz natural mais luz à esquerda, luz natural mais luz à direita, luz natural mais luz à esquerda e à direita, além de diferentes expressões. O modelo convolucional VGGNet foi utilizado para analisar as mesmas bases, para que os seus resultados pudessem ser comparados com os resultados do modelo proposto. A pesquisa apresenta na Tabela 3 os resultados do modelo com a pontuação final, como a pontuação sem a fusão de pontuações e do modelo VGGNet, nas base de dados HIT Lab2 e na base de dados Sun Win.

Tabela 3 – Resultados da pesquisa

Acurácia no HIT LAB2

	Alteração fraca de luminosidade	Alteração alta de luminosidade
VGG NET	**98.74**	**89.80**
Score sem fusão	**97.96**	**95.13**
Score com fusão	**99.56**	**95.31**

Acurácia no Sun Win

	Alteração fraca de luminosidade	Alteração alta de luminosidade
VGG NET	**99.26**	**80.34**
Score sem fusão	**98.14**	**91.62**
Score com fusão	**99.89**	**93.98**

Fonte: (GUO; WU; XU, 2017)

4.2.2 Reconhecimento de Imagens na Biomedicina

Este trabalho apresentado por (GIBSON et al., 2018), fala sobre a plataforma NiftyNet, utilizada para o aprendizado profundo em imagens médicas. A infra-estrutura NiftyNet fornece um canal modular de aprendizado profundo para uma variedade de aplicativos de imagens médicas, incluindo aplicativos de segmentação, regressão, geração de imagens e aprendizado de representação, além de um banco de dados de redes pré-treinadas para aplicações e ferramentas específicas para facilitar a adaptação da pesquisa de aprendizado profundo a novas aplicações clínicas. A NiftyNet, implementada sobre a estrutura do TensorFlow, suporta recursos como a visualização TensorBoard, ferramenta do TensorFlow para visualização de imagens 2D, 3D

e gráficos. Três aplicações são apresentadas: segmentação de múltiplos órgãos abdominais de tomografia computadorizada, regressão de imagem para predizer mapas de atenuação de tomografia computadorizada a partir de imagens de ressonância magnética cerebral e geração de imagens de ultrassom simuladas para poses anatômicas especificadas. O TensorFlow fornece todas as ferramentas para a construção da rede neural, assim como aplicação de aprendizado eficiente com os recursos de hardware disponíveis. Entretanto, não fornece nenhuma funcionalidade específica para processamento de imagens médicas, ou interfaces de alto nível para médicos comuns realizarem as tarefas de análise da imagem. Nesse ponto, o NiftyNet é quem fornece a otimização necessária.

As classes do aplicativo NiftyNet encapsulam pipelines de análise padrão para diferentes aplicações de análise, ligando quatro componentes: um leitor (*Reader*) para carregar dados de arquivos, um modelo (*Sampler*) para gerar amostras apropriadas para processamento, uma *Network* para processar as entradas e um manipulador de saída (*output handler*), compreendendo a perda (*Loss*), o otimizador (*Optimizer*) durante o treinamento, além de um agregador (*Aggregator*) durante a inferência e avaliação.

O NiftyNet foi utilizado pelos autores porque atualmente suporta: segmentação de imagens, regressão de imagens, representação de modelos de imagens (através de aplicações de codificação automática), geração de imagens (via codificador automático e gerador de conflitos contraditórios redes (GANs)), além de ser projetado de forma modular para suportar a adição de novos tipos de aplicativos, encapsulando fluxos de trabalho de aplicativos típicos em classes de aplicativos.

A arquitetura de rede neural utilizada foi a *DenseVNet*, disponível no NiftyNet, usando uma estrutura com *downsampling*, *upsampling* e *skip connections*. O *downsampling* é uma sequência de blocos de convoluções em que as entradas são características concatenadas de todos os blocos de convolução precedentes. O *upsampling* e *skip connections* são as convoluções da rede. A perda foi implementada externamente com um arquivo de configuração.

Os resultados foram analisados com base em análises de imagens realizadas antes do NiftyNet. A implementação anterior também foi desenvolvida com o TensorFlow para o aprendizado profundo e códigos MATLAB personalizados por terceiros. A implementação pré-NiftyNet não era propícia para distribuir o código ou a rede treinada e faltava visualizações para monitorar o desempenho da segmentação durante o treinamento. Problemas que não ocorrem no NiftyNet.

4.3 COMPARAÇÃO DAS APLICAÇÕES

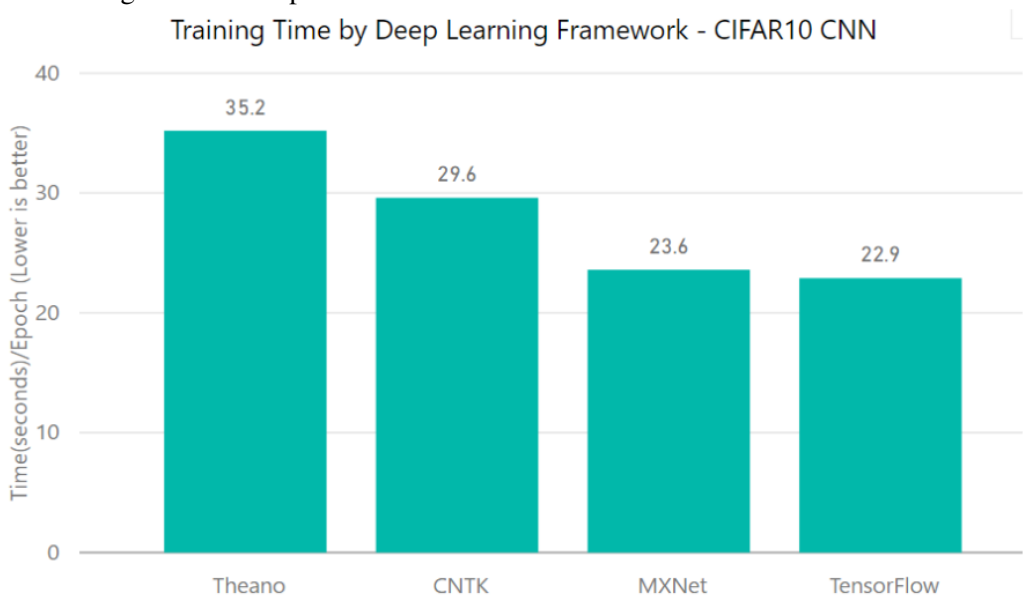
Existem diversos *frameworks* disponíveis para a implementação de redes neurais e utilização de métodos de aprendizado de máquina. Dentre os principais estão o Theano, TensorFlow, Torch, Caffe, MXNet, Neon e CNTK. Essa seção tem como objetivo analisá-los, para que se tenha conhecimento das vantagens de desvantagens de cada um.

4.3.1 Comparação dos Frameworks

Um estudo sobre velocidade de treinamento realizado por (BHATIA, 2017) testou os *frameworks* TensorFlow - na versão 1.3.0-, MXNet - na versão 1.2.2 -, Theano - na versão 0.9.0 -, utilizando como *front end* o software Keras - na versão 2.0.8. Esse estudo foi utilizado como base inicial para a análise das aplicações porque foi o trabalho de comparação que utilizou versões dos *frameworks* bem recentes, apesar de existirem hoje versões mais atuais. O Keras foi utilizado como *front end* porque permite ao desenvolvedor utilizar várias estruturas de aprendizado profundo sem a necessidade de alterar nenhum código. Em setembro de 2017, data em que o estudo foi realizado, as versões dos *frameworks* citados acima eram as versões estáveis mais recentes disponíveis. Os testes foram realizados utilizando arquitetura de rede neural convolucional, realizando analises nas imagens dos *datasets* CIFAR10 e MNIST, classificando 10 classes de imagens em cada um deles. O aprendizado de máquina foi realizado em um máquina virtual Azure NC6 VM, utilizando a GPU Nvidia Tesla K80, configurada com o driver Nvidia-CUDA 8.0.61.

O primeiro teste do autor foi realizado sobre o *dataset* CIFAR10, que revelou que o TensorFlow, em termos de velocidade de treinamento por época, é superior aos demais, tendo apenas uma pequena vantagem sobre o segundo colocado, o MXNet. A Figura 13 demonstra o resultado do teste, sendo possível observar o tempo em segundos para cada época de treinamento em cada um dos *frameworks*. Vale lembrar que, como se trata de tempo de treinamento, o *framework* com a menor barra no gráfico é considerado melhor, uma vez que isso indica uma maior velocidade.

