
Tera

10/dezembro/2019

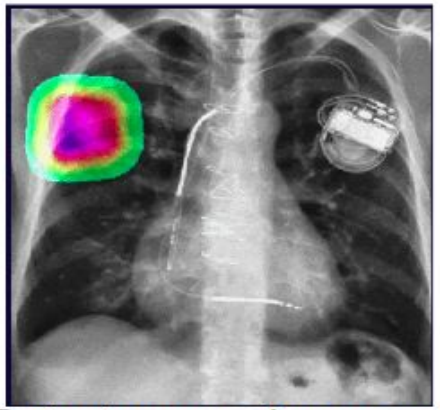
Deep Learning

Jéssica dos Santos @j3ssicaSant0s

Aplicações de Deep Learning?

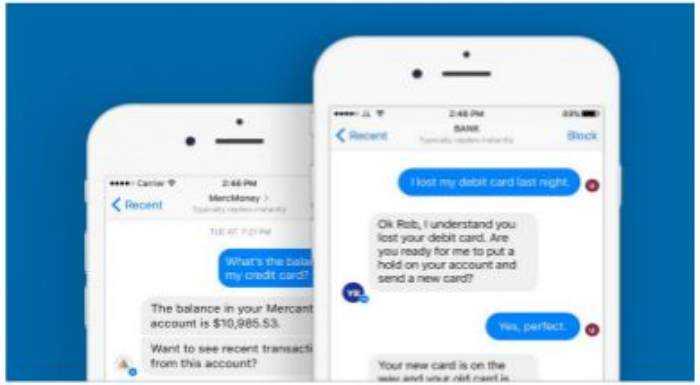
Calcular preço do imóvel (regressão)

Predizer se um crédito será aprovado (classificação)



Detectar uma doença em um exame (classificação de imagens)

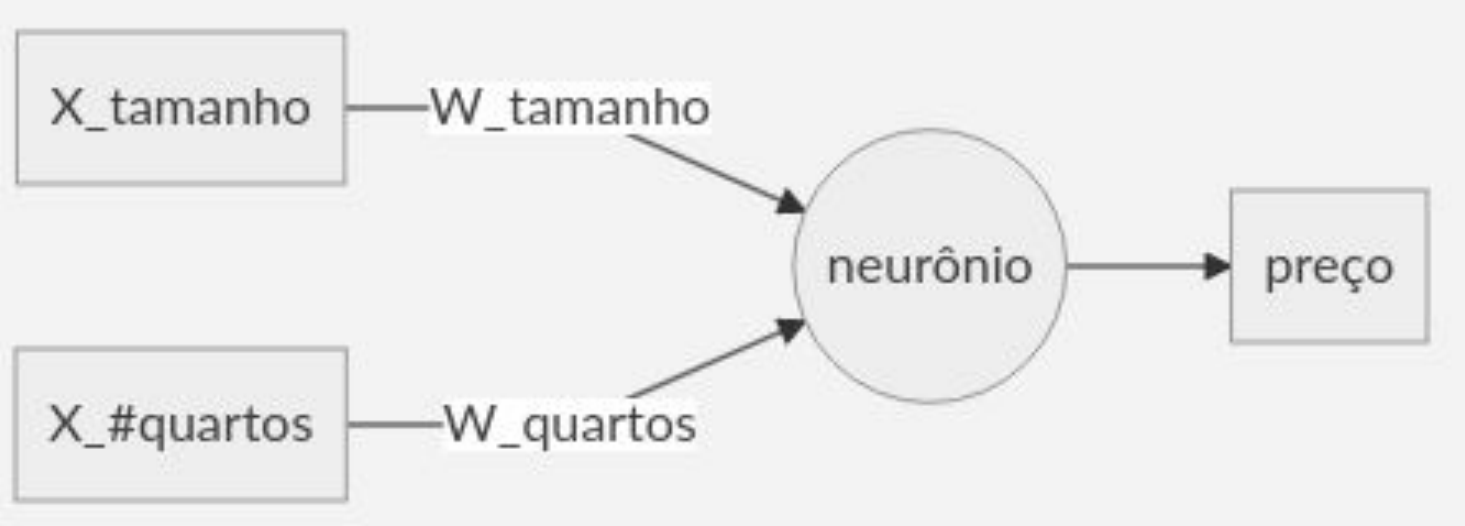
Predizer aumento de vendas (análise temporal)



Criar um chatbot de atendimento (processamento de linguagem natural)

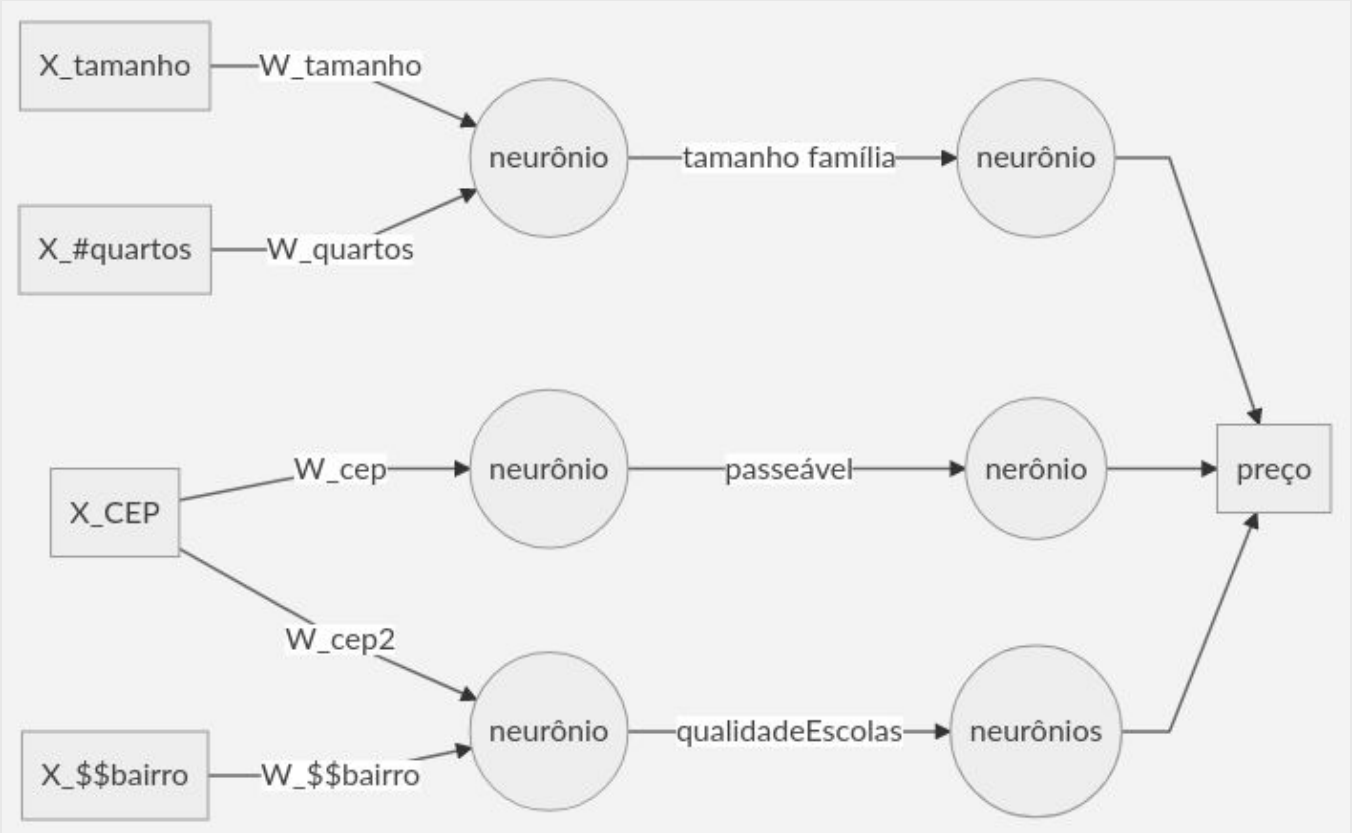
O que é uma Rede Neural?

