CURSO CIÊNCIA DA COMPUTAÇÃO

Trabalho de Conclusão de Curso

UMA SOLUÇÃO PARA SISTEMAS DE LOGIN BASEADA NO RECONHECIMENTO FACIAL

Patterson Antonio da Silva Junior – CA: 14055899 – pattersonjunior@gmail.com

Orientador: Mauricio Wieler Orellana

São Paulo

2018

CURSO CIÊNCIA DA COMPUTAÇÃO

Trabalho de Conclusão de Curso

UMA SOLUÇÃO PARA SISTEMAS DE LOGIN BASEADA NO RECONHECIMENTO FACIAL

Projeto referente ao Trabalho de Conclusão de Curso apresentado à Universidade Cidade de São Paulo, como exigência parcial estabelecida para a obtenção do título de bacharel em Ciência da Computação.

Professor Orientador: Mauricio Wieler Orellana

São Paulo
2018

Patterson Antonio da Silva Junior – CA: 14055899 – pattersonjunior@gmail.com

UMA SOLUÇÃO PARA SISTEMAS DE LOGIN BASEADA NO RECONHECIMENTO FACIAL

Trabalho de Conclusão de Curso apresentado à Universidade Cidade de São Paulo como exigência parcial para obtenção do título de Bacharel em Ciência da Computação.

Trabalho aprovado. São Paulo, DATA DA APROVAÇÃO:

_____

Orientador

_____

Professor
Convidado 1

_____

Professor
Convidado 2

São Paulo

2018

Este trabalho é dedicado a meus pais por todo o incentivo e ajuda para que isso fosse possível.

AGRADECIMENTOS

A esta universidade, seu corpo docente, direção e administração que oportunizaram a janela que hoje me permite vislumbrar um horizonte superior, eivado pela acendrada confiança no mérito e ética aqui presentes.

A todas as pessoas que de uma alguma forma me ajudaram a acreditar em mim, quero deixar um agradecimento eterno, porque sem elas não teria sido possível.

“A inteligência artificial é a nova eletricidade”

(Andrew Yan-Tak Ng)

RESUMO

Este trabalho apresenta o estudo e o desenvolvimento de um sistema de verificação e reconhecimento facial, que recebe como entrada uma imagem de um rosto humano e retorna uma saída positiva caso o rosto na imagem de entrada seja de uma pessoa registrada em uma base de dados, ou uma saída negativa caso a pessoa não exista na base de dados. O projeto do sistema consiste em detectar a face de uma pessoa numa imagem capturada por uma câmera e reconhecer a pessoa na imagem caso uma imagem da face desta pessoa esteja previamente cadastrada na base de dados. A detecção facial foi implementada através da abordagem de Redes Neurais Convolucionais, que são uma variação de Redes Neurais Profundas mais apropriadas para lidar com imagens e detecção facial, pois extraem características das imagens, como os traços do rosto, e descartam áreas desnecessárias da imagem o que faz com que este método seja mais rápido e mais preciso em relação a um modelo de rede neural comum. A métrica usada para o reconhecimento facial baseia-se na distância entre as imagens, ou seja, no quanto a imagem de uma face humana difere de uma outra, isso comparando as características comuns entre as duas imagens e não comparando-as pixel a pixel. Para calcular a distância entre duas imagens de faces humanas usou-se o método *One-Shot Learning* com Redes Neurais Siamesas. Este método consiste em receber duas imagens de faces humanas e propagá-las uma única vez através das camadas da rede siamesa, as características das imagens são comparadas na ultima camada da rede siamesa.

Palavras-chave: Reconhecimento Facial; Verificação Facial; Rede Neural; Inteligência Artificial; Desenvolvimento; Sistema; Python; Aprendizagem de Máquina; Aprendizagem Profunda, Redes Convolucionais; Redes Siamesas.

ABSTRACT

This work presents the study and development of a facial recognition and verification system that receives as input an image of a human face and returns a positive output if the face in the input image is from a person registered in a database, or a negative output if the person does not exist in the database. The system design consists of detecting the face of a person in an image captured by a camera and recognizing the person in the image if an image of that person's face is previously registered in the database. Face detection was implemented through the Convolutional Neural Networks approach, which is a variation of Neural Networks most appropriate to deal with images and facial detection, as they extract features of the images, such as facial features, and discard unnecessary areas of the image. which makes this method faster and more accurate than a common neural network model. The metric used for facial recognition is based on the distance between the images, that is, how much the image of a human face differs from another one, that is, comparing the common characteristics between the two images instead of comparing them pixel by pixel. To calculate the distance between two images of human faces, the One-Shot Learning method with Siamese Neural Networks was used. This method consists of receiving two images of human faces and propagating them once through the layers of the Siamese network, then the characteristics of the images are compared in the last layer of the Siamese network.

Keywords: Facial Recognition; Facial Verification; Neural Network; Artificial Intelligence; Development; System; Python; Machine Learning; Deep Learning, Convolutional Networks; Siamese Networks.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

As câmeras de vigilância convencionais são úteis apenas quando alguém está assistindo o conteúdo capturado por elas. Apesar de termos a sensação de que essas câmeras são como olhos digitais que nos assistem ou assistem por nós, as câmeras de segurança, em sua maioria, apenas providenciam evidências caso algo der errado. É preciso um humano assistindo, muitas vezes, diversos monitores (VINCENT, 2018).

Mas segundo Vincent (2018) a Inteligência Artificial está mudando rapidamente essa realidade, está dando às câmeras digitais a capacidade de analisar vídeos sem a intervenção humana. Isso ajuda policiais a detectar crimes e acidentes – além de possuir uma série de aplicações industriais e científicas.

As questões de segurança e segurança pública influenciam profundamente as formas como as cidades são projetadas. A obsessão com a segurança foi reivindicada como sendo a narrativa mestra do design urbano contemporâneo (KOSKELA, 2002).

O medo que as pessoas têm em andar nas ruas tem aumentado drasticamente devido ao constante aumento da violência e falta de segurança que vem crescendo nos últimos anos. A obsessão pela segurança foi reforçada recentemente, quando os atentados terroristas de 11 de setembro de 2001 nos Estados Unidos provocaram uma inclinação para o fim da segurança pública do espectro de privacidade/segurança e uma centralização do poder em muitas sociedades ocidentais (GRAY, 2003).

Pode-se pensar em instalar recursos de monitoramento por câmeras em diversas áreas de uma cidade com a finalidade de melhorar a segurança, no entanto isto não é suficiente para resolver o problema. Melhor que instalar câmeras para auxiliar no monitoramento de pessoas seria programá-las para reconhecer suas faces, esse processo automatizado ajudaria a distinguir mais

rapidamente possíveis causadores de tumultos, foragidos da justiça e suspeitos de terrorismo.

Sistemas de reconhecimento facial são ferramentas para detecção e seleção de pessoas suspeitas de agirem perigosamente no ambiente urbano. Ao mesmo tempo eles reduzem o medo através da coleta de informações (GRAY, 2003).

Os sistemas de reconhecimento facial também têm se mostrado uma boa solução no auxílio à busca de pessoas desaparecidas. Na China, em virtude de um sistema online de reconhecimento de imagens, pessoas desaparecidas têm sido encontradas, com a contribuição desta aplicação muitas pessoas estão localizando parentes perdidos. Segundo Lo (2017), o sistema compara com uma base de dados, os retratos enviados por parentes que estão buscando por seus filhos perdidos e ou seus pais.

Os sistemas de reconhecimento facial poderão classificar áreas como sendo de risco e até mesmo alertar sobre regiões perigosas permitindo que as pessoas possam evitar tais locais assim precavendo-as de possíveis ameaças como assalto, sequestro ou até assassinato. Isso também permite que pessoas se sintam seguras ao saber quando estão em ambientes classificados como seguros ou mais precavidas em ambientes considerados como inseguros (GRAY, 2003).

Para Gray (2003), os sistemas ampliam a visão e a conscientização, com a esperança de que as câmeras estarão aptas a detectar uma ameaça antes que ela se torne realidade.

Há também a questão da autenticação de usuários em sistemas, hoje em dia, devido à constante ameaça de hackers que usam diversos tipos de algoritmos para desvendar senhas. As senhas foram feitas para ser algo que um indivíduo possa lembrar facilmente, mas ao mesmo tempo não ser fácil de ser adivinhada ou quebrada. É onde o problema reside, pois, as pessoas tendem a escolher senhas fáceis de lembrar, o que as tornam fáceis de adivinhar (MACÊDO, 2017).

Estes ataques funcionam porque muitos usuários de computadores e empresas insistem em usar palavras comuns como senhas (ROUSE, 2005).

Devido a essas ameaças nos vemos obrigados a criar senhas cada vez mais complexas. Segundo Fetter (2013), hoje, quando nos cadastramos em algum serviço, somos orientados a utilizar um número mínimo de caracteres. Muitos mecanismos, inclusive, não aceitam senhas com menos de oito caracteres. Além do comprimento, a inserção de caracteres maiúsculos e minúsculos, números e símbolos são constantemente recomendados, pois permitem o aumento de a segurança da senha.

O problema é que desta forma torna-se cada vez mais complicado e enfadonho criar senhas que sejam robustas e de fácil memorização ao mesmo tempo e, além disso, sempre que uma senha é violada ou quando o usuário esquece a senha, ele deve criar uma nova e assim ele passa novamente pelo aborrecimento de criar outra senha complexa e, para piorar, nem mesmo a senha mais forte é invulnerável, se um banco de dados com as senhas de todos os usuários de um serviço for vazado, as senhas poderão cair em mãos erradas não importando sua complexidade (HAMANN, 2017).

Toda essa dificuldade acaba levando os usuários a criar senhas com pouca variação e quantidade de caracteres para que sejam fáceis de memorizar, o que as torna alvos fáceis de possíveis invasores. Por conta disso, os usuários também costumam usar a mesma senha para vários sites, Segundo Boyde (2006), isso é um risco, visto que uma vez que alguém descobre a senha de um site, pode obter acesso automaticamente a diversos outros.

Segundo Adams e Sasse (1999), muitos usuários contornam tais restrições para produzir senhas que eles achem fáceis de lembrar. Isso é um perigo visto que o crescente poder de processamento dos computadores facilita que hackers possam “quebrar” essas senhas.

Segundo Canaltech (2017), graças ao *deep learning*, a tecnologia do reconhecimento facial já está bastante avançada, sendo capaz de identificar faces com clareza mesmo em condições de baixa iluminação, ou sob ângulos tortos. E o mercado já está tão confiante nessa tecnologia que, na China,

alguns bancos já a estão usando para conceder empréstimos, enquanto há, em algumas estações de trens, a possibilidade de pagar pela passagem usando esse tipo de biometria em vez de pagar com dinheiro ou cartões.

Com base nos fatos apresentados percebe-se que autenticação de usuários por meio do reconhecimento facial como alternativa à autenticação por senhas tradicionais tem se mostrado uma forte tendência para os próximos anos e grandes empresas já estão investindo fortemente nesta tecnologia. Entre as principais novidades da Apple para os novos *iPhones* estão o fim do botão de iniciar e o desbloqueio da tela por meio de reconhecimento facial (DW, 2017).

A empresa de pagamentos Mastercard já está testando no Brasil uma nova solução para facilitar a vida do usuário quanto aos pagamentos digitais, trata-se da tecnologia *Mastercard Identity Check* que é baseada em biometria e reconhecimento facial e já existe em 14 países, funciona através de *smartphones*, *tablets* e computadores conectados à internet (CANALTECH, 2016).

O uso da tecnologia de reconhecimento facial também se aplica no combate a roubos de veículos, sabendo que uso de veículos é necessário para todos atualmente e que as pessoas gastam muito com seguros, uma tecnologia que dificulte os roubos se mostra muito importante se possibilitar que o dono do veículo possa, por si só, prevenir o roubo. Através de sistemas de reconhecimento facial a prevenção do furto de veículos pode ser feita remotamente por uma pessoa autorizada (AGRAWAL; LADHAKE, 2016).

A autenticação de usuários também pode ser utilizada para reconhecer condutores de veículos, podendo o próprio carro, em caso de furto, mandar uma foto do condutor para o *smartphone* do proprietário, assim o dono poderia usar o próprio *smartphone* para desativar as funções do motor, o veículo também poderia ser ligado através do reconhecimento facial bastando reconhecer a face do seu proprietário primeiro para, na sequência, permitir que seja dada a partida com a chave.

Diante do exposto, um trabalho que demonstre o desenvolvimento de um software que permita o reconhecimento de faces humanas em câmeras de

segurança, que seja implementado através de algoritmos de inteligência artificial voltados para reconhecimento de padrões em imagens, e que seja capaz de comparar as faces identificadas com as contidas em uma base de dados e que possa ser usado na autenticação de usuários em sistemas, se justifica por oferecer enorme relevância para o contexto em que se aplica.

1.1 OBJETIVOS

Este trabalho tem como principal objetivo implementar uma solução tecnológica dedicada ao reconhecimento facial usando a abordagem das redes neurais de aprendizagem profunda.

Visando alcançar o objetivo principal, alguns objetivos específicos se mostram necessários, são eles:

- Realizar um estudo sobre os algoritmos e técnicas de Aprendizagem Profunda como Redes Convolucionais para reconhecimento facial.
- Implementar um sistema de reconhecimento facial usando a abordagem da Aprendizagem Profunda.
- Escrever sobre os resultados obtidos.

2 REVISÃO BIBLIOGRÁFICA

Com o intuito de dar uma maior compreensão às ideias que serão discutidas neste artigo e uma correta fundamentação dos conceitos envolvidos na elaboração da solução proposta, serão abordados na sequência, temas importantes que darão sustentação ao presente trabalho.

2.1 INTRODUÇÃO À INTELIGÊNCIA ARTIFICIAL

Por muito tempo a Inteligência Artificial foi, para a maioria das pessoas, tema de filmes de ficção, algo futurista e distante da realidade, e, de fato foi assim por muito tempo, mas atualmente, com o aumento do poder de processamento dos computadores e a quantidade de informação disponível e também os avanços nas pesquisas neste campo, a IA tem se tornado cada vez mais uma realidade nos últimos anos e, praticamente, já faz parte das nossas vidas.

RUSSEL e NORVING (1995, p. 3, tradução nossa) enfatizam dois motivos para se estudar IA:

> A Inteligência Artificial busca entender entidades inteligentes. Assim, uma das razões para estudá-la é aprender mais sobre nós mesmos. Mas, ao contrário da filosofia e da psicologia, que também estão preocupados com a inteligência, a IA se esforça em construir entidades inteligentes bem como compreendê-las. Outra razão para estudar IA é que as entidades inteligentes construídas são úteis e interessantes por si só, tanto que a IA já produziu muitos produtos importantes e impressionantes, mesmo na fase inicial do seu desenvolvimento. Apesar de não podermos prever o futuro, é evidente que computadores com inteligência no nível humano (ou melhor) causariam um enorme impacto no nosso cotidiano e no futuro da civilização.

De acordo com Russel e Norving (1995), a IA aborda um grande enigma, entender como um cérebro, biológico ou eletrônico poderia perceber, compreender, prever e manipular um mundo muito maior e mais complexo do que ele, e como fazer algo com essas propriedades. São questões complicadas, mas ao contrário da busca de viagens mais rápidas do que a luz ou de um dispositivo antigravidade, o pesquisador da IA tem evidências sólidas

de que essa busca é possível. Tudo o que pesquisador precisa fazer é olhar para espelho para ver um exemplo de um sistema inteligente.