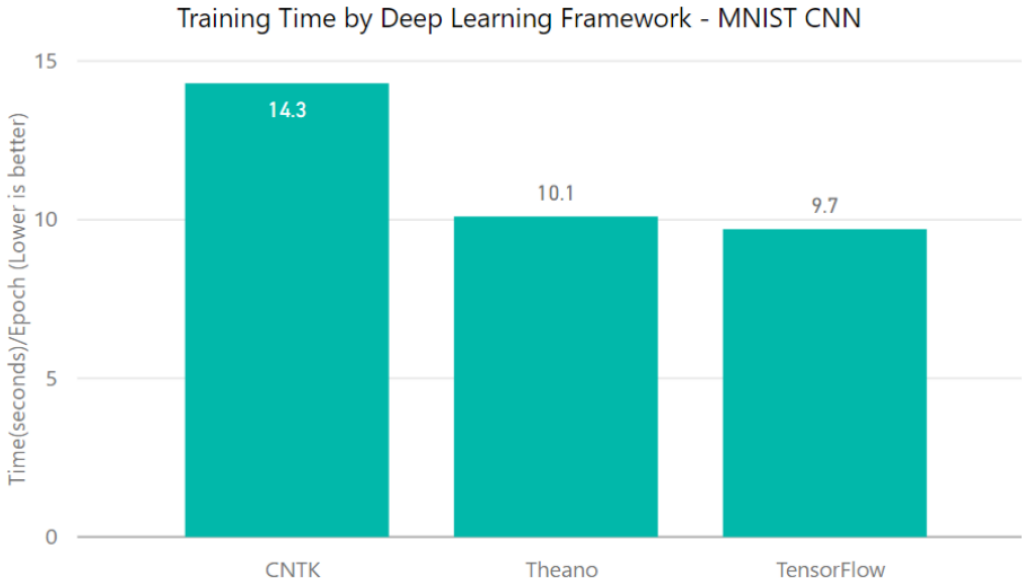
Figura 13 – Tempo de Treinamento dos Frameworks - Dataset CIFAR10



Fonte: (BHATIA, 2017)

segundo teste realizado sobre o *dataset* MNIST, o *framework* MXNet não foi testado, pois, segundo o autor, seriam necessárias alterações significativas no código da rede neural devido a falta de suporte do MXNet para as versões mais recentes do Keras, pelo menos no período em que o estudo foi realizado. Entretanto, as demais aplicações foram testadas, demonstrando mais uma vitória do TensorFlow sobre as demais. A Figura 14 demonstra o resultado do teste, em que pode-se observar o tempo em segundos para cada época de treinamento em cada um dos *frameworks*.

Figura 14 – Tempo de Treinamento dos Frameworks - Dataset MNIST



Fonte: (BHATIA, 2017)

Como citado anteriormente, o estudo realizado por (BHATIA, 2017), foi utilizado como base inicial de estudo, por ter sido realizado testes em apenas 4 *frameworks*, analisando imagens do *dataset* CIFAR-10, e apenas 3 *frameworks*, analisando imagens do *dataset* MNIST, sem falar que o único quesito comparado foi velocidade de treinamento, é necessário mais análises para chegar a uma conclusão de qual a melhor aplicação a ser utilizada.

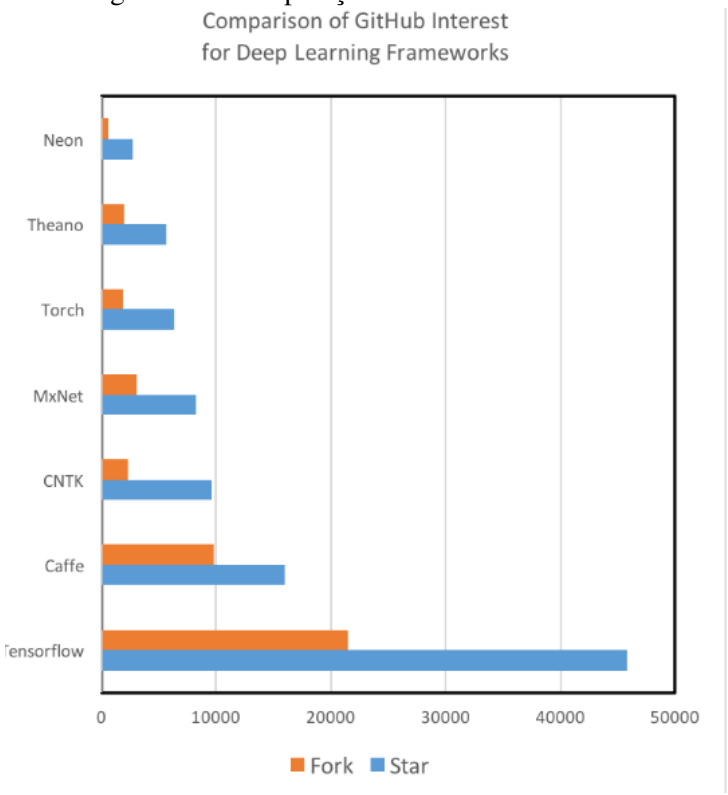
O *benchmarking* pulicado por (RUBASHKIN, 2017), compara as aplicações Theano, TensorFlow, Torch, Caffe, MXNet, Neon e CNTK, em uma combinação de experiências subjetivas passadas pelo autor no reconhecimento de imagens e no reconhecimento de fala. Além desse estudo ter sido realizado em mais softwares, o autor não analisa somente a velocidade de treinamento; incluindo, portanto, outros quesitos importantes.

O primeiro quesito comparado foi as linguagens de programação que cada aplicação suporta. Essa comparação, segundo o autor, é bastante importante, pois o programador deve sempre que possível usar uma estrutura que suporte a linguagem na qual ele esteja mais familiarizado. Contudo, o autor recomenda que o programador deva conhecer as linguagens C ++ e Lua. Nesse quesito não é possível elencar um vencedor, porque se trata de um quesito subjetivo.

O segundo quesito leva em conta a quantidade e a qualidade de materiais disponíveis

sobre as aplicações. Um dos critérios para medir isso foi o engajamento da comunidade GitHub em cada aplicação. A Figura 15 mostra a comparação do interesse de cada *framework* no GitHub, enquanto a Figura 16 mostra a quantidade de contribuições de cada *framework* no GitHub. Nesse quesito o TensorFlow apresenta vantagem sobre os outros *frameworks*.

Figura 15 – Comparação de Interesse no GitHub



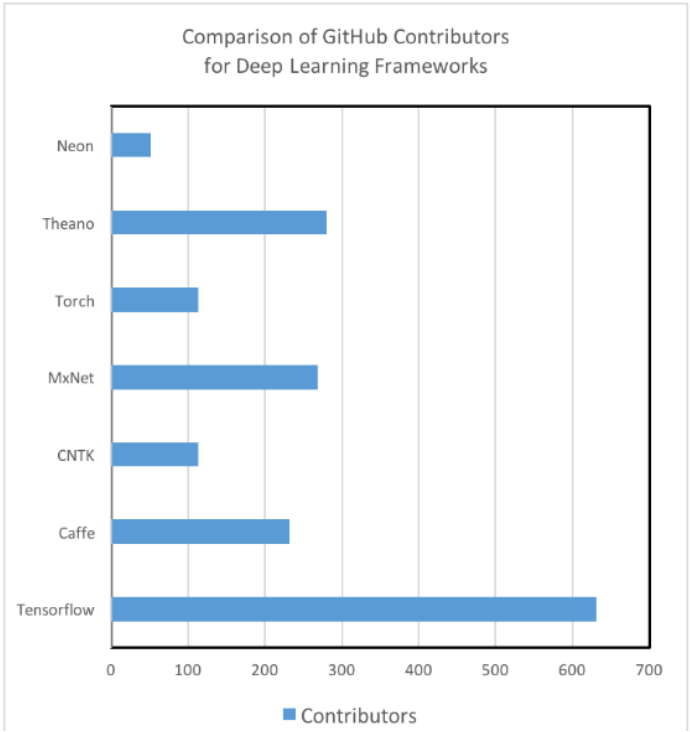
Fonte: (RUBASHKIN, 2017)

O terceiro quesito, considerou a capacidade do *framework* em modelar redes neurais convolucionais. Dentre os recursos analisados está o espaço de oportunidade para definir modelos, a disponibilidade de camadas pré-construídas e as ferramentas e funções disponíveis para conectar essas camadas. Esse é um quesito importante a ser considerado para o desenvolvimento de redes neurais voltadas ao reconhecimento de imagem, uma vez que as CNNs são as redes neurais mais utilizadas para esse fim. Nessa questão o Tensorflow e o Torch empataram na primeira colocação.

O quarto quesito analisou a capacidade de modelagem em redes neurais recorrentes. As RNNs são utilizadas principalmente por aplicações voltadas ao reconhecimento de fala, previsão de séries temporais, criação de legendas de imagens entre outras tarefas que requerem processamento de informações sequenciais.