Uma rede neural simples pode ser comparada a uma função de regressão linear. Como exemplo, se quisermos predizer o valor de uma casa baseado no tamanho e no número de quartos da mesma, pode-se construir uma rede neural simples com apenas um neurônio, que recebe tamanho e # quartos como entrada, computa através de uma função linear e retorna como output o preço:



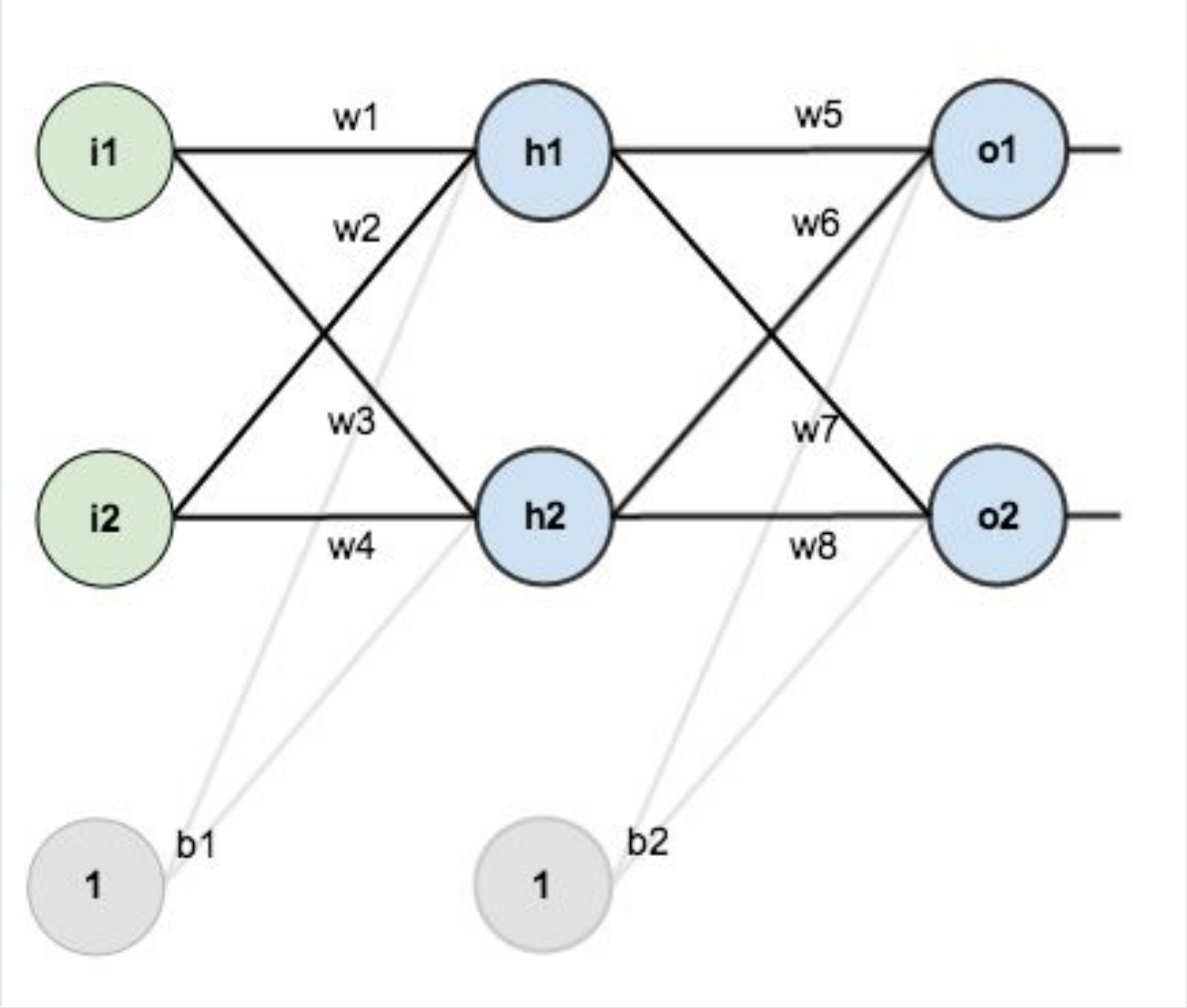
O que é uma Rede Neural?

Uma rede mais complexa seria como o exemplo abaixo adicionando mais valores de entrada e neurônios na camada intermediária (chamada de hidden Layer ou camada oculta):



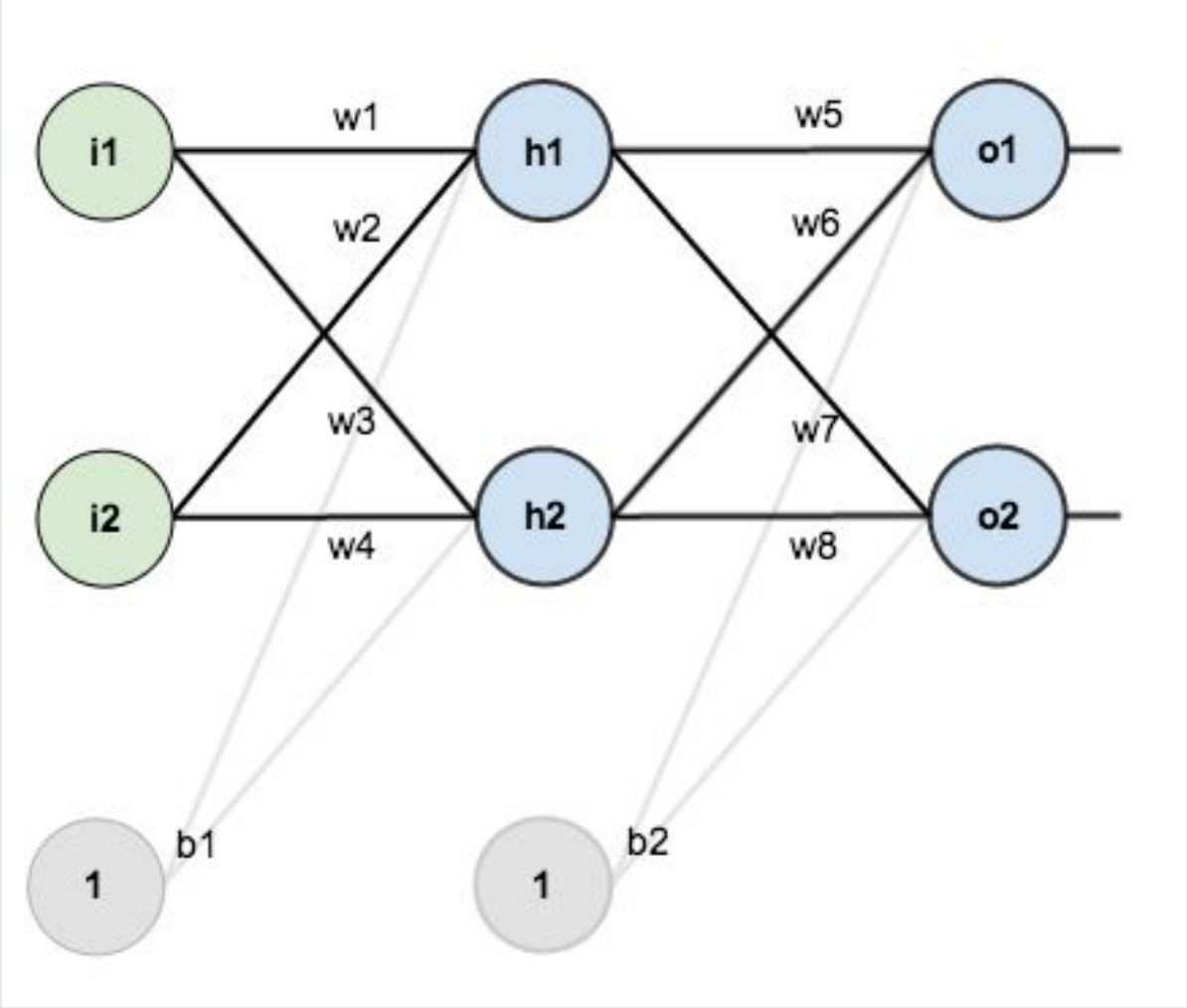
O que é uma Rede Neural?

Na prática precisamos apenas definir as variáveis de entrada (X) e dizer qual a saída para cada exemplo. Todos os neurônios de entrada são ligados aos neurônios da camada oculta. A relação entre as variáveis, ou seja, os pesos, é o que a rede vai aprender.



O que é uma Rede Neural?

