Visão Computacional com CNNs e Transformers

Aula 1: Arquiteturas CNN Clássicas e Modernas — Evolução, Transfer Learning e Além da Classificação

Etapa / Aula 01

Faculdade Infnet • Pós-Graduação em Inteligência Artificial & Visão Computacional

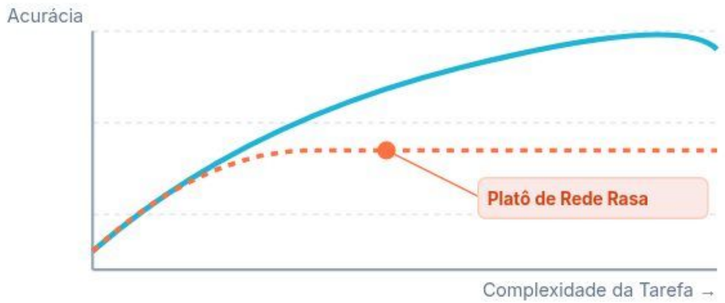
- LeNet-5 & AlexNet
- GoogLeNet & ResNet
- Xception & SENet
- VGG, MobileNet & EfficientNet
- GPU RAM: Inferência vs Treino
- RevNets (Reversible Nets)
- ResNet-34 em PyTorch
- Transfer Learning no TorchVision
- Detecção (YOLO), Tracking & Segmentação

O Limite das Redes Rasas: Platô de Capacidade

A composição de características visuais complexas exige profundidade hierárquica

Curva de Capacidade Representacional

Platô Precoce



Rede Rasa (3–5 camadas) — Rede Profunda (Hierárquica)

L1-2 **Nível Baixo (Bordas & Cores)**
Gradientes locais, arestas 45°/90°, manchas de cor

L3-6 **Nível Médio (Texturas & Padrões)**
Malhas, curvas, cantos, círculos e listras

L7+ **Nível Alto (Partes & Semântica)**
Rodas, faróis, olhos, focinhos, objetos inteiros

💡 **Conclusão:** Redes rasas não têm profundidade suficiente para compor representações semânticas de alto nível.

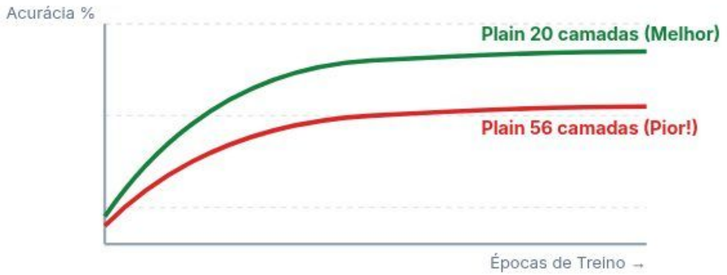
SITUAÇÃO-PROBLEMA | O Grande Paradoxo

O Paradoxo da Degradação e o Desvanecimento de Gradiente

Por que simplesmente empilhar camadas convencionais fazia a rede piorar até no treino?

O Paradoxo da Degradação (He et al., 2015)

Acurácia Treino/Teste



⚠️ **Não é Overfitting:** A rede de 56 camadas tem acurácia *menor* até no treino! É um bloqueio de otimização dos gradientes.

Desvanecimento do Gradiente $\nabla \mathcal{L} \to 0$

Regra da Cadeia

Propagação da Derivada em L Camadas:

$$\frac{\partial \mathcal{L}}{\partial W_1} = \frac{\partial \mathcal{L}}{\partial a_L} \prod_{l=2}^{L} \left(W_l^T \cdot \sigma'(z_l) \right)$$

Fluxo do Gradiente (Backprop ←)

Perda Exponencial de Sinal

L1 Entrada	←	L20 Oculta	←	L40 Oculta	←	L56 Saída
≈ 0.0001		≈ 0.08		≈ 0.45		1.00

💡 As sucessivas multiplicações matriciais amortecem o gradiente até zero nas camadas iniciais.

O Mapa das Grandes Soluções Arquiteturais

As inovações matemáticas que destravaram o treinamento de redes ultra-profundas

1. Bottlenecks 1x1

- GoogLeNet / Inception (2014)
- Compressão dimensional de canais
- Reduz custos quadráticos em até 75%

2. Conexões Residuais

- ResNet (2015) — F(x) + x
- Atalhos diretos de identidade
- Supervia de gradiente (+1) no backward

3. Convoluções Separáveis

- Xception & MobileNet (2017)
- Desacoplamento: Espaço 2D vs Canais
- Economia de ~8x a 9x em FLOPs

4. Atenção aos Canais

- SENet (2017) — Squeeze & Excitation
- Recalibração adaptativa dinâmica
- < 1% de parâmetros extras

Esses quatro pilares formam a espinha dorsal de praticamente todas as CNNs modernas em produção hoje.

