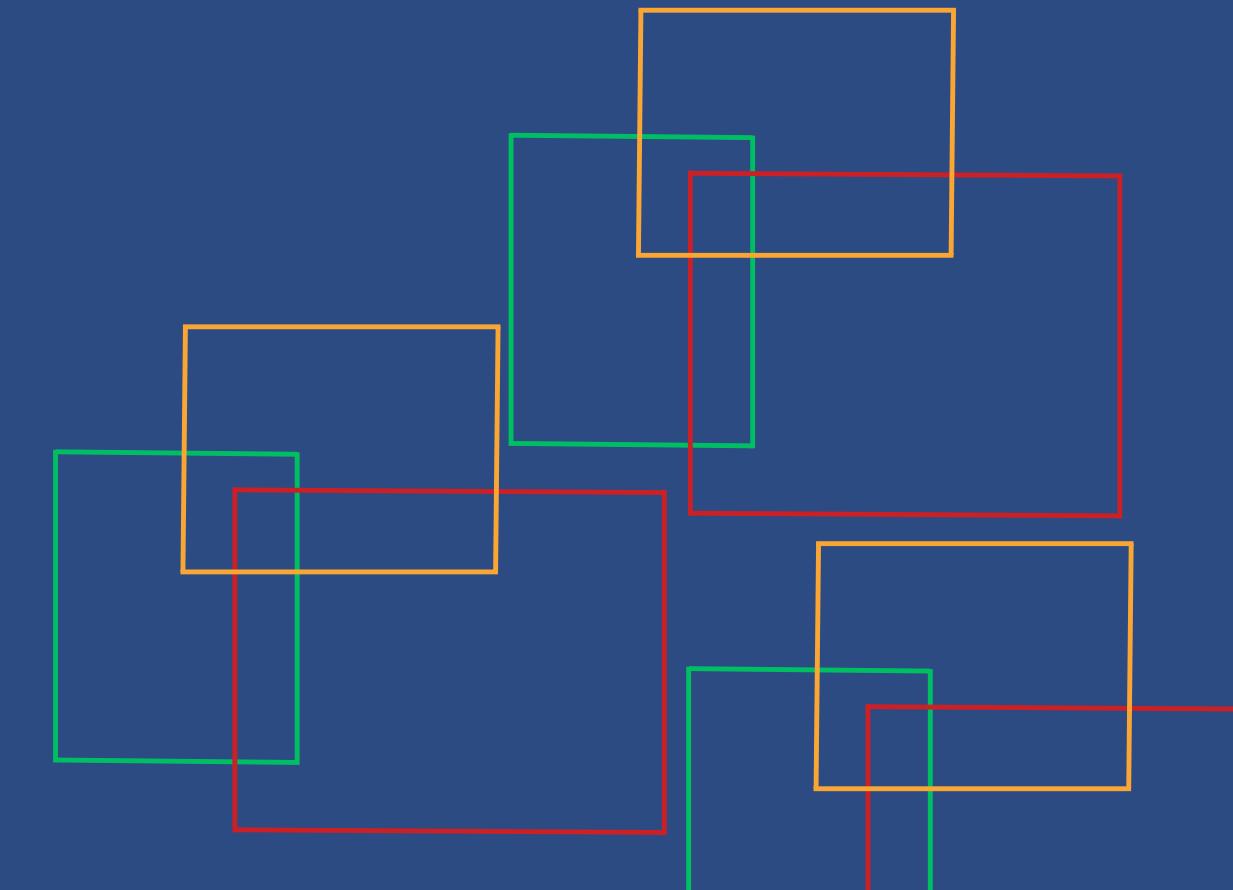


SEMANA ACADÊMICA DA COMPUTAÇÃO 2025



Redes Neurais Aplicadas à Detecção de Objetos em Imagens com YOLO

Kananda Winter
Maria Luiza Prata
Milena Ferreira



Objetivos

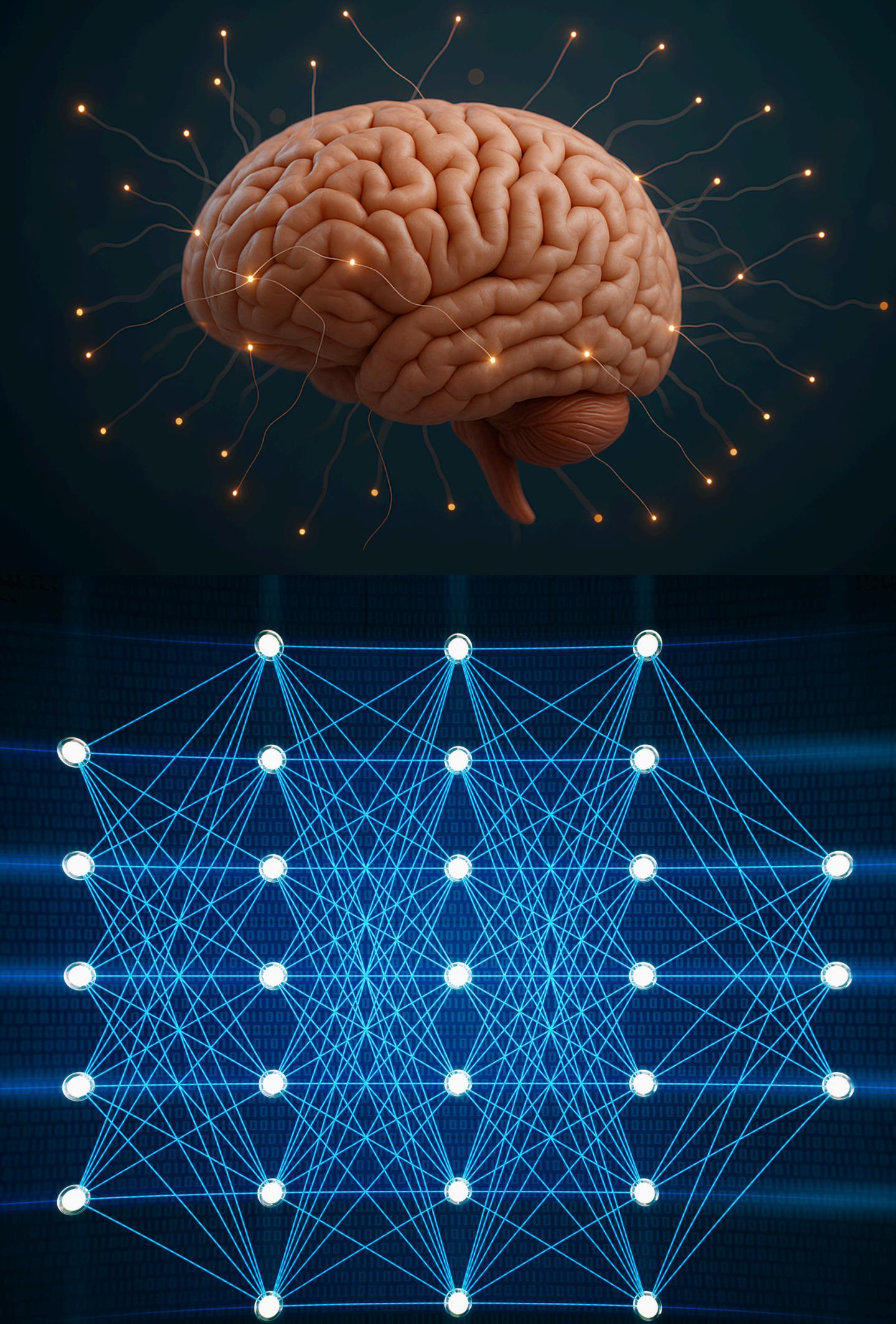
- Entender o que são redes neurais e como funcionam.
- Aprender como redes neurais lidam com imagens (CNNs).
- Conhecer o algoritmo YOLO e como ele detecta objetos.
- Ver onde o YOLO é usado na prática.
- Saber como testar e usar o YOLO em projetos reais.

O que são
redes neurais?

O que são redes neurais?

Redes neurais são modelos computacionais inspirados no funcionamento do cérebro humano.

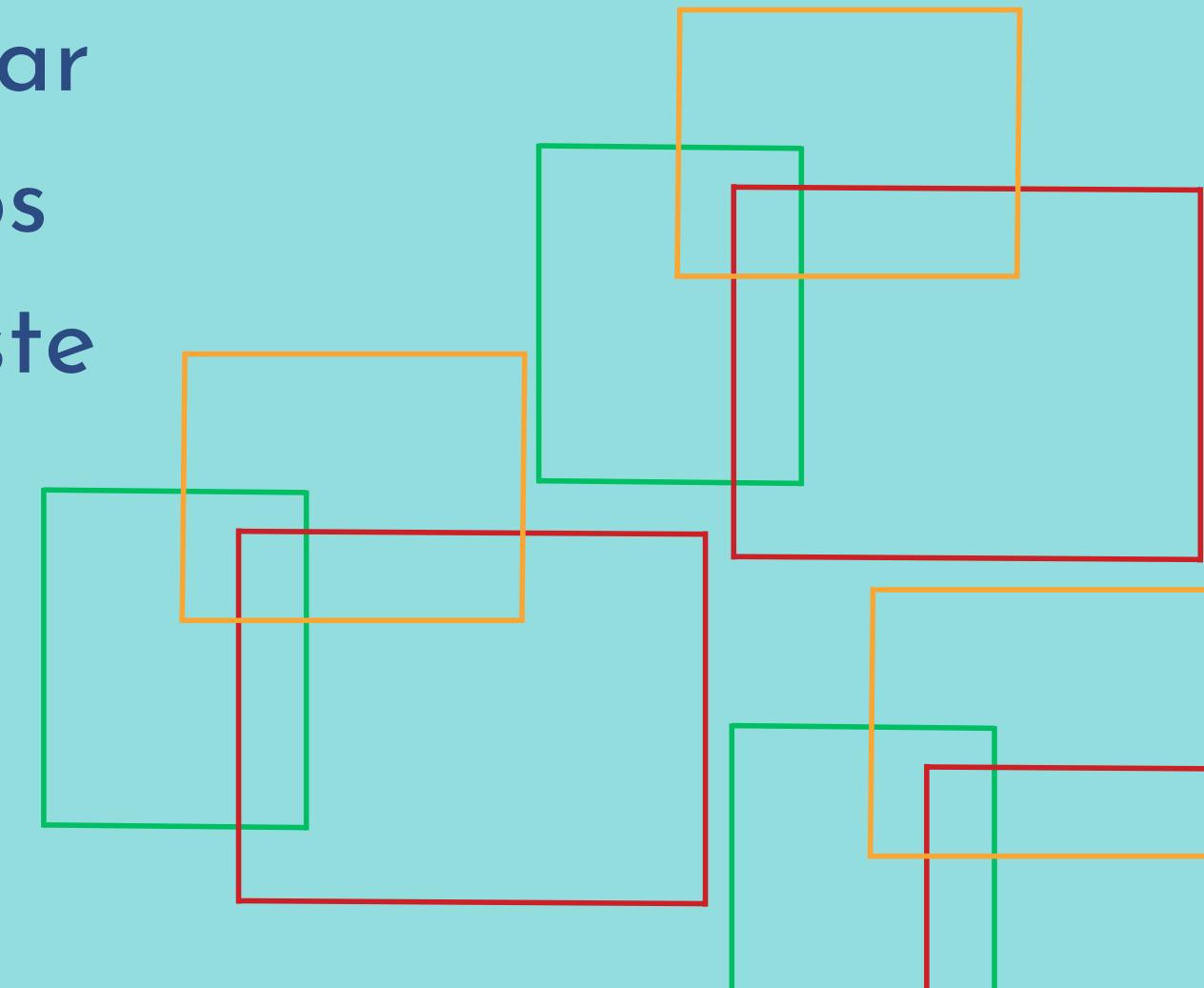
Elas são utilizadas principalmente em tarefas de inteligência artificial e aprendizado de máquina (machine learning), como reconhecimento de voz, imagem, tradução automática, detecção de objetos, entre muitas outras.



Como funciona uma rede neural?

Uma rede neural é composta por camadas de "neurônios" artificiais. Esses neurônios recebem dados de entrada, processam essas informações e repassam o resultado para os próximos neurônios, formando uma estrutura em camadas.

Durante esse processo, a rede aprende a identificar padrões nos dados, ajustando automaticamente os "pesos" das conexões entre os neurônios. Esse ajuste acontece por meio de um processo de treino, que permite à rede melhorar suas respostas com o tempo.



Estrutura básica da rede neural:

1º - Camada de entrada:

Recebe os dados brutos (por exemplo, pixels de uma imagem, valores numéricos, etc).

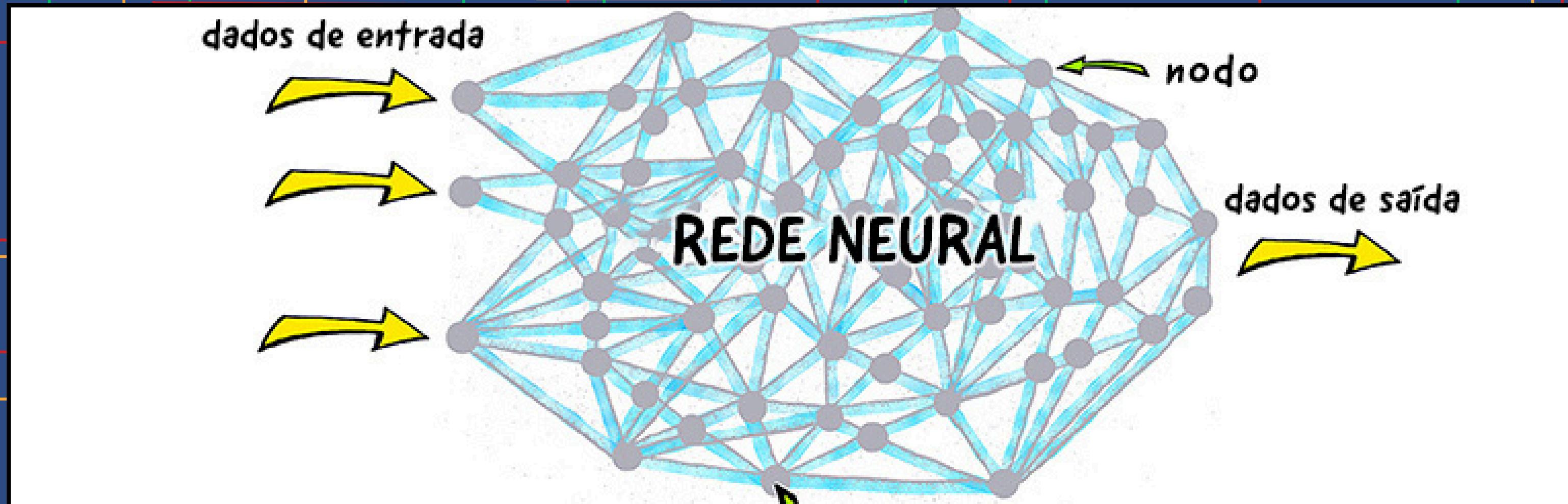
2º - Camadas ocultas:

Fazem os cálculos e extraem padrões. Quanto mais camadas e neurônios, mais complexa e poderosa é a rede (isso é o chamado deep learning ou aprendizado profundo).

3º - Camada de saída:

Entrega o resultado final, como uma classe (ex: "gato" ou "cachorro") ou um valor numérico (ex: previsão de preço).



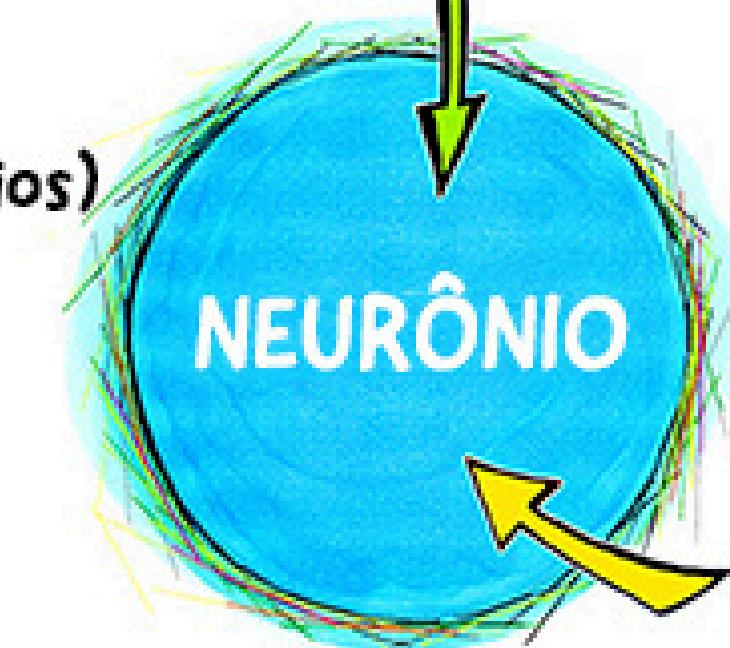


dados de entrada

(dados que entram nos neurônios)

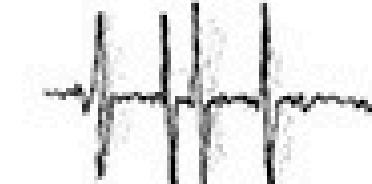


001011010



dados de saída

(dados que saem dos neurônios)



010110100

**cálculos
matemáticos**

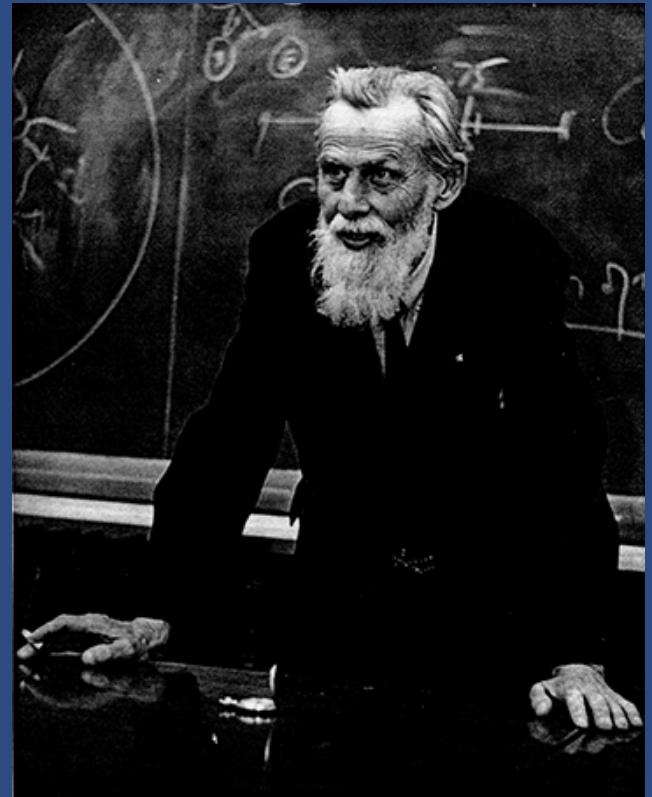


**Como surgiram
as redes neurais?**

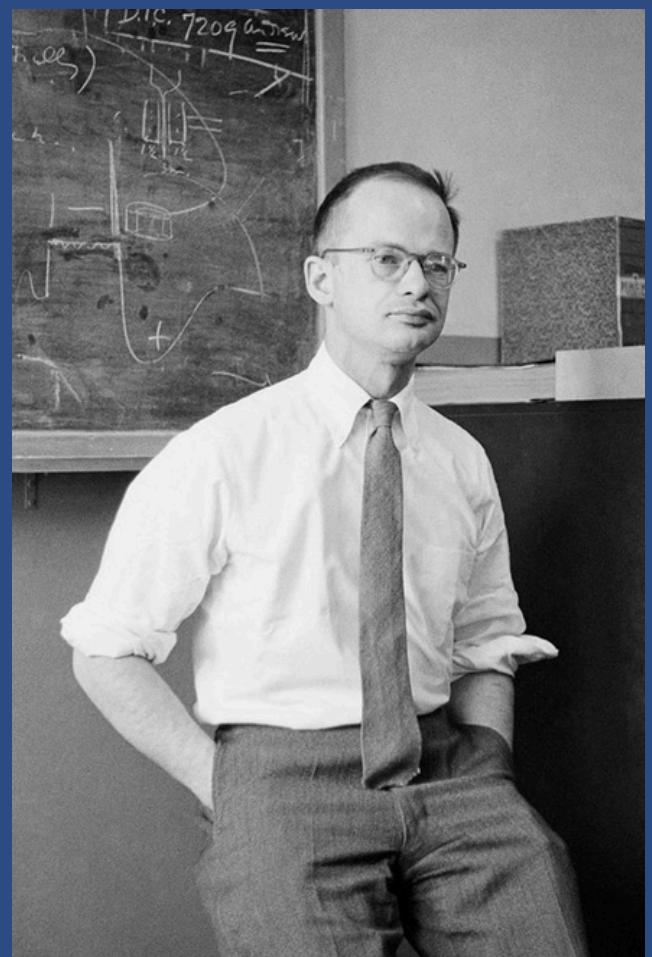
Como surgiram as redes neurais

MCP – O primeiro neurônio artificial

- Em 1943 Warren McCulloch e Walter Pitts criam o primeiro modelo matemático de um neurônio artificial
- Modelo: entradas binárias + pesos → soma → limiar
→ saída (0 ou 1)
- Simulava funções lógicas como AND e OR
- Base teórica para o desenvolvimento das redes neurais



Warren McCulloch (1898 - 1969)

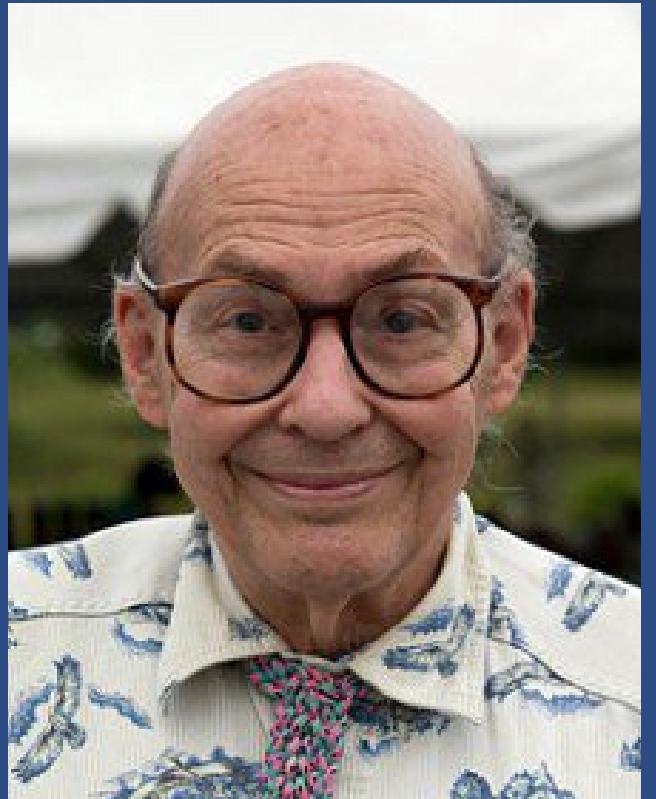


Walter Pitts (1923 - 1969)

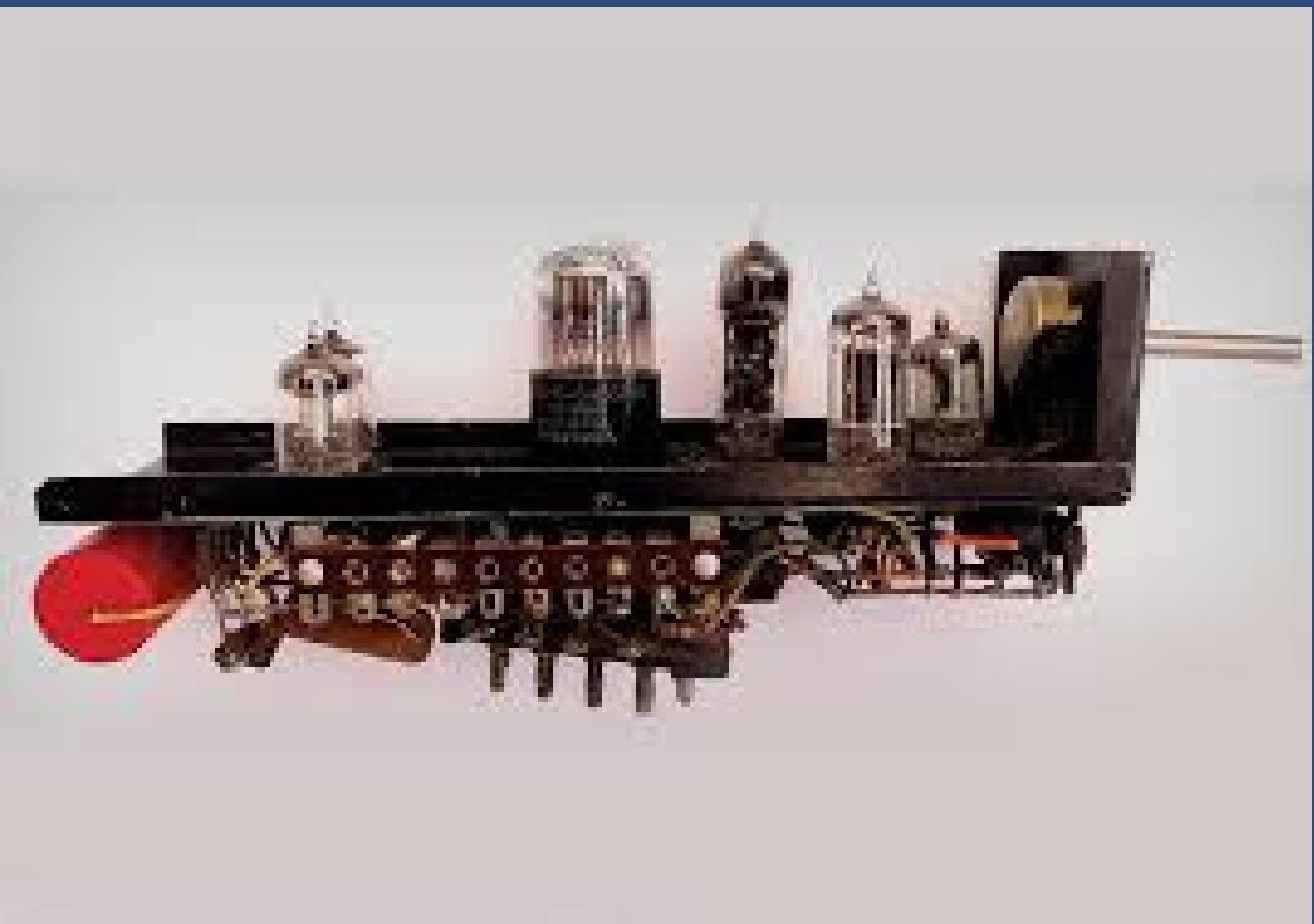
Como surgiram as redes neurais

SNARC – O primeiro computador neural

- Em 1951, Marvin Minsky e Dean Edmonds criam o SNARC: um computador com 40 neurônios artificiais simulados
- Usava componentes analógicos e sinapses ajustáveis
- Aplicava ideias de reforço e aprendizagem
- Tentava simular fisicamente o comportamento do cérebro



Marvey Minsty (1927 - 2016)

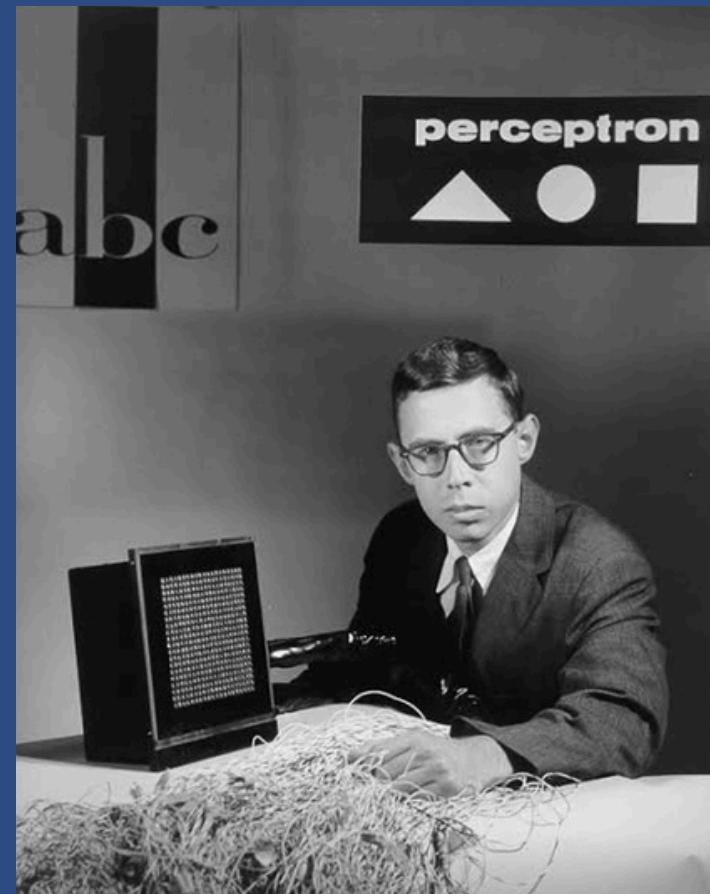


SNARC - Calculadora de Reforço Analógico Neural Estocástico

Como surgiram as redes neurais

Perceptron – A primeira rede que aprende

- Em 1957, Frank Rosenblatt desenvolve o Perceptron, inspirado no sistema visual humano
- Aprendizado supervisionado: ajustava pesos com base em erros
- Implementado em FORTRAN e depois em hardware (Mark I Perceptron)
- Considerado o primeiro sistema de IA com grande impacto



Frank Rosenblatt (1928 - 1971)

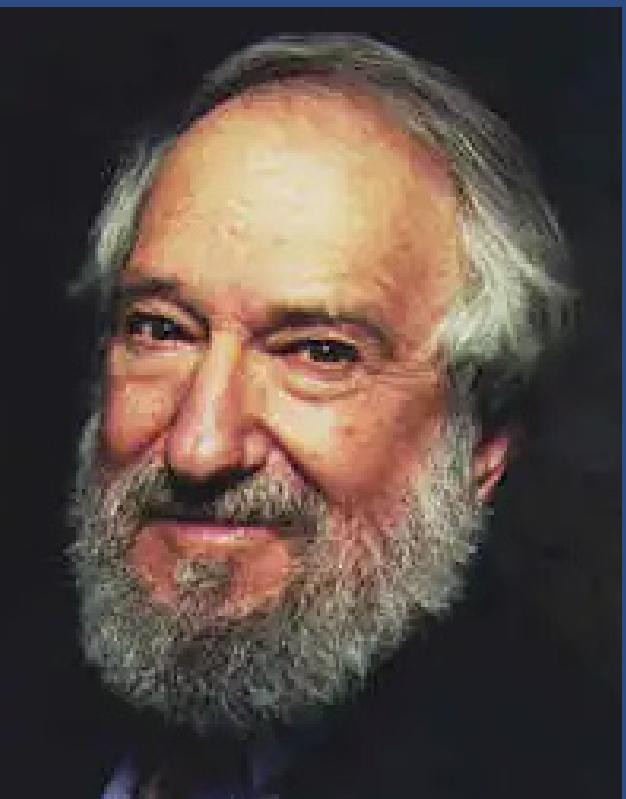


Mark I Perceptron

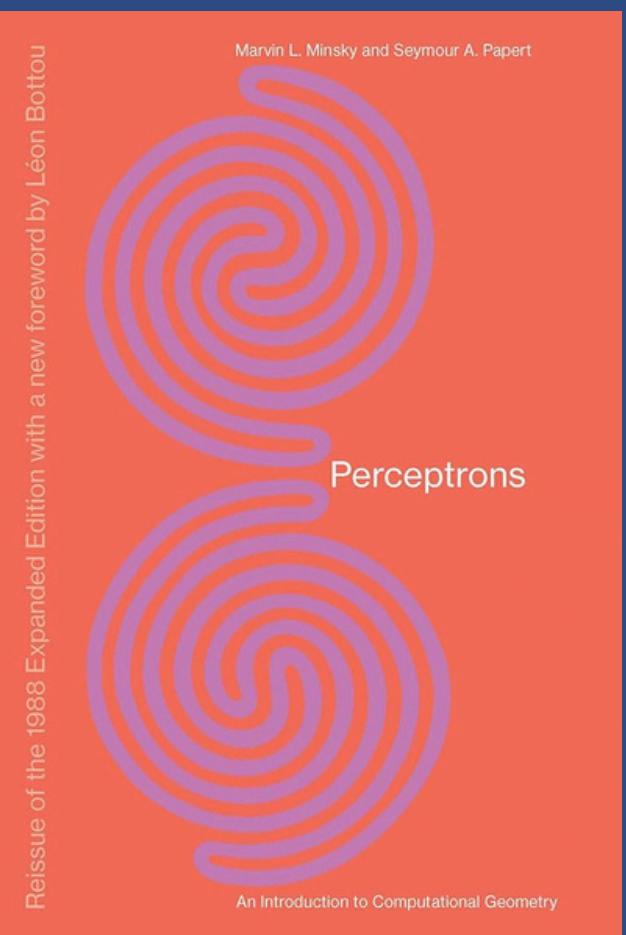
Como surgiram as redes neurais

O inverno das redes neurais (1969 - 1980)

- Em 1969, Warren Minsky e Seymour Papert publicam o livro *Perceptrons: Uma Introdução a Geometria Computacional*
- Mostram que o Perceptron de Rosenblatt não resolve o problema do XOR
- O livro é interpretado como uma crítica definitiva às redes neurais
- Resulta em financiamentos cortados e desinteresse científico nas redes neurais.



Seymour Papert (1928 - 2016)



Perceptrons: an introduction
to computational geometry

Como surgiram as redes neurais

Backpropagation

- Em 1986, Geoffrey Hinton, David Rumelhart e Ronald Williams escrevem o artigo “Learning representations by back-propagating errors”
- O algoritmo permite treinar redes com camadas ocultas e ajusta os pesos com base no erro
- Supera as limitações do Perceptron, encerrando o “inverno das redes neurais”
- Estabelece as bases para o surgimento do deep learning



Geoffrey Hinton



David Rumelhart



Ronald Williams

Como surgiram as redes neurais

A evolução das redes neurais

- A partir dos anos 2000, dois fatores mudam tudo:
 - Big Data → grandes volumes de dados
 - GPUs → processamento rápido e paralelo
- Possibilitam aplicações reais, como reconhecimento de imagens, tradução automática e assistentes de voz
- Novas arquiteturas: Redes Neurais Convolucionais (CNNs) para imagens, Redes Recorrentes (RNNs) para sequências temporais e, mais recentemente, os Transformers para linguagem natural
- Redes neurais se tornam o pilar da IA moderna



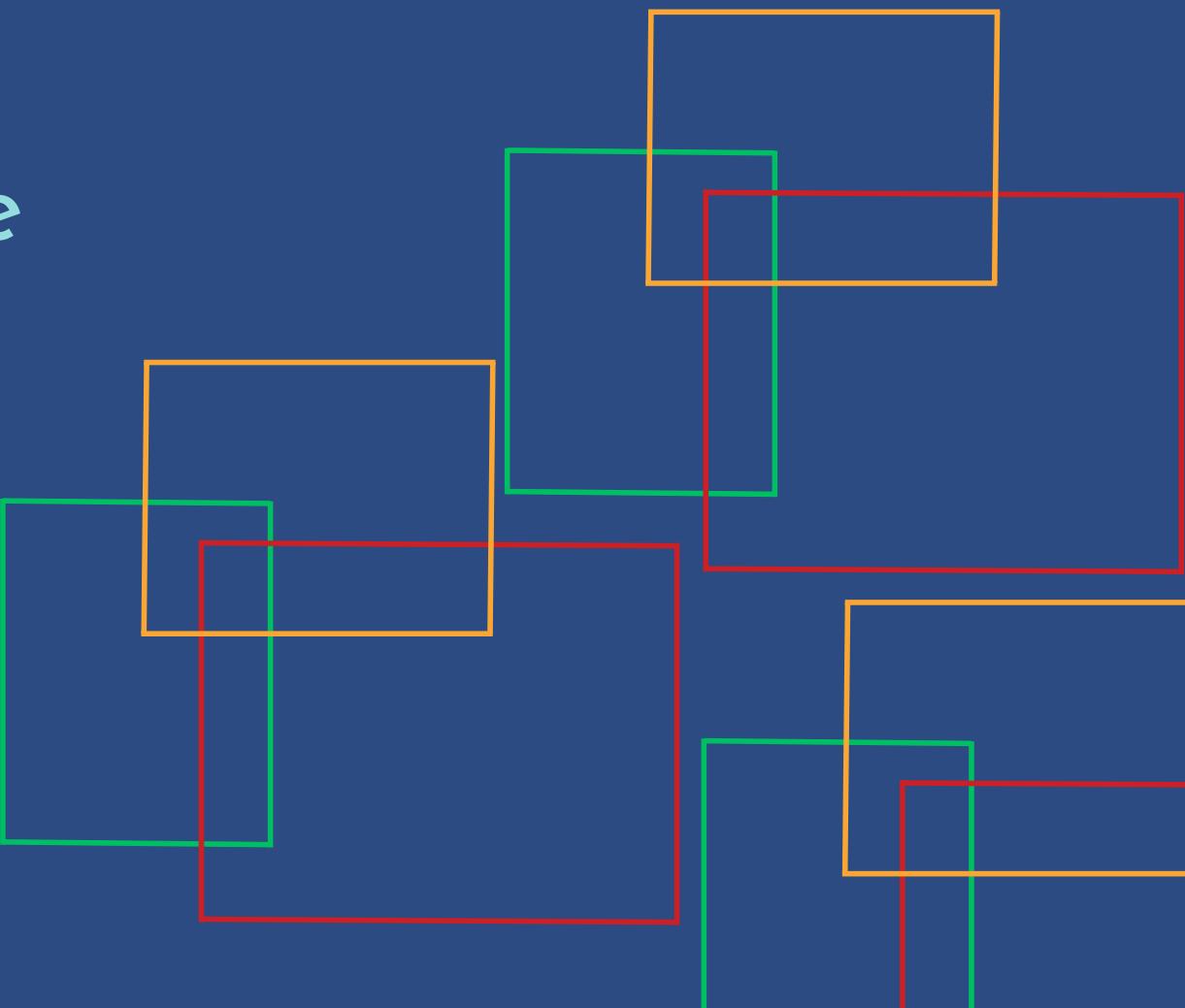
Os "pais do deep learning" que receberam o Prêmio Turing de 2018 por revolucionarem a inteligência artificial moderna.

O que é um
neurônio artificial?

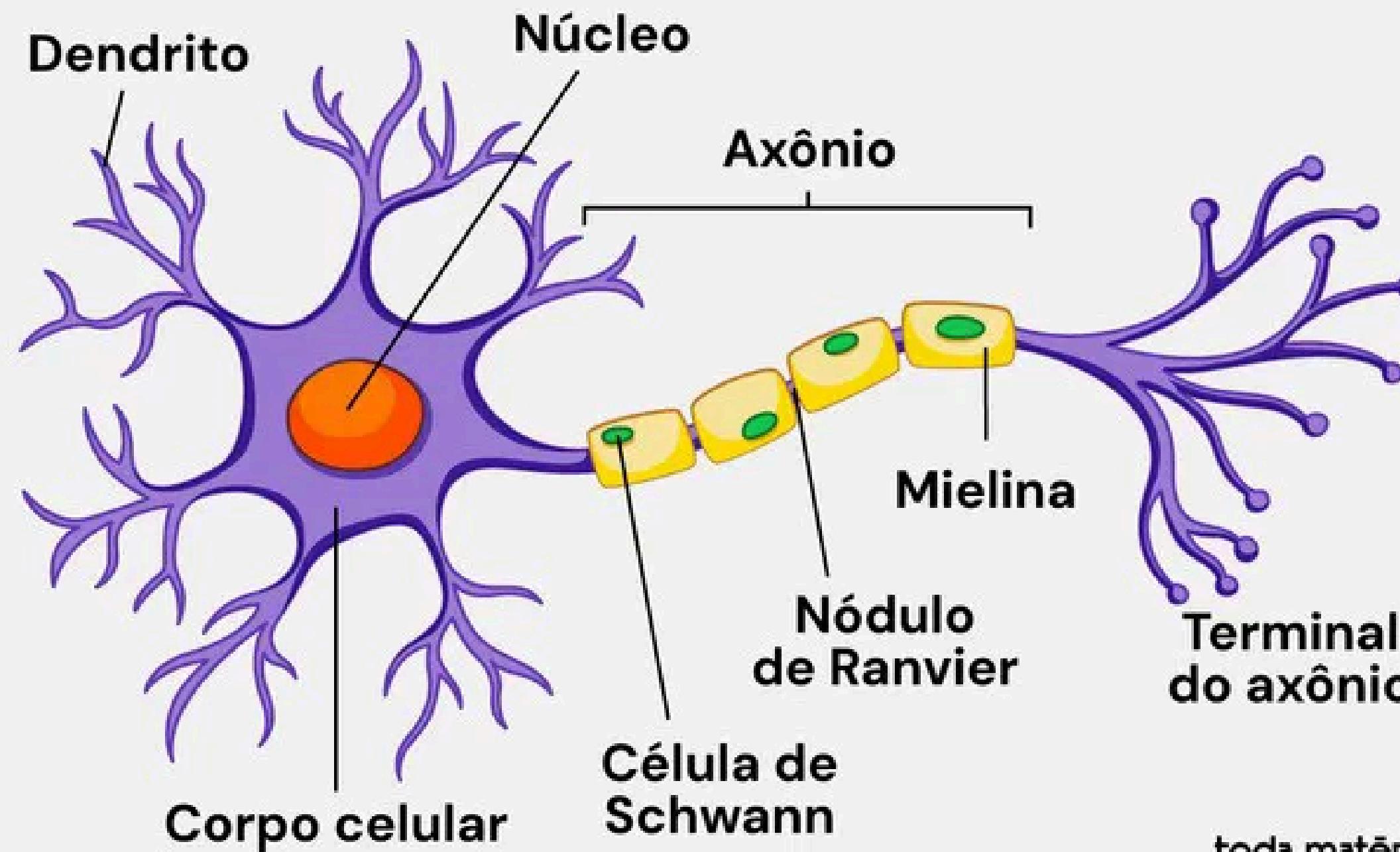
O que é um neurônio artificial?

Um neurônio artificial é um modelo matemático inspirado no neurônio biológico. Ele recebe valores de entrada, faz uma combinação ponderada dessas entradas e aplica uma função de ativação para produzir uma saída.

