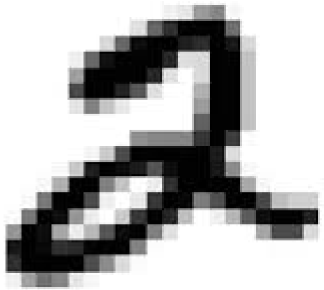
Deep Neural Networks
Scanning for patterns
(aka convolutional networks)

Bhiksha Raj

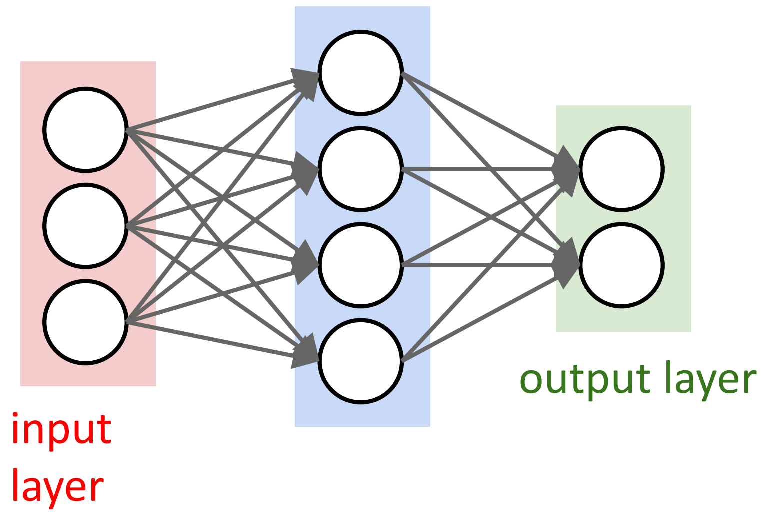
Story so far

- **MLPs are universal function approximators**
  - Boolean functions, classifiers, and regressions
- **MLPs can be trained through variations of gradient descent**
  - Gradients can be computed by backpropagation

The model so far

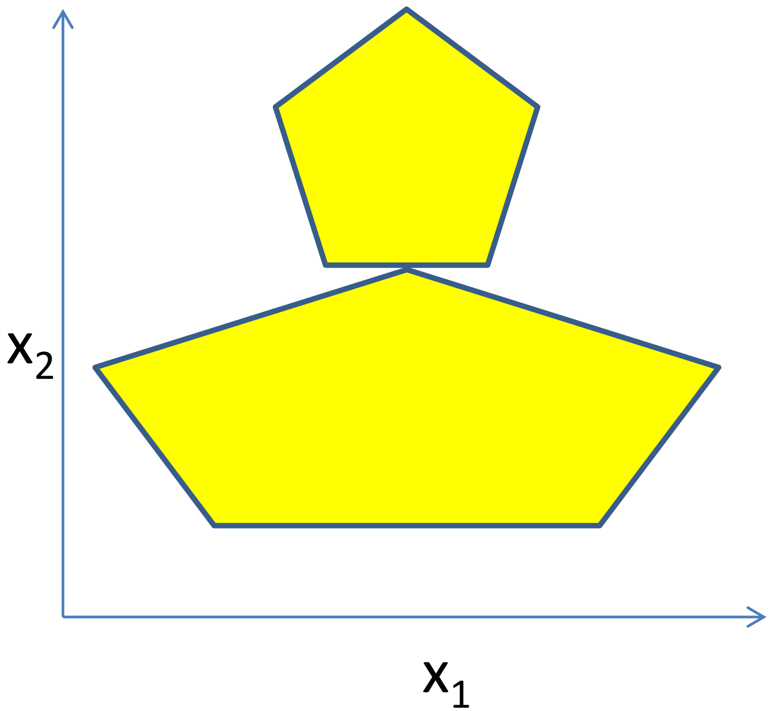


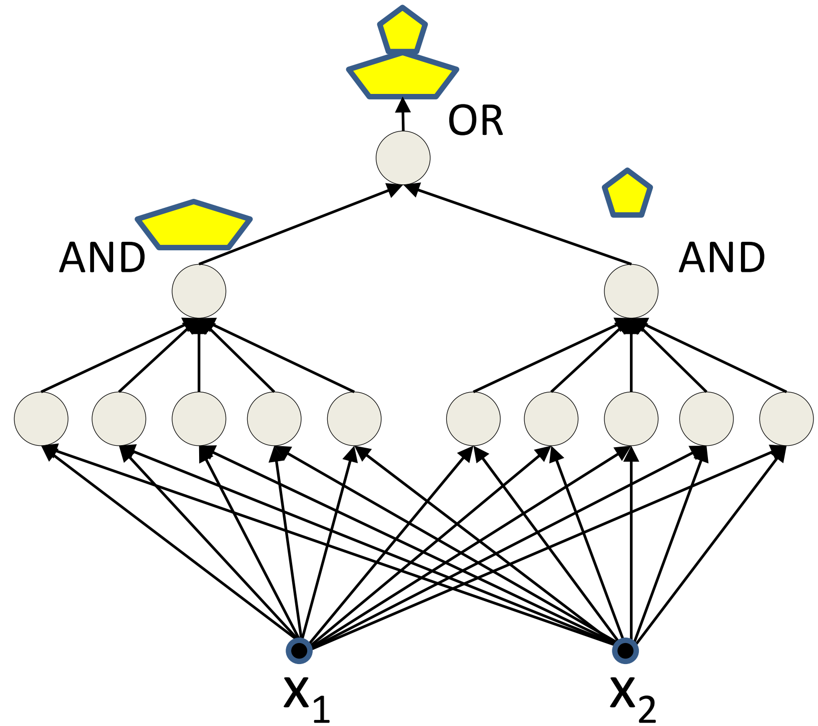
Or, more generally
a vector input



- Can recognize patterns in data
  - E.g. digits
  - Or any other vector data

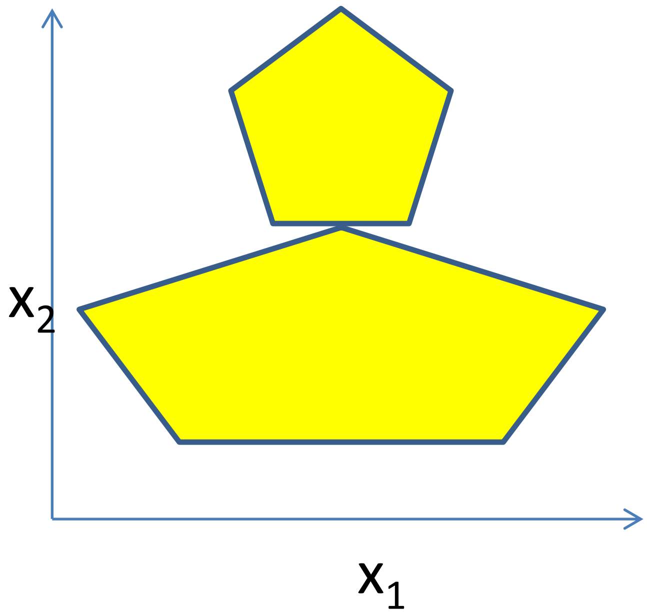
An important observation

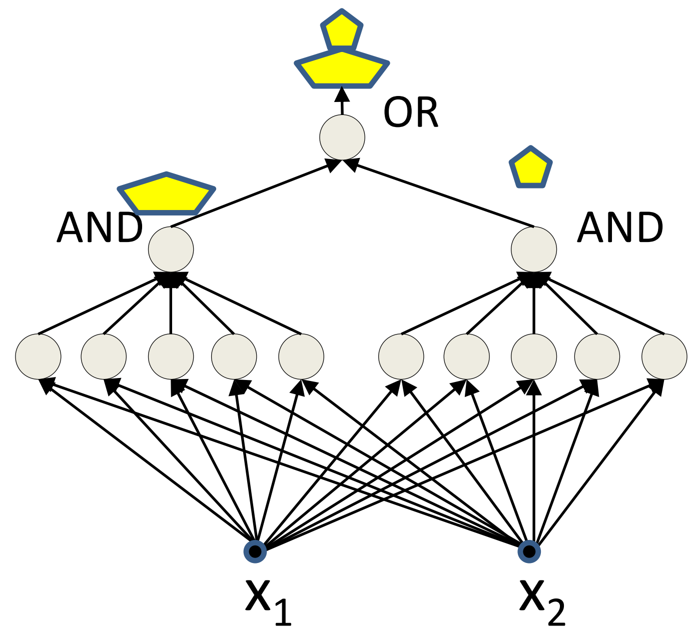




- The lowest layers of the network capture simple patterns
  - The linear decision boundaries in this example
- The next layer captures more complex patterns
  - The polygons
- The next one captures still more complex patterns..

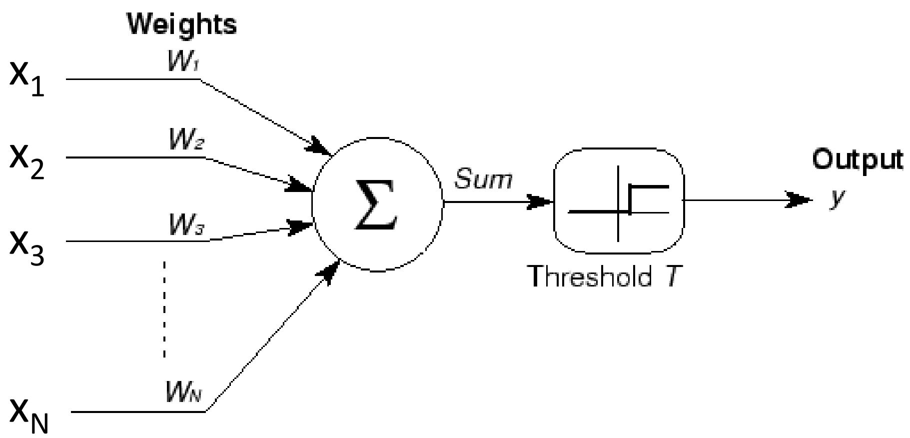
An important observation





- **The neurons in an MLP *build up* complex patterns from simple pattern hierarchically**
  – Each layer learns to "detect" simple combinations of the patterns detected by earlier layers
- **This is because the basic units themselves are simple**
  – Typically linear classifiers or thresholding units
  – Incapable of individually holding complex patterns

What do the neurons capture?



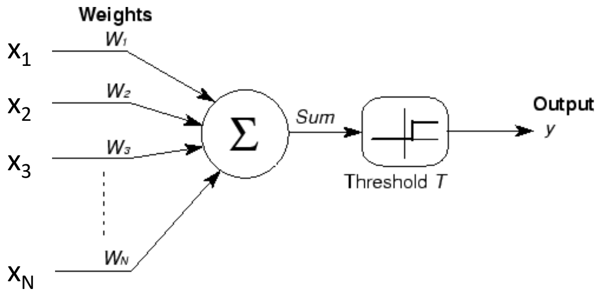
$$y = \begin{cases} 1 \; if \; \sum_i w_i x_i \geq T \\ 0 \; else \end{cases}$$

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

- To understand the behavior of neurons in the network, lets consider an individual perceptron
  - The perceptron is fully represented by its weights
  - For illustration, we consider a simple threshold activation

- What do the *weights* tell us?
  - *The perceptron "fires" if the* ***inner product*** *between the weights and the inputs exceeds a threshold*

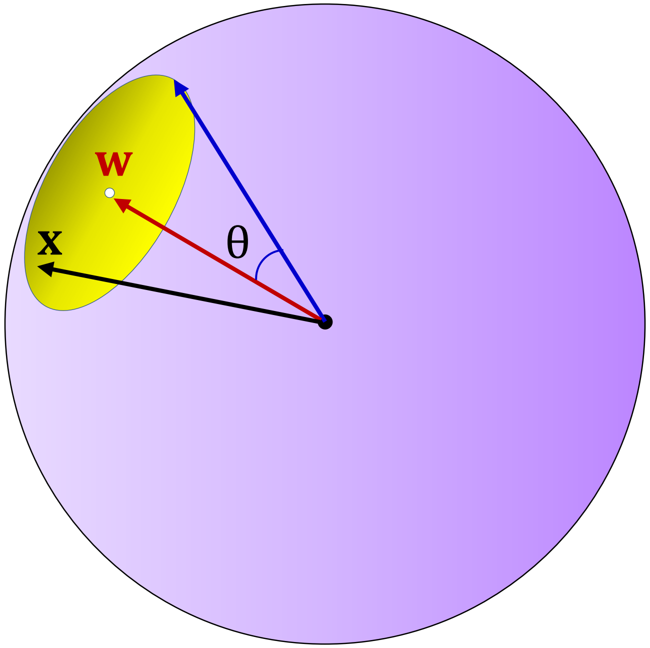
The weight as a "template"



$$\mathbf{x}^T\mathbf{w} > \mathrm{T}$$

$$\Rightarrow \cos\theta > \frac{\mathrm{T}}{|\mathbf{x}||\mathbf{w}|}$$

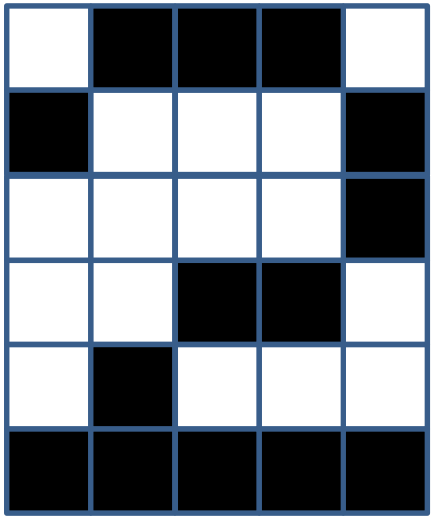
$$\Rightarrow \theta < \cos^{-1}\left(\frac{\mathrm{T}}{|\mathbf{x}||\mathbf{w}|}\right)$$



- A perceptron fires if its input is within a specified angle of its weight
  - Represents a convex region on the surface of the sphere!
- I.e. the perceptron fires if the input vector is close enough to the weight vector
  - If the input pattern matches the weight pattern closely enough

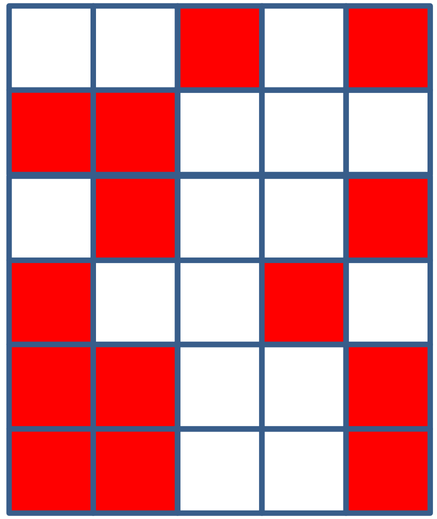
The weights as a correlation filter

W



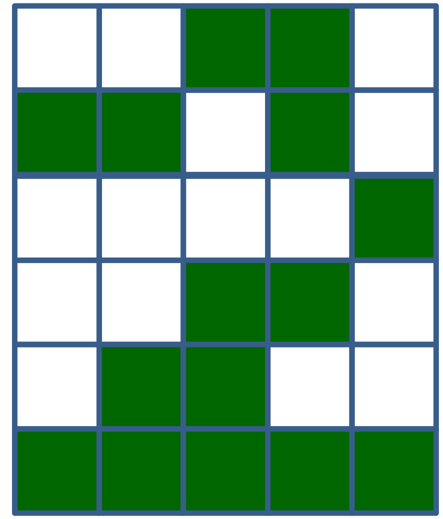
$$y = \begin{cases} 1 \; if \; \sum_i w_i x_i \geq T \\ 0 \; else \end{cases}$$

X



Correlation = 0.57

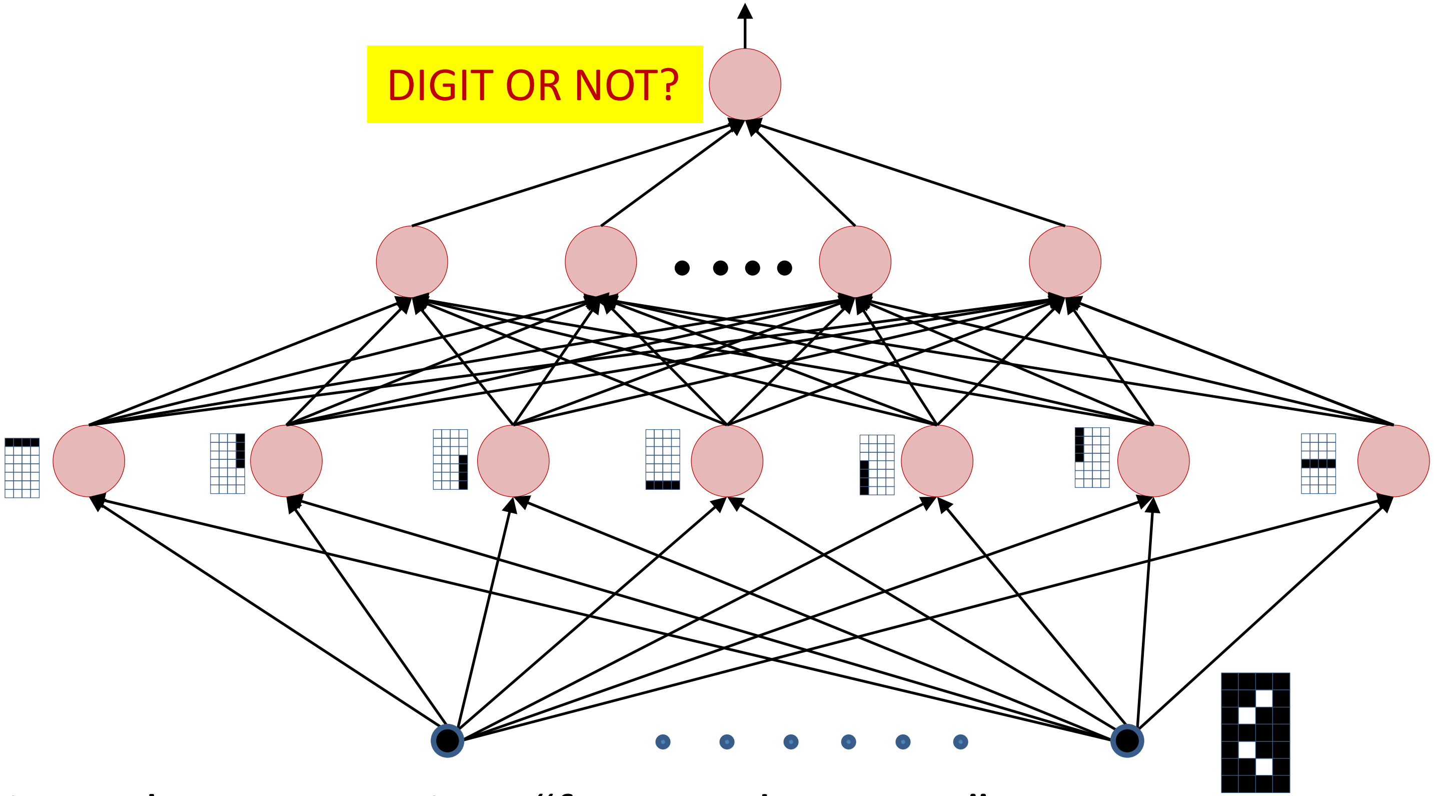
X



Correlation = 0.82

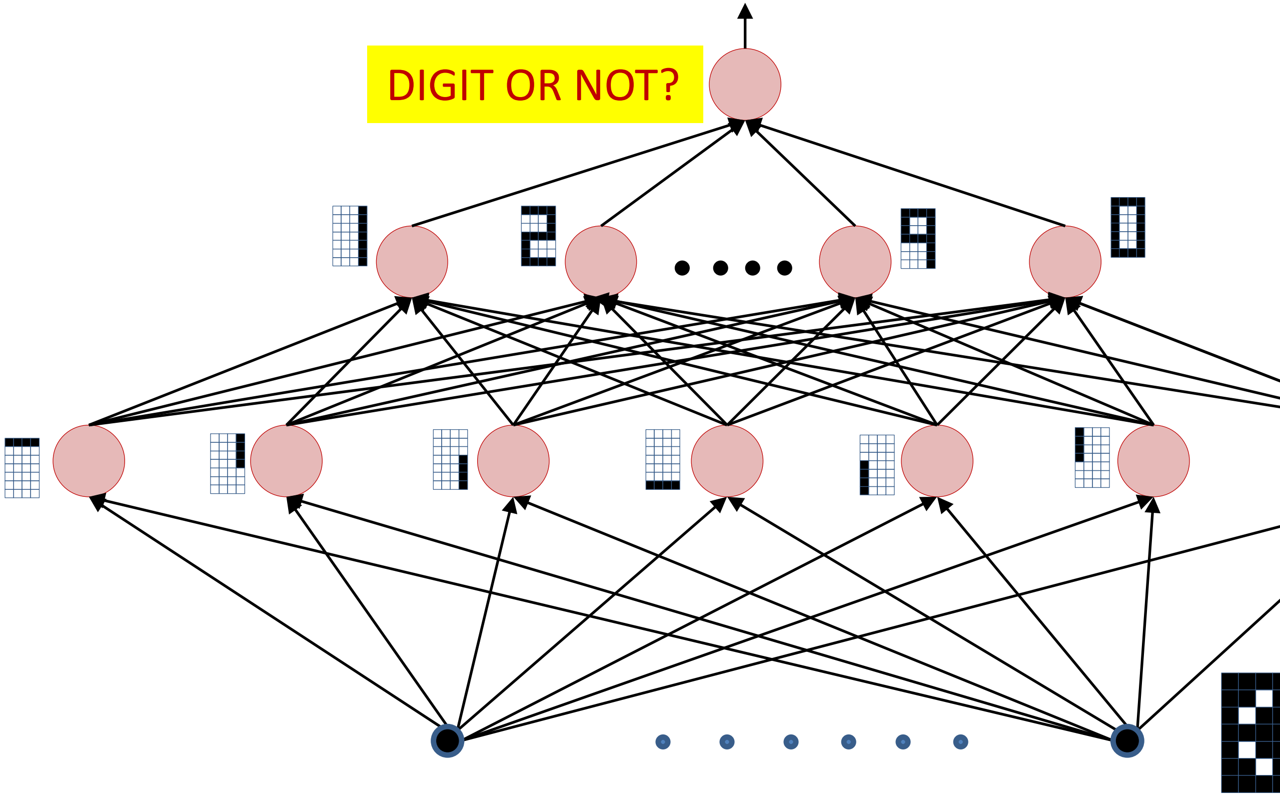
- If the *correlation* between the weight pattern and the inputs exceeds a threshold, fire
- The perceptron is a *correlation filter!*

The MLP as a Boolean function over feature detectors



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

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**
    - They OR or AND the patterns from the lower layer

Story so far

- **MLPs are Boolean machines**
  - They represent Boolean functions over linear boundaries
  - They can represent arbitrary boundaries

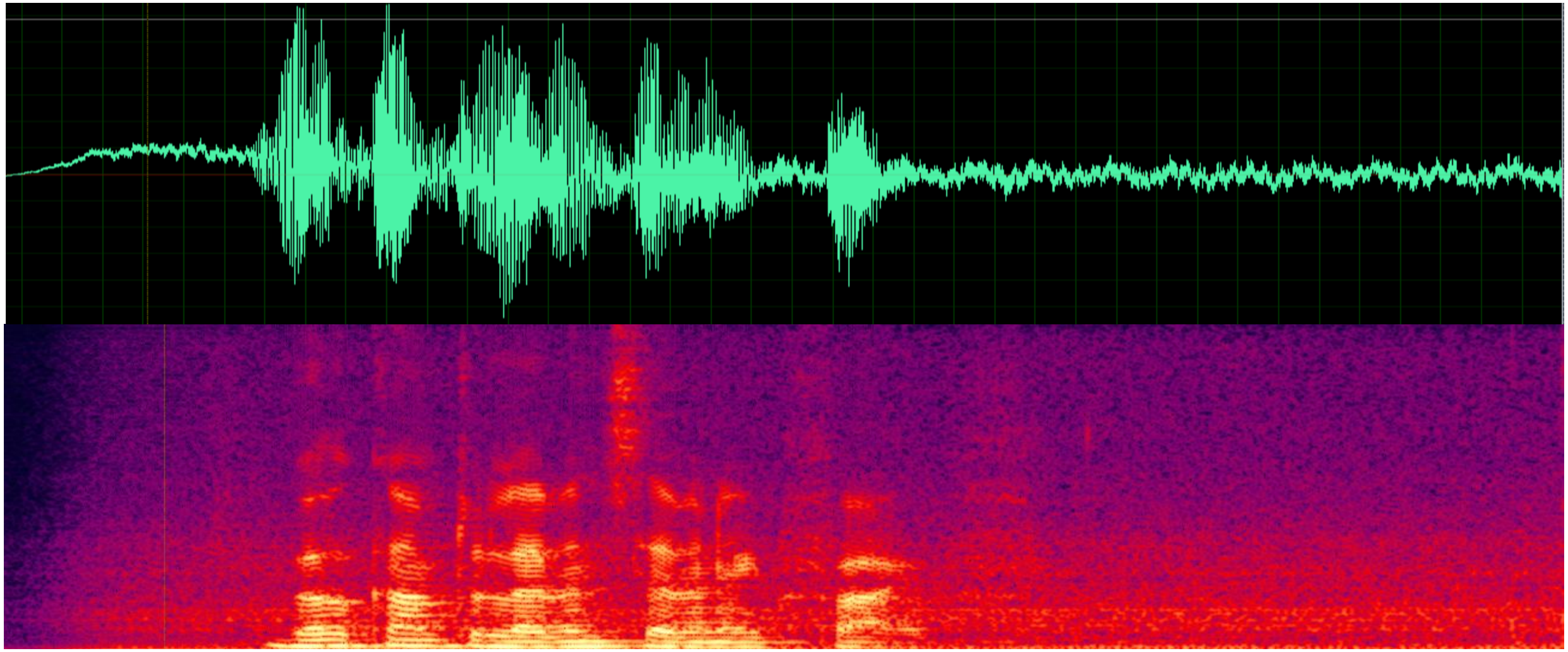
- **Perceptrons are correlation filters**
  - They detect patterns in the input

- **Layers in an MLP are detectors of increasingly complex patterns**
  - Patterns of lower-complexity patterns

- **MLP in classification**
  - The network will fire if the combination of the detected basic features matches an "acceptable" pattern for a desired class of signal
    - E.g. Appropriate combinations of (Nose, Eyes, Eyebrows, Cheek, Chin) → Face

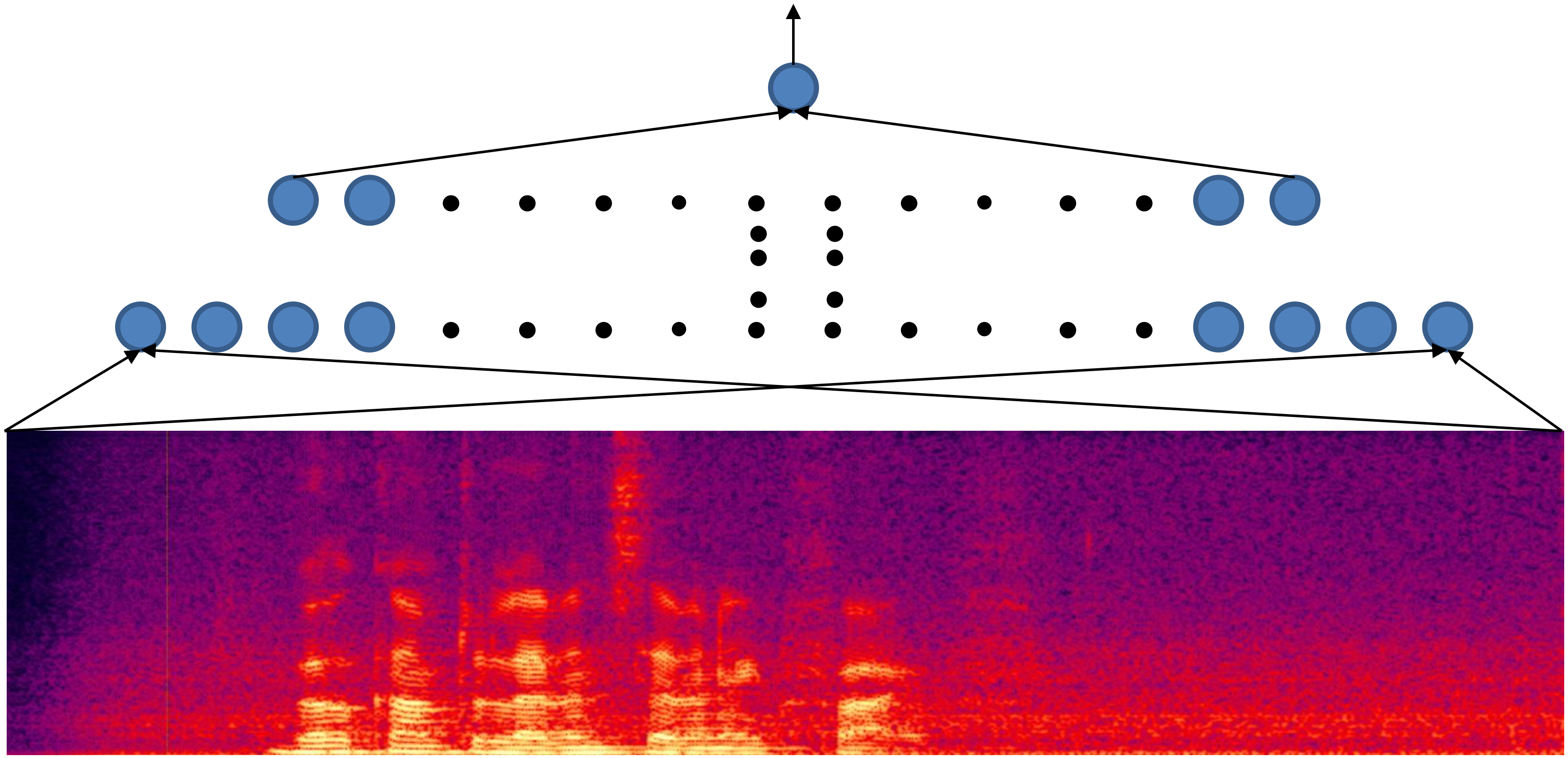
Changing gears..

A problem



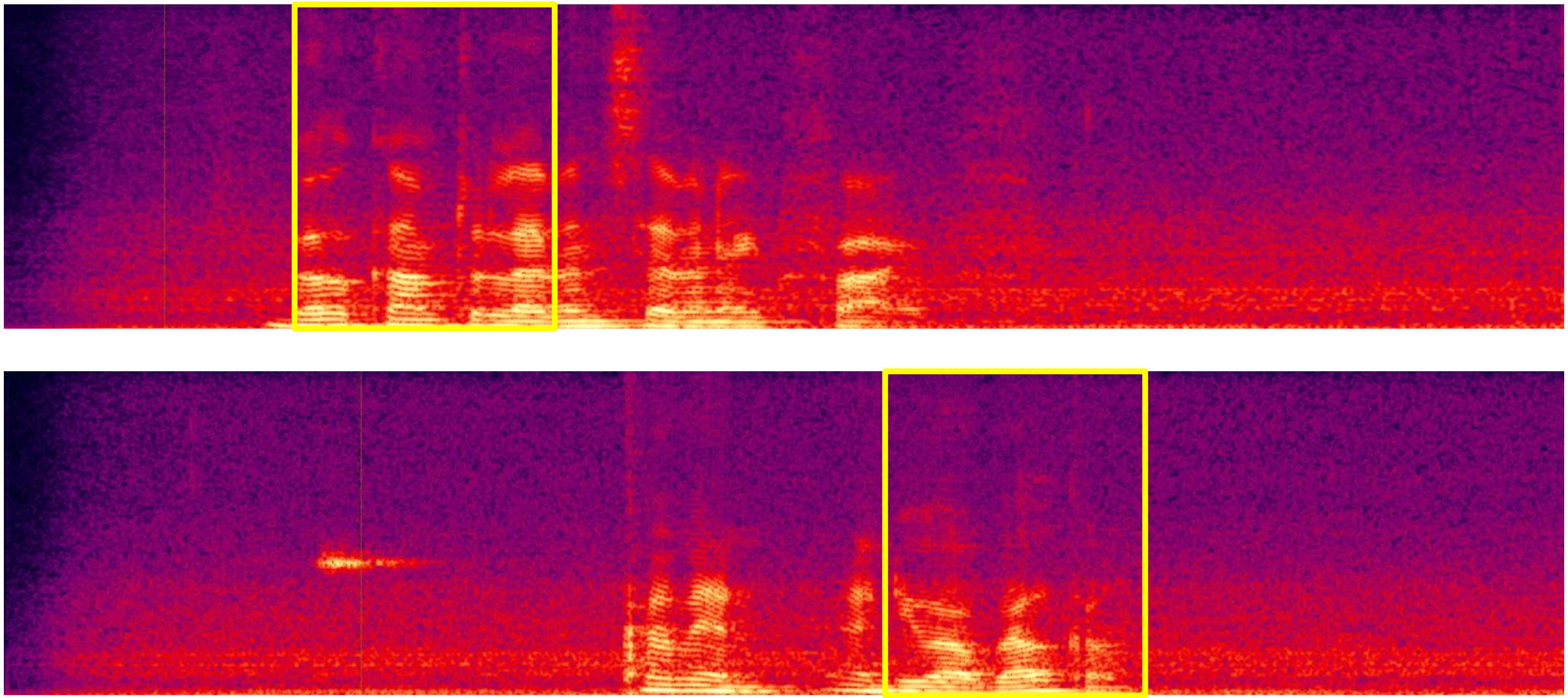
- Does this signal contain the word “Welcome”?
- Compose an MLP for this problem.
  – Assuming all recordings are exactly the same length..

Finding a Welcome



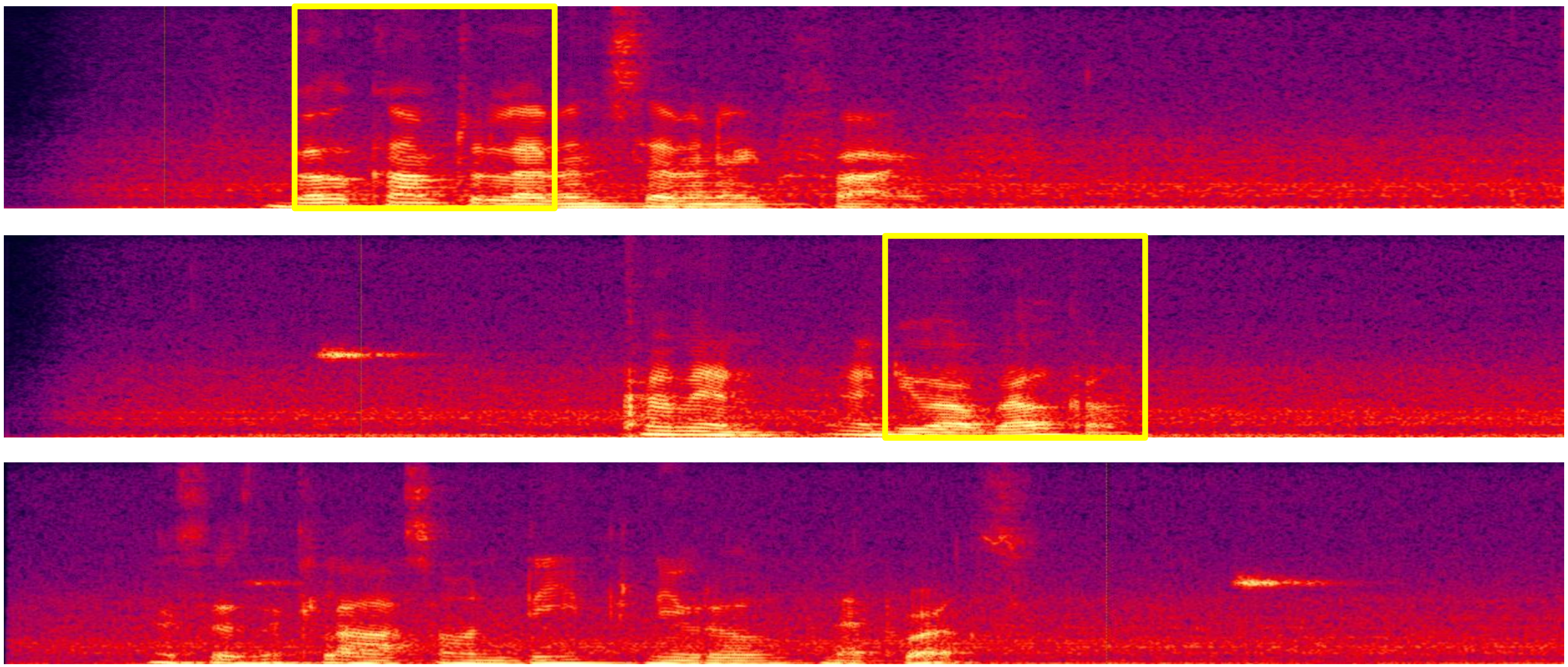
- Trivial solution: Train an MLP for the entire recording

Finding a Welcome



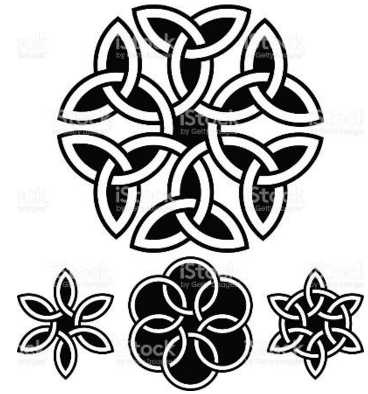
- Problem with trivial solution: Network that finds a "welcome" in the top recording will not find it in the lower one
  - Unless trained with both
  - Will require a very large network and a large amount of training data to cover every case

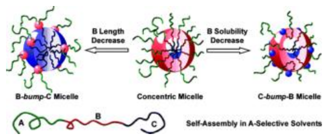
Finding a Welcome

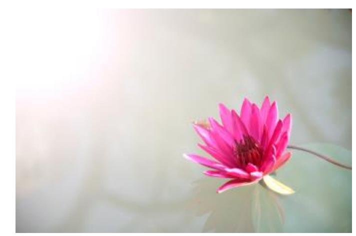


- Need a *simple* network that will fire regardless of the location of “Welcome”
  - and not fire when there is none

Flowers

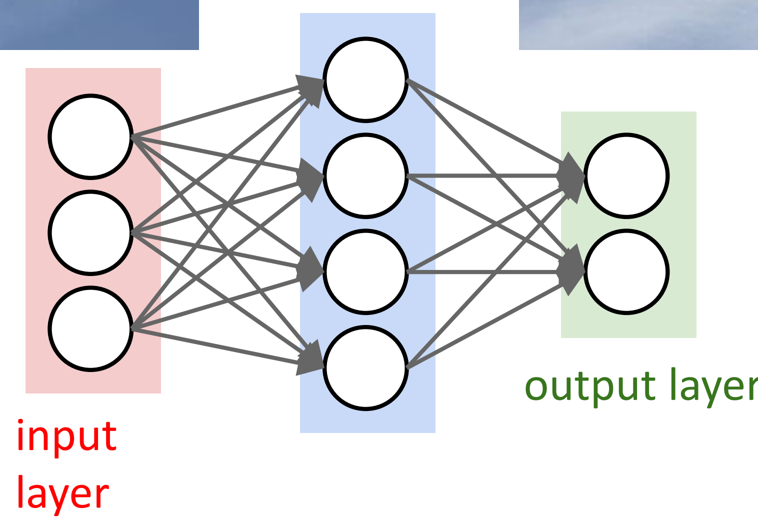






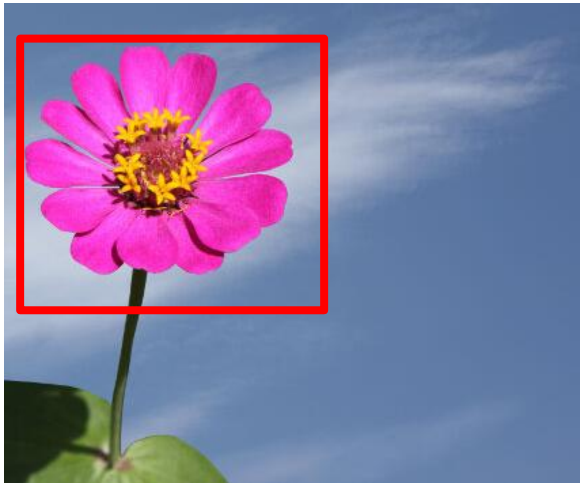
- Is there a flower in any of these images

A problem



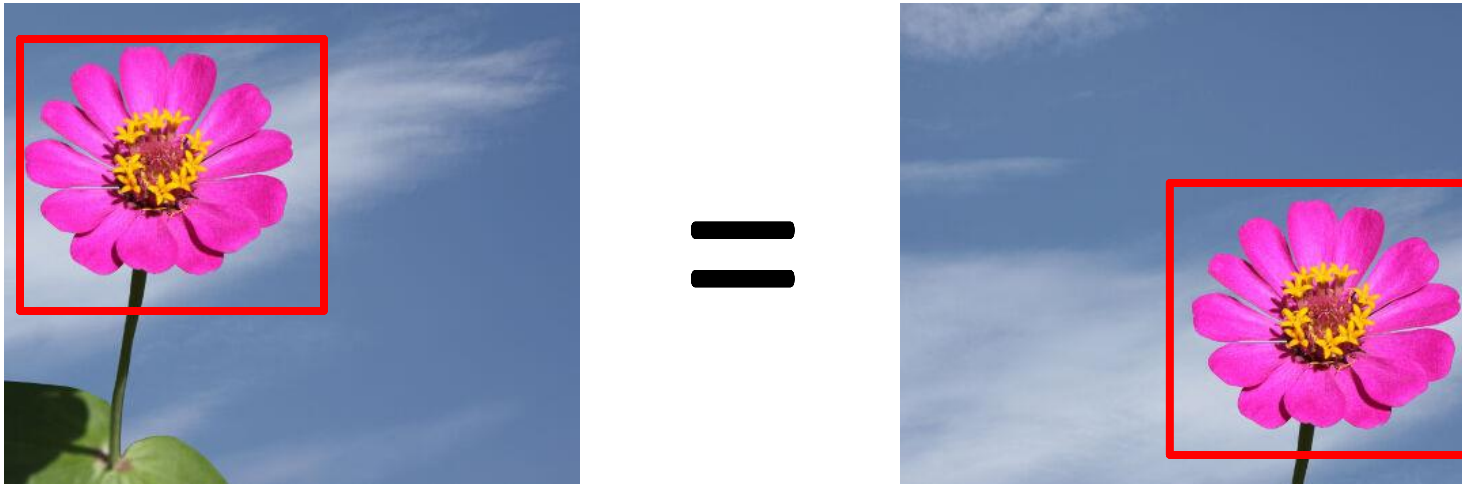
- Will an MLP that recognizes the left image as a flower also recognize the one on the right as a flower?

A problem



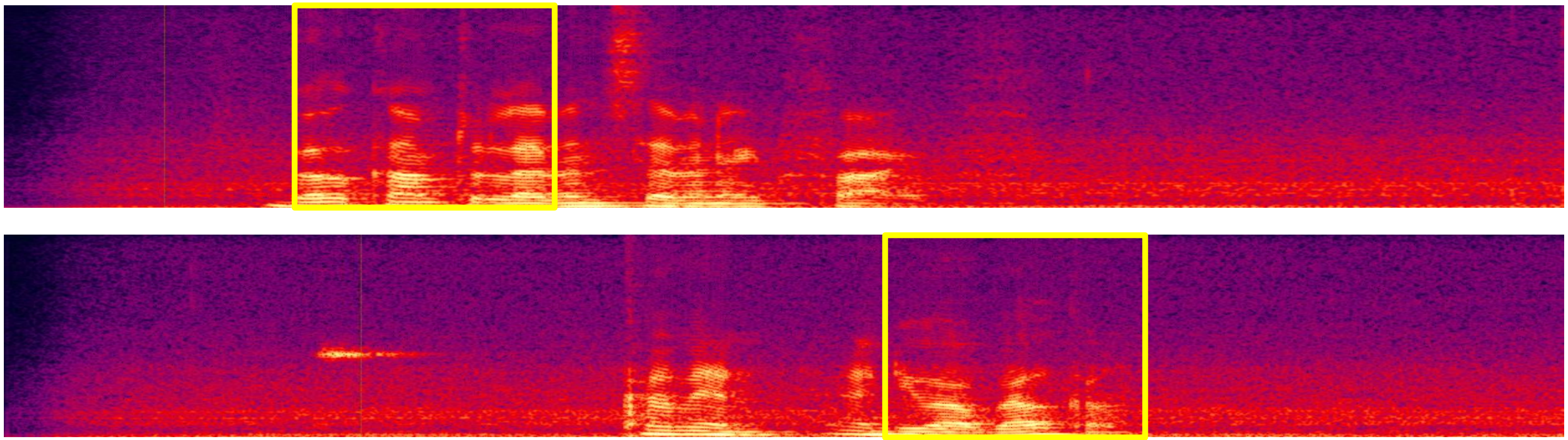
- Need a network that will "fire" regardless of the precise location of the target object

The need for *shift invariance*



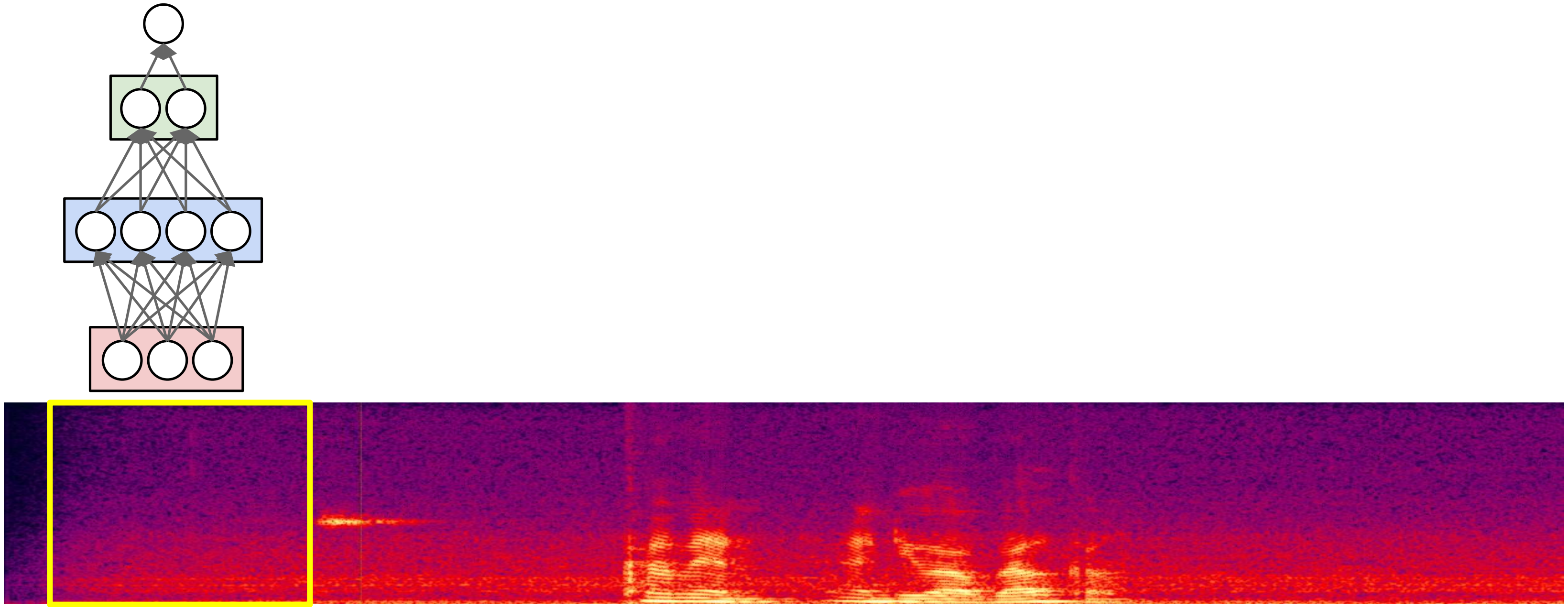
- In many problems the *location* of a pattern is not important
  - Only the presence of the pattern
- Conventional MLPs are sensitive to the location of the pattern
  - Moving it by one component results in an entirely different input that the MLP wont recognize
- Requirement: Network must be *shift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

The need for *shift invariance*



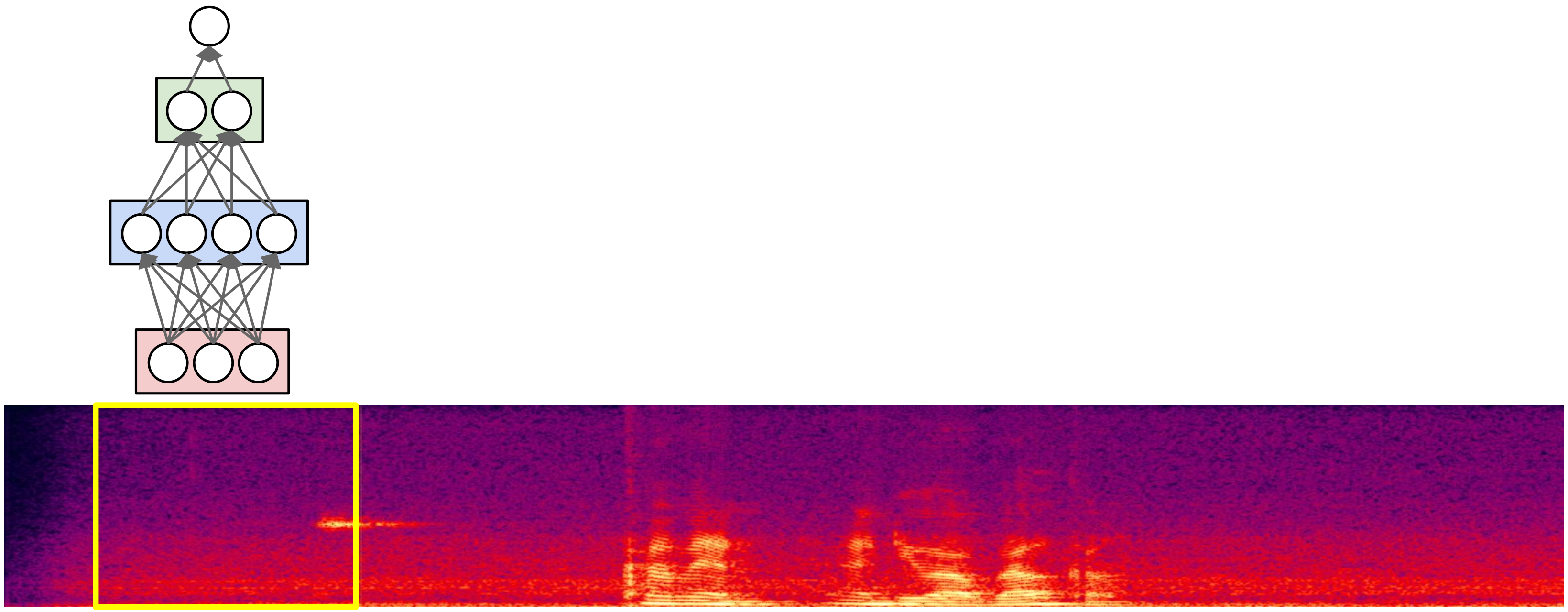
- In many problems the *location* of a pattern is not important
  - Only the presence of the pattern
- Conventional MLPs are sensitive to the location of the pattern
  - Moving it by one component results in an entirely different input that the MLP wont recognize
- Requirement: Network must be *shift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

Solution: Scan



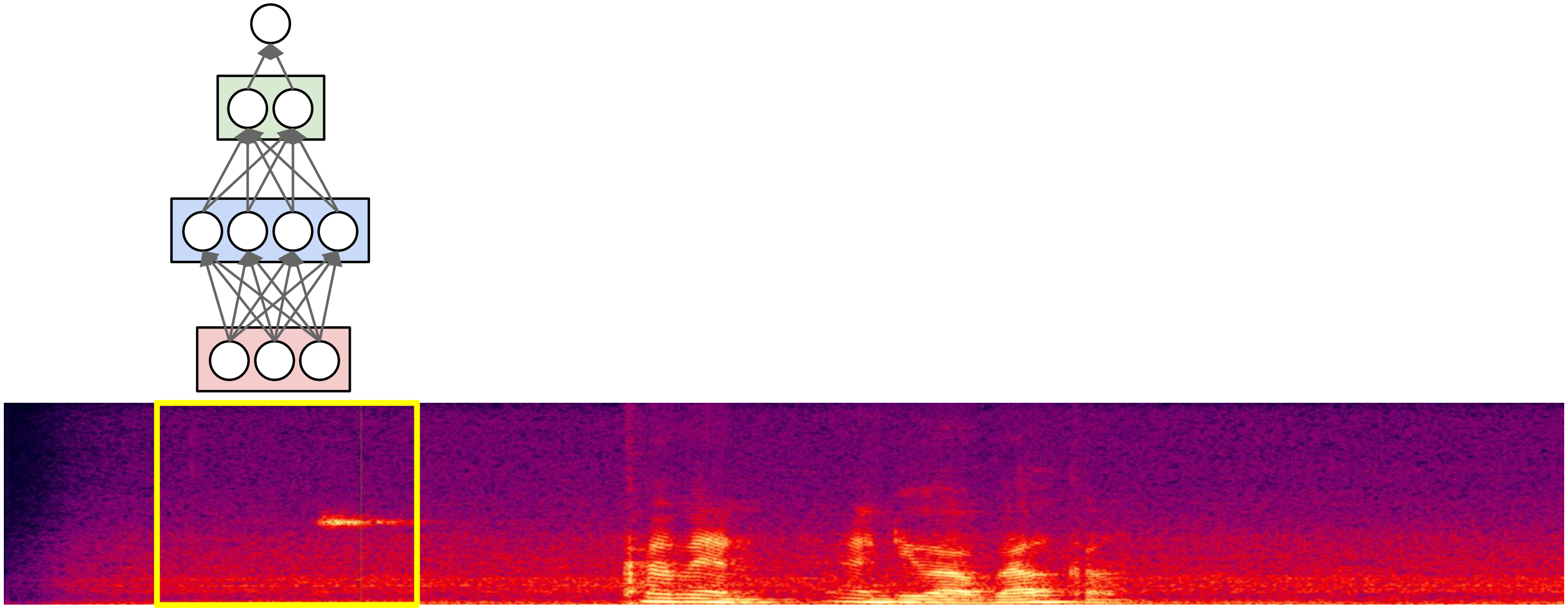
- Scan for the target word
  - The spectral time-frequency components in a "window" are input to a "welcome-detector" MLP

