UNIVERSIDADE FEDERAL DE UBERLÂNDIA

João Lucas Almeida Moura

REDES NEURAIS: UMA APLICAÇÃO DE VISÃO COMPUTACIONAL

Uberlândia

2024

João Lucas Almeida Moura

REDES NEURAIS: UMA APLICAÇÃO DE VISÃO COMPUTACIONAL

Trabalho de Conclusão de Curso submetido ao curso de Engenharia Mecatrônica da Universidade Federal de Uberlândia como requisito parcial para obtenção do título de bacharel em Engenharia Mecatrônica.
Orientador Prof. Dr. Fernando Lourenço de Souza

Uberlândia
2024

João Lucas Almeida Moura

REDES NEURAIS: UMA APLICAÇÃO DE VISÃO COMPUTACIONAL

Trabalho de Conclusão de Curso submetido ao curso de Engenharia Mecatrônica da Universidade Federal de Uberlândia como requisito parcial para obtenção do título de bacharel em Engenharia Mecatrônica.

Uberlândia, Julho 10, 2024.

Banca de Avaliação:

Prof. Dr. Fernando Lourenço
Universidade Federal de Uberlândia - FEMEC

Prof. Dr Márcio Peres de Souza
Universidade Federal de Uberlândia - FEMEC

Profa. Dra. Larissa Rocha Pereira
Universidade Federal de Uberlândia - FEMEC

AGRADECIMENTOS

Agradeço, em primeiro lugar, aos meus pais, por todo o incentivo e apoio ao longo dessa jornada. Seu esforço e dedicação ao me proporcionar a oportunidade de estudar foram essenciais para que eu pudesse chegar até aqui. Sempre me inspiraram a seguir em frente e a nunca desistir dos meus sonhos. Aos professores sou profundamente grato por compartilharem o conhecimento e serem uma fonte constante de aprendizado e inspiração. Suas orientações foram fundamentais para minha formação, tanto profissional quanto pessoal. Por fim, agradeço aos meus colegas e companheiros de curso pela parceria e pelo companheirismo. Juntos enfrentamos desafios e celebramos conquistas, e cada um de vocês contribuiu para que essa trajetória fosse mais leve e significativa.

"Tudo o que um homem é capaz de imaginar, um outro é capaz de realizar."
Júlio Verne

RESUMO

Redes neurais artificiais têm se tornado essenciais em diversas áreas devido à sua capacidade de resolver problemas complexos. A análise de diferentes arquiteturas é importante para otimizar o desempenho dos modelos em tarefas específicas. Este Trabalho de Conclusão de Curso explora a aplicação de redes neurais artificiais, com foco nas arquiteturas de neurônios totalmente conectadas e convolucionais, em problemas de visão computacional. O objetivo principal é comparar a performance de diferentes arquiteturas utilizando o dataset Fashion MNIST, composto por imagens de artigos de moda. Foram implementados e treinados modelos de redes sequenciais e convolucionais com técnicas de regularização, como Batch Normalization e Dropout, para analisar o impacto dessas abordagens no desempenho da classificação de imagens. Os resultados indicam que redes convolucionais superam as redes sequenciais em termos de acurácia e capacidade de generalização, especialmente quando combinadas com técnicas de normalização e regularização. Conclui-se que a arquitetura convolucional, embora mais complexa, oferece melhores resultados em tarefas de visão computacional.

Palavras-chave: Redes Neurais, Visão Computacional, Batch Normalization, Dropout, Fashion MNIST

ABSTRACT

Artificial neural networks have become essential in various fields due to their ability to solve complex problems. Analyzing different architectures is important to optimize model performance in specific tasks. This undergraduate thesis explores the application of artificial neural networks, focusing on fully connected and convolutional neuron architectures, in computer vision problems. The main objective is to compare the performance of these architectures using the Fashion MNIST dataset, composed of images of fashion articles. Sequential and convolutional network models were implemented and trained with regularization techniques such as Batch Normalization and Dropout to analyze the impact of these approaches on image classification performance. The results indicate that convolutional networks outperform sequential networks in terms of accuracy and generalization capacity, especially when combined with normalization and regularization techniques. It is concluded that the convolutional architecture, although more complex, offers better results in computer vision tasks.

Keywords: Neural Networks, Computer Vision, Batch Normalization, Dropout, Fashion MNIST

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Com o crescente avanço das tecnologias computacionais, o uso de técnicas de Inteligência Artificial (IA) tem se popularizado para a análise de grandes volumes de dados complexos. Dentre essas técnicas, as redes neurais se destacam por sua capacidade de modelar dados em diversos formatos, como textos, imagens e sons, permitindo que padrões ocultos sejam identificados e processados de forma eficiente. Esse avanço é particularmente relevante em áreas que demandam precisão e rapidez, como a visão computacional. Nos últimos anos, as redes neurais têm ganhado ainda mais relevância em diferentes domínios, sendo amplamente aplicadas em tarefas de classificação, detecção de objetos e análise preditiva. Originadas há várias décadas, essas estruturas foram aprimoradas ao longo do tempo, evoluindo de modelos simples, como o Perceptron, para arquiteturas mais complexas; redes multicamadas e convolucionais; que oferecem maior capacidade de generalização e adaptabilidade em tarefas complexas. Dada a crescente demanda por soluções automatizadas que possam processar e interpretar grandes volumes de dados visuais, este trabalho visa explorar a eficácia das redes neurais em tarefas de visão computacional. A escolha por esse tema se justifica pelo impacto dessas tecnologias em áreas como monitoramento de segurança, automação industrial e análise de imagens médicas, onde a precisão e a eficiência são fundamentais. Especificamente, a utilização do dataset Fashion MNIST, composto por imagens de artigos de moda, que apresenta um cenário desafiador ao permitir avaliar o desempenho de diferentes arquiteturas de redes neurais em tarefas de classificação de imagens. Esse banco de imagens foi escolhido devido a complexidade das imagens que exigem uma estrutura mais elaborada para conseguir uma precisão considerável. Assim, o objetivo geral deste trabalho é comparar o desempenho de redes neurais sequenciais e convolucionais em tarefas de classificação de imagens, destacando suas vantagens, limitações e características específicas.

2 REDES SEQUENCIAIS

As redes neurais são uma das metodologias mais utilizadas atualmente na criação de inteligência artificial, com aplicações que vão desde algoritmos de visão computacional até processamento de sons, interpretação de linguagem natural, entre outros. A primeira concepção de um neurônio artificial foi idealizada em 1943 por Warren S. McCulloch e Walter Pitts, marcando o início do desenvolvimento das redes neurais artificiais. Posteriormente, em 1958, Frank Rosenblatt desenvolveu o Perceptron, um modelo de neurônio artificial mais avançado que se tornou a base para redes neurais modernas.

Apesar de suas origens antigas, o avanço e a aplicação das redes neurais permaneceram limitados por várias décadas, principalmente devido às restrições de processamento computacional da época. Foi somente com o avanço da capacidade computacional e o surgimento de novos algoritmos que as redes neurais passaram a ser amplamente exploradas e implementadas em diversas áreas da inteligência artificial.

2.1 NEURÔNIO ARTIFICIAL

O modelo de neurônio proposto por McCulloch e Pitts, em 1943, é um dos primeiros modelos neurônios artificiais e se utilizado da para resolver operações booleanas. Este modelo considera o neurônio como uma unidade que realiza operações lógicas básicas. O modelo é descrito pela Equação 1:

$$y = \sigma\left(\sum_{i=1}^{n} w_i x_i - b\right) \tag{1}$$

Onde:

- n é o número de amostras;
- y é o valor predito pelo modelo;
- x_i são as entradas do sistema;
- w_i é o peso associado a cada entrada;
- b é o *bias* (viés);
- σ é a função de ativação, que no modelo original é uma função degrau.

McCulloch e Pitts, em 1943, basearam seu modelo em observações de neurônios biológicos, especificamente no conceito de que um neurônio dispara um impulso nervoso quando a soma ponderada de seus sinais de entrada excede um certo limiar. Um problema marcante desse modelo é a falta de uma metodologia de autoaprendizado do sistema, ou seja, os pesos de cada entrada deveriam ser definidos manualmente de forma a atingir o resultado desejado.

