Redes Neurais Convolucionais

1. Introdução a aprendizado profundo (*deep learning*)

É provável que o leitor já tenha ouvido algo acerca de redes neurais profundas (DNNs, do inglês *deep neural networks*) ou aprendizado profundo (DL, do inglês *deep learning*). A razão para isso é o enorme sucesso que essa abordagem vem tendo em domínios tão diversos quanto reconhecimento de imagens/áudio e tradução automática de textos (LeCun et al., 2015).

Se fosse necessário fornecer uma explicação para o crescimento acelerado do interesse por DL, ela teria de passar por dois fatores cruciais do mundo contemporâneo:

- O constante aumento da quantidade e disponibilidade de dados de todos os tipos.

- O significativo desenvolvimento de *hardware* capaz de permitir o treinamento de estruturas massivas (explorando, por exemplo, paralelismo).

Um aspecto marcante em *deep learning* é que os modelos tipicamente lidam com os dados brutos do problema, *i.e.*, sem que um estágio de pré-processamento e extração de atributos (*feature extraction*) entre em cena. Em suma, a ideia é "*deixar os dados falarem por si mesmos*".

Sendo assim, a partir da informação disponível na entrada, o modelo deve **aprender diferentes níveis de representação** que permitam a solução do problema. Isso contrasta com diversas abordagens clássicas em que se busca definir as representações dos dados por meio de engenharia de atributos *a priori*, conforme ilustra a Figura 1.

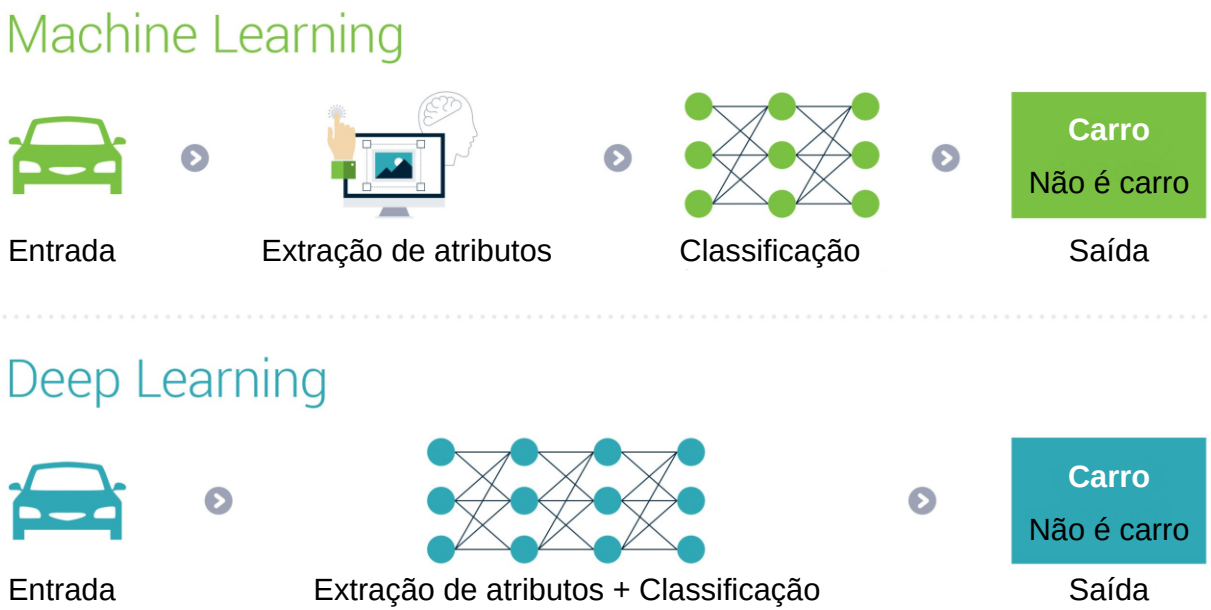

Figura 1 – Abordagem clássica com engenharia de atributos e *deep learning*. Adaptada de https://www.inteliment.com/blog/our-thinking/lets-understand-the-difference-between-machine-learning-vs-deep-learning/.

Na realidade, muitos modelos de *deep learning* constroem uma hierarquia de representações, a qual permite que a máquina aprenda aspectos mais complicados presentes nos dados construindo-os a partir de características mais elementares. Ao

mesmo tempo, o modelo também aprende como processar estas representações para realizar a tarefa desejada (*e.g.*, classificar uma imagem).

2. Redes convolucionais

As redes neurais convolucionais (CNNs ou ConvNets, do inglês *convolutional neural networks*) foram desenvolvidas tendo como inspiração a estrutura e o funcionamento do córtex visual de mamíferos* (HUBEL & WIESEL, 1959; FUKUSHIMA, 1980; LECUN ET AL., 1998), e atingiram em anos recentes níveis desempenho sobre-humano em algumas tarefas desafiadoras de visão computacional. Além disso, CNNs também são bem-sucedidas em outras tarefas, como reconhecimento de voz e processamento de linguagem natural. Por simplicidade, vamos nos concentrar nesta exposição ao contexto de processamento de imagens.

---

* Por exemplo, Hubel e Wiesel (1959) definiram o conceito de campo receptivo local, *i.e.*, que neurônios reagem apenas a estímulos visuais localizados em uma região limitada do campo visual.

Redes convolucionais empregam a operação de *convolução* no lugar da transformação afim inerente a uma camada do tipo *perceptron* (GOODFELLOW ET AL., 2016). Essa operação, que é linear, permite explorar informações em estruturas organizadas no tempo, como séries temporais, ou no espaço, como imagens. Outra operação tipicamente presente em CNNs é chamada de *agrupamento* (ou *pooling*), a qual realiza uma sub-amostragem da entrada, resumindo a informação.

2.1. Recordando a operação de convolução

Dados dois sinais unidimensionais (1D) $x(n)$ e $w(n)$, a convolução entre ambos é definida como (GONZALEZ & WOODS, 2010):

$$y(n) = \sum_{k=-\infty}^{\infty} x(k)w(n-k) = (x * w)(n) \qquad (1)$$

É possível estender esta operação para o caso de duas dimensões, o que é interessante, por exemplo, quando se lida com imagens:

$$S(i,j) = (X * K)(i,j) = \sum_m \sum_n X(m,n)K(i-m, j-n) \quad (2)$$

Como a convolução é comutativa, temos que:

$$S(i,j) = (K * X)(i,j) = \sum_m \sum_n K(m,n)X(i-m, j-n) \quad (3)$$

Uma operação bastante parecida é conhecida como correlação-cruzada. Nela, não há inversão de sinal entre $X(\cdot)$ e $K(\cdot)$, o que é pertinente, embora a comutatividade no sentido estrito se perca (GONZALEZ & WOODS, 2010):

$$S(i,j) = (X \circledast K)(i,j) = \sum_m \sum_n X(m,n)K(i+m, j+n) \quad (4)$$

De qualquer modo, vamos empregar o termo *convolução* daqui em diante, embora a operação explorada nas CNNs seja, a rigor, uma correlação cruzada.

2.2. Camada convolucional

Uma camada convolucional é composta por um ou mais *kernels* (ou máscaras), os quais são responsáveis por processar o dado recebido através da operação de convolução.

Um *kernel* de convolução corresponde a um filtro espacial retangular. Um exemplo de *kernel* 3×3 é mostrado a seguir:

$k_{-1,-1}$	$k_{-1,0}$	$k_{-1,1}$
$k_{0,-1}$	$k_{0,0}$	$k_{0,1}$
$k_{1,-1}$	$k_{1,0}$	$k_{1,1}$

Como evidenciado na expressão em (4), o *kernel* percorre toda a imagem de entrada, sendo que, em cada posição, a resposta do neurônio é obtida através da soma dos *pixels* da imagem dentro da vizinhança definida pelo *kernel*, multiplicados pelos coeficientes do *kernel*. Vejamos algumas ilustrações deste processo.

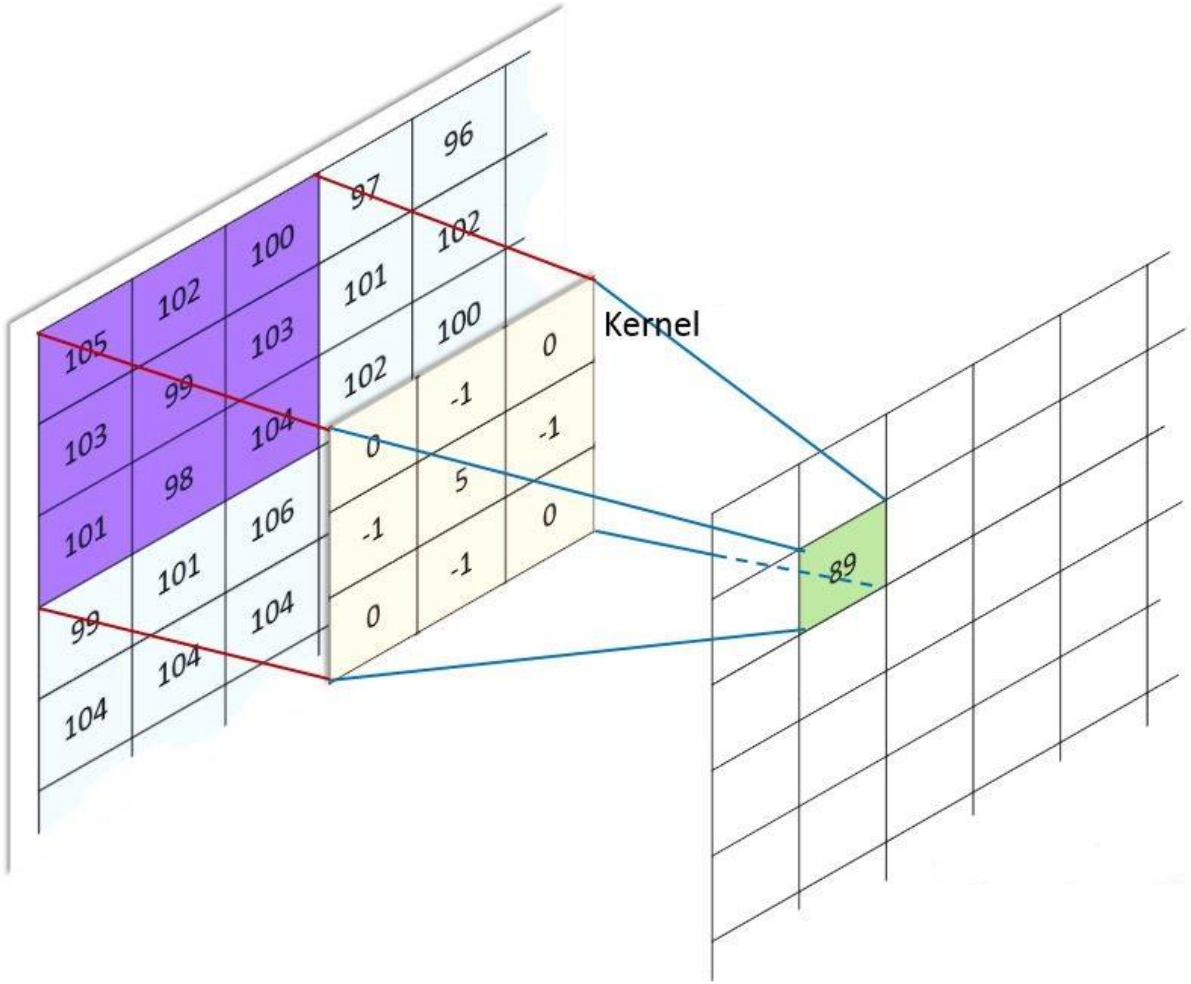

Figura 2 – Cálculo do resultado da convolução (um único neurônio). Adaptada de https://embarc.org/embarc_mli/doc/build/html/MLI_kernels/convolution_2d.html.

