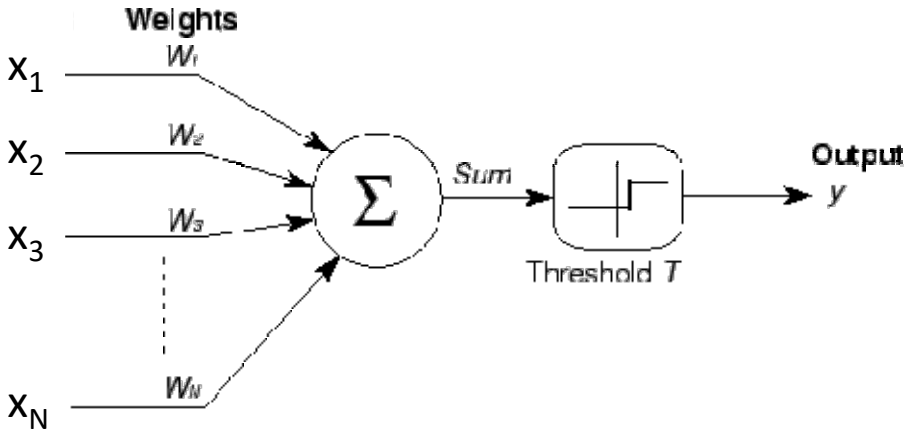
What do the neurons capture?



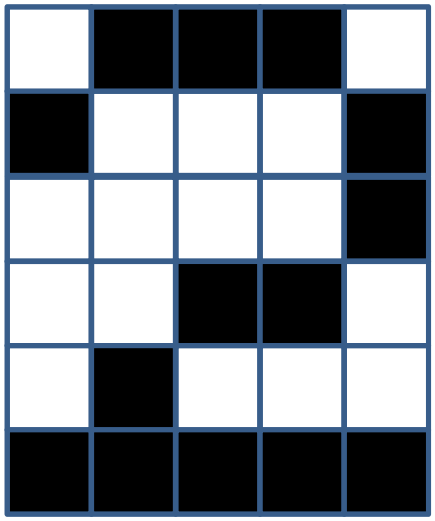
$$y = \begin{cases} 1 \; if \; \sum_i w_i x_i \geq T \\ 0 \; else \end{cases}$$

$$y = \begin{cases} 1 \; if \; \mathbf{x}^T \mathbf{w} \geq T \\ 0 \; else \end{cases}$$

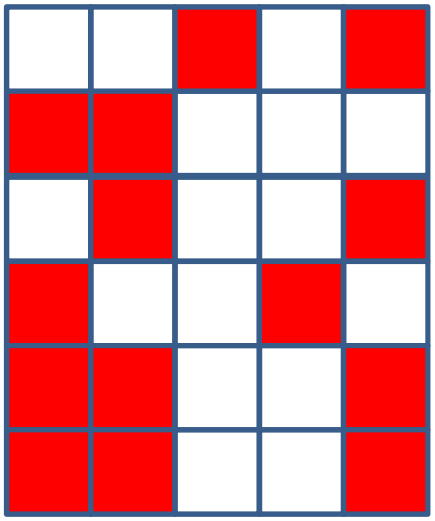
- What do the *weights* tell us?
  - Using example of threshold activation
- The perceptron "fires" if the **correlation** between the weights and the inputs exceeds a threshold
  - **The perceptron fires if the input pattern looks like pattern of weights**

The weights as a correlation filter

W

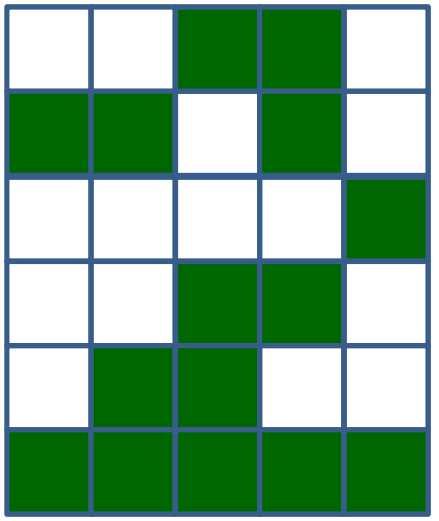


X



Correlation = 0.57

X

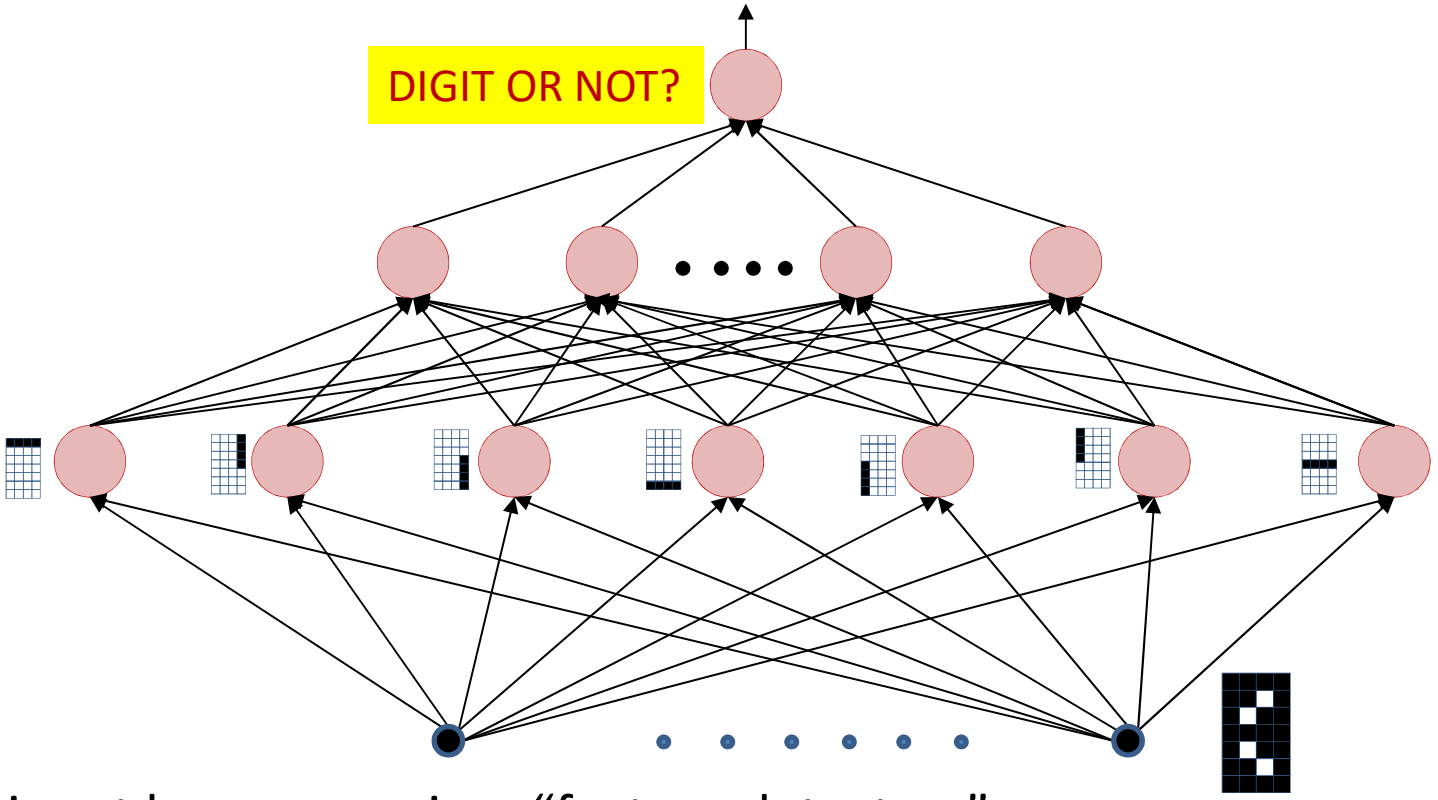


Correlation = 0.82

$$y = \begin{cases} 1 \; if \sum_i w_i x_i \geq T \\ 0 \; else \end{cases}$$

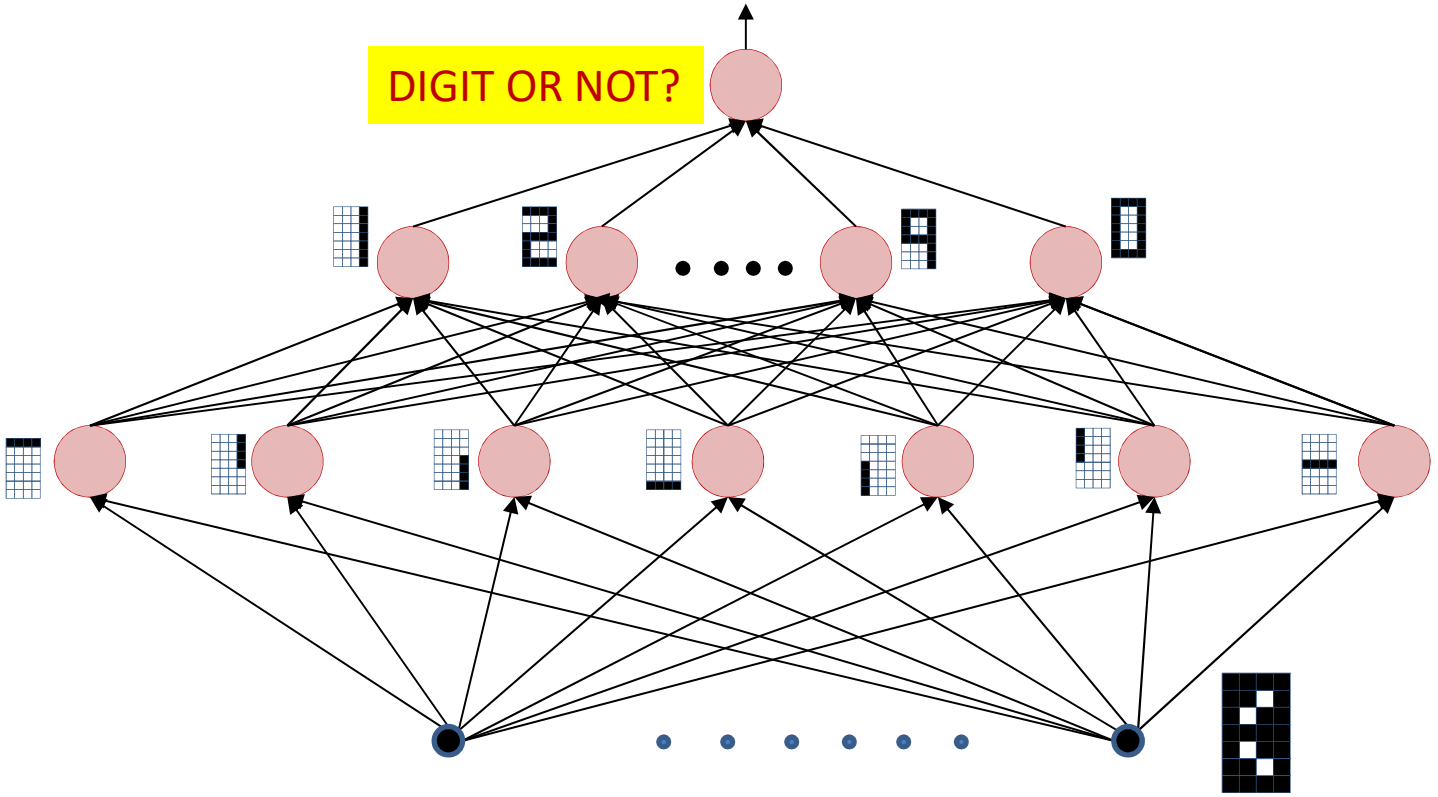
- The green pattern looks more like the weights pattern (black) than the red pattern
  - The green pattern is more *correlated* with the weights

The MLP as a function over feature detectors



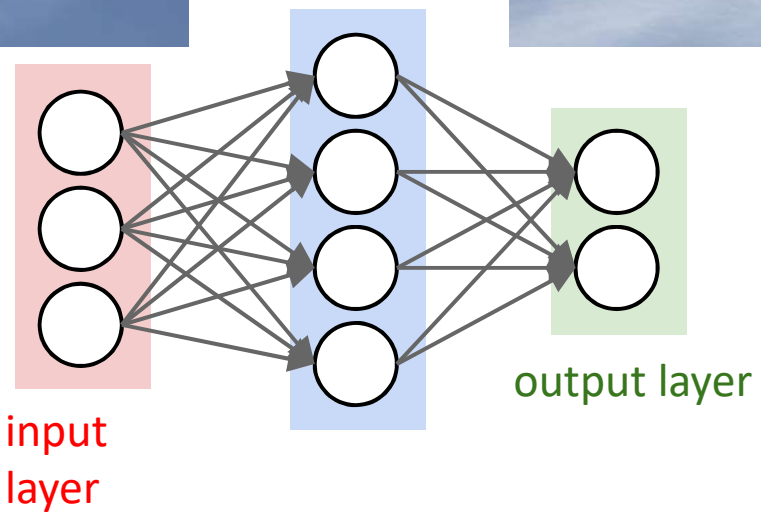
- The input layer comprises “feature detectors”
  - Detect if certain patterns have occurred in the input
- The network is a function over the feature detectors
- I.e. it is important for the *first* layer to capture relevant patterns

Distributed representations: The MLP as a cascade of feature detectors



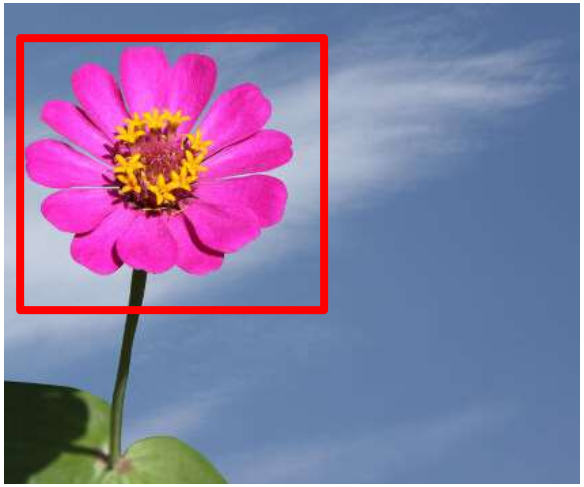
- The network is a cascade of feature detectors
  - **Higher level neurons compose complex templates from features represented by lower-level neurons**

A problem



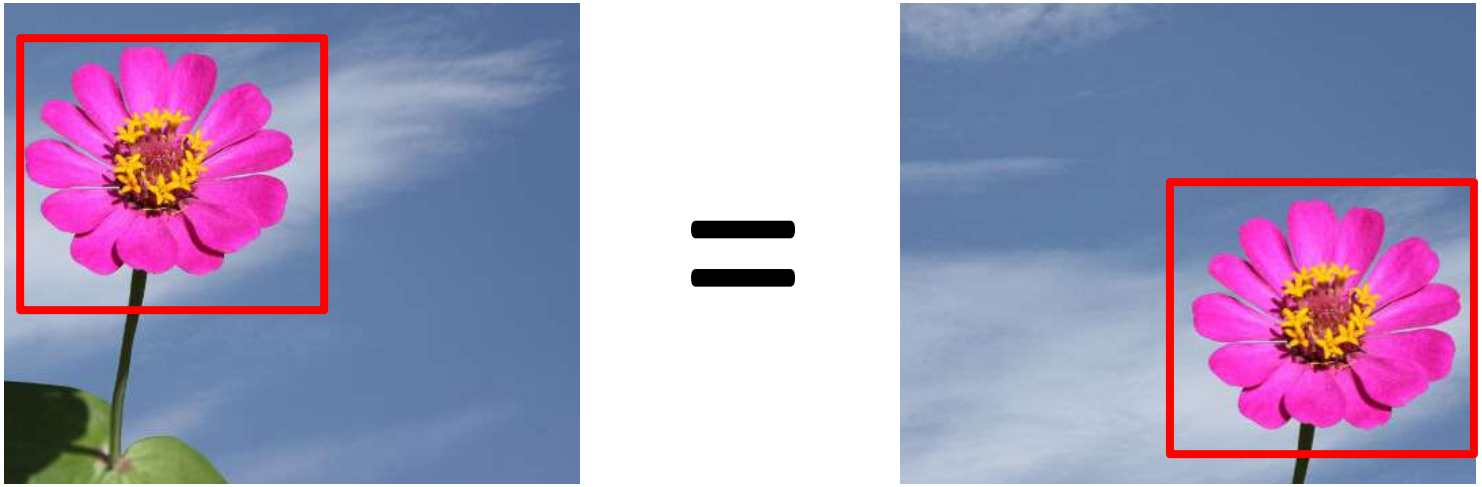
- Will an MLP that recognizes the left image as a flower also recognize the one on the right as a flower?

A problem



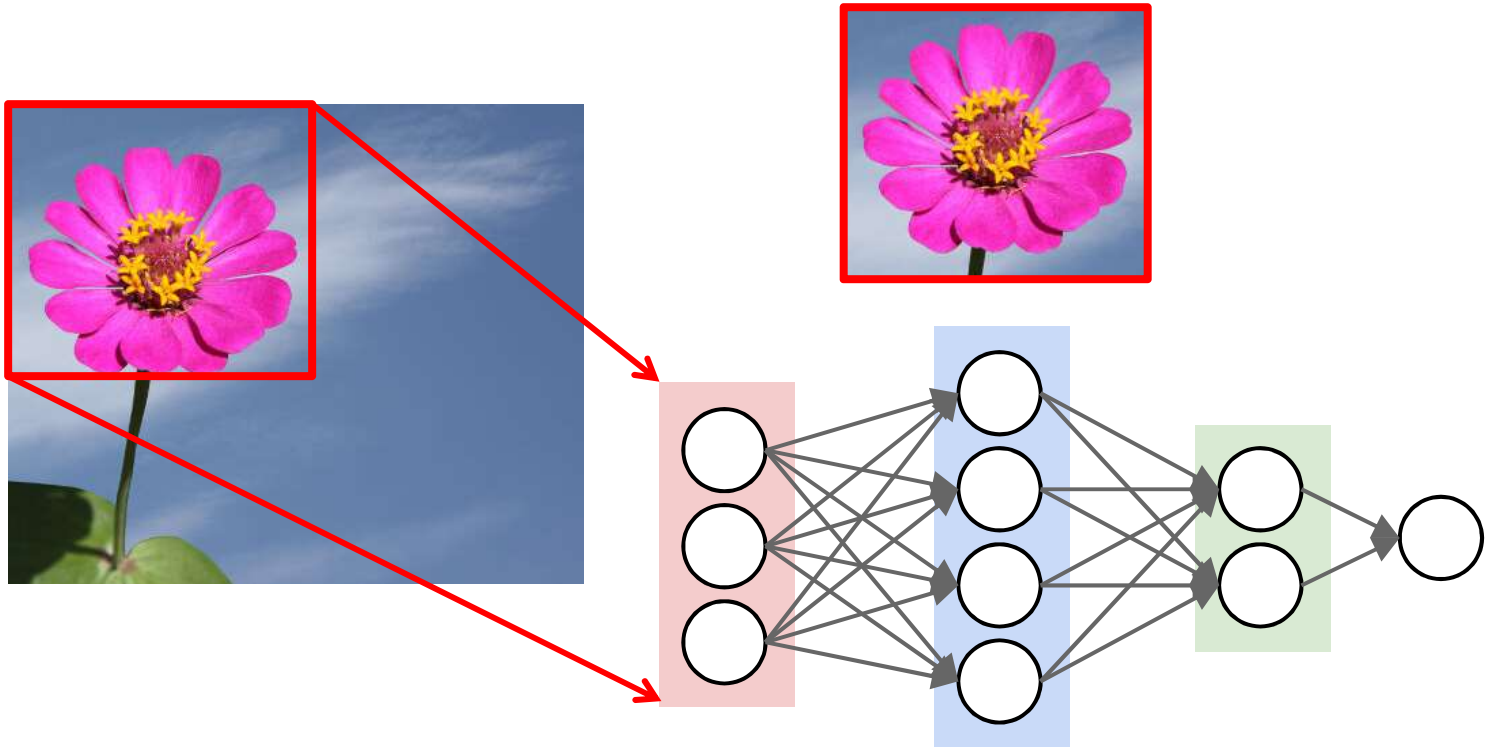
- Need a network that will “fire” regardless of the precise location of the target object

The need for *shift invariance*



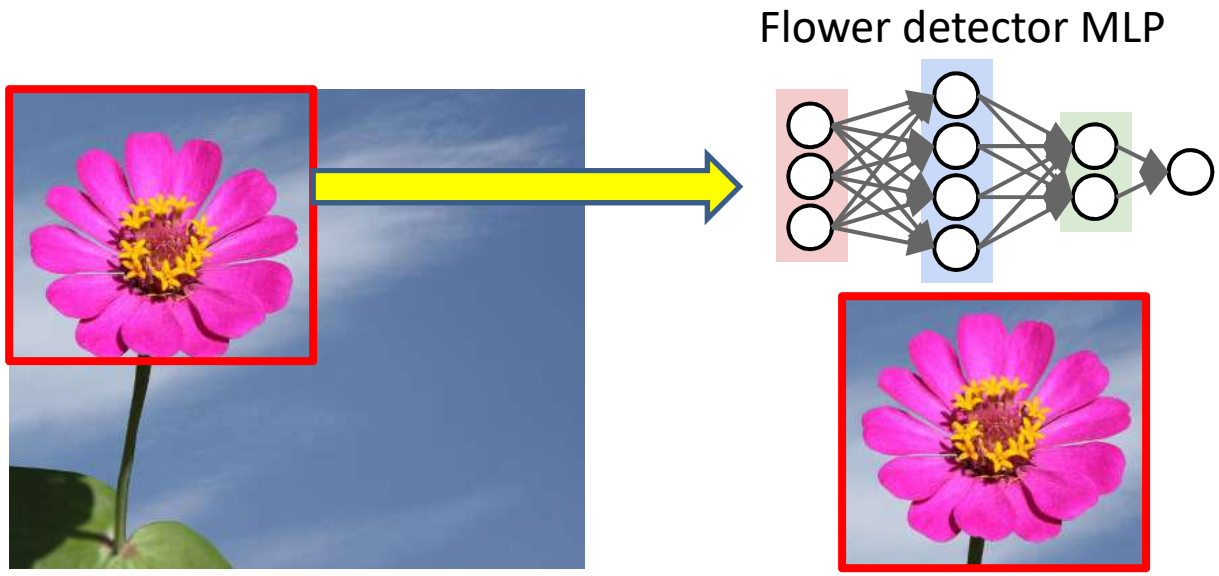
- In many problems the *location* of a pattern is not important
  - Only the presence of the pattern
- Conventional MLPs are sensitive to the location of the pattern
  - Moving it by one component results in an entirely different input that the MLP wont recognize
- Requirement: Network must be *shift invariant*

The 2-d analogue: Does this picture have a flower?



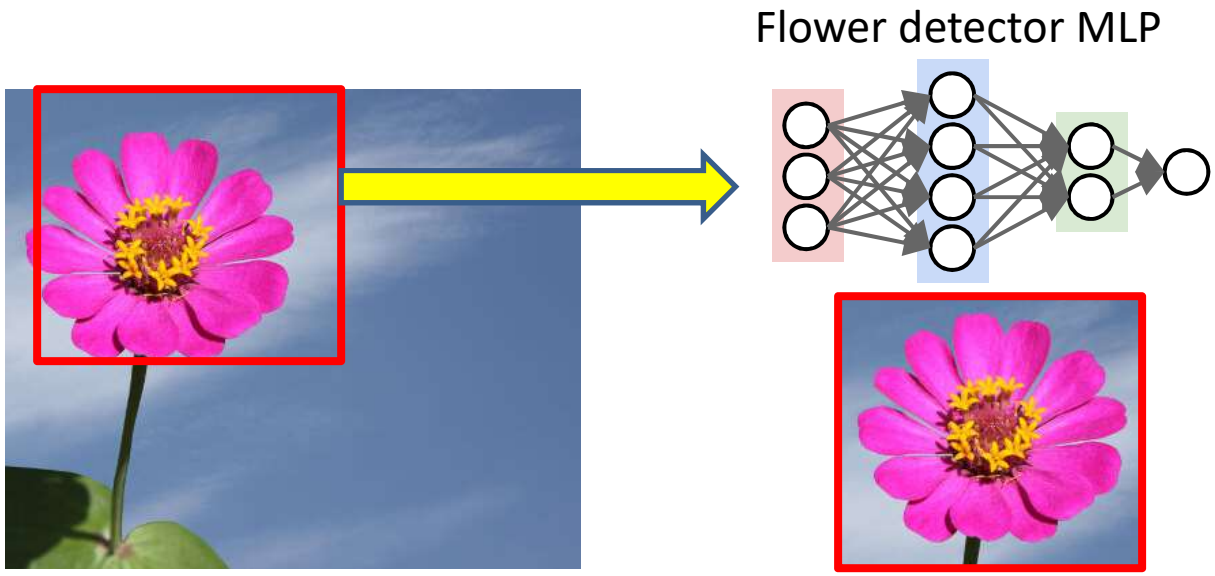
- *Scan* for the desired object
  - "Look" for the target object at each position

