



Uma Introdução às Redes Neurais Convolucionais

**Tópicos Especiais em Matemática Aplicada
Tópicos Especiais em Análise de Dados pra Saúde**

Prof. Dr. Vinicius de Carvalho Rispoli (FCTE/UnB)
16 de setembro de 2025

Sumário

- Parte 1
 - Introdução
- Parte 2
 - Convolução
- Parte 3
 - Um pouco sobre o *overfitting* (sobreajuste)

Parte 1

Por que a visão computacional importa?

Imagen médica: a visão computacional é usada em imagens médicas para auxiliar em tarefas como diagnóstico baseado em imagem e planejamento de tratamento.

Vigilância: A visão computacional é usada em sistemas de vigilância para detectar e rastrear pessoas ou veículos e para reconhecer rostos ou placas de veículos.

Veículos autônomos: a visão computacional é usada em veículos autônomos para permitir tarefas como detecção de objetos, manutenção de faixa e reconhecimento de sinais de trânsito.

...

Por que a visão computacional importa?

Classificação de imagens

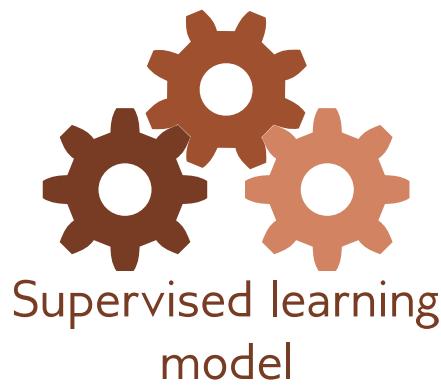
Real world input



Model
input

$$\begin{bmatrix} 124 \\ 140 \\ 156 \\ 128 \\ 142 \\ 157 \\ \vdots \end{bmatrix}$$

Model



Model
output

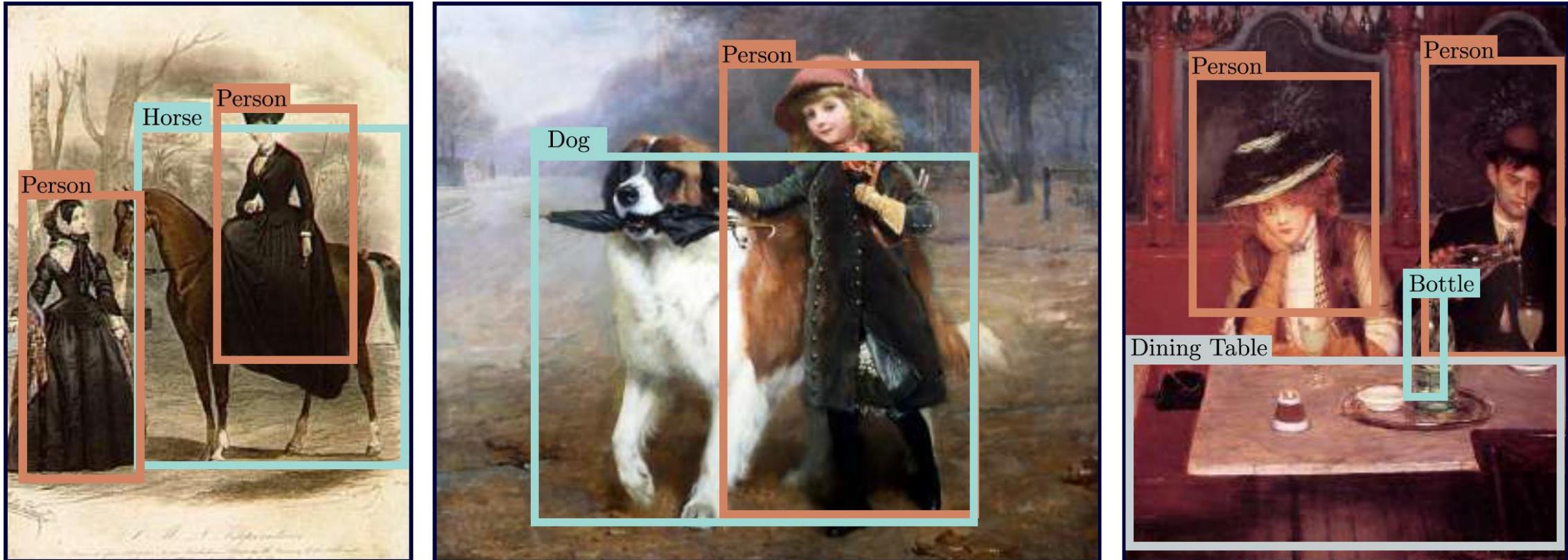
$$\begin{bmatrix} 0.00 \\ 0.00 \\ 0.01 \\ 0.89 \\ 0.05 \\ 0.00 \\ \vdots \\ 0.01 \end{bmatrix}$$

Real world output

Aardvark
Apple
Bee
Bicycle
Bridge
Clown
:

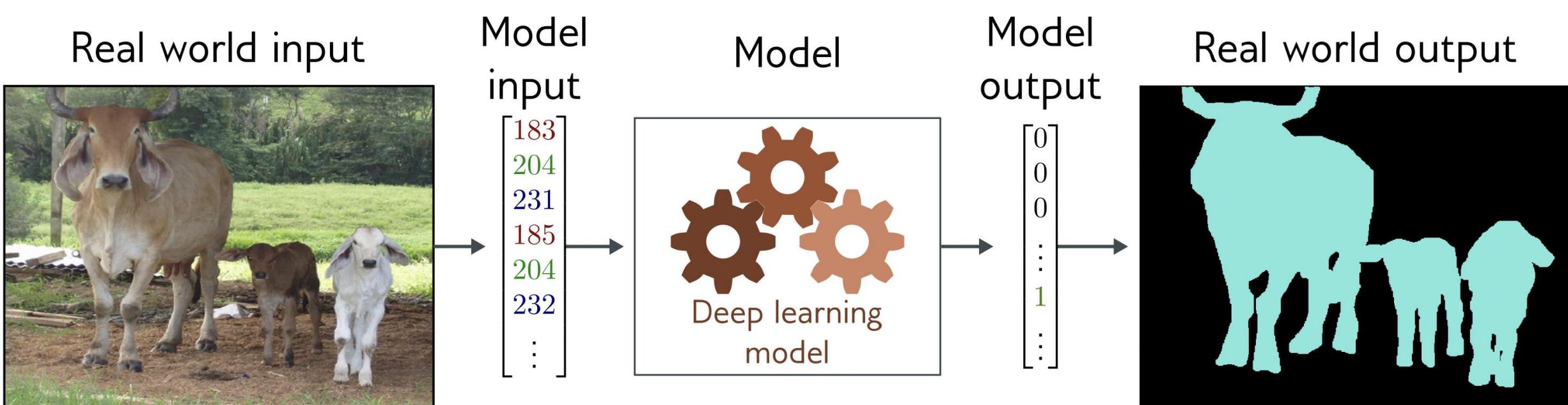
Por que a visão computacional importa?

Detecção de objetos



Por que a visão computacional importa?

Segmentação de imagens



Por que a visão computacional importa?

Pensando no problema do auxílio ao diagnóstico:

- Necessário um bom volume de imagens de cada classe;
- RM, por exemplo, possui usualmente uma resolução de 320x320 ⇒ um vetor com 102.400 entradas ⇒ 10 bilhões de parâmetros para peso.
- Como seria tratar 50000 imagens neste formato?

Pensando agora num problema de classificação de animais:

- Por exemplo, será bom desenvolver uma rede que “perceba” que as figuras abaixo são exatamente a mesma imagem, a menos de uma translação.



Qual a necessidade de outra arquitetura?

As redes *feedforward* funcionariam para os problemas observados anteriormente?

- Sim! Mas ela precisaria de uma grande quantidade de parâmetros e esforço computacional para funcionar adequadamente.

Redes neurais convolucionais são projetadas para lidar com dados estruturados em grade, como imagens, onde há uma forte dependência local entre os itens vizinhos da grade.

Para dados estruturados em grade, redes neurais convolucionais são computacionalmente mais eficientes do que as redes neurais *feedforward*. Isso ocorre principalmente porque as redes neurais convolucionais exigem menos parâmetros do que redes *feedforward*.

Qual a necessidade de outra arquitetura?

Os dados de imagem exibem duas propriedades estruturais principais, a saber, **invariância a translação e localidade**.

- **Invariância a translação:** a decisão de classificação para cada imagem é independente da posição do animal na imagem. Um gato é um gato, independentemente de aparecer no topo ou na parte inferior de uma imagem.
- **Localidade:** a decisão de classificação não depende realmente de um pixel que esteja longe do animal na imagem. Um gato ainda é um gato, independentemente de pixels distantes corresponderem a um edifício ou a uma árvore.
- Lembre-se de que, ao usar redes neurais totalmente conectadas para imagens, o primeiro passo é converter cada imagem em um vetor de recursos de entrada. Ao fazer isso, podemos perder as duas propriedades estruturais mencionadas acima.

O que muda para uma rede convolucional?

Possuem estrutura semelhante à rede *feedforward*

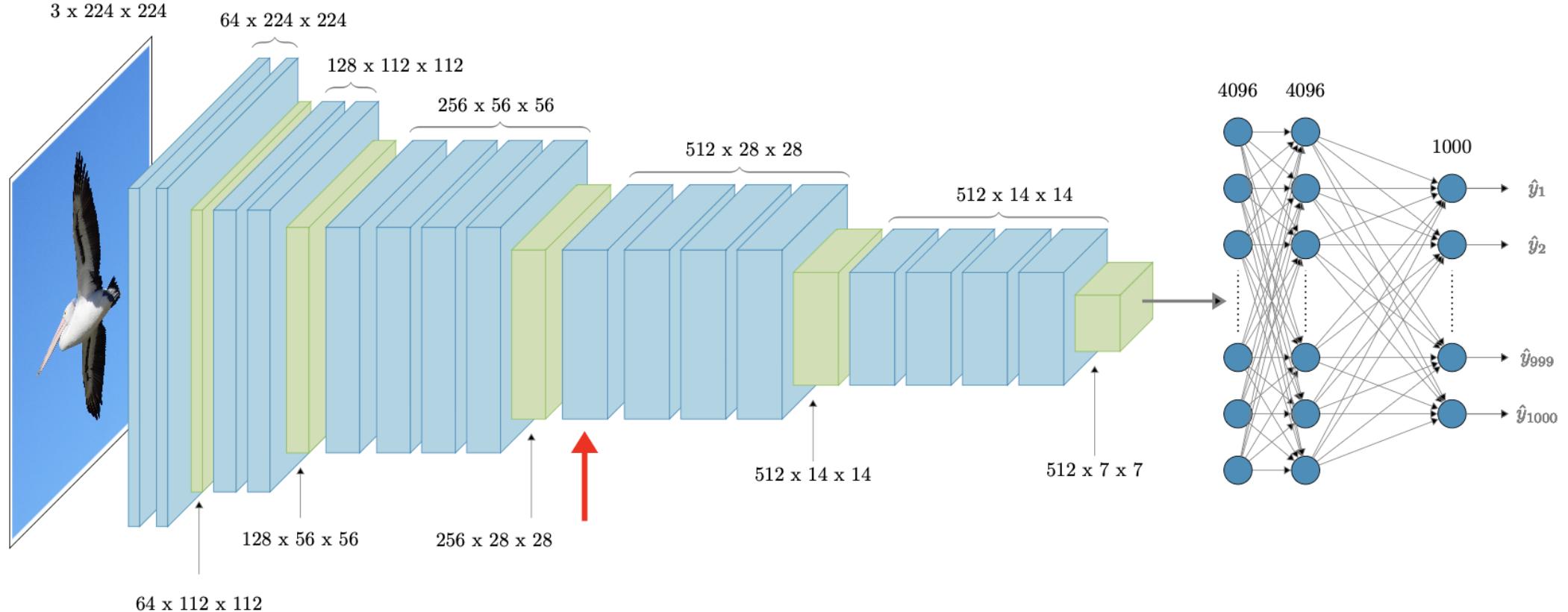
Redes *feedforward* usuais não funcionam muito bem com imagens grandes e nem são capazes compreender adequadamente as propriedades estruturais das imagens.

A CNN, por outro lado, trabalha analisando as imagens pedaço por pedaço.

As CNNs possuem uma estrutura ligeiramente diferente.

- **Camada convolucional:** serve para extrair informações das imagens através da convolução
- **Camada de pooling:** serve para reduzir a dimensão das imagens provenientes da camada convolucional
 - Tipos de pooling: máximo, médio e soma

O que muda para uma rede convolucional?



Estrutura da Rede Convolucional VGG19

Parte 2

Invariância

- Uma função $f[x]$ é **invariante** a uma transformação $t[]$ se:

$$f[t[x]] = f[x]$$

i.e., a saída da função é a mesma, mesmo depois que a transformação é aplicada.

Exemplo:

- A imagem foi transladada, mas a nossa classificação precisa fornecer o mesmo resultado.



Equivariância

- A função $f[x]$ é **equivariante** a transformação $t[]$ se:

$$f[t[x]] = t[f[x]]$$

i.e., a saída é transformada da mesma forma que a entrada.

Exemplo:

- A imagem foi transladada e queremos que a segmentação também sofra uma translação igual.



Convolução* em 1D

- Contínua $\rightarrow (f * h)(x) = \int_{\mathbb{R}} f(z)h(x - z)dz$

- Discreta \rightarrow vetor de entrada \mathbf{x} :

$$\mathbf{x} = [x_1, x_2, \dots, x_I]$$

- A saída é uma soma ponderada dos vizinhos:

$$z_i = \omega_1 x_{i-1} + \omega_2 x_i + \omega_3 x_{i+1}$$

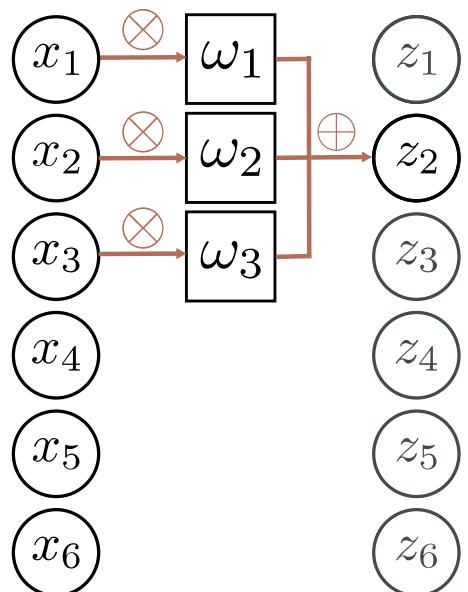
- **kernel convolucional ou filtro:**

$$\boldsymbol{\omega} = [\omega_1, \omega_2, \omega_3]^T$$

Tamanho do kernel= 3

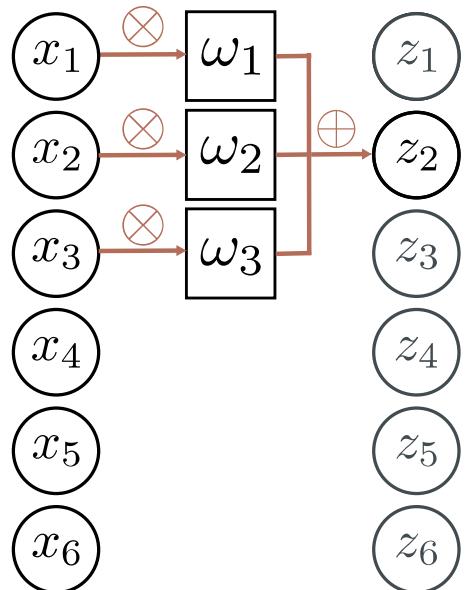
Convolução com kernel de tamanho 3

a)

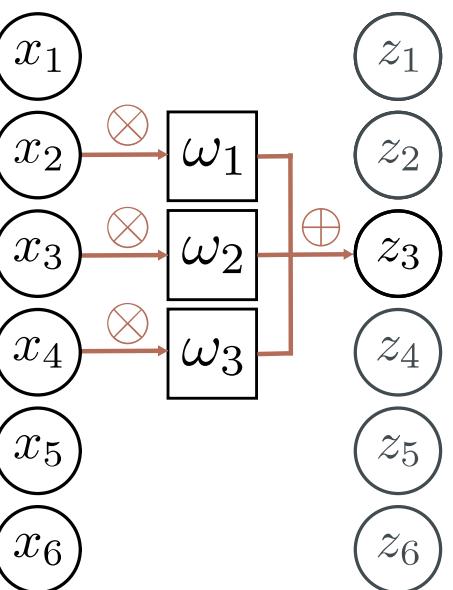


Convolução com kernel de tamanho 3

a)

