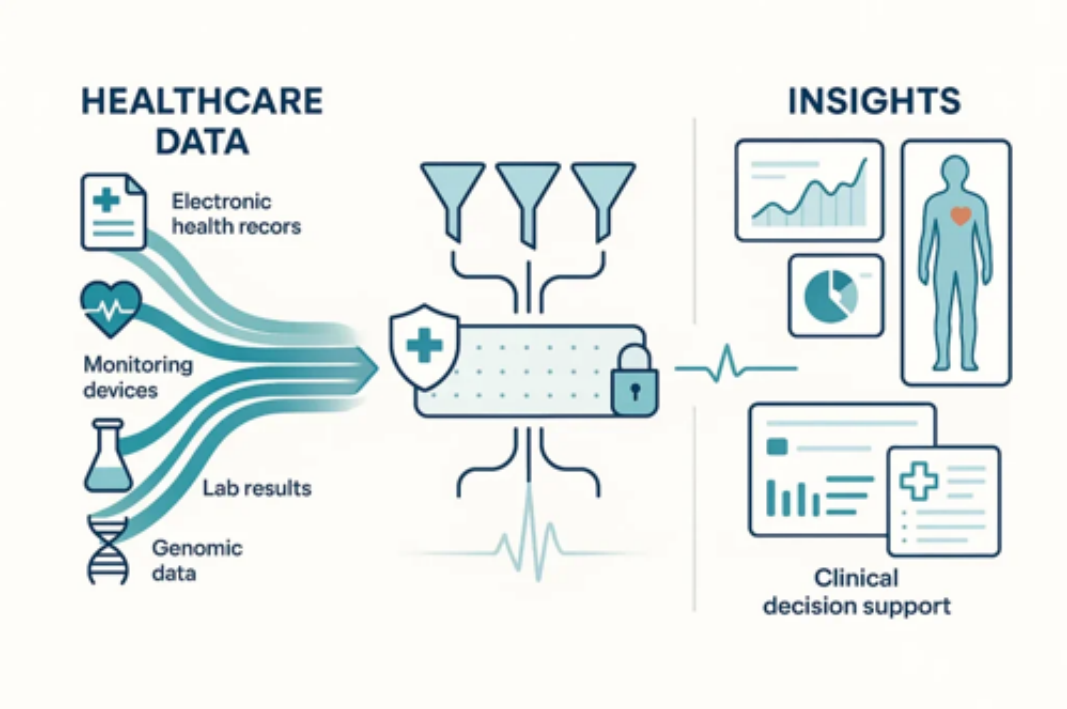
# Visão Computacional

## Transfer Learning e Arquiteturas de CNN

Prof. Dr. Denis Mayr Lima Martins

Pontifícia Universidade Católica de Campinas

Qual o **maior** problema em depender **apenas** de aprendizado **supervisionado** tradicional?



