Advanced Institute for Artificial Intelligence

Redes Neurais Convolucionais

Advanced Institute for Artificial Intelligence – AI2

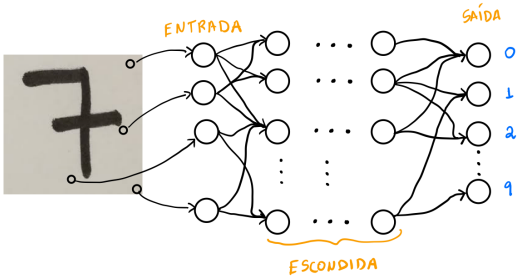
https://advancedinstitute.ai

Introdução

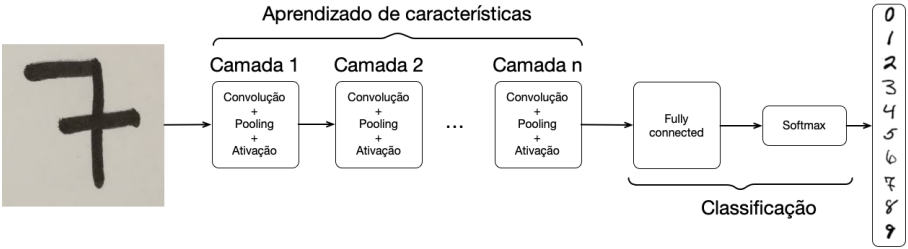
Técnicas de **aprendizado em profundidade**, do inglês *deep learning*, pertencem a um ramo da área de aprendizado de máquina e que fazem uso de redes neurais com diversas camadas. A ideia é, basicamente, empregar camadas para aprender, progressivamente, diferentes níveis de características a partir de um dado de entrada. Os níveis de abstração vão aumentando à medida que mais camadas são utilizadas para extração e aprendizado das características.

Dentre as técnicas de aprendizado em profundidade, uma atenção especial tem sido dada às Redes Neurais Convolucionais, do inglês *Convolutional Neural Networks* - CNNs. Tais modelos possuem uma alta capacidade de representação dos dados, com resultados promissores em inúmeras áreas do conhecimento.

A ideia consiste, basicamente, em utilizar "dados crus" (*raw data*) como entrada e permitir com que a rede aprenda as características que são mais importantes para o problema em questão. Desta forma, eliminamos a necessidade de extrair características de maneira manual (*handcrafted features*).



De maneira geral, CNNs são compostas por dois módulos principais: (i) aprendizado de características e (ii) classificação. O aprendizado de características é realizado por meio de operações de convolução, agrupamento *pooling* e aplicação da função de ativação. Já a etapa de classificação é composta, usualmente, por camadas do tipo *fully connected* e uma camada de saída do tipo *softmax*.

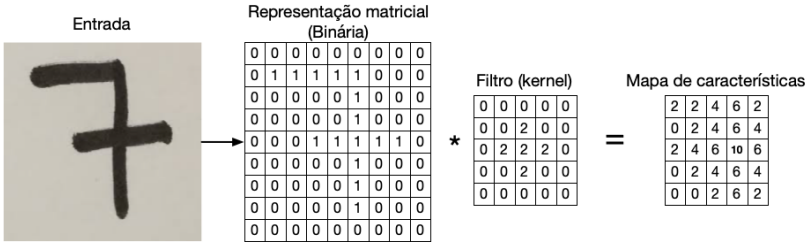


Mas o que torna CNNs tão interessantes para diversas tarefas de classificação de padrões? O segredo está na etapa de **aprendizado de características**, em que informações importantes (textura, por exemplo) são aprendidas em diferentes níveis. O interessante é que essas redes são menos susceptíveis à problemas de rotação, translação e escala.

