UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS E TECNOLOGIA PARA SUSTENTABILIDADE
DEPARTAMENTO DE FÍSICA, QUÍMICA E MATEMÁTICA

Letícia de Almeida Flauzino

Visão Computacional com Redes Neurais Convolucionais: Uma Abordagem Histórica, Teórica e Prática

Sorocaba

2024

Letícia de Almeida Flauzino

Visão Computacional com Redes Neurais Convolucionais: Uma Abordagem Histórica, Teórica e Prática

Trabalho de Conclusão de Curso apresentado ao curso de Licenciatura Plena em Matemática para obtenção do título de Licenciada em Matemática.

Orientação: Prof. Dr. Renato Fernandes Cantão

Sorocaba

2024

Flauzino, Letícia de Almeida

Visão Computacional com Redes Neurais Convolucionais: Uma abordagem histórica, teórica e prática / Letícia de Almeida Flauzino -- 2024.
33f.

TCC (Graduação) - Universidade Federal de São Carlos, campus Sorocaba, Sorocaba
Orientador (a): Prof. Dr. Renato Fernandes Cantão
Banca Examinadora: Prof. Dr. Antonio Luís Venezuela, Profa. Dra. Silvia Maria Simões de Carvalho
Bibliografia

1. Visão Computacional. 2. Redes Neurais Convolucionais. I. Flauzino, Letícia de Almeida. II. Título.

Ficha catalográfica desenvolvida pela Secretaria Geral de Informática (SIn)

DADOS FORNECIDOS PELO AUTOR

Bibliotecário responsável: Maria Aparecida de Lourdes Mariano - CRB/8 6979

FUNDAÇÃO UNIVERSIDADE FEDERAL DE SÃO CARLOS

COORDENAÇÃO DO CURSO DE LICENCIATURA EM MATEMÁTICA DE SOROCABA - CCML-So/CCTS

Rod. João Leme dos Santos km 110 - SP-264, s/n - Bairro Itinga, Sorocaba/SP, CEP 18052-780
Telefone: (15) 32298874 - http://www.ufscar.br

DP-TCC-FA nº 8/2024/CCML-So/CCTS

Graduação: Defesa Pública de Trabalho de Conclusão de Curso

Folha Aprovação (GDP-TCC-FA)

FOLHA DE APROVAÇÃO

LETÍCIA DE ALMEIDA FLAUZINO

VISÃO COMPUTACIONAL COM REDES NEURAIS CONVOLUCIONAIS: UMA ABORDAGEM HISTÓRICA, TEÓRICA E PRÁTICA

Trabalho de Conclusão de Curso

Universidade Federal de São Carlos – Campus Sorocaba

Sorocaba, 16 de setembro de 2024

ASSINATURAS E CIÊNCIAS

Cargo/Função	Nome Completo
Orientador	Prof. Dr. Renato Fernandes Cantão
Membro da Banca 1	Prof. Dr. Antonio Luís Venezuela
Membro da Banca 2	Profa. Dra. Silvia Maria Simões de Carvalho

Documento assinado eletronicamente por **Antonio Luis Venezuela**, **Docente**, em 16/09/2024, às 16:52, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Silvia Maria Simoes de Carvalho**, **Docente**, em 16/09/2024, às 19:08, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Renato Fernandes Cantao**, **Docente**, em 17/09/2024, às 09:56, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://sei.ufscar.br/autenticacao, informando o código verificador **1565497** e o código CRC **8EB77390**.

Referência: Caso responda a este documento, indicar expressamente o Processo nº 23112.025847/2024-11

SEI nº 1565497

Modelo de Documento: Grad: Defesa TCC: Folha Aprovação, versão de 02/Agosto/2019

À Tânia e em memória de José Paulo, sem os quais eu não estaria aqui, literalmente.

Agradecimentos

Agradeço primeiramente aos meus pais, Tânia e Paulo, por todos os esforços empenhados na minha criação. Sou eternamente grata e os amo infinitamente.

Ao meu irmão Rafael, por trazer tanta alegria pra minha vida, e aos meus familiares, por todo o apoio.

Ao Prof. Dr. Renato Fernandes Cantão, por ter me guiado neste caminho e acreditado em mim mesmo quando eu não acreditava.

A cada professor que já passou pela minha vida, em especial aos docentes da Universidade Federal de São Carlos. Que eu possa mudar vidas através da educação, como fizeram comigo.

Ao Matheus, por me incentivar em cada um dos meus objetivos e caminhar comigo pelas estradas da vida.

A todos os amigos e colegas que encontrei na Universidade, que, ao compartilhar conhecimentos, risadas e lágrimas, tornaram esse processo mais prazeroso.

Resumo

FLAUZINO, Letícia de Almeida. Visão Computacional com Redes Neurais Convolucionais: Uma Abordagem Histórica, Teórica e Prática. 2024. Trabalho de Conclusão de Curso (Licenciatura Plena em Matemática) – Universidade Federal de São Carlos, Sorocaba, 2024.

Este trabalho apresenta uma revisão histórica, teórica e prática do uso de redes neurais convolucionais aplicadas em visão computacional, iniciando com os principais pontos na linha do tempo do desenvolvimento da Inteligência Artificial e das Redes Neurais. Na sequência, são apresentadas a arquitetura e a descrição dos componentes de uma rede neural convolucional, passando por neurônios artificiais, funções de ativação, camada convolucional, camada de subamostragem e camada completamente conectada. As técnicas de regularização mais comuns também são abordadas. Quatro redes convolucionais de alto desempenho estão presentes neste trabalho: *LeNet-5*, *AlexNet*, *VGGNet* e *GoogleLeNet*. Por fim, utiliza-se os bancos de dados *MNIST* e *Cifar10*, para uma avaliação do desempenho de três das redes mencionadas. A rede *LeNet-5* apresenta excelentes resultados, devido à sua simplicidade e especificidade da tarefa. A rede *AlexNet* apresenta resultados medianos, afetados pela limitação computacional. Já a rede *VGGNet* apresenta resultados insatisfatórios, sendo uma rede com alto custo computacional e grandes prejuízos com as restrições necessárias para sua utilização.

Palavras-chave: visão computacional; redes neurais convolucionais.

Abstract

This paper presents a historical, theoretical, and practical review of convolutional neural networks applied to computer vision. It provides the main points in the timeline of the development of Artificial Intelligence and Neural Networks. Then, the architecture and description of the components of a convolutional neural network are introduced, going through artificial neurons, activation functions, convolutional layer, subsampling layer and fully connected layer. The most common regularization techniques are presented. Four high-performance convolutional networks are examined: LeNet-5, AlexNet, VGGNet and GoogleLeNet. Finally, the MNIST and Cifar10 databases are used to evaluate the performance of three of the mentioned networks. The LeNet-5 network presents excellent results, due to its simplicity and task specificity. The AlexNet network presents average results, affected by computational limitations. The VGGNet network presents unsatisfactory results, being a network with high computational cost and great losses with the necessary restrictions for its reproduction.

Keywords: computer vision; convolutional neural networks.

Lista de Figuras

Lista de Tabelas

Sumário

1 Introdução

1.1 Objetivo

O presente trabalho tem como objetivo apresentar uma breve revisão histórica, teórica e prática do uso de Redes Neurais Convolucionais aplicadas em Visão Computacional.

1.2 Inteligência Artificial

O mundo digital tem experimentado um rápido crescimento nas últimas décadas, trazendo ao nosso alcance uma série de novos recursos e possibilidades. Com a evolução da tecnologia, elementos que antes pareciam ficção científica tornaram-se parte do nosso cotidiano. Desde *smartphones* cada vez mais avançados, até inteligência artificial e robótica, vemos uma mudança constante em nossa forma de viver e nos relacionar com a tecnologia.

É possível notar o impacto cada vez maior da inteligência artificial em nossas vidas. Sua história remonta a décadas passadas, mas é na atualidade que estamos testemunhando o seu maior desenvolvimento e evolução.

Os avanços recentes não se limitaram à popularização de recursos tecnológicos, evidenciando-se também no desenvolvimento de novas ferramentas e ainda novas aplicações de componentes já conhecidos. Podemos citar as Redes Neurais Artificiais, que foram mencionadas pela primeira vez em 1947, sendo hoje uma vasta área de pesquisa com inúmeras aplicações.

A ideia de Inteligência Artificial (IA) já estava presente na antiguidade, onde se ambicionava a criação de máquinas que pudessem pensar e realizar tarefas da mesma forma que os seres humanos. Um exemplo é o deus Hefesto que, segundo a mitologia grega, criava humanoides e outros seres feitos de metal para ajudá-lo em suas tarefas, os chamados autômatos (CARTWRIGHT, 2023).

No entanto, foi durante a Segunda Guerra Mundial que pesquisas para este fim começaram a ganhar força, impulsionadas pela necessidade de desenvolver tecnologias e armamentos de ponta. A introdução das Redes Neurais (RN) se deu com as primeiras ideias formais de IA, quando em 1943 um artigo escrito por Warren McCulloch e Walter Pitts apresentou o funcionamento dos neurônios cerebrais e como estes poderiam auxiliar na construção das "máquinas inteligentes" (MCCULLOCH; PITTS, 1943). Com este trabalho, abriu-se caminho para o estudo das aplicações de RN em IA e do funcionamento do sistema nervoso.

Ainda fora do campo da computação, em 1949 o psicólogo especializado em neuropsicologia Donald Hebb descreveu em seu trabalho *A Organização do Comportamento* (do original em Inglês, *The Organization of Behavior*) o funcionamento dos neurônios e suas ligações, propondo que a cada ligação entre neurônios estes se fortalecem. Este conceito, conhecido como Lei de Hebb,

estrutura o aprendizado humano e serviu como base para o estudo das redes neurais artificiais (HEBB, 1949).

Em 1956 aconteceu a *Conferência de Dartmouth*, organizada por John McCarthy e considerada o marco inicial da IA. Nessa conferência, onde McCarthy cunhou o termo *Inteligência Artificial*, foi estabelecido um dos seus principais propósitos: "cada aspecto da aprendizagem ou da inteligência seja descrito com tanta precisão que uma máquina possa ser feita para simulá-lo" (MCCARTHY et al., 1955). Vários dos participantes deste congresso contribuíram para o progresso desta área, como Marvin Minsky, Allen Newell, Herb Simons e o próprio John McCarthy.

Em 1959 surge pela primeira vez o termo *Machine Learning* (em Português, Aprendizado de Máquina) com Arthur Samuel, para descrever o que, em palavras a ele atribuídas, seria "um campo de estudo que dá aos computadores a habilidade de aprender sem terem sido programados para tal" (SAMUEL, 1959).

A primeira aplicação de RN no cotidiano foi através de MADALINE, que eliminava eco nas ligações telefônicas. Ela foi desenvolvida em 1959 por Bernard Widrow e Marcian Hoff, e está em uso comercial até hoje (WIDROW; HOFF, 1960).

Entre 1966 e 1972, no Instituto de Pesquisa de Standford (SRI), foi desenvolvido um projeto de robô chamado de SHAKEY. Este projeto ressaltou grandes limitações e dificuldades da área de IA, como a capacidade de percepção, execução de tarefas e tomada de decisões. Ainda assim, foi o primeiro robô autônomo móvel e ganhou destaque por sua inovação para a época em que foi criado.

Muita pesquisa foi feita nos anos seguintes à Conferência de Dartmouth, graças a diversos investimentos por parte do governo e de organizações privadas. Porém, com as limitações tecnológicas da época, havia muita teoria e poucos avanços práticos satisfatórios. Deste modo, com a frustração das expectativas criadas sobre a IA, houve um grande corte no financiamento dos estudos. O período entre a década de 70 e o ano de 1981 foi conhecido como *o inverno da IA*, com poucos investimentos e estudos na área.

No final da década de 70 surgiram os sistemas inteligentes, que, partindo do pressuposto que as limitações da IA se davam também pela abordagem focada em resolver problemas muito grandes com métodos estatísticos, tinham como objetivo solucionar problemas muito específicos, que dependiam de conhecimento e experiência humanos. Estes sistemas armazenavam informações explicitamente, de modo que era possível criar um conjunto de regras que levasse à solução.

Por terem aplicações comerciais, os sistemas inteligentes fizeram com que a área de Inteligência Artificial voltasse a receber financiamentos. Um sistema de grande importância desenvolvido na época foi o MYCIN, criado para ser um assistente médico especializado em doenças sanguíneas. Desenvolvido por um amplo time da Universidade de Stanford, o grande diferencial desse sistema era ter o conhecimento de grandes especialistas da escola de medicina de Stanford (SHORTLIFFE; BUCHANAN, 1975).

O programa foi aperfeiçoado durante cinco anos, e continha todas as características essenciais necessárias para o bom funcionamento de um sistema inteligente. MYCIN simulava uma consulta,

fazendo perguntas sobre os sintomas e resultados de exames do paciente. Além disso, o sistema era capaz de explicar os motivos que levaram a determinado diagnóstico e a importância de cada pergunta feita, criando confiança por parte dos usuários.

Em testes, os resultados obtidos pelo MYCIN se mostraram tão bons quanto o julgamento de especialistas. No entanto, o tempo gasto em cada consulta para alimentar o sistema com as informações do paciente e a dificuldade de acesso ao programa, que funcionava através da ARPANet, inviabilizaram seu uso prático nos consultórios. Ainda assim, trata-se de um sistema distinto por demonstrar pela primeira vez a capacidade de uma IA em realizar uma tarefa com a mesma habilidade humana.

Com o desempenho do MYCIN e de outros sistemas da época, ressurgiu o interesse pela área de sistemas inteligentes. Numa época de pós-industrialização, a possibilidade de que a IA pudesse trazer novos caminhos ainda mais rentáveis reacendeu o investimento na área.

Em 1982, um artigo escrito por John Hopfield da Caltech (HOPFIELD, 1982) foi apresentado à Academia Nacional de Ciências. Com o carisma de Hopfield aliado a sua lógica matemática, foi apresentado o que as RN conseguiriam fazer, não somente na imitação do sistema nervoso, mas na prática. Hopfield sugeriu que as conexões entre os neurônios artificiais passassem a ser bidirecionais, diferente do caminho com um único sentido utilizado até então.

