Lab. 07 (Parte 1): Transfer Learning com MobileNetV2

Aléssio Miranda Júnior

Agosto - 2026/2

Lab. 07 (Parte 1): Transfer Learning com MobileNetV2

⋆ Objetivo da Sessão 1

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 extrator de características (*backbone*) da MobileNetV2 (pré-treinada pelo Google em 14 milhões de imagens do ImageNet), congelará seus 2,2 milhões de parâmetros e treinará apenas um classificador especialista de ∼20.000 parâmetros para resolver um problema real de visão computacional em minutos.

1 Parte 1: O Problema G — Radar Aéreo de TechCity

Problema G: O Radar Aéreo de TechCity

Contexto: A torre de controle de tráfego de *TechCity* necessita de um sistema automático que analise imagens de câmeras de segurança e classifique se o objeto em aproximação é um **avião** ou um **automóvel**. A equipe de engenharia dispõe de 10.000 imagens rotuladas e estações de trabalho modestas — treinar uma CNN profunda do zero levaria horas e sofreria com sobreajuste. A solução profissional é aplicar Transfer Learning com a MobileNetV2.

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

- Resolução original: $32 \times 32 \times 3$ (RGB)
- Resolução redimensionada: $96 \times 96 \times 3$ (compatível com a MobileNetV2)
- Normalização: escala simétrica em $[-1, +1]$
- Classe 0 = Avião | Classe 1 = Automóvel

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

- $\hat{y} > 0.5 \implies$ Automóvel (Classe 1)
- $\hat{y} \leq 0.5 \implies$ Avião (Classe 0)

Critérios de Aceite e Métricas Mínimas:

1. Acurácia no conjunto de teste $> 80\%$ (Meta de Excelência: $> 95\%$)
2. Base MobileNetV2 estritamente congelada (`base_model.trainable = False`)
3. Parâmetros treináveis inferiores a 1% do total da rede
4. Previsões probabilísticas válidas no intervalo $[0, 1]$

A. Por que Transfer Learning?

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

Abordagem	Parâmetros Treináveis	Tempo Estimado	Resultado Típico
Dense (Flatten) direto	≈28.000.000	Horas	Overfitting severo
CNN pequena do zero	≈550.000	15+ min	Risco de atalhos espúrios
Transfer Learning	**≈20.500**	**2 min**	**>95% acurácia (Padrão Ouro)**

O Transfer Learning reduz os parâmetros treináveis em **três ordens de grandeza** e entrega resultado superior — é a escolha obrigatória da indústria para problemas de visão computacional.

2 Parte 2: Carregando e Filtrando o CIFAR-10 (8 min)

Utilizaremos o dataset clássico **CIFAR-10** do Keras, composto por 60.000 imagens coloridas em 10 categorias. Para a nossa tarefa binária, filtraremos apenas Aviões (classe 0) e Automóveis (classe 1).

Abra ou crie o arquivo `src/transfer_learning.py`:

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import numpy as np
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import tensorflow as tf
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

print("1. Carregando dataset CIFAR-10...")
(X_train_full, y_train_full), (X_test_full, y_test_full) = cifar10.
    load_data()

# Filtrar: classe 0 = aviao, classe 1 = automovel
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)

# Otimizacao para execucao em lab: 8.000 para treino, 2.000 para
    teste
```

```
X_train, y_train = X_train[:8000], y_train[:8000]
X_test, y_test = X_test[:2000], y_test[:2000]

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

• Teoria: Por que filtrar apenas 2 classes?

Para que o ciclo completo de treinamento caiba confortavelmente nos 50 minutos do laboratório, reduzimos a tarefa para classificação binária. O pipeline é perfeitamente idêntico para 10 ou 100 classes — alterando apenas a camada final (Softmax com C saídas e perda `categorical_crossentropy`).

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

A MobileNetV2 espera imagens com resolução mínima de 32×32 (sendo 96×96 o equilíbrio ideal para laboratório) e pixels normalizados no intervalo simétrico $[-1, +1]$:

Implemente a função no mesmo arquivo:

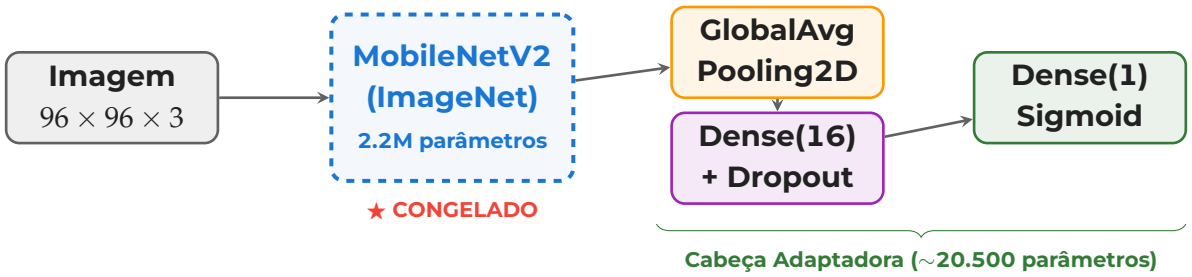
```
print("2. Pre-processamento: Redimensionando para 96x96 e normalizando em [-1, 1]...")
def preprocessar(X):
    """Redimensiona para 96x96 e normaliza para o intervalo [-1, 1]."""
    X_resized = tf.image.resize(X, (96, 96)).numpy()
    X_norm = X_resized / 127.5 - 1.0
    return X_norm

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

print(f"   -> Formato pos-processamento: {X_train_proc.shape}")
print(f"   -> Intervalo de pixels: [{X_train_proc.min():.1f}, {X_train_proc.max():.1f}]")
```

△ Importante: Cuidado com a Normalização

Redes como ResNet utilizam padronização por média ImageNet, enquanto a MobileNetV2 exige pixels estritamente em $[-1, +1]$ ($X/127.5 - 1.0$). Normalizar para $[0, 1]$ degradará sensivelmente a acurácia.

