Aula 7: Redes Convolucionais e Transfer Learning

Aléssio Miranda Júnior

Agosto - 2026/2

Capítulo em Revisão

Este conteúdo está sendo revisado. Algumas informações podem estar preliminares, conter erros de formatação ou sofrer alterações na versão final.

O que você verá neste capítulo

Até este ponto, todos os nossos modelos operam sobre dados tabulares — vetores unidimensionais de números organizados em planilhas CSV. Neste capítulo, rompemos essa barreira e entramos no domínio da **Visão Computacional**, onde a entrada não é uma linha de tabela, mas sim uma imagem inteira composta por milhões de pixels. Exploraremos o motivo pelo qual redes densas (MLP) fracassam em imagens de alta resolução, formalizaremos a operação matemática de **Convolução** e a arquitetura das CNNs (LeCun et al., 1998), e fecharemos com o paradigma mais transformador da indústria moderna: o **Transfer Learning** — a capacidade de reaproveitar cérebros treinados por terceiros.

1 O Limite das Redes Densas

Até agora, utilizamos arquiteturas Multilayer Perceptron (MLP) compostas exclusivamente por camadas Densas (`Dense`). Para dados tabulares em CSV, com dezenas ou centenas de colunas numéricas, elas funcionam maravilhosamente bem. Contudo, quando tentamos aplicá-las a imagens de alta resolução, encontramos um bloqueio arquitetural intransponível.

Uma imagem colorida de 1000×1000 pixels não é uma lista simples de números, mas sim um **tensor tridimensional** de $1000 \times 1000 \times 3$ (largura $\times$ altura $\times$ canais RGB), totalizando **3 milhões de variáveis de entrada**. Se a primeira camada oculta tiver apenas 1.000 neurônios, a quantidade de pesos (W) su-

perará 3×10^9 (3 bilhões), inviabilizando o treinamento em computadores convencionais.

A tabela a seguir ilustra a escala do problema:

Tipo de Entrada	Variáveis	Pesos (1000 neurônios)
CSV (30 colunas)	30	30.000
MNIST (28×28, cinza)	784	784.000
Foto celular (4000×3000, RGB)	36.000.000	**36 bilhões**

Tabela 1: Explosão paramétrica: o número de pesos cresce multiplicativamente com as dimensões da entrada. Para fotos de alta resolução, uma camada Dense torna-se computacionalmente impossível.

Além do custo computacional, há um problema conceitual mais grave: a camada Densa **destrói a estrutura espacial da imagem**. Ao achatar (*flatten*) a imagem em um vetor 1D, a rede perde toda informação sobre quais pixels são vizinhos, quais formam bordas e quais compõem objetos. Um gato continua sendo um gato independentemente de estar no canto superior ou no centro da foto — mas para a camada Densa, essas são entradas completamente diferentes.

△ Importante: A Explosão Combinatória dos Pesos

Para uma imagem modesta de 28×28 pixels em escala de cinza (como o dataset MNIST de dígitos manuscritos), uma camada densa com 128 neurônios gera $28 \times 28 \times 128 = 100.352$ pesos. Para uma foto de câmera de celular ($4000 \times 3000 \times 3$), a mesma camada geraria mais de **4,6 bilhões** de parâmetros. É computacionalmente impossível e estatisticamente absurdo.

2 Redes Neurais Convolucionais (CNNs)

A solução adotada pela IA moderna para a visão computacional são as **Redes Neurais Convolucionais** (CNNs — *Convolutional Neural Networks*). A arquitetura foi formalizada por Yann LeCun em 1998 com a rede LeNet-5, inspirada no funcionamento do córtex visual dos mamíferos descoberto por Hubel e Wiesel (1962), que receberam o Nobel de Medicina por esse trabalho.

A ideia fundamental é: em vez de conectar cada pixel a cada neurônio (conexão densa), usamos **Filtros** (também chamados de *Kernels*) — pequenas matrizes (tipicamente 3×3 ou 5×5) que **deslizam** (convolucionam) pela superfície da imagem, procurando padrões locais específicos.