Modelos Pré-treinados no PyTorch

LINHAGEM HISTÓRICA | LeCun et al. • 1998

LeNet-5: A Gênese das Redes Convolucionais

A arquitetura pioneira que estabeleceu o paradigma convolucional para leitura de dígitos manuscritos

Tabela Arquitetural: LeNet-5 (LeCun et al., 1998) ~60k Parâmetros

Clique na linha para inspecionar

Layer	Tipo	Mapas	Tamanho (H × W)	Kernel	Stride	Ativação	Pesos
In	Entrada (Imagem)	1	32 × 32	–	—	–	0
C1	Convolução	6	28 × 28	5 × 5	1	tanh	156
S2	Avg Pooling	6	14 × 14	2 × 2	2	tanh	12
C3	Convolução	16	10 × 10	5 × 5	1	tanh	1.516
S4	Avg Pooling	16	5 × 5	2 × 2	2	tanh	32
C5	Convolução	120	1 × 1	5 × 5	1	tanh	48.120
F6	Densa (FC)	—	84	–	—	tanh	10.164
Out	Saída (RBF / FC)	—	10	–	—	Euclidean RBF	0

Fórmula de Dimensão: $O = \frac{W-K+2P}{S} + 1$

Total: 60.000 parâmetros

Camada C1: Convolução 1

Convolução • Ativação: tanh

Saída
28 × 28

6 filtros 5×5 aprendem padrões locais de baixo nível (bordas e ângulos). Tamanho: 32 - 5 + 1 = 28.

Cálculo de Pesos Aprendíveis:
`(5 × 5 × 1 + 1) × 6 = 156`

O Padrão Fundamental que Vocês Já Conhecem

[Conv → Pool] → [Conv → Pool] → [FC → FC]

A LeNet-5 foi a primeira rede a consolidar a extração espacial hierárquica seguida de classificação densa.

💡 **Marco Histórico:** Com apenas 60k pesos, operava com 99.2% de precisão no MNIST e leu milhões de cheques nos bancos dos EUA nos anos 90.

LINHAGEM HISTÓRICA | ImageNet 2012 • Ponto de Virada

AlexNet: O Ponto de Virada do Deep Learning

A convergência histórica entre GPUs paralelas, função ReLU, Dropout e Big Data no ImageNet

Tabela Arquitetural: AlexNet (Krizhevsky et al., 2012)

~61M Parâmetros

Clique na linha para inspecionar

Layer	Tipo	Mapas	Tamanho (H × W)	Kernel	Stride	Ativação	Pesos
In	Entrada (RGB)	3	224 × 224 × 3	–	—	–	0
C1	Convolução	96	55 × 55	11 × 11	4	ReLU	35k
P1	Max Pooling	96	27 × 27	3 × 3	2	–	0
C2	Convolução	256	27 × 27	5 × 5	1	ReLU	307k
P2	Max Pooling	256	13 × 13	3 × 3	2	–	0
C3	Conv (Cross-GPU)	384	13 × 13	3 × 3	1	ReLU	885k
C4	Convolução	384	13 × 13	3 × 3	1	ReLU	663k
C5	Conv + Pool	256	6 × 6	3 × 3	1	ReLU	442k
FC6	Densa + Dropout	—	4096	–	—	ReLU + Dropout	37.7M
FC7	Densa + Dropout	—	4096	–	—	ReLU + Dropout	16.8M
Out	Saída (Softmax)	—	1000	–	—	Softmax	4.1M

Comparação com LeNet (60k): 1000× mais parâmetros — Total: 61.000.000 (61M)

Camada C1: Convolução 1

Convolução • Ativação: ReLU

Dimensão: 55 × 55

96 filtros amplos 11×11 com stride 4 para rápida redução espacial inicial.

Alocação em Hardware: 48 por GPU

Pesos na Camada: 35k

Ativação ReLU
Substituiu Tanh: convergiu 6× mais rápido sem saturação.

Dropout (p = 0.5)
Regularizou as camadas densas FC6 e FC7 contra overfitting.

2× GPUs GTX 580
Divisão em 2 GPUs (3GB cada) com conexões cruzadas.

Data Augmentation
Cortes 224×224, espelhamento horizontal e PCA de cor.

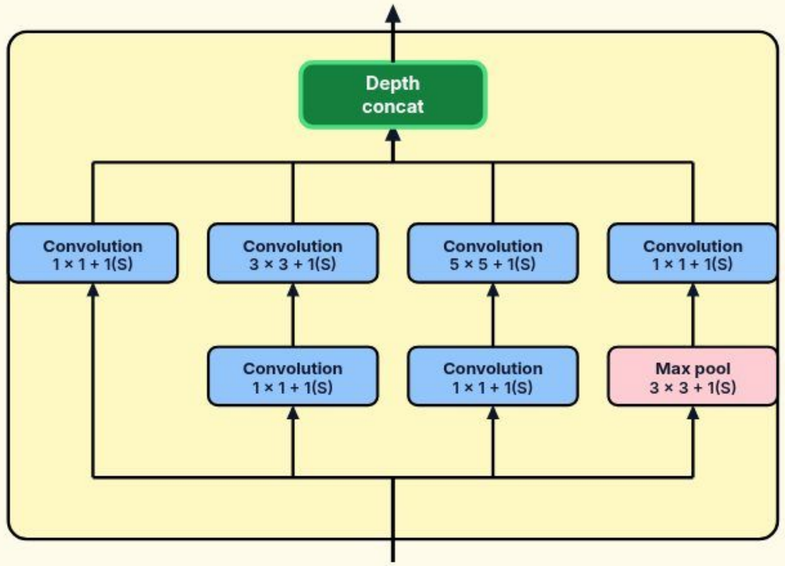
🚀 **Ponto de Virada:** Venceu o ImageNet 2012 reduzindo o erro Top-5 de 26% para 15.3%, iniciando a era do Deep Learning.