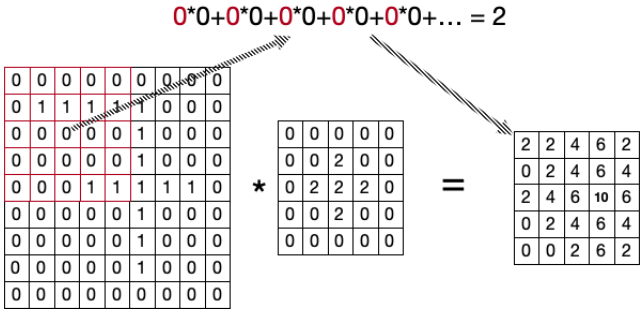
Para entender o seu funcionamento, vamos estudar, primeiramente, o funcionamento de suas camadas de aprendizado de características e depois partir para as camadas de classificação. Como mencionado, cada camada da etapa de aprendizado é composta, basicamente, por operações de (ii) convolução, (iii) pooling e (iii) ativação.

Convolução

A primeira operação que veremos é a de **convolução**, que é amplamente utilizada em tarefas de processamento de imagens e visão computacional, tais como filtragem de imagens (borramento e ruído) e detecção de bordas, por exemplo. Vejamos um exemplo.



A posição central da matriz de entrada 9×9 é substituída pelo valor obtido pela sua convolução com a máscara 5×5 e o valor armazenado na matriz de saída 5×5 (mapa de características). O procedimento é repetido até que toda a matriz de entrada tenha sido avaliada. Note que um trecho da matriz é descartado, por isso que a saída possui dimensões menores (podemos contornar isso com o *zero padding* ou espelhamento da imagem).

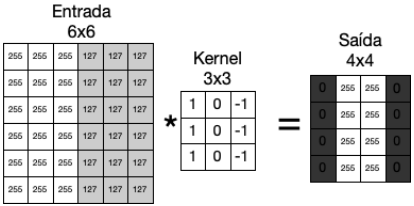


Como calcular o tamanho da saída? Suponha que as dimensões do dado de entrada e da máscara sejam, respectivamente, $m \times m$ e $n \times n$. O dado de saída terá a dimensão de $m - \lfloor 2n/2 \rfloor \times m - 2\lfloor n/2 \rfloor$.

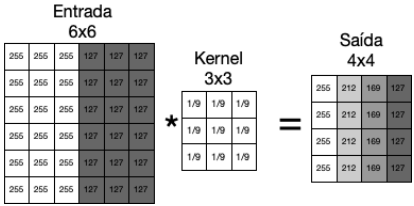
Quais os hiperparâmetros envolvidos em uma operação de convolução? **Parâmetros** $\times$ **hiperparâmetros**.

- Qual tipo de máscara (*kernel*) iremos utilizar?
- Qual a melhor dimensão?
- Quantos filtros serão empregados?
- Qual o valor do *stride*?

Dependendo do tipo de máscara utilizado, diferentes informações são obtidas na saída (*feature maps*). Que tipo de informação de saída temos nos exemplos abaixo?

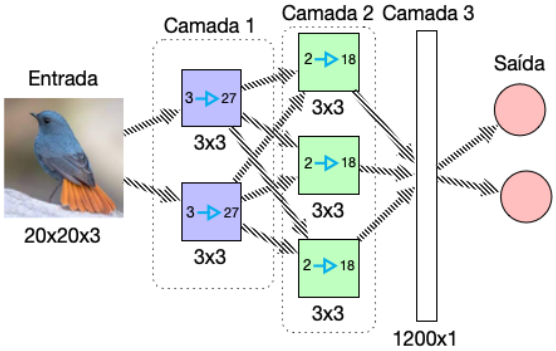


(a) Filtro passa-altas



(b) Filtro passa-baixas

O que temos, na prática, são valores nas máscaras que podem ser interpretados como **pesos** que serão aprendidos pela CNN durante o seu processo de treinamento. Como calcular a quantidade desses parâmetros? Vejamos o exemplo abaixo (estamos assumindo *padding*).



Camada	Tipo	Entrada	Saída
Entrada	Dado	0	3
Intermediária 1	Convolucional	3	(27+27)+2 biases = 56
Intermediária 2	Convolucional	2	54+3 biases = 57
Intermediária 3	Flattened	-	20*20*3 = 1.200
Saída	Densa	1.200	1.200*2+2 biases = 2402

Quantidade de parâmetros a serem aprendidos: **2.515**.

