

# Dem4AI

Redes Neuronais

**Autores:**

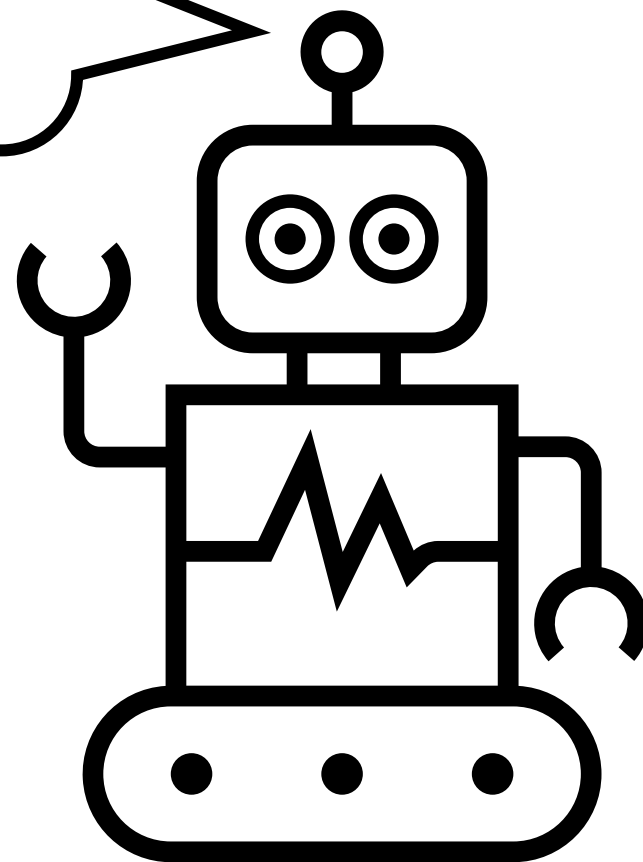
Francisco António Lisboa Guarda N°2201675

Samuel Domingos Lourenço N°2200904

**Orientadores:**

Luís Manuel Conde Bento

Mónica Jorge Carvalho de Figueiredo



1. Redes Neurais Profundas (Deep Neural Networks - DNNs)
  - O que são?
  - Exemplo
  - Estrutura
  - Tipos
  
2. Tipos de Redes Neurais
  1. Redes Feedforward
  2. Redes Recorrentes
  3. Redes Convolucionais
  4. Redes Autoencoder

# Capítulo 1.

---

## Redes Neurais Profundas

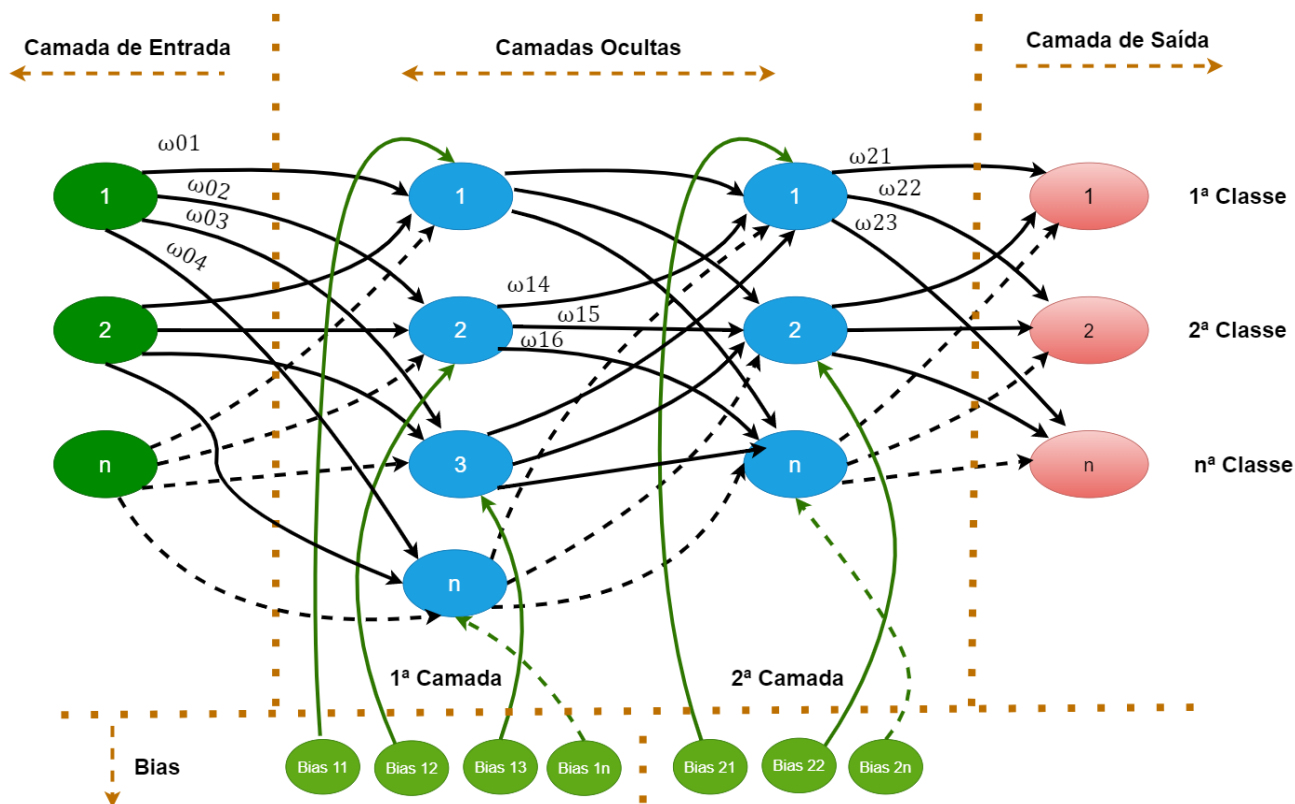
Devido à sua grande variedade de utilidade e prestabilidade, as redes neuronais começam a ser cada vez mais faladas e a ter cada vez mais impacto em tudo à nossa volta. Estes são alguns dos exemplos de utilizações para as redes neuronais atualmente:

- Diagnóstico médico por classificação de imagens;
- Controlo de qualidade;
- Reconhecimento visual em carros autónomos (Ex.: Teslas);
- Reconhecimento facial para identificar rostos (Ex.: Desbloqueio por reconhecimento facial nos smartphones);
- Chatbots automatizados (Ex.: ChatGPT);
- Organização automática e classificação de dados escritos.



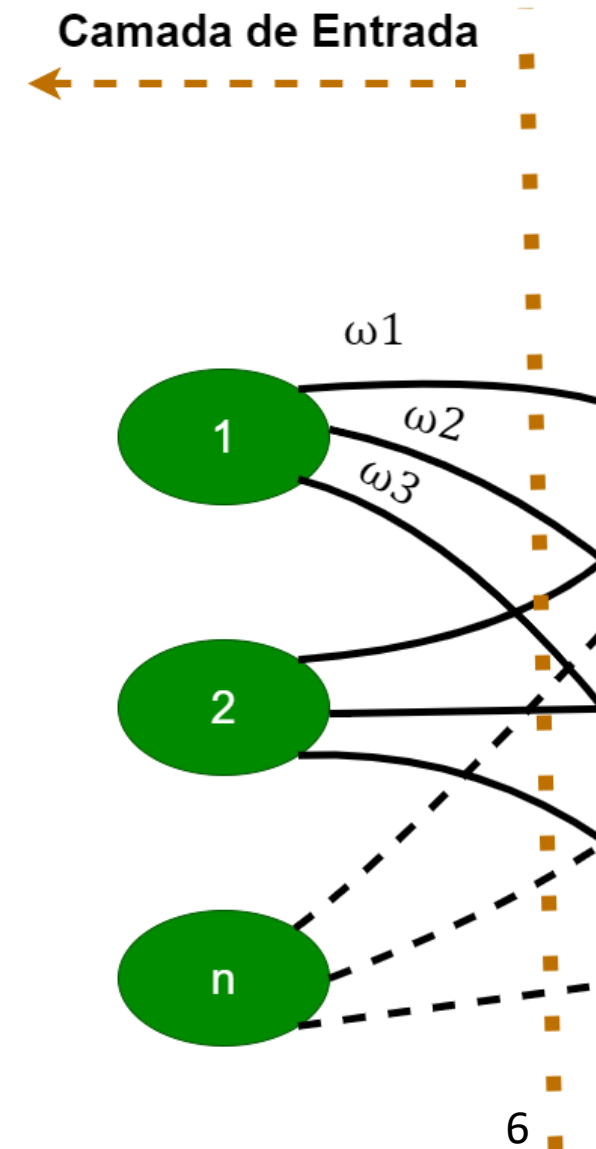
## Introdução às Redes Neuronais Profundas

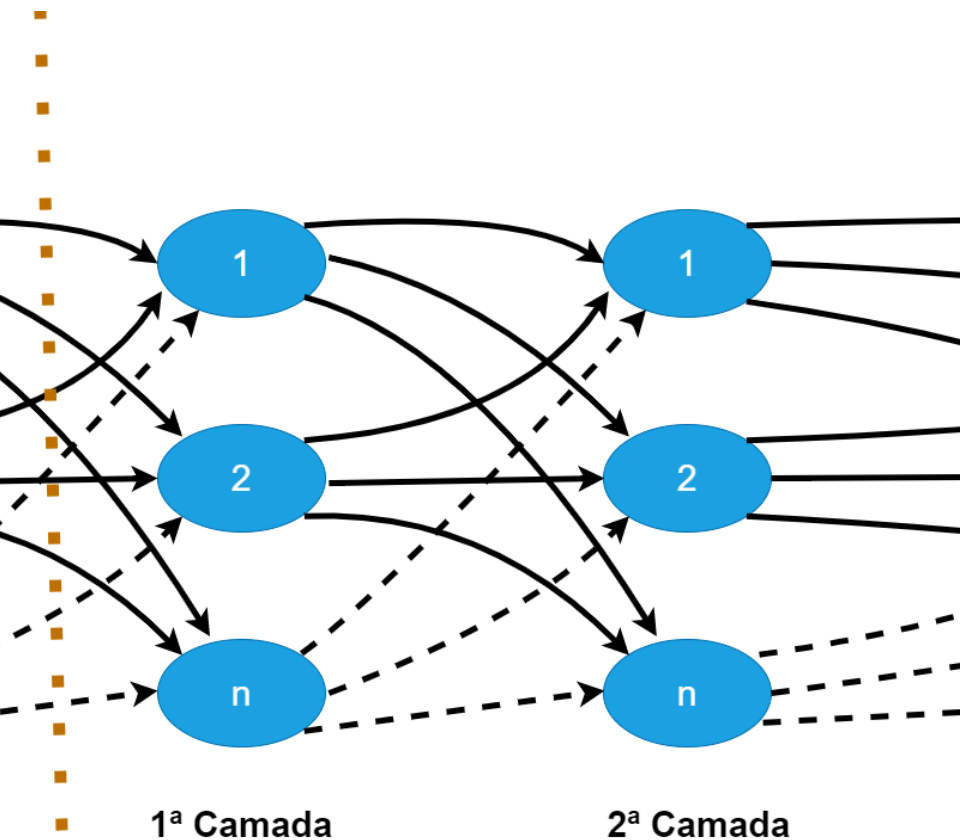
As redes neuronais artificiais são compostas por várias camadas de perceptrões, contendo, assim como estes, uma camada de entrada e uma camada de saída, mas com outras camadas entre estas duas, chamadas camadas ocultas.



Tal como num neurónio artificial simples (um perceptrão), as redes neuronais podem ter uma ou várias entradas. O número delas vai depender das características que temos no nosso conjunto de dados.

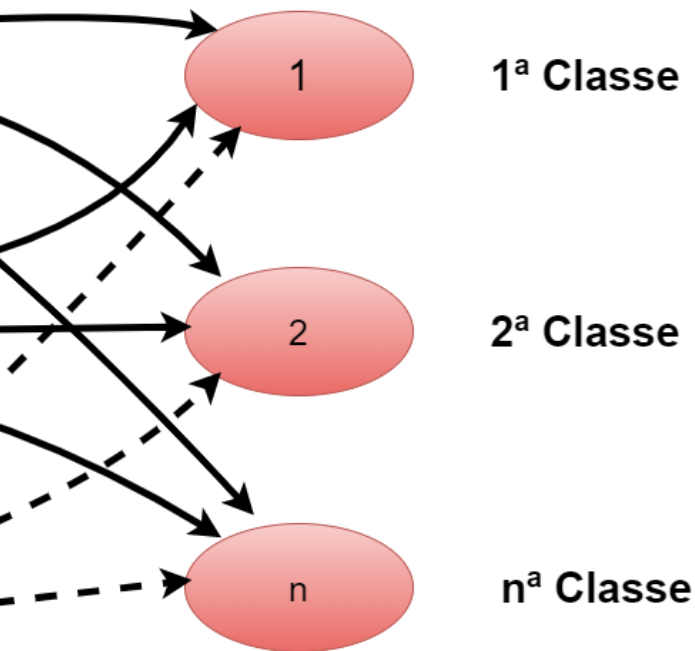
As entradas vão ligar à primeira camada oculta como no neurónio artificial que vimos no dispositivo anterior.





A camada oculta é constituída por neurónios “intermédios”. Numa rede neuronal podem existir uma ou "n" camadas.

Neste exemplo, a rede possui duas camadas a 1ª camada e a 2ª camada, mas poderá ter mais.



Na camada de saída podemos ter "n" saídas, a que se dão o nome de classes.

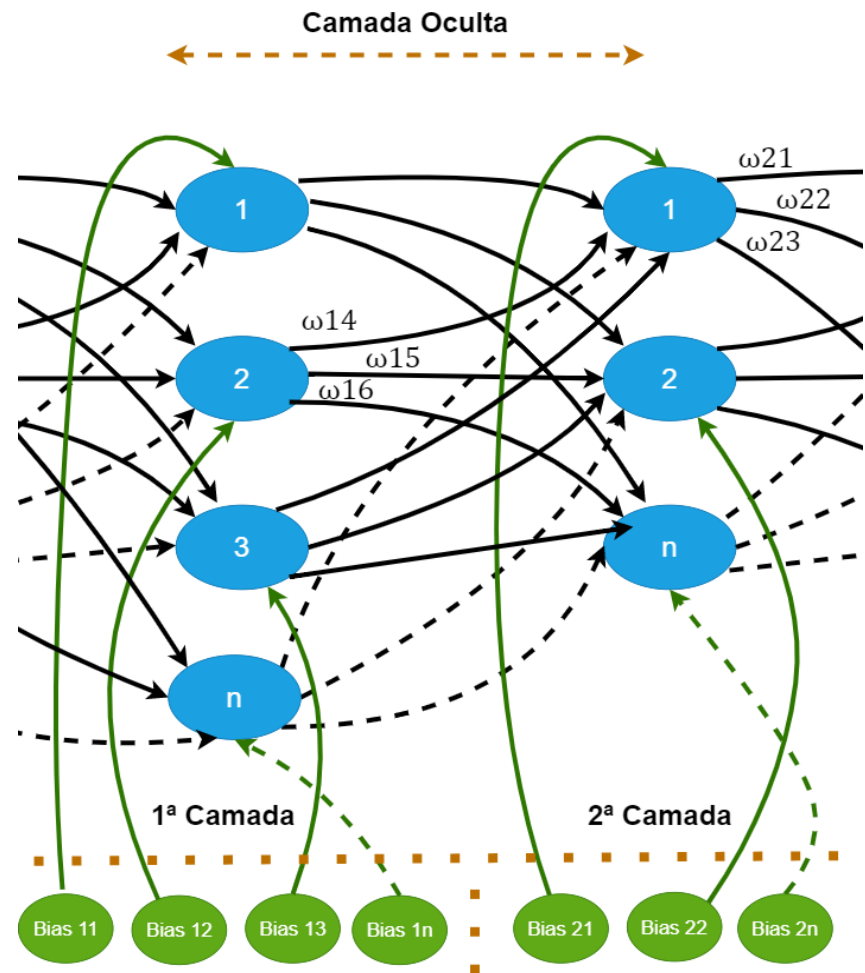
O número de classes vai depender do número de opções possíveis.

Usando o exemplo anterior da maçã e da banana falado na introdução à inteligência artificial: Se fosse adicionada uma laranja ao exemplo, a rede neuronal iria ter 3 classes, uma para cada fruta.



Tal como no perceptrão, nas redes neuronais é aplicado um bias, neste caso, a cada perceptrão das camadas ocultas.

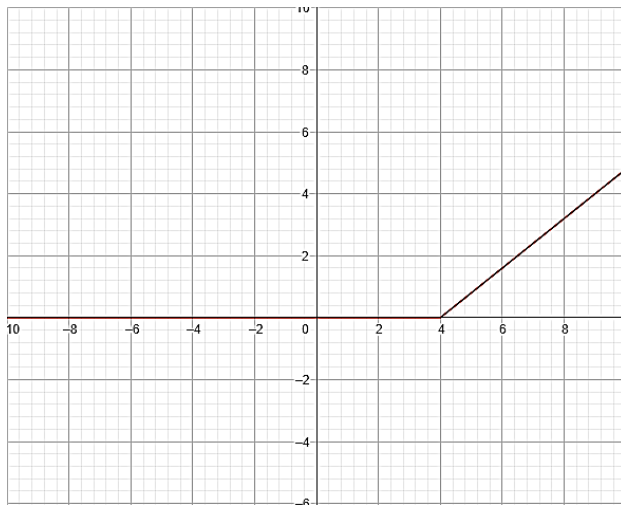
Cada bias está ligado a um perceptrão que permite ajustar o limiar da função de ativação.



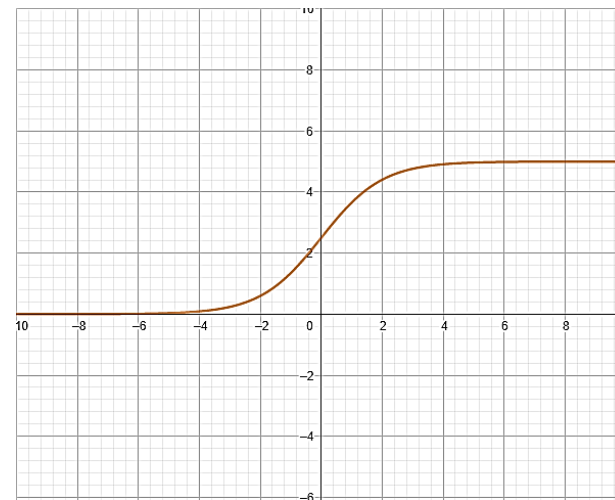
Todos os perceptrões da camada oculta têm uma função de ativação associada.

Como apresentado anteriormente, existem múltiplos tipos de funções, nomeadamente as de degrau unitário, as lineares e as não lineares.

Dentro de todas estas, duas das mais usadas são:



*ReLú*



*Sigmoid*

# O que são Híper Parâmetros?

Os híper parâmetros são parâmetros de configuração em Redes Neurais que são definidos antes do treino de um modelo e, tipicamente, não podem ser aprendidos com os dados durante o processo, existindo certas exceções.

Estes incluem: o número de camadas ocultas (por exemplo, 3,5,...), o número de neurónios em cada camada (por exemplo, 512, 1024, 256,...), a taxa de aprendizagem (por exemplo, 0.1, 0.01, ...), e o número de épocas para treinar o modelo.



**Hyperparameters**



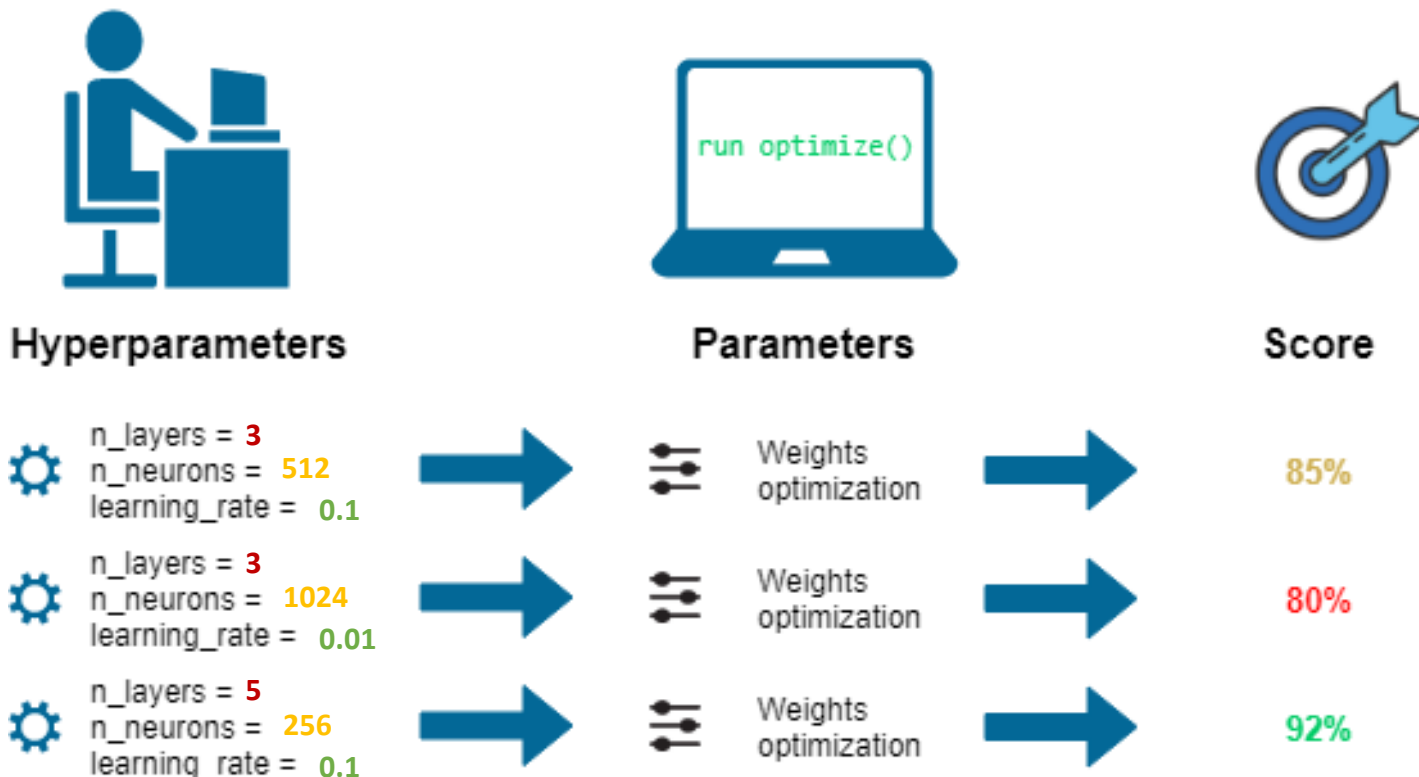
**Parameters**



**Score**



Os valores dos híper parâmetros têm um impacto significativo no desempenho do modelo, e encontrar os valores ideais geralmente é um processo de tentativa e erro que requer experimentação.



## O que são Datasets?

Datasets são conjuntos de dados que são usados para treinar as Redes Neurais. Usando o exemplo passado, da classificação de maçãs e bananas, a tabela representa os dados do dataset que usamos para treinar a rede neuronal.

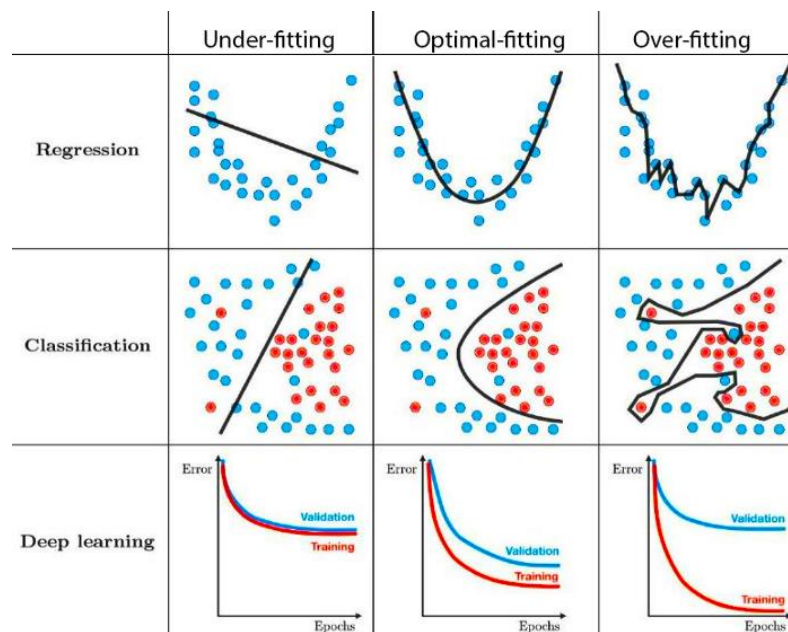
Usando os dados que indicam qual a fruta e qual a sua massa ou forma, podemos treinar a Rede Neuronal de modo a que, quando adicionados novos dados, a rede seja capaz de classificar corretamente as frutas.

Fruta	Forma	Massa de 0 a 1
Maçã	0.22	0.93
Banana	0.7	0.74
Maçã	0.2	0.91
Banana	0.9	0.63
Banana	0.84	0.71
Maçã	0.38	0.73
Banana	0.68	0.53
Banana	0.71	0.62
Maçã	0.29	0.81
Maçã	0.85	0.83

## Problema de Overfitting

Ao utilizarmos grandes redes neuronais com datasets pequenos, podemos encontrar problemas de sobre-ajuste (“overfitting”) dos pesos ou bias.

