UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
CAMPUS CHAPECÓ
CURSO DE LICENCIATURA EM MATEMÁTICA

TIAGO CRUZARO

O MÉTODO DE APRENDIZAGEM PROFUNDA

CHAPECÓ
2025

TIAGO CRUZARO

O MÉTODO DE APRENDIZAGEM PROFUNDA

Trabalho de Conclusão de Curso apresentado ao Curso de Licenciatura em Matemática da Universidade Federal da Fronteira Sul (UFFS), como requisito para obtenção do título de Licenciado em Matemática.

Orientador: Prof. Dr. Paulo Rafael Bösing

CHAPECÓ
2025

Bibliotecas da Universidade Federal da Fronteira Sul - UFFS

Cruzaro, Tiago
O MÉTODO DE APRENDIZAGEM PROFUNDA / Tiago Cruzaro. -- 2025.
83 f.:il.

Orientador: Doutor Paulo Rafael Bösing

Trabalho de Conclusão de Curso (Graduação) - Universidade Federal da Fronteira Sul, Curso de Licenciatura em Matemática, Chapecó,SC, 2025.

1. Gradiente descendente. 2. Retropropagação,. 3. Retropropagação. 4. Redes neurais. I. Bösing, Paulo Rafael, orient. II. Universidade Federal da Fronteira Sul. III. Título.

Elaborada pelo sistema de Geração Automática de Ficha de Identificação da Obra pela UFFS com os dados fornecidos pelo(a) autor(a).

TIAGO CRUZARO

O MÉTODO DE APRENDIZAGEM PROFUNDA

Trabalho de Conclusão de Curso apresentado ao Curso de Licenciatura em Matemática da Universidade Federal da Fronteira Sul (UFFS), como requisito para obtenção do título de Licenciado em Matemática.

Este trabalho foi defendido e aprovado pela banca em 17/07/2025.

BANCA EXAMINADORA

Documento assinado digitalmente
gov.br
PAULO RAFAEL BOSING
Data: 22/07/2025 20:54:26-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Paulo Rafael Bösing – UFFS
Orientador

Documento assinado digitalmente
gov.br
ANTONIO MARCOS CORREA NERI
Data: 22/07/2025 22:03:28-0300
Verifique em https://validar.iti.gov.br

Prof. Me. Antonio Marcos Correa Neri – UFFS
Avaliador

Documento assinado digitalmente
gov.br
ROSANE ROSSATO BINOTTO
Data: 23/07/2025 10:28:35-0300
Verifique em https://validar.iti.gov.br

Prof.ª Dr.ª Rosane Rossato Binotto – UFFS
Avaliadora

DEDICATÓRIA

Dedico este trabalho a minha mãe, pelo amor, apoio e incentivo em todos os momentos da minha vida.

“Being brave means knowing that when you fail, you don’t fail forever.”
(Lana Del Rey)

AGRADECIMENTOS

Agradeço, com carinho e admiração, a todos os professores que fizeram parte da minha trajetória no curso de Licenciatura em Matemática da Universidade Federal da Fronteira Sul. Cada contribuição, em aula ou fora dela, foi essencial para a construção deste percurso. Ao professor Paulo Rafael Bösing, meu orientador, sou especialmente grato pela orientação dedicada durante este Trabalho de Conclusão de Curso e também pela confiança e incentivo desde a iniciação científica. Sua escuta atenta e suas sugestões foram fundamentais para o amadurecimento deste trabalho. À professora Rosane Rossato Binotto, pela parceria durante a monitoria e pelas trocas que enriqueceram minha formação. Agradeço também aos professores, Antonio Marcos Correa Neri, Divane Marcon e Lúcia Menoncini, que estiveram à frente da coordenação do curso em diferentes momentos da minha caminhada. O trabalho e o comprometimento de vocês foram fundamentais para garantir a organização, o acolhimento e a qualidade do curso ao longo desses anos.

RESUMO

O presente trabalho tem o objetivo de mostrar o funcionamento das Redes Neurais Artificiais (RNA) através do viés matemático. Para isso, foi abordada a matemática que fundamenta o treinamento das redes. São apresentados conceitos básicos para o funcionamento das RNAs como o perceptron, as funções de ativação, funções de custo e sobre regularização. Além disso, uma parte prática com a construção e treinamento de uma rede, em *Python*, com uso do banco de dados MNIST é apresentada. O treinamento e testes práticos empregam um modelo capaz de reconhecer dígitos de 0 a 9 manuscritos e fotografados por pessoas reais. Os testes foram realizados levando em consideração a capacidade de acerto do modelo em situações em que as imagens estavam ou não no padrão dos dados presentes no MNIST. A execução desta etapa foi fundamental para explorar a importância de um bom conjunto de treinamento, visando melhorar a execução do código.

Palavras-Chave: Aprendizagem profunda, Gradiente descendente, Retropropagação, Redes neurais.

ABSTRACT

This work aims to demonstrate the functioning of Artificial Neural Networks (ANNs) through a mathematical perspective. To this end, the mathematics involved in the training of neural networks is addressed. Basic concepts essential to the functioning of ANNs are presented, such as the perceptron, activation functions, cost functions, and regularization techniques. In addition, a practical component is included with the construction and training of a network using Python and the MNIST dataset. The training and testing apply a model capable of recognizing handwritten digits from 0 to 9, photographed by real people. The tests were conducted considering the model's accuracy in situations where the images did or did not follow the visual pattern of the MNIST dataset. This stage was fundamental to exploring the importance of a well-prepared training set in improving the model's performance.

Keywords: Deep learning, Gradient descent, Back propagation, Neural network.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

A aprendizagem profunda, do inglês *Deep Learning*, é um tópico dentro da área de inteligência artificial que vem ganhando espaço na mídia. Esse modelo bastante é interessante, pois com a aprendizagem profunda os modelos conseguem detectar características importantes de conjuntos de dados, aprendendo sobre eles de maneira autônoma, sem precisar de uma pessoa dizendo para a máquina o que deve ser observado.

As Redes Neurais Artificiais (RNAs) são parte fundamental do campo do *Machine Learning*, ou aprendizagem de máquinas. Seu funcionamento, originalmente proposto por Rosenblatt (1958), baseia-se nas células neurais. A primeira ideia que fundamenta as RNAs chama-se Perceptron (ROSENBLATT, 1958), e já era capaz de realizar somas ponderadas das entradas de um modelo e aplicar funções de ativação responsáveis pela não linearidade do modelo. Atualmente, existem diversas funções de ativação, como a Sigmoide, a Tangente Hiperbólica, e a Unidade Linear Retificada (ReLU), sendo essas utilizadas em diferentes modelos e com diferentes objetivos (GHARAT, 2019).

Além das funções de ativação, existem também as funções de perda ou de custo, que buscam avaliar e ajustar o desempenho das RNAs. Essas funções ajudam no processo de treinamento e aprendizado da rede neural, pois visam minimizar os erros, deixando o resultado mais preciso. As funções de custo mais comuns são a Norma $\mathcal{L}_p$, o Erro Quadrático Médio e o Erro Médio Absoluto (FILHO, 2018).

Para a implementação das RNAs é indispensável o conhecimento sobre diversos conceitos matemáticos, como os de álgebra linear e cálculo diferencial, além de ter conhecimento acerca de linguagens de programação, (que neste trabalho é o *Python*). A abordagem matemática será complementada por uma parte prática com a implementação computacional de experimentos cujo intuito é demonstrar a eficácia das ideias apresentadas na aplicação das RNAs em problemas reais.

De maneira geral, o presente trabalho busca trazer uma visão geral a respeito do assunto de Redes Neurais Artificiais por meio de um olhar da perspectiva matemática. A metodologia utilizada traz uma parte de revisão matemática, buscando melhor compreender o funcionamento das RNAs.

1.1 JUSTIFICATIVA

A motivação de estudar o método de aprendizagem profunda vem desde a iniciação científica que abordou assuntos voltados à inteligência artificial (IA). Ademais, frequentemente ouvimos falar sobre o crescimento e desenvolvimento de tecnologias que utilizam IA para o seu funcionamento. Porém, geralmente isso se assemelha a mágica, pois não há uma explicação teórica clara sobre o seu funcionamento, o que me deixava bastante curioso e me incentivou a querer entender as bases matemáticas envolvidas e sobre como utilizar a programação para aprofundar e aplicar os conhecimentos.

Dentre os assuntos relacionados a IA, o de RNAs me chamou atenção pela semelhança com um modelo biológico de neurônios e sobre como o treinamento ou o aprendizado do sistema acontecia, sendo necessário resolver milhares ou até milhões de equações para atingir um resultado próximo do desejado. Após, as RNAs conseguem continuar "aprendendo" por conta própria, chegando cada vez mais próximas dos resultados desejados.

De maneira geral, a parte que fundamenta as RNAs é simples, sendo o principal objetivo minimizar uma função de custo. Contudo, a simplicidade é encantadora, pensando nas diferentes aplicações que essas redes podem oferecer, como processamento de imagens e de textos, reconhecimento de padrões, modelagem de dados visando prever possíveis resultados novos, entre outros.

2. CONCEITOS SOBRE REDES NEURAIS ARTIFICIAIS

Neste capítulo apresentamos os conceitos necessários para entender o método de aprendizagem profunda por meio de uma abordagem matemática. Estudar cada componente de maneira individual é essencial para entender o trabalho de todos os elementos de uma forma conjunta.

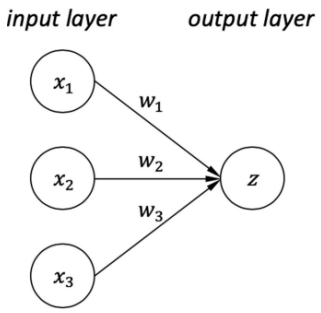
2.1 PERCEPTRON

Os modelos de base neural têm o neurônio como sua unidade básica. O neurônio é uma célula especializada na transmissão de informações, recebendo impulsos elétricos pelos seus dendritos e, quando polarizado, envia as informações para o próximo neurônio através de seus axônios. Na Aprendizagem Profunda (*Deep Learning*), os neurônios são formalizados como entrada e saída de dados. Um neurônio em uma camada agrega os sinais que estão sendo transmitidos de seus neurônios de entrada da camada anterior. Esse sinal agregado será então “processado” por uma função de ativação que determinará o comportamento do neurônio.

De maneira geral, um neurônio artificial terá pelo menos duas camadas, sendo elas entrada e saída. A equação abaixo ajuda a entender como é representado o processo de tomada de uma decisão simples:

$$z = w_1 x_1 + w_2 x_2 + w_3 x_3 \tag{2.1}$$

Figura 2.1: RNA com três neurônios de entrada e um neurônio de saída



Fonte: Dong, Ding e Zhang (2020)

Na equação (2.1), z representa o valor de saída e cada w_i representa o peso atrelado ao dado de entrada x_i, ou seja, quanto mais relevante para a tomada de decisão for w_i, maior será o seu valor absoluto. Em modelos onde o peso é igual a

zero, a informação da entrada correspondente é descartada ao final do processo de tomada de decisão. Modelos de redes neurais que seguem essa estrutura recebem o nome de Perceptron.

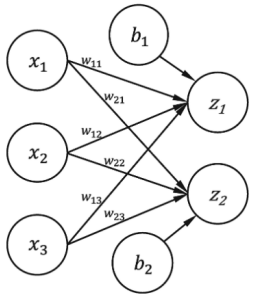
Alguns modelos apresentam um escalar extra adicionado ao modelo, chamado de *bias* (denotado por b), ou viés, para ajustar o valor de saída. Dessa forma, a equação da tomada de decisão pode ser formalizada da seguinte forma:

$$z = w_1 x_1 + w_2 x_2 + w_3 x_3 + b.$$

Uma rede neural pode ter várias camadas e várias saídas. Abaixo temos o exemplo de uma rede neural com duas saídas, que pode ser representada pelas equações:

$$z_1 = w_{11} x_1 + w_{12} x_2 + w_{13} x_3 + b_1;$$
$$z_2 = w_{21} x_1 + w_{22} x_2 + w_{23} x_3 + b_2.$$

Figura 2.2: RNA duas saídas



Fonte: Dong, Ding e Zhang (2020)

Observe na Figura 2.2 que as camadas de saída z_i estão conectadas a todas as camadas de entrada x_i. Quando isso acontece, a camada de saída é dita camada densa ou camada totalmente conectada.

Uma camada densa pode ser representada matematicamente como o produto de matrizes:

$$z = W\mathbf{x} + b$$

Em que $W \in \mathbb{R}^{m \times n}$, $z \in \mathbb{R}^m$, $\mathbf{x} \in \mathbb{R}^n$ e $b \in \mathbb{R}^m$ representam, respectivamente, a matriz com os pesos w_{ij}, as saídas, as entradas e os vieses associados aos dados. Conforme segue,

$$\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \quad .$$

2.2 FUNÇÃO DE ATIVAÇÃO

A representação de modelos de rede neural utiliza duas operações fundamentais, multiplicação e adição de matrizes, ambas lineares. O que significa que apresentam uma capacidade limitada em relação à representação de dados. De modo geral, problemas reais precisam de operadores não lineares para melhor aproximar os dados reais, e para introduzir esses operadores são utilizadas as funções de ativação, aplicadas de elemento em elemento, ou de neurônio em neurônio.

A implementação de funções de ativação permitem que alterações nos pesos e vieses mudem os valores da saída do neurônio, verificando se a informação fornecida é útil ou deve ser ignorada. Esse tipo de alteração é crucial para que os neurônios artificiais desenvolvam a aprendizagem. A função de ativação é um "portão" matemático entre a entrada que alimenta o neurônio atual e sua saída que vai para a próxima camada. As funções de ativação basicamente decidem se o neurônio deve ser ativado ou não, conforme Gharat (2019). Na sequência estão listadas as principais funções de ativação para aplicações em problemas envolvendo redes neurais. As figuras seguintes trazem a representação gráfica de cada uma das funções de ativação.

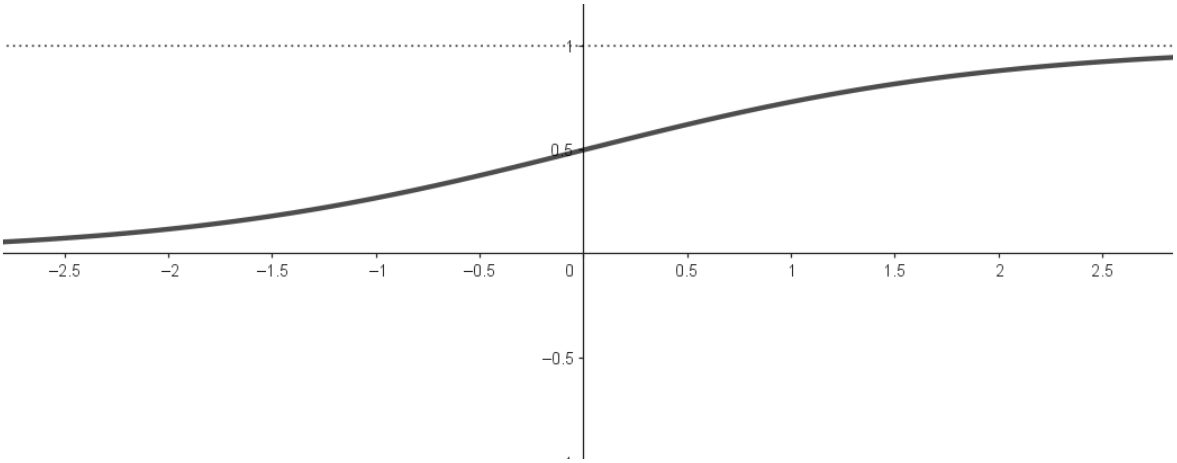
2.2.1 Função Sigmoide

A forma de não-linearidade Sigmoide, é definida pela função:

$$f(z) = \frac{1}{1 + e^{-z}}, \quad z \in \mathbb{R}. \tag{2.2}$$

Essa função é muito útil para problemas cuja saída deva ser uma porcentagem, pois para qualquer argumento (entrada), ela tem como saída valores do intervalo $(0,\ 1)$. Na Figura 2.3 apresentamos o gráfico da função de Sigmoide.

Figura 2.3: Função Sigmoide



Fonte: Elaboração do Autor (2024)

2.2.2 Função Tangente Hiperbólica (Tanh)

A função tangente hiperbólica se aplica principalmente para resolver problemas com valores de mesmo sinal (ou de classes), pois seus valores de saída estão ao intervalo de $(-1,\ 1)$. Aqui, valores próximos a 1 indicam pertencer a uma classe, enquanto valores próximos a -1 indicam pertencer a outra classe.

Para Gharat (2019) a principal vantagem dessa função é sua diferenciabilidade, ou seja, é possível encontrar sua inclinação em qualquer ponto. Além disso, como já citado, uma das vantagens é que seus valores de *output* ficam limitados ao intervalo (-1,1).

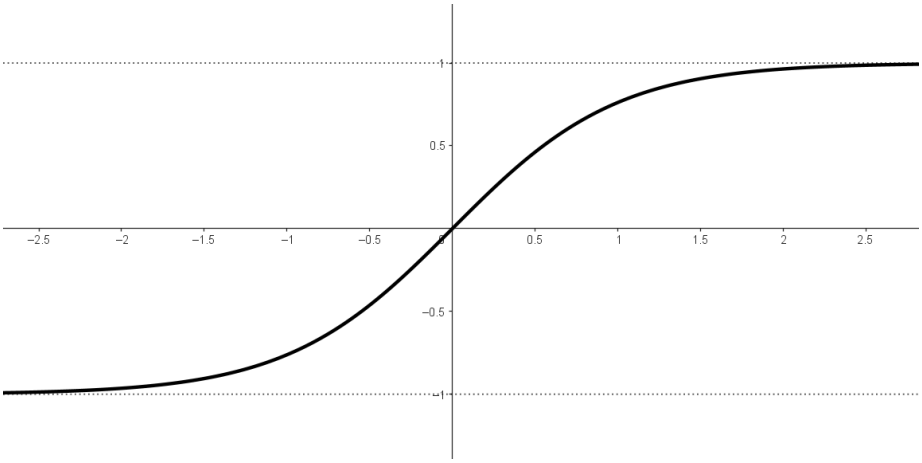
$$f(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}, \quad z \in \mathbb{R}.$$

Na realidade, a função Tanh é a função Sigmoide reescalada e transladada por um viés, dessa forma, ela apresenta todas as vantagens da função supracitada. A representação gráfica da função Tanh é ilustrada na Figura 2.4.

2.2.3 Unidade Linear Retificada (ReLu)

Do inglês *Rectified Linear Unit* (Unidade Linear Retificada), a função ReLu é uma função de ativação amplamente utilizada em modelos de redes neurais artificiais.

