UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Uma Proposta de Arquitetura Paralela de Rede Neural Convolucional para Detecção de Padrões em Sequências de DNA

Daniel Ferreira Teodosio

Monografia - MBA em Inteligência Artificial e Big Data

ICMC USP SÃO CARLOS

Daniel Ferreira Teodosio

Uma Proposta de Arquitetura Paralela de Rede Neural Convolucional para Detecção de Padrões em Sequências de DNA

Monografia apresentada ao Departamento de Ciências de Computação do Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo - ICMC/USP, como parte dos requisitos para obtenção do título de Especialista em Inteligência Artificial e Big Data.

Área de concentração: Inteligência Artificial

Orientador: Prof. Dr. Ricardo Cerri

Versão original

São Carlos

2024

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

Teodosio, Daniel Ferreira
T314p Uma Proposta de Arquitetura Paralela de Rede Neural Convolucional para Detecção de Padrões em Sequências de DNA / Daniel Ferreira Teodosio; orientador Ricardo Cerri. -- São Carlos, 2024.
54 p.

Trabalho de conclusão de curso (MBA em Inteligência Artificial e Big Data) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2024.

1. Aprendizado de máquina. 2. Redes Neurais Convolucionais. 3. Identificação de padrões. 4. Bioinformática. 5. Genética. I. Cerri, Ricardo, orient. II. Título.

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

Daniel Ferreira Teodosio

A Parallel Architecture Proposal of Convolutional Neural Network for Pattern Detection in DNA Sequences

Monograph presented to the Departamento de Ciências de Computação do Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo - ICMC/USP, as part of the requirements for obtaining the title of Specialist in Artificial Intelligence and Big Data.

Concentration area: Artificial Intelligence

Advisor: Prof. Dr. Ricardo Cerri

Original version

São Carlos

2024

"Não fosse isso e era menos,
não fosse tanto e era quase"
Paulo Leminski

RESUMO

Teodosio, D. F. **Uma proposta de arquitetura paralela de rede neural convolucional para detecção de padrões em sequências de DNA**. 2024. 54 p. Monografia (MBA em Inteligência Artificial e Big Data) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2024.

A informação genética dos seres vivos, responsável por determinar suas características, está quimicamente codificada em sequências de bases nitrogenadas no núcleo celular, o DNA. Tal informação é transcrita em RNA e traduzida em aminoácidos no citoplasma celular. Os padrões contidos no DNA ou no RNA, que contêm informação útil, seja informação genética ou padrões que servem a propósitos estruturais no processo de transcrição e tradução da informação genética, são, por vezes, demasiadamente complexos e variam em forma e tamanho. Sendo assim, métodos de aprendizado profundo podem ser um caminho eficaz na identificação e classificação de padrões em sequêcias de DNA. Este trabalho apresenta uma proposta de identificação de padrões em sequências de DNA, usando como base, redes neurais convolucionais em uma arquitetura paralela, de forma a extrair com base em diferentes representações de uma mesma amostra de um conjunto de treino, características que uma vez concatenadas, possam se complementar e melhorar o processo de classificação pela camada densa. Os *datasets* utilizados neste trabalho são constituídos de sequencias formadas pelos caracteres A, C, T, G, representando as quatro bases nitrogendadas que formam o DNA e para cada sequência podem ser atribuídas duas classes, conter ou não conter determinado padrão. Como forma de transformar uma sequência de DNA em uma matriz de forma a preservar os padrões posicionais úteis no processo de classificação das amostras pela rede convolucional, foram extraídas subsequências, para cada sequência de DNA, sendo que tais subsequências foram organizadas em forma de colunas e depois substituídas por um vetor binário denso, tomando a forma de uma matriz binária. As diferentes representações de cada matriz numérica, representante de um elemento transformado do *dataset*, dizem respeito aos diferentes tamanhos da janela de convolução que os diferentes segmentos da arquitetura convolucional paralela proposta possuem na sua primeira camada convolucional, portanto se trata de uma análise com base em diferentes tamanhos de janela de convolução na entrada da rede. Foram feitos experimentos treinando variações da arquitetura proposta, no que diz respeito ao número de segmentos paralelos, e também foram feitos experimentos com segmentos sequênciais componentes da arquitetura em questão, de forma a extrair para uma dada implementação, o valor da média de múltiplos experimentos de métricas úteis para avaliação do classificador, como acurácia, F1-score, precisão e revocação, além de valores referentes a média de tempo por época necessários para treinar cada implementação e a média da quantidade de RAM alocada durante o treinamento. Os experimentos foram feitos para dois *datasets* diferentes

e, para ambos, as arquiteturas paralelas, com diferentes números de segmentos, obtiveram, na maioria das vezes, um melhor resultado de acurácia, em relação aos segmentos isolados que as compunham. Além disso os melhores resultados gerais para cada *dataset*, foram obtidos com alguma variação da arquitetura paralela proposta. Porém a quantidade de memória alocada durante o treino e o tempo necessário para o treino da rede, cresceram linearmente em relação ao número de segmentos paralelos e os experimentos não mostram uma correlação direta entre o número de segmentos paralelos e a qualidade do classificador.

Palavras-chave: Aprendizado de máquina. Redes Neurais Convolucionais. Identificação de padrões. Bioinformática. Genética.

ABSTRACT

Teodosio, D. F. **A Parallel Architecture Proposal of Convolutional Neural Network for Pattern Detection in DNA Sequences**. 2024. 54 p. Monograph (MBA in Artificial Intelligence and Big Data) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2024.

The genetic information of living beings, responsible for determining their characteristics, is chemically encoded in sequences of nitrogenous bases in the cell nucleus, the DNA. This information is transcribed into RNA and translated into amino acids in the cell cytoplasm. The patterns contained in DNA or RNA, which contain useful information, whether genetic information or patterns serving structural purposes in the process of transcription and translation of genetic information, are sometimes exceedingly complex and vary in shape and size. Thus, deep learning methods can be an effective way to identify and classify patterns in DNA sequences. This work presents a proposal for identifying patterns in DNA sequences using convolutional neural networks in a parallel architecture, to extract, based on different representations of the same sample from a training set, features that, once concatenated, can complement each other and improve the classification process by the dense layer. The datasets used in this work consist of sequences formed by the characters A, C, T, G, representing the four nitrogenous bases that form DNA, and for each sequence, two classes can be assigned, having or not having a certain pattern. To transform a DNA sequence into a matrix to preserve positional patterns useful in the sample classification process by the convolutional network, subsequences were extracted for each DNA sequence, organized into columns, and then replaced by a dense binary vector, forming a binary matrix. The different representations of each numerical matrix, representing a transformed dataset element, relate to the different convolution window sizes that the different segments of the proposed parallel convolutional architecture have in their first convolutional layer, thus it is an analysis based on different convolution window sizes at the network input. Experiments were conducted training variations of the proposed architecture, regarding the number of parallel segments, and also experiments with sequential segments of the architecture in question, to extract for a given implementation, the average value of multiple experiments of useful metrics for classifier evaluation, such as accuracy, F1-score, precision, and recall, as well as values related to the average time per epoch needed to train each implementation and the average amount of RAM allocated during training. Experiments were conducted for two different datasets, and for both, the parallel architectures with different numbers of segments often achieved better accuracy results compared to the isolated segments that composed them. Additionally, the best overall results for each dataset were obtained with some variation of the proposed parallel architecture. However, the amount of memory allocated during training and the time required for network training

grew linearly with the number of parallel segments, and the experiments did not show a direct correlation between the number of parallel segments and the classifier quality.

Keywords: Machine learning. Convolutional Neural Networks. Pattern Recognition. Bioinformatics. Genetics.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

DNA	Deoxyribonucleic Acid
MLP	Multi Layer Perceptron
RAM	Random Acess Memory
RNA	Ribonucleic Acid
ReLu	Rectified Linear Unit
XOR	Exclusive OR

SUMÁRIO

1 INTRODUÇÃO

O DNA (*Deoxyribonucleic Acid*), contido no núcleo celular, é responsável por codificar as informações genéticas dos seres vivos. Tal informação é armazenada na forma de grandes sequências de nucleotídeos A, C, T, G (adenina, citosina, timina, guanina). A informação contida no DNA é então transcrita em cadeias de RNA (*Ribonucleic Acid*), que é formado pelas mesmas bases nitrogenadas, porém com a uracila no lugar da timina. Por fim, a informação genética é traduzida com a produção de aminoácidos no citoplasma celular, com base na informação sequencial contida no RNA transcrito (Alberts, 2017). Os padrões, que contêm informações úteis para o processo de transcrição e tradução e que podem estar presentes em grandes sequências de DNA e RNA são, por vezes, demasiadamente complexos e intrincados para serem identificados por inspeção visual. Sendo assim, métodos de aprendizado de máquina podem ser um caminho a ser explorado na identificação de tais padrões.

Ainda no contexto de aprendizado de máquina, as redes neurais convolucionais, ou CNN's (*Convolutional Neural Networks*), tem se mostrado muito eficientes na identificação de características de alto nível (Lecun *et al.*, 1998) e, por esse motivo, muito utilizadas no processamento digital de imagens para classificação e reconhecimento de objetos. Porém, a capacidade das CNN's de identificar características espaciais sutis, pode ser aplicada em outros contextos como, por exemplo, no processamento de linguagem natural tal qual em (Kim, 2014). Para tanto, são feitas transformações nas amostras que farão parte do treinamento e validação da rede, criando representações destas que permitam utilizar o potencial da CNN de identificar padrões espaciais de interesse. Retornando às sequências de DNA ou RNA e levando em consideração a tarefa de classificá-las no que diz respeito a possuírem ou não determinados padrões importantes no processo de codificação de informação genética, podemos considerar as grandes sequências de nucleotídeos tal qual um texto, sendo possível assim usar CNN's para identificação de padrões em sequências de DNA e RNA.

A proposta desse trabalho está centrada em uma abordagem para classificação de longas sequências de nucleotídeos, no que diz respeito a terem ou não determinado padrão escolhido, que visa usar como entrada para o treinamento da rede, simultaneamente, diferentes aspectos de uma mesma amostra do conjunto de treinamento da rede.

Partindo do trabalho desenvolvido em (Giang *et al.*, 2016), que transforma uma sequência extensa de nucleotídeos em um conjunto de subsequências derivadas, para fins de criar uma representação matricial, que preserva a informação posicional da sequência de nucleotídeos original e que serve como elemento amostral de treino para rede e tendo como motivação os resultados consideráveis reportados em (Johnson; Zhang, 2014) e em

(Kim, 2014), na extração e utilização de diferentes aspectos de uma mesma amostra do conjunto de treino no processo de treinamento da rede, este trabalho propõe um estudo focado nos possíveis benefícios de usar uma arquitetura de CNN paralela, que extrai concomitantemente características de alto nível de diferentes aspectos de uma mesma amostra do conjunto de treino. A hipótese que motiva tal estudo é que diferentes aspectos de uma mesma amostra possam se complementar entre si, adicionando informação útil no treinamento da rede.

O estudo que este trabalho propõe tem como objetivo reportar possíveis melhorias e benefícios na classificação de cadeias de nucleotídeos, mediante ao fato das mesmas conterem ou não um dado padrão, usando a extração em paralelo de características de diferentes representações de uma mesma amostra, o método será aplicado para diferentes bases de dados. Para tal, será implementada uma CNN com uma arquitetura paralela e serão feitos experimentos com diferentes níveis de paralelismo. Também será feita uma reflexão, no que diz respeito à viabilidade de tal abordagem, mediante ao possível aumento de recursos computacionais e tempo de treinamento da rede.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Considerações iniciais

No amplo espectro de técnicas do universo do aprendizado de máquina, esse trabalho vai tratar exclusivamente de redes neurais, mais especificamente de redes neurais convolucionais. Portanto, vamos explorar os conceitos que dizem respeito à essa arquitetura de rede neural. Vamos fazer uma breve retrospectiva histórica que nos conduza até o surgimento de nossa arquitetura de interesse e então vamos tratar dos fundamentos da mesma e da aplicação desta no domínio da bioinformática e da genética.

