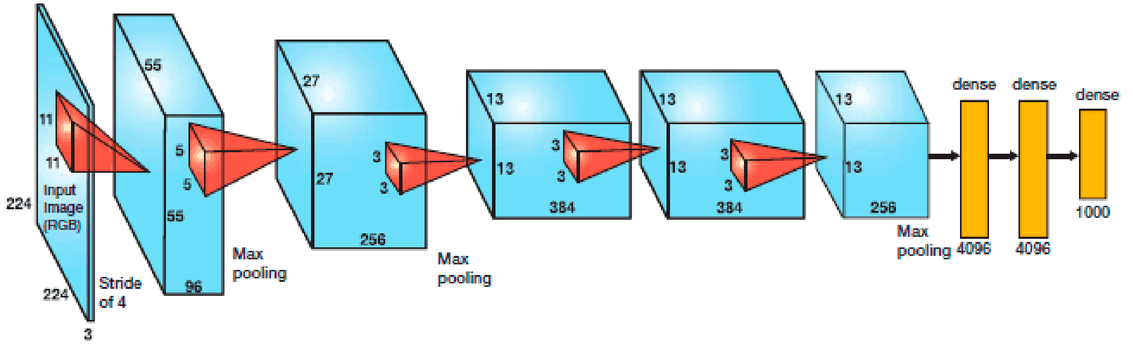
Classic CNN Structure



Convolutional layers

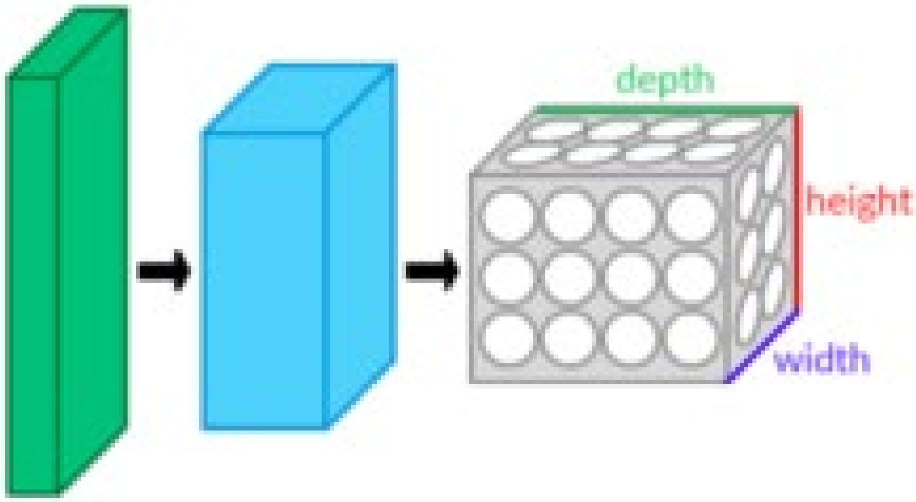
2D convolution with
Activation and
pooling / sub-sampling

Fully connected layers

Matrix multiplication &
activation

- Starts with convolutional layers.
  Each layer does
  - 2D convolution with several kernels
  - Activation (e.g., ReLU)
  - Sub-sampling or pooling
- Finish with fully connected (or dense) layers.
  Each layer does . . .
  - Matrix multiplication
  - Activation

Tensors



- Input and output of each layer is a tensor
  - A multidimensional array
- Examples of tensors
  - Grayscale image: $(Height, Width)$
  - Color image: $(Height, Width, Chan)$
    $Chan \in \{R, G, B\}$
  - Batch of images: $(Sample, Height, Width, Chan)$
- Example: A batch of 100 color images with 256×384 pixels has shape: $(100, 256, 384, 3)$
- The number of dimensions is called the order or rank
  - Note that rank has a different meaning in linear algebra
  - So, we will use order

What Do Convolutional Layers Do?

- Each convolutional layer has:
  - Weight tensor: W size $(K_1, K_2, N_{in}, N_{out})$
  - Bias vector, b: size N_{out}
- Takes input tensor U creates output tensor
- Convolutions performed over space and added over channels