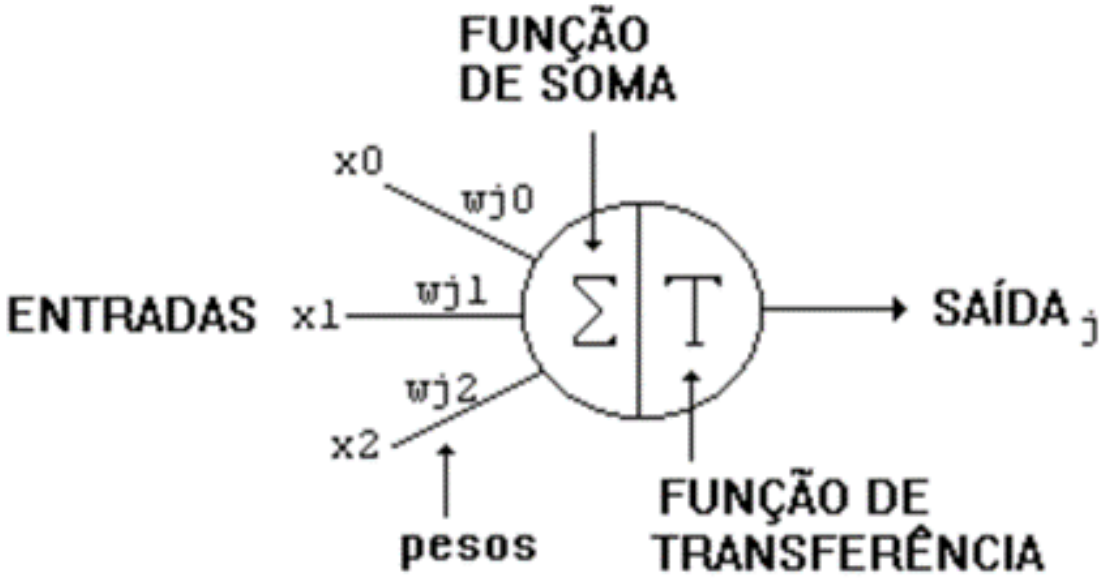
No entanto, a principal falha do modelo de McCulloch-Pitts reside na sua simplicidade e rigidez. Este modelo, apesar de suas limitações, demonstrou que redes de neurônios

artificiais poderiam realizar qualquer computação que pudesse ser descrita logicamente, estabelecendo as bases teóricas para o desenvolvimento das redes neurais.

2.2 PERCEPTRON DE ROSENBLATT

O Perceptron, introduzido por Frank Rosenblatt em 1958, representou um avanço significativo no campo das redes neurais. Este modelo é capaz de aprender e adaptar-se aos dados de entrada, ajustando os pesos de forma a minimizar o erro na classificação. Esse modelo é constituído de uma camada interligada de Unidades Lógicas Lineares.

Figura 1 – Unidade Lógica Linear (TLU)



Fonte: https://cerebromente.org.br/n05/tecnologia/rna.htm

Na Figura 1, é possível vizualizar a representação de um neurônio artificial, conforme proposto por Rosenblatt. A equação que descreve a saída do Perceptron é similar ao modelo de McCulloch-Pitts, mas inclui um processo de aprendizagem. Outro ponto importante a salientar é que nesse modelo ainda se usa a função de ativação sendo uma função degrau, o que resultava em transtornos ao se tentar representar padrões mais complexos, observe a equação que descreve a saida dos neurônios na Equação 1.

O processo de aprendizagem do Perceptron envolve a atualização dos pesos w_i e do b de acordo com a regra de aprendizado do Perceptron, visível na Equação 2 e Equação 3:

$$w_i = w_i + \eta(d - y)x_i \tag{2}$$

$$b_i = b_i + \eta(d - y) \tag{3}$$

Onde:

- d é o valor desejado (*target*);
- η é a taxa de aprendizagem.

Essa capacidade de ajustar os pesos permitiu que o Perceptron resolvesse problemas de classificação linearmente separáveis, embora apresentasse limitações para problemas não linearmente separáveis, como o problema XOR.

O problema XOR (ou "OU Exclusivo") é um problema clássico em aprendizado de máquina e redes neurais que destaca as limitações dos modelos de Perceptron de camada única. XOR é uma função lógica que retorna verdadeiro se, e somente se, as entradas diferirem. No contexto de um Perceptron de camada única, a principal limitação é que ele só pode resolver problemas que são linearmente separáveis, ou seja, problemas onde as classes podem ser separadas por uma linha reta em um espaço bidimensional. No caso do problema XOR, não é possível separar as classes com uma única linha reta, pois os pontos (0, 1) e (1, 0) pertencem a uma classe, e os pontos (0, 0) e (1, 1) pertencem a outra, formando um padrão que requer pelo menos uma curva ou duas linhas para ser separado corretamente.

Essa limitação foi um obstáculo significativo na evolução das redes neurais durante as décadas de 1960 e 1970. Somente com a introdução de redes neurais de múltiplas camadas, utilização de outras funções de ativação e do algoritmo de retropropagação (*backpropagation*) foi possível superar essa barreira. Redes multicamadas conseguem criar representações internas que transformam problemas não linearmente separáveis em problemas linearmente separáveis em um espaço de características mais alto, permitindo a resolução do problema XOR e outros problemas complexos.

2.3 REDES MULTICAMADAS (*MLP*)

As Redes Neurais Multicamadas, também conhecidas como *MLP (Multilayer Perceptron)*, representam uma classe mais avançada de redes neurais (HAYKIN, 1999). Uma *MLP* típica consiste em uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída (BISHOP, 1995). Cada neurônio em uma camada está conectado a cada neurônio na próxima camada, formando uma estrutura totalmente conectada. A saída de uma *MLP* é dada pela Equação 4:

$$Y = \sigma(WX + B) \tag{4}$$

Onde:

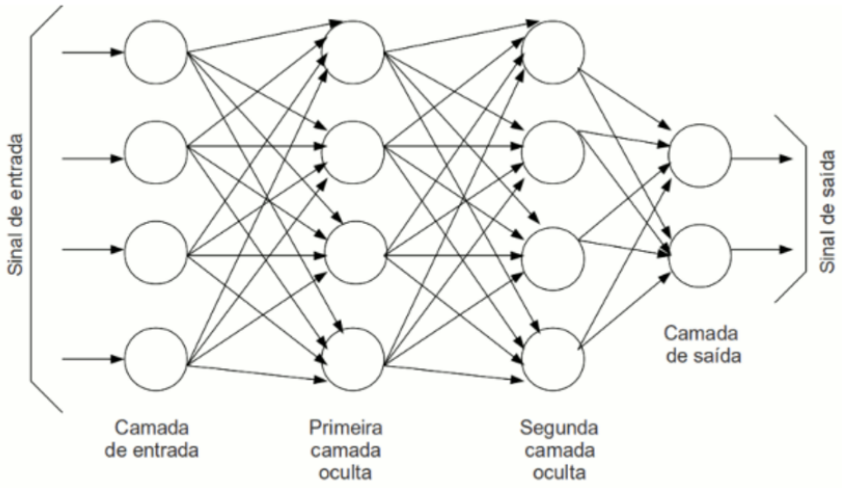
- Y é o vetor de saída;
- W é a matriz de pesos;
- X é o vetor de entrada;
- B é o vetor de *bias*.

A matriz de pesos W, o vetor de entrada X e o vetor de *bias* B são representados como:

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{bmatrix} \quad X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad B = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \tag{5}$$

Cada neurônio em uma camada está conectado a todos os neurônios na camada seguinte, permitindo que a rede capture e aprenda representações mais complexas dos dados de entrada, e convirja mais rápido se comparada ao Perceptron, como possível de se ver na Figura 2.

Figura 2 – Perceptron Multicamada (*MLP*)



Fonte: https://www.monolitonimbus.com.br/redes-neurais-artificiais/

As camadas intermediárias entre a entrada e a saída são chamadas de camadas ocultas; as camadas perto da entrada são chamadas de camadas inferiores; e as camadas perto da saída são chamadas de camadas superiores. Contudo, devido ao aumento de camadas a regra de aprenzidado proposta por Frank Rosenblatt não consegue lidar com esse aumento de complexidade de forma satisfatoria, havendo a necessidade de se desenvolver outros sistemas de aprendizado.

2.3.1 Sistema de Retropropagação

O modelo de Perceptron de Rosenblatt introduziu sistemas de aprendizado que permitem modificar os pesos para obter saídas cada vez mais próximas dos resultados desejados (ROSENBLATT, 1958). Este modelo considera uma taxa de aprendizado e as entradas do sistema para ajustar os pesos. No entanto, quando aplicado a redes multicamadas, o cálculo dos ajustes dos pesos se torna mais complexo e o tempo de convergência aumenta (MINSKY; PAPERT, 1969). Para resolver esses problemas, desenvolveu-se o sistema de retropropagação.

O sistema de retropropagação foi introduzido por Rumelhart, Hinton e Williams, em 1986. Este método permite a atualização eficiente dos pesos em redes neurais multicamadas ao calcular o gradiente negativo da função de erro em relação a cada peso.

