ELEG5491: Introduction to Deep Learning

Convolutional Neural Networks

Prof. LI Hongsheng

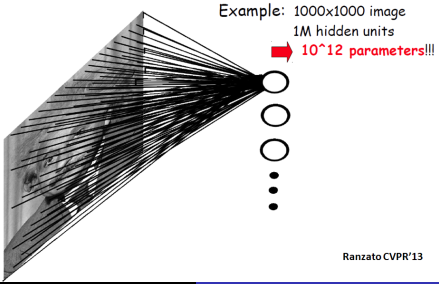
e-mail: hsli@ee.cuhk.edu.hk
Department of Electronic Engineering
The Chinese University of Hong Kong

Feb. 2023

Outline

Convolutional Neural Network

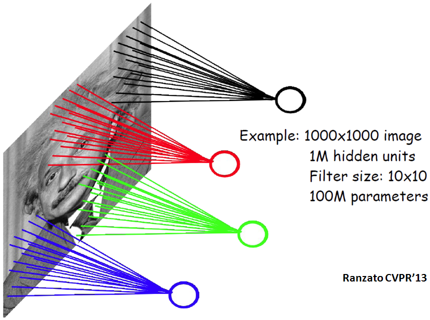
- There are data of grid-like structures, for instance,
  - 1D grid: sequential data
  - 2D grid: natural images
  - 3D grid: video, 3D image volumes
- Problem of fully-connected neural networks on handling such image data
  - The number of input values are generally quite large
  - The number of weights grows substantially as the size of the input images
  - Pixels in distance are less correlated



Ranzato CVPR'13

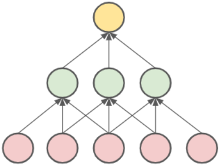
A locally connected neural networks

- Sparse connectivity: a hidden unit is only connected to a local patch (weights connected to the patch are called filter or kernel)
- It is inspired by biological systems, where a cell is sensitive to a small sub-region of the input space, called a receptive field. Many cells are tiled to cover the entire visual field



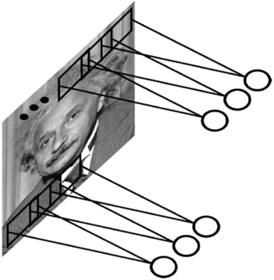
Locally connected neural networks

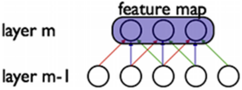
- The learned filter can be considered as a spatially local pattern to capture local information
- A hidden neuron (unit) at a higher layer has a larger receptive field in the input
- Stacking many such layers leads to “filters” (not anymore linear) which become increasingly “global”, independent of different locations



Shared weights at different spatial locations

- In addition to local connectivity, we here also require the weights to be shared at all spatial locations
- Hidden nodes at different locations share the same weights. It greatly reduces the number of parameters to learn
- Such a property is called *translation invariance*: captures statistics in local patches and they are independent of locations
  - For example, similar image edges may appear at different locations





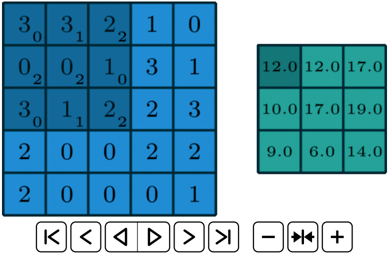
Weights with the same color have identical values

2D convolution in CNN

- Given an input feature map (or image) of spatial size $P \times Q$, the convolution with a single-channel 3×3 kernel operates as follows

0	1	2
2	2	0
0	1	2

Kernel

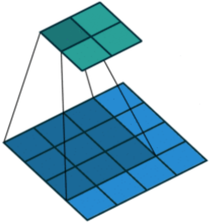


- 2D convolution in 2D Convolutonal Neural Networks (CNN) with single-channel input $x \in \mathbb{R}^{P \times Q}$ and single-channel kernel $W \in \mathbb{R}^{s \times t}$, can be formulated as

$$y(i,j) = \sum_{u=-\lfloor s/2 \rfloor}^{\lfloor s/2 \rfloor} \sum_{v=-\lfloor t/2 \rfloor}^{\lfloor t/2 \rfloor} w(u,v) \cdot x(i+u, j+v) + b$$