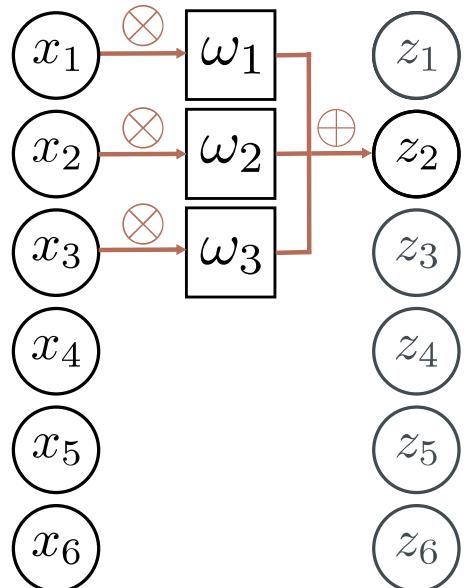


b)

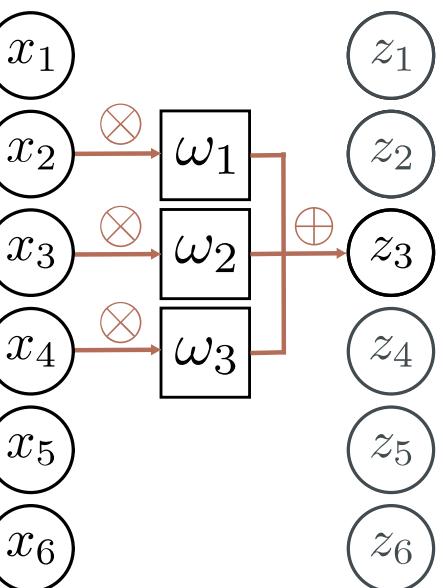


Convolução com kernel de tamanho 3

a)

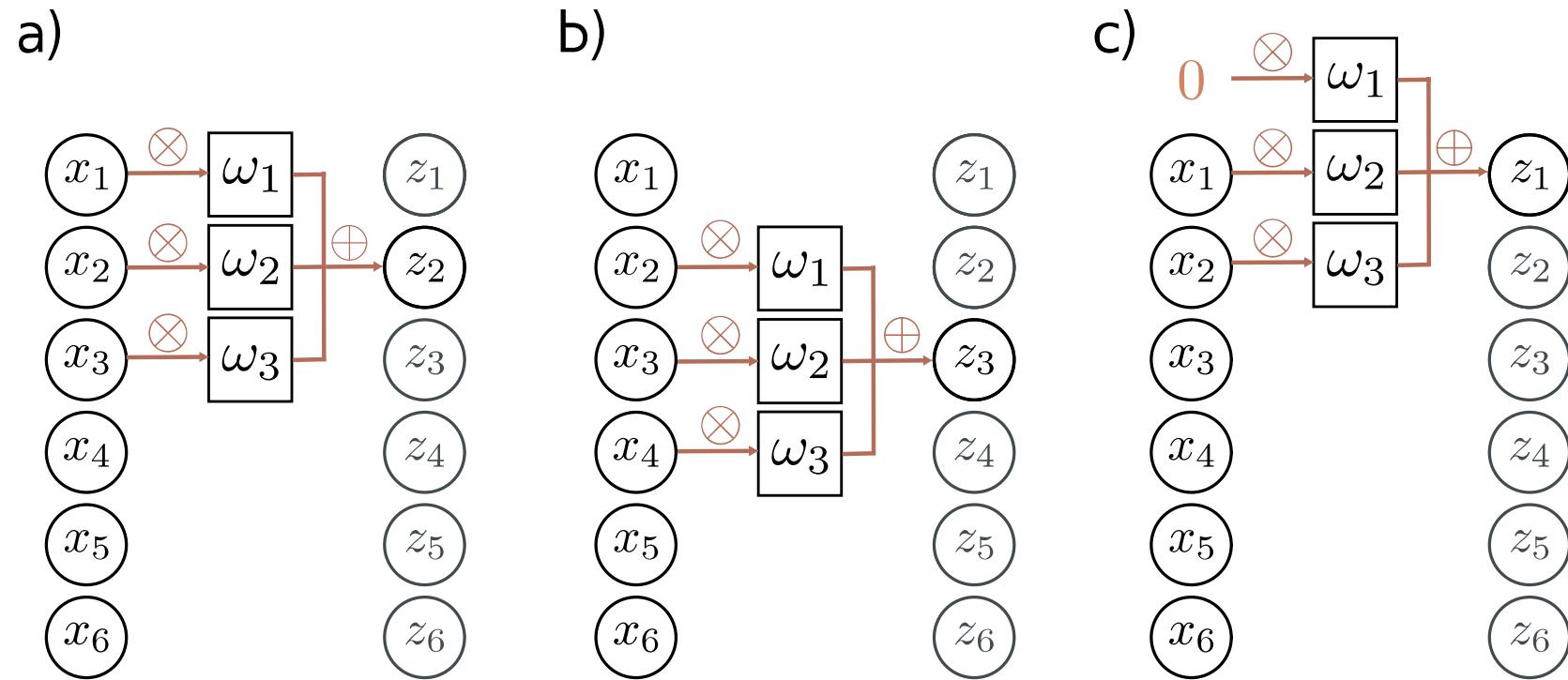


b)



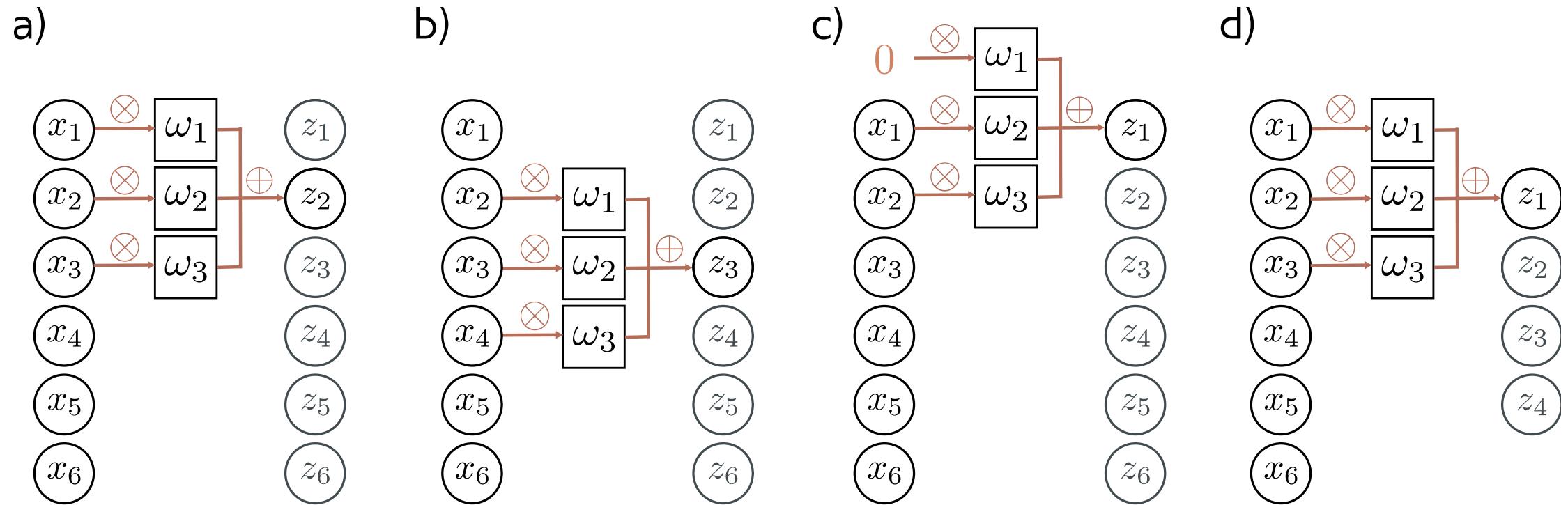
Equivariante a translação da entrada
 $f[t[x]] = t[f[x]]$

Zero padding



Trate as posições que estão além dos limites da entrada como zero.

Convoluçãoções ‘válidas’

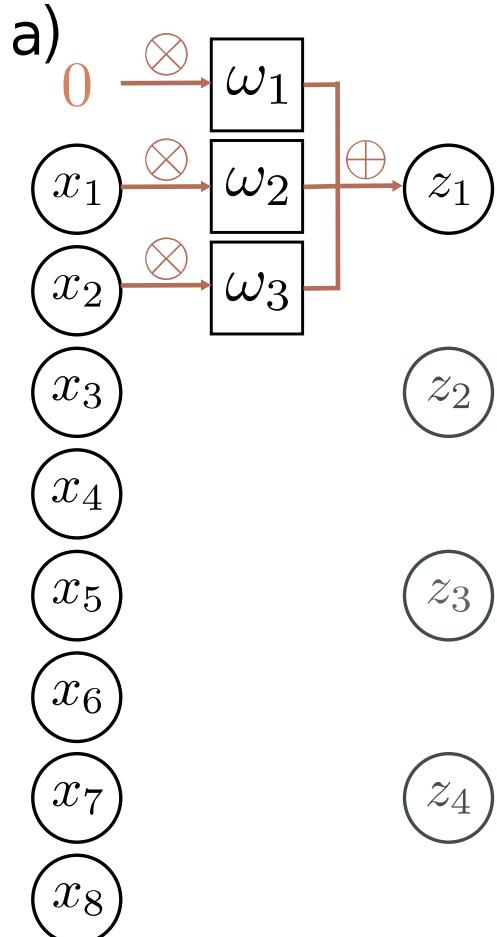


Processa apenas as posições onde o kernel cabe inteiramente na imagem (saída menor).

Stride, tamanho de kernel, e dilatação

- **Stride** = desloca por k posições para cada saída
 - Diminui o tamanho da saída em relação à entrada
- **Tamanho de kernel** = pondera um número diferente de entradas para cada saída
 - Combina informação oriunda de uma região maior
 - Kernel de tamanho 5 usa 5 parâmetros
- **Convoluçãoes dilatadas** ou **atrous** = intercalam valores do kernel com zeros
- Combina informação oriunda de uma região maior
 - Poucos parâmetros

Stride, tamanho do kernel, e dilatação

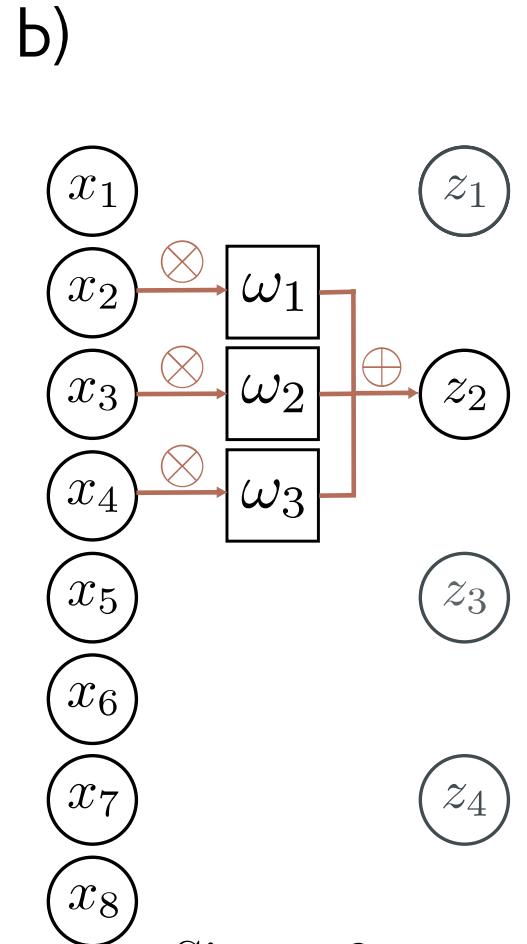
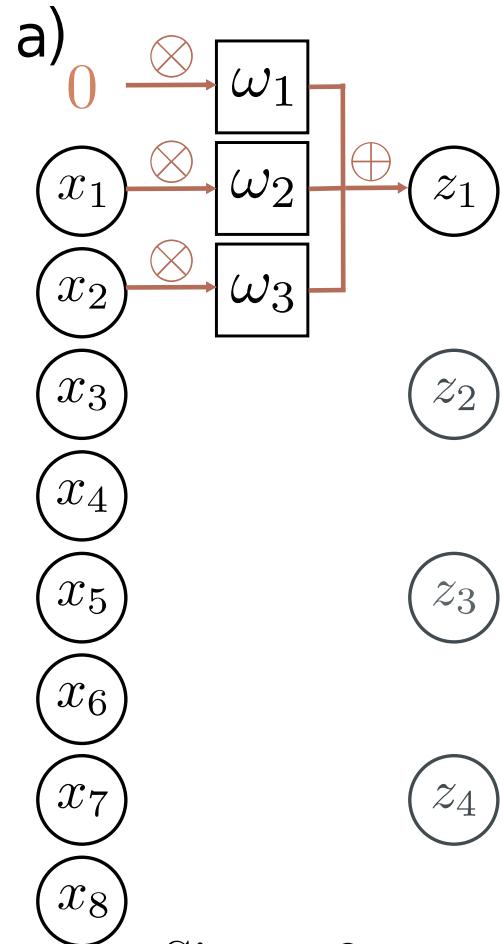


Size = 3

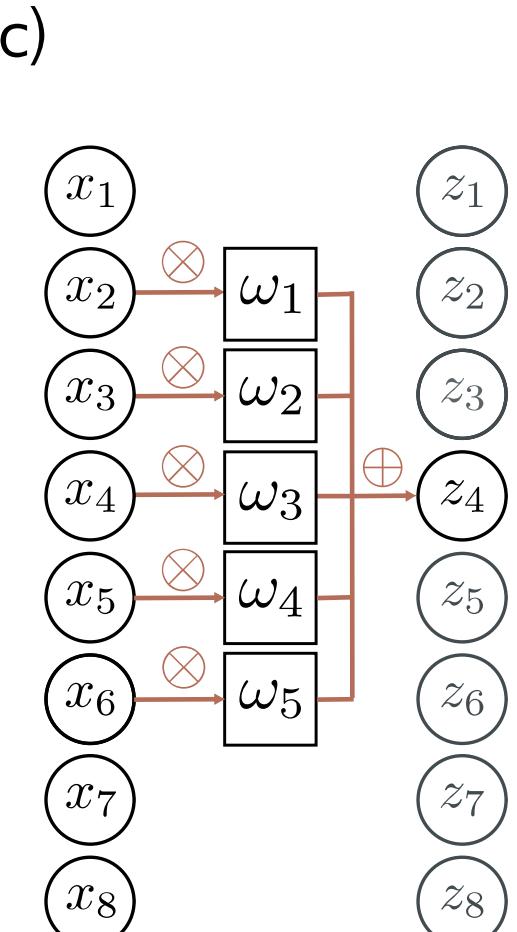
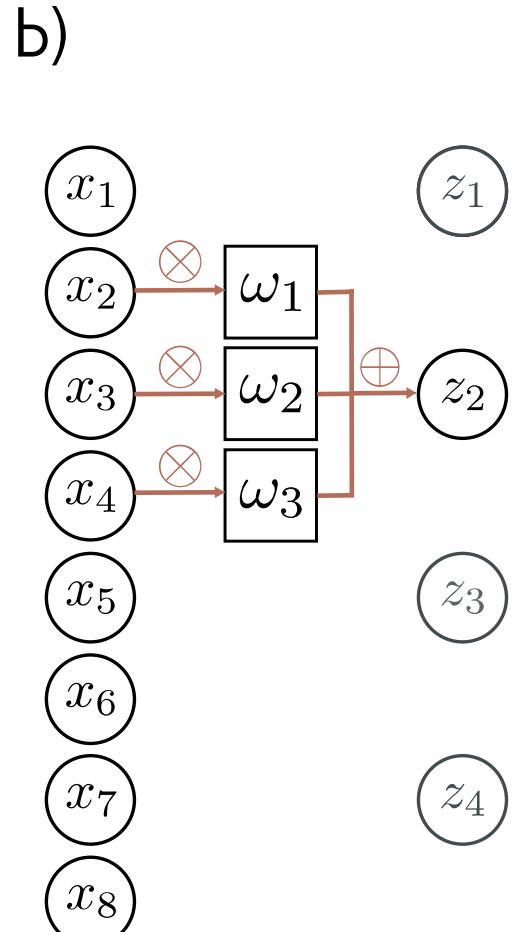
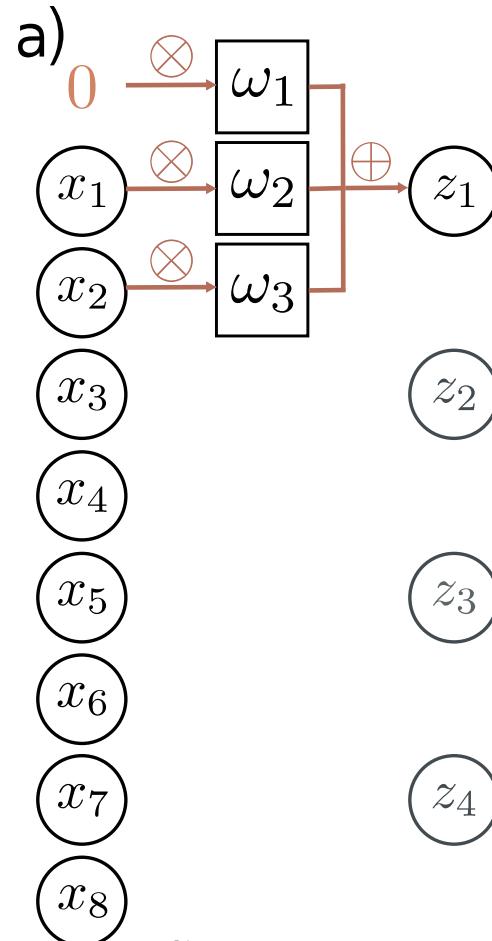
Stride = 2

