Inteligência Artificial 2

Lab 07: Conclusões e Análise Empírica

Prof. Aléssio Miranda Júnior
alessio@cefetmg.br

CEFET-MG - Campus Timoteo
Dep. Engenharia de Computação

Transfer Learning com MobileNetV2

Onde Paramos no Laboratório?

No **Lab 07 (O Radar Aéreo de TechCity)**, exploramos um estudo de caso aplicado à **visão computacional (imagens)**: classificar **Avião (0) vs. Automóvel (1)** com uma fatia reduzida do dataset **CIFAR-10**.

Para isso, adotamos a **MobileNetV2** (leve e ideal para CPU), cientes de que o ecossistema oferece dezenas de outros backbones consolidados (ResNet, EfficientNet, ConvNeXt, VGG).

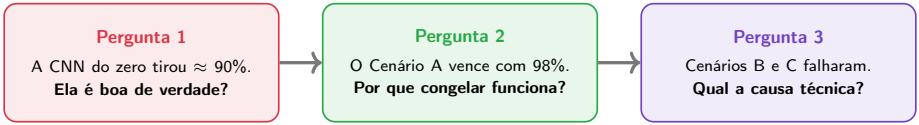
Ao longo do roteiro, executamos 4 configurações arquiteturais:

- **Cenário 0:** CNN simples do zero com pesos aleatórios.
- **Cenário A (Padrão Ouro):** MobileNetV2 congelada + `GlobalAveragePooling2D`.
- **Cenário B (Descongelado):** MobileNetV2 descongelada logo no início do treino.
- **Cenário C (Flatten):** Substituição do GAP por uma camada `Flatten()`.

Objetivo Desta Sessão

Compreender os fundamentos teóricos e matemáticos que explicam os resultados observados nesse estudo de caso de imagens.

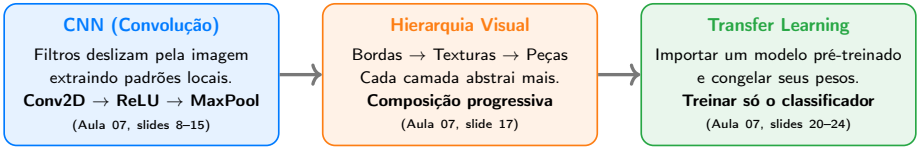
3 Perguntas que Vamos Responder Hoje



Roteiro

Vamos percorrer cada cenário na ordem em que vocês os executaram, parando para construir os conceitos teóricos **quando** eles forem necessários para explicar o que aconteceu.

Revisão Rápida: O que Já Sabemos da Aula Teórica



Esses 3 conceitos são os **pré-requisitos** para entender tudo que vem a seguir. Agora vamos confrontá-los com os dados reais que vocês obtiveram no laboratório.

Cenário 0: A CNN do Zero — O Resultado Surpreendente

No Cenário 0, vocês construíram uma **CNN simples** (2 camadas Conv2D + MaxPool + Dense) com pesos 100% aleatórios. O esperado era que ela tivesse dificuldade com apenas 10.000 imagens...

A Pergunta Essencial

Será que essa CNN realmente **aprendeu** a distinguir aviões de carros? Ou será que ela encontrou um **atalho**?

Por que 90%? O Fenômeno do Shortcut Learning

A CNN simples **não aprendeu a geometria** de asas ou rodas. Ela encontrou um **atalho estatístico** baseado nas cores de fundo:

Avião no CIFAR-10

$\approx$ 90% das fotos têm fundo **azul (céu)** ou branco (nuvens)

Carro no CIFAR-10

$\approx$ 90% das fotos têm fundo **cinza/preto (asfalto)** ou cenário urbano

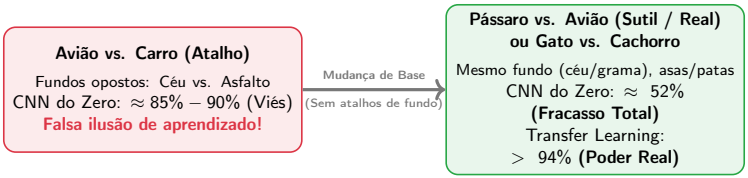
A rede memorizou: **"muito pixel azul no topo $\rightarrow$ Avião"**. Isso não é *visão computacional* — é *detecção de cor de fundo*!

