Inteligência Artificial 2

Redes Convolucionais e Transfer Learning

Prof. Aléssio Miranda Júnior
alessio@cefetmg.br

CEFET-MG - Campus Timoteo
Dep. Engenharia de Computação

18 de agosto de 2026

Onde Paramos?

Na aula anterior, montamos o **arsenal anti-Overfitting**:

- **Regularização L2**: penaliza pesos grandes na loss.
- **Dropout**: desliga neurônios aleatoriamente durante o treino.
- **Early Stopping**: para o treinamento automaticamente no ponto ótimo.
- **Adam**: otimizador adaptativo padrão da indústria.

A Pergunta de Hoje

Todos os nossos modelos processam **tabelas CSV**.
E se a entrada for uma **imagem**?

Objetivos da Sessão

- **Limite das MLPs:** Por que camadas Dense fracassam em imagens.
- **Tensores de Imagem:** Representação numérica $H \times W \times C$.
- **Convolução:** Filtros que deslizam pela imagem extraindo padrões locais.
- **Pooling:** Redução dimensional progressiva.
- **Hierarquia de Abstração:** Bordas $\rightarrow$ texturas $\rightarrow$ objetos.
- **Transfer Learning:** Reaproveitar cérebros treinados pelo Google/Meta.
- **Freezing + Fine-tuning:** Congelar pesos e treinar só o classificador.

O Mundo Além do CSV

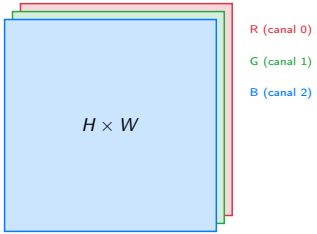
- Até agora: dados tabulares em CSV (n linhas $\times$ p colunas).
- Hoje: **imagens** — tensores de milhões de pixels.
- Próximo módulo: **texto** — sequências de palavras.

Imagens: Tensores 3D

- Uma foto colorida = tensor $H \times W \times 3$ (RGB).
- Foto 1000×1000:

$$1000 \times 1000 \times 3 = \mathbf{3.000.000}$$

- Cada pixel tem 3 valores: R, G, B.
- Em escala de cinza: $H \times W \times 1$.



Tensor: $(H, W, 3)$

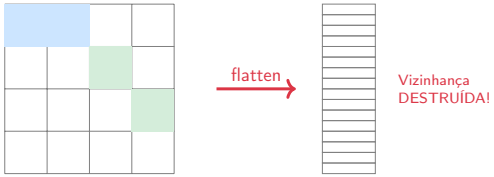
A Explosão Paramétrica

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**

Impossível!

Uma única camada Dense com 1000 neurônios em uma foto de celular geraria **36 bilhões de parâmetros**. Nenhuma GPU do mundo consegue treinar isso.

O Problema do Flatten



- O *flatten* estira a imagem 2D em um fio 1D.
- O pixel da bochecha e o pixel do olho **perdem a vizinhança geométrica**.
- A rede fica **cega a formas**: não sabe que "esses pixels formam um rosto".

A Solução: Não Conectar Tudo a Todos

Inspiração Biológica: O Córtex Visual

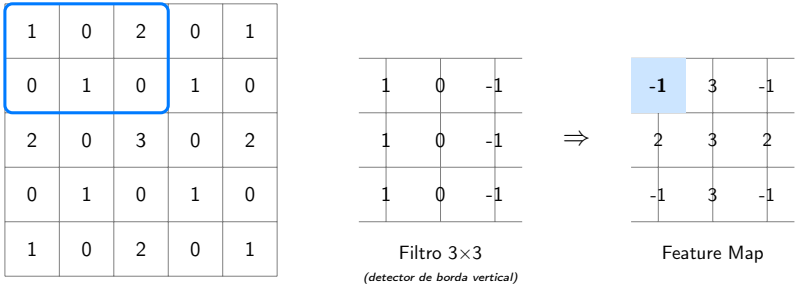
Hubel & Wiesel (Nobel 1962): o cérebro **não** processa todos os pixels de uma vez. Células especializadas varrem **pequenas regiões** da imagem, detectando bordas, curvas e padrões locais.