Solution: Scan



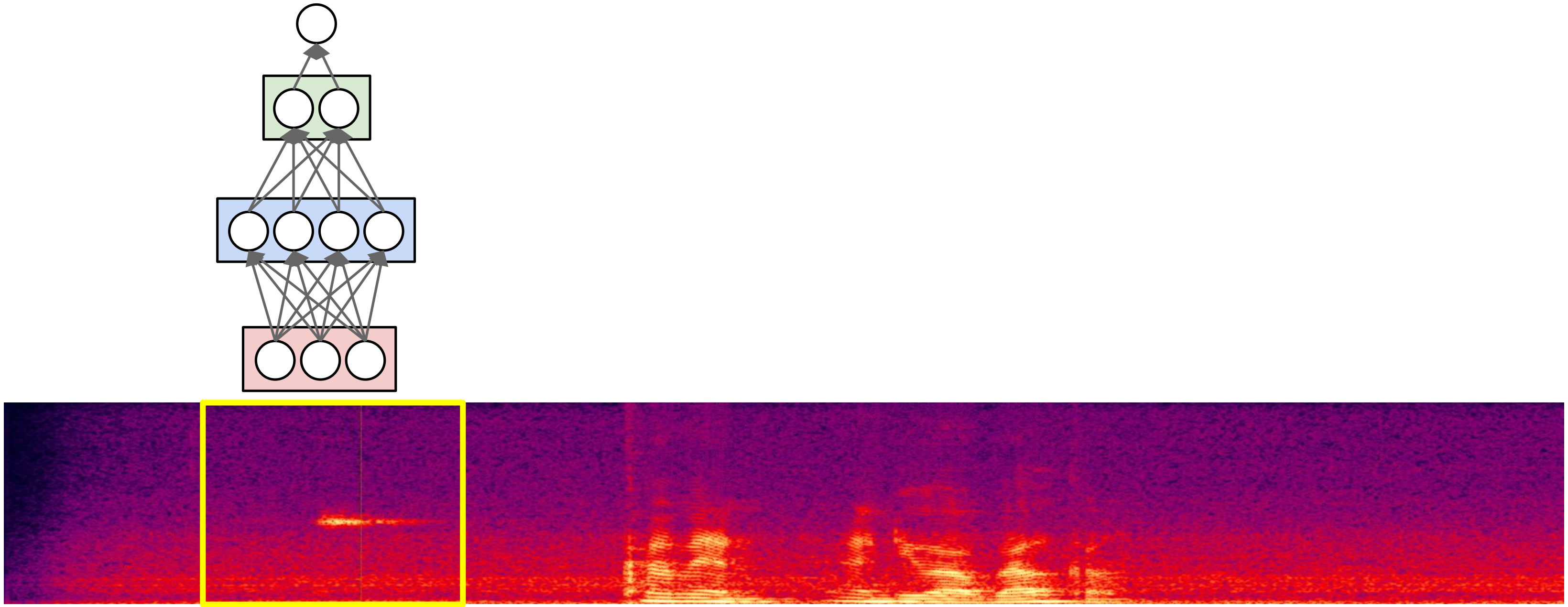
- Scan for the target word
  - The spectral time-frequency components in a "window" are input to a "welcome-detector" MLP

Solution: Scan



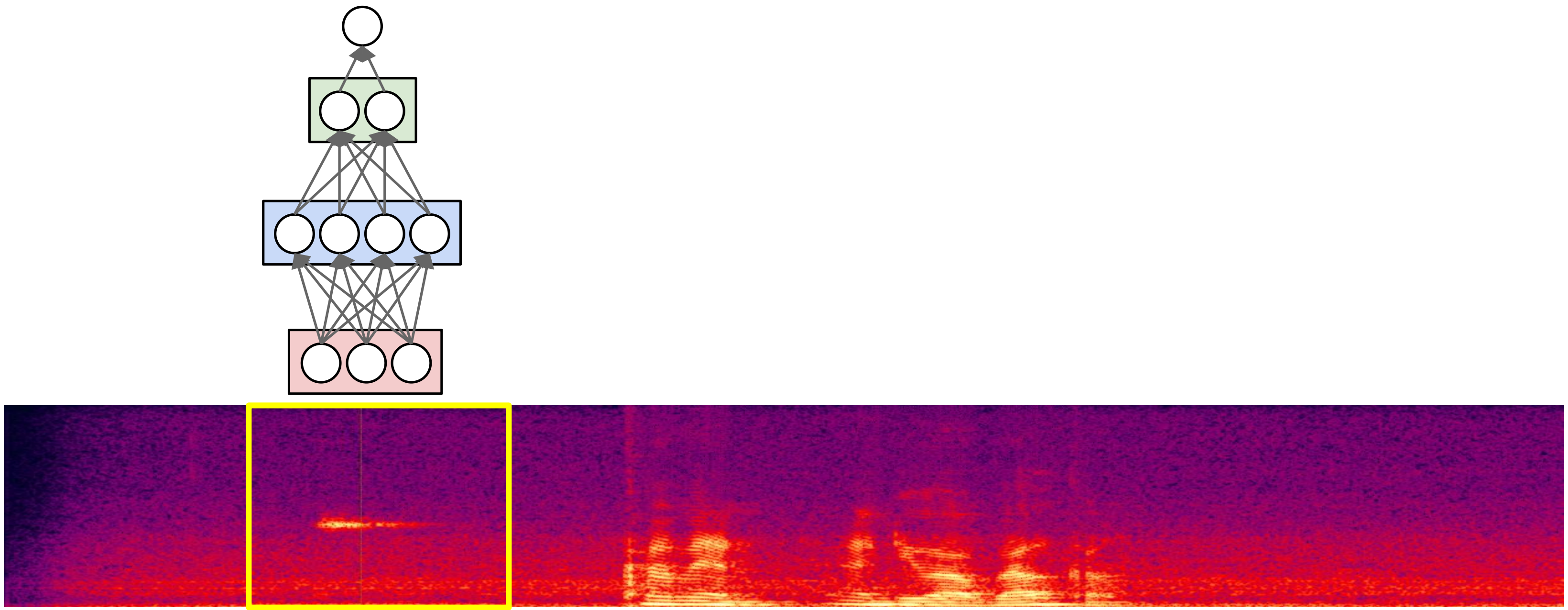
- Scan for the target word
  - The spectral time-frequency components in a “window” are input to a “welcome-detector” MLP

Solution: Scan



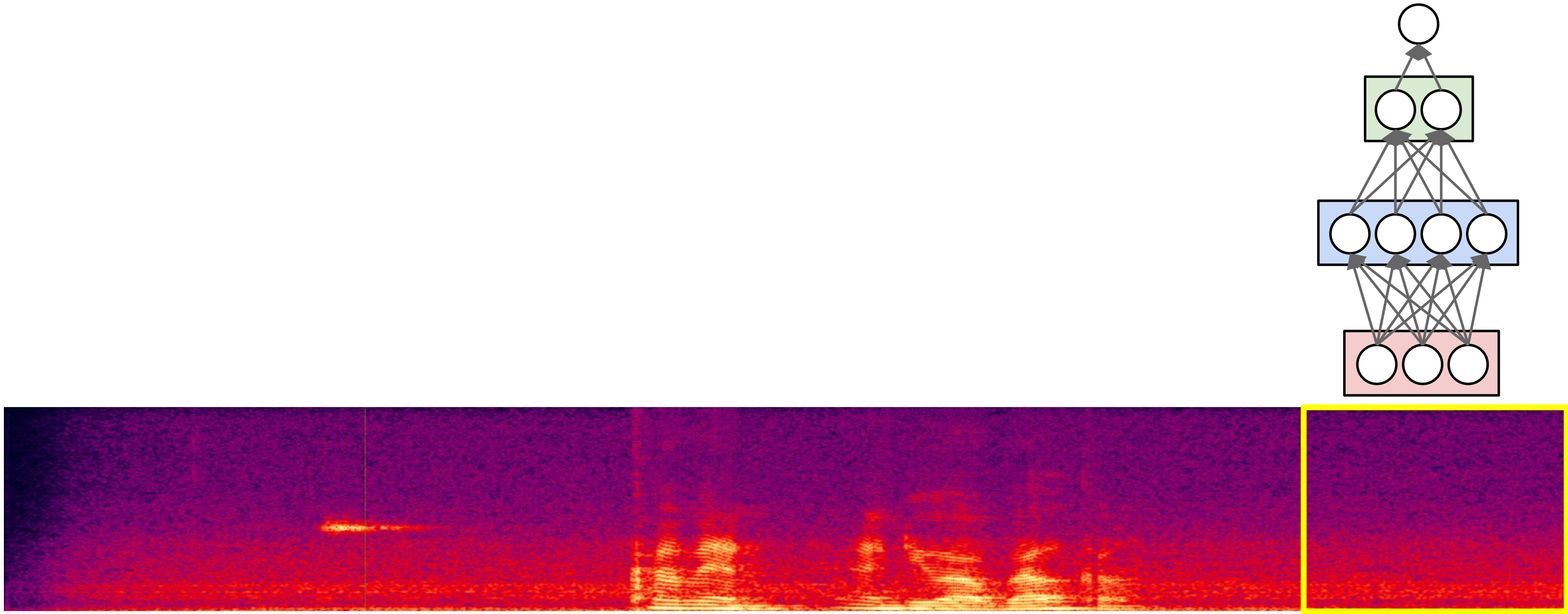
- Scan for the target word
  - The spectral time-frequency components in a “window” are input to a “welcome-detector” MLP

Solution: Scan



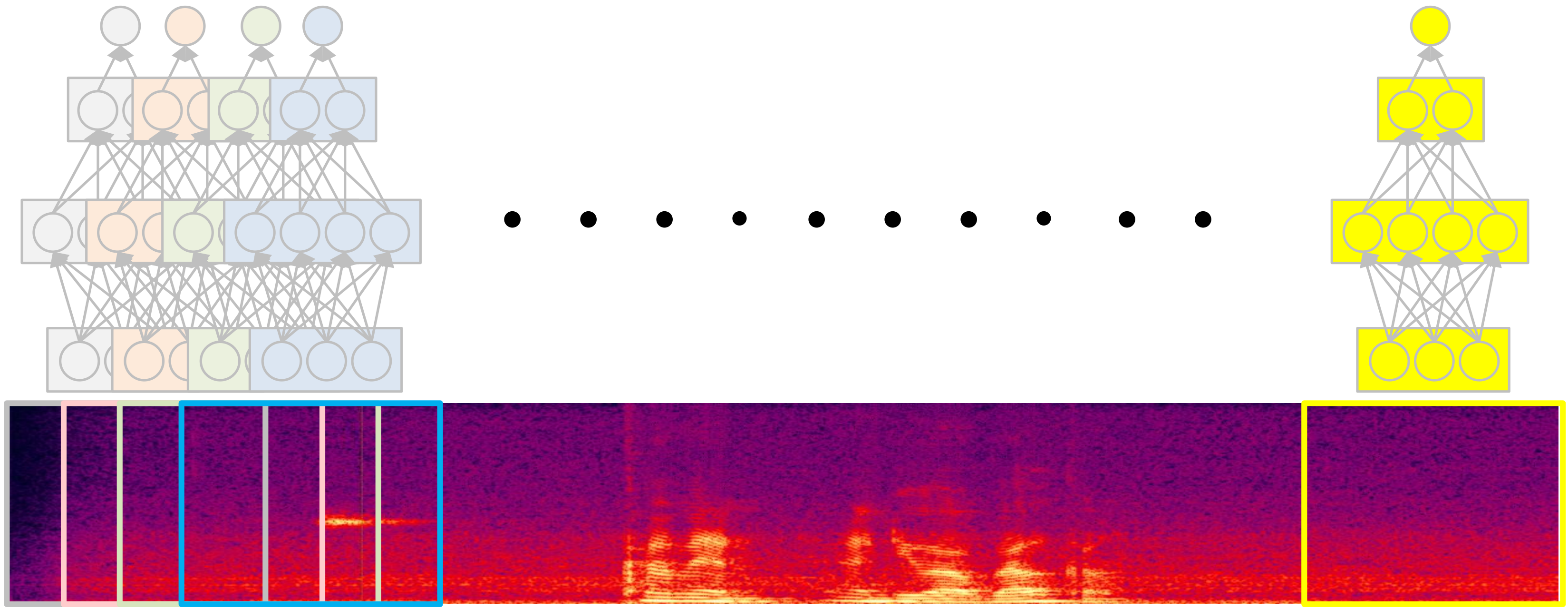
- Scan for the target word
  - The spectral time-frequency components in a “window” are input to a “welcome-detector” MLP

Solution: Scan



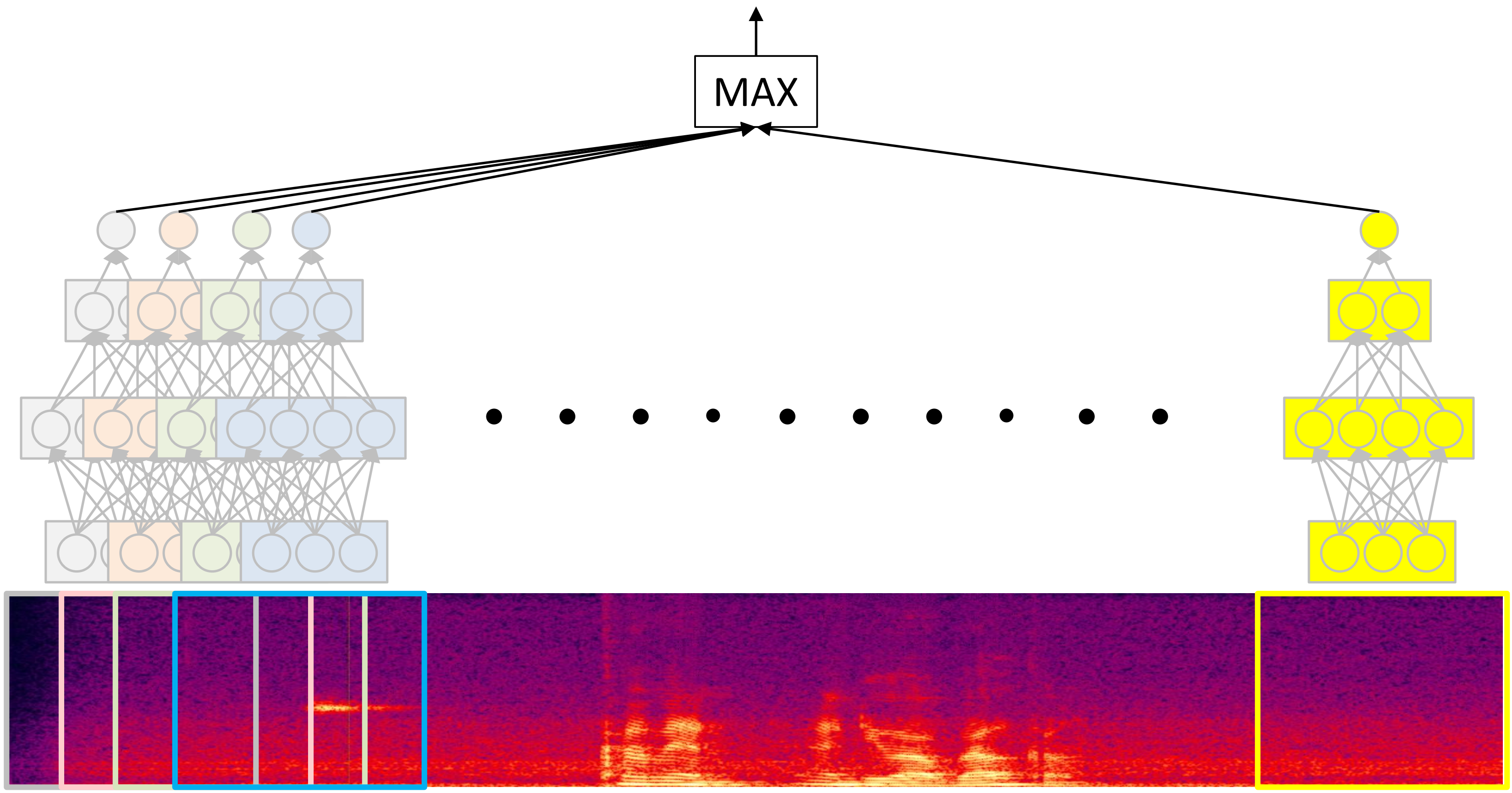
- Scan for the target word
  - The spectral time-frequency components in a “window” are input to a “welcome-detector” MLP

Solution: Scan



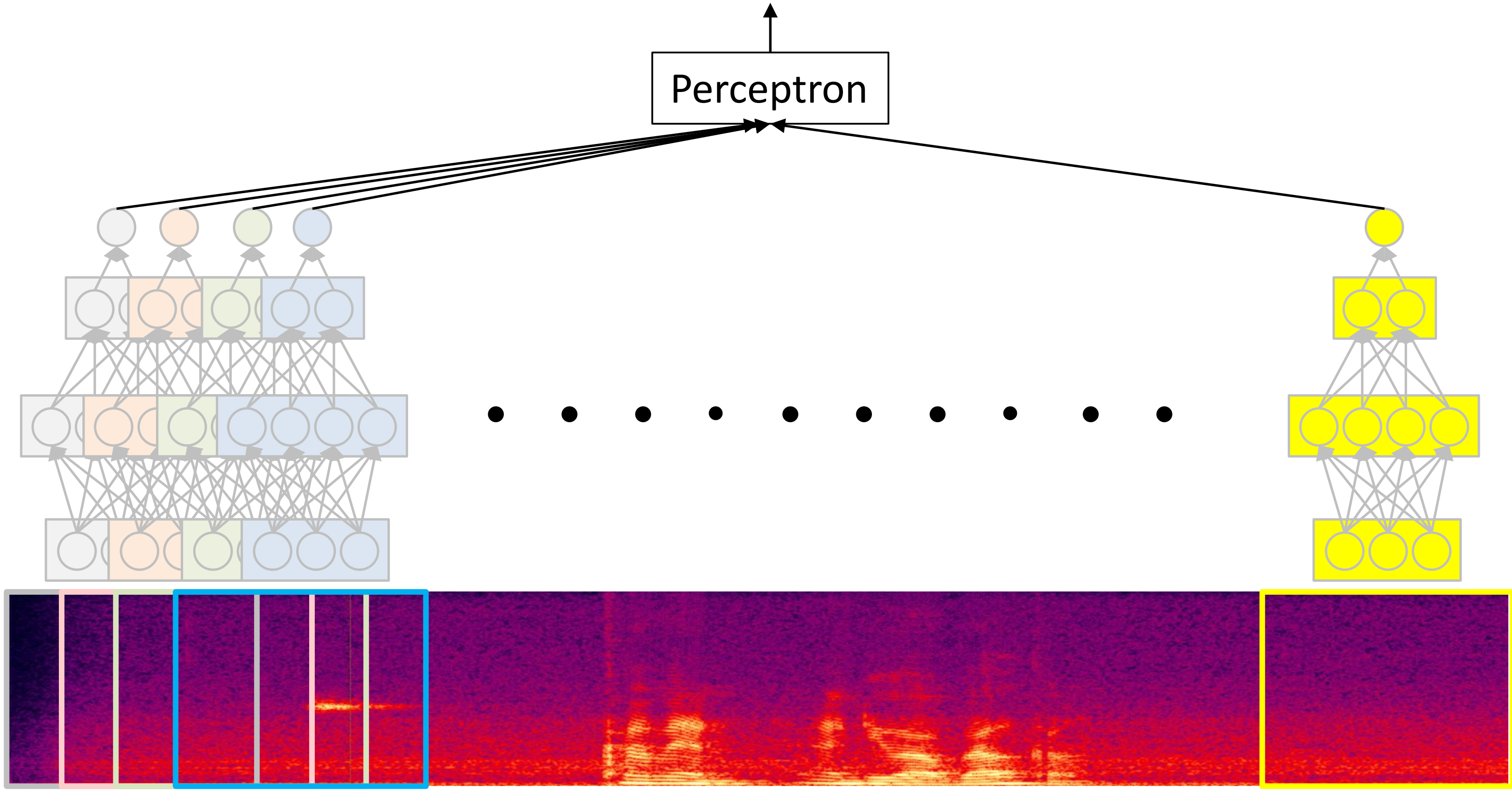
- "Does welcome occur in this recording?"
  - We have classified many "windows" individually
  - "Welcome" may have occurred in any of them

Solution: Scan



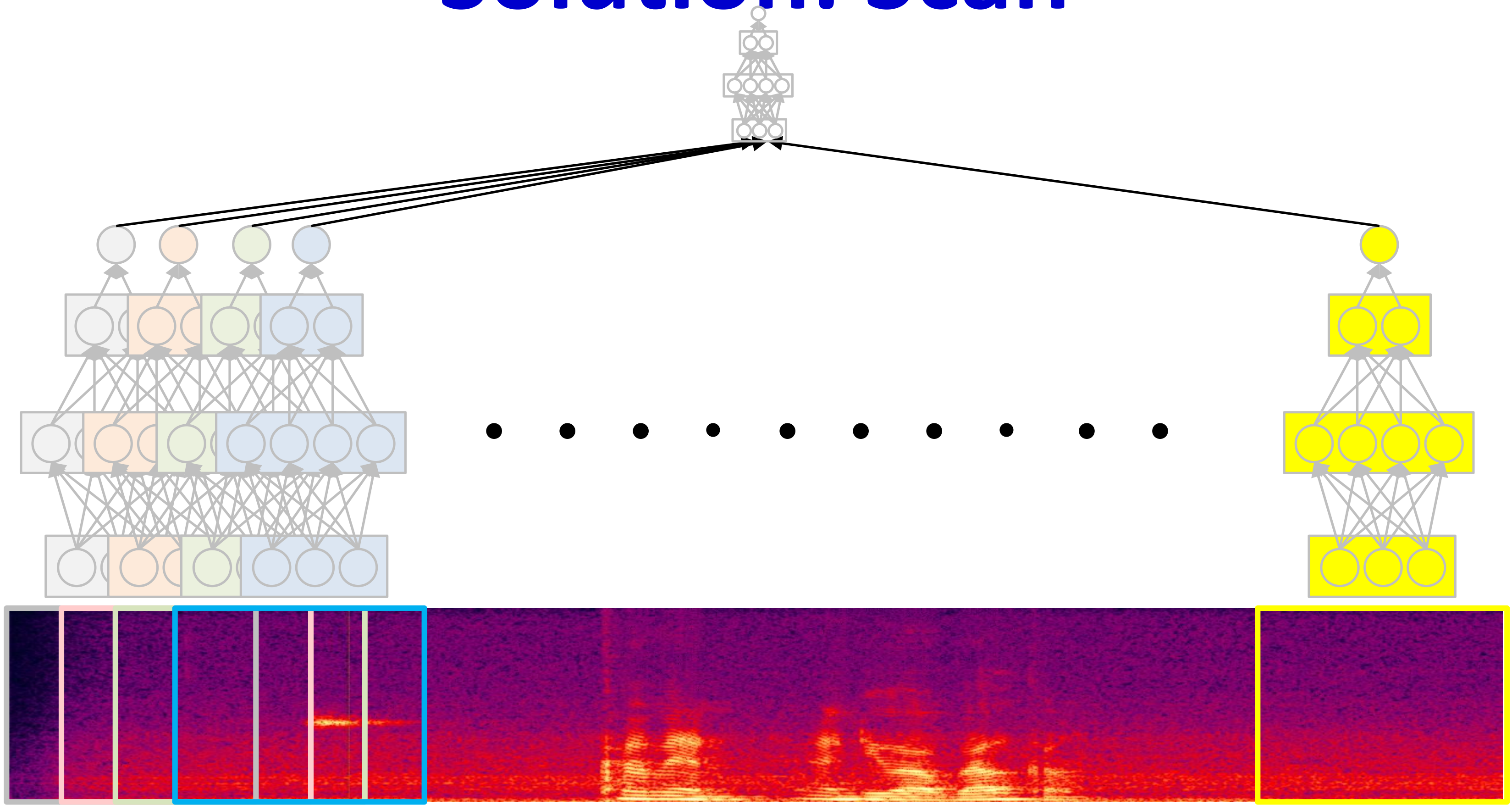
- "Does welcome occur in this recording?"
  - Maximum of all the outputs (Equivalent of Boolean OR)

Solution: Scan



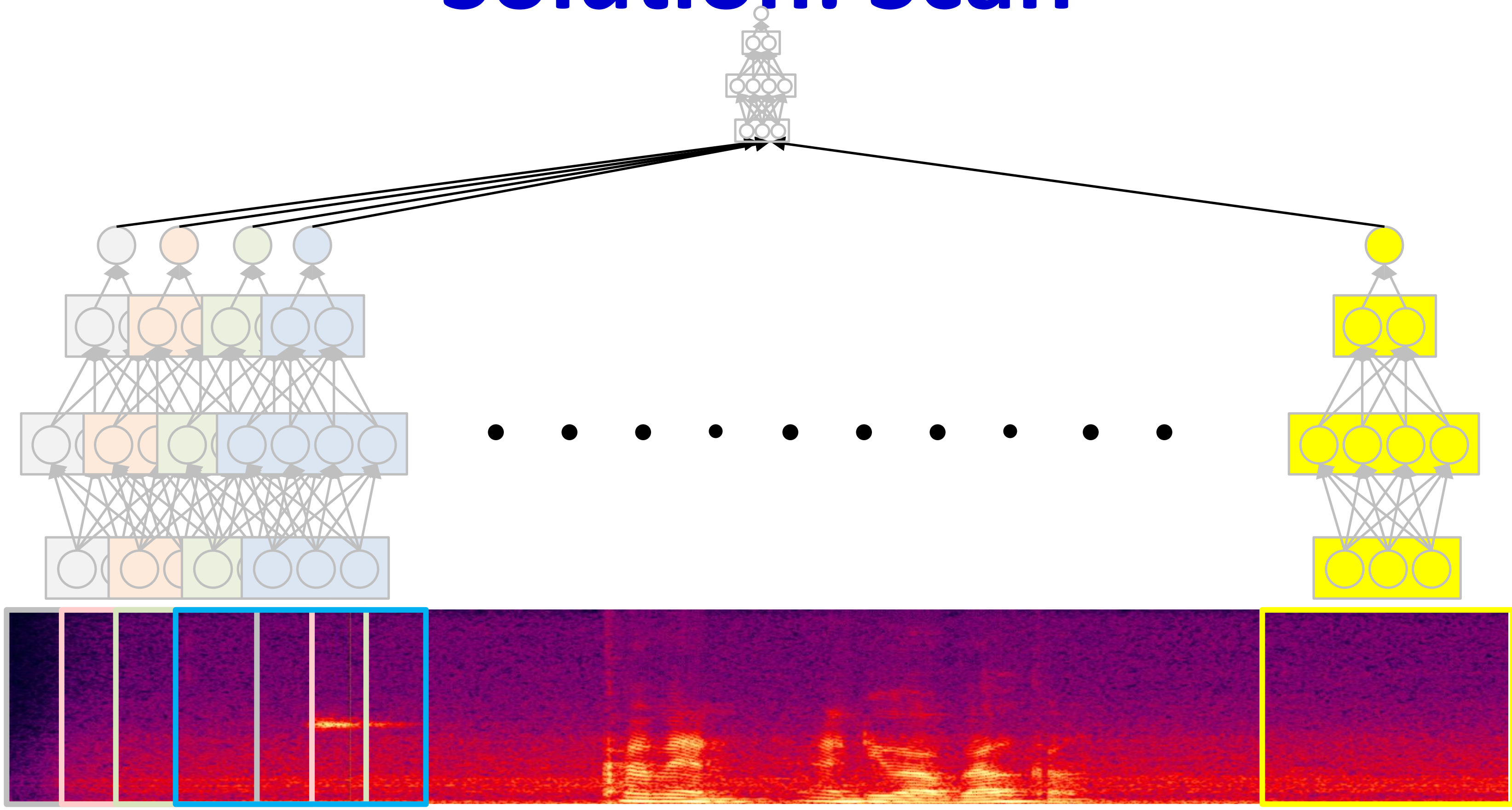
- "Does welcome occur in this recording?"
  - Maximum of all the outputs (Equivalent of Boolean OR)
  - Or a proper softmax/logistic
    - Finding a welcome in adjacent windows makes it more likely that we didn't find noise

Solution: Scan



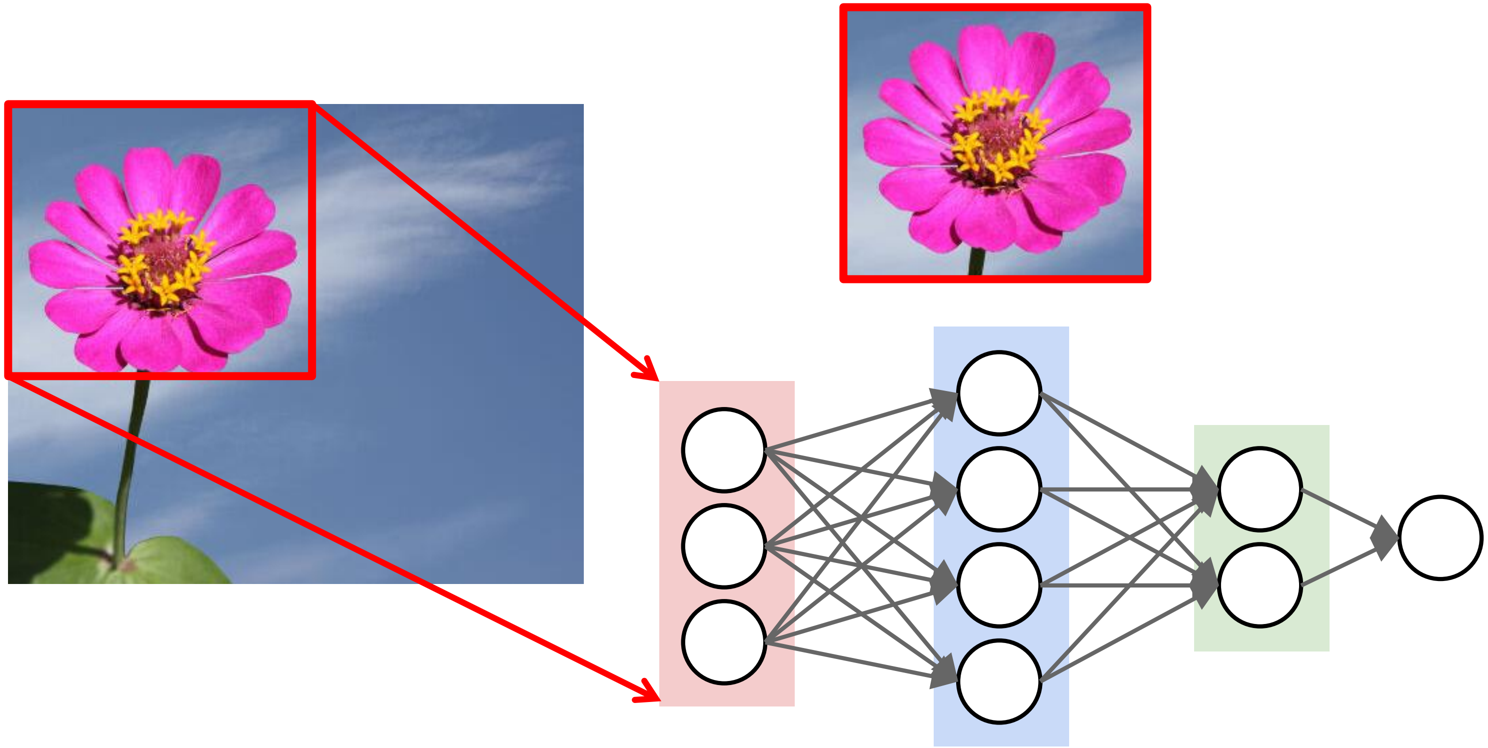
- "Does welcome occur in this recording?"
  - Maximum of all the outputs (Equivalent of Boolean OR)
  - Or a proper softmax/logistic
    - Adjacent windows can combine their evidence
  - Or even an MLP

Solution: Scan



- The entire operation can be viewed as one giant network
  - With many subnetworks, one per window
  - Restriction: All subnets are identical

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

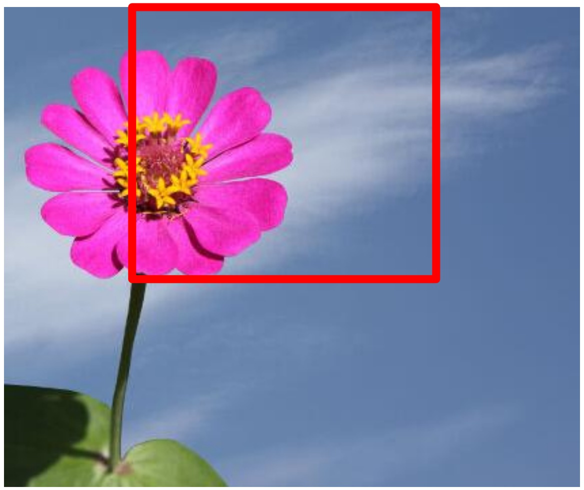
Solution: Scan

- *Scan* for the desired object

Solution: Scan

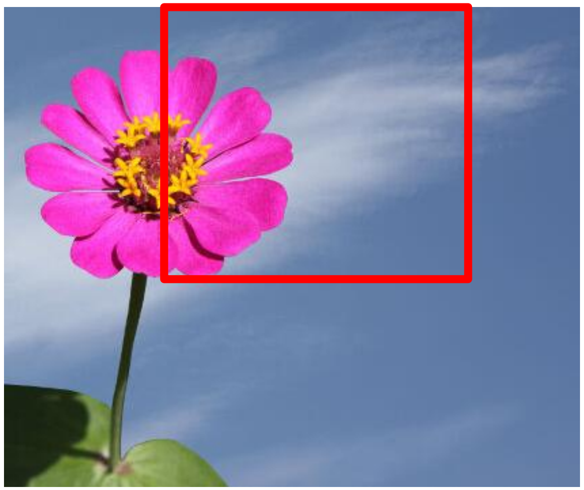
- *Scan* for the desired object

Solution: Scan



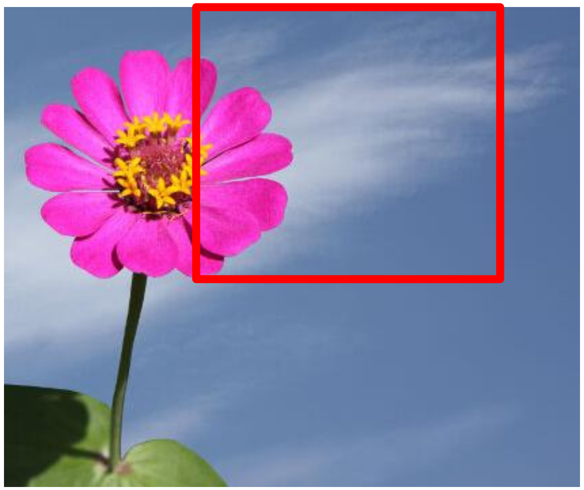
- *Scan* for the desired object

Solution: Scan



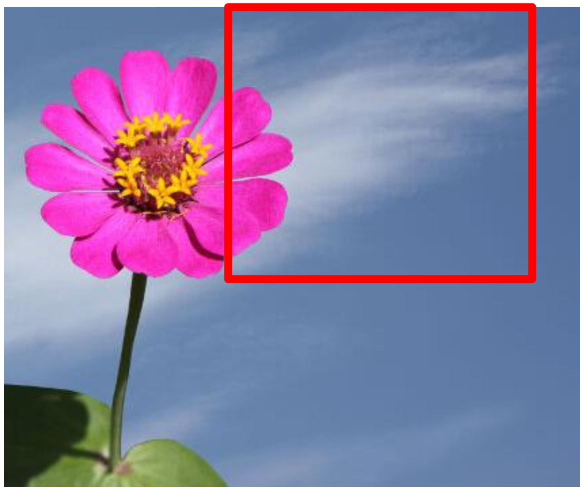
- *Scan* for the desired object

Solution: Scan



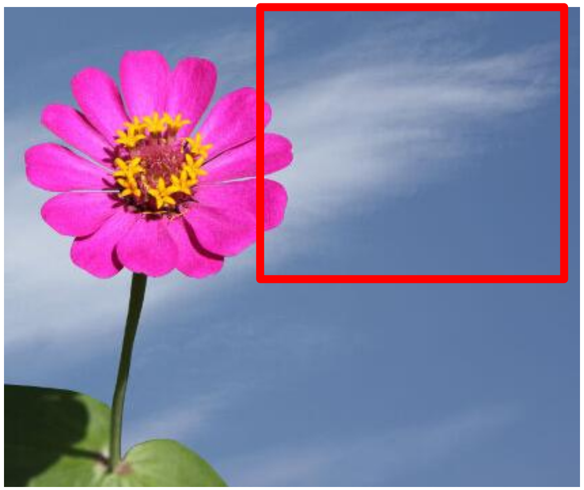
- *Scan* for the desired object

Solution: Scan



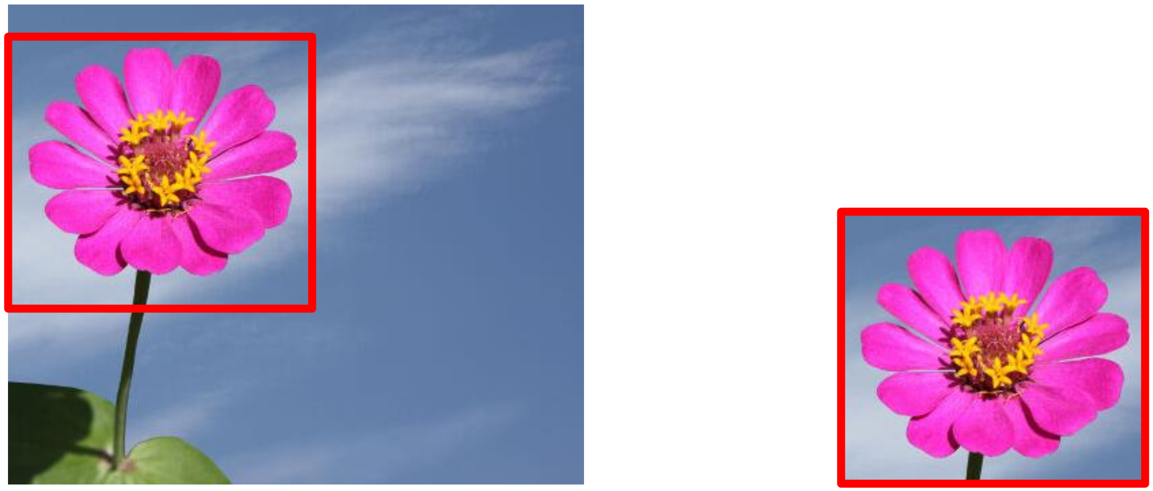
- *Scan* for the desired object

Solution: Scan



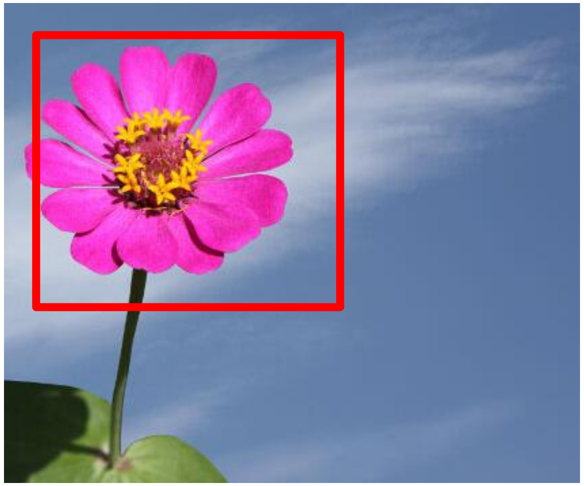
- *Scan* for the desired object

Solution: Scan



- *Scan* for the desired object

Solution: Scan

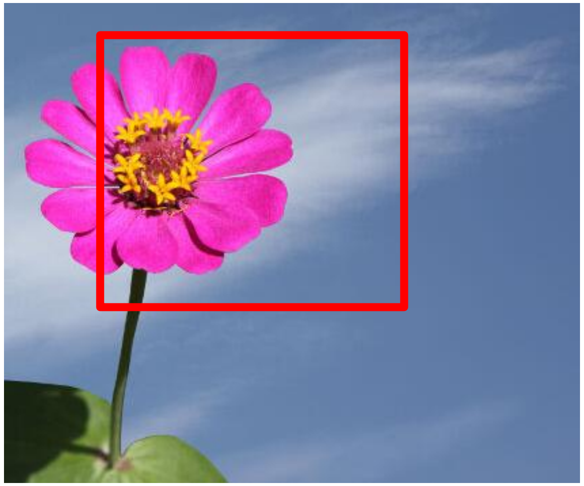


- *Scan* for the desired object

Solution: Scan

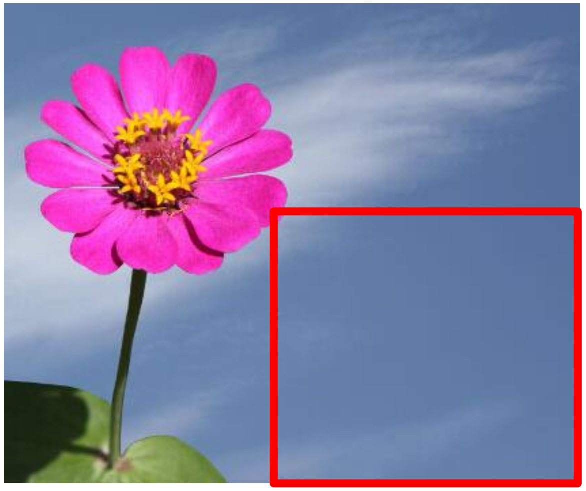
- *Scan* for the desired object

Solution: Scan



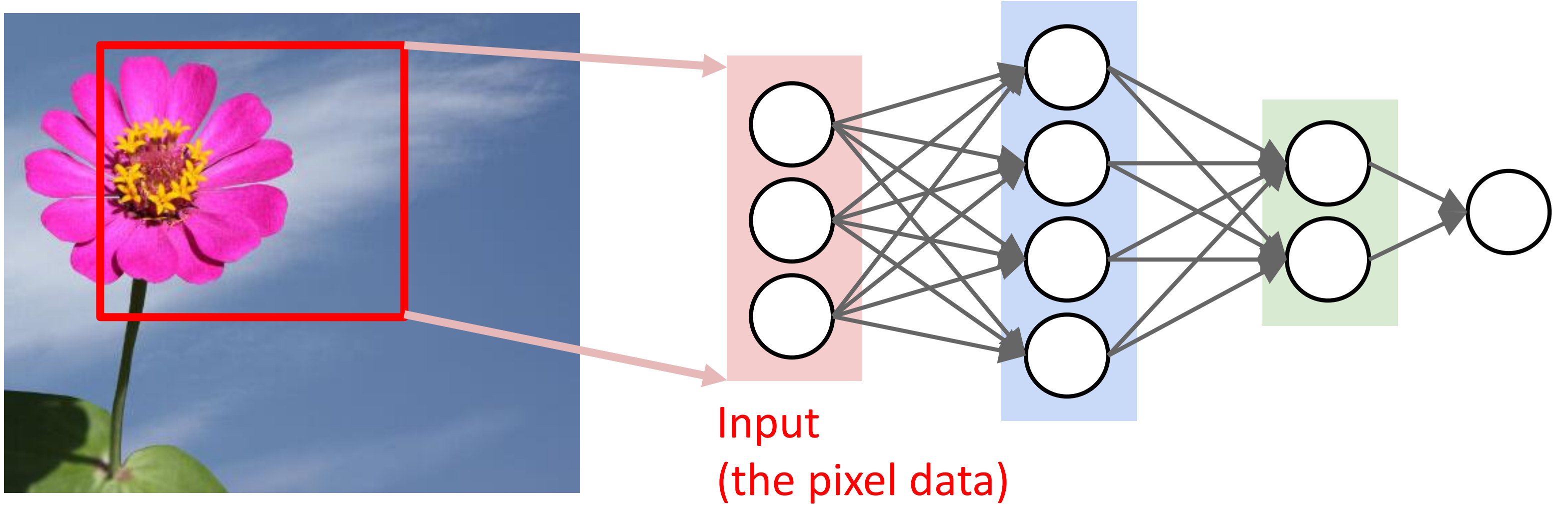
- *Scan* for the desired object

Solution: Scan



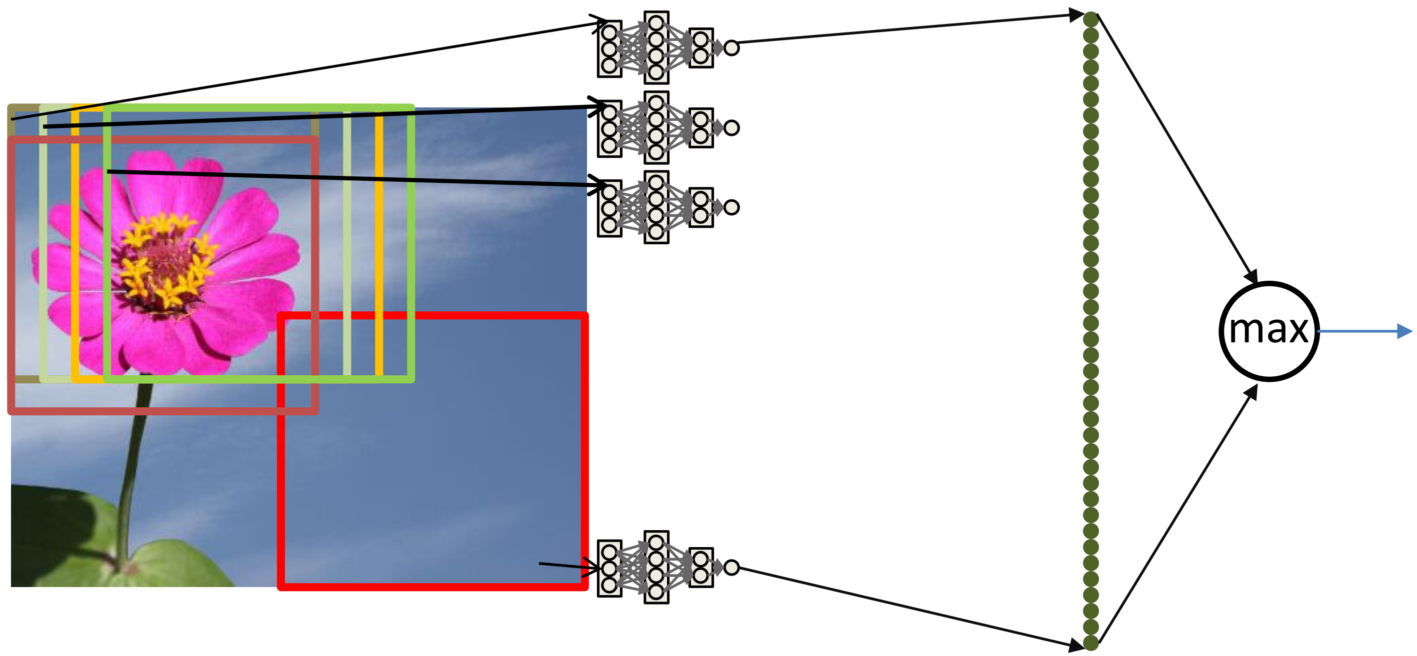
- *Scan* for the desired object

Scanning



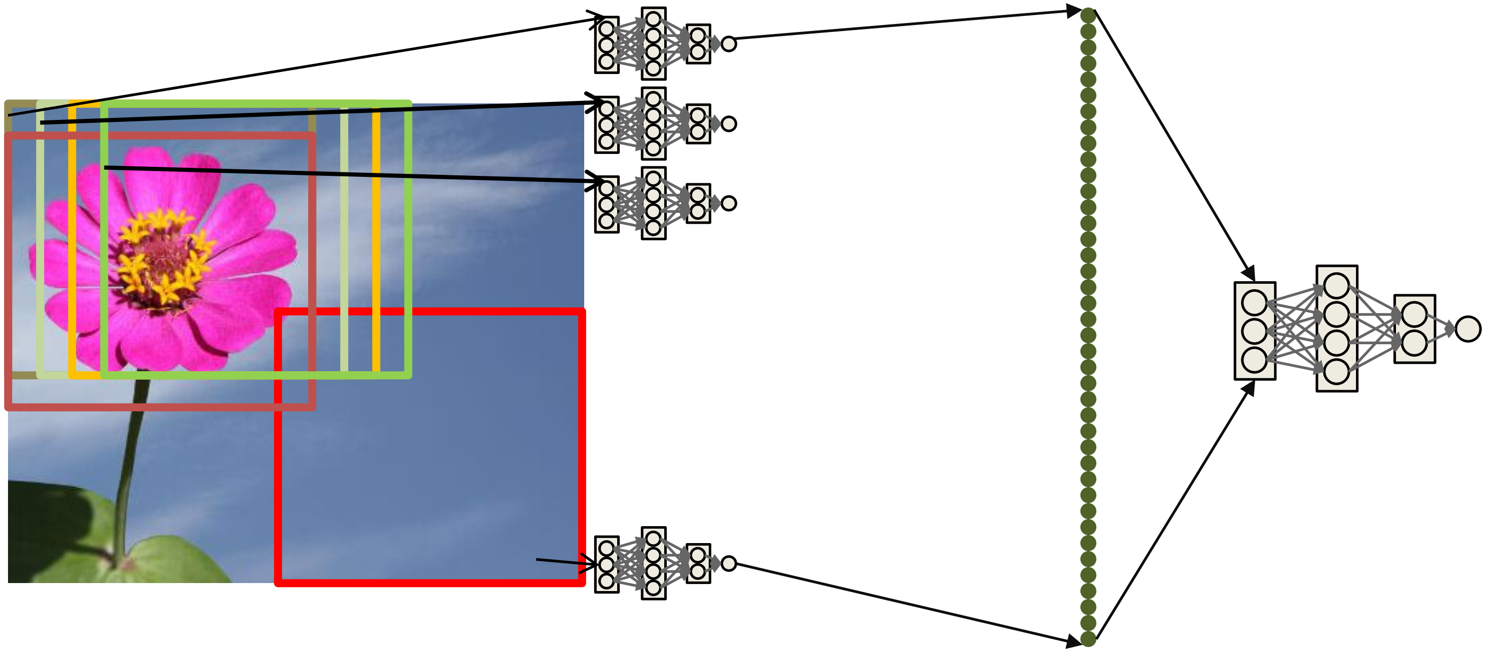
- *Scan* for the desired object
- At each location, the entire region is sent through an MLP

Scanning the picture to find a flower



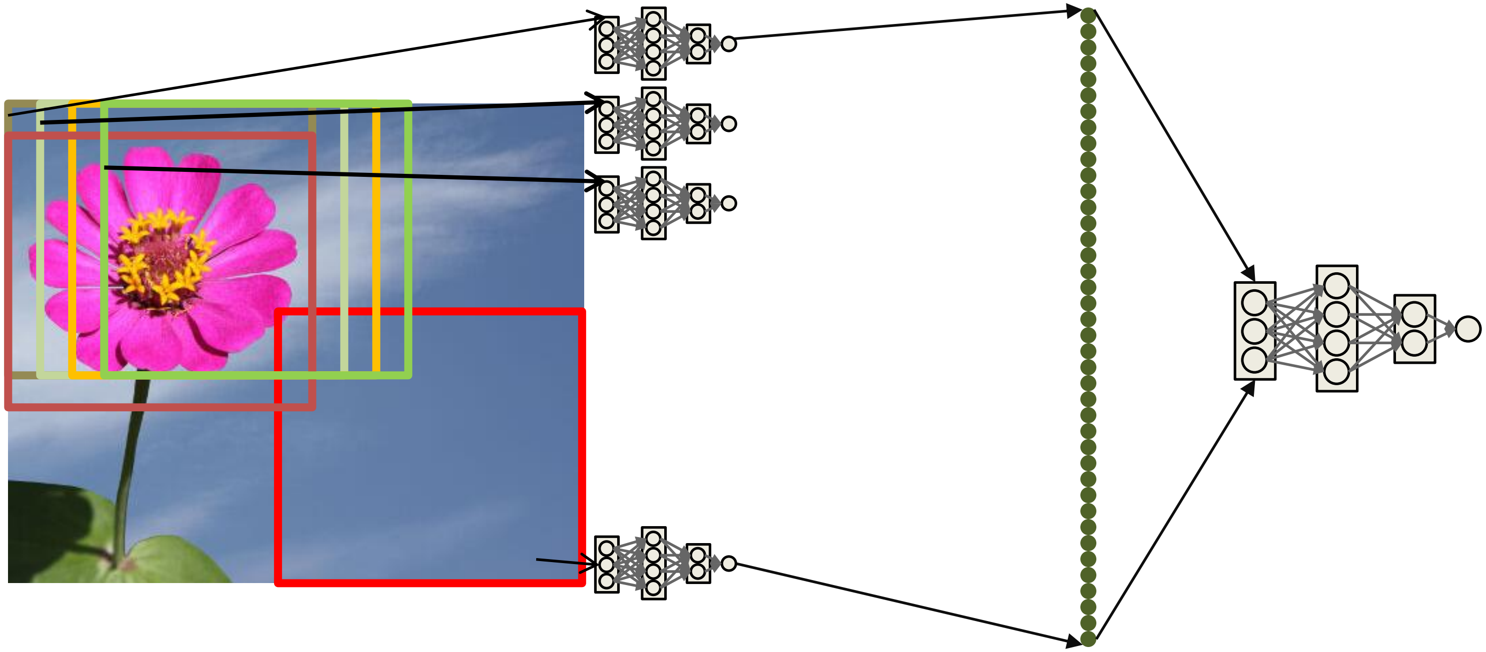
- Determine if any of the locations had a flower
  - We get one classification output per scanned location
    - The score output by the MLP
  - Look at the maximum value