Em 1985 aconteceu nos Estados Unidos a primeira reunião anual de RN para Computação, promovida pelo Instituto Americano de Física. No ano seguinte, o que hoje é conhecido como *Redes de Retropropagação* foi desenvolvido por três grupos independentes, e que, apesar de ser mais lenta, produzia resultados mais precisos (WERBOS, 1970; PARKER, 1985; RUMELHART; HINTON; WILLIAMS, 1986).

Desde então, com o avanço da tecnologia de hardware, muitos novos progressos foram feitos na área de IA, com diversas aplicações práticas.

Em se tratando das RN com aplicação em reconhecimento de padrões, como texturas e contornos, a primeira delas foi desenvolvida por Kunihiko Fukushima em 1979, que a batizou de *Neocognitron* (FUKUSHIMA; MIYAKE, 1982). O Neocognitron possibilitou ao computador o reconhecimento de padrões visuais. Esta rede foi pioneira na utilização de conceitos que são utilizados até hoje em visão computacional, e que permitiram que novas variedades fossem elaboradas. Em 1989, Yann LeCun combinou a retropropagação com Redes Neurais Convolucionais (RNC) e desenvolveu um algoritmo capaz de reconhecer letras e números manuscritos (LECUN et al., 1989).

A partir de 1999, os computadores evoluíram em capacidade de processamento e quantidade de memória, além do desenvolvimento das Unidades de Processamento Gráfico (GPUs), o que fez com que a velocidade de processamento dos computadores saltasse 1000 vezes em 10 anos (MCCLANAHAN, 2011).

Outro grande passo para a aplicação das RNC em reconhecimento de imagens foi dado em 2009 por Fei-Fei Li, uma professora que criou uma base de dados gratuita na época com mais de 1 milhão de imagens apropriadamente catalogadas para serem usadas em treinamentos de

redes neurais, batizada de ImageNet (DENG et al., 2009). Atualmente, o projeto contém mais de 14 milhões de imagens.

Nos últimos anos as RN vêm se tornando cada vez mais integradas ao cotidiano, sendo alvo de investimentos, estudos, expectativas e inovações. Temos algoritmos que aprendem com a interação do usuário em redes sociais, mecanismos de busca que filtram e classificam os resultados e até mesmo assistentes pessoais controlados por voz, mostrando a grande utilidade desta tecnologia. Em novembro de 2022, o lançamento de um assistente via texto chamado ChatGPT atraiu grande atenção do público em geral pela capacidade de desenvolver conversas complexas sobre uma grande variedade de assuntos (SOLAIMAN et al., 2019).

1.3 Visão Computacional

Visão computacional é um campo da Inteligência Artificial que estuda como as máquinas podem interpretar, compreender e extrair informações de imagens e vídeos. A visão computacional usa técnicas de processamento de imagens, reconhecimento de padrões e aprendizado de máquina para solucionar problemas relacionados à visão.

O desenvolvimento das pesquisas em visão computacional teve início no começo da década de 60, quando Lawrence Roberts, que futuramente criaria a Internet, publicou a dissertação *Machine Perception of Three-Dimensional Solids* (do Inglês, Percepção de Máquina de Sólidos Tridimensionais, em tradução livre) (ROBERTS, 1963). Nele, Roberts apresenta um sistema de visão computacional capaz de perceber objetos tridimensionais a partir de imagens bidimensionais. Pouco tempo depois, Woody Bledsoe contribuiu para o reconhecimento facial com seu trabalho *A Facial Recognition Project Report* (do Inglês, Relatório de um Projeto em Reconhecimento Facial) (BLEDSOE, 1964). Baseados em modelos matemáticos, os algoritmos descritos por Bledsoe comparavam imagens de faces e determinavam se eram as mesmas ou diferentes. Em 1979, Fukushima e Miyake (1982) desenvolveram o *Neocognitron*, pioneiro na utilização de conceitos de visão computacional para reconhecimento de padrões visuais.

No início, as técnicas de visão computacional consistiam basicamente em comparar regiões de uma imagem com imagens de objetos básicos, mas novas técnicas foram sendo desenvolvidas até que nas décadas de 80 e 90 os trabalhos de Geoffrey Hinton, Yoshua Bengio e Yann LeCun com Redes Neurais possibilitaram novas aplicações.

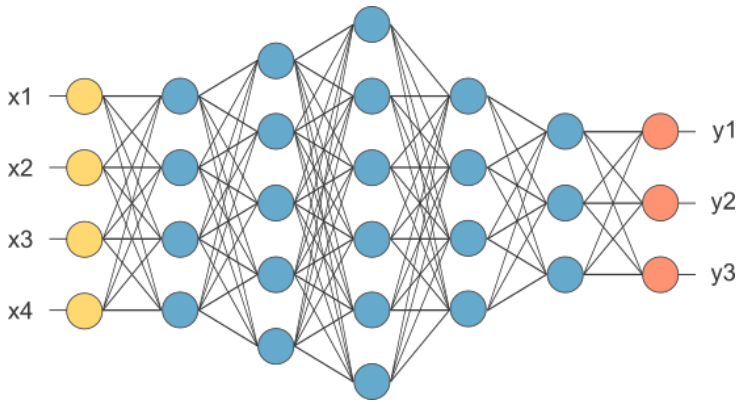
Os avanços na visão computacional são notáveis nos dias atuais, com aplicações cada vez mais amplas em diversos setores, como jogos, vigilância e segurança, monitoramento de saúde, análise de exames médicos e outros. Com grande aplicabilidade e possibilidades lucrativas, a área vem apresentando numerosas inovações, cada vez mais eficazes.

1.4 Redes Neurais

Redes Neurais Artificiais (RNA) são parte de uma área de pesquisa de Aprendizado de Máquina que visa utilizar processos similares aos que ocorrem na troca de informações dos neurônios no sistema nervoso para aproximação de funções desconhecidas dados domínio e imagem, característica usada em problemas de classificação como a visão computacional. A RNA processa as informações fornecidas, modelando os dados de modo a obter com precisão dados preditivos sobre situações que não lhe foram apresentadas. Através do sucesso ou fracasso em suas previsões, a RNA ajusta seus parâmetros para diminuir o erro, passando assim por um processo de "aprendizagem".

Uma RNA é composta de *neurônios* (ou unidades, ou ainda nós) de entrada, que recebem as informações a serem processadas; neurônios de saída, que fornecem os resultados do processamento; e neurônios escondidos, que ficam entre os de entrada e saída, onde se localizam funções de ativação que processarão os dados. A esses neurônios agrupados dá-se o nome de *camadas*. Uma RNA é composta então de uma camada de entrada, uma ou mais camadas escondidas e uma camada de saída, conforme ilustrado na Figura 1. O número de neurônios, a forma de conexão entre estes e o número de camadas escondidas em uma RNA variam de acordo com sua arquitetura e objetivos.

Figura 1 – Exemplo de configuração de uma Rede Neural, onde x_1, x_2, x_3 e x_4 representam os dados de entrada e y_1, y_2 e y_3 representam os dados de saída. Em amarelo à esquerda estão os neurônios que compõem a camada de entrada, em laranja à direita os da camada de saída e no meio, em azul, as camadas escondidas.

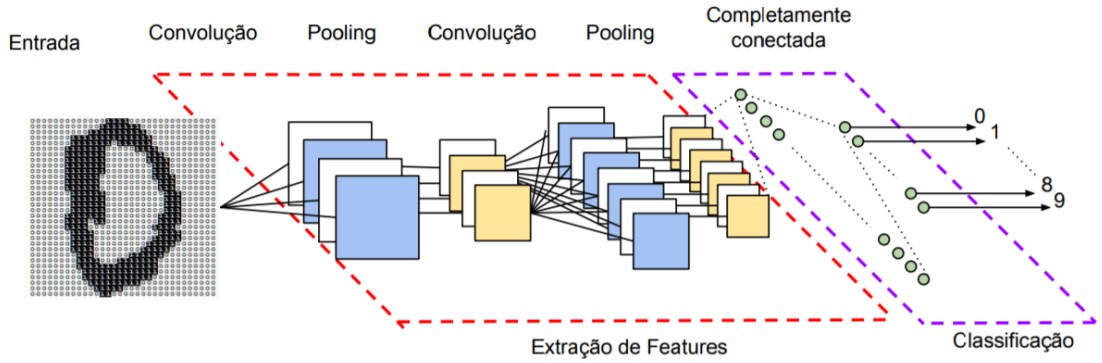


Fonte: Retirado de `http://www2.decom.ufop.br/imobilis/fundamentos-de-redes-neurais/`.

As Redes Neurais Convolucionais (RNC), baseadas no córtex visual dos seres vivos (HUBEL; WIESEL, 1968), são consideradas o estado da arte para visão computacional (ARAI; KAPOOR, 2020). Sua arquitetura tem como característica três tipos de camadas: camadas de convolução, camadas de *pooling* e camadas completamente conexas, como ilustrado na Figura 2. Uma RNC pode ter diferentes quantidades de cada um desses tipos.

A *camada de convolução*, que está sempre localizada logo após a entrada dos dados, é onde a maior parte do esforço computacional ocorre. É formada por neurônios com mapas de características, cada um deles ligado à uma região da camada anterior, chamada de campo receptivo.

Figura 2 – Uma RNC e a organização de suas camadas.



Fonte: Retirado de Vargas, Paes e Vasconcelos (2016).

Neurônios com os mesmos mapas de características possuem o mesmo peso, reduzindo a complexidade e tornando o algoritmo mais fácil de ser treinado.

A *camada de pooling* é responsável por diminuir a dimensão dos mapas de ativação, controlando o sobreajuste e reduzindo a computação necessária para o algoritmo. Um tipo muito usado é a *max pooling*, que recebe os valores máximos do mapa de ativação, de acordo com um filtro de dimensão.

As últimas camadas são normalmente do tipo *completamente conexa*, no qual todos os seus neurônios estão ligados a todos os neurônios da camada anterior. É a camada onde os dados serão classificados de acordo com as classes inicialmente fornecidas. Algumas arquiteturas mais recentes de RNC substituem a camada completamente conexa por uma camada de *pooling* (ZEILER; FERGUS, 2014).

2 Fundamentação teórica

2.1 Redes Neurais Artificiais

Nas palavras de Haykin (2001), "uma *rede neural* é uma máquina que é projetada para *modelar* a maneira como o cérebro realiza uma tarefa particular ou função de interesse". Baseadas no córtex cerebral dos seres vivos, as Redes Neurais Artificiais (RNA) procuram simular o processo de aprendizagem ocorrido no cérebro, de modo a obter um processamento de informações mais rápido e mais preciso.

As RNA são compostas por *neurônios artificiais*, que são as unidades computacionais onde o processamento de informações de fato acontece. Esses neurônios são organizados em *camadas*, que transmitem seus resultados para a camada seguinte.

Uma RNA típica apresenta três tipos de camadas: a *camada de entrada*, que recebe os dados a serem processados; uma ou mais *camadas ocultas*, que processam a informação; e a *camada de saída*, que fornece o resultado obtido (*cf.* Figura 1). A quantidade de camadas escondidas e a sua estrutura definem a arquitetura da RNA, e depende fortemente do problema a ser tratado.

Cabe um aprofundamento no estudo dos neurônios artificiais, já que esses são o elemento básico de construção de qualquer tipo de RNA.

2.1.1 Neurônios Artificiais

Os neurônios artificiais — também chamados de nós da rede — são as unidades básicas de processamento de informação. São eles que recebem os dados de entrada, extraem e processam suas características através de uma função de ativação, e passam adiante um valor de saída, que será recebido pela próxima camada (podendo inclusive ser a camada final, de saída). Há diferenças na utilidade e na importância das características representadas nos dados entrada. Para representar essa variabilidade, cada nó de entrada recebe um valor de peso, conhecido como peso de conexão ou peso sináptico, que reflete sua relevância dentro do modelo geral. Quanto maior o valor do peso atribuído, maior a importância daquela característica para o objeto (ELGENDY, 2020).

De acordo com Haykin (2001), o modelo neuronal possui três elementos básicos:

1. *Sinapses* ou *elos de conexão*, que podem variar entre valores positivos ou negativos. Um peso sináptico w_{kj} multiplica o valor x_j, na sinapse j do neurônio k, com $k, j \in \mathbb{N}$.

2. Um componente de soma responsável por adicionar os sinais de entrada, ponderados pelos pesos das sinapses correspondentes no neurônio; as operações mencionadas constituem um *combinador linear*.

3. Uma *função de ativação*, tipicamente não-linear, e que será mais explorada à frente. Essa função também pode ser conhecida como restritiva, pois limita o intervalo permitido da amplitude do sinal de saída a um valor finito. Normalmente, o intervalo da amplitude de saída de um neurônio é expresso como o intervalo fechado unitário $[0, 1]$, ou alternativamente, como $[-1, 1]$.

4. Ocasionalmente, um modelo neural também pode incorporar um viés (ou *bias*) externo, representado por b_k. Esse viés tem o efeito de aumentar ou diminuir a entrada líquida da função de ativação, dependendo se é positivo ou negativo, respectivamente.

Dessa forma, pode-se descrever um neurônio artificial matematicamente como:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \qquad \text{combinador linear} \tag{2.1}$$

e

$$y_k = \varphi(u_k + b_k), \qquad \text{função de ativação} \tag{2.2}$$

onde os sinais de entrada são representados por $x \in \mathbb{R}^m$, enquanto w_{kj} refere-se aos pesos sinápticos j associados ao neurônio k. O combinador linear utiliza esses sinais de entrada para produzir u_k, resultado intermediário da operação do neurônio u_k, e o termo b_k representa o bias adicionado a essa combinação. A função de ativação φ é aplicada, resultando em y_k, o sinal de saída do neurônio k. Apesar das diferentes modelagens possíveis para as RNA, a presença de neurônios artificiais é central em todas elas.