Figura 2.4: Função Tanh



Fonte: Elaboração do Autor (2024)

Apresenta a seguinte definição:

$$f(z) = \begin{cases} 0 & \text{quando } z \leq 0 \\ z & \text{quando } z > 0 \end{cases}, \quad z \in \mathbb{R}.$$

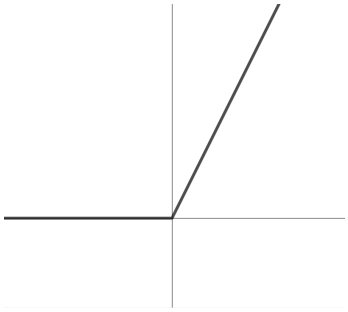
De maneira geral, a função ReLu retorna valores iguais a zero para entradas negativas, e mantém inalterados todos os valores que são positivos. Para Glorot, Bordes e Bengio (2011) essa função apresenta vantagens evidentes quando comparada à função Sigmoide ou Tanh, sendo elas:

- A facilidade no cálculo: para a implementação da ReLu, basta fazer uma comparação do seu valor com zero, e na sequência, a função é definida como zero ou como z, conforme o dado obtido na comparação. Já nas funções Sigmoide e Tanh há a exigência do cálculo de uma função exponencial, o que pode ser custoso para o processador, especialmente em redes com muitas entradas ou camadas (DONG; DING; ZHANG, 2020).
- A facilidade para otimizar: A função ReLu é quase linear, e é constituída apenas de duas funções lineares, essa propriedade faz com que seu gradiente seja grande e consistente (DONG; DING; ZHANG, 2020).

Podemos visualizar a função ReLu graficamente conforme a imagem abaixo:

Xu *et al.* (2015) estudam a eficácia da função ReLu no sentido de que ela pode gerar uma perda muito grande de informações ao considerar o valor zero como retorno para argumentos não positivos. Para solucionar o problema, os autores pro-

Figura 2.5: Função Relu



Fonte: Elaboração do Autor (2024)

põem o modelo conhecido como *Leaky ReLU*, definido da seguinte forma:

$$f(z) = \begin{cases} \alpha z & \text{quando } z \leq 0 \\ z & \text{quando } z > 0 \end{cases}, \quad z \in \mathbb{R},\ \alpha \in (0,1).$$

O escalar α representa um número positivo muito pequeno que se faz presente para que alguma informação do valor negativo possa ser retida, evitando a desativação completa do neurônio. Existe ainda a função ReLu Paramétrica, proposta por He *et al.* (2015), que traz a mesma ideia da *Leaky ReLU*, porém, nessa versão, o escalar α é um parâmetro treinado.

2.2.4 Função Softmax

Para Gharat (2019), a Função *Softmax* calcula a distribuição de probabilidades do evento em 'n' eventos diferentes. Ela é definida da seguinte forma:

$$f(z_i) = \frac{e^{z_i}}{\sum_{k=1}^{n} e^{z_k}}, \quad z \in \mathbb{R}^n,\ n \in \mathbb{N}.$$

Desse modo, além de transformar os valores de saída como nas funções supracitadas, a função também divide o valor encontrado pela soma das saídas. Com isso, a função mostra a probabilidade do valor encontrado pertencer a uma determinada classe.

> Na prática, a função softmax é geralmente usada apenas na camada de saída para normalizar o vetor de saída z em um vetor de probabilidade, onde cada entrada é não negativa e as entradas somam um. Portanto, a função *softmax* é amplamente utilizada para classificação (DONG; DING; ZHANG, 2020).

2.3 FUNÇÕES DE CUSTO

Segundo Alake (2020), as funções de custo, também chamadas de funções de perda, atuam como guias para o aprendizado de algum algoritmo, pois é através delas que é possível mensurar a taxa de aprendizagem de um modelo de *Machine Learning* (ML). Elas atuam quantificando o erro entre a previsão e o alvo do modelo de aprendizagem. Para Dong, Ding e Zhang (2020), diminuir o valor obtido na função de perda significa otimizar os parâmetros da rede neural, conseguindo valores mais próximos de dados reais. A modelagem de funções de perda é uma abordagem efetiva para melhorar o processo de otimização e, por consequência, achar modelos com melhor desempenho (GUERRERO PEÑA, 2019). Na sequência estão citadas as principais funções de perda para problemas envolvendo redes neurais artificiais.

2.3.1 Norma $\mathcal{L}_p$

A norma-p mede o "tamanho" de um vetor $\mathbf{x}$. E é definida da seguinte forma:

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}, \quad \mathbf{x} \in \mathbb{R}^n,\ p \in \mathbb{N},\ p \geq 1.$$

Em problemas de aprendizagem de máquina, a norma-p pode ser usada para medir a diferença entre dois vetores que, de maneira geral, representam o vetor com os valores reais $\mathbf{y}$, também chamado de alvo, e o vetor $\hat{\mathbf{y}}$ com as previsões do modelo. Dessa forma, a aplicação da norma fica da seguinte maneira:

$$\mathcal{L}_p = \|\mathbf{y} - \hat{\mathbf{y}}\|_p^p = \sum_{i=1}^{N} |y_i - \hat{y}_i|^p, \quad \mathbf{y}, \hat{\mathbf{y}} \in \mathbb{R}^N,\ p \in \mathbb{N},\ N \in \mathbb{N},\ p \geq 1.$$

2.3.2 Erro Quadrático Médio (MSE)

O Erro Quadrático Médio, do inglês *Mean Squared Error* (*MSE*), pode ser entendido como a média do quadrado da norma $\mathcal{L}_2$, definida conforme a equação

abaixo:

$$MSE = \frac{1}{N}\|\mathbf{y} - \hat{\mathbf{y}}\|_2^2 = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2, \quad \mathbf{y}, \hat{\mathbf{y}} \in \mathbb{R}^N,\ N \in \mathbb{N},\ N \geq 1. \tag{2.3}$$

Nesse caso, N representa o número de amostras analisadas, $\mathbf{y}$ e $\hat{\mathbf{y}}$ representam os valores reais e a previsão dos dados, respectivamente. Para Filho (2018), a grande vantagem do *MSE* é que por conta dos erros estarem elevados ao quadrado, diferenças menores acabam ganhando menos importância, enquanto as diferenças maiores ganham mais peso e se destacam.

2.3.3 Erro Médio Absoluto (MAE)

O Erro Médio Absoluto, do inglês, *Mean Absolute Error* (*MAE*), mede a média da diferença absoluta entre os valores previstos pelo modelo e os valores observados (FILHO, 2023). O *MAE* é bastante semelhante ao *(MSE)*, porém ele é uma aplicação da norma $\mathcal{L}_1$, podendo ser definido da seguinte forma:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|, \quad \mathbf{y}, \hat{\mathbf{y}} \in \mathbb{R}^N,\ N \in \mathbb{N},\ N \geq 1.$$

A análise de erro com o *MAE* é mais fácil de interpretar que os dados da *MSE*, pois seus dados são fornecidos na mesma unidade que os dados originais. Já os dados obtidos através do *MSE* estarão na unidade ao quadrado da unidade original. Contudo, esse método se destaca em aplicações onde o objetivo é penalizar erros maiores, como problemas de otimização.

2.3.4 Entropia Cruzada

A Função de Custo *Cross-Entropy*, ou Entropia Cruzada, é famosa pelo uso em tarefas em que há mais de uma entrada para ser classificada (problemas desta natureza são chamados de multiclasses). Assim como nos demais casos, ela mede a diferença entre a distribuição real dos rótulos **(y)** e a distribuição prevista pela rede **($\hat{y}$)**, representando essa diferença por meio da fórmula:

$$\ell(y, \hat{y}) = -\sum_{i=1}^{C} y_i \log(\hat{y}_i),$$

em que C é o número de classes, y_i é o valor real (1 para a classe correta, 0 para as demais) e $\hat{y}_i$ é a probabilidade prevista para a classe i. Essa função é especialmente eficaz quando combinada com a ativação *softmax* na camada de saída, pois permite simplificar o gradiente da perda em relação à saída da rede como:

$$\nabla_{z^{[L]}} \ell(y, \hat{y}) = -(y - \hat{y}),$$

o que torna o cálculo do erro mais eficiente durante a retropropagação dos gradientes, como discutido no artigo de Damadi, Moharrer e Cham (2023).

2.4 REGULARIZAÇÃO

O processo de regularização dos dados, em uma rede neural, é essencial, pois é nesse momento em que são aplicados métodos para garantir o funcionamento da rede no treinamento, nos testes e também com os novos dados. Lucas (2024, s/p.) define a regularização dos dados da seguinte forma:

> Regularização é um conjunto de métodos usados para induzir simplicidade nos modelos de *Machine Learning*, adicionando uma penalidade às suas complexidades. O objetivo é prevenir o *overfitting*, onde o modelo se ajusta demais aos dados de treinamento e perde a capacidade de generalizar para novos dados.

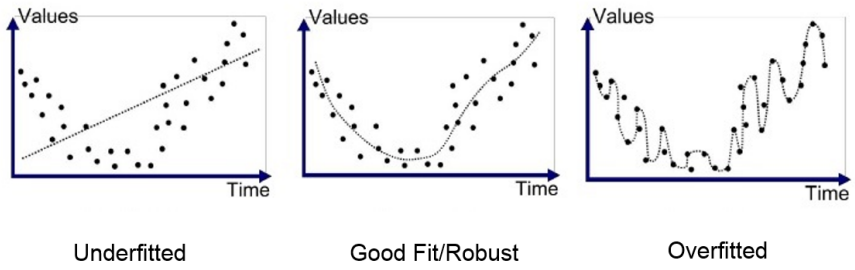
2.4.1 Overfitting e Underfitting

O processo de otimização aplicado ao conjunto de dados de treinamento busca minimizar o erro do modelo, ou seja, minimizar a função de perda. Contudo, a otimização aplicada ao conjunto de treinamento do modelo não garante o seu bom funcionamento para o conjunto de testes. Em alguns casos, o modelo foi tão otimizado ao conjunto de treinamento que acaba sendo adequado apenas para esses dados, o que é chamado de *overfitting*.

Para Branco (2020), quando isso acontece, os dados de treino apresentam resultados excelentes, porém o desempenho do modelo cai drasticamente com os dados de teste. Uma analogia, visando o entendimento do *overfitting*, é que o modelo de rede neural ficou tão bom em prever o resultado com os dados de treinamento, que acabou decorando os possíveis resultados, comprometendo a sua capacidade de fazer generalizações com outros dados.

Por outro lado, o processo de *underfitting* é justamente o oposto. Nesse caso, o modelo é muito simples e não apresenta a capacidade de ajustar os dados de treinamento para fazer previsões, resultando em erros grotescos no momento dos treinamentos e dos testes. Graficamente, pode-se representar um modelo em *underfitting* e em *overfitting* conforme a Figura 2.6.

Figura 2.6: Gráfico comparativo entre *underfitting* e *overfitting*



Fonte: Branco (2020)

2.4.2 Decaimento de Pesos

A técnica de Decaimento de Pesos, do inglês *Weight Decay*, é uma das mais aplicadas para a regularização de dados de uma rede neural. A ideia da técnica é adicionar um termo de penalidade à função de perda do modelo. A função de perda com a penalização da norma do parâmetro pode ser definida como:

$$\mathcal{L}_{total} = \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) + \lambda\Omega(\theta).$$

Aqui, $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$ é a função de perda original, computada com base no conjunto de dados reais $\mathbf{y}$ e no conjunto de dados previstos $\hat{\mathbf{y}}$. Além disso, $\Omega(\theta)$ é a função responsável pela penalização da norma do parâmetro $\theta = \theta(\mathbf{W}, \mathbf{b})$, e λ é um valor pequeno responsável por controlar a força da regularização dos dados. Aqui, $\mathbf{W}$ representa a matriz de pesos e $\mathbf{b}$ o vetor de viés das camadas da rede neural, ambos sendo os parâmetros ajustáveis durante o treinamento. As duas funções de penalização mais utilizadas são a $\Omega(\theta) = \mathcal{L}_1 = ||\mathbf{W}||_1$ e $\Omega(\theta) = \mathcal{L}_2 = ||\mathbf{W}||_2^2$. Segundo Lucas (2024), a regularização $\mathcal{L}_1$ acrescenta à rede uma penalidade referente à soma dos valores absolutos dos pesos da rede. Já a regularização $\mathcal{L}_2$ adiciona uma penalidade à função de custo proporcional à soma dos quadrados dos coeficientes dos pesos do modelo.

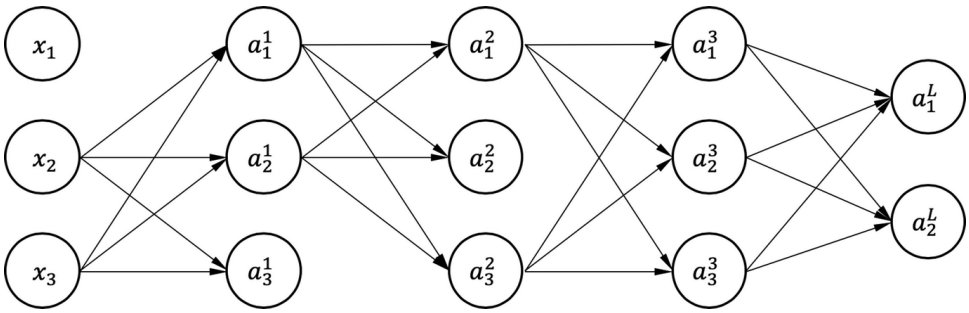
Como, de maneira geral, os parâmetros de uma rede neural são valores menores que 1, a função $\mathcal{L}_1$ pode levar a uma penalização mais significativa que $\mathcal{L}_2$, visto que $|w| > w^2$ quando $|w| < 1$. Dessa forma, a função de perda com $\mathcal{L}_1$ incentiva os parâmetros da rede a terem valores bastante pequenos, mais próximos de zero. Essa característica faz com

que a rede selecione ou descarte características de entrada para definir seus parâmetros, escolhendo o número zero para descartar valores muito pequenos. Aqui, *w* representa um peso individual da rede neural, ou seja, um coeficiente que multiplica uma das entradas em uma camada; ele faz parte da matriz de pesos **W**.

2.4.3 Dropout

Em alguns casos, redes neurais profundas com muitos pesos podem sofrer da coadaptação de neurônios, levando ao *overfitting*. Nesse processo, os neurônios de cada camada da rede artificial são dependentes dos demais neurônios, e se um deles falhar, pode resultar em uma falha para todo o sistema. Buscando solucionar esse problema, a técnica de *Dropout* proposta por Hinton *et al.* (2012) consiste em, aleatoriamente, definir algumas das saídas ocultas com o valor zero, semelhante a uma desconexão dos neurônios de uma camada para outra, conforme ilustra a Figura 2.7.

Figura 2.7: Processo de Dropout



Fonte: Dong, Ding e Zhang (2020)

Dessa forma, os parâmetros restantes da rede neural são atualizados. Esse processo é repetido mais vezes, primeiro restaurando os neurônios que foram ignorados e depois escolhendo um novo conjunto de neurônios para desconectar, permitindo a atualização dos pesos e vieses.

2.5 REDES NEURAIS CONVOLUCIONAIS (CNNS)

Desenvolvidas como uma alternativa às redes neurais baseadas em perceptrons multicamadas, as Redes Neurais Convolucionais (CNNs) foram introduzidas por Yann LeCun, inicialmente aplicadas ao reconhecimento de padrões em documentos. Segundo LeCun *et al.* (1998), as redes neurais convolucionais, projetadas especificamente para lidar com a variabilidade das formas 2D, superam todas as outras técnicas, desde então, as CNNs têm se consolidado como ferramentas essenciais em diversas áreas, incluindo visão computacional,

previsão de séries temporais, processamento de linguagem natural e aprendizado por reforço.

2.5.1 Camada Convolucional

A camada convolucional tem como inspiração os estudos publicados por Hubel e Wiesel (1962) sobre o funcionamento do córtex visual de gatos e macacos. Nesse estudo, os pesquisadores perceberam que um conjunto de neurônios com funções distintas era responsável por processar as informações de um estímulo visual, como a percepção da cor, dos ângulos ou das bordas da imagem, conseguindo assim fazer o reconhecimento visual da cena à frente. Com base na descoberta de Hubel e Wiesel (1962), o modelo proposto por Lecun *et al.* (1998) utiliza um conjunto menor de neurônios para fazer o processamento dos dados de entrada de uma imagem.

Diferente do modelo tradicional de *Multilayer Perceptron* (MLP) proposto por Rumelhart, Hinton e Williams (1986), no qual todas as camadas estão conectadas, através da camada convolucional as CNNS podem ser conectadas apenas localmente, o que diminui consideravelmente o número de conexões, facilitando o processamento da rede. Para Dong, Ding e Zhang (2020) o principal benefício da camada convolucional é que geralmente ela pode ser treinada mais rapidamente, visto que precisa de menos conexões com camadas anteriores quando comparada a uma camada densa tradicional.

As operações dentro da camada convolucional se baseiam em filtros responsáveis por entender e captar as informações mais importantes da entrada. Dizer que foi feito o processo de convolução para uma camada de entrada W significa dizer que a ela foi aplicado um filtro de tamanho F e calculado o produto escalar entre os valores de W e de F; em seguida, o filtro é deslizado para o próximo bloco de entradas. O *stride* S, ou passo, é responsável por determinar a distância entre um bloco de entradas e outro. Além disso, vale destacar que, em alguns casos, é necessário adicionar zeros às bordas para garantir que os valores sejam bem considerados; esse processo é chamado de *padding* P.

3. O MÉTODO DA APRENDIZAGEM PROFUNDA

Neste capítulo, vamos explorar o Método da Aprendizagem Profunda com foco na estrutura e no treinamento de Redes Neurais Artificiais. Iniciamos com um modelo simples, com um neurônio por camada, e avançamos para redes mais complexas com múltiplos neurônios e camadas, detalhando os processos de treinamento e aprendizagem envolvidos.

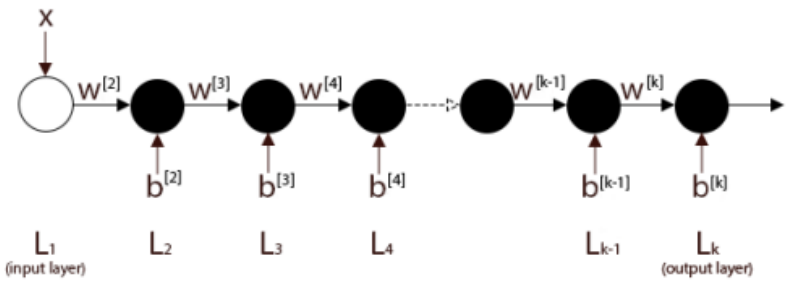
3.1 REDE NEURAL COM UM NEURÔNIO POR CAMADA

Nesta seção abordamos o aprendizado da RNA utilizando um neurônio por camada. O objetivo é apenas introduzir a ideia do aprendizado, mas sem interações com outros neurônios.

Na primeira etapa, chamada de *feedforward*, nossa Rede definirá seus pesos e vieses, então há a aplicação da função de custo para avaliar os resultados. Na segunda e última etapa, é descrita a retropropagação (*backpropagation*), onde os pesos e vieses da rede são redefinidos para melhorar os resultados da rede neural. Nesta seção, utilizamos aprendizado supervisionado, ou seja, já conhecemos os valores de entrada e de saída de um certo conjunto de dados, e o objetivo é treinar a rede para funcionar com dados novos.

Suponha um conjunto de N exemplos de treinamento, $(x^{(i)}, y(x^{(i)}))$, com $i = 1, \dots, N$. Para “alimentar" a nossa RNA com pesos e vieses, pegamos uma entrada $x^{(i)} = x$ de um exemplo de treinamento aleatório. Para esse modelo, cada camada L_i, $i = 1, \dots, k$, apresenta apenas um neurônio (conforme a Figura 3.1), cujo valor depende da saída da camada anterior. Nesse contexto, $x^{(i)}$ representa o valor de entrada, enquanto $y(x^{(i)})$ é a saída esperada (rótulo) associada à entrada.