Its just a giant network with common subnets



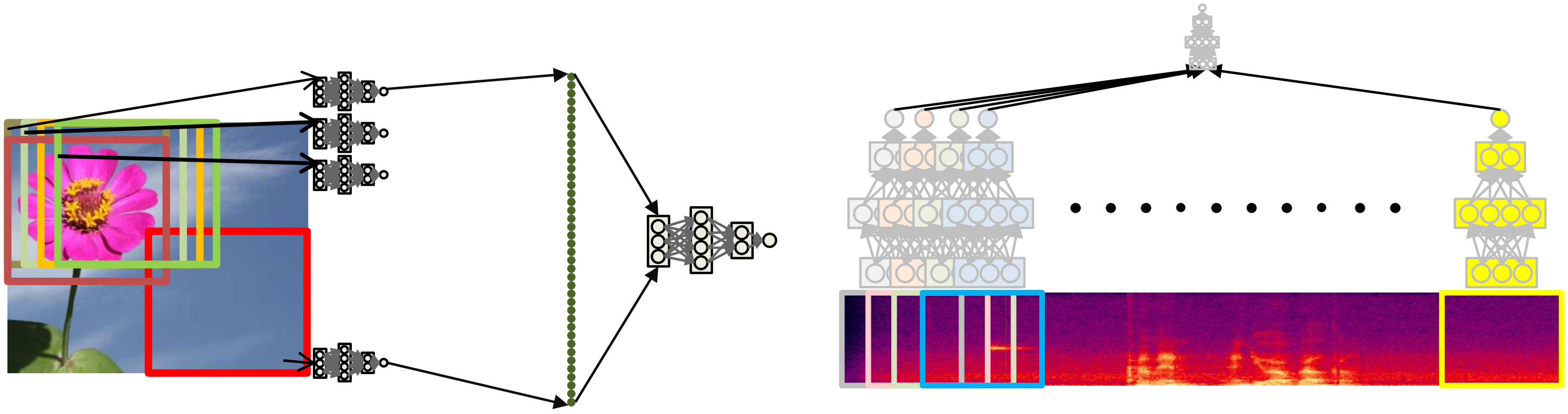
- Determine if any of the locations had a flower
  - We get one classification output per scanned location
    - The score output by the MLP
  - Look at the maximum value
  - Or pass it through an MLP

Its just a giant network with common subnets



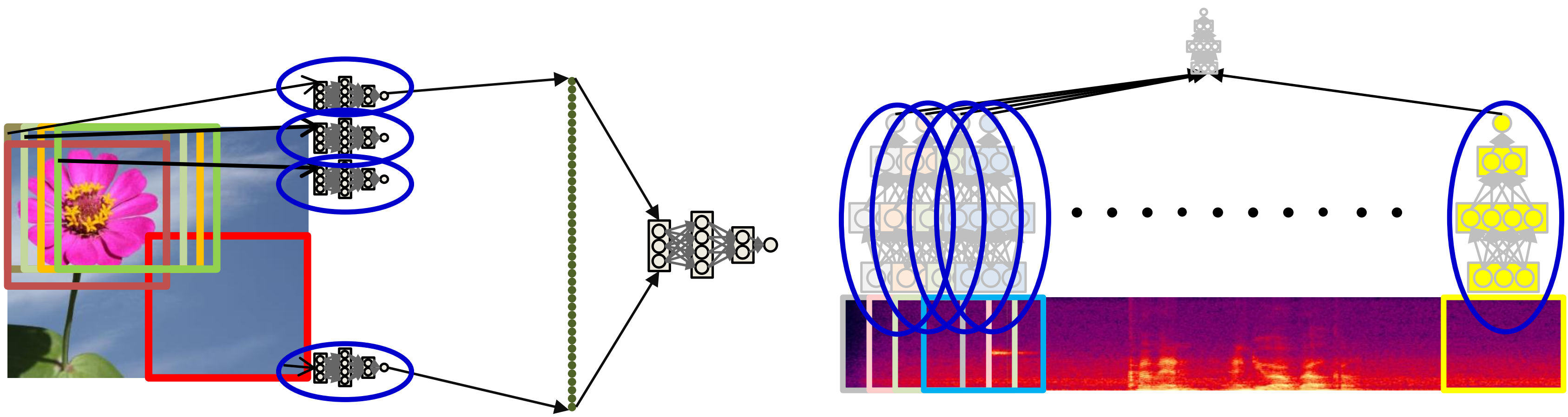
- The entire operation can be viewed as a single giant network
  – Composed of many “subnets” (one per window)
  – With one key feature: all subnets are identical

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

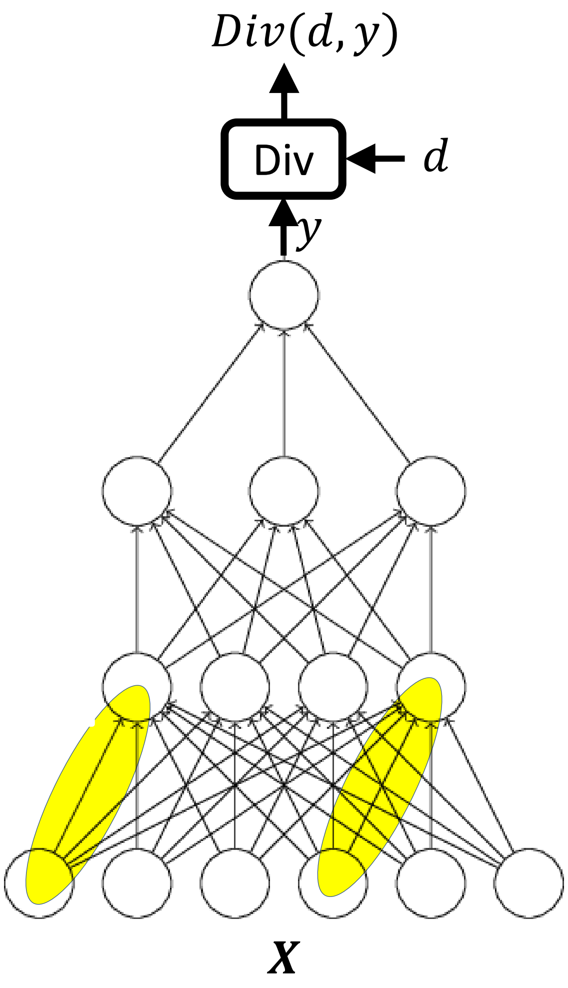
Learning in shared parameter networks

- Consider a simple network with shared weights

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

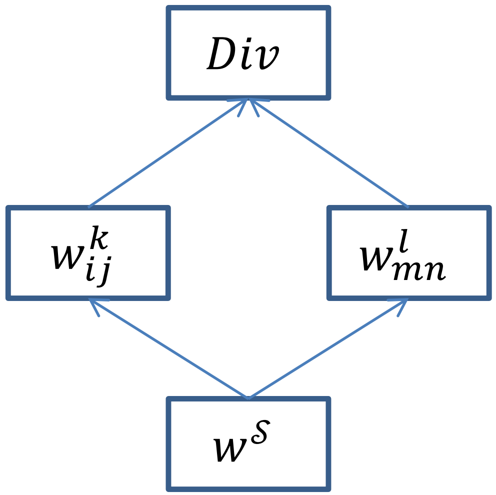
  - A weight w_{ij}^{k} is required to be identical to the weight w_{mn}^{l}

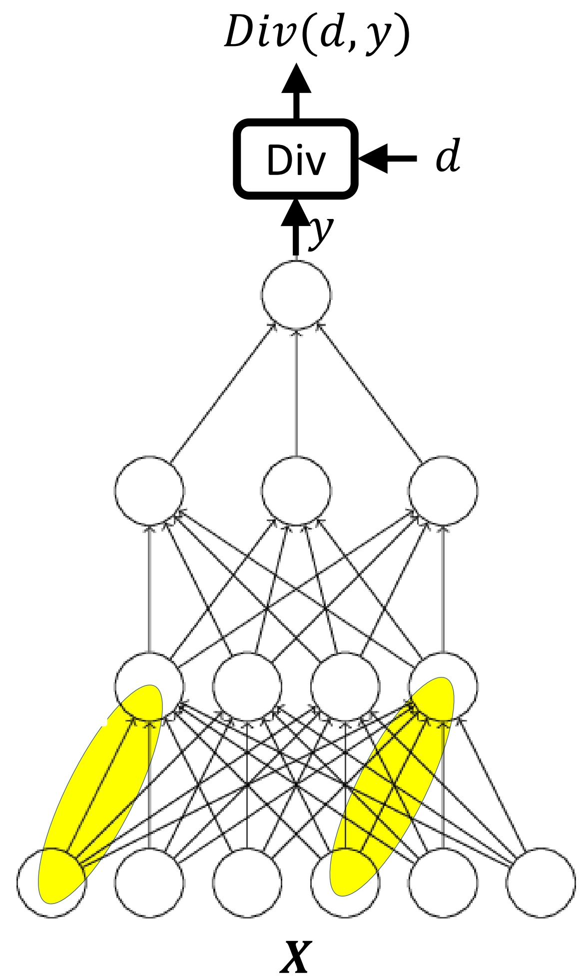
- For any training instance $\boldsymbol{X}$, a small perturbation of $w^{\mathcal{S}}$ perturbs both w_{ij}^{k} and w_{mn}^{l} identically
  - Each of these perturbations will individually influence the divergence $Div(d, y)$



Computing the divergence of shared parameters

Influence diagram



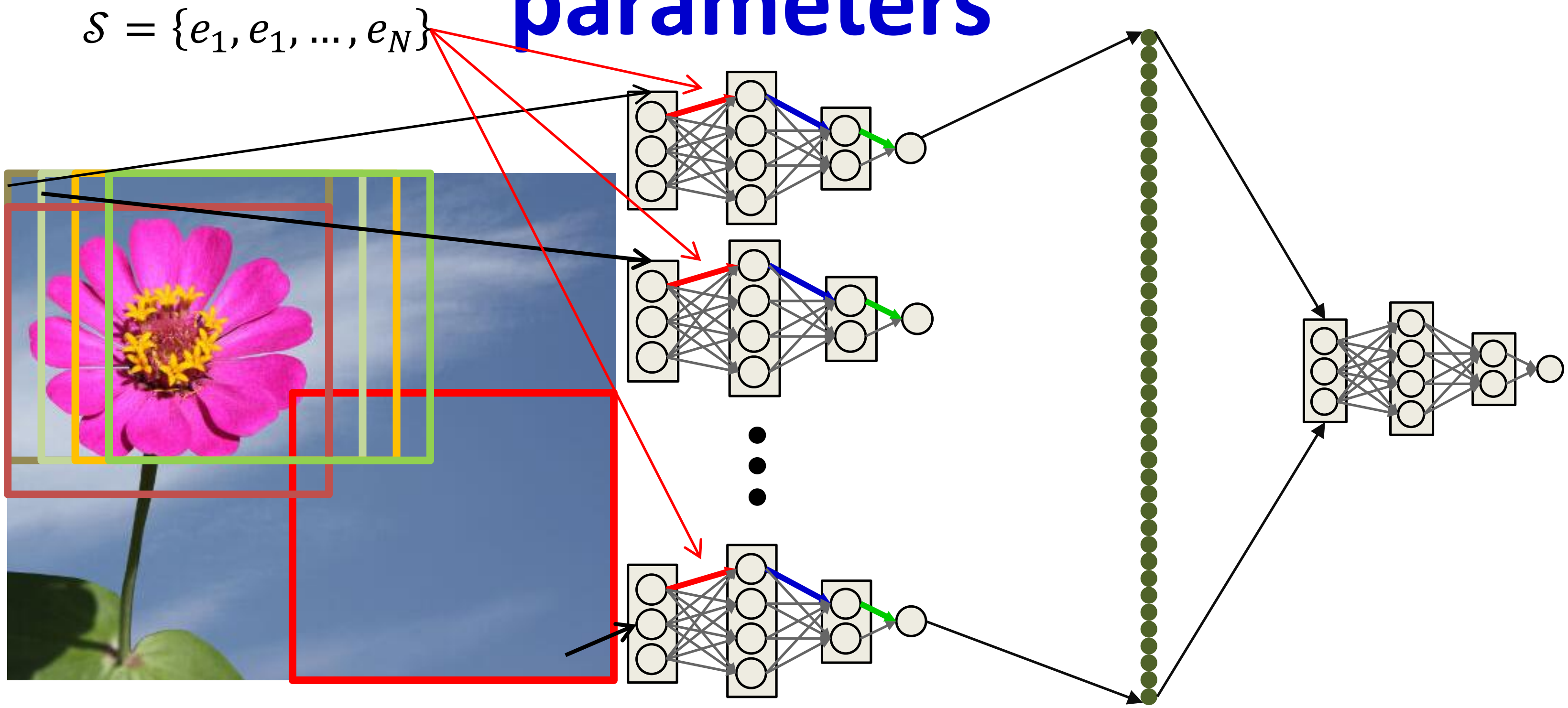


$$\frac{dDiv}{dw^{\mathcal{S}}} = \frac{dDiv}{dw_{ij}^{k}}\frac{dw_{ij}^{k}}{dw^{\mathcal{S}}} + \frac{dDiv}{dw_{mn}^{l}}\frac{dw_{mn}^{l}}{dw^{\mathcal{S}}}$$

$$= \frac{dDiv}{dw_{ij}^{k}} + \frac{dDiv}{dw_{mn}^{l}}$$

- 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

$$\boxed{\frac{dDiv}{dw^{\mathcal{S}}} = \sum_{e \in \mathcal{S}} \frac{dDiv}{dw^{e}}}$$

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

Standard gradient descent training of networks

Total training error:

$$Err = \sum_{\boldsymbol{t}} Div(\boldsymbol{Y_t}, \boldsymbol{d_t}; \mathbf{W}_1, \mathbf{W}_2, \dots, \mathbf{W}_K)$$

- Gradient descent algorithm:
- Initialize all weights $\mathbf{W}_1, \mathbf{W}_2, \dots, \mathbf{W}_K$
- Do:
  – For every layer k for all i, j, update:
    - $w_{i,j}^{(k)} = w_{i,j}^{(k)} - \eta \frac{dErr}{dw_{i,j}^{(k)}}$
- Until Err has converged

Training networks with shared parameters

- Gradient descent algorithm:
- Initialize all weights $\mathbf{W}_1, \mathbf{W}_2, \dots, \mathbf{W}_K$
- 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:

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

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

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

$$\nabla_{\mathcal{S}} Div += \frac{dDiv}{dw_{i,j}^{(k)}}$$

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

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

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

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

$$\nabla_{\mathcal{S}} Div += \frac{dDiv}{dw_{i,j}^{(k)}}$$

Computed by Backprop

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

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

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

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

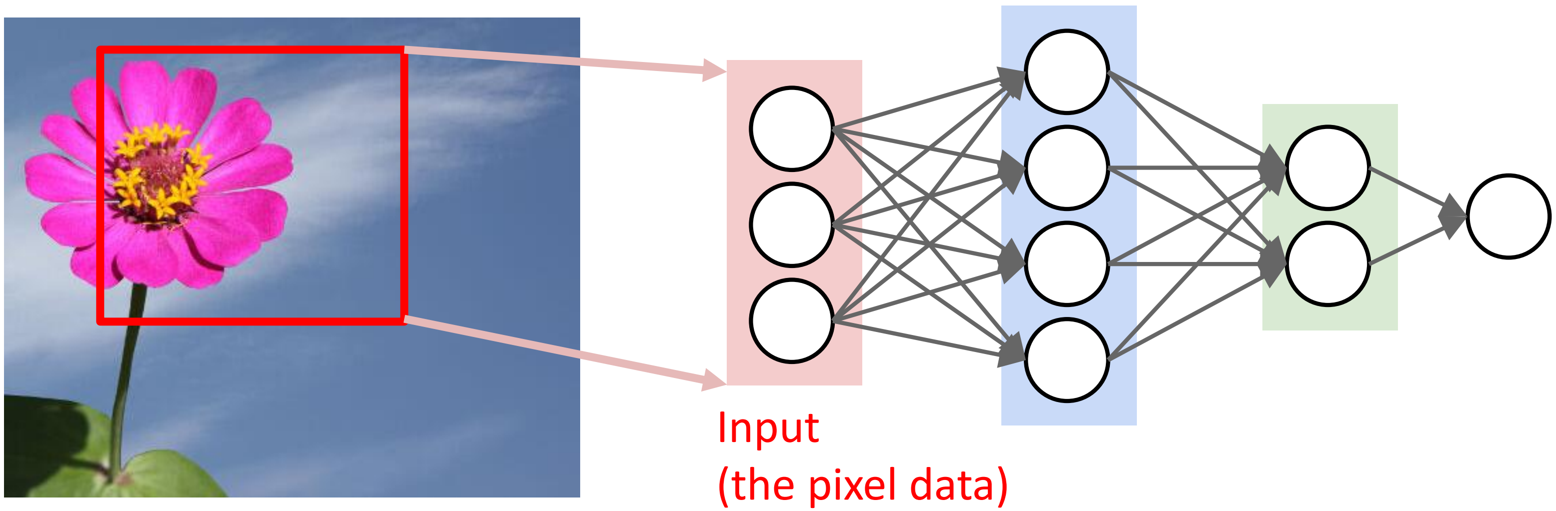
$$w_{i,j}^{(k)} = w^{\mathcal{S}}$$

- Until Err has converged

Story so far

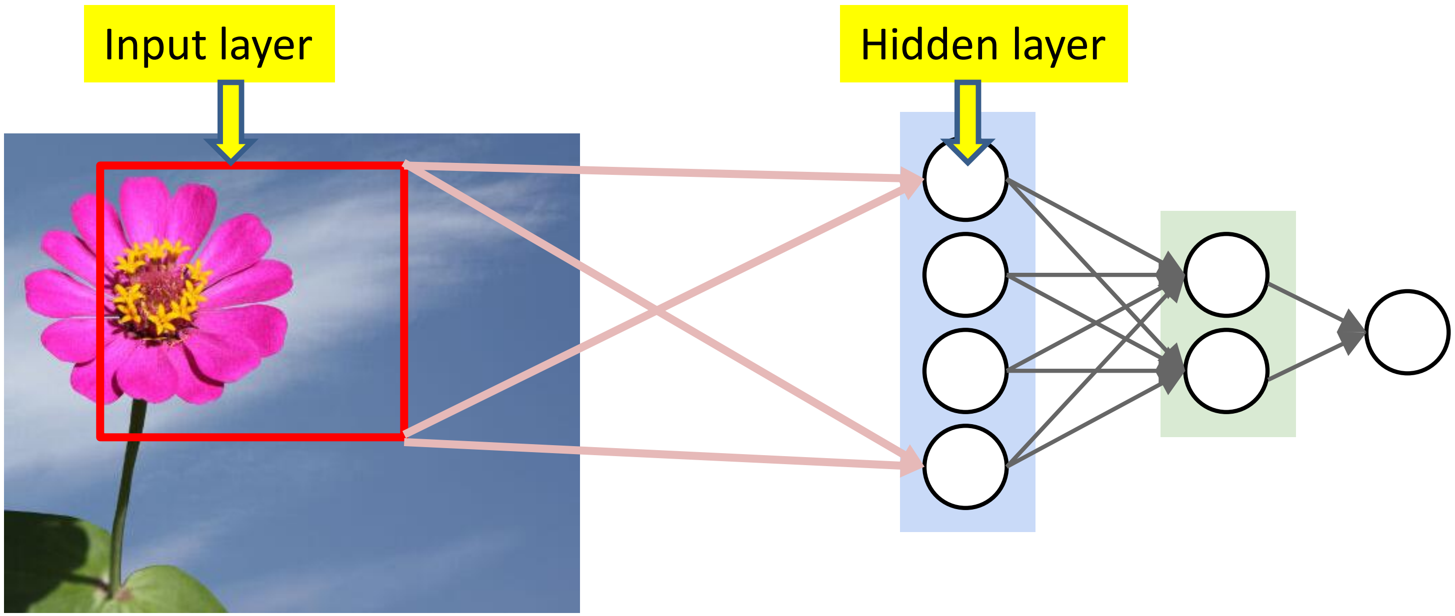
- Position-invariant pattern classification can be performed by scanning
  - 1-D scanning for sound
  - 2-D scanning for images
  - 3-D and higher-dimensional scans for higher dimensional data
- Scanning is equivalent to composing a large network with repeating subnets
  - The large network has shared 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: A closer look



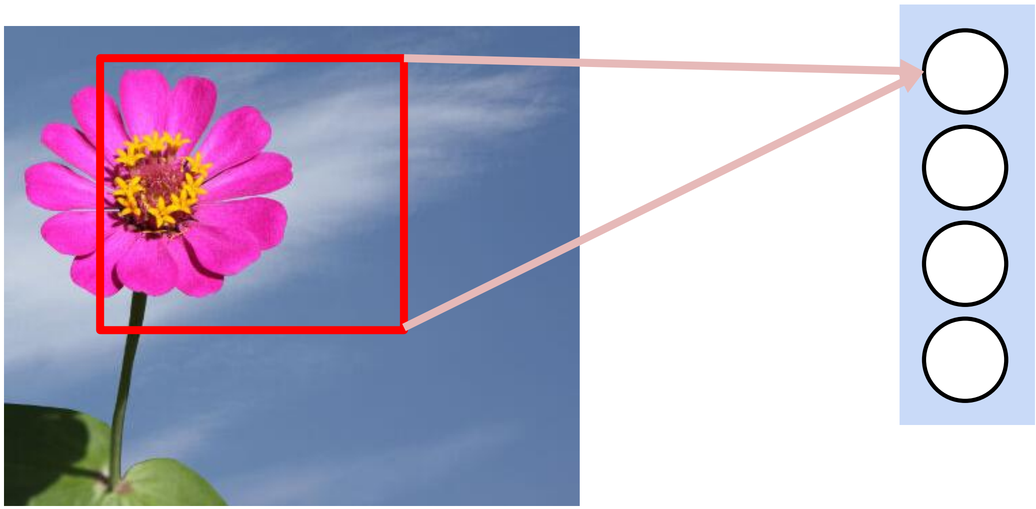
- *Scan* for the desired object
- At each location, the entire region is sent through an MLP

Scanning: A closer look



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

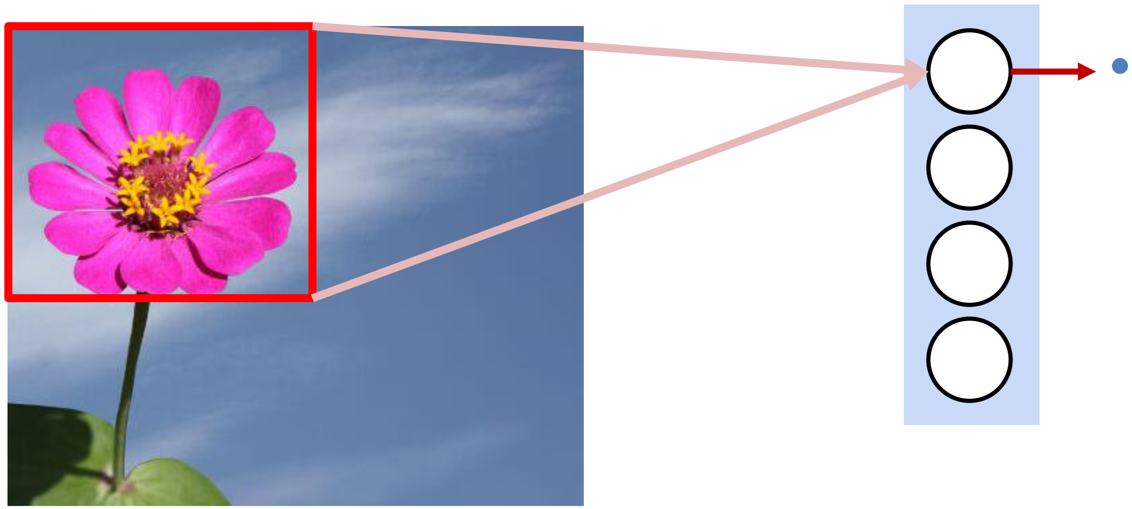
Scanning: A closer look



- Consider a single neuron

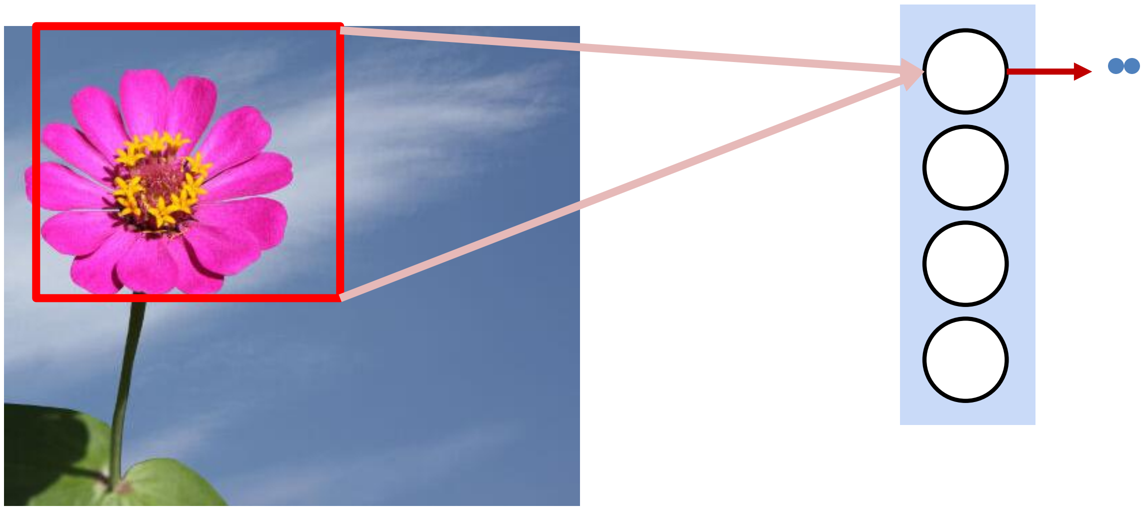
Scanning: A closer look

$$activation\left(\sum_{i,j} w_{ij}p_{ij} + b\right)$$



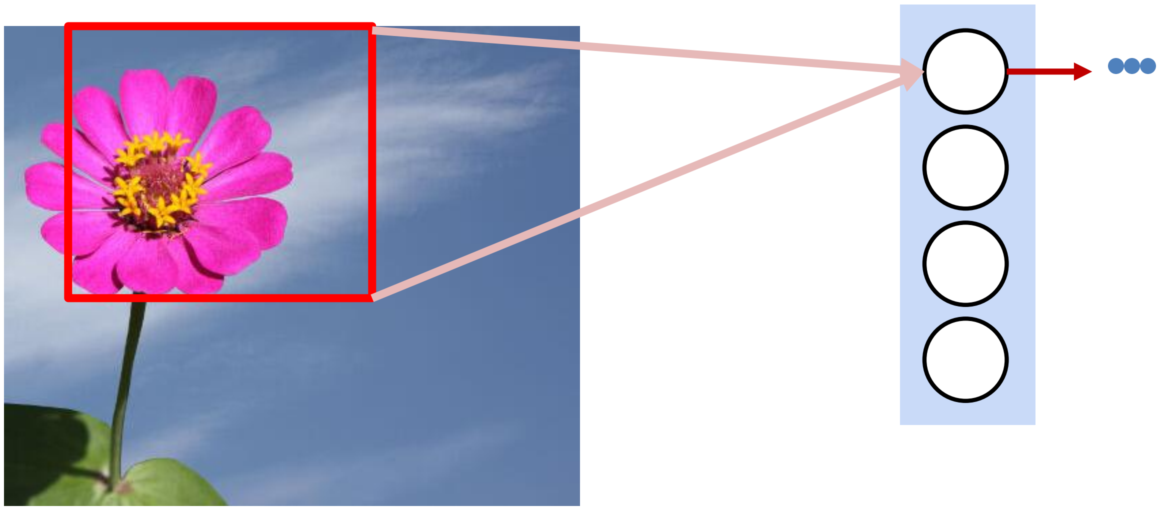
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the part of the picture in the box as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture

Scanning: A closer look



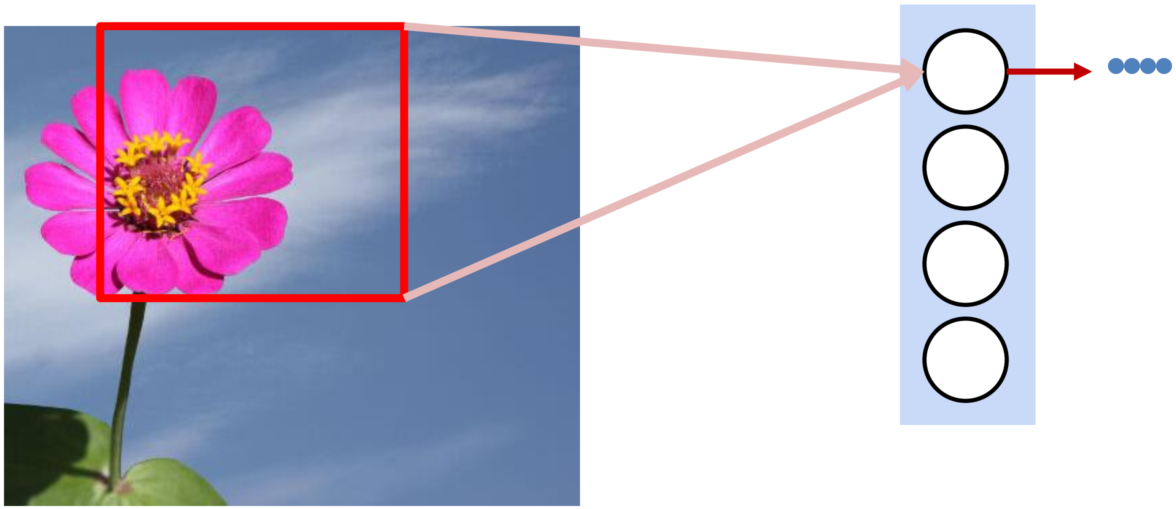
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture

Scanning: A closer look



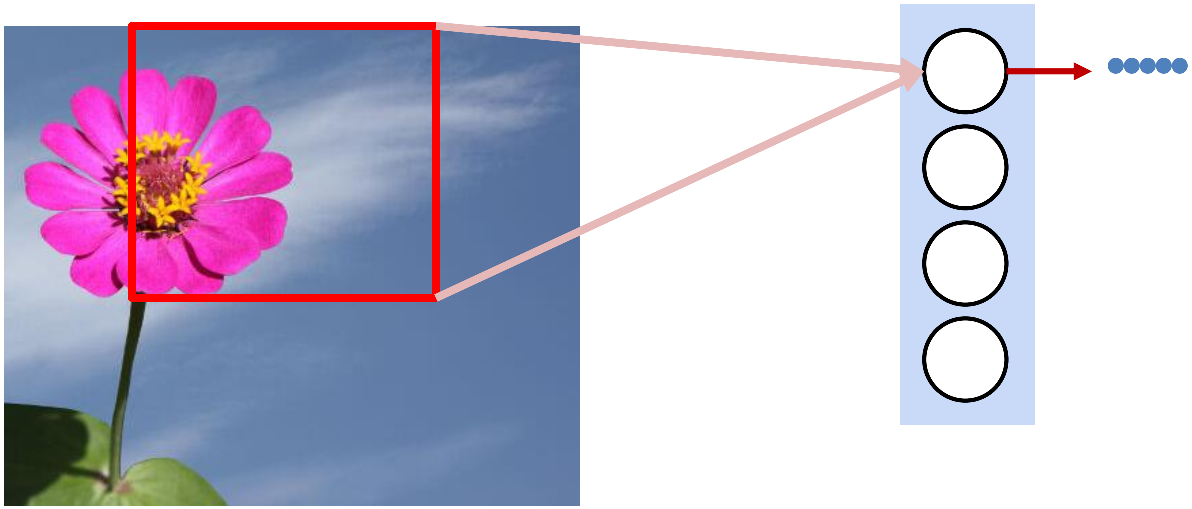
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  – We could arrange the outputs of the neurons for each position correspondingly to the original picture

Scanning: A closer look



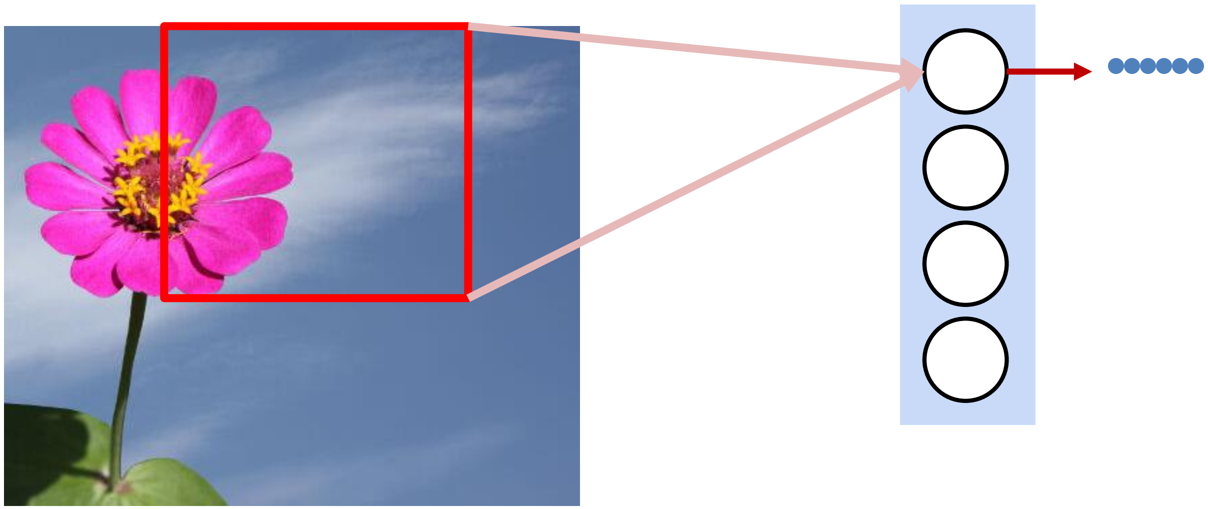
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  – We could arrange the outputs of the neurons for each position correspondingly to the original picture

Scanning: A closer look



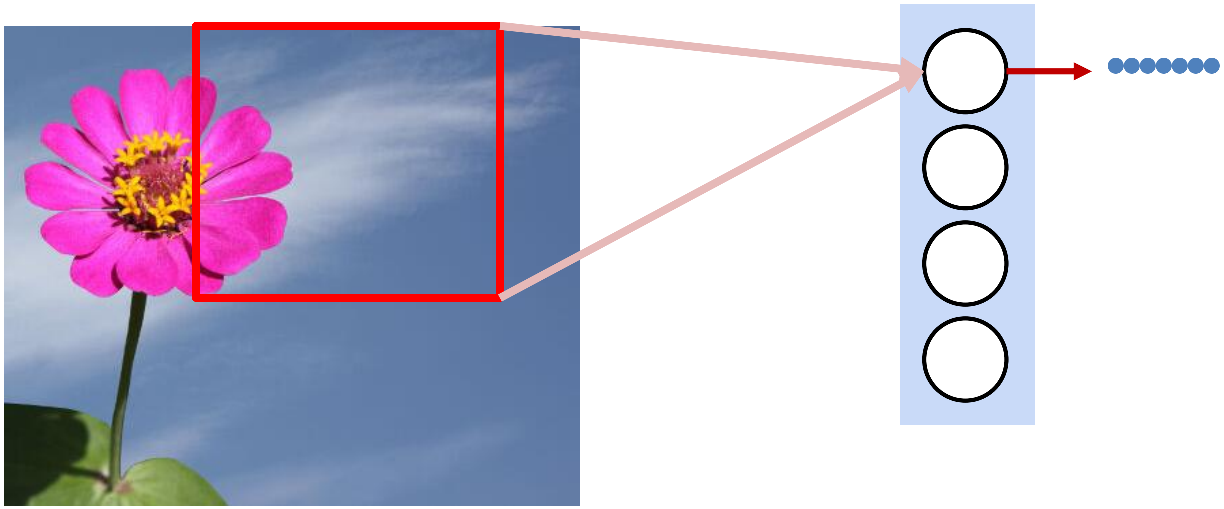
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  – We could arrange the outputs of the neurons for each position correspondingly to the original picture

Scanning: A closer look



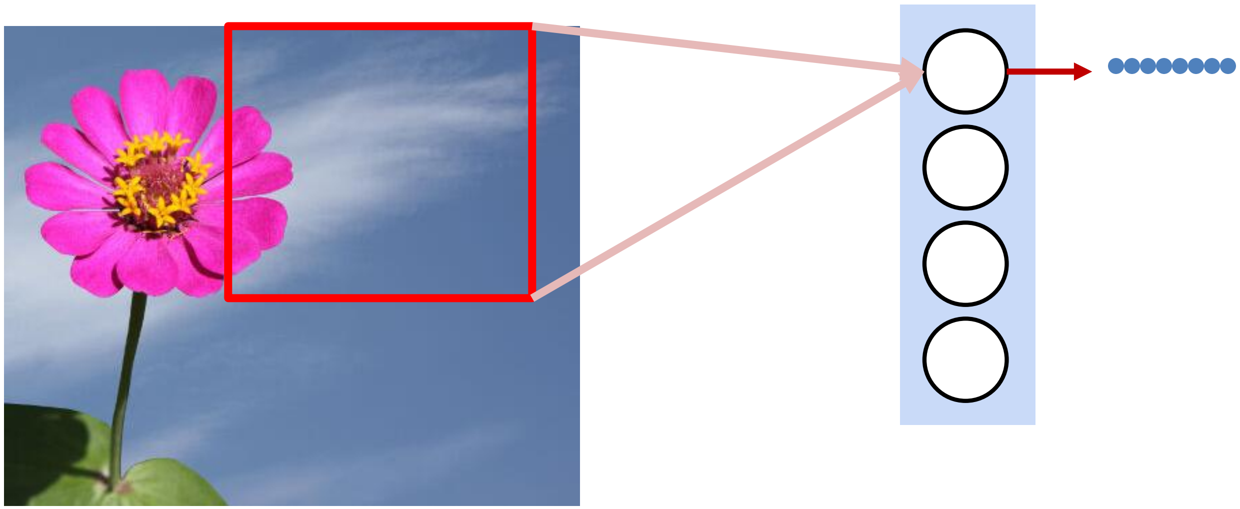
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture

Scanning: A closer look



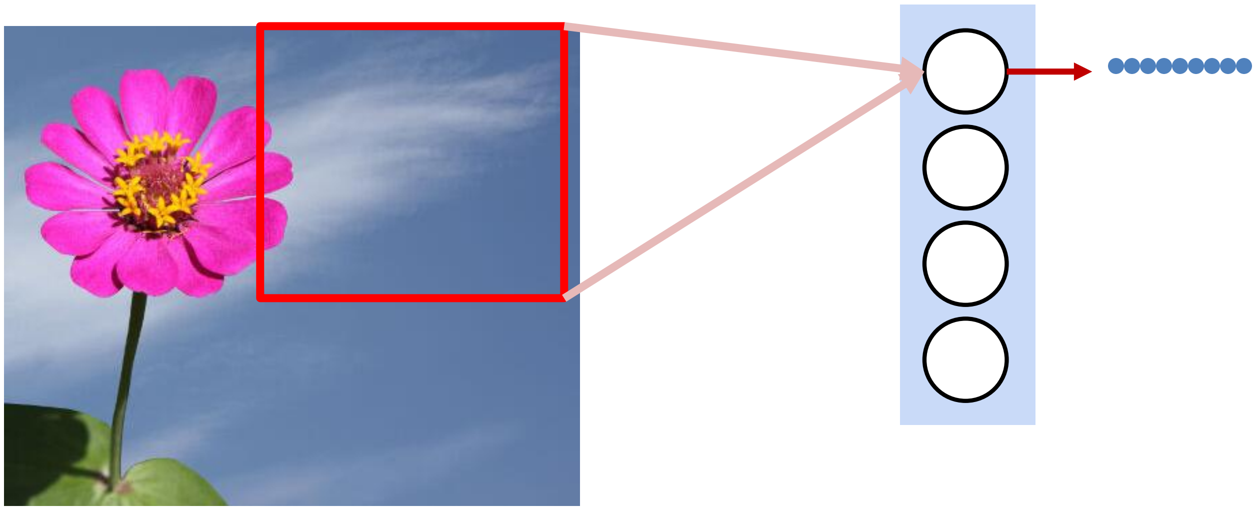
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  – We could arrange the outputs of the neurons for each position correspondingly to the original picture

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

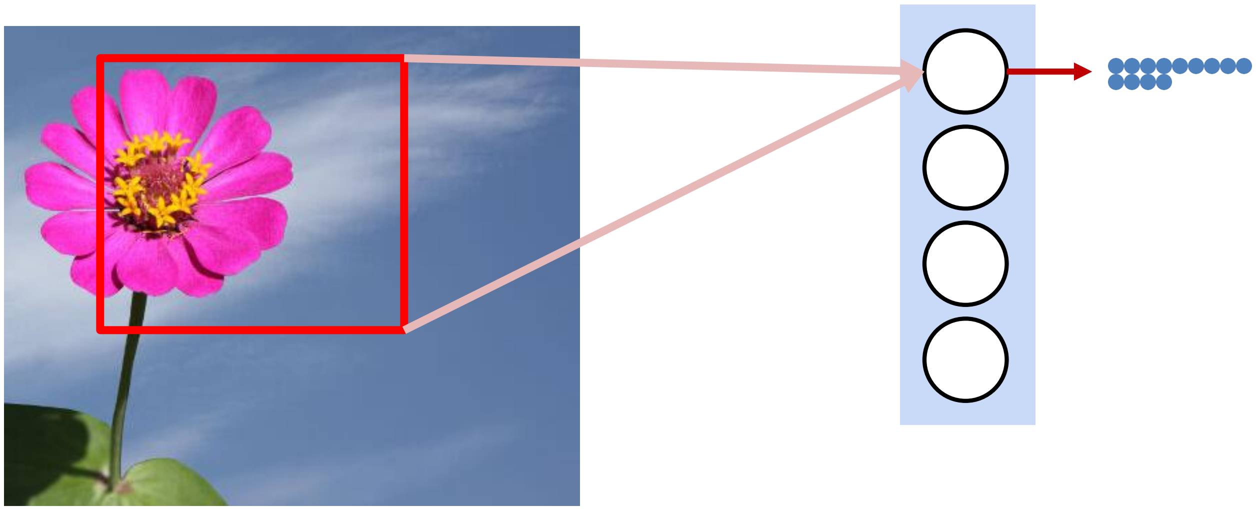
Scanning: A closer look

- 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

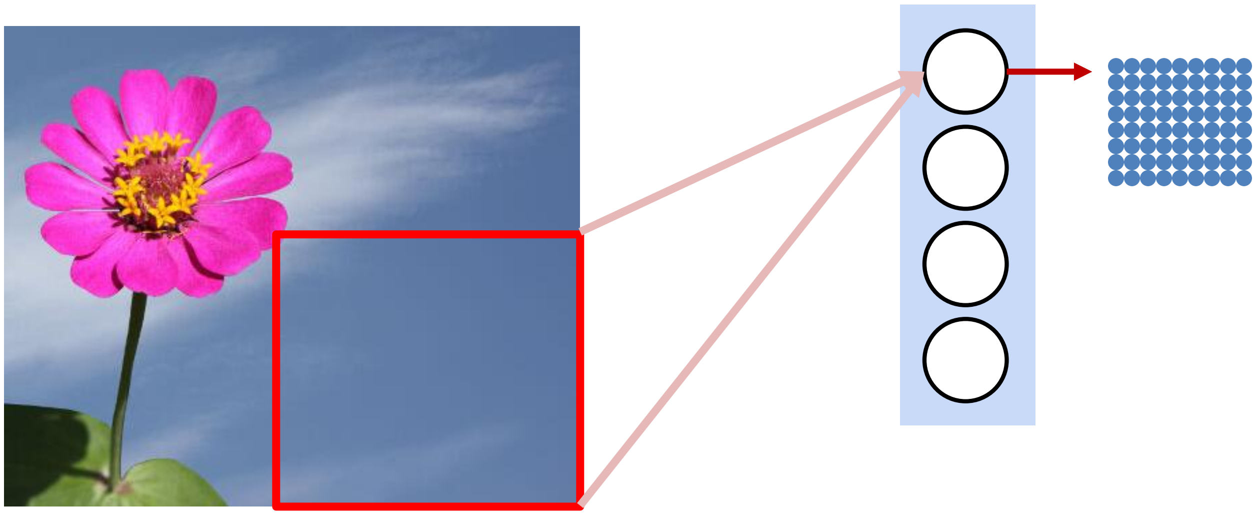
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture

Scanning: A closer look



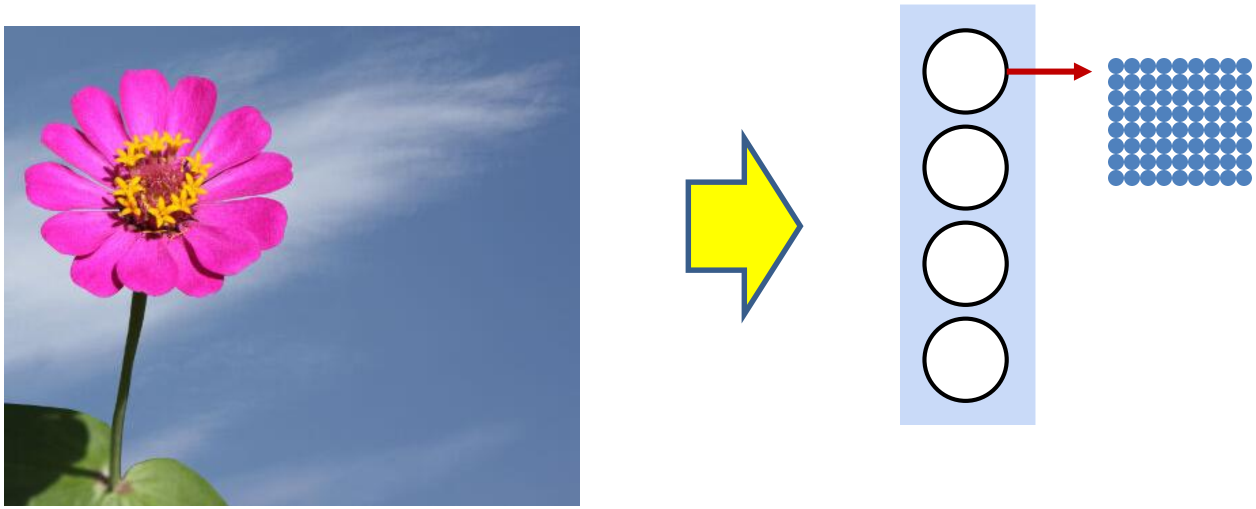
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture

Scanning: A closer look



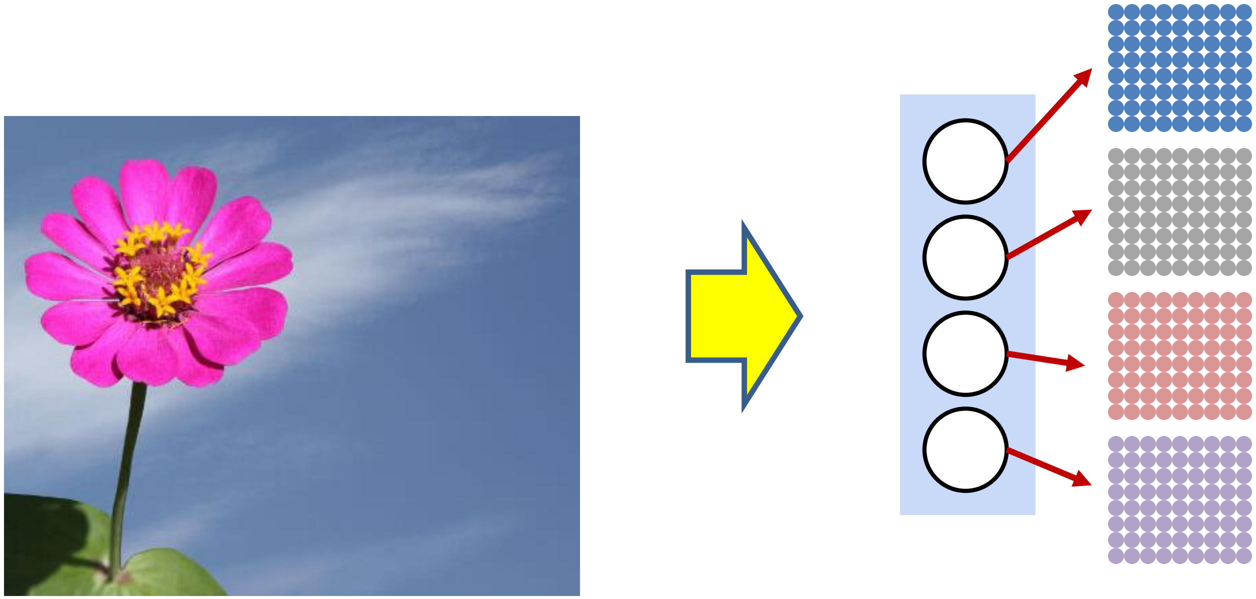
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture
- Eventually, we can arrange the outputs from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

Scanning: A closer look



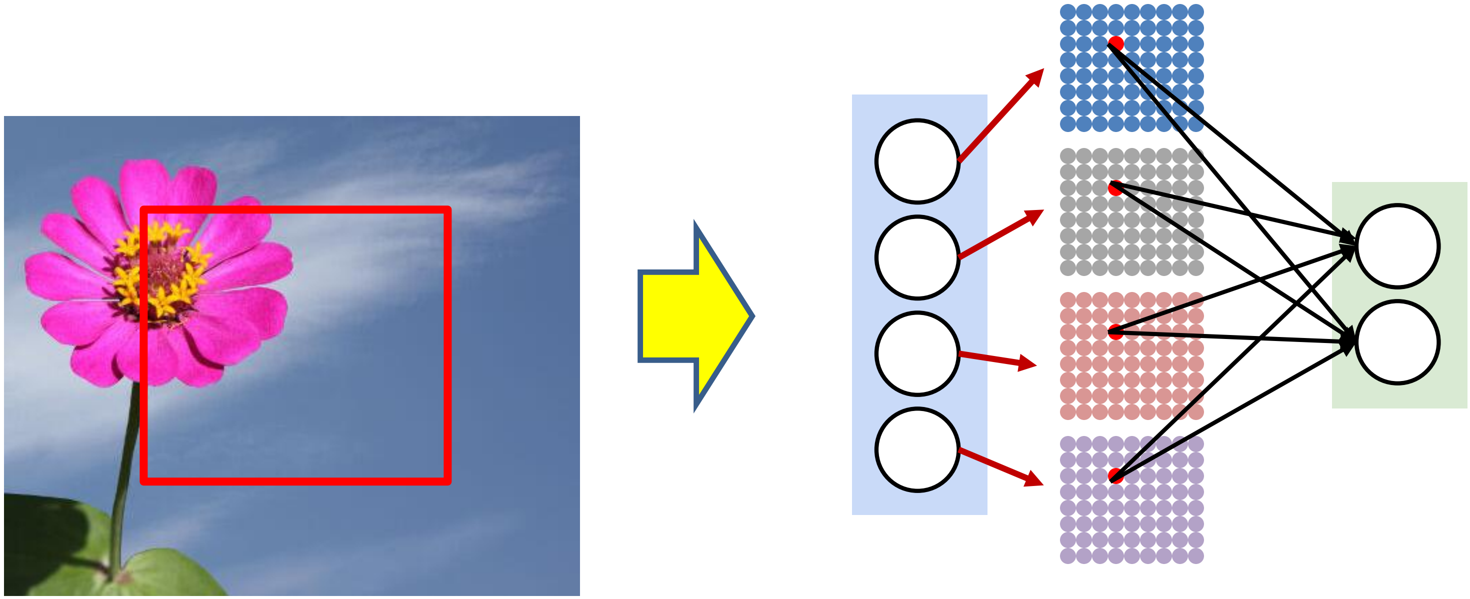
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture
- Eventually, we can arrange the outputs from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

Scanning: A closer look



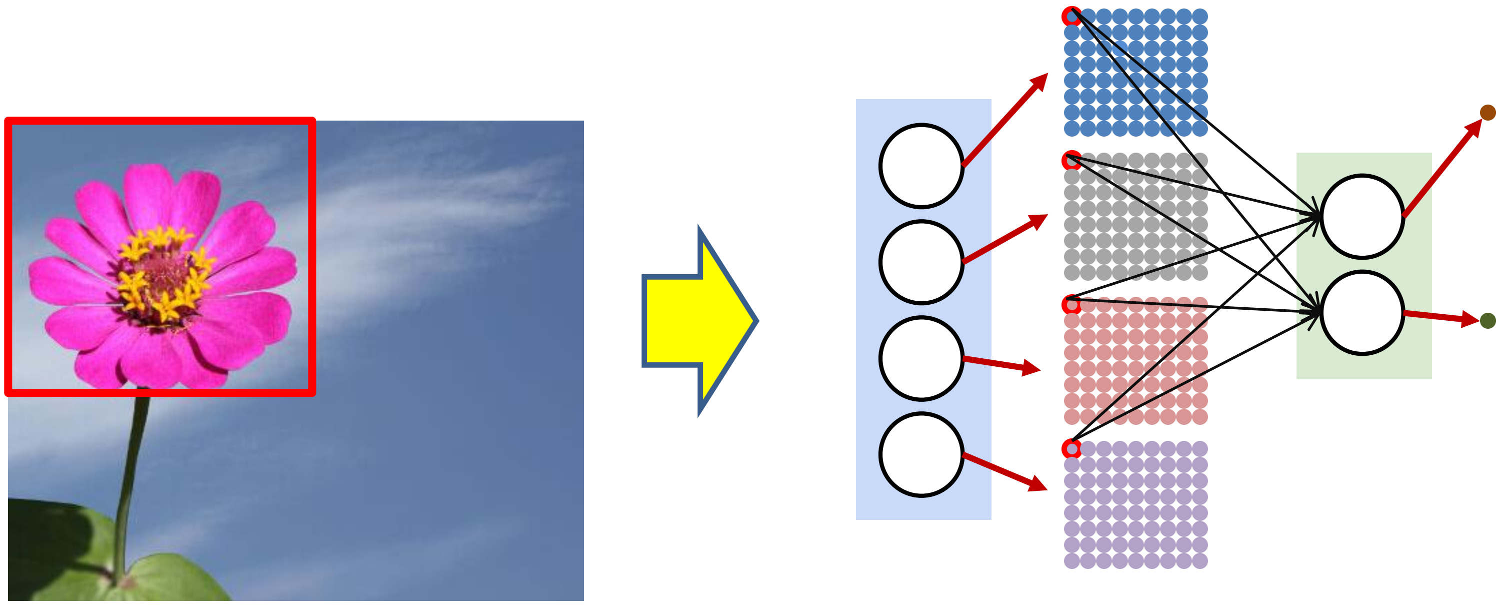
- Similarly, each perceptron's outputs from each of the scanned positions can be arranged as a rectangular pattern