O processo começa com a comparação da saída da rede neural com os resultados reais ou desejados, utilizando uma função de erro, como o erro quadrático médio (*MSE - Mean Squared Error*) (HAYKIN, 1999), observado na Equação 6:

$$E = \frac{1}{n}\sum_{i=1}^{n}(d_i - y_i)^2 \tag{6}$$

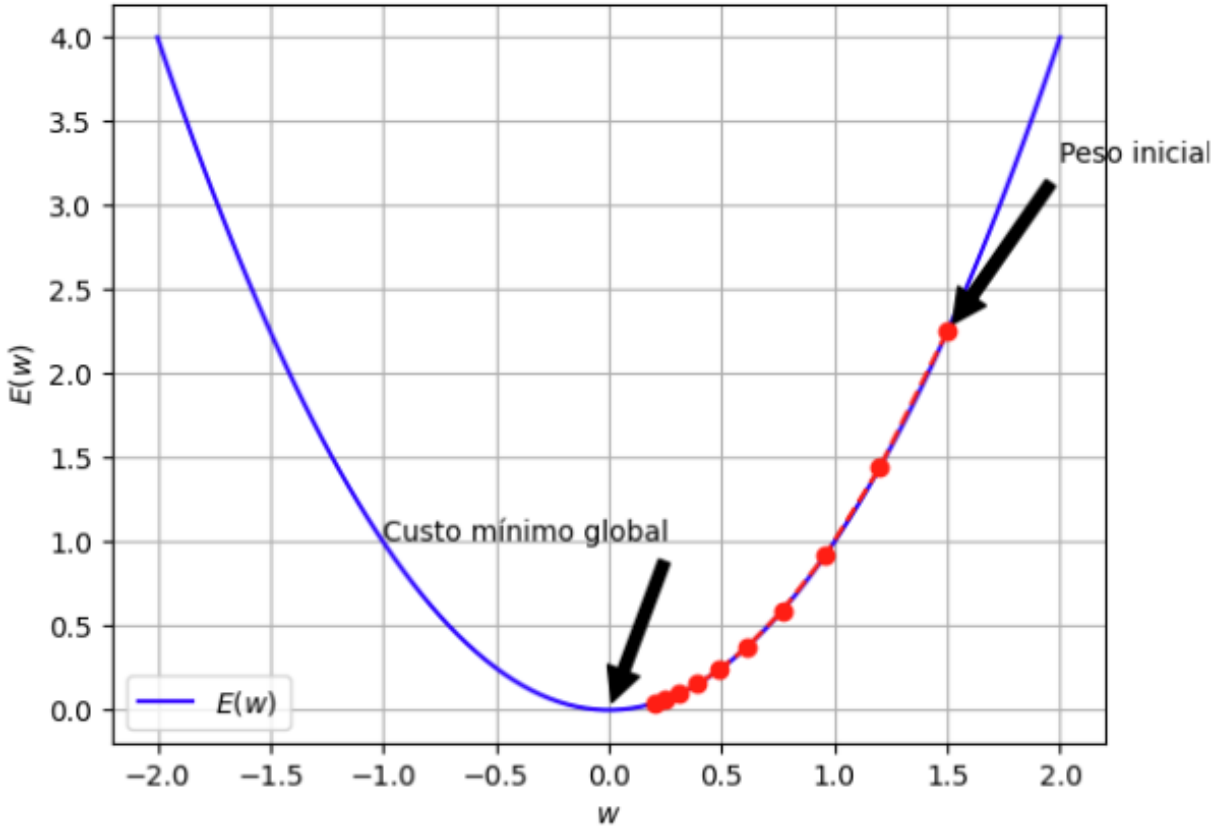
A regra da cadeia é então usada para calcular o gradiente da função de erro em

relação a cada peso w_{ij} (GOODFELLOW; BENGIO; COURVILLE, 2016). A derivada parcial do erro em relação a um peso específico é calculada na Equação 7:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial w_{ij}} \tag{7}$$

A retropropagação utiliza o conceito de gradiente, que aponta para a direção na qual a função de erro aumenta mais rapidamente. Ao se mover na direção oposta ao gradiente, ou seja, na direção do negativo do gradiente, a função de erro diminui de forma mais eficiente (BISHOP, 1995). Essa abordagem garante que a rede neural ajuste seus pesos de maneira a reduzir o erro de forma mais rápida e precisa, observe na Figura 3.

Figura 3 – Trajetória da Descida do Gradiente para Minimização da Função de Custo



Como visível na imagem acima, o gradiente varia em seu percurso em direção ao mínimo global. Quanto mais distante o peso inicial estiver do ponto de mínimo, maior será a magnitude do gradiente, resultando em ajustes mais significativos nos pesos. À medida que os pesos se aproximam do mínimo global, a magnitude do gradiente diminui, resultando em ajustes menores e mais precisos. Esse comportamento adaptativo permite que o processo de otimização seja eficiente, evitando grandes oscilações ao redor do mínimo e garantindo uma convergência mais estável e rápida (RUMELHART; HINTON; WILLIAMS, 1986).

Em redes neurais treinadas por retropropagação, o processo de aprendizagem é realizado ao longo de múltiplas iterações denominadas épocas. Cada época consiste em

apresentar todo o conjunto de dados de treinamento à rede, permitindo que os pesos sejam ajustados iterativamente. Durante uma época, para cada exemplo de treinamento, a saída da rede é calculada e comparada com o valor desejado. O erro resultante é então propagado para trás através da rede. Dessa forma, uma época envolve a atualização de todos os pesos e vieses da rede uma ou várias vezes. Os pesos e o viés são atualizados conforme as Equações 8 e 9:

$$w_{ij} = w_{ij} - \eta \frac{\partial E}{\partial w_{ij}} \tag{8}$$

$$b_j = b_j - \eta \frac{\partial E}{\partial b_j} \tag{9}$$

Ao final de cada época, todos os pesos na rede terão sido atualizados com base em todas as amostras de treinamento, promovendo uma aprendizagem abrangente. Esse processo iterativo permite que a rede ajuste gradualmente seus parâmetros internos para minimizar a função de erro global.

À medida que o treinamento avança através de várias épocas, a rede neural tende a convergir para um conjunto de pesos que reduz significativamente o erro entre as saídas previstas e os valores desejados. Este refinamento contínuo dos pesos melhora a capacidade da rede de generalizar para novos dados não vistos, aumentando sua eficácia em tarefas de previsão e classificação (GOODFELLOW; BENGIO; COURVILLE, 2016).

2.3.2 Funções de Ativação Distintas

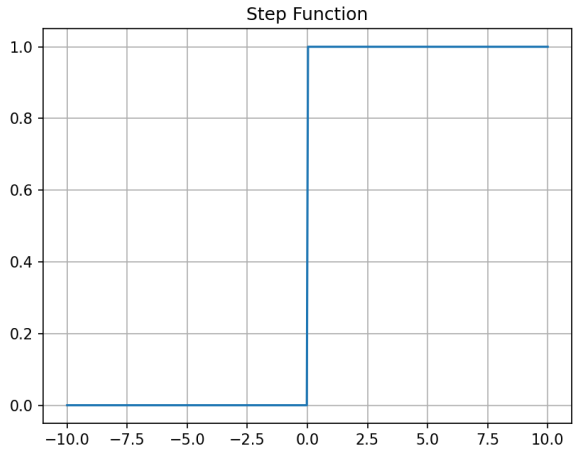
As funções de ativação são componentes essenciais nas redes neurais artificiais, influenciando diretamente a capacidade da rede de aprender e representar padrões complexos nos dados de entrada. Elas introduzem não linearidades nos modelos, permitindo que as redes neurais resolvam problemas que não podem ser resolvidos com modelos lineares simples. Antes de discutir as funções de ativação, é importante entender dois problemas comuns no treinamento de redes profundas: desvanecimento do gradiente e explosão do gradiente (GOODFELLOW; BENGIO; COURVILLE, 2016).

O problema de desvanecimento do gradiente ocorre quando os gradientes das camadas iniciais da rede se tornam extremamente pequenos, tornando a atualização dos pesos ineficaz e resultando em um treinamento muito lento ou até estagnado (BENGIO et al., 1994). Por outro lado, o problema de explosão do gradiente acontece quando os gradientes se tornam excessivamente grandes, causando grandes oscilações nos pesos e instabilidade no treinamento. Ambos os problemas são fortemente influenciados pela escolha das funções de ativação e pela inicialização aleatória dos pesos.

As funções de ativação transformam a soma ponderada das entradas de um neurônio em uma saída que pode ser utilizada na próxima camada da rede. Sem funções de ativação, uma rede neural composta por várias camadas equivaleria a uma única camada linear, incapaz de modelar a complexidade dos dados (HAYKIN, 1999).

A função degrau (*Step Function*) foi uma das primeiras funções de ativação utilizadas, mapeando a entrada para zero ou um, dependendo de um limiar (MCCULLOCH; PITTS, 1943). Embora tenha introduzido uma forma rudimentar de não linearidade, sua incapacidade de realizar diferenciação contínua limitou seu uso em modelos mais complexos, abaixo a representação gráfica da função na Figura 4.