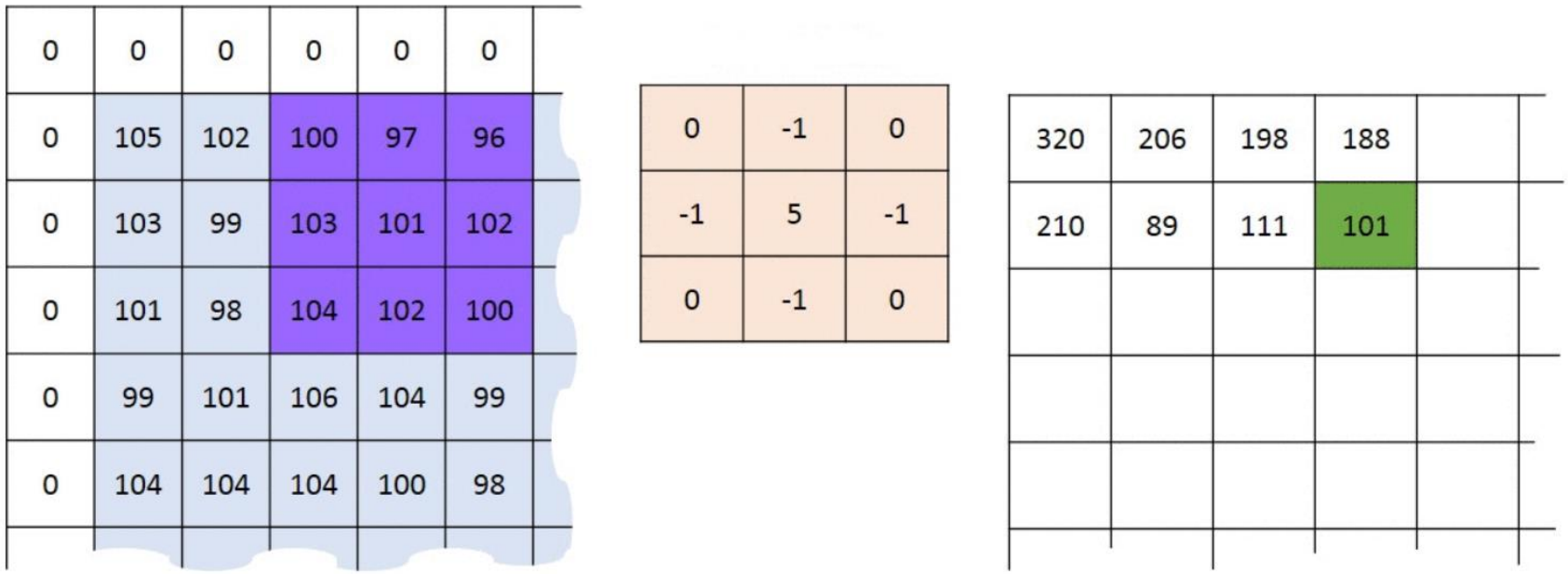

Figura 3 – Respostas de vários neurônios com base na convolução da imagem com o *kernel*. Adaptada de https://stats.stackexchange.com/questions/296679/what-does-kernel-size-mean.

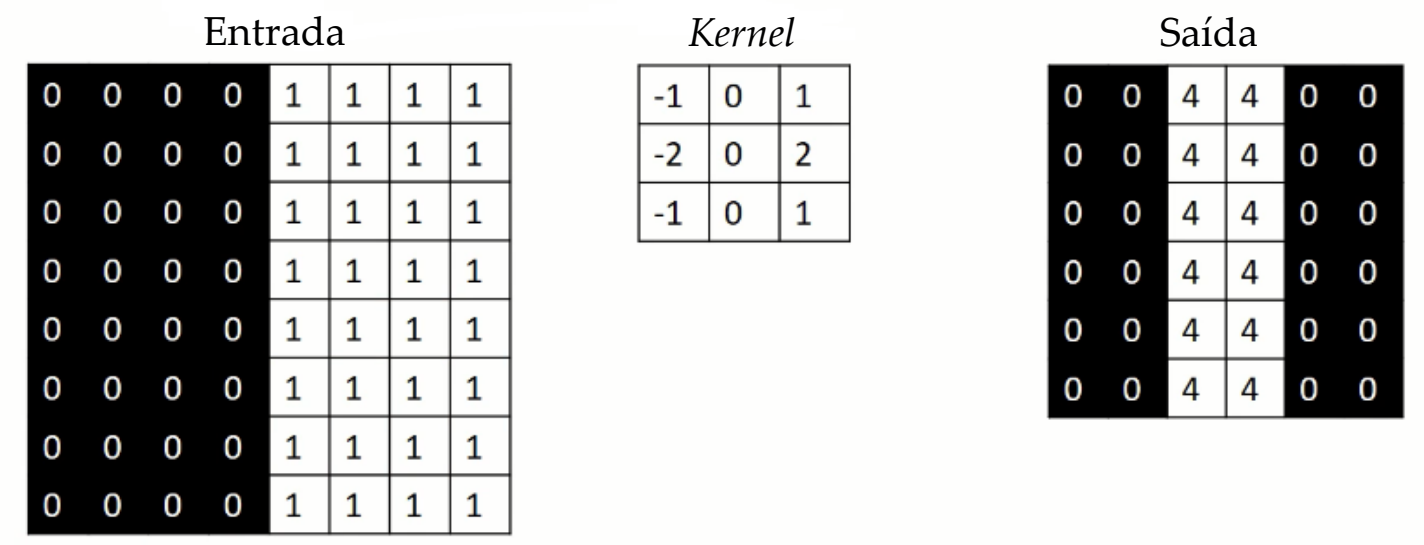

Figura 4 – Resultado completo de uma convolução.

Conceitos importantes em CNNs:

- Cada filtro (*kernel*) gera uma nova imagem como resposta à entrada, a qual é chamada de *feature map*, *activation map* ou, ainda, de canal. Uma camada convolucional contendo N *kernels* gera, portanto, um volume de saída com N canais (*feature maps*).
- Cada *"pixel"* em um *feature map* pode ser considerado como um neurônio, e é influenciado por um subconjunto dos *pixels* da imagem de entrada, os quais

definem o chamado *campo receptivo* daquele neurônio. A Figura 5 apresenta uma estrutura com duas camadas convolucionais colocadas em sequência, destacando os campos receptivos.

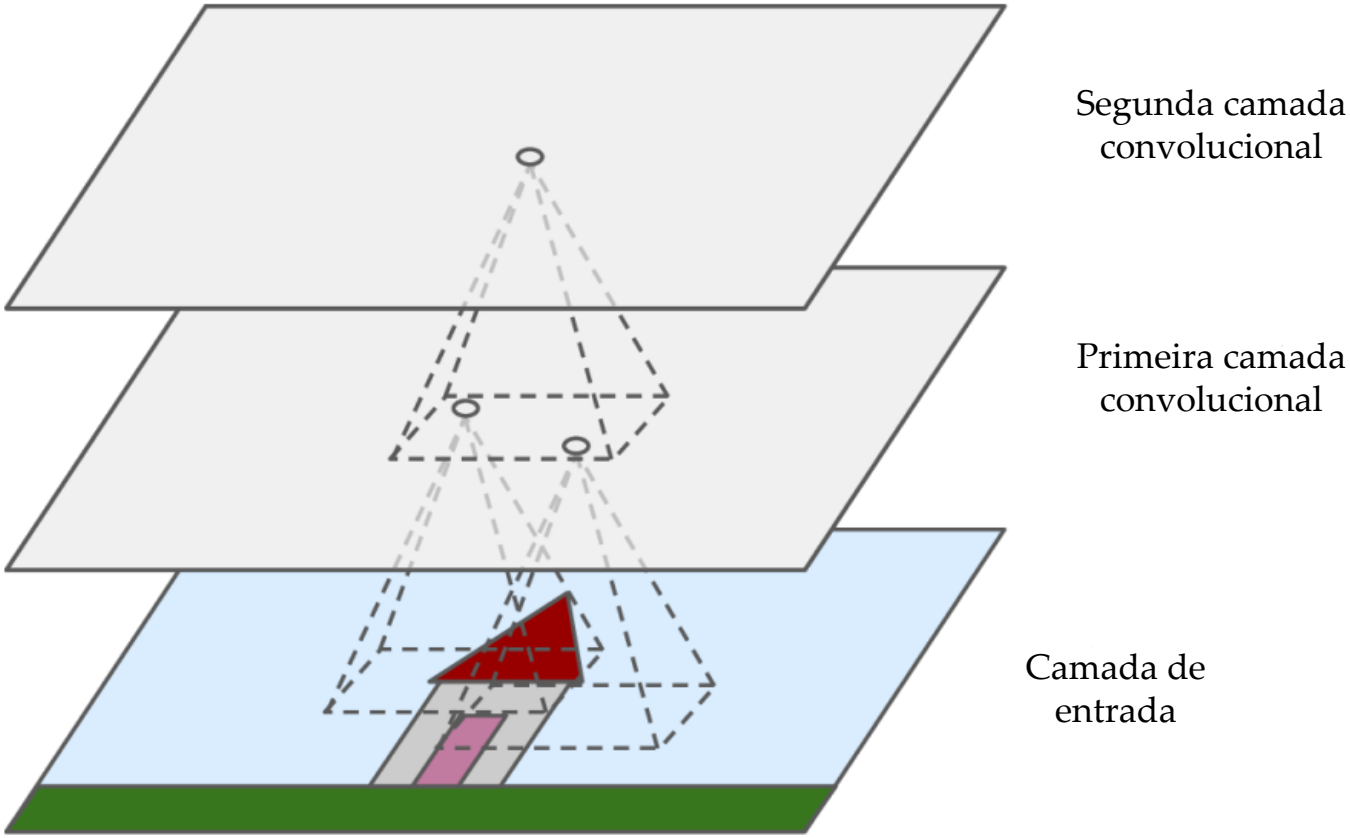

Figura 5 – Camadas convolucionais, com destaque para os campos receptivos. Adaptada de (GÉRON, 2019).

- Quando camadas convolucionais são empilhadas, ocorre um progressivo aumento do campo receptivo em relação ao dado de entrada da rede. A Figura 6 evidencia este aspecto. Por sua vez, a Figura 7 ilustra a ação de uma camada convolucional com quatro *kernels*.

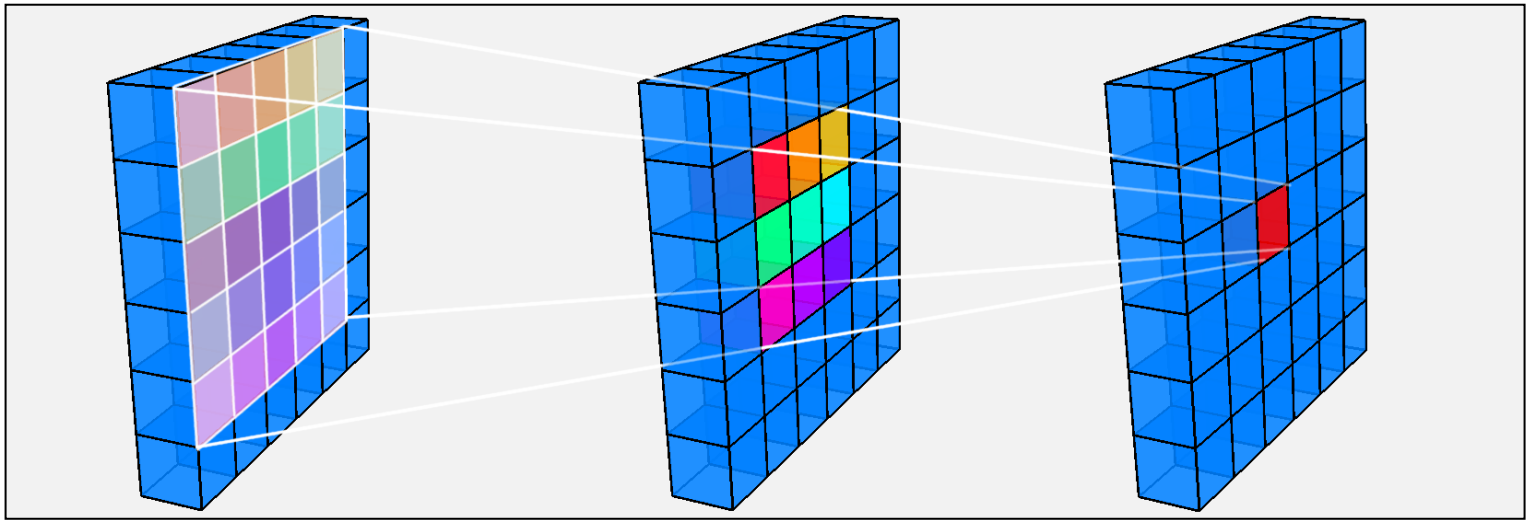

Figura 6 – Ampliação do campo receptivo à medida que a profundidade na rede aumenta. Extraída de (Marques, 2017).

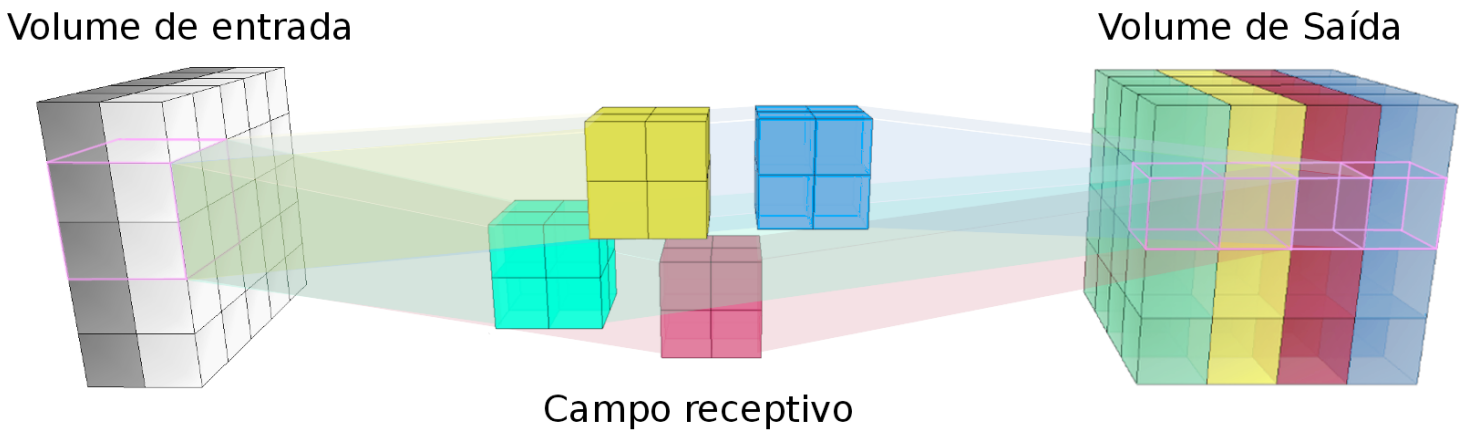

Figura 7 – Camada convolucional com quatro *kernels*. Em destaque, os campos receptivos associados a alguns neurônios dos canais de saída. Extraída de (Marques, 2017).