Nessa figura temos 2 neurônios na camada de entrada (2 variáveis de entrada), 2 na camada oculta e 2 na camada de saída (2 classes como resultado). Essa rede irá aprender 8 pesos (w*) e 4 bias (b*)



“As Redes Neurais Artificiais são baseadas na biologia, tendo como unidade principal o neurônio artificial, que simula o comportamento do neurônio biológico. No modelo computacional de um neurônio, os sinais interagem entre os neurônios, de acordo com o peso dado à relação entre eles (ou seja, cada aresta ligando um neurônio ao outro possui um peso w). A ideia é que os pesos sejam aprendidos e controlem a força de influência de um neurônio em outro. Essa interação é modelada por uma função, que geralmente assume a forma de uma soma ponderada.”

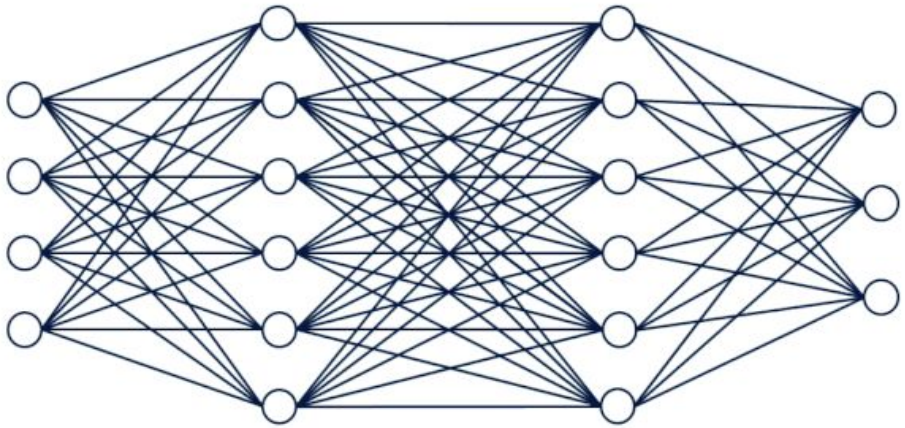
Vamos à prática - Construindo o modelo:

<Notebook>

Função de Ativação

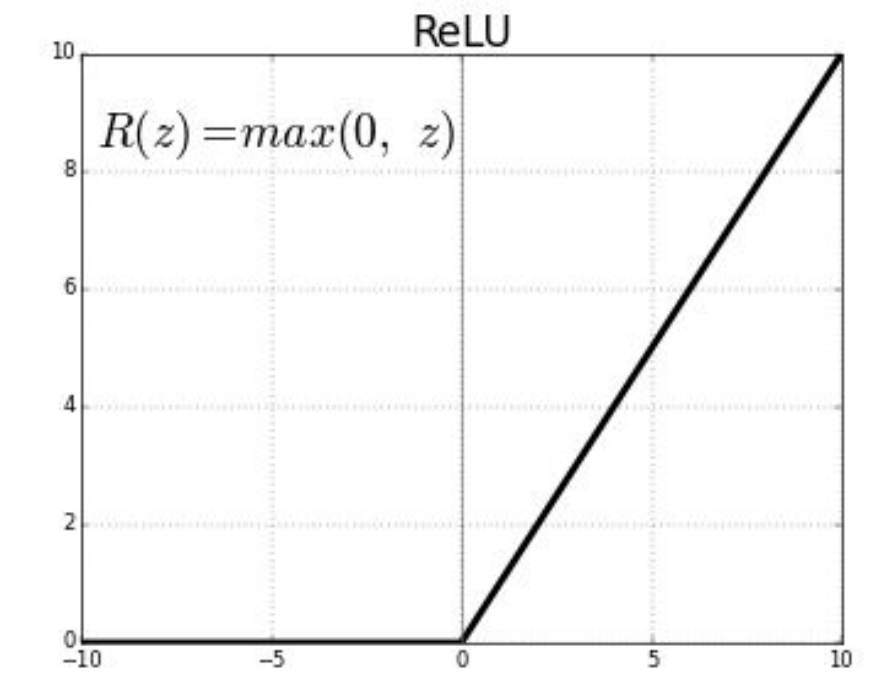
Em cada neurônio da rede há uma função de ativação, que decide se o neurônio deve ser **ativado**, e transmitir informações para a próxima camada.

$$Y = \text{Activation}(\Sigma(weight * input) + bias)$$



Função de Ativação

A função mais comum nas camadas intermediárias é a **ReLU**

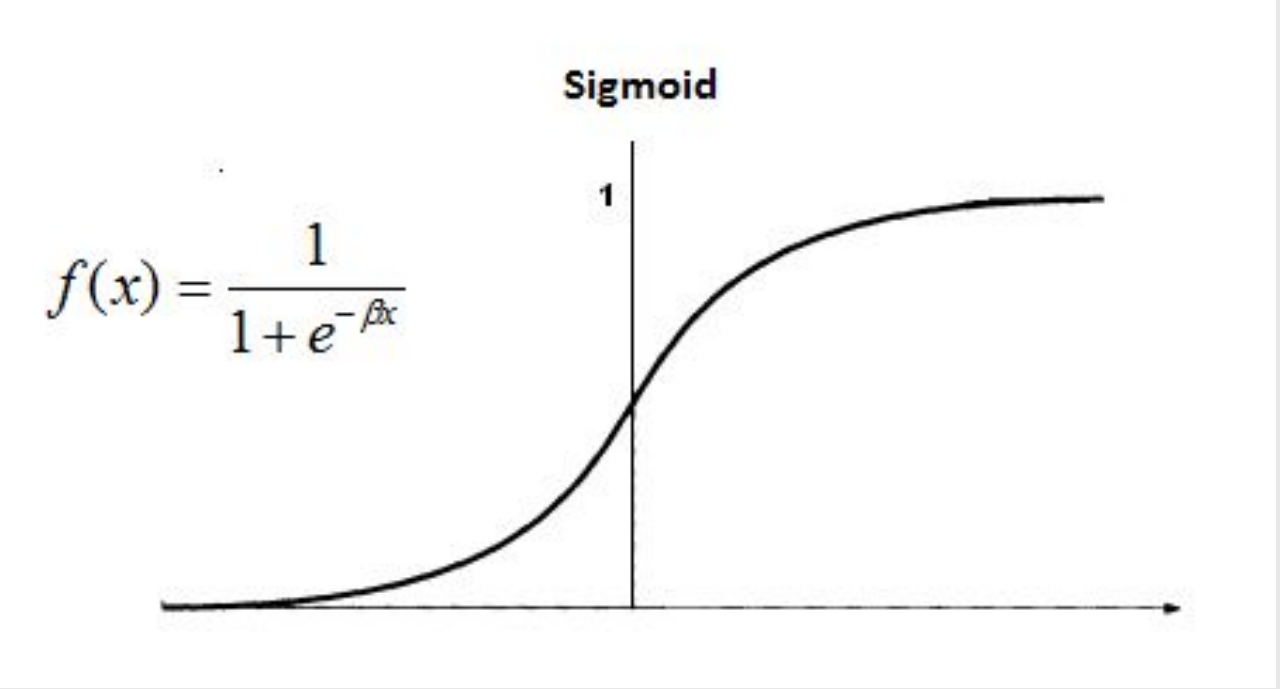


*Basicamente o neurônio é ativado se for maior do que 0. Há também a **elu** que aplica uma exponencial nos valores negativos*

Função de Ativação

Na camada de saída a rede precisa nos retornar a probabilidade.

