Notas de Aula - Semana 6

Introdução às Redes Neurais Convolucionais

Curso de Redes Neurais

Eduardo Adame

24 de setembro de 2025

Sumário

1 Introdução **3**

2 Motivação: As Limitações das MLPs para Imagens **3**
2.1 O Problema da Explosão de Parâmetros 3
2.2 Perda de Estrutura Espacial 4
2.3 Ausência de Invariância à Translação 4

3 A Natureza Hierárquica da Visão **4**
3.1 Como o Cérebro Processa Informação Visual 4
3.2 Construção de Representações Complexas 5

4 Convolução: O Coração das CNNs **5**
4.1 Operação de Convolução 5
4.2 Filtros como Detectores de Características 6

5 Arquitetura de uma CNN **6**
5.1 Camadas Convolucionais 6
5.1.1 Hiperparâmetros das Camadas Convolucionais 7
5.2 Camadas de Pooling 7
5.2.1 Motivação para Pooling 8
5.3 Arquitetura Típica 8

6 Comparação Detalhada: CNN vs MLP **8**
6.1 Análise de Parâmetros 8
6.2 Propriedades Comparativas 9

7 Implementação Prática **9**
7.1 Exemplo em TensorFlow/Keras 9
7.2 Análise da Arquitetura 10

8 Técnicas Avançadas e Boas Práticas **11**
8.1 Data Augmentation 11
8.2 Transfer Learning 11
8.3 Batch Normalization 11

9 Aplicações e Arquiteturas Famosas **11**
9.1 Evolução das Arquiteturas 11
9.2 Aplicações Modernas 12

10 Desafios e Limitações **12**
10.1 Limitações das CNNs 12
10.2 Direções Futuras 13

11 Conclusões **13**

1 Introdução

As Redes Neurais Convolucionais (CNNs) representam um dos avanços mais significativos em aprendizado de máquina, especialmente para processamento de imagens. Enquanto as redes neurais tradicionais (Multi-Layer Perceptrons - MLPs) tratam todos os inputs de forma independente, as CNNs foram projetadas especificamente para explorar a estrutura espacial presente em dados visuais.

Este documento explora a motivação por trás das CNNs, seus componentes fundamentais, e como elas superam as limitações das MLPs para tarefas de visão computacional. Veremos que as CNNs são, essencialmente, MLPs especializadas que incorporam conhecimento sobre a natureza hierárquica e local dos padrões visuais.

2 Motivação: As Limitações das MLPs para Imagens

2.1 O Problema da Explosão de Parâmetros

Quando aplicamos MLPs diretamente a imagens, enfrentamos imediatamente um problema de escala. Considere uma imagem colorida modesta de 200×200 pixels com três canais de cor (RGB). Isso resulta em $200 \times 200 \times 3 = 120.000$ valores de entrada. Em uma MLP totalmente conectada, cada neurônio da primeira camada oculta precisa de uma conexão com cada um desses 120.000 pixels.

Observação 2.1: Cálculo de Parâmetros

Para uma única camada totalmente conectada processando uma imagem 200×200 RGB para outra representação do mesmo tamanho:

$$\text{Parâmetros} = (200 \times 200 \times 3)^2 = 14.400.000.000 \text{ pesos}$$

Isso é 14,4 bilhões de parâmetros para **apenas uma camada**!

Esse número astronômico de parâmetros traz várias consequências negativas:

- **Requisitos computacionais**: Memória e processamento tornam-se proibitivos
- **Overfitting**: Com tantos parâmetros, o modelo pode facilmente memorizar os dados de treino
- **Necessidade de dados**: Seriam necessários conjuntos de dados enormes para treinar adequadamente
- **Tempo de treinamento**: A convergência seria extremamente lenta

2.2 Perda de Estrutura Espacial

Definição 2.1: Achatamento de Dados

MLPs requerem que imagens 2D (ou 3D com canais de cor) sejam "achatadas"em vetores 1D. Esse processo destrói completamente a topologia espacial da imagem:

$$\text{Imagem}_{H\times W\times C} \to \text{Vetor}_{H\cdot W\cdot C\times 1}$$

onde H é altura, W é largura, e C é o número de canais.

