

**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE
SANTA CATARINA - CÂMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO SUPERIOR DE ENGENHARIA ELETRÔNICA**

ANA CLÁUDIA BANDERCHUK

TÍTULO DO TCC

Florianópolis, 2021

**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE
SANTA CATARINA - CÂMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO SUPERIOR DE ENGENHARIA ELETRÔNICA**

ANA CLÁUDIA BANDERCHUK

TÍTULO DO TCC

Trabalho de conclusão de curso submetido
ao Instituto Federal de Educação, Ciência
e Tecnologia de Santa Catarina como parte
dos requisitos para obtenção do título de
Engenheiro Eletrônico

Orientador:
Prof. Dr. Fernando Santana Pacheco

Florianópolis, 2021

TÍTULO DO TCC

ANA CLÁUDIA BANDERCHUK

Este Trabalho foi julgado adequado para obtenção do Título de Engenheiro Eletrônico em abril de 2021 e aprovado na sua forma final pela banca examinadora do Curso de Engenharia Eletrônica do instituto Federal de Educação Ciência, e Tecnologia de Santa Catarina.

Florianópolis, 12 de março, 2021.

Banca Examinadora:

Fernando Santana Pacheco, Dr.

RESUMO

Este trabalho apresenta

Palavras-chave: um. dois.

ABSTRACT

This papper presents

Keywords:

LISTA DE ILUSTRAÇÕES

| | |
|---|----|
| Figura 1 – Exemplos de problemas e seus níveis de complexidade para computadores ou seres humanos resolvê-los. | 8 |
| Figura 2 – Diagrama de Venn da Inteligência Artificial e suas áreas de estudo. | 9 |
| Figura 3 – Fluxograma que diferencia as áreas de estudo da Inteligência Artificial e suas etapas de processamento de dados. | 10 |
| Figura 4 – Ilustração das camadas de um modelo de Aprendizado Profundo. | 11 |
| Figura 5 – Comparação entre os modelos Aprendizado de Máquina e Aprendizado Profundo. | 11 |
| Figura 6 – Estrutura básica de uma Rede Neural com duas camadas. | 12 |
| Figura 7 – Diagrama de um Neurônio de uma Rede Neural. | 13 |
| Figura 8 – Exemplos de funções de ativação de um neurônio para uma Rede Neural. | 14 |
| Figura 9 – Estapas da atualização dos pesos de um nó em uma Rede Neural. | 16 |
| Figura 10 – Método do gradiente para uma variável. | 16 |
| Figura 11 – Matriz de Confusão para classificação binária. | 18 |
| Figura 12 – Ilustração das medidas de Precisão e Sensibilidade. | 19 |
| Figura 13 – Curva da Precisão em função da Sensibilidade para quatro classes distintas. | 19 |
| Figura 14 – Componentes de uma Rede Neural Convolucional típica. | 21 |
| Figura 15 – Exemplo de convolução entre dois vetores bidimensionais. | 22 |
| Figura 16 – Operações de convolução em uma imagem. | 23 |
| Figura 17 – Exemplos de funções utilizadas para operação de <i>pooling</i> | 24 |
| Figura 18 – Arquitetura da Rede Neural YOLO. | 25 |
| Figura 19 – Comparação da precisão média em função do processamento em <i>frames</i> por segundo para diferentes redes neurais utilizadas em detecção de objetos. | 25 |
| Figura 20 – Comparação da precisão média em função do processamento em <i>frames</i> por segundo para diferentes redes neurais utilizadas em detecção de objetos. | 26 |
| Figura 21 – Diferença entre as tarefas de classificação e detecção de objetos em uma imagem. | 28 |

SUMÁRIO

| | | |
|----------|---|-----------|
| 1 | INTRODUÇÃO | 7 |
| 1.1 | Justificativa | 7 |
| 1.2 | Descrição do problema | 7 |
| 1.3 | Objetivo geral | 7 |
| 1.4 | Objetivos específicos | 7 |
| 2 | FUNDAMENTAÇÃO TEÓRICA | 8 |
| 2.1 | Inteligência Artificial | 8 |
| 2.2 | Redes Neurais | 11 |
| 2.2.1 | Estrutura de Uma Rede Neural | 12 |
| 2.2.2 | Funções de Ativação | 13 |
| 2.2.3 | Treinamento de Uma Rede Neural | 15 |
| 2.2.4 | Métricas de Avaliação | 17 |
| 2.2.4.1 | Matriz de Confusão | 17 |
| 2.2.4.2 | Acurácia | 17 |
| 2.2.4.3 | Precisão e Sensibilidade | 18 |
| 2.2.4.4 | Precisão Média | 18 |
| 2.2.4.5 | <i>F-Score</i> | 20 |
| 2.2.5 | Redes Neurais Convolucionais | 20 |
| 2.2.5.1 | Operação de Convolução | 20 |
| 2.2.5.2 | Operação de <i>Pooling</i> | 22 |
| 2.2.5.3 | <i>You Only Look Once</i> | 23 |
| 2.3 | Classificação e Detecção de Objetos | 27 |
| 2.3.1 | IoU | 28 |
| 2.3.2 | <i>Datasets</i> | 28 |
| 2.3.3 | Defeitos em Placas de Circuito Impresso | 28 |
| 2.4 | Frameworks e Bibliotecas | 28 |
| 2.4.1 | OpenCV | 28 |
| 2.4.2 | Darknet | 28 |
| 2.4.3 | Flask | 28 |
| | REFERÊNCIAS | 29 |
| | APÊNDICES | 33 |
| | APÊNDICE A – NOTAÇÕES DE REDES NEURAIS | 34 |

1 INTRODUÇÃO

Placas de circuito impresso são de longe o método mais utilizado para o projeto de eletrônicos modernos. Elas consistem em um sanduíche de uma ou mais camadas de cobre intercaladas com uma ou mais camadas de material isolante (ZUM-BAHLEN, 2008) e servem de suporte para os componentes eletrônicos responsáveis pelo funcionamento de um circuito eletrônico.

À medida em novas tecnologias são desenvolvidas, as placas de circuito impresso estão se tornando cada vez mais sofisticadas e delicadas (HU; WANG, 2020), de modo que a detecção de incertezas, tolerâncias, defeitos e erros de posição relativa associados ao processo de fabricação (LETA; FELICIANO; MARTINS, 2008) deve ser feita com mais cautela a fim de garantir o funcionamento do produto final.

Dessa forma, automatizar a inspeção de defeitos se tornou essencial para aprimorar a qualidade do processo de fabricação, já que técnicas de medição por visão computacional apresentam melhor regularidade, precisão e repetibilidade quando comparada a inspeção humana que, além da imprecisão e não-repetibilidade, está sujeita a subjetividade, fadiga e lentidão e está, ainda, associada a um alto custo (LETA; FELICIANO; MARTINS, 2008).

A inspeção ótica automatizada (AOI) vem sido amplamente utilizada para detectar defeitos durante o processo de fabricação de uma PCB (CHIN; HARLOW, 1982). Conforme a evolução dessa tecnologia, três principais métodos de detecção tem se destacado: métodos comparativos, não-referenciais e híbridos (WU; WANG; LIU, 1996). O método comparativo, mais utilizado entre eles, está suscetível à interferência da iluminação e ruídos externos, além de necessitar mecanismos de alinhamento precisos para realização da comparação (HU; WANG, 2020).

FALTA TERMINAR A INTRODUÇÃO TESTE

1.1 Justificativa

1.2 Descrição do problema

1.3 Objetivo geral

1.4 Objetivos específicos

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Inteligência Artificial

Apesar de ter chamado atenção nos últimos anos com a quantidade enorme de dados adquiridos e processados por grandes corporações como Google, Facebook, Amazon, e Apple (RUSSELL; MOSKOWITZ; JALAIAN, 2020), o termo ‘Inteligência Artificial’ não é tão atual assim. Ele foi utilizado pela primeira vez em 1955 por John McCarthy (ABRAHAM et al., 2021), que conduziu no ano seguinte um workshop cuja premissa central da proposta considera que o comportamento humano inteligente consiste em processos que podem ser formalizados e reproduzidos por uma máquina (DICK, 2019).

Um dos objetivos da Inteligência Artificial, de acordo com Mitchell, Michalski e Carbonell (2013), é fazer com que computadores realizem tarefas mais inteligentes de forma com que não haja necessidade dos seres humanos executá-las. Entretanto, um dos grandes desafios da área de IA atualmente é a execução de atividades consideradas simples e corriqueiras para pessoas, como reconhecimento de objetos e fala (GOODFELLOW; BENGIO; COURVILLE, 2016). Para ilustrar isso, a tabela da Figura 1 apresenta a diferença entre o nível de dificuldade que um computador ou ser humano tem para resolver determinado problema.

Figura 1 – Exemplos de problemas e seus níveis de complexidade para computadores ou seres humanos resolvê-los.

| Problema | Computador | Humano |
|---|------------|---------|
| Multiplicar milhares de números grandes rapidamente | FÁCIL | DIFÍCIL |
| Identificar rostos em uma foto de uma multidão de pessoas | DIFÍCIL | FÁCIL |

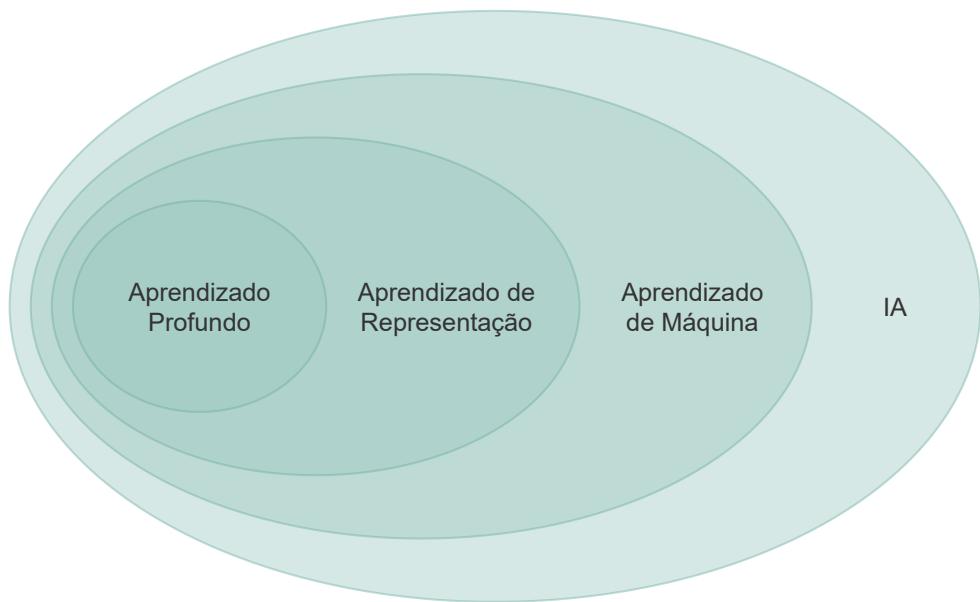
Fonte: Adaptado de Rashid (2016).