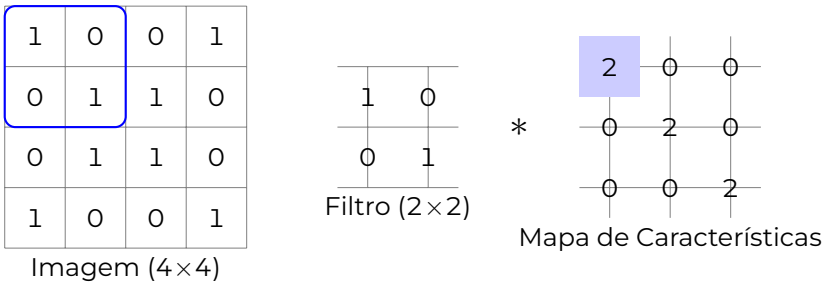


Figura 1: Processo de convolução: o filtro 2×2 desliza pela imagem, calculando o produto escalar em cada posição. O resultado é um Mapa de Características (*Feature Map*) que indica onde o padrão foi detectado.

2.1 A Hierarquia de Abstração

O poder das CNNs emerge quando empilhamos múltiplas camadas convolucionais. Cada camada detecta padrões progressivamente mais abstratos:

1. **Camadas Iniciais (Baixo Nível):** Detectam primitivas geométricas — bordas horizontais, verticais, diagonais e gradientes de cor.
2. **Camadas Intermediárias (Médio Nível):** Combinam as bordas para detectar texturas e formas (olhos, orelhas, rodas, janelas).
3. **Camadas Profundas (Alto Nível):** Reconhecem objetos complexos completos (rostos, carros, cachorros, edifícios).

• Teoria: Compartilhamento de Pesos — O Segredo da Eficiência

A grande vantagem computacional da CNN é o **compartilhamento de pesos**: o mesmo filtro 3×3 (apenas 9 parâmetros) é reutilizado em **toda a extensão** da imagem. Isso reduz drasticamente o número de parâmetros em comparação com uma camada Densa. Uma camada convolucional com 32 filtros de 3×3 em uma imagem com 3 canais de cor tem apenas $32 \times (3 \times 3 \times 3 + 1) = 896$ parâmetros — contra os bilhões da camada Densa.

2.2 Pooling e a Redução Dimensional

Após cada convolução, aplicamos uma camada de **Pooling** (tipicamente *Max Pooling* 2×2), que reduz as dimensões espaciais pela metade, mantendo apenas os valores mais relevantes de cada região. Isso diminui o custo computacional progressivamente e aumenta a invariância posicional do modelo.

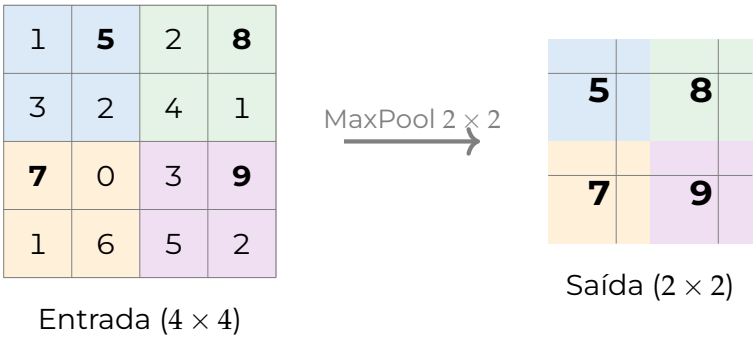


Figura 2: Operação de Max Pooling 2 × 2: cada região colorida é reduzida ao seu valor máximo. A dimensão espacial cai pela metade, mas a informação mais relevante é preservada.

