Deep Learning

Conexões Residuais

Lucas Silveira Kupssinskü

Machine Learning Theory and Applications Laboratory
PUCRS

agosto de 2026

Visão Geral

Visão Geral

Contexto

- Até agora vimos que:
  - MLPs são bons para resolver problemas de regressão e classificação.
  - Camadas convolucionais são boas para trabalhar com imagens.
  - Aumentar a profundidade da rede tende a melhorar os resultados.
  - Gradiente que se dissipa e gradiente explosivo são problemas, principalmente em redes mais profundas.

O que vamos ver agora

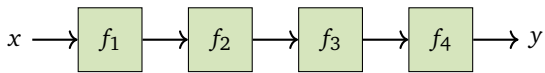
- **O problema:** redes muito profundas podem treinar **pior** que redes mais rasas, o **problema da degradação**, apesar de terem mais capacidade.
- **A solução: conexões residuais** (ou *skip connections*), atalhos **aditivos** que somam a entrada de volta à saída de cada bloco.
- E o *batch norm*, para estabilizar a variância das ativações ao longo da profundidade.

Visão Geral

Processamento Sequencial

- Tanto as CNNs quanto os MLPs vistos até aqui processam sequencialmente cada uma das suas camadas.

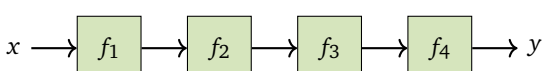
$$
\begin{aligned}
h_1 &= f_1[x, \theta_1] \\
h_2 &= f_2[h_1, \theta_2] \\
h_3 &= f_3[h_2, \theta_3] \\
&\vdots \\
h_n &= f_n[h_{n-1}, \theta_n]
\end{aligned}
$$



Composição de Funções

- Uma forma alternativa de pensar é escrever a mesma expressão como uma **composição de funções**:

$$h_1 = f_1[x, \theta_1]$$
$$h_n = f_n[\ldots f_3[f_2[f_1[x, \theta_1], \theta_2], \theta_3], \theta_n]$$



Profundidade vs Acurácia

- **AlexNet**[1].
  - 8 camadas.
  - Acurácia 84.7% (top-5).
- **VGG-16**[2].
  - 16 camadas.
  - Acurácia 92.7% (top-5).
- Por que não aumentar ainda mais a profundidade?
  - A partir de certo ponto, a acurácia **piora**: é o **problema da degradação**.
  - **Não é *overfitting***: o **erro de treino** também aumenta.
  - Paradoxo: uma rede de 56 camadas deveria, no mínimo, igualar uma de 20 copiando as camadas extras como **identidade**, mas, na prática, treina pior.

[1] Alex Krizhevsky, Ilya Sutskever e Geoffrey E. Hinton. "ImageNet classification with deep convolutional neural networks". Em: NeurIPS. vol. 25. 2012.

[2] Karen Simonyan e Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". Em: arXiv preprint arXiv:1409.1556 (2014).

Gradiente Quebrado (*Shattered Gradient*)

- A causa da **degradação** ainda não é completamente compreendida.
- **Uma das hipóteses** é a do "gradiente quebrado" (*shattered gradient*)[3].
- Pequenas mudanças nos pesos causam mudanças erráticas nas camadas seguintes.
    - A diferença do passo "infinitesimal" das equações para o passo finito utilizado na prática faz cada vez mais diferença.

[3] David Balduzzi et al. "The shattered gradients problem: If resnets are the answer, then what is the question?" Em: ICML. 2017, pp. 342–350.

Gradiente Quebrado: Intuição

- Pela regra da cadeia:

$$\frac{\partial f_4}{\partial f_1} = \frac{\partial f_4}{\partial f_3}\frac{\partial f_3}{\partial f_2}\frac{\partial f_2}{\partial f_1}$$

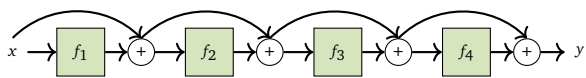
- Uma mudança em f_1 altera todas as funções subsequentes.
- Em redes muito profundas, $\partial y/\partial x$ se torna extremamente errático em relação à entrada.
- Quanto mais camadas, menos correlacionado o gradiente fica entre passos próximos.