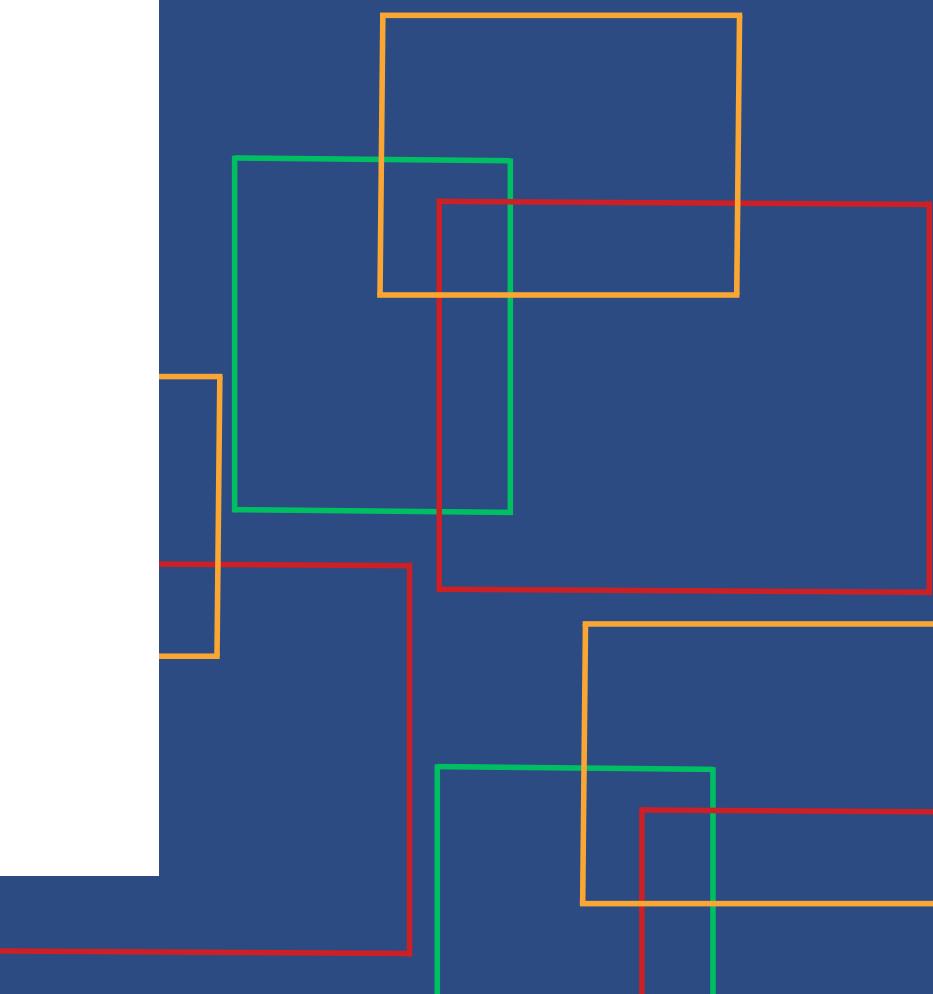
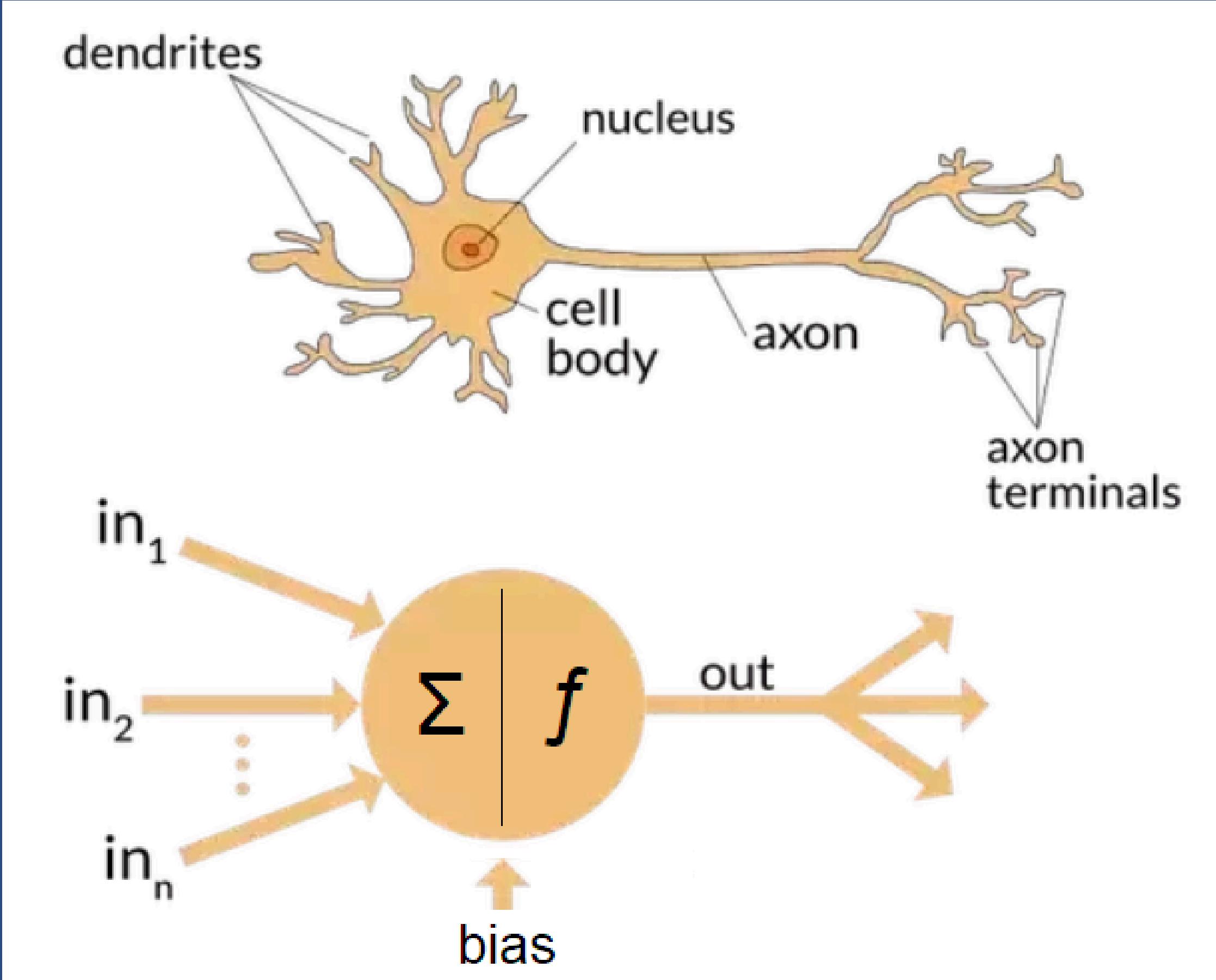
Ele realiza um pequeno cálculo com os dados que recebe e decide se deve “ativar” (passar um valor adiante) com base nesse cálculo.



Estrutura dos Neurônios



toda matéria

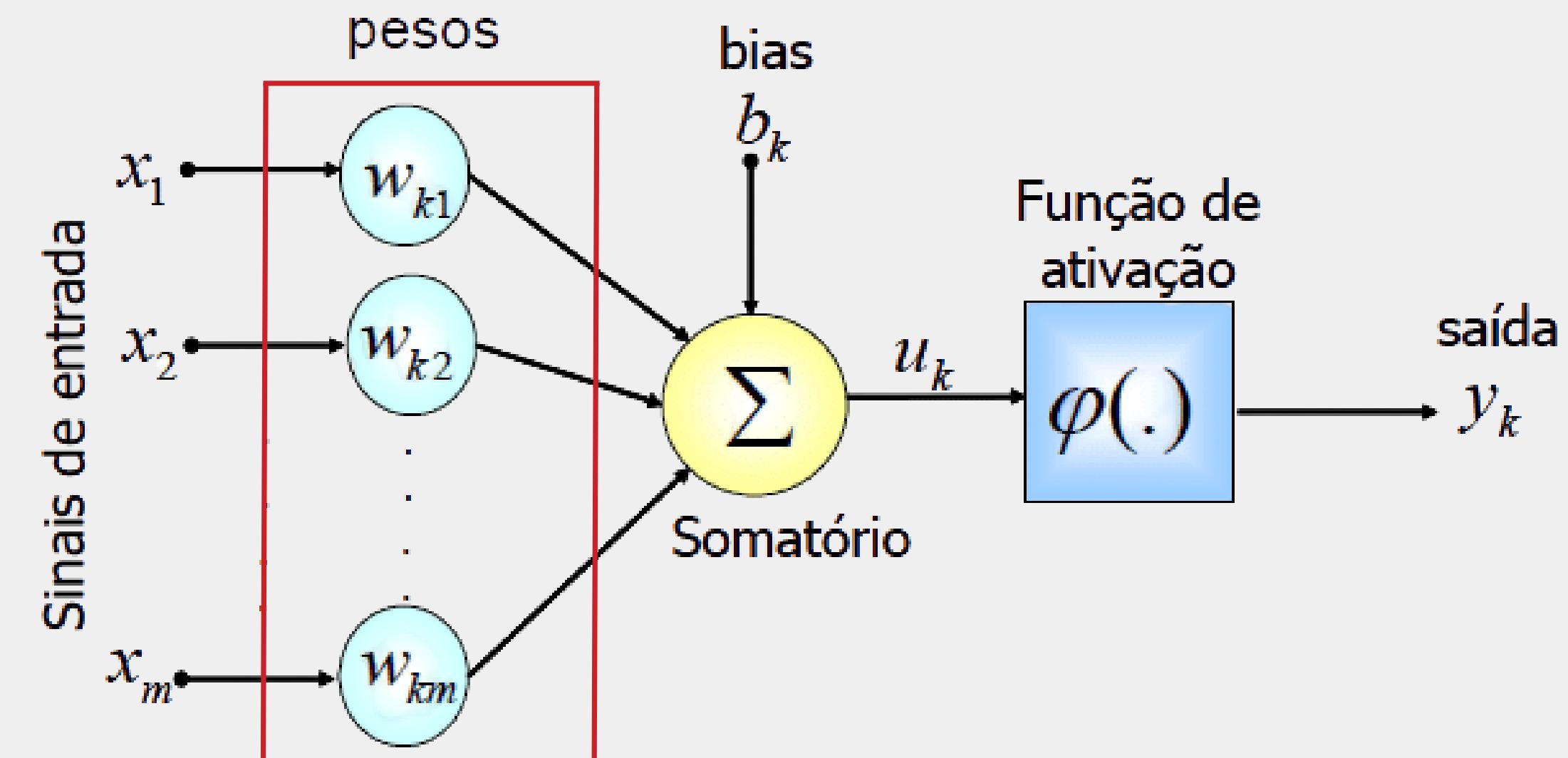


Partes de um neurônio artificial

Pesos (w):

Cada entrada que chega ao neurônio (ex. um número), passa por uma conexão chamada sinapse. Essas conexões têm pesos, que indicam o quanto aquela entrada é importante.

Por exemplo, se uma entrada tiver um peso maior, ela vai influenciar mais no resultado. Na prática, cada entrada x é multiplicada pelo seu respectivo peso w .



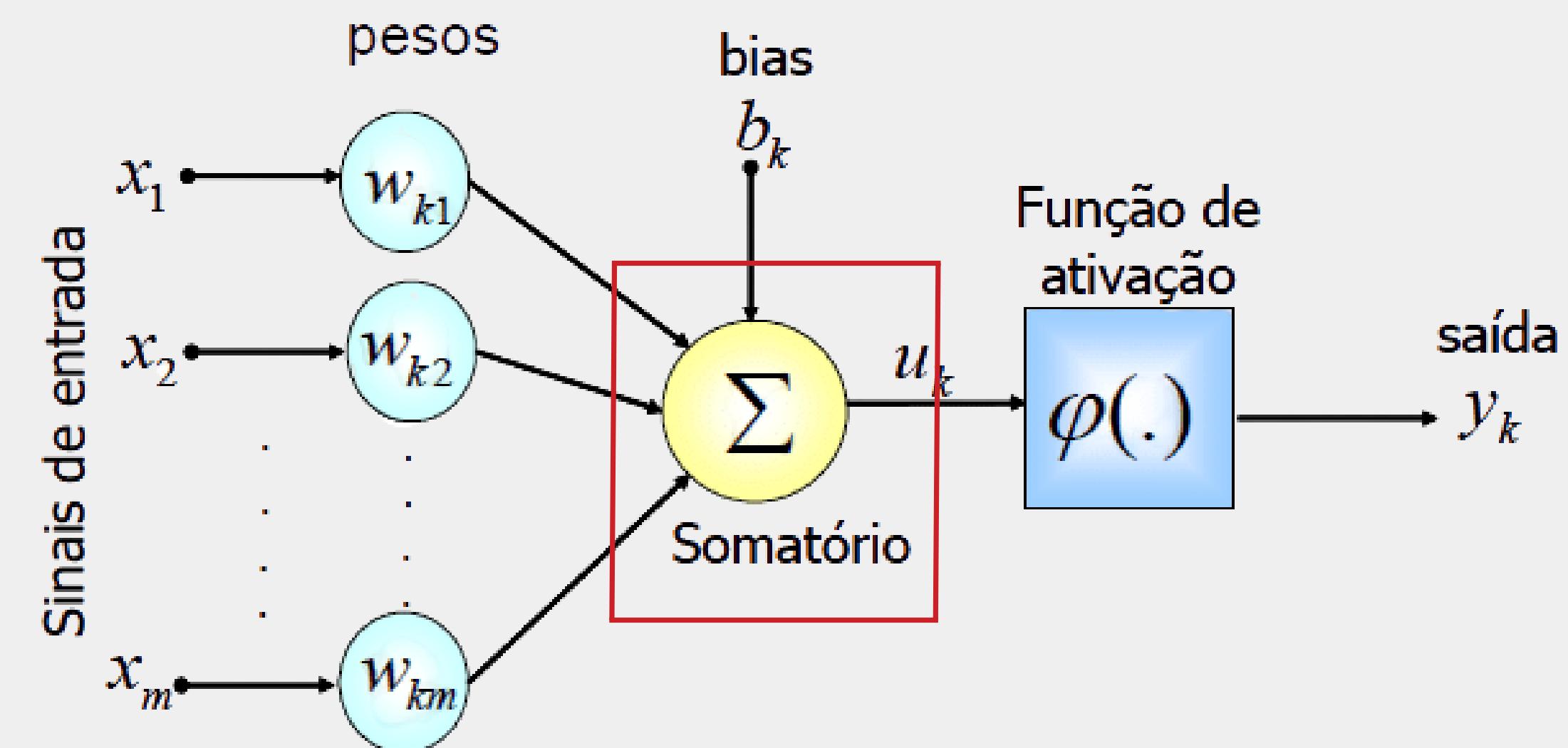
Partes de um neurônio artificial

Somatório(Σ):

Depois, todos esses valores multiplicados (entrada \times peso)
são somados.

Essa soma é o que a gente chama de entrada líquida do
neurônio.

É como se o neurônio estivesse
calculando uma “nota geral”
com base em tudo o que
recebeu.



Partes de um neurônio artificial

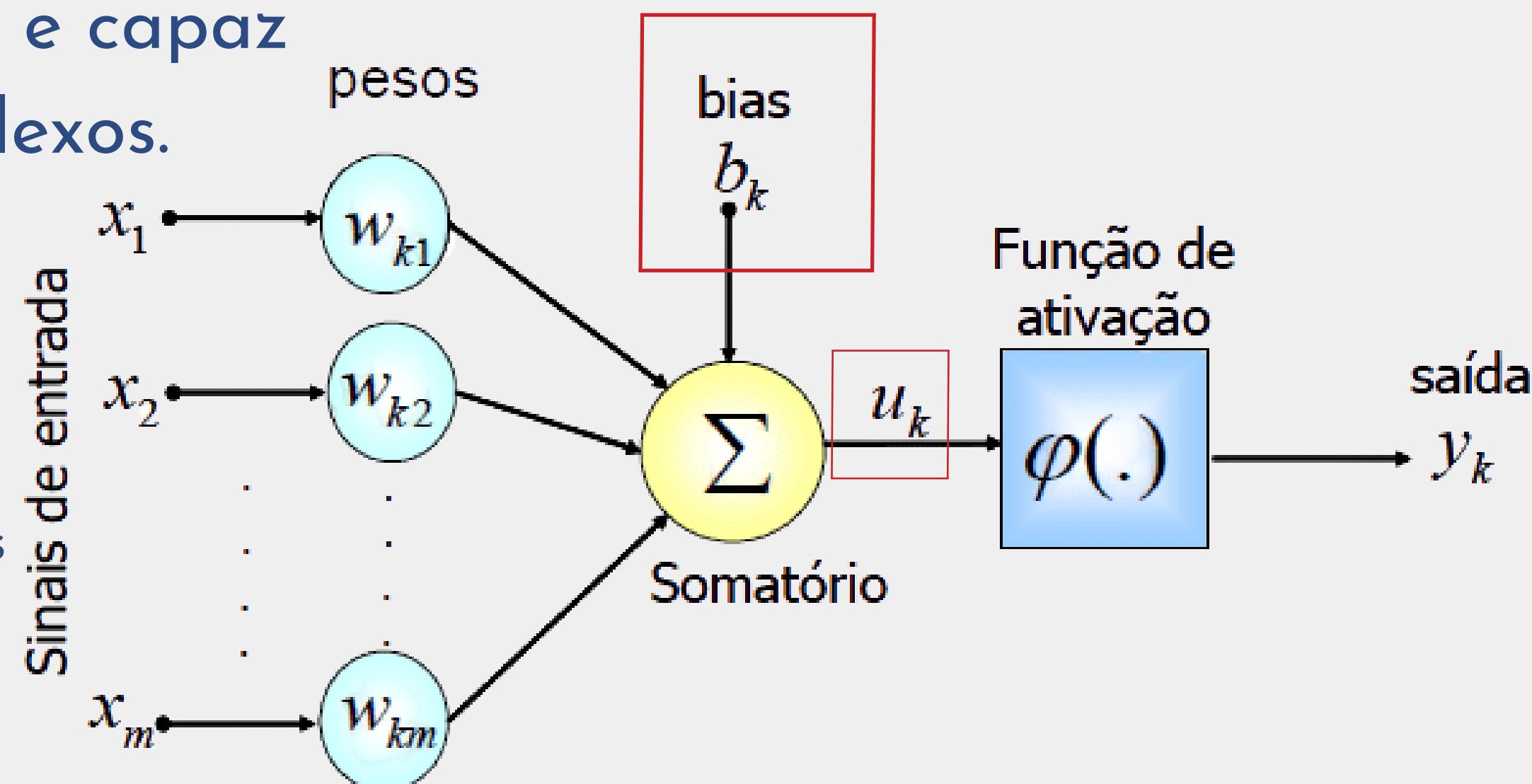
Bias (b):

O bias é um número fixo adicionado à soma. Ele serve para ajustar ou deslocar esse valor final, empurrando ele mais pra cima ou pra baixo.

Isso torna o modelo mais flexível e capaz de aprender padrões mais complexos.

$$U_k = \sum_{j=1}^m w_{kj} \cdot x_j$$

- x_1, x_2, \dots, x_m : sinais de entrada
- $w_{k1}, w_{k2}, \dots, w_{km}$: pesos sinápticos associados ao neurônio
- U_k : saída do somador (combinador linear)

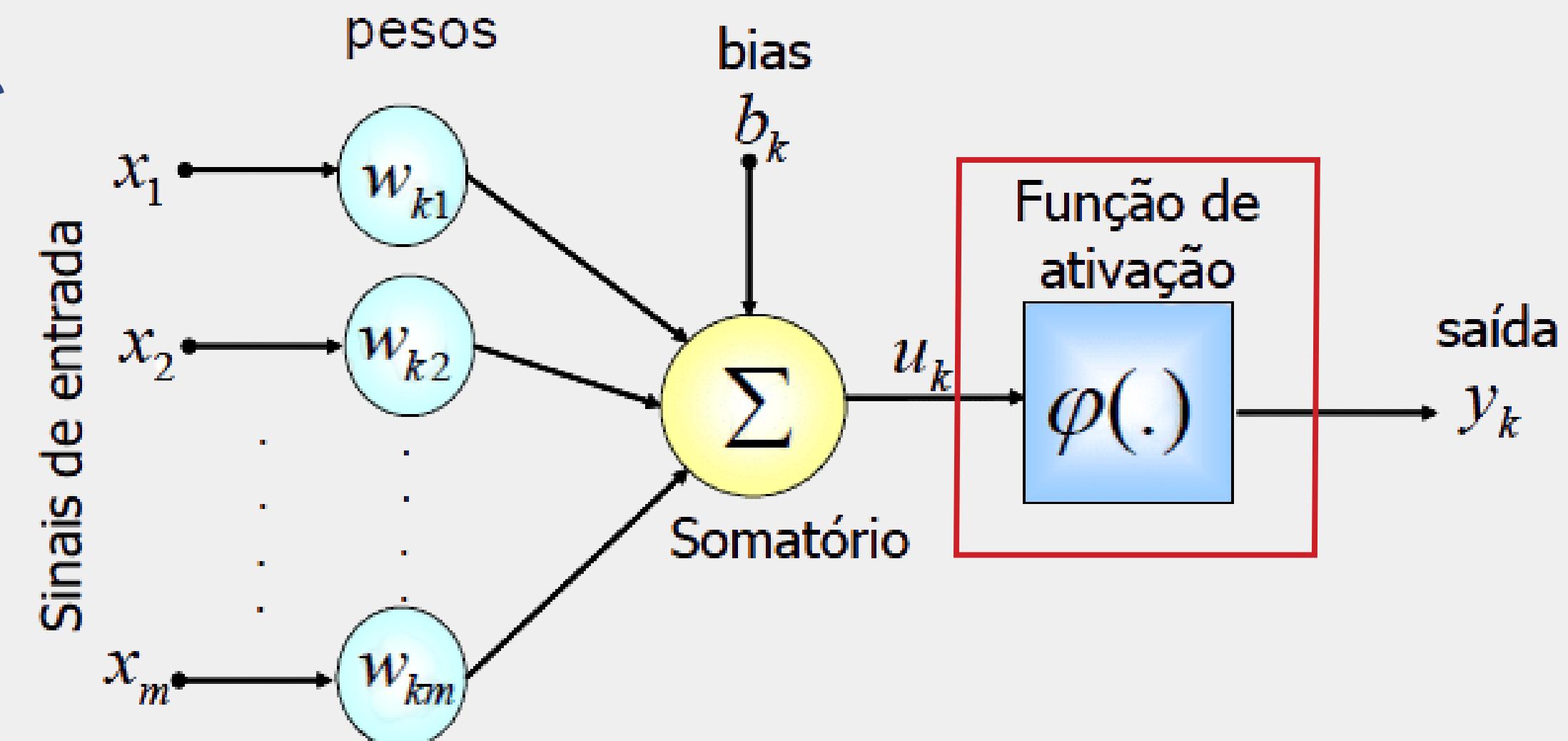


Partes de um neurônio artificial

Função de ativação (φ):

A soma final (com o bias) passa por uma função de ativação, que transforma esse valor em algo mais "controlado".

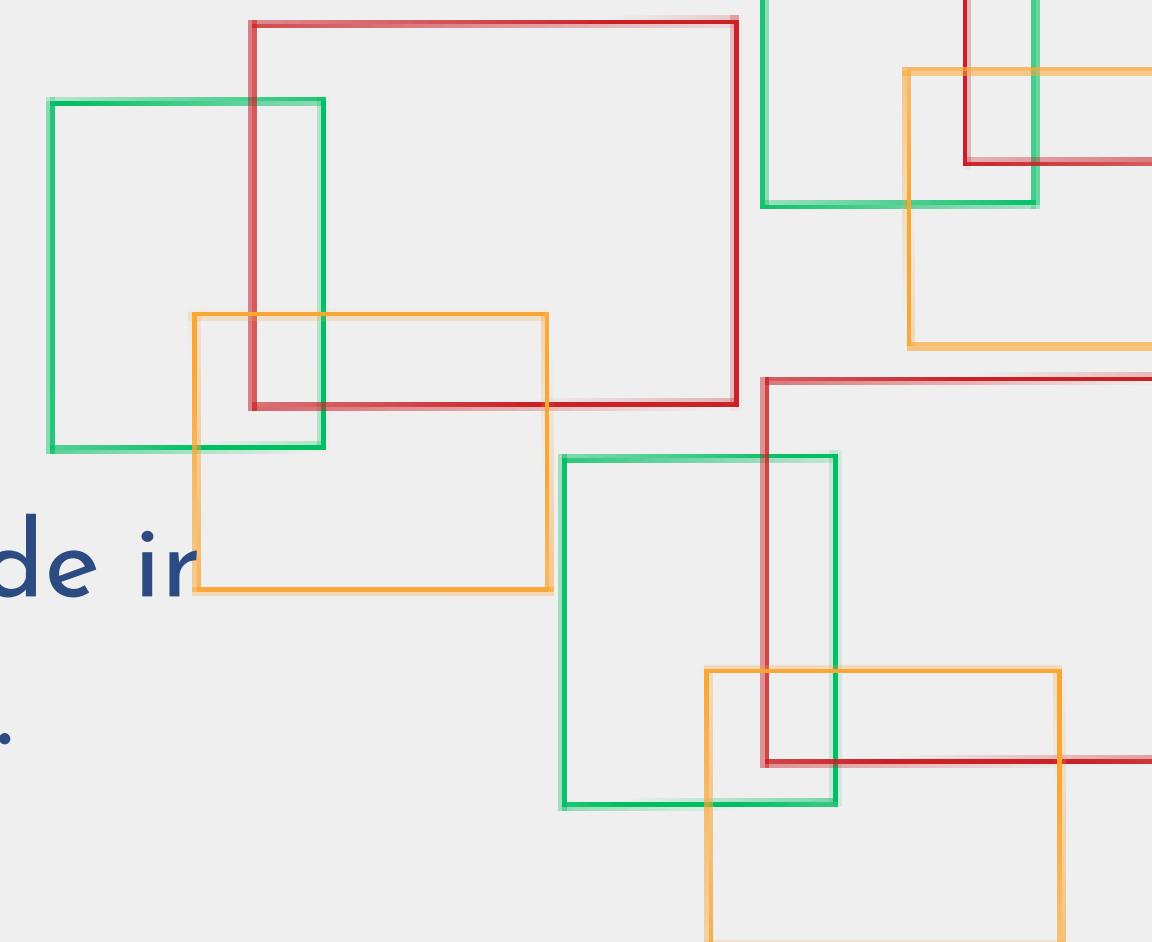
Essa função decide se o neurônio será ativado ou não, e geralmente limita a saída para ficar entre 0 e 1, ou entre -1 e 1. Isso dá estabilidade e permite que a rede funcione de forma mais previsível.



Partes de um neurônio artificial

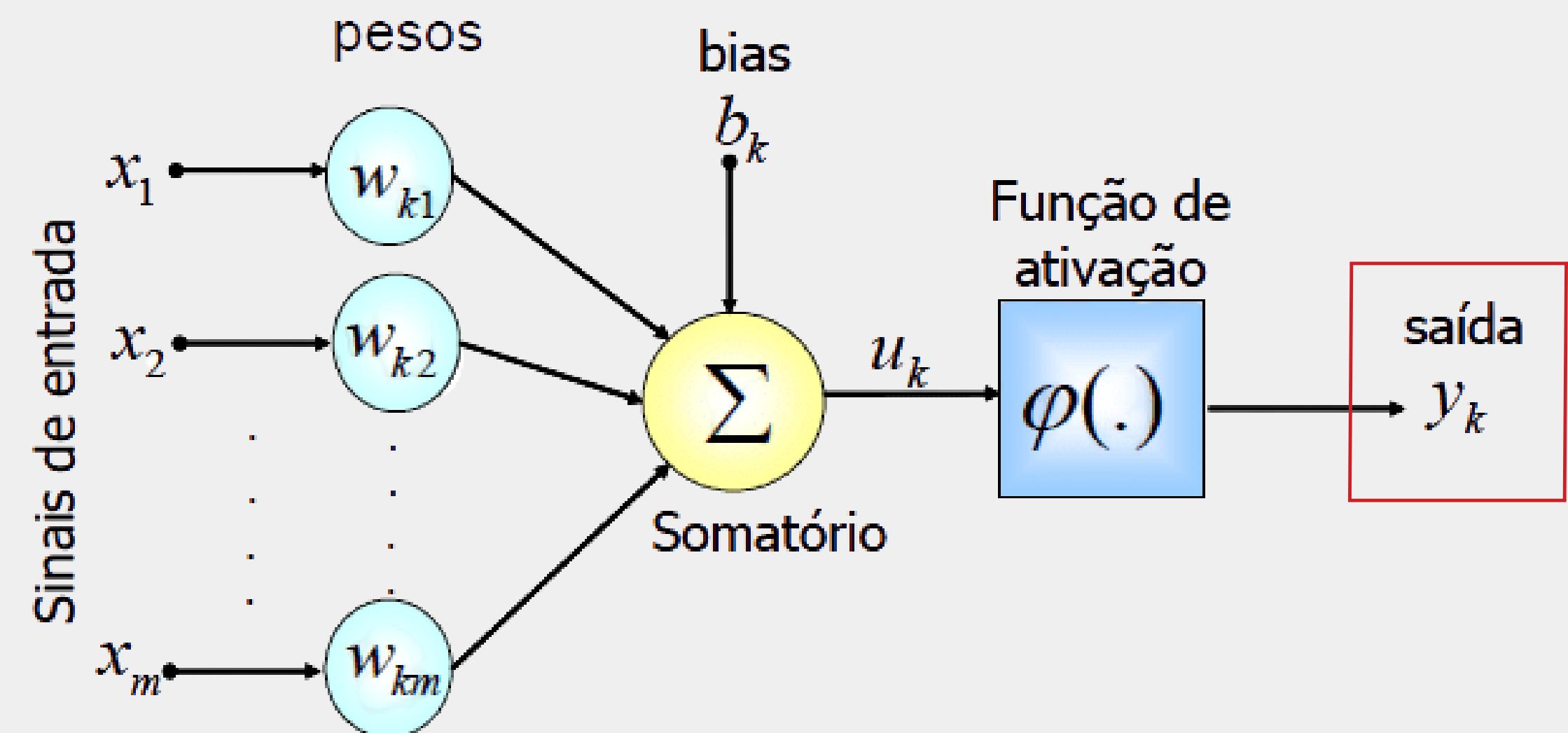
Saída (y):

Depois disso tudo, o neurônio gera uma saída, que pode ir para outros neurônios ou ser o resultado final da rede.



$$y_k = \varphi(U_k + b_k)$$

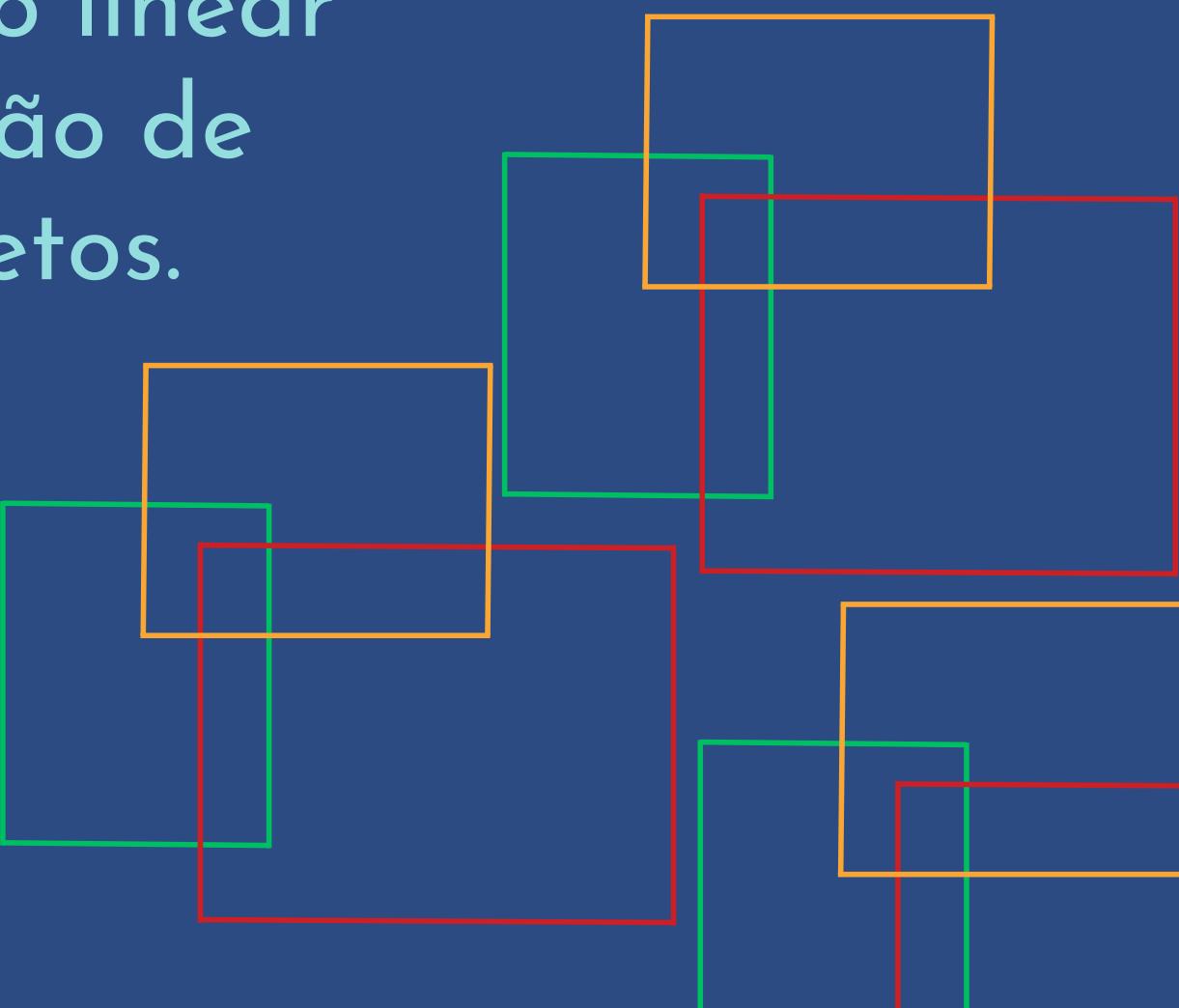
- b_k : bias aplicado ao neurônio
- φ : função de ativação (como ReLU, sigmoide etc.)
- y_k : saída final do neurônio após a ativação



Funções de ativação

As funções de ativação são responsáveis por transformar a saída de um neurônio artificial, tornando-a não linear e permitindo que a rede aprenda padrões mais complexos.

Sem elas, uma rede neural seria apenas um modelo linear e não conseguiria resolver tarefas como classificação de imagens, tradução automática ou detecção de objetos.

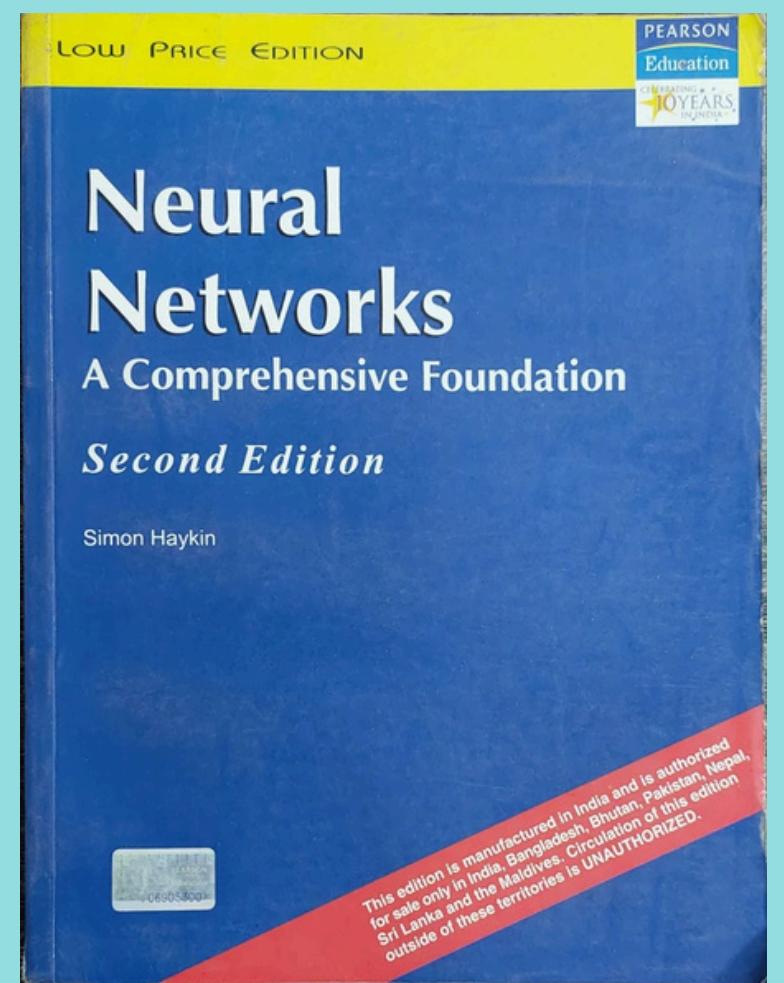


Simon Haykin (1931 - 2025) foi um engenheiro, professor e pesquisador canadense amplamente reconhecido na área de **redes neurais artificiais e **processamento de sinais**.**



Simon Haykin (1931- 2025)

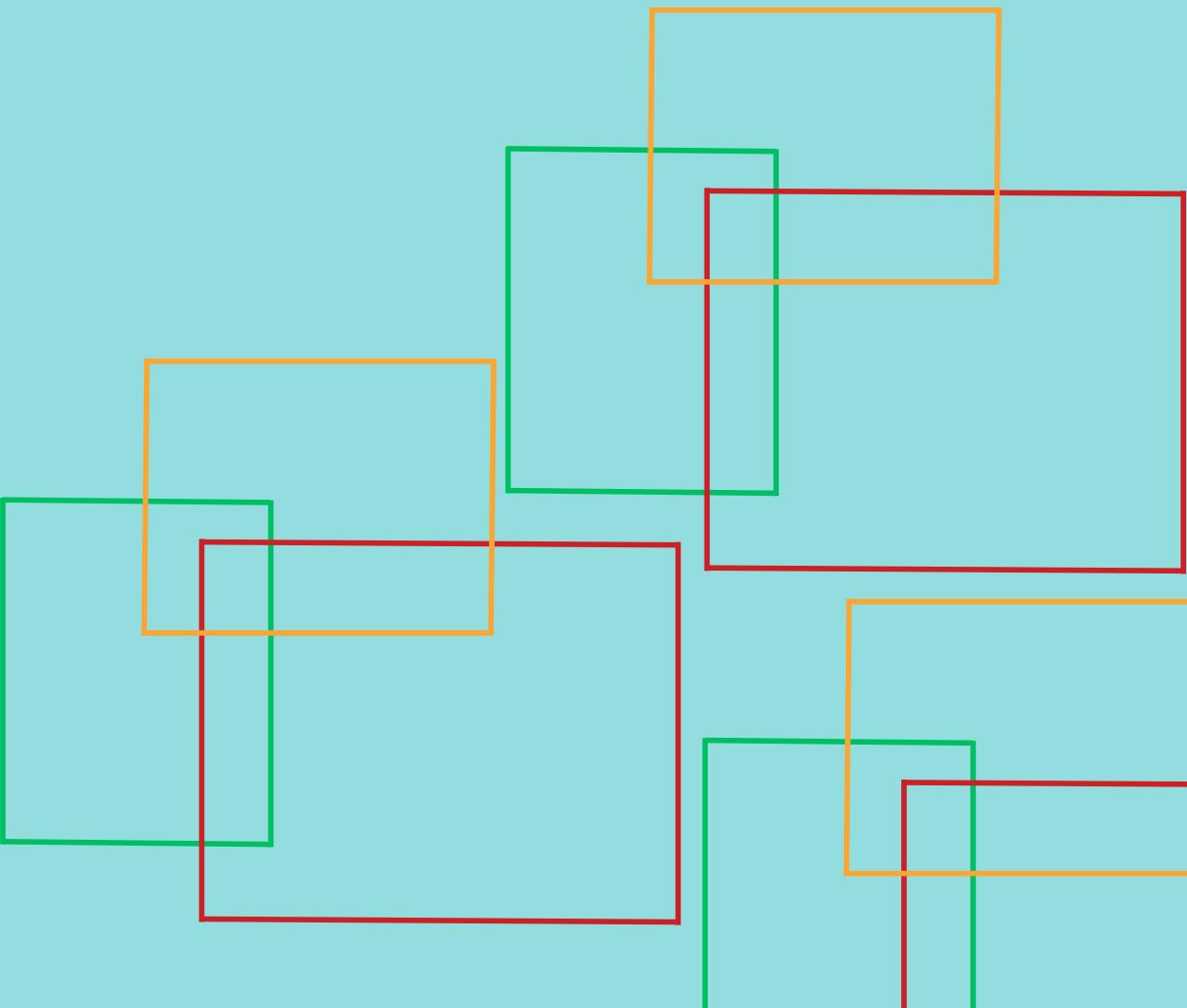
Em seu livro “**Neural Networks: A Comprehensive Foundation**”, especialmente na 2ª edição (1998), ele apresenta três funções de ativação fundamentais para entender o comportamento das redes.



Neural Networks, 2nd Edition, 1994

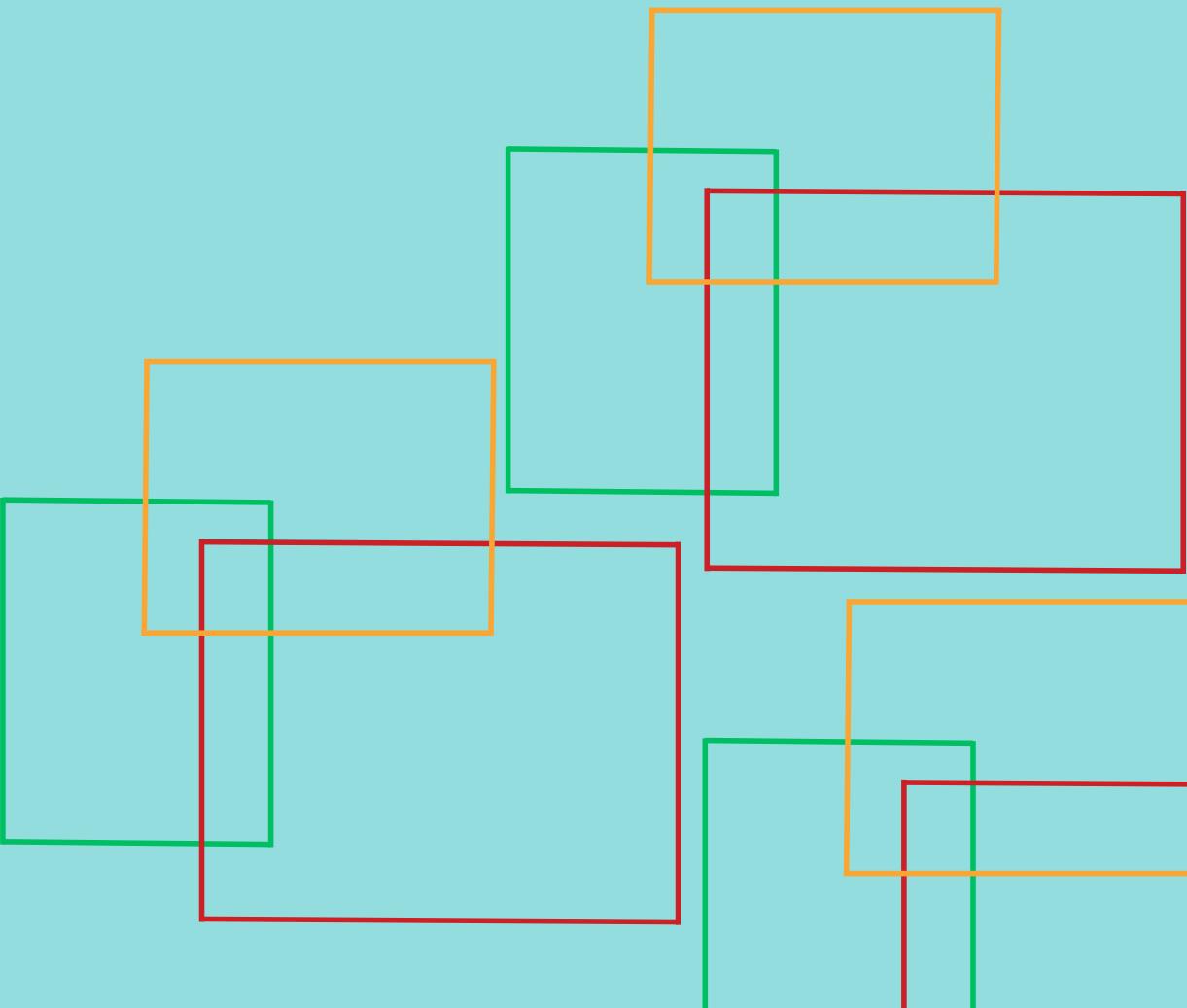
Funções de ativação segundo Simon Haykin (1998):

- Função de Limiar (Degrau)
- Função Linear por Partes
- Função Sigmóide



Funções de ativação segundo Simon Haykin (1998):

- Função de Limiar (Degrau)
- Função Linear por Partes
- Função Sigmóide

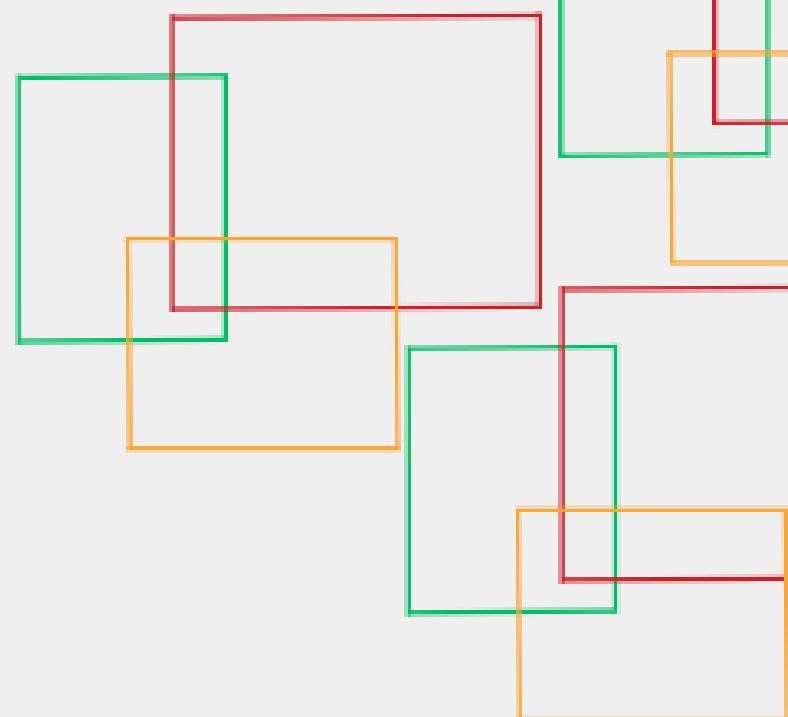


Função de Limiar (Degrau)

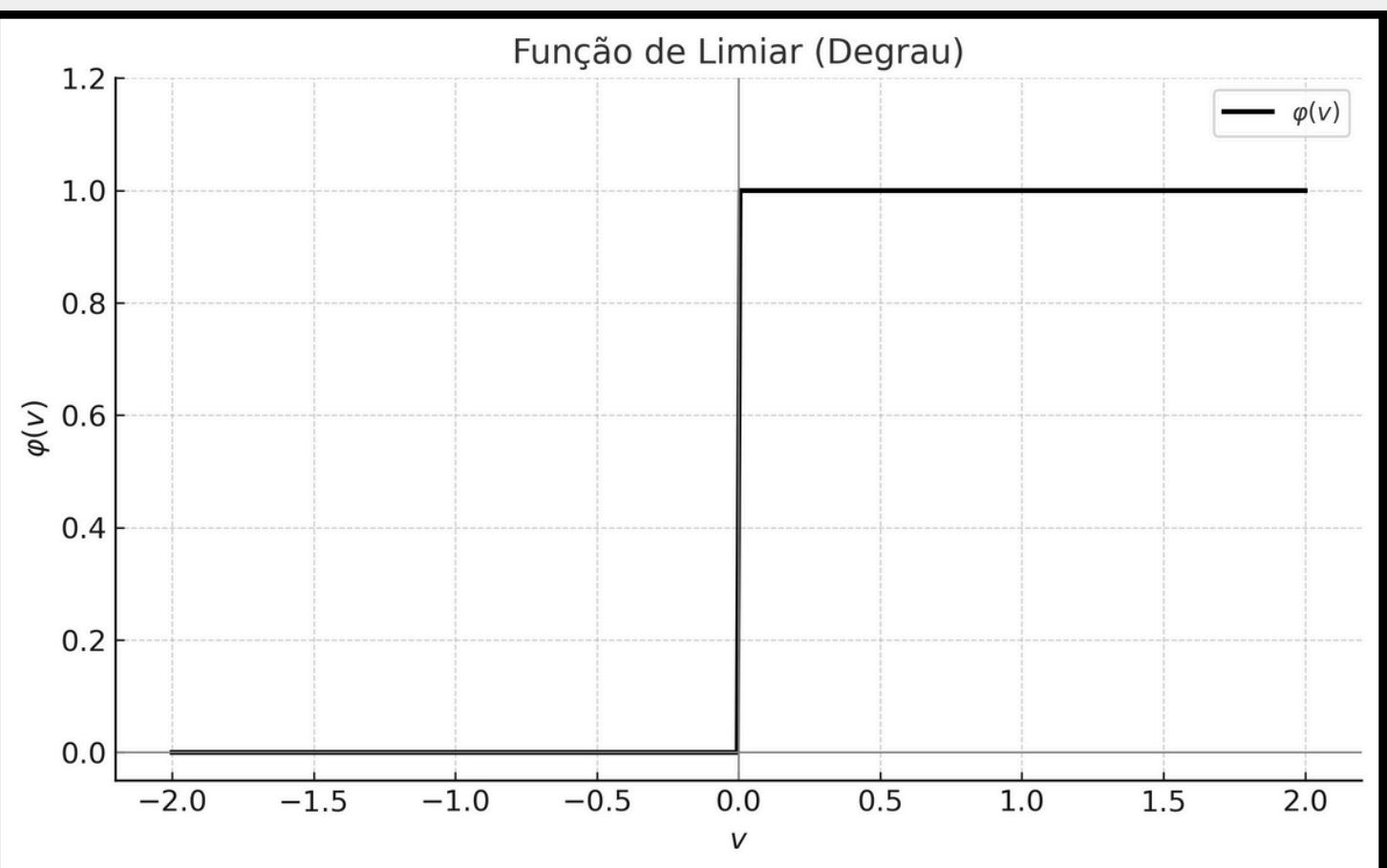
É a mais simples das funções de ativação. Ela produz apenas dois valores, funcionando como um interruptor digital

Se a entrada for maior que um valor limite estabelecido (o limiar), a saída é 1; caso contrário, é 0.