2.1.2 Função de Ativação

A principal escolha a se fazer para um modelo neuronal é a função de ativação utilizada. Esta função trará não-linearidade para a rede, possibilitando a aprendizagem de padrões mais complexos. Sem ela, a rede corresponderia a uma regressão linear simples (DATA SCIENCE ACADEMY, 2022).

A função de ativação, também chamada de *função de limiar*, restringe o valor da saída para um intervalo finito, de modo que a RN possa fazer classificações. As funções de ativação mais tradicionalmente usadas, *ReLu*, Sigmoide e Tangente hiperbólica, serão abordadas na sequência e mostradas na Figura 3.

ReLU

Sendo a mais utilizada atualmente em Redes Neurais (RASAMOELINA; ADJAILIA; SINČÁK, 2020), a função *ReLU*, do inglês *Rectified Linear Unit*, se popularizou devido à sua alta performance e simplicidade. Esta função pode ser definida como:

$$\text{ReLU}(x) = \max\{0, x\}. \tag{2.3}$$

Nota-se que a ligação do neurônio só é ativada caso o valor de entrada seja maior do que zero; do contrário, a função retorna o valor nulo.

Sigmoide

Sendo muito utilizada, a função sigmoide, ou função logística, é não-linear e costuma ser aplicada em contextos de previsão e probabilidade (ELGENDY, 2020), já que os valores retornados se encontram no intervalo $[0, 1]$. Pode ser escrita como:

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}. \tag{2.4}$$

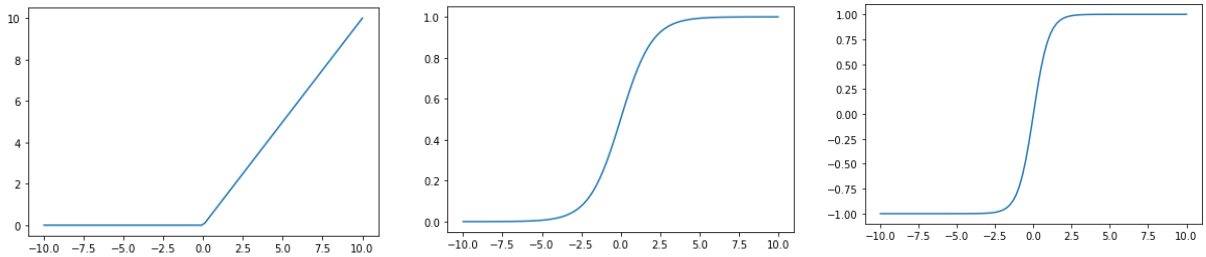
Tangente hiperbólica

A função tangente hiperbólica (tanh) é similar à função sigmoide, mas tem seus resultados variando entre -1 e 1. Por ser simétrica à origem, torna o modelo mais fácil de treinar diminuindo o erro (AGGARWAL, 2018). É descrita como:

$$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1. \tag{2.5}$$

Figura 3 – Apresentação das três funções de ativação mencionadas.

(a) ReLU. (b) Sigmoide. (c) Tangente hiperbólica tanh.



Fonte: Feito pela autora.

2.2 Redes Neurais Convolucionais

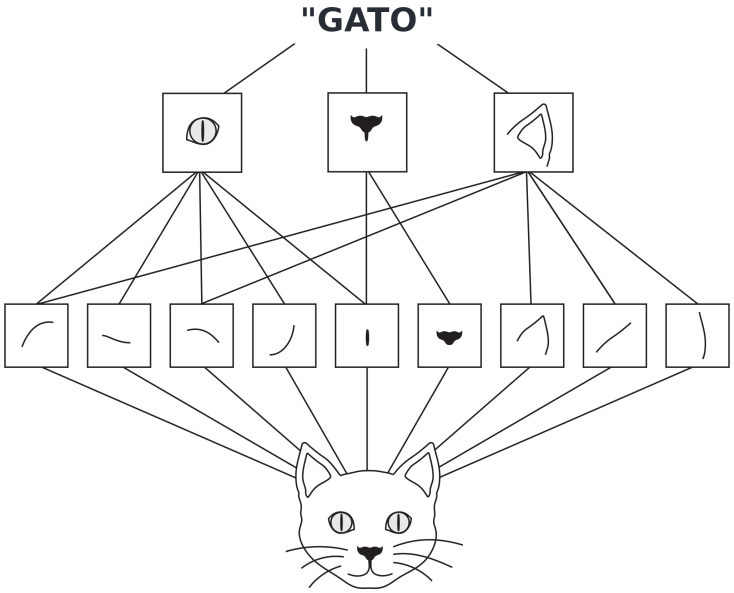
A Redes Neurais Artificiais são baseadas no córtex cerebral dos seres vivos, sua arquitetura assemelhando-se à maneira como os neurônios cerebrais se conectam e identificam informações. Nas Redes Neurais Convolucionais (RNC), o modelo aprende e extrai as características do objeto de entrada subdividindo-o em regiões menores, semelhante ao processo ocorrido no cérebro de felinos (AGGARWAL, 2018). Sua arquitetura tem como característica três tipos de camadas: camadas de convolução, camadas de *pooling* e camadas completamente conexas. O maior diferencial da RNC para as demais arquiteturas de redes neurais se dá pela característica tridimensional de suas camadas.

Chollet (2017) destaca duas propriedades das RNC:

- A RNC consegue identificar as características mesmo quando houver translação da imagem, fazendo com que o modelo seja capaz de fazer generalizações melhores com um número menor de dados de entrada.

- A cada camada de convolução subsequente, os neurônios passam a processar características mais complexas reunindo as características vizinhas, como mostra a Figura 4.

Figura 4 – Hierarquia espacial no mundo visual: a partir do conceito de "bordas" → conceito de "objetos locais" (olhos, orelhas) → finalmente chega-se aos conceitos de alto nível ("gato").



Fonte: Retirado de Chollet (2017).

2.2.1 Camada Convolucional

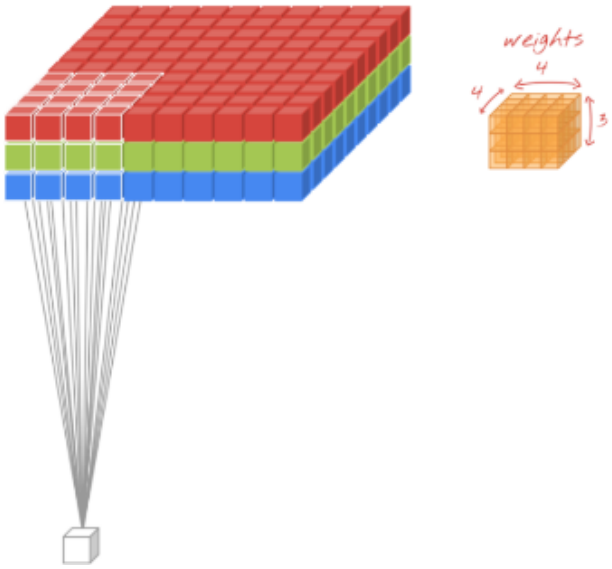
A *camada de convolução*, que é sempre a primeira camada do modelo, é onde a maior parte do esforço computacional ocorre. Com o intuito de diminuir a quantidade de conexões realizadas pela rede, a camada convolucional associa seus neurônios a regiões específicas da camada anterior, chamadas de *campos receptivos*. Dessa forma, os neurônios extraem características locais, e não globais.

Cada neurônio aplica um filtro ao seu campo receptivo, gerando um *mapa de características*, que indica a presença e a intensidade de determinado atributo naquela região. Um mapa de características de uma camada q, com dimensões $L_q \times B_q \times d_q$ é um tensor tridimensional, sendo dois eixos associados ao tamanho (altura L_q e largura B_q), e o terceiro uma dimensão de profundidade (d_q). Na primeira camada convolucional de um modelo treinado com imagens, cada valor representa um *pixel*[1] e a profundidade possui dimensão de acordo com a quantidade de informações de cor da imagem. Por exemplo, em uma imagem RGB, a dimensão é 3, representando as cores vermelho, verde e azul (*cf.* Figura 5); já em uma imagem preto-e-branco a dimensão tem profundidade igual a 1, visto que representa apenas uma escala de cinza (CHOLLET, 2017).

Neurônios com os mesmos mapas de características possuem o mesmo peso, reduzindo a complexidade e tornando o algoritmo mais fácil de ser treinado.

[1] Unidade de medida padrão para imagens.

Figura 5 – Exemplo de um filtro tridimensional.



Fonte: Adaptado de Lakshmanan, Görner e Gillard (2021).

O filtro de uma camada q é comumente quadrado em sentido espacial e possui a mesma profundidade que a camada em que se encontra. Suas dimensões são definidas como $F_q \times F_q \times d_q$, sendo comum que F_q assuma valores pequenos e ímpares, como 3 ou 5.

A convolução ocorre quando o filtro é aplicado no mapa de características correspondente, pixel a pixel, num processo de *deslizamento* (Figura 6). O filtro passa pela entrada ponto a ponto, fazendo a multiplicação no local. A soma desses valores compõe o mapa de ativação para aquele ponto (Figura 7). Dessa forma, a convolução pode ser escrita na forma de um produto escalar (TEUWEN; MORIAKOV, 2020):

$$(x \cdot w)(t) = \sum_a x(t)w(t-a), \tag{2.6}$$

sendo x os dados de entrada, t é o parâmetro de convolução e w o filtro, resultando no mapa de ativação.

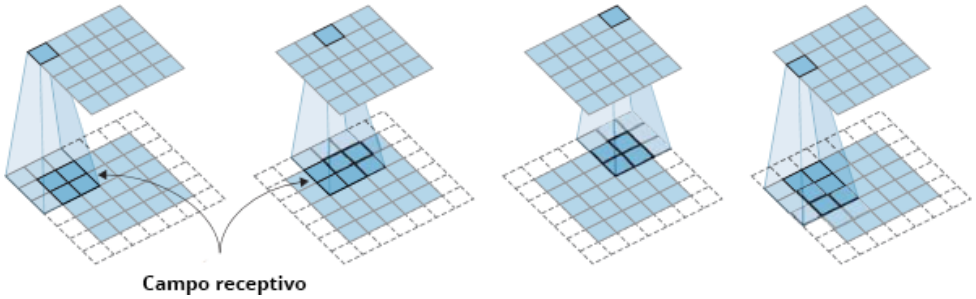
Como o foco deste trabalho é a utilização de RNCs em visão computacional, as entradas serão dadas em duas dimensões. Assim, a convolução 2D pode ser escrita como (TEUWEN; MORIAKOV, 2020):

$$(x \cdot w)(i,j) = \sum_m \sum_n x(i,j)w(i-m, j-n). \tag{2.7}$$

Padding

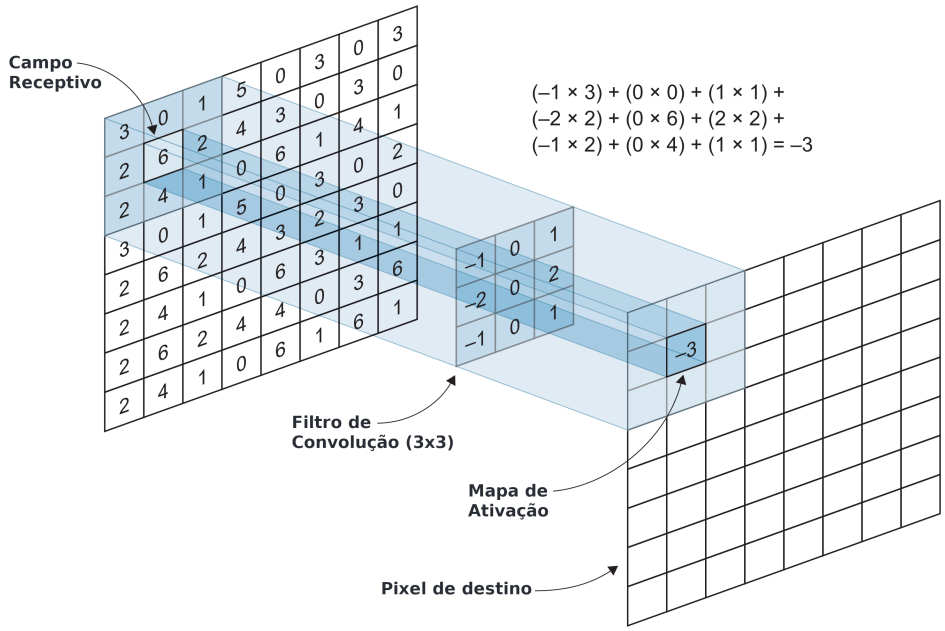
Padding, que significa *preenchimento*, ou *zero-padding* é um parâmetro utilizado para evitar que o processo de convolução resulte em um mapa de ativação menor do que o mapa de ativação recebido. Como a convolução diminui as dimensões do mapa em $((W_q - 1)/2)$, sendo W_q a dimensão do filtro w na camada q, adiciona-se $(W_q - 1)$ colunas e linhas de zeros ao redor dos dados de

Figura 6 – Exemplo de um filtro *deslizando* por um mapa de ativação.



Fonte: Adaptado de Elgendy (2020).

Figura 7 – Exemplo de um filtro sendo aplicado em seu campo receptivo, gerando um mapa de ativação.



Fonte: Adaptado de Elgendy (2020).

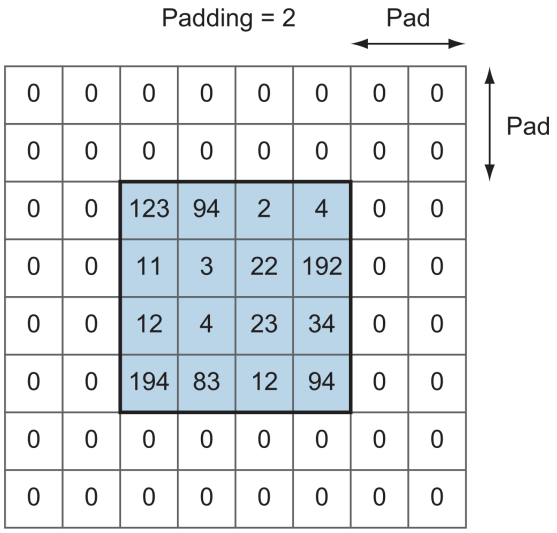
entrada, de modo que nenhuma característica importante das bordas seja perdida (AGGARWAL, 2018).