Scanning: A closer look



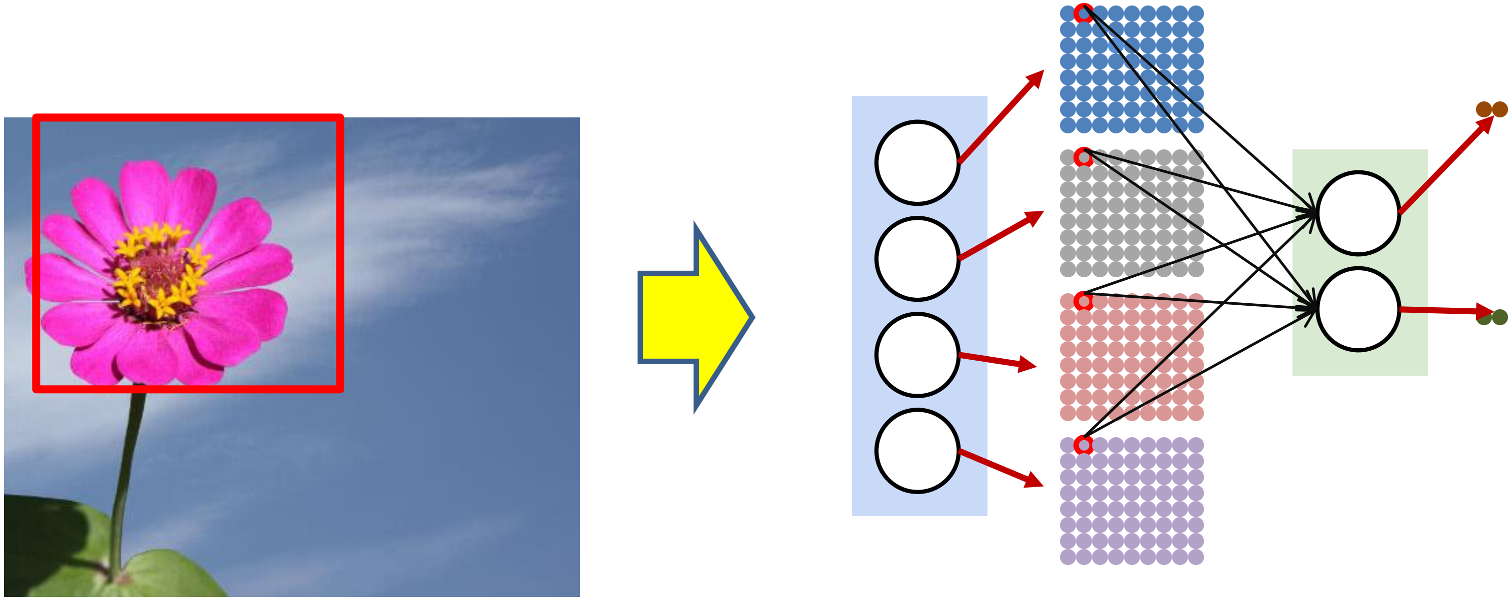
- To classify a specific “patch” in the image, we send the first level activations from the positions corresponding to that position to the next layer

Scanning: A closer look



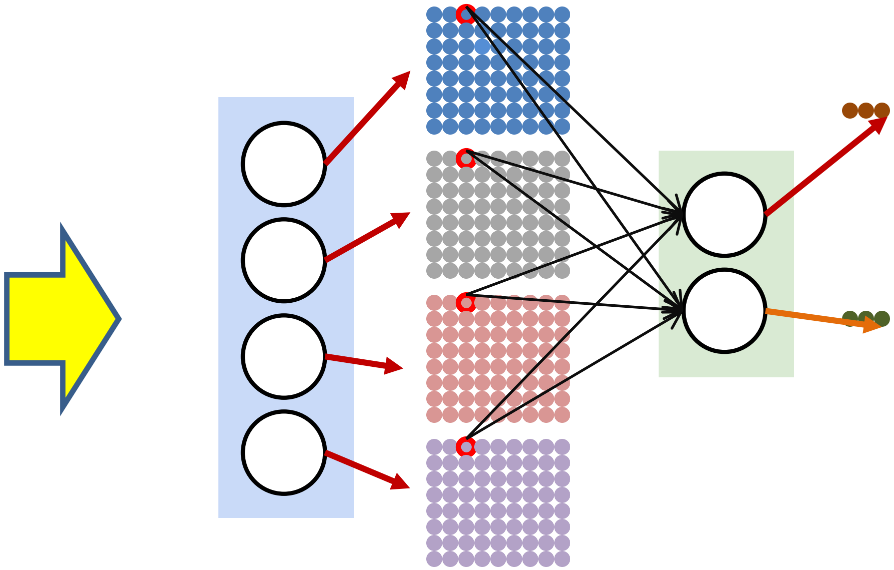
- We can recurse the logic
  - The second level neurons too are “**scanning**” the rectangular outputs of the first-level neurons
  - (Un)like the first level, they are jointly scanning *multiple* “pictures”
    - Each location in the output of the second level neuron considers the corresponding locations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

Scanning: A closer look



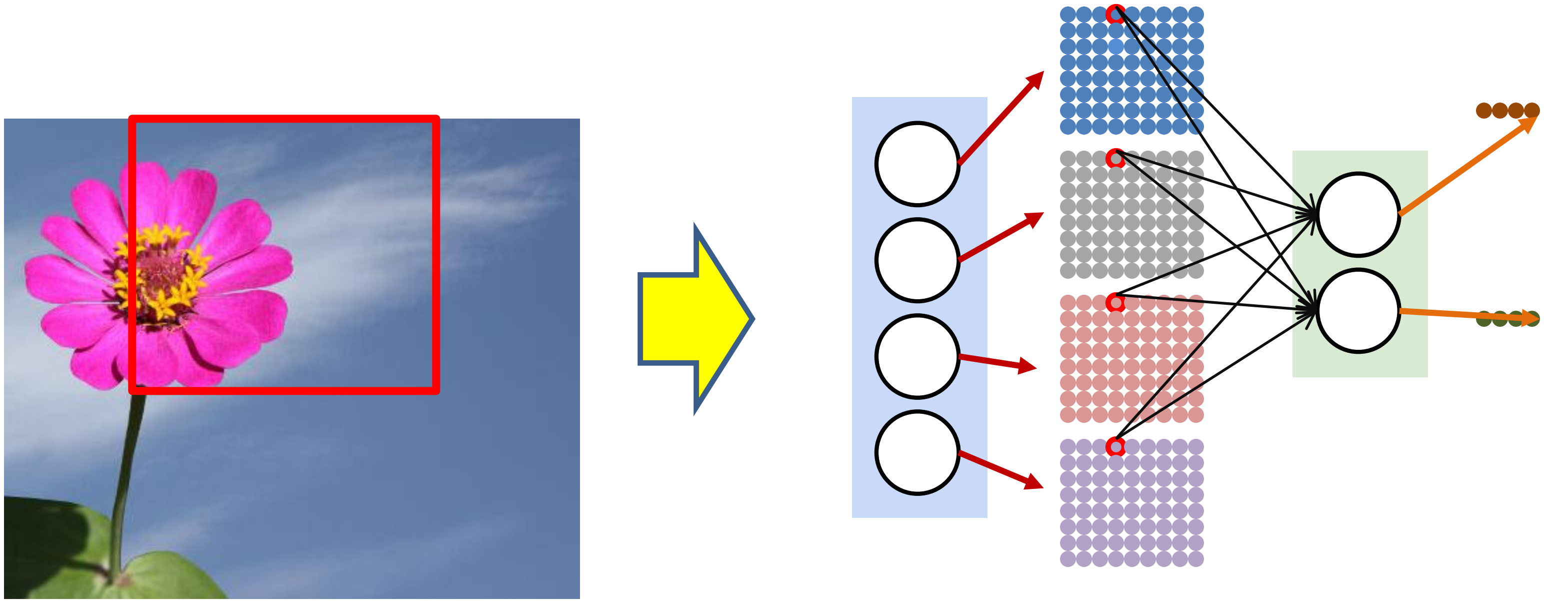
- We can recurse the logic
  - The second level neurons too are "**scanning**" the rectangular outputs of the first-level neurons
  - (Un)like the first level, they are jointly scanning *multiple* "pictures"
    - Each location in the output of the second level neuron considers the corresponding locations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

Scanning: A closer look



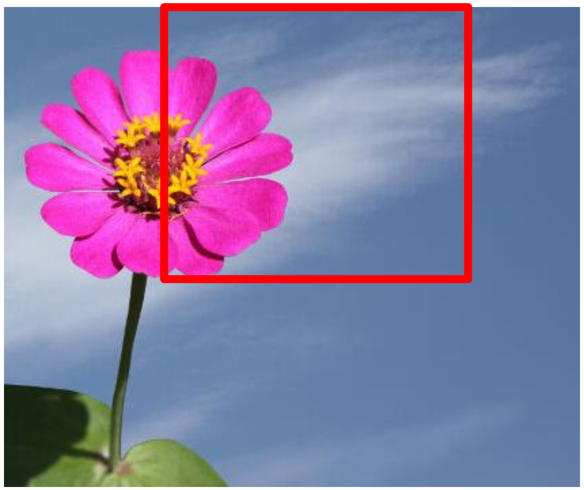
- We can recurse the logic
  - The second level neurons too are "**scanning**" the rectangular outputs of the first-level neurons
  - (Un)like the first level, they are jointly scanning *multiple* "pictures"
    - Each location in the output of the second level neuron considers the corresponding locations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

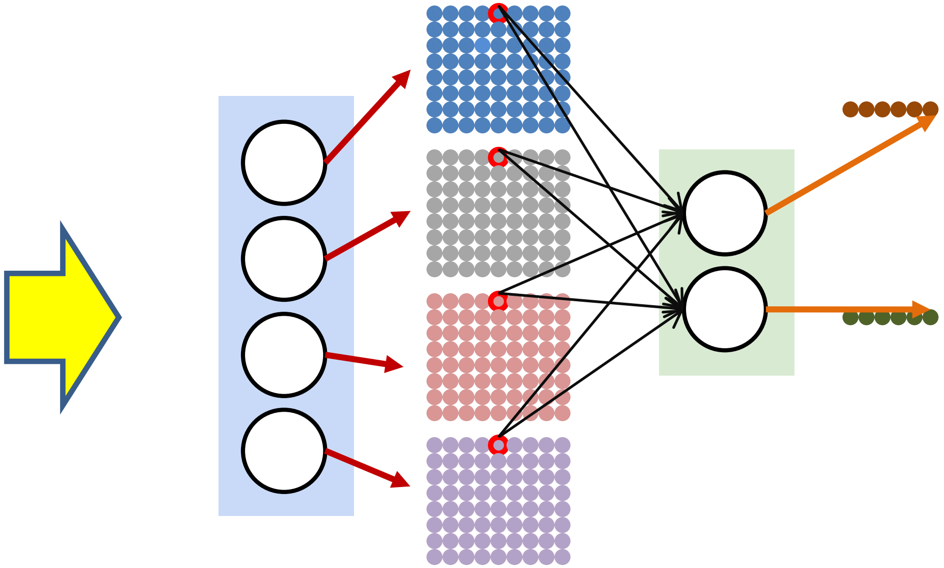
Scanning: A closer look



- 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

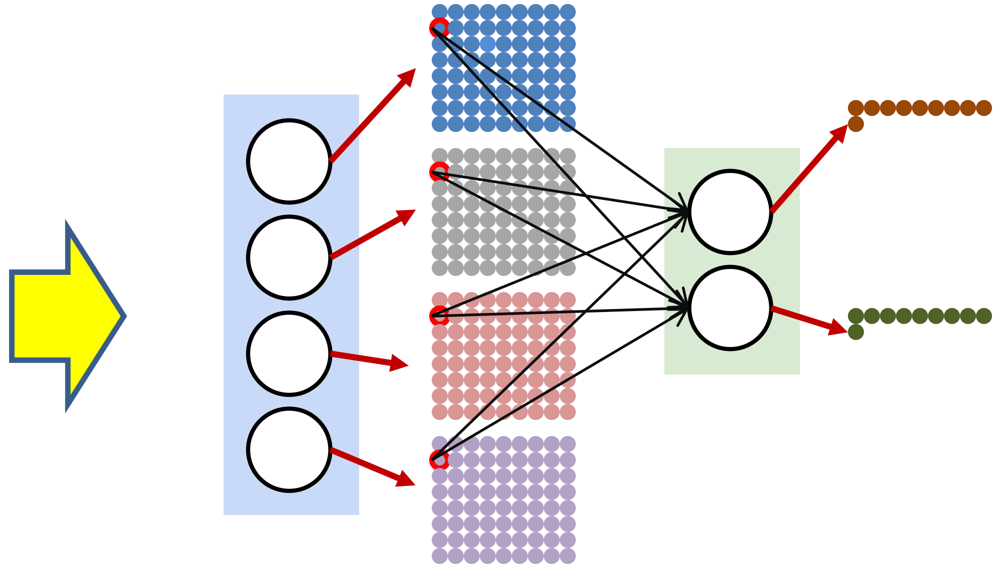
Scanning: A closer look





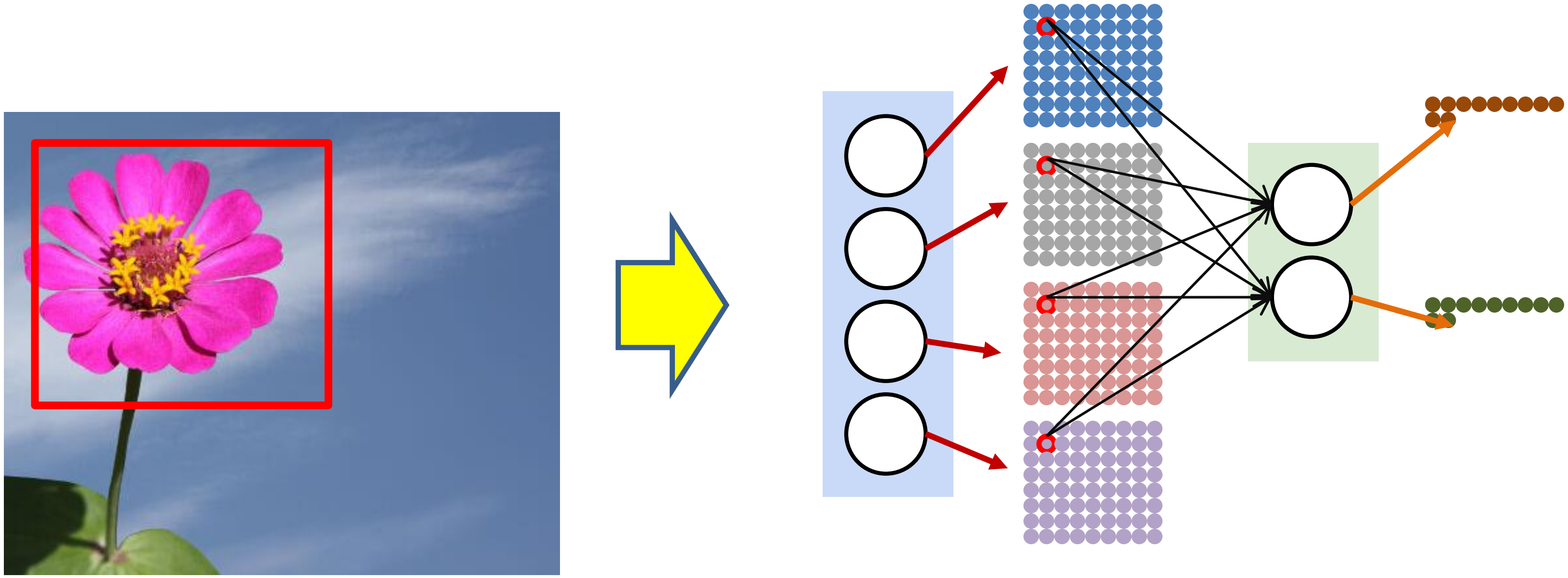
- We can recurse the logic
  - The second level neurons too are "**scanning**" the rectangular outputs of the first-level neurons
  - (Un)like the first level, they are jointly scanning *multiple* "pictures"
    - Each location in the output of the second level neuron considers the corresponding locations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

Scanning: A closer look



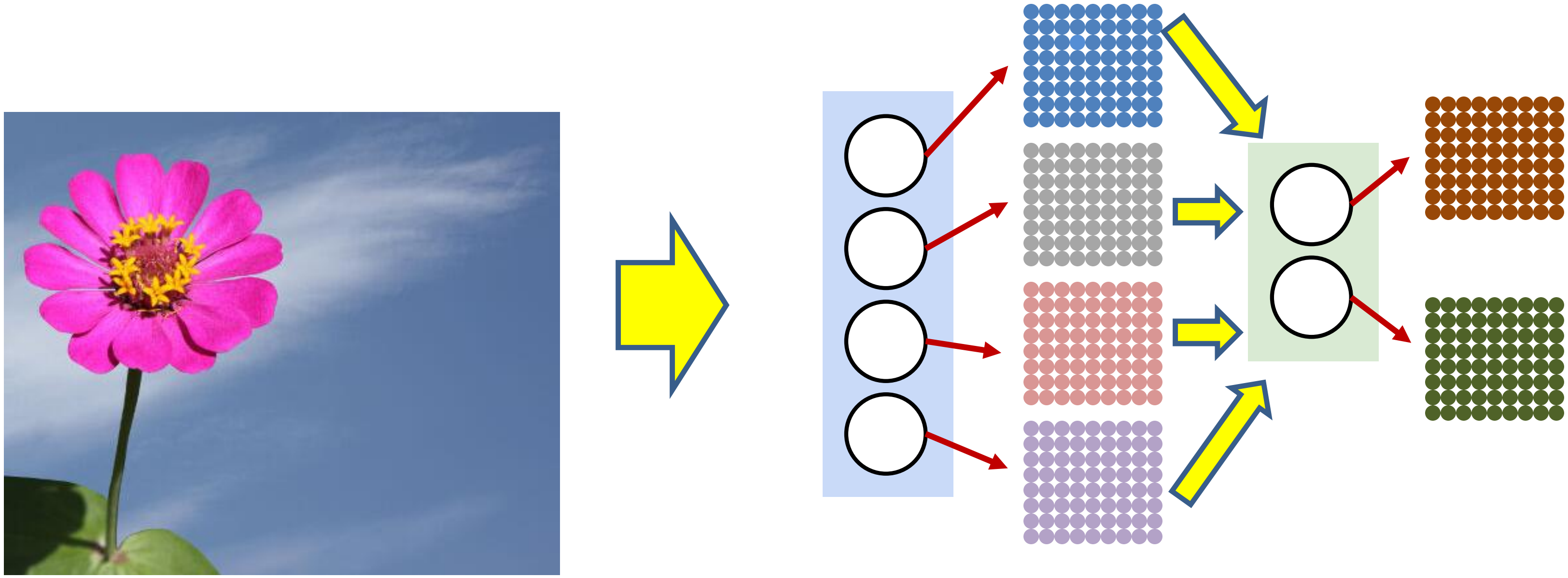
- We can recurse the logic
  - The second level neurons too are "**scanning**" the rectangular outputs of the first-level neurons
  - (Un)like the first level, they are jointly scanning *multiple* "pictures"
    - Each location in the output of the second level neuron considers the corresponding locations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

Scanning: A closer look



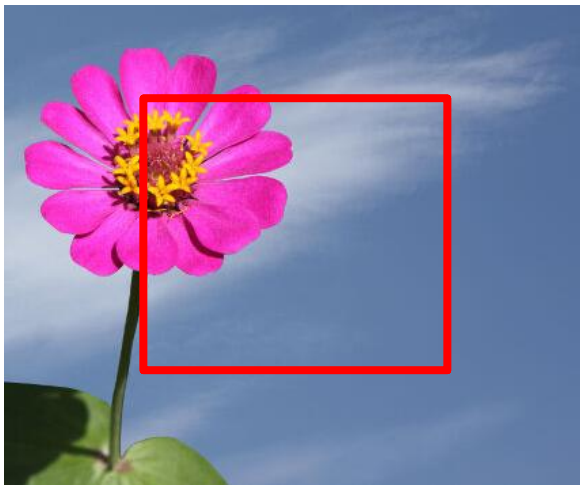
- We can recurse the logic
  - The second level neurons too are "**scanning**" the rectangular outputs of the first-level neurons
  - (Un)like the first level, they are jointly scanning *multiple* "pictures"
    - Each location in the output of the second level neuron considers the corresponding locations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

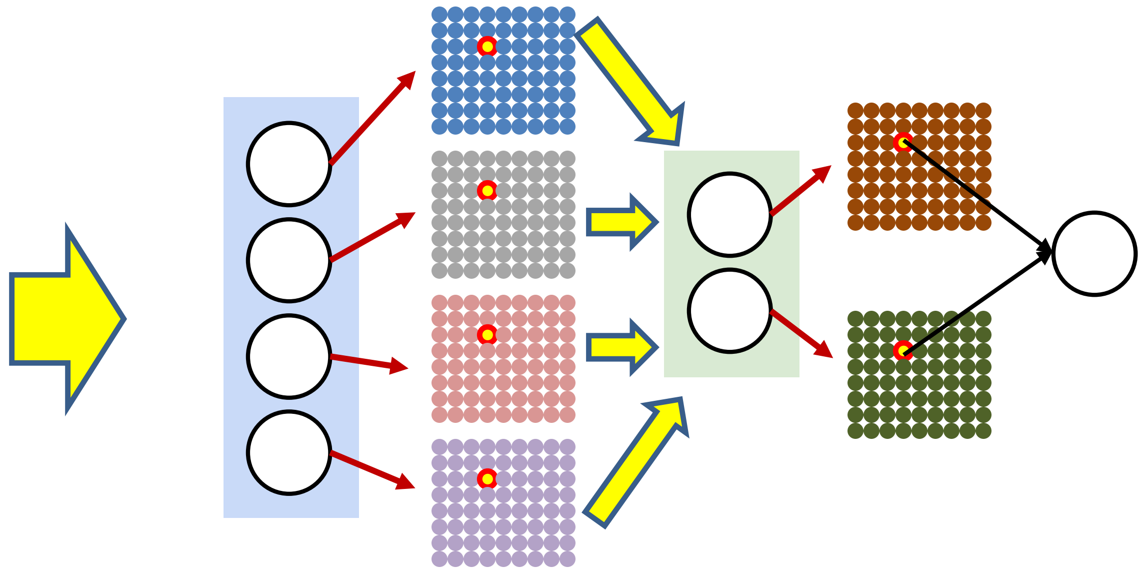
Scanning: A closer look



- 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

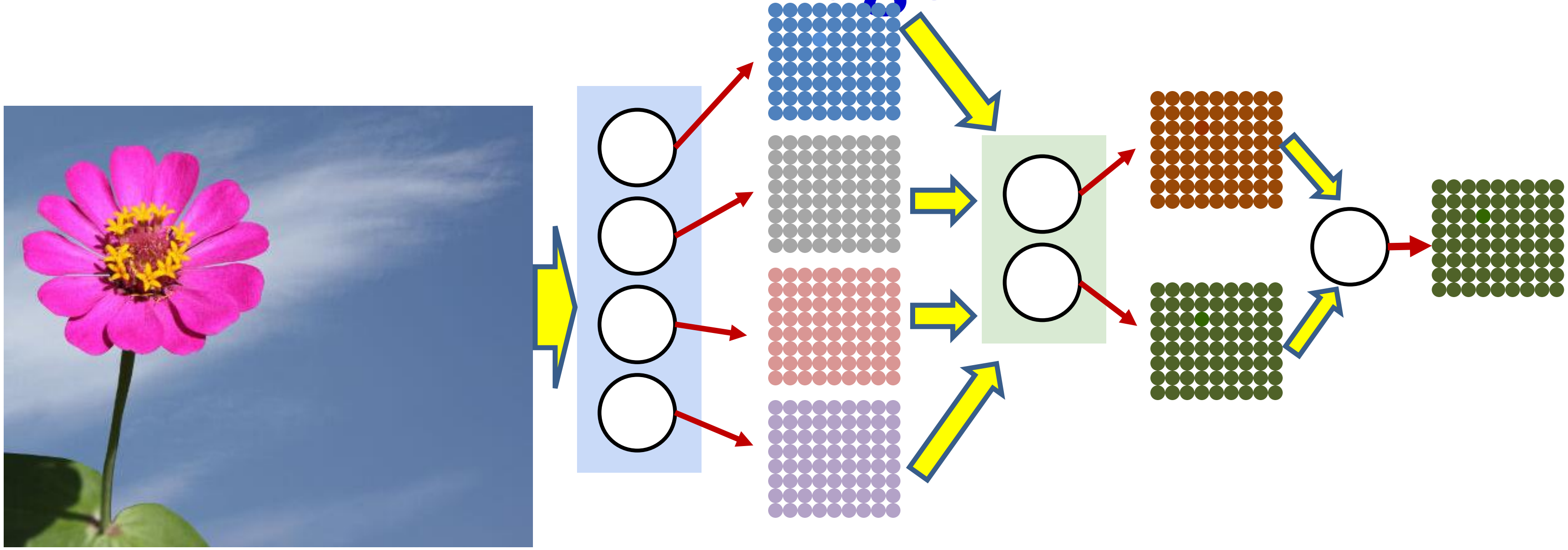
Scanning: A closer look





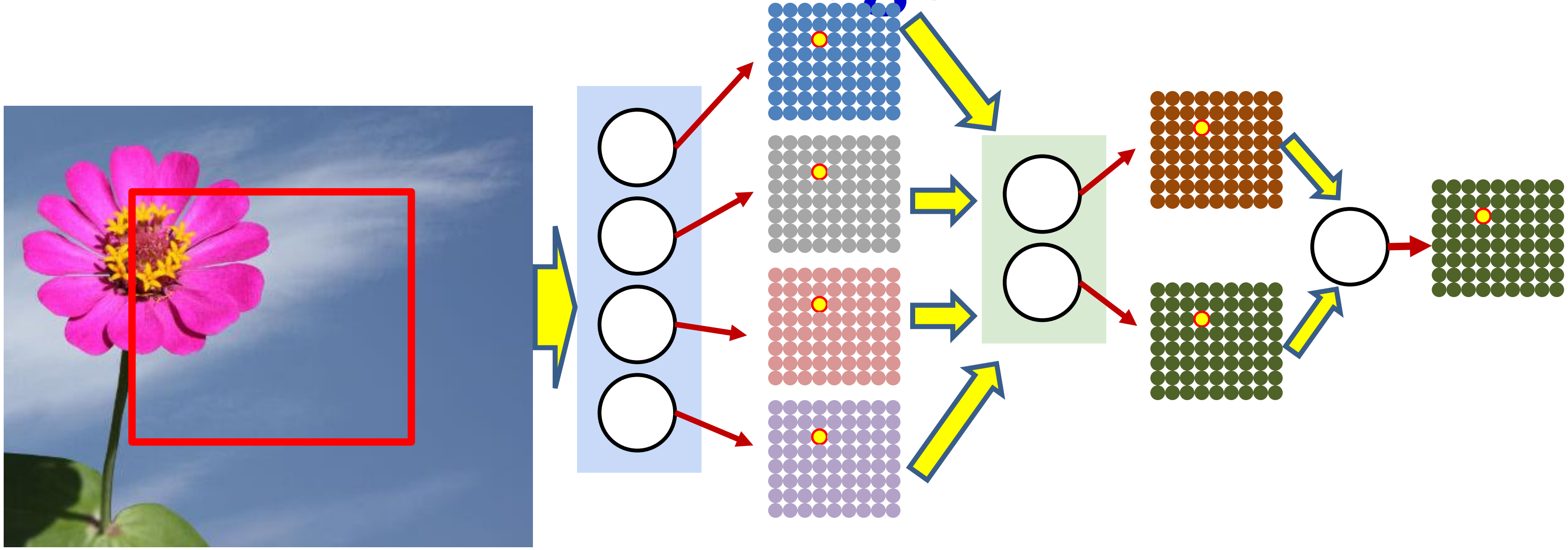
- To detect a picture ***at any location*** in the original image, the output layer must consider the corresponding outputs of the last hidden layer

Detecting a picture anywhere in the image?



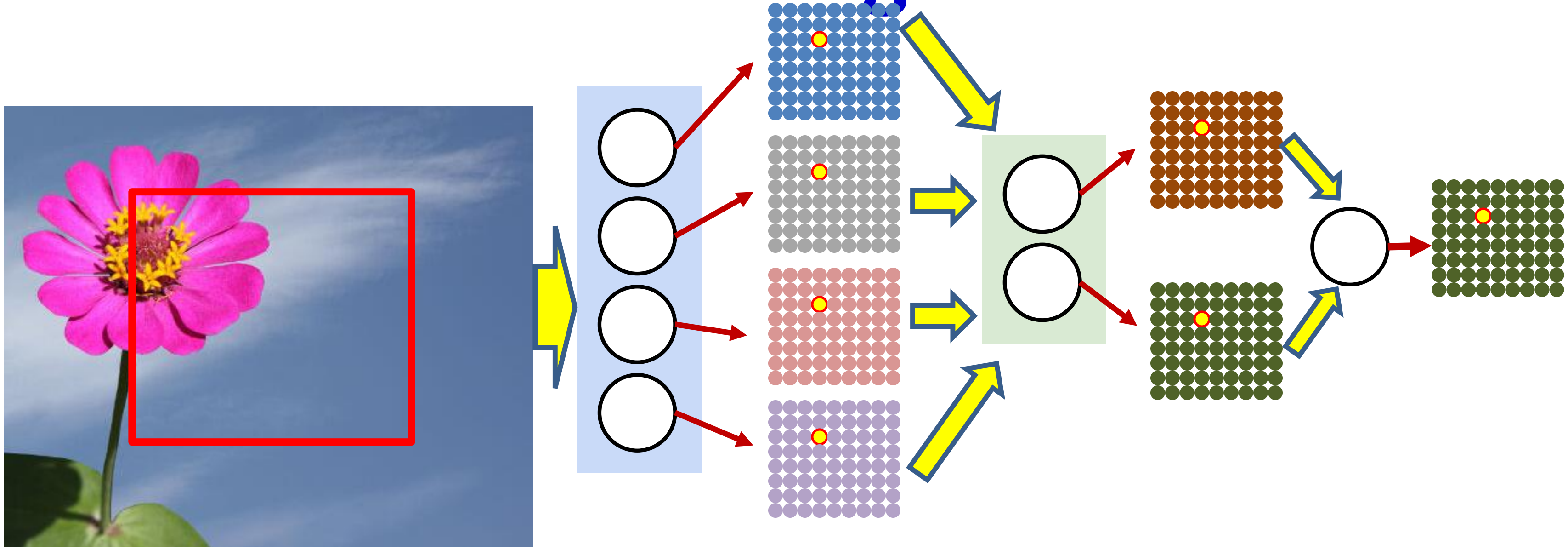
- Recursing the logic, we can create a map for the neurons in the next layer as well
  - The map is a flower detector for each location of the original image

Detecting a picture anywhere in the image?



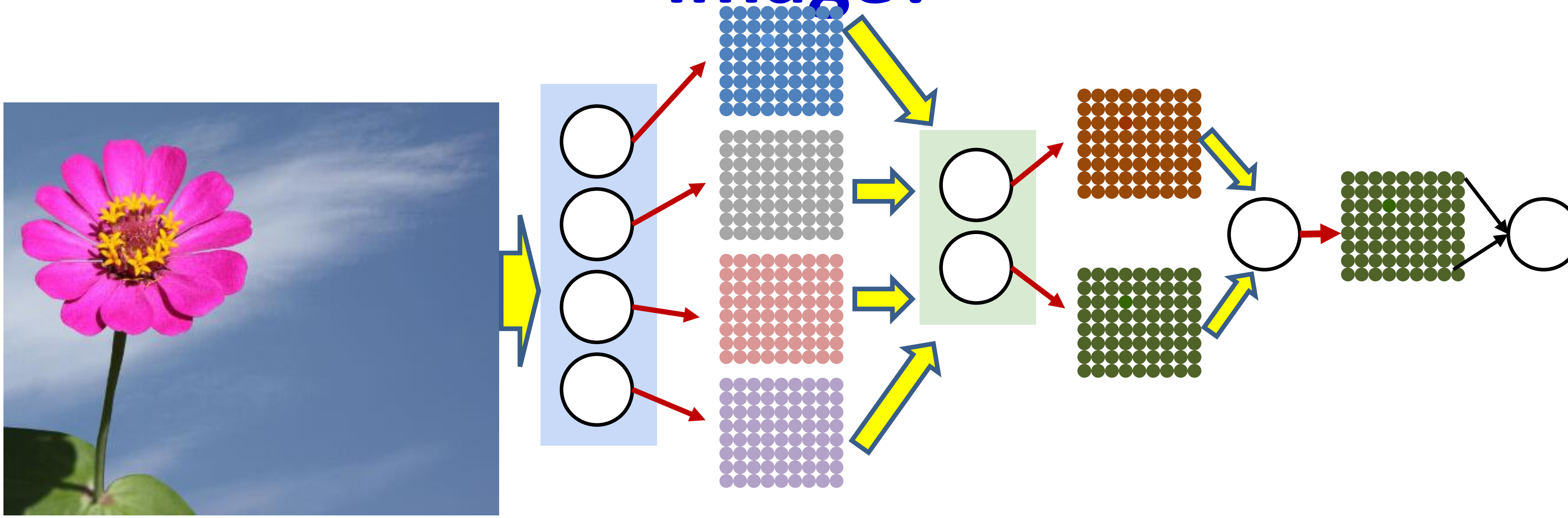
- To detect a picture ***at any location*** in the original image, the output layer must consider the corresponding output of the last hidden layer

Detecting a picture anywhere in the image?



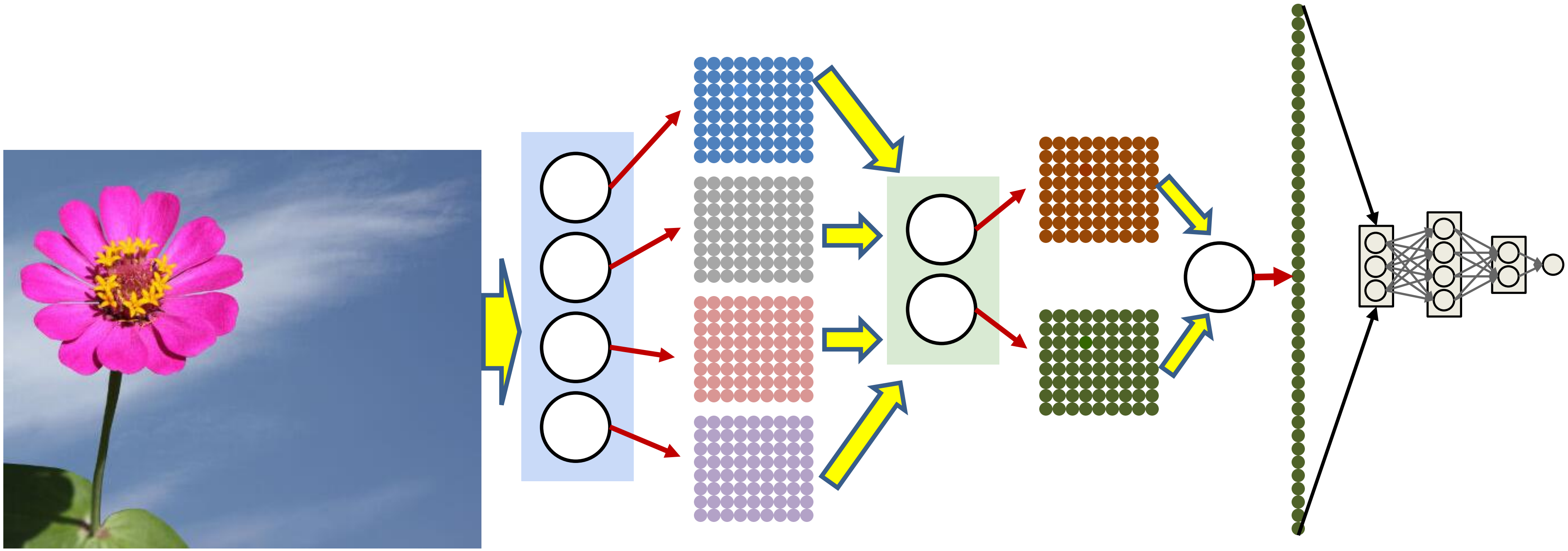
- To detect a picture ***at any location*** in the original image, the output layer must consider the corresponding output of the last hidden layer
- Actual problem? Is there a flower in the image
  - Not "detect the location of a flower"

Detecting a picture anywhere in the image?



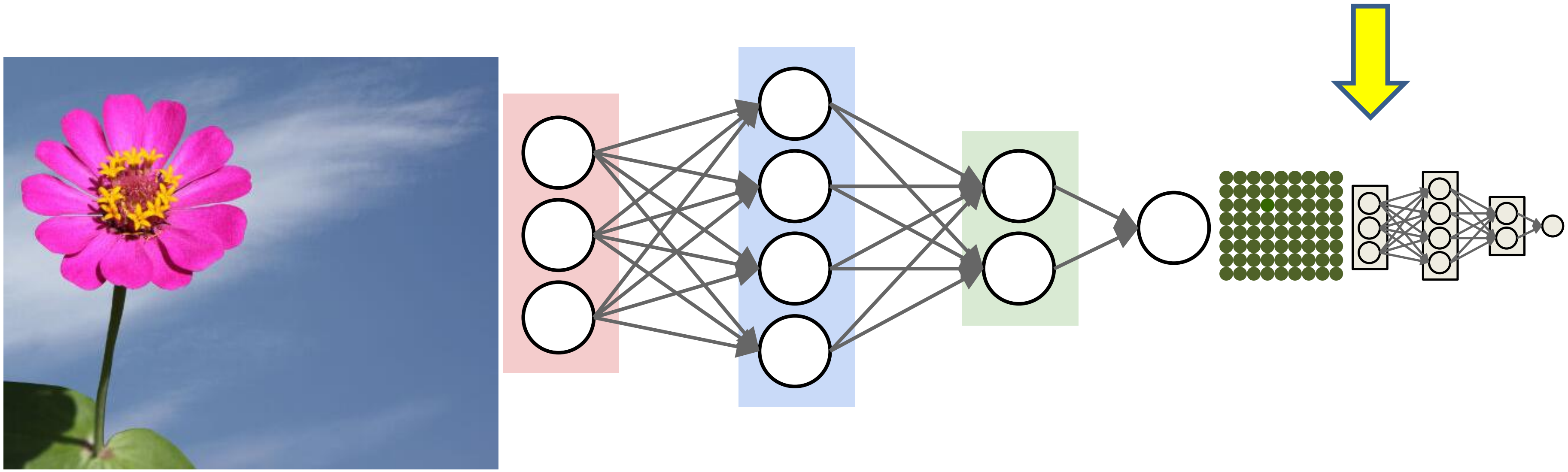
- Is there a flower in the picture?
- The output of the almost-last layer is also a grid/picture
- The entire grid can be sent into a final neuron that performs a logical “OR” to detect a picture
  - Finds the *max* output from all the positions
  - Or..

Detecting a picture in the image



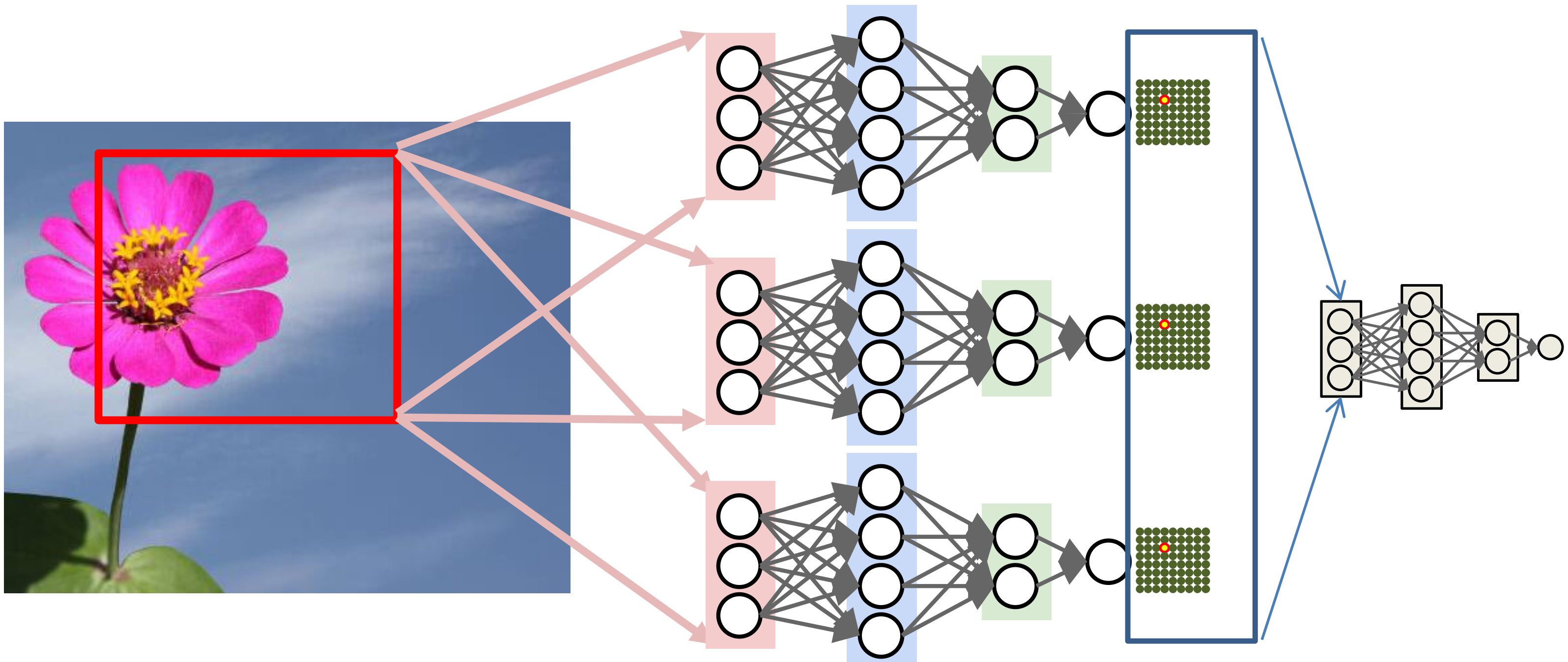
- Redrawing the final layer
  - "Flatten" the output of the neurons into a single block, since the arrangement is no longer important
  - Pass that through an MLP

Generalizing a bit



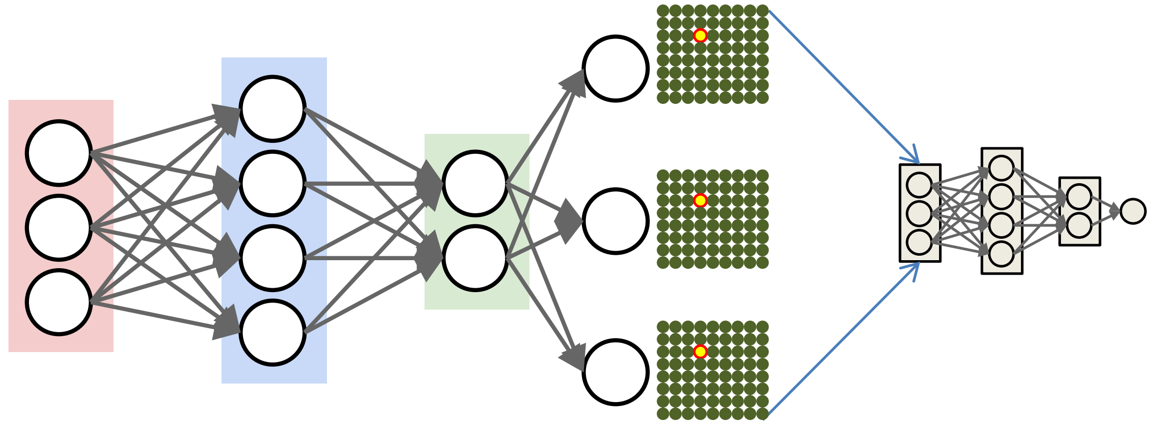
- At each location, the net searches for a flower
- The entire map of outputs is sent through a follow-up perceptron (or MLP) to determine if there really is a flower in the picture

Generalizing a bit



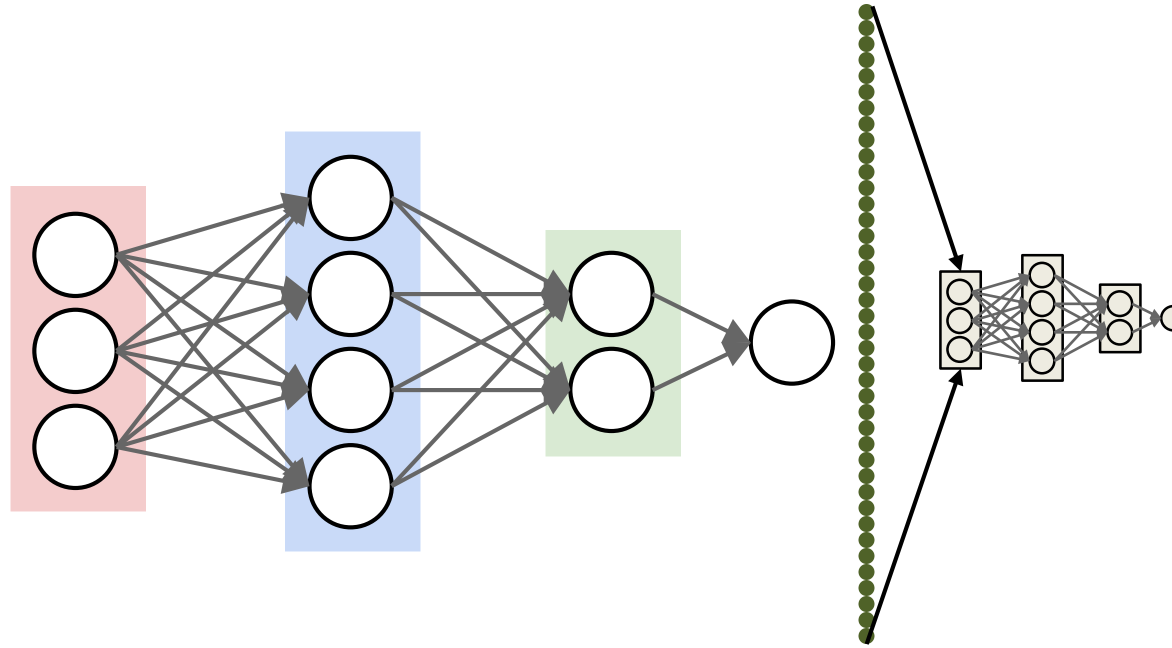
- The final objective is determine if the picture has a flower
- No need to use only one MLP to scan the image
  - Could use multiple MLPs..

Generalizing a bit..



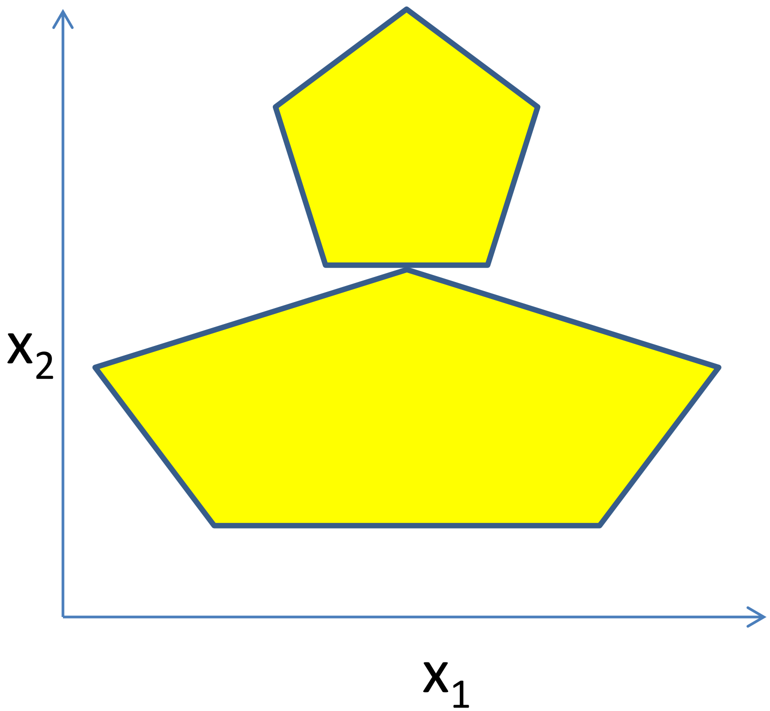
- The final objective is determine if the picture has a flower
- No need to use only one MLP to scan the image
  - Could use multiple MLPs..
  - Or a single larger MLPs with multiple output
    - Each providing independent evidence of the presence of a flower

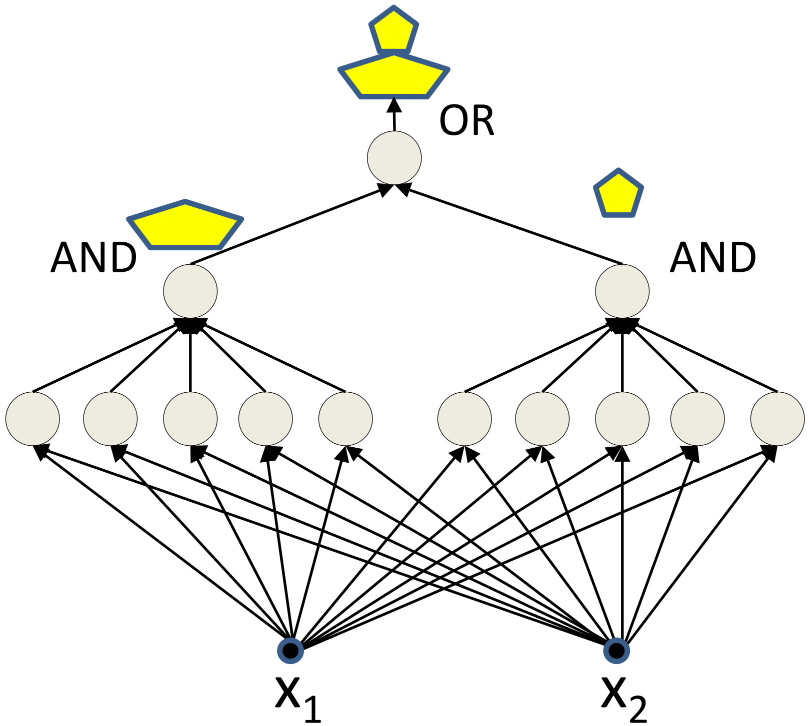
For simplicity..



- We will continue to assume the simple version of the model for the sake of explanation

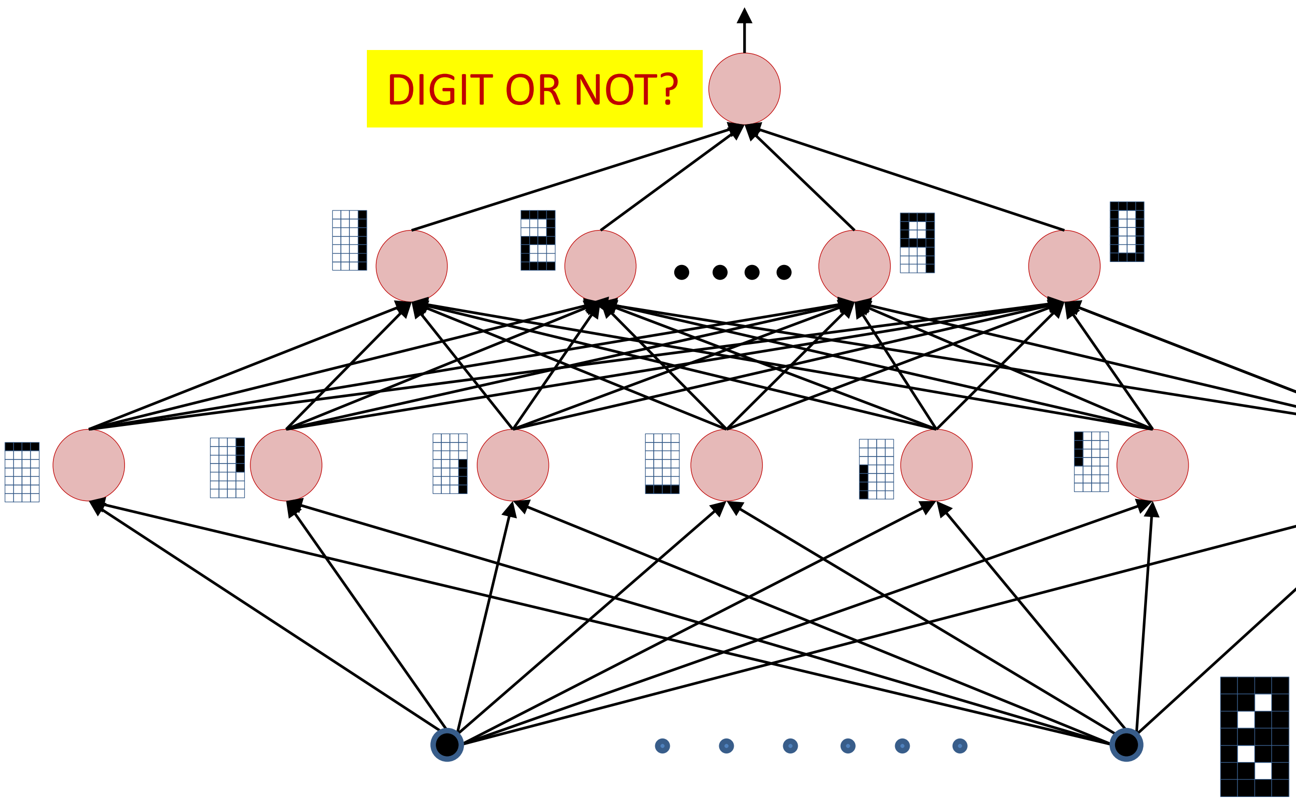
Recall: What does an MLP learn?





