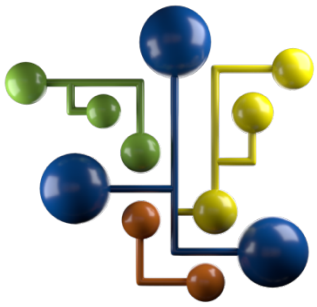


Data Science Academy

www.datascienceacademy.com.br

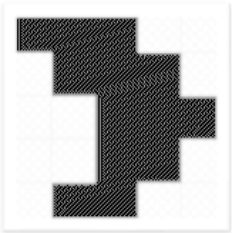
Deep Learning Frameworks

Arquitetura
Convolutional Neural Networks

Arquitetura – Convolutional Neural Network

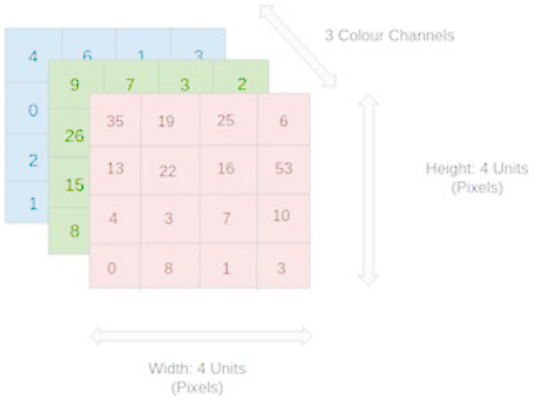
Para explicar a arquitetura das CNNs, vamos considerar a classificação e reconhecimento de dígitos, que estudamos desde o capítulo anterior e compreender como funciona a convolução.

Como sabemos uma imagem real está associada a uma grade (grid) composta por um grande número de pequenos quadrados, chamados pixels. Esta imagem abaixo representa uma imagem em preto e branco com um grid de pixels 5x5:

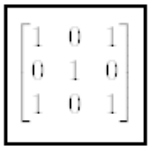


Cada elemento da grade corresponde a um pixel e, no caso de uma imagem em preto e branco, assume um valor igual a 1, que está associado com a cor preta ou o valor 0, que está associado à imagem branca. Em uma imagem de escala de cinza, os valores para cada elemento da grade estão no intervalo [0-255], onde 0 está associado a preto e 255 a branco.

Finalmente, uma imagem colorida é representada por um grupo de três matrizes, cada uma correspondendo a um canal de cor (vermelho, verde, azul). E cada elemento de cada matriz pode variar ao longo de um intervalo inteiro [0.255] que especifica o brilho da cor fundamental (ou cor de base). Esta representação é mostrada na figura a seguir onde cada matriz representada é um tamanho 4x4 e o número de canais de cor é três:

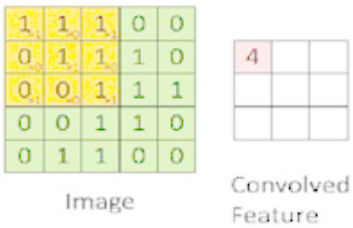


Vamos agora focar a imagem em preto-e-branco (matriz 5x5), e suponhamos que flua nessa matriz, de cima para baixo e da esquerda para a direita, uma segunda matriz de dimensões inferiores, por exemplo uma matriz 3x3, como mostrado na figura a seguir:



Kernel filter

Essa matriz de fluxo é chamada de filtro de kernel ou detector de característica. Enquanto o filtro do kernel se move ao longo da matriz de entrada (ou imagem de entrada), ele executa um produto escalar, entre os valores do kernel e os da parte da matriz à qual é aplicado. O resultado é uma nova matriz, chamada de matriz de convolução. A figura a seguir mostra o procedimento de convolução, a característica convolvida (a matriz 3x3 resultante) é gerada pela operação de convolução, fluindo o filtro do kernel (a matriz 3x3) na imagem de entrada (a matriz 5x5):