Quando achatamos uma imagem, pixels que eram vizinhos podem ficar distantes no vetor resultante. Por exemplo, dois pixels adjacentes horizontalmente estarão separados por W posições no vetor achatado. Isso significa que a MLP precisa aprender essas relações espaciais do zero, sem nenhuma estrutura inerente que facilite esse aprendizado.

2.3 Ausência de Invariância à Translação

Um dos problemas mais críticos das MLPs para visão é a falta de **invariância à translação**. Se treinamos uma MLP para reconhecer um gato no canto superior esquerdo de uma imagem, ela não necessariamente reconhecerá o mesmo gato se ele aparecer no canto inferior direito.

Teorema 2.1: Invariância à Translação

Uma função f é invariante à translação se:

$$f(T_\delta(x)) = f(x)$$

onde T_δ é um operador de translação por δ pixels. MLPs não possuem essa propriedade naturalmente, enquanto CNNs a incorporam através do compartilhamento de pesos.

3 A Natureza Hierárquica da Visão

3.1 Como o Cérebro Processa Informação Visual

O sistema visual humano processa informações de forma hierárquica, uma descoberta fundamental de Hubel e Wiesel que lhes rendeu o Prêmio Nobel. Neurônios no córtex visual primário (V1) respondem a características simples como bordas em orientações específicas. Áreas superiores combinam essas características simples em representações cada vez mais complexas.

Observação 3.1: Hierarquia Visual Biológica

O processamento visual no cérebro segue uma hierarquia clara:

1. **V1**: Detectores de bordas e orientação
2. **V2**: Formas simples e texturas
3. **V4**: Formas intermediárias e cores
4. **IT (Córtex Inferotemporal)**: Objetos completos e faces

3.2 Construção de Representações Complexas

As CNNs imitam essa hierarquia natural. Consideremos como uma CNN pode aprender a reconhecer um gato:

Exemplo 3.1: Hierarquia de Features para Reconhecimento de Gato

1. **Primeira camada**: Aprende detectores de bordas em várias orientações
2. **Segunda camada**: Combina bordas para formar círculos e curvas
3. **Terceira camada**: Combina círculos para formar olhos; curvas para formar orelhas
4. **Camadas posteriores**: Combina olhos, orelhas e textura de pelo para reconhecer face de gato
5. **Camada final**: Integra todas as características para classificação

Essa abordagem hierárquica é fundamental porque permite que a rede reutilize características aprendidas. Um detector de bordas aprendido pode ser útil para reconhecer não apenas gatos, mas também cachorros, carros, ou qualquer outro objeto.

4 Convolução: O Coração das CNNs

4.1 Operação de Convolução

A operação de convolução é o elemento fundamental que dá nome às CNNs. Em vez de conectar cada neurônio a todos os pixels da imagem, a convolução usa um pequeno filtro (ou kernel) que "desliza"sobre a imagem.

Definição 4.1: Convolução Discreta 2D

Dados uma imagem I e um kernel K de tamanho $m \times n$, a convolução é definida como:

$$(I * K)(i, j) = \sum_{p=0}^{m-1} \sum_{q=0}^{n-1} I(i + p, j + q) \cdot K(p, q)$$

onde (i, j) é a posição na imagem de saída.

A convolução tem várias propriedades importantes que a tornam ideal para processamento de imagens:

- **Localidade**: Cada neurônio vê apenas uma pequena região da imagem
- **Compartilhamento de pesos**: O mesmo filtro é aplicado em toda a imagem
- **Equivariância à translação**: Se a entrada se move, a saída se move proporcionalmente

4.2 Filtros como Detectores de Características

Filtros convolucionais atuam como detectores de características locais. Diferentes configurações de pesos no filtro permitem detectar diferentes padrões:

Exemplo 4.1: Filtros Clássicos

Detector de Bordas Verticais (Sobel):

$$K_{vertical} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Detector de Bordas Horizontais:

$$K_{horizontal} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Filtro de Nitidez (Sharpening):

$$K_{sharp} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Em CNNs, esses filtros não são definidos manualmente, mas **aprendidos através de backpropagation**. Isso permite que a rede descubra automaticamente quais características são mais úteis para a tarefa em questão.

