Introduction to Convolutional Neural Network

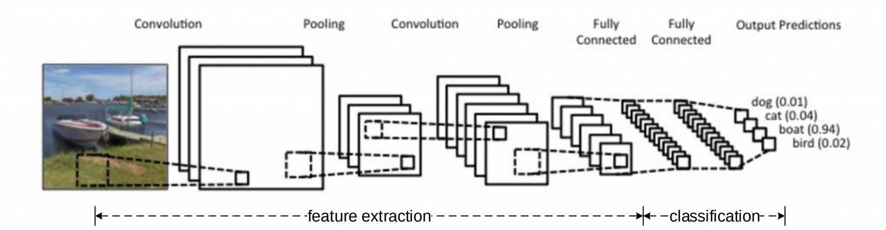
2019 - 2020 - 2021

Ando Ki, Ph.D.
adki@future-ds.com

Table of contents

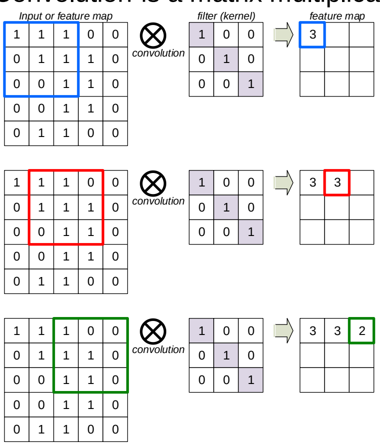
CNN: Convolutional Neural Network

- CNN is a neural network that uses convolution in place of general matrix multiplication in at least one of their layers.
- General form of CNN (Convolutional Neural Network) for image classification
  - Feature extraction
    - Convolution
    - Pooling (sub-sampling)
  - Classification
    - Regression



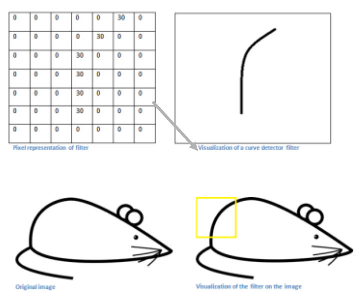
CNN: convolution

- Convolution is a matrix multiplication



When the value is large after convolution, it means there is a feature about it.

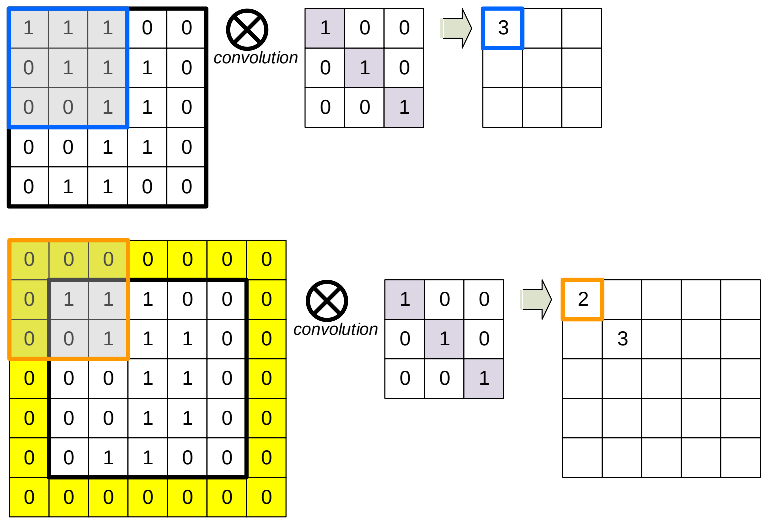
- It can be seen as a feature extractor



https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/

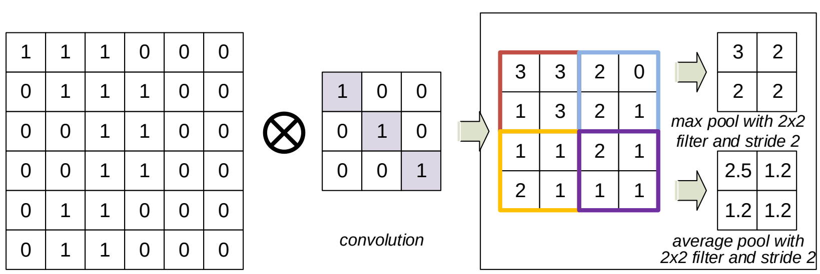
CNN: convolution padding

- No padding
  - **Valid padding**
- Zero padding
  - **Same padding** due to input and out have the same dimensions.



