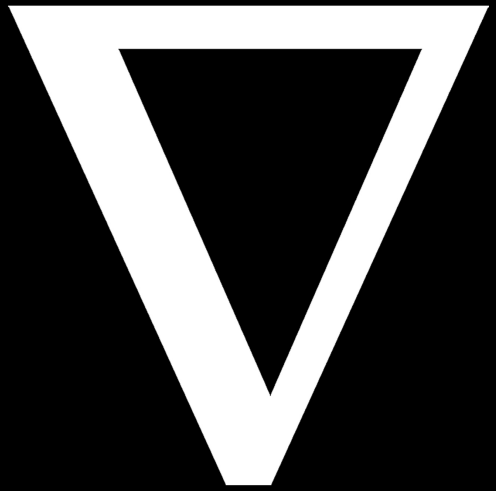


data

Redes Neurais Convolucionais

Luiz Felipe Diniz Costa
/in/lfelipediniz

Presença

- Linktree: Presente na bio do nosso instagram
- Presença ficará disponível até 1 hora antes da próxima aula
- É necessário 70% de presença para obter o certificado

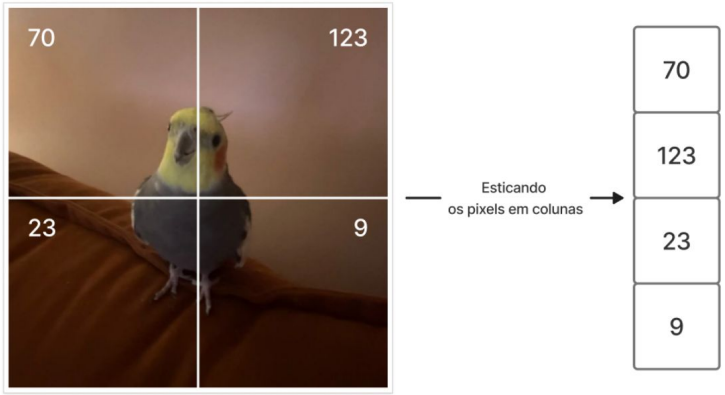
Presença

Introdução e Motivação

O que significa achatar uma imagem?

Quando transformamos pixels em vetores

- Redes tradicionais tratam imagens como vetores
- A estrutura espacial (vizinho de pixel) é ignorada



Por que isso é um problema?

Redes fully-connected perdem padrões locais

- Flattening destrói a noção de vizinhança entre pixels
- Rede precisa reaprender padrões simples (ex: bordas) do zero
- Resultado: mais parâmetros, mais custo, menos eficiência
- Perda da estrutura que deveria ser preservada

Como imagens realmente são representadas

A estrutura tridimensional de uma imagem

- Tensores tridimensionais
  - Altura (H)
  - Largura (W)
  - Canais (C)
- Cada canal carrega uma camada de informação separada
  - C = 1 (tons de cinza)
  - C = 3 (RGB)
  - C > 3 (visão avançada)

A solução: redes convolucionais (CNNs)

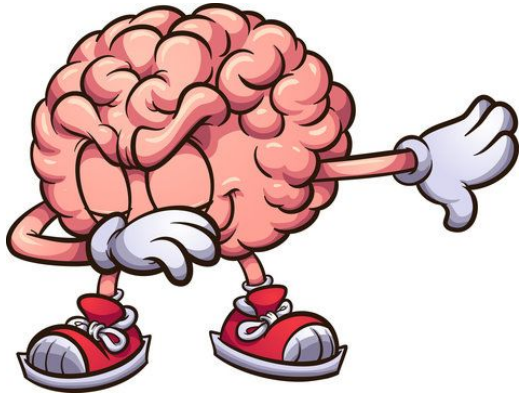
Preservando a estrutura da imagem

- CNNs trabalham com tensores diretamente
- Aplicam convoluções, pooling, normalização (tudo espacial)
- Aprendem filtros automaticamente para detectar padrões locais
- Menos parâmetros, mais rápido, mais eficaz

O que vem a seguir?

Explorando cada bloco de uma CNN na prática

- Intuição da convolução com exemplo numérico
- O que são filtros (kernels) e como funcionam
- Visualização de feature maps
- Motivação para padding, stride, pooling e batch norm

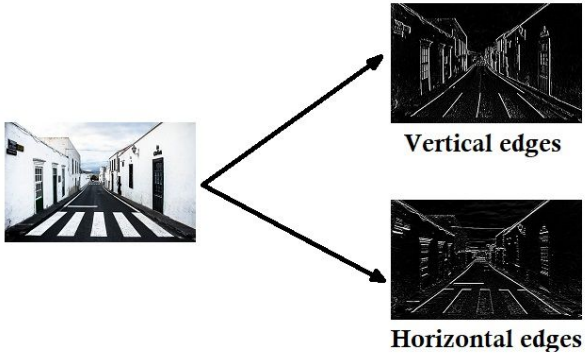


Intuição

O que queremos detectar em imagens?

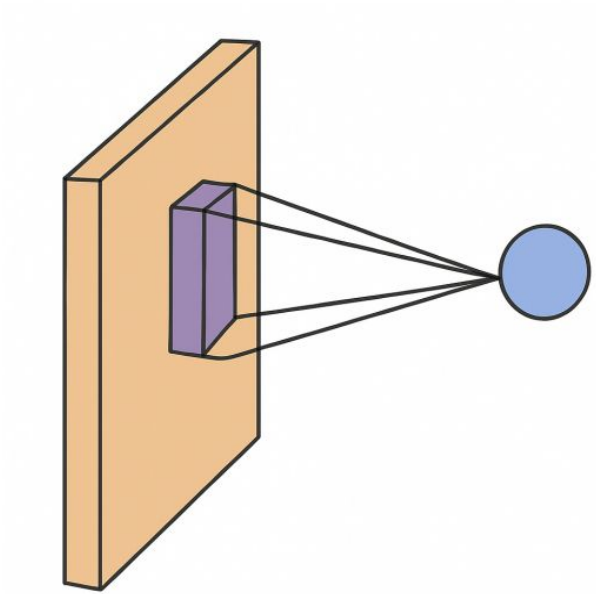
Bordas, texturas, padrões locais

- Para interpretar uma imagem, precisamos localizar padrões visuais
- Exemplos: uma borda, um canto, uma curva
- Redes tradicionais não conseguem fazer isso bem
- CNNs aprendem a detectar esses padrões automaticamente



A ideia de uma janela deslizante

Como observar pequenos pedaços da imagem

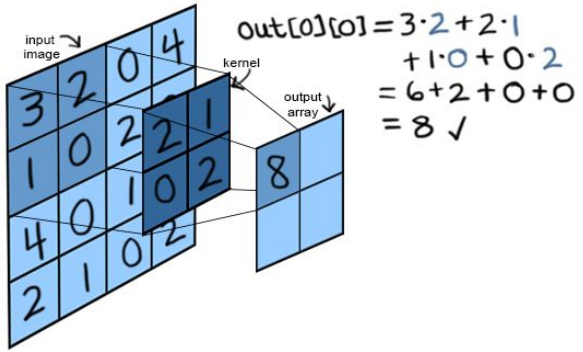


- Em vez de olhar tudo de uma vez, olhamos uma região por vez
- Essa região é chamada de **Receptive Field**
- Conseguimos capturar detalhes em cada região

O que é um filtro (kernel)?

Pequena matriz de pesos aprendida pela rede

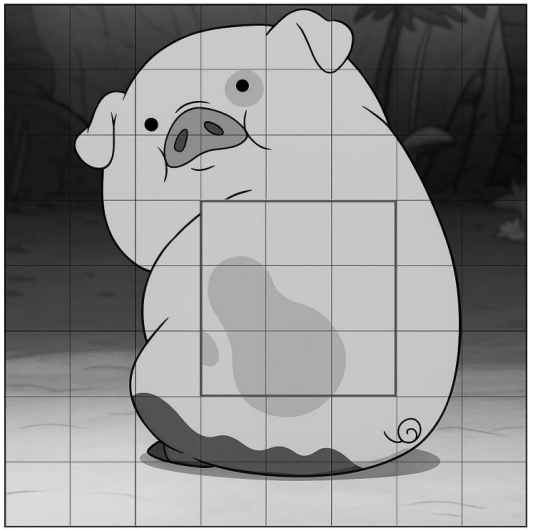
- O filtro é como uma lupa que destaca certos padrões
- Cada filtro tem pesos fixos que são multiplicados pela imagem
- Ele percorre a imagem inteiro, sempre com os mesmos pesos
- Exemplo: filtro 3×3 que detecta bordas verticais



Exemplo visual: imagem do porquinho Waddles

Aplicando um filtro a uma imagem real

- Imagem convertida para tons de cinza
- Sobrepomos uma grade para destacar uma região 3×3
- Essa região será usada no cálculo da convolução