Dilation = 1

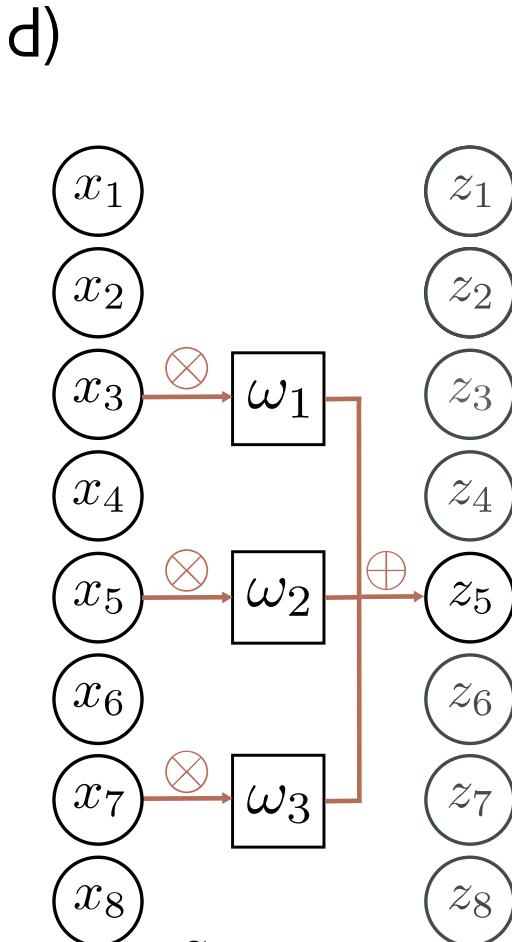
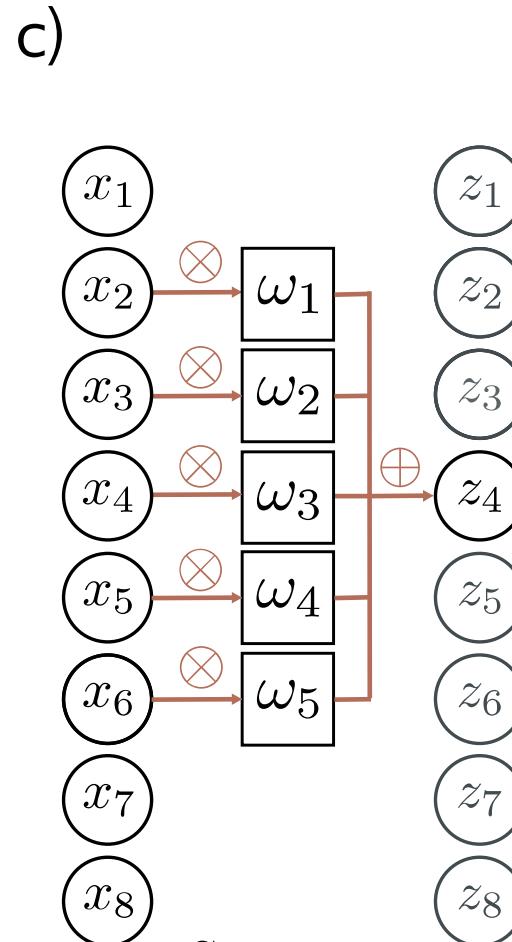
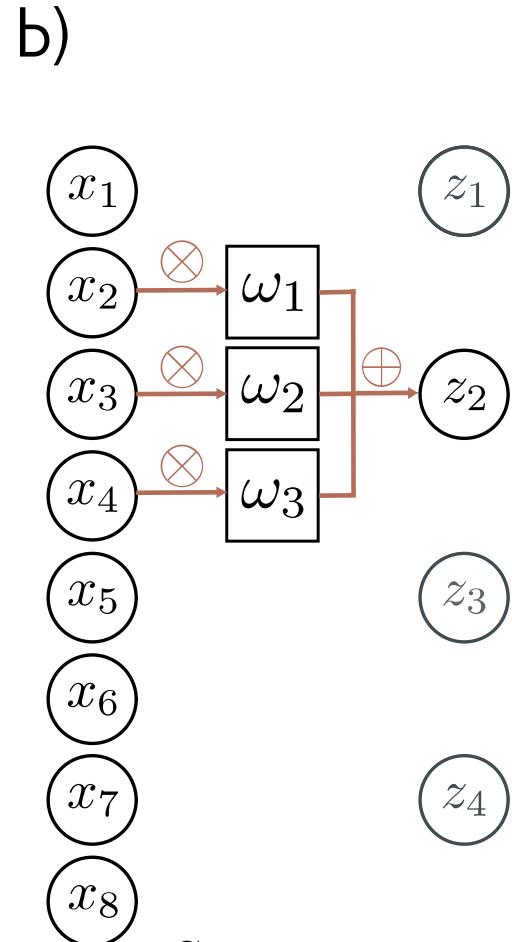
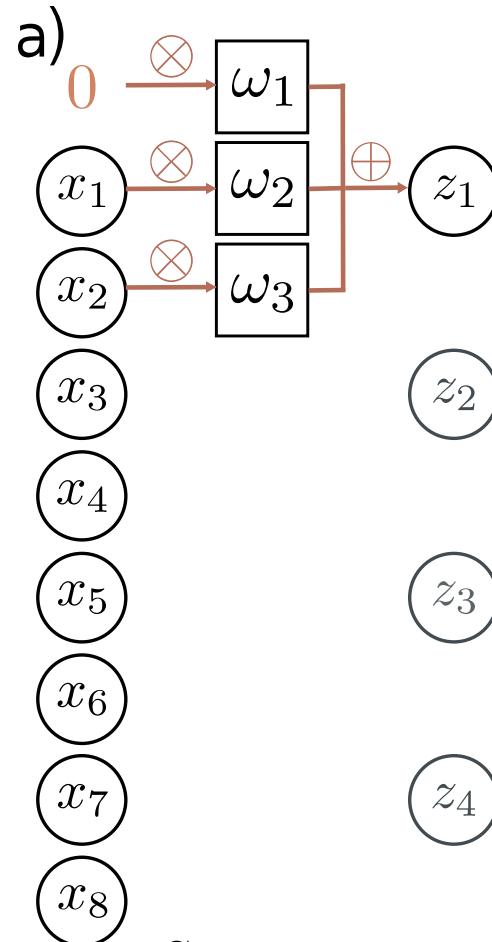
Stride, tamanho do kernel, e dilatação



Stride, tamanho do kernel, e dilatação



Stride, tamanho do kernel, e dilatação



Camada convolucional

Agora que vimos o que é convolução, o que fazer com elas?

Nas redes convolucionais adicionamos camadas que farão operações de convoluções, como mostrado anteriormente.

Mas, quais filtros são usados?

Os filtros são inicializados aleatoriamente e são aprendidos com o processo iterativo das redes.

Esses filtros que são aprendidos extraem informações referentes aos padrões das imagens.

Camada convolucional

$$\begin{aligned} h_i &= \text{a} [\beta + \omega_1 x_{i-1} + \omega_2 x_i + \omega_3 x_{i+1}] \\ &= \text{a} \left[\beta + \sum_{j=1}^3 \omega_j x_{i+j-2} \right] \end{aligned}$$

Camada convolucional (caso especial de MLP)

Rede convolucional:

$$\begin{aligned} h_i &= a [\beta + \omega_1 x_{i-1} + \omega_2 x_i + \omega_3 x_{i+1}] \\ &= a \left[\beta + \sum_{j=1}^3 \omega_j x_{i+j-2} \right] \end{aligned}$$

Rede neural MLP:

$$h_i = a \left[\beta_i + \sum_{j=1}^D \omega_{ij} x_j \right]$$

Camada convolucional (caso especial de MLP)

Rede convolucional:

$$\begin{aligned}
 h_i &= a [\beta + \omega_1 x_{i-1} + \omega_2 x_i + \omega_3 x_{i+1}] \\
 &= a \left[\beta + \sum_{j=1}^3 \omega_j x_{i+j-2} \right]
 \end{aligned}$$

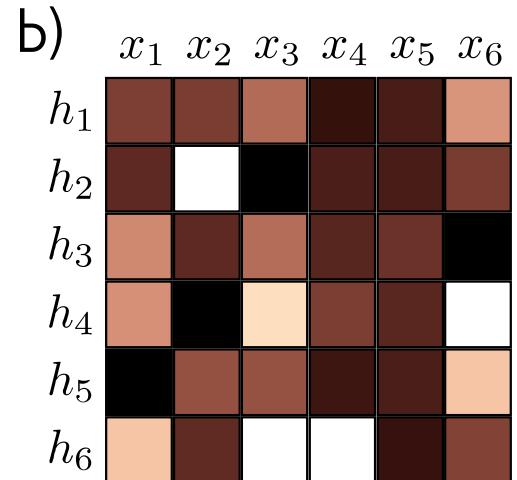
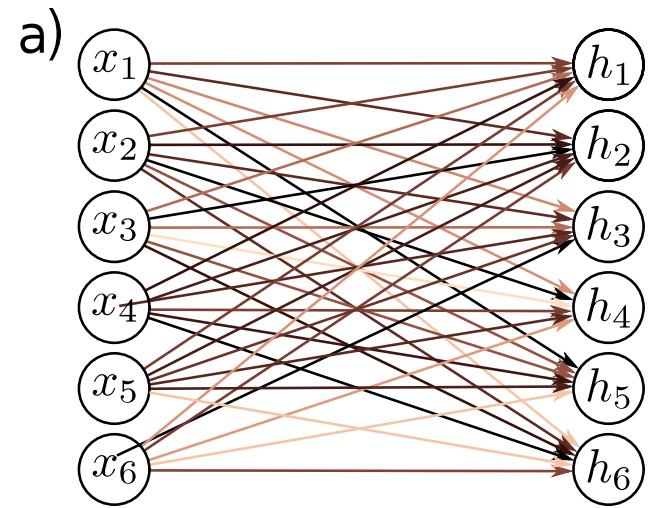
3 pesos, 1 viés

Rede neural MLP:

$$h_i = a \left[\beta_i + \sum_{j=1}^D \omega_{ij} x_j \right]$$

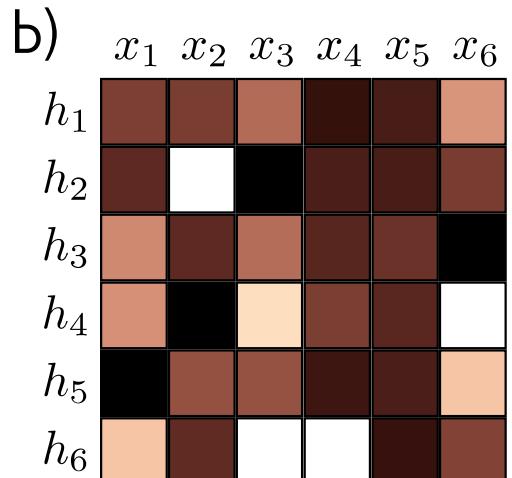
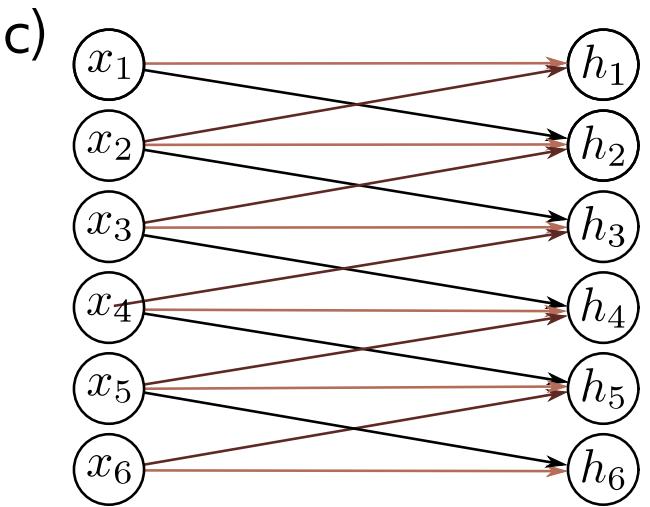
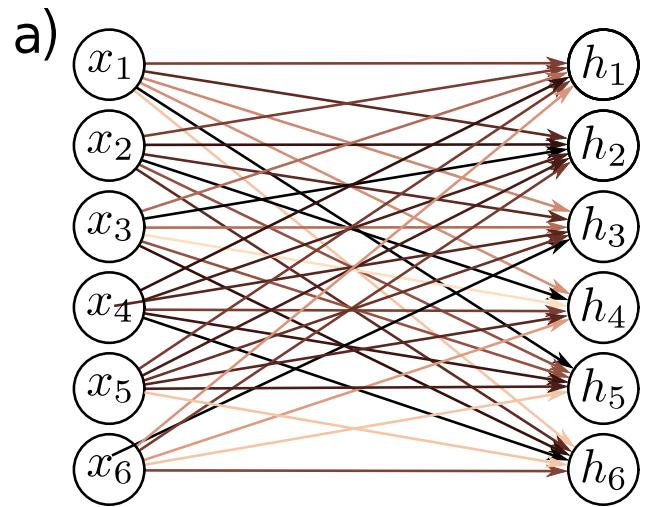
D² pesos, D viés

Camada convolucional (caso especial de MLP)

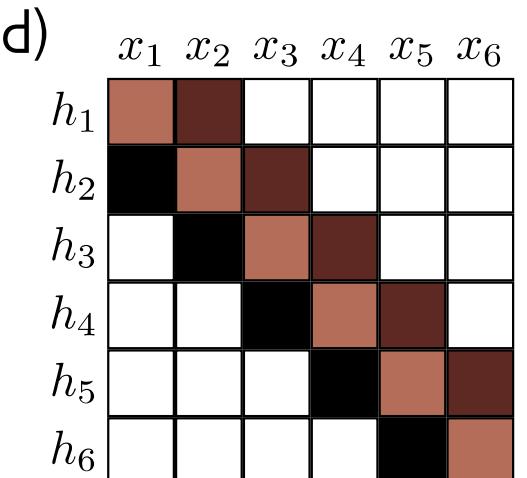


Fully connected network

Camada convolucional (caso especial de MLP)