2D convolution in CNN

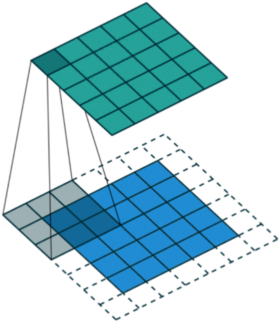
- Given an input feature map (or image) $x \in \mathbb{R}^{P \times Q}$, the parameters to be learned during training is the kernel W and the bias parameter b
- The kernel size s and t are generally chosen as the odd numbers
- Note that in conventional signal processing theory, the above operation is called "correlation" instead of "convolution"
- The results of convolution in CNNs are called "*feature maps*"
- Without extra procedures, the resulting feature maps would be smaller than the input feature maps



Padding

- The input feature map $x \in \mathbb{R}^{P \times Q}$ can be padded with zeros on four sides to ensure the output feature map has the same spatial size $P \times Q$
- The padding sizes on the four sides are usually chosen as $\lfloor s/2 \rfloor$ and $\lfloor t/2 \rfloor$

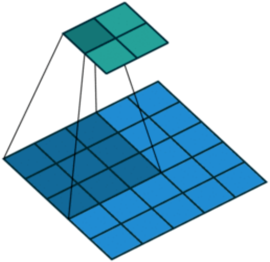
The padding size for 3×3 kernels are 1 for the four sides



Stride

- The idea of the stride is to skip some of the slide locations of the kernel
- A stride of 1 means to pick slides a pixel apart, so basically every single slide, acting as a standard convolution
- A stride of 2 means picking slides 2 pixels apart, skipping every other slide in the process, downsizing by roughly a factor of 2, a stride of 3 means skipping every 2 slides, downsizing roughly by factor 3, and so on.

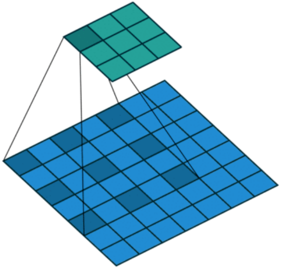
A stride 2 convolution



Dilation

- Image understanding tasks generally requires to use image contextual information at each spatial location
- A $k \times k$ kernel can cover image patches of size $k \times k$, what if we want to use the same number of parameters to cover larger image regions
- Dilated convolution with coefficient l. Each pair of neighboring kernel weights are l pixels away

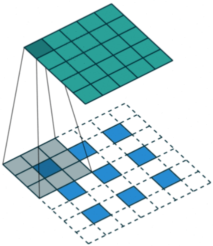
Dilated convolution ($l = 2$)



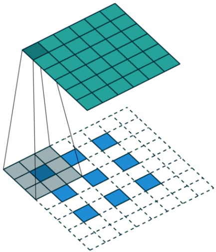
Deconvolution (transposed convolution)

- The above operations maintain or decrease spatial sizes of the input feature maps
- In some scenarios, one would like to increase the spatial size of feature maps
- Deconvolution (some researchers argue that it should be named as *transposed convolution*) is one common option to do so
- Another even simpler operation but sometimes more effective operation is *bilinear interpolation*

1×1 padding, stride 2, transposed

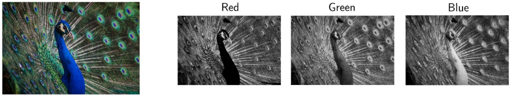


1×2 padding, stride 2, transposed (odd)



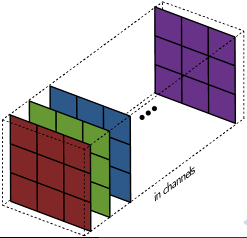
The multi-channel version

- The above equation handles input feature maps (images) with only 1 channel
- However, there exist images of multiple channels. For instance, the colorful images consist of R, G, B channels



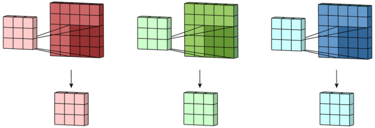
- To handle an input feature map $x \in \mathbb{R}^{P \times Q \times C}$, the filter should be extended to $w \in \mathbb{R}^{k \times k \times C}$