Simplificando, a rede neuronal irá ficar demasiada adaptada à base de dados utilizada inicialmente, o que irá prejudicar a rede durante a classificação quando esta receber novos dados de entrada.



O problema de não utilizar diferentes conjuntos de dados (Datasets) é que a rede irá utilizar todas as amostras para aprender.

Esta irá funcionar bem apenas para amostras relativamente parecidas com os dados usados. Caso se coloque uma maçã com parâmetros ligeiramente diferentes da gama utilizada no treino, ela poderá ser mal classificada (“overfitting”).

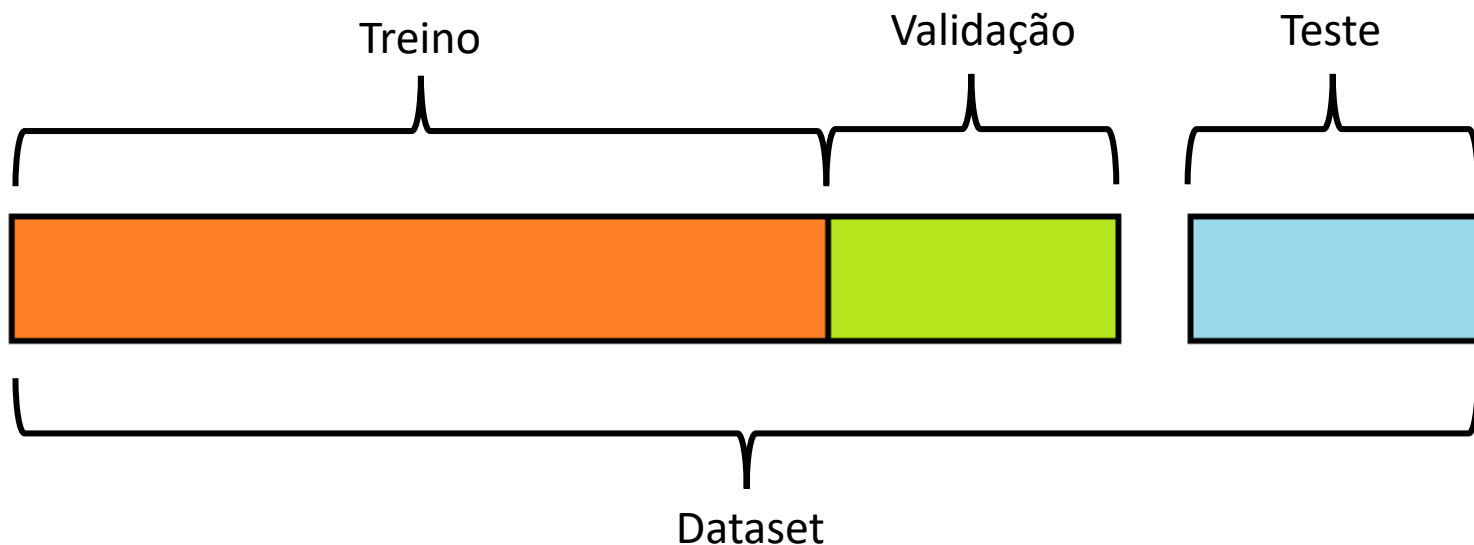
**Treino**

Fruta	Forma	Massa de 0 a 1	Amostra
Maçã	0.22	0.93	1
Banana	0.7	0.74	2
Maçã	0.2	0.91	3
Banana	0.9	0.63	4
Banana	0.84	0.71	5
Maçã	0.38	0.73	6
Banana	0.68	0.53	7
Banana	0.71	0.62	8
Maçã	0.29	0.81	9
Maçã	0.85	0.83	10

De forma a combater o sobre-ajuste (overfitting) dos pesos e bias, é habitual fazer-se a divisão do dataset pretendido em 3 sub-datasets:

- **Treino**
- **Validação**
- **Teste**

Desta maneira, como usamos valores diferentes para treinar, testar e validar os pesos e os bias, a rede neuronal não irá ficar ajustada apenas aos valores de um dataset, sendo desta maneira mais fácil evitar o overfitting, permitindo que a rede neuronal esteja mais apta para receber dados novos.





Utilizando o exemplo das maçãs e bananas, verificamos que temos poucos dados. Como tal, de forma a combater o overfitting, podemos dividir o dataset da seguinte forma:

Com esta divisão, a Rede Neuronal não ficará tão adaptada ao pequeno conjunto de dados do dataset usado, o que permitirá que, ao receber dados novos, esta esteja melhor preparada para classificar as novas frutas.

	Fruta	Forma	Massa de 0 a 1	Amostra
<b>Treino</b>	Maçã	0.22	0.93	1
	Banana	0.7	0.74	2
	...	...	...	...
	Banana	0.9	0.63	1010
	Banana	0.84	0.71	1011
	Maçã	0.38	0.73	1012
<b>Validação</b>	Banana	0.68	0.53	1013
	...	...	...	...
	Maçã	0.29	0.81	1658
	Maçã	0.85	0.83	1659
<b>Teste</b>	Maçã	0.68	0.53	1660
	...	...	...	...
	Banana	0.29	0.81	1699
	Maçã	0.85	0.83	1700

Nota: Quanto mais amostras tiver melhor será para a rede classificar o tipo de fruta.

O sub-dataset de treino, é o que usado para treinar o modelo (pesos e bias de uma rede neuronal). O modelo analisa e aprende com os dados deste.

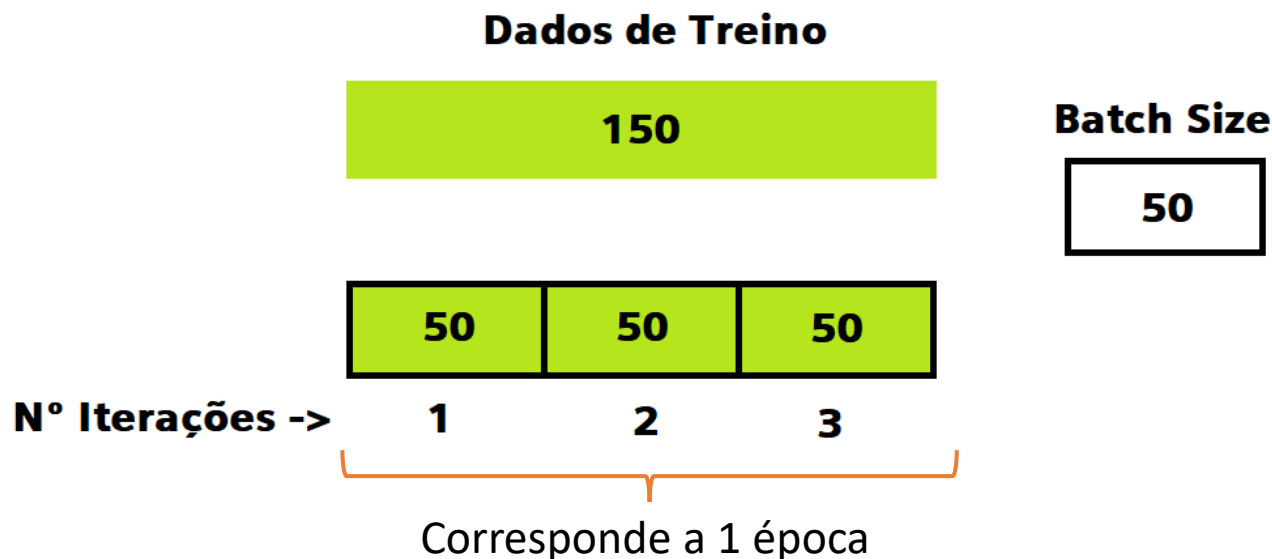
Os dados deste sub-dataset não são usados nos outros sub-datasets (validação e testes) de maneira a podermos testar a rede com dados que esta ainda não conhece, permitindo que aprenda os valores de pesos e Bias da melhor forma para qualquer que seja caso apresentado.

**Treino**

Fruta	Forma	Massa de 0 a 1	Amostras
Maçã	0.22	0.93	1
Banana	0.7	0.74	2
...	...	...	...
Banana	0.9	0.63	1010
Banana	0.84	0.71	1011
Maçã	0.38	0.73	1012

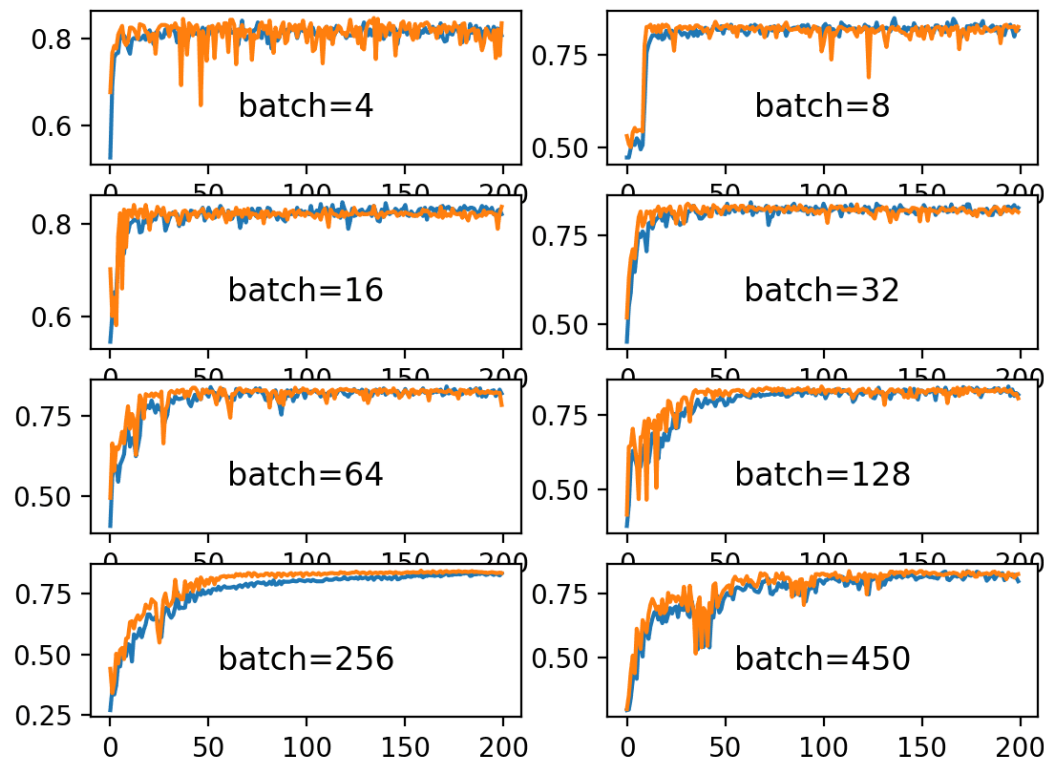
O *batch size* de uma rede neuronal é o nome dado ao número de dados de treino que são processados simultaneamente pela rede antes que os pesos sejam atualizados e a rede avance para a próxima iteração do algoritmo de treino.

Exemplo: Temos 150 dados de treino. Se o nosso *batch size* (tamanho do lote) for de 50 são necessárias 3 iterações para fazer uma etapa.



Ao utilizar um tamanho pequeno de lote (batch) o resultado de aprendizagem será rápido, mas instável, com alta variação na precisão da classificação.

Se usarmos tamanhos maiores de lote, desacelera a aprendizagem, mas a rede será mais estável, com menor variação na precisão da classificação.



O sub-dataset de validação é usado para avaliar a rede neuronal. Os dados deste são usados apenas de forma a ajustar os hiper parâmetros (taxa de aprendizagem, número de épocas, etc) da rede.

A rede analisa ocasionalmente os dados, mas nunca "aprende" com eles. Ou seja, o sub-dataset de validação afeta a rede neuronal, mas apenas de forma indireta.

**Validação**

Fruta	Forma	Massa de 0 a 1	Amostras
Banana	0.68	0.53	1013
...	...	...	...
Maçã	0.29	0.81	1658
Maçã	0.85	0.83	1659

O sub-dataset de teste vai ser o verdadeiro teste à rede neuronal. É através deste que se avaliam e comparam diferentes modelos de redes neuronais. É usado apenas quando um modelo já foi treinado completamente (usando os sub-datasets de treino e validação).

O dataset de teste geralmente é “bem constituído”. Este contém dados cuidadosamente escolhidos que abrangem as várias hipóteses que a rede neuronal enfrentaria, quando usado no mundo real.

**Teste**

Fruta	Forma	Massa de 0 a 1	Amostras
Maçã	0.68	0.53	1660
...	...	...	...
Banana	0.29	0.81	1286
Maçã	0.85	0.83	1287

Existem ainda outras alternativas para combater o overfitting. O processo de Data Augmentation aborda o overfitting na raiz do problema, o conjunto de dados de treino. Este processo é feito, partindo da suposição de que mais informações podem ser extraídas do conjunto de dados original por meio de aumento dos dados disponíveis, designado por *aumentação*, ou em inglês, “*augmentation*”.



Esta *aumentação*, incrementa artificialmente o tamanho do conjunto de dados de treino por distorção de dados ou sobre amostragem.

Estas “distorções” transformam os dados existentes mas de maneira a que a sua classificação seja preservada.

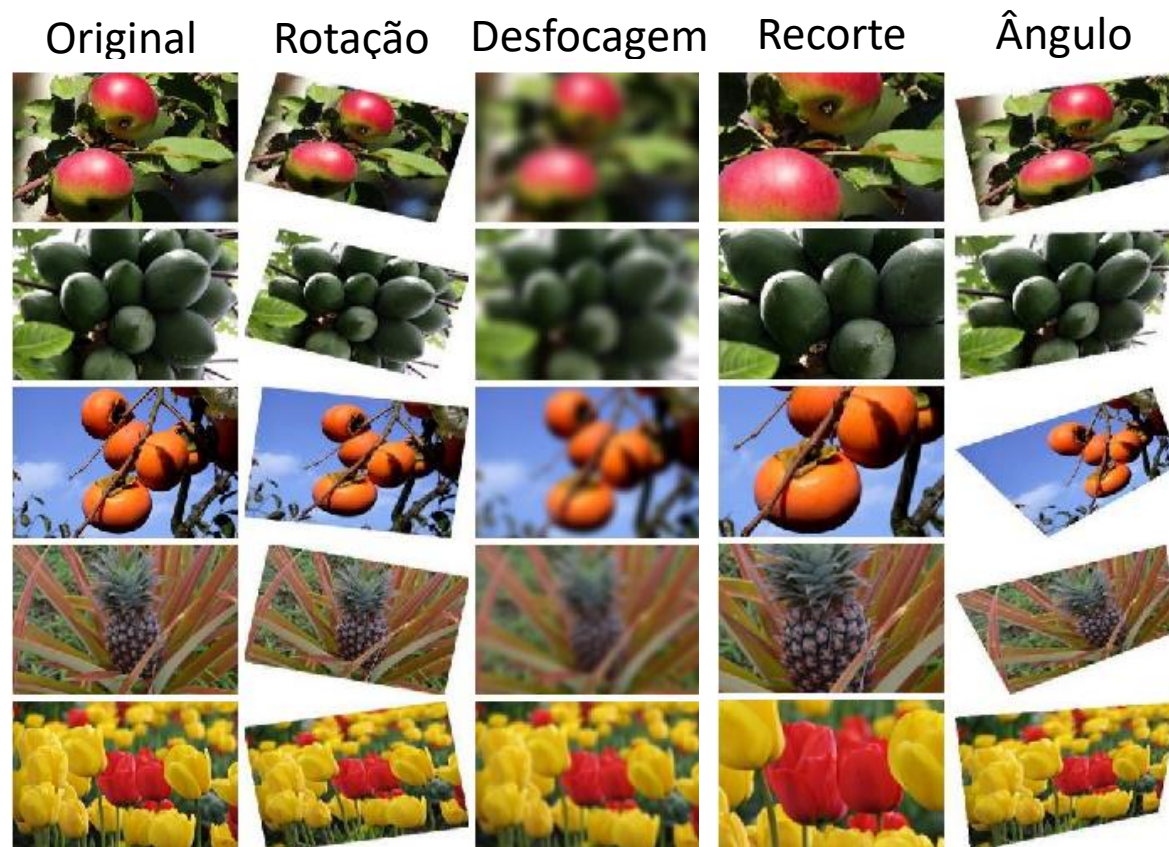
Por exemplo, usando transformações de cores, como podemos ver na seguinte imagem:



É alterado o brilho, o contraste, a saturação e o tom mas de maneira a que se consiga continuar a identificar o Tigre durante a classificação.



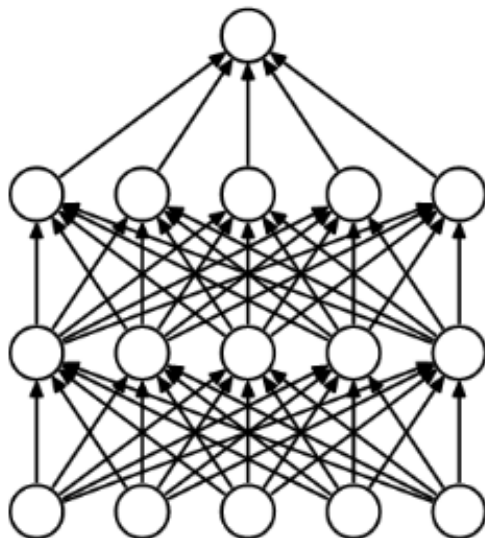
Para além das distorções apresentadas anteriormente, existem outros tipos de data augmentation que se podem utilizar, como podemos observar nas seguintes imagens:



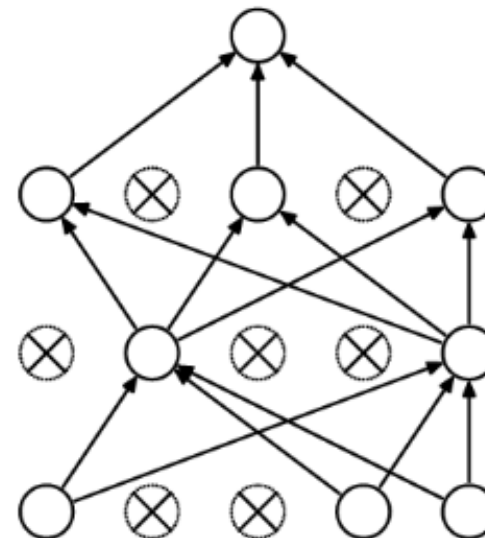
## O que é o Dropout?

Outro método que permite combater o overfitting é o Dropout. O conceito deste processo é o seguinte:

Durante a aprendizagem, algumas saídas das camadas são ignoradas aleatoriamente ou retiradas (“dropped out”). Isso tem o efeito de fazer com que a camada pareça e seja tratada como uma camada com um número diferente de nós e conectividade com a camada anterior.



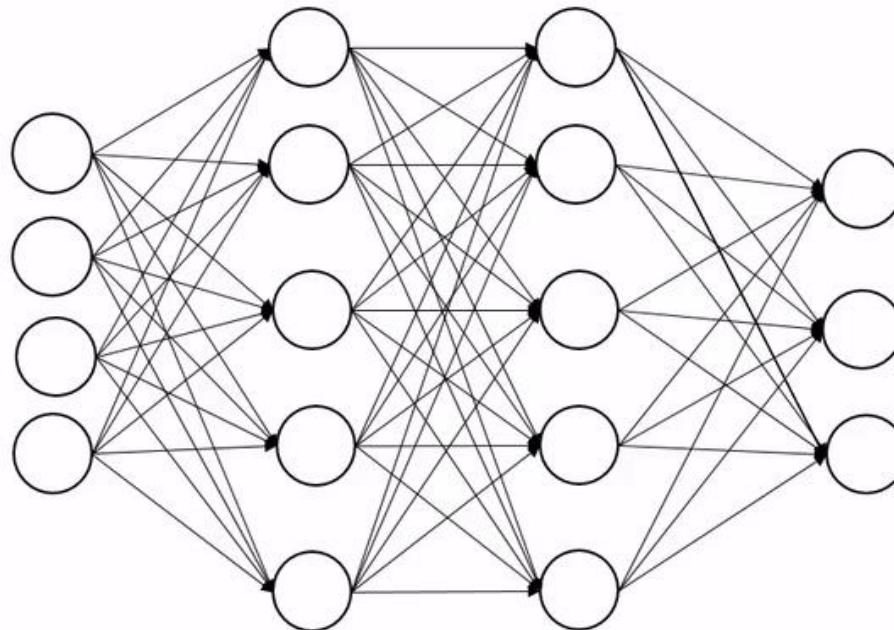
*Rede Neuronal “Normal”*



*Rede Neuronal com Dropout*

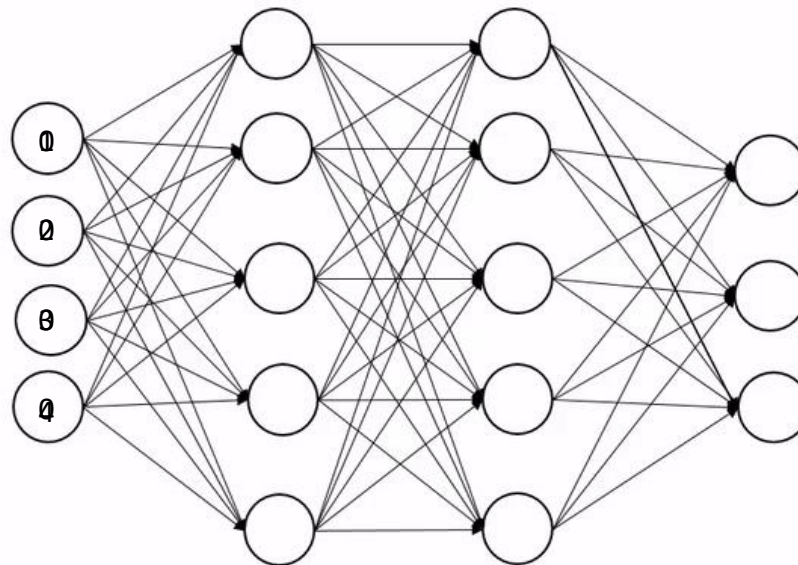
Cada atualização de uma camada durante a aprendizagem é realizada com uma “visualização” diferente da camada configurada.

Esta regularização vai ajudar ao dimensionamento correto dos pesos e bias, no entanto, pode resultar numa aprendizagem mais demorada.

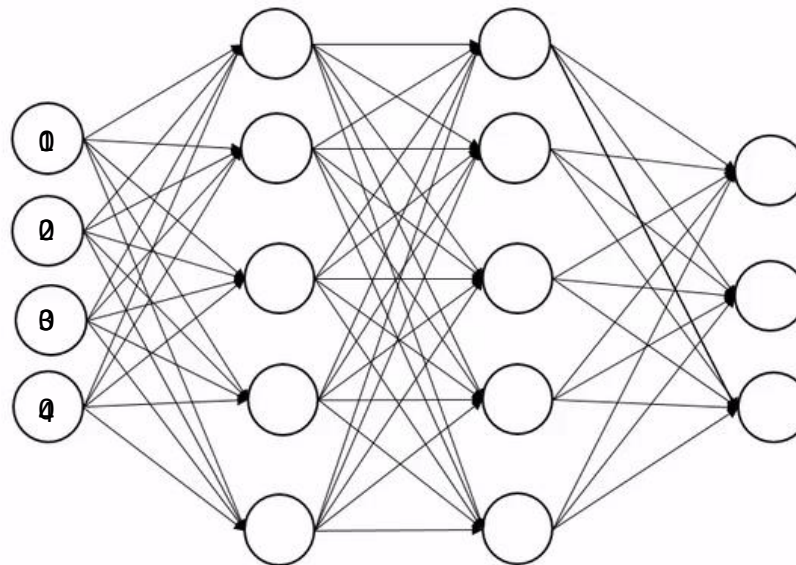


Vamos tentar entender com uma dada entrada  $x$ :  $\{1, 2, 3, 4\}$ . Temos uma camada de dropout com probabilidade  $p = 0,25$ . Durante a propagação direta (aprendizagem) da entrada  $x$ , 25% dos valores seriam descartados, ou seja, o  $x$  poderia tornar-se  $\{1, 0, 3, 4\}$  ou  $\{1, 2, 0, 4\}$ , e por aí adiante.