2.2 Panorama histórico dos modelos conexionistas

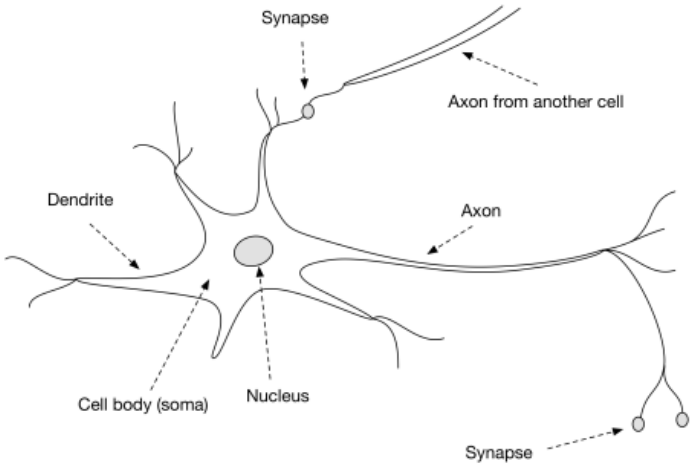
Ainda na década de 40 do século passado, McCulloch e Pitts publicaram um estudo onde propunham um modelo simples de neurônio articificial, baseado, em certos aspectos, no neurônio biológico, que poderiam ser usados em conjunto para resolver problemas de lógica proposicional (McCulloch; Pitts, 1943).

Na década de 50, continuando na linha de reproduzir artificialmente certos aspectos do aprendizado biológico dos seres vivos, Frank Rosemblat propôs seu perceptron (Rosenblatt, 1958), um tanto quanto diferente do neurônio artificial de McCulloch e Pitts. O perceptron de Rosemblat podia realizar classificação de problemas linearmente separáveis. A forma como o perceptron de Rosemblat é capaz classificar padrões linearmente separáveis se dá por meio de um algoritmo iterativo de treinamento, baseado na obra de Donald Hebb, *The Organization of Behavior* (Hebb, 1949). A obra em questão sugere que quando um neurônio biológico aciona outro neurônio frequentemente, a conexão entre estes se torna mais forte.

A representação do conhecimento através da relevância das conexões entre os neurônios biológicos norteia a regra de treinamento do perceptron, que consiste em atualizar iterativamente as conexões (ou pesos) da rede, em função do erro da rede no que diz respeito ao valor esperado na saída (usando exemplos de treino).

Segue abaixo a representação de um neurônio biológico. Tais células possuem ramificações chamadas de dendritos e uma ramificação especialmente grande, chamada axônio. Os axônios possuem nas suas extremidades sinapses, que se ligam às ramificações de outros neurônios. Os neurônios produzem pequenos potenciais elétricos, transmitidos através das sinapses por meio de neurotransmissores.

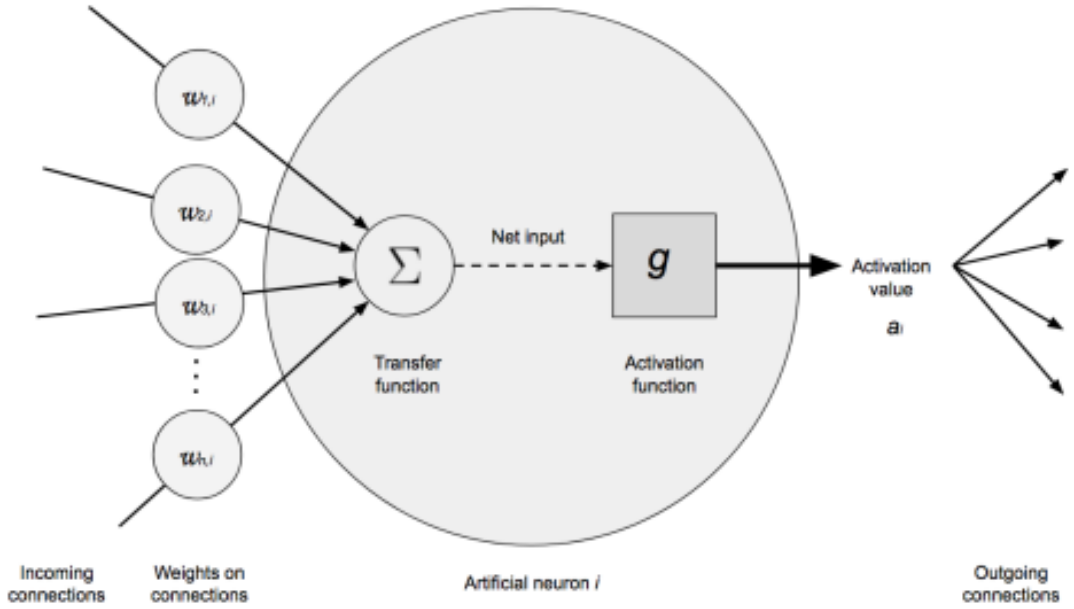
Figura 1 – Neurônio biológico



Fonte:(Patterson; Gibson, 2017)

O perceptron de Rosemblat é composto por uma camada de neurônios de entrada, que representam os valores de entrada da rede, ligadas através de pesos ajustáveis, representados por conexões, a um neurônio de saída que aplica uma transformação – chamada de função de ativação – à soma das entradas multiplicadas pelos seus respectivos pesos.

Figura 2 – Perceptron de Rosemblat



Fonte:(Patterson; Gibson, 2017)

Segue abaixo a fórmula do ajuste iterativo dos pesos do perceptron em função do erro da rede:

$$w_{i,j}^{atualizado} = w_{i,j}^{antigo} + \eta(y - \hat{y})x_i \tag{2.1}$$

Como mostra a fórmula 2.1, o ajuste de cada conexão, em cada iteração, é proporcional à diferença entre o valor esperado na saída da rede e o valor obtido (y e $\hat{y}$, respectivamente), além de cada respectiva entrada x_i e do parâmetro η, que controla a taxa de aprendizado dos parâmetros da rede. A saída $\hat{y}$ da rede é o resultado da aplicação de uma função de ativação ao somatório das entradas da rede multiplicadas pelos seus respectivos pesos. Uma vez que as conexões da rede convergem de forma a atribuir o correto valor na saída para cada uma das entradas, o conhecimento da rede na resolução do problema está representado internamente pelo valor dos parâmetros ou conexões da rede. Tal conceito não muda, em essência, para arquiteturas que foram propostas posteriormente e que vamos discutir adiante.

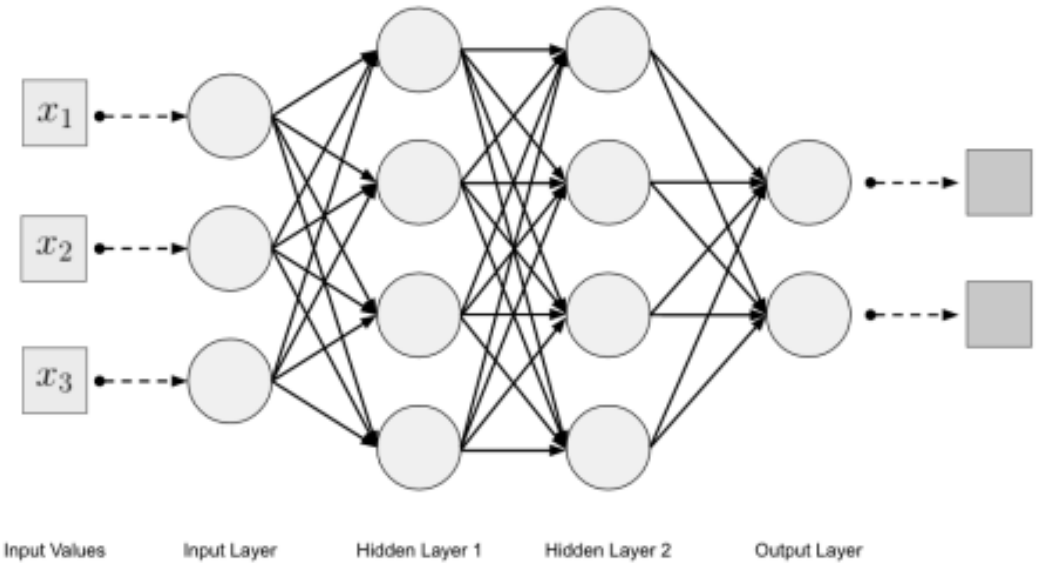
2.3 Multilayer perceptron, conceitos e fundamentos

2.3.1 Arquitetura de rede

Como mencionado anteriormente, o perceptron de Rosemblat é capaz de aprender a classificar padrões linearmente separáveis, contudo, problemas simples como o de aprender as saídas de uma porta lógica XOR (*exclusive OR*), é um problema que está além do que o perceptron simples consegue aprender (Géron, 2019). Em seu trabalho de 1969 (Minsky; Papert, 1969), Minsky e Papert exploraram as limitações do perceptron. Tais limitações somadas à falta de recursos computacionais mais robustos, conduziu a um desinteresse nas redes neurais durante à década de 70 até meados da década de 80.

Porém as limitações do perceptron simples de Rosemblat – no que diz respeito à classificação de problemas que não são linearmente separáveis – podem ser contornadas empilhando camadas de perceptrons tal qual a figura 3.

Figura 3 – Arquitetura MultiLayer Perceptron



Fonte:(Patterson; Gibson, 2017)