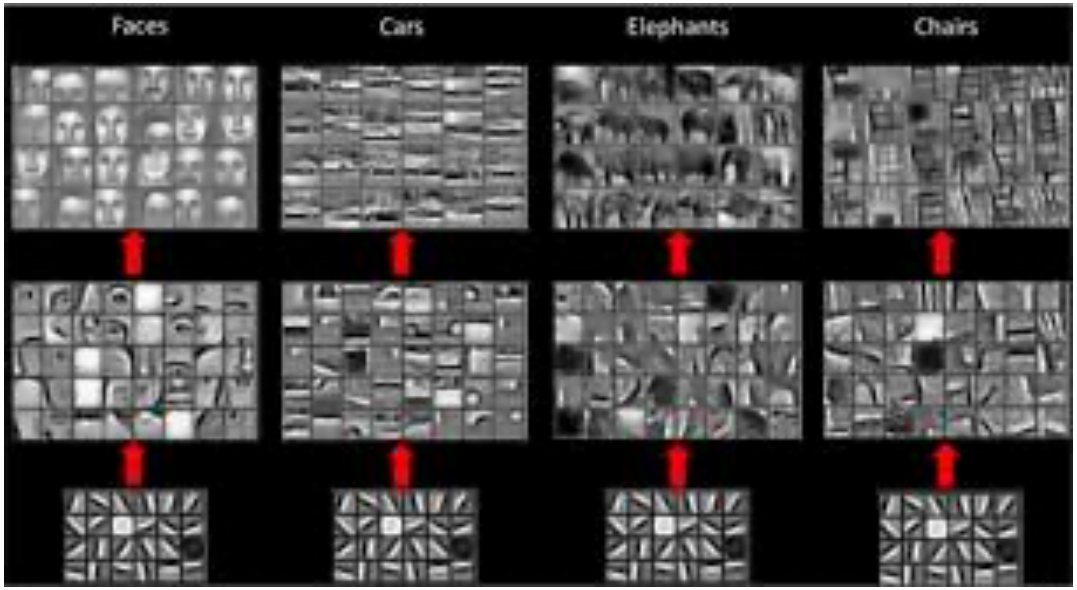

Dados anotados (labeled data) na área da Saúde são caros e difíceis de obter em larga escala. Fonte: Macgence.

# Treinamento Supervisionado Tradicional

- **Escassez de Dados Rotulados:** Obter dados rotulados é caro e demorado.
- **Generalização Limitada:** Modelos treinados em datasets específicos podem ter dificuldades para generalizar para novos cenários.
- **Necessidade de Engenharia de Features:** Requer conhecimento especializado para selecionar e criar features relevantes.

# O Papel das Features "Universais"

- As **features de baixo nível** (ex: bordas, cantos) são **universalmente úteis** para tarefas de Visão Computacional, independentemente do domínio.
- **Camadas Convolucionais Iniciais:** Estas são o **extrator de características** central de qualquer CNN.
- **Importância da Generalização:** O conhecimento destas camadas (ex: conv1, conv2) tende a se generalizar bem para *datasets* drasticamente diferentes (ex: imagens naturais para imagens médicas).




Features extraídas (i.e., aprendidas) por CNNs. Fonte: Cross Validated.

# Objetivos de Aprendizagem

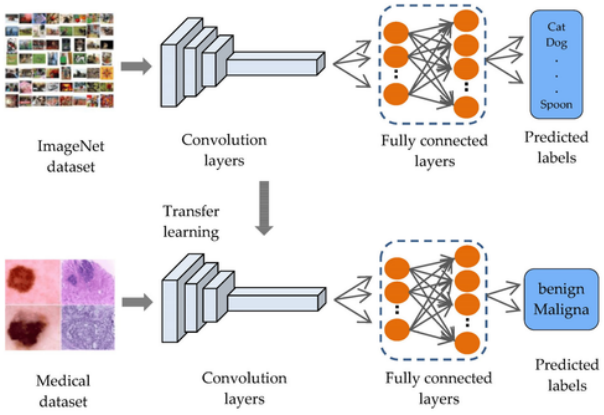
- Compreender o conceito de Transfer Learning em Redes Neurais.
- Entender o processo de Pré-Treinamento e como utilizar modelos Pré-Treinados.
- Compreender estratégias de Finetuning.
- Explorar arquiteturas famosas de CNNs.

# Transfer Learning

O Transfer Learning é popular em Deep Learning, pois as Redes Neurais Profundas (DNNs) exigem muitos dados e o custo de treinamento é alto.

**Motivações Principais:**

- **Escassez de Dados:** Resolve a limitação de dados rotulados insuficientes para novos projetos.
- **Custo Computacional:** Reduz o tempo e os recursos (GPU/CPU) necessários para treinar modelos do zero. O treinamento completo é computacionalmente intensivo.
- **Ganhos Empíricos:** Acelera a **convergência** e, em muitos casos, melhora a **acurácia**.




Transfer Learning. Fonte: https://toloka.ai/blog/transfer-learning/.

# Tipos de Transfer Learning

- **Transferência Indutiva:** Tarefas de origem e destino são **diferentes** (ex: pré-treinar para classificação de imagem, adaptar para detecção de objetos).
    - Pode ser Supervisionada (dados rotulados disponíveis).
- **Transferência Transdutiva (Adaptação de Domínio):** As tarefas são as **mesmas**, mas os domínios (distribuição dos dados) são diferentes.
    - Ex: Classificador de carros treinado com fotos de rua (Origem) e adaptado para fotos de satélite (Destino).
- **Transferência Não-Supervisionada:** Tarefas diferentes e não há dados rotulados na tarefa alvo (foco na estrutura dos dados).