Dense (MLP)
Cada pixel → cada neurônio
Bilhões de conexões

→

Conv2D (CNN)
Filtro 3×3 → vizinhança local
Centenas de conexões

A Operação de Convolução



O filtro desliza por toda a imagem, calculando o **produto escalar** em cada posição.

Stride e Padding: Os Ajustes Finos

- **Stride (Passo):** O número de pixels que o filtro pula a cada deslizada.
  - Stride = 1: desliza pixel a pixel.
  - Stride = 2: pula de dois em dois, o que **reduz a imagem pela metade** rapidamente.
- **Padding (Preenchimento):** Quando o filtro chega na borda da imagem, ele costuma "encolher"a saída (ex: $5 \times 5 \rightarrow 3 \times 3$).
  - `padding='same'`: o Keras adiciona uma moldura de pixels de valor zero (zeros) ao redor da imagem original. Assim, a saída tem o mesmo tamanho da entrada.

Como Funciona o Cálculo?

1	0	2
0	1	0
2	0	3

Pedaço da Imagem

$\odot$

1	0	-1
1	0	-1
1	0	-1

Filtro

$$
\begin{aligned}
&(1 \cdot 1) + (0 \cdot 0) + (2 \cdot -1) \\
&+(0 \cdot 1) + (1 \cdot 0) + (0 \cdot -1) \\
&+(2 \cdot 1) + (0 \cdot 0) + (3 \cdot -1) \\
&= 1 + 0 - 2 + 0 + 0 + 0 + 2 + 0 - 3 \\
&= \textbf{-2}
\end{aligned}
$$

O resultado -2 vai para a posição correspondente no **Feature Map**.

Compartilhamento de Pesos: O Segredo

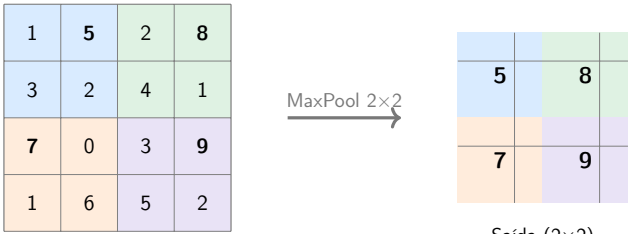
A Grande Vantagem da CNN

O **mesmo filtro** 3×3 (apenas **9 parâmetros**) é reutilizado em **toda a extensão** da imagem. Ele não muda conforme a posição.

Camada	**Parâmetros**	**Obs.**
Dense ($1000 \times 1000 \times 3 \rightarrow 128$)	384.000.128	Impossível
Conv2D (32 filtros 3×3, 3 canais)	**896**	Viável!

$32 \times (3 \times 3 \times 3 + 1) = 896$ parâmetros. Contra **384 milhões** da Dense.

MaxPooling: Redução Dimensional



Entrada (4×4)

Saída (2×2)

- Reduz dimensões pela **metade**, mantendo o valor **máximo** de cada região.
- Diminui custo computacional e aumenta **invariância posicional**.

Data Augmentation: Multiplicando Dados

Imagens são difíceis de conseguir e fáceis de sofrer overfitting. A solução?

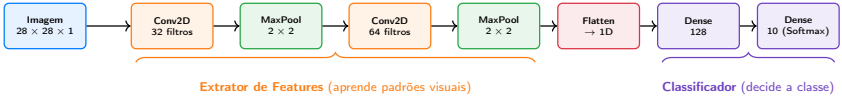
- Pegue uma foto de um gato no treino.
- **Vire horizontalmente** (flip): a rede acha que é outro gato.
- **Rotacione 10 graus**: outro gato.
- **Dê um zoom leve**: outro gato!

No Keras, fazemos isso em tempo real (on-the-fly) com a camada `RandomFlip` ou o `ImageDataGenerator`. Assim, a rede nunca vê a mesma imagem duas vezes do mesmo jeito.

A Hierarquia de Abstração

- Cada camada usa os **Feature Maps da anterior** como entrada.
- É exatamente como o **córtex visual**: células simples $\rightarrow$ células complexas $\rightarrow$ reconhecimento.