Matriz da imagem: valores reais da região 3×3

Intensidade dos pixels em escala de cinza

- Cada valor representa um pixel
- Usaremos essa matriz no próximo passo

$$X = \begin{bmatrix} 189.59 & 194.95 & 190.90 \\ 172.70 & 185.25 & 193.31 \\ 175.50 & 168.63 & 186.22 \end{bmatrix}$$

Aplicando um filtro de borda vertical

Multiplicação ponto a ponto

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

Multiplicamos cada elemento de X pelo correspondente em F e depois somamos todos os valores

Resultado: o valor da convolução

Um número que representa uma ativação

- Resultado: **32.64**
- Significa: há uma borda vertical nesta região
- Esse valor ocupa uma posição no novo mapa de ativação
- Ao repetir esse processo por toda a imagem → **feature map**

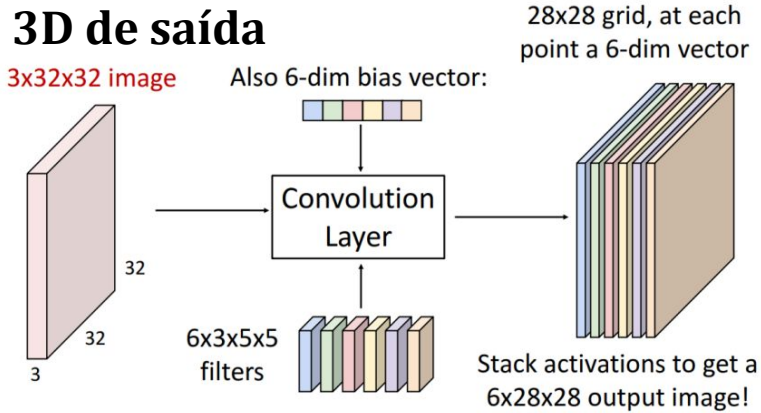
Arquitetura Convolucional

O que acontece depois da convolução?

Cada filtro gera um mapa de ativação

- Cada filtro convolucional analisa a imagem e produz um mapa
- Usamos vários filtros → geramos vários mapas (um por filtro)
- Esses mapas empilhados formam um **volume 3D de saída**

Mostra aplicação de 6 filtros 3×5×5 na primeira camada convolucional



O que a rede aprende com isso?

De bordas simples a formas complexas

- Camadas iniciais aprendem padrões locais (bordas, texturas)
- Camadas intermediárias e finais combinam essas informações
- A rede passa a “reconhecer” partes e estruturas de objetos
- O empilhamento de camadas cria representações hierárquicas

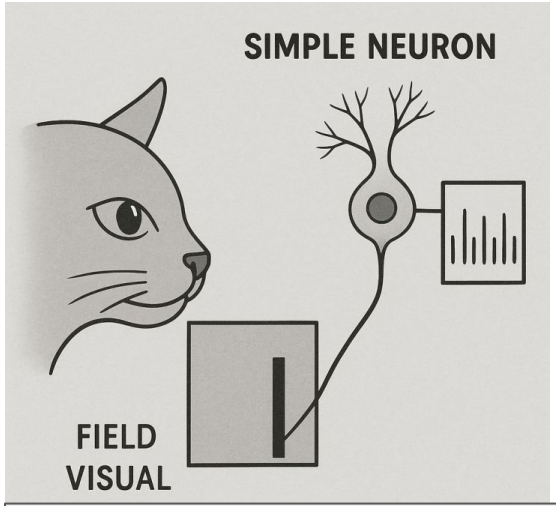
Um modelo inspirado no cérebro

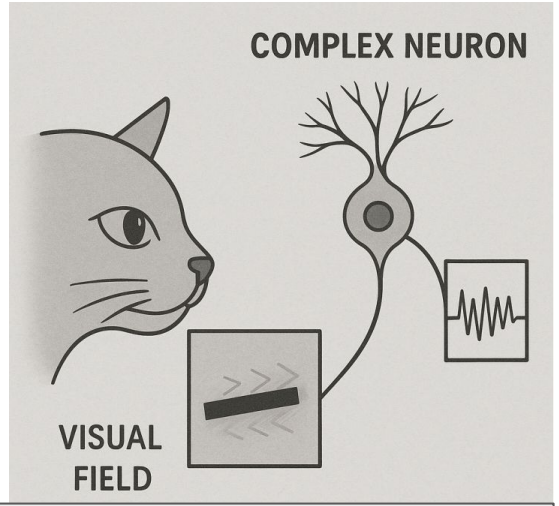
Neurônios simples e complexos (Hubel e Wiesel)

- Experimentos em gatos: neurônios simples reagem a linhas específicas
- Neurônios complexos reagem a combinações e movimentos
- CNNs imitam essa hierarquia natural da visão biológica

Um modelo inspirado no cérebro

Neurônios simples e complexos (Hubel e Wiesel)





Neurônio simples ativado por linha / Neurônio complexo por movimento

Como as camadas se Organizam?

Estrutura básica das CNNs

- Entrada → Convolução → ReLU → Pooling → nova camada
- Cada nova camada recebe um volume de mapas da anterior
- Quanto mais profunda a camada, mais abstrata é a informação
- Essa estrutura modular é a base de arquiteturas como LeNet, VGG, ResNet

Conclusão

Uma arquitetura com propósito claro

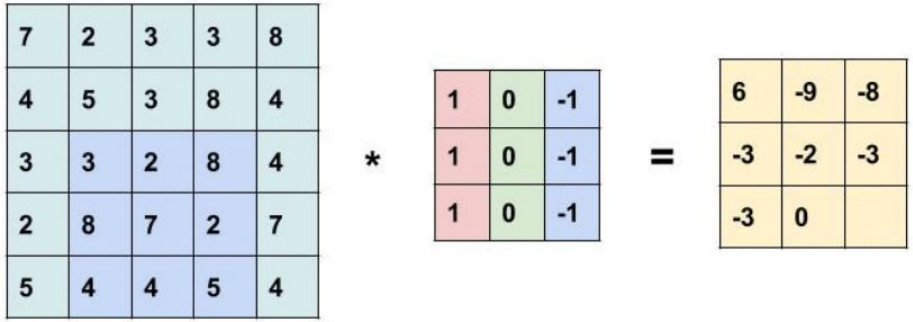
- CNNs não são apenas operadores isolados
- São camadas empilhadas com lógica de aprendizado hierárquico
- Próximo passo: como garantir que as camadas preservem o tamanho e vejam mais contexto? Essa pergunta nos leva para a próxima seção!

Padding e Receptive Field

O problema nas bordas

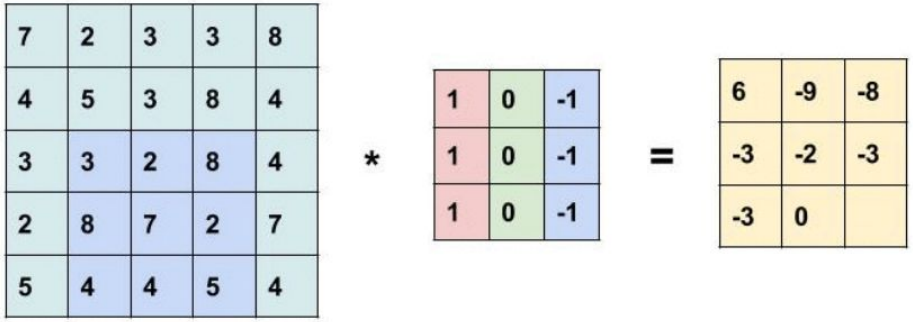
Por que a imagem “encolhe” sem padding

- Ao aplicar um kernel 3×3, não conseguimos usá-lo nas bordas
- Cada camada sem padding reduz o tamanho da imagem
- Parte da informação da imagem original é descartada



O problema nas bordas

Por que a imagem “encolhe” sem padding



$$W_{\text{out}} = W_{\text{in}} - K + 1$$

A solução: usar padding

Adicionando uma moldura de zeros

- Adicionamos zeros ao redor da imagem (padding de 1 pixel)
- Isso permite que o kernel também atue nas borda
- A saída mantém o mesmo tamanho da entrada