Solution: Scan



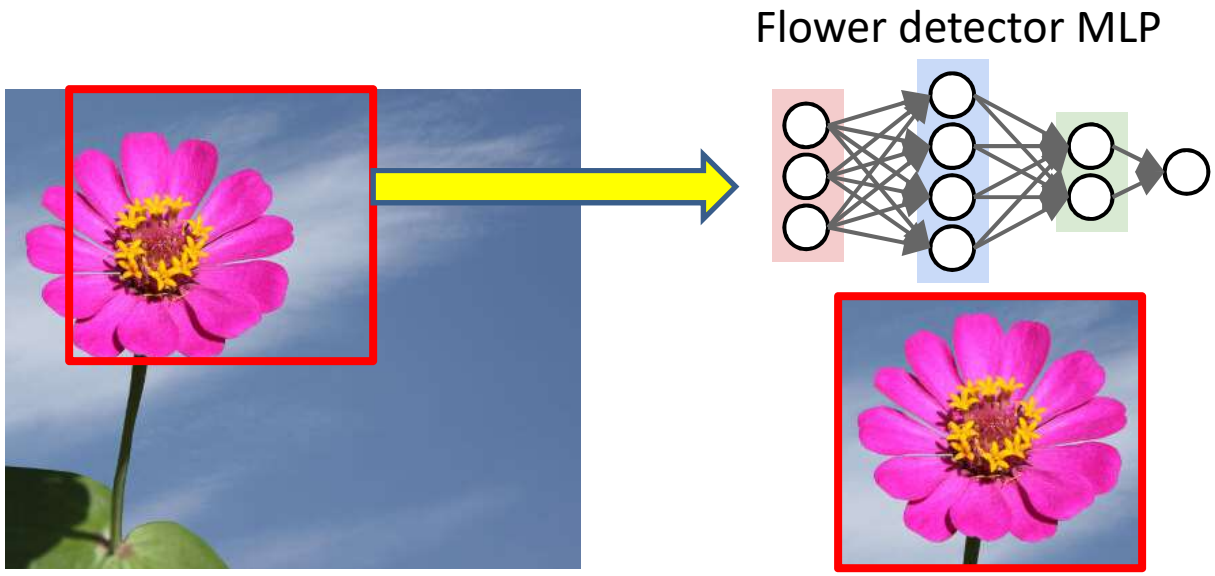
- *Scan* for the desired object

Solution: Scan



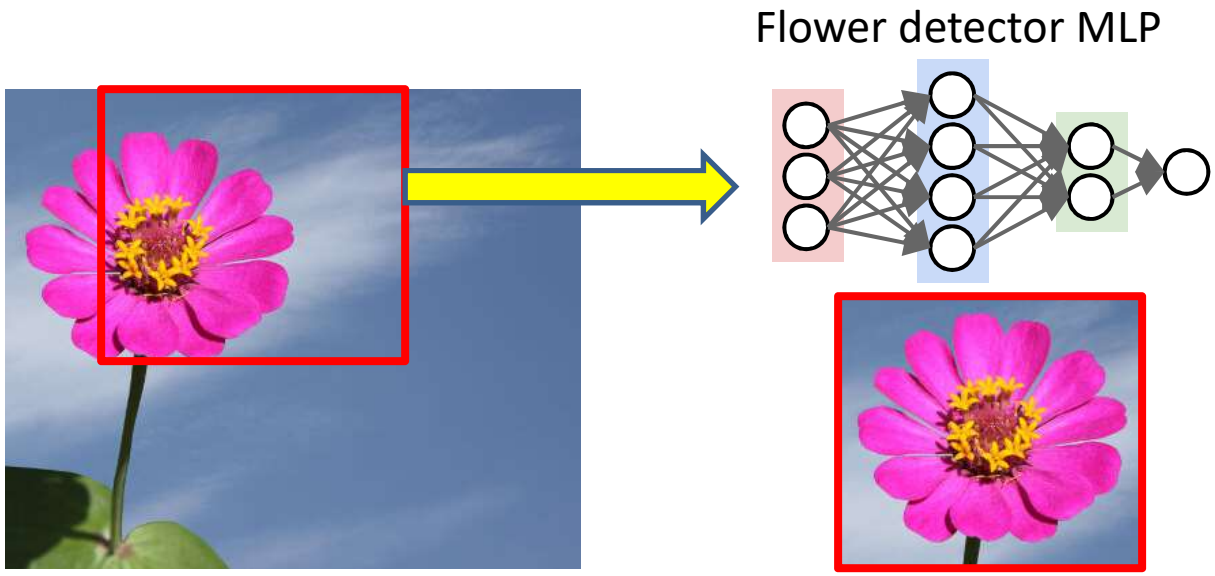
- *Scan* for the desired object

Solution: Scan



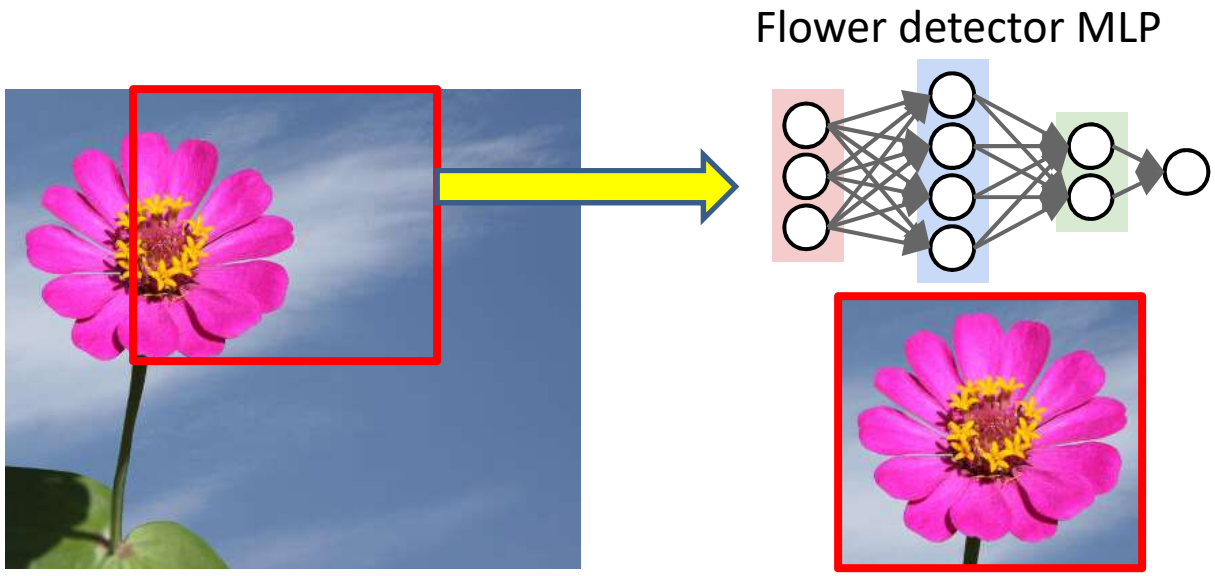
- *Scan* for the desired object

Solution: Scan



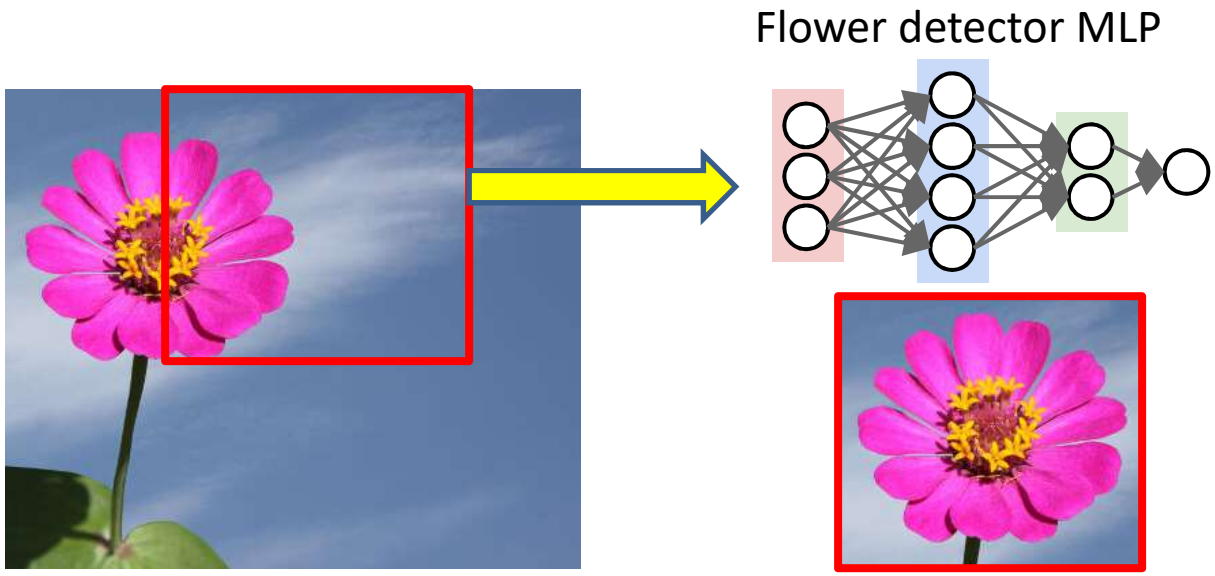
- *Scan* for the desired object

Solution: Scan



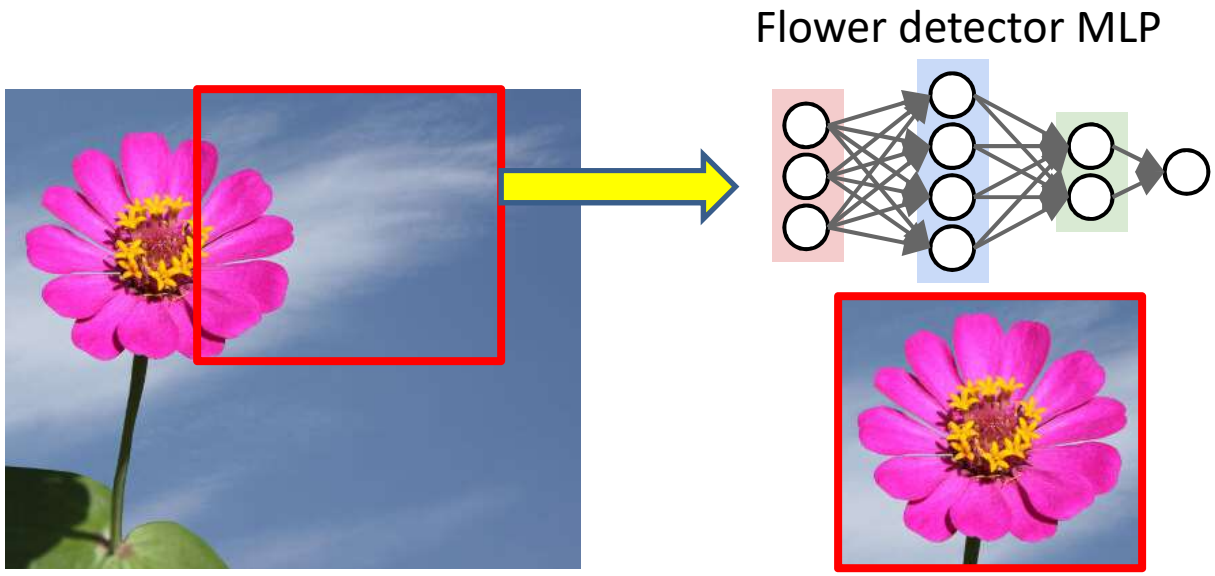
- *Scan* for the desired object

Solution: Scan



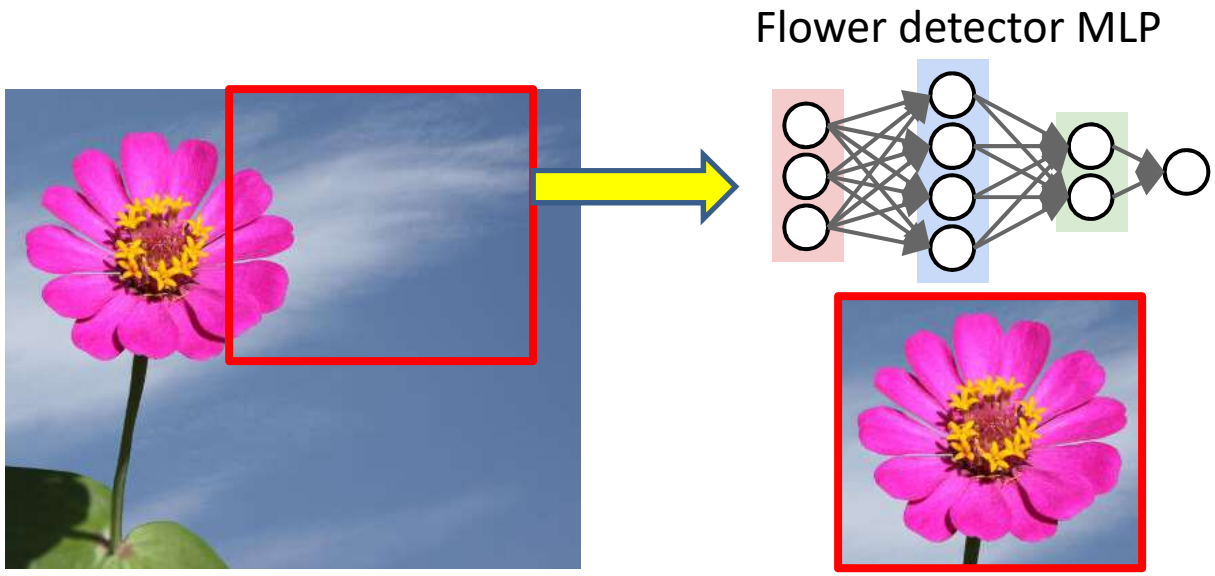
- *Scan* for the desired object

Solution: Scan



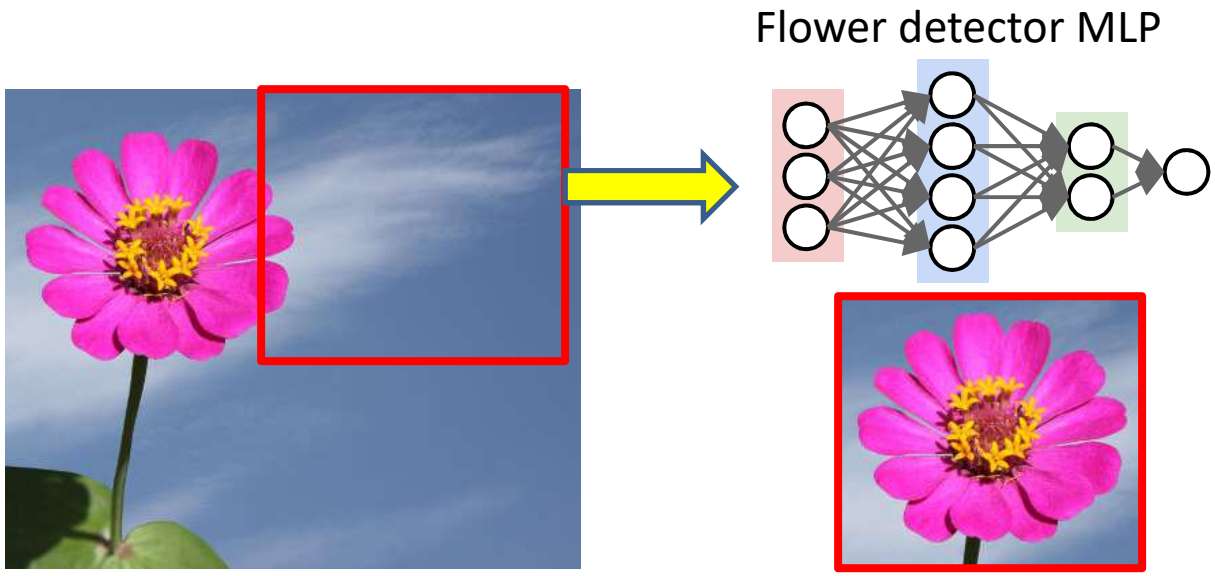
- *Scan* for the desired object

Solution: Scan



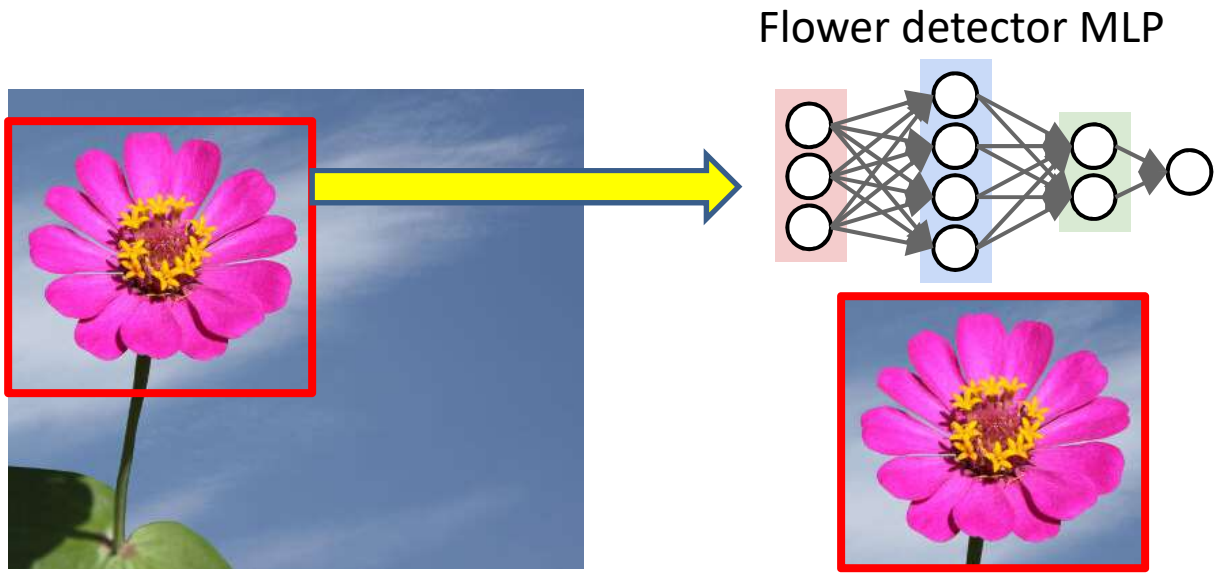
- *Scan* for the desired object

Solution: Scan



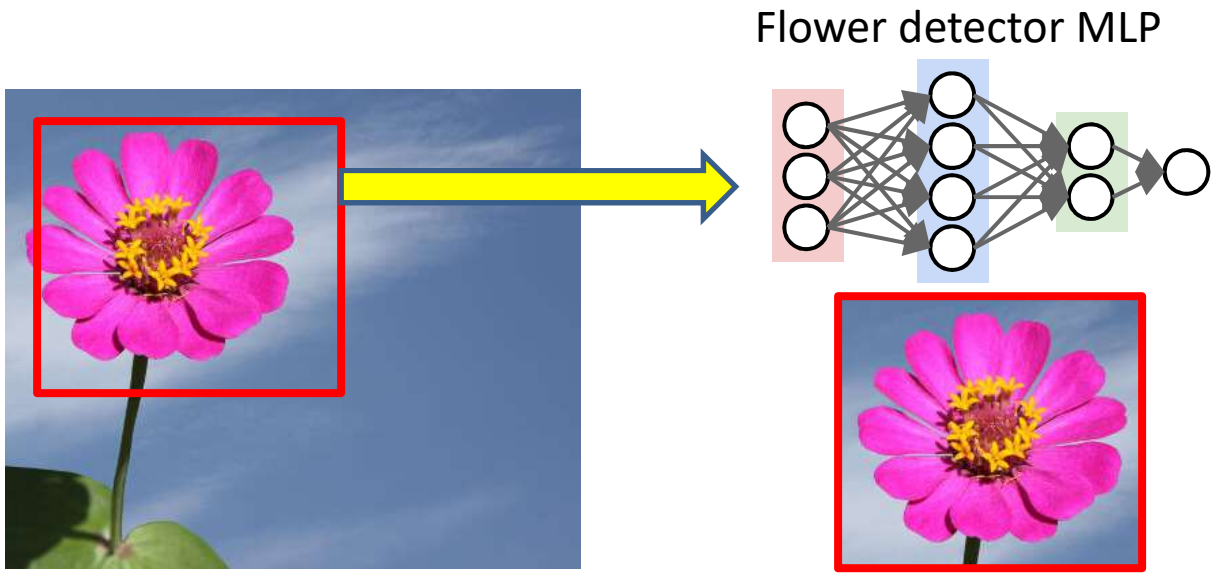
- *Scan* for the desired object

Solution: Scan



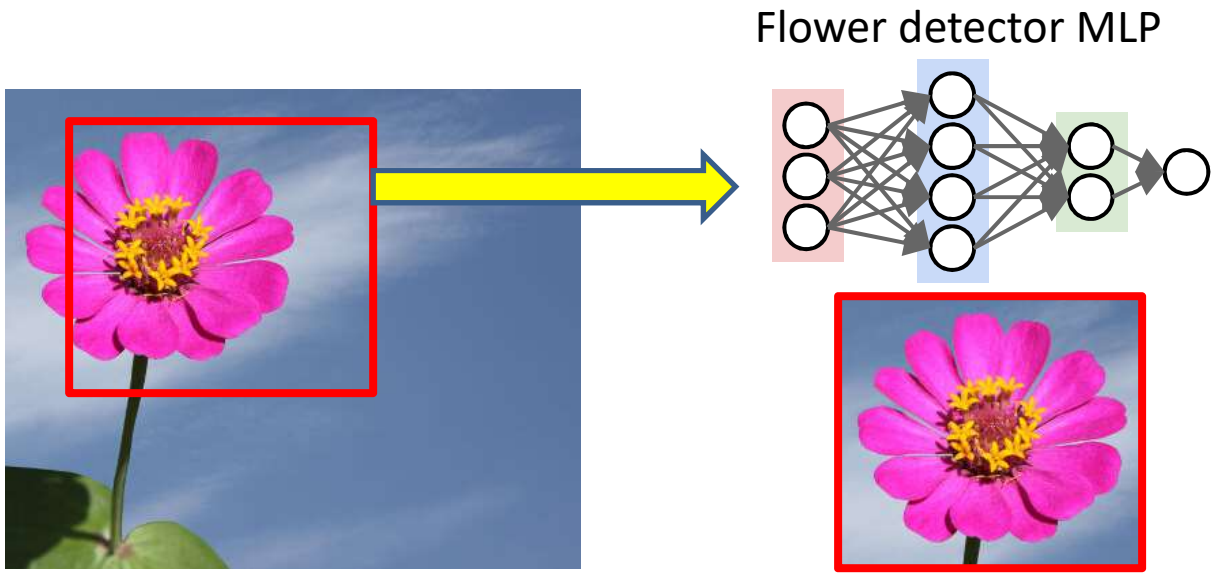
- *Scan* for the desired object

Solution: Scan



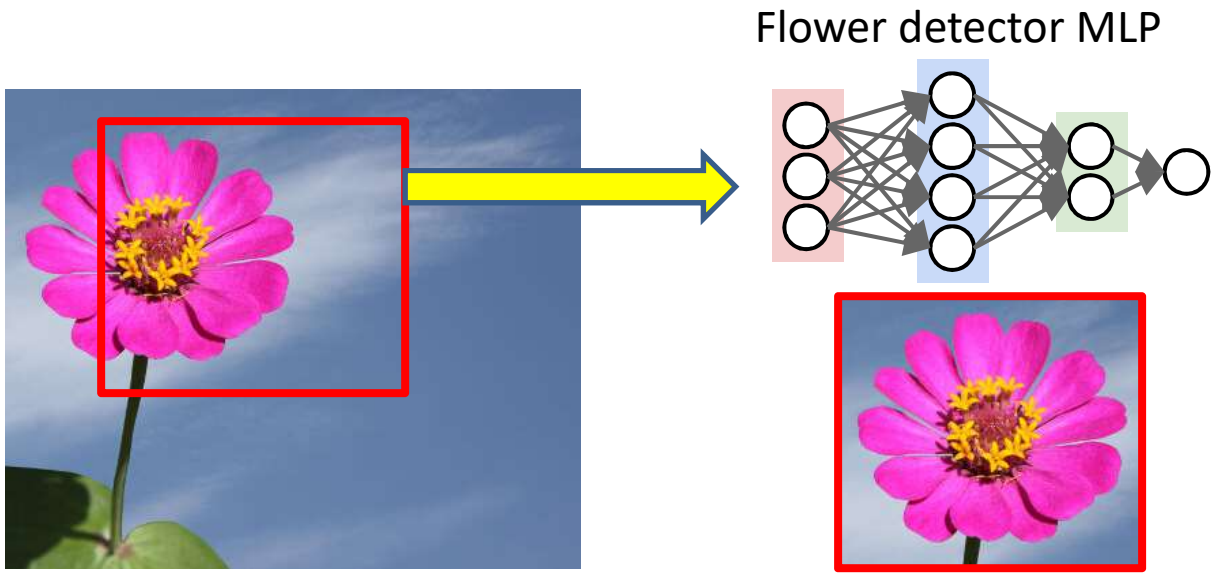
- *Scan* for the desired object

Solution: Scan



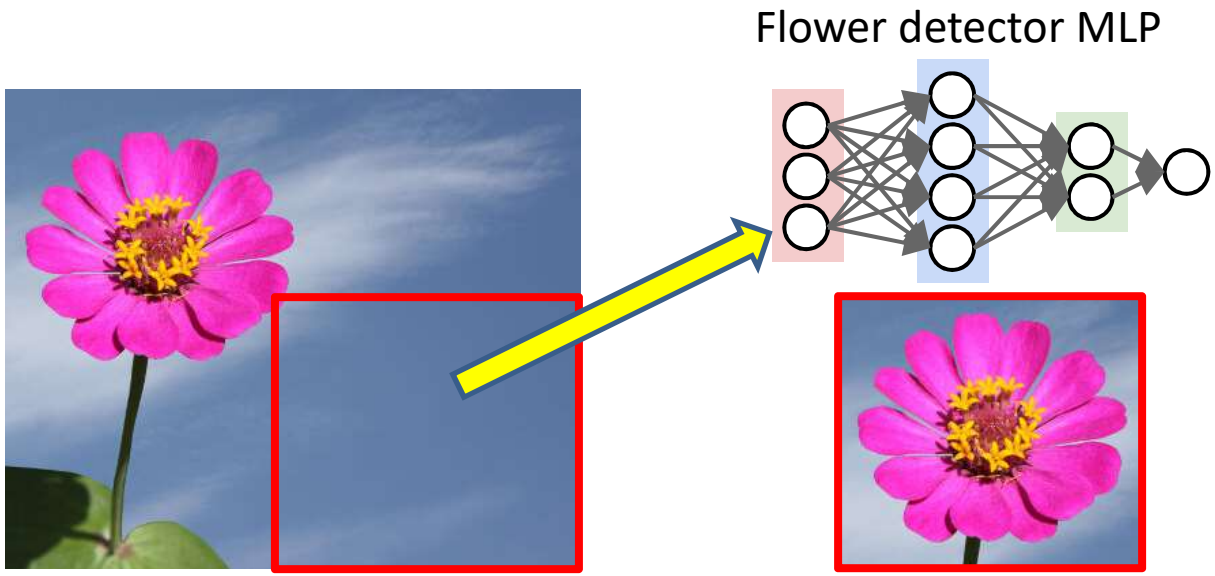
- *Scan* for the desired object

Solution: Scan



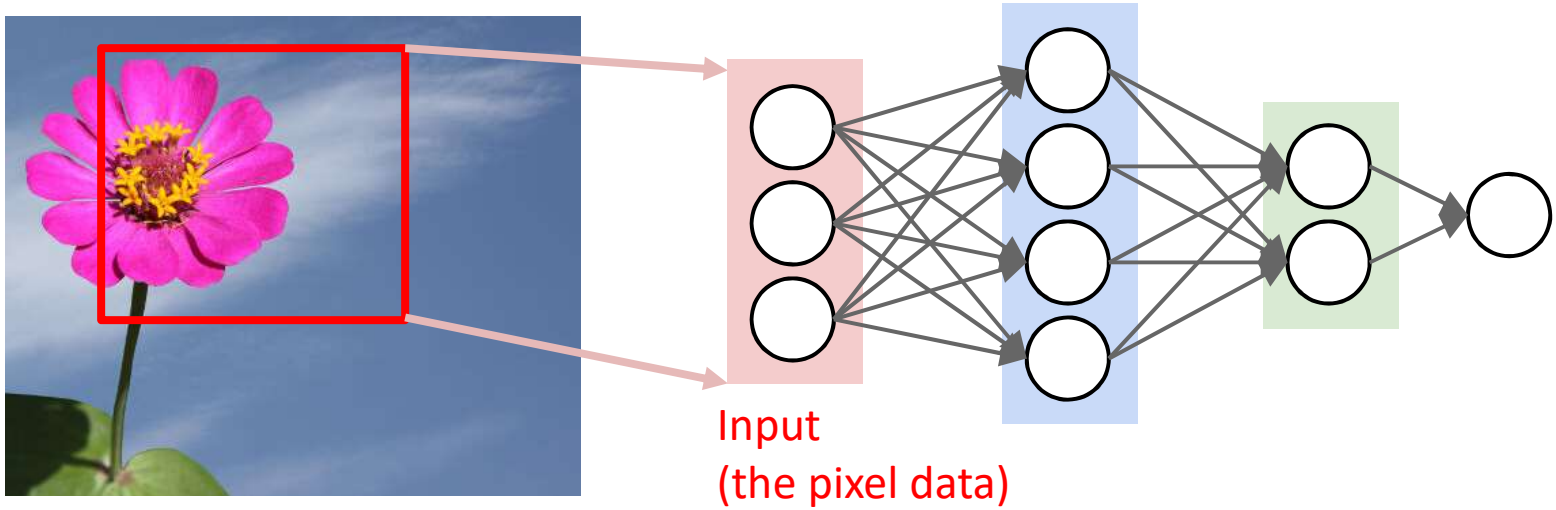
- *Scan* for the desired object

Solution: Scan



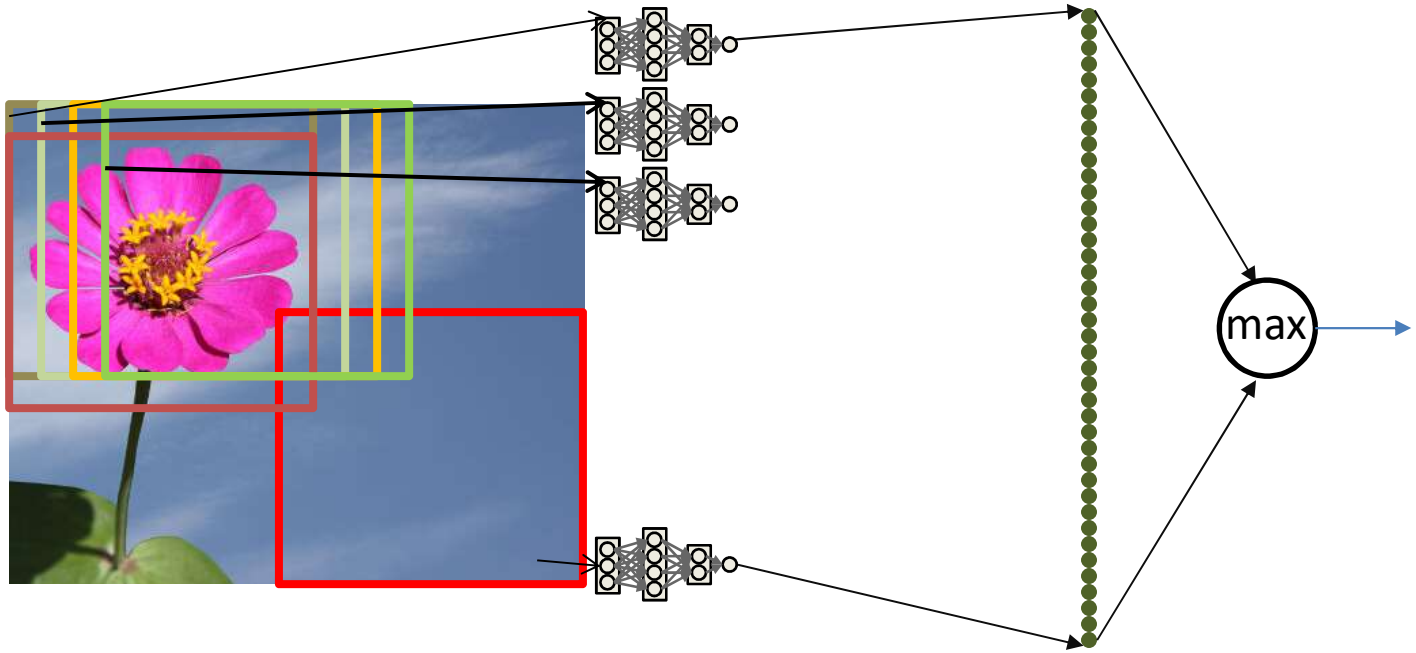
- *Scan* for the desired object

Scanning



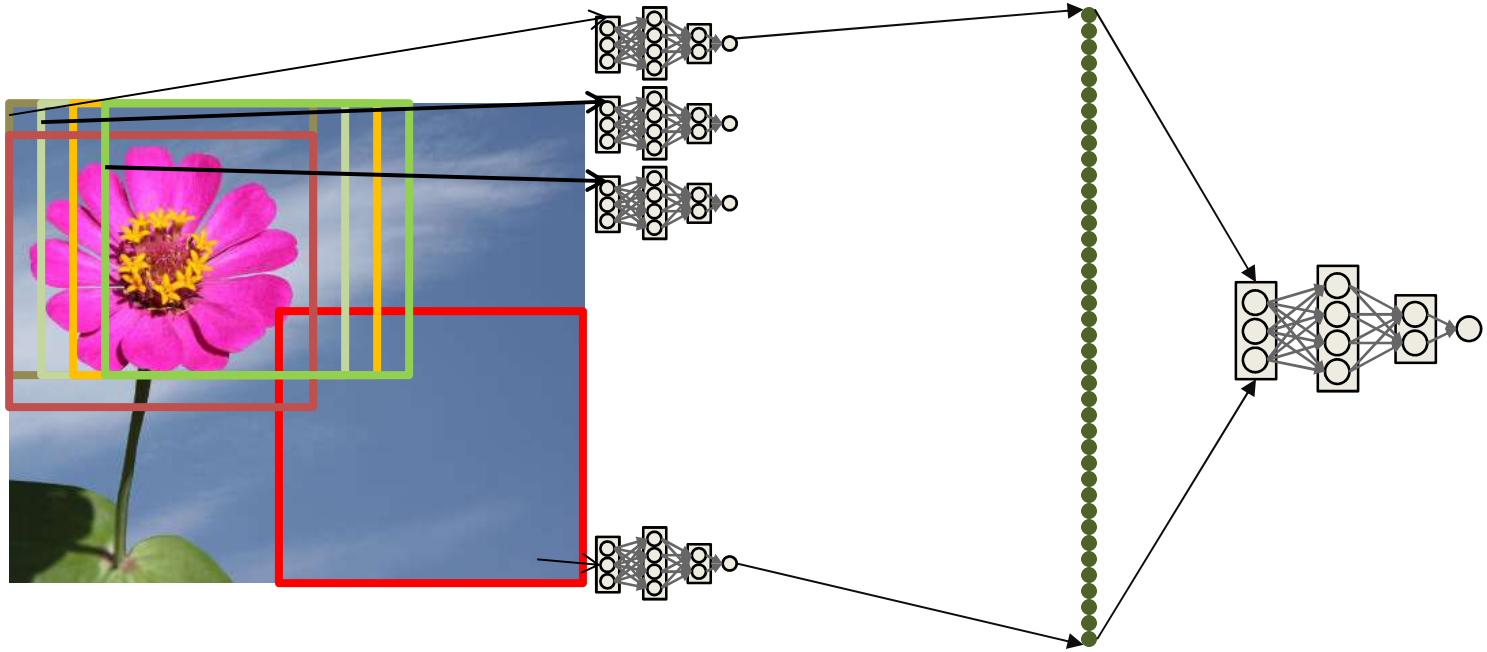
- *Scan* for the desired object
- At each location, the entire region is sent through an MLP

Scanning the picture to find a flower



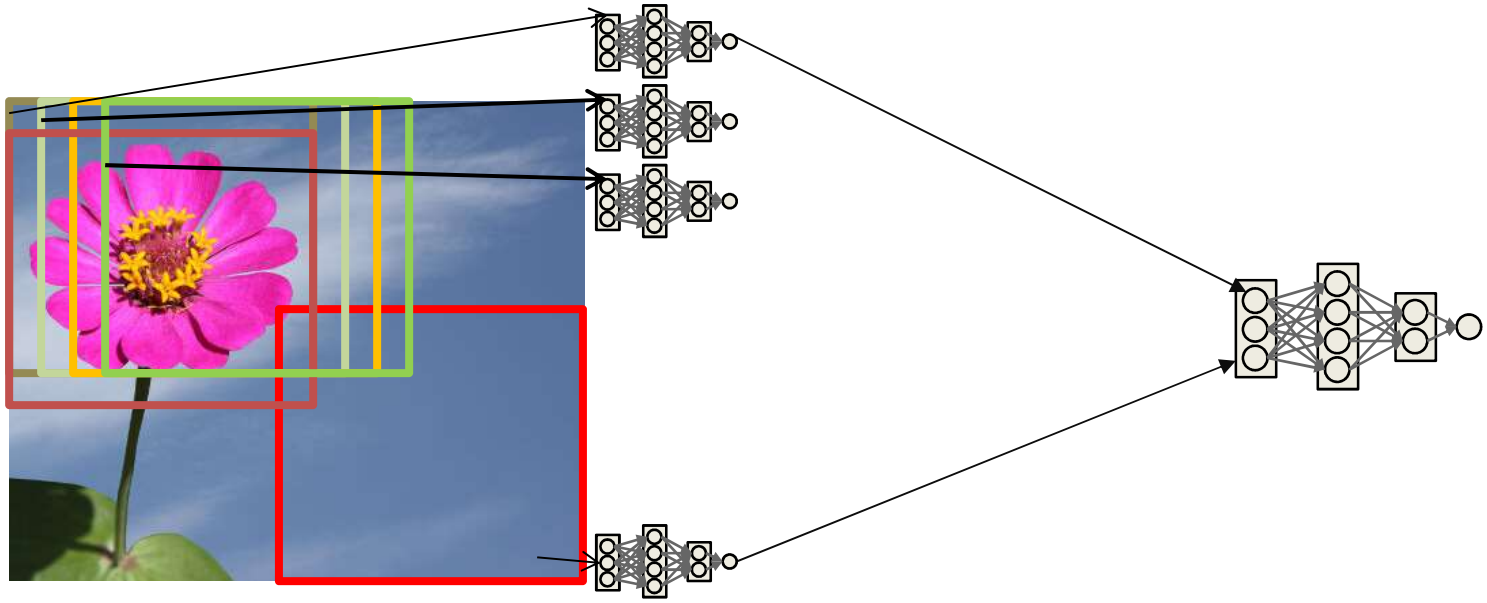
- Determine if any of the locations had a flower
  - We get one classification output per scanned location
    - Each dot in the right represents the output of the MLP when it classifies one location in the input figure
      - The score output by the MLP
  - Look at the maximum value

Its just a giant network with common subnets



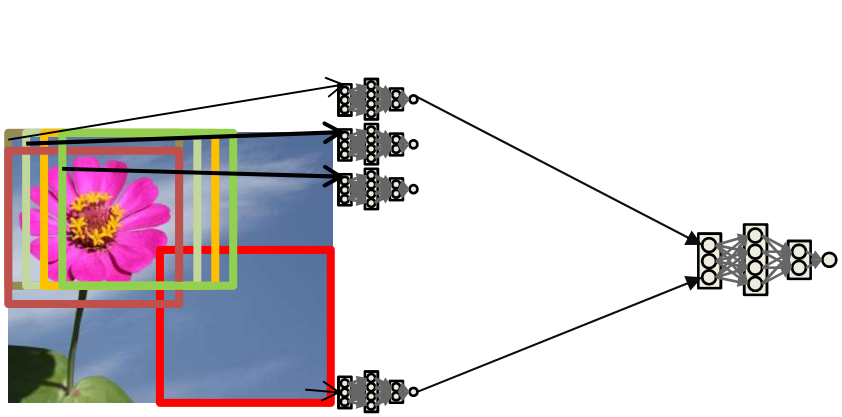
- Determine if any of the locations had a flower
  - Each dot in the right represents the output of the MLP when it classifies one location in the input figure
    - The score output by the MLP
  - Look at the maximum value
  - Or pass it through a softmax or even an MLP

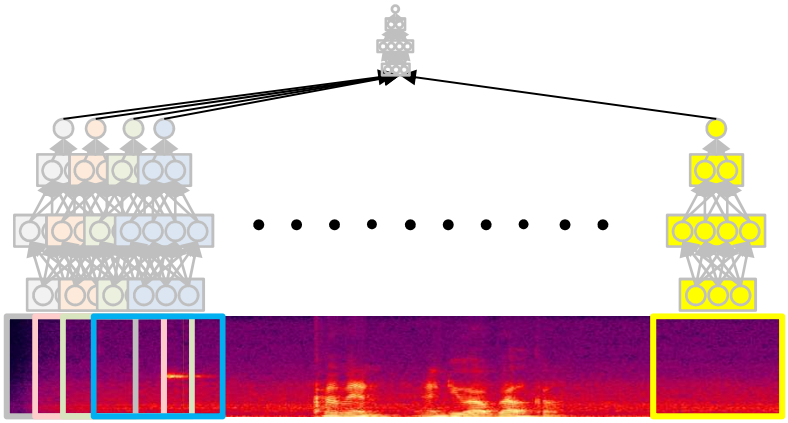
Its just a giant network with common subnets



- The entire operation can be viewed as a single giant network
  - Composed of many “subnets” (one per window)
  - With one key feature: all subnets are identical

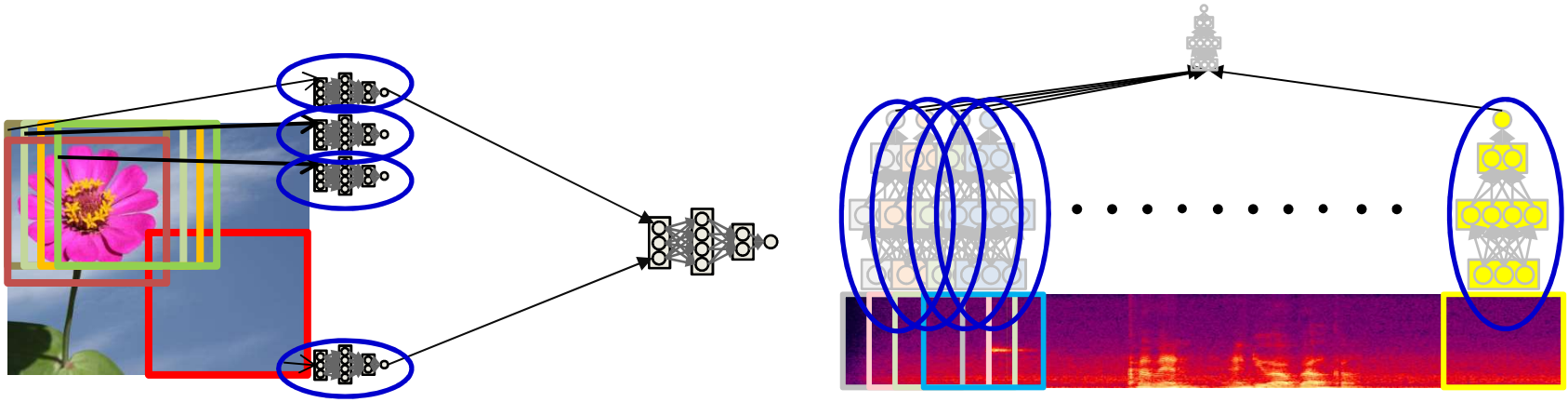
Training the network





- These are really just large networks
- Can just use conventional backpropagation to learn the parameters
  - Provide many training examples
    - Images with and without flowers
    - Speech recordings with and without the word “welcome”
  - Gradient descent to minimize the total divergence between predicted and desired outputs
- Backprop learns a network that maps the training inputs to the target binary outputs

Training the network: constraint



- These are *shared parameter* networks
  - All lower-level subnets are identical
    - Are all searching for the same pattern
  - Any update of the parameters of one copy of the subnet must equally update *all* copies

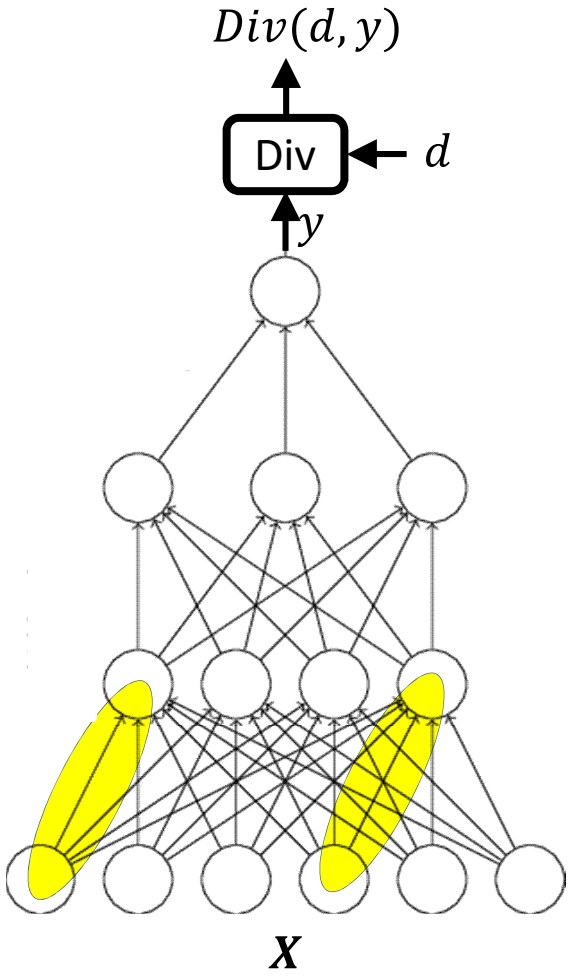
Learning in shared parameter networks

- Consider a simple network with shared weights

$$w_{ij}^{k} = w_{mn}^{l} = w^{\mathcal{S}}$$

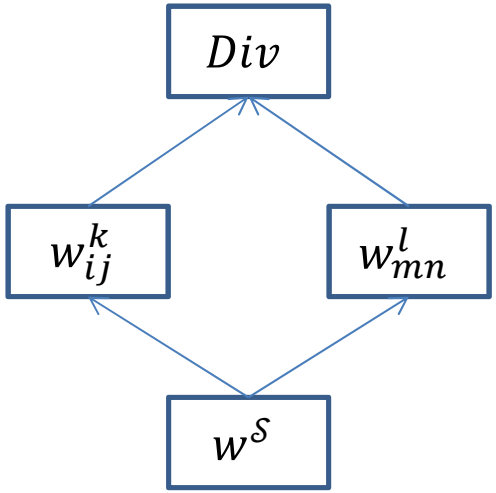
  - A weight w_{ij}^{k} is required to be identical to the weight w_{mn}^{l}

- For any training instance $\boldsymbol{X}$, a small perturbation of $w^{\mathcal{S}}$ perturbs both w_{ij}^{k} and w_{mn}^{l} identically
  - Each of these perturbations will individually influence the divergence $Div(d, y)$



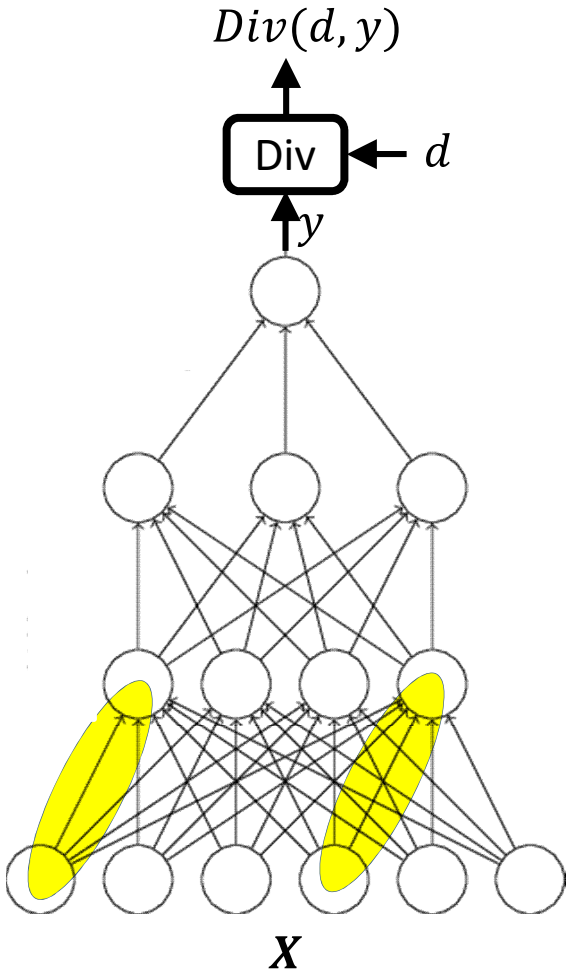
Computing the divergence of shared parameters

Influence diagram



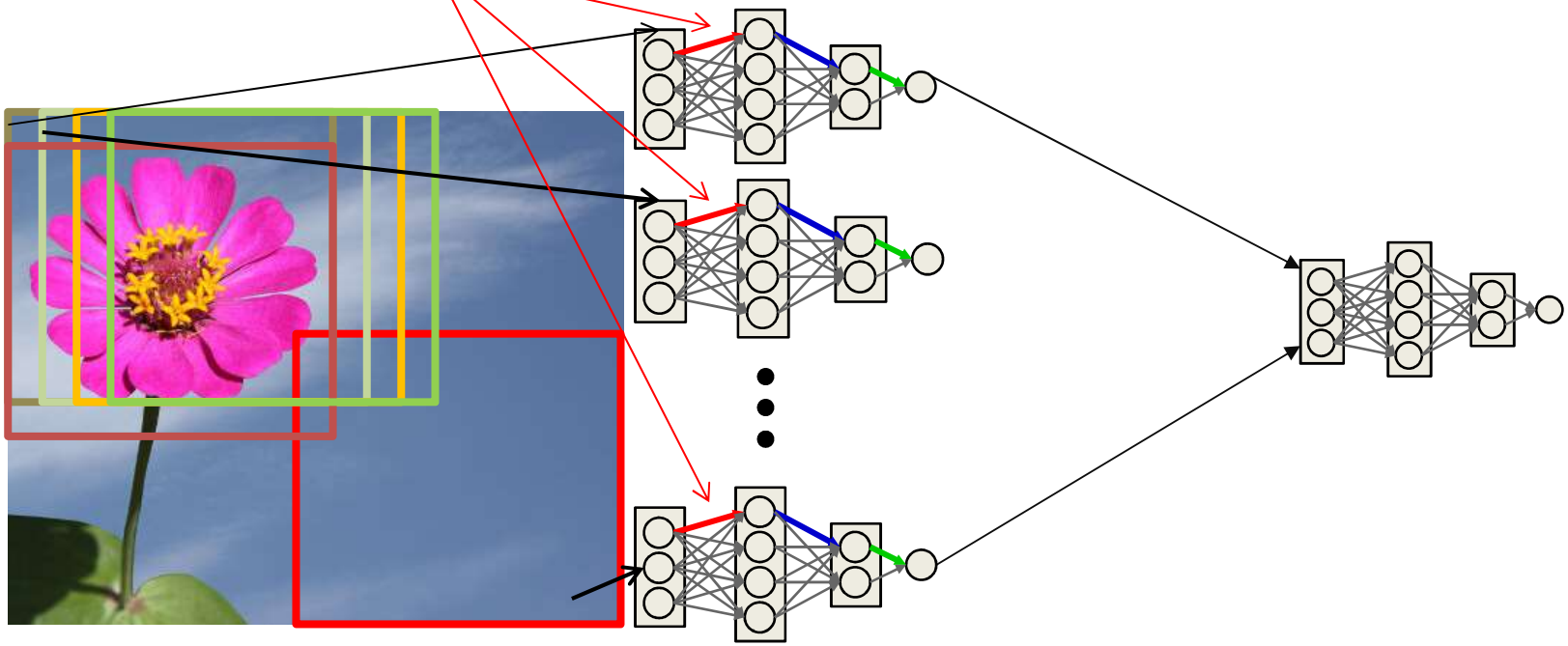
$$\frac{dDiv}{dw^{s}} = \frac{\partial Div}{\partial w_{ij}^{k}}\frac{dw_{ij}^{k}}{dw^{s}} + \frac{\partial Div}{\partial w_{mn}^{l}}\frac{dw_{mn}^{l}}{dw^{s}}$$

$$= \frac{\partial Div}{\partial w_{ij}^{k}} + \frac{\partial Div}{\partial w_{mn}^{l}}$$



- Each of the individual terms can be computed via backpropagation

Computing the divergence of shared parameters



- More generally, let $\mathcal{S}$ be any set of edges that have a common value, and $w^{\mathcal{S}}$ be the common weight of the set
  - E.g. the set of all red weights in the figure

$$\frac{dDiv}{dw^{\mathcal{S}}} = \sum_{e \in \mathcal{S}} \frac{\partial Div}{\partial w^e}$$

- The individual terms in the sum can be computed via backpropagation

Training networks with shared parameters

- Gradient descent algorithm:
- Initialize all weights $\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_K$
- Do:
  - For every set $\mathcal{S}$:
    - Compute:

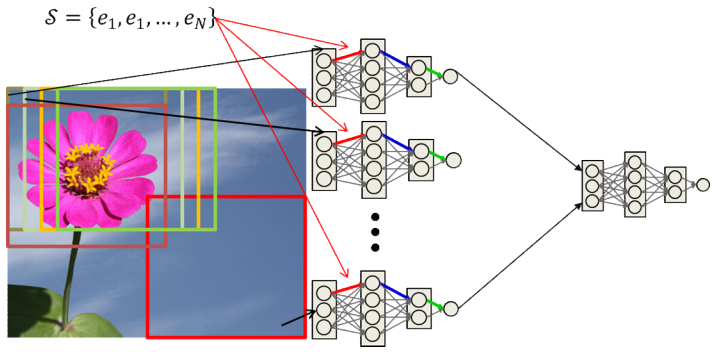
$$\nabla_{\mathcal{S}} Err = \frac{dErr}{dw^{\mathcal{S}}}$$

$$w^{\mathcal{S}} = w^{\mathcal{S}} - \eta \nabla_{\mathcal{S}} Err$$

    - For every $(k, i, j) \in \mathcal{S}$ update:

$$w_{i,j}^{(k)} = w^{\mathcal{S}}$$

- Until Err has converged



Training networks with shared parameters

- Gradient descent algorithm:
- Initialize all weights $\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_K$
- Do:
  - For every set $\mathcal{S}$:
    - Compute:

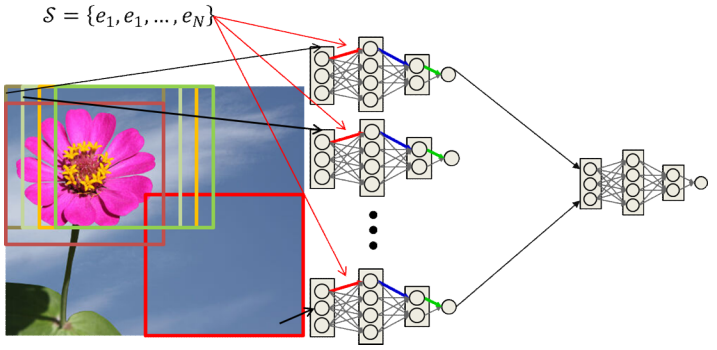
$$\nabla_{\mathcal{S}} Err = \frac{dErr}{dw^{\mathcal{S}}}$$

$$w^{\mathcal{S}} = w^{\mathcal{S}} - \eta \nabla_{\mathcal{S}} Err$$

    - For every $(k, i, j) \in \mathcal{S}$ update:

$$w_{i,j}^{(k)} = w^{\mathcal{S}}$$

- Until Err has converged



Training networks with shared parameters

- For every training instance X
  - For every set $\mathcal{S}$:
    - For every $(k, i, j) \in \mathcal{S}$:

$$\nabla_{\mathcal{S}} Div \mathrel{+}= \frac{\partial Div}{\partial w_{i,j}^{(k)}}$$

    - $\nabla_{\mathcal{S}} Err \mathrel{+}= \nabla_{\mathcal{S}} Div$
  - Compute:

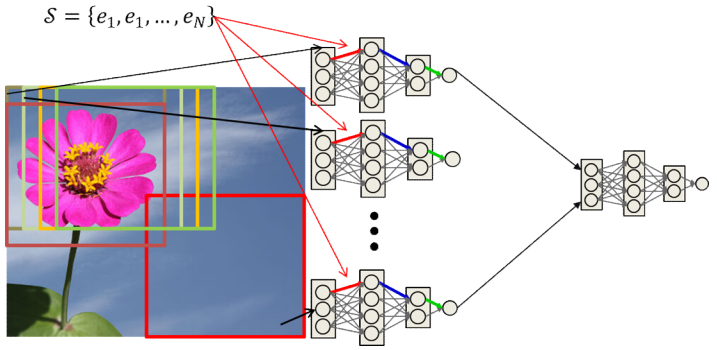
$$\nabla_{\mathcal{S}} Err = \frac{dErr}{dw^{\mathcal{S}}}$$

$$w^{\mathcal{S}} = w^{\mathcal{S}} - \eta \nabla_{\mathcal{S}} Err$$

  - For every $(k, i, j) \in \mathcal{S}$ update:

$$w_{i,j}^{(k)} = w^{\mathcal{S}}$$

- Until Err has converged



Training networks with shared parameters

- For every training instance X
  - For every set $\mathcal{S}$:
    - For every $(k, i, j) \in \mathcal{S}$:

$$\nabla_{\mathcal{S}} Div \mathrel{+}= \frac{\partial Div}{\partial w_{i,j}^{(k)}}$$

Computed by Backprop

    - $\nabla_{\mathcal{S}} Err \mathrel{+}= \nabla_{\mathcal{S}} Div$
  - Compute:

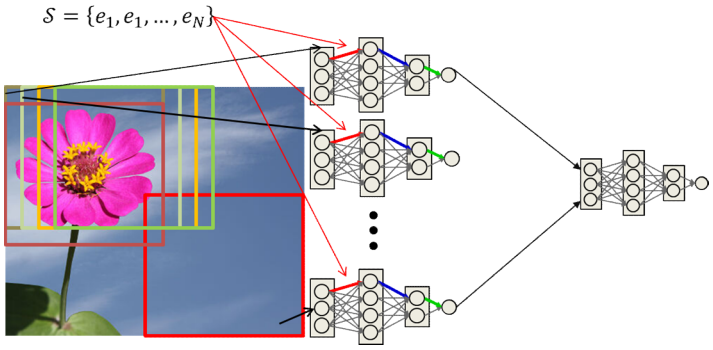
$$\nabla_{\mathcal{S}} Err = \frac{dErr}{dw^{\mathcal{S}}}$$

$$w^{\mathcal{S}} = w^{\mathcal{S}} - \eta \nabla_{\mathcal{S}} Err$$

  - For every $(k, i, j) \in \mathcal{S}$ update:

$$w_{i,j}^{(k)} = w^{\mathcal{S}}$$

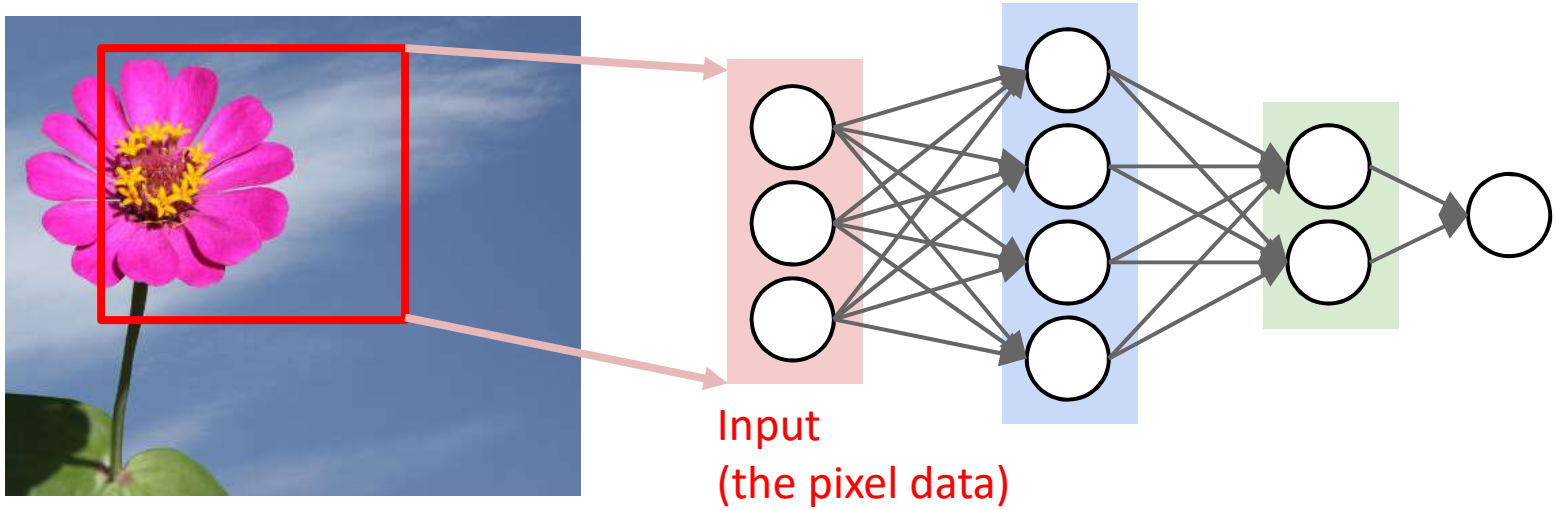
- Until Err has converged



Story so far

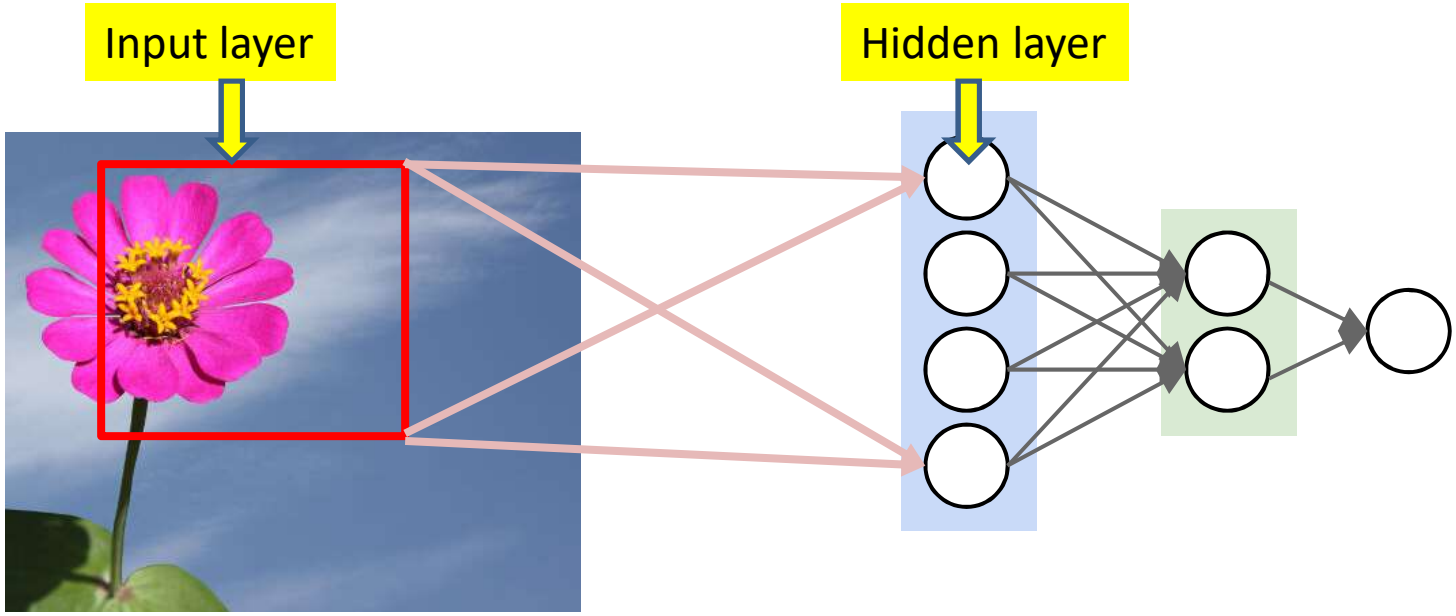
- Position-invariant pattern classification can be performed by scanning
  - 1-D scanning for sound
  - 2-D scanning for images
  - 3-D and higher-dimensional scans for higher dimensional data
- Scanning is equivalent to composing a large network with repeating subnets
  - The large network has shared subnets
- Learning in scanned networks: Backpropagation rules must be modified to combine gradients from parameters that share the same value
  - The principle applies in general for networks with shared parameters