O quinto quesito verifica a arquitetura. O objetivo aqui é verificar a facilidade de criar e treinar uma rede neural. Portanto, nessa análise verifica-se a simplicidade e a modularidade do software no desenvolvimento. O TensorFlow, o Torch e o MXNet têm uma arquitetura mais simples e modular, se comparados com estruturas como o Caffe, o qual exige uma quantidade significativa de trabalho para criar uma nova camada. Além disso, o TensorFlow possui uma

Figura 16 – Comparação de Contribuições no GitHub



Fonte: (RUBASHKIN, 2017)

vantagem na depuração e monitoramento, já que traz consigo a ferramenta *web* Tensorboard.

O sexto quesito é quanto a velocidade de treinamento nos modelos de rede convolucionais e recorrentes. Esse quesito é importante para os fins desse estudo, uma vez que o modelo de rede neural mais adequado para a análise de imagens é o modelo convolucional. O *framework* Torch ganhou nesse ponto.

O sétimo quesito se preocupa com o suporte disponibilizado pelo *framework* para a realização de treinamentos em GPUs. Treinamentos realizados em GPUs tendem a ser muito mais rápidos do que os realizados por CPU, principalmente quando se trata de *big data*. Portanto, possuir suporte para treinamentos em GPUs é importante quando se trabalha com uma grande quantidade de dados. O Mxnet possui vantagens nesse quesito sobre os demais *frameworks*.

Finalizando, o quesito número oito se refere a compatibilidade da aplicação com o Keras. Nesse caso, o Keras seria utilizado como o *front-end* e o *framework* seria utilizado como o *back end*.

A Tabela 4 mostra uma tabela com as pontuações dos *frameworks* nos quesitos citados acima segundo (RUBASHKIN, 2017).

Tendo como base os resultados dos testes das duas pesquisas, chegou-se a conclusão que o TensorFlow é a melhor opção, principalmente por apresentar uma ótima capacidade de modelagem de redes neurais convolucionais e por ser de fácil utilização para modelar arquiteturas de *front end*. Além disso, o TensorFlow possui um bom desempenho na questão de velocidade de treinamento. Por esses motivos o TensorFlow será o *framework* utilizado no desenvolvimento do software que será proposto no próximo capítulo.

Tabela 4 – Comparação de Aplicações

	Linguagens	Tutoriais e Materias de Treinamento	Compatibilidade com modelos CNN	Compatibilidade com modelos RNN	Arquitetura: Facilidade e Modularidade	Velocidade	Suporte em GPU	Compatibilidade com o Keras
Theano	Python, C++	++	++	++	+	++	+	+
TensorFlow	Python	+++	+++	++	+++	++	++	+
Torch	Lua, Python	+	+++	++	++	+++	++	
Caffe	C++	+	++		+	+	+	
MXNet	R, Python, Julia, Scala	++	++	+	++	++	+++	
Neon	Python	+	++	+	+	++	+	
CNTK	C++	+	+	+++	+	++	+	

Fonte: (RUBASHKIN, 2017)

4.3.2 Testes Realizados

Essa seção apresenta dois testes que foram realizados pelo autor. O primeiro tem o objetivo de comparar o desempenho de uma rede neural convolucional com o desempenho de uma rede neural MLP. Já o segundo tem por objetivo verificar o funcionamento do TensorFlow no reconhecimento de imagens, além de testar três arquiteturas de redes neurais distintas.

No primeiro teste duas redes neurais foram testadas, uma MLP e outra CNN. Ambas tinham como objetivo classificar 10.000 dígitos do *dataset* MNIST. A implementação e os parâmetros de treinamento utilizados estão descritos na Tabela 5 e na Tabela 6.

Tabela 5 – Parâmetros da Rede de Teste MLP

Parâmetros da Rede	
Neurônios da Primeira Camada	784 (28x28)
Neurônios da Segunda Camada	30
Neurônios da Camada de Saída	10

Parâmetros de Aprendizado	
Passo de Aprendizagem	0.1
Épocas de Treinamento	30

Fonte: (Elaborado pelo Autor, 2018)

É importante observar que a diferença entre a arquitetura da rede MLP para a arquitetura da rede CNN é apenas a camada convolucional. O teste foi realizado dessa forma para que ambas as redes competissem em igualdade. A quantidade de neurônios da primeira camada foi ajustada para que cada neurônio recebesse um píxel da imagem de entrada. E as imagens possuem o tamanho de 28x28, sendo assim a primeira camada de ambas as redes possui 784 neurônios. A camada oculta, ou seja, a segunda camada, conta com 30 neurônios. Esse número foi escolhido para tornar a rede pequena e, consequentemente, tornar o treinamento rápido. A camada de saída utiliza 10 neurônios pois ela é utilizada somente para mostrar o resultado da

Tabela 6 – Parâmetros da Rede de Teste CNN

Parâmetros da Rede	
Neurônios da Primeira Camada	784 (28x28)
Neurônios da Segunda Camada	30
Neurônios da Camada de Saída	10

Camada Convolucional	
Tamanho do Mapa de Recursos	20x24x24
Tamanho do Filtro	5x5
Tamanho da Camada de Pooling	20x12x12

Parâmetros de Aprendizado	
Passo de Aprendizagem	0.1
Épocas de Treinamento	30

Fonte: (Elaborado pelo Autor, 2018)

análise, lembrando que os dígitos do *dataset* MNIST variam de 0 a 9. Dessa forma, o neurônio que fica ativo na camada de saída após a analise indica qual foi resultado da classificação. Portanto, se, por exemplo, o neurônio de saída de número 5 ficar ativo, indica que a rede acredita que o dígito de entrada foi o número 5. Para o treinamento de máquina, foi utilizado os métodos de aprendizado supervisionado descida gradiente e *backpropagation*. O passo de aprendizado utilizado foi de 0.1. Esse passo poderia ter sido maior para aumentar a velocidade de convergência, mas esse foi o valor escolhido para que não houvesse muito variância de acurácia entre uma época e outra.

O resultado do teste mostrou que a CNN alcançou uma acurácia significativamente maior, alcançando ao final de 10 épocas, a acurácia de 96,28%, enquanto que a MLP alcançou a acurácia de 93,61% ao final das mesmas 10 épocas. A diferença se tornou ainda mais expressiva quando o número de épocas de treinamento aumentou. Em outro teste com as mesmas redes, analisando o mesmo *dataset*, porém agora utilizando 60 épocas de treinamento, a rede CNN alcançou o ótimo resultado de 98,17% de acurácia, enquanto que a rede MLP alcançou o resultado de 93,82% de acurácia.

Com o resultado dos testes foi possível concluir que uma rede neural convolucional é mais eficiente do que uma rede neural MLP. Portanto, o modelo de rede neural convolucional será o utilizado daqui para frente nesse documento.

No segundo teste, o *framework* TensorFlow foi utilizado em uma aplicação para a classificação de flores. Esse teste utilizou um *dataset* de treinamento com 5 classes de flores, contendo margaridas - com 633 imagens -, dentes-de-leão - com 898 imagens -, rosas - com 641 imagens -, girassóis - com 699 imagens - e tulipas - com 799 imagens. Os testes foram realizados utilizando três arquiteturas de rede distintas, a MobileNet, o Inception-V3 e o Inception-V4. As três arquiteturas de rede neural contaram com uma base pré-processada pela Google no conjunto de dados do ImageNet Large Visual Recognition Challenge, que pode realizar o reconhecimento de 1.000 classes diferentes de imagens. Portanto, o treinamento realizado apenas incrementou o conhecimento da rede para realizar a diferenciação das classes de flores citadas. Foi usado 5.000 épocas de treinamento com passo de aprendizado 1. O código

utilizado para a construção das redes foi uma adaptação do *codelab* (BRAIN, 2018b). Após o treinamento da rede, foi realizado um novo teste com 75 imagens - 15 para cada classe de flor - e essas imagens não estavam dentre as imagens de treinamento da rede. Essas novas imagens serão chamadas de dados não treinados para os fins desse experimento. Esse teste foi realizado para verificar se a rede realmente conseguiu ter a capacidade de generalização, podendo, portanto, classificar imagens nunca vistas antes. O critério adotado para verificar se a rede acerta ou erra na classificação foi baseado no *score* da saída. O acerto foi contabilizado se o *score* da classe era maior que 60%, caso contrário a contabilização foi dada como errada. O critério foi realizado dessa forma pois o algoritmo não mostra na sua saída apenas a classe que ele acredita ser a correta, ao invés disso, ele apresenta um *score* para cada uma, como pode ser visto na Figura 17. A imagem mostra a análise de uma imagem com um buquê de margaridas. Pode-se ver que a saída da rede indicou a probabilidade de na imagem haver margaridas, assim como a probabilidade de haver as outras classes de flores. Nessa imagem de exemplo, a probabilidade de haver margaridas foi de 94,53%, a de haver girassóis de 5,39%, a de haver rosas de 0,72%, a de haver tulipa 0,01% e a de haver dente-de-leão 0%. O treinamento da rede, assim como a análise das imagens, foram realizados via CPU, na CPU Intel Core I3-4130, sistema operacional Windows 10, 4GB de memória RAM.