GoogLeNet: A Anatomia do Módulo Inception

Extração multirresolução em 4 ramos paralelos com redução de dimensão 1×1 e concatenação de profundidade

Módulo Inception (GoogLeNet / Szegedy et al., 2014)

Com Reduções 1×1 (Padrão) | Ingênuo (Sem 1×1)



Convolução (Azul) Max Pooling (Rosa) Concatenação (Verde)

+ 1(S) = Stride 1, Same Padding

Depth Concat: Concatenação de Profundidade

Fusão Multi-Escala

Como todos os 4 ramos utilizam stride 1 e same padding, todas as 4 saídas possuem exatamente a mesma resolução espacial (H × W). Elas são empilhadas ao longo dos canais.

Fluxo de Canais e Dimensões:

$$C_{total} = C_1 + C_2 + C_3 + C_4 = 256$$

1. Extração Multi-Escala em Paralelo

A rede não precisa escolher se 1×1, 3×3 ou 5×5 é melhor: ela aprende a dosagem ótima de cada escala em paralelo.

2. Redução de Dimensão 1×1 (Bottlenecks)

Sem a compressão 1×1 antes das convoluções 3×3 e 5×5, a quantidade de FLOPs e canais explodiria rapidamente a cada estágio.

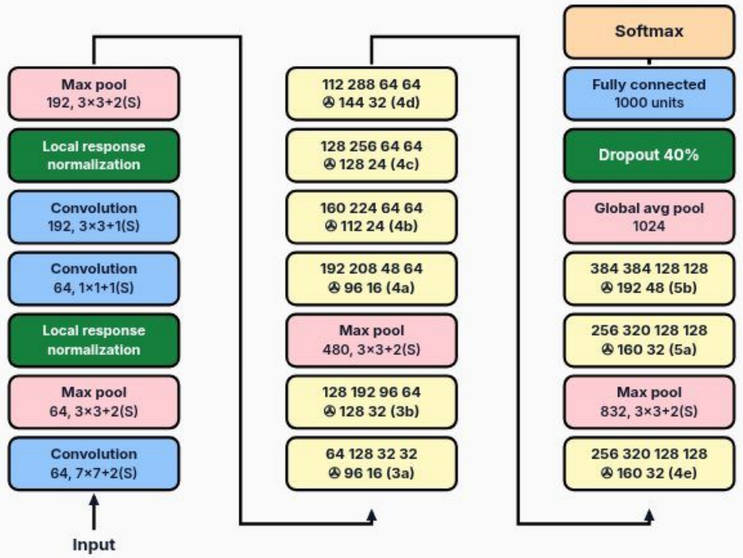
🏆 **Resultado:** O GoogLeNet (22 camadas) atingiu 6.7% de erro Top-5 com apenas **6.8M parâmetros** (quase 10x menos que a AlexNet!).

GoogLeNet: Arquitetura Completa e Fluxo em Cascata

A distribuição dos 9 módulos Inception entre o Stem inicial e o Global Average Pooling

GoogLeNet: Arquitetura Completa (22 Camadas) 9 Módulos Inception

Clique nas seções para explorar



Conv Pool Inception Norm/Drop

✇ = Módulo Inception (9 blocos no total)

2. Estágio Inception 3 (Módulos 3a e 3b)

Seção Ativa

Primeiro par de módulos Inception operando na resolução espacial de 28×28. Os módulos 3a e 3b aumentam a capacidade semântica de 192 para 480 canais.

- Inception 3a: Entrada 192 canais → Saída Depth Concat com 256 canais.
- Inception 3b: Entrada 256 canais → Saída Depth Concat com 480 canais.
- Seguido por MaxPool 3×3 (stride 2) que reduz a resolução espacial para 14×14.

Eliminação das Camadas Densas (GAP)

A AlexNet gastava 58M de pesos nas camadas FC. O GoogLeNet substituiu tudo por Global Average Pooling, usando apenas 1 camada linear no final.

Classificadores Auxiliares (Auxiliary Loss)

Em 4a e 4d, saídas intermediárias injetavam gradiente adicional no treino para vencer o desvanecimento em 22 camadas.

🏗️ **Visão Macro:** A rede inteira empilha **9 módulos Inception** em 3 resoluções ($28\times 28 \to 14\times 14 \to 7\times 7$), totalizando apenas **6.8M parâmetros**.

O Truque da Convolução 1x1 (Bottlenecks)

Compressão linear de canais que reduz o custo computacional em mais de 75%

Abordagem Direta Sem Bottleneck

462M FLOPs

Entrada **[256 canais, 28×28]** → Conv 3×3 → Saída **[256 canais, 28×28]**

Cálculo de Operações:

$$28 \times 28 \times (3 \times 3 \times 256) \times 256 \approx 462.4 \text{ MFLOPs}$$

❌ Custo quadrático no número de canais: $\mathcal{O}(C_{in} \cdot C_{out} \cdot K^2)$

Com Bottleneck 1×1 (GoogLeNet)

115M FLOPs (4x mais rápido!)