Scanning in 2D: A closer look



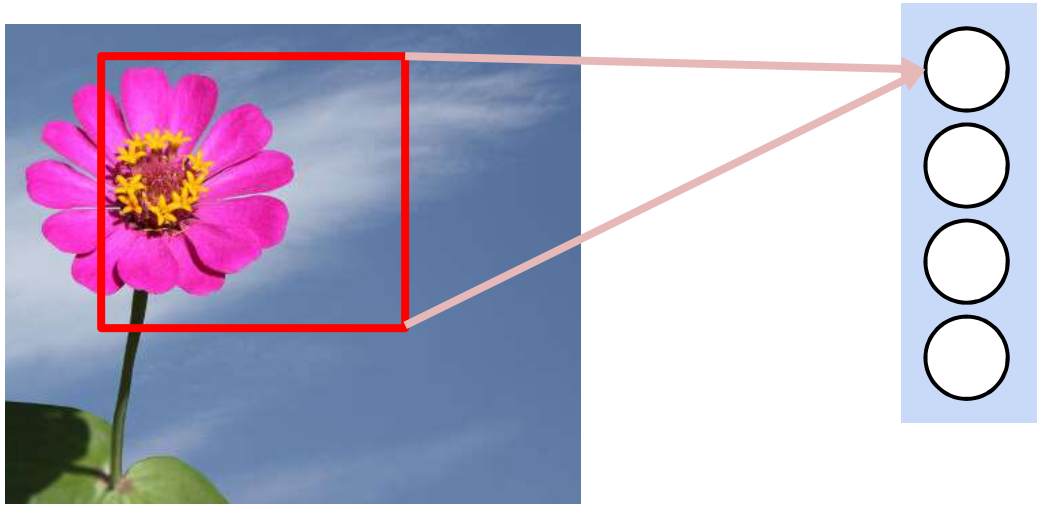
- *Scan* for the desired object
- At each location, the entire region is sent through an MLP

Scanning: A closer look



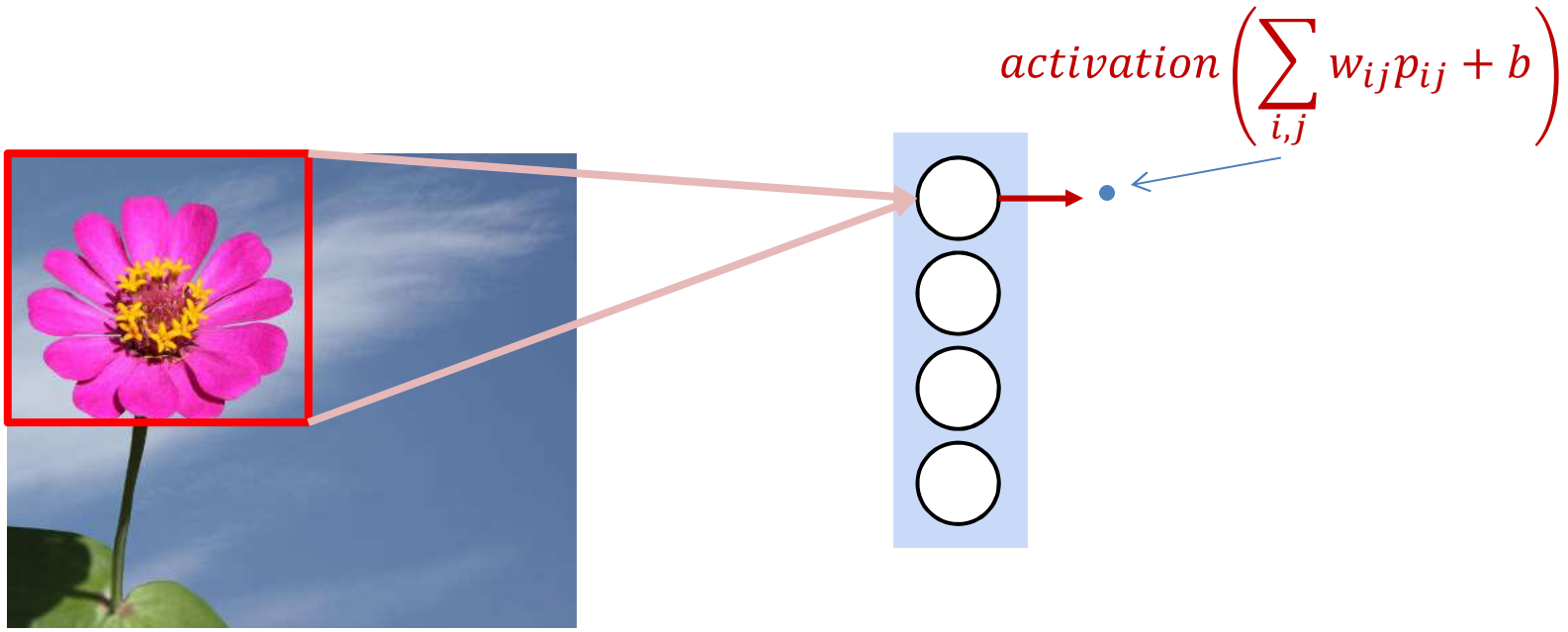
- The “input layer” is just the pixels in the image connecting to the hidden layer

Scanning: A closer look



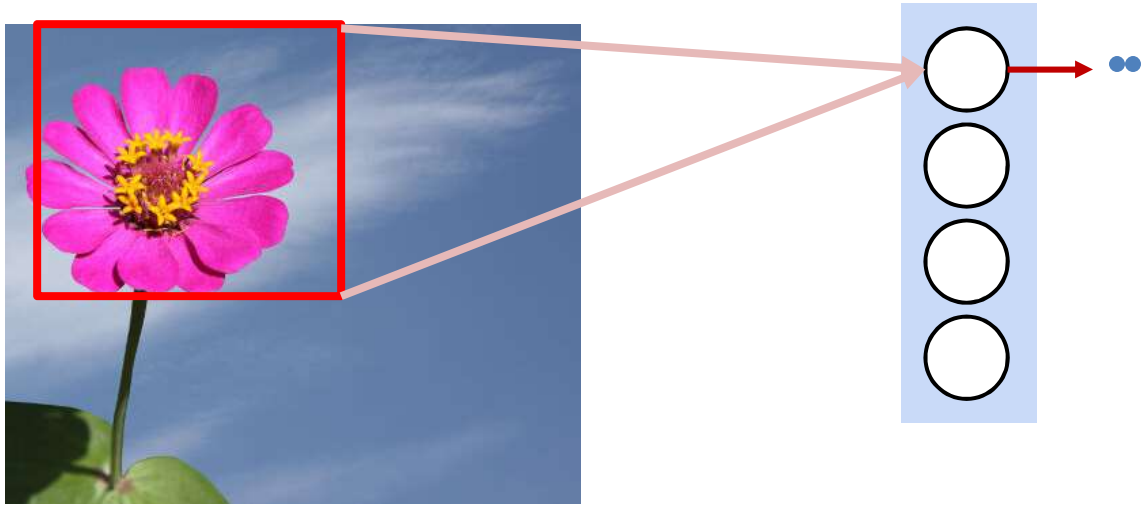
- Consider a single neuron

Scanning: A closer look



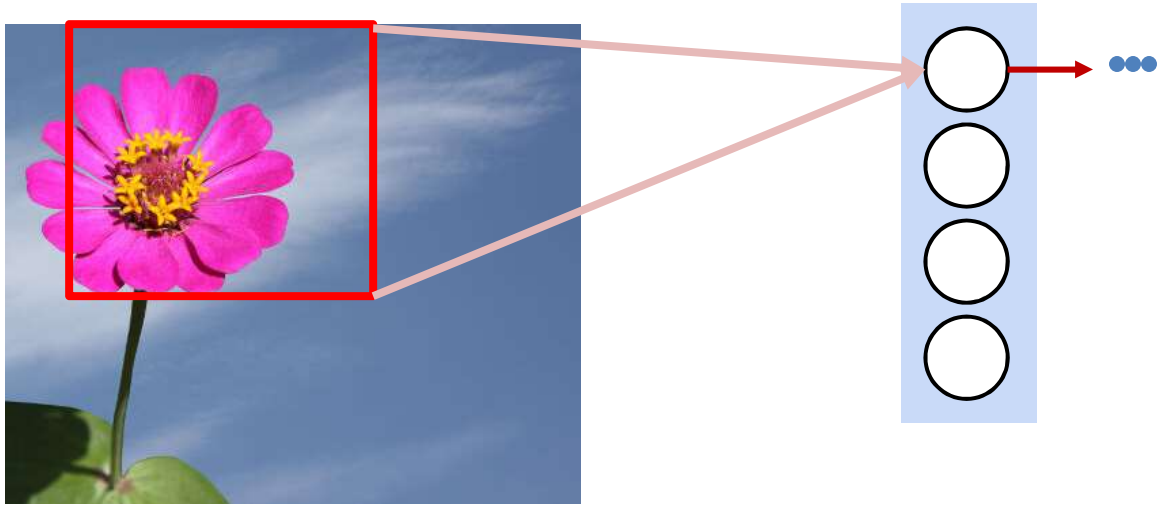
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the part of the picture in the box as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture

Scanning: A closer look



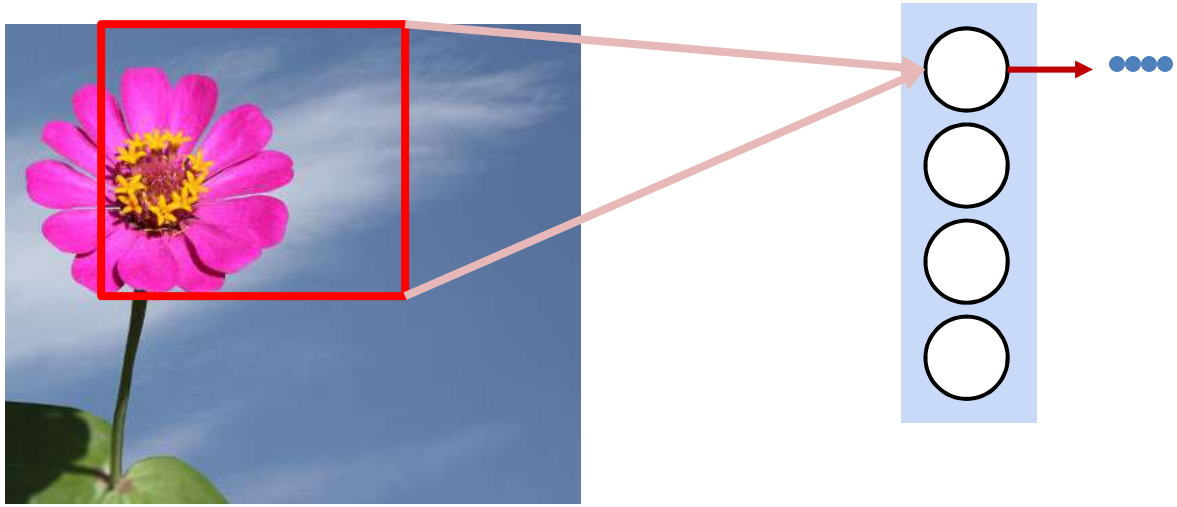
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture

Scanning: A closer look



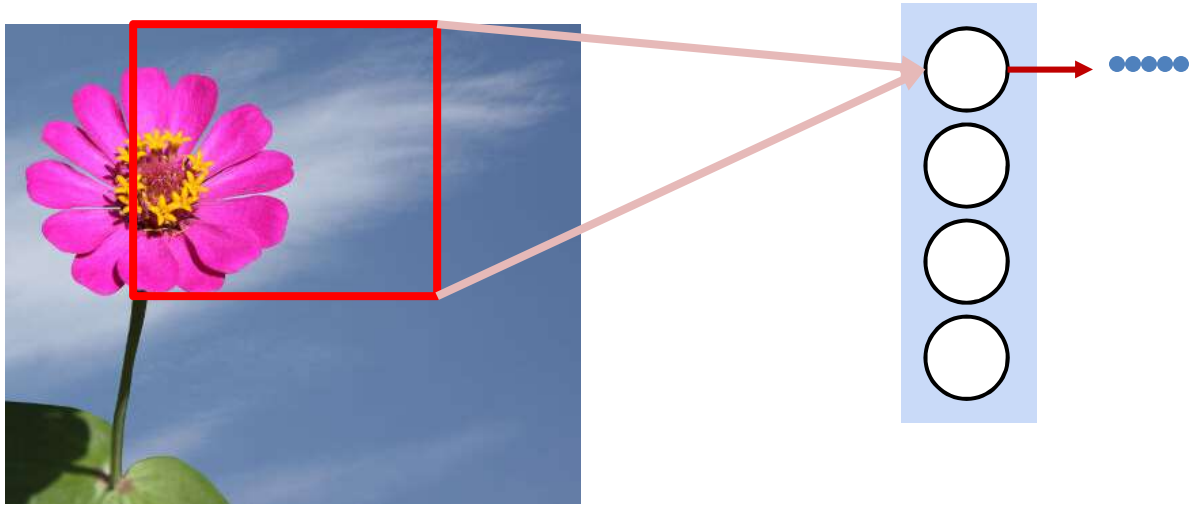
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture

Scanning: A closer look



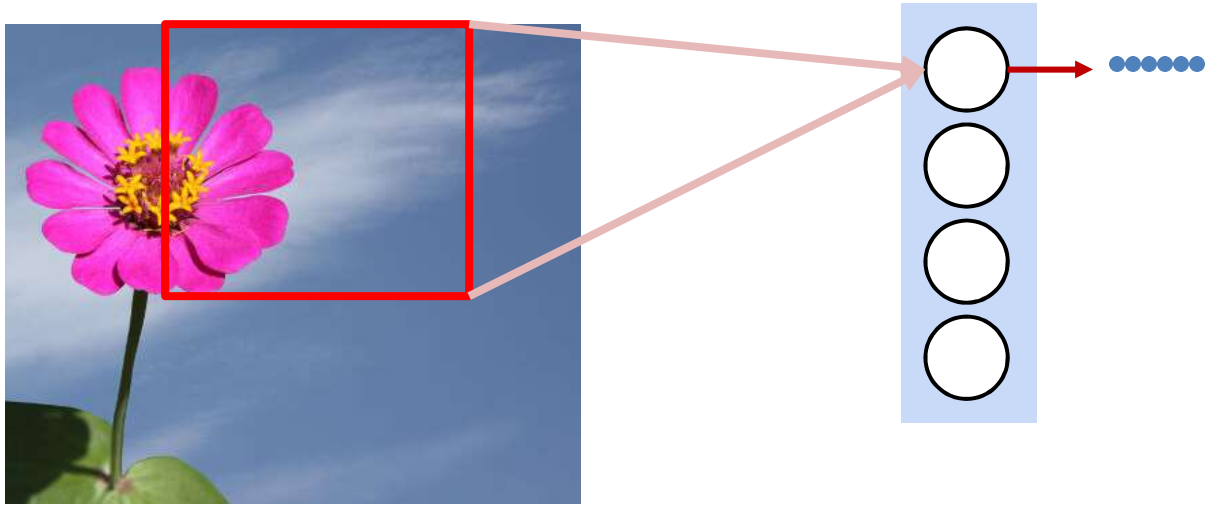
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture

Scanning: A closer look



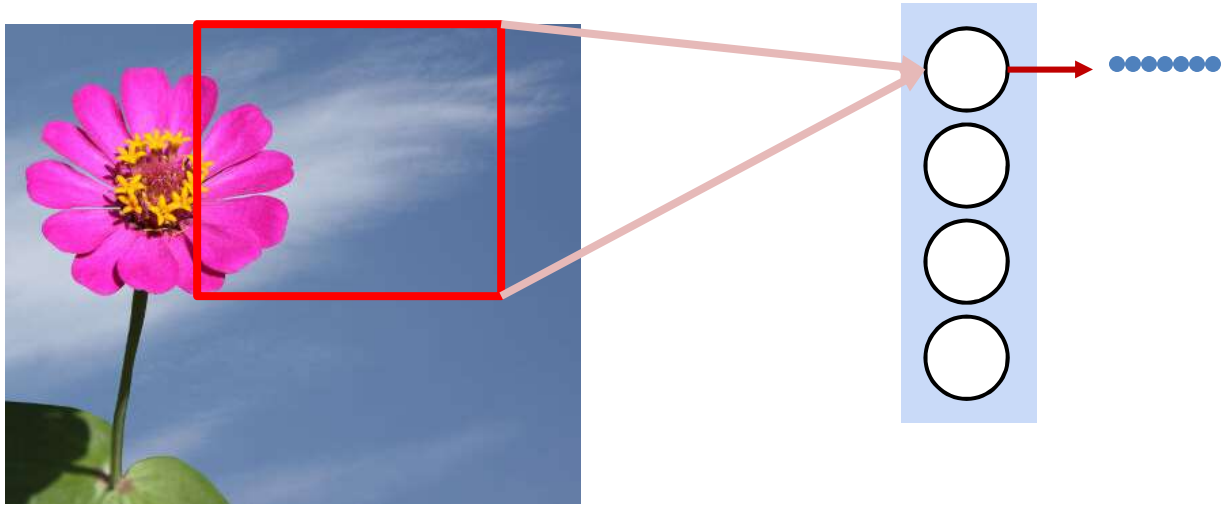
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture

Scanning: A closer look



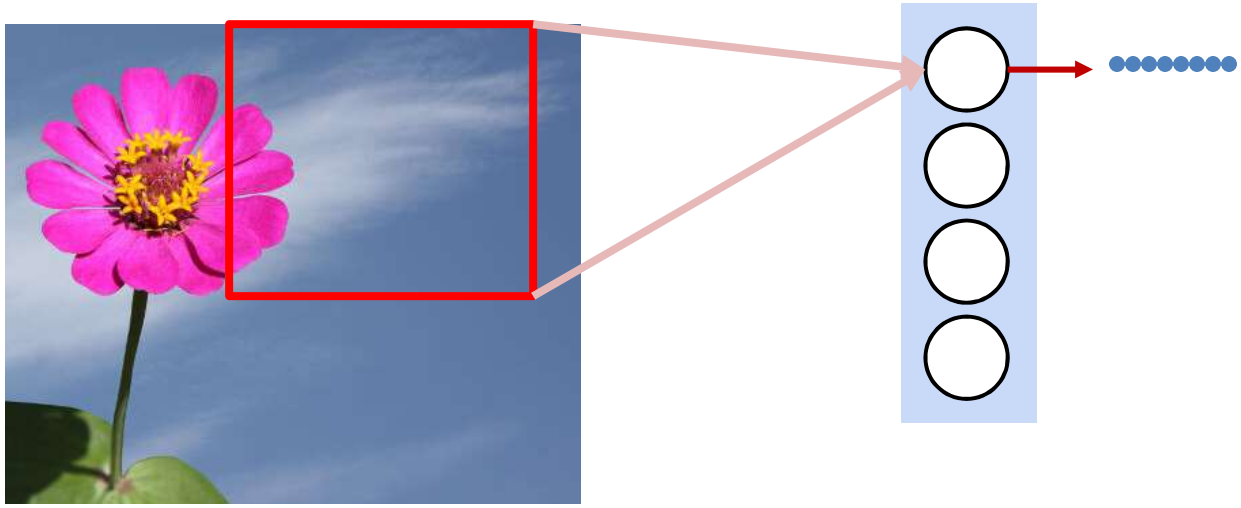
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture

Scanning: A closer look



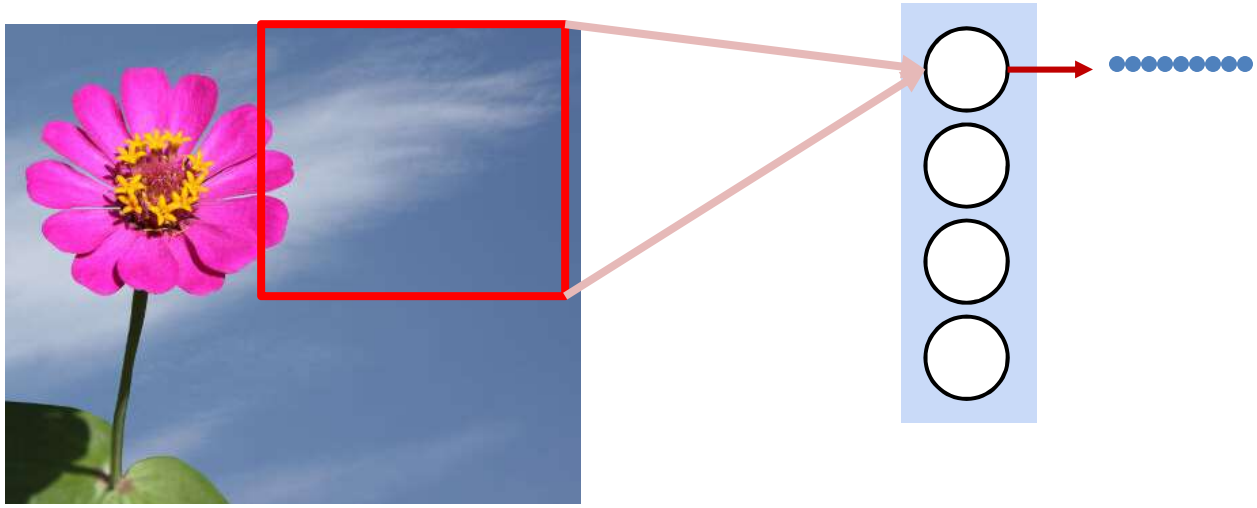
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture

Scanning: A closer look



- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture

Scanning: A closer look



- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture

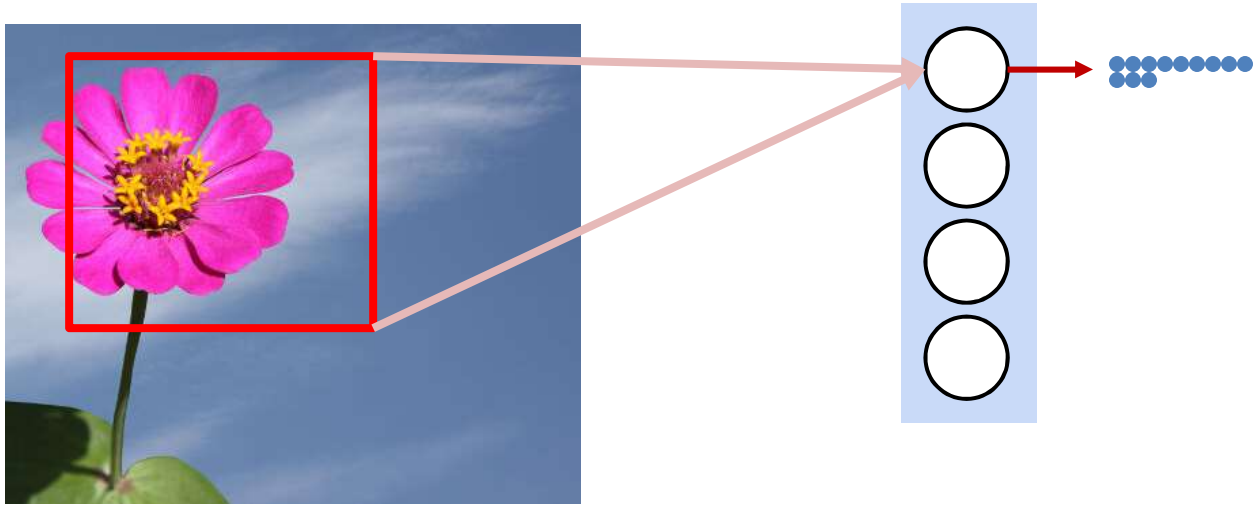
Scanning: A closer look

- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture

Scanning: A closer look

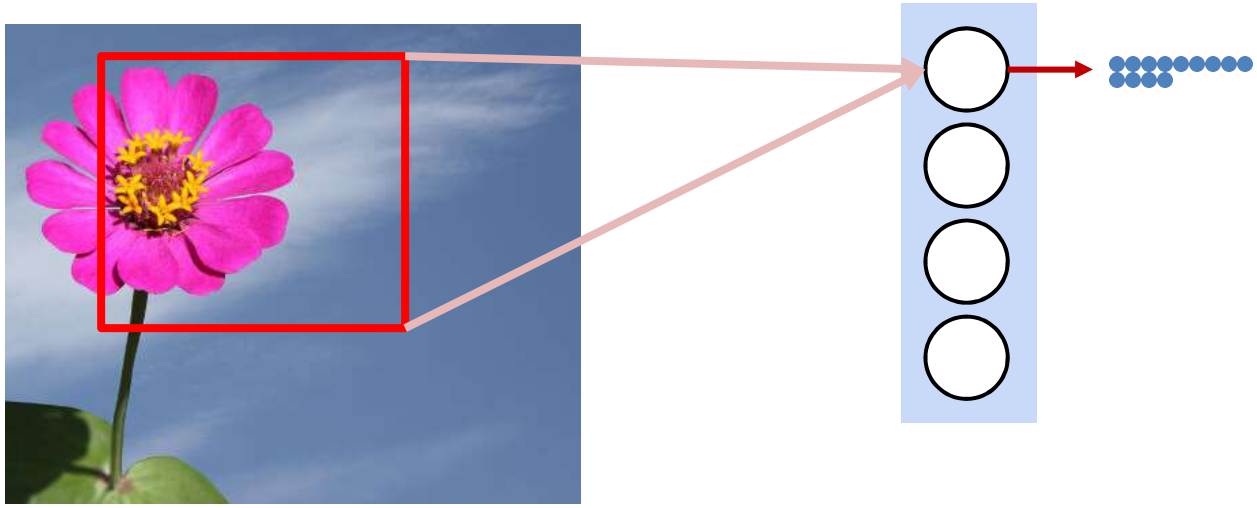
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture

Scanning: A closer look



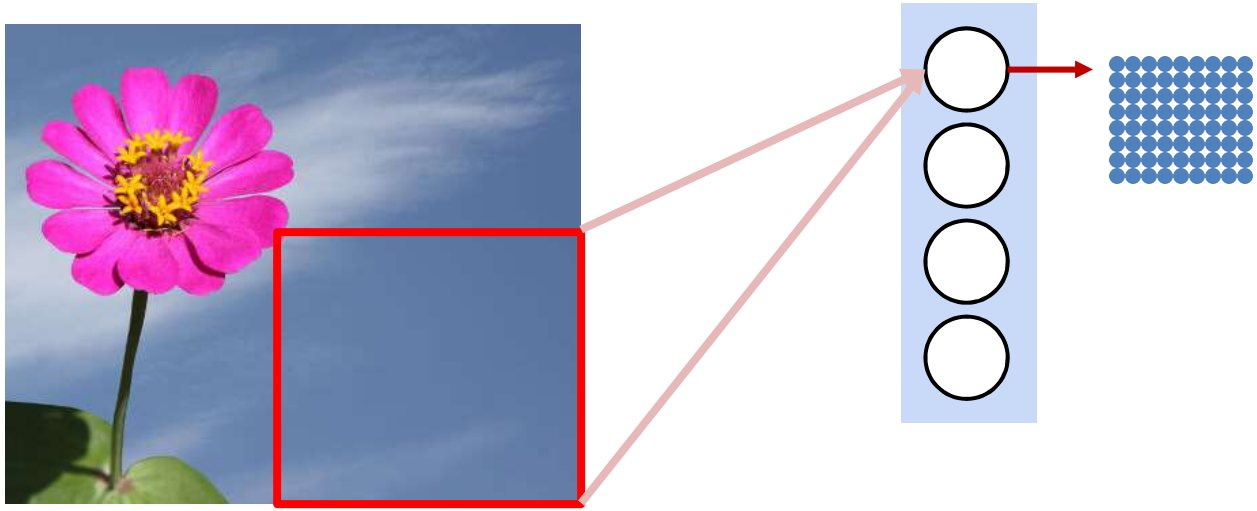
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture

Scanning: A closer look



- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture

Scanning: A closer look

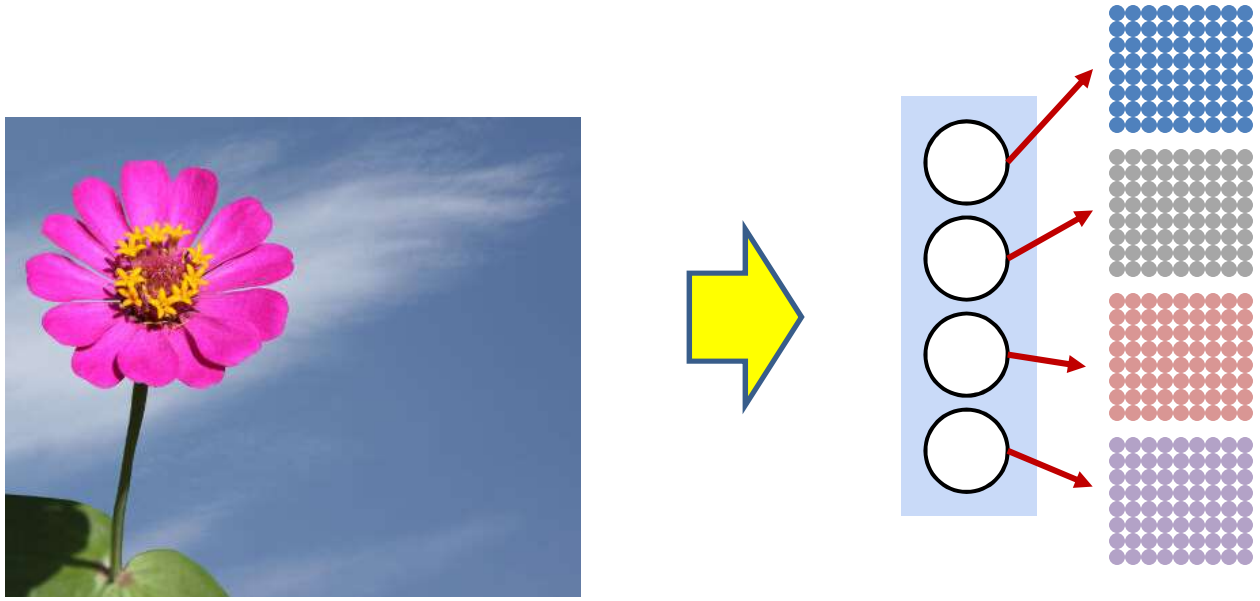


- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture
- Eventually, we can arrange the outputs from the response at the scanned positions into a rectangle that's proportional in size to the original picture

Scanning: A closer look

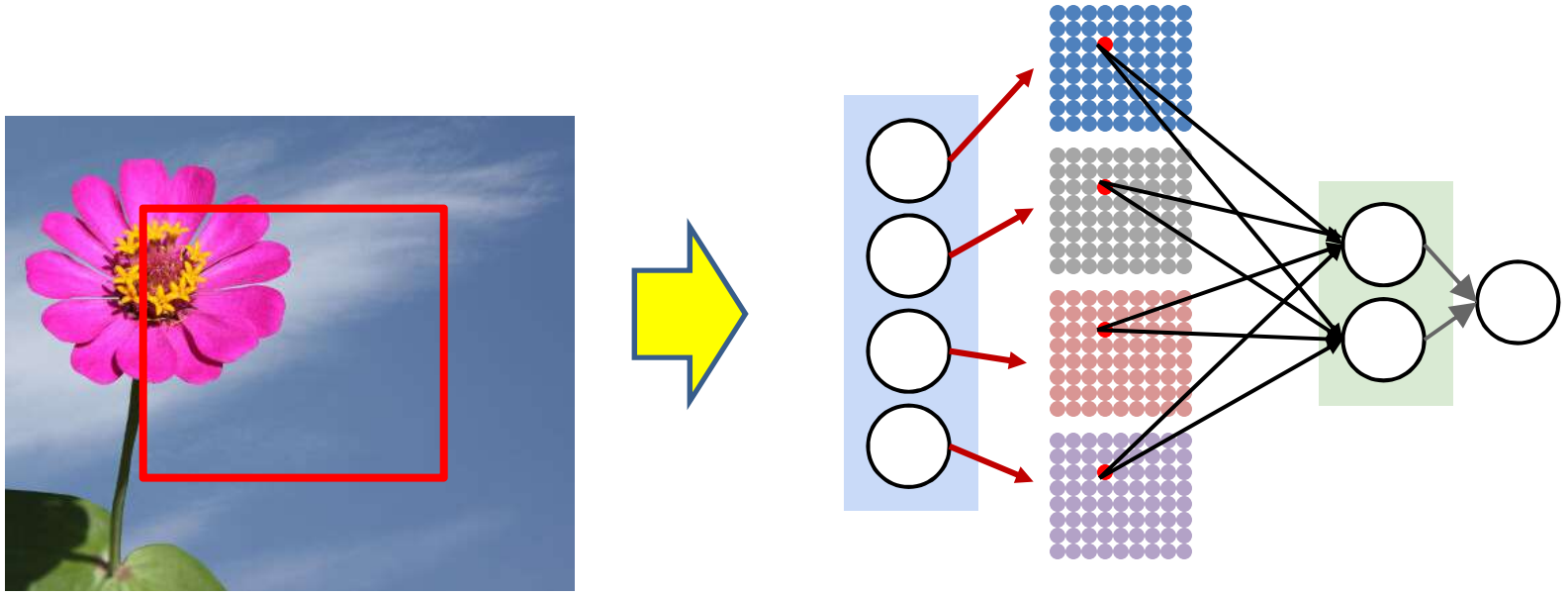
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture
- Eventually, we can arrange the outputs from the response at the scanned positions into a rectangle that's proportional in size to the original picture

Scanning: A closer look



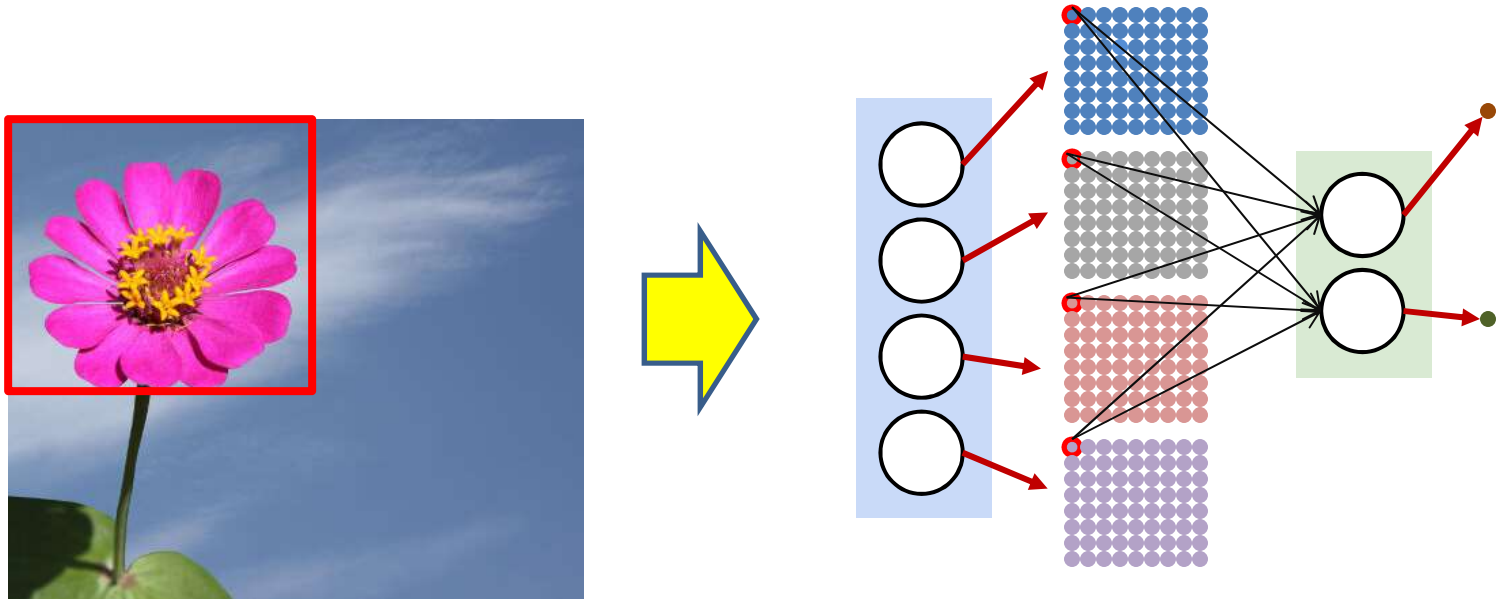
- Similarly, each first-layer perceptron's outputs from the scanned positions can be arranged as a rectangular pattern

Scanning: A closer look



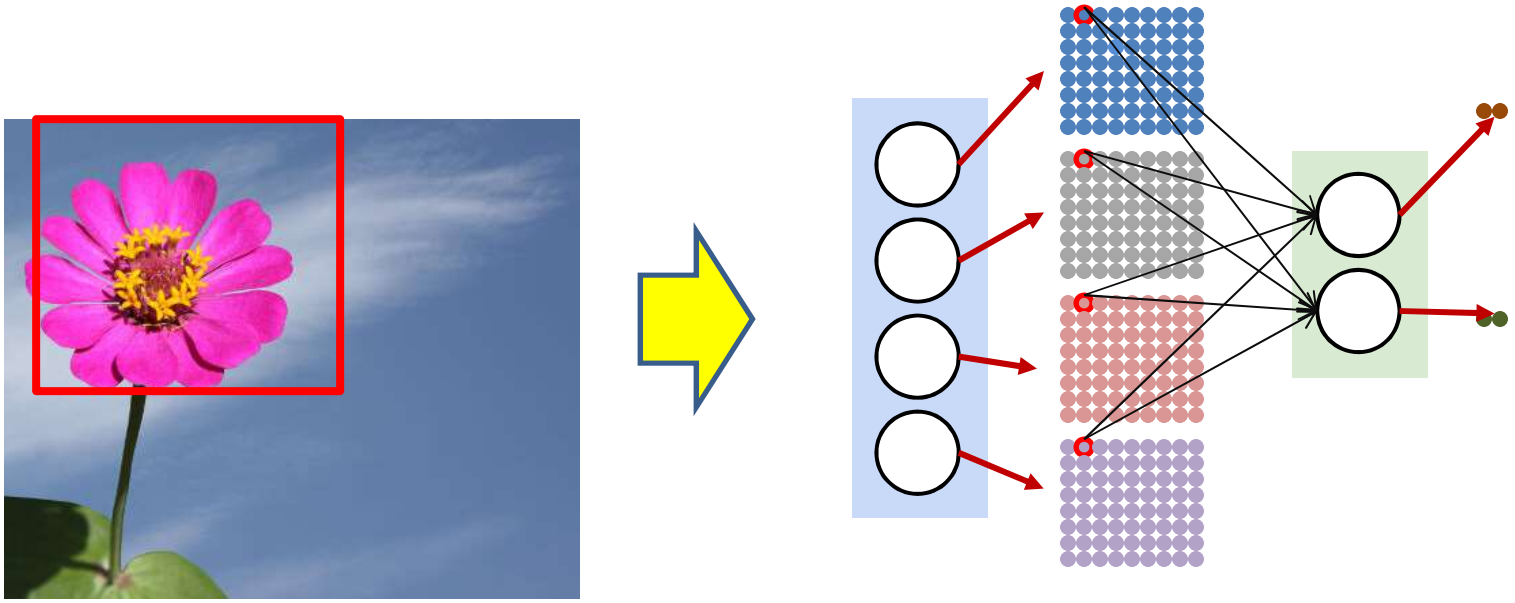
- To classify a specific “patch” in the image, we send the first level activations from the positions corresponding to that position to the next layer

