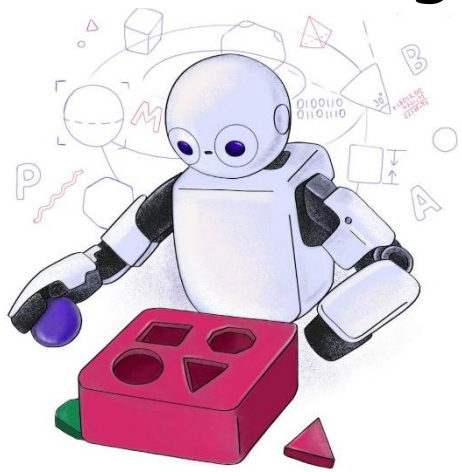
TP557 - Tópicos avançados em IoT e Machine Learning:

Introduzindo Convoluções



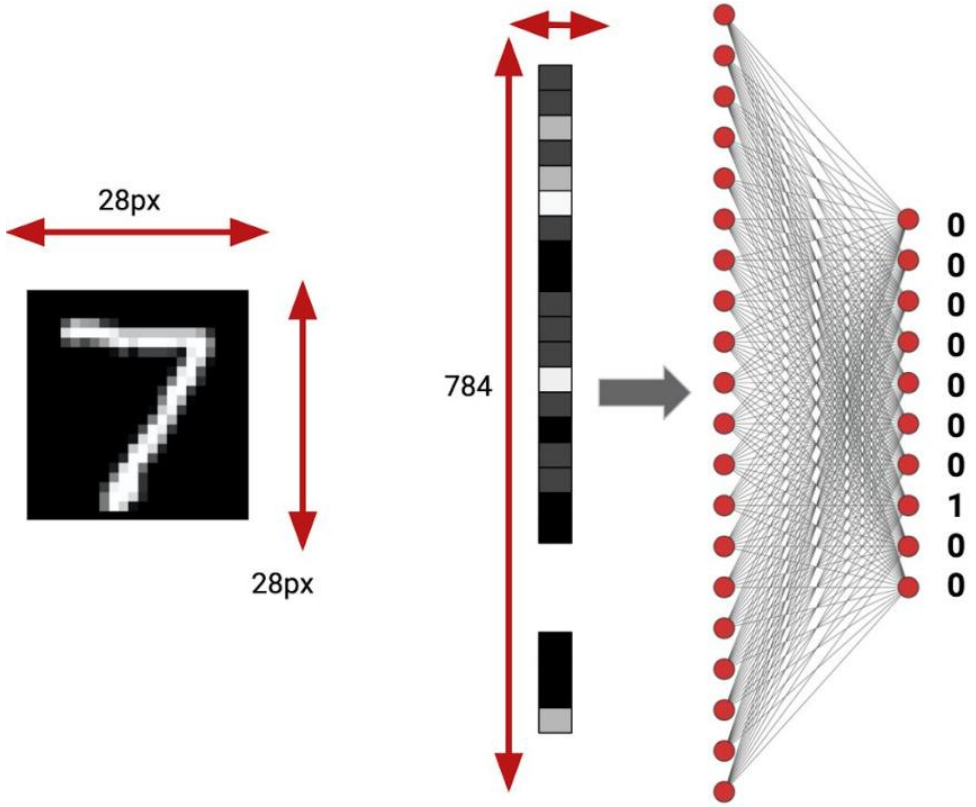
Inatel

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

O que vamos ver?

- Até agora, nossas redes neurais continham apenas dois tipos de camadas: densas e de achatamento.
- Porém, um outro tipo muito importante são as ***camadas convolucionais***.
- Essas camadas formam as *Convolutional Neural Networks* (CNNs).
- A principal diferente para uma DNN é que ao invés de aprender os pesos das camadas densas, uma CNN aprende os valores de ***filtros de convolução*** (ou apenas ***filtros***).
  - Esses filtros são muito eficientes em “compreender” o conteúdo de uma imagem ou vídeo.
- CNNs são usadas em tarefas de visão computacional, como, por exemplo, reconhecimento de objetos, detecção de padrões, segmentação de imagens, rastreamento de objetos, etc.

Dados tabulares

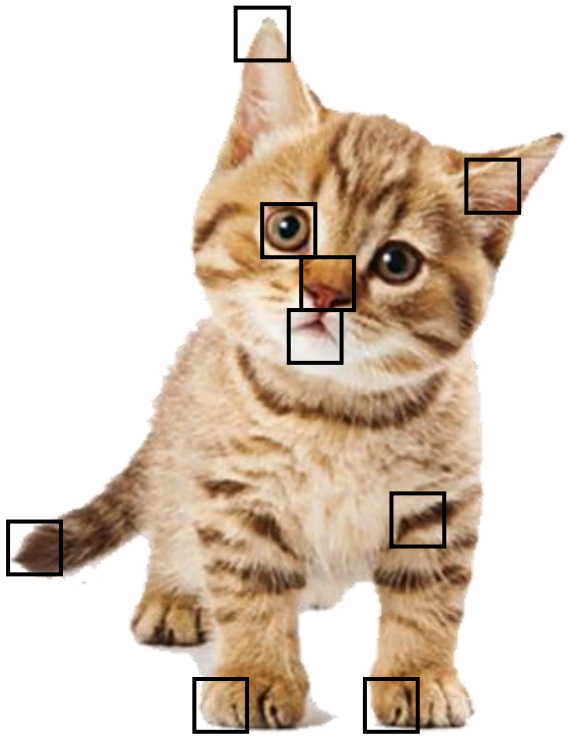


- DNNs são ideais para dados tabulares, onde os exemplos são representados por linhas e os atributos por colunas.
- Ao analisar esses dados, o objetivo de uma DNN é descobrir padrões que envolvem interações entre os atributos, sem presumir uma estrutura espacial (em termos de posicionamento físico) específica entre eles.
- Em contraste, imagens têm uma estrutura espacial que pode ser explorada por modelos de ML.

Estrutura espacial

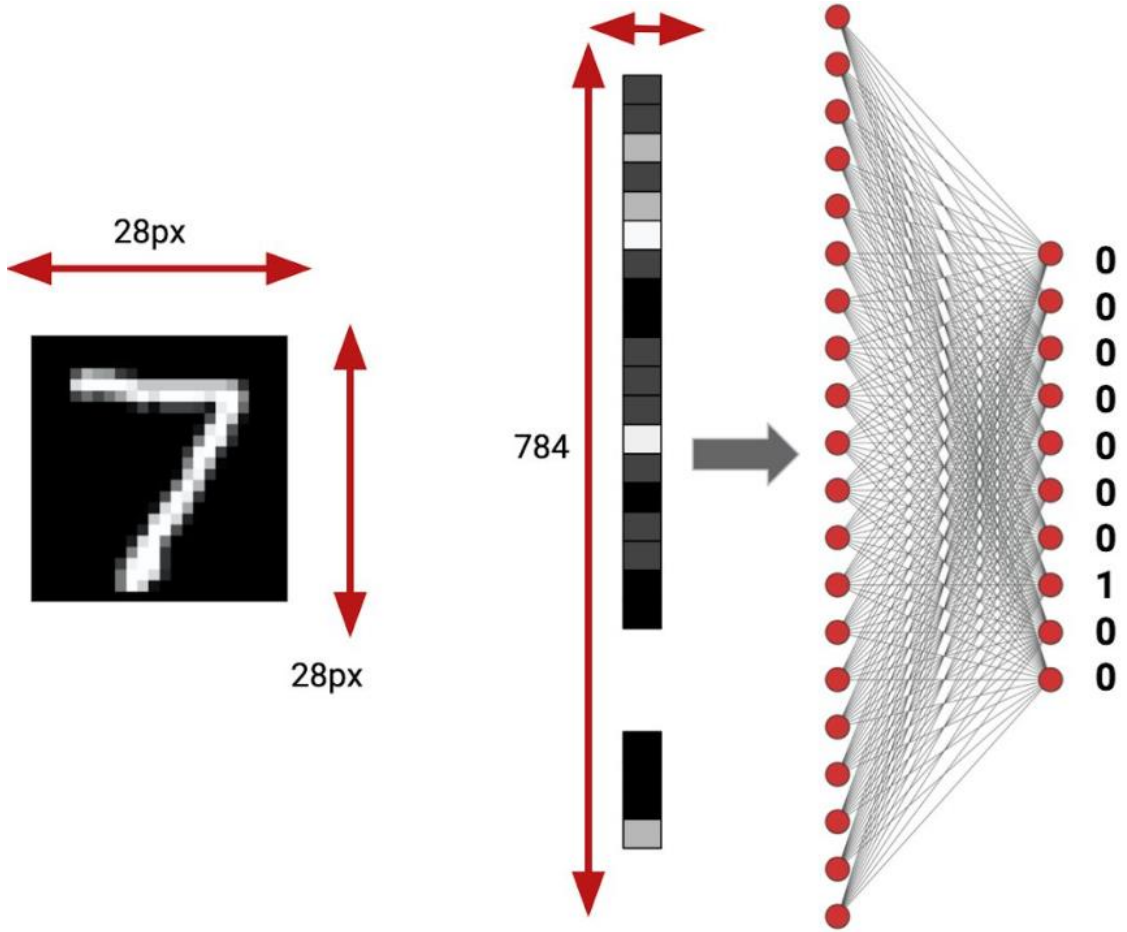
- DNNs não levam em consideração a estrutura espacial dos dados, tratando *pixels* de entrada distantes uns dos outros da mesma forma que *pixels* próximos.
  - Por exemplo, se embaralharmos os pixels das imagens de entrada de uma DNN treinada para classificar imagens de gatos e cachorros, ela ainda identificará os animais, mesmo as imagens não mais fazendo sentido visual algum.

Estrutura espacial



- DNNs ignoram a ***localidade*** de referência em dados com topologia de grade (como imagens), tanto computacional quanto semanticamente.
- Assim, a conectividade total dos neurônios é um desperdício para propósitos como o reconhecimento de imagens que são dominados por ***padrões espacialmente locais***.

Imagens simples

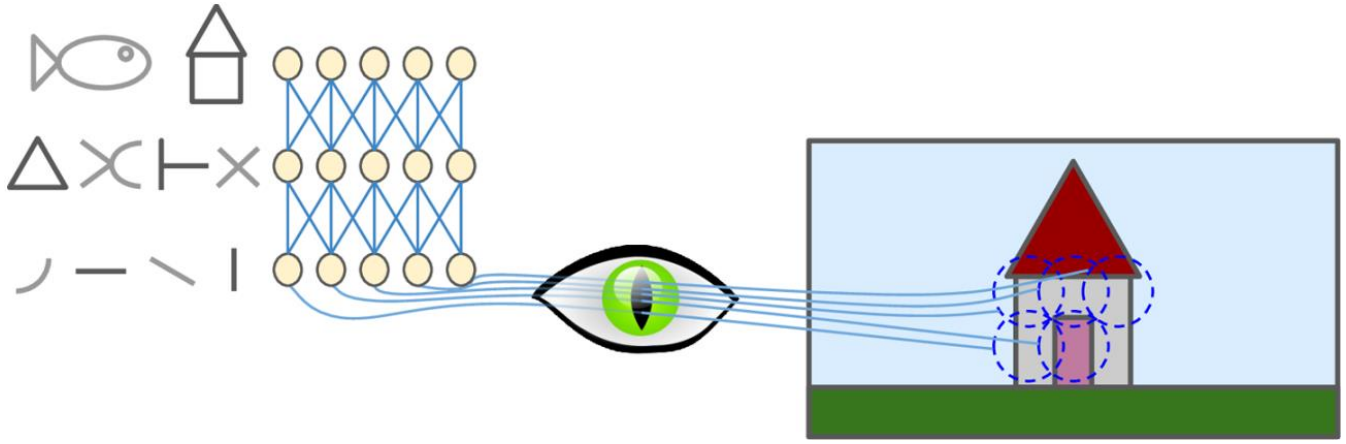


- Até o momento, as imagens que usamos nos problemas de classificação eram bem simples.
- Eram imagens em tons de cinza, com objetos centralizados, sem muita variação em termos de rotação, iluminação, escala, com um mesmo fundo, sem oclusões (i.e., partes do objeto obstruídas), etc.

Imagens complexas

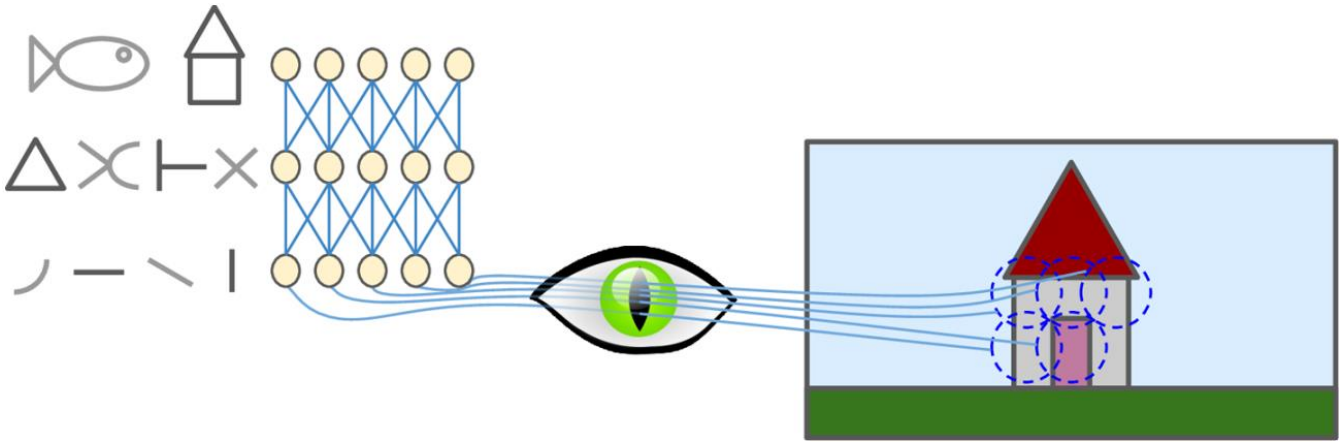
- Mas e quando as imagens são mais complexas?
- Com cores, resoluções variadas, objetos não centralizados, com variação em termos de rotação, iluminação, escala, diferentes fundos, oclusões, etc.
- Usar ***filtros de convolução*** pode nos ajudar com esses problemas.
- Por exemplo, e se eu quiser classificar entre pessoas e cavalos?