0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

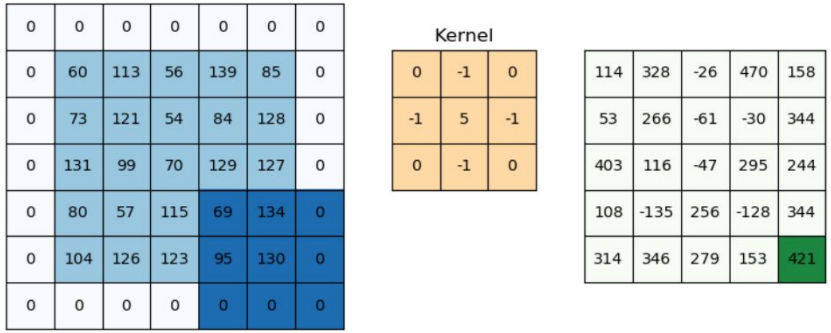
Kernel

0	-1	0
-1	5	-1
0	-1	0

114	328	-26	470	158
53	266	-61	-30	344
403	116	-47	295	244
108	-135	256	-128	344
314	346	279	153	421

Cálculo do tamanho de saída com padding

Como manter as dimensões iguais



0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

0	-1	0
-1	5	-1
0	-1	0

114	328	-26	470	158
53	266	-61	-30	344
403	116	-47	295	244
108	-135	256	-128	344
314	346	279	153	421

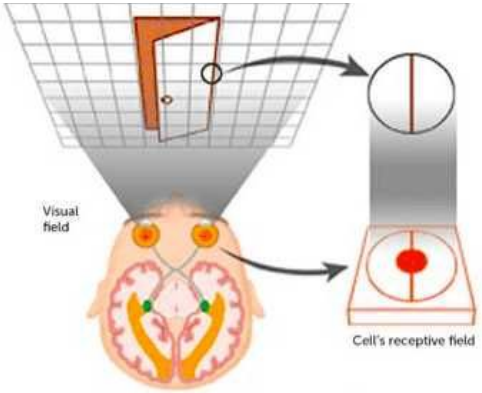
$$W_{\text{out}} = \frac{W - K + 2P}{S} + 1$$

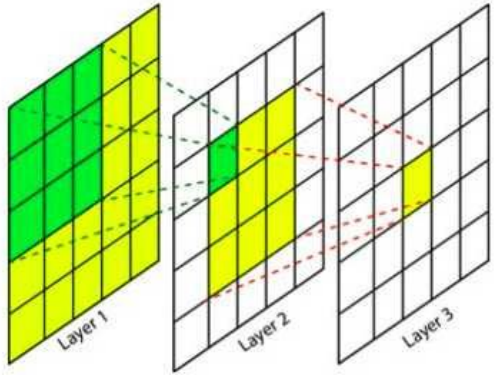
- Resultado: P = 1 → **same padding**

O que é o Receptive Field?

A área da imagem que influencia uma ativação

- Cada neurônio na saída é influenciado por uma região da entrada
- Depende de kernel, stride, padding e profundidade
- Quanto mais camadas, maior pode ser o Receptive Field





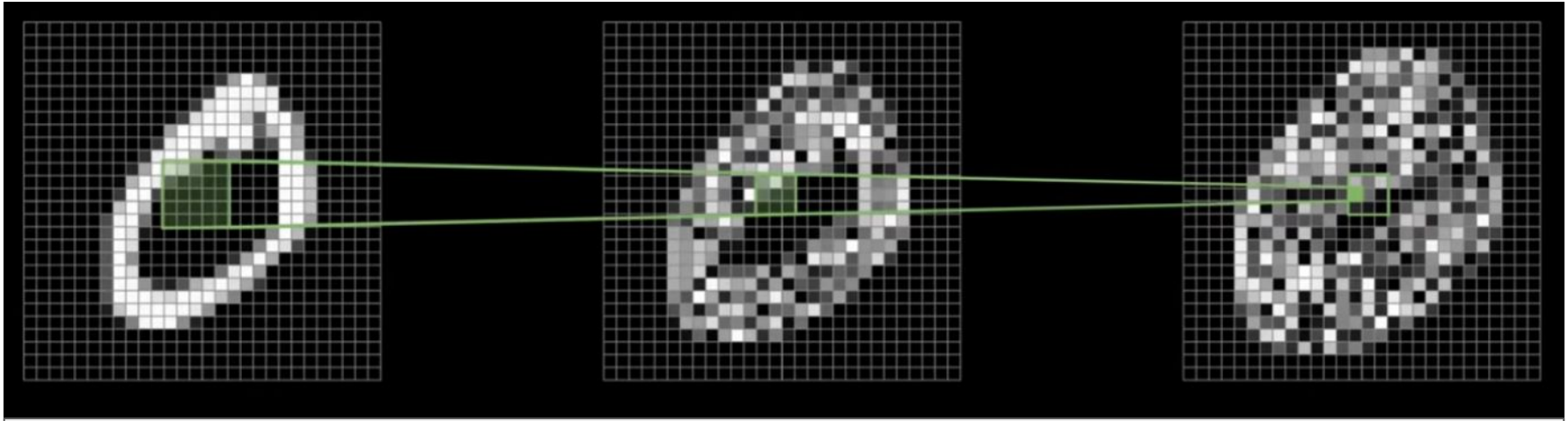
Como o Receptive Field cresce?

Empilhar camadas aumenta a visão da rede

- A primeira camada vê diretamente a entrada
- A segunda camada vê a saída da primeira (que já é um "bloco" da entrada)
- A terceira vê a segunda... e assim por diante
- O Receptive Field cresce cumulativamente a cada camada

Como o Receptive Field cresce?

Empilhar camadas aumenta a visão da rede



Crescimento do Receptive Field em profundidade

Como o Receptive Field cresce?

Empilhar camadas aumenta a visão da rede

Como calcular esse crescimento, camada por camada?

Cálculo do Receptive Field

Fórmulas gerais para qualquer arquitetura

$$j_\ell = j_{\ell-1} \cdot s_\ell$$

$$RF_\ell = RF_{\ell-1} + (k_\ell - 1) \cdot j_{\ell-1}$$

espaçamento entre pontos da imagem original

tamanho do campo receptivo na ℓ-ésima camada

Exemplo prático: 3 camadas 3×3 com stride = 1

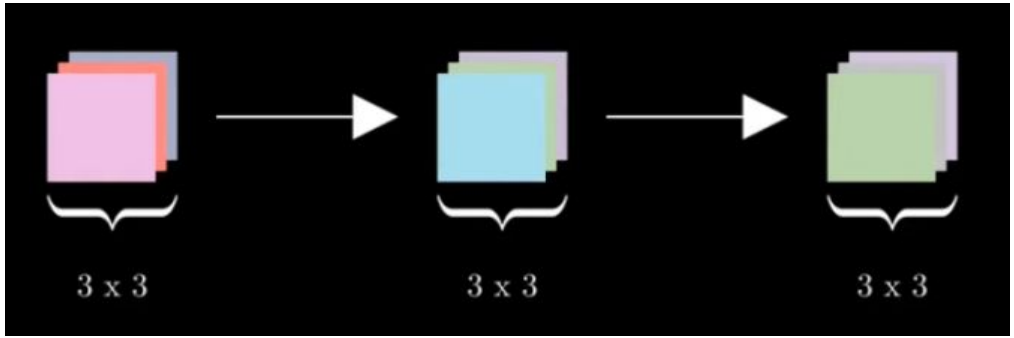
Mostrando o crescimento real do campo receptivo

Camada 1: RF = 3

Camada 2: RF = 5

Camada 3: RF = 7

Cada camada acrescenta 2
(por conta do kernel 3×3)



Fórmula Geral do Receptive Field

Forma fechada para qualquer arquitetura convolucional

$$RF_{\text{final}} = 1 + \sum_{l=1}^{L} (k_l - 1) \cdot \prod_{i=1}^{l-1} s_i$$

k_l: tamanho do kernel da camada l

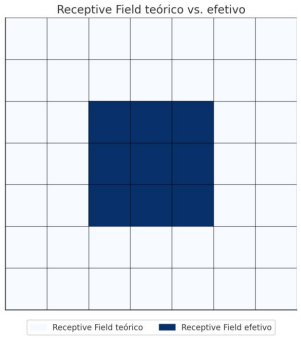
s_i: stride da camada i

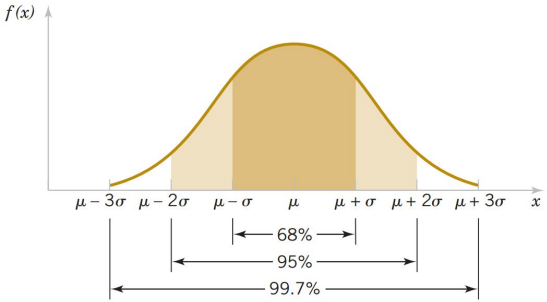
- Lida com qualquer sequência de convoluções e poolings
- Útil para calcular RF de redes reais como ResNet ou YOLO

Teórico vs. Efetivo

Nem todos os pixels do campo teórico contribuem igualmente

- O campo **teórico** é o tamanho total da área de influência
- Mas na prática, **os gradientes se concentram no centro**
- Isso define o **campo receptivo efetivo**





Muitas vezes assume forma parecida com uma distribuição Gaussiana

Stride e
Convoluções Especiais

Controle de deslocamento com Stride

Ao aplicar um kernel sobre a imagem, controlamos o quanto ele se move com o hiperparâmetro **stride** S.