CNN: Pooling

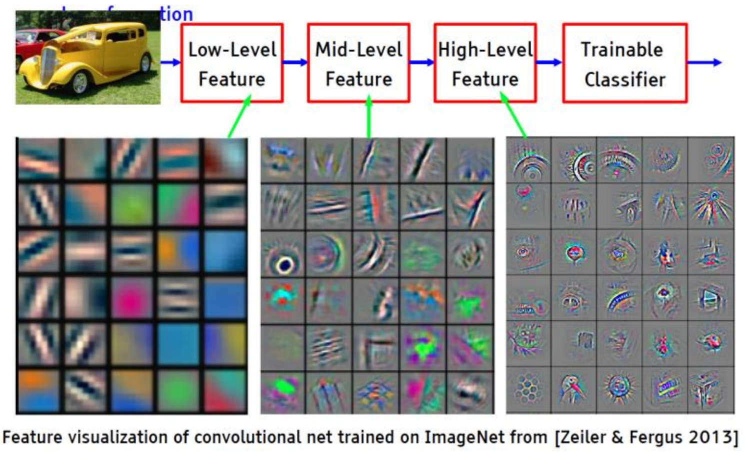
- Pooling, i.e., sub-sampling
  - Max pooling
  - Average pooling



How to choose filters

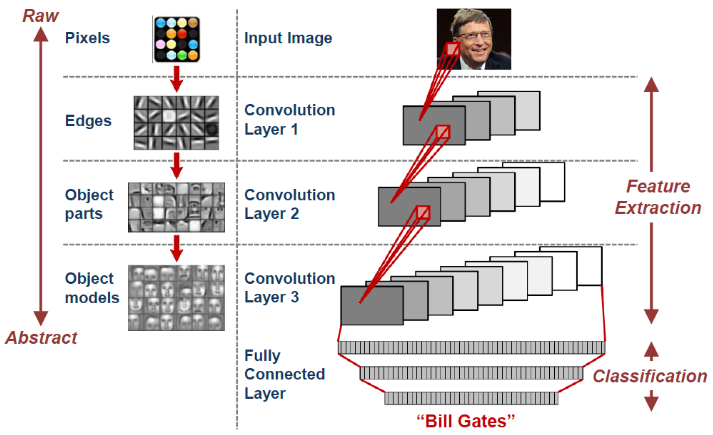
- With CNN/ConvNet the goal is to learn the filters; you don't actually design these filters (or kernels). They will be learned during training as long as the training converges.
- Initializing the these filter parameters with good defaults before starting the training is key to convergence especially in very deep networks.
- Convolution filters can be initialized in one of the following ways.
  - 1. Randomly assigning weights for the different filters.
  - 2. Handcrafting the weights of the different filters to detect specific features during convolution.
  - 3. Learning filter weights using unsupervised training schemes.