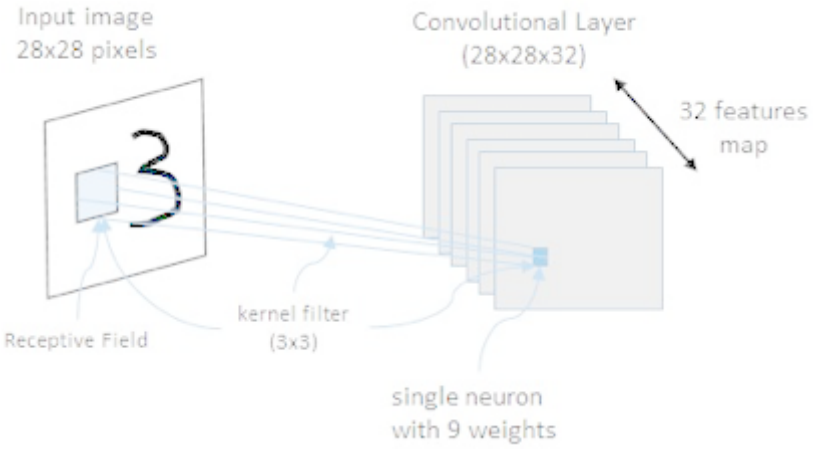


Image

Convolved Feature

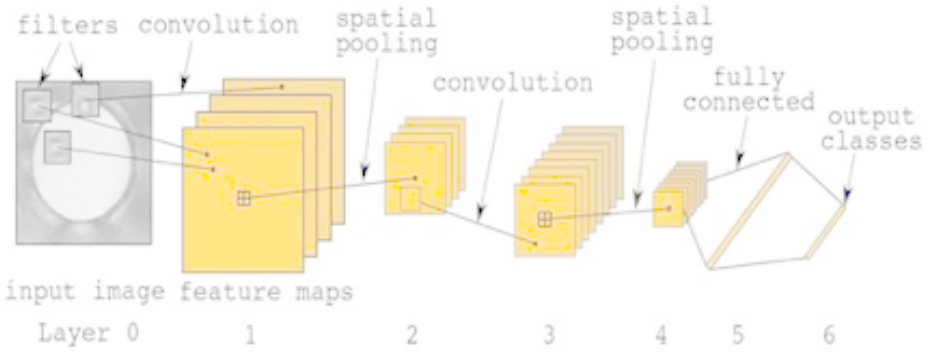
Tomando como exemplo a matriz de entrada 5x5 como mostrado anteriormente, uma CNN consiste em uma camada de entrada constituída por 25 neurônios (5x5 = 25) cuja tarefa é adquirir o valor de entrada correspondente a cada pixel e transferi-lo para a camada oculta seguinte. Em uma rede multicamada, as saídas de todos os neurônios da camada de entrada seriam conectadas a cada neurônio da camada oculta (camada totalmente conectada, estudada no capítulo anterior). Nas redes CNN, o esquema de conexão que define a camada convolucional que vamos descrever é significativamente diferente.

O uso de uma ou mais dessas camadas em uma CNN é indispensável. Em uma camada convolucional, cada neurônio é conectado a uma determinada região da área de entrada chamada “campo receptivo local”. Por exemplo, usando um filtro kernel 3x3, cada neurônio terá um viés e 3x3=9 pesos conectados a um único campo receptivo. É claro que, para reconhecer uma imagem de forma eficaz, precisamos de filtros de kernel diferentes aplicados ao mesmo campo receptivo, porque cada filtro deve reconhecer a imagem de uma característica diferente. O conjunto de neurônios que identificam o mesmo recurso define um único mapa de recursos. A figura a seguir mostra uma arquitetura CNN em ação. A imagem de entrada de um tamanho de 28x28 será analisada por uma camada convolucional composta por um mapa de 32 características com um tamanho de 28x28. A figura também mostra um campo receptivo e o filtro do kernel de um tamanho 3x3:

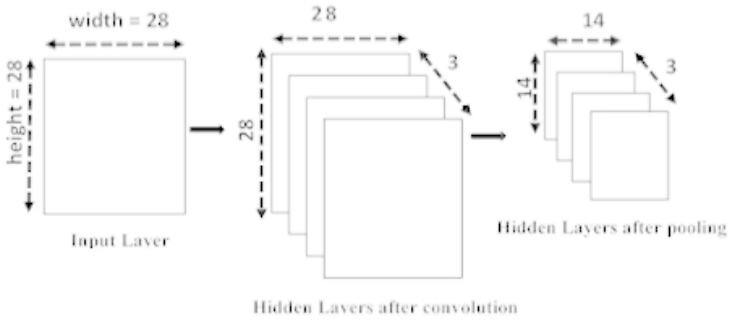


Uma CNN pode consistir em várias camadas de convolução ligadas numa cascata. A saída de cada camada de convolução é um conjunto de mapas de características (cada um gerado por um único filtro de kernel), e todas essas matrizes definem uma nova entrada que será usada pela próxima camada. Geralmente em uma CNN, cada neurônio produz uma saída, após um limiar de ativação, proporcional à entrada e não limitado. Geralmente, a função de ativação usada é a função ReLU.