Neurônios biológicos



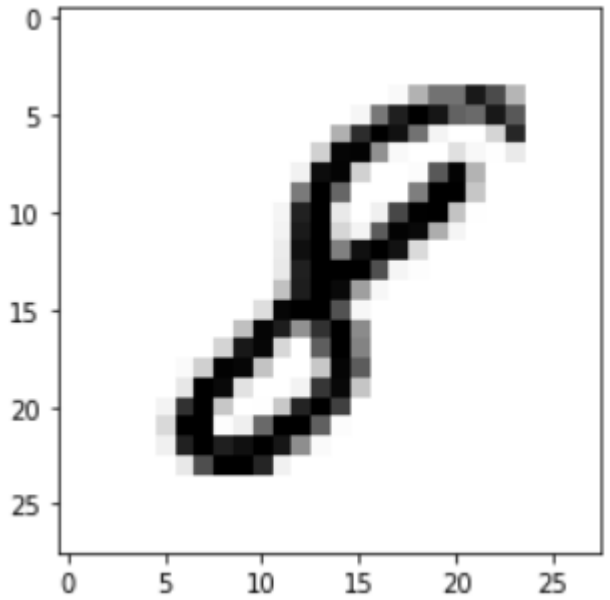
- Os neurônios biológicos no córtex visual respondem a padrões específicos em pequenas regiões do campo visual chamadas ***campos receptivos***.
- À medida que o sinal visual percorre as camadas do cérebro, os neurônios respondem a padrões mais complexos em campos receptivos maiores.

Neurônios biológicos



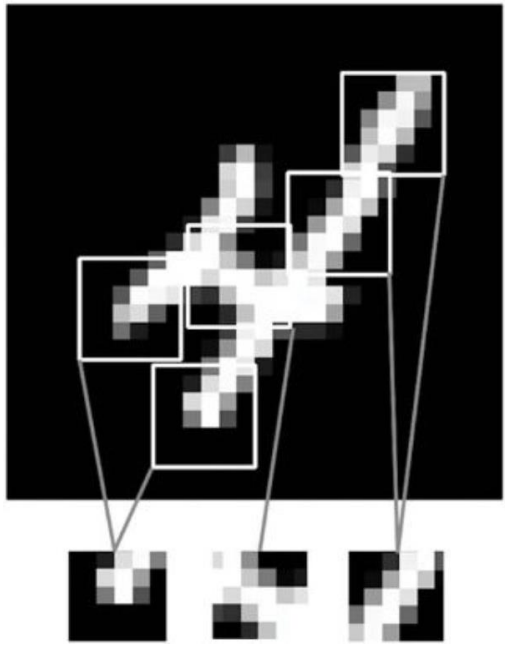
- Alguns neurônios reagem apenas a imagens de linhas horizontais, enquanto outros reagem apenas a linhas com orientações diferentes.
- Outros neurônios têm campos receptivos maiores e reagem a padrões mais complexos que são combinações de padrões de nível inferior.

Diferença entre camadas densas e convolucionais



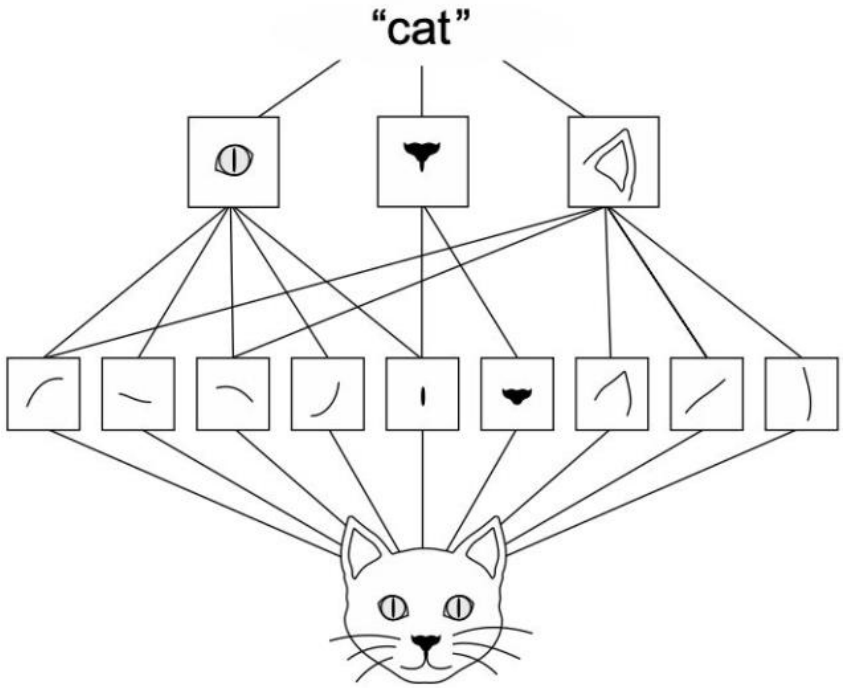
- Como veremos, a diferença fundamental entre uma camada densamente conectada e uma de convolução é:
- As camadas densas aprendem ***padrões globais*** em seu espaço de atributos.
- Por exemplo, para um dígito da base de dados MNIST, as camadas densas aprendem ***padrões envolvendo todos os pixels***.

Diferença entre camadas densas e convolucionais



- As camadas de convolução ***aprendem padrões locais***.
- No caso de imagens, as camadas de convolução aprendem padrões encontrados em pequenas janelas 2D das entradas.
- Para aprender esses padrões, a camada utiliza ***filtros de convolução***, também chamados de *kernels*.

Diferença entre camadas densas e convolucionais



- Camadas convolucionais aprendem ***hierarquias espaciais*** de padrões.
- Uma primeira camada de convolução aprenderá pequenos padrões locais, como arestas (i.e., bordas), uma segunda camada de convolução aprenderá padrões maiores criados a partir da combinação das características das primeiras camadas e assim por diante.

Canais

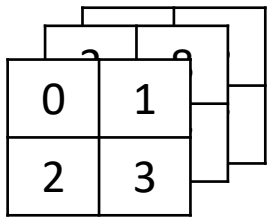
- Antes de falarmos sobre convolução, vamos falar sobre cores.
- Até agora, ignoramos que imagens, em geral, consistem em três canais: vermelho (R), verde (G) e azul (B).
- Imagens coloridas têm canais RGB para indicar a quantidade de vermelho, verde e azul.
- Em suma, as imagens não são objetos bidimensionais, mas sim tensores de três dimensões, caracterizados por altura, largura e canal.

Filtros de convolução ou *kernels*

Kernel 2D

0	1
2	3

Kernel 3D



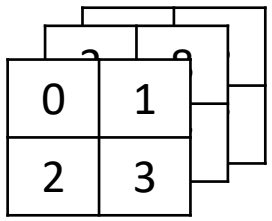
- Um ***kernel*** é um tensor (em geral em 3D) responsável por detectar características específicas em uma imagem.
- Ele percorre uma imagem e realiza ***operações convolução*** entre seus valores e os dos pixels na região da imagem correspondente a ele.

Filtros de convolução ou *kernels*

Kernel 2D

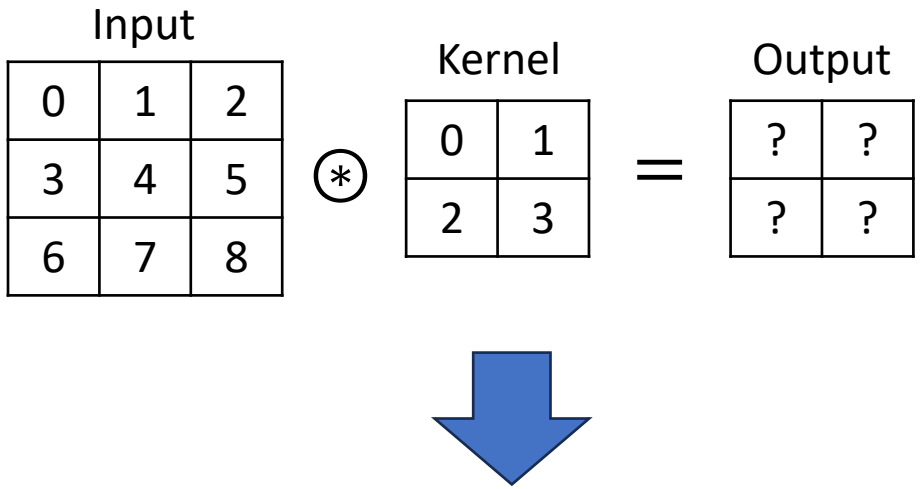
0	1
2	3

Kernel 3D



- Cada ***kernel*** é projetado para ***extrair/detectar um tipo particular de característica***, como bordas, texturas, padrões ou partes específicas de objetos.
- Os ***kernels*** aprendem automaticamente a ***detectar*** as características mais relevantes para a tarefa específica em mãos.
  - Eles aprendem, a partir de seu treinamento, os valores dos elementos do tensor.

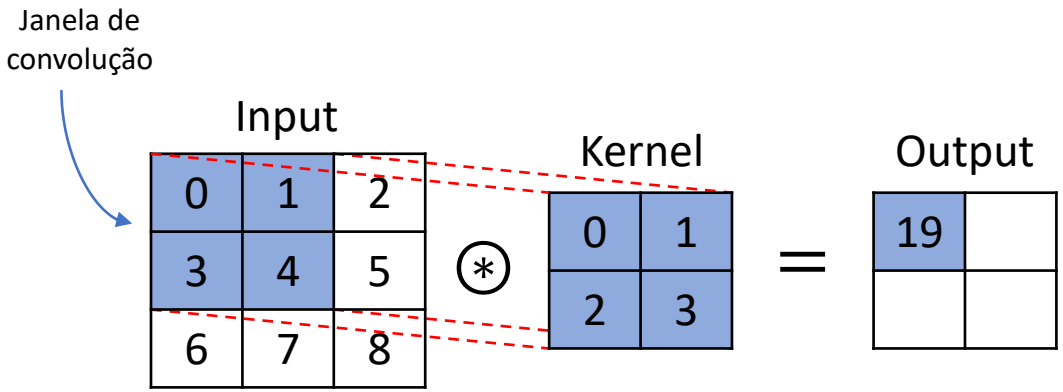
Operação de convolução com 1 canal



$$o(i,j) = \sum_m \sum_n I(i+m, j+n)\, K(m,n)$$

- Vamos ignorar os canais por enquanto e ver como uma operação de convolução funciona com dados bidimensionais.
- O símbolo ⊛ representa a operação de “convolução”.
- A entrada é chamada de ***input feature map***.
- O ***filtro*** também é chamado de “***kernel***”.
- O operação é representada pela equação ao lado.

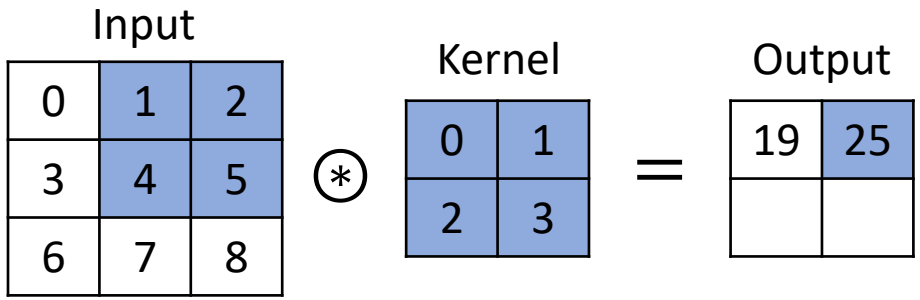
Operação de convolução com 1 canal



- Ao calcular a convolução, começamos com a ***janela de convolução*** no canto superior esquerdo do tensor de entrada.

$$\text{output}(i,j)$$
$$= \sum_m \sum_n I(i+m, j+n)\, K(m,n)$$
$$= 0 * 0 + 1 * 1 + 3 * 2 + 4 * 3 = 19$$

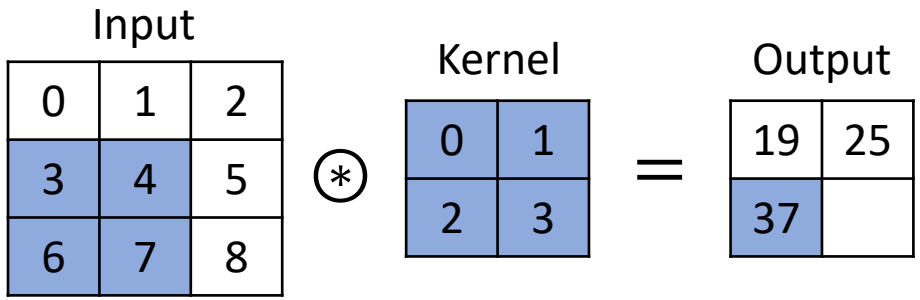
Operação de convolução com 1 canal



- Em seguida, deslizamos a janela um elemento para a direita.

$$\text{output}(i,j)$$
$$= \sum_m \sum_n I(i+m, j+n)\, K(m,n)$$
$$= 1 * 0 + 2 * 1 + 4 * 2 + 5 * 3 = 25$$

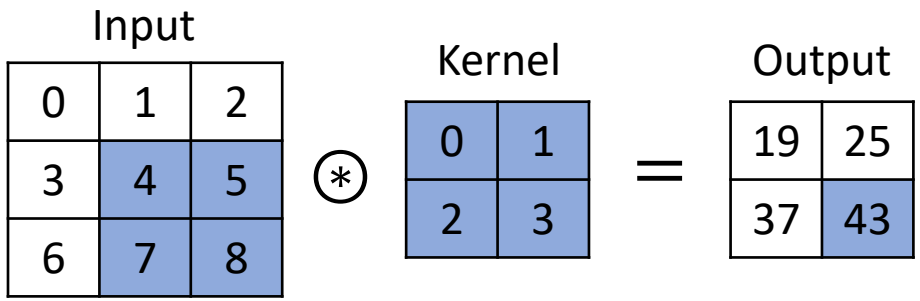
Operação de convolução com 1 canal



- Ao chegar-se ao final das colunas do tensor de entrada, volta-se ao seu início, deslizando a janela um elemento para baixo, ou seja, uma linha.

$$\text{output}(i,j)$$
$$= \sum_m \sum_n I(i+m, j+n)\, K(m,n)$$
$$= 3 * 0 + 4 * 1 + 6 * 2 + 7 * 3 = 37$$

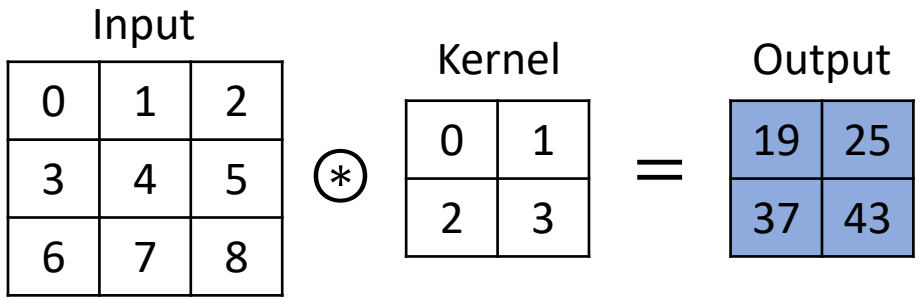
Operação de convolução com 1 canal



- Em seguida, deslizamos a janela um elemento para a direita.
- Esse processo se repete até que a janela de convolução tenha percorrido todo o tensor de entrada.

$$\text{output}(i,j)$$
$$= \sum_m \sum_n I(i+m, j+n)\, K(m,n)$$
$$= 4 * 0 + 5 * 1 + 7 * 2 + 8 * 3 = 43$$