Deep learning: Learning Hierarchical Representations

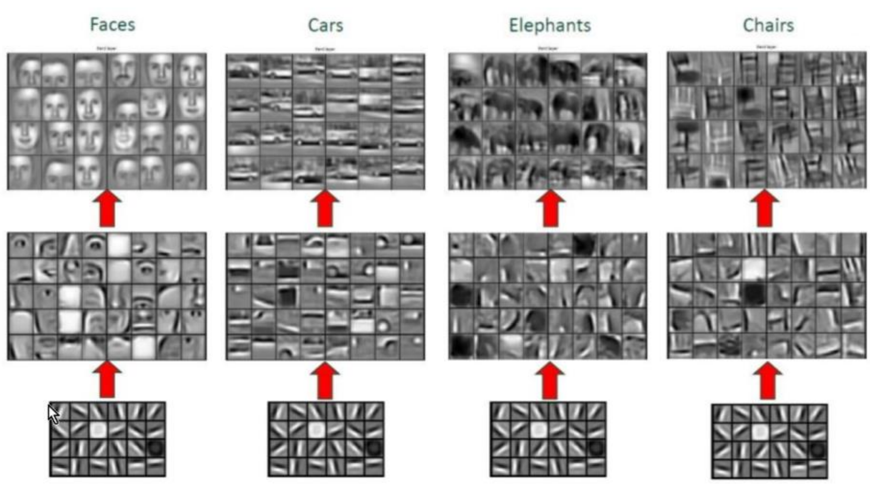


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

CNN abstraction

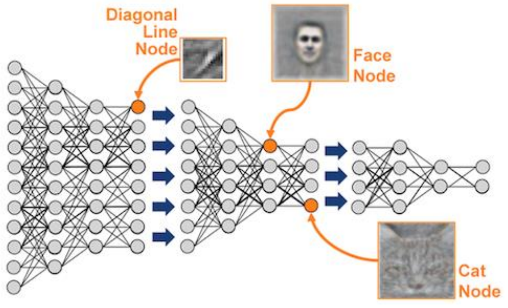


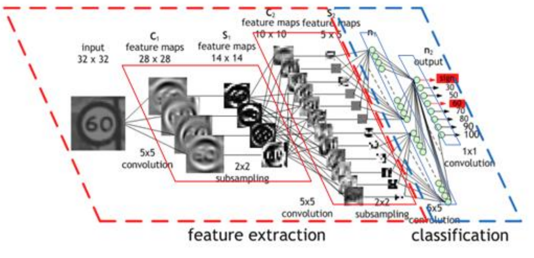
CNN abstraction



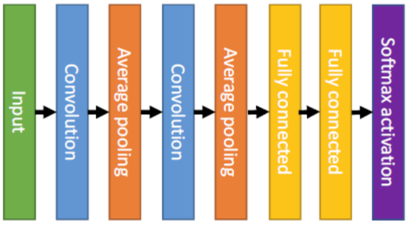
Convolutional deep belief networks for scalable unsupervised learning of hierarchical representation", Lee et al., 2012

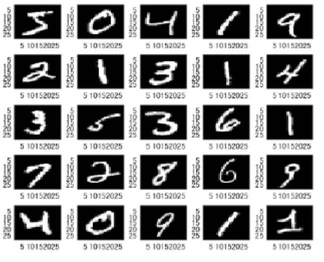
CNN examples

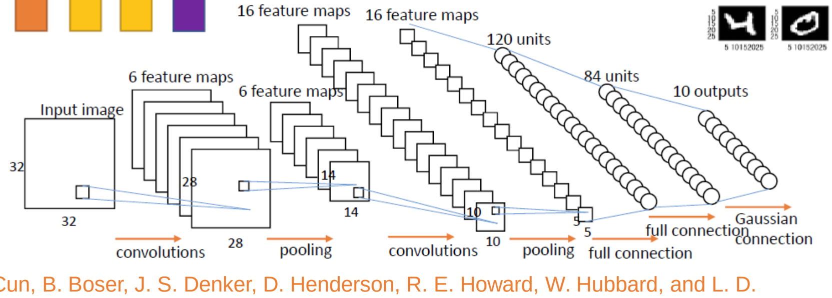




CNN examples: LeNet-5 (1989)







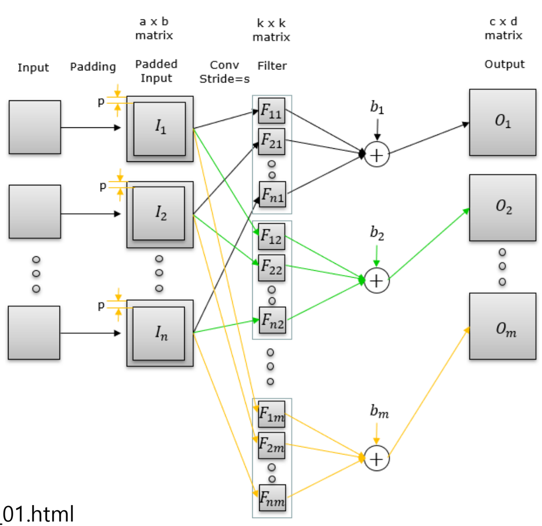
Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. NIPS 1989.