[256] → **Conv 1×1 (64)** → **Conv 3×3 (64)** → **Conv 1×1 (256)**

Cálculo de Operações:

$$(1 \times 1 \times 256 \times 64 + 3 \times 3 \times 64 \times 64 + 1 \times 1 \times 64 \times 256) \times 28^2 \approx 115 \text{ MFLOPs}$$

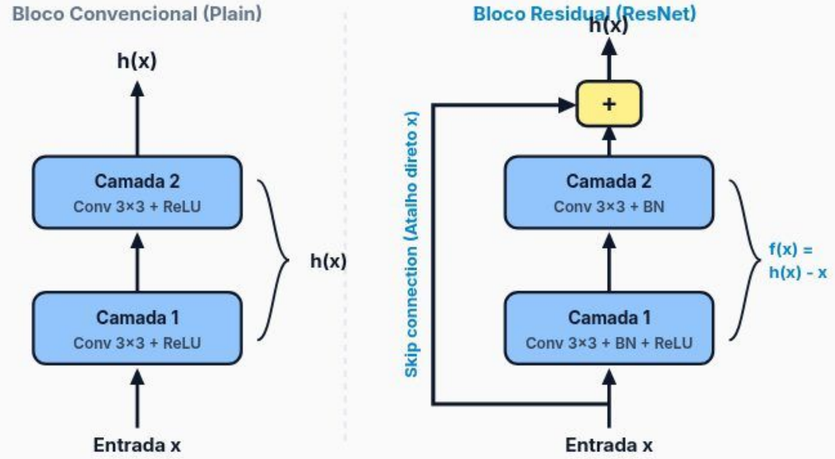
✓ **75% de economia** com a mesma capacidade expressiva!

ResNet: O Conceito do Aprendizado Residual

Por que aprender a perturbação f(x) = h(x) - x com atalho direto é mais fácil do que aprender h(x) do zero

A Ideia Central: Aprendizado Residual (He et al., 2015)

Simular Caso Ótimo: Identidade



💡 A rede só precisa aprender a modificação residual f(x) necessária sobre a entrada x.

A Formulação Matemática Residual

Seja $h(\mathbf{x})$ o mapeamento que queremos aprender. A ResNet divide o problema em dois termos:

$$h(\mathbf{x}) = f(\mathbf{x}) + \mathbf{x} \iff f(\mathbf{x}) = h(\mathbf{x}) - \mathbf{x}$$

Por que é mais fácil? Em vez de aprender toda a estrutura da imagem do zero, as camadas só precisam aprender o *resíduo* (a pequena correção ou detalhe novo a ser somado).

1. Facilidade de Aprender a Identidade

Se uma camada profunda for inútil, empurrar os pesos para zero ($f(\mathbf{x}) \to 0$) é muito fácil. A rede preserva o desempenho sem piorar (degradação zero).

2. A "Supervia" de Gradiente (+1)

Na derivada $\frac{\partial h}{\partial \mathbf{x}} = \frac{\partial f}{\partial \mathbf{x}} + 1$, o termo $+1$ garante que o gradiente sempre flui intacto até a primeira camada da rede!

🏆 **Resultado Seminal:** Permitiu que a ResNet treinasse redes de **34, 50, 101 e até 152 camadas** com erro de apenas 3.57% no ImageNet 2015.

ARQUITETURAS MODERNAS | ImageNet 2015 • Vencedor Absoluto

ResNet: Explorador de Profundidade e Gradientes

Simule o fluxo de gradiente através de dezenas de camadas com e sem conexões de atalho

Profundidade da Rede (Camadas): 34

Sem Skip (Plain Net) | **Com Skip (ResNet: F(x) + x)**

Fluxo do Gradiente no Backward Pass (Camadas Finais → Iniciais)

Gradiente Preservado

O atalho de identidade (+ x) cria uma via expressa linear para o gradiente, permitindo treinar 100+ camadas.

L1 100%	L2 99%	L3 99%	L4 98%	L5 97%	L6 96%	L7 96%	L8 95%	L9 94%	L10 93%	L11 93%	L12 92%
L13 91%	L14 90%	L15 90%	L16 89%	L17 88%	L18 88%	L19 87%	L20 86%	L21 85%	L22 85%	L23 84%	L24 83%

Gradiente com Conexão Residual:

$$\frac{\partial \mathcal{L}}{\partial x_l} = \frac{\partial \mathcal{L}}{\partial x_L}\left(\frac{\partial \mathcal{F}}{\partial x_l} + 1\right)$$

→ O termo "+ 1" garante propagação ininterrupta!

Impacto no Treinamento

Erro no Treino (Loss)	Top-1 Val Accuracy
0.145	**76.1%**

Comportamento ResNet: Adicionar mais camadas melhora consistentemente a acurácia sem sofrer do problema de degradação.

Xception: Convoluções Espacialmente Separáveis

Desacoplando a filtragem espacial 2D da combinação linear entre canais

1. Convolução Padrão 3D

Acoplado (Pesado)



Custo: H × W × K² × C_in × C_out FLOPs

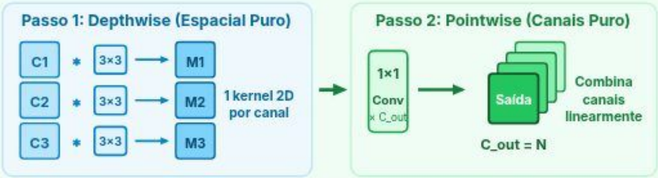
Operação acoplada em 1 único passo:

`Filtro 3D: K × K × C_in × C_out`

⚠️ **Gargalo:** Cada filtro precisa correlacionar o espaço 2D e todas as combinações de canais ao mesmo tempo.

