Deep Neural Networks
Scanning for patterns (aka convolutional networks)

Bhiksha Raj

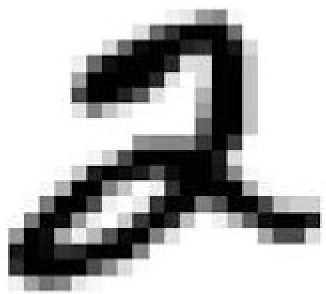
11-785, Spring 2020

Story so far

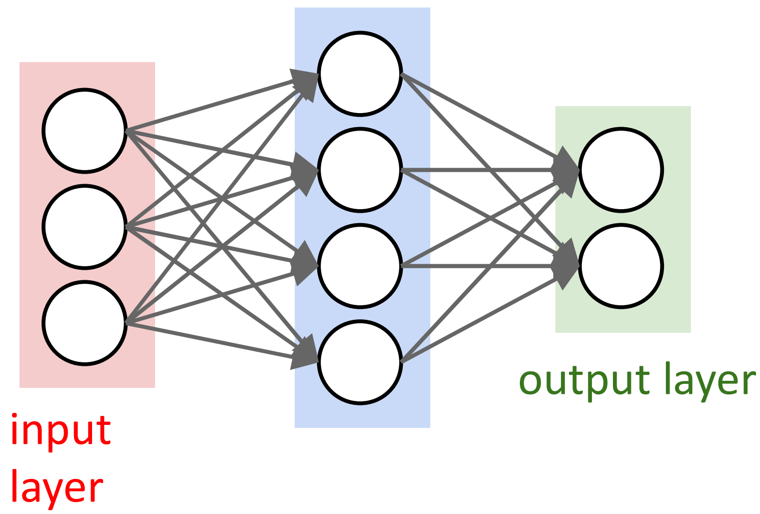
- **MLPs are universal function approximators**
  – Boolean functions, classifiers, and regressions

- **MLPs can be trained through variations of gradient descent**
  – Gradients can be computed by backpropagation

The model so far

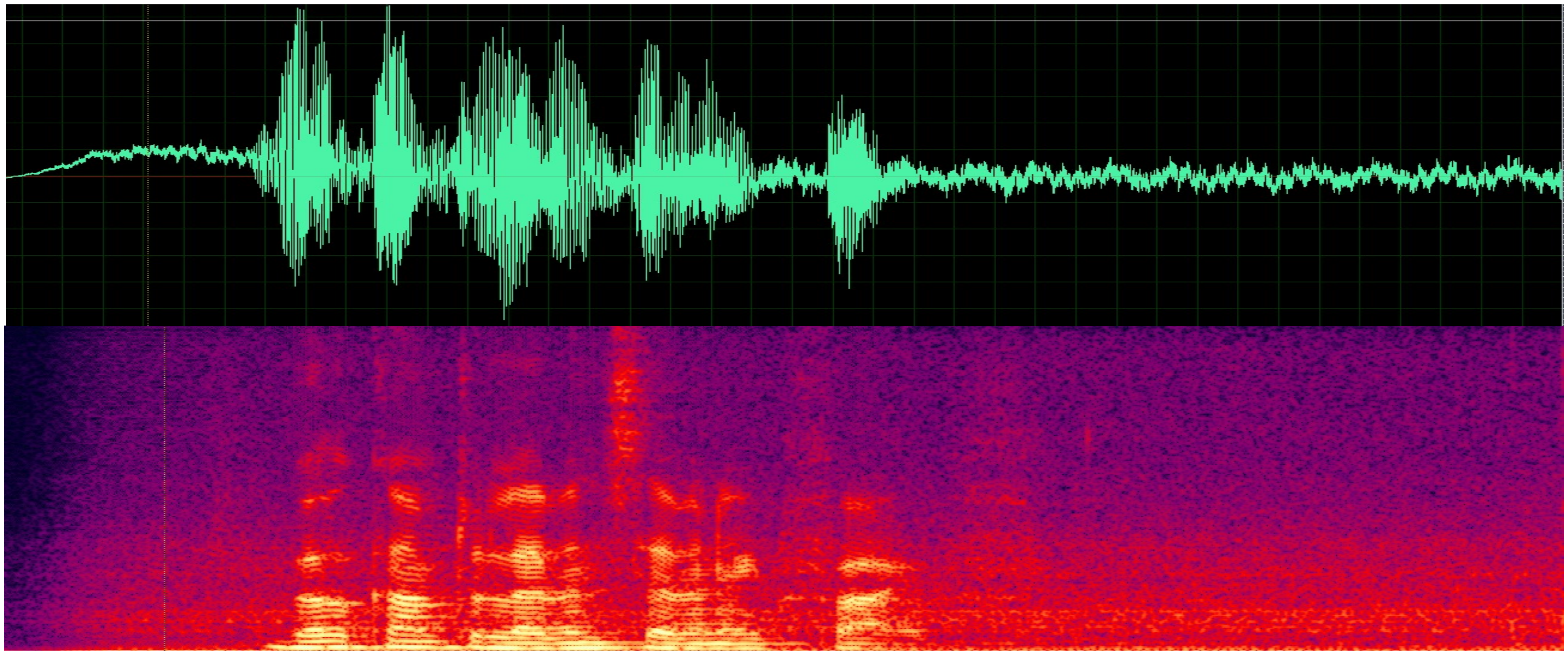


Or, more generally
a vector input



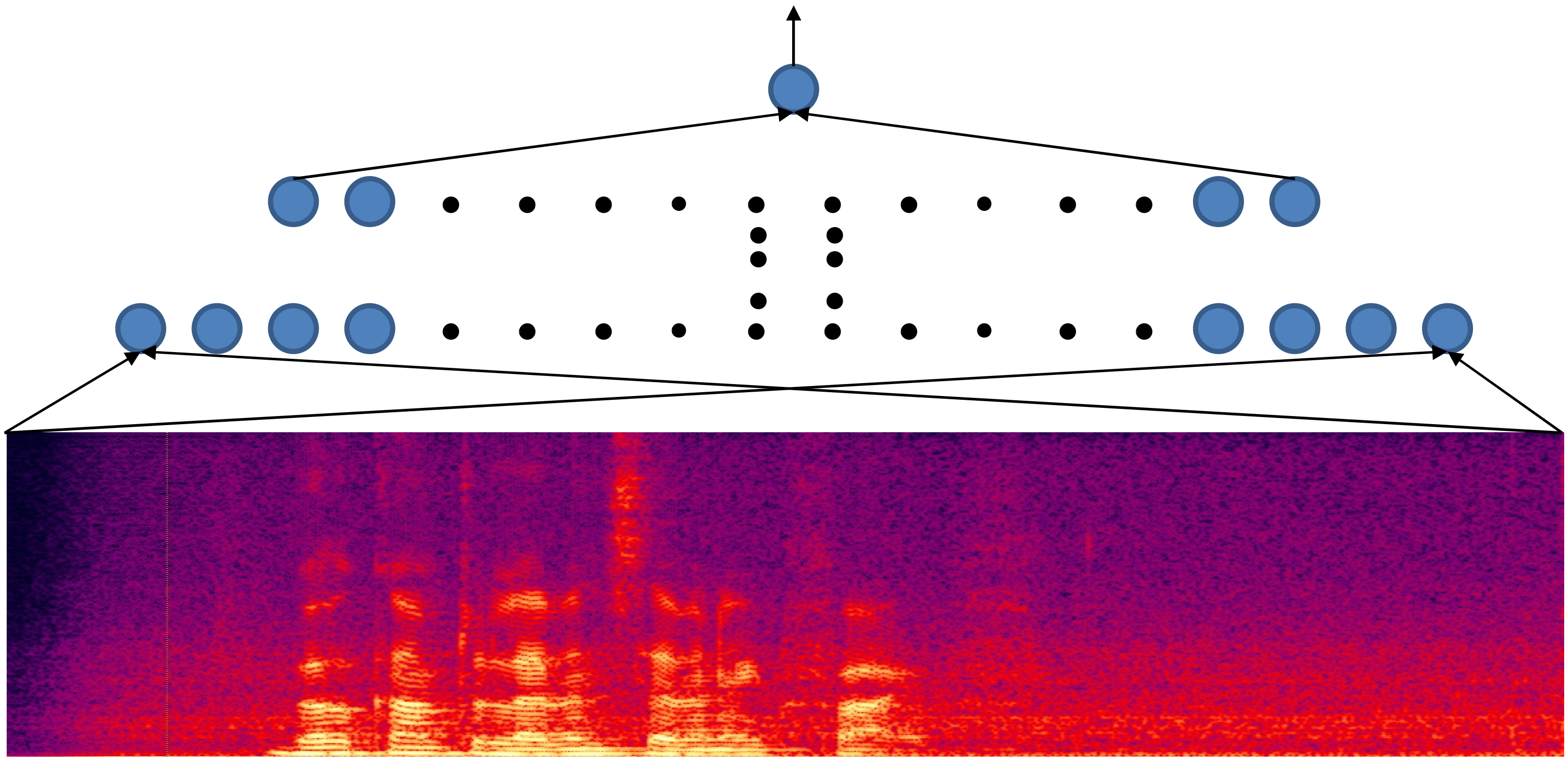
- Can recognize patterns in data
  - E.g. digits
  - Or any other vector data

A new problem



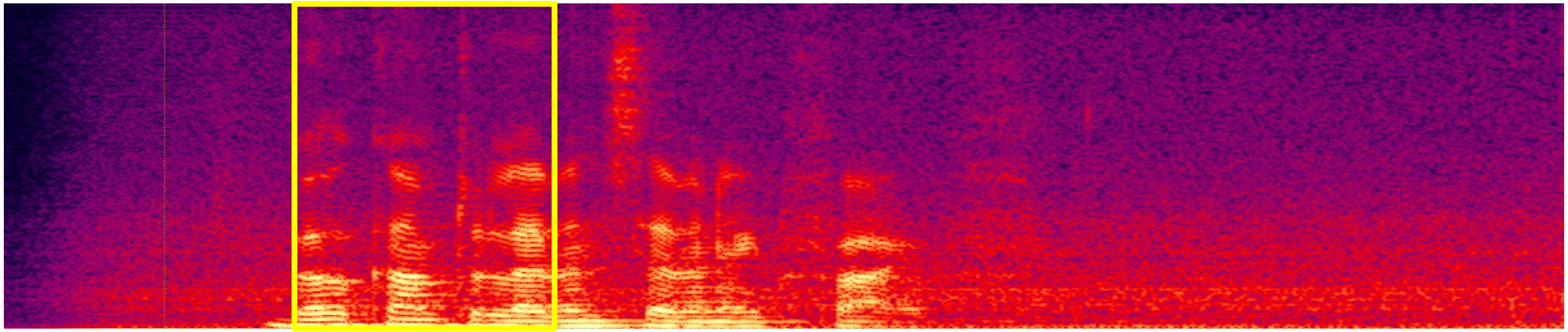
- Does this signal contain the word “Welcome”?
- Compose an MLP for this problem.
  - Assuming all recordings are exactly the same length..

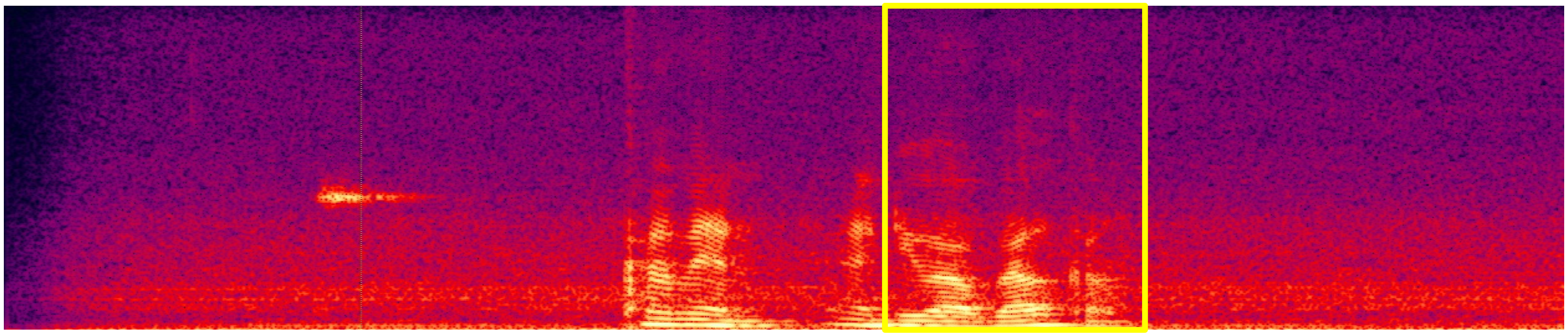
Finding a Welcome



- Trivial solution: Train an MLP for the entire recording

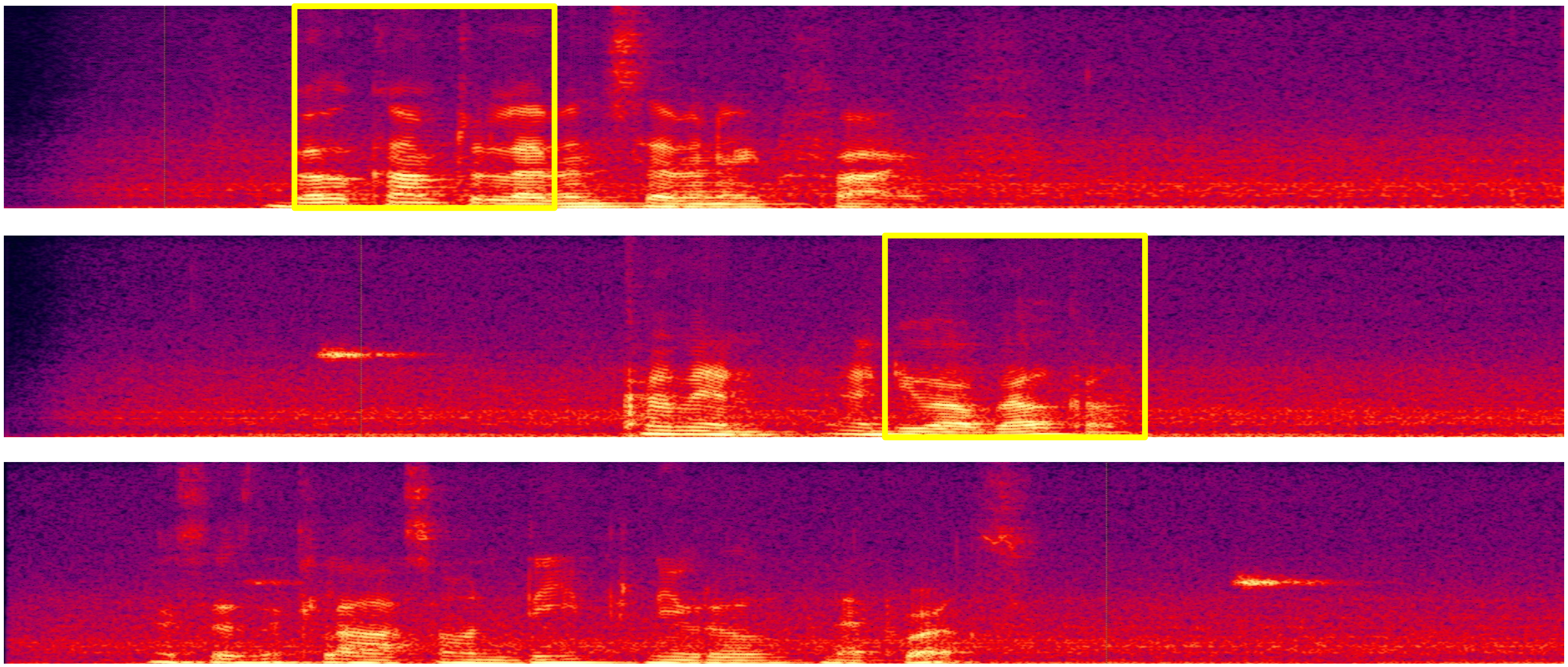
Finding a Welcome





- 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

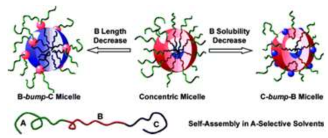
Finding a Welcome



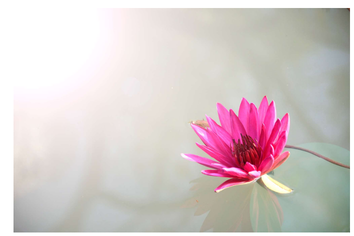
- Need a *simple* network that will fire regardless of the location of "Welcome"
  - and not fire when there is none

Flowers



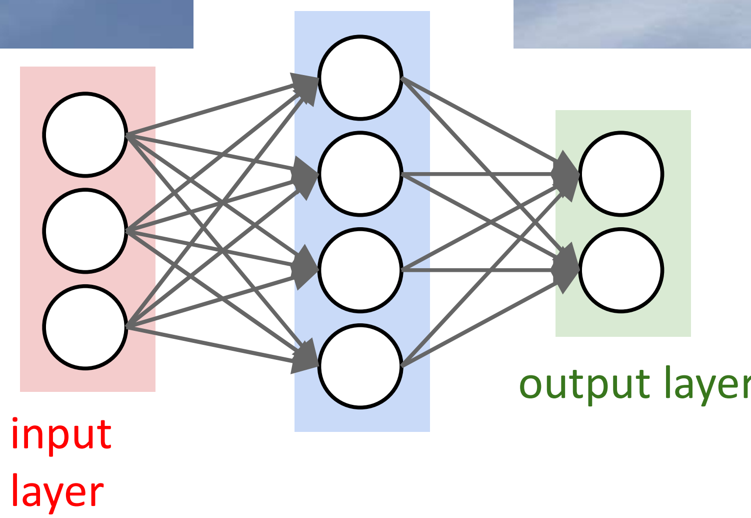







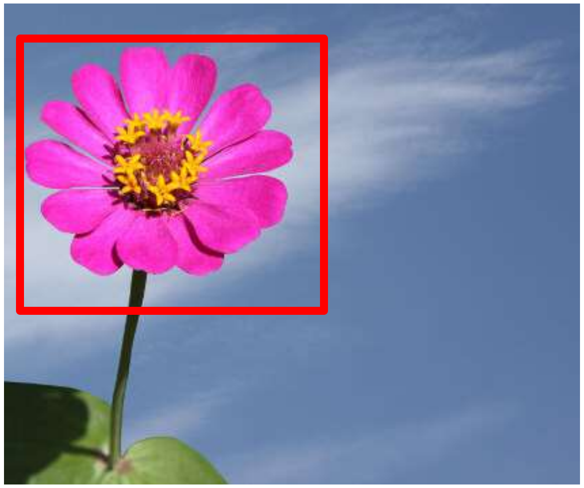
- Is there a flower in any of these images

A problem



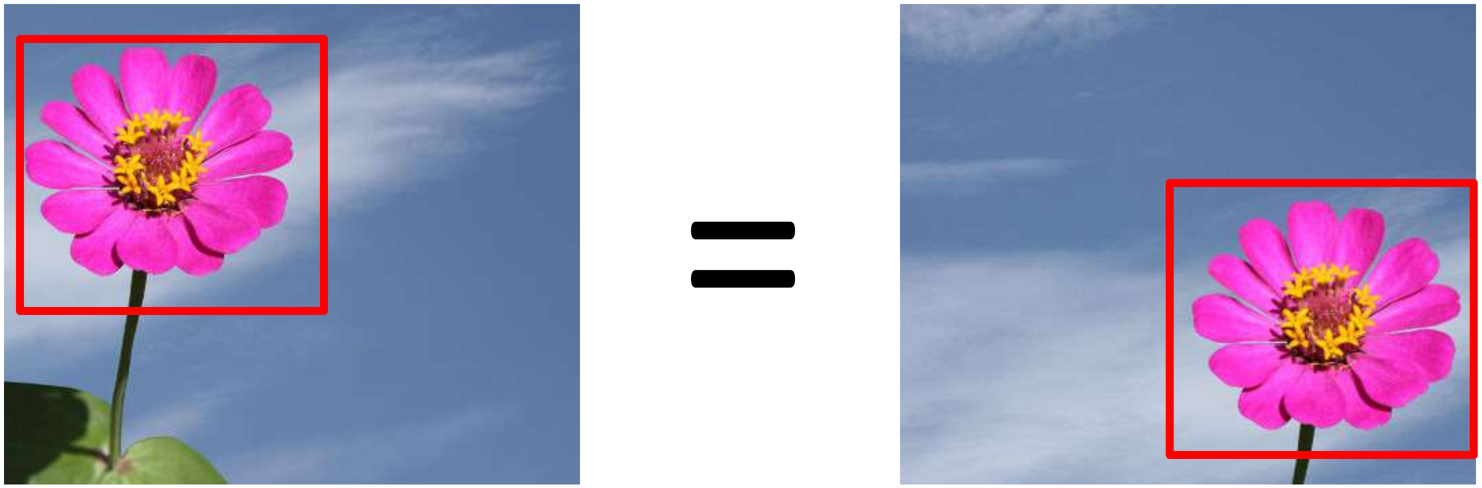
- Will an MLP that recognizes the left image as a flower also recognize the one on the right as a flower?