Shortcut Learning (Aprendizado por Atalho)

Fenômeno bem documentado na literatura: a rede explora correlações espúrias (cor de fundo, textura do dataset) em vez de aprender as features discriminativas reais do objeto (Geirhos et al., 2020).

A Mudança de Base: Revelando o Fracasso Real

Para eliminar o atalho e expor a limitação real da CNN do zero, basta escolher pares de classes onde o fundo **não ajuda**:

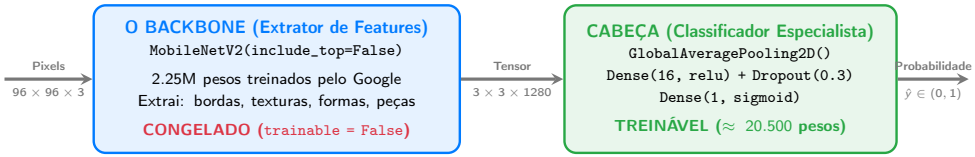


Resposta à Pergunta 1

Não, a CNN do zero **não aprendeu morfologia visual**. Ela apenas **memorizou uma correlação de contexto**. Quando forçamos uma tarefa sem atalhos de cor, ela colapsa para o chute aleatório ($\approx 50\%$), comprovando que o Transfer Learning é indispensável.

Anatomia do Transfer Learning: Backbone e Cabeça

Já que a CNN do zero falha, precisamos de uma estratégia melhor. O **Transfer Learning** divide o modelo em duas metades cirúrgicas:



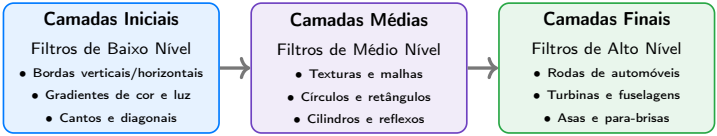
- O **Backbone** converte matrizes de pixels em um **vetor de conceitos visuais**.
- A **Cabeça** apenas aprende uma regra lógica: *"Se tiver asas → Avião; se tiver rodas → Carro"*.

De onde vêm os ≈ 20.500 pesos treináveis?

- `GlobalAveragePooling2D`: comprime $3 \times 3 \times 1280 \rightarrow 1.280$ escalares (**0 pesos**).
- `Dense(16)`: $(1.280 \times 16) + 16$ (bias) $= 20.496$ pesos.
- `Dense(1)`: $(16 \times 1) + 1$ (bias) $= 17$ pesos.
- **Total Exato:** $20.496 + 17 = 20.513$ parâmetros treináveis!

Por que o Backbone do Google é Poderoso?

A `MobileNetV2` foi treinada no **ImageNet** (1,4 milhão de imagens, 1.000 classes). Seus filtros formam uma **hierarquia de abstração** progressiva:

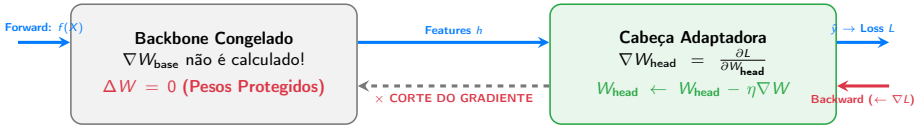


A Grande Sacada

As camadas iniciais e médias são **universais**: qualquer problema de visão computacional precisa detectar bordas e texturas. Não precisamos reaprendê-las do zero — basta **herdar** e **proteger** esses filtros!

A Mecânica do Congelamento (`trainable=False`)

Proteger os filtros do Google significa **impedir que o Backpropagation os altere**. Quando executamos `base_model.trainable = False`:



- O gradiente flui da Loss até a cabeça adaptadora.
- Na fronteira com o backbone, o TensorFlow **interrompe o cálculo dos gradientes**.
- O Backbone atua como uma **função determinística estática** $f(X)$ — um extrator fixo.