Foi amplamente utilizada nos primeiros modelos de redes neurais (Perceptron) mas sua falta de diferenciabilidade impede o uso em algoritmos mais complexos (backpropagation)

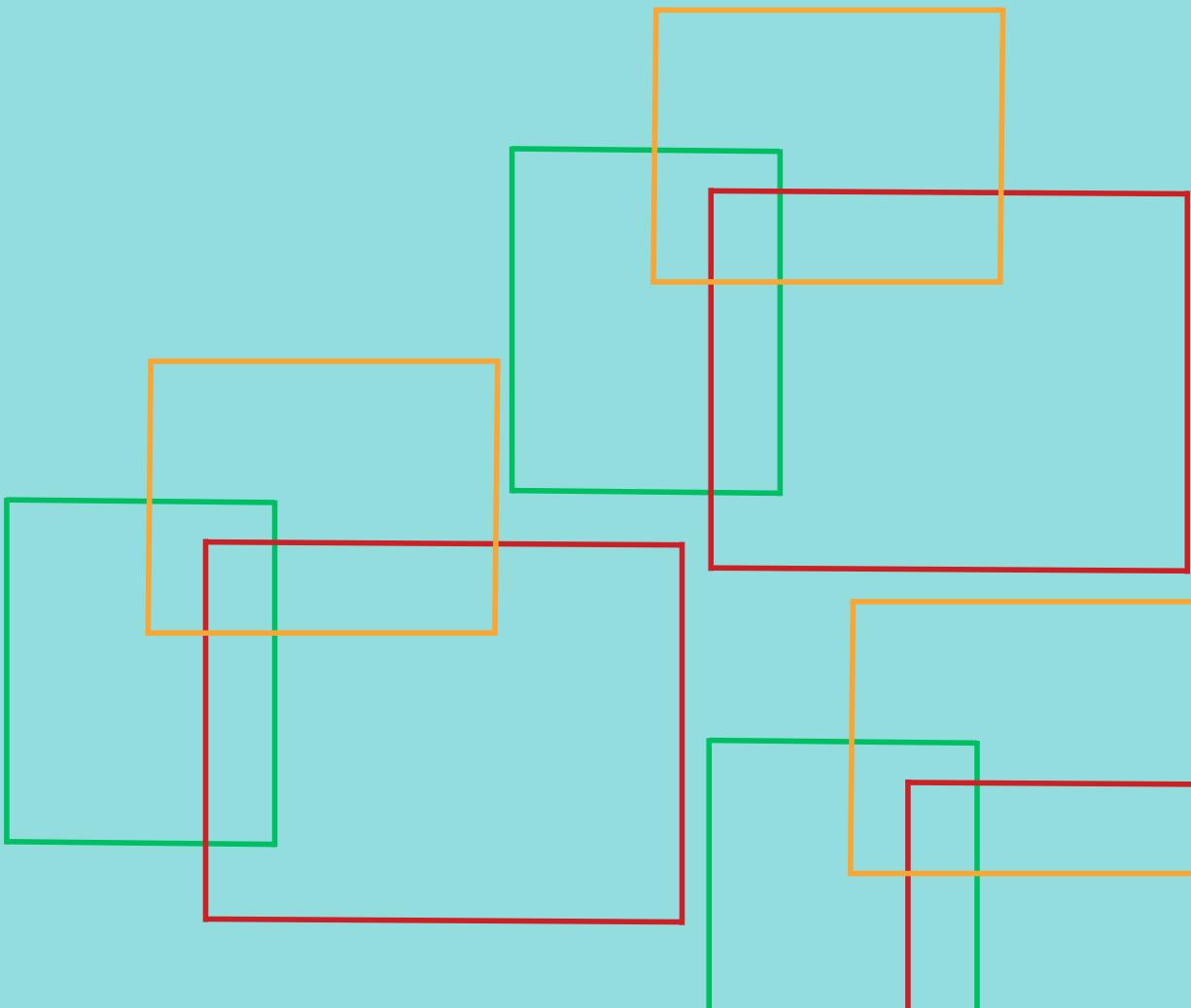


$$\varphi(v) = \begin{cases} 1, & \text{se } v \geq 0 \\ 0, & \text{se } v < 0 \end{cases}$$



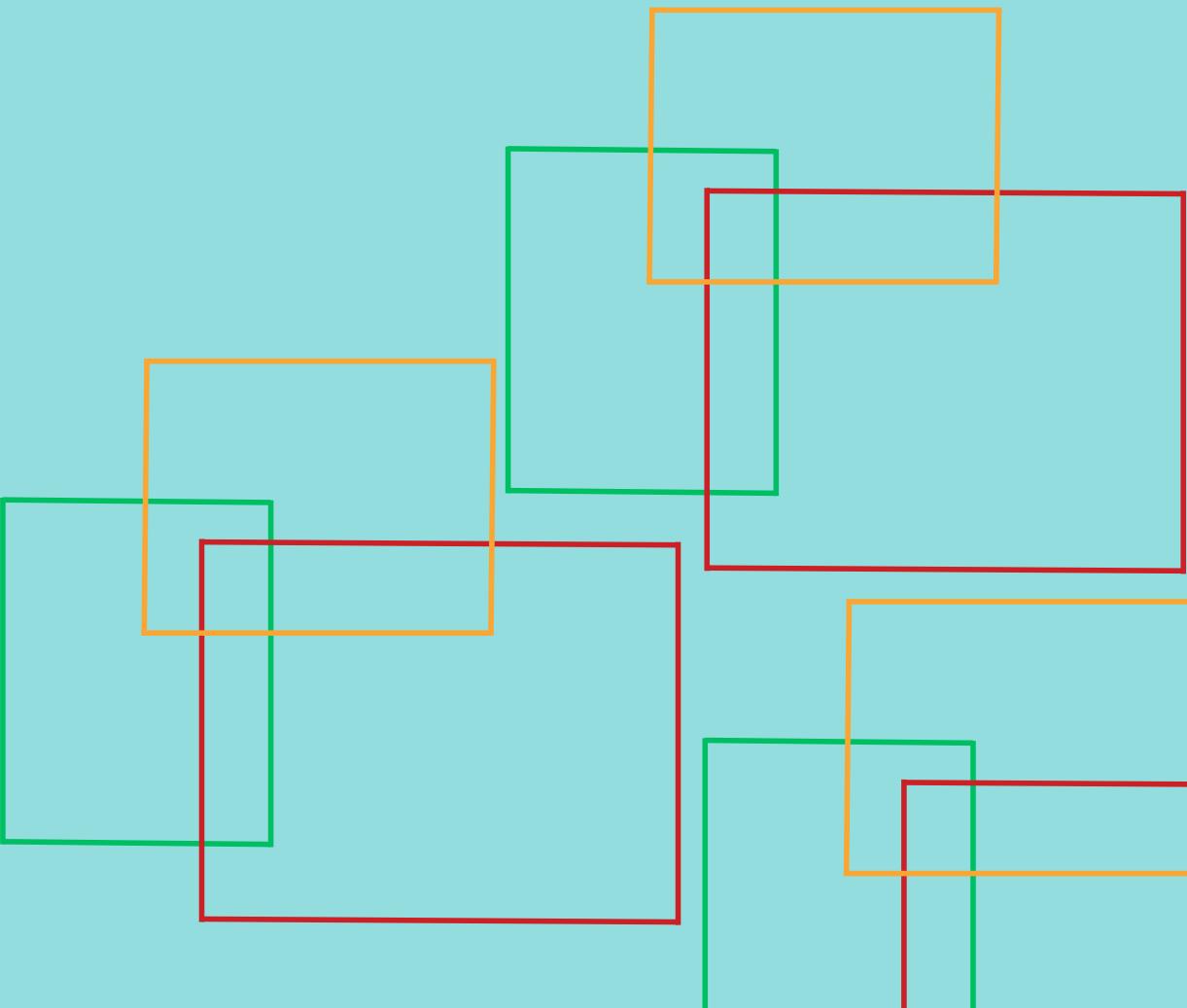
Funções de ativação segundo Simon Haykin (1998):

- Função de Limiar (Degrau)
- Função Linear por Partes
- Função Sigmóide



Funções de ativação segundo Simon Haykin (1998):

- Função de Limiar (Degrau)
- Função Linear por Partes
- Função Sigmóide



Função Linear por Partes

Representa um meio-termo entre simplicidade e funcionalidade.

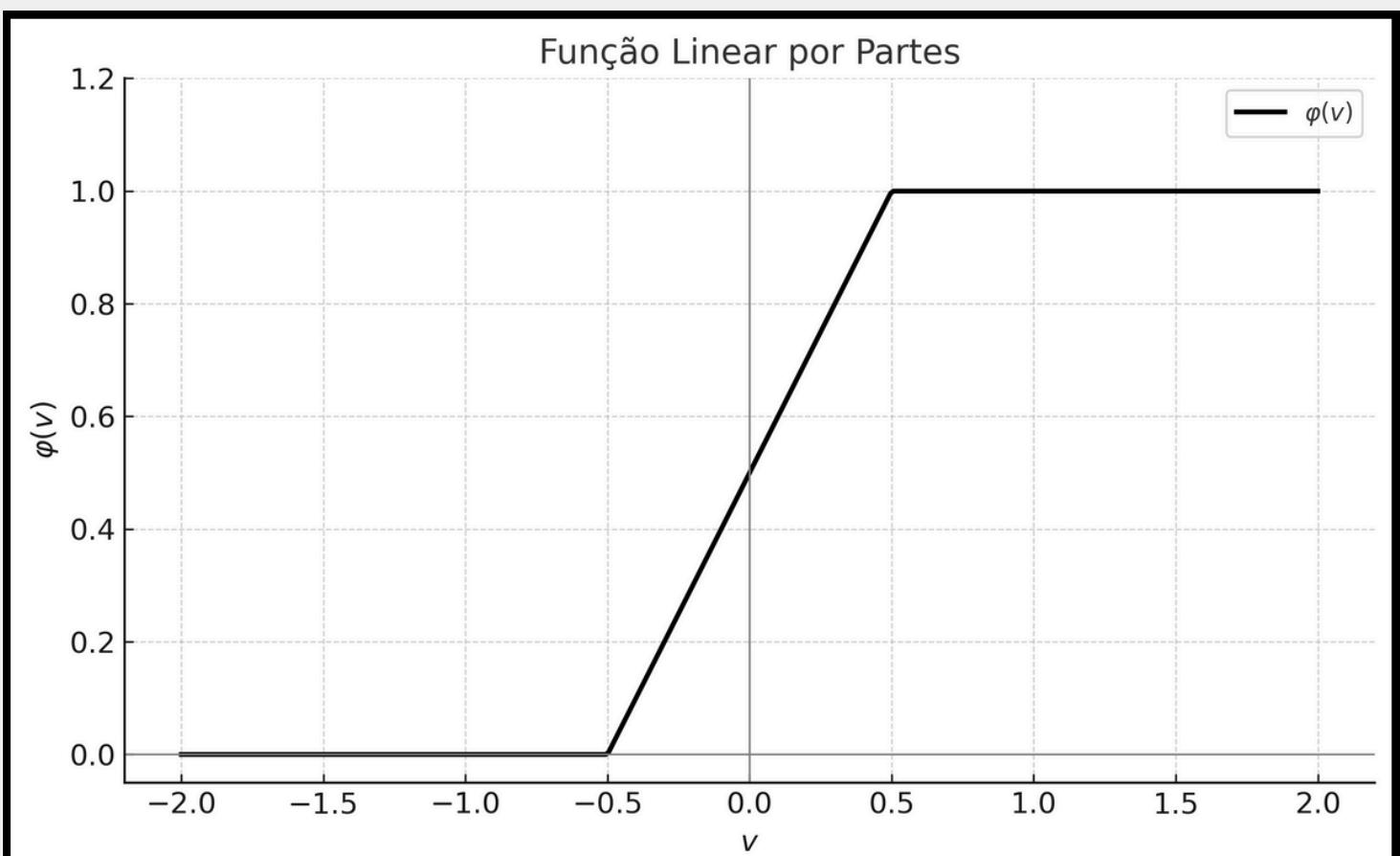
Ela possui três regiões distintas de comportamento:

- Para entradas muito baixas, a saída é 0 (saturação inferior)
- Na faixa intermediária, a saída cresce linearmente com a entrada
- Para entradas muito altas, a saída se estabiliza em 1 (saturação superior)

Ela é diferenciável na região linear central, tornando-a compatível com algoritmos mais complexos como o backpropagation.

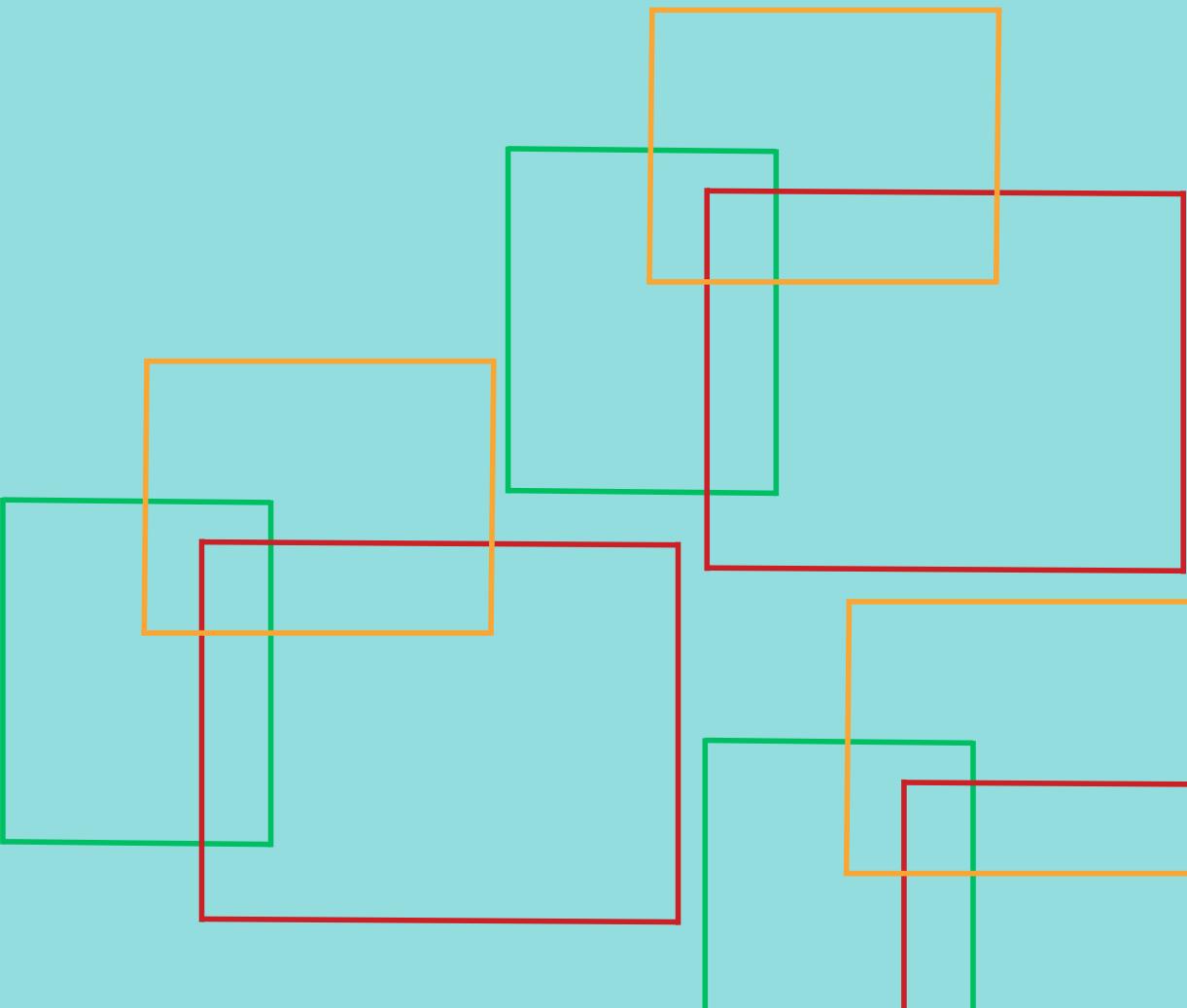


$$\varphi(v) = \begin{cases} 1, & v \geq +\frac{1}{2} \\ v, & +\frac{1}{2} > v > -\frac{1}{2} \\ 0, & v \leq -\frac{1}{2} \end{cases}$$



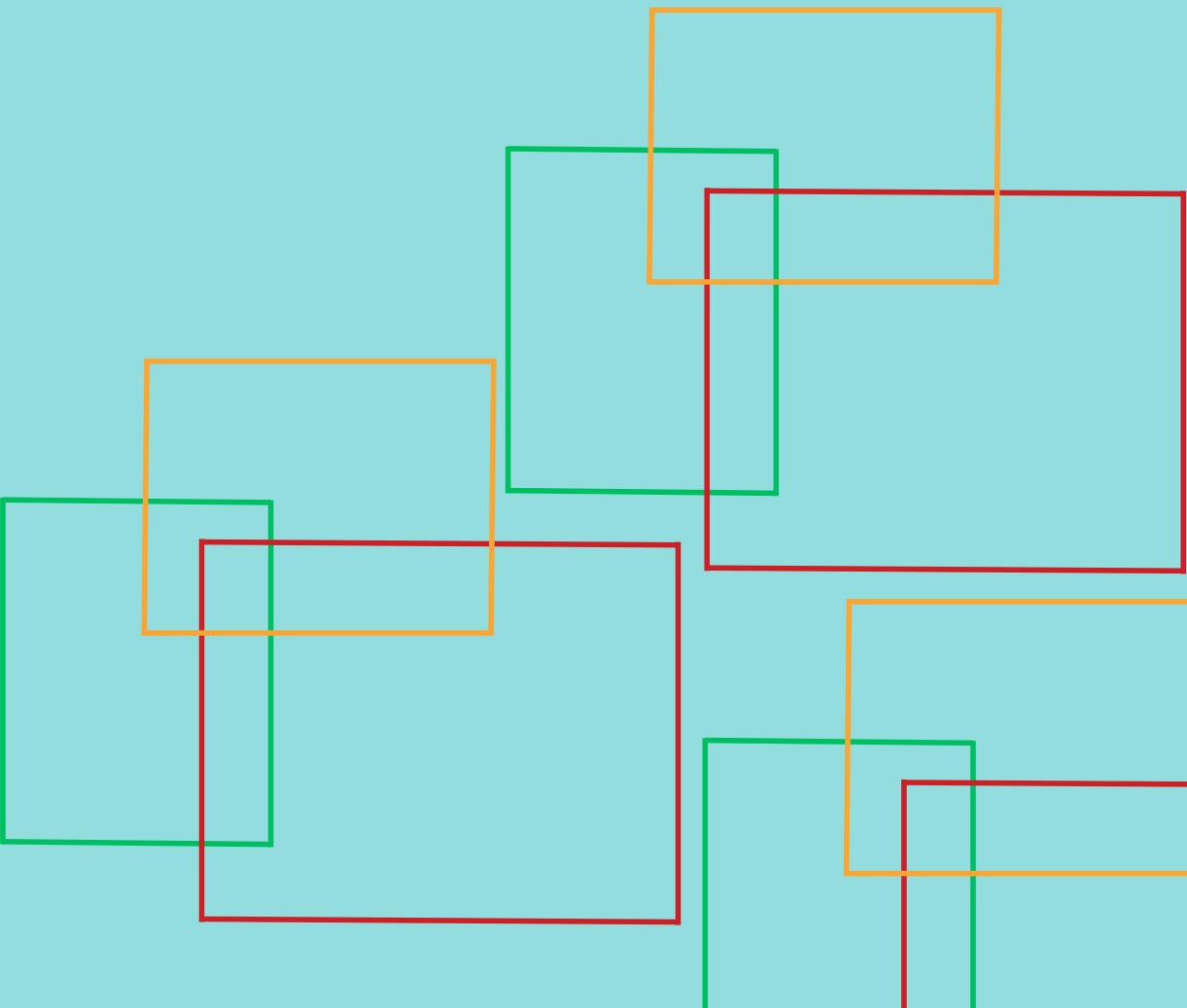
Funções de ativação segundo Simon Haykin (1998):

- Função de Limiar (Degrau)
- Função Linear por Partes
- Função Sigmóide



Funções de ativação segundo Simon Haykin (1998):

- Função de Limiar (Degrau)
- Função Linear por Partes
- Função Sigmóide

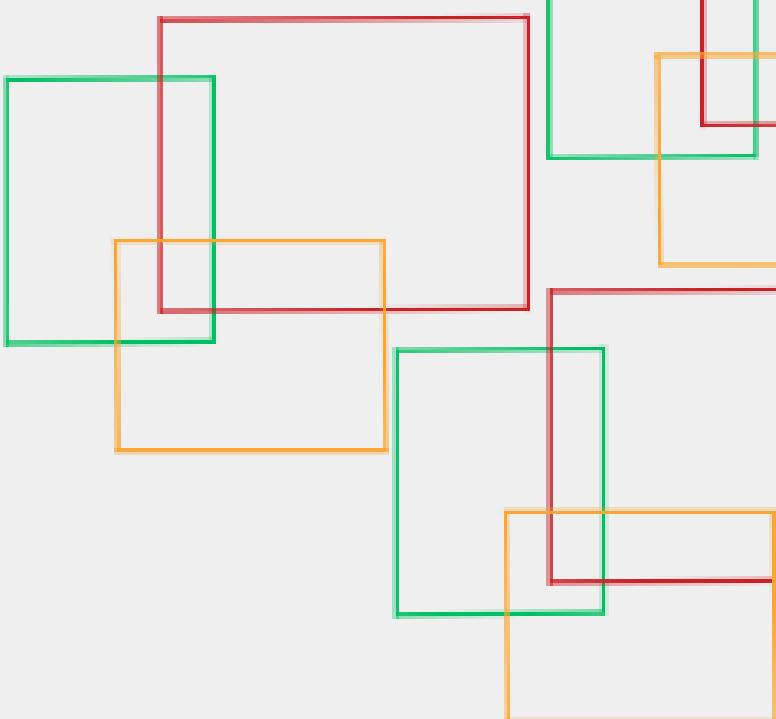


Função Sigmóide

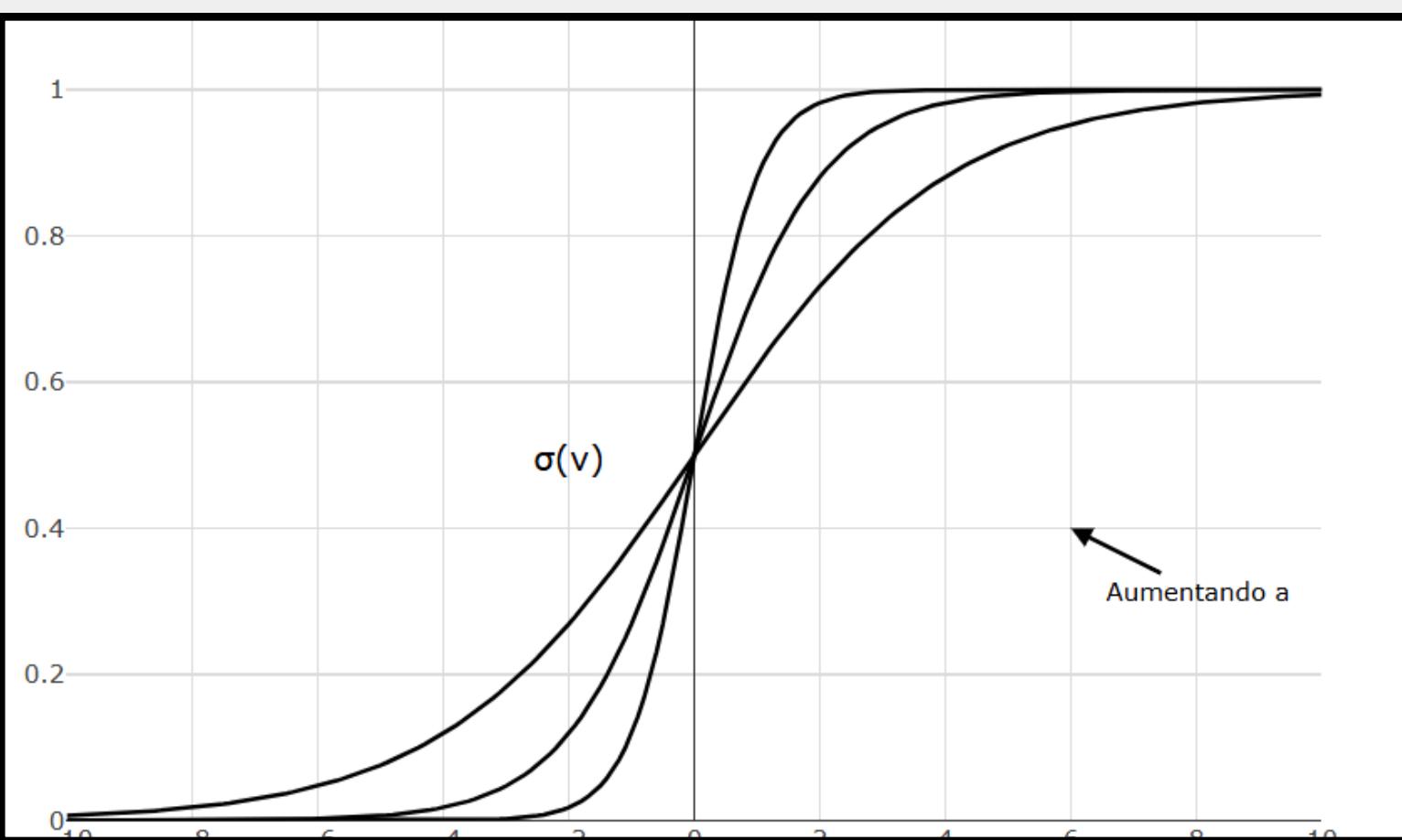
Tem um formato em “S” e transforma qualquer valor real em um número entre 0 e 1. Ela oferece uma transição suave entre os extremos e é diferenciável em todos os pontos, tornando-a compatível com o algoritmo backpropagation.

Foi amplamente utilizada em redes neurais artificiais, especialmente em tarefas de classificação binária, por gerar saídas interpretáveis como probabilidades.

Tende a saturar nos extremos, dificultando o treinamento de redes profundas.



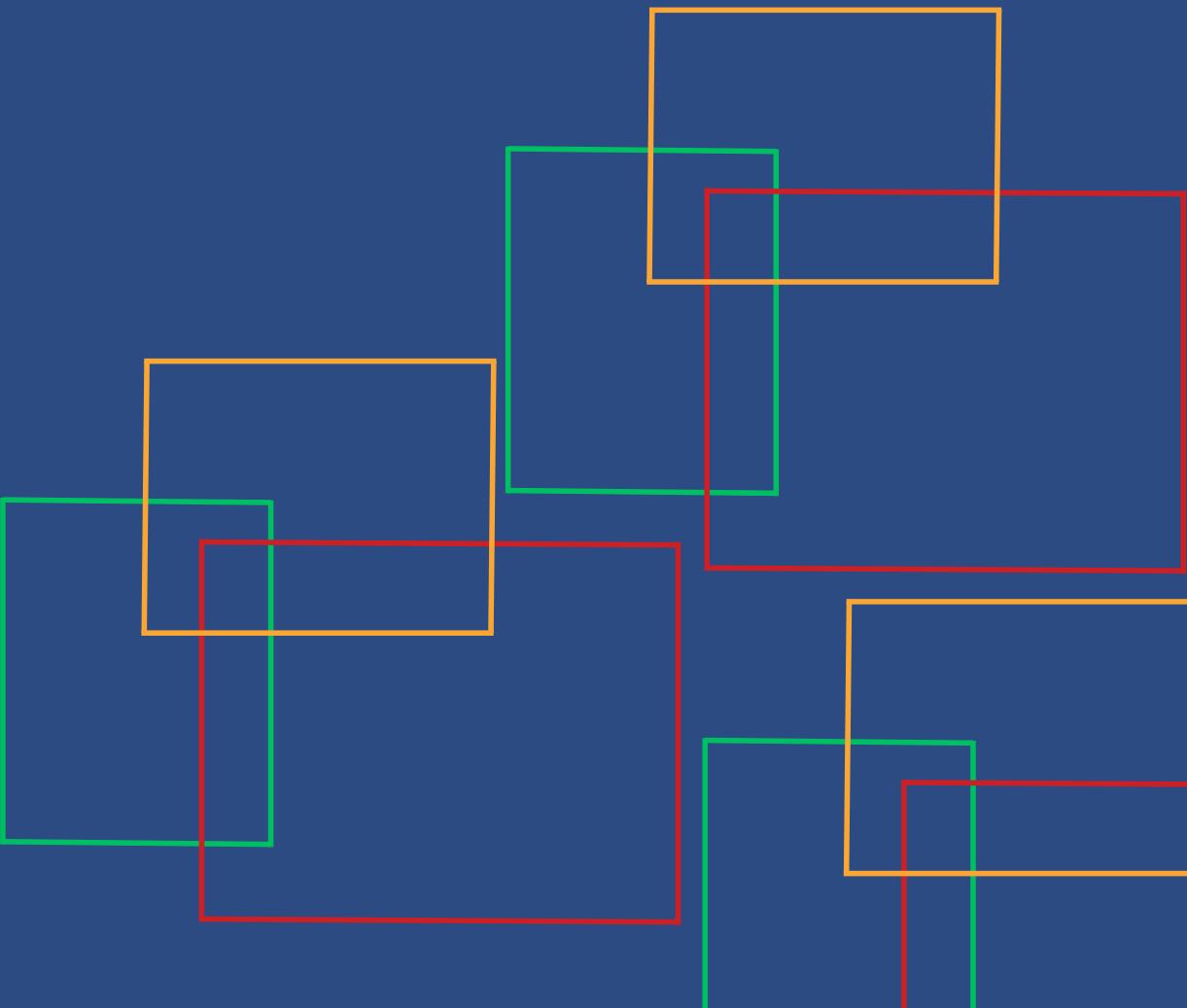
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



Evolução das Funções de Ativação

Com a chegada das redes profundas (deep learning), novas funções de ativação foram propostas para resolver problemas como o desvanecimento do gradiente ou melhorar a velocidade de treinamento. Algumas delas:

- ReLU (Rectified Linear Unit):
 - Retorna 0 para valores negativos e o valor original para positivos. Amplamente usada em CNNs e RNNs.
- Leaky ReLU:
 - Variante da ReLU que permite uma pequena inclinação mesmo para valores negativos.
- Tanh (Tangente hiperbólica):
 - Saída entre -1 e 1, centrada em zero.
- Softmax:
 - Transforma os valores em probabilidades que somam 1, usada em classificação com múltiplas classes.



**Como as redes
neurais aprendem?**

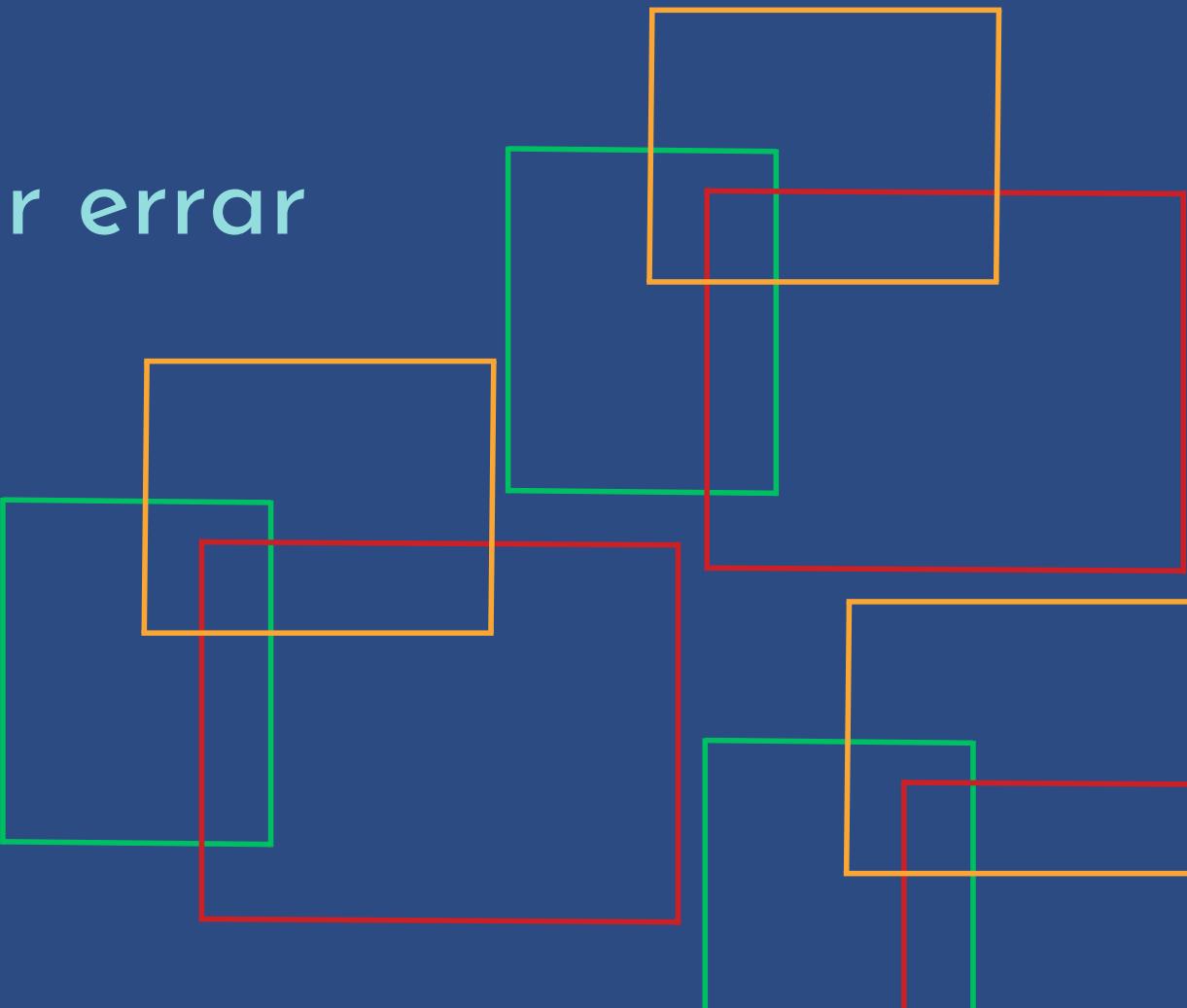
Aprendizado Supervisionado

As redes neurais geralmente aprendem por aprendizado supervisionado, que é quando a rede recebe entradas com saídas conhecidas (rótulos).

A cada exemplo, a rede gera uma resposta, compara com o valor esperado e calcula o erro

Em seguida, ela ajusta os pesos internos para tentar errar menos na próxima vez.

Esse ciclo de tentativa, erro e correção se repete milhares de vezes até que a rede aprenda os padrões.



Como a rede neural aprende na prática

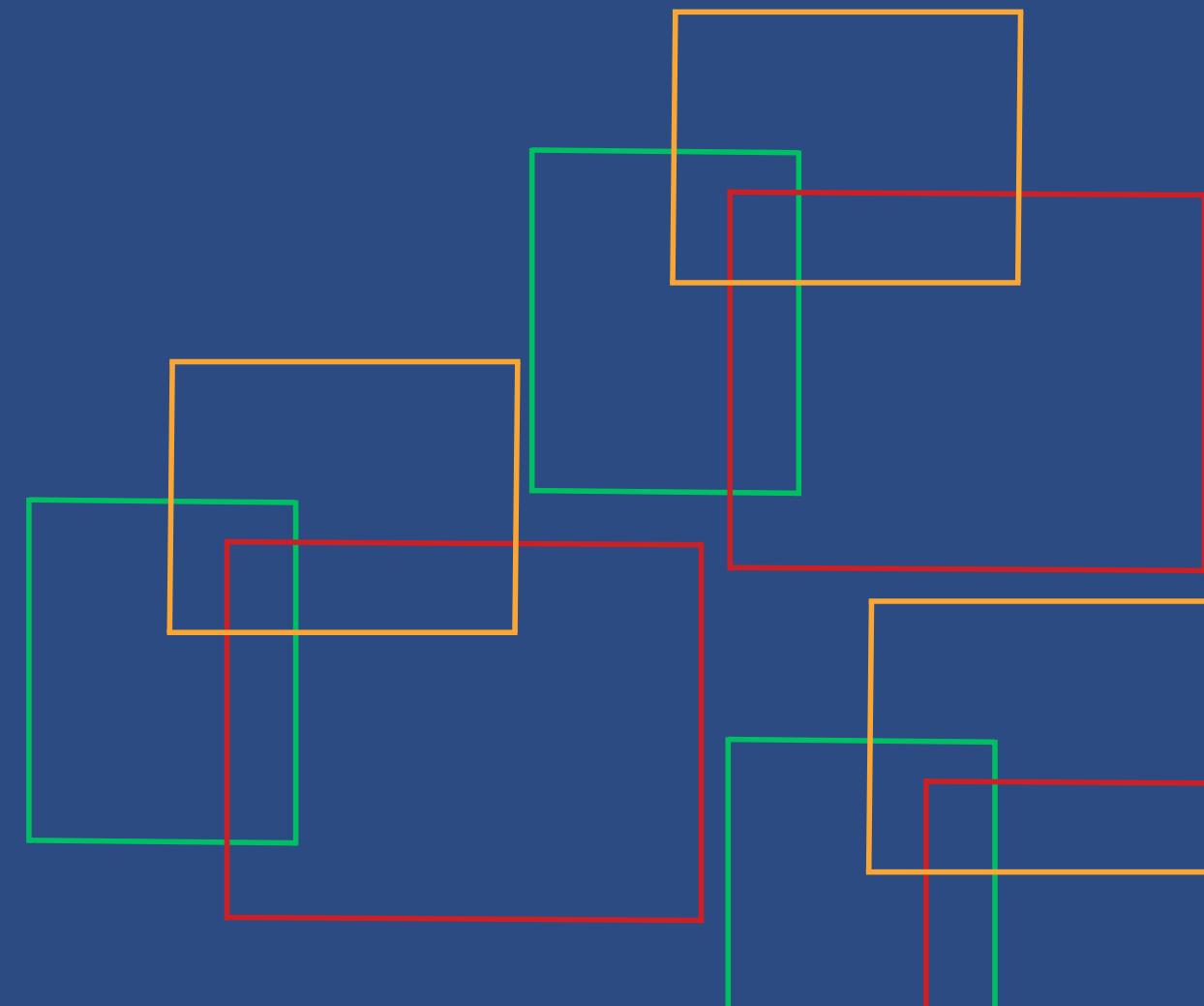
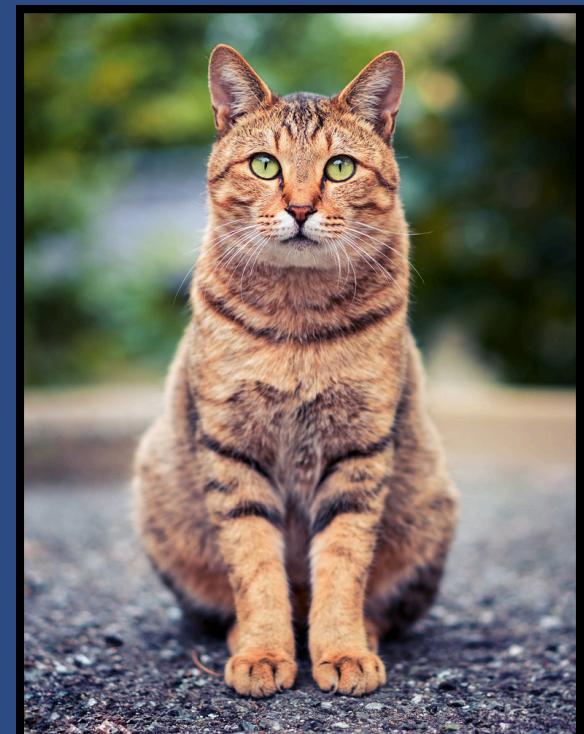
O treinamento de uma rede neural segue este ciclo:

1. Entrada de dados:

Pode ser uma imagem, um vetor numérico ou qualquer tipo de informação que a rede deve processar.