A Inteligência Artificial engloba a área de estudo do Aprendizado de Máquina, que por sua vez englobam as áreas do Aprendizado de Representação e Aprendizado Profundo, conforme o diagrama de Venn apresentado na Figura 2.

Para evidenciar as diferenças entre essas áreas de estudo, em comparação com simples Sistemas Baseados em Regras, Goodfellow, Bengio e Courville (2016) apresenta um fluxograma com as etapas de processamento entre a entrada e a saída de dados para cada uma delas, presente na Figura 3.

Sistemas Baseados em Regras não possuem componentes capazes de

Figura 2 – Diagrama de Venn da Inteligência Artificial e suas áreas de estudo.

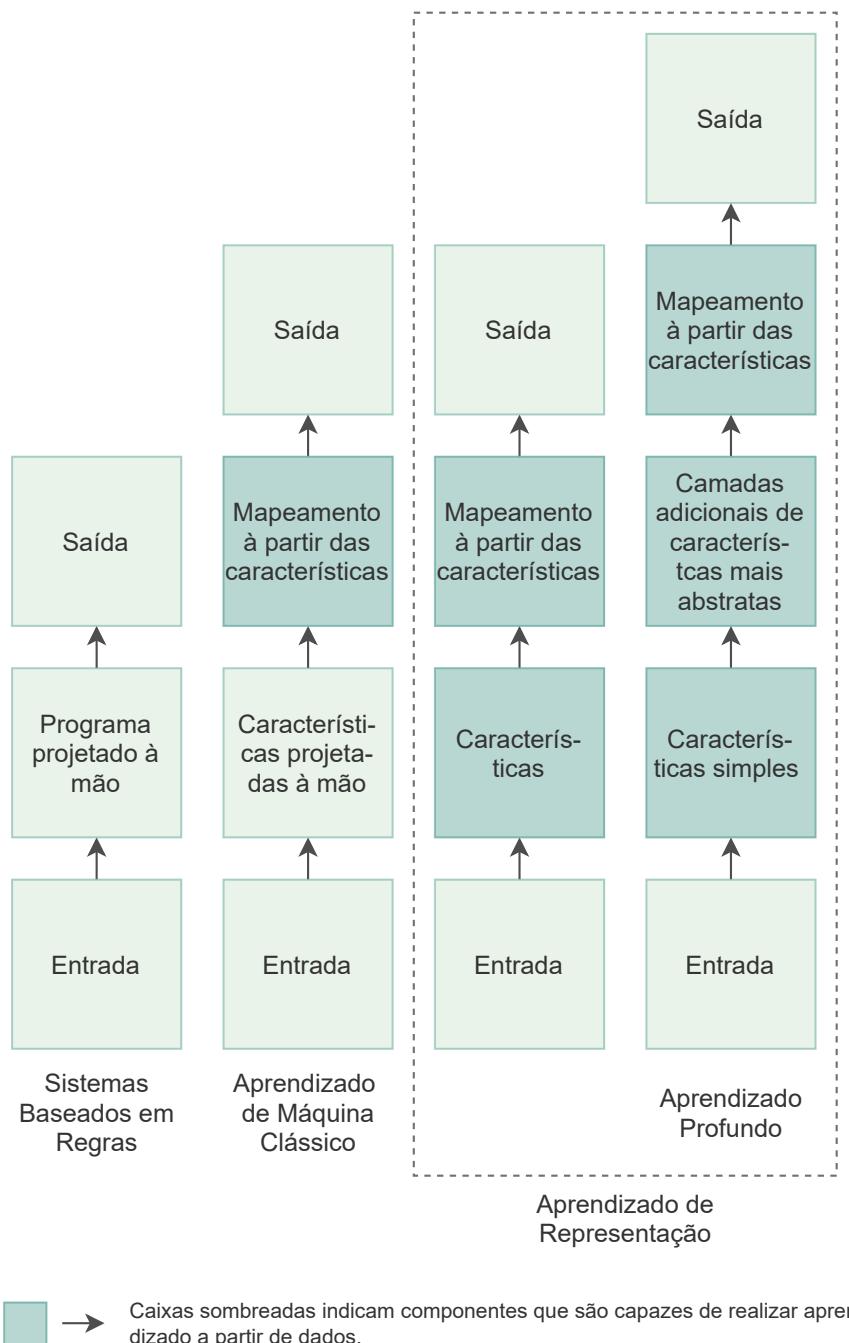


Fonte: Adaptado de Goodfellow, Bengio e Courville (2016).

realizar qualquer aprendizado a partir e dados (GOODFELLOW; BENGIO; COURVILLE, 2016) e são utilizados para resolução de problemas e/ou execução de tarefas que podem ser descritos por uma lista de regras formais, como por exemplo jogar Xadrez. Ao contrários desses sistemas, o Aprendizado de Máquina (do inglês *Machine Learning*) utiliza algoritmos computacionais para transformar características reunidas empiricamente em modelos utilizáveis (EDGAR; MANZ, 2017) possuindo a capacidade “de se aprimorar [...], aprendendo novos conhecimentos ao invés de serem programado com eles” (WOOLF, 2009).

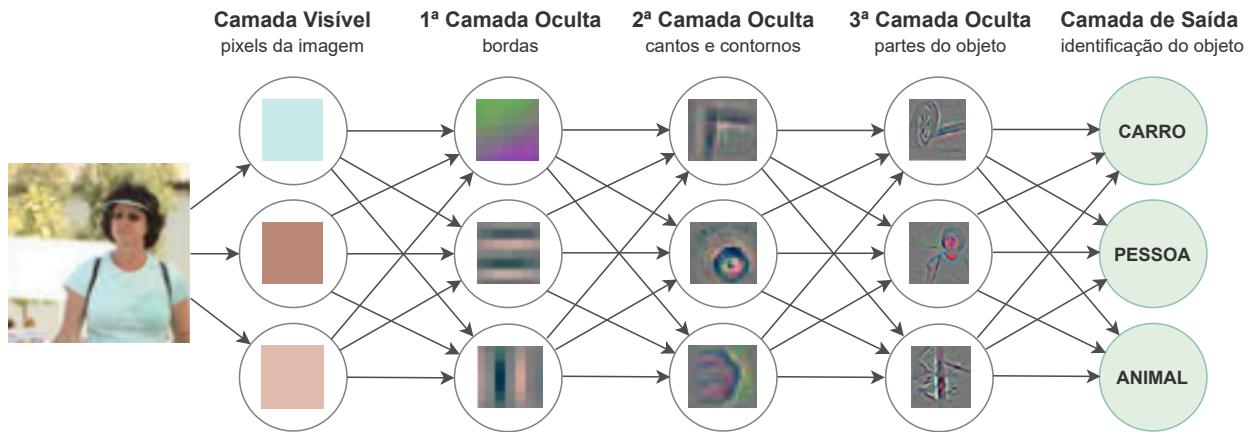
No Aprendizado de Representação (do inglês *Feature Learning* ou *Representation Learning*) entretanto, não há necessidade de mapear manualmente essas características (GOODFELLOW; BENGIO; COURVILLE, 2016). Ou seja, por conta própria e de forma abstrata, os algoritmos são capazes de extrair as características importantes para a construção dos modelos utilizando redes neurais (ROBINS, 2020) (LESORT et al., 2018). O Aprendizado Profundo, por sua vez, resolve a dificuldade que o Aprendizado de Representação possui de extrair características abstratas de alto nível, tais como sotaques de um locutor (GOODFELLOW; BENGIO; COURVILLE, 2016). Segundo Mao et al. (2019), o Aprendizado Profundo “imita a função que o cérebro humano possui de interpretar dados usando redes neurais de várias camadas”. Um exemplo dessas variadas camadas é apresentado na Figura 4, onde características distintas são extraídas por cada camada. Uma comparação com o modelo de

Figura 3 – Fluxograma que diferencia as áreas de estudo da Inteligência Artificial e suas etapas de processamento de dados.

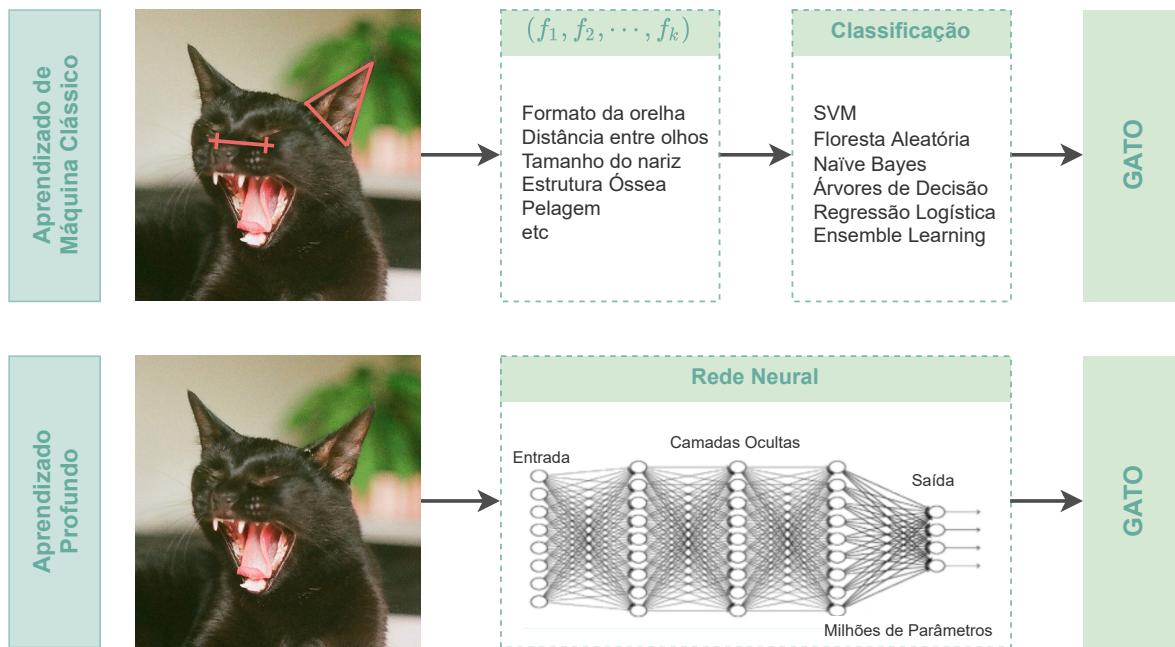


Fonte: Adaptado de Goodfellow, Bengio e Courville (2016).