It can be viewed as a collection of 2D kernels

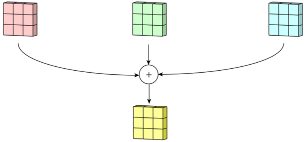


The multi-channel version

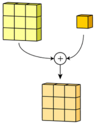
- Each of the kernels of the filter "slides" over their respective input channels, producing a processed version of each channel



- Each of the per-channel processed versions are then summed together to form one output channel

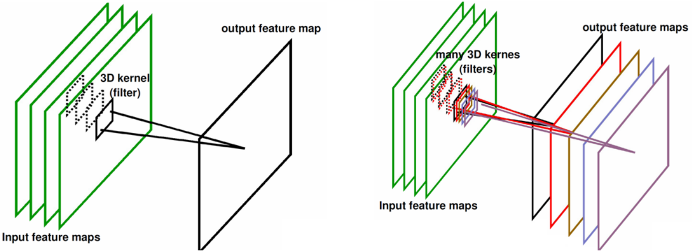


- The bias gets added to the output channel so far to produce the final output channel



The multi-channel version

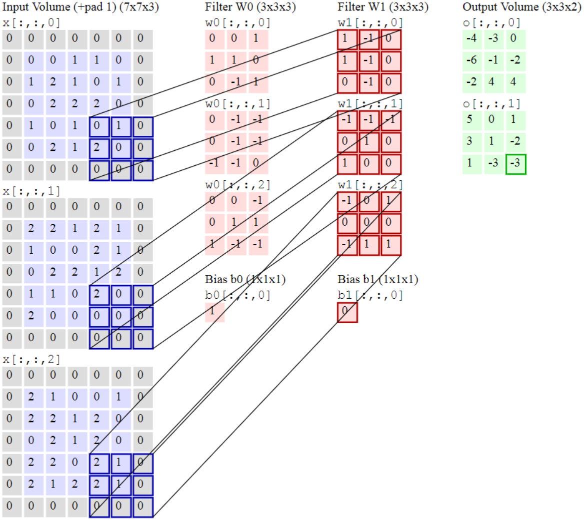
- Therefore, given a multi-channel feature map $x \in \mathbb{R}^{P \times Q \times C}$ as input, each filter $w \in \mathbb{R}^{k \times k \times C}$ generates one single-channel feature map
- For the same input feature map, there can be multiple filters operating on the input and each of them generates one channel of feature map
- The feature map can be concatenated along the channel dimension to generate a multi-channel
- D filters lead to an D-dimensional output feature map $y \in \mathbb{R}^{P \times Q \times D}$



Ranzato CVPR'13

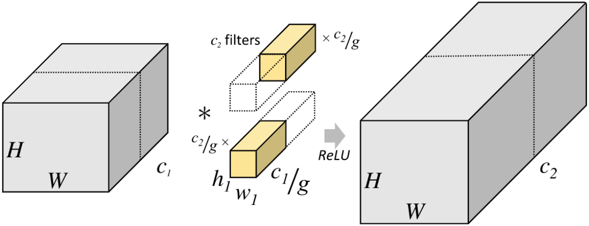
The multi-channel version

- Here is another illustration from CS231n course of Stanford



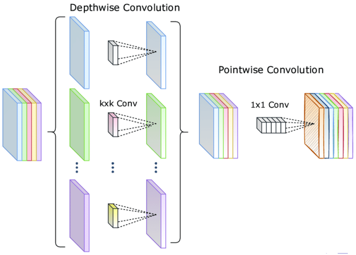
Grouped convolution

- The input feature maps can be divided into multiple groups along the channel dimension
- Convolution filters are applied to only one of the groups
- The different feature groups generally have different convolution filters
- The below example shows that a grouped convolution with input feature dimension c_1, output feature dimension c_2, and 2 feature groups