Exemplo entrada: imagem ou vetor

0	1	2	3	4
4	7	2	0	6



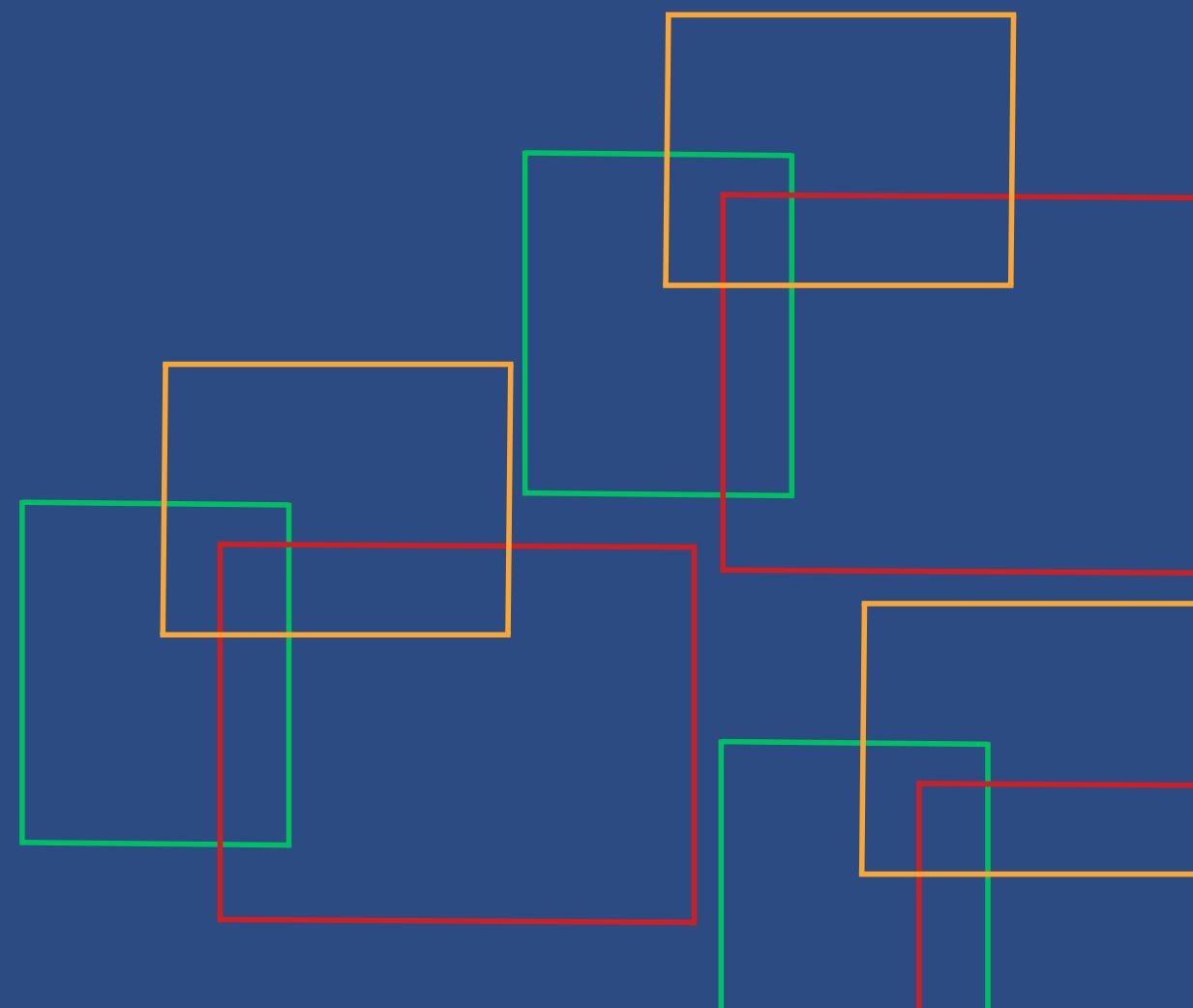
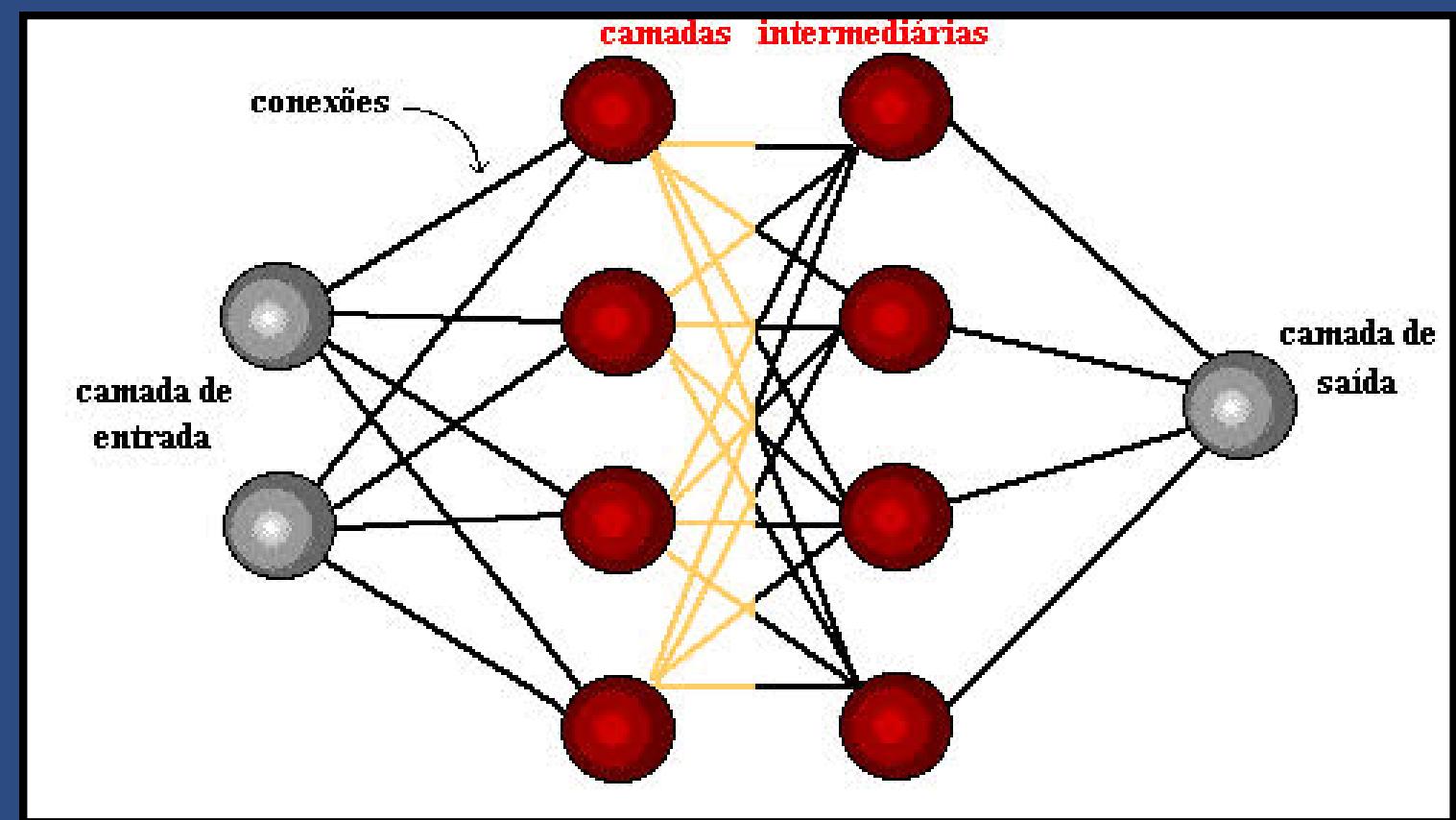
Como a rede neural aprende na prática

O treinamento de uma rede neural segue este ciclo:

2. Propagação direta (Forward):

A entrada passa pelas camadas da rede neural.

Cada camada transforma os dados aplicando pesos e funções de ativação, até gerar uma saída final.



Como a rede neural aprende na prática

O treinamento de uma rede neural segue este ciclo:

3. Cálculo do erro:

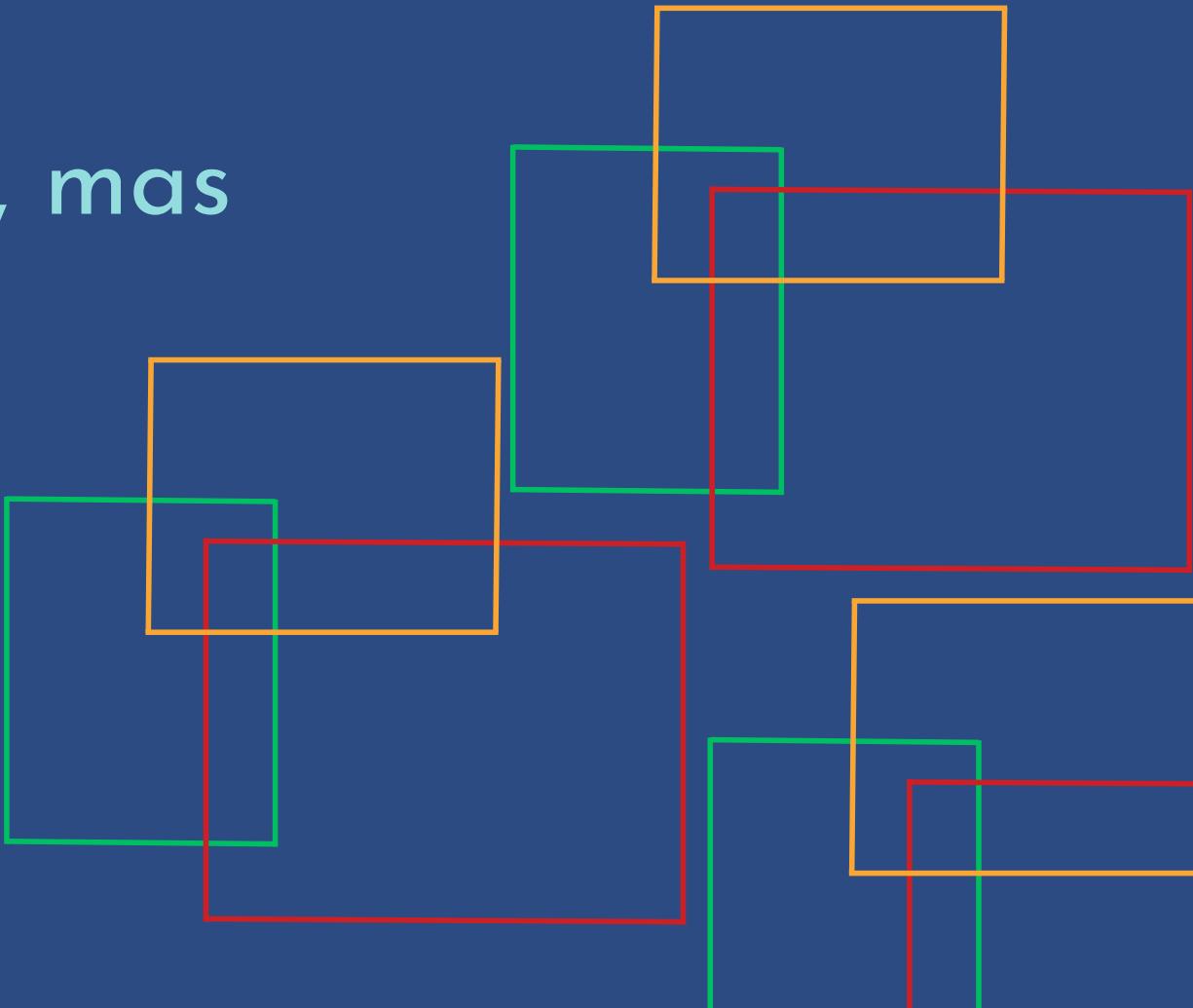
A saída da rede é comparada com o valor real (o rótulo do exemplo).

A diferença entre os dois é o erro, que indica o quanto a rede errou.

Exemplo:

A rede diz que é um cachorro com 70% de certeza, mas era um gato.

$$\text{Erro} = 100\% - 70\% = 30\%$$



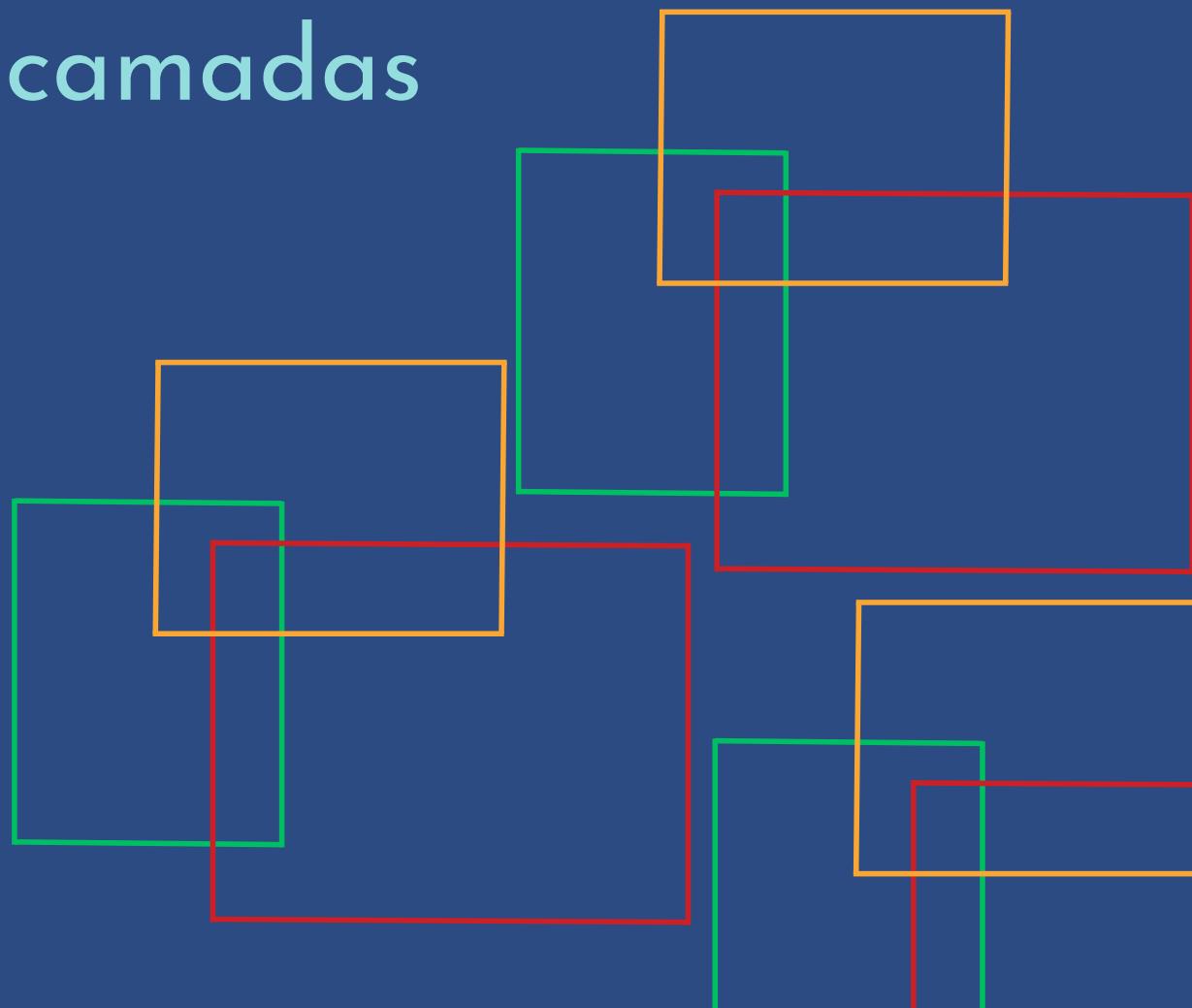
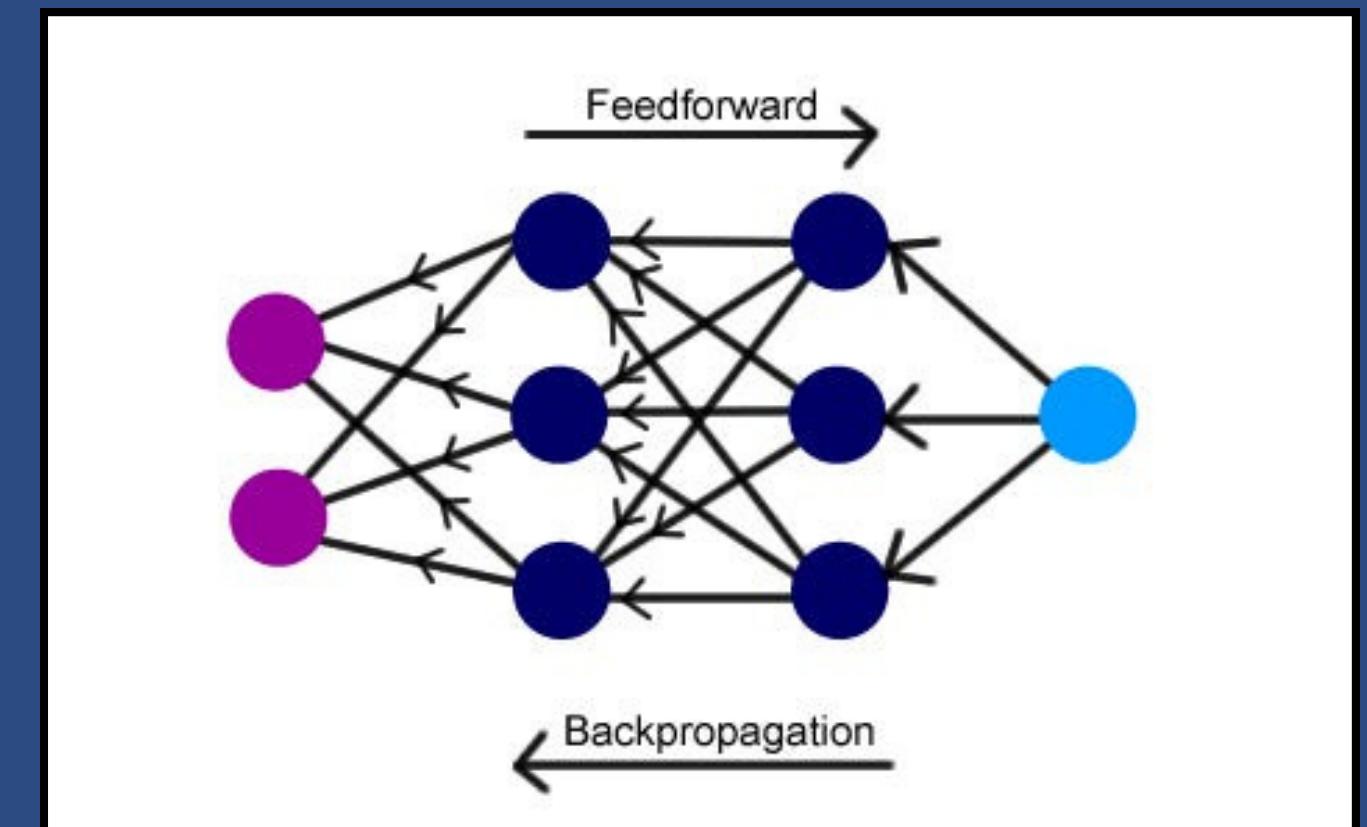
Como a rede neural aprende na prática

O treinamento de uma rede neural segue este ciclo:

4. Retropropagação do erro (backpropagation)

Backpropagation é o algoritmo que permite às redes neurais aprenderem com seus erros. Ele ajusta os pesos "caminhando para trás" pelas camadas, determinando quanto cada neurônio contribuiu para o erro.

Sem ele, o treinamento eficiente de redes com multicamadas seria impossível.



Como a rede neural aprende na prática

O treinamento de uma rede neural segue este ciclo:

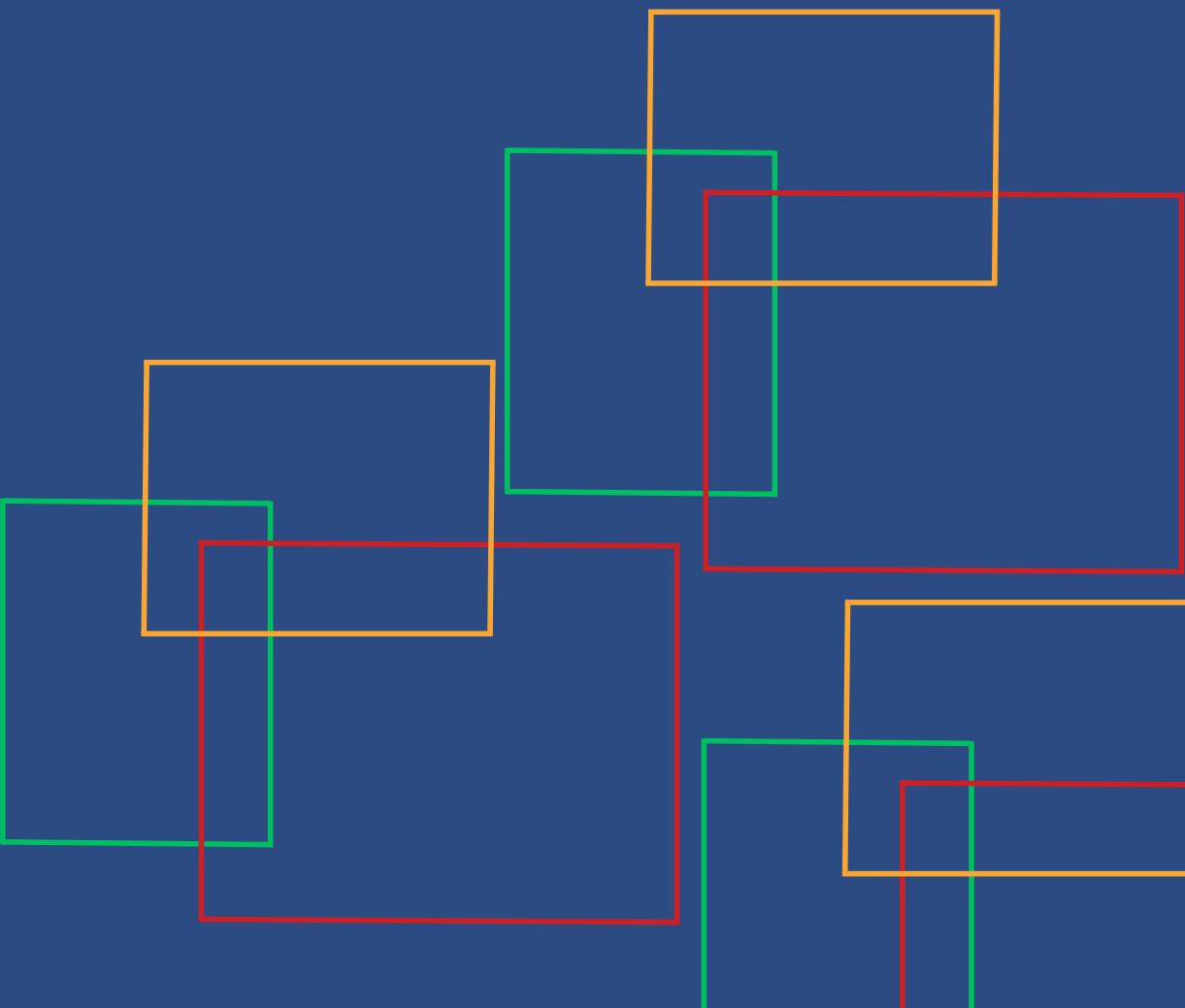
5. Repetição:

O processo é repetido com novos exemplos, centenas ou milhares de vezes.

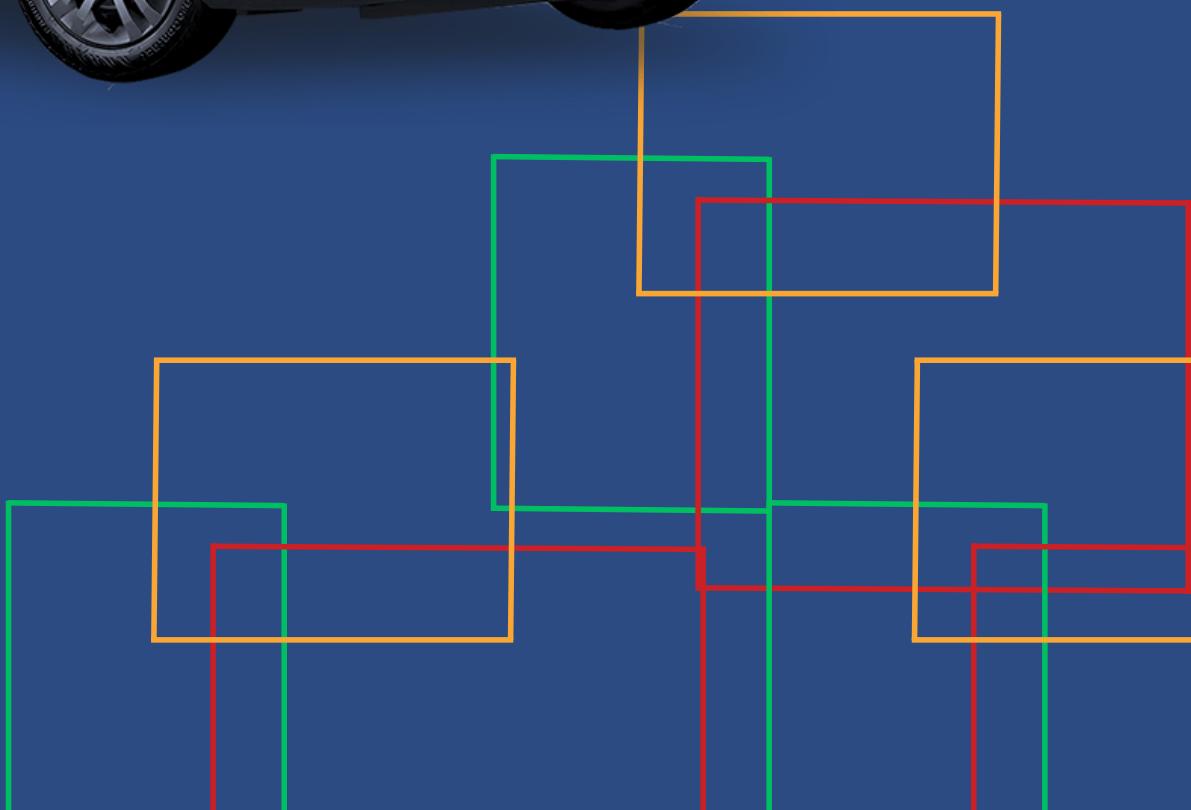
Para aprender bem, a rede precisa de:

- Muitos exemplos rotulados
- Várias iterações de treinamento
- Cálculo e correção de erros em cada ciclo

A rede vai melhorando a cada ciclo, aprendendo a generalizar padrões e errar menos.



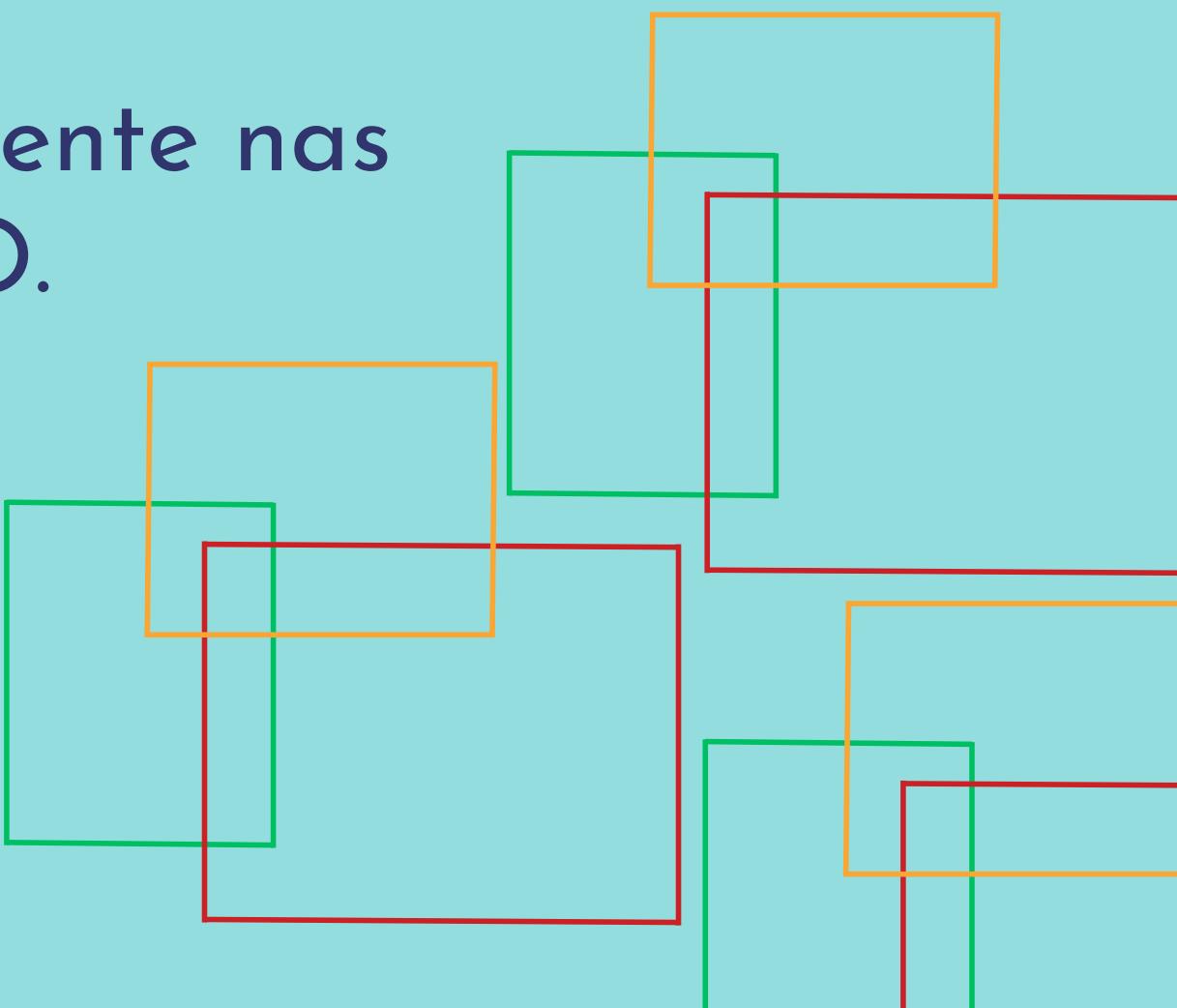
Exemplo:



Tipos de redes neurais

Existem diversos tipos de redes neurais, cada uma especializada para diferentes tarefas: classificação de dados, processamento de imagens, análise de sequências temporais, processamento de linguagem natural, entre outras aplicações.

Vamos conhecer os tipos mais utilizados, especialmente nas aplicações com visão computacional, como o YOLO.



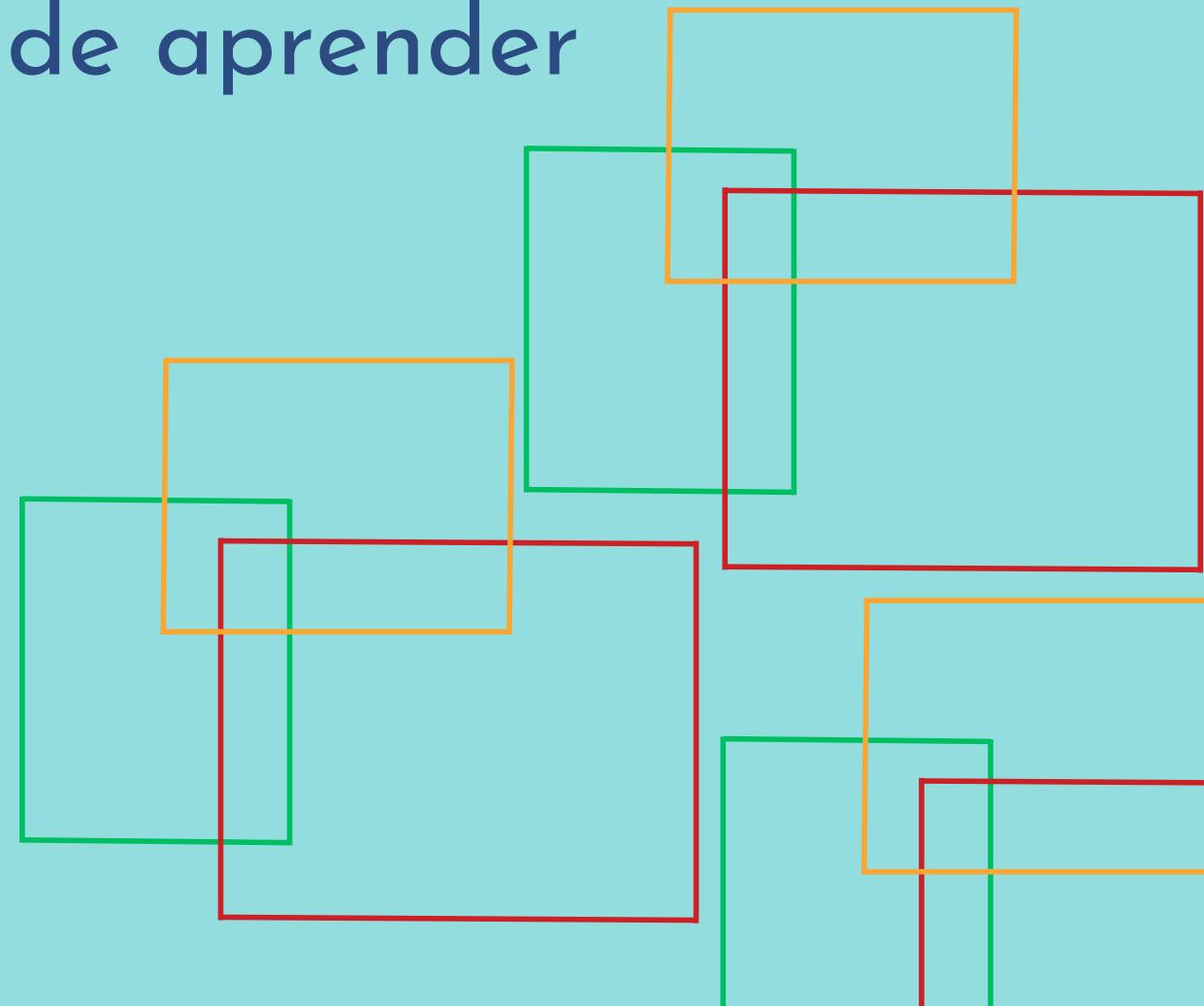
Perceptron e MLP

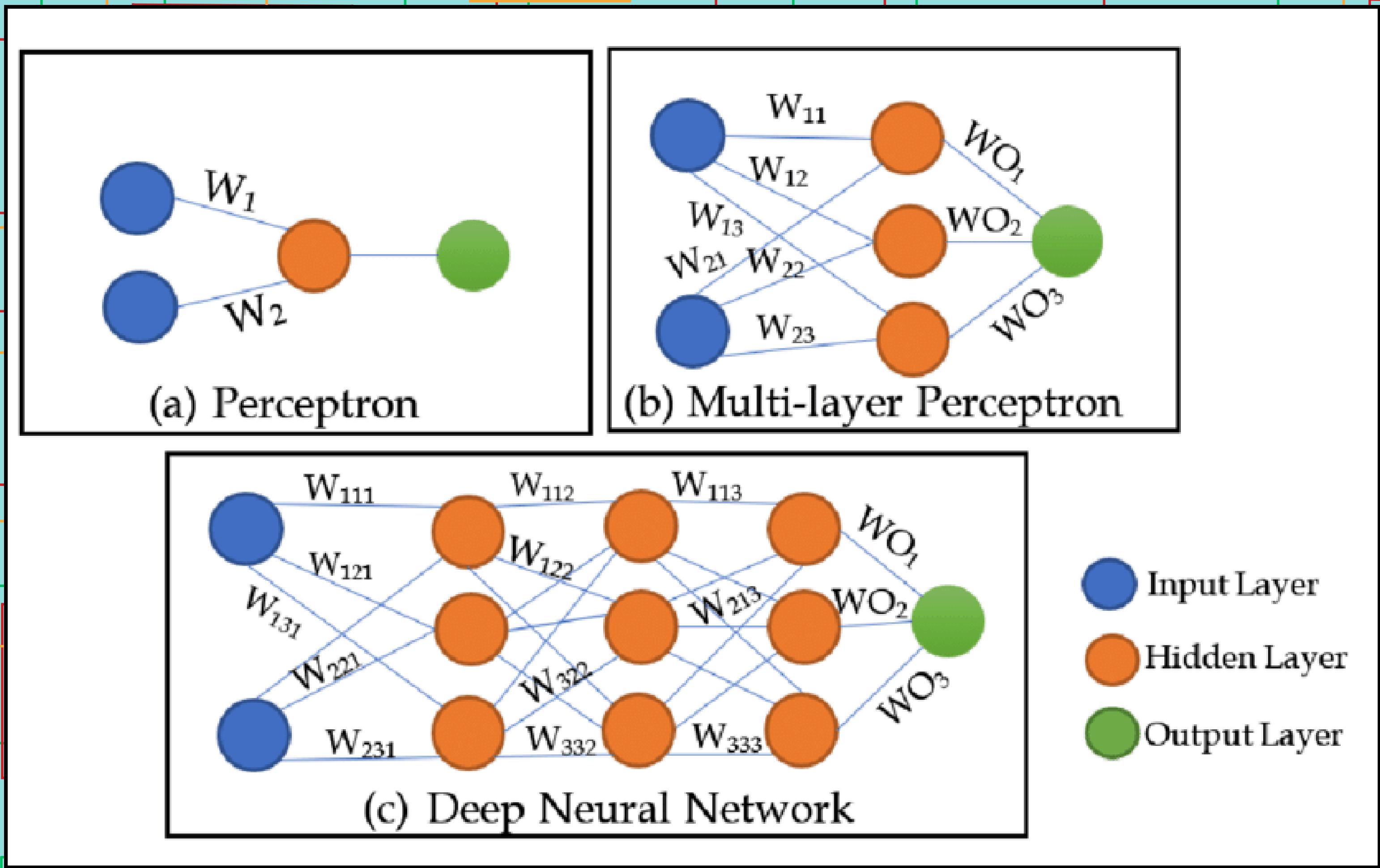
O Perceptron é o modelo mais simples, criado nos anos 1950, com apenas uma camada de neurônios. Resolve problemas simples como AND/OR.

A MLP (Multilayer Perceptron) expande esse conceito, com várias camadas ocultas e funções de ativação. É capaz de aprender padrões complexos em dados estruturados.

Usadas em:

- Classificação básica
- Predição de valores
- Reconhecimento de padrões



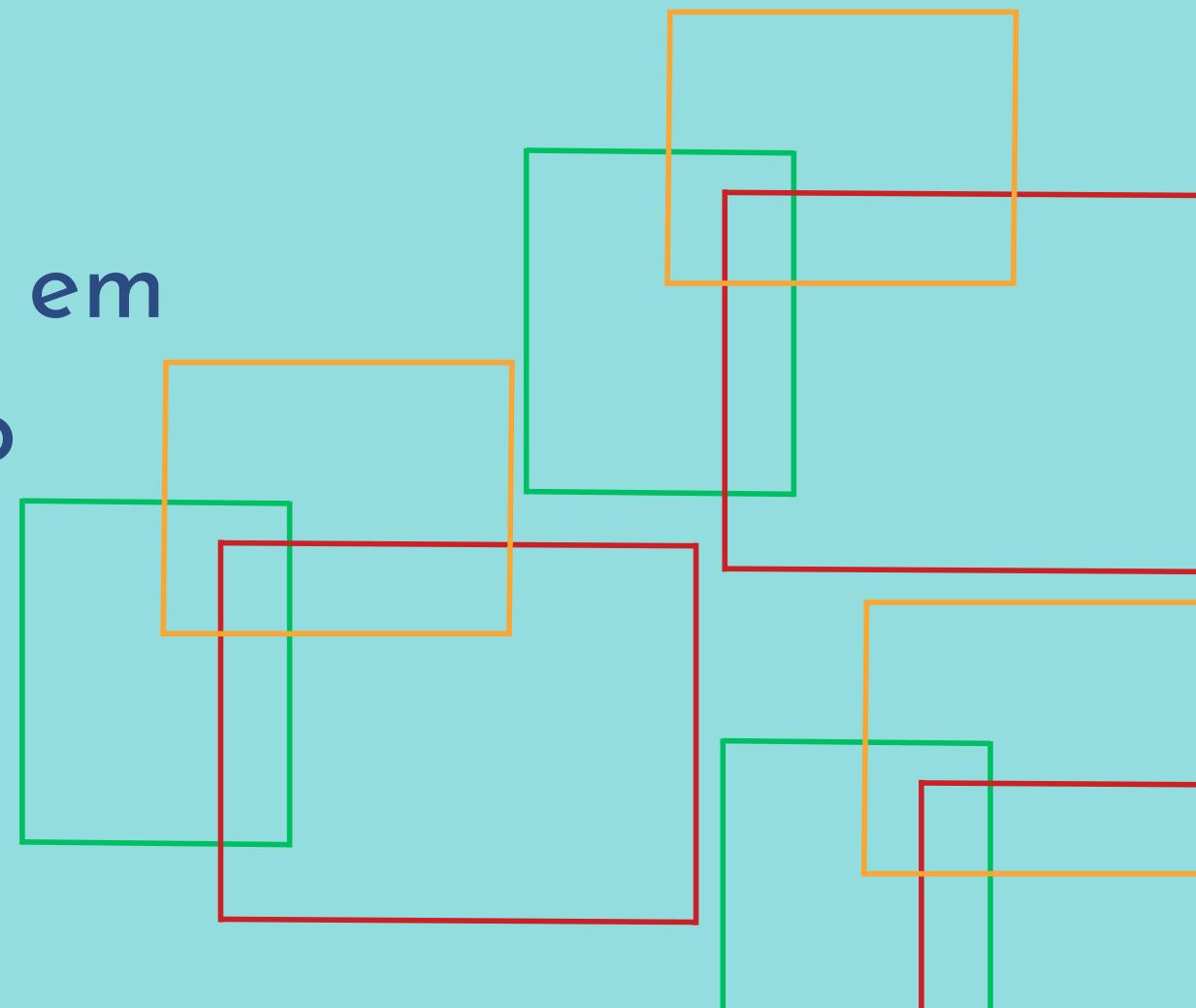


Redes Neurais Convolucionais (CNN)

As redes convolucionais são usadas para visão computacional, como detecção de objetos e reconhecimento de imagens.

Elas detectam padrões visuais em camadas: de bordas e formas simples até objetos completos.

São a base de algoritmos como o YOLO, usados em detecção de objetos, também em reconhecimento facial e em imagens médicas.

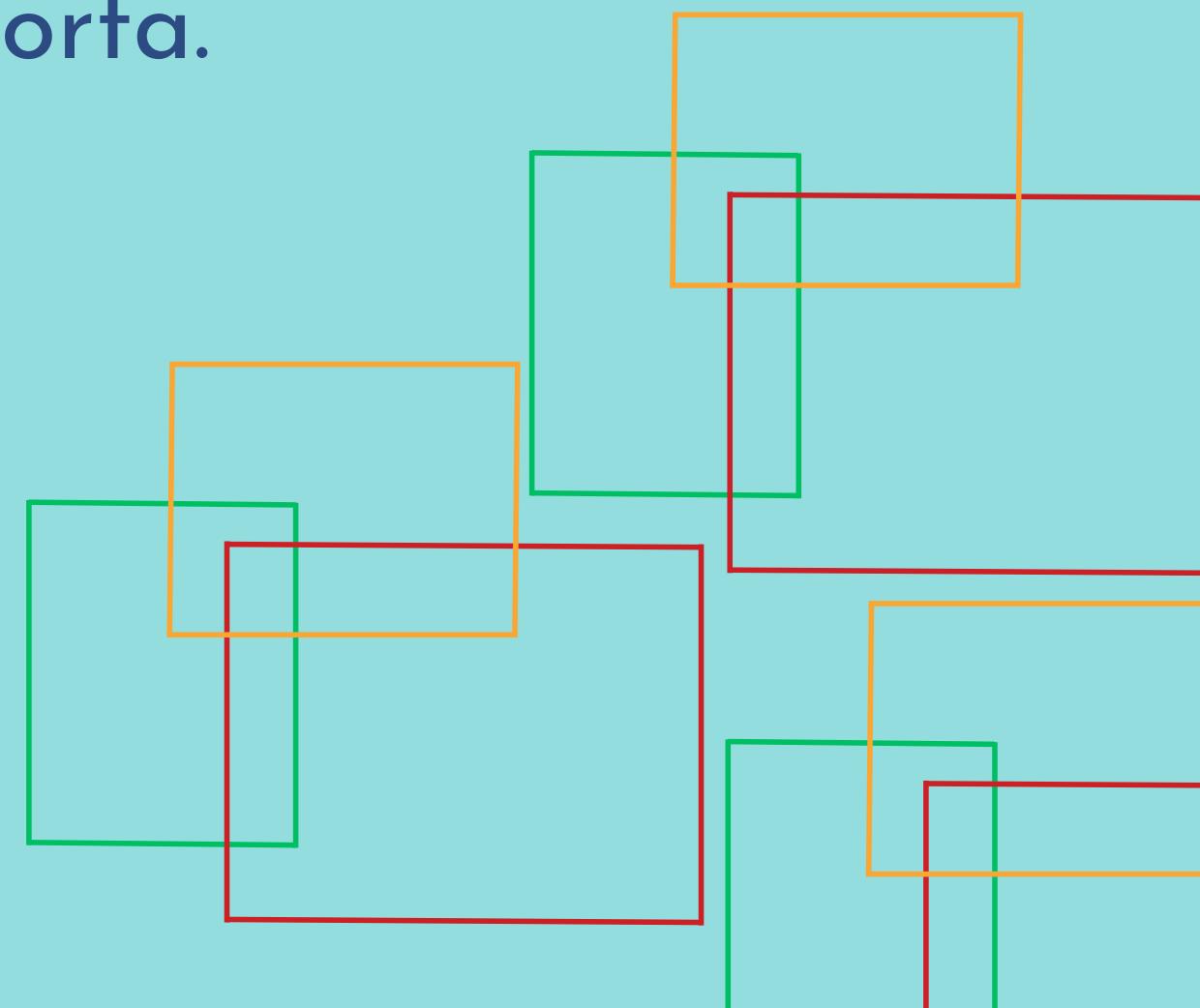


Redes Recorrentes (RNN)

As redes recorrentes é uma rede com laços (recorrência), onde a saída de um neurônio influencia ele mesmo no próximo instante.

Feita para dados sequenciais, onde a ordem importa.

LSTMs (Long Short-Term Memory) e GRUs (Gated Recurrent Unit) são variações mais eficientes, com memória de longo prazo, usadas em texto, fala e séries temporais

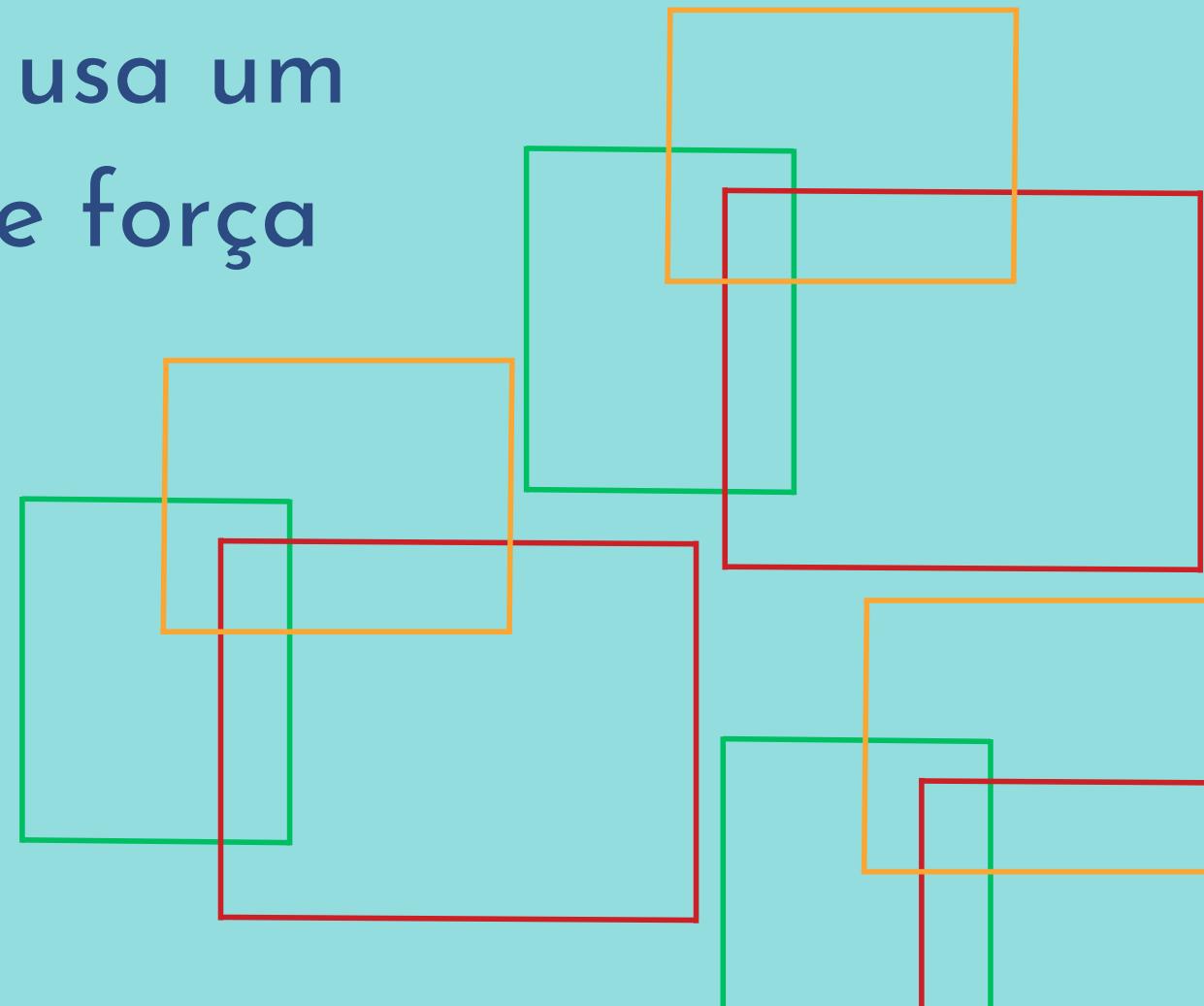


Transformers

Os Transformers é um tipo de rede neural moderna, criado para processar sequências como textos, códigos e imagens de forma paralela e altamente eficiente.