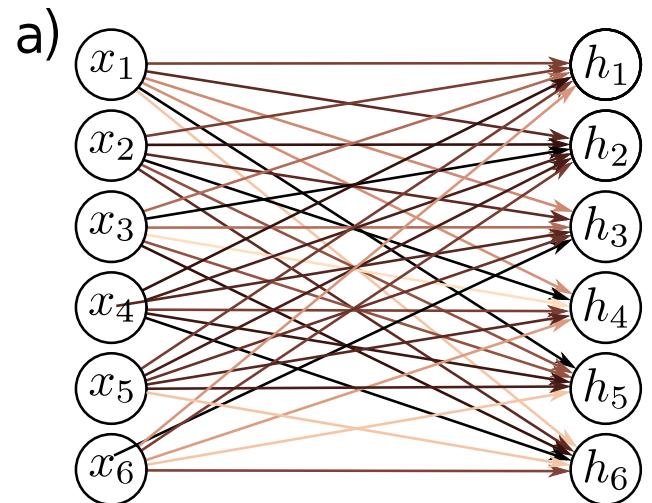


Fully connected network



Convolution, kernel 3,
stride 1, dilation 1

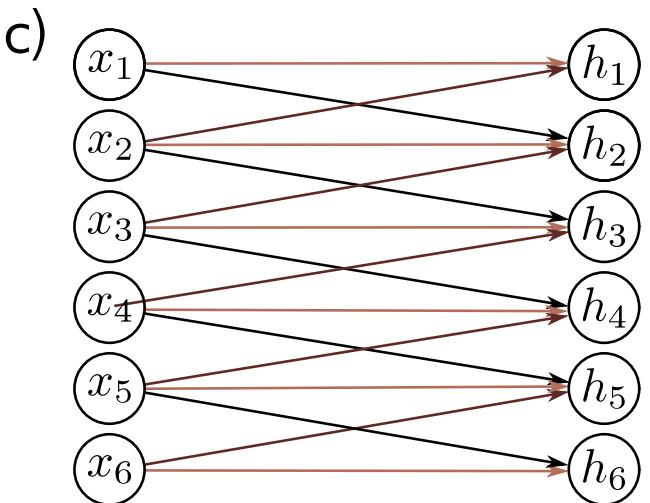
Camada convolucional (caso especial de MLP)



b)

	x_1	x_2	x_3	x_4	x_5	x_6
h_1	Dark Brown	Dark Brown	Light Brown	Dark Brown	Dark Brown	Light Brown
h_2	Dark Brown	White	Black	Dark Brown	Dark Brown	Dark Brown
h_3	Light Brown	Dark Brown	Light Brown	Dark Brown	Dark Brown	Black
h_4	Light Brown	Black	Light Brown	Dark Brown	Dark Brown	White
h_5	Black	Dark Brown	Dark Brown	Dark Brown	Dark Brown	Light Brown
h_6	Light Brown	Dark Brown	White	Dark Brown	Dark Brown	Dark Brown

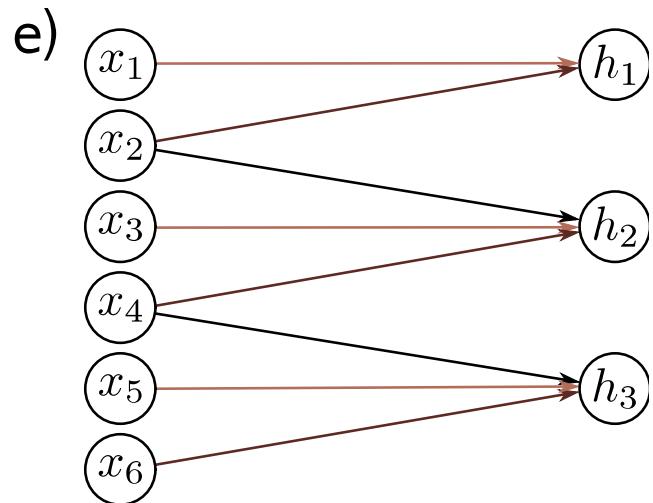
Rede Neural MLP



d)

	x_1	x_2	x_3	x_4	x_5	x_6
h_1	Light Brown	Dark Brown	Light Brown			
h_2	Black	White	Dark Brown			
h_3	White	Black	Light Brown	Dark Brown		
h_4			Black	Light Brown	Dark Brown	
h_5				Black	Light Brown	Dark Brown
h_6					Light Brown	Dark Brown

Convolução, kernel 3, stride 1,
dilatação 1, zero padding



f)

	x_1	x_2	x_3	x_4	x_5	x_6
h_1	Light Brown	Dark Brown				
h_2	White	Black	Light Brown	Dark Brown		
h_3				Black	Light Brown	Dark Brown

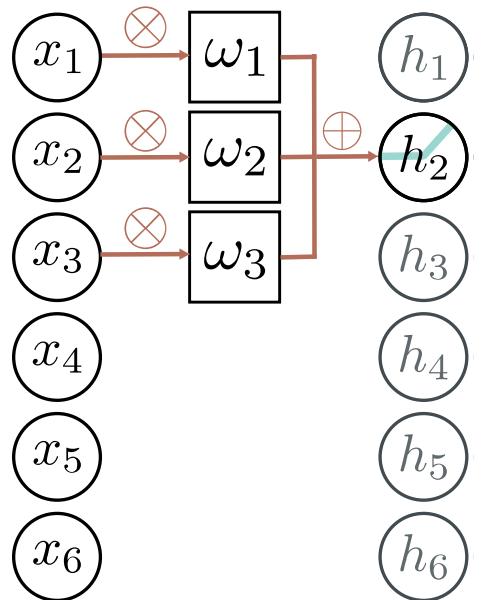
Convolução, kernel 3, stride 2,
dilatação 1, zero padding

Canais

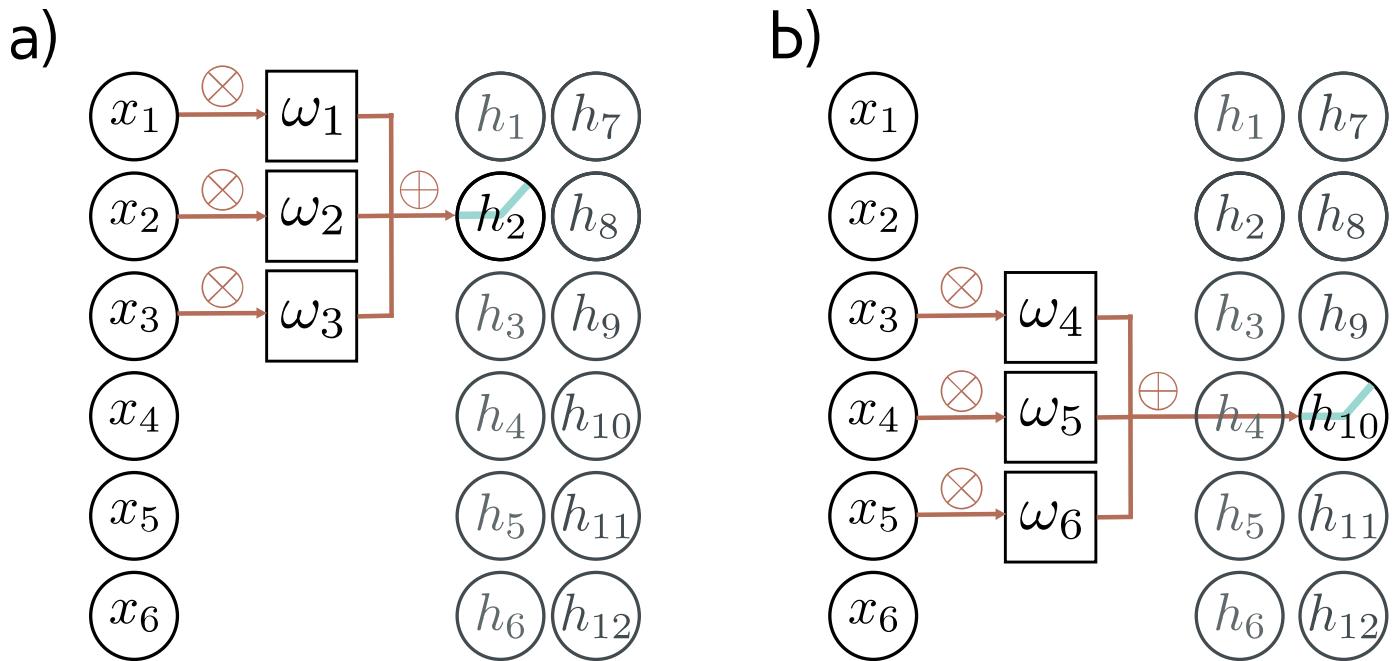
- A operação convolucional calcula a média das entradas
- E depois passa pela função ReLU.
- Isso acarreta perda de informação.
- Solução:
 - aplicar várias convoluções e empilhá-las em canais
 - Às vezes também chamados de mapas de características/recursos

Dois canais de saída, um canal de entrada

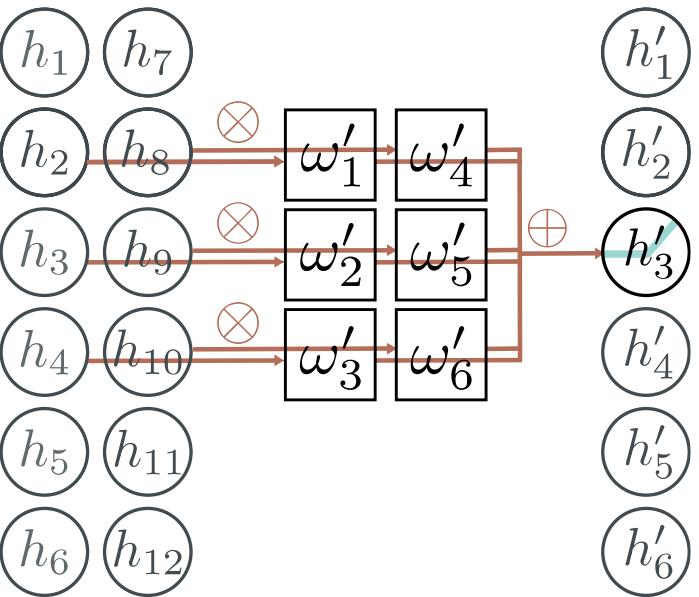
a)



Dois canais de saída, um canal de entrada



Dois canais de entrada, um canal de saída



Quantos parâmetros?

- Se há C_i canais de entrada e tamanho de kernel K

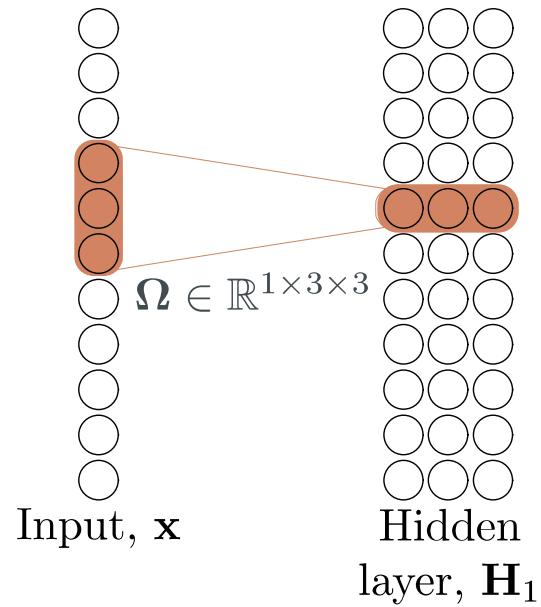
$$\Omega \in \mathbb{R}^{C_i \times K} \quad \beta \in \mathbb{R}$$

- Se há C_i canais de entrada e C_o canais de saída

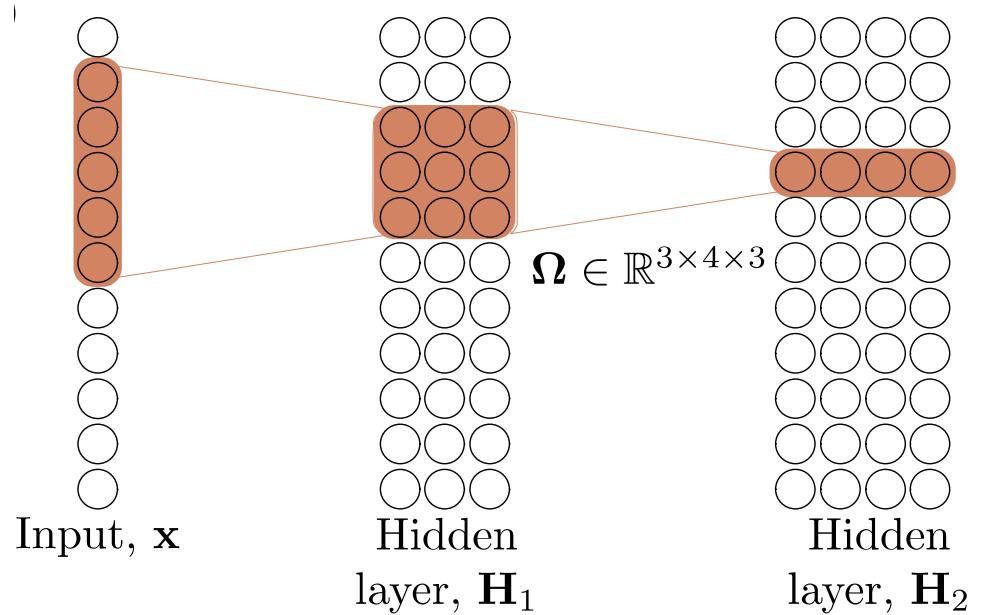
$$\Omega \in \mathbb{R}^{C_i \times C_o \times K} \quad \beta \in \mathbb{R}^{C_o}$$

Receptive fields

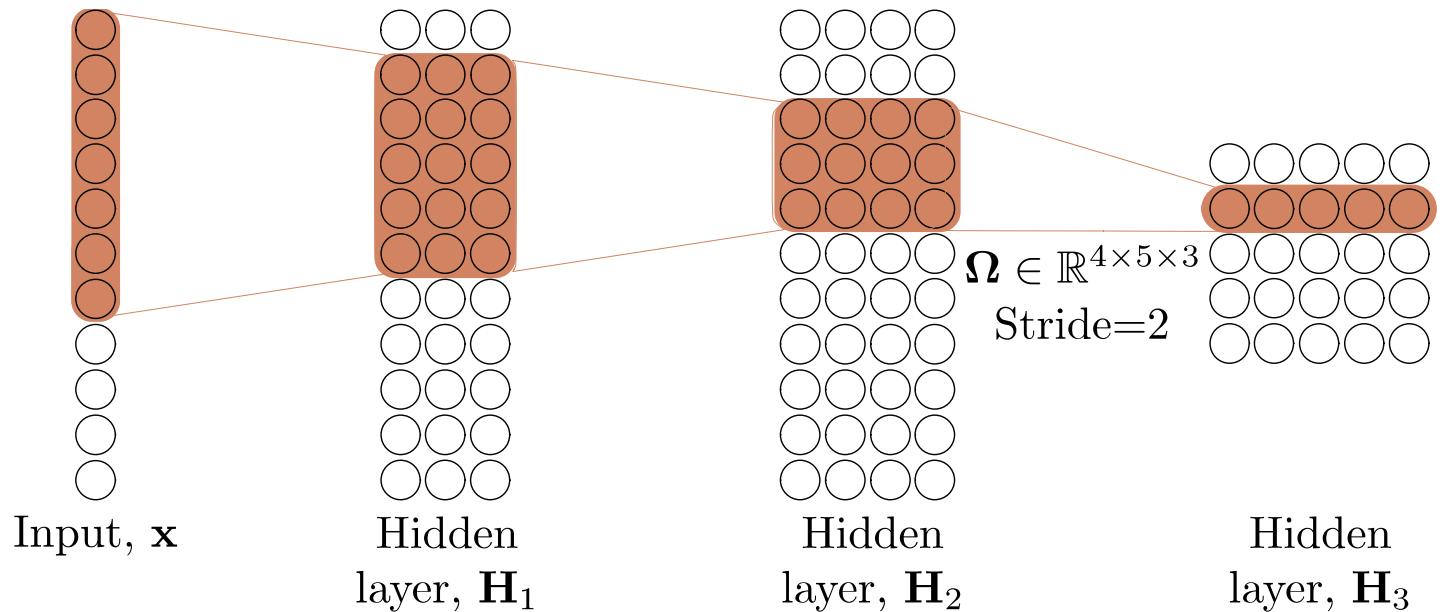
- Os receptive fields de uma unidade oculta na rede é a região da entrada original que as alimentam.