Depthwise convolution followed pointwise convolution

- If we separate a c-channel feature map into c groups, i.e., each channel as a separate group, grouped convolution on such separate single-channel feature maps are named depthwise convolution
- Depthwise convolution (e.g., 3×3, 5×5, etc.) is lightweight compared to ordinary convolution. But the resulting output feature maps do not have contain any cross-channel information, which is unfavourable for various learning problems
- Depthwise convolution is therefore generally followed by a 1×1 convolution (also named as pointwise convolution) to achieve cross-channel information fusion and control the output channel number

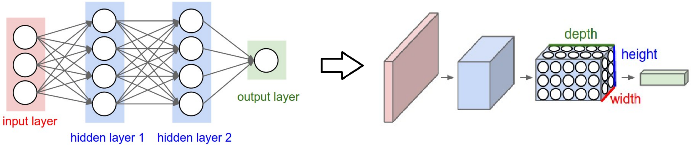


Convolution in 2D CNNs

- If the input feature map x has the shape $\mathbb{R}^{P \times Q \times C_{in}}$ and output feature map y has the shape $\mathbb{R}^{P \times Q \times C_{out}}$
- If we would like to use a filter of spatial size is $k \times k$, the filter should have C_{out} kernels of $\mathbb{R}^{k \times k \times C_{in}}$
- Combining all operations, if the input feature map is of size $L_{in} = [P, Q]$, the output feature map size is

$$L_{out} = \left\lfloor \frac{L_{in} + 2 \times \text{ padding } - \text{ dilation } \times (\text{kernel\_size} - 1) - 1}{\text{stride}} + 1 \right\rfloor$$

- Extension from fully-connected neural networks to Convolutional Neural Networks (CNN)



Forward and backward computation of convolution

- We illustrate how to calculate the gradients of with simple 1D convolution
- Forward input: $x = [x_1, x_2, \cdots, x_7]$; forward output: $y = [y_1, y_3, \cdots, y_7]$
- Forward computation (we use $w(i)$ and w_i interchangeably):

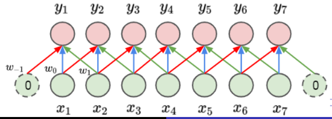
$$y_i = \sum_{k=-1}^{1} w_k \cdot x_{i+k} + b \quad \Leftrightarrow \quad y = w * x + b \quad \text{(sometimes just } y = w * x\text{)}$$

- Backward input: $\frac{\partial J}{\partial y_i}$ for $i = 1, 3, \ldots, 7$; backward output:

$$\frac{\partial J}{\partial x_i} = \sum_j \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial x_i} = \sum_{k=-1}^{1} \frac{\partial J}{\partial y_{i+k}}\frac{\partial y_{i+k}}{\partial x_i} = \sum_{k=-1}^{1} \frac{\partial J}{\partial y_{i+k}}\frac{\partial y_{i+k}}{\partial x_i} = \sum_{k=-1}^{1} w_{-k}\frac{\partial J}{\partial y_{i+k}}$$

$$\frac{\partial J}{\partial x} = \mathrm{rot}_{180^\circ}(w) * \frac{\partial J}{\partial y} \quad \text{(`}*\text{' represents convolution),} \quad \frac{\partial J}{\partial w_k} = \sum_{\text{all } i} \frac{\partial J}{\partial y}(i) \cdot x(i+k)$$

Arrows of the same color denote the same convolution weights



Backward computation of convolution

- Backward computation of learnable parameters w and b

$$\frac{\partial J}{\partial w_k} = \sum_{i=1}^{7} \frac{\partial J}{\partial y_i} \cdot \frac{\partial y_i}{\partial w_k} = \sum_{i=1}^{7} \frac{\partial J}{\partial y_i} \cdot x_{i+k}$$

$$\frac{\partial J}{\partial b} = \sum_{i=1}^{7} \frac{\partial J}{\partial y_i} \cdot \frac{\partial y_i}{\partial b} = \sum_{i=1}^{7} \frac{\partial J}{\partial y_i}$$