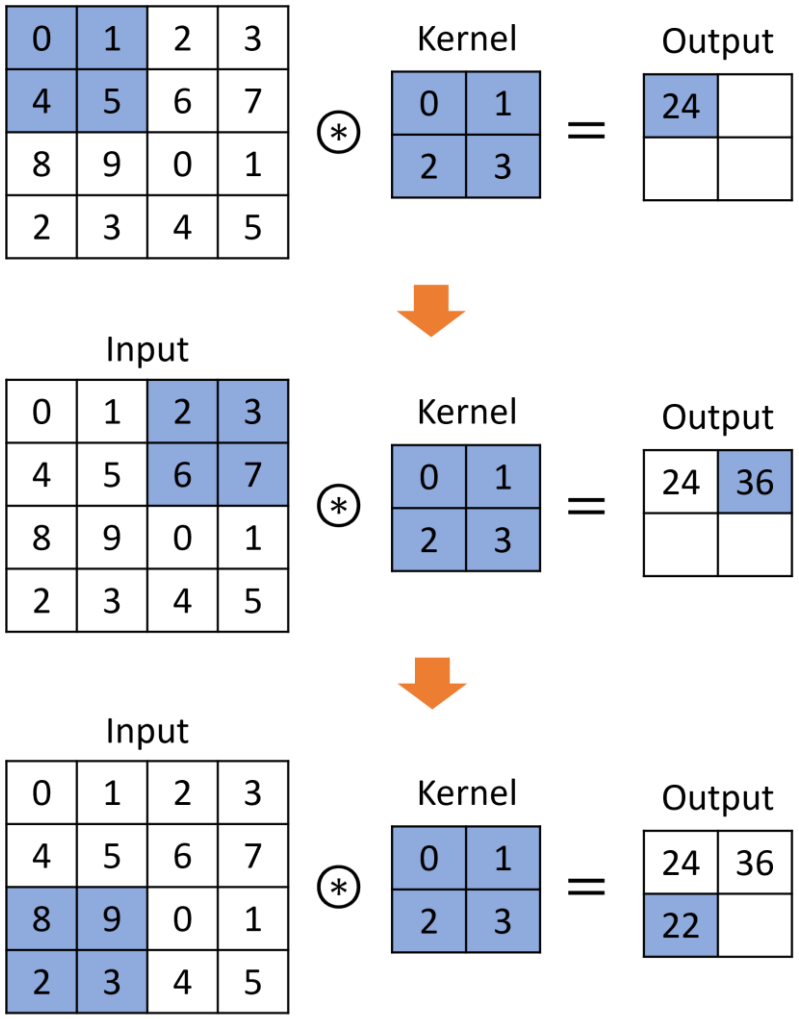
Mapa de características



$$\text{output}(i,j) = \sum_m \sum_n I(i+m, j+n)\, K(m,n)$$

- Lembrando que o objetivo dos *kernels* é extrair características.
- Portanto, o resultado da operação de convolução é chamado de ***mapa de características***.
- Pois ele pode ser considerado como as representações (i.e., características) aprendidas nas dimensões espaciais (por exemplo, largura e altura) para a camada subsequente.

Stride



- Neste exemplo anterior, deslizamos a janela um elemento por vez.
- Porém, às vezes, seja por eficiência computacional ou porque desejamos reduzir a resolução, movemos a janela mais de um elemento por vez.
- Esse parâmetro é chamado de *stride*.
- No exemplo ao lado, o *stride* é de 2 para deslizamentos ao longo das colunas e linhas.
  - Porém, ele pode ser diferente para deslocamentos ao longo das linhas e colunas.

Convolução ou correlação cruzada?

$$\text{output}(i,j) = K(i) \circledast I(j)$$
$$= \sum_m \sum_n K(i-m, j-n) I(m,n)$$

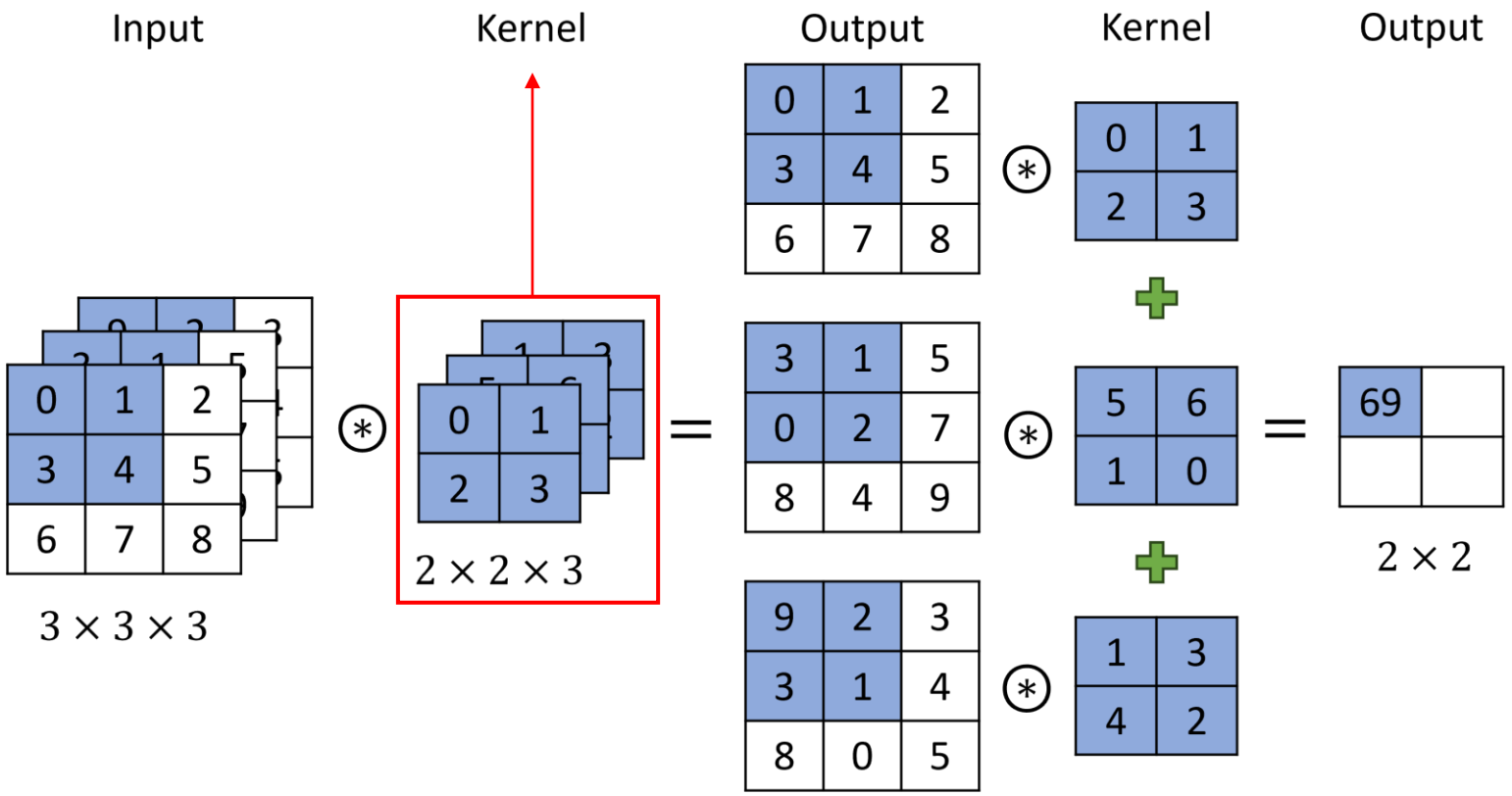
Convolução

$$\text{output}(i,j)$$
$$= \sum_m \sum_n K(i+m, j+n) I(m,n)$$

Correlação cruzada

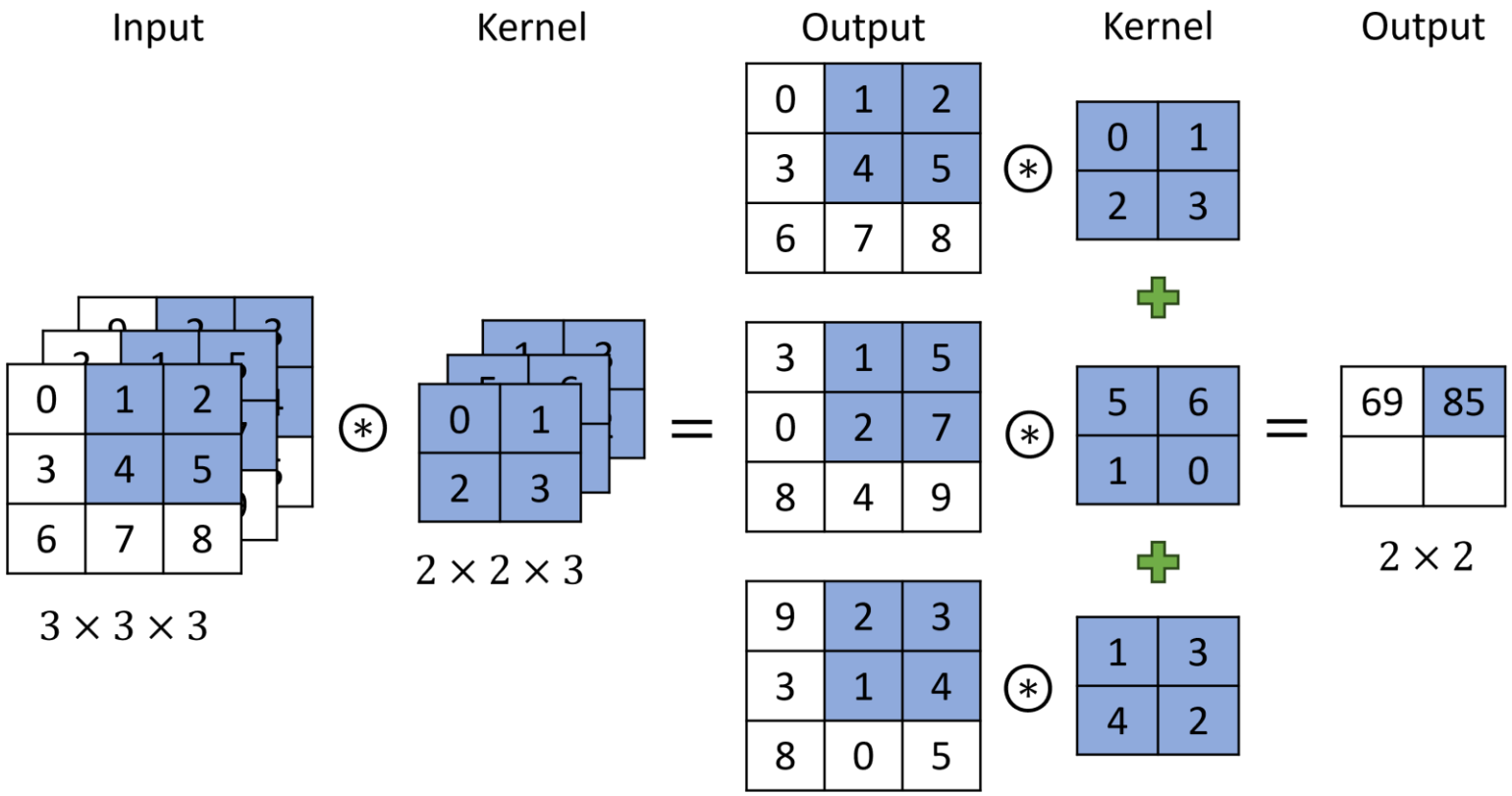
- As operações em uma CNN, embora sejam chamadas de convoluções, são implementadas como correlações cruzadas na maioria das bibliotecas.
  - Correlações são mais eficientes (sem inversão) e simples de serem implementadas.
- Ao contrário da operação de convolução, as CNNs não invertem o ***kernel*** (ou o sinal de entrada).
- No entanto, ***isso não importa***, pois os ***kernels*** são ***aprendidos*** e podem se adaptar tanto à correlação cruzada quanto à convolução.

Operação de convolução com 3 canais

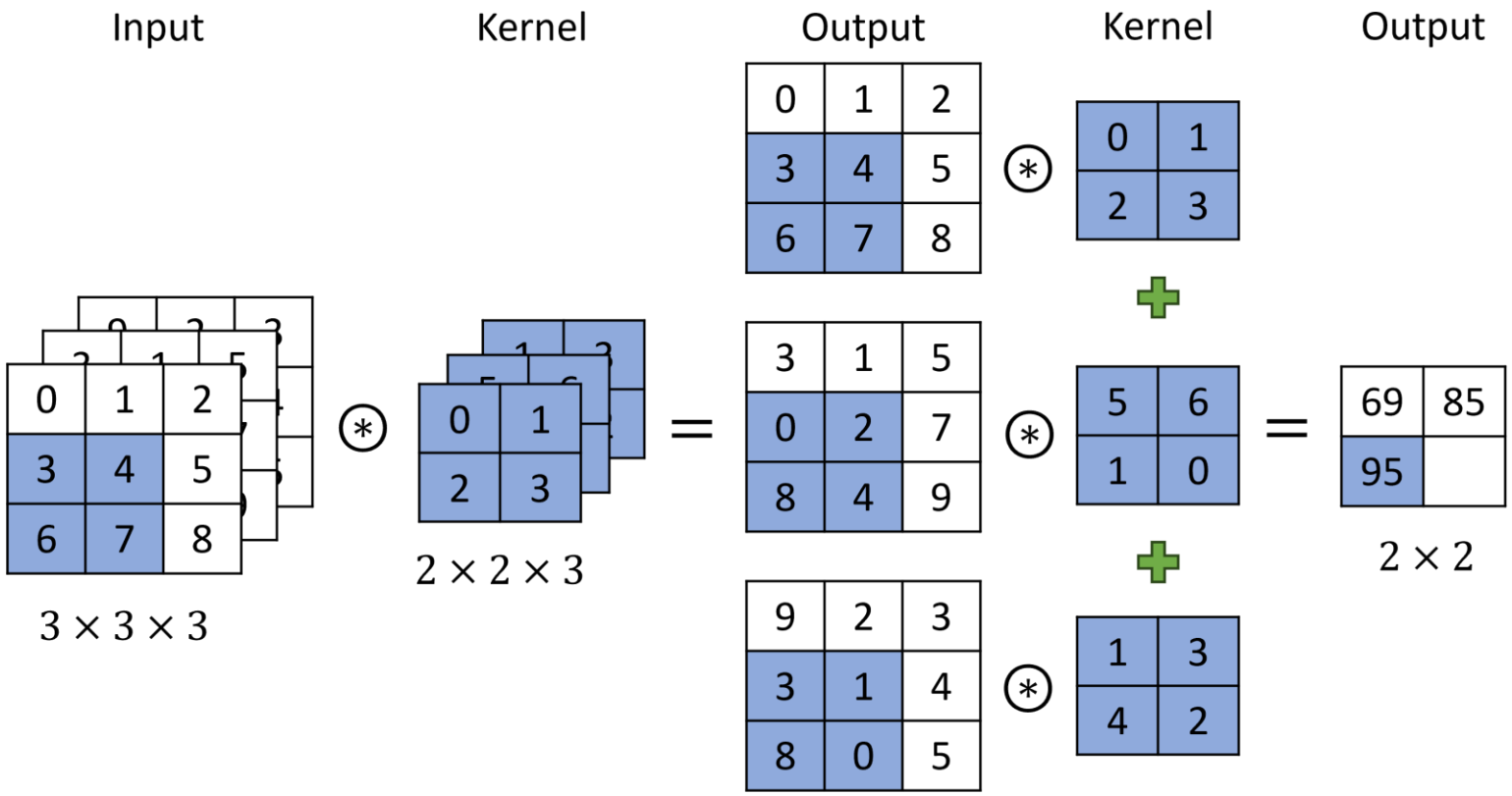


- Em geral, se a imagem tem 3 dimensões, o ***kernel*** também terá 3 dimensões.
- Para entender a operação, podemos dividi-la em 3 operações de convolução separadas que têm seus resultados somados ao final para gerar a saída.
- Usando um *stride* = 1, temos.