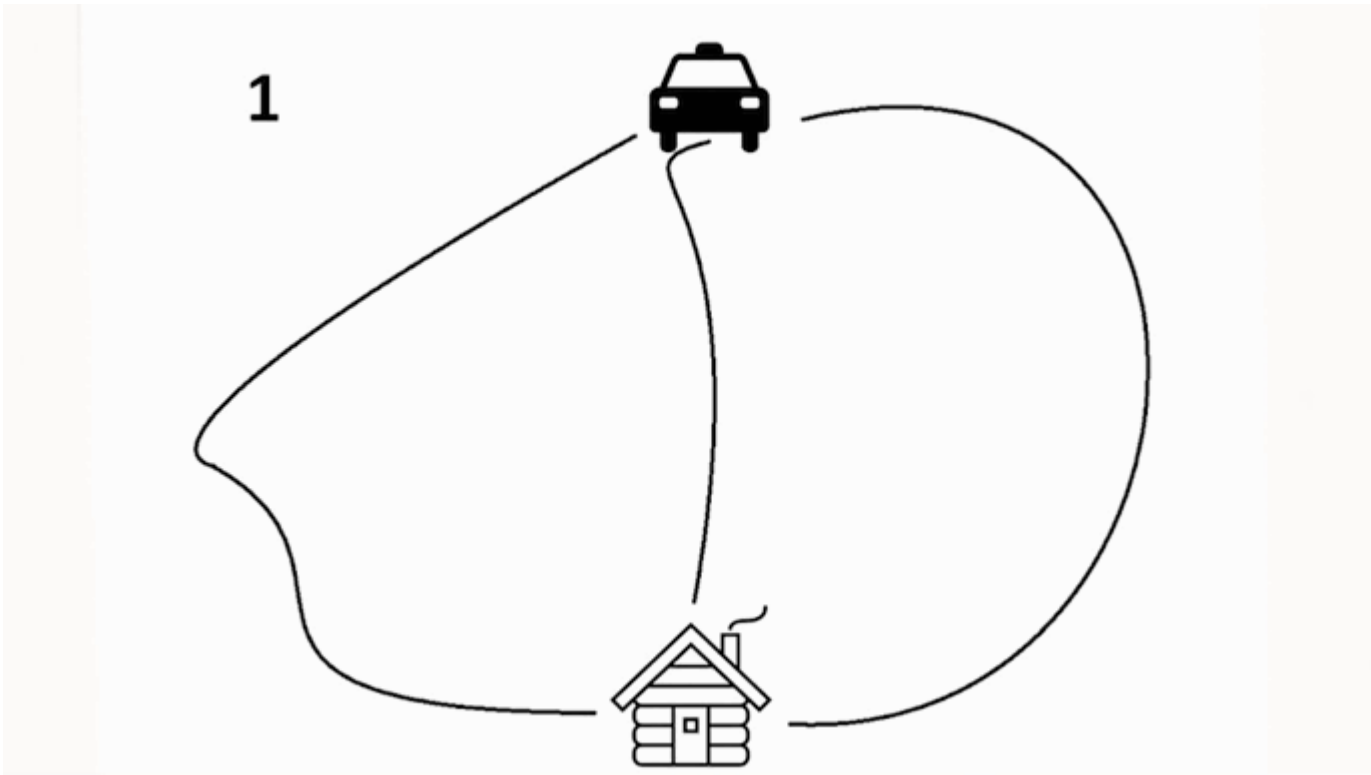
Por exemplo, se as camadas ocultas tiverem 1000 perceptrões e um dropout for aplicado com probabilidade = 0,5 (50%), então 500 perceptrões seriam descartados aleatoriamente em cada época.



Geralmente, para as camadas de entrada, a probabilidade de dropout é mais próxima de 0, sendo 0,2 o recomendado.. Para as camadas ocultas, quanto maior a probabilidade de queda, mais disperso é o modelo, onde 0,5 é a probabilidade de manutenção mais otimizada, que indica a queda de 50% dos perceptrões.

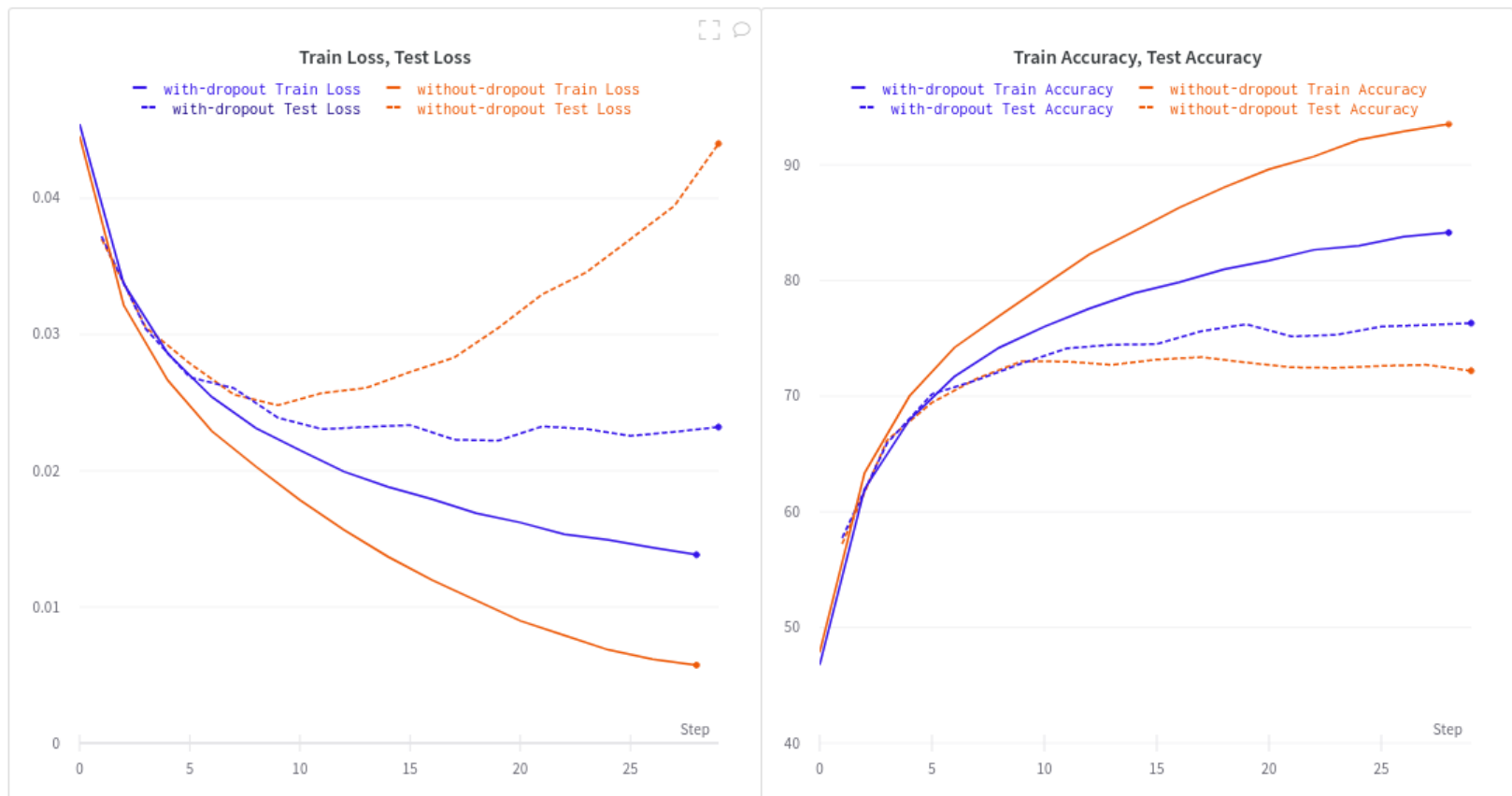


Dando um exemplo do dia a dia: ao voltar para casa de carro geralmente, seguimos sempre o mesmo caminho porque nos habituámos a este, mesmo que exista um caminho mais rápido. Agora aplicando o conceito de *dropout* (aleatoriamente fechando caminhos), para simular, utilizam-se algumas cancelas que impedem a passagem do carro. Ao colocar algumas cancelas aleatoriamente poderíamos descobrir um novo caminho que poderá ser mais rápido para casa do que os outros.



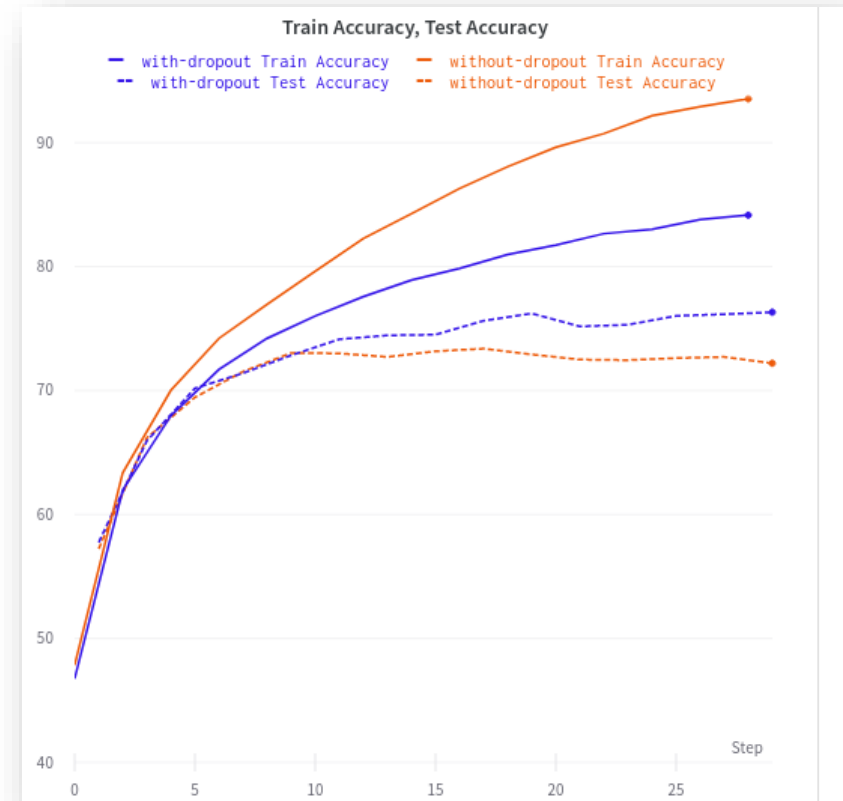


Analisando os gráficos seguintes é possível ver a diferença que faz a implementação do dropout numa rede neuronal. Com este método é possível diminuir a diferença (erro) entre a classificação usando os sub-datasets de treino ou teste.



É possível verificar que, sem a utilização de dropout (linhas **laranja**), existe uma enorme discrepância entre a precisão do modelo para o dataset de treino (linha contínua) e de teste (linha tracejada). O que significa que a rede neuronal está demasiado adaptada ao sub-dataset de treino, não estando bem preparada para classificar dados fora deste.

Já, ao analisarmos as linhas **azuis**, situação com dropout, verificamos que a precisão entre os dois sub-datasets é mais concordante, logo irá aprender melhor de forma a poder classificar dados novos de forma correta.





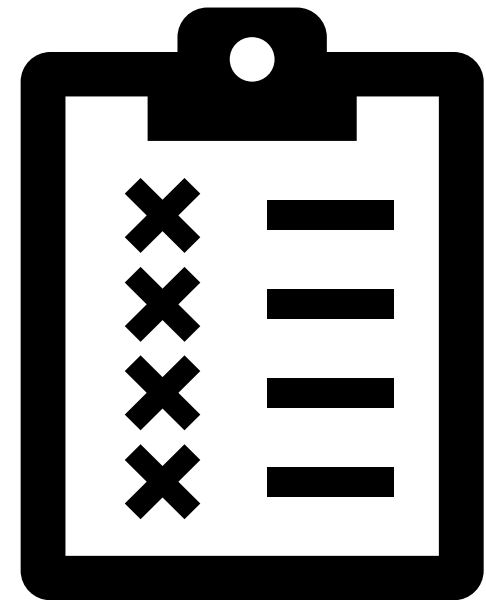
# Capítulo 2.

---

## Tipos de Redes Neuronaais

Existem vários tipos de redes neuronais profundas, sendo que cada uma tem a sua função. As redes neuronais que iremos abordar são:

- 2.1 - Redes Neurais Feedforward;
- 2.2 - Redes Neurais Convolucionais (RNCs);
- 2.3 - Redes Neurais Recorrente (RNRs);
- 2.4 - Redes Neurais Autoencoder.



# Capítulo 2.1

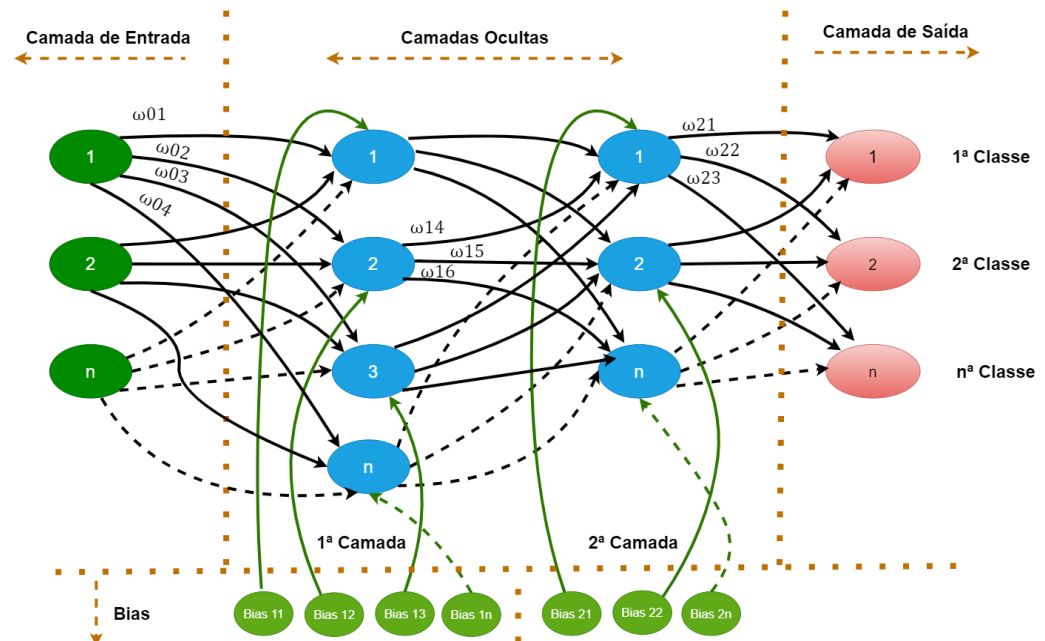
---

## Redes neuronais feedforward

## Redes neuronais feedforward

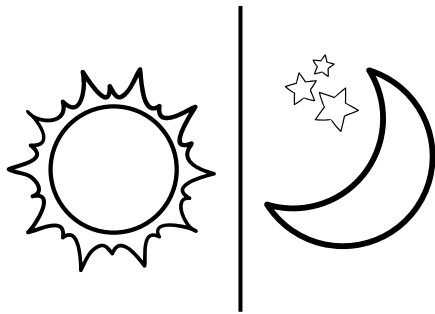
As redes neuronais feedforward ao contrário das redes neuronais recorrentes, que serão faladas mais tarde, não devolve os valores para trás. Estas são DNNs “simples”. Ou seja, tal como os perceptrões, estas são compostas pelas seguintes camadas já conhecidas:

- Camada de Entrada
- Camadas Ocultas
- Camada de Saída

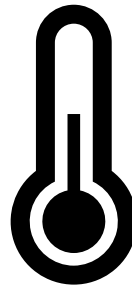


De forma a percebermos melhor como funcionam estas redes, iremos resolver o seguinte problema – determinar se está a chover ou não a partir de 3 valores de entrada:

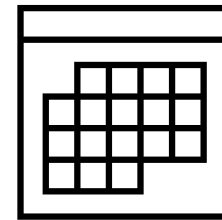
1. Dia/Noite
2. Temperatura
3. Mês



Entrada 1



Entrada 2



Entrada 3

Vamos supor que o valor limite (para a saída) seja 20, ou seja, se a saída for maior que 20, significa que está a chover, caso contrário, está um dia de sol. Dada uma tupla de dados com entradas ( $x_1, x_2, x_3$ ) como (0, 12, 11), pesos iniciais da rede feedforward ( $w_1, w_2, w_3$ ) como (0, 1, 1, 1) e bias como (1, 0, 0).

É possível ver como a rede neuronal calcula os dados em três etapas simples:

## 1. Multiplicação de pesos e entradas:

A entrada é multiplicada pelos valores dos pesos atribuídos, que neste caso seriam os seguintes:

$$(x_1 * w_1) = (0 * 0,1) = 0$$

$$(x_2 * w_2) = (1 * 12) = 12$$

$$(x_3 * w_3) = (11 * 1) = 11$$

## 2. Somando os bias:

Na próxima etapa, o produto encontrado na etapa anterior é adicionado aos seus respectivos bias. As entradas modificadas são então somadas em um único valor.

$$(x1 * w1) + b1 = 0 + 1$$

$$(x2 * w2) + b2 = 12 + 0$$

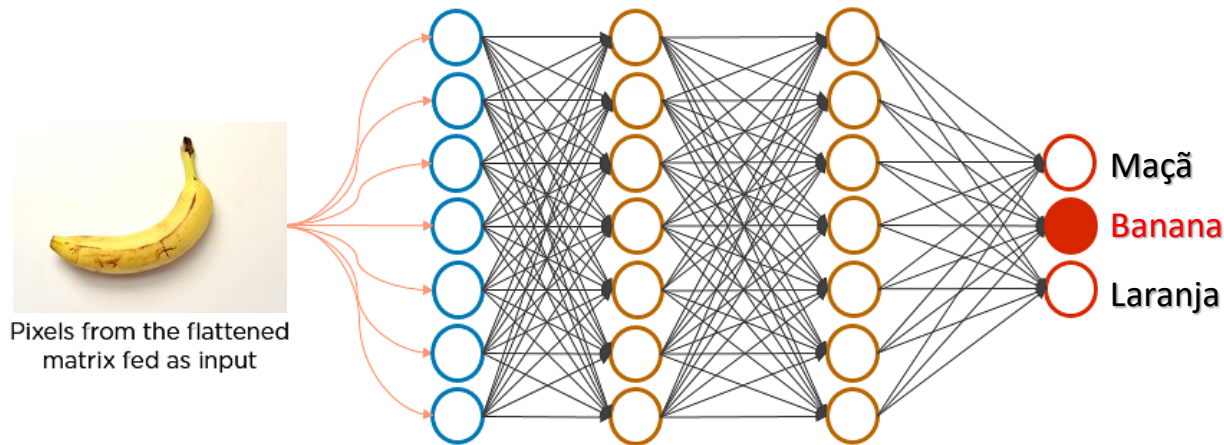
$$(x3 * w3) + b3 = 11 + 0$$

$$\text{Soma Ponderada} = (x1 * w1) + b1 + (x2 * w2) + b2 + (x3 * w3) + b3 = 23$$

## 3. Função de Ativação:

Usando a função de degrau unitário, sabendo que para um valor de soma ponderada superior a 20, sabemos que está a chover, e para menos de 20 não está a chover, e sabendo que a nossa soma ponderada é 23, então sabemos que está a chover.

Por exemplo de uma classificação de classes “maçã”, “banana” e “laranja”, temos:



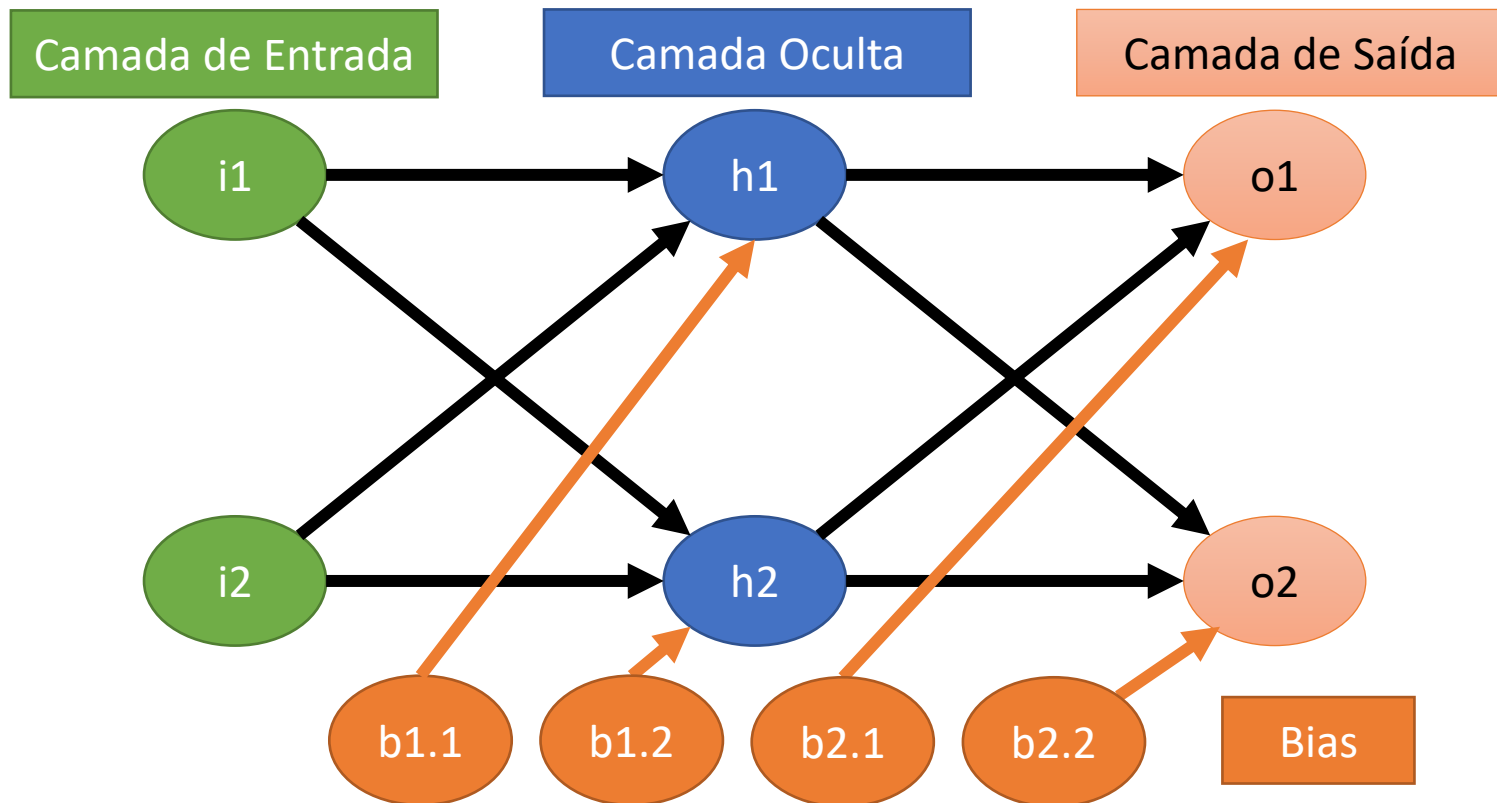
O processo realizado por uma redes neuronais feedforward resume-se nas seguintes etapas:

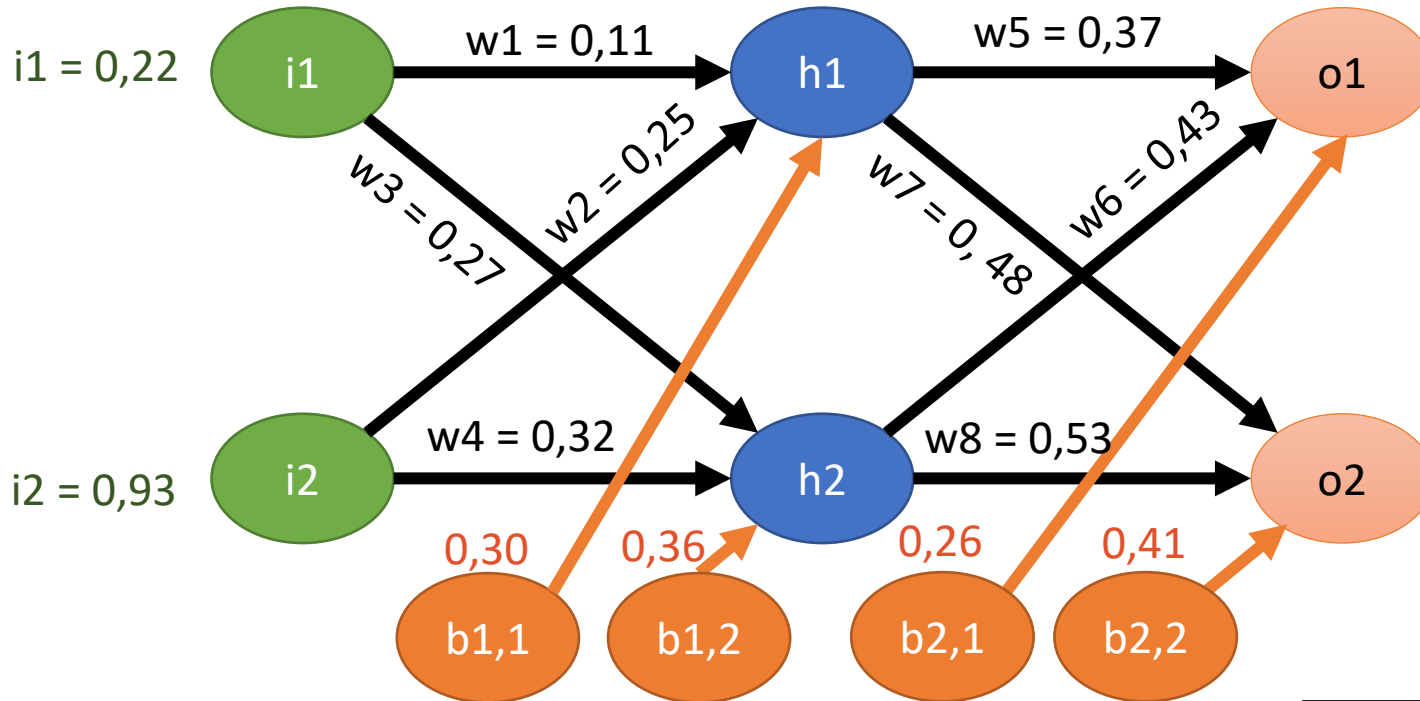
1. Os neurónios nas camadas ocultas detetam uma determinada característica, por exemplo, a cor;
2. Preserva o seu valor;
3. Comunica esse valor para as classes “maçã”, “banana” e “laranja”;
4. Ambas as classes verificam a característica e decidem se é relevante para elas.