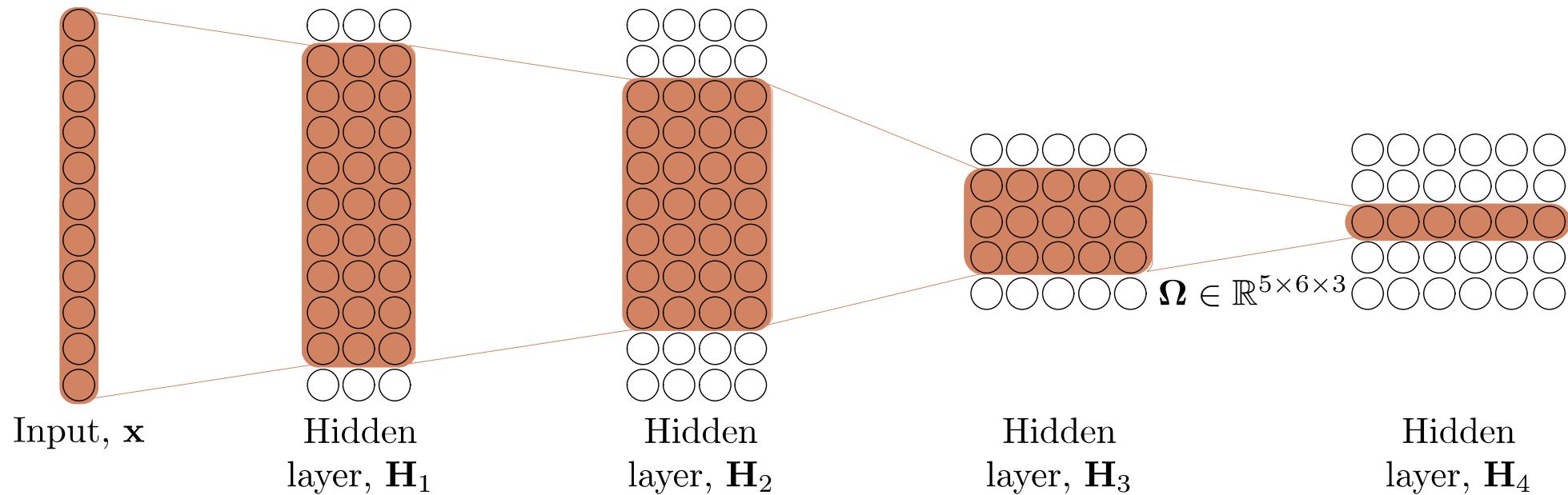
Receptive fields



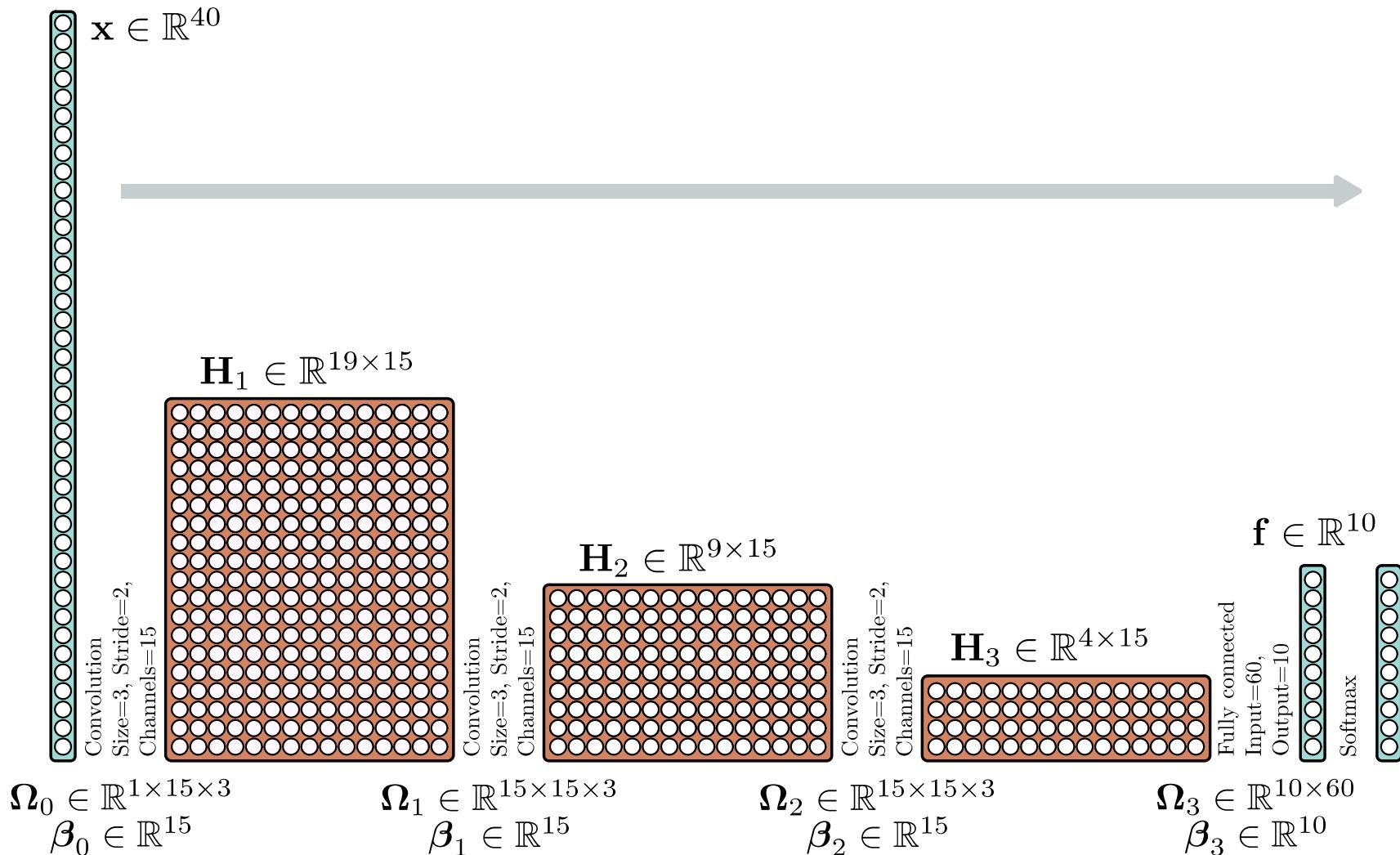
Receptive fields



Receptive fields



MNIST-1D: rede convolucional



Convolução 2D

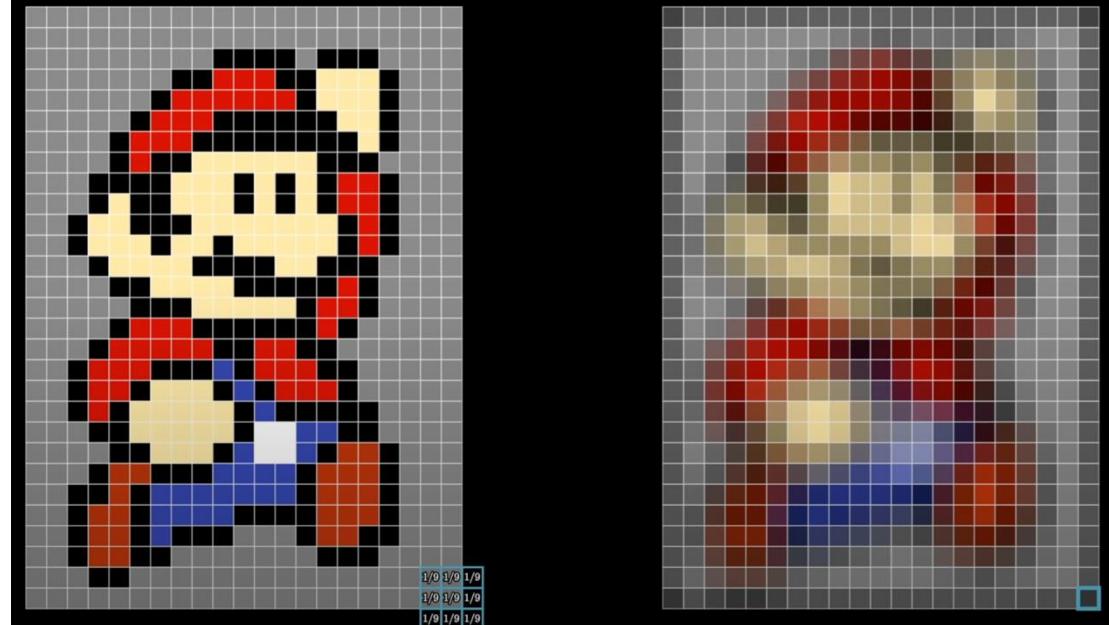
- Contínua $\rightarrow (f * h)(x, y) = \iint_{\mathbb{R}^2} f(z, w)h(x - z, y - w)dzdw$
- Convolução em 2D
 - Soma ponderada sobre uma região $K \times K$
 - $K \times K$ pesos
- Constrói uma camada convolucional adicionando o viés e passando pela função de ativação

$$h_{i,j} = a \left[\beta + \sum_{m=1}^3 \sum_{n=1}^3 \omega_{m,n} x_{i+m-2, j+n-2} \right]$$

Filtros e Convolução



Aqui foi aplicado um filtro de detecção de bordas no Kirby e um de borramento no Mario. No caso das redes neurais iremos filtrá-las para obter estruturas simplificadas das imagens.



Filtragem e Convolução

No caso da detecção de bordas com o filtro de Sobel foram usadas.

- Detecção horizontal

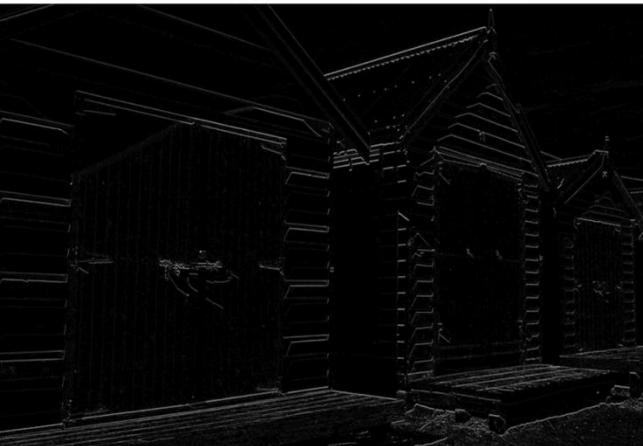
$$h_1 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$



(a)

- Deteção vertical

$$h_2 = h_1^T = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$



(c)



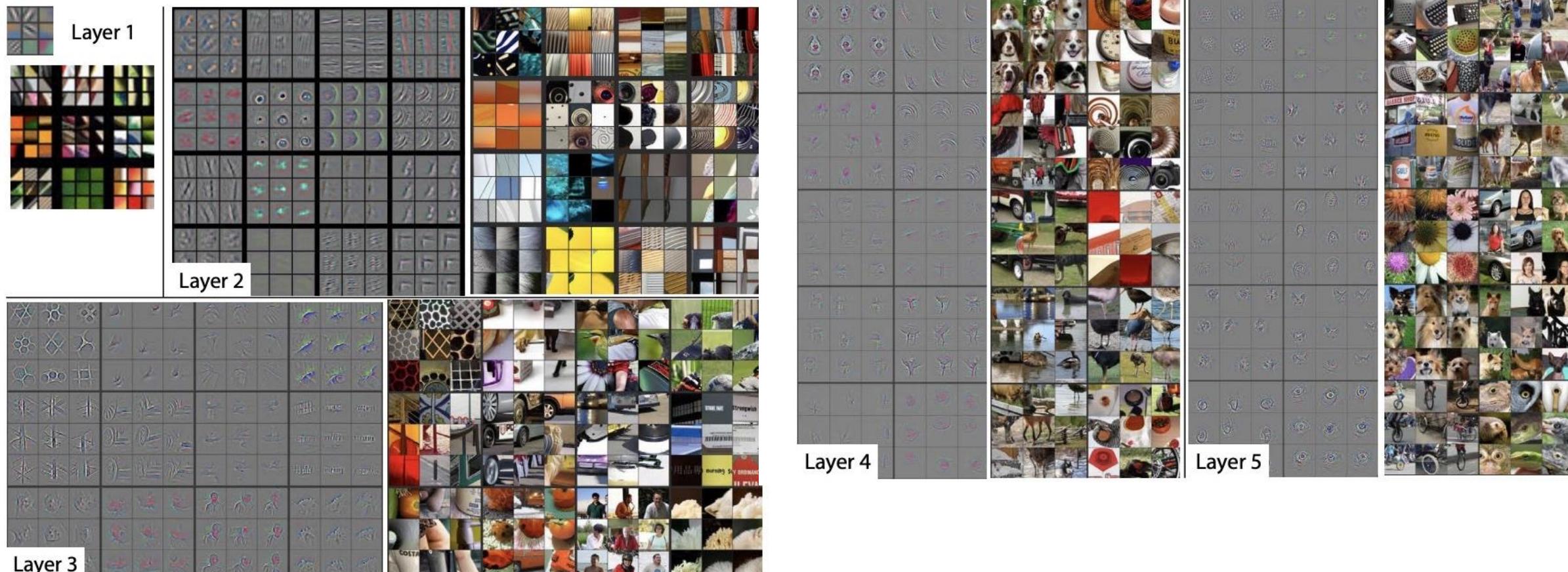
(b)



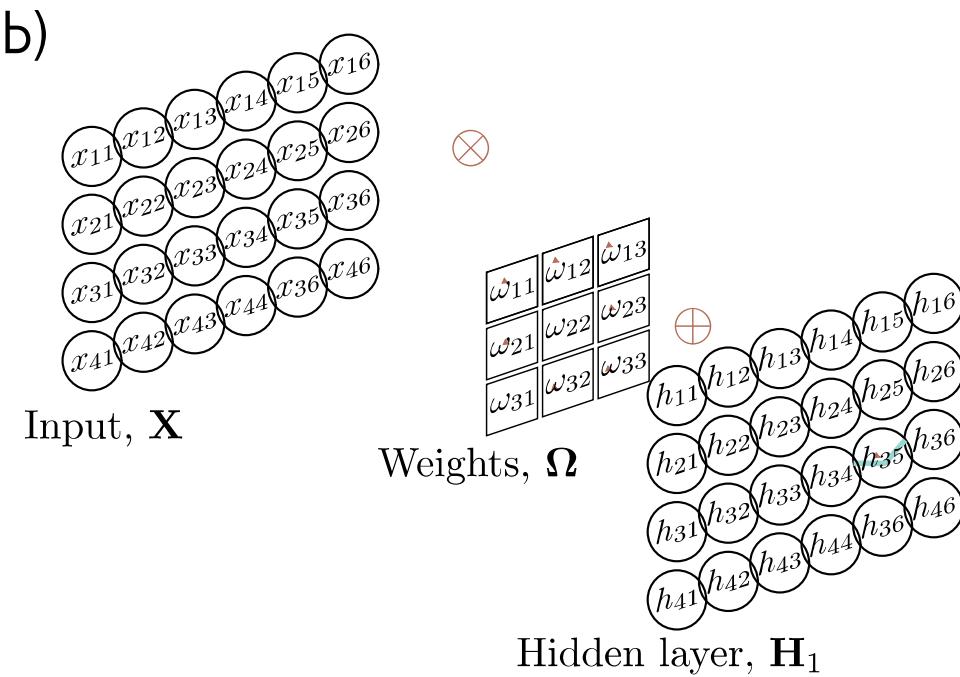
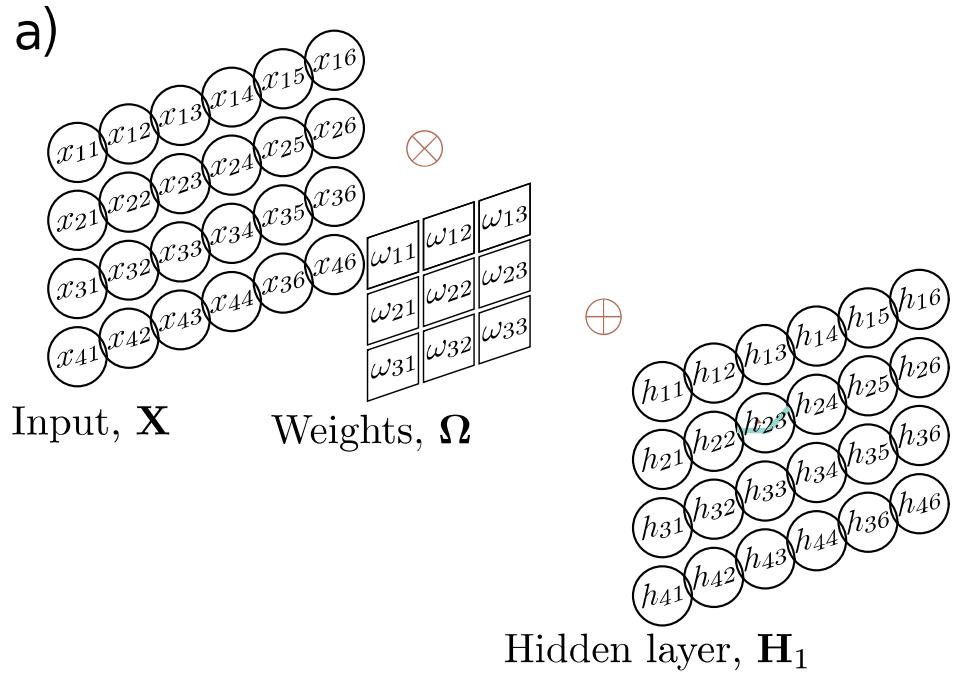
(d)