Figura 17 – Sistema de scores do teste

Fonte: (Elaborado pelo Autor, 2018)

A MobileNet é uma arquitetura de rede neural convolucional pequena, que tem como principal objetivo ser utilizado em equipamentos *mobile*. Nesse teste ela foi quem obteve o tempo treinamento mais rápido, entretanto vale lembrar que isso se deve muito ao fato de ser

uma arquitetura de pequeno porte. Sua acurácia sobre os dados de treinamento foi bastante satisfatória chegando a 92.2%. Contudo, a acurácia sobre os dados não treinados caiu significativamente, errando na classificação 15 vezes, alcançando a acurácia de 88,75%.

A Inception-V3, que é uma arquitetura mais robusta se comparado com a MobileNet, alcançou o melhor resultado em termos de acurácia na análise dos dados não treinados, e o segundo lugar na análise dos dados treinados, além de ser a segunda em termos de velocidade. Na análise dos dados de treinamento a acurácia foi de 91.3%, enquanto que na análise dos dados não treinados a acurácia foi de 92,5%, errando em apenas 10 imagens.

A Inception-V4, versão mais recente da arquitetura do tipo Inception, alcançou um desempenho um tanto quanto desanimador, apesar dos testes conduzidos por (SZEGEDY et al., 2016) - testes que sem dúvida são muito mais exigentes e completos que esse apresentado - indicarem que essa solução obteria resultados melhores. Além de ter obtido o tempo mais longo de treinamento, a sua acurácia nos dados de treinamento foi de 87,3%, enquanto que sua acurácia nos dados não treinados foi de 89,5%, com 14 erros, ficando em segundo lugar.

Na Tabela 7 é possível ver os resultados do teste.

Tabela 7 – Resultados do Teste TensorFlow for Poets

Arquitetura	Acur. Dados não Treinados	Acur. Dados Treinados	Tempo Treinamento
Inception V3	**92,50%**	91,30%	31min
MobileNet	88,75%	**92,20%**	**12min**
Inception V4	89,50%	87,30%	38min

Fonte: (Elaborado pelo Autor, 2018)

De acordo com os testes foi possível inferir que a arquitetura Inception-V3 é a arquitetura que apresenta o melhor desempenho nos dados não treinados. No próximo capítulo, onde é apresentado o software que foi desenvolvido, as três redes neurais (a Mobilenet V1 224 (HOWARD et al., 2017), a Inception V3 (SZEGEDY et al., 2015b) e a Inception V4 (SZEGEDY et al., 2016)) foram utilizadas.

5 IMPLEMENTAÇÃO DO PROJETO

Esse capítulo apresenta o software desenvolvido. Esse trabalho integra um projeto realizado em conjunto com o graduando Matheus Zimmermann Suzin (SUZIN, 2018). O software implementado teve como objetivo o reconhecimento facial em tempo real com uma rede neural convolucional. Os testes de funcionamento foram feitos no bloco I da Universidade de Caxias do Sul, no teste com alunos da universidade e, no teste com pessoas de fora do ambiente acadêmico, na residência do participante. Infelizmente, poucos estudantes aceitaram ceder suas imagens para esse experimento, o que resultou em uma base de dados menor do que a esperada inicialmente. Contudo, o experimento foi realizado e pode-se chegar em alguns resultados.

5.1 MÉTODO

O presente trabalho foi realizado de acordo com o seguinte cronograma:

1. **Primeira Fase:** Coleta das imagens faciais das pessoas que aceitaram participar da amostra do estudo. Tal fase teve como objetivo constituir a base de dados para a alimentação da rede neural. **Duração decorrida: 1 semana;**

2. **Segunda Fase:**Implementação do software utilizando o *framework* TensorFlow (BRAIN, 2018a) e as arquiteturas CNNs, InceptionV3, Inception V4 e MobileNet. Nessa fase foi realizada a implementação do software. **Duração decorrida: 8 semanas;**

3. **Terceira Fase:** Testes de funcionamento do software implementado na sala de aula escolhida, na presença dos alunos e professor. E testes de funcionamento do software na residência dos participantes que estão fora do ambiente acadêmico. **Duração decorrida: 3 semanas;**

4. **Quarta Fase:** Redação do texto e finalização da segunda etapa deste estudo. O período foi reservado para o registro dos conhecimentos adquiridos e dos resultados obtidos nas fases de experimentação para o trabalho de conclusão de curso. **Duração decorrida: 2 semanas;**

5.2 APRESENTAÇÃO DO SOFTWARE

O software foi desenvolvido em um sistema IoT (Internet das coisas). Ele conta com a utilização de três equipamentos que ficam conectados entre si (um servidor de rede neural, uma máquina cliente e um celular), além de uma base de dados na nuvem. O programa possui, basicamente, três etapas: treinamento da rede neural, análise das imagens das pessoas em tempo real e, finalizando, o armazenamento da informação na nuvem.

A primeira etapa, o treinamento da rede neural, é realizada em um servidor próprio para esse propósito. Sua função é treinar e criar a base de conhecimento, arquivo que contém a arquitetura da rede com todos os pesos e vieses adquiridos no treinamento com os métodos de aprendizado de máquina.

A segunda etapa, a análise das imagens das pessoas em tempo real, é realizada por uma máquina cliente, que nesse experimento é um Raspeberry Pi. O Raspeberry Pi lê o arquivo de conhecimento que foi criado na etapa anterior, pelo servidor da rede neural, e, com isso, carrega o grafo já capacitado para realizar a análise das imagens. A câmera utilizada para a captura das imagens foi a do celular Motorola E5, em conjunto com o aplicativo IP Webcam (KHLEBOVICH, 2017). O Raspeberry PI, através da rede *wi-fi*, captura as imagens do celular e realiza todo o processamento referente a identificação das pessoas que aparecem nas imagens.

A terceira etapa, o armazenamento da informação na nuvem, é, assim como na segunda etapa, realizada pela máquina cliente. Com a identificação já realizada o Raspeberry Pi envia as informações referentes a ela para a nuvem, em uma base de dados que foi disponibilizada no Google Cloud.

A visão geral do projeto, que foi descrito nos parágrafos anteriores, é ilustrada na Figura 18, onde pode ser visto as etapas: de treinamento da rede neural, de análise das imagens das pessoas em tempo real e de armazenamento das informações na nuvem.