A problem



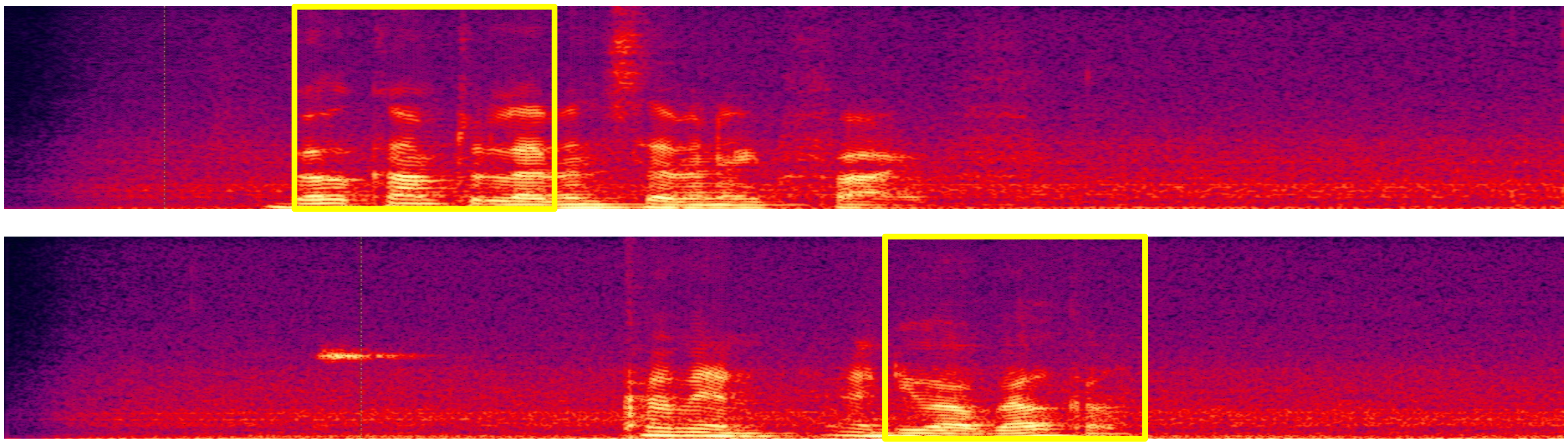
- Need a network that will "fire" regardless of the precise location of the target object

The need for *shift invariance*



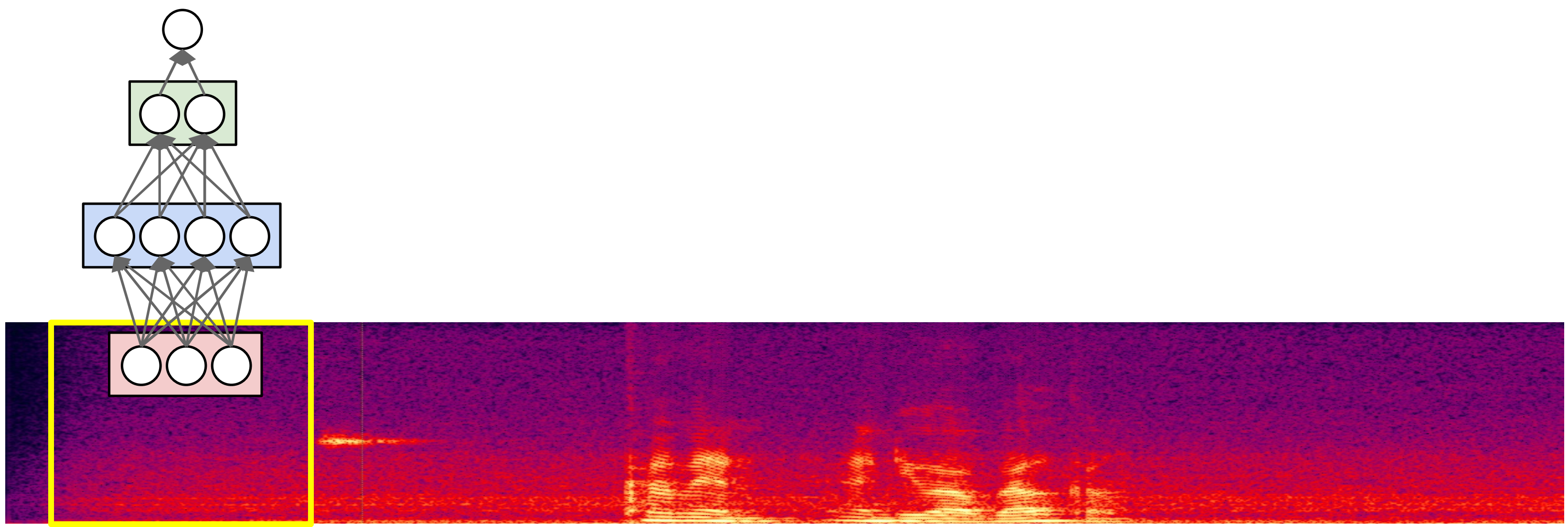
- In many problems the *location* of a pattern is not important
  - Only the presence of the pattern
- Conventional MLPs are sensitive to the location of the pattern
  - Moving it by one component results in an entirely different input that the MLP wont recognize
- Requirement: Network must be *shift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

The need for *shift invariance*



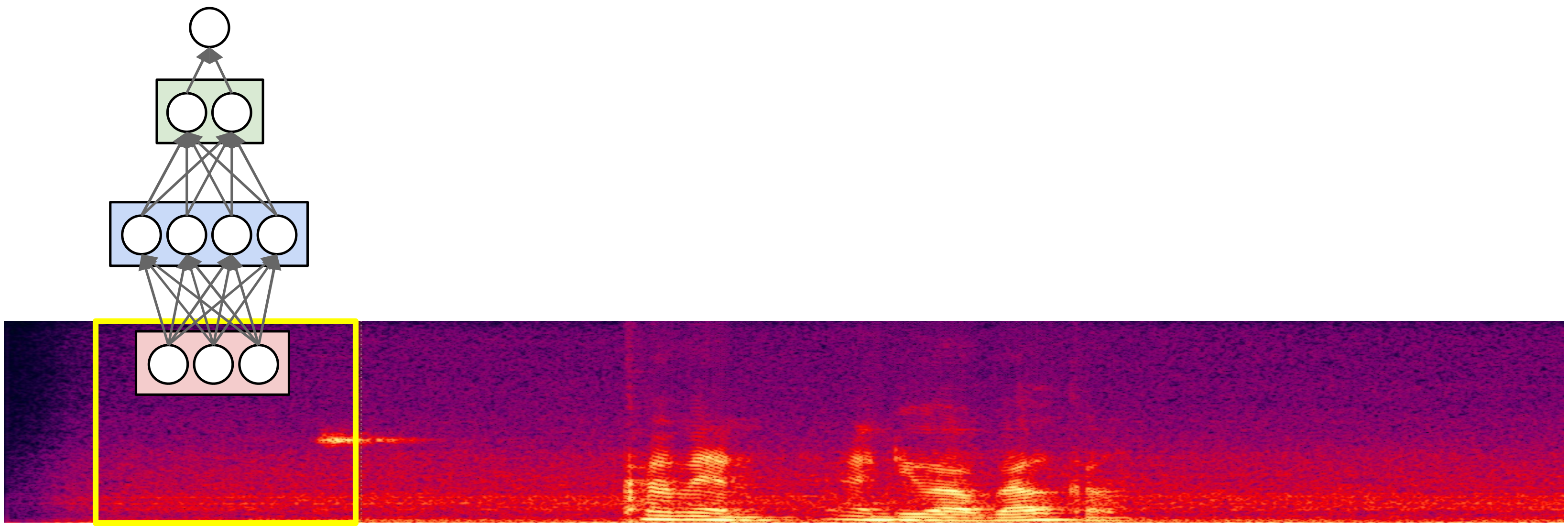
- In many problems the *location* of a pattern is not important
  - Only the presence of the pattern
- Conventional MLPs are sensitive to the location of the pattern
  - Moving it by one component results in an entirely different input that the MLP wont recognize
- Requirement: Network must be *shift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

Solution: Scan



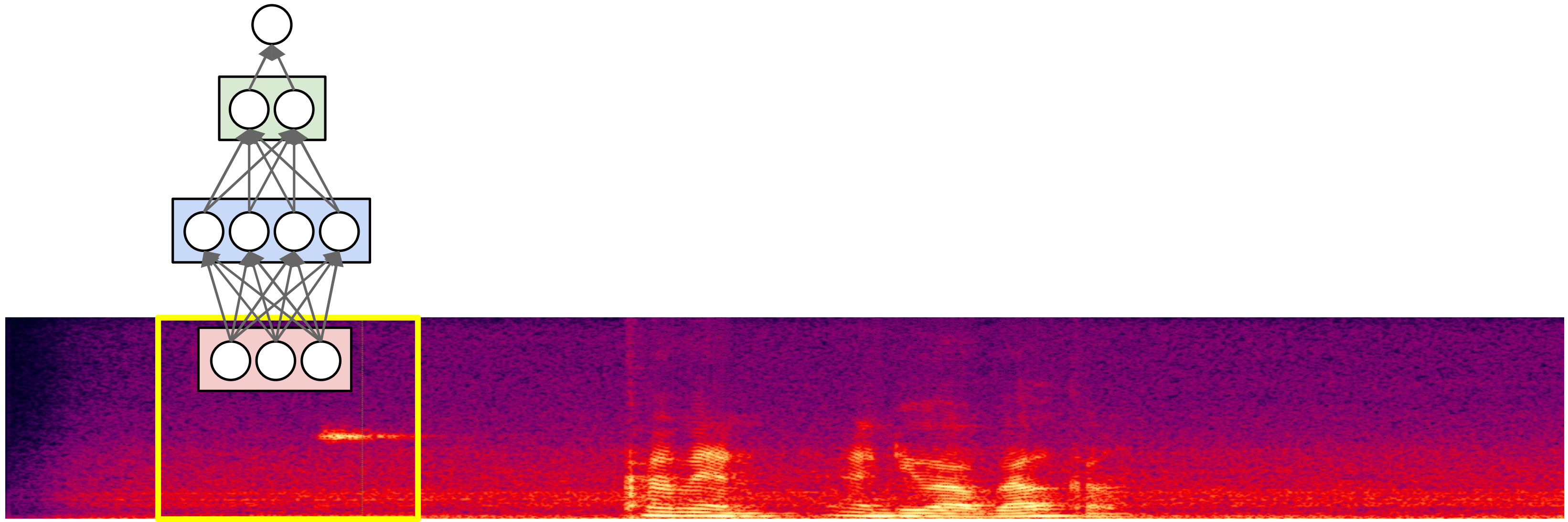
- Scan for the target word
  – The spectral time-frequency components in a "window" are input to a "welcome-detector" MLP

Solution: Scan



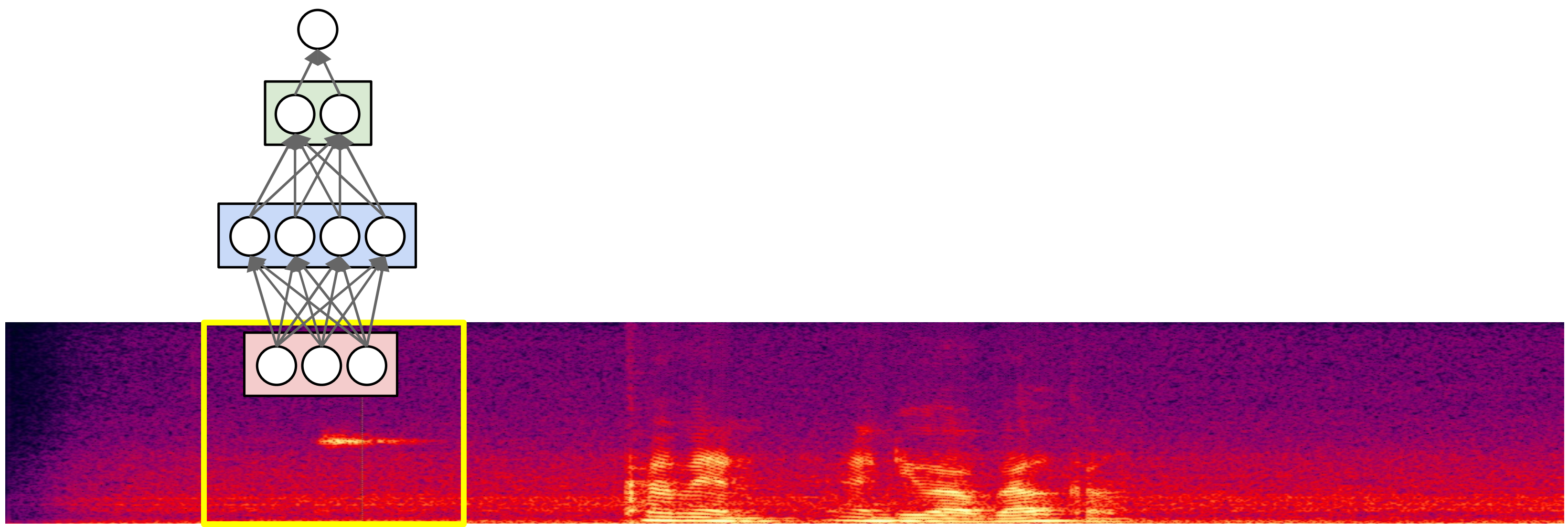
- Scan for the target word
  - The spectral time-frequency components in a "window" are input to a "welcome-detector" MLP

Solution: Scan



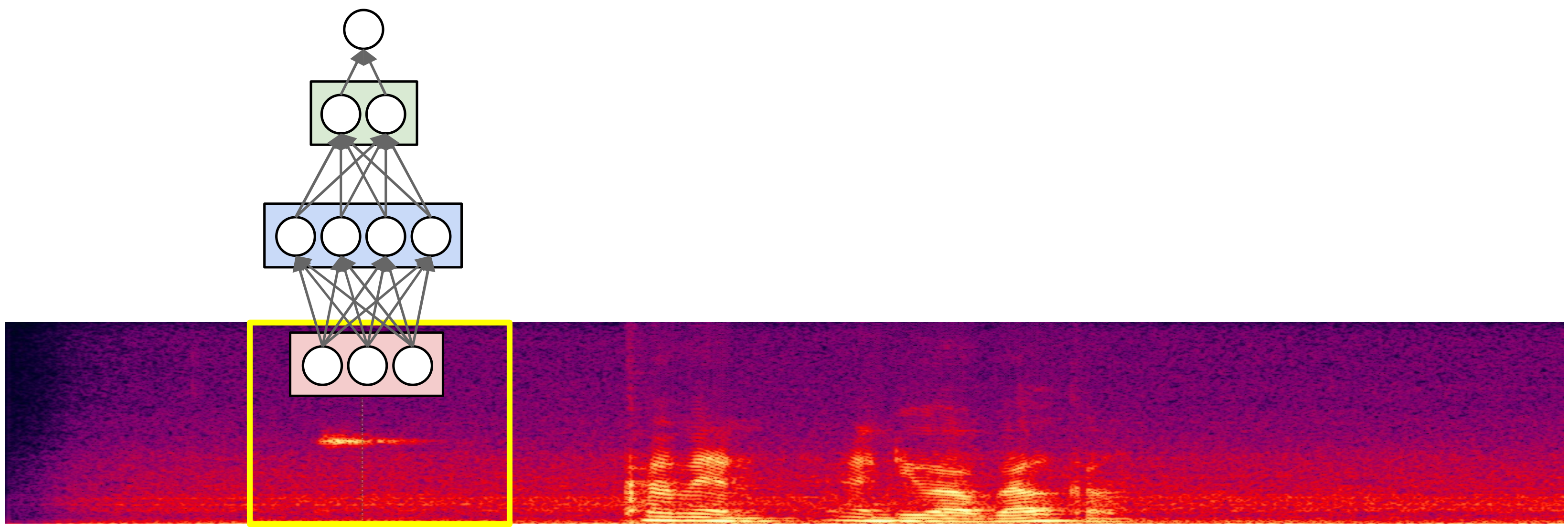
- Scan for the target word
  - The spectral time-frequency components in a "window" are input to a "welcome-detector" MLP

Solution: Scan



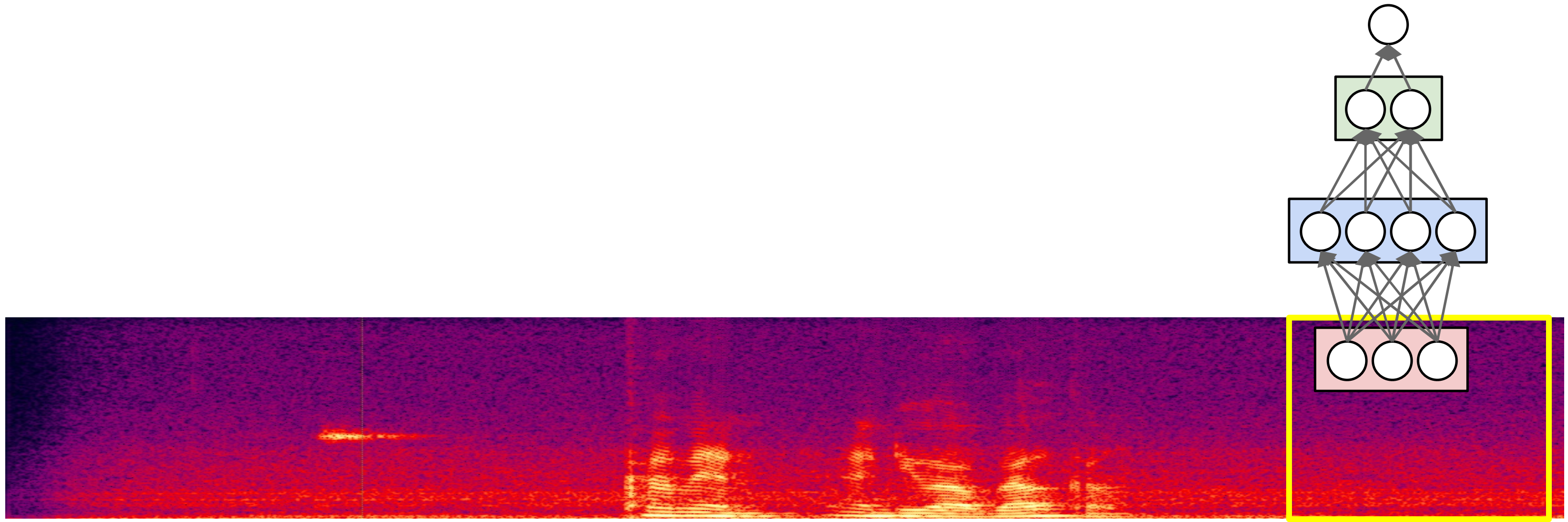
- Scan for the target word
  - The spectral time-frequency components in a "window" are input to a "welcome-detector" MLP

Solution: Scan



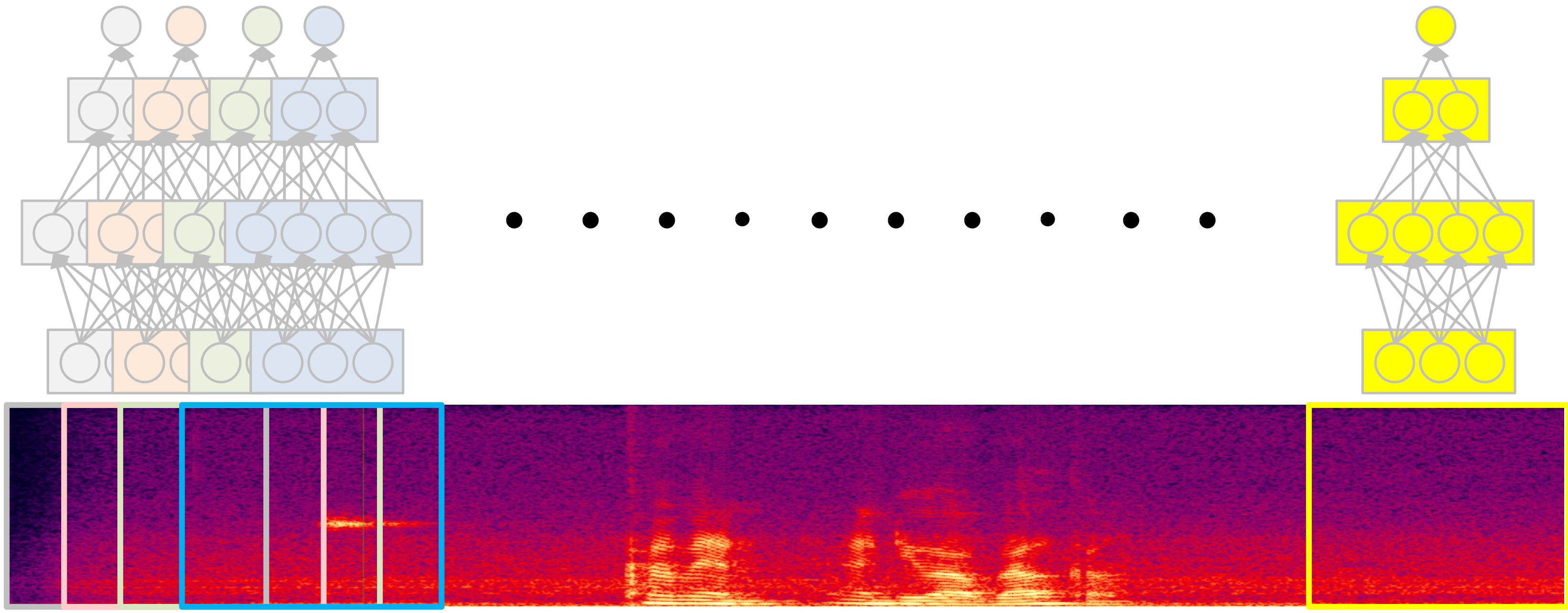
- Scan for the target word
  - The spectral time-frequency components in a "window" are input to a "welcome-detector" MLP