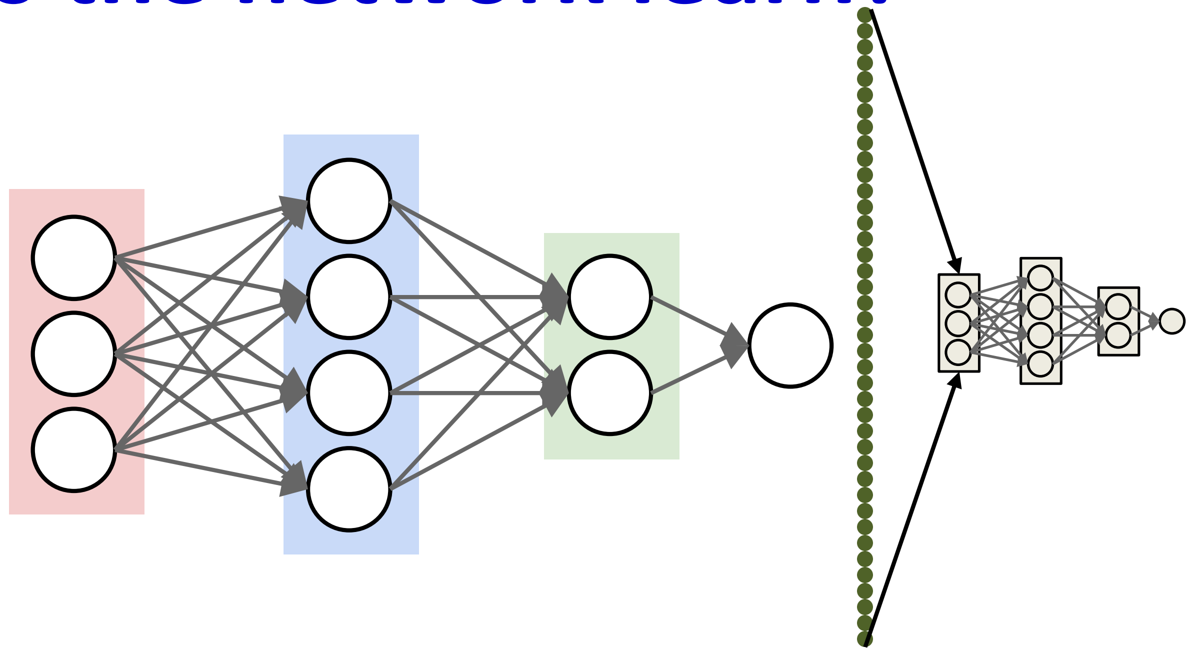
- The lowest layers of the network capture simple patterns
  - The linear decision boundaries in this example
- The next layer captures more complex patterns
  - The polygons
- The next one captures still more complex patterns..

Recall: How does an MLP represent patterns



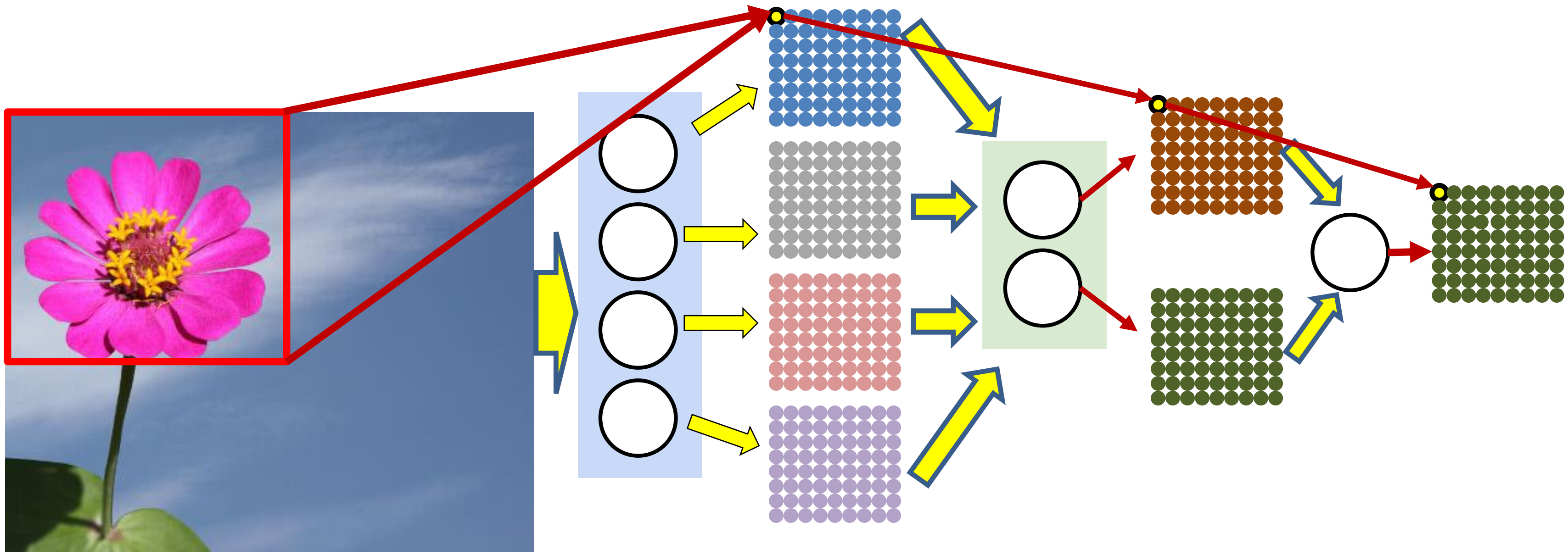
- **The neurons in an MLP *build up* complex patterns from simple pattern hierarchically**
  – Each layer learns to “detect” simple combinations of the patterns detected by earlier layers

Returning to our problem: What does the network learn?



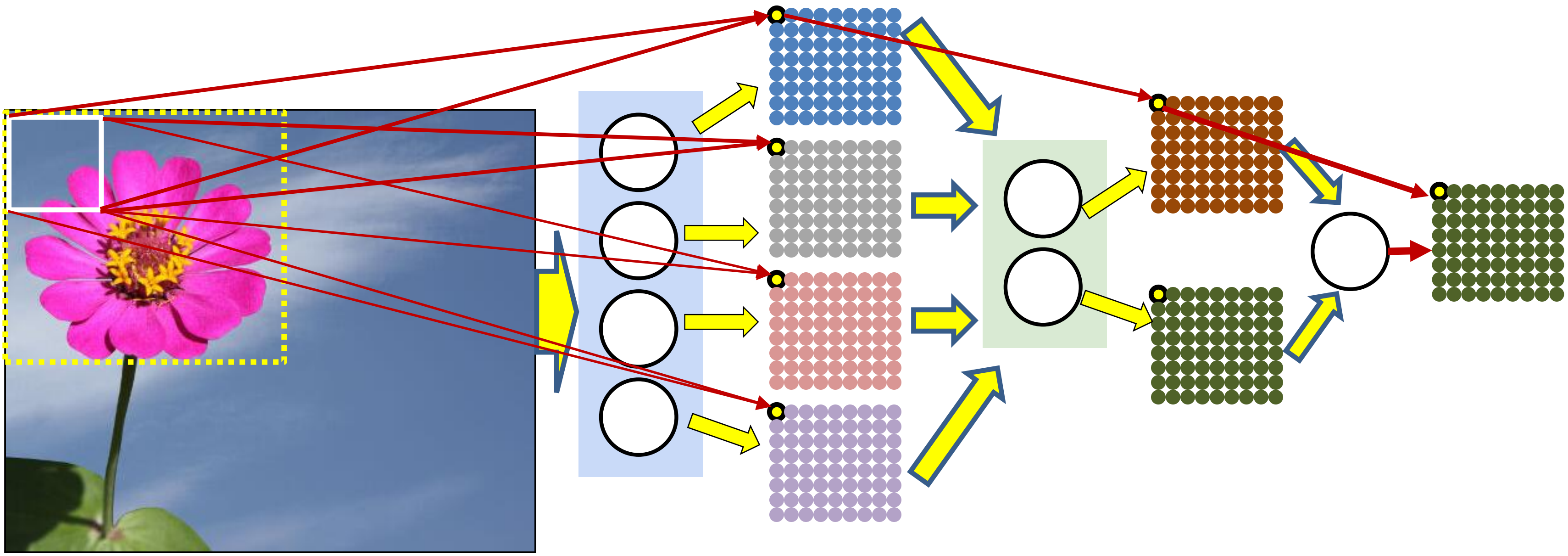
- The entire MLP looks for a flower-like pattern at each location

The behavior of the layers



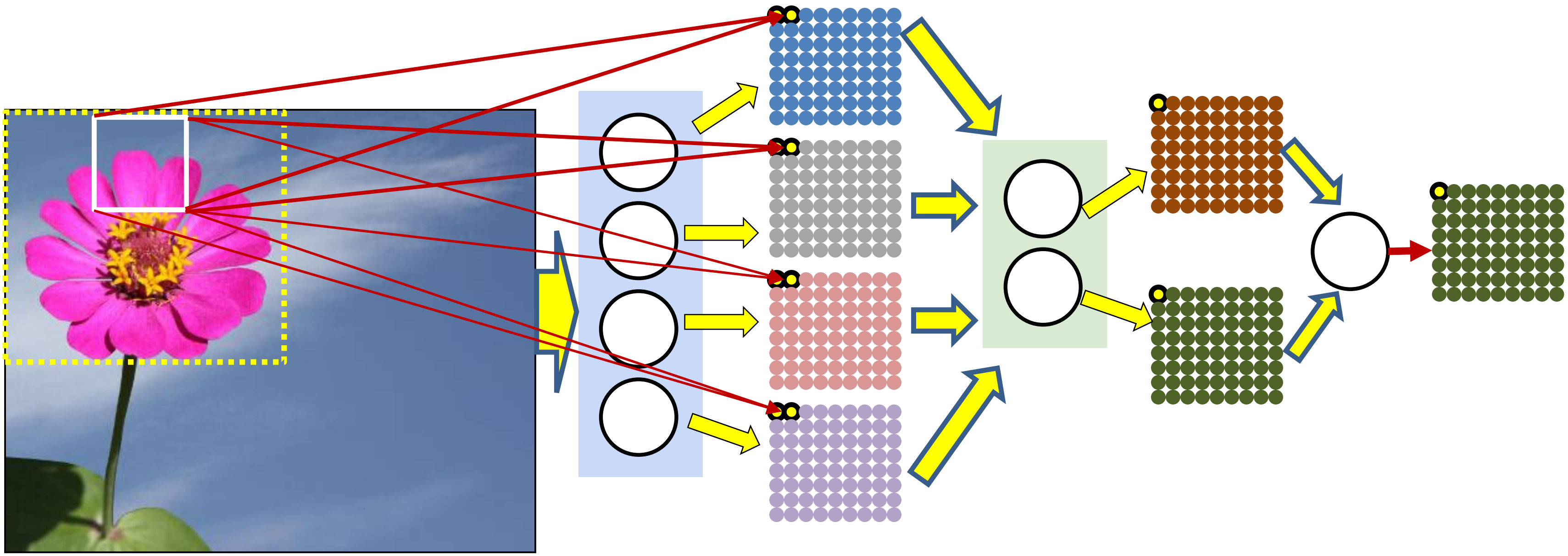
- The first layer neurons "look" at the entire "block" to extract block-level features
  - Subsequent layers only perform classification over these block-level features
- **The first layer neurons is responsible for evaluating the entire *block* of pixels**
  - **Subsequent layers only look at a *single*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

Distributing the scan



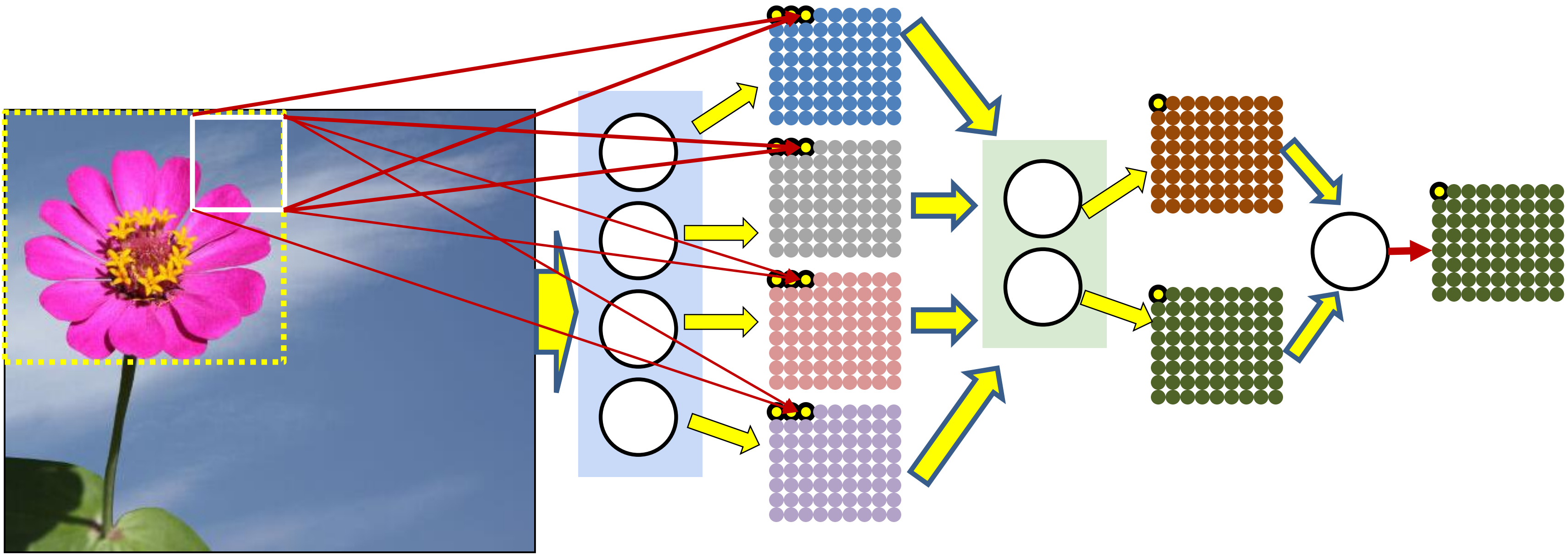
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates blocks of outputs from the first layer

Distributing the scan



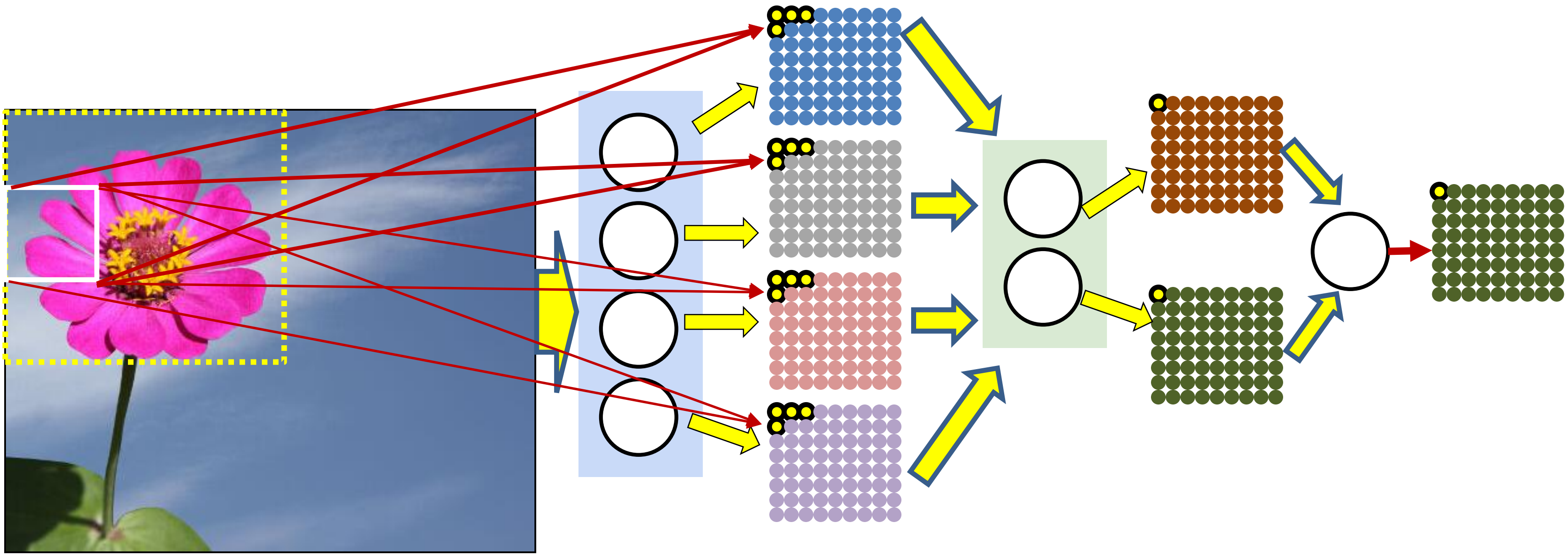
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates blocks of outputs from the first layer

Distributing the scan



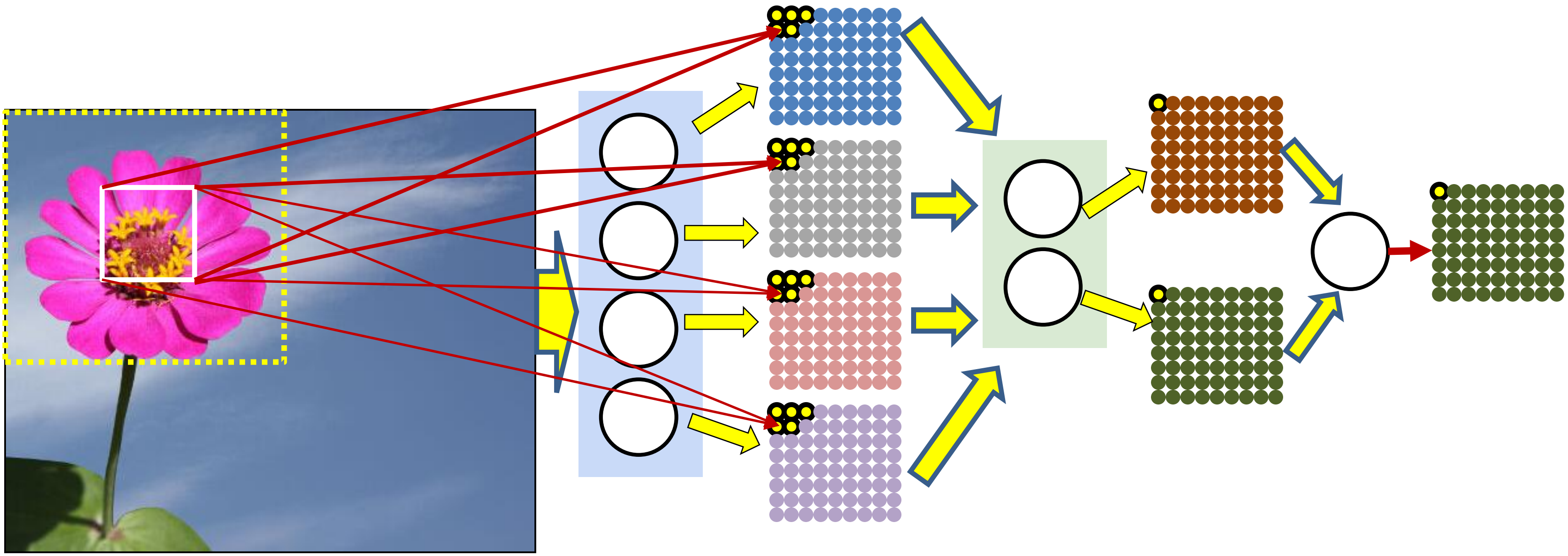
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates blocks of outputs from the first layer

Distributing the scan



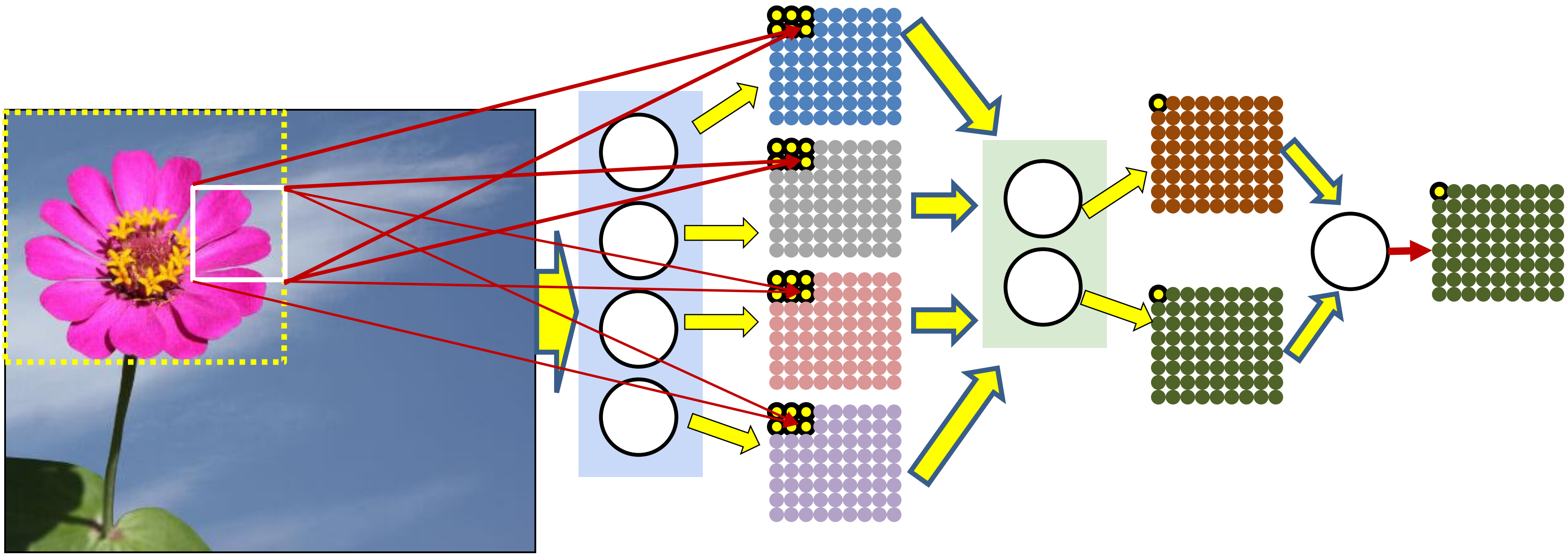
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates blocks of outputs from the first layer

Distributing the scan



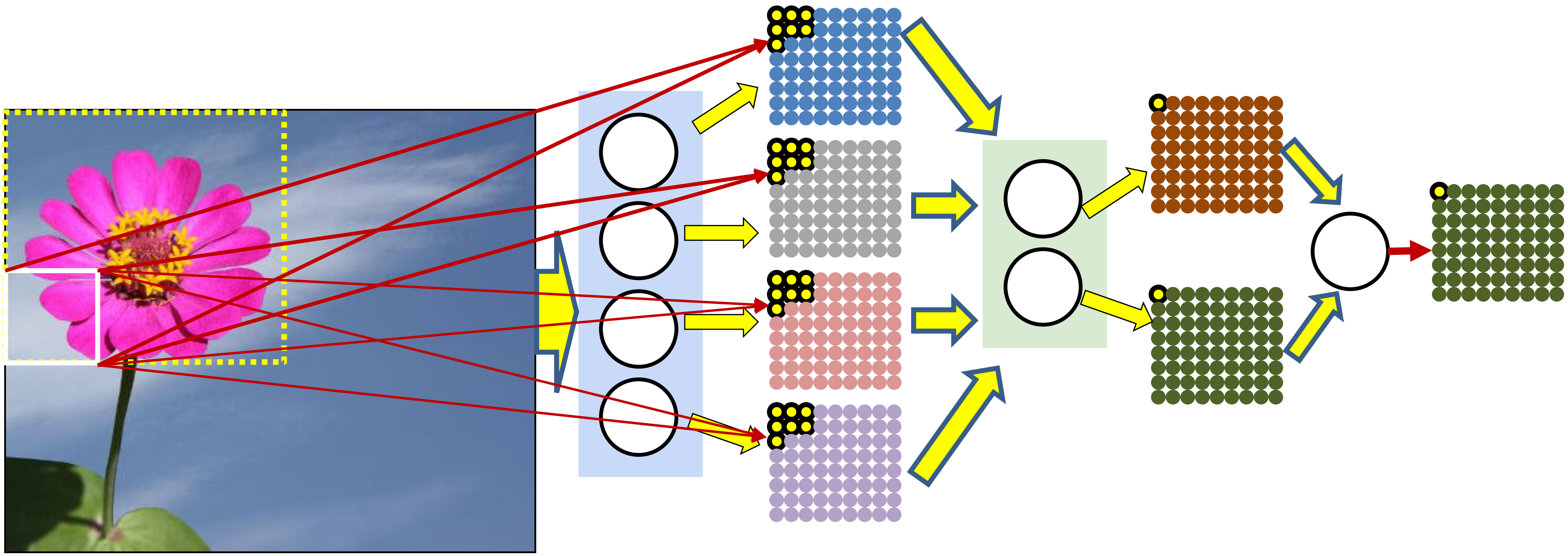
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates blocks of outputs from the first layer

Distributing the scan



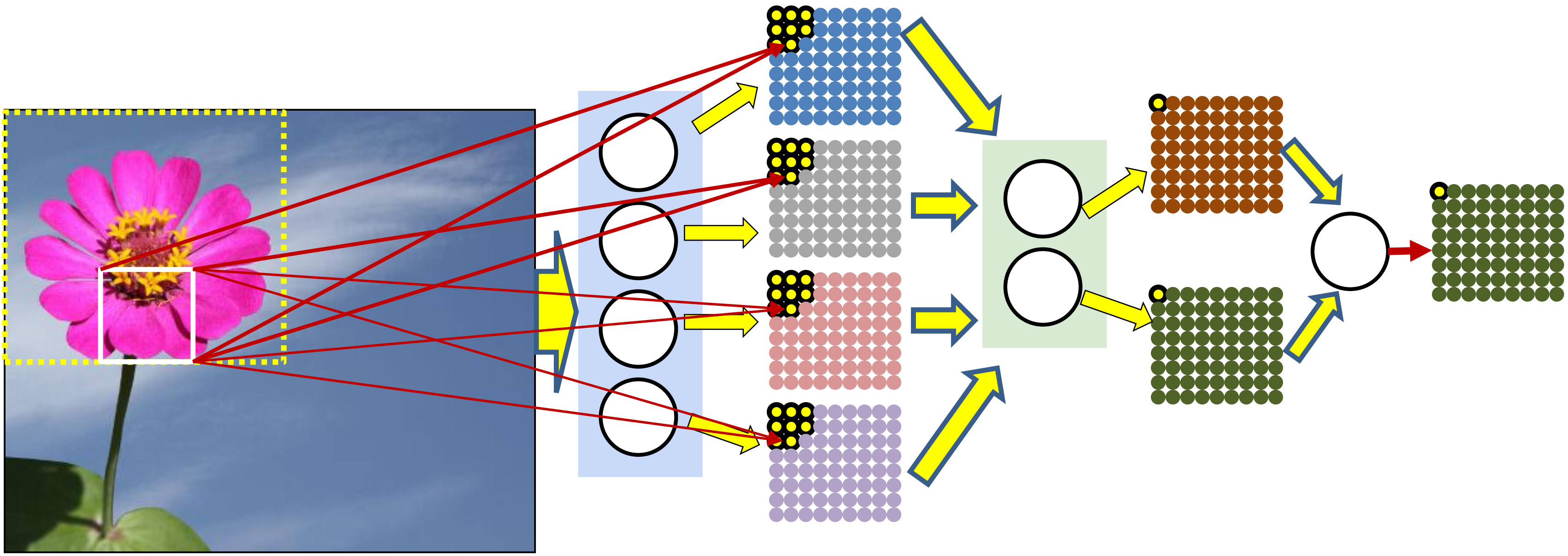
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates blocks of outputs from the first layer

Distributing the scan



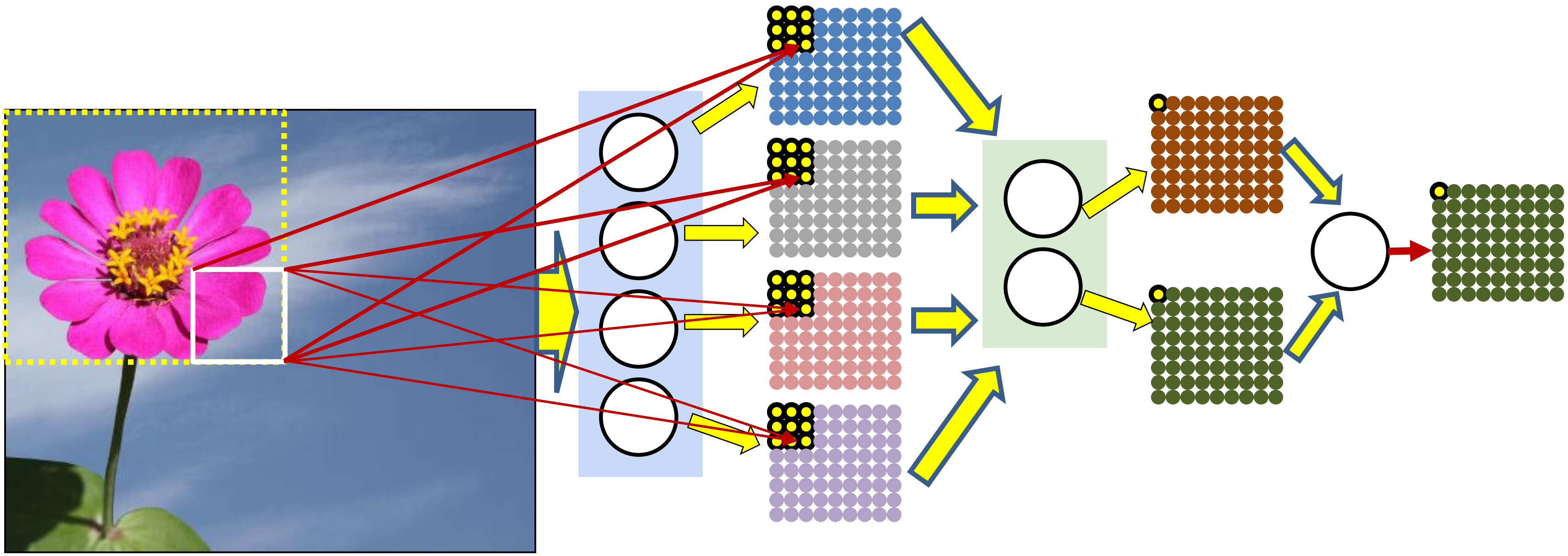
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates blocks of outputs from the first layer

Distributing the scan



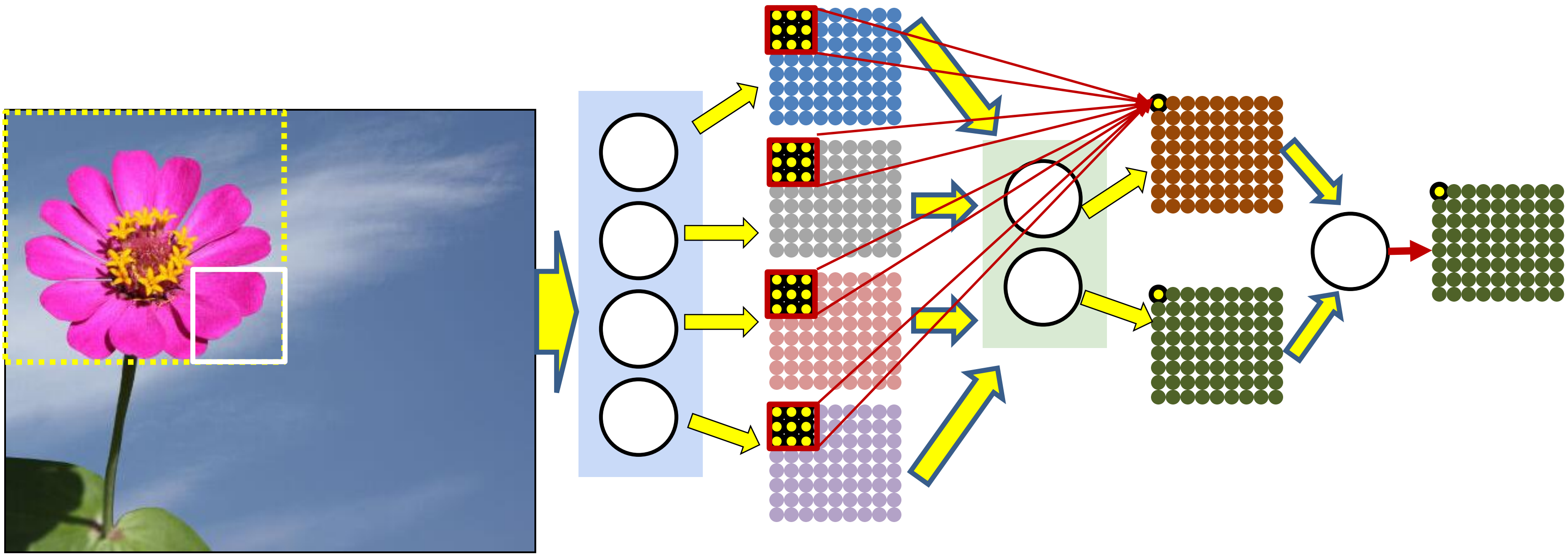
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates blocks of outputs from the first layer

Distributing the scan



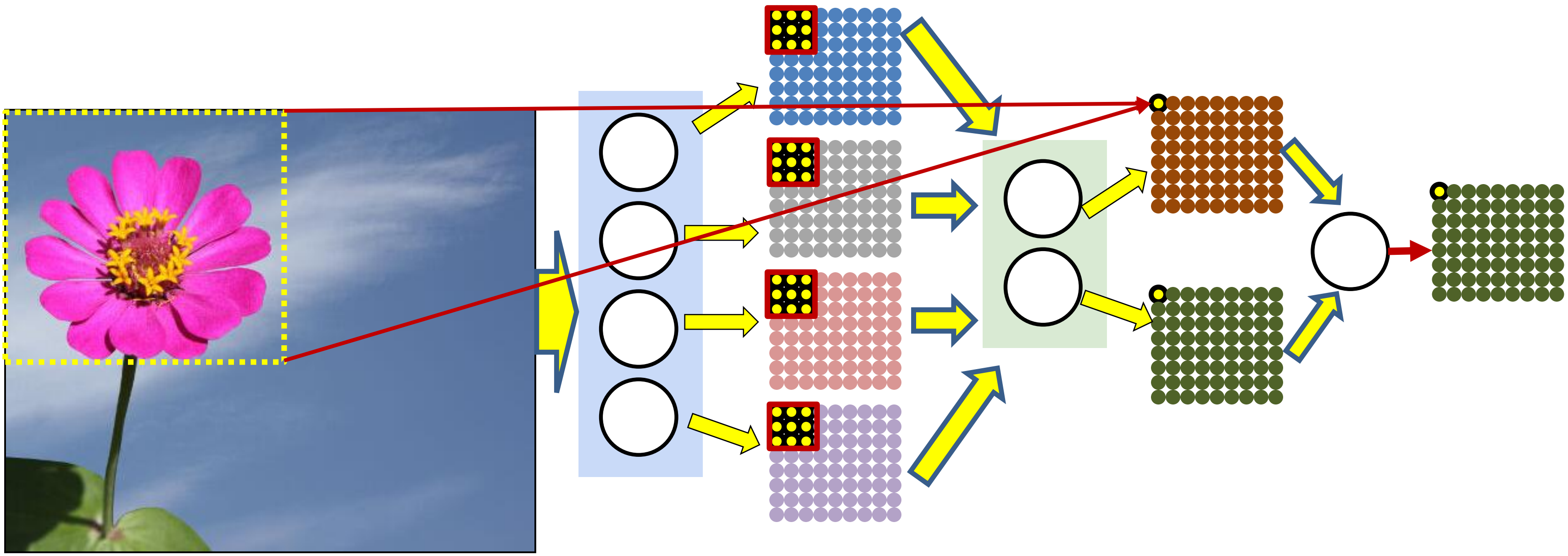
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates blocks of outputs from the first layer

Distributing the scan



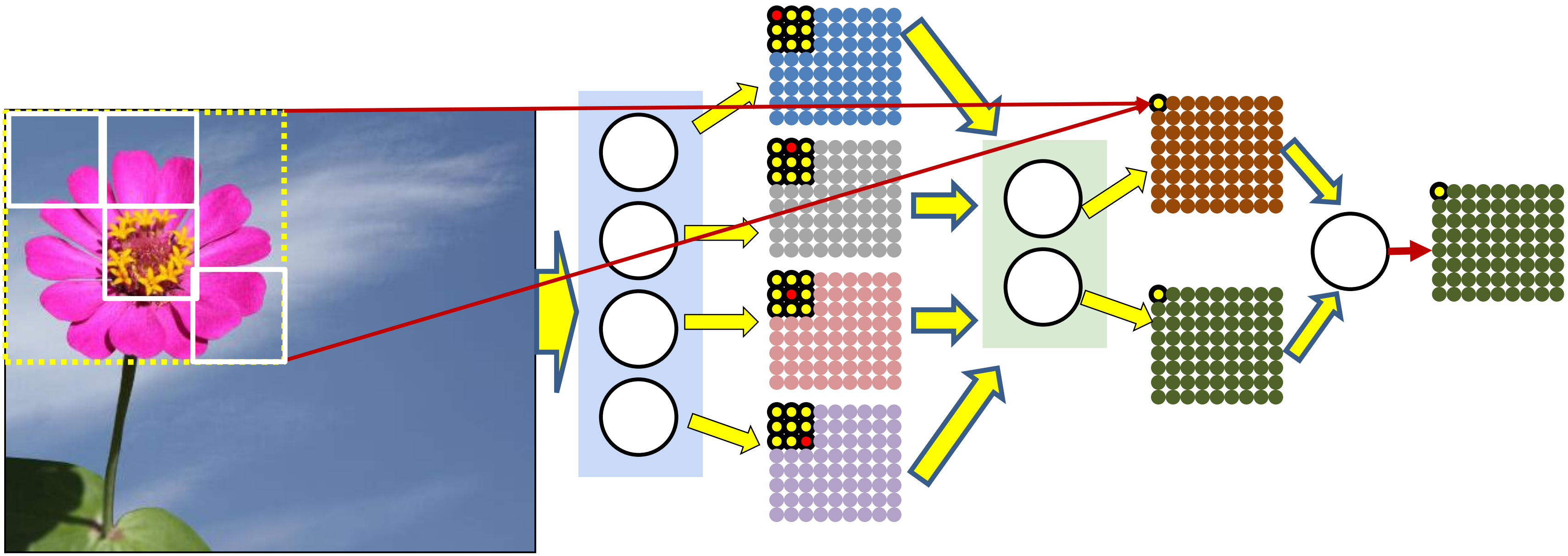
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates blocks of outputs from the first layer

Distributing the scan



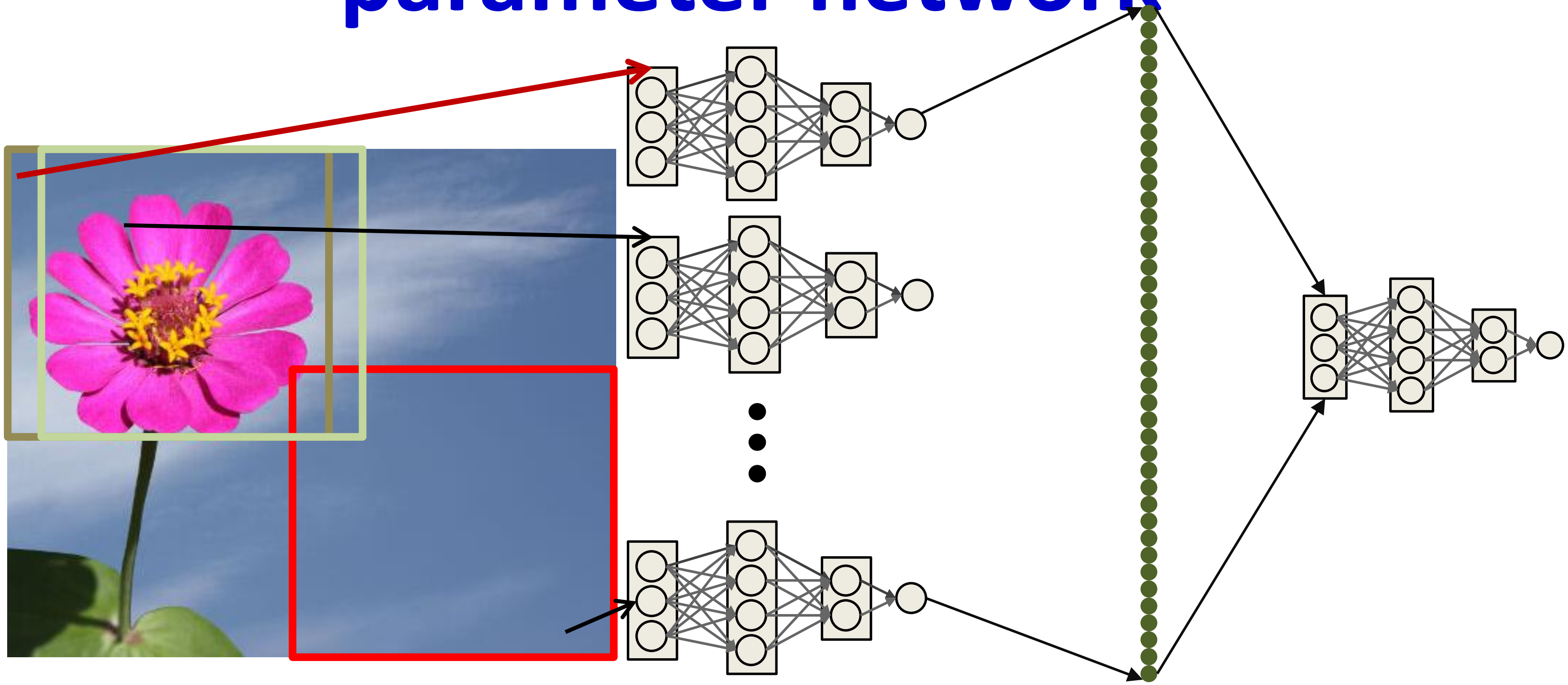
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates blocks of outputs from the first layer
  - This effectively evaluates the larger block of the original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

Distributing the scan



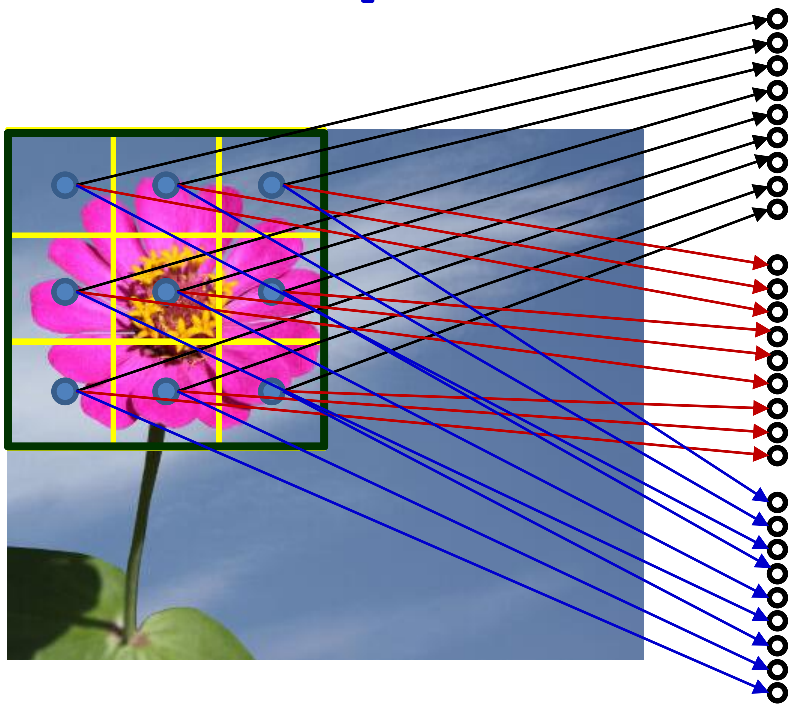
- The higher layer implicitly learns the *arrangement* of sub patterns that represents the larger pattern (the flower in this case)

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



- With a minor modification…

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

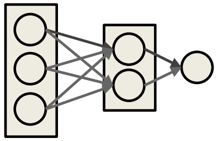


Each arrow represents an entire set of weights over the smaller cell

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

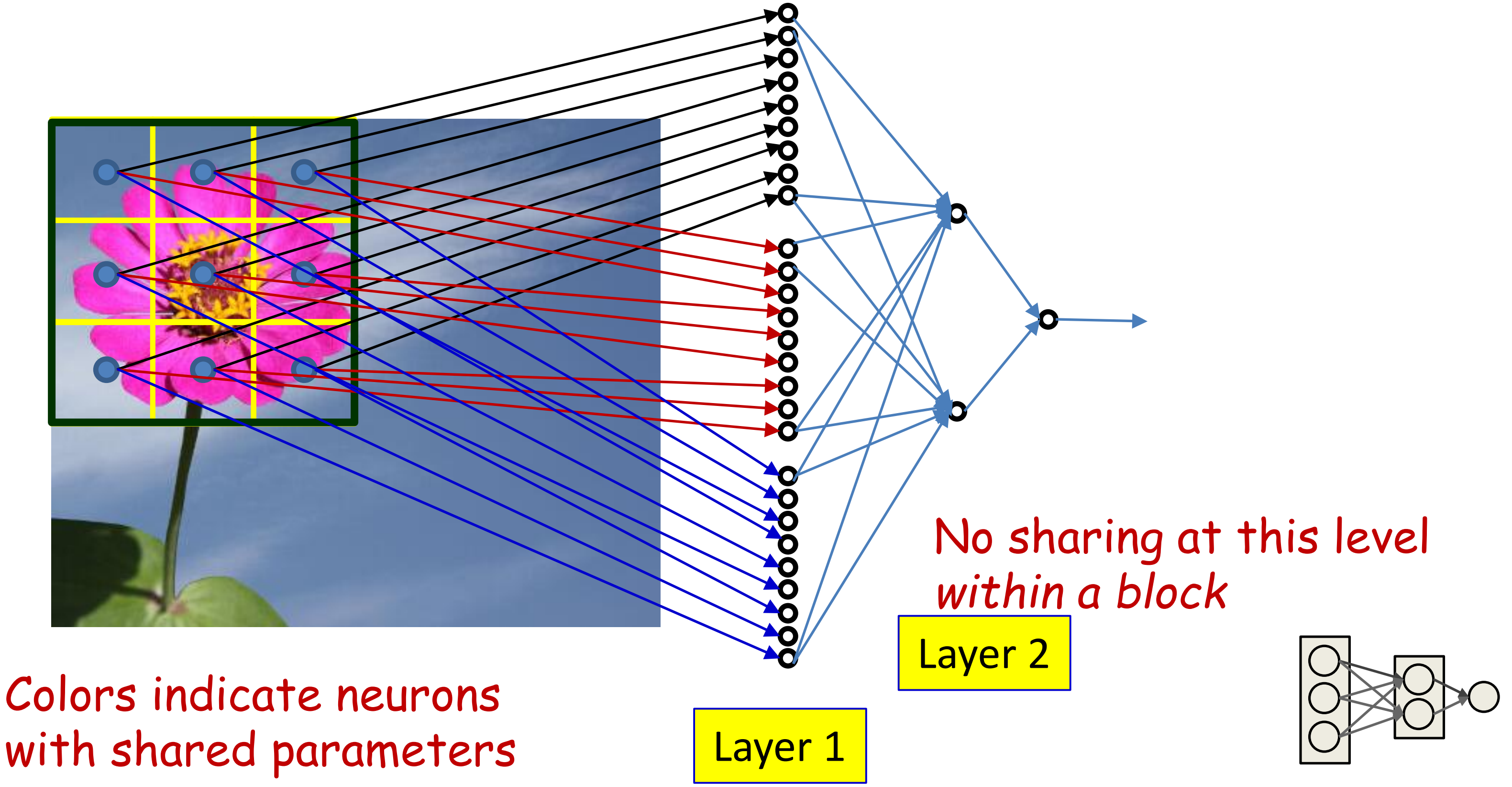
Colors indicate neurons with shared parameters

Layer 1



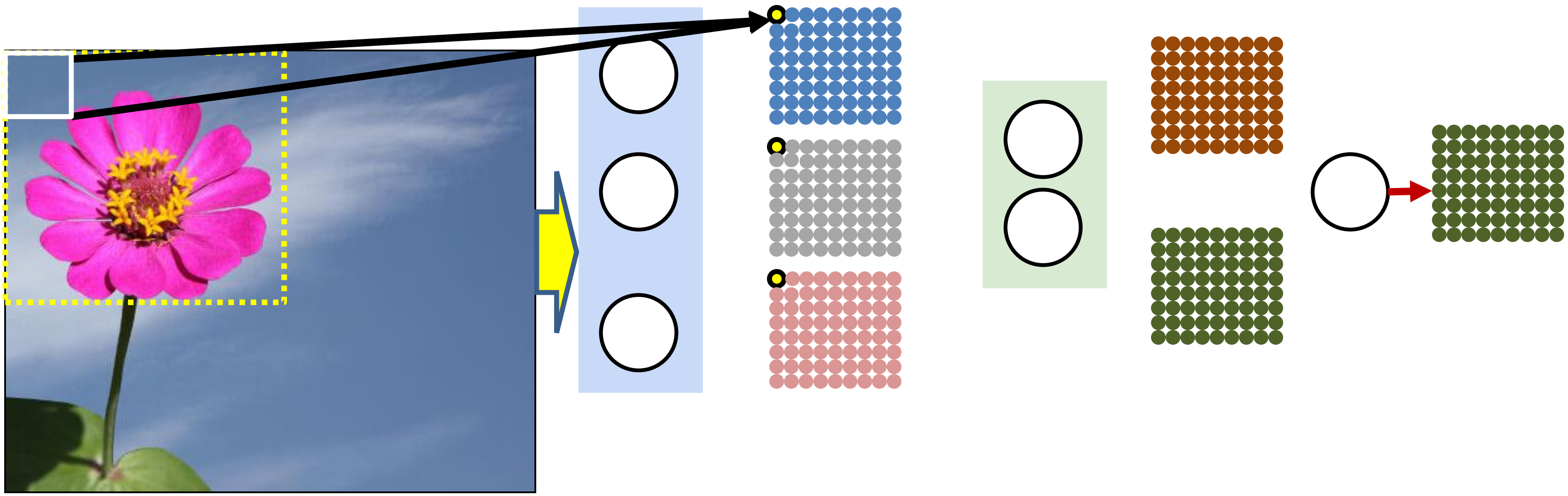
- The network that analyzes individual blocks is now itself a shared parameter network..

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



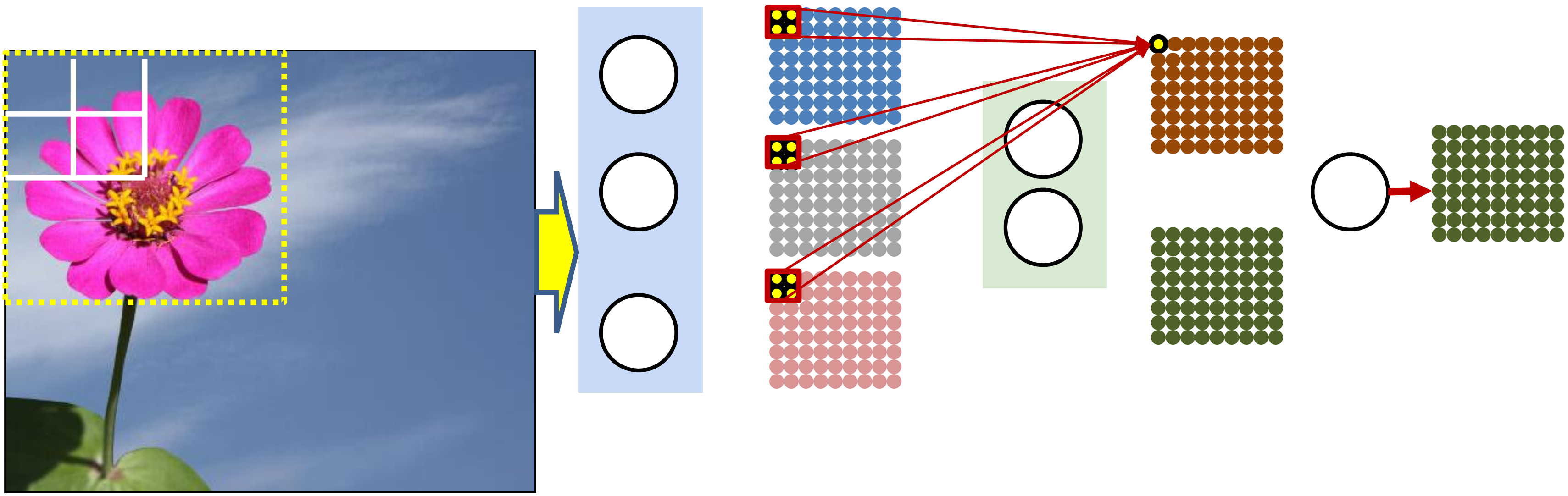
- The network that analyzes individual blocks is now itself a shared parameter network..