Aprendizado de Máquina é ilustrado na Figura 5.

Figura 4 – Ilustração das camadas de um modelo de Aprendizado Profundo.

Fonte: Adaptado de Goodfellow, Bengio e Courville (2016).

Figura 5 – Comparação entre os modelos Aprendizado de Máquina e Aprendizado Profundo.

Fonte: Adaptado de Robins (2020).

2.2 Redes Neurais

Ao contrário da abordagem convencional de programação, onde dizemos a um computador o que deve ser feito ao dividir um problema em pequenas tarefas para que ele execute, uma Rede Neural (do inglês *Neural Network*) utiliza dados

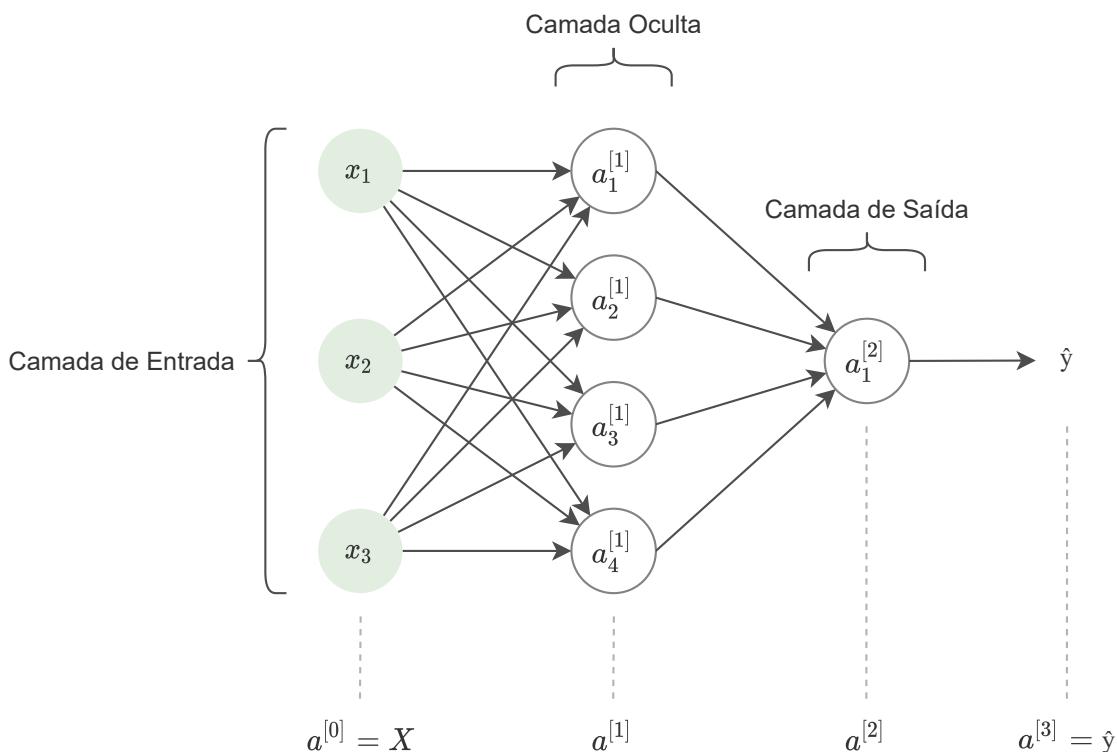
observacionais para aprender como resolver o problema (NIELSEN, 2018).

Walczak e Cerpa (2003) define Redes Neurais Artificiais como modelos que “simulam a atividade elétrica do cérebro e do sistema nervoso”. Porém enquanto alguns tipos de redes neurais tem sido utilizadas para entender o funcionamento do cérebro, na perspectiva de Aprendizado Profundo elas não são projetadas para serem modelos realistas da função biológica (GOODFELLOW; BENGIO; COURVILLE, 2016).

2.2.1 Estrutura de Uma Rede Neural

Uma Rede Neural é composta por camadas de nós, conhecidos como neurônios, e conexões que interligam as saídas e entradas desses nós. A estrutura básica de uma Rede Neural é ilustrada na Figura 6, onde a primeira camada de neurônios é a camada de entrada, a última camada é a camada de saída e as camadas entre elas são chamadas de camadas ocultas. Na Figura 6, vetor X contém os valores de entrada, os vetores $a^{[l]}$ representam as funções de ativação referentes à l -ésima camada, e \hat{y} é o vetor de saída com os valores preditos. As notações utilizadas nesse capítulo respeitam as propostas por Ng (2020) presentes no Apêndice A.

Figura 6 – Estrutura básica de uma Rede Neural com duas camadas.

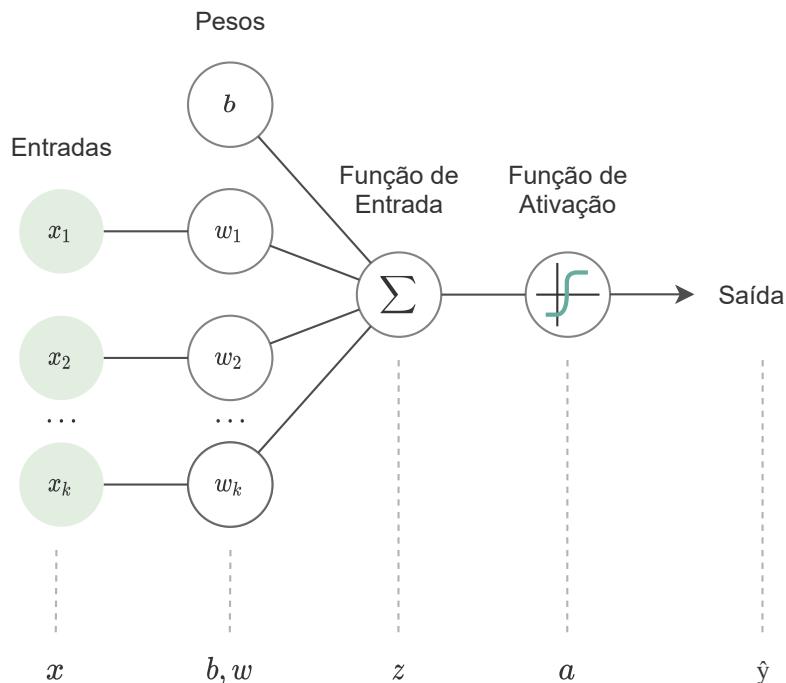


Fonte: Elaboração própria (2021).

São nos neurônios que ocorrem as computações. A Figura 7 mostra um

diagrama de um nó de uma Rede Neural, onde os pesos representados por b e w_k são responsáveis por atribuir significância às entradas com relação à tarefa que o algoritmo está tentando aprender (NICHOLSON, 2020). Esses produtos são então somados e passam por uma função de ativação.

Figura 7 – Diagrama de um Neurônio de uma Rede Neural.



Fonte: Adaptado de Nicholson (2020).

2.2.2 Funções de Ativação

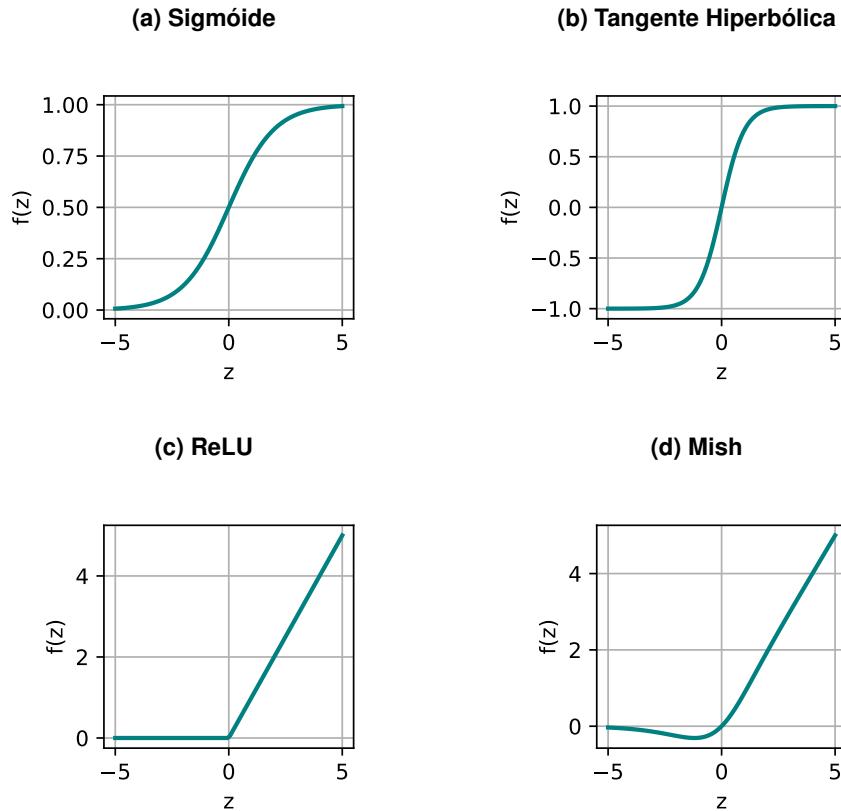
As funções de ativação desempenham um papel crucial na dinâmica de desempenho e treinamento em redes neurais (MISRA, 2019), determinando se um sinal deve progredir e em que medida ele deve progredir através da rede (NICHOLSON, 2020). Alguns exemplos de função de ativação estão na Figura 8.