Em vez de deslocar o kernel uma posição por vez (S=1), podemos avançar de 2 em 2, 3 em 3, etc.

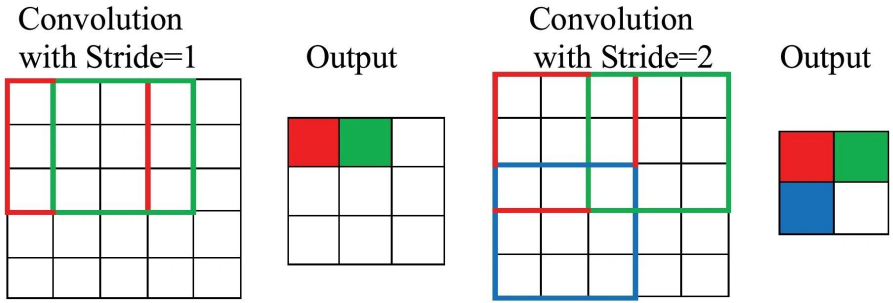
Isso afeta diretamente:

- O **tamanho da saída**
- A **resolução do mapa de características**
- O **custo computacional**

Comparando Stride 1 e Stride 2

Controlando o “passo” da convolução

- Stride 1 gera saída maior: mais detalhada, mais custosa.
- Stride 2 pula pixels → saída menor, mais econômica.



Como stride afeta o Receptiva Field?

Crescimento mais rápido com stride > 1

- Strides maiores aumentam o salto dos neurônios
- Faz o Receptive Field crescer mais rapidamente
- Também reduz dimensionalidade espacial da saída
- Exemplo: stride = 2 dobra o salto a cada camada

Variedades de convolução

- Além da convolução padrão, existem versões especiais que ajudam a:
  - Aumentar o Receptive Field
  - Reduzir o número de parâmetros
  - Controlar melhor a complexidade da rede

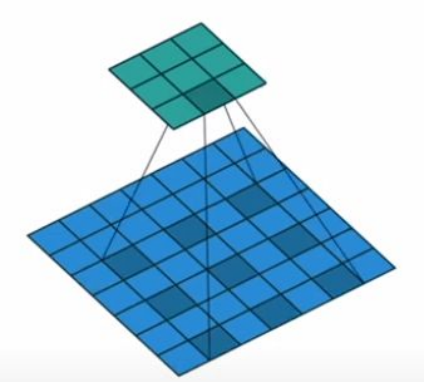
Dilated Convolutions

Expandindo o alcance sem aumentar parâmetros

- A dilatação **insere espaçamentos internos** no kernel
- Permite cobrir uma área maior com o **mesmo número de pesos**
- Tamanho efetivo cresce:

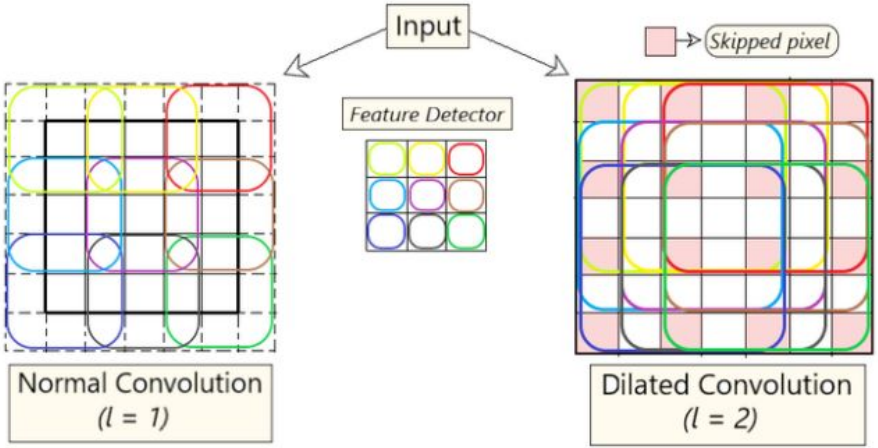
$$K_{\text{eff}} = K + (K-1) \cdot (D-1)$$

$$K = 3, D = 2 \Rightarrow K_{\text{eff}} = 5$$



Dilated Convolutions

Expandindo o alcance sem aumentar parâmetros

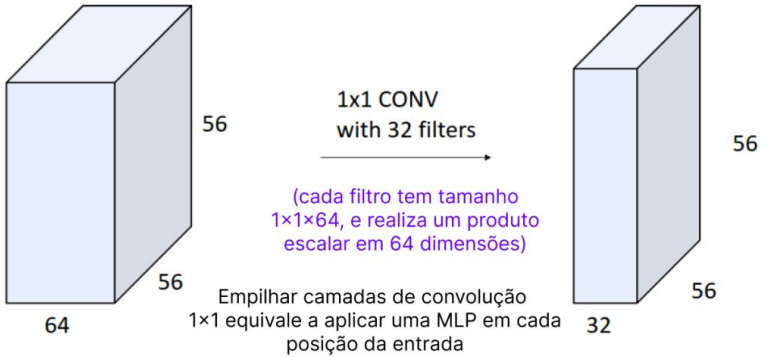


Muito útil em segmentação semântica e tarefas que exigem contexto amplo **sem perder resolução**

Convoluções 1×1

Aplicações: Inception, ResNet, MobileNet

- Atua ponto a ponto no mapa de ativação.
- Permite **projetar ou combinar canais** sem alterar resolução espacial.
- Serve como adaptador entre blocos convolucionais.



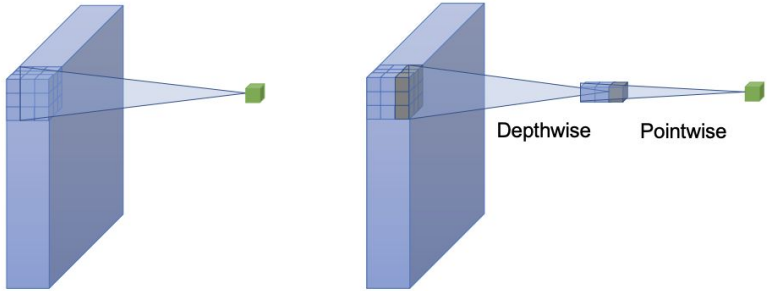
Depthwise separable convolutions

- Separa o processo em duas etapas:
  - **Depthwise:** convolução K×K em cada canal individualmente
  - **Pointwise:** convolução 1×1 para combinar os canais
- Reduz drasticamente o custo computacional

Comparação de parâmetros:

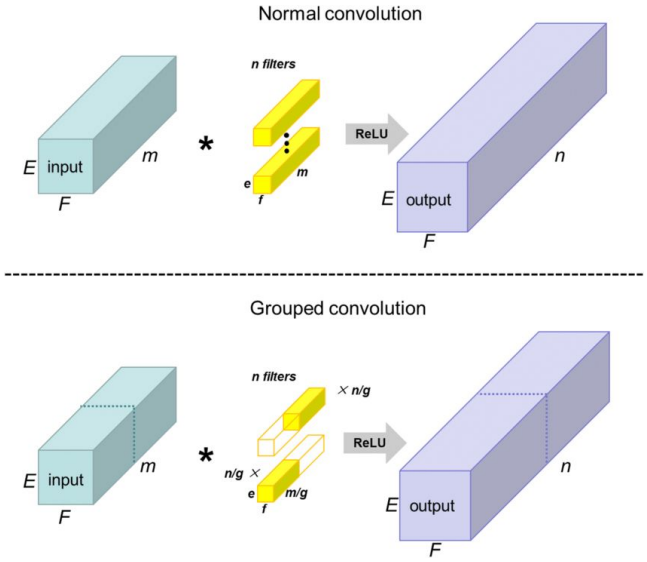
$$\text{Normal: } K^2 \cdot C_{\text{in}} \cdot C_{\text{out}}$$

$$\text{Separable: } K^2 \cdot C_{\text{in}} + C_{\text{in}} \cdot C_{\text{out}}$$



Grouped Convolutions

- **Dividimos os canais da entrada em G grupos**
- Cada grupo é processado por um subconjunto de filtros
- Isso reduz o número total de multiplicações e parâmetros
  É como ter **várias mini-convoluções paralelas**



Com $G = C_{\text{in}}$, ela vira uma **depthwise convolution**

Pooling

Por que usar pooling?