Scanning: A closer look



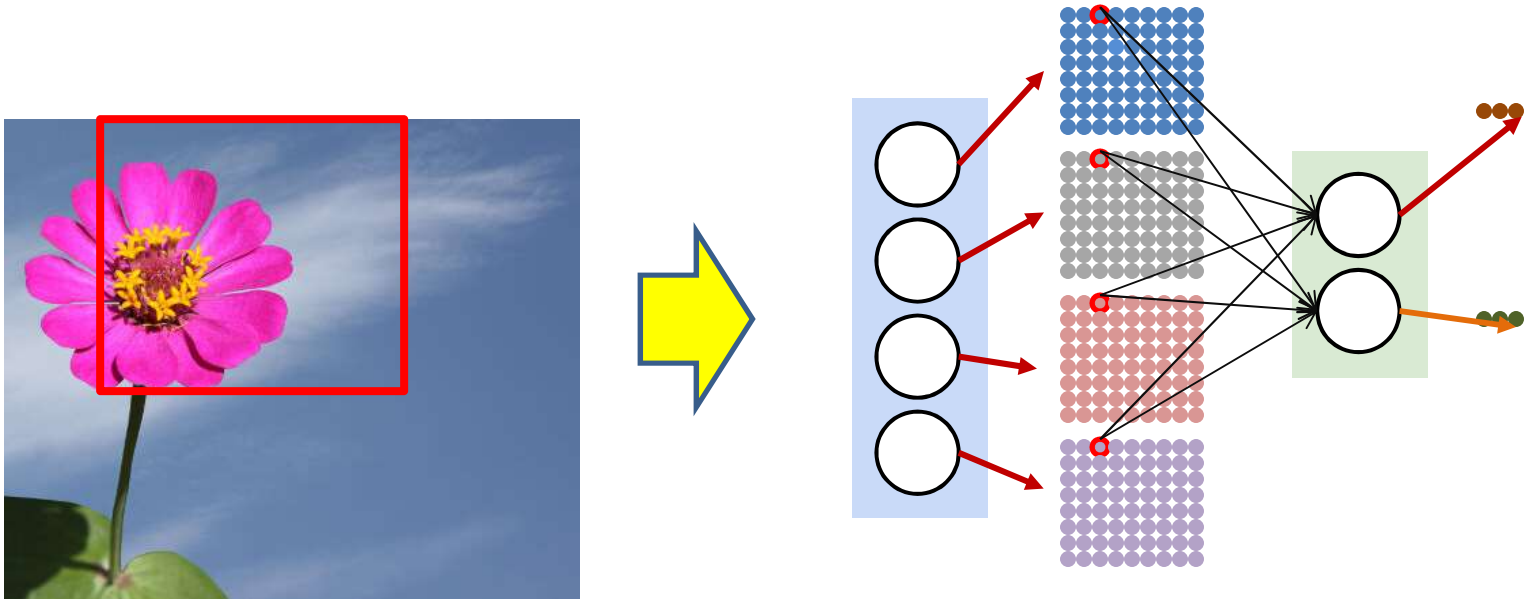
- We can recurse the logic
  - The second level neurons too are "**scanning**" the rectangular outputs of the first-level neurons
  - (Un)like the first level, they are jointly scanning *multiple* "pictures"
    - Each location in the output of the second level neuron considers the corresponding locations from the outputs of all the first-level neurons

Scanning: A closer look



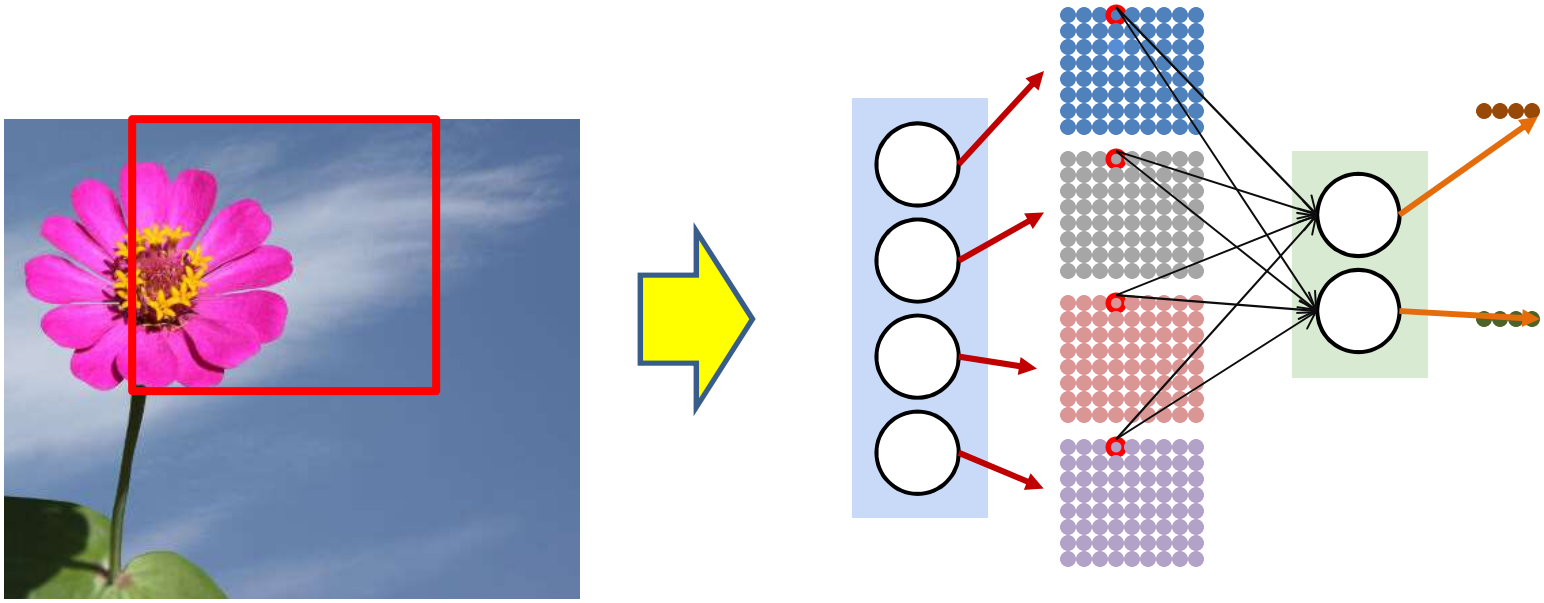
- We can recurse the logic
  - The second level neurons too are "**scanning**" the rectangular outputs of the first-level neurons
  - (Un)like the first level, they are jointly scanning *multiple* "pictures"
    - Each location in the output of the second level neuron considers the corresponding locations from the outputs of all the first-level neurons

Scanning: A closer look



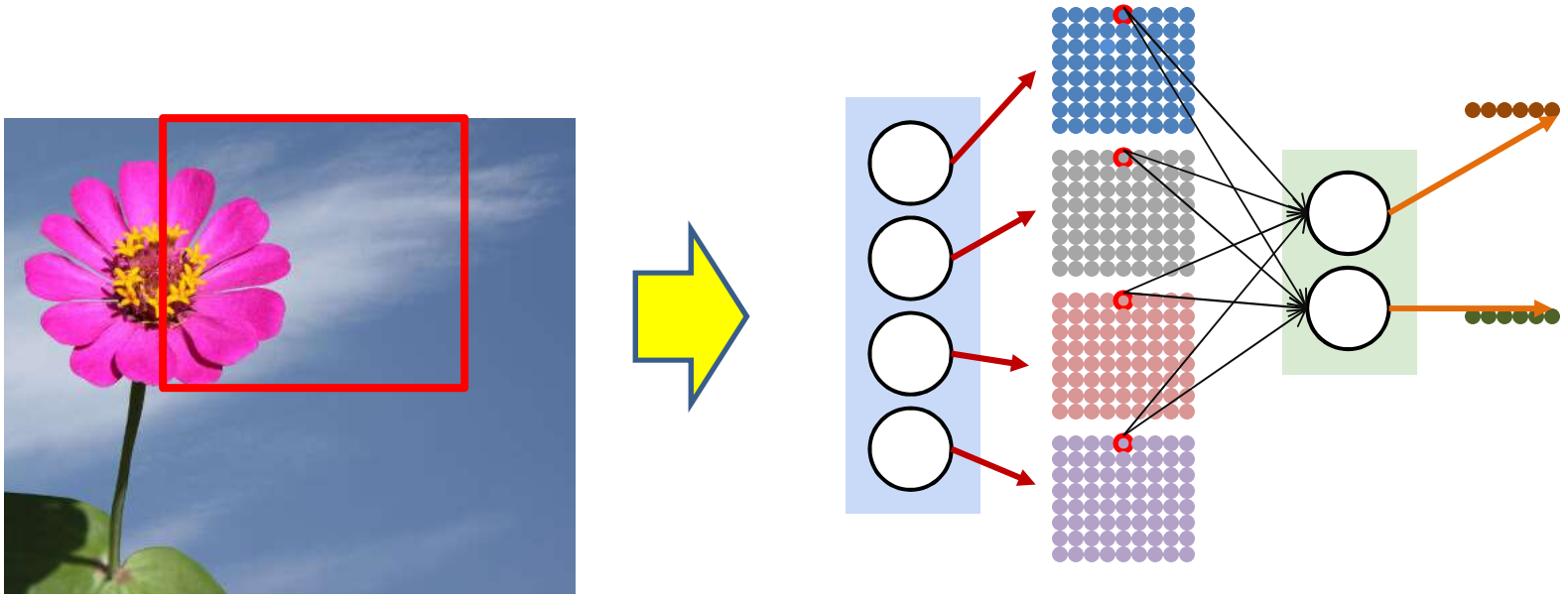
- We can recurse the logic
  - The second level neurons too are "**scanning**" the rectangular outputs of the first-level neurons
  - (Un)like the first level, they are jointly scanning *multiple* "pictures"
    - Each location in the output of the second level neuron considers the corresponding locations from the outputs of all the first-level neurons

Scanning: A closer look



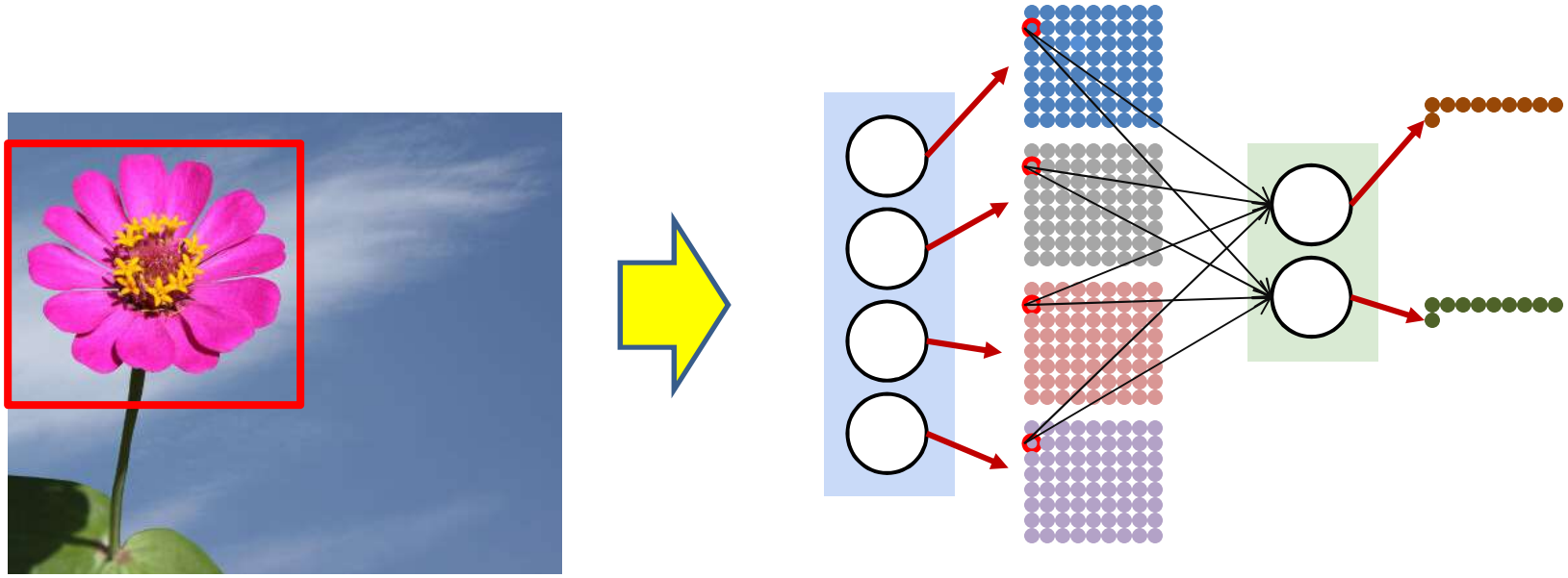
- We can recurse the logic
  - The second level neurons too are "**scanning**" the rectangular outputs of the first-level neurons
  - (Un)like the first level, they are jointly scanning *multiple* "pictures"
    - Each location in the output of the second level neuron considers the corresponding locations from the outputs of all the first-level neurons

Scanning: A closer look



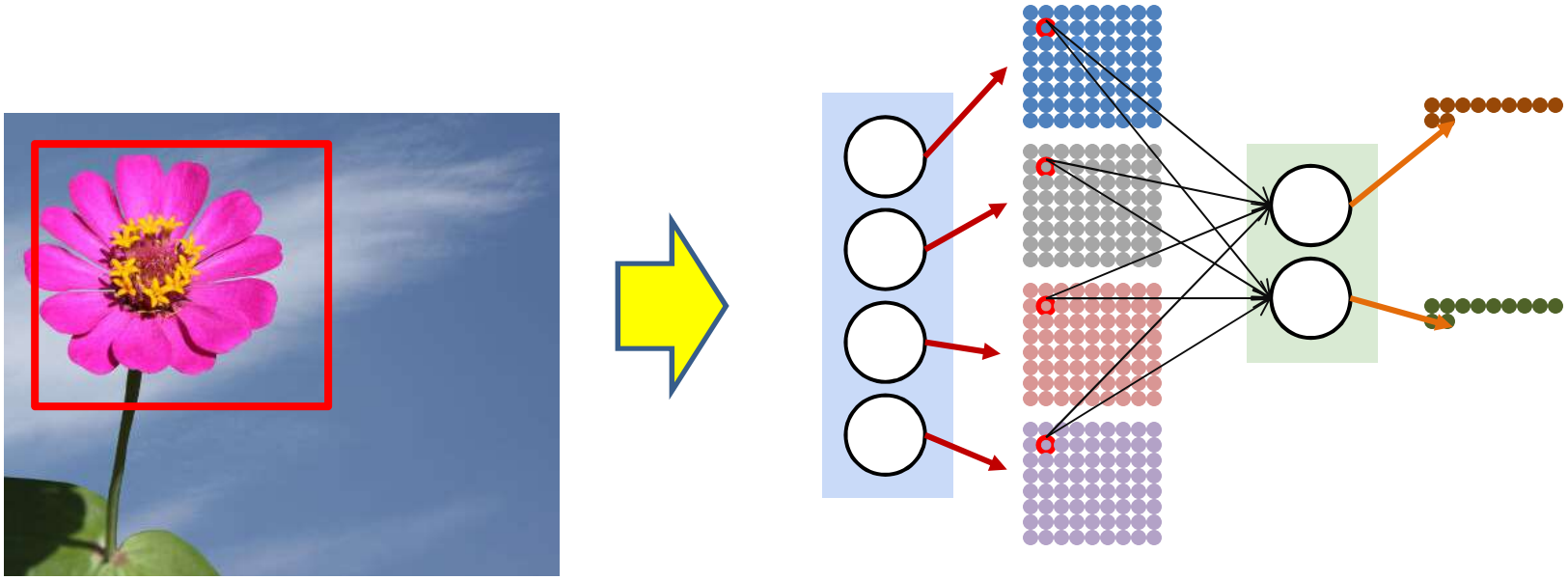
- We can recurse the logic
  - The second level neurons too are "**scanning**" the rectangular outputs of the first-level neurons
  - (Un)like the first level, they are jointly scanning *multiple* "pictures"
    - Each location in the output of the second level neuron considers the corresponding locations from the outputs of all the first-level neurons

Scanning: A closer look



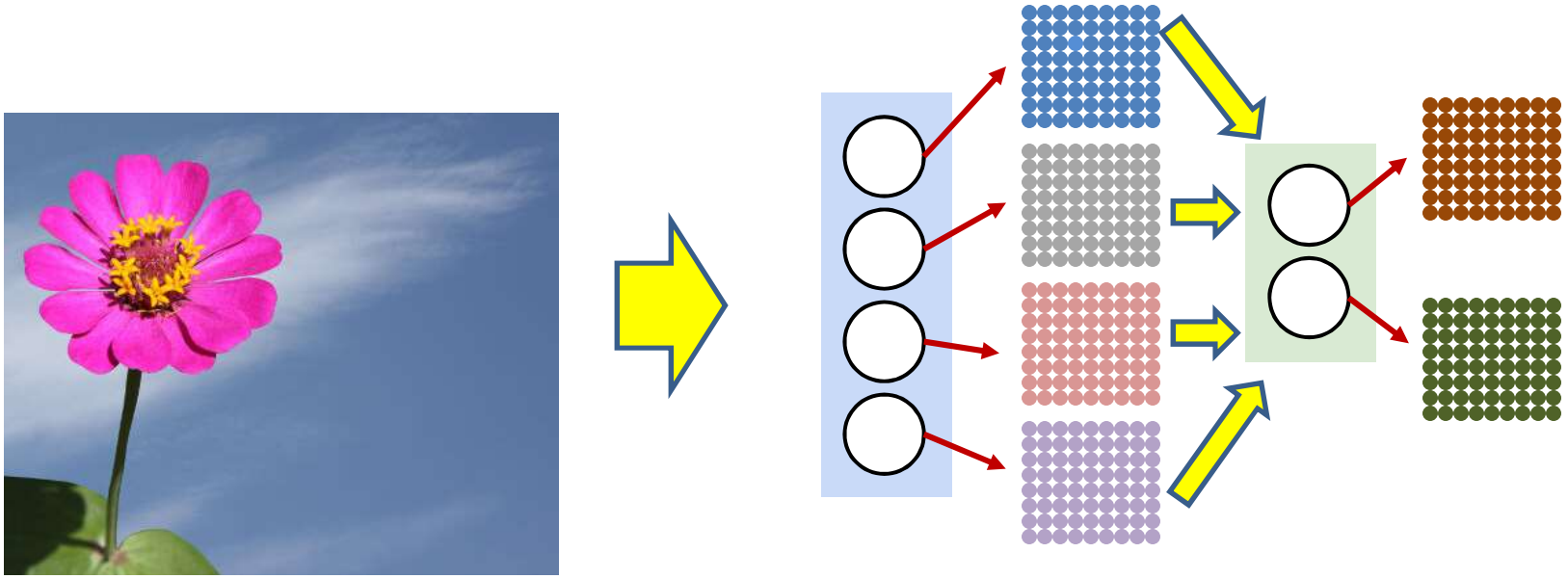
- We can recurse the logic
  - The second level neurons too are "**scanning**" the rectangular outputs of the first-level neurons
  - (Un)like the first level, they are jointly scanning *multiple* "pictures"
    - Each location in the output of the second level neuron considers the corresponding locations from the outputs of all the first-level neurons

Scanning: A closer look



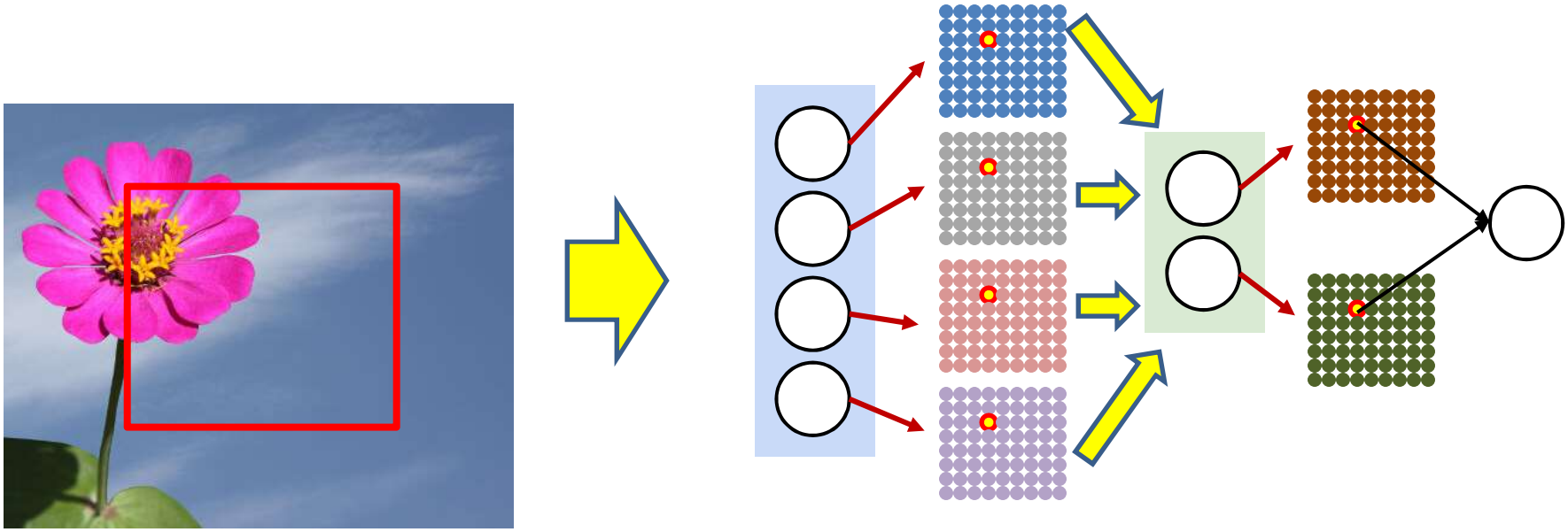
- We can recurse the logic
  - The second level neurons too are "**scanning**" the rectangular outputs of the first-level neurons
  - (Un)like the first level, they are jointly scanning *multiple* "pictures"
    - Each location in the output of the second level neuron considers the corresponding locations from the outputs of all the first-level neurons

Scanning: A closer look



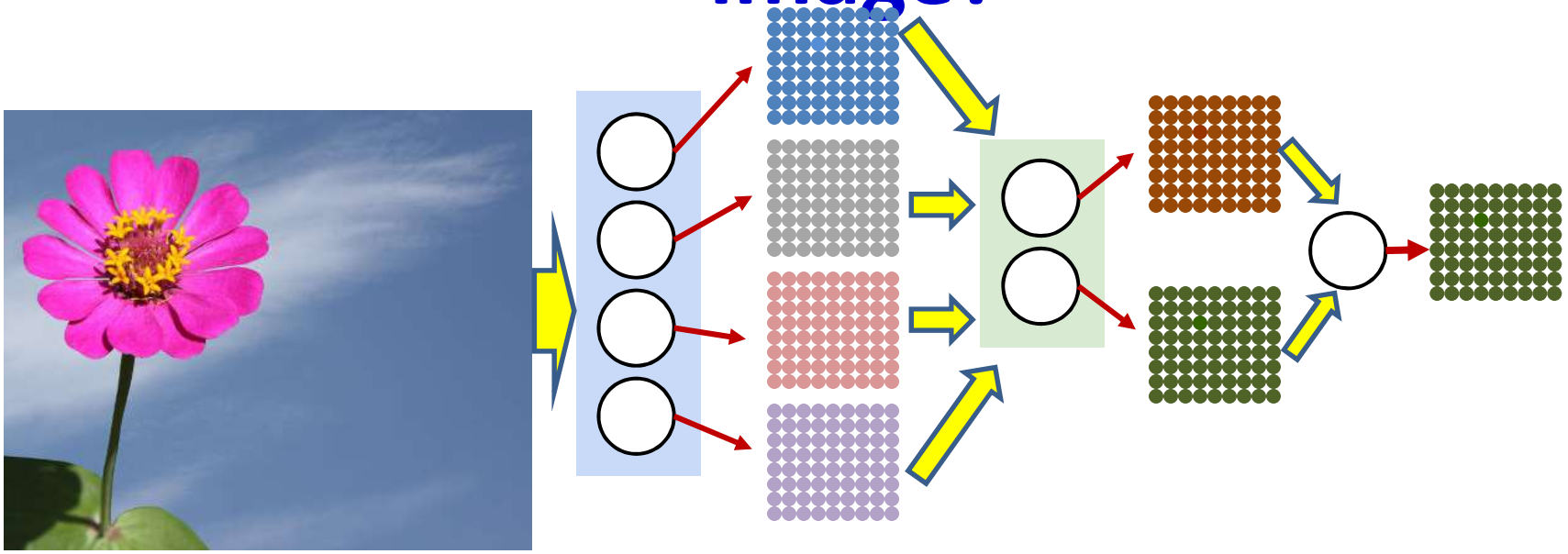
- We can recurse the logic
  - The second level neurons too are "**scanning**" the rectangular outputs of the first-level neurons
  - (Un)like the first level, they are jointly scanning *multiple* "pictures"
    - Each location in the output of the second level neuron considers the corresponding locations from the outputs of all the first-level neurons

Scanning: A closer look



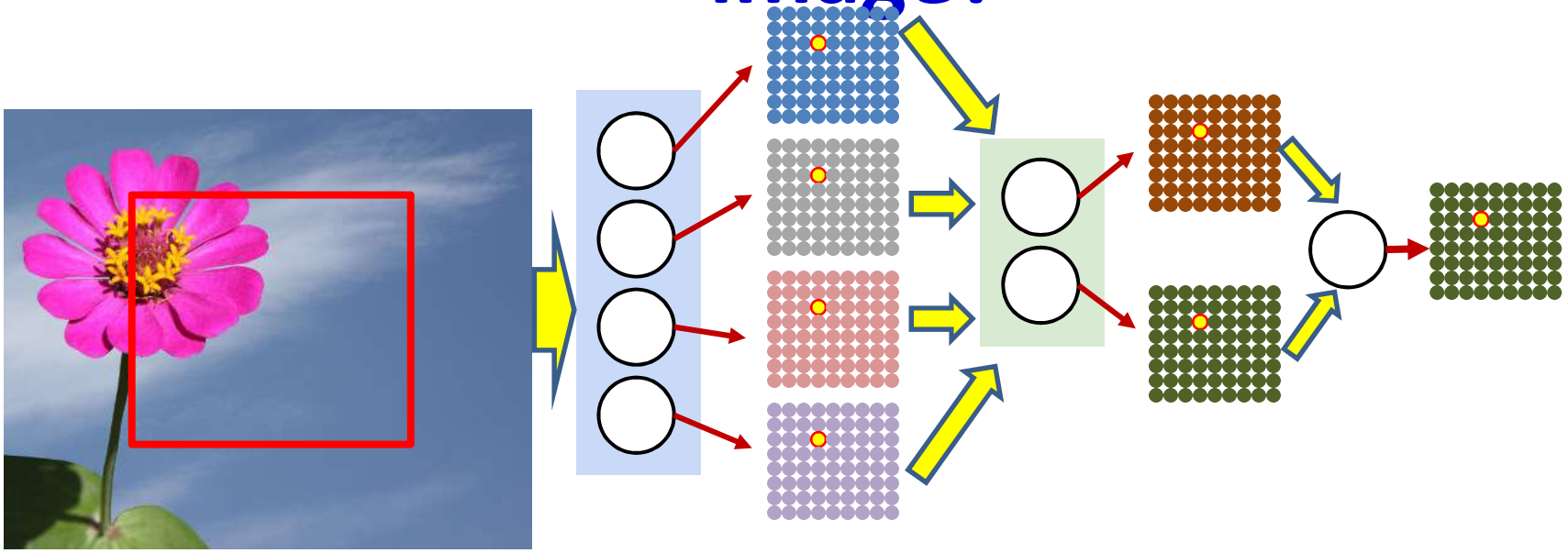
- To detect a picture ***at any location*** in the original image, the output layer must consider the corresponding outputs of the last hidden layer

Detecting a picture anywhere in the image?



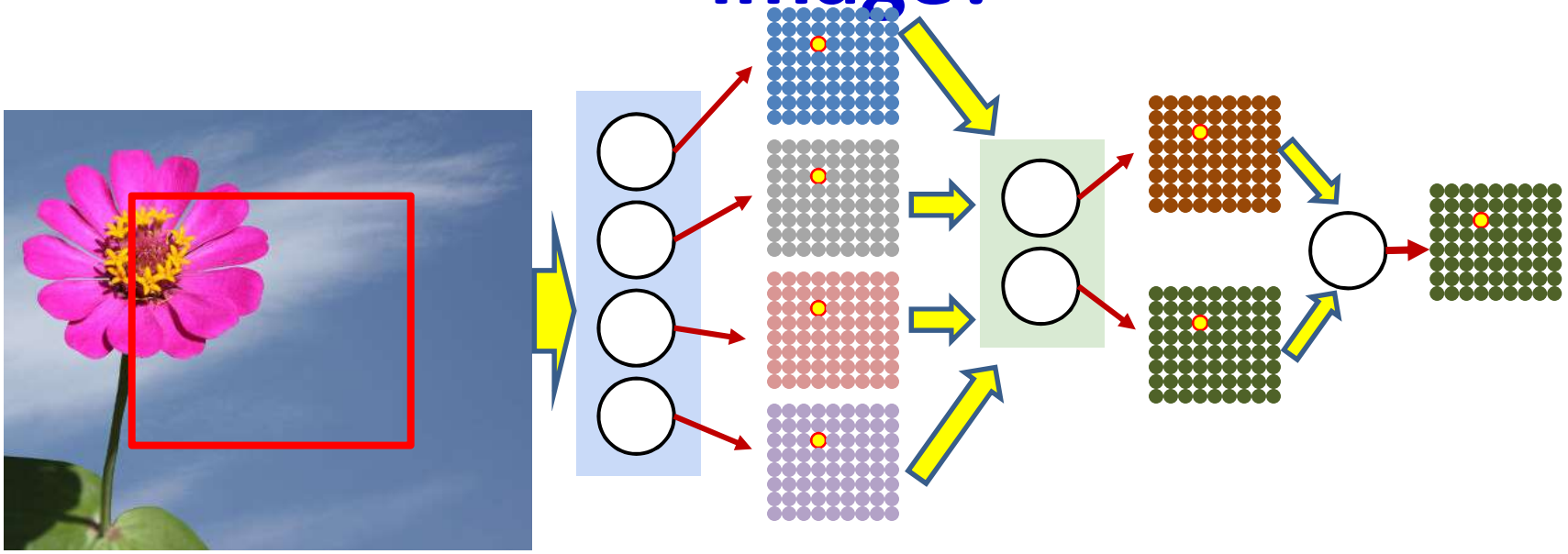
- Recursing the logic, we can create a map for the neurons in the next layer as well
  - The map is a flower detector for each location of the original image

Detecting a picture anywhere in the image?



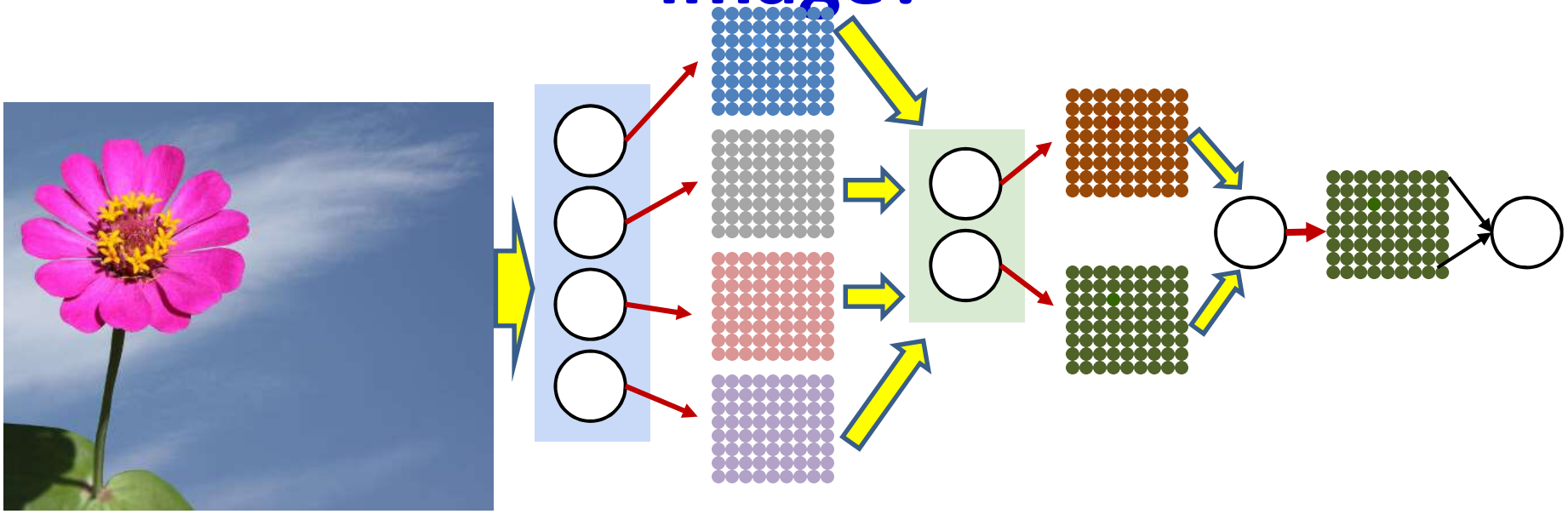
- To detect a picture ***at any location*** in the original image, the output layer must consider the corresponding output of the last hidden layer

Detecting a picture anywhere in the image?



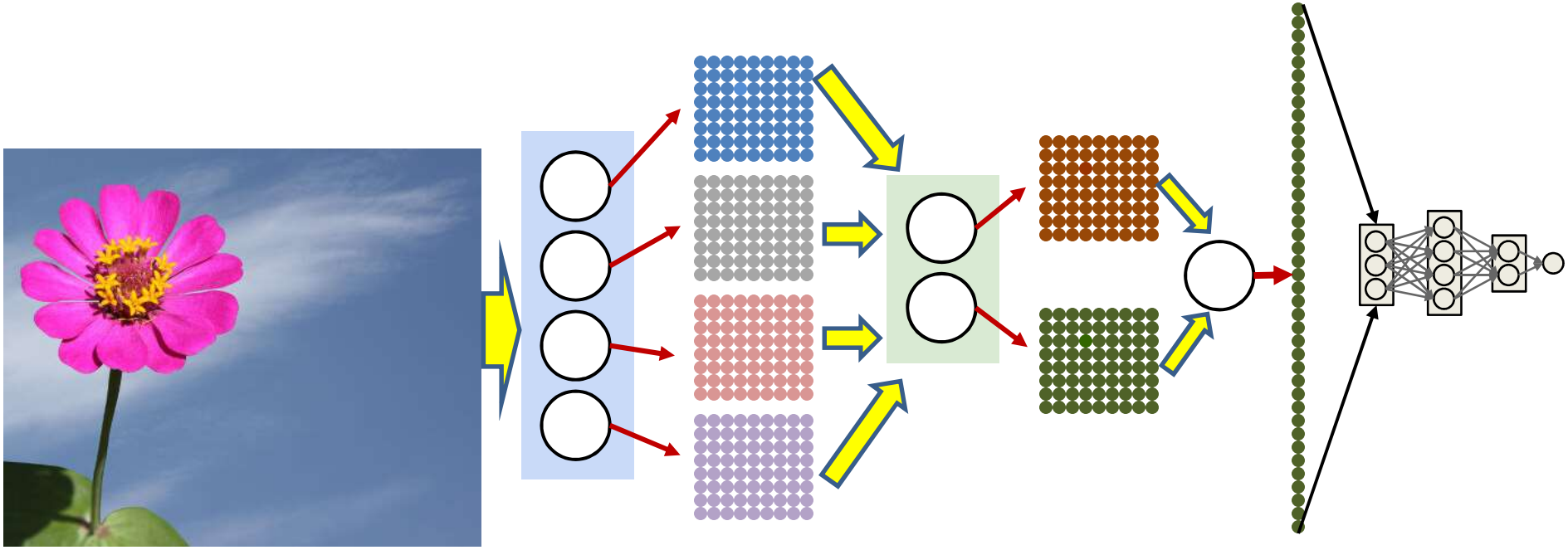
- To detect a picture ***at any location*** in the original image, the output layer must consider the corresponding output of the last hidden layer
- Actual problem? Is there a flower in the image
  - Not “detect the location of a flower”

Detecting a picture anywhere in the image?



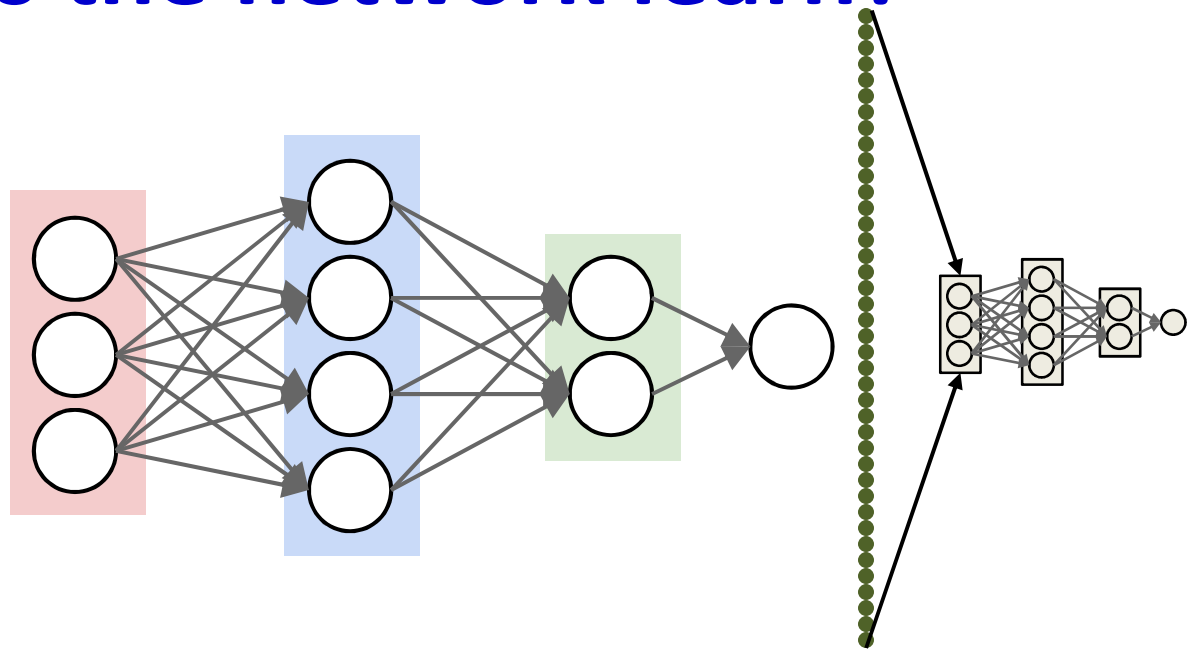
- Is there a flower in the picture?
- The output of the almost-last layer is also a grid/picture
- The entire grid can be sent into a final neuron that performs a logical “OR” to detect a flower in the full picture
  - Finds the *max* output from all the positions
  - Or a softmax, or a full MLP..

Detecting a picture in the image



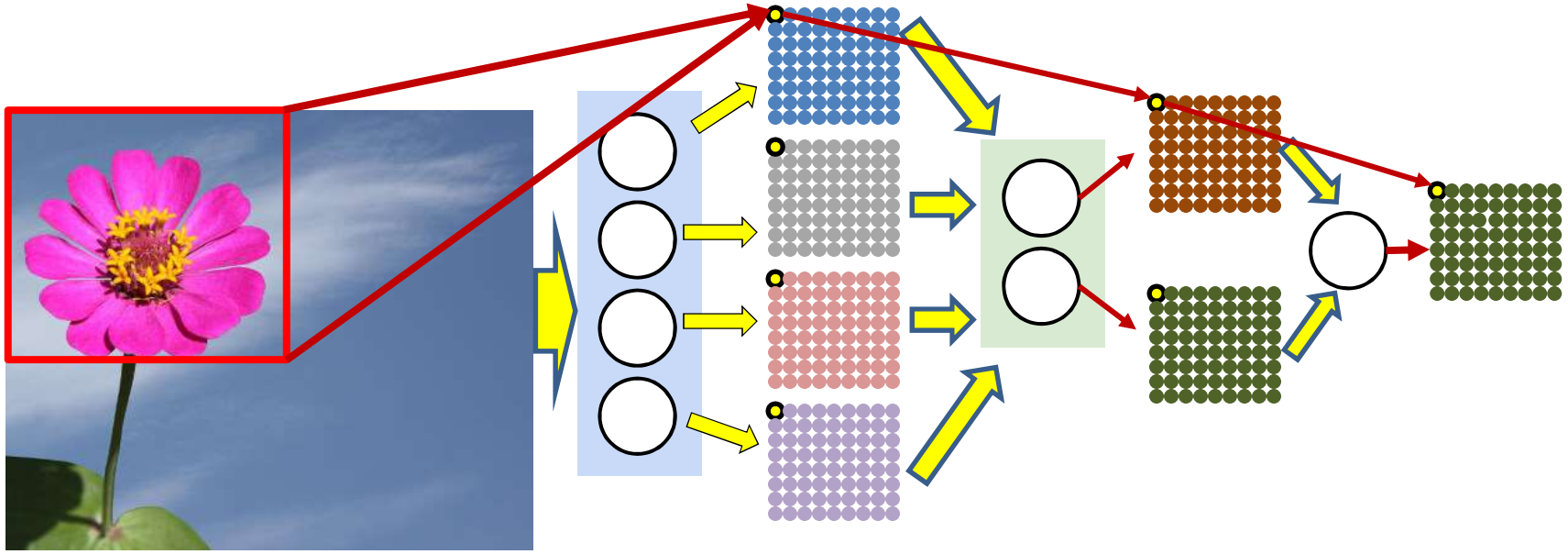
- Redrawing the final layer
  - "Flatten" the output of the neurons into a single block, since the arrangement is no longer important
  - Pass that through a max/softmax/MLP

Returning to our problem: What does the network learn?



