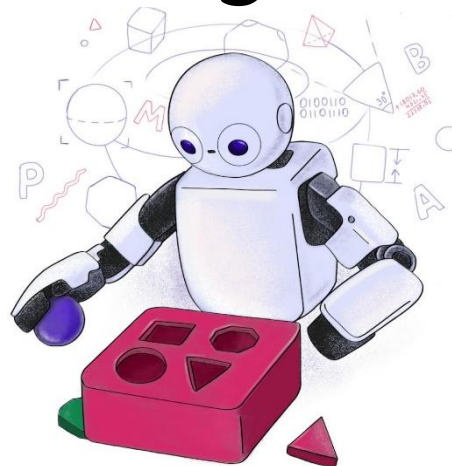


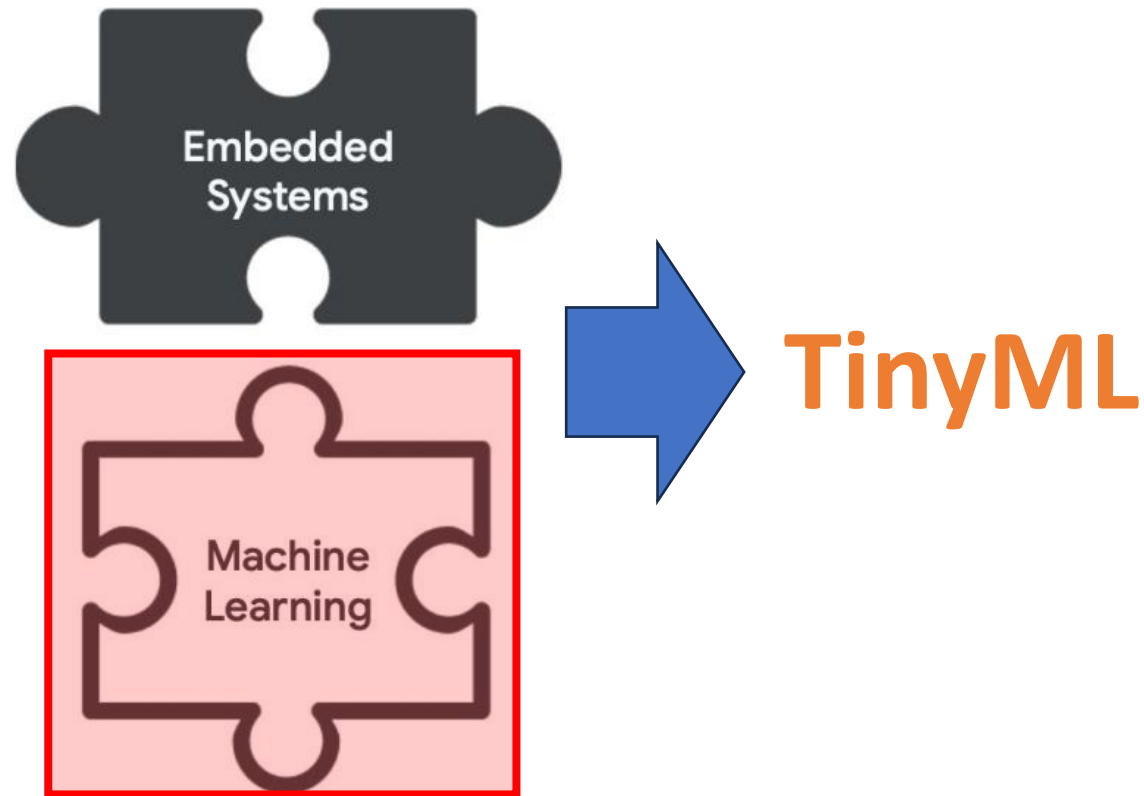
Desafios do TinyML: Machine Learning



Inatel

Felipe Augusto Pereira de Figueiredo
felipe.figueiredo@inatel.br

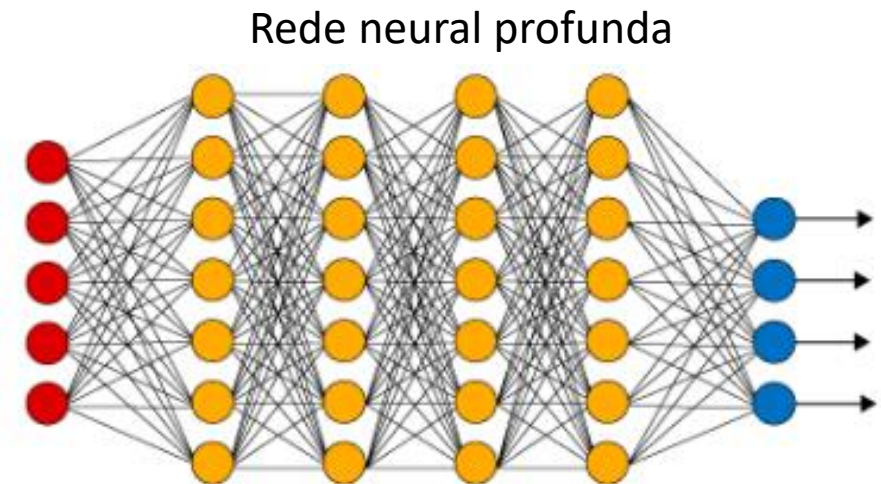
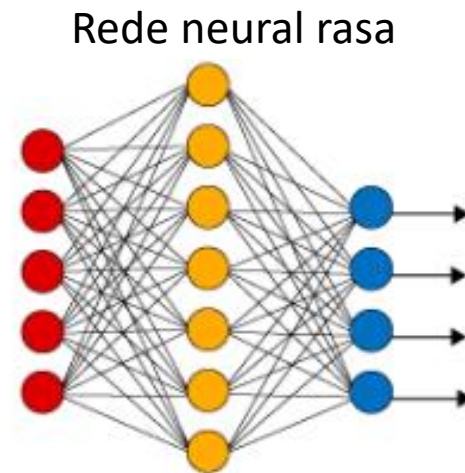
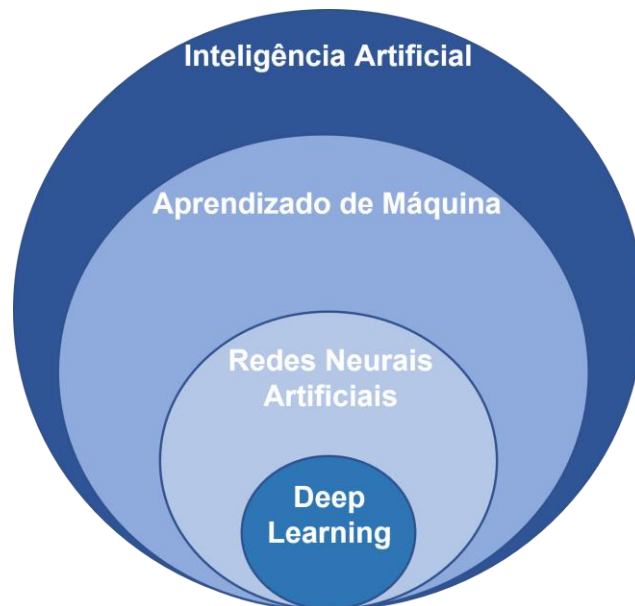
Desafios da execução de ML em sistemas embarcados



- Anteriormente, falamos das ***limitações de HW e SW quando usamos sistemas embarcados.***
- Agora, veremos os ***desafios para execução algoritmos de ML***, mais especificamente de *Deep Learning*, ***nestes dispositivos pequenos, com restrições de custo, recursos computacionais e consumo.***

Deep Learning ou Aprendizado Profundo

- **Subárea** do aprendizado de máquina que usa **redes neurais artificiais com muitas camadas ocultas** para aprender com dados.
- Por terem uma **grande capacidade de aprendizado**, em geral, precisam de uma **grande quantidade de dados** para aprenderem (i.e., encontrar uma solução geral).

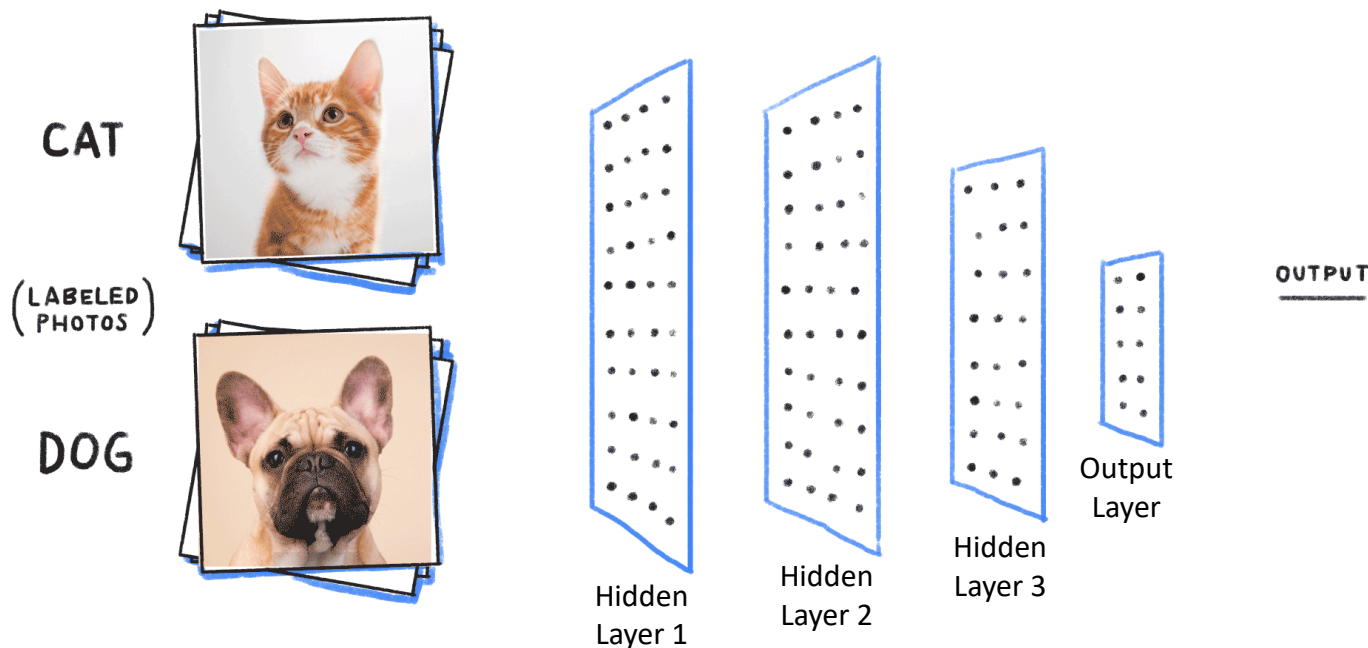


● Input Layer

● Hidden Layer

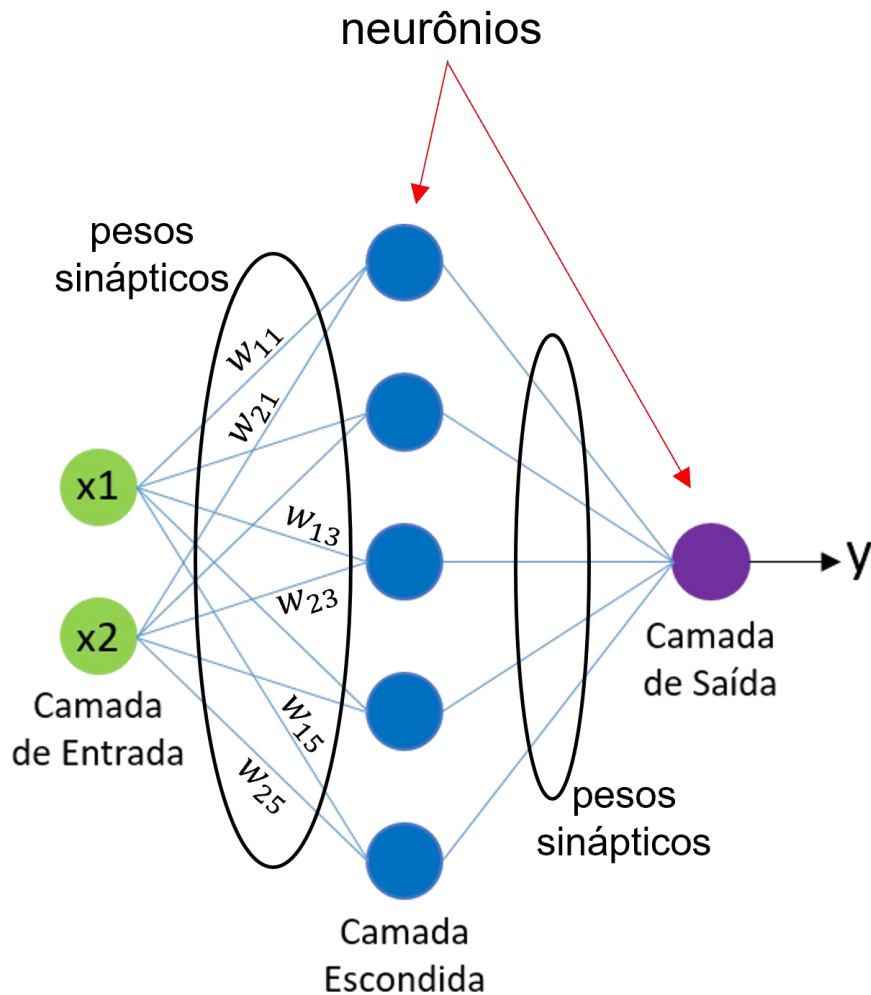
● Output Layer

Deep Learning



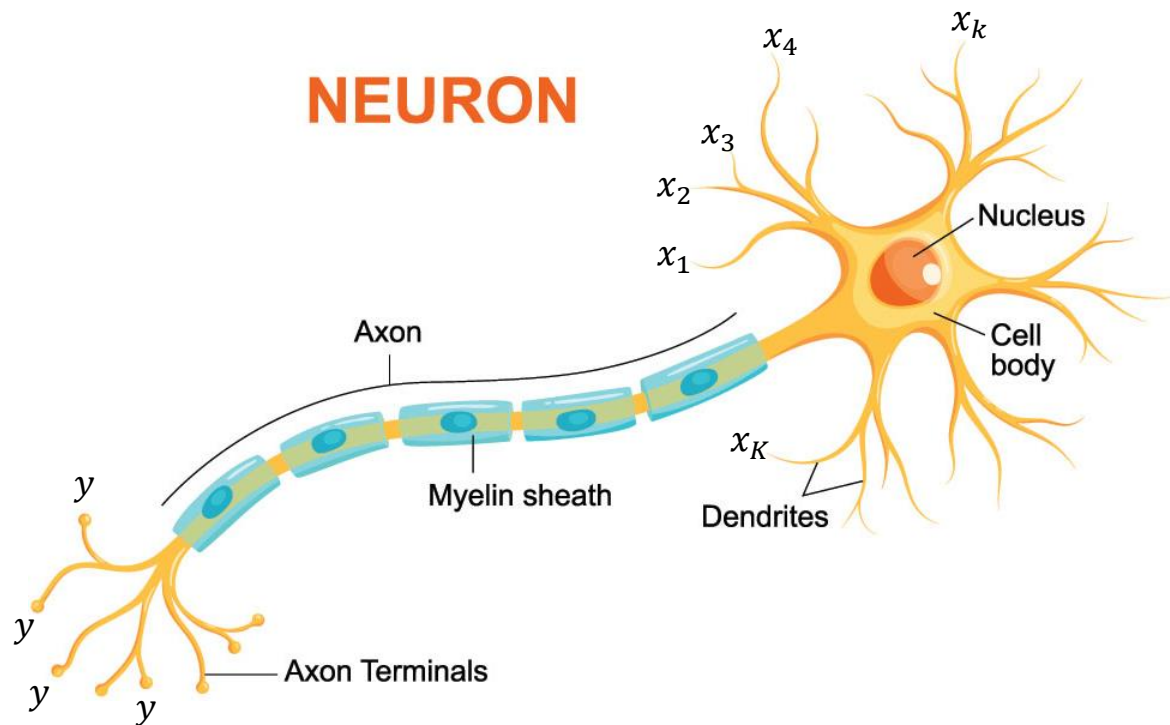
- Um exemplo muito comum de uso do *deep learning* é a **classificação de imagens**.
- Se tivermos uma base de dados com imagens de gatos e cachorros, podemos **treinar uma rede neural profunda para identificá-los** em imagens.
- O **objetivo** é obter um **modelo que generalize**, ou seja, que identifique gatos ou cachorros não vistos durante o treinamento.

O que é uma rede neural artificial?



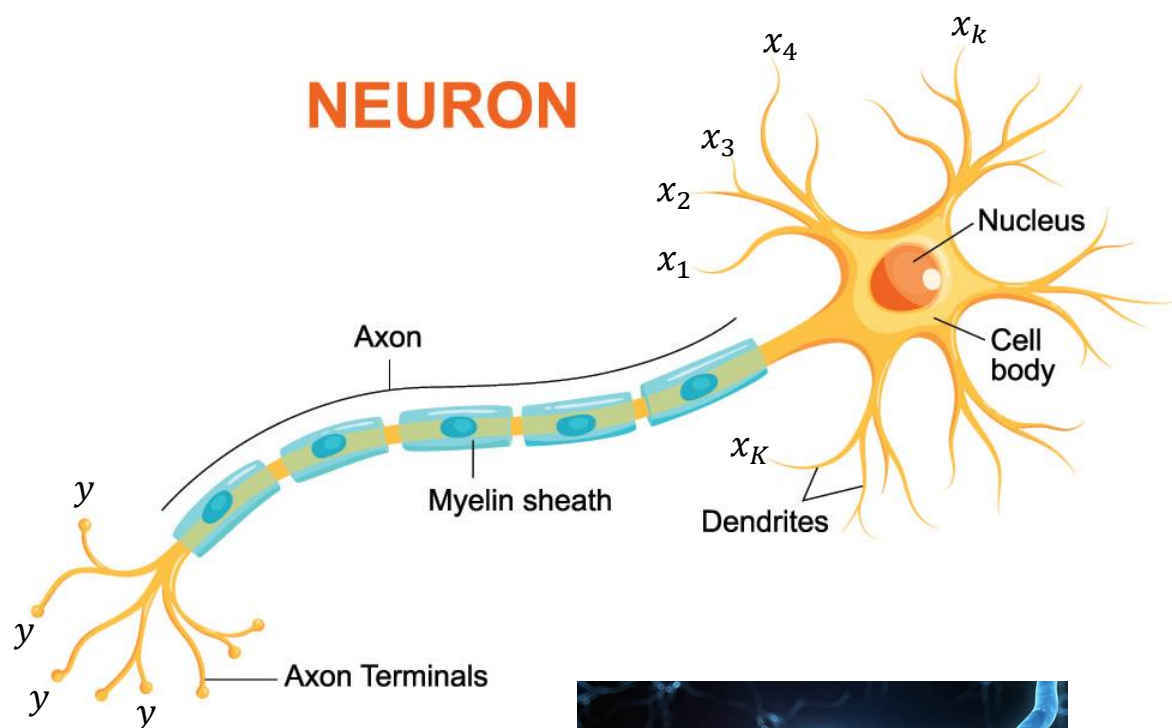
- É uma **conexão de camadas de neurônios artificiais**, ou também chamados de nós.
- Os **neurônios artificiais** são **modelos matemáticos inspirados no funcionamento de um neurônio biológico**.
- Os neurônios das diferentes camadas são conectados através dos **pesos sinápticos**, que determinam a força daquela conexão.

Neurônio biológico



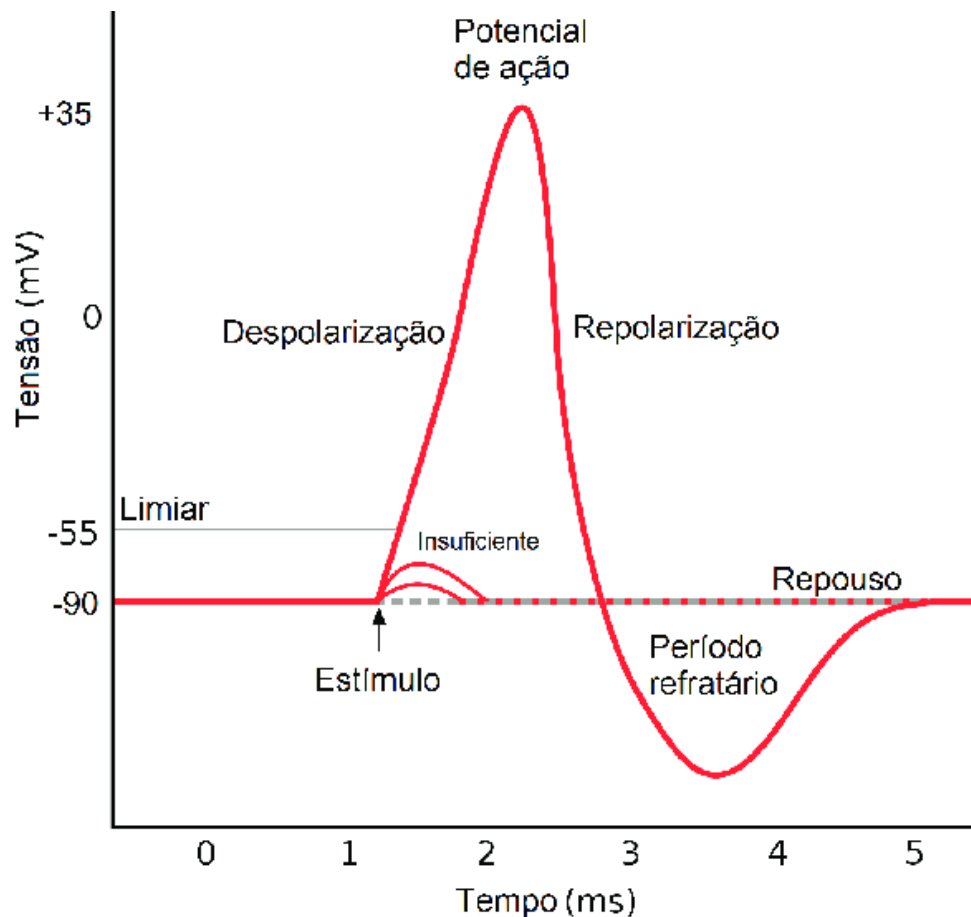
- São **células** que possuem **mecanismos eletroquímicos** para realizar a **transmissão de sinais elétricos** (i.e., informações) **ao longo do sistema nervoso**.
- Têm três partes fundamentais: os **dendritos**, o **axônio** e o **corpo celular**, também chamado de **soma**.
- Os **dendritos recebem estímulos** vindos **de outros neurônios** e os levam em direção ao **corpo celular**.

Neurônio biológico



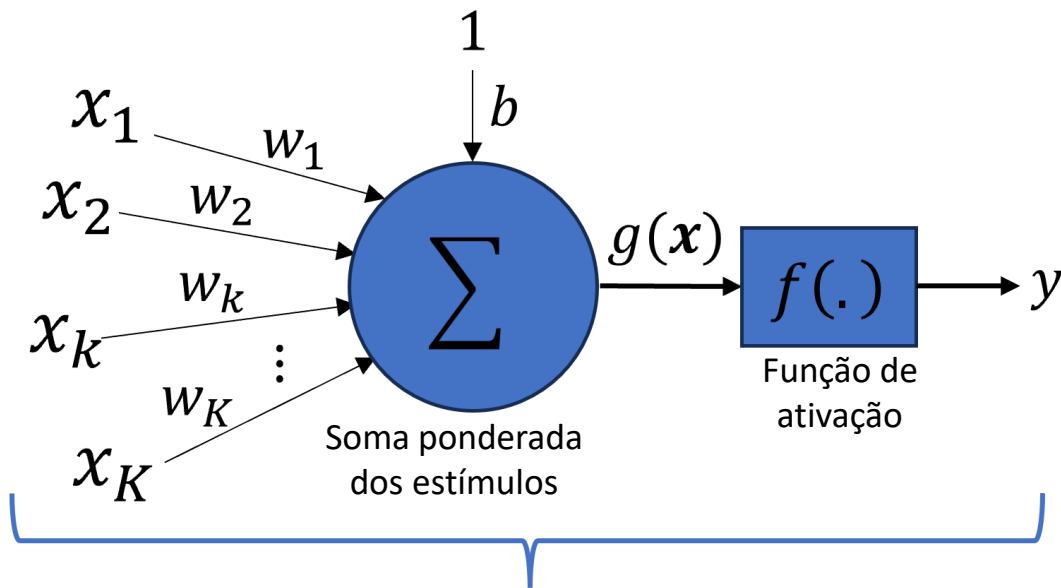
- O **corpo celular** realiza a **integração** dos **estímulos e gera impulsos**.
- O **axônio** envia **impulsos** a outros neurônios através de seus **terminais**.
- Os neurônios se comunicam uns com os outros através das **sinapses**.
- **Sinapses** são os **pontos de contato** entre os **dendritos** de um neurônio e os **terminais do axônio** de outros neurônios.

Neurônio biológico



- Em termos bem simples, o funcionamento do **neurônio** pode ser explicado da seguinte forma:
 - Ele **recebe estímulos elétricos** a partir dos dendritos.
 - Esses **estímulos são somados** no corpo celular (*soma*).
 - Se a **soma dos estímulos** exceder um certo **limiar de ativação**, o **neurônio** gera um **impulso** (ou **potencial de ação**) que é enviado pelos terminais do axônio a outros neurônios através das **sinapses**.