Solution: Scan



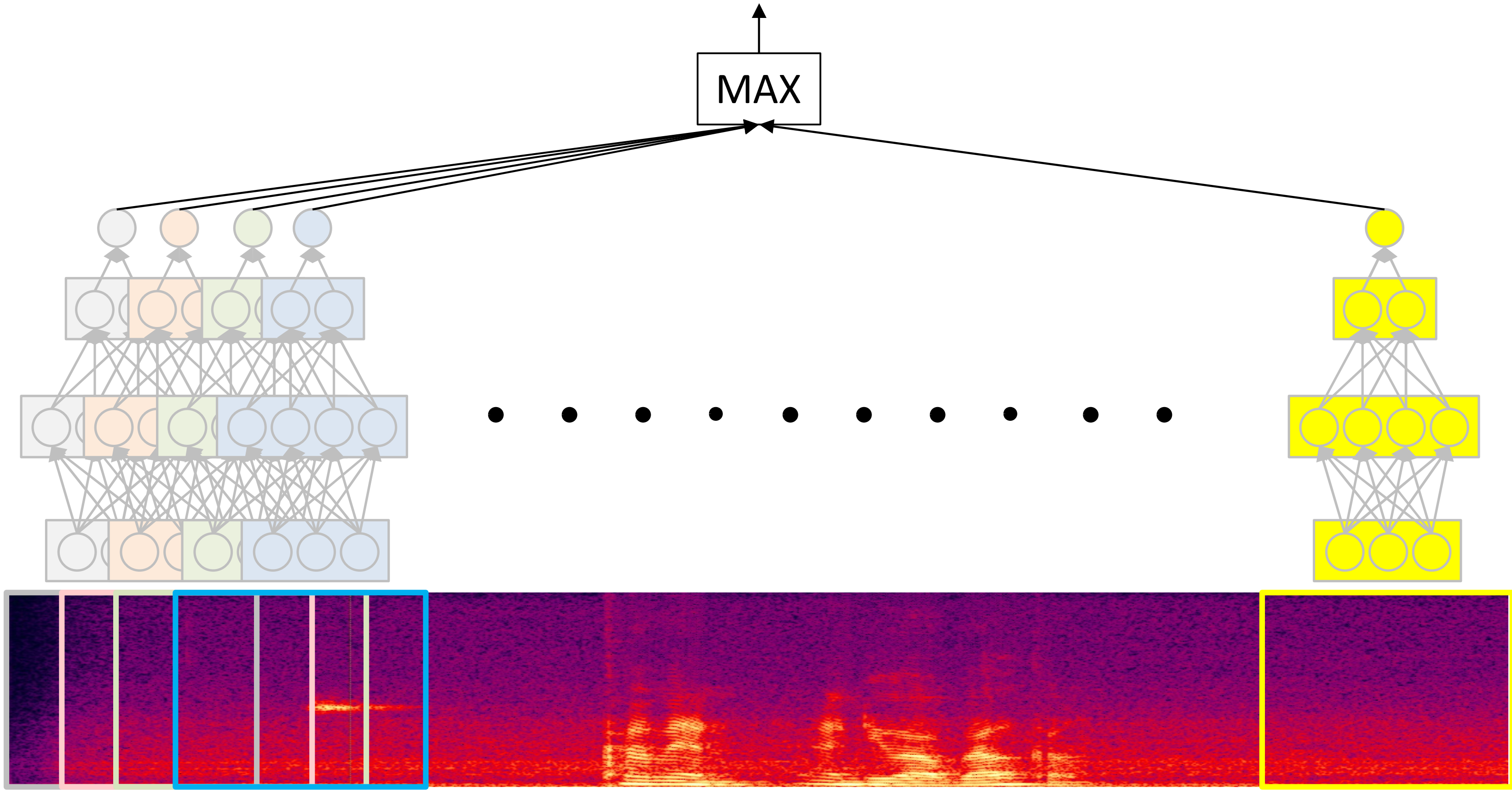
- Scan for the target word
  - The spectral time-frequency components in a "window" are input to a "welcome-detector" MLP

Solution: Scan



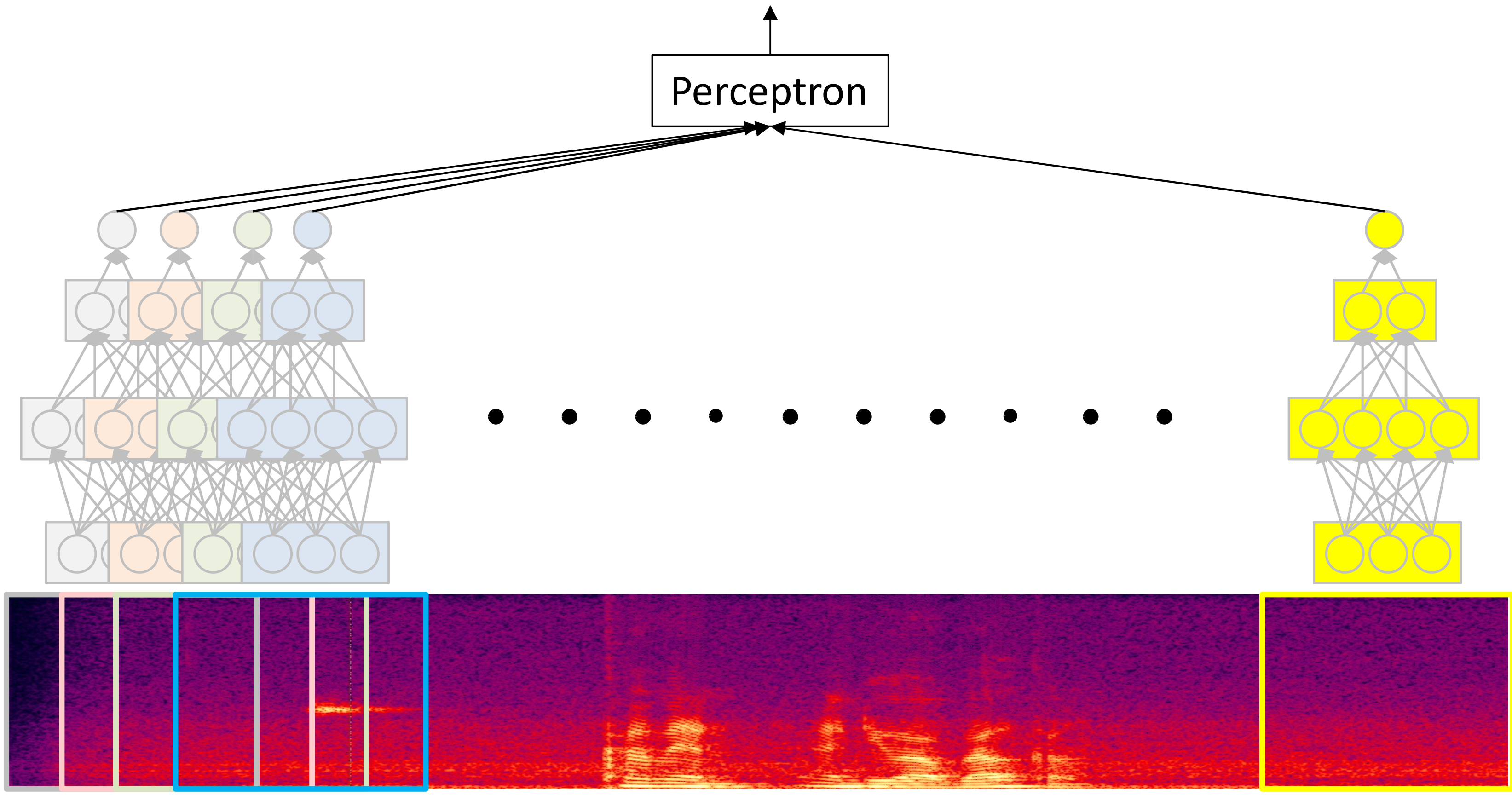
- "Does welcome occur in this recording?"
  - We have classified many "windows" individually
  - "Welcome" may have occurred in any of them

Solution: Scan



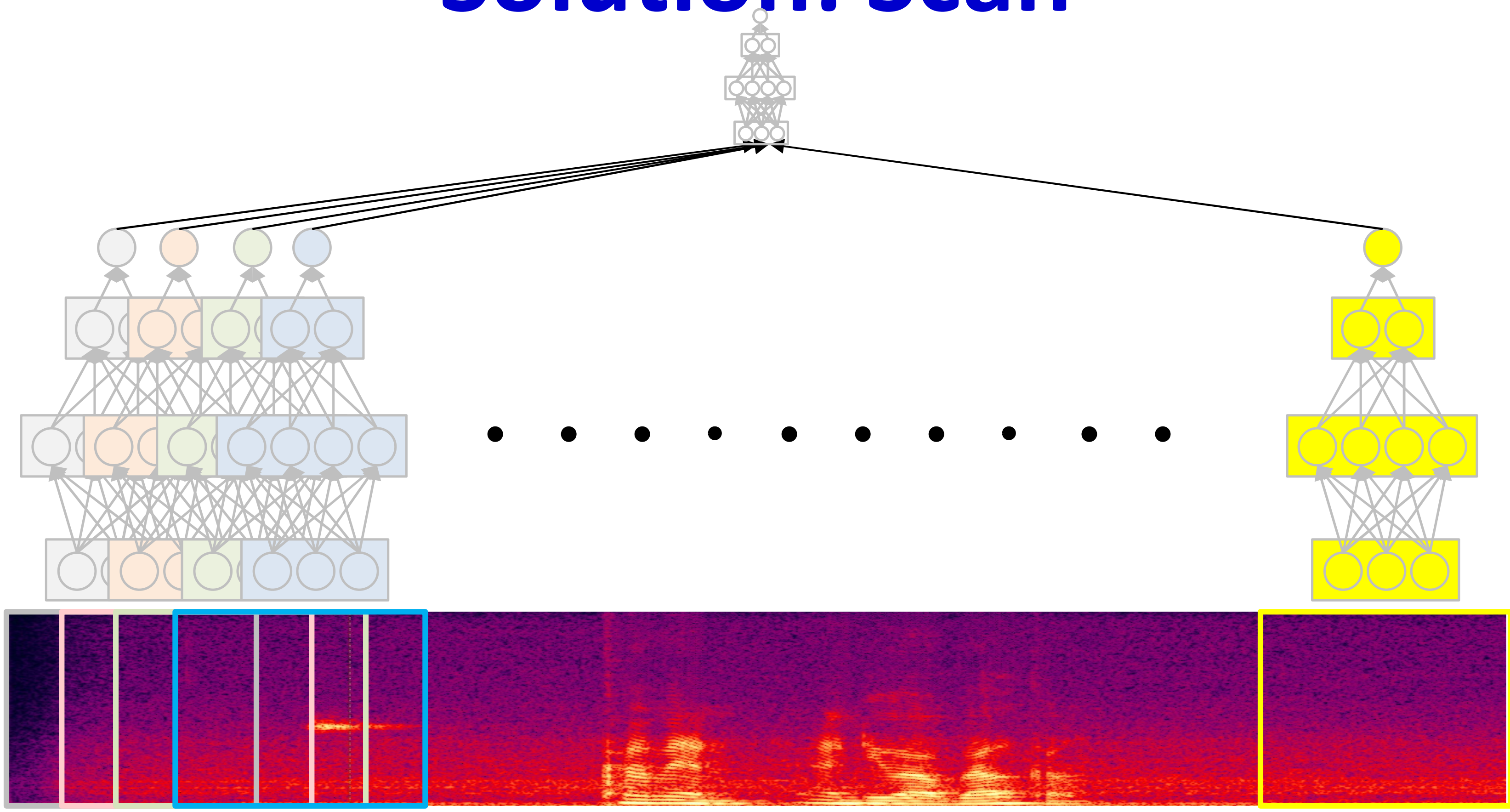
- "Does welcome occur in this recording?"
  - Maximum of all the outputs (Equivalent of Boolean OR)

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



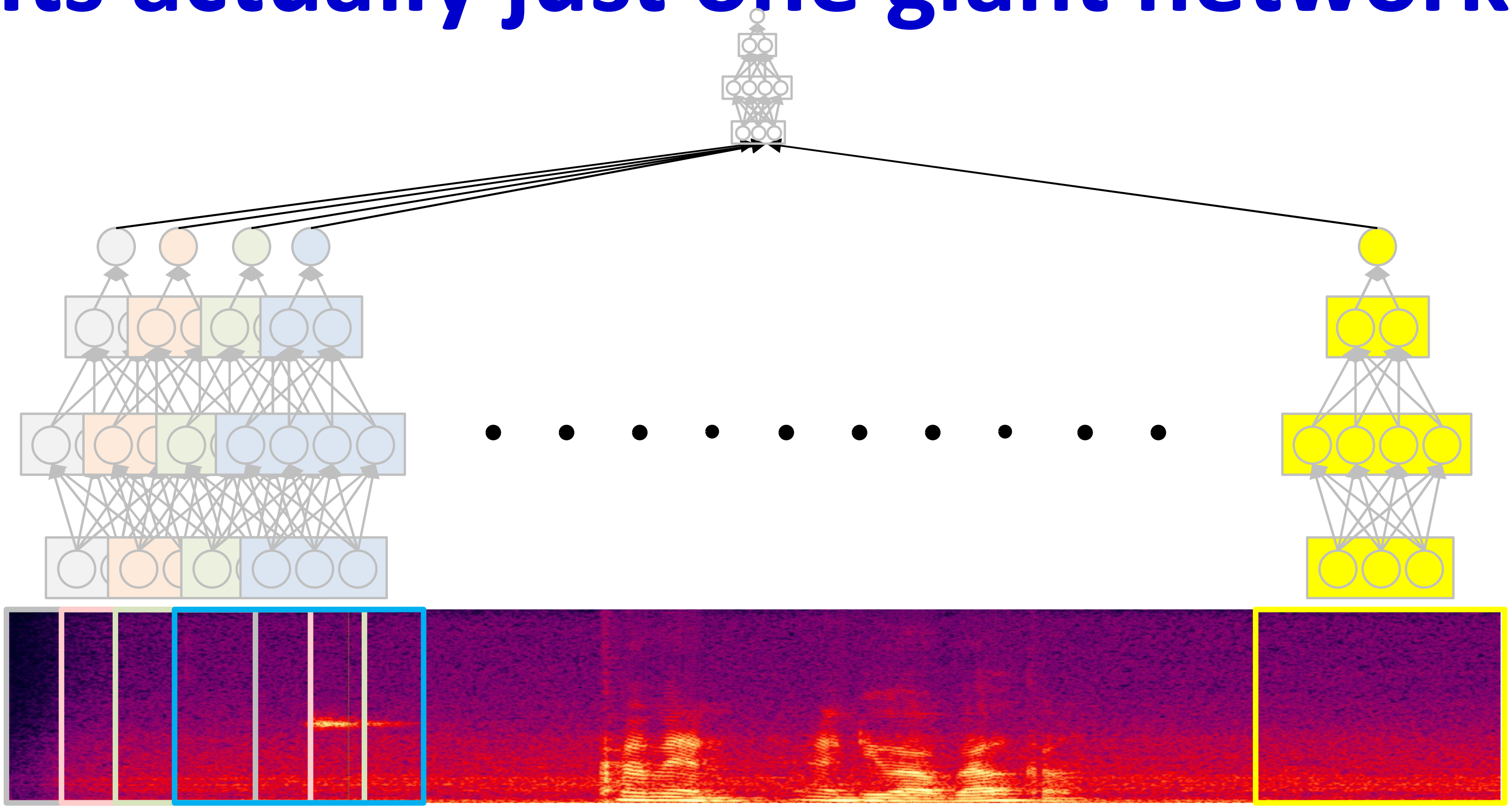
- "Does welcome occur in this recording?"
  - Maximum of all the outputs (Equivalent of Boolean OR)
  - Or a proper softmax/logistic
    - Adjacent windows can combine their evidence
  - Or even an MLP

Scanning with an MLP

- `K` = width of "patch" evaluated by MLP

```
For t = 1:T-K+1
   XSegment = x(:, t:t+K-1)
   y(t) = MLP(XSegment)

Y = softmax(y(1)..y(T-K+1))
```

Its actually just one giant network



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

Scanning with an MLP

- `K` = width of "patch" evaluated by MLP

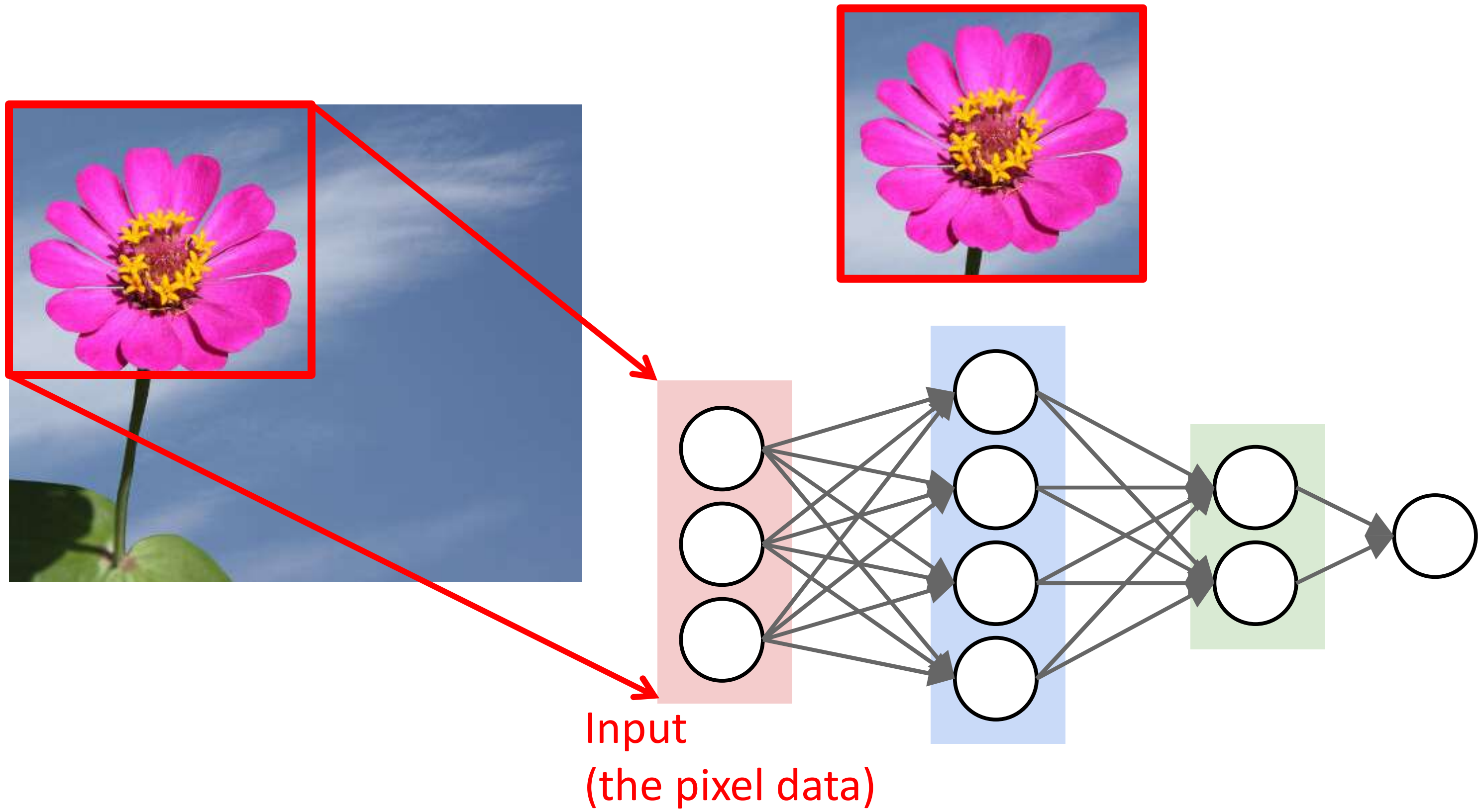
```
For t = 1:T-K+1
   XSegment = x(:, t:t+K-1)
   y(t) = MLP(XSegment)

Y = softmax(y(1)..y(T-K+1))
```