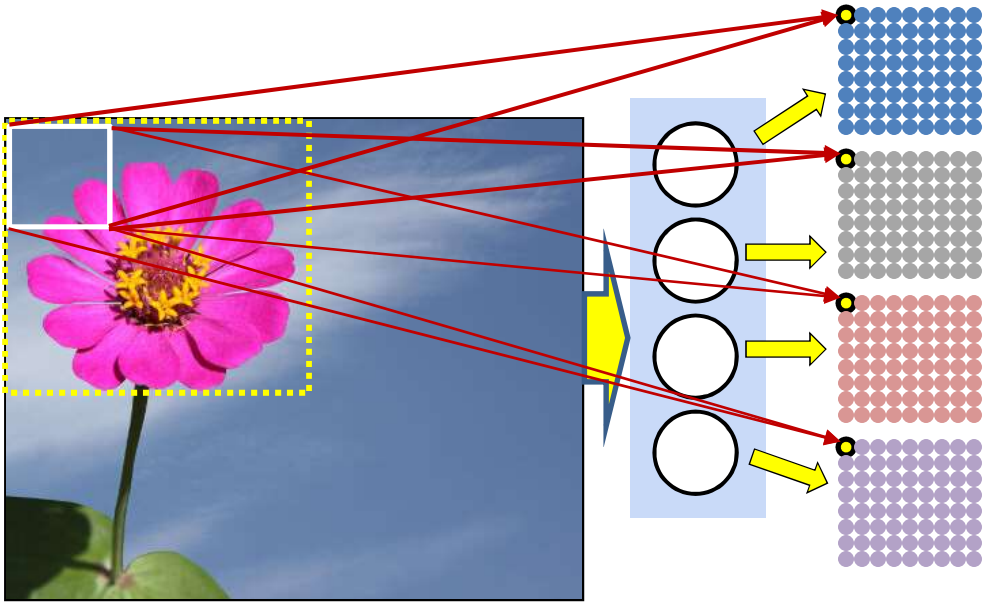
- The entire MLP looks for a flower-like pattern at each location

The behavior of the layers



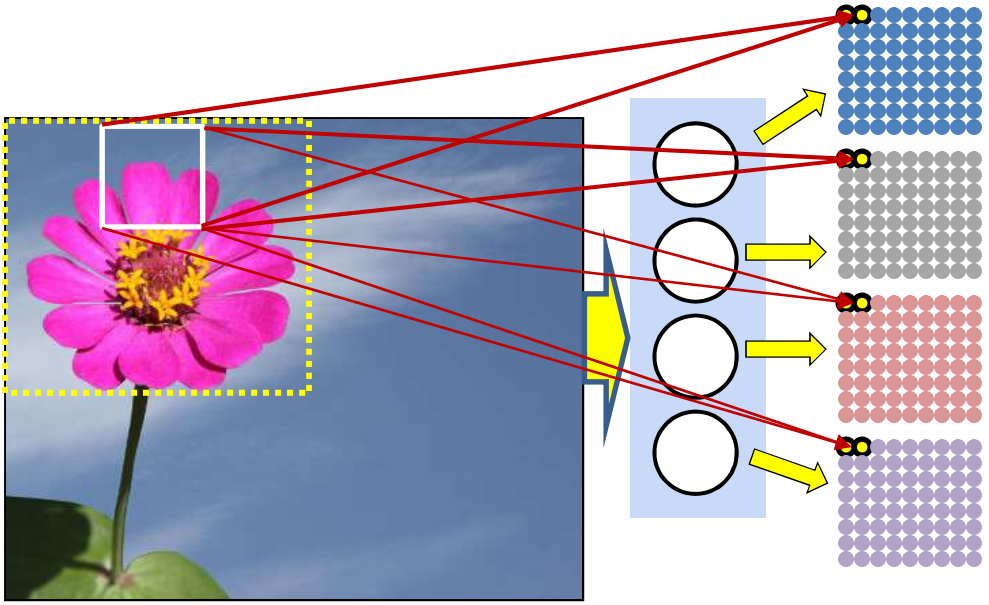
- The first layer neurons “look” at the entire “block” to extract block-level features
  - Subsequent layers only perform classification over these block-level features
- **The first layer neurons is responsible for evaluating the entire *block* of pixels**
  - **Subsequent layers only look at a *single* pixel in their input maps**

Distributing the scan



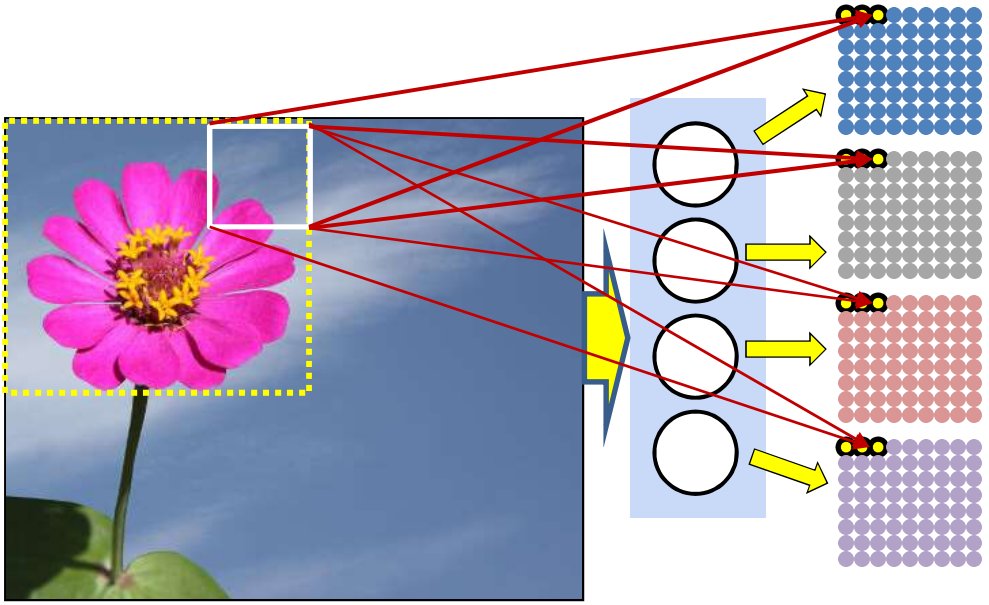
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels

Distributing the scan



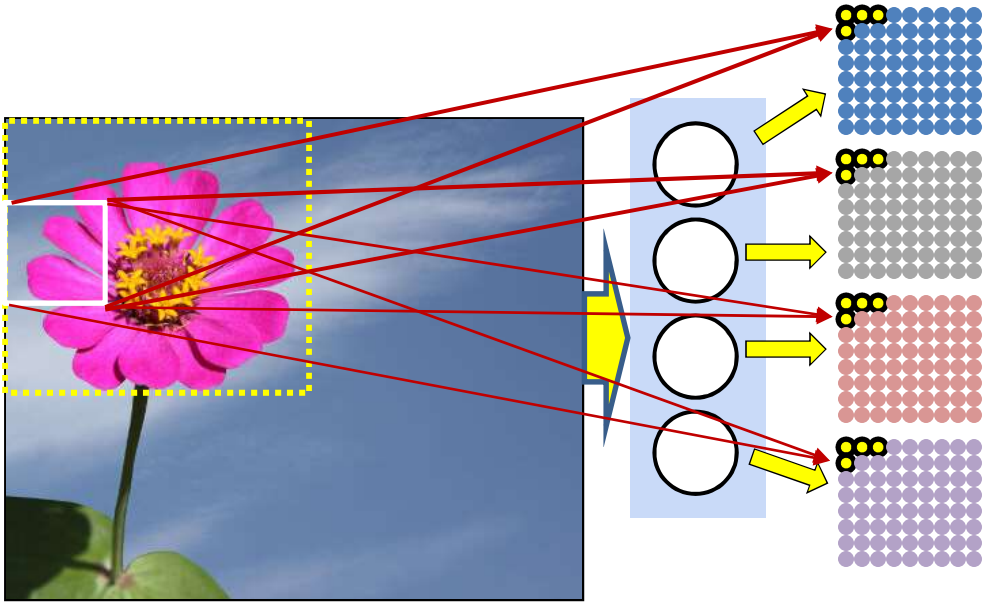
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels

Distributing the scan



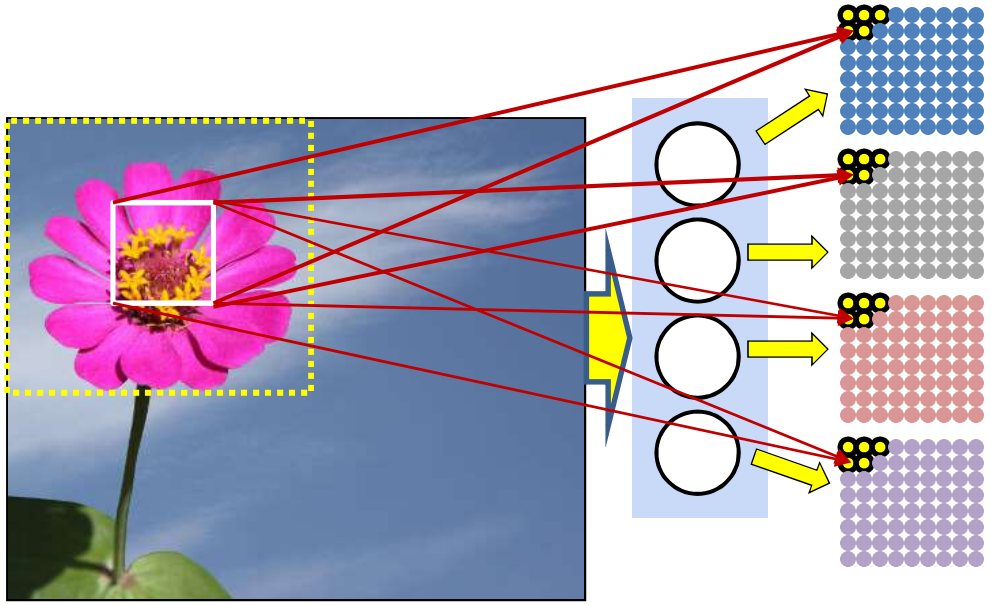
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels

Distributing the scan



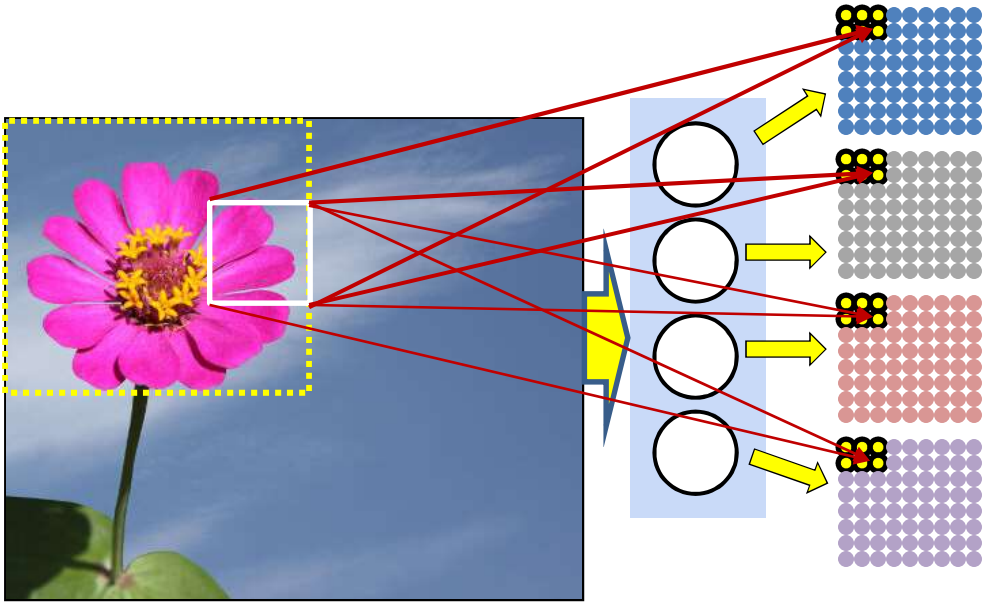
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels

Distributing the scan



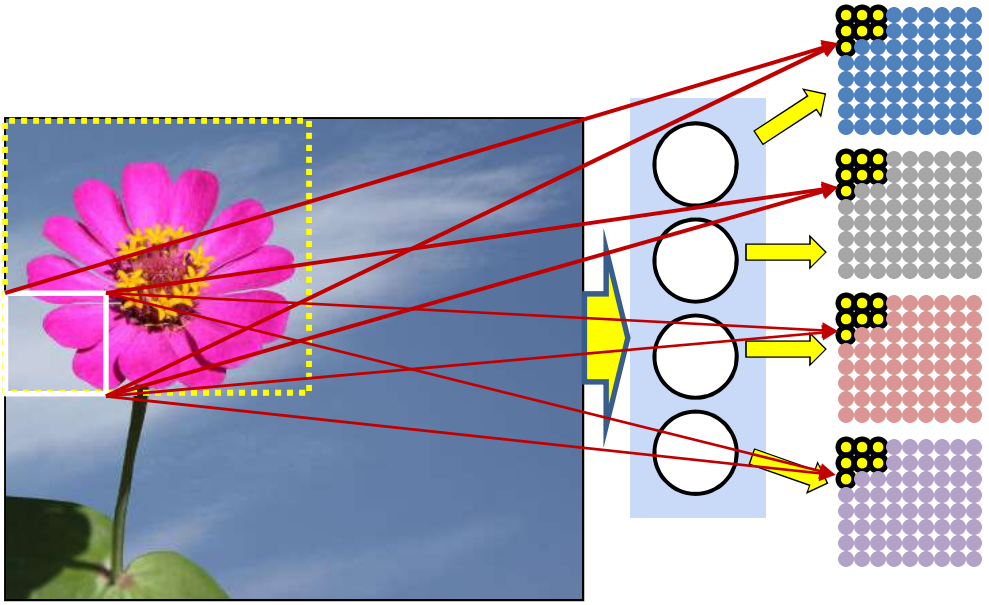
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels

Distributing the scan



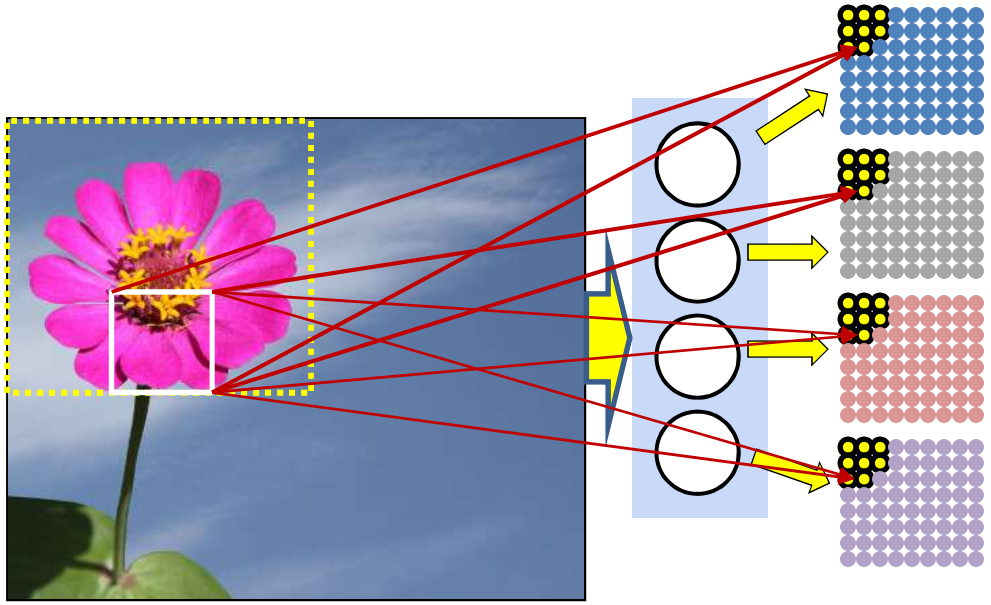
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels

Distributing the scan



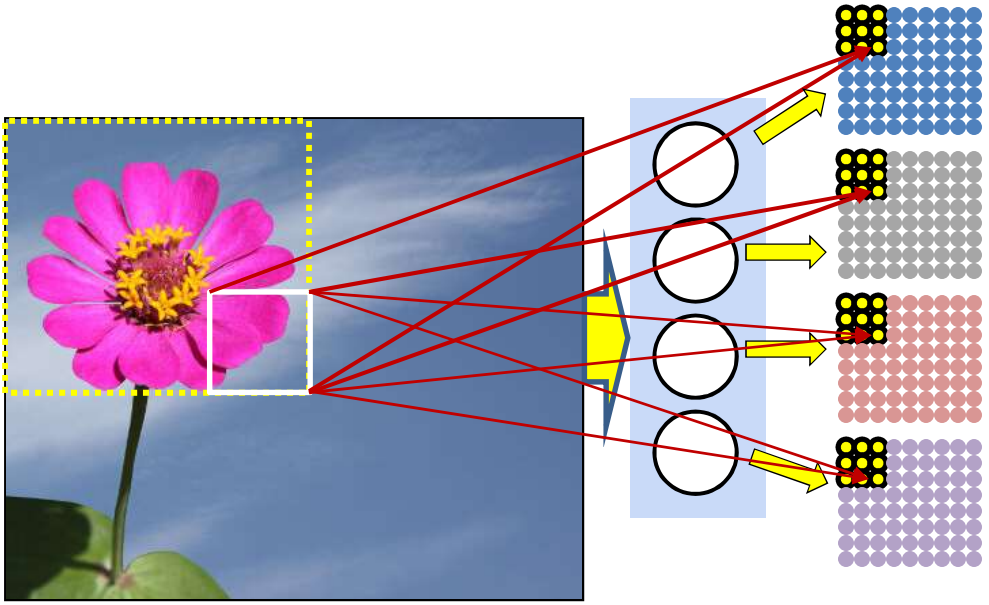
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels

Distributing the scan



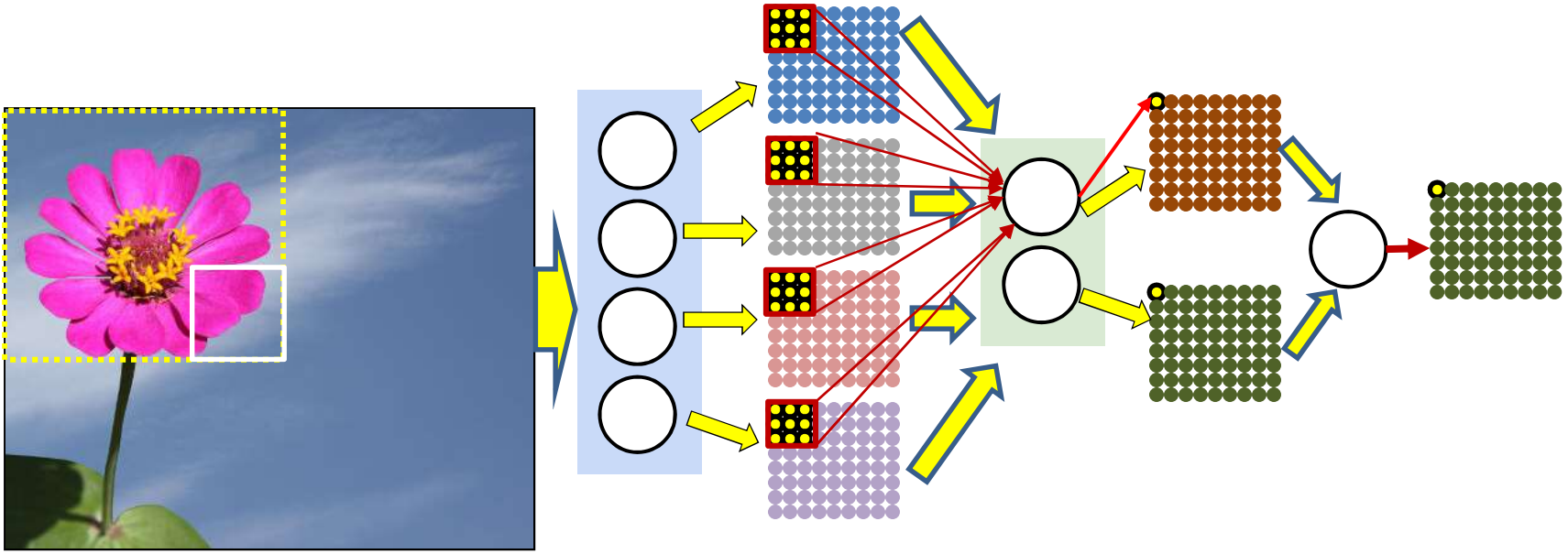
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels

Distributing the scan



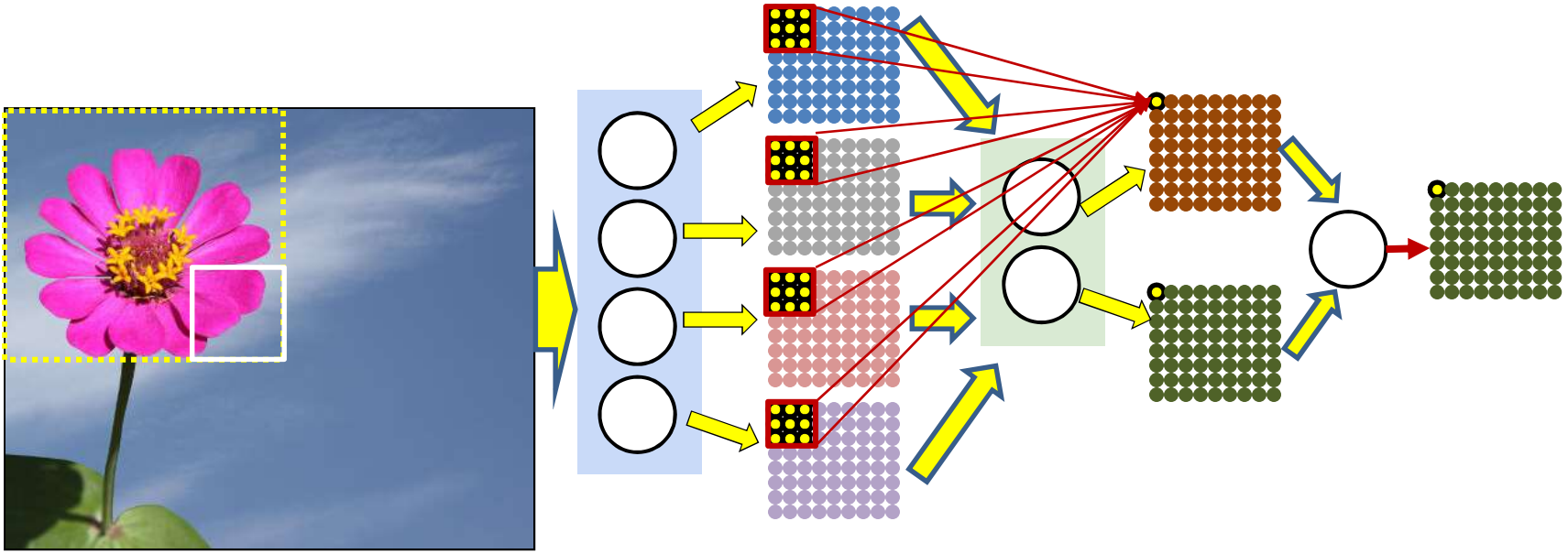
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels

Distributing the scan



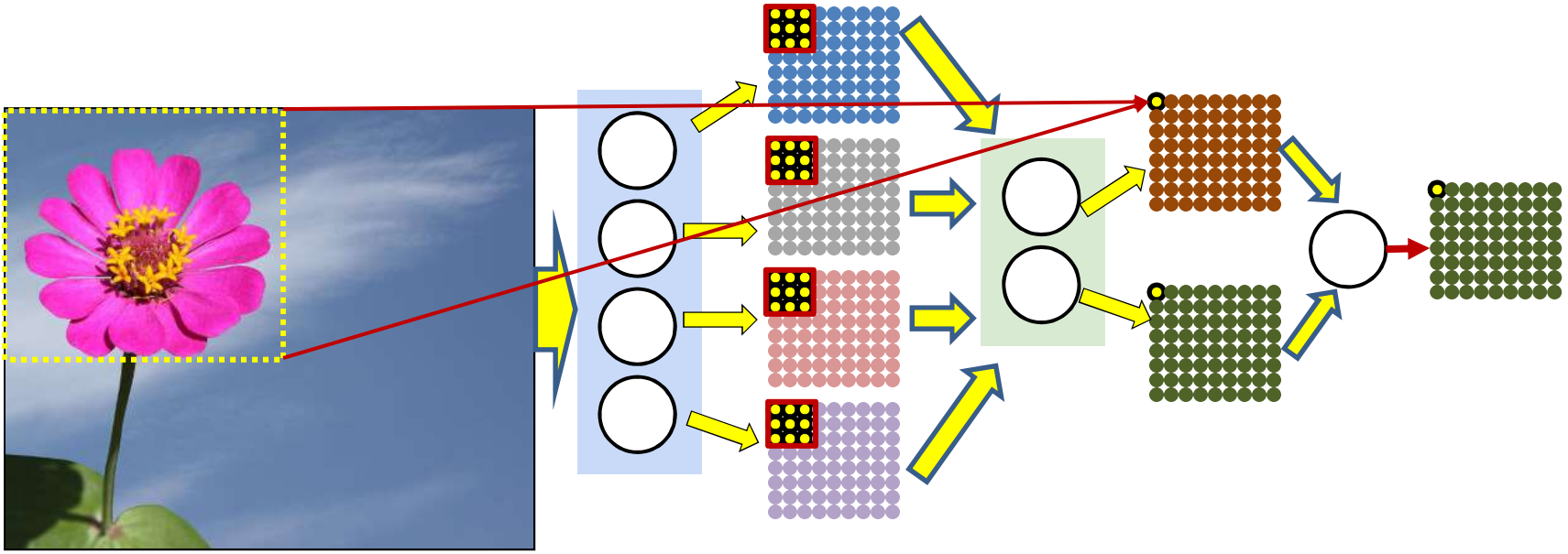
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates blocks of outputs from the first layer

Distributing the scan



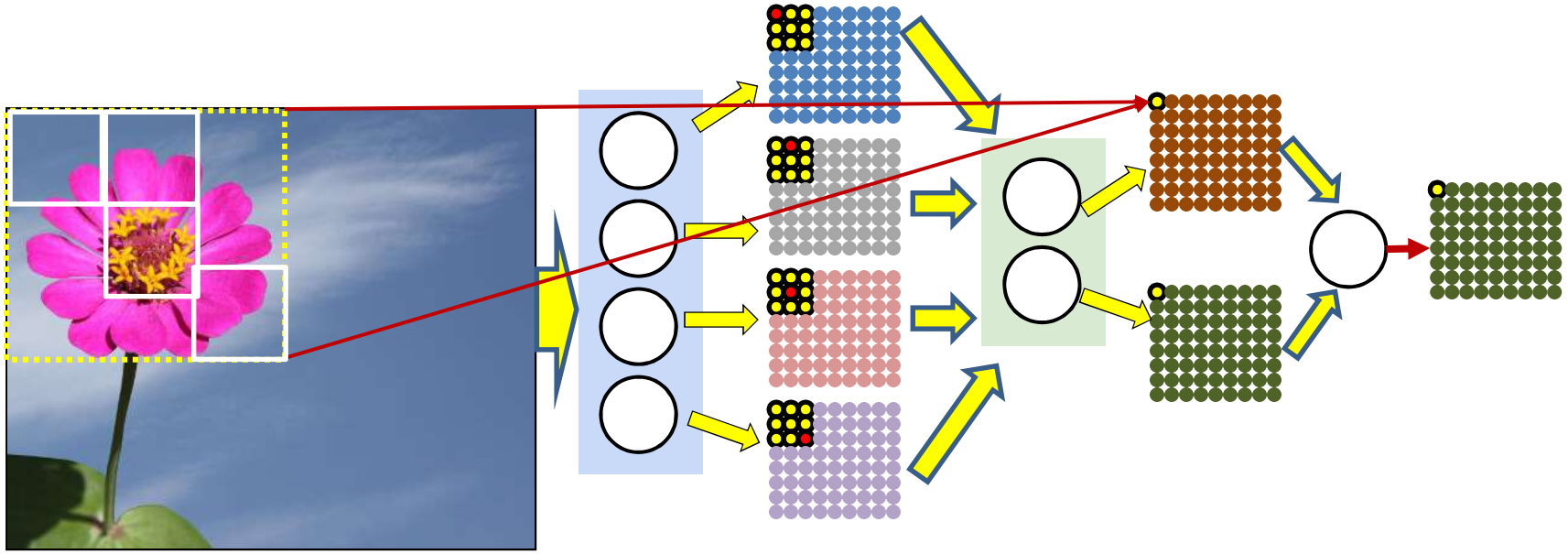
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates blocks of outputs from the first layer

Distributing the scan



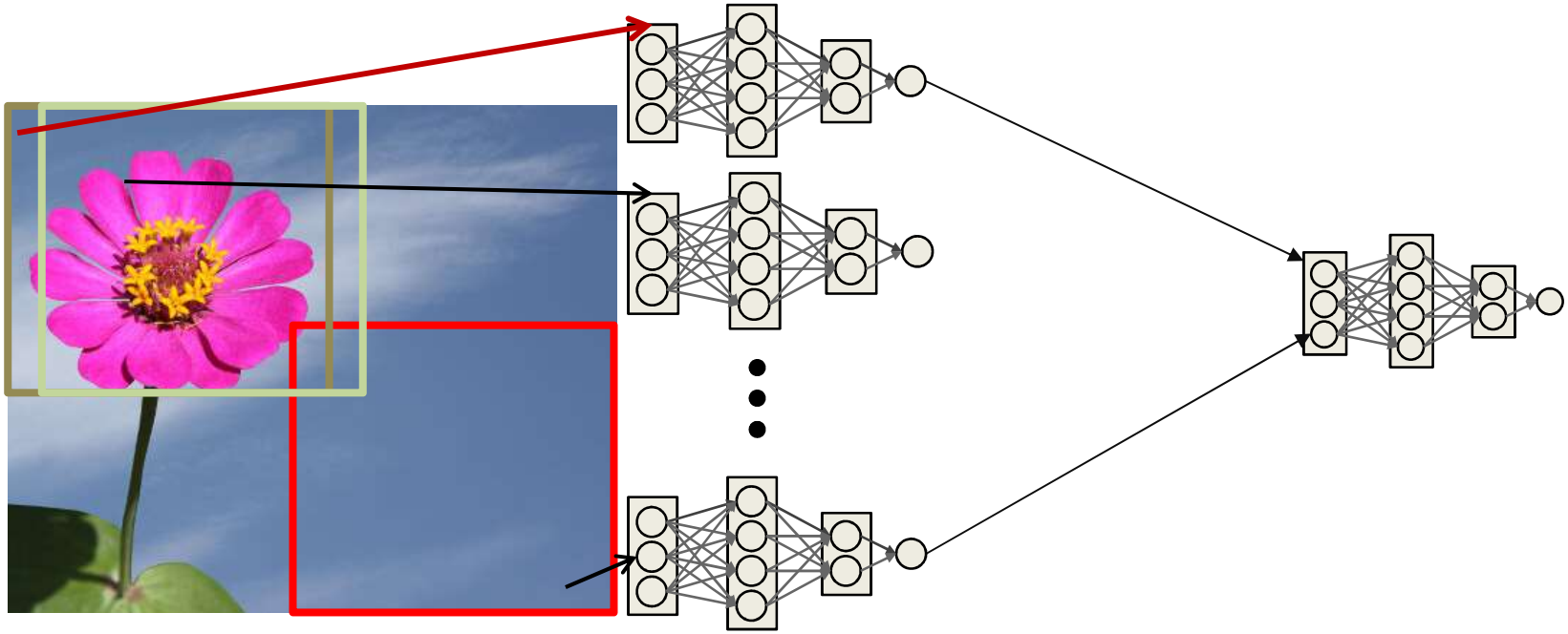
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates blocks of outputs from the first layer
  - This effectively evaluates the larger block of the original image

Distributing the scan



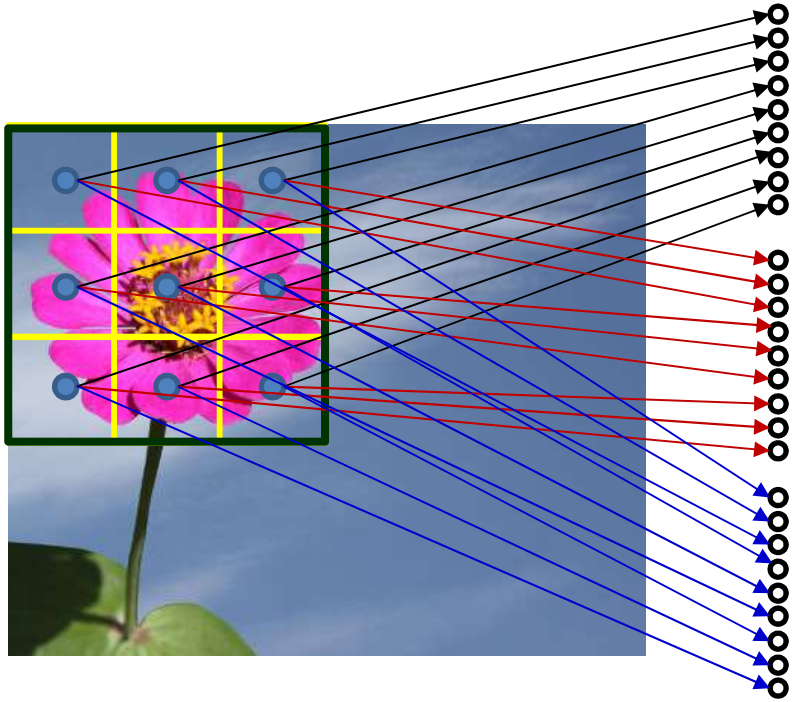
- The higher layer implicitly learns the *arrangement* of sub patterns that represents the larger pattern (the flower in this case)

This is *still* just scanning with a shared parameter network



- With a minor modification…

This is *still* just scanning with a shared parameter network

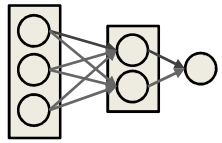


Each arrow represents an entire set of weights over the smaller cell

The pattern of weights going out of any cell is identical to that from any other cell.

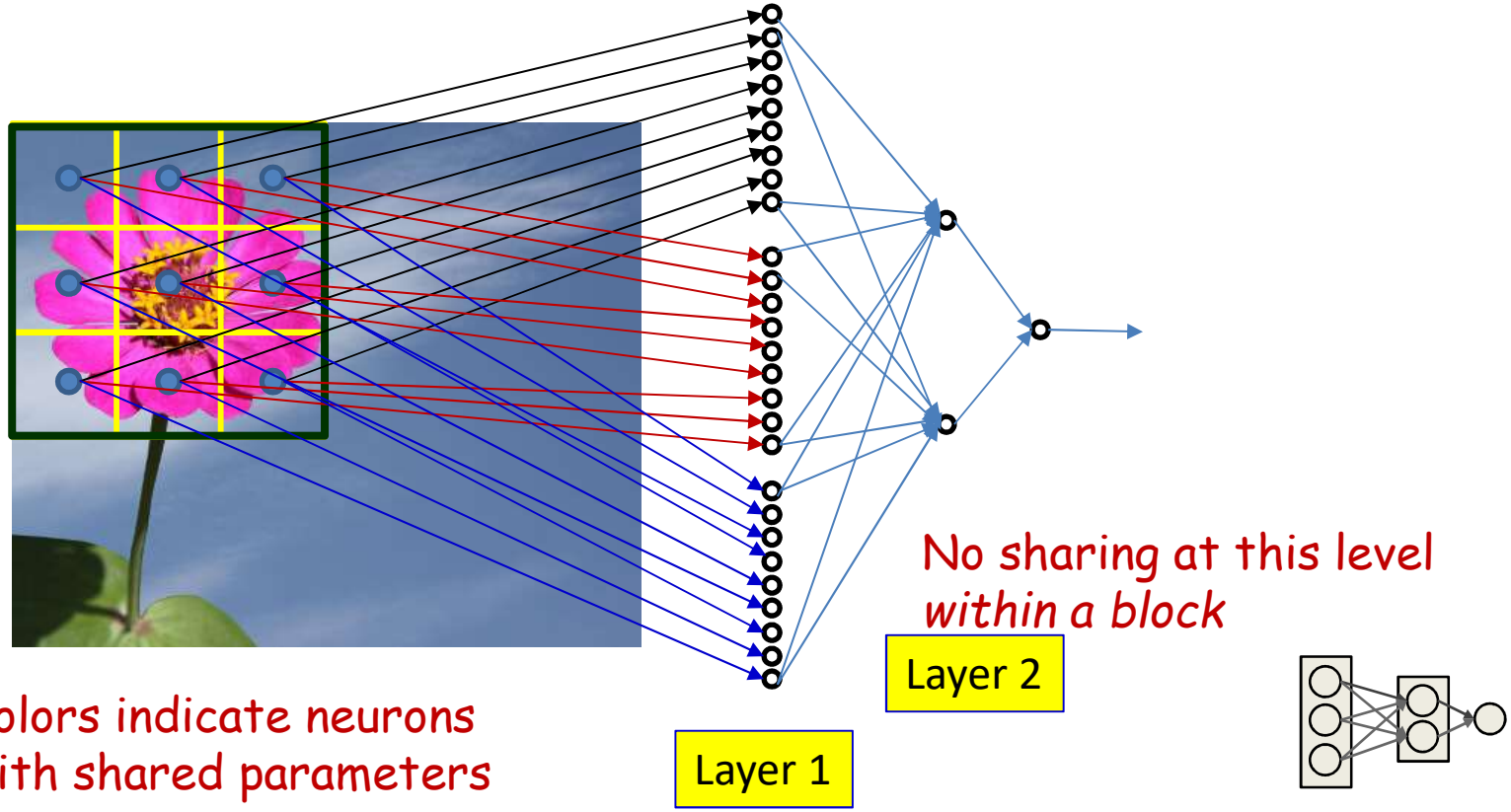
Colors indicate neurons with shared parameters

Layer 1



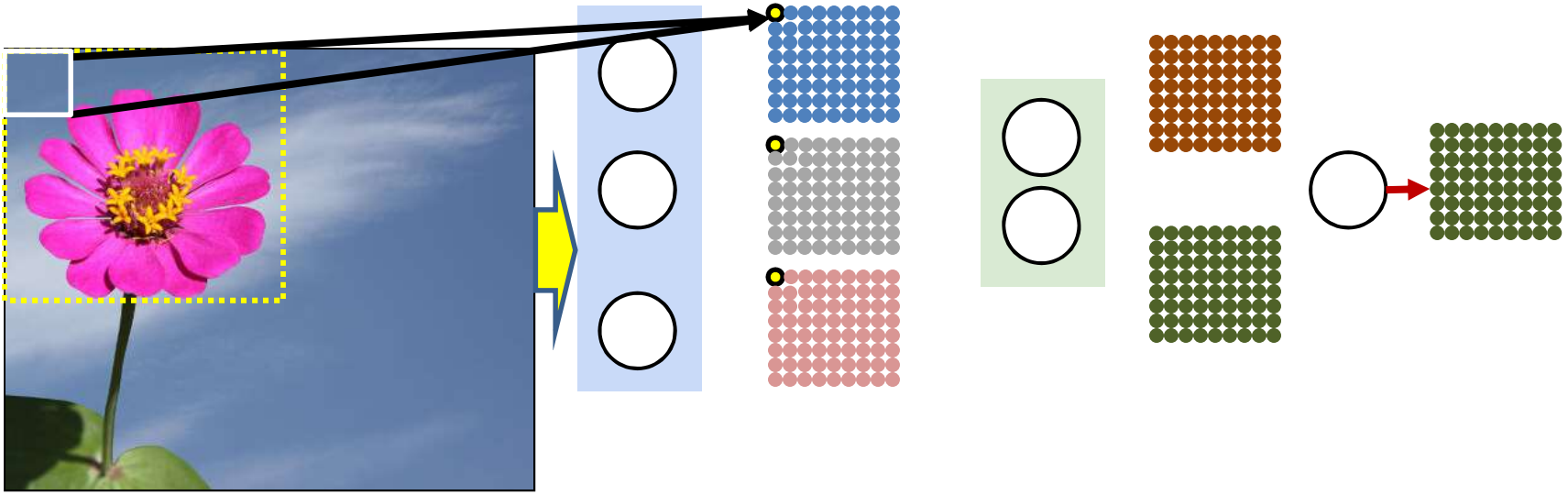
- The network that analyzes individual blocks is now itself a shared parameter network..

This is *still* just scanning with a shared parameter network



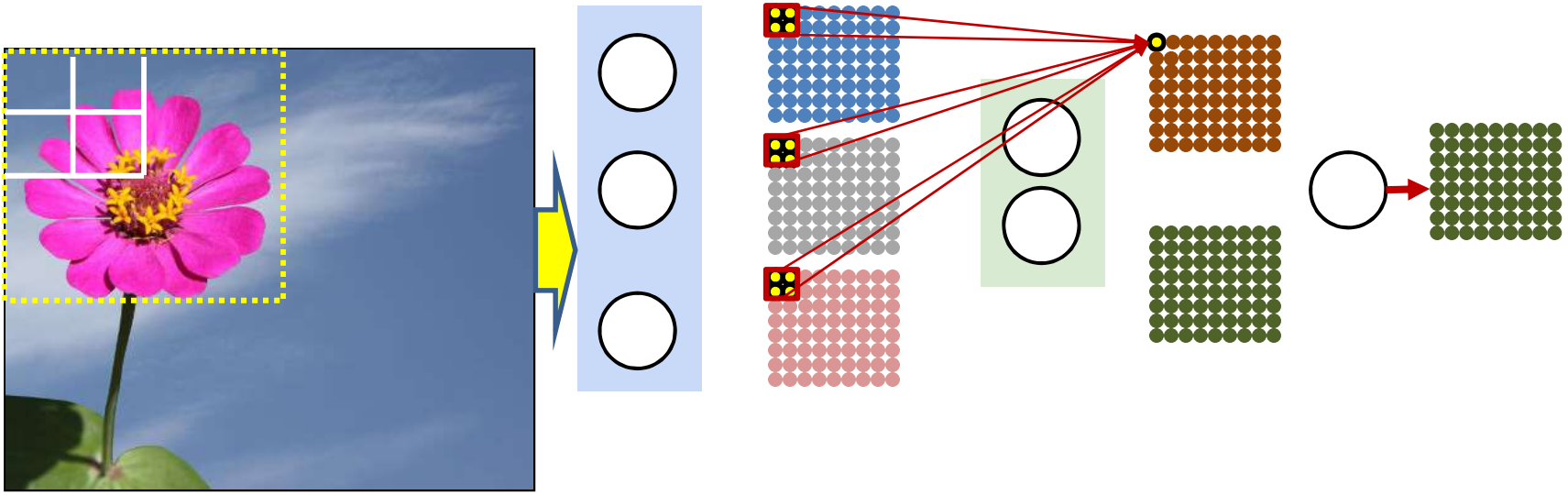
- The network that analyzes individual blocks is now itself a shared parameter network..