Visão Geral

Conexões Residuais (*Residual* ou *Skip Connections*)

- Criamos ramos no grafo computacional, adicionando uma cópia da entrada a cada camada:

$$
\begin{aligned}
h_1 &= x + f_1[x, \theta_1] \\
h_2 &= h_1 + f_2[h_1, \theta_2] \\
h_3 &= h_2 + f_3[h_2, \theta_3] \\
h_4 &= h_3 + f_4[h_3, \theta_4]
\end{aligned}
$$



- **Por que "residual"?** O bloco aprende só o **resíduo** $f(x)=H(x)-x$ (a correção a aplicar sobre x). Se a transformação ideal for a identidade, basta $f \to 0$, fácil de aprender, o que resolve o paradoxo da degradação.
- Cada bloco f_i com sua conexão residual é um *residual block*; a saída precisa ter o mesmo tamanho da entrada.

Expansão da Expressão

- Abrindo as expressões anteriores temos:

$$
\begin{aligned}
y = {} & x + f_1[x] \\
& + f_2\big[\, x + f_1[x] \,\big] \\
& + f_3\big[\, x + f_1[x] + f_2[x + f_1[x]] \,\big] \\
& + f_4\big[\, x + f_1[x] + f_2[x + f_1[x]] + f_3[x + f_1[x] + f_2[x + f_1[x]]] \,\big]
\end{aligned}
$$

- Observe que a saída é uma soma de várias contribuições, cada uma percorrendo um **caminho** diferente da entrada até a saída.

Múltiplos Caminhos da Entrada à Saída

- Em uma rede com 4 blocos residuais, f_1 contribui por $\mathbf{8} = 2^{4-1}$ caminhos distintos para a saída.
- De forma geral, em uma rede com n blocos residuais existem $\mathbf{2^n}$ caminhos da entrada à saída (cada bloco escolhe entre ser pulado ou aplicado), e 2^{n-1} desses caminhos passam por cada função f_i específica.
- Cada caminho pode ser "ativo" ou "ignorado" por um bloco, dependendo da combinação das ramificações.
- A saída pode ser entendida como um ensemble implícito de redes de profundidades diferentes[4].

[4] Andreas Veit, Michael Wilber e Serge Belongie. "Residual networks behave like ensembles of relatively shallow networks". Em: NeurIPS. 2016.

Gradiente em Redes Residuais

- A derivada com relação a f_1 também tem múltiplos termos:

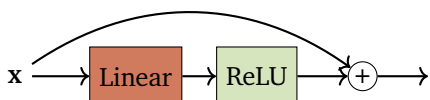
$$\frac{\partial y}{\partial f_1} = I + \frac{\partial f_2}{\partial f_1} + \left(\frac{\partial f_3}{\partial f_1} + \frac{\partial f_3}{\partial f_2}\frac{\partial f_2}{\partial f_1}\right) + \left(\frac{\partial f_4}{\partial f_1} + \frac{\partial f_4}{\partial f_2}\frac{\partial f_2}{\partial f_1} + \frac{\partial f_4}{\partial f_3}\frac{\partial f_3}{\partial f_1} + \frac{\partial f_4}{\partial f_3}\frac{\partial f_3}{\partial f_2}\frac{\partial f_2}{\partial f_1}\right)$$

- Intuição: a derivada de $x + f(x)$ é $1 + f'(x)$; o **"1"** (o termo I) é um **piso** que impede o gradiente de zerar.
- O termo I garante um caminho de gradiente que **não passa** por nenhum produto de derivadas.
  - Mesmo que algumas derivadas se aproximem de zero, o gradiente não desaparece completamente.
- Isso ataca diretamente o problema do gradiente que se dissipa.

Visão Geral

Bloco Residual

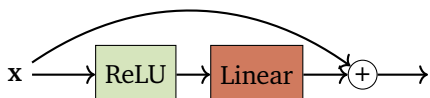
- Até aqui parece que $f_i[x]$ poderia ser uma camada qualquer.
  - Tecnicamente é verdade, mas na prática algumas escolhas funcionam melhor.
- Abaixo uma ilustração do que seria uma conexão residual em uma camada típica:



- Nessa configuração, a conexão residual só consegue **aumentar** o valor da representação recebida como entrada, nunca diminuir.

Bloco Residual: Inverter Ordem

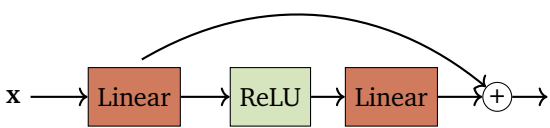
- Inverter a ordem da ativação e da transformação linear permite que a representação sofra adições e subtrações.