Reduzindo resolução sem perder padrões

- Após convoluções, os mapas de ativação ainda têm resolução alta

- **Pooling reduz a resolução espacial**, mantendo os valores mais representativos. Ajudando em:
  - Reduzir o custo computacional
  - Tornar a rede mais robusta a pequenas variações

Intuição

- Imagine que tenhamos uma imagem HD

Intuição

- Temos 3 canais de cores
- 1280 pixels de largura
- 720 pixels de altura

Intuição

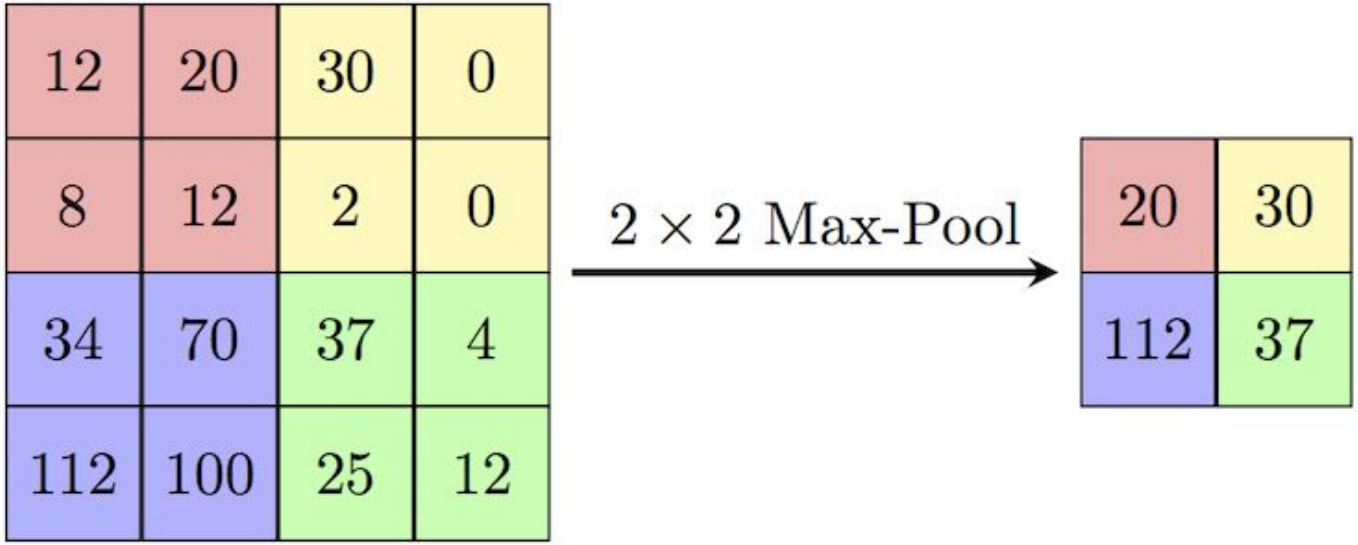
- Temos 3 canais de cores
- 1280 pixels de largura
- 720 pixels de altura
- 3_x1280_x720 = 2.764.800 valores de pixel!

Intuição

- Problema: muitos valores de pixel para analisar

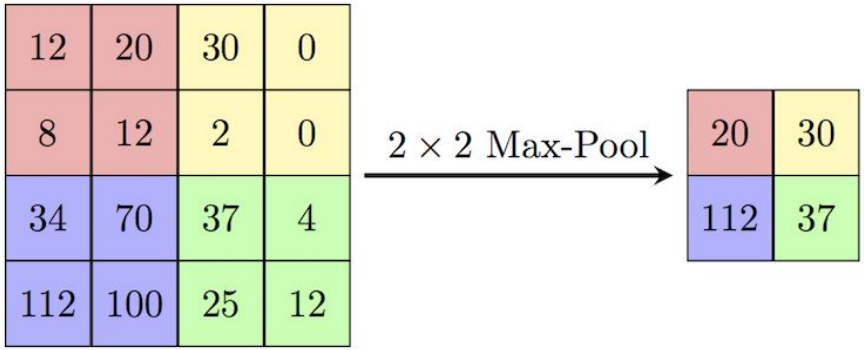
Intuição

- Solução: Pegar apenas os mais significativos para a nossa tarefa



Tipos de Pooling

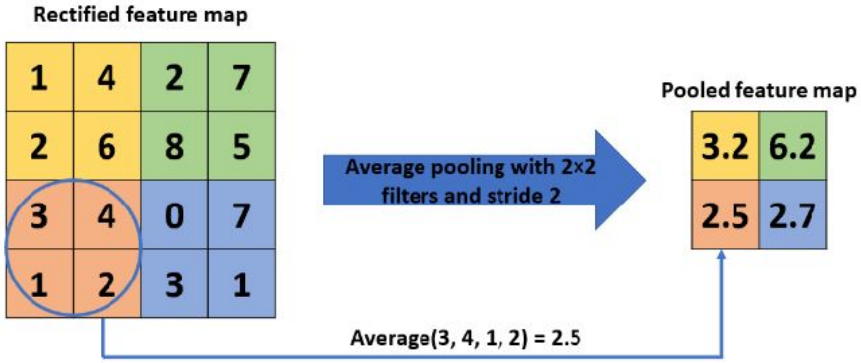
- Max-Pooling



- Max pooling mantém as ativações mais fortes
- Funciona como um "destilador de informações"

Tipos de Pooling

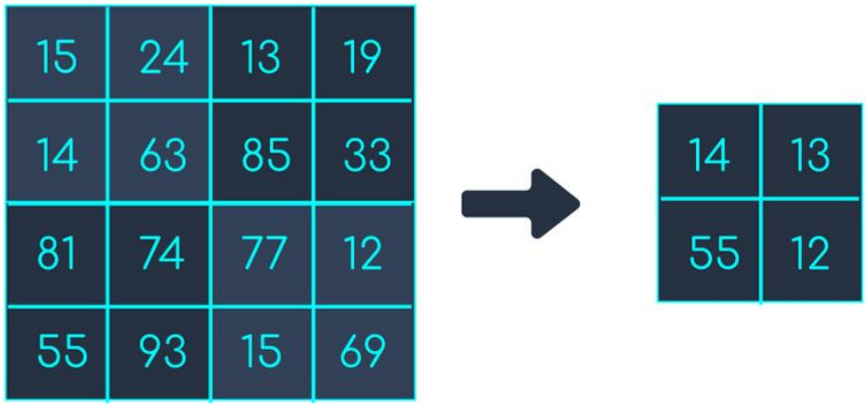
- Average-Pooling



- Similar ao max pooling, mas calcula a **média dos valores** de cada região
- Produz resultados mais suavizados

Tipos de Pooling

- Min-Pooling



- Mesmo processo do max/avg pooling, mas agora capturamos **o menor valor** de cada região
- Útil em situações específicas onde valores baixos são mais informativos (como detecção de áreas escuras)

Pooling vs Stride

Qual a diferença entre pooling e stride?

- Ambos reduzem resolução
- **Pooling:** faz uma operação estatística (ex: máximo ou média)
- **Stride:** é parte da convolução - a própria operação de extração é espaçada

OBS: Complementaridade (podem ser usados juntos ou separadamente)

Pooling e ReLU: combinando tudo

Pooling e Ativação (ReLU)

- **ReLU:** função não linear que zera valores negativos: $f(x) = \max(0, x)$
- Mantém o gradiente "vivo"
- Acelera o treinamento
- Produz ativações mais esparsas
- Pipeline Moderno:

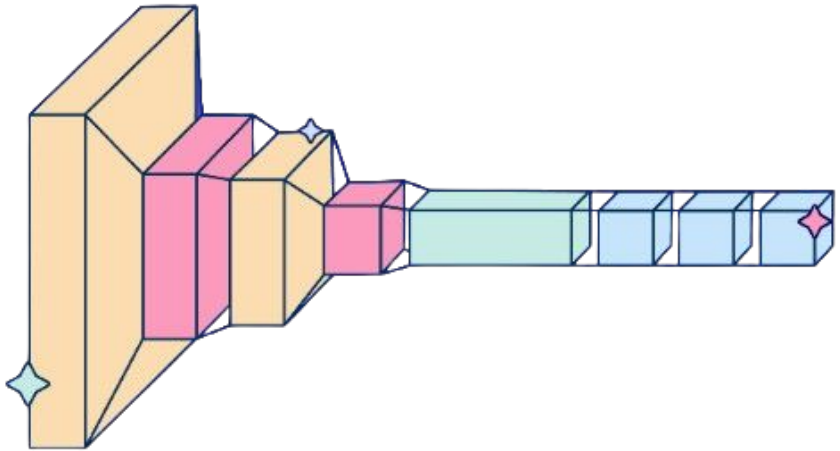
$$\text{Conv} \rightarrow \text{ReLU} \rightarrow \text{Pool} \rightarrow \text{ReLU}$$