Figura 3.1: RNA com um neurônio por camada



Fonte: Kuntze (2019)

A Figura 3.1 ilustra o modelo com uma entrada $x \in \mathbb{R}$. O peso de cada camada L_l é representado por $W^{[l]} \in \mathbb{R}$ para $l = 2, ..., k$. Do mesmo modo, o viés de cada camada L_l pode ser escrito como $b^{[l]} \in \mathbb{R}$ para $l = 2, ..., k$. A saída de cada neurônio da camada l é chamada de

$a^{[l]}$. Mais especificamente, a ativação $a^{[l]}$ é calculada a partir da ativação da camada anterior $a^{[l-1]}$ por meio da fórmula:

$$a^{[l]} = \sigma(w^l a^{[l-1]} + b^{[l]}). \tag{3.1}$$

Podemos observar com mais clareza a equação (3.1) em camadas consecutivas:

- Camada L_2:

$$a^{[2]} = \sigma(w^2 x + b^{[2]})$$

- Camada L_3:

$$a^{[3]} = \sigma(w^3 a^{[2]} + b^{[3]}) = \sigma(w^3 \sigma(w^2 x + b^{[2]}) + b^{[3]})$$

- Camada L_4:

$$a^{[4]} = \sigma(w^4 a^{[3]} + b^{[4]}) = \sigma(w^4 \sigma(w^3 \sigma(w^2 x + b^{[2]}) + b^{[3]}) + b^{[4]})$$

$$\vdots$$

- Camada L_k:

$$a^{[k]} = \sigma(w^{[k]} a^{[k-1]} + b^{[k]}) = \sigma\left(w^{[k]}\left(\sigma\left(w^{[k-1]}\left(\ldots \sigma\left(w^{[2]} x + b^{[2]}\right)\ldots\right) + b^{[k-1]}\right) + b^{[k]}\right)\right). \tag{3.2}$$

A equação (3.2) representa a saída da camada L_k, ou seja, a ativação final da rede neural após o processamento completo dos dados de entrada. Ela é construída a partir de uma sequência de operações (multiplicações por pesos $w^{[l]}$ e somas com viés $b^{[l]}$), seguidas da aplicação da função de ativação σ em cada camada.

Desde a entrada x até a última camada, as operações na rede são feitas passo a passo, de forma sequencial. Em cada etapa, aplicamos uma transformação linear (multiplicação por um peso e soma com um viés) seguida por uma função de ativação. O resultado dessa etapa é então passado como entrada para a próxima camada. Esse processo se repete até alcançar a camada final. Esse tipo de construção, onde cada camada depende diretamente da anterior e as informações fluem apenas para frente, é característico de um modelo do tipo *feedforward*. O resultado, $a^{[k]}$, pertence a $\mathbb{R}$ pois o modelo apresentado possui um único neurônio por camada.

Para que a rede aprenda, é necessário otimizar seus parâmetros (pesos e viés) de forma que a saída $a^{[k]}$ se aproxime o máximo possível do valor esperado $y(x^{(i)})$. Para isso, utilizamos uma função de custo (Seção 2.3), a qual mede o erro entre a saída da rede e o valor real.

Assumindo, por exemplo, que a função de custo utilizada, denotada por simplicidade por C, seja o erro quadrático médio (MSE), dada pela equação (2.3), temos:

$$C = \frac{1}{N}\sum_{i=1}^{N}(y(x^{(i)}) - a^{[k]}(x^{(i)}))^2. \tag{3.3}$$

Lembrando que na equação (3.3), $y(x^{(i)})$ representa a saída conhecida (ou valor esperado) para a entrada $x^{(i)}$, enquanto $a^{[k]}(x^{(i)})$ é a saída produzida pela rede na última camada, isto é, a camada L_k. O termo $\left(y(x^{(i)}) - a^{[k]}(x^{(i)})\right)^2$ corresponde ao erro ao quadrado cometido pela rede para o exemplo i. O somatório desses erros para todos os N exemplos do conjunto de treinamento, dividido por N, resulta no valor da função de custo C.

A saída $a^{[k]}(x^{(i)})$, utilizada na equação do erro quadrático médio (3.3), é resultado direto do processo de *feedforward* descrito anteriormente. Conforme mostrado na equação (3.2), essa saída é obtida por meio de uma composição de funções envolvendo os pesos e os vieses de todas as camadas da rede. Nela, a cada camada, o valor de entrada é transformado por uma operação linear (peso e viés) seguida pela aplicação da função de ativação σ. O resultado é passado como entrada para a próxima, até que a predição final $a^{[k]}$ seja obtida. Assim, cada peso $w^{[l]}$ e cada viés $b^{[l]}$, com $l = 2, \ldots, k$, participa diretamente da formação da saída da rede, e, consequentemente, da função de custo que será minimizada durante o treinamento.

Após calcular o erro por meio da função de custo, é possível atualizar os valores dos pesos e dos vieses de cada camada utilizando o processo de descida do gradiente. Para isso, calcula-se a derivada parcial da função de custo em relação a cada peso e cada viés da rede. Essas derivadas indicam a direção na qual os parâmetros devem ser ajustados para minimizar o erro, permitindo que a rede aprenda e melhore seu desempenho.

Por exemplo, em uma rede com três camadas (isto é, $k = 3$), temos, conforme a notação da Figura 3.1, os seguintes parâmetros x, $w^{[2]}$, $b^{[2]}$, $w^{[3]}$ e $b^{[3]}$. Em que:

- x: entrada da rede ($a^{[1]}$);
- $w^{[2]}$: peso entre as camadas L_1 e L_2;
- $b^{[2]}$: viés da camada L_2;
- $w^{[3]}$: peso entre as camadas L_2 e L_3;
- $b^{[3]}$: viés da camada L_3.

Para representar o processo de *feedforward*, definimos ainda as variáveis auxiliares:

$$z^{[2]} = w^{[2]}x + b^{[2]}, \quad a^{[2]} = \sigma(z^{[2]}), \quad z^{[3]} = w^{[3]}a^{[2]} + b^{[3]}, \quad a^{[3]} = \sigma(z^{[3]}),$$

Em que:

- $z^{[l]}$ representa a entrada linear do neurônio da camada l;
- $a^{[l]}$ é a saída (ou ativação) do neurônio da camada l;

- σ é a função de ativação, dada pela equação (2.2).

Pelo momento vamos assumir que a função de ativação é a Função Sigmoide:

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

A derivada da função Sigmoide, necessária para o processo de retropropagação, é:

$$\begin{aligned}
\sigma'(x) &= \frac{d}{dx}\left((1 + e^{-x})^{-1}\right) \quad \text{(regra da cadeia)} \\
&= -1 \cdot (1 + e^{-x})^{-2} \cdot \frac{d}{dx}(1 + e^{-x}) \\
&= -1 \cdot (1 + e^{-x})^{-2} \cdot (-e^{-x}) \\
&= \frac{e^{-x}}{(1 + e^{-x})^2}.
\end{aligned}$$

Observando que

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad \text{e} \quad 1 - \sigma(x) = \frac{e^{-x}}{1 + e^{-x}},$$

segue que

$$\begin{aligned}
\sigma(x)(1 - \sigma(x)) &= \frac{e^{-x}}{(1 + e^{-x})^2} \quad \text{ou} \\
\sigma'(x) &= \sigma(x)(1 - \sigma(x)).
\end{aligned}$$

Para esse exemplo de três camadas, e utilizando o erro quadrático médio (MSE) como função de custo, e tomando um único termo no somatório (3.3) para simplificar, temos:

$$C(W^{[2]}, b^{[2]}, W^{[3]}, b^{[3]}) = \frac{1}{2}(y - a^{[3]})^2 = \frac{1}{2}(y - \sigma(z^{[3]}))^2.$$

A atualização dos parâmetros exige o cálculo das derivadas parciais de C em relação a cada parâmetro. Primeiramente, vamos calcular a derivada da função de custo em relação ao peso $W^{[3]}$. Como a função de custo depende da saída da rede $a^{[3]}$, que por sua vez depende de $z^{[3]}$, e este depende de $W^{[3]}$, temos uma composição de funções. Aplicamos a **regra da cadeia**:

$$\frac{\partial C}{\partial W^{[3]}} = \frac{\partial C}{\partial a^{[3]}} \cdot \frac{\partial a^{[3]}}{\partial z^{[3]}} \cdot \frac{\partial z^{[3]}}{\partial W^{[3]}}$$

Em que:

- $\frac{\partial C}{\partial a^{[3]}} = -(y - a^{[3]})$: derivada da função de custo em relação à saída da rede.
- $\frac{\partial a^{[3]}}{\partial z^{[3]}} = \sigma(z^{[3]})(1 - \sigma(z^{[3]}))$: derivada da função de ativação sigmoide.
- $\frac{\partial z^{[3]}}{\partial W^{[3]}} = a^{[2]}$: derivada da entrada linear da camada 3 com respeito ao peso.

Portanto, a derivada fica:

$$\frac{\partial C}{\partial W^{[3]}} = -(y - a^{[3]}) \cdot \sigma(z^{[3]})(1 - \sigma(z^{[3]})) \cdot a^{[2]}.$$

Como $\frac{\partial C}{\partial a^{[3]}} = -(y - \sigma(z^{[3]}))$, $\frac{\partial \sigma}{\partial z^{[3]}} = \sigma(z^{[3]})(1 - \sigma(z^{[3]}))$ e $\frac{\partial z^{[3]}}{\partial W^{[3]}} = a^{[2]}$, temos que

$$\frac{\partial C}{\partial W^{[3]}} = -(y - \sigma(z^{[3]}))\sigma(z^{[3]})(1 - \sigma(z^{[3]}))a^{[2]}.$$

De forma análoga ao cálculo da derivada em relação ao peso $W^{[3]}$, podemos aplicar a regra da cadeia para obter a derivada da função de custo em relação ao viés $b^{[3]}$:

$$\frac{\partial C}{\partial b^{[3]}} = \frac{\partial C}{\partial a^{[3]}} \cdot \frac{\partial a^{[3]}}{\partial z^{[3]}} \cdot \frac{\partial z^{[3]}}{\partial b^{[3]}}.$$

Sabemos que:

- $\frac{\partial C}{\partial a^{[3]}} = -(y - a^{[3]})$;
- $\frac{\partial a^{[3]}}{\partial z^{[3]}} = \sigma(z^{[3]})(1 - \sigma(z^{[3]}))$;
- $\frac{\partial z^{[3]}}{\partial b^{[3]}} = 1$, pois $z^{[3]} = W^{[3]}a^{[2]} + b^{[3]}$.

Obtemos que:

$$\frac{\partial C}{\partial b^{[3]}} = -(y - a^{[3]}) \cdot \sigma(z^{[3]})(1 - \sigma(z^{[3]})).$$

Com $\frac{\partial C}{\partial W^{[3]}}$ e $\frac{\partial C}{\partial b^{[3]}}$ calculados, podemos realizar a atualização dos parâmetros da camada 3 utilizando o método da descida do gradiente. Isso significa ajustar os valores desses parâmetros na direção oposta ao gradiente da função de custo em relação a eles, visando minimizar o erro. As atualizações são dadas por:

$$W^{[3]} \leftarrow W^{[3]} - \eta \cdot \sigma(z^{[3]})(1 - \sigma(z^{[3]})) \cdot (a^{[3]} - y) \cdot a^{[2]}$$

$$b^{[3]} \leftarrow b^{[3]} - \eta \cdot \sigma(z^{[3]})(1 - \sigma(z^{[3]})) \cdot (a^{[3]} - y),$$

em que η representa a taxa de aprendizado, que controla o tamanho do passo dado em cada atualização.

De modo semelhante, podemos aplicar a regra da cadeia para calcular as derivadas da função de custo em relação aos parâmetros da camada 2. O objetivo é expressar essas derivadas diretamente em função das ativações e derivadas das funções de ativação.

A derivada da função de custo em relação ao peso $W^{[2]}$ é dada por:

$$\frac{\partial C}{\partial W^{[2]}} = \frac{\partial C}{\partial a^{[3]}} \cdot \frac{\partial a^{[3]}}{\partial z^{[3]}} \cdot \frac{\partial z^{[3]}}{\partial a^{[2]}} \cdot \frac{\partial a^{[2]}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial W^{[2]}}.$$

Calculando cada termo:

- $\frac{\partial C}{\partial a^{[3]}} = -(y - a^{[3]})$;
- $\frac{\partial a^{[3]}}{\partial z^{[3]}} = \sigma(z^{[3]})(1 - \sigma(z^{[3]}))$;
- $\frac{\partial z^{[3]}}{\partial a^{[2]}} = W^{[3]}$;
- $\frac{\partial a^{[2]}}{\partial z^{[2]}} = \sigma(z^{[2]})(1 - \sigma(z^{[2]}))$;
- $\frac{\partial z^{[2]}}{\partial W^{[2]}} = a^{[1]}$.

Logo:

$$\frac{\partial C}{\partial W^{[2]}} = -(y - a^{[3]}) \cdot \sigma(z^{[3]})(1 - \sigma(z^{[3]})) \cdot W^{[3]} \cdot \sigma(z^{[2]})(1 - \sigma(z^{[2]})) \cdot a^{[1]}.$$

De forma análoga, a derivada em relação ao viés $b^{[2]}$ é:

$$\frac{\partial C}{\partial b^{[2]}} = -(y - a^{[3]}) \cdot \sigma(z^{[3]})(1 - \sigma(z^{[3]})) \cdot W^{[3]} \cdot \sigma(z^{[2]})(1 - \sigma(z^{[2]})).$$

Com essas expressões, os parâmetros da camada 2 são atualizados da seguinte forma:

$$W^{[2]} \leftarrow W^{[2]} - \eta \cdot \frac{\partial C}{\partial W^{[2]}}, \quad b^{[2]} \leftarrow b^{[2]} - \eta \cdot \frac{\partial C}{\partial b^{[2]}}.$$

Pelo exposto, conseguimos visualizar como os mecanismos de aprendizado funcionam em uma rede neural simples, com um neurônio por camada. Esse modelo, embora elementar, já permite compreender os princípios fundamentais dos ajustes de pesos e vieses por meio da propagação direta e da retropropagação. Esses conceitos formam a base para o desenvolvimento e a compreensão de arquiteturas mais elaboradas de redes neurais.

3.2 REDE NEURAL COM QUATRO CAMADAS

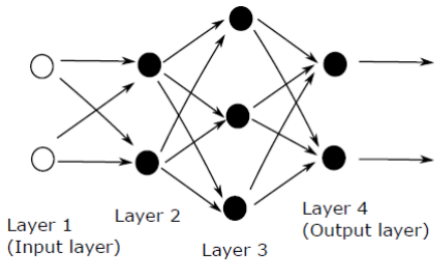
Nesta seção, abordamos o funcionamento de uma rede neural artificial com quatro camadas, cada uma contendo múltiplos neurônios. O objetivo é apresentar com rigor matemático o processo de aprendizado, detalhando a propagação direta e a retropropagação para atualização dos parâmetros.

Considere uma entrada $x \in \mathbb{R}^2$ e uma rede com quatro camadas, onde cada camada l possui pesos W e vieses b.

A entrada $x \in \mathbb{R}^2$ será representada por um vetor coluna:

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}.$$

Figura 3.2: RNA com 4 camadas



Fonte: Kuntze (2019)

A saída da primeira camada oculta é obtida por uma transformação linear seguida de uma função de ativação não linear σ:

$$W^{[1]} \in \mathbb{R}^{2\times 2}, \quad b^{[1]} \in \mathbb{R}^{2\times 1}$$

$$z^{[1]} = W^{[1]}x + b^{[1]}, \quad a^{[1]} = \sigma(z^{[1]}),$$

em que $z^{[1]}$ é o vetor de pré-ativação e $a^{[1]}$ é a saída da camada após aplicar σ elemento a elemento.

De modo detalhado, temos:

$$W^{[1]} = \begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} \\ w_{21}^{[1]} & w_{22}^{[1]} \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad b^{[1]} = \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \end{bmatrix}.$$

De onde segue que,

$$z^{[1]} = W^{[1]}x + b^{[1]} = \begin{bmatrix} w_{11}^{[1]}x_1 + w_{12}^{[1]}x_2 + b_1^{[1]} \\ w_{21}^{[1]}x_1 + w_{22}^{[1]}x_2 + b_2^{[1]}. \end{bmatrix}.$$

Aplicando a função de ativação σ, elemento a elemento, temos:

$$a^{[1]} = \sigma(z^{[1]}) = \begin{bmatrix} \sigma(w_{11}^{[1]}x_1 + w_{12}^{[1]}x_2 + b_1^{[1]}) \\ \sigma(w_{21}^{[1]}x_1 + w_{22}^{[1]}x_2 + b_2^{[1]}) \end{bmatrix} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \end{bmatrix}.$$

De forma análoga, a segunda camada oculta, layer 3, também realiza uma transformação afim seguida por uma ativação:

$$W^{[2]} \in \mathbb{R}^{3\times 2}, \quad b^{[2]} \in \mathbb{R}^{3\times 1}$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}, \quad a^{[2]} = \sigma(z^{[2]}).$$

Como a entrada da segunda camada oculta é o vetor $a^{[1]}$, com dois elementos, a pré-ativação $z^{[2]}$ é obtida por:

$$W^{[2]} = \begin{bmatrix} w_{11}^{[2]} & w_{12}^{[2]} \\ w_{21}^{[2]} & w_{22}^{[2]} \\ w_{31}^{[2]} & w_{32}^{[2]} \end{bmatrix}, \quad a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \end{bmatrix}, \quad b^{[2]} = \begin{bmatrix} b_1^{[2]} \\ b_2^{[2]} \\ b_3^{[2]} \end{bmatrix}.$$

Multiplicando:

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} = \begin{bmatrix} w_{11}^{[2]}a_1^{[1]} + w_{12}^{[2]}a_2^{[1]} + b_1^{[2]} \\ w_{21}^{[2]}a_1^{[1]} + w_{22}^{[2]}a_2^{[1]} + b_2^{[2]} \\ w_{31}^{[2]}a_1^{[1]} + w_{32}^{[2]}a_2^{[1]} + b_3^{[2]} \end{bmatrix}$$

Aplicando a função de ativação σ, elemento a elemento:

$$a^{[2]} = \sigma(z^{[2]}) = \begin{bmatrix} \sigma(w_{11}^{[2]}a_1^{[1]} + w_{12}^{[2]}a_2^{[1]} + b_1^{[2]}) \\ \sigma(w_{21}^{[2]}a_1^{[1]} + w_{22}^{[2]}a_2^{[1]} + b_2^{[2]}) \\ \sigma(w_{31}^{[2]}a_1^{[1]} + w_{32}^{[2]}a_2^{[1]} + b_3^{[2]}) \end{bmatrix} = \begin{bmatrix} a_1^{[2]} \\ a_2^{[2]} \\ a_3^{[2]} \end{bmatrix}.$$

A camada final da rede, layer 4, produz a predição $\hat{y}$, também por meio de uma combinação linear seguida de uma ativação não linear σ. As dimensões dos parâmetros são:

$$W^{[3]} \in \mathbb{R}^{2\times 3}, \quad b^{[3]} \in \mathbb{R}^{2\times 1}, \quad a^{[2]} = \begin{bmatrix} a_1^{[2]} \\ a_2^{[2]} \\ a_3^{[2]} \end{bmatrix}.$$

E $z^{[3]}$ é dado por:

$$z^{[3]} = W^{[3]}a^{[2]} + b^{[3]} = \begin{bmatrix} w_{11}^{[3]}a_1^{[2]} + w_{12}^{[3]}a_2^{[2]} + w_{13}^{[3]}a_3^{[2]} + b_1^{[3]} \\ w_{21}^{[3]}a_1^{[2]} + w_{22}^{[3]}a_2^{[2]} + w_{23}^{[3]}a_3^{[2]} + b_2^{[3]} \end{bmatrix}.$$

Aplicando a função de ativação σ elemento a elemento, obtemos a saída final da rede:

$$a^{[3]} = \sigma(z^{[3]}) = \begin{bmatrix} \sigma(w_{11}^{[3]}a_1^{[2]} + w_{12}^{[3]}a_2^{[2]} + w_{13}^{[3]}a_3^{[2]} + b_1^{[3]}) \\ \sigma(w_{21}^{[3]}a_1^{[2]} + w_{22}^{[3]}a_2^{[2]} + w_{23}^{[3]}a_3^{[2]} + b_2^{[3]}) \end{bmatrix} = \hat{\mathbf{y}} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \end{bmatrix}.$$

A função de custo utilizada, assim como na seção anterior, será o *Erro Quadrático Médio* (MSE), definida como:

$$C = \frac{1}{N}\sum_{i=1}^{N}\left(y(x^{(i)}) - a^{[3]}(x^{(i)})\right)^2$$

em que N é o número de amostras do conjunto de dados, $y(x^{(i)})$ representa a saída esperada para a entrada $x^{(i)}$, e $a^{[3]}(x^{(i)})$ é a saída da rede para essa entrada.