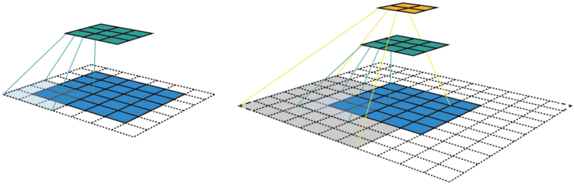
- The calculation can be generalized to 2D convolution
  - Forward input: $x \in \mathbb{R}^{P\times Q}$, parameters $w \in \mathbb{R}^{s\times t}$; forward output: $y = w * x + b$ ('*' denotes convolution)
  - Backward input: $\frac{\partial J}{\partial y} \in \mathbb{R}^{P\times Q}$; backward output:

$$\frac{\partial J}{\partial x} = \mathrm{rot}_{180^\circ}(w) * \frac{\partial J}{\partial y}$$

$$\frac{\partial J}{\partial W}(u,v) = \sum_i \sum_j \frac{\partial J}{\partial y}(i,j) \cdot x(i+u, j+v) \qquad \frac{\partial J}{\partial b} = \sum_i \sum_j \frac{\partial J}{\partial y}(i,j)$$

Receptive field of Convolution Neural Network

- A convolutional neural network can be stacked for multiple times
- Max pooling and strided convolution are constantly used to quickly decrease spatial dimension of feature maps
- The **receptive field** of a feature can be briefly defined as the region in the input image pixel space that the feature is calculated from



Two consecutive convolution with kernel size $k = 3 \times 3$, padding size $p = 1 \times 1$, stride $s = 2 \times 2$. (Right)

Receptive field of Convolution Neural Network

- To calculate the receptive field at each layer, we define the following notation
  - r_{in}: the current receptive field
  - j (jump): the distance between two adjacent features
  - k, p, s: kernel size, padding size, and stride size

$$\begin{aligned} j_{\text{out}} &= j_{\text{in}} \times s \\ r_{\text{out}} &= r_{\text{in}} + (k-1) \times j_{\text{in}} \end{aligned}$$

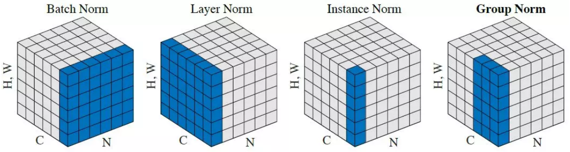
- For the very first input to a network, we always have $r_0 = 1$ and $j_0 = 1$
- Given the previous example, we have

$$\begin{aligned} r_1 &= r_0 + (k-1) \times j_0 = 1 + (3-1) \times 1 = 3, \quad j_1 = j_0 \times 2 = 2 \\ r_2 &= r_1 + (k-1) \times j_1 = 3 + 2 \times 2 = 7, \quad j_2 = j_1 \times 2 = 4 \end{aligned}$$

- One should ensure that the receptive field is large enough for each features for different tasks

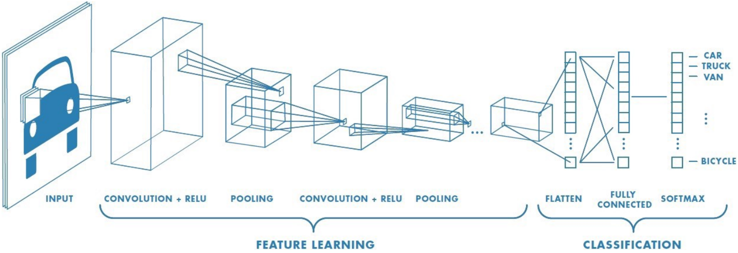
Non-linearity function, 2D dropout and 2D Batch Normalization layers

- Previously introduced non-linearity function layers can all be adopted, including sigmoid function, softmax function (along chosen dimension), ReLU layer, PReLU layer
- 2D Dropout layer: Randomly zeros out entire some channels of every instances. Each channel will be zeroed out independently on every forward call with probability p using samples from a Bernoulli distribution
- 2D Batch Normalization: feature vectors of length C at each pixel location of the 2D feature map $P \times Q \times C$ is treated as a sample to calculate the sample mean and sample standard deviation for normalization