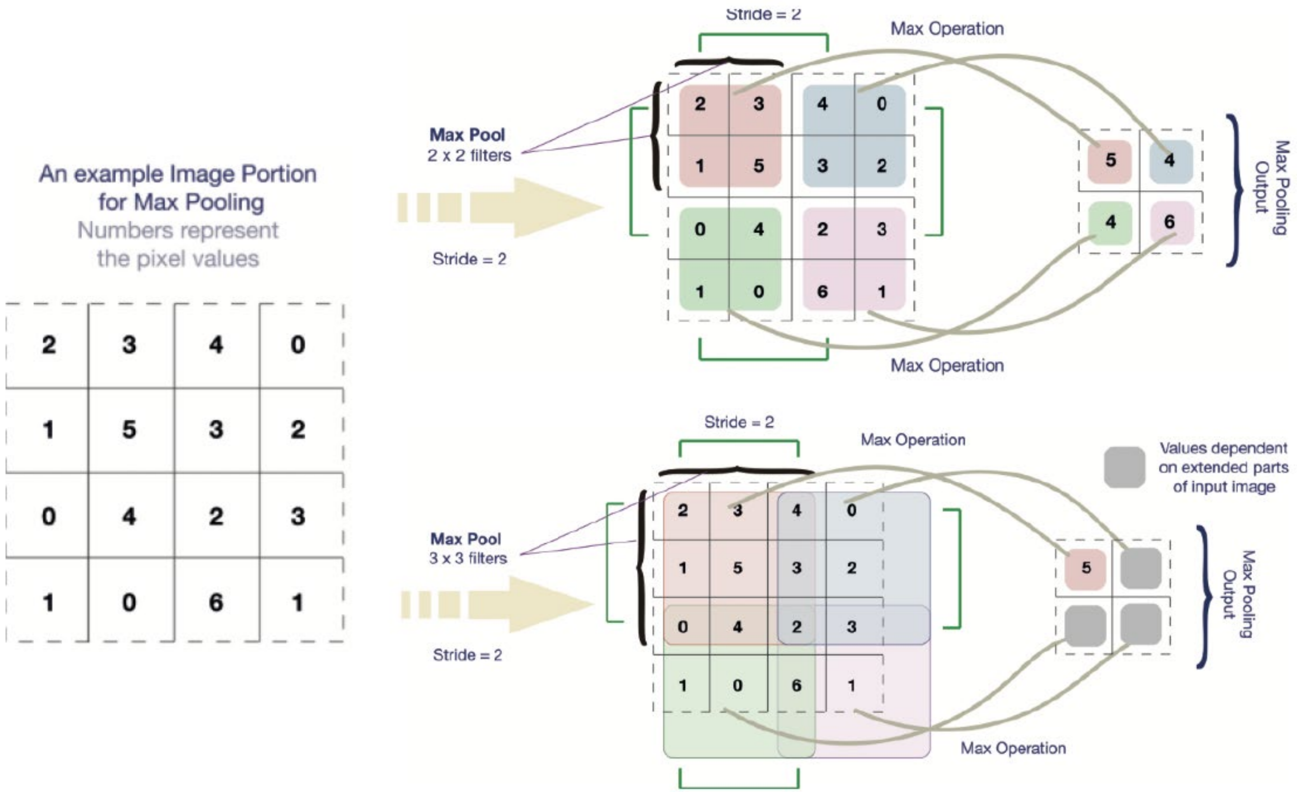
$$Z[i_1, i_2, m] = \sum_{k_1=0}^{K_1-1} \sum_{k_2=0}^{K_2-1} \sum_{n=0}^{N_{in}-1} W[k_1, k_2, n, m] X[i_1 + k_1, i_2 + k_2, n] + b[m]$$

- For each output channel m, input channel n
  - Computes 2D convolution with $W[:,:,n,m]$ (2D filters of size $K_1 \times K_2$)
  - Sums results over n
  - Different 2D filter for each input channel and output channel pair

Subsampling and Pooling

- After convolution and activation, there is often a data-reduction stage
- There are many options here. Some popular ones are . . .
- Subsampling:
  - keep the top-left pixel from every $S \times S$ region, S is called the stride
  - Implemented as part of convolution (no wasted computations!)
  - Called “downsampling” in signal processing
- Max pooling:
  - Keep the largest value in each $K \times K$ region
  - Shift the region by stride S horizontally & vertically
- Average pooling:
  - Keep the largest value in each $K \times K$ region
  - Shift the region by stride S horizontally & vertically
  - Called ”decimation” in signal processing
- The above is performed independently on every channel and batch item

Max Pooling Illustrated



What Dense Layers Do?

- Say that the last convolutional layer produced (after pooling) a tensor of shape (B, N_1, N_2, C)
- Just before the first dense layer, we flatten (i.e., reshape) into matrix $\boldsymbol{U}$
  - Shape is (B, N_{in}), $N_{in} = N_1 N_2 C$
- Then output is performed with matrix multiplication:

$$Z[i,k] = \sum_{j=1}^{N_{in}} W[j,k]\, U[i,j] + b[k], \qquad k = 0, \ldots, N_{out}$$

  - Weights W: shape (N_{in}, N_{out})
  - Bias b: Shape $(N_{out},)$
- Same as the linear stages of the 2-layer neural network from the last unit!

Convolution vs Fully Connected

- Using convolution layers greatly reduces number of parameters
- Ex: Suppose input is $(*, N_1, N_2, N_{in})$ output is $(*, M_1, M_2, N_{out})$
  - Ex: AlexNet 2nd layer $(*, 55, 55,96) \rightarrow (*, 55, 55, 256)$
- Convolutional network with (K_1, K_2) size filters
  - Requires $K_1 K_2 N_{in} N_{out}$ weights and N_{out} biases
  - Example: AlexNet 2nd layer with $K_1 = K_2 = 5$ filters has $6.1(10)^5$ weights and 25 biases
- But, a fully-connected layer with same size inputs and outputs:
  - Would require $N_1 N_2 N_{in} M_1 M_2 N_{out}$ weights and $N_1 N_2 N_{out}$ biases
  - Example: AlexNet 2nd layer would need $2.2(10)^{11}$ weights and $7.7(10)^5$ biases
- Convolutional layers exploit translation invariance
  - Local features are small and could be located