No Keras, a arquitetura de uma CNN clássica é:

```
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.layers import Flatten, Dense

model = Sequential()
# Camada Convolucional: 32 filtros de 3x3
model.add(Conv2D(32, (3,3), activation='relu',
                 input_shape=(28, 28, 1)))
model.add(MaxPooling2D((2,2)))

# Segunda camada convolucional
model.add(Conv2D(64, (3,3), activation='relu'))
model.add(MaxPooling2D((2,2)))

# Transicao: achatar para a camada densa final
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))  # 10 classes
```

3 A Filosofia do Transfer Learning

Treinar uma rede CNN gigante do zero, como a célebre **ResNet-50** (He et al., 2015) com 50 camadas profundas e 25 milhões de parâmetros, para que ela aprenda a identificar desde arranha-céus até raças de cães, custaria **centenas de milhares de dólares** em aluguel de GPUs na nuvem por semanas.

Na engenharia de IA corporativa, nós **não reinventamos a roda**. Aplicamos o paradigma do **Transfer Learning** (Transferência de Aprendizado), um dos conceitos mais transformadores da IA moderna.

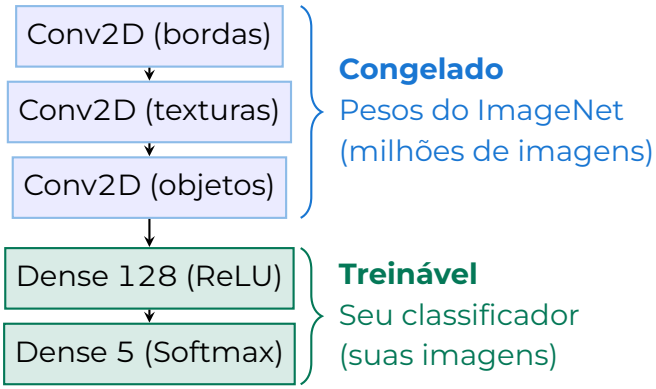


Figura 3: Arquitetura de Transfer Learning: as camadas convolucionais pré-treinadas são congeladas (não sofrem atualização de pesos), e apenas o classificador final é treinado com os dados do problema específico.

O ecossistema Keras e outras bibliotecas já fornecem os modelos *pré-treinados*. São matrizes de pesos (W) que foram treinadas pelo Google, Microsoft ou universidades em supercomputadores com datasets massivos como o **ImageNet** (14 milhões de imagens em 21.000 categorias). Nós fazemos o download dessa "inteligência empacotada", **congelamos a sabedoria** dela (as camadas convolucionais) e utilizamos essa visão aguçada para extrair características das *nossas próprias imagens*, adicionando apenas um pequeno classificador no final.