Table of contents

Convolution 2D

torch.nn.Conv2d(in_channels=n,
    out_channels=m, kernel_size=k,
    stride=s, padding=p)

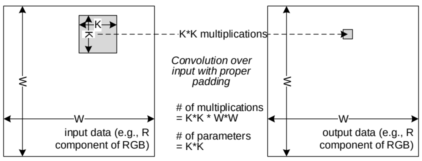
Note that there are nxm filters (kernels)



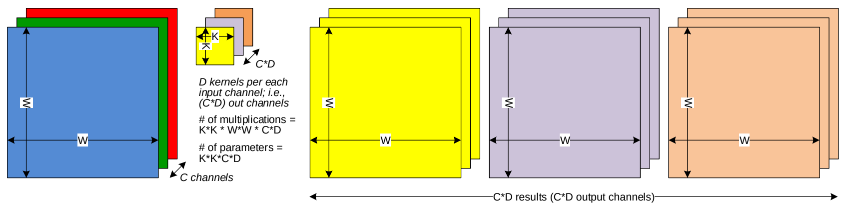
http://sharetechnote.com/html/Python_PyTorch_nn_conv2D_01.html

Standard convolution

- Single channel with single kernel

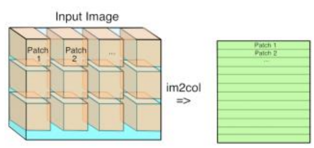


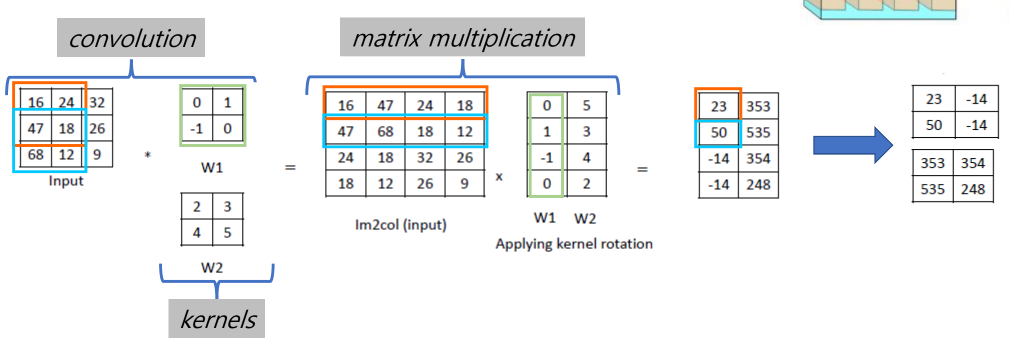
- Multiple channels and kernels



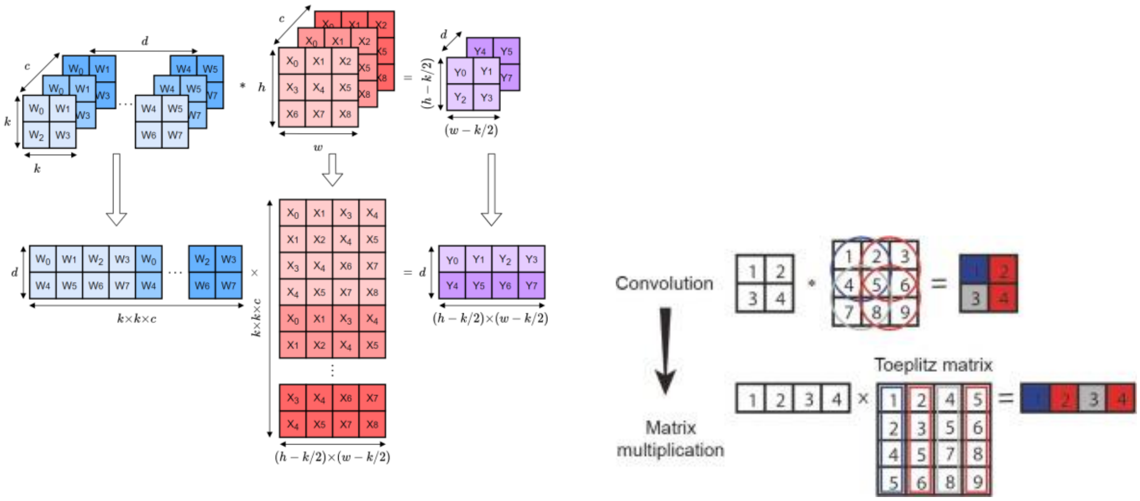
Convolution by matrix multiplication

- 1-channel 2D convolution example: 3x3 input with 2x2 two kernel case
  - ▶ Refer to 'im2col'



