Lab. 07: Visão Computacional com Transfer Learning

Aléssio Miranda Júnior

Agosto - 2026/2

Lab. 07: Transfer Learning com MobileNetV2

⋆ 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.

1 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.

2 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`:

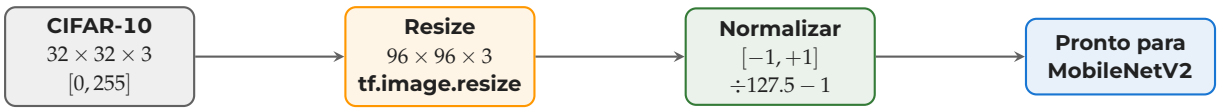
```
import numpy as np
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

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.

3 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:



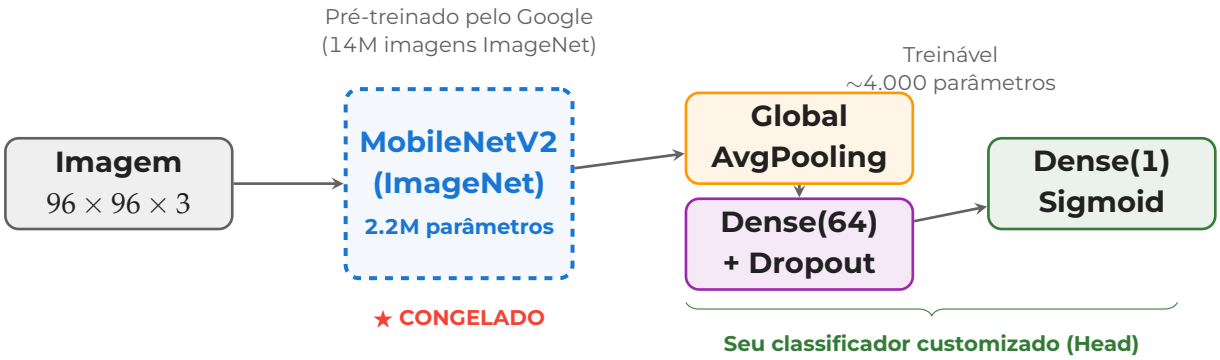

```
def preprocessar(X):
    """Redimensiona para 96x96 e normaliza para [-1, 1]."""
    # Redimensionar de 32x32 para 96x96
    X_resized = tf.image.resize(X, (96, 96)).numpy()
    # Normalizar para [-1, 1] (padrão do MobileNetV2)
    X_norm = X_resized / 127.5 - 1.0
    return X_norm

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.

4 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.

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

```
# 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.

6 Parte 6: Avaliação e Gráfico (5 min)

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

```
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.

7 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.

8 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 $\sim$4.000 parâmetros do classificador customizado (Head) e atingiu $>$**80% de acurácia** em um classificador de imagens funcional.
- Entendeu o pipeline completo: CIFAR-10 $\rightarrow$ Resize (96×96) $\rightarrow$ Normalizar ($[-1,+1]$) $\rightarrow$ MobileNetV2 congelada $\rightarrow$ GAP $\rightarrow$ Dense $\rightarrow$ 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.