CNN with max pooling layers

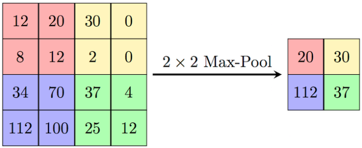
- Convolution operation alone with padding will result in the feature maps of the same spatial sizes
- However, for image classification, we would like to summarize the input image into a 1D feature vector and then use a final linear classifier to classify it into pre-defined classes
- We need max pooling layers to gradually decrease the spatial size of the feature maps and eventually encode input image into 1D feature vectors



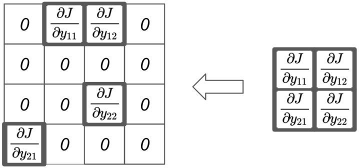
Pooling layers

- A 2×2 max pooling with stride 2 is the mostly common choice to decrease spatial sizes
- Each channel of the input feature map is max pooled independently
- Input feature map size: $P \times Q \times C$; output feature map size: $P/2 \times Q/2 \times C$

Forward computation



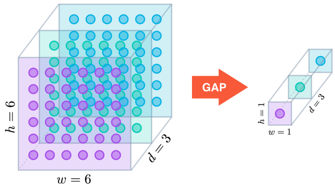
Backward computation



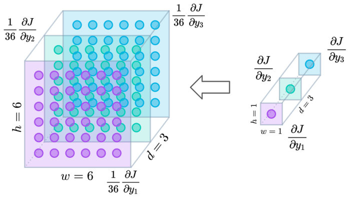
Global average pooling layer

- Global average pooling is commonly utilized as the last layer to convert 2D feature maps to 1D feature vectors
- Feature map of each channel is independently averaged to obtain a 1D feature vector

Forward computation

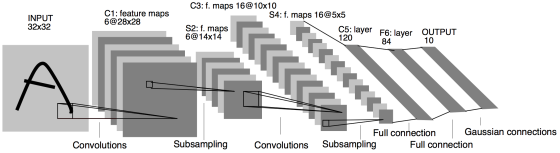


Backward computation



LeNet-5 for hand written digit recognition

- Instead of vectorizing (flattening) the input digit images, LeNet-5 propose to use CNN for recognizing hand written digits
- It consits of 3 convolutional layers (C1, C3 and C5), 2 sub-sampling (pooling) layers (S2 and S4), and 1 fully connected layer (F6), that are followed by the output layer
- Convolutional layers use 5×5 convolutions with stride 1
- Sub-sampling layers are 2×2 average pooling layers with stride 2
- Tanh functions are utilized as non-linearity functions

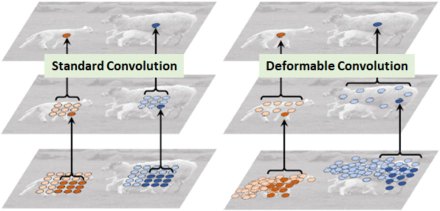


Deformable Convolution [Dai et al. ICCV'17]

- The above mentioned **regular convolution** operates on the input 2D feature maps of grid structure with a static kernel
- We reformulate the regular convolution as

$$y\left(p_0\right)=\sum_{p_n \in \mathcal{R}} w\left(p_n\right) \cdot x\left(p_0+p_n\right)$$

- $p_0=(x_0, y_0) \in \mathbb{R}^2$ the coordinates of the pixel of interest
- $\mathcal{R}$ denotes the local neighborhood defined by a kernel, e.g., 3×3 and 5×5 local grid centered at each input pixel
- $p_n=(x_n, y_n) \in \mathbb{R}^2$ is local coordinates of kernel weights, e.g., $(-1,1),(-1,0),(-1,1), \ldots,(1,-1),(1,0),(1,1)$ for 3×3 kernels



Deformable Convolution [Dai et al.]