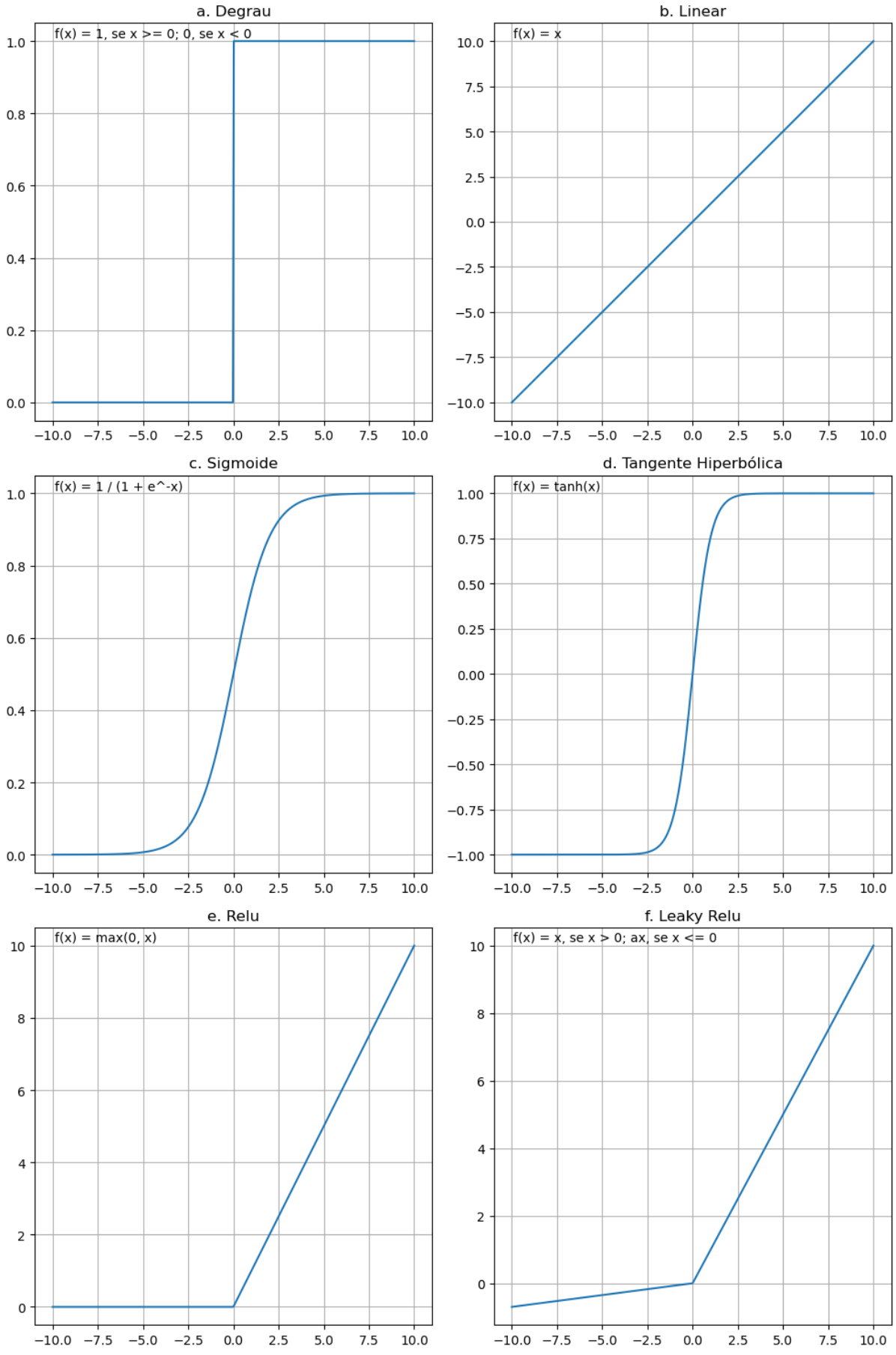
A arquitetura de rede mostrada acima é chamada de multilayer perceptron e vamos nos referir a mesma como MLP (*MultiLayer Perceptron*). A MLP, é composta por uma camada de neurônios de entrada que simplesmente propagam para frente os valores de entrada da rede, um conjunto de camadas de neurônios ocultas, onde todos os neurônios estão ligados por pesos da rede à todos os neurônios da camada seguinte e, por fim, existe uma camada de saída composta por um ou mais neurônios. Tal arquitetura de rede funciona como um aproximador de funções genérico, podendo ser usado em tarefas de regressão ou classificação (Haykin, 2009).

2.3.2 Funções de ativação

Em uma rede MLP, a saída de um neurônio de camada oculta que, eventualmente, serve de entrada para outro neurônio oculto, é modulada por funções de ativação que aplicam uma transformação em escala à soma dos pesos de entrada do respectivo neurônio multiplicados pelas suas respectivas entradas. Anteriormente, havíamos comentado sobre a função de ativação para o perceptron de Rosemblat. Nesse caso a função de ativação é uma função degrau simples, onde a resposta é igual a 1 para uma entrada maior ou igual a 0 e é 0, caso contrário.

Na figura 4 são mostradas outras funções de ativação comumente usadas em redes neurais.

Figura 4 – Funções de ativação



Fonte: Autor

As funções sigmoide e tangente hiperbólica pertencem ao grupo de funções sigmoidais e adicionam, quando usadas, um fator de não linearidade à rede. A função sigmoide também pode ser usada como função de saída para redes de classificação. A função ReLu, também mostrada na figura 4, mostrou resultados superiores às funções sigmoidais em redes profundas (Géron, 2019), porém, a ReLu tende, eventualmente, a causar a morte de neurônios da rede, situação na qual certos neurônios passam a ter valor sempre igual a zero na fase de treinamento. Para sanar ou diminuir a tendência que tal problema ocorra, a função Leaky ReLu tem como saída pequenos valores negativos para entradas menores que zero.

Vale mencionar ainda a função de ativação softmax, que força a saída dos neurônios da última camada da rede – que representam as diferentes classes que podem ser atribuídas às amostras de entrada – a atribuir valores análogos à uma distribuição de probabilidade, para as classe possíveis, sendo uma generalização da função sigmoide para múltiplas classes.

2.3.3 Funções de perda

O valor que serve como referência para o ajuste dos parâmetros da rede é calculado levando em consideração os valores de saída da rede, para tanto é necessário utilizar uma função que transforma a saída na rede em uma quantidade representativa do quão próximo a rede está da resposta esperada, estas são denominadas funções de perda. Como mencionado anteriormente, as redes MLP podem ser usadas como aproximadores genéricos de funções, para regressão e classificação, portanto, vamos abordar algumas funções de perda, que se destinam a tais propósitos. Vamos denominar a função de perda por $P(W)$, onde W representa os parâmetros treináveis da rede, para sinalizar que o erro da rede é uma função dos seus parâmetros ajustáveis.

2.3.3.1 Função de erro médio quadrático e erro médio absoluto

Dado um conjunto de N amostras do conjunto de treino, o erro médio quadrático para as N amostras pode ser calculado pela fórmula 2.2, $\hat{Y}$ e Y são vetores que representam a saída obtida e a saída esperada para a i-ésima amostra, respectivamente.

$$P(W) = \frac{1}{N}\sum_{i=1}^{N}(\hat{Y}_i - Y_i)^2 \tag{2.2}$$

Em vez de calcular a média da diferença quadrática para um conjunto de tamanho N da rede, também é possível calcular a média do erro absoluto, como na fórmula 2.3

$$P(W) = \frac{1}{N}\sum_{i=1}^{N}|\hat{Y}_i - Y_i| \tag{2.3}$$

Ambas as funções de erro mostradas acima são, normalmente, aplicadas à redes aproximadoras regressoras.

2.3.3.2 Função de entropia cruzada e função de perda de articulação

A funções mostradas até agora dizem respeito à redes que se destinam a serem regressores, para redes onde o objetivo é atribuir uma classe de um conjunto finito de classes à uma dada amostra de entrada, podemos usar funções de perda mais adequadas como a função de entropia cruzada representada na fórmula 2.4, para N amostras e M classes. Aqui $\hat{y_{ij}}$ e y_{ij} não são vetores, mas sim a saída da rede e o valor correto esperado para o j-ésimo neurônio em relação à i-ésima amostra.

$$P(W) = -\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(\hat{y}_{ij}) \tag{2.4}$$

A função de entropia cruzada acima tende a ser aplicada em situações onde a probabilidade de uma dada amostra pertencer a uma dada classe é o fator mais importante a ser levado em consideração. Quando temos uma questão relacionada à classificação binária – uma entrada da rede é ou não de uma determinada classe – A função de perda de articulação (*hinge function* em inglês), pode ser uma melhor opção e a mesma está representada na fórmula 2.5.

$$P(W) = \frac{1}{N}\sum_{i=1}^{N}\max(0, 1 - y_{ij} \times \hat{y}_{ij}) \tag{2.5}$$

2.3.4 Métricas de avaliação de classificadores

No que diz respeito a um dado classificador, podendo ser uma rede neural ou qualquer outro, existem métricas que representam numericamente a eficiência do mesmo. Usar uma medida ou outra depende das circunstâncias e do contexto no qual o classificador vai ser usado (Géron, 2019). Seguem abaixo algumas métricas usadas para avaliar a performance de um dado classificador,onde VP, FP, VN, FN, significam “verdadeiro positivo”, “falso positivo”, “verdadeiro negativo”, “falso negativo”, respectivamente:

$$Precisão = \frac{VP}{VP + FN} \tag{2.6}$$

$$Sensibilidade = \frac{VP}{VP + FP} \tag{2.7}$$

$$Acurácia = \frac{VP + VN}{VP + FN + VN + FP} \tag{2.8}$$

$$F1 - score = \frac{2}{\frac{1}{Precisão} + \frac{1}{Sensibilidade}} \tag{2.9}$$

Caso seja mais importante que o classificador não gere previsões de falso positivo, a métrica de precisão pode ser mais eficaz, caso o classificador seja treinado para identificar uma taxa alta dos casos de verdadeiro positivo, a medida de sensibilidade (também chamada de revocação) é mais apropriada. Caso a precisão e a sensibilidade sejam importantes, a medida de acurácia, dada pela fórmula 2.8 pode ser uma opção, porém, para conjuntos onde existe um desbalanceamento grande do número de amostras para as diferentes classes, a métrica de acurácia deixa de ser um bom indicador da qualidade do classificador. Como alternativa, temos a métrica F1-score, que só é alta se a precisão e a sensibilidade forem altas (Géron, 2019).

2.3.5 Treinamento de redes MLP

No caso do perceptron de Rosemblat, que possui somente uma camada oculta, a regra de aprendizado iterativa de Hebb é o suficiente para ajustar a única camada de pesos da rede em função do erro na saída na rede, porém no caso da arquitetura MLP, que possui, eventualmente, várias camadas de neurônios conectadas pelos pesos da rede, surge a questão de como treinar tal arquitetura de forma que os pesos da rede acabem convergindo de forma a minimizar o erro na saída da rede.

2.4 Gradiente descendente e algoritmo de backpropagation.

Por um certo tempo a maneira mais eficiente de como treinar uma rede MLP foi uma questão em aberto entre os pesquisadores. Porém, em 1986 Rummelhart, Willians e Hinton publicaram um artigo, mostrando um método eficaz de ajustar os pesos das camadas mais internas da rede (Rumelhart; Hinton; Williams, 1986), baseado em retropropagar a "culpa" pelo erro na saída da rede para as camadas mais internas de neurônios até chegar à entrada da rede.

Dada uma função $f(x_1, x_2, ..., x_n)$, o vetor gradiente corresponde a derivada parcial da função em questão, em relação a cada um de seus parâmetros, como mostrado em 2.10, além disso, o vetor gradiente, para um dado ponto na curva ou superfície n-dimensional representada pela função em questão, representa a direção no qual o deslocamento leva ao maior incremento ou decremento no valor da função, dependendo do sentido do deslocamento.

$$\nabla f = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_3}, ..., \frac{\partial f}{\partial x_n}\right] \tag{2.10}$$

Partindo deste princípio o algoritmo de *backpropagation* calcula, iterativamente, o gradiente descendente, partindo de um ponto qualquer de uma curva, com o objetivo

de encontrar, no menor número de iterações possíveis, o ponto de mínimo da função. O tamanho do deslocamento em cada etapa define o comportamento da descida até o ponto de mínimo. Caso o deslocamento seja relativamente pequeno em cada iteração, a descida tende a ser mais suave, porém com o custo de demorar mais, ao passo que um deslocamento maior conduz à uma descida possivelmente mais rápida, mas também mais instável, além disso, dependendo das características da curva ou superfície em questão, o mínimo global – o ponto mais baixo da curva ou da superfície em questão – pode não ser atingido e o gradiente descendente pode ficar preso em um mínimo local (Patterson; Gibson, 2017).