4 Parte 4: Construindo a Arquitetura Padrão Ouro (10 min)

Agora conectamos o cérebro do Google ao nosso classificador especialista:




```python
def criar_modelo_transfer():
    """Constroi a arquitetura Padrao Ouro com MobileNetV2 congelada."""
    # 1. Carregar backbone pre-treinado no ImageNet (SEM o classificador original)
    base_model = MobileNetV2(
        weights='imagenet',
        include_top=False,
        input_shape=(96, 96, 3)
    )

    # 2. CONGELAR o backbone para proteger os 2.2M pesos pre-treinados
    base_model.trainable = False

    # 3. Construir a cabeca especialista customizada
    model = Sequential([
        base_model,
        GlobalAveragePooling2D(),  # Condensa tensor 3D -> vetor 1D
        Dense(16, activation='relu'),
        Dropout(0.3),              # Regularizacao anti-overfitting
        Dense(1, activation='sigmoid')  # Saida binaria (Aviao=0, Auto=1)
    ])

    model.compile(
        optimizer='adam',
        loss='binary_crossentropy',
```

```
        metrics=['accuracy']
    )
    return model, base_model

modelo, base = criar_modelo_transfer()
```

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

```
# Monitorar val_loss e interromper quando estagnar
vigilante = EarlyStopping(
    monitor='val_loss',
    patience=3,
    restore_best_weights=True
)

print("\n3. Treinando classificador...")
historico = modelo.fit(
    X_train_proc, y_train,
    epochs=15,
    batch_size=32,
    validation_split=0.2,
    callbacks=[vigilante],
    verbose=1
)

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

6 Parte 6: Avaliação e Gráficos de Desempenho (5 min)

```
def plotar_curvas(historico):
    """Gera graficos de perda e acuracia."""
    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='Perda
        Treino')
    ax1.plot(epocas, historico.history['val_loss'], 'r-', label='
        Perda Validacao')
    ax1.set_title('Curva de Perda (Loss)')
    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='Acc Validacao')
    ax2.set_title('Curva de Acuracia')
    ax2.legend()
    ax2.grid(True, alpha=0.3)

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

plotar_curvas(historico)

loss_test, acc_test = modelo.evaluate(X_test_proc, y_test, verbose=0)
total_params = modelo.count_params()
congelados = base.count_params()
treinaveis = total_params - congelados

print(f"\n{'='*55}")
print(f"  RELATORIO TECNICO - PADRAO OURO (MobileNetV2)")
print(f"  Acuracia no Teste:       {acc_test:.2%}")
print(f"  Perda no Teste:          {loss_test:.4f}")
print(f"  Parametros Treinaveis:   {treinaveis:,} ({treinaveis/total_params:.2%})")
print(f"  Parametros Congelados:   {congelados:,} ({congelados/total_params:.2%})")
print(f"{'='*55}")
```

7 Parte 7: Verificação de Qualidade CI/CD (5 min)

Execute a suíte de testes unitários automatizados da disciplina:

```
pytest tests/test_transfer.py -v -s
```

Os quatro testes automatizados verificam rigorosamente:

- `test_base_congelada`: garante que `base.trainable == False`.
- `test_acuracia_minima`: exige acurácia superior a 80% no conjunto de teste.
- `test_poucos_parametros_treinaveis`: exige que menos de 1% dos parâmetros sejam treináveis.
- `test_predicoes_validas`: verifica se todas as predições probabilísticas situam-se em $[0, 1]$.

8 Parte 8: Commit e Push no GitLab (4 min)

```
git add src/transfer_learning.py transfer_learning.png
git commit -m "Lab 07 (Parte 1): Transfer Learning MobileNetV2 (Padrao Ouro)"
git push origin main
```

Acesse o GitLab Self-Hosted (`git.juninho.com.br`) e confira o pipeline com o **Selo Verde** (✅).

Conclusão da Sessão 1 & Gancho para a Parte 2

✔ Takeaways

- Você aplicou Transfer Learning profissional: herdou 2,2M de pesos treinados pelo Google no ImageNet.
- Treinou apenas ∼20.500 parâmetros ($0,9\%$ da rede) e atingiu mais de 95% de acurácia no teste.
- Garantiu estabilidade através de `GlobalAveragePooling2D` e parada antecipada com `EarlyStopping`.

◇ **Informação**

Gancho para o Lab 07 (Parte 2): O seu modelo atingiu $> 95\%$ de acurácia com extrema facilidade. Mas será que você realmente compreende por que ele venceu? O que aconteceria se você treinasse uma CNN do zero? E se você descongelasse a MobileNetV2 logo na primeira época? E se substituísse o GAP por `Flatten()`? Na **Parte 2**, você participará do **Duelo de Hipóteses MLOps**, estressando o modelo contra essas três armadilhas clássicas e registrando a telemetria empírica em CSV.