As CNNs também usam camadas de pooling posicionadas imediatamente após as camadas convolucionais. Uma camada de agrupamento divide uma região convolucional em sub-regiões e seleciona um único valor representativo (max-pooling ou pooling médio) para reduzir o tempo computacional das camadas subsequentes e aumentar a robustez do recurso em relação à sua posição espacial. A última camada oculta de uma rede convolucional é geralmente uma rede totalmente conectada com uma função de ativação softmax para a camada de saída.

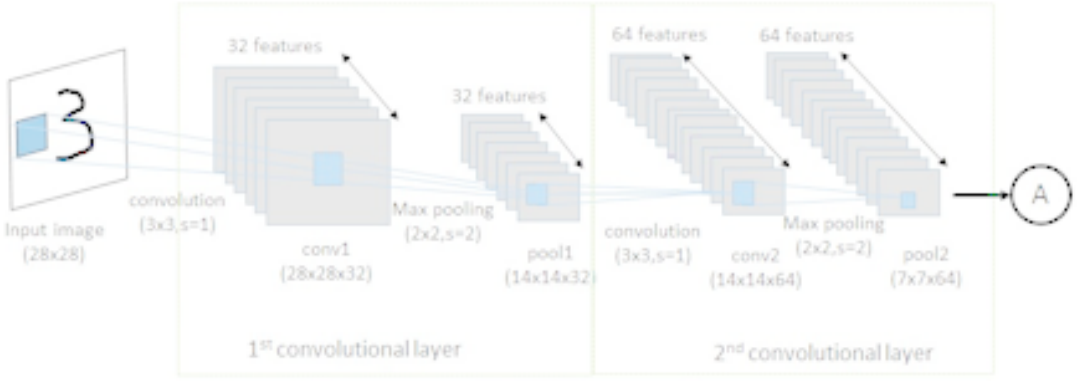


No TensorFlow, a função tf.nn.conv2d() computa convoluções 2D a partir do tensor de input. A esse resultado adicionamos o bias. A função tf.nn.relu() é usada como função de ativação nas camadas ocultas. Aplicamos a ReLu aos valores de retorno das camadas de convolução. O parâmetro padding indica que o tensor de output terá o mesmo tamanho do tensor de entrada. Após a operação de convolução, realizamos o passo de pooling que simplifica a informação de output previamente criada pela camada de convolução. Aplicando essas operações, teremos esse resultado:

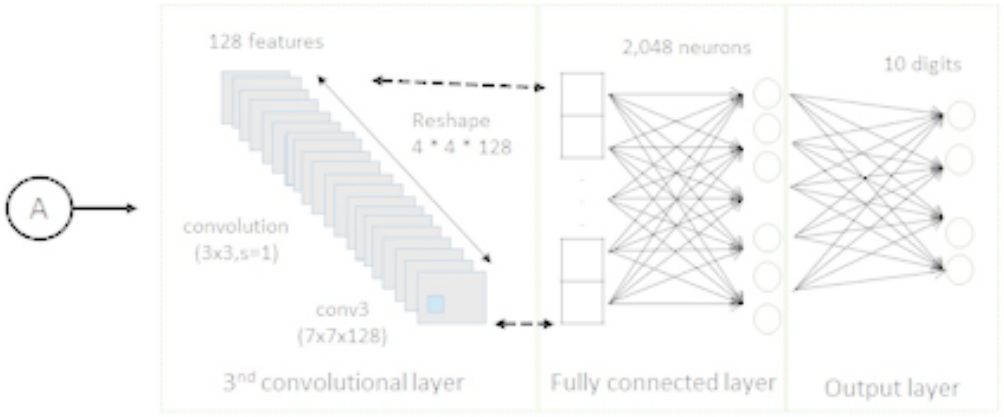


A última operação é reduzir o overfitting, aplicando a função tf.nn.dropout() no TensorFlow. O dropout vai eliminar algumas unidades (nas camadas ocultas, de entrada e de saída) na rede neural. A decisão sobre qual neurônio será eliminado é randômica e aplicamos uma probabilidade para isso. Esse parâmetro pode ser ajustado para otimizar o desempenho da rede. Mas ainda não acabou!!

Repetimos o processo na segunda camada de convolução. Na segunda camada, aplicamos as mesmas operações aplicadas na primeira camada de convolução



Agora sim, por último, construímos a camada totalmente conectada (de forma similar ao que fizemos no capítulo anterior) e treinamos nosso modelo:



Este não é um conceito fácil de compreender. Então vamos construir uma CNN com TensorFlow e praticar!