Pode-se pensar que todas as boas ideias já foram tomadas por cientistas do passado como Galileu, Newton, Einstein e muitos outros, e que levaria muitos anos de estudo antes que alguém pudesse contribuir com novas ideias. Na IA, por outro lado, ainda há aberturas para um possível Einstein em tempo integral devido ao fato de que a IA ainda é considerada uma disciplina recente já que foi iniciada formalmente em 1956, o que sugere que ainda há muito o que ser descoberto (RUSSEL; NORVING, 1995).

A IA atualmente engloba uma grande variedade de subcampos, de áreas de propósito geral, como percepção e raciocínio lógico a tarefas específicas, como jogar xadrez, provar teoremas matemáticos, escrever poesia e diagnosticar doenças. Muitas vezes, cientistas de outras áreas vêm para a IA, onde eles encontram meios de sistematizar e automatizar tarefas intelectuais as quais eles trabalharam durante toda sua vida. Similarmente, os que trabalham na IA podem escolher aplicar os seus métodos em qualquer área de esforço intelectual humano (RUSSEL; NORVING, 1995).

2.1.1 As Definições de Inteligência Artificial

Podemos ver o porquê da IA ser um campo importante e entusiasmante, mas o que é IA exatamente, como defini-la?

Pode-se dizer que Inteligência Artificial, em uma definição moderna, é “o estudo e design de agentes inteligentes” sendo que um agente inteligente é um sistema capaz de perceber seu ambiente e tomar ações as quais maximizem suas chances de sucesso (SCIENCEDAILY, 2018).

A IA não trata apenas de simular o pensamento humano, por um lado pode-se aprender algo sobre como fazer máquinas resolverem problemas observando as pessoas ou seus métodos, por outro lado, muito do trabalho da IA envolve estudar os problemas que o mundo apresenta para a inteligência em vez de estudar pessoas ou animais. Os pesquisadores de IA são livres para

usar métodos que não são observados em pessoas ou que envolvam muito mais computação do que as pessoas possam fazer (MCCARTHY, 2007).

2.1.2 IA fraca e IA forte

Há diversas abordagens de estudo na Inteligência Artificial, duas vertentes teóricas que se destacam são a Inteligência Artificial Forte e Inteligência Artificial Fraca.

A Inteligência Artificial Fraca diz respeito à construção de máquinas ou softwares de certa forma inteligentes, porém, eles não são capazes de raciocinar por si próprios. São sistemas focados em realizar tarefas específicas, respeitam regras encadeadas e tomam decisões analisando múltiplas condições do tipo *se-então*. Assim, nesse caso, a máquina não raciocina de verdade, pois ela precisa que especialistas humanos forneçam o conhecimento para que o software possa executar e tomar decisões. Portanto, a IA Fraca apenas simula inteligência, porém, não é de fato inteligente. (GRANATYR, 2017).

Já a Inteligência Artificial Forte está focada em criar máquinas autoconscientes, que não apenas simulem raciocínio, mas que possam pensar. Dessa forma, uma máquina ao escrever uma poesia, por exemplo, deveria ter consciência do que escreveu e não apenas organizar as palavras a fim de formar frases coerentes. Ou seja, a máquina saberia a razão que a levou a manipular tais símbolos e, possivelmente, poderia ter pensado ou demonstrado emoções. Portanto, na visão da IA Forte, o software precisaria ter a consciência do que escreveu (GRANATYR, 2017).

2.1.3 Uma Breve História da Inteligência Artificial

Foi em 1956, quando o termo “Inteligência Artificial” foi cunhado pelo cientista da computação John McCarthy (RUSSEL; NORVING, 1995).

Apesar de estar em alta atualmente, a IA não é um novo campo de estudo. Se nós ignorarmos o caminho de raciocínio puro filosófico que vai do Grécia antiga até Hobbes, Leibniz e Pascal, a IA, como sabemos, foi oficialmente iniciada em 1956 no *Dartmouth College*, onde os mais eminentes especialistas se reuniram para discutir ideias sobre a simulação da inteligência.

Tudo isso ocorreu somente alguns anos antes de que o escritor Isaac Asimov estabelecesse suas próprias três leis de robótica, porém, mais relevantemente após Turing propor um famoso artigo em 1950 onde, pela primeira vez, e ele propõe uma máquina pensante e um teste para avaliar se essa máquina mostra, de fato, alguma inteligência.

Quando o grupo de pesquisa da *Dartmouth* levou à público os conteúdos e as ideias surgidos da reunião de verão, um fluxo de financiamento do governo foi reservado para o estudo da criação de uma inteligência não-biológica (COREA, 2017).

No entanto, a IA não progrediu muito naquela época pois os computadores precisavam de hardware mais potente para poder guardar informações e executar dados complexos, além disso, os computadores da época eram caros e somente prestigiadas universidades e grandes companhias de tecnologia poderiam se dar ao luxo de se aventurar nessas “águas desconhecidas”.

É nos anos 80 que a IA recebe um novo impulso, devido ao aumento da capacidade dos de processamento e armazenamento dos computadores, novos investidores aparecem. John Hopfield e David Rumelhart popularizam o termo *deep learning* técnicas que permitiram que computadores aprendessem por experiência. (ANYOHA, 2017).

Em 1996 o computador da IBM, *Deep Blue* mostra que a IA avançou ao ponto de uma máquina ser capaz de enfrentar o maior campeão de xadrez da época, considerado o melhor jogador de todos os tempos, o russo Garry Kasparov, que ganhou de *Deep Blue* neste ano, mas, em 1997 após ser atualizado, Deep Blue foi capaz de derrotar Kasparov em um jogo de seis partidas (GREENEMEIER, 2017).

2.1.4 Alguns Avanços na inteligência artificial

Segundo Araújo (2017), muitas pessoas têm medo da Inteligência Artificial, as pessoas temem que as máquinas nos substituam, ou que possam se tornar mais inteligentes e nos dominar. Mas, apesar de muita gente temer a IA, ela tem se mostrado benéfica ao ser humano até o momento.

Araújo (2017), cita ainda que a Inteligência Artificial tem se mostrado benéfica ao ser humano em diversas áreas, por exemplo:

- A geração de diagnósticos bastando o paciente informar o que está sentindo onde um sistema de IA pode predizer diagnósticos e indicar medicações auxiliando os médicos, e também pode sugerir medicações com base na análise de exames clínicos anteriores.
- Na segurança da informação a IA pode bloquear ataques de hackers rapidamente, pois os mecanismos de IA estão aprendendo rapidamente como os hackers realizam os ataques e os bloqueia com bastante eficiência, além disso, cada novo ataque é uma nova experiência para a IA, de forma que quanto mais se ataca um computador, mais a IA aprende a defendê-lo.
- A IA está aprendendo com os hábitos e comportamentos humanos. Um exemplo são as redes sócias baseados em

> informações que já estão cadastradas e até mesmos nas postagens; um mecanismo de análise cognitiva e de inteligência por traz destas redes conseguem saber seu estado emocional, se seu estado político está mais aflorado, sua preferência de compras, viagens, inclusive analisam até fotos postadas com amigos ou marcadas por eles.

Araújo (2017), afirma que estes são alguns exemplos de como a IA está produzindo bons resultados e com isso torna a vida das pessoas melhor e resolve problemas de forma mais rápida e assertiva.

2.1.5 A presença da Inteligência Artificial nas nossas vidas

GUGELMIN (2016), afirma que a presença da Inteligência Artificial pode ser notada em diversas tecnologias. Para ele o *Deep Blue* é um exemplo clássico de aplicação da inteligência artificial, criado com a intenção de superar os melhores jogadores de xadrez.

Ele comenta que na rede de internet empresas como Google, Amazon possuem algoritmos de IA capazes de influenciarem os internautas na hora de fazer as compras virtuais.

Segundo ele, os assistentes de voz tais como *Siri*, *Cortana* e *Google Now* são exemplos de tecnologias desenvolvidas com algoritmos de IA. A Siri adquire mais informações ao analisar as decisões tomadas por seus usuários para que assim consiga prever antes mesmo de uma busca o que o usuário pretende consumir.

Esta mesma tecnologia atende milhares de solicitações que para cada resposta dada a Siri verifica qual foi o comportamento do usuário para que nas próximas solicitações traga resultados mais precisos.

O comum entre estas tecnologias é que elas aprendem com as experiências humanas. Narula (2018), lista em uma postagem eletrônica como

algumas companhias têm utilizado a Inteligência Artificial. O autor diz que o Google utiliza informações anônimas de localizações dos usuários para calcular a velocidades nas vias. Segundo o mesmo a companhia Waze em 2013 com a aquisição de dados advinda de usuários conseguiu incorporar informações sobre construções e acidentes de tráfico. Com as informações adquiridas os algoritmos puderam reduzir o tempo das viagens sugerindo as melhores rotas.

Diz ainda que o Uber é outro exemplo que utiliza *Machine Learning* para prever a demanda por viagens para garantir que não ocorra de aumentar os valores das viagens a fim de atender a demanda e ter que controlar a quantidade de motoristas. A respeito do uso comercial de voos, o autor cita o jornal *The New York Times*, o qual informa que o tempo estimado de um piloto Boeing é de 7 minutos, os quais são dedicados para a aterrissagem e decolagem e o tempo restante a aeronave permanece no piloto automático.

Narula (2018), cita que em um futuro não muito distante a Inteligência Artificial irá reduzir o tempo gasto entre as viagens utilizando-se dos carros autônomos, o que resultará numa redução de 90% dos acidentes, viagens compartilhadas reduzirão a quantidade de veículos nas rodovias acima dos 75%, e sistemas de sinalização de tráficos inteligentes reduzirão em até 40% o tempo de espera, as viagens em geral passarão por uma redução de 26% segundo uma pesquisa piloto.

O autor ressalta a importância do *Machine Learning* nas filtragens dos *Spams*, onde é empregado na sua mais alta capacidade. As regras simples de filtragem de *Spams* não são eficientes ao combate de *Spammers* devido estes atualizarem rapidamente as mensagens burlando assim as regras de filtragem. Todavia, os filtradores de Spam devem aprender uma vasta variedade de sinais, tais como palavras, metadados. Através de algoritmos de aprendizagem de máquina o Gmail consegue com sucesso filtrar 99.9% das mensagens *Spam*.

Escrevendo sobre plágio, Narula (2018), diz que historicamente, a detecção de plágio para textos regulares depende de uma base robusta de dados referentes a matérias a fim de serem comparados com os textos dos estudantes, por outro lado, ML pode ajudar a detectar o plágio de recursos que

não estão alocados em uma base de dados, mesmo que sejam baseados em outros idiomas ou mesmo que não tenham sido digitalizados. Duas pesquisas utilizaram ML para predizer, com 87% de precisão, quando um código estava sendo plagiado.

Ainda segundo a postagem de Narula, ao adicionamos fotos, a rede social Facebook ativa seu serviço automaticamente para destacar as faces e sugestiona os amigos aos quais podemos fazer identificações. Mas como esta identificação dos amigos é feita instantaneamente? O Facebook utiliza IA para reconhecer faces. A companhia tem investido fortemente na ML não apenas em sua rede social como também na compra de *startups* de reconhecimento facial como a Face.com e a Faciometrics. O autor diz que em 2016 o Facebook anunciou uma nova iniciativa em IA, o *DEEPTEXT*, uma engenharia que compreende texto, que as companhias declaram ser capaz de entender aproximadamente como os seres humanos, o contexto de milhares de postagens por segundo, abrangendo mais de 20 idiomas. O *DEEPTEXT* é utilizado no Facebook para detectar as intenções, por exemplo, permitindo que os usuários ao mencionarem “Eu preciso de um serviço de corrida” possam chamar uma viagem Uber, ao mesmo tempo em que a ferramenta pode distinguir quando não se trata de uma solicitação por um serviço de corrida tal como na expressão “Eu gosto de andar a burros”.

2.2 APRENDIZAGEM DE MÁQUINA

A aprendizagem de máquina (do inglês *Machine Learning*) é uma das áreas da ciência da computação que mais crescem. O termo aprendizagem de máquinas refere-se à detecção automatizada de padrões significantes em dados. (SHALEV-SHWARTZ; BEN-DAVID, 2014).

Para Marr (2016), a Aprendizagem de Máquina é uma aplicação da IA que se baseia na ideia de que podemos dar às máquinas acesso aos dados e deixá-las aprender por si mesmas.

Na definição de Mitchell (1997, pg. 2, tradução nossa) “Se diz que um programa de computador aprende pela experiência E com respeito a uma tarefa T e uma performance P, se sua performance nas tarefas em T, medidas por P, melhoram com a experiência E”.

Compartilhando tópicos comuns com áreas da matemática como estatística, teoria da informação e teoria dos jogos, a Aprendizagem de Máquina é naturalmente um campo da Ciência da Computação que tem o objetivo de programar máquinas capazes de aprender. A Aprendizagem de Máquina pode ser vista como um ramo da IA, uma vez que, afinal, a habilidade de transformar experiência em conhecimento ou detectar padrões significativos em dados sensoriais complexos é um ponto chave da inteligência humana e animal. (RUSSEL; NORVING, 1995).

Porém, Russel e Norving (1995), afirmam também que, deve-se notar que em contraste com a IA tradicional, a Aprendizagem de Máquina não tenta construir uma imitação automatizada de comportamento inteligente, mas sim usar a força e as habilidades especiais dos computadores para complementar a inteligência humana, muitas vezes executando tarefas que vão além das capacidades humanas. Por exemplo, a capacidade de digitalizar e processar enormes bancos de dados permite que os programas de aprendizagem de máquina detectem padrões que estão fora do alcance da percepção humana.

2.2.1 Tipos de Aprendizagem de Máquina

O aprendizado possui um domínio muito amplo, por isso o campo da Aprendizagem de Máquina se dividiu em vários subcampos para lidar com diferentes tipos de aprendizagem. Dois dos principais ramos de aprendizagem são a Aprendizagem supervisionada (*Supervised Learning*) e a Aprendizagem não-supervisionada (*Unsupervised Learning*).

2.2.1.1 Aprendizagem Supervisionada

Nessa abordagem de aprendizagem, a aprendizagem supervisionada descreve um cenário onde a experiência (um exemplo de treino), contém informações significantes (por exemplo, uma imagem rotulada como sendo ou não um rosto humano) que não estejam presentes nos exemplos de teste nos quais a experiência obtida deve ser aplicada. Assim, a experiência obtida é usada para predizer tal informação ausente nos exemplos de teste. Podemos então dizer que a máquina adivinha se uma imagem – por exemplo – contém ou não o rosto de uma pessoa com base nos exemplos de treino que ela usou para aprender (SHALEV-SHWARTZ; BEN-DAVID, 2014, p. 23).

2.2.1.2 Aprendizagem Não Supervisionada

Já na aprendizagem não-supervisionada, não há distinção entre os exemplos de treino e os exemplos de teste. A máquina processa os dados de entrada com o objetivo de gerar um resumo ou versão compactada desses dados. Ela agrupa um conjunto de dados em subconjuntos de objetos semelhantes entre si. É como dizer que a máquina “olha” para o conjunto de dados e seleciona os mais semelhantes entre e os agrupa em grupos distintos (SHALEV-SHWARTZ; BEN-DAVID, 2014, p. 23).

2.2.2 Aprendizagem Profunda

Apesar de já existir a bastante tempo, a Aprendizagem Profunda vem ganhando fama recentemente graças ao aumento do poder de processamento dos computadores e a grande quantidade de informação que temos disponível hoje, antigamente as máquinas não tinham o poder de processamento necessário para processar essa grande quantidade de dados. Esta tecnologia está por trás do funcionamento de carros autômatos, permitindo que eles

reconheçam placas de “pare”, ou distingam um pedestre de um poste, por exemplo, e é também, a tecnologia chave por trás do controle de voz em diversos dispositivos (MATH WORKS, 2018).