- Durante o cálculo da convolução, é possível considerar uma versão ampliada da imagem de entrada através da adição de *pixels* iguais a zero em suas bordas. Esta ideia, conhecida como *zero padding*, possibilita a manutenção do tamanho original da imagem, fazendo com que o elemento central do *kernel* de convolução possa ser posicionado uma vez sobre todos os *pixels* da imagem original. A Figura 8 traz

um exemplo de convolução com *zero padding*, novamente destacando os campos receptivos.

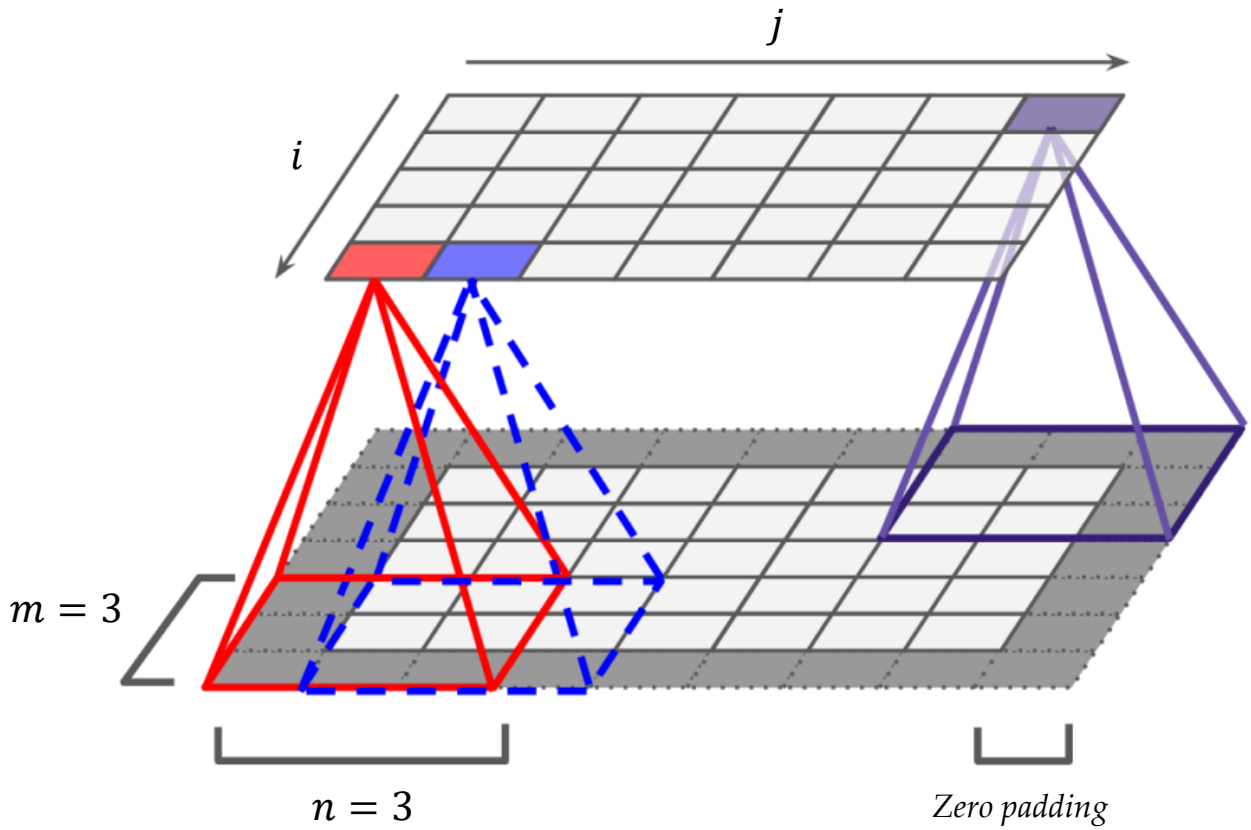

Figura 8 – Processo de convolução com *kernel* 3 × 3 e *zero padding*. Adaptada de (GÉRON, 2019).

- O tamanho do deslocamento do *kernel* durante a convolução, i.e., a quantidade de *pixels* com que o *kernel* é movido horizontal e verticalmente para gerar os elementos do *feature map* é conhecido como *stride*. Quanto maior o valor do *stride*, mais rapidamente é reduzida a dimensão da imagem. A Figura 9 apresenta ilustra uma convolução com *stride* diferente de um.

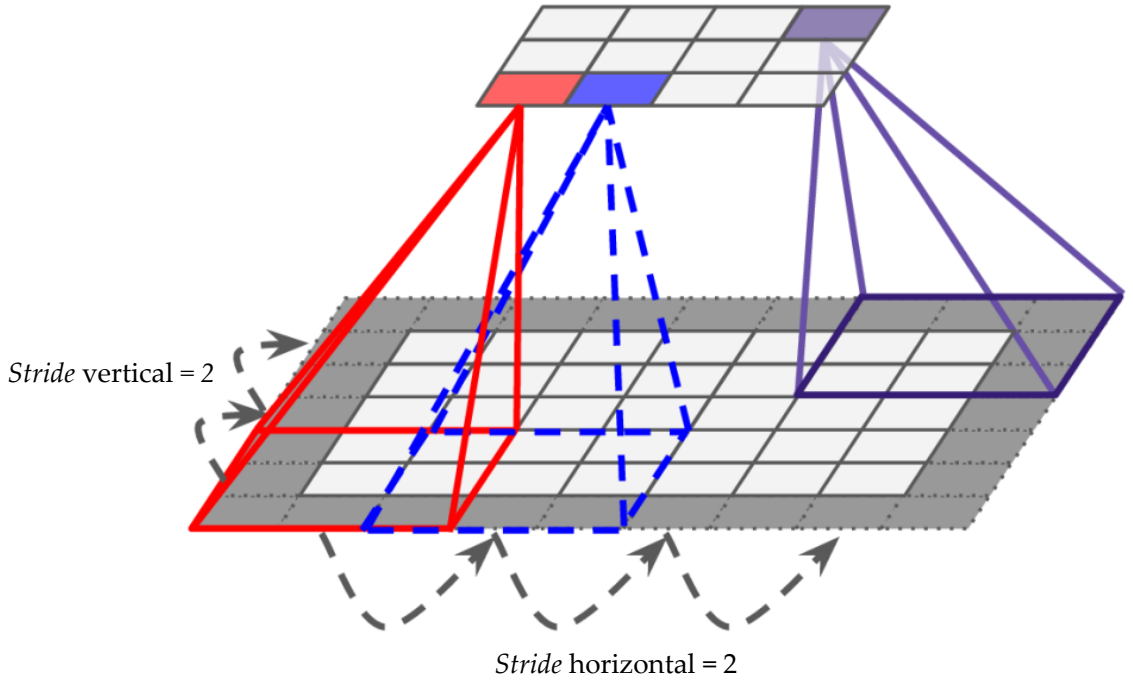

Figura 9 – Redução de dimensionalidade usando um *stride* igual a 2. Adaptada de (GÉRON, 2019).

- Todos os neurônios em um *feature map* compartilham os mesmos parâmetros (*weight sharing*), pois estão associados ao mesmo *kernel* de convolução. Isso leva a uma redução expressiva no número de parâmetros ajustáveis da camada, especialmente se comparado ao que seria exigido por uma camada densa (*fully connected*). A Figura 10 evidencia essa discrepância na quantidade de parâmetros correspondente a uma MLP e uma CNN.

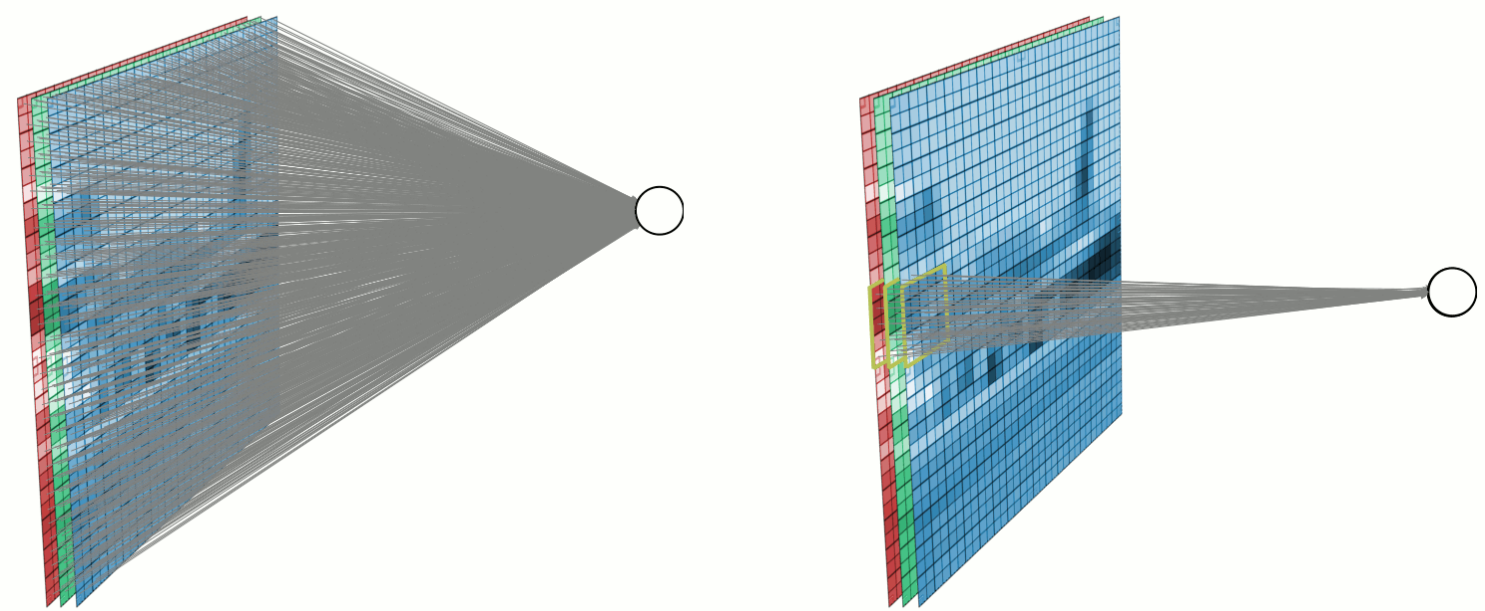

Figura 10 – Contraste entre uma camada densa e uma convolucional. Extraída de (Marques, 2017).

É interessante notar também que, em uma camada convolucional, temos interações esparsas de entrada e saída, especialmente quando o *kernel* tem dimensão menor que a da entrada. Isso significa que um *pixel* de entrada exerce influência sobre um grupo relativamente pequeno de neurônios do *feature map*.

- O mesmo conjunto de parâmetros aprendido é usado em diferentes partes da entrada, o que é excelente do ponto de vista de custo de armazenamento. Além disso, uma vez que a CNN tenha aprendido a reconhecer um determinado padrão em uma certa localização da imagem, ela também é capaz de reconhecê-lo em qualquer outra posição (equivariância à translação).
- Uma CNN treinada em um conjunto de imagens de tamanho $p \times q$ pode receber entradas com outras dimensões, pois o número de neurônios em cada *feature map* é automaticamente determinado com base nas dimensões da entrada e do *kernel*, bem como na extensão do *zero padding* e no valor do *stride*.

- Uma camada convolucional tipicamente prevê a aplicação de uma função de ativação $\varphi(\cdot)$ sobre o resultado da convolução.

Todos os conceitos apresentados podem ser diretamente estendidos para o caso de processamento de imagens coloridas. Neste cenário, a imagem de entrada é representada como um tensor contendo os valores de intensidade de cada *pixel* para os três canais (R, G, B) de cores. Analogamente, cada *kernel* de convolução passa a cobrir uma vizinhança cúbica, ponderando e combinando, assim, as informações dos canais para gerar o *feature map*.

As Figuras 11 e 12 mostram o processo de geração de *feature maps* considerando uma imagem colorida de entrada.