2.2.2 Camada de *Pooling*

A camada de *pooling*, também chamada de camada de *subamostragem*, é responsável por diminuir a quantidade de parâmetros a serem passados para a próxima camada. Este processo é necessário para a redução da complexidade e do custo computacional da rede, que aumenta a cada camada convolucional.

De maneira similar à camada convolucional, a camada de *pooling* trabalha em pequenas regiões do mapa de entrada, fazendo o "deslizamento" por completo. É possível utilizar a função *max pooling* ou *average pooling*, onde a primeira irá selecionar o maior valor da região onde está sendo aplicada, e a segunda irá calcular a média entre os valores. A Figura 9 exemplifica a função *max*

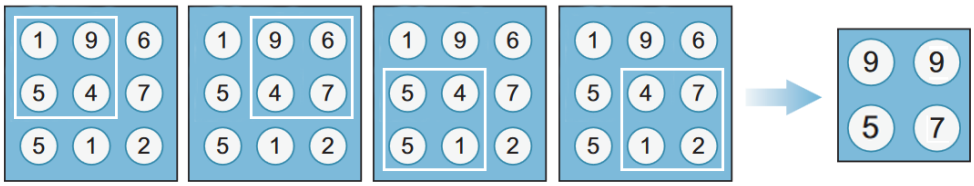
Figura 8 – Exemplo de *padding* aplicado em um mapa.



Fonte: Retirado de Elgendy (2020).

pooling, a mais comum.

Figura 9 – Uma operação de *max pooling* aplicada em um mapa 3×3, reduzindo-o para 2×2.



Fonte: Adaptado de Elgendy (2020).

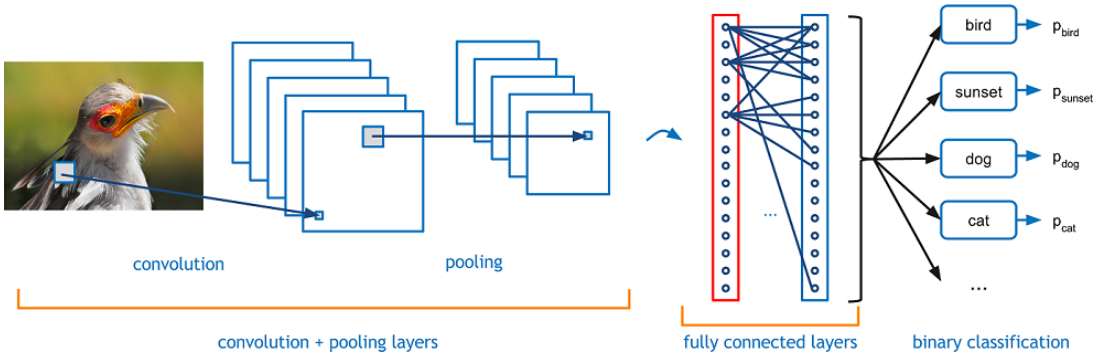
Além da redução de dimensões, a camada de *pooling* garante a invariância de translação da imagem, assegurando que características possam ser encontradas em qualquer ponto da imagem, mesmo que esta seja deslocada, rotacionada ou alongada (AGGARWAL, 2018).

2.2.3 Camada completamente conectada

A última camada de uma Rede Neural Convolucional costuma ser do tipo *camada completamente conectada*, mas em algumas arquiteturas esta pode ser substituída por uma camada de *pooling*. Um modelo pode ter uma ou mais camadas completamente conectadas ao final, variando de acordo com a aplicação e arquitetura.

Nesta camada, que também é conhecida como *camada densa*, cada neurônio se conecta a todos os neurônios da camada anterior, como nas Equações (2.1) e (2.2). É nessa etapa onde as características extraídas nas camadas convolucionais e de *pooling* anteriores serão utilizadas para a classificação dos dados.

Figura 10 – Etapas de aplicação de uma Rede Neural Convolucional.



Fonte: Adaptado de Deshpande (2016).

2.2.4 Regularização

Dois grandes problemas quando se trata do treinamento de modelos são *overfitting* e *underfitting*, ou, respectivamente, sobreajuste e subajuste. O sobreajuste ocorre quando o modelo se ajusta tão bem aos dados fornecidos que não consegue identificar padrões generalizados e realizar previsões para dados desconhecidos. Já o problema de subajuste é o contrário, quando o modelo é muito simples para o conjunto de dados e portanto, também não é capaz de fazer previsões.

Esses dois processos ocorrem em todas as RN. No começo do treinamento, a rede ainda não aprendeu o suficiente sobre as generalizações dos dados, então ocorre o *underfitting*. Com o avanço nas iterações, o modelo vai aprendendo mais e as métricas de avaliação mostram melhora. A partir de um certo ponto, no entanto, as métricas começam a decair com o desempenho no grupo de validação. Aqui começa a ocorrer o *overfitting*, onde o modelo encontrou padrões muito específicos que não são uteis para previsões. O ideal então, é a construção de um modelo que possa encontrar características que sejam relevantes nos dados.

Segundo Chollet (2017), a melhor maneira de evitar essas duas situações é obter mais dados. Como nem sempre essa solução é possível, outras técnicas podem ser utilizadas para evitar ou diminuir esses problemas. A seguir, serão apresentadas algumas dessas técnicas mais utilizadas para esse fim.

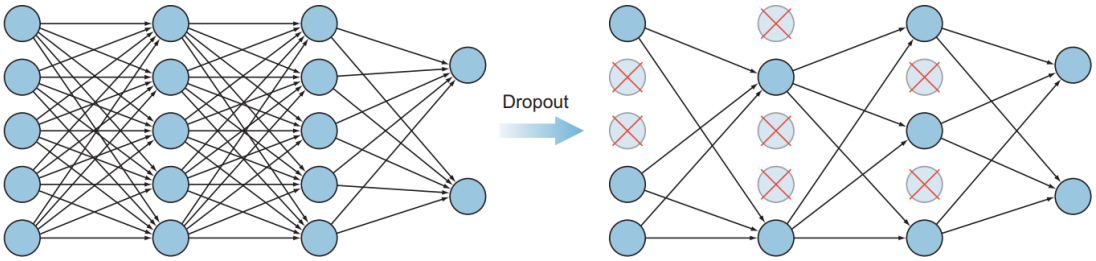
2.2.4.1 *Dropout*

Uma das técnicas mais utilizadas é a adição de camadas de *dropout*, por vezes traduzida como *desativação aleatória* (ELGENDY, 2020). Essa camada desativa aleatoriamente uma porcentagem dos neurônios da rede. A ideia é enfraquecer conexões que não sejam relevantes, forçando os neurônios a trabalharem de maneira conjunta, sem que alguns padrões se sobressaiam a outros.

2.2.4.2 Redução do modelo

Em alguns casos, o sobreajuste acontece porque a rede é muito complexa para os dados disponíveis, ou seja, tem mais camadas do que o necessário. Quanto mais camadas, mais neurônios e

Figura 11 – Aplicação da camada de *Dropout*. Alguns neurônios são desligados, forçando os demais a abandonarem algumas características.



Fonte: Retirado de Elgendy (2020).

maior capacidade de memória terá essa rede, o que possibilita que muitas características sejam armazenadas, inclusive aquelas que não fazem parte da generalização dos dados. A maneira mais simples então é reduzir o número de camadas, para que os neurônios possam se concentrar nos padrões mais relevantes.

A dificuldade, no entanto, é diminuir a rede o suficiente para evitar o sobreajuste, mas não de maneira que a rede fique mais simples do que o necessário, gerando subajuste.

2.2.4.3 Regularização L2

Esse método funciona diminuindo os valores dos pesos de cada neurônio, para que a rede se torne mais simples e com valores mais uniformes. Para isso uma penalidade, descrita na Equação (2.8), é adicionada à função de perda que, tendo um valor maior, força os neurônios a diminuírem seus pesos

$$L_2 = \lambda \times \sum \|w\|^2, \tag{2.8}$$

onde o termo λ representa um parâmetro de regularização, enquanto o vetor w contém os pesos de cada neurônio.

2.2.4.4 *Early stopping*

É possível monitorar o momento em que uma rede começa a sobreajustar aos dados observando-se o valor de perda para o conjunto de validação. O *early stopping*, (parada antecipada, em tradução livre), consiste em parar o aprendizado da rede no momento em que esse valor começar a diminuir, pois é provável que ela esteja se adaptando exageradamente ao conjunto de treino.

3 Arquiteturas avançadas de Redes Neurais Convolucionais

Neste capítulo, serão apresentadas algumas estruturas de redes Neurais Convolucionais famosas por seu desempenho em competições como a ILSVRC — *ImageNet Large Scale Visual Recognition Challenge* (do Inglês, Desafio de Reconhecimento Visual de Larga Escala da ImageNet) (RUSSAKOVSKY et al., 2014).

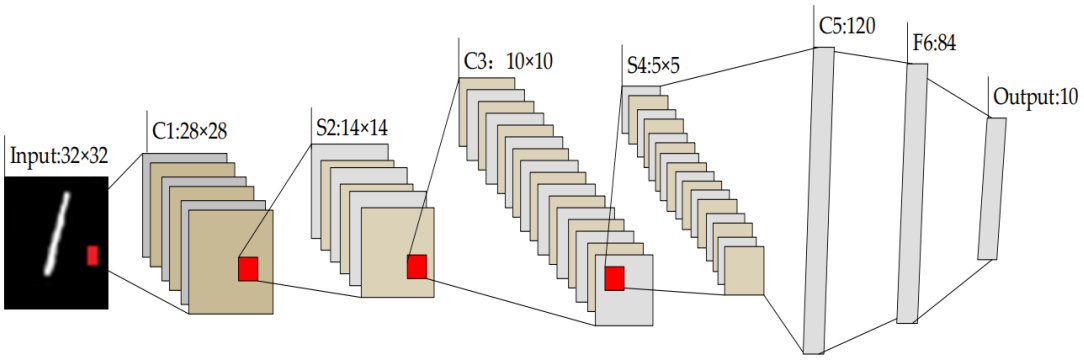
3.1 *LeNet-5*

Uma das primeiras redes convolucionais de sucesso foi a *LeNet-5* de 1998 (LECUN et al., 1998), batizada em homenagem a Yann Lecun. Seu primeiro objetivo era classificar imagens de dígitos manuscritos, tendo depois sido adaptada para uso em caixas eletrônicos, estando em uso ainda hoje em alguns locais (ZHANG et al., 2020). Por ter um objetivo específico e utilizar imagens simples, a rede apresenta bons resultados mesmo tendo poucas camadas convolucionais.

Sua arquitetura é composta por três camadas convolucionais, duas camadas de *pooling* e duas camadas completamente conectadas. Condizendo com a época em que foi desenvolvida, o modelo utiliza a função de ativação sigmoide e *average pooling* como subamostragem. É por vezes definida como a arquitetura clássica de uma RNC.

Um exemplo de implementação da arquitetura *LeNet-5* pode ser apreciada no Apêndice A.1.

Figura 12 – Arquitetura da rede *LeNet-5*. C1, C3 e C5 são camadas convolucionais, S2 e S4 são camadas de *average pooling* e a camada F6 é do tipo completamente conectada.



Fonte: Retirado de Wei et al. (2019).

A primeira camada recebe uma imagem em escala de cinza no tamanho $32 \times 32 \times 1$. Essa imagem passa então por seis filtros 5×5, resultando em uma imagem $28 \times 28 \times 6$. A segunda camada, sendo de subamostragem, passa a imagem por um filtro 2×2, resultando em uma imagem de dimensões $14 \times 14 \times 6$. Na terceira camada, há 16 filtros de tamanho 5×5, no entanto apenas 10 se conectam a cada um dos filtros da camada anterior, conforme a Tabela 1. Desse modo, o

número de conexões é reduzido de maneira assimétrica e o resultado é uma imagem de tamanho $10 \times 10 \times 16$.

Tabela 1 – Cada coluna indica qual filtro da camada S2 se conecta com determinado filtro da camada C3.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	×				×	×	×			×	×	×	×		×	×
1	×	×				×	×	×			×	×	×	×		×
2	×	×	×				×	×	×			×		×	×	×
3		×	×	×			×	×	×	×			×		×	×
4			×	×	×			×	×	×	×		×	×		×
5				×	×	×			×	×	×	×		×	×	×

Fonte: Adaptado de Lecun et al. (1998).

A quarta camada da rede é similar à segunda camada, sendo novamente de subamostragem com filtro 2×2. Assim, a imagem é reduzida para $5 \times 5 \times 16$. A quinta camada é do tipo completamente conectada, com 120 mapas de características. Já na sexta camada, também do tipo completamente conectada, são 84 mapas. Por fim, na última camada uma função probabilística apresenta a qual das 10 classes de dígitos a imagem pertence.

3.2 *AlexNet*

Vencedora do prêmio ILSVRC de 2012, o modelo *AlexNet* (KRIZHEVSKY; SUTSKEVER; HINTON, 2012) surgiu com o intuito de ser capaz de aprender funções mais complexas do que o *LeNet-5*. Apesar de apresentar uma estrutura similar, a rede *AlexNet* traz melhorias significativas: uma quantidade de parâmetros mil vezes maior e a função de ativação ReLU.

Foi com esse modelo que as Redes Neurais Convolucionais passaram a ser utilizadas em áreas mais complexas de visão computacional, pois mesmo com o sucesso da rede *LeNet-5*, esta havia sido desenvolvida e treinada com um banco de dados de pouca variação e imagens pequenas.

A rede *AlexNet* foi treinada com o banco de imagens *ImageNet* [1], lançado em 2009. Fei-Fei Li e outros pesquisadores reuniram mais de um milhão de imagens separadas em mil categorias, na intenção de que

> (...) a escala, precisão, diversidade e estrutura hierárquica do ImageNet possam oferecer oportunidades incomparáveis aos pesquisadores na comunidade de visão computacional e além dela (LI et al., 2009).