# AlexNet

- Possui 8 camadas (5 convolucionais e 3 totalmente conectadas).
- Entrada de imagens $227 \times 227 \times 3$.
- Primeira camada (CONV1) usou $96$ filtros de $11 \times 11$ com *stride* 4.
- O *input* original era dividido em duas GPUs devido à limitação de memória (3 GB na GTX 580).

**Inovações Chave:**

- **ReLU** (Rectified Linear Unit): Foi a primeira utilização de ReLU em larga escala, substituindo *tanh* e *sigmoid*.
- **Max Pooling Sobreposto (Overlap Pooling):** Utiliza filtros $3 \times 3$ com *stride* 2, resultando em regiões de *pooling* que se sobrepõem.
- **Dropout:** Utilizado com probabilidade $0.5$ nas camadas totalmente conectadas (FC) para prevenir **overfitting**.

> *Paper:*
> *https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76*
> *Paper.pdf*

# AlexNet

---

```python
import torch
import torchvision.models as models

# Carrega o modelo pré-treinado
alexnet = models.alexnet(pretrained=True)
```

# VGG

---

- A VGG16 (16 camadas de peso) e VGG19 (19 camadas de peso) usam exclusivamente camadas convolucionais $3 \times 3$ com *stride* 1 e *padding* 1, e camadas de *max pooling* $2 \times 2$ com *stride* 2.
- Uma pilha de três camadas convolucionais $3 \times 3$ (com *stride* 1) tem o **mesmo campo receptivo efetivo** que uma única camada $7 \times 7$.
- Paper: https://arxiv.org/pdf/1409.1556

**Vantagens do Filtro Pequeno ($3 \times 3$):**

- Permite incorporar **mais não-linearidades** (funções de ativação ReLU).
- Resulta em **menos parâmetros** em comparação com um único filtro grande equivalente, por exemplo, $3 \times (3^2C^2)$ versus $7^2C^2$.

**Desvantagens:**

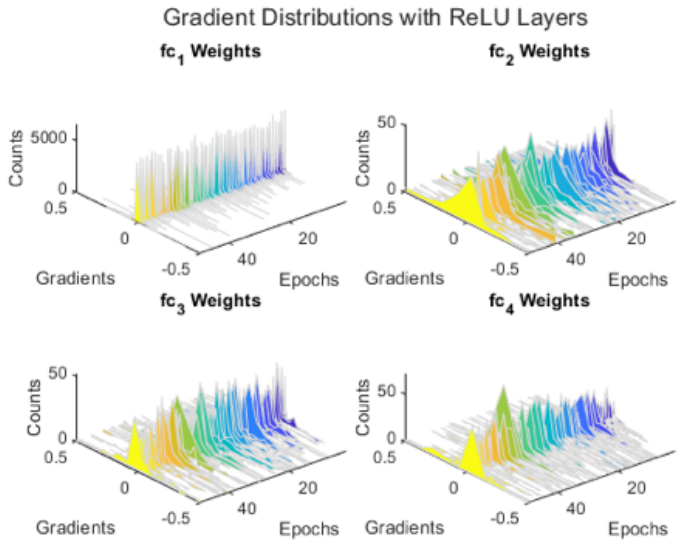
- Possui um **grande número de parâmetros** e é lenta, totalizando 138 milhões de parâmetros na VGG16.

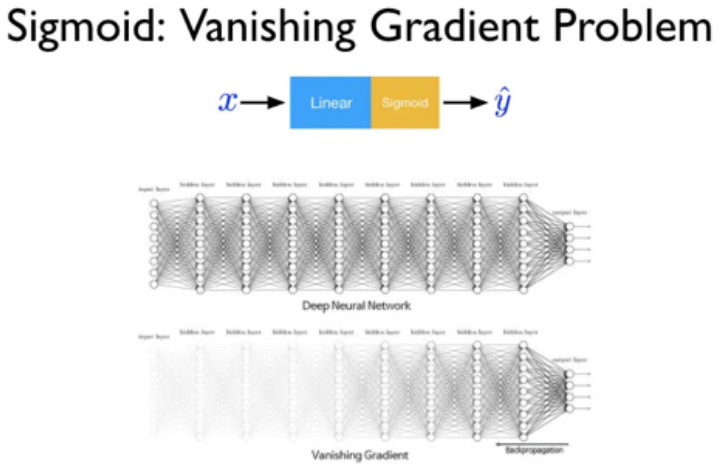
# VGG

```
import torch
import torchvision
import torch.nn as nn
from torchvision import models

model = models.vgg16(pretrained=True)
```