This logic can be recursed



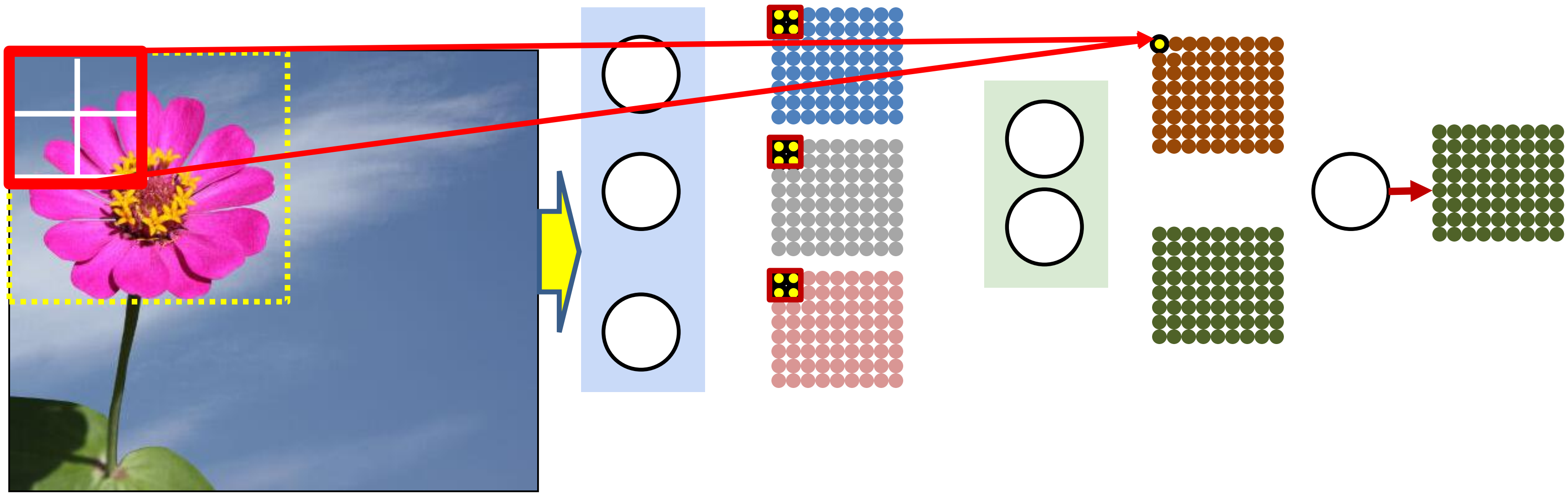
- Building the pattern over 3 layers

This logic can be recursed



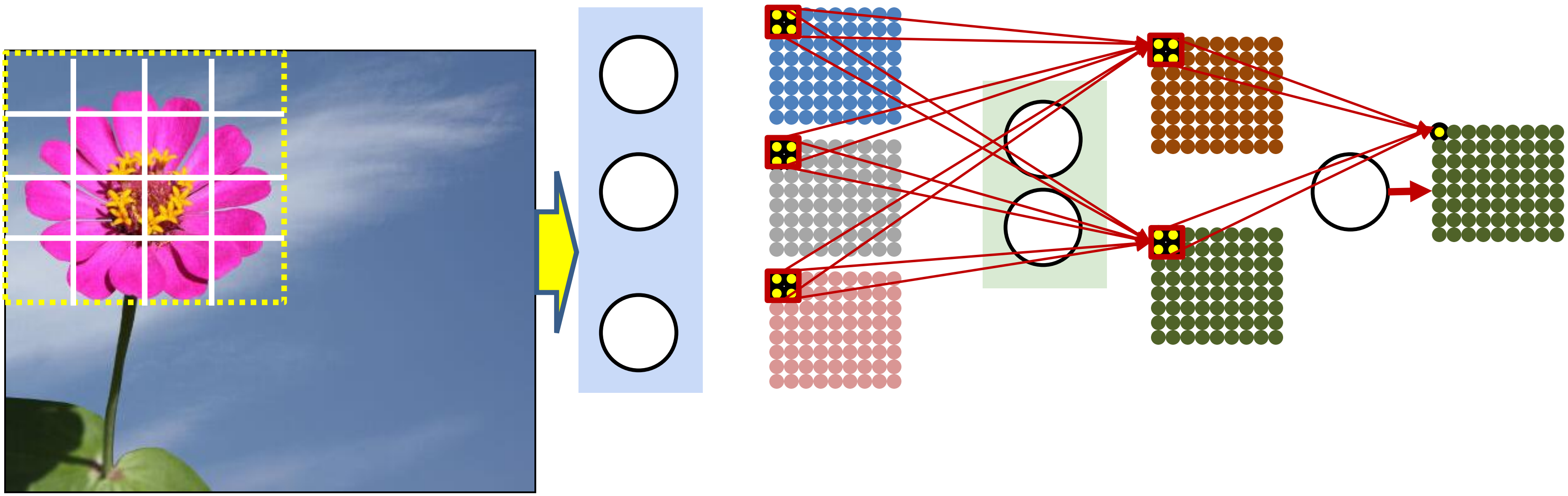
- Building the pattern over 3 layers

This logic can be recursed



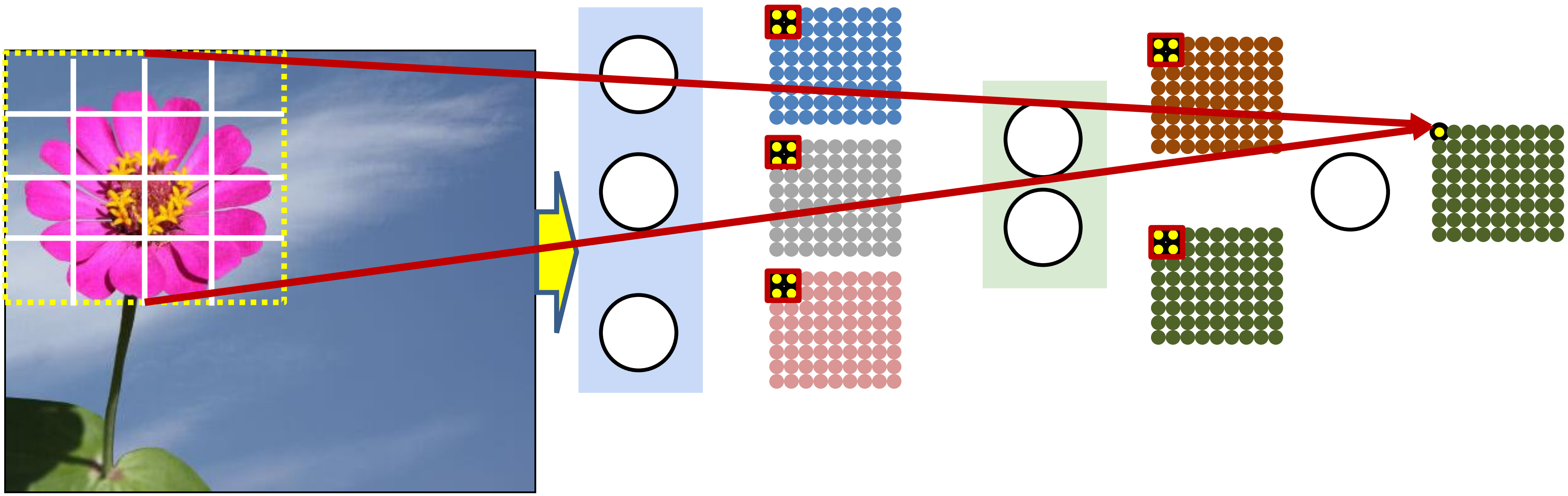
- Building the pattern over 3 layers

This logic can be recursed



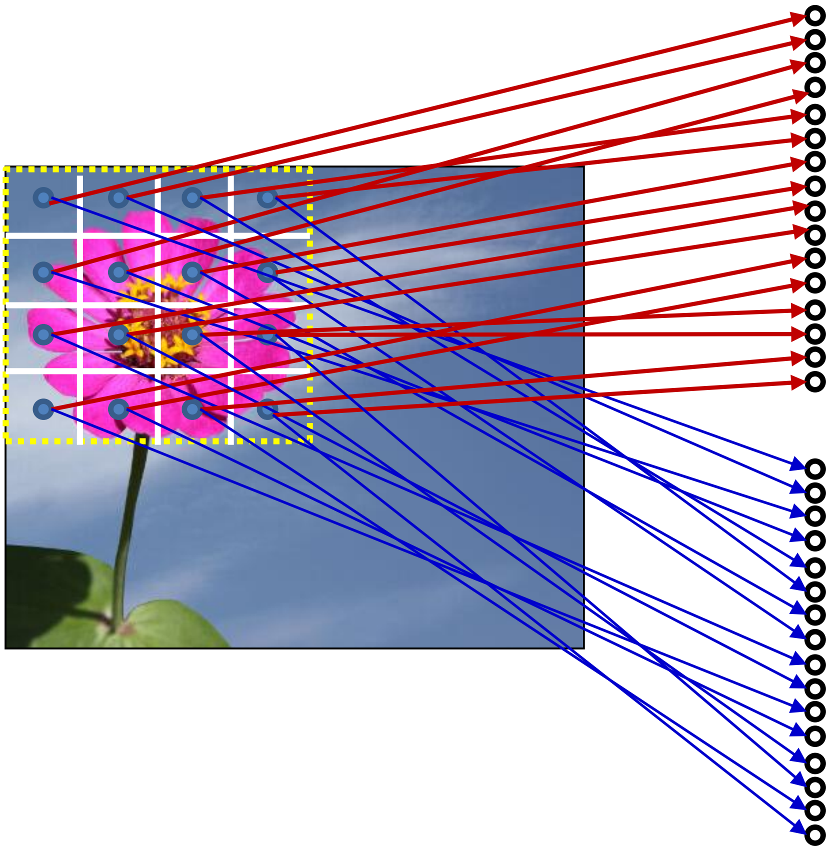
- Building the pattern over 3 layers

This logic can be recursed



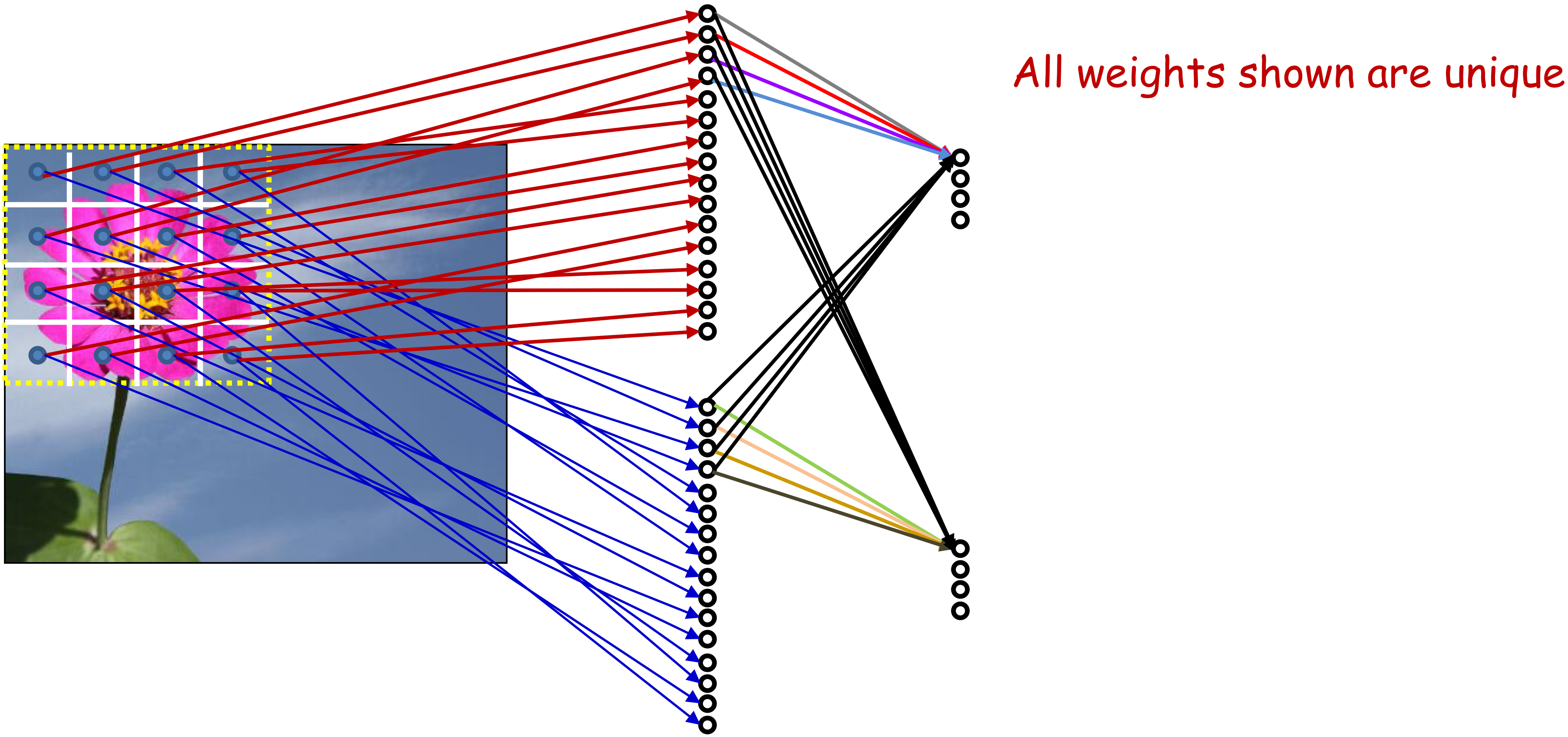
- Building the pattern over 3 layers

The 3-layer shared parameter net



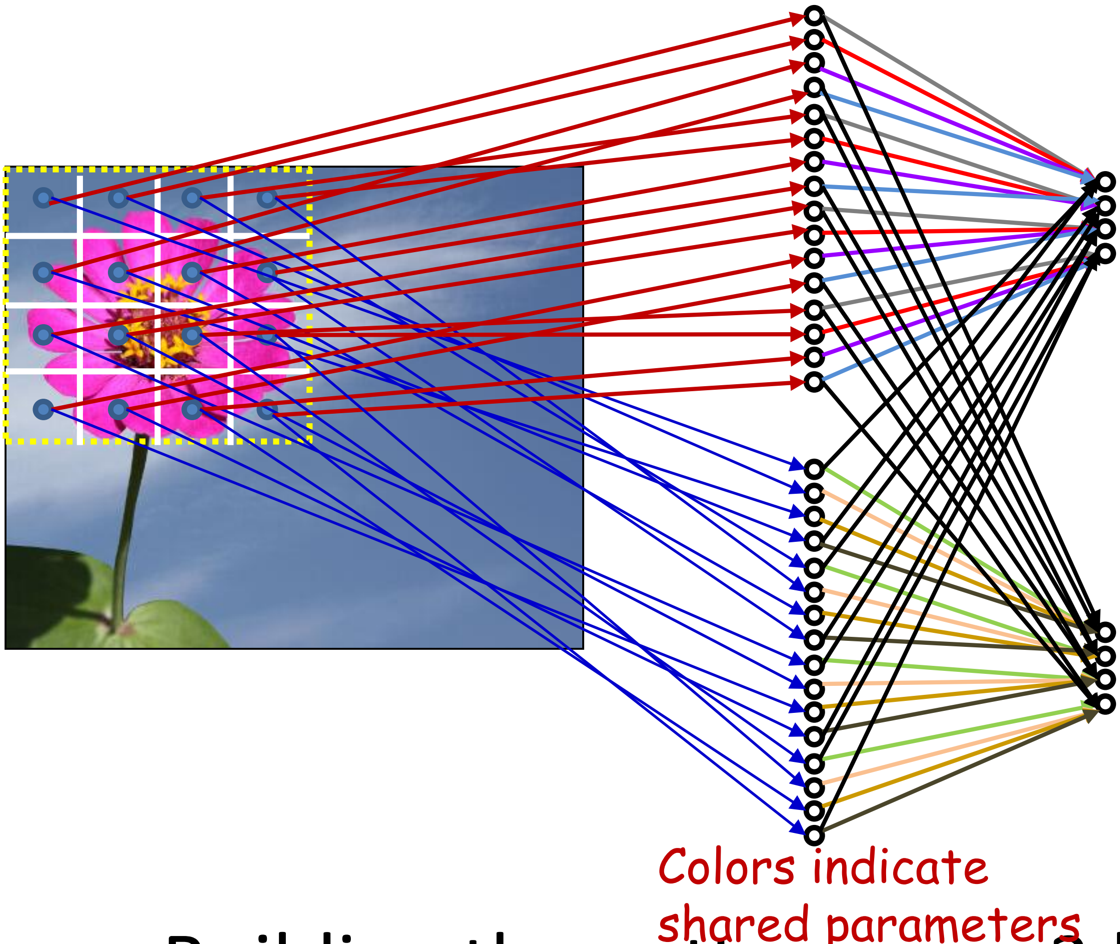
- Building the pattern over 3 layers

The 3-layer shared parameter net



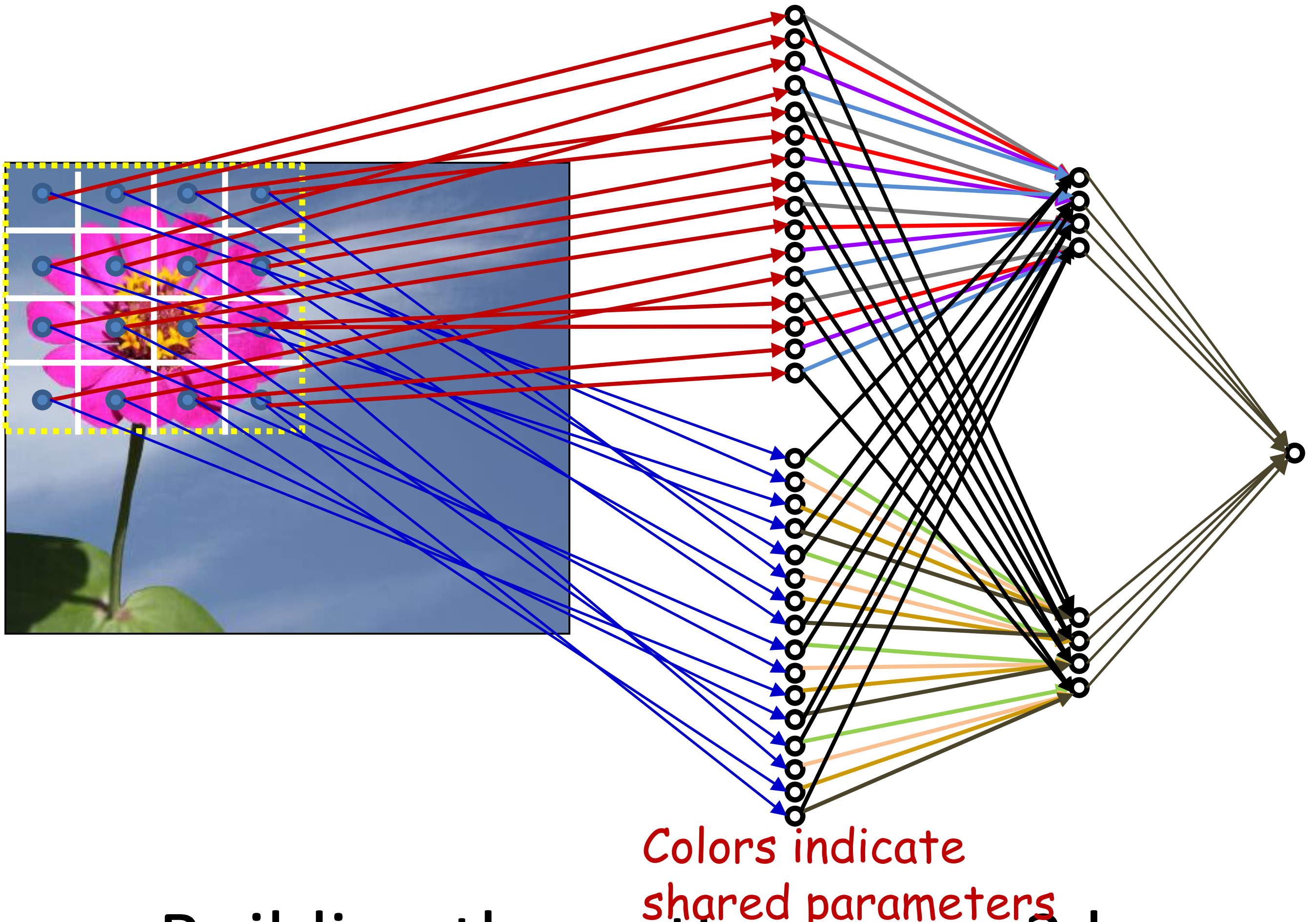
- Building the pattern over 3 layers

The 3-layer shared parameter net



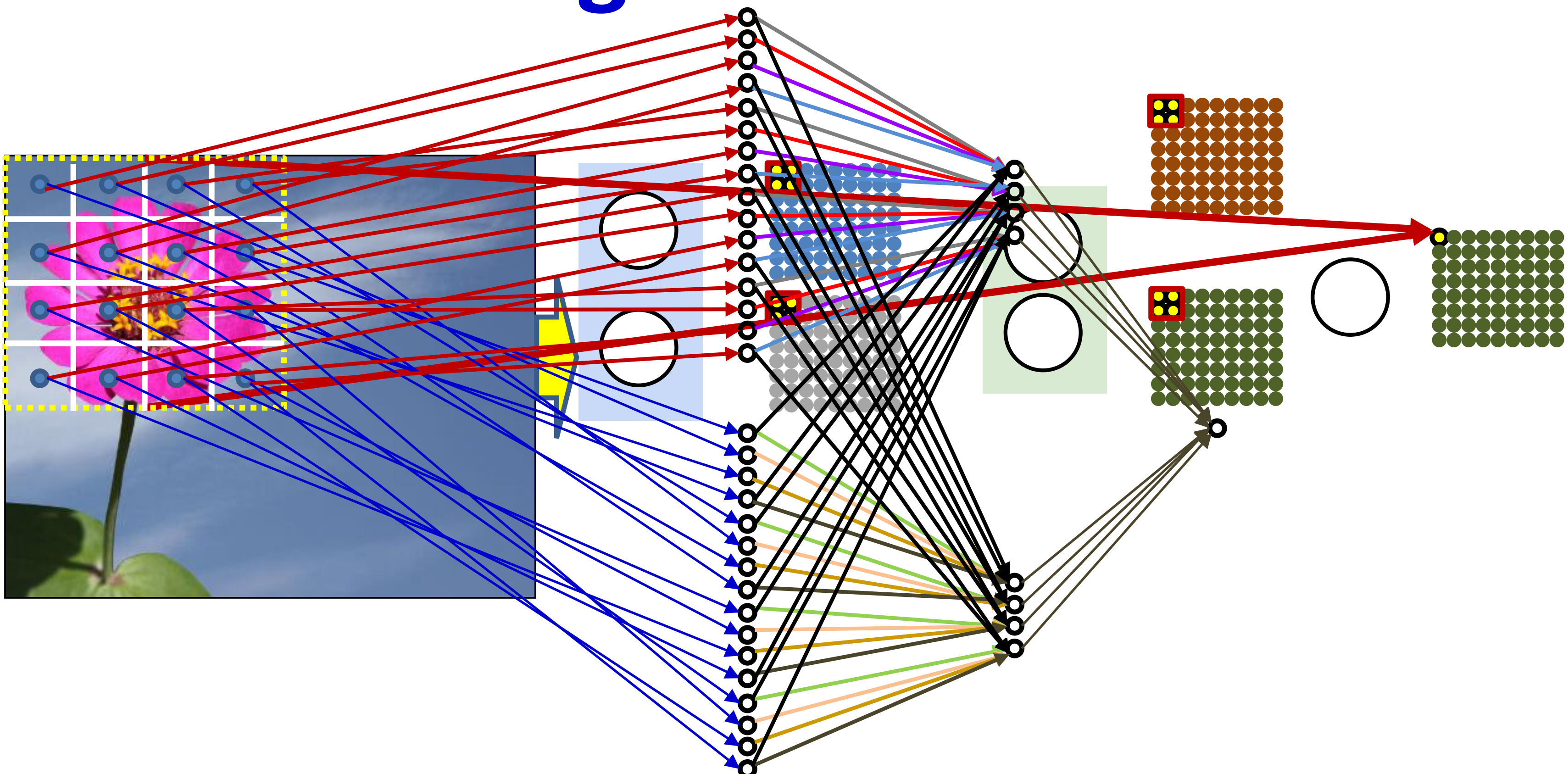
- Building the pattern over 3 layers

The 3-layer shared parameter net



- Building the pattern over 3 layers

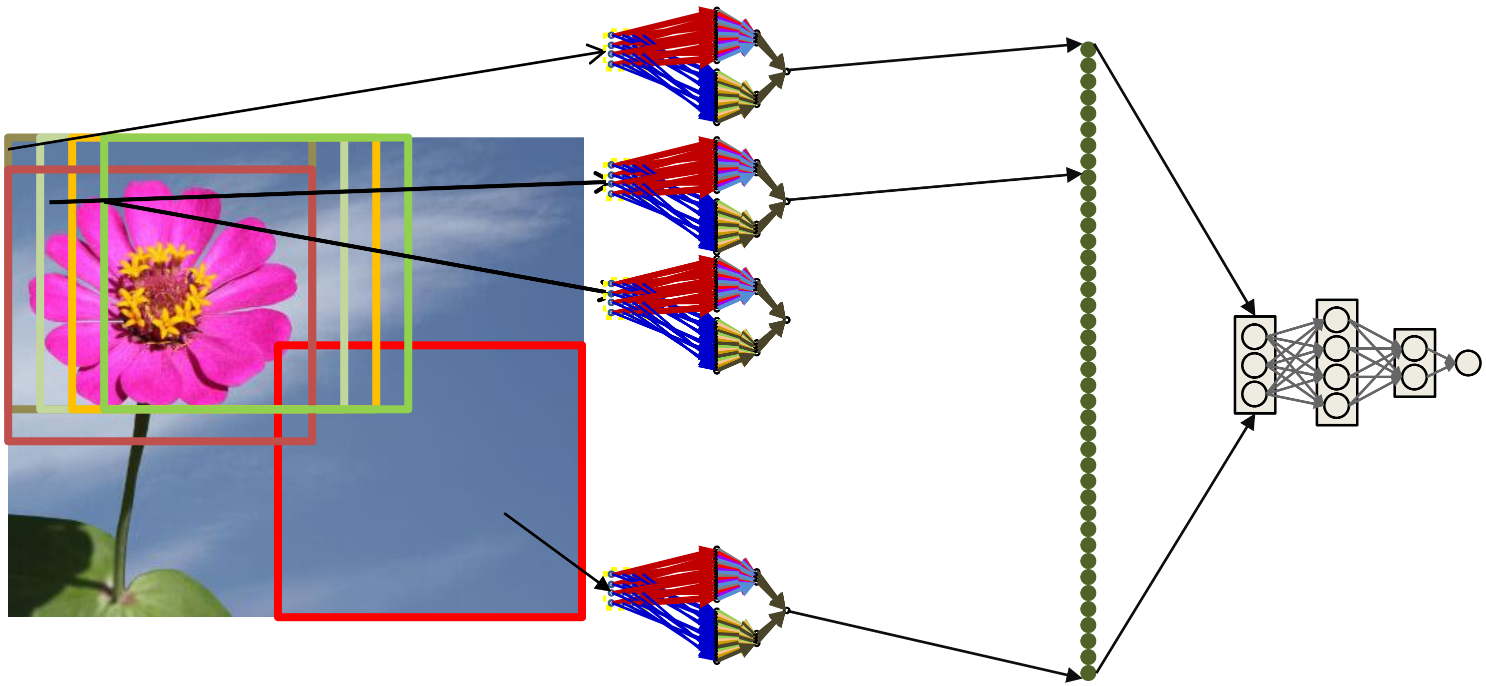
This logic can be recursed



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

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

Using hierarchical build-up of features



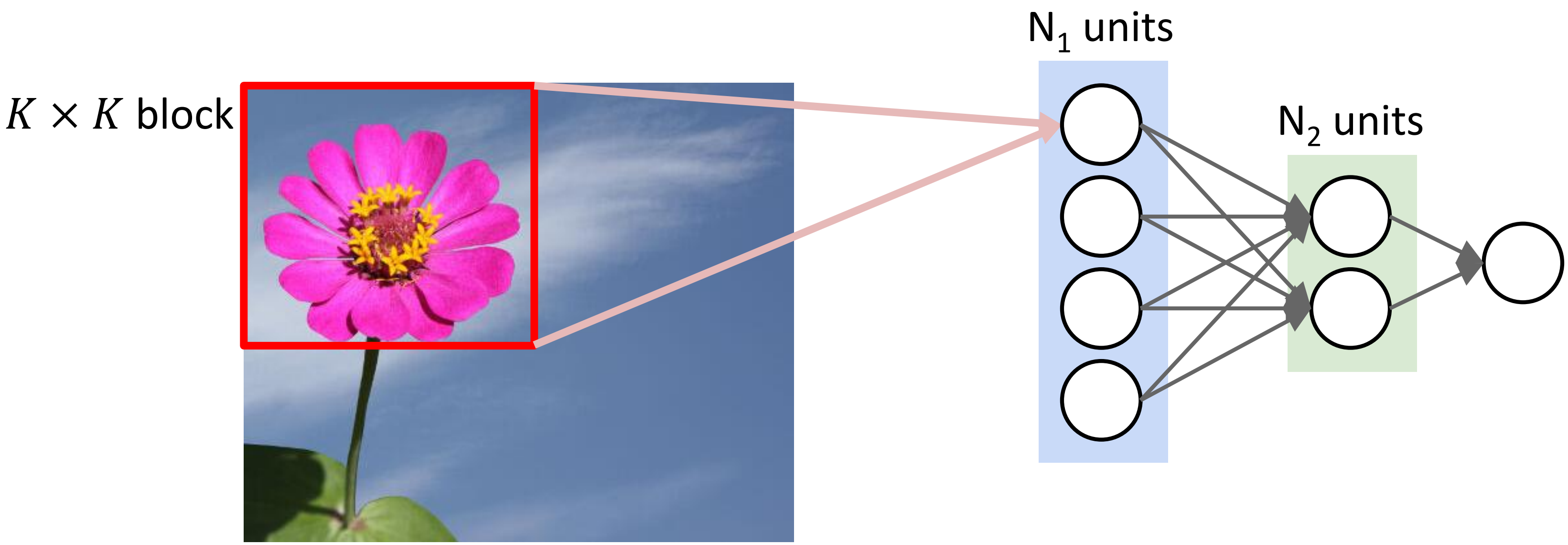
- We scan the figure using the shared parameter network
- The entire operation can be viewed as a single giant network
  - Where individual subnets are themselves shared-parameter nets

Why distribute?

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

- Number of parameters…

Parameters in *Undistributed* network

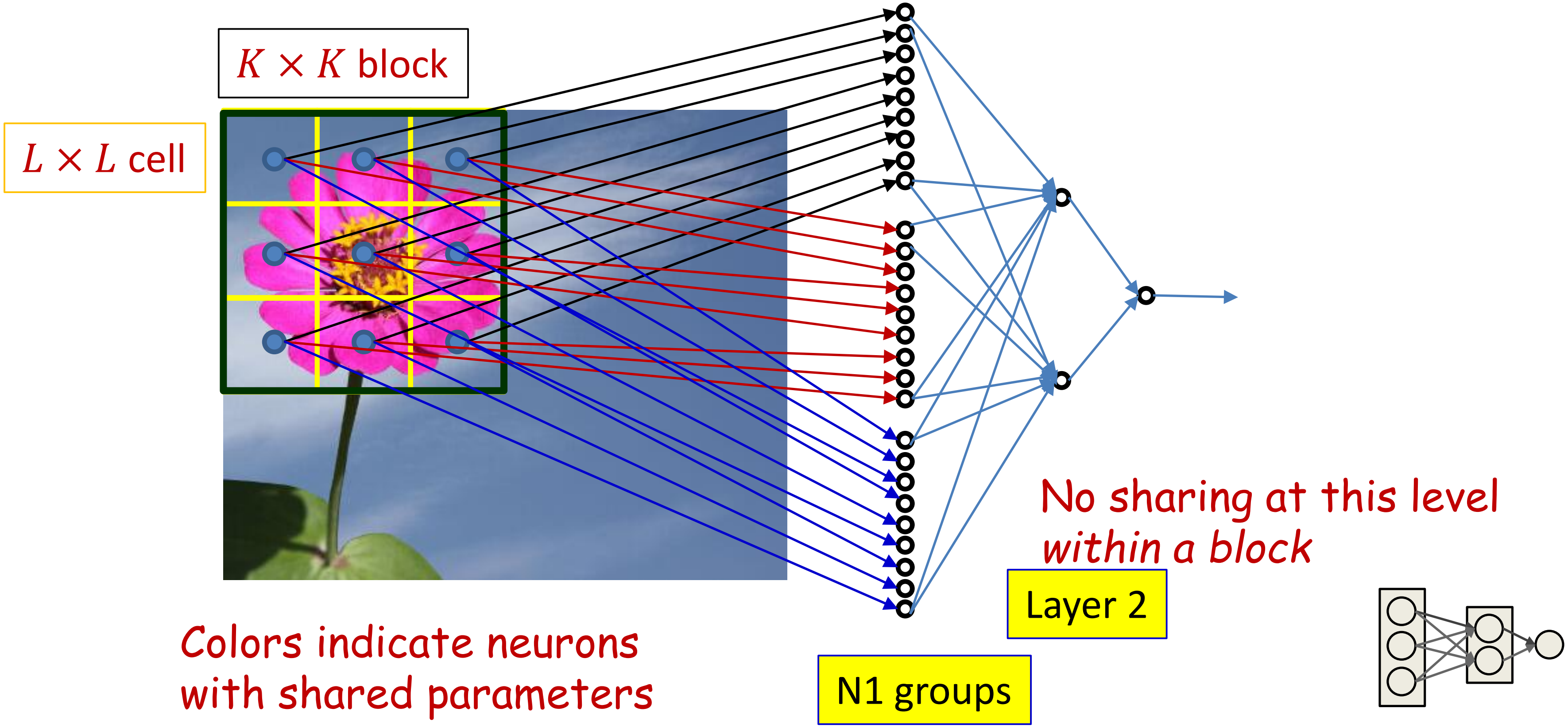


- Only need to consider what happens in *one* block
  - All other blocks are scanned by the same net

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

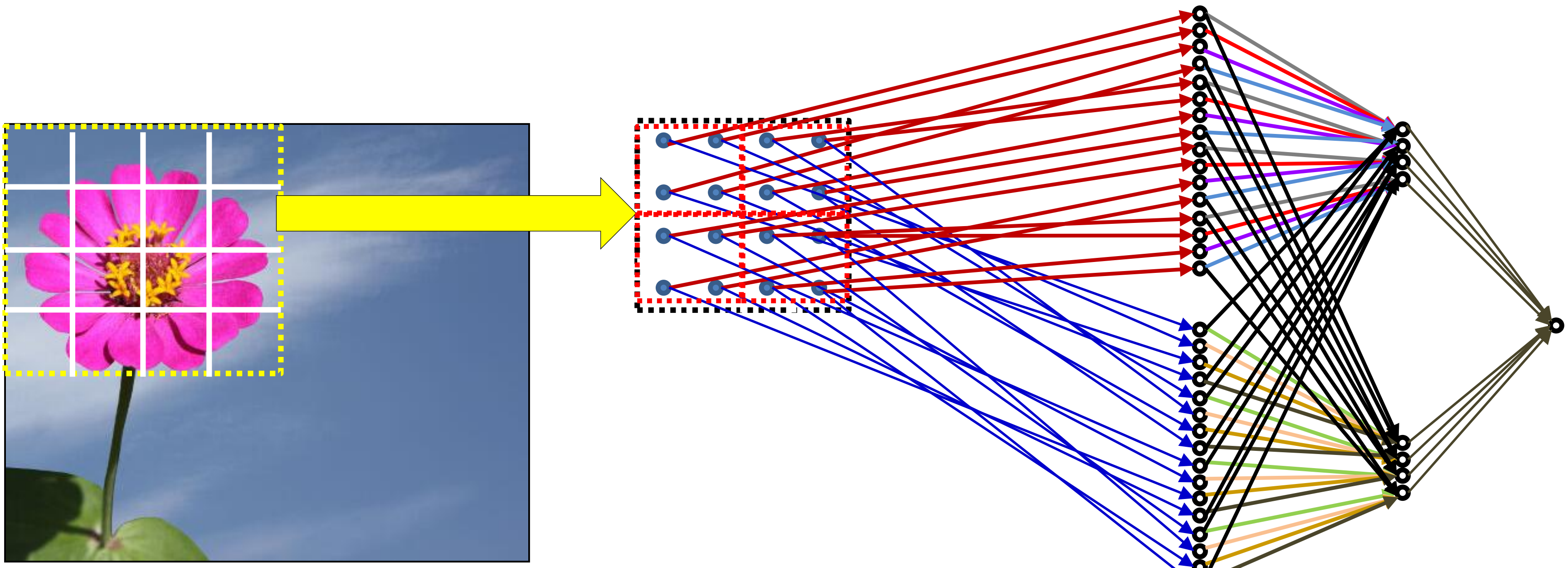
- Total parameters: $\mathcal{O}(K^2 N_1 + N_1 N_2 + N_2 N_3 \dots)$
  - Ignoring the bias term

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: $\mathcal{O}\left(L^2 N_1 + \left(\frac{K}{L}\right)^2 N_1 N_2 + N_2 N_3 \dots\right)$

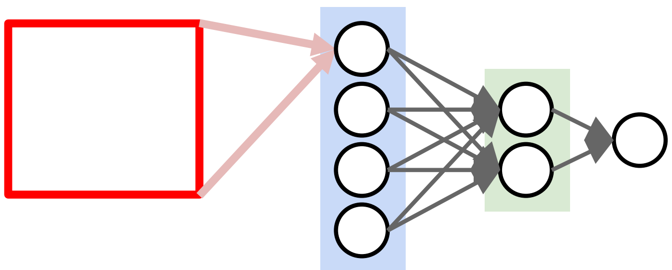
When distributed over 3 layers



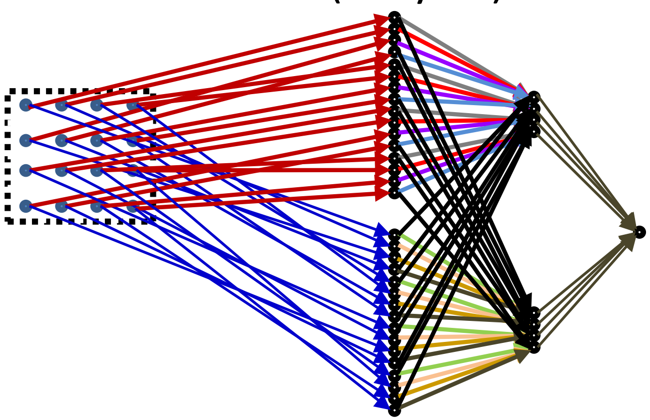
- First layer: N_1 lower-level (groups of) units, each looks at L_1^2 pixels
  - $N_1(L_1^2 + 1)$ weights
- Second layer: N_2 (groups of) units looking at groups of $L_2 \times L_2$ connections from each of N_1 first-level neurons
  - $(L_2^2 N_1 + 1)N_2$ weights
- Third layer:
  - $(\left(\frac{K}{L_1 L_2}\right)^2 N_2 + 1)N_3$ 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)

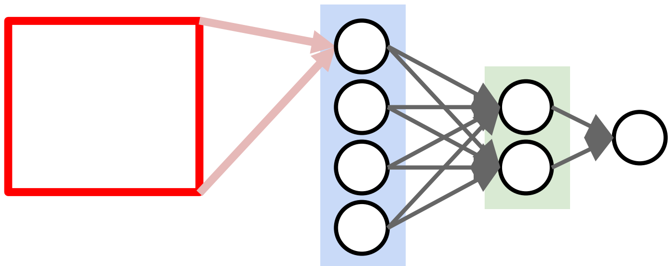


- $\mathcal{O}(K^2N_1 + N_1N_2 + N_2N_3 \ldots)$
- For this example, let $K = 16, N_1 = 4, N_2 = 2, N_3 = 1$
- Total 1034 weights

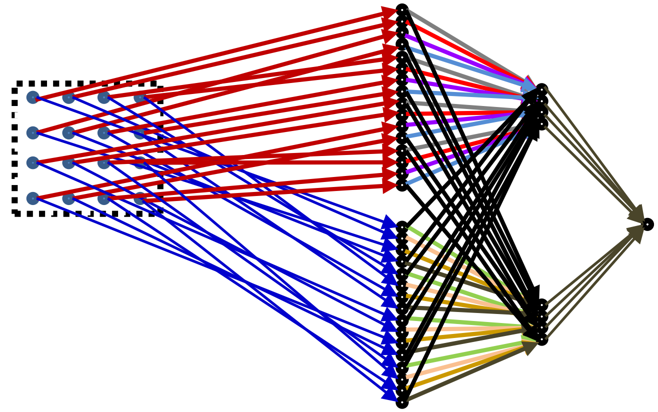
- $\mathcal{O}\Big(L_1^2N_1 + L_2^2N_1N_2 +$

Comparing Number of Parameters

Conventional MLP, not distributed



Distributed (3 layers)



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

- $\mathcal{O}\Bigg(L_1^2 N_1 +$

  $\sum_{i<nconv-1} L_i^2 N_i N_{i+1} +$

  $\left(\frac{K}{\prod_i hop_i}\right)^2 N_{nconv-1} N_{nconv} + $

These terms dominate..

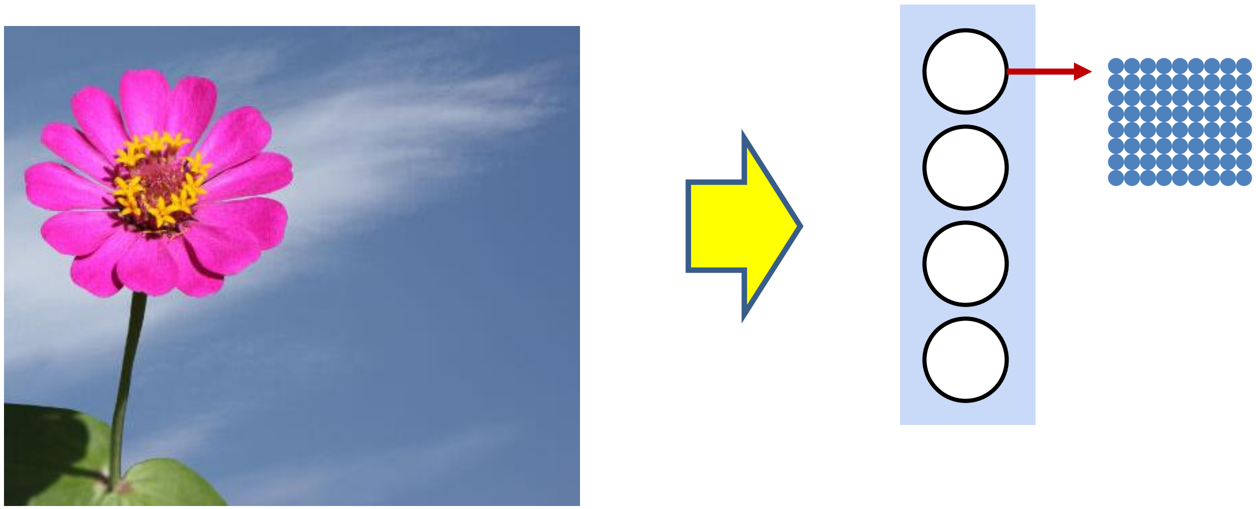
Why distribute?

- 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

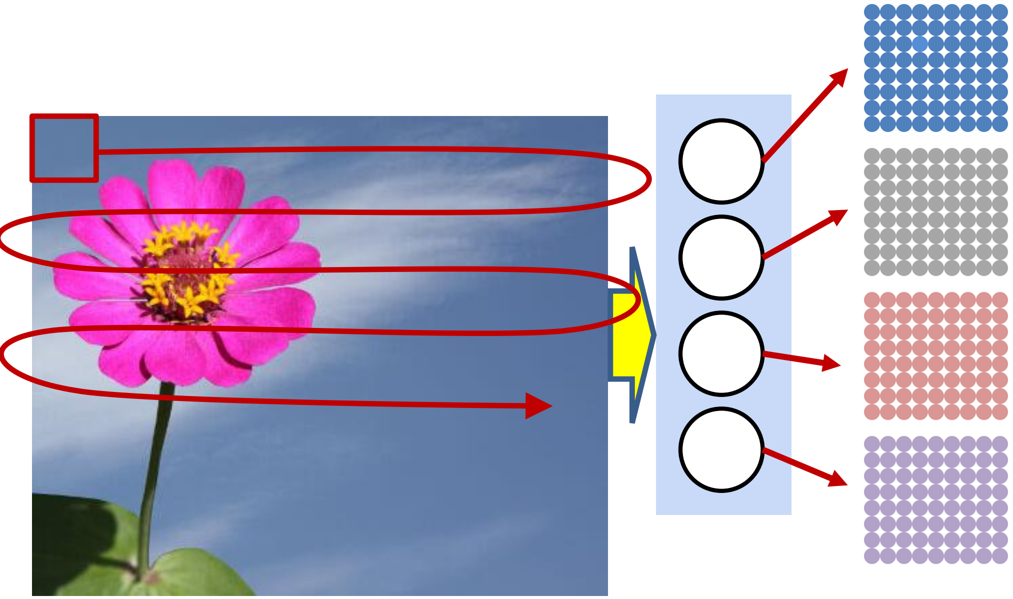
- Key intuition: Regardless of the distribution, we can view the network as "scanning" the picture with an MLP
  - The only difference is the manner in which parameters are shared in the MLP

Hierarchical composition: A different perspective



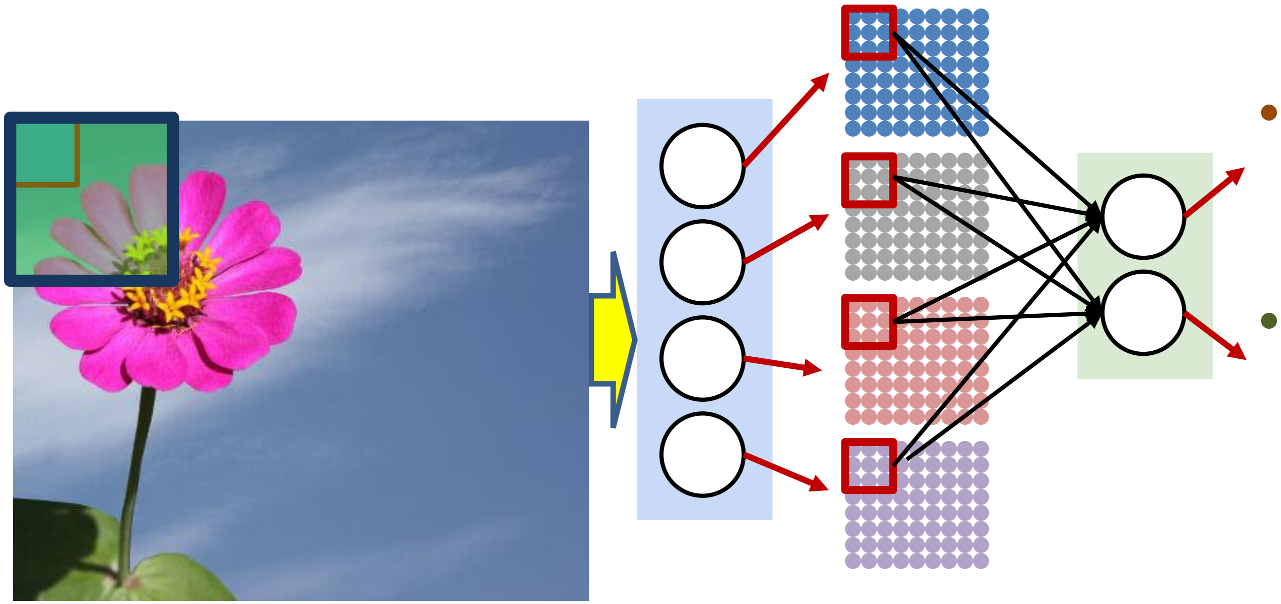
- The entire operation can be redrawn as before as maps of the entire image

Building up patterns



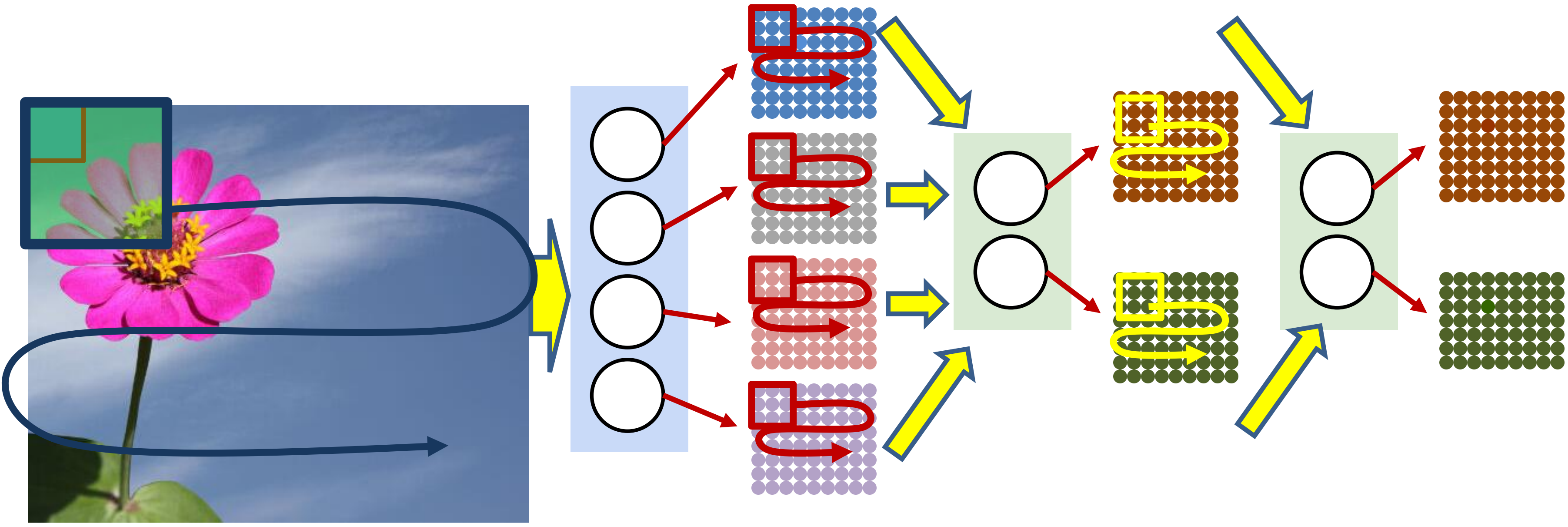
- The first layer looks at small *sub* regions of the main image
  - Sufficient to detect, say, petals

Some modifications



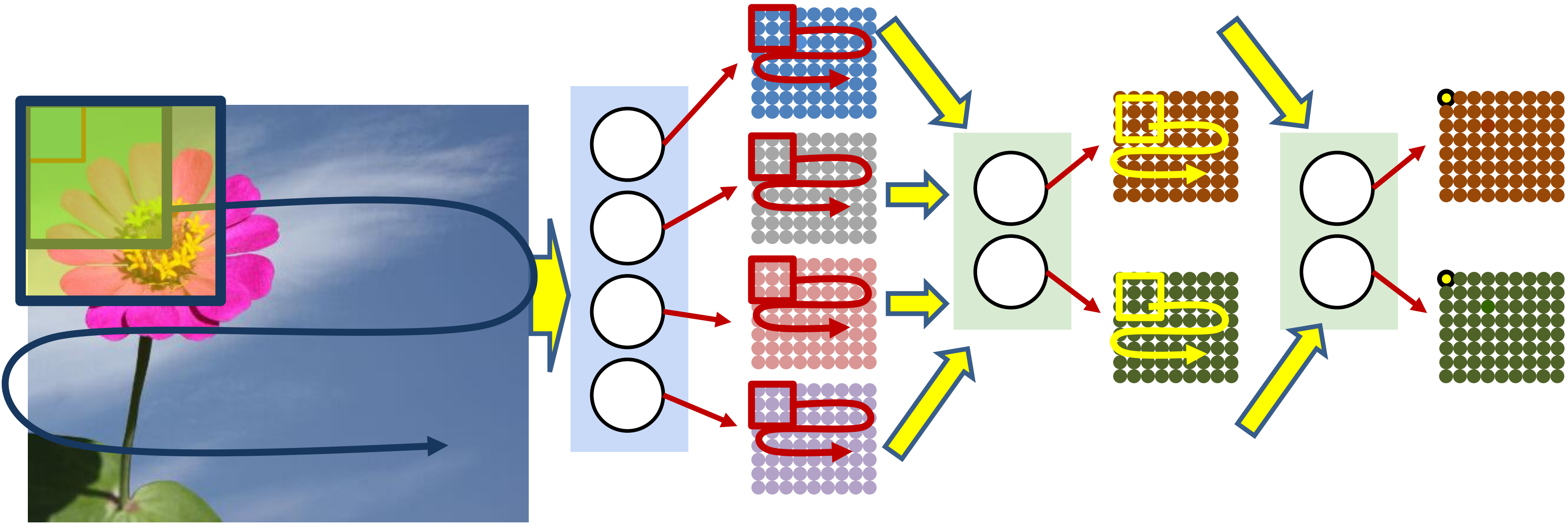
- The first layer looks at *sub* regions of the main image
  - Sufficient to detect, say, petals
- The second layer looks at *regions* of the output of the first layer
  - To put the petals together into a flower
  - This corresponds to looking at a larger region of the original input image

Some modifications



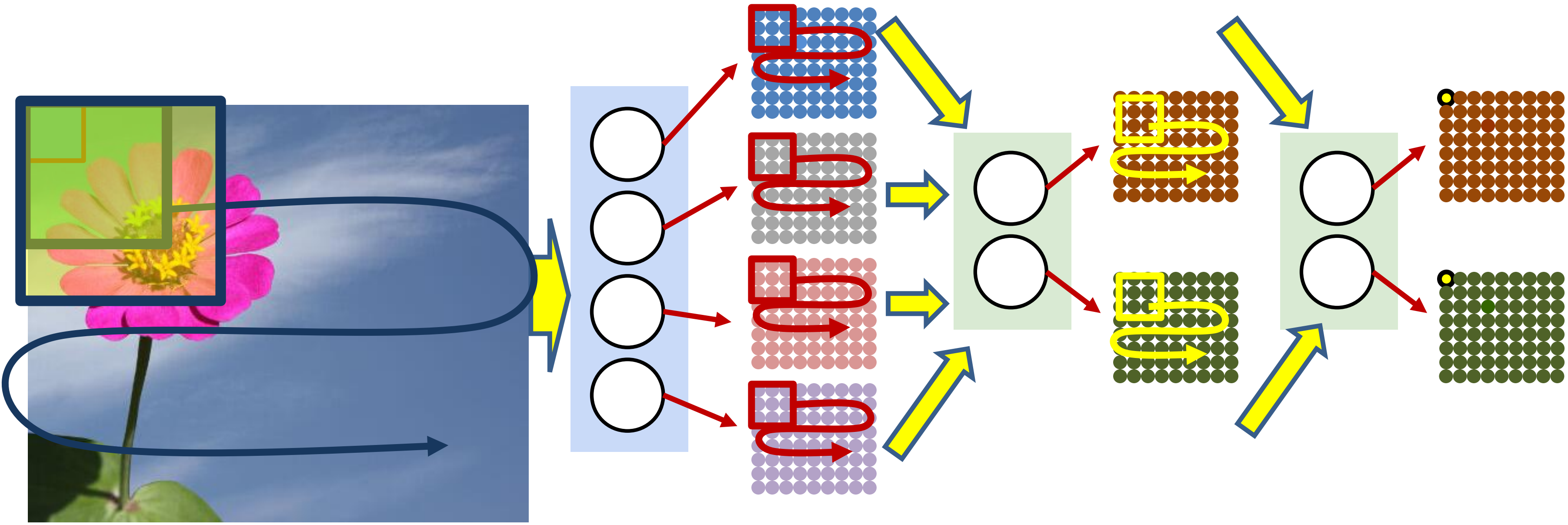
- The first layer looks at *sub* regions of the main image
  - Sufficient to detect, say, petals
- The second layer looks at *regions* of the output of the first layer
  - To put the petals together into a flower
  - This corresponds to looking at a larger region of the original input image
- We may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