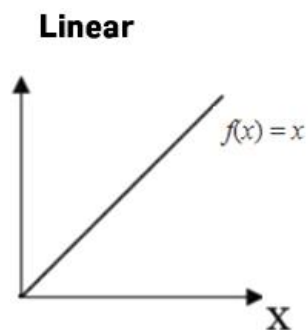
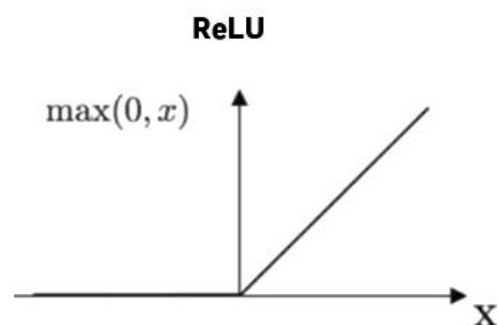
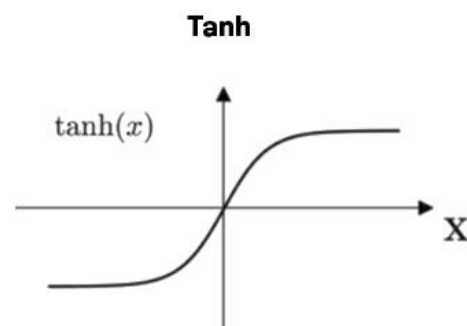
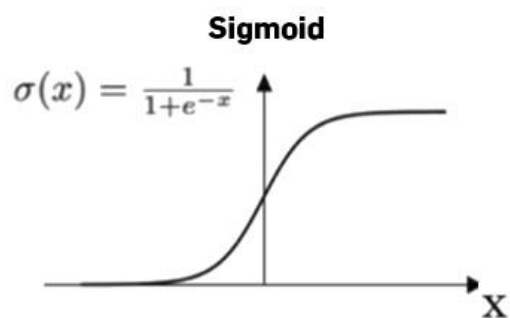
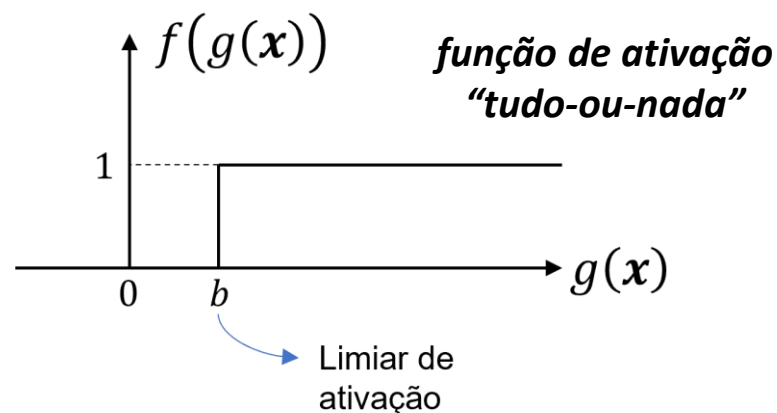
Neurônio artificial



$$y = f(g(\mathbf{x})) = f\left(\left(\sum_{i=1}^K w_i x_i\right) + b\right)$$

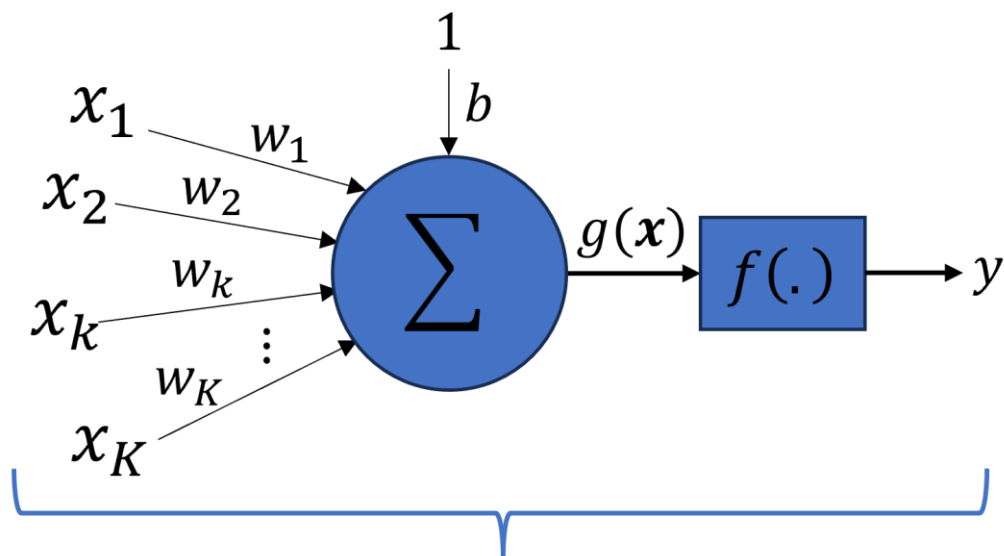
- Baseado no entendimento do funcionamento do neurônio biológico, a partir de meados da década de 40, pesquisadores propuseram o modelo matemático ao lado.
 - É uma **simplificação e abstração** do funcionamento do neurônio biológico.
- Os **estímulos**, $x_k, k = 1, \dots, K$, são multiplicados pelos **pesos sinápticos**, $w_k, k = 1, \dots, K$, somados com o **peso de bias**, b , e o resultado é passado por uma **função de ativação**, $f(\cdot)$.

Neurônio artificial



- O **peso de bias**, b , permite **ajustar o limiar de ativação** (ou ponto de disparo) da função de ativação.
- No início dos estudos dos neurônios, a **função de ativação** era do tipo **tudo-ou-nada**, ou seja, ativava ou não (degrau unitário).
- Com o decorrer do tempo, outras funções com características diferentes foram propostas, tais como as funções sigmoide, tangente hiperbólica, *relu*, linear, etc.

Neurônio artificial



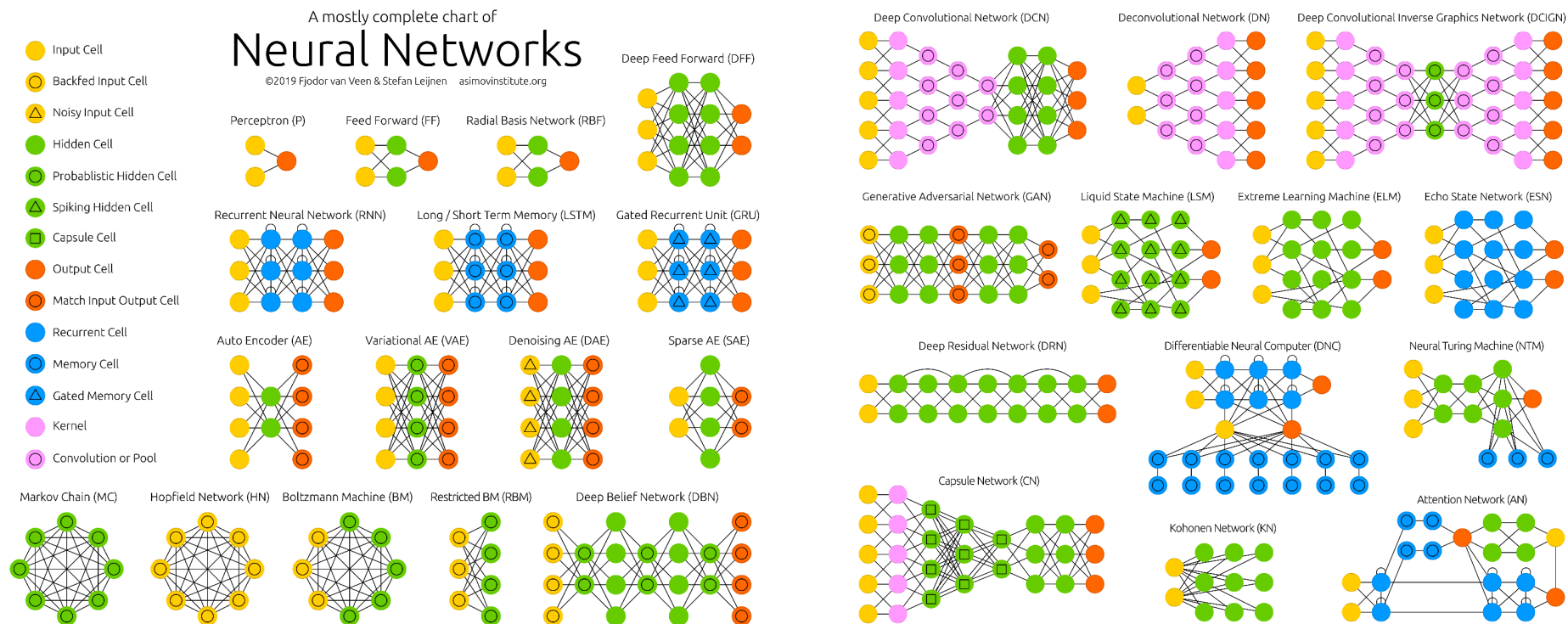
$$y = f \left(\left(\sum_{i=1}^K w_i x_i \right) + b \right)$$

Pesos ou
parâmetros

- O **objetivo** do treinamento de uma rede neural artificial é **encontrar os valores ideais dos pesos** (bias e sinápticos) **de todos os neurônios** de forma que a rede **resolva uma tarefa específica**, como, por exemplo, a classificação de imagens.
- Os pesos são ajustados através de um **processo de otimização iterativo** chamado de **retropropagação do erro**, onde apresenta-se ao modelo as **entradas e saídas esperadas** e o processo **minimiza o erro** entre a **saída da rede** e os **valores de saída esperados**.

Arquiteturas de redes neurais artificiais

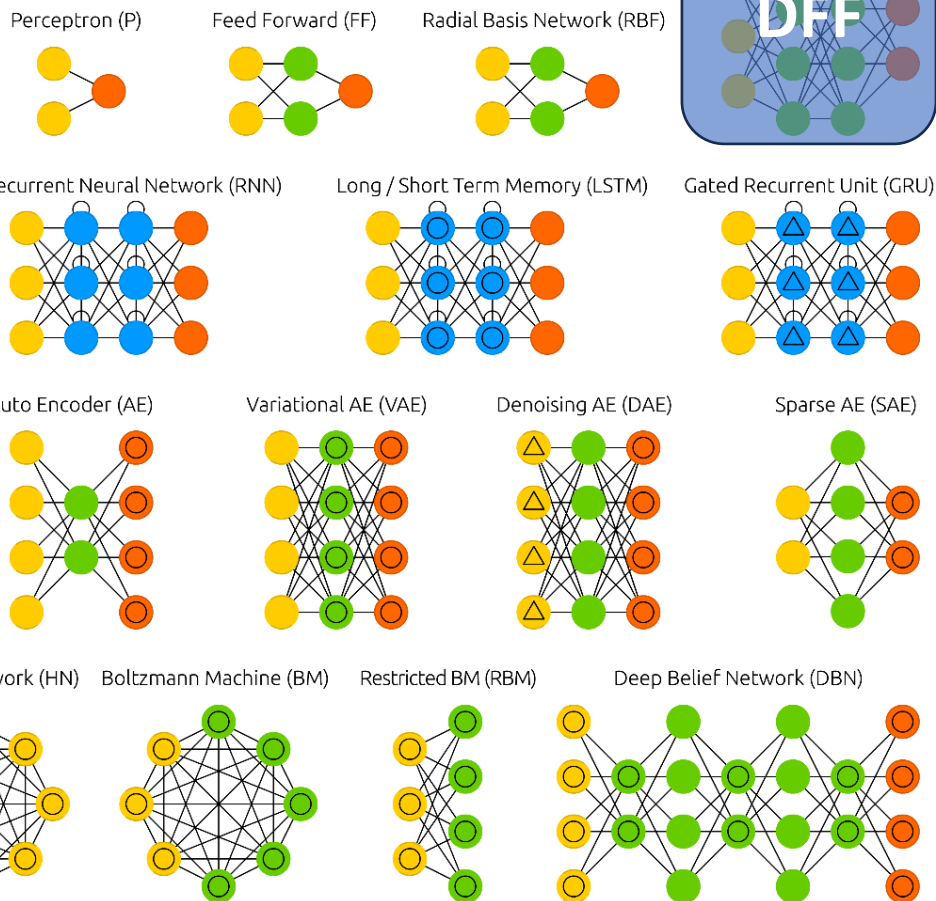
- Diferentes **tipos de neurônios**, a **quantidade de camadas** e de **neurônios** em cada uma delas e a **forma como eles estão conectados**, resultam em **arquiteturas** diferentes.



Arquiteturas de redes neurais artificiais

A mostly complete chart of Neural Networks

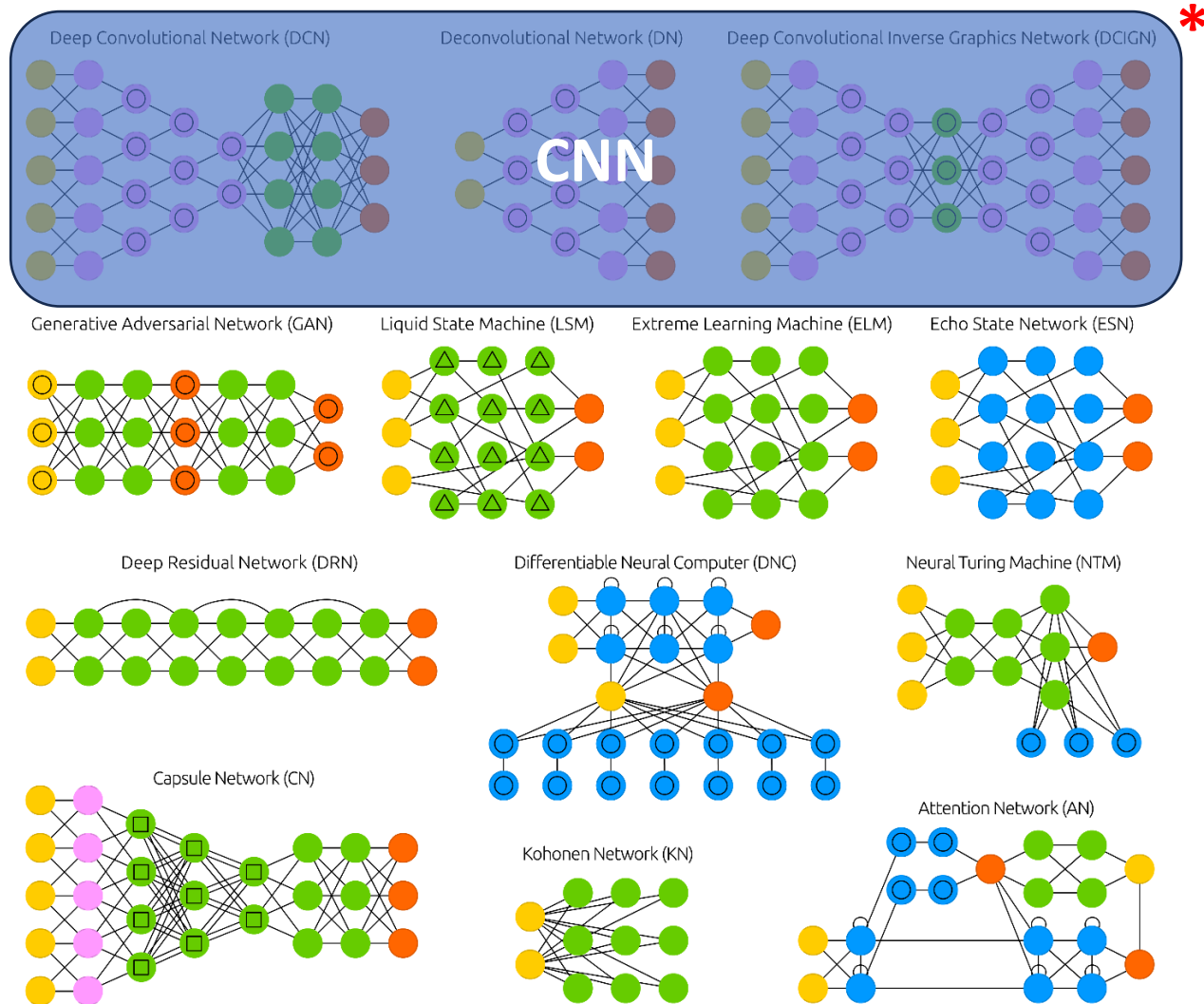
©2019 Fjodor van Veen & Stefan Leijnen asimovinstitute.org



- **DFFs:** também chamadas de *dense neural networks* (DNN), são redes ***densamente*** conectadas (todas as saídas de uma camada se conectam a todos os nós da próxima) com 2 ou mais camadas ocultas.
- O termo ***feed forward*** vem do fato de que as conexões são sempre no sentido da ***entrada para a saída***.
- São usadas em tarefas de ***aproximação de funções*** (i.e., regressão e classificação).

* Modelo que será muito usado no curso.

Arquiteturas de redes neurais artificiais



- **CNNs:** ou redes neurais convolucionais, são redes que utilizam **camadas de convolução**, que aplicam **filtros** para **extrair características relevantes** dos dados de entrada, em geral imagens.
- São usadas em aplicações de **visão computacional**, como classificação de imagens, processamento de vídeos, detecção de objetos em imagens ou vídeos.

Arquiteturas de redes neurais artificiais

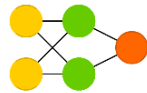
A mostly complete chart of Neural Networks

©2019 Fjodor van Veen & Stefan Leijnen asimovinstitute.org

Perceptron (P)



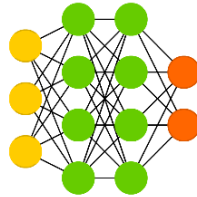
Feed Forward (FF)



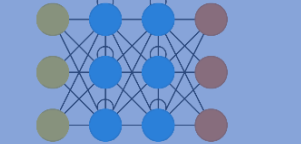
Radial Basis Network (RBF)



Deep Feed Forward (DFF)



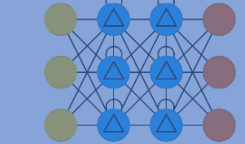
Recurrent Neural Network (RNN)



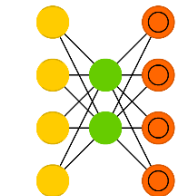
Long / Short Term Memory (LSTM)



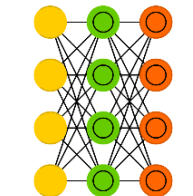
Gated Recurrent Unit (GRU)



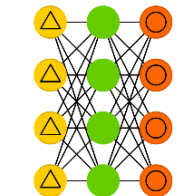
Auto Encoder (AE)



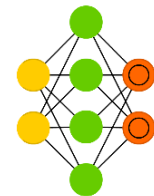
Variational AE (VAE)



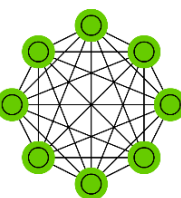
Denoising AE (DAE)



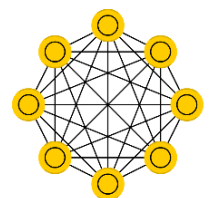
Sparse AE (SAE)



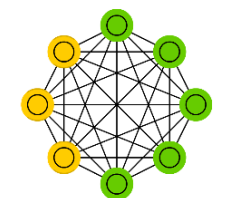
Markov Chain (MC)



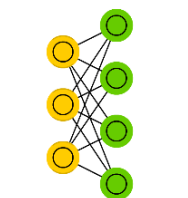
Hopfield Network (HN)



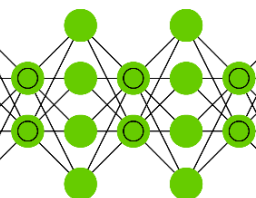
Boltzmann Machine (BM)



Restricted BM (RBM)



Deep Belief Network (DBN)

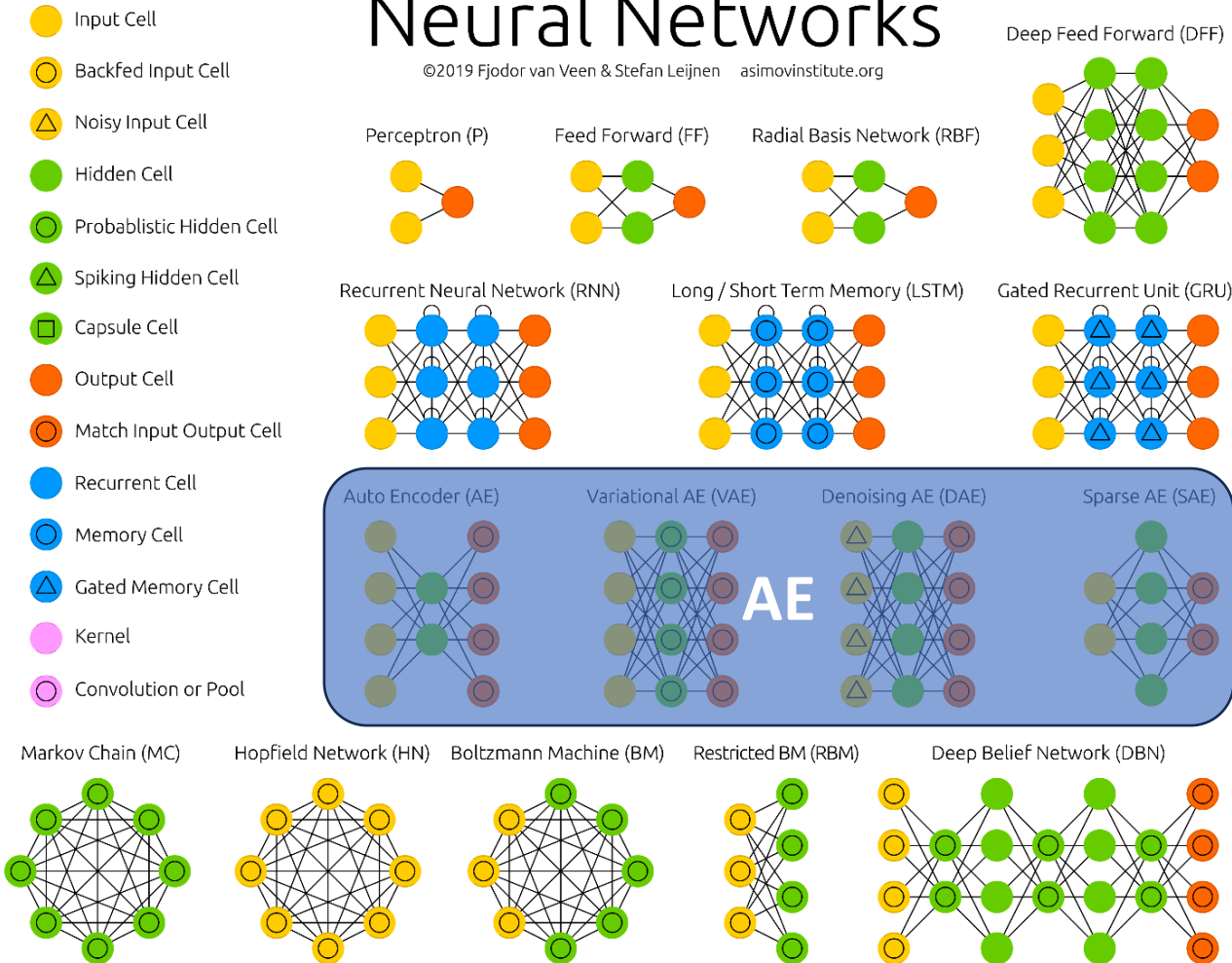


- **RNNs:** ou redes neurais *recorrentes*, possuem **conexões que formam loops**, permitindo que **informações anteriores sejam armazenadas e influenciem as entradas futuras**, ou seja, elas possuem **memória**.
- Ou seja, elas consideram o **contexto anterior** ao fazer previsões.
- Essa capacidade as torna especialmente úteis em **tarefas que envolvem dados sequenciais ou temporais**, como **análise de texto, previsão de séries temporais e processamento de fala e áudio**.

Arquiteturas de redes neurais artificiais

A mostly complete chart of Neural Networks

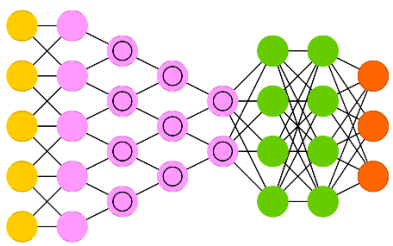
©2019 Fjodor van Veen & Stefan Leijnen asimovinstitute.org



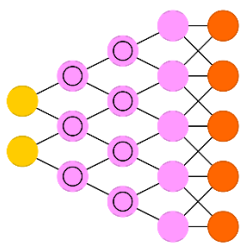
- **AEs:** os *autoencoders* têm como objetivo aprender uma **representação latente** (i.e., características importantes, mas ocultas) dos dados de entrada.
- Consistem em duas partes: o **codificador**, que **mapeia** os dados de entrada em uma **representação latente** (de maior ou menor dimensão), e o **decodificador**, que **reconstrói** os **dados originais** a partir dela.
- São usados em tarefas de **redução de dimensionalidade**, **remoção de ruído**, **compressão de dados**, **geração de dados sintéticos** e **redundância**.