O ponto principal do algoritmo de *backpropagation* reside em considerar o erro da rede como uma função – função de custo, ou função de perda – a ser minimizada pelo método do gradiente descendente. Para isso, considera-se que o erro ou função de custo na saída da rede é uma função dos pesos da rede. O vetor gradiente mostrado em 2.10, representa a taxa de variação da função em relação aos seus respectivos parâmetros, sendo assim para a função de perda $P(W)$, o vetor gradiente usado para diminuir a função de custo é dado por:

$$\nabla P = \left[\frac{\partial P}{\partial w_1}, \frac{\partial P}{\partial w_2}, \frac{\partial P}{\partial w_3}, ..., \frac{\partial P}{\partial w_n} \right] \tag{2.11}$$

Para uma rede onde os pesos da rede estão espalhados em múltiplas camadas, o algoritmo de *backpropagation* calcula as derivadas parciais do vetor gradiente 2.11 iterativamente, percorrendo a rede da saída até a entrada, atualizando os pesos da rede em cada iteração. As funções de perda mostradas como exemplo, 2.2, 2.3, 2.4, 2.5, levam em consideração a média das N amostras de treino, porém, o cálculo do erro, também pode ser feito usando *mini-batches* ou pequenos subconjuntos do conjunto completo de treino. O parâmetro η, que controla a taxa de aprendizado, mostrado na equação 2.1, também é usado no algoritmo de *backpropagation*, para controlar a taxa de aprendizado.

Quando o algoritmo de *backpropagation* itera sobre todos os elementos do conjunto de treino da rede, dizemos que uma época foi completa. O algoritmo de *backpropagation* continua o processo de ajuste dos parâmetros da rede, até que seja alcançado um número de épocas estabelecido. Um subconjunto de teste, pode ser criado, a partir de elementos do *dataset* que não farão parte do processo de treino da rede, para o teste da efetividade do treino da rede ao seu término.

A ideia do treinamento da rede neural, é que a mesma consiga generalizar respostas adequadas, mesmo para um dado elemento que não faz parte do seu conjunto de treino. Para um dado conjunto de treino, a métrica utilizada na medição da qualidade do treino tende a crescer e o erro médio da rede tende a diminuir com o passar das épocas de treino, porém, em dado momento do treino, a rede começa a perder o poder de generalização e começa a se especializar no conjunto de treino utilizado no processo de treinamento.

Tal fenômeno é conhecido como *overfitting* (Haykin, 2009). Uma forma de prevenir o processo de *overfitting* é criar um subconjunto do *dataset*, que não participa do processo de treinamento, e que, periodicamente, é usado para medir se a rede ainda está tendo ganho na métrica usada para medir a qualidade do seu treino ou se um processo de estagnação ou retrocesso foi alcançado, sendo que em tal circunstância, o treinamento da rede é interrompido.

2.5 Redes neurais convolucionais

2.5.1 Motivação para as arquiteturas convolucionais

No que diz respeito a arquitetura MLP, apesar do grande poder de generalização e capacidade de aprendizado das mesmas por meio do ajuste de seus parâmetros treináveis internos, a mesma possui limitações para certas tarefas, como na classificação de imagens, uma vez que as amostras de treino dentro de uma mesma classe possuam muita variação entre si. Além disso, no que diz respeito à imagens, que são formadas por pixels, a depender da resolução das imagens que venham a fazer parte do conjunto de treino, o número de parâmetros ou conexões da rede pode ficar muito grande, tornando o treinamento demorado e exigindo muitos recursos computacionais (Lecun *et al.*, 1998).

Além disso, para a detecção de características sutis em imagens é plausível que sejam necessárias redes de grande profundidade com um número grande de camadas. Uma forma de contornar tais problemas é através de um pré-processamento. Através do conhecimento prévio do domínio no qual a rede neural vai ser usada, é possível extrair e criar representações para características de uma dada imagem e usar tais características extraídas como entrada para o treinamento da rede. Porém, a necessidade do conhecimento prévio e da necessidade de criar extratores de características para diferentes tarefas referentes a diferentes domínios, abre uma brecha para uma solução de caráter mais genérico no que diz respeito à classificação de imagens, uma solução que não precise de conhecimento prévio para extrair as características que são de interesse para a classificação de imagens pertencentes a um dado contexto (Lecun *et al.*, 1998). Nesse ponto entram a redes convolucionais.

2.5.2 Paralelo biológico

Assim como no caso das redes MLP, também podemos traçar paralelos em relação ao funcionamento do cérebro dos animais no caso das redes convolucionais. O trabalho de Hubbel e Wiesel de 1962 (HUBEL; WIESEL, 1962) sobre o acionamento de neurônios nos cérebros de gatos teve como conclusão que diferentes regiões do córtex visual são acionadas por diferente partes componentes de uma imagem. Certas áreas são ativadas por bordas e linhas em determinados ângulos, outras respondem à luminosidade ou movimento.

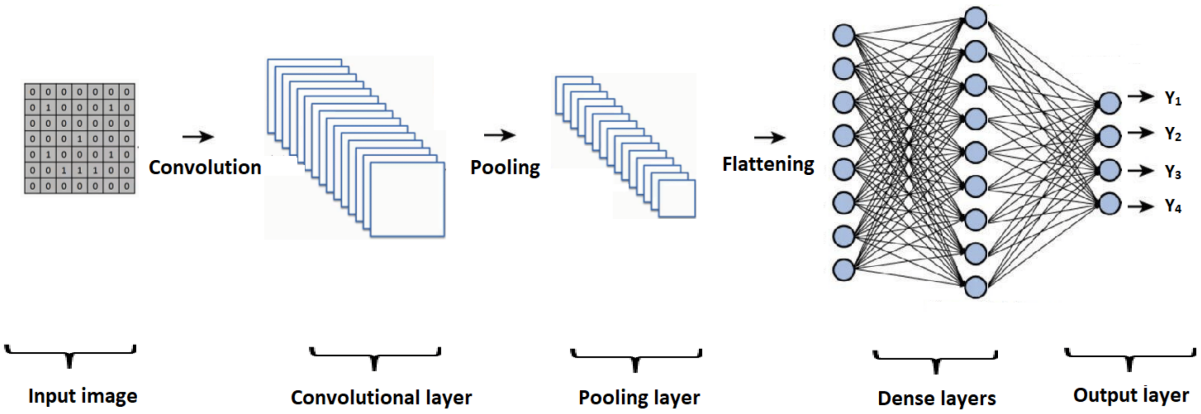
Como vamos elaborar mais à frente as redes neurais convolucionais trazem no seu cerne a ideia de campos receptivos que respondem significativamente somente a certos padrões da imagem que serve de entrada para rede.

2.5.3 Arquitetura convolucional básica

Retornando ao que foi mencionado anteriormente em relação à falta de praticidade na necessidade de criar soluções customizadas para extrair características de imagens com o intuito de realizar tarefas de classificação para um determinado contexto de imagens, as redes convolucionais trazem uma proposta parecida, no que diz respeito a extrair características importantes para classificação que servem de entrada para uma MLP.

Porém, nas redes convolucionais, o processo de extração de características das imagens é feito por filtros de convolução, sendo que não é necessário muito conhecimento prévio sobre o domínio ao qual pertencem as imagens, pois os filtros são ajustáveis e fazem parte do conjunto de parâmetros treináveis da rede, sendo assim, tais filtros são inferidos no processo de treinamento (Lecun *et al.*, 1998).

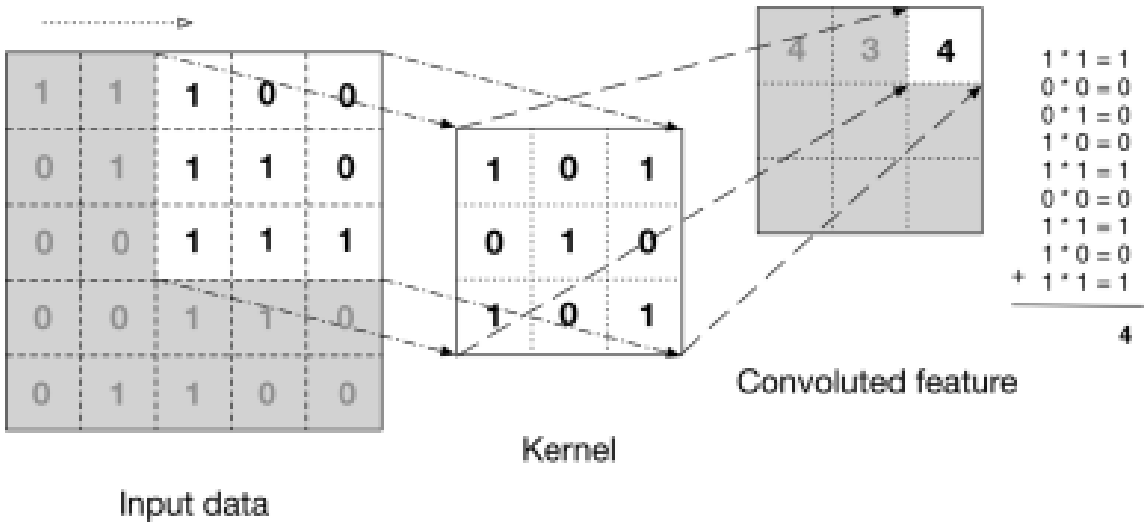
Figura 5 – Exemplo de arquitetura convolucional



Fonte:(Maeda-Gutiérrez *et al.*, 2020)

A base da proposta das arquiteturas convolucionais está nas camadas de convolução e na sua capacidade de filtrar padrões da imagem de entrada. A operação de convolução pode ser interpretada como o deslizamento de um ou mais filtros – ou campos receptivos – ao longo da imagem de entrada, que pode ser representada por uma matriz numérica, no caso de imagens monocromáticas, ou, múltiplas matrizes representando diferentes canais no caso de imagens coloridas.

Figura 6 – Operação de convolução



Fonte:(Patterson; Gibson, 2017)

A convolução consiste do produto escalar da região englobada, em um dado momento da filtragem, pelo kernel de convolução (campo receptivo, ou janela de convolução). Quando a operação de convolução é feita ao longo de toda a imagem, o resultado é um mapa de características que contêm as características que foram filtradas, em função dos parâmetros que compõem o kernel de convolução (Patterson; Gibson, 2017).

Além da operação de convolução, a operação de *pooling* – que normalmente está situada entre duas camadas de convolução – também está presente em boa parte das arquiteturas convolucionais. A operação de *pooling* consiste em diminuir a resolução, ou a quantidade de informação de um dado mapa de caraterísticas. Tal processo, assim como a operação de convolução, também pode ser interpretado como uma janela que desliza ao longo da imagem ou mapa de características derivado e dependendo do tipo de *pooling*, mapeia um conjunto de elementos do mapa de características para um único elemento, que pode ser calculado pela média dos elementos englobados pela janela de *pooling*, ou alguma outra operação, como o valor máximo dentro da janela de *pooling* (Patterson; Gibson, 2017).

As operações de convolução e *pooling* em conjunto, sanam, até certo ponto, os problemas de usar imagens como entrada de uma rede MLP. Por exemplo, a quantidade de parâmetros treináveis nas redes convolucionais, mesmo com uma grande quantidade filtros, tende a ser bem menor do que a quantidade de conexões, caso uma MLP simples fosse usada no processo de classificação.

Outro ponto a ser notado é que, uma vez que os parâmetros de um dado kernel de convolução tenham convergido, pós treinamento, para identificar determinado padrão, tal padrão pode ser identificado em qualquer local da imagem, uma vez que o kernel

se desloca por toda a representação matricial da imagem. Como consequência disso, as redes convolucionais possuem certa resiliência à translações e imagens que não estão centralizadas. Uma parte dessa robustez também se deve às camadas de *pooling*, que, ao diminuírem a resolução dos mapas de característica, diminuem a sensibilidade da rede à pequenas rotações na imagem (Lecun *et al.*, 1998). As características, filtradas ao longo das camadas de convolução e *pooling*, são transformadas em um vetor, que serve de entrada para uma rede MLP, responsável pela classificação das características extraídas.