- Porém agora temos um problema novo: se a entrada chegar toda negativa, a ReLU "mata" a propagação *forward*.

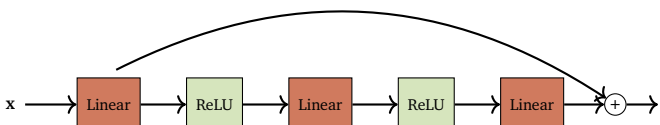
Bloco Residual: Linear Inicial

- Solução: começamos a rede com uma transformação linear **antes** dos blocos residuais.
- Dentro do bloco usamos a ordem ReLU → Linear, que pode somar ou subtrair.



Bloco Residual: Múltiplas Transformações

- Podemos adicionar mais de uma transformação no mesmo bloco residual.



- A ramificação “salta” por cima de um conjunto inteiro de operações.

Aumentando a Profundidade

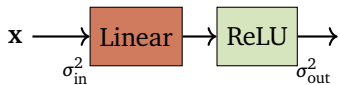
- Ao adicionar conexões residuais conseguimos dobrar a profundidade das redes convolucionais e ainda assim manter o aumento de performance.
  - Mas ainda existem fenômenos que nos impedem de ir mais profundamente.
  - Para entender isso, precisamos estudar a **variância das ativações** em redes com *skip connections*.

Visão Geral

Variância em Linear + ReLU

- A inicialização He[5] garante que a variância de uma variável aleatória após uma transformação Linear + ReLU não será modificada:

$$\sigma_{\text{in}}^2 = \sigma_{\text{out}}^2$$

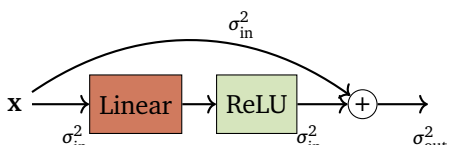


[5] Kaiming He et al. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". Em: ICCV. 2015.

Variância com Conexão Residual

- Porém agora temos a *skip connection*.
- Como a variância da saída se comporta?

$$\sigma^2_{\text{out}} = ?$$



Variância de uma Soma

- Voltando aos fundamentos:

$$
\begin{aligned}
\operatorname{Var}[X] &= \mathbb{E}\left[(X - \mathbb{E}[X])^2\right] \\
\operatorname{Var}[X+Y] &= \mathbb{E}\left[\left((X+Y) - \mathbb{E}[X+Y]\right)^2\right] \\
&= \mathbb{E}\left[(X - \mathbb{E}[X])^2\right] + 2\,\mathbb{E}\left[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])\right] + \mathbb{E}\left[(Y - \mathbb{E}[Y])^2\right] \\
&= \operatorname{Var}[X] + 2\operatorname{Cov}(X,Y) + \operatorname{Var}[Y]
\end{aligned}
$$

Variância Após o Bloco Residual

- Assumindo independência entre X (entrada) e Y (saída do ramo processado):

$$\mathrm{Var}[X+Y] = \mathrm{Var}[X] + \underbrace{2\,\mathrm{Cov}(X,Y)}_{\to 0} + \mathrm{Var}[Y] = \mathrm{Var}[X] + \mathrm{Var}[Y]$$

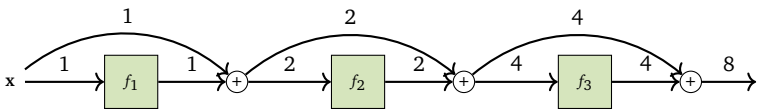
- Como ambos os ramos têm variância σ^2_{in}, a variância de saída é:

$$\sigma^2_{\text{out}} = 2\,\sigma^2_{\text{in}}$$

- Ou seja, a variância **dobra** depois de cada bloco residual.
- A premissa de independência entre X e Y é razoável **na inicialização**, quando ativações ainda não foram acopladas pelo treino. Durante o treinamento a covariância deixa de ser nula, mas a intuição de variância crescente continua valendo como motivação.

Variância Explosiva

- Em uma rede com muitas camadas, rapidamente podemos exceder a precisão de ponto flutuante.



- Sob a hipótese de independência, a variância dobra a cada bloco. A solução para esse problema é o *batch norm*.
- Em ResNets reais a variância **não** dobra estritamente, justamente porque o *batch norm* renormaliza as ativações após cada bloco. O argumento aqui motiva por que essa renormalização é necessária.