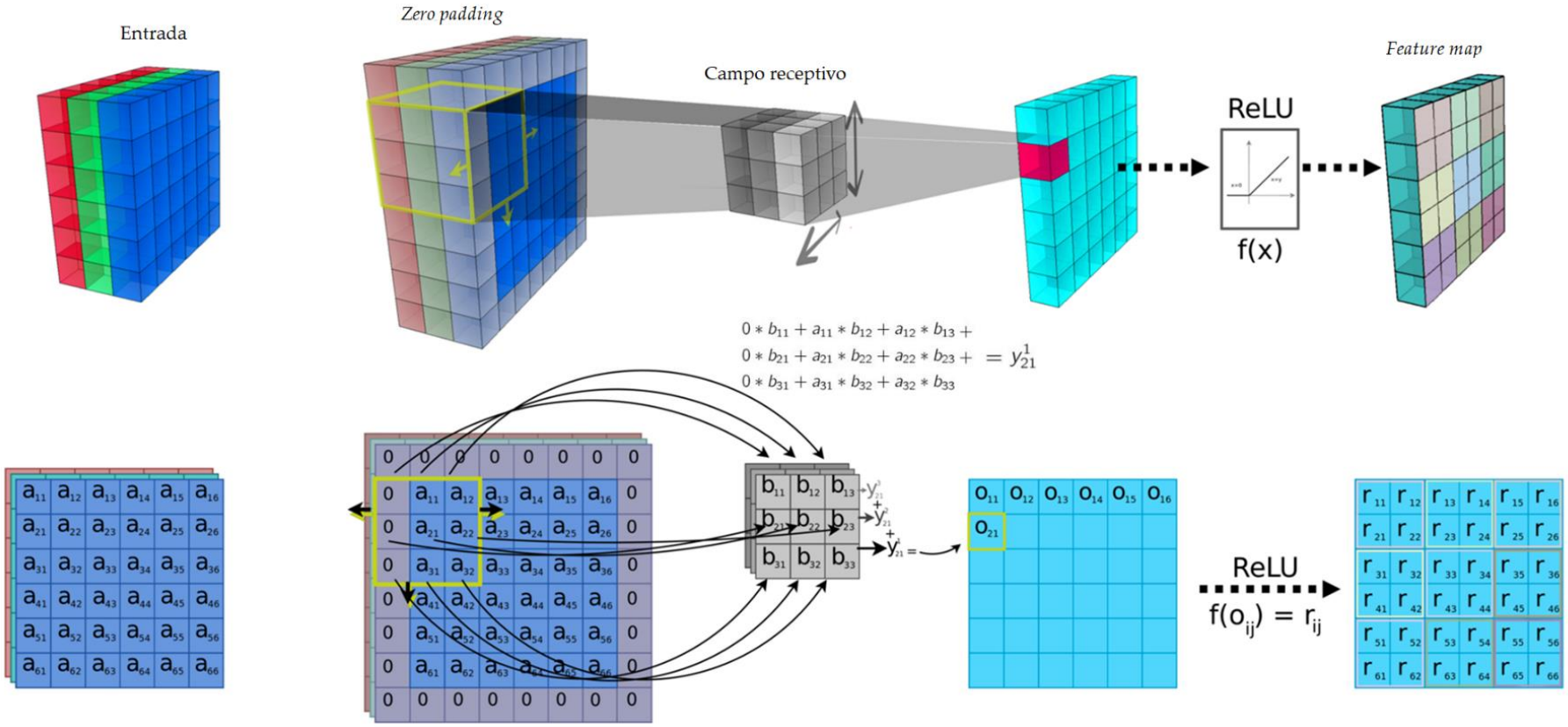

Figura 11 – Operação de um *kernel* 3D em uma camada convolucional. Adaptada de (Marques, 2017).

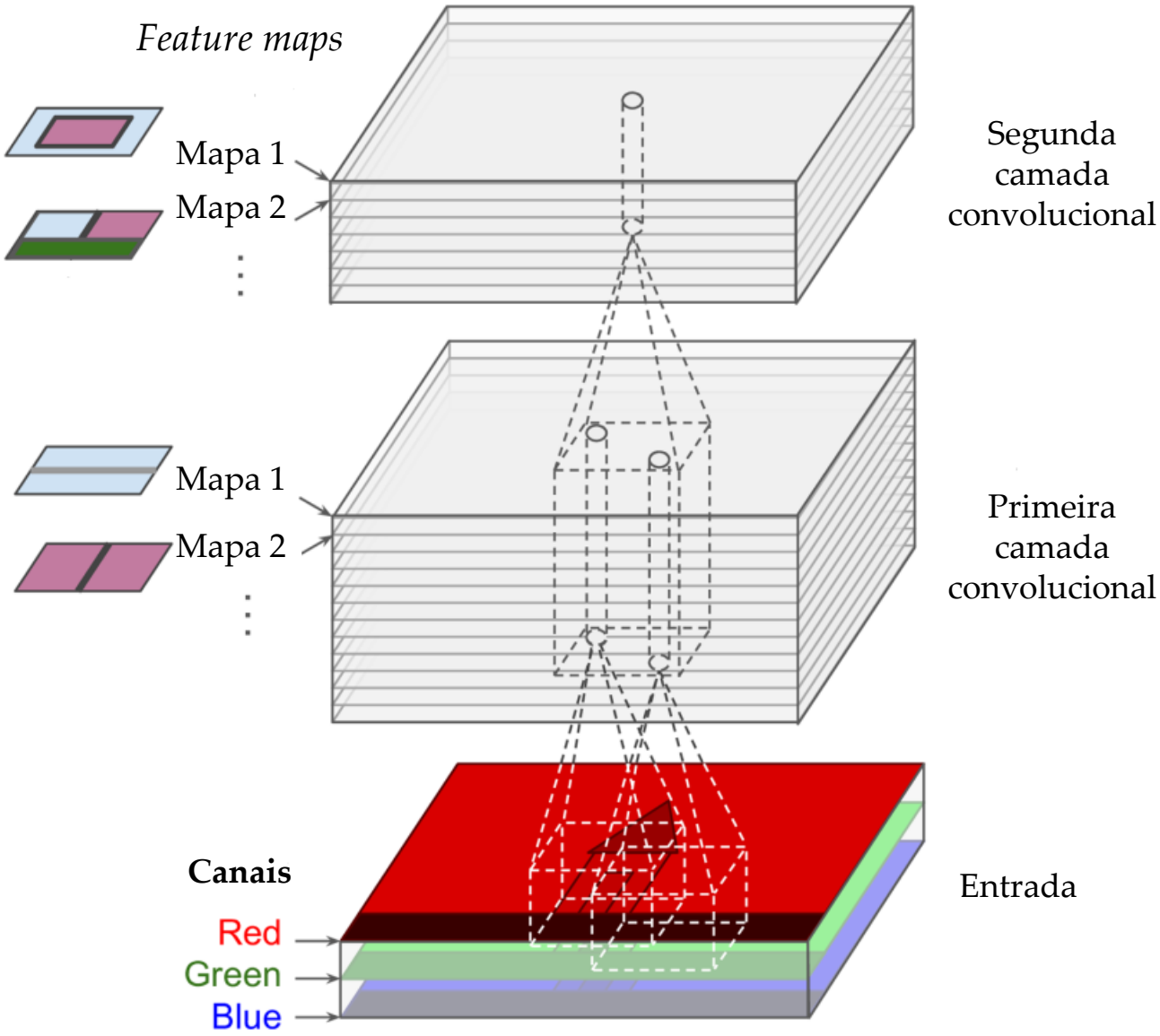

Figura 12 – Camadas convolucionais com múltiplos *feature maps* e imagens de três canais. Adaptada de (GÉRON, 2019).

2.3. Camada de *pooling*

O objetivo da camada de agrupamento (*pooling*) é sub-amostrar a imagem de entrada, fornecendo um "sumário" da informação em *pixels* vizinhos, a fim de reduzir a carga computacional, o uso de memória e o número de parâmetros da rede.

Assim como em uma camada convolucional, a camada de *pooling* requer a definição: (1) do tamanho do campo receptivo retangular, (2) do *stride*, e (3) do tipo de *padding*.

Por outro lado, em vez de conter pesos, a camada de *pooling* faz uso de uma função de agregação, a qual, justamente, será aplicada sobre os elementos da entrada contidos dentro do campo receptivo.

Dois tipos de *pooling* bastante usuais em CNNs são o *average pooling* e o *max-pooling*. No primeiro caso, o valor médio dos *pixels* contidos no campo receptivo é calculado,

enquanto o máximo valor é passado para a saída no segundo caso. As Figuras 13 e 14 exibem a ação de uma camada *max-pooling*.

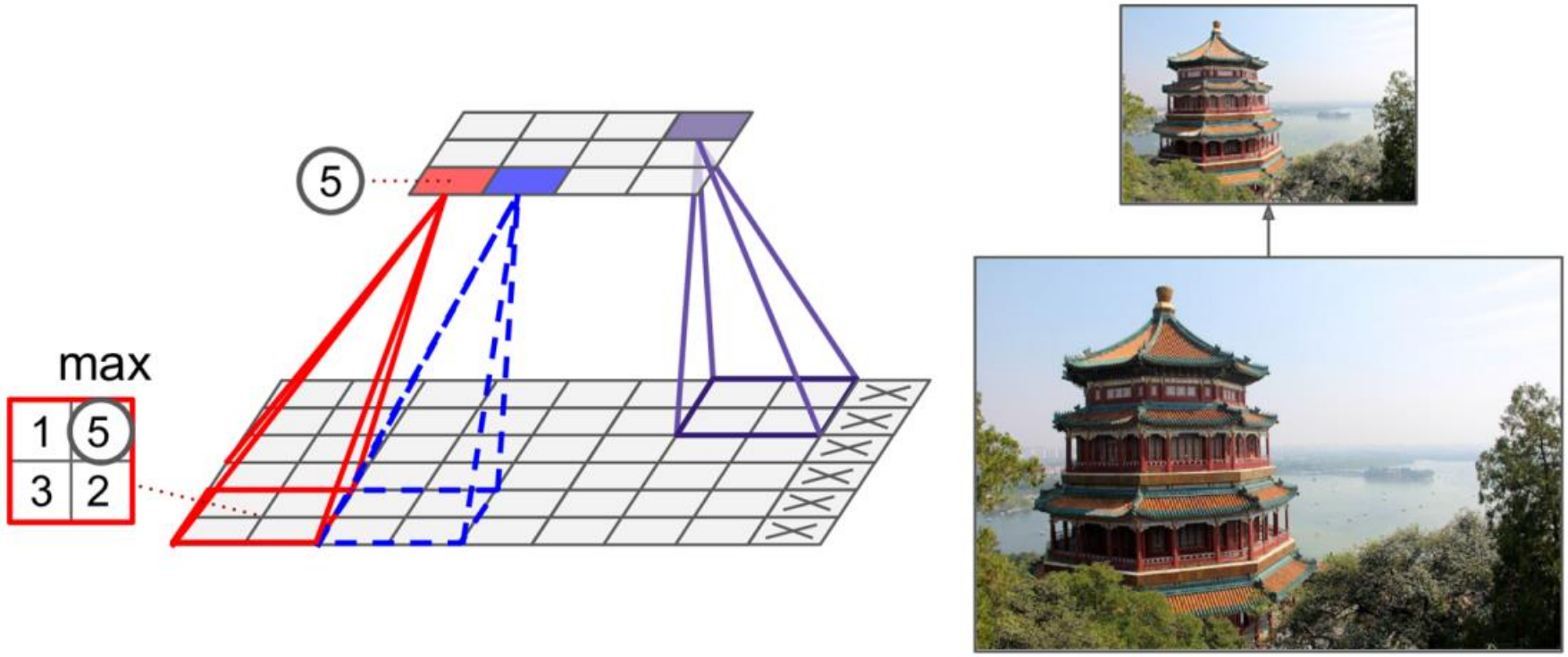

Figura 13 – Aplicação de *max-pooling* (*kernel* 2 × 2, *stride* = 2, sem *padding*). Adaptada de (GÉRON, 2019).

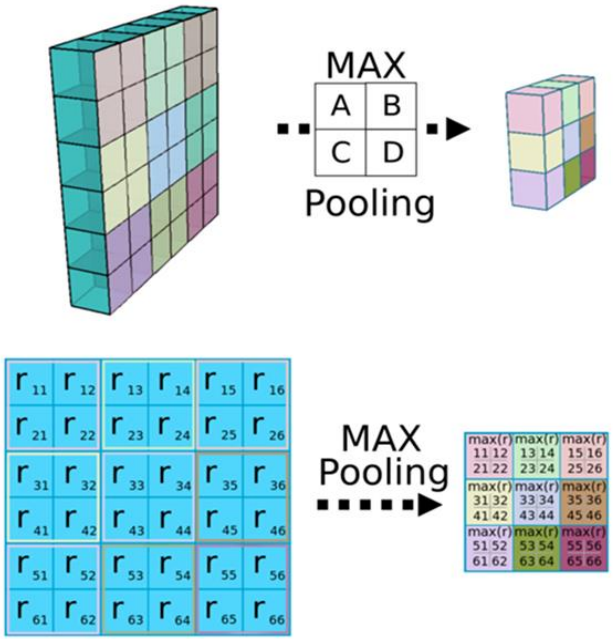

Figura 14 – Operação da camada de *max-pooling*. Adaptada de (MARQUES, 2017).