Just the final layer of the overall MLP

Scanning with an MLP

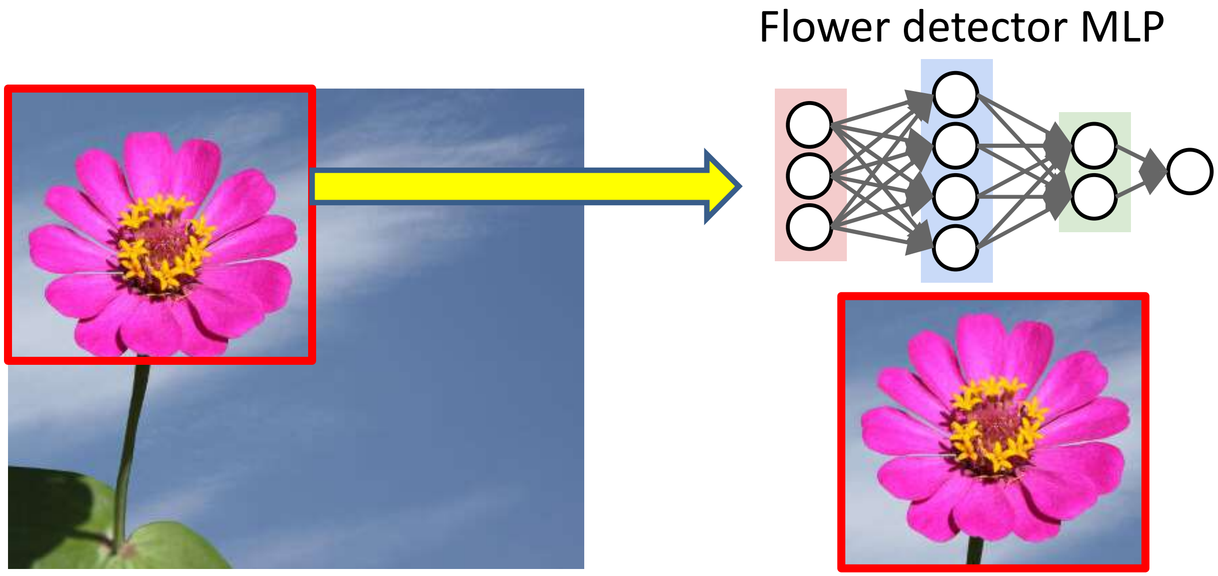
```
Y = giantMLP(x)
```

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



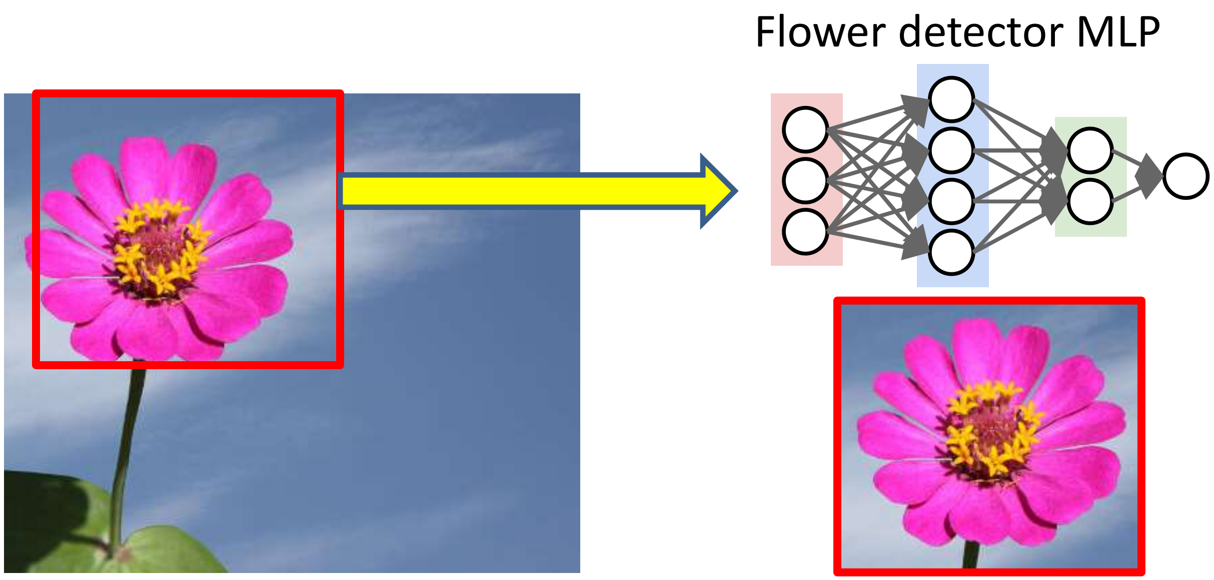
- *Scan* for the desired object
  – “Look” for the target object at each position

Solution: Scan



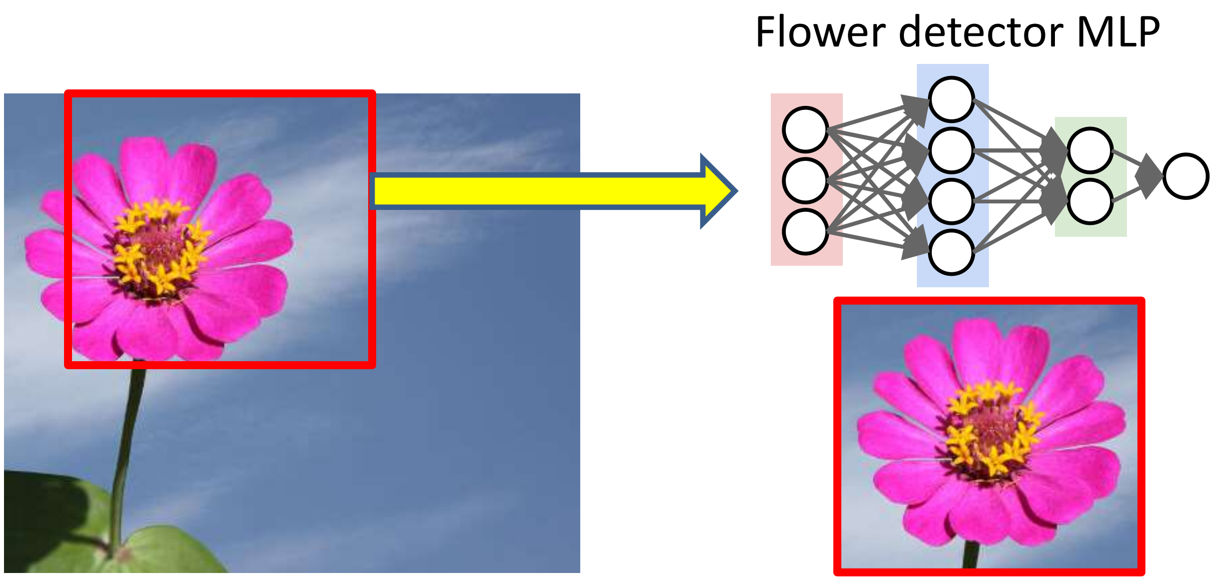
- *Scan* for the desired object
- At each location, the entire region is sent through the MLP

Solution: Scan



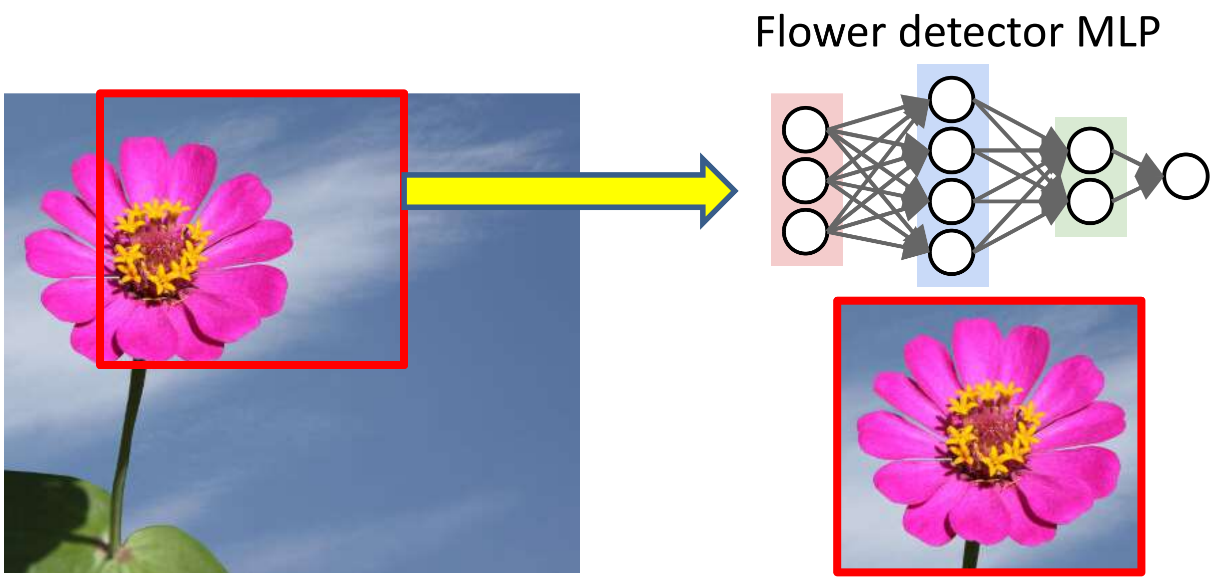
- *Scan* for the desired object
- At each location, the entire region is sent through the MLP

Solution: Scan



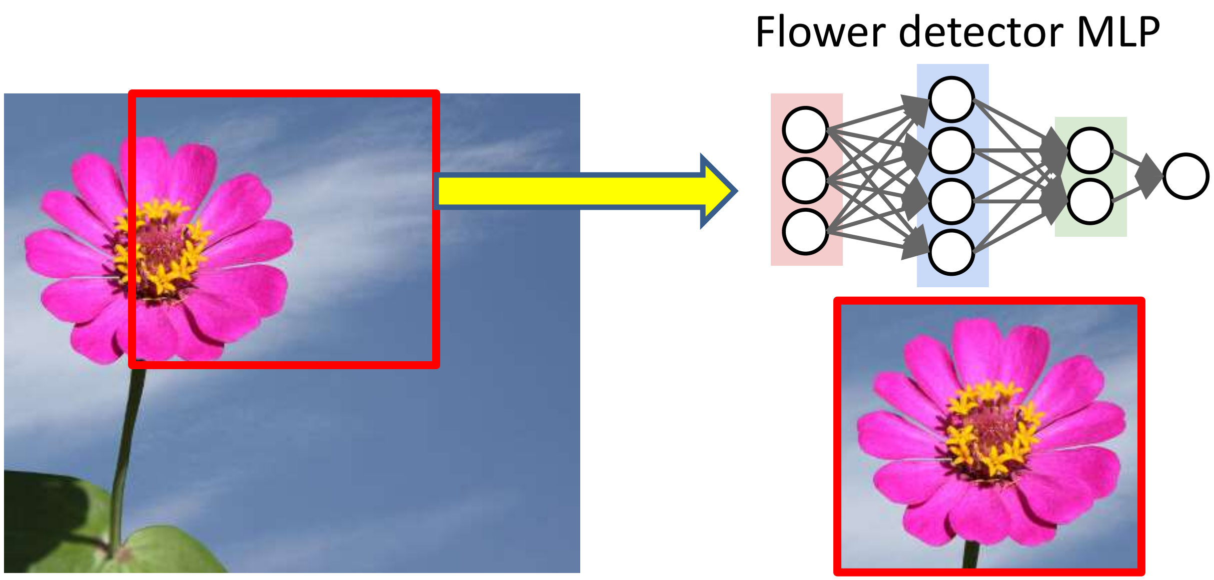
- *Scan* for the desired object
- At each location, the entire region is sent through the MLP

Solution: Scan



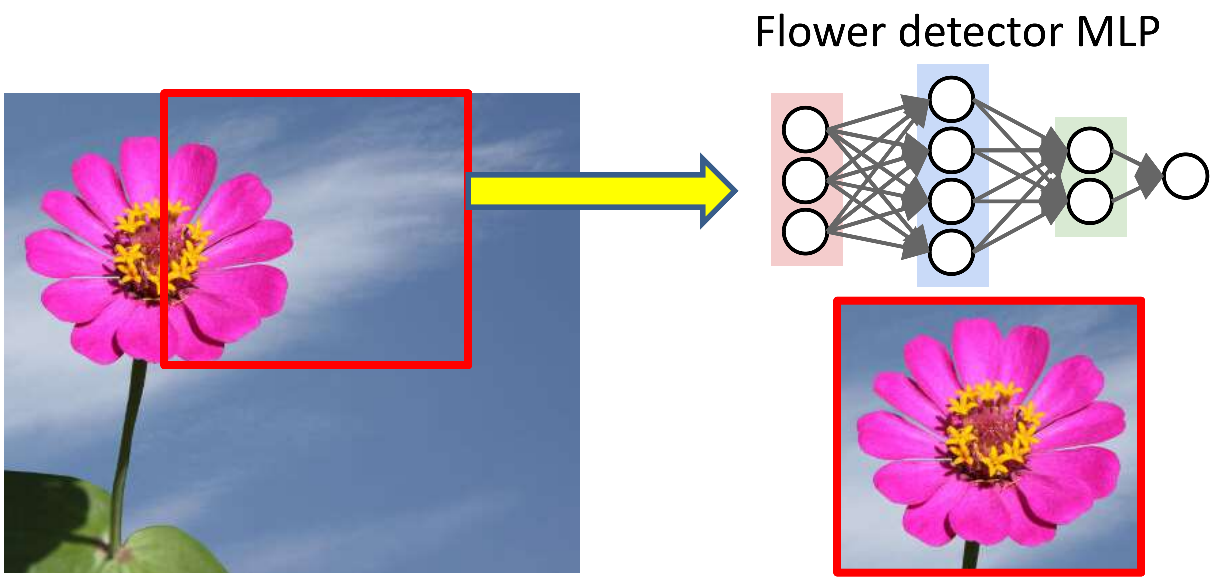
- *Scan* for the desired object
- At each location, the entire region is sent through the MLP

Solution: Scan



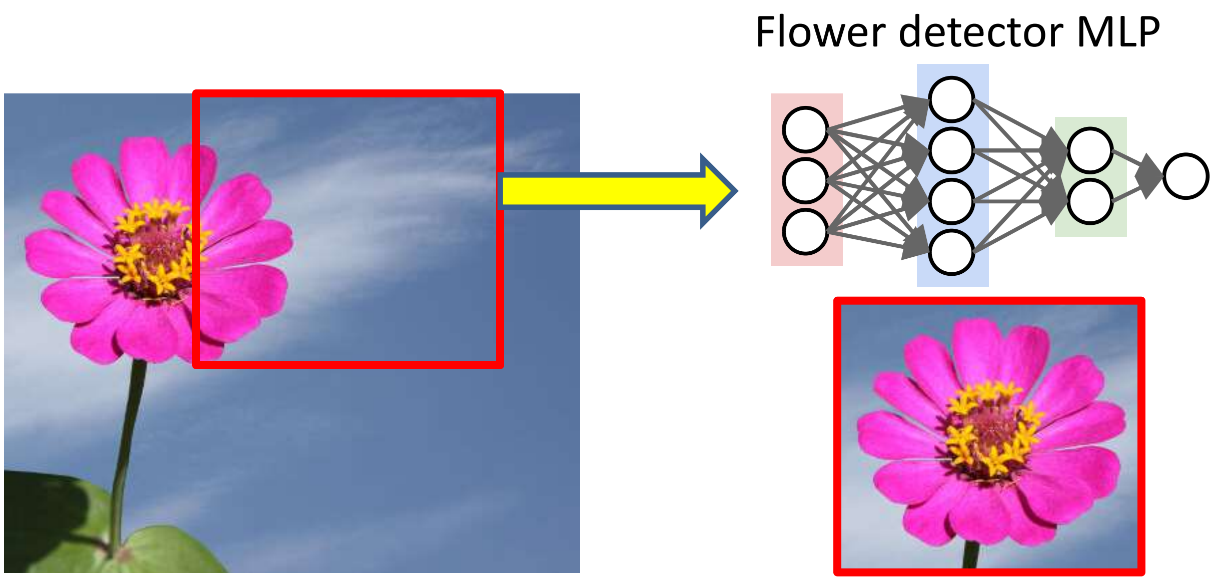
- *Scan* for the desired object
- At each location, the entire region is sent through the MLP

Solution: Scan



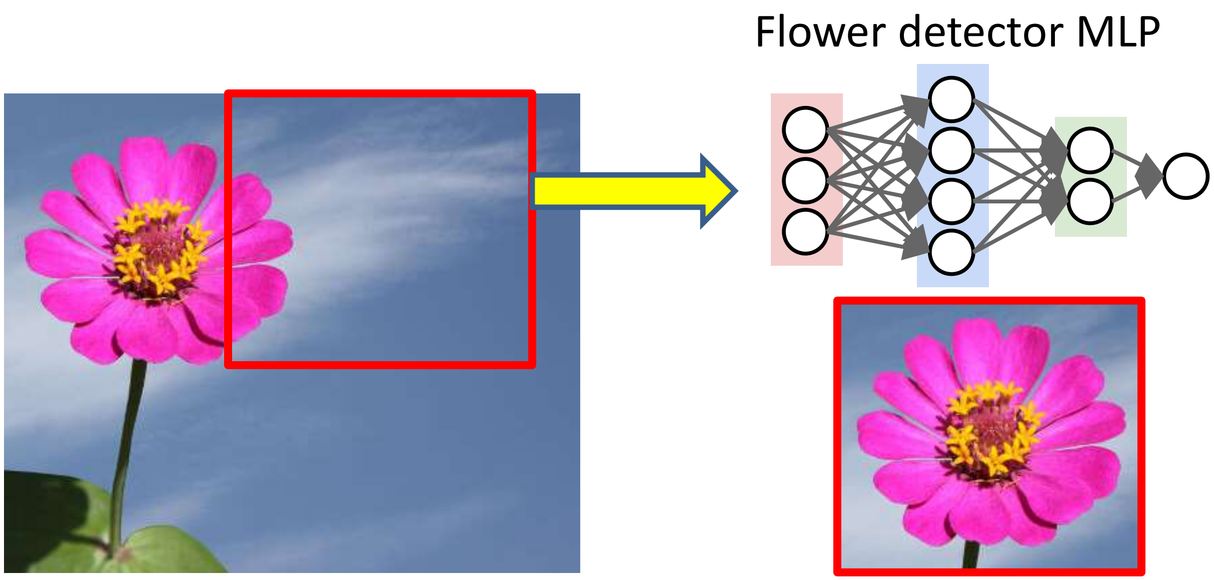
- *Scan* for the desired object
- At each location, the entire region is sent through the MLP

Solution: Scan



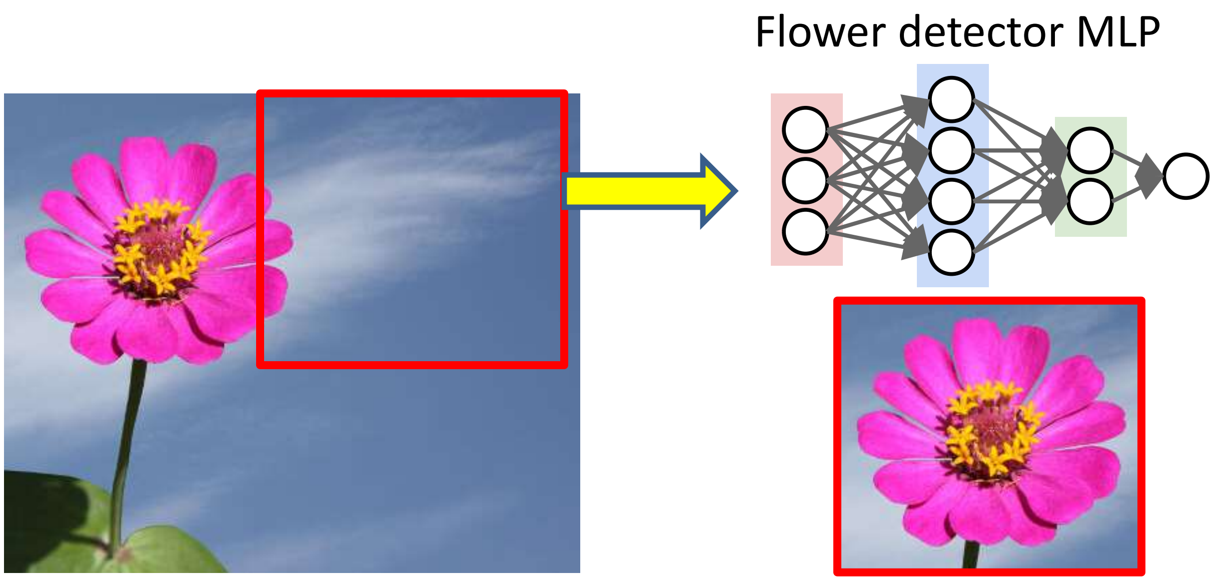
- *Scan* for the desired object
- At each location, the entire region is sent through the MLP

Solution: Scan



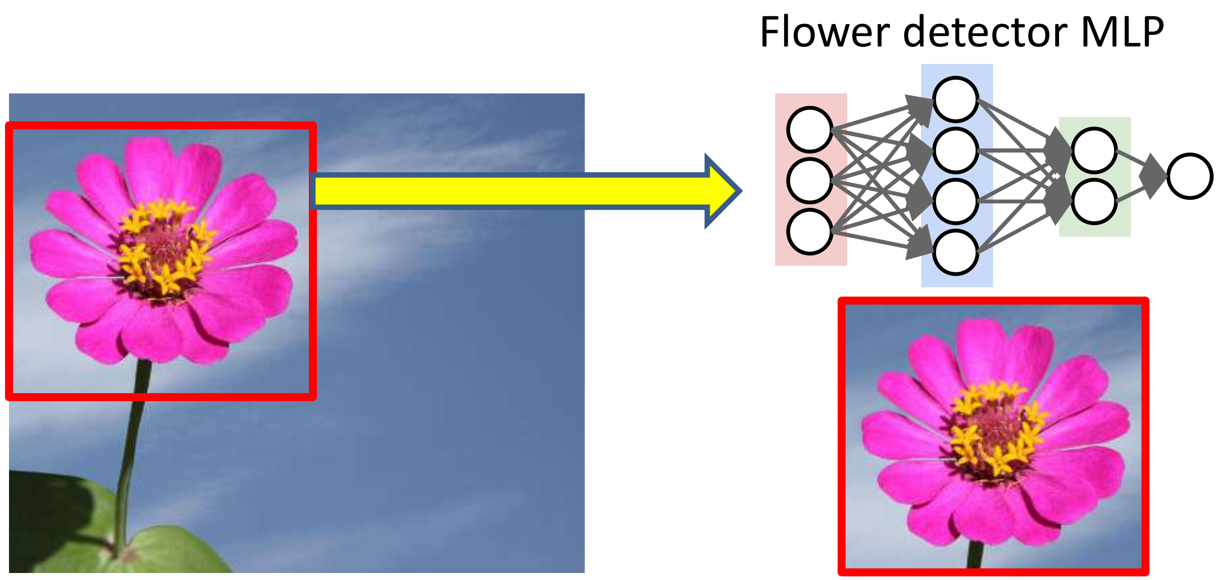
- *Scan* for the desired object
- At each location, the entire region is sent through the MLP