Arquiteturas de redes neurais artificiais

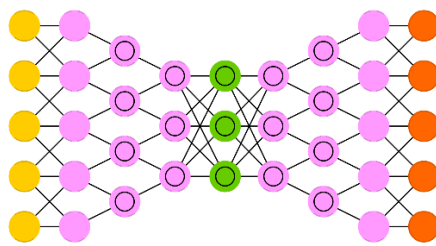
Deep Convolutional Network (DCN)



Deconvolutional Network (DN)



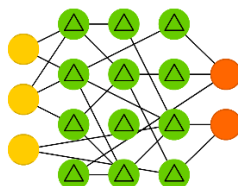
Deep Convolutional Inverse Graphics Network (DCIGN)



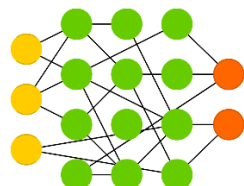
Generative Adversarial Network (GAN)



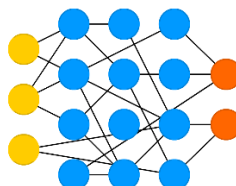
Liquid State Machine (LSM)



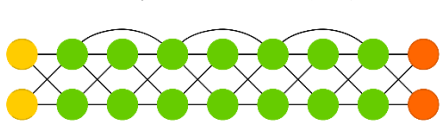
Extreme Learning Machine (ELM)



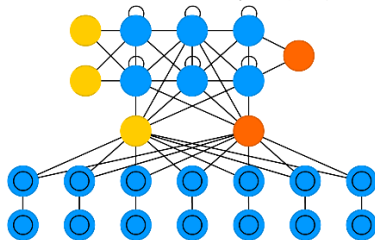
Echo State Network (ESN)



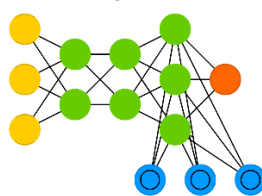
Deep Residual Network (DRN)



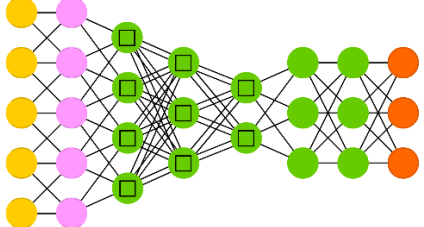
Differentiable Neural Computer (DNC)



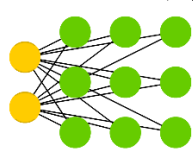
Neural Turing Machine (NTM)



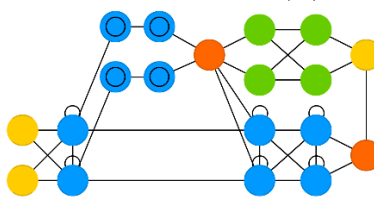
Capsule Network (CN)



Kohonen Network (KN)



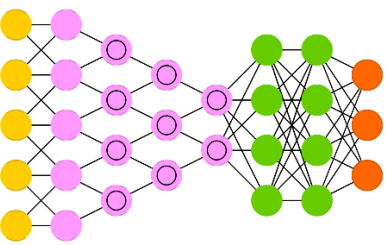
Attention Network (AN)



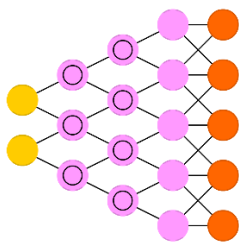
- **GANs:** ou *redes adversárias generativas*, são um tipo especial de rede neural que consiste em **duas redes em competição**: o **gerador** e o **discriminador**.
- O **gerador** *cria dados sintéticos* que se assemelham a dados reais, enquanto o **discriminador** tenta *distinguir entre dados reais e sintéticos*.
- O **objetivo** é *enganar o discriminador*.
- São usadas para **geração de imagens, vídeos e sons** realistas.

Arquiteturas de redes neurais artificiais

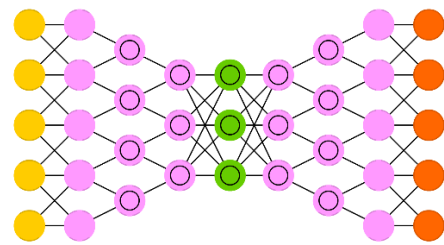
Deep Convolutional Network (DCN)



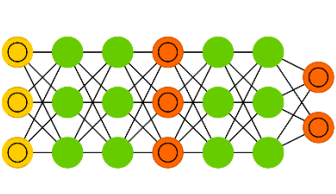
Deconvolutional Network (DN)



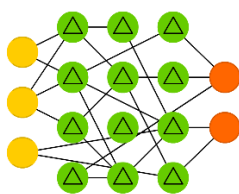
Deep Convolutional Inverse Graphics Network (DCIGN)



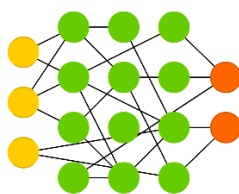
Generative Adversarial Network (GAN)



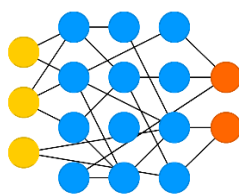
Liquid State Machine (LSM)



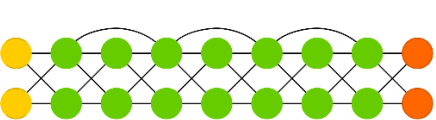
Extreme Learning Machine (ELM)



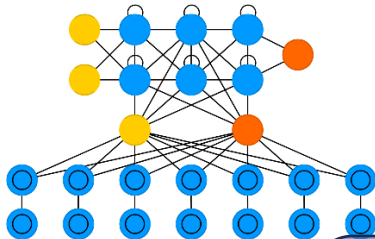
Echo State Network (ESN)



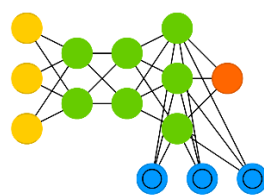
Deep Residual Network (DRN)



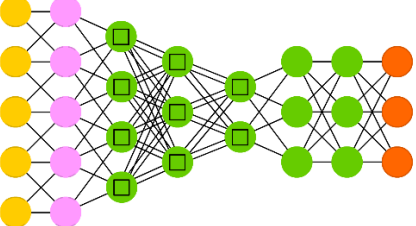
Differentiable Neural Computer (DNC)



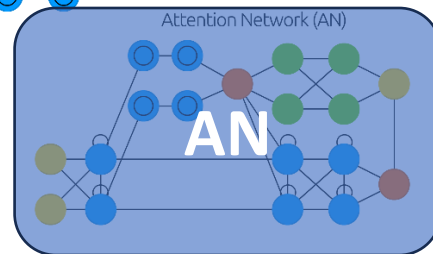
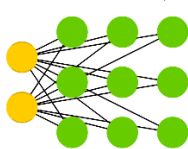
Neural Turing Machine (NTM)



Capsule Network (CN)



Kohonen Network (KN)



- **ANs:** ou **redes de atenção**, são redes que se concentram em **destacar partes importantes** dos dados de entrada, dando-lhes **maior peso durante** o processamento.
- Elas **atribuem pesos diferentes** às partes da entrada e as combinam de forma ponderada, **focando nas partes mais importantes e ignorando as menos importantes**.
- Propostas para lidar com as **limitações das RNNs** com dependências de longo prazo.
- São usadas em tarefas que envolvem sequências de dados, como **tradução automática, processamento de linguagem natural e reconhecimento de fala**.

Crescimento do tamanho dos modelos



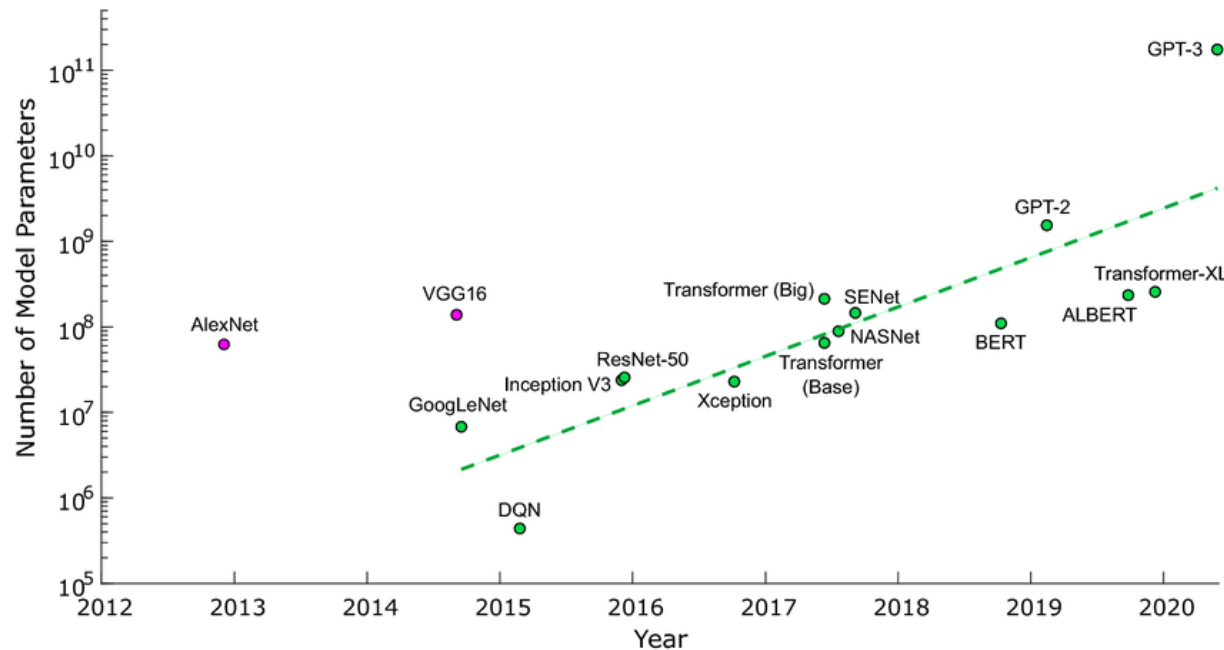
- Ao longo dos anos, para resolver ***problemas mais e mais complexos*** com ***ótimos resultados***, as ***arquiteturas*** das redes neurais têm se ***tornado maiores e mais complexas***.
- OBS.: A ***complexidade*** ou ***capacidade*** de uma rede neural está ***relacionada*** com sua ***quantidade de camadas e nós***.

Crescimento do tamanho dos modelos



- A ***complexidade computacional*** de uma rede neural é ***diretamente proporcional ao seu número de conexões***.
- Portanto, ***quanto mais camadas e nós, mais memória e cálculos matemáticos*** são necessários e, ***consequentemente, maior será o consumo de memória e energia***.
- Até recentemente, ***não havia*** uma ***preocupação com as eficiências computacional e energética*** de soluções envolvendo IA.

Crescimento do tamanho dos modelos

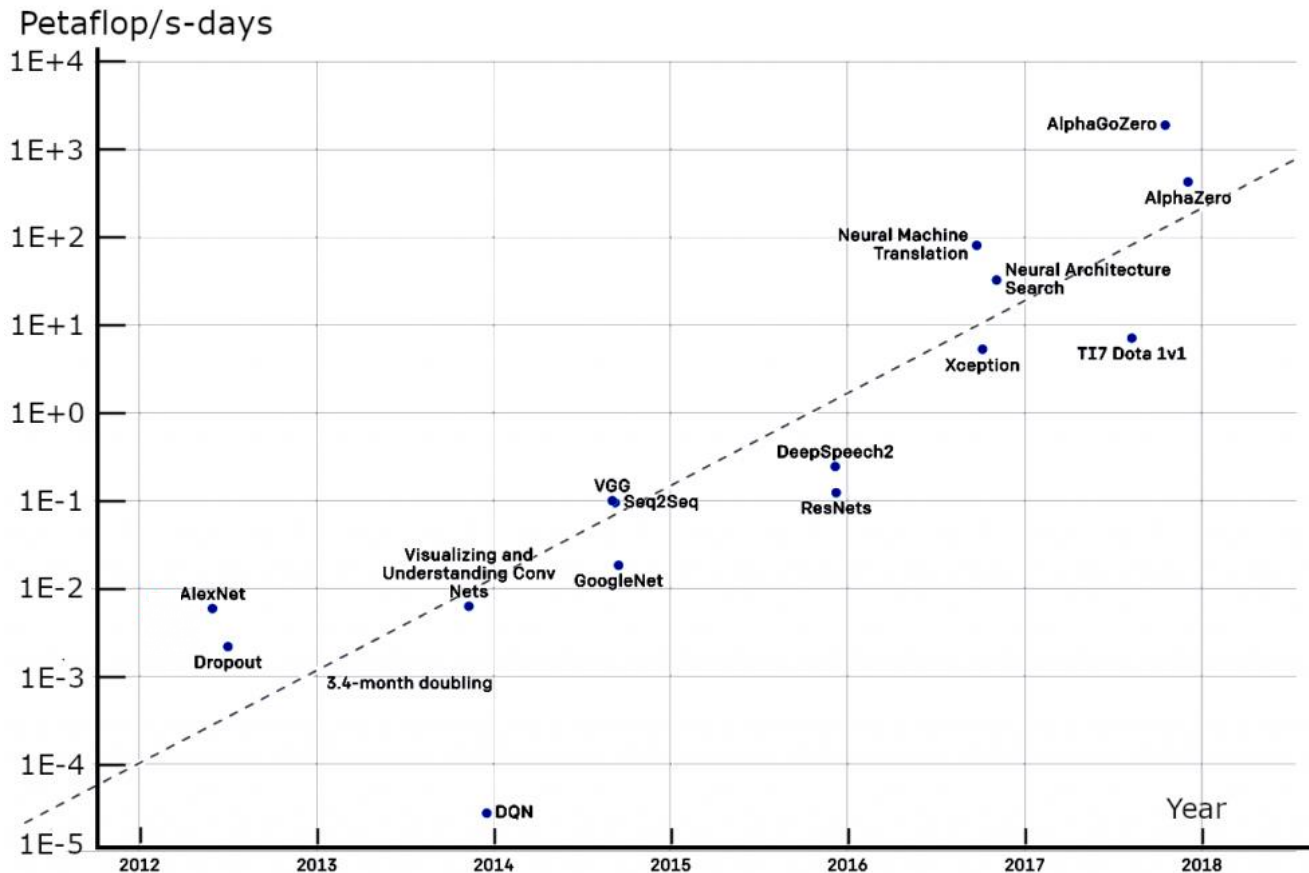


Se cada parâmetro corresponde a uma variável do tipo *float* (4 bytes), precisamos de $1.76 \times 10^{12} \times 4 \sim 7 \times 10^{12}$ bytes, ou seja, **7 Terabytes** para armazenar o modelo GPT-4!

- Os modelos não param de crescer!
- Isso se deve aos ***aumentos da disponibilidade de dados*** e do ***poder de computacional*** e ao desenvolvimento de ***novas técnicas de aprendizado*** (principalmente de aprendizado profundo).
- Vejamos o modelo de linguagem GPT:
 - O GPT-2 tinha aproximadamente 1.5 bilhão de parâmetros (i.e., pesos sinápticos).
 - Já os GPT-3/3.5 têm aproximadamente 175 bilhões de parâmetros.
 - E estima-se que o GPT-4 tenha 1.76 trilhão de parâmetros.

Necessidades computacionais (2012 - Atual)

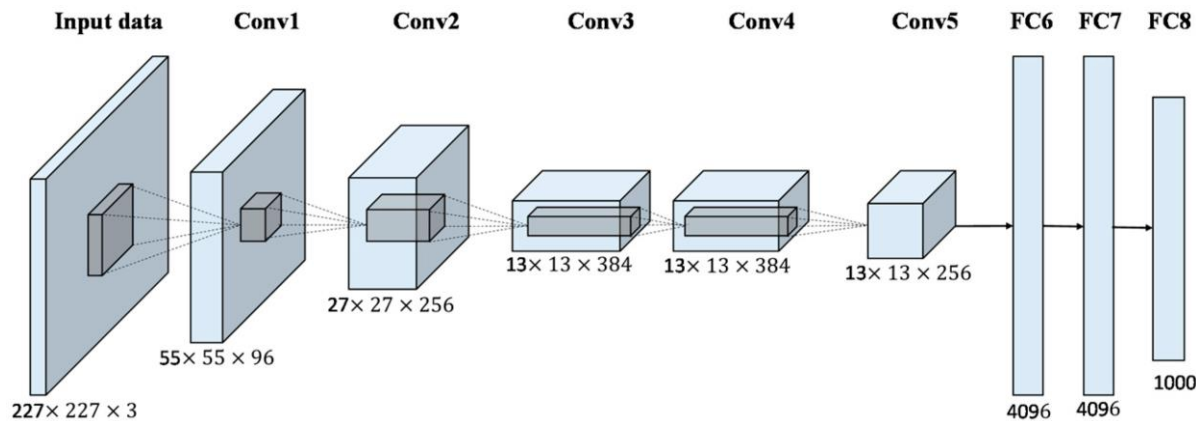
AlexNet to AlphaGo Zero: A 300,000x Increase in Compute (Log Scale)



Flops/s-day: quantidade de operações em ponto flutuante por segundo por dia necessárias para treinar o modelo.

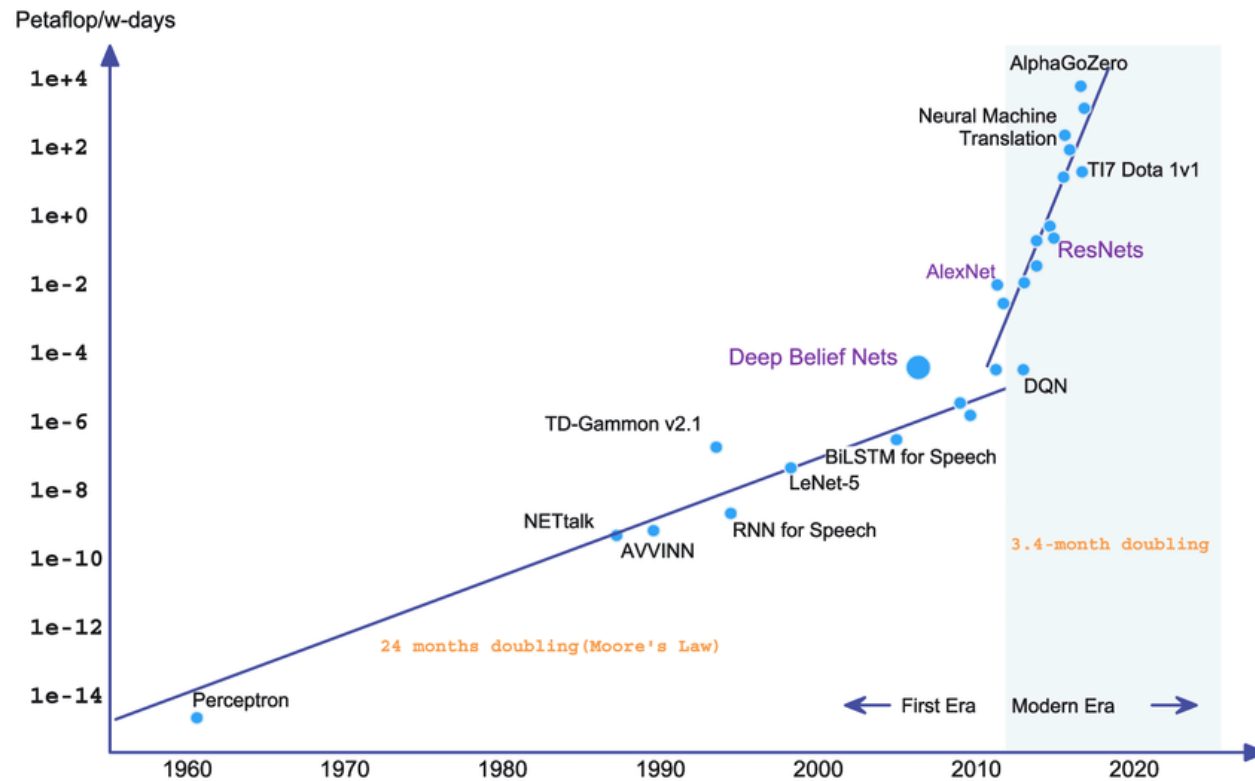
- Nos últimos anos, o ***poder computacional*** necessário para treinar os modelos de ML amplamente usados hoje ***teve que crescer 300.000 vezes***.
- Essa tendência de redes imensas começou em 2012 com a AlexNet.
- Rede neural desenvolvida na universidade de Toronto no Canadá.

Necessidades computacionais (2012 - Atual)



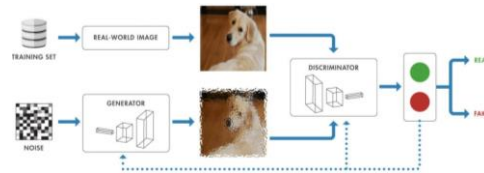
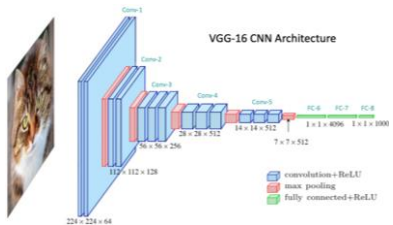
- **CNN** que **bateu** com folga todos os **recordes anteriores** do desafio *ImageNet Large Scale Visual Recognition* de 2012.
- Tem **60 milhões de parâmetros**, o que na época a tornou uma das maiores e mais complexas redes neurais.
- **Popularizou** o uso de **camadas convolucionais**, impulsionando a revolução do aprendizado profundo.
- Introduziu o uso da função de ativação **ReLU**, que mitiga o problema do desaparecimento do gradiente.
- Foi uma das **primeiras CNNs a usar GPUs** para reduzir o tempo de treinamento.
- Estabeleceu a base para muitas arquiteturas de CNNs subsequentes.

Necessidades computacionais (desde 1958)



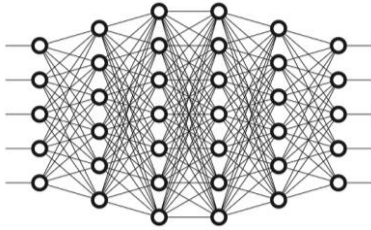
- Nos últimos 10 anos, a quantidade de cálculos necessários cresceu extraordinariamente rápido.
- Na “primeira era” do ML, a quantidade de cálculos e, consequentemente, o **tamanho dos modelos, dobrava a cada dois anos** aproximadamente.
- Na “era moderna”, os requisitos de computação praticamente dobram a cada **3/4 meses**.

Necessidades computacionais (desde 1958)



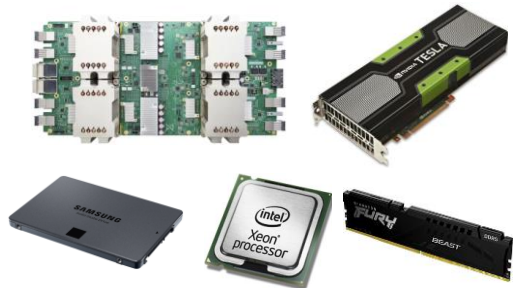
- O que aconteceu nesses últimos 10 anos que explica esse boom?
 - Disponibilidade de **grandes volumes de dados** devido à internet (> 100 TB/dia).
 - **Aumento do poder de computacional** através de GPUs, FPGAs e CPUs com múltiplos cores.
 - **Surgimento de novos algoritmos de ML**, como redes neurais profundas, redes adversárias generativas (GANs), redes de atenção, *transformers*, *deep reinforcement-learning*, etc.
 - Disponibilidade de **frameworks e bibliotecas amigáveis**, como TensorFlow e PyTorch, que facilitam o desenvolvimento de soluções com ML.
 - O surgimento de **serviços de computação em nuvem** ofereceu **acesso econômico e escalável** a recursos de computação.