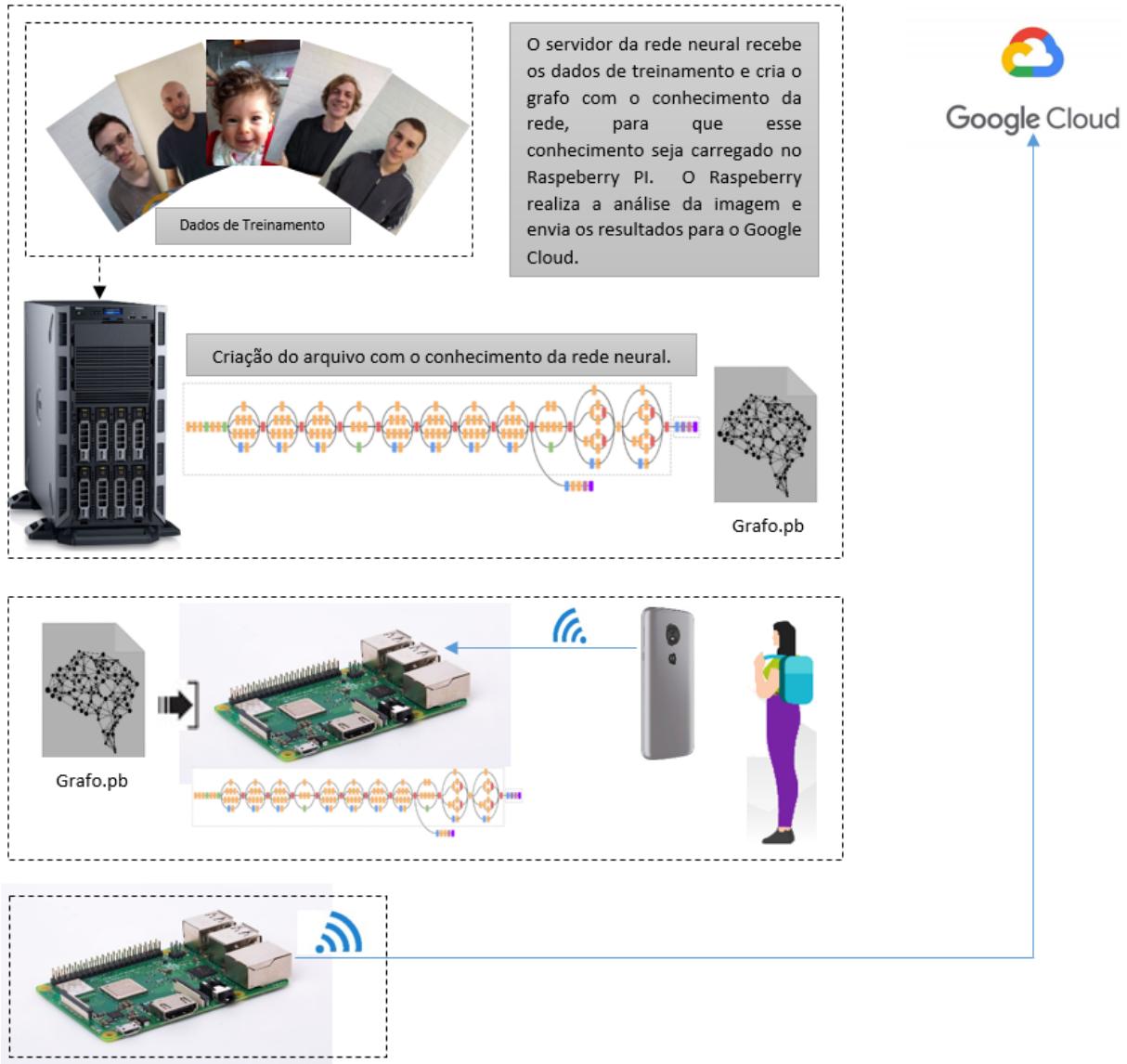
5.3 MATERIAIS E MODELOS UTILIZADOS

O hardware utilizado para o servidor da rede neural foi um Intel Core I3-4130 de 3.40GHz, com 8GB de memória RAM e sistema operacional Windows 10X64. Para a máquina cliente foi utilizado um Raspberry Pi 3 Model B+, com uma CPU quad-core de 1.4 GHz, memória RAM de 1GB e sistema operacional Linux. A câmera utilizada, tanto para alimentar a base de dados de treinamento, tanto para a análise dos dados, foi a câmera do celular Motorola E5, que possui 13Mpx de resolução. O sistema operacional utilizado pelo celular foi o Android 8.0 Oreo.

A implementação do software foi feita utilizando a linguagem de programação Python, na versão 3.5.2 e o *framework* Tensorflow, na versão 1.8.0. Apesar do *framework* Tensorflow disponibilizar todo o suporte para a realização de treinamento da rede neural em GPUs, o que tornaria a etapa de treinamento mais rápida, nesse projeto o treinamento foi realizado em CPU, pois não havia a disponibilidade de GPUs com suporte a linguagem de programação CUDA.

O software foi implementado para utilizar 3 arquiteturas de rede neural convolucional pré-treinadas, a Mobilenet V1 224 (HOWARD et al., 2017), a Inception V3 (SZEGEDY et al., 2015b) e a Inception V4 (SZEGEDY et al., 2016).

Figura 18 – Infraestrutura



Fonte: (Elaborado pelo Autor, 2018)

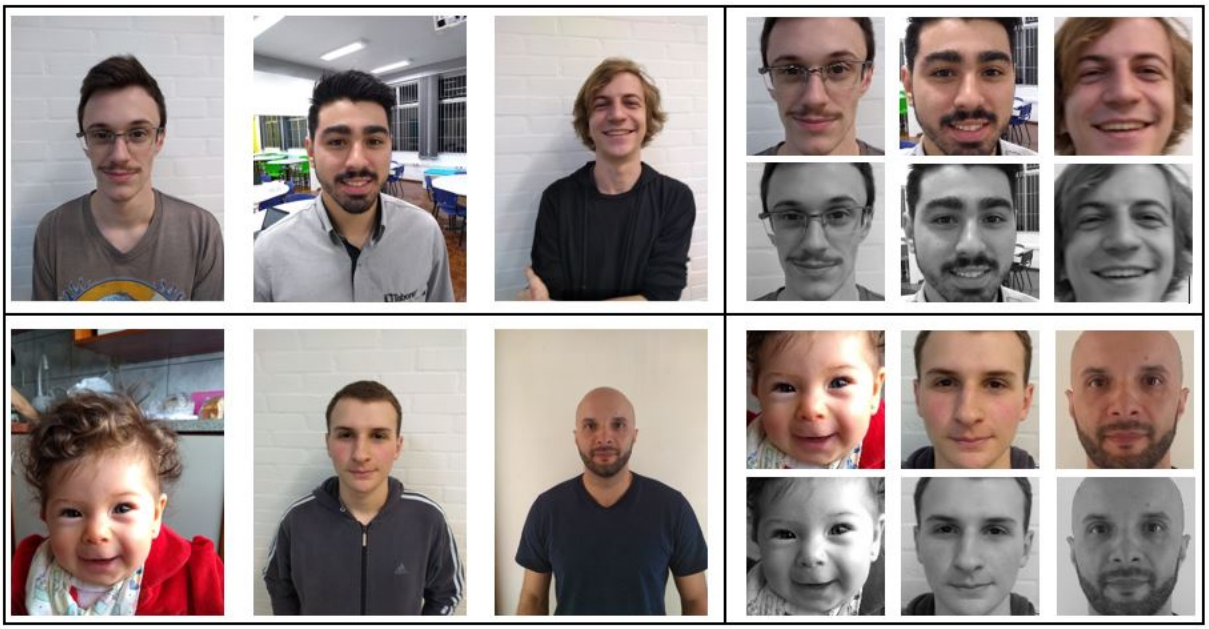
5.4 COLETA E PRÉ-PROCESSAMENTO DE IMAGENS

Inicialmente, foi necessário realizar a coleta de imagens das faces de alunos dispostos a participar do experimento. A amostra de faces de voluntários foi formada por 10 pessoas. Essas imagens foram coletadas no bloco I da Universidade de Caxias do Sul e contou com a cooperação de 5 alunos, além de mais 5 pessoas de fora do ambiente acadêmico. Cada uma delas foi fotografada em diversos ângulos, posições e expressões faciais diferentes. A base de dados ainda contou com outras fotos dos participantes que foram retiradas das suas redes sociais, afim de conceder um maior número de informações para a rede neural. O banco de dados de imagens obteve entre 74 e 160 fotos de cada pessoa, em um total de 1149 imagens.

Após a fase de coleta das imagens, o software as submete a uma fase de pré-processamento.

Esse pré-processamento captura somente as faces contidas na imagem e as copia em dois padrões de cores, no padrão de cor original e no padrão de cor em escala de cinza, como pode ser visto na Figura 19. Isso foi feito com o auxílio da biblioteca Opencv (MORDVINTSEV; K., 2013) para Python. O Opencv possui funções para o reconhecimento de faces através do seu classificador Cascade Classification (MORDVINTSEV; K., 2018). O objetivo do software em seccionar somente a face das pessoas é tentar melhorar o aprendizado da rede neural, pois, na fase de análise das imagens, somente as faces são passadas para o processamento da rede neural. As imagens da face na escala de cinza, tem o mesmo objetivo, melhorar o aprendizado da rede neural, porém elas sofrem essa transformação para diminuir o efeito causado pela luminosidade, fator que pode dificultar a busca de padrões na imagem.

Figura 19 – Pré-processamento das imagens de treinamento



Fonte: (Elaborado pelo Autor, 2018)

5.5 IMPLEMENTAÇÃO DA REDE NEURAL CONVOLUCIONAL

A implementação da rede neural foi realizada no servidor de rede neural. Ele realiza essa tarefa carregando uma das três arquiteturas de rede neural convolucional disponíveis, e, após, conecta uma camada de rede totalmente conectada no final do grafo carregado. A camada final, que é denominada de *softmax*, é quem informa o resultado final do processamento.

A construção do grafo de fluxo de dados é realizada lendo as informações da arquitetura pré-treinada que foi selecionada na parametrização do software. Como já dito, pode-se escolher entre as arquiteturas; Mobilenet V1 224 (HOWARD et al., 2017), Inception V3 (SZEGEDY et al., 2015b) e Inception V4 (SZEGEDY et al., 2016). Essas arquiteturas ficam salvas no servidor em um diretório especifico, no formato de arquivo '.pb'. O *framework* Tensorflow disponibiliza o método necessário para a leitura de modelos salvos no formato '.pb', construindo

o grafo a partir deles. A função que realiza essa tarefa é a 'ParseFromString', função essa dos objetos do tipo 'GrafDef'. Após a leitura do modelo e, consequentemente, com o grafo carregado na memória, dois tensores Tensorflow são adicionados ao grafo, um para decodificação e outro para codificação de imagens. O primeiro é o responsável pela extração dos sumários das imagens, ou seja, é o responsável em transformar a imagem em uma matriz com valores *float*. O segundo é responsável pela codificação de uma matriz com valores *float* em uma imagem no formato '.jpg'.