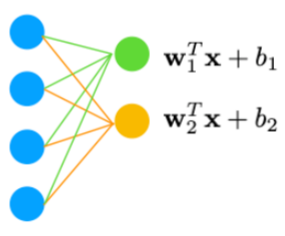


im2col



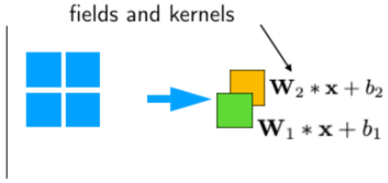
Fully connected layer by convolution (1/2)

- Convolution with kernels equal to the input size



Fully connected layer

remember, these also involve dot products between the receptive fields and kernels



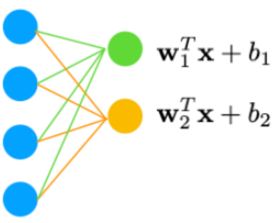
where $\mathbf{W}_1 = \begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{1,3} & w_{1,4} \end{bmatrix}$

$\mathbf{W}_2 = \begin{bmatrix} w_{2,1} & w_{2,2} \\ w_{2,3} & w_{2,4} \end{bmatrix}$

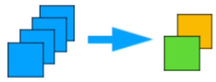
```
conv = torch.nn.Conv2d(in_channels=1,
               out_channels=2,

kernel_size=inputs.squeeze(dim=(0)).squeeze(dim=(0)).size())
print(conv.weight.size())
print(conv.bias.size())
```

Fully connected layer by convolution (2/2)

- Convolution with 1x1 kernels



Fully connected layer

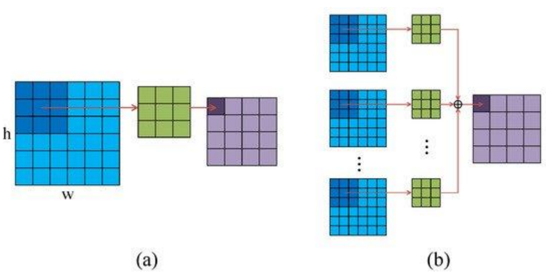


Or, we can concatenate the inputs into 1x1 images with 4 channels and then use 2 kernels (remember, each kernel then also has 4 channels)

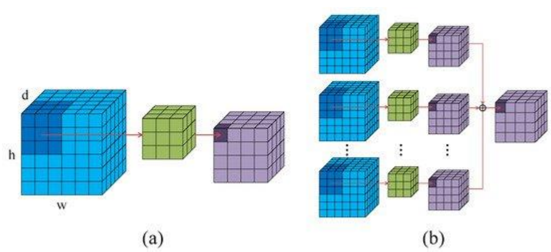
```
conv = torch.nn.Conv2d(in_channels=4,
                       out_channels=2,
                       kernel_size=(1, 1))

conv.weight.data = weights.view(2, 4, 1, 1)
conv.bias.data = bias
torch.relu(conv(inputs.view(1, 4, 1, 1)))
```


Convolution for multi-channel

- 2D convolution
  - single and multi-channel

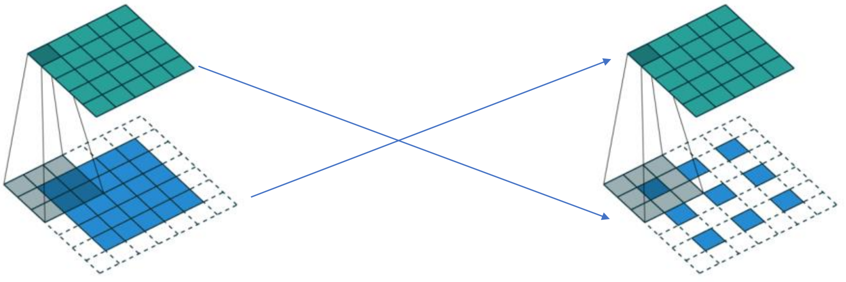


- 3D convolution
  - single and multi-channel



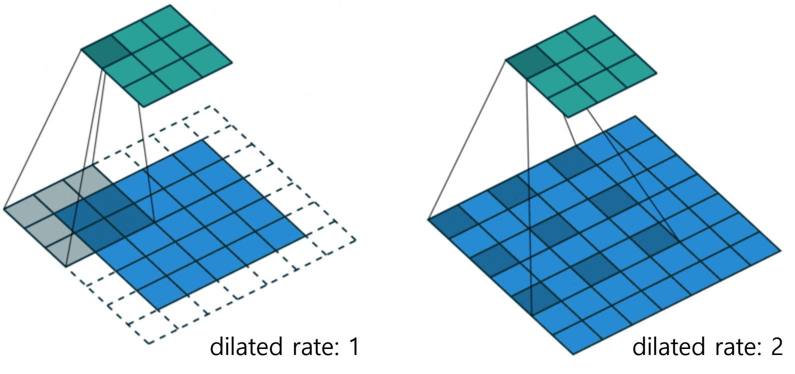
Convolution and deconvolution

- Standard convolution (discrete convolution)
  - to extract feature map
- Standard deconvolution
  - known as **transposed convolution**
  - to reconstruct original image
  - a reverse operation of convolution