Neste exemplo temos uma rede neuronal que identifica se é maçã ou banana. É constituída por duas entradas, dois perceptrões (uma camada oculta) e duas saídas.

A sua saída vai ter valores  $[1, 0]$  se for classificado como maçã e  $[0, 1]$  se for banana.





**Cálculos para a entrada h1 e h2:**

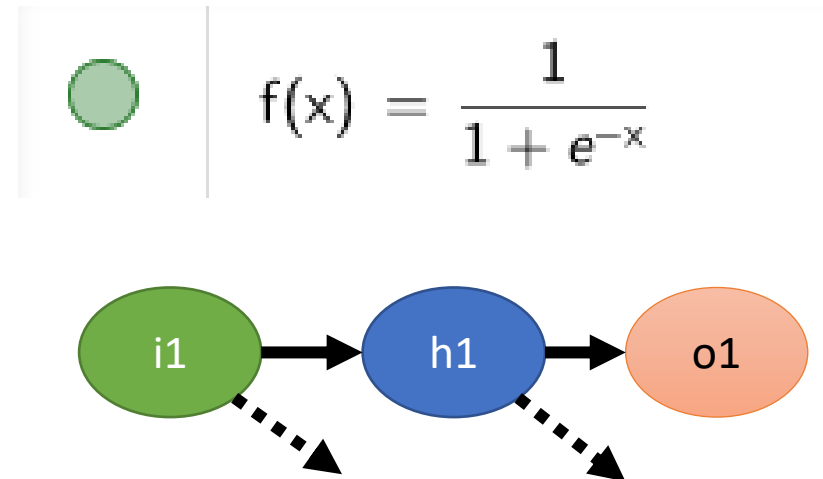
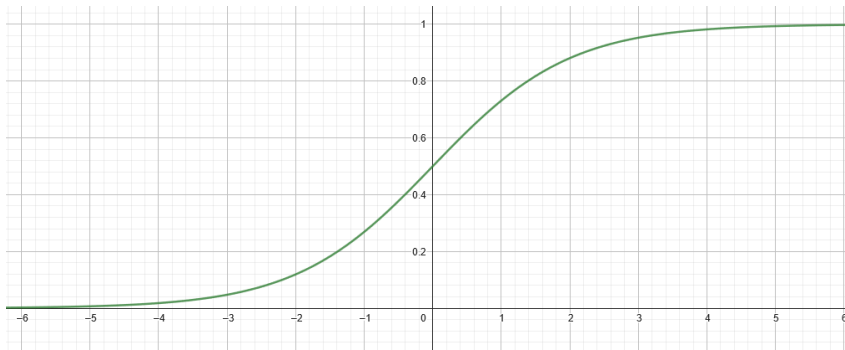
$$\begin{aligned} \text{Entrada}(h1) &= w1*i1 + w2*i2 + b1,1*1 \\ &= 0,11* 0,22 + 0,25*0,93 + 0,30*1 = \mathbf{0,55670} \end{aligned}$$

$$\text{Entrada}(h2) = w3*i1 + w4*i2 + b1,2*1 = \mathbf{0,71700}$$

**Resultados deste link:**

<https://colab.research.google.com/drive/1WsSyjNaYudU-O8wN9NeQIvvTIRQVzJW7?authuser=2#scrollTo=vlaJXYZIUGDJ>

Nesta rede as funções de ativação são função logística ou também conhecidas por função sigmoidal, onde os sua valores de entrada e de saída são compreendidos entre 0 e 1.



**Cálculos da função logica o1 e o2:**

$$\begin{aligned} \text{Saída}(h1) &= \frac{1}{1 + e^{-\text{Entrada}(h1)}} \\ &= \frac{1}{1 + e^{-0,55670}} = 0,63569 \end{aligned}$$

$$\begin{aligned} \text{Saída}(h2) &= \frac{1}{1 + e^{-\text{Entrada}(h2)}} = 0,67195 \end{aligned}$$

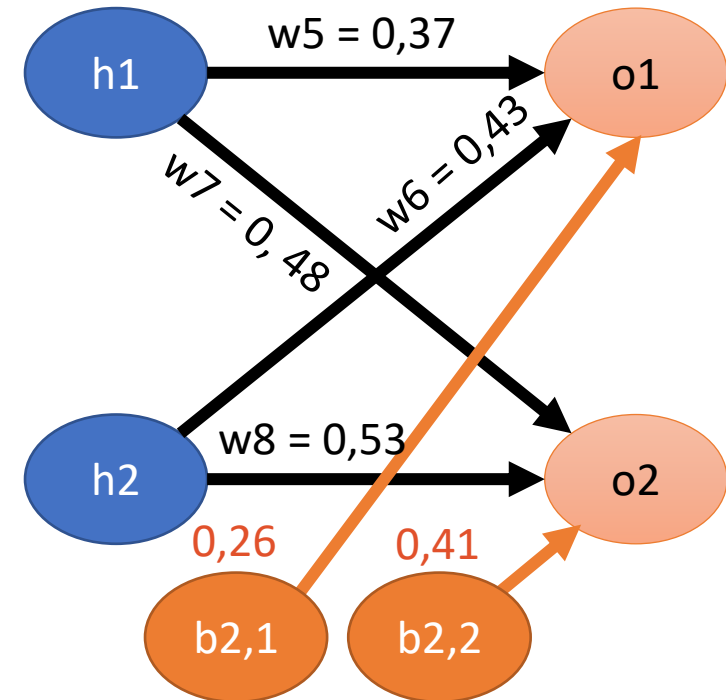
Agora fazemos o mesmo processo para o1 e o2:

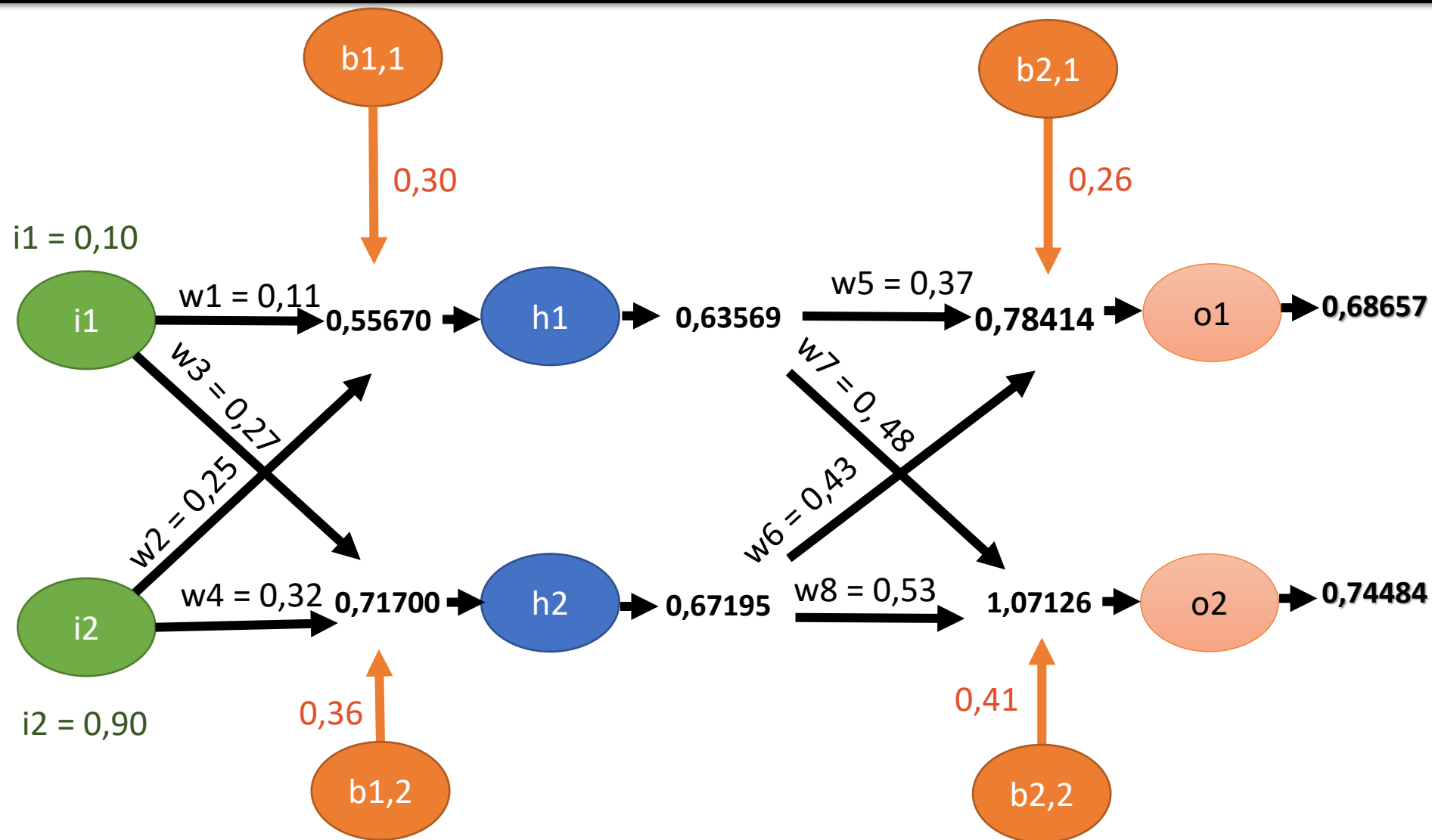
$$\begin{aligned} \text{Entrada(o1)} &= w5 * \text{Saída(h1)} + w6 * \text{Saída(h2)} + b2,1 * 1 \\ &= 0,37 * 0,63569 + 0,43 * 0,67195 + 0,26 * 1 \\ &= \mathbf{0,78414} \end{aligned}$$

$$\begin{aligned} \text{Entrada(o2)} &= w7 * \text{Saída(h1)} + w8 * \text{Saída(h2)} + b2,2 * 1 \\ &= \mathbf{1,07126} \end{aligned}$$

$$\begin{aligned} \text{Saída(o1)} &= \frac{1}{1 + e^{-\text{Entrada(o1)}}} \\ &= \frac{1}{1 + e^{-\mathbf{0,78414}}} = \mathbf{0,68657} \end{aligned}$$

$$\text{Saída(o2)} = \frac{1}{1 + e^{-\text{Entrada(o2)}}} = \mathbf{0,74484}$$





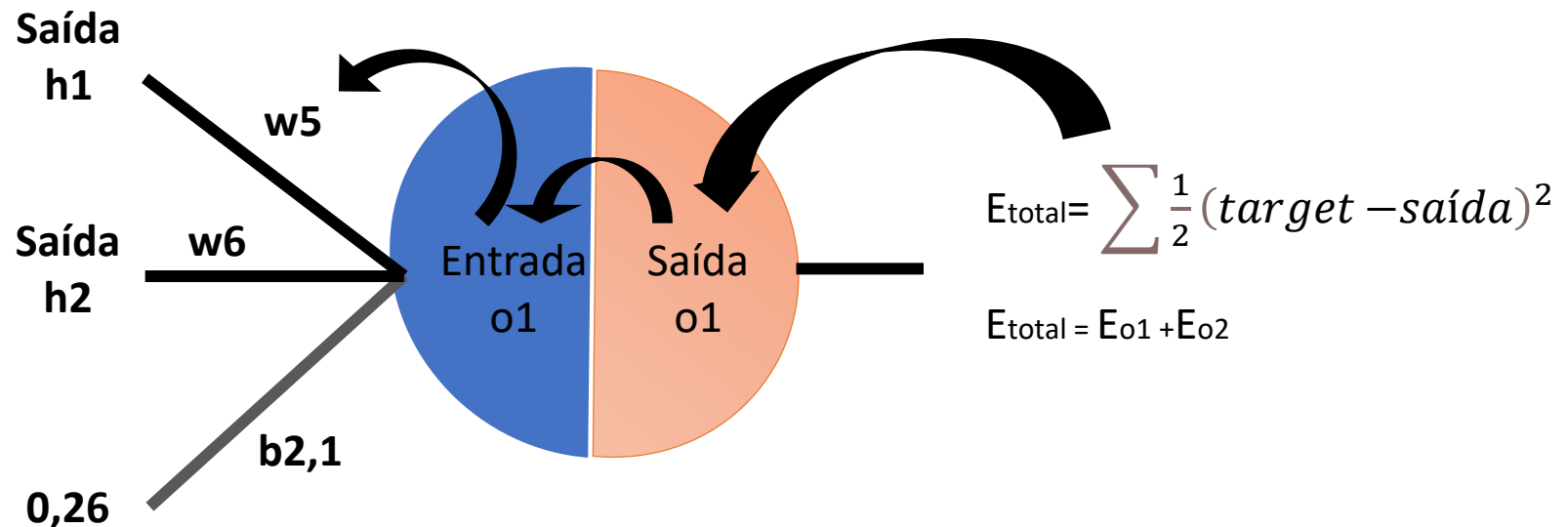
## Erro Quadrático

O erro quadrático é uma medida que calcula a diferença entre os valores observados e os valores previstos, elevando essa diferença ao quadrado.

$$\begin{aligned} E_{\text{total}} &= \sum \frac{1}{2} (\text{target} - \text{saída})^2 = \frac{1}{2} (\text{targeto1} - \text{Saída(o1)})^2 + \frac{1}{2} (\text{targeto2} - \text{Saída(o2)})^2 \\ &= \frac{1}{2} (0,01 - 0,6866)^2 + \frac{1}{2} (0,99 - 0,7448)^2 \\ &= \mathbf{0,25893} \end{aligned}$$

## Regra da cadeia

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial Saída(o1)} * \frac{\partial Saída(o1)}{\partial Entrada(o1)} * \frac{\partial E_{total}}{\partial w_5}$$



## Regra da cadeia

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial Saída(o1)} * \frac{\partial Saída(o1)}{\partial Entrada(o1)} * \frac{\partial Entrada(o1)}{\partial w_5}$$

$$E_{total} = \frac{1}{2} (targeto1 - Saída(o1))^2 + \frac{1}{2} (targeto2 - Saída(o2))^2 (=)$$

$$\frac{\partial E_{total}}{\partial Saída(o1)} = 2 * \frac{1}{2} (targeto1 - Saída(o1))^2 * -1 + 0 (=)$$

$$\frac{\partial E_{total}}{\partial Saída(o1)} = - (targeto1 - Saída(o1)) (=)$$

$$\frac{\partial E_{total}}{\partial Saída(o1)} = - (0,01 - 0,63569) (=)$$

$$\frac{\partial E_{total}}{\partial Saída(o1)} = \mathbf{0,67657}$$

Repetir mesmo processo:

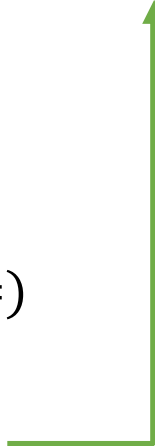
$$\frac{\partial E_{total}}{\partial Saída(o2)} = \mathbf{-0,24516}$$



$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial Saída(o1)} * \frac{\partial Saída(o1)}{\partial Entrada(o1)} * \frac{\partial Entrada(o1)}{\partial w_5}$$

$$Saída(o1) = \frac{1}{1+e^{-Entrada(o1)}} (=)$$

$$\frac{\partial Saída(o1)}{\partial Entrada(o1)} = Saída(o1) * (1 - Saída(o1)) (=)$$

$$\frac{\partial Saída(o1)}{\partial Entrada(o1)} = 0,68657 * (1 - 0,68657) = \mathbf{0,21519}$$


Repetir mesmo processo:

$$\frac{\partial Saída(o2)}{\partial Entrada(o2)} = \mathbf{0,19005}$$

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial Saída(o1)} * \frac{\partial Saída(o1)}{\partial Entrada(o1)} * \frac{\partial Entrada(o1)}{\partial w_5}$$

$$Entrada(o1) = w_5 * Saída(h1) + w_6 * Saída(h2) + b_{2,1} * 1 (=)$$

$$\frac{\partial Entrada(o1)}{\partial w_5} = 1 * Saída(h1) * w_5^{(1-1)} + 0 + 0 (=)$$

$$\frac{\partial Entrada(o1)}{\partial w_5} = Saída(h1) (=)$$

$$\frac{\partial Entrada(o1)}{\partial w_5} = \mathbf{0,63569}$$

Repetir mesmo processo:

$$\frac{\partial Entrada(o1)}{\partial w_7} = \mathbf{0,67195}$$

Depois de calcular todos os valores substituímos:

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial Saída(o1)} * \frac{\partial Saída(o1)}{\partial Entrada(o1)} * \frac{\partial Entrada(o1)}{\partial w_5}$$

$$\frac{\partial E_{total}}{\partial w_5} = 0,67657 * 0,21519 * 0,63569$$
$$\frac{\partial E_{total}}{\partial w_5} = \mathbf{0,09255}$$

Repetir mesmos processo:

$$\frac{\partial E_{total}}{\partial w_7} = \mathbf{-0,03131}$$

Calcular novos pesos com uma taxa de aprendizagem  $\eta = 0,5$ :

$$\begin{aligned}w5^+ &= w5 - \eta * \frac{\partial E_{total}}{\partial w5} \\&= 0,37 - 0,5 * 0,09255 \\&= \mathbf{0,32372}\end{aligned}$$

$$\begin{aligned}w6^+ &= \mathbf{0,38372} \\w7^+ &= \mathbf{0,49654} \\w8^+ &= \mathbf{0,54565}\end{aligned}$$

Calcular novos bias:

$$\begin{aligned}b1,1 &= 1 - \eta * \frac{\partial E_{total}}{\partial w5} \\&= 1 - 0,5 * 0,09255 \\&= \mathbf{0,29920}\end{aligned}$$

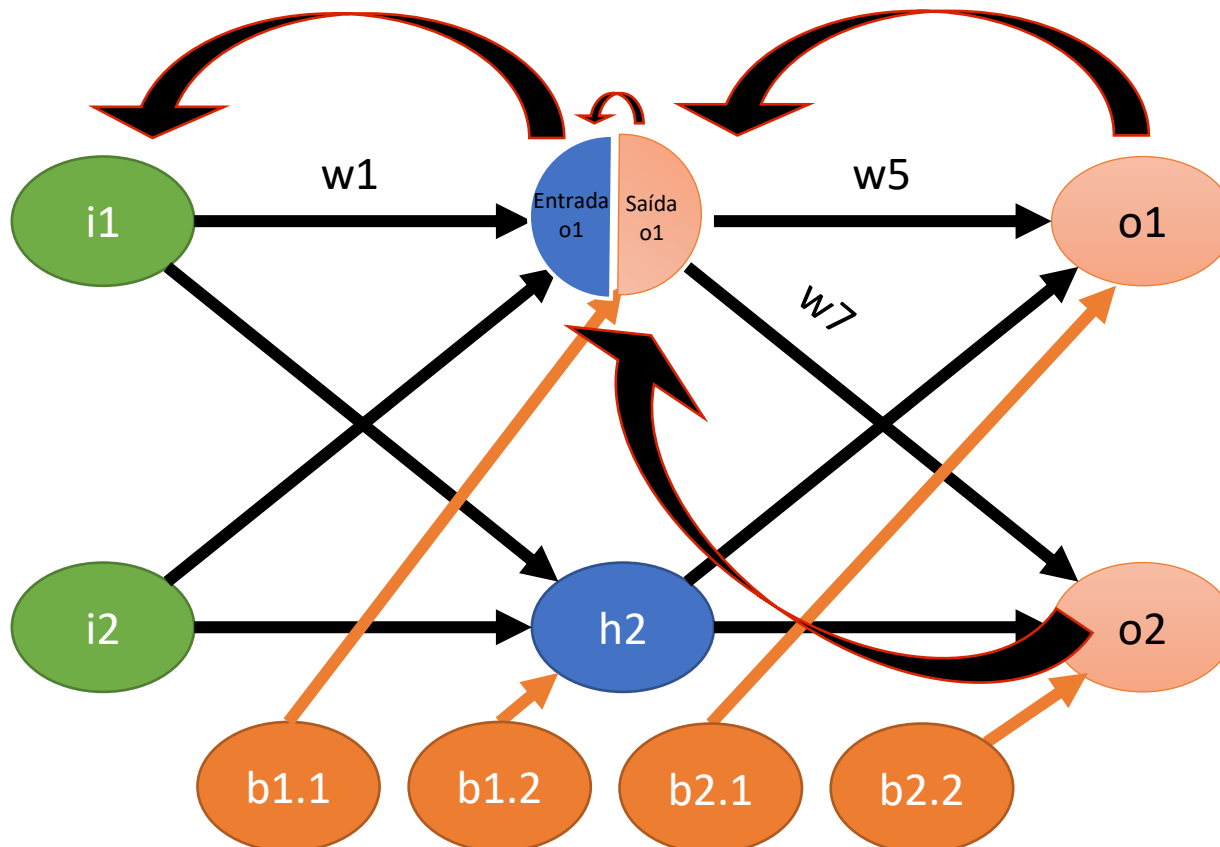
$$\begin{aligned}b1,2 &= 1 - \eta * \frac{\partial E_{total}}{\partial w7} \\&= \mathbf{0,35677}\end{aligned}$$

**Regra da cadeia**

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial Saída(h1)} * \frac{\partial Saída(h1)}{\partial Entrada(h1)} * \frac{\partial Entrada(h1)}{\partial w_1}$$



$$\frac{\partial E_{total}}{\partial Saída(h1)} = \frac{\partial E_{o1}}{\partial Saída(h1)} + \frac{\partial E_{o2}}{\partial Saída(h1)}$$



$$\frac{\partial E_{total}}{\partial Saída(h1)} = \frac{\partial E_{o1}}{\partial Saída(h1)} + \frac{\partial E_{o2}}{\partial Saída(h1)}$$

$$\frac{\partial E_{o1}}{\partial Saída(h1)} = \frac{\partial E_{o1}}{\partial Entrada(h1)} * \frac{\partial Entrada(o1)}{\partial Saída(h1)} = 0,13870 * 0,37 = \mathbf{0,05540}$$

$$\frac{\partial E_{o1}}{\partial Entrada(h1)} = \frac{\partial E_{o1}}{\partial Saída(o1)} * \frac{\partial Saída(o1)}{\partial Entrada(o1)} (=)$$

$$\frac{\partial E_{o1}}{\partial Entrada(h1)} = 0,67657 * 0,21519 = \mathbf{0,13850}$$

Resultados calculados anteriormente

$$Entrada(o1) = w5 * Saída(h1) + w7 * Saída(h2) + b2,1 * 1 (=)$$

$$\frac{\partial Entrada(o1)}{\partial Saída(h1)} = w5 = \mathbf{0,37}$$

Nota :

$$\frac{\partial E_{o1}}{\partial Saída(o1)} = \frac{\partial E_{total}}{\partial Saída(o1)}$$

Derivada em  
ordem "Saída(h1)"  
54

Repetir o mesmo processo para  $\frac{\partial Eo2}{\partial Saída(h1)}$ :

$$\frac{\partial Eo2}{\partial Saída(h1)} = \mathbf{-0,01906}$$

Substituir os valores:

$$\frac{\partial Etotall}{\partial Saída(h1)} = \frac{\partial Eo1}{\partial Saída(h1)} + \frac{\partial Eo2}{\partial Saída(h1)} = \mathbf{0,05540 + (- 0,01906) = 0,03635}$$

Agora vamos descobrir  $\frac{\partial Etotall}{\partial w1} = \frac{\partial Etotall}{\partial Saída(h1)} * \frac{\partial Saída(h1)}{\partial Entrada(h1)} * \frac{\partial Entrada(h1)}{\partial w1}$

Calcular  $\frac{\partial Entrada(h1)}{\partial w1}$ :

$$Saída(h1) = \frac{1}{1+e^{-Entrada(h1)}} (=) \frac{\partial Saída(h1)}{\partial Entrada(h1)} = Saída(h1) * (1 - Saída(h1)) (=)$$

$$\frac{\partial Saída(h1)}{\partial Entrada(h1)} = 0,63569 * (1 - 0,63569) = \mathbf{0,24130}$$

Calcular  $\frac{\partial \text{Entrada}(h1)}{\partial w1}$  :

$$\text{Entrada}(h1) = w1 * i1 + w2 * i2 + b1,1 * 1 (=)$$

$$\frac{\partial \text{Entrada}(h1)}{\partial w1} = i1 = \mathbf{0,01}$$

Depois de calcular todos os valores substituímos:

$$\frac{\partial E_{total}}{\partial w1} = \frac{\partial E_{total}}{\partial \text{Saída}(h1)} * \frac{\partial \text{Saída}(h1)}{\partial \text{Entrada}(h1)} * \frac{\partial \text{Entrada}(h1)}{\partial w1} (=)$$

$$\frac{\partial E_{total}}{\partial w1} = 0,03635 * 0,24130 * 0,01 (=)$$

$$\frac{\partial E_{total}}{\partial w1} = \mathbf{0,00044}$$

Repetir o mesmo processo para  $\frac{\partial E_{total}}{\partial w2}$



Calcular novos pesos com uma taxa de aprendizagem  $\eta = 0,5$ :

$$\begin{aligned}w1^+ &= w1 - \eta * \frac{\partial E_{total}}{\partial w1} \\&= 0,11 - 0,5 * 0,00044 \\&= 0,3237\end{aligned}$$

$$\begin{aligned}w2^+ &= \mathbf{0,19956} \\w3^+ &= \mathbf{0,24978} \\w4^+ &= \mathbf{0,29956}\end{aligned}$$

Calcular novos bias:

$$\begin{aligned}b2,1 &= b2,1 - \eta * \frac{\partial E_{total}}{\partial w1} \\&= 0,26 - 0,5 * 0,00044 \\&= \mathbf{0,55891}\end{aligned}$$

$$\begin{aligned}b2,2 &= b2,2 - \eta * \frac{\partial E_{total}}{\partial w2} \\&= \mathbf{0,61137}\end{aligned}$$

Resultados finais retirados de (código Python):

<https://colab.research.google.com/drive/1WsSyjNaYudU-O8wN9NeQIvvTIRQVzJW7?authuser=2#scrollTo=vlaJXYZIUGDJ>

(Baseado em: <https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/>)

# Capítulo 2.2

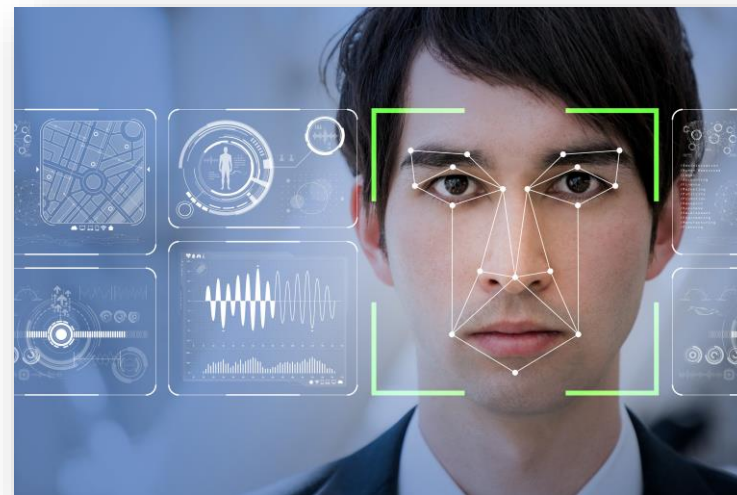
---

Redes neuronais  
convolucionais (RNCs)

Primeiramente, iremos abordar as Redes Convolucionais (RNCs). Estas são maioritariamente utilizadas para análise de imagens. É através da utilização destas que os nossos telemóveis são capazes de fazer reconhecimento facial, por exemplo.