Some modifications



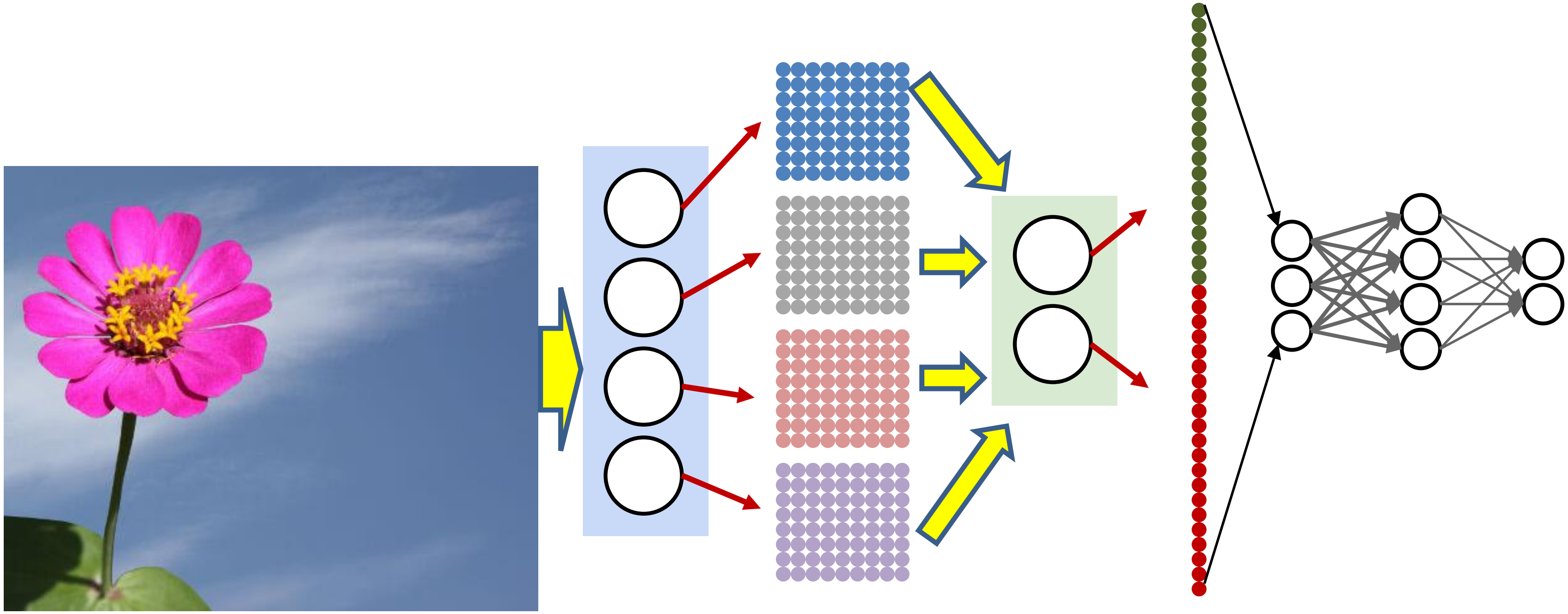
- The first layer looks at *sub* regions of the main image
  - Sufficient to detect, say, petals
- The second layer looks at *regions* of the output of the first layer
  - To put the petals together into a flower
  - This corresponds to looking at a larger region of the original input image
- We may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

Terminology



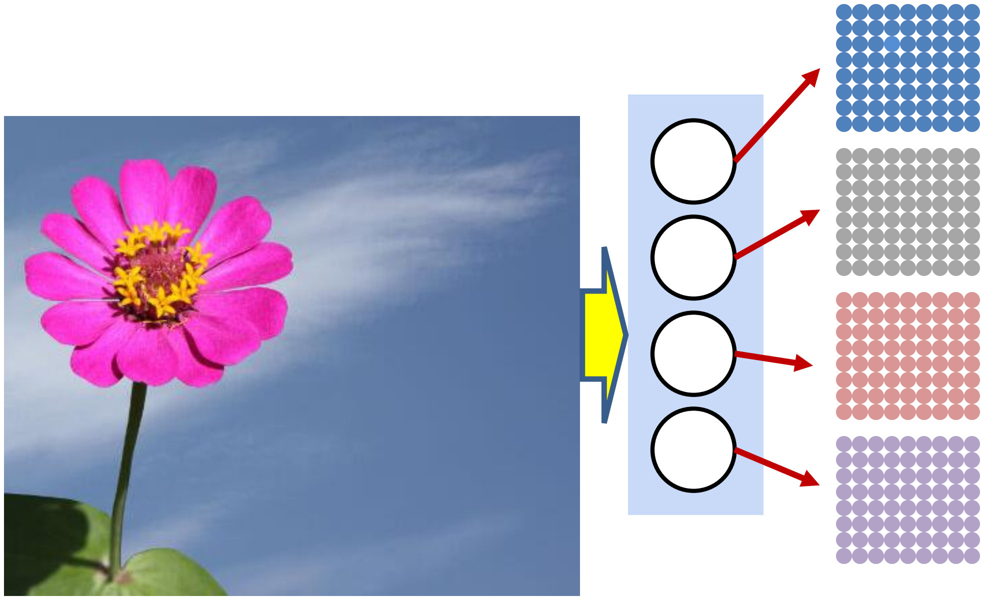
- The pattern in the *input* image that each neuron sees is its “Receptive Field”
  - The squares show the *sizes* of the receptive fields for the first, second and third-layer neurons
- The actual receptive field for a first layer neurons is simply its arrangement of weights
- For the higher level neurons,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

Some modifications



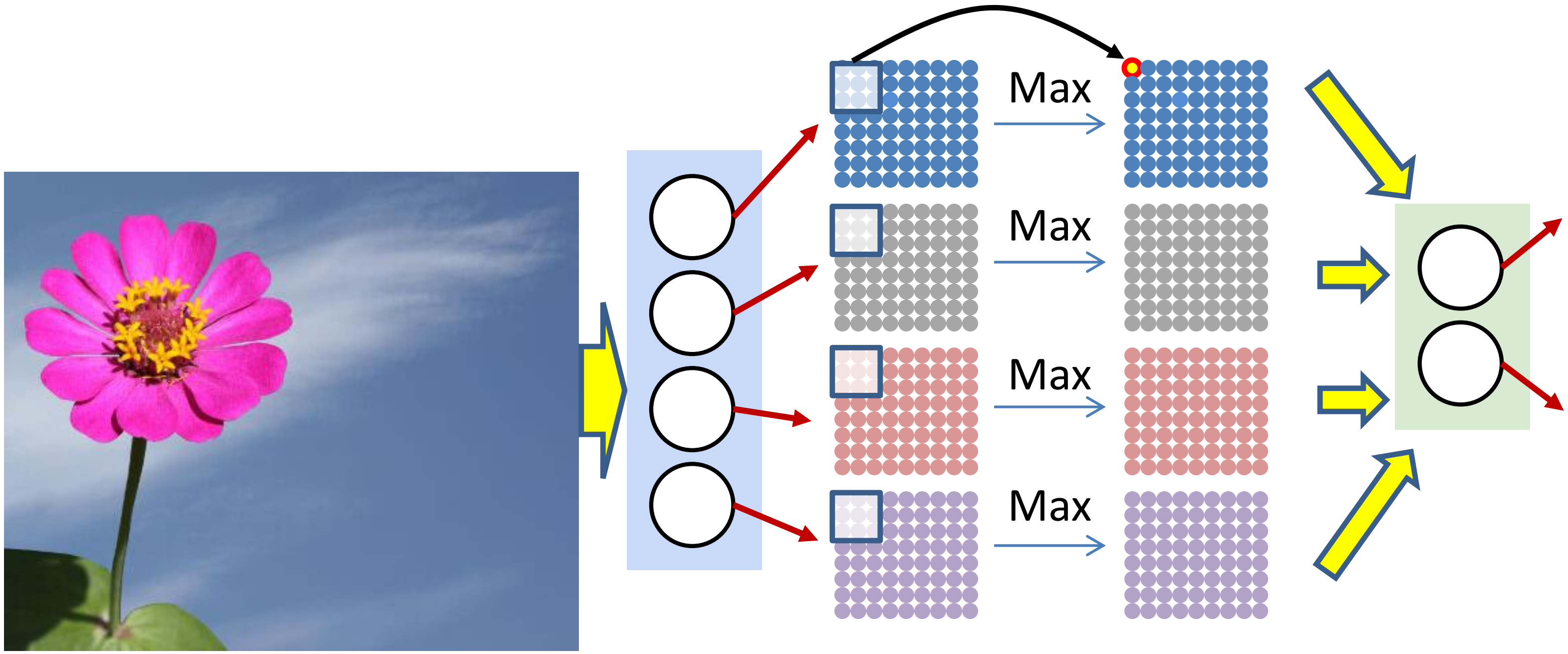
- The final layer may feed directly into a multi layer perceptron rather than a single neuron
- This is exactly the shared parameter net we just saw

Accounting for jitter



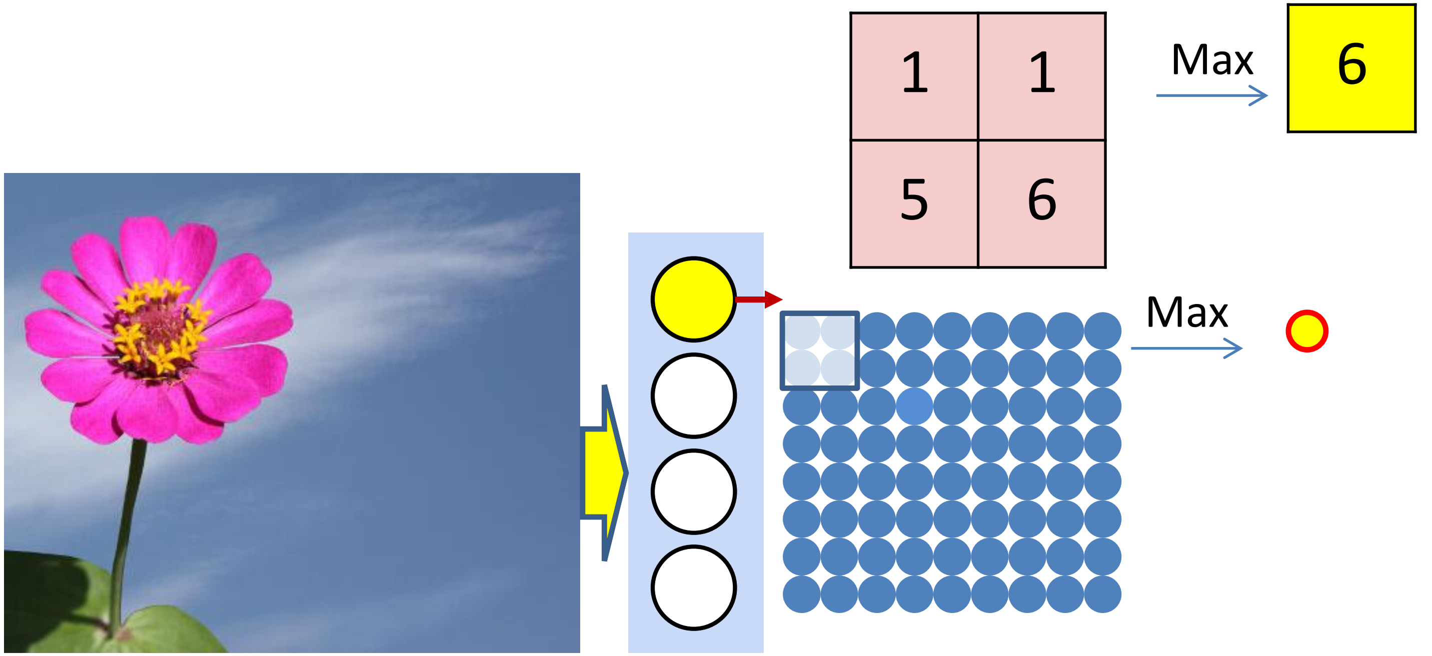
- We would like to account for some jitter in the first-level patterns
  - If a pattern shifts by one pixel, is it still a petal?

Accounting for jitter



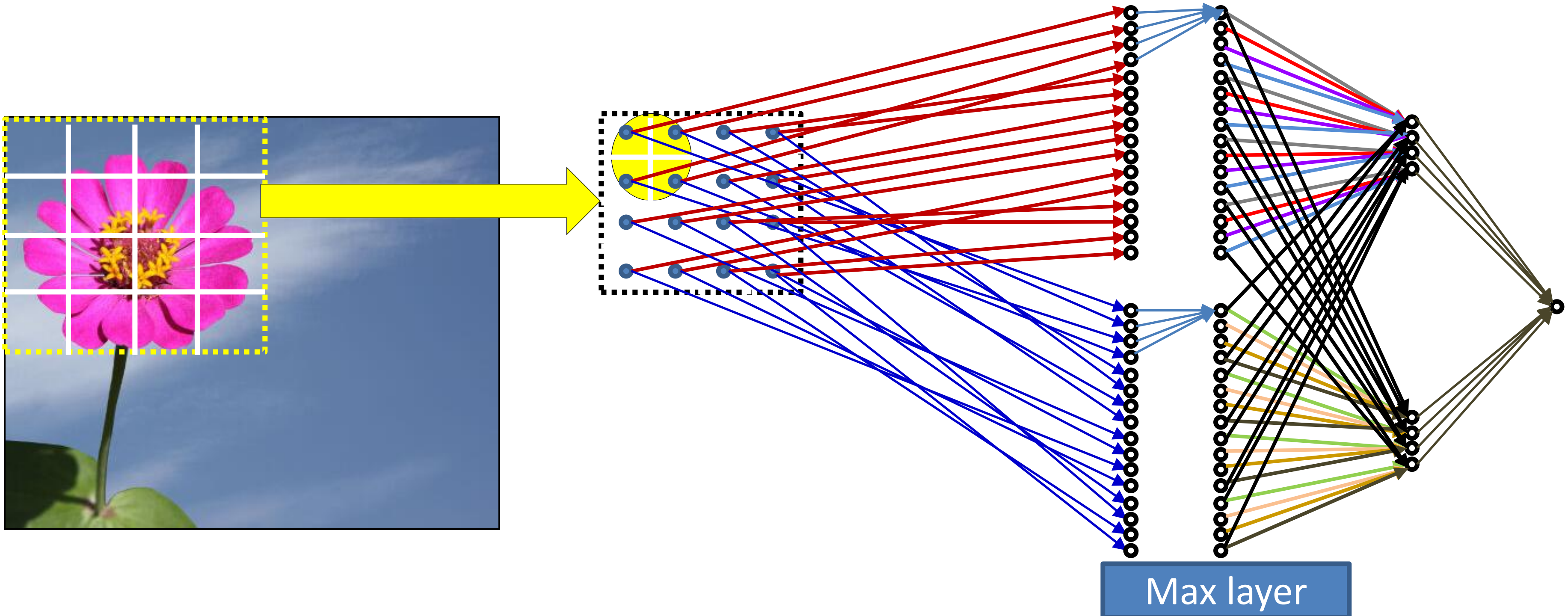
- We would like to account for some jitter in the first-level patterns
  - If a pattern shifts by one pixel, is it still a petal?
  - A small jitter is acceptable
    - Replace each value by the maximum of the values within a small region around it
      - ***Max filtering or Max pooling***

Accounting for jitter



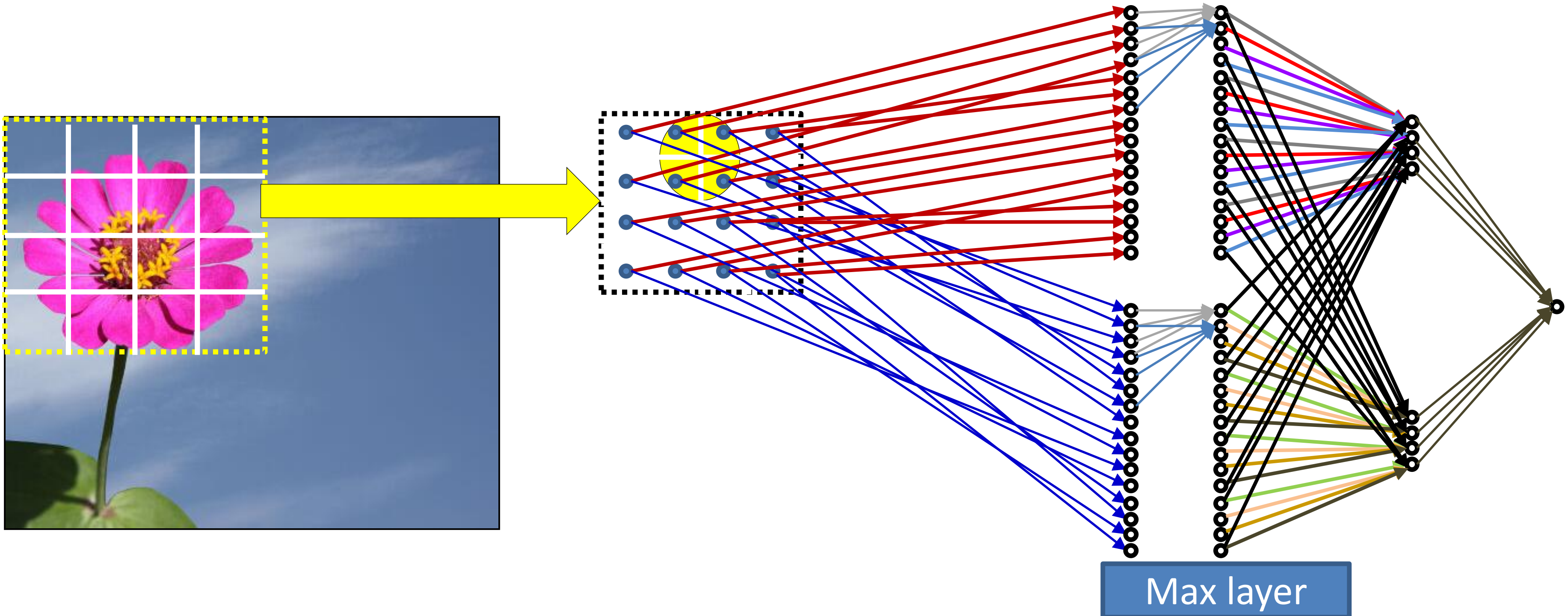
- We would like to account for some jitter in the first-level patterns
  - If a pattern shifts by one pixel, is it still a petal?
  - A small jitter is acceptable
    - Replace each value by the maximum of the values within a small region around it
      - ***Max filtering*** or ***Max pooling***

The max operation is just a neuron



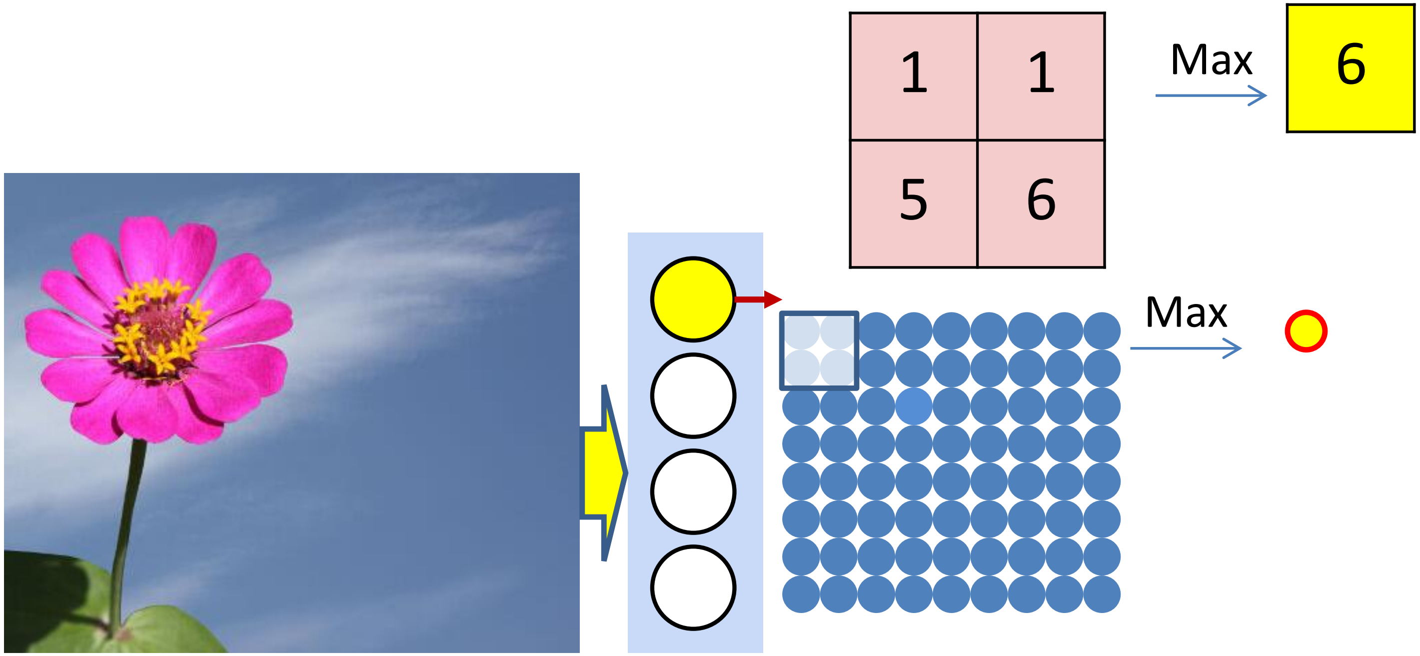
- The max operation is just another neuron
- Instead of applying an activation to the weighted sum of inputs, each neuron just computes the maximum over all inputs

The max operation is just a neuron



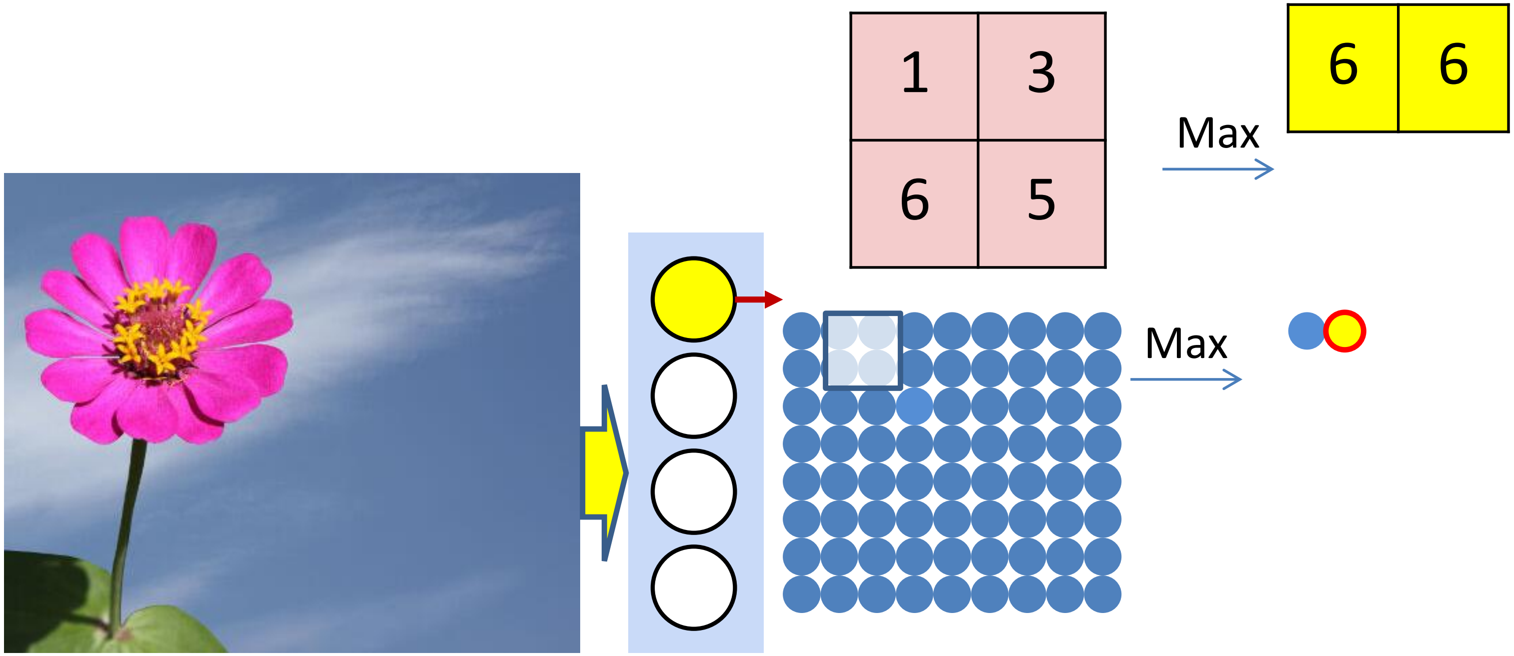
- The max operation is just another neuron
- Instead of applying an activation to the weighted sum of inputs, each neuron just computes the maximum over all inputs

Accounting for jitter



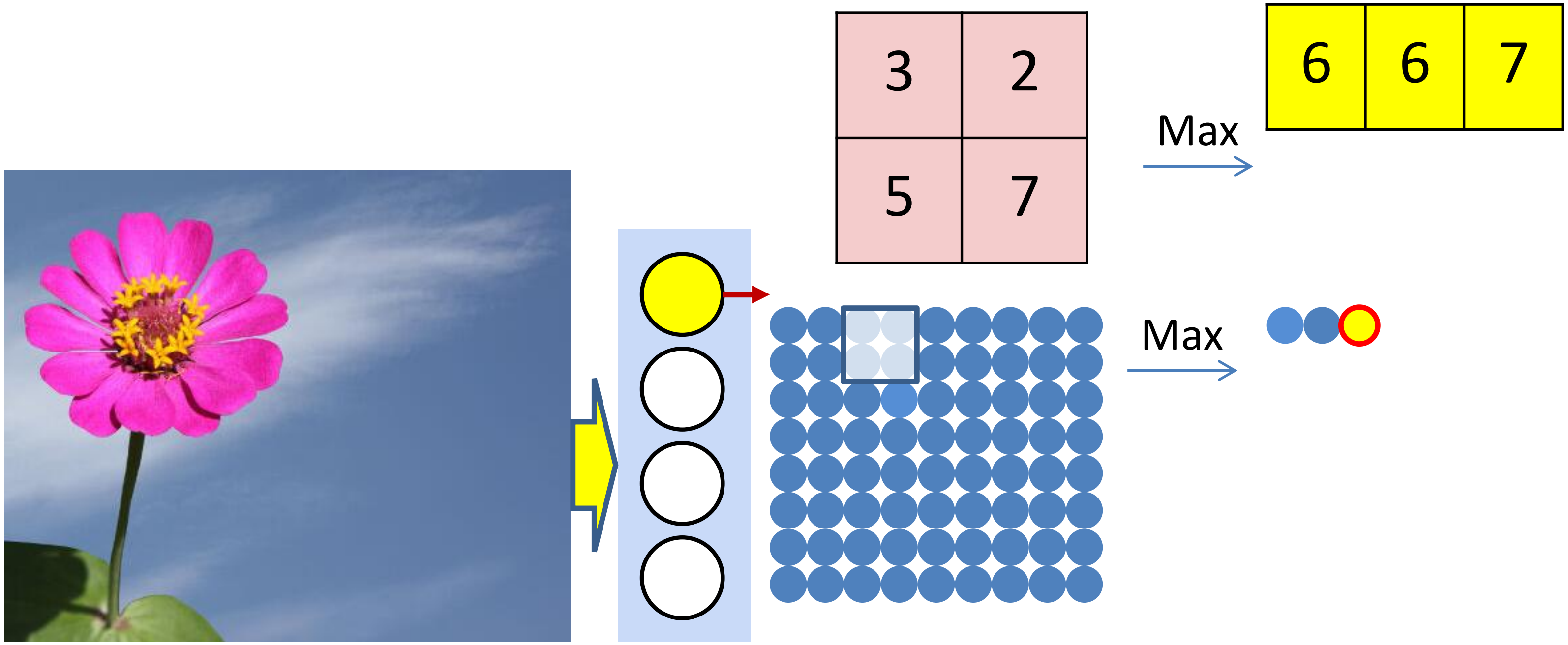
- The max filtering can also be performed as a scan

Accounting for jitter



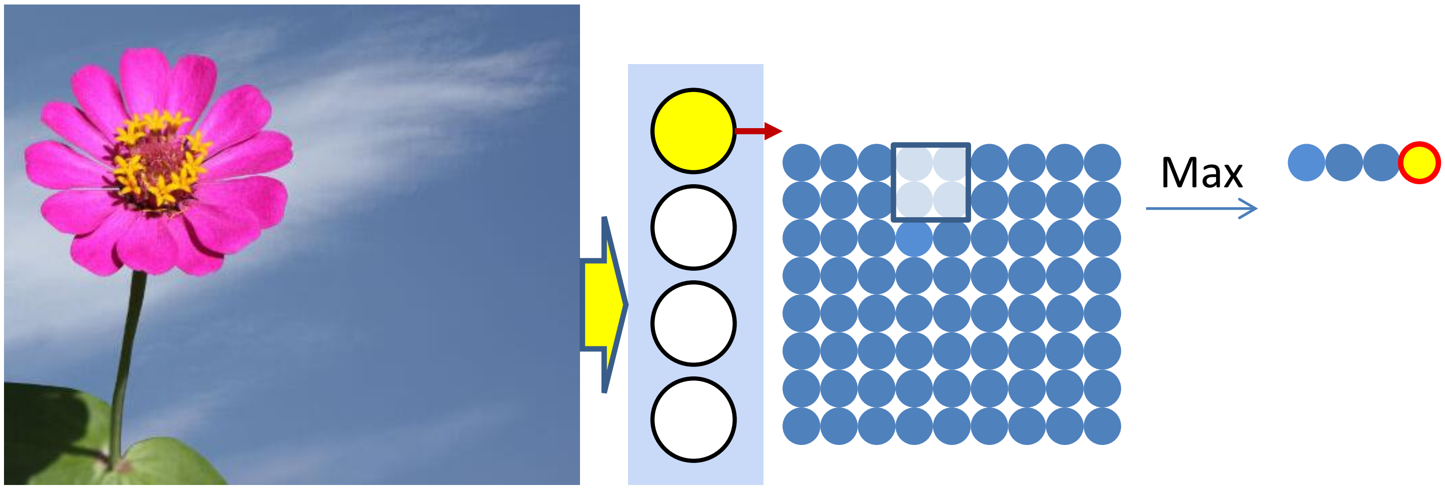
- The “max filter” operation too “scans” the picture

Accounting for jitter



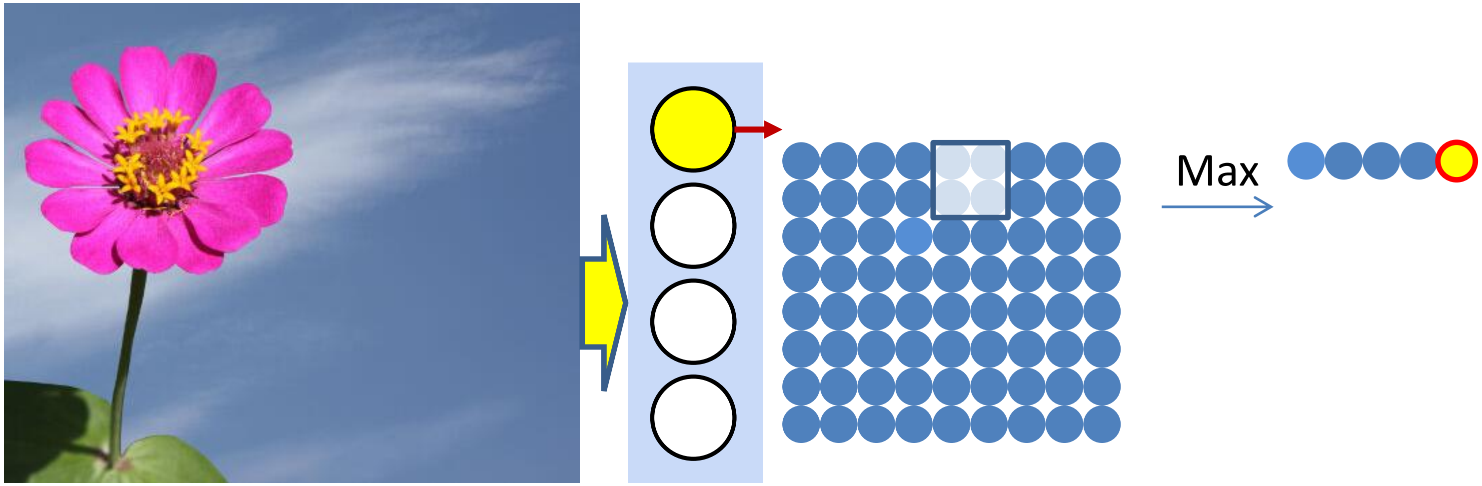
- The “max filter” operation too “scans” the picture

Accounting for jitter



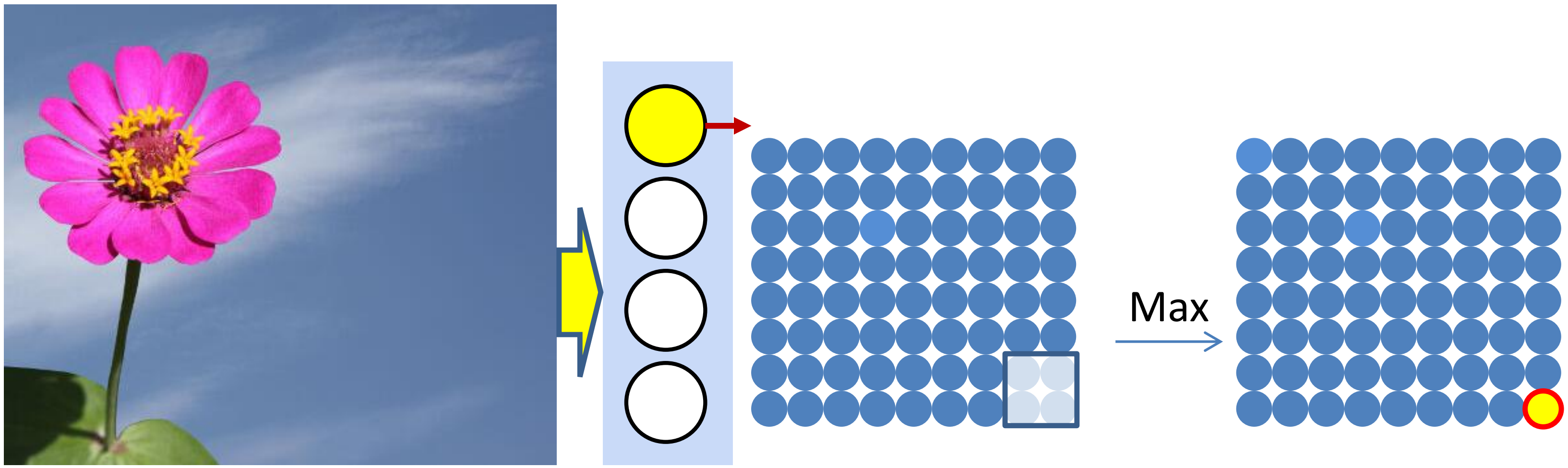
- The “max filter” operation too “scans” the picture

Accounting for jitter



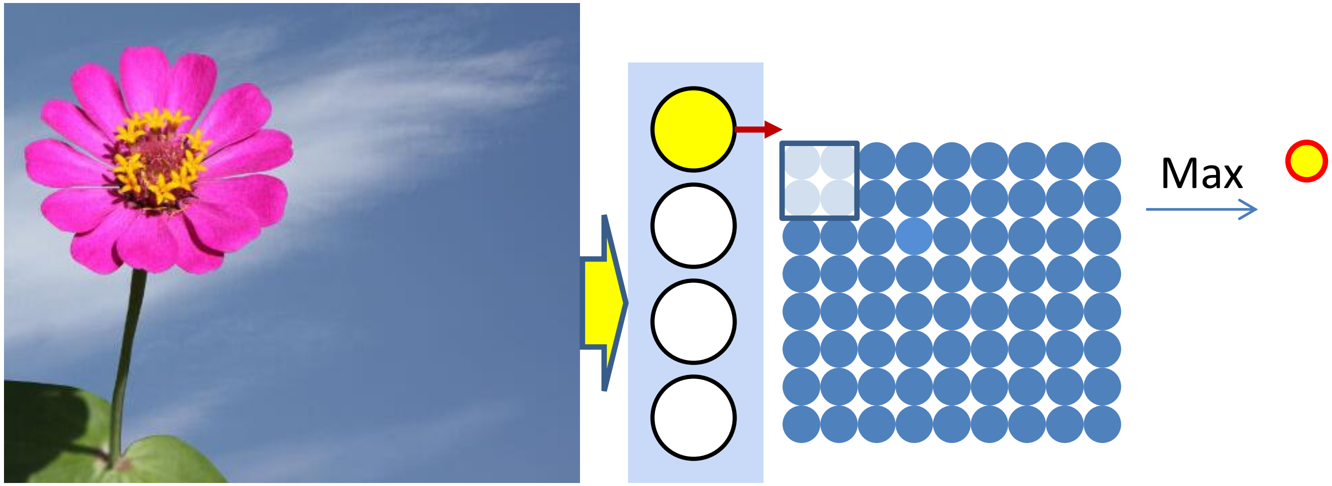
- The “max filter” operation too “scans” the picture

Accounting for jitter



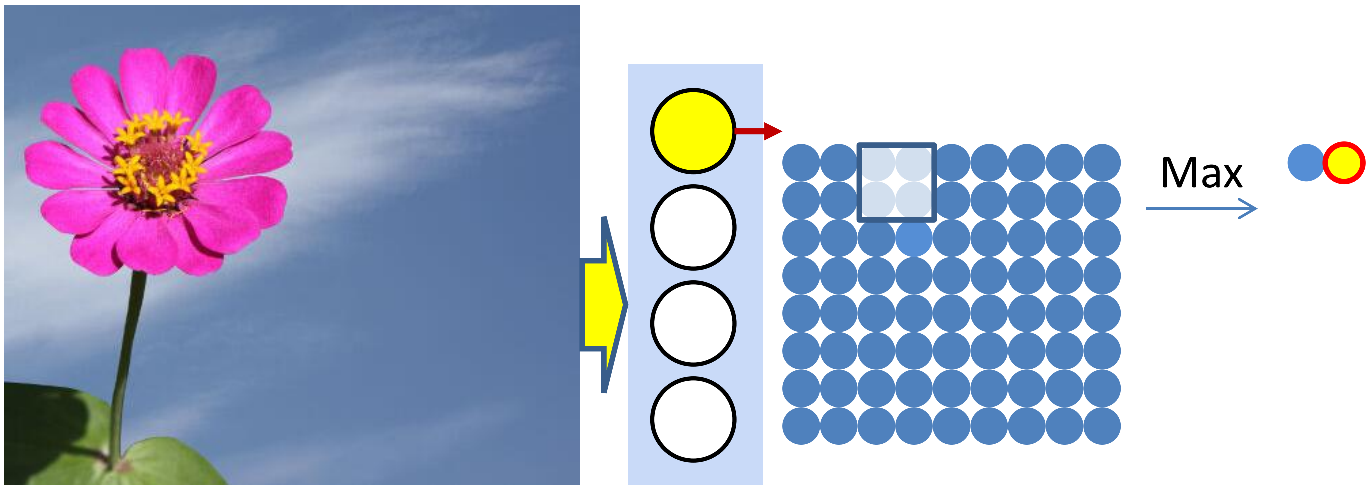
- The “max filter” operation too “scans” the picture

“Strides”



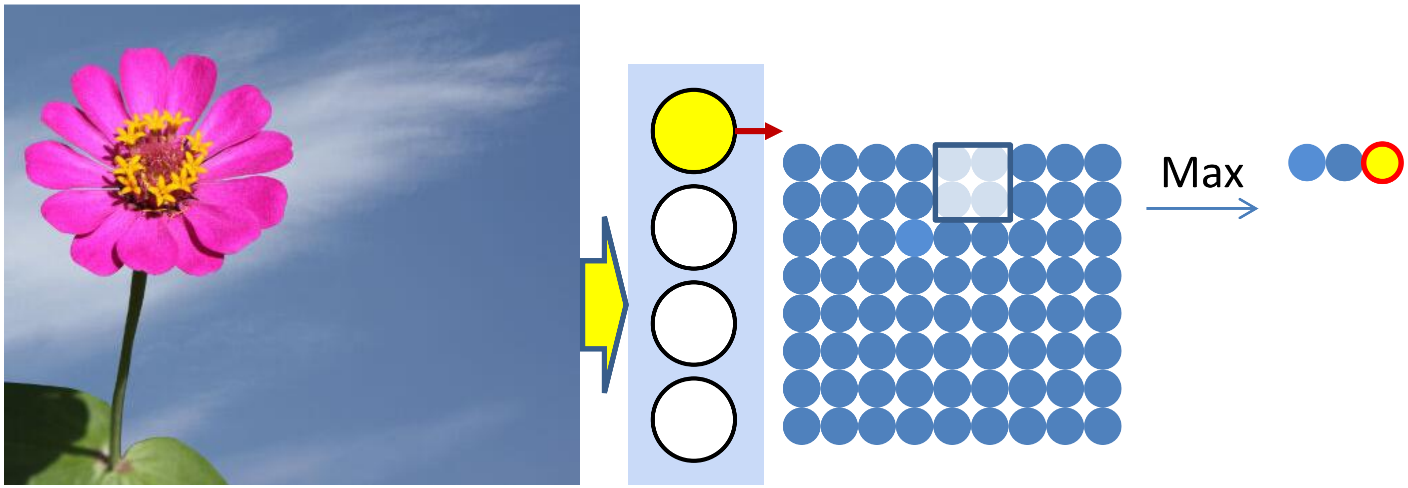
- The “max” operations may “stride” by more than one pixel

“Strides”



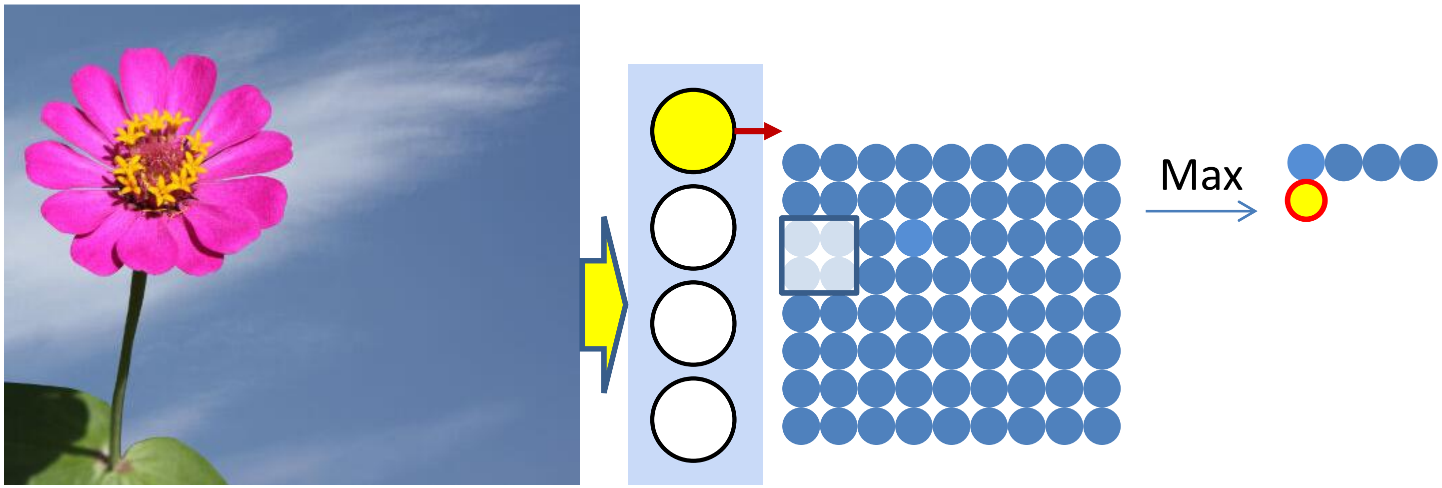
- The “max” operations may “stride” by more than one pixel

“Strides”



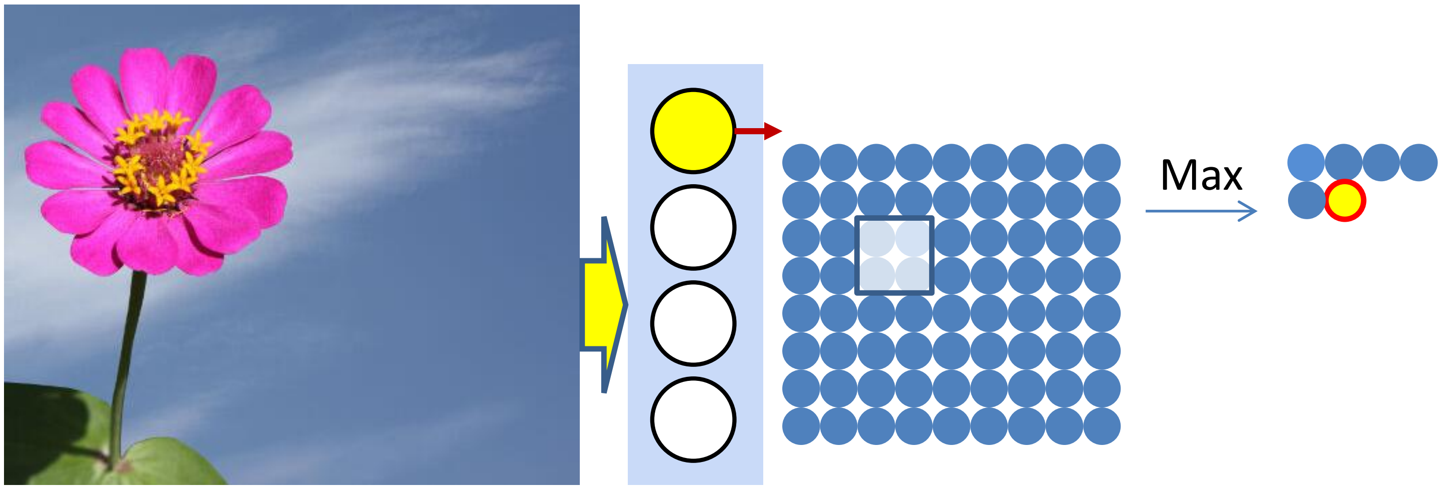
- The “max” operations may “stride” by more than one pixel

“Strides”



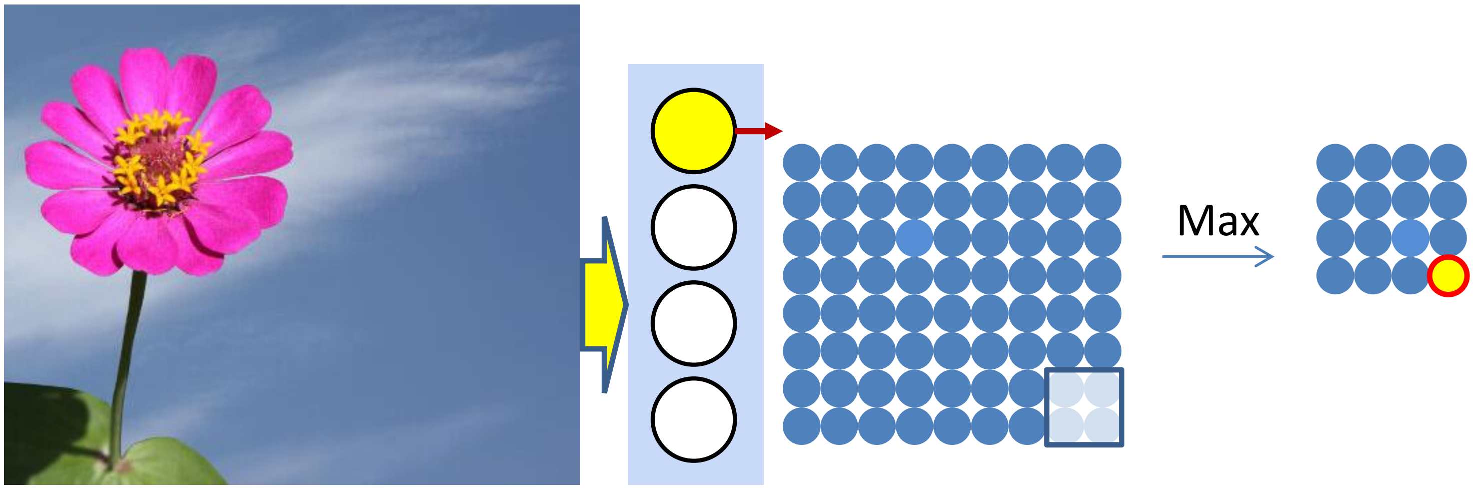
- The “max” operations may “stride” by more than one pixel

“Strides”



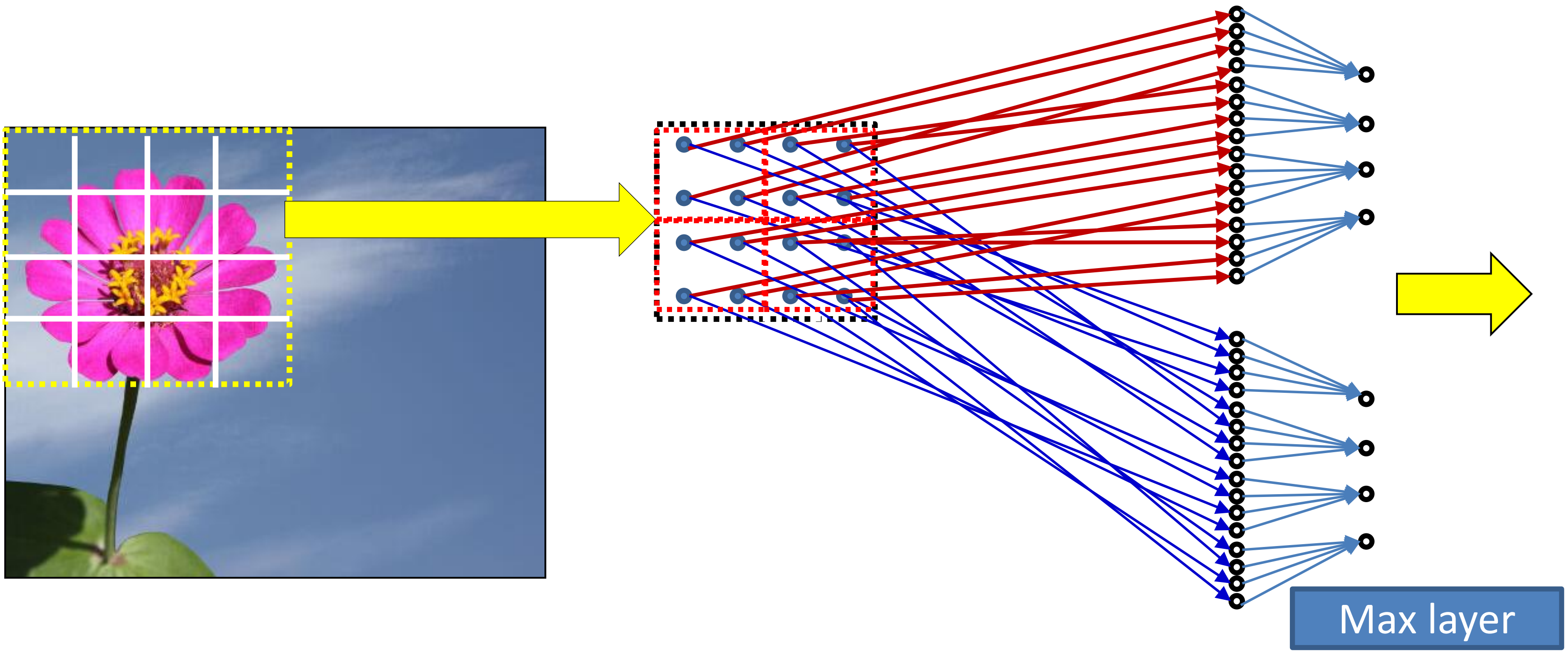
- The “max” operations may “stride” by more than one pixel

"Strides"



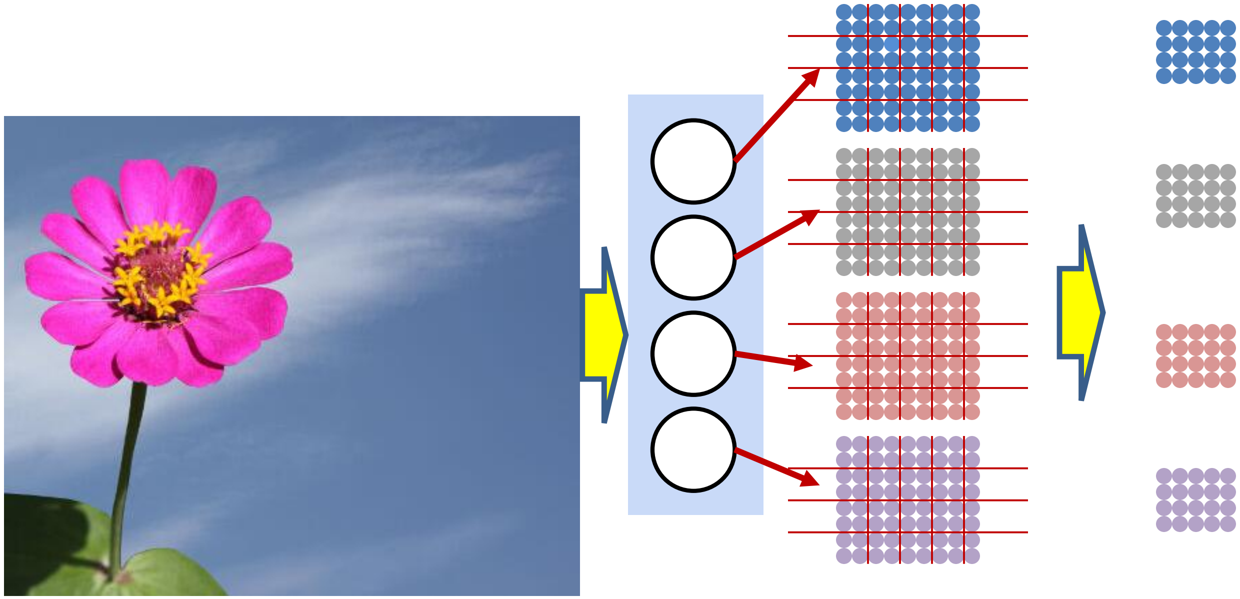
- The "max" operations may "stride" by more than one pixel
  - This will result in a *shrinking* of the map
  - The operation is usually called "pooling"
    - Pooling a number of outputs to get a single output
    - Also called "Down sampling"

Shrinking with a max



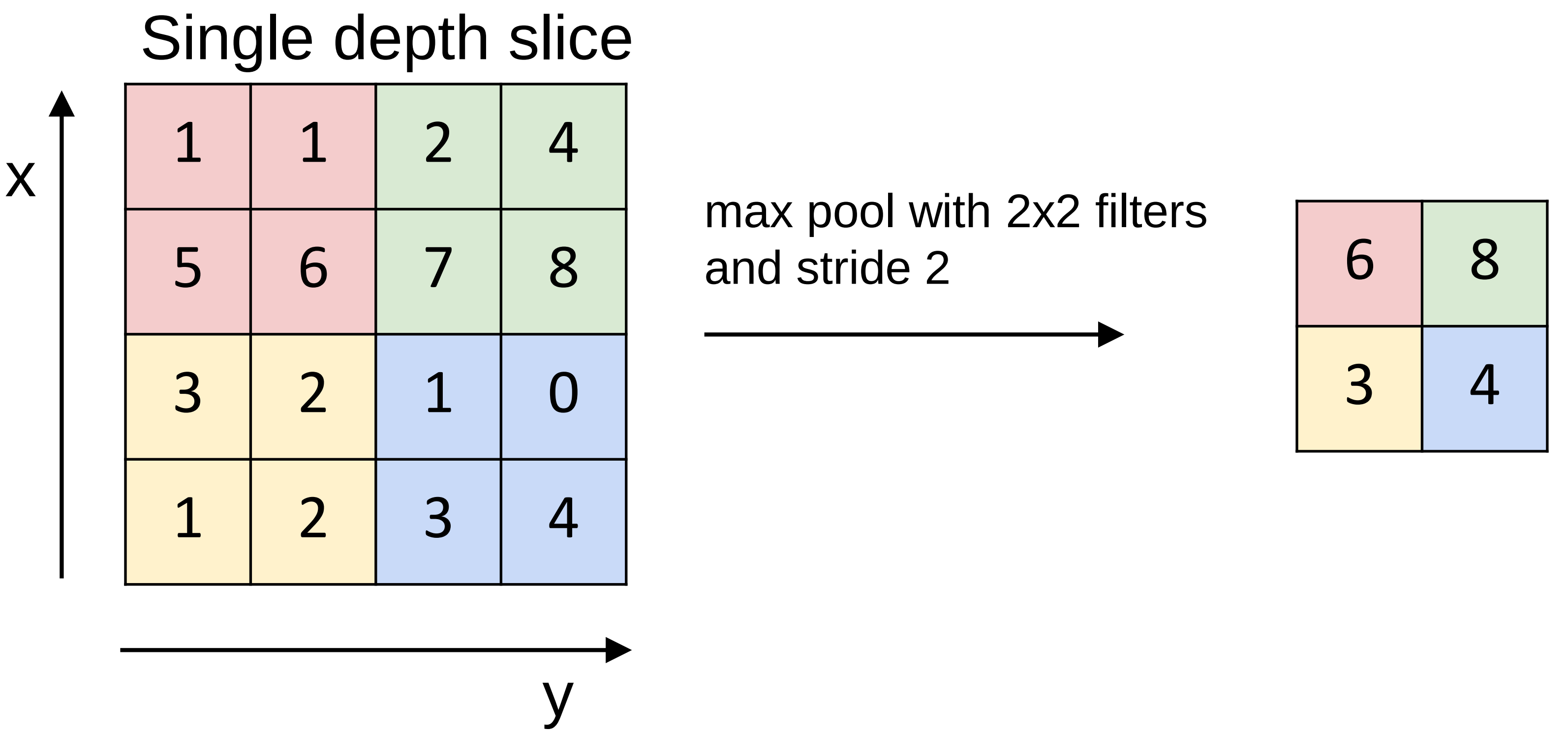
- In this example we actually *shrank* the image after the max
  - Adjacent "max" operators did not overlap
  - The stride was the size of the max filter itself

Non-overlapped strides

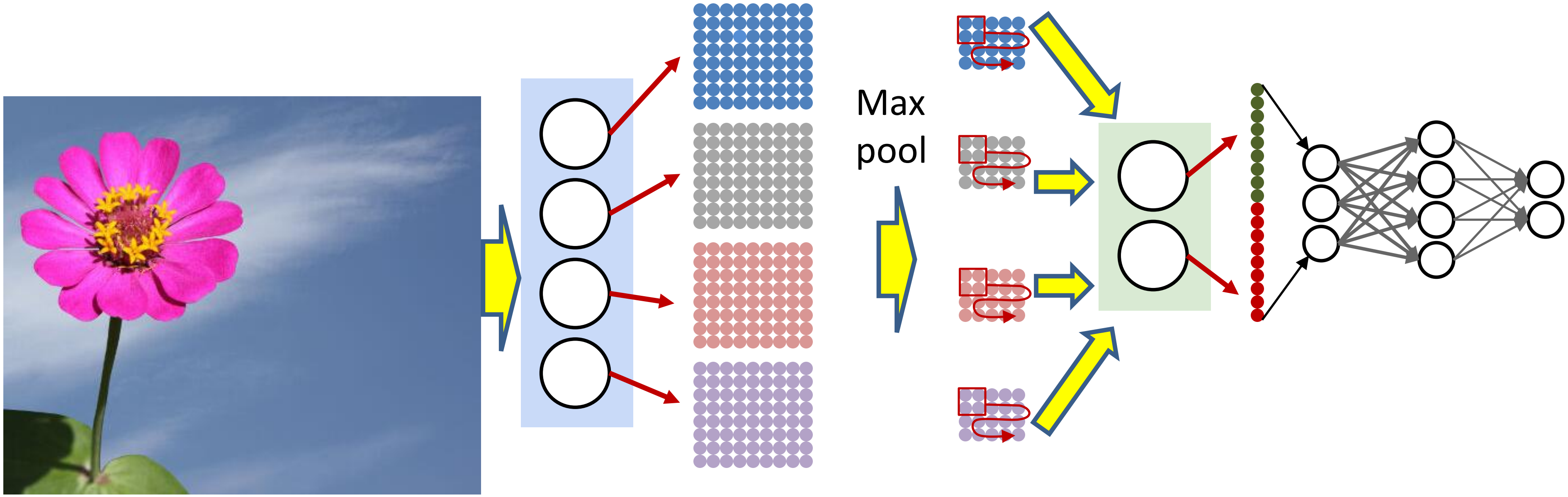


- Non-overlapping strides: Partition the output of the layer into blocks
- Within each block only retain the *highest* value
  – If you detect a petal anywhere in the block, a petal is detected..