Ao contrário das RNNs, ele não depende da ordem sequencial para aprender padrões, em vez disso, usa um mecanismo chamado atenção, que permite à rede focar nas partes mais importantes da entrada.

É considerado o modelo dominante da IA atual, sendo base para o BERT, ChatGPT.



Outros tipos de redes neurais:

Autoencoders: aprendem a reconstruir a própria entrada, comprimindo e descomprimindo os dados. Úteis para redução de dimensionalidade, remoção de ruído e pré-dimensionamento de dados

GANs (Redes Adversárias Generativas): são duas redes que competem: uma gera dados falsos e a outra tenta identificar o que é real. Usadas para geração de imagens realistas, deepfakes, arte digital e melhoria de qualidade de imagens.

Híbridas e variantes: redes que combinam arquiteturas ou versões otimizadas de modelos maiores. Usadas para criar soluções personalizadas em aplicações complexas.



Tipo de rede X Forma de funcionamento

Tipo de Rede Neural:

É o modelo em si: um conjunto de camadas, conexões, funções de ativação e arquitetura específica.

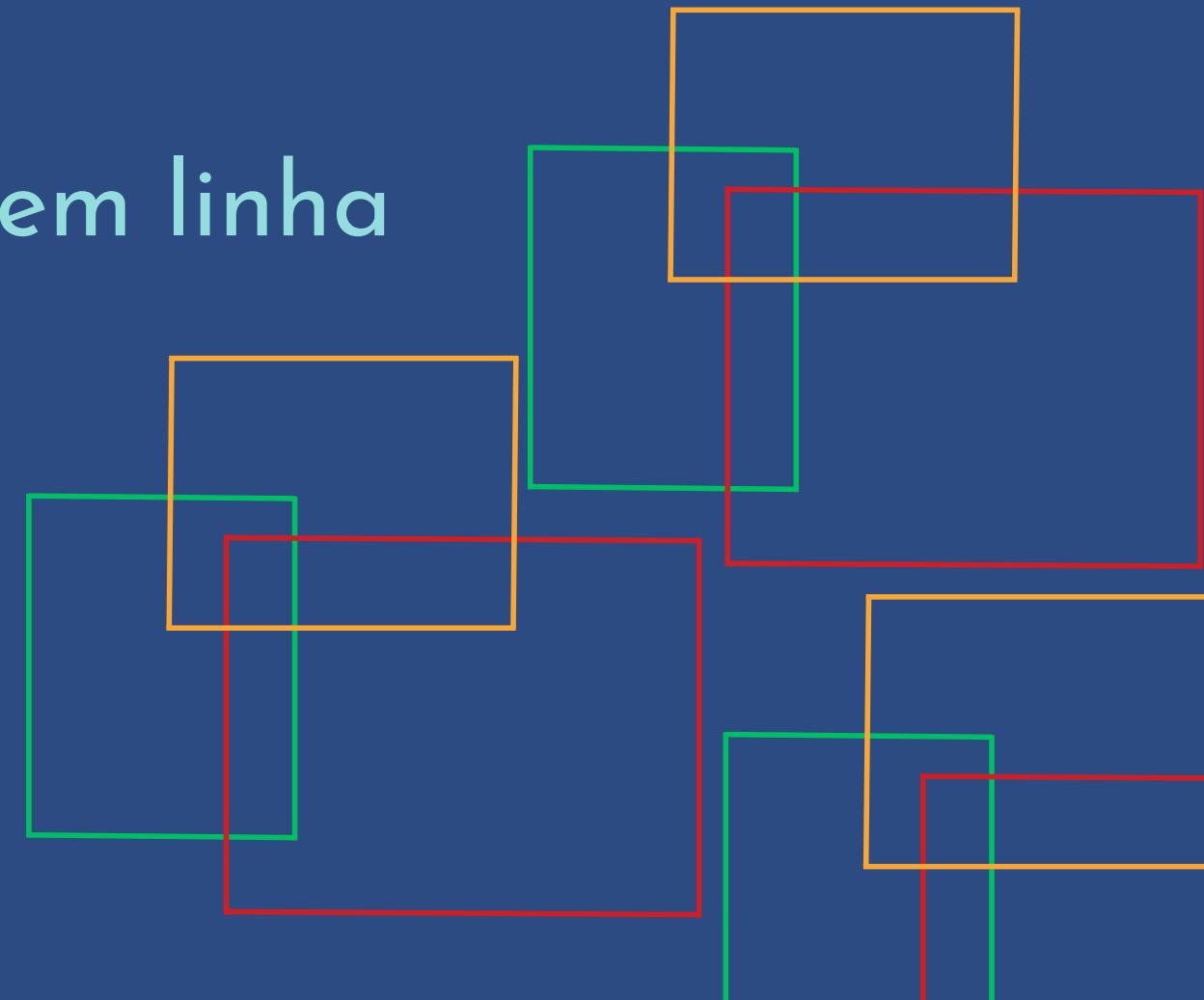
Ex: Perceptron, MLP, CNN, RNN, Transformer...

Forma de Funcionamento (Fluxo de Dados):

É como os dados se movimentam dentro da rede: em linha reta, com memória, com foco seletivo...

O tipo é o "modelo"

A forma de funcionamento é o "caminho que os dados seguem"

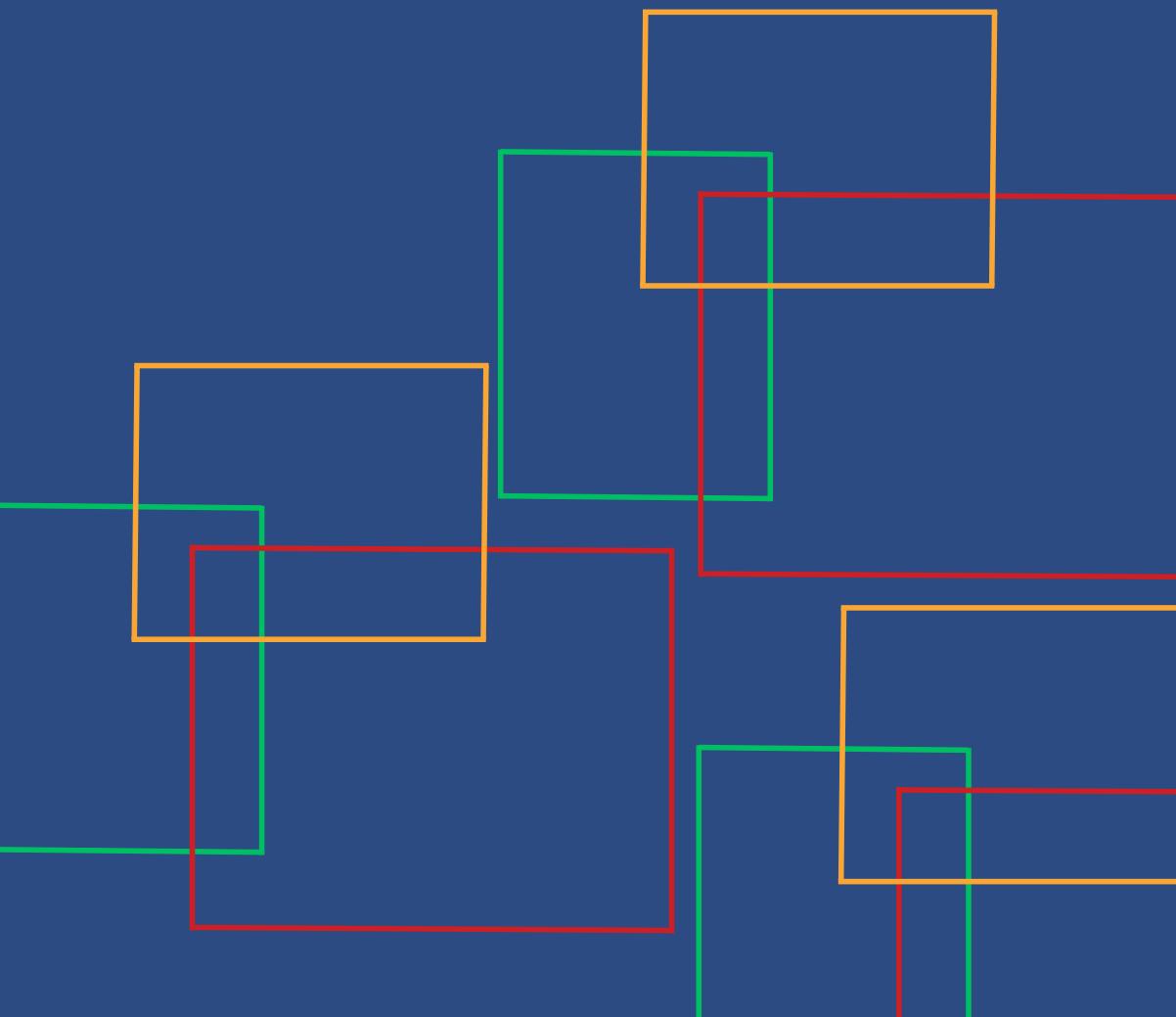


Forma de Funcionamento de Redes Neurais

Feedforward (Direto):

- Os dados fluem apenas para frente: entrada → camadas ocultas → saída.
- Não há memória ou ciclos.
- Simples e eficiente.

Usado em: Perceptron, MLP, CNN.



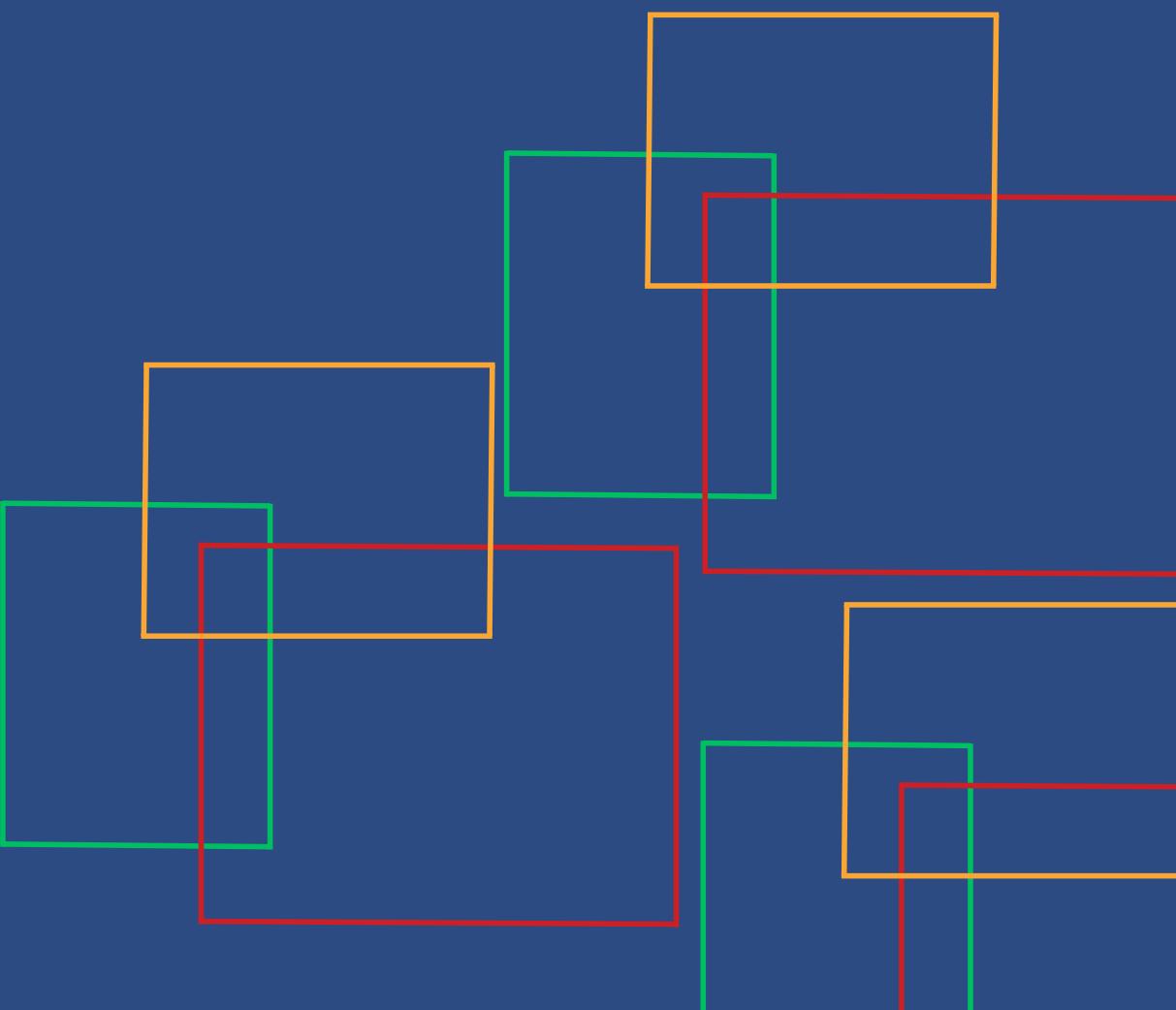
Forma de Funcionamento de Redes Neurais

Recorrente:

- Os dados fluem para frente, mas também retornam para trás.
- Permite memória de curto ou longo prazo.

Usado em: RNN, LSTM, GRU.

Ideal para texto, fala e séries temporais.

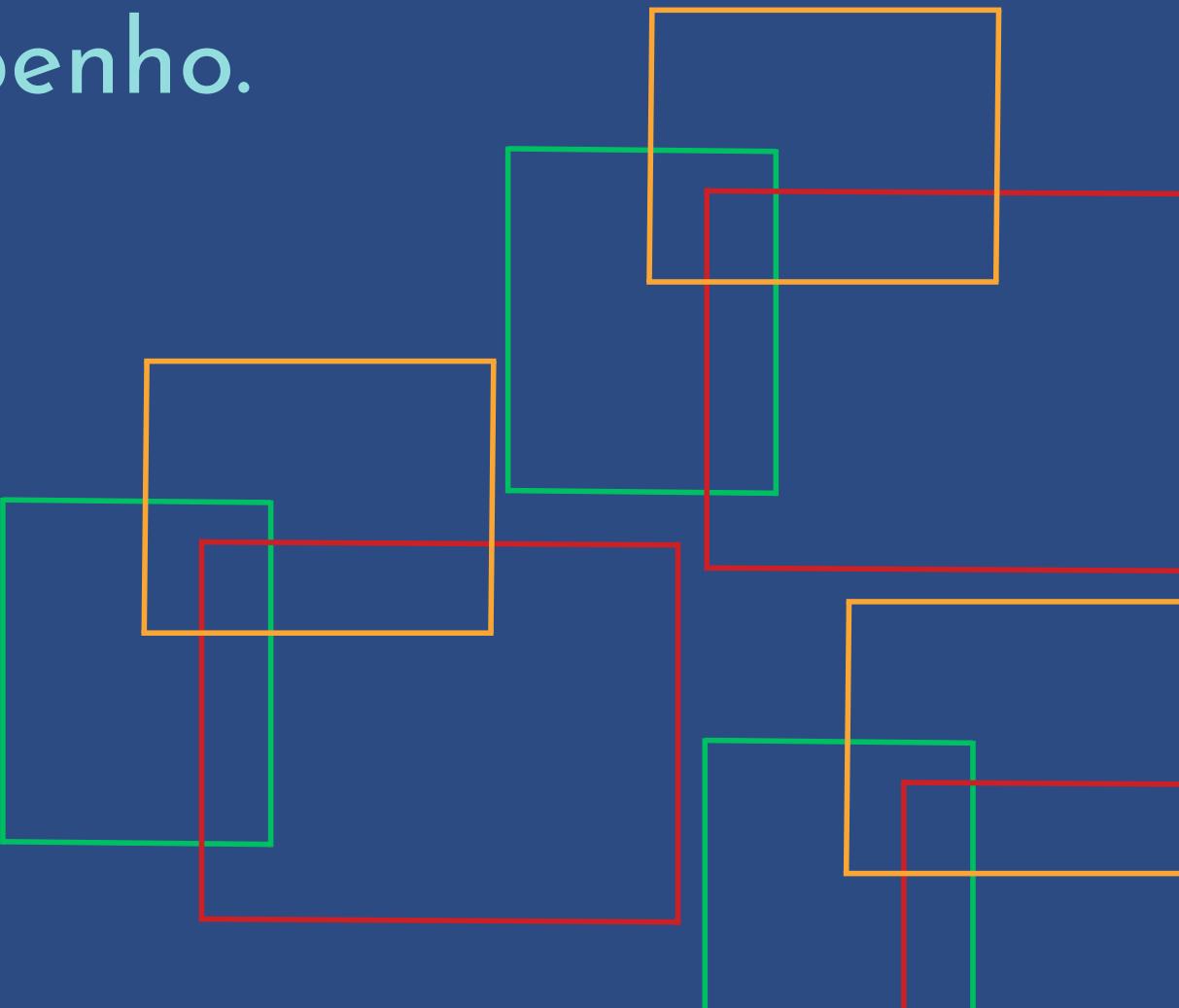


Forma de Funcionamento de Redes Neurais

Com Atenção (Attention-based):

- Os dados ainda fluem para frente, mas agora com atenção a partes específicas da entrada.
- Permite treinar em paralelo, com alto desempenho.

Usado em: Transformer, BERT, GPT.

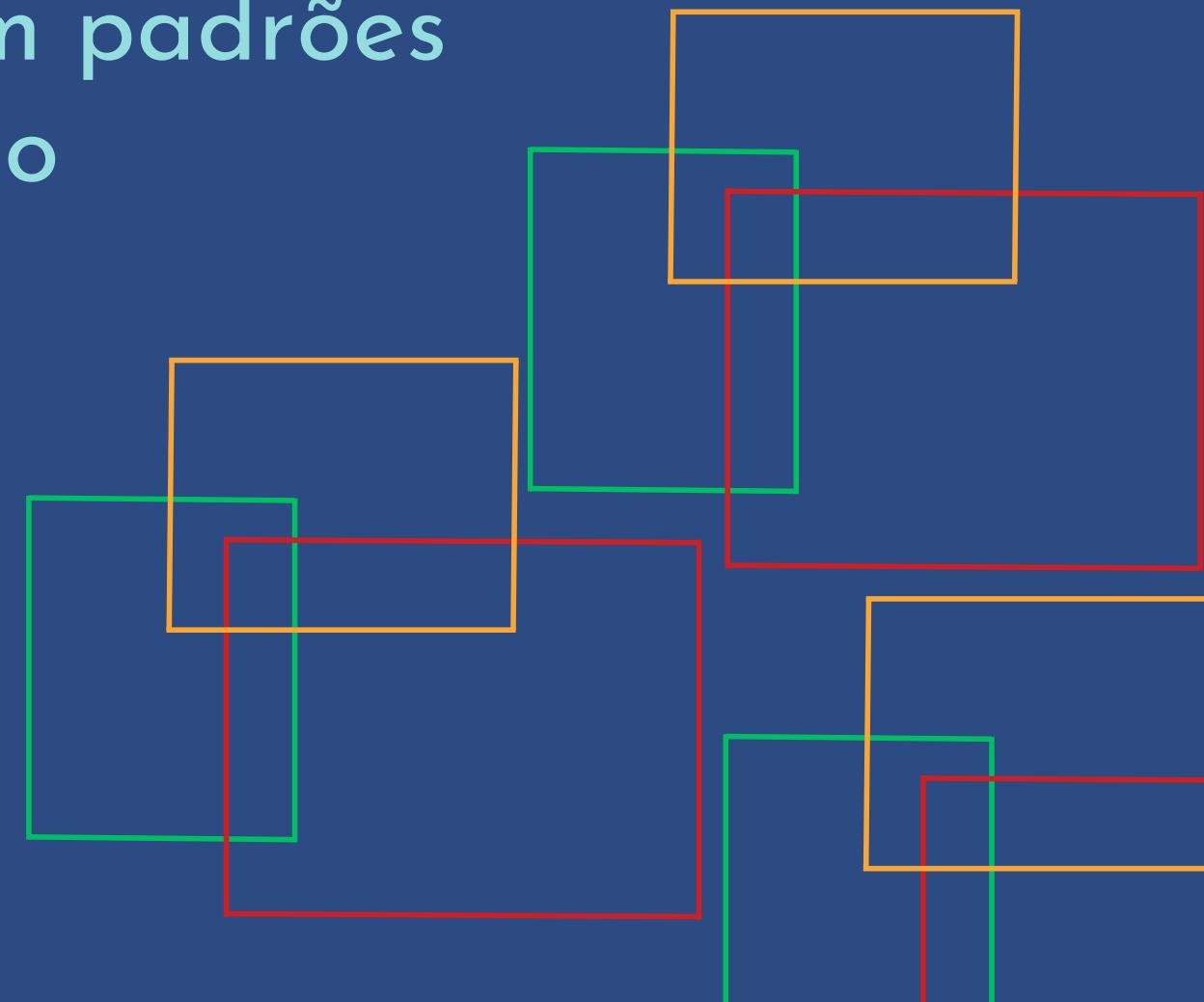


CNN - Redes Neurais Convolucionais

As CNNs (Convolutional Neural Networks) são tipo especial de rede neural projetada para lidar com dados estruturados em grade, como imagens.

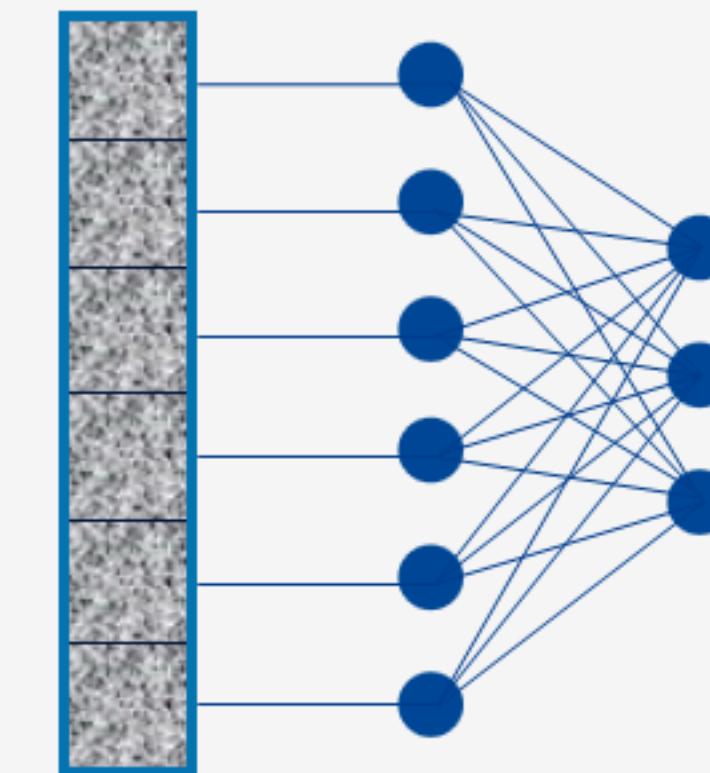
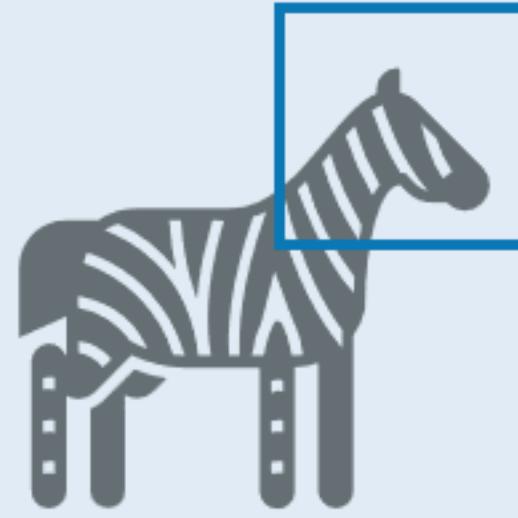
Diferente das redes densas (como as MLPs), que tratam todos os pixels de forma igual, as CNNs aprendem padrões visuais como bordas, texturas e formas, observando pequenas regiões da imagem por vez.

Isso permite que a rede construa uma visão hierárquica da imagem, indo de padrões simples (como bordas) até formas complexas (como objetos inteiros).

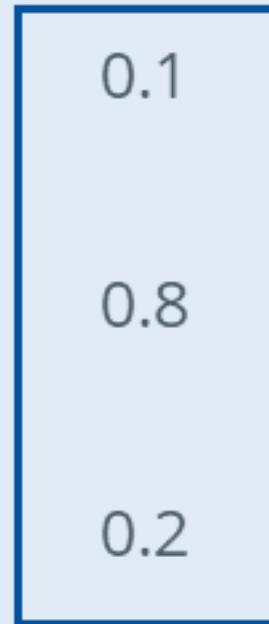


CONVOLUTIONAL NEURAL NETWORK (CNN)

INPUT



OUTPUT



← Feature Maps →

← Fully Connected Layer →

← Extração de recursos →

← Classificação →

← Avaliação probabilística →

IONOS

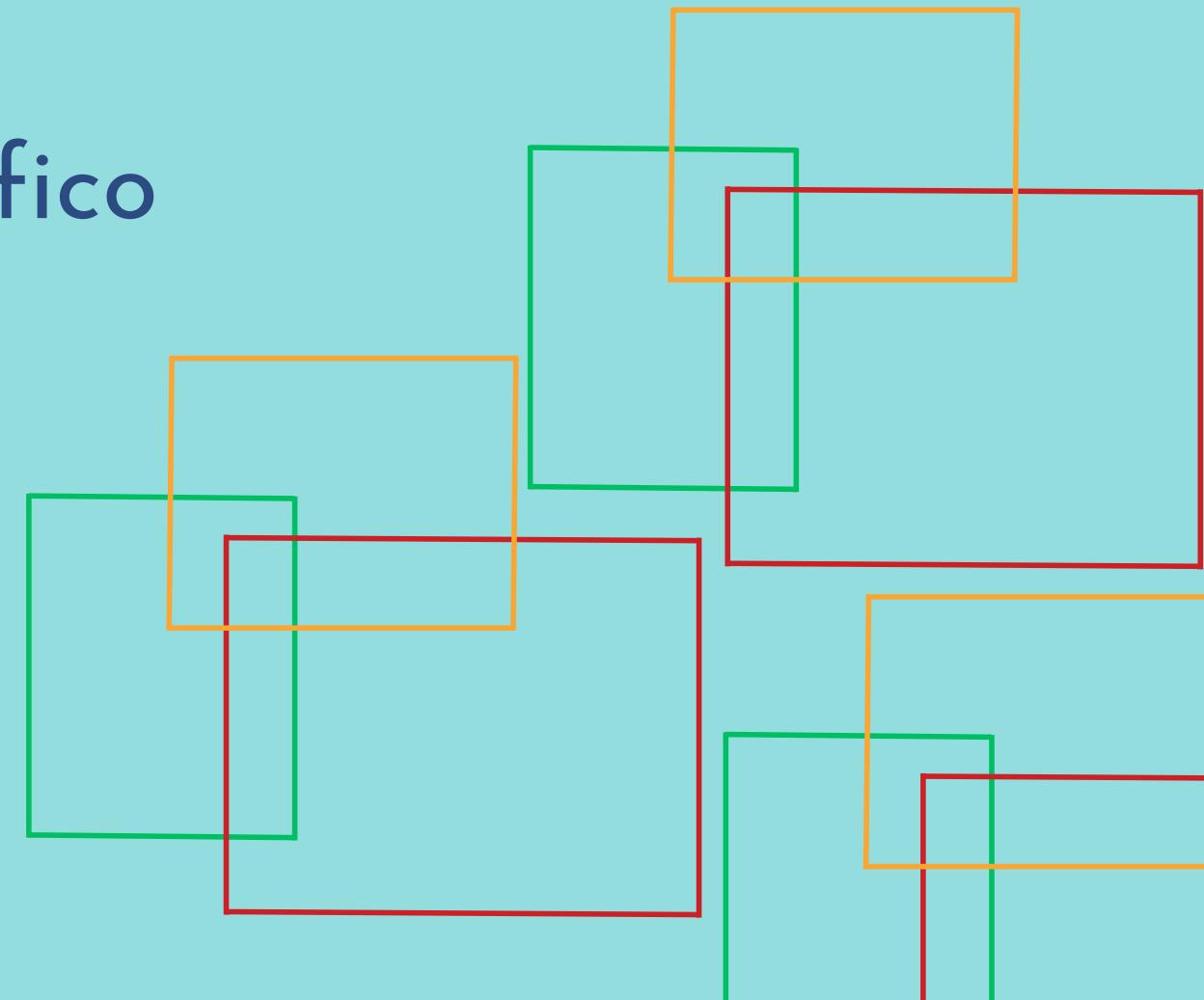
Principais componentes das CNNs:

Camadas Convolucionais:

As camadas convolucionais são o núcleo das CNNs.

Elas aplicam filtros (ou kernels) que percorrem a imagem em pequenas regiões, extraindo padrões locais como bordas, curvas ou texturas.

Cada filtro aprende a reconhecer um tipo específico de característica visual, e quanto mais camadas, mais complexos são os padrões que a rede consegue identificar.



Principais componentes das CNNs:

Kernels (ou filtros):

Um kernel é uma pequena matriz, geralmente 3×3 ou 5×5 , usada para analisar pequenas regiões da imagem.

Ele se move sobre a imagem original, multiplicando seus valores pelos valores dos pixels da região que cobre.

O resultado dessas multiplicações é somado, gerando um novo valor em um mapa de características.

Cada filtro aprende a detectar um tipo de padrão visual: bordas, cantos, texturas, etc.



Source layer

5	2	6	8	2	0	1	2
4	3	4	5	1	9	6	3
3	9	2	4	7	7	6	9
1	3	4	6	8	2	2	1
8	4	6	2	3	1	8	8
5	8	9	0	1	0	2	3
9	2	6	6	3	6	2	1
9	8	8	2	6	3	4	5

Convolutional
kernel

-1	0	1
2	1	2
1	-2	0

Destination layer

$$(-1 \times 5) + (0 \times 2) + (1 \times 6) + \\ (2 \times 4) + (1 \times 3) + (2 \times 4) + \\ (1 \times 3) + (-2 \times 9) + (0 \times 2) = 5$$

Principais componentes das CNNs:

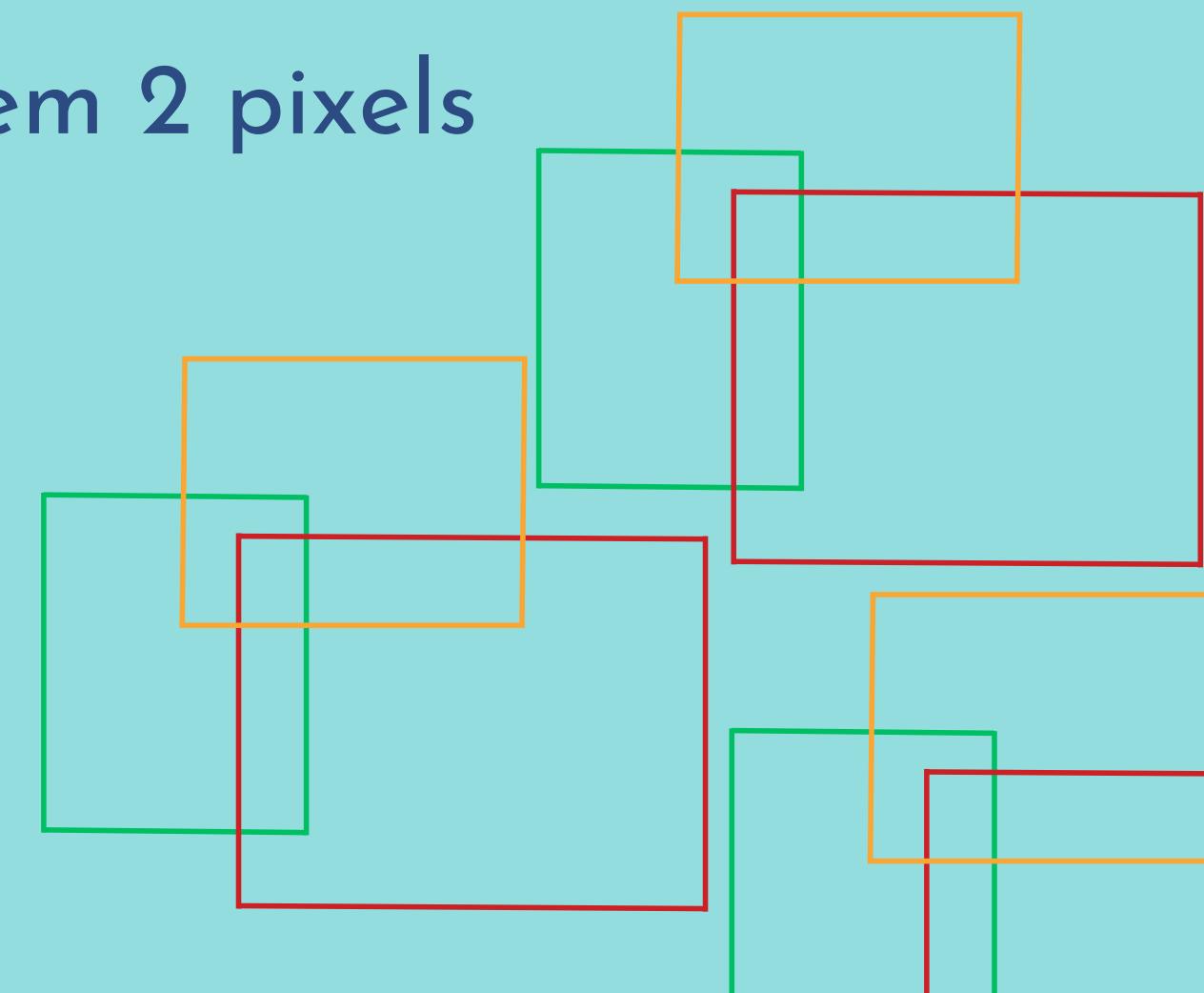
Stride:

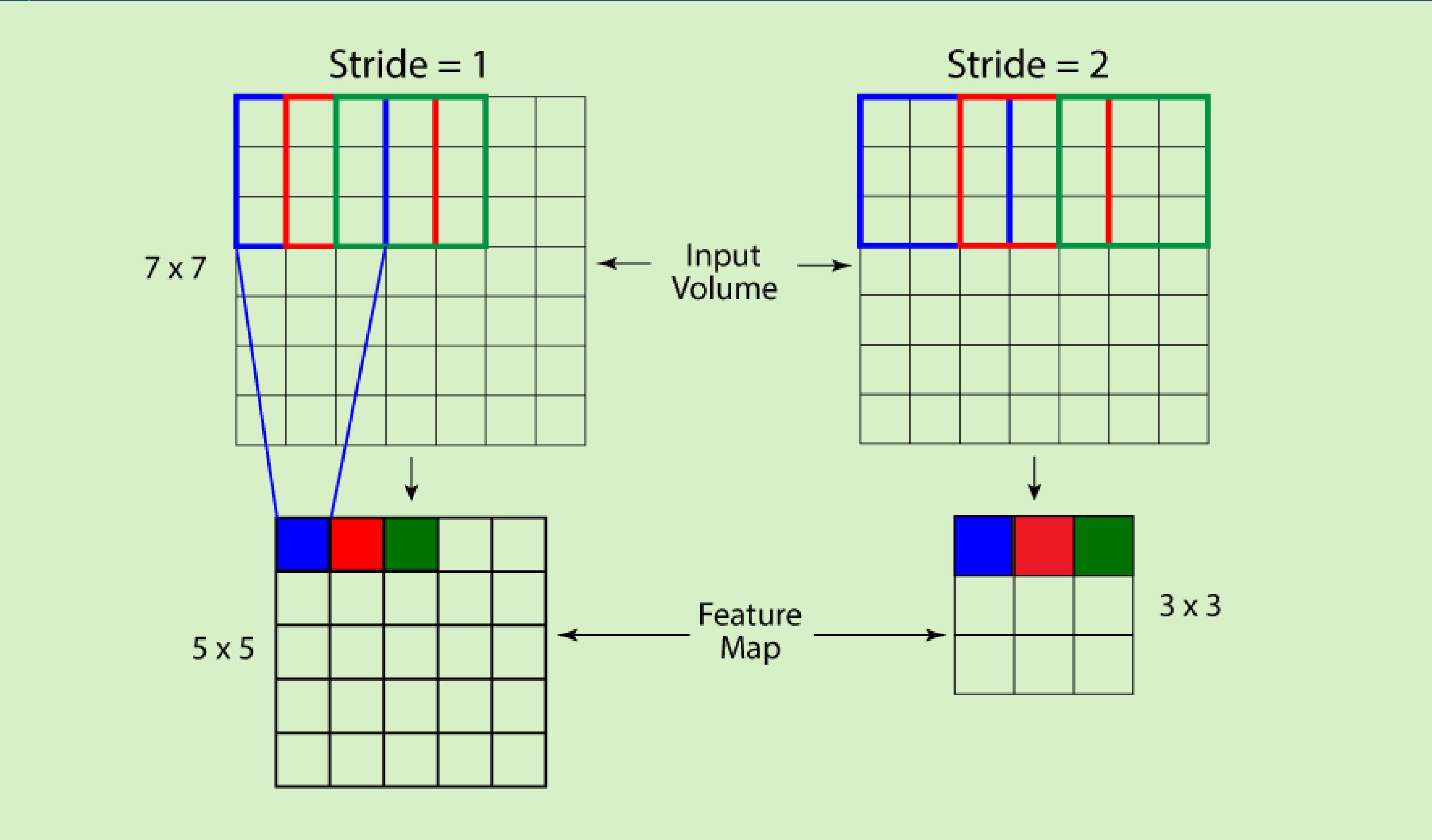
O stride define quantos pixels o kernel avança a cada movimento.

Um stride = 1 significa que o kernel anda de 1 em 1 pixel (mais detalhado, saída maior).

Um stride = 2 significa que o kernel anda de 2 em 2 pixels (menos detalhado, saída menor).

Isso impacta diretamente o tamanho da saída e o tempo de processamento.





Principais componentes das CNNs:

Padding:

O padding adiciona uma moldura de zeros nas bordas da imagem para que o kernel possa processar todos os pixels, incluindo os das extremidades.

Isso mantém o tamanho da imagem de saída mais próximo do original e evita perda de informação nas extremidades.

- **Sem padding:** a imagem vai encolhendo a cada camada.
- **Com padding:** mantém o tamanho da entrada e da saída.



PADDING

Input

0	0	0	0	0	0	0
0						0
0						0
0						0
0						0
0						0
0	0	Padding			0	0

Kernel

x

Output

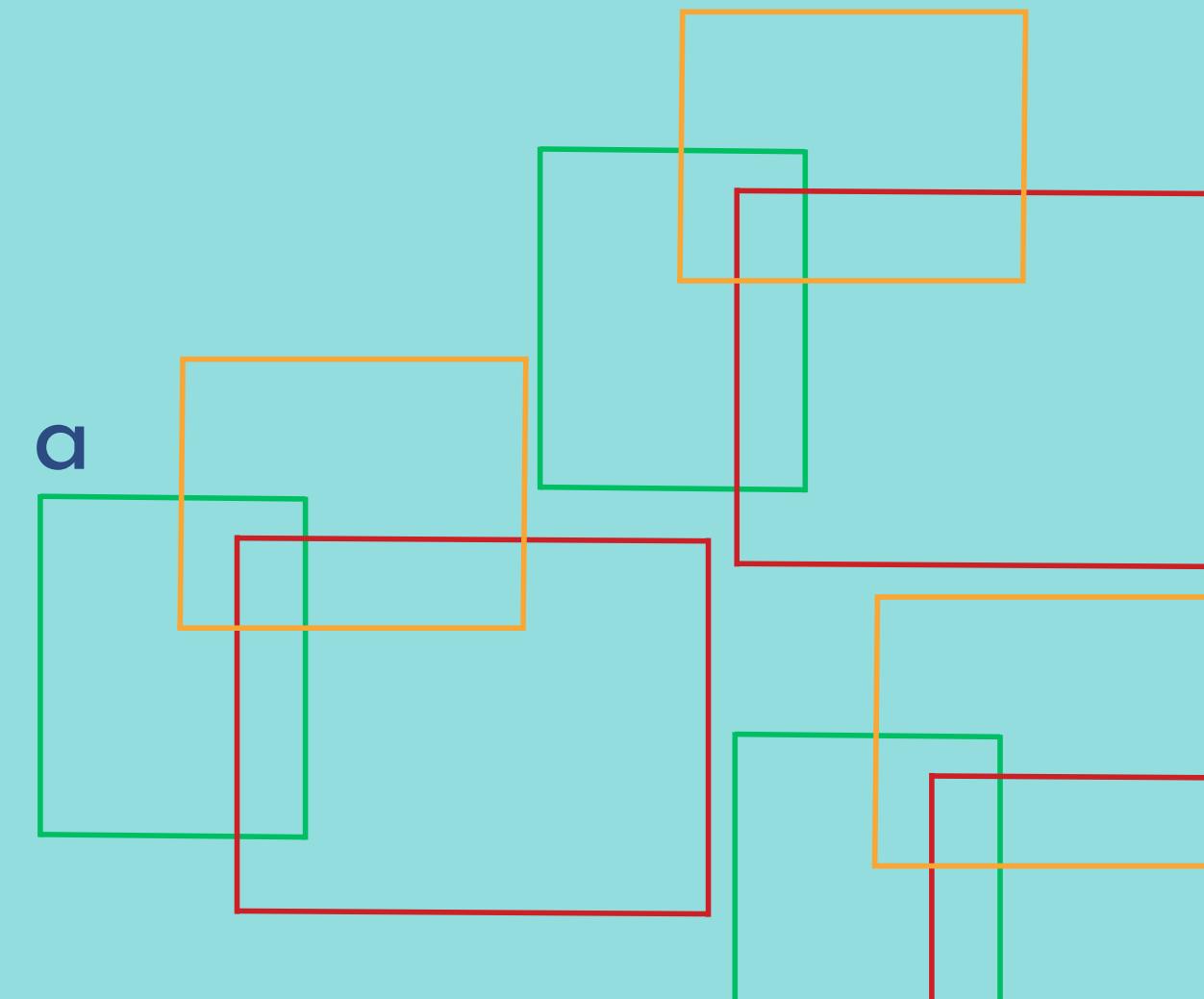
Feature Map

Principais componentes das CNNs:

Camadas de ativação (ReLU):

Após a convolução, entra a camada de ativação, que introduz não linearidade ao modelo. A mais usada é a ReLU (Rectified Linear Unit), que transforma todos os valores negativos em zero, mantendo os positivos.

Isso permite que a rede aprenda relações mais complexas nos dados. Sem essa não linearidade, a rede só conseguiria aprender funções simples e lineares.