2. Depthwise Separable (Xception / MobileNet)

Desacoplado (~8.5x mais rápido)



Custo: H × W × (K² × C_in + C_in × C_out) FLOPs

1. Depthwise (Espaço)
1 filtro 2D por canal
`K × K × C_in`

2. Pointwise (Canais)
Conv 1×1 linear
`1 × 1 × C_in × C_out`

$$\frac{\text{Custo Depthwise}}{\text{Custo Padrão}} = \frac{1}{C_{out}} + \frac{1}{K^2} \approx \frac{1}{9} \quad (\text{para } K = 3)$$

✓ Economia de **~88% de FLOPs** sem comprometer a capacidade representacional!

Galeria de Arquiteturas Notáveis

A linha do tempo da inovação: da força bruta ao escalonamento composto

VGG-16 / VGG-19 (2014)

- Pilhas homogêneas de filtros pequenos 3×3.
- Excelente para extração de features e Neural Style Transfer.
- Muito pesada: 138M parâmetros (dominada por camadas FC).

DenseNet (2017)

- Conecta cada camada a todas as camadas seguintes via concatenação.
- Reutilização máxima de características (Feature Reuse).
- Excelente eficiência de parâmetros, mas alto consumo de VRAM.

MobileNet Family (v1-v3)

- Inverted Residuals & Linear Bottlenecks (v2).
- Otimização por Neural Architecture Search (NAS) e h-swish (v3).
- Projetada para smartphones, drones e microcontroladores.

EfficientNet (v1-v2)

- Compound Scaling: balanceia profundidade (d), largura (w) e resolução (r).
- Máxima acurácia no ImageNet com até 8.4x menos parâmetros.
- Padrão estado da arte em eficiência antes dos ViTs.

Evolução: Força Bruta (VGG) → Atalhos (DenseNet) → Edge Computing (MobileNet) → Escalonamento Guiado por AutoML (EfficientNet).

Como Escolher a Arquitetura CNN Certa?

Navegando pela Tabela 12-3 do TorchVision: Acurácia Top-1/Top-5 vs Parâmetros vs GFLOPs

Porte: **Todos (16)** | Mobile & Edge (<10M) | Equilibrados (10M-50M) | Alta Capacidade (>50M)

Ordenar por: Tamanho (Parâmetros ↑)

Tabela 12-3: Modelos Pré-Treinados no TorchVision (Ordenados por Tamanho) — 16 modelos exibidos

Modelo (Class Name)	Top-1 Acc	Top-5 Acc	Params	GFLOPs
MobileNet v3 small	67.7%	87.4%	2.5M	0.1
EfficientNet B0	77.7%	93.5%	5.3M	0.4
GoogLeNet	69.8%	89.5%	6.6M	1.5
DenseNet 121	74.4%	92.0%	8.0M	2.8
EfficientNet v2 small	84.2%	96.9%	21.5M	8.4
ResNet 34	73.3%	91.4%	21.8M	3.7
Inception V3	77.3%	93.5%	27.2M	5.7
ConvNeXt Tiny	82.6%	96.1%	28.6M	4.5
DenseNet 161	77.1%	93.6%	28.7M	7.7
ResNet 152	82.3%	96.0%	60.2M	11.5
AlexNet	56.5%	79.1%	61.1M	0.7
EfficientNet B7	84.1%	96.9%	66.3M	37.8
ResNeXt 101 32×8D	82.8%	96.2%	88.8M	16.4
EfficientNet v2 large	85.8%	97.8%	118.5M	56.1
VGG 11 with BN	70.4%	89.8%	132.9M	7.6
ConvNeXt Large	84.4%	97.0%	197.8M	34.4

Intermediária

Padrão da Indústria

ResNet 34

A espinha dorsal mais versátil e confiável para Transfer Learning em visão computacional geral.

Top-1 Acc	Top-5 Acc	Parâmetros	GFLOPs
73.3%	91.4%	21.8M	3.7

</> Chamada no PyTorch (Weights Enum):

```
from torchvision.models import resnet34, ResNet34_Weights
model = resnet34(weights=ResNet34_Weights.DEFAULT)
```

💡 **Recomendação de Uso:** Linha de base sólida para classificação, detecção e segmentação.

Memória GPU RAM: Inferência vs Treinamento

Entendendo a física da VRAM para evitar erros CUDA Out of Memory (OOM)

Inferência (torch.no_grad)

~120 MB VRAM

Apenas pesos estáticos + ativação da camada atual. Ativações anteriores são descartadas imediatamente.

✓ Complexidade de Memória: O(1)

Treinamento (Backpropagation)

~8.5 GB VRAM

TODOS os mapas de ativação de TODAS as camadas precisam ser salvos para a regra da cadeia no backward pass.