Qual o papel dos valores dos hiperparâmetros de uma CNN?

- Tamanho do *kernel*: possui um papel muito importante em tarefas de classificação de imagens. *Kernels* pequenos extraem uma quantidade maior de informações (**informações locais**) pois as reduções de tamanho entre as camadas são menores permitindo, assim, arquiteturas mais profundas. Por outro lado, *kernels* maiores geram reduções mais rápidas no tamanho dos *feature maps* e extraem informações mais **globais**.
- Valor de *stride*: possui um impacto similar ao tamanho do *kernel*, em que maiores valores proporcionam reduções mais rápidas dos *feature maps*. Valores de *stride* menores resultam em mais características aprendidas.

Dado que valores menores de *stride* e de tamanho de *kernels* proporcionam o aprendizado de mais características, por que não adotá-los sempre? **Isto requer bases de dados maiores.**

Resumindo, temos que tomar decisões sobre os seguintes itens acerca de uma camada de convolução:

1. Tipo do *padding*.
2. Tamanho do *kernel*.
3. Valor de *stride*.
4. Quantidade de filtros.

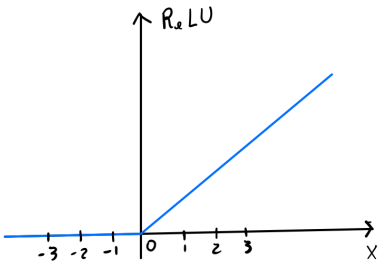
Ativação

CNNs, usualmente, são compostas por inúmeras camadas, não sendo interessante fazer uso de funções de ativação do tipo sigmoide ou tangente hiperbólica (problemas de saturação). Outra característica interessante das funções de ativação é sua característica de **não linearidade**, permitindo com que a rede aprenda funções de decisão não lineares.

Umas das funções de ativação mais utilizadas é a ReLU (*Rectified Linear Unit*) devido à sua simplicidade e alto grau de não linearidade. Sua formulação é dada como segue:

$$\text{ReLU(x)} = \max\{0, x\}. \tag{1}$$

Segue, abaixo, o gráfico da função ReLU(x). Note que a função só retorna algo quando sua entrada possui valor maior do que 0, ajudando no tempo de treinamento.



A derivada da função ReLU(x) é dada como segue:

$$\text{ReLU(x)}' = \begin{cases} 0 \text{ caso } x \leq 0 \\ 1 \text{ caso contrário.} \end{cases} \tag{2}$$

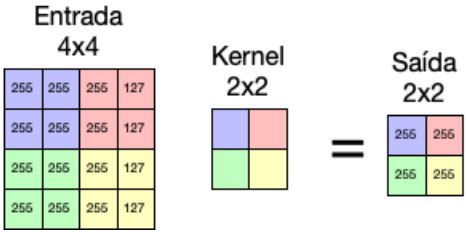
Pooling

Existem diferentes tipos de operações de agrupamento ou *pooling*, cujo objetivo principal é **diminuir a resolução** (*downsampling*) dos *feature maps* e adicionar **propriedades de invariância** à rede. A diminuição do tamanho dos *feature maps* acarreta no decréscimo do número de parâmetros a serem aprendidos pela rede, permitindo um treinamento mais eficiente.

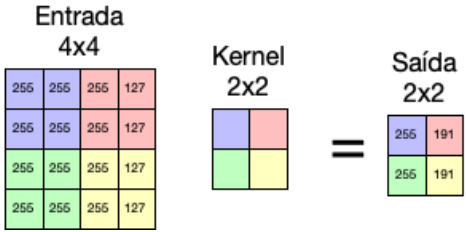
Dentre os principais tipos de *pooling*, podemos citar:

- *Max-Pooling*
- *Average Pooling*
- *Global Pooling*

Seguem algumas ilustrações para exemplificar o funcionamento dos tipos de *pooling* mencionados anteriormente.