▶ Prática: Transfer Learning no Keras com ResNet50

```python
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import GlobalAveragePooling2D

# 1. Carregar o cerebro pre-treinado (sem a cabeca)
base_model = ResNet50(
    weights='imagenet',        # Pesos do ImageNet
    include_top=False,         # Remove o classificador original
    input_shape=(224, 224, 3)
)

# 2. Congelar todas as camadas do cerebro
base_model.trainable = False

# 3. Adicionar nosso classificador customizado
model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(5, activation='softmax')  # 5 classes nossas
])

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Com menos de 20 linhas de código, reaproveitamos 25 milhões de parâmetros treinados em semanas de GPU por engenheiros do Google. Nosso treino customizado levará minutos.

▷ **Exemplo: Modelos Pré-Treinados Populares no Keras**

Modelo	Parâmetros	Top-1 Acc	Uso Ideal
VGG16 (Simonyan & Zisserman, 2014)	138M	71.3%	Ensino e prototipação
ResNet50 (He et al., 2015)	25M	76.0%	Equilíbrio precisão/custo
InceptionV3 (Szegedy et al., 2015)	24M	77.9%	Objetos multi-escala
MobileNetV2 (Sandler et al., 2018)	**3.4M**	71.3%	Celulares e embarcados
EfficientNetB0 (Tan & Le, 2019)	5.3M	77.1%	Estado da arte em eficiência

3.1 Quando Usar Transfer Learning?

A decisão de quando e como aplicar Transfer Learning depende da quantidade de dados disponíveis e da semelhança entre o domínio original (ImageNet) e o domínio alvo:

Cenário	Abordagem	Justificativa
Poucas imagens (<1000)	TL + Freeze total	Pouco dado = alto risco de Overfit
Dados moderados (1k–10k)	TL + Fine-tuning parcial	Ajustar últimas camadas convolucionais
Milhões de imagens	Treinar do zero	Dados suficientes para aprender tudo
Domínio muito diferente	TL com cautela	Ex: raio-X médico ≠ fotos do ImageNet

Tabela 2: Guia prático para a escolha da estratégia de Transfer Learning baseada na quantidade de dados e semelhança com o domínio original.

4 Conclusão

A revolução do Deep Learning moderno na visão computacional só foi possível graças a dois pilares: as **CNNs** (que resolveram o problema da explosão de parâmetros e da destruição da estrutura espacial) e o **Transfer Learning** (que democratizou o acesso a modelos treinados com recursos computacionais inimagináveis).

A capacidade de carregar um "cérebro visual" treinado em 14 milhões de imagens e reaproveitá-lo com poucas linhas de código transformou o mercado: o trabalho pesado de dias de processamento matemático se condensou na simples escolha do modelo certo e no treinamento de um classificador leve. Na próxima e última aula deste módulo, fecharemos o arco com as métricas definitivas de **Avaliação** (Precision, Recall, F1-Score) e consolidaremos tudo no **Trabalho Prático 1**.

✔ Takeaways

- Camadas **Dense** fracassam em imagens de alta resolução: a explosão de parâmetros (3×10^9 para uma foto 1000×1000) e a destruição da estrutura espacial pelo *flatten* são bloqueios intransponíveis.
- A **Convolução** resolve ambos os problemas: filtros pequenos (3×3) deslizam pela imagem, compartilhando pesos e preservando vizinhanças.
- CNNs criam uma **hierarquia de abstração**: bordas $\rightarrow$ texturas $\rightarrow$ objetos completos, inspirada no córtex visual dos mamíferos.
- O **Transfer Learning** permite reaproveitar modelos treinados em milhões de imagens (ImageNet), congelando as camadas convolucionais e treinando apenas o classificador final.
- No Keras, modelos pré-treinados como **ResNet50**, **MobileNetV2** e **EfficientNet** estão disponíveis em `tf.keras.applications` com uma única linha de código.

✎ Questões: Autoavaliação