# Vanishing Gradients



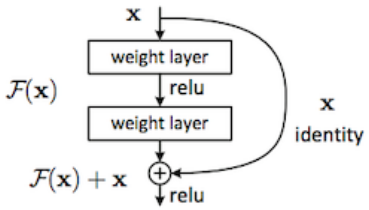



# ResNet

- ResNet (Residual Network) resolve o problema de **degradação** que ocorre em redes muito profundas, onde adicionar mais camadas pode levar a um erro de treinamento e teste maior.
- **Degradação vs. Otimização:** Redes profundas (sem atalhos) sofrem de **degradação**, onde o erro de treinamento piora com o aumento da profundidade.

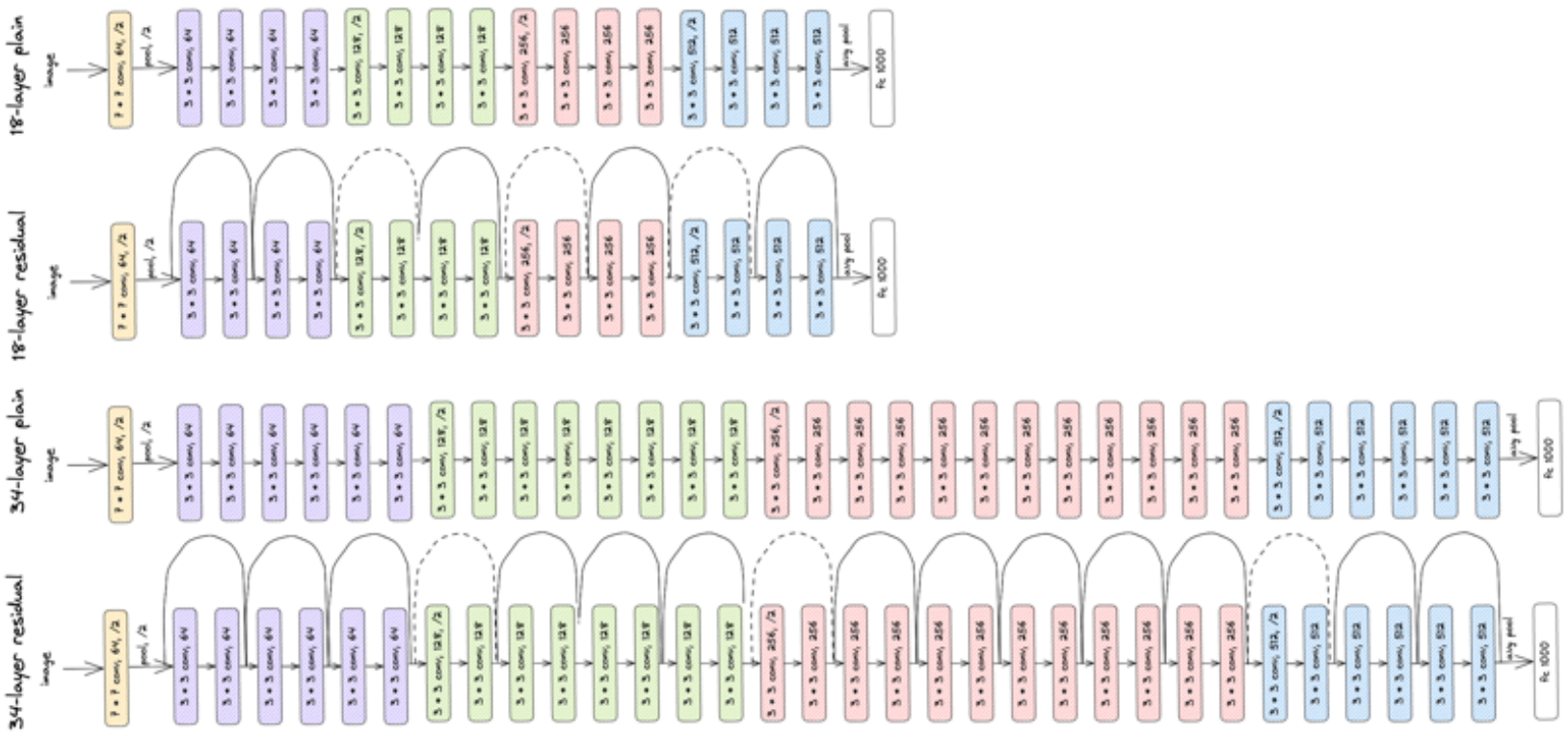
**Conexões Residuais (Skip Connections):**