A operação de *pooling* também é interessante por tornar a representação aproximadamente invariante a pequenas translações da entrada. Ou seja, se a entrada for transladada ligeiramente, os valores das saídas (com *pooling*) ficarão majoritariamente iguais.

2.4. Arquiteturas de CNNs

As arquiteturas de CNNs tipicamente empilham algumas camadas convolucionais em sequência para, então, aplicar uma camada de *pooling*. Esta estrutura, com camadas convolucionais e *pooling*, normalmente é repetida algumas vezes. Com isso, a imagem de entrada se torna cada vez menor à medida que atravessa a rede, mas, ao mesmo tempo, ela também fica mais e mais profunda (*i.e.*, com mais *feature maps*), por conta das camadas convolucionais.

Então, no topo desta pilha, uma rede *feedforward* convencional é acrescentada, composta por algumas camadas totalmente conectadas, e, finalmente, a camada de saída gera a resposta da rede ao padrão de entrada. Por exemplo, podemos ter uma camada *softmax* para fornecer as probabilidades de a imagem pertencer a cada classe do problema. A Figura 15 esboça uma arquitetura genérica de CNN.

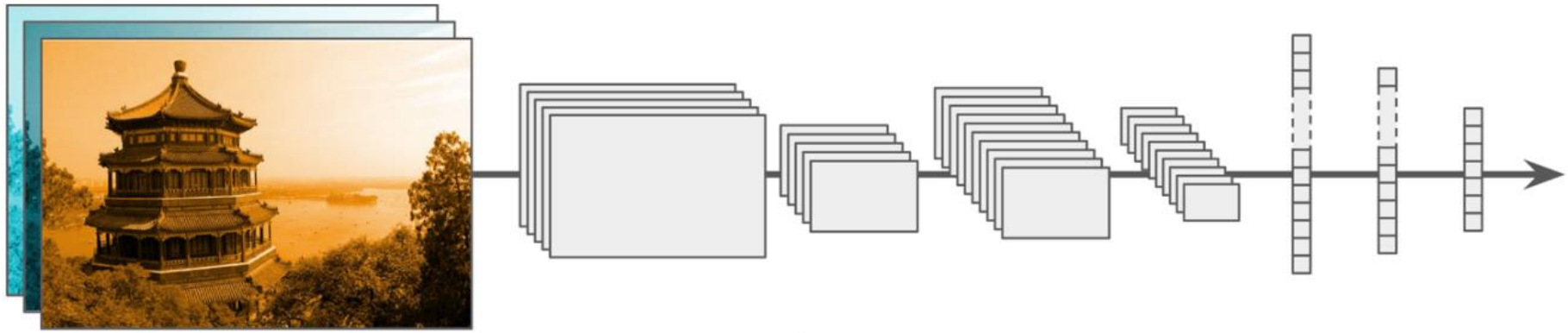

Figura 15 – Arquitetura típica de uma CNN. Adaptada de (GÉRON, 2019).

Agora, faremos um breve passeio por algumas arquiteturas famosas a fim de destacar as ideias criativas que levaram *deep learning* a atingir desempenhos impressionantes em visão computacional.

2.4.1. Fazendo História: AlexNet

O trabalho (KRIZHEVSKY ET AL., 2012) ocupa lugar de destaque na história das redes convolucionais profundas. Nele, apresenta-se uma arquitetura que conseguiu superar os melhores resultados da época, na competição ImageNet LSVRC-2010, por uma larga margem.

A arquitetura utilizada foi a mostrada na Figura 16.

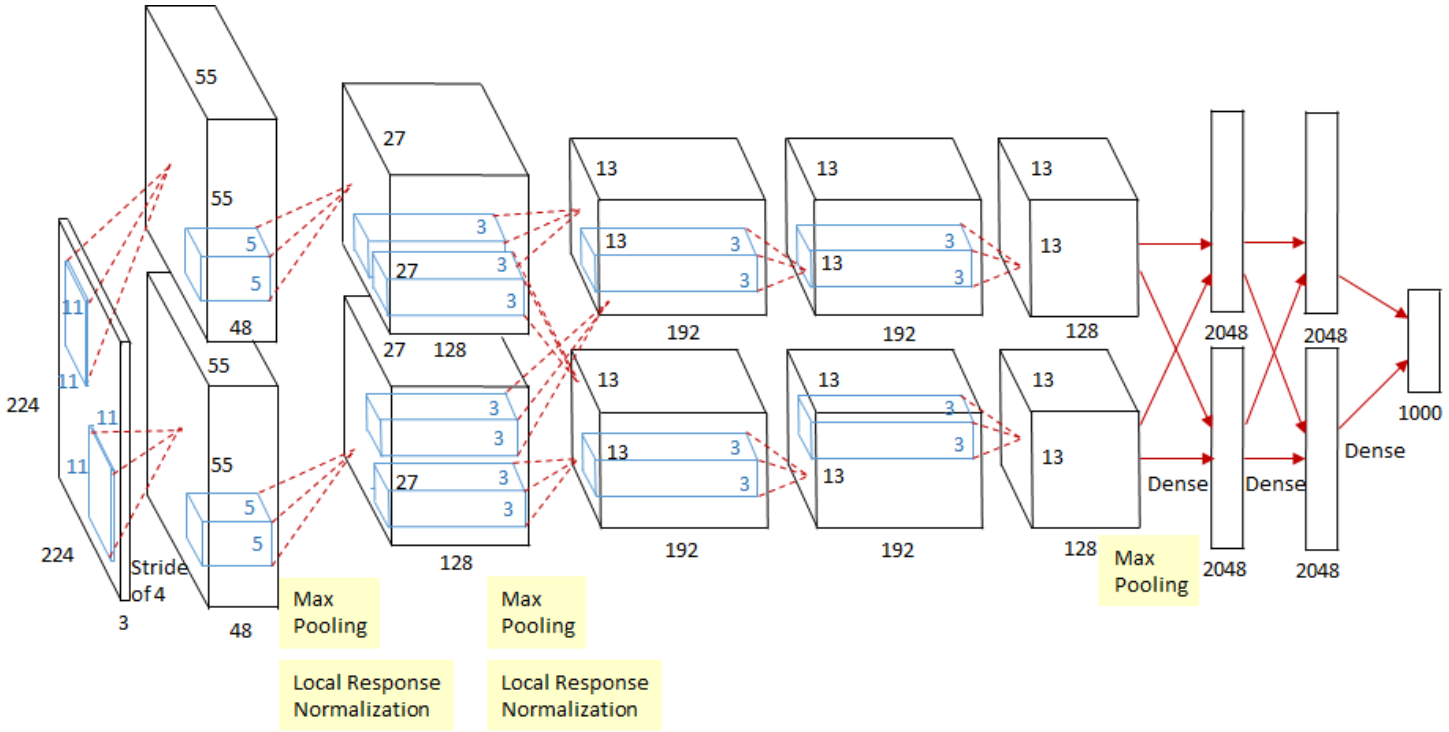

Figura 16 – Arquitetura da AlexNet (KRIZHEVSKY ET AL., 2012).

O conjunto de dados popularmente conhecido como ImageNet possui cerca de 1.200.000 imagens pertencentes a 1000 categorias. As imagens foram re-amostradas quando necessário para que tivessem sempre o tamanho fixo de 256 × 256 pixels.

Interessantemente, no artigo, discutem-se as vantagens de se usar ReLUs, o que reforça seu caráter pioneiro. Em seguida, fala-se sobre a implementação computacional e se explica que houve uma divisão dos *kernels* da rede em duas GPUs, o que explica a "simetria" das partes de cima e de baixo da Figura 16.

De uma perspectiva global, há oito camadas com pesos sinápticos: cinco camadas convolucionais, e três camadas totalmente conectadas (densas ou *fully-connected*). A última camada totalmente conectada é do tipo *softmax*, e gera estimativas da probabilidade de que uma entrada pertença a cada uma das 1000 classes.

A primeira camada convolucional filtra uma imagem $224 \times 224 \times 3$[†] com 96 *kernels* de tamanho $11 \times 11 \times 3$ (*stride* de 4 pixels); a segunda camada convolucional toma como resposta a saída da primeira camada (após *pooling*) e a filtra com 256 *kernels* de tamanho $5 \times 5 \times 48$; a terceira camada tem 384 kernels de tamanho $3 \times 3 \times 256$; a

[†] Veremos logo a seguir por que a imagem não tem 256×256 como se esperava.

quarta camada tem 384 *kernels* de tamanho 3 × 3 × 192 e a quinta camada tem 256 *kernels* de tamanho 3 × 3 × 192. As duas primeiras camadas densas têm 4096 neurônios.

A rede tem 60 milhões de parâmetros, o que gera preocupação com sobreajuste (*overfitting*). Uma primeira estratégia para combater esse problema foi realizar *data augmentation*, ou seja, aumentar artificialmente o tamanho do conjunto de dados sintetizando novos dados a partir dos existentes. Isso foi feito extraindo aleatoriamente *patches* de 224 × 224 das imagens 256 × 256 (e reflexões horizontais dos mesmos), aumentando o tamanho do conjunto por um fator de 2048. Sem esse esquema, o sobreajuste impediria o uso de uma rede profunda.

Outra forma de realizar *data augmentation* se deu por meio de análise de componentes principais no âmbito dos canais RGB das imagens.

Para combater o sobreajuste, também foi preciso recorrer a uma estratégia de regularização muito elegante, conhecida como *dropout*, que será descrita mais adiante.

O treinamento teve por base o algoritmo de gradiente estocástico (SGD, *stochastic gradient descent*) com *mini-batch* de 128 amostras. Usou-se um termo de momento e *weight decay*. O passo de adaptação (*learning rate*) foi igual para todas as camadas e ajustado segundo uma heurística simples baseada no erro de validação. A rede foi treinada por ~90 ciclos (épocas), num período de 5 a 6 dias, em duas GPUs NVIDIA GTX 580 3GB. Os resultados obtidos são apresentados na Figura 17.

Model	Top-1	Top-5
Sparse coding [2]	*47.1%*	*28.2%*
SIFT + FVs [24]	*45.7%*	*25.7%*
CNN	**37.5%**	**17.0%**

(a) ILSVRC-2010

Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
SIFT + FVs [7]	—	—	*26.2%*
1 CNN	40.7%	18.2%	—
5 CNNs	38.1%	16.4%	**16.4%**
1 CNN*	39.0%	16.6%	—
7 CNNs*	36.7%	15.4%	**15.3%**

(b) ILSVRC-2012

Figura 17 – Taxas de erro de classificação da AlexNet (KRIZHEVSKY ET AL., 2012).

Uma avaliação qualitativa da operação da AlexNet pode ser feita com base na Figura 18.

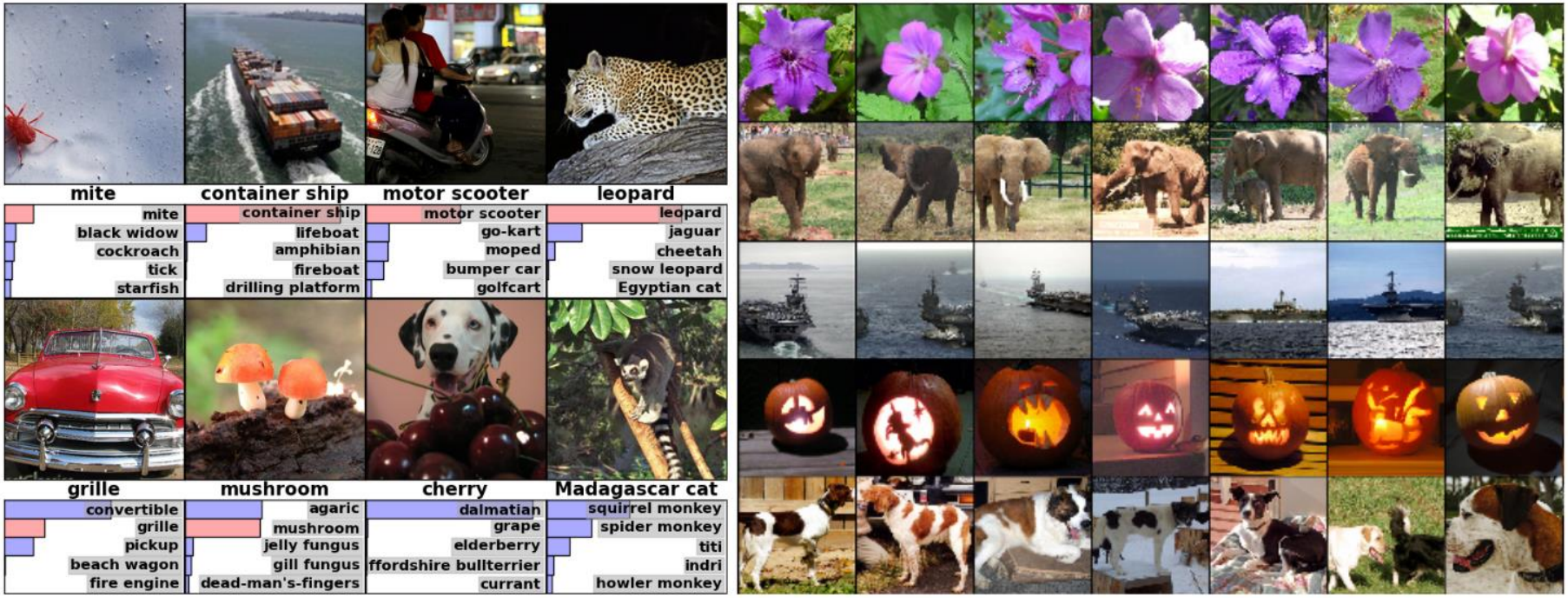

Figura 18 – À esquerda, oito imagens do conjunto de teste da ILSVRC-2010 são apresentadas, juntamente com os cinco rótulos indicados como mais prováveis pela AlexNet. À direita, cinco imagens de teste da ILSVRC-2010 são mostradas na primeira coluna. As colunas restantes exibem as seis imagens do conjunto de treinamento que produzem os vetores de atributos na última camada escondida com a menor distância Euclidiana em relação ao vetor de atributos das imagens de teste (KRIZHEVSKY ET AL., 2012).