5 Arquitetura de uma CNN

5.1 Camadas Convolucionais

As camadas convolucionais são o componente principal das CNNs. Cada camada convolucional contém múltiplos filtros, cada um produzindo um "mapa de características" (feature map) diferente.

Definição 5.1: Camada Convolucional

Uma camada convolucional com N filtros transforma uma entrada com C_{in} canais em uma saída com $C_{out} = N$ canais:

$$\text{Input}_{H \times W \times C_{in}} \xrightarrow{N \text{ filtros}} \text{Output}_{H' \times W' \times N}$$

onde H' e W' dependem do padding e stride utilizados.

5.1.1 Hiperparâmetros das Camadas Convolucionais

Observação 5.1: Hiperparâmetros Principais

- **Tamanho do filtro**: Geralmente 3×3, 5×5, ou 7×7. Filtros menores (3×3) são preferidos em redes modernas
- **Número de filtros**: Tipicamente aumenta em camadas mais profundas (32 → 64 → 128 → 256)
- **Stride**: Passo do filtro. Stride=1 mantém resolução espacial, stride=2 reduz pela metade
- **Padding**: Adiciona zeros nas bordas. "Same"padding preserva dimensões, "valid"padding as reduz

5.2 Camadas de Pooling

Pooling é uma operação de subamostragem que reduz a dimensionalidade espacial dos mapas de características, diminuindo o número de parâmetros e computação na rede.

Definição 5.2: Max Pooling

Max pooling com janela $k \times k$ e stride s seleciona o valor máximo em cada região:

$$y_{i,j} = \max_{p,q \in R_{i,j}} x_{p,q}$$

onde $R_{i,j}$ é a região $k \times k$ começando na posição $(i \cdot s, j \cdot s)$.

Teorema 5.1: Efeito do Pooling na Dimensionalidade

Para uma entrada de tamanho $H \times W$, pooling com janela $k \times k$ e stride s produz saída de tamanho:

$$H_{out} = \left\lfloor \frac{H - k}{s} \right\rfloor + 1, \quad W_{out} = \left\lfloor \frac{W - k}{s} \right\rfloor + 1$$

Max pooling 2×2 com stride 2 reduz cada dimensão espacial pela metade.

5.2.1 Motivação para Pooling

O pooling serve múltiplos propósitos:

1. **Redução de parâmetros**: Diminui o tamanho dos feature maps
2. **Invariância local**: Pequenas translações na entrada não afetam a saída
3. **Campo receptivo maior**: Neurônios em camadas posteriores "veem"regiões maiores da imagem original
4. **Controle de overfitting**: Menos parâmetros significa menor capacidade de memorização

5.3 Arquitetura Típica

Uma CNN típica segue um padrão de camadas que alternam entre convolução e pooling, seguidas por camadas totalmente conectadas no final:

Algoritmo 5.1: Arquitetura CNN Padrão

1. **Bloco 1**: Conv → ReLU → Conv → ReLU → MaxPool
2. **Bloco 2**: Conv → ReLU → Conv → ReLU → MaxPool
3. **Bloco 3**: Conv → ReLU → Conv → ReLU → MaxPool
4. **Flatten**: Achata o tensor 3D em vetor 1D
5. **Classificador**: FC → ReLU → FC → ReLU → FC → Softmax

6 Comparação Detalhada: CNN vs MLP

6.1 Análise de Parâmetros

A diferença no número de parâmetros entre CNNs e MLPs é dramática. Vamos comparar duas arquiteturas para processar imagens 32×32 em escala de cinza:

Exemplo 6.1: Comparação de Parâmetros

MLP com uma camada oculta de 100 neurônios:

- Entrada para oculta: $32 \times 32 \times 100 = 102.400$ parâmetros
- Oculta para saída (10 classes): $100 \times 10 = 1.000$ parâmetros
- Total: 103.400 parâmetros

CNN simples:

- Conv1 (32 filtros 3×3): $3 \times 3 \times 1 \times 32 = 288$ parâmetros
- Conv2 (64 filtros 3×3): $3 \times 3 \times 32 \times 64 = 18.432$ parâmetros
- FC (após pooling para 7×7×64): $7 \times 7 \times 64 \times 10 = 31.360$ parâmetros
- Total: 50.000 parâmetros

A CNN tem **metade dos parâmetros** mas geralmente alcança melhor performance!

6.2 Propriedades Comparativas

Observação 6.1: Vantagens das CNNs sobre MLPs

1. **Eficiência de parâmetros**: CNNs usam compartilhamento de pesos
2. **Preservação de estrutura**: Mantém relações espaciais
3. **Invariância à translação**: Built-in através de convoluções
4. **Composicionalidade**: Hierarquia natural de características
5. **Generalização**: Melhor performance com menos dados

7 Implementação Prática

7.1 Exemplo em TensorFlow/Keras

Vamos implementar uma CNN simples para o dataset MNIST, demonstrando os conceitos discutidos:

```
import tensorflow as tf
from tensorflow.keras import layers, models

def criar_cnn_mnist():
    """
    Cria uma CNN simples para classificação de dígitos MNIST.
    Arquitetura: Conv->Pool->Conv->Pool->FC->Output
    """
    model = models.Sequential([
        # Primeira camada convolucional
```

```
        layers.Conv2D(32, (3, 3), activation='relu',
                      input_shape=(28, 28, 1)),
        layers.MaxPooling2D((2, 2)),

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

        # Terceira camada convolucional
        layers.Conv2D(64, (3, 3), activation='relu'),

        # Flatten e camadas densas
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])

    return model

# Compilar modelo
model = criar_cnn_mnist()
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Visualizar arquitetura
model.summary()
```

7.2 Análise da Arquitetura

Vamos analisar como as dimensões mudam através da rede:

Observação 7.1: Fluxo de Dimensões

Para entrada MNIST (28×28×1):

1. Input: 28×28×1
2. Após Conv2D(32, 3×3): 26×26×32 (sem padding)
3. Após MaxPool(2×2): 13×13×32
4. Após Conv2D(64, 3×3): 11×11×64
5. Após MaxPool(2×2): 5×5×64
6. Após Conv2D(64, 3×3): 3×3×64
7. Após Flatten: 576
8. Após Dense(64): 64
9. Output: 10

Total de parâmetros: 100k (compare com 800k para MLP equivalente)

8 Técnicas Avançadas e Boas Práticas

8.1 Data Augmentation

Data augmentation é crucial para melhorar a generalização de CNNs, especialmente com datasets limitados:

Algoritmo 8.1: Pipeline de Data Augmentation

```
data_augmentation = tf.keras.Sequential([
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
    layers.RandomFlip("horizontal"),
    layers.RandomTranslation(0.1, 0.1),
    layers.RandomContrast(0.1)
])
```

8.2 Transfer Learning

Transfer learning permite aproveitar redes pré-treinadas em grandes datasets:

Observação 8.1: Estratégias de Transfer Learning

1. **Feature Extraction**: Congela camadas convolucionais, treina apenas classificador
2. **Fine-tuning**: Descongela últimas camadas convolucionais para ajuste fino
3. **Inicialização**: Usa pesos pré-treinados como ponto de partida

8.3 Batch Normalization

Batch normalization acelera o treinamento e melhora a estabilidade:

Definição 8.1: Batch Normalization

Para cada mini-batch, normaliza as ativações:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$y_i = \gamma \hat{x}_i + \beta$$

onde μ_B e σ_B^2 são média e variância do batch, γ e β são parâmetros aprendidos.