2.6 Aprendizado profundo aplicado a problemas de sequenciamento genético

A informação genética dos seres vivos, que está codificada quimicamente em forma de sequências de bases nitrogenadas que formam o DNA, contido no interior das células dos organismos multicelulares e unicelulares, normalmente, se apresenta em padrões complexos. Os processos de replicação da informação genética durante a reprodução celular e os processos que transcrevem DNA em RNA e que traduzem RNA nas proteínas necessárias nos processos biológicos dependem destas sequências. Tais sequências podem ser informação com DNA codificante, indicadores que sinalizam o início de sequências codificadoras no DNA, longas sequências, que não necessariamente contêm informação útil para codificar aminoácidos, mas que possuem função estrutural nos processos celulares.

Tais padrões, por vezes possuem uma variedade significativa de tamanho e forma (Alberts, 2017). Com a ascensão dos métodos de aprendizado profundo, tarefas relacionadas à classificar trechos de DNA e RNA que contenham determinados padrões, podem ser feitas por meio de treinamento de modelos de aprendizado profundo.

2.6.1 Trabalhos relacionados à classificação de padrões em sequências de DNA e RNA usando redes convolucionais

Uma grande multiplicidade de trabalhos tem demonstrado a eficácia de abordagens de aprendizado profundo na classificação ou identificação de cadeias de DNA ou RNA que contenham alguma característica de interesse. Usando técnicas baseadas em modelos conexionistas ou não, tais trabalhos mostram que é possível alcançar uma taxa de acurácia na classificação de tais sequências que eventualmente ultrapassa métodos mais antigos e tradicionais de identificação de padrões, mesmo que estes sejam o estado da arte do que se propõem a fazer.

No trabalho (Cruz *et al.*, 2020) os autores usam uma arquitetura de rede convolucional para classificação de sequências de elementos transponíveis contidos em sequências de DNA. Elementos transponíveis são sequências de nucleotídeos que se repetem ao longo da cadeia de DNA, não tendo muitas vezes qualquer função, mas podendo também modificar trechos de DNA codificante (Alberts, 2017). Para realizar tal classificação, cada uma das bases nitrogenadas contida nas sequências de treinamento foram transformadas em vetores

esparsos e cada sequência foi representada por uma matriz para servir de entrada para a arquitetura de rede construída.

Em (Zeng *et al.*, 2016) também é usada uma arquitetura de rede convolucional, porém neste caso, a tarefa de classificação está relacionada a identificação de sequências que possuem diferentes fatores de transcrição. Fatores de transcrição são sequências que sinalizam o início de um trecho codificante de DNA, porém, tais sinalizadores se apresentam em uma variedade muito grande de possíveis padrões, no que diz respeito ao tamanho da cadeia e ordem das bases nitrogenadas (Alberts, 2017). Mesmo com a complexidade da tarefa, os autores relataram resultados satisfatórios ao explorar diferentes arquiteturas de rede.

O método proposto em (Giang *et al.*, 2016), faz uso de uma arquitetura convolucional, mas tem como diferencial a forma como representa as sequências de nucleotídeos do conjunto de dados. Neste trabalho, as sequências são trabalhadas como se fossem um texto e subsequências são criadas a partir das sequências originais, criando um vocabulário que é mapeado para vetores esparsos, a partir dos quais, representações matriciais para as sequências são criadas. A abordagem em questão é aplicada para diferentes bases de dados, destinadas ao treinamento de diferentes tarefas de classificação. Os autores, em seus resultados, afirmam ter alcançado o estado de arte no que diz respeito à alguns dos conjuntos de dados usados para validação.

3 METODOLOGIA

3.1 Considerações iniciais

Nesta seção será detalhada a metodologia que será aplicada aos experimentos, feitos a fim de gerar indicadores que permitam analisar os possíveis benefícios de utilizar o modelo de rede convolucional paralelo mencionado anteriormente neste trabalho, no que diz respeito a possíveis melhorias na performance de classificação da rede e também levando em conta os possíveis gastos de recursos computacionais oriundos da arquitetura proposta. Porém, primeiro vamos explicar os detalhes de implementação da arquitetura de rede.

3.2 Implementação da arquitetura de rede proposta

Como mencionado anteriormente na introdução, este trabalho propõe a implementação de uma arquitetura paralela de CNN (*Convolutional Neural Network*), que para uma dada amostra de treino, extraia características de diferentes representações da mesma, concomitantemente, com a pretensão de que isso possa adicionar informação útil na etapa de classificação das características filtradas pela rede.

A arquitetura proposta pode ser representada de forma genérica pela figura 7. Para cada segmento paralelo da rede, o número de camadas convolucionais, *pooling*, quantidade dos mapas de características criados em cada camada é a mesma, mudando somente as dimensões do kernel de convolução das primeiras camadas convolucionais de cada segmento paralelo – ou seja o tamanho do kernel de convolução da primeira camada convolucional de cada segmento paralelo. A razão para isso é que na metodologia abordada para os experimentos, foram feitos testes usando diferentes implementações, com diferentes quantidades de segmentos paralelos e, para a análise na diferença de assertividade das implementações, optamos por restringir a análise dos diferentes resultados à mudança de apenas um fator, o tamanho da janela de convolução das primeiras camadas convolucionais. Do contrário, o número de hiperparâmetros diferentes que poderiam ser combinados geraria uma quantidade demasiadamente grande de cenários, tornando a análise dos resultados muito complexa.

Na arquitetura de rede proposta na figura 7, o número de mapas de características aumenta a medida que a rede vai ficando mais profunda. A razão disso é que as camadas convolucionais superficiais ficaram encarregadas de filtrar características de nível mais baixo dos padrões apresentados como entrada, porém essas características de baixo nível podem ser combinadas numa variedade grande de maneiras para criar representações de características de nível mais alto (Géron, 2019). Isso explica o número crescente de filtros

ao longo das camadas de convolução. No final, as características extraídas são concatenadas e são classificadas por uma MLP.

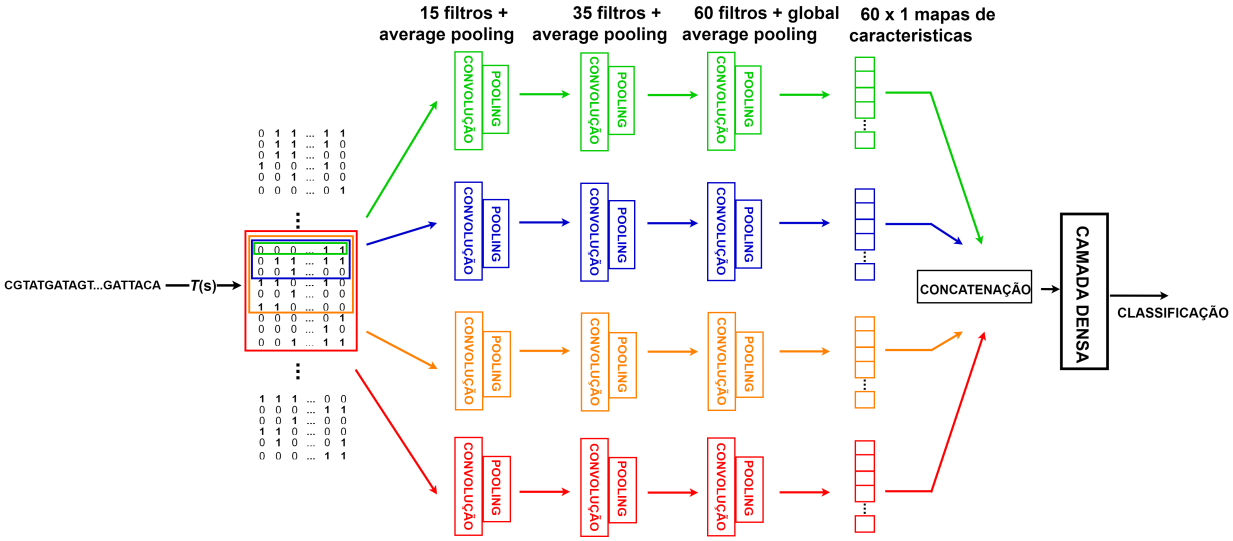
Na nossa abordagem, para cada segmento paralelo, foram utilizadas 3 camadas de filtros, com um número de filtros de 15, 35 e 60, respectivamente. Após as duas primeiras camadas de filtros, foram utilizadas camadas de *pooling* do tipo *average pooling*, que diminui a resolução dos mapas de característica oriundos da filtragem, mapeando um conjunto de elementos englobados pela janela de *pooling* para um único valor, calculando a média dos mesmos. A camada de *pooling* usada depois da última camada de filtragem é do tipo *global average pooling*, que mapeia todos os elementos de um mapa de características para um único elemento, calculando a média dos mesmos. Como consequência, cada segmento paralelo produz um vetor de 60x1 elementos, que são concatenados em um único vetor, que serve de entrada para a camada densa de classificação, que corresponde à MLP.

Optou-se por não usar um número grande de neurônios na MLP, sendo este restrito a 3 camadas ocultas e 1 camada de saída, com 100, 32, 10 e 1 neurônios, respectivamente. A função de ativação escolhida para os neurônios ocultos, foi a Leaky ReLu. A função de perda utilizada durante o treinamento, foi a função de entropia cruzada.

No que diz respeito aos *datasets* que foram utilizados nos experimentos, abordaremos isso melhor mais a frente, porém em todos os casos, a nossa classificação é binária. Sendo assim, a última camada da MLP poderia ter sido uma softmax, com dois neurônios. Outra possibilidade é a de usar um único neurônio na saída com uma função de ativação sigmoidal contida no intervalo [0, 1] e definindo um limiar de 0,5 para classificação. Na nossa abordagem, optamos pela segunda opção.

Toda implementação da rede foi feita usando a API de alto nível Keras da biblioteca TensorFlow (Géron, 2019).

Figura 7 – Arquitetura da rede paralela



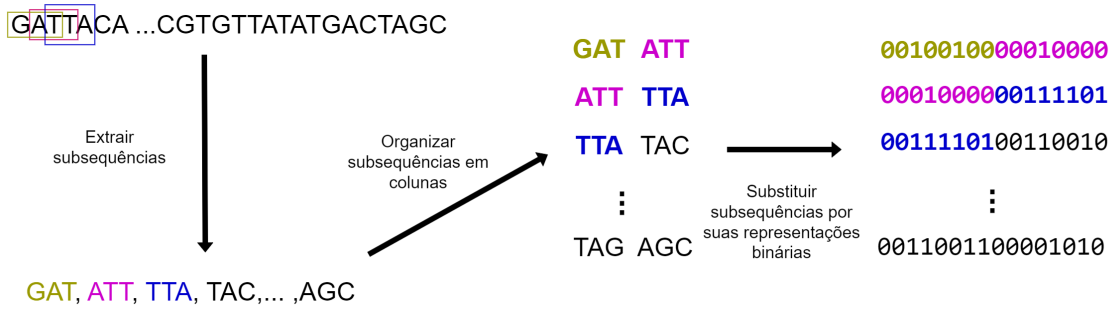
Fonte: Autor

Tal qual mostrado na figura 7, acima, os elementos dos *datasets* a serem usados, são compostos por cadeias de caracteres (A, C, G, T), representando uma sequência de DNA. Porém, é necessário criar a partir dos elementos disponíveis nos *datasets*, uma representação matricial que preserve os padrões espaciais e posicionais dos elementos em questão. A transformação $T(s)$ – onde s é uma sequência de DNA qualquer do *dataset*, será explicada na seção seguinte.