Um modelo computacional pode, na aprendizagem profunda, executar tarefas de classificação diretamente de imagens, texto ou som. Os modelos de aprendizagem profunda são treinados usando uma grande quantidade de dados rotulados e arquiteturas de redes neurais artificiais que possuem muitas camadas (MATH WORKS, 2018).

2.2.2.1 O que é Aprendizagem Profunda?

Aprendizagem profunda é uma técnica de aprendizagem de máquina onde os computadores são ensinados a fazer o que é comum à seres humanos, como aprender a partir de exemplos (MATHWORKS, 2018).

Apesar do termo ser chamativo, Aprendizagem Profunda (do inglês *Deep Learning*), é realmente um termo que descreve certos tipos de redes neurais artificiais e algoritmos relacionados que consomem dados de entrada geralmente muito brutos. Esses dados de entrada são processados pelos algoritmos, através de muitas camadas de transformação não-lineares que calculam uma saída de destino (CASTROUNIS, 2016).

Um algoritmo de Aprendizagem Profunda baseado no modelo de redes neurais, possui um número de camadas maior que os chamados Algoritmos de Aprendizagem Rasos (*Shallow Learning Algorithms*), que tendem a ser menos complexos (CASTROUNIS, 2016).

Em redes neurais profundas, cada camada é um conjunto de neurônios (também chamados de unidades), cada uma dessas camadas treina em um conjunto distinto de características (*features*) baseadas na saída da camada anterior. Conforme a rede neural avança (*feed forward*), mais complexas são as características que os neurônios podem reconhecer, e eles fazem isso

agregando e recombinando características das camadas anteriores (DEEPLEARNING4J, 2017).

Essa hierarquia de características é de uma complexidade e abstração crescentes. Isso torna as redes neurais profundas capazes de lidar com grandes conjuntos de dados multidimensionais podendo receber bilhões de parâmetros que passam por funções não-lineares (DEEPLEARNING4J, 2017).

Além disso, as redes neurais profundas conseguem lidar com dados não-estruturados, ou seja, imagens, vídeo, áudio e textos, que são a maior parte dos dados existentes no mundo (DEEPLEARNING4J, 2017).

A aprendizagem profunda pode, por exemplo, receber um milhão de imagens e classificá-las em *clusters* (grupos) de acordo com suas similaridades. Essa é a base para os chamados álbuns de foto inteligentes (DEEPLEARNING4J, 2017).

As redes Neurais profundas terminam em uma camada de saída (*output layer*), um classificador logístico que atribui uma probabilidade para um resultado ou rótulo. Isso é o que chamamos de predição. Um uma imagem pode ser classificada pela rede neural como tendo a probabilidade de 90% de ser uma pessoa (DEEPLEARNING4J, 2017).

2.3 REDES NEURAIS ARTIFICIAIS

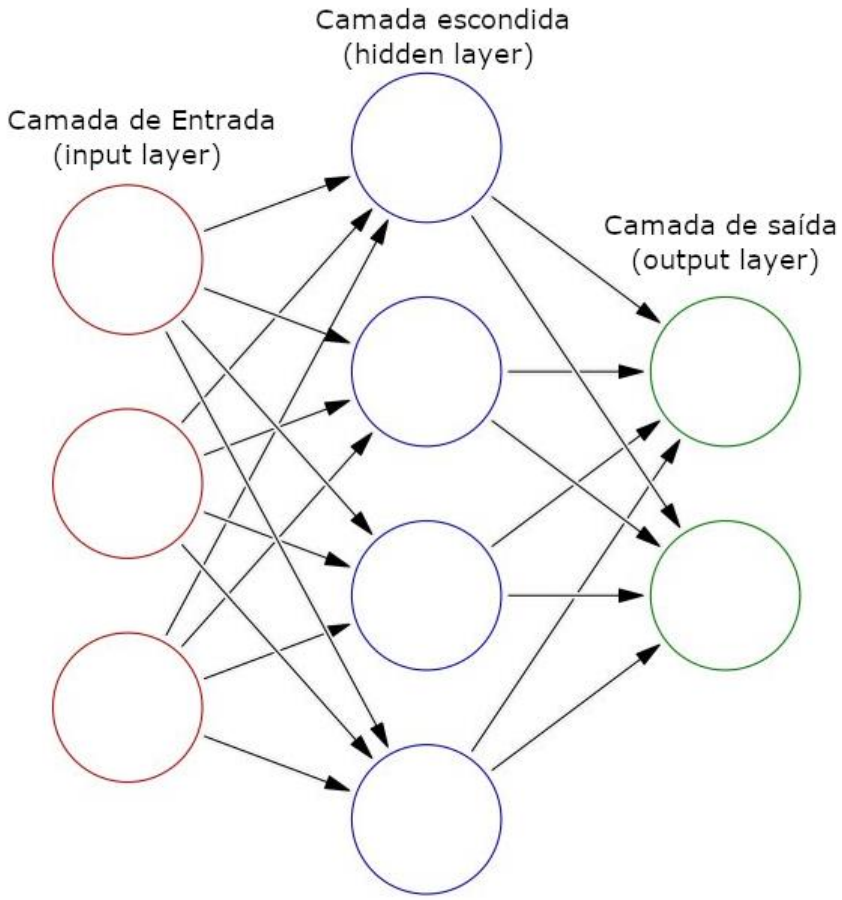
Para entender melhor o que é a Aprendizagem Profunda e como ela funciona precisamos, entender o conceito de Redes Neurais Artificiais.

Segundo Russel e Norving (1995), o aprendizado com redes neurais foi proposto em meados do século 20. Ele produz um paradigma de aprendizado efetivo e recentemente tem se mostrado capaz de alcançar desempenho de ponta em várias tarefas de aprendizagem. Uma rede neural artificial é um modelo de computação inspirado na estrutura das redes neurais do cérebro. Em modelos simplificados do cérebro, consiste em um grande número de dispositivos de computação básicos (neurônios) conectados entre si em uma

rede de comunicação complexa, através da qual o cérebro é capaz de realizar cálculos altamente complexos. As redes neurais artificiais são construções de computação formais que são modeladas após este paradigma de computação.

De acordo com Russel e Norving (1995), uma rede neural pode ser descrita como um grafo direcionado cujos nós correspondem a neurônios e as arestas correspondem às ligações entre eles. Cada neurônio recebe como entrada uma soma ponderada das saídas dos neurônios conectados às bordas de entrada.

Figura 1 - Exemplo de uma Rede Neural

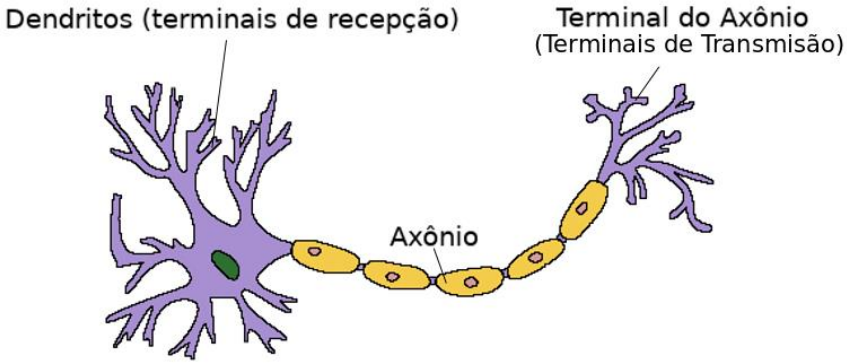


Fonte: Portilla (2016)

2.3.1 Neurônios Artificiais

Os neurônios biológicos, apesar de serem células de extrema complexidade, possuem uma natureza computacional essencial em termos de entrada e saída que é relativamente direta. Possuindo múltiplos dendritos (receptores de estímulos nervosos) e apenas um axônio (responsável por transmitir impulsos nervosos para a as células seguintes). Um neurônio recebe as entradas através dos seus dendritos e transmite uma saída através de seu axônio. As entradas e saídas assumem a forma de impulsos elétricos. O neurônio então resume suas entradas e se o total de impulsos elétricos exceder seu limite de disparo, ou seja, se for ativado, um novo impulso é disparado pelo seu único axônio. Por fim, o axônio distribui o sinal através de suas sinapses ramificadas que, em conjunto, alcançam milhares de neurônios adjacentes (ROELL, 2017).

Figura 2- Neurônio Biológico

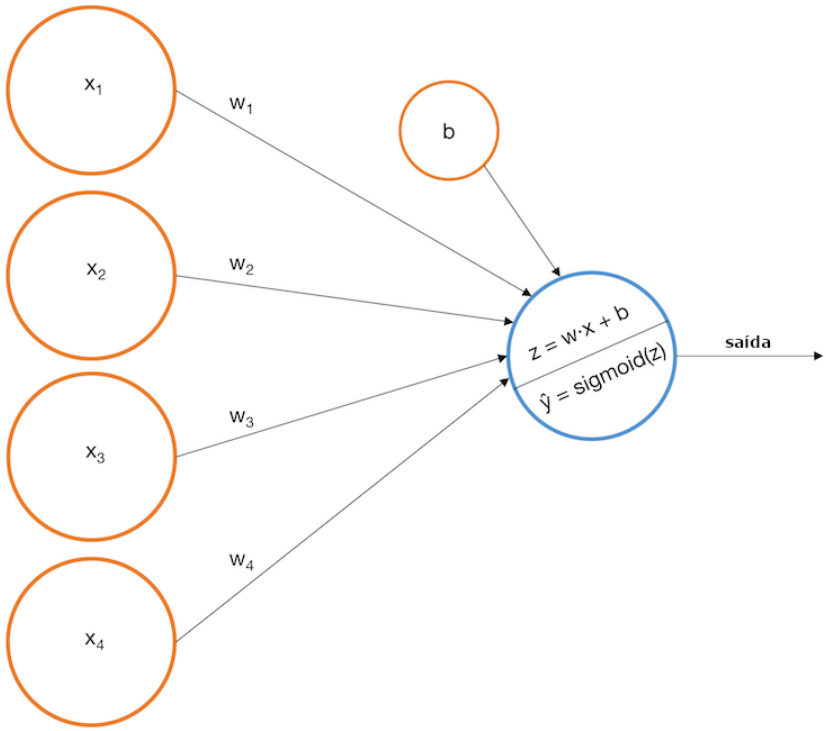


Fonte: O Autor

Como dissemos anteriormente, os neurônios artificiais são a base da construção de uma rede neural artificial. Além disso, conforme Berger (2016), eles não apenas recebem seu nome inspirados em suas contrapartes

biológicas, mas também são modelados baseando-se no comportamento dos neurônios do nosso cérebro.

Figura 3 - Neurônio Artificial



Fonte: Gorman (2017)

Da mesmam forma que um neurônio biológico possui dendritos para receber sinais, um corpo celular que os processa e um axônio para distribuir sinais para os outros neurônios, um neurônio artificial possui diversos canais de entrada, uma faze de processamento de dados e uma saída que pode se espalhar para vários outros neurônios artificiais BERGER (2016).

As conexões entre os neurônios podem se tornar mais fortes ou menos fortes. Conforme Berger (2016), as entradas (ou sinais) que o neurônio artificial recebe podem ser aumentadas ou diminuídas devido ao fato de que cada entrada é multiplicada por um valor de peso. Isso significa que, para um neurônio com três entradas, por exemplo, ele terá três pesos que podem ser ajustados individualmente. A rede neural pode ajustar esses pesos no decorrer da fase de aprendizagem com base no erro do último resultado do teste.

Como falamos anteriormente, o neurônio resume os sinais de entrada recebidos. Dessa forma, segundo Berger (2016), em cada fase da rede neural, os sinais de entrada modificados são resumidos para um único valor. Em cada etapa também é adicionado um valor de deslocamento à soma. Esse deslocamento é chamado de viés (*bias*, em inglês), e esse viés também é ajustado pela rede neural durante a aprendizagem.

É aí que está o “pulo do gato”. Berger (2016) afirma que, inicialmente, cada neurônio é inicializado com pesos e vieses aleatórios. E esses pesos e vieses são gradualmente atualizados em cada iteração de aprendizagem, de forma que os próximos resultados se aproximam cada vez mais da saída desejada. E assim a rede neural aprende gradualmente até chegar a um estado onde os padrões desejados estejam "aprendidos". Por fim, o resultado do cálculo do neurônio é convertido em um sinal de saída. Isso ocorre através do alinhamento do resultado com o uso de uma função de ativação (BERGER, 2016).

2.3.2 A Função de Ativação

O que a função de ativação faz é limitar a amplitude do valor de saída do neurônio, ela mapeia valor para um intervalo entre 0 até 1, ou -1 até 1, etc. (dependendo do tipo de função de ativação usado). As funções de ativação podem ser lineares ou não-lineares. No caso das redes neurais, são usadas as funções não lineares, pois assim o modelo da rede neural pode se adaptar a dados não lineares (HAYKIN, 2008, p. 10).

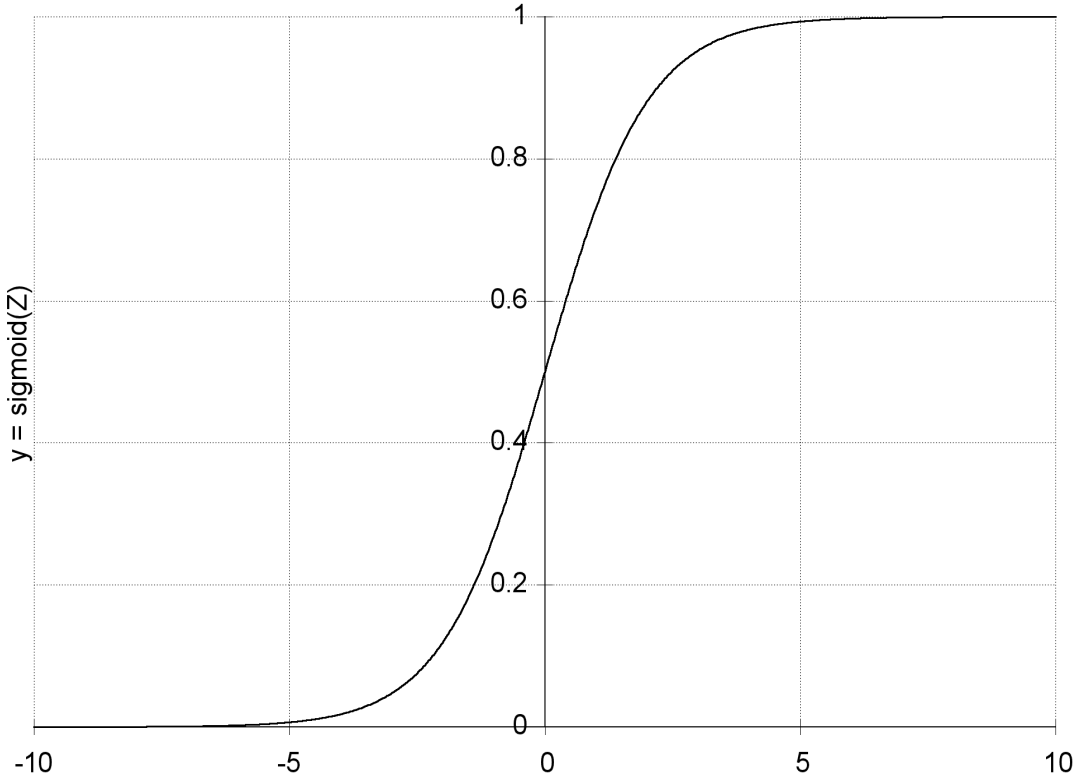
2.3.2.1 Função de Ativação Sigmoid

A função *sigmoid* cuja formula matemática é f(z) = 1/(1+ e^ (-z)) foi uma das mais usadas para mapear a saída de um neurônio para um valor entre 0 e 1. Ela é usada quando precisamos saber a probabilidade de que algo seja

verdadeiro ou falso (como um problema de classificação *verdadeiro* ou *falso*), ou seja quando um valor pode variar entre 0 e 1. O gráfico da função *sigmoid* se parece com uma letra “S” (SHARMA, 2017).