9 Aplicações e Arquiteturas Famosas

9.1 Evolução das Arquiteturas

A evolução das CNNs pode ser traçada através de arquiteturas marcos:

Observação 9.1: Arquiteturas Históricas

- **LeNet-5 (1998)**: Pioneira, 7 camadas, 60k parâmetros
- **AlexNet (2012)**: Vencedora ImageNet, 8 camadas, 60M parâmetros
- **VGGNet (2014)**: Filtros 3×3 uniformes, 16-19 camadas, 138M parâmetros
- **GoogLeNet (2014)**: Módulos Inception, 22 camadas, 5M parâmetros
- **ResNet (2015)**: Conexões residuais, 152+ camadas, 60M parâmetros
- **EfficientNet (2019)**: Otimização de arquitetura, escalável

9.2 Aplicações Modernas

As CNNs revolucionaram múltiplas áreas:

Exemplo 9.1: Aplicações de CNNs

Visão Computacional:

- Classificação de imagens (ImageNet, CIFAR)
- Detecção de objetos (YOLO, Faster R-CNN)
- Segmentação semântica (U-Net, DeepLab)
- Reconhecimento facial (FaceNet, DeepFace)

Medicina:

- Diagnóstico de câncer em imagens médicas
- Análise de retina para detecção de doenças
- Segmentação de órgãos em tomografias

Indústria:

- Inspeção de qualidade automatizada
- Veículos autônomos
- Agricultura de precisão

10 Desafios e Limitações

10.1 Limitações das CNNs

Apesar do sucesso, CNNs têm limitações importantes:

Observação 10.1: Limitações Conhecidas

1. **Necessidade de dados**: Requerem grandes datasets para treinar do zero
2. **Interpretabilidade**: Difícil entender o que a rede aprendeu
3. **Sensibilidade a adversários**: Pequenas perturbações podem enganar a rede
4. **Falta de entendimento geométrico**: Não capturam relações parte-todo naturalmente
5. **Custo computacional**: Treinamento e inferência podem ser caros

10.2 Direções Futuras

Pesquisas atuais buscam superar essas limitações:

Observação 10.2: Tendências de Pesquisa

- **Vision Transformers**: Aplicação de attention mechanisms para visão
- **Capsule Networks**: Melhor modelagem de hierarquias parte-todo
- **Neural Architecture Search**: Automação do design de arquiteturas
- **Efficient CNNs**: Modelos menores e mais rápidos para dispositivos móveis
- **Self-supervised learning**: Reduzir dependência de dados rotulados

11 Conclusões

As Redes Neurais Convolucionais representam uma especialização bem-sucedida das MLPs para o domínio visual. Ao incorporar conhecimento sobre a estrutura hierárquica e local das imagens através de convoluções e pooling, as CNNs conseguem alcançar performance superior com ordens de magnitude menos parâmetros.

Os princípios fundamentais das CNNs - localidade, compartilhamento de pesos, e hierarquia composicional - não são apenas truques de engenharia, mas refletem propriedades fundamentais do processamento visual biológico e da estrutura dos dados visuais.

Observação 11.1: Resumo dos Conceitos Principais

1. CNNs são MLPs especializadas que exploram estrutura espacial
2. Convolução permite detecção local de características com compartilhamento de pesos
3. Pooling reduz dimensionalidade e adiciona invariância local
4. Arquitetura hierárquica constrói representações complexas de simples
5. Transfer learning e data augmentation são essenciais na prática
6. CNNs revolucionaram visão computacional mas têm limitações conhecidas

O sucesso das CNNs em visão computacional inspirou aplicações em outros domínios com estrutura local, incluindo processamento de áudio (espectrogramas), processamento de texto (CNNs 1D), e até mesmo grafos (Graph CNNs). O princípio de explorar estrutura conhecida nos dados através de arquiteturas especializadas continua sendo uma das ideias mais poderosas em deep learning.