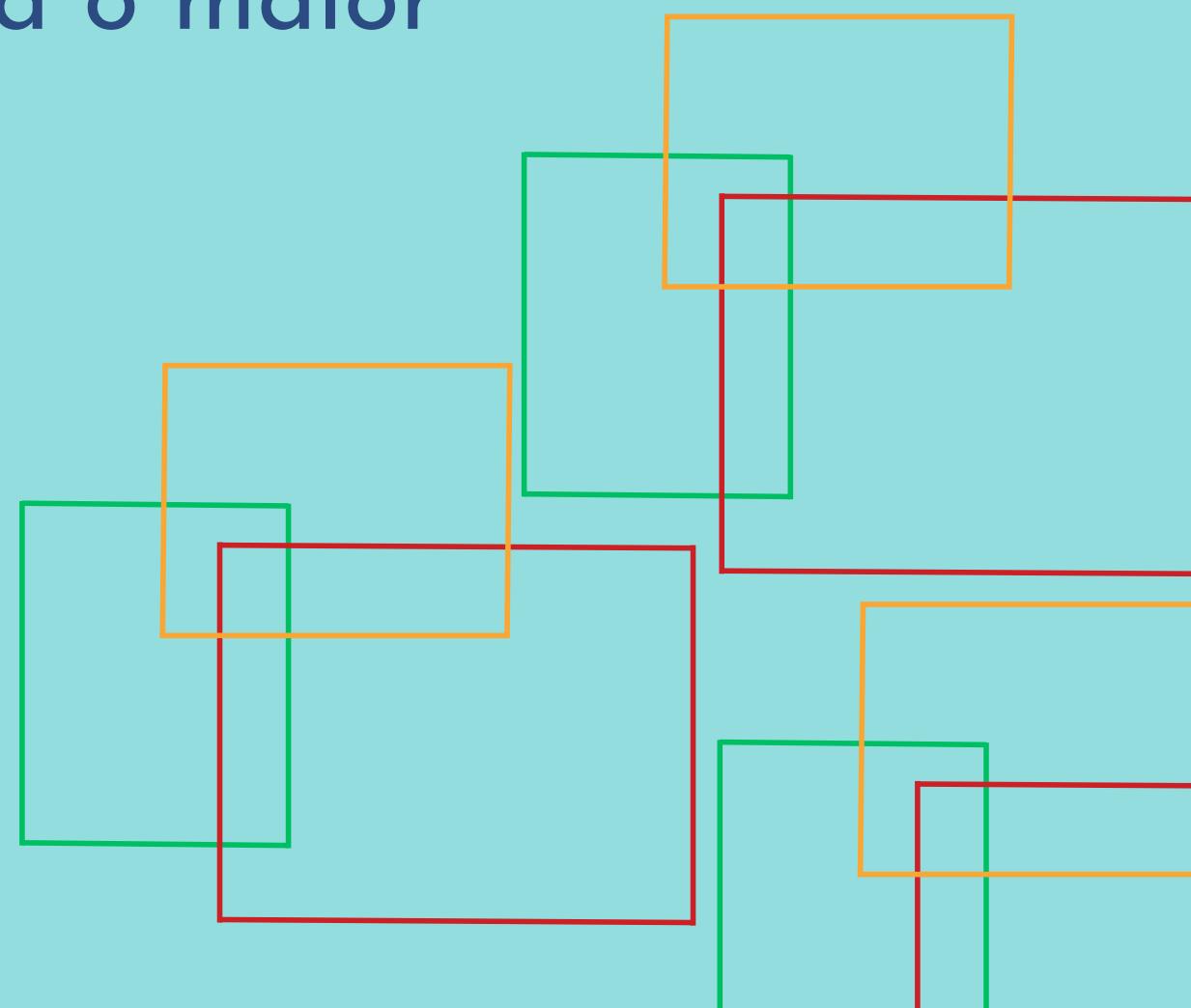
Principais componentes das CNNs:

Camadas de pooling:

As camadas de pooling servem para reduzir a dimensão dos dados, preservando apenas as informações mais relevantes.

A técnica mais comum é o max pooling, que pega o maior valor em cada região.

Isso ajuda a diminuir o número de parâmetros, evitar overfitting e acelerar o treinamento.



POOLING

Max pooling

32	19
20	27

Average pooling

15	14
11	16

32	10	11	17
4	14	9	19
20	4	16	27

Principais componentes das CNNs:

Flatten:

O Flatten é uma operação de transição que ocorre entre a camada de pooling e a camada totalmente conectada, onde ele converte mapas de características 2D em um vetor 1D.

Camadas convolucionais e de pooling trabalham com dados 2D (largura x altura x canais), mas a camada totalmente conectada precisa de dados 1D (vetor linear).



Pooled Feature Map

1	1	0
4	2	1
0	2	1

Flattening

1
1
0
4
2
1
0
2
1

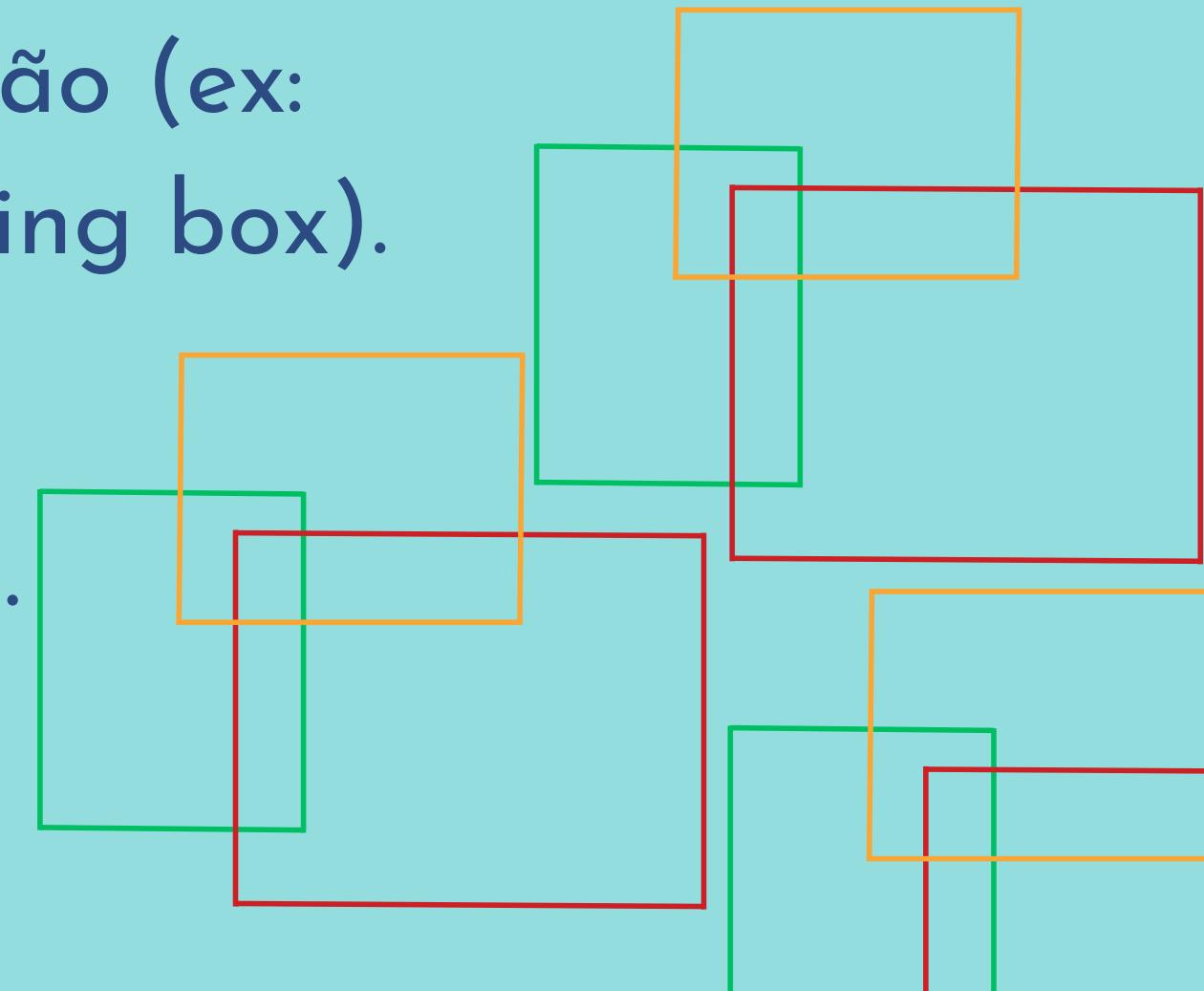
Principais componentes das CNNs:

Camadas totalmente conectadas:

Nas etapas finais, a CNN usa camadas densas (fully connected), iguais às da MLP.

Cada neurônio se conecta a todos os da camada anterior e gera a saída final da rede, como uma classificação (ex: “gato” ou “cachorro”) ou uma localização (bounding box).

Essas camadas combinam tudo o que foi aprendido antes e transformam em uma decisão.

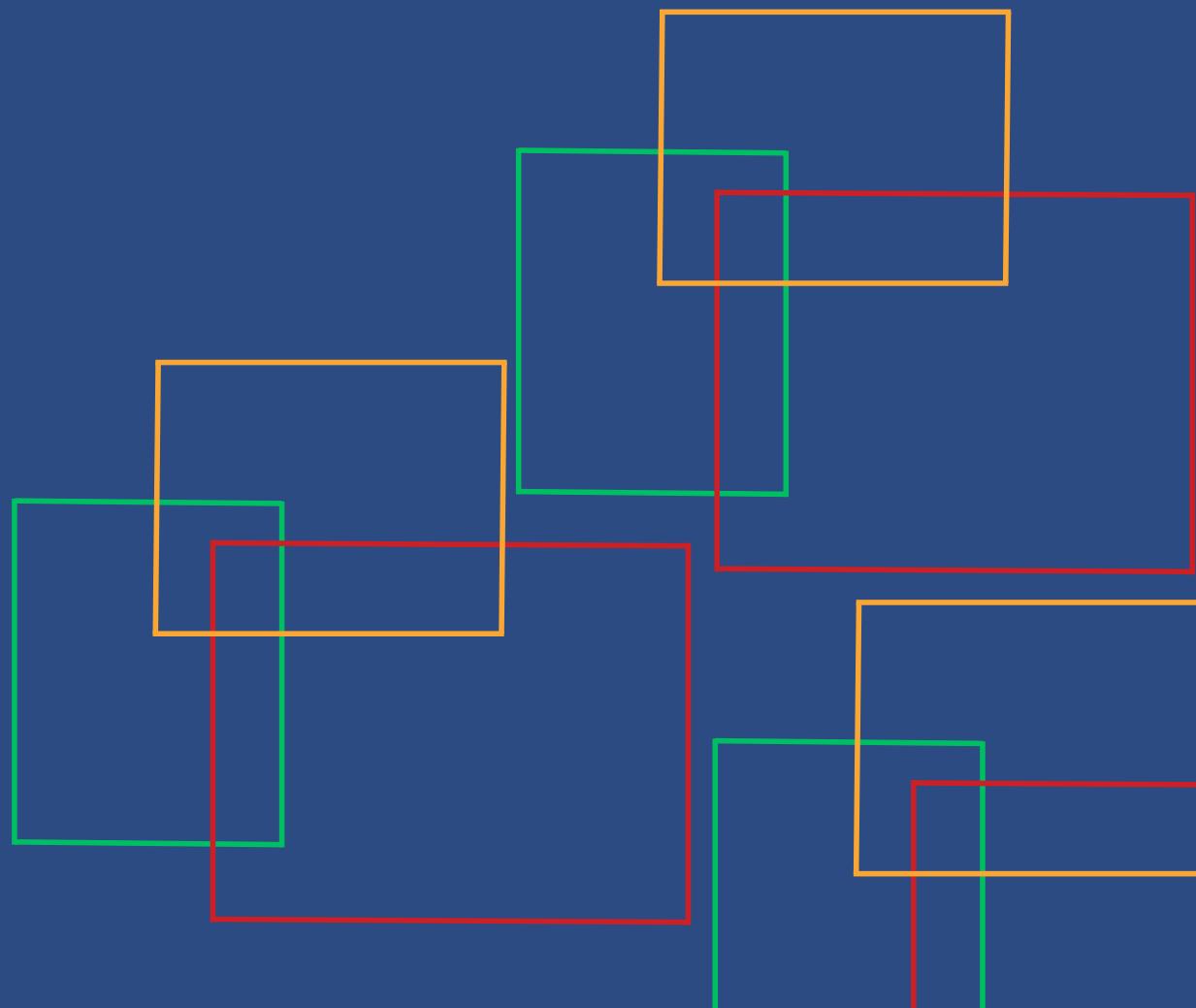


Por que CNNs funciona tão bem com imagens?

- Localidade: analisam regiões pequenas de cada vez.
- Compartilhamento de pesos: menos parâmetros, mais eficiência.
- Hierarquia de padrões:

Início: aprende bordas e cores.

Fim: reconhece objetos complexos.

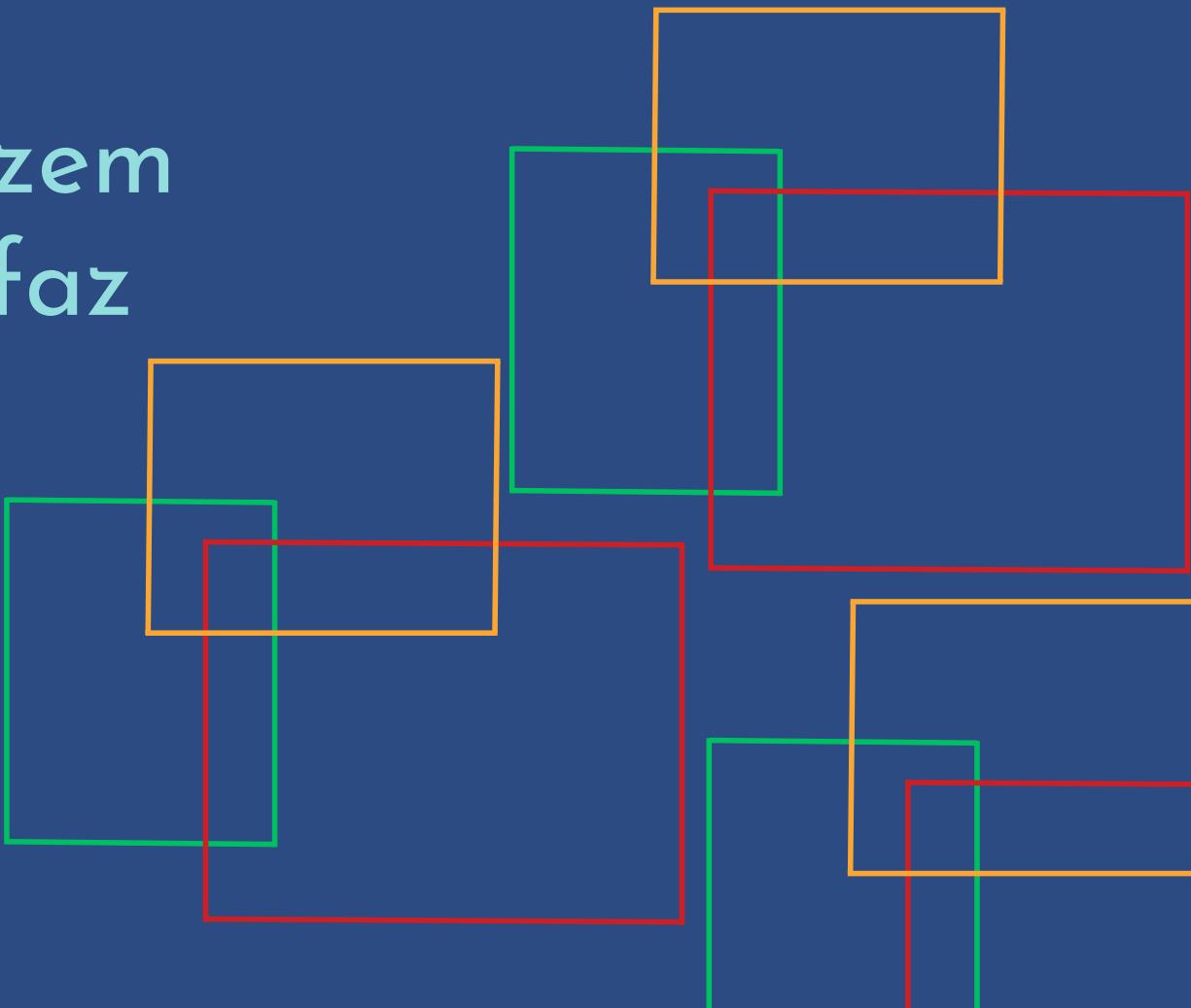


CNNs com YOLO:

YOLO (You Only Look Once) é uma arquitetura baseada em CNNs, projetada para detecção de objetos em tempo real.

Ela usa camadas convolucionais para extrair padrões (bordas, formas, objetos).

Diferença chave: Enquanto CNNs tradicionais fazem classificação ("o que é esta imagem?"), o YOLO faz detecção ("o que e onde estão os objetos?").

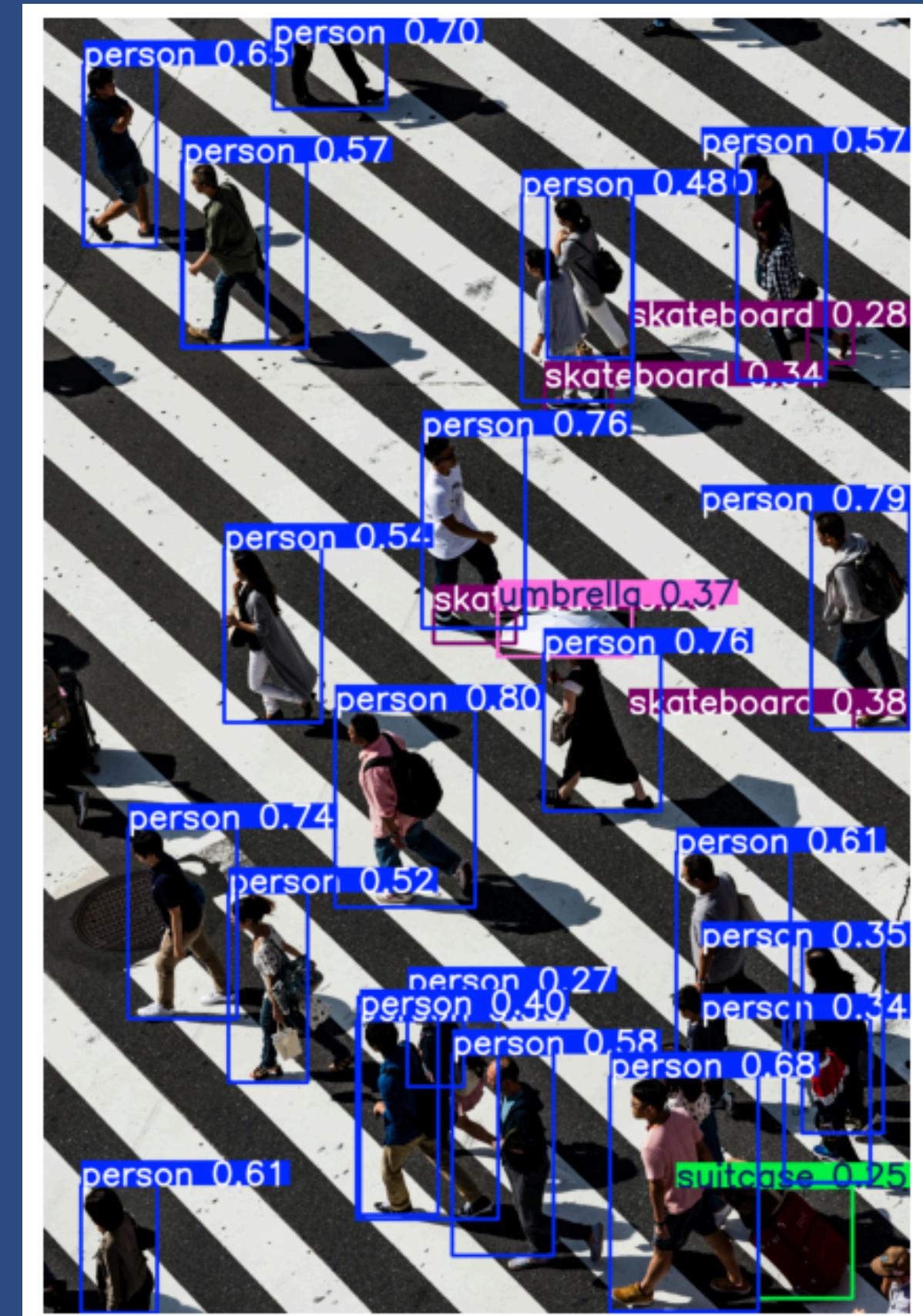


O que é YOLO?

Lançado em 2015, o algoritmo YOLO (You Only Look Once) representou uma revolução na detecção de objetos.

Sua abordagem inovadora alcançou alta precisão, mas com uma velocidade muito superior aos métodos da época, processando imagens em menos de 0.05 segundos.

Essa eficiência permitiu sua aplicação em tempo real (acima de 30 FPS) e, somado ao fato de ser totalmente open source, o consolidou como uma tecnologia de grande sucesso.



Classificação de Imagens e Detecção de Objetos

80

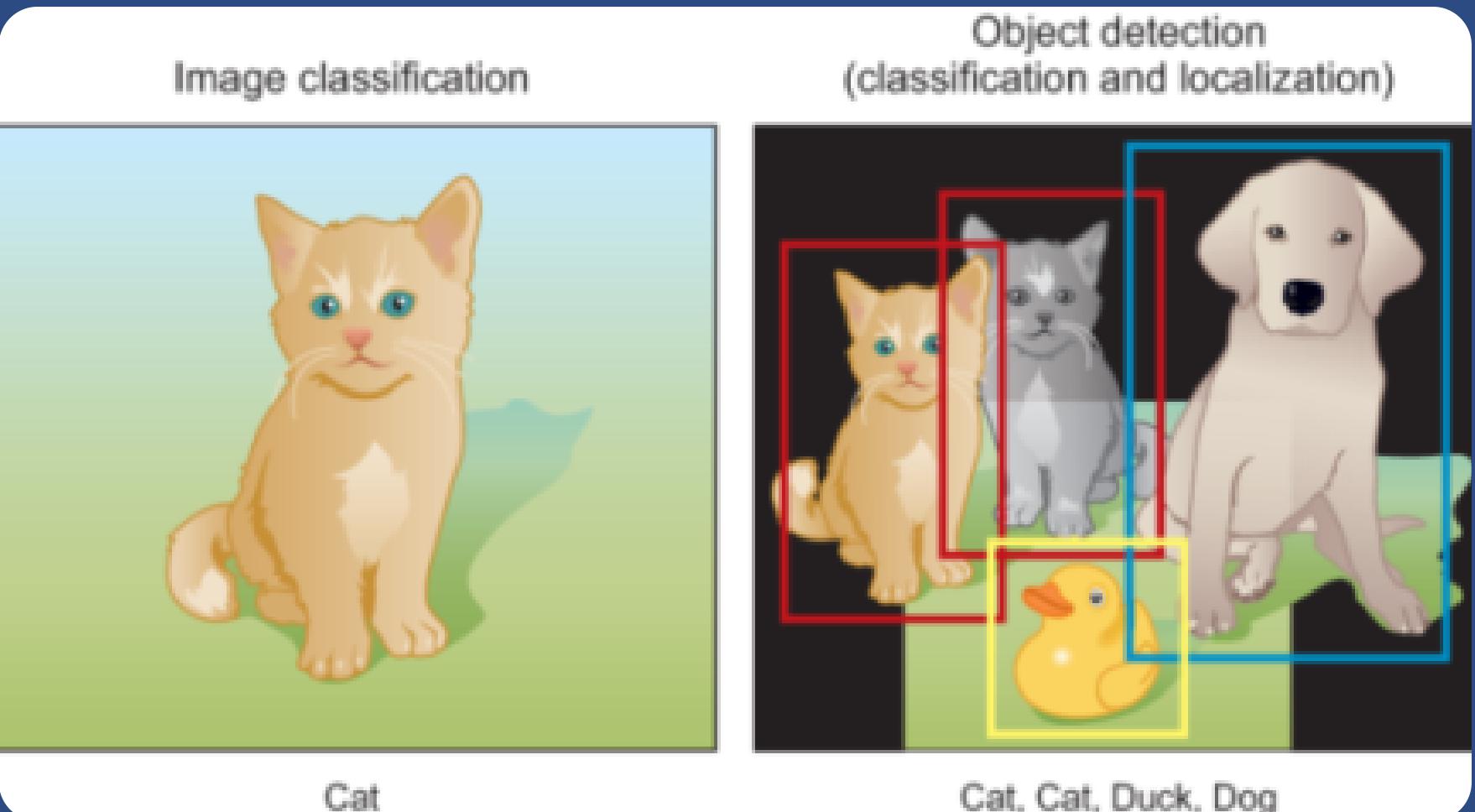
Classes base do **COCO Dataset**
(pessoas, carros, animais, objetos cotidianos)

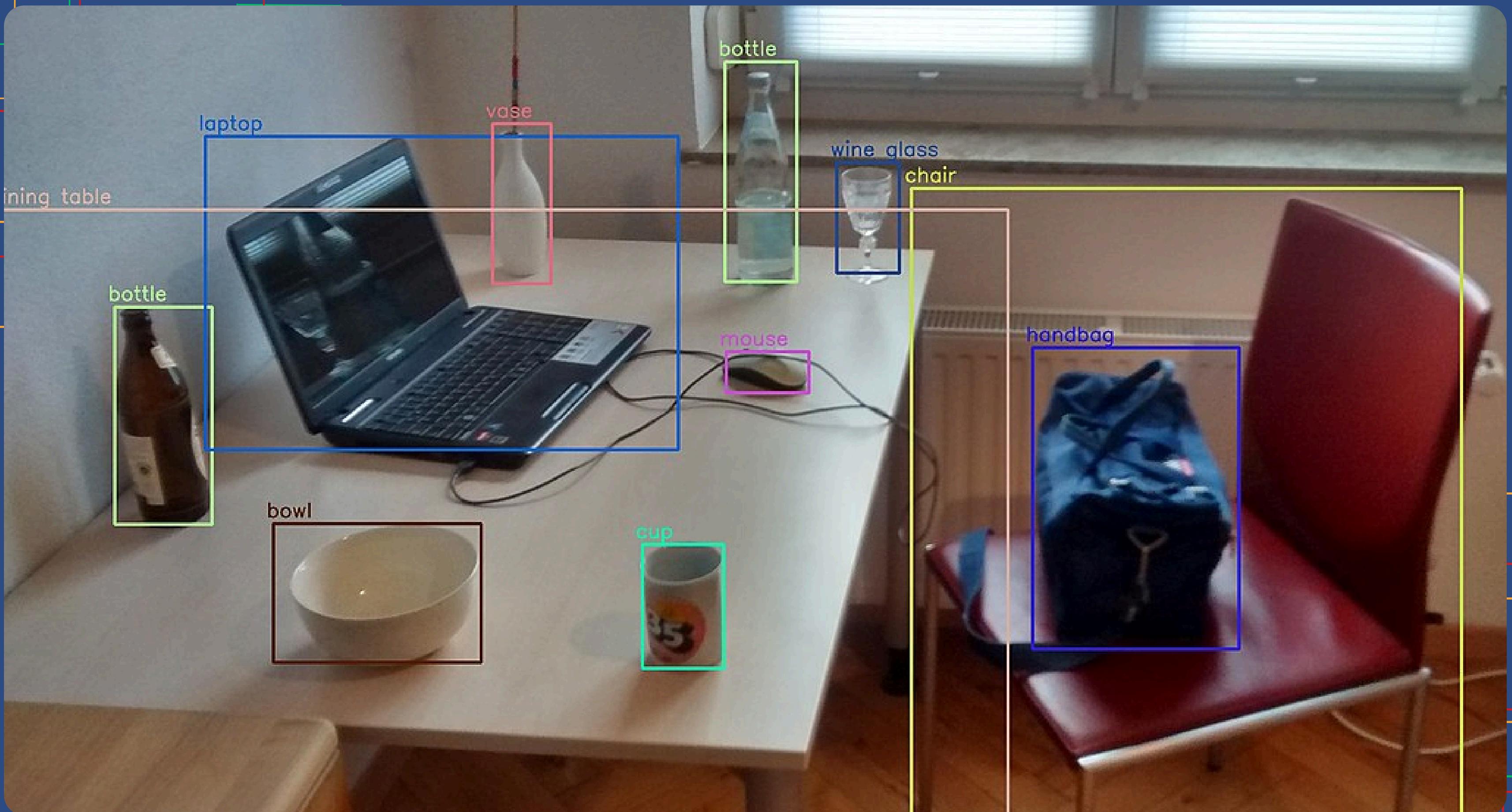
500+

Classes em modelos especializados
(datasets customizados e específicos)

A detecção de objetos além de prever qual a classe, identifica a localização do objeto na imagem.

Por exemplo, um classificador de cães e gatos vai retornar como resultado a classe (cão ou gato) e o grau de certeza sobre aquela predição. Caso na imagem tenha tanto um cão quanto um gato, treinamos classificadores multiclasse que poderão detectar os dois tipos de objeto.





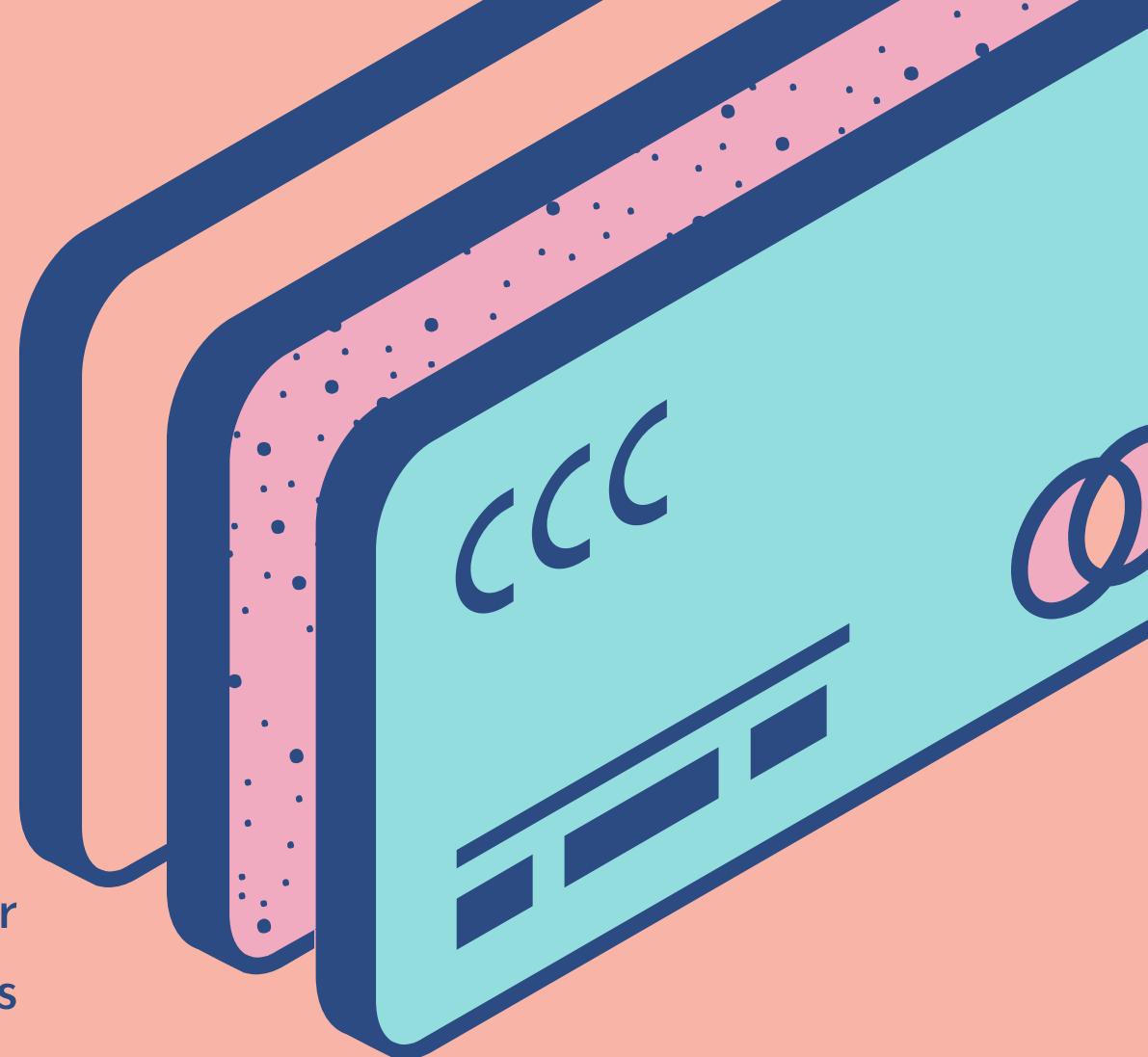
Vantagens do YOLO na Detecção de Objetos

Facilidade de Implementação

O YOLO revolucionou a acessibilidade da detecção de objetos ao permitir implementações complexas com apenas algumas linhas de código Python. Frameworks como Ultralytics YOLO eliminam barreiras técnicas através de instalação via pip, configuração automática de dependências e modelos pré-treinados prontos para uso.

A versatilidade de entrada aceita naturalmente imagens, vídeos, webcams e streams RTSP, enquanto as saídas podem ser personalizadas em formatos como JSON, XML ou anotações visuais, facilitando integração com sistemas existentes.

O deployment é igualmente flexível, funcionando desde dispositivos embarcados como Raspberry Pi até servidores cloud de alta performance. A mesma base de código pode ser adaptada para smartphones através de conversões automáticas para TensorFlow Lite, ou escalada para processamento massivo em clusters.



Vantagens do YOLO na Detecção de Objetos

Versatilidade de Aplicação

A versatilidade do YOLO transcende a detecção básica de objetos, adaptando-se organicamente a qualquer domínio que envolva análise visual. Na segurança, transforma vigilância tradicional em sistemas inteligentes que identificam comportamentos suspeitos. Na medicina, revoluciona diagnósticos por imagem detectando anomalias com precisão comparável à humana.

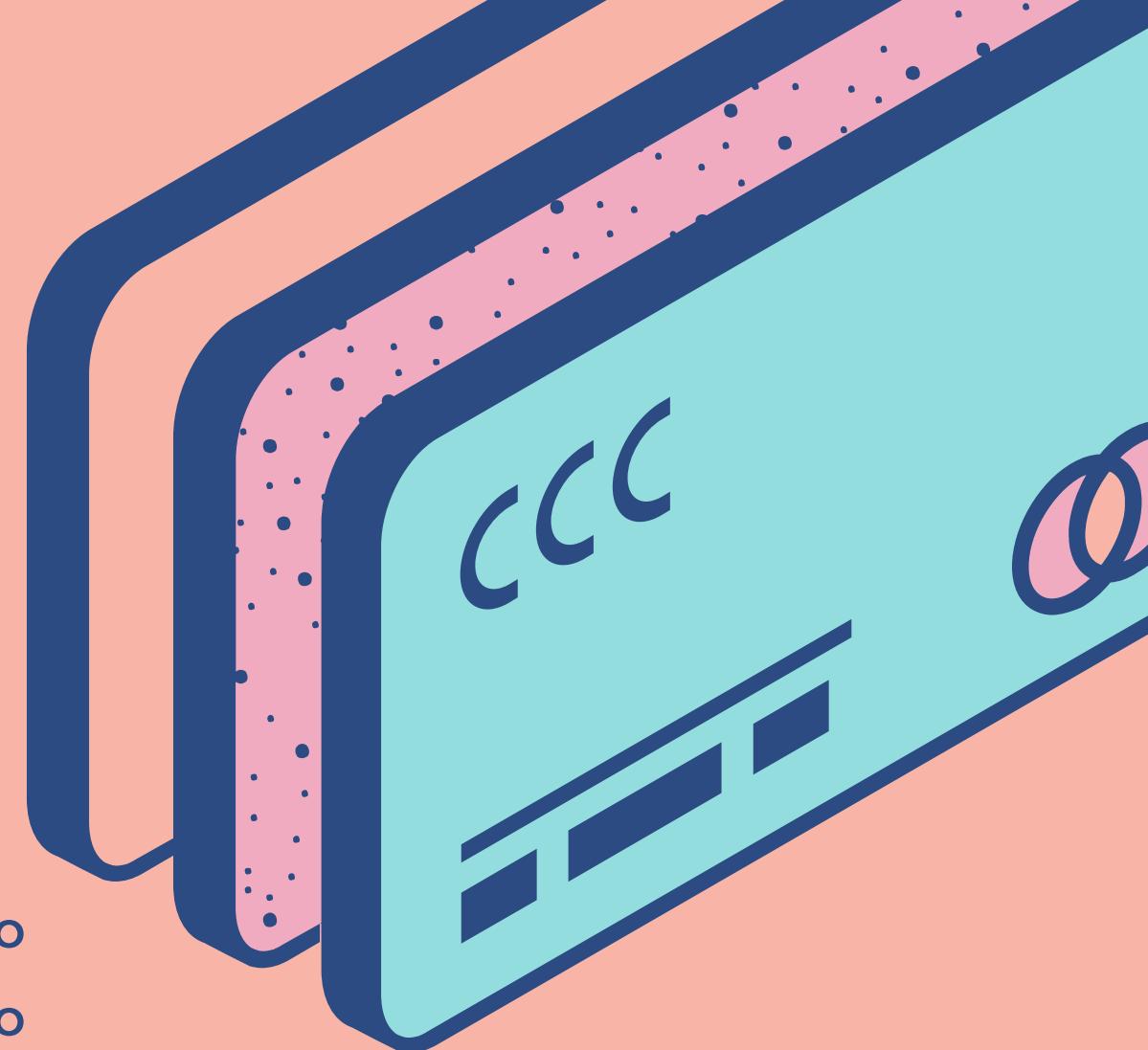
O que torna essa versatilidade impressionante é a facilidade de personalização através de transfer learning. Com datasets relativamente pequenos, organizações podem adaptar modelos pré-treinados para necessidades específicas, permitindo aplicações em agricultura para identificação de pragas, varejo para checkout automático, e praticamente qualquer setor que necessite de análise visual inteligente.



Vantagens do YOLO na Detecção de Objetos

Precisão e Eficiência

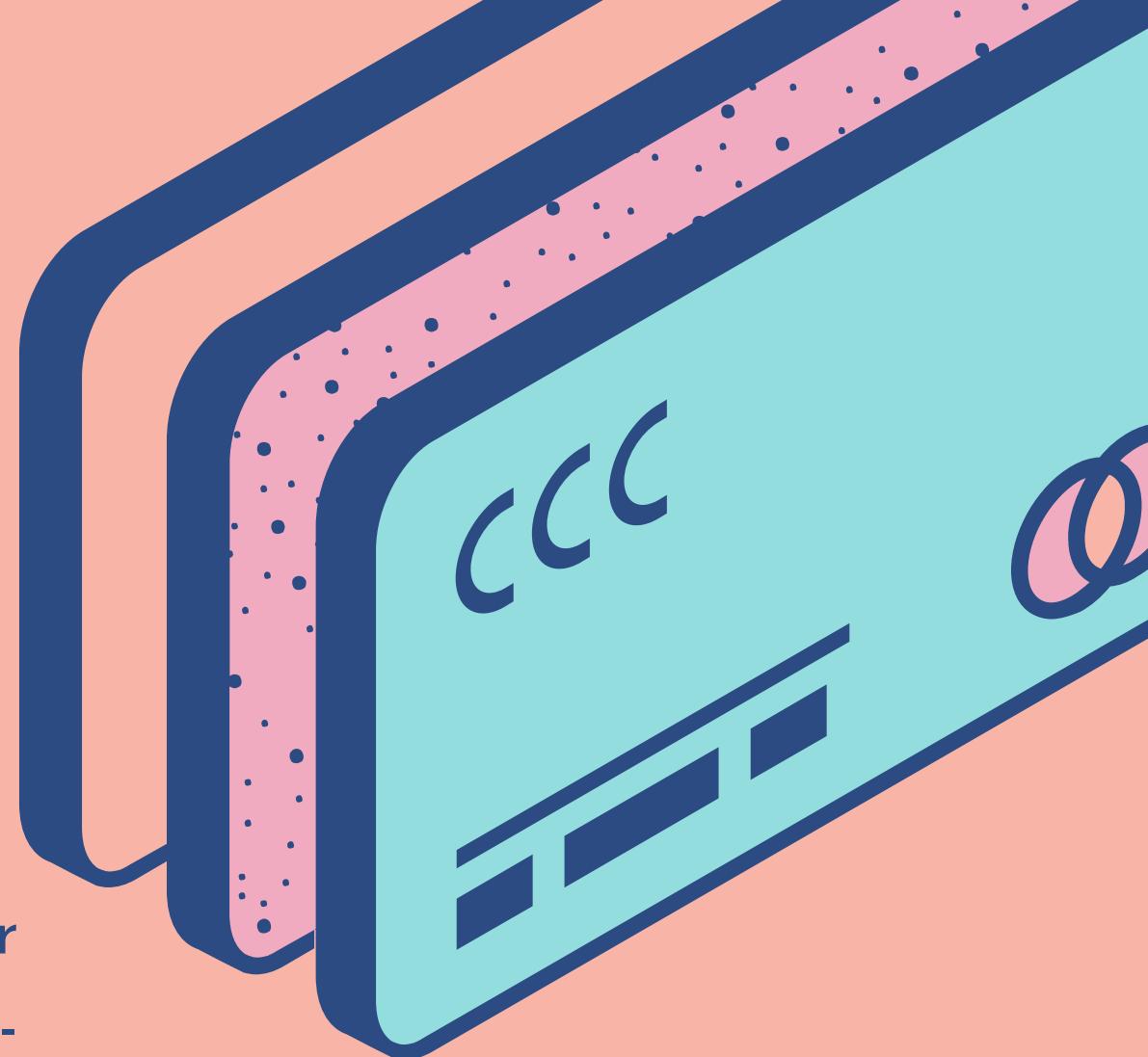
O YOLO consegue manter alta precisão na detecção de objetos mesmo processando imagens em velocidades extremas. Suas arquiteturas são otimizadas para balancear acurácia e performance computacional, oferecendo diferentes versões (nano, small, medium, large, extra-large) que permitem ajustar a relação precisão vs velocidade conforme a necessidade específica. Com mAP (mean Average Precision) chegando a 50%+ em datasets padrão, o YOLO também é eficiente energeticamente, sendo ideal para dispositivos móveis e embarcados.



Vantagens do YOLO na Detecção de Objetos

Velocidade

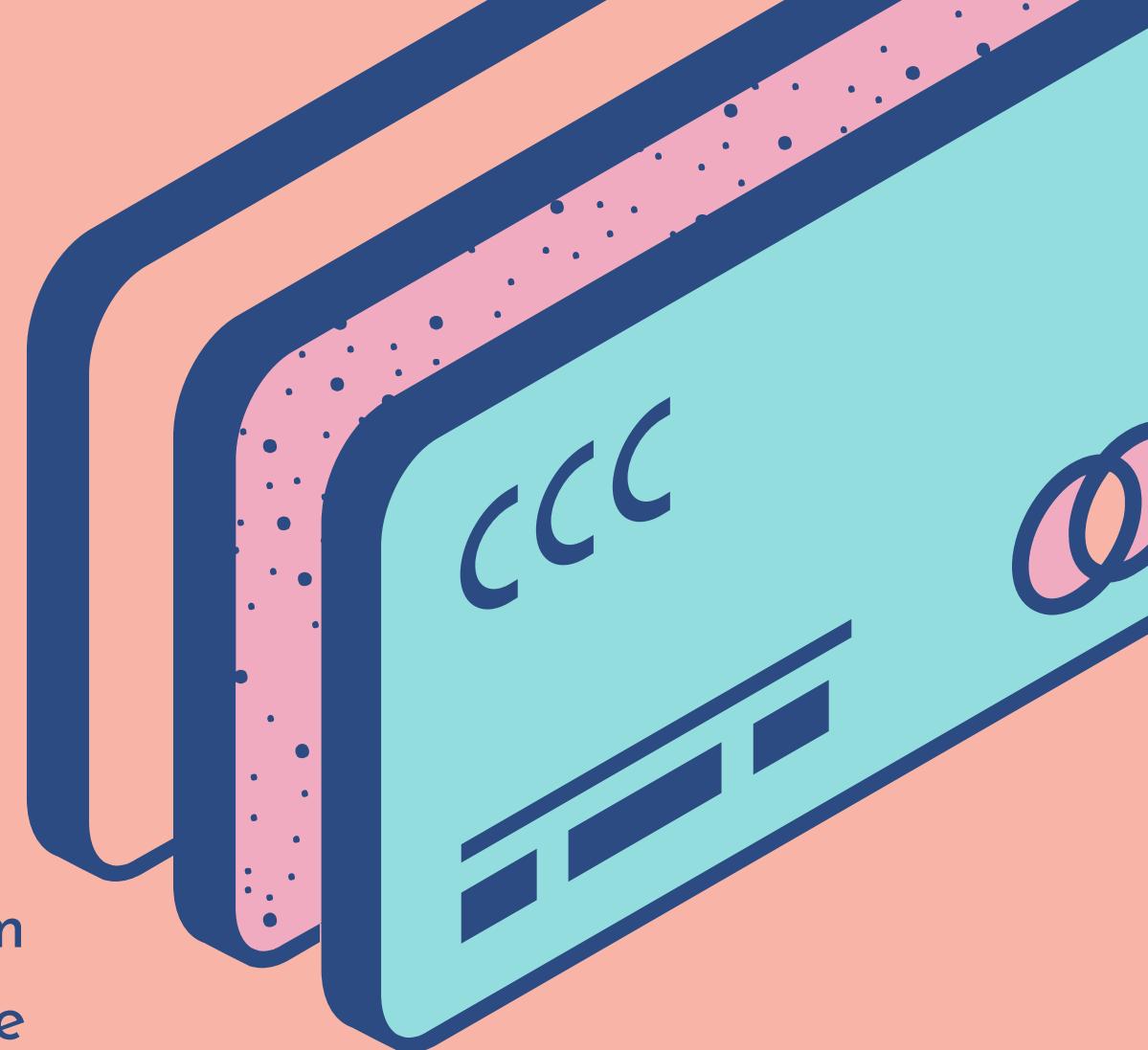
Uma das maiores vantagens do YOLO é sua capacidade de processar centenas de frames por segundo. Enquanto métodos tradicionais como R-CNN processam cerca de 0.05 FPS, o YOLO pode atingir mais de 300 FPS em versões otimizadas. Isso é possível porque analisa a imagem inteira em uma única passada pela rede neural, ao contrário de abordagens que fazem múltiplas análises sequenciais.