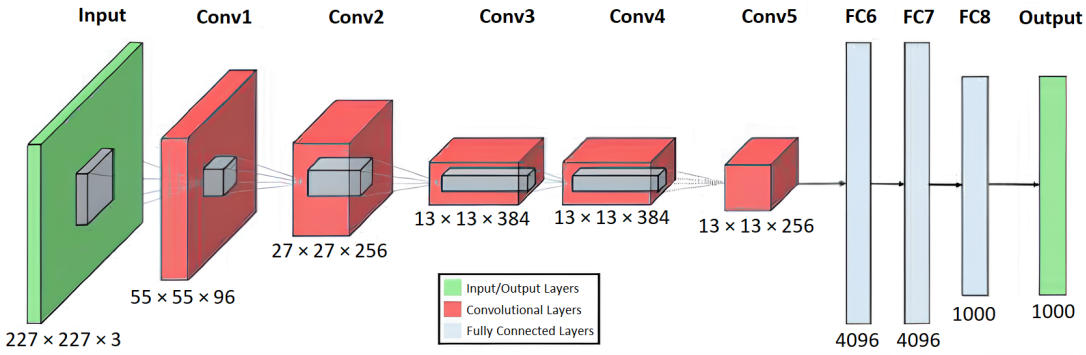
Sua arquitetura é composta por cinco camadas convolucionais, três camadas de *pooling* e duas camadas completamente conectadas. Devido ao formato das imagens do *ImageNet*, os dados de entrada tem tamanho 227×227 e três canais de cor, correspondentes ao RGB. Já a camada de

[1] Disponível em `https://www.image-net.org/`.

saída possui mil unidades, correspondentes às mil categorias do banco de dados. Devido ao alto número de parâmetros do modelo, as camadas completamente conectadas possuem uma função de *dropout* que, conforme descrito na Seção 2.2.4.1, desliga de maneira aleatória um conjunto de neurônios a cada etapa do treinamento.

Um exemplo de implementação da arquitetura *AlexNet* pode ser apreciada no Apêndice A.2.

Figura 13 – Arquitetura da rede *AlexNet*.



Fonte: Retirado de Putzu, Piras e Giacinto (2020).

A primeira camada recebe uma imagem $227 \times 227 \times 3$, onde a terceira dimensão corresponde aos canais de cores RGB. Essa imagem passa para uma camada de convolução com 96 filtros de tamanho 11×11, resultando em uma imagem $55 \times 55 \times 96$. Em seguida, uma camada de *max pooling* com filtro 3×3 retorna uma imagem de tamanho $27 \times 27 \times 96$. A terceira camada é de convolução, contendo 256 filtros de tamanho 5×5, resultando em uma imagem $27 \times 27 \times 256$, que passa novamente por uma camada de *max pooling* 3×3, gerando uma imagem de $13 \times 13 \times 256$.

Na quinta e sexta camadas, a imagem passa por convolução com 384 filtros 3×3, tendo assim dimensões $13 \times 13 \times 384$. Na sétima camada convolucional, são 256 filtros 3×3, que geram uma imagem $13 \times 13 \times 256$. A oitava camada é de subamostragem, com filtros 3×3, resultando em uma imagem $6 \times 6 \times 256$, para então passar por duas camadas completamente conectadas de 4096 filtros e uma camada completamente conectada final, onde a imagem é classificada em uma das 1000 categorias.

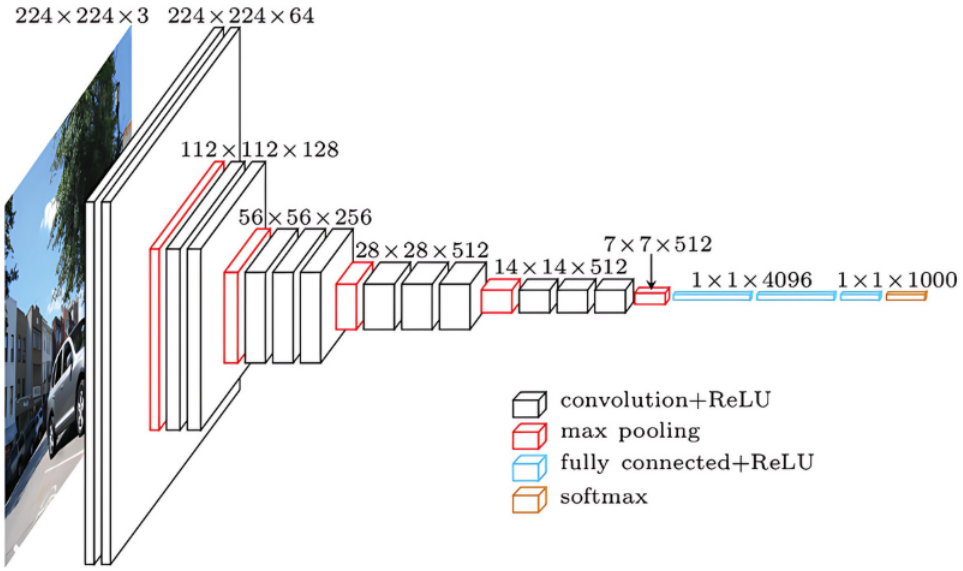
3.3 *VGGNet*

Em 2014, na Universidade de Oxford, foi proposta a ideia de utilizar blocos de camadas em uma rede. Tendo surgido em um Grupo de Geometria Visual (em Inglês, *Visual Geometry Group*, ou VGG), o modelo *VGGNet* foi batizado com o mesmo nome (SIMONYAN; ZISSERMAN, 2014).

Cada bloco é composto por duas ou três camadas de convolução e uma camada de pooling, sendo a rede formada por cinco blocos e duas camadas completamente conectadas. Algumas variações podem ocorrer no número de camadas de convolução no modelo, já que os autores fizeram testes com diferentes arquiteturas com o mesmo modelo utilizando blocos.

Um exemplo de implementação da arquitetura *VGGNet* pode ser apreciada no Apêndice A.3.

Figura 14 – Arquitetura da rede *VGGNet*, ilustrando suas camadas convolucionais, camadas de subamostragem e camadas completamente conectadas.



Fonte: Retirado de Bezdan e Bacanin (2019).

Uma característica desse modelo é possuir todos os filtros das camadas convolucionais no mesmo tamanho 3×3, o menor possível. A camada de entrada recebe imagens no tamanho $224 \times 224 \times 3$, ou seja, imagens com escala de cor RGB. A cada bloco da rede as dimensões espaciais da imagem têm seu tamanho reduzido pela metade, enquanto o número de parâmetros é dobrado. Isso leva a *VGGNet* a ser uma rede com excelentes resultados de classificação, mas sua profundidade requer um alto custo computacional.

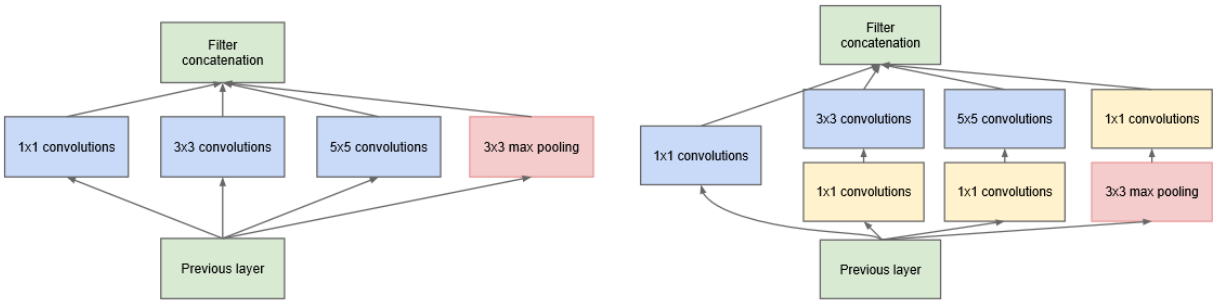
3.4 *GoogLeNet*

Vencedora do prêmio ILSVRC de 2014, o *GoogLeNet* utilizou *blocos Inception*, blocos de camadas similares aos blocos da rede *VGGNet*. A maior diferença se dá pelo processamento ocorrido nesses blocos, que são compostos de camadas convolucionais com diferentes tamanhos de filtros. Ao final de cada bloco, os mapas de ativação são concatenados em uma única saída, que passa para a camada seguinte (SZEGEDY et al., 2014). A ideia para os blocos *inception* está em introduzir dispersão, para possibilitar uma rede mais profunda e mais ampla, diminuindo os impactos no *overfitting* e no custo computacional.

Szegedy et al. (2014) apresentam duas versões para o bloco *inception*, a versão *naïve* (*ingênua*, em tradução livre), e a versão com redução de dimensões. A primeira utiliza quatro camadas: convolução 1×1, convolução 3×3, convolução 5×5 e *max pooling* 3×3. No entanto, o custo computacional das camadas maiores torna-se um problema devido ao número de operações realizadas, criando a necessidade da segunda versão do bloco, com camadas convolucionais 1×1 conforme ilustra a Figura 15.

Posteriormente foram lançadas outras versões, com pequenos ajustes e melhorias. As diferen-

Figura 15 – Composição do bloco *inception* nas duas versões, *naïve* e com redução de dimensões.



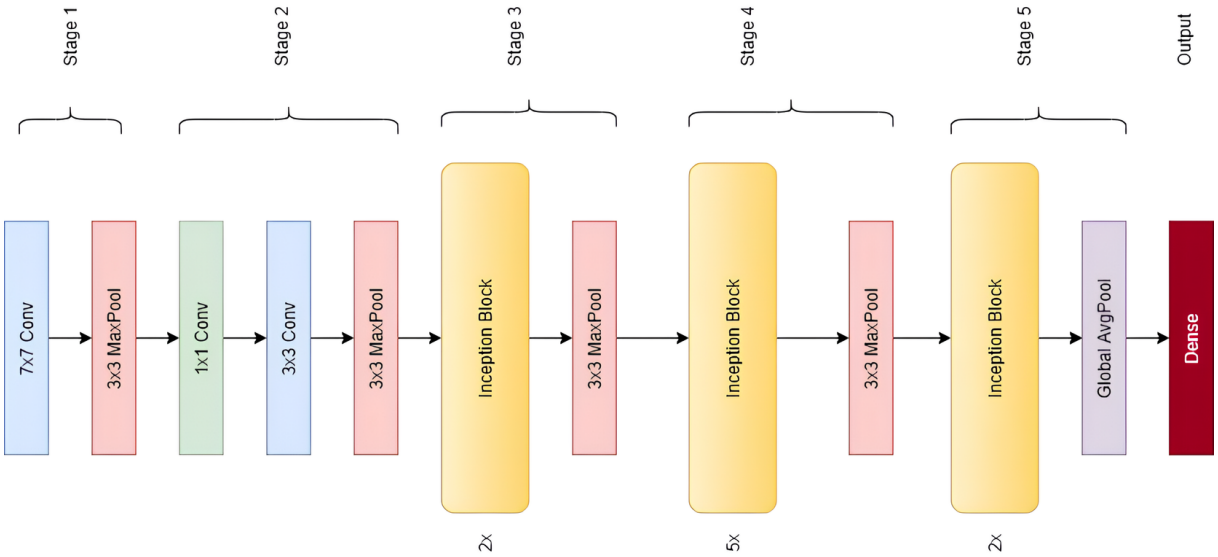
(a) Inception module, naïve version (b) Inception module with dimension reductions

Fonte: Retirado de Szegedy et al. (2014).

ças entre as versões incluem blocos *inception* modificados, mudança na camada de *max pooling* para *mean pooling* e ainda o uso de redes auxiliares, que geram classificações auxiliares antes do fim do processamento da rede. No entanto, Szegedy et al. (2014) nomeou apenas a rede enviada para a competição ILSVRC como *GoogLeNet*, sendo as outras versões chamadas de *Inception Net*. No total, foram quatro versões diferentes para o modelo.

A arquitetura completa da *GoogLeNet* possui nove módulos *inception* separados por camadas de *max pooling* 3×3. No Apêndice A.4 é apresentada uma versão simplificada de sua implementação.

Figura 16 – Arquitetura da rede *GoogLeNet*, com 9 blocos *inception* separados por camadas de *max pooling*.



Fonte: Adaptado de Sarkar (2020).

4 Metodologia

Neste capítulo, serão descritos os procedimentos executados para a avaliação das redes neurais no Capítulo 5.

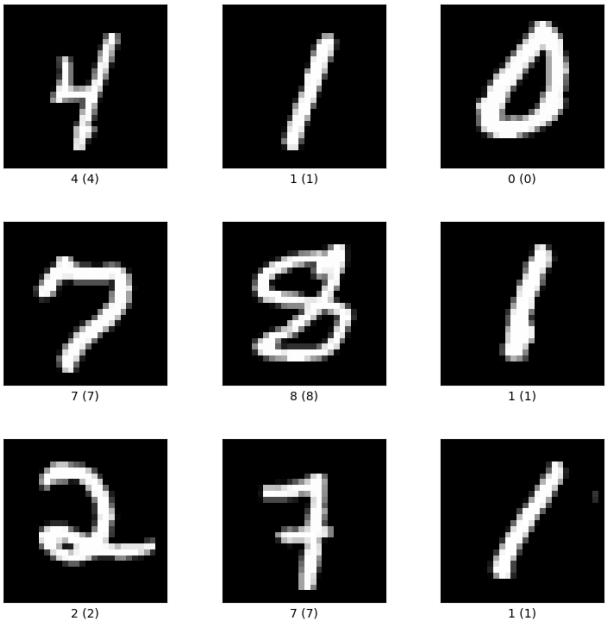
4.1 Coleta de dados

Para o treinamento das redes neurais serão utilizados dois bancos de dados, o *MNIST* e o *Cifar-10*. Em ambos os casos, 20% dos dados de treinamento foram separados para validação do modelo.