Por ser uma probabilidade (de 0 a 1), nós usamos a função **sigmoid**:



*Há também a **softmax**, usada na camada de output para problemas de multiclasse, a soma das probabilidades de todas as classes dará 1.*

Função de Ativação

Observações:

1- Modelos regressivos não possuem função de ativação na camada de saída

2 - Essas funções são as mais comuns e mais utilizadas, porém existem outras e outras podem ser criadas

Mais funções de ativação disponíveis no Keras: https://keras.io/activations/

Explicações extras: http://deeplearningbook.com.br/funcao-de-ativacao/

Vamos à prática - Construindo o modelo: <Notebook>

Aprendizado da Rede:

Para aprender os parâmetros *w* e *b* é preciso uma **função de custo**. Primeiro, vamos definir uma função de perda ou **Loss Function** de modo que **quanto mais próximo da resposta certa, menor seja o valor dessa função**:

$$L(\hat{y}, y) = -(y \log \hat{y} + (1 - y) \log (1 - \hat{y}))$$ (binary_crossentropy)

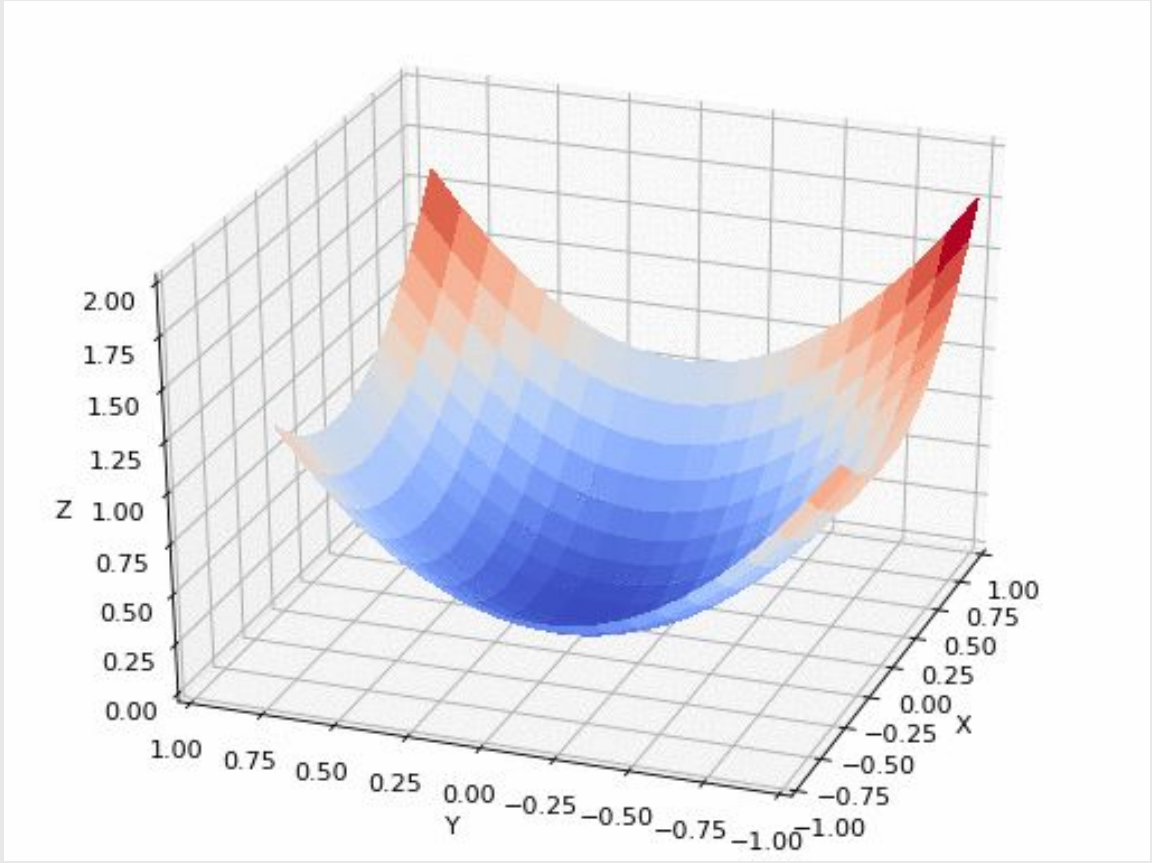
Se uma instância tem label 1, então $(1-y)$ *é* 0 *, deixando apenas o lado esquerdo da equação. Pra que ele seja o menor possível,* $\hat{y}$ *precisa ser o maior possível, no caso o mais próximo de 1. O oposto também se aplica para quando o label é 0.*

Aprendizado da Rede:

Com isso temos a função de Custo:

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^i, y^i)$$

Dado nosso custo, queremos encontrar w e b que minimize esse custo. Para isso utilizamos o **Gradiente Descendente.** A função de custo é uma função convexa, como uma bacia, então o que o gradiente faz é ir descendo o mais rápido possível até chegar no fundo da bacia, no menor ponto, independente do ponto inicial.

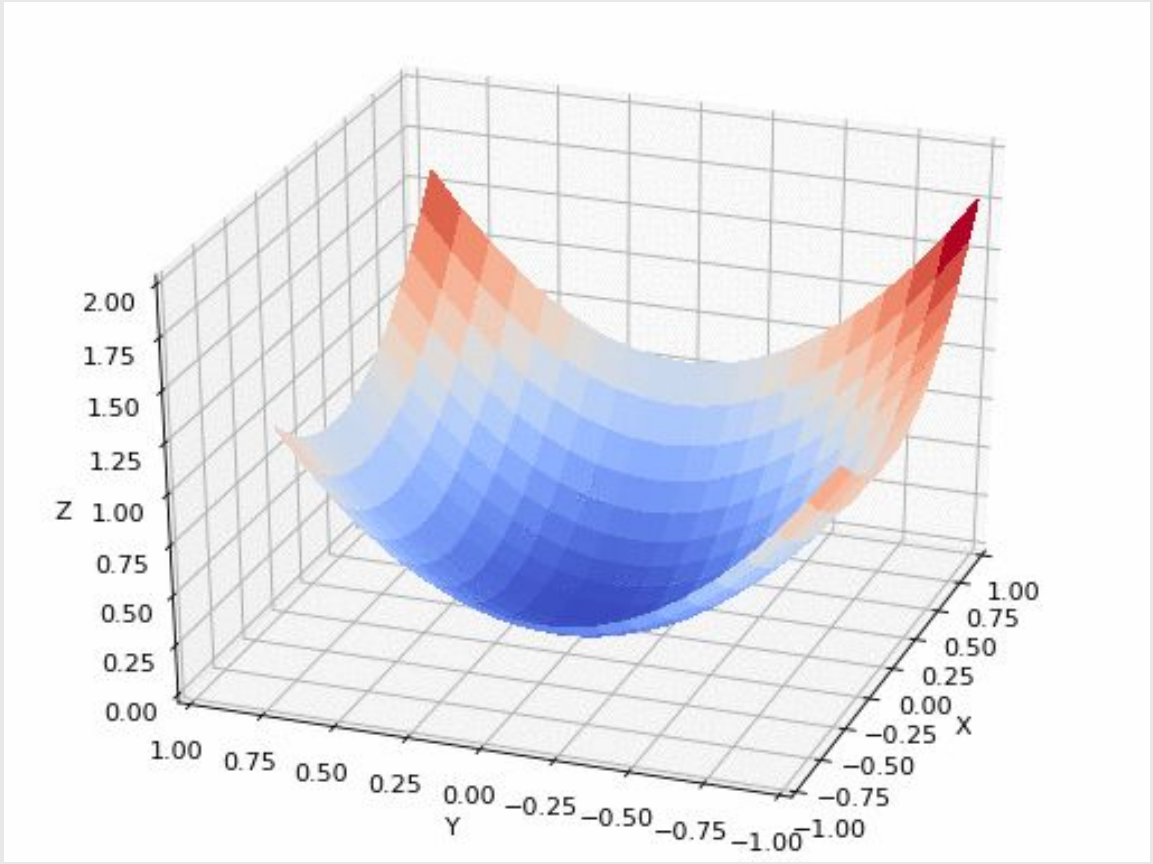


Aprendizado da Rede:

Para fazer essa "descida", utilizaremos a **derivada do custo** e uma taxa de aprendizado ou **learning rate**, da seguinte forma:

A cada iteração do algoritmo temos , sendo α o learning rate.

De modo geral, atualizamos w e b a cada iteração, sendo a velocidade controlada pelo learning rate, até chegarmos no ponto mínimo de custo.