Cenário A: O Padrão Ouro — Código e Resultados

```python
def criar_modelo_transfer():
    # 1. Cerebro pre-treinado (sem cabeca)
    base_model = MobileNetV2(
        weights='imagenet',
        include_top=False,
        input_shape=(96, 96, 3))

    # 2. CONGELAMENTO
    base_model.trainable = False

    # 3. CABECA ADAPTADORA
    modelo = Sequential([
        base_model,
        GlobalAveragePooling2D(),
        Dense(16, activation='relu'),
        Dropout(0.3),
        Dense(1, activation='sigmoid')
    ])

    modelo.compile(
        optimizer='adam',
        loss='binary_crossentropy',
        metrics=['accuracy'])
    return modelo, base_model
```

Resultado

Acurácia: 98,1%

Loss: 0.0481

Treináveis: 20.513

Tempo/Época: $\approx$ 5s (CPU)

Treinamos apenas 0, 9% da rede e superamos a CNN do zero com folga!

Cenário B: A Armadilha do Descongelamento Precoce

Se congelar funciona tão bem, surge a tentação: *"E se eu descongelar tudo e deixar a rede inteira aprender?"*

No Cenário B, fizemos exatamente isso: `base_model.trainable = True` **desde o início**.

Resultado: Acurácia caiu para 91,2% (pior que o Cenário A!)

Descongelar **piorou** a rede. Mas por quê? A resposta está na **matemática do Backpropagation**.

A Matemática: Por que o Gradiente Destrói os Pesos?

Quando o backbone está descongelado, o Backpropagation propaga o erro pela **Regra da Cadeia**:

$$\frac{\partial L}{\partial W_{\text{base}}} = \underbrace{\frac{\partial L}{\partial \hat{y}}}_{\text{Erro Máximo!}} \cdot \underbrace{\frac{\partial \hat{y}}{\partial h_{\text{head}}} \cdot \frac{\partial h_{\text{head}}}{\partial h_{\text{base}}}}_{\text{Pesos aleatórios caóticos}} \cdot \frac{\partial h_{\text{base}}}{\partial W_{\text{base}}}$$

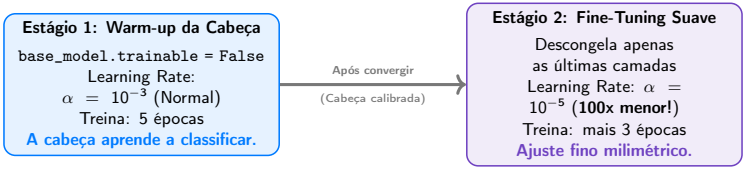
1. **Época 1, Lote 1:** Cabeça densa com pesos aleatórios gera $\hat{y} \approx 0.5 \rightarrow \frac{\partial L}{\partial \hat{y}}$ é **enorme**.
2. O gradiente atravessa os pesos caóticos da cabeça, gerando **ruído amplificado**.
3. Atualização no backbone: $W_{\text{base}} \leftarrow W_{\text{base}} - \eta \cdot \frac{\partial L}{\partial W_{\text{base}}}$.
4. Os $2,2$M pesos pré-treinados levam um **choque descalibrado**, destruindo filtros de bordas e texturas.

Catastrophic Forgetting (Esquecimento Catastrófico)

A rede "esquece" o conhecimento prévio refinado do ImageNet em poucos lotes de treino.

A Solução: Fine-Tuning em 2 Estágios

Se quisermos descongelar o backbone com segurança, existe um **protocolo obrigatório** na indústria:



Regra de Ouro

Nunca descongele sem antes fazer o warm-up da cabeça, e **sempre** use um Learning Rate $100\times$ menor (10^{-5}) para não destruir os pesos pré-treinados.