4.1.1 O banco de imagens MNIST

MNIST database, sigla para *Modified National Institute of Standards and Technology database* (do Inglês, Banco de Dados Modificado do Instituto Nacional de Padrões e Tecnologia) é um banco de dados criado em 1994 a partir da modificação do banco de dados NIST (BOTTOU et al., 1994). São 60 000 imagens de treinamento e 10 000 de teste de dígitos manuscritos, de tamanho 28×28 em escala de cinza.

Figura 17 – Exemplos de imagens do banco de dados *MNIST*.

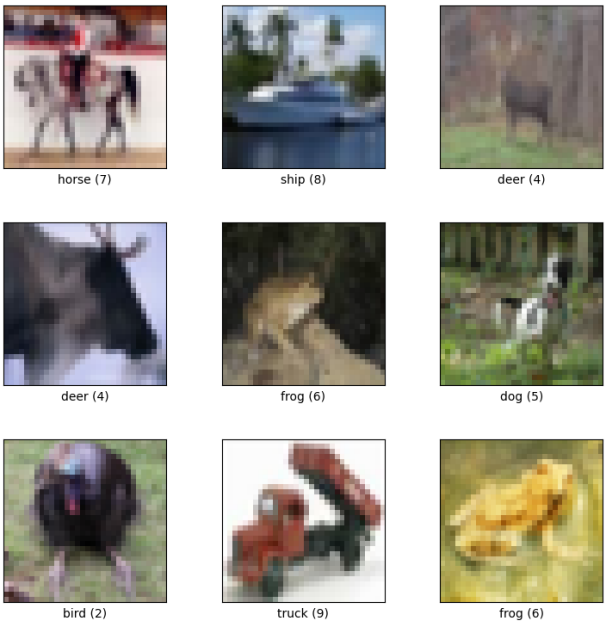


Fonte: Feito pela autora, a partir de imagens do banco de dados MNIST.

4.1.2 O banco de imagens Cifar-10

Publicado em 2009, o banco de dados Cifar-10 é nomeado em agradecimento ao *Canadian Institute for Advanced Research* (do Inglês, Instituto Canadense de Pesquisas Avançadas), financiador do projeto. O banco de dados consiste em 60 000 imagens coloridas de tamanho 32×32 distribuídas em 10 categorias: aviões, carros, pássaros, gatos, veados, cães, sapos, cavalos, navios e caminhões.

Figura 18 – Exemplos de imagens do banco de dados Cifar-10. Por conta de suas dimensões reduzidas (32×32) a resolução é muito baixa.



Fonte: Feito pela autora, a partir de imagens do banco de dados Cifar-10.

4.2 Tratamento e adequação das imagens

Antes de serem usadas no treinamento das redes, as imagens de ambos os bancos de dados passaram por algumas adequações:

Normalização A normalização garante que todas as imagens estejam na mesma escala, assim os valores de cada pixel foram divididos por 255. Isso transforma a escala de cinza original $[0, 255]$ em um intervalo mais uniforme, entre $[0, 1]$.

Alterações nas dimensões Nos bancos de dados utilizados, as imagens são dadas no tamanho 28×28 para o MNIST e 32×32 para o Cifar-10. No entanto, as redes que serão treinadas utilizam diferentes tamanhos de entrada, então alguns ajustes foram necessários para o sucesso do treinamento.

Para a rede *LeNet-5*, a camada original recebe imagens no tamanho 32×32. Para adaptá-la ao banco MNIST, foi feita uma alteração no atributo *input shape*, que determina o tamanho dos dados que a primeira camada deve esperar receber.

Já para as redes *AlexNet* e *VGGNet*, os dados de entrada são de formato 224×224, muito diferente do tamanho original das imagens do banco Cifar-10. Nesse caso, optou-se pelo redimensionamento das imagens para o tamanho necessário.

4.3 Treinamento das redes

Todas as redes foram treinadas utilizando-se o *Google Colab Pro*[1], com unidades de processamento GPU V100[2], disponibilizada pela plataforma. Para a implementação das redes, foram utilizados os pacotes *Tensorflow* (TENSORFLOW..., 2015), *Keras* (CHOLLET et al., 2015) e *Numpy* (HARRIS et al., 2020) e a linguagem de programação *Python*[3], versão 3.

4.3.1 Parâmetros de treinamento

De todos os parâmetros que podem ser usados na configuração do treinamento da rede, dois são mandatórios: número de *epochs* (épocas, em Português) e *batch size* (tamanho do lote, em Português).

- Uma *epoch* trata-se da passagem dos dados pelo modelo completo. Assim, a cada *epoch* o modelo reajusta seus valores durante o treinamento. Quantidades pequenas de *epochs* podem não extrair o máximo de aprendizado da rede, enquanto quantidades muito grandes podem favorecer o sobreajuste.

- Já o *batch size* divide os dados em lotes que serão utilizados para treinar a rede. Por exemplo, dado um conjunto com 1000 imagens e um *batch size* igual a 100, a cada *epoch* o modelo irá selecionar as primeiras 100 imagens e fazer o treinamento; em seguida, pega as 100 imagens seguintes e alimenta a rede, e assim até que todos os dados tenham sido utilizados. Um número pequeno para o *batch size* permite que a rede utilize menos memória, o que diminui o tempo necessário para treinamento. No entanto, lotes muito pequenos podem afetar negativamente o desempenho da rede.

4.3.2 Métricas de desempenho

Existem diversas métricas que podem ser utilizadas para avaliar a qualidade do treinamento de um modelo de classificação. A seguir, serão apresentados as quatro que serão utilizadas neste trabalho.

[1] `https://colab.research.google.com/`

[2] A V100 Tensor Core é uma GPU da NVIDIA com arquitetura especificamente projetada para Aprendizado de Máquina. Mais informações em `https://www.nvidia.com/en-us/data-center/v100/`.

[3] `https://www.python.org`

Taxa de perda Há várias maneiras de se calcular a taxa de perda, também chamada de taxa de erro ou *loss*, de um modelo. A escolhida para este trabalho foi a *Categorical Crossentropy*, muito utilizada em modelos de classificação com mais de duas classes. Nesse método, a taxa é calculada por

$$loss = -\sum y_i \times \log p_i,$$

onde y_i representa o valor real da i-ésima classe e p_i representa a probabilidade prevista pelo modelo para a i-ésima classe. Quanto menor a taxa de perda, melhor é o desempenho do modelo.

Taxa de exatidão Também chamada de acurácia, a taxa de exatidão é dada pela razão entre os acertos do modelo e o total de dados. Quanto mais próximo de 1 for o resultado, melhor é o desempenho do modelo.

$$\text{exatidão} = \frac{\text{acertos}}{\text{total de dados}}.$$

Taxa de precisão A taxa de precisão é calculada individualmente para cada classe do modelo, e no final, é obtida pela média entre os valores de cada classe. O cálculo é feito com a divisão entre os verdadeiros positivos (VP) de uma classe divididos pela soma entre verdadeiros positivos e falso positivos (FP), onde um resultado é considerado verdadeiro positivo quando o modelo classifica como pertencente à classe correta, e falso positivo quando o modelo classifica de acordo com aquela classe, mas o exemplo pertence à outra classe. Assim como a exatidão, quanto mais próximo de 1 for o resultado, melhor é o desempenho do modelo.

$$\text{precisão} = \frac{VP}{VP + FP}.$$

Taxa de sensibilidade Sensibilidade, ou *recall*, assim como a precisão, é calculada para cada classe e em seguida apresenta-se a média entre os resultados. Essa taxa é dada pela divisão entre os resultados verdadeiros positivos pela soma entre os verdadeiros positivos e falsos negativos (FN), onde falsos negativos são resultados que pertencem àquela classe, mas foram identificadas erroneamente pelo modelo. Novamente, quanto mais próximo de 1 for o resultado, melhor é o desempenho do modelo.

$$\text{recall} = \frac{VP}{VP + FN}.$$

4.4 Versões dos softwares utilizados

Na Tabela 2 podem ser conferidas as versões utilizadas de cada pacote usado nas implementações. Essas versões eram as disponibilizadas pelo *Google Colab Pro* à época da execução dos testes.

Tabela 2 – Versões dos software utilizados.

Software	Versão
`Tensorflow`	2.14.0
`Keras`	2.14.0
`Numpy`	1.23.5

Fonte: Feito pela autora.

5 Comparação de arquiteturas de RN

Nesse capítulo, as redes neurais apresentadas no Capítulo 3 serão implementadas e avaliadas de acordo com seu desempenho com o banco de dados MNIST para a *LeNet-t*, e o banco Cifar-10 para as redes *AlexNet* e *VGGNet*, conforme explicado na Metodologia, Capítulo 4. A rede *GoogLeNet* não será avaliada devido a sua alta complexidade e demanda computacional.

5.1 Desempenho da rede *LeNet-5*

O teste da rede *LeNet-5* será feito com o banco de dados *MNIST*, descrito na Seção 4.1.1, por se tratar de uma rede com menor complexidade feita para imagens pequenas e em escala de cinza.

Originalmente, a camada de entrada da rede *LeNet-5* recebe imagens no tamanho 32×32, então uma adaptação foi feita na camada de entrada como descrito na Seção 4.2. Deste modo, a arquitetura do modelo ficou como descrito na Tabela 3.

Tabela 3 – Arquitetura interna da rede *LeNet-5* no Tensorflow. Os tipos de camada convolucional e *average pooling* correspondem às abordadas na Seção 2.2. A camada densa é uma camada padrão, como explicado na Seção 2.1 e a camada *flatten* transforma os dados de um tensor 3D para um vetor simples.

CAMADA	TIPO	DIMENSÕES	# DE PARÂMETROS
C1	convolucional	$28 \times 28 \times 6$	156
S2	*average pooling*	$14 \times 14 \times 6$	0
C3	convolucional	$14 \times 14 \times 6$	2416
S4	*average pooling*	$7 \times 7 \times 6$	0
FLAT	*flatten*	784	0
C5	densa	120	94 200
F6	densa	84	10 164
OUTPUT	densa	10	850

Fonte: Feito pela autora.

Tabela 4 – Avaliação das métricas do modelo *LeNet-5*.

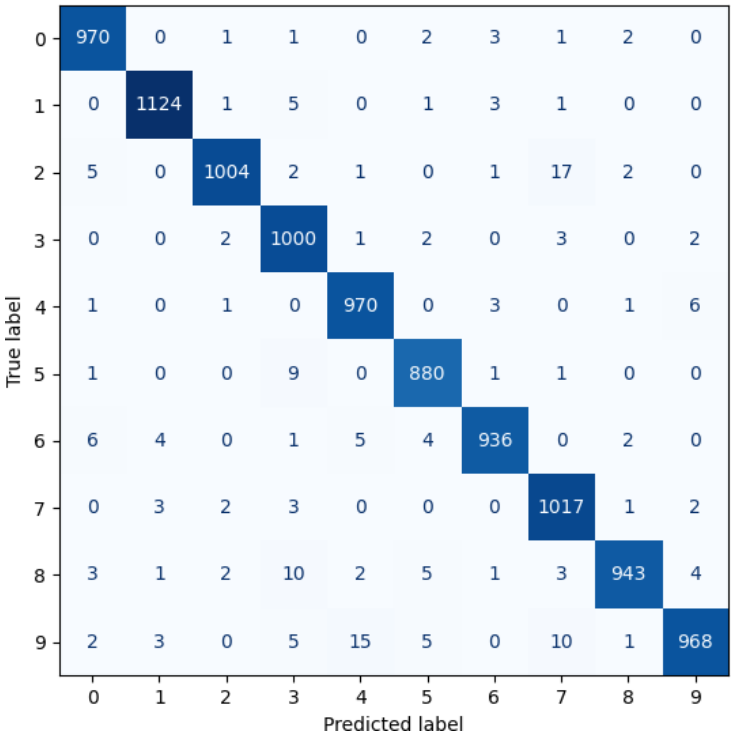
Épocas	Perda	Exatidão	Precisão	Sensibilidade
10	0,0637	0,9817	0,9829	0,9806
20	0,0726	0,9830	0,9834	0,9827
30	0,0768	0,9840	0,9848	0,9838
40	0,0821	0,9849	0,9850	0,9848
50	0,0717	0,9875	0,9878	0,9871
100	0,1328	0,9861	0,9862	0,9861

Fonte: Feito pela autora.

O resultado do treinamento com diferentes quantidades de épocas é apresentado na Tabela 4. Observa-se que conforme o número de épocas aumenta, aumentam também os valores de todas as métricas até que comecem a diminuir a partir de 50 épocas, indicando este valor como uma boa escolha para o treinamento. Enquanto aumentar exatidão, precisão e sensibilidade seja um bom indicativo, a taxa de perda também sofre aumento, o que indica o oposto. No entanto, a perda continua com um valor baixo e as demais com valores bem próximos de 1, indicando um excelente desempenho da rede *LeNet-5*.

Na Figura 19 é apresentada a matriz de confusão (STEHMAN, 1997) para os resultados da previsão com a *LeNet-5*. Cada linha da matriz representa o dígito verdadeiro, como rotulado previamente; já cada coluna representa o dígito que a rede classificou. Nas intersecções entre linhas e colunas estão as quantidades de previsões. Como pode-se verificar, a diagonal da matriz de confusão (verdadeiros positivos) tem valores no mínimo duas ordens de grandeza maiores que no resto da matriz, indicando que a precisão da rede é alta. Fora da diagonal estão os falsos positivos: o número 8, por exemplo, foi "visto" pela rede como um 3 em 10 casos do conjuntos de testes.

Figura 19 – Matriz de confusão representando o desempenho da *LeNet-5* no conjunto de dados MNIST.



O bom desempenho desta arquitetura está associado à sua simplicidade e à adequação com a tarefa proposta, afinal a mesma foi desenvolvida justamente para o banco de dados utilizado neste trabalho. Porém, sua simplicidade representa também uma limitação, já que o desempenho seria inferior para tarefas mais difíceis, como por exemplo com imagens mais detalhadas e complexas.