OBS: Max Pooling também é não linear, mas atua espacialmente

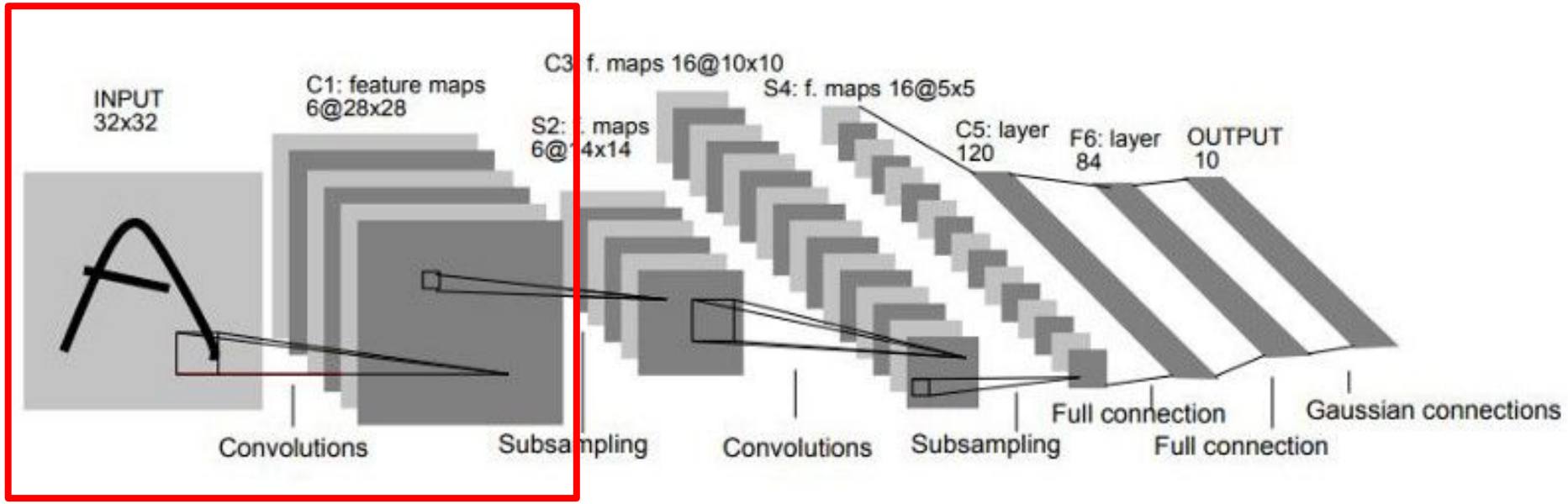
LeNet-5

O que é?

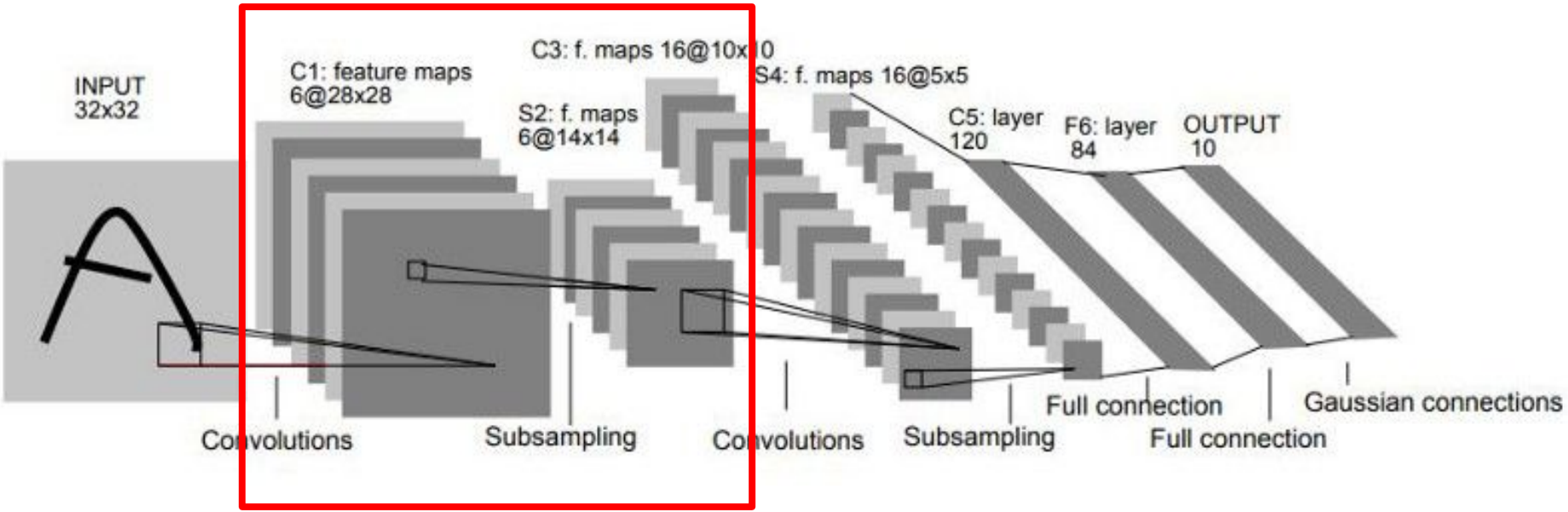
- Primeiro modelo funcional criado para ler dígitos escritos à mão (MNIST)
- Arquitetura simples, mas eficiente



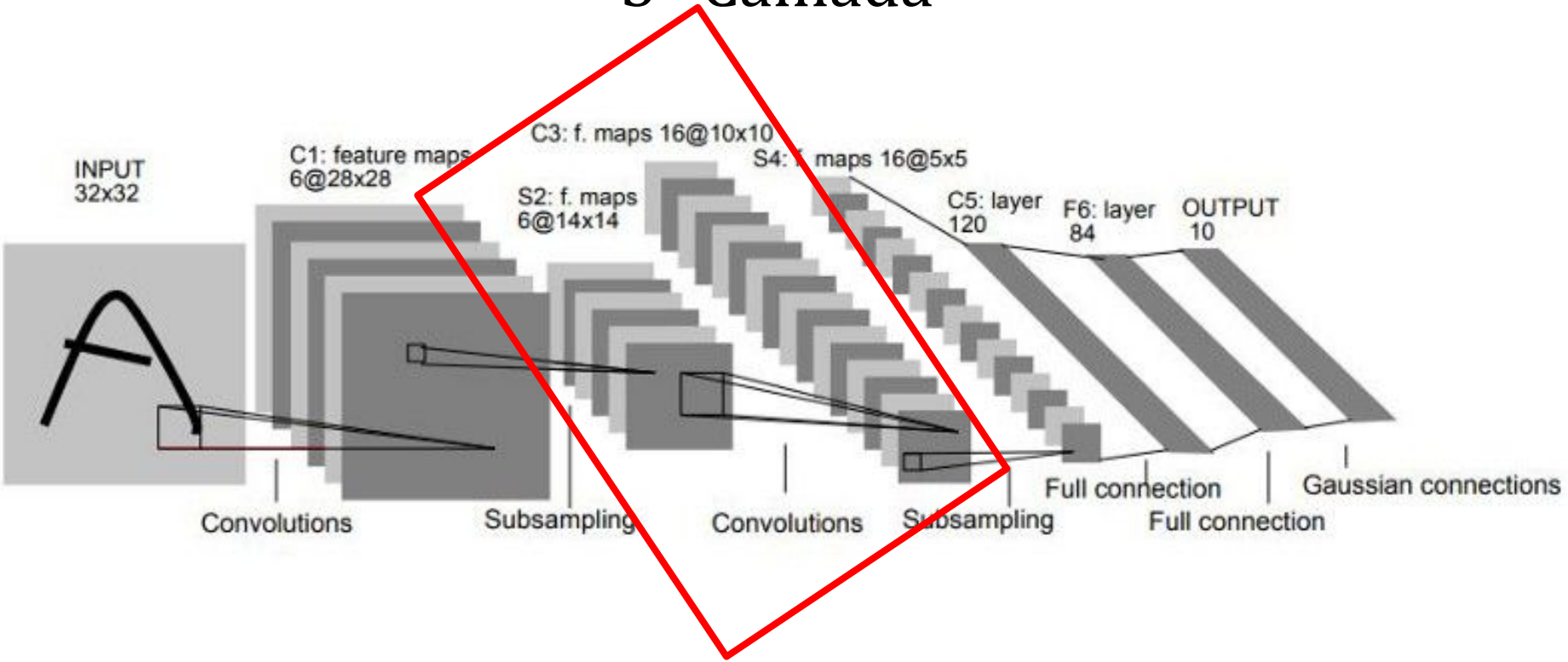
1ª Camada


INPUT
32x32
C1: feature maps
6@28x28
S2: f. maps
6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer
120
F6: layer
84
OUTPUT
10
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection
Gaussian connections