This logic can be recursed



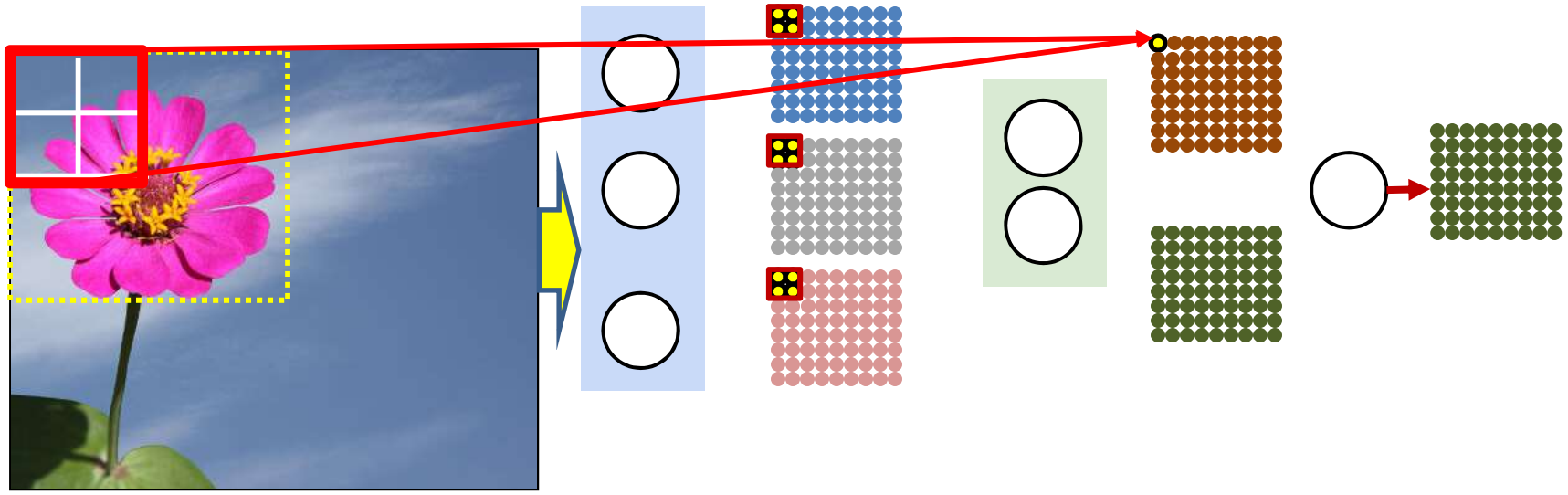
- Building the pattern over 3 layers

This logic can be recursed



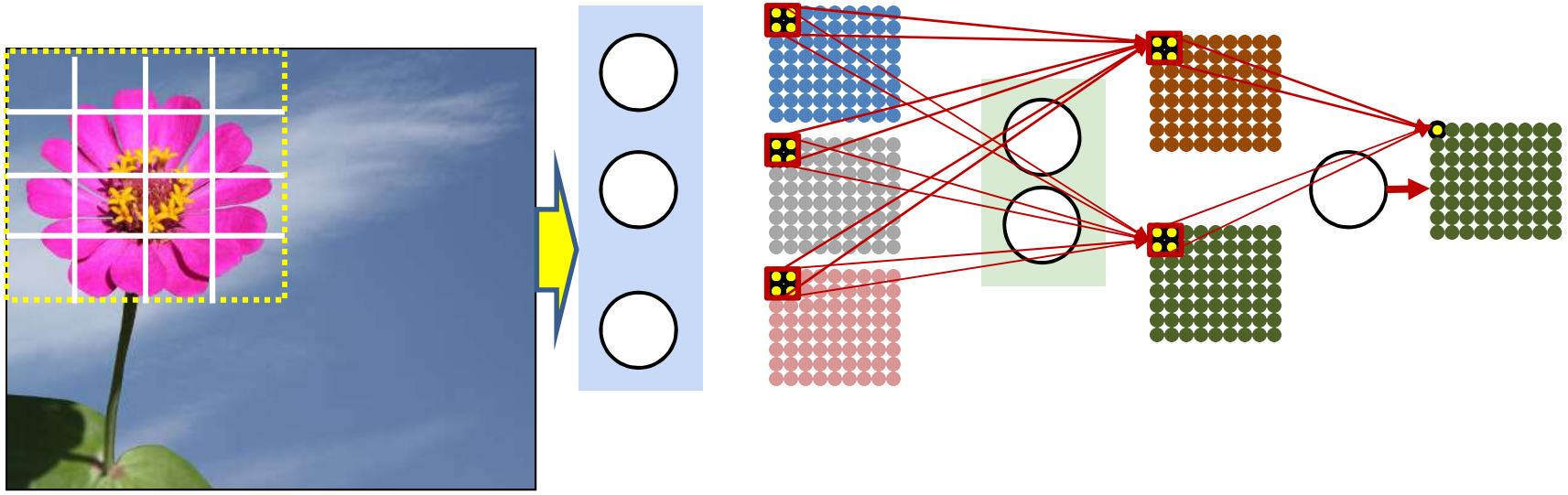
- Building the pattern over 3 layers

This logic can be recursed



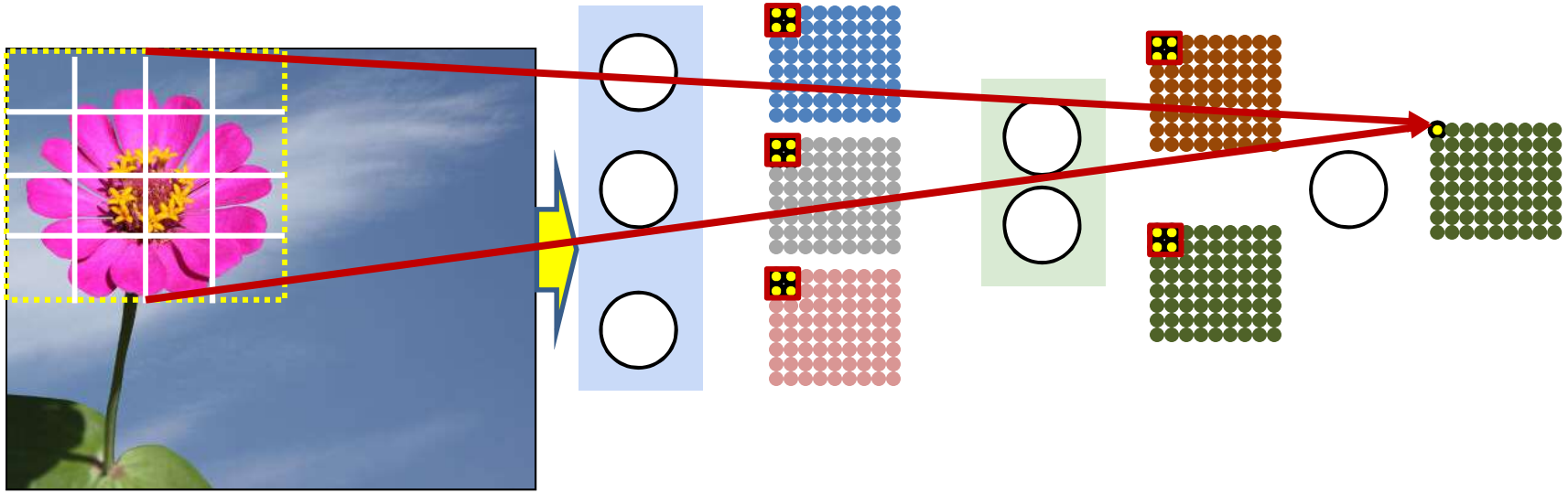
- Building the pattern over 3 layers

This logic can be recursed



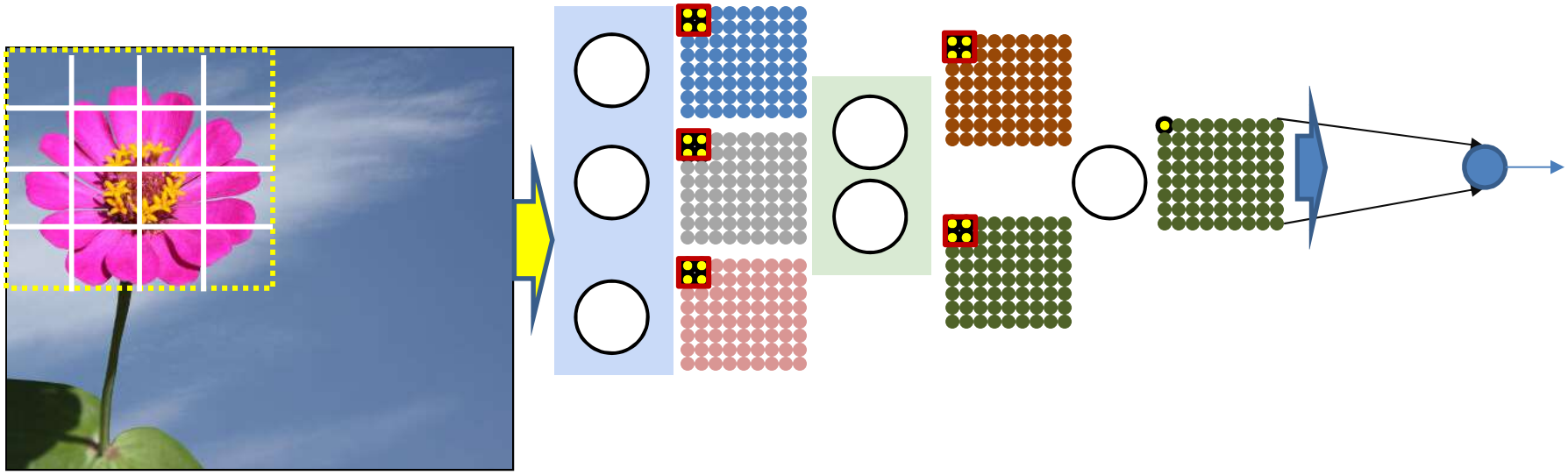
- Building the pattern over 3 layers

This logic can be recursed



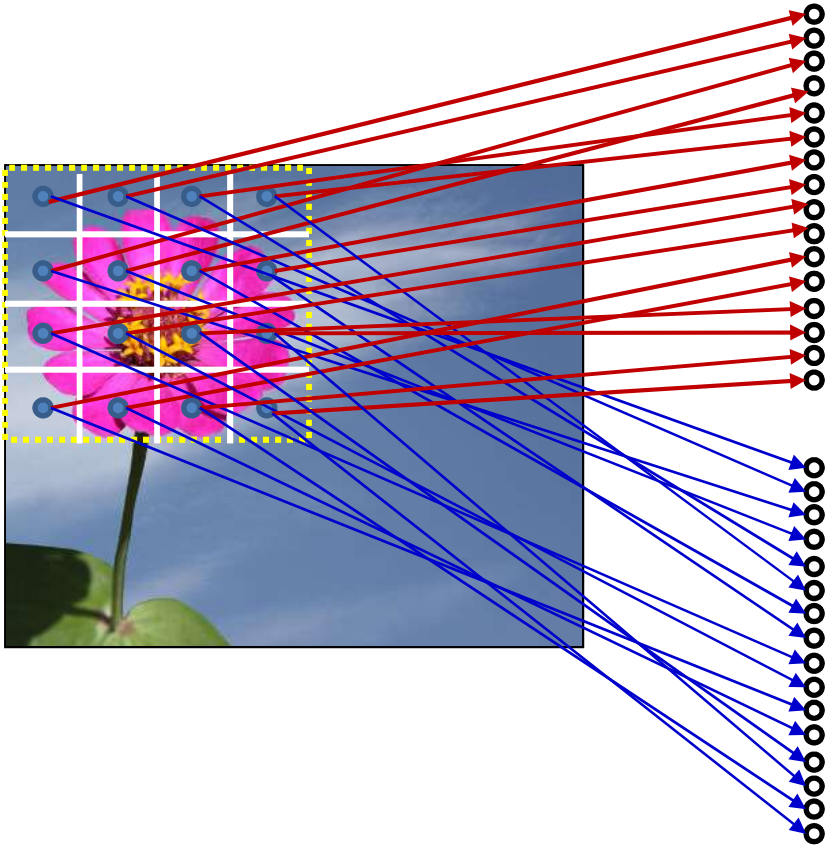
- Building the pattern over 3 layers

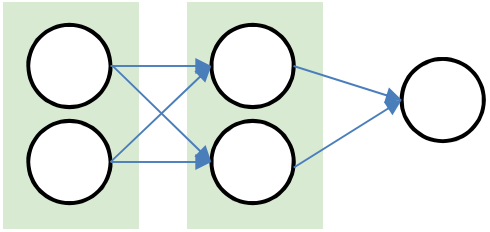
Does the picture have a flower



- Building the pattern over 3 layers
- The final classification for the entire image views the outputs from all locations, as seen in the final map

The 3-layer shared parameter net

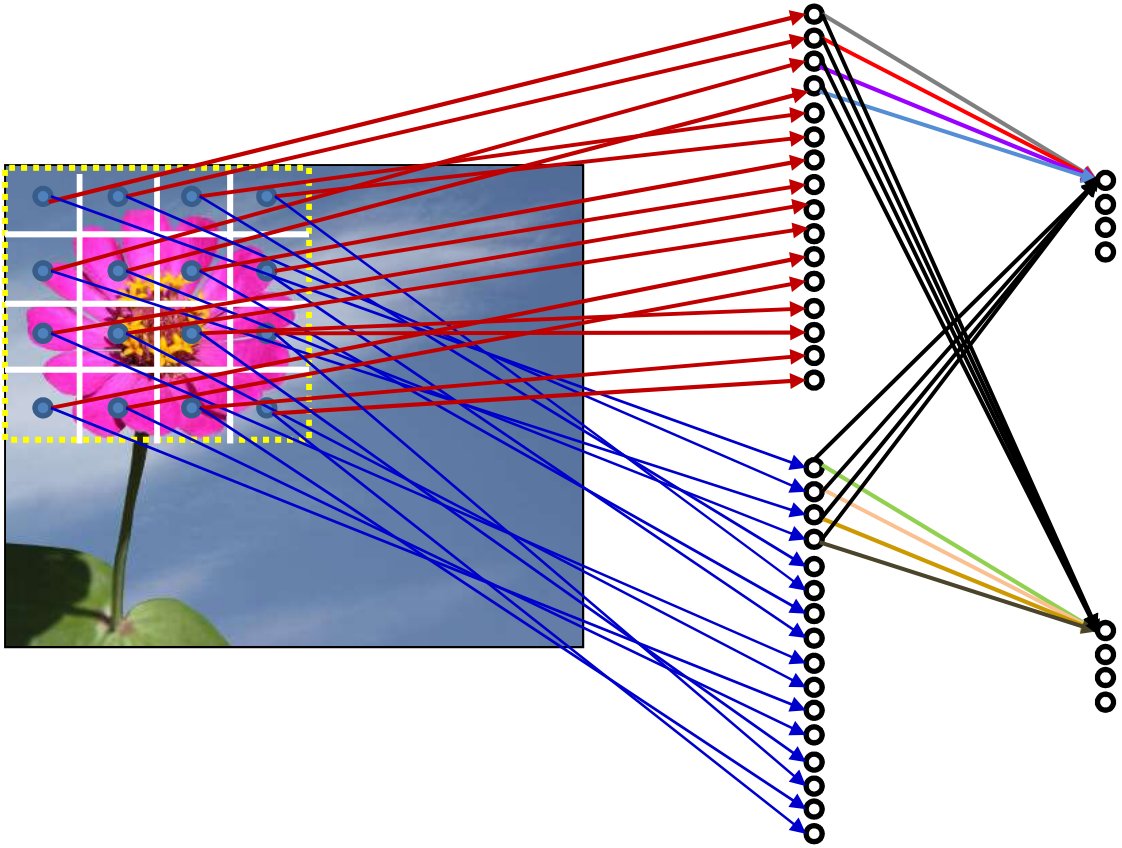




Showing a simpler 2x2x1 network to fit on the slide

- Building the pattern over 3 layers

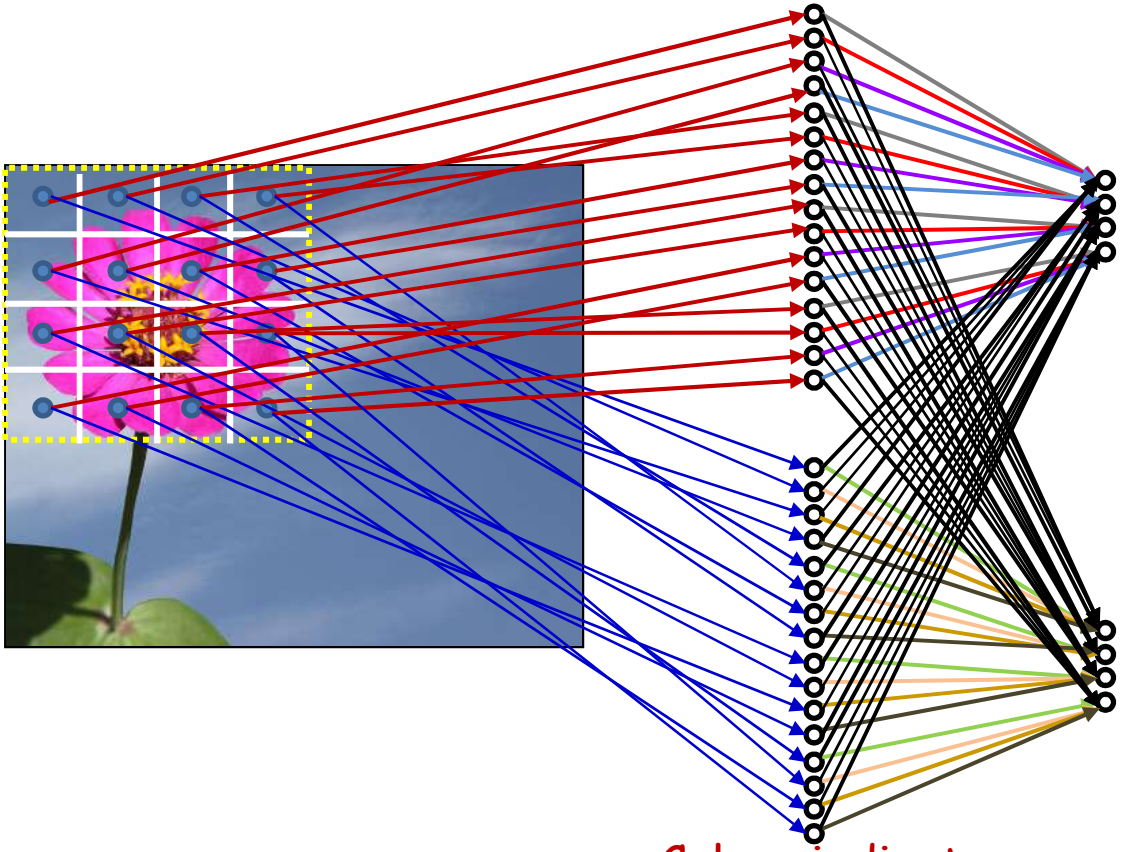
The 3-layer shared parameter net



All weights shown are unique

- Building the pattern over 3 layers

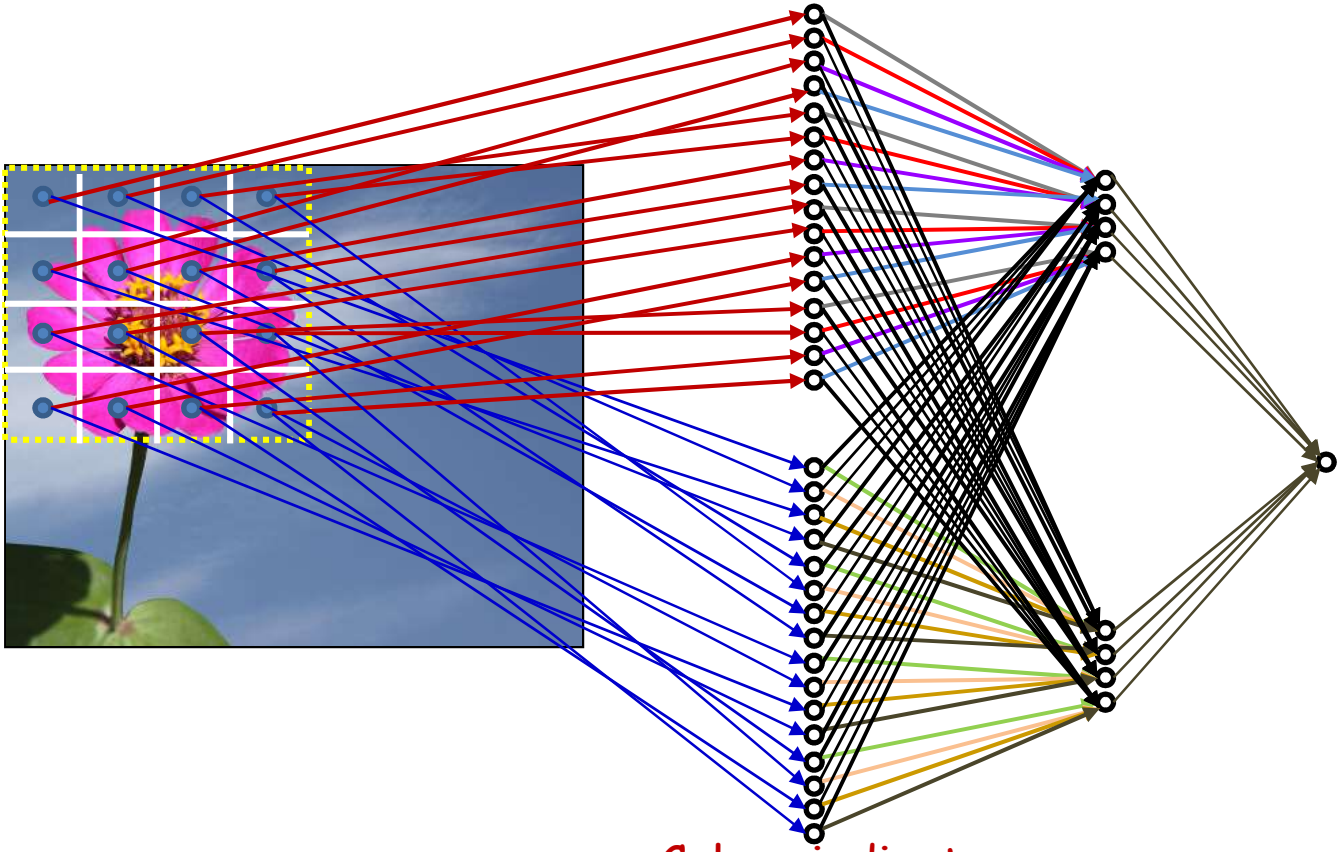
The 3-layer shared parameter net



Colors indicate shared parameters

- Building the pattern over 3 layers

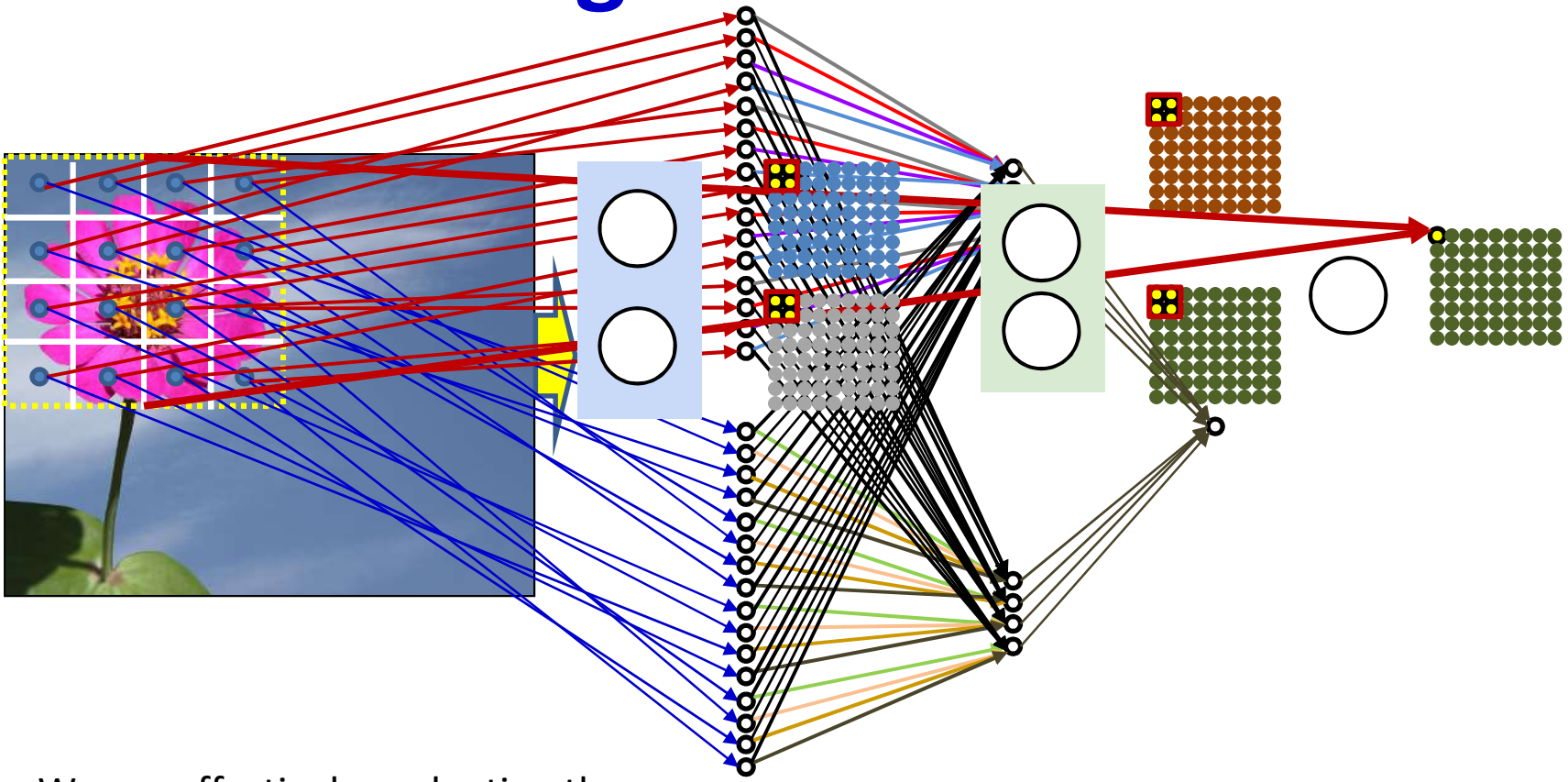
The 3-layer shared parameter net



Colors indicate shared parameters

- Building the pattern over 3 layers

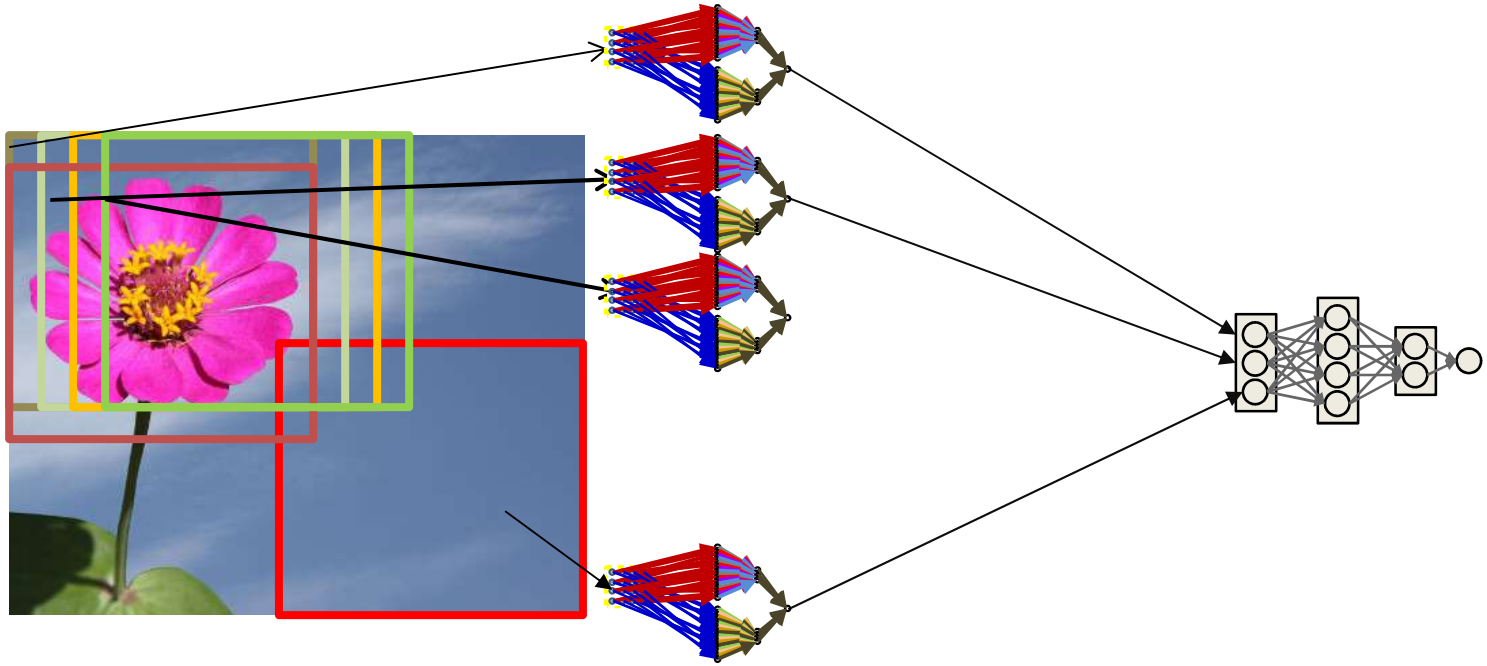
This logic can be recursed



We are effectively evaluating the yellow block with the shared parameter net to the right

Every block is evaluated using the same net in the overall computation

Using hierarchical build-up of features



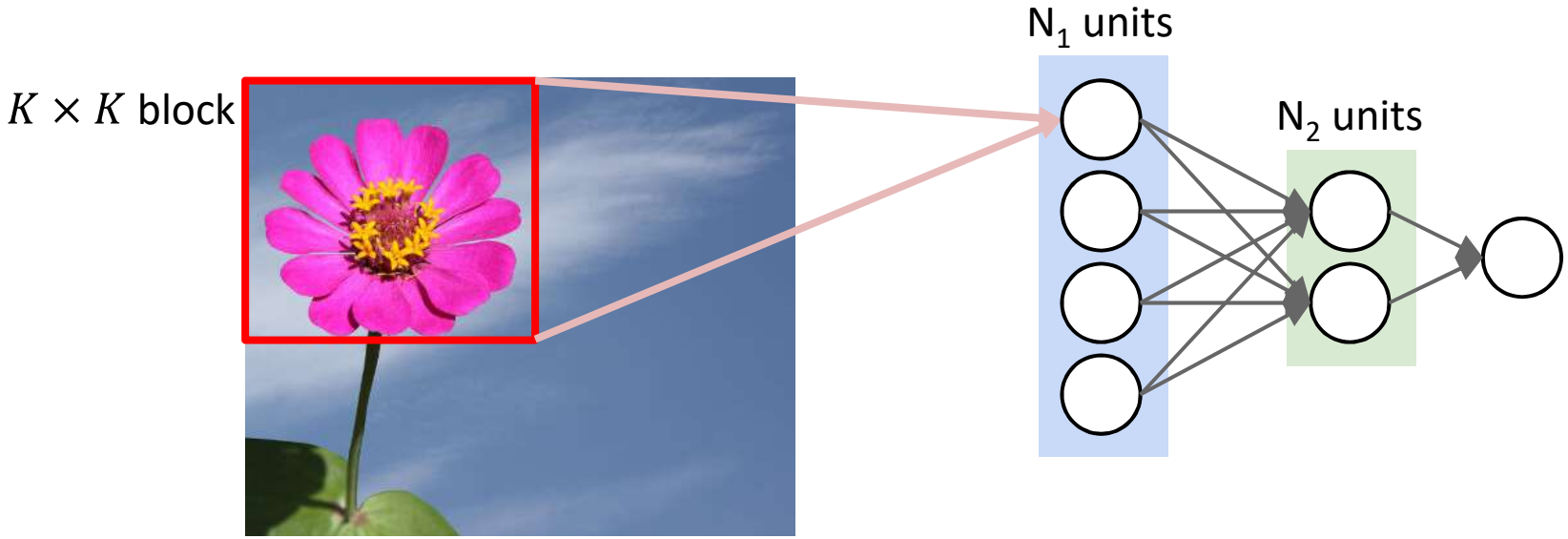
- The individual blocks are now themselves shared-parameter networks
- We scan the figure using the shared parameter network
- The entire operation can be viewed as a single giant network
  - Where individual subnets are themselves shared-parameter nets

Why distribute?

- Distribution forces *localized* patterns in lower layers
  - More generalizable

- Number of parameters…

Parameters in *Undistributed* network

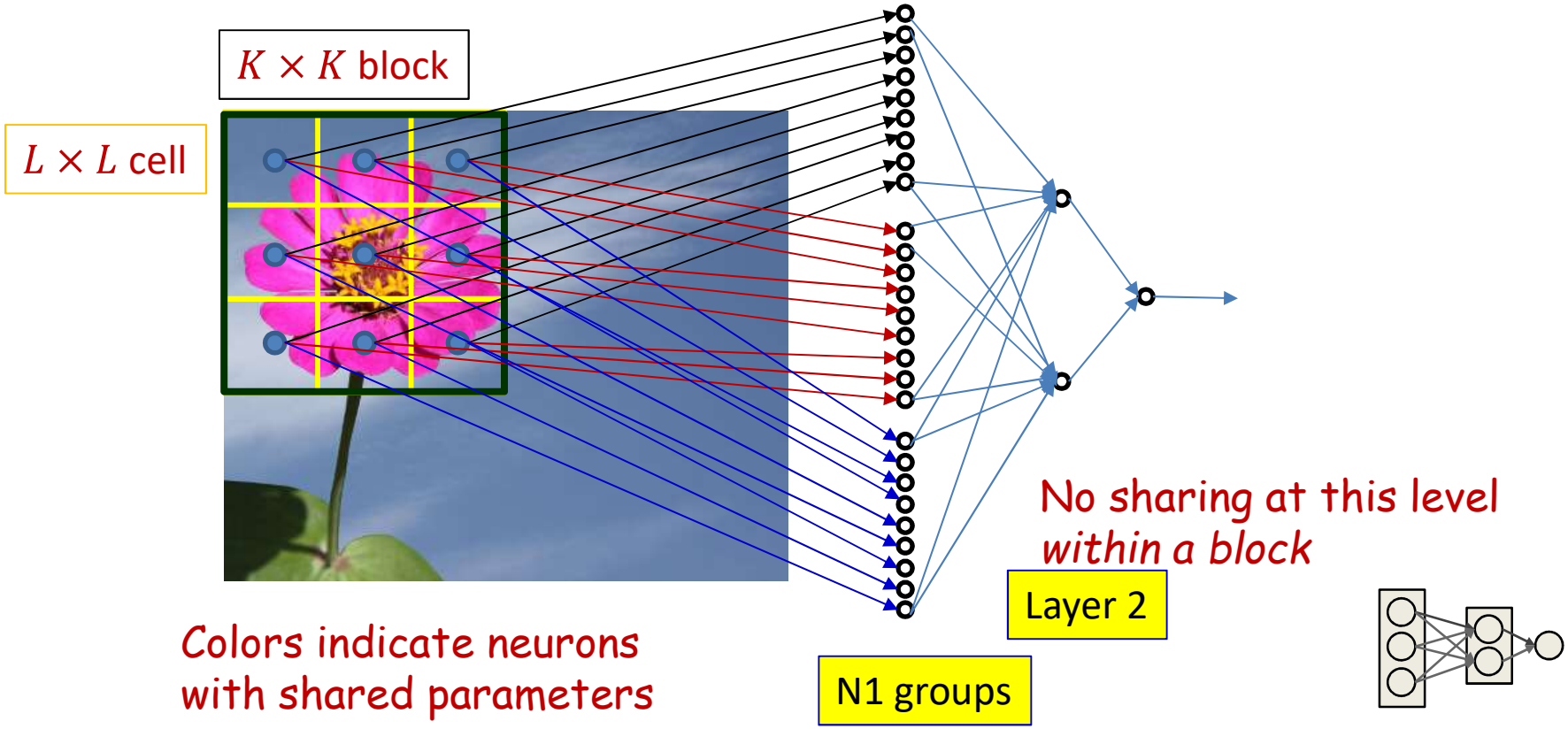


- Only need to consider what happens in *one* block
  - All other blocks are scanned by the same net

- $(K^2 + 1)N_1$ weights in first layer
- $(N_1 + 1)N_2$ weights in second layer
  - $(N_{i-1} + 1)N_i$ weights in subsequent ith layer

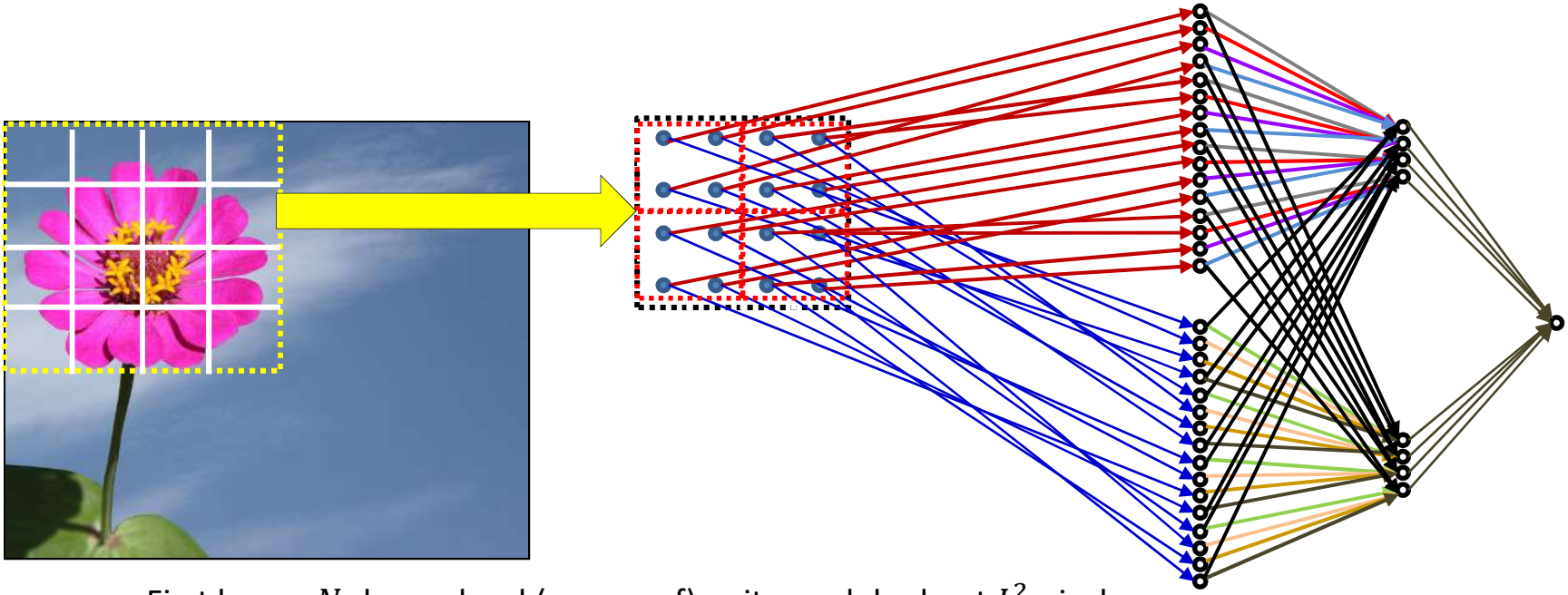
- Total parameters: $\mathcal{O}(K^2 N_1 + N_1 N_2 + N_2 N_3 \ldots)$
  - Ignoring the bias term

When distributed over 2 layers



- First layer: N_1 lower-level units, each looks at L^2 pixels
  - $N_1(L^2+1)$ weights
- Second layer needs $\left(\left(\frac{K}{L}\right)^2 N_1 + 1\right)N_2$ weights
- Subsequent layers needs $N_{i-1}N_i$ when distributed over 2 layers only
  - Total parameters: $\mathcal{O}\left(L^2N_1 + \left(\frac{K}{L}\right)^2 N_1N_2 + N_2N_3 \ldots\right)$

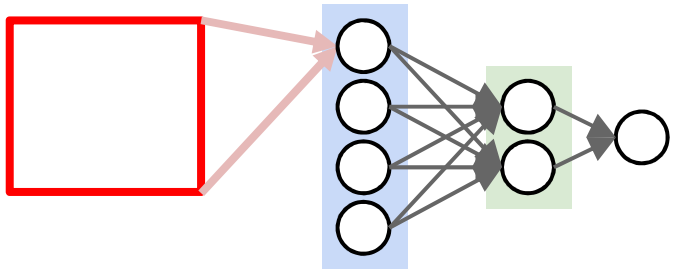
When distributed over 3 layers



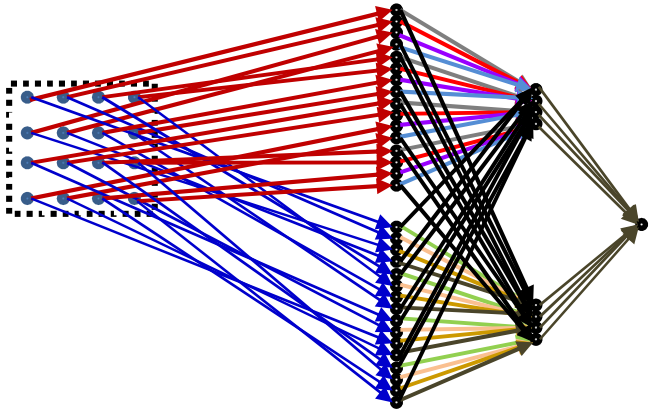
- First layer: N_1 lower-level (groups of) units, each looks at L_1^2 pixels
  - $N_1(L_1^2 + 1)$ weights
- Second layer: N_2 (groups of) units looking at groups of $L_2 \times L_2$ connections from each of N_1 first-level neurons
  - $(L_2^2 N_1 + 1)N_2$ weights
- Third layer:
  - $(\left(\frac{K}{L_1 L_2}\right)^2 N_2 + 1)N_3$ weights
- Subsequent layers need $N_{i-1}N_i$ neurons
  - Total parameters: $\mathcal{O}\left(L_1^2 N_1 + L_2^2 N_1 N_2 + \left(\frac{K}{L_1 L_2}\right)^2 N_2 N_3 + \cdots\right)$

Comparing Number of Parameters

Conventional MLP, not distributed



Distributed (3 layers)

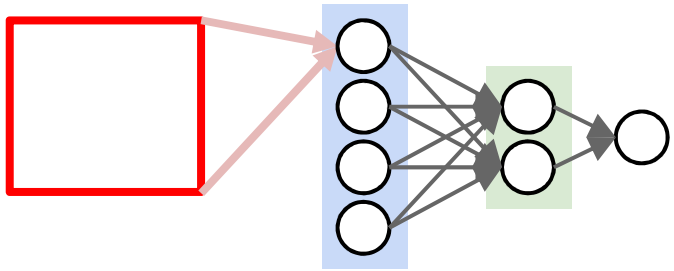


- $\mathcal{O}(K^2 N_1 + N_1 N_2 + N_2 N_3 \dots)$
- For this example, let $K = 16, N_1 = 4, N_2 = 2, N_3 = 1$
- Total 1034 weights

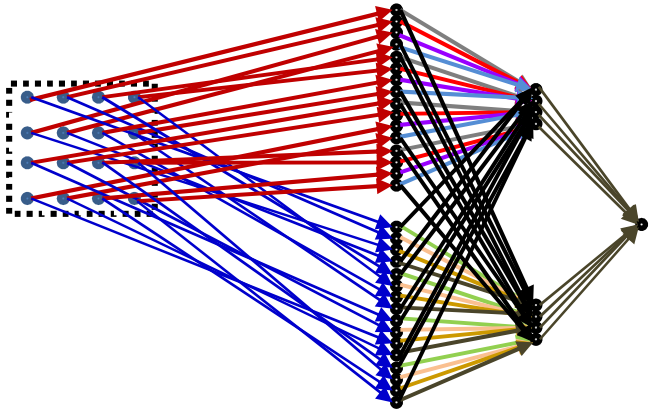
- $\mathcal{O}\left(L_1^2 N_1 + L_2^2 N_1 N_2 + \left(\frac{K}{L_1 L_2}\right)^2 N_2 N_3 + \cdots\right)$
- Here, let $K = 16,\ L_1 = 4,\ L_2 = 4,\ N_1 = 4, N_2 = 2, N_3 = 1$
- Total 64+128+8 = 160 weights

Comparing Number of Parameters

Conventional MLP, not distributed



Distributed (3 layers)



- $\mathcal{O}(K^2 N_1 + \sum_i N_i N_{i+1})$

- $\mathcal{O}\Big(L_1^2 N_1 + \sum_{i<nconv-1} L_i^2 N_i N_{i+1} + \left(\frac{K}{\prod_i hop_i}\right)^2 N_{nconv} \quad N_{nconv} + \sum_{i \in flat} N_i N_{i+1}\Big)$

These terms dominate..

Why distribute?