Filtragem e Convolução

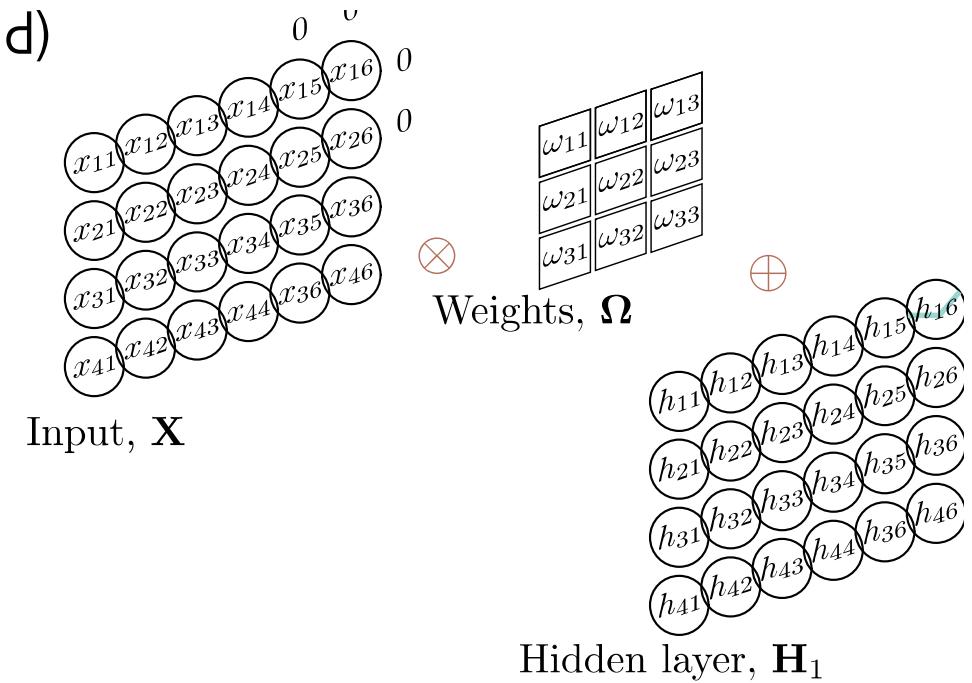
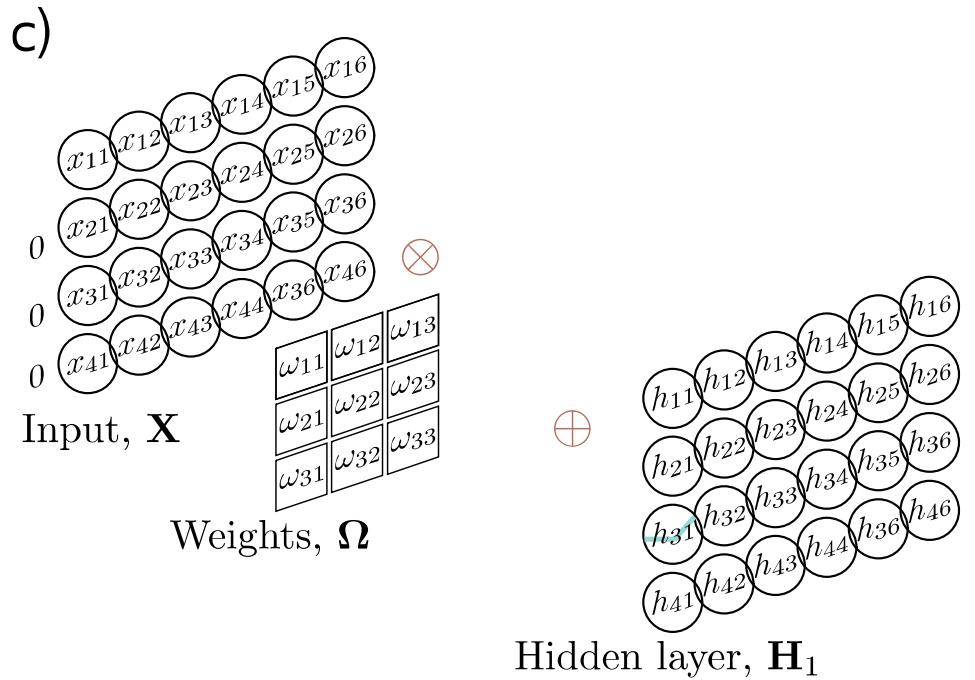
No trabalho “Visualizing and Understanding Convolutional Networks”, de Zeiler e Fergus, foi proposto um método para visualizar e entender as representações internas aprendidas pelas CNNs. A figura a seguir fornece a compreensão de como a informação dos detalhes é extraída das imagens.



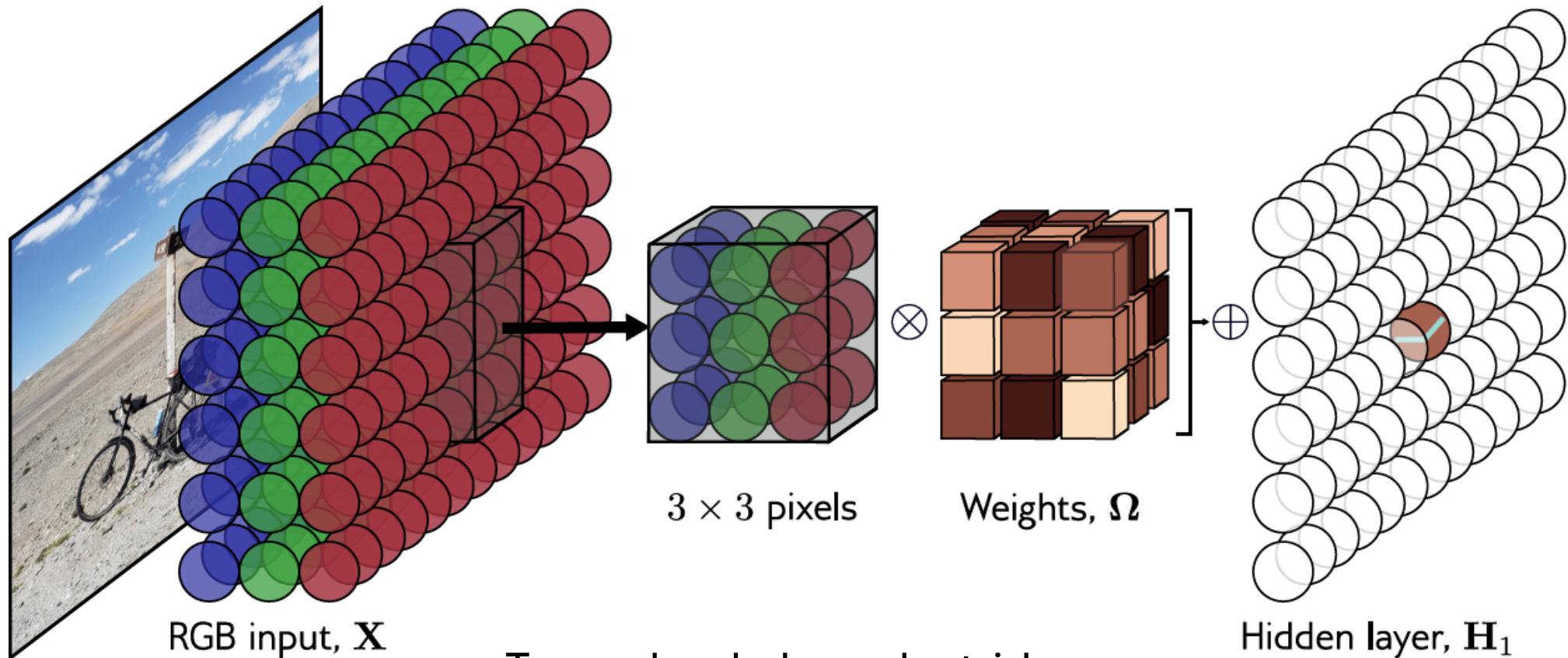
Convolução 2D



Convolução 2D



Canais na Convolução 2D



Tamanho do kernel, stride,
dilatação funciona da mesma
forma já esperada.

Quantos parâmetros?

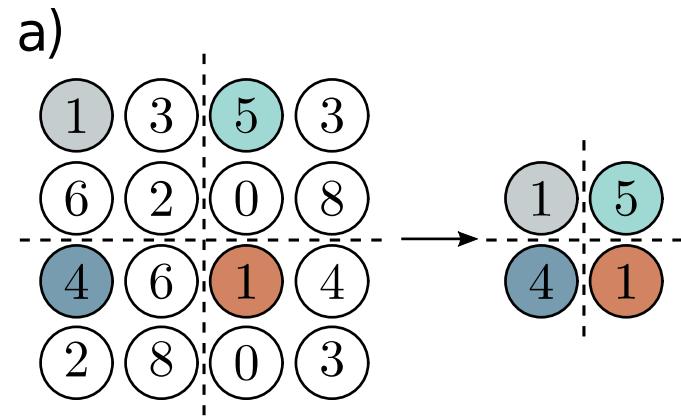
- Se há C_i canais de entrada e tamanho de kernel K x K

$$\omega \in \mathbb{R}^{C_i \times K \times K} \quad \beta \in \mathbb{R}$$

- Se há C_i canais de entrada e C_o canais de saída

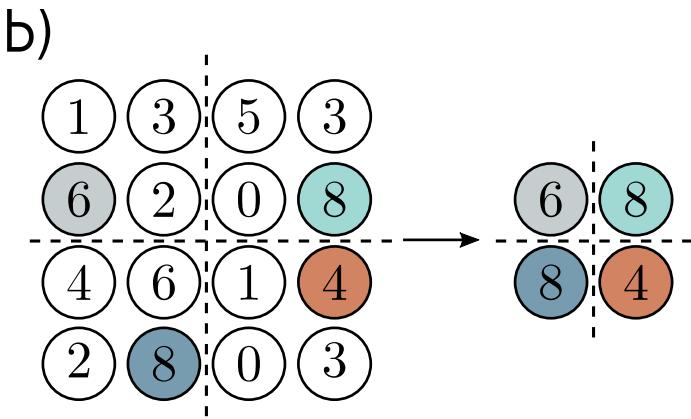
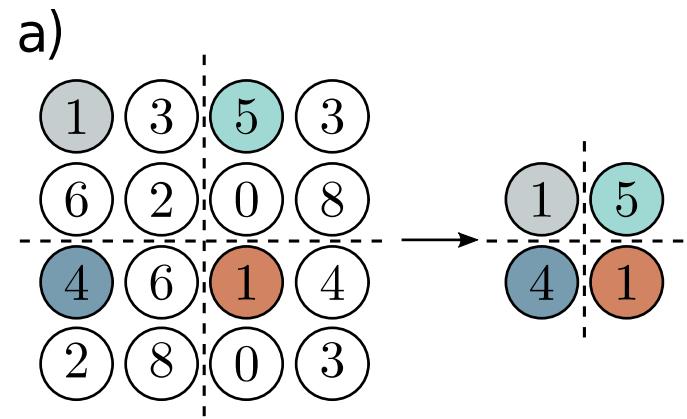
$$\omega \in \mathbb{R}^{C_i \times C_o \times K \times K} \quad \beta \in \mathbb{R}^{C_o}$$

Downsampling (pooling)



Sample every other
position (equivalent
to stride two)

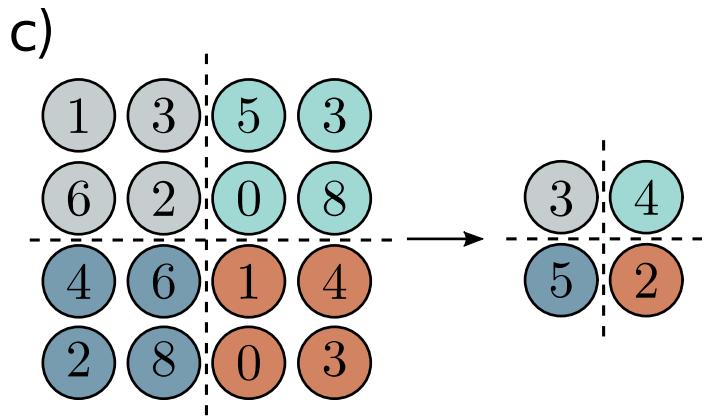
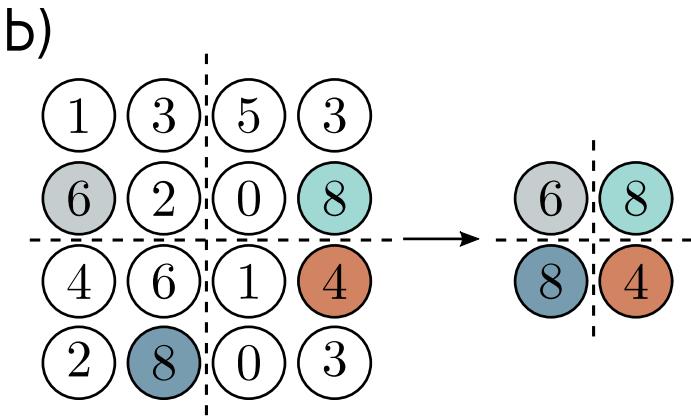
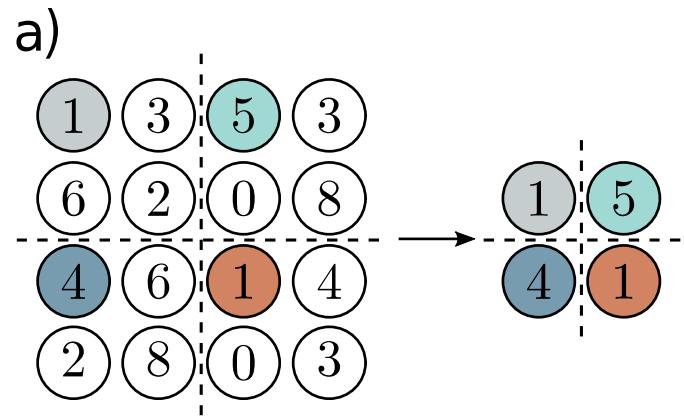
Downsampling



Sample every other position (equivalent to stride two)

Max pooling
 (partial invariance to translation)