Apesar de ser fácil de entender e de aplicar, a função *sigmoid* caiu em desuso, pois causava um problema chamado Desvanecimento de Gradiente (V*anishing Gradient*), que ocorre quando os valores retornados pelos nurônios e mapeados pela função, tornam-se pequenos demais ao se aproximar de um dos limites (0 ou 1). Esse problema faz com que o algorítmo que atualiza os parâmetros de peso da rede neural, o Gradiente Descendente, se torne cada vez mais lento a cada iteração, assim o gradiente demora demais para convergir (atingir o local mínimo global). A função *sigmoid*, ainda é usada, mas normalmente a vemos apenas na última camada de uma rede, que justamente a saída final que deve ser um valor de predição exatamente entre 0 e 1 (WALIA, 2017).

Figura 4 - Função de ativação Sigmoid

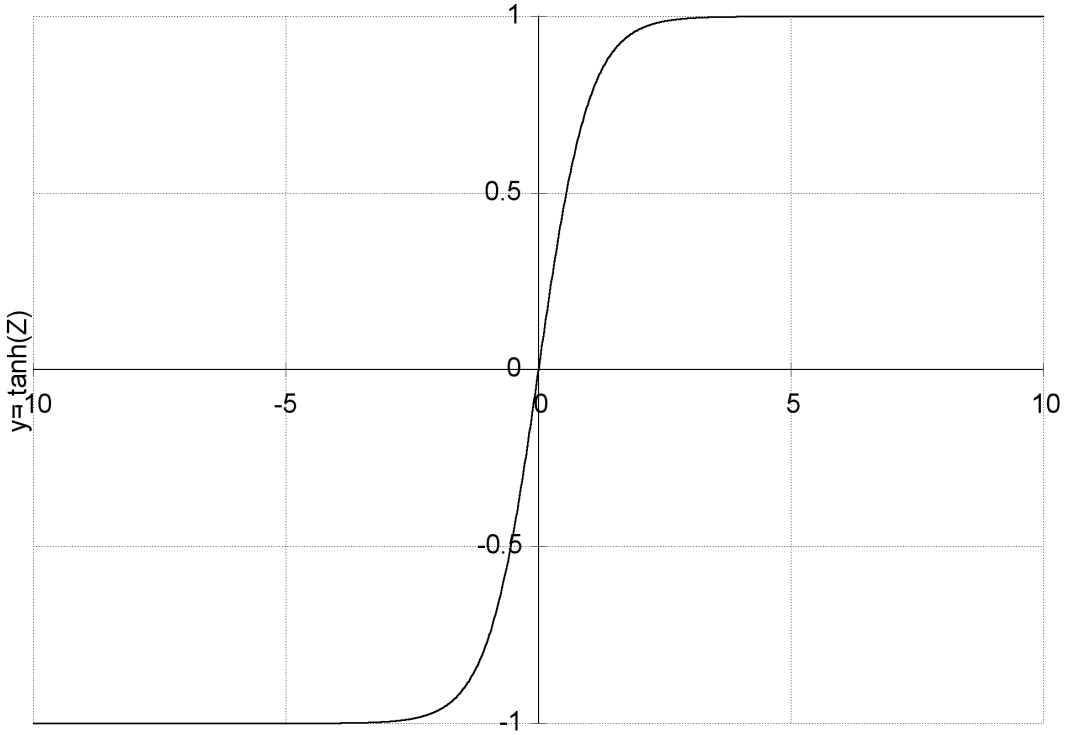


Fonte: O Autor

2.3.2.2 Função de Ativação TanH

Uma alternativa à *sigmoid*, a função *TanH* (*Hyperbolic Tangent function*) passou a ser usada em seu lugar. Com a formula f(z) = 1—exp(-2z) / 1 + exp(-2z). Essa função mapeia a saída de um neurônio para um intervalo entre -1 e 1, sua saída é centralizada em 0, ao contrário da *sigmoid* cuja saída é centralizada em 0,5. A *TanH* se aproxima mais da identidade, sendo assim uma alternativa mais atraente do que a *sigmoid* para servir de ativação. Mas ainda sofre do problema de Desvanecimento de Gradiente (WALIA, 2017).

Figura 5 - Função de ativação TanH

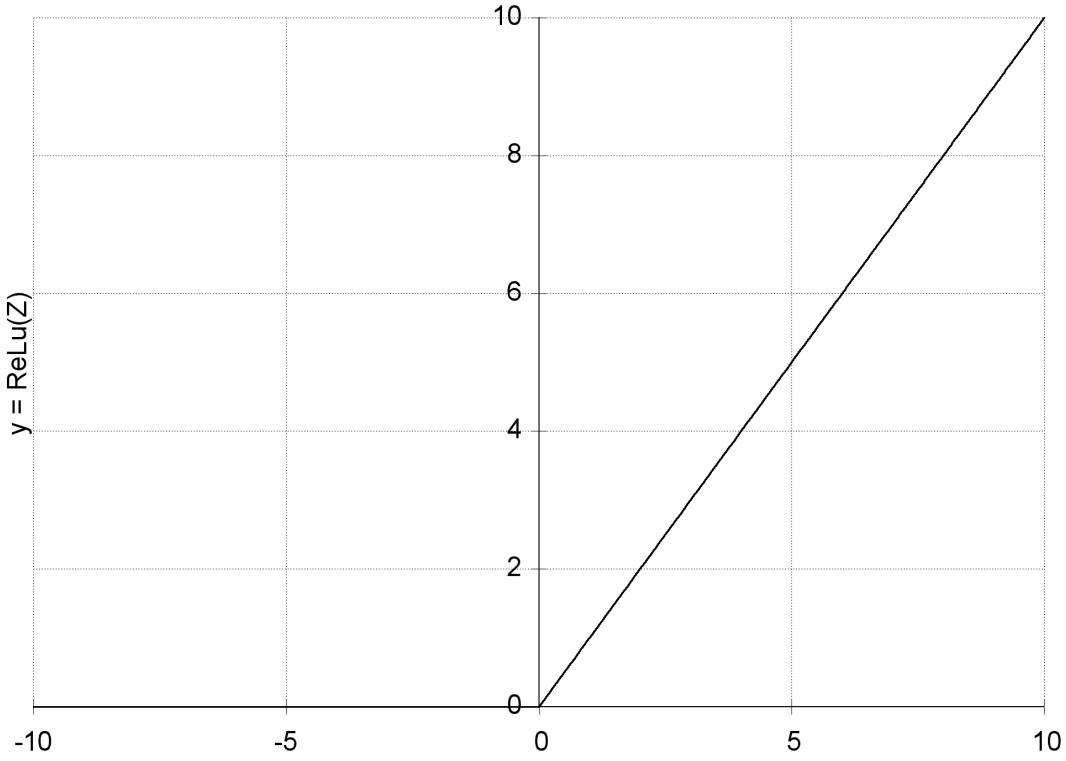


Fonte: O Autor

2.3.2.3 Função de Ativação ReLu

Evitando o problema do Desvanecimento de Gradiente. A função ReLU (*Rectified Linear Unit*) tornou-se, nos ultimos anos, a função mais usada em redes neurais de aprendizagem profunda. Ela mapeia o valor de saída de um neurônio para um intervalo entre 0 e +*infinito*, ou seja, f(z) é zero quando z é menor que zero e f (z) é igual a z quando z é igual ou maior que zero. Sua formula fica da seguinte maneira: *R(z) = max(0, z). Ou seja, se x < 0, R(x) = 0 e se x >= 0, R(x) = x.* Assim podemos ver logo de cara que essa função é muito simples e eficiente (WALIA, 2017).

Figura 6 - Função de ativação ReLu



Fonte: O Autor

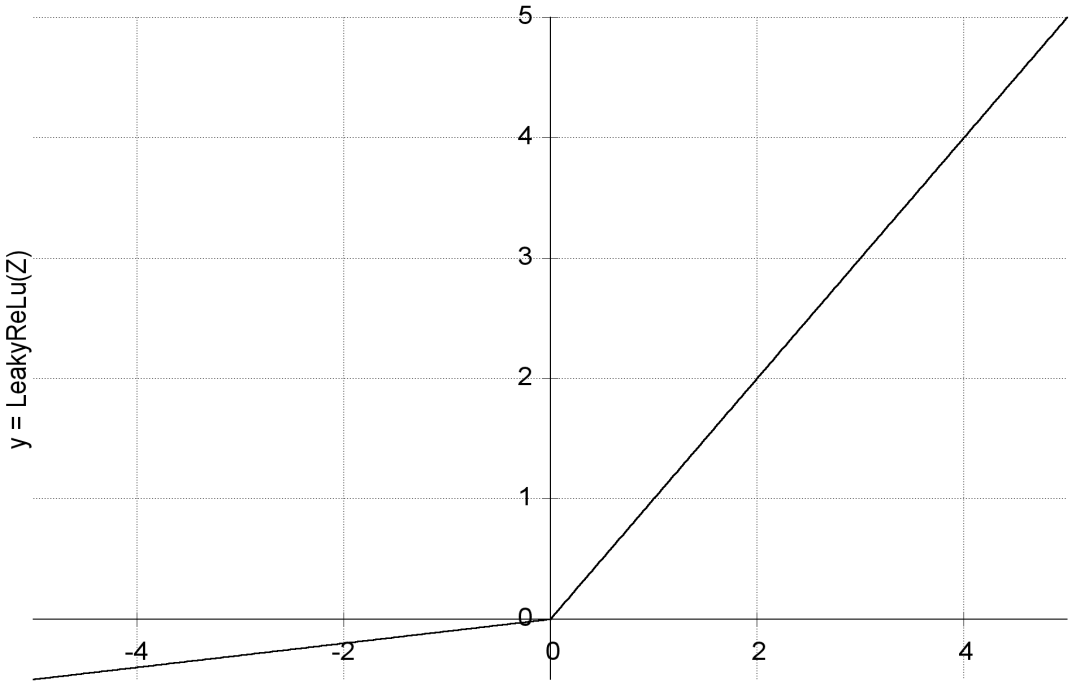
Mas a ReLu possui uma limitação: é aconselhável usá-la somente nas camadas escondidas (*hidden layers*) da rede neural – as camadas que ficam no caminho entre a camada de entrada (*input layer*) e a camada de saída

(*output layer*) da rede neural – por isso a função *sigmoid* ainda é usada na camada de saída (SHARMA, 2017).

Um outro problema que vale a pena ressaltar sobre a ReLu é que alguns neurônios tendem a ‘morrer’ durante o treinamento do modelo de rede neural, trata-se de um fenômeno que faz com que o neurônio passe a produzir apenas zeros em sua saída. Isso ocorre quando a soma ponderada que o neurônio aplica nos parâmetros de peso antes de chamar a função ReLu se torna negativa, assim a ReLu mapeia essa saída para zero, faxendo com que a saída final do neurônio seja zero. Nesse ponto, a derivada da função também é zero, isso faz com que os parâmetros de peso do neurônio deixem de ser atualizados pelo gradiente descendente (FACURE, 2017).

Para corrigir o problema dos neurônios que estão “morrendo”, foi introduzida uma modificação para melhorar a ReLu. Esta modificação consiste em introduzir uma leve inclinação na parte negativa de seu domíno para assim manter as atualizações ativas. Dessa forma surge uma nova variante da ReLu, a função *Leaky ReLu* (ReLu Vazada). Sua formula é a seguinte: $LeakyReLU(x,\alpha)=max\{\alpha x,x\}$. Assim o valor da ativação é 1 se x for maior ou igual a zero, ou alpha caso contrário (FACURE, 2017).

Figura 7- Função de Ativação Leaky ReLu



Fonte: O Autor

Para corrigir esse problema, outra modificação foi introduzida, chamada Leaky ReLu, para corrigir o problema dos neurônios que estão morrendo. Introduz uma pequena inclinação para manter as atualizações ativas (FACURE, 2017).

2.3.3 Gradiente Descendente

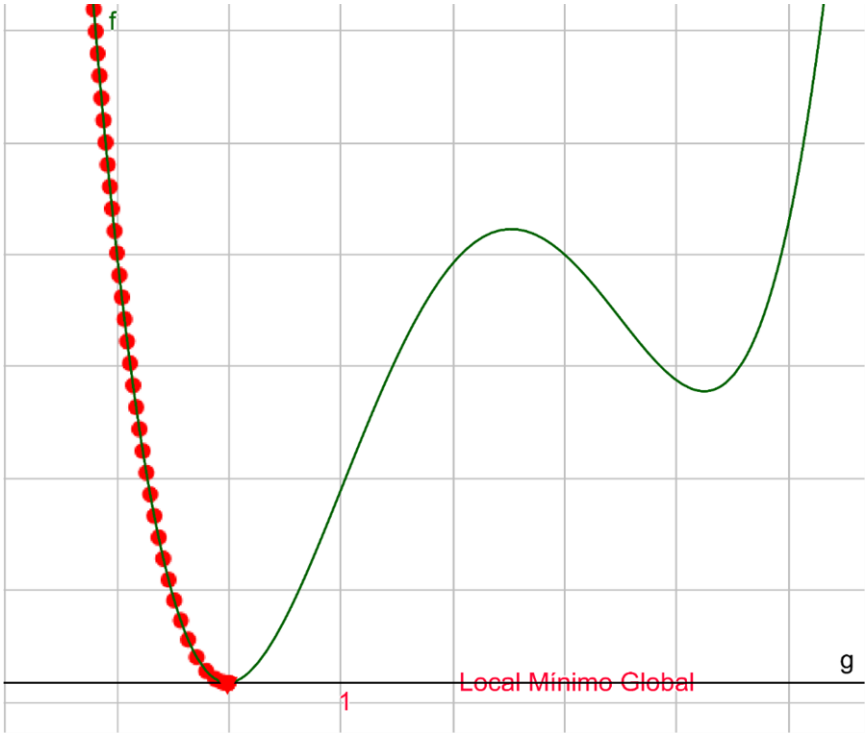
O algoritmo usado para atualizar gradualmente os pesos da rede neural até que eles se ajustem aos dados, ou seja, até que a rede tenha aprendido, é chamado de Gradiente Descendente. Consiste em minimizar uma função de erro (o erro é a diferença entre o resultado predizido pela rede e o valor correto rotulado no exemplo de treino). O Gradiente Descendente atualiza os pesos da rede neural até que o valor do erro se aproxime o mais próximo possível de zero.

Sobre minimizar uma função, CHOLLET (2017, p. 48, tradução nossa) afirma que:

> Dada uma função diferenciável, é teoricamente possível achar o seu mínimo analiticamente. Sabe-se que o mínimo de uma função é um ponto onde sua derivada é zero, então tudo o que é preciso fazer é encontrar todos os pontos onde a derivada atinge o zero e checar em qual desses pontos a função atinge o seu menor valor.

Baseando-se no valor do erro, o Gradiente Descendente vai atualizando aos poucos os parâmetros de peso de forma que o erro vai cada vez mais diminuindo e se aproximando cada vez mais de um local mínimo global (o menor dos locais mínimos, onde a função atinge seu menor valor). Porém dependendo do passo em que o gradiente atualiza os parâmetros, ele pode passar do local mínimo, e em vez de diminuir o erro pode passar a aumentá-lo. Então esse "passo" conhecido como taxa de aprendizagem (*learning rate*) deve ser definido como sendo um valor pequeno para garantir que o gradiente vai se aproximar do local mínimo global, mas não ultrapassá-lo, porém, se a taxa de aprendizagem for muito pequena, o gradiente vai se tornar muito lento. Cabe ao desenvolvedor testar o gradiente descendente com diferentes valores para a taxa de aprendizagem e escolher o melhor (CHOLLET, 2017).