Vantagens do YOLO na Detecção de Objetos

Desempenho em Tempo Real

O YOLO foi projetado especificamente para aplicações em tempo real, com latência inferior a 10ms por inferência. Isso permite análise instantânea de streams de vídeo, câmeras de segurança e sistemas autônomos sem delay perceptível. É capaz de processar streaming contínuo sem necessidade de buffering.





Evolução

Partindo de sua já estabelecida eficiência com a predição em única passada, a evolução do YOLO focou em aprimorar outros aspectos do modelo, como a precisão em diferentes cenários.

A evolução continuou com o YOLOv3 em 2018, que, apesar de não ser mais rápido, resolveu um problema crucial: a detecção de objetos pequenos, graças à predição em 3 escalas diferentes. Posteriormente, o YOLOv4 consolidou esse avanço, demonstrando ser o melhor detector de objetos em tempo real da sua época, de acordo com as métricas do renomado dataset MS COCO.

Evolução das Versões

YOLOv1 (2015): A versão original. Pioneira na abordagem de detecção em uma única etapa. Utiliza uma rede neural única para prever bounding boxes e probabilidades de classe diretamente da imagem completa.

YOLOv2/YOLO9000(2016): Melhorias significativas na velocidade de processamento. Introduziu Batch Normalization, Anchor Boxes (caixas pré-definidas que ajudam o modelo a prever bounding boxes de diferentes formas e tamanhos) e detecção multi-escala.

YOLOv3 (2018): Aumentou a precisão com a arquitetura Darknet-53. Melhorou a detecção de objetos pequenos e introduziu múltiplas escalas para detecção.

YOLOv4 (2020): Foco em otimização e técnicas de treinamento. Utiliza CSPDarknet53 como backbone. Introduziu técnicas como Mosaic Data Augmentation, Self-Adversarial Training (SAT), Cross mini-batch Normalization (CmBN) e Modified Spatial Attention Module (SAM).

YOLOv5 (2020): Lançado pela Ultralytics. Mais leve e eficiente, com melhor desempenho e modelos menores. Construído em PyTorch, facilitando a implementação e o uso.

YOLOX (2021): Introduziu a detecção "anchor-free", simplificando o processo de treinamento e melhorando a performance em alguns cenários.

Evolução das Versões

YOLOv6 (2022): Focado em aplicações industriais, buscando um equilíbrio entre velocidade e precisão. Utiliza EfficientRep como backbone e Rep-PAN para o neck.

YOLOv7 (2022): Introduziu o conceito de "trainable bag-of-freebies", com refinamentos no treinamento e otimização. Foco em Model Re-parameterization, Dynamic Label Assignment e Extended and Compound Scaling.

YOLOv8 (2023): Lançado pela Ultralytics. Redesenho da arquitetura com detecção "anchor-free" dinâmica. Suporta não apenas detecção de objetos, mas também segmentação e pose estimation.

YOLOv9 (2024): Introduziu extração de características baseada em Transformer e detecção multi-escala.

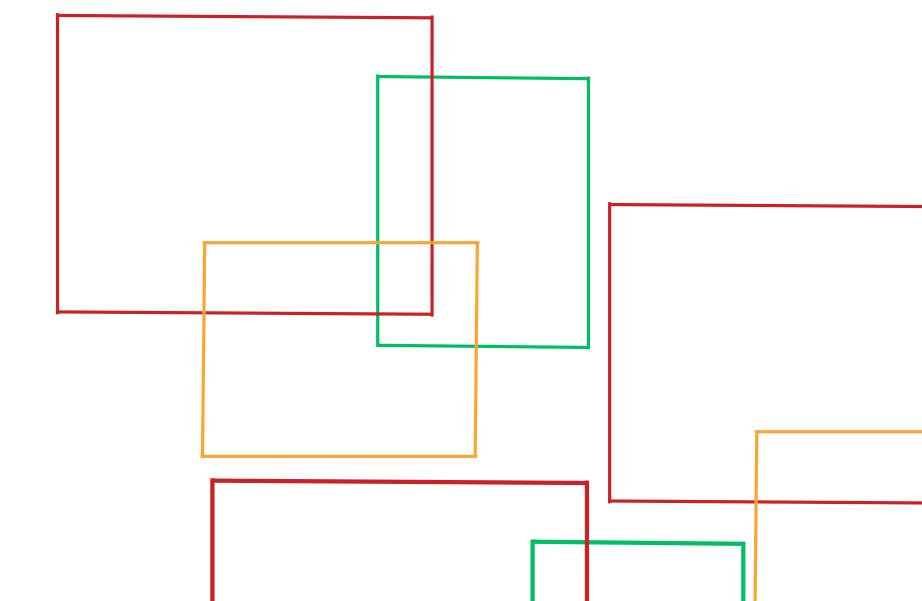
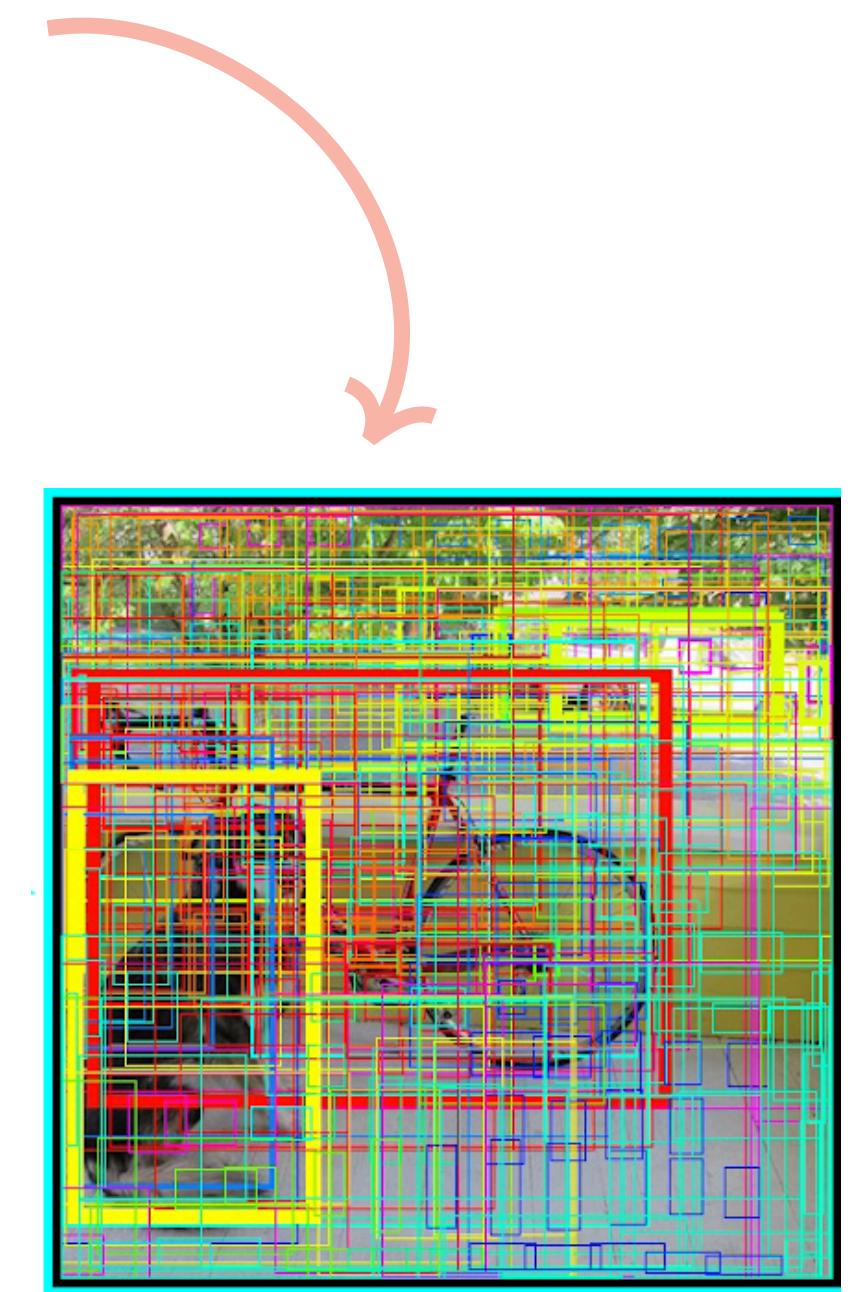
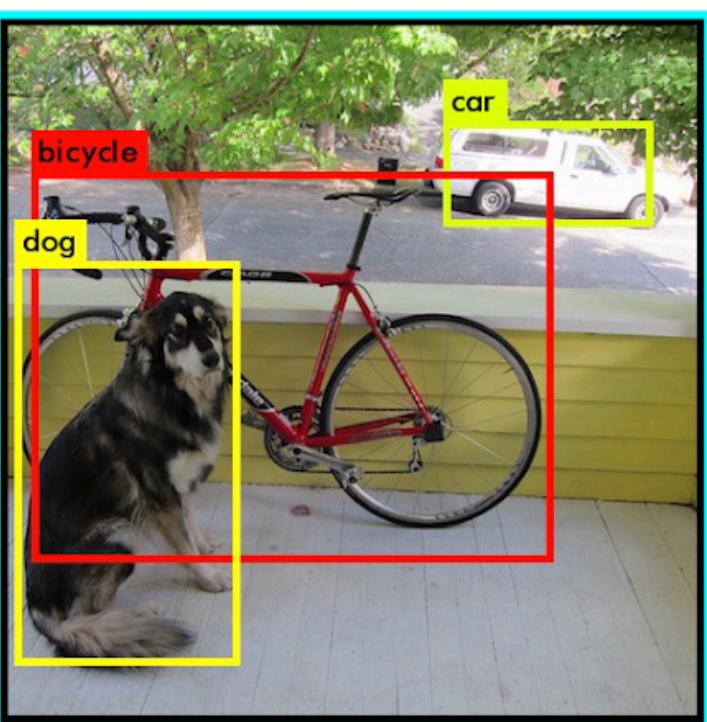
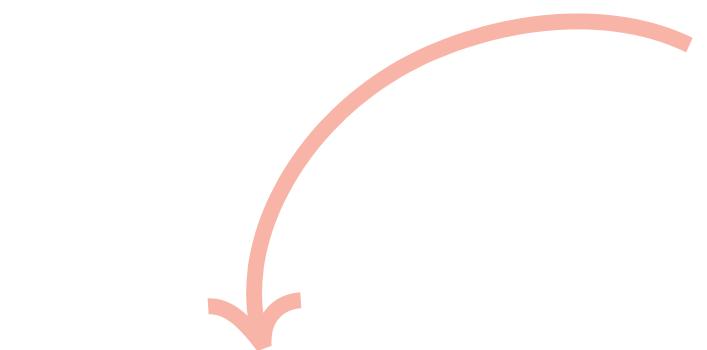
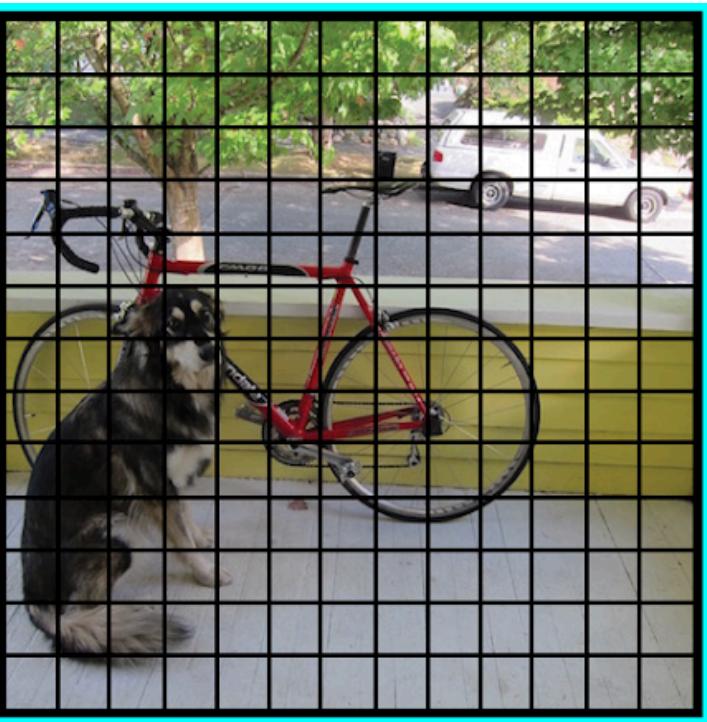
YOLOv10 (2024): Foco em treinamento consciente de quantização e design amigável para hardware de borda (edge AI).

YOLOv11 (2025): Modelos híbridos CNN-Transformer, prometendo maior velocidade, precisão e eficiência.

Funcionamento

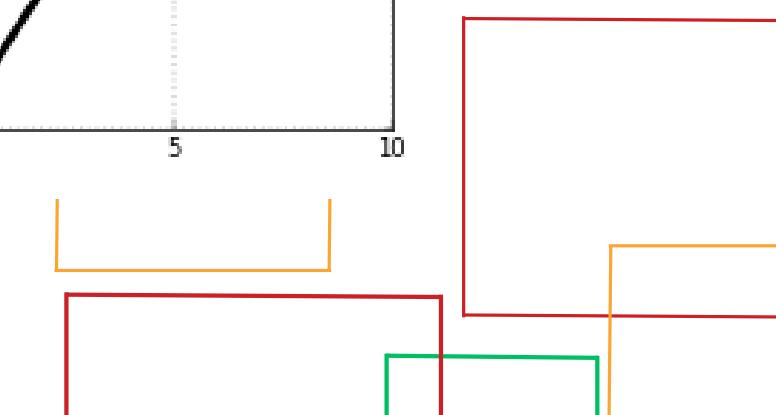
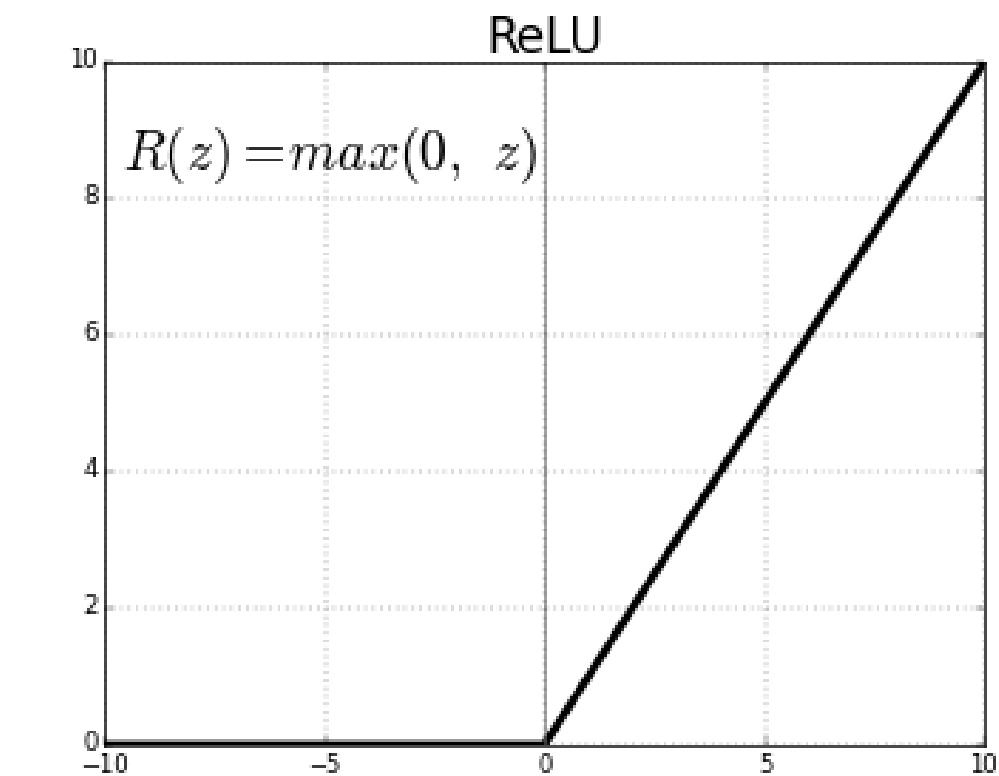
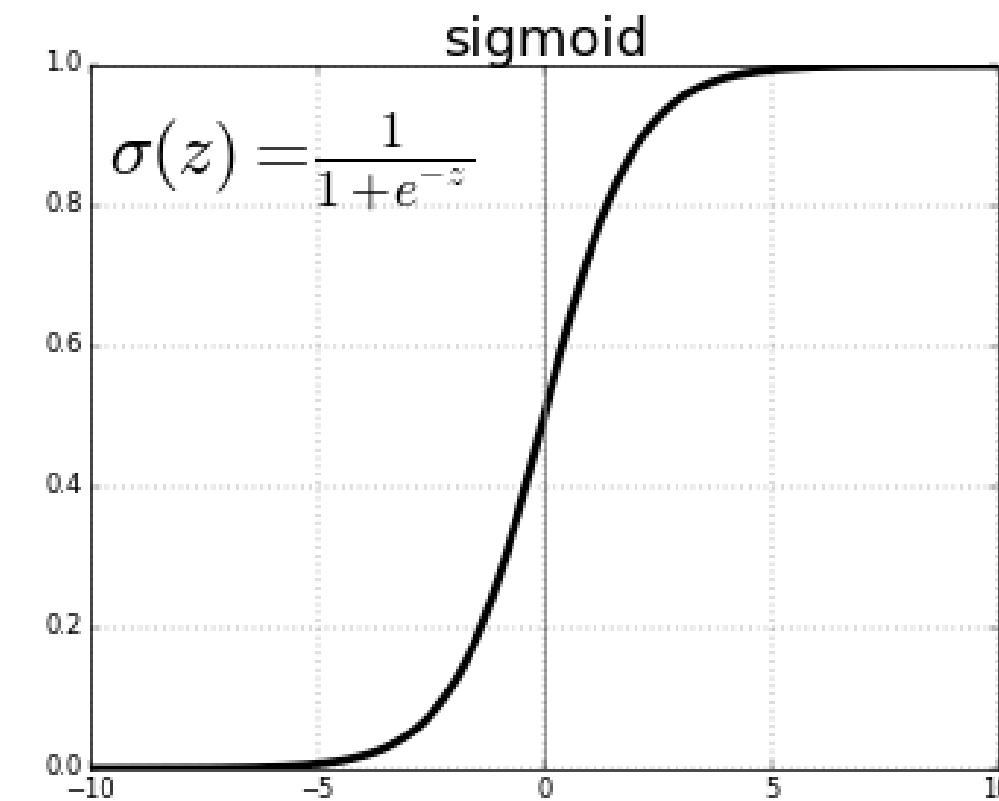
O algoritmo YOLO recebe uma imagem como entrada e utiliza uma rede neural convolucional profunda para detectar objetos. A arquitetura da CNN que forma o backbone do YOLO é pré-treinada (geralmente com ImageNet) e, em seguida, adaptada para a tarefa de detecção. A camada final totalmente conectada do YOLO prevê as probabilidades de classe e as coordenadas das caixas delimitadoras.

O YOLO divide a imagem de entrada em uma grade $S \times S$. Se o centro de um objeto cair em uma célula da grade, essa célula é responsável por detectar o objeto. Cada célula da grade prevê B caixas delimitadoras e scores de confiança para essas caixas. Esses scores de confiança refletem o quanto confiante o modelo está de que a caixa contém um objeto ou quanto precisa ele considerar a caixa prevista.



Funções de Ativação

As funções de ativação são componentes essenciais das redes neurais que determinam como os neurônios "respondem" às entradas recebidas. Elas controlam quais informações passam adiante na rede e quais são bloqueadas. No contexto do YOLO, diferentes funções de ativação são estrategicamente utilizadas em diferentes partes da arquitetura da rede para otimizar tanto a velocidade quanto a precisão da detecção de objetos.



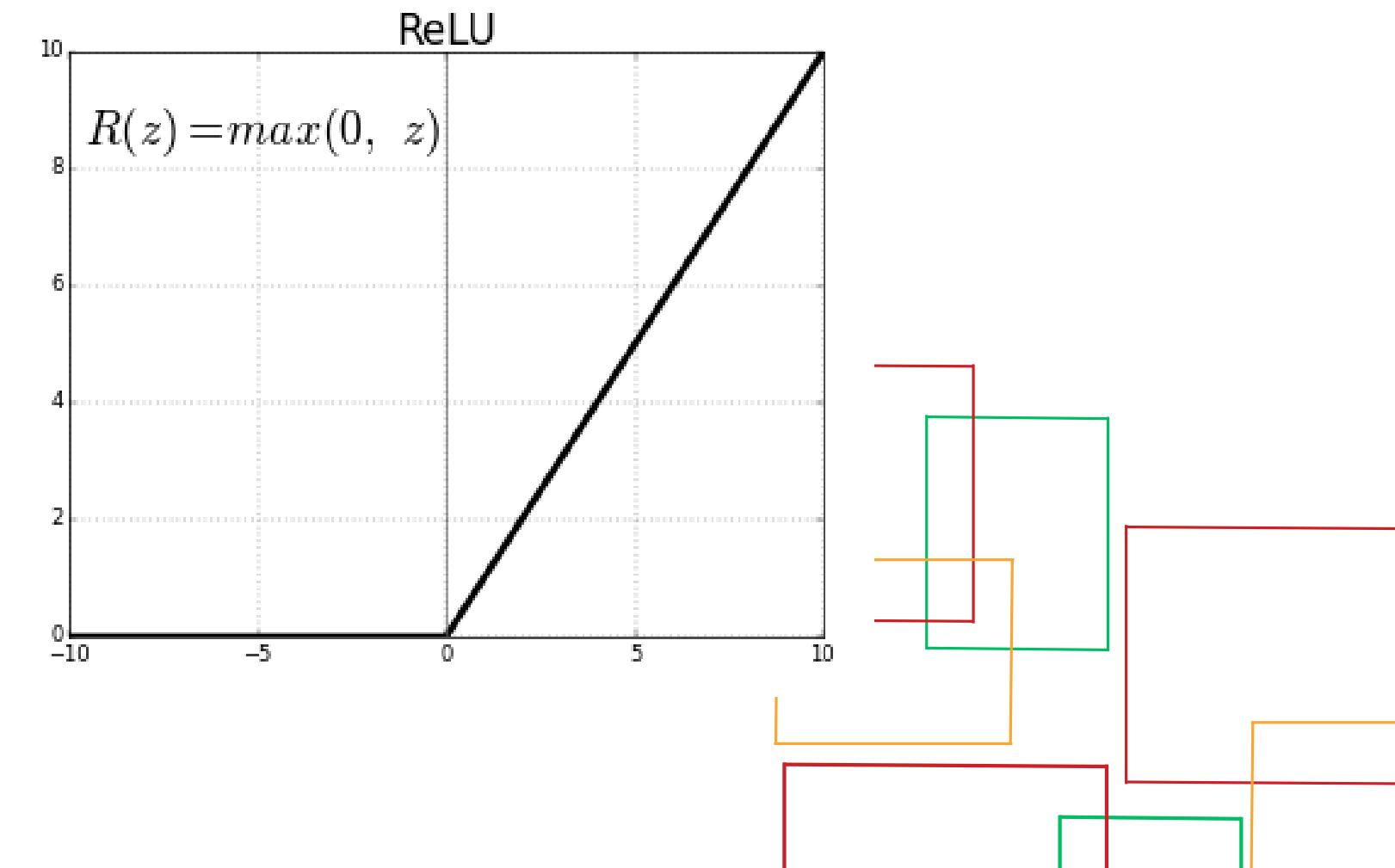
ReLU (Rectified Linear Unit)

Usada nas camadas intermediárias (hidden layers)

A função ReLU é definida matematicamente como $f(x) = \max(0, x)$, o que significa que se a entrada for positiva, ela passa inalterada, mas se for negativa ou zero, a saída será sempre zero. Esta simplicidade matemática traz enormes benefícios computacionais.

No YOLO, a ReLU é amplamente utilizada nas camadas convolucionais intermediárias porque oferece várias vantagens cruciais. Primeiro, sua computação é extremamente rápida, envolvendo apenas uma simples comparação, o que é fundamental para o processamento em tempo real. Segundo, ela resolve o problema dos gradientes que desaparecem durante o treinamento, permitindo que redes mais profundas sejam treinadas eficientemente.

A ReLU também cria sparsity na rede, ou seja, muitos neurônios ficam com valor zero, o que economiza recursos computacionais e memória. Nas camadas de extração de características do YOLO, como no backbone Darknet, a ReLU permite que a rede aprenda características complexas como bordas, texturas e formas de objetos de maneira eficiente.



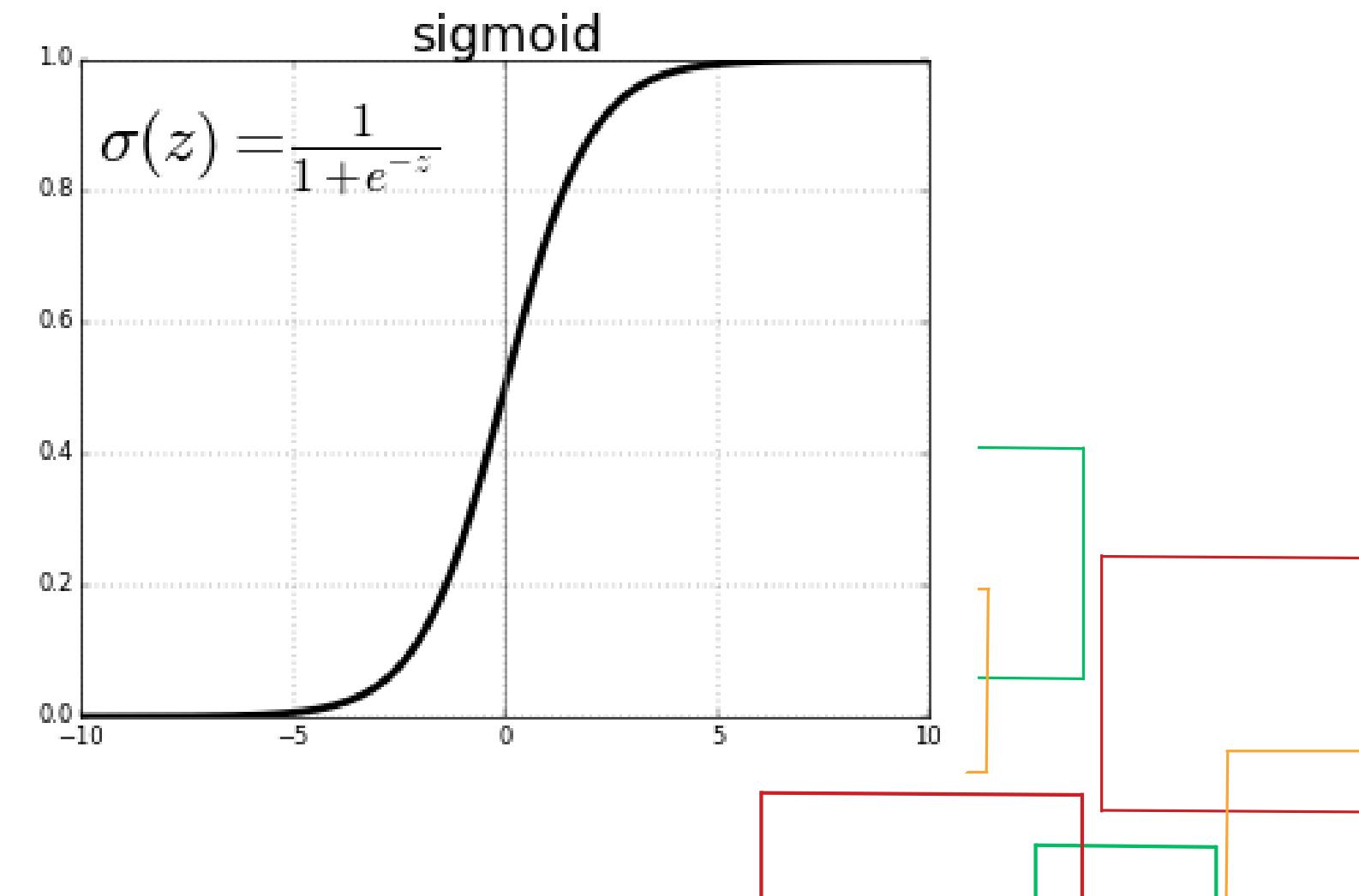
Sigmoid $\sigma(x) = 1/(1 + e^{-x})$

Usada nas camadas de saída (output layers)

A função sigmoid tem uma característica muito especial: independentemente do valor de entrada, sua saída sempre estará entre 0 e 1, criando uma curva suave em formato de "S". Esta propriedade matemática a torna perfeita para representar probabilidades e valores normalizados.

No YOLO, a sigmoid é estrategicamente utilizada nas camadas de saída porque as previsões da rede precisam estar em formatos específicos. Para o confidence score, a sigmoid converte qualquer valor real em uma probabilidade válida entre 0 e 1, representando a confiança de que existe um objeto na bounding box. Para as coordenadas das bounding boxes, especificamente x e y do centro, a sigmoid normaliza esses valores para ficarem entre 0 e 1, representando a posição relativa dentro da célula do grid.

A sigmoid também é fundamental para gerar probabilidades de classe válidas. Quando o YOLO precisa determinar se um objeto detectado é um carro, pessoa ou bicicleta, a sigmoid garante que essas probabilidades estejam no formato correto para interpretação e para o cálculo da loss function durante o treinamento.



Combinação Estratégica com YOLO

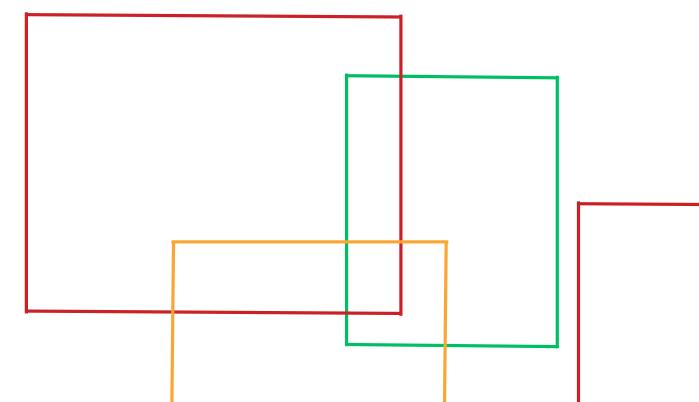
A arquitetura do YOLO utiliza uma combinação inteligente dessas funções de ativação. O fluxo típico começa com a imagem de entrada passando pelas camadas convolucionais que utilizam ReLU para extrair características. Essas camadas intermediárias com ReLU são responsáveis por identificar padrões como bordas, texturas e formas mais complexas.

Nas camadas finais, a sigmoid é aplicada para converter essas características extraídas em previsões interpretáveis. Por exemplo, um valor interno da rede pode ser -2.3, mas após passar pela sigmoid, se torna 0.09, representando 9% de confiança na presença de um objeto. Da mesma forma, coordenadas podem ser normalizadas para $x=0.3$ e $y=0.7$, indicando que o centro do objeto está a 30% da largura e 70% da altura da célula específica do grid.

Vantagens de Utilização

A utilização de ReLU nas camadas intermediárias traz benefícios significativos para o treinamento e desempenho do YOLO. O treino se torna mais rápido devido à simplicidade computacional, há melhor propagação de gradientes permitindo redes mais profundas, e a sparsity criada reduz o overfitting naturalmente.

Por outro lado, a sigmoid na saída garante que os outputs sejam interpretáveis como probabilidades, com valores sempre limitados entre 0 e 1, e compatíveis com as loss functions utilizadas durante o treinamento. Esta combinação cria um sistema onde a velocidade de processamento não compromete a qualidade das previsões.



Passo a passo

1. Divisão da Imagem em Grid

O algoritmo primeiro divide a imagem de entrada em um grid de $S \times S$ células, cujo tamanho pode variar conforme a versão do modelo.

Cada célula do grid é responsável por detectar objetos cujo centro está dentro dela

O tamanho do grid varia conforme a versão do YOLO:

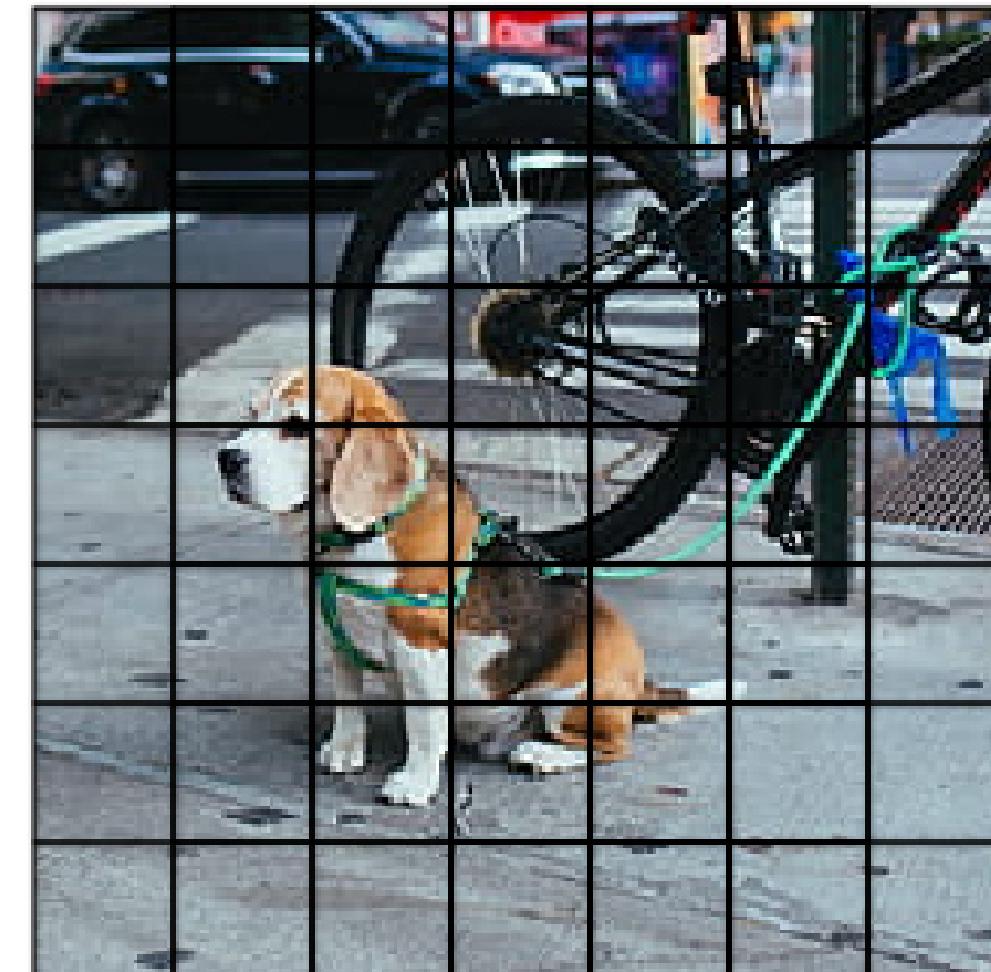
- **YOLOv1:** $7 \times 7 = 49$ células
- **YOLOv2:** $13 \times 13 = 169$ células
- **YOLOv3/v4/v5:** Múltiplas escalas (13×13 , 26×26 , 52×52)

Vantagens:

Processamento paralelo: Todas as células são processadas simultaneamente

Velocidade: Uma única passada pela rede neural

Eficiência computacional: Não precisa de múltiplas janelas deslizantes



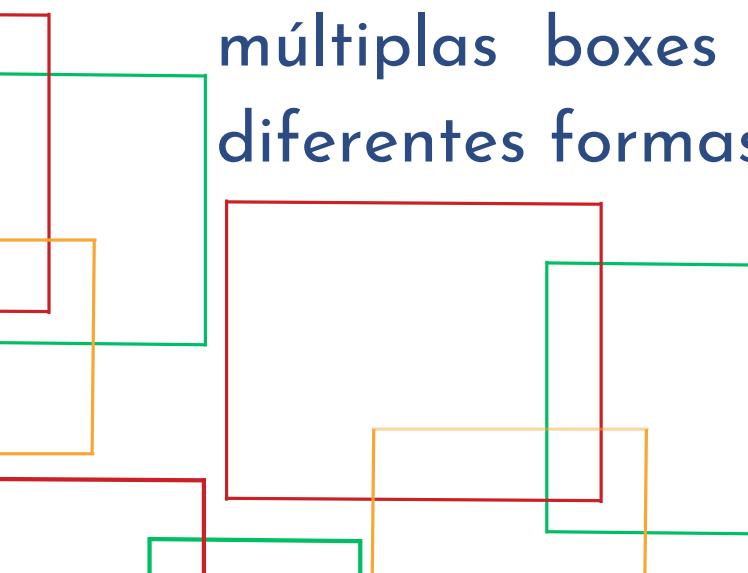
Passo a passo

2. Previsão por Célula do Grid

Cada célula do grid é responsável por prever múltiplas caixas delimitadoras e uma pontuação de confiança que indica a certeza de conter um objeto.

O que cada célula prevê:

Bounding Boxes (Caixas Delimitadoras): As bounding boxes delimitam onde os objetos estão na imagem. Cada uma contém: coordenadas do centro (x, y) relativas à célula, e dimensões (w, h) proporcionais à imagem inteira. Cada célula prevê múltiplas boxes (2-5) para capturar objetos com diferentes formas.



Confidence Score (Pontuação de Confiança): Combina duas informações: $P(\text{Objeto})$ (probabilidade de haver um objeto) e IoU (qualidade da localização).

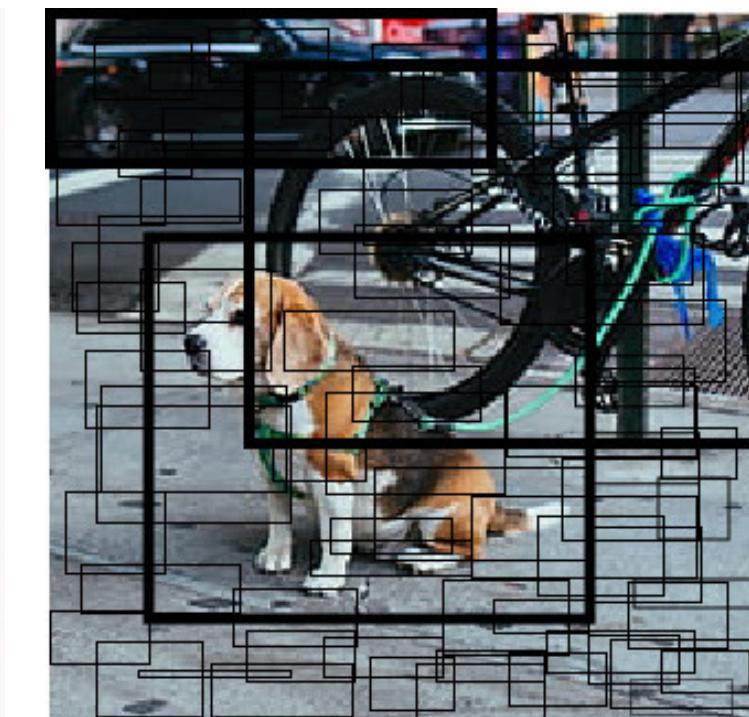
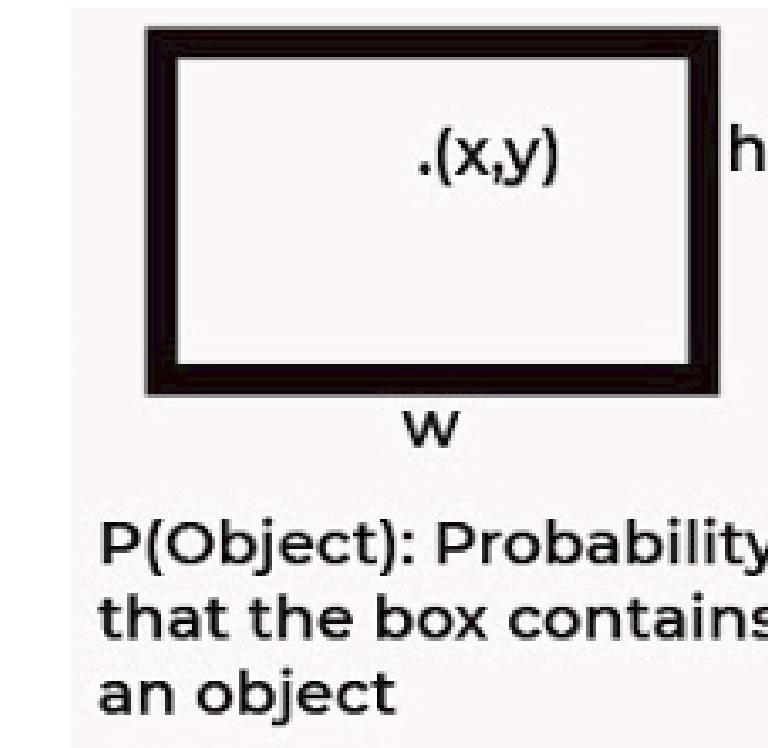
Fórmula: $\text{Confidence} = P(\text{Objeto}) \times \text{IoU}$.

Score alto = objeto presente + localização precisa.

IoU (Intersection over Union): Mede sobreposição entre boxes.

$\text{IoU} = \text{Área_Interseção} / \text{Área_União}$.

Valores: 1.0 = sobreposição perfeita,
0.0 = nenhuma sobreposição.



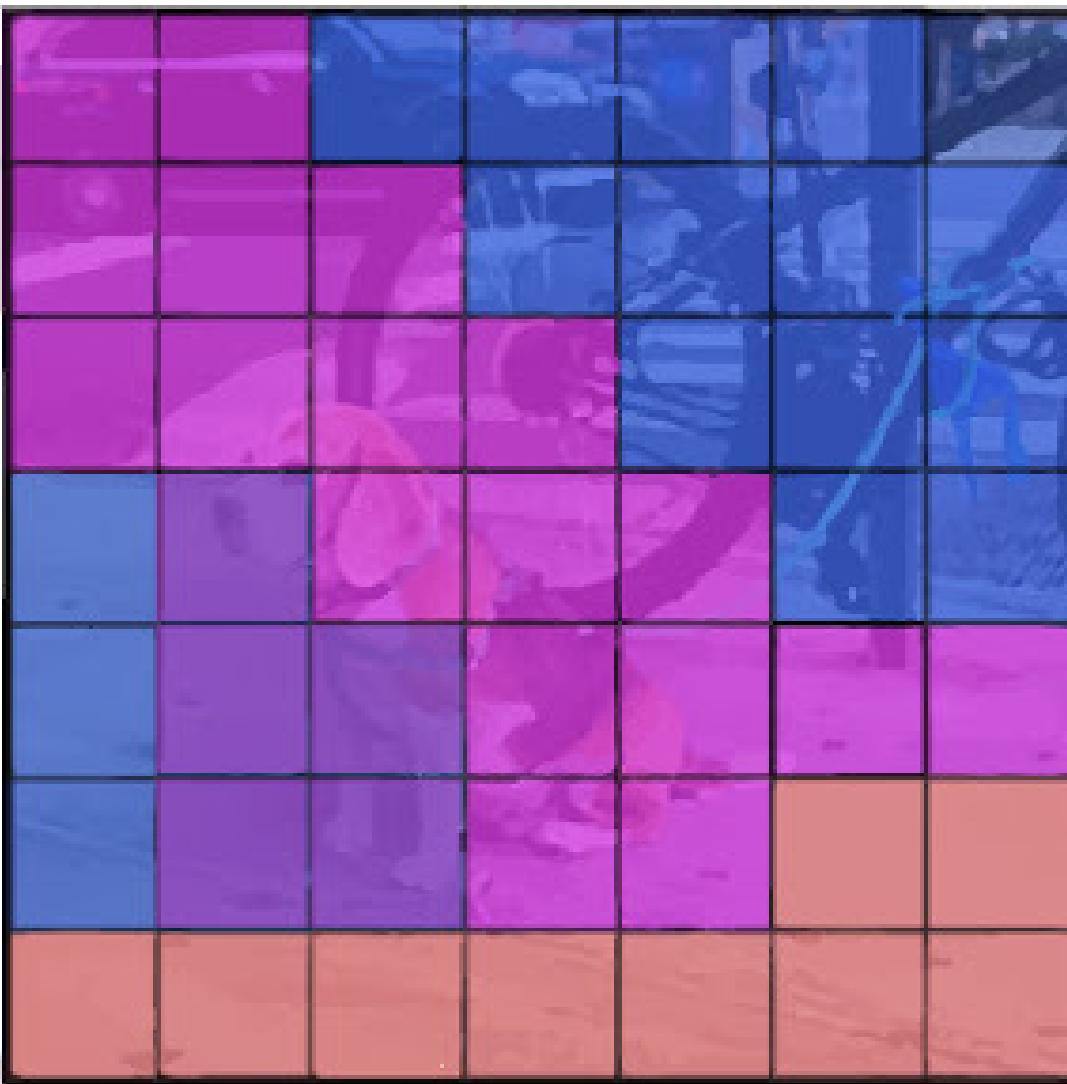
Passo a passo

3. Classificação de Objetos

Cada célula prevê probabilidades condicionais para cada classe possível, representando $P(\text{Classe}_i | \text{Objeto})$. Por exemplo: [0.8 pessoa, 0.1 carro, 0.05 bicicleta]. O score final combina confiança e classificação:

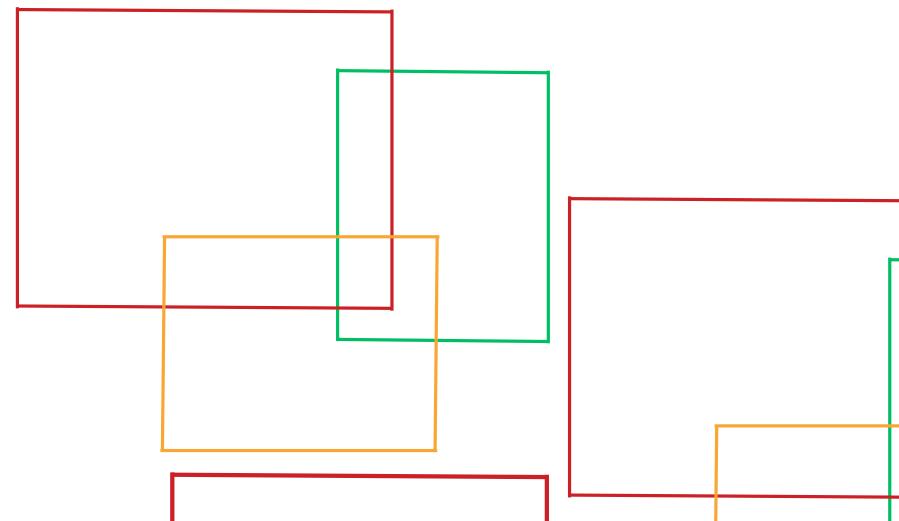
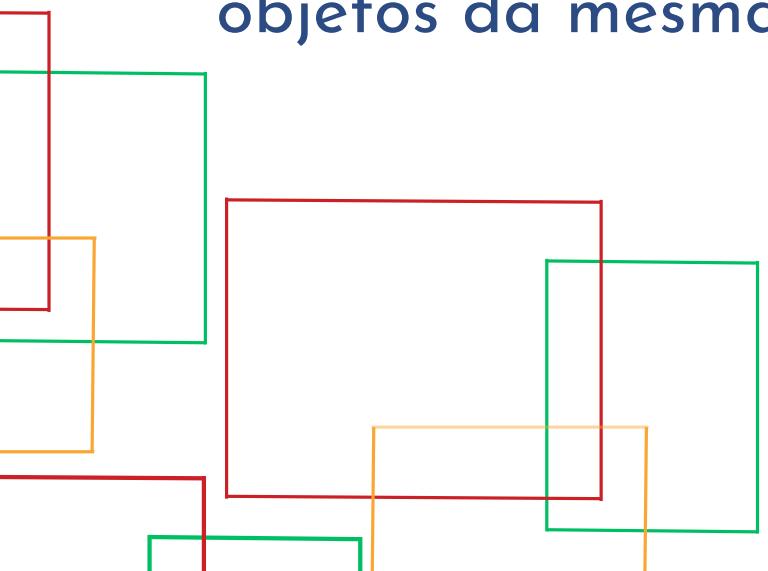
$$\text{Score_final} = \text{Confidence} \times P(\text{Classe}_i | \text{Objeto}).$$

Limitação importante: Uma célula prevê apenas uma classe, mesmo que contenha múltiplas bounding boxes. Isso causa dificuldades quando objetos da mesma classe estão muito próximos.