Aprendizado da Rede:

Otimizadores:

SGD: algoritmo de gradient padrão (geralmente utilizado com **Momentum:** ao invés de usar só o gradiente do passo atual para calcular a direção da curva, olha também para alguns passos anteriores)

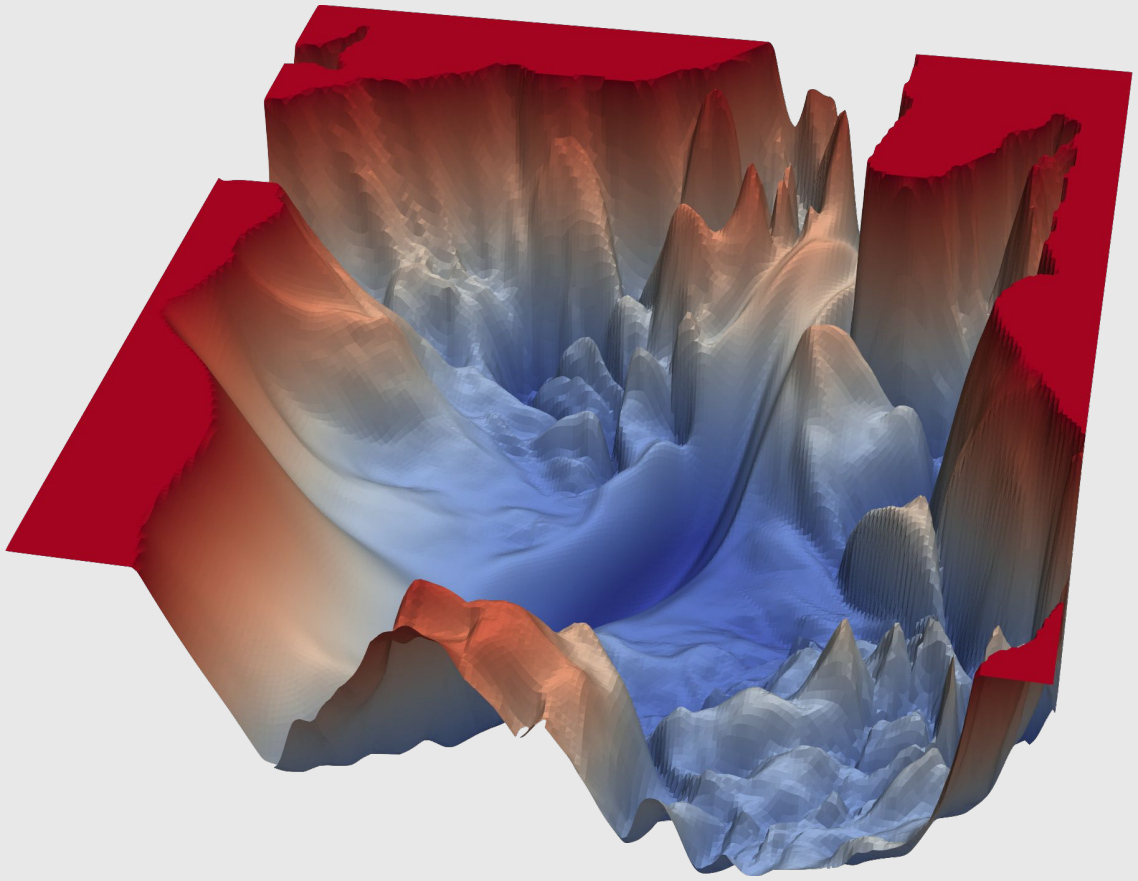
Adam: Adam pode ser visto como uma combinação de RMSprop e SGD com momentum. Ele usa os gradientes quadrados para escalar o learning rate (RMSprop) e aproveita o momentum usando a média móvel do gradiente em vez do próprio gradiente.

Mais info: https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/

Aprendizado da Rede:

Observações:

A função nunca é uma bacia perfeita, está mais para um “vale”. Por isso mais do que o método de otimização é importante prestar atenção no **learning rate.** Se muito alto é possível ficar preso em um mínimo local, se muito baixo pode ficar muito lento.



Vamos à prática - Construindo o modelo: <Notebook>

Treinamento da Rede:

- **Número de épocas**: Quantas vezes a rede vai passar por todos as instâncias

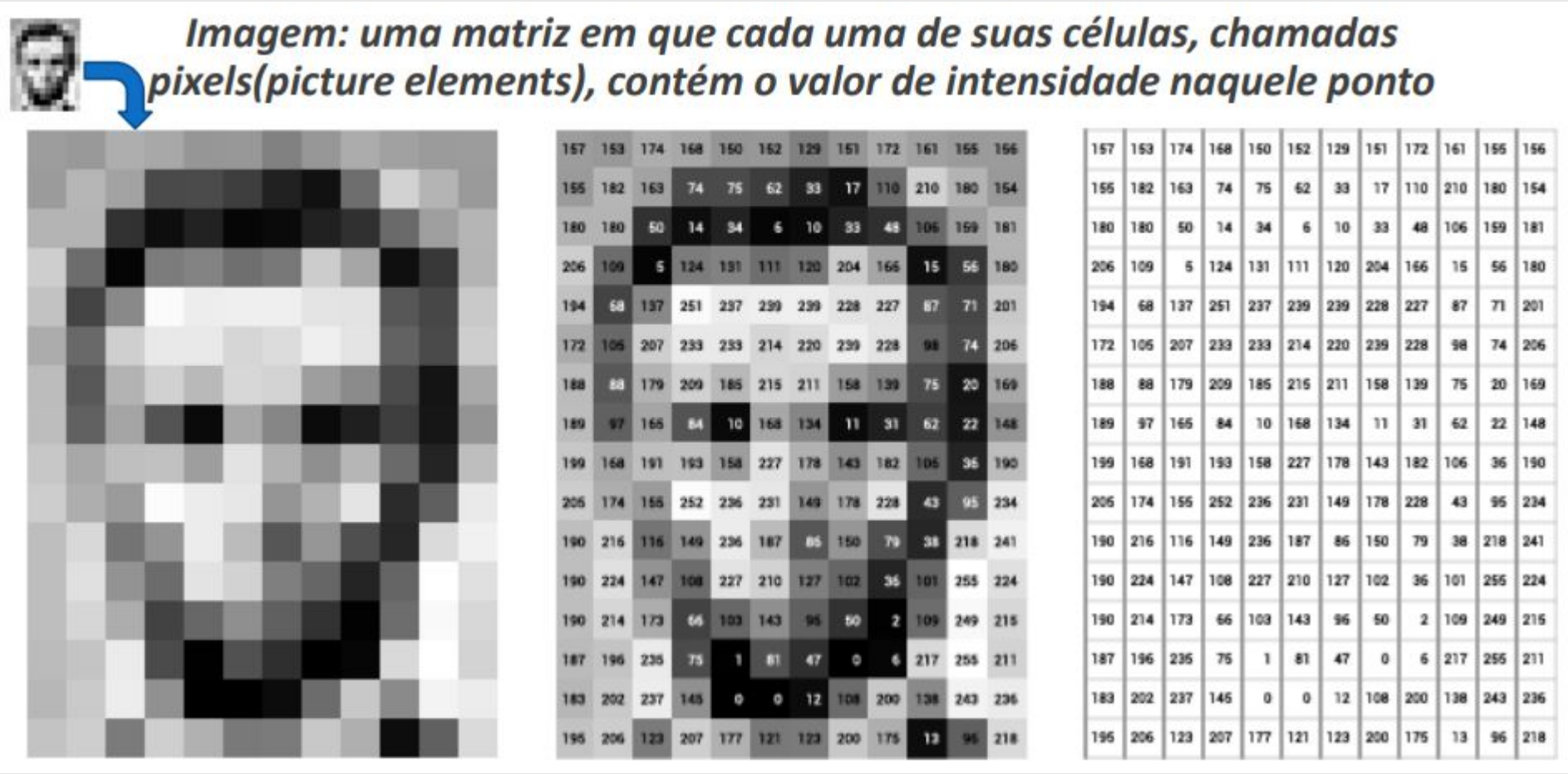
- **Tamanho do batch**: Qual o tamanho do bloco que ela vai usar, ou seja, quantas instâncias por vez passarão pela rede

Redes Convolucionais

CNN (Convolutional Neural Network)

Redes convolucionais são usadas principalmente para classificação de imagens.