<https://payspacemagazine.com/tech/face-recognition-technology-to-change-the-world/>

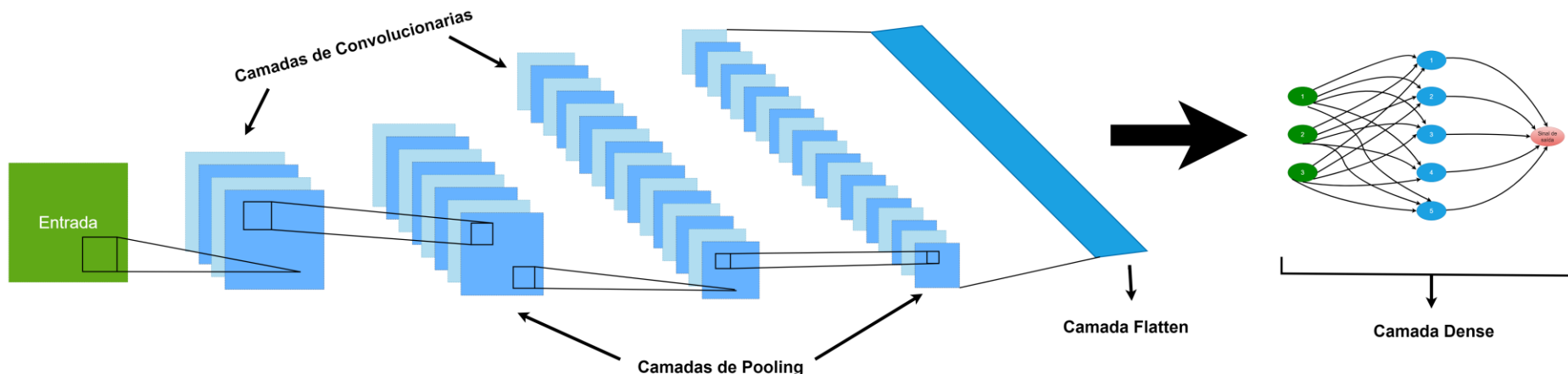


<https://www.telusinternational.com/insights/ai-data/article/what-is-facial-recognition>

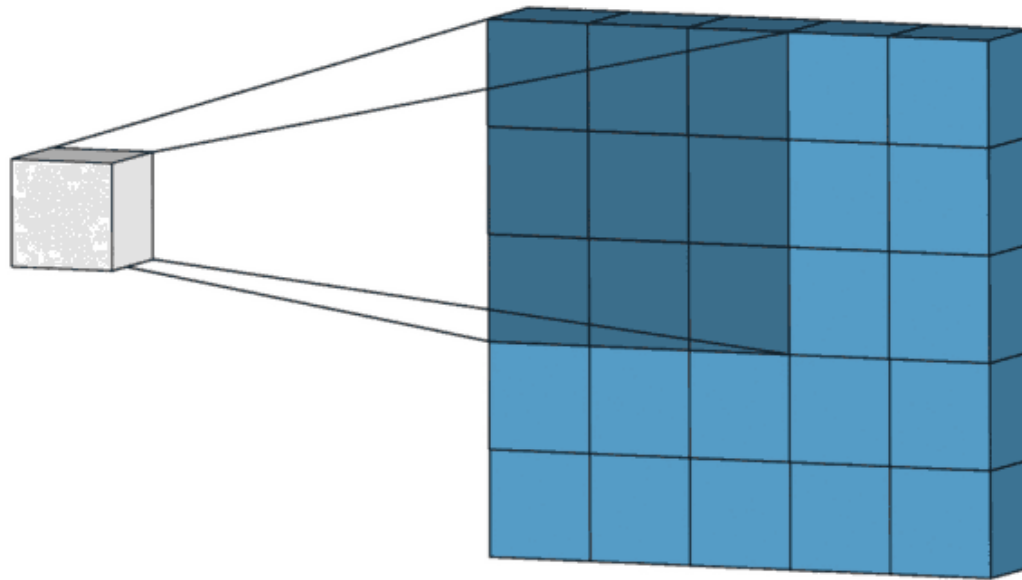
Na imagem seguinte podemos ver a estrutura de uma rede convencional e as várias camadas que a constituem.

Numa rede neuronal convolucional temos as seguinte camadas:

- ✓ Camadas Convolucionárias
- ✓ Camadas Pooling;
- ✓ Camadas Flatten;
- ✓ Camadas “Fully-Connected” (ou “Dense”).

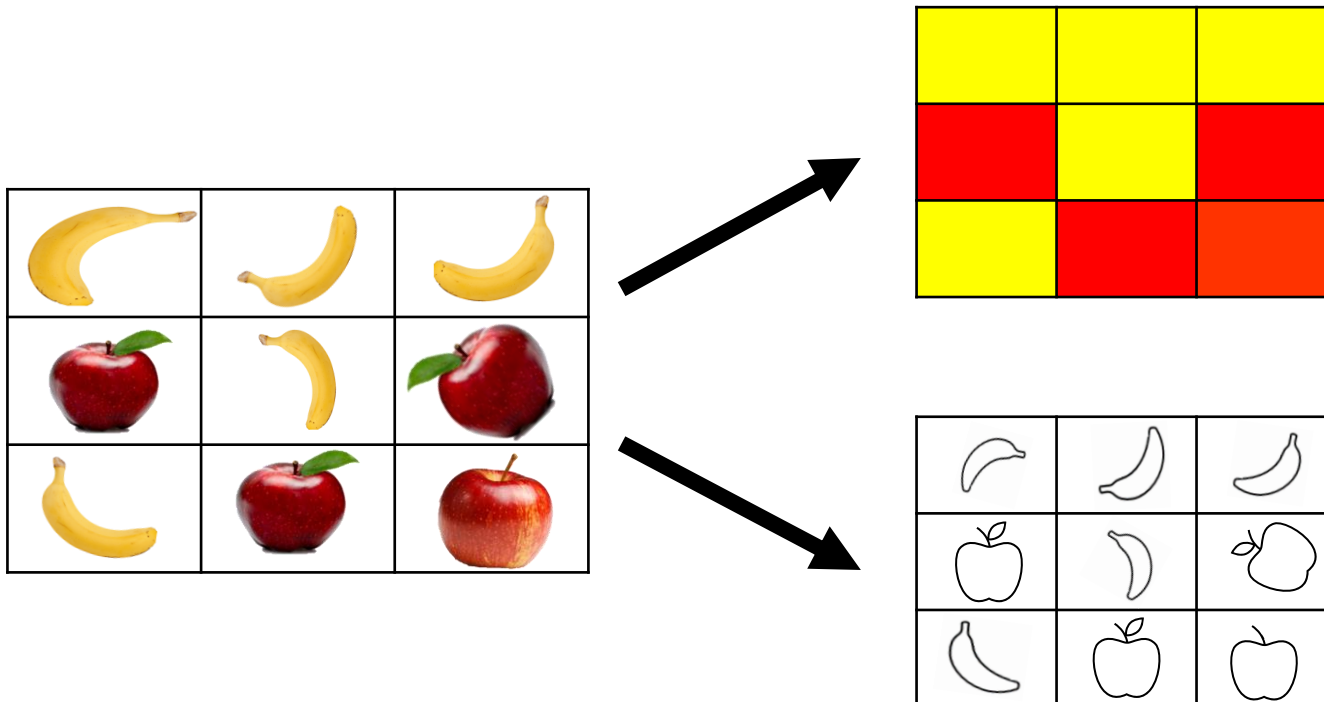


Nesta camada, os dados de entrada são recebidos em forma de matriz, que, posteriormente, é usada para realizar o produto escalar, entre esta e outra matriz, a que chamamos de filtro/kernel (conjunto de parâmetros que a rede neuronal aprende).



Nas Redes Convolucionais são usados filtros (kernels), que dividem a informação em sub-blocos. Cada filtro extrai uma característica diferente.

Neste caso abaixo é aplicado a convolução na matriz e a informação é filtrada identificam os padrões como cores e formas mas podia ser outro padrão como texturas.



Tal como nos valores dos pesos, explicados anteriormente, inicialmente, os valores dos filtros são atribuídos aleatoriamente.

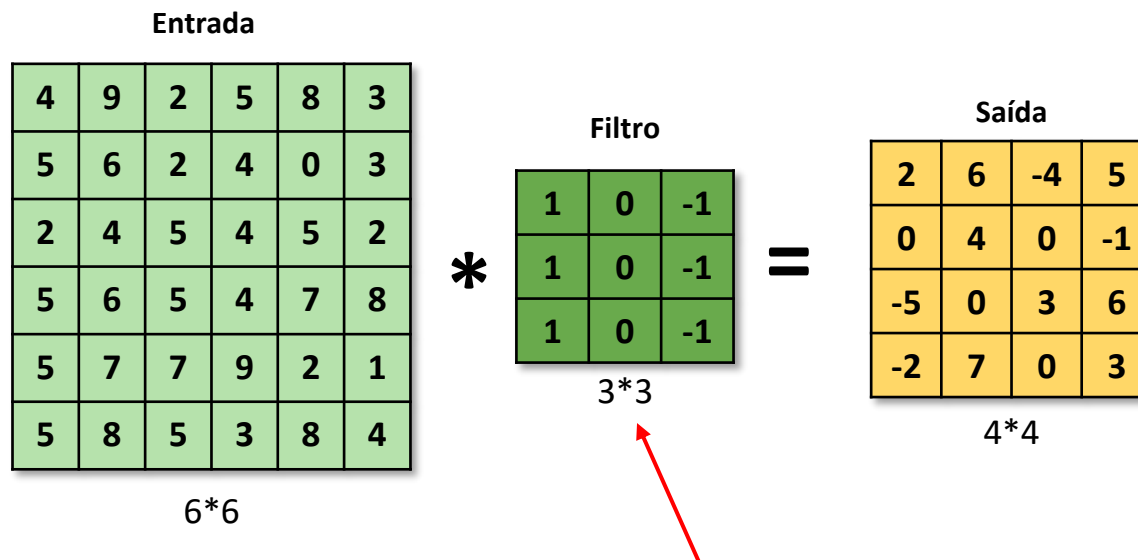
Para além disso, as Redes Neurais Convolucionais também têm a capacidade de aprender e modificar os valores dos filtros, estes não têm de ser deduzidos pelo utilizador.

0	0	0
0	0	0
0	0	0



Um filtro em uma CNN é como uma matriz de pesos com a qual multiplicamos uma parte da imagem de entrada para gerar uma saída convolucional.

- Supondo que temos uma imagem de tamanho  $6 \times 6$ , atribuímos aleatoriamente um filtro de tamanho  $3 \times 3$ , que é então multiplicado por diferentes seções  $3 \times 3$  da imagem para formar o que é conhecido como saída convolucional, neste caso de  $4 \times 4$ .



O tamanho do filtro geralmente é menor que o tamanho da imagem original.

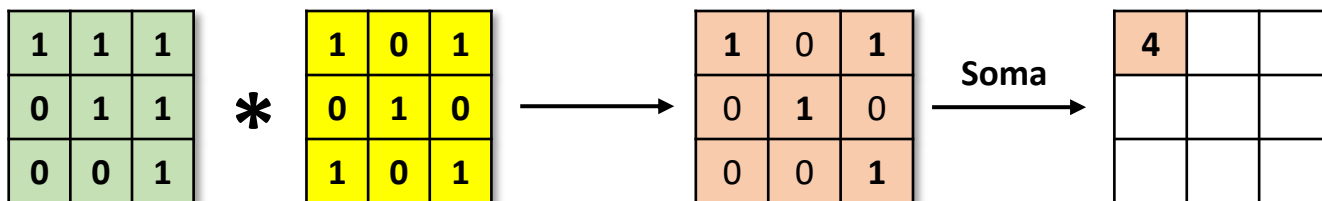
Utilizando um exemplo mais simples, temos o seguinte. Como no exemplo anterior, usamos um filtro de matriz  $3 \times 3$ , mas, desta vez, este é usado para filtrar uma entrada de matriz  $5 \times 5$ .

Entrada					Filtro		
1	1	1	0	0	1	0	1
0	1	1	1	0	0	1	0
0	0	1	1	1	0	1	0
0	0	1	1	0	1	0	1
0	1	1	0	0			

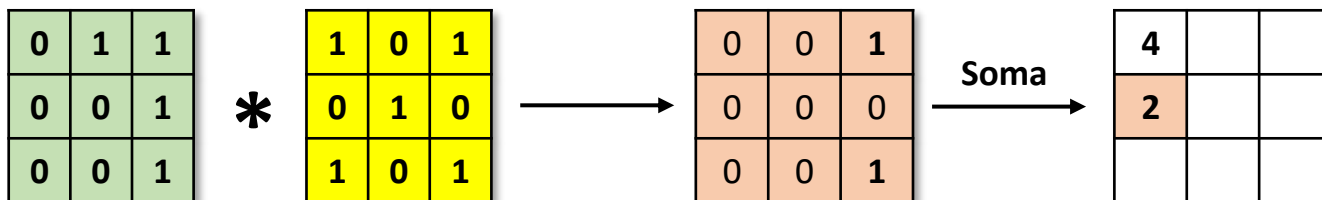
Este filtro é então multiplicado por diferentes secções  $3 \times 3$  da entrada, somando-se os valores no fim para formar a matriz de saída. Começando pela primeira secção temos:

1	1	1	0	0	→	1	1	1
0	1	1	1	0		0	1	1
0	0	1	1	1		0	0	1
0	0	1	1	0				
0	1	1	0	0				

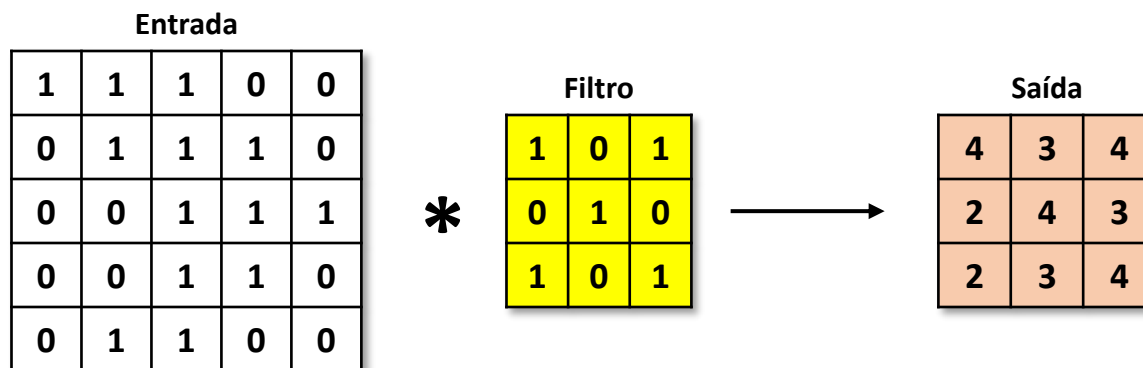
Sendo assim temos, para a primeira secção:



Seguindo o exemplo da primeira, para a segunda secção temos:



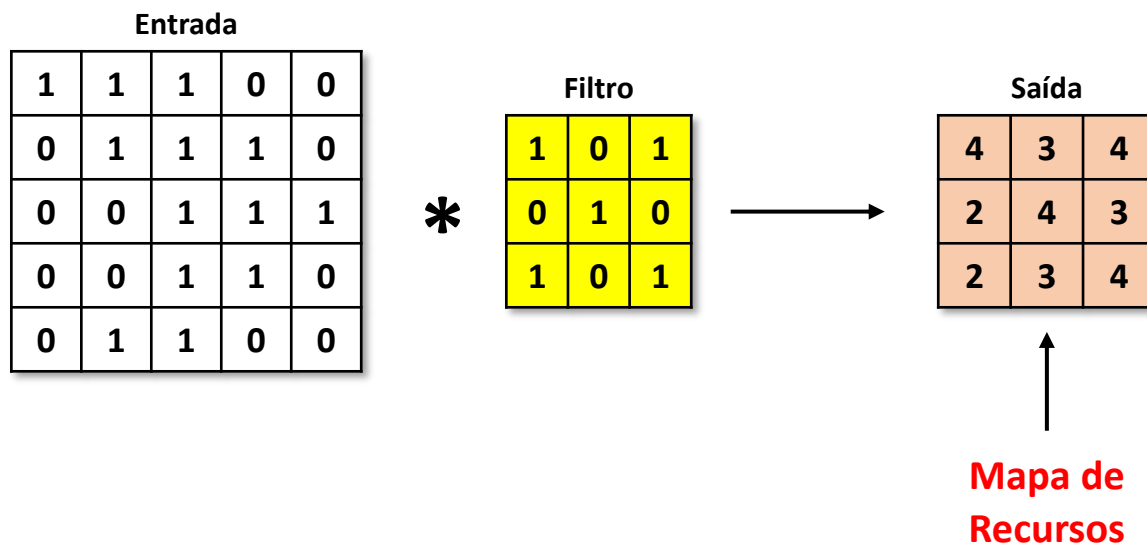
Ao realizarmos a mesma operação para todas as secções, temos como resultado final a seguinte matriz de saída (mapa de recursos):



Durante a retropropagação, tal como se fossem pesos, os valores do filtro são atualizados para minimização dos erros.

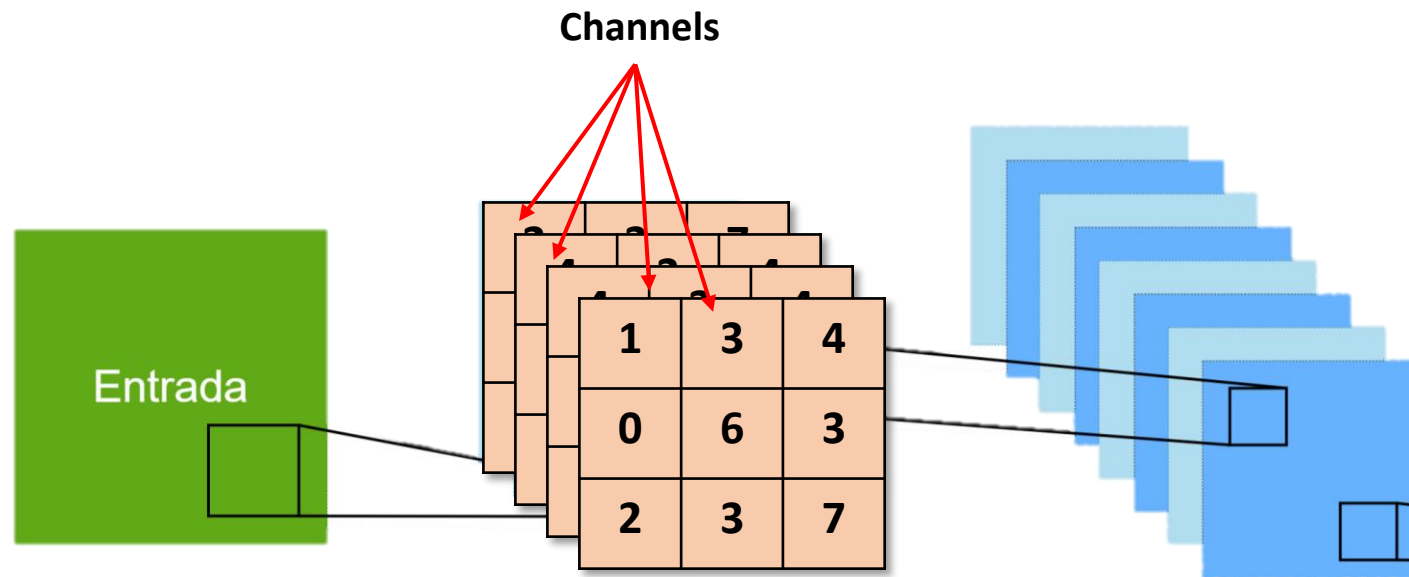
Os mapas de recursos são as matrizes obtidas aplicando os filtros imagem de entrada ou à saída do mapa de recursos das camadas anteriores.

Aproveitando o exemplo anterior, a matriz a que se chamou saída, é o nosso mapa de recursos obtido aplicando o filtro à nossa matriz de entrada:



Resumindo, os mapas de características são gerados usando o processo de convolução, onde o filtro é aplicado a uma imagem de entrada de maneira a obtermos as “informações” mais importantes de entrada.

Os Channels são o nome que se dá aos diferentes mapas de recursos que saem depois da utilização de filtros diferentes. Neste caso, podemos observar que na segunda camada existem 4 channels, cada um com o seu mapa de recurso diferente.

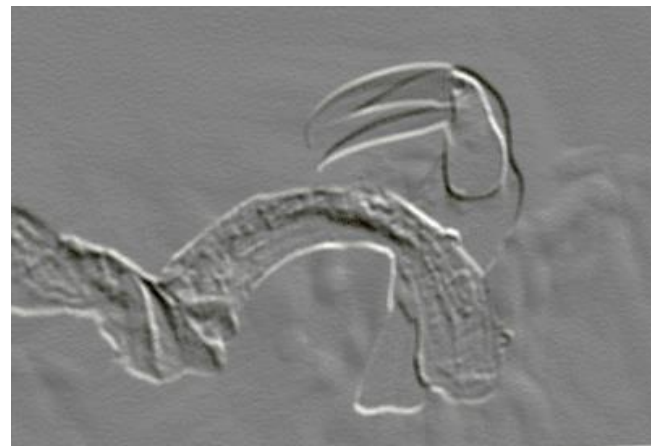


Tendo a imagem seguinte de um pássaro numa árvore, podemos ver um exemplo do que seria um feature map obtido através dessa imagem, depois de aplicado um filtro.



**Entrada**

**Convolução**



**Feature Map**

Dependendo do filtro, o resultado final que obtemos no feature map poderá ser diferente. Na imagem seguinte é possível ver outro exemplo de um feature map que poderíamos obter através de outra imagem de entrada.



**Entrada**

**Convolução**

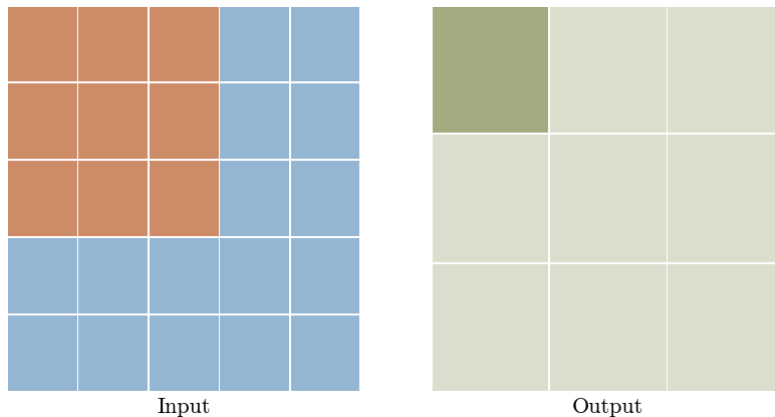


**Feature Map**

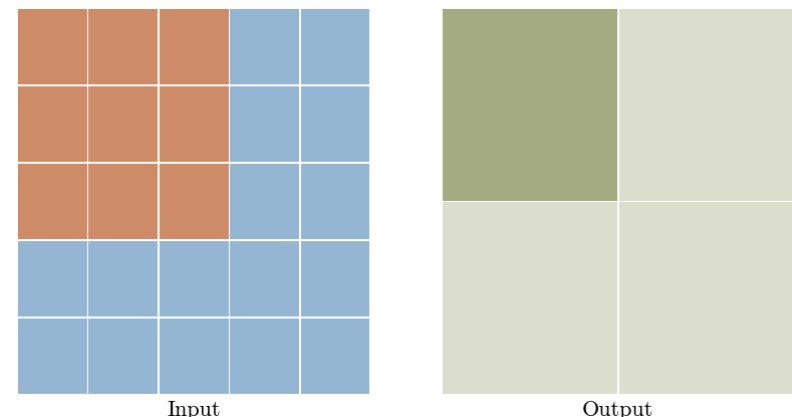


Nem sempre, nas camadas convolucionais, se move o filtro pixel por pixel. Nestas, existe um parâmetro chamado passo (stride) que dita a movimentação do filtro.