- A arquitetura ResNet introduz um **bloco residual** que permite que a entrada $x$ seja adicionada à saída das camadas subsequentes: $H(x) = F(x) + x$.
- Isso torna a **propagação do gradiente** muito mais fácil (fluxo de gradiente direto).
- O bloco permite que as camadas aprendam a mapear o **resíduo** $F(x)$, tornando mais fácil para a rede aprender a função identidade se as camadas não forem úteis.
- Paper: https://arxiv.org/pdf/1512.03385 (com mais de 29 mil citações)




Skip connection.

# ResNet




Arquitetura ResNet. Fonte: evbn.org.

# ResNet

```
import torch
from torch import nn


class ResLayer(nn.Module):
     def __init__(self,ni,no,kernel_size,stride=1):
        super(ResLayer, self).__init__()
        padding = kernel_size - 2
        self.conv = nn.Sequential(
            nn.Conv2d(ni, no, kernel_size, stride,
                      padding=padding),
            nn.ReLU()
        )

    def forward(self, x):
        return self.conv(x) + x
```

> *Veja também: https://nn.labml.ai/resnet/index.html*

# ResNet

| layer name | output size | 18-layer | 34-layer | 50-layer | 101-layer | 152-layer |
|---|---|---|---|---|---|---|
| conv1 | 112×112 | 7×7, 64, stride 2 | | | | |
| conv2_x | 56×56 | 3×3 max pool, stride 2 | | | | |
| | | $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$ | $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$ | $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$ | $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$ | $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$ |
| conv3_x | 28×28 | $\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$ | $\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$ | $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$ | $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$ | $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 8$ |
| conv4_x | 14×14 | $\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$ | $\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$ | $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$ | $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 23$ | $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 36$ |
| conv5_x | 7×7 | $\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$ | $\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$ | $\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$ | $\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$ | $\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$ |
| | 1×1 | average pool, 1000-d fc, softmax | | | | |
| FLOPs | | $1.8\times10^9$ | $3.6\times10^9$ | $3.8\times10^9$ | $7.6\times10^9$ | $11.3\times10^9$ |

Arquitetura ResNet. Fonte: He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