3.2.1 Método de geração de representação matricial para as sequências de nucleotídeos

A transformação $T(s)$ – onde s é uma sequência qualquer de nucleotídeos – aplicada para toda sequência s, pertencente a um *dataset* de treino, é a mesma desenvolvida em (Giang *et al.*, 2016). A ideia consiste em extrair subsequências da sequência original deslizando uma janela ao longo desta. O tamanho do deslocamento e da janela podem variar, mas para este trabalho, os valores escolhidos para o tamanho da janela e do deslocamento são 3 e 1, respectivamente. As subsequências são organizadas, conforme a figura 8, em duas colunas e, ao longo de várias linhas, na ordem em que são extraídas da sequência original. Porém, para toda linha a partir da segunda, a subsequência da primeira coluna é a subsequência da segunda coluna da linha anterior.

Figura 8 – Criando representação matricial



Fonte: Autor

Uma vez que organizamos as subsequências em colunas, estas foram substituídas por uma representação vetorial e assim teremos criado uma representação matricial que servirá de entrada para nossa rede, com uma regra de formação que preserva as características espaciais do elemento original do *dataset*.

Em (Giang *et al.*, 2016), para cada possível sequência de 3 nucleotídeos, foi associado um vetor esparso de dimensões 64 x 1. Para um alfabeto de 4 letras e uma subsequência de 3 nucleotídeos, existem 4^3 ou 64 possibilidades, uma para cada posição do vetor esparso que substitui cada subsequência.

Porém, com o intuito de usar uma representação densa em vez de vetores esparsos, neste trabalho, foi criado um dicionário, associando cada uma das possíveis 64 subsequências a uma representação binária, de 8 bits, correspondente a sua respectiva posição em uma organização por ordem alfabética. Por exemplo, para a subsequência AAA está associada a primeira posição então sua representação vetorial seria 00000001. Para AAB, na posição dois, seria 00000010. Para AAC, na posição três seria 00000011 e assim por diante, conforme figura 8.

Em geral, os trabalhos que usam redes convolucionais na identificação de padrões em sequências de DNA e RNA usam vetores esparsos, como em (Cruz *et al.*, 2020) e (Zeng *et al.*, 2016). Porém, estamos partindo do pressuposto da capacidade, por parte das redes convolucionais, de identificar padrões espaciais mesmo usando essa representação diferente, pois o fato de que cada subsequência está associada unicamente a um vetor não muda e o padrão posicional é preservado.

3.2.2 Extração paralela de características

No método de criação da representação matricial da rede, abordado na seção anterior, as sequências de nucleotídeos, que fazem parte dos conjuntos de dados de treino da rede, são subdivididas como um texto que é separado em suas respectivas palavras. No caso, para cada segmento paralelo da rede, as dimensões do filtro de convolução aplicadas

à representação matricial de entrada da rede são diferentes, divergindo em relação às dimensões do kernel de convolução. As características extraídas por cada segmento paralelo se dariam, portanto, com base em diferentes níveis de granularidade no que diz respeito à sequência original. Retomando o paralelo entre a sequência de DNA e um texto, seria como extrair informação sobre sílabas, palavras ou frases que compõem um texto, de forma paralela.

Por fim, essas diferentes características, extraídas a partir de diferentes níveis de granularidade, foram concatenadas e classificadas por uma MLP. Partimos da hipótese, embasada pelos resultados de trabalhos mencionados anteriormente, que tal abordagem de extração paralela de características pode adicionar informação útil no processo de classificação e melhorar a assertividade da rede.

3.3 Planejamento dos experimentos

Voltando à arquitetura proposta na figura 7, no que diz respeito ao nível de paralelismo máximo que queremos testar, optamos por fazer experimentos na rede com até 4 segmentos em paralelo. Como também já mencionado, cada segmento paralelo se diferencia pelo tamanho da janela de convolução da primeira camada de convolução. O tamanho das janelas de convolução escolhidas para cada segmento são 1x16, 3x16, 6x16, 9x16.

3.3.1 Etapa de testes com implementações não paralelas

Inicialmente, o treino e coleta de métricas ocorreu em implementações não paralelas. As métricas relativas à assertividade – como acurácia, precisão, sensibilidade, F1-score – foram extraídas mediante os resultados de treino com os diferentes tamanhos da janela de convolução mencionados anteriormente. Além disso também foram extraídas as métricas referentes à performance computacional (métricas de tempo e de memória utilizada).

3.3.2 Etapa de testes com implementações paralelas

Em um segundo momento foram feitos experimentos utilizando implementações paralelas com o intuito de extrair as mesmas métricas mencionadas anteriormente. A dinâmica de testes seguiu a seguinte ordem:

1. Testes com dois segmentos em paralelo com dimensões da janela de convolução da primeira camada de 1x16 e 3x16.
2. Testes com três segmentos em paralelo com dimensões da janela de convolução da primeira camada de 1x16, 3x16, 6x16.
3. Testes com quatro segmentos em paralelo com dimensões da janela de convolução da primeira camada de 1x16, 3x16, 6x16, 9x16.

3.3.3 Métodos de extração de métricas

O modo e extração das métricas quantitativas, mencionadas anteriormente, se deu da seguinte forma:

Tanto para os testes com implementações não paralelas, quanto para os testes com implementações paralelas, cada implementação foi treinada para dois *datasets* diferentes e para todas elas foram usados os mesmos hiperparâmetros, no que diz respeito à taxa de aprendizado e as dimensões da MLP de classificação. Foram usados *mini-batches* de tamanho 32, durante o processo de treinamento e o número de épocas foi variável. No caso, o número de épocas de treino usadas foi a quantidade necessária para que a rede pudesse atingir o seu melhor resultado, ou seja, até que a rede mostrasse sinais de *overfitting* em relação à validação que foi feita ao longo do processo de treino.

Para garantir que as métricas de avaliação da rede representassem de forma fidedigna a qualidade do classificador, para cada implementação, foram criados conjuntos de treino, teste e validação de forma aleatória, para cada *dataset*, na proporção de 70%, 15%, 15%, respectivamente. Tal processo foi repetido 20 vezes para cada implementação e as métricas extraídas foram obtidas através da média dos resultados obtidos.

No que diz respeito às métricas de performance computacional, a quantidade de memória utilizada para cada treino de cada implementação foi obtido através da biblioteca Psutil (Rodola, 2021), que permite aferir a quantidade memória alocada para um dado método. Os valores em questão também foram ponderados, calculando a média em relação ao número de execuções. O tempo médio de execução para as épocas do treino das implementações também foi calculado e armazenado para análise de resultados.

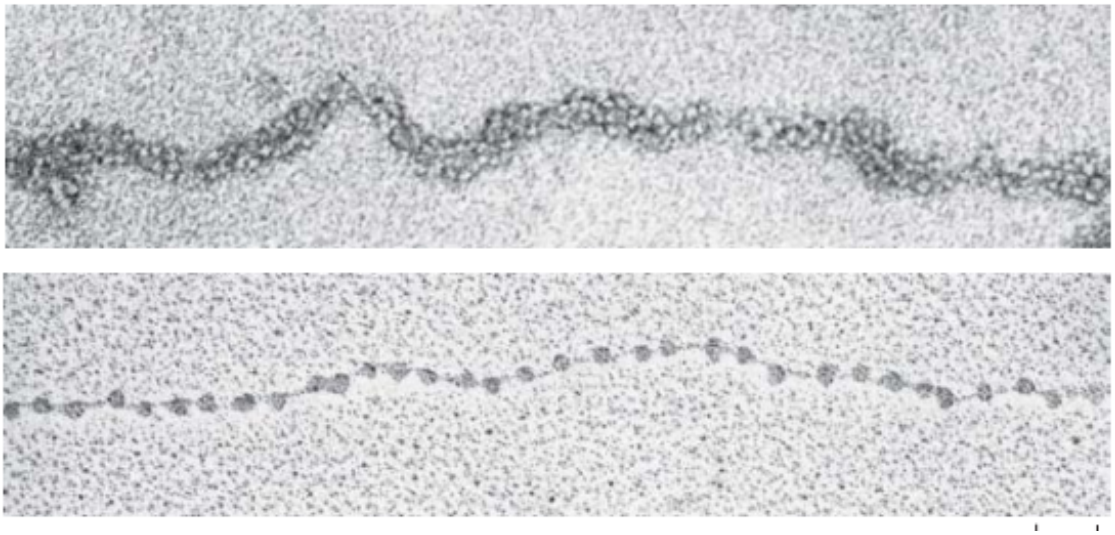
3.3.4 Considerações acerca dos resultados produzidos pela metodologia de teste proposta

As informações a respeito da qualidade como classificador das implementações paralelas e não paralelas, assim como as informações de performance computacional, tem como finalidade, uma vez que comparadas, tirar conclusões sobre dois aspectos. Primeiramente, a averiguação sobre a existência de indícios que a abordagem paralela proposta obtém resultados de classificação melhor que os segmentos que a compõem de forma separada, sendo assim a hipótese de que as características extraídas paralelamente melhorariam a classificação da rede seria corroborada. O segundo aspecto que é possível de ser avaliado com base nos dados obtidos dos testes, diz respeito à ponderação de até que ponto vale a pena adicionar segmentos paralelos na rede em relação à possível melhoria na rede como classificador, em relação a quantidade adicional de recursos computacionais de memória e do tempo necessário para treinar a rede.

3.4 Datasets usados nos experimentos

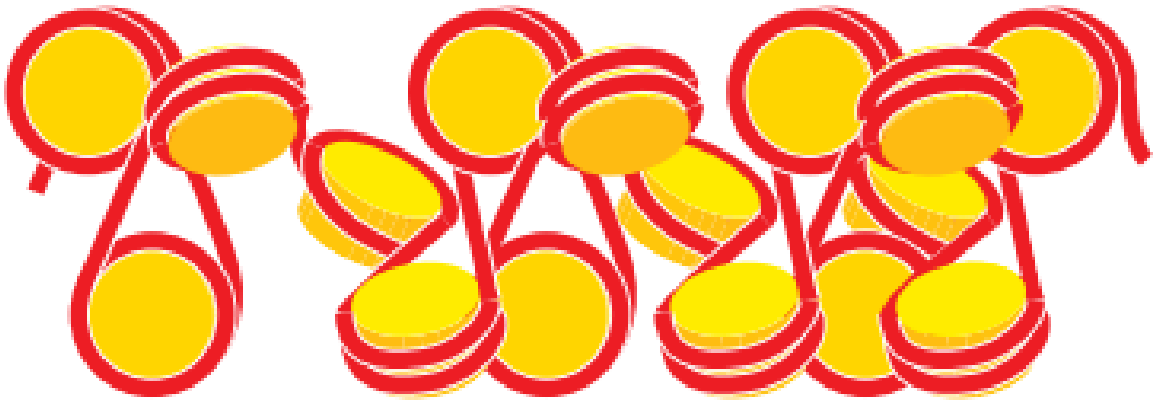
O DNA celular, encontra-se compactado na cromatina, esta, por sua vez é composta por DNA enovelado em agrupamentos de proteínas, tais agrupamentos são chamados de nucleossomos (Alberts, 2017). A figura 9 mostra a cromatina em seu estado compactado e em um estado artificialmente descompactado, onde é possível observar os nódulos referentes aos nucleossomos ligados por cadeias de DNA não enovelado, assim como na figura 10, onde é feita uma representação visual da maneira como parte da fita de DNA é compactada em volta dos nucleossomos e uma parte está solta e liga um nucleossomo ao outro. O núcleo proteico que serve como base do nucleossomo é formado por proteínas histonas de quatro tipos, h3, h4, h2a e h2b.

Figura 9 – Cromatina compactada e descompactada



Fonte: (Alberts, 2017)

Figura 10 – DNA envolto em nucleossomos



Fonte: (Alberts, 2017)

Os *datasets* utilizados nos experimentos são um subconnjunto dos *datasets* usados em (Giang *et al.*, 2016), tais *datasets* são originados do estudo feito em (Pokholok *et al.*, 2005) que diz respeito à mapear trechos de DNA envolto em nucleossomos, usando diferentes tipos de histonas que compõem o núcleo do nucleossomo como referência para o mapeamento.