Visão Geral

Batch Norm

- É uma normalização aplicada a ativações em camadas ocultas da rede[6].
  - Desloca e reescala média e desvio padrão dos *batches* para valores aprendidos durante o treinamento.
- Necessita que utilizemos *batch size* > 1.

[6] Sergey Ioffe e Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". Em: ICML. 2015, pp. 448–456.

Batch Norm: Como Funciona

- Computa a média e o desvio padrão empíricos do *mini batch* atual:

$$\mu_h = \frac{1}{|B|} \sum_{i \in B} h_i, \qquad \sigma_h = \sqrt{\frac{1}{|B|} \sum_{i \in B} (h_i - \mu_h)^2}$$

- Padroniza as ativações para média 0 e desvio padrão 1:

$$h_i \leftarrow \frac{h_i - \mu_h}{\sigma_h + \epsilon}, \quad \forall\, i \in B$$

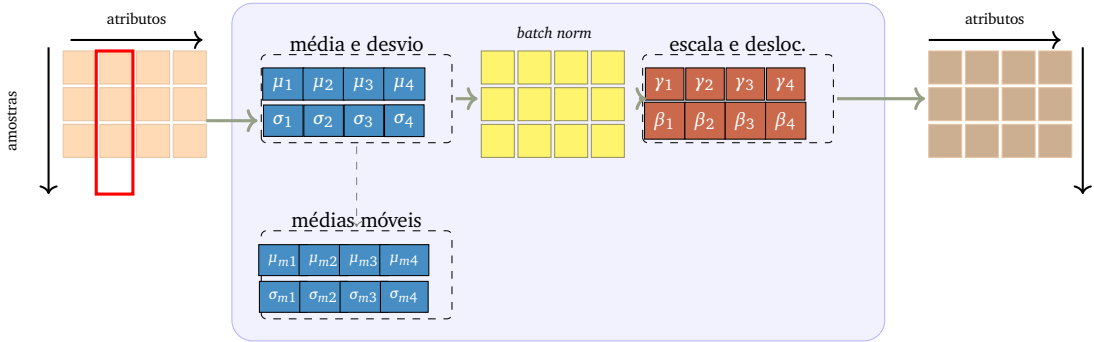
- Reescala as ativações por γ e desloca por β:

$$h_i \leftarrow \gamma\, h_i + \beta, \quad \forall\, i \in B$$

Batch Norm: Parâmetros

- Existe um γ, um β, um μ e um σ por unidade oculta em uma camada densa.
  - São 4 valores por unidade oculta. Apenas γ e β são **treináveis** via *backprop*.
  - μ e σ não são treinados, são calculados a partir do *mini batch* a cada passo e **mantidos como buffers** (médias móveis μ_{mov} e σ_{mov}) para uso em inferência.
  - γ e β são inicializados em 1 e 0, respectivamente.
- Em uma camada convolucional, existe um conjunto $(\gamma, \beta, \mu, \sigma)$ por **canal**.

Batch Norm: Estrutura



Batch Norm: Exemplo

- Considere um *mini batch* com uma *feature* de valores $\{3, -1, 2\}$.
  - Média: $\mu = 1.33$. Desvio padrão: $\sigma = 1.7$.
  - Padronizando: $\frac{3-1.33}{1.7} \approx 0.98$, $\frac{-1-1.33}{1.7} \approx -1.37$, $\frac{2-1.33}{1.7} \approx 0.39$.
- Com $\gamma = 2$ e $\beta = 1$ aprendidos:
  - $h_1 = 2 \cdot 0.98 + 1 = 2.96$.
  - $h_2 = 2 \cdot (-1.37) + 1 = -1.74$.
  - $h_3 = 2 \cdot 0.39 + 1 = 1.78$.
- Em treinamento, mantemos uma média móvel exponencial (EMA) das estatísticas:

$$\mu_{\text{mov}} \leftarrow \alpha\, \mu_{\text{mov}} + (1 - \alpha)\, \mu, \qquad \sigma_{\text{mov}} \leftarrow \alpha\, \sigma_{\text{mov}} + (1 - \alpha)\, \sigma$$

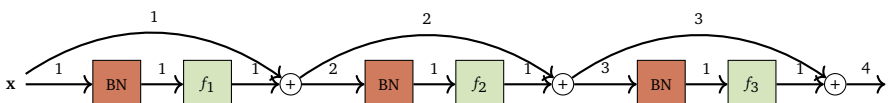
- **Atenção à convenção do PyTorch**: o argumento `momentum` em `nn.BatchNorm*` corresponde a $(1 - \alpha)$ nesta fórmula (peso da estatística atual), não a α (peso da média móvel). O valor padrão é `momentum=0.1`.