- Distribution forces *localized* patterns in lower layers
  - More generalizable

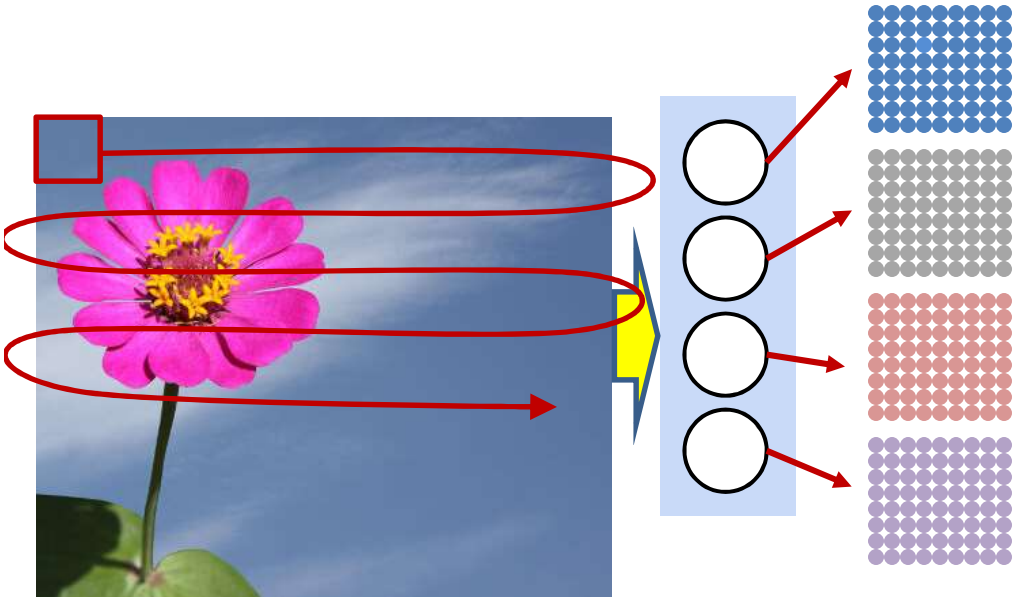
- Number of parameters…
  - Large (sometimes order of magnitude) reduction in parameters
    - Gains increase as we increase the depth over which the blocks are distributed

- **Key intuition: Regardless of the distribution, we can view the network as "scanning" the picture with an MLP**
  - **The only difference is the manner in which parameters are shared in the MLP**

Hierarchical composition: A different perspective

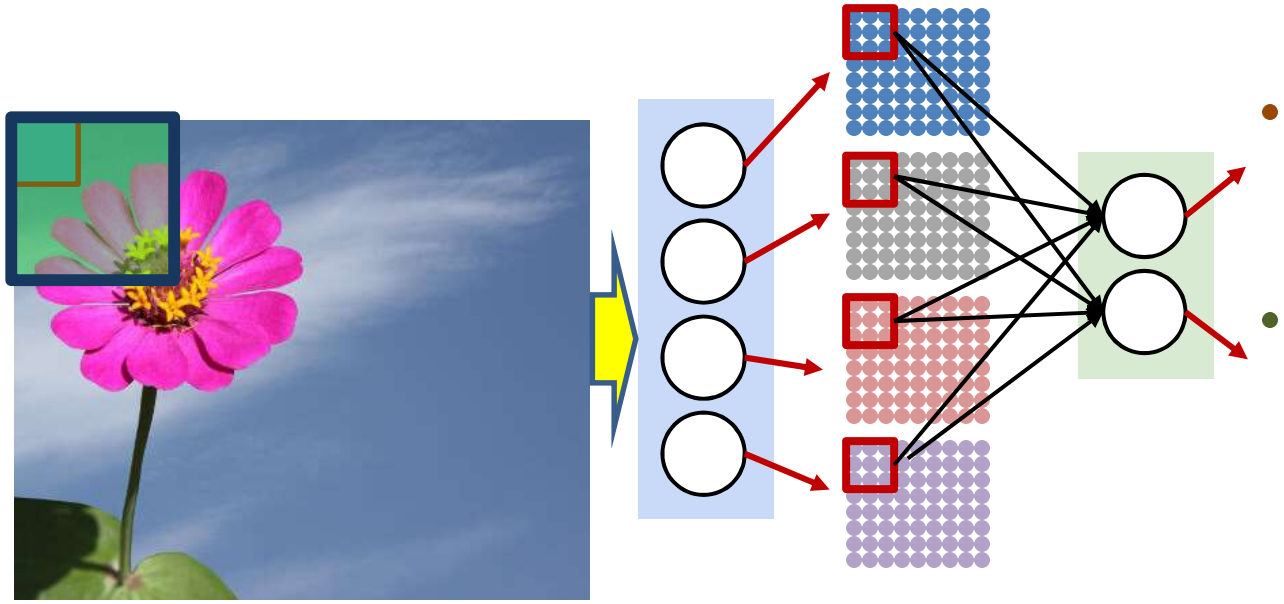
- The entire operation can be redrawn as before as maps of the entire image

Building up patterns



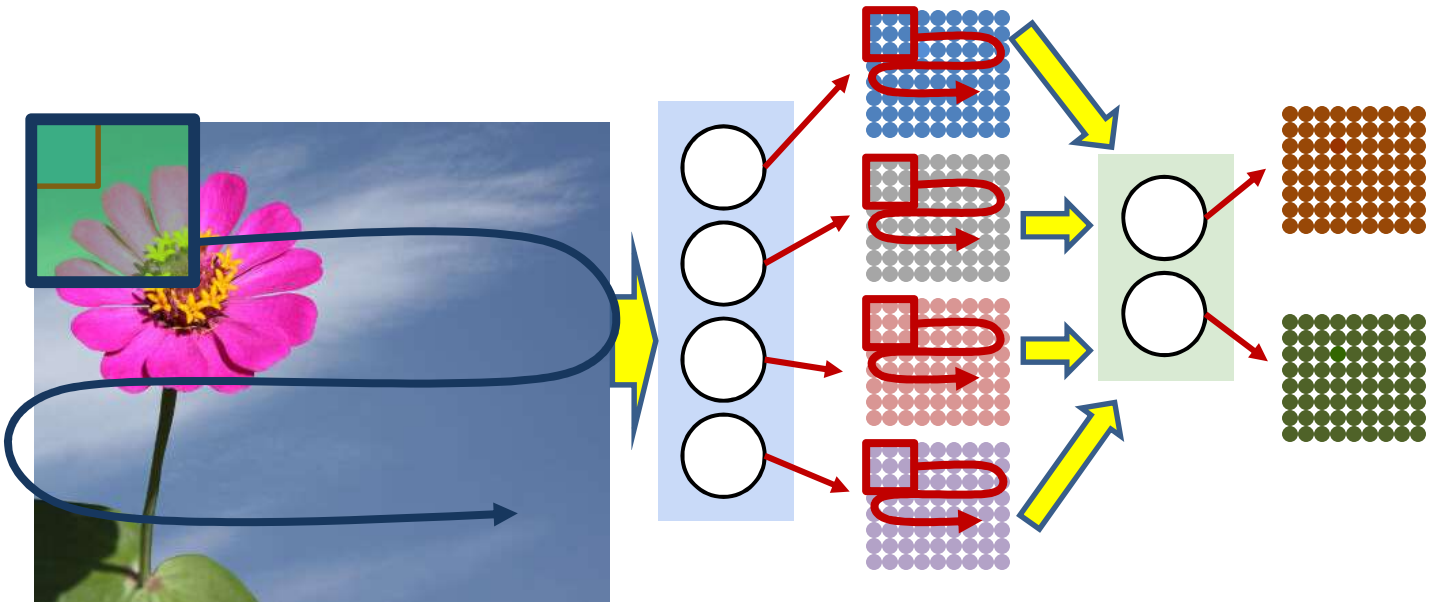
- The first layer looks at small *sub* regions of the main image
  - Sufficient to detect, say, petals

Some modifications



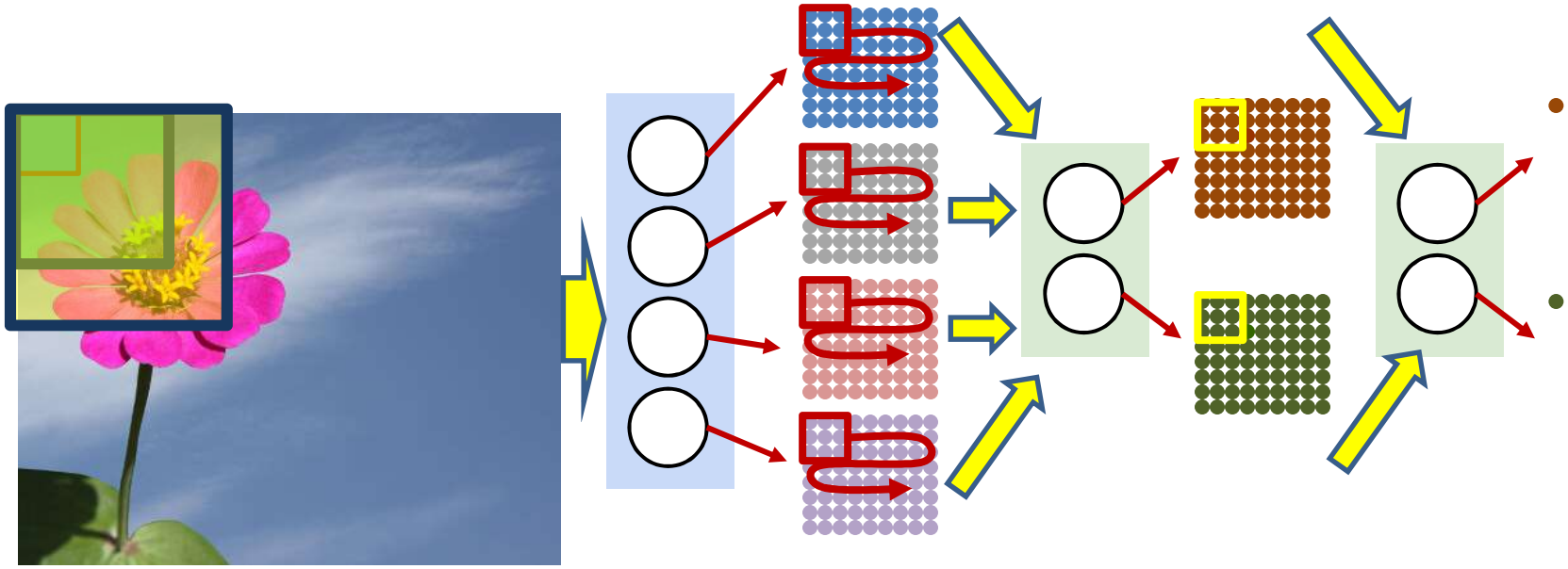
- The first layer looks at *sub* regions of the main image
  - Sufficient to detect, say, petals
- The second layer looks at *regions* of the output of the first layer
  - To put the petals together into a flower
  - This corresponds to looking at a larger region of the original input image

Some modifications



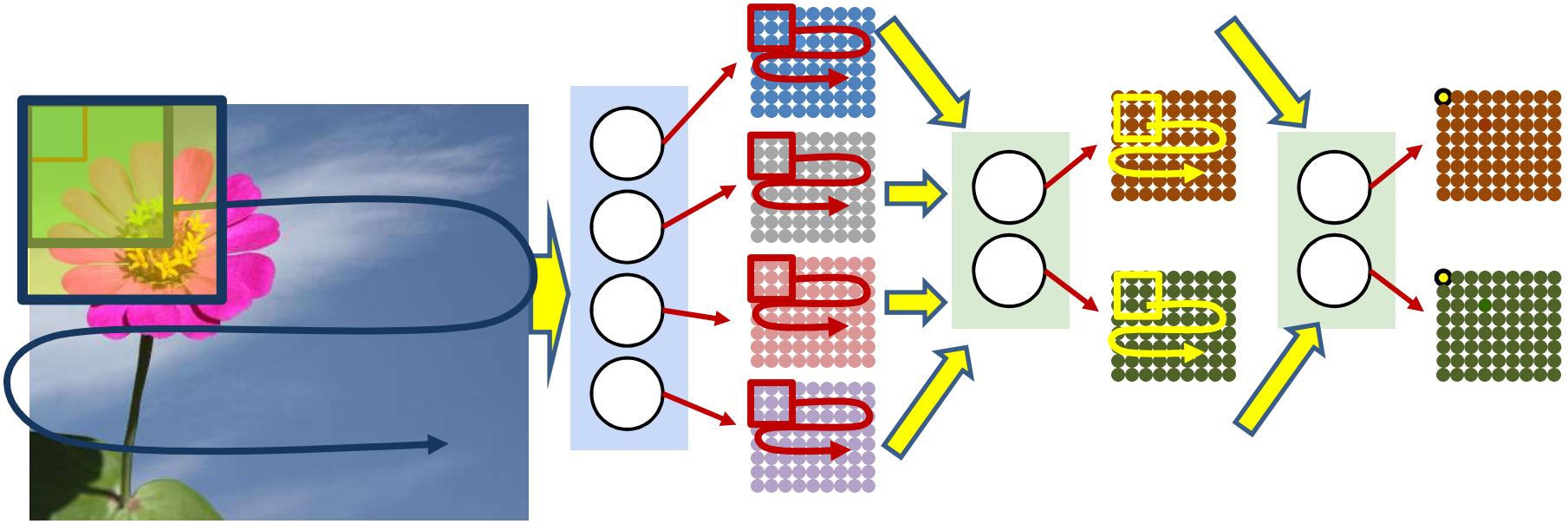
- The first layer looks at *sub* regions of the main image
  - Sufficient to detect, say, petals
- The second layer looks at *regions* of the output of the first layer
  - To put the petals together into a flower
  - This corresponds to looking at a larger region of the original input image

Some modifications



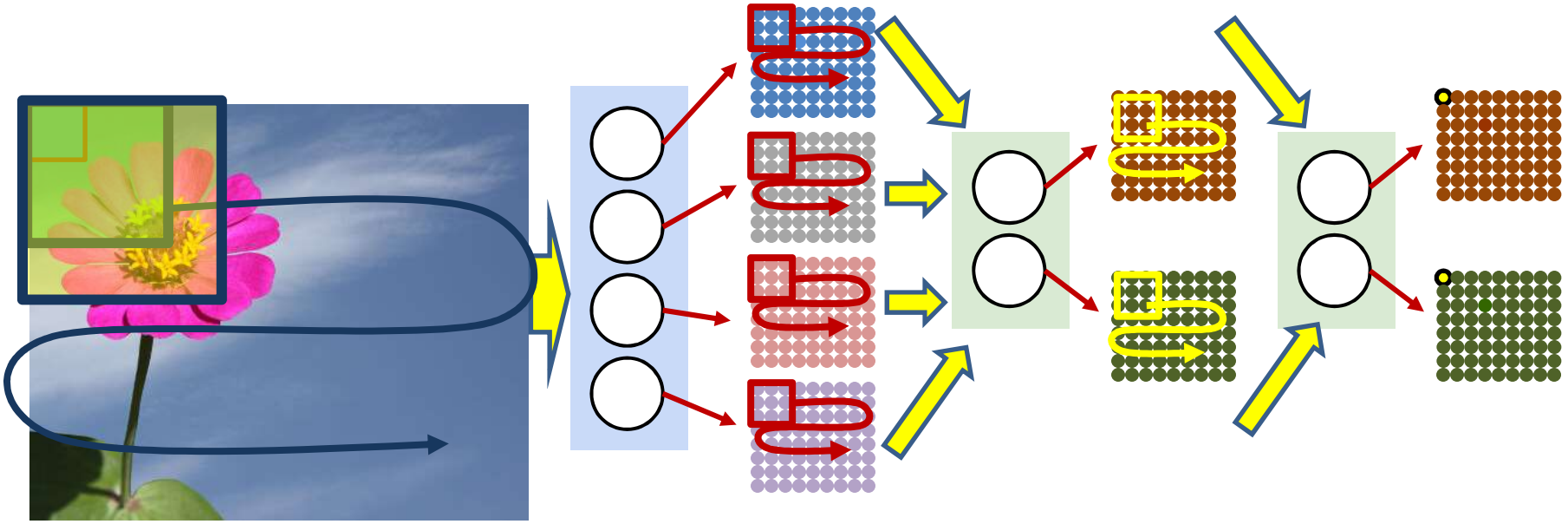
- The first layer looks at *sub* regions of the main image
  - Sufficient to detect, say, petals
- The second layer looks at *regions* of the output of the first layer
  - To put the petals together into a flower
  - This corresponds to looking at a larger region of the original input image
- We may have any number of layers in this fashion

Some modifications



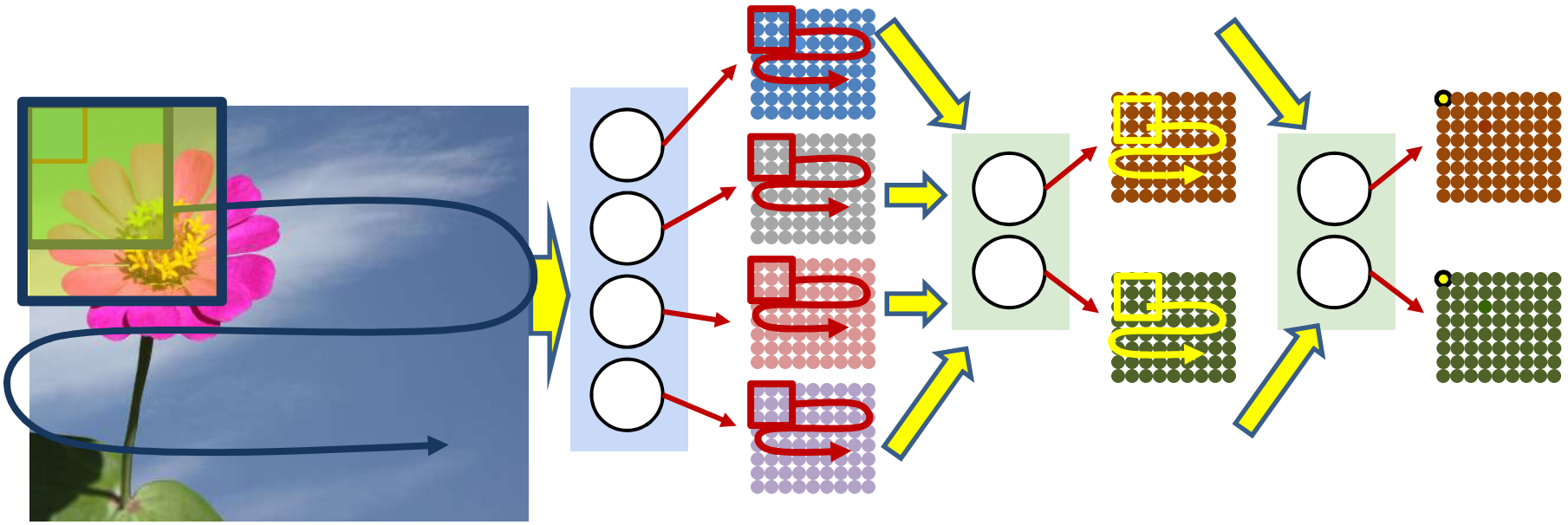
- The first layer looks at *sub* regions of the main image
  - Sufficient to detect, say, petals
- The second layer looks at *regions* of the output of the first layer
  - To put the petals together into a flower
  - This corresponds to looking at a larger region of the original input image
- We may have any number of layers in this fashion

Terminology



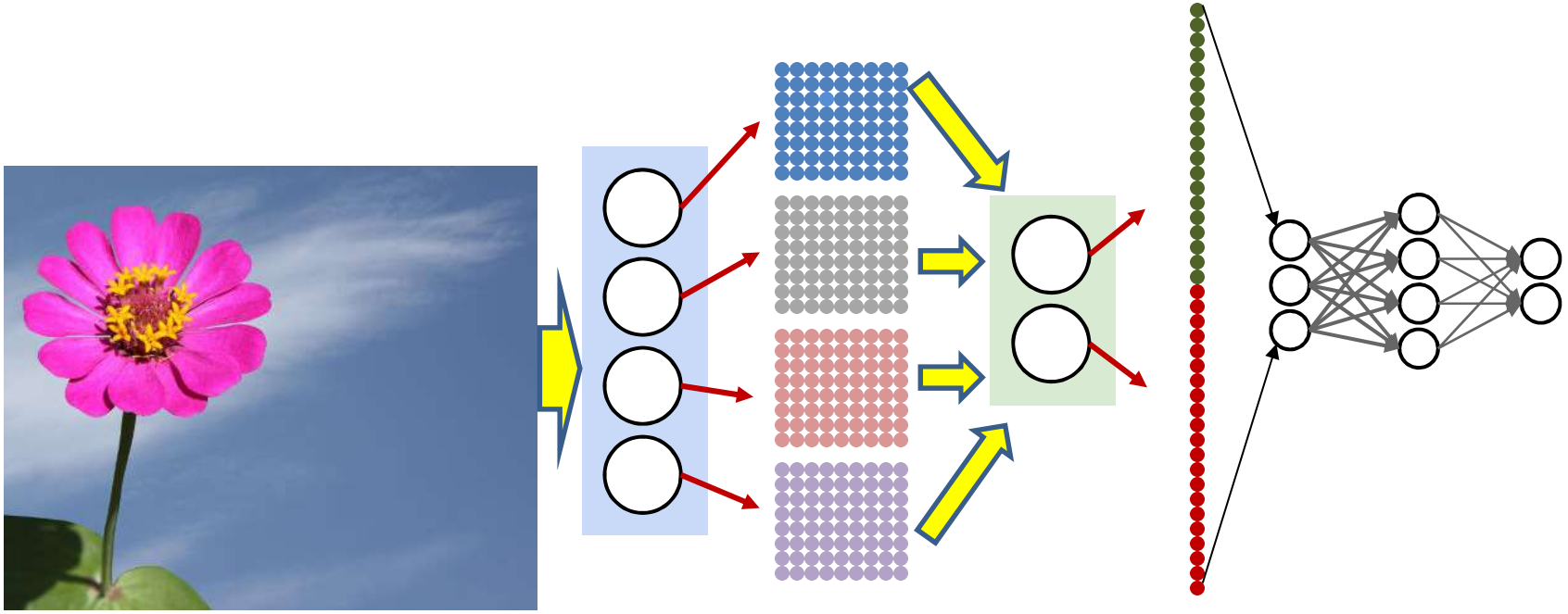
- Each of the scanning neurons is generally called a "filter"
  - Its really a correlation filter as we saw earlier
  - Each filter scans for a pattern in the map it operates on

Terminology



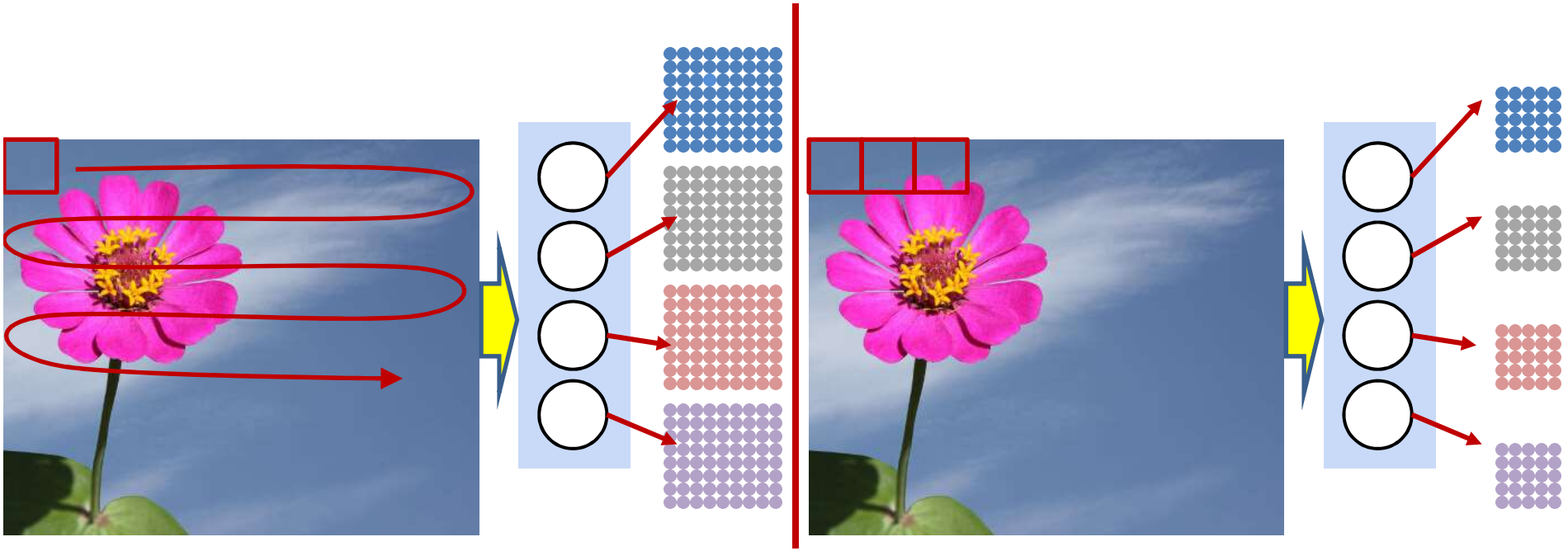
- The pattern in the *input* image that each filter sees is its "Receptive Field"
  - The squares show the *sizes* of the receptive fields for the first, second and third-layer neurons
- The actual receptive field for a first layer filter is simply its arrangement of weights
- For the higher level filters, the actual receptive field is not immediately obvious and must be *calculated*
  - What patterns in the input do the filters actually respond to?
  - Will not actually be simple, identifiable patterns like "petal" and "inflorescence"

Some modifications



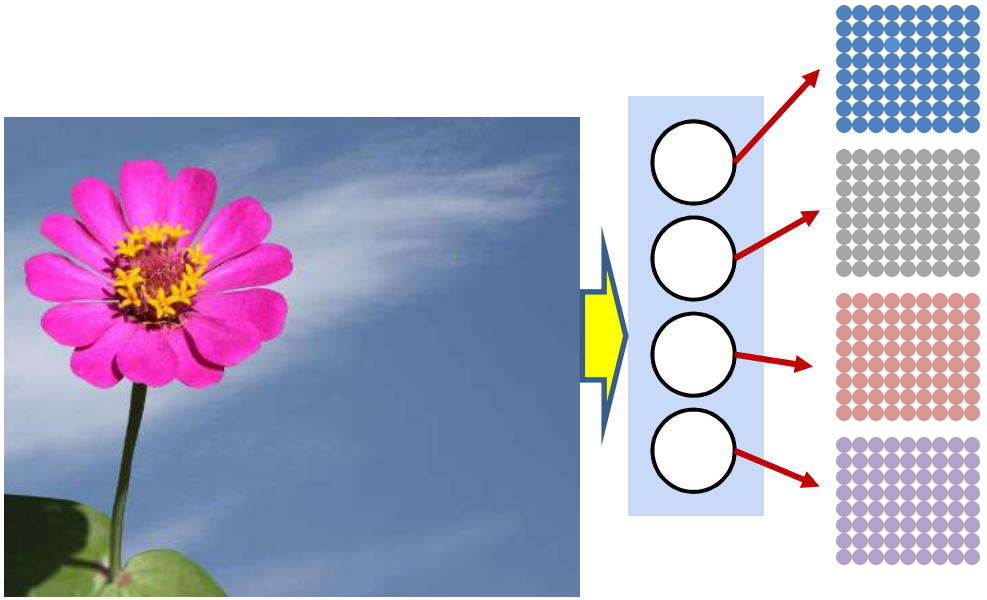
- The final layer may feed directly into a multi layer perceptron rather than a single neuron
- This is exactly the shared parameter net we just saw

Modification 1: Convolutional "Stride"



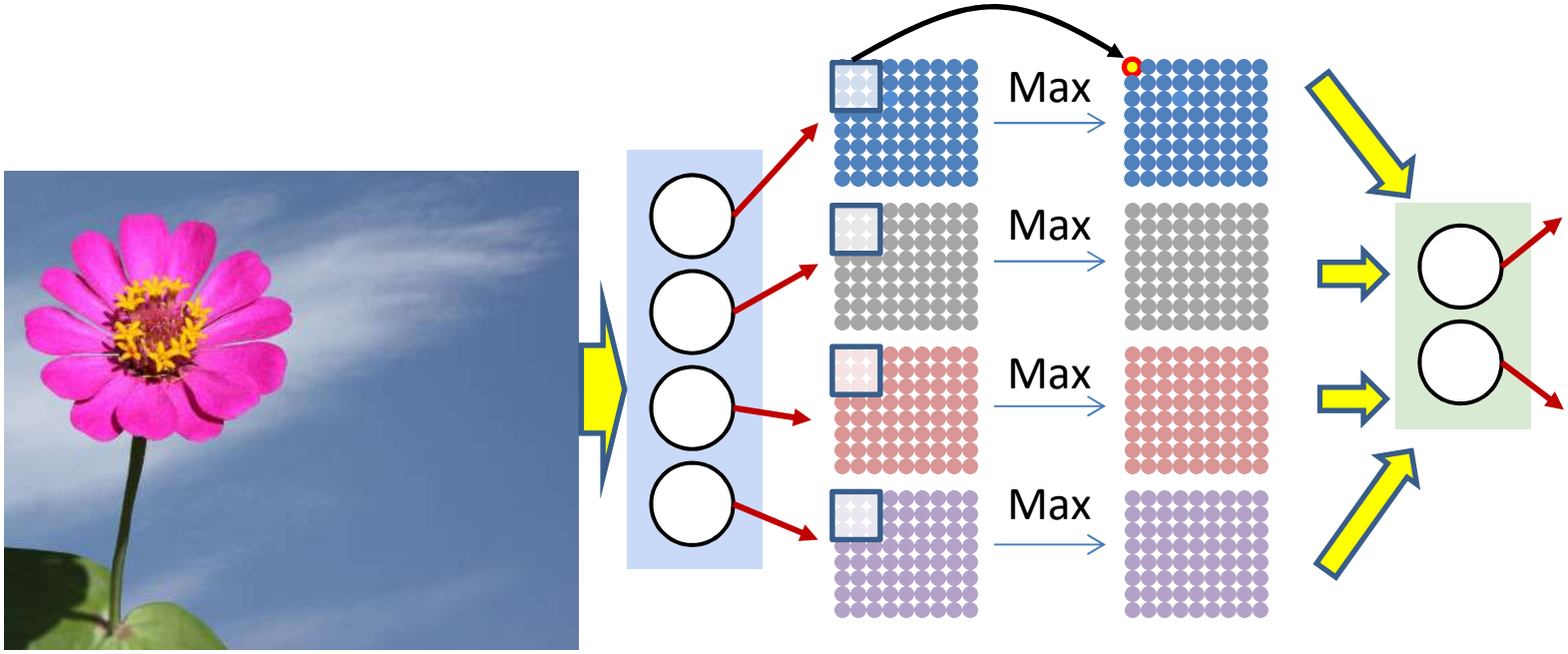
- The scans of the individual "filters" may advance by more than one pixel at a time
  - The "stride" may be greater than 1
  - Effectively increasing the granularity of the scan
    - Saves computation, sometimes at the risk of losing information
- This will result in a reduction of the size of the resulting maps
  - They will shrink by a factor equal to the stride
- This can happen at any layer

Accounting for jitter



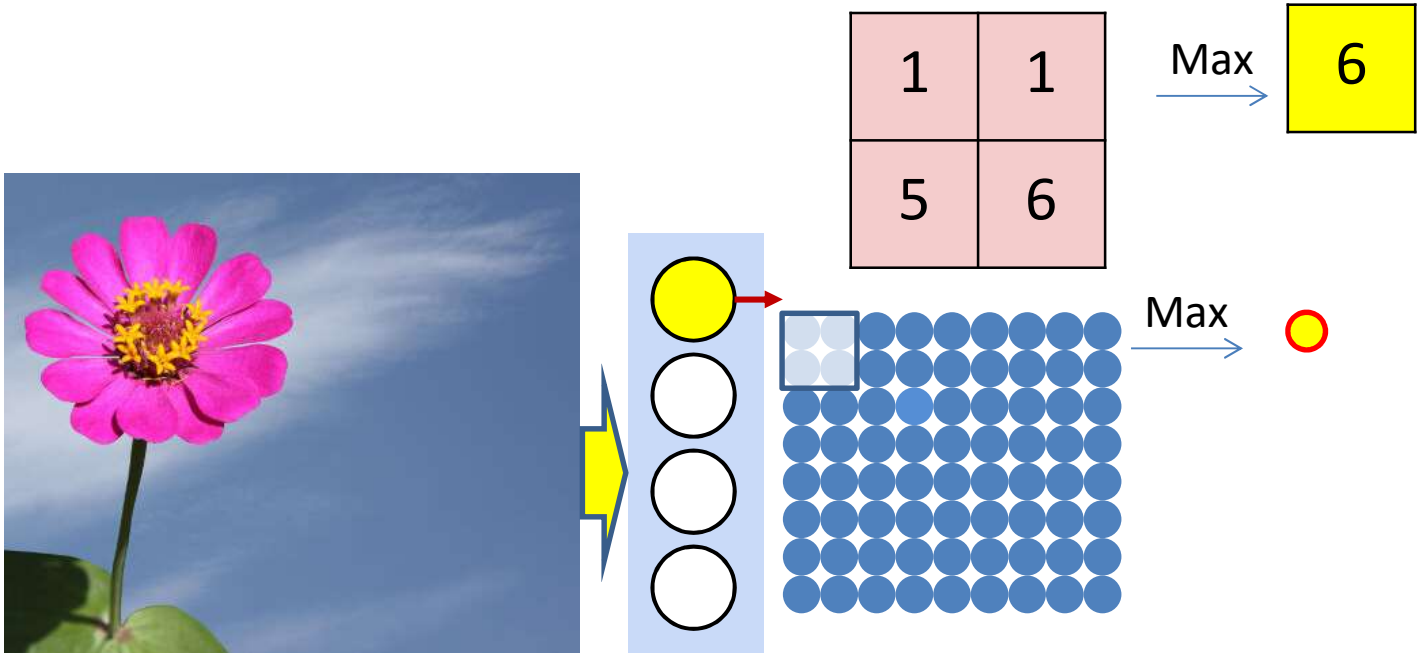
- We would like to account for some jitter in the first-level patterns
  - If a pattern shifts by one pixel, is it still a petal?

Accounting for jitter



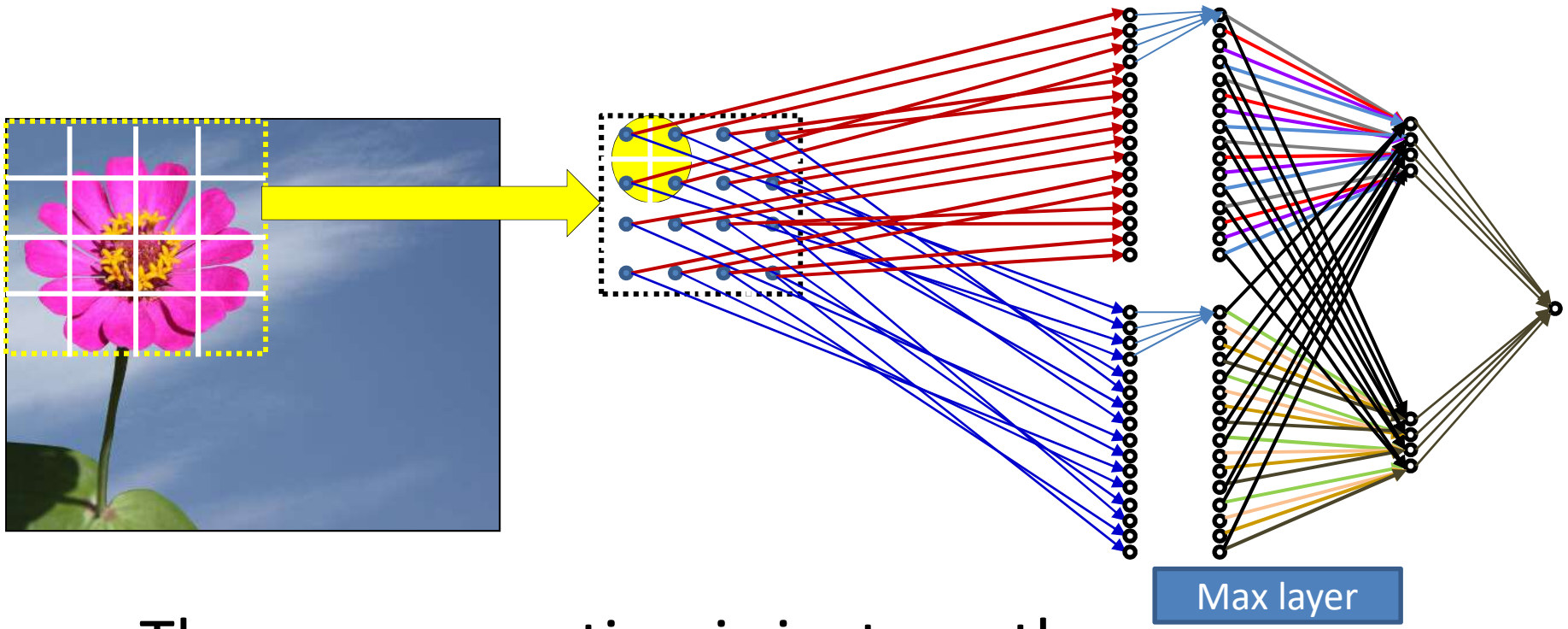
- We would like to account for some jitter in the first-level patterns
  - If a pattern shifts by one pixel, is it still a petal?
  - A small jitter is acceptable
    - Replace each value by the maximum of the values within a small region around it
      - ***Max filtering or Max pooling***

Accounting for jitter



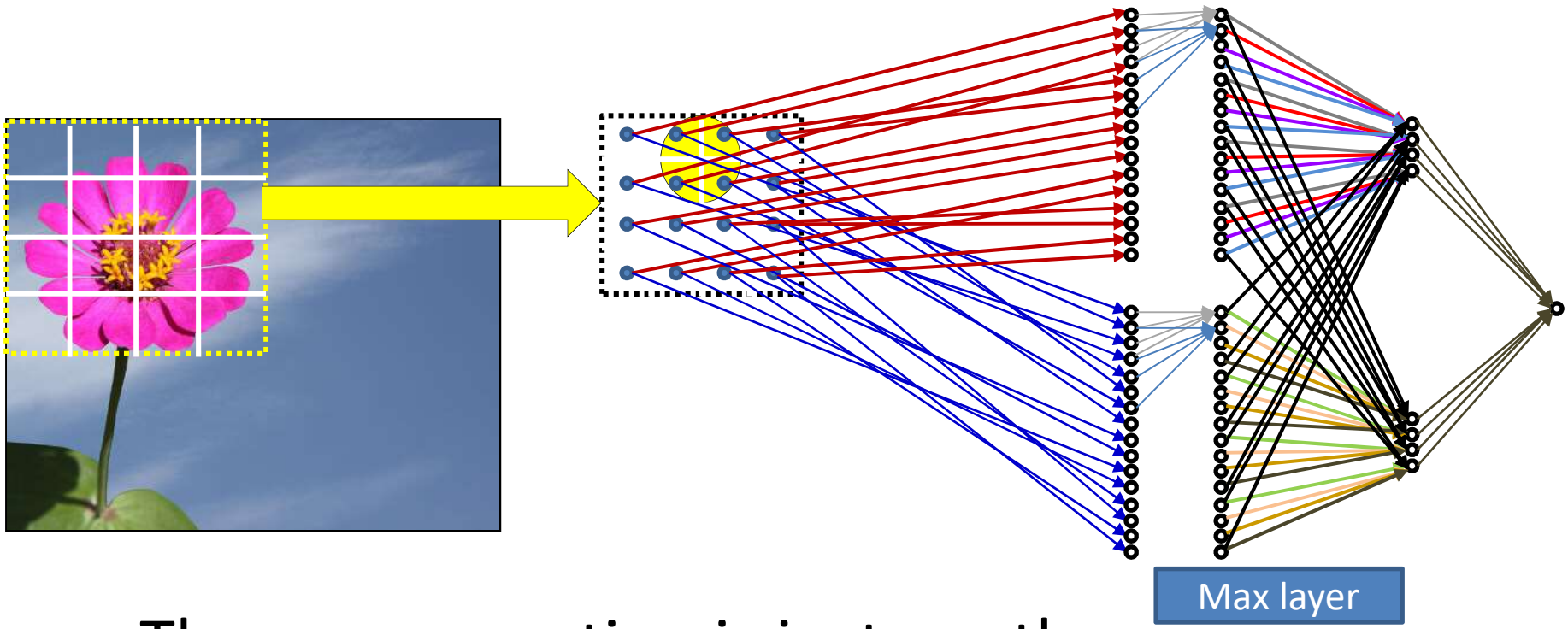
- We would like to account for some jitter in the first-level patterns
  - If a pattern shifts by one pixel, is it still a petal?
  - A small jitter is acceptable
    - Replace each value by the maximum of the values within a small region around it
      - ***Max filtering*** **or** ***Max pooling***

The max operation is just a neuron



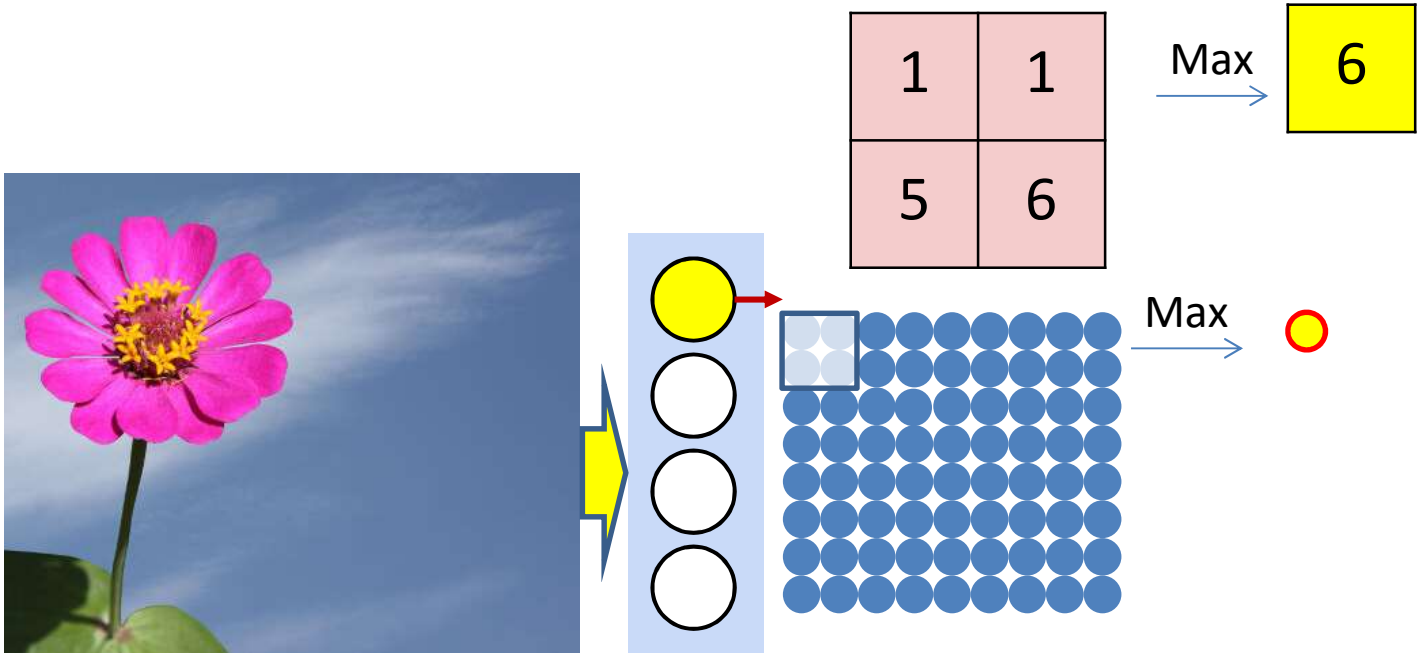
- The max operation is just another neuron
- Instead of applying an activation to the weighted sum of inputs, each neuron just computes the maximum over all inputs