Figura 8 - Gradiente Descendente se aproximando do Local Mínimo Global



Fonte: O Autor

2.3.4 O algoritmo Back-propagation

Talvez uma das partes mais complicadas quando se fala de redes neurais, é o algoritmo de retropropagação (*back-propagation*). Em redes neurais, quando queremos minimizar um erro, precisamos retropropagar (propagar o valor do erro para trás na rede neural), encontrando a derivada do erro em relação a cada peso e subtraindo esse valor do valor do peso. Propagando para frente, vemos o quão bem a nossa rede neural está se comportando e encontramos o erro. Depois de encontrar um valor de erro fazemos a retropropagação e usamos o Gradiente Descendente para atualizar novos valores de pesos. Em seguida, encaminhamos novamente a propagação para ver o desempenho desses pesos e, em seguida, será feita a retropropagação novamente para atualizar os pesos. E assim continua até atingirmos alguns mínimos para o valor de erro (CHOLLET, 2017).

Quando se usa *frameworks* de alto nível para se criar um modelo de rede neural, algorítmos como o *back-propagation* são “escondidos” pelos *frameworks*, bastando ao desenvolvedor criar a lógica do *forward-propagation* (que consiste apenas em propagar os dados através das camadas da rede neural e obter uma predição na camada final), e então o *framework* se encarrega de fazer a retropropagação.

Como usaremos o Keras, que é um *framework* de altíssimo nível para redes neurais, não entraremos em detalhes mais técnicos sobre o back-propagation, mas uma breve explicação matemática para a retropropagação é a seguinte: como vimos anteriormente, se uma função é diferenciável, podemos computar suas derivadas. Segundo Chollet (2017), uma rede neural consiste, na prática, numa série de operações com tensores alinhados uns com os outros, cada um dos quais possuindo uma derivada conhecida. Isso nos permite calcular as derivadas parciais de cada parâmetro de peso em relação ao erro e assim obter a contribuição de cada parãmetro para com o valor de erro obtido na última camada.

De acordo com CHOLLET (2017, p. 51, tradução nossa):

> O cálculo nos diz que uma cadeia de funções pode ser derivada usando a seguinte identidade, chamada regra da cadeia: f (g (x)) = f '(g (x)) * g' (x). A aplicação da regra da cadeia ao cálculo dos valores de gradiente de uma rede neural dá origem a um algoritmo chamado Retropropagação (também às vezes chamado de diferenciação no modo reverso). A retropropagação começa com o valor do erro final e trabalha de volta das camadas superiores para as camadas inferiores, aplicando a regra da cadeia para calcular a contribuição de cada parâmetro no valor de erro.

2.4 REDES NEURAIS CONVOLUCIONAIS

As redes Neurais Convolucionais ou Convnets inspiram-se biológicamente no modelo do cortex visual. O córtex visual possui pequenas regiões de células que são sensíveis a regiões específicas do campo visual. Essa ideia foi ampliada por um fascinante experimento de Hubel e Wiesel, em 1962, onde mostraram que algumas células neuronais individuais no cérebro respondiam (ou ativavam-se) somente na presença de bordas de certa orientação. Por exemplo, alguns neurônios dispararam quando expostos a arestas verticais e outros quando exibiram arestas horizontais ou diagonais. Eles descobriram que todos esses neurônios estavam organizados em uma arquitetura colunar e que, juntos, eram capazes de produzir percepção visual. Essa ideia de especialização de componentes dentro de um sistema com tarefas específicas (células neuronais do córtex visual que procuram características específicas) é a ideia base por trás das Convnets (DESHPANDE, 2016).

Geralmente as redes neurais possuem todos os neurônios de uma cada camada conectados a todos os neurônios da camada seguinte e da camada anterior, por isso essas camadas são chamadas de Camadas Completamente Conectadas (*Fully Connected Layers*) ou Camadas Densamente Conectadas (*Densely Connected Layers*). As Convnets, são exceção a essa regra, pois não precisam ter todos os neurônios de cada camada conectados aos nurônios da camada seguinte. Em uma Convnet as ativações de uma próxima camada dependem apenas de um pequeno número de ativações da camada anterior,

ou seja, as conexões entre as camadas convolucionais são esparsas se comparadas às conexões entre as camadas dendas.

Essas conecções esparsas entre as unidades de cada camada (os neurônios também são chamados de unidades) faz com que os neurônios de cada próxima camada recebam somente as variações de pesos condizentes ao campo receptivo de cada neurônio, de forma que cada unidade não será responsável por variações fora do seu campo de recepção. Ou seja, se na primeira camada foram reconhecidos traços de um rosto, os neurônios da segunda camada poderão reconhecer partes do rosto, como olhos, boca, orelha, e um neurônio que reconhece os padrões de um olho, não será afetado pelas mesmas variações de entrada que um neurônio responsável por reconhecer uma boca, por exemplo. Essa arquitetura garante, portanto, que os "filtros" (falaremos sobre filtros mais adiante) aprendidos produzam a resposta mais forte a um padrão de entrada espacialmente local. (DEEPLEARNING 2018).

Podemos então dizer que a diferença entre uma camada completamente conectada e uma camada convolucional é que as camadas densas aprendem padrões globais (padrões envolvendo todos os pixels em uma imagem, por exemplo) enquanto que as camadas de convolução aprendem padrões locais, no caso de imagens, padrões encontrados em pequenas janelas ou partes da imagem (CHOLLET, 2017).

Segundo (CHOLLET, 2017), essa característica dá às redes convolucionais duas propriedades importantes:

Os padrões aprendidos são reutilizáveis em cada imagem de exemplo. Após aprender um certo padrão no canto inferior direito de uma imagem, por exemplo, a camada convolucional pode reconhecê-lo em qualquer lugar, como no canto superior esquerdo, por exemplo. Uma rede densamente conectada teria que aprender o padrão novamente se ele aparecer em um novo local. Isso torna as convnet eficientes ao processar imagens. Com isso elas conseguem aprender com menos amostras de treinamento para aprender representações que possuem generalizações.

Elas podem aprender hierarquias espaciais de padrões. Cada camada de convolução aprende padrões cada vez mais complexos a medida que a rede neural avança para as camadas mais profundas. Por exemplo: uma primeira camada de convolução aprende pequenos padrões locais, como traços e bordas, já uma seguda camada de convolução consegue aprender padrões maiores formados dos recursos reconhecidos pela camada anterior, e assim por diante. Assim as convnets aprendem com cada vez mais eficiência cada vez mais visuais complexos e abstratos. Isso permite que as convnets aprendam com eficiência conceitos visuais cada vez mais complexos e abstratos.

2.4.1 O Funcionamento das Convnets

Foram explicadas as principais ideias por trás da abordagem por rede convolucional e o porquê de serem superiores às redes com camadas completamente conectadas, mas como uma rede convolucional realmete funciona?

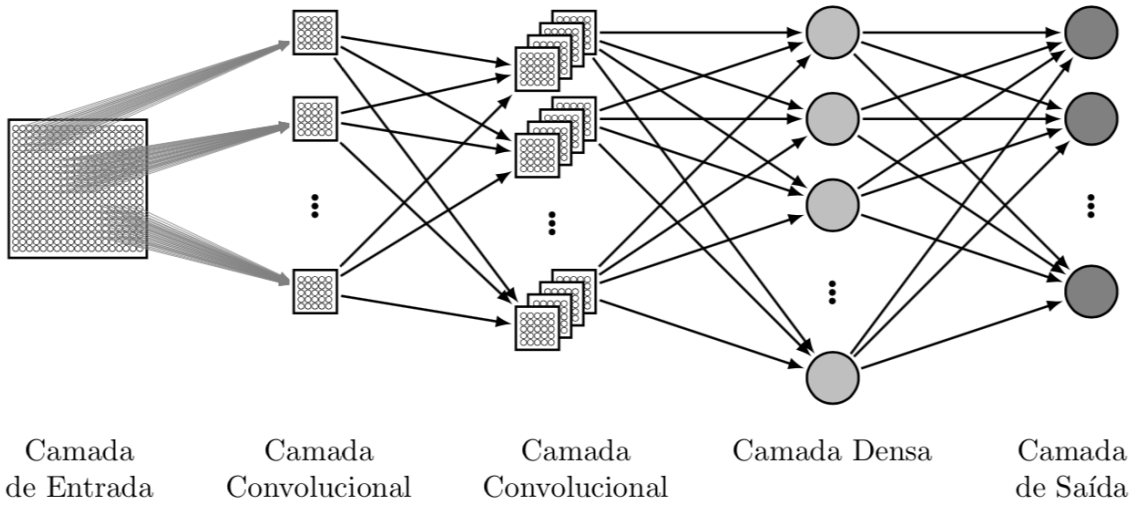
Como mencionado anteriormente, uma camada de convolução não possui todos os seus neurônios completamente conectados com os neurônios da camada seguinte, o que faz com que a conexão entre as camadas da Convnet sejam esparsas. Assim, em uma determinada camada, ao invés de ligar cada entrada a cada neurônio (como uma rede comum faria), as Redes Neurais Convolucionais restringem as conexões intencionalmente para que qualquer neurônio aceite as entradas apenas a partir de uma pequena subseção da camada anterior. Essas subseções são como filtros que se ativam ao identificar um padrão. Por exemplo, um filtro pode ser uma matriz de 5 x 5 ou 3 x 3 pixels, que itera sobre a imagem de tamanho *n* x *m* recebida e ativa-se quando encontra um determinado padrão que ele deseja filtrar, como uma aresta vertical ou horizontal (RAVINDRA, 2017).

Na sequência, os dados da camada de convolução são recebidos por uma camada de agrupamento (*pooling layer*). Ela usa a saída dos filtros para

gerar uma nova matriz agrupada que contém apenas as partes da imagem que são importantes descartando o restante, o que minimiza os cálculos que precisam ser feitos, evitando o problema de superposição. Em termos simples, a superposição ou *overfitting* ocorre quando um modelo se adapta muito bem aos dados em que foi treinado, o que não é bom, pois lidar muito bem com dados nunca vistos antes (RAVINDRA, 2017).

Com o tamanho da entrada reduzido drasticamente pelas camadas de convolução e de agrupamento, a nova matriz resultante, que deve ser algo que uma rede normal é capaz de manipular e ao mesmo tempo preservando as partes mais significativas dos dados, é usada como entrada para uma rede neural completamente conectada. No passo final, a saída representará a confiança do sistema de que a imagem recebida é, por exemplo, um rosto humano (RAVINDRA, 2017).

Figura 9 - Exemplo de topologia de uma Rede Neural Convolucional.



Fonte: Sakurai (2017)

2.5 RECONHECIMENTO FACIAL

No reconhecimentofacial, é preciso reconhecer a identidade de uma pessoa apenas alimentando o sistema com uma imagem do rosto dessa pessoa. Assim, se a pessoa não for reconhecida, isso significa que a imagem dessa pessoa não está na base de dados do sistema (DOUKKALI, 2017).

Este problema não pode ser resolvido somente usando uma rede neural convolucional, pois elas não funcionam com poucos treinos de exemplo. Além disso, não é conveniente retreinar o modelo sempre que adicionamos a imagem de ima nova pessoa no sistema. Mas, existe uma solução para esse problema, pode-se usar Redes Neurais Siamesas para reconhecimento facial (DOUKKALI, 2017).

2.5.1 One-shot learning

Para resolver o problema de analisar duas imagens – como as fotos de duas pessoas, por exemplo – e dizer se são pessoas diferentes ou se são a mesma pessoa nas duas fotos, seria demorado e bastante custoso conseguir uma quantidade muito grande de fotos da mesma pessoa para treinar um modelo de rede neural para aprender a reconhecê-la. Obviamente o problema se torna mais complicado quando queremos que nosso modelo seja capaz de identificar várias pessoas diferentes. Seria necessário ter milhares de imagens de varias pessoas afim de treinar uma rede neural para reconhecê-las (DOUKKALI, 2017).

Na aprendizagem profunda, normalmente precisamos de uma grande quantidade de dados e, quanto mais dados, melhores são os resultados. Porém, seria melhor aprender com poucos dados, afinal, nem sempre podemos conseguir uma grande quantidade de dados (DOUKKALI, 2017).

Então surge a ideia do One-Shot Learning. Gupta (2017) afirma que através dessa abordagem exige que tenhamos apenas um exemplo de

treinamento de cada classe que desejamos prever. O modelo ainda é treinado em várias instâncias, mas elas só precisam estar em um domínio semelhante ao do nosso exemplo de treino. No caso do reconhecimento facial, treina-se uma rede neural com várias imagens de rostos de várias pessoas diferentes, em diversos ângulos e ilumiações diferentes, assim o modelo aprende a reconhecer faces humanas. Então, se quisermos reconhecer uma determinada pessoa em uma imagem, precisamos de apenas uma foto dessa pessoa, e então perguntamos ao modelo se esta pessoa está na imagem. Isso levando em conta que o modelo não foi treinado com imagens da pessoa em questão.

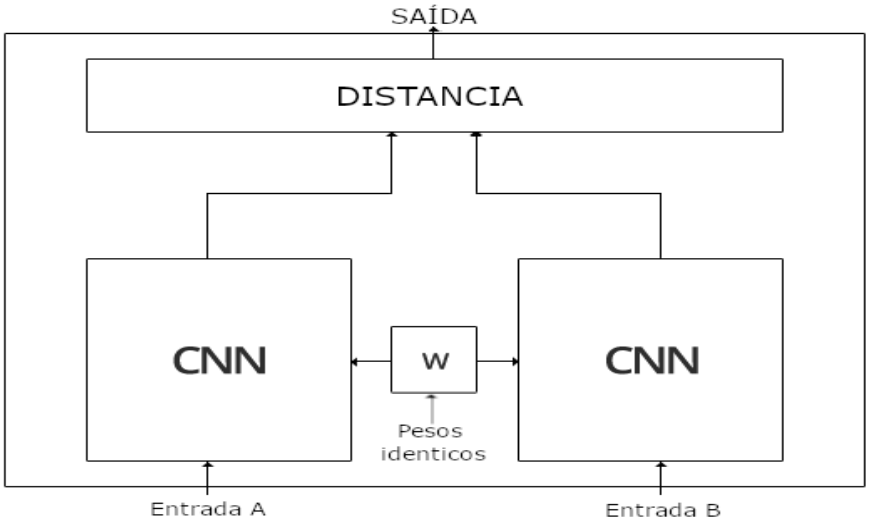
Então pode-se dizer que a ideia do One-Shot Learning é fazer com que uma máquina olhe uma vez e já seja capaz de reconhecer. “Como seres humanos, nós podemos reconhecer uma pessoa por sua face simplesmente conhecendo-a uma única vez, e isso é desejável para os computadores porque muitas vezes os dados são mínimos” (Gupta, 2017).

2.5.2 Rede Neural Siamesa

De acordo com Doukkali (2017, tradução nossa) “o objetivo de uma rede neural siamesa é descobrir o quão similares duas coisas comparáveis são”. Esse modelo de rede neural consiste em duas subredes identicas contentendo os mesmos parâmetros e pesos.

As duas redes irmãs são redes convolucionais, com suas camadas de convolução e de agrupamento seguidas de uma camada totalmente conectada e finalmente um vetor com as saídas preditivas. Cada rede recebe uma imagem em sua entrada e codifica esta imagem através do *forward propagation*, na camada de saída de cada rede é obtido um vetor com os dados da imagem codificados (Gupta, 2017).