Solution: Scan



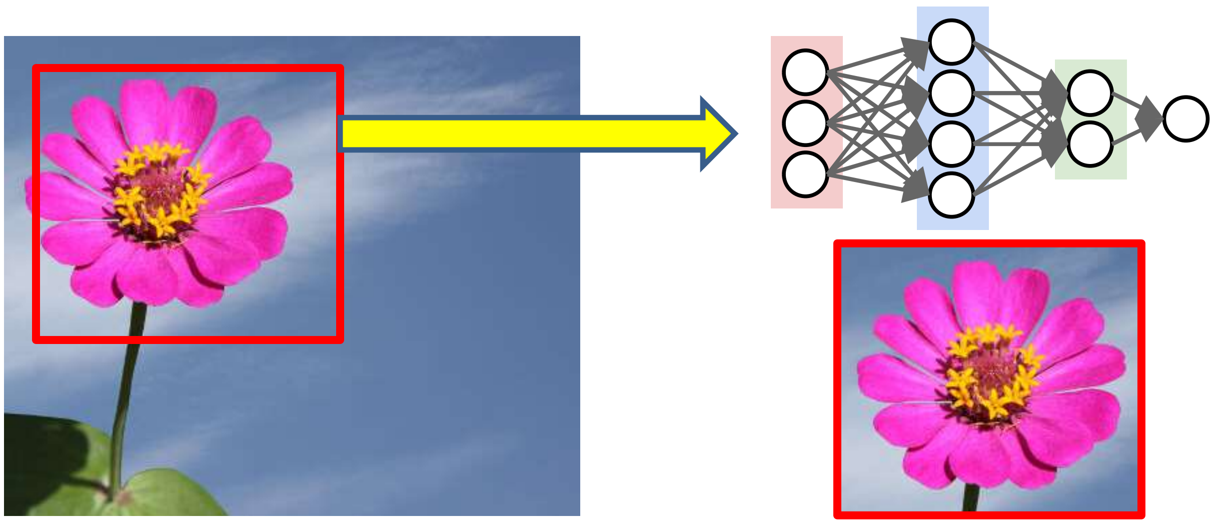
- *Scan* for the desired object
- At each location, the entire region is sent through the MLP

Solution: Scan



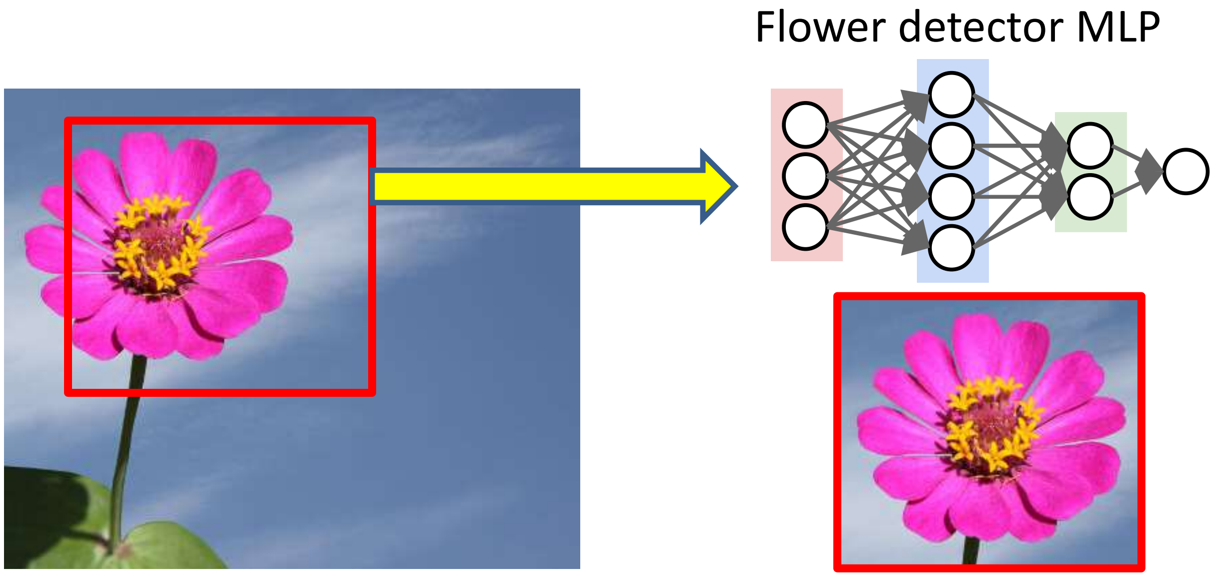
- *Scan* for the desired object
- At each location, the entire region is sent through the MLP

Solution: Scan



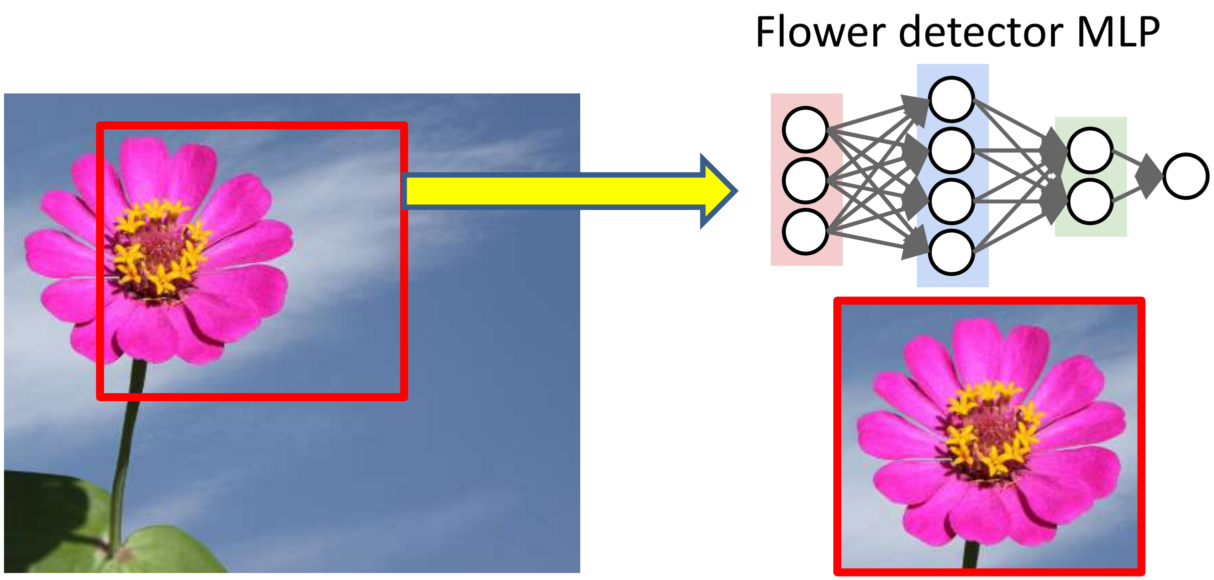
- *Scan* for the desired object
- At each location, the entire region is sent through the MLP

Solution: Scan



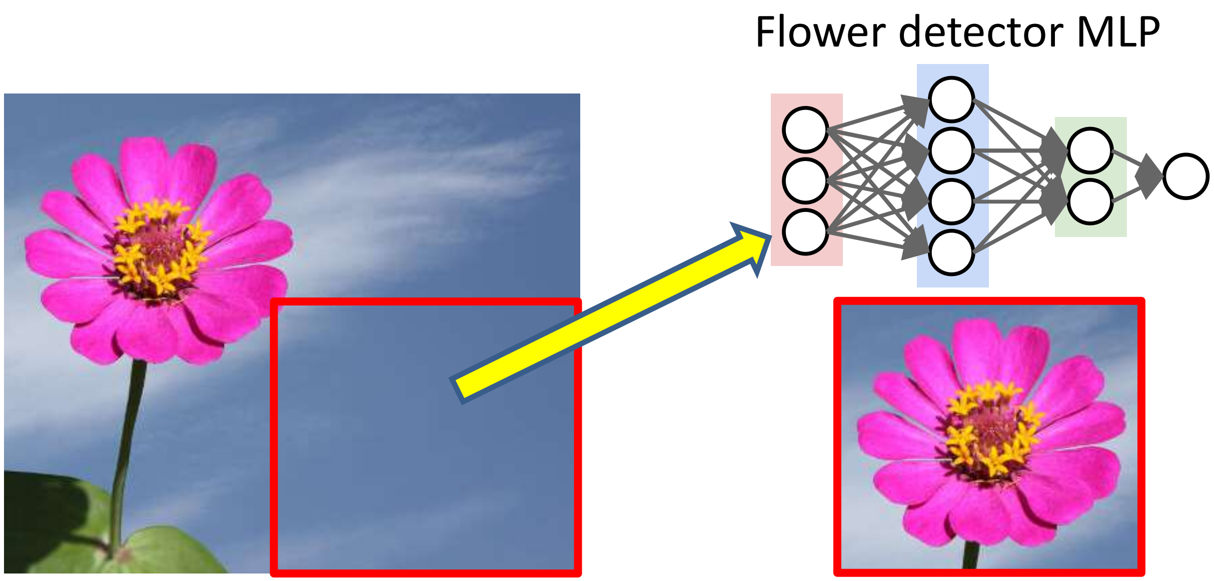
- *Scan* for the desired object
- At each location, the entire region is sent through the MLP

Solution: Scan



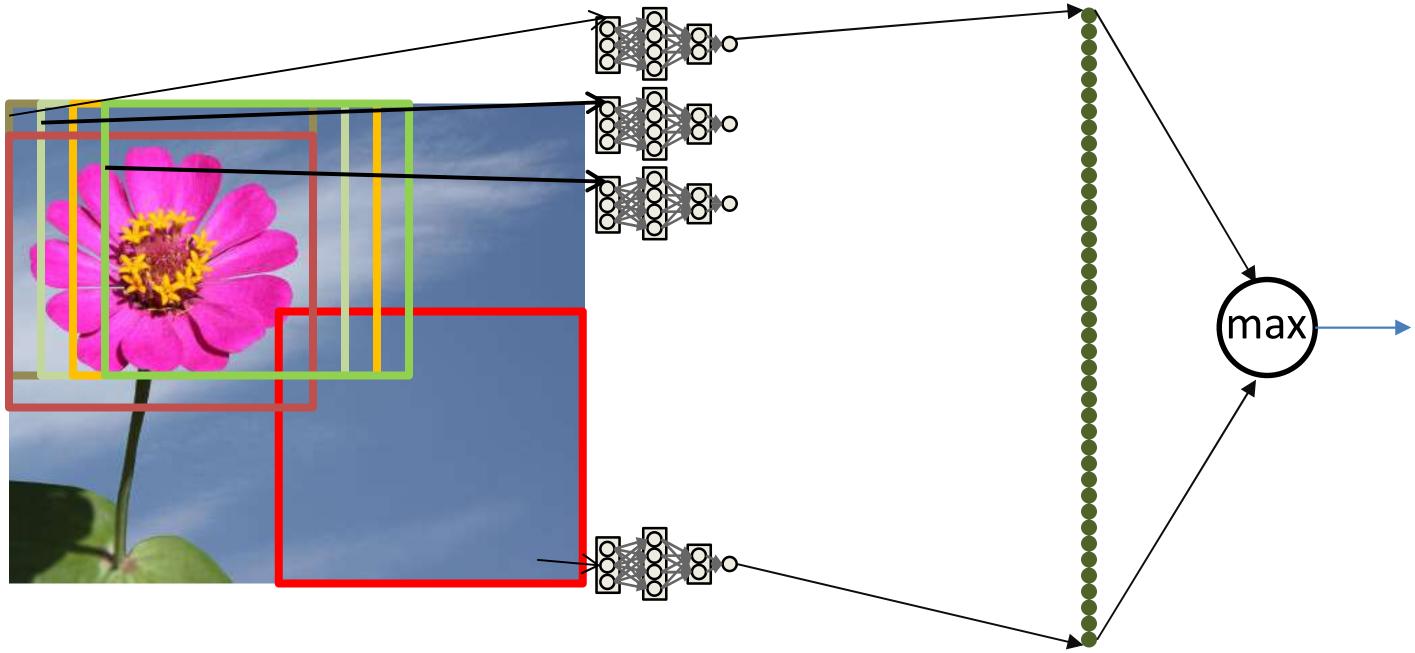
- *Scan* for the desired object
- At each location, the entire region is sent through the MLP

Solution: Scan



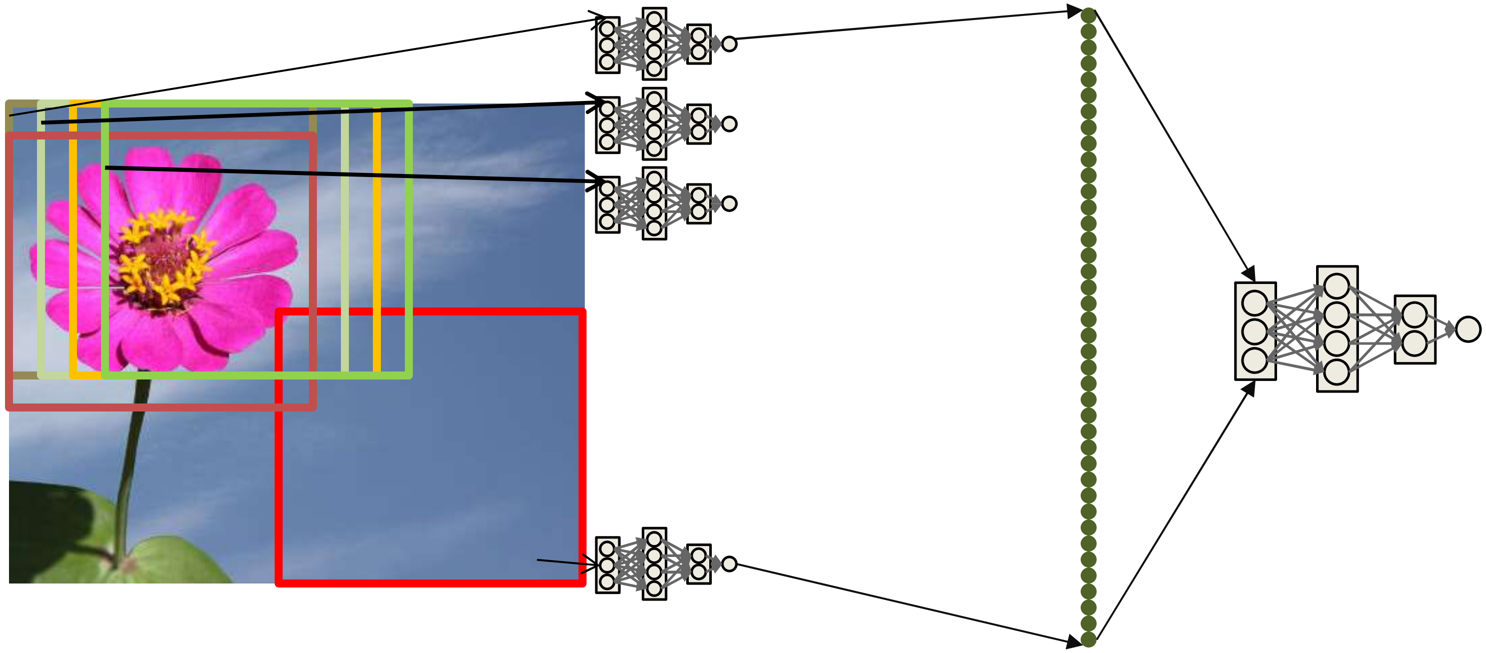
- *Scan* for the desired object
- At each location, the entire region is sent through the MLP

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
    - If the picture has a flower, the location with the flower will result in high output value

Scanning the picture to find a flower



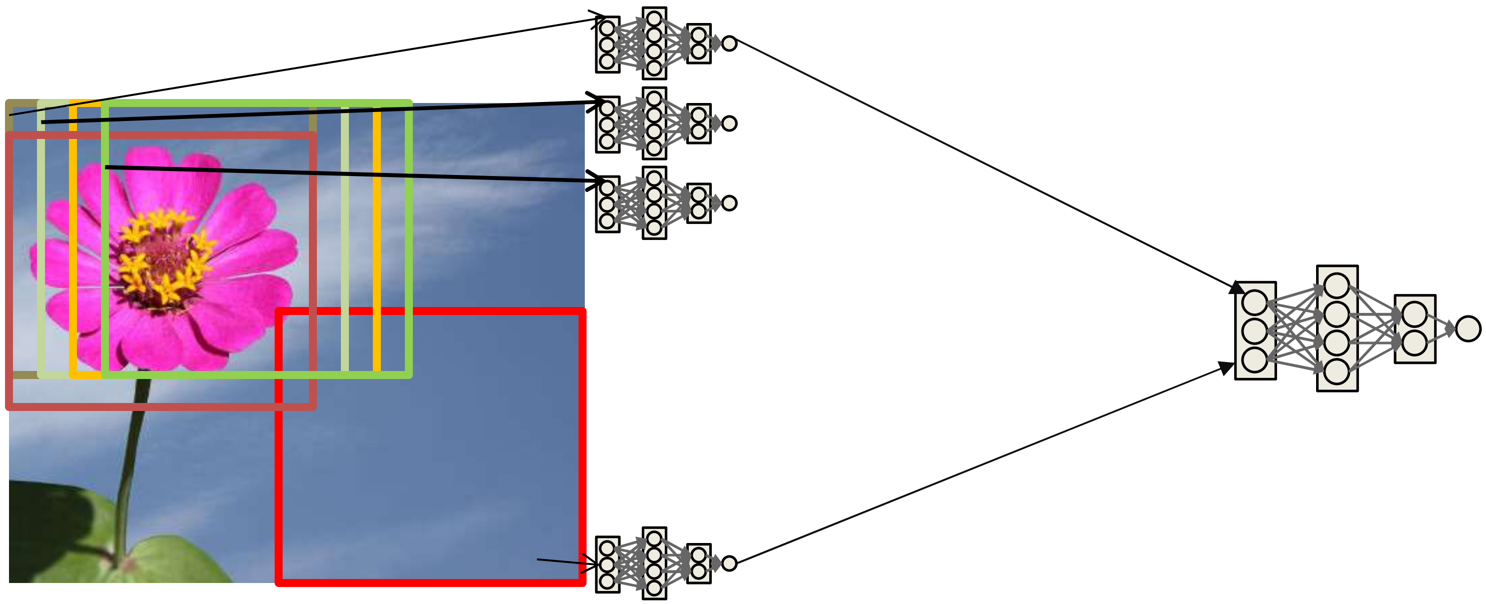
- Determine if any of the locations had a flower
    - Each dot in the right represents the output of the MLP when it classifies one location in the input figure
        - The score output by the MLP
  - Look at the maximum value
  - Or pass it through a softmax or even an MLP

Scanning with an MLP

- `KxK` = size of "patch" evaluated by MLP
- `W` is width of image
- `H` is height of image

```
For i = 1:W-K+1
    For j = 1:H-K+1
        ImgSegment = Img(i:i+K-1, j:j+K-1)
        y(i,j) = MLP(ImgSegment)

Y = softmax( y(1,1)..y(W-K+1,H-K+1) )
```

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
- The network is *shift invariant*.