(a) Max-Pooling (stride = 2)



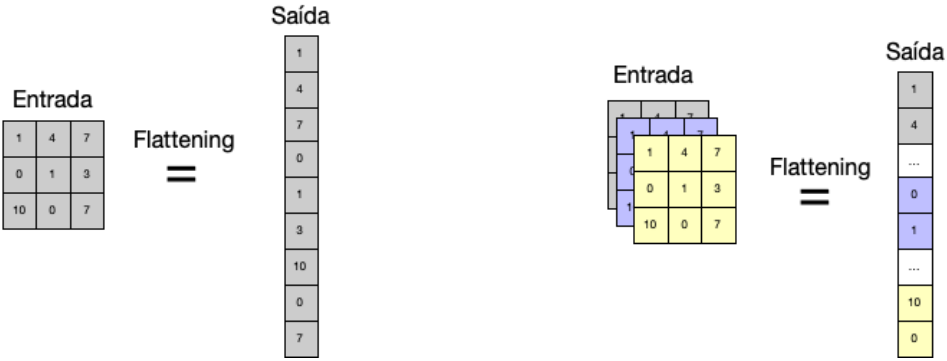
(b) Average Pooling (stride = 2)

Já a técnica de *global pooling* é mais radical no contexto de um *downsampling*, pois reduz todo o *feature map* em um único valor. Neste caso, podemos utilizar tanto *max-pooling* quanto *average pooling*.

Geralmente, camadas de *max-pooling* tendem a prover resultados melhores, pois é mais informativo utilizarmos o maior valor dentro de uma janela do que “mascará-los" por meio do seu valor médio.

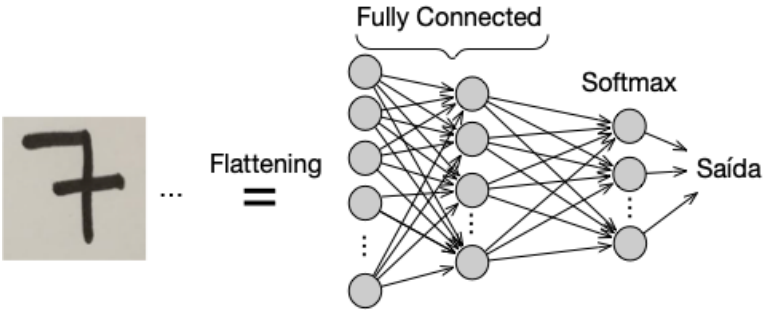
Flattening

Antes de enviar nossos dados que passaram pelas camadas de convolução, *pooling* e ativação para as camadas do tipo *fully connected* ou *dense layers*, precisamos "achatar" o **tensor** (dados). A operação nesta camada é bastante simples, dado que recebemos uma entrada com múltiplas dimensões (*feature maps*) e a saída é um vetor unidimensional, como ilustrado abaixo.

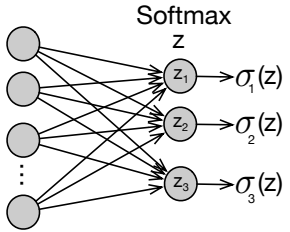


Fully Connected + Dropout

A parte final consiste em adotar camadas do tipo *fully connected*, de maneira similar à uma Rede Neural MLP, com uma saída do tipo *softmax* no final. Também é bastante comum adotarmos uma técnica de regularização conhecida por *Dropout*, a qual "remove" neurônios de maneira aleatória visando acelerar o processo de treinamento e prevenir *overfitting*.



A função *softmax* $\sigma : \mathbb{R}^K \rightarrow [0,1]^K$ é uma generalização da função logística, em que K corresponde ao número de classes.



A sua formulação é dada como segue:

$$\sigma_i(\mathbf{z}) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}. \tag{3}$$

Por que *softmax* e não a função logística? Usualmente, a função logística é aplicada em cada neurônio de saída sem considerarmos todos os outros. No caso, *softmax* acaba resultando em uma probabilidade do neurônio de cada classe responder à um estímulo (dado) de entrada.

Tipos de Operações de Convolução

Existem diferentes maneiras de executarmos a mesma tarefa de convolução, tais como:

- Convolução Separável (*Separable Convolution*)
- Convolução Dilatada (*Dilated Convolution*) e
- Convolução Deformável (*Deformable Convolution*)

Usualmente, as duas primeiras produzem o mesmo resultado no final, mas conseguem melhorar a eficiência do modelo em relação às convoluções tradicionais.