Como a saída $a^{[3]}(x^{(i)})$ é calculada por meio de uma composição de funções, conforme o processo de *feedforward*:

$$\begin{aligned}
z^{[1]} &= w^{[1]}x + b^{[1]} \\
a^{[1]} &= \sigma(z^{[1]}) \\
z^{[2]} &= w^{[2]}a^{[1]} + b^{[2]} \\
a^{[2]} &= \sigma(z^{[2]}) \\
z^{[3]} &= w^{[3]}a^{[2]} + b^{[3]} \\
a^{[3]} &= \sigma(z^{[3]}).
\end{aligned}$$

Portanto, a saída final da rede pode ser escrita como:

$$a^{[3]} = \sigma\left(w^{[3]} \cdot \sigma\left(w^{[2]} \cdot \sigma\left(w^{[1]}x + b^{[1]}\right) + b^{[2]}\right) + b^{[3]}\right).$$

Com a saída definida, é possível aplicar o algoritmo de *backpropagation* para ajustar os pesos $w^{[l]}$ e os vieses $b^{[l]}$, com $l = 1, 2, 3$, para minimizar o custo C. Como já mencionado, esse processo exige o cálculo das derivadas parciais da função de custo em relação a cada parâmetro da rede, para em seguida utilizar o método do gradiente descendente.

As derivadas são calculadas de forma recursiva a partir da última camada (3) até a primeira, utilizando a regra da cadeia. No nosso caso, considerando somente um termo do somatório, queremos derivar a função de custo.

$$C = (y_1 - a_1^{[3]})^2 + (y_2 - a_2^{[3]})^2,$$

em relação aos pesos $w_{ij}^{[3]}$, que conectam os neurônios $a_j^{[2]}$ da segunda camada oculta aos neurônios de saída $a_i^{[3]}$. Sabemos que:

- $a_i^{[3]} = \sigma(z_i^{[3]})$, onde σ é a função de ativação;
- $z_i^{[3]} = \sum_j w_{ij}^{[3]} a_j^{[2]} + b_i^{[3]}$.

Com isso, aplicamos a regra da cadeia para a função de custo em relação a cada peso como:

$$\frac{\partial C}{\partial w_{ij}^{[3]}} = \frac{\partial C}{\partial a_i^{[3]}} \cdot \frac{\partial a_i^{[3]}}{\partial z_i^{[3]}} \cdot \frac{\partial z_i^{[3]}}{\partial w_{ij}^{[3]}}.$$

Cada um desses termos possui um significado específico:

- $\frac{\partial C}{\partial a_i^{[3]}} = -2(y_i - a_i^{[3]})$: representa a taxa de variação da função de custo com relação à saída da rede;
- $\frac{\partial a_i^{[3]}}{\partial z_i^{[3]}} = \sigma'(z_i^{[3]})$: é a derivada da função de ativação aplicada na saída;

- $\frac{\partial z_i^{[3]}}{\partial w_{ij}^{[3]}} = a_j^{[2]}$: é a derivada da entrada linear do neurônio em relação ao peso correspondente.

Substituindo esses termos na regra da cadeia, obtemos as derivadas explícitas de cada peso da terceira camada:

- Para o peso $w_{11}^{[3]}$, que conecta o neurônio $a_1^{[2]}$ ao neurônio de saída $a_1^{[3]}$:

$$\frac{\partial C}{\partial w_{11}^{[3]}} = -2(y_1 - a_1^{[3]}) \cdot \sigma'(z_1^{[3]}) \cdot a_1^{[2]}.$$

- Para o peso $w_{12}^{[3]}$, que conecta $a_2^{[2]}$ a $a_1^{[3]}$:

$$\frac{\partial C}{\partial w_{12}^{[3]}} = -2(y_1 - a_1^{[3]}) \cdot \sigma'(z_1^{[3]}) \cdot a_2^{[2]}.$$

- Para $w_{13}^{[3]}$, que conecta $a_3^{[2]}$ a $a_1^{[3]}$:

$$\frac{\partial C}{\partial w_{13}^{[3]}} = -2(y_1 - a_1^{[3]}) \cdot \sigma'(z_1^{[3]}) \cdot a_3^{[2]}.$$

- Para $w_{21}^{[3]}$, que conecta $a_1^{[2]}$ a $a_2^{[3]}$:

$$\frac{\partial C}{\partial w_{21}^{[3]}} = -2(y_2 - a_2^{[3]}) \cdot \sigma'(z_2^{[3]}) \cdot a_1^{[2]}.$$

- Para $w_{22}^{[3]}$:

$$\frac{\partial C}{\partial w_{22}^{[3]}} = -2(y_2 - a_2^{[3]}) \cdot \sigma'(z_2^{[3]}) \cdot a_2^{[2]}.$$

- Para $w_{23}^{[3]}$:

$$\frac{\partial C}{\partial w_{23}^{[3]}} = -2(y_2 - a_2^{[3]}) \cdot \sigma'(z_2^{[3]}) \cdot a_3^{[2]}.$$

Repetindo o processo para os vieses temos:

$$\frac{\partial C}{\partial b_i^{[3]}} = \frac{\partial C}{\partial a_i^{[3]}} \cdot \frac{\partial a_i^{[3]}}{\partial z_i^{[3]}} \cdot \frac{\partial z_i^{[3]}}{\partial b_i^{[3]}},$$

em que:

- $\frac{\partial C}{\partial a_i^{[3]}} = -2(y_i - a_i^{[3]})$
- $\frac{\partial a_i^{[3]}}{\partial z_i^{[3]}} = \sigma'(z_i^{[3]})$
- $\frac{\partial z_i^{[3]}}{\partial b_i^{[3]}} = 1$

Logo, as derivadas ficam:

- Para $b_1^{[3]}$:

$$\frac{\partial C}{\partial b_1^{[3]}} = -2(y_1 - a_1^{[3]}) \cdot \sigma'(z_1^{[3]}).$$

- Para $b_2^{[3]}$:

$$\frac{\partial C}{\partial b_2^{[3]}} = -2(y_2 - a_2^{[3]}) \cdot \sigma'(z_2^{[3]}).$$

Essas expressões mostram que cada derivada depende da diferença entre a saída da rede e o valor esperado, da derivada da função de ativação aplicada à entrada do neurônio de saída e da ativação do neurônio da camada anterior correspondente. Esse processo é fundamental para que os pesos sejam ajustados corretamente durante o treinamento, por meio do algoritmo de retropropagação.

As derivadas são calculadas de forma recursiva a partir da última camada ($l = 3$) até a primeira, utilizando a regra da cadeia. No que segue, aplicamos esse processo para as camadas intermediárias.

Cada peso $w_{ij}^{[2]}$ conecta o neurônio *j* da *layer 2* ao neurônio *i* da *layer 3*. De forma análoga, cada viés $b_i^{[2]}$ está associado ao neurônio *i* da camada 2. Para os pesos da segunda camada, $w_{ij}^{[2]}$, utilizamos novamente a regra da cadeia.

$$\frac{\partial C}{\partial w_{ij}^{[2]}} = \sum_k \left(\frac{\partial C}{\partial a_k^{[3]}} \cdot \frac{\partial a_k^{[3]}}{\partial z_k^{[3]}} \cdot \frac{\partial z_k^{[3]}}{\partial a_i^{[2]}} \right) \cdot \frac{\partial a_i^{[2]}}{\partial z_i^{[2]}} \cdot \frac{\partial z_i^{[2]}}{\partial w_{ij}^{[2]}}.$$

Cada termo da cadeia corresponde a:

- $\frac{\partial C}{\partial a_k^{[3]}} = -2(y_k - a_k^{[3]})$: taxa de variação do custo em relação à k-ésima saída da rede;
- $\frac{\partial a_k^{[3]}}{\partial z_k^{[3]}} = \sigma'(z_k^{[3]})$: derivada da função de ativação na k-ésima camada de saída;
- $\frac{\partial z_k^{[3]}}{\partial a_i^{[2]}} = w_{ki}^{[3]}$: peso que conecta a ativação $a_i^{[2]}$ ao neurônio de saída k;
- $\frac{\partial a_i^{[2]}}{\partial z_i^{[2]}} = \sigma'(z_i^{[2]})$: derivada da função de ativação da camada intermediária;
- $\frac{\partial z_i^{[2]}}{\partial w_{ij}^{[2]}} = a_j^{[1]}$: ativação da camada anterior.

O índice k utilizado no somatório representa os neurônios da camada de saída. Como cada neurônio da camada intermediária $a_i^{[2]}$ pode influenciar múltiplas saídas $a_k^{[3]}$, é necessário considerar o impacto de todas essas conexões ao calcular a derivada do custo em relação ao peso $w_{ij}^{[2]}$. Por isso, somamos sobre todos os neurônios da camada de saída, acumulando as contribuições parciais de cada caminho possível entre o peso analisado e as saídas da rede.

Para os vieses $b_i^{[2]}$, a derivada segue a mesma lógica, com a diferença de que:

$$\frac{\partial z_i^{[2]}}{\partial b_i^{[2]}} = 1,$$

Assim, encontramos os seguintes valores:

- $\frac{\partial C}{\partial w_{11}^{[2]}} = \left[-2(y_1 - a_1^{[3]}) \cdot \sigma'(z_1^{[3]}) \cdot w_{11}^{[3]} - 2(y_2 - a_2^{[3]}) \cdot \sigma'(z_2^{[3]}) \cdot w_{21}^{[3]}\right] \cdot \sigma'(z_1^{[2]}) \cdot a_1^{[1]}$
- $\frac{\partial C}{\partial w_{12}^{[2]}} = \left[-2(y_1 - a_1^{[3]}) \cdot \sigma'(z_1^{[3]}) \cdot w_{11}^{[3]} - 2(y_2 - a_2^{[3]}) \cdot \sigma'(z_2^{[3]}) \cdot w_{21}^{[3]}\right] \cdot \sigma'(z_1^{[2]}) \cdot a_2^{[1]}$
- $\frac{\partial C}{\partial w_{21}^{[2]}} = \left[-2(y_1 - a_1^{[3]}) \cdot \sigma'(z_1^{[3]}) \cdot w_{12}^{[3]} - 2(y_2 - a_2^{[3]}) \cdot \sigma'(z_2^{[3]}) \cdot w_{22}^{[3]}\right] \cdot \sigma'(z_2^{[2]}) \cdot a_1^{[1]}$
- $\frac{\partial C}{\partial w_{22}^{[2]}} = \left[-2(y_1 - a_1^{[3]}) \cdot \sigma'(z_1^{[3]}) \cdot w_{12}^{[3]} - 2(y_2 - a_2^{[3]}) \cdot \sigma'(z_2^{[3]}) \cdot w_{22}^{[3]}\right] \cdot \sigma'(z_2^{[2]}) \cdot a_2^{[1]}$
- $\frac{\partial C}{\partial w_{31}^{[2]}} = \left[-2(y_1 - a_1^{[3]}) \cdot \sigma'(z_1^{[3]}) \cdot w_{13}^{[3]} - 2(y_2 - a_2^{[3]}) \cdot \sigma'(z_2^{[3]}) \cdot w_{23}^{[3]}\right] \cdot \sigma'(z_3^{[2]}) \cdot a_1^{[1]}$
- $\frac{\partial C}{\partial w_{32}^{[2]}} = \left[-2(y_1 - a_1^{[3]}) \cdot \sigma'(z_1^{[3]}) \cdot w_{13}^{[3]} - 2(y_2 - a_2^{[3]}) \cdot \sigma'(z_2^{[3]}) \cdot w_{23}^{[3]}\right] \cdot \sigma'(z_3^{[2]}) \cdot a_2^{[1]}$
- $\frac{\partial C}{\partial b_1^{[2]}} = \left[-2(y_1 - a_1^{[3]}) \cdot \sigma'(z_1^{[3]}) \cdot w_{11}^{[3]} - 2(y_2 - a_2^{[3]}) \cdot \sigma'(z_2^{[3]}) \cdot w_{21}^{[3]}\right] \cdot \sigma'(z_1^{[2]}) \cdot 1$
- $\frac{\partial C}{\partial b_2^{[2]}} = \left[-2(y_1 - a_1^{[3]}) \cdot \sigma'(z_1^{[3]}) \cdot w_{12}^{[3]} - 2(y_2 - a_2^{[3]}) \cdot \sigma'(z_2^{[3]}) \cdot w_{22}^{[3]}\right] \cdot \sigma'(z_2^{[2]}) \cdot 1$
- $\frac{\partial C}{\partial b_3^{[2]}} = \left[-2(y_1 - a_1^{[3]}) \cdot \sigma'(z_1^{[3]}) \cdot w_{13}^{[3]} - 2(y_2 - a_2^{[3]}) \cdot \sigma'(z_2^{[3]}) \cdot w_{23}^{[3]}\right] \cdot \sigma'(z_3^{[2]}) \cdot 1$

Cada peso $w_{ij}^{[1]}$ conecta o neurônio de entrada x_j ao neurônio $a_i^{[1]}$ da primeira camada oculta. Cada viés $b_i^{[1]}$ está associado ao neurônio $a_i^{[1]}$. Na primeira camada oculta, para os pesos $w_{ij}^{[1]}$, repetimos a regra da cadeia, levando em conta que cada peso influencia todos os caminhos até a saída. Assim:

$$\frac{\partial C}{\partial w_{ij}^{[1]}} = \sum_k \sum_r \left(\frac{\partial C}{\partial a_k^{[3]}} \cdot \frac{\partial a_k^{[3]}}{\partial z_k^{[3]}} \cdot \frac{\partial z_k^{[3]}}{\partial a_r^{[2]}} \cdot \frac{\partial a_r^{[2]}}{\partial z_r^{[2]}} \cdot \frac{\partial z_r^{[2]}}{\partial a_j^{[1]}} \cdot \frac{\partial a_j^{[1]}}{\partial z_j^{[1]}} \cdot \frac{\partial z_j^{[1]}}{\partial w_{ij}^{[1]}} \right).$$

Aqui, o índice r percorre todos os neurônios da camada intermediária, e cada termo corresponde a um caminho possível de retropropagação até o peso analisado. Para os vieses $b_i^{[1]}$, a derivada será idêntica, substituindo $\partial z_i^{[1]} / \partial w_{ij}^{[1]}$ por 1.

$\frac{\partial C}{\partial w_{11}^{[1]}} = \left[-2(y_1 - a_1^{[3]}) \cdot \sigma'(z_1^{[3]}) \cdot w_{11}^{[3]} \cdot \sigma'(z_1^{[2]}) \cdot w_{11}^{[2]} - 2(y_2 - a_2^{[3]}) \cdot \sigma'(z_2^{[3]}) \cdot w_{21}^{[3]} \cdot \sigma'(z_1^{[2]}) \cdot w_{11}^{[2]}\right] \cdot \sigma'(z_1^{[1]}) \cdot x_1;$

$\frac{\partial C}{\partial w_{12}^{[1]}} = \left[-2(y_1 - a_1^{[3]}) \cdot \sigma'(z_1^{[3]}) \cdot w_{11}^{[3]} \cdot \sigma'(z_1^{[2]}) \cdot w_{11}^{[2]} - 2(y_2 - a_2^{[3]}) \cdot \sigma'(z_2^{[3]}) \cdot w_{21}^{[3]} \cdot \sigma'(z_1^{[2]}) \cdot w_{11}^{[2]}\right] \cdot \sigma'(z_1^{[1]}) \cdot x_2;$

$\frac{\partial C}{\partial w_{21}^{[1]}} = \left[-2(y_1 - a_1^{[3]}) \cdot \sigma'(z_1^{[3]}) \cdot w_{12}^{[3]} \cdot \sigma'(z_2^{[2]}) \cdot w_{21}^{[2]} - 2(y_2 - a_2^{[3]}) \cdot \sigma'(z_2^{[3]}) \cdot w_{22}^{[3]} \cdot \sigma'(z_2^{[2]}) \cdot w_{21}^{[2]}\right] \cdot \sigma'(z_2^{[1]}) \cdot x_1;$

$\frac{\partial C}{\partial w_{22}^{[1]}} = \left[-2(y_1 - a_1^{[3]}) \cdot \sigma'(z_1^{[3]}) \cdot w_{12}^{[3]} \cdot \sigma'(z_2^{[2]}) \cdot w_{21}^{[2]} - 2(y_2 - a_2^{[3]}) \cdot \sigma'(z_2^{[3]}) \cdot w_{22}^{[3]} \cdot \sigma'(z_2^{[2]}) \cdot w_{21}^{[2]}\right] \cdot \sigma'(z_2^{[1]}) \cdot x_2;$

$\frac{\partial C}{\partial b_1^{[1]}} = \left[-2(y_1 - a_1^{[3]}) \cdot \sigma'(z_1^{[3]}) \cdot w_{11}^{[3]} \cdot \sigma'(z_1^{[2]}) \cdot w_{11}^{[2]} - 2(y_2 - a_2^{[3]}) \cdot \sigma'(z_2^{[3]}) \cdot w_{21}^{[3]} \cdot \sigma'(z_1^{[2]}) \cdot w_{11}^{[2]}\right] \cdot \sigma'(z_1^{[1]});$

$$\frac{\partial C}{\partial b_2^{[1]}} = \left[-2(y_1 - a_1^{[3]}) \cdot \sigma'(z_1^{[3]}) \cdot w_{12}^{[3]} \cdot \sigma'(z_2^{[2]}) \cdot w_{21}^{[2]} - 2(y_2 - a_2^{[3]}) \cdot \sigma'(z_2^{[3]}) \cdot w_{22}^{[3]} \cdot \sigma'(z_2^{[2]}) \cdot w_{21}^{[2]}\right] \cdot \sigma'(z_2^{[1]}).$$

Com essas expressões, os parâmetros de cada camada são atualizados utilizando o método do gradiente descendente. Para uma taxa de aprendizado η, as atualizações seguem as fórmulas:

$$W^{[3]} \leftarrow W^{[3]} - \eta \cdot \frac{\partial C}{\partial W^{[3]}}, \quad b^{[3]} \leftarrow b^{[3]} - \eta \cdot \frac{\partial C}{\partial b^{[3]}};$$

$$W^{[2]} \leftarrow W^{[2]} - \eta \cdot \frac{\partial C}{\partial W^{[2]}}, \quad b^{[2]} \leftarrow b^{[2]} - \eta \cdot \frac{\partial C}{\partial b^{[2]}};$$

$$W^{[1]} \leftarrow W^{[1]} - \eta \cdot \frac{\partial C}{\partial W^{[1]}}, \quad b^{[1]} \leftarrow b^{[1]} - \eta \cdot \frac{\partial C}{\partial b^{[1]}}.$$

Essas atualizações fazem com que cada peso e viés seja ajustado na direção de maior redução do erro, com base nos gradientes calculados pelas regras da retropropagação. Com todos os gradientes calculados, o algoritmo de retropropagação fornece uma maneira sistemática de ajustar cada peso e viés da rede de forma eficiente. Através da aplicação iterativa dessas atualizações, a rede neural é treinada para reduzir o erro entre as saídas previstas e os valores esperados, permitindo que ela aprenda padrões a partir dos dados.