# U-Net

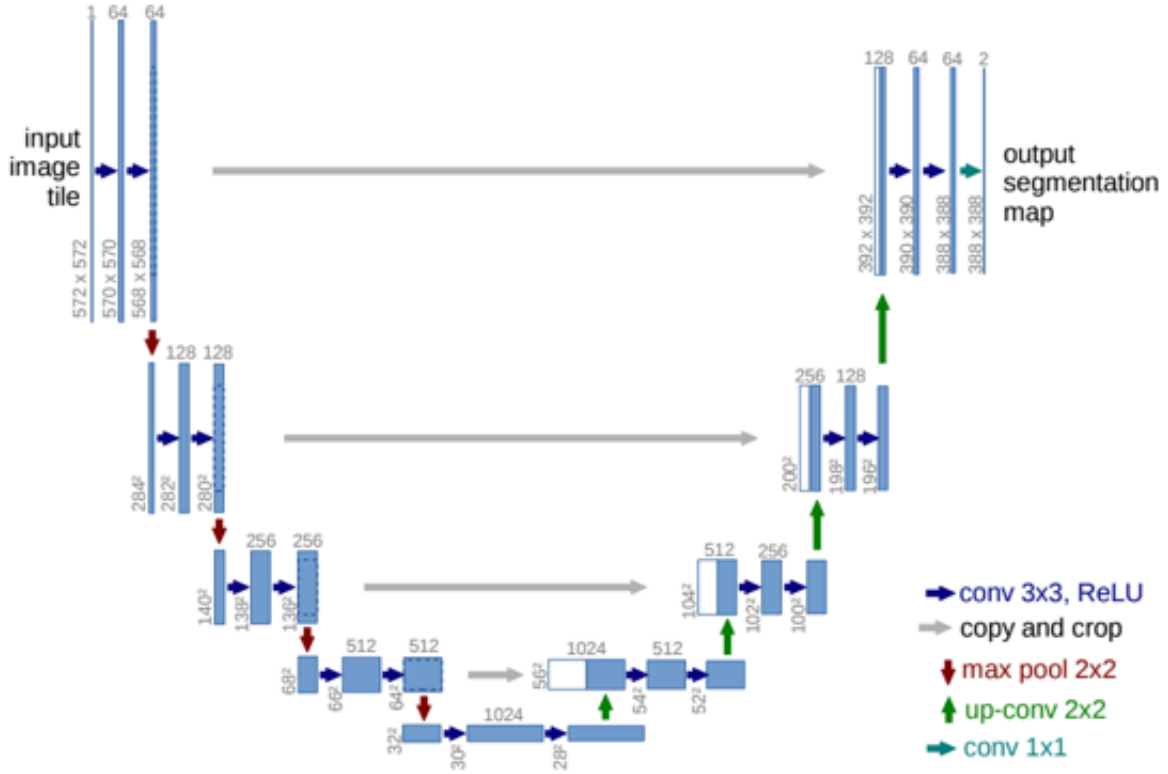



**Fig. 1.** U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

Arquitetura U-Net. Fonte: Labml.

# Armadilhas Comuns no Transfer Learning

A aplicação incorreta do Transfer Learning pode resultar em consequências negativas, conhecidas como **transferência negativa**.

| Armadilha | Descrição | Dicas de Mitigação |
|---|---|---|
| **Overfitting** em Conjuntos Pequenos | Adaptação excessiva a poucos exemplos no *dataset* alvo. | Usar **taxas de aprendizado muito baixas** (1e-4 a 1e-5), **regularização** (dropout, weight decay), e *data augmentation*. |
| **Esquecimento Catastrófico** | Perda do conhecimento original (da tarefa fonte) durante o fine-tuning para a tarefa alvo. | Aplicar **congelamento seletivo** nas camadas iniciais ou usar técnicas avançadas como regularização elástica. |
| **Deslocamento de Domínio** (Transferência Negativa) | O conhecimento transferido prejudica o desempenho porque os domínios de origem e destino são muito dissimilares. | Garantir que a **distribuição** dos dados de origem e destino sejam semelhantes e que as tarefas sejam relacionadas. |