Como Implementar na Prática: Salvar e Retomar

Veja como implementar o fluxo de 2 estágios com persistência de checkpoints em disco:

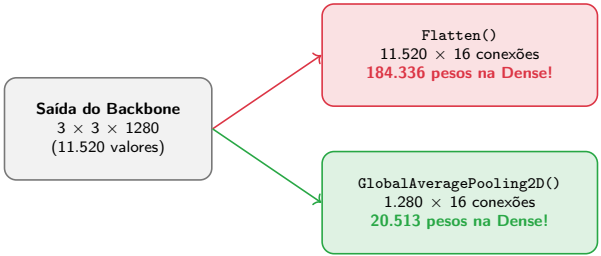
```
# --- ESTAGIO 1: Warm-up da cabeca (base congelada) ---
modelo.compile(optimizer=Adam(learning_rate=1e-3), loss='binary_crossentropy', metrics=['accuracy'])
modelo.fit(X_train, y_train, epochs=5, validation_data=(X_val, y_val))

# SALVA O CHECKPOINT DO ESTAGIO 1 (garantia de rollback!)
modelo.save("modelo_estagio1_warmup.keras")

# --- ESTAGIO 2: Descongelamento cirurgico + Retomada ---
base_model.trainable = True
for layer in base_model.layers[:-20]: # descongela apenas as ultimas 20 camadas
    layer.trainable = False

# OBRIGATORIO: Recompilar com Learning Rate 100x MENOR!
modelo.compile(optimizer=Adam(learning_rate=1e-5), loss='binary_crossentropy', metrics=['accuracy'])

# Continua o treino a partir dos pesos ja calibrados
modelo.fit(X_train, y_train, epochs=3, validation_data=(X_val, y_val))
modelo.save("modelo_estagio2_final.keras")
```

Boas Práticas de Engenharia

Salvar o checkpoint (`.keras`) ao fim do Estágio 1 evita reiniciar do zero se o Estágio 2 divergir.

Cenário C: A Explosão Dimensional do Flatten

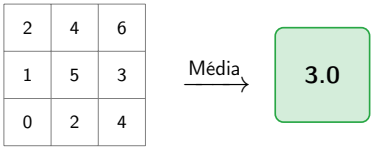
No Cenário C, mantivemos o backbone congelado (como o A), mas trocamos `GlobalAveragePooling2D()` por `Flatten()`. Veja o que aconteceu com o número de conexões:



- O `Flatten()` **achata** todos os $3 \times 3 \times 1280 = 11.520$ valores em um único vetor.
- Isso multiplica o número de parâmetros treináveis em quase $10\times$ (184k vs 20k).
- Resultado: acurácia caiu para $95,5\%$ e o modelo ficou mais lento e vulnerável a overfitting.

Como o GAP Resolve: A Matemática da Contração Espacial

O `GlobalAveragePooling2D` resolve a explosão com uma operação elegante — a **média aritmética** por canal:



$\text{GAP}_k = \frac{1}{H \times W} \sum_{i,j} X_{i,j,k}$

Soma dos 9 pixels: 27

Média: $27/9 = 3.0$

Zero parâmetros adicionais!

- O GAP resume a **intensidade média** de cada filtro convolucional em toda a imagem.
- Transforma $3 \times 3 \times 1280$ em um vetor de 1280 valores — sem criar nenhum peso novo.
- Elimina a dependência de posições fixas de pixels (**invariância à translação**).

Comparativo Completo: GAP vs. Flatten

Propriedade	`GlobalAveragePooling2D()`	`Flatten()`
Tamanho do Vetor Gerado	1.280 neurônios	11.520 neurônios
Parâmetros da Camada Densa	$\approx$ 20.500	$\approx$ 184.300
Sensibilidade à Resolução	Aceita resoluções dinâmicas	Exige tamanho rígido
Invariância à Posição	**Alta** (média global)	Baixa (depende do índice exato)
Risco de Overfitting	**Quase Nulo**	Alto
Padrão na Indústria	**Sim (Moderno)**	Obsoleto para CNNs profundas

Resposta à Pergunta 3 (Cenário C)

O Cenário C perdeu para o A porque o `Flatten()` introduz 9$\times$ mais parâmetros sem ganho de informação, aumentando o overfitting e destruindo a invariância espacial.