1 **[Direta]** Calcule quantos parâmetros (pesos) uma camada Dense com 256 neurônios teria se recebesse como entrada uma imagem RGB de 100×100 pixels. Compare com o número de parâmetros de uma camada Conv2D com 32 filtros de 3×3.

2 **[Direta]** Explique o papel do `MaxPooling2D` na arquitetura de uma CNN. O que aconteceria se removêssemos todas as camadas de pooling?

3 **[Direta]** No Transfer Learning, por que usamos `include_top=False` ao carregar um modelo pré-treinado? O que contém o "top" que estamos descartando?

4 **[Reflexiva]** O Transfer Learning pressupõe que "bordas e texturas são universais". Essa premissa é válida para qualquer domínio? Discuta um cenário em que transferir pesos do ImageNet poderia **não** funcionar bem.

5 **[Reflexiva]** O compartilhamento de pesos nas CNNs é uma forma de **regularização implícita**. Explique por que, conectando com o conceito de Overfitting que vimos na Aula 05.

Lab. 07: Transfer Learning com MobileNetV2

Capítulo em Revisão

Este conteúdo está sendo revisado. Algumas informações podem estar preliminares, conter erros de formatação ou sofrer alterações na versão final.

⋆ Objetivo do Laboratório

Nos Labs 01–06, todos os seus modelos processaram **tabelas numéricas** (CSVs). Hoje você dá o salto para o mundo das **imagens**. Em vez de treinar uma CNN do zero por horas, aplicará **Transfer Learning**: importará o "cérebro" da MobileNetV2 (pré-treinada pelo Google em 14 milhões de imagens), congelará seus 2,2 milhões de parâmetros e retreinará apenas um classificador de ∼4.000 parâmetros para resolver um problema real de visão computacional — tudo em minutos.

5 Parte 1: O Problema G — Estilo Maratona

Problema G: O Radar Aéreo de TechCity

Contexto: A torre de controle de tráfego de *TechCity* precisa de um sistema automático que analise imagens de câmeras de segurança e classifique se o objeto é um **avião** ou um **automóvel**. A equipe de engenharia tem apenas 10.000 imagens rotuladas e uma GPU modesta — treinar uma CNN do zero levaria dias e provavelmente sofreria Overfitting severo. A solução? Reaproveitar um modelo pré-treinado pelo Google.

Modelo de Entrada (X): Imagens RGB do CIFAR-10 redimensionadas:

- Resolução original: $32 \times 32 \times 3$ (RGB)
- Resolução após resize: $96 \times 96 \times 3$ (compatível com MobileNetV2)
- Normalização: pixels no intervalo $[-1, +1]$
- Classe 0 = Avião | Classe 1 = Automóvel

Modelo de Saída ($\hat{y}$): Probabilidade via Sigmoid $\in (0, 1)$:

- $\hat{y} > 0.5 \quad \rightarrow$ Automóvel
- $\hat{y} \leq 0.5 \quad \rightarrow$ Avião

Critérios de Aprovação:

1. Acurácia no teste $> 80\%$
2. Base MobileNetV2 congelada (`trainable=False`)
3. Parâmetros treináveis $< 1\%$ do total
4. Previsões válidas no intervalo $[0, 1]$

A. Por que Transfer Learning?

A tabela abaixo mostra por que treinar do zero é inviável com poucos dados:

Abordagem	Parâmetros Treináveis	Tempo Estimado	Resultado
Dense (Flatten) direto	≈28.000.000	Horas	Overfitting severo
CNN do zero	≈500.000	30+ min	Resultado modesto
Transfer Learning	**≈4.000**	**3 min**	**>80% acurácia**

O Transfer Learning reduz os parâmetros treináveis em **3 ordens de grandeza** e entrega resultado superior — é a escolha profissional para problemas com poucos dados.

6 Parte 2: Carregando o Dataset CIFAR-10 (8 min)

Usaremos o dataset **CIFAR-10** do Keras, que contém 60.000 imagens coloridas 32×32 em 10 classes. Filtraremos apenas Aviões (classe 0) e Automóveis (classe 1) para classificação binária.

Crie o arquivo `src/transfer_learning.py`:

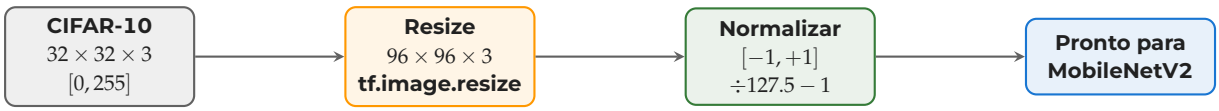
```python
import numpy as np
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import (Dense, Dropout,
    GlobalAveragePooling2D)
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import EarlyStopping
import tensorflow as tf