A função Sigmóide da Figura 8a, definida por Sharma (2017) pela Equação 2.1, é geralmente utilizada em classificações binárias, pois o resultado está sempre no intervalo entre zero e um (NG, 2020).

$$f(z) = \sigma(z) = \frac{1}{1 + e^{-z}} \quad (2.1)$$

Para a otimização dos pesos durante o treinamento de uma rede neural, em geral é necessário calcular o gradiente das funções de ativação. Por isso, de acordo

Figura 8 – Exemplos de funções de ativação de um neurônio para uma Rede Neural.



Fonte: Elaboração própria (2021).

com Sharma (2017), o uso da tangente hiperbólica como função de ativação é preferível pois tem gradientes que não estão restritos a variar em uma certa direção e são mais acentuados quando comparado à Sigmóide. A definição da tangente hiperbólica da Figura 8b está na Equação 2.2.

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \quad (2.2)$$

Porém, para valores muito grandes ou muito pequenos de z , a derivada resultante, utilizada no gradiente, tende a ser nula tanto para a sigmóide quanto para a tangente hiperbólica (NG, 2020) o que geralmente causa lentidão no treinamento (MISRA, 2019). Para que isso não ocorra, utiliza-se a função ReLU, do inglês *Rectified Linear Unit*, plotada na Figura 8c e definida na Equação 2.3. A ReLU é a função de ativação mais utilizada (NG, 2020) e é a função padrão recomendada para uso com a maioria das redes neurais (GOODFELLOW; BENGIO; COURVILLE, 2016).

$$f(z) = \text{ReLU}(z) = \max(0, z) \quad (2.3)$$

Com a evolução das Redes Neurais na última década, diversas funções de ativação vem sendo propostas com o objetivo de melhorar o desempenho do treinamento. Um exemplo delas é a Mish (Figura 8d), proposta por Misra (2019) cuja função é definida pela Equação 2.4.

$$f(z) = z \cdot \tanh(\ln(1 + e^z)) \quad (2.4)$$

Segundo Misra (2019), ao contrário da ReLU, a função Mish é continuamente diferenciável, evitando efeitos colaterais indesejados quando utiliza-se a otimização baseada no método do gradiente.

2.2.3 Treinamento de Uma Rede Neural

Para o treinamento de uma Rede Neural, utiliza-se um conjunto de exemplos de treinamento $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$, onde $x^{(i)}, y^{(i)}$ são, respectivamente, a entrada e a saída real do i -ésimo exemplo de treinamento de um conjunto contendo m exemplos de treinamento (NG, 2020). Para cada um desses exemplos, o objetivo é fazer o cálculo dos pesos w e b de forma que a saída estimada $\hat{y}^{(i)}$ seja próxima da saída real $y^{(i)}$.

O peso b é chamado de *bias* e não é multiplicado a nenhum valor da entrada para caso todos os valores de x sejam nulos, a saída $\hat{y}^{(i)}$ seja diferente de zero. O cálculo desses pesos é feito de forma iterativa e geralmente são iniciados de forma aleatória.

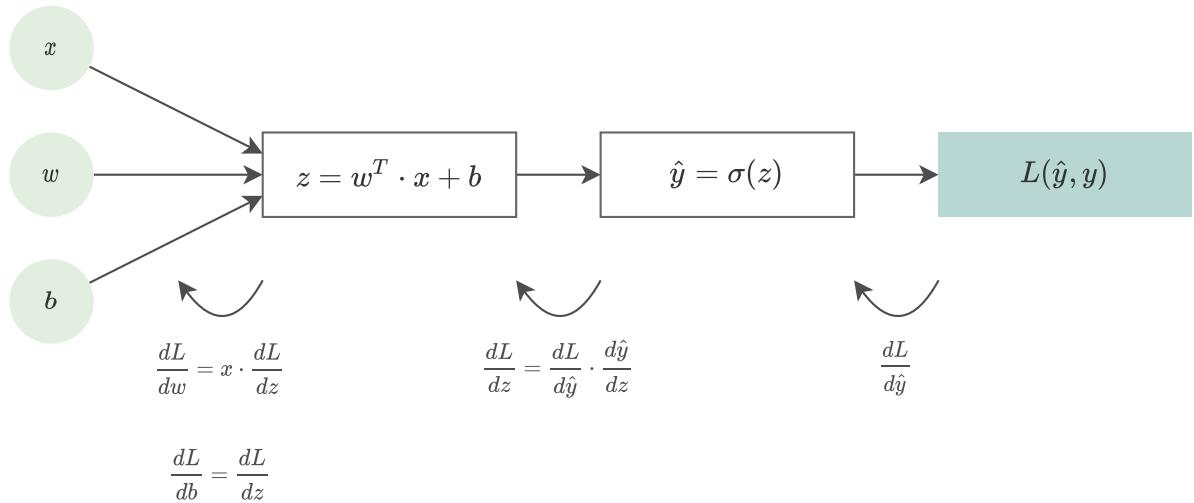
O processo de cálculo considerando um nó de uma rede neural está ilustrado na Figura 9, onde a função de ativação utilizada é a sigmoide e todos os valores calculados e os de entrada são representados na forma vetorial.

A primeira etapa a ser computada após a passagem das entradas pela função de ativação é função de perda, que mede quão boa é a saída prevista \hat{y} em comparação aos valores verdadeiros de y para apenas um conjunto de treinamento (NG, 2020).

Um exemplo de função de perda que compara \hat{y} com y pode ser definida pela Equação 2.5 (NG, 2020), onde o objetivo do treinamento é obter $L(y, \hat{y}) = 0$.

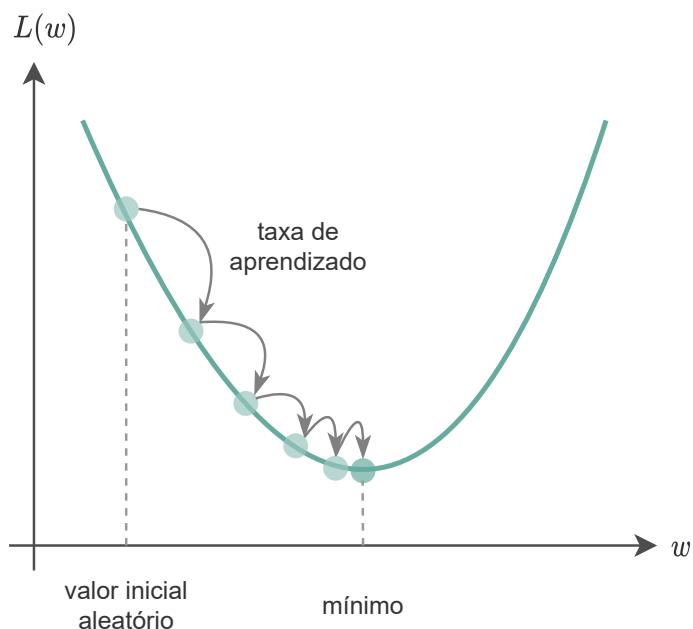
$$L(y, \hat{y}) = -(y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y})) \quad (2.5)$$

Dessa forma, como $\hat{y} = f(w, b)$, é necessário encontrar valores para os pesos w e b que minimizem a função de perda $L(y, \hat{y})$. Para isso, utiliza-se o método do gradiente, um algoritmo de primeira ordem iterativo utilizado em otimização para encontrar o

Figura 9 – Estapas da atualização dos pesos de um nó em uma Rede Neural.

Fonte: Adaptado de Ng (2020).

mínimo local de uma função (YAN, 2021), exemplificado na Figura 10 para uma função de perda considerando apenas a variável w .

Figura 10 – Método do gradiente para uma variável.

Fonte: Adaptado de Sauer (2020).

Após serem computadas as derivadas da Figura 9, os pesos w e b são atualizados conforme a Equação 2.6 e a Equação 2.7 onde α representa uma taxa de aprendizado (NG, 2020).

$$w = w - \alpha \cdot \frac{dL}{dw} \quad (2.6)$$

$$b = b - \alpha \cdot \frac{dL}{db} \quad (2.7)$$

A atualização desses pesos é feita até que a função de perda resulte valores muito próximos de zero ou quando o erro está dentro de uma faixa aceitável de acordo com a métrica de avaliação utilizada.

2.2.4 Métricas de Avaliação

Determinar os objetivos de treinamento em uma rede neural em termos de qual métrica de erro será utilizada é uma etapa necessária, já que para a maioria das aplicações, é impossível obter erro zero absoluto. (GOODFELLOW; BENGIO; COURVILLE, 2016).

2.2.4.1 Matriz de Confusão

A Matriz de Confusão é uma técnica muito popular usada tanto em problemas de classificação binária quanto problemas de classificação com diversas classes, representando a contagem dos resultados da predição feitos pela rede neural e dos valores verdadeiros (KULKARNI; CHONG; BATARSEH, 2020). Um exemplo de matriz de confusão utilizada em classificações binárias está na Figura 11.

2.2.4.2 Acurácia

Uma das métricas mais comuns utilizadas durante a classificação é a Acurácia (KULKARNI; CHONG; BATARSEH, 2020), que pode ser calculada a partir da matriz de confusão utilizando a Equação 2.8, definida pela razão entre o número de previsões corretas e o número total de previsões.

$$\text{Acurácia} = \frac{VN + VP}{VN + FP + FN + VP} \quad (2.8)$$

A Acurácia pode causar uma impressão incorreta se utilizada como métrica no treinamento de um conjunto de dados desequilibrados, portanto outros métodos baseados na matriz de confusão podem ser utilizados (KULKARNI; CHONG; BATARSEH, 2020), como os métodos de Precisão e Sensibilidade.

Figura 11 – Matriz de Confusão para classificação binária.

| | | Classe Prevista | |
|------------------|---|-------------------------------|-------------------------------|
| | | A | B |
| Classe Observada | A | Verdadeiros Negativos (VN) | Falsos Positivos (FP) |
| | B | Falsos Negativos (FN) | Verdadeiros Positivos (VP) |

Fonte: Adaptado de Kulkarni, Chong e Batarseh (2020).

2.2.4.3 Precisão e Sensibilidade

A medida de Precisão (do inglês *Precision*), definida pela Equação 2.9, representa a fração de detecções feitas de forma correta; enquanto a Sensibilidade, conhecida do inglês como *Sensitivity* ou *Recall*, representa apenas a fração das previsões verdadeiras (GOODFELLOW; BENGIO; COURVILLE, 2016), como mostra a Equação 2.10. Ambas métricas fornecem informações importantes a respeito do treinamento, porém o objetivo é melhorar a Sensibilidade sem que a Precisão seja afetada (CHAWLA, 2005 apud KULKARNI; CHONG; BATARSEH, 2020). Uma ilustração das medidas de Precisão e Sensibilidade está na Figura 12.