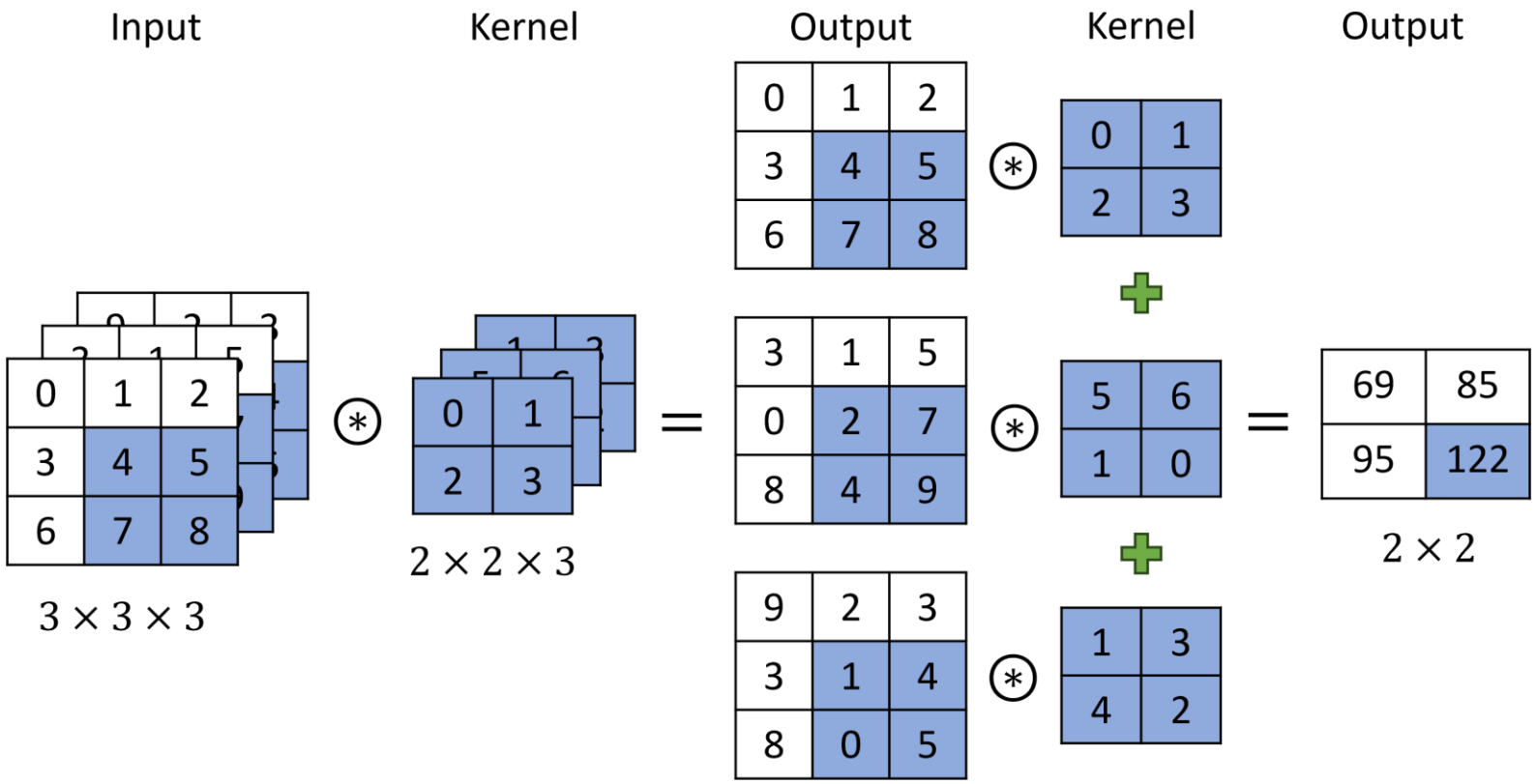
Operação de convolução com 3 canais



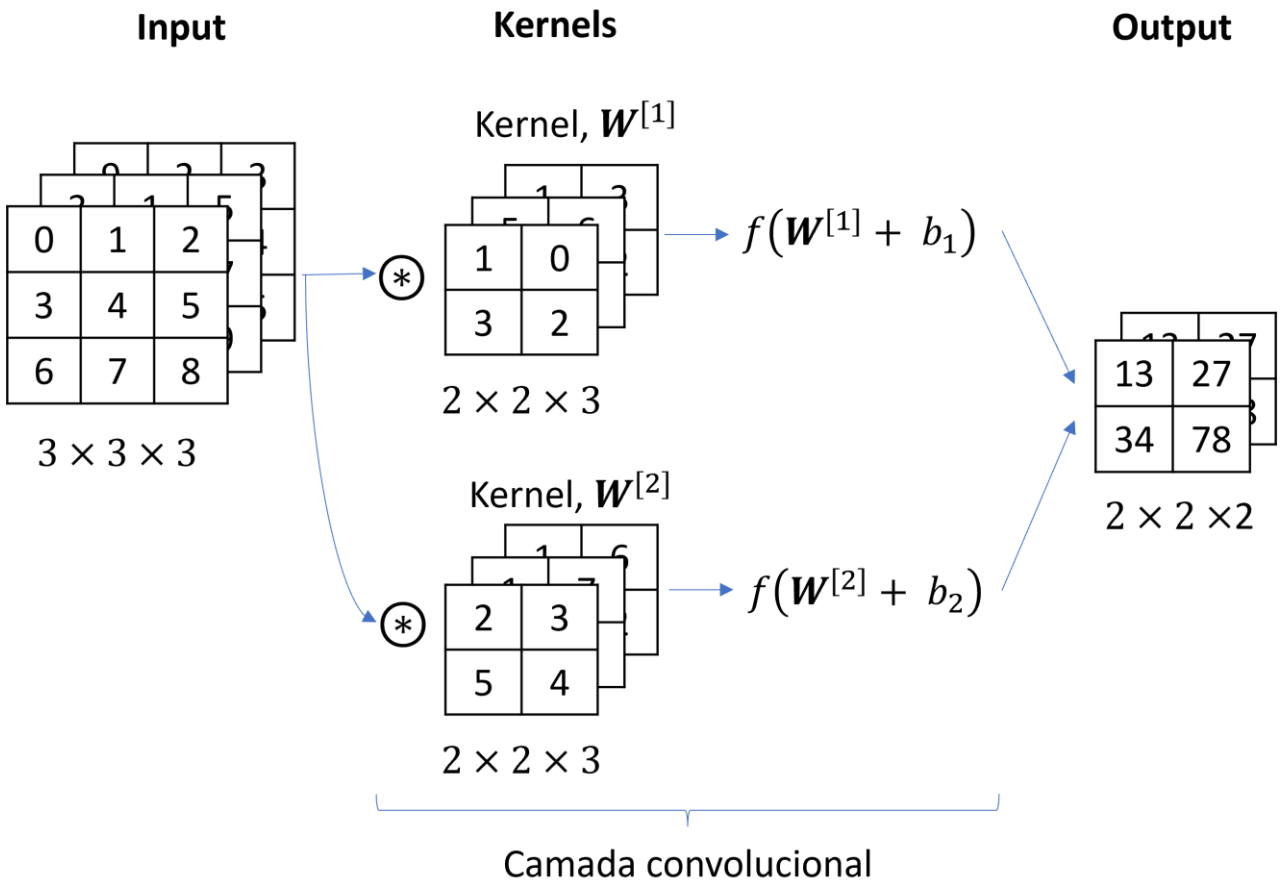
Operação de convolução com 3 canais



Operação de convolução com 3 canais

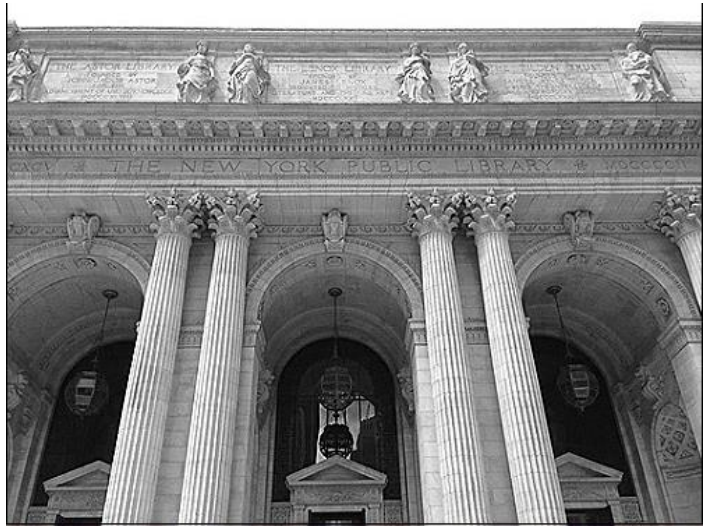


Kernels diferentes para características diferentes



- Em geral, cada camada de uma rede convolucional possui vários *kernels*.
- Cada *kernel* detecta uma característica diferente.
- A saída de cada *kernel* tem um valor de bias somado a ela e o resultado é passado por uma função de ativação, $f(.)$, (e.g., ReLU).
- A saída de uma camada é o resultado do empilhamento de várias matrizes.

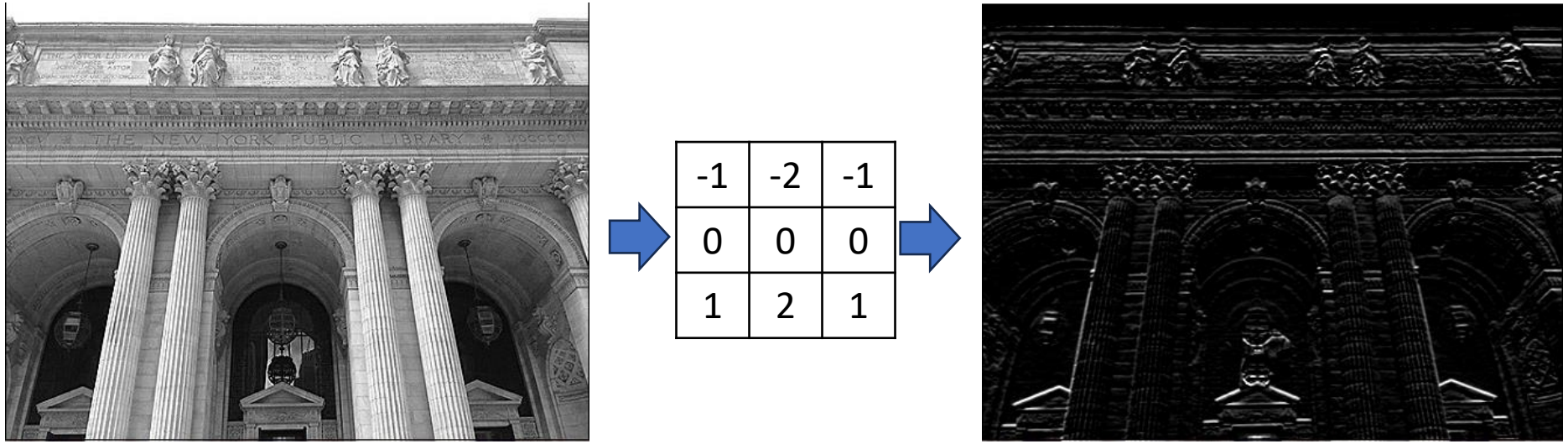
Aplicando *kernels* a imagens



-1	0	1
-2	0	2
-1	0	1

- Considerem a imagem à esquerda.
- Se aplicarmos o *kernel* (i.e.,filtro) mostrado, obteremos os resultados à direita.
- Ele realça muito as linhas verticais e escurece todo o resto.
- Portanto, podemos considerar este *kernel* como um ***detector de linhas verticais***.

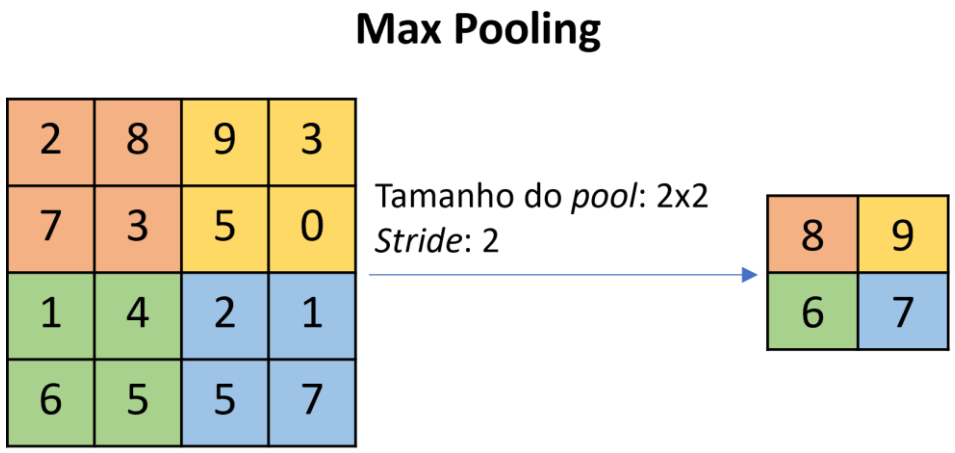
Aplicando *kernels* a imagens



- De forma similar, este filtro pode detectar linhas horizontais, escurecendo quase tudo na imagem que não seja uma linha horizontal.
- Ao aplicar filtros como esses, podemos remover quase tudo, exceto uma característica distinta.
- Esse processo é chamado de ***extração de características***.
  - Processo que determina as partes mais importantes de uma imagem.