3.3 REDE NEURAL COM MÚLTIPLOS NEURÔNIOS E CAMADAS

Dando continuidade às seções anteriores, nesta etapa generalizamos a estrutura da rede neural para o caso com múltiplos neurônios por camada e um número arbitrário de camadas. Este modelo permite aplicações práticas de aprendizado profundo, e os conceitos fundamentais apresentados anteriormente permanecem válidos.

Considere $L \in \mathbb{N}$ como o número total de camadas da nossa Rede Neural. Para cada camada $\ell \in \{1, 2, \ldots, L\}$, n_l denota o número de neurônios da camada l. Definimos:

- $W^{[\ell]} \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$: matriz de pesos,
- $b^{[\ell]} \in \mathbb{R}^{n_\ell}$: vetor de vieses,
- $z^{[\ell]} \in \mathbb{R}^{n_\ell}$: entrada linear da camada,
- $a^{[\ell]} \in \mathbb{R}^{n_\ell}$: ativação da camada,
- $\sigma^{[\ell]} : \mathbb{R} \to \mathbb{R}$: função de ativação aplicada componente a componente.

A entrada da rede é o vetor $x = a^{[0]}$, e a saída da camada ℓ é definida por:

$$a^{[\ell]} = \sigma^{[\ell]}(z^{[\ell]}), \qquad z^{[\ell]} = W^{[\ell]} a^{[\ell-1]} + b^{[\ell]}.$$

A predição final da rede é dada por $a^{[L]} = \hat{y}$. A composição da rede pode ser representada como:

$$\hat{y} = f(x; \theta) = a^{[L]} = \sigma^{[L]} \circ \varphi^{[L]}(z^{[L]}) \circ \cdots \circ \sigma^{[1]} \circ \varphi^{[1]}(x),$$

em que $\varphi^{[\ell]}(x) = W^{[\ell]}x + b^{[\ell]}$, e $\theta = \{W^{[\ell]}, b^{[\ell]}\}_{\ell=1}^{L}$ representam o conjunto de parâmetros treináveis.

Para realizar o treinamento da rede, utilizamos uma função de custo C que mede a diferença entre a predição $\hat{y}$ e a saída esperada y. Um exemplo comum é o erro quadrático médio (considerando

um único termo, ou teste).

$$C(a^{[L]}, y) = \|a^{[L]} - y\|^2$$

Nosso objetivo é encontrar os parâmetros θ que minimizam essa função de custo em um conjunto de treinamento.

Para minimizar C, utilizamos o método de gradiente descendente, calculando as derivadas parciais de C em relação a todos os seus parâmetros $W^{[\ell]}$ e $b^{[\ell]}$. Esse cálculo é feito por meio do algoritmo de *(*backpropagation).

Definimos o vetor de erro da camada ℓ como:

$$\delta^{[\ell]} = \frac{\partial C}{\partial z^{[\ell]}} \in \mathbb{R}^{n_\ell}.$$

Esse vetor é calculado recursivamente:

- Camada de saída ($\ell = L$):
$$\delta^{[L]}(z^{[L]}) = \sigma'^{[L]}(z^{[L]}) \circ (a^{[L]} - y).$$
- Camadas intermediárias ($\ell = L-1, \ldots, 1$):
$$\delta^{[\ell]}(z^{[\ell]}) = \sigma'^{[\ell]}(z^{[\ell]}) \circ \left((W^{[\ell+1]})^\top \delta^{[\ell+1]}(z^{[\ell+1]}) \right).$$

Nessas expressões, o símbolo $\circ$ representa o produto de Hadamard (multiplicação, elemento a elemento), e $\sigma'^{[\ell]}(z^{[\ell]})$ é o vetor das derivadas da função de ativação.

Os gradientes da função de custo em relação aos pesos e vieses são dados por:

$$\frac{\partial C}{\partial W^{[\ell]}} = \delta^{[\ell]}(z^{[\ell]}) \cdot (a^{[\ell-1]})^\top, \quad \frac{\partial C}{\partial b^{[\ell]}} = \delta^{[\ell]}(z^{[\ell]}).$$

Utilizando a regra da descida do gradiente, os parâmetros são atualizados por:

$$W^{[\ell]} \leftarrow W^{[\ell]} - \eta \cdot \delta^{[\ell]}(z^{[\ell]}) \cdot (a^{[\ell-1]})^\top$$

$$b^{[\ell]} \leftarrow b^{[\ell]} - \eta \cdot \delta^{[\ell]}(z^{[\ell]}),$$

em que $\eta > 0$ é a taxa de aprendizado.

Essas etapas são repetidas iterativamente sobre o conjunto de testes até que a função de custo atinja um valor mínimo satisfatório. Esse procedimento permite que a rede neural aprenda representações adequadas para realizar tarefas como classificação ou regressão com precisão.

4. APRENDIZAGEM PROFUNDA EM PYTHON COM MNIST

Atualmente, ao se pensar em Inteligência Artificial, a principal linguagem de programação utilizada é o *Python*. Sua diagramação é simples e organizada, o que facilita a produção e desenvolvimento de algoritmos sofisticados para a indústria e para o meio acadêmico. Além disso, essa linguagem de programação oferece vários conjuntos de bibliotecas já pensadas para o aprendizado de máquinas, como o *Numpy*, *TensorFlow* e *Keras*, que serão abordados neste trabalho. A ideia neste capítulo é utilizar a programação para treinar uma RN capaz de reconhecer dígitos reais de 0 a 9, para isso, foi utilizado como base de treinamento o conjunto de dados MNIST (*Modified National Institute of Standards and Technology*).

Para Jocher, Derrenger e Noyce (2024), o conjunto de dados MNIST é uma grande base de dados de dígitos manuscritos normalmente utilizada para treinar vários sistemas de processamento de imagem e modelos de aprendizagem automática. Esse banco conta com cerca de 60 mil imagens para treinamento e mais de 10 mil imagens de dígitos manuscritos. As imagens são mostradas em escala de cinza e contam com a dimensão de 28 pixels de altura por 28 pixels de largura. Os caracteres armazenados neste banco de dados seguem o padrão abaixo:

Figura 4.1: Caracteres MNIST

Fonte: Jocher, Derrenger e Noyce (2024)

É possível perceber um padrão na escolha e formatação dos números presentes no banco de dados. Todos os algarismos apresentam linhas grossas na cor preta e estão dispostos em um fundo branco, sem a presença de luz e sombra nas fotos. Por isso, é importante que quando apresentamos imagens reais, elas sigam esse padrão. Caso sejam utilizadas imagens com outras cores, ou com grande influência de luz e sombra, o modelo não será efetivo, pois sua base de treinamento não apresentava essas informações, o que pode comprometer os resultados.

4.1 PYTHON E MNIST

Conforme já citado, vamos iniciar aqui a implementação prática de uma Rede Neural Convolucional (CNN - Seção 2.4.3) para reconhecimento de dígitos manuscritos, com base na abordagem anteriormente trabalhada para compreensão de seu treinamento. O código foi desenvolvido em *Python* com o auxílio da biblioteca *TensorFlow* para facilitar a construção de modelos de aprendizado profundo.

O código completo pode ser encontrado no Apêndice 1 deste trabalho, abaixo são comentadas as partes essencias e que fazem relação com o método de aprendizagem de máquinas abordado.

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

A estrutura utilizada é do tipo Convolucional (Seção 2.4.3), composta por camadas que extraem características da imagem e, após isso, as camadas conectadas são responsáveis pela classificação final. Esse tipo de rede é amplamente utilizado em tarefas de reconhecimento de imagens, como a identificação de dígitos manuscritos, daí a escolha por essa estrutura. As camadas das redes convolucionais seguem o mesmo princípio de treinamento: a retropropagação do erro (*backpropagation*), que calcula os ajustes dos parâmetros da rede com base no erro cometido na saída, conforme aprofundado no Capítulo 3.

Além da estrutura convolucional da rede, as funções de ativação utilizadas desempenham um papel essencial no funcionamento da nossa Rede Neural, conforme visto na Seção 2.2. No código implementado, é utilizada a função ReLU (2.2.3) nas camadas intermediárias e a função *Softmax* (2.2.4) na camada de saída.

A função ReLU (2.2.3) é usada pois permite à RN comportamentos não lineares, possibilitando que ela consiga atingir resultados mais complexos. Já a função Softmax é usada na camada de saída para converter os resultados da rede em probabilidades associadas a cada classe. Essa ativação é essencial para problemas de classificação multiclasse como os representados pelo MNIST. Desta forma, tanto ReLU quanto Softmax são escolhas práticas e também matematicamente justificadas dentro do processo de *backpropagation*.

```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Nestas linhas de código, define-se a função de custo como a *Cross-Entropy*, conforme discutido na Seção 2.3.4, por ser a mais adequada para classificação multiclasse. A métrica *accuracy* é utilizada para acompanhar a evolução do desempenho da rede ao longo do treinamneto, medindo sua capacidade de classificar corretamente os dígitos de 0 a 9 no MNIST.

Embora o processo de treinamento ainda não tenha sido discutido em capítulos anteriores, ele é essencial para que a rede neural aprenda com os dados. A seguir, será apresentado como esse processo ocorre na prática.