⚠️ Complexidade de Memória: O(L × Batch)

Onde a GPU RAM Realmente Vai no Treinamento? (Batch=64)

ResNet-50



- **Pesos (W):** ~100 MB (4%)
- **Gradientes (dW):** ~100 MB (4%)
- **Adam (m, v):** ~200 MB (8%)
- **Ativações:** ~7.2 GB (84%) 🔥

💡 **Regra de Ouro:** Se você tiver erro OOM (Out of Memory), reduza o `batch_size` ou use *Gradient Checkpointing / RevNets*.

TORCHVISION MODERNO | Weights Enum API

Modelos Pré-Treinados no TorchVision

Aproveitando pesos de alta precisão e pipelines de transformações automáticas

Nova API de Pesos (torchvision.models)

- Substitui o antigo e ambíguo pretrained=True.
- Padrão moderno com enum de checkpoint explícito:
- from torchvision.models import resnet34, ResNet34_Weights
- weights = ResNet34_Weights.DEFAULT
- model = resnet34(weights=weights)

Transformações Automáticas Perfeitas

- Garante a normalização exata (média e desvio padrão do ImageNet):
- preprocess = weights.transforms()
- tensor = preprocess(imagem_pil)
- Evita bugs silenciosos de pré-processamento incorreto em produção.

Regra de Ouro: Sempre utilize weights.transforms() para garantir que a imagem de teste receba o mesmo tratamento estatístico usado no treinamento original do modelo.

TRANSFER LEARNING | Laboratório Interativo

Transfer Learning: Feature Extraction vs Fine-Tuning

Simule o congelamento de parâmetros, troca do classificador e taxas de aprendizado diferenciais

🔒 Feature Extraction (Backbone Fixo) | ⚡ Fine-Tuning Parcial (Layer4 + FC) | 🔥 Fine-Tuning Completo

Classes de Saída: **2** — 2 (Binário: Ex. Cão vs Gato)

Estado dos Parâmetros no PyTorch

ResNet-34 Backbone

🔒	**conv1 + bn1** Camada de Entrada (Low-level edges/colors)	requires_grad = False
🔒	**layer1 (3 blocks)** Texturas e Padrões Básicos	requires_grad = False
🔒	**layer2 (4 blocks)** Motivos e Partes Simples	requires_grad = False
🔒	**layer3 (6 blocks)** Estruturas de Objetos Complexos	requires_grad = False
🔒	**layer4 (3 blocks)** Semântica de Alto Nível / Classes	requires_grad = False
🔓	**fc (Classifier)** Nova Cabeça Customizada (2 classes)	requires_grad = True

💡 *Dica:* Bloquear camadas iniciais preserva detectores de borda universais e reduz o tempo de treinamento em mais de 70%.

script_transfer_learning.py — Copiar

```
import torch
import torch.nn as nn
from torchvision.models import resnet34, ResNet34_Weights

# 1. Carrega modelo pré-treinado no ImageNet-1k
weights = ResNet34_Weights.DEFAULT
model = resnet34(weights=weights)

# 2. Congela todos os parâmetros do backbone
for param in model.parameters():
    param.requires_grad = False

# 3. Substitui a cabeça de classificação (fc)
in_features = model.fc.in_features  # 512
model.fc = nn.Linear(in_features, 2)

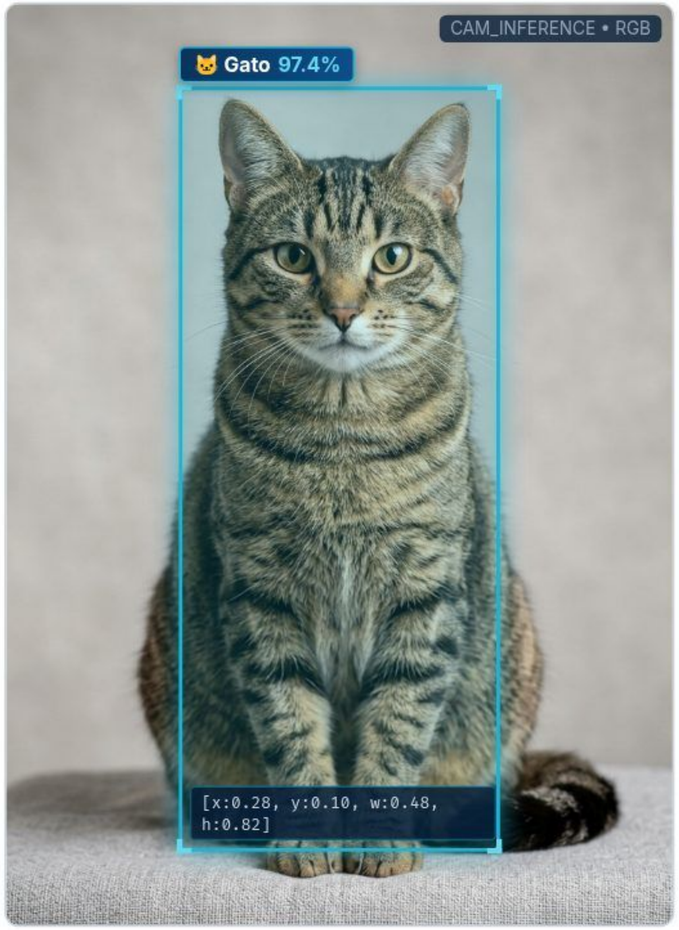
# 4. Apenas os pesos de model.fc serão atualizados no otimizador
optimizer = torch.optim.Adam(model.fc.parameters(), lr=1e-3)
```