Consequências do aumento da capacidade



Cloud TPU

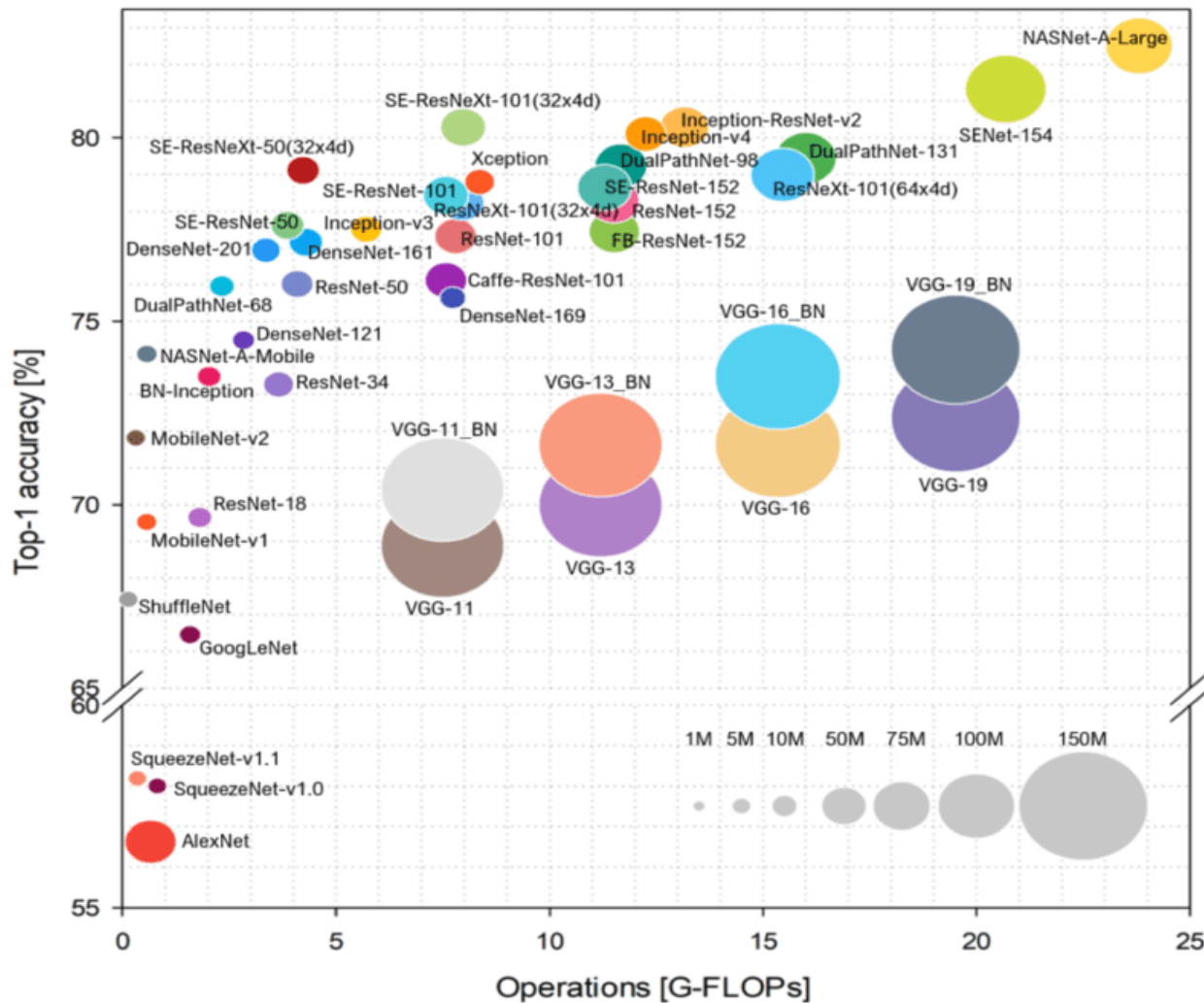
TinyML



- Entretanto, esses ***modelos*** mais e mais ***complexos, necessitam,*** consequentemente, de ***muito mais*** capacidade de ***armazenamento, energia, dispositivos de processamento*** mais ***poderosos*** e muito mais ***caros*** e que acabam ***ocupando grandes espaços.***
- ***Como podemos colocar tudo isso em um dispositivo tinyML?***

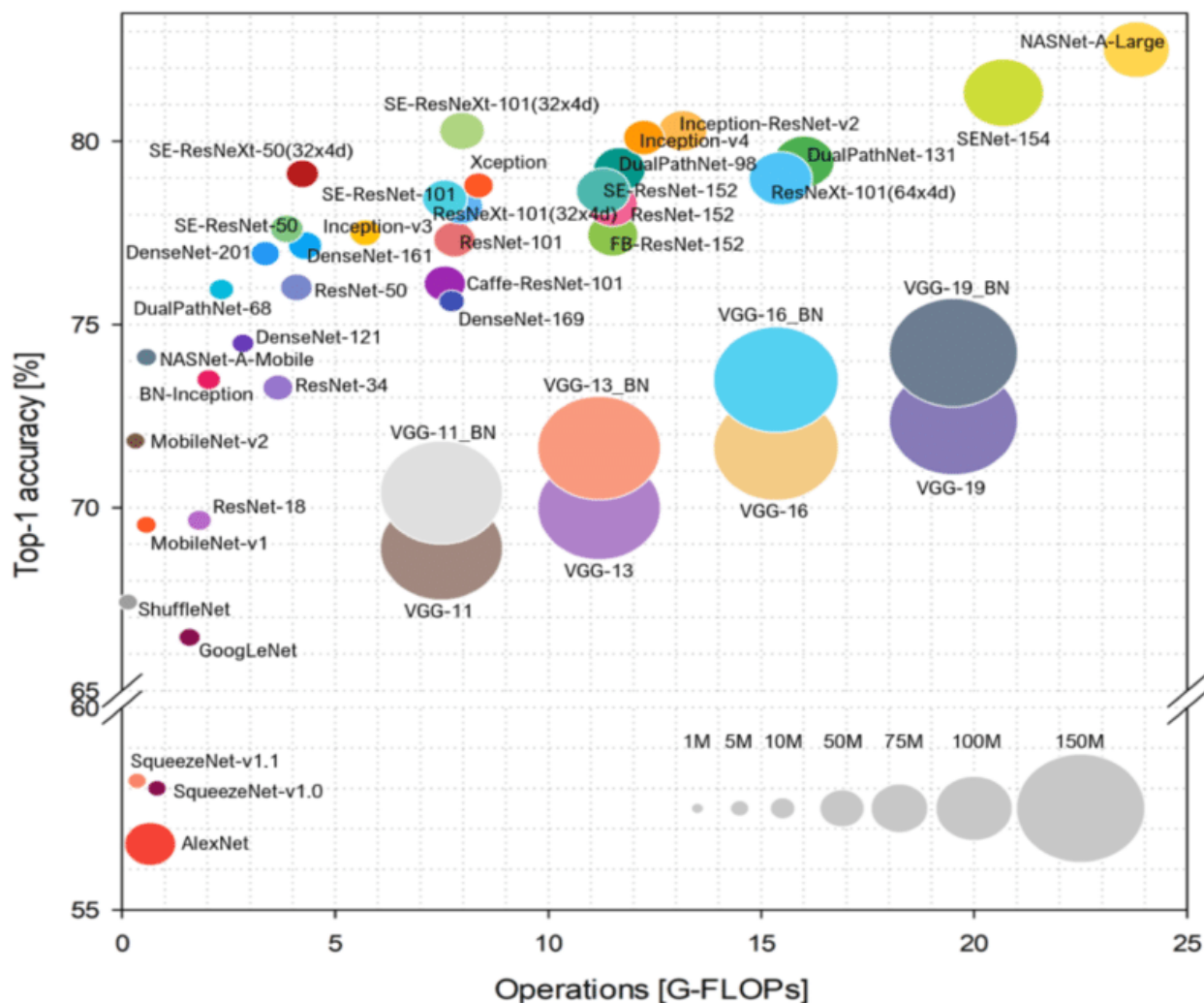
TPU: Tensor Processing Unit, é um ASIC altamente ***especializado para realizar cálculos matriciais e operações tensoriais***, o que acelera o treinamento e inferência de modelos de aprendizado de máquina, especialmente aqueles que usam o TensorFlow.

Evolução dos modelos de ML



- A figura ao lado mostra a **evolução dos modelos** de ML na tarefa de **classificação de imagem** usando o conjunto de dados ImageNet-1K (1000 classes) como **benchmark**.
- O eixo x mostra o **custo computacional** do modelo (i.e., quantidade de **flops** necessárias para uma classificação).
- O eixo y mostra a **acurácia** (taxa de acertos).
- O tamanho de cada círculo corresponde à **complexidade** (i.e., quantidade de parâmetros) do modelo.

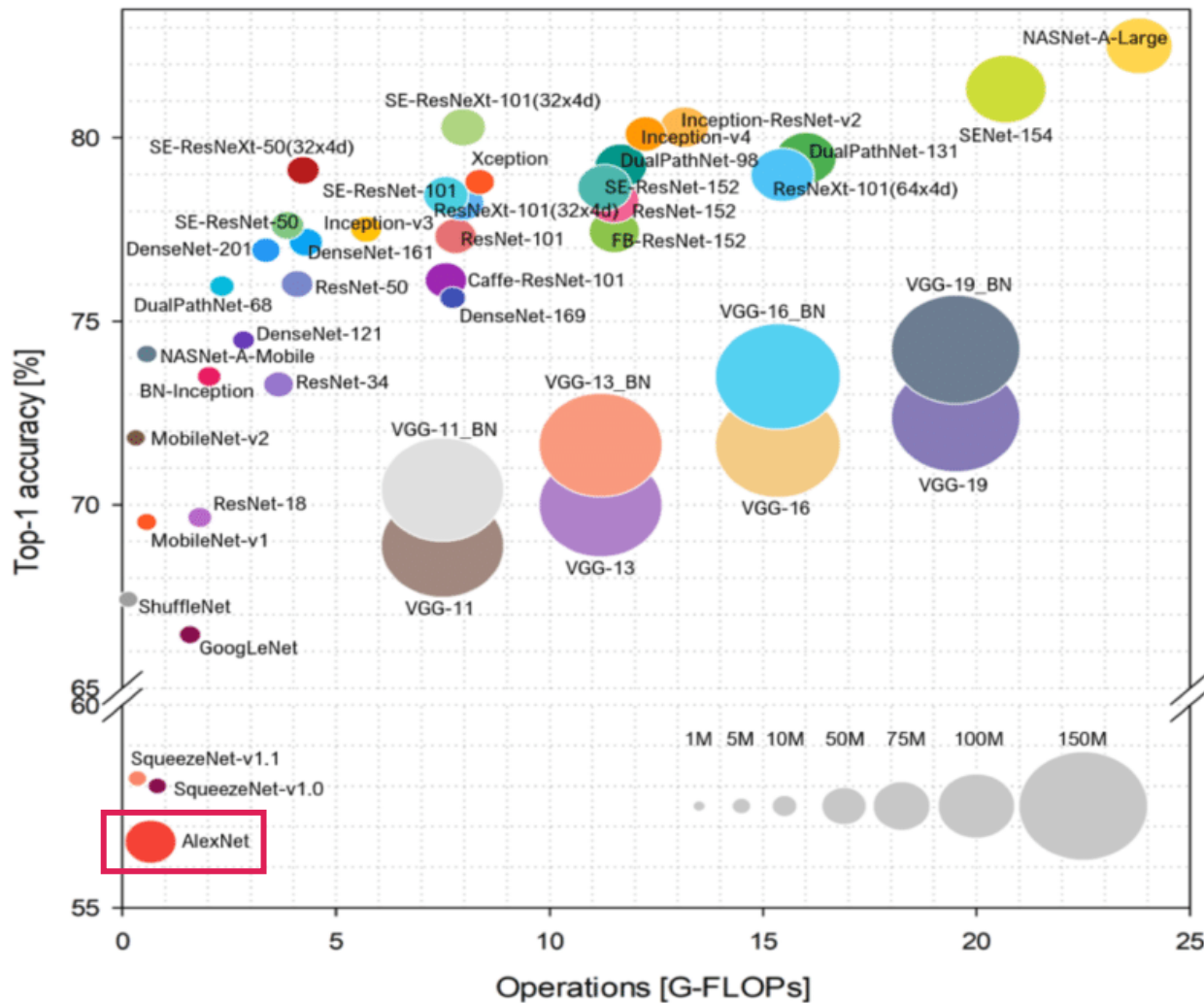
Evolução dos modelos de ML



- Para execução em dispositivos *tinyML*, devemos procurar por modelos com:
 - *Poucos parâmetros* (i.e., círculos pequenos), pois a quantidade de parâmetros está associada à ***quantidade de memória demandada pelo modelo***.
 - *Quantidade necessária de flops baixa*, pois a quantidade de operações está diretamente ***relacionada ao consumo de energia***. Além disso, a quantidade de operações do modelo deve ser suportada pelo poder computacional da CPU/GPU.
 - *Alta acurácia*.

Fonte: Bianco, Simone, et al. "*Benchmark analysis of representative deep neural network architectures.*" *IEEE access* 6 (2018): 64270-64277.

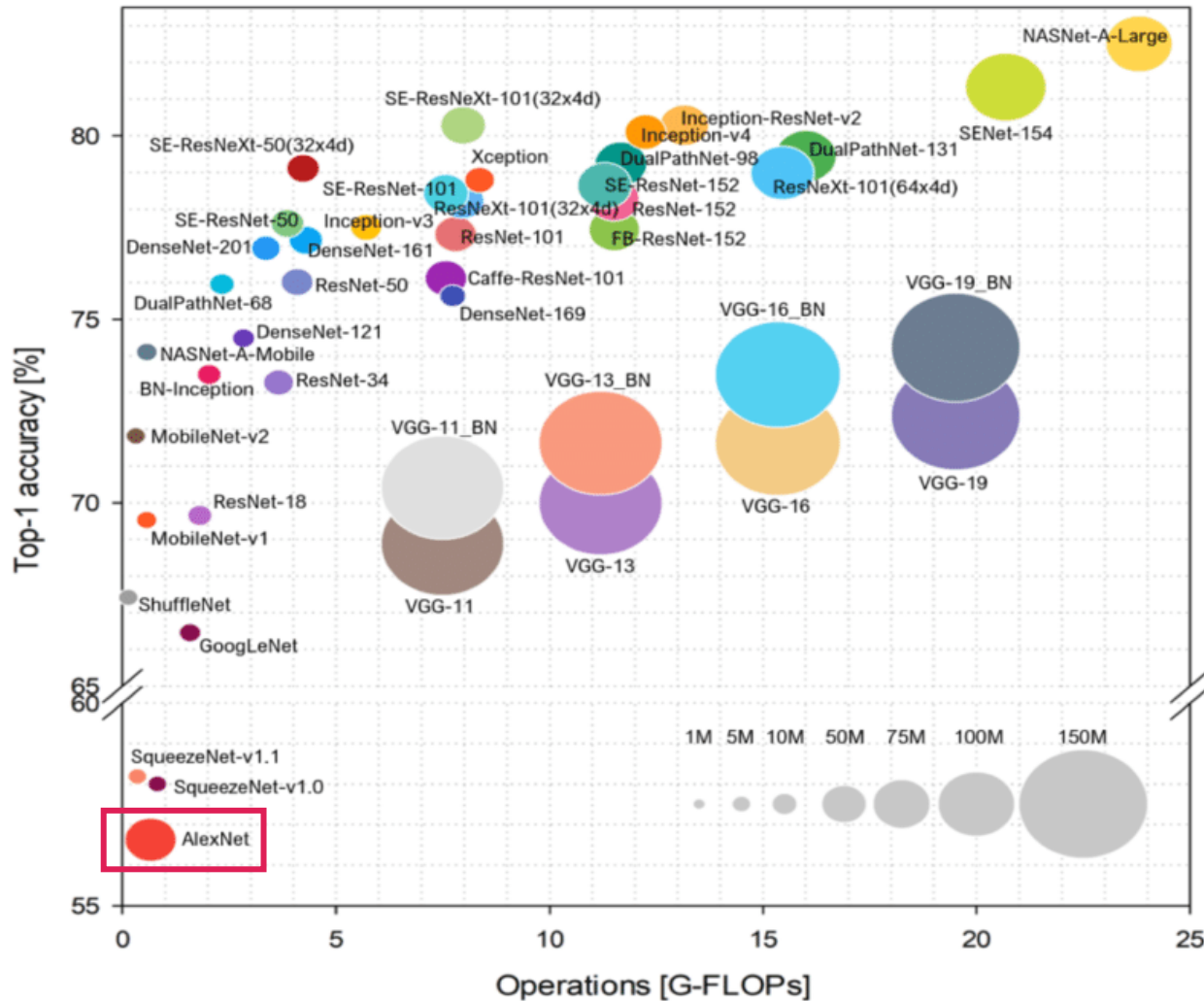
Evolução dos modelos de ML



AlexNet (Universidade de Toronto, Canadá - 2012)

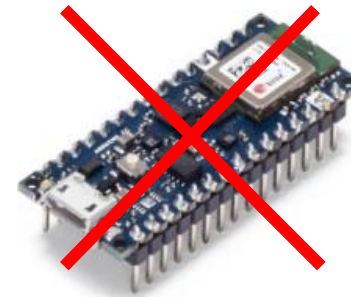
- Acurácia: 57.1%
- Tamanho: 61 MB
- Camadas: 5 convolucionais e 3 densas.

Evolução dos modelos de ML



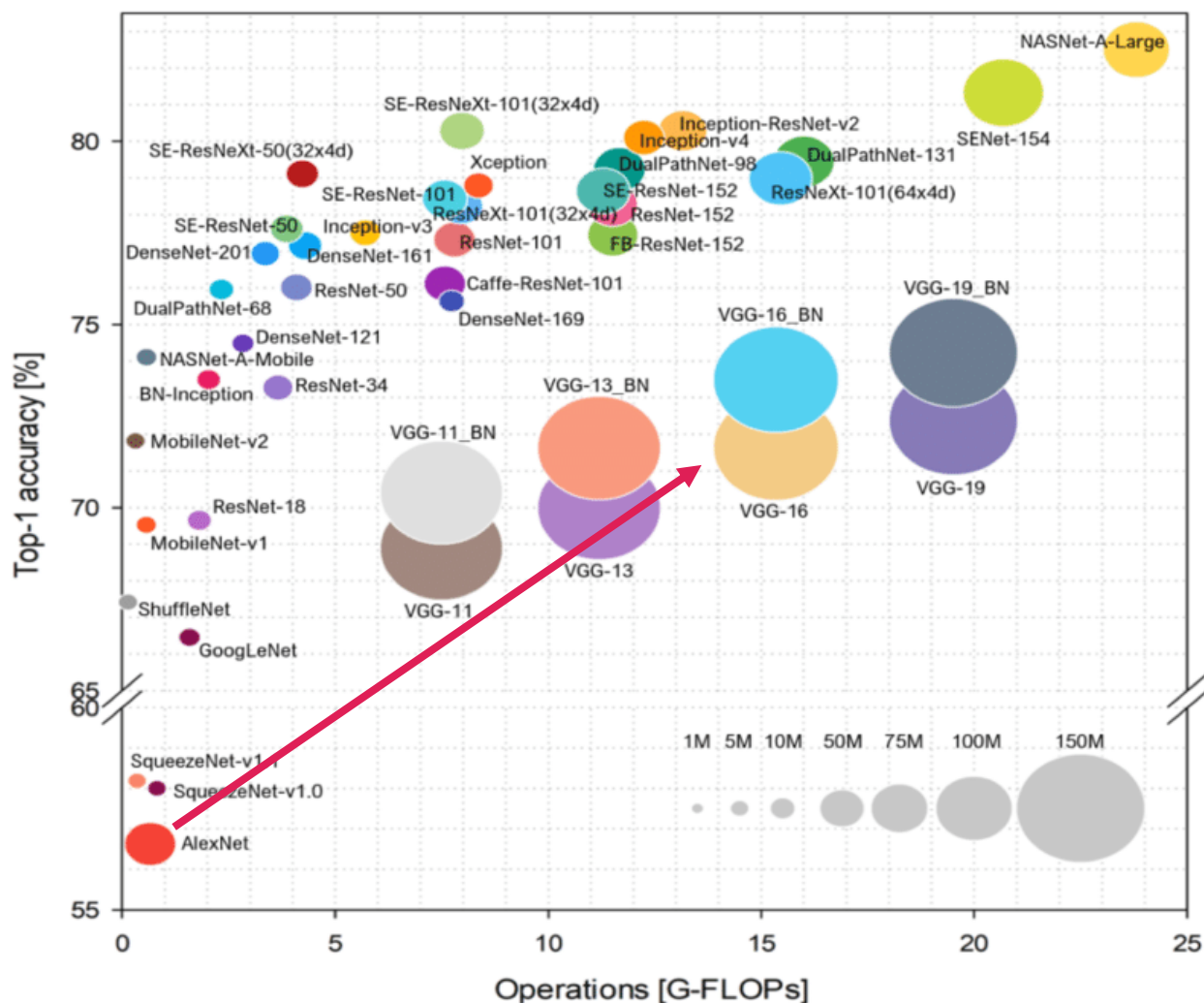
AlexNet (Universidade de Toronto, Canadá - 2012)

- Acurácia: 57.1%
- **Tamanho: 61 MB**



Memória RAM: 256 KB

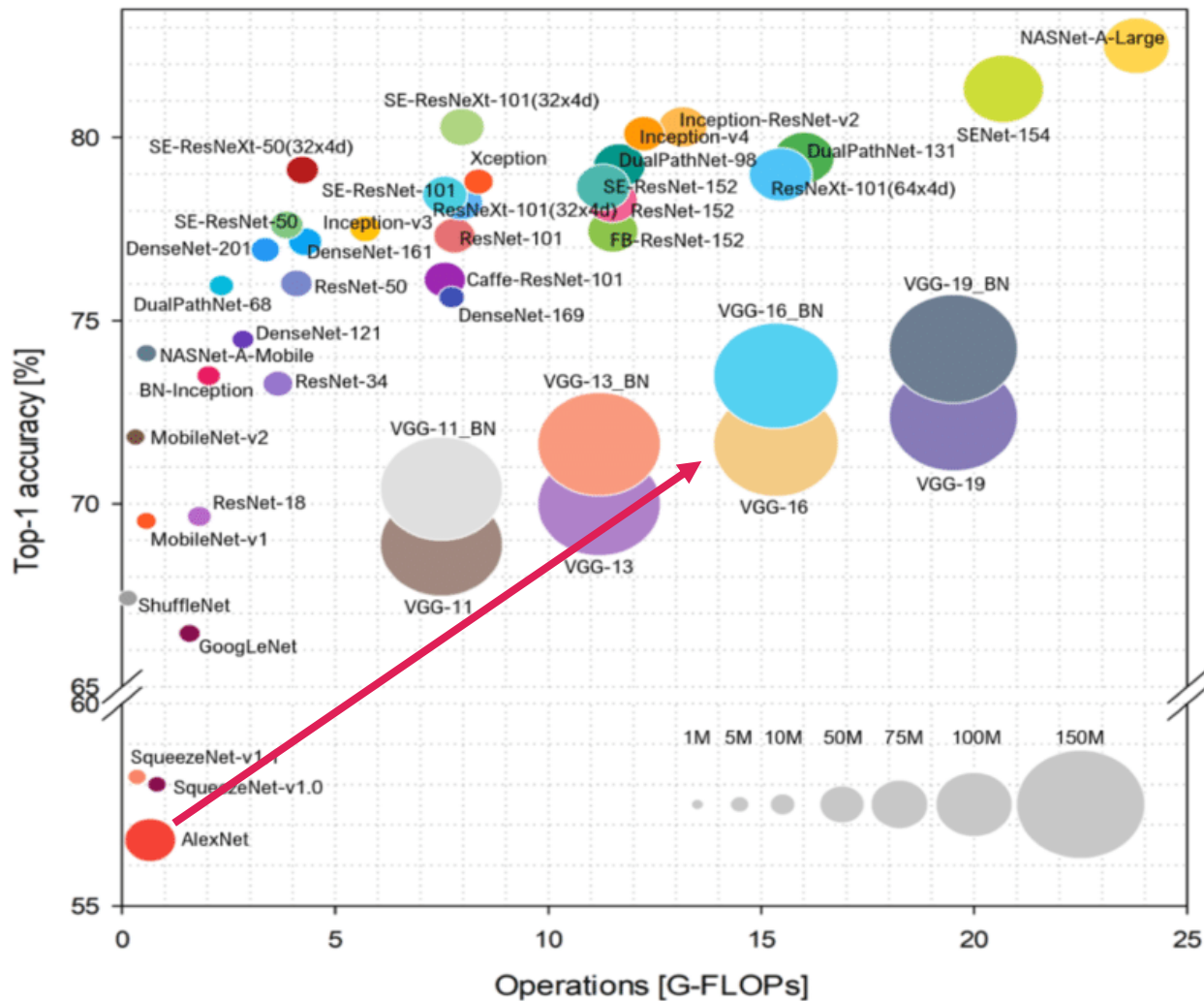
Evolução dos modelos de ML



VGG-16 (Universidade de Oxford, Reino Unido - 2014)

- Acurácia: 71.5%
- Tamanho: 528 MB
- Camadas: 13 convolucionais e 3 densas.
- Em dois anos, saímos de um modelo pequeno, requerendo poucos *flops* (baixo consumo), mas com baixa acurácia para modelos mais precisos, porém mais de 8 vezes maiores e requerendo 15 vezes mais *flops* (alto consumo).