Figura 4 – Função Degrau (*Step Function*)

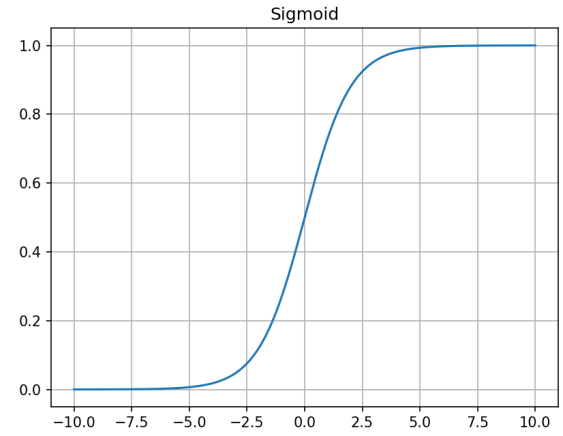


A função sigmoide (*Sigmoid*), função logística, mapeia a entrada em um intervalo entre 0 e 1, utilizando a Equação 10:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{10}$$

Assim sendo, pode-se obter a seguinte representação gráfica na Figura 5:

Figura 5 – Função Logística (*Sigmoid*)



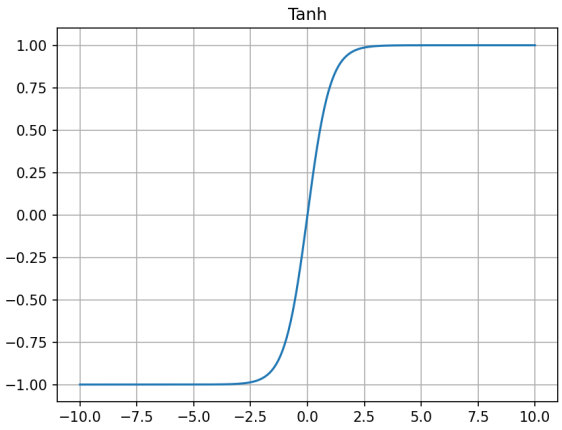
Esta função é útil para modelar probabilidades e valores contínuos em uma faixa limitada. No entanto, a função sigmoide pode causar o problema de desvanecimento do gradiente em redes profundas, onde os gradientes das camadas iniciais se tornam muito pequenos para atualizar os pesos efetivamente (GOODFELLOW; BENGIO; COURVILLE, 2016).

A função tangente hiperbólica (*Tanh*) mapeia a entrada em um intervalo entre -1 e 1, utilizando a Equação 11:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{11}$$

Obtendo assim a Figura 6:

Figura 6 – Tangente Hiperbólica



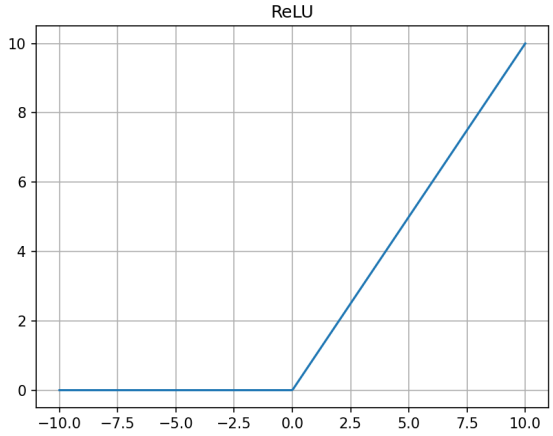
Oferecendo uma saída centrada em zero, pode ajudar na convergência mais rápida em comparação com a função sigmoide. No entanto, assim como a sigmoide, a tangente hiperbólica também sofre do problema de desvanecimento do gradiente em redes profundas (BENGIO et al., 1994).

A *ReLU* (*Rectified Linear Unit*) mapeia a entrada para zero se for negativa, ou a mantém inalterada se for positiva, utilizando a Equação 12:

$$\text{ReLU}(x) = \max(0, x) \tag{12}$$

Com isso tem-se a seguinte representação gráfica da Figura 7:

Figura 7 – *Rectified Linear Unit*



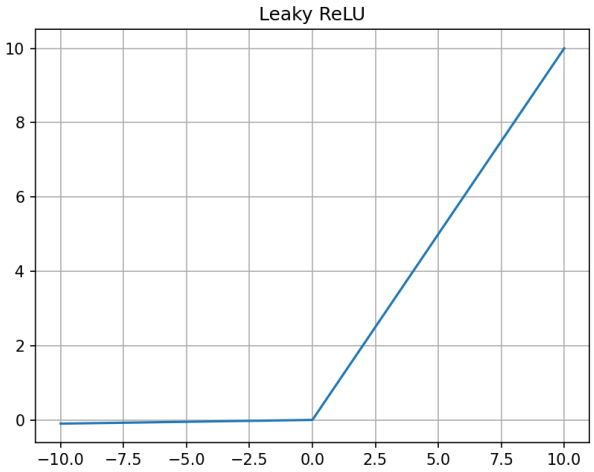
A *ReLU* introduz uma não linearidade que permite a diferenciação e acelera a convergência (NAIR; HINTON, 2010). Contudo, pode causar o problema de *neurônios mortos* durante o treinamento se a entrada for constantemente negativa, levando a neurônios que nunca são atualizados.

A *Leaky ReLU* é similar à *ReLU*, mas permite pequenos gradientes para valores negativos, utilizando a Equação 13:

$$\text{Leaky ReLU}(x) = \max(\alpha x, x) \tag{13}$$

Onde α é um pequeno valor positivo, a representação gráfica da função esta na Figura 8:

Figura 8 – *Leaky Rectified Linear Unit*



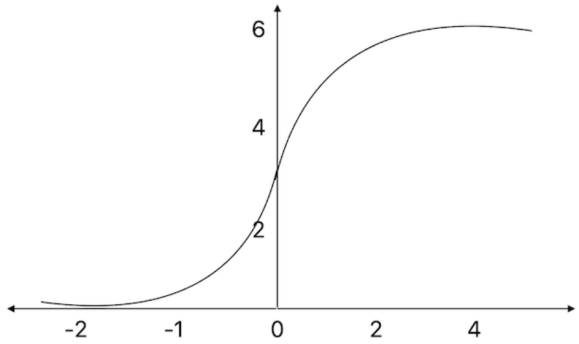
A função resolve o problema dos neurônios mortos na *ReLU*, mas ainda pode sofrer de instabilidade em algumas configurações (MAAS; HANNUN; NG, 2013).

A função *softmax* mapeia um vetor de valores para uma distribuição de probabilidade, utilizando a Equação 14:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{14}$$

Observe a representação gráfica na Figura 9:

Figura 9 – *Softmax Function*



Fonte: https://pt.linux-console.net/?p=25833

A função *Softmax* é particularmente útil para tarefas de classificação multiclasse, transformando os valores de entrada em uma probabilidade que soma 1 (BISHOP, 1995).

A escolha da função de ativação pode impactar significativamente a performance da rede neural. Funções como $ReLU$ e suas variantes são amplamente utilizadas em redes profundas modernas devido à sua simplicidade e eficiência computacional (GOODFELLOW; BENGIO; COURVILLE, 2016). No entanto, é crucial entender as limitações e características de cada função para evitar problemas como desvanecimento do gradiente ou *neurônios mortos*, garantindo um treinamento mais eficiente e modelos mais robustos.

Além disso, a média das saídas das funções de ativação pode ter um impacto significativo no treinamento da rede neural. Por exemplo, funções de ativação como a sigmoide e a tangente hiperbólica têm saídas médias diferentes (0,5 e 0, respectivamente), o que pode afetar a distribuição dos gradientes durante o treinamento (GLOROT; BENGIO, 2010). Uma saída média não centrada em zero, como no caso da sigmoide, pode causar um desbalanceamento na atualização dos pesos, levando a uma convergência mais lenta. Em contraste, funções de ativação como a $ReLU$, que têm uma média de saída positiva, pode acelerar a convergência, mas ao custo de potencialmente causar um desbalanceamento na distribuição das ativações.

A inicialização aleatória dos pesos também desempenha um papel crucial na dinâmica do treinamento. Inicializações inadequadas podem exacerbar os problemas de desvanecimento do gradiente ou explosão do gradiente (HE et al., 2015).

2.4 NORMALIZAÇÃO EM *BATCH*

A Normalização em *Batch* (*BN*) é uma técnica introduzida por Ioffe e Szegedy, em 2015, que visa melhorar a eficiência e a estabilidade do treinamento de redes neurais profundas. A *BN* normaliza as ativações de cada camada para que tenham média zero e variância unitária durante o treinamento, permitindo o uso de taxas de aprendizado mais altas e tornando o processo de treinamento mais estável e eficiente.

Matematicamente, para um mini-batch $B = \{x_1, x_2, \ldots, x_m\}$ de ativações de uma camada, a *BN* calcula a média μ_B e a variância σ_B^2 do *mini-batch* nas Equações 15 e 16:

$$\mu_B = \frac{1}{m} \sum_{i=1}^{m} x_i \tag{15}$$

$$\sigma_B^2 = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_B)^2 \tag{16}$$

Com esses valores, as ativações são normalizadas na Equação 17:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{17}$$

Onde ϵ é um pequeno valor constante adicionado para evitar divisão por zero. Após a normalização, as ativações normalizadas $\hat{x}_i$ são escaladas e deslocadas utilizando os parâmetros γ e β, que são aprendíveis durante o treinamento da Equação 18:

$$y_i = \gamma \cdot \hat{x}_i + \beta \tag{18}$$

A normalização das ativações estabiliza a distribuição dos gradientes durante o treinamento, mitigando problemas comuns em redes profundas, como os desvanecimento do gradiente e explosão do gradiente (BENGIO et al., 1994; GLOROT; BENGIO, 2010). Isso ocorre porque a normalização garante que os gradientes permaneçam em uma faixa útil, evitando que se tornem muito pequenos ou excessivamente grandes ao serem propagados pelas camadas da rede.