Os tensores são os objetos principais do *framework* Tensorflow. Esses objetos representam uma computação parcialmente definida que eventualmente produzem um valor de saída. Os programas TensorFlow funcionam primeiro construindo um grafo de objetos tensores, detalhando como cada tensor é calculado com base nos outros tensores disponíveis e, em seguida, executando partes deste grafos para obter os resultados desejados (BRAIN, 2018c).

A criação da camada *softmax* é feita com base na quantidade de rótulos que se pretende identificar, para cada um dos rótulos é criado um novo tensor. A camada *softmax* é a camada de saída do modelo e tem a função de informar o resultado final da classificação. O resultado final da classificação é informado através de um *array* com o tamanho igual a quantidade de rótulos. Esse *array* é composto de valores *float* que variam entre 0 e 1, sendo que a soma de todos os valores do *array* é sempre igual a 1. Portanto, pode-se dizer, em outras palavras, que a camada *softmax* retorna a probabilidade que a imagem de entrada possui de ser pertencente a cada um dos rótulos. Nesse experimento a camada *softmax* teve 10 tensores de saída, um para cada um dos 10 participantes.

Após ler o modelo pré-treinado criar ao menos um tensor de decodificação de imagens, um tensor de codificação de imagens e conectar uma camada *softmax* na saída do modelo, a rede neural convolucional já está montada. Porém, ela ainda não está apta para realizar análises, pois necessita passar primeiro pela etapa de treinamento.

5.6 TREINAMENTO DA REDE NEURAL

Para o aprendizado de máquina, foram utilizados os métodos de descida gradiente e *backpropagation*, ambos abordados no Capítulo 2 desse documento. Os parâmetros de aprendizagem que definiram a quantidade de épocas de treinamento e passo de aprendizado, foram definidos com base em testes, a fim de a rede neural não perder acurácia, nem por *overfitting*, nem por *underfitting*. Para isso foi utilizada a ferramenta Tensorboard, que mostra de forma gráfica todas as informações da etapa de treinamento.

O software realiza uma preparação das imagens do banco de dados antes de iniciar o processo de treinamento, separando-as em três grupos; imagens de treinamento, imagens de teste e imagens de validação. O primeiro grupo é formado pelas imagens que são utilizadas no treinamento da rede neural, ou seja, é a partir da análise dessas imagens que a rede neural ajusta seus pesos e vieses. O segundo grupo é formado pelas imagens que são utilizadas pela

rede neural nos testes do aprendizado. Quando necessário, na hipótese de um erro na identificação da classe no momento do teste, a rede neural utiliza a imagem que foi erroneamente classificada para o aprendizado. O último grupo é formado pelas imagens que são usadas na etapa de validação da rede neural. A etapa de validação é parecida com a etapa de teste, mas com uma diferença, a rede neural não utiliza as imagens de validação para aprender em caso de erros na identificação. Ao invés de aprender com erro, a rede neural salva os resultados da validação para que eles sejam analisados pelo operador. A quantidade de imagens utilizadas para o treinamento, o teste e a validação, pode variar, contudo, para os fins desse experimento, as quantidades utilizadas foram parametrizadas em 80%, 10% e 10% do total de imagens consecutivamente.

A classe *train* do Tensorflow possui funções prontas para a realização do treinamento da rede neural. Essas funções são adicionadas ao grafo através da abertura de um escopo. Para esse experimento a função *train* utilizada foi a 'GradientDescentOptimizer' (BRAIN, 2018d). Essa função foi parametrizada com um passo de aprendizado de 0,1. Esse valor foi escolhido para que o aprendizado da rede fosse lento e gradual, com poucas mudanças nos pesos e vieses da rede com o passar das épocas de treinamento. A quantidade de épocas de treinamento utilizada foi de 1.250, pois a acurácia da rede nos dados de validação não aumentou a partir desse ponto.

Para agilizar o treinamento da rede neural foi criado um *cache* em disco dos sumários das imagens que já foram decodificadas. A imagem, antes de ser enviada para a rede neural, precisa ser redimensionada e decodificada para as dimensões esperada pela arquitetura. Para os modelos Inception V3 e Inception V4, as dimensões da imagem devem ser de 299x299 pixels, e para o modelo Mobilenet V1 224, as dimensões da imagem devem ser de 224x224 pixels. O tensor de decodificação é quem realiza essa tarefa, redimensionando a imagem e a decodificando para, após, salvar os seus sumários no *cache* em disco. Quando a rede neural precisa utilizar uma imagem, seja para o treinamento, seja para o teste, seja para a validação, ela procura os sumários no *cache* em disco. Caso eles não existam no *cache* em disco, a rede neural aciona o tensor de decodificação para a criação do *cache*.

Com a rede neural convolucional criada e com o escopo de treinamento definido, o treinamento é iniciado. O programa seleciona, de forma aleatória, 10 imagens que pertencem ao grupo de treinamento por época. Essas imagens passam pelo processamento da rede neural convolucional e o método de aprendizado altera os pesos e vieses da rede neural de acordo com o passo de aprendizado. A cada 10 épocas de treinamento, há uma etapa de teste e uma etapa de validação, até que todas as épocas de treinamento terminem, que nesse caso são 1.250 épocas. As etapas de teste e de validação são feitas com 10 imagens, selecionadas aleatoriamente do grupo de imagens de teste e do grupo de imagens validação respectivamente.

Após a realização de todas as épocas de treinamento, o programa salva o grafo treinado no formato de arquivo '.pb'. Esse grafo, a partir desse momento, já pode ser copiado e utilizado na máquina cliente.

5.7 EXECUÇÃO NOS DISPOSITIVOS FÍSICOS

Na etapa de análise das imagens, após a rede estar treinada e apta para realizar a identificação, o software realiza a análise seguindo os seguintes passos: filmagem das pessoas pela câmera do celular, captura da imagem pelo Raspeberry PI, procura de faces na imagem, envio das faces encontradas para a rede neural e persistência das informações de identificação.

O Raspeberry PI utiliza as mesmas funções que são utilizadas na implementação da rede neural para carregar o grafo. A diferença é que, nesse momento, não é carregado apenas o modelo pré-treinado, ao invés disso, é carregado toda a rede neural, com todos os pesos, vieses e tensores configurados. Ou seja, essa etapa carrega a rede neural completa e apta para realizar a identificação de pessoas nas imagens.

A captura das imagens no Raspeberry PI foi realizada, como já mencionado, por um celular Motorola E5, em conjunto com o aplicativo para *smartphone* IP Webcam (KHLEBOVICH, 2017). Esse aplicativo disponibiliza as imagens capturadas pela câmera do celular para acesso *web*, através do link http://'IP do Celular':8080/shot.jpg. A captura da imagem é feita com o auxílio da biblioteca Python urllib (FOORD, 2018), que fornece funcionalidades para a captura de informações por rede. Após a imagem ser capturada, ela é enviada para a análise do OpenCV (MORDVINTSEV; K., 2013).

A biblioteca OpenCV para Python possui funções para o reconhecimento de faces de pessoas através do seu classificador Cascade Classification (MORDVINTSEV; K., 2018). O Cascade Classification é configurado a partir da leitura de arquivos no formato XML. Nesses arquivos de configuração há as informações que são necessárias para a tomada de decisão. Existem XMLs para diversas situações, desde o reconhecimento de olhos até o reconhecimento de pessoas em imagens. Nesse experimento foi utilizado 3 XMls, o haarcascade_frontalface_alt2 (PISAREVSKY, 2013a), o haarcascade_profileface (PISAREVSKY, 2013b) e o haarcascade_frontalface_alt_tree (PISAREVSKY, 2013c), ambos utilizados para o reconhecimento de faces em imagens. Todas as imagens são analisadas por esses classificadores, que têm a função de, ou passar a imagem da face para o processamento da rede neural, ou ignorar a imagem. Se a imagem conter uma ou mais faces, o OpenCV secciona as faces encontradas e as envia para o processamento da rede neural. Se não houver nem uma face na imagem, a mesma é ignorada.

A rede neural é a responsável pela identificação das faces e é gerenciada pelo *framework* Tensorflow. Ela recebe todas as imagens de faces que foram encontradas na etapa anterior. Após o processamento, a rede neural retorna um *array* com a probabilidade que a face tem de pertencer a cada um dos 10 rótulos. O software analisa qual o rótulo que recebeu a maior probabilidade e verifica se essa probabilidade está acima ou abaixo do nível de confiança. O nível de confiança é um parâmetro do software que é utilizado para evitar que probabilidades baixas fossem contabilizadas. Nos experimentos o nível de confiança utilizado foi de 85%. Isso significa que a rede somente contabiliza a identificação se a probabilidade de a imagem pertencer ao rótulo for maior ou igual a 85%, caso contrário a rede informa que a face é desconhecida.

Nos casos em que a face é reconhecida, a rede neural codifica a imagem e a salva nas faces identificas.