# Carregar CIFAR-10
(X_train_full, y_train_full), (X_test_full, y_test_full) = \
    cifar10.load_data()

# Filtrar: classe 0=avião, classe 1=automóvel
classes = [0, 1]
mask_train = np.isin(y_train_full.flatten(), classes)
mask_test = np.isin(y_test_full.flatten(), classes)

X_train = X_train_full[mask_train]
y_train = (y_train_full[mask_train].flatten() == 1).astype(int)
X_test = X_test_full[mask_test]
y_test = (y_test_full[mask_test].flatten() == 1).astype(int)

print(f"Treino: {X_train.shape[0]} imagens")
print(f"Teste:  {X_test.shape[0]} imagens")
```

• Teoria: Por que filtrar apenas 2 classes?

Para que o treino caiba nos 50 minutos do lab, reduzimos para classificação binária: **Avião (0) vs Automóvel (1)**. O conceito é idêntico para 10 classes — muda apenas a última camada (Softmax com 10 saídas em vez de Sigmoid com 1). Cada classe tem ≈5.000 imagens de treino e 1.000 de teste.

7 Parte 3: Pré-processamento para MobileNetV2 (8 min)

A MobileNetV2 espera imagens maiores que 32×32 e pixels normalizados para $[-1, +1]$. Observe a transformação:



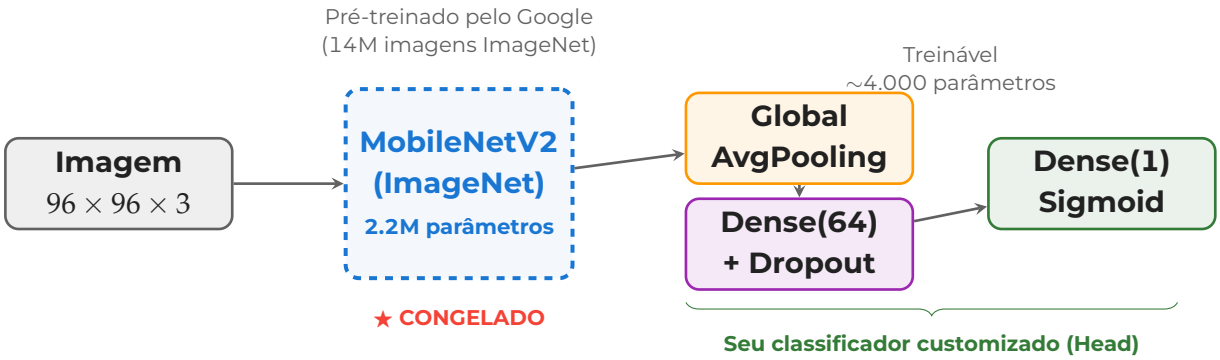

```python
def preprocessar(X):
    """Redimensiona para 96x96 e normaliza para [-1, 1]."""
    # Redimensionar de 32x32 para 96x96
    X_resized = tf.image.resize(X, (96, 96)).numpy()
    # Normalizar para [-1, 1] (padrão do MobileNetV2)
    X_norm = X_resized / 127.5 - 1.0
    return X_norm

X_train_proc = preprocessar(X_train)
X_test_proc = preprocessar(X_test)

print(f"Shape pós-processamento: {X_train_proc.shape}")
print(f"Intervalo: [{X_train_proc.min():.1f}, "
      f"{X_train_proc.max():.1f}]")
```

△ Importante: Por que 96×96 e não 224×224?

Usamos 96×96 em vez de 224×224 para que o treino caiba em GPUs modestas e termine dentro dos 50 minutos. A MobileNetV2 aceita múltiplas resoluções (mínimo: 32×32). Em produção, use resoluções maiores para maior precisão — o trade-off é tempo de treino vs qualidade.

8 Parte 4: Montando o Modelo com Transfer Learning (10 min)

Agora o passo central: carregar o "cérebro" pré-treinado, congelá-lo e adicionar nosso classificador. Observe o diagrama da arquitetura:




```python
def criar_modelo_transfer():
    """Modelo com Transfer Learning usando MobileNetV2."""
    # 1. Carregar a base pré-treinada (SEM o topo)
    base_model = MobileNetV2(
        weights='imagenet',
        include_top=False,          # Remove classificador original
        input_shape=(96, 96, 3)
    )

    # 2. CONGELAR o cérebro (não treinar os pesos do Google)
    base_model.trainable = False

    # 3. Montar modelo completo com nosso classificador
    model = Sequential([
        base_model,
        GlobalAveragePooling2D(),  # Reduz mapa 3D -> vetor 1D
        Dense(64, activation='relu'),
        Dropout(0.3),
        Dense(1, activation='sigmoid')  # Binário: avião vs auto
    ])