Além disso, a *BN* reduz a sensibilidade da rede à inicialização dos pesos, tornando o treinamento mais robusto e menos suscetível a problemas causados por escolhas inadequadas de inicialização (HE et al., 2015). Sem *BN*, a escolha da inicialização pode influenciar fortemente a convergência do modelo; entretanto, a normalização das ativações em cada camada torna o treinamento mais robusto a essa variabilidade inicial.

Outro benefício importante da Normalização em *Batch* é sua contribuição como uma forma de regularização. Durante o treinamento, a *BN* adiciona um leve ruído ao processo devido ao uso de mini-batches, o que age de maneira similar a outras técnicas de regularização, como *dropout*, ajudando a prevenir o *overfitting* (SRIVASTAVA et al., 2014). O *overfitting* ocorre quando um modelo de aprendizado de máquina se ajusta excessivamente aos dados de treinamento, capturando não apenas os padrões relevantes,

mas também o ruído e as variações aleatórias presentes nos dados. Isso resulta em um desempenho inferior em dados novos e não vistos, pois o modelo não generaliza bem além do conjunto de treinamento. Ao introduzir esse leve ruído e normalizar as ativações intermediárias, a *BN* ajuda a reduzir a possibilidade de *overfitting*, melhorando a capacidade de generalização do modelo e promovendo um desempenho mais robusto em aplicações práticas.

A *BN* também permite o uso de funções de ativação saturantes, como a sigmoide ou tangente hiperbólica, sem os problemas associados de gradientes desaparecendo, pois a normalização das ativações ajuda a manter os gradientes em uma faixa adequada (IOFFE; SZEGEDY, 2015).

A implementação prática da *BN* é direta e pode ser aplicada após camadas convolucionais ou densas, mas antes das funções de ativação (NAIR; HINTON, 2010). Isso melhora a estabilidade do treinamento, além de acelerar a convergência do modelo, permitindo que ele atinja melhor performance em menos iterações.

Portanto, a Normalização em *Batch* é uma técnica poderosa que resolve diversos problemas no treinamento de redes neurais profundas, desde a estabilização dos gradientes até a regularização do modelo. Sua inclusão em redes neurais modernas é quase padrão, devido aos benefícios em termos de desempenho e robustez do treinamento.

2.5 *DROPOUT*

O *Dropout* é uma técnica de regularização introduzida por Srivastava et al., em 2014, que visa prevenir o*overfitting* em redes neurais profundas, melhorando a capacidade de generalização do modelo. O *Dropout* funciona "desligando"aleatoriamente uma fração de neurônios durante o treinamento, forçando a rede a não depender excessivamente de nenhum neurônio específico e a aprender representações mais robustas.

Durante o treinamento, o *Dropout* desativa, de forma aleatória, uma proporção de neurônios em cada camada da rede. A probabilidade de "desligamento"de cada neurônio é definida por um hiperparâmetro p, que geralmente varia entre 0,2 e 0,5, dependendo da arquitetura da rede e do problema em questão (HINTON et al., 2012). Matematicamente, se h_i representa a ativação de um neurônio i na camada oculta, a aplicação do *Dropout* pode ser representada pela Equação 19:

$$\tilde{h}_i = h_i \cdot r_i \tag{19}$$

onde r_i é uma variável binária aleatória, tal que:

$$r_i \sim \text{Bernoulli}(p)$$

Essa operação implica que, durante o treinamento, apenas uma sub-rede, composta pelos neurônios não "desligados", é efetivamente treinada em cada iteração (SRIVASTAVA

et al., 2014).

O *Dropout* contribui para a melhoria da capacidade de generalização da rede ao reduzir a coadaptação entre neurônios, um fenômeno onde neurônios se tornam excessivamente dependentes de outros neurônios específicos para produzir a saída correta (HINTON et al., 2012). Isso força a rede a aprender representações mais distribuídas e menos dependentes de características específicas, resultando em um modelo mais robusto.

Durante a fase de inferência, todos os neurônios são utilizados, mas as ativações são escaladas pela probabilidade p de não serem desligadas durante o treinamento. Esta operação garante que a saída da rede na inferência seja consistente com a saída média durante o treinamento (SRIVASTAVA et al., 2014). Matematicamente, se $\hat{y}_i$ é a saída do neurônio i durante a inferência, ela é dada pela Equação 20:

$$\hat{y}_i = h_i \cdot p \tag{20}$$

Embora o *Dropout* aumente a robustez e a generalização da rede, ele também pode levar a um aumento no tempo de treinamento, pois o modelo precisa compensar a perda temporária de informações durante o treinamento. No entanto, o ganho em termos de redução do *overfitting* geralmente compensa o aumento no tempo de treinamento (SRIVASTAVA et al., 2014).

Além disso, o *Dropout* pode ser combinado com outras técnicas de regularização, como *Normalização em Batch*, para melhorar ainda mais a estabilidade e a performance da rede (IOFFE; SZEGEDY, 2015). Em redes neurais profundas, o *Dropout* é frequentemente aplicado em camadas densas (*fully connected*), mas também pode ser utilizado em camadas convolucionais com ajustes adequados nos hiperparâmetros (TOMCZAK, 2013).

O *Dropout* é uma técnica de regularização eficaz que ajuda a reduzir o *overfitting* em redes neurais profundas, promovendo uma maior capacidade de generalização. Sua aplicação tem se tornado comum em modelos modernos de aprendizado profundo devido aos seus benefícios em termos de robustez e desempenho. A inclusão do *Dropout* em uma rede neural deve ser cuidadosamente ajustada, considerando a arquitetura da rede e o problema específico a ser resolvido (GOODFELLOW; BENGIO; COURVILLE, 2016).

3 REDES NEURAIS CONVOLUCIONAIS (*CNNS*)

As Redes Neurais Convolucionais (*CNNs*) são uma classe de redes neurais projetadas especificamente para processar dados que possuem uma estrutura em grade, como imagens (LECUN et al., 1998; GOODFELLOW; BENGIO; COURVILLE, 2016). As *CNNs* revolucionaram a área de visão computacional, permitindo avanços significativos em tarefas como classificação de imagens, detecção de objetos e reconhecimento facial (KRIZHEVSKY; SUTSKEVER; HINTON, 2012; LECUN; BENGIO; HINTON, 2015). Inspiradas pela estrutura e funcionamento do córtex visual dos animais, as *CNNs* são capazes de aprender representações hierárquicas e invariantes a partir dos dados de entrada (FUKUSHIMA, 1980; HUBEL; WIESEL, 1962).

Um ponto crucial é que, em redes neurais convencionais com camadas totalmente conectadas, ao se analisar imagens, é necessário garantir que o número de neurônios na camada de entrada seja equivalente ao número de elementos (pixels) contidos na imagem (GOODFELLOW; BENGIO; COURVILLE, 2016). Em imagens modernas de alta resolução, isso exigiria uma quantidade enorme de neurônios, aumentando significativamente o poder computacional necessário. As redes convolucionais, por outro lado, utilizam camadas convolucionais e de *pooling*. As camadas convolucionais filtram características específicas da imagem, destacando-as, enquanto as camadas de *pooling* reduzem a dimensionalidade da imagem, preservando as características mais importantes para a análise ou classificação, mesmo em resoluções menores (GU et al., 2018).

3.1 CAMADA CONVOLUCIONAL

A camada convolucional é o componente fundamental das *Redes Neurais Convolucionais (CNNs)*. Em vez de conectar cada neurônio a todos os neurônios da camada anterior, como nas redes totalmente conectadas, uma camada convolucional conecta cada neurônio apenas a uma pequena região da camada anterior, conhecida como campo receptivo. Essa abordagem permite que a rede aprenda características locais, preservando a estrutura espacial dos dados de entrada (LECUN et al., 1998).

Os principais componentes de uma camada convolucional são os filtros (ou *kernels*). Esses filtros são pequenos vetores de pesos que percorrem a entrada, realizando operações de convolução para produzir mapas de ativação. Cada filtro é treinado para detectar características específicas, como bordas, texturas ou padrões mais complexos (SZEGEDY et al., 2015). A operação de convolução pode ser descrita pela seguinte Equação 21:

$$(f * x)(i, j) = \sum_{m} \sum_{n} x(i + m, j + n) f(m, n) \tag{21}$$

Com base na Equação 21, pode-se utilizar alguns *kernels* conhecidos para exemplifi-

car, como o *kernel* para detecção de linhas verticais e horizontais (GONZALEZ; WOODS, 2008), Equação 22:

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{22}$$

Para exemplificar o uso de filtros, considere a Figura 10 de um templo chinês disponível no próprio *framework TensorFlow*, uma biblioteca utilizada para criar inteligências artificiais em *Python*.