The max operation is just a neuron



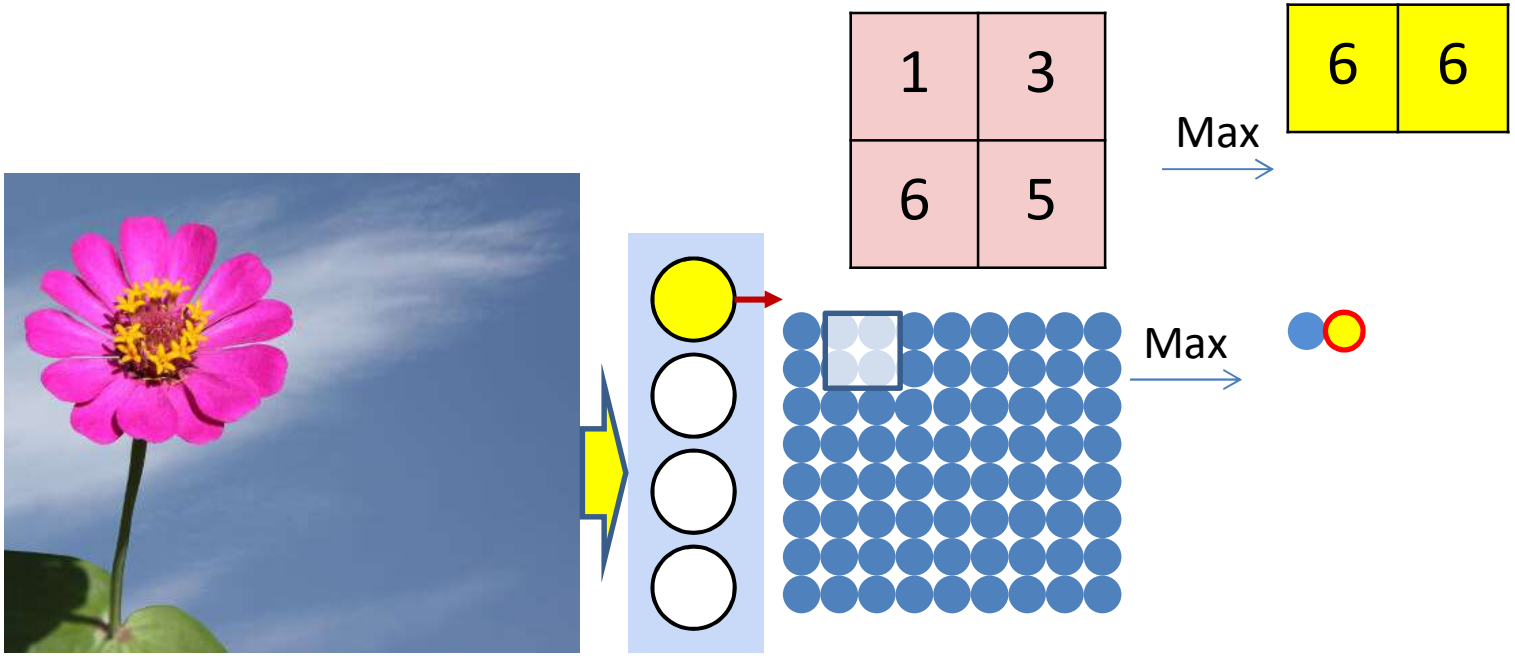
- The max operation is just another neuron
- Instead of applying an activation to the weighted sum of inputs, each neuron just computes the maximum over all inputs

Accounting for jitter



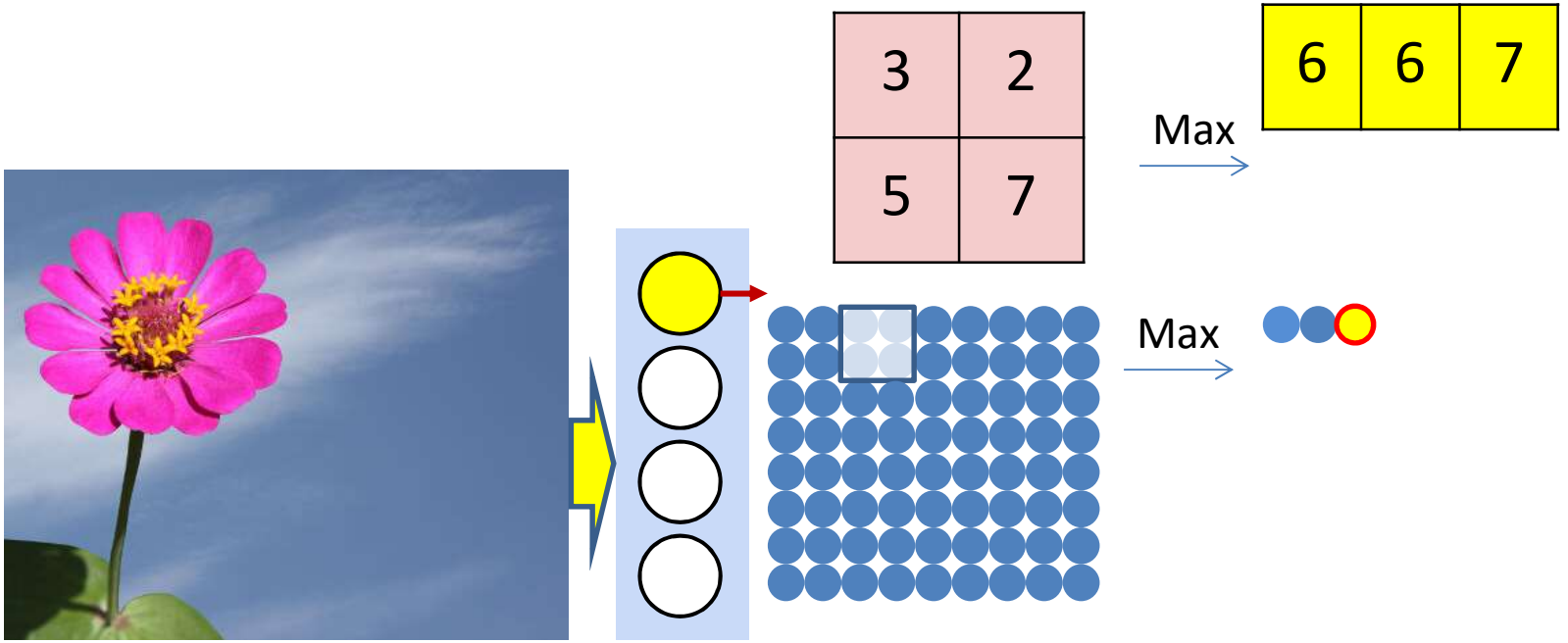
- The max filtering can also be performed as a scan

Accounting for jitter



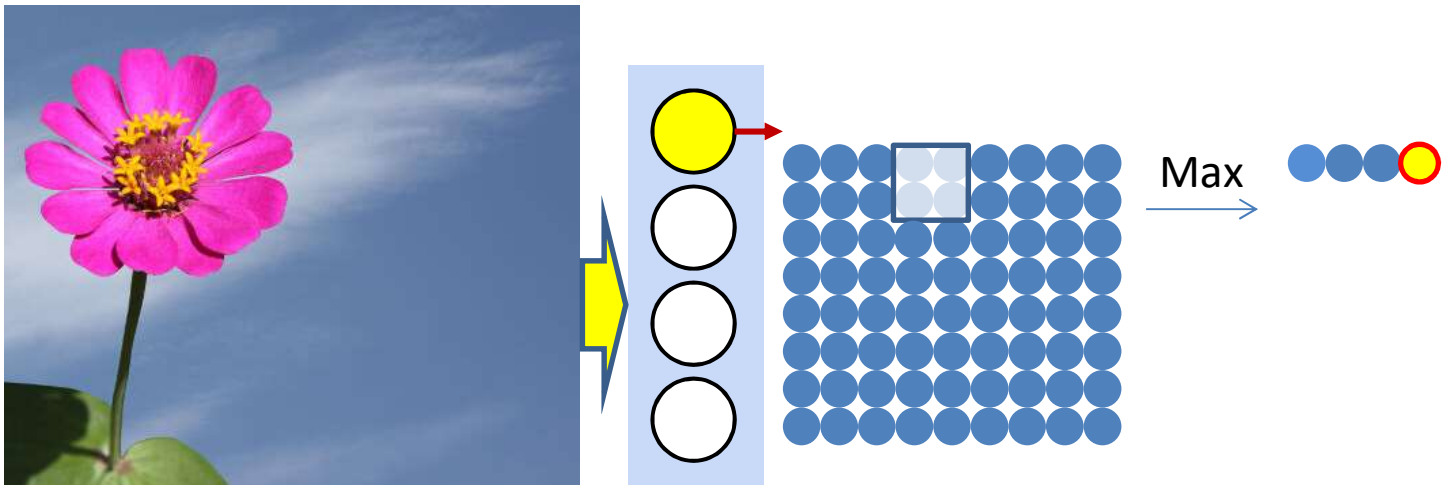
- The “max filter” operation too “scans” the picture

Accounting for jitter



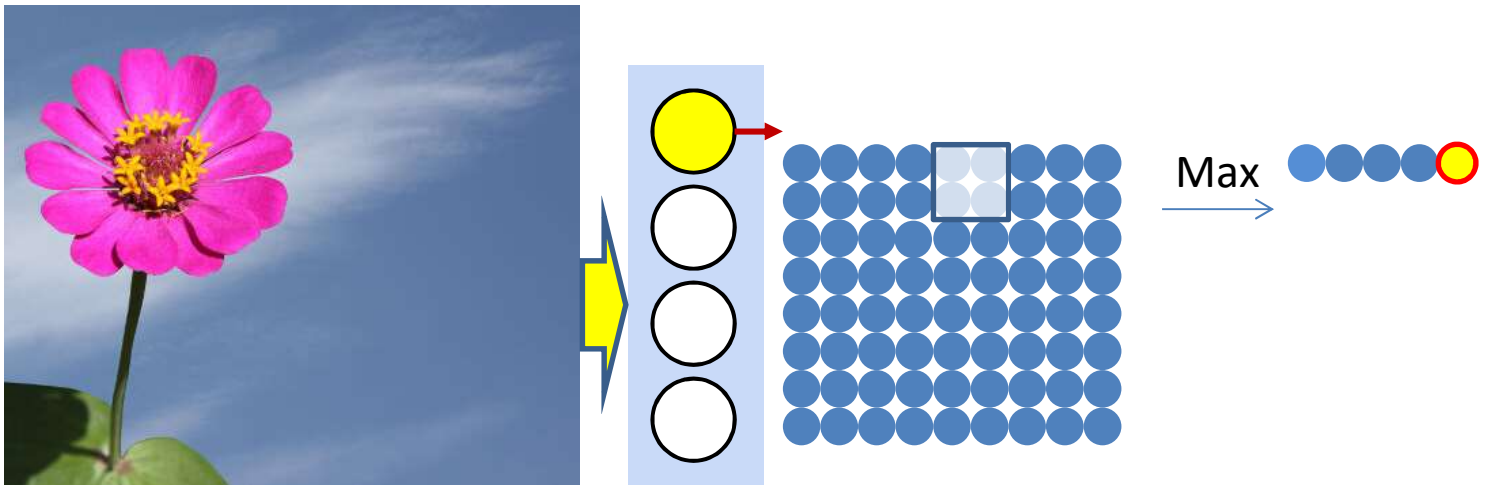
- The “max filter” operation too “scans” the picture

Accounting for jitter



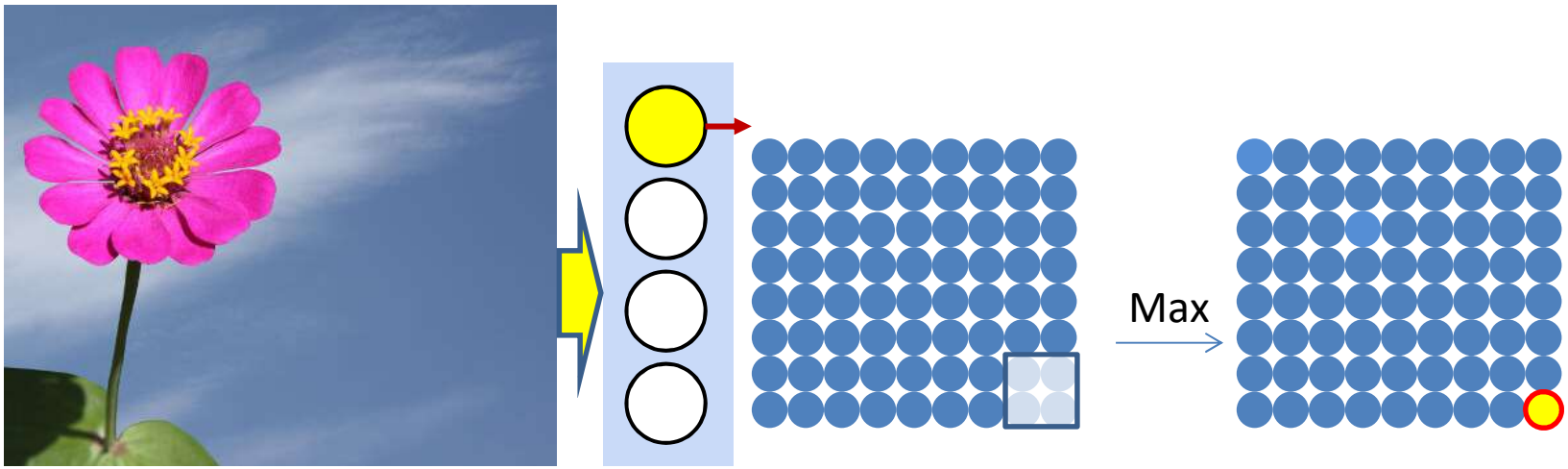
- The “max filter” operation too “scans” the picture

Accounting for jitter



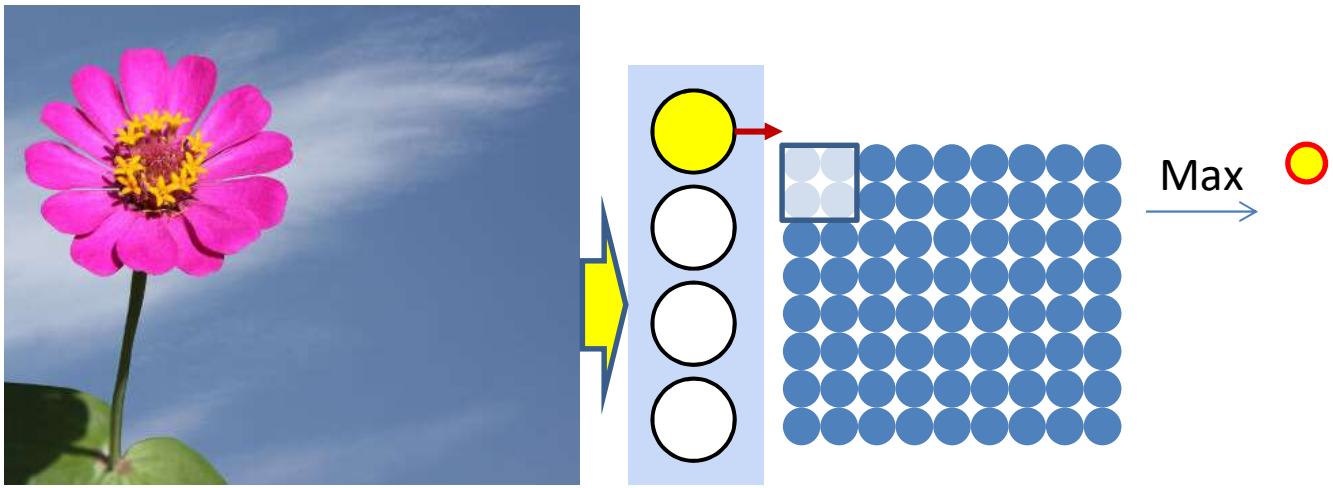
- The “max filter” operation too “scans” the picture

Accounting for jitter



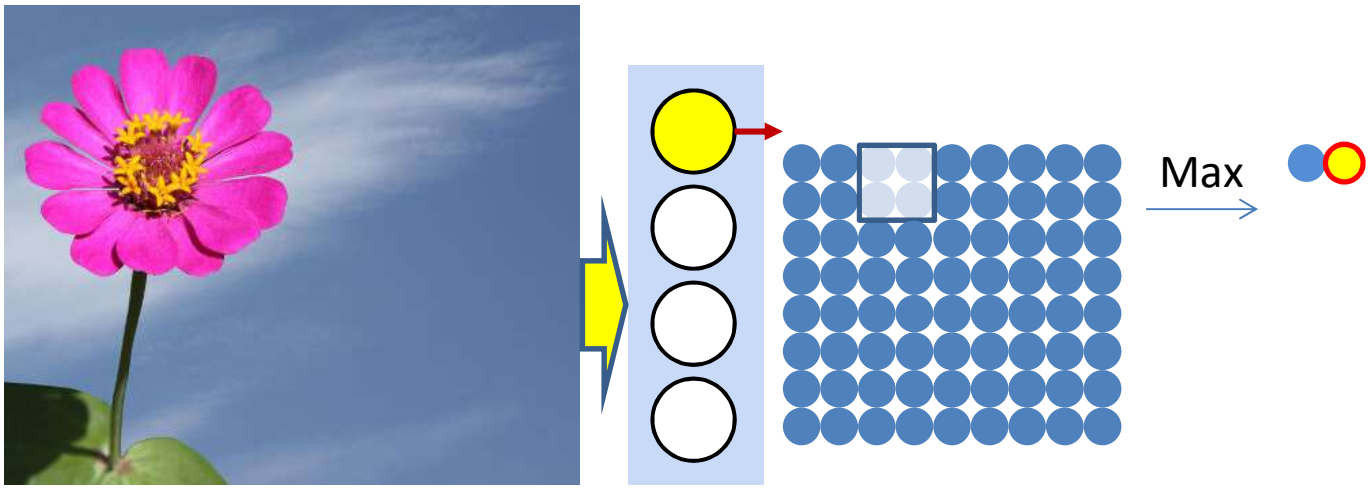
- The “max filter” operation too “scans” the picture

Max pooling “Strides”



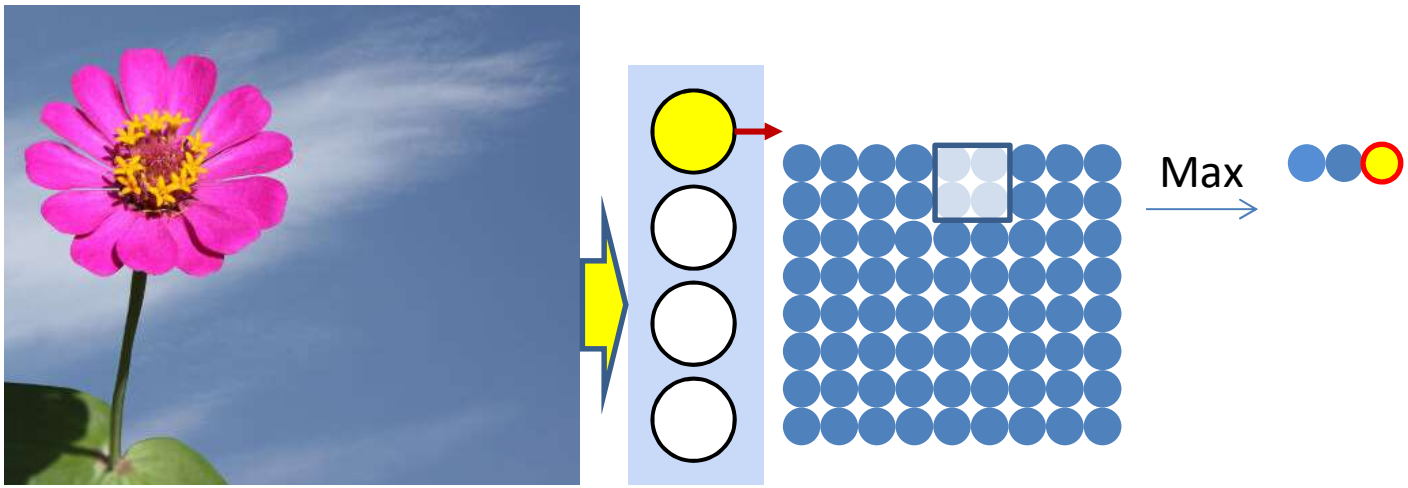
- The “max” operations may “stride” by more than one pixel

Max pooling “Strides”



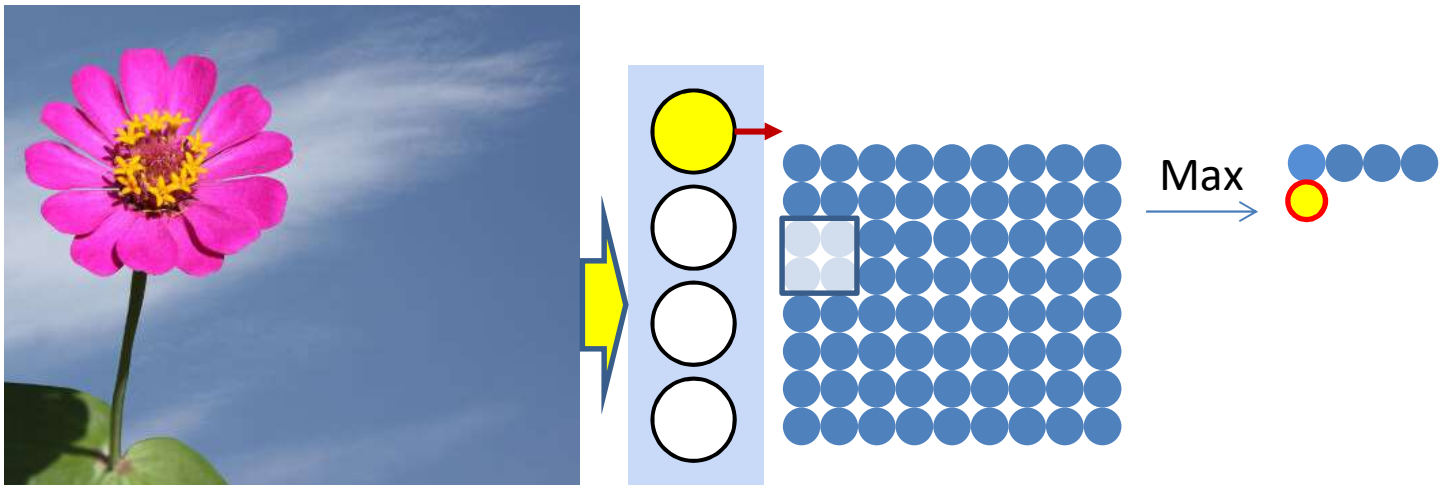
- The “max” operations may “stride” by more than one pixel

Max pooling “Strides”



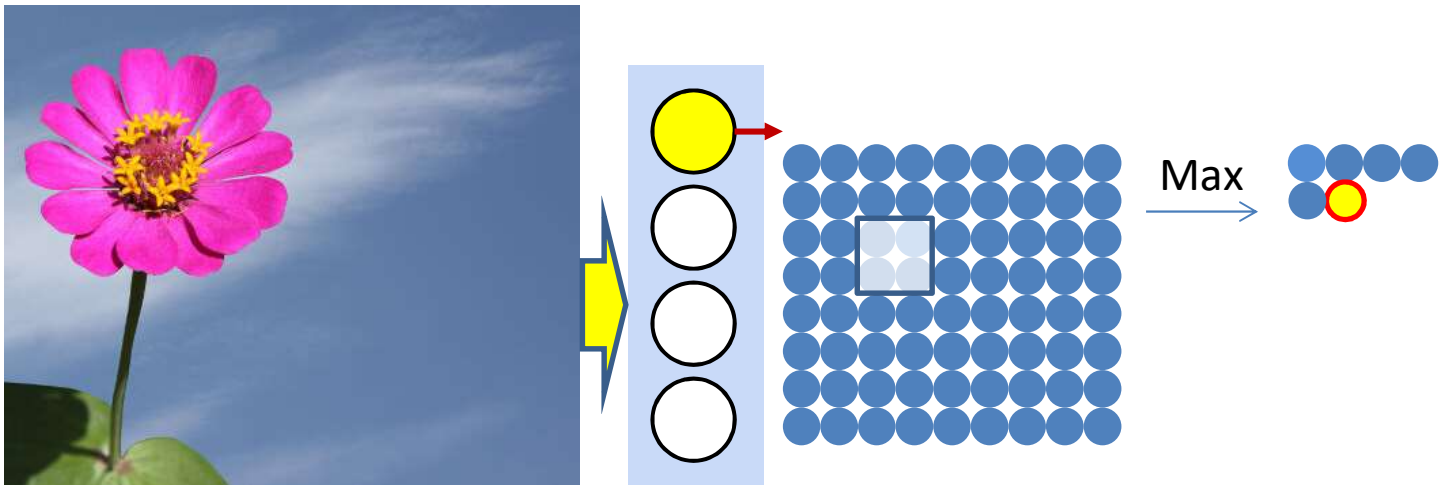
- The “max” operations may “stride” by more than one pixel

Max pooling “Strides”



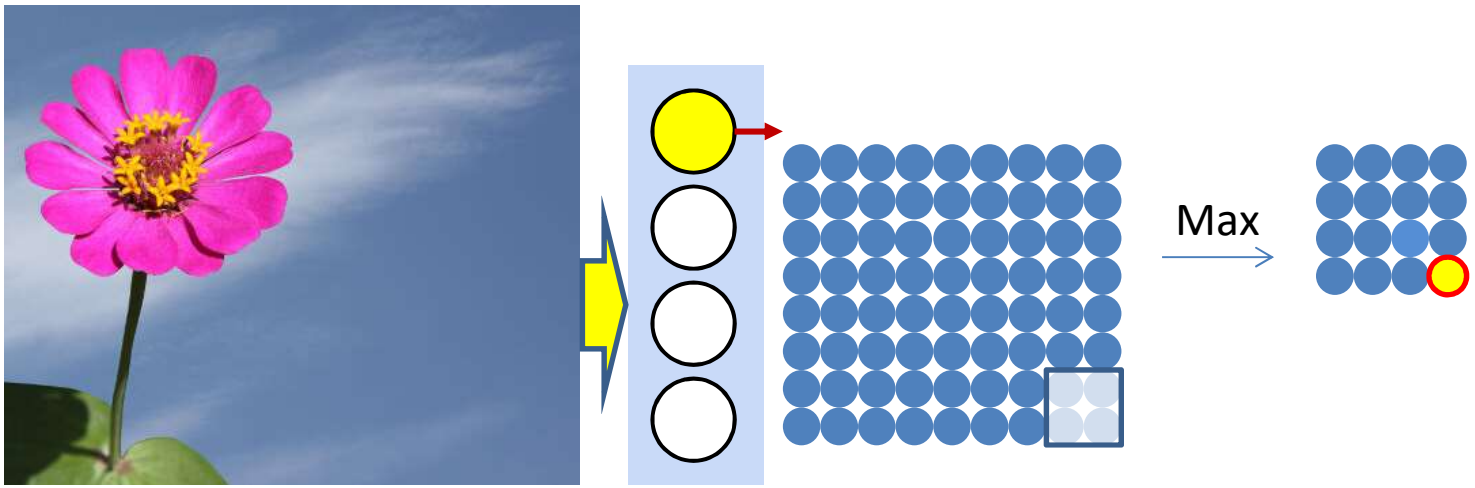
- The “max” operations may “stride” by more than one pixel

Max pooling “Strides”



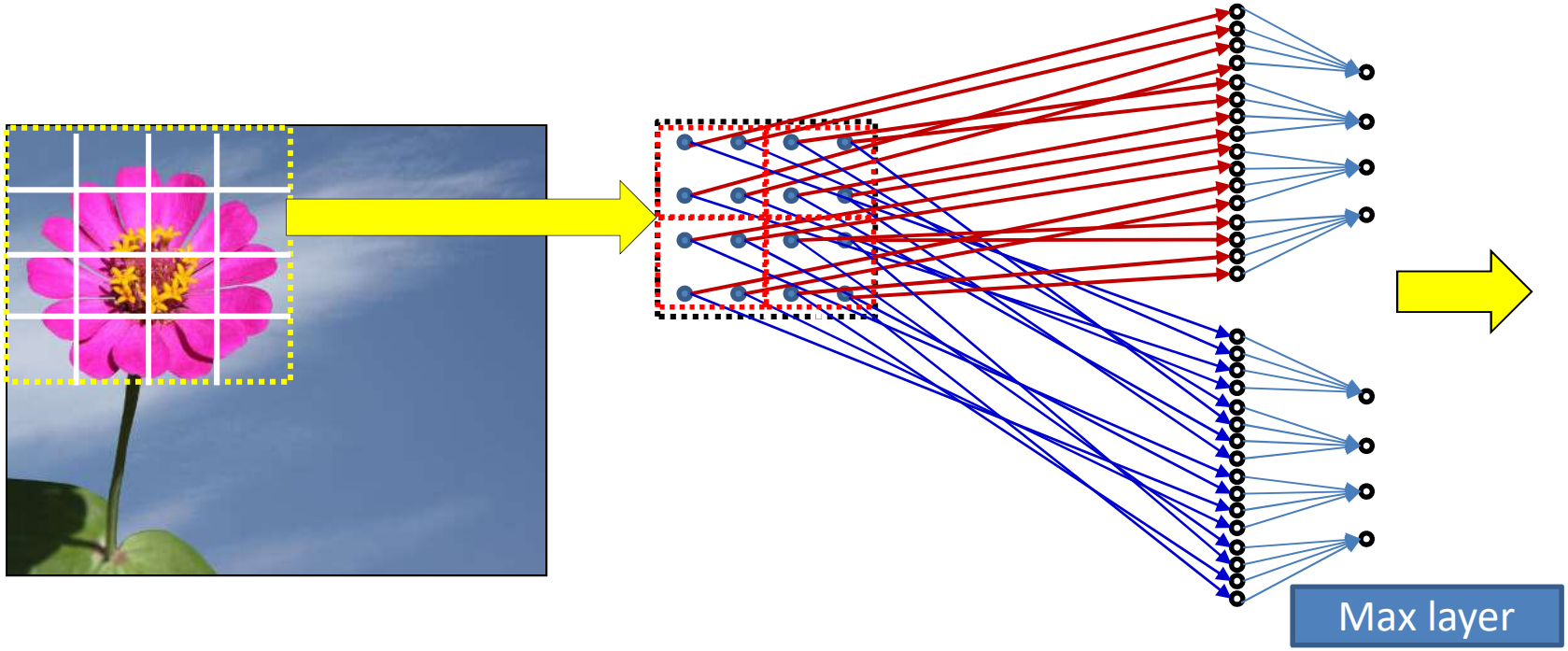
- The “max” operations may “stride” by more than one pixel

Max pooling “Strides”



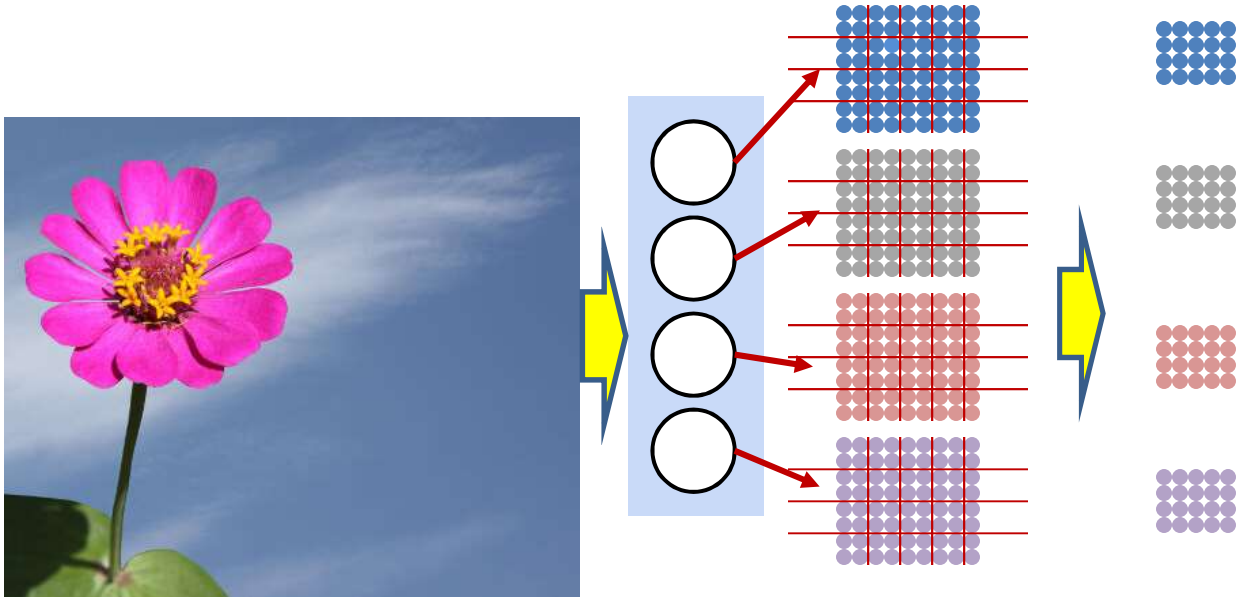
- The “max” operations may “stride” by more than one pixel
  - This will result in a *shrinking* of the map
  - The operation is usually called “pooling”
    - Pooling a number of outputs to get a single output
    - When stride is greater than 1, also called “Down sampling”

Shrinking with a max



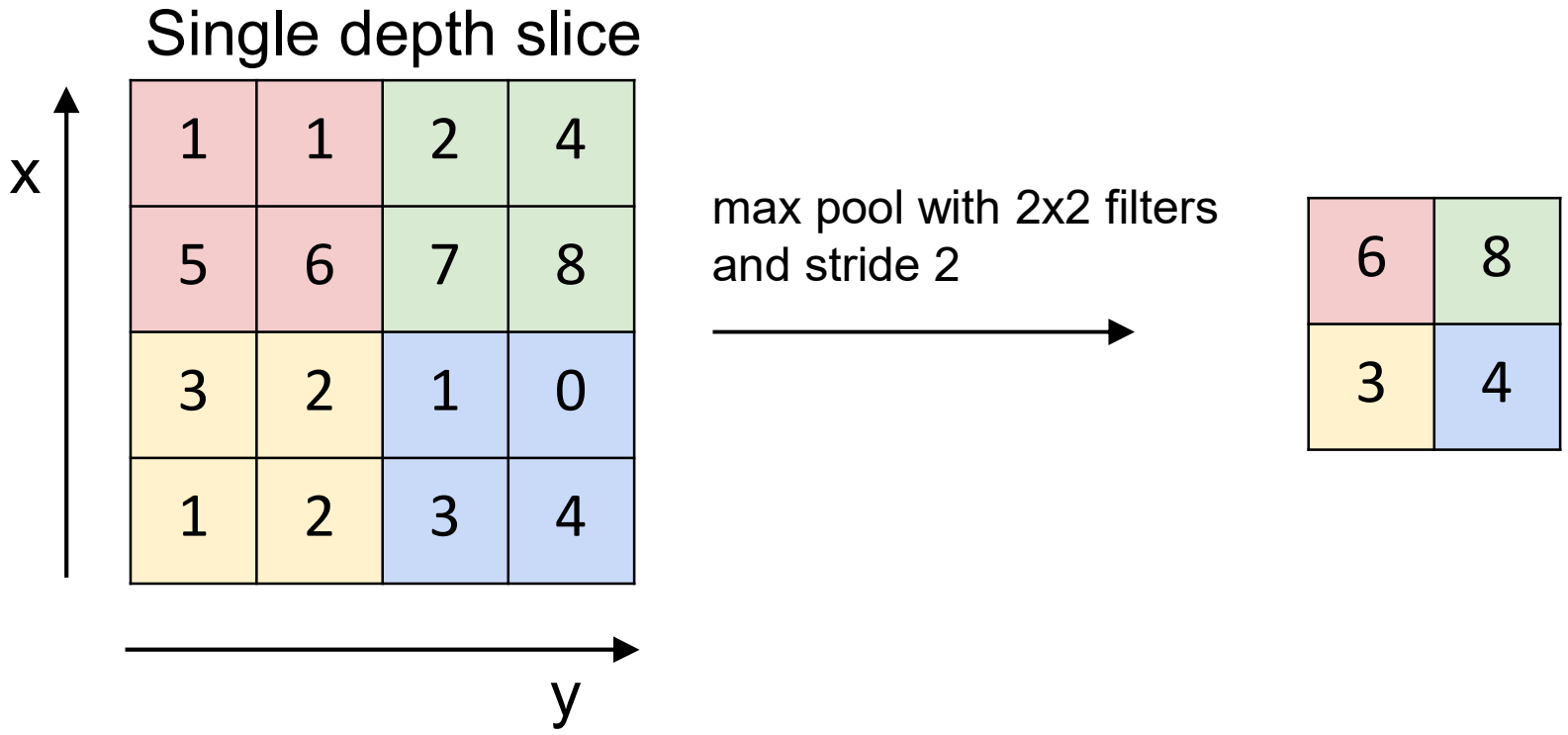
- In this example we *shrank* the image after the max
  - Adjacent “max” operators did not overlap
  - The stride was the size of the max filter itself

Non-overlapped strides

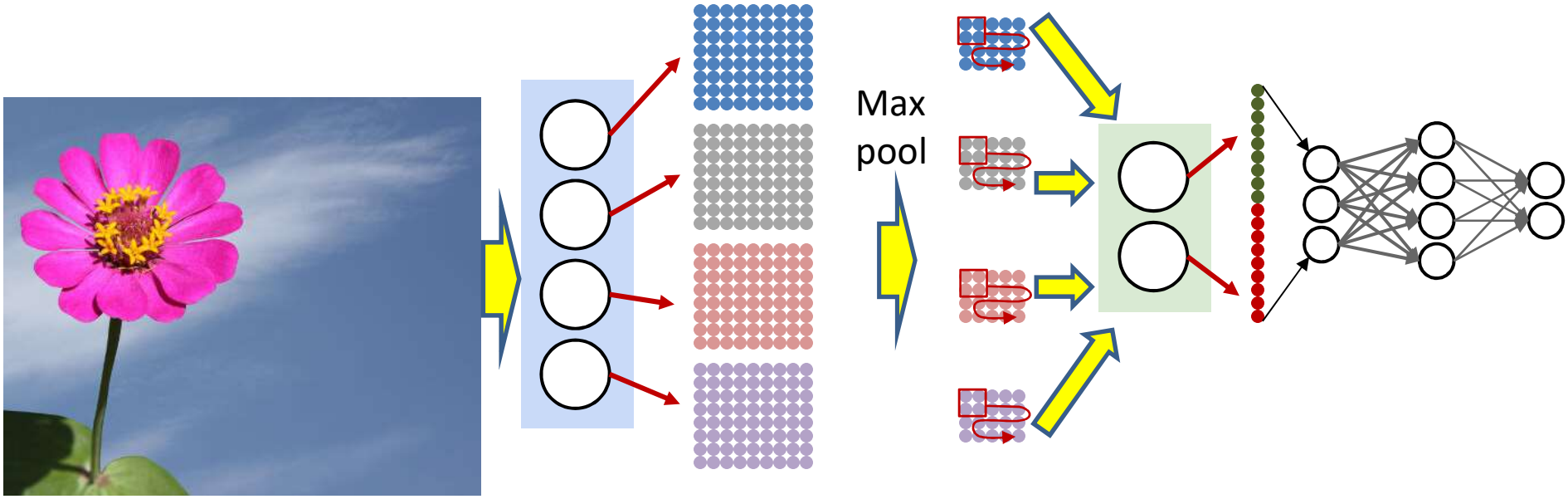


- Non-overlapping strides: Partition the output of the layer into blocks
- Within each block only retain the *highest* value
  - If you detect a petal anywhere in the block, a petal is detected..

Max Pooling

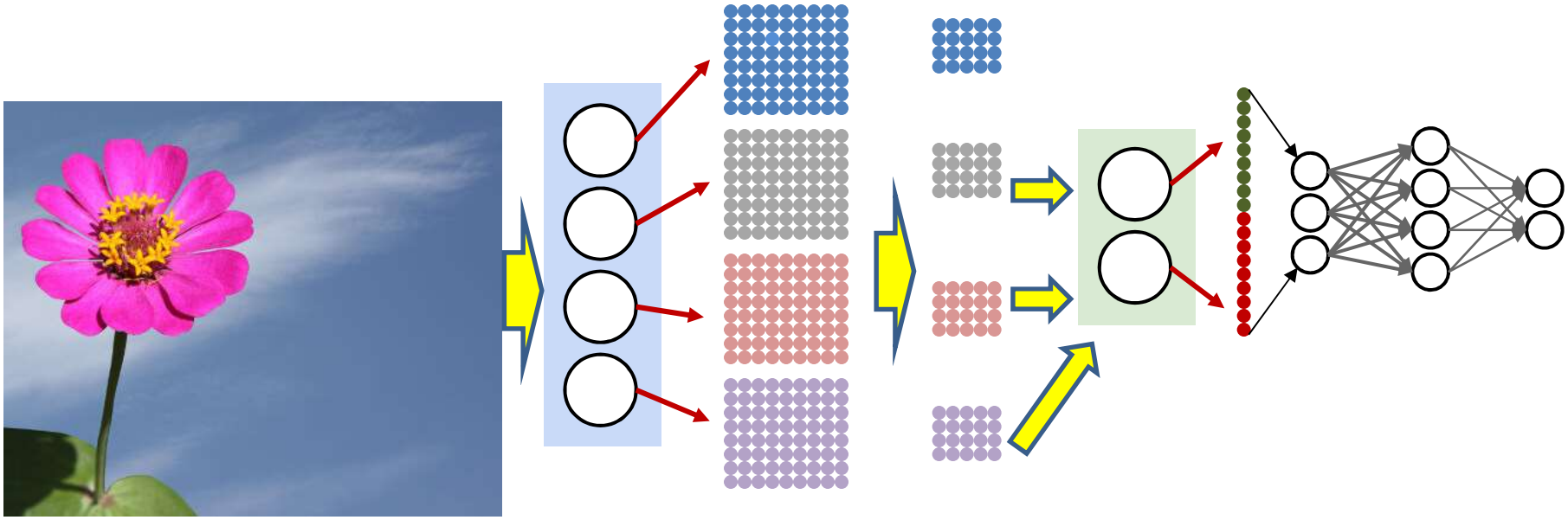

Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

Higher layers



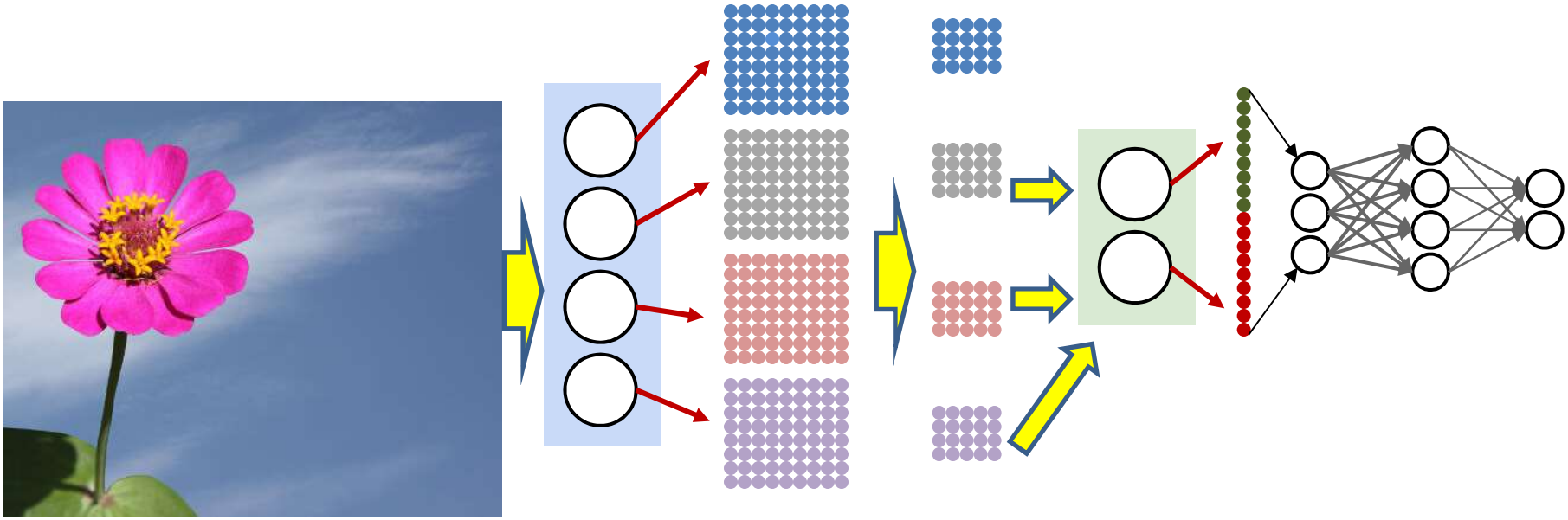
- The next layer works on the *max-pooled* maps

The overall structure



- In reality we can have many layers of “convolution” (scanning) followed by max pooling (and reduction) before the final MLP
  - The individual perceptrons at any “scanning” or “convolutional” layer are called “filters”
    - They “filter” the input image to produce an output image (map)
  - The individual *max* operations are also called *max pooling* or *max filters*

The overall structure



- This entire structure is called a ***Convolutional Neural Network***

Convolutional Neural Network