```
model.fit(datagen.flow(x_train, y_train_cat, batch_size=64),
          epochs=10,
          validation_data=(x_test, y_test_cat))
```

O treinamento da rede acontece com o uso do método *fit()*, que é o comando responsável por fazer a rede "aprender" a partir dos dados. Esse processo ocorre ao longo de 10 rodadas (chamadas de épocas), em que a rede ajusta seus parâmetros com base nos erros que comete ao tentar prever os resultados. Para tornar o aprendizado mais eficiente, utiliza-se o `datagen.flow()`, que gera variações das imagens originais. Isso ajuda a rede a reconhecer padrões mesmo quando há alterações nas imagens, tornando-a mais robusta e preparada para lidar com diferentes situações, que é o caso quando lidamos com dígitos reais, afinal, dependendo da pessoa escrevendo ou do material utilizado para escrever, o resultado do algarismo será diferente.

4.2 TESTE COM IMAGENS REAIS

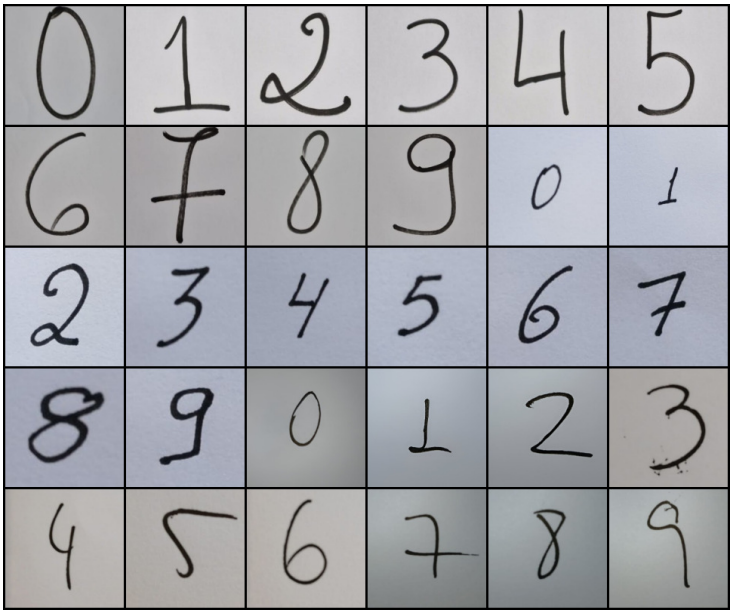
A etapa de testes com dígitos escritos por pessoas reais é fundamental para a validação do código, afinal, se ele não servir para identificar algarismos significa que o modelo não foi suficientemente treinado, ou seja, ele é modelo *underffited*, conforme abordado na Seção (2.4.1). O objetivo com a execução do código é que o modelo seja capaz de reconhecer a maioria dos algarismos, identificando corretamente os números escritos fora do banco de dados MNIST.

Os dígitos utilizados para os testes foram escritos à mão pelos Professores Paulo Bösing, orientador do autor deste trabalho, Antonio Marcos Correa Neri e Rosane Rossato Binotto, que são os professores que compõem a banca avaliadora do trabalho. Foram utilizadas folhas de papel branco comum e caneta preta para a escrita, além disso, as imagens foram capturadas por meio de fotografia utilizando os celulares dos respectivos professores. Esse tipo de captura apresenta variação na espessura dos dígitos, o que é natural quando a escrita ocorre à mão livre, presença de luz, sombra e variação na iluminação, comum quando se é fotografado um papel. Por conta dessas características naturais da fotografia, os testes foram realizados em duas etapas.

A primeira, com as imagens da forma que foram enviadas, sem passar por nenhum pré-processamento, ou seja, sem fazer alterações de brilho, sombra ou contraste. A segunda etapa contou com o pré-processamento das imagens, alterando suas características para deixá-las mais próximas das imagens encontradas no banco de dados MNIST, com o fundo claro e traços pretos bem definidos. Essa comparação entre imagens com e sem pré-processamento é importante porque permite avaliar a robustez do modelo frente a diferentes condições visuais e de iluminação. As imagens capturadas sem tratamento refletem situações reais de uso, com variações naturais como sombras e espessuras dos traços, já as imagens pré-processadas se aproximam das condições ideais do banco MNIST, utilizado para o treinamento do modelo.

Os testes foram realizados utilizando a plataforma Google Colab, permitindo a execução de código *Python* com suporte à biblioteca *TensorFlow* em nuvem. O código foi escrito e rodado em um notebook Samsung Galaxy Book 2, equipado com processador Intel Core i7 de 12ª geração, 8 GB de memória RAM e sistema operacional Windows 11. Nas Figuras 4.2 e 4.3 são apresentados os dígitos reais, sem, e com tratamento, respectivamente, escritos pelos professores envolvidos no trabalho.

Figura 4.2: Dígitos sem pré-processamento



Fonte: Acervo pessoal (2025)

4.2.1 Testes com Imagens Reais

Abaixo estão apresentados os resultados da classificação dos dígitos manuscritos. Na Tabela 4.1 para cada número de 0 a 9, há três linhas: a primeira escrita pelo Professor Antônio, a seguinte pelo Professor Paulo e última pela Professora Rosane. As imagens foram utilizadas em seu formato original, sem aplicação de pré-processamento, ou seja, sem tratamento de imagem. Essa organização permite analisar o desempenho do modelo frente às variações individuais de escrita e às imperfeições típicas de capturas fotográficas. O detalhamento completo dos resultados pode ser consultado no Apêndice B.

Tabela 4.1: Resultados dos testes de reconhecimento - Sem pré-processamento

Dígito Esperado	Saída do Modelo
0	3
0	9
0	0
1	1
1	1
1	1
2	2
2	2
2	2
3	3
3	3
3	3

Continua na próxima página

Tabela 4.1 – continuação

Dígito Esperado	Saída do Modelo
4	4
4	4
4	4
5	5
5	9
5	5
6	6
6	6
6	6
7	7
7	3
7	4
8	7
8	8
8	8
9	3
9	9
9	5

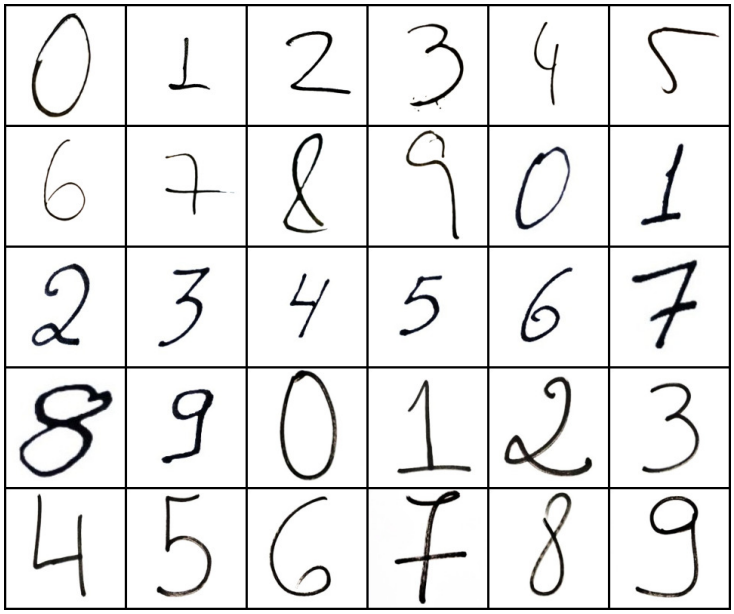
A Tabela 4.1 apresenta os resultados da classificação automática de dígitos manuscritos, utilizando imagens capturadas em condições reais e sem qualquer tipo de pré-processamento. No total, foram analisadas 30 imagens, distribuídas igualmente entre os dígitos de 0 a 9. Dentre essas, o modelo acertou 22 classificações e cometeu 8 erros, o que corresponde a uma taxa de acerto de aproximadamente 73,33%.

É importante destacar que as imagens utilizadas neste teste apresentam características visuais significativamente distintas daquelas presentes no conjunto de dados MNIST. No banco de dados os números aparecem com traços nítidos e fundo padronizado branco, enquanto as imagens reais analisadas aqui possuem variações na espessura dos traços, presença de sombras e problemas como sombras e iluminação desigual. Essas variáveis podem influenciar negativamente o desempenho do modelo, uma vez que ele foi treinado com dados idealizados e pode ter dificuldade em generalizar para condições não vistas durante o treinamento.

4.2.2 Testes com Imagens Reais Com Pré-Processamento

Nesta etapa, foram realizados novos testes com as mesmas imagens manuscritas dos professores, mas agora com aplicação de pré-processamento para ajustar brilho, contraste e remover sombras. O objetivo é aproximar visualmente as imagens do padrão do MNIST. Para cada dígito de 0 a 9, há três linhas: uma do Professor Antônio, uma do Professor Paulo e uma da Professora Rosane, com a mesma apresentação discutida anteriormente. Essa padronização busca verificar se a semelhança com os dados de treinamento melhora o desempenho do modelo. O detalhamento completo dos resultados pode ser consultado no Apêndice C.

Figura 4.3: Dígitos com pré-processamento



Fonte: Acervo pessoal (2025)

Tabela 4.2: Resultados dos testes de reconhecimento - Com pré-processamento

Dígito Esperado	Saída do Modelo
0	0
0	0
0	0
1	1
1	1
1	1
2	2
2	2
2	2
3	3
3	3
3	3
4	4
4	4
4	4
5	5
5	9
5	5
6	6
6	6
6	6
7	7
7	7
7	4

Continua na próxima página

Tabela 4.2 – continuação

Dígito Esperado	Saída do Modelo
8	8
8	8
8	8
9	7
9	9
9	5

A Tabela 4.2 apresenta os resultados da classificação de dígitos manuscritos com aplicação de tratamento nas imagens. O modelo acertou 26 e cometeu 4 erros, representando uma taxa de acerto de 86,67%. Diferentemente do teste anterior, nesta etapa foram aplicadas técnicas de ajuste de brilho, contraste e remoção de sombras, com o objetivo de aproximar as imagens reais do padrão empregado no MNIST, que foi utilizado para o treinamento do modelo pois, ao tornar as imagens mais semelhantes às de treinamento, o teste busca verificar se essa compatibilização visual contribui para a melhoria no desempenho do código.

4.2.3 Comparação dos Resultados

Comparando o desempenho de ambos os modelos fica bastante evidente a importância do treinamento para as Redes Neurais Artificiais e como ele influencia no resultado final encontrado. A taxa de acerto quando os testes ocorreram com dígitos que não passaram por um tratamento visual para se assemelhar às imagens do MNIST foi consideravelmente menor que no segundo teste, uma diferença de 13,34% na taxa de acerto, o que evidencia a importância de múltiplos cenários para um treinamento eficiente e capaz de gerar maior capacidade de generalização.

A diferença observada nos dois cenários é consequência direta da otimização da função de custo por meio do algoritmo de *backpropagation* e da descida de gradiente. Durante a fase em que o modelo está aprendendo, o treinamento, os pesos e viéses da rede são ajustados com base no gradiente de erro, dependendo diretamente dos dados apresentados. Nas imagens que passaram por tratamento, ficando visualmente mais semelhantes ao MNIST, a rede consegue classificar as características com mais facilidade, aumentando a taxa de acerto.

Contudo, quando a imagem não passa por esse tratamento, além dos números, elas apresentam ruídos visuais, problemas de luz e sombra além de tons com menos contraste, o que dificulta o reconhecimento de padrões, pois o código não está preparado para essa apresentação dos dados. Isso evidencia que a rede não fica limitada ao reconhecimento apenas dos padrões treinados, porém apresenta um desempenho inferior quando passa por esse cenário.

De modo geral, esse capítulo apresentou uma implementação prática de uma Rede Neural e também evidenciou o impacto do treinamento na eficiência do modelo em aplicações reais. O tratamento das fotos foi essencial para aproximá-las do padrão original do MNIST, permitindo ao modelo aplicar de forma mais precisa os padrões aprendidos. Isso reforça a importância da seleção do conjunto de dados de treinamento para ajustar a precisão do modelo, pois ele pode ser responsável pelo desempenho não adequado do modelo.

5. CONSIDERAÇÕES FINAIS

Este trabalho teve como objetivo mostrar e compreender, mediante um viés matemático e aplicado, o que estrutura o modelo de Aprendizagem Profunda, o qual serve como base para as Redes Neurais Artificiais. Para isso, foram apresentados conceitos fundamentais como a existência do Perceptron e das funções de ativação, redes neurais multicamadas e convolucionais. Outros temas abordados foram os das funções de custo, retropropagação, regularização e o método da descida gradiente, como mecanismos para dar mais precisão e confiabilidade ao treinamento das Redes.

Além disso, foi utilizado o *Python* como linguagem de programação para aplicar os conceitos vistos por meio de um algoritmo que utiliza o conjunto de dados MNIST para treinamento. Dessa forma, foi possível perceber a relação íntima entre teoria e prática, unindo a abordagem matemática ao desenvolvimento computacional. Os testes realizados com imagens reais mostraram resultados bastante positivos no reconhecimento dos números, o que mostra a eficácia do modelo implementado, além da validação dos conceitos estudados.

Este trabalho buscou desmitificar o funcionamento das RN que vêm crescendo exponencialmente no mundo contemporâneo. Apesar da sua aparente complexidade, elas apresentam um aporte matemático que pode ser considerado simples, e que pode ser aprofundado dentro dos estudos da matemática. Espera-se que com este trabalho mais estudantes sintam-se estimulados a conhecer e explorar a área de Redes Neurais, percebendo que a matemática, que muitas vezes é abstrata, é uma poderosa ferramenta para a criação e expansão da tecnologia.

REFERÊNCIAS

ALAKE, Richmond. **Explicação das Funções de Perda no Aprendizado de Máquina**. 2020. Disponível em: https://www.datacamp.com/pt/tutorial/loss-function-in-machine-learning. Acesso em: 5 set. 2024.

BRANCO, Henrique. **Overfitting e underfitting em Machine Learning**. 2020. Disponível em: https://abracd.org/2020/08/21/overfitting-e-underfitting-em-machine-learning/. Acesso em: 14 set. 2024.

DAMADI, Saeed; MOHARRER, Golnaz; CHAM, Mostafa. **The Backpropagation Algorithm for a Math Student**. 2023. Disponível em: https://arxiv.org/abs/2301.09977. Acesso em: 17 jun. 2025.

DONG, Hao; DING, Zihan; ZHANG, Shanghang. **Deep Reinforcement Learning: Fundamentals, Research and Applications**. Cham, Switzerland: Springer Nature, jun. 2020.

FILHO, Mario. **As Métricas Mais Populares para Avaliar Modelos de Machine Learning**. 2018. Disponível em: https://mariofilho.com/as-metricas-mais-populares-para-avaliar-modelos-de-machine-learning/. Acesso em: 8 set. 2024.

_____. **MAE (Erro Médio Absoluto) em Machine Learning**. 2023. Disponível em: https://mariofilho.com/mae-erro-medio-absoluto-em-machine-learning/. Acesso em: 8 set. 2024.

GHARAT, Snehal. **What, Why and Which? Activation Functions**. [S.l.: s.n.], 2019. Disponível em: https://medium.com/@snaily16/what-why-and-which-activation-functions-b2bf748c0441.

GLOROT, Xavier; BORDES, Antoine; BENGIO, Yoshua. Deep Sparse Rectifier Neural Networks. In: PROCEEDINGS of the Fourteenth International Conference on Artificial Intelligence and Statistics. Fort Lauderdale, FL, USA: PMLR, 2011. P. 315–323. Disponível em: https://proceedings.mlr.press/v15/glorot11a.html. Acesso em: 4 set. 2024.

GUERRERO PEÑA, Fidel Alejandro. **Loss Function Modeling for Deep Neural Networks Applied to Pixel-Level Tasks**. 2019. Tese (Doutorado) – Universidade Federal de Pernambuco, Recife. Disponível em: https://repositorio.ufpe.br/handle/123456789/36676. Acesso em: 5 set. 2024.

HE, Kaiming *et al.* **Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification**. 2015. Disponível em: https://arxiv.org/abs/1502.01852. Acesso em: 4 set. 2024.

HINTON, Geoffrey E. *et al.* **Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors**. 2012. Disponível em: https://arxiv.org/abs/1207.0580. Acesso em: 16 set. 2024.

HUBEL, David H.; WIESEL, Torsten N. Receptive Fields, Binocular Interaction and Functional Architecture in the Cat's Visual Cortex. **The Journal of Physiology**, Wiley, v. 160, n. 1, p. 106–154, 1962. Disponível em: https://pmc.ncbi.nlm.nih.gov/articles/PMC1359523/. Acesso em: 21 set. 2024.

JOCHER, Glenn; DERRENGER, Paula; NOYCE, Matthew. **Conjunto de Dados MNIST**. 2024. Disponível em: https://docs.ultralytics.com/pt/datasets/classify/mnist/#can-i-use-ultralytics-hub-to-train-models-on-custom-datasets-like-mnist. Acesso em: 17 jun. 2025.

KUNTZE, Laura. **Mathematical Steps for a Deep Learning Approach**. 2019. Disponível em: https://pure.tue.nl/ws/portalfiles/portal/128003982/BEPTWDeepLearningLauraKuntzeFinal_niet_vertrouwelijk_.pdf. Acesso em: 16 jan. 2025.

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. Disponível em: http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf.

LUCAS, Hudson Barroso. **Regularização em Machine Learning: Técnicas e Benefícios para Modelos Mais Robustos**. 2024. Disponível em: https://iacomcafe.com.br/regularizacao-machine-learning-l1-l2-dropout-early-stopping-batch-normalization/. Acesso em: 14 set. 2024.

ROSENBLATT, Frank. The perceptron: a probabilistic model for information storage and organization in the brain. **Psychological review**, American Psychological Association, v. 65, n. 6, p. 386, 1958. Disponível em: https://www.ling.upenn.edu/courses/cogs501/Rosenblatt1958.pdf.

RUMELHART, David E.; HINTON, Geoffrey E.; WILLIAMS, Ronald J. Learning Representations by Back-Propagating Errors. **Nature**, v. 323, p. 533–536, 1986. Disponível em: https://api.semanticscholar.org/CorpusID:205001834. Acesso em: 30 set. 2024.

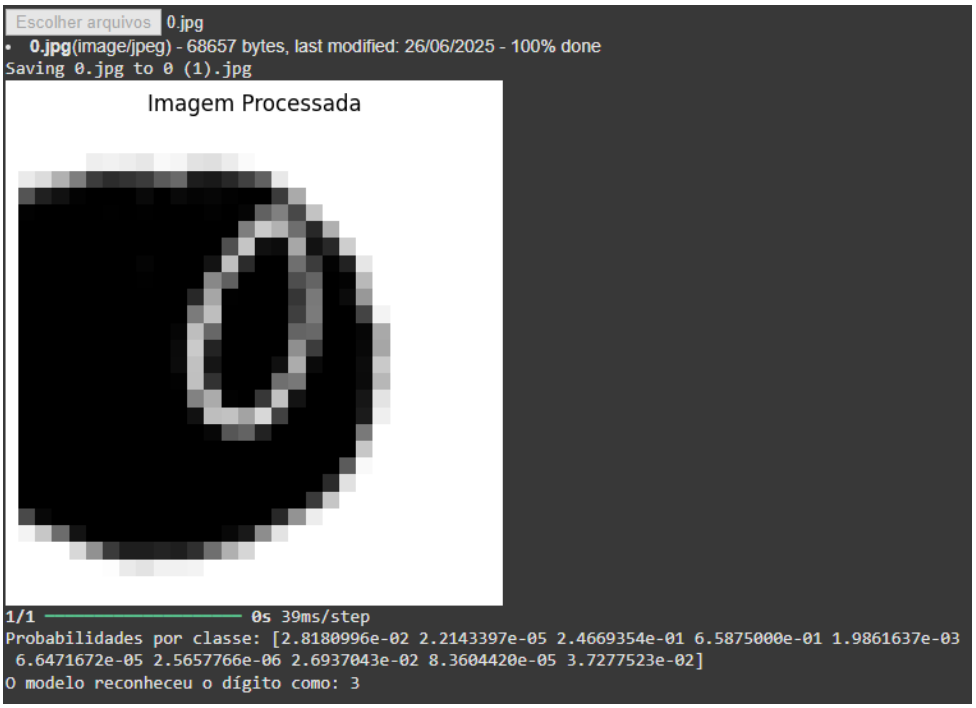
XU, Bing *et al.* **Empirical Evaluation of Rectified Activations in Convolutional Network**. 2015. Disponível em: https://arxiv.org/abs/1505.00853. Acesso em: 4 set. 2024.

APÊNDICE A – CÓDIGO *PYTHON* UTILIZADO

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.preprocessing.image import ImageDataGenerator

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
x_train = x_train.reshape(-1, 28, 28, 1)
x_test  = x_test.reshape(-1, 28, 28, 1)

y_train_cat = to_categorical(y_train, 10)
y_test_cat = to_categorical(y_test, 10)

datagen = ImageDataGenerator(
    rotation_range=10,
    zoom_range=0.1,
    width_shift_range=0.1,
    height_shift_range=0.1
)
datagen.fit(x_train)

model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
    MaxPooling2D((2,2)),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D((2,2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(datagen.flow(x_train, y_train_cat, batch_size=64),
          epochs=10, validation_data=(x_test, y_test_cat))

from google.colab import files
from PIL import Image, ImageOps, ImageFilter
import numpy as np
import matplotlib.pyplot as plt
```

```
uploaded = files.upload()
img_path = list(uploaded.keys())[0]

img = Image.open(img_path).convert("L")
img = ImageOps.invert(img)
img = img.point(lambda x: 0 if x < 100 else 255)
img = img.filter(ImageFilter.MaxFilter(3))

img_array = np.array(img)
coords = np.column_stack(np.where(img_array > 0))
if coords.any():
    y0, x0 = coords.min(axis=0)
    y1, x1 = coords.max(axis=0)
    digit = img.crop((x0, y0, x1, y1)).resize((20, 20), Image.LANCZOS)
else:
    digit = img.resize((20, 20), Image.LANCZOS)

canvas = Image.new("L", (28, 28))
canvas.paste(digit, (4, 4))

img_array = np.array(canvas) / 255.0
img_array = img_array.reshape(1, 28, 28, 1)

plt.imshow(img_array[0].reshape(28, 28), cmap="gray")
plt.title("Imagem centralizada")
plt.axis("off")
plt.show()

prediction = model.predict(img_array)
print("Probabilidades:", prediction[0])
print("Classe prevista:", np.argmax(prediction))
```

APÊNDICE B – RESULTADOS DE CLASSIFICAÇÃO - SEM PRÉ-PROCESSAMENTO



Resultado do teste com o dígito 0 sem pré-processamento - Antônio

Resultado do teste com o dígito 1 sem pré-processamento - Antônio

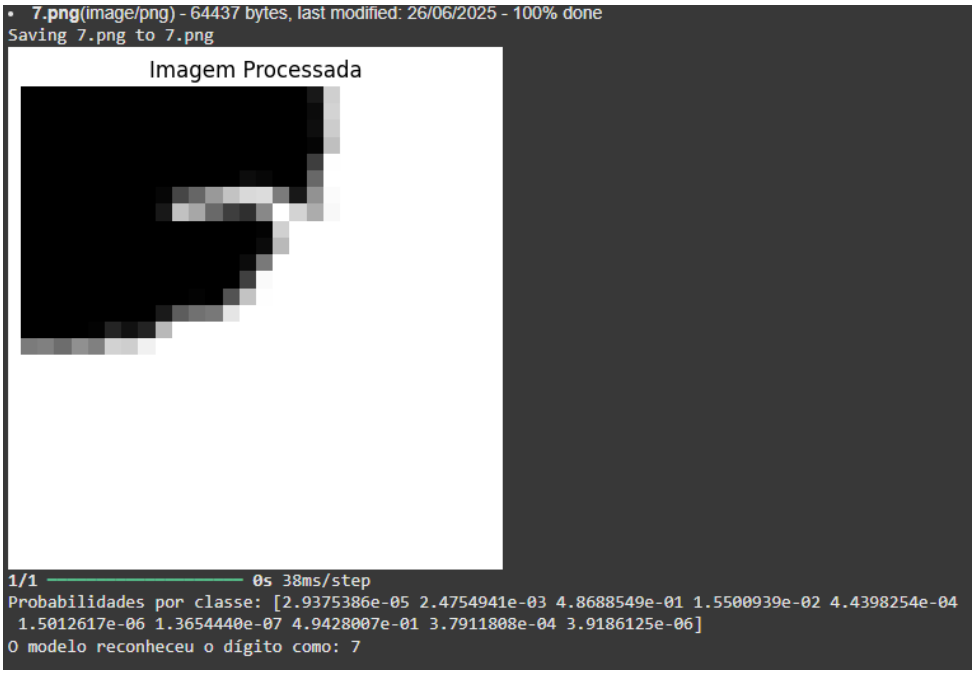
Resultado do teste com o dígito 2 sem pré-processamento - Antônio

Resultado do teste com o dígito 3 sem pré-processamento - Antônio

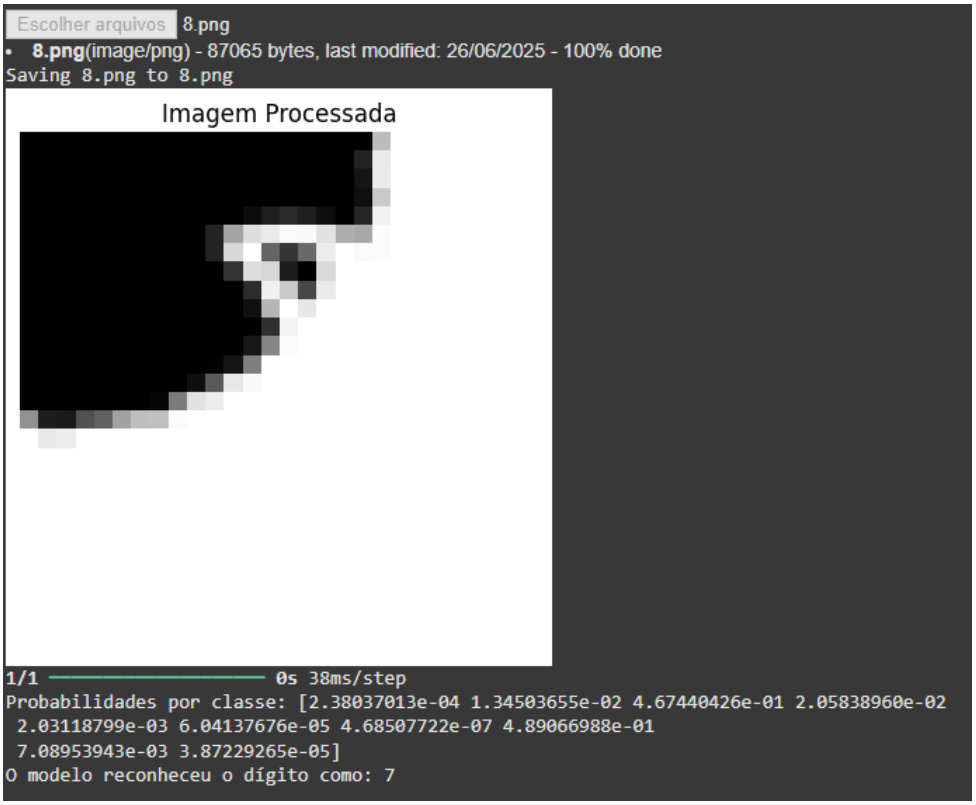
Resultado do teste com o dígito 4 sem pré-processamento - Antônio