**Passo = 1**



**Passo = 2**

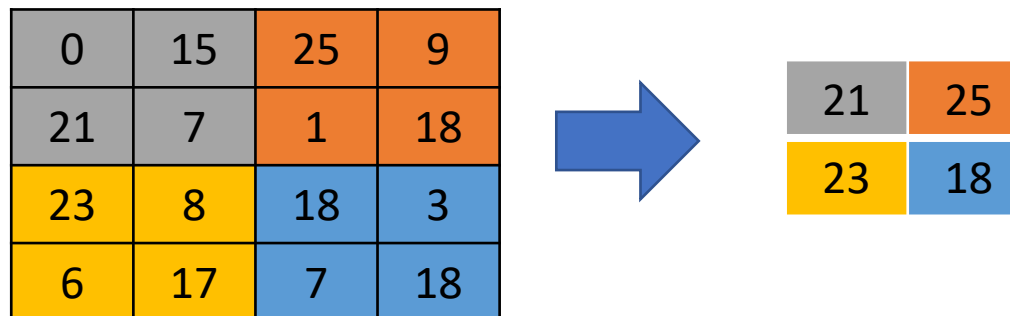


De forma simples, o passo (stride) é um parâmetro do filtro da rede neuronal que modifica o quanto o filtro se move sobre a imagem. Por exemplo, se o passo de uma rede neuronal for definido como 1, o filtro moverá um pixel ou unidade de cada vez, se for definido como 2 irá mover-se 2 pixéis, e assim sucessivamente.

## Camadas de Pooling

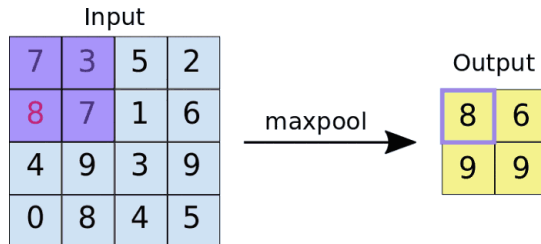
Esta camada recebe o mapa de recursos da camada anterior, analisa, reduz o tamanho dos dados e envia para a próxima camada.

O exemplo mais utilizado para demonstrar a aplicação desta camada é o max pooling. Este divide em parte e reduz o tamanho, retirando os valores mais altos como podemos ver abaixo.



As camadas de Pooling são usadas para reduzir as dimensões do mapa de características. Assim, reduz o número de parâmetros a aprender e a quantidade de computação realizada na rede. Existem 3 tipos de camadas de Pooling:

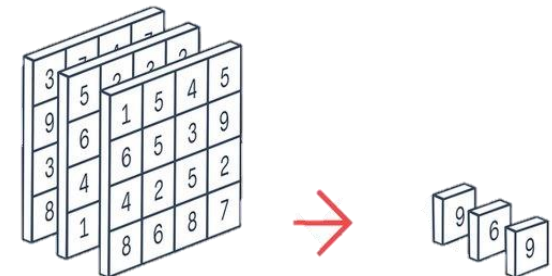
## Max Pooling



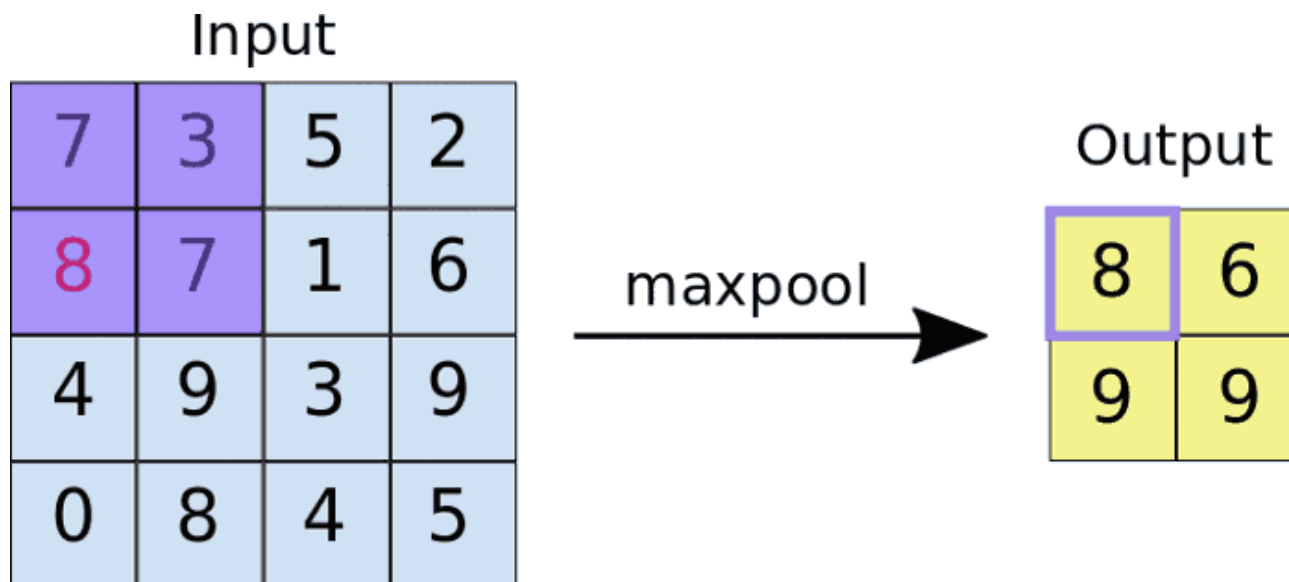
## Average Pooling



## Global Pooling



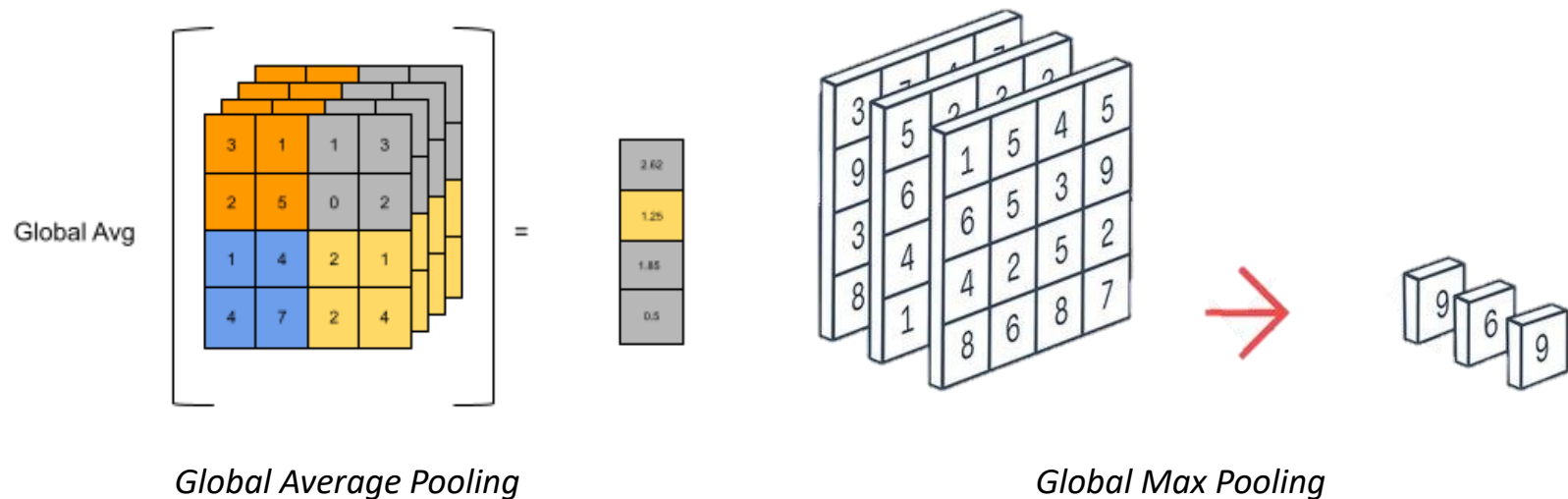
Max Pooling é uma função que seleciona o elemento máximo da região do mapa de características coberto pelo filtro. Assim, a saída após a camada de max pooling seria um mapa de características contendo os recursos com valor superior do mapa de recursos anterior.



O Average Pooling calcula a média dos elementos presentes na região do mapa de características coberto pelo filtro. Assim, enquanto o Max Pooling fornece o recurso mais proeminente, o Average Pooling fornece a média das características.

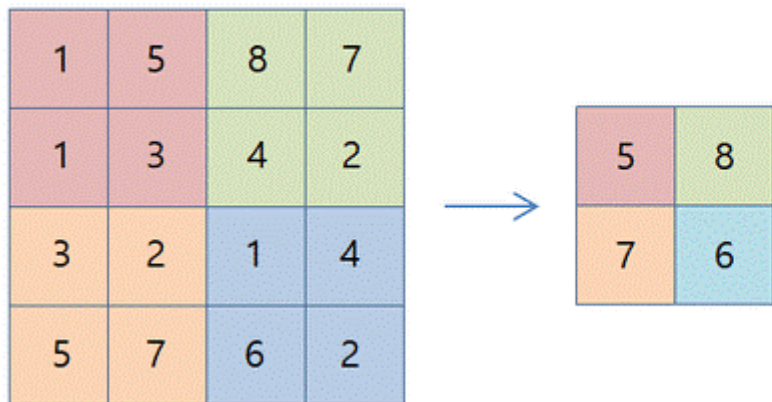


Com o Global Pooling é possível reduzir as dimensões das camadas da rede, por exemplo, de 3 dimensões para 1 dimensão. Dentro do Global Pooling, podemos ainda usar as duas funções anteriores, Max Pooling e Average Pooling, tendo assim como funções a Global Max Pooling e Global Average Pooling.

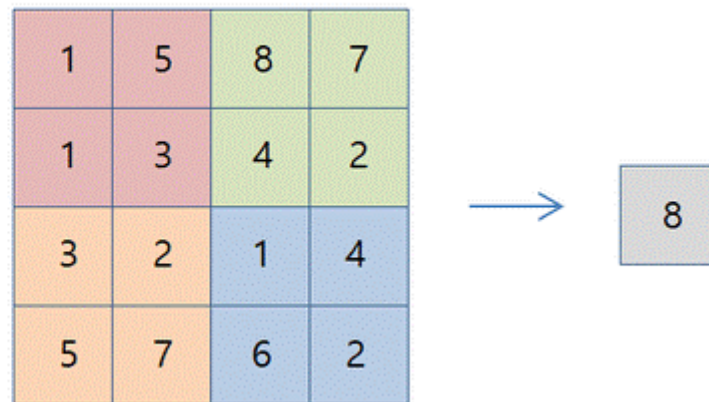


Enquanto na Max Pooling, obtemos uma matriz de acordo com o filtro utilizado (neste caso 2x2), na Global Max Pooling é apenas colocado na saída o maior valor de toda o mapa de características para cada camada.

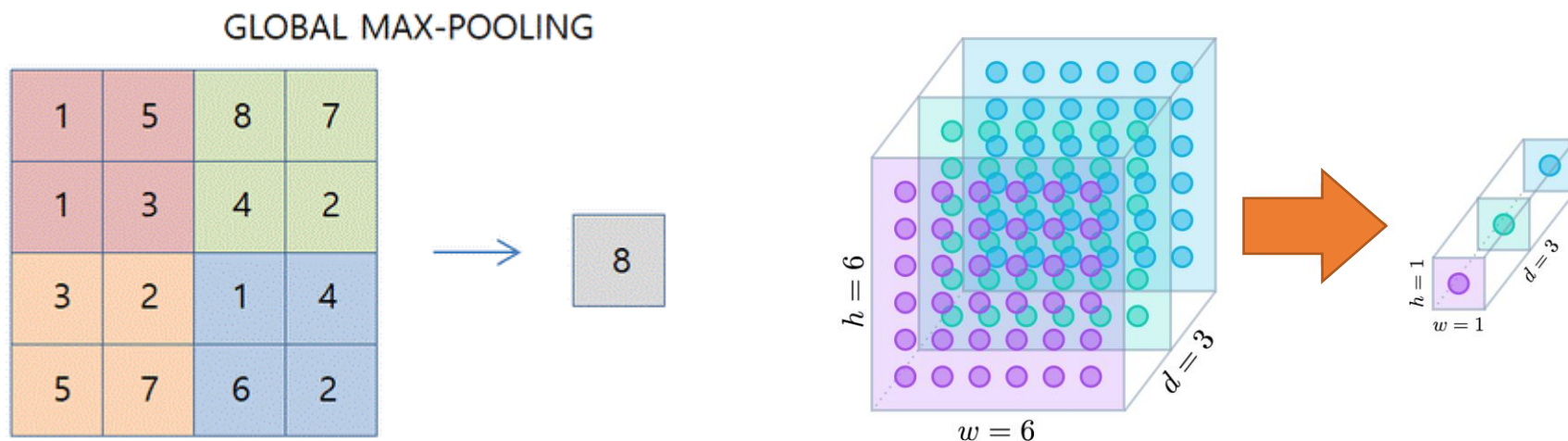
MAX-POOLING



GLOBAL MAX-POOLING

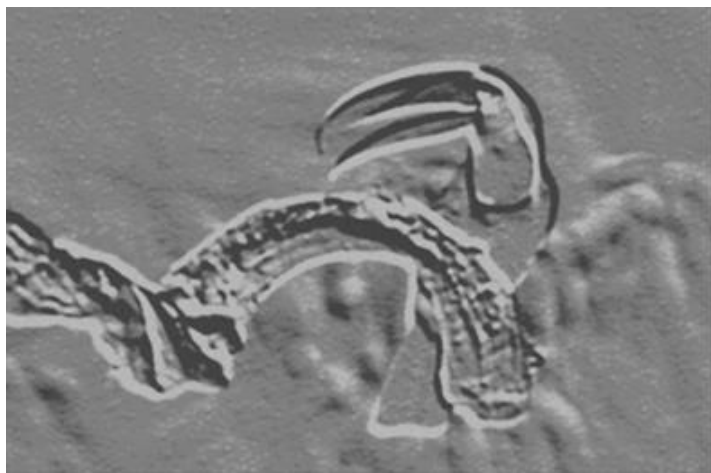


Enquanto na Max Pooling, obtemos uma matriz de acordo com o filtro utilizado (neste caso 2x2), na Global Max Pooling é apenas colocado na saída o maior valor de toda o mapa de características para cada camada.





Usando as imagens obtidas anteriormente, é possível ver um exemplo do resultado obtido através dessas imagens, no final de passarem pelas camadas de pooling.



**Feature Map**

**Pooling**



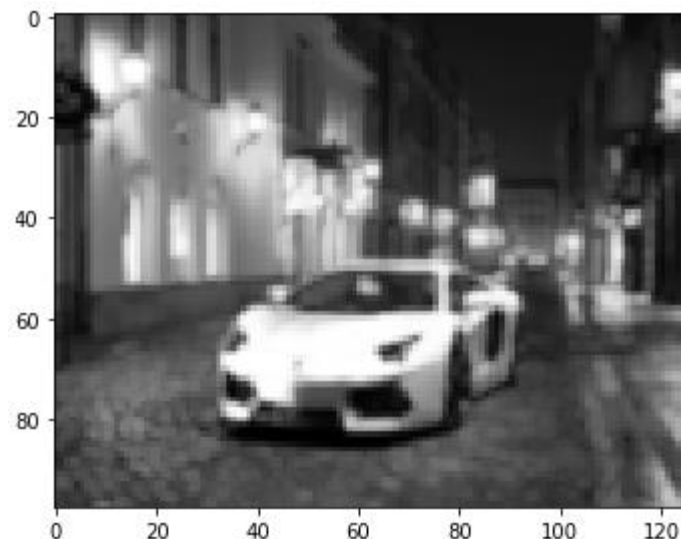
**Resultado**

Tal como no exemplo anterior, é possível perceber que os resultados poderão variar dependendo do feature map obtido anteriormente, assim como da operação de pooling usada.



**Feature Map**

**Pooling**



**Resultado  
(Max Pooling)**

É possível ver que na imagem seguinte, o pixel A, passa pelo filtro utilizado apenas 1 vez, diminuindo a sua influência na matriz de saída.

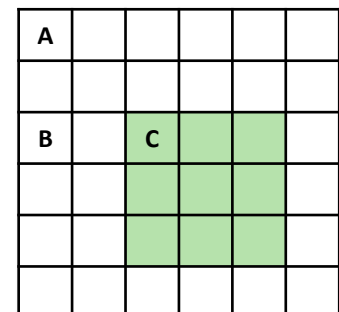
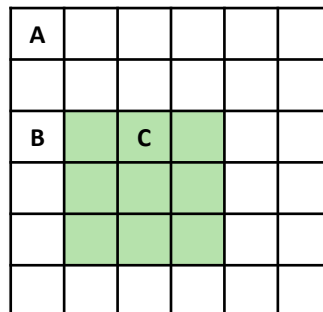
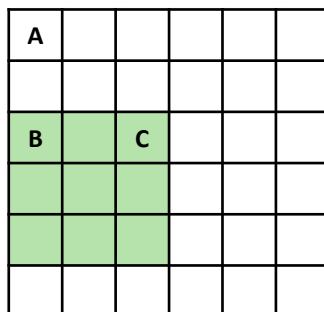
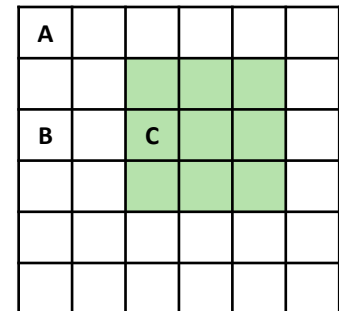
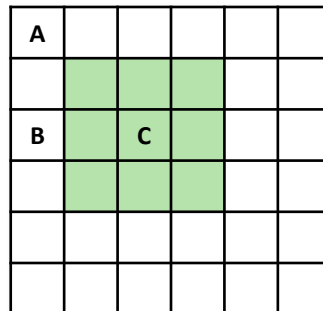
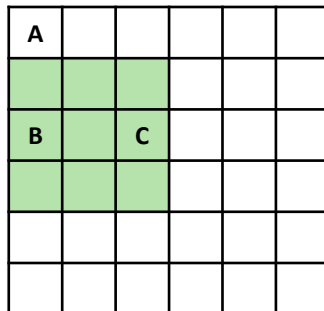
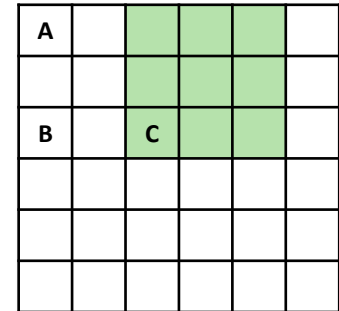
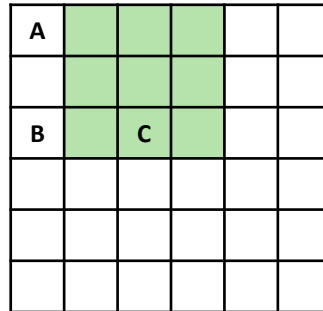
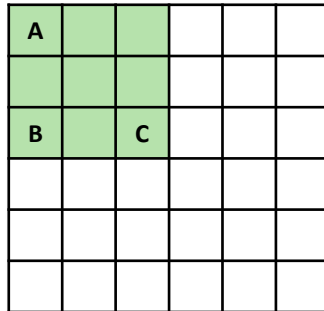
Já no caso do pixel B, podemos observar que este irá passar pelo filtro 3 vezes. Ou seja, terá uma maior importância na aprendizagem da rede neuronal.

A					
B		C			

A					
B		C			

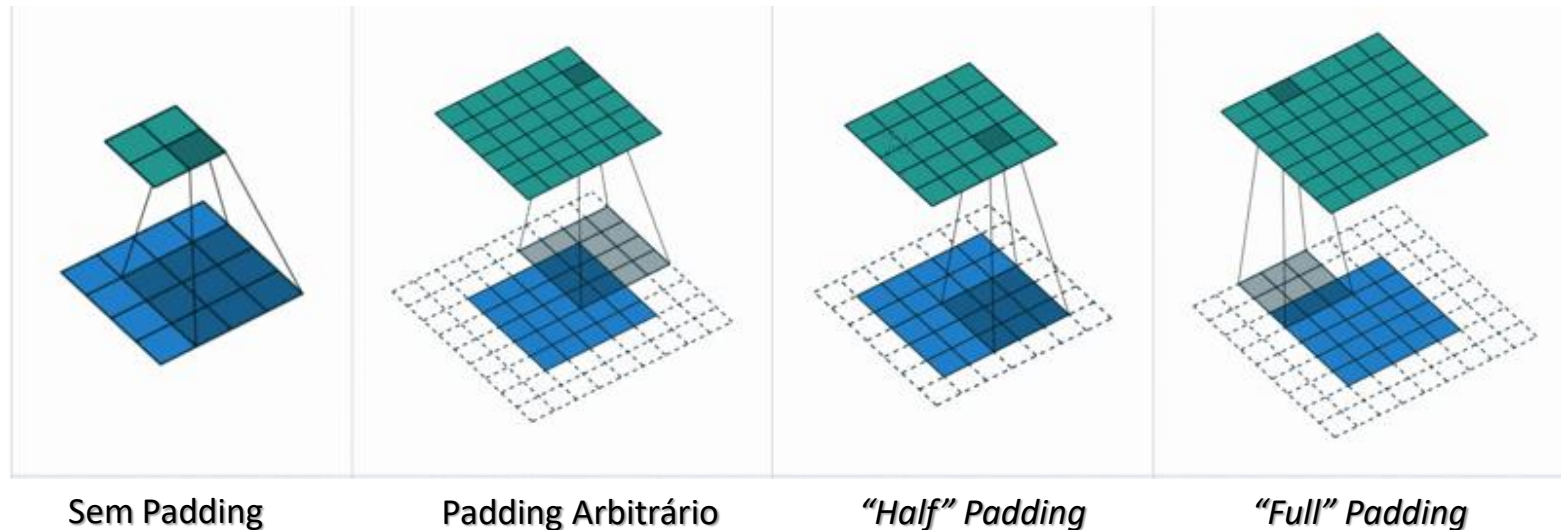
A					
B		C			

Por fim, para, por exemplo, o pixel C, vemos que este irá passar pelo filtro 9 vezes o que significa que o modelo irá funcionar muito bem com o pixel C (visto que é “analisado” diversas vezes ) e interpretará mal o pixel A (pois só é “analisado” 1 vez).



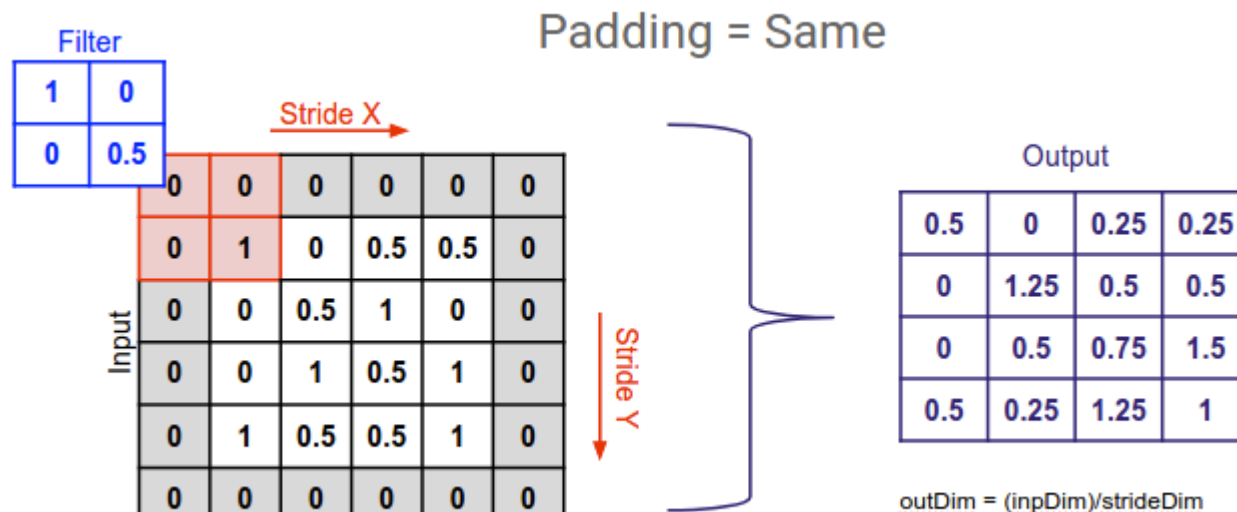
Este problema irá causar a perda de informações disponíveis nos cantos, Além disso, também a saída das camadas será reduzida e as informações reduzidas criarão confusão para as próximas camadas. Este problema do modelo pode ser reduzido pelas camadas de preenchimento (padding).