Scanning with an MLP

- `KxK` = size of "patch" evaluated by MLP
- `W` is width of image
- `H` is height of image

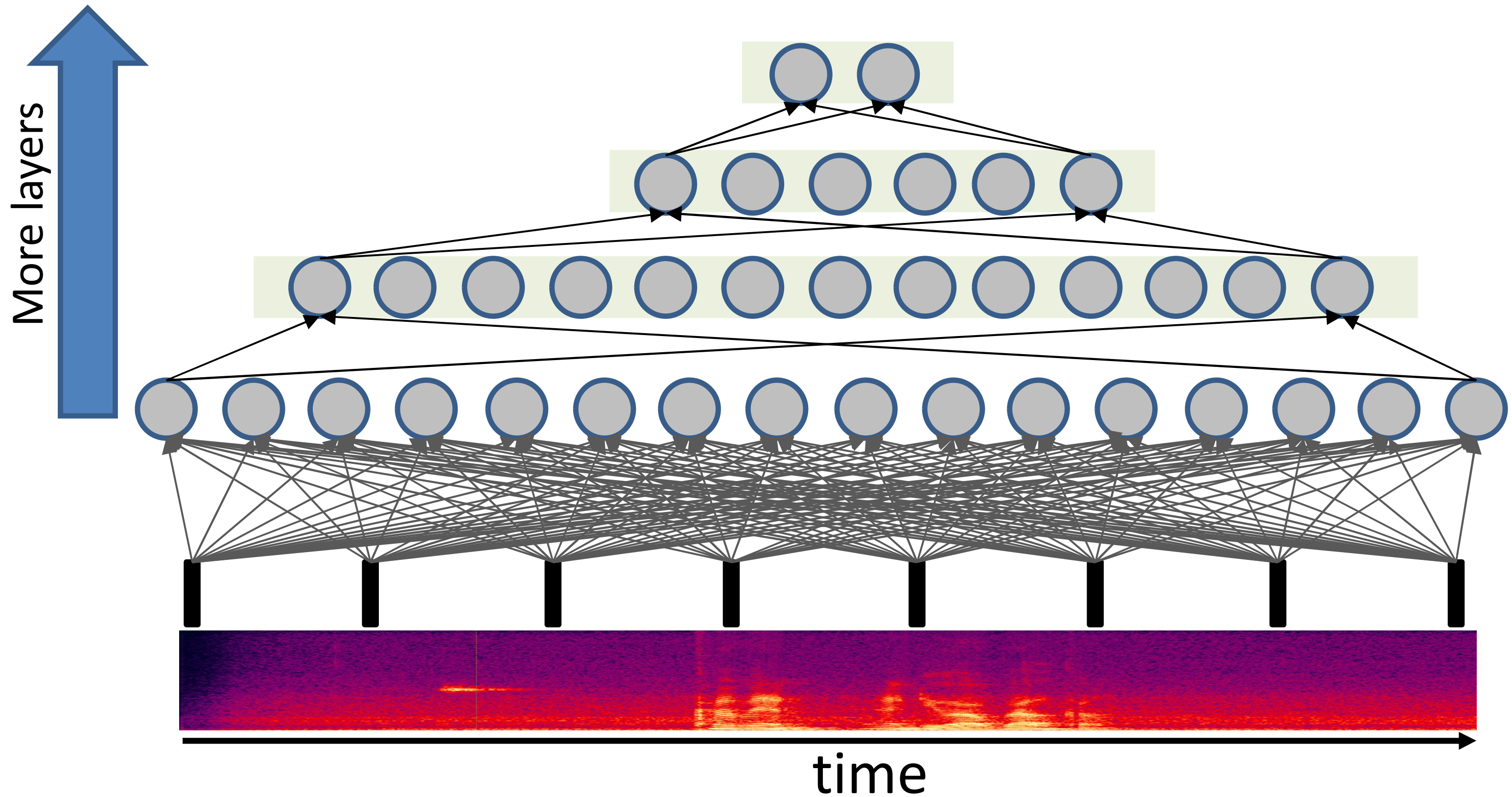
```
For i = 1:W-K+1
    For j = 1:H-K+1
        ImgSegment = Img(i:i+K-1, j:j+K-1)
        y(i,j) = MLP(ImgSegment)

Y = softmax( y(1,1)..y(W-K+1,H-K+1) )
```

Just the final layer of the overall MLP

Scanning with an MLP

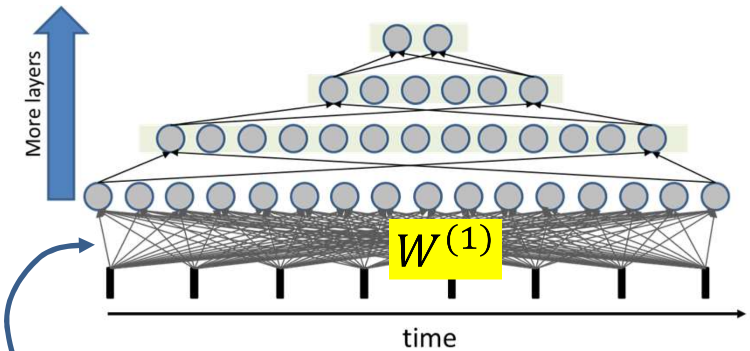
```
Y = giantMLP(img)
```

Regular networks vs. scanning networks



- In a ***regular MLP*** every neuron in a layer is connected by a unique weight to every unit in the previous layer
  – All entries in the weight matrix are unique
  – The weight matrix is (generally) full

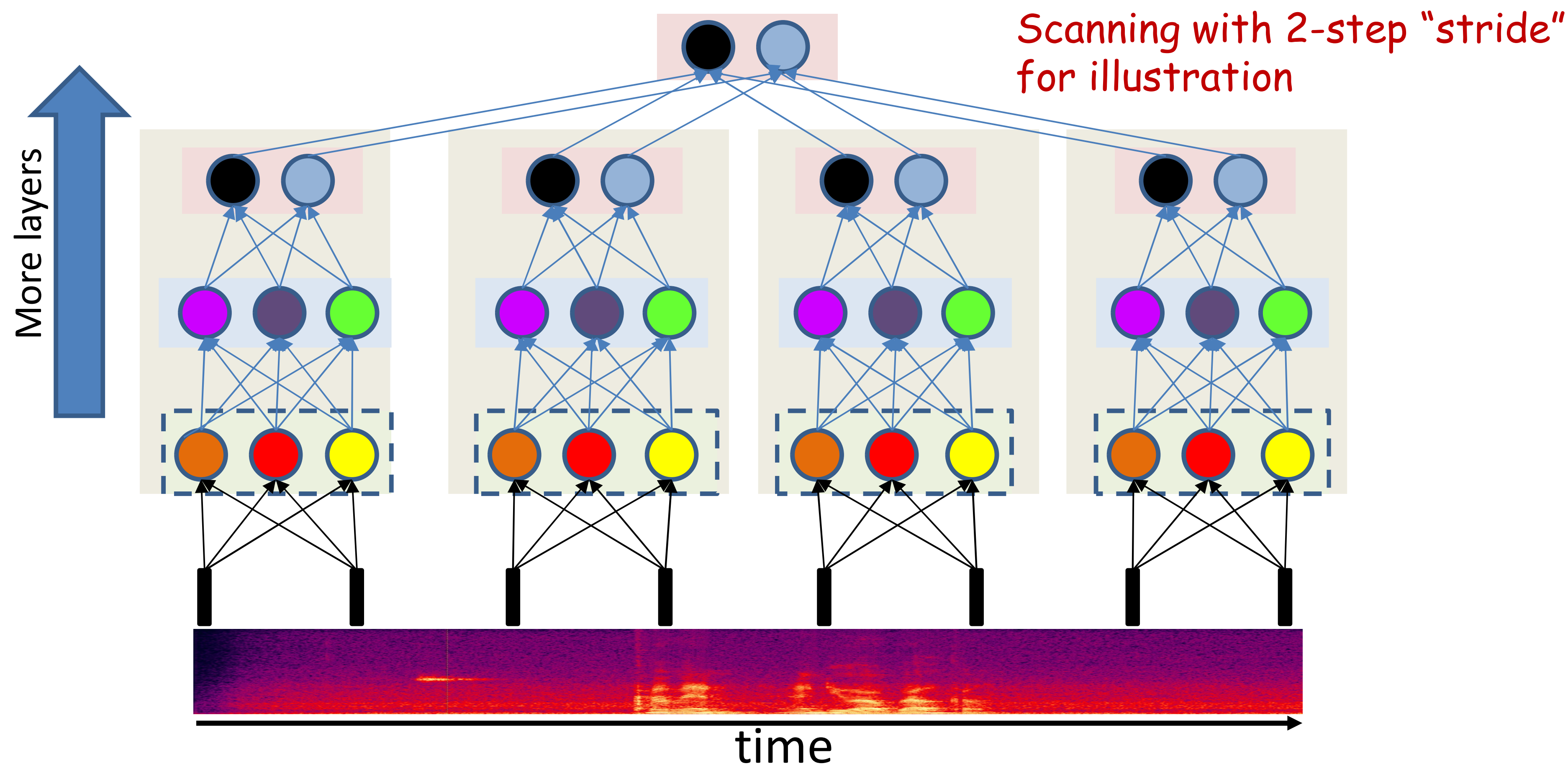
Regular network



$$W^{(1)} = \begin{bmatrix} w_{11} & w_{21} & w_{31} & w_{41} & \cdots & \cdots & \cdots & w_{N1} \\ w_{12} & w_{22} & w_{32} & w_{42} & \cdots & \cdots & \cdots & w_{N2} \\ w_{13} & w_{23} & w_{33} & w_{43} & \cdots & \cdots & \cdots & w_{N3} \\ w_{14} & w_{24} & w_{34} & w_{44} & \cdots & \cdots & \cdots & w_{N4} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{1M} & w_{2M} & w_{3M} & w_{4M} & \cdots & \cdots & \cdots & w_{NM} \end{bmatrix}$$

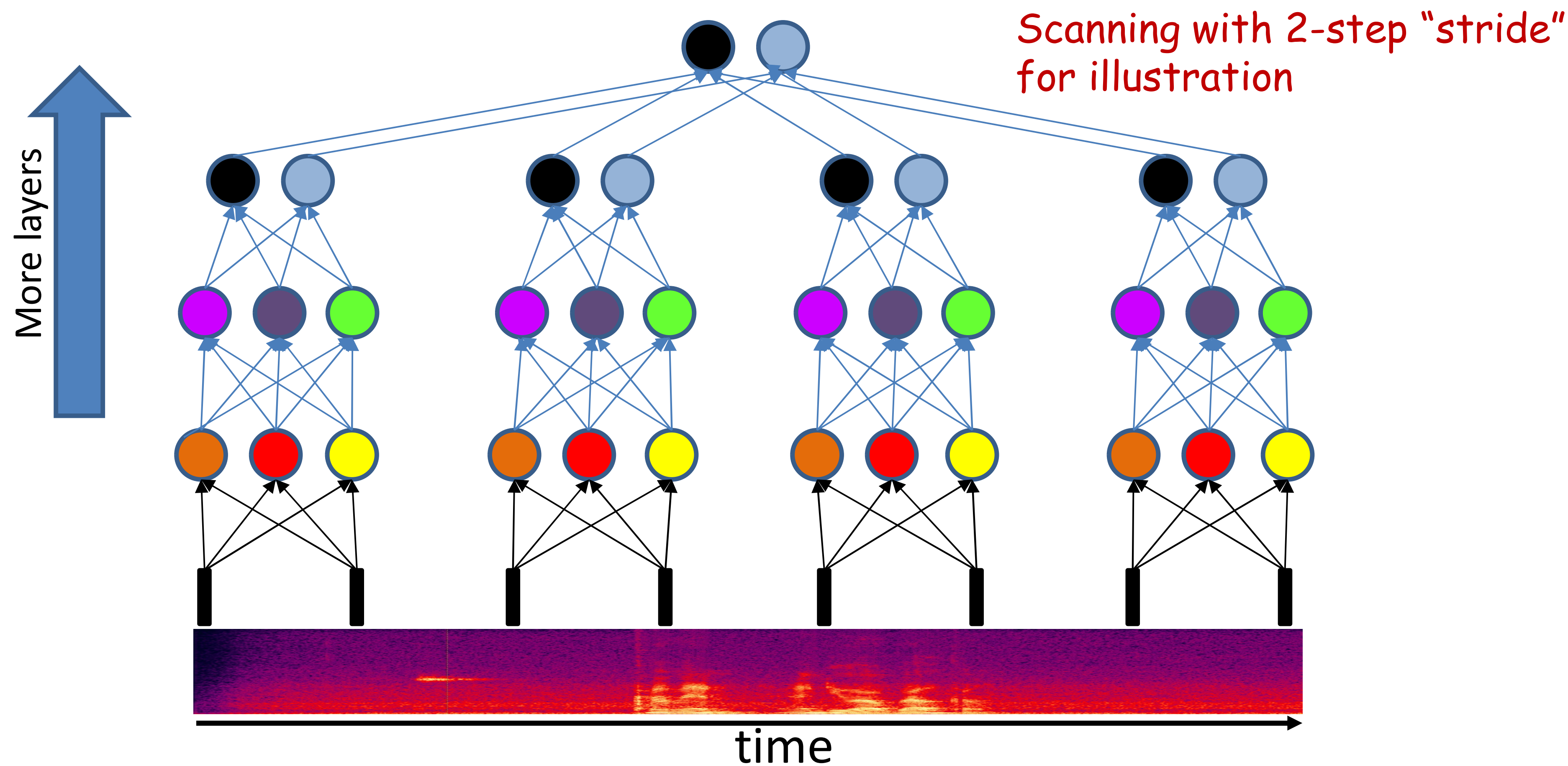
- Consider the first layer
  - Assume N inputs and M outputs
- The weights matrix is a full $M \times N$ matrix
  - Requiring MN unique parameters

Scanning networks



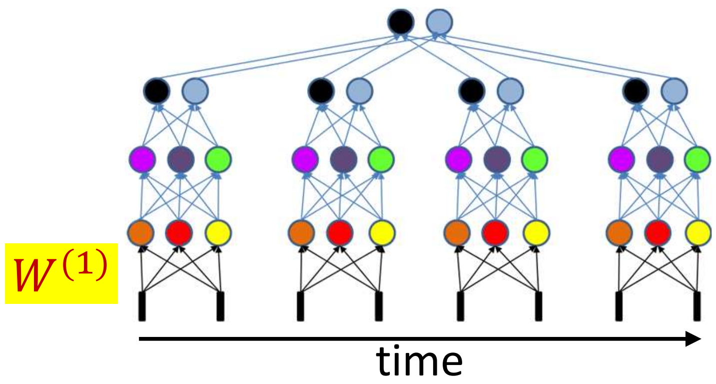
- In a ***scanning MLP*** each neuron is connected to a subset of neurons in the previous layer
  - The weights matrix is sparse
  - The weights matrix is block structured ***with identical blocks***

Scanning networks



- In a ***scanning MLP*** each neuron is connected to a subset of neurons in the previous layer
  - The weights matrix is sparse
  - The weights matrix is block structured ***with identical blocks***

Scanning networks



$$W^{(1)} = \begin{bmatrix} w_{11} & w_{12} & 0 & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ w_{21} & w_{22} & 0 & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ w_{31} & w_{32} & 0 & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & 0 & w_{11} & w_{12} & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & w_{21} & w_{22} & 0 & 0 & 0 & \vdots & \vdots \\ 0 & 0 & w_{31} & w_{32} & 0 & 0 & 0 & \vdots & \vdots \\ 0 & 0 & 0 & 0 & w_{11} & w_{12} & 0 & 0 & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & w_{31} & w_{32} \end{bmatrix}$$

- In a ***scanning MLP*** each neuron is connected to a subset of neurons in the previous layer
  - The weights matrix is sparse
  - The weights matrix is block structured ***with identical blocks***
  - **The network is a *shared parameter* model**

Scanning networks

$W^{(1)}$

$W^{(1)}$

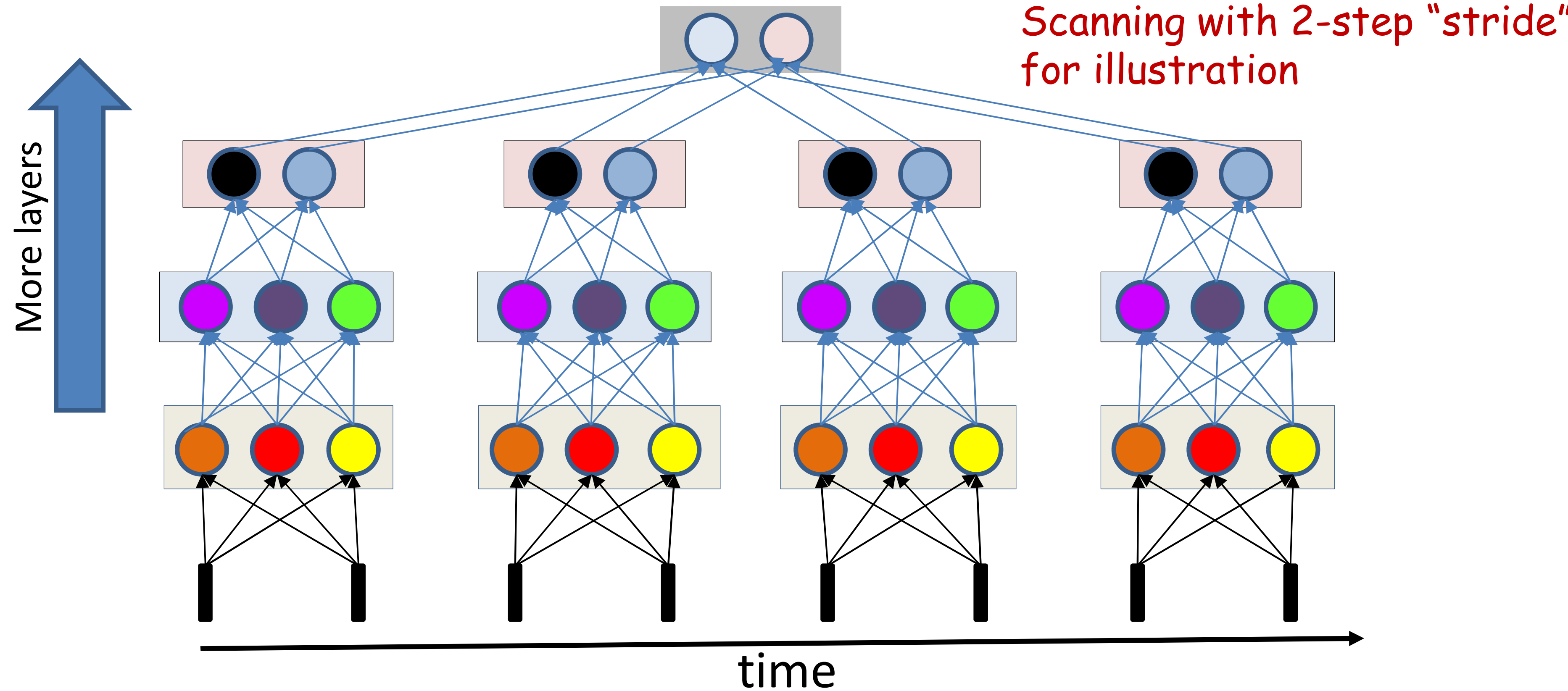
time

$$W^{(1)} = \begin{bmatrix} w_{11} & w_{12} & 0 & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ w_{21} & w_{22} & 0 & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ w_{31} & w_{32} & 0 & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & 0 & w_{11} & w_{12} & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & w_{21} & w_{22} & 0 & 0 & 0 & \vdots & \vdots \\ 0 & 0 & w_{31} & w_{32} & 0 & 0 & 0 & \vdots & \vdots \\ 0 & 0 & 0 & 0 & w_{11} & w_{12} & 0 & 0 & \vdots \end{bmatrix}$$

Effective in any situation where the data are expected to be composed of similar structures at different locations

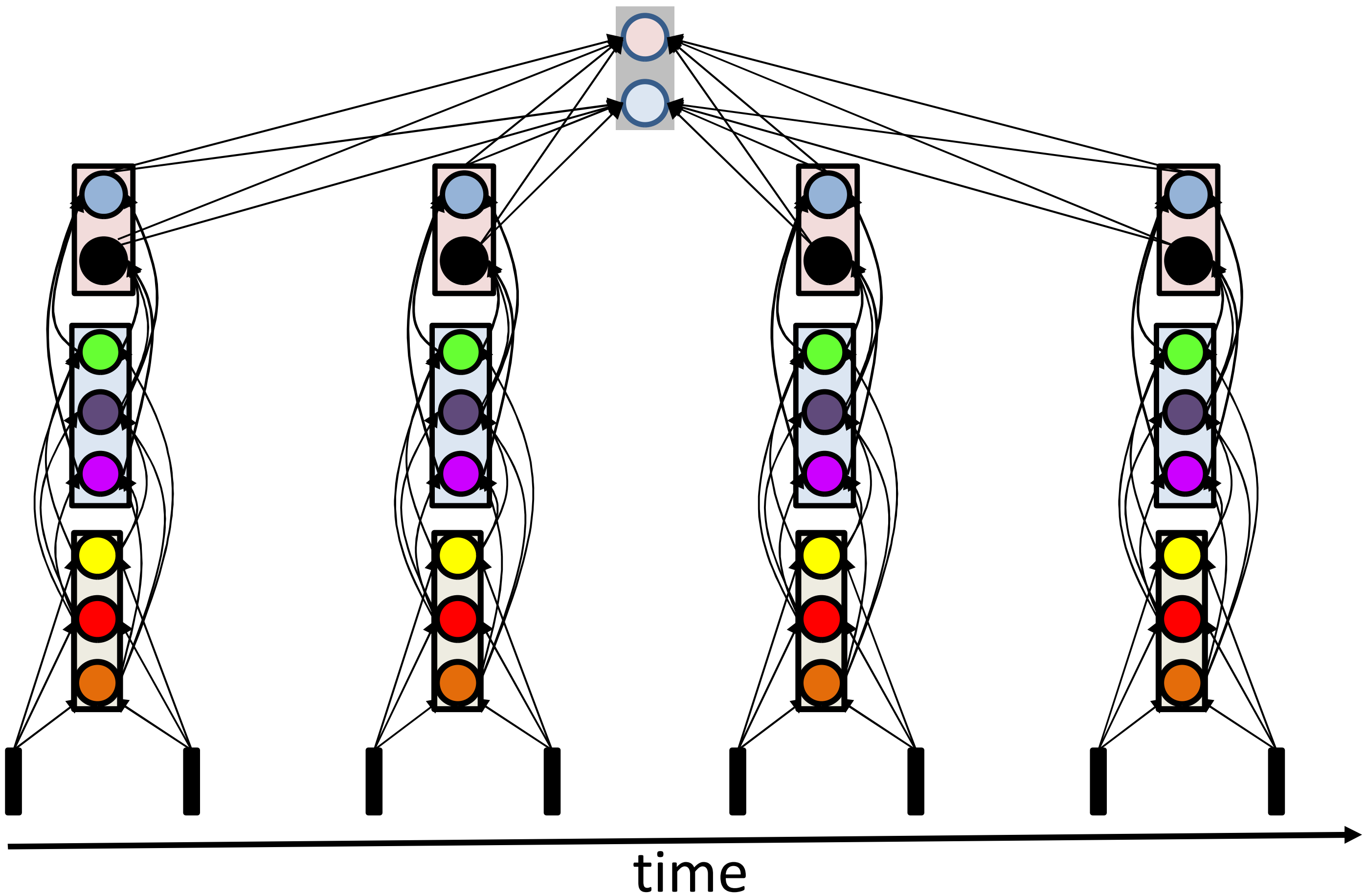
- In a ***scanning MLP*** each neuron is connected to a subset of neurons in the previous layer
  - The weights matrix is sparse
  - The weights matrix is block ***structured with identical blocks***
  - *The network is a shared-parameter model*
- *Also, far fewer parameters (we return to this topic shortly)*