Figura 10 – Templo Chinês antes do filtro.

Fonte: https://scikit-learn.org/stable/auto_examples/cluster/plot_color_quantization.html#sphx-glr-auto-examples-cluster-plot-color-quantization-py.

Após a aplicação dos filtros, a imagem resultante, em preto e branco, foi obtida utilizando um *stride* de um, observe na Figura 11.

Figura 11 – Templo Chinês após a aplicação dos filtros: (a) Filtro Vertical e (b) Filtro Horizontal.

Para controlar a dimensão dos mapas de ativação, as camadas convolucionais utilizam técnicas como *stride* e *padding*. O *stride* (passo) determina o número de *pixels* que o filtro se desloca na imagem a cada aplicação. Um *stride* maior reduz o tamanho do mapa de ativação, enquanto um *stride* menor mantém mais detalhes espaciais. Por exemplo, um *stride* de 1 move o filtro um *pixel* de cada vez, enquanto um *stride* de 2 move o filtro dois *pixels* de cada vez, resultando em um mapa de ativação menor (GOODFELLOW; BENGIO; COURVILLE, 2016).

O *padding* adiciona bordas artificiais à imagem de entrada para garantir que o filtro possa ser aplicado em todas as regiões da imagem. Existem dois tipos principais de *padding*: *válido* e *mesmo* (*same*). O *padding válido* não adiciona bordas, resultando em um mapa de ativação menor do que a entrada. O *padding mesmo* (ou *zero-padding*) adiciona zeros nas bordas da imagem, preservando o tamanho da entrada após a convolução (GU et al., 2018).

3.2 CAMADA DE *POOLING*

A camada de *pooling*, ou amostragem, é utilizada para reduzir a dimensionalidade espacial dos mapas de ativação, mantendo as características mais importantes e reduzindo a carga computacional da rede. O *pooling* ajuda a tornar a representação mais robusta a variações na posição dos recursos detectados e a reduzir o risco de *overfitting*. A operação de *pooling* mais comum é o *max-pooling*, que toma o valor máximo dentro de uma janela de tamanho fixo ao percorrer o mapa de ativação (SZEGEDY et al., 2015).

Por exemplo, em uma operação de *max-pooling* com uma janela de 2x2 e *stride* 2, a saída para cada janela seria o valor máximo dos quatro valores dentro dessa janela, Equação 23:

$$y(i,j) = \max\{x(i+m, j+n)\} \tag{23}$$

Outras formas de *pooling* incluem o *average pooling*, que calcula a média dos valores dentro da janela, e o *global pooling*, que reduz cada mapa de ativação a um único valor, calculando a média ou o valor máximo de todos os valores no mapa de ativação. Essas técnicas ajudam a resumir a informação e a manter as características mais salientes dos dados de entrada, permitindo que as camadas subsequentes da rede se concentrem em padrões de nível mais alto (GOODFELLOW; BENGIO; COURVILLE, 2016).

4 METODOLOGIA

A estrutura básica de uma rede neural é relativamente simples de ser implementada. No entanto, quando sistemas como retropropagação, normalização em *Batch* e outras estruturas são adicionados, o código pode se tornar menos flexível e significativamente mais complexo. Para lidar com essa complexidade e aumentar a eficiência no treinamento e na execução dos modelos, existem algumas estruturas prontas disponíveis.

Atualmente, há vários *frameworks* com esse objetivo, sendo os dois mais populares: *TensorFlow* e o *PyTorch*.

Ambos os *frameworks* são projetos de código aberto (*open source*) e oferecem suporte à execução dos modelos em *GPUs* (Unidades de Processamento Gráfico), o que maximiza a velocidade de treinamento e execução de redes neurais. Neste projeto, optou-se pelo uso do *TensorFlow*, devido à sua eficiência e simplicidade na implementação.

Além disso, foi utilizado o *dataset Fashion MNIST*, que é uma variante do *MNIST* tradicional, composta por imagens de artigos de moda. Este *dataset* foi escolhido por fornecer todos os rótulos necessários para as tarefas de classificação propostas neste trabalho, e por ser mais complexo do que o *MNIST*, sendo necessário modelos mais sofisticados para alcançar alta precisão, observe uma amostra do *dataset* na Figura 12.

Figura 12 – *Fashion MNIST*

No caso, a rede, após ser treinada, receberá a imagem conforme apresentada na Figura 12 e a classificará em uma das 10 classes possíveis (Camiseta/Top, Calça, Suéter, Vestido, Casaco, Sandália, Camisa, Tênis, Bolsa, Bota). O *dataset* foi dividido em dois: um contendo 80% das imagens, destinado ao conjunto de treinamento, e outro contendo 20% das imagens, destinado ao conjunto de validação. O primeiro grupo foi utilizado para realizar a retropropagação, ou seja, para atualizar os pesos e vieses da rede. O segundo conjunto serve para analisar o comportamento da rede neural com dados que ela nunca

viu, permitindo verificar se a rede sofreu ou não *overfitting*.

Considerando essas informações, testou-se quatro arquiteturas diferentes para analisar a diferença entre os resultados de performance do modelo. Durante o treinamento, métricas como acurácia de treinamento e perda de treinamento foram monitoradas para avaliar o desempenho da rede no conjunto de treinamento. A acurácia de treinamento representa a proporção de exemplos do conjunto de treinamento que a rede classificou corretamente, indicando o quão bem o modelo está aprendendo os padrões presentes nesses dados. A perda de treinamento, geralmente calculada por uma função de custo como o erro médio, quantifica o erro médio entre as predições do modelo e os rótulos reais, servindo como um indicador de quão bem o modelo está sendo ajustado aos dados de treinamento.

Paralelamente, a acurácia de validação e a perda de validação foram utilizadas para avaliar a capacidade de generalização do modelo em dados não vistos durante o treinamento. A acurácia de validação indica a proporção de exemplos corretamente classificados no conjunto de validação, fornecendo uma estimativa de como o modelo pode performar em dados reais ou novos. A perda de validação mede o erro do modelo no conjunto de validação, sendo crucial para identificar problemas como o *overfitting*, onde o modelo pode apresentar baixo erro no conjunto de treinamento, mas alto erro no conjunto de validação devido ao ajuste excessivo aos dados de treinamento (GOODFELLOW; BENGIO; COURVILLE, 2016).

A análise dessas métricas ao longo das épocas permite ajustar os hiperparâmetros e a arquitetura da rede neural para melhorar o desempenho geral. Um comportamento desejável é observar tanto a perda de treinamento quanto a perda de validação diminuindo de forma consistente, enquanto as acurácias correspondentes aumentam. Discrepâncias significativas entre essas métricas podem indicar a necessidade de ajustes, como a aplicação de técnicas de regularização ou a modificação da complexidade do modelo para alcançar um equilíbrio adequado entre aprendizagem e generalização (BISHOP, 1995). A diferença no tempo de treinamento entre épocas com diferentes estruturas de camadas foi muito baixa; portanto, o tempo não foi um parâmetro considerável, sendo o número de épocas mais relevante.

5 RESULTADOS

Conside a Tabela 1 como a comparação entre a perfomance dos modelos.

Tabela 1 – Comparação dos Resultados dos Modelos

Modelo	Acurácia Treinamento	Perda Treinamento	Acurácia Validação	Perda Validação
Sequencial (3 camadas)	94,28%	16,59%	89,86%	28,29%
Sequencial + BN	97%	9%	89%	34%
Sequencial + BN + Dropout	89,9%	26,88%	90,12%	27%
Convolucional + BN + Dropout	93,22%	18,99%	91,48%	23,71%

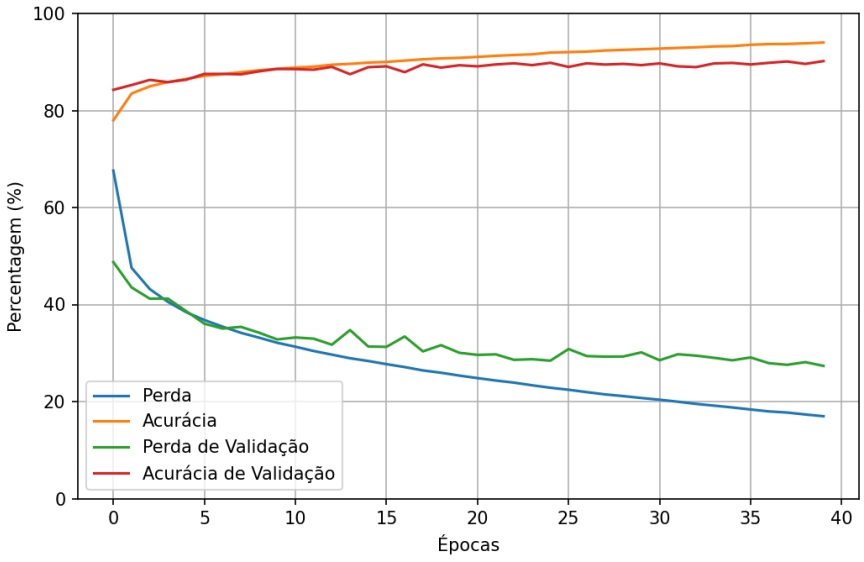
Observa-se que o modelo convolucional com normalização em *batch* e *dropout* apresentou os melhores resultados no conjunto de validação, indicando uma boa capacidade de generalização considerando os parametros utilizados no treinamento.