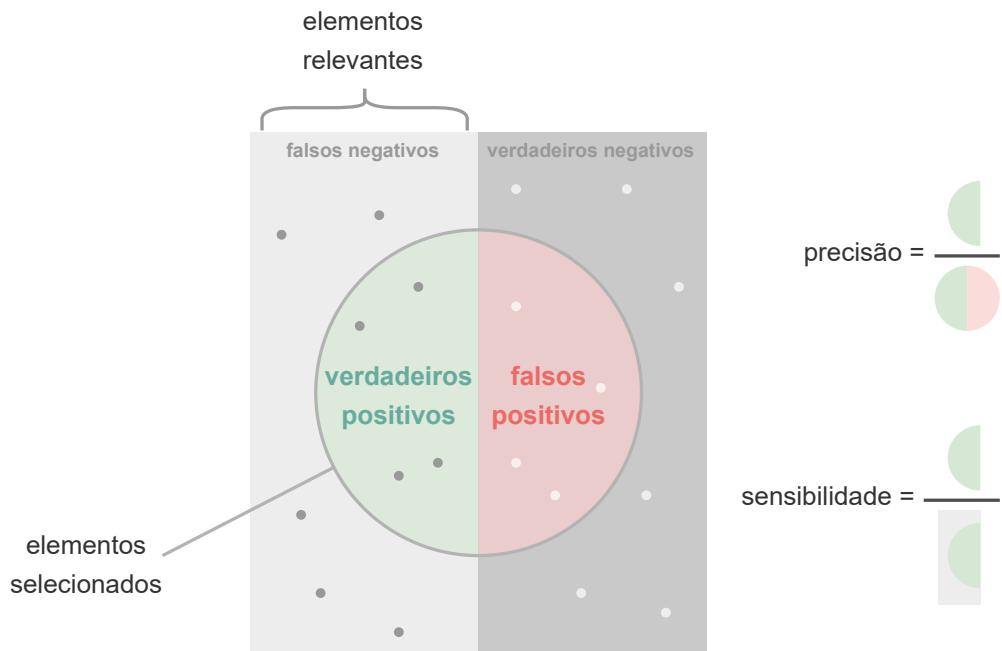
$$\text{Precisão} = \frac{\text{VP}}{\text{VP} + \text{FP}} \quad (2.9)$$

$$\text{Sensibilidade} = \frac{\text{VP}}{\text{VP} + \text{FN}} \quad (2.10)$$

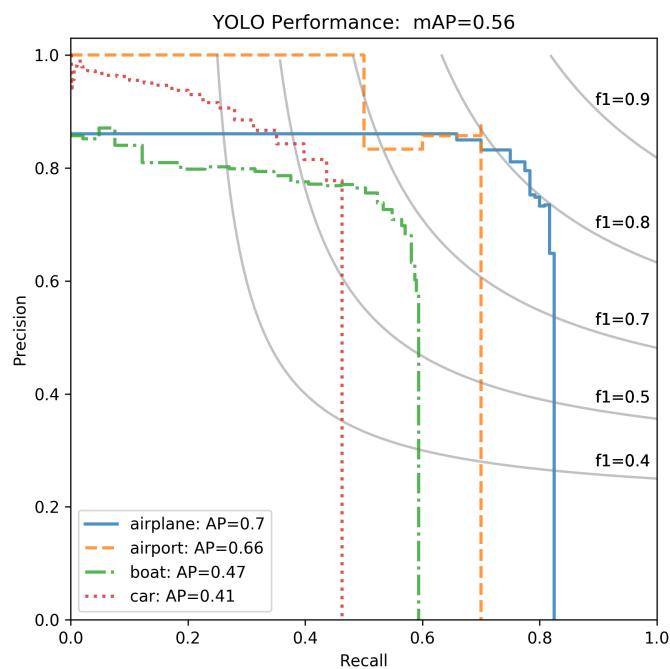
Uma curva de Precisão em função da Sensibilidade pode ser construída ao variar o *threshold* da probabilidade de um resultado pertencer a determinada classe (STEEN, 2020). Um exemplo dessa curva está na Figura 13, onde o eixo horizontal indica a Sensibilidade e o eixo vertical, a Precisão para quatro classes distintas.

2.2.4.4 Precisão Média

O valor da Precisão Média de cada classe, do inglês Average Precision (AP), é encontrado ao calcular a área sob a curva da Precisão em função da Sensibilidade. Já a média desses valores, conhecida no inglês como Mean Average Precision (mAP) é

Figura 12 – Ilustração das medidas de Precisão e Sensibilidade.

Fonte: Adaptado de Tan (2019).

Figura 13 – Curva da Precisão em função da Sensibilidade para quatro classes distintas.

Fonte: Etten (2018).

feita como maneira de avaliar um conjunto de treino completo, levando em conta todas as classes treinadas (TAN, 2019).

2.2.4.5 F-Score

A medida de *F-Score* procura encontrar o equilíbrio entre a Precisão e a Sensibilidade (MISHRA, 2018) e é feita por meio de uma média harmônica ponderada entre essas medidas (KULKARNI; CHONG; BATARSEH, 2020), conforme a Equação 2.11 . O resultado está sempre dentro do intervalo entre zero e um e quanto maior o valor de *F-Score*, melhor é a performance do modelo (MISHRA, 2018).

$$F - Score = 2 \cdot \frac{\text{Precisão} \cdot \text{Sensibilidade}}{\text{Precisão} + \text{Sensibilidade}} \quad (2.11)$$

2.2.5 Redes Neurais Convolucionais

Redes Neurais Profundas são caracterizadas por sua grande quantidade de camadas ocultas. Cada uma das camadas é responsável pelo treinamento de um conjunto distinto de dados com base na saída da camada anterior, de forma que quanto mais há avanço pela rede neural, mais complexos são as características que os nós podem reconhecer (NICHOLSON, 2020), como ilustrado na Figura 4.

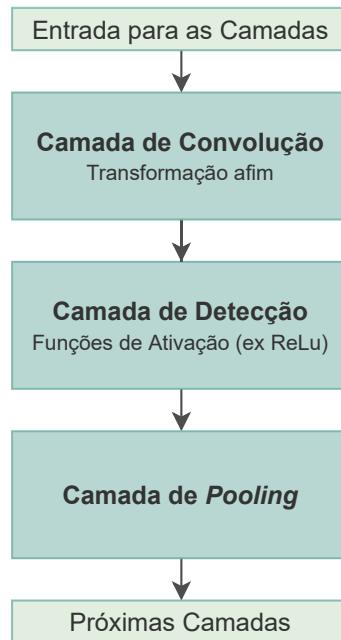
Redes Neurais Convolucionais, do inglês *convolutional neural networks* (CNN), são um tipo de rede neural profunda originalmente projetadas para análise de imagens (ZHU et al., 2018) e tem sido empregadas para esse tipo de processamento desde 1995 (YAN, 2021). O uso de CNNs reduz os requisitos de memória e possui uma melhor eficiência estática quando comparada a redes neurais tradicionais (GOODFELLOW; BENGIO; COURVILLE, 2016).

CNNs sempre contém dois tipos de operações básicas: as operações de convolução e *pooling* (ZHU et al., 2018), além de utilizarem funções de ativação. Os componentes de uma CNN típica estão ilustrados na Figura 14.

2.2.5.1 Operação de Convolução

Convolução é uma operação matemática onde duas funções produzem uma terceira função que expressa como o formato de uma é modificado ou filtrado pela outra (YAN, 2021). A operação de convolução em uma rede neural utiliza múltiplos tipos de *kernel* para extrair as características do conjunto de dados da entrada (ZHU et al., 2018), geralmente uma imagem, e cria mapas de características a partir dela (GHOLAMALINEZHAD; KHOSRAVI, 2020). O *kernel* usualmente é um vetor de parâmetros multidimensional que é adaptado pelo algoritmo de aprendizado (GOODFELLOW;

Figura 14 – Componentes de uma Rede Neural Convolucional típica.



Fonte: Adaptado de Goodfellow, Bengio e Courville (2016).

BENGIO; COURVILLE, 2016), assim como ocorre a atualização dos pesos w e b vistos na subseção 2.2.3.

Na sua forma discreta, a operação de convolução pode ser definida pela Equação 2.12 (GOODFELLOW; BENGIO; COURVILLE, 2016), onde x representa a entrada e w representa o *kernel* utilizado.

$$y[n] = (x * w)[n] = \sum_{a=-\infty}^{\infty} x[a] \cdot w[n - a] \quad (2.12)$$

Para imagens, as convoluções ocorrem em mais que um eixo ao mesmo tempo. Se a entrada é uma imagem bidimensional I , o *kernel* K provavelmente também será bidimensional, como define a Equação 2.13 (GOODFELLOW; BENGIO; COURVILLE, 2016).

$$Y[i, j] = (I * K)[i, j] = \sum_m \sum_n I[m, n] \cdot K[i - m, j - n] \quad (2.13)$$

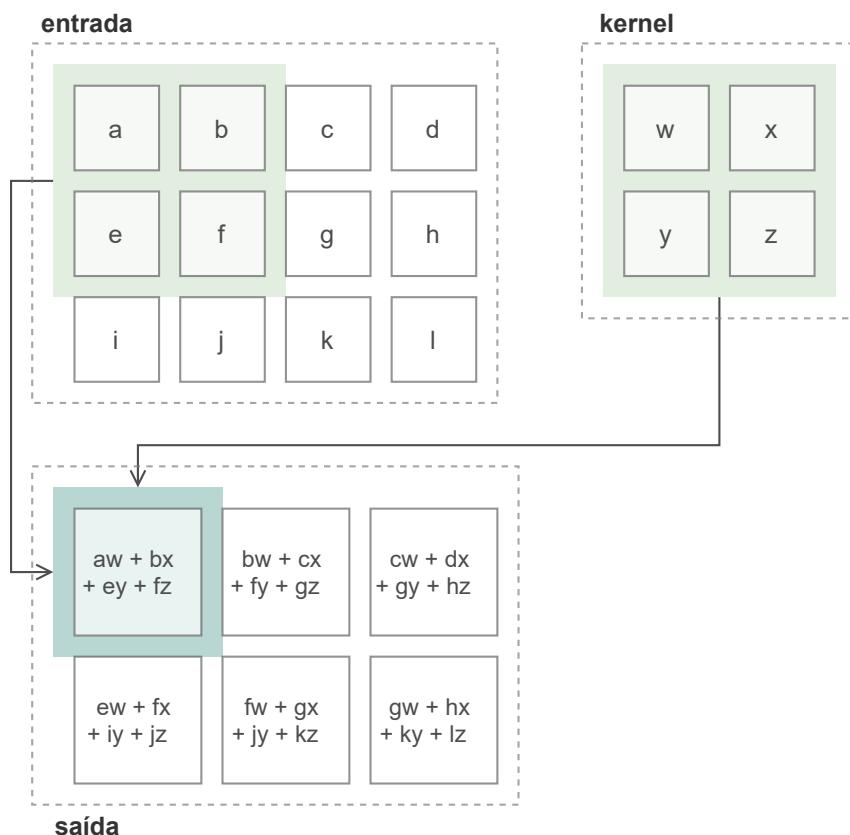
Porém, utilizando a propriedade comutativa da convolução ao inverter a ordem dos fatores I e K , a implementação se torna mais direta, já que há menos variação no intervalo válido de valores de m e n (GOODFELLOW; BENGIO; COURVILLE,