Estas camadas irão ajudar e permitir que os dados que eram mais “ignorados” pela rede, tenham uma maior importância aquando da aprendizagem.



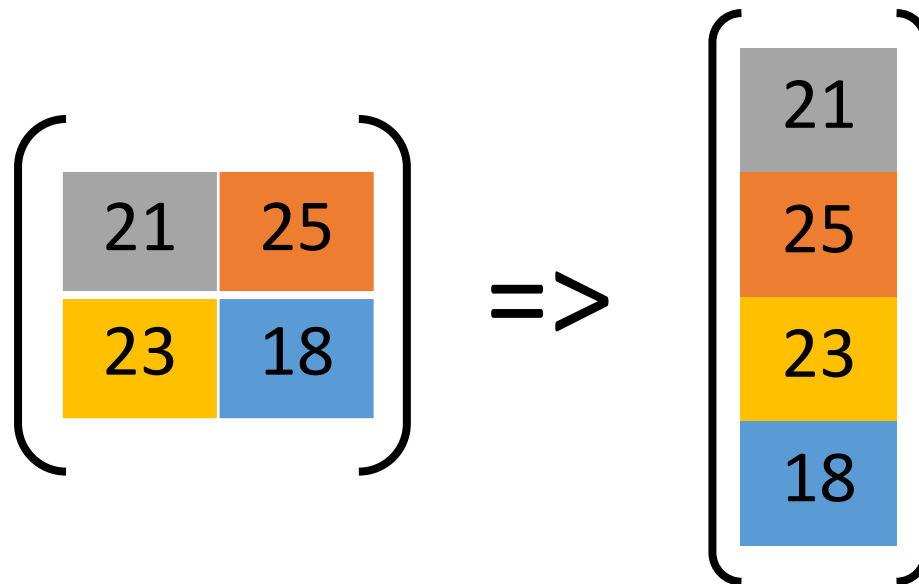
As camadas de preenchimento (padding) são usadas em casos em que queremos aumentar o tamanho da saída e “guarda” as informações apresentadas nos cantos. Estas camadas ajudam adicionando linhas e colunas extras na dimensão externa das imagens. Portanto, o tamanho dos dados de entrada permanecerá semelhante aos dados de saída.

O “padding” é simplesmente um processo de adicionar camadas de zeros às nossas imagens de entrada para evitar os problemas mencionados acima.

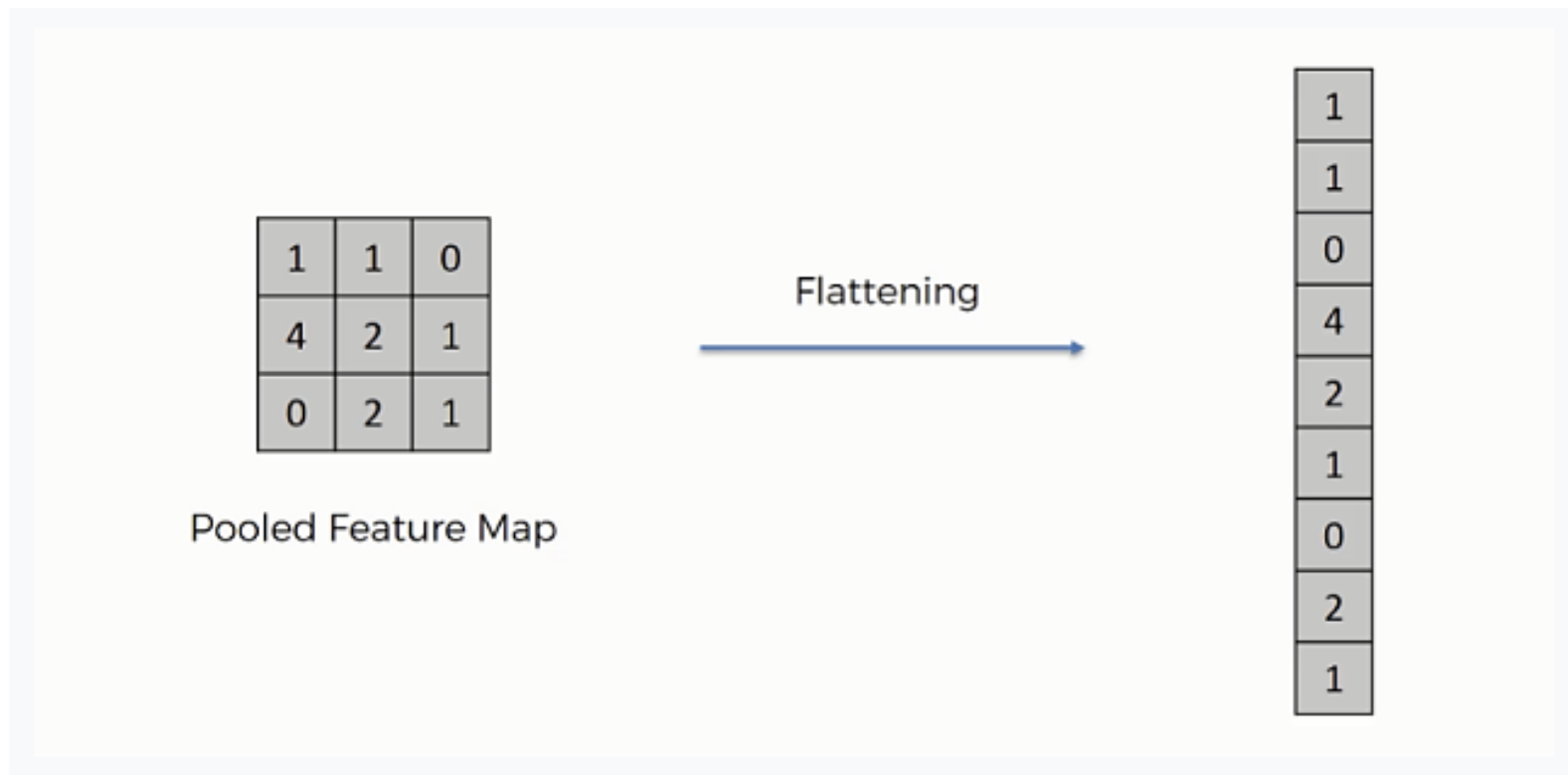


## Operação “Flatten”

De forma simples, os dados finais das últimas duas camadas são transformados numa matriz linear de uma única dimensão.



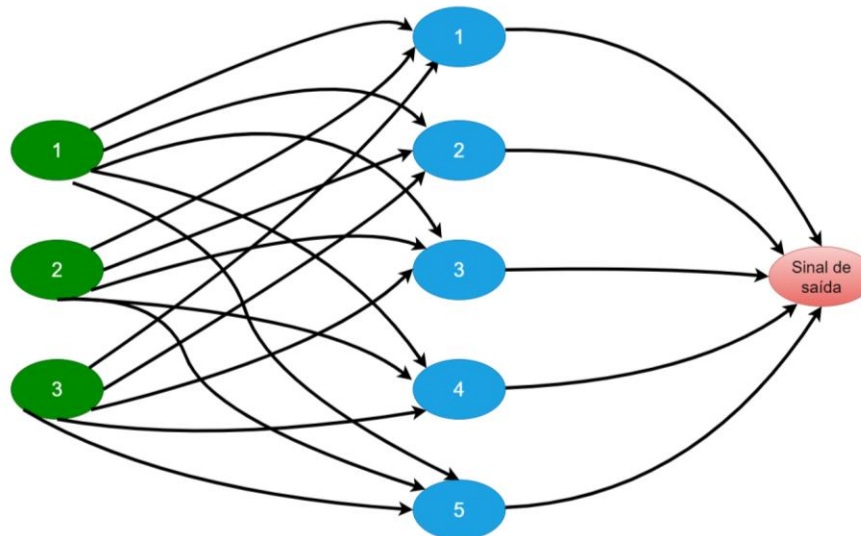
Seja qual for o tamanho da matriz (mapa de recursos) que provém das camadas de pooling, esta irá, como o nome da etapa indica, ser achatada e agrupada em uma coluna como na imagem abaixo.





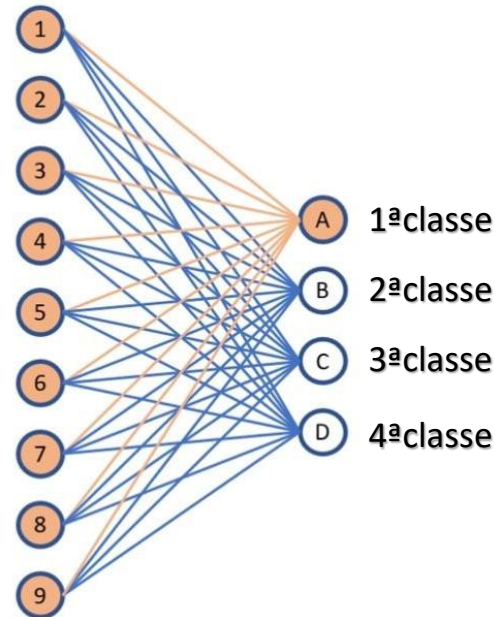
As camadas “Fully Connected”, por vezes chamadas de camadas “Dense”, formam as últimas camadas da rede.

Estas funcionam simplesmente como redes neuronais feedforward. A entrada para a camada “Fully Connected” é a saída da camada Pooling ou Convolucional, que passa pela operação de “Flatten” e, em seguida, serve de entrada nesta última camada.



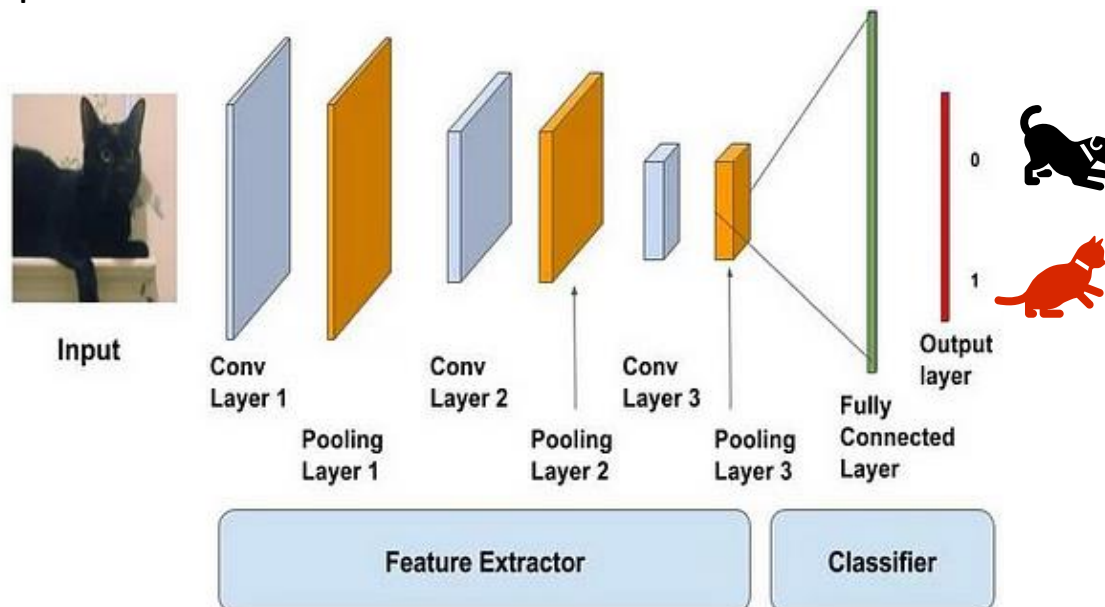
Nota: A imagem é meramente ilustrativa, porque o número de neurónios na camada “Fully Connected” corresponde ao número de classes na tarefa de classificação.

O objetivo da rede neuronal artificial nesta camada é utilizar os dados e combinar os recursos em uma ampla variedade de atributos que tornam a rede convolucional mais capaz de classificar imagens, que é o objetivo da criação de uma rede neuronal convolucional.



Usando o exemplo anterior, mas, desta vez distinguindo apenas gatos e cães, irá, resumidamente, acontecer o seguinte ao longo de toda a rede, com uma imagem de um gato como entrada:

1. Nas camadas Convolucionais e de Pooling são extraídas as características principais do gato (Ex.: Olhos, nariz forma da cabeça, etc.);
2. Na camada Fully-Connected, recebem-se os valores retirados das características e verifica-se se estas pertencem a um gato ou a um cão, sendo enviado o resultado para a Output Layer que determina qual o animal detetado pela rede.



# Capítulo 2.3

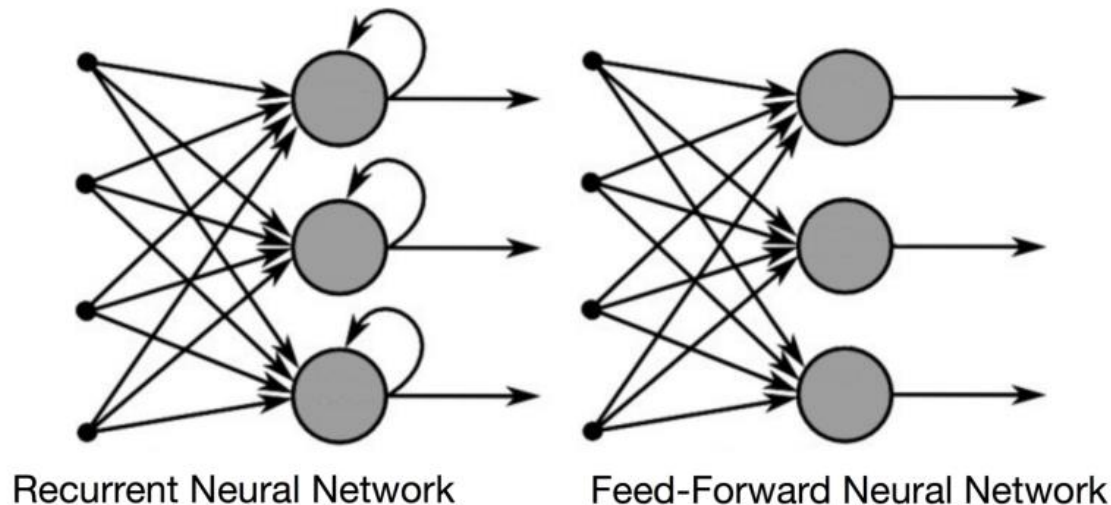
---

Redes neuronais  
recorrentes (RNRs)

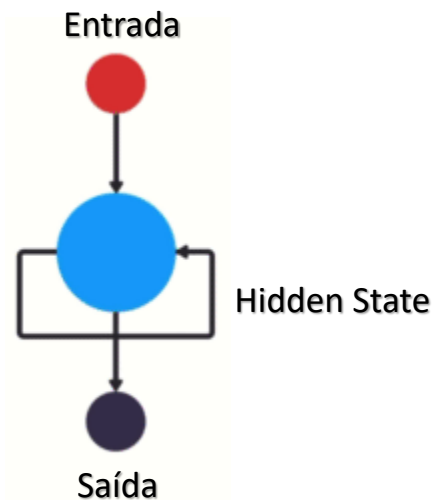
Uma rede neuronal recorrente (RNN) é um tipo de rede neuronal artificial que usa dados sequenciais ou dados de séries temporais. Esses algoritmos de aprendizagem profunda são comumente usados para problemas ordinais ou temporais, como tradução de linguagem, processamento de linguagem natural, reconhecimento de voz e legenda de imagens. Estes são incorporados em aplicações populares, como a Siri, pesquisa por voz e Google Tradutor.



Uma característica distintiva das redes recorrentes é que elas partilham parâmetros entre cada camada da rede. Enquanto as redes feedforward têm pesos diferentes em cada nó, as redes neurais recorrentes partilham o mesmo parâmetro de peso dentro de cada camada da rede. Dito isto, esses pesos continuam a ser ajustados por meio dos processos de retro propagação e descida de gradiente para facilitar o aprendizado.



Como as redes neurais feedforward e convolucionais (RNCs), as redes neurais recorrentes utilizam dados de treino para aprender. Estas no entanto, distinguem-se por possuírem “memória”, pois recebem informações de entradas anteriores para influenciar a entrada e a saída atuais.



Digamos que queremos construir um chatbot, software muito popular hoje em dia. Digamos que o chatbot possa classificar as intenções do texto inserido pelos utilizadores.



Para resolver este problema, primeiro, “codificamos” a sequência de texto usando uma RNN. Em seguida, alimentamos a saída da RNN a uma rede neuronal feedforward que classificará as intenções.

Tendo, por exemplo, o utilizador a digitar “What time is it?”, para começar, dividimos a frase em palavras individuais. O RNN funciona sequencialmente, então passamos para rede apenas uma palavra de cada vez.

What time is it?



O primeiro passo é inserir “What” na RNN. O RNN “codifica” “What” e produz uma saída.

What time is it ?

Para a próxima etapa, inserimos a palavra “tempo” e o “hidden state” da etapa anterior. Assim, a RNN agora tem informações sobre a palavra “What” e “time”.



Repetimos esse processo, até a etapa final. É possível ver na etapa final, que o RNN “utilizou” as informações de todas as palavras nas etapas anteriores.



Como a saída final foi criada a partir do restante da sequência, devemos ser capazes de usar a saída final e passá-la para a camada de feedforward para classificar uma intenção.



É possível reparar numa estranha distribuição de cores nas camadas ocultas. Esta distribuição representa um problema com as RNRs conhecido como memória de curto prazo.



“Estado” final das Camadas Ocultas

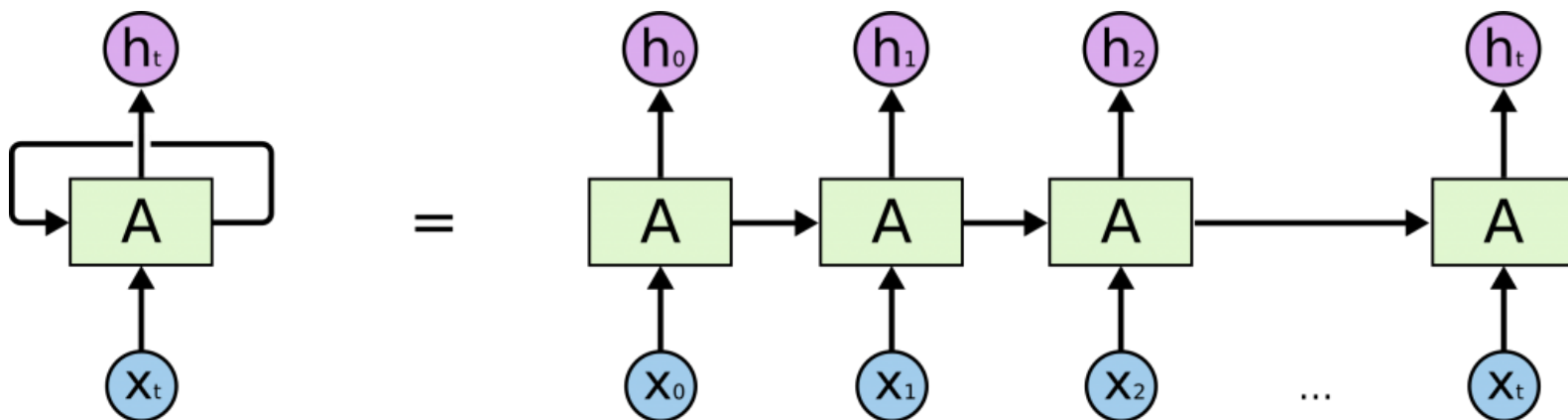
A memória de curto prazo é causada por um problema de desaparecimento do gradiente, problema que também prevalece em outras arquiteturas de redes neuronais.

À medida que o RNN processa mais etapas, ele tem problemas para reter as informações das etapas anteriores. Como é possível verificar, a informação da palavra “**what**” e “**time**” é quase inexistente na etapa final.

A rede neuronal LSTM funciona de forma muito semelhante às redes neurais biológicas (cérebro humano), sendo possível dividir em dois tipos de memórias dos quais são:

- Memórias de longo prazo é a informação que é armazenada que pode ser depois recuperada.
- Memória de curto prazo é a informação que é retida por alguns segundo e depois descartada.

Assim, não temos a problema de esquecimento de alguma informação mais importante ao passar de camada a camada.



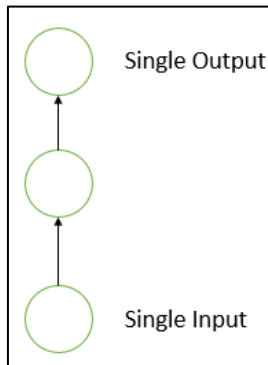
Durante o processo, as RNNs tendem a deparar-se com dois problemas, conhecidos como gradientes explosivos e gradientes que desaparecem. Essas questões são definidas pelo tamanho do gradiente. Quando o gradiente é muito pequeno, este continua a diminuir, atualizando os parâmetros de peso até que se tornem insignificantes, ou seja, 0. Quando isso ocorre, o algoritmo deixa de aprender.

Gradiente pequeno  $\longrightarrow$  Pesos  $\Downarrow\Downarrow$   $\longrightarrow$  Pesos Finais  $\approx$  0

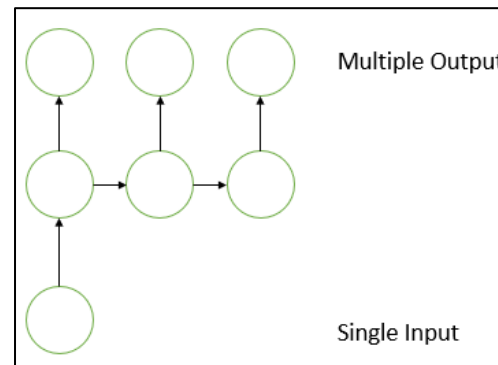
Os gradientes explosivos ocorrem quando o gradiente é muito grande, criando um modelo instável. Neste caso, os pesos do modelo tornam-se muito grandes e, eventualmente, serão representados como NaN (Not a Number). Uma solução para esses problemas é reduzir o número de camadas ocultas na rede neuronal, eliminando parte da complexidade do modelo RNN.

Gradiente grande  $\longrightarrow$  Pesos  $\uparrow\uparrow$   $\longrightarrow$  Pesos Finais  $\approx \infty$  (NaN)

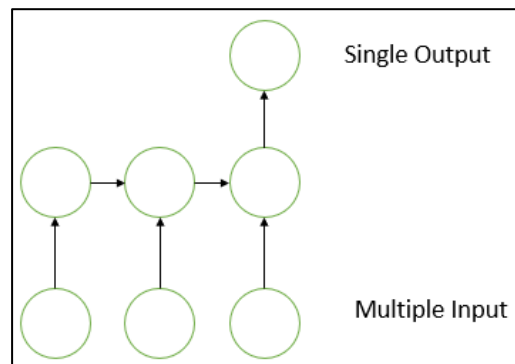
Como sabemos, nas redes feedforward temos uma entrada para uma saída e, embora tenhamos visualizado as redes neurais recorrentes dessa maneira nos diagramas acima, na verdade elas não têm essa restrição. Em vez disso, suas entradas e saídas podem variar em tamanho, e diferentes tipos de RNNs são usados para diferentes casos de uso, como criação de música ou tradução automática.



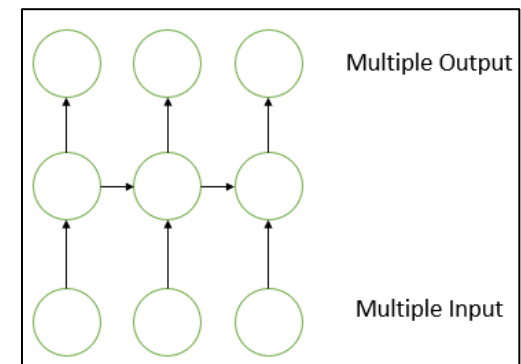
**One-to-One**



**One-to-Many**

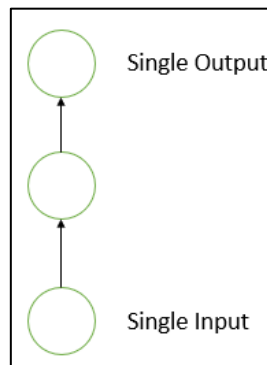


**Many-to-One**

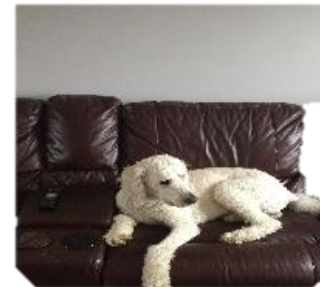
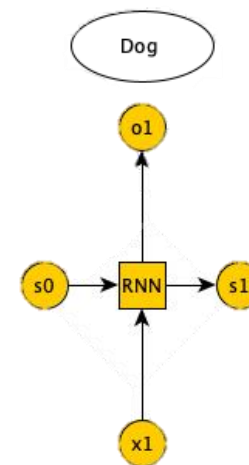


**Many-to-Many**

No primeiro caso, “One-to-One”, como podemos observar, temos apenas uma entrada para uma única saída. Um exemplo de onde este tipo de RNNs são usadas é a classificação de imagens.

