Redes Convolucionais

Created	@May 13, 2025 8:48 PM
Tags	

1. Introdução às Redes Neurais Convolucionais (CNNs)

CNNs são, fundamentalmente, redes neurais que incorporam **operações de convolução**. Elas são usadas principalmente para **processamento de imagens**. Sua criação data de **1989** por **Yann LeCun** com a rede LeNet-5, mas sua popularidade explodiu **após 2012**, impulsionada por redes como a AlexNet.

A motivação para o uso de CNNs, especialmente com imagens, vem das limitações das Multilayer Perceptrons (MLPs). Imagens possuem características que as CNNs exploram de maneira mais eficiente:

- **Alta dimensionalidade**: Uma imagem 512×512 tem 262144 pixels. Em uma MLP, mesmo com 100 unidades na primeira camada oculta, isso resultaria em mais de 26 milhões de pesos apenas para essa camada, consumindo muita memória.
- **Alta coesão espacial**: Pixels próximos em uma imagem são altamente correlacionados. As CNNs aproveitam essa propriedade.
- **Semântica estável frente a transformações geométricas**: Uma característica visual (como a orelha de um animal) mantém seu significado mesmo em locais diferentes da imagem. CNNs conseguem lidar com isso sem reaprender o padrão em cada local.

Historicamente, a ideia por trás das CNNs tem raízes nas experimentações de David Hubel e Torsten Wiesel no córtex visual de gatos, que sugeriram que o processamento visual começa identificando padrões simples como arestas.

Benefícios das Camadas Convolucionais:

- Processam **regiões da imagem separadamente**, o que permite o **compartilhamento de pesos**.

- Utilizam **menos parâmetros** em comparação com camadas totalmente conectadas.
- Exploram a **correlação entre pixels próximos**.
- Não precisam reaprender os mesmos padrões em diferentes posições na imagem.

2. Conceitos Chave em CNNs

1. **Convolução**: É a operação central. Em 1D, transforma um vetor de entrada em um vetor de saída, onde cada elemento de saída é a soma ponderada de uma pequena região do vetor de entrada. Os pesos utilizados nessa soma são compartilhados por toda a entrada. Esse conjunto de pesos é chamado de **kernel** ou **filtro**. A convolução pode ser estendida naturalmente para 2D.
2. **Camada Convolucional**: Realiza a operação de convolução, adiciona um **bias** e aplica uma **função de ativação**. É um caso especial de camada totalmente conectada. As saídas de um filtro são armazenadas em **feature maps** (ou canais).
3. **Parâmetros Treináveis**: Em uma camada convolucional com Ci canais de entrada, Co canais de saída e kernel KxK, o número de parâmetros é dado por Ci × Co × K × K (para os kernels) mais Co (para os biases). CNNs demonstram que é possível alcançar ótimos resultados com significativamente menos parâmetros treináveis do que MLPs, como visto no exemplo do MNIST 1D.
4. **Campo Receptivo (Receptive Field)**: É a região da imagem de entrada que influencia a saída de uma unidade específica em uma camada oculta (um feature map). Em camadas mais profundas, o campo receptivo aumenta. Um campo receptivo pequeno pode perder informações relevantes, enquanto um muito grande pode se tornar semelhante a uma camada densa. Aplicar duas convoluções 3×3 sucessivas é geralmente preferível a uma única 5×5, pois resulta no mesmo campo receptivo, mas com menos parâmetros e mais não-linearidade.
5. **Padding**: Método para lidar com as bordas da imagem durante a convolução.
    - **Zero Padding**: Completa a entrada com zeros.

- **Valid Padding**: Desconsidera as bordas, aplicando a convolução apenas onde o kernel cabe.
- **Circular/Periódico**: Pouco utilizado.

6. **Stride**: Define o tamanho do passo (transladação) do kernel pela entrada. Um stride maior que 1 pode ser usado para **downsampling**.
7. **Kernel Size**: O tamanho do filtro (HxW). Permite integrar informações espaciais mais distantes, mas aumenta o número de pesos.
8. **Dilation**: Permite integrar informações espaciais mais distantes sem aumentar o número de parâmetros, equivalente a ter um kernel maior com alguns pesos zerados.
9. **Convolução 1×1**: Usada para mudar o número de canais de um feature map sem agrupar informações espacialmente.

3. Camada Convolucional — o que ela faz?

Imagine que você tem uma régua com 3 números e quer "deslizar" ela por uma lista de dados (como uma imagem ou sequência). A cada posição, você faz uma conta com os números dessa régua e os dados por onde ela está passando. Isso é a **convolução**!