2016), pois o *kernel* é menor. Além disso, para desinverter o *kernel* em relação à imagem, utiliza-se a operação chamada correlação cruzada (GOODFELLOW; BENGIO; COURVILLE, 2016), como mostra a Equação 2.14.

$$Y[i, j] = (I * K)[i, j] = \sum_m \sum_n I[i + m, j + n] \cdot K[m, n] \quad (2.14)$$

Um exemplo de uma convolução descrita pela Equação 2.14 utilizando um *kernel* de 2 x 2 está na figura Figura 15.

Figura 15 – Exemplo de convolução entre dois vetores bidimensionais.



Fonte: Adaptado de Goodfellow, Bengio e Courville (2016).

Exemplos de convolução em imagem utilizando distintos tipos de *kernel* está na Figura 16.

2.2.5.2 Operação de Pooling

As operações de *pooling* geralmente são utilizadas após as camadas de convolução para simplificar as informações das saídas dessas camadas (NIELSEN,

Figura 16 – Operações de convolução em uma imagem.

| Operação | Kernel | Imagen Resultante |
|-------------------|--|---|
| identidade | $\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$ |  |
| detecção de borda | $\begin{pmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{pmatrix}$ $\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$ |   |

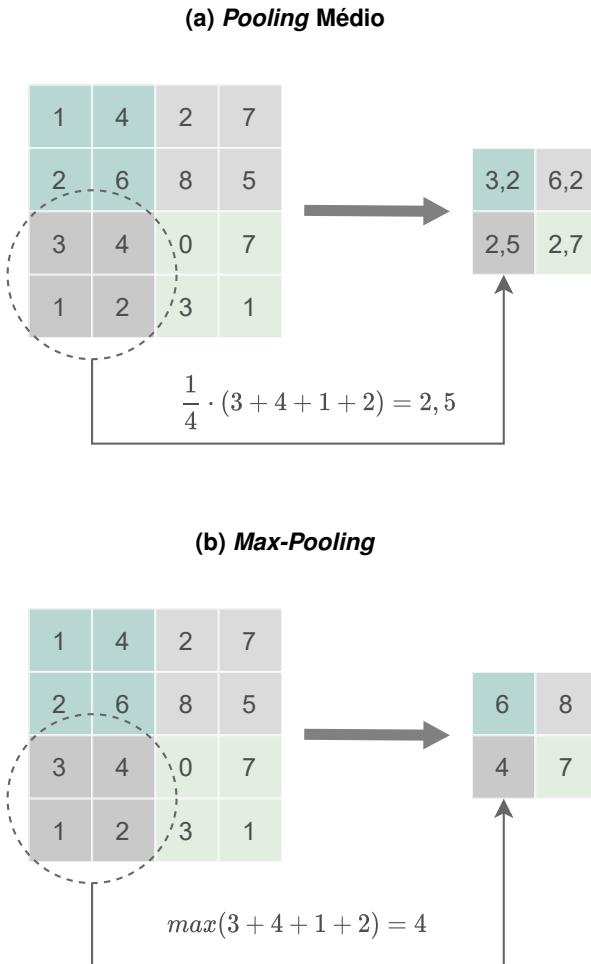
Fonte: Adaptado de Albawi, Mohammed e Al-Zawi (2017).

2018) ao reduzir o tamanho do mapa de características (GHOLAMALINEZHAD; KHOSRAVI, 2020) e ajudam a tornar as saídas aproximadamente invariantes a pequenas modificações da entrada (GOODFELLOW; BENGIO; COURVILLE, 2016).

Espera-se que uma camada de *pooling* ideal extraia somente as informações úteis e descarte detalhes irrelevantes (GHOLAMALINEZHAD; KHOSRAVI, 2020). A Figura 17 apresenta algumas das funções de *pooling*. A operação de *Max-Pooling* seleciona o pixel com maior valor dentre os pixels de uma região (Figura 17b) e a operação de *Pooling* médio (Figura 17a) faz a média entre esses pixels.

2.2.5.3 You Only Look Once

You Only Look Once (YOLO), cuja tradução direta do inglês é “você olha apenas uma vez”, é uma rede neural de passagem única (YAN, 2021) onde, ao contrário de outras abordagens, uma única rede neural é aplicada na imagem para fazer a

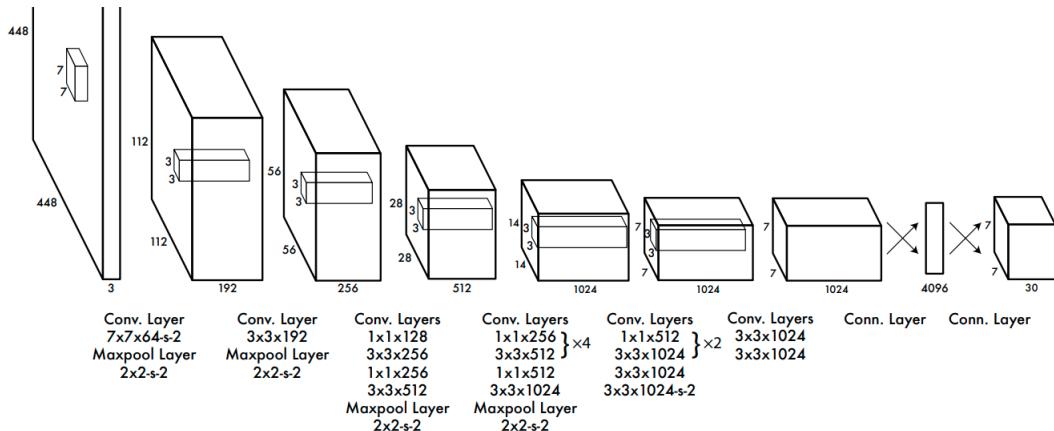
Figura 17 – Exemplos de funções utilizadas para operação de pooling.

Fonte: Adaptado de Gholamalinezhad e Khosravi (2020).

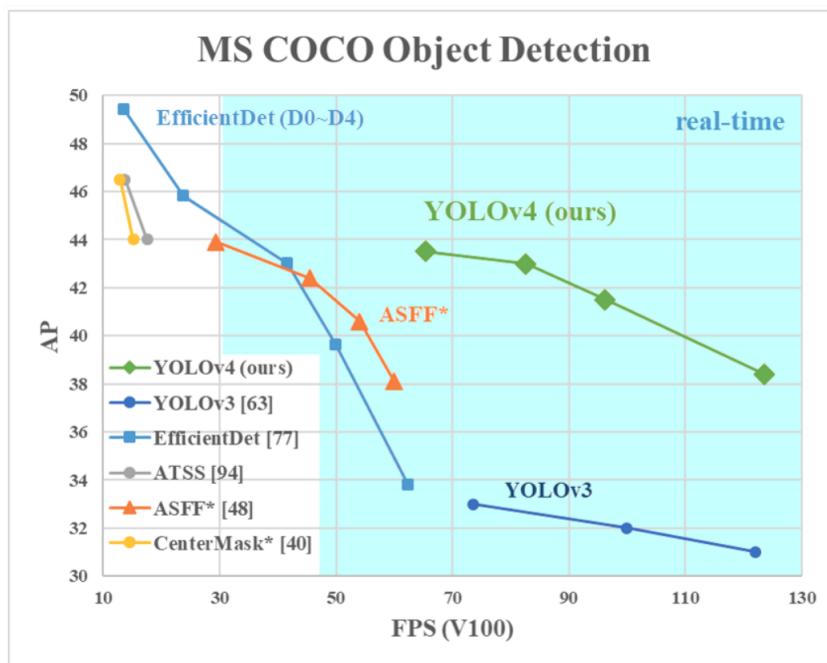
localização e a detecção do objeto, dividindo a imagem em regiões menores e fazendo a previsão e a probabilidade da detecção para cada uma das regiões (REDMON; FARHADI, 2018).

A rede neural YOLO possui 24 camadas convolucionais seguidas por duas camadas totalmente conectadas (YAN, 2021). A arquitetura básica da primeira versão está na Figura 18.

Atualmente, a rede neural YOLO está na versão 4, apresentando otimização em funções de ativação, processamento dos dados, treinamento, funções de perda, etc, em comparação com a versão anterior (WANG et al., 2021). Uma comparação de desempenho com outras redes neurais utilizadas para detecção de objetos em tempo real está no gráfico da Figura 19, que avalia a precisão média em função da quantidade de frames por segundo que a rede consegue processar, utilizando o dataset *Microsoft COCO: Common Objects in Context* (LIN et al., 2014).

Figura 18 – Arquitetura da Rede Neural YOLO.

Fonte: Redmon et al. (2015).