Scanning networks



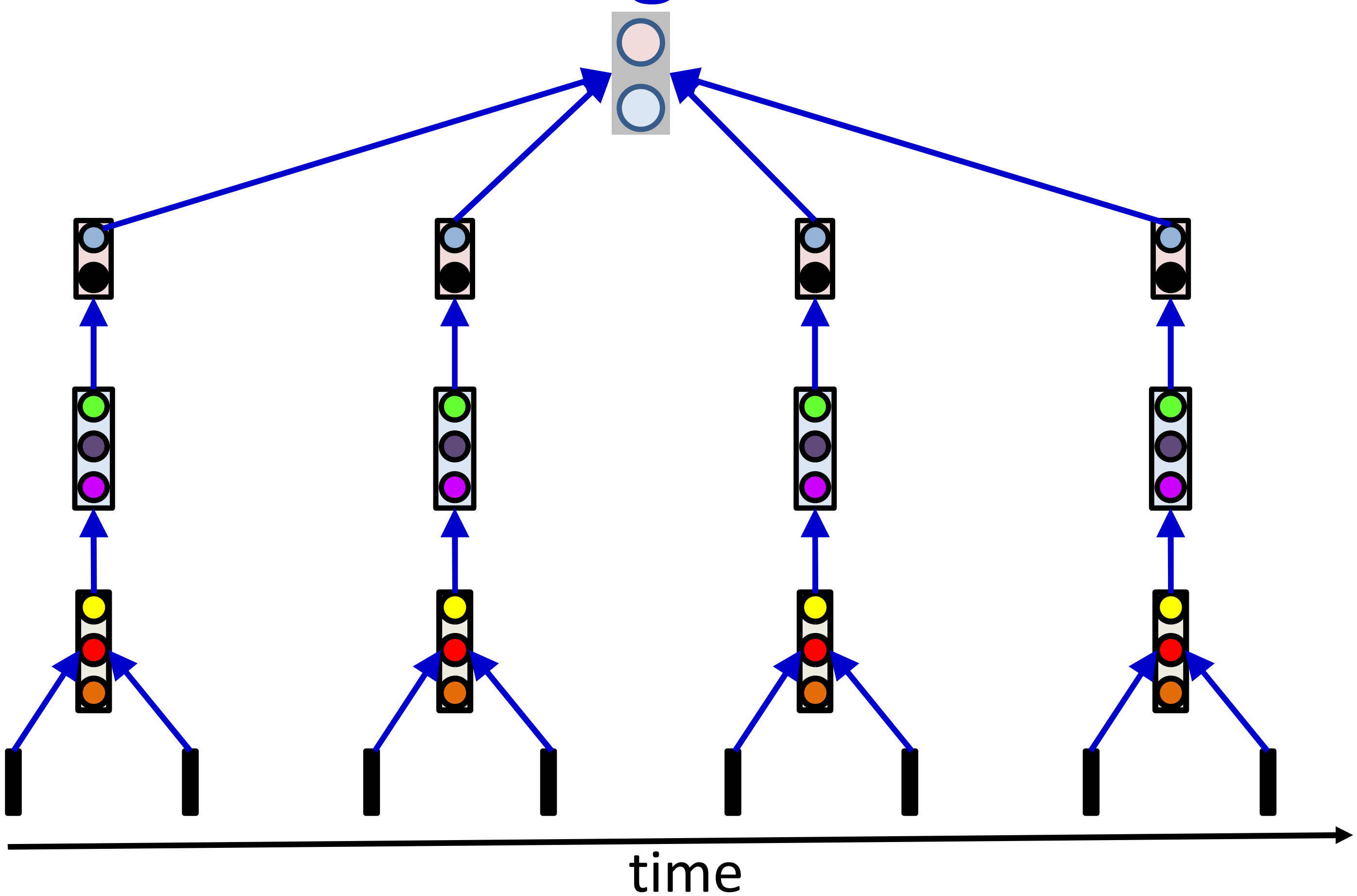
- Modifying the visualization for intuition..
  – Will still be the same network

Scanning networks



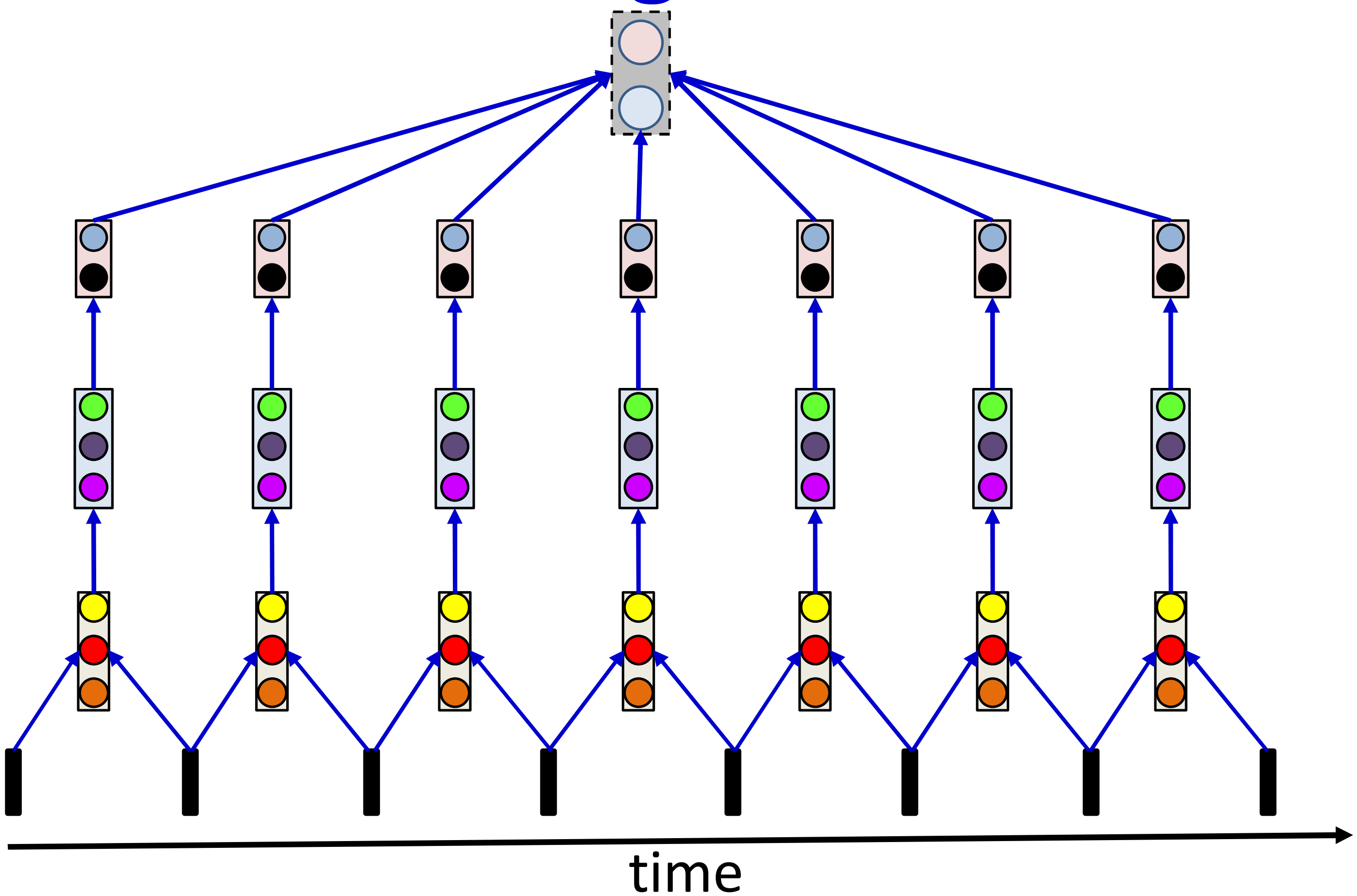
- A modified drawing
  - Indicates progression of time/space
    - The progression of “bars” of neurons is indicative of time
    - Note: bars at the lowest level are also *vectors* of inputs
  - More appropriate
    - Since vertical bars are vectors

Scanning networks



- A modified drawing
  - Indicates progression of time/space
  - An arrow from one bar to another implies connections from *every* node in the source bar to *every* node in the destination bar
    - For N source-bar nodes and M destination-bar nodes, NxM connections

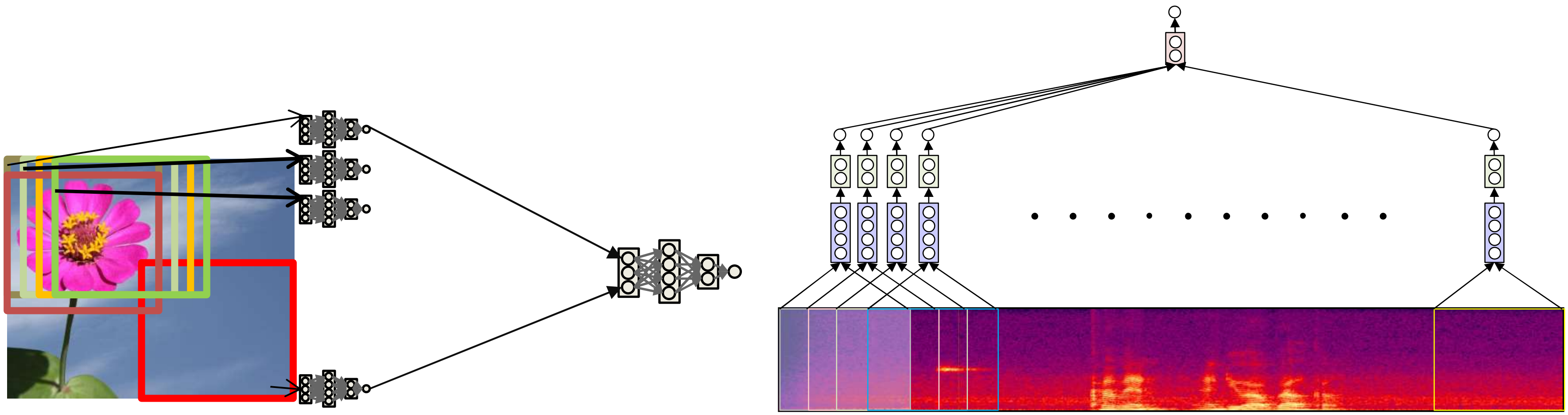
Scanning networks



Visualizing scanning with a stride of 1

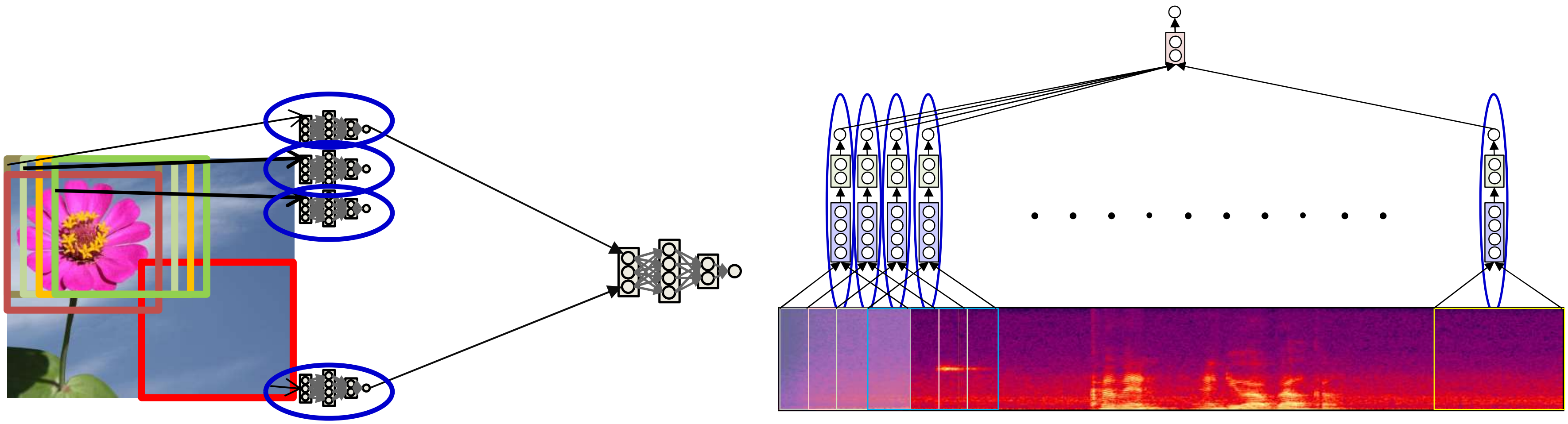
- A modified drawing
  - Indicates progression of time/space
  - An arrow from one bar to another implies connections from *every* node in the source bar to *every* node in the destination bar
    - For N source-bar nodes and M destination-bar nodes, NxM connections

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
      - Target output 1 for flower images, 0 for non-flower images
    - Speech recordings with and without the word "welcome"
      - Target output 1 for "welcome" recordings, 0 for recordings without "welcome"
  - Gradient descent to minimize the total divergence between predicted and desired outputs
- Will actually learn the lower-level flower (or welcome) detector

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

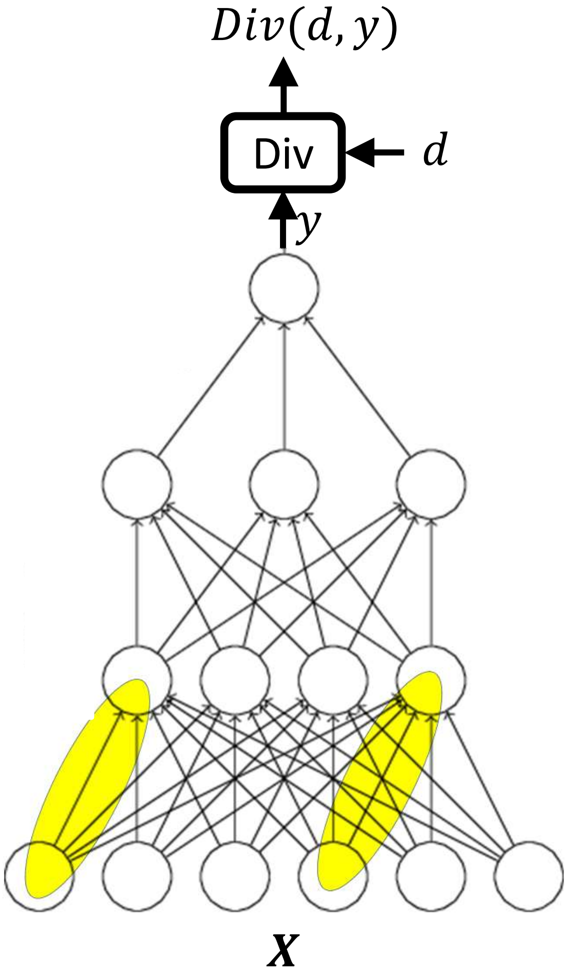
Learning in shared parameter networks

- Consider a simple network with shared weights

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

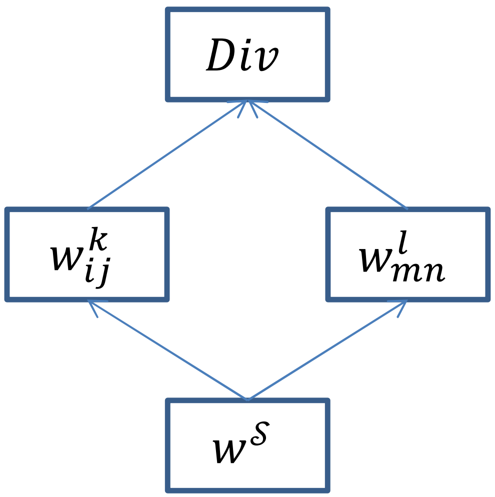
  - A weight w_{ij}^{k} is required to be identical to the weight w_{mn}^{l}

- 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 $Div(d, y)$



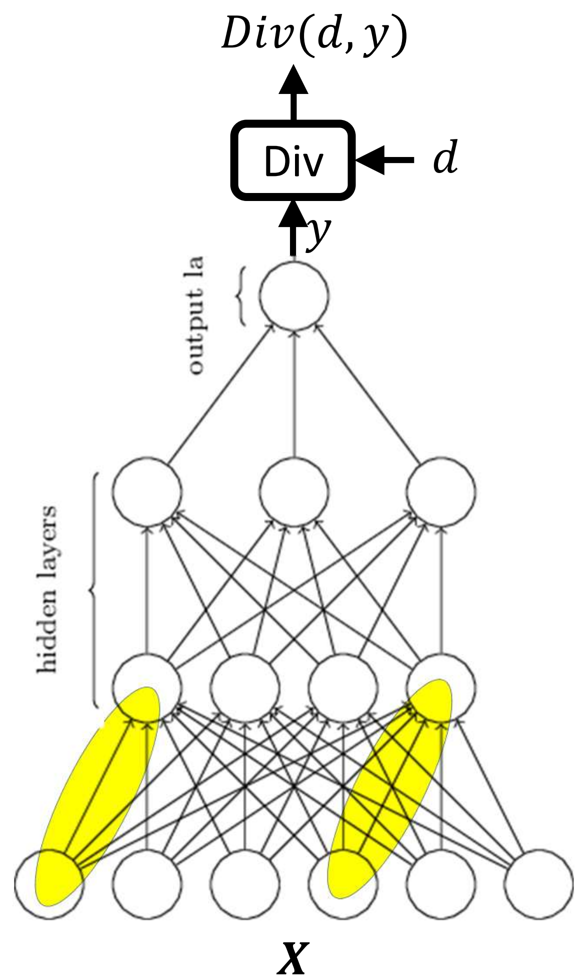
Computing the divergence of shared parameters

Influence diagram



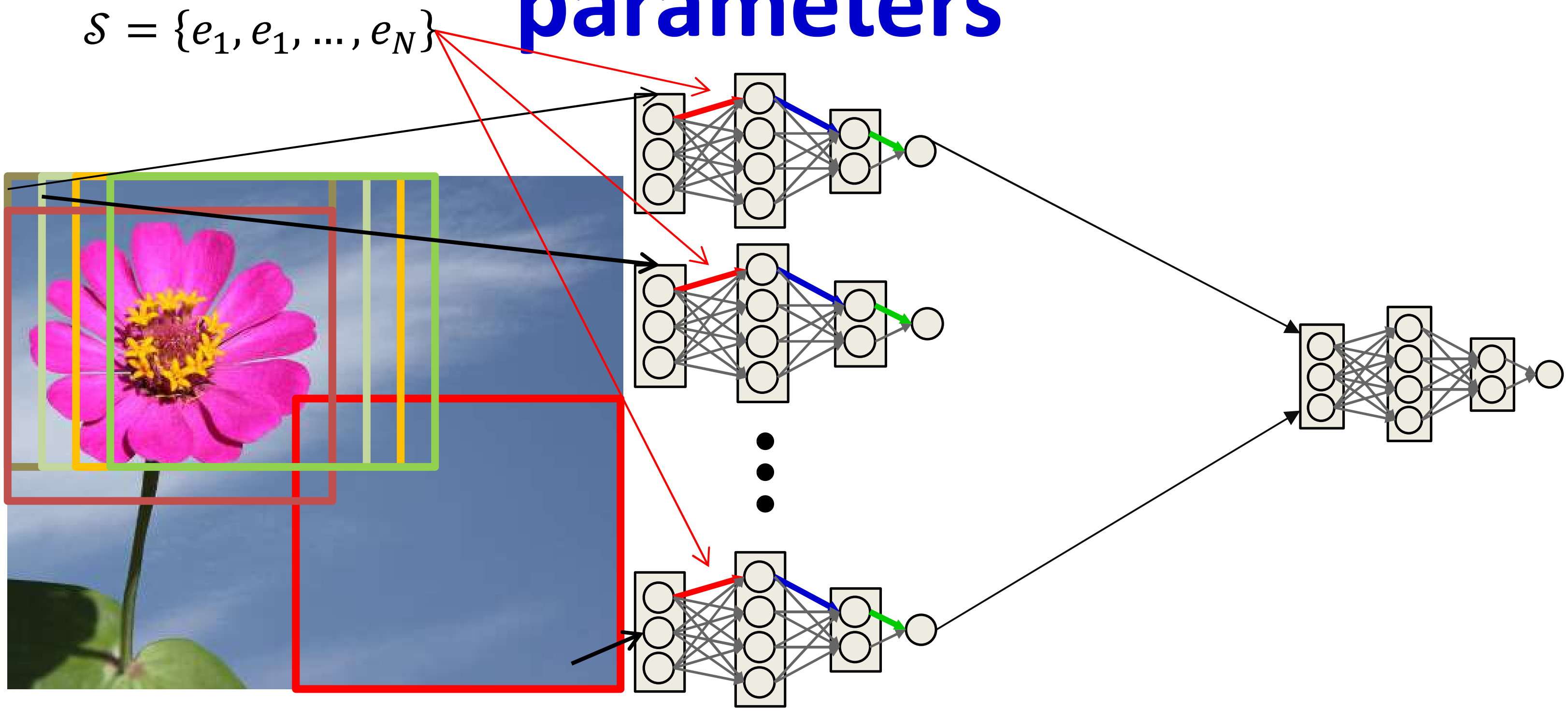
$$\frac{dDiv}{dw^{\mathcal{S}}} = \frac{\partial Div}{\partial w_{ij}^{k}}\frac{dw_{ij}^{k}}{dw^{\mathcal{S}}} + \frac{\partial Div}{\partial w_{mn}^{l}}\frac{dw_{mn}^{l}}{dw^{\mathcal{S}}}$$