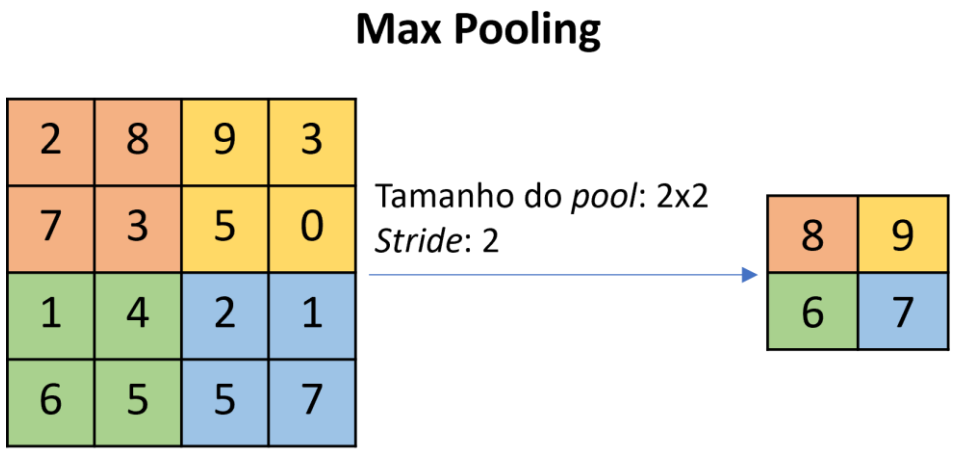
fonte: https://setosa.io/ev/image-kernels/

Camada de *Pooling* (ou subamostragem)



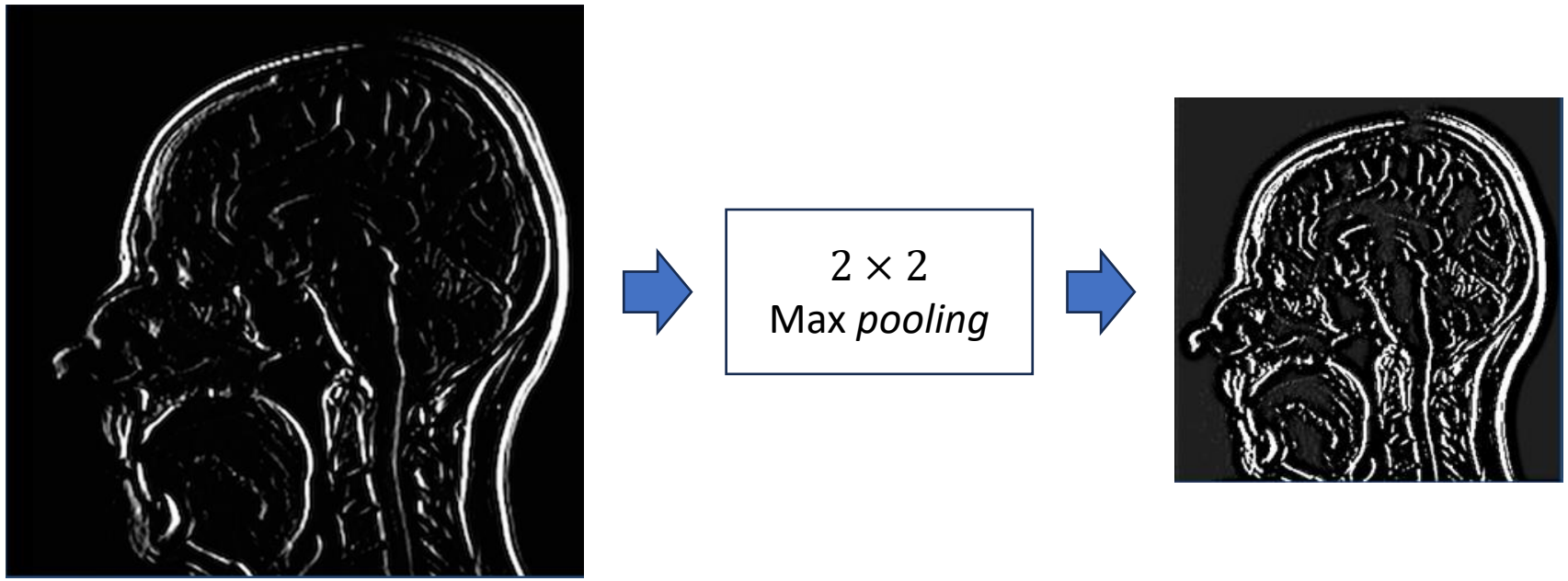
- Aplicada, em geral, após uma camada de convolução.
- Ela subamostra sua entrada.
- O objetivo da subamostragem é reduzir a carga computacional, o uso de memória e o número de parâmetros (limitando assim o risco de sobreajuste).
- Além disso, ela ajuda a tornar a rede mais robusta a pequenas mudanças na posição das características, o que é útil em tarefas de reconhecimento de objetos.

Camada de *Pooling* (ou subamostragem)



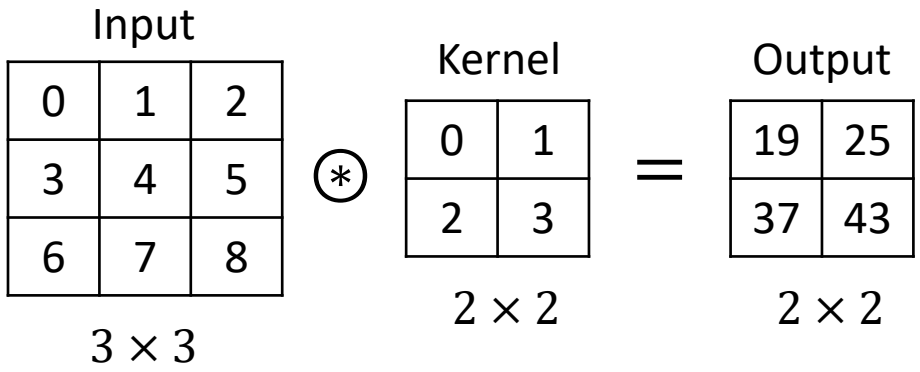
- A maneira mais comum de subamostrar é aplicar uma operação $\max(.)$ ao resultado de cada *kernel*.
- Somente o valor máximo de entrada em cada campo receptivo da camada de *pooling* passa para a próxima camada, enquanto as outras entradas são descartadas.
- O *pooling* é normalmente aplicado a cada canal de entrada de forma independente, de forma que a profundidade de saída seja igual a de entrada.

Camada de *Pooling* (ou subamostragem)



- Adicionalmente, as camadas de *pooling* ajudam a capturar e reter as características mais importantes, ao mesmo tempo que descartam informações menos relevantes ou ruidosas.
- Portanto, elas compactam os dados sem perder as características importantes.

Padding ou preenchimento

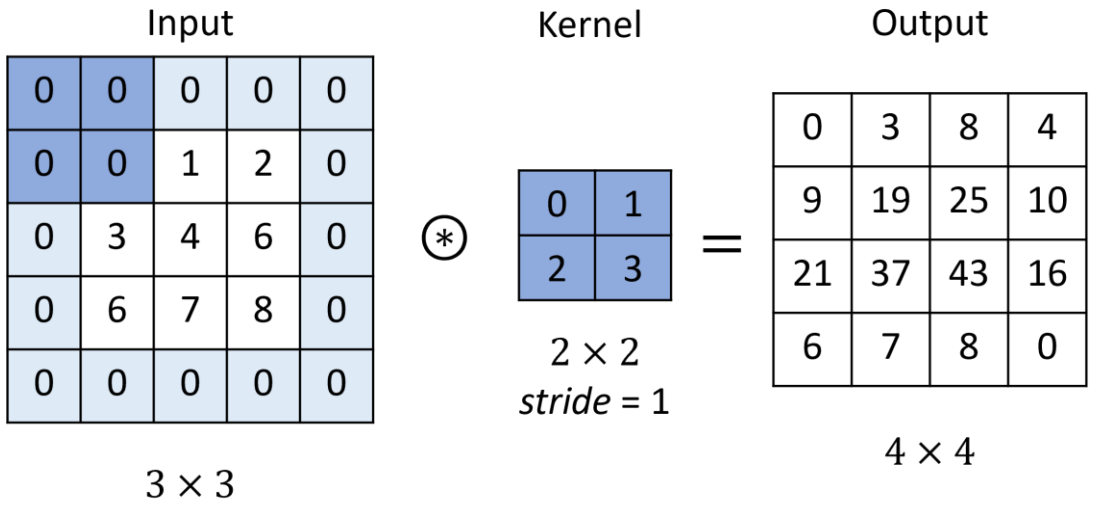


$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

n_{out} : dimensão da saída
n_{in} : dimensão da entrada
k : dimensão do kernel
p : quantidade camadas de *padding*
s : tamanho do *stride*
$\lfloor x \rfloor$: função piso retorna o maior inteiro menor ou igual a x.

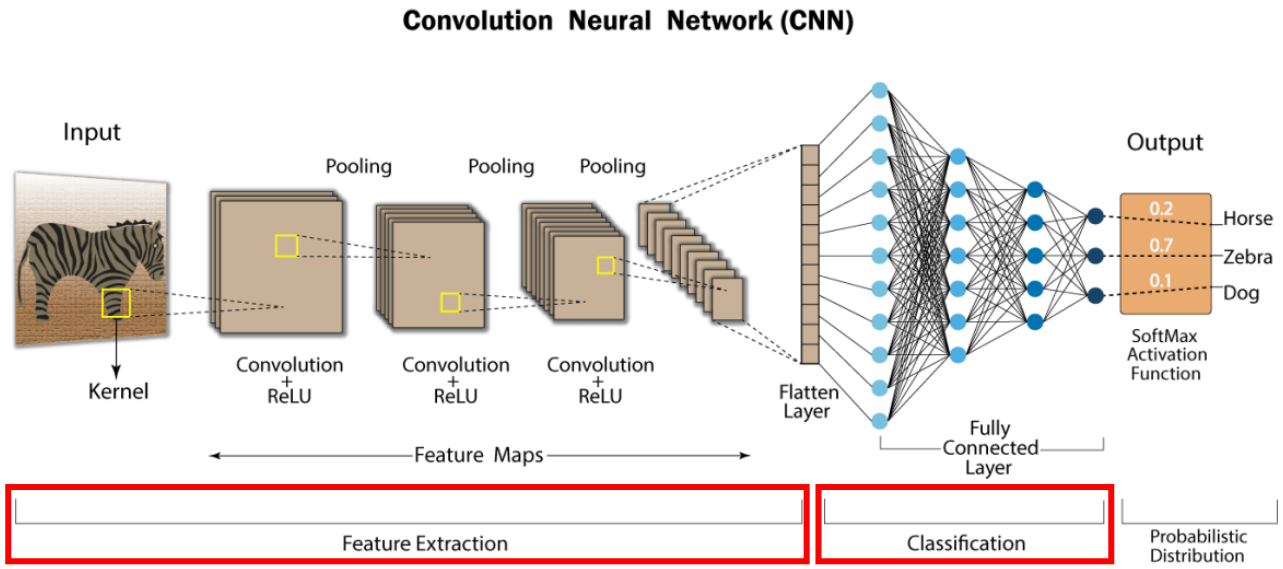
- Depois de aplicar muitas convoluções sucessivas, as imagens tendem a se tornarem consideravelmente menores do que as da entrada.
- Se temos uma imagem de entrada com 240×240 *pixels*, após dez camadas de convolução 5×5 com *stride* igual a 1, ela é reduzida para 200×200 *pixels*.
- Isso reduz a imagem em ≈17%, e consequentemente, faz com que qualquer informação interessante nas bordas da imagem desapareçam.

Padding ou preenchimento



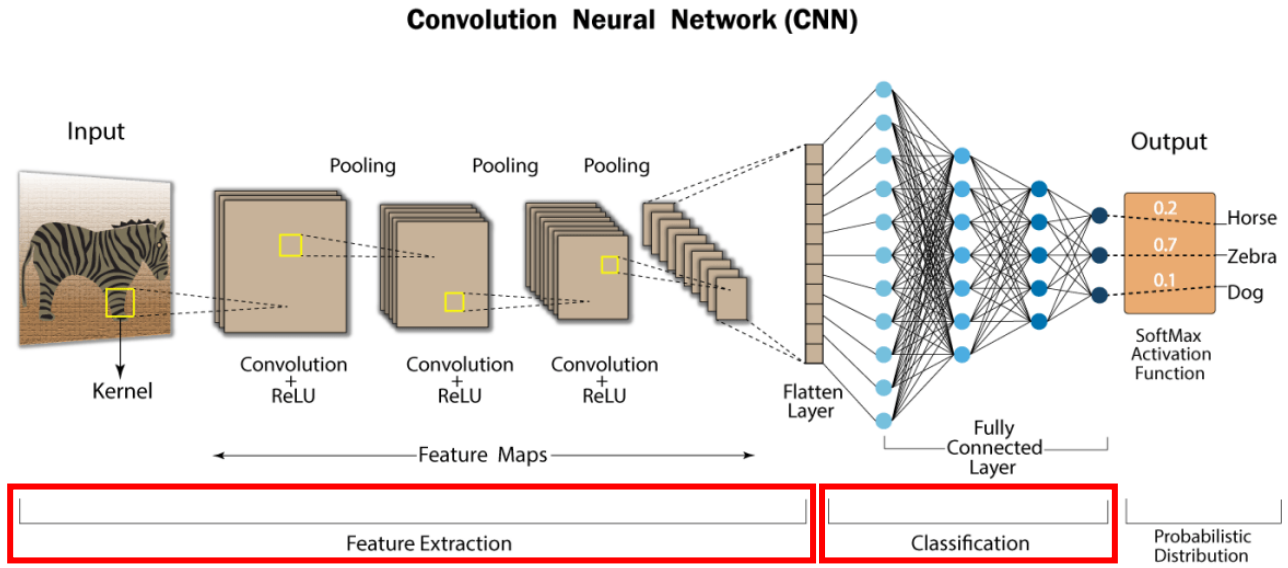
- O *padding* é usado para controlar o tamanho dos mapas de características após uma camada convolucional.
- *Pixels* de preenchimento são adicionados ao redor da borda da imagem de entrada, aumentando assim seu tamanho efetivo.
  - Normalmente, os pixels são feitos iguais a zero.
- Em tarefas de classificação de imagens, é comum aplicar *padding* nas camadas iniciais para preservar informações de borda, enquanto camadas finais não o aplicam para reduzir a dimensionalidade.

Camadas densas



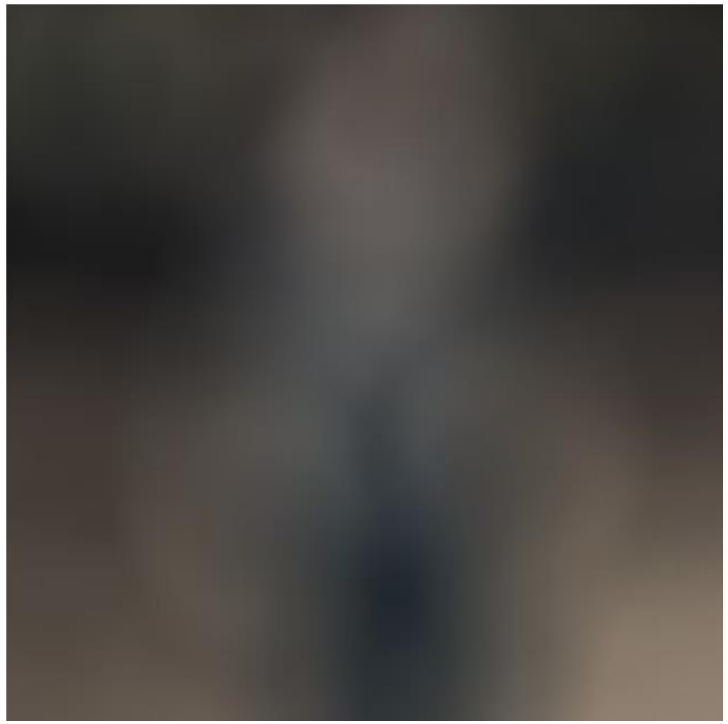
- Depois das camadas convolucionais, as CNN apresentam algumas camadas densas.
- Elas são responsáveis por realizar a classificação ou regressão propriamente dita.
- As camadas densas recebem as características extraídas pelas camadas convolucionais e combinam essas informações para aprender padrões complexos e realizar a tarefa de classificação ou regressão.