5.2 Desempenho da rede *AlexNet*

O teste da rede *AlexNet* será feito com o banco de dados Cifar-10, descrito na Seção 4.1.2. Conforme descrito na Seção 4.2, para a compilação do modelo foi necessário redimensionar as imagens para o tamanho 224×224. A arquitetura do modelo implementado tem a estrutura descrita na Tabela 5

Tabela 5 – Arquitetura interna da rede *AlexNet* no Tensorflow. Os tipos de camada convolucional e *max pooling* correspondem às abordadas na Seção 2.2. A camada densa é uma camada padrão, como explicado na Seção 2.1 e a camada *flatten* transforma os dados de um tensor 3D para um vetor simples.

CAMADA	TIPO	DIMENSÕES	# DE PARÂMETROS
C1	convolucional	$56 \times 56 \times 96$	34 944
S2	*max pooling*	$24 \times 27 \times 96$	0
C3	convolucional	$27 \times 27 \times 256$	614 656
S4	*max pooling*	$13 \times 13 \times 256$	0
C5	convolucional	$13 \times 13 \times 384$	885 120
C6	convolucional	$13 \times 13 \times 384$	1 327 488
C7	convolucional	$13 \times 13 \times 256$	884 992
S8	*max pooling*	$6 \times 6 \times 256$	0
FLAT	*flatten*	9216	0
F9	densa	4096	37 752 832
DROPOUT	*dropout*	4096	0
F10	densa	4096	16 781 312
DROPOUT	*dropout*	4096	0
OUTPUT	densa	10	40 970

Pelo número de parâmetros do modelo, pode-se perceber o aumento considerável na complexidade em relação à rede *LeNet-5*. No modelo *LeNet-5*, a camada com maior número de parâmetros, 94 200, ocupa aproximadamente 368 KB de memória. Já para o modelo *AlexNet*, a camada com maior número de parâmetros ocupa um valor próximo de 144 MB de memória, cerca de 400 vezes mais.

Além dos parâmetros, o banco de dados com imagens 224×224 ocupa um espaço muito maior na memória computacional, e portanto, não foi possível realizar o treinamento com 100% dos dados, sendo necessário reduzi-lo para 25%. O resultado do treinamento com diferentes quantidades de épocas é apresentado na Tabela 6.

É possível observar o desempenho comparável ao obtido pela rede *LeNet-5*, porém com menos épocas (embora cada época gaste um tempo computacional muito maior). Esse desempenho poderia ser ainda melhor, não fossem:

- Redução no banco de dados: A redução do tamanho do conjunto de treinamento: utilizar um banco de dados de 60 000 imagens em tamanho 224×224 exigiria um custo computacional

Tabela 6 – Avaliação das métricas do modelo *AlexNet*.

Épocas	Perda	Exatidão	Precisão	Sensibilidade
10	1,1767	0,5737	0,7232	0,4139
20	0,4255	0,8491	0,8887	0,8173
30	0,2181	0,9300	0,9405	0,9195

Fonte: Feito pela autora.

muito alto, não sendo possível a realização em computadores de uso doméstico nem mesmo com o uso do *Google Colab*[1].

- Uso do banco de dados Cifar-10: A rede *AlexNet* foi criada e otimizada para o banco de dados *ImageNet* (LI et al., 2009), como parte da competição *ImageNet Large Scale Visual Recognition Challenge* de 2012. Nesse banco de dados atualmente constam 14,2 milhão de imagens com mais de 21 000 categorias. Desse modo, a alteração das imagens de treino leva a uma alteração em seu desempenho.

Na Figura 20 é apresentada a matriz de confusão para os resultados da previsão com a *AlexNet*. Cada linha da matriz representa a classe original atribuída às imagens, enquanto cada coluna representa a classe a rede obteve. Nas intersecções entre linhas e colunas estão as quantidades de previsões. Como pode-se verificar, a diagonal da matriz de confusão (verdadeiros positivos) tem valores no mínimo uma ordem de grandeza maior que no resto da matriz, indicando que a precisão da rede é razoável, mesmo com o conjunto reduzido de dados de treinamento. Fora da diagonal estão os falsos positivos: *caminhão*, por exemplo, foi classificado pela rede como um *automóvel* em 79 casos do conjuntos de testes. Porém, é interessante observar que o contrário, *automóveis* classificados como *caminhões* possui apenas 21 falsos positivos.

5.3 Desempenho da rede *VGGNet*

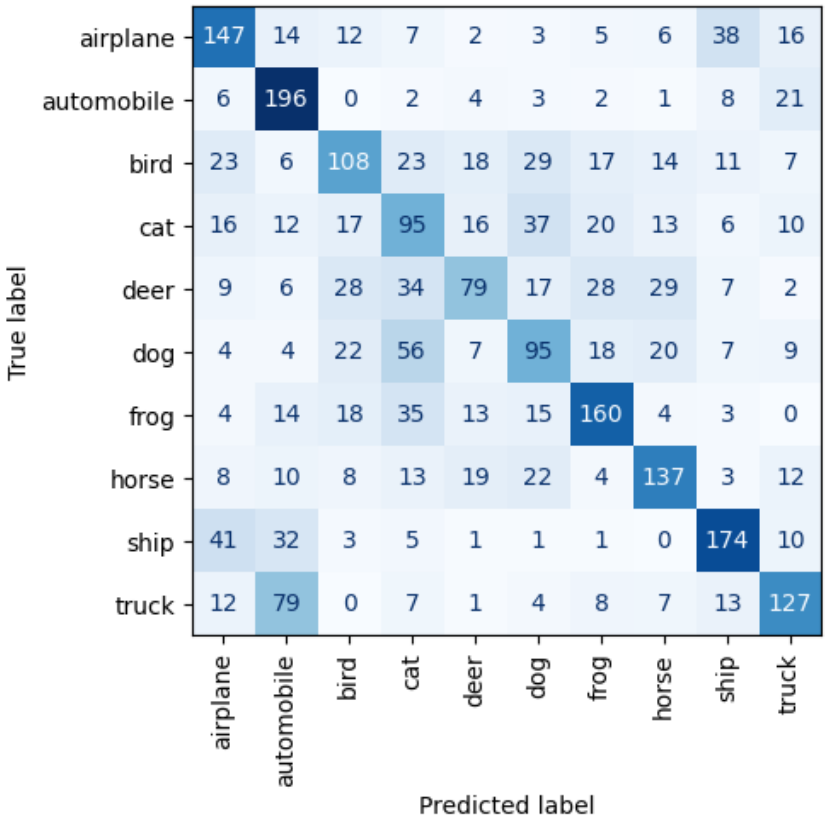
Assim como o modelo *AlexNet*, o modelo *VGGNet* também foi treinado com o banco de dados Cifar-10, com imagens adaptadas para 224×224. Por ser ainda mais complexa, foi necessário reduzir o banco de dados para 0,5% dos dados disponíveis, o que corresponde à 300 imagens. A arquitetura possui o formato descrito na Tabela 7.

O custo computacional aumenta consideravelmente em comparação com a rede *Alexnet*. Enquanto na última a camada com maior número de parâmetros utilizava cerca de 144 MB de memória, na *VGGNet* a camada com mais parâmetros ocupa cerca de 392 MB, sendo mais do que o dobro. O maior número de camadas também contribui para o custo computacional, tornando a *VGGNet* uma rede cara em termos de recursos.

O resultado do treinamento com diferentes quantidades de épocas é apresentado na Tabela 8.

[1] Em sua versão gratuita. Naturalmente mais poder computacional pode ser adquirido, em dólares.

Figura 20 – Matriz de confusão representando o desempenho da *AlexNet* no conjunto de dados Cifar-10.



Não foi possível o treinamento com mais épocas, devido ao alto consumo de memória necessário.

Os mesmos fatores que influenciaram negativamente na Seção 5.2 podem ser aplicados aqui. Os resultados obtidos mostram que a redução da quantidade de dados é um fator muito prejudicial para o treinamento.

O uso de uma rede com a complexidade da *VGGNet* deve ser avaliado de acordo com a tarefa e recursos disponíveis. No caso deste trabalho, não havia memória suficiente para que a rede pudesse alcançar todo seu potencial, o que resultou em métricas extremamente pobres.

Tabela 7 – Arquitetura interna da rede *VGGNet* no Tensorflow. Os tipos de camada convolucional e *max pooling* correspondem às abordadas na Seção 2.2. A camada densa é uma camada padrão, como explicado na Seção 2.1 e a camada *flatten* transforma os dados de um tensor 3D para um vetor simples.

CAMADA	TIPO	DIMENSÕES	# DE PARÂMETROS
C1	convolucional	$224 \times 224 \times 64$	1792
C2	convolucional	$224 \times 224 \times 64$	36 928
S3	*max pooling*	$112 \times 112 \times 96$	0
C4	convolucional	$112 \times 112 \times 128$	73 856
C5	convolucional	$112 \times 112 \times 128$	147 584
S6	*max pooling*	$56 \times 56 \times 128$	0
C7	convolucional	$56 \times 56 \times 256$	295 168
C8	convolucional	$56 \times 56 \times 256$	590 080
C9	convolucional	$56 \times 56 \times 256$	590 080
S10	*max pooling*	$28 \times 28 \times 256$	0
C11	convolucional	$28 \times 28 \times 512$	1 180 160
C12	convolucional	$28 \times 28 \times 512$	2 359 808
C13	convolucional	$28 \times 28 \times 512$	2 359 808
S14	*max pooling*	$14 \times 14 \times 512$	0
C15	convolucional	$14 \times 14 \times 512$	2 359 808
C16	convolucional	$14 \times 14 \times 512$	2 359 808
C17	convolucional	$14 \times 14 \times 512$	2 359 808
S18	*max pooling*	$7 \times 7 \times 512$	0
FLAT	*flatten*	25088	0
F19	densa	4096	102 764 544
F20	densa	4096	16 781 312
OUTPUT	densa	10	40 970

Tabela 8 – Avaliação das métricas do modelo *VGGNet*.

Épocas	Perda	Exatidão	Precisão	Sensibilidade
10	2,2904	0,0600	0,0000	0,0000
50	2,2934	0,1800	0,0000	0,0000
100	2,2904	0,1800	0,0000	0,0000
150	2,3065	0,0600	0,0000	0,0000

Fonte: Feito pela autora.

6 Considerações Finais

As Redes Neurais são ferramentas computacionais de imensa capacidade, com aplicações em diversas áreas. Para tarefas de visão computacional, as Redes Neurais Convolucionais representam o estado da arte. No entanto, a escolha do modelo mais adequado deve considerar fatores como a complexidade da tarefa e do desempenho necessário, além do custo computacional relacionado. Neste trabalho foram implementadas e avaliadas as redes *LeNet-5*, *AlexNet* e *VGGNet*, com os bancos de dados MNIST e Cifar-10.

A rede *LeNet-5* apresentou excelente desempenho para a classificação de dígitos manuscritos, com cerca de 98% de exatidão e de precisão, valor ainda mais alto do que o apresentado por LeCun et al. (1989), de 95%. A simplicidade da rede e a simplicidade da tarefa corroboraram para o resultado obtido.

O modelo *AlexNet* apresentou desempenho mediano. As causas prováveis são a mudança de banco de dados em relação ao utilizado originalmente no treinamento e a diminuição da quantidade de dados disponíveis, tendo sido treinada com 30 000 imagens, 10% do total do banco de dados. Outras técnicas de normalização poderiam ter sido exploradas para potencializar a rede, ou ainda, a técnica de *transfer learning*, que consiste na utilização de uma rede pré-treinada. Como o intuito do trabalho envolvia a implementação das redes, optou-se por não utilizar-se dessa técnica, mas pode ser uma alternativa para obter melhores resultados.

De maneira semelhante, a rede *VGGNet* também teve seu desempenho afetado pela falta de recursos computacionais suficientes para seu treinamento, apresentando resultados muito aquém de sua capacidade, já que se tornou uma rede excessivamente complexa para o treinamento com apenas 300 imagens (uma amostra realmente muito pequena). O uso de *transfer learning* ou de outras ferramentas de regularização poderiam melhorar o resultado, mas o principal ajuste necessário é a disponibilidade de recursos computacionais.

Em conclusão, é possível perceber a utilidade e o potencial do uso de redes neurais, sendo necessário uma avaliação do problema a ser aplicado para a escolha ou implementação do modelo mais adequado. Futuros trabalhos de estudo da área podem abordar outras aplicações de Redes Convolucionais na área de visão computacional, como melhoria na qualidade ou geração de imagens, ou ainda outras áreas de aplicação, como *chatbots* e reconhecimento de voz.

Referências

AGGARWAL, Charu C. **Neural Networks and Deep Learning**. Springer International Publishing, 2018. DOI: 10.1007/978-3-319-94463-0. Disponível em: <https://doi.org/10.1007/978-3-319-94463-0>.

COMPUTER VISION CONFERENCE (CVC). **Advances in Computer Vision**. Edição: Kohei Arai e Supriya Kapoor: Springer International Publishing, 2020. v. 1. DOI: 10.1007/978-3-030-17795-9.

BEZDAN, Timea; BACANIN, Nebojsa. Convolutional Neural Network Layers and Architectures. In: p. 445–451. DOI: 10.15308/Sinteza-2019-445-451.

BLEDSOE, Woodrow Wilson. **A Facial Recognition Project Report**. 1964.

BOTTOU, L. et al. Comparison of classifier methods: a case study in handwritten digit recognition. In: PROCEEDINGS of the 12th IAPR International Conference on Pattern Recognition, Vol. 3 - Conference C: Signal Processing (Cat. No.94CH3440-5). 1994. v. 2, 77–82 vol.2. DOI: 10.1109/ICPR.1994.576879.

CARTWRIGHT, Mark. **Hefesto**. Edição: Joana Ribeiro. Mar. 2023. Disponível em: <https://www.worldhistory.org/trans/pt/1-10802/hefesto/>.

CHOLLET, F. et al. **Keras**. 2015. Disponível em: <https://keras.io>.

CHOLLET, François. **Deep learning with Python**. 1. ed. New York: Manning Publications, 2017.

DATA SCIENCE ACADEMY. **Deep Learning Book**. 2022. https://www.deeplearningbook.com.br/funcao-de-ativacao/.