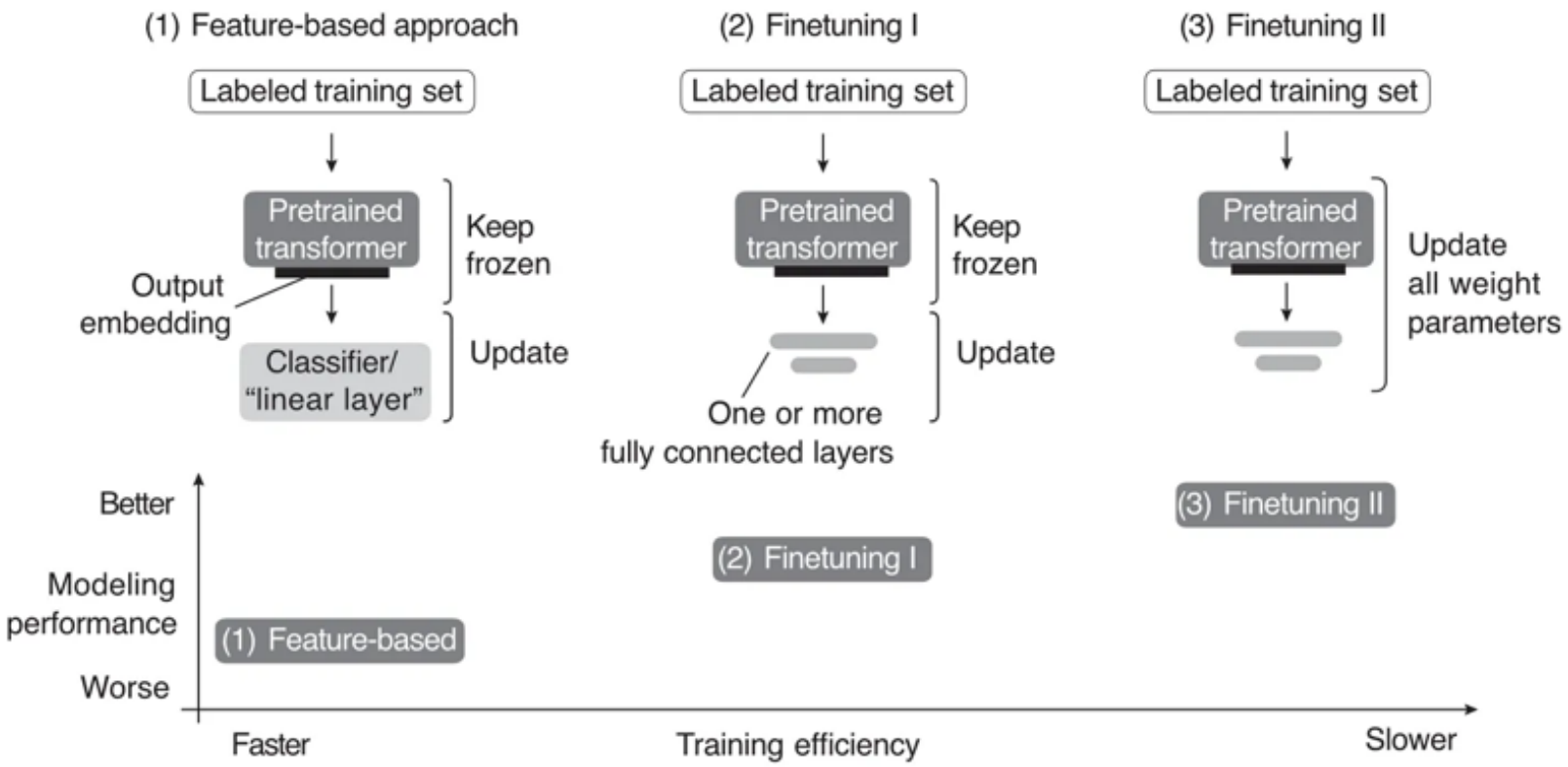
# Fine-Tuning

O Fine-tuning é o processo de retreinar um modelo pré-treinado usando uma taxa de aprendizado menor para se adaptar a uma tarefa específica.

**Técnicas de Ajuste de Pesos:**

- **Congelar Camadas (Feature Extraction Puro):** As camadas inferiores (extrator de características) são mantidas intactas (*frozen*), preservando as **features universais** (bordas, texturas). Apenas a camada final de classificação é treinada do zero.
- **Taxas de Aprendizado Diferenciadas (Layer-wise Learning Rate Decay):** Aplica-se taxas de aprendizado progressivamente menores para camadas mais próximas da entrada. Isso evita a destruição de conhecimento fundamental.
- **Treinamento em Poucas Épocas:** Geralmente, 2–10 épocas são suficientes para a nova tarefa, monitorando com *early stopping*.

# Estratégias de Uso de Modelos Pré-Treinados




Abordagens de uso. Fonte: Raschka.

# Fine-tuning: Exemplo

```
model = models.vgg16(pretrained=True)

# Congela os pesos da rede original
for param in model.parameters():
    param.requires_grad = False

model.classifier = nn.Sequential(
    nn.Flatten(),
    nn.Linear(512, 128),
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.Linear(128, 1),
    nn.Sigmoid()
)
```

# Conclusão e Próximos Passos

- **Necessidade Crucial:** Transfer Learning é indispensável para superar a **escassez de dados** e o **alto custo computacional** em deep learning.
- **Transferência Hierárquica:** CNNs aprendem **features hierárquicas**; as camadas iniciais (bordas, blobs) são universais e devem ser preservadas (congeladas).
- **Fine-tuning é Delicado:** O ajuste fino requer o uso de **baixas taxas de aprendizado** para evitar a sobreescrita do conhecimento valioso pré-adquirido.
- **ResNet é o Default Atual:** Arquiteturas como **ResNet** (152 camadas, conexões residuais) se tornaram o padrão, superando problemas de otimização em redes extremamente profundas.
- Outras arquiteturas famosas: Inception, DenseNet, U-Net, EfficientNet, MobileNet.
- Exemplos de código em https://github.com/rasbt/deeplearning-models
- Repositório de Modelos: https://github.com/huggingface/pytorch-image-models