Além disso, a inclusão de técnicas como a normalização em *batch* mostrou-se fundamental para melhorar o desempenho dos demais modelos, será visto com mais detalhes na representação gráfica. O impacto do *dropout* também foi perceptível.

5.1 REDE NEURAL SEQUENCIAL COM TRÊS CAMADAS

Uma rede neural com três camadas pode ser capaz de lidar com dados complexos, desde que a quantidade de neurônios em cada camada seja suficiente. No entanto, a adição de mais camadas pode acelerar a velocidade de convergência do treinamento e ajudar a reduzir o problema de *overfitting*. Com base nisso, foi treinada uma rede utilizando o *dataset Fashion MNIST* com três camadas. O gráfico abaixo ilustra o progresso do treinamento ao longo das épocas. Confira os valores nominais ao final do treinamento na Tabela 1, visualiza-se progressão do modelo ao decorrer das 40 épocas utilizadas no treinamento, observe na Figura 13.

Figura 13 – Relatório de treinamento: Rede Sequencial.

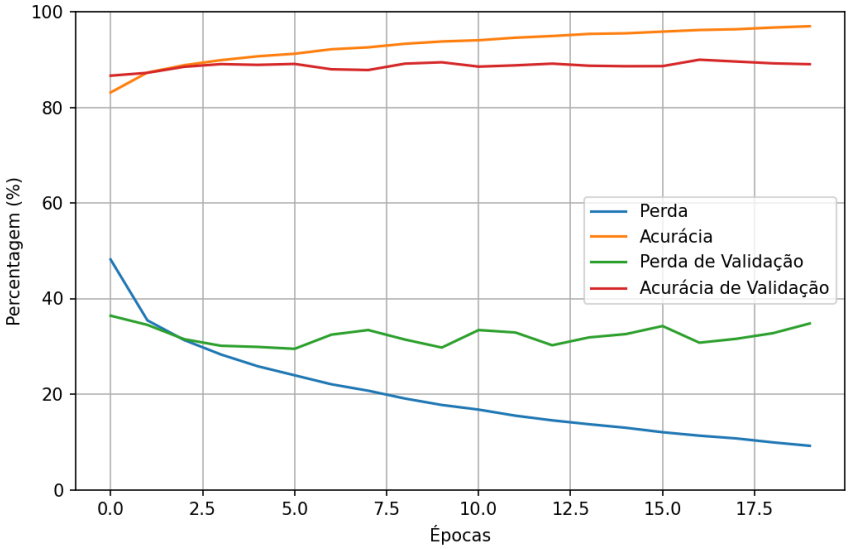


Com base no grafico representado na Figura 13, observou-se um certo distanciamento entre as curvas de desempenho do conjunto de treinamento e as do conjunto de validação. Isso demonstra um grau de *overfitting*, evidenciado pela diferença entre a acurácia e a perda do treinamento e da validação.

5.2 REDE NEURAL SEQUENCIAL COM NORMALIZAÇÃO EM *BATCH*

Este modelo utilizou camadas de normalização em *batch* após cada camada densa de neurônios, mantendo as demais características do modelo anterior. Devido à capacidade dessa técnica de acelerar a velocidade de convergência, foram utilizadas apenas 20 épocas, metade da quantidade usada no modelo anterior, observe na Figura 14.

Figura 14 – Relatório de treinamento: Rede Sequencial com normalização em *Batch*.



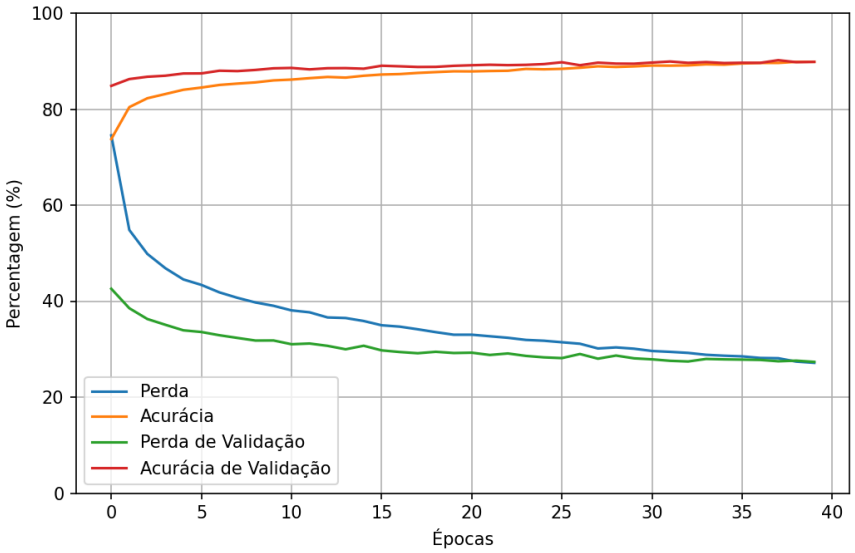
Com a inclusão da normalização em *batch*, é possível visualizar um menor grau de oscilação das curvas de acurácia e perda, tanto no treinamento quanto na validação. Essa característica está mais visível nas curvas de acurácia. No restante, é possível perceber uma maior distância entre as curvas de acurácia e perda, tanto no conjunto de treinamento quanto no conjunto de validação, se comparado à Figura 13. Isso demonstra que o modelo teve um *overfitting* maior, os valores nominais ao final do treinamento são encontrados na Tabela 1.

5.3 REDE NEURAL SEQUENCIAL COM NORMALIZAÇÃO EM *BATCH* E *DROPOUT*

Para tentar mitigar o problema de *overfitting* observado nos modelos anteriores, este modelo faz uso da técnica de *Dropout*. Foi utilizada a mesma arquitetura do modelo anterior, adicionando camadas de *Dropout*. considerando a complexidade adicional intro-

duzida pelo *Dropout*, o número de épocas foi aumentado para 40, visando minimizar os efeitos adversos dessa técnica, Figura 15.

Figura 15 – Relatório de treinamento: Rede Sequencial com normalização em *Batch* e *Dropout*.

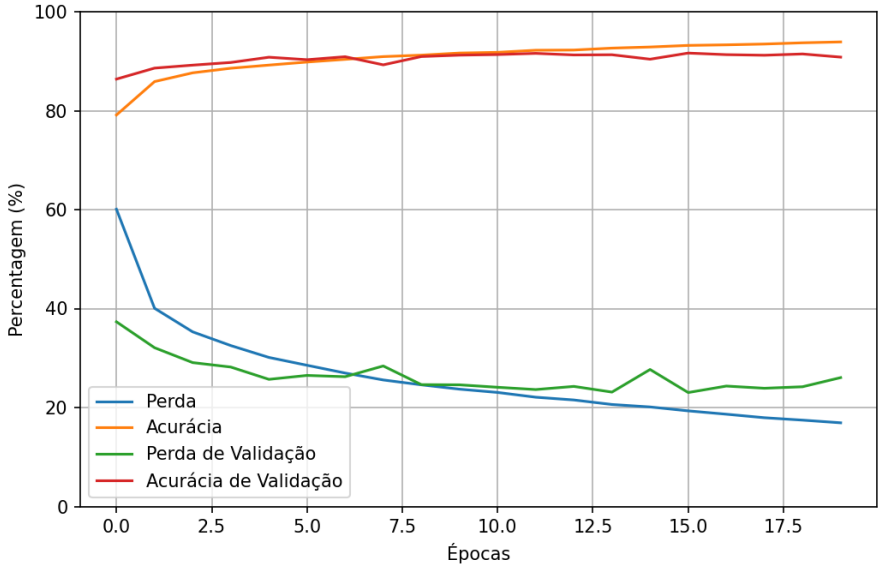


Na Figura 15 é possível ver uma oscilação ainda menor em todas as curvas presentes no grafico. É possível perceber também que as curvas de acurácia e de perda, tanto do conjunto de treinamento quanto de validação, convergem para o mesmo ponto, mostrando assim que o modelo responde bem a imagens com as quais não foi treinado, ou seja, o modelo esta generalizando bem o padrão aprendido co conjunto de treinamento, os valores nominais sãoo apresentados na Tabela 1.

5.4 REDE NEURAL CONVOLUCIONAL COM NORMALIZAÇÃO EM *BATCH* E *DROPOUT*

A Figura 16 mostra a evolução da rede neural que incorpora todos os métodos discutidos neste trabalho, utilizando três camadas convolucionais e duas camadas densas. Nesse treinamento, foi utilizada somente 20 épocas devido à melhoria em termos de convergência advinda da rede convolucional na análise de imagens.