Convolução separável

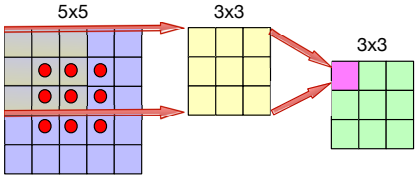
Consiste em "quebrar" a convolução em *kernels* com tamanhos menores. Existem dois tipos de convolução separável:

- Convolução Separável Espacial (*Spatially Separable Convolution*) e
- Convolução Separável em Profundidade (*Depthwise Separable Convolution*)

Vejamos, agora, cada um desses tipos.

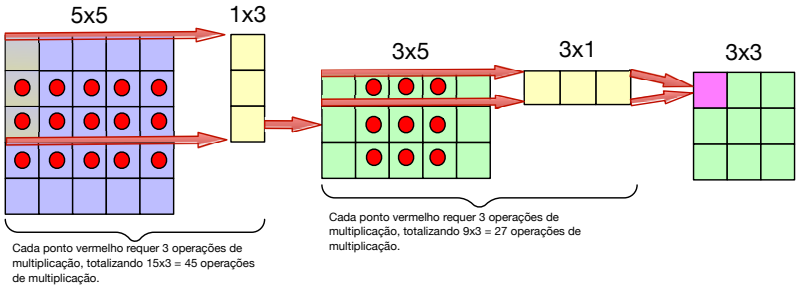
Convolução Separável Espacial

A convolução separável espacial, no caso 2D, consiste em aplicar duas convoluções 1D de maneira sequencial, de tal forma que o resultado final será o mesmo.



Cada um dos pontos vermelhos requer $3 \times 3 = 9$ operações de multiplicação. Desta forma, a convolução de todo o dado de entrada requer $9 \times 9 = 81$ operações de multiplicação.

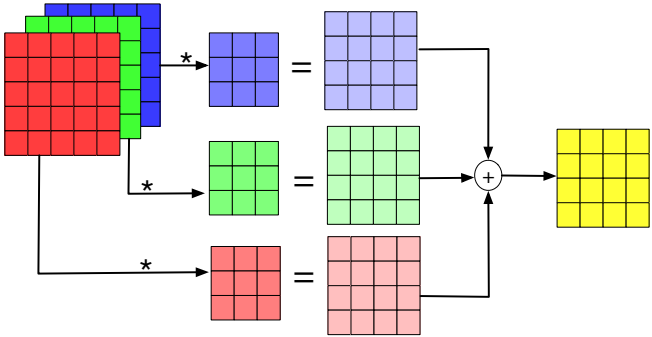
Vejamos, agora, o que acontece quando “quebramos" uma convolução 2D em duas convoluções 1D.



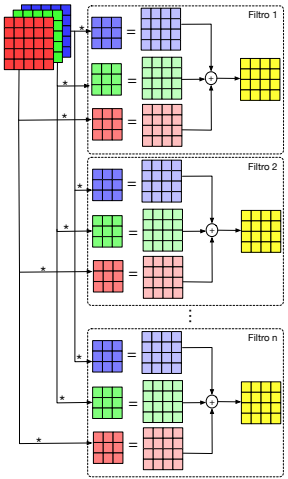
No total, teremos $45 + 27 = 72$ operações de multiplicação.

Convolução Separável em Profundidade

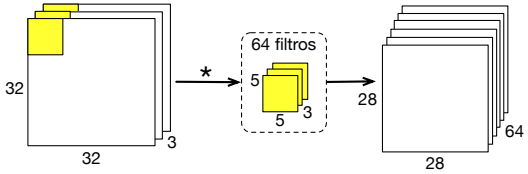
A convolução separável em profundidade é aplicada em dados de entrada multidimensionais (imagens coloridas, por exemplo). A ideia consiste em aplicar a convolução em cada canal (dimensão) separávelmente, e depois aplicar uma convolução do tipo pontual (*pointwise convolution*). Primeiro, vamos entender como é feita a convolução em dados de entrada com múltiplas dimensões.