Batch Norm: Inferência

- Em treinamento usamos a média e o desvio padrão do próprio *mini batch*.
- Em inferência (sem *mini batch*), precisamos de estatísticas fixas.
- Solução: usar a média móvel acumulada durante o treinamento.
  - μ_{mov} e σ_{mov} são guardados como parte da camada e usados em substituição a μ e σ no momento da inferência.

Skip Connections + Batch Norm

- Com *batch norm* ativo, conseguimos treinar redes muito mais profundas, sem explosão nem dissipação de gradiente.



Batch Norm: Outras Vantagens

- Torna a rede invariante a mudanças de escala nos parâmetros.
  - Se os parâmetros dobrarem de magnitude, as ativações também dobram, e a variância também dobra (a primeira etapa do *batch norm* compensa esse aumento).
- *Forward* mais estável.
- Permite taxas de aprendizado (η) mais elevadas.
- Regulariza o processo de treinamento.

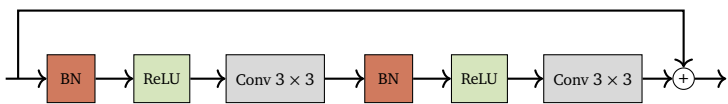
Visão Geral

Skip Connections + Batch Norm

- Usando *skip connections* e *batch norm* temos redes convolucionais com aplicações em praticamente todas as áreas de visão computacional.
  - Permitem treinamento de redes com $\sim$ 1000 camadas ocultas.
- Ainda hoje são o padrão para a maioria das atividades de visão computacional, apesar do recente sucesso de *transformers*.

ResNet

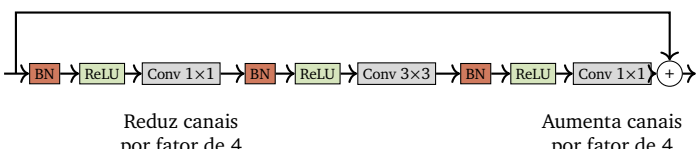
- Redes convolucionais constituídas de blocos residuais[7].
  - Cada bloco foi projetado por “tentativa e erro”.
  - Apresenta resultados bons, mas tem um número elevado de parâmetros.



[7] Kaiming He et al. “Deep residual learning for image recognition”. Em: CVPR. 2016, pp. 770–778.

ResNet: Bloco com Gargalo

- *Bottleneck Residual Block*, introduzido para reduzir o número de parâmetros treináveis.
  - A primeira convolução 1×1 **reduz** o número de canais.
  - A segunda convolução 1×1 **aumenta** o número de canais para possibilitar a adição no fim do bloco.

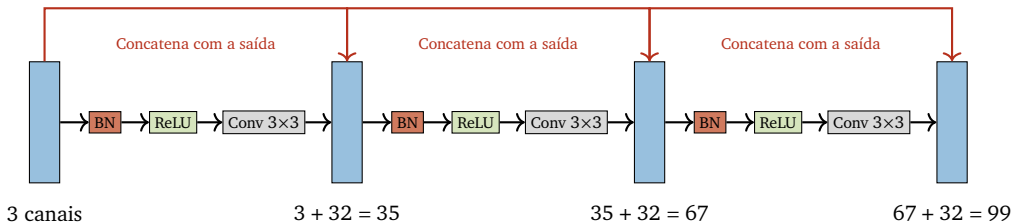


ResNet-200

- 4.8% de erro no ImageNet (*top-5*, teste multi-escala).
- Utiliza o bloco residual com gargalo.
- Estrutura geral: começa com uma convolução 7×7 com *stride* 2 e *max pooling*, seguida por blocos residuais agrupados por resolução.
  - Estágio 1: 56×56, 256 canais (64 internos).
  - Estágio 2: 28×28, 512 canais (128 internos).
  - Estágio 3: 14×14, 1024 canais (256 internos), 36 blocos.
  - Estágio 4: 7×7, 2048 canais (512 internos).
- No fim, *average pooling* global, camada totalmente conectada e *softmax* para 1000 classes.