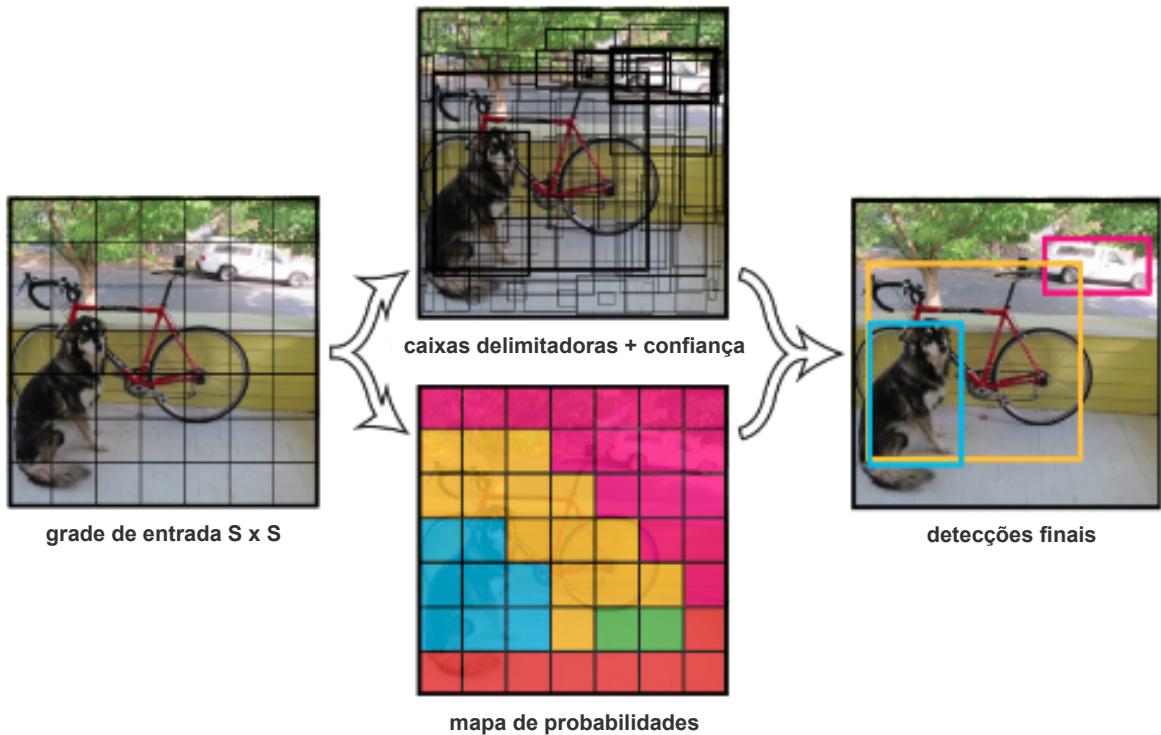
Figura 19 – Comparação da precisão média em função do processamento em frames por segundo para diferentes redes neurais utilizadas em detecção de objetos.

Fonte: Bochkovskiy, Wang e Liao (2020).

A YOLO divide a imagem em imagens menores no formato de uma grade, e para cada uma dessas divisões, são previstas caixas delimitadoras, com suas confiâncias e probabilidades (REDMON et al., 2015). A maioria dessas divisões não contém um objeto detectado e a filtragem é feita com base na probabilidade da classe do objeto, onde apenas as caixas delimitadoras de maior probabilidade permanecerão (SIVARAJKUMAR, 2019). Esse processo é ilustrado na Figura 20.

A camada final é responsável por prever as probabilidades das classes

Figura 20 – Comparação da precisão média em função do processamento em *frames* por segundo para diferentes redes neurais utilizadas em detecção de objetos.



Fonte: Adaptado de Redmon et al. (2015).

treinadas e as coordenadas das caixas delimitadoras (REDMON et al., 2015). Tanto a largura e a altura das caixas delimitadoras quanto as coordenadas x e y são normalizadas pela altura e largura da imagem, de modo que os valores fique dentro do intervalo entre zero e um (REDMON et al., 2015).

Para o treinamento, são necessárias algumas configurações de arquivos, além da adição das imagens do *dataset* utilizado, a partir do *download* do repositório do projeto disponibilizado por Bochkovskiy (2020). As instruções a serem seguidas, conforme Bochkovskiy (2020), são:

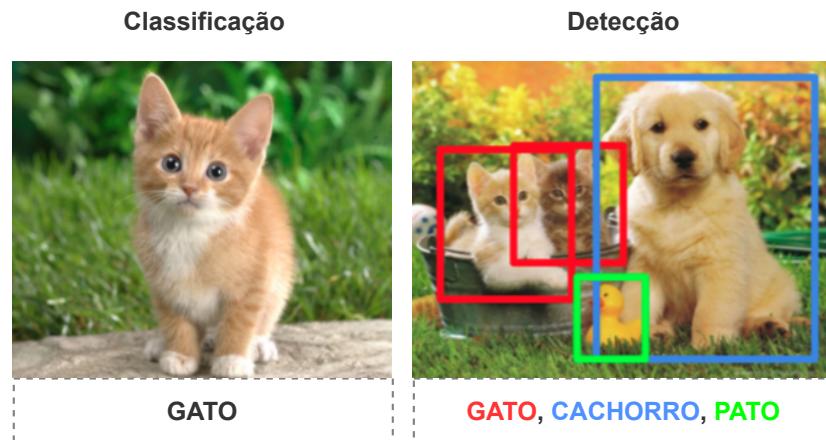
- Criar o arquivo “yolo-obj.cfg” com o mesmo conteúdo de “yolov4-custom.cfg”;
- Alterar “yolo-obj.cfg” para:
 - Considerar *batches* com 64 imagens;
 - Subdividir cada *batch* em 16 imagens (para processamento paralelo em treinamentos utilizando GPUs);
 - Utilizar um número máximo de *batches* igual ao produto $n_{classes} \cdot 2000$
 - Alterar o número de passos de acordo com o número máximo de *batches*;

- Alterar a quantidade de filtros considerando a fórmula $(n_{classes} + 5) \cdot 3$.
- Criar o arquivo “obj.names” com os nomes das classes utilizadas, com cada nome em uma linha nova;
- Alterar no arquivo “obj.data” o número de classes utilizadas;
- Colocar as imagens de treinamento e teste no diretório “obj” que se encontra dentro de “data”;
- Criar um arquivo de texto com a extensão “.txt” para cada uma das imagens do *dataset* no mesmo diretório em que elas estão contendo os objetos e suas respectivas coordenadas, seguindo o padrão $< objectclass >< x_{center} >< y_{center} >< width >< height >$ de forma que cada linha represente um objeto, onde
 - $< objectclass >$ indica a classe do objeto de acordo com a ordem definida no arquivo “obj.names” com valores dentro do intervalo $[0, n_{classes} - 1]$;
 - $< x_{center} >< y_{center} >< width >< height >$ são valores em ponto flutuante dentro do intervalo $[0.0, 1.0]$, normalizados de acordo com o tamanho da imagem utilizada.
- Criar os arquivos de texto “train.txt” e “test.txt” que contém o nome das imagens utilizadas no treinamento e teste, respectivamente, onde para cada imagem utiliza-se uma nova linha;
- Fazer o *Download* de pesos pré-treinados, com intuído de acelerar o treinamento;
- Iniciar o treinamento.

2.3 Classificação e Detecção de Objetos

A classificação e detecção de um objeto em uma imagem são duas tarefas distintas. Segundo Chen et al. (2015), “a tarefa de classificação de objetos visa prever a existência de objetos dentro das imagens, enquanto a tarefa de detecção de objetos visa localizar os objetos”, como mostra a Figura 21.

Figura 21 – Diferença entre as tarefas de classificação e detecção de objetos em uma imagem.



Fonte: Adaptado de Sivarajkumar (2019).

2.3.1 IoU

2.3.2 *Datasets*

2.3.3 Defeitos em Placas de Circuito Impresso

2.4 Frameworks e Bibliotecas

2.4.1 OpenCV

2.4.2 Darknet

2.4.3 Flask

REFERÊNCIAS

- ABRAHAM, T. et al. Contributors. In: COHEN, S. (Ed.). *Artificial Intelligence and Deep Learning in Pathology*. Elsevier, 2021. p. xiii–xiv. ISBN 978-0-323-67538-3. Disponível em: <<https://www.sciencedirect.com/science/article/pii/B9780323675383010022>>. Citado na página 8.
- ALBAWI, S.; MOHAMMED, T. A.; AL-ZAWI, S. Understanding of a convolutional neural network. In: *2017 International Conference on Engineering and Technology (ICET)*. IEEE, 2017. Disponível em: <<https://doi.org/10.1109%2Ficengtechnol.2017.8308186>>. Citado na página 23.
- BOCHKOVSKIY, A. *Darknet*. 2020. Disponível em: <<https://github.com/AlexeyAB/darknet>>. Citado na página 26.
- BOCHKOVSKIY, A.; WANG, C.-Y.; LIAO, H.-Y. M. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. Citado na página 25.
- CHAWLA, N. V. Data mining for imbalanced datasets: An overview. In: _____. *Data Mining and Knowledge Discovery Handbook*. Boston, MA: Springer US, 2005. p. 853–867. ISBN 978-0-387-25465-4. Disponível em: <https://doi.org/10.1007/0-387-25465-X_40>. Citado na página 18.
- Chen, Q. et al. Contextualizing object detection and classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 37, n. 1, p. 13–27, 2015. Citado na página 27.
- CHIN, R. T.; HARLOW, C. A. Automated visual inspection: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-4, n. 6, p. 557–573, 1982. Citado na página 7.
- DICK, S. Artificial intelligence. *Harvard Data Science Review*, v. 1, n. 1, 7 2019. [Https://hdsr.mitpress.mit.edu/pub/0aytgrau](https://hdsr.mitpress.mit.edu/pub/0aytgrau). Disponível em: <<https://hdsr.mitpress.mit.edu/pub/0aytgrau>>. Citado na página 8.
- EDGAR, T. W.; MANZ, D. O. Chapter 6 - machine learning. In: EDGAR, T. W.; MANZ, D. O. (Ed.). *Research Methods for Cyber Security*. Syngress, 2017. p. 153–173. ISBN 978-0-12-805349-2. Disponível em: <<https://www.sciencedirect.com/science/article/pii/B9780128053492000066>>. Citado na página 9.
- ETTEN, A. V. Satellite imagery multiscale rapid detection with windowed networks. *CoRR*, abs/1809.09978, 2018. Disponível em: <<http://arxiv.org/abs/1809.09978>>. Citado na página 19.
- GHOLAMALINEZHAD, H.; KHOSRAVI, H. *Pooling Methods in Deep Neural Networks, a Review*. 2020. Citado 3 vezes nas páginas 20, 23 e 24.
- GOODFELLOW, I. J.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. <<http://www.deeplearningbook.org>>. Citado 12 vezes nas páginas 8, 9, 10, 11, 12, 14, 17, 18, 20, 21, 22 e 23.