No caso de múltiplos filtros, a convolução com múltiplas dimensões pode ser representada da seguinte forma.

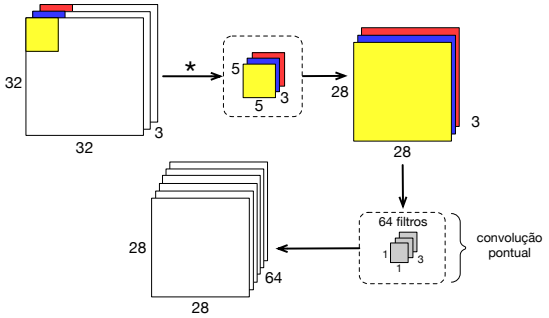


Vejamos, primeiramente, um exemplo de convolução com múltiplos filtros convencional.



Aqui temos uma imagem de entrada 32×32 com 3 canais (pode ser uma imagem colorida, por exemplo) cuja convolução será dada com uma máscara $5 \times 5 \times 3$. Serão aplicados 64 filtros de tal forma que o *feature map* de saída possui a dimensão $28 \times 28 \times 64$. **Assim, teremos uma quantidade de $5 \times 5 \times 3 \times 28 \times 28 \times 64 = 3.763.200$ operações de multiplicação.**

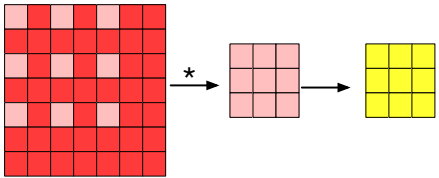
Vejamos, agora, um exemplo de convolução separável em profundidade.



Nossa entrada $32 \times 32 \times 3$ passará por uma convolução com uma máscara $5 \times 5 \times 3$, gerando uma saída com dimensões $28 \times 28 \times 3$ e resultando em $5 \times 5 \times 3 \times 28 \times 28 = 58.800$ operações de multiplicação. Em seguida, os *feature maps* passarão por uma convolução com uma máscara $1 \times 1 \times 3$ com 64 filtros, resultando em $64 \times 1 \times 1 \times 3 \times 28 \times 28 = 150.528$ operações. **Ao final, teremos $58.800 + 150.528 = 290.328$ operações de multiplicação.**

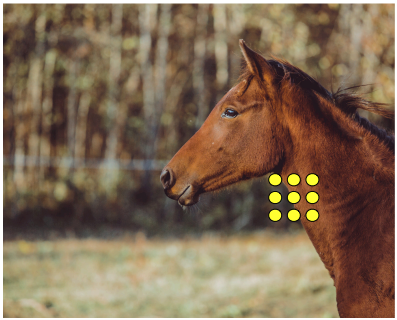
Convolução Dilatada

Operações de convolução convencionais processam todos os vizinhos em conjunto, ou seja, levam em consideração todas as características que estão delimitadas pelo tamanho da máscara. Entretanto, para algumas aplicações, é interessante "pularmos" alguns vizinhos no sentido de ter uma *visão mais ampla* durante o processo de convolução aumentando um parâmetro apenas (quantidade de vizinhos a serem pulados).



Convolução Deformável

Operações de convolução convencionais são bastante rígidas no que diz respeito ao formato das máscaras, isto é, são sempre matrizes quadradas. **E se aprendêssemos o formato dessas máscaras?** Esta é a ideia principal por conta das convoluções deformáveis.



(a) Convolução convencional

(b) Convolução deformável

Treinamento

O processo de treinamento de uma rede neural profunda consiste em encontrar os valores dos parâmetros que minimizam uma função de custo. Quais são esses parâmetros? **Os pesos das máscaras de convolução e das conexões das camadas fully connected.** Note que o tamanho das máscaras, quantidade de filtros e valores de *stride* são denominados de **hiperparâmetros** e, portando, não são aprendidos.

No que diz respeito ao treinamento, precisamos escolher dois itens: (i) função de custo e (ii) técnica otimizadora. Usualmente, adota-se a **entropia cruzada** para a função de custo e a técnica *Adam* para a etapa de otimização.