DenseNet

- Em vez de **somar** a representação da entrada com a saída, podemos **concatenar** os canais da entrada com os canais de saída[8].
  - Menos comum que ResNet, mas atinge resultados similares.
  - Quando ocorre *downsampling*, a representação não é concatenada.



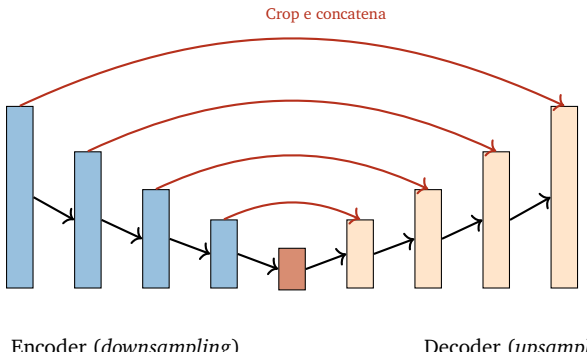
[8] Gao Huang et al. "Densely connected convolutional networks". Em: CVPR. 2017, pp. 4700–4708.

U-Net

- Arquitetura *encoder-decoder* com *skip connections* laterais[9]. Originalmente proposta para segmentação biomédica.
- Características essenciais:
  1. **Dois caminhos**: o *encoder* (contracting path) reduz a resolução espacial e aumenta a profundidade, o *decoder* (expanding path) faz o inverso, recuperando a resolução original via *upsampling* / convolução transposta.
  2. **Skip connections laterais**: *features* do *encoder* são **concatenadas** (não somadas, como nas ResNets) com as do *decoder* na mesma resolução, preservando detalhes espaciais que seriam perdidos no *bottleneck*.
- Muito utilizada em segmentação semântica e em modelos de difusão.
  - Como a convolução no paper original foi feita sem *padding*, é necessário cortar (*crop*) o lado do *encoder* antes de concatenar para que as dimensões espaciais batam.

[9] Olaf Ronneberger, Philipp Fischer e Thomas Brox. "U-Net: Convolutional networks for biomedical image segmentation". Em: MICCAI. 2015, pp. 234–241.

U-Net: Arquitetura



O *encoder* reduz a resolução; o *decoder* a recupera; as *skip connections* laterais **concatenam** (não somam) *features* da mesma resolução, preservando detalhes espaciais.

Visão Geral

Referências I

- Sugere se **fortemente** a leitura do Capítulo 11 de *Understanding Deep Learning*[10].
  - Disponível em `https://udlbook.github.io/udlbook/`.
- Artigos de referência sobre o tema:
  - He et al., *Deep Residual Learning for Image Recognition*, CVPR 2016.
  - Ioffe e Szegedy, *Batch Normalization*, ICML 2015.
  - Huang et al., *Densely Connected Convolutional Networks*, CVPR 2017.
  - Ronneberger et al., *U-Net*, MICCAI 2015.
  - Balduzzi et al., *The Shattered Gradients Problem*, ICML 2017.

[1] David Balduzzi et al. “The shattered gradients problem: If resnets are the answer, then what is the question?” Em: ICML. 2017, pp. 342–350.

[2] Kaiming He et al. “Deep residual learning for image recognition”. Em: CVPR. 2016, pp. 770–778.

[3] Kaiming He et al. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. Em: ICCV. 2015.

[4] Gao Huang et al. “Densely connected convolutional networks”. Em: CVPR. 2017, pp. 4700–4708.

[5] Sergey Ioffe e Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. Em: ICML. 2015, pp. 448–456.

[6] Alex Krizhevsky, Ilya Sutskever e Geoffrey E. Hinton. “ImageNet classification with deep convolutional neural networks”. Em: NeurIPS. Vol. 25. 2012.

[7] Simon J. D. Prince. Understanding Deep Learning. MIT Press, 2023.

Referências II

[8] Olaf Ronneberger, Philipp Fischer e Thomas Brox. “U-Net: Convolutional networks for biomedical image segmentation”. Em: MICCAI. 2015, pp. 234–241.

[9] Karen Simonyan e Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. Em: arXiv preprint arXiv:1409.1556 (2014).

[10] Andreas Veit, Michael Wilber e Serge Belongie. “Residual networks behave like ensembles of relatively shallow networks”. Em: NeurIPS. 2016.

[10] Simon J. D. Prince. Understanding Deep Learning. MIT Press, 2023.