Dog

Cycle

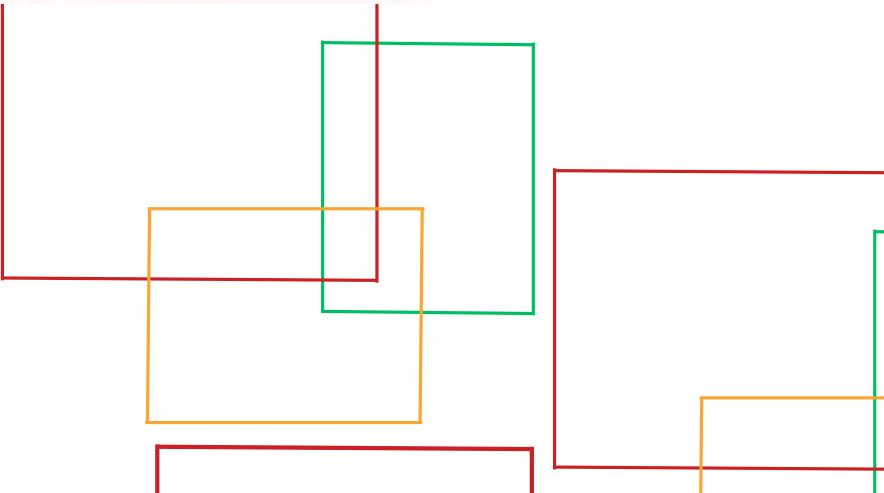
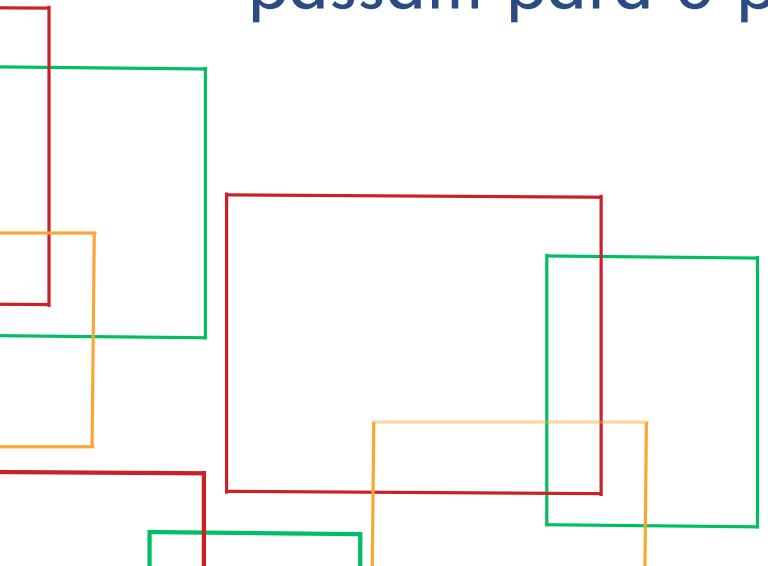
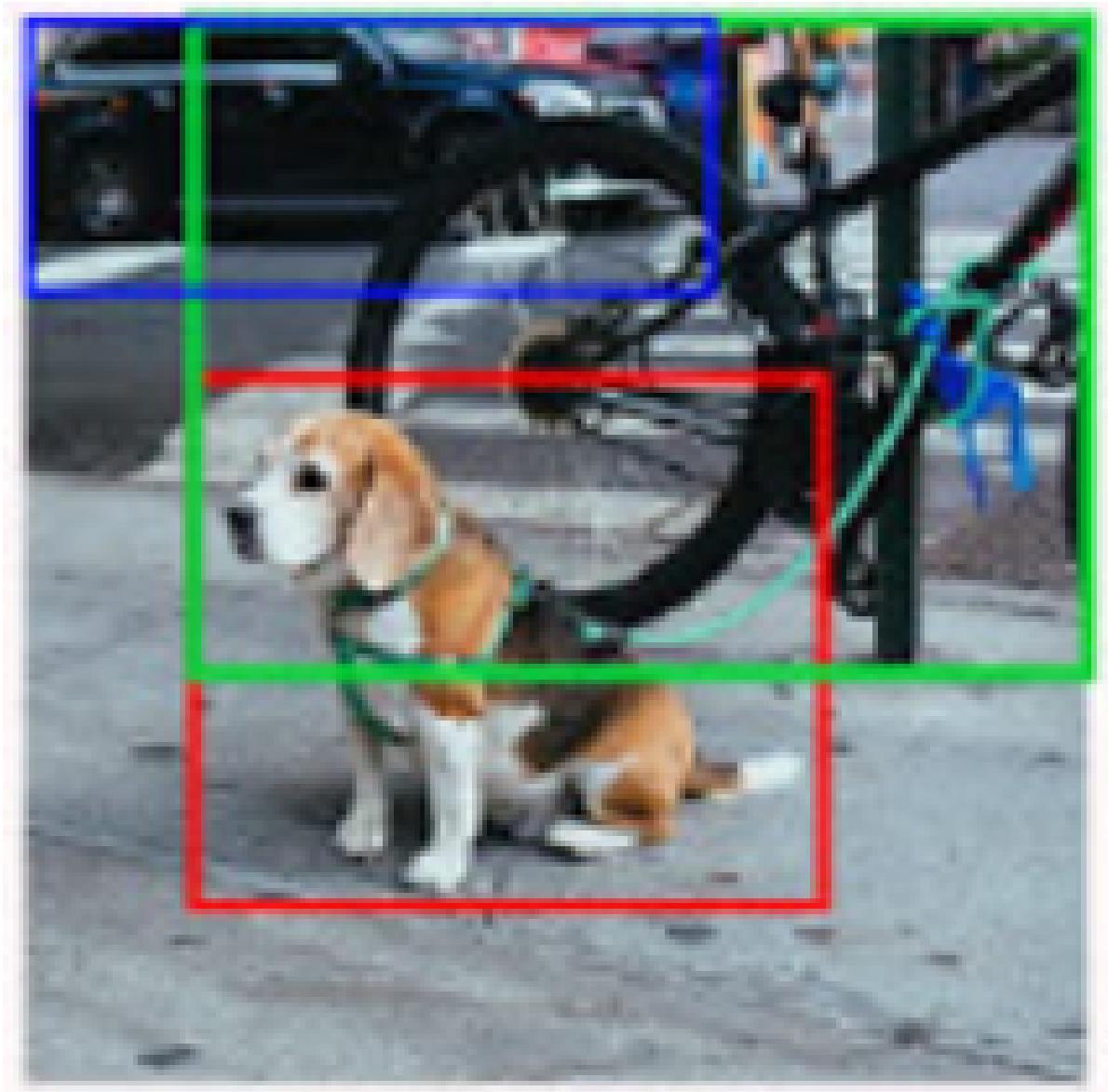


Passo a passo

4. Limiar de Confiança (Threshold)

Nesse passo, um limiar de confiança (threshold) é aplicado para filtrar as caixas com pontuação baixa, mantendo apenas as detecções mais prováveis. O threshold filtra essas detecções antes do processamento final.

Como funciona: Todas as detecções com score final abaixo do threshold (tipicamente 0.3 a 0.7) são descartadas imediatamente. **Por exemplo**, se o threshold é 0.5, apenas detecções com $\text{score} \geq 0.5$ passam para o próximo passo.



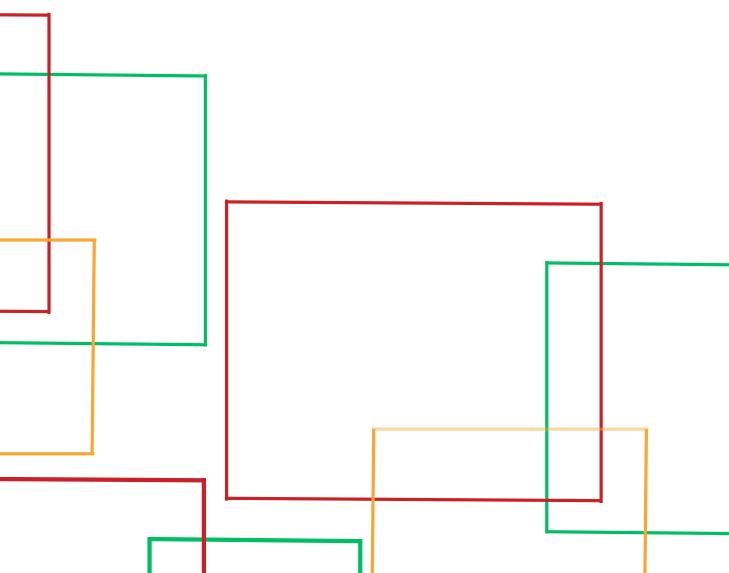
Passo a passo

5. Non-Maximum Suppression (NMS)

Após o threshold, ainda temos múltiplas detecções do mesmo objeto - diferentes bounding boxes podem detectar a mesma pessoa ou carro. O NMS resolve isso selecionando apenas a melhor detecção por objeto.

Como funciona:

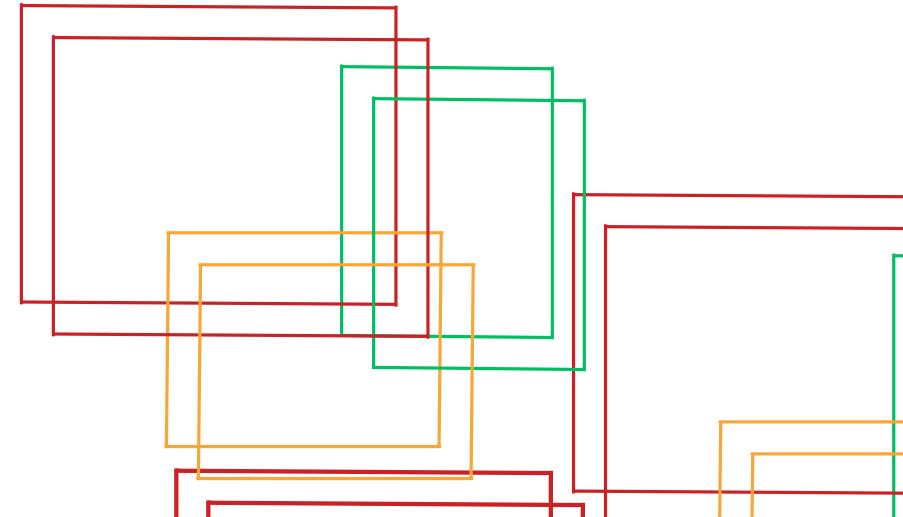
1. Ordena todas as detecções restantes por score (maior → menor)
2. Seleciona a detecção com maior score
3. Remove todas as outras detecções que tenham $\text{IoU} > \text{threshold_NMS}$ (tipicamente 0.4-0.6) com a selecionada
4. Repete o processo com as detecções restantes



Métricas de Avaliação

Intersection over Union (IoU):

- Boa detecção: $\text{IoU} > 0.5$
- Excelente detecção: $\text{IoU} > 0.7$





Processo de Treinamento

Preparação dos Dados

O treinamento do YOLO começa com a preparação cuidadosa do dataset. Cada imagem precisa estar acompanhada de suas anotações, que contêm as coordenadas das bounding boxes e as classes dos objetos. Essas anotações geralmente estão em formato texto, onde cada linha representa um objeto com suas coordenadas normalizadas (`x_center`, `y_center`, `width`, `height`) e o ID da classe.

O dataset é então dividido em conjuntos de treinamento, validação e teste. Durante a preparação, as imagens passam por técnicas de data augmentation como rotação, redimensionamento, alteração de brilho e contraste para aumentar a diversidade dos dados e melhorar a generalização do modelo.



Processo de Treinamento

Função de Perda (Loss Function)

O YOLO utiliza uma função de perda composta que combina três componentes principais: localização, confiança e classificação. Esta função multi-part é fundamental para o sucesso do treinamento.

Loss de Localização: Penaliza erros nas coordenadas das bounding boxes. Utiliza erro quadrático médio para as coordenadas (x , y) e para as dimensões (w , h), mas com pesos diferentes para dar mais importância a objetos pequenos.

Loss de Confiança: Tem duas partes - uma para células que contêm objetos (deve ser alta) e outra para células vazias (deve ser baixa). O peso da loss para células vazias é menor para evitar que dominem o treinamento, já que a maioria das células não contém objetos.

Loss de Classificação: Usa erro quadrático médio entre as probabilidades previstas e reais das classes, mas apenas para células que realmente contêm objetos.



Processo de Treinamento

Processo de Treinamento

O treinamento segue o processo padrão de redes neurais com algumas especificidades do YOLO. Inicialmente, a rede pode ser pré-treinada em ImageNet para aprender características gerais de imagens, depois fine-tuned para detecção de objetos

Durante cada época, as imagens são processadas em batches. Para cada imagem, a rede faz previsões para todas as células do grid, calcula a loss comparando com as anotações reais, e ajusta os pesos através de backpropagation. O otimizador (geralmente SGD ou Adam) atualiza os parâmetros da rede baseado nos gradientes calculados.

Um aspecto importante é o warm-up learning rate, onde a taxa de aprendizado começa baixa e aumenta gradualmente nas primeiras épocas, depois diminui ao longo do treinamento. Isso ajuda na estabilidade do treinamento.



Processo de Treinamento

Monitoramento e Validação

Durante o treinamento, várias métricas são monitoradas para avaliar o progresso. A loss total deve diminuir consistentemente, mas é importante observar cada componente separadamente. A loss de localização indica quanto bem o modelo está aprendendo a posicionar as boxes, enquanto a loss de classificação mostra a capacidade de reconhecer objetos.

O conjunto de validação é usado para detectar overfitting. Se a performance no treino continua melhorando mas no validação estagna ou piora, é sinal de overfitting. Técnicas como early stopping podem ser aplicadas baseadas na performance de validação.

Visualizações periódicas das previsões em imagens de validação ajudam a entender qualitativamente como o modelo está evoluindo, mostrando se está aprendendo a detectar objetos corretamente ou se há padrões problemáticos.

Overfitting: Com datasets pequenos, o modelo pode decorar os exemplos. Técnicas como dropout, data augmentation e regularização são essenciais para manter a generalização.



Processo de Treinamento

Resultado Final

Após o treinamento completo, o modelo deve ser capaz de detectar objetos das classes treinadas com boa precisão e velocidade. O sucesso é medido pela capacidade de generalizar para imagens não vistas durante o treinamento, mantendo alta precisão na detecção e baixa taxa de falsos positivos.

O modelo treinado consiste nos pesos da rede neural que codificam todo o conhecimento adquirido sobre detecção de objetos. Estes pesos podem então ser utilizados para inferência em tempo real, cumprindo o objetivo principal do YOLO de combinar precisão com velocidade.

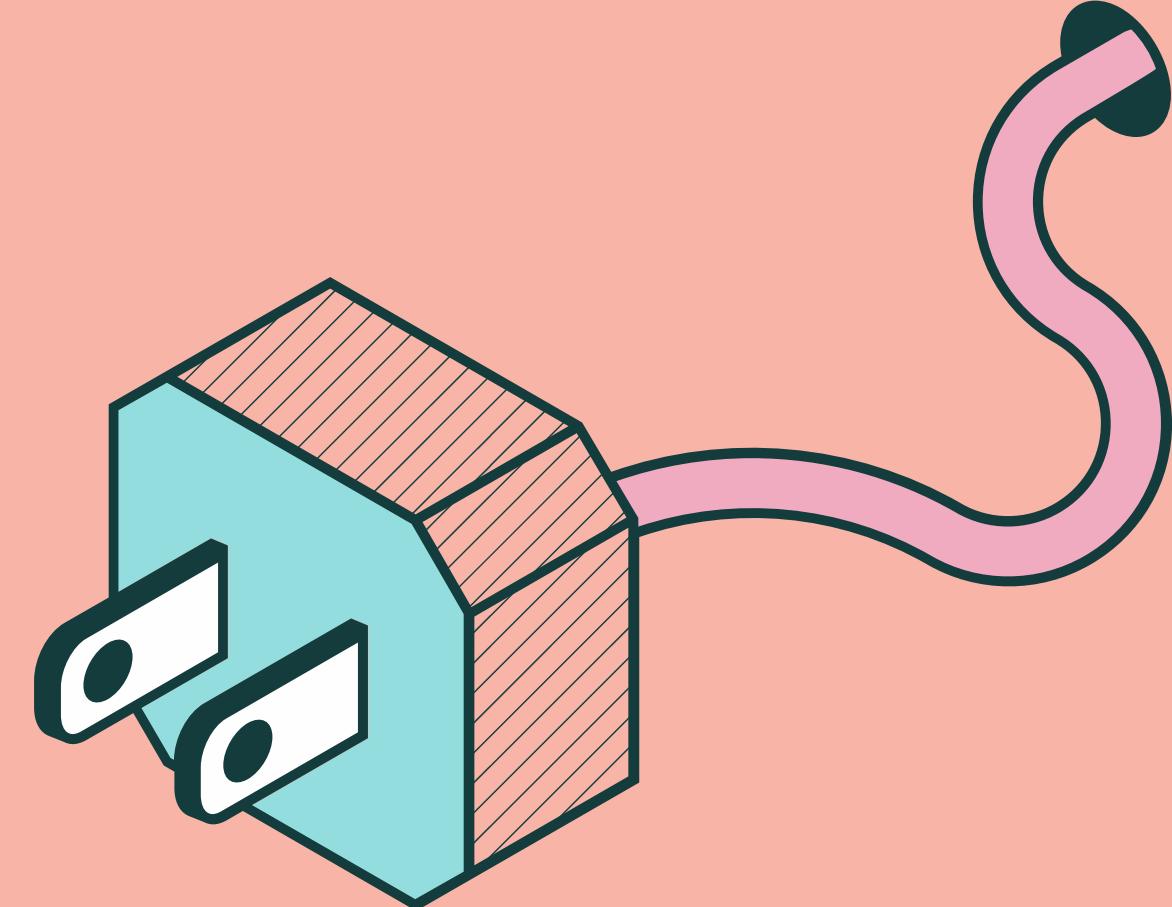
Limitações

Objetos Pequenos

O YOLO tem dificuldade para detectar objetos muito pequenos na imagem, especialmente quando são menores que o tamanho das células do grid. Bandos de pássaros, objetos distantes ou detalhes pequenos podem passar despercebidos.

Objetos Muito Próximos

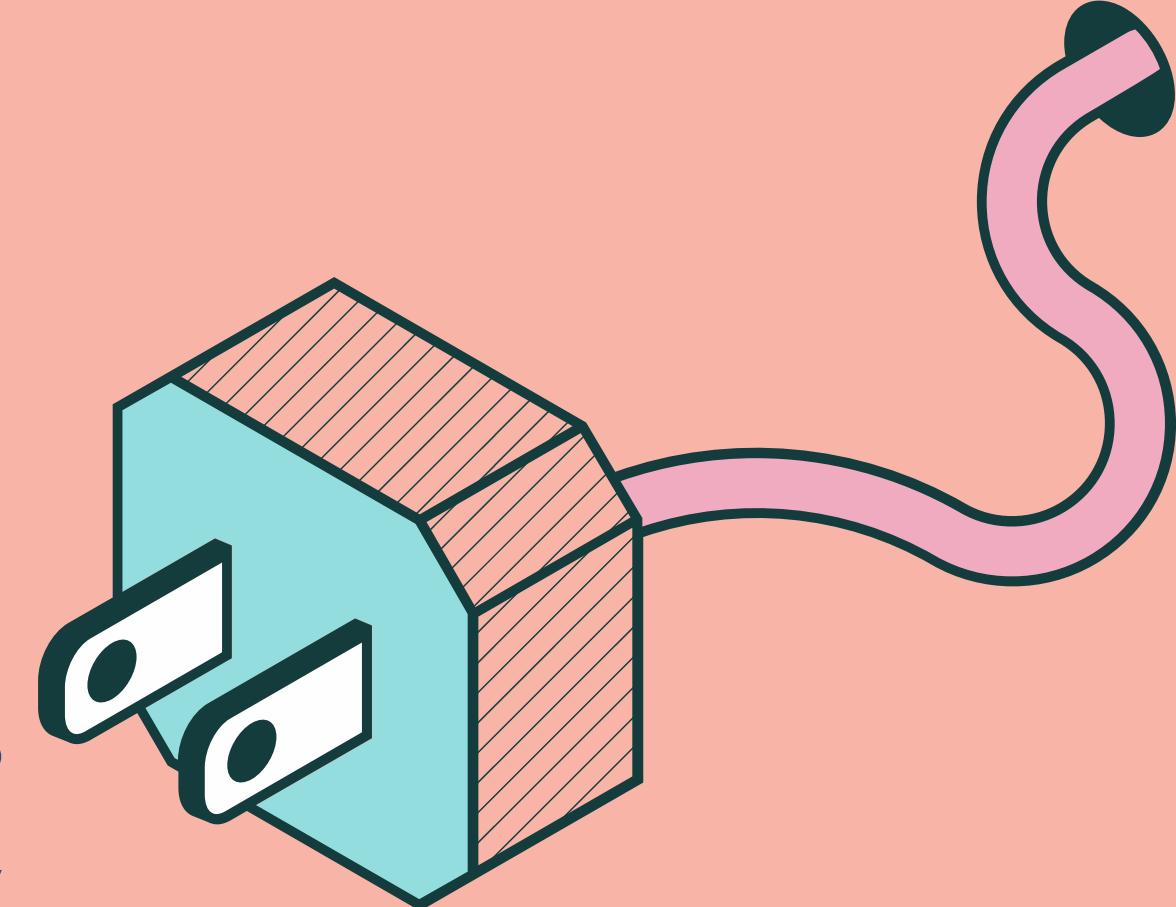
Quando objetos da mesma classe estão muito próximos ou sobrepostos, o YOLO pode ter dificuldade para separá-los, já que cada célula do grid prevê apenas uma classe. Por exemplo, um grupo de pessoas aglomeradas pode ser detectado como uma única pessoa.



Limitações

Formas Incomuns

Objetos com proporções muito diferentes dos padrões de treinamento (aspect ratios incomuns) podem ser mal detectados. Por exemplo, objetos muito alongados ou muito achatados.



Precisão vs Velocidade

Para manter a velocidade, o YOLO às vezes sacrifica precisão. Detectores mais lentos como R-CNN podem ser mais precisos em cenários complexos.

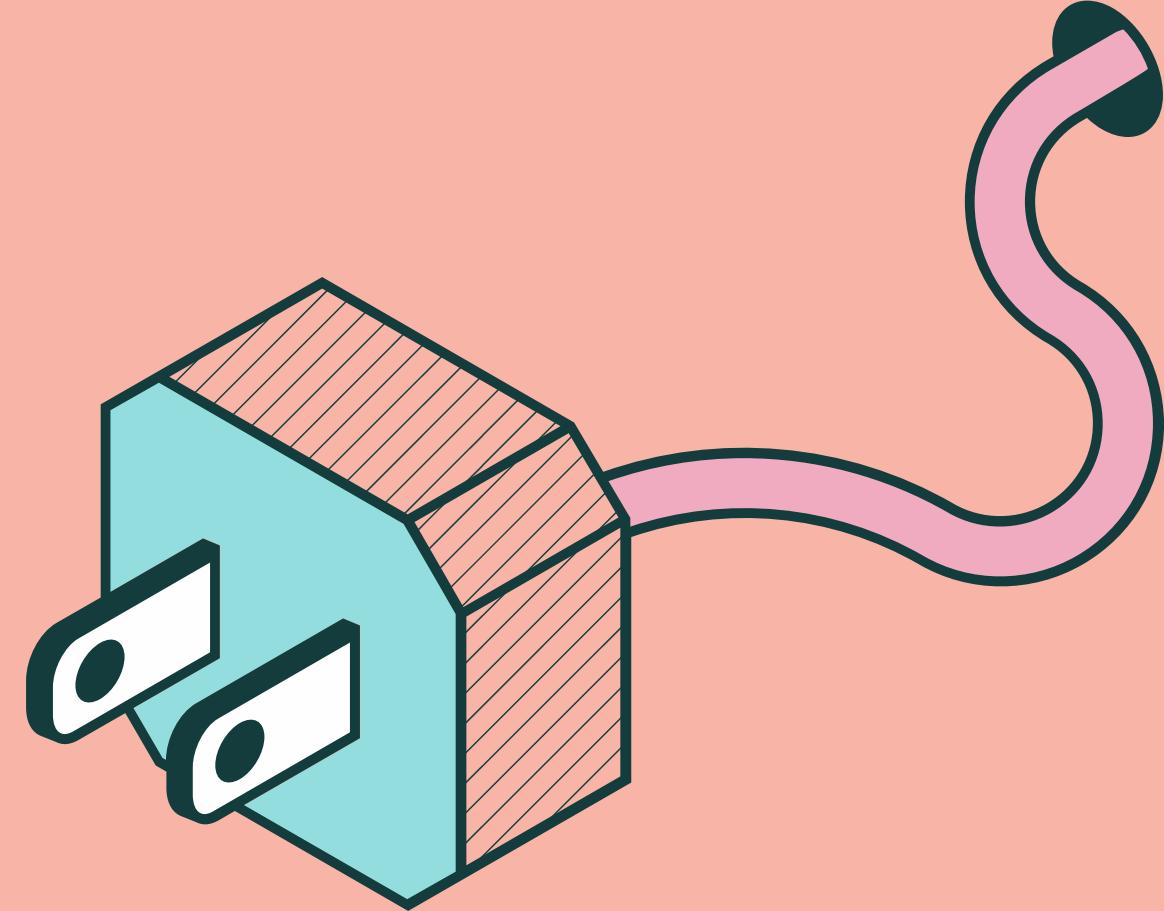
Limitações

Dependência de Dados de Treinamento

Como qualquer rede neural, o YOLO só detecta bem classes que foram vistas durante o treinamento. Objetos muito diferentes do dataset de treino podem não ser reconhecidos.

Background Complexo

Em imagens com fundos muito complexos ou padrões que "confundem" a rede, podem ocorrer falsos positivos.



Comparação com Outros Métodos de Detecção

R-CNN (Region-based CNN)

O R-CNN funciona em duas etapas distintas: primeiro usa selective search para identificar cerca de 2000 regiões candidatas na imagem, depois cada região é processada individualmente por uma CNN para classificação e localização. Esta abordagem de duas etapas é extremamente precisa, especialmente para objetos pequenos e sobrepostos, mas muito lenta.

Características: Alta precisão, melhor localização através de refinamento iterativo, mas processamento de vários segundos por imagem. O Fast R-CNN e Faster R-CNN melhoraram a velocidade mantendo a filosofia de duas etapas.

SSD (Single Shot MultiBox Detector)

O SSD combina velocidade e precisão usando detecção em uma única passada (single shot) com múltiplas escalas de feature maps. Divide a imagem em grids de diferentes resoluções - grids menores para objetos grandes e maiores para objetos pequenos. Baseado em VGG com camadas adicionais para detecção multi-escala.

Características: Velocidade intermediária (20-60 FPS), melhor detecção de objetos pequenos que YOLO original, precisão competitiva, mas maior complexidade arquitetural e consumo de memória que YOLO.

YOLO vs R-CNN vs SSD

Velocidade de Processamento

O YOLO é claramente o vencedor em velocidade, processando imagens em tempo real com 45+ FPS. O SSD oferece velocidade intermediária com 20-60 FPS dependendo da configuração. O R-CNN é o mais lento, processando apenas algumas imagens por segundo, sendo inadequado para aplicações em tempo real.

Esta diferença de velocidade vem da arquitetura fundamental: YOLO processa a imagem inteira uma única vez, SSD processa em múltiplas escalas mas ainda em uma passada, enquanto R-CNN precisa processar milhares de regiões candidatas separadamente.

Precisão e Detecção

Em termos de precisão bruta, especialmente para objetos pequenos e cenários complexos, o R-CNN tradicionalmente leva vantagem devido ao seu processo iterativo de refinamento. O SSD oferece um bom equilíbrio, sendo mais preciso que YOLO para objetos pequenos mas menos que R-CNN.

O YOLO compensa com melhor contexto global - ele "vê" a imagem inteira simultaneamente, resultando em menos falsos positivos de background. Esta visão global ajuda na detecção de objetos em contextos complexos onde métodos baseados em regiões podem falhar.

Onde o YOLO é utilizado?

Veículos Autônomos

Detecção de pedestres, outros veículos e obstáculos.

O YOLO é fundamental para sistemas de direção autônoma, processando imagens das câmeras em tempo real para identificar elementos críticos na estrada. Detecta pedestres atravessando, ciclistas, outros carros, motocicletas, sinais de trânsito e obstáculos como cones ou detritos. A velocidade do YOLO é essencial - precisa processar 30+ quadros por segundo para permitir reações rápidas do sistema de controle do veículo. Também funciona em diferentes condições climáticas e de iluminação.

Segurança e Vigilância

Monitoramento de câmeras para identificar e rastrear indivíduos em tempo real.

Sistemas de segurança usam YOLO para análise automática de feed de câmeras, detectando pessoas em áreas restritas, comportamentos suspeitos ou objetos abandonados. Pode identificar múltiplas pessoas simultaneamente, rastrear seus movimentos e gerar alertas automáticos. Aplicações incluem segurança aeroportuária, monitoramento de perímetros industriais, controle de acesso em edifícios e análise de multidões em eventos. O processamento em tempo real permite resposta imediata a situações de risco.

Onde o YOLO é utilizado?

Medicina

Análise de imagens médicas (raios-x, tomografias) para identificação de anomalias e doenças.

Na área médica, YOLO auxilia radiologistas na detecção de patologias em exames de imagem. Pode identificar nódulos pulmonares em raios-X do tórax, tumores em tomografias, fraturas ósseas e outras anomalias. O sistema acelera o diagnóstico ao destacar áreas suspeitas para análise médica mais detalhada. Também é usado em dermatologia para detecção de lesões de pele, oftalmologia para identificar doenças retinianas e patologia para análise de células cancerígenas em microscopia.

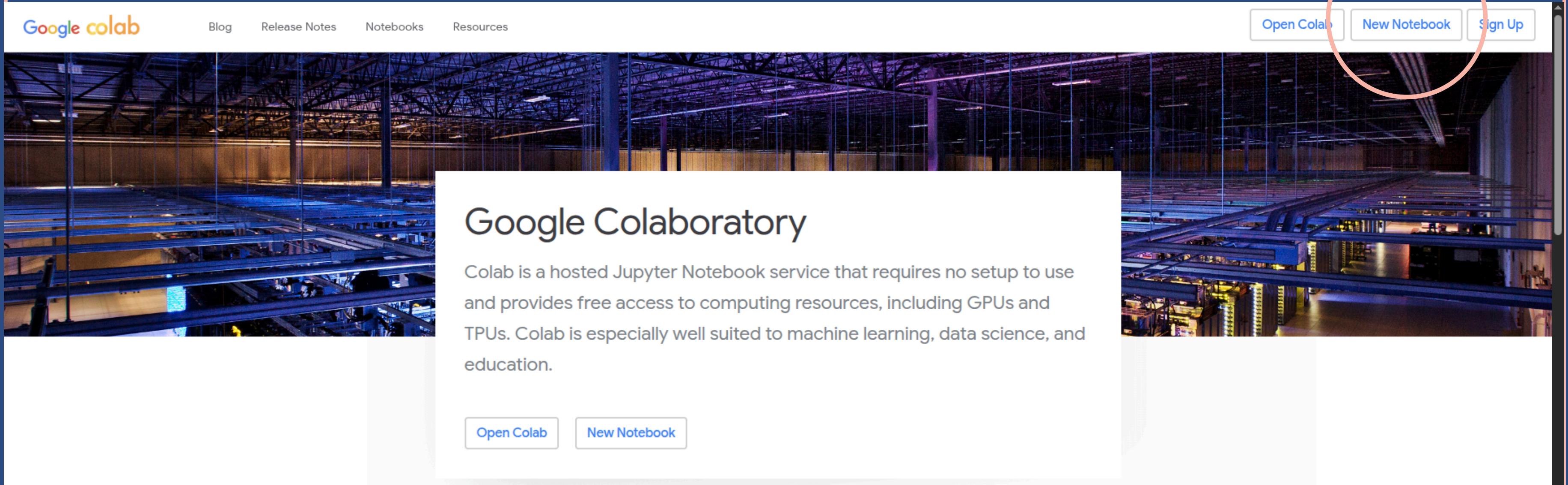
Indústria e Robótica

Sistema de controle de qualidade, para a detecção automática de defeitos.

YOLO revoluciona o controle de qualidade industrial ao detectar defeitos em produtos na linha de produção com velocidade e precisão. Identifica rachaduras, riscos, componentes faltantes, soldas defeituosas e variações dimensionais em peças manufaturadas. Em robótica, permite que robôs "vejam" e manipulem objetos específicos, realizem montagem automatizada e naveguem em ambientes complexos. Aplicações incluem inspeção de PCBs eletrônicos, verificação de embalagens, classificação automática de produtos e robôs de picking em armazéns.

HORA DE PRATICAR!

GOOGLE COLAB



The screenshot shows the Google Colab homepage. At the top left is the "Google colab" logo. To its right are four navigation links: "Blog", "Release Notes", "Notebooks", and "Resources". On the far right of the header are three buttons: "Open Colab", "New Notebook", and "Sign Up". A large, faint image of a multi-story industrial or data center building with complex steel structures and lighting serves as the background. In the center foreground, there's a white rectangular area containing the title "Google Colaboratory" and a descriptive paragraph about Colab's features. Below this text are two buttons: "Open Colab" and "New Notebook". The entire page has a dark blue header and footer.

Google colab

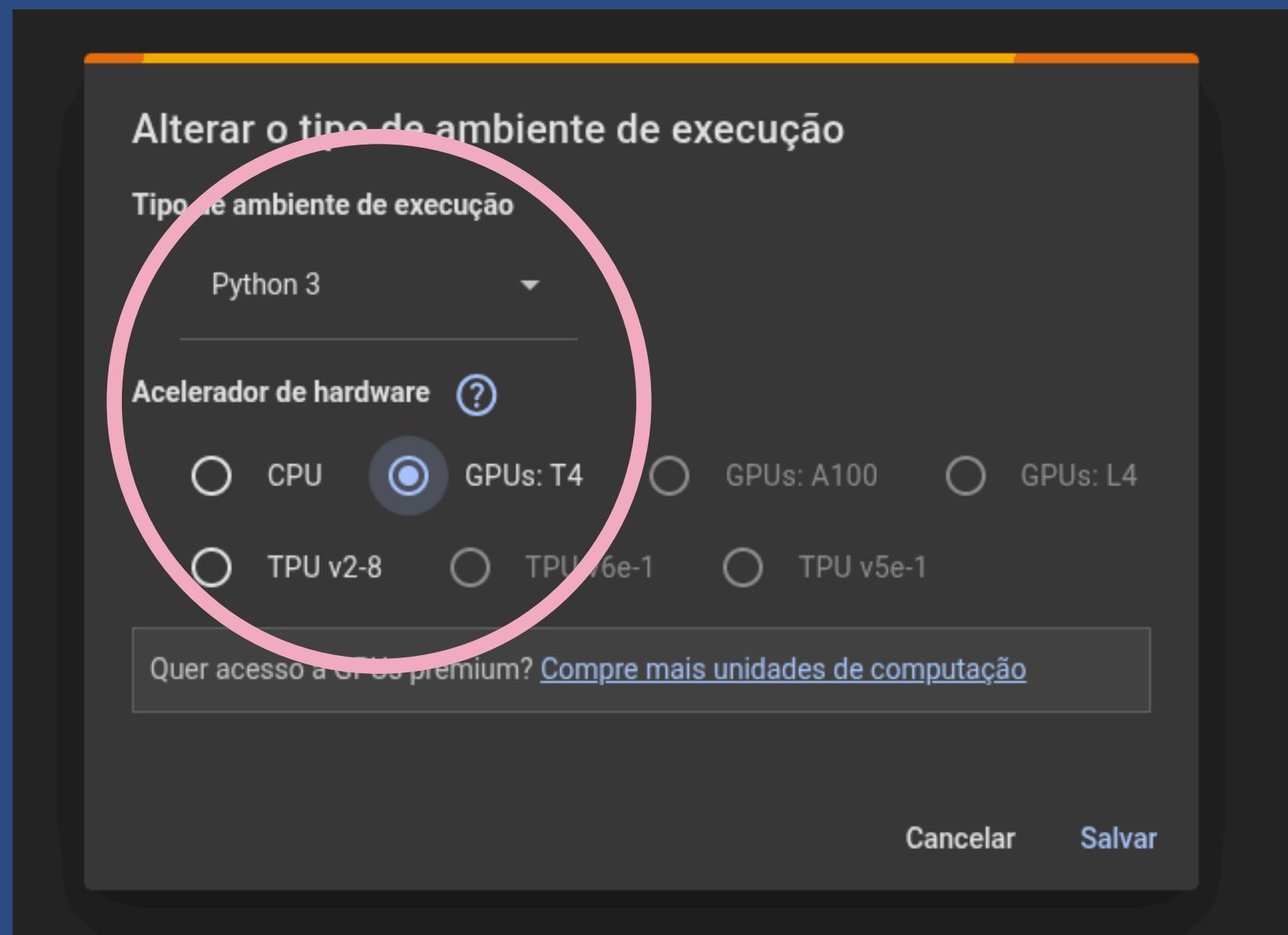
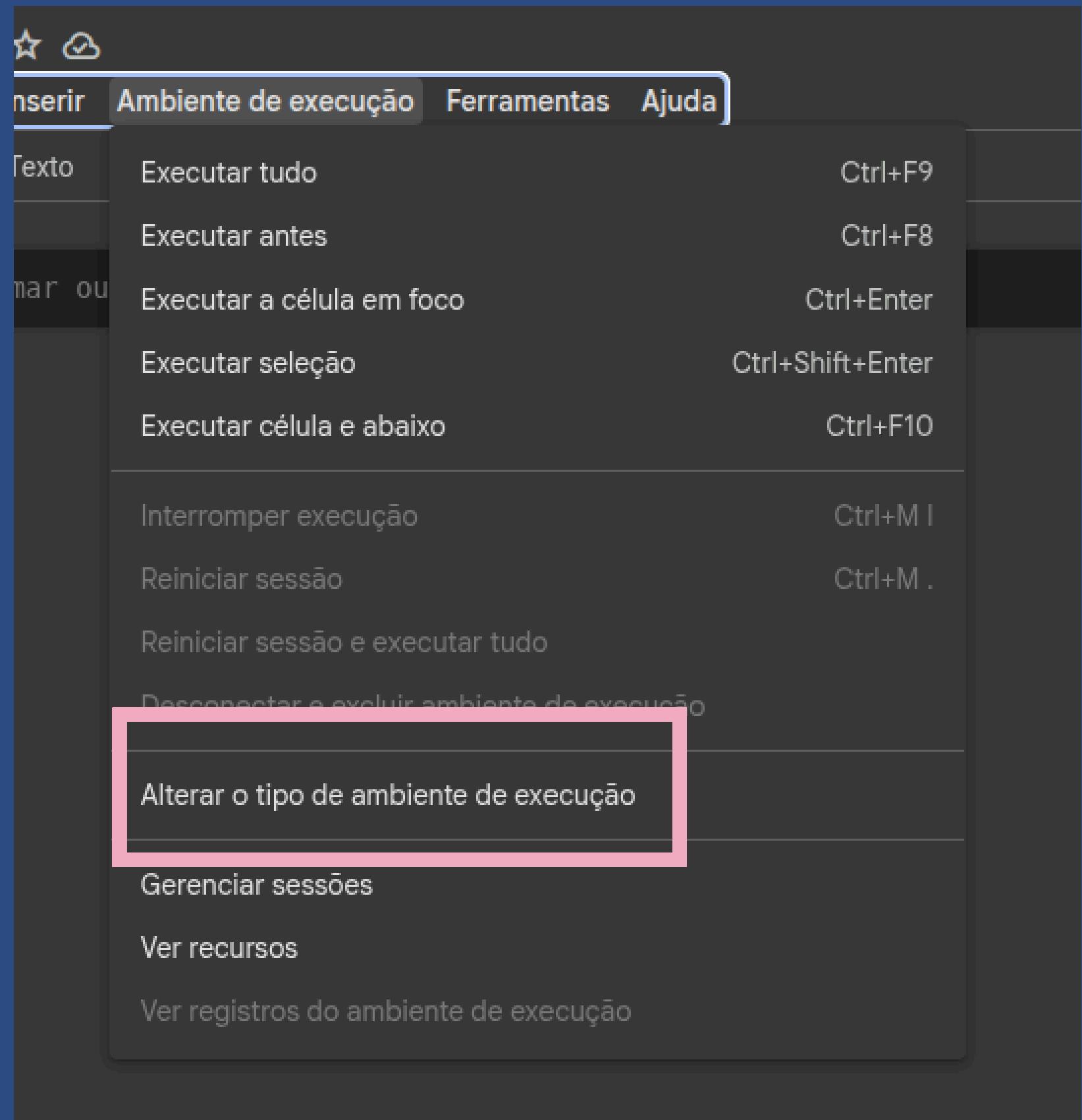
Blog Release Notes Notebooks Resources

Open Colab New Notebook Sign Up

Google Colaboratory

Colab is a hosted Jupyter Notebook service that requires no setup to use and provides free access to computing resources, including GPUs and TPUs. Colab is especially well suited to machine learning, data science, and education.

Open Colab New Notebook





Maria Luiza Prata

Malu-Prata

Graduanda em Ciência da Computação
pela UFPel.

Edit profile

7 followers · 7 following

Brasil

marialuizabprata@hotmail.com

in/maria-luiza-prata-a342812b3

Malu-Prata / README.md



Ooi! Eu sou a Malu!

- 🔭 Estudante de ciência da computação na UFPel, atualmente trabalho na PROGEP-UFPel e estou aprendendo Java, PostgreSQL e C!
- 🌟 Faço parte do projeto Gurias da Comp, que tem como objetivo apoiar e fortalecer a permanência feminina nos cursos de computação!

☆ ESTRELAS 0 ⚡ SEGUIDORES 5

🌐 Redes Sociais

[LINKEDIN](#) [INSTAGRAM](#)

🤖 Linguagens e Tecnologias



📊 Estatísticas

Estatísticas do GitHub de Maria Luiza Prata

☆ Total de estrelas:

0

⌚ Total de Commits:

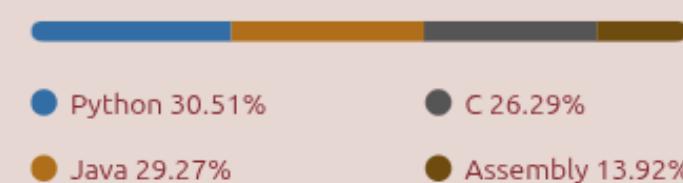
76

↗️ Total de PRs:

0



Tecnologias



Type ▾Language ▾Sort ▾New

Minicurso_YOLO

Public

Comandos usados durante a apresentação do minicurso "Detecção de imagens com Redes Neurais e YOLO", apresentado durante a SACOMP da UFPel.



Star



Updated 13 minutes ago

MÃONA MASSA!!!

OBRIGADA PELA
ATENÇÃO