A última etapa é a persistência das informações e a apresentação dos resultados. Nesse momento, o software monta os dados em um objeto no formato JSON (CROCKFORD, 2018), com informações sobre a identificação, como: nome da pessoa identificada, imagem da face que teve sucesso no reconhecimento, hora, sala e data onde ocorreu a identificação. Esse objeto é codificado em base64 e enviado para base de dados na nuvem, o Google Cloud, com o protocolo de envio mqtt (OBERMAIER, 2014). Por fim, é apresentado na tela da máquina cliente a imagem em tempo real com a face da pessoa identificada marcada com um retângulo azul, com o seu nome acima do retângulo.

As etapas descritas acima podem ser observadas na Figura 20.

5.8 AVALIAÇÃO DOS RESULTADOS

Nessa seção é apresentado como os testes foram realizados e quais foram os resultados obtidos. Os testes com o software foram realizados em sala de aula, para os alunos da Universidade, e na residência do participante, para pessoas de fora do ambiente acadêmico, com as arquiteturas de rede Inception V3, Inception V4 e Mobilenet V1 224. Contudo, os resultados da arquitetura Inception V4 não foram contabilizados, uma vez que o Raspeberry PI não conseguiu carregar o modelo por falta de memória RAM.

5.8.1 Regras de Avaliação

A captura das imagens dos participantes foi feita filmando-os com a câmera do celular. A filmagem foi feita sem interrupções por um tempo de aproximadamente 10 segundos e com a câmera sendo movida para os lados, para frente e para trás. Apesar do ângulo da filmagem ser levemente alterado para os lados, a câmera ficou sempre focada para a frente da face do participante, pois o software não analisa imagens de perfil. A distância entre a câmera e o participante variou entre 30 centímetros e 1 metro, mais ou menos a mesma distância que as fotos para o banco de dados de imagem foram realizadas. O ambiente onde os testes foram realizados foi o mesmo ambiente em que as fotos para o treinamento foram registradas, sem variações significativas de luminosidade.

Por se tratar de uma filmagem em tempo real, dificilmente o software consegue reconhecer o participante em todos os *frames* capturados. Houve a ocorrência de erros e de resultados que ficaram abaixo do nível de confiança durante a identificação, conforme a posição da câmera. Portanto, foi necessário criar um padrão para poder contabilizar se a identificação ocorreu com sucesso ou com falha. O padrão seguiu as seguintes regras:

- Os *frames* que eram analisados e que ficaram abaixo do nível de confiança, apresentando como resultado pessoa desconhecida, não foram contabilizados, nem como erro, nem

Figura 20 – Estrutura da Análise

Fonte: (Elaborado pelo Autor, 2018)

como acerto.

- O restante dos *frames*, que apresentaram como resultado algum dos rótulos, eram analisados. Se na maioria desses *frames* o resultado apresentado era o correto, esse teste foi considerado correto. Por outro lado, se na maioria dos *frames* o resultado apresentado era incorreto, o teste foi considerado como incorreto.

Dessa forma, o processo de testes foi realizado e os resultados estão descritos na próxima seção.

5.8.2 Resultados Obtidos

A arquitetura Inception V3 obteve o melhor resultado em termos de acurácia, alcançando o resultado de 100%. Todos os participantes foram identificados corretamente. Pôde-se observar que houve poucos *frames* que apresentaram erros na identificação e que, na maioria das vezes, os *frames* que não apresentaram a identificação correta era devido ao resultado ficar abaixo do nível de confiança. O tempo de processamento por *frame* não foi contabilizado nos testes. Contudo, pode-se observar que o tempo de análise da arquitetura Inception V3 era um pouco mais elevado se comparado com a arquitetura Mobilenet V1 224. Entretanto, esse maior tempo de análise, tempo esse que fica em torno de 0,8 segundos, não chega a ser muito significativo. Outro fator que pôde ser observado, foi que a identificação obteve melhores resultados quando a filmagem da pessoa ocorria em mais ou menos 45 cm distância.

A arquitetura Mobilenet V1 224 obteve um resultado inferior em termos de acurácia, alcançando o resultado 80%, o que significa que dois dos dez participantes não foram identificados com sucesso. Porém, esse resultado de 80%, foi computado de acordo com as regras de avaliação. Diz-se isso porque esses dois participantes que não foram identificados com sucesso obtiveram falha na maioria dos *frames* analisados, mas em algum momento da filmagem chegou a ocorrer a identificação correta. Assim como ocorrera na arquitetura Inception V3, os melhores resultados ocorreram quando a filmagem foi realizada a uma distância de mais ou menos 45 cm, e a maioria das vezes em que a identificação era incorreta era devido ao resultado ficar abaixo do nível de confiança.

Chegou-se à conclusão que a arquitetura Inception V3 é melhor para os objetivos do projeto. Pois, apesar de necessitar de um tempo maior para realizar a análise, ela compensa com uma acurácia superior, o que garante uma maior confiabilidade para o software.

6 CONSIDERAÇÕES FINAIS

A pesquisa reuniu informações sobre as áreas de redes neurais, aprendizado de máquina e mais especificamente o reconhecimento facial. Com as informações coletadas e com os testes realizados foi possível observar que, apesar das dificuldades de se obter um sistema de reconhecimento facial confiável, é possível alcançar bons resultados.

6.1 SÍNTESE DO TRABALHO

Nesse trabalho foi apresentado métodos de aprendizado de máquina e redes neurais artificiais, com ênfase maior nos métodos de aprendizado de máquina supervisionado e redes neurais convolucionais. As redes neurais convolucionais são, atualmente, as arquiteturas de redes neurais que alcançam a melhor taxa de acertos na classificação de imagens, e os métodos de aprendizado de máquina supervisionados são os mais utilizados para casos de classificação. Por esses motivos, foram os escolhidos para a implementação de um software que tem o objetivo de realizar o reconhecimento facial de pessoas em tempo real.

Foi realizado um estudo entre os *frameworks* mais utilizados em sistemas de que utilizam aprendizado de máquina e rede neural. Ao final do estudo, o *framework* Tensorflow foi o escolhido para ser utilizado na implementação do software.

O software, que foi implementado na linguagem de programação Python, utilizou três arquiteturas de rede neural convolucional para realizar o reconhecimento de 10 pessoas. Esse software operou realizando o treinamento da rede neural em um servidor próprio para esse fim, e a análise das pessoas em um equipamento cliente.

Os resultados obtidos foram, de certa forma, satisfatórios, uma vez que foi possível reconhecer os participantes do projeto.

6.2 TRABALHOS FUTUROS

Esse trabalho foi desenvolvido em um sistema com uma única rede neural operando por vez, pois a tarefa de identificação proposta era em um *dataset* fechado. Em um sistema de identificação com *dataset* fechado, a rede neural sempre apresenta como resultado uma das classes que ela foi treinada para identificar, por mais que a imagem não seja pertencente a nem uma classe. O que significa que, se a rede neural tentar identificar uma pessoa que não está entre as pessoas que ela foi treinada para reconhecer, o resultado não será, por exemplo, pessoa desconhecida. Ao invés disso, a rede neural sempre irá informar que a pessoa pertence a uma das classes do *dataset* de treinamento. Para um trabalho futuro, o software utilizará duas redes neurais convolucionais trabalhando hierarquicamente; uma primeira, que realizará a verificação, e uma segunda, que realizará a identificação. A primeira rede neural, a rede de verificação, terá o objetivo de analisar se a pessoa é ou não uma pessoa conhecida. Caso essa pessoa seja

conhecida, a imagem será encaminhada a segunda rede neural, a rede de identificação, que irá identificar quem é a pessoa. Com isso, o software poderá operar com um *dataset* aberto.

Além do software trabalhar com um *dataset* aberto, o trabalho futuro incluíra mais opções de arquiteturas de rede neural convolucional para utilização, tanto para a rede neural de verificação, tanto para a rede neural de identificação.

Finalizando, outro trabalho futuro será um estudo mais aprofundado nos métodos de aprendizado de máquina e nos métodos de coleta de imagens para a base de conhecimento.

REFERÊNCIAS

BEZERRA, E. **TensorFlow - Image Recognition**. 2018. Disponível em: <http://sbbd2016.fpc.ufba.br/sbbd2016/minicursos/minicurso3.pdf>. Acesso em: 19 de junho 2018.

BHATIA, J. **The Search for the Fastest Keras Deep Learning Backend**. 2017. Disponível em: <https://www.kdnuggets.com/2017/09/search-fastest-keras-deep-learning-backend.html> Acesso em: 29 maio 2018.

BRAGA, A. d. P.; CARVALHO, A. C. P. d. L. F.; LUDERMIR, T. B. Redes neurais artificiais : teoria e aplicações. , Rio de Janeiro, RJ, Brasil, 2007.

BRAIN, G. **Tensorflow: large-scale machine learning on heterogeneous systems**. 2018. Disponível em: <https://www.tensorflow.org>. Acesso em: 01 maio 2018.