- The deformable convolution is operated first on the regular grid with shared kernel weights, but each of which shifted by a learnable offset $\Delta p_n = (\Delta x_n, \Delta y_n) \in \mathbb{R}^2$
- The first convolution outputs a feature map of dimension $2N$ if information from N spatial locations needs to be aggregated for each pixel
- The predicted shift Δp_n is added to the coordinates of the retrieved features $p_0 + p_n$. Bilinear sampling will be used if the shifted coordinates $p_0 + p_n + \Delta p_n$ are not integers

$$y(p_0) = \sum_{p_n \in \mathcal{R}} w(p_n) \cdot x(p_0 + p_n + \Delta p_n)$$

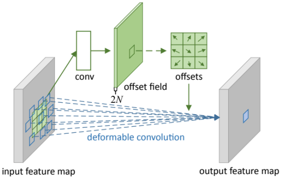
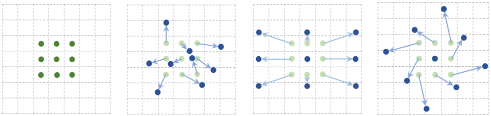


Illustration of deformable convolution



Examples of $p_n \in \mathcal{R}$ and $p_n + \Delta p_n$

Examples of Deformable Convolution

- Deformable convolution used in object detection. Each image triplet shows the sampling locations ($9 \times 3 = 729$ red points in each image) in three levels of 3×3 deformable filters for 3 points of interest (green)

Examples of Deformable Convolution

- Deformable convolution used in object detection. Each image triplet shows the sampling locations ($9 \times 3 = 729$ red points in each image) in three levels of 3×3 deformable filters for 3 points of interest (green)

Deformable Convolution v2 [Zhu et al.]

- Deformable convolution v1 only shifts kernel weight locations but didn't change the kernel weights for each pixel of interest p_0
- Deformable convolution v2 moves one step further and modulate each kernel weight $w(p_n)$ with a coefficient Δm_k

$$y(p_0) = \sum_{p_n \in \mathcal{R}} w(p_n) \cdot x\left(p_0 + p_n + \Delta p_n\right) \cdot \Delta m_n$$

- $\Delta m_k \in [0, 1]$ modulation scalar for the n-th location
- Both Δp_k and Δm_k are obtained via a convolution layer outputting feature maps of $3N$ channels
- The first $2N$ channels records the predicted offsets Δp_k. The remaining N channels are fed to a sigmoid layer to obtain modulation scalars Δm_k
- In general, slightly more computation than deformable convolution v1 but also slightly higher performance

Partial Convolution (sparse convolution)

- For the task of image inpainting, it aims to fill up the contents in the holes of an input image. However, there exist invalid pixels that hinder the regular convolution
- **Partial convolution** is formulated as

$$x' = \begin{cases} \frac{\mathrm{sum}(\mathbf{1})}{\mathrm{sum}(m)} \sum_{\text{all } i} m_i w_i x_i + b, & \text{if } \mathrm{sum}(m) > 0 \\ 0, & \text{otherwise} \end{cases}$$

- x and m denote feature (or pixel) values for the current convolution window and the corresponding binary mask, respectively. $\mathbf{1}$ has the same shape as m but with all one's. x' is the output feature value
- The scaling factor $\mathrm{sum}(\mathbf{1})/\mathrm{sum}(m)$ applies appropriate scaling to adjust the varying amount of valid (unmasked) inputs

Partial Convolution (sparse convolution)

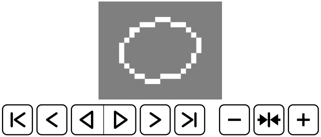
- After each partial convolution, we update the binary mask as well

$$m' = \begin{cases} 1, & \text{if } \mathrm{sum}(M) > 0 \\ 0, & \text{otherwise} \end{cases}$$

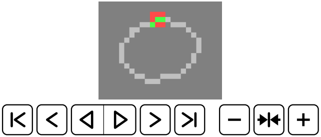
- If the convolution was able to condition its output on at least one valid input value, then we mark that location to be valid
- The sparse convolution can be stacked for multiple layers as regular convolution does

Submanifold Sparse Convolution

- The regular convolution and above mentioned sparse convolution gradually increase the number of valid pixels (features)
- With regular 3×3 convolutions, the set of valid (green, active, non-zero, etc.) sites grows rapidly



- With **submanifold sparse convolution**, the set of valid (green, active, non-zero, etc.) is unchanged. Non-valid sites (red) have no computational overhead



Submanifold Sparse Convolution

- The submanifold sparse convolution has exactly the same formula as partial convolution
- The only difference is that submanifold convolution is only performed at active sites