Classificação com Localização de Objeto Único

Prevendo simultaneamente a categoria e as coordenadas contínuas da Bounding Box [x, y, w, h]

1. Imagem de Entrada

Tensor [3, 224, 224]



Comparar IoU com Ground Truth

2. Backbone ResNet

Compartilhado

Stem + Residual Blocks
Convoluções 7×7, 3×3 + MaxPool

↓

Estágios Residuais (1 a 4)
Mapas espaciais 7×7 × 512 canais

↓

Global Average Pooling (GAP)
Achata tensor 7×7 para vetor 1D

↓

VETOR DE FEATURES LATENTE
[512 dimensões]

💡 Um único forward pass extrai representação rica para ambas as tarefas.

3. Dupla Cabeça + Perdas

Multitarefa

Cabeça 1: Classe ("O quê?")

Linear(512, C) → Softmax

Gato:	97.4%
Cão:	1.8%
Pássaro:	0.8%

Cross-Entropy Loss:

$$\mathcal{L}_{CE} = -\sum y_i \log(\hat{y}_i)$$

Cabeça 2: BBox ("Onde?")

Linear(512, 4) → Regressão

x: 0.28	y: 0.10
w: 0.48	h: 0.82

Normalizado entre [0.0, 1.0]

Smooth L1 Loss:

$$\mathcal{L}_{\text{Smooth L1}}(\text{box}, \hat{\text{box}})$$

Função de Custo Total (Multitarefa):

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{CE}} + \lambda \cdot \mathcal{L}_{\text{Smooth L1}}$$

Métrica IoU:
IoU = 0.91 (≥ 0.50 ✅)

Detecção de Objetos: Two-Stage vs One-Stage

Localizando múltiplos objetos através de propostas de região versus grade direta em tempo real

Two-Stage (Faster R-CNN)

Foco: Máxima Precisão (mAP)

- Etapa 1 (RPN): Rede escaneia e gera ~2000 regiões candidatas de interesse.
- Etapa 2 (RoI Align): Recorta, redimensiona e classifica cada proposta.
- Desempenho: 5 a 15 FPS (mais lento, porém cirúrgico em objetos pequenos).

📌 **Ideal para exames médicos, satélites e imagens de altíssima resolução.**

One-Stage (Família YOLO / SSD)

Foco: Tempo Real (30 a 120+ FPS)

- Passada Única: Divide a imagem em grade e prevê caixas e classes diretamente.
- Filtragem NMS: Elimina caixas sobrepostas com base em confiança e IoU.
- Desempenho: Altíssima velocidade para inferência em borda e vídeo em tempo real.

🚀 **Ideal para carros autônomos, robôs móveis e monitoramento urbano.**

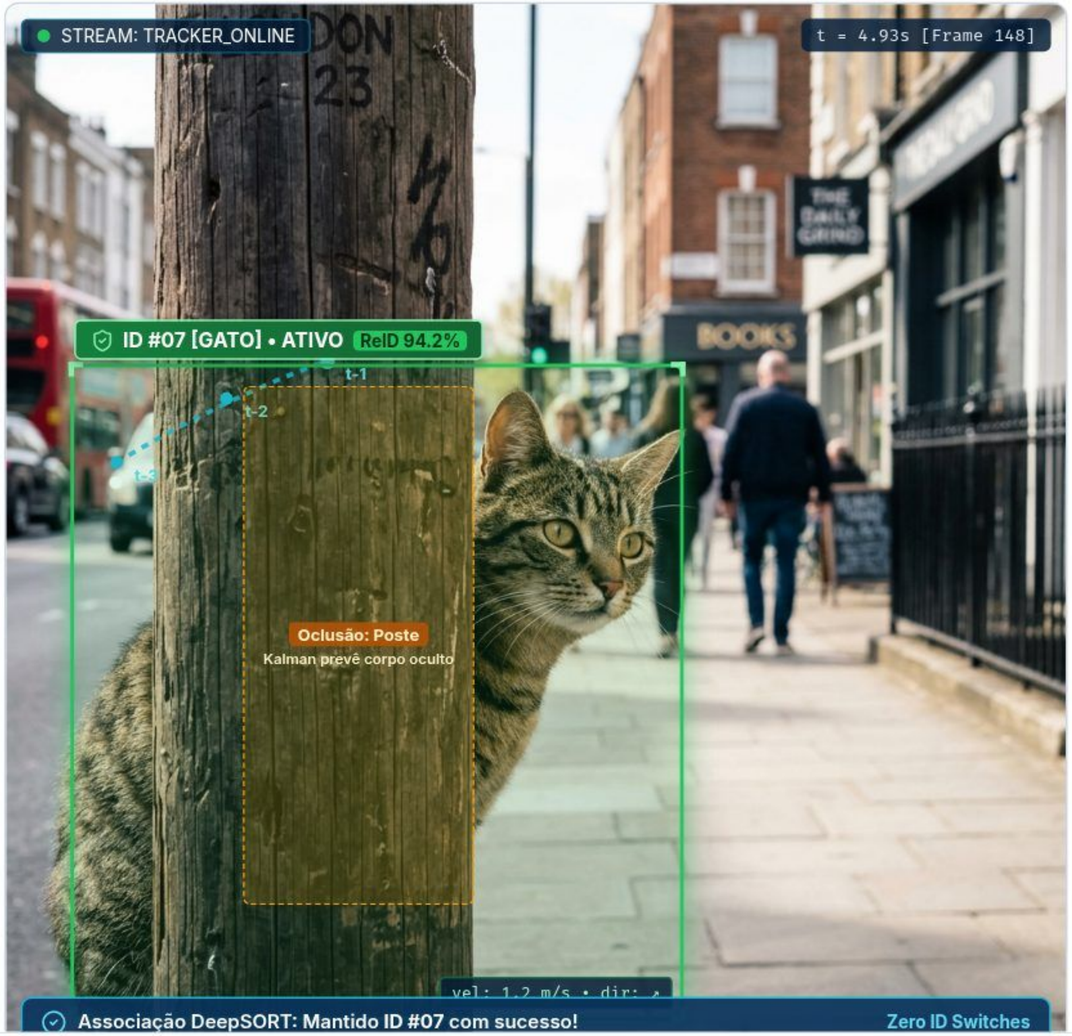
VISÃO ESPAÇOTEMPORAL | DeepSORT • Fluxo de Vídeo