Downsampling



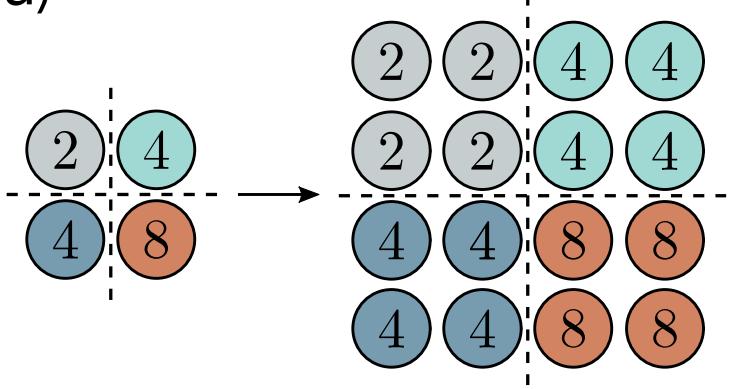
Sample every other position (equivalent to stride two)

Max pooling
(partial invariance to translation)

Mean pooling

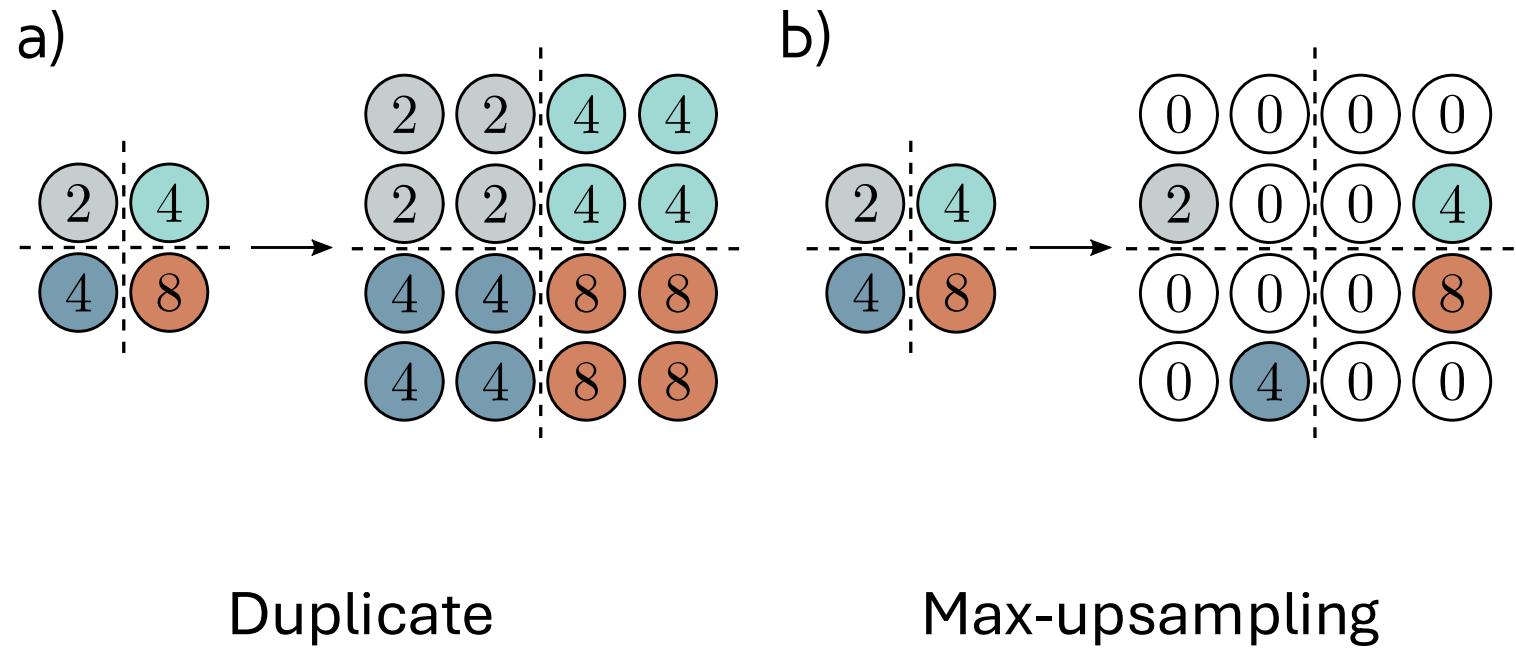
Upsampling

a)

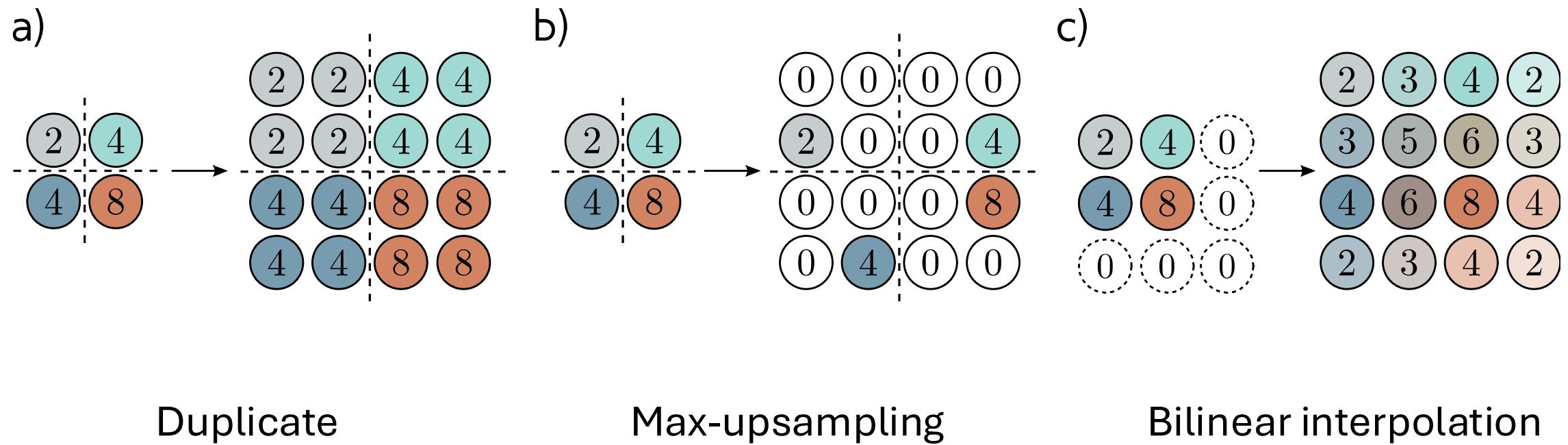


Duplicate

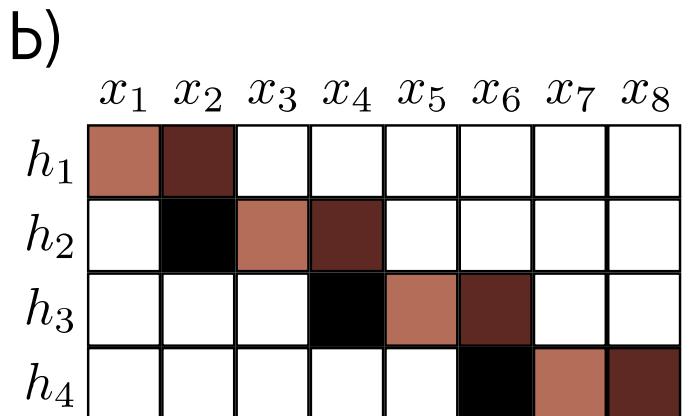
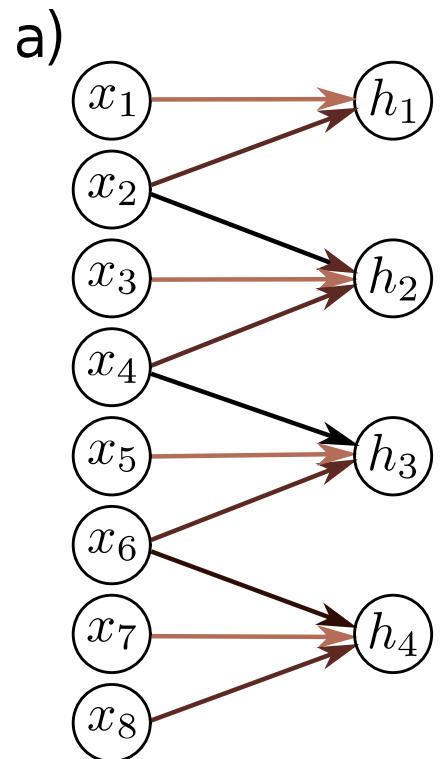
Upsampling



Upsampling

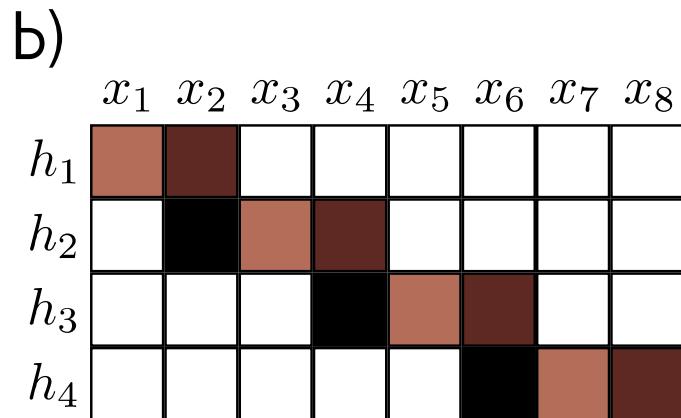
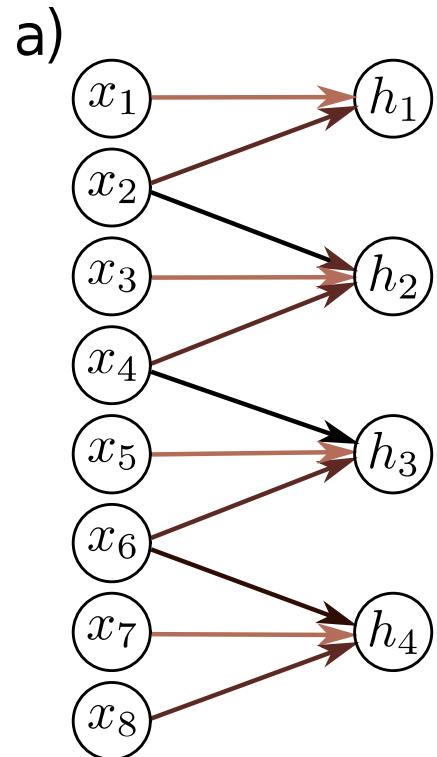


Convoluçãoes transpostas

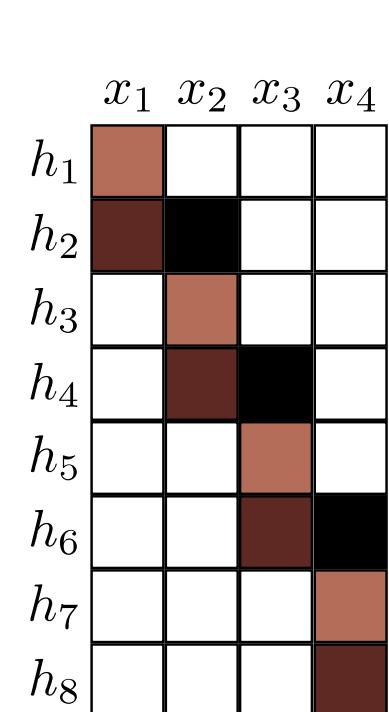
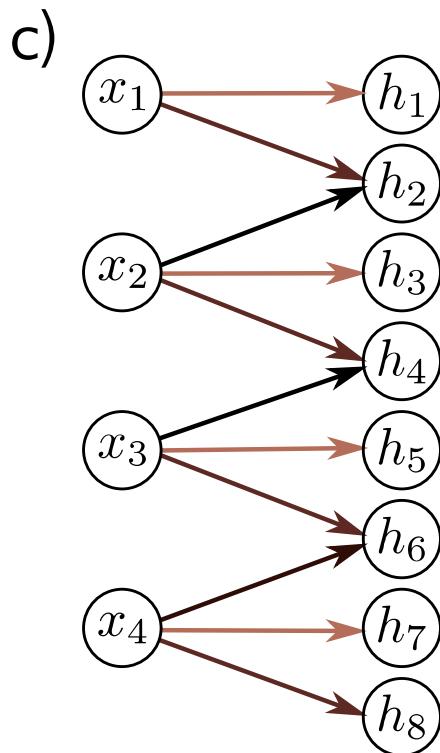