Figura 10 - Arquitetura de uma Rede Neural Siamesa



Fonte: Wicklin (2018)

Inicialmente, quando pensamos em comparar duas imagens para descobrir se são a foto da mesma pessoa nas duas, poderíamos pensar em comparar pixel por pixel e ver se os padrões “batem”. Mas essa é, obviamente uma abordagem pobre porque se em uma imagem a pessoa está numa posição diferente da outra, ou em un canto diferente, o modelo vai predizer que as imagens são diferentes (ou seja, não é a mesma pessoa), isso sem considerar ainda as diferenças de iluminaçao de cada imagem que também podem influenciar na predição.

E é aí que a abordagem por redes neurais siamesas se mostra eficiente, pois não compara as imagens em seu estado bruto, mas as codifica com base em suas características e compara suas codificações afim de descobrir o quão diferente ou distante uma mimagem é da outra.

Segundo Doukkalli (2017), para comparar as duas imagens X1 e X2 recebidas pelas duas redes irmãs, calculamos a distância entre suas codificações f(X1) e f(X2). Se for menor que um limiar previamente escolhido (um hyperparâmetro), significa que as duas figuras são a mesma (há uma distância aceitavel entre elas) pessoa, se não, então são duas pessoas diferentes (há uma distância muito grande entre elas).

2.5.3 Função de erro tripla

Usamos essa função para podermos aprender os parâmetros para obter uma boa codificação para a imagem de entrada, afim de obter uma rede neural que saiba diferenciar imagens (DOUKKALI, 2017).

No lugar da função de erro convencional, usamos uma função de erro tripla, que é simplesmente uma função de erro usando três imagens: uma âncora A, uma imagem positiva P (a mesma pessoa que a âncora) e uma imagem negativa N (uma pessoa diferente da âncora). Desta forma, esperamos que a distância d(A, P) entre a codificação da âncora e a codificação da imagem positiva seja menor ou igual a distância entre a codificação da âncora e a codificação da o exemplo negativo d(A, N) (DOUKKALI, 2017).

Há ainda problema de que o modelo pode aprender a fazer a mesma codificação para imagens diferentes, isso que significa que as distâncias serão zero e, infelizmente, satisfará a função de erro tripla. Por isso é adicionado uma margem *alpha* (um hiperparâmetro), assim evita-se que isso aconteça e sempre haja um intervalo entre A e P versus A e N (DOUKKALI, 2017).

$$d(A,P) + \alpha \leq d(A,N)$$

$$\|f(A)\text{-}f(P)\|^2 + \alpha \leq \|f(A)\text{-}f(N)\|^2$$

$$\boxed{\|f(A)\text{-}f(P)\|^2 \text{-} \|f(A)\text{-}f(N)\|^2 + \alpha \leq 0}$$

$$L(A,P,N) = \mathbf{max}\left(\|f(A)\text{-}f(P)\|^2 \text{-} \|f(A)\text{-}f(N)\|^2 + \alpha,\ 0\right)$$

O que a função acima quer dizer é que enquanto o máximo entre d (A, P) - d (A, N) + alfa é menor ou igual a zero, a perda L (A, P, N) é zero, mas se

for maior que zero, a perda será positiva e a função tentará minimizá-la para zero ou menor que zero.

Então temos a nova função de custo que é a soma de todas as perdas individuais em triplas diferentes de todo o conjunto de treinamento.

2.5.4 O conjunto de treino

O conjunto de treino deve conter multiplas imagens da mesma pessoa para ter os pares A e P, e quando o modelo tiver sido treinado, poderemos reconhecer uma pessoa com uma única foto (DOUKKALI, 2017).

2.5.5 A escolha das triplas para treinar o modelo.

Não se deve escolher as triplas aleatóriamente, pois se as imagens positivas e negativas forem muito diferentes, pois a distância entre elas será muito grande na maioria das vezes e será difícil satisfazer a restrição da função de erro. Assim a descida do gradiente aprenderá muito pouco com a descida do gradiente. Portanto, é importante formar grupos A, P e N de forma que P não esteja tão distante de N. dessa forma o gradiente aprende com exemplos difíceis e aprende mais (DOUKKALI, 2017).

2.6 A LINGUAGEM PYTHON

Apesar de, somente nos últimos anos, ter se destacado e se tornado uma das mais populares linguagens de programação de software livre atualmente em uso, a linguagem Python é mais antiga que Java, R e Javascript, mesmo que muitos ainda a vejam como uma novidade, afirma Wood (2015).

2.6.1 Uma breve história da linguagem Python

A linguagem Python surgiu em 1989, criada por Guido Van Rossum, inicialmente como um projeto de passatempo, e o nome da linguagem é famoso, não por ser o nome de uma cobra constritora, mas por referenciar o programa de comédia britânico *Monty Python Flying Circus*, afirma Wood (2015).

Python nasceu da linguagem ABC, que um projeto do instituto holandês de pesquisa CWI, onde Van Rossum trabalhou, e do sistema operacional distribuído Amoeba. Wood (2015) afirma que Van Rossum criou Python para ser uma linguagem de *scripts* para o sistema Amoeba, e que a facilidade de se entender a linguagem e seu suporte à multiplas plataformas era um dos principais pontos fortes da linguagem, uma inovação vital na era dos primeiros computadore pessoais.

Durante os anos noventa Python cresceu adquirindo novas ferramentas de programação funcioal. E também duarnte este período, Python desempenhou um papel importante na na iniciativa Programação de Computadores para Todos (CPE4E - Computer programming for Everybody), de Van Rossum. O CP4E tinha o objetivo de tornar a programação mais acessível para leigos, e encorajar a programação a um nível básico de alfabetização como um conhecimento essencial, como Inglês e Matemática, e devido à acessibilidade e sintaxe limpas, ele desempenhou um papel fundamental nisso. O CP4E está agora inativo, mas apesar disso, Python continua fácil e é comum que novos programadores queiram aprendê-la, Wood (2015).

Conforme a linguagem Python crescia nos anos noventa, sugia um problema de aceitação, que era a dependencia de Guido Van Rossum. Conforme Wood (2015), a preocupação com essa dependência resulta no lançamento da versão 2.0 no ano 2000 pela equipe BeOpen Python Labs. Assim, o caráter da linguagem se tornou ainda mais aberto e seu prcesso de desenvolvimento ficou mais voltado para a comunidade, com mais

transparência. Dessa forma, talvez a maior mudança na versão 2.0 não tenha sido o código, mas a forma como Python é desenvolvida.

Wood (2015) afirma que a versão 2.7 é usada até hoje e tem seu suporte previsto para até 2020, porém, a versão 3 da linguagem Python foi lançada em 2008 sob a premissa quase impensável de se fazer uma revisão completa na linguagem, sem compatibilidade retroativa. Segundo Wood (2015) essa decisão foi controversa, e nasceu em parte do desejo de “limpar a casa” em Python. Essa mudança focou-se em remover construtores e módulos duplicados afim de garantir que em Python 3 haveria uma – e somente uma – maneira óbvia de se fazer as coisas. Mesmo com a ferramenta 2to3, que é capaz de reconhecer trechos do código do Python 2 que precisariam mudar para funcionar em Python 3, muitos usuários preferiram manter suas bases de código em Python 2, porém, o crescimento do Python 3 era inevitável, mesmo com discussões acirradas na comunidade Python.

Dessa forma o futuro da linguagem Python se mostra promissor. Wood (2015, tradução nossa) ressalta que:

> A base de usuários do Python é vasta e crescente - não vai desaparecer tão cedo. Utilizada por empresas como Nokia, Google e até mesmo pela NASA por sua sintaxe fácil, parece ter um futuro brilhante à frente, apoiado por uma enorme comunidade de desenvolvedores de sistemas operacionais. Seu suporte a vários paradigmas de programação, incluindo programação Python orientada a objetos, programação Python funcional e modelos de programação paralela, torna-a uma opção altamente adaptativa - e sua absorção continua crescendo.

2.6.2 Por que escolhemos Python?

São vários os motivos que nos levaram a escolher Python para este projeto.

Python é uma linguagem fácil de usar e entender, conforme Martelli (2006), Python é uma linguagem de altíssimo nível (VHLL – *Very-high-level Language*) e, portanto, está mais distante do nível de máquina que clássicas linguagens compiladas como C e C++, e é simples e mais estável que outras

linguagens de alto nível como C# e Java, e por isso proporciona uma alta produtividade de programação.

Outro bom motivo para se usar Python é o fato de ela ser uma linguagem interpretada. Nelli (2015) afirma que os códigos Python necessitam apenas de um interpretador para serem executados e, portanto, não há tempo de compilação. Além disso, Python é também uma linguagem altamente iterativa, o programador pode executar trechos de código na linha de comandos e ver seu resultado sem ter que escrever um *script* inteiro para depois executar e ver o resultado.

Python usa o paradigma da Orientação a objetos, porém sua estrutura é flexível o bastante para usar outros paradigmas. Segundo Nelli (2015), Python permite outras abordagens em relação à orientada a objetos, como a funcional ou a vetorial.

Python é uma linguagem portátil. Um código Python pode rodar em qualquer sistema que tenha um interpretador de linguagem Python instalado, o código não precisa ser alterado e, isso lhe dá uma vantagem chave que é a portabilidade, (NELLI, 2015).

Outra vantagem é que Python é uma linguagem de código aberto (*open source*). Sua principal implementação, a Cpython, é completamente livre e de código aberto. Uma extensa comunidade está frequentemente melhorando o código dessa linguagem e tornando suas bibliotecas ainda mais ricas e eficientes, (NELLI, 2015).

Como dissemos anteriormente, Python é simples de aprender e usar. Como mencionado por Nelli (2015), talvez a simplicidade seja o aspecto mais importante, pois é o primeiro aspecto que um programador, até mesmo um novato percebe ao se deparar com esta linguagem. Mas o fato de ser simples não é uma limitação para Python, uma vez que é uma linguagem que está se espalhando por todas as áreas da computação.

Citamos grandes motivos para se usar a linguagem Python, porém o que mais nos motivou a usá-la neste projeto é o fato de que Python é uma

linguagem bastante popular no ramo da Aprendizagem de Máquina. Segundo Stoltzefus (201-?) existem muitas razões para isso.

Além de ser simples e fácil de aprender, como já dissemos anteriormente, sua sintaxe é elegante e bastante próxima da matemática. Stoltzefus (201-?) diz também que a semântica de Python é apontada por especialistas como sendo uma correspondência particular com muitas ideias matemáticas comuns, isso torna curta a curva de aprendizagem para se aplicar essas ideias na linguagem Python.

2.7 TENSORFLOW KERAS E NUMPY

2.7.1 Tensorflow

Além de ser poderosa por si só, Python conta com diversos *frameworks* para Aprendizagem de Máquina, um deles é o Tensorflow.

Tensorflow é uma biblioteca de código aberto desenvolvida na Google pela *Brain Team*, quase todos os aplicativos do Google usam Tensorflow para Aprendizagem de Maquina e, portanto, quando usamos esses aplicativos estamos indiretamente utilizando modelos que foram criados com Tensorflow (PRASANNA, 2017).

Conforme Prasanna (2017), o Tensorflow é uma estrutura computacional que expressa algoritmos que envolvem um grande número de operações *Tensor*, uma vez que as redes neurais podem ser expressadas como gráfos computacionais, elas podem ser implementadas usando o Tensorflow como uma série de operações em tensores. Tensores são matrizes n-dimencionais que representam nossos dados.

A maior vantagem do Tensorflow é o paralelismo, o que significa que seus grafos computacionais são executados em paralelo, o que nos dá controle total sobre a execução e podemos agendar diferentes operações em diferentes processadores como CPU, GPU, etc (PRASANNA, 2017, tradução nossa).

Outra característica importante que vale ressaltar sobre o Tensorflow é sua velocidade de processamento. Prasanna (2017) afirma que o Tensorflow é otimizado para ser rápido fazendo uso de técnicas para operações de algebra linear mais rápidas.

2.7.2 Keras

Keras é uma API de rede neural de alto nível, escrito em Python e capaz de executar em cima de várias bibliotecas de nível inferior, usadas como backends, incluindo o Tensorflow (apesar de o Tensorflow já ser de nível alto, porém, Keras é ainda mais alto nível). Keras pode ser usado como uma interface para o Tensorflow mais simples de usar, assim tornando a vida do desenvolvedor mais fácil. Keras foi projetado com foco em permitir a experimentação rápida, sendo capaz de ir da ideia ao resultado com o mínimo de atraso possível (KERAS DOCUMENTATION, 2018)

O código Keras é portátil, isso significa que pode-se implementar uma rede neural em Keras usando o Theano (um outro framework de rede neural) como um back-end e, em seguida, especificar o back-end para ser executado posteriormente no TensorFlow, sem a necessidade de qualquer outra alteração no código (MAYO, 2017).

2.7.3 Numpy

Optamos por utilizar a biblioteca Numpy por ser muito utilizada para cálculos matemáticos em Python. Segundo Prasanna (2017) a numpy está entre as maiores biblitecas matemáticas e científicas da linguagem Python. O próprio Tensorflow, bem como outras plataformas, faz uso da numpy internamente para executar diversas operações nos *Tensors*. Uma das principais características da Numpy é sua interface de *arrays.* “Um *array* é uma estrutura de dados que contem um grupo de elementos. Tipicamente esses

elementos são do mesmo tipo de dado, como *integer* ou *string*.” (TECHTERMS, 2007, tradução nossa).

Essa interface permite expressar diversos tipos de dados como imagens e som, na forma de *arrays* de números reais de N-dimensões. Conhecer sobre numpy é muito importante para Aprendizagem de Máquina (PRASANNA, 2017).

2.8 JUPYTER NOTEBOOK

Como ferramenta de desenvolvimento para este projeto, escolhemos o Jupyter Notebook por ser uma excelente ferramenta baseada na web que permite prootipágem rápida e compartilhamento de projetos relacionados a dados.

O Jupyter Notebook é um aplicativo servidor-cliente que pode ser executado em um desktop local que não requer acesso à internet (funciona rodando em *localhost*) ou pode ser instalado em um servidor remoto e acessado via internet. Essa aplicação permite editar e executar documentos *notebook* através de um navegador web (JUPYTER/IPYTHON NOTEBOOK QUICK START GUIDE, 2015).

Os documentos *Notebook* são arquivos produzidos pelo Jupyter que podem conter código (como código python por exemplo), e elementos *rich text*, como parágrafos, equações, figuras, links e etc. Esses documentos podem ser usados para fazer a análise de dados e exibir seus resultados (JUPYTER/IPYTHON NOTEBOOK QUICK START GUIDE, 2015).

O Jupyter também possui um *Dashboard* (o painel de controle do *notebook*) que mostra arquivos locais e permite abrir documentos *notebook* possibilitando sua exibição, edição e execução (JUPYTER/IPYTHON NOTEBOOK QUICK START GUIDE, 2015).

A vantagem de se usar o Jupyter Notebook é que, além de executar o código, ele o armazena e também a sua saída no documento *notebook*.

Quando o documento é salvo, ele é enviado do navegador para o servidor do *notebook*, que o salva no disco como um arquivo JSON com uma extensão .ipynb (JUPYTER DOCUMENTATION, 2015).

3 DESENVOLVIMENTO

Neste capítulo é apresentada a implentação da solução de reconhecimento facial e detalnes de seu funcionamento com explicações detalhadas sobre suas funções mais importantes. O software foi desenvolvido com a ferramenta de programação *Jupyter Notebook*.