Max Pooling

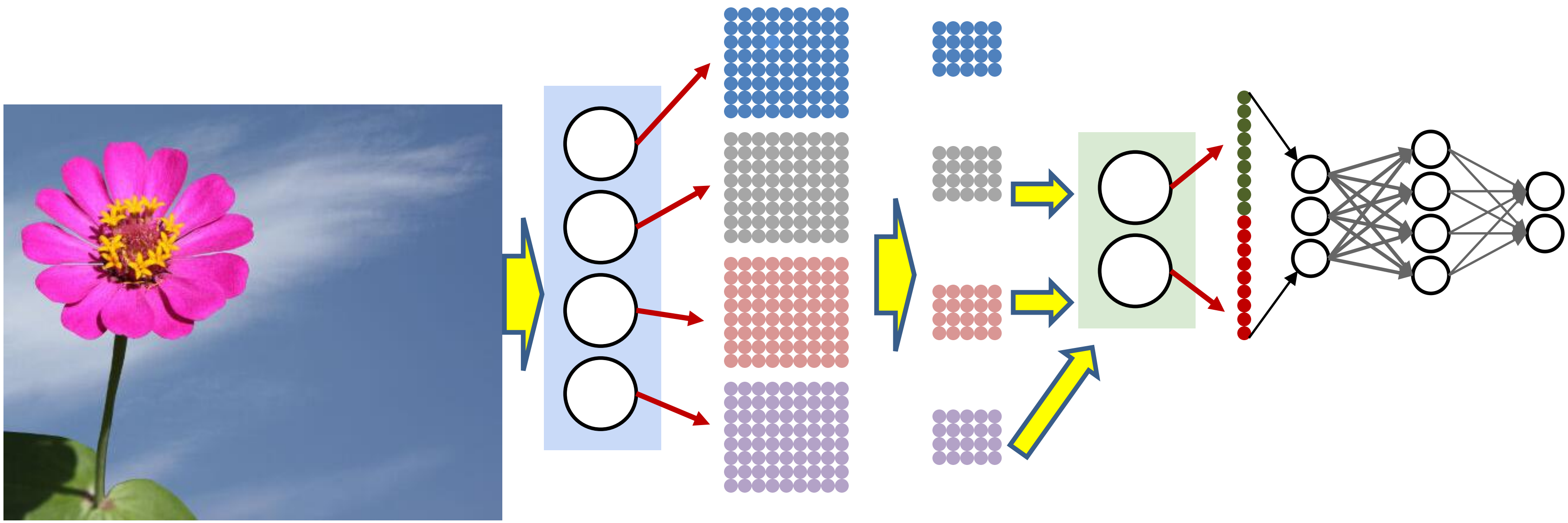

Single depth slice
x
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
y
max pool with 2x2 filters
and stride 2
6 8
3 4

Higher layers



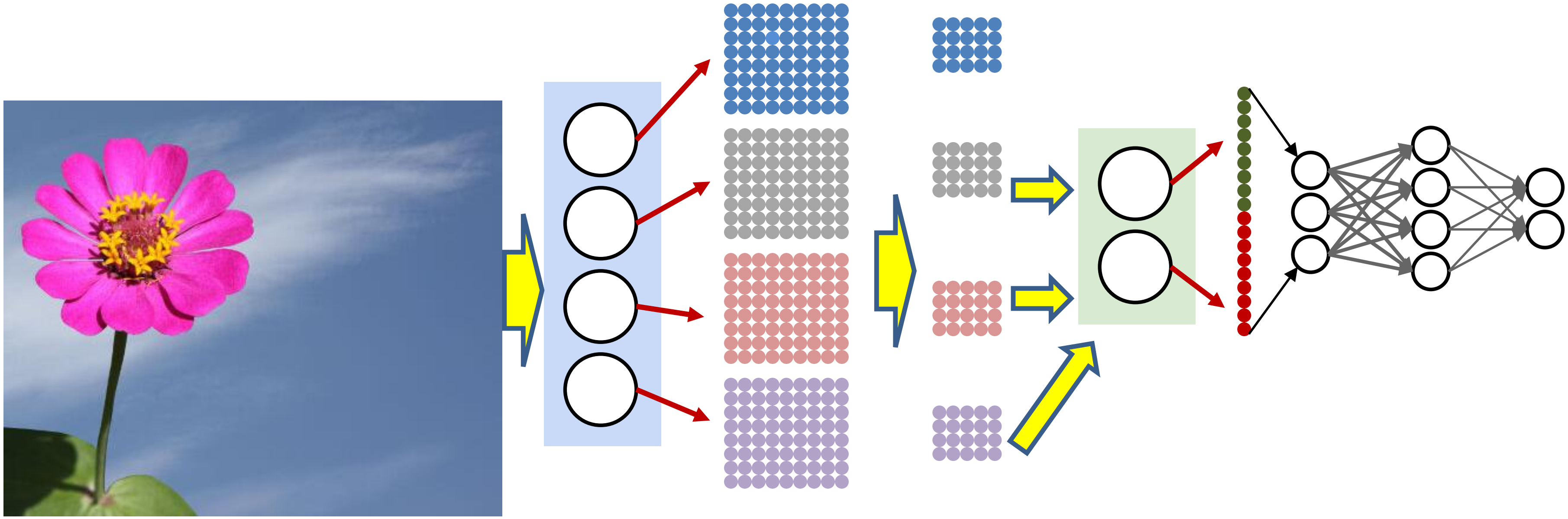
- The next layer works on the *max-pooled* maps

The overall structure



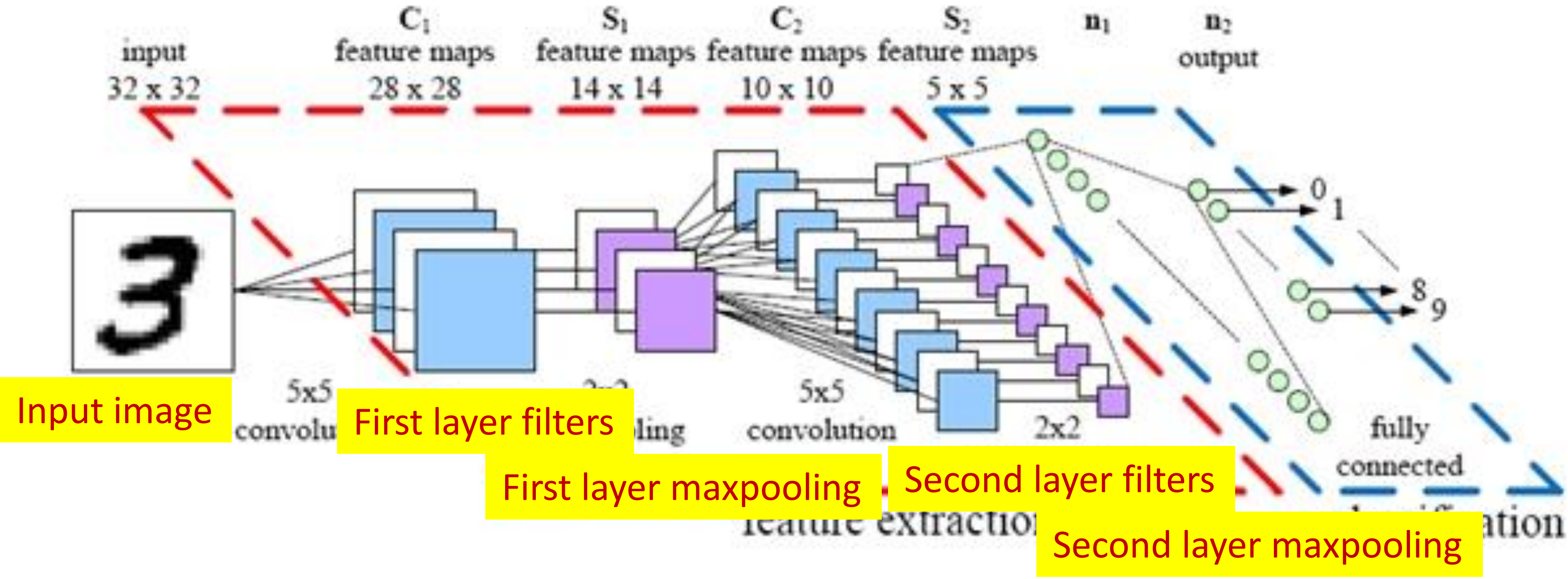
- In reality we can have many layers of "convolution" (scanning) followed by max pooling (and reduction) before the final MLP
  - The individual perceptrons at any "scanning" or "convolutive" layer are called "filters"
    - They "filter" the input image to produce an output image (map)
  - As mentioned,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

The overall structure

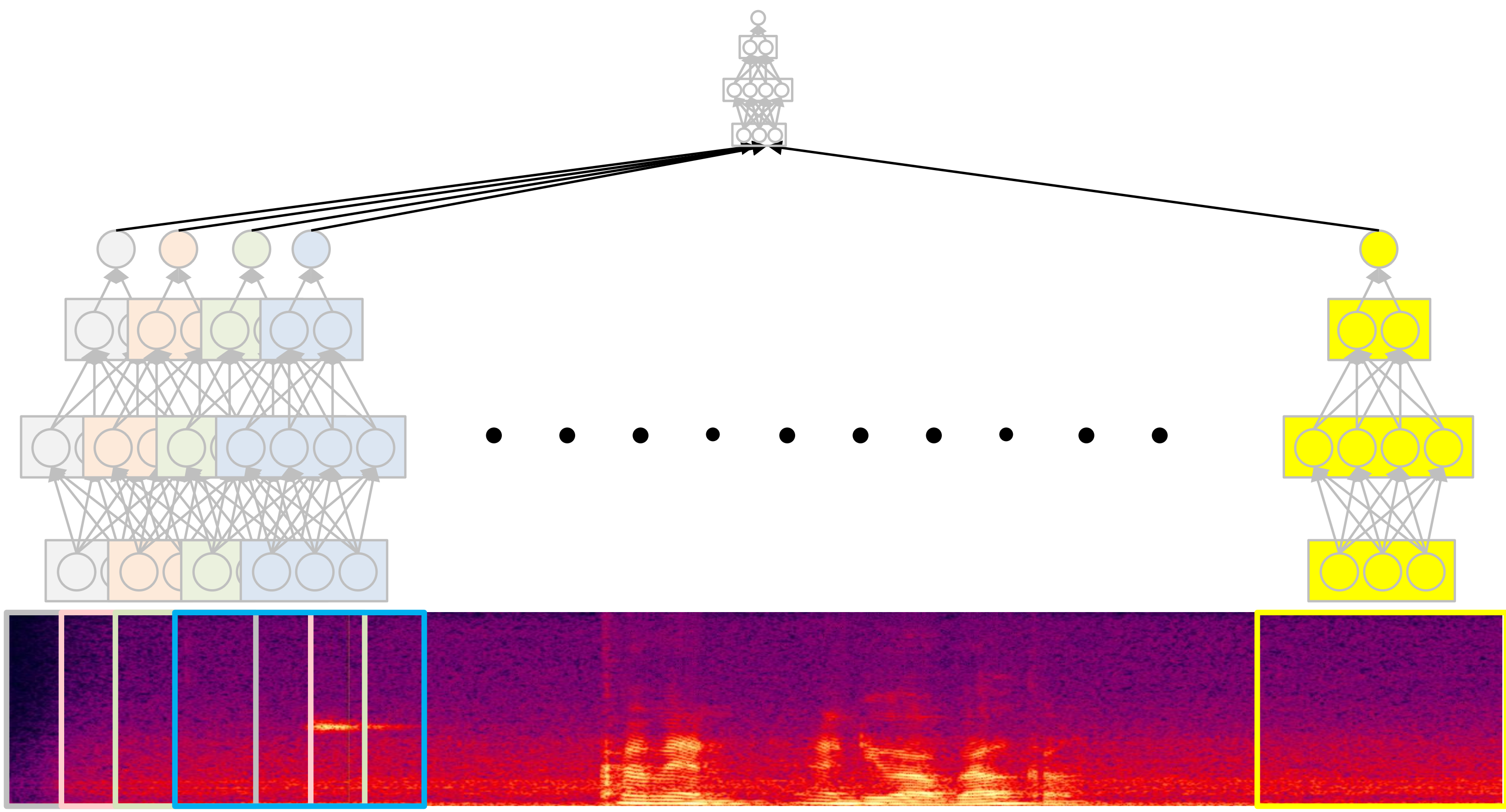


- This entire structure is called a ***Convolutive Neural Network***

Convolutive Neural Network

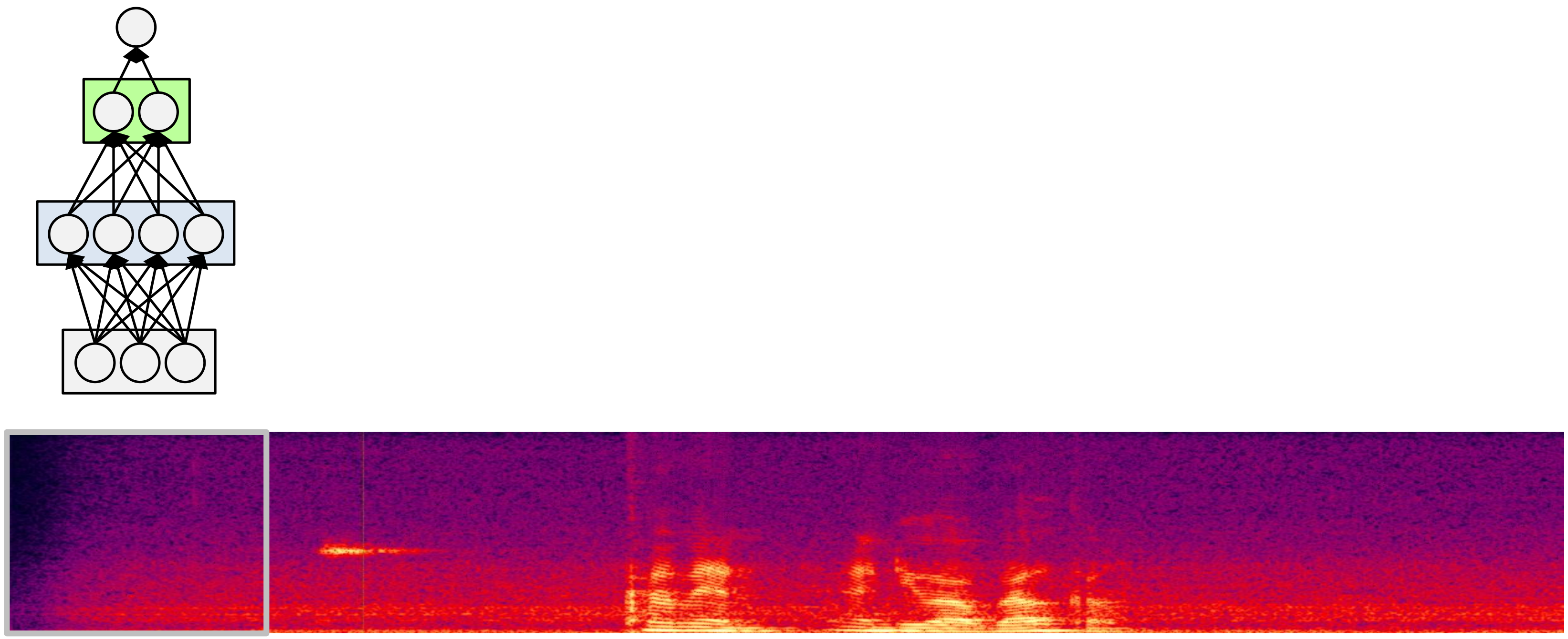


1-D convolution



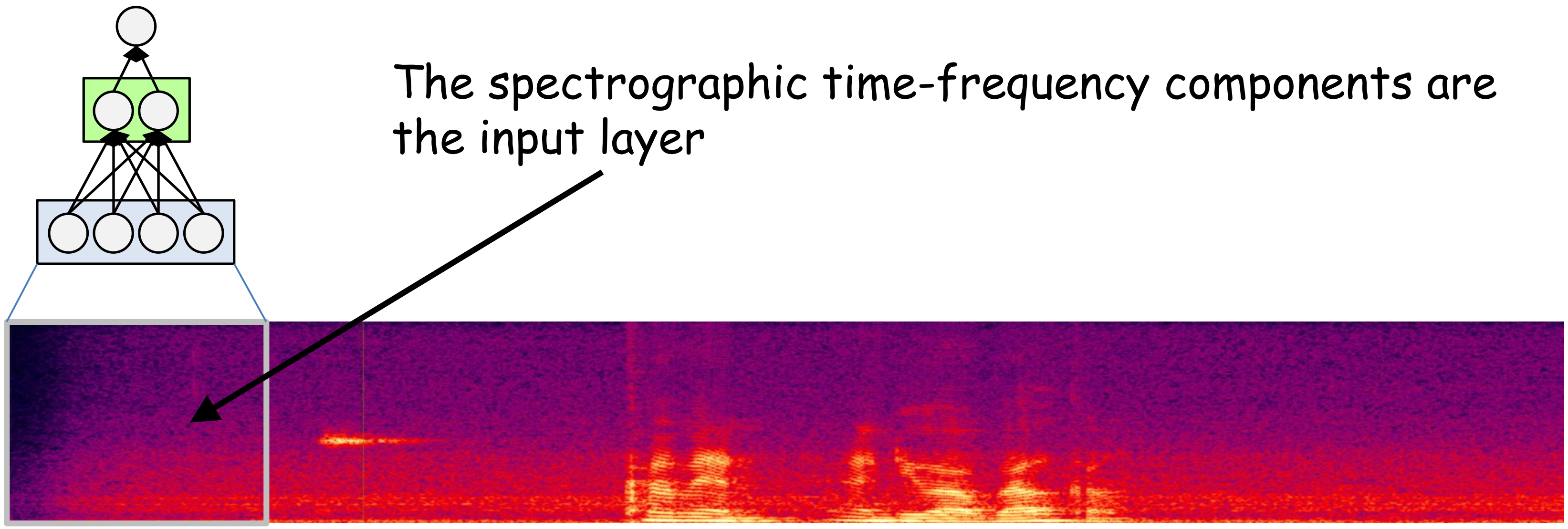
- The 1-D scan version of the convolutional neural network is the *time-delay neural network*
  – Used primarily for speech recognition

1-D scan version



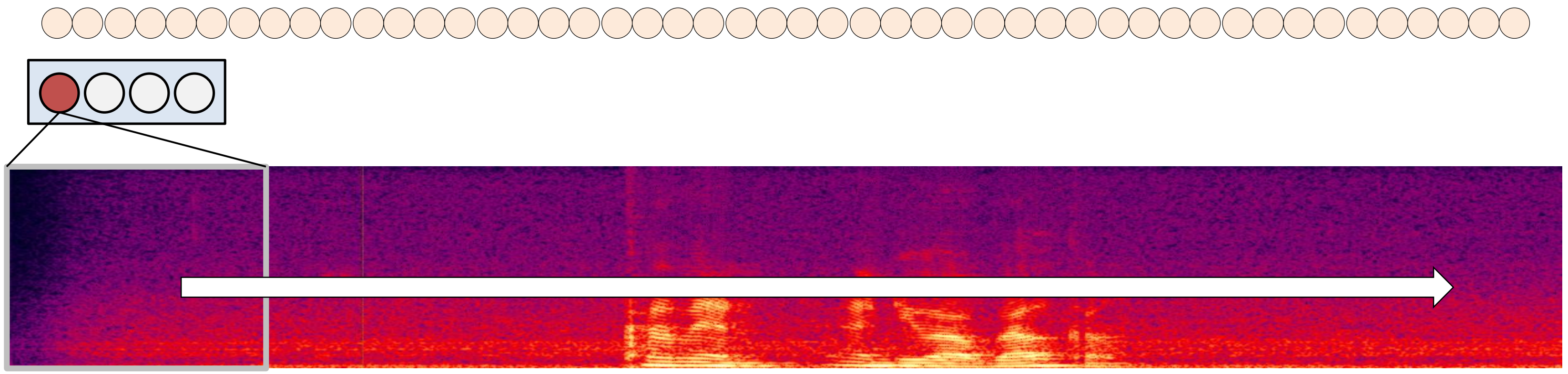
- The 1-D scan version of the convolutional neural network

1-D scan version



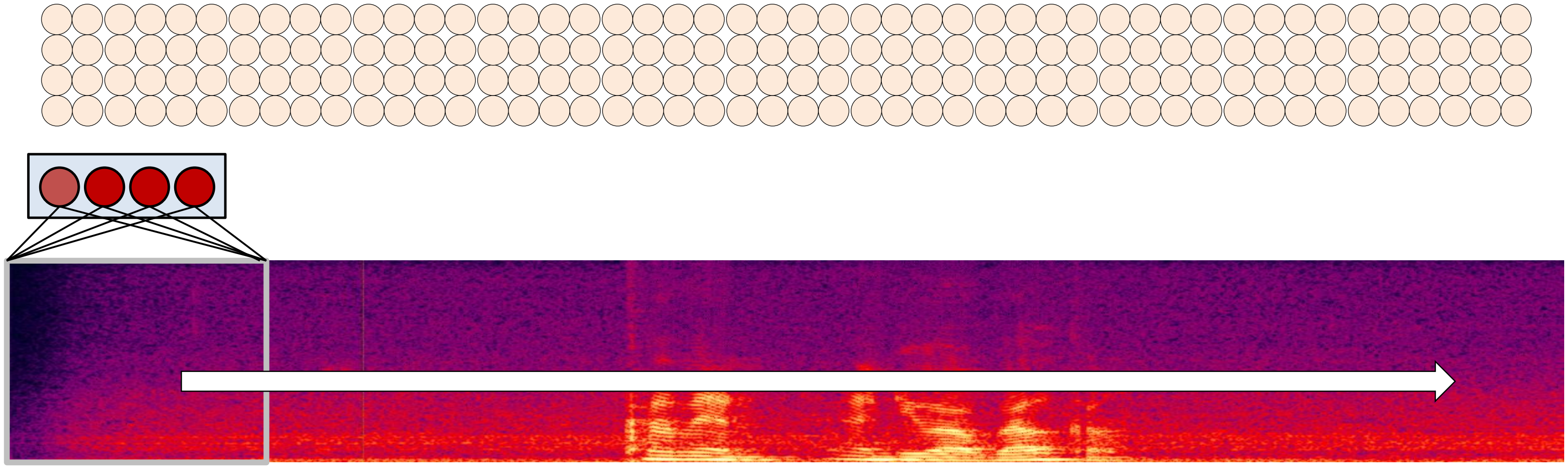
- The 1-D scan version of the convolutional neural network

1-D scan version



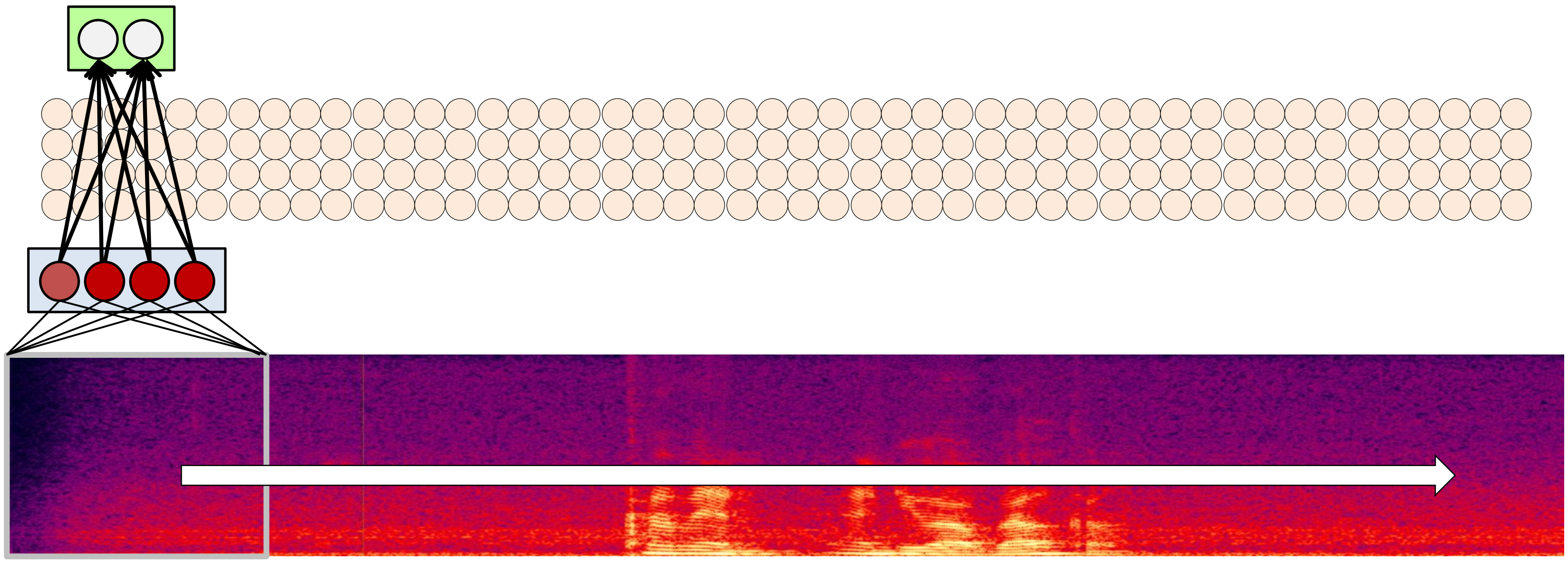
- The 1-D scan version of the convolutional neural network

1-D scan version



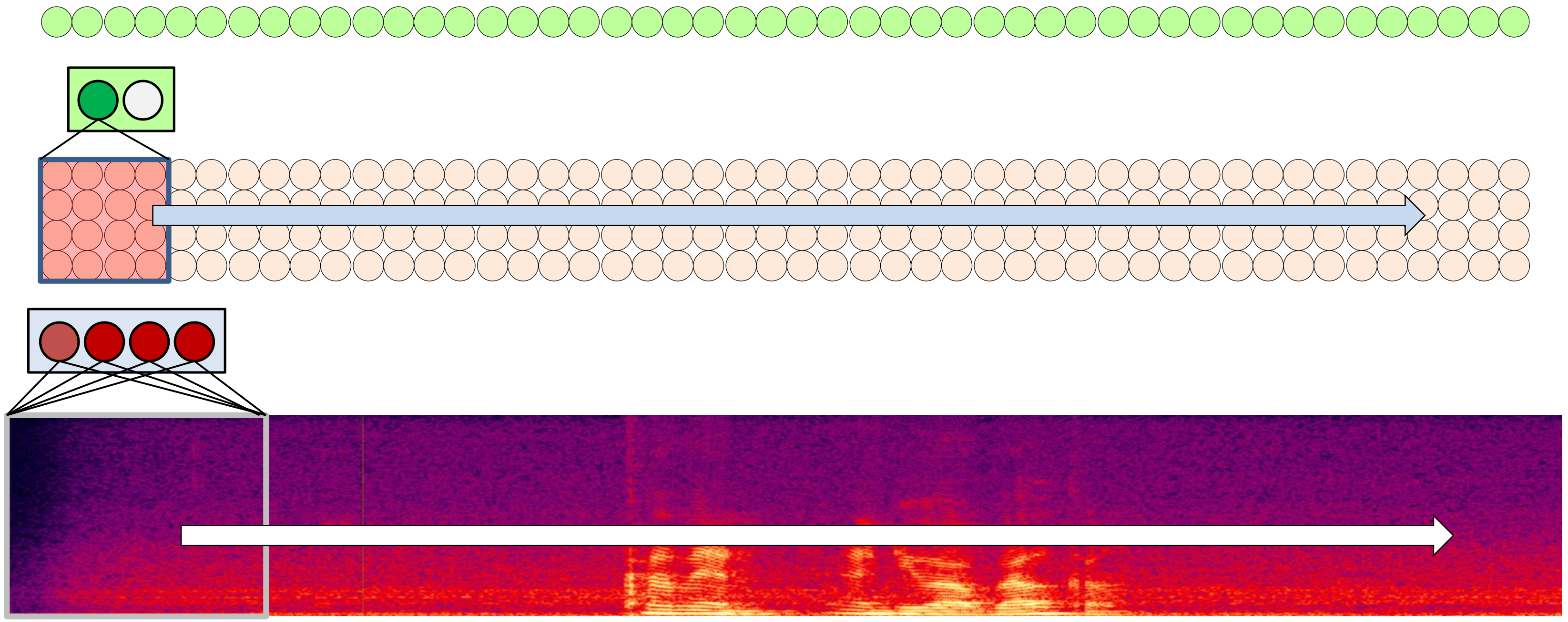
- The 1-D scan version of the convolutional neural network

1-D scan version



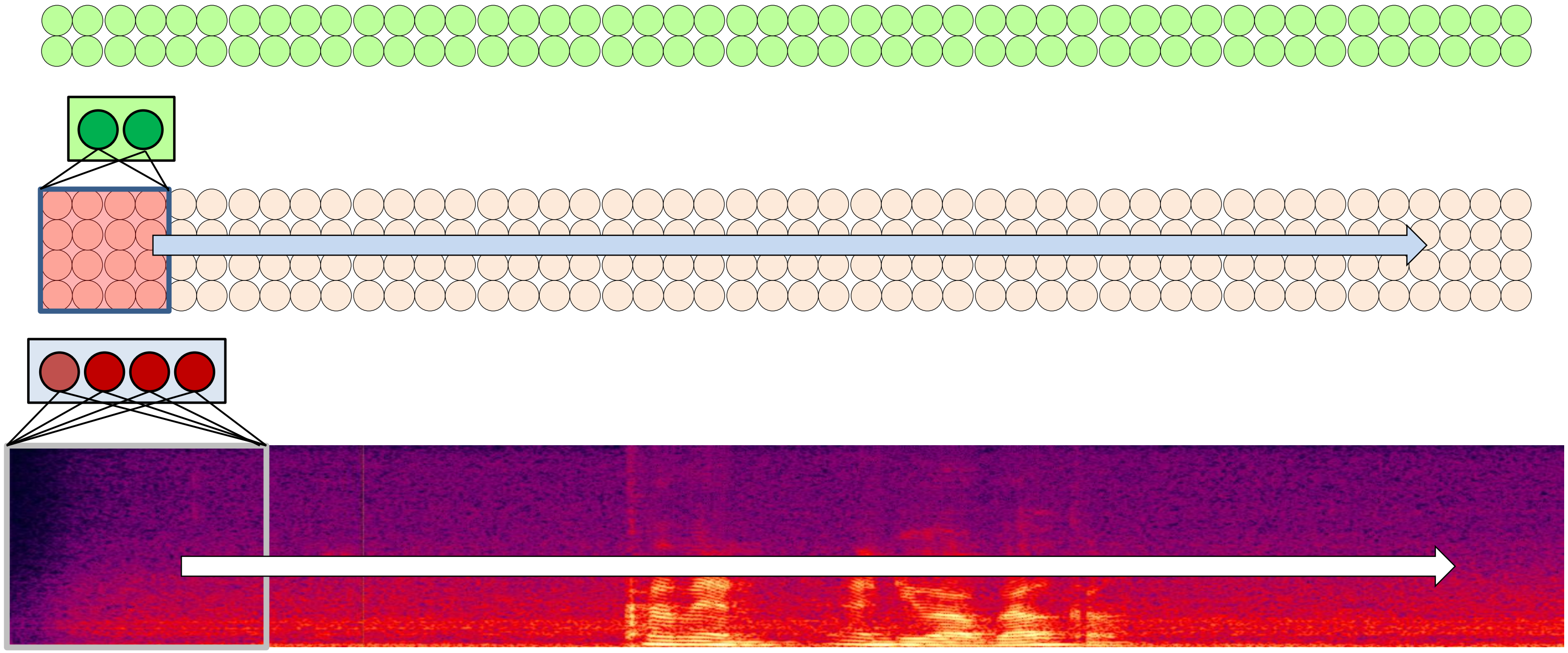
- The 1-D scan version of the convolutional neural network
  - Max pooling optional
    - Not generally done for speech

1-D scan version



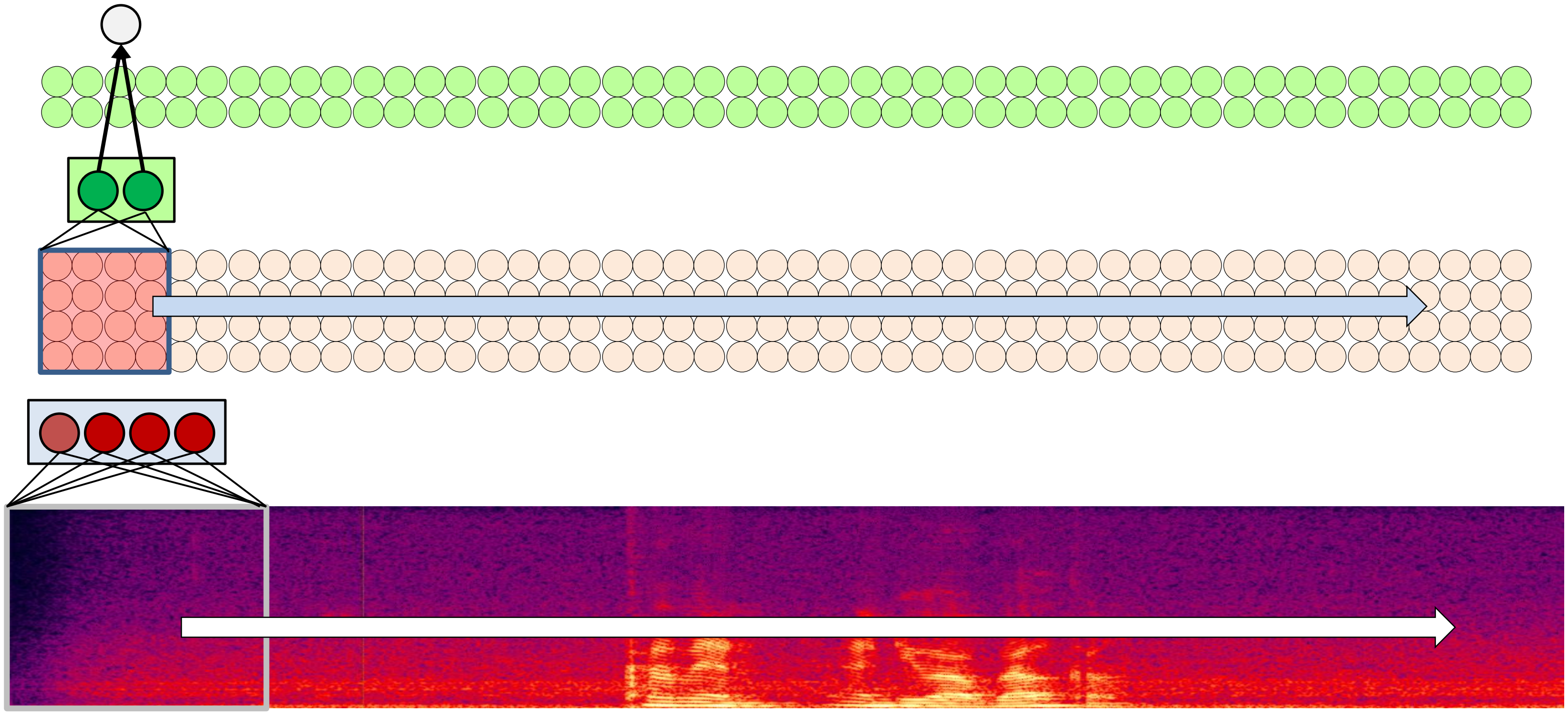
- The 1-D scan version of the convolutional neural network
  - Max pooling optional
    - Not generally done for speech

1-D scan version



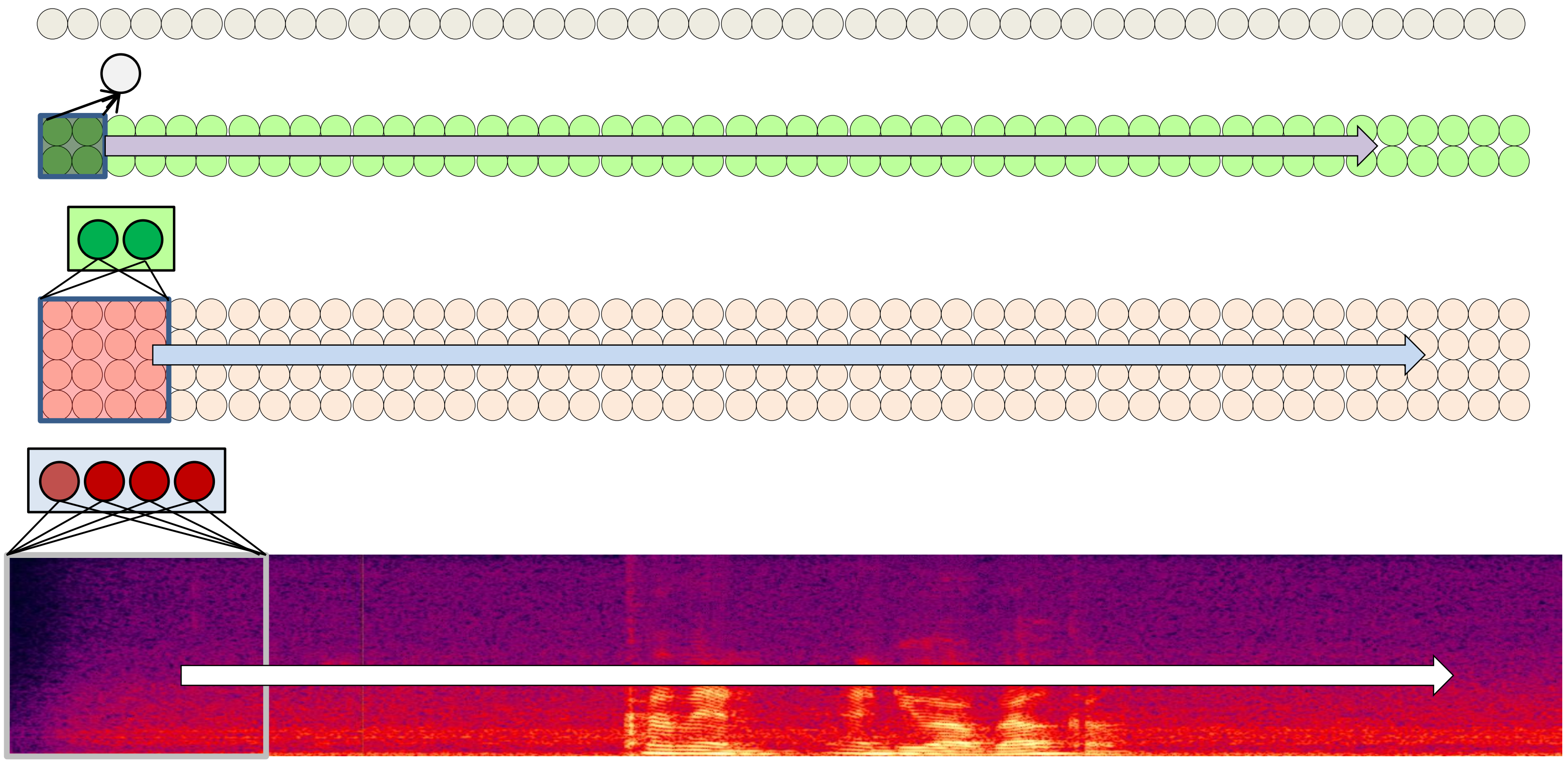
- The 1-D scan version of the convolutional neural network
  - Max pooling optional
    - Not generally done for speech

1-D scan version



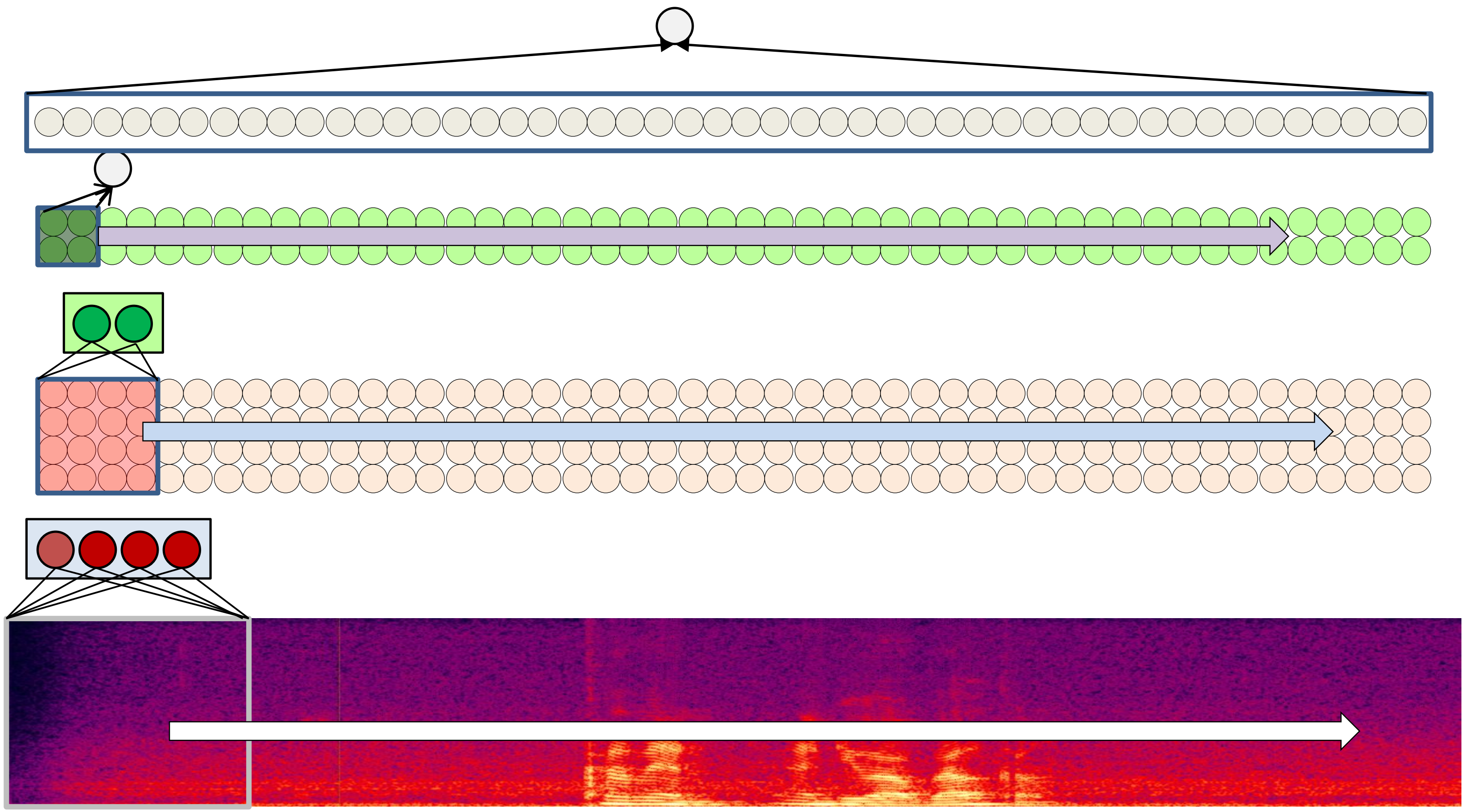
- The 1-D scan version of the convolutional neural network
  - Max pooling optional
    - Not generally done for speech

1-D scan version



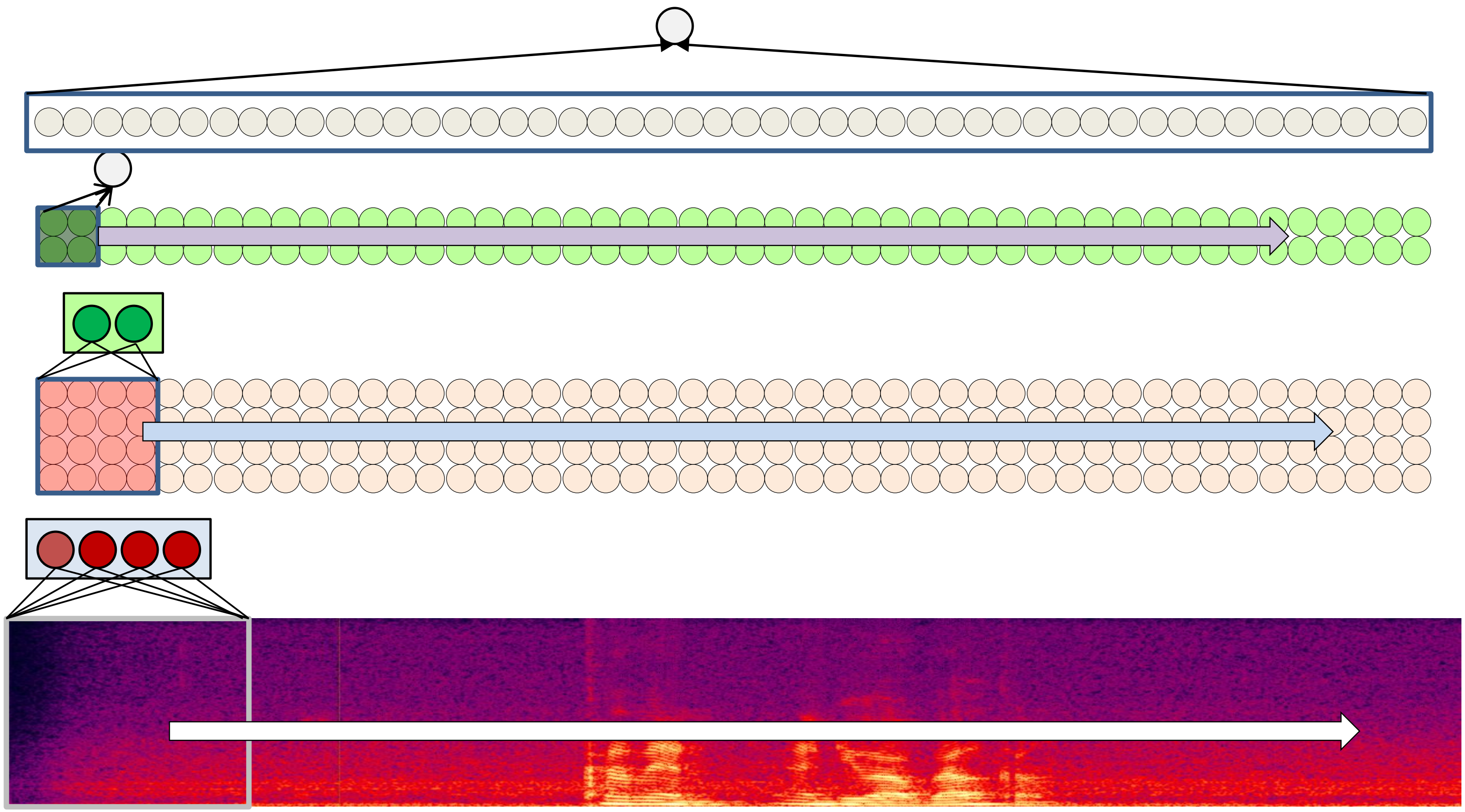
- The 1-D scan version of the convolutional neural network
  - Max pooling optional
    - Not generally done for speech

1-D scan version



- The 1-D scan version of the convolutional neural network
- A final perceptron (or MLP) to aggregate evidence
  - "Does this recording have the target word"

Time-Delay Neural Network



- This structure is called the ***Time-Delay Neural Network***

Story so far

- Neural networks learn patterns in a hierarchical manner
  - Simple to complex
- Pattern classification tasks such as “does this picture contain a cat” are best performed by scanning for the target pattern
- Scanning for patterns can be viewed as classification with a large shared-parameter network

- Scanning an input with a network and combining the outcomes is equivalent to scanning with individual neurons
  - First level neurons scan the input
  - Higher-level neurons scan the “maps” formed by lower-level neurons
  - A final “decision” layer (which may be a max, a perceptron, or an MLP) makes the final decision
- At each layer, a scan by a neuron may optionally be followed by a “max” (or any other) “pooling” operation to account for deformation

- For 2-D (or higher-dimensional) scans, the structure is called a convnet
- For 1-D scan along time, it is called a Time-delay neural network