Arquitetura Completa de uma CNN



A CNN tem duas partes: o **extrator convolucional** (que "vê") e o **classificador denso** (que "decide").

CNN Clássica no Keras

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

model = Sequential()
model.add(Conv2D(32, (3,3), activation='relu',
                 input_shape=(28, 28, 1)))
model.add(MaxPooling2D((2,2)))   # 28x28 -> 14x14
model.add(Conv2D(64, (3,3), activation='relu'))
model.add(MaxPooling2D((2,2)))   # -> 5x5
model.add(Flatten())             # 2D -> 1D
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))  # 10 classes
```

Conv2D + MaxPooling → extrai features. **Flatten + Dense** → classifica.

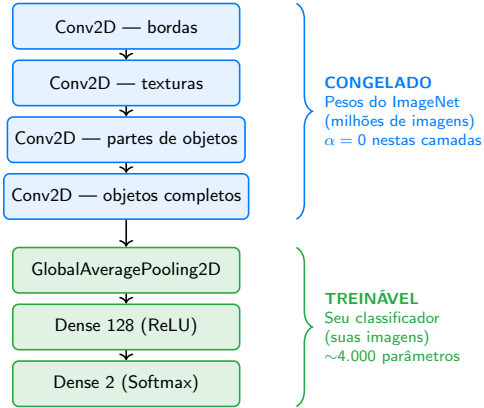
O Custo de Treinar do Zero

- Treinar uma CNN de 50 camadas do zero = semanas em supercomputadores.
- Nenhuma startup paga US$ 100k para classificar "cachorro vs gato".
- Solução: **roubar a inteligência** de quem já treinou.

Transfer Learning: A Filosofia

1. O Google/Meta treinou redes gigantes no **ImageNet** (14M imagens, 21k categorias).
2. A rede aprendeu: bordas, texturas, olhos, rodas, rostos...
3. Os pesos finais (matrizes W) são **publicados gratuitamente**.
4. Nós **baixamos** essa inteligência empacotada.

Congelamento (Freezing)



Transfer Learning em 10 Linhas

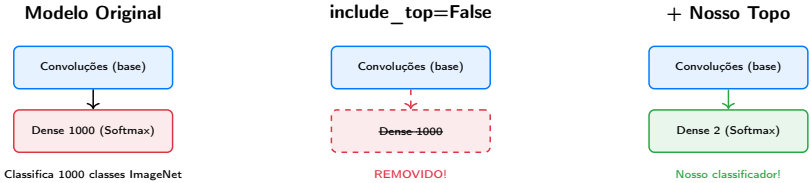
```
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import GlobalAveragePooling2D

# 1. Baixar o cérebro pré-treinado (SEM o topo)
base = MobileNetV2(weights='imagenet',
                   include_top=False,
                   input_shape=(96, 96, 3))