Os *datasets* em questão são o H3 e H4ac, cada um composto por 14965 e 34096 amostras, respectivamente. As amostras de cada um destes é formada por uma sequência de 500 caracteres que representam os nucleotídeos de um dado trecho de DNA de levedura e duas classes podem ser associadas às mesmas, em caso positivo (representado por 1) a sequência possui regiões envoltas em núcleos de proteínas dos nuclessomos, caso negativo (representado por 0), não.

4 AVALIAÇÃO EXPERIMENTAL

4.1 Nomenclatura dos experimentos

Como mencionado, anteriormente, na explanação acerca da metodologia a ser utilizada, para cada um dos dois *datasets* escolhidos para serem explorados, foram realizados um total de 7 experimentos para extrair as métricas desejadas. Destes 7 experimentos, 4 são com arquiteturas não paralelas (ou sequenciais), com os diferentes tamanhos da janela de convolução da primeira camada convolucional – como também já mencionado anteriormente – e mais 3 experimentos com 2, 3 e 4 segmentos paralelos, combinando os segmentos sequenciais com diferentes tamanhos da janela de convolução da primeira camada convolucional.

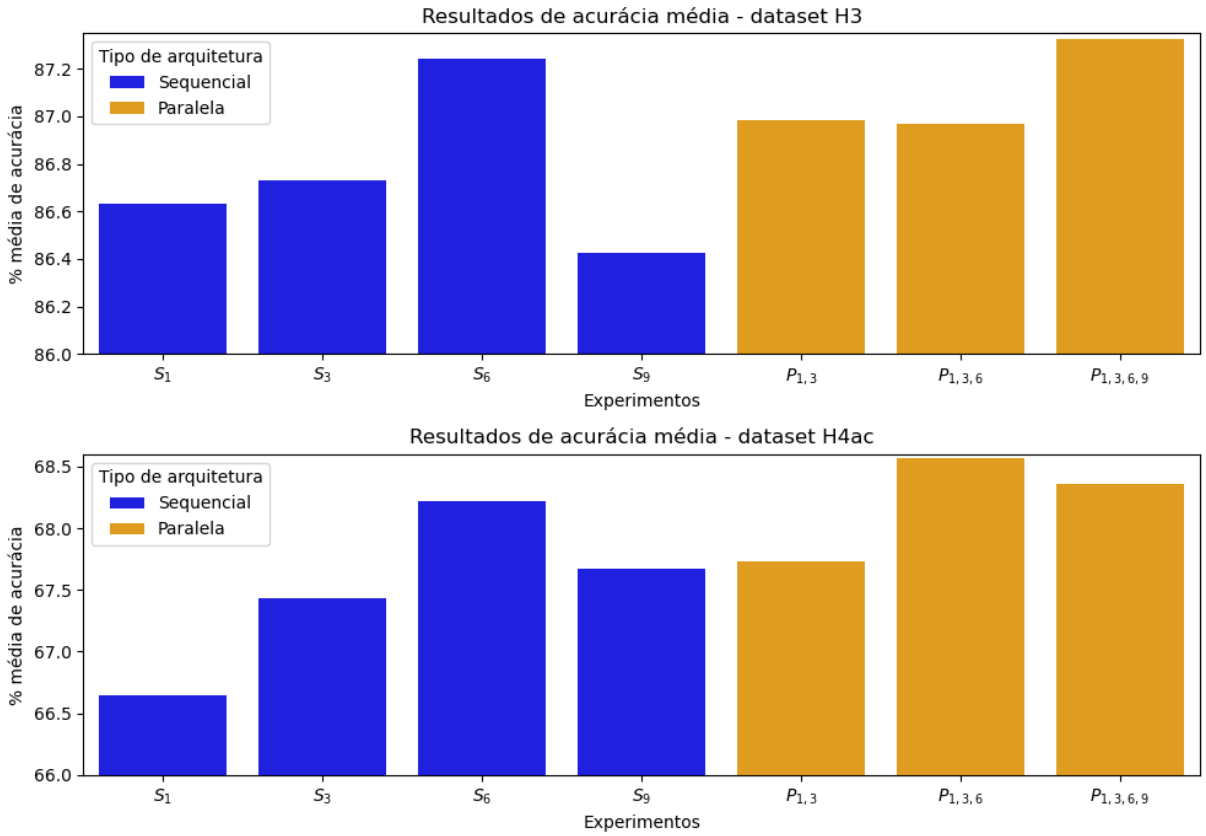
Para podermos diferenciar cada experimento ao nos referirmos aos mesmos na análise dos resultados obtidos, vamos designar os experimentos com arquiteturas não paralelas ou sequenciais pela letra "S" acompanhada de um subscrito, indicando o tamanho da janela de convolução em questão. Para as implementações paralelas identificaremos cada experimento pela letra "P" acompanhada por uma sequência de números subscritos indicando o tamanho da janela de convolução da primeira camada dos segmentos que compõem a mesma. Segue abaixo a listagem com os rótulos aplicados aos experimentos:

- $\mathbf{S}_1$: Experimentos feitos com arquitetura sequencial com janela de convolução na primeira camada de 1x16;
- $\mathbf{S}_3$: Experimentos feitos com arquitetura sequencial com janela de convolução na primeira camada de 3x16;
- $\mathbf{S}_6$: Experimentos feitos com arquitetura sequencial com janela de convolução na primeira camada de 6x16;
- $\mathbf{S}_9$: Experimentos feitos com arquitetura sequencial com janela de convolução na primeira camada de 9x16;
- $\mathbf{P}_{1,\,3}$: Experimentos feitos com arquitetura paralela com janelas de convolução na primeira camada de 1x16 e 3x16;
- $\mathbf{P}_{1,\,3,\,6}$: Experimentos feitos com arquitetura paralela com janelas de convolução na primeira camada de 1x16, 3x16 e 6x16;
- $\mathbf{P}_{1,\,3,\,6,\,9}$: Experimentos feitos com arquitetura paralela com janelas de convolução na primeira camada de 1x16, 3x16, 6x16 e 9x16.

4.2 Resultados obtidos em relação à qualidade do classificador

A métrica escolhida para avaliar o desempenho dos diferentes classificadores oriundos das diferentes arquiteturas testadas – correspondente a cada experimento – foi a acurácia. Apesar de tal métrica poder levar a resultados enganosos para conjuntos de dados com classes desbalanceadas, foram tomados o cuidados para que cada classe representasse aproximadamente 50% do conjunto de treino, mantendo o balanceamento. Na figura 11 e na tabela 1 e 2 os resultados obtidos são sumarizados.

Figura 11 – Acurácia média dos experimentos



Fonte: Autor

4.2.1 Resultados obtidos em relação a trabalhos anteriores com os mesmos datasets

Em relação ao trabalho que também usa os *datasets* que foram explorados aqui, no caso (Giang *et al.*, 2016), os melhores resultados de acurácia obtidos para os *datasets* H3 e H4ac, foram 87,33% e 68,57%, respectivamente, e os mesmos ficaram abaixo dos resultados obtidos no trabalho em questão para os mesmos *datasets*, que foram 88,99% e 77,40%, respectivamente. Tal diferença pode ter se dado pela profundidade da arquitetura usada em tal trabalho – podendo uma arquitetura mais profunda e com mais filtros ser capaz de obter melhores resultados. Outros possíveis motivos são a escolha dos componentes da

Tabela 1 – Peformance dos classificadores no dataset H3

Experimento	Acurácia (%)	F1-score	Precisão	Sensibilidade
$\mathbf{S}_1$	86,630	0,871	0,861	0,881
$\mathbf{S}_3$	86,734	0,871	0,874	0,868
$\mathbf{S}_6$	87,243	0,875	0,867	0,884
$\mathbf{S}_9$	86,424	0,866	0,866	0,866
$\mathbf{P}_{1,\ 3}$	86,986	0,873	0,867	0,879
$\mathbf{P}_{1,\ 3,\ 6}$	86,967	0,875	0,862	0,888
$\mathbf{P}_{1,\ 3,\ 6,\ 9}$	87,328	0,877	0,873	0,882

Fonte: Autor

Tabela 2 – Peformance dos classificadores no dataset H4ac

Experimento	Acurácia (%)	F1-score	Precisão	Sensibilidade
$\mathbf{S}_1$	66,648	0,668	0,667	0,672
$\mathbf{S}_3$	67,435	0,675	0,668	0,683
$\mathbf{S}_6$	68,215	0,681	0,685	0,679
$\mathbf{S}_9$	67,673	0,667	0,683	0,653
$\mathbf{P}_{1,\ 3}$	67,734	0,676	0,684	0,670
$\mathbf{P}_{1,\ 3,\ 6}$	68,572	0,686	0,688	0,684
$\mathbf{P}_{1,\ 3,\ 6,\ 9}$	68,357	0,678	0,687	0,669

Fonte: Autor

rede como funções de ativação, tamanho de *batch* e outros hiperparâmetros que possam afetar o desempenho da rede.

Porém, os resultados obtidos são coerentes com os resultados obtidos em (Giang *et al.*, 2016) no que diz respeito a disparidade dos resultados em relação aos dois *datasets*. O *dataset* H4ac, obteve como melhor média de acurácia um valor consideravelmente abaixo do que foi obtido para o *dataset* H3, indicando, talvez padrões mais complexos no que diz respeito a classificação para este *dataset*.

4.2.2 Comparação dos resultados obtidos pelas arquiteturas paralelas em relação as arquiteturas sequenciais

Tomando como base os resultados representados graficamente pela figura 11, podemos notar que as arquiteturas paralelas em geral obtém resultados melhores do que os segmentos isolados que as compõem, com exceção do experimento $\mathbf{P}_{1,\ 3,\ 6}$, que ficou abaixo do experimento $\mathbf{S}_6$, no *dataset* H3. Apesar da exceção mencionada, os resultados obtidos

mostraram uma tendência das arquiteturas paralelas de obterem melhores resultados na classificação. Além disso, para os dois *datasets*, os melhores resultados de média de acurácia obtidos na classificação, são de arquiteturas paralelas, $\mathbf{P}_{1,\,3,\,6,\,9}$ para o *dataset* H3 e $\mathbf{P}_{1,\,3,\,6}$ para o *dataset* H4ac.

Contudo, tais resultados trazem também indícios que a performance obtida não está unicamente associada ao número de segmentos paralelos das arquiteturas paralelas testadas. No *dataset* H3, a arquitetura do experimento $\mathbf{P}_{1,\,3}$ obteve um resultado ligeiramente melhor que o experimento com a arquitetura $\mathbf{P}_{1,\,3,\,6}$. No *dataset* H4ac, o melhor resultado obtido é o da arquitetura referente ao experimento $\mathbf{P}_{1,\,3,\,6}$, que por sua vez não é o experimento com o maior número de segmentos paralelos, para este *dataset*.

4.2.3 Resultados referentes a recursos computacionais e tempo de treinamento

As figuras 12 e 13 resumem os resultados da média de quantidade de memória alocada e média de tempo de execução por época para os *datasets* H3 e H4ac.

As médias de tempo de execução por época foram obtidas pelo relatório gerado pelo Keras durante o treinamento da rede e os resultados em questão são a média calculada para cada experimento. Para as implementações com arquitetura sequencial, não foi feita diferenciação referente às diferentes implementações e a média foi calculada levando em consideração os resultados das diferentes implementaçãoes sequênciais, tanto para o cálculo de média de memória alocada e tempo por época.