2.4.2. GoogLeNet

A arquitetura GoogLeNet foi a vencedora da competição ILSVRC de 2014, atingindo uma taxa de erro (top-5) inferior a 7% (SZEGEDY ET AL., 2015). Este excelente desempenho veio em grande parte do fato de a rede ser muito mais profunda que as CNNs anteriores. Ao mesmo tempo, a GoogLeNet possui bem menos parâmetros que a AlexNet (cerca de 6 milhões de pesos, em vez de 60 milhões).

Diferencial: módulo *inception*.

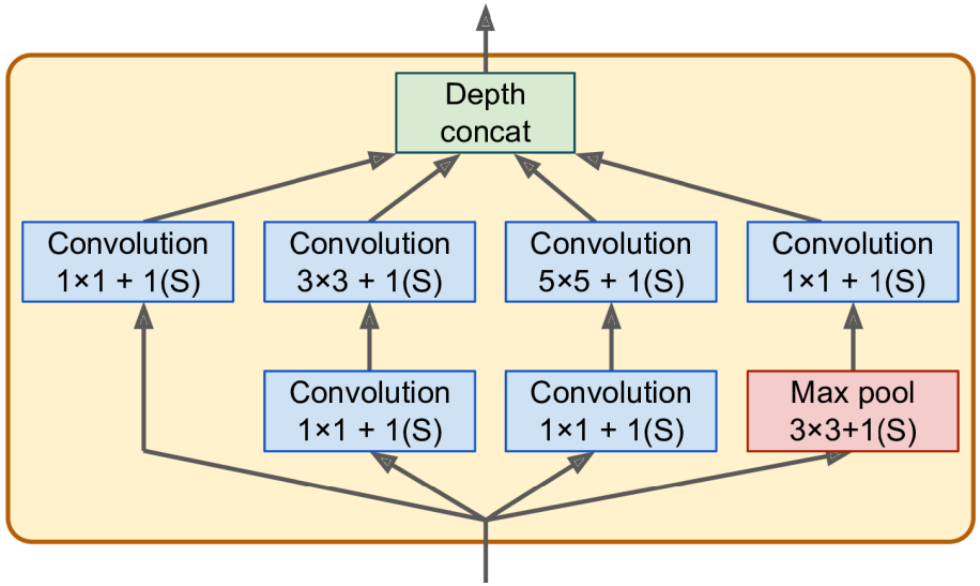

Figura 19 – Estrutura do módulo *inception*. Extraída de (GÉRON, 2019).

- A notação "$3 \times 3 + 1(S)$" indica que a camada convolucional usa um *kernel* 3×3 com *stride* unitário e *padding* para manter o tamanho da entrada inalterado.
- Essencialmente, a imagem é passada por diferentes camadas convolucionais, que exploram tamanhos de *kernel* distintos, o que permite que elas capturem padrões em diferentes escalas.
- Ao final, faz-se a concatenação de todos os *feature maps*, o que possível já que todas as camadas mantêm intacto o tamanho (altura e largura) das imagens.
- Embora as camadas $1 \times 1 + 1(S)$ não consigam capturar padrões espaciais, elas podem identificar padrões na dimensão da profundidade (i.e., entre os canais).

2.4.3 ResNet

O modelo vencedor da ILSVRC 2015, conhecido como *Residual Network* (ResNet) (He et al., 2016), foi capaz de alcançar um erro de classificação (top-5) abaixo de

3,6%. Esta arquitetura seguiu a tendência de usar redes bem mais profundas, mas com menos parâmetros ajustáveis.

O maior ponto de destaque, porém, está associado à inserção de conexões que criam atalhos para o fluxo de informação e, também, para a propagação dos gradientes. As chamadas *skip connections* levam a entrada de uma camada a um ponto mais adiante da estrutura, como ilustrado na Figura 20.

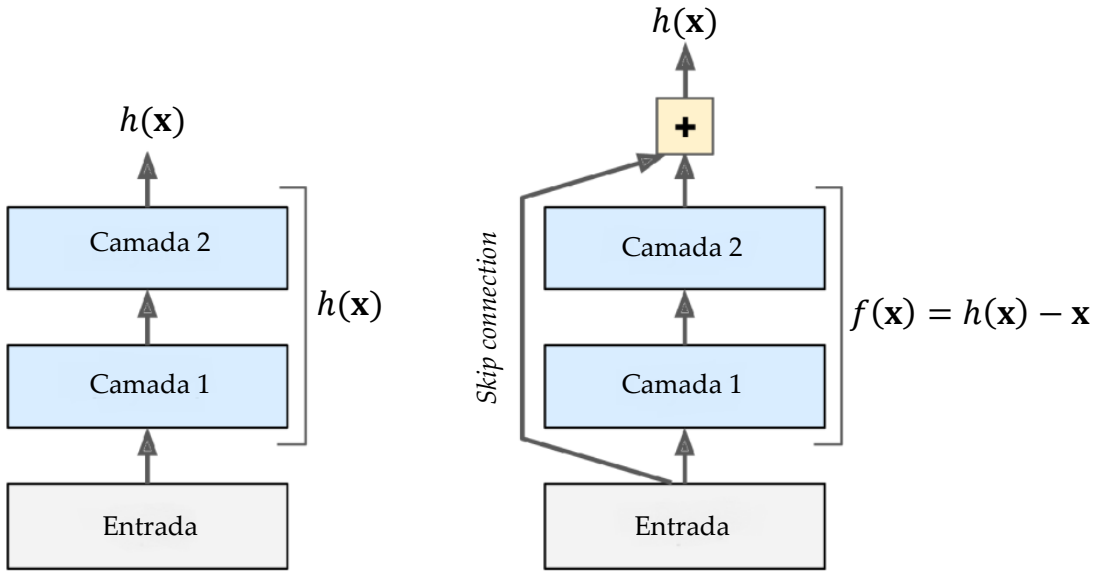

Figura 20 – Contraste entre uma estrutura sem e com a ideia de *residual learning*. Adaptada de (GÉRON, 2019).

Se a entrada $\mathbf{x}$ é somada à saída da rede e gostaríamos de aproximar a função $h(\mathbf{x})$, então a rede é forçada a modelar $f(\mathbf{x}) = h(\mathbf{x}) - \mathbf{x}$, em vez de $h(\mathbf{x})$. Esta abordagem é conhecida como *residual learning*.

A Figura 21 exibe um resumo da arquitetura da ResNet, com destaque para as *residual units*, nas quais há as *skip connections*.

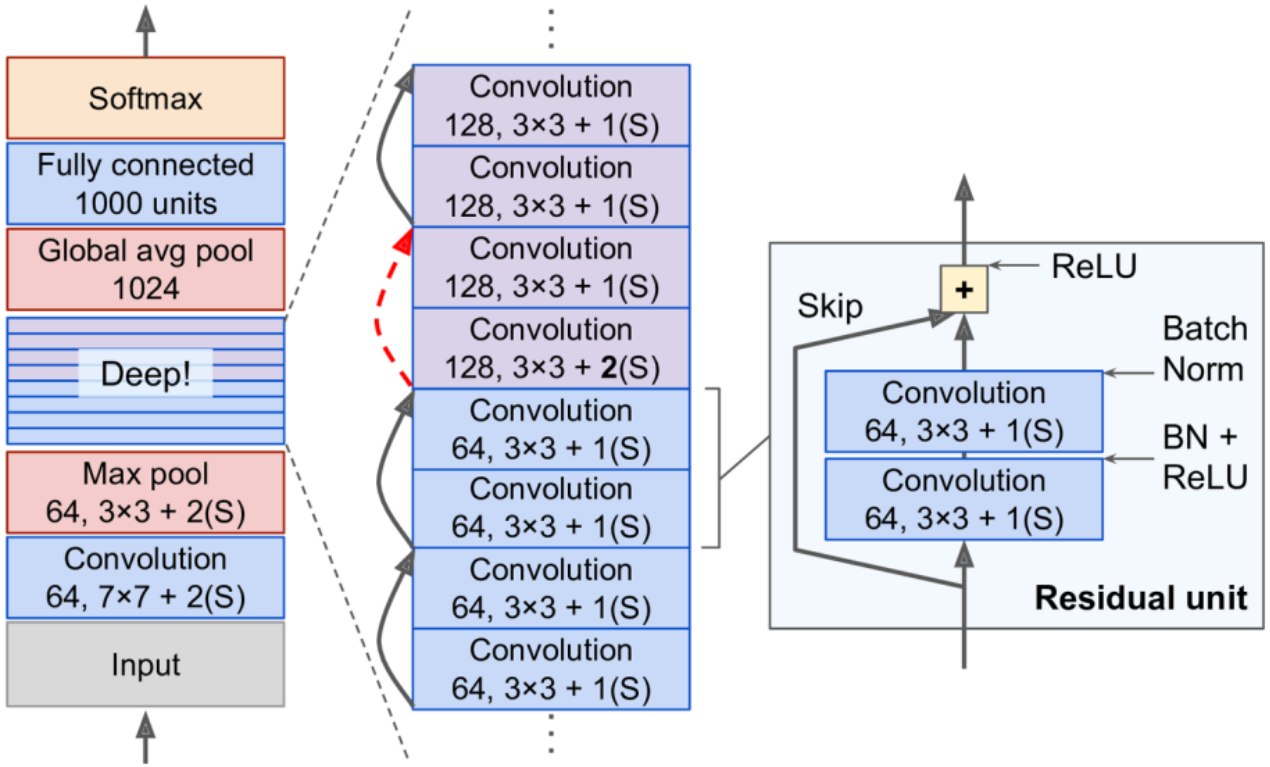

Figura 21 – Arquitetura da ResNet. Extraída de (GÉRON, 2019).