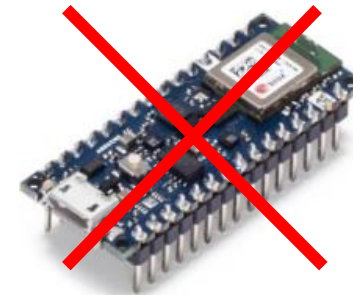
Evolução dos modelos de ML



VGG-16 (Universidade de Oxford, Reino Unido - 2014)

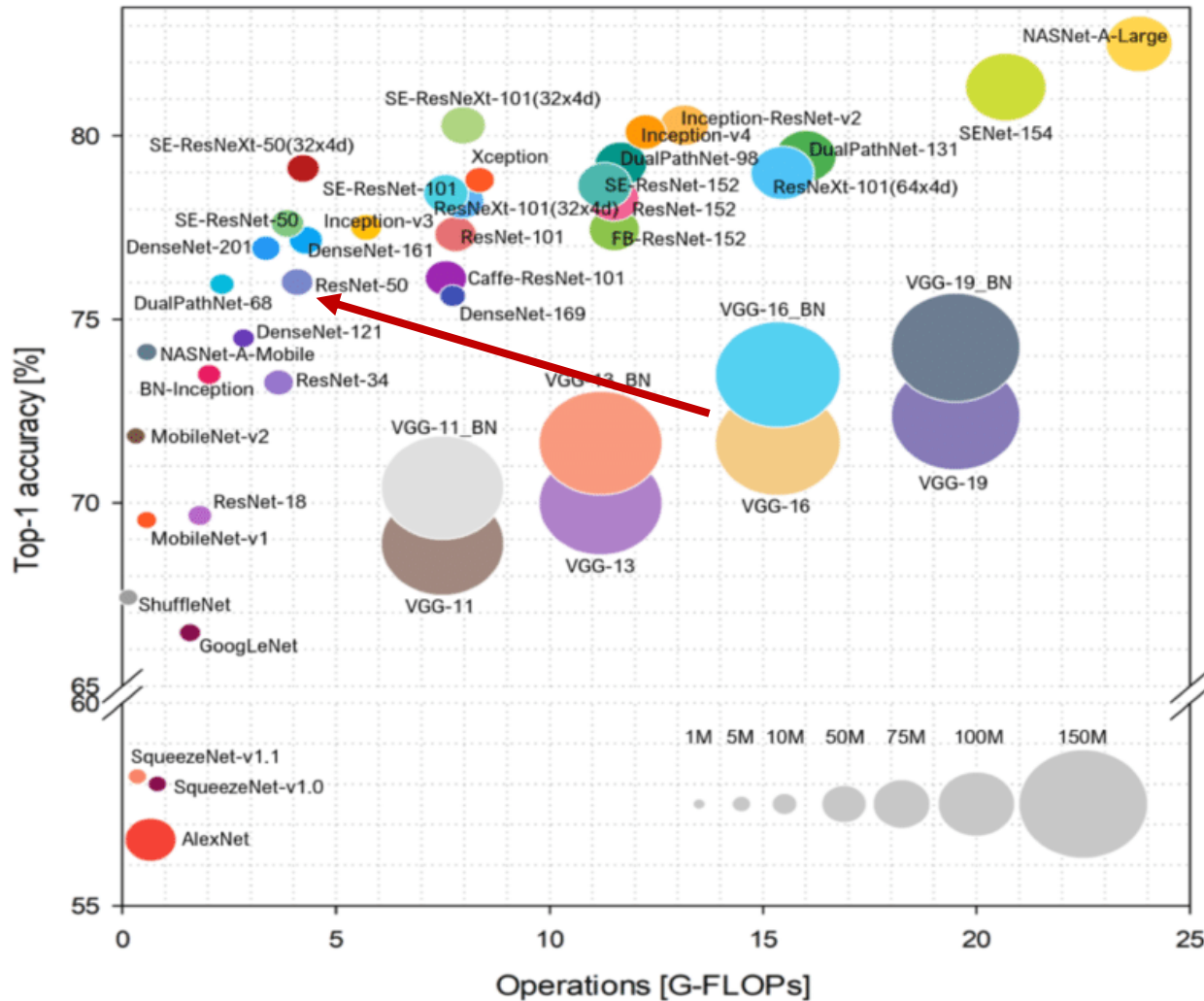
■ **Acurácia: 71.5%**

■ **Tamanho: 528 MB**



Memória RAM: 256 KB

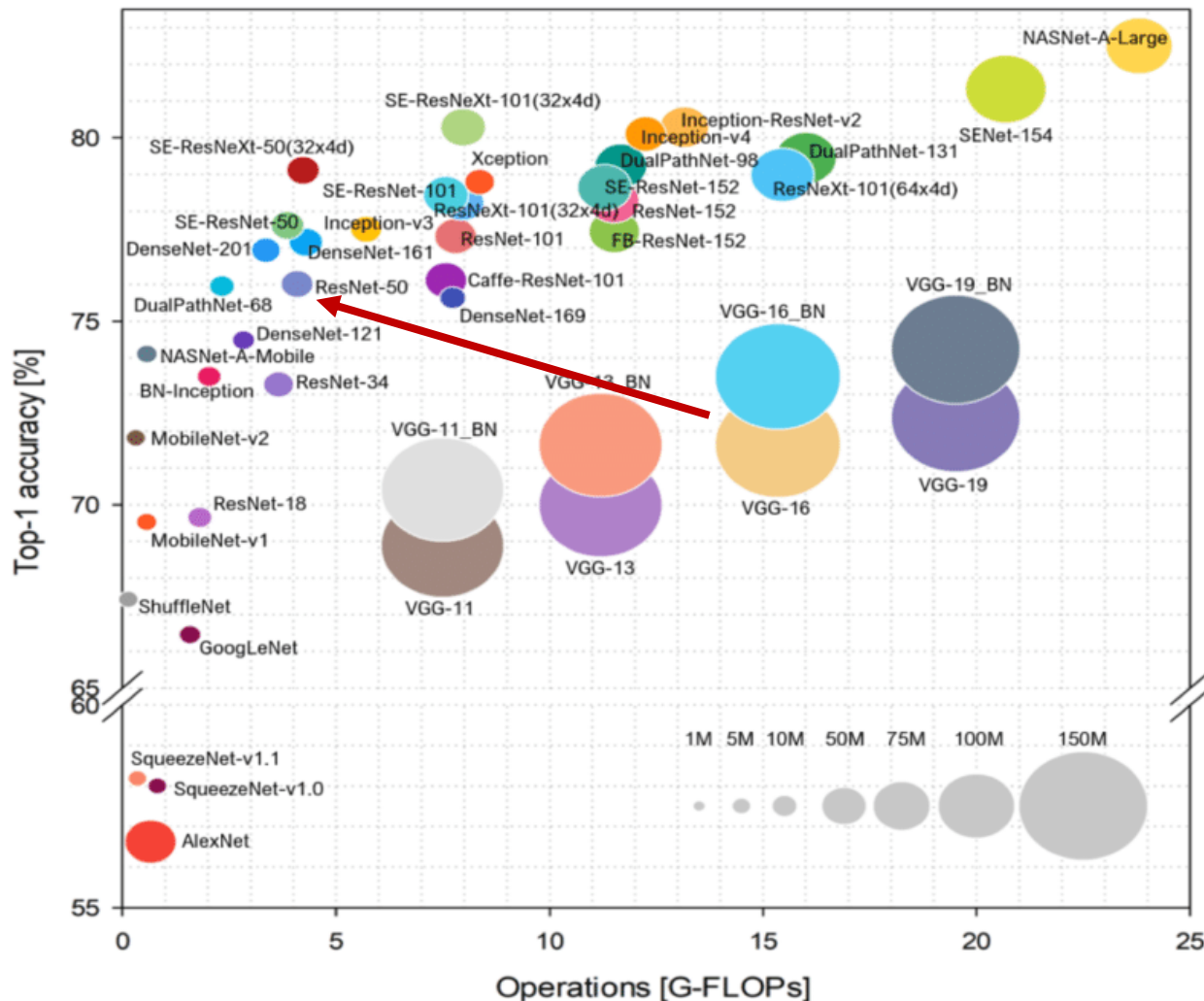
Evolução dos modelos de ML



ResNet-50 (Microsoft - 2015)

- Acurácia: 75.8%
- Tamanho: 22.7 MB
- Camadas: 48 convolucionais, 2 de *pooling* (agrupamento) e 1 densa.

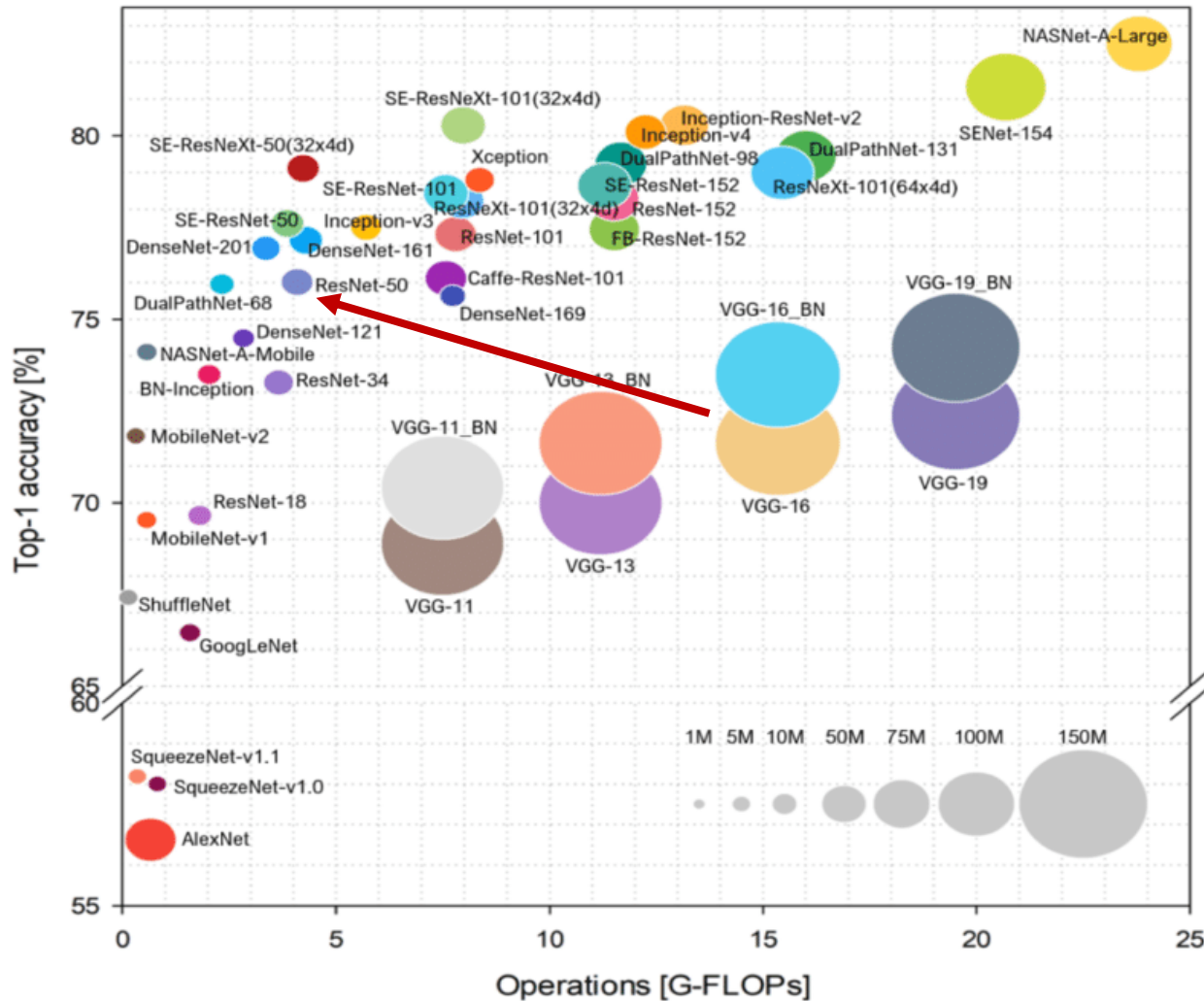
Evolução dos modelos de ML



ResNet-50 (Microsoft - 2015)

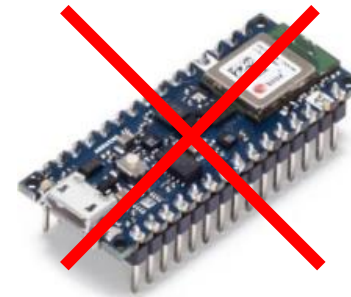
- Arquitetura muito mais eficiente do que a VGGNet pois introduz
 - **Dropout**: desliga aleatoriamente algumas das conexões da rede durante o treinamento para evitar o sobreajuste.
 - **Blocos residuais**: são atalhos entre camadas convolucionais que mitigam o problema do desaparecimento do gradiente.
- Em **um ano**, saímos de um modelo **extremamente grande** e que **requer muitos flops**, mas razoavelmente preciso para um modelo **23 vezes menor, mais preciso** e necessitando de **3 vezes menos flops**.

Evolução dos modelos de ML



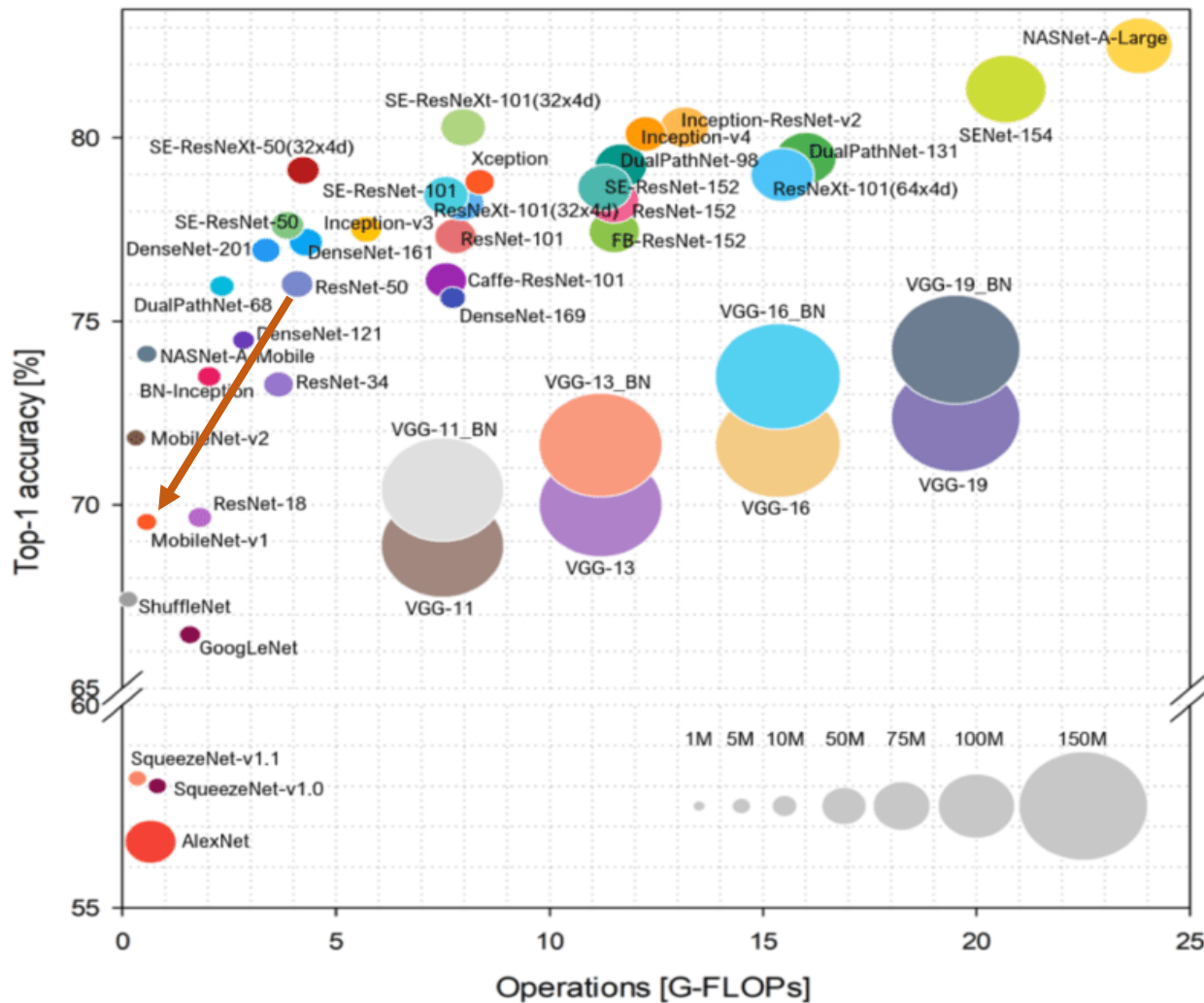
ResNet-50 (Microsoft - 2015)

- Acurácia: 75.8%
- Tamanho: 22.7 MB



Memória RAM: 256 KB

Evolução dos modelos de ML



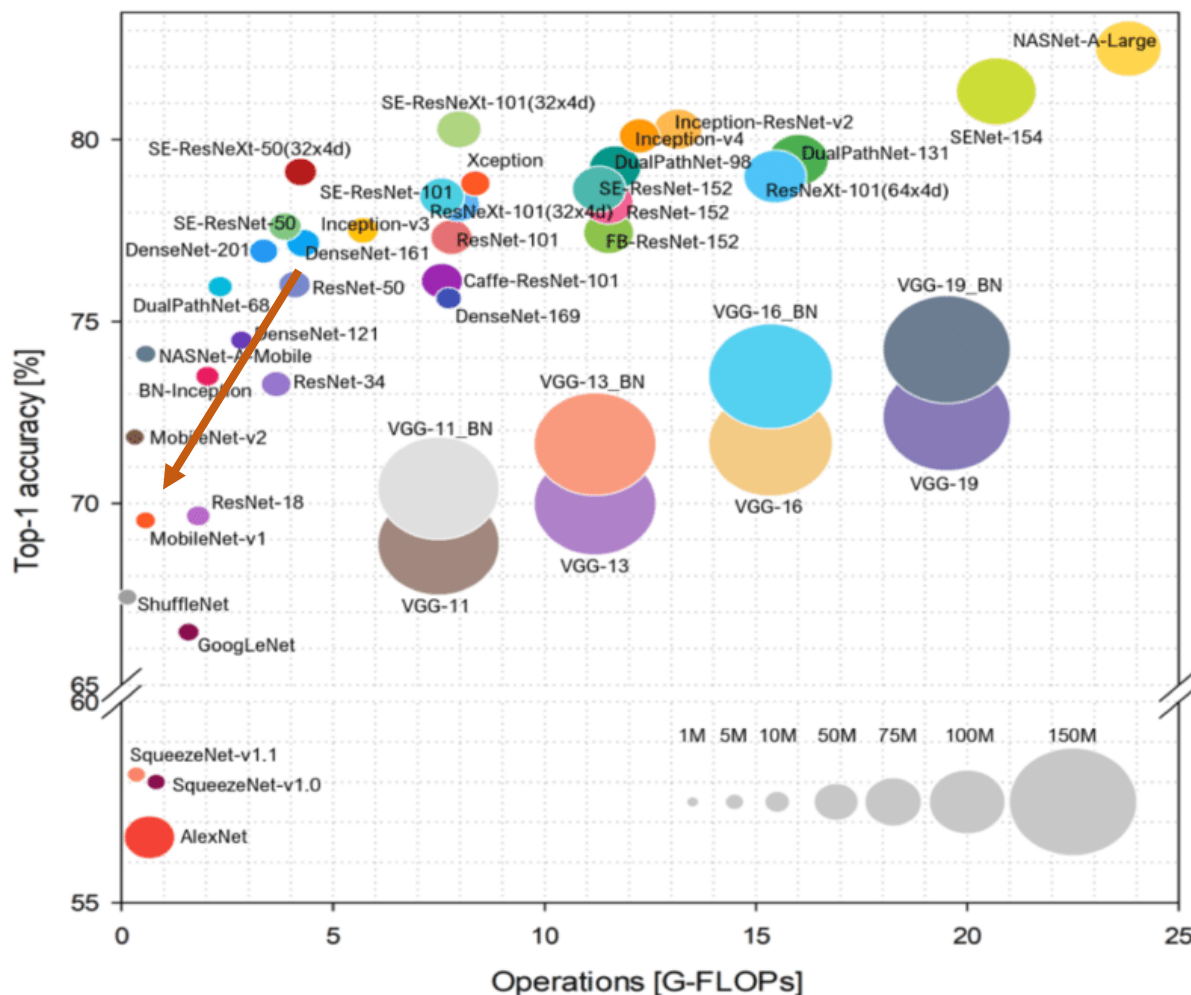
MobileNetv1 (Google - 2015)

■ **Acurácia: 70.6%**

■ **Tamanho: 16.9 MB**

- Camadas: 27 convolucionais e 2 densas.
- Adequada para aplicações de visão computacional em **dispositivos móveis e embarcados** (edgeML).

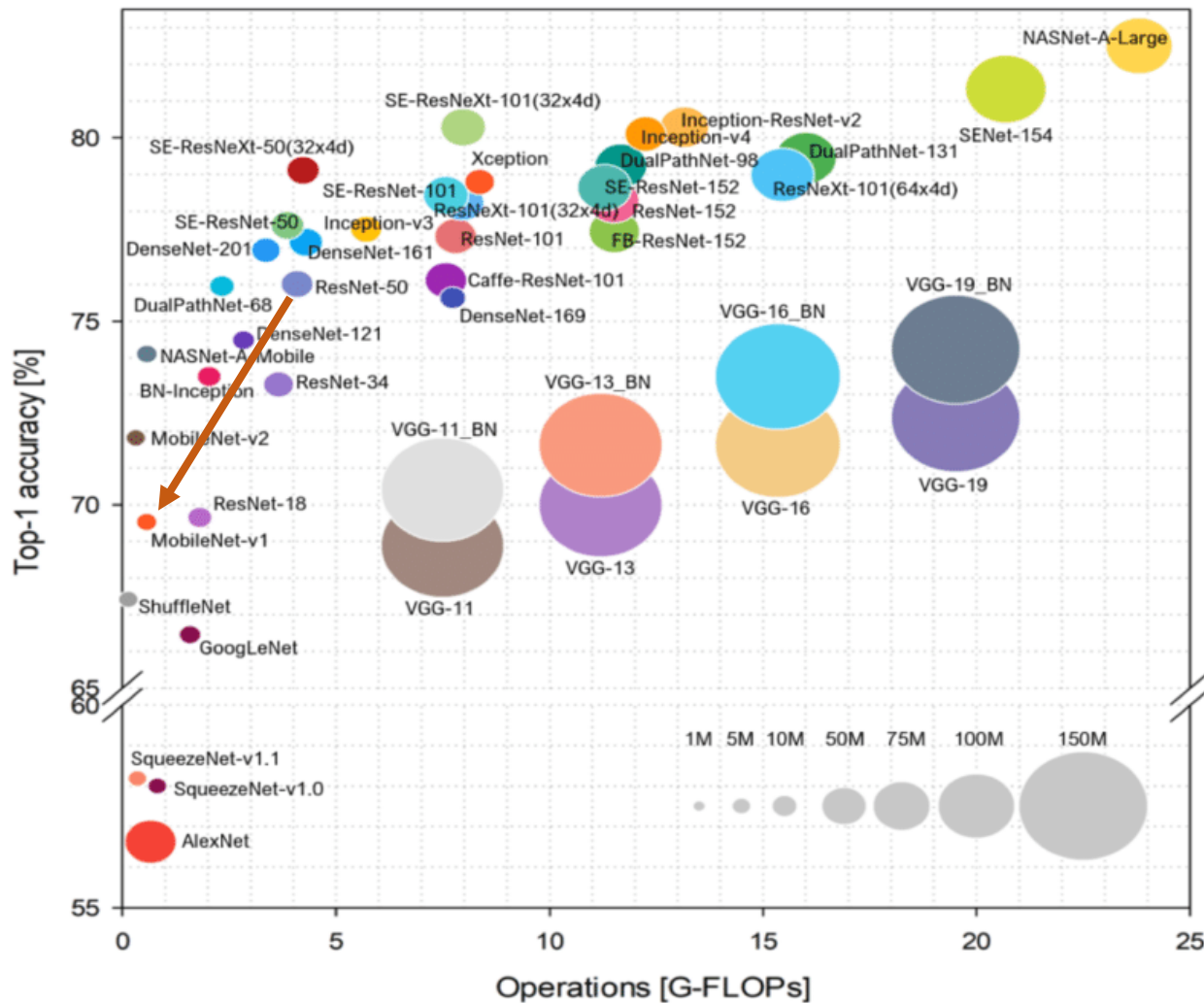
Evolução dos modelos de ML



MobileNetv1 (Google - 2015)

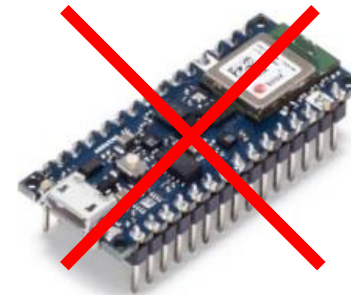
- Usa camadas de **convolução separável em profundidade** em vez de convolução tradicional.
- **Divide** a convolução tradicional em **duas etapas**: uma **convolução em profundidade** (que lida com canais) seguida de uma **convolução em ponto** (que lida com pixels individuais).
- Isso **reduz** drasticamente o **tamanho do modelo**, a **quantidade de cálculos** e **melhora sua eficiência computacional**.