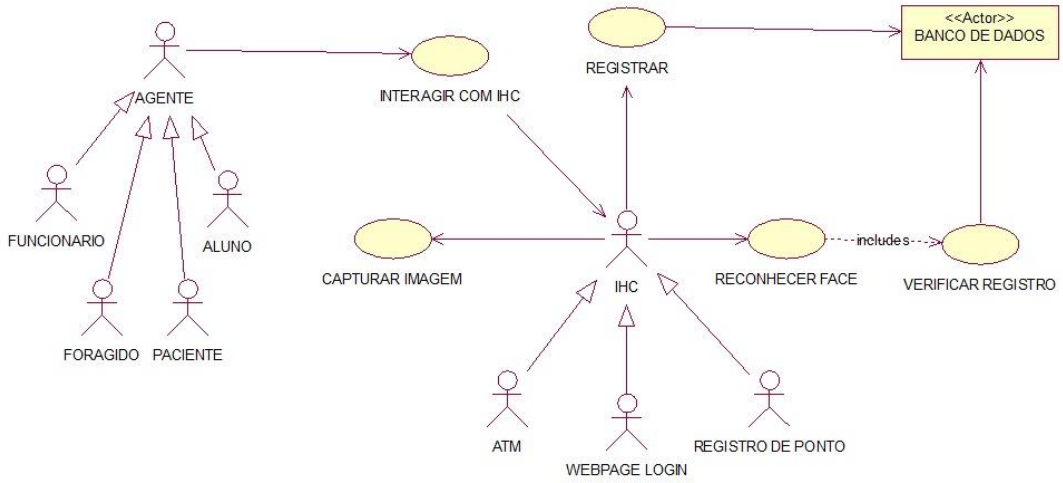
3.1 DESCRIÇÃO DA SOLUÇÃO TECNOLÓGICA

Foi utilizada uma Rede Neural Convolucional já treinada para o reconhecimento facial. A base de dados é um arquivo binário contendo contendo os dados das pessoas cadastradas na base de dados bem como a referência para suas imagens (o caminho de cada imagem no disco). As imagens com as faces das pessoas são guardadas em um diretório */imagens*.

O sistema consegue identíficar uma pessoa comparando sua foto com as fotos guardadas na base de dados, sendo que existe na base apenas uma foto de cada pessoa registrada no banco. Com o uso da técnica *One-Shot Learning* com redes neurais siamesas, o sistema consegue identificar rapidamente se a pessoa está ou não na base de dados, para isso usamos um hiperparâmetro ALPHA no vaor de X o que significa que o sistema considera que uma pessoa existe na base de dados se a distância da imagem de sua foto for menor ou igual à uma das imagens guardads no sistema.

3.2 DIAGRAMA DE CASOS DE USO

Figura 11 - Diagrama de casos de uso do sistema



Fonte: O Autor

3.3 RECONHECIMENTO FACIAL APLICADO A UM SISTEMA DE LOGIN

Este software tem o objetivo de solucionar o problema do login de usuário pela abordagem de reconhecimento facial.

O Reconhecimento facial se depara comumente com dois problemas:

- **Verificação Facial** - "esta é a pessoa reivindicada?". Por exemplo, num sistema onde uma pessoa precisa usar um cartão de acesso em uma catraca, uma câmera apontada para o rosto da pessoa pode verificar se a pessoa que está usando o cartão é a pessoa correta (o dono do cartão de acesso). Um celular que desbloqueia usando seu rosto também está usando a verificação facial. Este é um problema de correspondência 1:1.

- **Reconhecimento Facial** - "quem é essa pessoa?" Por exemplo, ao invés de usar um cartão de acesso, uma pessoa só precisaria ter sua face capturada por uma câmera, o sistema então compara a imagem captada com as imagens de rostos registradas na base de dados, se a pessoa existir na base, será reconhecida e poderá passar pela catraca. Este é um problema de correspondência de 1:K.

Utilizamos uma Rede Neural que codifica a imagem de um rosto em um vetor de 128 números. Comparando tais vetores, pode-se determinar se duas imagens correspondem à mesma pessoa.

No desenvolvimento desta aplicação foram abordados:

- A implementação da função de erro tripla.
- Foi usado um modelo pré-treinado de rede neural para mapear imagens em um vetor codificado de 128 posições.
- Essas codificações são então usadas para realizar a verificação facial e o reconhecimento facial.

O código abaixo carrega as bibliotecas necessárias para o projeto. A maioria são módulos do framework de *Machine Learning* Keras e bibliotecas úteis da linguagem Python.

```
# Importando módulos

from keras.models import Sequential
from keras.layers import Conv2D, ZeroPadding2D, Activation, Input,
concatenate
from keras.models import Model
from keras.layers.normalization import BatchNormalization
from keras.layers.pooling import MaxPooling2D, AveragePooling2D
from keras.layers.merge import Concatenate
from keras.layers.core import Lambda, Flatten, Dense
from keras.initializers import glorot_uniform
from keras.engine.topology import Layer
from keras import backend as K
K.set_image_data_format('channels_first')
import cv2
import os
import numpy as np
from numpy import genfromtxt
import pandas as pd
```

```
import tensorflow as tf
from fr_utils import *
from inception_blocks_v2 import *

%matplotlib inline    # Configuração necessária para o Jupyter exibir os
# gráficos gerados pela biblioteca matplotlib.
%load_ext autoreload
%autoreload 2

np.set_printoptions(threshold=np.nan)
```

3.3.1 Codificando imagens de rostos em um vetor de 128 posições

3.3.1.1 Usando uma ConvNet para calcular codificações

O modelo de rede neural usa muitos dados e muito tempo para treinar.

Como dito anteriormente, foi utilizado um modelo de rede neural pré-treinado. Nós apenas precisamos carregar os parâmetros de peso treinados pelo modelo.

Mas é interessante saber que:

- Esta rede recebe imagens RGB de 96 x 96 dimensões em sua entrada. Especificamente, a entrada é um rosto (ou um conjunto de imagens de rostos) como um tensor de dimensões $(m, n_C, n_H, n_W) = (m, 3, 96, 96)$, onde 'm' é o número de exemplos de treino, o '3' corresponde às três camadas RGB e 96 x 96 é a dimensão das imagens.
- Sua saída é uma matriz de $(m, 128)$ que codifica cada imagem de rosto de entrada em um vetor de 128 posições.

O código a seguir cria uma instância do modelo para imagens de rostos.

```
# Instanciando o modelo
FRmodel = faceRecoModel(input_shape=(3, 96, 96))
```

Usando uma camada de 128 neurônios completamente conectados como sua última camada, o modelo garante que a saída será um vetor codificado de tamanho 128. Então essa codificação pode ser usada para comparar duas imagens de rostos. Para uma codificação ser boa é preciso que:

- A codificações de duas imagens da mesma pessoa sejam muito parecidas uma com a outra.
- As codificações de duas imagens de diferentes pessoas sejam muito diferentes.

A função de erro tripla formaliza isso, e tenta "empurrar" as codificações das duas imagens da mesma pessoa (Âncora e Positiva) o mais próximo possível uma da outra, enquanto "puxa" as codificações de imagens de diferentes pessoas (Âncora e Negativa) para mais longe uma da outra.

3.3.2 A Função de Erro Tripla

Para uma imagem x, denotamos sua codificação f (x), onde f é a função computada pela rede neural.

O treino usa triplas de imagens (A, P, N) onde:

- A é uma imagem "Âncora" -- a imagem de uma pessoa.
- P é uma imagem "Positiva" -- uma imagem da mesma pessoa da imagem âncora.
- N é uma imagem "Negativa" -- uma imagem de uma pessoa diferente da pessoa da imagem âncora.

Essas triplas vêm do nosso conjunto de dados de treino. Nós escrevemos $(A_{(i)}, P_{(i)}, N_{(i)})$ para denotar o i-ésimo treino de exemplo.

O Gradiente Descendente deve minimizar a seguinte função de erro tripla:

$$\mathcal{J} = \sum_{i=1}^{m} \Big[\underbrace{|| f(A^{(i)}) - f(P^{(i)}) ||_2^2}_{(1)} - \underbrace{|| f(A^{(i)}) - f(N^{(i)}) ||_2^2}_{(2)} + \alpha \Big]_+$$

Aqui, usa-se a notação "[z]+[z]+" para definir max (z ,0), deve-se notar que:

- O primeiro termo (1) é a distância quadrada entre a âncora "A" e a imagem positiva "P" para uma dada tripla; para que as imagens sejam iguais essa distância deve ser pequena.
- O segundo termo (2) é a distância quadrada entre uma âncora "A" e a imagem negativa "N" para uma dada tripla, essa distância deve ser relativamente grande.
- α é a margem. É um hiperparâmetro escolhido manualmente. Usamos α=0.2α=0.2.

O código a seguir define a função de erro tripla:

```
def triplet_loss(y_true, y_pred, alpha=0.2):
    """
    Implementação da função de erro tripla.

    Argumentos:
    y_true -- rótulos true (verdadeiros), necessários quando se define uma
perda em Keras, não é necessário nesta função.
    y_pred -- lista Python contendo três objetos:
            âncora -- as codificações para uma imagem de âncora, com as
dimensões (None, 128)
            positiva -- as codificações para imagens positivas, com as
dimensões (None, 128)
            negativa -- as codificações para imagens negativas, com as
dimensões (None, 128)

    Returns:
    loss -- número real, valor de perda (erro)
    """

    anchor, positive, negative = y_pred[0], y_pred[1], y_pred[2]

    # Computa a distância entre a âncora e a positiva
    pos_dist = tf.reduce_sum(tf.square(anchor - positive), axis=-1)

    # Computa a distância entre a âncora e a positiva
    neg_dist = tf.reduce_sum(tf.square(anchor - negative), axis=-1)
```

```
    # subtrai as duas distâncias anteriores e adiciona o alpha
    basic_loss = pos_dist - neg_dist + alpha

    # Pega o máximo entre basic_loss e 0.0
    loss = tf.reduce_sum(tf.maximum(basic_loss, 0.0))

    return loss
```

3.3.3 Carregando o modelo treinado

No código seguinte nós carregamos um modelo previamente treinado.

```
FRmodel.compile(optimizer='adam', loss=triplet_loss, metrics=['accuracy'])
load_weights_from_FaceNet(FRmodel)
```

Na linha acima, foi carregado um modelo de rede neural que usa o algoritmo de otimização Adam, ele é uma versão melhorada do gradiente descendente, sua implementação é por conta do framework Tensorflow. Também foi passada como parâmetro a função triplet_loss que definimos anteriormente.

3.4 APLICANDO O MODELO

Um dos problemas que se pode resolver com este modelo, é o da Verificação facial. Por exemplo: uma pessoa com um cartão de acesso tentando passar por uma catraca para ter acesso a algum prédio. A câmera capta a imagem do rosto da pessoa e verifica se essa pessoa que está usando o carão é a pessoa correta, ou seja, sem a necessidade de um ser humano para checar isso.

3.4.1 Verificação Facial

O código a seguir funciona como uma base de dados simples contendo um vetor codificado para cada pessoa com permissão para entrar (passar pela catraca). Para gerar essa codificação foi usado `img_to_encoding(image_path, model)` que basicamente executa o algoritmo *forward propagation* do modelo na imagem específica.

A base de dados é representada por um dicionário Python. Este dicionário mapeia cada nome de pessoa para um vetor codificado de 128 posições, ou seja, para cada chave "nome" o valor é um vetor codificado de uma imagem.

É importante lembrar também que as imagens são todas da mesma dimensão, 96 x 96 pixels.

```
database = {}
database["patterson"] = img_to_encoding("images/patterson.jpg", FRmodel)
database["marcelo"] = img_to_encoding("images/marcelo.jpg", FRmodel)
database["alexandre"] = img_to_encoding("images/alexandre.jpg", FRmodel)
```

Na sequência é implementada a função responsável por verificar se a imagem captada por uma câmera é realmente a pessoa dona do cartão de acesso. Para isso foram implementados os seguintes passos computamos a codificação da imagem de entrada (a imagem captada pela câmera), em seguida calculamos a distância entre essa codificação e a codificação da imagem de identidade (a imagem do dono do cartão de acesso), por último informamos se a pessoa pode ou não passar, isso se a distância for menor que 0.7 (70%), do contrário o acesso é negado.

```
def verify(image_path, identity, database, model):
    """
    Função que verifica se a pessoa na imagem "imago path" é a pessoa
correta “identity”.
```

```
    Argumentos:
    image_path - caminho para uma imagem
    identity - string, nome da pessoa que se deseja verificar a identidade.
Precisa ser uma pessoa registrada no sistema (ou seja, existir na base de
dados).
    database - dicionário python que mapeia os nomes das pessoas permitidas
(strings) para seus vetores codificados.
    model - o modelo da rede neural em Keras.

    Retorno:
    dist - a distância entre img_path e a imagem "identity" na base de
dados.
    door_open -- True, se a pessoa é permitida. False caso contrário.
    """

    # Computa a codificação da imagem
    encoding = img_to_encoding(image_path, model)

    # Computa a distância entre a imagem da câmera e a imagem da base
    dist = np.linalg.norm(encoding - database[identity])

    # Libera o acesso se a distância for menor que 0.7, do contrário nega o
acesso.
    if dist < 0.7:
        print("Olá " + str(identity) + ", bem-vindo!")
        door_open = True
    else:
        print("Você não é " + str(identity) + ", acesso negado.")
        door_open = False

    return dist, door_open
```

No exemplo a seguir, Patterson está tentando passar pela catraca. A câmera captura sua imagem ("images/camera_1.jpg"). O sistema deve decidir se ele pode ou não passar:

Figura 12 - Imagem de teste.

Fonte: O Autor

A função verify() criada anteriormente verifica se a imagem captada corresponde ao verdadeiro dono do cartão, ou seja, se é Patterson usando seu cartão de acesso correto.

```
# Verifica se a pessoa correta está solicitando acesso
verify("images/camera_1.jpg", "patterson", database, FRmodel)

>> Olá patterson, bem-vindo!
>> (0.6338978; True)
```

A função exibe uma mensagem de saudação, pois a pessoa solicitando acesso é uma pessoa permitida e está com o cartão de acesso correto. Além disso a função também retorna o resultado (0.6338978, True) o que significa que a distância da imagem capturada pela câmera para a imagem cadastrada na base é menor que 70% e, portanto, trata-se da mesma pessoa.

Para fins de comparação, a imagem cadastrada na base de dados é essa:

Figura 13 - Imagem de Âncora.

Fonte: O Autor

3.4.2 Reconhecimento Facial

No Reconhecimento Facial queremos que o sistema reconheça uma pessoa apenas capturando uma imagem do seu rosto, para isso é preciso comparar a codificação da imagem capturada com as codificações das imagens das pessoas cadastradas na base. Se uma das imagens tiver uma distância

menor ou igual a 0.7, então a pessoa da imagem capturada existe na base e, portanto, tem acesso liberado.

O código a seguir define uma função que faz esse trabalho. Primeiro é computada a codificação da imagem capturada, em seguida a função procura qual das imagens codificadas da base possui a menor distância comparada à codificação da imagem capturada.

```
def who_is_it(image_path, database, model):
    """
    Implementa o Reconhecimento Facial.

    Argumentos:
    image_path -- caminho para a imagem no disco
    database -- base de dados contendo codificações de imagem, juntamente
com o nome da pessoa na imagem
    model -- instancia do modelo de rede neural em Keras

    Returns:
    min_dist -- a distância mínima entre a codificação da image_path e as
codificações encodings da base
    identity -- string, o nome predizido para a pessoa em image_path
    """

    # Codifica a imagem capturada
    encoding = img_to_encoding(image_path, model)

    # Inicializa a menor distância "min_dist" com um valor muito grande,
100.
    min_dist = 100

    # Encontra a imagem com a codificação mais próxima da imagem capturada
    # Itera sobre o dicionário da base de dados obtendo as chaves (names) e
as codificações (db_enc).
    for (name, db_enc) in database.items():

        # Computa a distância entre a codificação da imagem capturada e a
codificação acorrente da base de dados
        dist = np.linalg.norm(encoding - db_enc)

        # Se a distância for menor que a distância mínima, então passa a
ser a nova distância mínima,
        # e a identidade passa a ser o nome corrente
        if dist < min_dist:
            min_dist = dist
            identity = name

    if min_dist > 0.7:
        print("Não existe na base de dados.")
    else:
        print("Olá " + str(identity) + ", a distância é " + str(min_dist))

    return min_dist, identity
```

No código abaixo a função who_is_it() verifica se Patterson está na base de dados.

```
# Verifica se a imagem capturada pertence a uma pessoa registrada na base.
who_is_it("images/camera_3.jpg", database, FRmodel)

>> Olá patterson, a distância é 0.6338978
>> (0.6338978, 'patterson')
```

A função exibe uma mensagem de saudação e informa qual foi a menor distância encontrada para a imagem capturada a imagem existente na base.