2.4.4. Extração de características

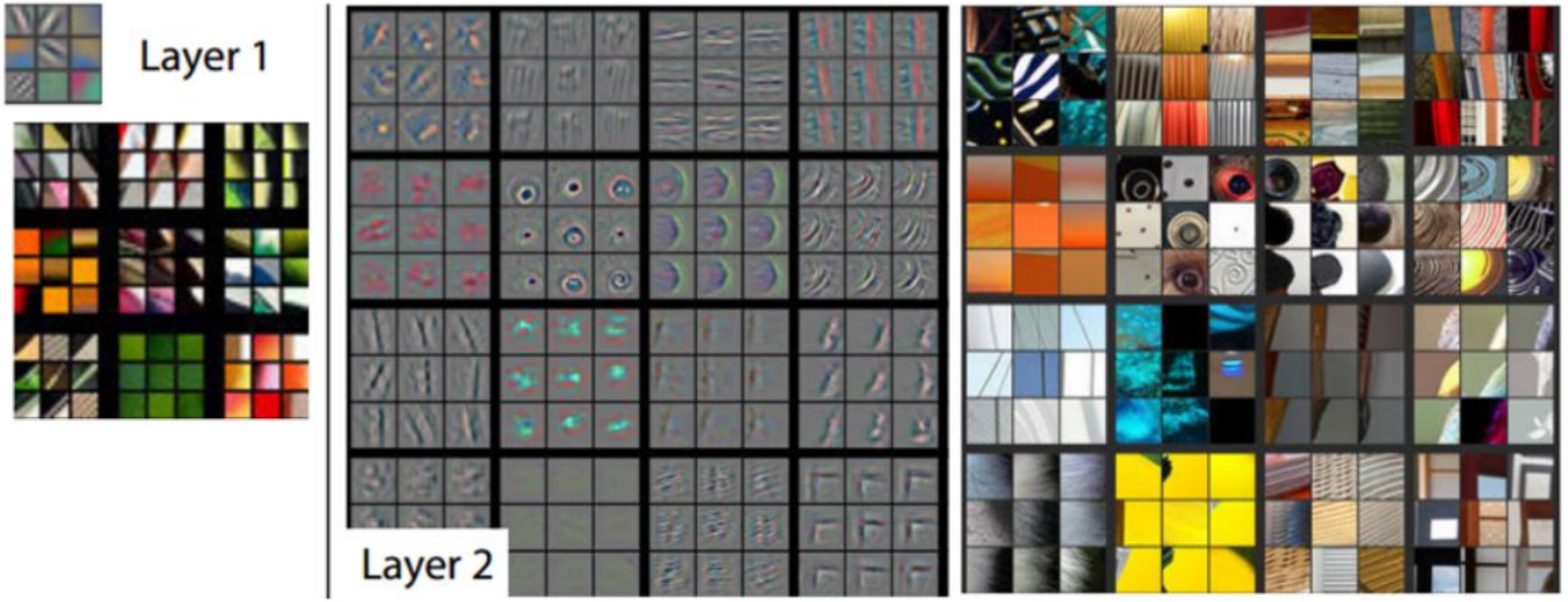

Figura 22 – A primeira camada de uma CNN atua como um detector de atributos de baixo nível, reconhecendo elementos mais simples, como bordas ou cores. Com a segunda camada, atributos circulares estão sendo detectados (ZEILER & FERGUS, 2013).

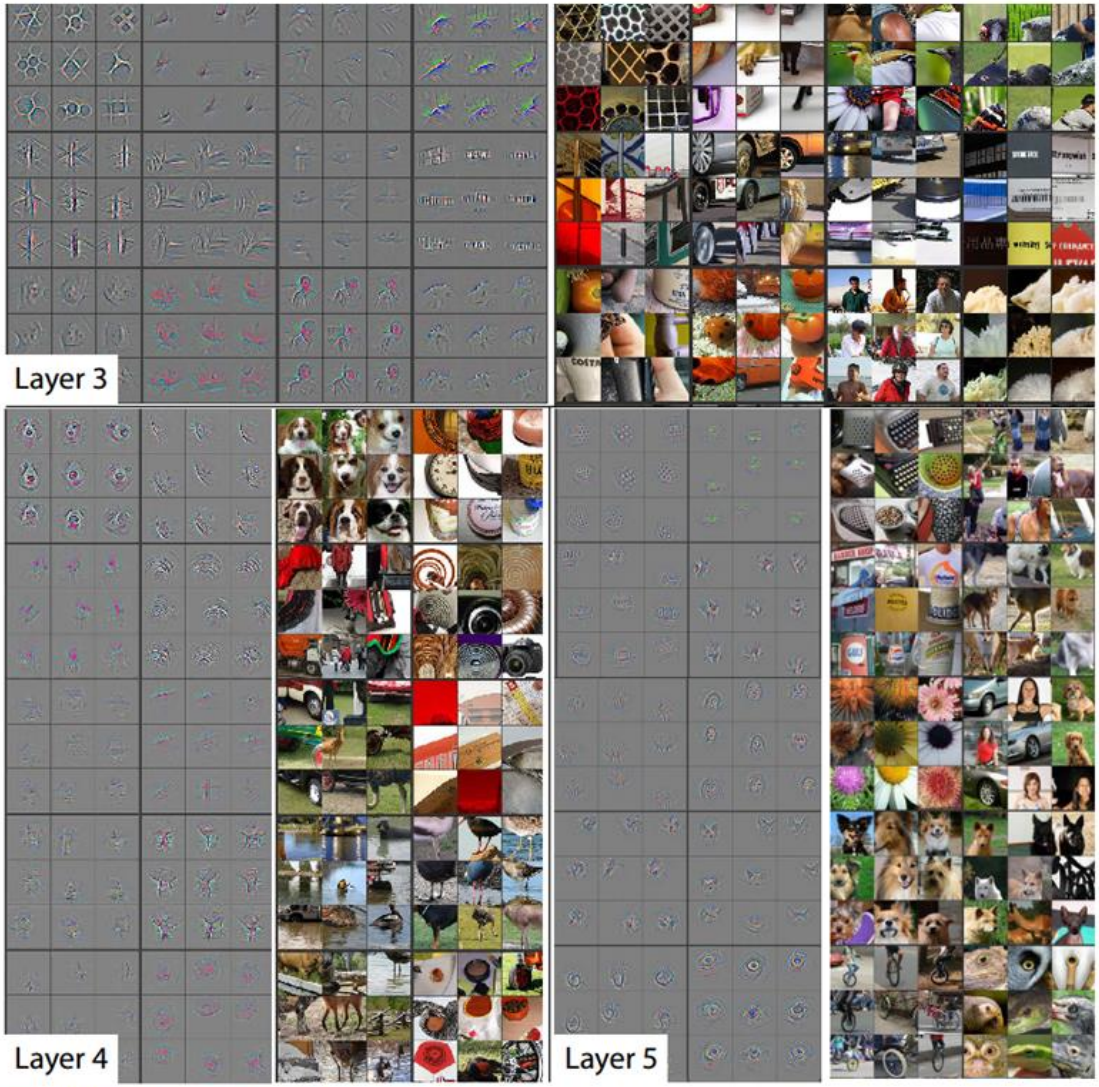

Figura 23 – As camadas subsequentes reconhecem características de um nível mais alto de abstração, como flores ou faces de animais (ZEILER & FERGUS, 2013).

2.4.5 Estado-da-arte na ImageNet

Para encerrar esta visão panorâmica sobre redes convolucionais, é pertinente salientar os avanços expressivos que tais estruturas conquistaram, tomando como exemplo o problema de classificação de imagens na base ImageNet.

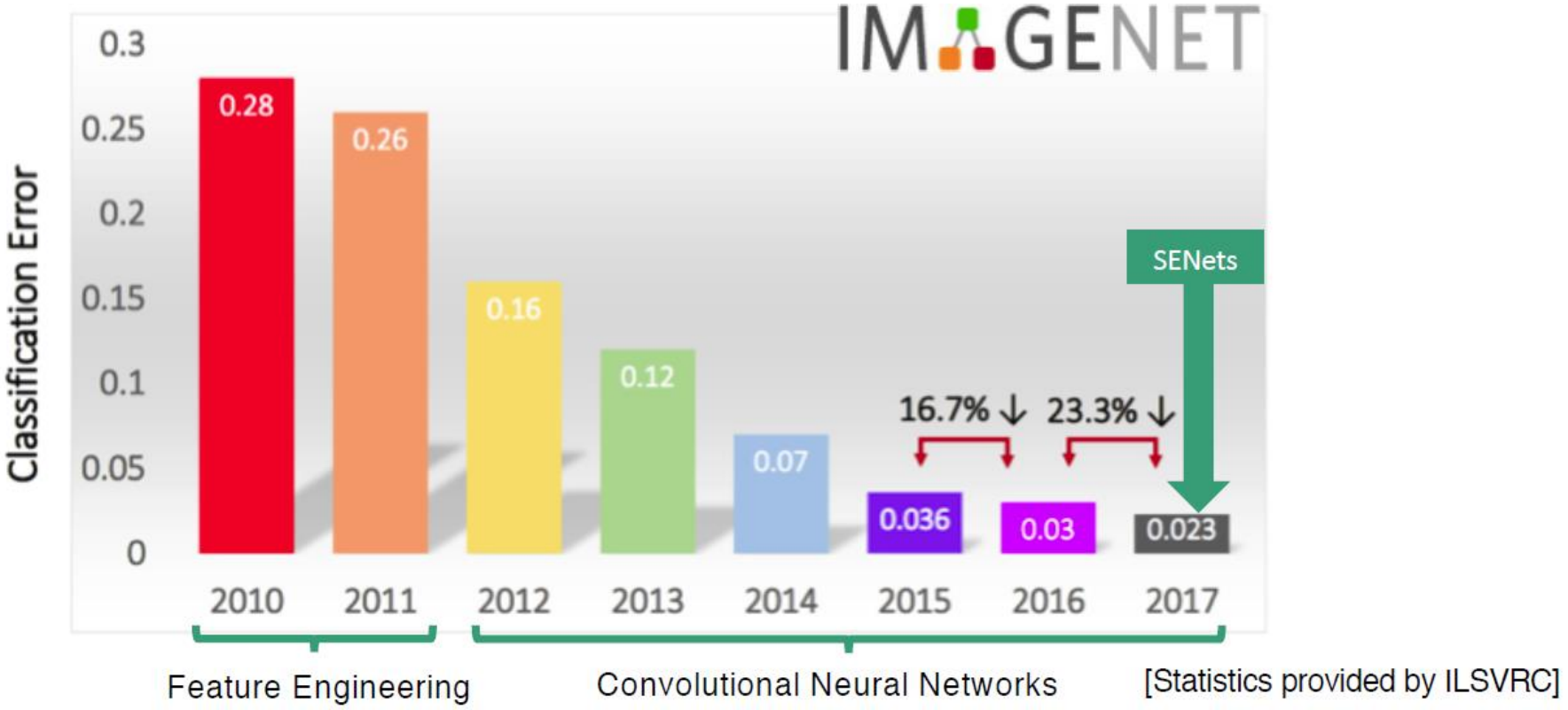

Figura 24 – Evolução do erro de classificação (top-5) desde 2010 na base ImageNet. Extraída de https://www.kaggle.com/getting-started/149448.

3. *Data Augmentation*

Durante a exposição sobre a AlexNet, mencionamos que os autores lançaram mão de alguns recursos para expandir a base de dados a fim de reduzir a chance de *overfitting* (KRIZHEVSKY ET AL., 2012). Este tipo de abordagem, denominada *data augmentation*, consiste em artificialmente aumentar o tamanho do conjunto de treinamento gerando diversas variações plausíveis para cada amostra.

As instâncias sintetizadas devem ser tão realistas quanto possível: idealmente, dada uma imagem do conjunto de treinamento ampliado, um avaliador humano não deveria ser capaz de dizer se ela é uma imagem que pertencia ao conjunto original ou não. Por exemplo, podemos suavemente deslocar, rotacionar e redimensionar as imagens de treinamento e acrescentar estas versões modificadas ao conjunto. Isto tende a forçar o modelo a ser mais tolerante a variações na posição, orientação e tamanho dos objetos nas imagens.

Para que o modelo seja mais tolerante a diferentes condições de iluminação, podemos gerar imagens com contrastes variados. Ademais, combinando estas transformações, podemos ampliar de forma significativa o tamanho do conjunto de treinamento. A Figura 25 apresenta algumas transformações que podem ser aplicadas sobre uma imagem para expandir o conjunto de treinamento.

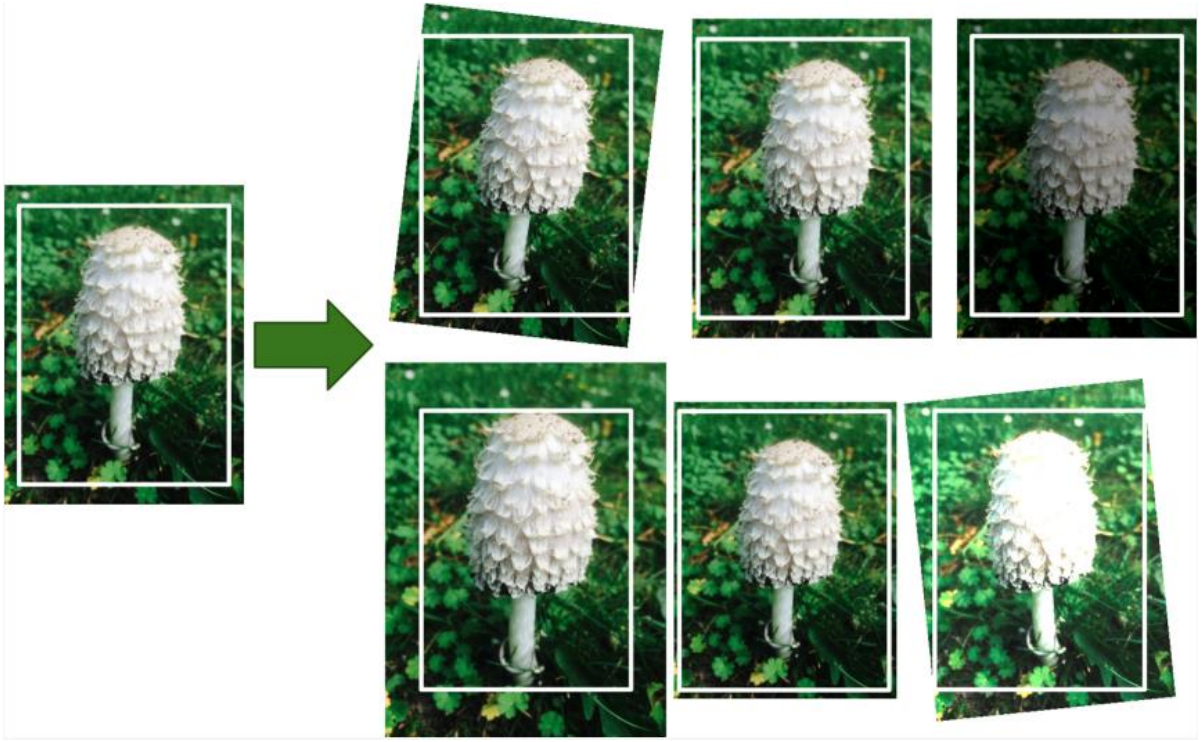

Figura 25 – Exemplos de instâncias obtidas através de transformações da imagem original (GÉRON, 2019).