Figura 16 – Relatório de treinamento: Rede Convolucional com normalização em *Batch* e *Dropout*.



É possível notar no gráfico acima uma oscilação maior das curvas comparadas à Figura 15. As curvas de acurácia e de perda não convergiram como à da Figura 15, etretanto ficaram bem proximas se comparada a da Figura 13 e 14. Contudo, esse modelo teve o melhor resultado nominal no conjunto de validação, Tabela 1, mostrando assim que o modelo aprendeu padrões úteis na classificação das imagens, sendo, portanto, o melhor modelo dentre os analisados nesse trabalho.

5.5 ARQUITETURA PROPOSTA

Experimentalmente, chegou-se à conclusão que a arquitetura da rede neural convolucional descrita abaixo foi a mais eficaz para o problema em questão. A estrutura utiliza uma combinação de camadas convolucionais, camadas de *pooling*, normalização por normalização em *Batch*, e regularização por *Dropout*, buscando maximizar a acurácia e minimizar o *overfitting*.

A arquitetura consiste nas seguintes camadas:

```
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape
       =(28, 28, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.BatchNormalization(),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.BatchNormalization(),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.5),
    layers.Dense(10, activation='softmax')
])
```

Quadro 5.1 – Definição de Camadas TensorFlow

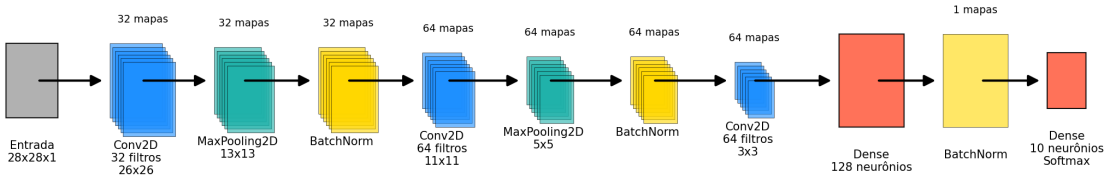
Essa arquitetura começa com uma camada convolucional de 32 filtros e um tamanho de *kernel* de 3x3, seguida por uma camada de *MaxPooling* que reduz a dimensionalidade espacial. A inclusão de normalização em *Batch* após cada camada convolucional garante a normalização das ativações, acelerando o processo de treinamento e ajudando a evitar o problema de *vanishing gradients*.

As camadas convolucionais subsequentes de 64 filtros continuam a extrair características mais complexas da imagem, enquanto as camadas de *MaxPooling* e normalização em *Batch* mantêm a estabilidade e eficiência do modelo. Após as camadas convolucionais, a rede é "achatada"por meio da camada *Flatten*, transformando as ativações 2D em um vetor 1D.

Uma camada densa de 128 neurônios com ativação *ReLU* é adicionada, seguida de normalização em *Batch* e *Dropout* com uma taxa de 50 %, o que ajuda a reduzir o *overfitting* ao desligar aleatoriamente neurônios durante o treinamento. Finalmente, uma camada *Dense* com 10 neurônios e ativação *softmax* realiza a classificação das 10 classes do *dataset Fashion MNIST*.

Essa arquitetura se mostrou eficaz na tarefa de classificação, combinando a extração de características robustas com regularização e técnicas de normalização para alcançar alta acurácia e boa generalização, abaixo tem-se uma representação grafica da arquitetura, na Figura 17.

Figura 17 – Modelo de arquitetura



6 CONCLUSÃO

Este trabalho explorou a aplicação de redes neurais, tanto sequenciais quanto convolucionais, no campo da visão computacional. Usando o conjunto de dados Fashion MNIST, foi possível realizar uma análise comparativa entre diferentes arquiteturas, mostrando assim a vantagem da arquitetura proposta no trabalho em detrimento das demais arquiteturas analisadas em termos de precisão e generalização. Os resultados obtidos evidenciam que a arquitetura convolucional apresenta desempenho superior em tarefas de classificação de imagens.

A utilização de redes convolucionais mostrou-se eficaz na extração de características visuais relevantes, permitindo uma maior acurácia nas classificações realizadas. As técnicas de regularização foram fundamentais para mitigar o overfitting, contribuindo para uma melhor generalização dos modelos. Esses dados confirmam que, em cenários onde a análise de imagens é necessária, as redes convolucionais oferecem uma solução robusta e eficiente.

Como possíveis evoluções deste trabalho, outras arquiteturas de redes neurais podem ser investigadas, como as redes neurais recorrentes (RNNs), além disso, redes mais avançadas, como as redes generativas adversariais (GANs) e as arquiteturas baseadas em transformadores, têm ganhado destaque na pesquisa em visão computacional, mostrando potencial para superar as arquiteturas convolucionais tradicionais em certas tarefas, como a geração de imagens e a detecção de objetos mais complexos.

REFERÊNCIAS

ABADI, Martín et al. TensorFlow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

ALBAWI, Saad; MOHAMMED, Tareq Abed; AL-ZAWI, Saad. Understanding of a Convolutional Neural Network. In: *Proceedings of 2017 International Conference on Engineering and Technology (ICET)*, p. 1–6, 2017.

BENGIO, Yoshua; SIMARD, Patrice; FRASCONI, Paolo. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, v. 5, n. 2, p. 157–166, 1994.

BISHOP, Christopher M. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

FUKUSHIMA, Kunihiko. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, v. 36, n. 4, p. 193–202, 1980.

GLOROT, Xavier; BENGIO, Yoshua. Understanding the difficulty of training deep feedforward neural networks. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, p. 249–256, 2010.

GONZALEZ, Rafael C.; WOODS, Richard E. *Digital Image Processing*. 3ª ed. Prentice Hall, 2008.

GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. *Deep Learning*. MIT Press, 2016.

GU, Jiuxiang et al. Recent advances in convolutional neural networks. *Pattern Recognition*, v. 77, p. 354–377, 2018.

HAYKIN, Simon. *Neural Networks: A Comprehensive Foundation*. 2ª ed. Prentice Hall, 1999.

HE, Kaiming; ZHANG, Xiangyu; REN, Shaoqing; SUN, Jian. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, p. 1026–1034, 2015.

HINTON, Geoffrey E.; SRIVASTAVA, Nitish; KRIZHEVSKY, Alex; SUTSKEVER, Ilya; SALAKHUTDINOV, Ruslan. Improving neural networks by

preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

HORNIK, Kurt; STINCHCOMBE, Maxwell; WHITE, Halbert. Multilayer feedforward networks are universal approximators. *Neural Networks*, v. 2, n. 5, p. 359–366, 1989.

HUBEL, David H.; WIESEL, Torsten N. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *Journal of Physiology*, v. 160, n. 1, p. 106–154, 1962.

IOFFE, Sergey; SZEGEDY, Christian. Batch Normalization: Accelerating deep network training by reducing internal covariate shift. In: *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, p. 448–456, 2015.

KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. ImageNet classification with deep convolutional neural networks. In: *Advances in Neural Information Processing Systems (NIPS)*, p. 1097–1105, 2012.

LECUN, Yann et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, v. 86, n. 11, p. 2278–2324, 1998.

LECUN, Yann; BENGIO, Yoshua; HINTON, Geoffrey. Deep learning. *Nature*, v. 521, p. 436–444, 2015.

MAAS, Andrew L.; HANNUN, Awni Y.; NG, Andrew Y. Rectifier nonlinearities improve neural network acoustic models. In: *Proceedings of the 30th International Conference on Machine Learning (ICML)*, 2013.

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

MINSKY, Marvin; PAPERT, Seymour. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, 1969.

MORENO, R. Redes neurais artificiais. 2019. Disponível em: <https://www.monolitonimbus.com.br/redes-neurais-artificiais/>. Acesso em: 10 out. 2023.

NAIR, Vinod; HINTON, Geoffrey E. Rectified linear units improve restricted Boltzmann machines. In: *Proceedings of the 27th International Conference on Machine Learning (ICML)*, p. 807–814, 2010.

NIELSEN, Michael A. *Neural Networks and Deep Learning*. Determination Press, 2015.

O'SHEA, Keiron; NASH, Ryan. An Introduction to Convolutional Neural Networks. *arXiv preprint arXiv:1511.08458*, 2015.

ROSENBLATT, Frank. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, v. 65, n. 6, p. 386–408, 1958.

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

SRIVASTAVA, Nitish; HINTON, Geoffrey; KRIZHEVSKY, Alex; SUTSKEVER, Ilya; SALAKHUTDINOV, Ruslan. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, v. 15, n. 1, p. 1929–1958, 2014.

SZEGEDY, Christian et al. Going deeper with convolutions. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, p. 1–9, 2015.

TAYE, Mohammad Mustafa. Theoretical Understanding of Convolutional Neural Network: Concepts, Architectures, Applications, Future Directions. *Journal of Imaging*, v. 9, n. 3, p. 75, 2023.

TOMCZAK, Jakub M. Cross-validation selector and dropout. *arXiv preprint arXiv:1305.6068*, 2013.