- HU, B.; WANG, J. Detection of pcb surface defects with improved faster-rcnn and feature pyramid network. In: . [S.I.: s.n.], 2020. v. 8, p. 108335–108345. Citado na página 7.
- KULKARNI, A.; CHONG, D.; BATARSEH, F. A. 5 - foundations of data imbalance and solutions for a data democracy. In: BATARSEH, F. A.; YANG, R. (Ed.). *Data Democracy*. Academic Press, 2020. p. 83–106. ISBN 978-0-12-818366-3. Disponível em: <<https://www.sciencedirect.com/science/article/pii/B9780128183663000058>>. Citado 3 vezes nas páginas 17, 18 e 20.
- State representation learning for control: An overview. *Neural Networks*, v. 108, p. 379–392, 2018. ISSN 0893-6080. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S0893608018302053>>. Citado na página 9.
- LETA, F. R.; FELICIANO, F. F.; MARTINS, F. P. R. Computer vision system for printed circuit bord inspection. In: *ABCM Symposium Series in Mechatronics*. [S.I.: s.n.], 2008. v. 3, p. 623 – 632. Citado na página 7.
- LIN, T. et al. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014. Disponível em: <<http://arxiv.org/abs/1405.0312>>. Citado na página 24.
- MAO, S. et al. Opportunities and challenges of artificial intelligence for green manufacturing in the process industry. *Engineering*, v. 5, n. 6, p. 995–1002, 2019. ISSN 2095-8099. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S2095809919300074>>. Citado na página 9.
- MISHRA, A. misc, *Metrics to Evaluate your Machine Learning Algorithm*. towardsdatascience.com, 2018. Disponível em: <<https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234>>. Citado na página 20.
- MISRA, D. Mish: A self regularized non-monotonic neural activation function. *arXiv preprint arXiv:1908.08681*, 2019. Citado 3 vezes nas páginas 13, 14 e 15.
- MITCHELL, R.; MICHALSKI, J.; CARBONELL, T. *An artificial intelligence approach*. [S.I.]: Springer, 2013. Citado na página 8.
- NG, A. Notas de Aula, *Introduction To Deep Learning*. 2020. COURSERA: Neural Networks and Deep Learning. Disponível em: <<https://www.coursera.org/learn/neural-networks-deep-learning>>. Citado 6 vezes nas páginas 12, 13, 14, 15, 16 e 17.
- NICHOLSON, C. misc, *A Beginner's Guide to Neural Networks and Deep Learning*. Pathmind A.I. Wiki, 2020. Disponível em: <<https://wiki.pathmind.com/neural-network#define>>. Citado 2 vezes nas páginas 13 e 20.
- NIELSEN, M. A. misc, *Neural Networks and Deep Learning*. Determination Press, 2018. Disponível em: <<http://neuralnetworksanddeeplearning.com/>>. Citado 2 vezes nas páginas 12 e 23.
- RASHID, T. *Make Your Own Neural Network*. 1. ed. [S.I.]: CreateSpace Independent Publishing Platform, 2016. ISBN 1530826608,9781530826605. Citado na página 8.

- REDMON, J. et al. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015. Disponível em: <<http://arxiv.org/abs/1506.02640>>. Citado 2 vezes nas páginas 25 e 26.
- REDMON, J.; FARHADI, A. Yolov3: An incremental improvement. *arXiv*, 2018. Citado na página 24.
- ROBINS, M. The difference between artificial intelligence, machine learning and deep learning. *Intel Artificial Intelligence*, 5 2020. Disponível em: <<https://www.intel.com.br/content/www/br/pt/artificial-intelligence/posts/difference-between-ai-machine-learning-deep-learning.html>>. Citado 2 vezes nas páginas 9 e 11.
- RUSSELL, S.; MOSKOWITZ, I. S.; JALAIAN, B. Chapter 4 - context: Separating the forest and the trees—wavelet contextual conditioning for ai. In: LAWLESS, W. F.; MITTU, R.; SOFGE, D. A. (Ed.). *Human-Machine Shared Contexts*. Academic Press, 2020. p. 67–91. ISBN 978-0-12-820543-3. Disponível em: <<https://www.sciencedirect.com/science/article/pii/B9780128205433000043>>. Citado na página 8.
- SAUER, A. misc, *Quick Guide to Gradient Descent and Its Variants*. Morioh, 2020. Disponível em: <<https://morioh.com/p/15c995420be6>>. Citado na página 16.
- SHARMA, S. Activation functions in neural networks. *towards data science*, v. 6, 2017. Citado 2 vezes nas páginas 13 e 14.
- SIVARAJKUMAR, S. misc, *Image Classification And Object Detection*. medium.com, 2019. Disponível em: <<https://medium.com/@sonish.sivarajkumar/image-classification-and-object-detection-c9803f854923>>. Citado 2 vezes nas páginas 25 e 28.
- STEEN, D. misc, *Precision-Recall Curves*. medium.com, 2020. Disponível em: <<https://medium.com/@douglaspsteen/precision-recall-curves-d32e5b290248>>. Citado na página 18.
- TAN, R. J. misc, *Breaking Down Mean Average Precision (mAP)*. towardsdatascience.com, 2019. Disponível em: <<https://towardsdatascience.com/breaking-down-mean-average-precision-map-ae462f623a52>>. Citado 2 vezes nas páginas 19 e 20.
- WALCZAK, S.; CERPA, N. Artificial neural networks. In: MEYERS, R. A. (Ed.). *Encyclopedia of Physical Science and Technology (Third Edition)*. Third edition. New York: Academic Press, 2003. p. 631–645. ISBN 978-0-12-227410-7. Disponível em: <<https://www.sciencedirect.com/science/article/pii/B0122274105008371>>. Citado na página 12.
- WANG, C. et al. Citrus recognition based on yolov4 neural network. *Journal of Physics: Conference Series*, IOP Publishing, v. 1820, n. 1, p. 012163, mar 2021. Disponível em: <<https://doi.org/10.1088/1742-6596/1820/1/012163>>. Citado na página 24.
- WOOLF, B. P. Chapter 7 - machine learning. In: WOOLF, B. P. (Ed.). *Building Intelligent Interactive Tutors*. San Francisco: Morgan Kaufmann, 2009. p. 221–297. ISBN

- 978-0-12-373594-2. Disponível em: <<https://www.sciencedirect.com/science/article/pii/B9780123735942000071>>. Citado na página 9.
- WU, W.-Y.; WANG, M.-J. J.; LIU, C.-M. Automated inspection of printed circuit boards through machine vision. *Computers in Industry*, v. 28, n. 2, p. 103–111, 1996. ISSN 0166-3615. Disponível em: <<https://www.sciencedirect.com/science/article/pii/0166361595000631>>. Citado na página 7.
- YAN, W. Q. *Computational Methods for Deep Learning: Theoretic, Practice and Applications*. [S.I.]: Springer, 2021. (Texts in Computer Science). ISBN 3030610802, 9783030610807. Citado 4 vezes nas páginas 16, 20, 23 e 24.
- ZHU, W. et al. Deep learning based soft sensor and its application on a pyrolysis reactor for compositions predictions of gas phase components. In: EDEN, M. R.; IERAPETRITOU, M. G.; TOWLER, G. P. (Ed.). *13th International Symposium on Process Systems Engineering (PSE 2018)*. Elsevier, 2018, (Computer Aided Chemical Engineering, v. 44). p. 2245–2250. Disponível em: <<https://www.sciencedirect.com/science/article/pii/B9780444642417503694>>. Citado na página 20.
- ZUMBAHLEN, H. Chapter 12 - printed circuit-board design issues. In: ZUMBAHLEN, H. (Ed.). *Linear Circuit Design Handbook*. Burlington: Newnes, 2008. p. 821–895. ISBN 978-0-7506-8703-4. Disponível em: <<https://www.sciencedirect.com/science/article/pii/B9780750687034000122>>. Citado na página 7.

APÊNDICES

APÊNDICE A – NOTAÇÕES DE REDES NEURAIS

Standard notations for Deep Learning

This document has the purpose of discussing a new standard for deep learning mathematical notations.

1 Neural Networks Notations.

General comments:

- superscript (i) will denote the i^{th} training example while superscript [l] will denote the l^{th} layer

Sizes:

· m : number of examples in the dataset

· n_x : input size

· n_y : output size (or number of classes)

· $n_h^{[l]}$: number of hidden units of the l^{th} layer

In a for loop, it is possible to denote $n_x = n_h^{[0]}$ and $n_y = n_h^{[\text{number of layers} + 1]}$.

· L : number of layers in the network.

Objects:

· $X \in \mathbb{R}^{n_x \times m}$ is the input matrix

· $x^{(i)} \in \mathbb{R}^{n_x}$ is the i^{th} example represented as a column vector

· $Y \in \mathbb{R}^{n_y \times m}$ is the label matrix

· $y^{(i)} \in \mathbb{R}^{n_y}$ is the output label for the i^{th} example

· $W^{[l]} \in \mathbb{R}^{\text{number of units in next layer} \times \text{number of units in the previous layer}}$ is the weight matrix,superscript [l] indicates the layer

· $b^{[l]} \in \mathbb{R}^{\text{number of units in next layer}}$ is the bias vector in the l^{th} layer

· $\hat{y} \in \mathbb{R}^{n_y}$ is the predicted output vector. It can also be denoted $a^{[L]}$ where L is the number of layers in the network.

Common forward propagation equation examples:

$a = g^{[l]}(W_x x^{(i)} + b_1) = g^{[l]}(z_1)$ where $g^{[l]}$ denotes the l^{th} layer activation function

$\hat{y}^{(i)} = softmax(W_h h + b_2)$

· General Activation Formula: $a_j^{[l]} = g^{[l]}(\sum_k w_{jk}^{[l]} a_k^{[l-1]} + b_j^{[l]}) = g^{[l]}(z_j^{[l]})$

· $J(x, W, b, y)$ or $J(\hat{y}, y)$ denote the cost function.

Examples of cost function:

· $J_{CE}(\hat{y}, y) = -\sum_{i=0}^m y^{(i)} \log \hat{y}^{(i)}$

· $J_1(\hat{y}, y) = \sum_{i=0}^m |y^{(i)} - \hat{y}^{(i)}|$