4 CONSIDERAÇÕES FINAIS

Este trabalho apresentou um estudo sobre sistemas de reconhecimento facial, abordando desde a parte teórica até os passos necessários para o desenvolvimento de um programa computacional baseado nos métodos estudados. Foi verificado que para a implementação de um sistema de reconhecimento facial são necessários algoritmos robustos, que possam lidar com o problema de reconhecer faces mesmo com variação de iluminação e posição do indivíduo. Constituindo-se de um método de verificação facial e de um de reconhecimento facial ambos capazes de extrair características, o sistema desenvolvido é capaz de reconhecer faces de forma rápida e com boa precisão. Para a detecção facial foi usada a método de Redes Convolucionais, por ser o tipo de rede neural mais adequado para lidar com imagens. Os métodos de verificação e reconhecimento facial fizeram uso da técnica *One-Shot Learning* com uma Rede Siamesa treinada para detectar distâncias entre imagens de faces humanas. Notamos que o sistema tende a detectar corretamente uma pessoa cuja imagem esteja cadastrada numa base de dados, porém, possui dificuldades se a pessoa estiver muito próxima da câmera ou muito longe.

REFERÊNCIAS

ADAMS, A.; SASSE, M. A. Users are not the enemy: Why users compromisse security mechanisms and how to take remedial measures. **Communications of the ACM**, 1999. Disponível em: <http://discovery.ucl.ac.uk/20247/>. Acesso em: 04 set. 2017.

AGRAWAL, N. J.; LADHAKE, S. A. A Review on Anti Theft Mechanism through Face Recognition. **International Journal of Advanced Research in Computer Engineering & Technology**, 2016. Disponível em: <http://ijarcet.org/wp-content/uploads/IJARCET-VOL-5-ISSUE-2-338-341.pdf>. Acesso em: 04 set. 2017.

ANYOHA, Rockwell. The History of Artificial Intelligence. **Harvard University**, [S.L], ago. 2017. Disponível em: <http://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/>. Acesso em: 11 mar. 2018.

ARAÚJO, Juarez. Os avanços da inteligência artificial para o bem da humanidade. **Inforchannel**, [S.L], mai. 2017. Disponível em: <https://inforchannel.com.br/2017/05/29/os-avancos-da-inteligencia-artificial-para-o-bem-da-humanidade/>. Acesso em: 11 mar. 2018.

BERGER, Christoph. Perceptrons - the most basic form of a neural network. **Applied Go Go beyond the Go tutorials**, [S.L], set. 2016. Disponível em: <https://appliedgo.net/perceptron//?src=t>. Acesso em: 01 abr. 2018.

BOYDE, Gustavo. **A importância de criar senhas seguras**. 2016. Disponível em: <http://computerworld.com.br/importancia-de-criar-senhas-seguras>. Acesso em: 19 nov. 2017.

CANALTECH. **Mastercard lança solução de pagamentos com biometria e reconhecimento facial**. 2016. Disponível em: <https://canaltech.com.br/seguranca/mastercard-lanca-solucao-de-pagamentos-com-biometria-e-reconhecimento-facial-84953/>. Acesso em: 05 nov. 2017.

CANALTECH. 2017. **Reconhecimento facial deve substituir senhas e até cartões de crédito.** Disponível em: <https://canaltech.com.br/software/reconhecimento-facial-deve-substituir-senhas-e-ate-cartoes-de-credito-101908/>. Acesso em: 05 nov. 2017.

CASTROUNIS, Alex. Artificial Intelligence, Deep Learning, and Neural Networks, Explained. **KDnuggets**, [S.L], out. 2016. Disponível em: <https://www.kdnuggets.com/2016/10/artificial-intelligence-deep-learning-neural-networks-explained.html>. Acesso em: 11 mar. 2018.

CHOLLET, François. **Deep learning with python**. 1 ed. [S.L.]: Manning Publications, 2017. 384 p.

COREA, Francesco. A Brief History of Artificial Intelligence. **KDnuggets**, [S.L], abr. 2017. Disponível em: <https://www.kdnuggets.com/2017/04/brief-history-artificial-intelligence.html>. Acesso em: 11 mar. 2018.

DEEPLEARNING. **Convolutional Neural Networks (LeNet)**. Disponível em: <http://deeplearning.net/tutorial/lenet.html>. Acesso em: 06 mai. 2018.

DEEPLEARNING4J. **Introduction to Deep Neural Networks (Deep Learning)**. Disponível em: <https://deeplearning4j.org/neuralnet-overview#concept>. Acesso em: 13 mar. 2018.

DESHPANDE, Adit. **A beginner's guide to understanding convolutional neural networks,** [S.L.], jul. 2016. Disponível em: <https://adeshpande3.github.io/a-beginner's-guide-to-understanding-convolutional-neural-networks/>.Acesso em: 06 mai. 2018.

DOUKKALI, Firdaouss. One-Shot Learning: Face Recognition using Siamese Neural Network. **Towards Data Science**, [S.L], dez. 2017. Disponível em: <https://towardsdatascience.com/one-shot-learning-face-recognition-using-siamese-neural-network-a13dcf739e>. Acesso em: 06 mai. 2018.

DW. **Apple lança iPhone X, com reconhecimento facial e sem botão**. 2017. Disponível em: <http://www.dw.com/pt-br/apple-lan%C3%A7a-iphone-x-com-reconhecimento-facial-e-sem-bot%C3%A3o/a-40477008>. Acesso em: 05 nov. 2017.

FACURE, Matheus. Funções de Ativação - Entendendo a importância da ativação correta nas redes neurais. **Quinhentos Nove**, [S.L], jul. 2017. Disponível em: <https://matheusfacure.github.io/2017/07/12/activ-func/>. Acesso em: 06 mai. 2018.

FETTER, Vanessa. **Dicas para criar senhas seguras**. 2013. Disponível em: <https://www.hostgator.com.br/blog/dicas-para-criar-senhas-seguras>. Acesso em: 19 nov. 2017.

GORMAN, Benjamin. Introduction To Neural Networks. **GormanAnalysis**, [S.L], nov. 2017. Disponível em: <https://gormanalysis.com/introduction-to-neural-networks/>. Acesso em: 12 mai. 2018.

GRANATYR, Jones. **IA Forte x IA fraca**. Inteligência Artificial Expert, [S.L], jan. 2017. Disponível em: <http://iaexpert.com.br/index.php/2017/01/17/ia-forte-x-ia-fraca/>. Acesso em: 12 mar. 2018.

GREENEMEIER, Larry. 20 Years after Deep Blue: How AI Has Advanced Since Conquering Chess. **Scientific American**, [S.L], fev. 2017. Disponível em: <https://www.scientificamerican.com/article/20-years-after-deep-blue-how-ai-has-advanced-since-conquering-chess/>. Acesso em: 12 mar. 2018.

GUGELMIN, Felipe. Entenda a importância da inteligência artificial e como ela molda o futuro. **Tecmundo**, [S.L], abr. 2016. Disponível em: <https://www.tecmundo.com.br/inteligencia-artificial/103793-inteligencia-artificial-importante-ela-molda-nosso-futuro.htm>. Acesso em: 12 mar. 2018.

GUPTA, Harshvardhan. One Shot Learning with Siamese Networks in PyTorch. **Hackernoon**, [S.L], jul. 2017. Disponível em: <https://hackernoon.com/one-

shot-learning-with-siamese-networks-in-pytorch-8ddaab10340e>. Acesso em: 06 mai. 2018.

GRAY, M. **Urban Surveillance and Panopticism:** will we recognize the facial recognition society?. Surveillance & Society, Ontario, v. 1, n. 3, p. 314-330, set. 2002. Disponível em: <https://ojs.library.queensu.ca/index.php/surveillance-and-society/article/view/3343>. Acesso em: 04 nov. 2017.

HAMANN, Renan. **Por que as pessoas ainda usam senhas péssimas na internet?**. 2017. Disponível em: <https://www.tecmundo.com.br/seguranca/113490-por-que-pessoas-ainda-usam-senhas-pessimas-internet.htm>. Acesso em: 19 nov. 2017.

HAYKIN, Simon. **Neural networks and learning machines**: 3. [S.L.]: Pearson, 2008. 936 p.

JUPYTER/IPYTHON NOTEBOOK QUICK START GUIDE. **What is the Jupyter Notebook?**. Disponível em: <http://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#what-is-the-jupyter-notebook>. Acesso em: 25 mar. 2018.

JUPYTER DOCUMENTATION. **How IPython and Jupyter Notebook work**. Disponível em: <http://jupyter.readthedocs.io/en/latest/architecture/how_jupyter_ipython_work.html#notebooks>. Acesso em: 25 mar. 2018.

KERAS DOCUMENTATION. **Keras**: The Python Deep Learning library. Disponível em: <https://keras.io/>. Acesso em: 06 mai. 2018.

KOSKELA, H. Video surveillance, gender, and the safety of public urban space: 'Peeping Tom' goes high tech?. **Urban Geography**, 2002. Disponível em: <http://www.tandfonline.com/doi/abs/10.2747/0272-3638.23.3.257>. Acesso em: 04 set. 2017.

MACÊDO, Diego. **Técnicas para quebrar uma senha**. 2017. Disponível em: <https://www.diegomacedo.com.br/tecnicas-para-quebrar-uma-senha/>. Acesso em: 19 nov. 2017.

MARR, Bernard. What Is The Difference Between Artificial Intelligence And *Machine Learning*?. **Forbes**, [S.L], p. 1, jun. 2016. Disponível em: <https://www.forbes.com/sites/bernardmarr/2016/12/06/what-is-the-difference-between-artificial-intelligence-and-machine-learning/#12e63452742b>. Acesso em: 11 mar. 2018.

MARTELLI, Alex. **Python in a Nutshell: A Desktop Quick Reference**. 2 ed. [S.L.]: O'Reilly Media, 2006. 738 p.

MATH WORKS. **What Is Deep Learning? 3 things you need to know**. Disponível em: <https://www.mathworks.com/discovery/deep-learning.html>. Acesso em: 11 mar. 2018.

MAYO, Matthew. 7 Steps to Mastering Deep Learning with Keras. **KDnuggets**, [S.L], out. 2017. Disponível em: <https://www.kdnuggets.com/2017/10/seven-steps-deep-learning-keras.html>. Acesso em: 06 mai. 2018.

MCCARTHY, John. WHAT IS ARTIFICIAL INTELLIGENCE?. **Computer Science Department Stanford University**, [S.L], nov. 2007. Disponível em: <http://jmc.stanford.edu/articles/whatisai/whatisai.pdf>. Acesso em: 11 mar. 2018.

MITCHELL, Tom M.. **Machine Learning**. 1 ed. [S.L.]: McGraw-Hill Science/Engineering/Math, 1997. 432 p.

NARULA, Gautam. Everyday Examples of Artificial Intelligence and *Machine Learning*. **Techemergence**, [S.L], jan. 2018. Disponível em: <https://www.techemergence.com/everyday-examples-of-ai/>. Acesso em: 12 mar. 2018.

NELLI, Fabio. **Python Data Analytics**: Data Analysis and Science Using Pandas, matplotlib, and the Python Programming Language. 1 ed. [S.L.]: Appress, 2015. 337 p.

PRASANNA, Narasimha. Best Python libraries for Machine Learning and Data Science. **Towards Data Science**, [S.L], out. 2017. Disponível em: <https://towardsdatascience.com/best-python-libraries-for-machine-learning-and-data-science-part-1-f18242424c38>. Acesso em: 25 mar. 2018.

PORTILLA, Jose. A Beginner's Guide to Neural Networks with R!. **KDnuggets**, [S.L], ago. 2016. Disponível em: <https://www.kdnuggets.com/2016/08/begineers-guide-neural-networks-r.html>. Acesso em: 12 mai. 2018.

RAVINDRA, Savaram. How Convolutional Neural Networks Accomplish Image Recognition?. **KDnuggets**, [S.L], out. 2017. Disponível em: <https://www.kdnuggets.com/2017/08/convolutional-neural-networks-image-recognition.html>. Acesso em: 06 mai. 2018.

ROELL, Jason. From Fiction to Reality: A Beginner's Guide to Artificial Neural Networks. **Towards Data Science**, [S.L], jun. 2017. Disponível em: <https://towardsdatascience.com/from-fiction-to-reality-a-beginners-guide-to-artificial-neural-networks-d0411777571b>. Acesso em: 01 abr. 2018.

ROUSE, M. **Dictionary Attack**, 2005. Disponível em: <http://searchsecurity.techtarget.com/definition/dictionary-attack>. Acesso em: 04 set. 2017.

RUSSELL, Stuart J.; NORVIG, Peter. **Artificial Intelligence**: A Modern Approach. 1 ed. [S.L.]: Prentice Hall, Englewood Cliff, 1995. 3 p.

SAKURAI, Rafael. Implementando a estrutura de uma Rede Neural Convolucional utilizando o MapReduce do Spark. **GitHub Pages**, [S.L], dez. 2017. Disponível em: <http://rafaelsakurai.github.io/cnn-mapreduce/>. Acesso em: 13 mai. 2018.

SCIENCEDAILY. **Artificial Intelligence**. Disponível em: <https://www.sciencedaily.com/terms/artificial_intelligence.htm>. Acesso em: 11 mar. 2018.

SHALEV-SHWARTZ, Shai; BENDAVID, Shai. **Understanding *Machine Learning***: From Theory to Algorithms. 1 ed. United States of America: Cambridge University Press, 2014. 449 p.

SHARMA, Sagar. Activation Functions: Neural Networks. **Towards Data Science**, [S.L], set. 2017. Disponível em: <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>. Acesso em: 06 mai. 2018.

STOLTZFUS, Justin. Why is Python so popular in machine learning?. **Techopedia**, [S.L], [201-?]. Disponível em: <https://www.techopedia.com/why-is-python-so-popular-in-machine-learning/7/32881>. Acesso em: 25 mar. 2018.

TECHTERMS. Array. Disponível em: <https://techterms.com/definition/array>. Acesso em: 25 mar. 2018.

VINCENT, James. Artificial intelligence is going to supercharge surveillance. What happens when digital eyes get the brains to match?. **The Verge**, [S.L], jan. 2018. Disponível em: <https://www.theverge.com/2018/1/23/16907238/artificial-intelligence-surveillance-cameras-security>. Acesso em: 18 mar. 2018.

WALIA, Anish Singh. Activation functions and it's types-which is better?. **Towards data science,** [S.L.], mai. 2017. Disponível em: <https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f>.Acesso em: 06 mai. 2018.

WICKLIN, Rick. Metric Learning and Artificial Neural Networks. **Deep Learning Thesis**, [S.L], fev. 2018. Disponível em: <http://deeplearningthesis.com/2018/02/02/metric-learning-cnn.html>. Acesso em: 13 mai. 2018.

WOOD, Sam. **A Brief History of Python. Packt**, [S.L], out. 2015. Disponível em: <https://www.packtpub.com/books/content/brief-history-python>. Acesso em: 25 mar. 2018.