Kernel size 3, Stride 2 convolution

Convoluçãoes transpostas

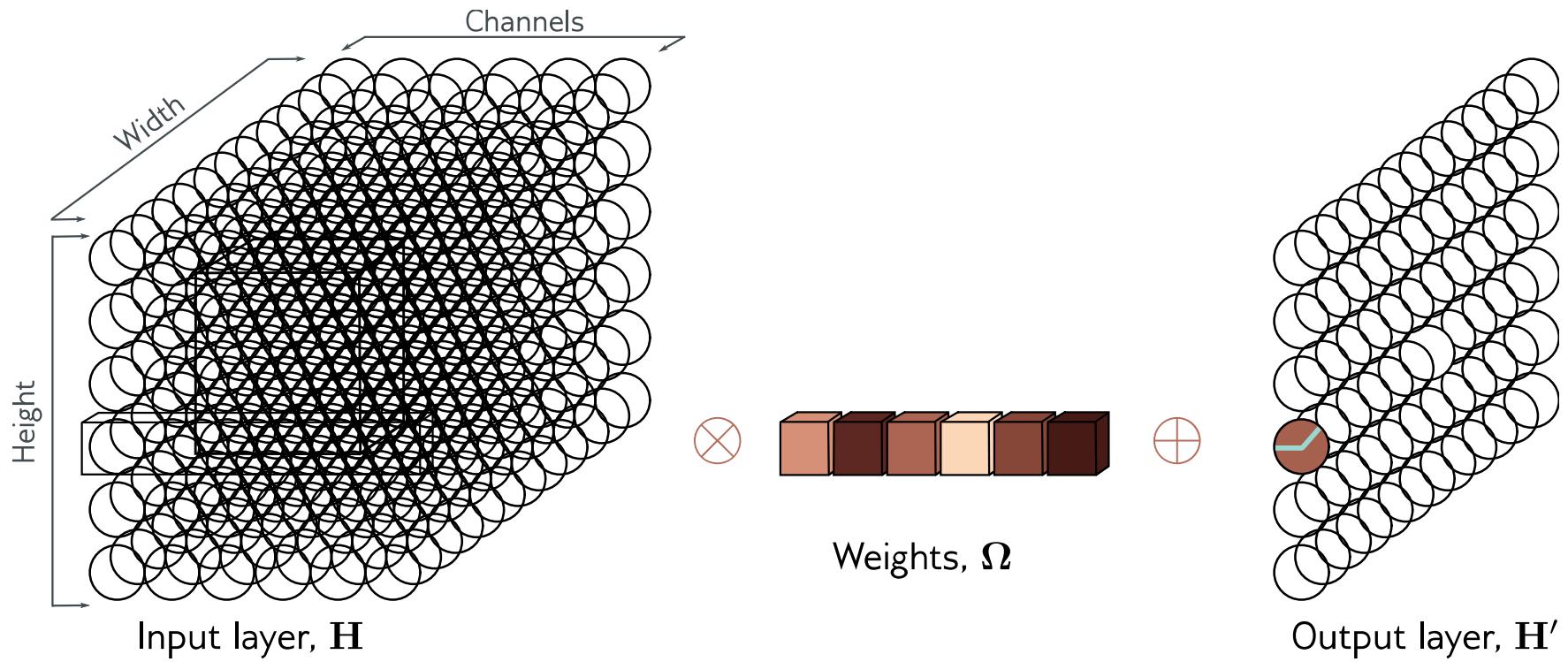


Kernel de tamanho 3, Stride 2 convolução



Convolução transposta

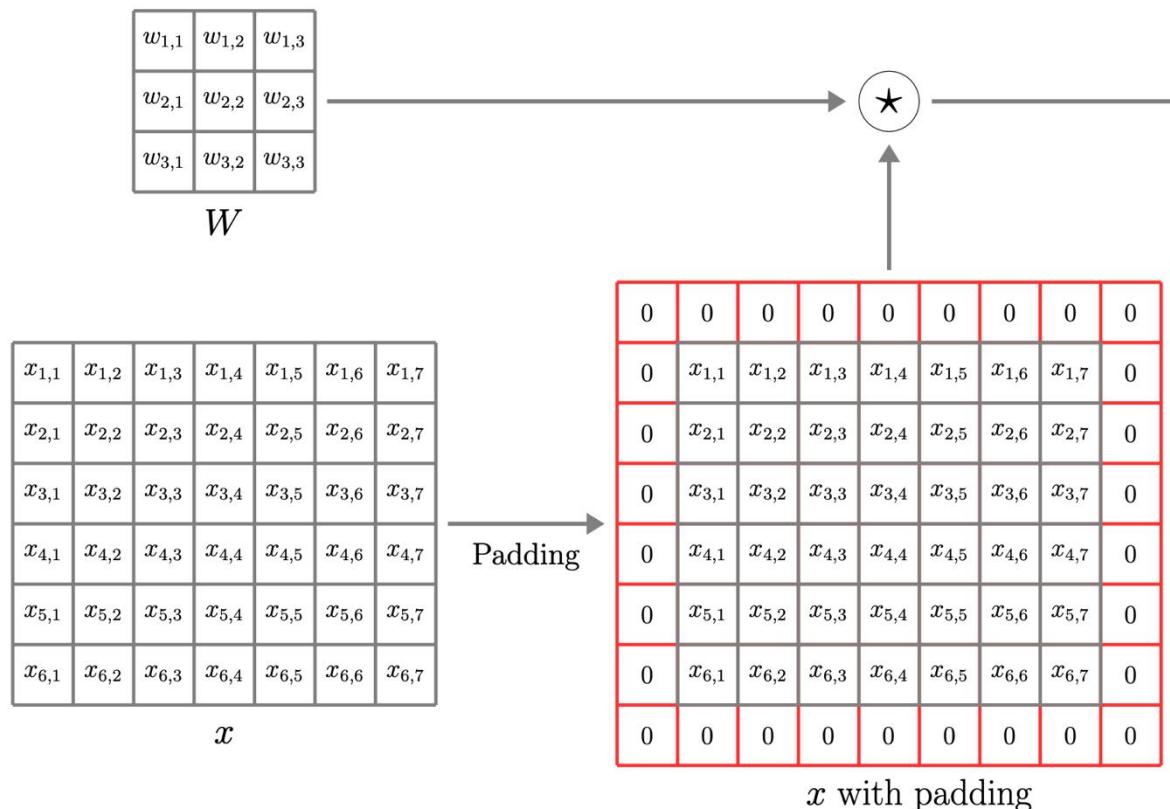
Convolução 1x1



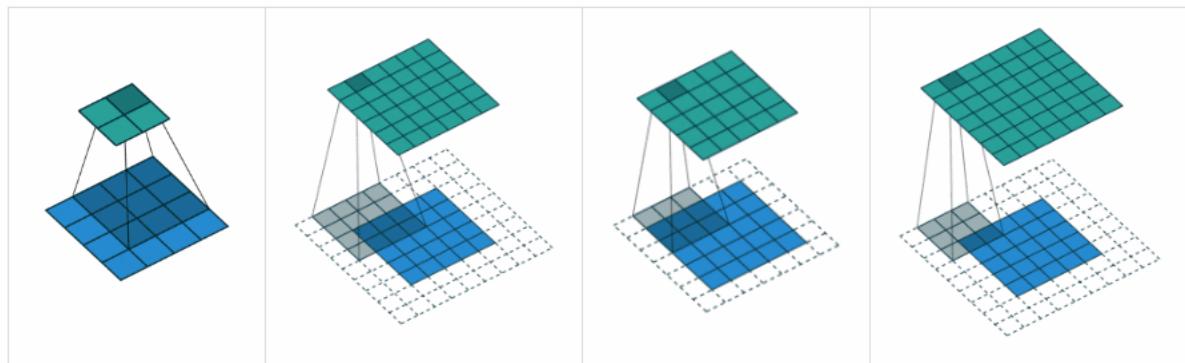
- Mistura canais
- Pode alterar o número de canais
- Equivalente a executar a mesma rede totalmente conectada em cada posição

Zero Padding 2D

Devido à operação de convolução, a dimensão da imagem de saída é menor que a dimensão da imagem de entrada. Para obtermos uma imagem de mesmo tamanho precisamos fazer um preenchimento na imagem: *zero padding*.



$z_{1,1}$	$z_{1,2}$	$z_{1,3}$	$z_{1,4}$	$z_{1,5}$	$z_{1,6}$	$z_{1,7}$
$z_{2,1}$	$z_{2,2}$	$z_{2,3}$	$z_{2,4}$	$z_{2,5}$	$z_{2,6}$	$z_{2,7}$
$z_{3,1}$	$z_{3,2}$	$z_{3,3}$	$z_{3,4}$	$z_{3,5}$	$z_{3,6}$	$z_{3,7}$
$z_{4,1}$	$z_{4,2}$	$z_{4,3}$	$z_{4,4}$	$z_{4,5}$	$z_{4,6}$	$z_{4,7}$
$z_{5,1}$	$z_{5,2}$	$z_{5,3}$	$z_{5,4}$	$z_{5,5}$	$z_{5,6}$	$z_{5,7}$
$z_{6,1}$	$z_{6,2}$	$z_{6,3}$	$z_{6,4}$	$z_{6,5}$	$z_{6,6}$	$z_{6,7}$



Parte 3

O que é *overfitting*?

O que é?

Ocorre quando o modelo aprende os dados de treinamento bem demais, memorizando não apenas os padrões úteis, mas também o ruído e os detalhes irrelevantes.

- É como um aluno que decora as respostas exatas para uma prova, em vez de entender a matéria. Ele acerta 100% daquela prova, mas falha em qualquer questão ligeiramente diferente.

Sintomas:

- Excelente desempenho nos dados de treino (acurácia alta, erro baixo).
- Desempenho ruim em dados novos (de validação/teste).

Consequência Principal:

O modelo não consegue generalizar seu conhecimento, tornando-se inútil para previsões no mundo real.

Regularização

Impõe (ou adiciona, ou altera) condições de forma que a solução da otimização apresente um melhor comportamento.

No caso das redes neurais a adição de um termo de regularização, por exemplo, é uma abordagem usual para evitar o *overfitting* e melhorar o desempenho da generalização.

Uma das maneiras de se executar uma regularização nas redes neurais é adicionar à função de perda um termo de regularização $R_\lambda(\theta)$, isto é $\mathcal{L}_2(\theta) = \mathcal{L}(\theta) + R_\lambda(\theta)$. Isso restringe a flexibilidade do modelo, mas às vezes essa restrição é necessária para evitar o *overfitting*.

Alguns modelos que são comuns para regularização são as funções penalidades L_1 : $R_\lambda(\theta) = \lambda \|\theta\|_1 = \lambda \sum_j |\theta_j|$, conhecida como regularização Lasso, e

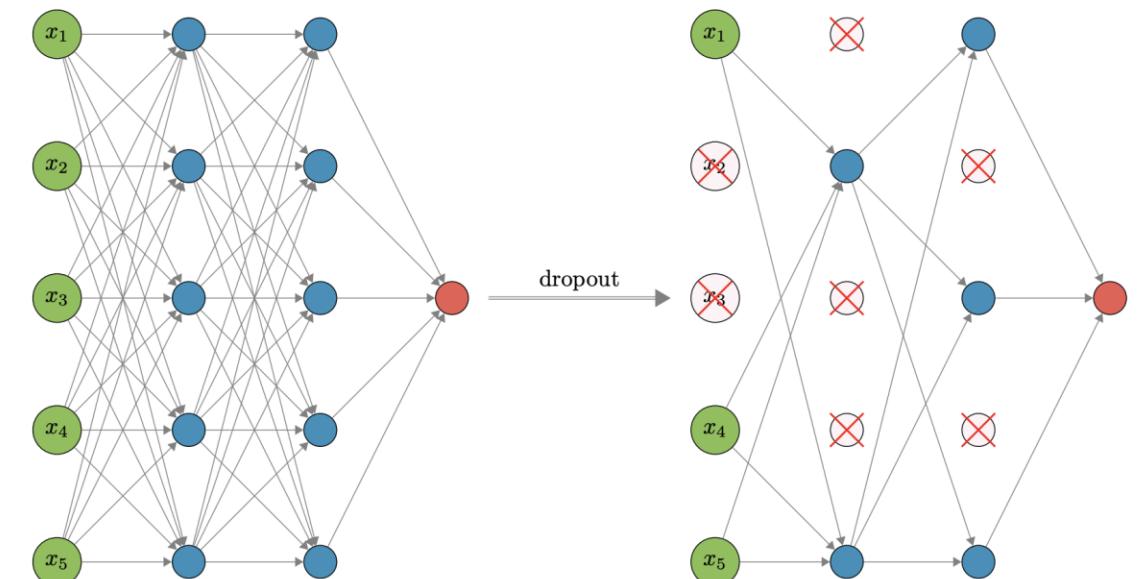
L_2 : $R_\lambda(\theta) = \frac{\lambda}{2} \|\theta\|_2^2 = \frac{\lambda}{2} \sum_j \theta_j^2$, conhecida como regularização de Tikhonov.

Dropout

Dropout é outra técnica de regularização em que, durante o treinamento da rede neural, **unidades** são desligadas aleatoriamente. **Missão: mitigar o overfitting.**

Desligar uma unidade significa removê-la temporariamente da rede, junto com todas as suas conexões de entrada e saída. Perceba que o termo **unidade** se refere tanto às entradas da rede como também aos neurônios artificiais.

Essa simples ideia simples mostrou-se capaz de melhorar o desempenho de redes neurais profundas em muitos casos testados empiricamente. Em virtude disso é parte integrante do treinamento redes profundas.



Dropout

Neste sentido, o **dropout** é uma técnica é utilizada para melhorar o desempenho das CNNs.

Usualmente nas CNNs o **dropout** é realizado após cada camada totalmente conectada na rede.

- Isso ajuda a regularizar a rede e evitar *overfitting* ⇒ melhora o desempenho de generalização.

Observação: é importante ajustar o hiperparâmetro de **dropout**, que é a probabilidade de **dropout**, para encontrar o valor ideal para sua rede e conjunto de dados. O **dropout** pode ser sensível à arquitetura da rede e à ordem em que as camadas são conectadas, então deve-se experimentar diferentes arquiteturas e conexões de camadas testar o desempenho. Finalmente, é importante observar que o **dropout** pode às vezes ser omitido em pequenas CNNs ou redes superficiais, pois os benefícios podem não ser significativos em comparação com a sobrecarga computacional adicionada.

Batch Normalization

A normalização em lote (*batch normalization*) tem como objetivo normalizar (ou padronizar) não apenas os dados de entrada, mas também valores de neurônios individuais dentro das camadas intermediárias ocultas ou da camada final da rede através da normalização das entradas das camadas, executando recentralização e redimensionamento.

Essa normalização dos valores de neurônios produz então um treinamento mais consistente, rápido, estável atenuando problemas de explosão ou desaparecimento nos gradientes.

Matematicamente, a normalização de um neurônio é feita recentralizando por uma média e redimensionando usando um desvio padrão.

Essa técnica também tem um efeito de regularização que ajuda na mitigação do *overfitting*.

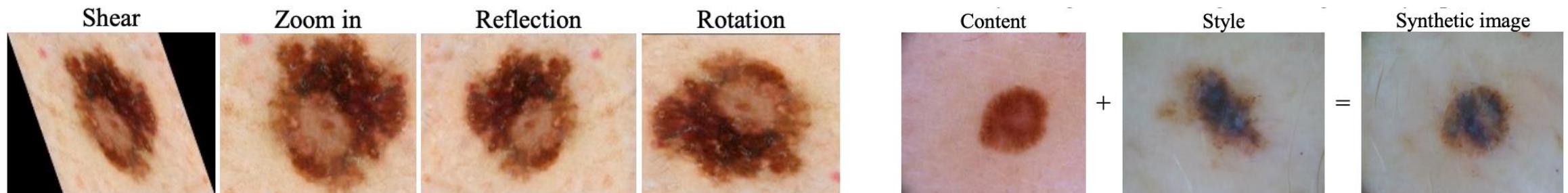
Data augmentation

Talvez essa relação não seja clara, mas o treinamento com poucos dados favorece a ocorrência do *overfitting*.

Uma forma de melhorar a quantidade de dados é aumentar artificialmente o conjunto de dados, essa técnica é chamada de *data augmentation*.

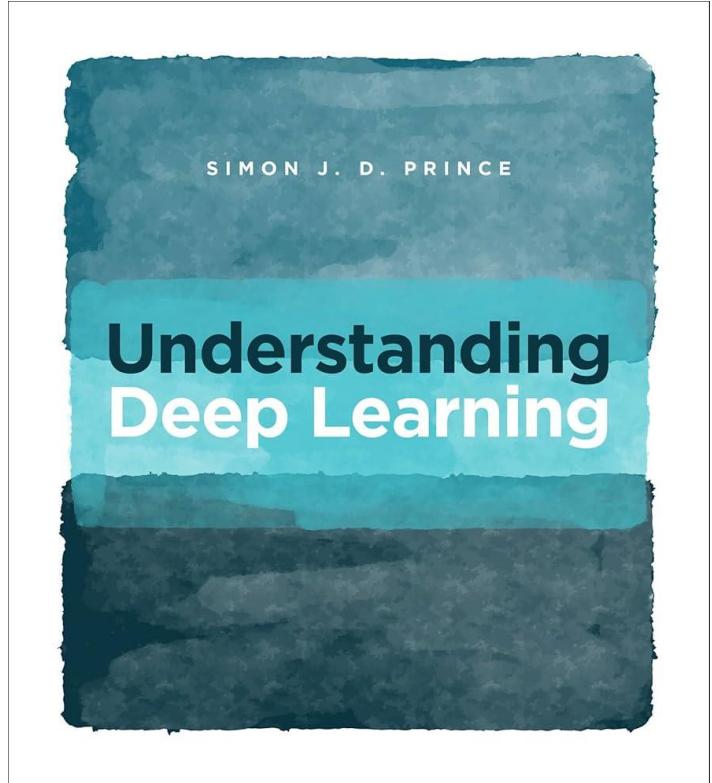
Há várias formas de se fazer o aumento do conjunto de dados.

- Transformações de imagem, como: rotação, corte, zoom
- Métodos baseados em histograma
- Generative Adversarial Networks
- ...

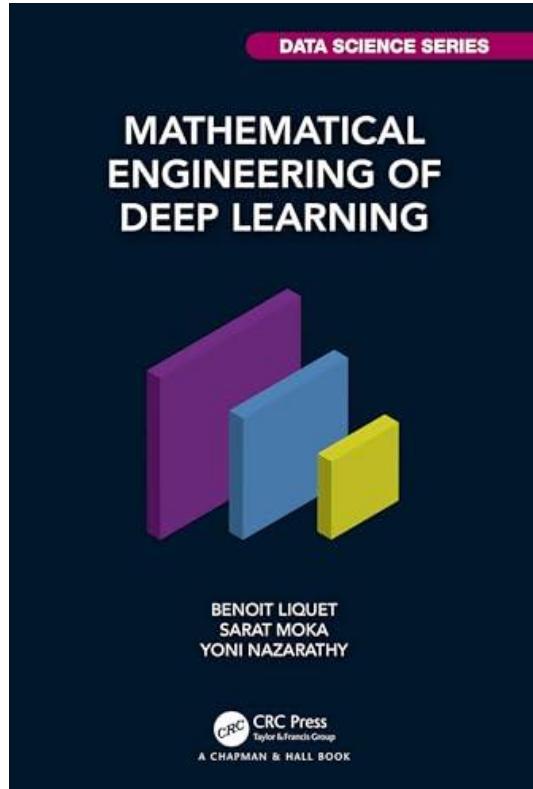


Referência: A. Mikołajczyk, M. Grochowski. Data augmentation for improving deep learning in image classification problem. *2018 International Interdisciplinary PhD Workshop (IIPhDW)*, Świnoujście, Poland, 117-122, 2018.

Referências



Understanding Deep Learning
Simon J. D. Prince
<https://udlbook.github.io/udlbook/>



Mathematical Engineering of Deep Learning
Benoit Liquet, Sarat Moka e Yoni Nazarathy
<https://deeplearningmath.org>

Por hoje é só!

FIM!

Obrigado pela atenção!