Resultado do teste com o dígito 5 sem pré-processamento - Antônio

Resultado do teste com o dígito 6 sem pré-processamento - Antônio

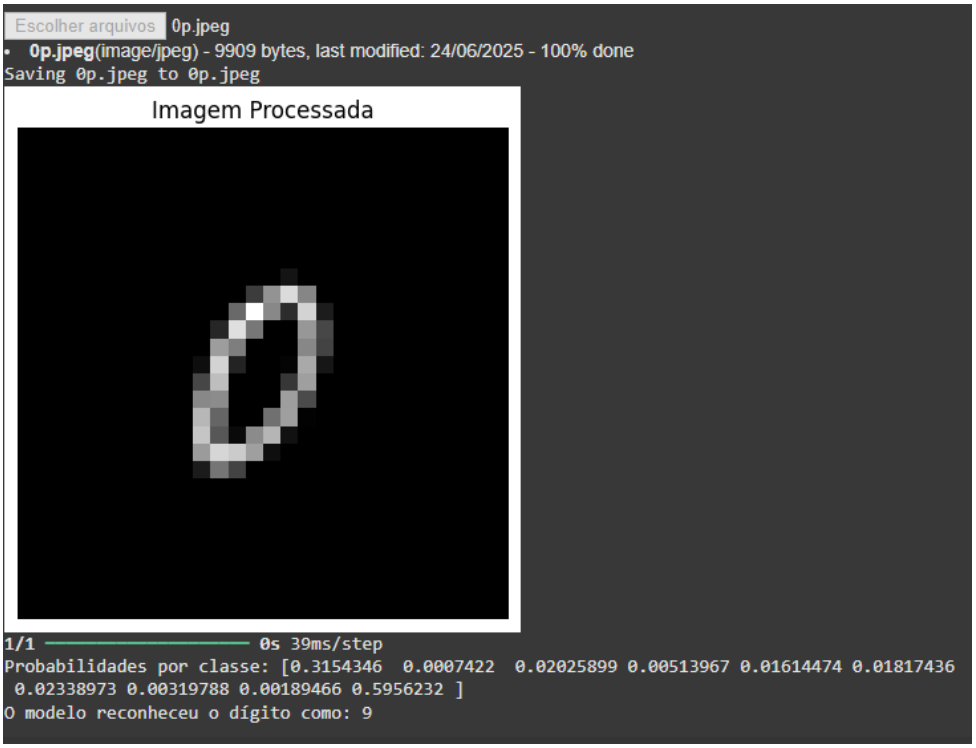


Resultado do teste com o dígito 7 sem pré-processamento - Antônio



Resultado do teste com o dígito 8 sem pré-processamento - Antônio

Resultado do teste com o dígito 9 sem pré-processamento - Antônio

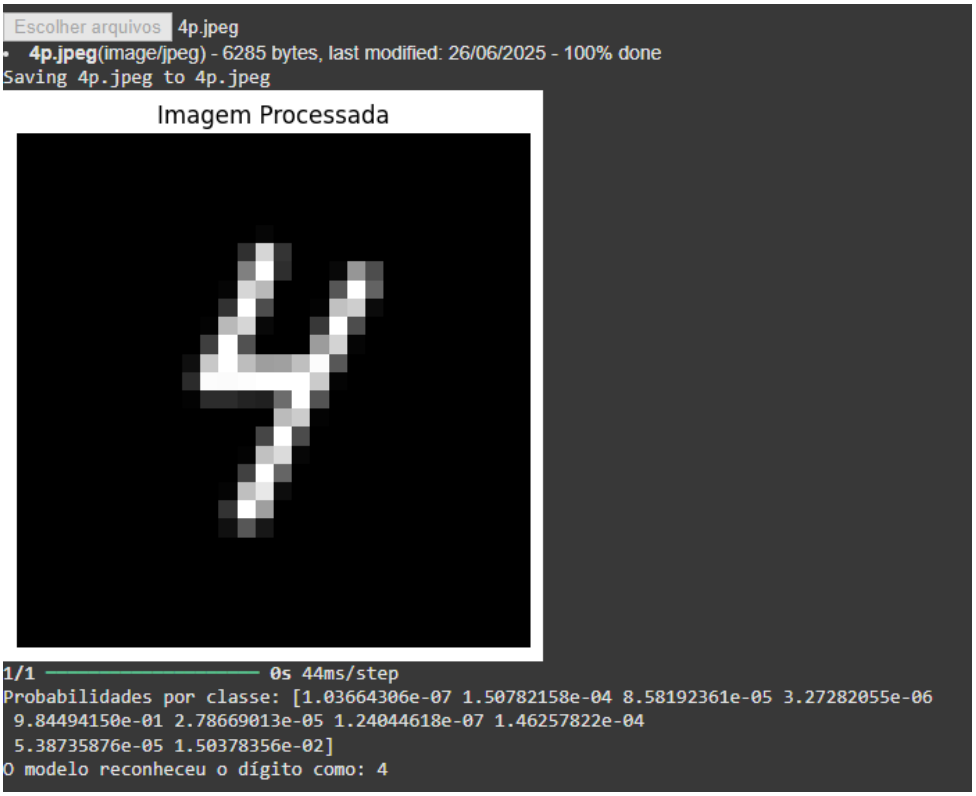


Resultado do teste com o dígito 0 sem pré-processamento - Paulo

Resultado do teste com o dígito 1 sem pré-processamento - Paulo

Resultado do teste com o dígito 2 sem pré-processamento - Paulo

Resultado do teste com o dígito 3 sem pré-processamento - Paulo



Resultado do teste com o dígito 4 sem pré-processamento - Paulo

Resultado do teste com o dígito 5 sem pré-processamento - Paulo

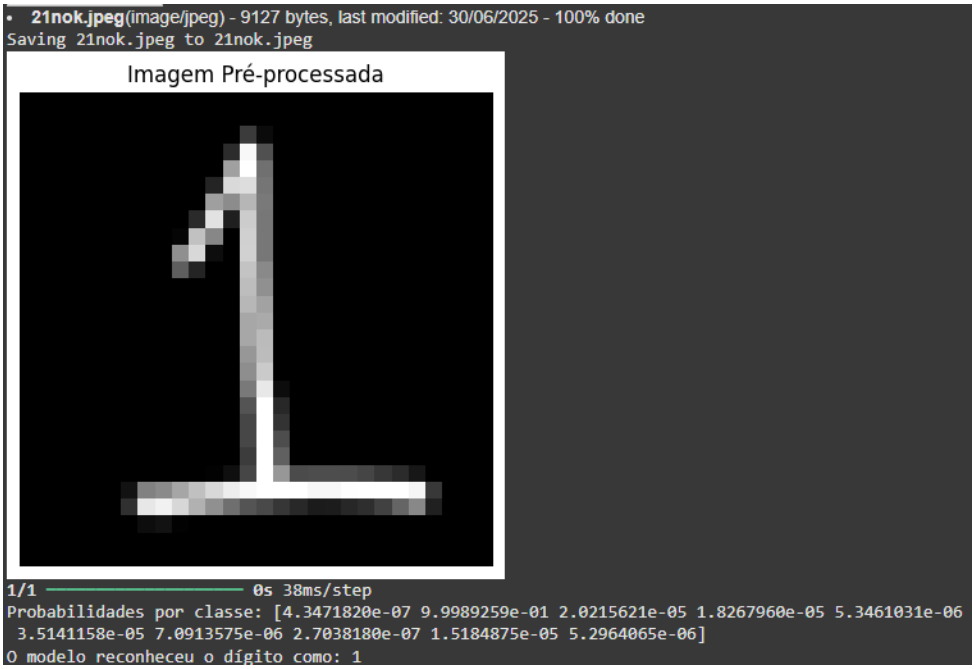
Resultado do teste com o dígito 6 sem pré-processamento - Paulo

Resultado do teste com o dígito 7 sem pré-processamento - Paulo

Resultado do teste com o dígito 8 sem pré-processamento - Paulo

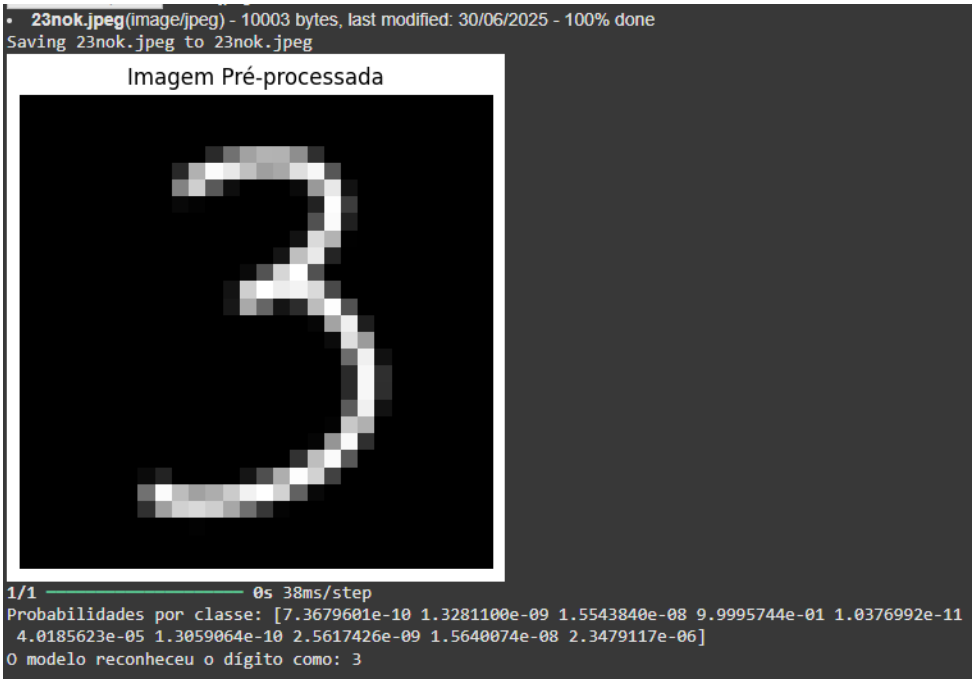
Resultado do teste com o dígito 9 sem pré-processamento - Paulo

Resultado do teste com o dígito 0 sem pré-processamento - Rosane



Resultado do teste com o dígito 1 sem pré-processamento - Rosane

Resultado do teste com o dígito 2 sem pré-processamento - Rosane



Resultado do teste com o dígito 3 sem pré-processamento - Rosane

Resultado do teste com o dígito 4 sem pré-processamento - Rosane

Resultado do teste com o dígito 5 sem pré-processamento - Rosane

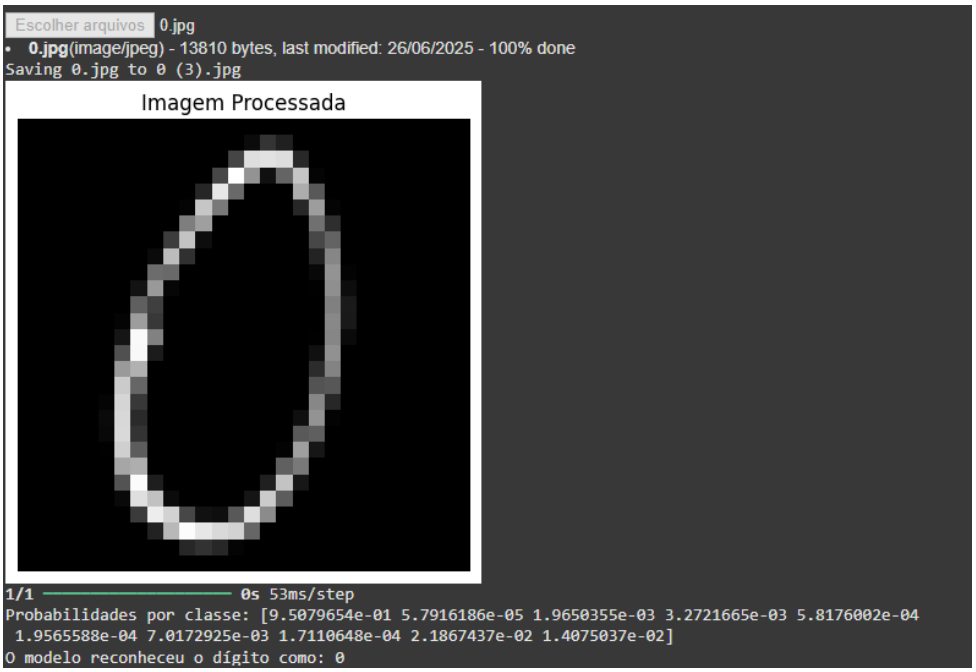
Resultado do teste com o dígito 6 sem pré-processamento - Rosane

Resultado do teste com o dígito 7 sem pré-processamento - Rosane

Resultado do teste com o dígito 8 sem pré-processamento - Rosane

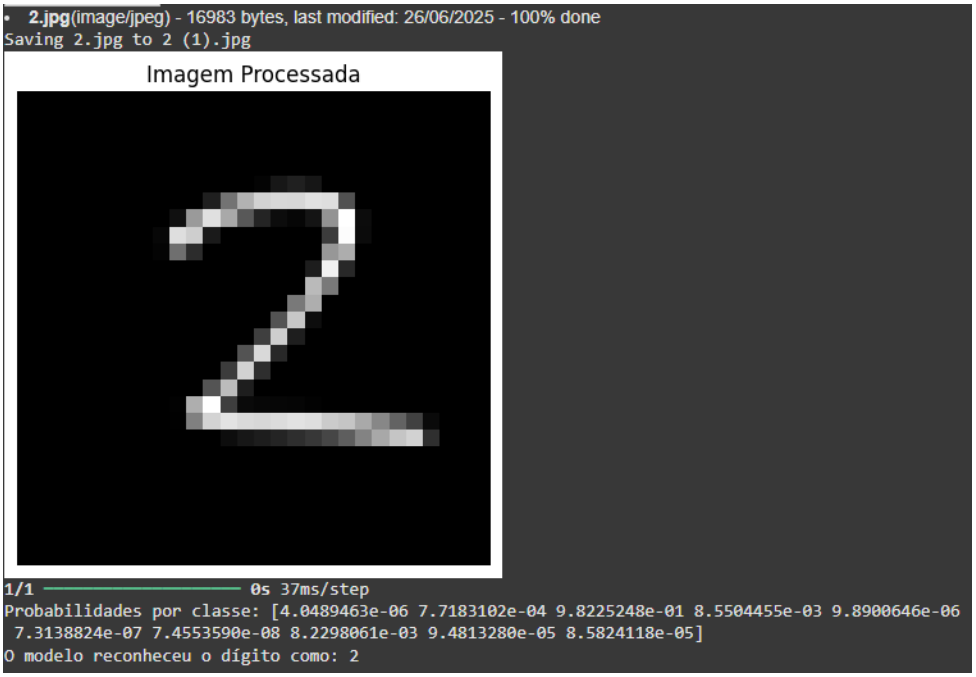
Resultado do teste com o dígito 9 sem pré-processamento - Rosane

APÊNDICE C – RESULTADOS DE CLASSIFICAÇÃO - COM PRÉ-PROCESSAMENTO



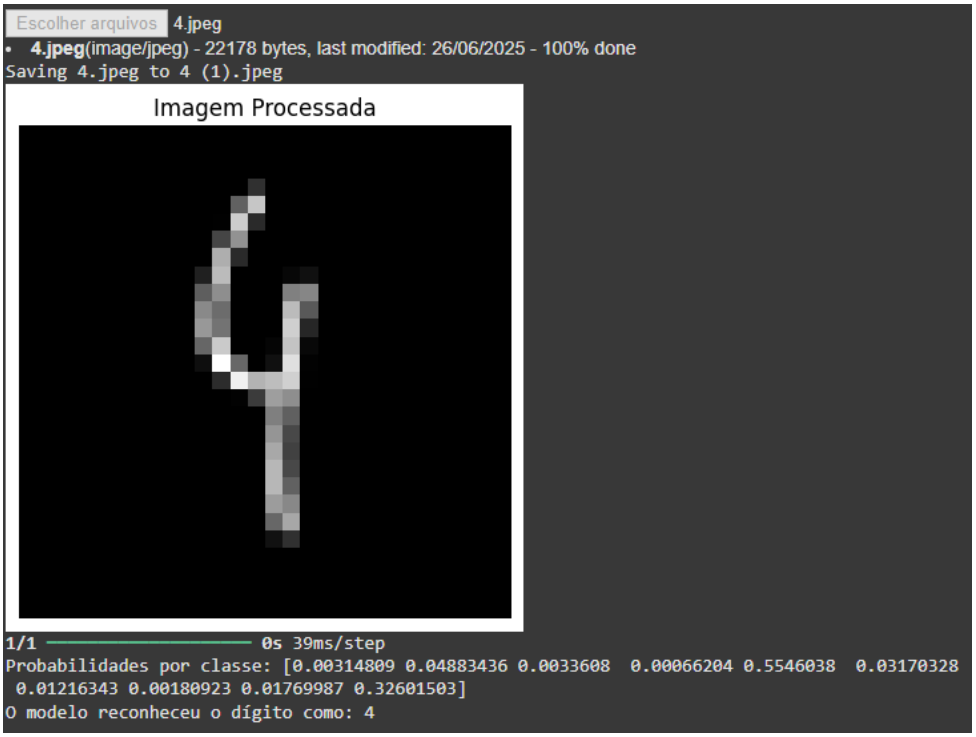
Resultado do teste com o dígito 0 com pré-processamento - Antônio

Resultado do teste com o dígito 1 com pré-processamento - Antônio



Resultado do teste com o dígito 2 com pré-processamento - Antônio

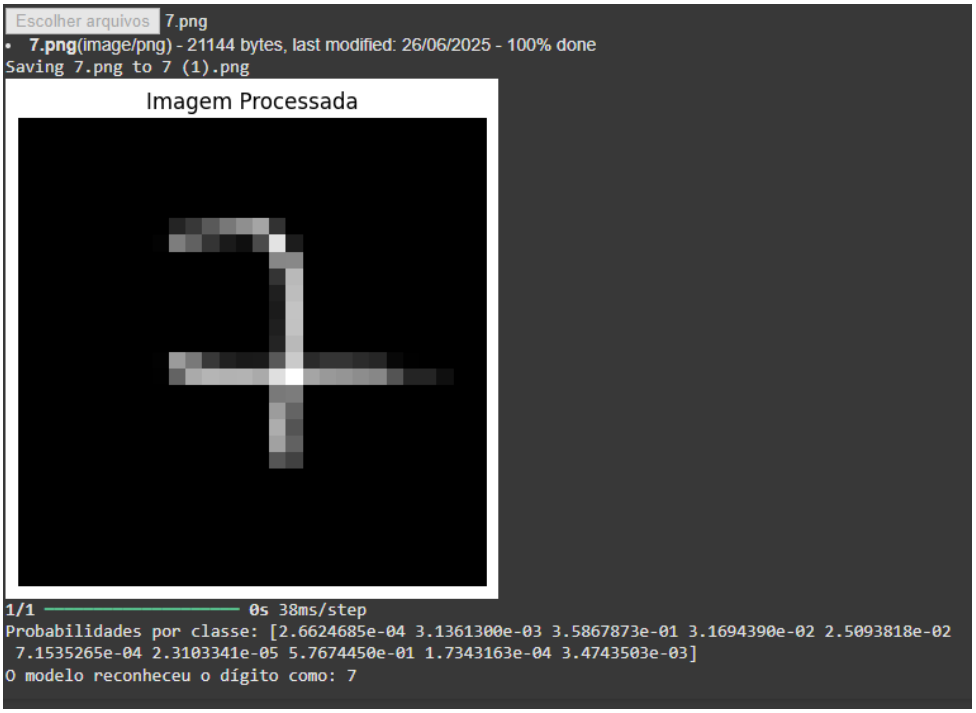
Resultado do teste com o dígito 3 com pré-processamento - Antônio