Os filtros são aprendidos!



- Nós não precisamos definir os filtros manualmente.
- Em vez disso, durante o ***treinamento***, as ***camadas convolucionais aprenderão automaticamente*** os filtros mais úteis para sua tarefa, e as camadas acima (i.e., densas) aprenderão a combiná-los em padrões mais complexos.

Exemplo de classificação de imagens



- Para entender melhor o que uma camada de convolução faz, vamos considerar um exemplo simples.
- Vamos supor que não conhecemos o conteúdo da imagem ao lado.
- Isso foi simulado deixando-a embaçada.

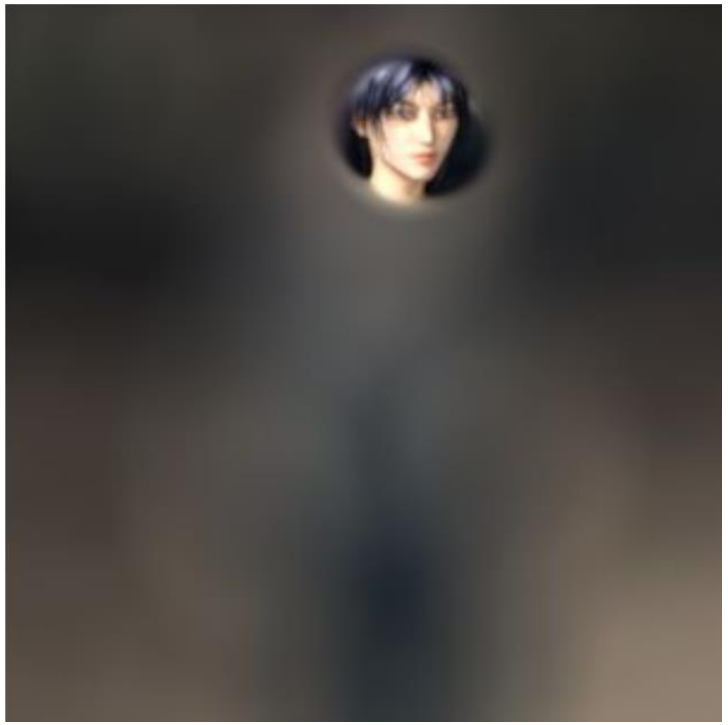
Exemplo de classificação de imagens

- Então, digamos que haja um filtro ou um conjunto de filtros que ao serem passados sobre a imagem extraem os pixels mostrados ao lado.
- Como podemos ver, são duas formas verticais, que se parecem com pernas humanas.

Exemplo de classificação de imagens

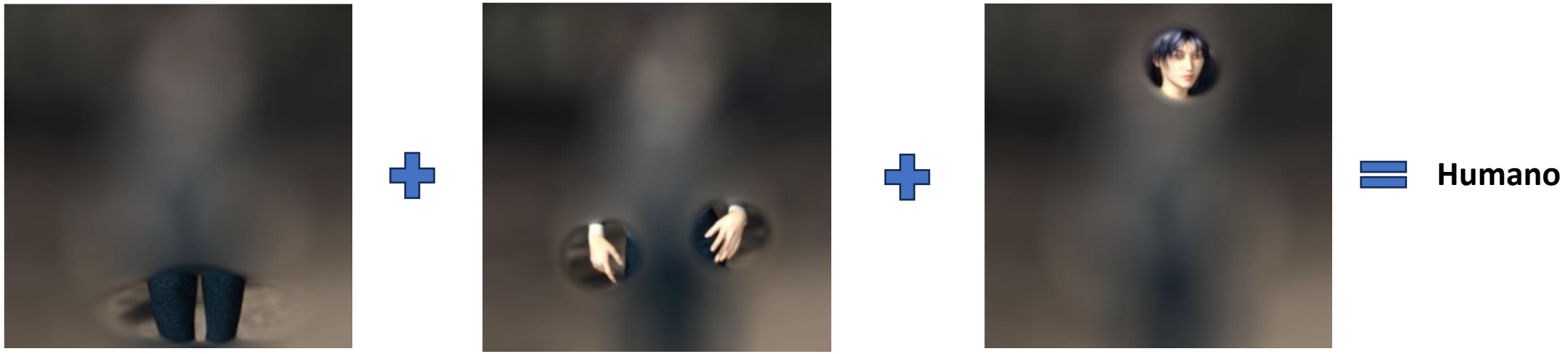
- Na sequência, outros filtros extraem as características mostradas ao lado.
- São formas com cilindros que se projetam a partir delas.
- Nosso cérebro vê isso e instantaneamente classifica como mãos.
- Porém uma rede neural não treinada ainda não sabe disso.
- Ela apenas sabe que um filtro pode extrair algo parecido com esse conjunto de pixels.

Exemplo de classificação de imagens



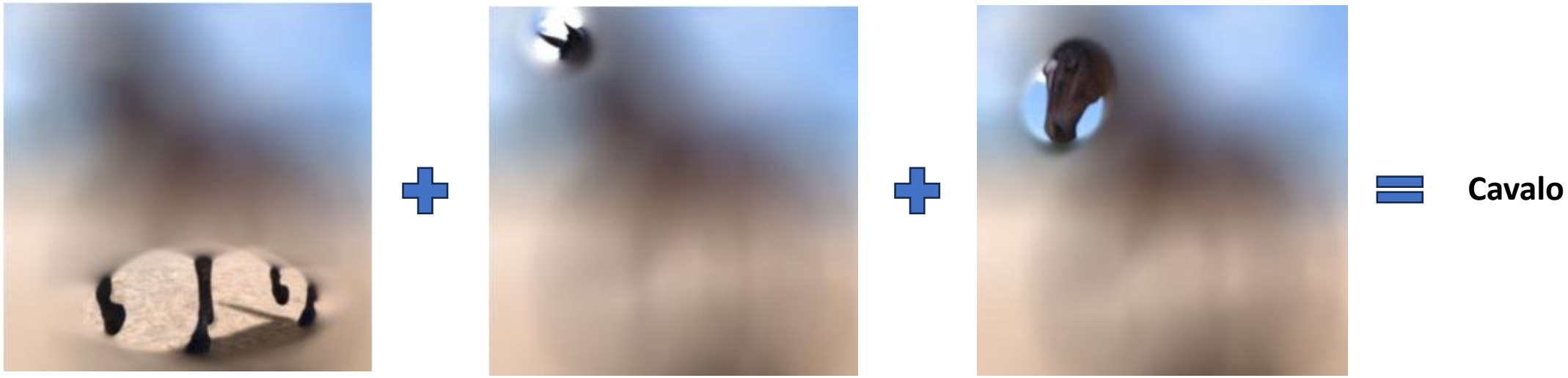
- Em seguida, um outros filtros extraem um círculo com outros dois círculos menores, uma saliência e algumas formas paralelas.
- Nosso cérebro reconhece esse conjunto de *pixels* como um rosto com olhos, nariz e boca.
- Mas, novamente, o modelo não tem contexto para decidir o que esses *pixels* são.
- Ele só sabe que um filtro específico pode extrair essas características.

Exemplo de classificação de imagens



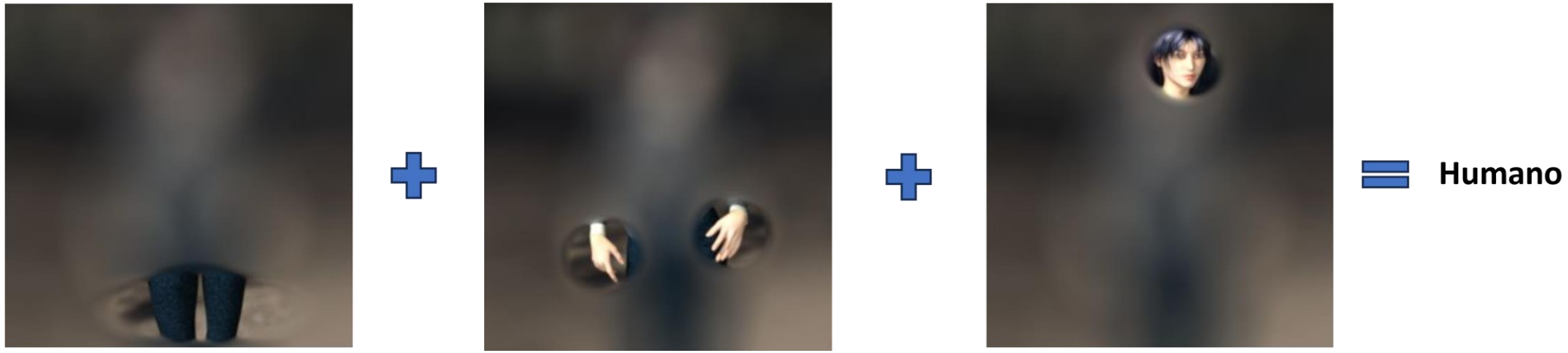
- Quando essas características estão presentes em uma imagem rotulada como humana, a rede neural pode ser treinada para aprender filtros que extraem essas informações.
- A rede pode combinar as características extraídas pelos três filtros (i.e., pernas, mãos e face) para detectar seres humanos em imagens inéditas.

Exemplo de classificação de imagens



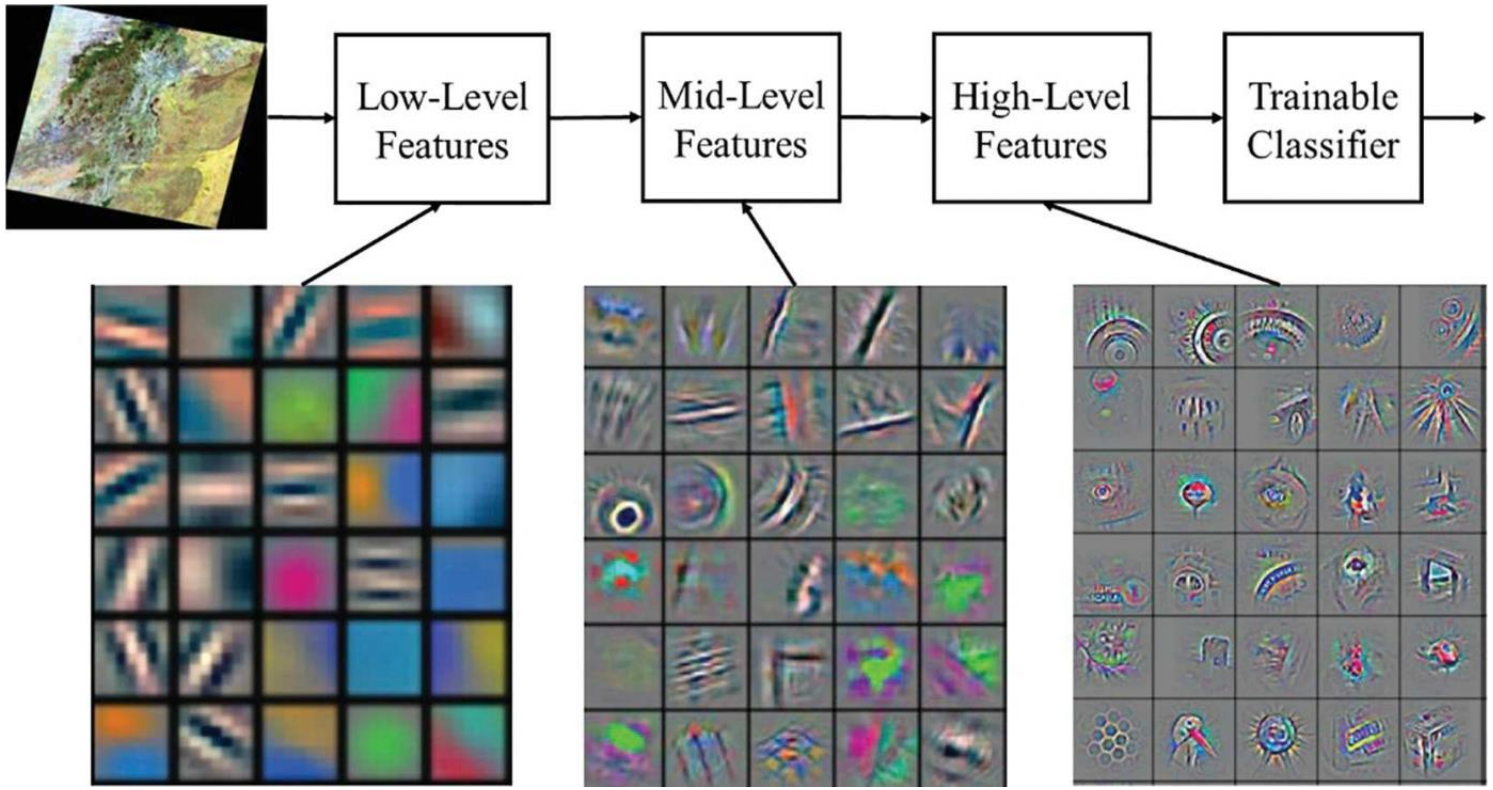
- A mesma rede pode ser treinada para identificar diferentes características presentes nas imagens de cavalos.
- Assim, ao aprender conjuntos de filtros que podem detectar humanos ou cavalos, temos um modelo de visão computacional que pode lidar com imagens complexas e predizer o que há nelas.

Observação



- Notem que usamos pernas, mãos e rostos como exemplos de características extraídas por um filtro.
- Essas são formas reconhecíveis pelo nosso cérebro, então as utilizamos para ilustrar o conceito.

Observação



- No entanto, quando filtros são treinados para detectar características em imagens, eles podem identificar coisas que são imperceptíveis para os seres humanos.
- Podem existir padrões de pixels que correspondam a uma imagem rotulada, mas que não tenham significado aparente para nós.