*imagem:

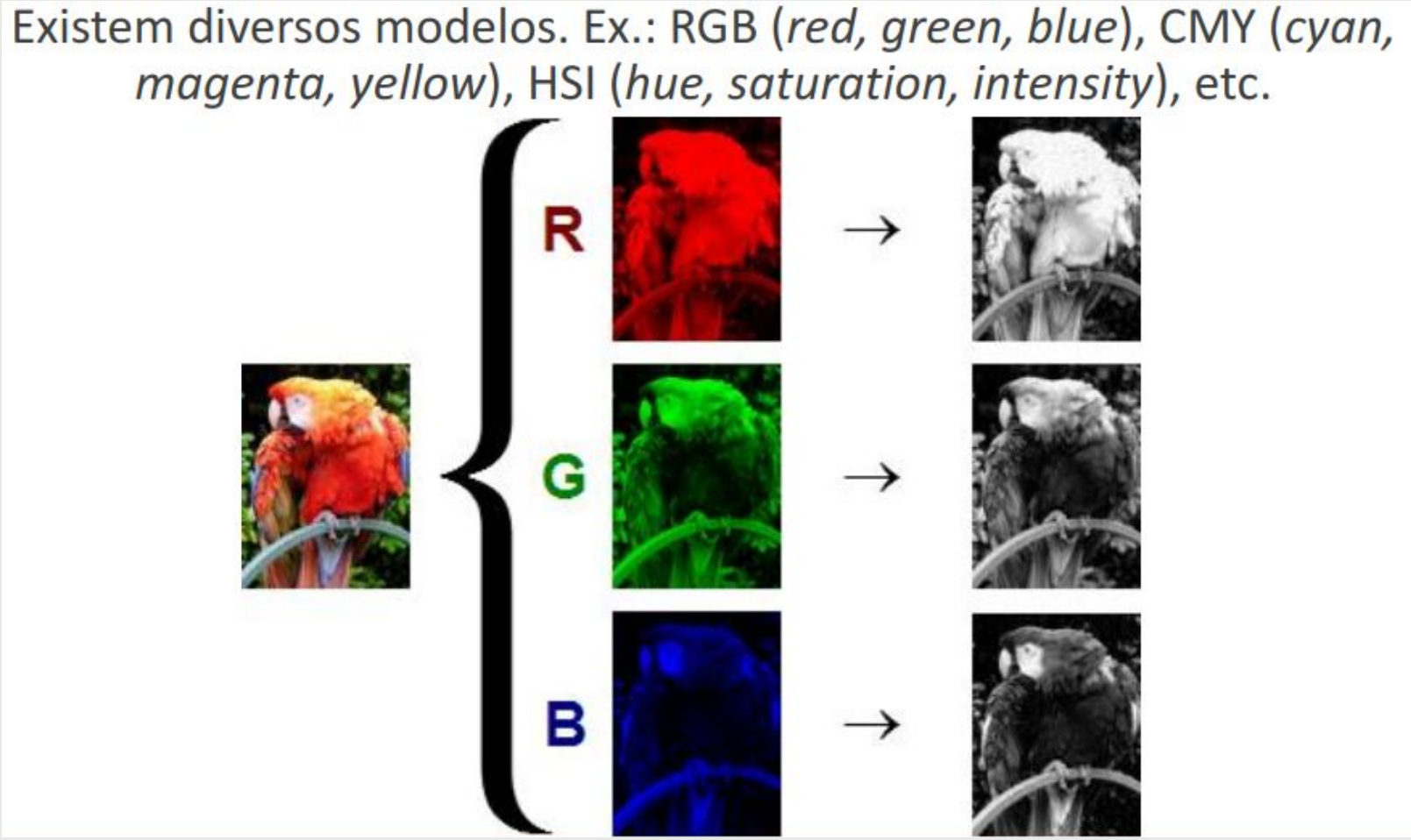


157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

*imagem:

No caso de imagens RGB(Red=vermelho, Green=verde, Blue=azul), existe uma matriz para cada cor (canal).

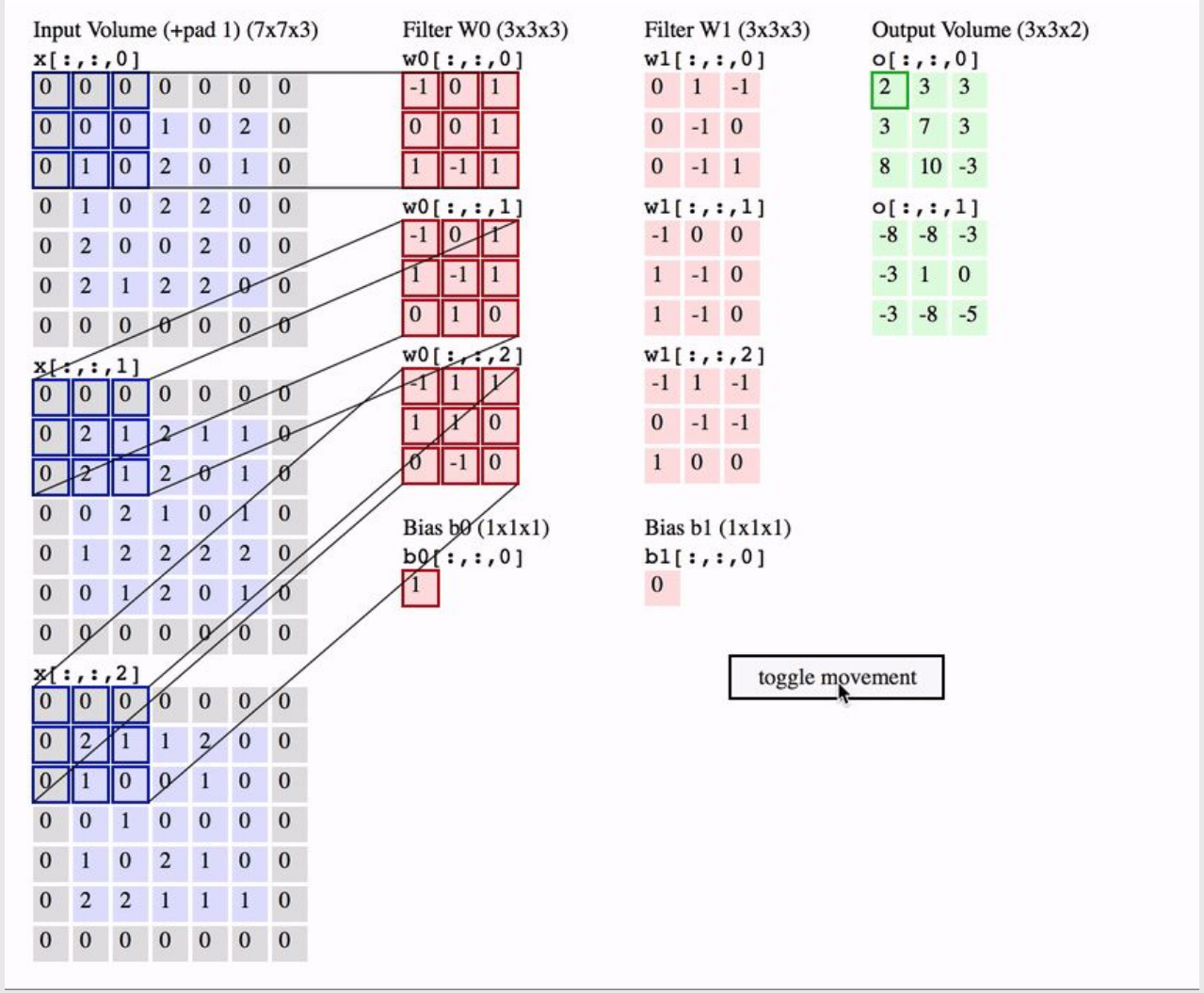


Como aprendem:

Filtros convolucionais são usados para extrair features de imagens.

Como aprendem:

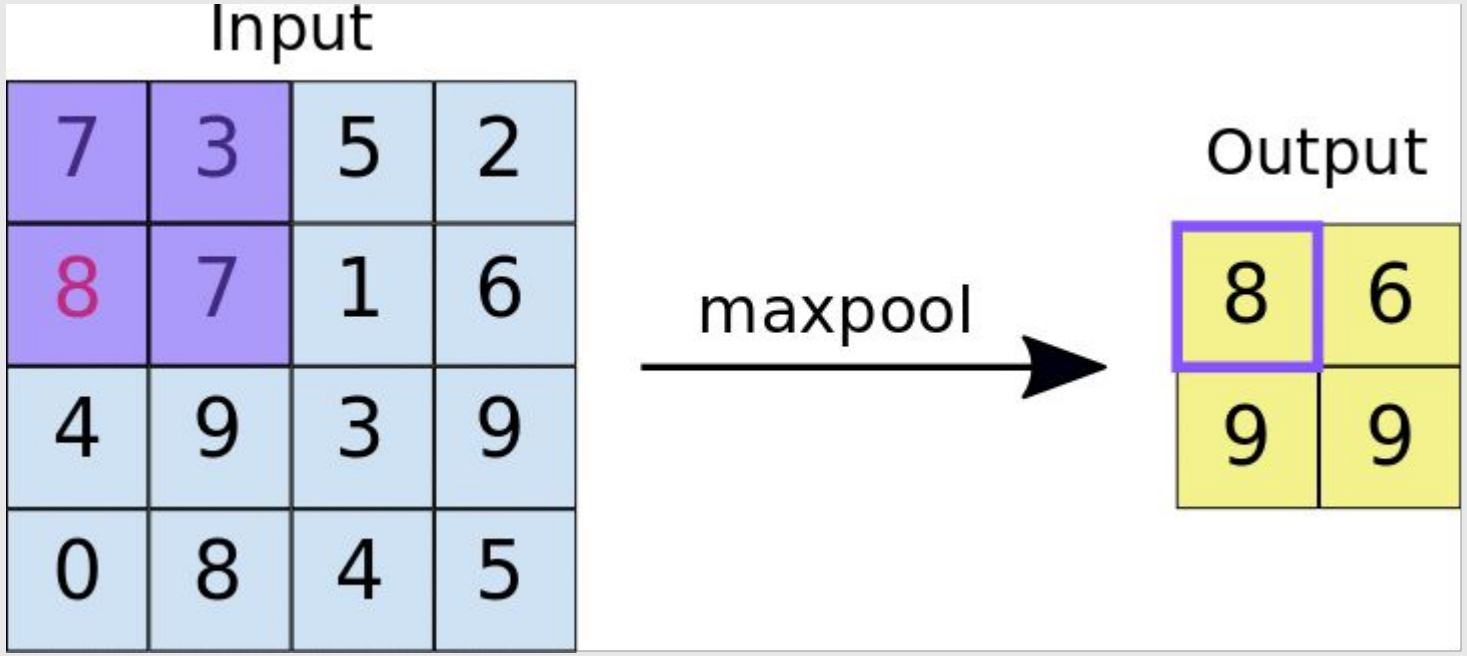
Nas redes convolucionais esses filtros são aplicados em várias camadas:



Vamos à prática - Construindo o modelo: <Notebook>

Outras camadas:

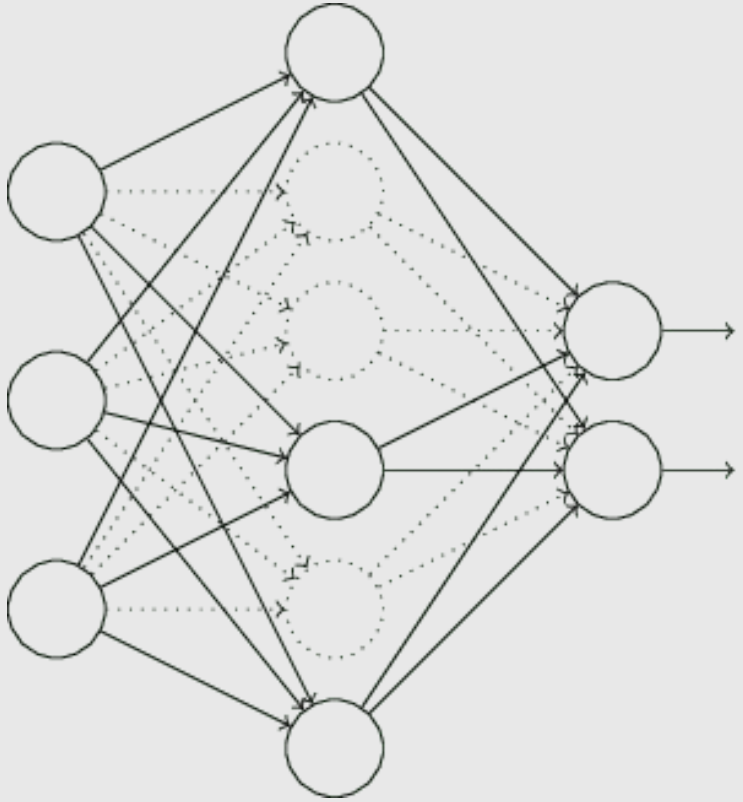
MaxPooling: Realiza o downsampling pós convolução



Diminui a quantidade de pixels a serem analisados nas camadas seguintes

Outras camadas:

Dropout: Em cada época desativa aleatoriamente um % de neurônios.

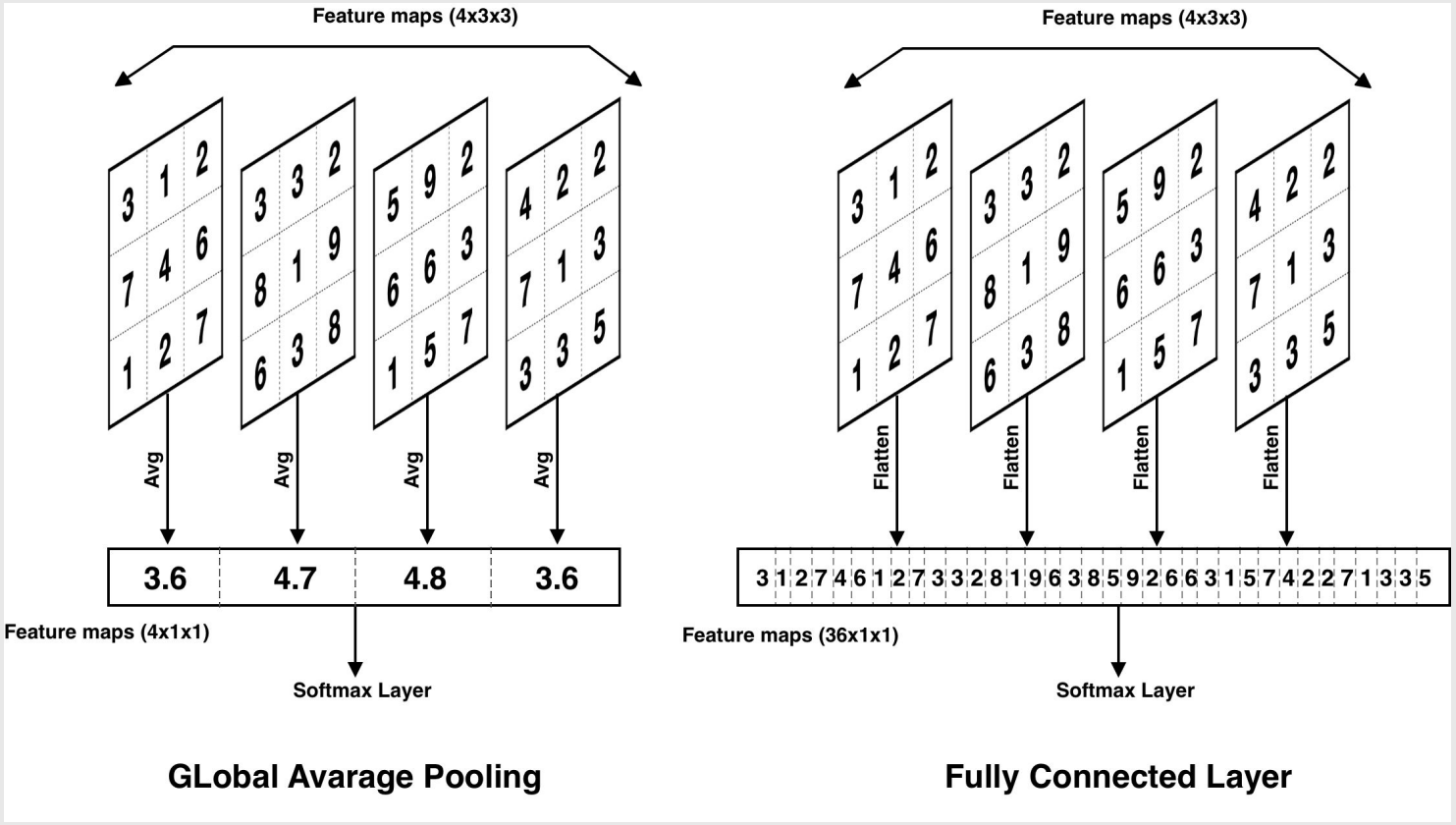


Diminui a chance de Overfitting

Outras camadas:

Quando saímos da convolução para ir para uma camada Densa precisamos voltar os valores para 1 Dimensão. Temos 2 possíveis maneiras para isso:

Global Average Pooling: Tiro a média geral de cada “mapa de feature”



Flatten: simplesmente “estica” tudo em uma dimensão

Vamos à prática - Construindo o modelo: <Notebook>

Pré-processamento:

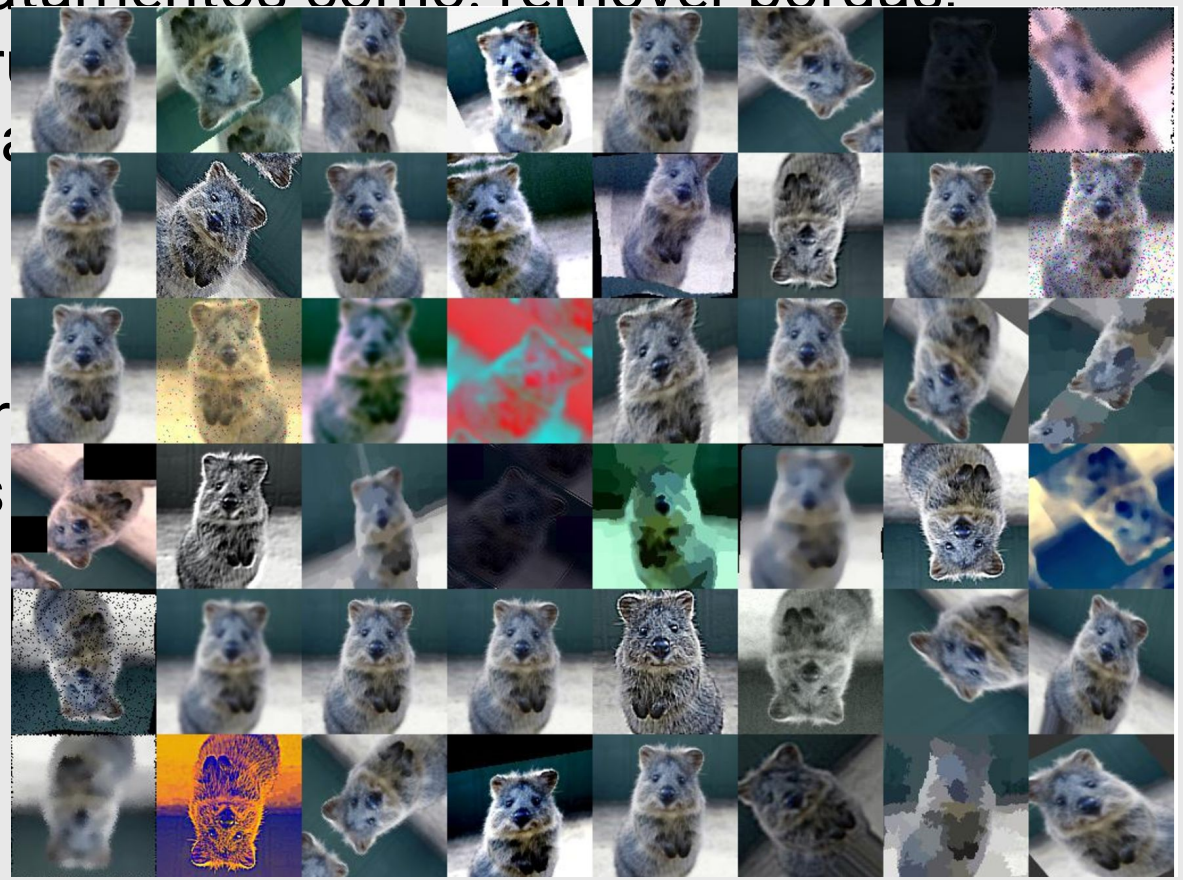
- **Tratamento das imagens:** É sempre uma boa prática normalizar as imagens. Também podemos adicionar outros tratamentos como: remover bordas, melhorar nitidez da imagem, remover ruídos, etc. (Uma boa biblioteca para ser estudada de processamento de imagens é o opencv)

- **Image Augmentation:** Processo de gerar mais imagens aplicando transformações em cima das originais para diversificar o dataset.

Pré-processamento:

- **Tratamento das imagens:** É sempre uma boa prática normalizar as imagens. Também podemos adicionar outros tratamentos como: remover bordas, melhorar nitidez da imagem, remover r
ser estudada de processamento de ima

- **Image Augmentation:** Processo de ger
transformações em cima das originais



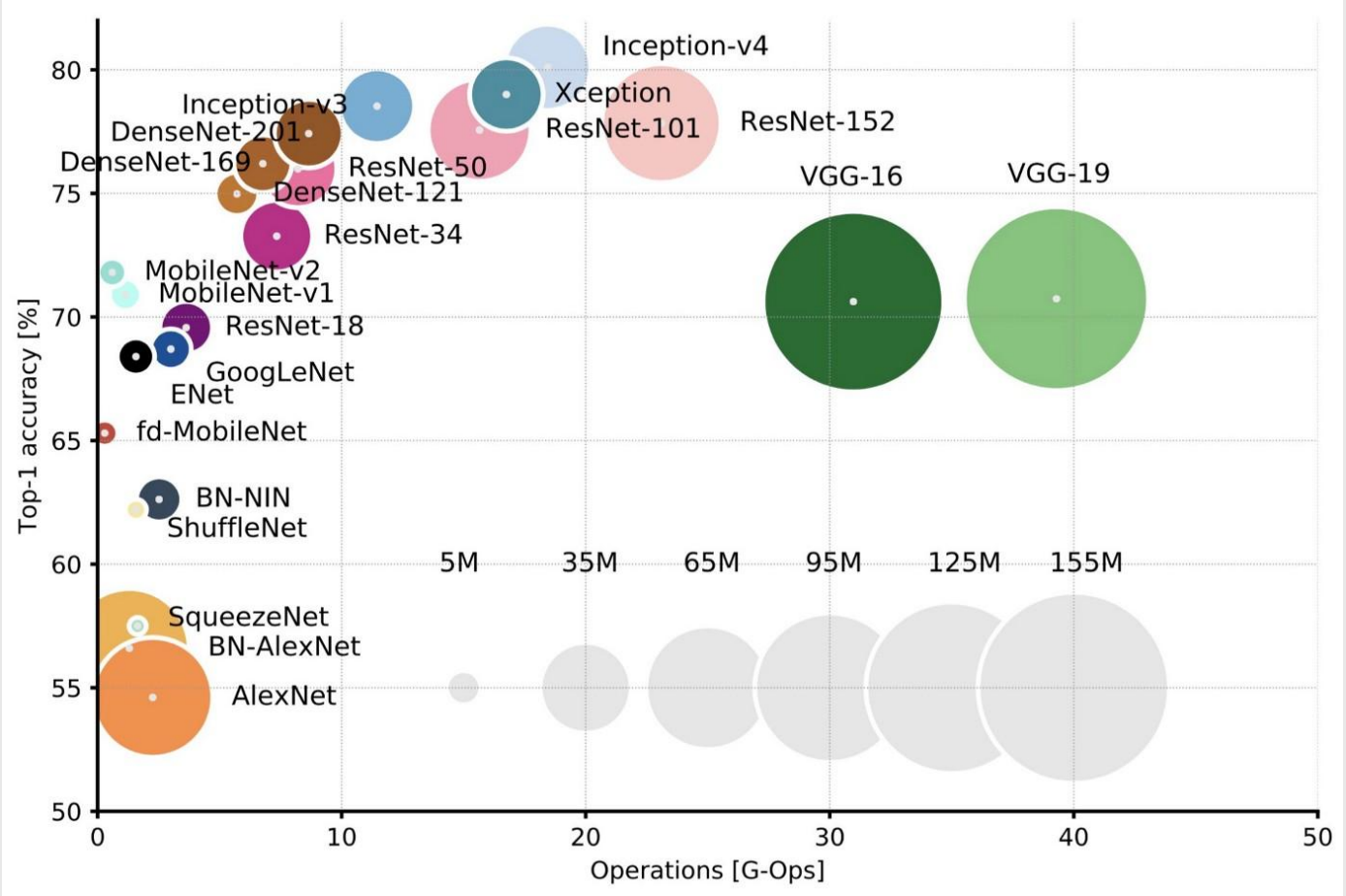
Treinamento:

- **Generator:** para leitura das imagens direto do disco. O image augmentor pode ocorrer aqui.

- **Callbacks:** funções chamadas durante o treinamento. Podem ser para: alteração do learning rate, parada do treinamento antes do total de épocas, salvar o modelo durante o treinamento, etc.

Transfer-Learning

Nem sempre precisamos definir a arquitetura da nossa rede do zero. Para reconhecimento de imagens existem arquiteturas já definidas e pré-treinadas com o Imagenet - um dataset com 1.2 milhões de imagens e 1000 categorias.



DÚVIDAS?!