DENG, Jia et al. ImageNet: A large-scale hierarchical image database. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition. 2009. P. 248–255. DOI: 10.1109/CVPR.2009.5206848.

DESHPANDE, Adit. **A beginner's guide to understanding Convolutional Neural Networks**. Jul. 2016. Disponível em: <https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/>.

ELGENDY, Mohamed. **Deep Learning for Vision Systems**. Shelter Island, NY: Manning Publications, 2020. ISBN 9781617296192.

FUKUSHIMA, Kunihiko; MIYAKE, Sei. Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Visual Pattern Recognition. In: AMARI, Shun-ichi; ARBIB, Michael A. (Ed.). **Competition and Cooperation in Neural Nets**. Berlin, Heidelberg: Springer Berlin Heidelberg, 1982. P. 267–285. ISBN 978-3-642-46466-9.

HARRIS, C. R. et al. Array programming with NumPy. **Nature**, v. 585, n. 7825, p. 357–362, set. 2020. Springer Science and Business Media LLC. DOI: 10.1038/s41586-020-2649-2. Disponível em: <https://doi.org/10.1038/s41586-020-2649-2>.

HAYKIN, S. **Redes Neurais: Princípios e Prática**. Bookman Editora, 2001. ISBN 9788577800865. Disponível em: <https://books.google.com.br/books?id=bhMwDwAAQBAJ>.

HEBB, Donald Olding. **The Organization of Behavior**: a Neuropsychological Theory. 1st edition: John Wiley & Sons, 1949. ISBN 9780805843002.

HOPFIELD, John J. Neural networks and physical systems with emergent collective computational abilities. **Biophysics**, v. 79, p. 2554–2558, 1982. Disponível em: <https://authors.library.caltech.edu/7427/1/HOPpnas82.pdf>.

HUBEL, D. H.; WIESEL, T. N. Receptive fields and functional architecture of monkey striate cortex. **The Journal of Physiology**, v. 195, n. 1, p. 215–243, 1968. DOI: 10.1113/jphysiol.1968.sp008455.

KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. ImageNet Classification with Deep Convolutional Neural Networks. In: PROCEEDINGS of the 25th International Conference on Neural Information Processing Systems - Volume 1. Lake Tahoe, Nevada: Curran Associates Inc., 2012. (NIPS'12), p. 1097–1105.

LAKSHMANAN, Valliappa; GÖRNER, Martin; GILLARD, Ryan. **Practical Machine Learning for Computer Vision**: End-to-End Machine Learning for Images. O'Reilly, 2021. ISBN 978-1-098-10236-4.

LECUN, Y. et al. Backpropagation Applied to Handwritten Zip Code Recognition. **Neural Computation**, v. 1, n. 4, p. 541–551, 1989. DOI: 10.1162/neco.1989.1.4.541.

LECUN, Y. et al. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, v. 86, n. 11, p. 2278–2324, 1998. DOI: 10.1109/5.726791.

LI, Fei-Fei et al. ImageNet: A large-scale hierarchical image database. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition. 2009. P. 248–255. DOI: 10.1109/CVPR.2009.5206848.

MCCARTHY, J. et al. **A proposal for the Dartmouth Summer Research Project on artificial intelligence**. 1955. 23 p.

MCCLANAHAN, Chris. **History and Evolution of GPU Architecture A Paper Survey**. 2011. Disponível em: <https://mcclanahoochie.com/blog/wp-content/uploads/2011/03/gpu-hist-paper.pdf>.

MCCULLOCH, Warren S.; PITTS, Walter. A logical calculus of the ideas immanent in nervous activity. **The Bulletin of Mathematical Biophysics**, v. 5, n. 4, p. 115–133, 1943. DOI: 10.1007/BF02478259.

PARKER, D. B. **Learning-Logic: Casting the Cortex of the Human Brain in Silicon**. MA, 1985.

PUTZU, Lorenzo; PIRAS, Luca; GIACINTO, Giorgio. Convolutional neural networks for relevance feedback in content based image retrieval: A Content based image retrieval system that exploits convolutional neural networks both for feature extraction and for relevance feedback. **Multimedia Tools and Applications**, v. 79, out. 2020. DOI: 10.1007/s11042-020-09292-9.

RASAMOELINA, Andrinandrasana David; ADJAILIA, Fouzia; SINČÁK, Peter. A Review of Activation Function for Artificial Neural Network. In: 2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI). 2020. P. 281–286. DOI: 10.1109/SAMI48414.2020.9108717.

ROBERTS, Lawrence. **Machine Perception of Three-Dimensional Solids**. Jan. 1963. ISBN 0-8240-4427-4.

RUMELHART, David E.; HINTON, Geoffrey E.; WILLIAMS, Ronald J. Learning representations by back-propagating errors. **Nature**, v. 323, n. 6088, p. 533–536, 1986. DOI: 10.1038/323533a0.

RUSSAKOVSKY, Olga et al. ImageNet Large Scale Visual Recognition Challenge. **CoRR**, abs/1409.0575, 2014. arXiv: 1409.0575. Disponível em: <http://arxiv.org/abs/1409.0575>.

SAMUEL, A. Some Studies in Machine Learning Using the Game of Checkers. **IBM J. Res. Dev.**, v. 3, p. 210–229, 1959.

SARKAR, Arjun. **Understanding Inception: Simplifying the Network Architecture — medium.com**. 2020. https://medium.com/swlh/understanding-inception-simplifying-the-network-architecture-54cd31d38949.

SHORTLIFFE, Edward H.; BUCHANAN, Bruce G. A model of inexact reasoning in medicine. **Mathematical Biosciences**, v. 23, n. 3, p. 351–379, 1975. ISSN 0025-5564. DOI: https://doi.org/10.1016/0025-5564(75)90047-4. Disponível em: <https://www.sciencedirect.com/science/article/pii/0025556475900474>.

SIMONYAN, Karen; ZISSERMAN, Andrew. Very Deep Convolutional Networks for Large-Scale Image Recognition. **arXiv 1409.1556**, set. 2014.

SOLAIMAN, Irene et al. **Release Strategies and the Social Impacts of Language Models**. 2019. arXiv: 1908.09203 [cs.CL]. Disponível em: <https://arxiv.org/abs/1908.09203>.

STEHMAN, Stephen V. Selecting and interpreting measures of thematic classification accuracy. **Remote Sensing of Environment**, v. 62, n. 1, p. 77–89, 1997. ISSN 0034-4257. DOI: https://doi.org/10.1016/S0034-4257(97)00083-7.

SZEGEDY, Christian et al. Going Deeper with Convolutions. **CoRR**, abs/1409.4842, 2014. arXiv: 1409.4842. Disponível em: <http://arxiv.org/abs/1409.4842>.

TENSORFLOW. 2015. Disponível em: <https://tensorflow.org>.

TEUWEN, Jonas; MORIAKOV, Nikita. Chapter 20 - Convolutional neural networks. In: ZHOU, S. Kevin; RUECKERT, Daniel; FICHTINGER, Gabor (Ed.). **Handbook of Medical Image Computing and Computer Assisted Intervention**. Academic Press, 2020. (The Elsevier and MICCAI Society Book Series). P. 481–501. ISBN 978-0-12-816176-0. DOI: https://doi.org/10.1016/B978-0-12-816176-0.00025-9. Disponível em: <https://www.sciencedirect.com/science/article/pii/B9780128161760000259>.

VARGAS, Ana Caroline Gomes; PAES, Aline; VASCONCELOS, Cristina Nader. Um Estudo sobre Redes Neurais Convolucionais e sua Aplicação em Detecção de Pedestres. In: CAPPABIANCO, F. A. M. et al. (Ed.). **Electronic Proceedings of the 29th Conference on Graphics, Patterns and Images (SIBGRAPI'16)**. São José dos Campos, SP, Brazil, out. 2016. Disponível em: <http://gibis.unifesp.br/sibgrapi16>.

WEI, Guangfen et al. Development of a LeNet-5 Gas Identification CNN Structure for Electronic Noses. **Sensors**, v. 19, n. 1, 2019. ISSN 1424-8220. DOI: 10.3390/s19010217. Disponível em: <https://www.mdpi.com/1424-8220/19/1/217>.

WERBOS, Paul. Applications of advances in nonlinear sensitivity analysis. In: SYST. Model. Optim. Jan. 1970. v. 38, p. 762–770. ISBN 3-540-11691-5. DOI: 10.1007/BFb0006203.

WIDROW, B.; HOFF, M. E. Adaptive Switching Circuits. In: IRE WESCON Convention Record. 1960. P. 96–104. DOI: 10.21236/AD0241531.

ZEILER, Matthew D.; FERGUS, Rob. Visualizing and Understanding Convolutional Networks. In: FLEET, David et al. (Ed.). **Computer Vision – ECCV 2014**. Cham: Springer International Publishing, 2014. P. 818–833. ISBN 978-3-319-10590-1.

ZHANG, Aston et al. **Dive into Deep Learning**. 2020. https://d2l.ai.

APÊNDICE A – Exemplos de implementação

Todas as implementações utilizam as bibliotecas *TensorFlow* (TENSORFLOW..., 2015), *Keras* (CHOLLET et al., 2015) e *Numpy* (HARRIS et al., 2020) em Python.

A.1 *LeNet-5*

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, AveragePooling2D, Flatten, Dense

# Criação do modelo LeNet-5
model = Sequential()

# Primeira camada de convolução
model.add(Conv2D(filters=6, kernel_size=(5, 5), activation='sigmoid'))

# Primeira camada de pooling
model.add(AveragePooling2D(pool_size=(2, 2)))

# Segunda camada de convolução
model.add(Conv2D(filters=16, kernel_size=(5, 5), activation='sigmoid'))

# Segunda camada de pooling
model.add(AveragePooling2D(pool_size=(2, 2)))

# Terceira camada de convolução
model.add(Conv2D(filters=120, kernel_size=(5, 5), activation='sigmoid'))

# Aplanar a saída da camada de pooling
model.add(Flatten())

# Primeira camada totalmente conectada
model.add(Dense(units=120, activation='sigmoid'))

# Segunda camada totalmente conectada
model.add(Dense(units=84, activation='sigmoid'))
```

```
# Camada de saída
model.add(Dense(units=10, activation='softmax'))
```

A.2 *AlexNet*

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

# Criação do modelo AlexNet
model = Sequential()

# Primeira camada de convolução
model.add(Conv2D(filters=96, kernel_size=(11, 11), strides=(4, 4),
          activation='relu', input_shape=(227, 227, 3)))

# Primeira camada de pooling
model.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))

# Segunda camada de convolução
model.add(
    Conv2D(filters=256, kernel_size=(5, 5), padding='same', activation='relu')
)

# Segunda camada de pooling
model.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))

# Terceira camada de convolução
model.add(
    Conv2D(filters=384, kernel_size=(3, 3), padding='same', activation='relu')
)

# Quarta camada de convolução
model.add(
    Conv2D(filters=384, kernel_size=(3, 3), padding='same', activation='relu')
)

# Quinta camada de convolução
model.add(
    Conv2D(filters=256, kernel_size=(3, 3), padding='same', activation='relu')
)
```

```
# Terceira camada de pooling
model.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))

# Aplanar a saída da última camada de convolução
model.add(Flatten())

# Primeira camada totalmente conectada com Dropout
model.add(Dense(units=4096, activation='relu'))
model.add(Dropout(0.5))

# Segunda camada totalmente conectada com Dropout
model.add(Dense(units=4096, activation='relu'))
model.add(Dropout(0.5))

# Camada de saída
model.add(Dense(units=1000, activation='softmax'))
```

A.3 *VGGNet*

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Criação do modelo VGGNet
model = Sequential()

# Bloco 1
model.add(Conv2D(64, (3, 3), activation='relu', padding='same',
          input_shape=(224, 224, 3)))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2), strides=(2, 2)))

# Bloco 2
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2), strides=(2, 2)))

# Bloco 3
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2), strides=(2, 2)))
```

```
# Bloco 4
model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2), strides=(2, 2)))

# Bloco 5
model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2), strides=(2, 2)))

# Camadas totalmente conectadas
model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dense(4096, activation='relu'))
model.add(Dense(1000, activation='softmax'))
```

A.4 *GoogLeNet*

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, AveragePooling2D,
                                    concatenate, Flatten, Dense

def InceptionV1():
    # Entrada da rede
    inputs = Input(shape=(224, 224, 3))

    # Bloco 1
    conv1_1 = Conv2D(
        64, (7, 7), strides=(2, 2), activation='relu', padding='same'
    )(inputs)
    maxpool1 = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(conv1_1)

    # Bloco 2
    conv2_1 = Conv2D(64, (1, 1), activation='relu', padding='same')(maxpool1)
    conv2_2 = Conv2D(192, (3, 3), activation='relu', padding='same')(conv2_1)
    maxpool2 = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(conv2_2)
```

```
    # Bloco 3 - Módulo Inception 3a
    inception3a_1x1 = Conv2D(
        64, (1, 1), activation='relu', padding='same'
    )(maxpool2)
    inception3a_3x3 = Conv2D(
        128, (3, 3), activation='relu', padding='same'
    )(inception3a_1x1)
    inception3a_5x5 = Conv2D(
        32, (5, 5), activation='relu', padding='same'
    )(inception3a_1x1)
    inception3a_maxpool = MaxPooling2D((3, 3), strides=(1, 1), padding='same')(maxpool2)
    inception3a_pool_proj = Conv2D(
        32, (1, 1), activation='relu', padding='same'
    )(inception3a_maxpool)
    inception3a_output = concatenate(
        [inception3a_3x3, inception3a_5x5, inception3a_pool_proj], axis=-1
    )

    # Restante dos módulos Inception replicam o mesmo padrão do módulo 3a.

    # Camadas totalmente conectadas
    avgpool = AveragePooling2D(pool_size=(7, 7), strides=(1, 1))(inception5b_output)
    flat = Flatten()(avgpool)
    dense = Dense(1000, activation='softmax')(flat)

    # Criação do modelo
    model = Model(inputs=inputs, outputs=dense)
    return model

# Criar a instância do modelo
model = InceptionV1()
```