Explorando CNNs

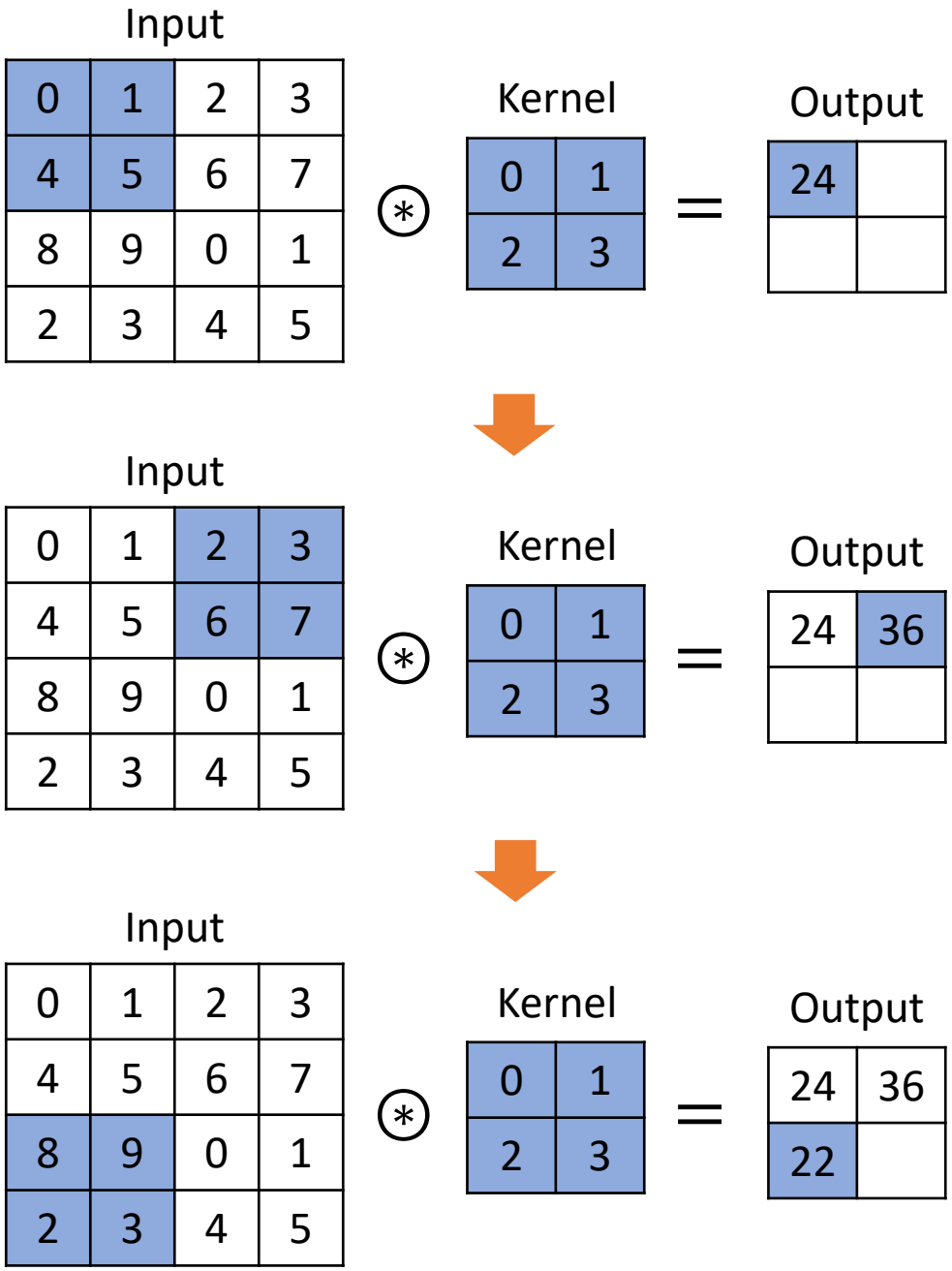
- CNN Explainer
  - https://poloclub.github.io/cnn-explainer/
- ConvNetJS MNIST demo
  - https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html
- ConvNetJS CIFAR-10 demo
  - https://cs.stanford.edu/people/karpathy/convnetj

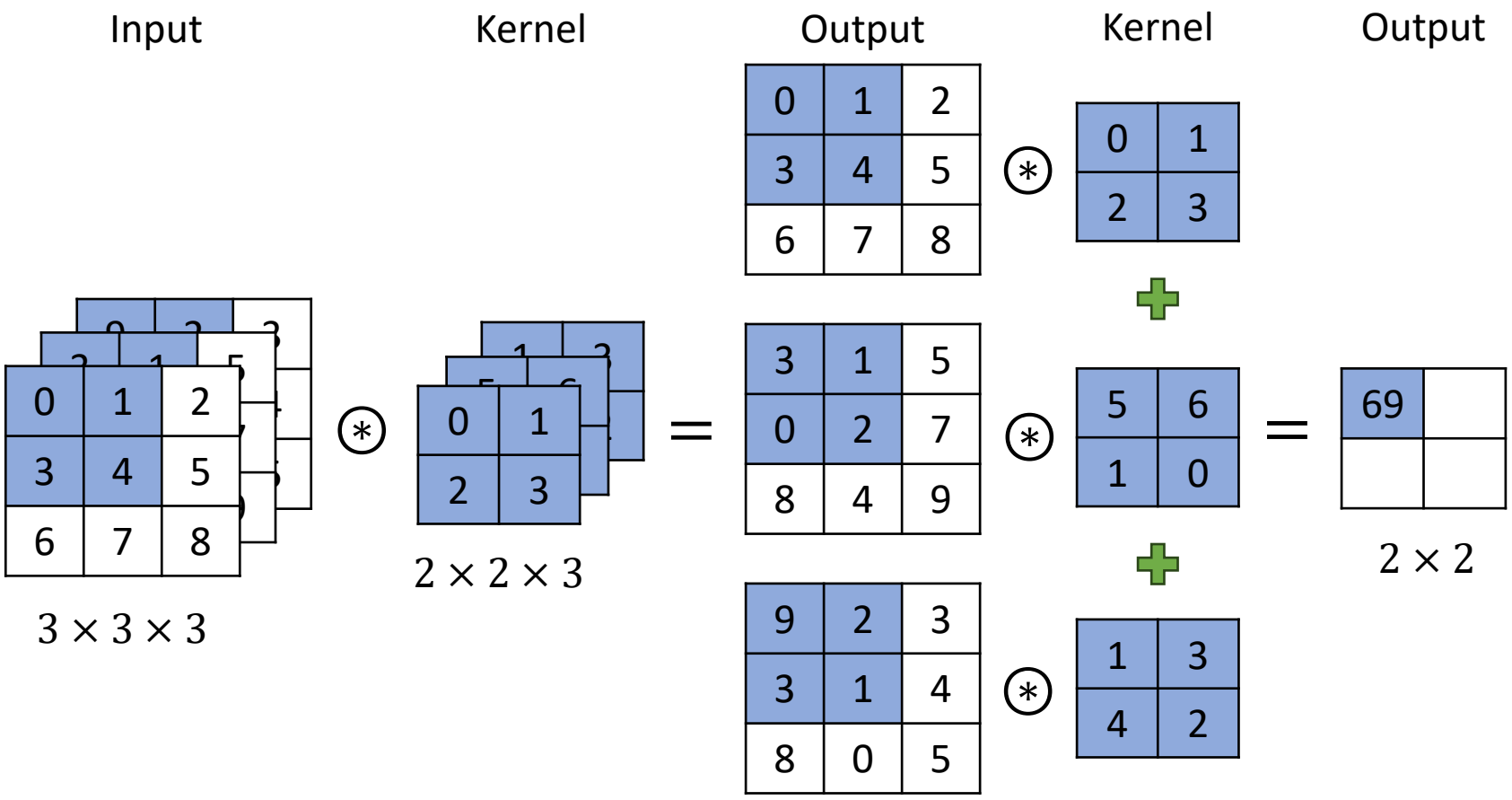
Atividades

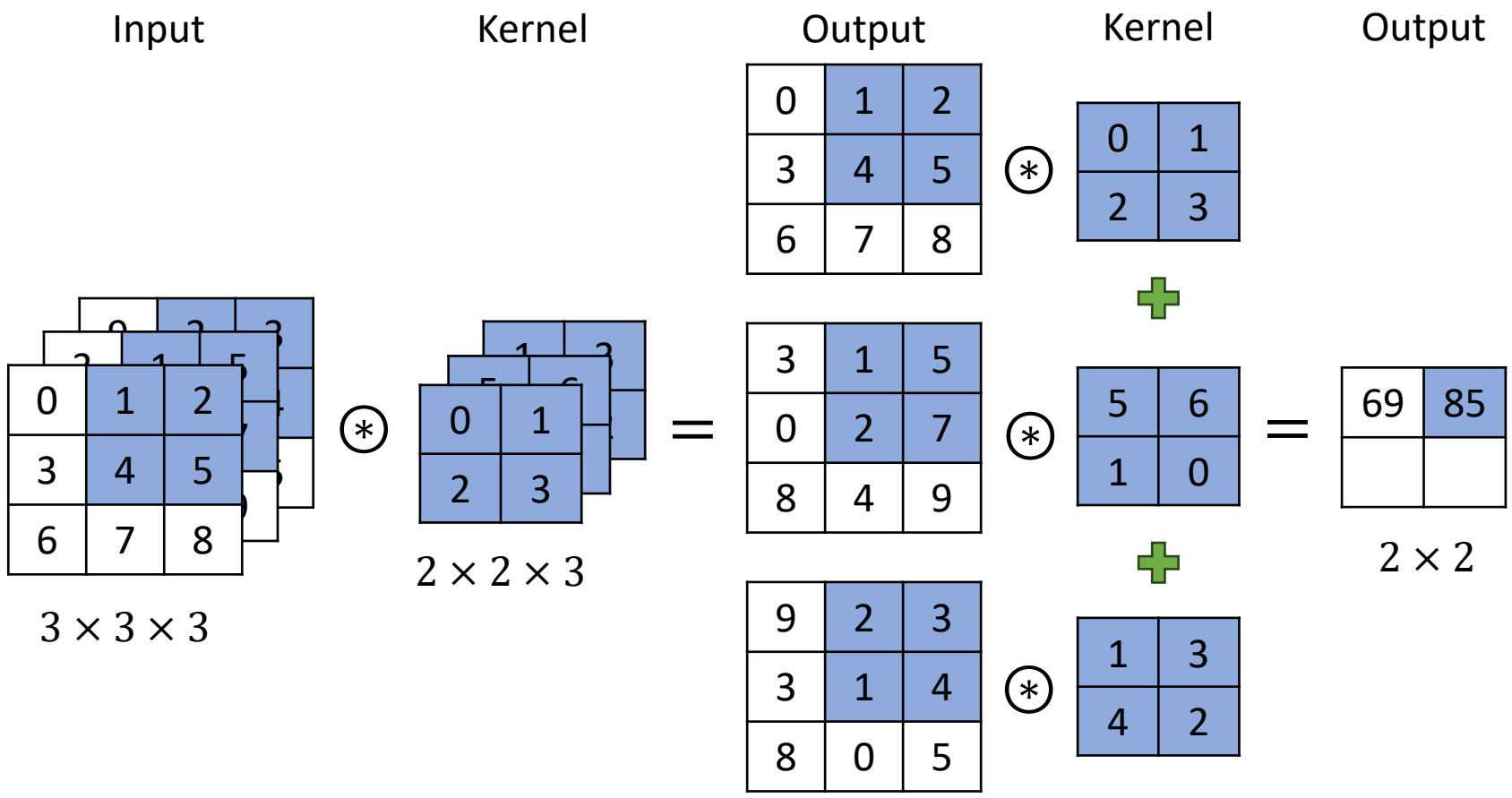
- Quiz: “***TP557 – Introduzindo Convoluções***”.
- Exercício: Redes neurais convolucionais

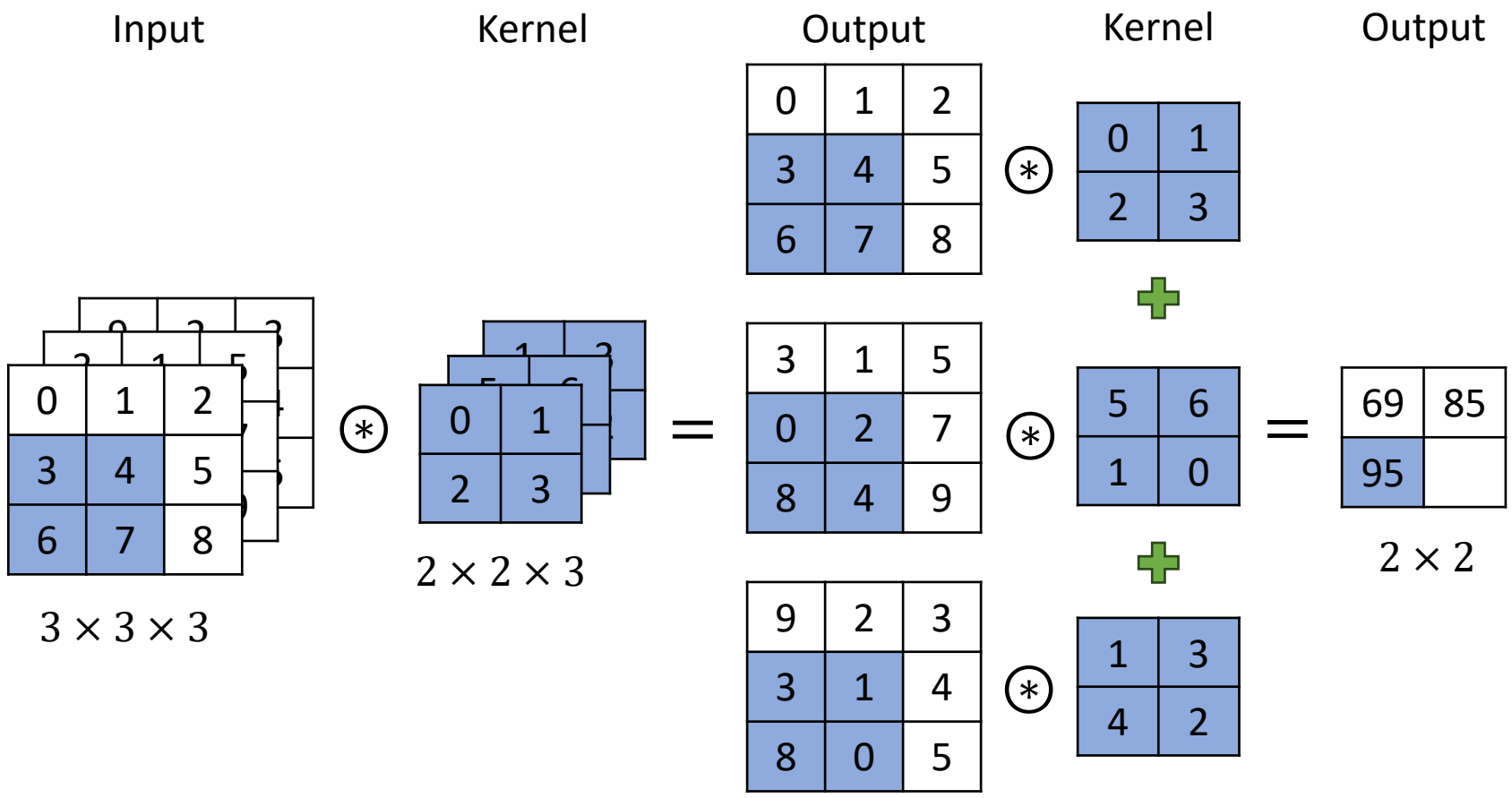
Perguntas?