Evolução dos modelos de ML



MobileNetv1 (Google - 2015)

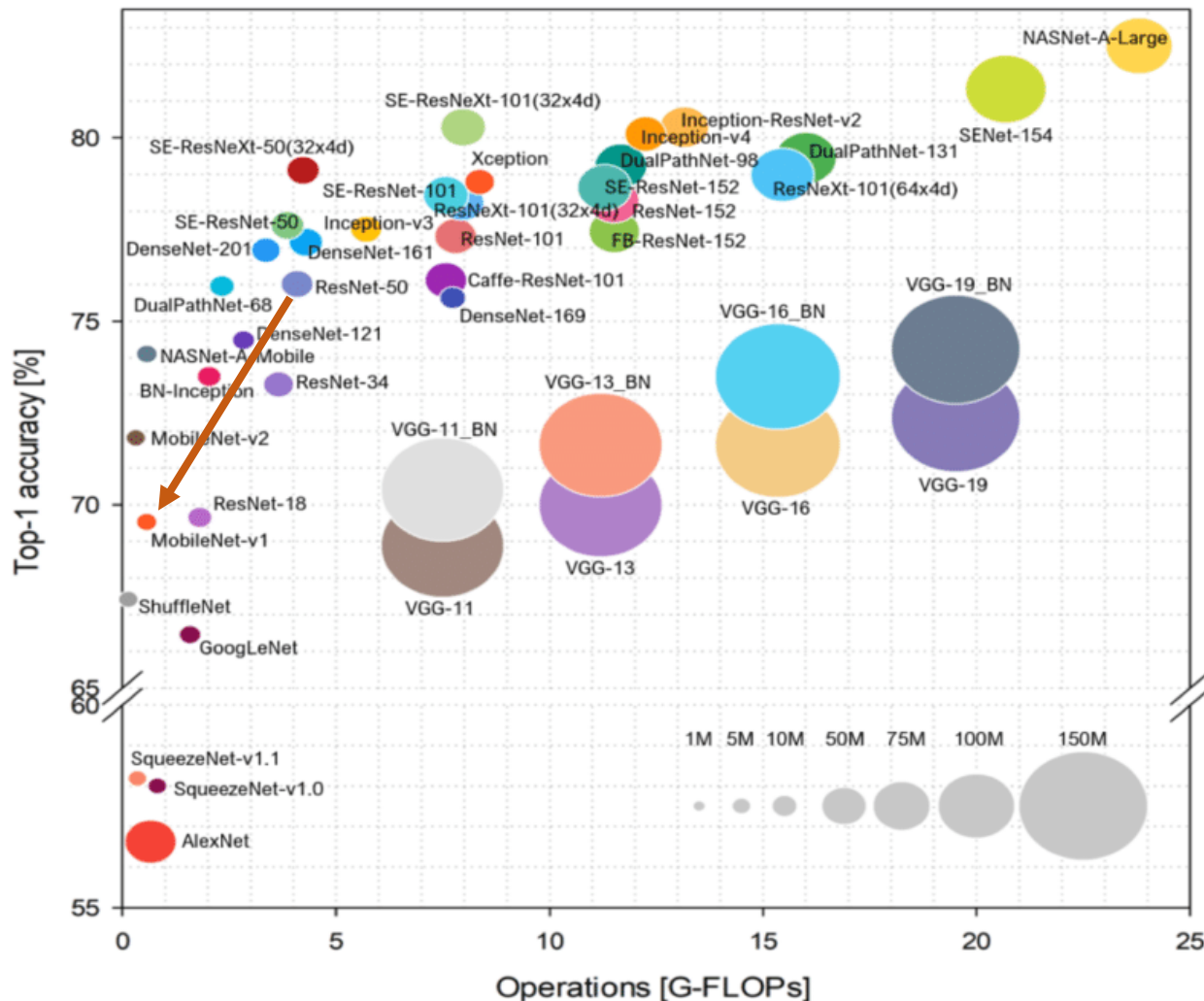
- **Acurácia: 70.6%**
- **Tamanho: 16.9 MB**



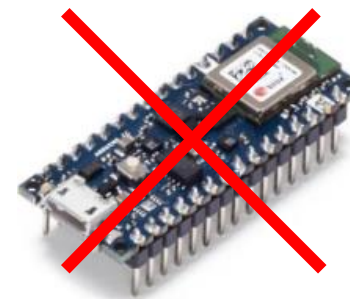
Memória RAM: 256 KB

Os modelos, até este ponto, ainda eram muito grandes para dispositivos IoT com recursos restritos.

Evolução dos modelos de ML

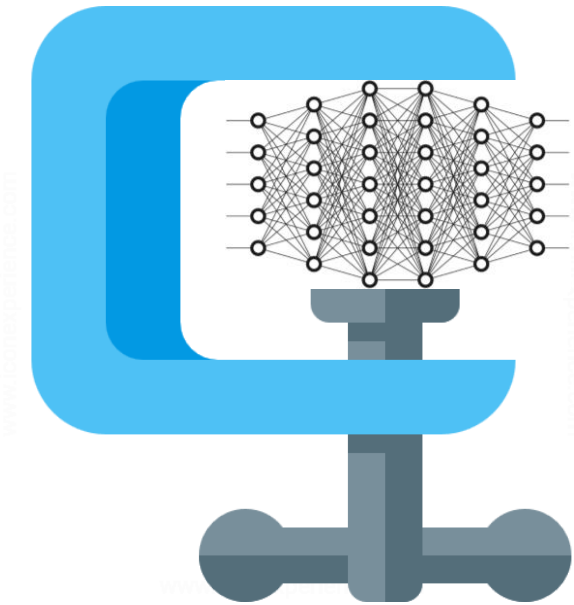


- Os modelos até este ponto ainda eram muito grandes para dispositivos IoT com recursos restritos.



Memória RAM: 256 KB

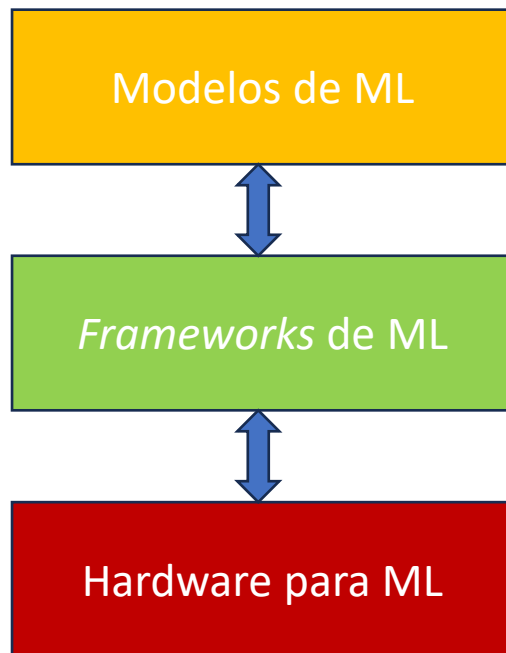
Como executar modelos de ML em dispositivos tinyML?



- Nitidamente, mesmo os menores modelos, não cabem em dispositivos tinyML (IoT).
- Portanto, surge a pergunta:

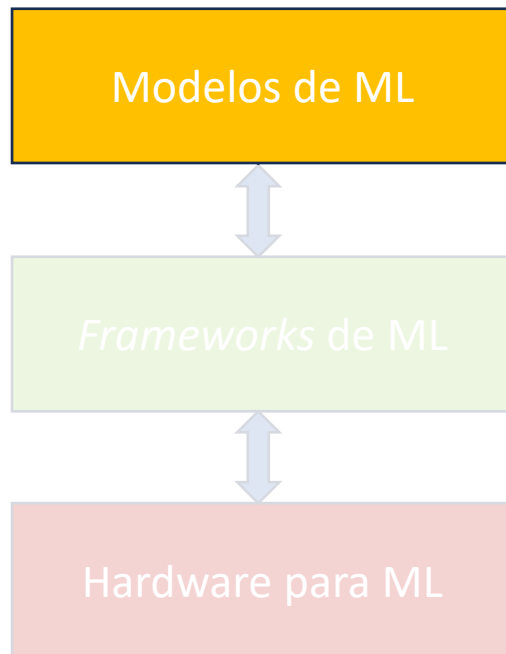
Como podemos *reduzir ainda mais* esses modelos para que *caibam na memória* destes dispositivos, *possam ser executados* por eles e *consumam pouca energia*?

Como executar modelos de ML em dispositivos tinyML?



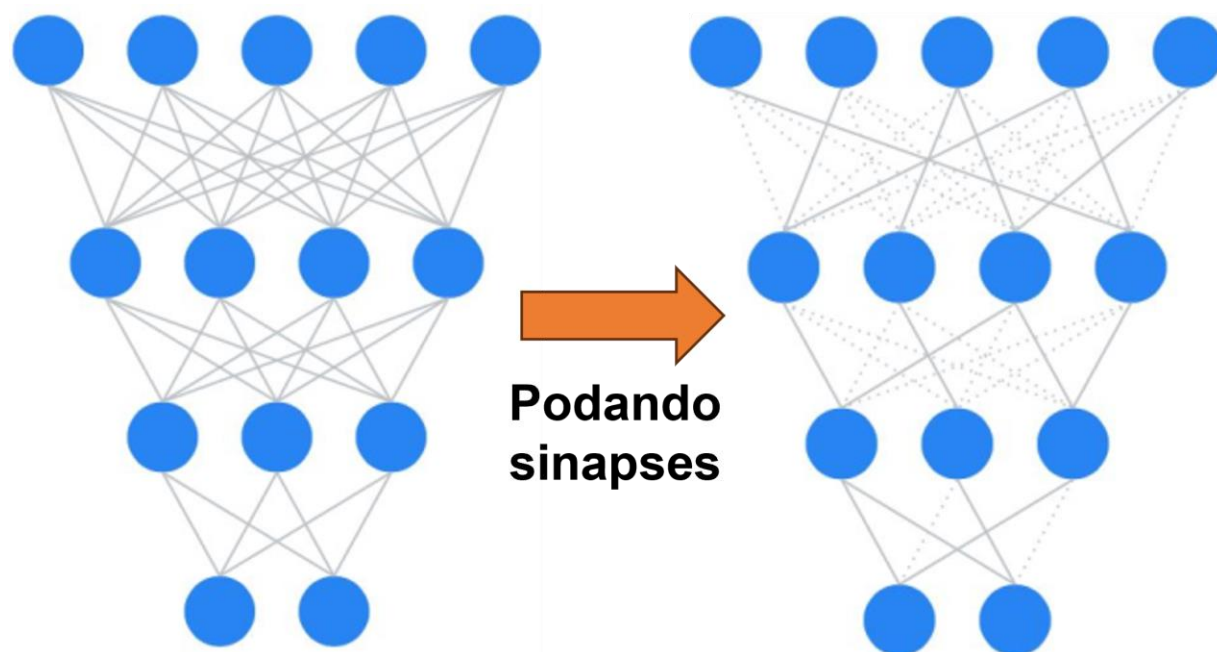
- Podemos lançar mão de três abordagens:
 - Redução/Compressão dos modelos.
 - *Frameworks* mais enxutos e eficientes.
 - Hardwares específicos/customizados para ML.

Técnicas de compressão dos modelos



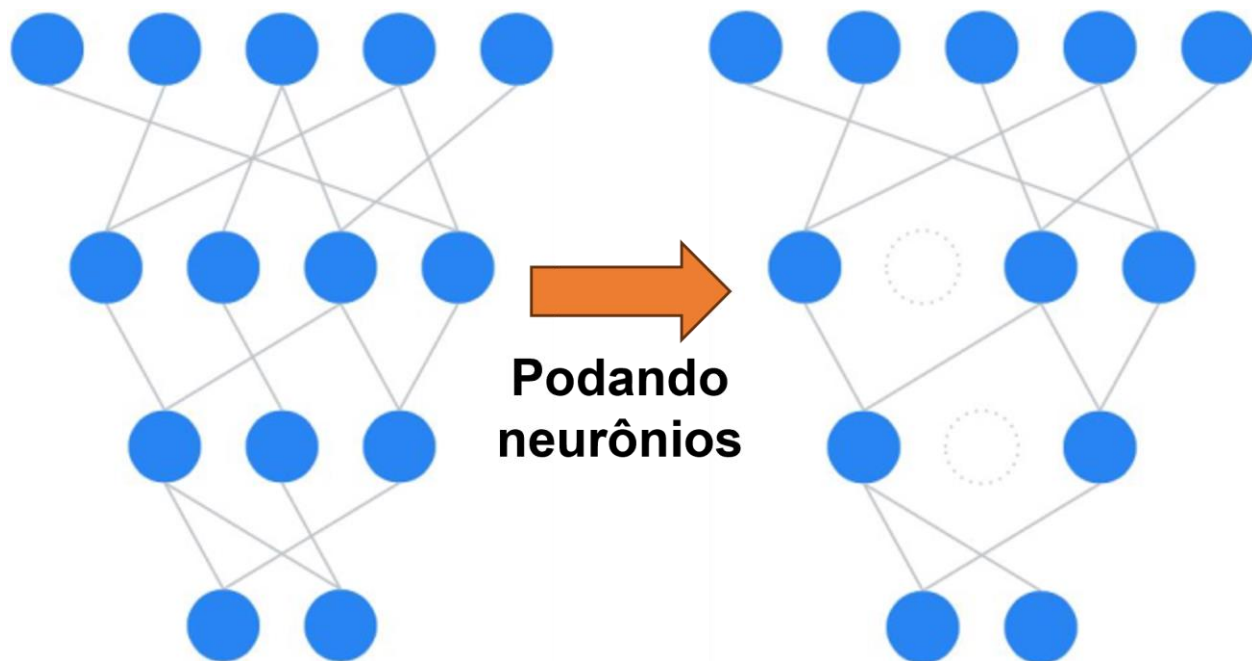
- Em geral, podemos utilizar três técnicas para reduzir o tamanho dos modelos sem perder muito de seu desempenho:
 - *Pruning* (ou poda);
 - Quantização;
 - *Clustering*;
 - *Knowledge Distillation* (ou destilação de conhecimento).

Pruning



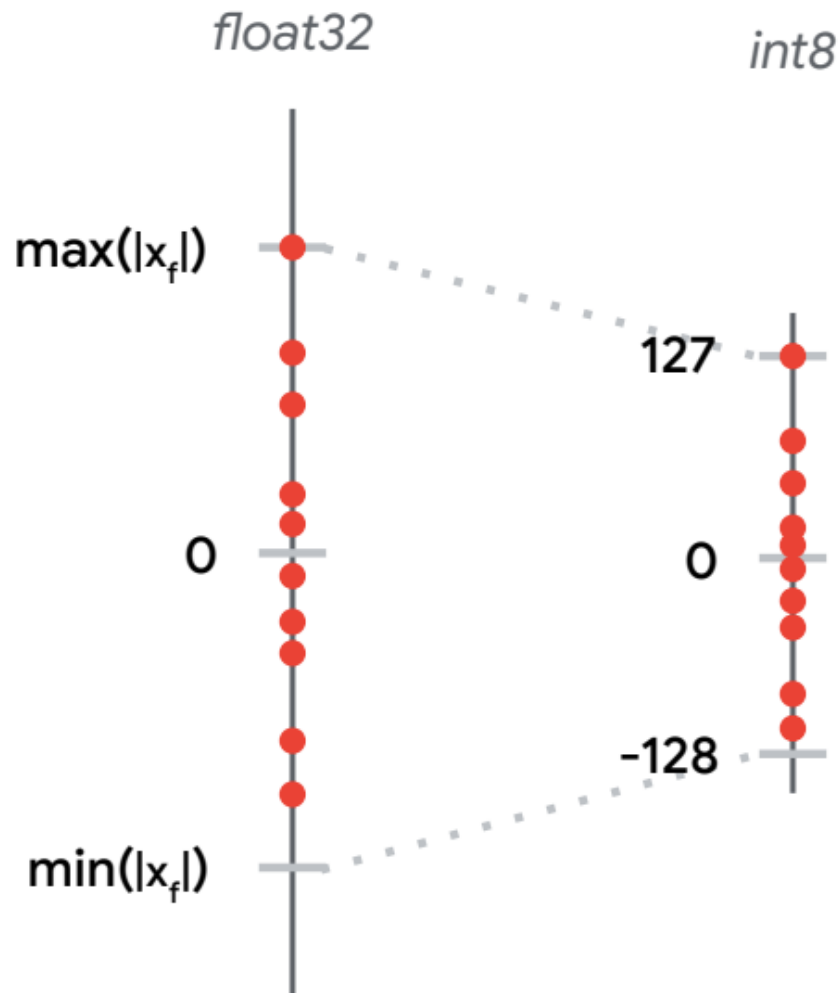
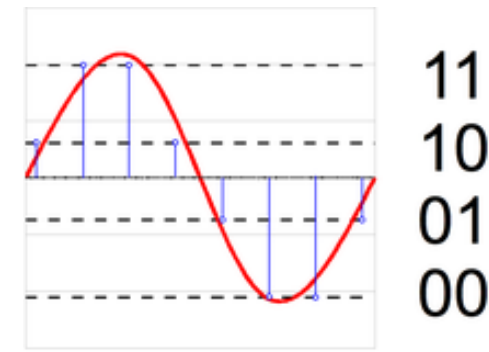
- **Remove conexões** para **reduzir** a quantidade de **cálculos** necessários (i.e., reduz a demanda computacional), e diminuir o **tamanho do modelo**.
- Técnica aplicada após o treinamento.
- Essa remoção pode ser baseada em critérios como, por exemplo, a **magnitude dos pesos**.
 - **Pesos com magnitudes próximas de zero são removidos** do modelo pois podem não interferir drasticamente no seu desempenho.

Pruning



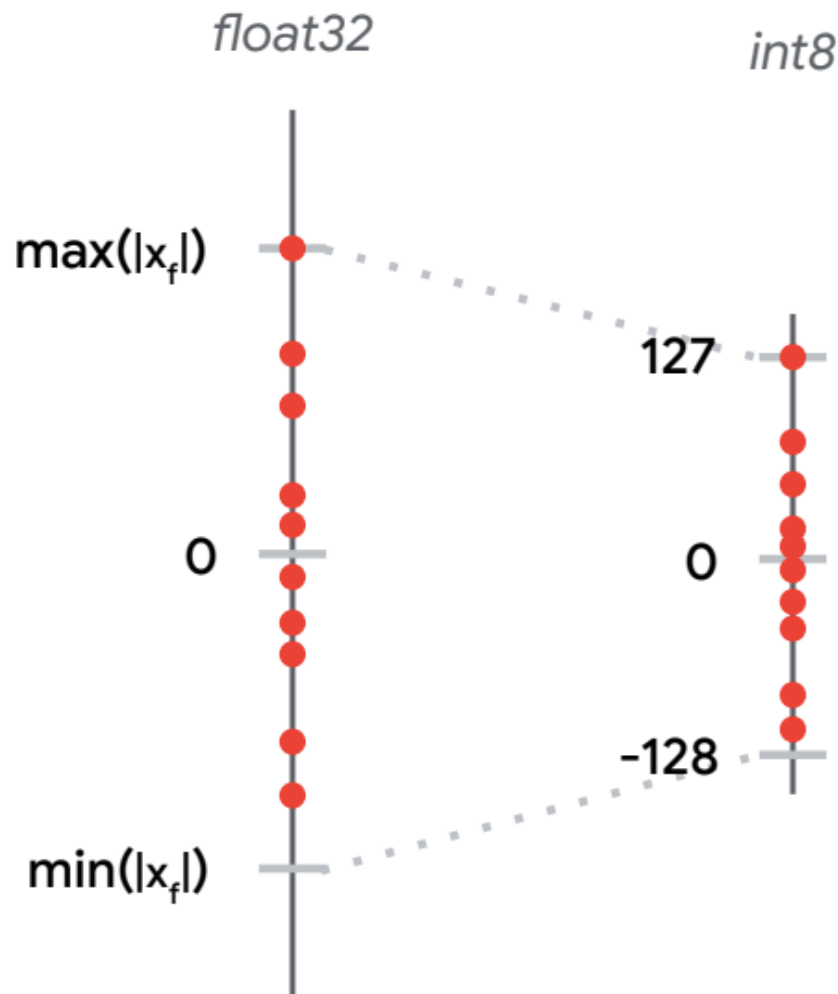
- Podemos também ser capazes de **remover neurônios completamente**.
- O modelo final **pode** continuar com o **mesmo desempenho ou perder parte dele**.
- Portanto, é preciso **testar e encontrar um balanço** entre redução do modelo e seu desempenho.
- Ajuda a **mitigar problemas de sobreajuste**, melhorando a generalização do modelo.
 - Modelos complexos tendem a sobreajustar.
- É uma das técnicas que utilizaremos em breve.

Quantização



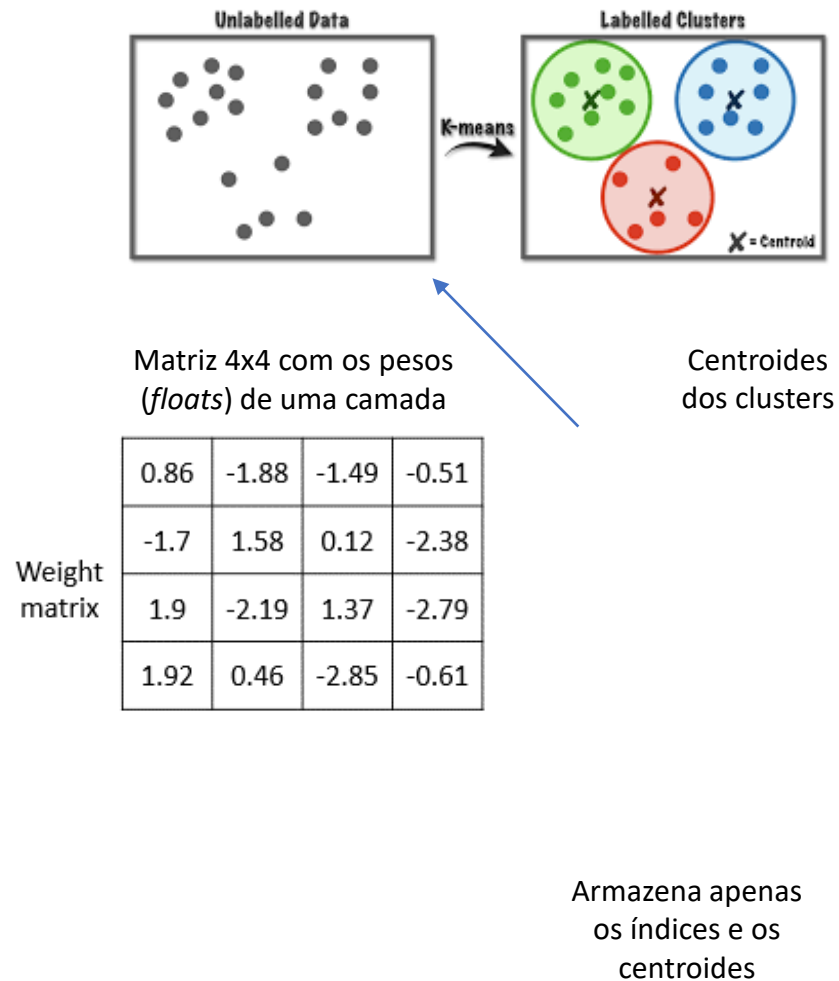
- Lembrando que sistemas embarcados ***não têm hardware sofisticado*** (e.g., FPU) e realizam apenas ***operações simples***.
- Assim, outra técnica que iremos usar bastante é chamada de ***quantização***.
 - É a ***modificação da representação numérica*** dos valores envolvidos nos cálculos.
- Ela ***reduz a precisão dos números*** usados para representar os parâmetros (pesos, ativações, etc.) dos modelos.
- Quantizar significa ***dividir o intervalo de variação de um número em um conjunto discreto de valores***.

Quantização



- A **quantização** transforma um número com um **grande intervalo de variação em outro com um intervalo menor**.
 - Por exemplo, transformar uma variável do tipo `float32`, que ocupa 4 bytes e tem intervalo $\pm 3.4 \times 10^{38}$, em uma do tipo `int8`, que ocupa 1 byte e tem intervalo de -128 a 127.
 - Nesse caso, temos uma redução de 4 vezes.
- A quantização **pode reduzir o desempenho do modelo**, mas, em muitos casos, ela **não é significativa** e **pode ser tolerada** quando desejamos utilizar dispositivos tinyML.

Clustering ou compartilhamento de pesos



- O *clustering* de pesos reduz o **tamanho de armazenamento e de transferência através da rede** de modelos de ML.
- O *clustering* reduz o tamanho do modelo substituindo **pesos semelhantes em uma camada por um mesmo valor** (i.e., o centroide mais próximo).
- Assim, podemos armazenar os índices e os centroides:
 - Neste caso, reduzimos o tamanho da matriz de 16 *floats* diferentes para 4 *floats* distintos e 16 índices de 2 bits.