Para os dois *datasets*, se traçarmos uma curva e considerando a quantidade de memória alocada e tempo por época de treinamento como uma função do número segmentos paralelos, para cada experimento, observamos uma dinâmica similar nos dois *datasets*. O crescimento da quantidade de memória alocada e da quantidade de tempo por época parecem crescer de forma aproximadamente linear, a medida que aumentamos o número de segmentos paralelos. No caso do *dataset* H4ac, a quantidade de memória são maiores devido ao tamanho do *dataset* que é maior.

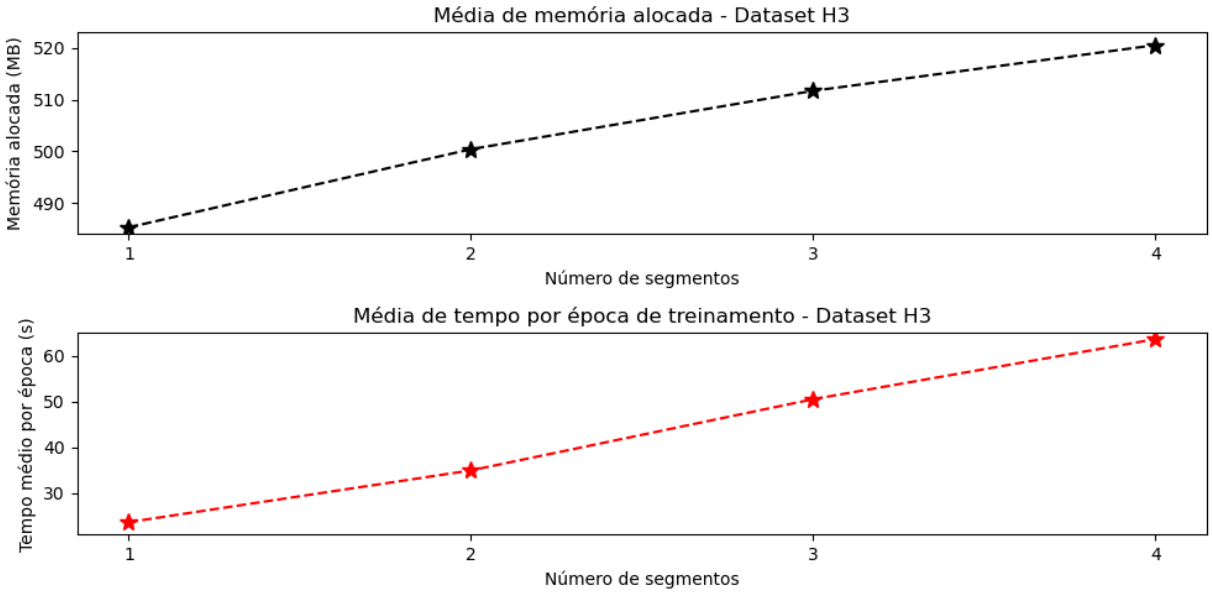
Na tabela 3, temos a listagem dos valores de médias de memória em megabytes e da média de tempo por época de treinamento em segundos e na figura 12 temos a representação visual de tais valores para o *dataset* H3.

Tabela 3 – Desempenho em função do número de segmentos para o dataset H3

Número de segmentos paralelos	Memória alocada (MB)	Tempo médio por época (s)
1	485,3	23,7
2	509,4	35,0
3	511,7	50,5
4	520,6	63,6

Fonte: Autor

Figura 12 – Médias de memória alocada em função do número de segmentos paralelos para o dataset H3



Fonte: Autor

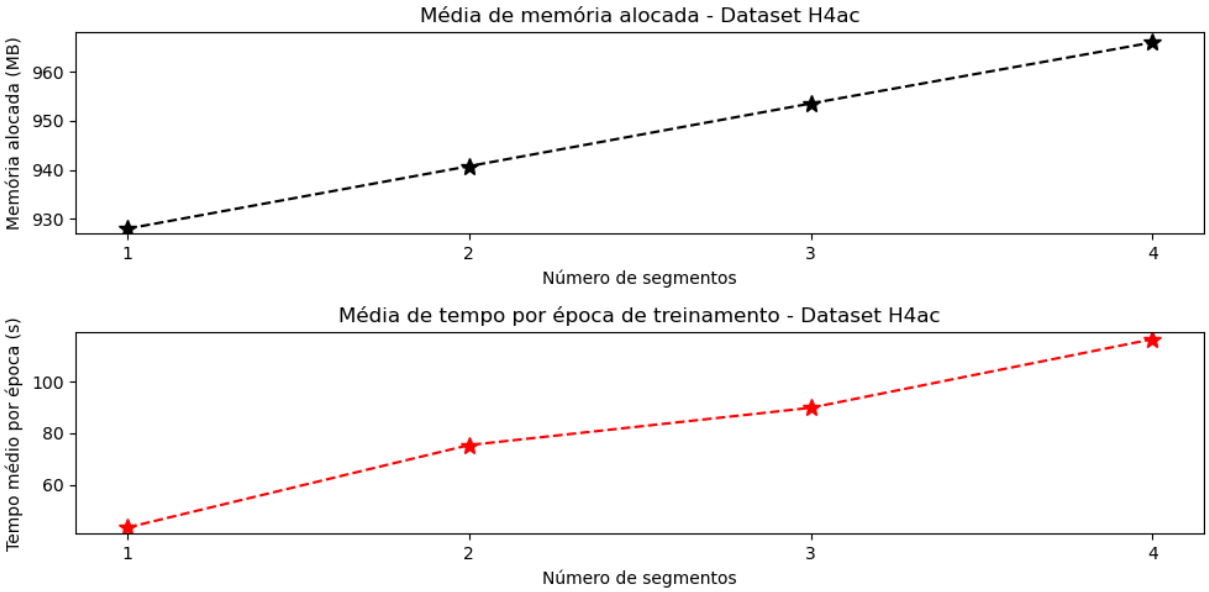
Abaixo, na tabela 4, e na figura 13, temos os valores de memória e tempo médio por época de treinamento para o *dataset* H4ac.

Tabela 4 – Desempenho em função do número de segmentos para o dataset H4ac

Número de segmentos paralelos	Memória alocada (MB)	Tempo médio por época (s)
1	928,0	43,3
2	940,7	75,3
3	953,5	89,8
4	966,0	116,4

Fonte: Autor

Figura 13 – Médias de memória alocada em função do número de segmentos paralelos para o dataset H4ac



Fonte: Autor

5 CONCLUSÕES

No que diz respeito à investigação feita neste trabalho, em relação às possíveis melhorias na identificação de padrões em sequências de DNA – usando a arquitetura paralela proposta, com diferentes níveis de paralelismo – os experimentos feitos, para os dois *datasets* escolhidos, deram indícios de que as implementações paralelas obtiveram resultados de generalização melhores em relação às implementações sequênciais, se levarmos em consideração o número de experimentos com arquiteturas paralelas que obtiveram resultados de acurácia maior que os segmentos separados que as compõem, para os dois *datasets*. Os melhores resultados de acurácia obtidos, de forma geral, também são oriundos de implementações paralelas, nos dois *datasets*.

No que diz respeito ao uso de recursos computacionais, os resultados medidos, indicaram, para ambos os *datasets*, uma tendência de crescimento próxima do linear tanto para alocação de memória durante a fase de treino quanto na quantidade de tempo média por época de treinamento. Apesar de os experimentos terem mostrado sinais de que a arquitetura paralela proposta se mostrou melhor que as arquiteturas sequênciais – na mairia das vezes –, não foi possível estabelecer uma correlação direta, entre a acurácia obtida pelas arquiteturas paralelas e o número de segmentos paralelos utilizados. Sendo assim, os experimentos não dão indícios sólidos de que vale a pena adicionar cada vez mais segmentos paralelos para obtenção de resultados melhores de generalização. A quantidade ótima de segmentos paralelos só foi identificada, para cada *dataset*, com base em experimentação e para o caso do dataset H4ac, por exemplo, usar 4 segmentos em vez de 3, só aumentaria a quantidade de memória usada no processo de treino e o tempo de treinamento do modelo.

REFERÊNCIAS

ALBERTS, B. **Molecular Biology of the Cell**. W.W. Norton, 2017. ISBN 9781317563754. Disponível em: https://books.google.com.br/books?id=jK6UBQAAQBAJ.

CRUZ, M. H. P. da *et al.* TERL: classification of transposable elements by convolutional neural networks. **Briefings in Bioinformatics**, v. 22, n. 3, p. bbaa185, 09 2020. ISSN 1477-4054. Disponível em: https://doi.org/10.1093/bib/bbaa185.

GÉRON, A. **Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems**. O'Reilly Media, Incorporated, 2019. ISBN 9781492032649. Disponível em: https://books.google.com.br/books?id=OCS1twEACAAJ.

GIANG, N. N. *et al.* Dna sequence classification by convolutional neural network. **Journal of Biomedical Science and Engineering**, v. 09, p. 280–286, 01 2016.

HAYKIN, S. S. **Neural networks and learning machines**. Third. Upper Saddle River, NJ: Pearson Education, 2009.

HEBB, D. **The Organization of Behavior: A Neuropsychological Theory**. Wiley, 1949. (A Wiley book in clinical psychology). ISBN 9780471367277. Disponível em: https://books.google.com.br/books?id=dZ0eDiLTwuEC.

HUBEL, D.; WIESEL, T. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. **The Journal of physiology**, v. 160, p. 106—154, January 1962. ISSN 0022-3751. Disponível em: https://www.ncbi.nlm.nih.gov/pmc/articles/pmid/14449617/?tool=EBI.

JOHNSON, R.; ZHANG, T. Effective use of word order for text categorization with convolutional neural networks. 12 2014.

KIM, Y. Convolutional neural networks for sentence classification. *In*: MOSCHITTI, A.; PANG, B.; DAELEMANS, W. (ed.). **Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)**. Doha, Qatar: Association for Computational Linguistics, 2014. p. 1746–1751. Disponível em: https://aclanthology.org/D14-1181.

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

MAEDA-GUTIéRREZ, V. *et al.* Comparison of convolutional neural network architectures for classification of tomato plant diseases. **Applied Sciences**, v. 10, n. 4, 2020. ISSN 2076-3417. Disponível em: https://www.mdpi.com/2076-3417/10/4/1245.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **The bulletin of mathematical biophysics**, Springer, v. 5, p. 115–133, 1943.

MINSKY, M.; PAPERT, S. **Perceptrons**. Cambridge, MA: MIT Press, 1969.

PATTERSON, J.; GIBSON, A. **Deep Learning: A Practitioner's Approach**. O'Reilly Media, 2017. ISBN 9781491914236. Disponível em: https://books.google.com.br/books?id=qrcuDwAAQBAJ.

POKHOLOK, D. K. *et al.* Genome-wide map of nucleosome acetylation and methylation in yeast. **Cell**, v. 122, n. 4, p. 517–527, 2005. ISSN 0092-8674. Disponível em: https://www.sciencedirect.com/science/article/pii/S0092867405006458.

RODOLA, G. **psutil: Cross-platform library for process and system monitoring in Python**. 2021. https://github.com/giampaolo/psutil.

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. **Psychological review**, v. 65 6, p. 386–408, 1958. Disponível em: https://api.semanticscholar.org/CorpusID:12781225.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning internal representations by error propagation, parallel distributed processing, explorations in the microstructure of cognition, ed. de rumelhart and j. mcclelland. vol. 1. 1986. **Biometrika**, v. 71, p. 599–607, 1986.

ZENG, H. *et al.* Convolutional neural network architectures for predicting DNA–protein binding. **Bioinformatics**, v. 32, n. 12, p. i121–i127, 06 2016. ISSN 1367-4803. Disponível em: https://doi.org/10.1093/bioinformatics/btw255.