BRAIN, G. **TensorFlow For Poets**. 2018. Disponível em: <https://codelabs.developers.google.com/codelabs/tensorflow-for-poets/> Acesso em: 02 junho 2018.

BRAIN, G. **Tensores**. 2018. Disponível em: <https://www.tensorflow.org/guide/tensors>. Acesso em: 04 julho 2018.

BRAIN, G. **tf.train.GradientDescentOptimizer**. 2018. Disponível em: <https://www.tensorflow.org/api_docs/python/tf/train/GradientDescentOptimizer>. Acesso em: 14 agosto 2018.

CAMPOS, V. et al. Distributed training strategies for a computer vision deep learning algorithm on a distributed gpu cluster. , Universitat Politécnica de Catalunya, Barcelona, Espanha, 2017.

COSTA, M. Deep learning online: um resumo do novo livro gratuito do mit. **InfoQ Brasil**, São Paulo, SP, Brasil, 2016.

CROCKFORD, D. **JSON encoder and decoder**. 2018. Disponível em: <https://docs.python.org/2/library/json.html>. Acesso em: 09 novembro 2018.

DAS, A. **Introduction to Q-Learning**. 2017. Disponível em: <https://towardsdatascience.com/introduction-to-q-learning-88d1c4f2b49c>. Acesso em: 06 junho 2018.

DOCS, M. Como escolher algoritmos de aprendizado de máquina do microsoft azure. **Microsoft docs**, 2017.

DONG, X.; ZHOU, D.-X. Learning gradients by a gradient descent algorithm. **Mathematical Analysis and Applications**, City University of Hong Kong, Hong Kong, China, 2008.

EDEN, T.; KNITTEL, A.; VAN UFFELEN, R. **Reinforcement Learning**. 2002. Disponível em: <https://towardsdatascience.com/introduction-to-q-learning-88d1c4f2b49c>. Acesso em: 01 junho 2018.

FACURE, M. **TensorFlow Detalhado**. 2017. Disponível em: <https://matheusfacure.github.io/2017/06/10/tf-detalhes/> Acesso em: 28 maio 2018.

FOORD, M. **urllib — URL handling modules**. 2018. Disponível em: <https://docs.python.org/3.5/library/urllib.html>. Acesso em: 01 novembro 2018.

GIBSON, E. et al. Niftynet: a deep-learning platform for medical imaging. **Elsevier Inc**, University College London, Londres, Inglaterra, 2018.

GUO, k.; WU, S.; XU, Y. Face recognition using both visible light image and near-infrared image and a deep network. **Elsevier Journal**, Shenzhen, China, v. 2, p. 39–47, 2017.

HAYKIN, S. **Redes neurais princípios e prática**. 2. ed. Porto Alegre, RS, Brasil: Bookman Companhia, 2001.

HAYKIN, S. **Neural networks and learning machines**. McMaster University, Canadá: Pearson Prentice Hall, 2008.

HOWARD, A. et al. Mobilenets: efficient convolutional neural networks for mobile vision applications. **arXiv:1704.04861**,

KHLEBOVICH, P. **Haar feature-based cascade classifier for object detection**. 2017. Disponível em: <https://ip-webcam.br.uptodown.com/android>. Acesso em: 08 novembro 2018.

LI, S. Z.; JAIN, A. K. Handbook of face recognition. **Springer Science and Business Media, Inc.**, China, 2005.

MAINI, V.; SABRI, S. **Machine learning for humans**. <https://medium.com/machine-learning-for-humans/why-machine-learning-matters-6164faf1df12> Acesso em: 13 maio 2018.

MARTíNEZ, R. A. A framework for designing the architectures of deep convolutional neural networks. **Entropy**, University of Bridgeport, CT, Estados Unidos, 2017.

MONARD, M. C.; BARANAUSKAS, J. A. Conceitos sobre aprendizado de máquina. In: **Sistemas inteligentes fundamentos e aplicações**. 1. ed. Barueri, SP, Brasil: Manole Ltda, 2003.

MONTAVON, G.; SAMEK, W.; MüLLER, K.-R. Methods for interpreting and understanding deep neural networks. **Elsevier Inc**, Technische Universität Berlin, Berlim, Alemanha, 2017.

MORDVINTSEV, A.; K., A. **Welcome to opencv-python tutorials's documentation!** 2013. Disponível em: <https://opencv-python-tutroals.readthedocs.io/en/latest/>. Acesso em: 06 novembro 2018.

MORDVINTSEV, A.; K., A. **Haar feature-based cascade classifier for object detection**. 2018. Disponível em: <https://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html>. Acesso em: 06 novembro 2018.

NIELSEN, M. A. **Neural networks and deep learning**. Nova York, NY, Estados Unidos: Determination Press, 2015.

OBERMAIER, D. **MQTT v3.1.1 now an OASIS Standard**. 2014. Disponível em: <http://mqtt.org/2014/11/mqtt-v3-1-1-now-an-oasis-standard>. Acesso em: 09 novembro 2018.

PEREZ, R. Algoritmo backpropagation. **Programar, Edição 57 - Julho 2017**,

PISAREVSKY, V. **haarcascade_frontalface_alt2.xml**. 2013. Disponível em: <https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade_frontalface_alt2.xml>. Acesso em: 06 novembro 2018.

PISAREVSKY, V. **haarcascade_profileface.xml**. 2013. Disponível em: <https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade_profileface.xml>. Acesso em: 06 novembro 2018.

PISAREVSKY, V. **haarcascade_frontalface_alt_tree.xml**. 2013. Disponível em: <https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade_frontalface_alt_tree.xml>. Acesso em: 06 novembro 2018.

POOLE, D.; MACKWORTH, A. **Artificial intelligence**. Nova York, NY, Estados Unidos: Cambridge University Press, 2010.

POOLE, D.; MACKWORTH, A. **Artificial intelligence 2e foundations of computational agents**. 2. ed. Nova York, NY, Estados Unidos: Cambridge University Press, 2017.

REIS, P.; RAVINDRA, S. Como as redes neurais convolucionais realizam o reconhecimento de imagem. **InfoQ Brasil**, São Paulo, SP, Brasil, 2017.

REVISTABW. Aprendizado de máquina. **Revista Brasileira de Web: Tecnologia**, 2015.Disponível em: <http://www.revistabw.com.br/revistabw/aprendizagem-computacional/>. Acesso em: jun. 2018.

REVISTABW. Aprendizado de máquina: aprendizado supervisionado. **Revista Brasileira de Web**, 2017. Disponível em: <http://www.revistabw.com.br/revistabw/aprendizagem-de-maquina-aprendizado-supervisionado/>. Acesso em: jun. 2018.

RUBASHKIN, M. **Getting Started with Deep Learning**. 2017. Disponível em: <https://www.kdnuggets.com/2017/03/getting-started-deep-learning.html> Acesso em: 01 junho 2018.

RUSSELL, S. J.; NORVIG, P. Artificial intelligence: a modern approach. , Upper Saddle River, Nova Jersey, Estados Unidos, 2003.

SUZIN, M. **Ambient intelligence na educação**: desenvolvimento de uma sala de aula inteligente utilizando iot. 2018. 42 p. Graduação em Ciências da Computação — Universidade de Caxias do Sul, Caxias do Sul, RS, Brasil, 2018.

SZEGEDY, C. et al. Rethinking the inception architecture for computer vision. **arXiv:1512.00567**, San Francisco, CA, Estados Unidos, 2015.

SZEGEDY, C. et al. Rethinking the inception architecture for computer vision. **arXiv:1512.00567v3 [cs.CV] 11 Dec 2015**, San Francisco, CA, Estados Unidos, 2015.

SZEGEDY, C. et al. Inception-v4, inception-resnet and the impact of residual connections on learning. **arXiv:1602.07261**, San Francisco, CA, Estados Unidos, 2016.

WEST, J. D. **A brief history of face recognition**. 2017. Disponível em: <https://www.facefirst.com/blog/brief-history-of-face-recognition-software/>. Acesso em: 01 abril 2018.

XIAO, Y. et al. Recognition of facial expressions using 2d dct and neural network. ,

XIMENES, L. Reconhecimento de faces: aplicação de algoritmos de redes neurais. , Campinas, SP, Brasil, 2011. Disponível em: <http://www.dca.fee.unicamp.br/ gudwin/courses/IA889/2011/IA889-07.pdf>. Acesso em: ago. 2018.

ZEILER, M.; FERGUS, R. Visualizing and understanding convolutional networks. **European Conference on Computer Vision - ECCV2014**, Nova York, NY, Estados Unidos, p. 818–833, 2014.

ÇARıKçı, M.; öZEN, F. A face recognition system based on eigenfaces method. **Procedia Technology**, Haliç University, Istanbul, Turquia, 2011.