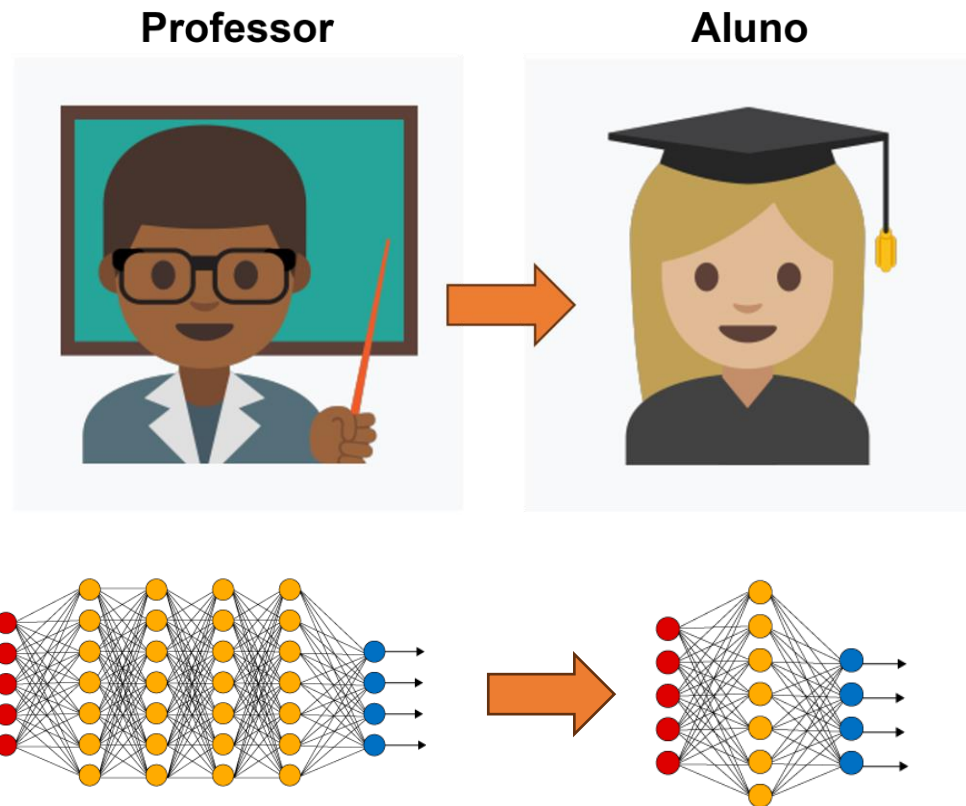
Clustering ou compartilhamento de pesos

Model	Dataset	Size of TensorFlow Lite Model (Compressed Via Zip)			Top-1 Accuracy		
		Original	Clustered	Reduction	Original	Clustered	Delta
ConvNet	MNIST	0.57 MB	0.09 MB	6.3x	99.40%	98.78%	-0.62%
MobileNetV2	ImageNet	12.90 MB	7.00 MB	1.8x	72.29%	72.31%	+0.02%
			2.60 MB	5.0x		69.33%	-2.96%
MobileNetV1	ImageNet	14.96 MB	8.42 MB	1.7x	71.02%	70.62%	-0.40%
			2.98 MB	5.0x		66.07%	-4.95%
DS-CNN-L	Speech Commands v0.02	1.50 MB	0.30 MB	5.0x	95.03%	94.71%	-0.32%



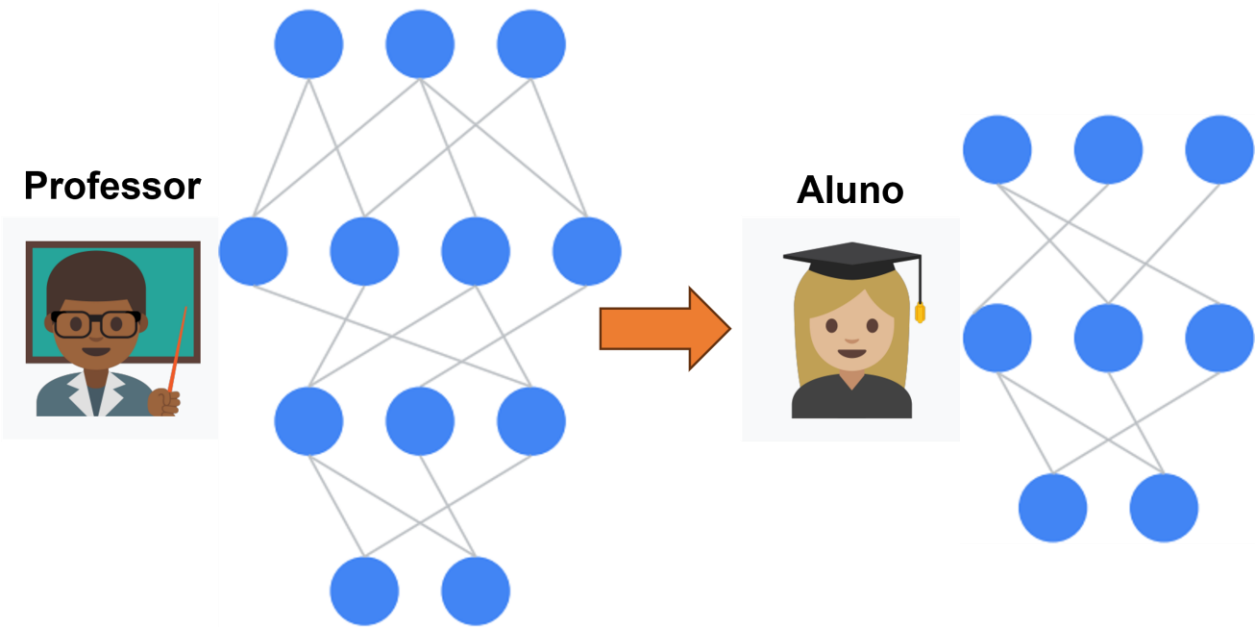
- Mesmo que armazenássemos na matriz os centroides de cada peso (i.e., 16 *floats*) ao invés dos índices, ferramentas de compactação, como o *zip*, podem aproveitar a redundância nos dados para obter maior compactação.
- A tabela ao lado mostra alguns resultados de *clustering*.
- Notem que os tamanhos são reduzidos, mas que há uma redução na acurácia.

Knowledge distillation ou destilação de conhecimento



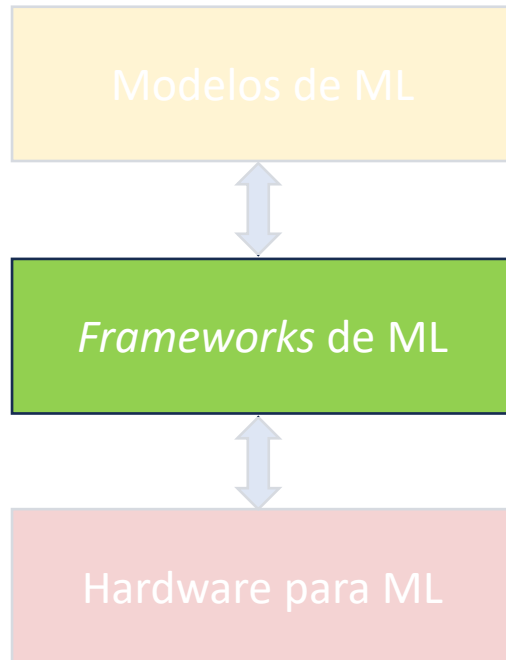
- É técnica de treinamento de modelos de ML em que um **modelo maior e mais complexo**, conhecido como "**professor**", é usado para **ensinar** (treinar) um **modelo menor e mais simples**, conhecido como "**aluno**".
- O objetivo é transferir o conhecimento ou a "**sabedoria**" do **professor** para o **aluno**, de forma que o **aluno** possa **obter desempenho comparável ou até mesmo melhor** do que o do **professor**, mas com **menor complexidade**.
- Em outras palavras, queremos que o aluno aprenda a **generalizar** como o professor.

Knowledge distillation



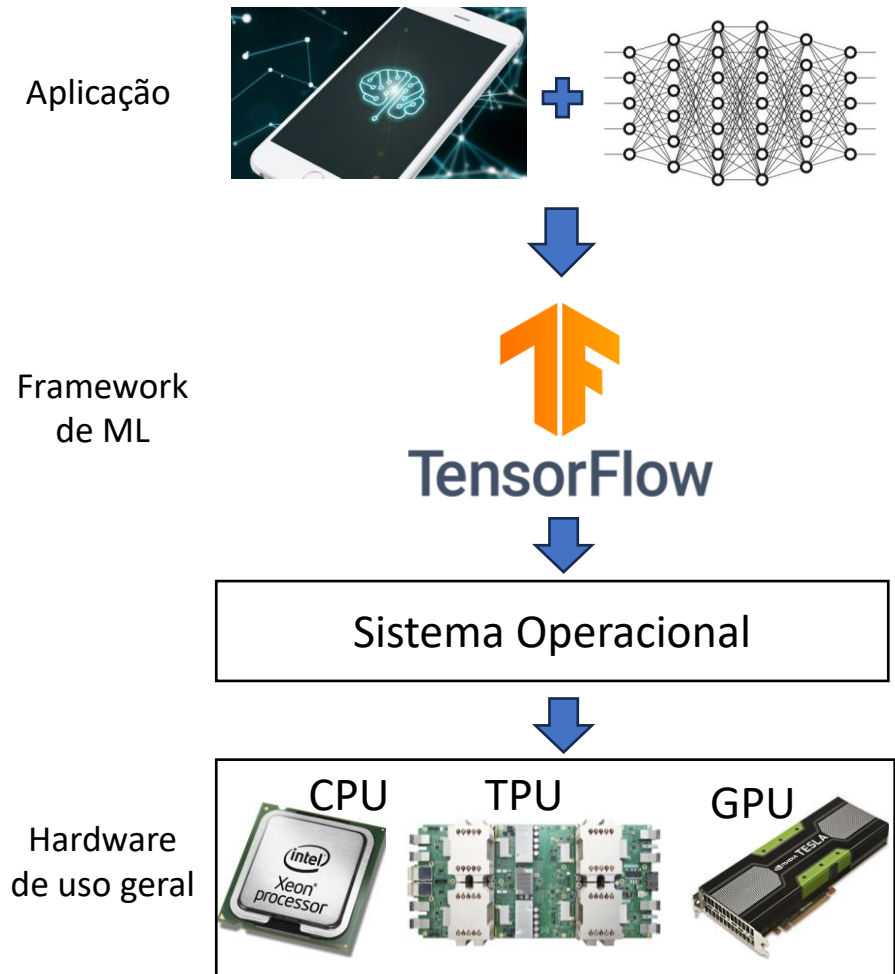
- O processo envolve alimentar o **mesmo conjunto de treinamento para ambos** os modelos e usar a **saída do professor como um alvo suave** (i.e., probabilidades) ao invés de um rígido para guiar o treinamento do modelo aluno.
- Dessa forma, o **aluno é incentivado a aprender com base nas características sutis e nuances aprendidas pelo professor**, melhorando assim sua capacidade de generalização e desempenho.

Frameworks de ML



- Um **framework** de ML é uma biblioteca ou um conjunto de ferramentas que **fornece funcionalidades e abstrações que facilitam a criação, treinamento e implantação** (i.e., inferência) de modelos de ML.
- Alguns exemplos populares de **frameworks** de ML incluem TensorFlow, PyTorch, Scikit-learn e MXNet, etc.
- Usaremos o **Tensorflow (TF)**, pois é focado no desenvolvimento de redes neurais (profundas).

Frameworks de ML



- Quando trabalhamos com aplicações envolvendo modelos de ML em **computadores de uso geral**, além da aplicação, SO e HW, nós usamos o **TensorFlow** para a **criação, treinamento e implantação dos modelos**.
- O **TensorFlow** é **projetado** para tarefas de **alto desempenho e uso intensivo dos recursos** disponíveis (como CPUs e/ou TPUs/GPUs) e usa operações em **ponto flutuante**.
- Ou seja, não há preocupação com as eficiências energética e computacional.

Frameworks de ML



- Porém, o **TensorFlow** é muito **grande**, **não foi pensado para execução em dispositivos embarcados** e realiza **mais tarefas do que precisamos**.
- Quando queremos **executar modelos de ML em dispositivos com recursos limitados**, precisamos de um **framework** mais **enxuto e eficiente**.
- O **objetivo** com dispositivos embarcados é apenas **fazer previsões** (i.e., inferências) com modelos treinados de forma simples e eficiente.

TensorFlow (TF) Lite



Less memory

Less compute power

Only focused on *inference*



- Usado em smartphones, tablets e sistemas embarcados.
- ***Subconjunto do TF***, portanto, é mais restrito.
- Projetado para ser ***leve e eficiente em termos de uso de CPU e memória***.
- Suporta técnicas de otimização, como ***quantização, pruning e clustering*** para reduzir o tamanho dos modelos e melhorar a eficiência de execução em HW com recursos limitados.
- Realiza ***apenas inferência****.
 - O modelo deve ser treinados em uma máquina com mais recursos.
- ***Compatível com modelos treinados no TF***, permitindo sua conversão.

*apenas executa o modelo treinado.

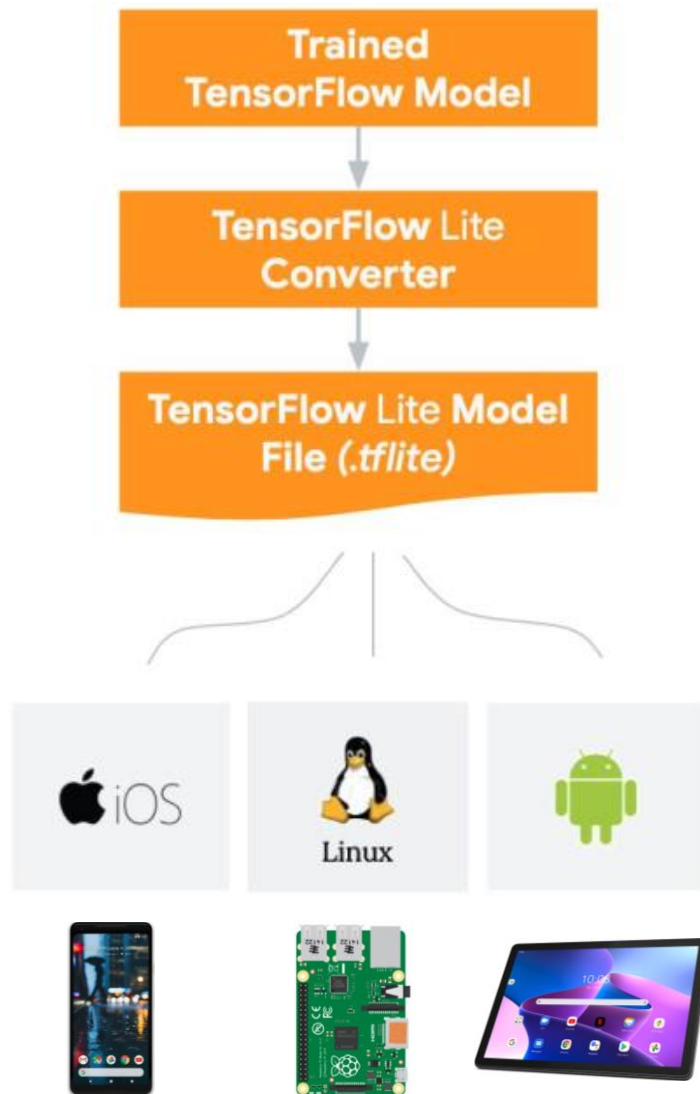
Sequência de trabalho com o TF Lite



TensorFlow Lite

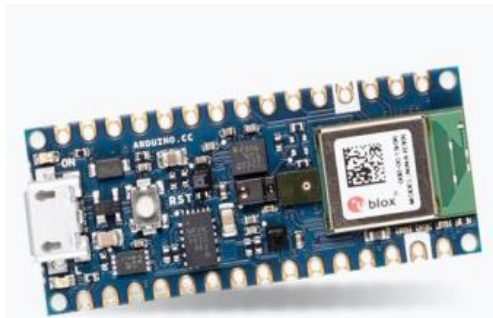
- É possível gerar um modelo do TF Lite de 3 formas:
 - Usando um modelo pré-treinado do TF Lite.
 - ✓ <https://www.tensorflow.org/lite/examples?hl=pt-br>
 - Criando um modelo do TensorFlow Lite.
 - ✓ Para isso, podemos usar o [TensorFlow Lite Model Maker](#)
 - Convertendo um modelo do TF em um do TF Lite.

Sequência de trabalho com o TF Lite



- O primeiro passo é **treinar um modelo criado com o TF** em um computador com muitos recursos computacionais.
- Após, usamos o **TF Lite converter para converter** esse modelo grande em um modelo pequeno.
 - Durante a conversão, podemos aplicar otimizações ao modelo.
- Esse processo gera um **arquivo com extensão .tflite**, o qual contém o modelo reduzido.
- Na sequência, **usamos este arquivo para implantar o modelo** em smartphones, tablets ou sistemas embarcados.
 - Basta carregar o arquivo com APIs específicas do TF Lite e realizar inferências.

TensorFlow (TF) Lite Micro



Even less memory

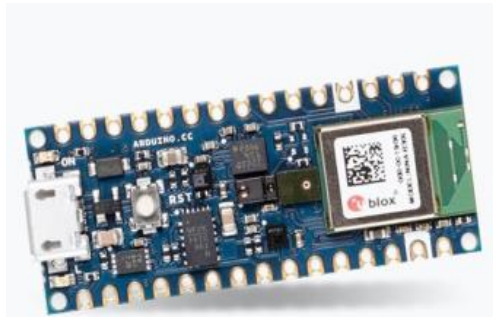
Even less compute power

Also, only focused on *inference*

?

- O TF Lite é a solução ideal para dispositivos edgeML.
 - EdgeML: dispositivos com capacidade computacional maior.
- Mas e quando falamos do uso de ML em *dispositivos com recursos extremamente limitados* e que requerem *baixíssimo consumo de energia*, como microcontroladores?

TensorFlow (TF) Lite Micro



Even less memory

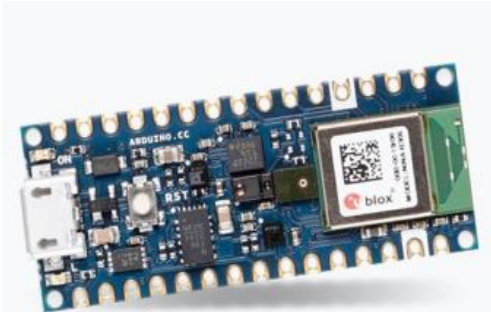
Even less compute power

Also, only focused on *inference*

?

- Como habilitamos a inferência nesses dispositivos?
- Eles têm ***capacidades computacionais muito menores*** em comparação com smartphones e tablets.
- Portanto, ***eles requerem uma implementação ainda mais enxuta e eficiente.***

TensorFlow (TF) Lite Micro



Even less memory

Even less compute power

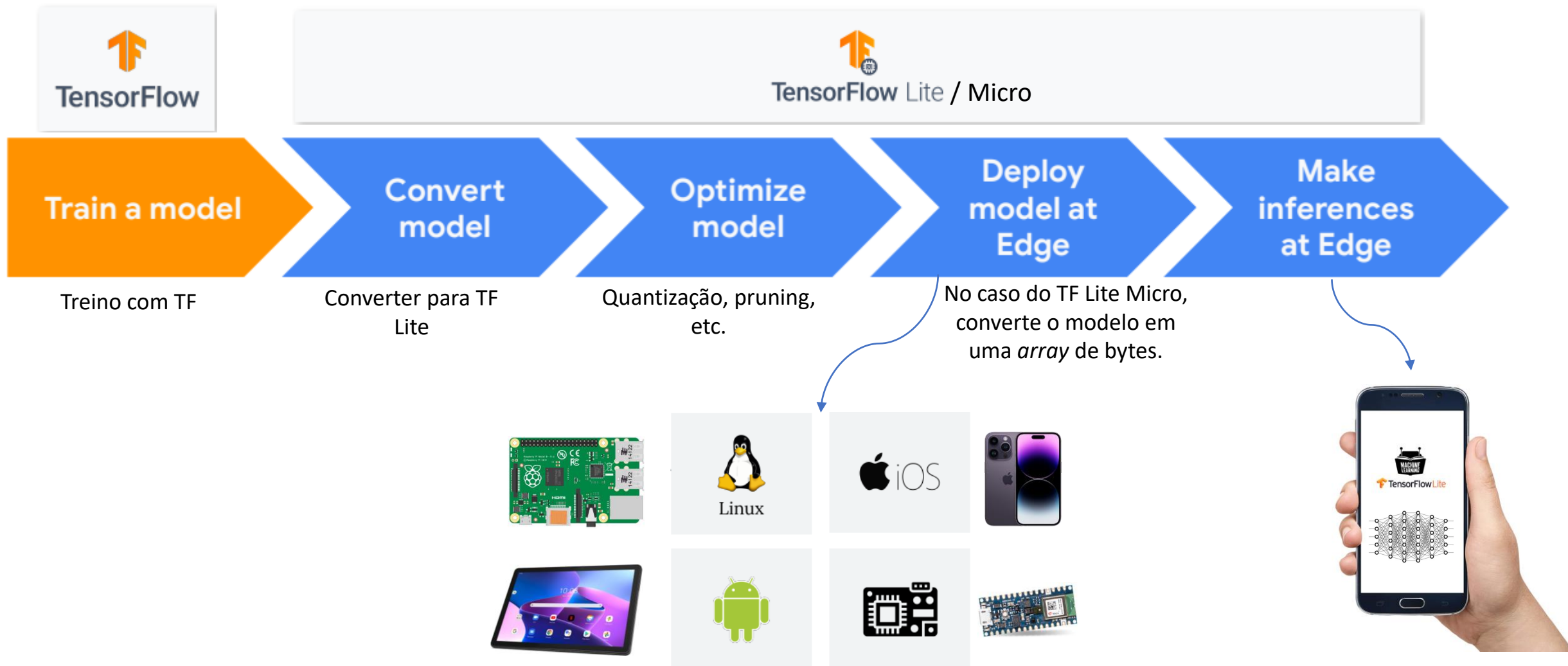
Also, only focused on *inference*



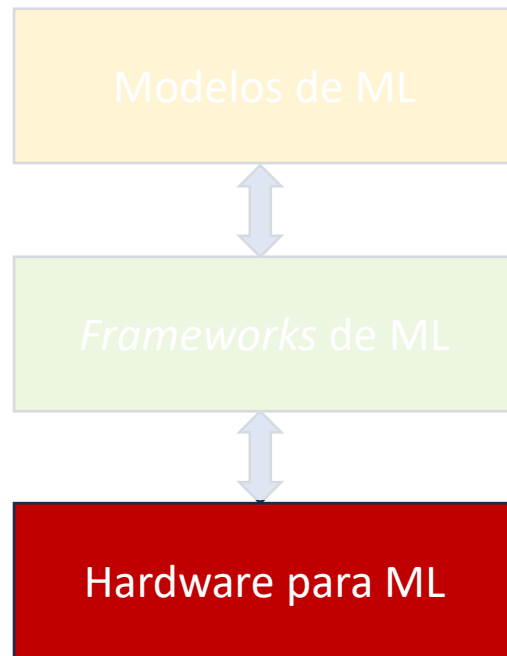
TensorFlow Lite
Micro

- O TF Lite Micro é uma versão ainda ***mais leve e otimizada do TF Lite***.
- Projetado para dispositivos embarcados de recursos extremamente limitados (CPU e memória), como microcontroladores.
- Escrito em C++ 17 e roda apenas em plataformas de 32 bits.
- Precisa apenas de ***16 KB de memória***.
- ***Não requer*** suporte de ***sistema operacional, bibliotecas C ou C++ padrão ou alocação dinâmica de memória***.
- Após a conversão do modelo TF em TF Lite, o converte para uma ***array de bytes*** e o ***compila junto ao programa***.

Pipeline de desenvolvimento

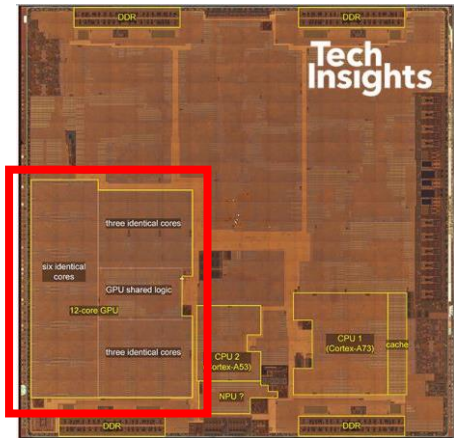


Hardware especializado

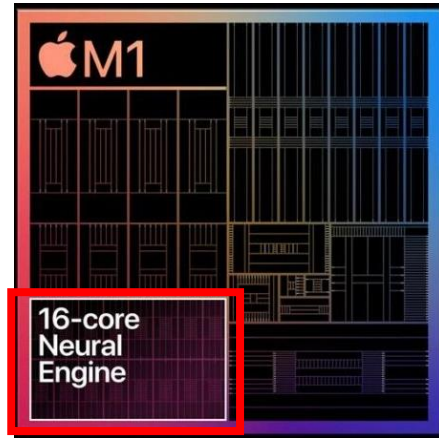


- Outra forma para habilitar o uso de ML em dispositivos embarcados é através de **hardwares otimizados** para a execução eficiente dos modelos.
- Algumas empresas têm lançados *Systems-on-a-Chip* (SoCs) e microcontroladores equipados com **aceleradores de ML**, também chamados de *neural processing units* (NPUs) ou *neural engines* (NEs).
- São **hardwares altamente especializados** e **projetados para acelerar aplicações de ML**.