4. *Dropout*

Dropout é uma técnica de regularização bastante elegante para modelos profundos (SRIVASTAVA ET AL., 2014), cuja ideia é bastante simples: em cada passo de treinamento, os neurônios têm uma probabilidade p de serem temporariamente desligados, o que significa que eles ficam "fora de cena" durante aquele passo, não participando nem da etapa *forward* nem da retropropagação de erro. A Figura 26 ilustra a aplicação de *dropout*.

Cada vez que se apresenta um padrão, a rede neural atua, na prática, com uma arquitetura distinta, mas com pesos em comum. Desta forma, cada neurônio é encorajado a desempenhar um papel útil para a rede por conta própria, sem depender tanto de outros neurônios vizinhos. Além disso, ele não pode limitar-se ao uso de algumas poucas entradas: todas devem ser consideradas em seu processamento.

Em suma, *dropout* contribui para tornar a rede mais robusta e que generalize melhor ao diminuir a ocorrência de co-adaptações complexas.

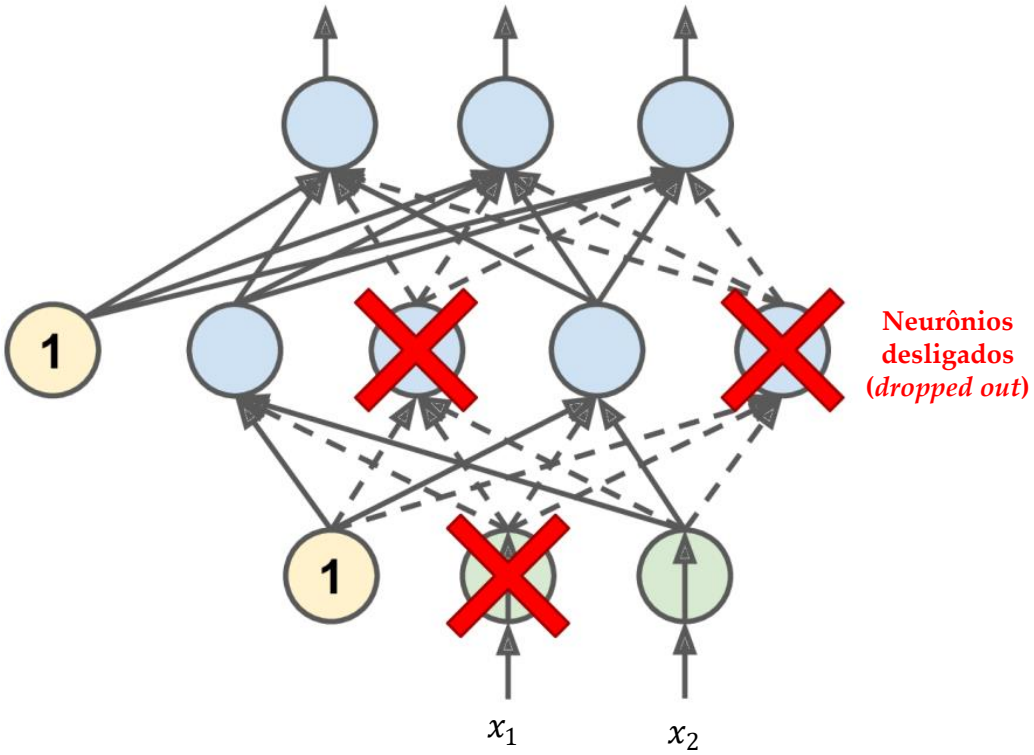

Figura 26 – Com *dropout*, alguns neurônios, inclusive os da camada de entrada, são aleatoriamente desligados. Nesta iteração, é como se eles não estivessem presentes na estrutura da rede, de modo que suas ativações são nulas e os seus pesos não participam do ajuste. Adaptada de (GÉRON, 2019).

Interessantemente, treinar uma rede neural com n unidades aplicando *dropout* pode ser interpretado como o treinamento implícito de 2^n redes neurais mais enxutas e que compartilham pesos sinápticos, com cada uma das possíveis redes neurais mais enxutas sendo treinada muito raramente. O comportamento da rede resultante pode, então, ser vista como uma média deste comitê de redes mais compactas.

Uma vez finalizado o treinamento, aplica-se uma correção sobre os pesos ou, analogamente, sobre as ativações dos neurônios para refletir o seu valor médio. Ou seja, uma única rede neural sem sofrer *dropout* é utilizada na etapa de teste, sendo que cada peso obtido no treinamento é multiplicado pela probabilidade de inclusão $(1 - p)$ de sua correspondente unidade.

5. Transferência de aprendizado

Transfer learning (TL) é o aprimoramento do aprendizado em uma nova tarefa através da transferência de conhecimento de uma tarefa relacionada que já foi aprendida (OLIVAS ET AL., 2009).

De forma resumida, a ideia trazida por TL consiste em aproveitar o conhecimento adquirido por um modelo em uma tarefa para auxiliar na solução de uma nova tarefa similar. Por exemplo, uma rede profunda treinada com sucesso na base ImageNet possui camadas especializadas em reconhecer diversos tipos de características nas imagens, desde bordas e contornos até partes de objetos. Então, diante de uma nova tarefa de classificação de imagens, é possível reutilizar aquela rede, com pesos já adaptados, como ponto de partida. Assim, em vez de iniciar do zero um treinamento, podemos explorar parte de uma rede profunda já treinada e,

então, fazer apenas uma etapa de refinamento do modelo considerando os dados do novo problema. A Figura 27 ilustra o uso de TL.

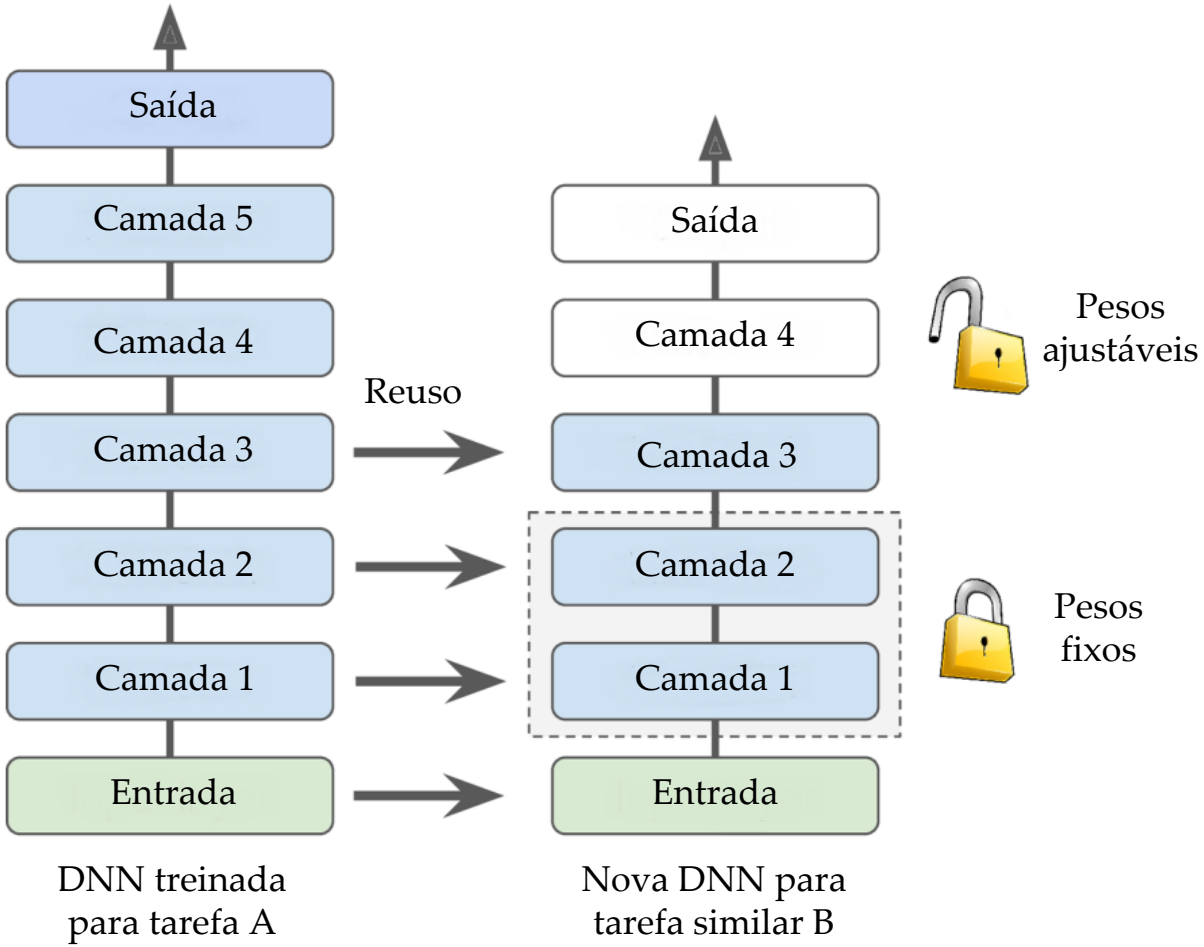

Figura 27 – Reuso de camadas pré-treinadas, com treinamento das camadas de saída. Adaptada de (GÉRON, 2019)

TL pode ser particularmente interessante quando a quantidade de dados disponíveis é bastante limitada, já que, neste contexto, o treinamento completo de um modelo profundo traz muitas preocupações com relação a *overfitting*.

6. Referências bibliográficas

LECUN, Y., BOTTOU, L., BENGIO, Y., HAFFNER, P., "Gradient-Based Learning Applied to Document Recognition", *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, 1998.

FUKUSHIMA, K., "Neocognitron: A Self-Organized Neural Network Model for a Mechanism of Patter Recognition Unaffected by Shift in Position", *Biological Cybernetics*, vol. 36, pp. 193-202, 1980.

GÉRON, A., **Hands-on Machine Learning with Scikit-Learn, Keras & Tensorflow**, O'Reilly Media, 2ª ed., 2019.

GONZALEZ, R. C., WOODS, R. E., **Digital Image Processing**, Pearson, 3ª ed., 2010.

GOODFELLOW, I., BENGIO, Y., COURVILLE, A., **Deep Learning**, MIT Press, 2016.

HE, K., ZHANG, X., REN, S., SUN, J., "Deep Residual Learning for Image Recognition", Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770-778, 2016.

HUBEL, D. H., WIESEL, T. N., "Receptive Fields of Single Neurons in the Cat's Striate Cortex", *The Journal of Physiology*, vol. 148, pp. 574-591, 1959.

KRIZHEVSKY, A., SUTSKEVER, I., HINTON, G. E., "ImageNet Classification with Deep Convolutional Neural Networks", Advances in Neural Information Processing Systems, 2012.

LECUN, Y., BENGIO, Y., HINTON, G., "Deep Learning", *Nature*, vol. 521, pp. 436 – 444, 2015.

MARQUES, A. C. R., "Contribuição à Abordagem de Problemas de Classificação por Redes Convolucionais Profundas", Tese de Doutorado, Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, 2017.

OLIVAS, E. S., GUERRERO, J. D. M., SOBER, M. M., BENEDITO, J. R. M., LOPEZ, A. J. S., **Handbook of Research on Machine Learning Applications and Trends: Algorithms**, Methods and Techniques, Information Science Reference, 2009.

SRIVASTAVA, N., HINTON, G., KRIZHEVSKY, A., SUTSKEVER, I., SALAKHUTDINOV, R. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", *Journal of Machine Learning Research*, vol. 15, pp. 1929-1958, 2014.

SZEGEDY, C., LIU, W., JIA, Y., SERMANET, P., REED, S., ANGUELOV, D., ERHAN, D., VANHOUCKE, V., RABINOVICH, A., "Going Deeper with Convolutions", Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CPVR), pp. 1-9, 2015.

VON ZUBEN, F. J., **Notas de Aulas do Curso "Redes Neurais" (IA353),** disponíveis em *http://www.dca.fee.unicamp.br/~vonzuben/courses/ia353.html*

ZEILER, M. D., FERGUS, R., "Visualizing and Understanding Convolutional Networks", *arXiv:1311.2901*, 2013.