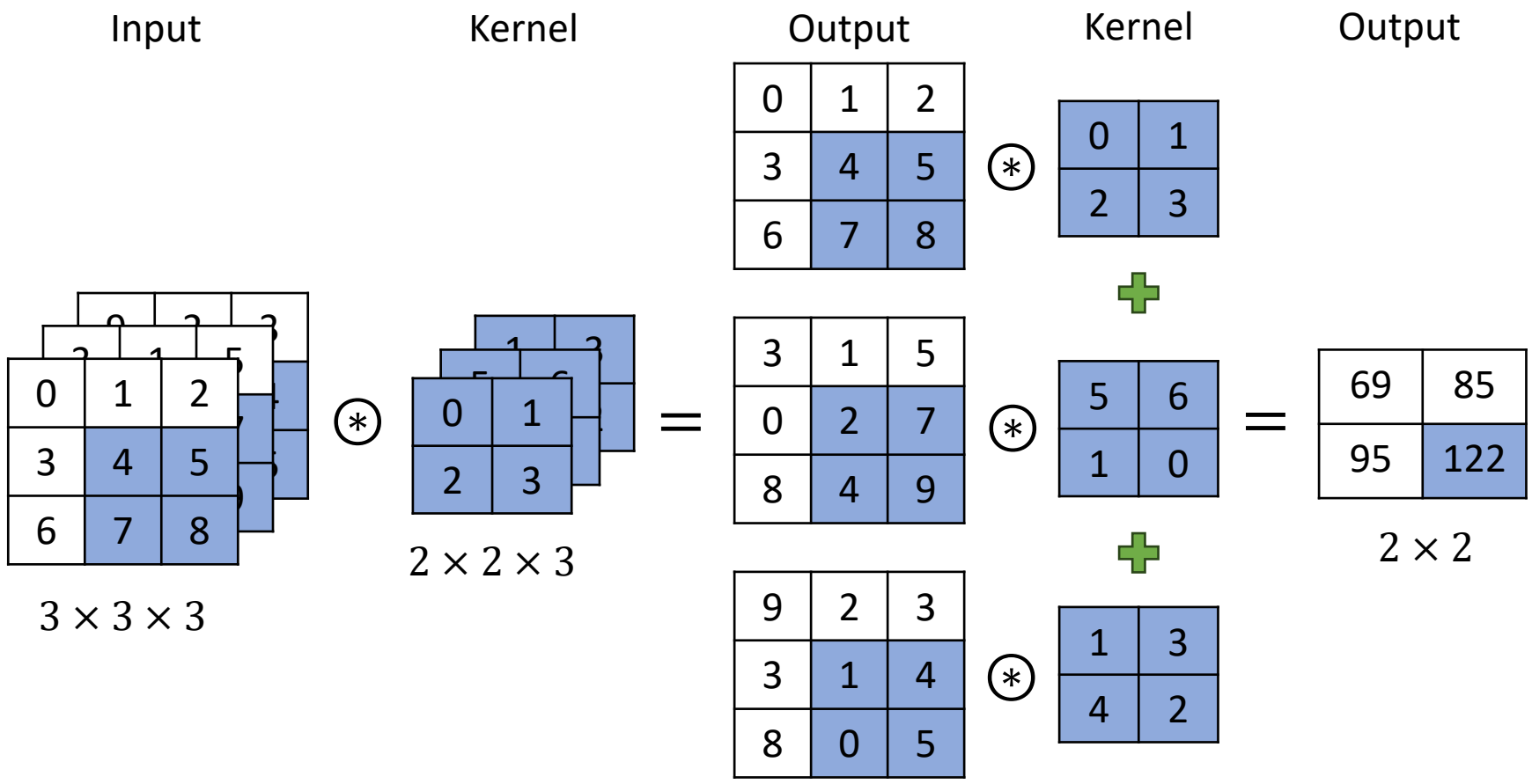
Obrigado!

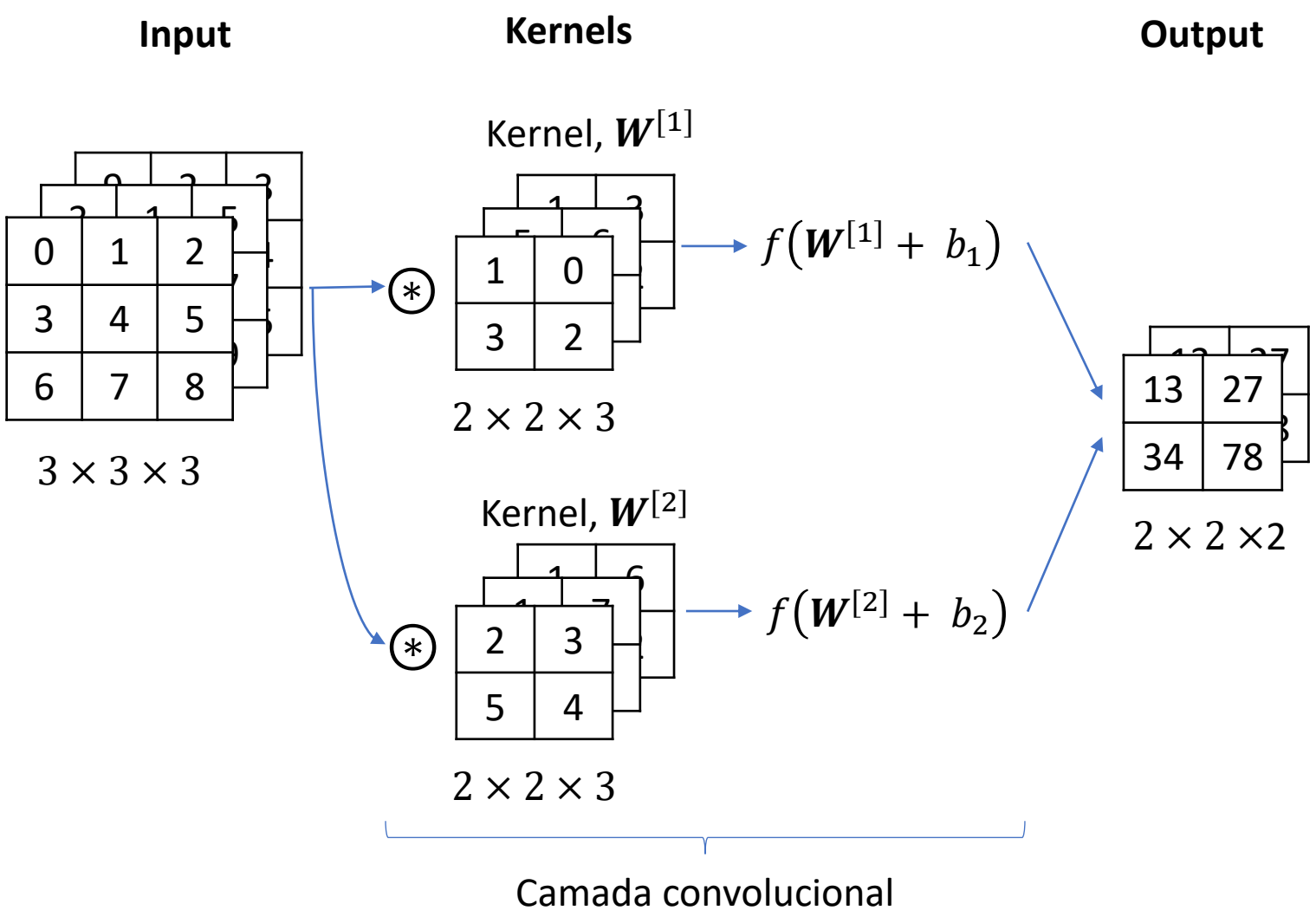

Input
0 1 2 3
4 5 6 7
8 9 0 1
2 3 4 5
⊛
Kernel
0 1
2 3
=
Output
24
Input
0 1 2 3
4 5 6 7
8 9 0 1
2 3 4 5
⊛
Kernel
0 1
2 3
=
Output
24 36
Input
0 1 2 3
4 5 6 7
8 9 0 1
2 3 4 5
⊛
Kernel
0 1
2 3
=
Output
24 36
22


Input
Kernel
Output
Kernel
Output
0 1 2
3 4 5
6 7 8
3 × 3 × 3
0 1
2 3
2 × 2 × 3
3 1 5
0 2 7
8 4 9
9 2 3
3 1 4
8 0 5
0 1
2 3
5 6
1 0
1 3
4 2
69
2 × 2

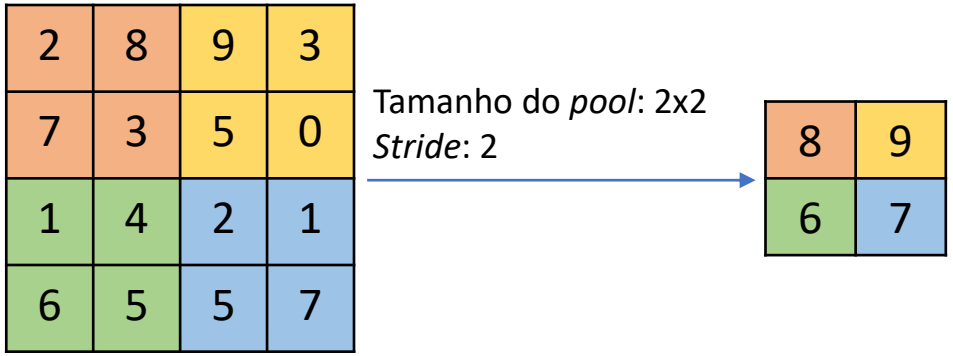

Input
Kernel
Output
Kernel
Output
0 1 2
3 4 5
6 7 8
3 × 3 × 3
0 1
2 3
2 × 2 × 3
3 1 5
0 2 7
8 4 9
9 2 3
3 1 4
8 0 5
0 1
2 3
5 6
1 0
1 3
4 2
69 85
2 × 2

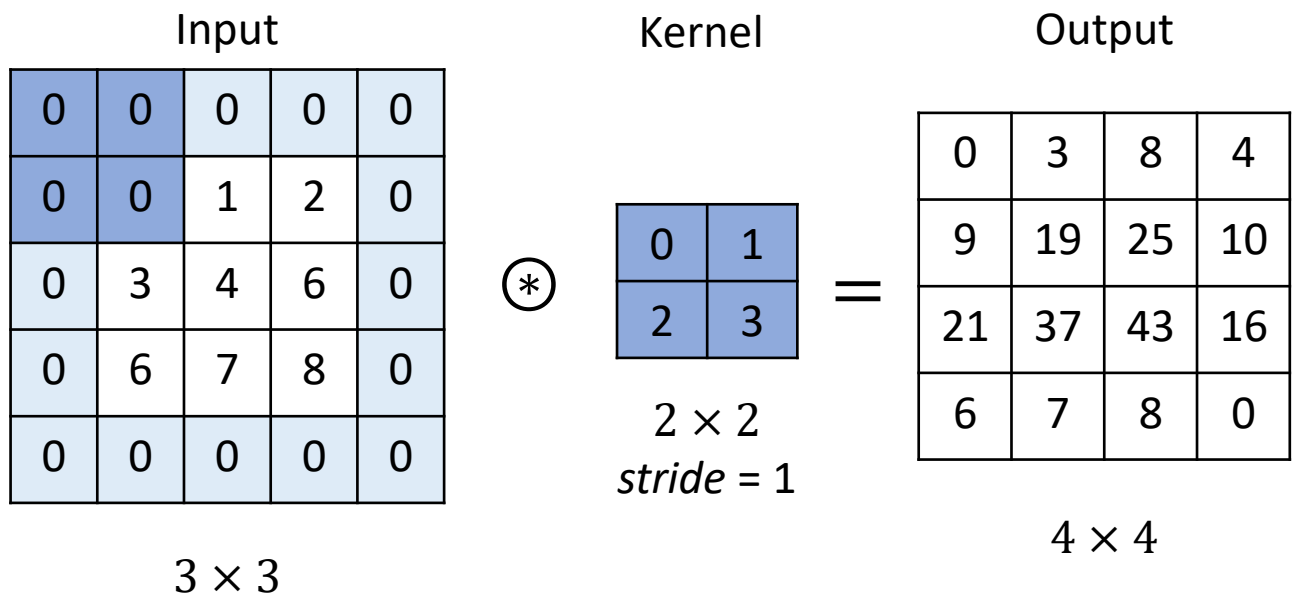

Input
Kernel
Output
Kernel
Output
0 1 2
3 4 5
6 7 8
3 × 3 × 3
0 1
2 3
2 × 2 × 3
3 1 5
0 2 7
8 4 9
9 2 3
3 1 4
8 0 5
0 1
2 3
5 6
1 0
1 3
4 2
69 85
95
2 × 2

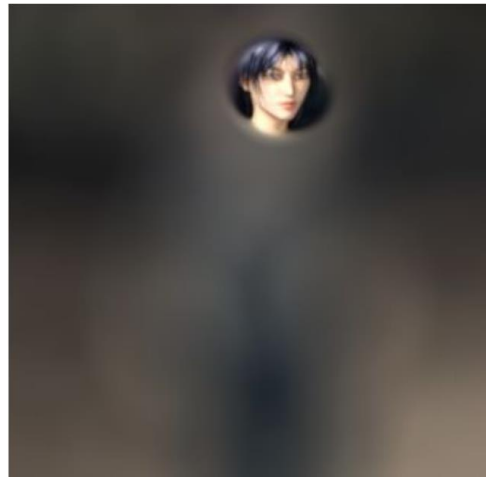

Input
Kernel
Output
Kernel
Output
0 1 2
3 4 5
6 7 8
3 × 3 × 3
0 1
2 3
2 × 2 × 3
3 1 5
0 2 7
8 4 9
9 2 3
3 1 4
8 0 5
0 1
2 3
5 6
1 0
1 3
4 2
69 85
95 122
2 × 2


Input
Kernels
Output
Kernel, $W^{[1]}$
0 1 2
3 4 5
6 7 8
$3 \times 3 \times 3$
1 0
3 2
$2 \times 2 \times 3$
$f(W^{[1]} + b_1)$
13 27
34 78
$2 \times 2 \times 2$
Kernel, $W^{[2]}$
2 3
5 4
$2 \times 2 \times 3$
$f(W^{[2]} + b_2)$
Camada convolucional

Max Pooling


2 8 9 3
7 3 5 0
1 4 2 1
6 5 5 7
Tamanho do *pool*: 2x2
Stride: 2
8 9
6 7


Input
Kernel
Output
0 0 0 0 0
0 0 1 2 0
0 3 4 6 0
0 6 7 8 0
0 0 0 0 0
3 × 3
⊛
0 1
2 3
2 × 2
stride = 1
=
0 3 8 4
9 19 25 10
21 37 43 16
6 7 8 0
4 × 4



Humano