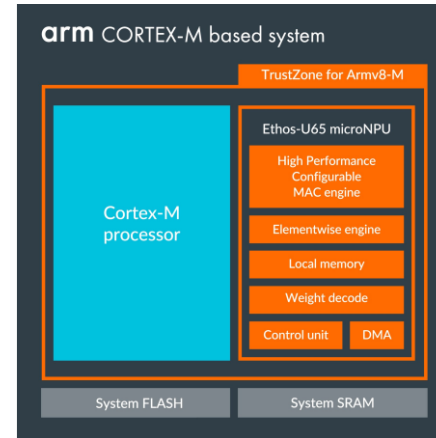
Hardware especializado



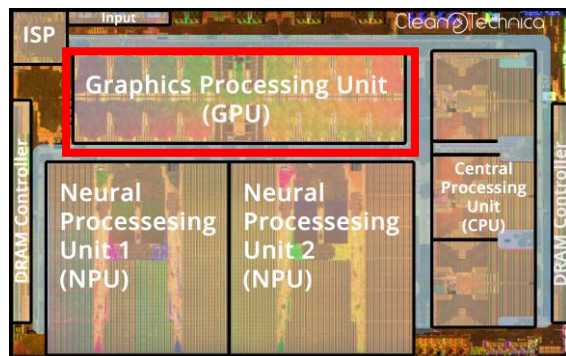
Huawei's Kirin 970



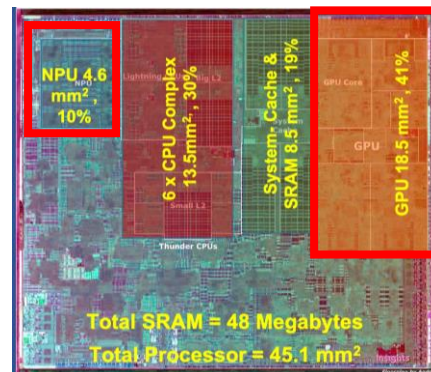
Apple's M1



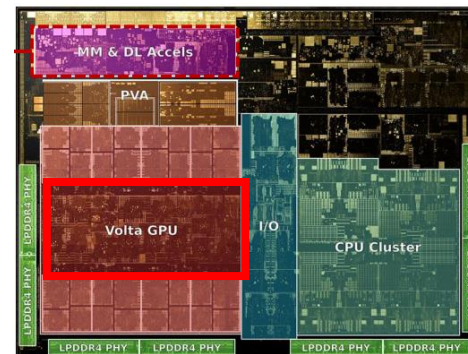
ARM's Cortex-M55, Ethos-U55, U65 e N78



Tesla's FSD



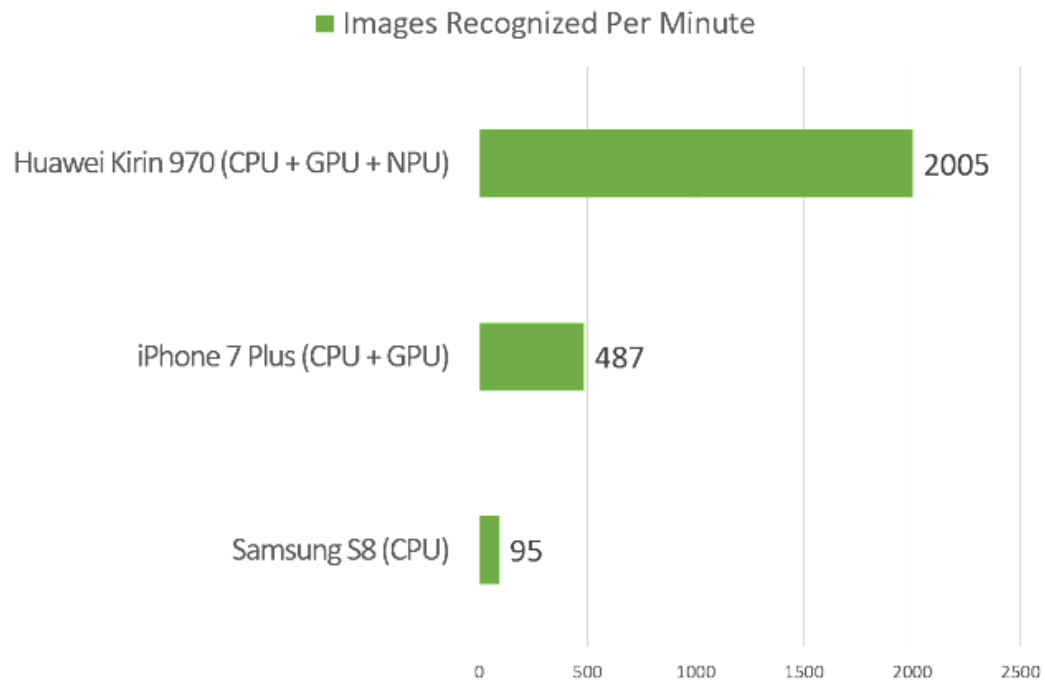
Apple's A13



NVIDIA's Xavier SoC

- Em geral, por serem muito especializadas, as NPUs oferecem uma execução muito mais rápida do que os CPUs e GPUs para tarefas de ML específicas.
- NPUs realizam *operações paralelas de multiplicação-acumulação* (MAC) e suportam *compactação e descompactação de pesos*, ajudando a *minimizar o uso da memória* do sistema.

Hardware especializado



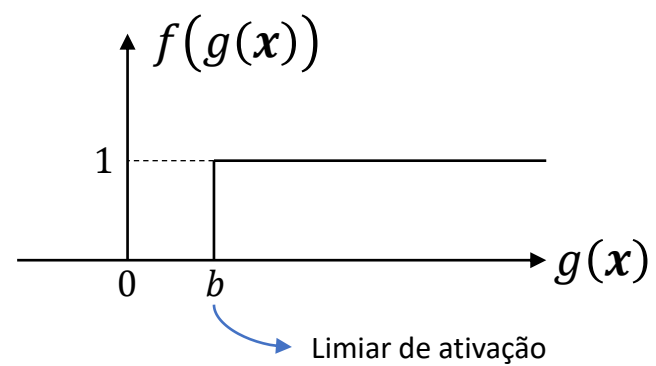
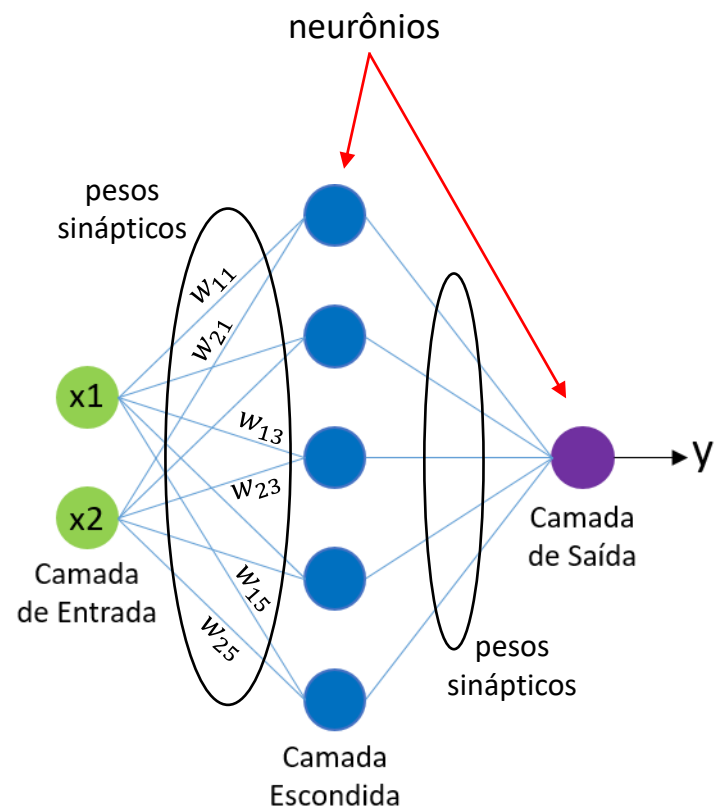
- Esses HWs especializados habilitam aplicações mais complexas como detecção de objetos e reconhecimento de fala em tempo real.
- A figura ao lado mostra que a NPU no Kirin 970 da Huawei faz com que ele supere de longe o iPhone 7 plus na tarefa de reconhecimento de imagens.

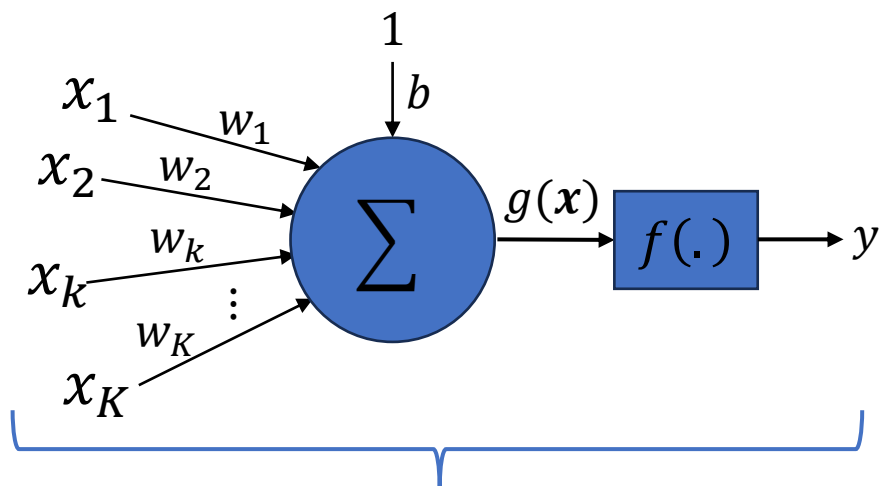
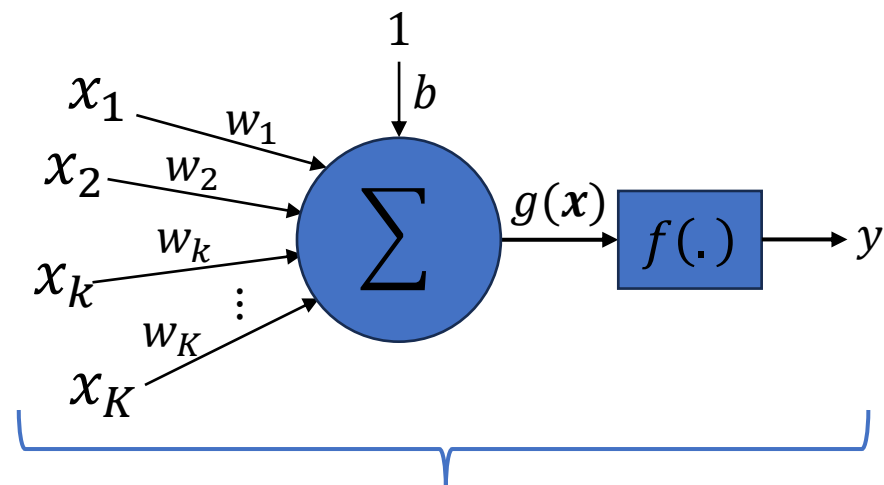
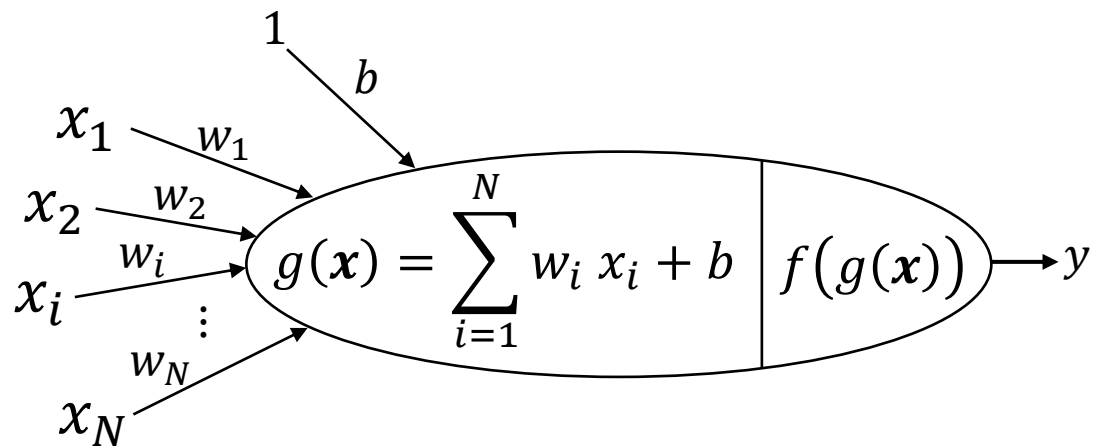
Atividades

- Quiz: “***TP557 - Desafios do TinyML - Machine Learning***”
- Assistir aos vídeos do Prof. Marcelo Rovai sobre Google Colab e Python.
 - **Google Colab intro:**
https://www.youtube.com/watch?v=m_Ueb88yd88&ab_channel=MarceloRovai
 - **Python review:**
https://www.youtube.com/watch?v=07tGfteD4_s&ab_channel=MarceloRovai
 - **Notebooks e documentos usados nos vídeos:** https://github.com/Mjrovai/UNIFEI-IESTI01-TinyML-2022.1/tree/main/00_Curse_Folder/1_Fundamentals/Class_04a

Perguntas?

Obrigado!

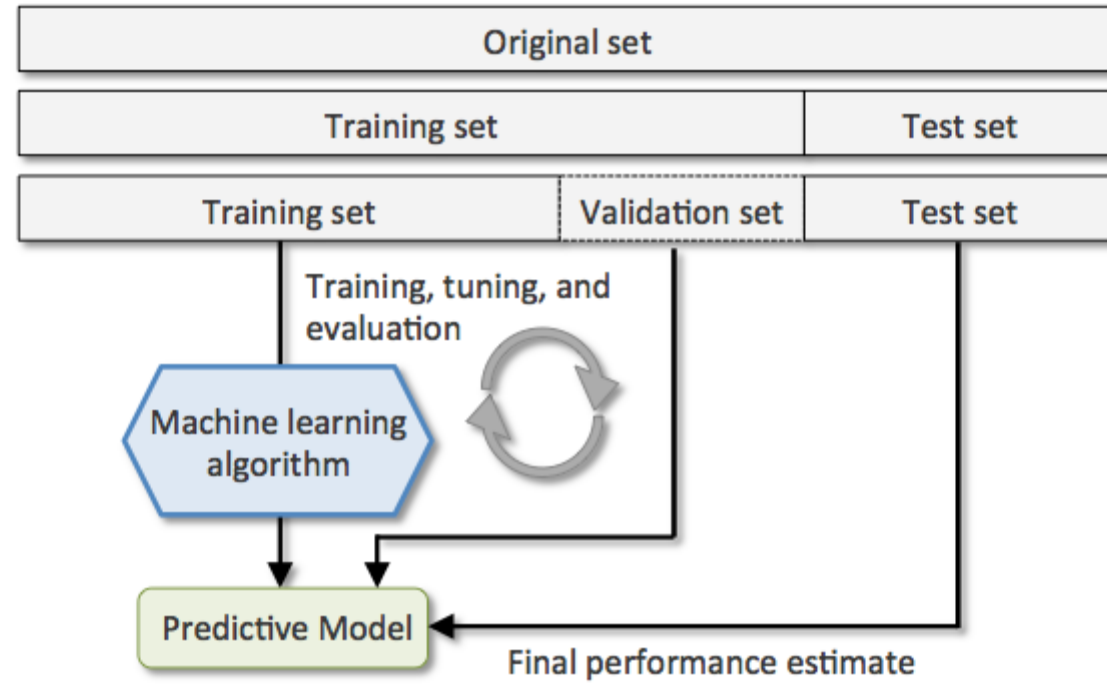


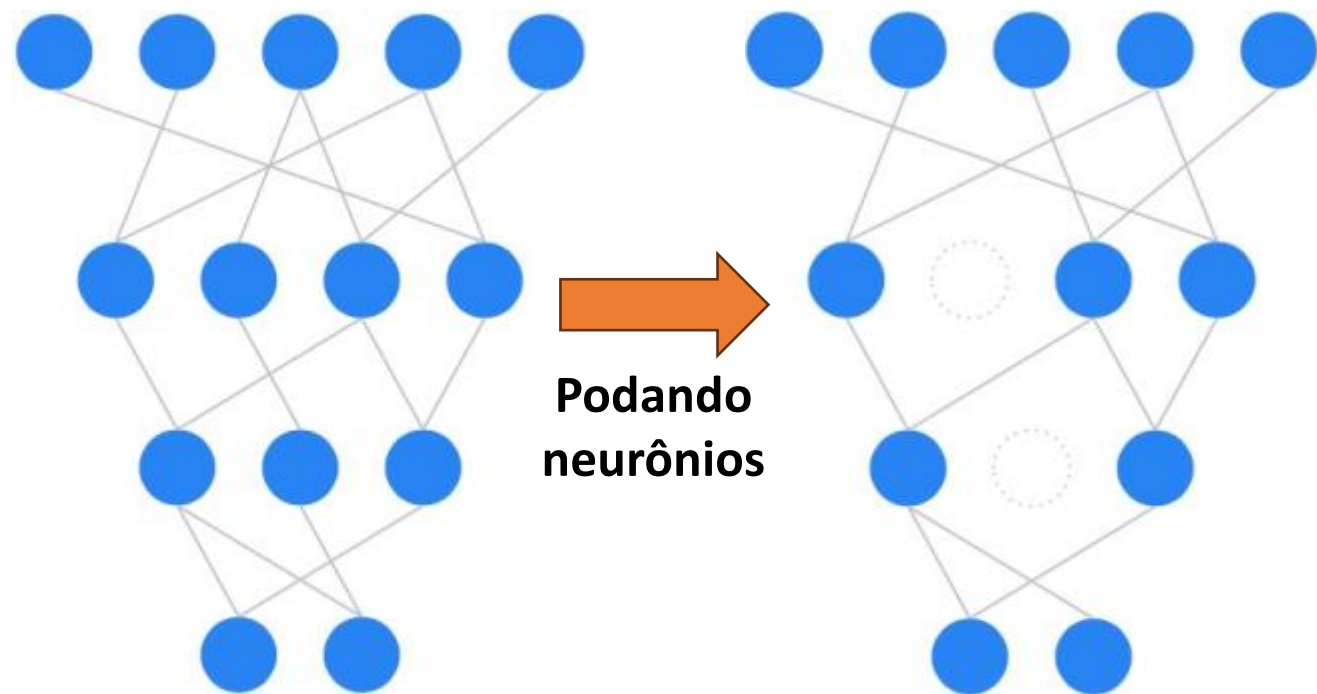
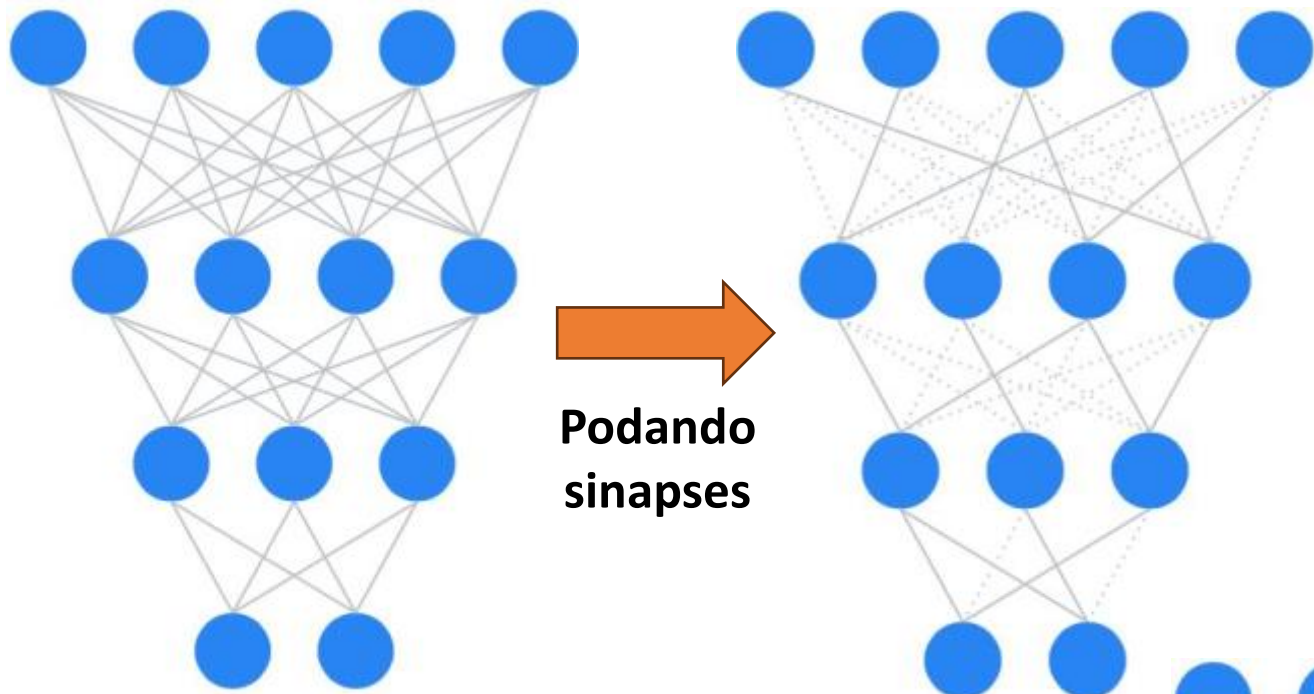


$$y = f(g(x)) = f\left(\left(\sum_{i=1}^K w_i x_i\right) + b\right)$$

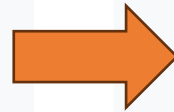
$$y = f\left(\left(\sum_{i=1}^K w_i x_i\right) + b\right)$$

Pesos ou
parâmetros

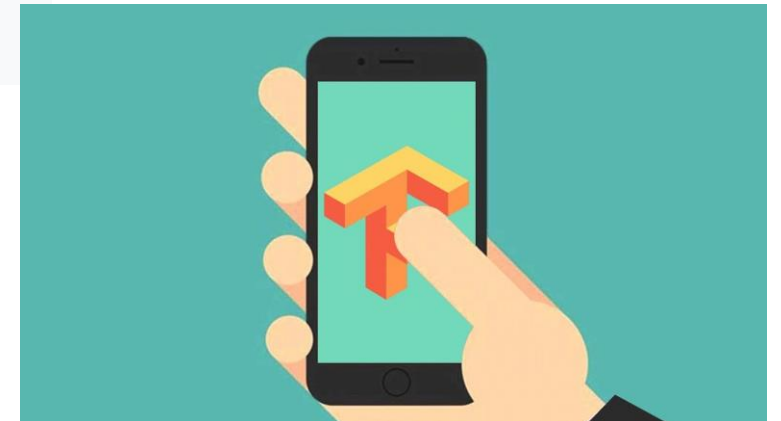


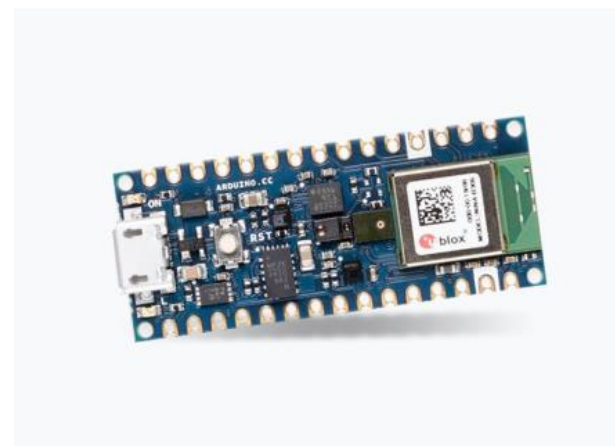
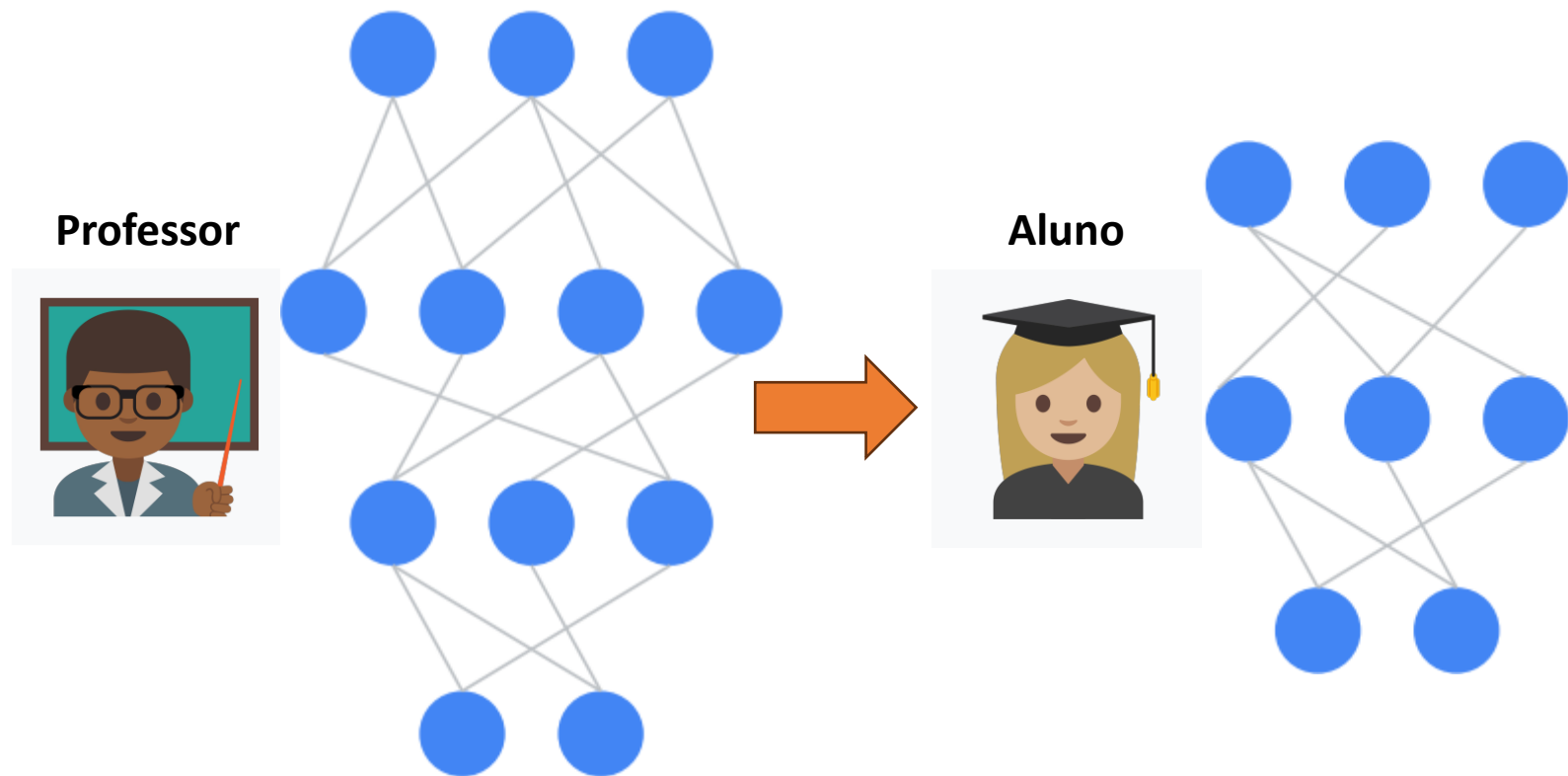


Professor

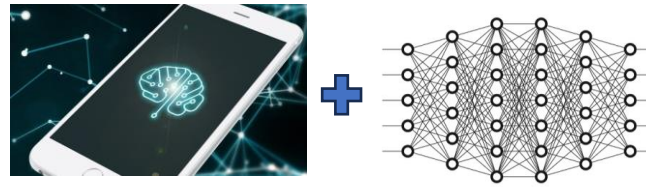


Aluno





Aplicação



Framework
de ML


TensorFlow

Sistema Operacional

Hardware
de uso geral