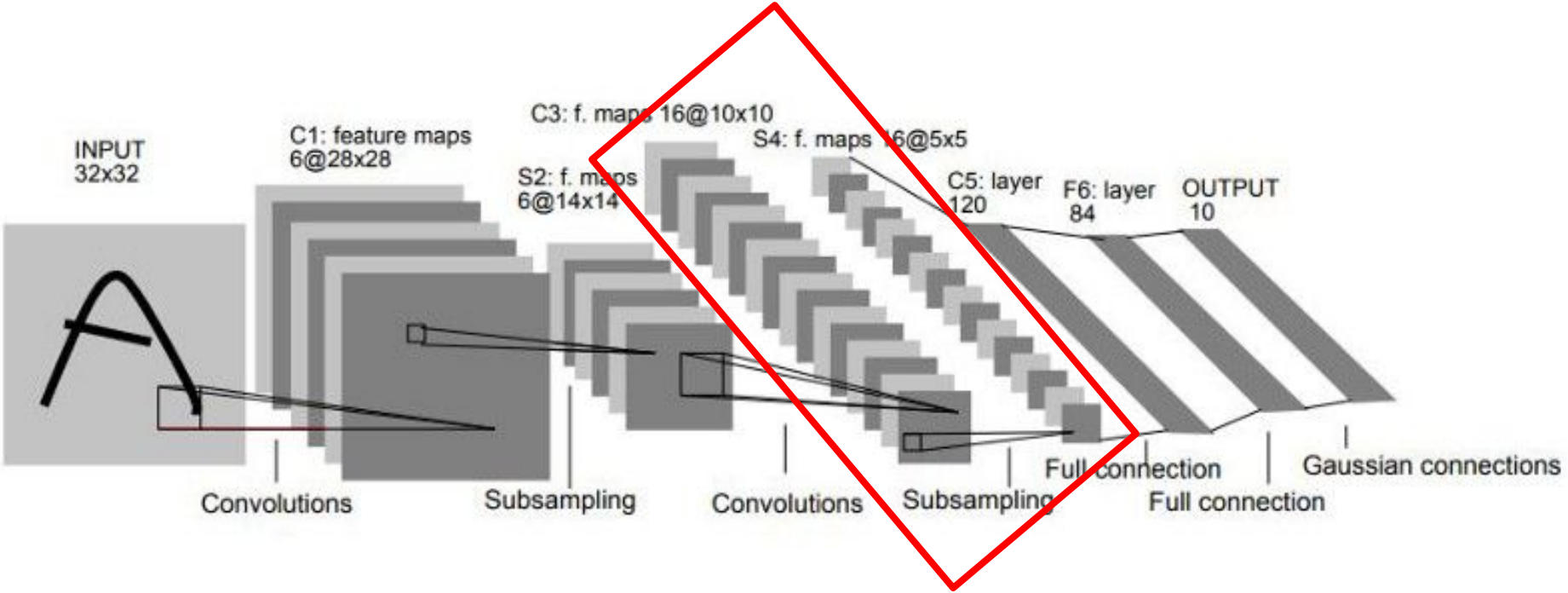
2ª Camada


INPUT
32x32
C1: feature maps
6@28x28
S2: f. maps
6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer
120
F6: layer
84
OUTPUT
10
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection
Gaussian connections

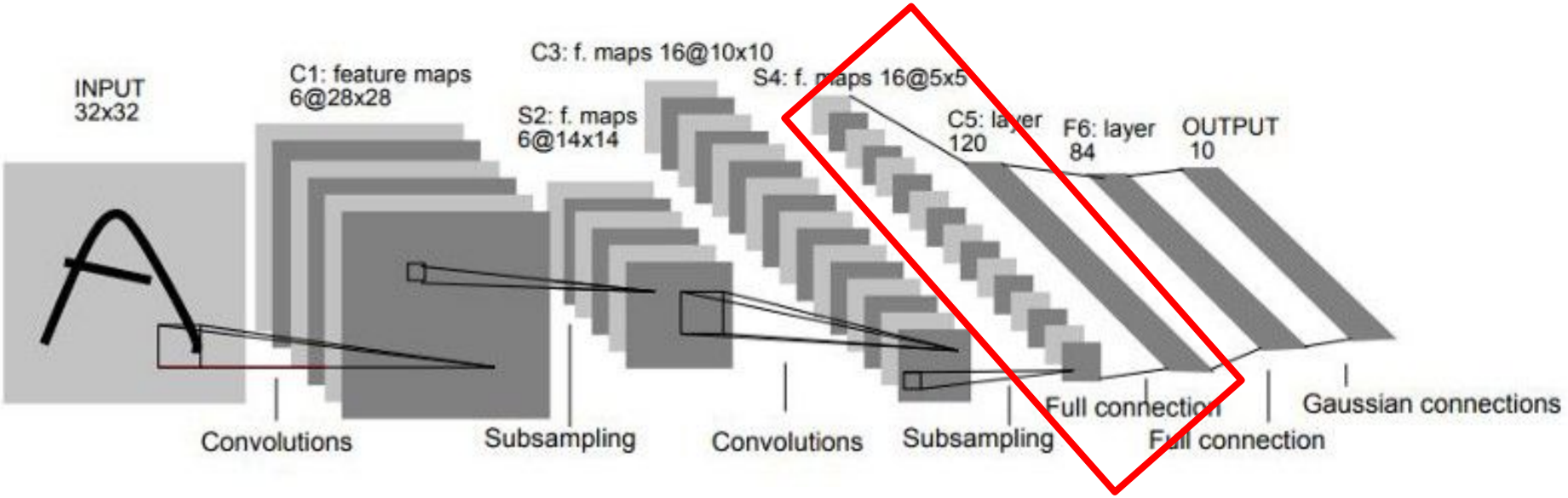
3ª Camada


INPUT
32x32
C1: feature maps
6@28x28
S2: f. maps
6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer
120
F6: layer
84
OUTPUT
10
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection
Gaussian connections

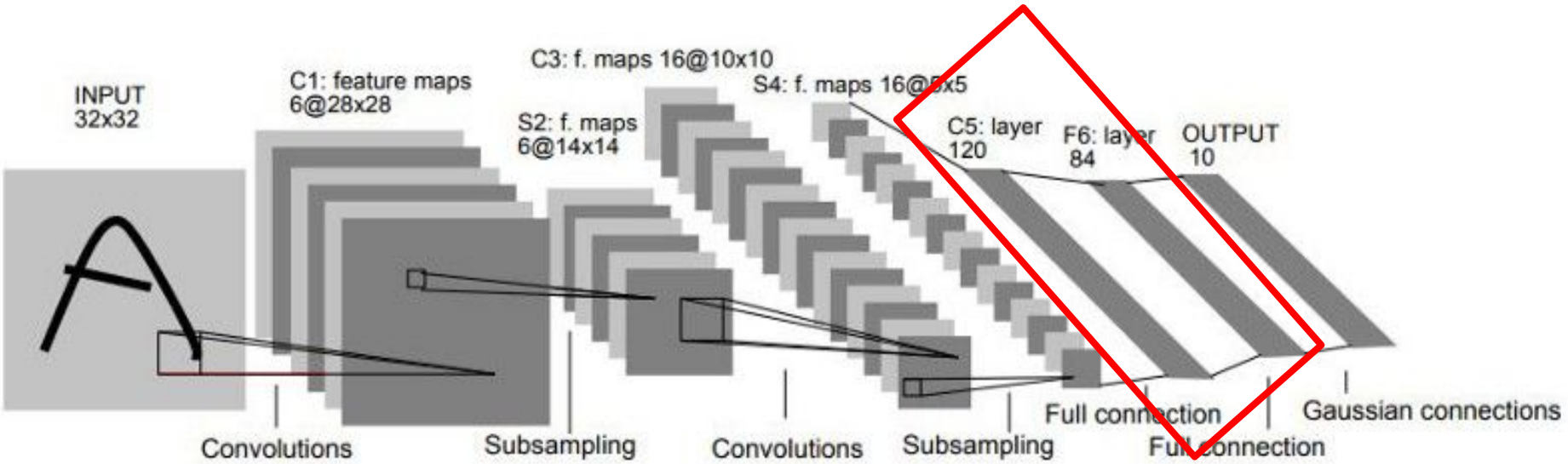
4ª Camada


INPUT
32x32
C1: feature maps
6@28x28
S2: f. maps
6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer
120
F6: layer
84
OUTPUT
10
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection
Gaussian connections