O Painel Completo: Os 4 Cenários Lado a Lado

Agora que entendemos a causa de cada resultado, vamos consolidar o diagnóstico:

Cenário 0
CNN do Zero
553k params

Cenário A
Congelado + GAP
20k params (Ouro)

Cenário B
Descongelado
2.27M params (Erro)

Cenário C
Flatten sem GAP
184k params (Explosão)

Cen.	Estratégia	Params. Treináveis	Loss	Acurácia	Diagnóstico (Causa Raiz)
0	CNN Conv2D do Zero	553.233 (100%)	0.6931	55.20%*	Shortcut Learning / Sem features
A	MobileNetV2 + GAP + Freeze	20.513 (0,9%)	0.0481	98.10%	**Padrão Ouro**
B	MobileNetV2 Descongelada	2.278.497 (100%)	0.1241	91.20%	Catastrophic Forgetting
C	MobileNetV2 + Flatten()	184.336 (7,6%)	0.0825	95.50%	Explosão Dimensional

*Em datasets sem viés de fundo / amostras reduzidas.

Os 3 Mandamentos do Transfer Learning

Na variável `CONCLUSAO_ALUNO` do código, o engenheiro sintetiza as 3 lições que acabou de comprovar empiricamente:

1. Congele Primeiro

Sempre treine a cabeça com a base congelada (`trainable=False`). Os pesos pré-treinados são valiosos demais para sofrer o gradiente caótico de uma cabeça aleatória.

Cenário B prova o custo de ignorar.

2. Use GAP

Sempre utilize `GlobalAveragePooling2D` antes de camadas densas. Reduz $11.520 \rightarrow 1.280$ sem parâmetros e dá invariância espacial.

Cenário C mostra a explosão do Flatten.

3. LR Baixo no Ajuste

Se for descongelar para fine-tuning, use Learning Rate $100\times$ menor (10^{-5}) e apenas após o warm-up da cabeça.

Protocolo de 2 Estágios da indústria.

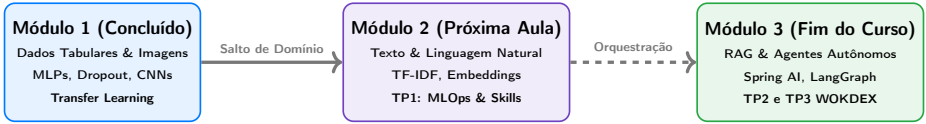
Telemetria MLOps

O histórico registrado no CSV atesta a reprodutibilidade e a governança das hipóteses testadas no pipeline.

Recapitulação da Sessão (Takeaways)

- **Cuidado com Shortcut Learning:** Acurácia alta nem sempre significa aprendizado real — verifique se a rede aprendeu features ou atalhos de contexto (Cenário 0 no CIFAR-10).
- **Backbone = Extrator Universal de Features:** Redes pré-treinadas (MobileNetV2, ResNet, etc.) transformam tensores brutos em representações semânticas ricas.
- **Freezing = Economia e Proteção:** Treinar $< 1\%$ da rede economiza 99% da computação e evita Catastrophic Forgetting (Cenário A vs. B).
- **GAP = Redução Inteligente:** Condensa mapas espaciais sem explodir parâmetros nem amarrar posições (Cenário A vs. C).
- **Generalização do Conceito:** Embora tenhamos testado em **imagens**, esse mesmo paradigma de Transfer Learning sustenta o estado da arte em **áudio, séries temporais e LLMs de texto**.

Onde Vamos Parar? (Conexão com o Módulo 2)



O Gancho para a Próxima Aula

Você dominou a manipulação de tabelas e tensores de imagens. Na próxima aula, entraremos no universo do **Processamento de Linguagem Natural (NLP)**: como transformar palavras e enunciados de código em tensores numéricos para alimentar a IA do seu TP1!