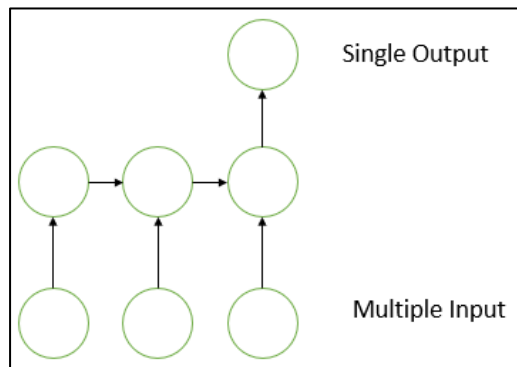


One-to-One





No tipo “Many-to-One”, iremos ter diversas entradas para uma única saída. A “Análise de Sentimentos” é um exemplo comum deste tipo de RNNs, onde é feita a análise de texto digital para determinar se o tom emocional da mensagem é positivo, negativo ou neutro.



Many-to-One



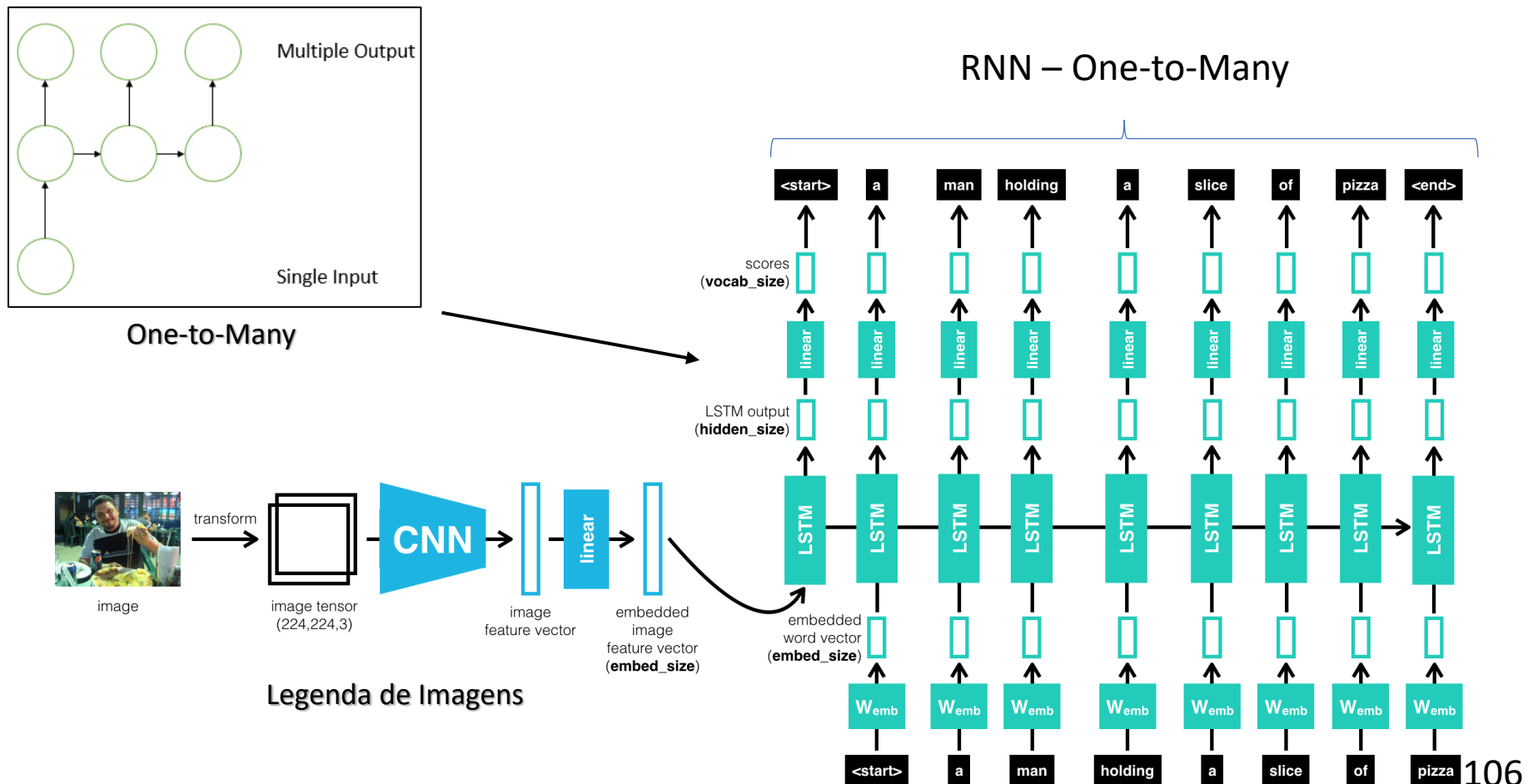
O meu número de telefone não está a funcionar.



O atendimento ao cliente é muito bom.

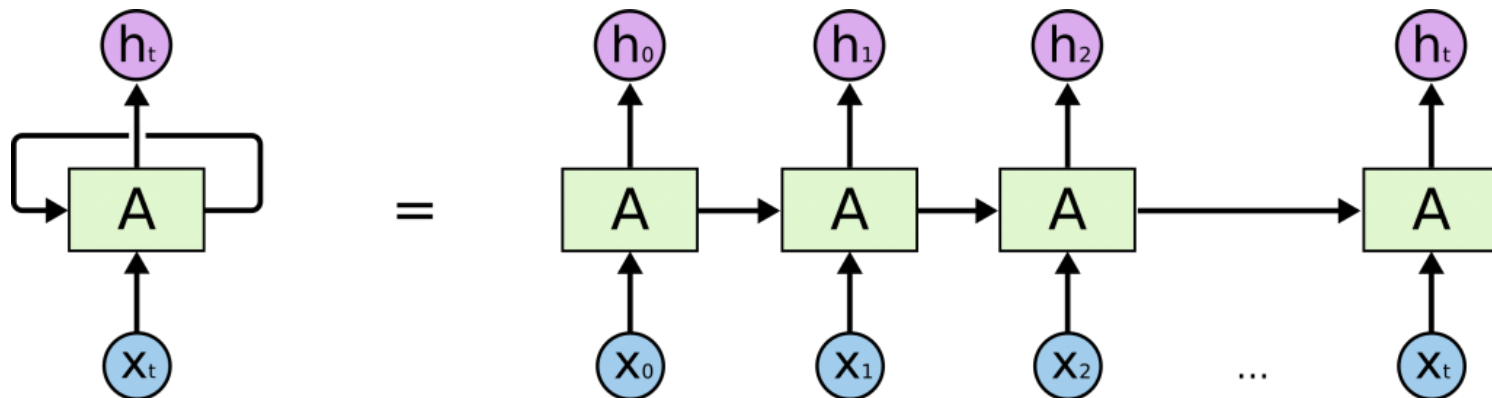


No terceiro tipo de RNNs, chamado “One-to-Many”, observa-se que existem apenas uma entrada, no entanto temos diversas saídas, ou seja, inversa à anterior. Este tipo é utilizado nomeadamente para legendar de imagens e gerar músicas.

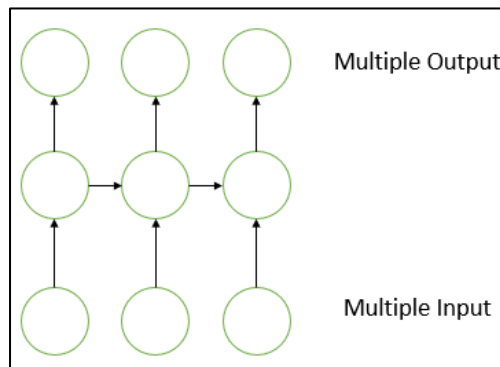


Um problema das RNRs é que ao longo do seu processo, passando de percepção para percepção, elas vão perdendo algumas informações.

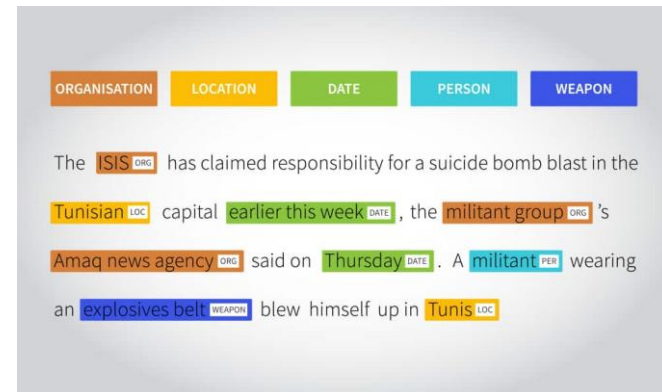
As redes LSTM, no entanto, não têm esse problema, porque possuem memórias de curto e longo prazo, tornando-se uma rede muito boa para lidar com problemas de sequência.



Por último, as RNNs “Many-to-Many”, como se pode perceber, possuem múltiplas entradas assim como saídas, podendo ter o mesmo número ou não. Este tipo é utilizado nomeadamente para criar máquinas de tradução ou para reconhecimento de entidades (como nomes de pessoas, organizações, locais, etc).



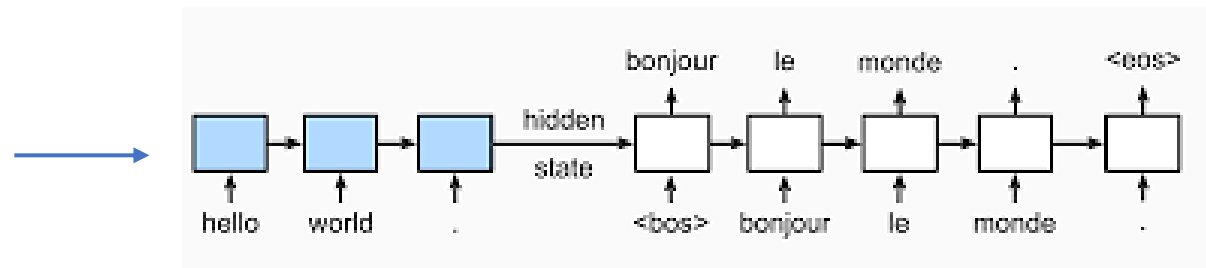
Many-to-Many



Reconhecimento de entidades



Tradução

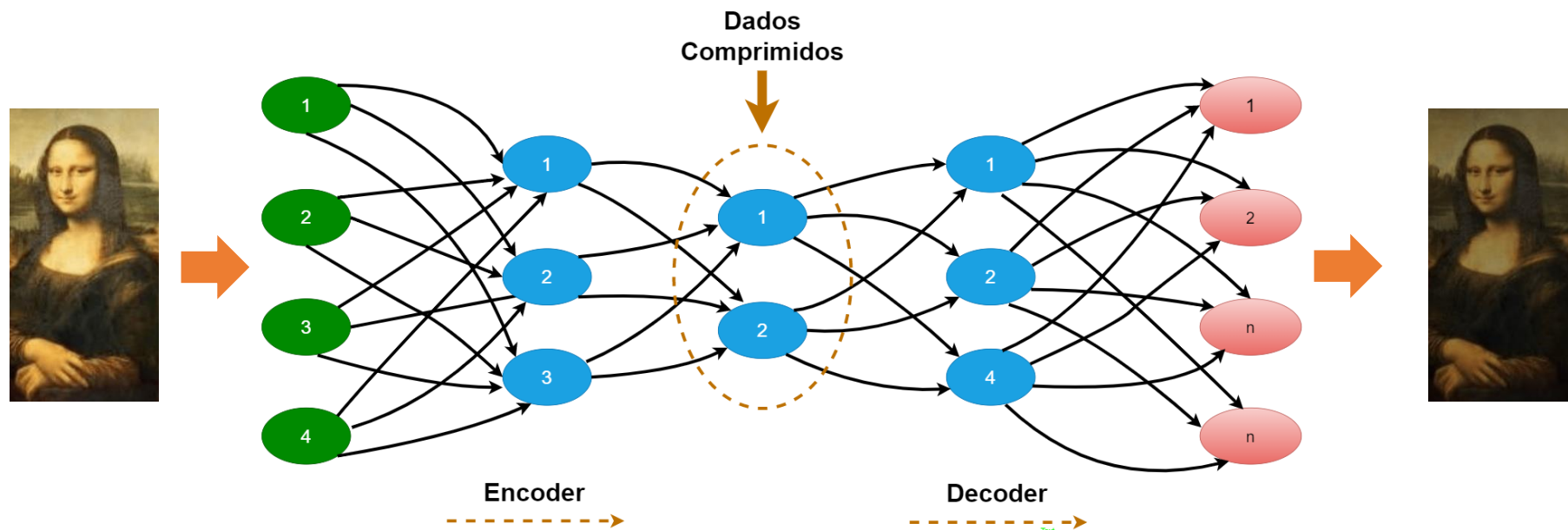


# Capítulo 2.4

---

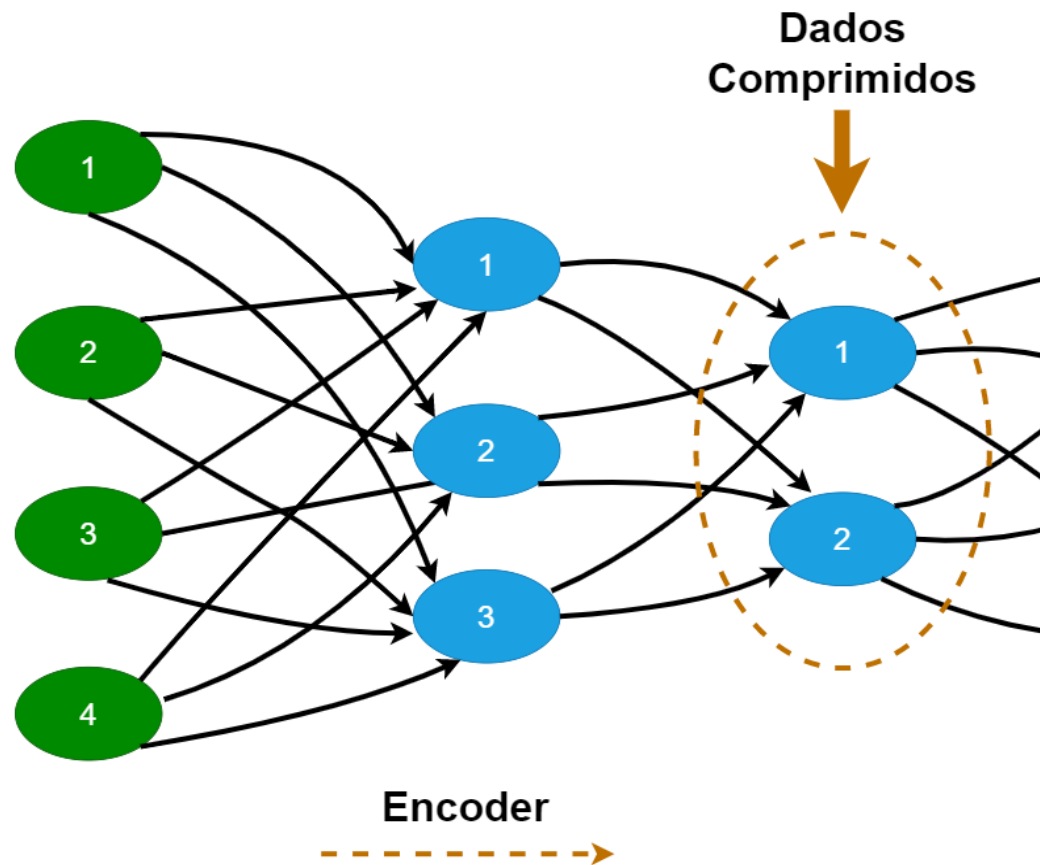
## Redes Neurais Autoencoder

As redes neurais autoencoders têm como objetivo reduzir os dados recebidos, processo a que se dá o nome de “encoder” (codificador), e de seguida reconstrói os dados que foram recebidos, processo de nome “decoder” (decodificador).



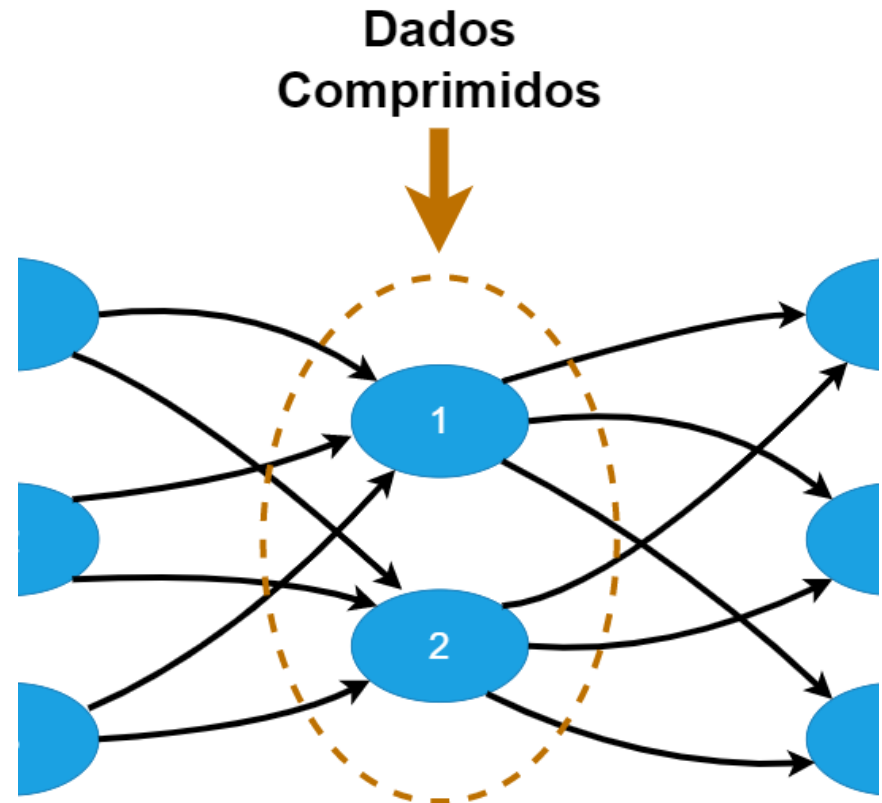
## Encoder

O Encoder é composto por blocos convolucionais que reduzem a dimensão da camada de entrada em uma secção compacta, a área a tracejado (Dados Comprimidos). Com o objetivo de capturar as partes mais importantes da camada de entrada.



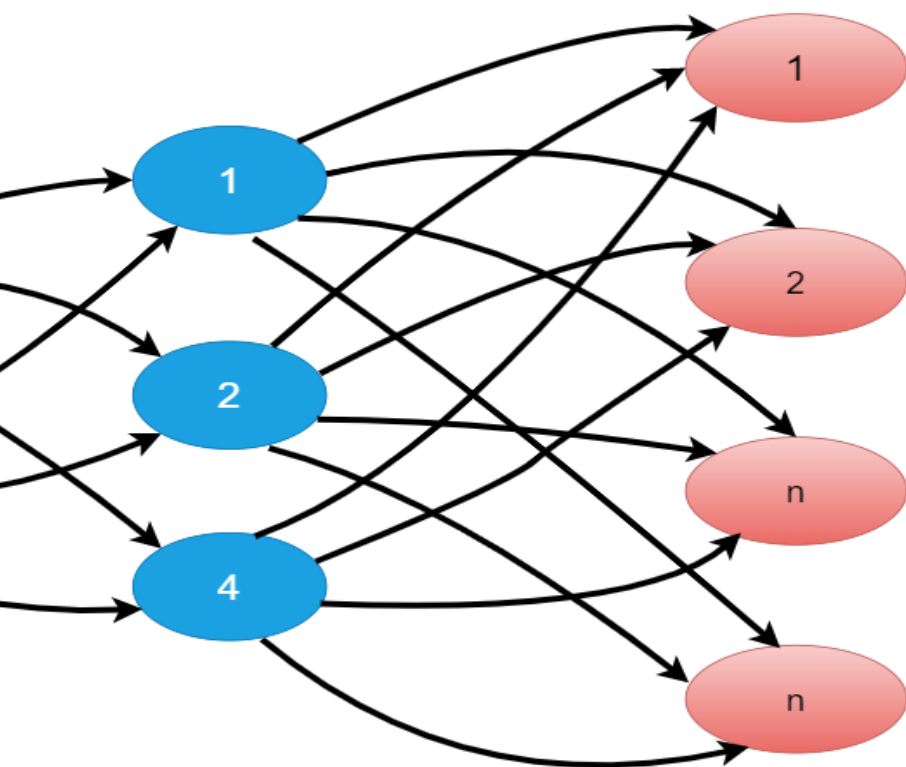
## Dados Comprimidos

A área a tracejado, a que se dá o nome de dados comprimidos, é onde se realiza a filtração da entrada obtendo-se uma versão reduzida desta. Com isso evitamos o overfitting, por outro lado poderá haver perda de alguma informação. Por isso esta etapa é a mais importante da rede neuronal.





## Decoder



**Decoder**

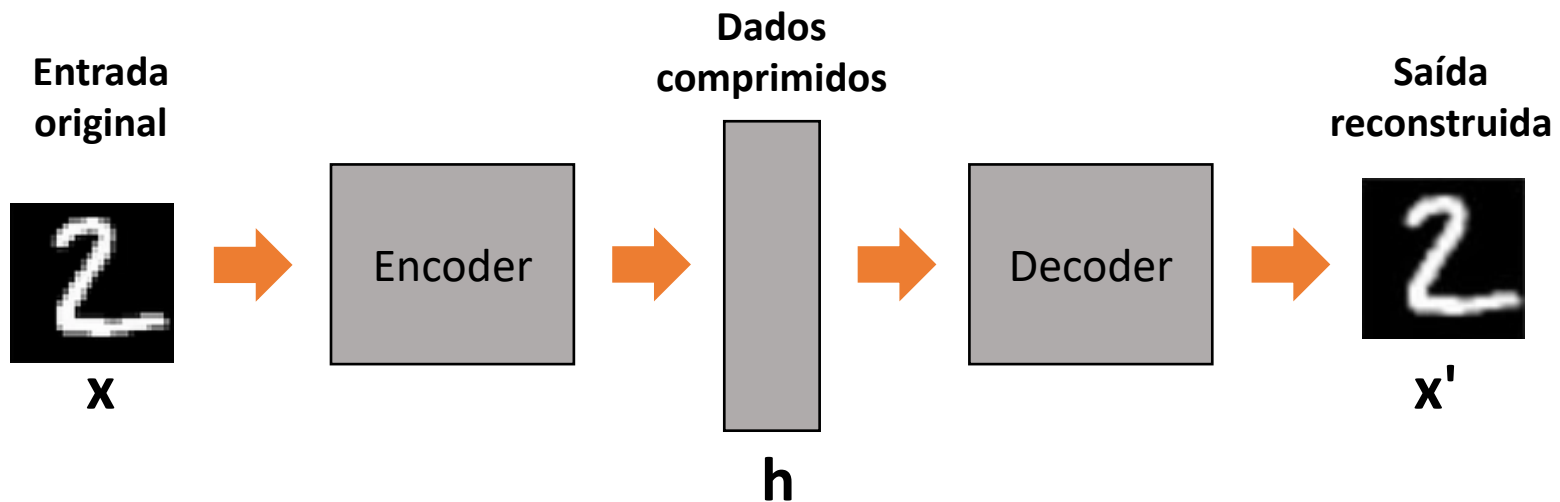


O decoder é a parte final da rede neuronal.

Este corresponde a um conjunto de blocos com a função de aumentar a resolução dos dados e reconstruí-los a partir dos dados comprimidos.

O encoder é onde se comprime a informação é representado pela função  $x ( f(x)= h )$ .

O decoder é onde restaura a informação original, esta ação é a inversa do encoder que é representada pela função  $h ( g(h)=x' )$ .



O autoencoder pode ser descrito da seguinte função:

$$g( f(x) ) = x'$$

Outros tipos de rede neuronais que utilizam “autoencoders”:

- Codificadores automáticos incompletos
- Autoencoders esparsos
- Codificadores automáticos contrativos
- Denoising autoencoders
- Autoencoders variacionais (para modelagem generativa)

<https://www.jeremyjordan.me/variational-autoencoders/>

<https://devpost.com/software/animation-autoencoder-wp0cdt>

<https://iq.opengenus.org/applications-of-autoencoders/>

<https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/>

<https://www.baeldung.com/cs/learning-rate-batch-size>

<https://analyticsindiamag.com/most-common-activation-functions-in-neural-networks-and-rationale-behind-it/>

<http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

<https://hackernoon.com/exploding-and-vanishing-gradient-problem-math-behind-the-truth-6bd008df6e25>

<https://www.deeplearningbook.com.br/redes-neurais-recorrentes/>

<https://viceri.com.br/insights/arquiteturas-de-redes-neurais-convolucionais-para-reconhecimento-de-imagens/>

[https://bdm.unb.br/bitstream/10483/31168/1/2021\\_LeonardoAlmeida\\_MarcusViniciusBorges\\_tcc.pdf](https://bdm.unb.br/bitstream/10483/31168/1/2021_LeonardoAlmeida_MarcusViniciusBorges_tcc.pdf)

<https://imasters.com.br/data/um-mergulho-profundo-nas-redes-neurais-recorrentes>

<https://insightlab.ufc.br/aprenda-a-criar-e-treinar-uma-rede-neural-convolucional-cnn>

<https://www.gormanalysis.com/blog/neural-networks-for-your-dog-1-1-introduction/>

<https://www.spicelogic.com/Blog/Perceptron-Artificial-Neural-Networks-10>