# 2. CONGELAR todas as camadas
base.trainable = False

# 3. Adicionar nosso classificador
model = Sequential([
    base,
    GlobalAveragePooling2D(),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(2, activation='softmax')  # Nosso problema
```

O que é `include_top=False`?



Removemos o "topo" (classificador de 1000 classes do Google) e plugamos o **nosso** classificador (2, 5 ou quantas classes precisarmos).

Modelos Pré-Treinados no Keras

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
EfficientNetB0	5.3M	77.1%	Estado da arte em eficiência

Todos disponíveis em `tf.keras.applications` com uma linha de código.

Quando Usar Transfer Learning?

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

Regra Prática

Na dúvida, comece **sempre** com Transfer Learning. Só treine do zero se tiver recursos e dados de sobra.

Recapitulação — O que Levar Desta Aula

1. Camadas Dense fracassam em imagens: **explosão de parâmetros** + perda da estrutura espacial via flatten.
2. **CNNs** usam filtros deslizantes (Conv2D) que **compartilham pesos** → eficiência radical.
3. **MaxPooling** reduz dimensões pela metade, mantendo o mais relevante.
4. A **hierarquia de abstração**: bordas → texturas → objetos completos.
5. **Transfer Learning**: baixar cérebro treinado (`weights='imagenet'`), congelar (`trainable=False`), treinar só o topo.
6. `include_top=False` remove o classificador original; plugamos o nosso.

Próxima Aula (Última do Módulo)

Métricas de Avaliação: Precision, Recall, F1-Score, Matriz de Confusão e **TP1**.

Missão: Classificar imagens reais usando Transfer Learning com MobileNetV2.

- Dataset: **CIFAR-10** (auto-download, sem arquivos externos).
- Modelo: MobileNetV2 congelada + Dense(64) + Dropout(0.3) + Dense(1, sigmoid).
- Alvo: **>80% de acurácia** com apenas ∼4.000 parâmetros treináveis.

Lab Passo 1: O CIFAR-10

Pela primeira vez, vocês não gerarão dados falsos. Usarão o lendário **CIFAR-10**:

- Um dataset de 60.000 imagens reais coloridas (RGB) divididas em 10 classes (aviões, gatos, cães, navios...).
- Resolução brutalmente baixa: 32×32 pixels. Até os humanos têm dificuldade de reconhecer o que tem nelas!
- Para este lab, selecionaremos apenas 2 classes (Aviões e Automóveis) para simplificar a vida da CPU/GPU.

Lab Passo 2: O Desafio do Resize

O nosso "cérebro pré-treinado"(MobileNetV2) não aceita imagens tão pequenas. Ele exige pelo menos 96×96.

- Vocês usarão a função `tf.image.resize()` para esticar as imagens de 32 para 96.
- O TensorFlow fará uma *interpolação bilinear* automática (borrando a imagem para ela crescer sem estourar os pixels).
- Logo após o resize, **Normalização**: dividiremos toda a matriz por 255.0. No mundo da imagem, os pixels vão de 0 a 255. Com a divisão, ficam todos harmoniosamente entre 0 e 1.

Lab Passo 3: O Cérebro do Google

Hora de chamar o "universitário":

```
from tensorflow.keras.applications import MobileNetV2

base = MobileNetV2(
    input_shape=(96, 96, 3),
    include_top=False,      # Arranca o topo
    weights='imagenet'      # Baixa a inteligencia
)

# OBRIGATÓRIO: Congelar a mente do mestre
base.trainable = False
```

Lab Passo 4: O Seu Classificador

Plugando a sua rede na base congelada:

```
model = Sequential([
    base,                              # Extrator
    GlobalAveragePooling2D(),          # Reduz tensor para vetor 1D
    Dense(64, activation='relu'),      # Sua inteligência
    Dropout(0.3),                      # Vacina anti-overfit
    Dense(1, activation='sigmoid') # Avião (0) ou Carro (1)
])
```

A base tem milhões de pesos. Como estão congelados, apenas os 82 mil pesos do topo serão treinados.

Lab Passo 5: Treino com EarlyStopping e Avaliação

- Compilem usando a métrica e a função de custo já conhecidas.
- Treinem com `epochs=200` e `patience=10` no EarlyStopping.
- Vocês perceberão que cada época de imagem é **mais lenta** do que nos labs anteriores. Processar imagens 96×96 requer muito processamento matemático.
- Se fizeram tudo certo, a validação saltará rapidamente para mais de 80% de acurácia em pouquíssimas épocas. A magia do Transfer Learning!

Lab Passo 6: CI/CD Pipeline (Selando a Aula)

Eternizando o trabalho na nuvem:

- `git add .`
- `git commit -m "Lab 07 - Transfer Learning com MobileNet"`
- `git push origin master`

No Servidor do GitLab

O servidor validará as imagens, usará o modelo treinado para prever Aviões e Carros em um dataset novo, e checará se superou o limite de aprovação. **Selo Verde** aguarda!

Dúvidas?

Email: alessio@cefetmg.br
Aula: Segunda e Quarta-Feira, 7h as 8:40h (Sala 39)
WhatsApp: Dúvidas rápidas

Lembrete

Confira sua Issue no GitLab!

Próxima aula: Métricas de Avaliação & TP1

100% da Aula 7 Concluída!