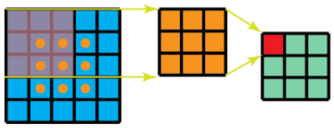
Dilated convolution (atrous convolutions)

- Similar with deconvolution used in real-time segmentation
- smaller kernel for wider view
- not reverse operation (i.e, not reconstruction of original image)

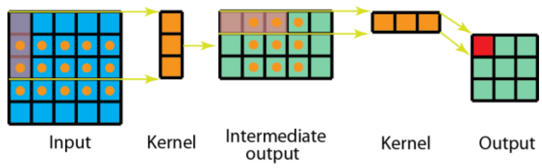


Separable convolution: spatially separable

- standard convolution
  - multiplications
    - K*K * W*W



- spatially separable convolution
  - multiplications
    - K*W*W + K*W*W
      - 2/K ratio comparing to standard convolution



- kernel divided

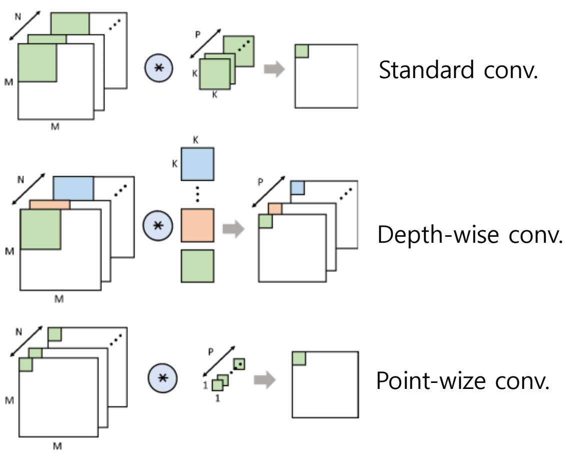
$$\begin{bmatrix}-1 & 0 & 1\\-2 & 0 & 2\\-1 & 0 & 1\end{bmatrix} = \begin{bmatrix}1\\2\\1\end{bmatrix} \times \begin{bmatrix}-1 & 0 & 1\end{bmatrix}$$

$$\begin{bmatrix}3 & 6 & 9\\4 & 8 & 12\\5 & 10 & 15\end{bmatrix} = \begin{bmatrix}3\\4\\5\end{bmatrix} \times \begin{bmatrix}1 & 2 & 3\end{bmatrix}$$

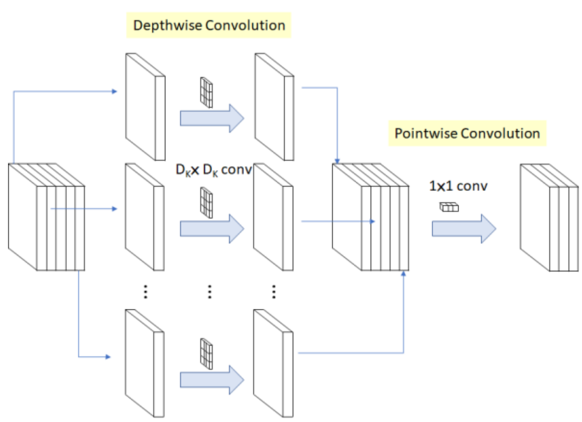
Although spatially separable convolutions save cost, it is rarely used in deep learning. One of the main reason is that _not all kernels can be divided into two, smaller kernels_..

Separable convolution: depthwise separable

- Standard convolution and depthwise separable (no channel-wise conv)



- MobileNet case



Separable convolution: depthwise separable

- Standard convolution
  - uses kernels of a number of output channels
- Depth wise separable
  - Depthwise convolution: filtering stage
    - uses kernels of a number of input channels
  - Pointwise convolution: combining state
    - uses kernels of a number of output channels