Rastreamento de Objetos (Object Tracking)

Mantendo identificadores únicos e trajetórias consistentes mesmo com oclusões temporárias

Câmera de Monitoramento • Frame #148

30 FPS • Ao Vivo



1. Estimativa Cinemática: Filtro de Kalman

Trajetória Espacial

Modela posição [x, y], proporção de aspecto e velocidades lineares [v_x, v_y]. Prediz onde o gato estará no próximo instante mesmo quando oculto pelo poste.

Métrica Espacial (Mahalanobis): $d^{(1)}(i,j) \leq t^{(1)}$

2. Re-Identificação Visual (ReID Embedding)

Assinatura Visual

CNN siamesa extrai um vetor compacto de 128 dimensões da pelagem/face do gato. Compara a similaridade de cosseno com a galeria dos últimos 100 frames.

Similaridade Cosseno: $r_j^T r_k^{(i)} \geq 0.85$ **(Match: 94.2% ✅)**

3. Associação Ótima: Algoritmo Húngaro

Atribuição 1-para-1

Combina proximidade cinemática + aparência visual em uma matriz de custo ponderada. Resolve o problema de atribuição linear em tempo polinomial.

$$C_{i,j} = \lambda \cdot D_{\text{Kalman}} + (1 - \lambda) \cdot D_{\text{ReID}}$$

VISÃO DENSA | Predição por Pixel • U-Net

Segmentação Semântica: U-Net & FCN

Arquiteturas Encoder-Decoder com skip connections para geração de máscaras de alta resolução

Estrutura em "U" (Encoder-Decoder)

U-Net (2015)

Encoder L1 (224×224) — Skip Connection Longa —▶ Decoder L1 (224×224)

Encoder L2 (112×112) — Skip Connection —▶ Decoder L2 (112×112)

⚡ Bottleneck Central (Semântica Profunda: 28×28)

As pontes horizontais (skip connections) transferem contornos nítidos para o decoder.

Classificação por Pixel (Máscara Densa)

Saída [C, H, W]

Função de Custo Cross-Entropy por Pixel:

$$\mathcal{L}_{\text{pixel}} = -\frac{1}{HW}\sum_{i=1}^{H}\sum_{j=1}^{W}\sum_{c=1}^{C} y_{i,j,c}\log(\hat{y}_{i,j,c})$$

Produz máscaras com **a mesmíssima resolução** da imagem original!

SÍNTESE VISUAL | Laboratório Interativo

Comparativo Integrado: As 5 Grandes Tarefas de Visão

Visualize a mesma cena do mundo real sob a perspectiva de cada paradigma de visão computacional

Classificação Simples | Classificação + Localização | Detecção de Objetos (YOLO) | Rastreamento (Tracking) | Segmentação Semântica



1. Classificação Simples

Uma imagem → Uma classe

Prevê uma distribuição de probabilidade global sobre classes discretas para a imagem inteira.

Formato do Tensor de Saída:

```
[Batch, Num_Classes]
```

Função de Custo Típica (Loss):

$$\mathcal{L}_{CE} = -\sum y_i \log(\hat{y}_i)$$

CONSOLIDAÇÃO Desafio Prático

Quiz Interativo de Engenharia de CNNs

Teste seus conhecimentos práticos sobre arquiteturas, fluxo de gradientes e gerenciamento de hardware

Questão 1 de 3 Pontuação: **0 / 3** Reiniciar

Por que o bloco residual da ResNet (F(x) + x) permite treinar redes com mais de 100 camadas sem o problema de degradação?

- Porque ele reduz o número de parâmetros da rede a quase zero através de convoluções 1×1.
- Porque a conexão de identidade cria um caminho direto para o gradiente fluir no backpropagation (termo + 1), evitando o desvanecimento.
- Porque ele descarta metade dos canais da imagem antes de aplicar as convoluções 3×3.
- Porque ele substitui a função de ativação ReLU por Sigmoid em todas as camadas intermediárias.

Confirmar Resposta