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

modelo, base = criar_modelo_transfer()
```

• Teoria: O que acontece em cada linha

Linha	Efeito
`include_top=False`	Descarta o classificador original (1.000 classes ImageNet)
`trainable = False`	Congela todos os 2,2M de pesos da MobileNetV2
`GlobalAveragePooling2D`	Comprime o mapa de features 3D em um vetor 1D
`Dense(64) + Dense(1)`	Nosso classificador customizado (~4.000 parâmetros)

Resultado: treinamos $< 1\%$ dos parâmetros, mas herdamos 100% do "conhecimento visual" do Google.

9 Parte 5: Treinamento com Early Stopping (10 min)

```python
# Callback: parar quando val_loss estagnar
vigilante = EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True
)

# Treinar (apenas o topo!)
historico = modelo.fit(
    X_train_proc, y_train,
    epochs=30,
    batch_size=32,
    validation_split=0.2,
    callbacks=[vigilante],
    verbose=1
)

epocas_executadas = len(historico.epoch)
print(f"\nTreino concluído em {epocas_executadas} épocas")
```

B. Modelos do Zoo Keras — Referência

A MobileNetV2 é apenas um dos modelos disponíveis. A tabela abaixo mostra o "zoológico" do Keras para referência futura:

Modelo	Parâmetros	Top-1 Acc	Uso Ideal
VGG16	138M	71.3%	Ensino e prototipação
ResNet50	25M	76.0%	Equilíbrio precisão/custo
InceptionV3	24M	77.9%	Objetos multi-escala
MobileNetV2	**3.4M**	71.3%	**Celulares e embarcados** ← Lab
EfficientNetB0	5.3M	77.1%	Estado da arte em eficiência

Todos disponíveis em `tf.keras.applications` com uma única linha de código. A MobileNetV2 é a mais leve — ideal para lab com GPU modesta.

10 Parte 6: Avaliação e Gráfico (5 min)

Plote as curvas de aprendizado e avalie no conjunto de teste:

```python
def plotar_curvas(historico):
    """Gera gráfico de diagnóstico do Transfer Learning."""
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
    epocas = range(1, len(historico.history['loss']) + 1)

    ax1.plot(epocas, historico.history['loss'], 'b-',
             label='Loss Treino')
    ax1.plot(epocas, historico.history['val_loss'], 'r-',
             label='Val Loss')
    ax1.set_title('Perda')
    ax1.legend()
    ax1.grid(True, alpha=0.3)

    ax2.plot(epocas, historico.history['accuracy'], 'b-',
             label='Acc Treino')
    ax2.plot(epocas, historico.history['val_accuracy'], 'r-',
             label='Val Acc')
    ax2.set_title('Acurácia')
    ax2.legend()
    ax2.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.savefig('transfer_learning.png', dpi=150)
    plt.close()
    print("Gráfico salvo: transfer_learning.png")

if __name__ == "__main__":
    plotar_curvas(historico)

    # Avaliação no teste
    _, acc_test = modelo.evaluate(
        X_test_proc, y_test, verbose=0)

    total = modelo.count_params()
    congelados = base.count_params()
```

```
    treinaveis = total - congelados

    print(f"\n{'='*50}")
    print(f"  TRANSFER LEARNING - MobileNetV2")
    print(f"  Acurácia no Teste:       {acc_test:.2%}")
    print(f"  Parâmetros treináveis:   {treinaveis:,}")
    print(f"  Parâmetros congelados:   {congelados:,}")
    print(f"  Razão treinável/total:   "
          f"{treinaveis/total:.2%}")
    print(f"  Épocas executadas:       {epocas_executadas}")
    print(f"{'='*50}")
```

• Teoria: O Paradoxo da Eficiência

Com Transfer Learning, você treina ∼4.000 parâmetros e atinge >80% de acurácia. Sem TL, precisaria treinar ∼500.000 parâmetros (CNN do zero) ou ∼28.000.000 (Dense puro) e provavelmente obteria resultado pior. O segredo é que os 2,2M de parâmetros congelados já "sabem ver" — eles detectam bordas, texturas e formas que são universais para qualquer problema de imagem.

11 Parte 7: Garantia de Qualidade — Testes CI/CD (5 min)

Os testes já estão prontos no arquivo `tests/test_transfer.py` do seu repositório:

```
import numpy as np
from src.transfer_learning import (
    modelo, base, historico,
    X_test_proc, y_test
)

def test_base_congelada():
    """A base MobileNetV2 DEVE estar congelada."""
    assert base.trainable == False, \
        "Base deveria estar congelada (trainable=False)"

def test_acuracia_minima():
    """Transfer Learning deve atingir >80% no teste."""
    _, acc = modelo.evaluate(
        X_test_proc, y_test, verbose=0)
    assert acc > 0.80, \
        f"Acurácia {acc:.2%} baixa para Transfer Learning"

def test_poucos_parametros_treinaveis():
    """Apenas o topo deve ter parâmetros treináveis."""
    total = modelo.count_params()
```

```
    congelados = base.count_params()
    treinaveis = total - congelados
    assert treinaveis < total * 0.01, \
        f"Muitos parâmetros treináveis: {treinaveis:,}"

def test_predicoes_validas():
    """O modelo deve gerar probabilidades entre 0 e 1."""
    preds = modelo.predict(X_test_proc[:10], verbose=0)
    assert np.all(preds >= 0) and np.all(preds <= 1)
    assert preds.shape == (10, 1)
```

1. Rode os testes no terminal: `pytest tests/test_transfer.py -v`.
2. Se a tela exibir **4 passed** em verde, sua implementação está correta.

12 Parte 8: Commit e Push (5 min)

Com todos os testes passando localmente, efetue o envio para o GitLab:

```
git add src/transfer_learning.py transfer_learning.png
git commit -m "Lab 07: Transfer Learning com MobileNetV2 (CIFAR-10)"
git push origin main
```

O GitLab CI interceptará seu push e executará o `pytest` automaticamente. O **Selo Verde** confirma sua pontuação.

△ Importante: Se o Pipeline Falhar

Os erros mais comuns são: (1) CIFAR-10 não baixar automaticamente no container CI — o Keras faz download na primeira execução, e o CI precisa de acesso à internet; (2) a normalização não aplicar $\div 127.5 - 1$ (MobileNetV2 espera $[-1, +1]$, não $[0, 1]$); (3) esquecer `base_model.trainable = False` e treinar 2,2M de parâmetros por acidente.

Conclusão: O que Você Construiu Hoje

✔ Takeaways

- Você aplicou **Transfer Learning real**: carregou a MobileNetV2 pré-treinada em 14 milhões de imagens do ImageNet e congelou seus 2,2M de parâmetros.
- Treinou apenas ∼4.000 parâmetros do classificador customizado (Head) e atingiu >**80% de acurácia** em um classificador de imagens funcional.
- Entendeu o pipeline completo: CIFAR-10 → Resize (96×96) → Normalizar ($[-1,+1]$) → MobileNetV2 congelada → GAP → Dense → Sigmoid.
- Usou **Early Stopping** para parar no ponto ótimo e plotou as curvas de aprendizado para confirmar que o modelo generaliza.
- Os 4 testes CI/CD verificam: base congelada, acurácia $> 80\%$, $< 1\%$ de parâmetros treináveis e previsões válidas.
- Conheceu o "Zoo Keras" de modelos pré-treinados: VGG16, ResNet50, EfficientNet — todos acessíveis com uma linha de código.

◇ **Informação**

Gancho para a Próxima Aula: Seu classificador atinge 85% de acurácia. Mas **o que significa** esse número? Se o modelo erra 15% dos diagnósticos médicos, quantos pacientes saudáveis são operados desnecessariamente? E quantos pacientes doentes são mandados para casa? Na próxima aula (fechamento do Módulo 1), você empacotará o modelo numa **API REST com FastAPI**, containerizará tudo com **Docker** e enfrentará as métricas que **realmente importam**: **Precision**, **Recall**, **F1-Score** e a **Matriz de Confusão** — as ferramentas que distinguem um erro aceitável de um erro catastrófico.