Resultado do teste com o dígito 4 com pré-processamento - Antônio

Resultado do teste com o dígito 5 com pré-processamento - Antônio

Resultado do teste com o dígito 6 com pré-processamento - Antônio



Resultado do teste com o dígito 7 com pré-processamento - Antônio

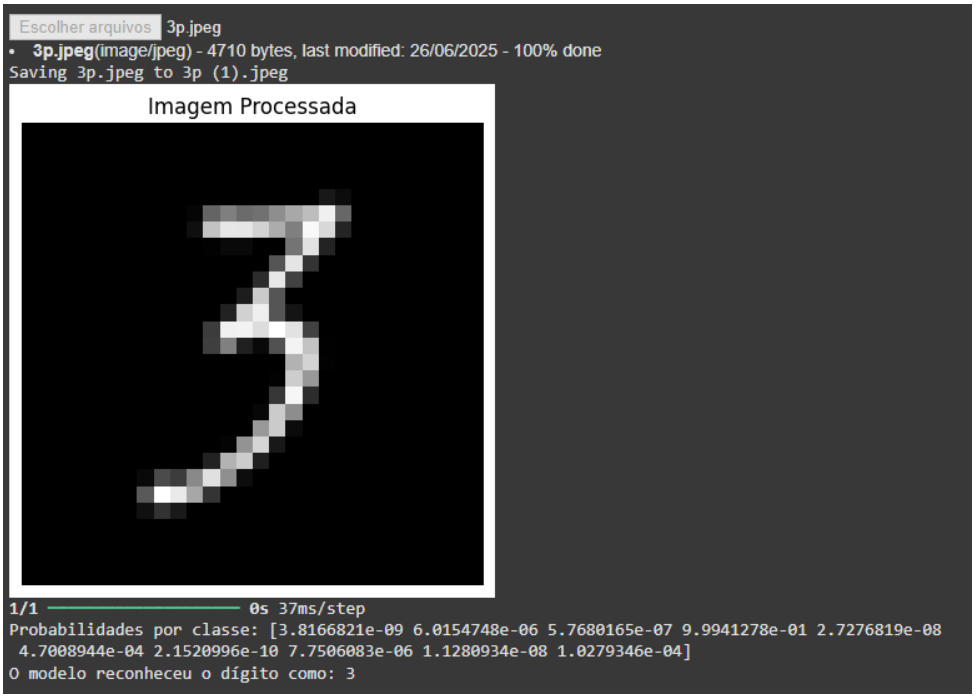
Resultado do teste com o dígito 8 com pré-processamento - Antônio

Resultado do teste com o dígito 9 com pré-processamento - Antônio

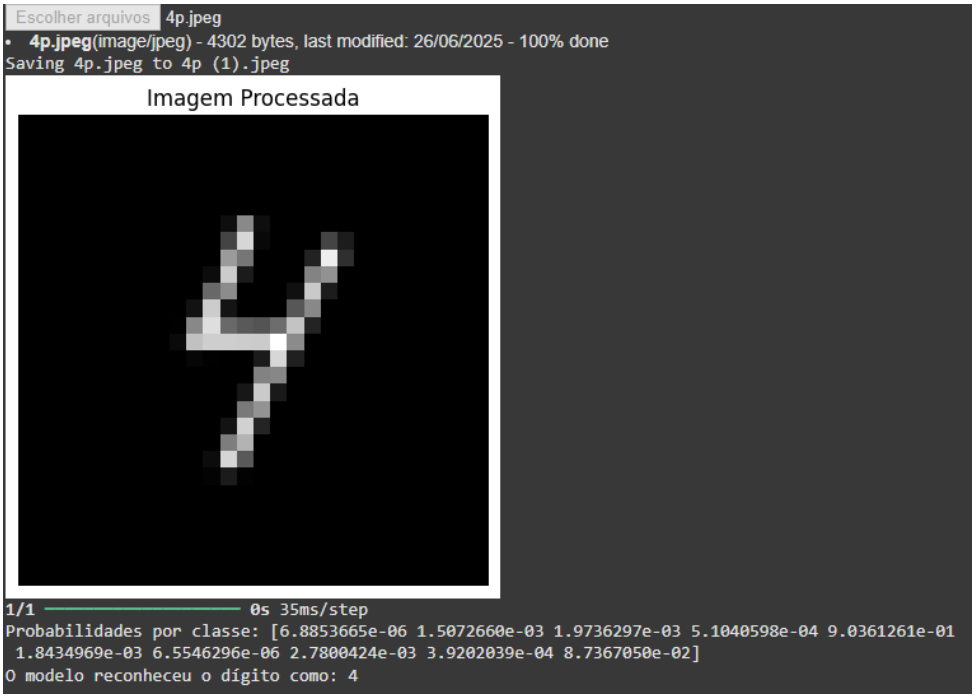
Resultado do teste com o dígito 0 com pré-processamento - Paulo

Resultado do teste com o dígito 1 com pré-processamento - Paulo

Resultado do teste com o dígito 2 com pré-processamento - Paulo



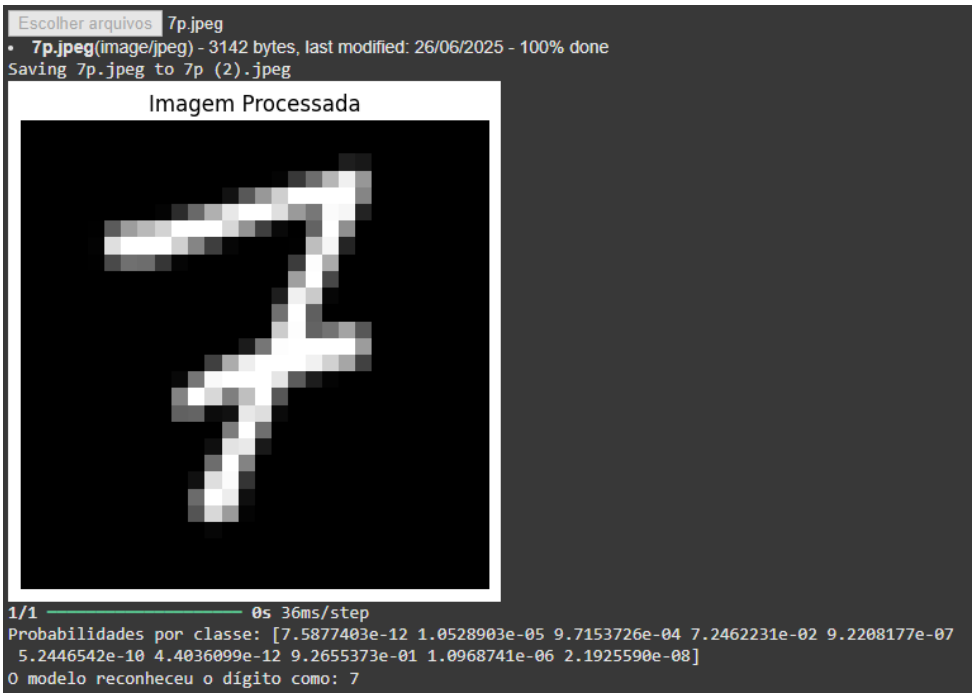
Resultado do teste com o dígito 3 com pré-processamento - Paulo



Resultado do teste com o dígito 4 com pré-processamento - Paulo

Resultado do teste com o dígito 5 com pré-processamento - Paulo

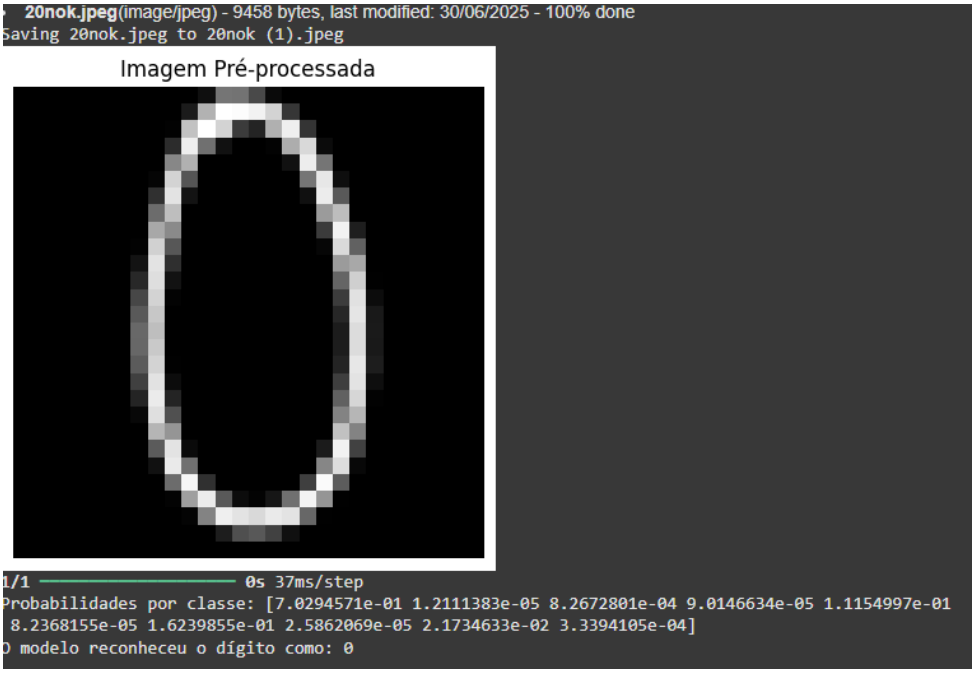
Resultado do teste com o dígito 6 com pré-processamento - Paulo



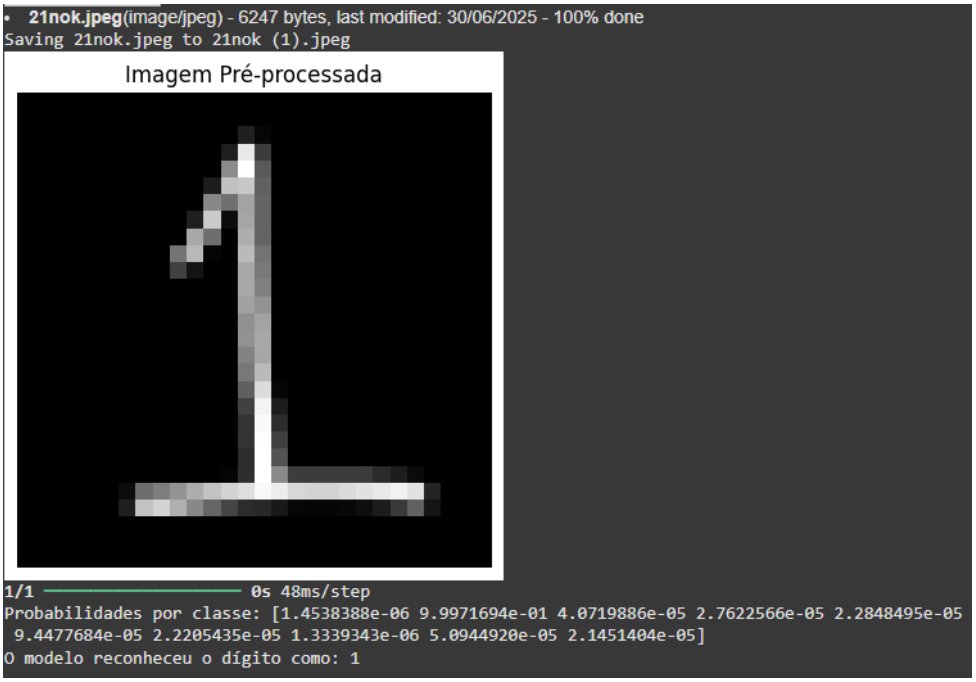
Resultado do teste com o dígito 7 com pré-processamento - Paulo

Resultado do teste com o dígito 8 com pré-processamento - Paulo

Resultado do teste com o dígito 9 com pré-processamento - Paulo



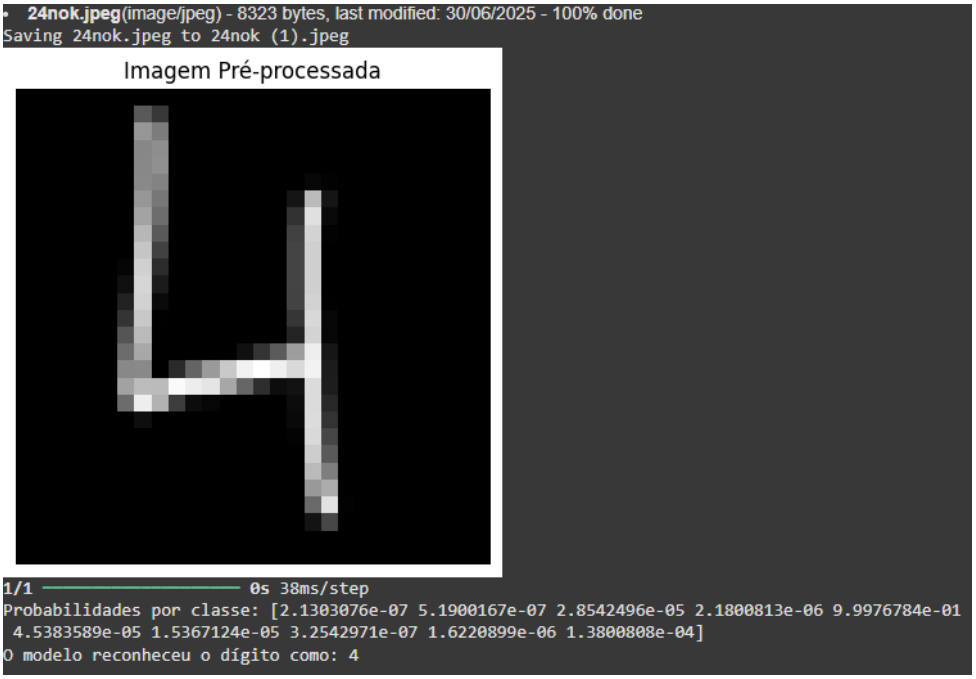
Resultado do teste com o dígito 0 com pré-processamento - Rosane



Resultado do teste com o dígito 1 com pré-processamento - Rosane

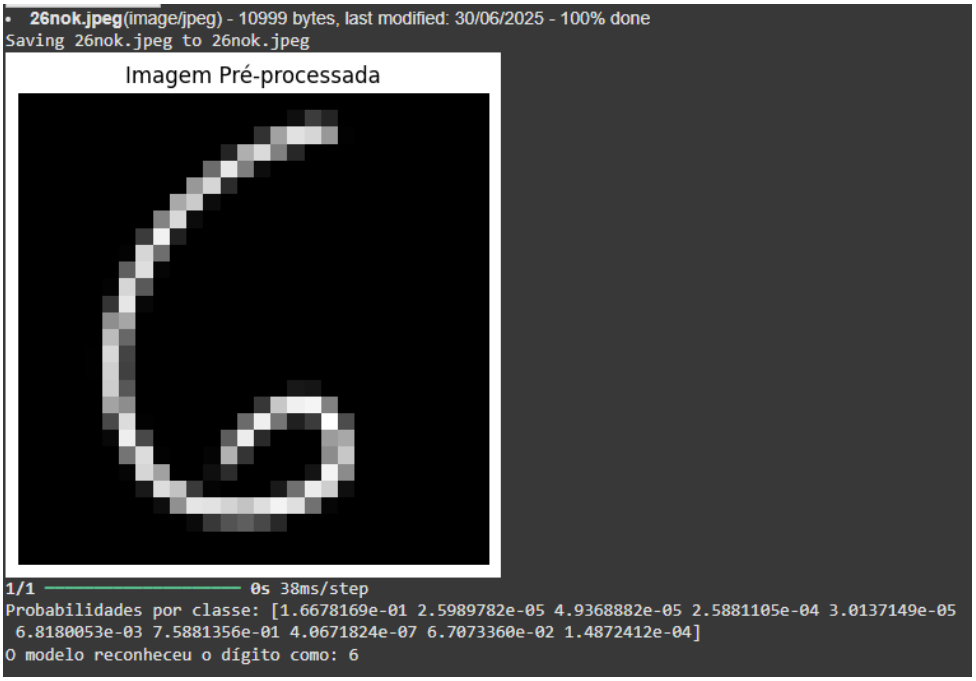
Resultado do teste com o dígito 2 com pré-processamento - Rosane

Resultado do teste com o dígito 3 com pré-processamento - Rosane



Resultado do teste com o dígito 4 com pré-processamento - Rosane

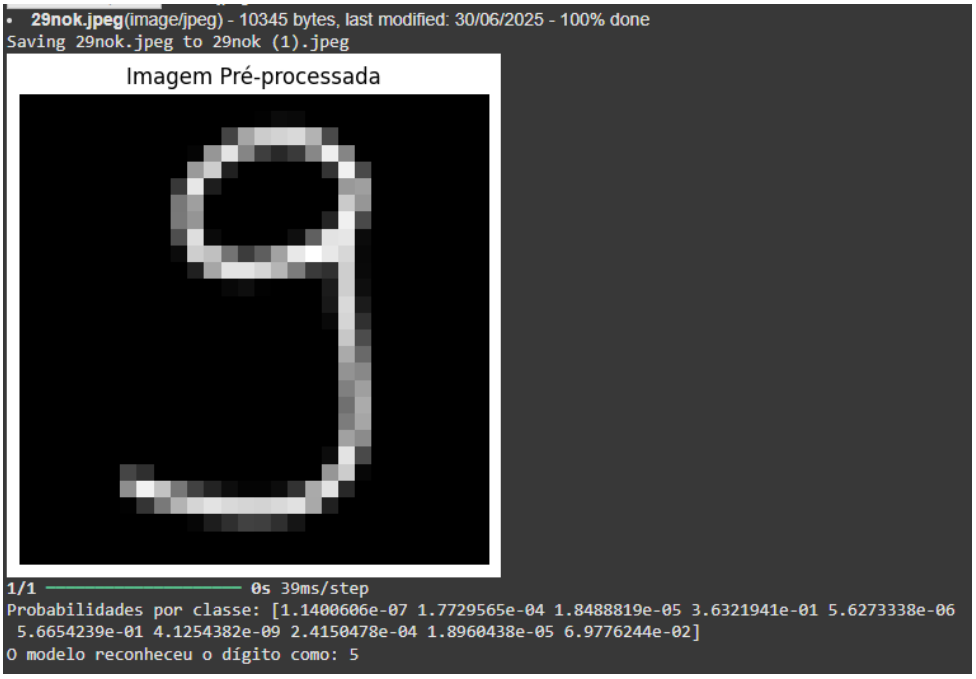
Resultado do teste com o dígito 5 com pré-processamento - Rosane



Resultado do teste com o dígito 6 com pré-processamento - Rosane

Resultado do teste com o dígito 7 com pré-processamento - Rosane

Resultado do teste com o dígito 8 com pré-processamento - Rosane



Resultado do teste com o dígito 9 com pré-processamento - Rosane