A conta feita é assim:

```
resultado = bias + (peso1 × valor da esquerda) + (peso2 × valor do meio) + (p
eso3 × valor da direita)
```

Esses **pesos** e o **bias** são ajustados durante o treinamento. A saída dessa conta vira um **pixel do novo mapa**, chamado de **feature map** (ou canal). Se você tiver vários desses "detectores", cria vários canais.

4. Campo Receptivo — o que é isso?

É só o **pedacinho da imagem (ou dado)** que cada neurônio da camada olha. Ele **não vê tudo**, apenas uma janelinha. Esse pedaço é o "campo de visão" dele, que chamamos de **campo receptivo**.

Se você empilhar várias camadas, o campo receptivo aumenta: cada neurônio passa a "ver" mais da imagem original, mesmo que indiretamente.

5. Exemplo de CNN — estrutura e desempenho

Imagine uma rede com:

- 3 camadas convolucionais (que extraem padrões)
- 1 camada final totalmente conectada (que toma decisões)

Essa rede teve:

- Apenas **2050 parâmetros** (pesos e bias a aprender)
- Foi treinada com 100 mil exemplos
- Usou **SGD** (um tipo de treino por tentativa e erro)
- **Aprendeu melhor** do que uma rede tradicional (MLP), mesmo sendo menor

---

6. Quantos parâmetros uma camada convolucional tem?

Fórmula:

```
número de parâmetros = canais de entrada × canais de saída × tamanho do k
ernel × tamanho do kernel + bias
```

Exemplo:

- 3 canais de entrada
- 5 canais de saída
- kernel 3×3

Cálculo:

```
3 × 5 × 3 × 3 = 135 (pesos) + 5 (bias) = 140 parâmetros
```

O slide dizia 185, mas errou na conta. 😅

7. Campo Receptivo em 2D (imagem)

Em imagens, usamos filtros como 3×3 (pequenos quadrados). Duas camadas 3×3 juntas "enxergam" como se fosse uma camada 5×5 — **mas com menos pesos** e **mais flexibilidade**.

Por isso, ao invés de fazer uma camada grande logo de cara, é melhor usar duas pequenas em sequência.

Invariância e Equivariância

Estes são conceitos importantes relacionados a como as CNNs respondem a transformações na entrada:

- **Invariância**: Uma função `f` é invariante a uma transformação `t` se o resultado **não muda** após a transformação, ou seja, `f(x) = f(t(x))`.
- **Equivariância**: Uma função `f` é equivariante a uma transformação `t` se aplicar a transformação `t` à entrada `x` resulta na mesma transformação aplicada à saída da função `f`, ou seja, `f(t(x)) = t(f(x))`. Um exemplo é a segmentação de imagem: transladar a imagem de entrada deve resultar na segmentação transladada. Camadas convolucionais são naturalmente equivariantes a translações devido ao compartilhamento de pesos.

8. Downsampling e Upsampling

Operações que alteram a **resolução espacial dos feature maps**.

Downsampling (Subamostragem): Reduz a resolução espacial, geralmente para tornar a computação mais eficiente e aumentar o campo receptivo efetivo das camadas seguintes.

Métodos comuns:

- **Stride > 1**: A própria convolução é feita com passos maiores.
- **Max Pooling**: Seleciona o **maior valor** em uma janela da imagem.
  - Exemplo: `[[1, 3], [2, 4]]` → `max = 4`.
- **Average Pooling**: Faz a **média** dos valores na janela.
  - Exemplo: `[[1, 3], [2, 4]]` → média = 2.5.

Upsampling (Sobreamostragem): Aumenta a resolução espacial, útil para gerar imagens maiores, como em segmentação ou geração de imagens.

Métodos comuns:

- **Duplicação de valores (Nearest Neighbor)**: Copia os valores existentes.
  - Exemplo: `[1, 2]` → `[1, 1, 2, 2]`.
- **Max Unpooling**: Restaura a posição do valor máximo da operação de Max Pooling anterior.
- **Interpolação Bilinear**: Estima novos valores com base nos vizinhos.
- **Convolução Transposta**: Também chamada de *deconvolução*, é uma forma de convolução que aumenta o tamanho da imagem. Cada valor da entrada contribui para vários valores da saída, com base nos pesos do kernel.

9. Convolução Transposta — aumentar o tamanho

É o oposto da convolução normal: **serve para aumentar** a imagem. Por isso, também é chamada de **"deconvolução"**.

Você "espalha" cada valor da entrada na saída, multiplicando pelos pesos do kernel e somando as contribuições. Isso é útil, por exemplo, para reconstruir uma imagem maior (como em redes que geram imagens).

10. Convolução 1x1 — só muda os canais

Aqui você **não mistura os pixels vizinhos**, só **muda o número de canais** (como trocar de 64 canais para 32, por exemplo).

É como fazer uma conta simples para cada pixel, mas com todos os canais — muito usada para deixar a rede mais leve ou combinar informações.