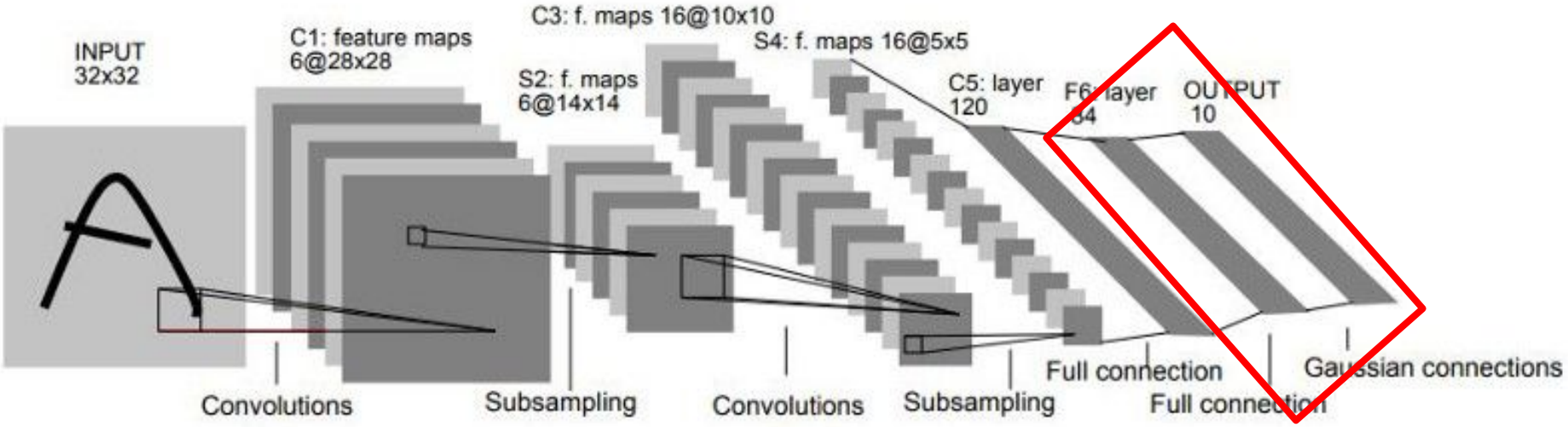
5ª Camada


INPUT
32x32
C1: feature maps
6@28x28
S2: f. maps
6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer
120
F6: layer
84
OUTPUT
10
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection
Gaussian connections

6ª Camada


INPUT
32x32
C1: feature maps
6@28x28
S2: f. maps
6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer
120
F6: layer
84
OUTPUT
10
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection
Gaussian connections

7ª Camada


INPUT
32x32
C1: feature maps
6@28x28
S2: f. maps
6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer
120
F6: layer
84
OUTPUT
10
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection
Gaussian connections

Parâmetros e eficiência

Por que LeNet-5 era tão poderosa?

- Usa menos parâmetros que redes fully-connected
- Tira proveito da **localidade** (kernels pequenos)
- Usa compartilhamento de pesos
- Primeiro grande exemplo de *feature hierarchy*

LeNet hoje: base para muitas arquiteturas

A base das CNNs modernas

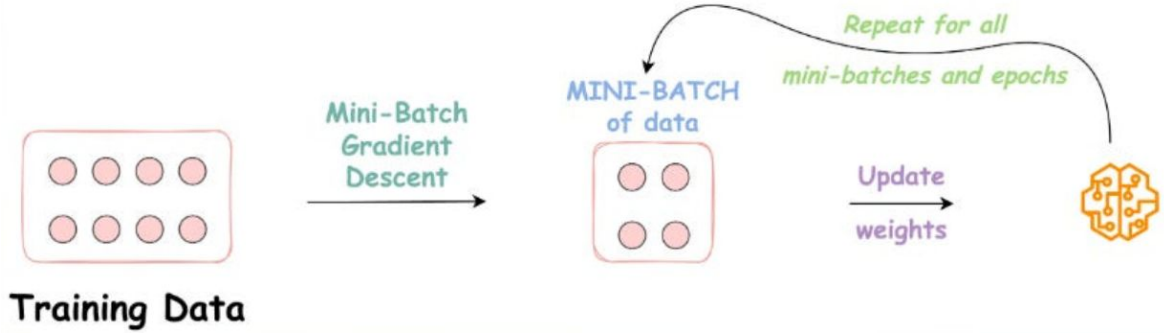
- Embora antiga, a LeNet introduziu:
  - Padrão Conv → ReLU → Pool
  - Uso de filtros com stride
  - Feature maps hierárquicos
- Arquiteturas modernas (ResNet, AlexNet, VGG) evoluem esse modelo

Batch Normalization

O que é um mini-batch?

O lote de exemplos usado em cada passo do treinamento

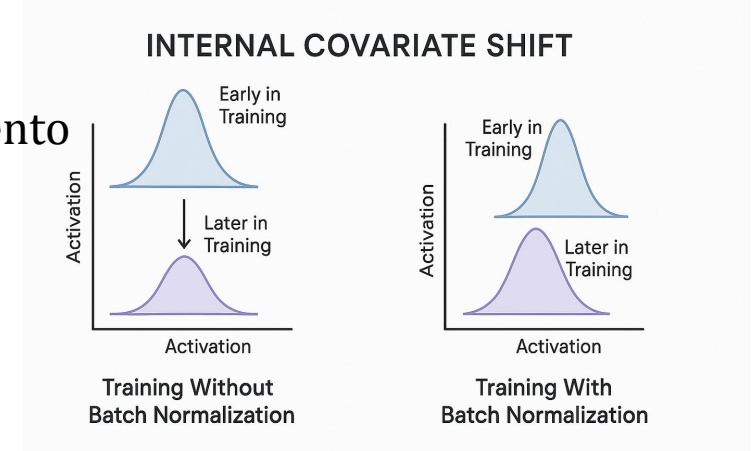
- Durante o treinamento, não usamos todos os dados de uma vez
- Em vez disso, dividimos o dataset em pequenos lotes chamados **mini-batches**
- Cada forward/backward pass ocorre sobre um mini-batch



O problema da instabilidade nas ativações

O que acontece quando empilhamos muitas camadas?

- Em redes profundas, as ativações mudam de distribuição ao longo do tempo
- Obriga as próximas camadas a reajustarem constantemente
- Isso causa instabilidade e lentidão no treinamento
- Chamamos isso de **internal covariate shift**



A solução: Batch Normalization

Normalizando as ativações em cada mini-batch

- BatchNorm corrige o problema ao normalizar as ativações
- É aplicada **antes da função de ativação**
- Atua canal por canal em redes convolucionais
- Deixa os valores com **média 0 e variância 1** no mini-batch

Etapa 1: média e variância

Cálculo das estatísticas do mini-batch

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$$

- μ_B : média das ativações do canal no mini-batch
- σ_B^2 : variância no canal
- Estatísticas são computadas **independentemente para cada canal**

Etapa 2: normalização

Centralização e escalonamento

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

- Centra as ativações e ajusta a escala
- ϵ : para evitar divisão por 0
- A saída $\hat{x}_i$ tem média 0 e variância 1

Etapa 3: reescala e deslocamento

Ajuste com parâmetros treináveis

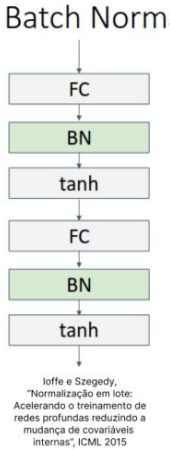
$$y_i = \gamma \hat{x}_i + \beta$$

- γ e β : são aprendidos durante o treinamento
- Permitem que a rede desfaça ou ajuste a normalização
- Isso mantém a capacidade representacional da rede

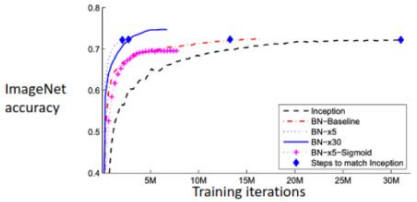
Efeitos práticos da BatchNorm

BN acelera e estabiliza o treinamento

- Permite usar **learning rates maiores**
- Acelera a convergência
- Reduz overfitting
  (efeito regularizador leve)
- Deixa as redes menos sensíveis à inicialização
- **Sem custo na inferência**:
  pode ser fundida com a convolução



- Torna redes profundas muito mais fáceis de treinar
- Permite taxas de aprendizado maiores e convergência mais rápida
- Deixa as redes mais robustas à inicialização dos pesos
- Atua como uma forma de regularização durante o treinamento
- Não adiciona custo na inferência: pode ser fundida com a camada convolucional



Prática

data@icmc.usp.br

@data.icmc

/c/DataICMC

/icmc-data

data.icmc.usp.br

obrigado!