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



- 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 $\mathbf{W}_1, \mathbf{W}_2, \dots, \mathbf{W}_K$
- Do:
  - For every set $\mathcal{S}$:
    - Compute:

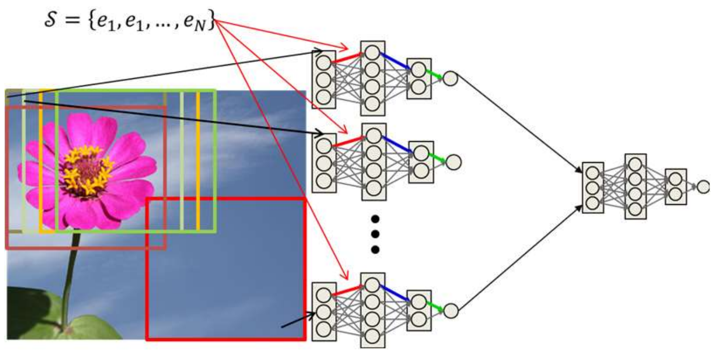
$$\nabla_{\mathcal{S}} Loss = \frac{dLoss}{dw^{\mathcal{S}}}$$

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

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

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

- Until $Loss$ 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:

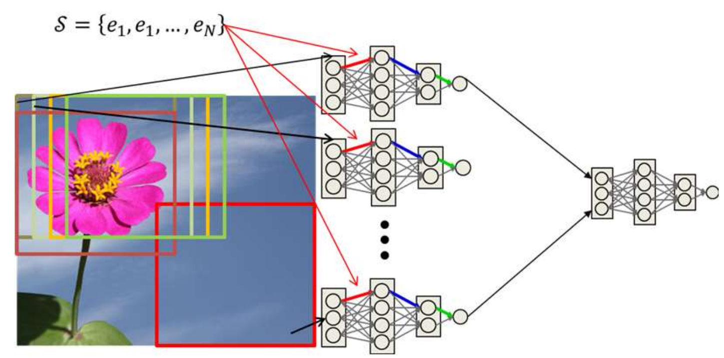
$$\nabla_{\mathcal{S}} Loss = \frac{dLoss}{dw^{\mathcal{S}}}$$

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

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

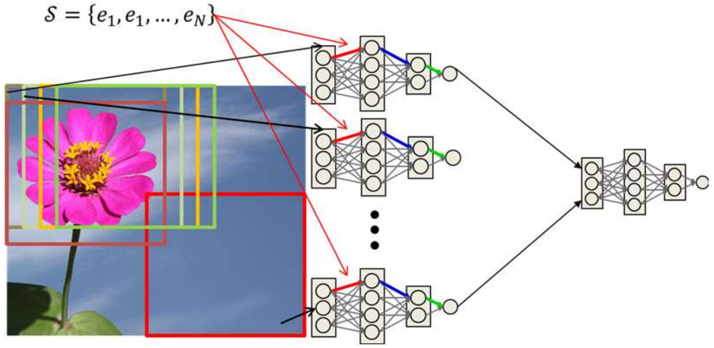
$$w_{i,j}^{(k)} = w^{\mathcal{S}}$$

- Until $Loss$ has converged



Training networks with shared parameters

- For every training instance X
  - For every set $\mathcal{S}$:
    - For every $(k,i,j) \in \mathcal{S}$:
$$\frac{dLoss}{dw^{\mathcal{S}}} += \frac{\partial Div}{\partial w_{i,j}^{(k)}}$$
    - $\nabla_{\mathcal{S}} Loss = \frac{dLoss}{dw^{\mathcal{S}}}$
  - Compute:
$$\nabla_{\mathcal{S}} Loss = \frac{dLoss}{dw^{\mathcal{S}}}$$
$$w^{\mathcal{S}} = w^{\mathcal{S}} - \eta \nabla_{\mathcal{S}} Loss^T$$
  - For every $(k,i,j) \in \mathcal{S}$ update:
$$w_{i,j}^{(k)} = w^{\mathcal{S}}$$
- Until $Loss$ has converged



Training networks with shared parameters

- For every training instance X
  - For every set $\mathcal{S}$:
    - For every $(k, i, j) \in \mathcal{S}$:
      $$\frac{dLoss}{dw^{\mathcal{S}}} += \frac{\partial Div}{\partial w_{i,j}^{(k)}}$$
    - $\nabla_{\mathcal{S}} Loss = \frac{dLoss}{dw^{\mathcal{S}}}$

Computed by Backprop

- Compute:

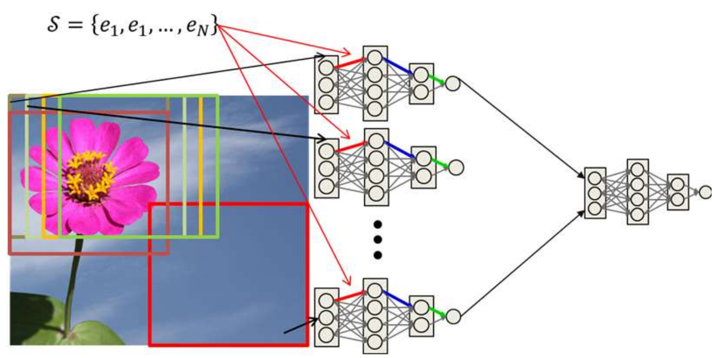
$$\nabla_{\mathcal{S}} Loss = \frac{dLoss}{dw^{\mathcal{S}}}$$

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

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

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

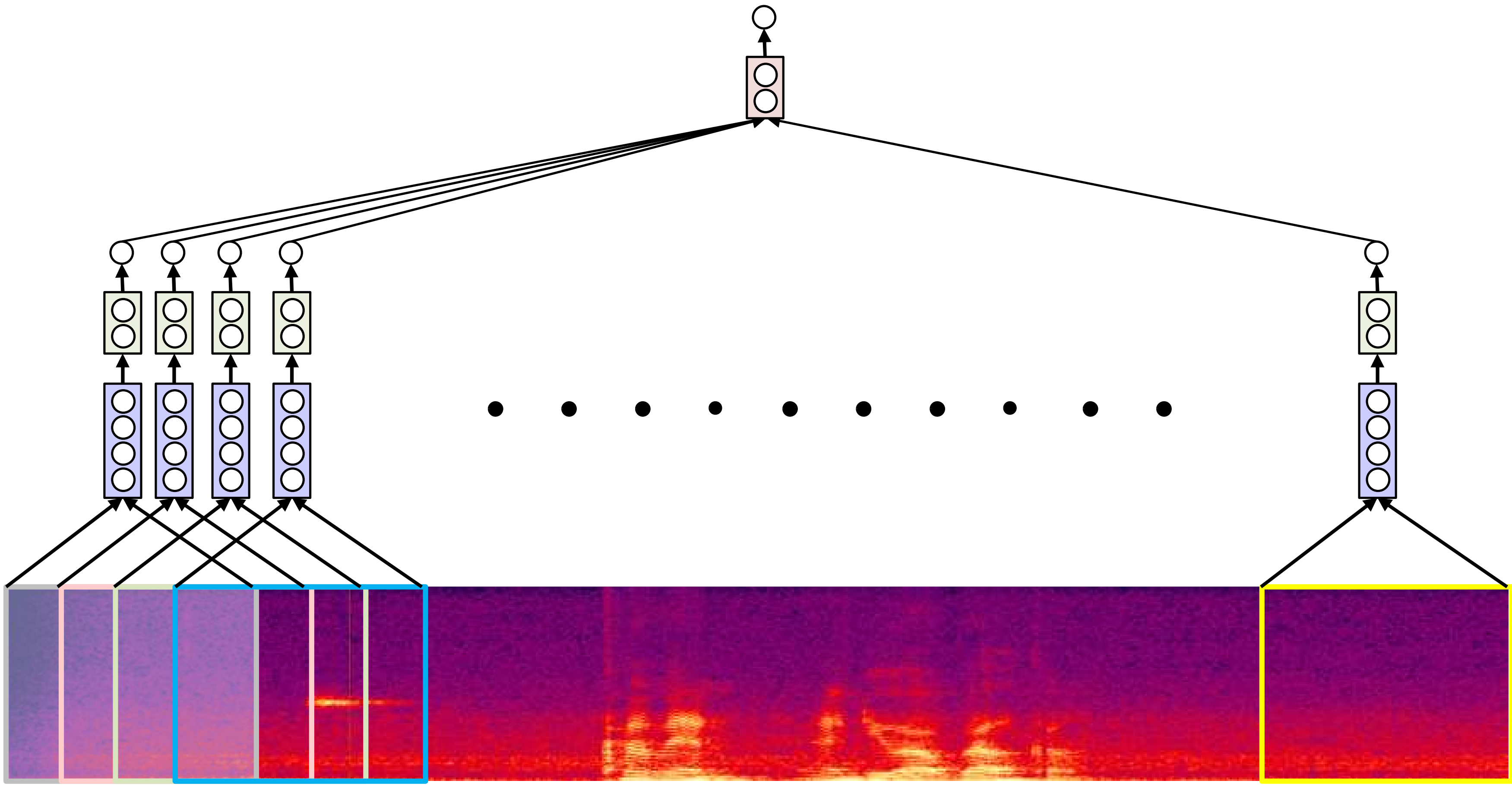
- Until $Loss$ has converged

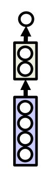
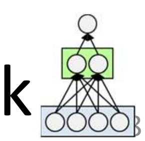


Story so far

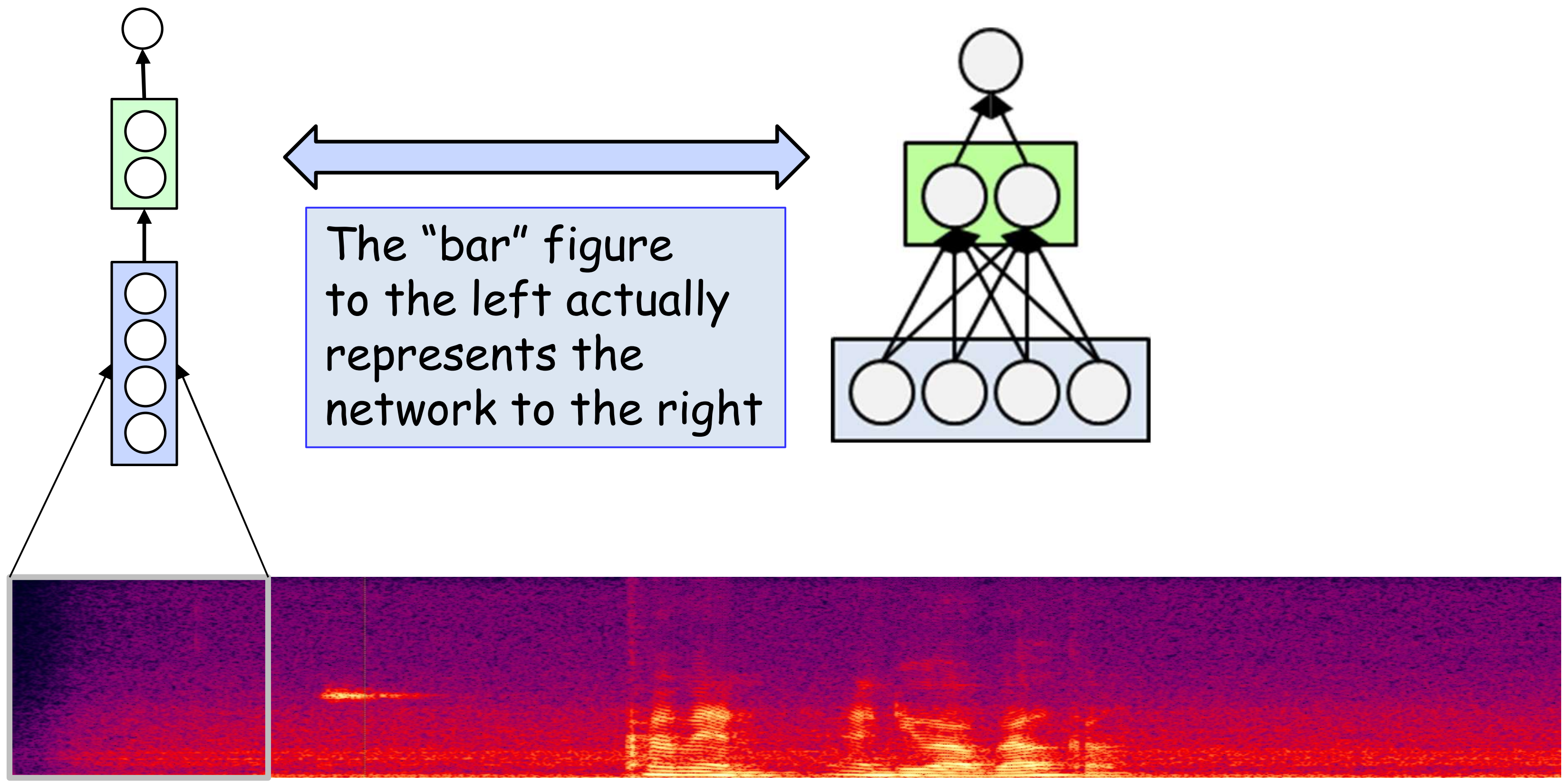
- Position-invariant pattern classification can be performed by scanning
  - 1-D scanning for sound
  - 2-D scanning for images
  - 3-D and higher-dimensional scans for higher dimensional data
- Scanning is equivalent to composing a large network with repeating subnets
  - The large network has shared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



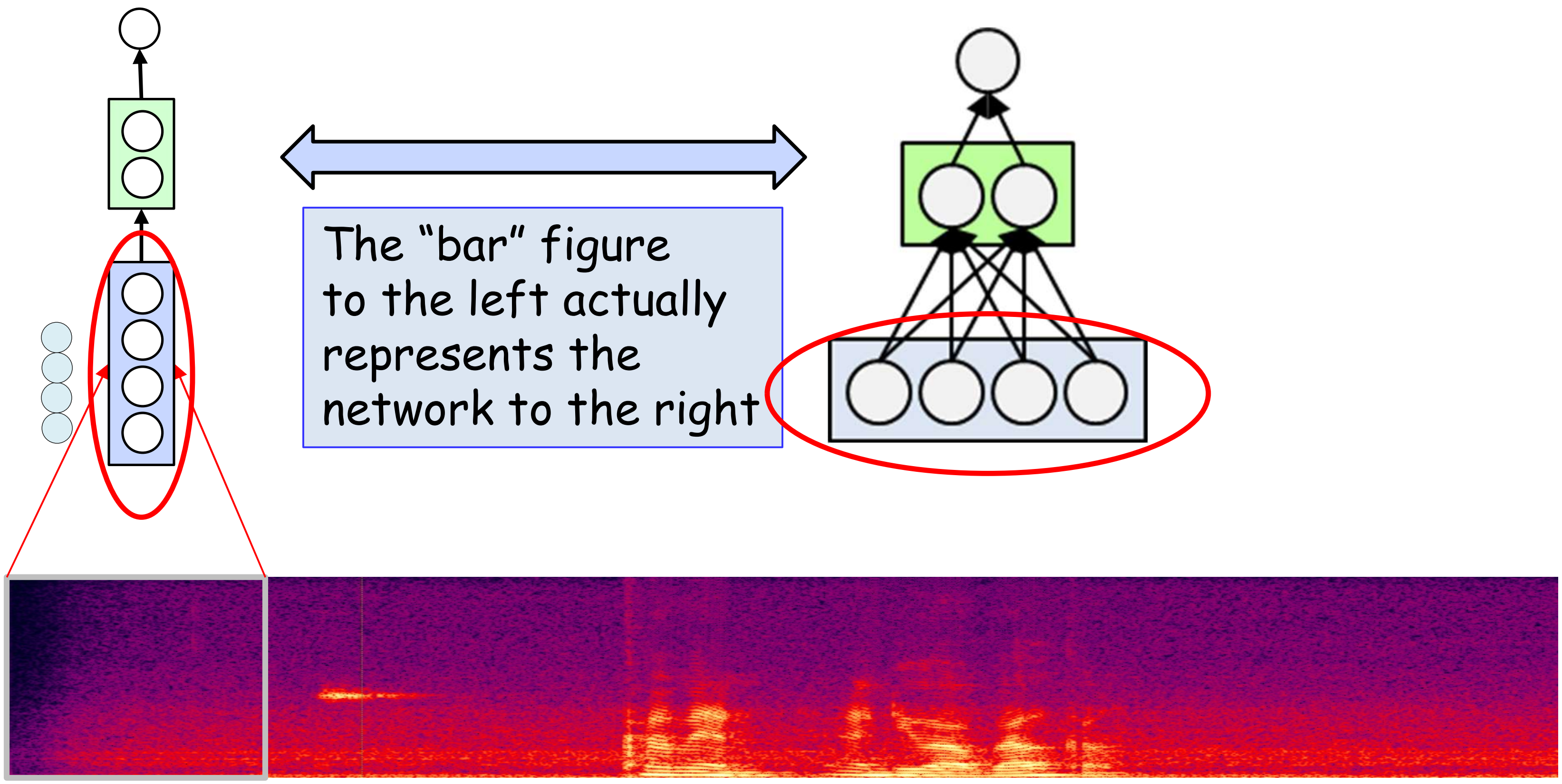
- The entire MLP operates on each "window" of the input
  - Using the "bar" visual  to represent the network 

Scanning



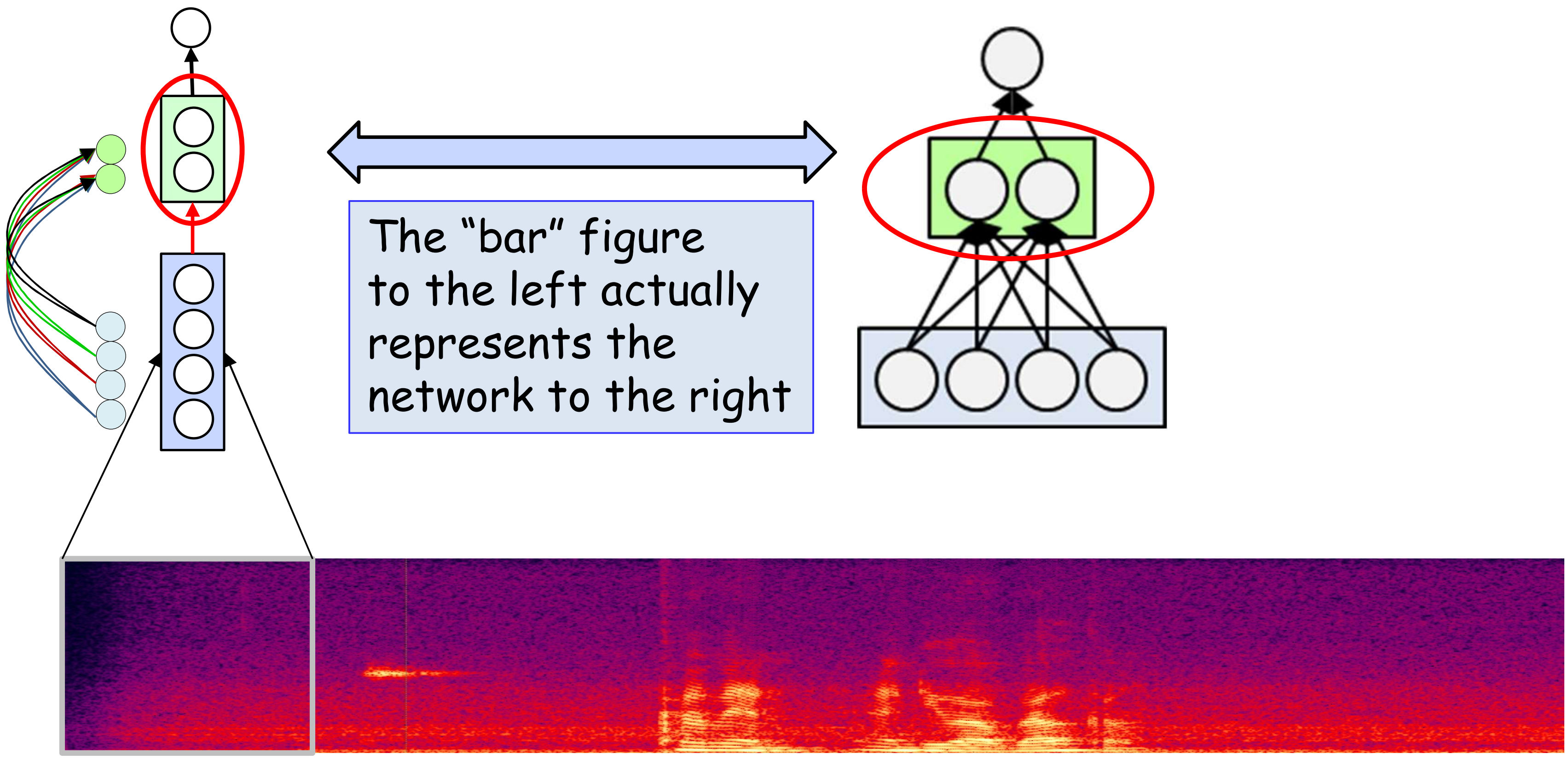
- At each location, each neuron computes a value based on its inputs
  - Which may either be the input image or the outputs of the previous layer

Scanning



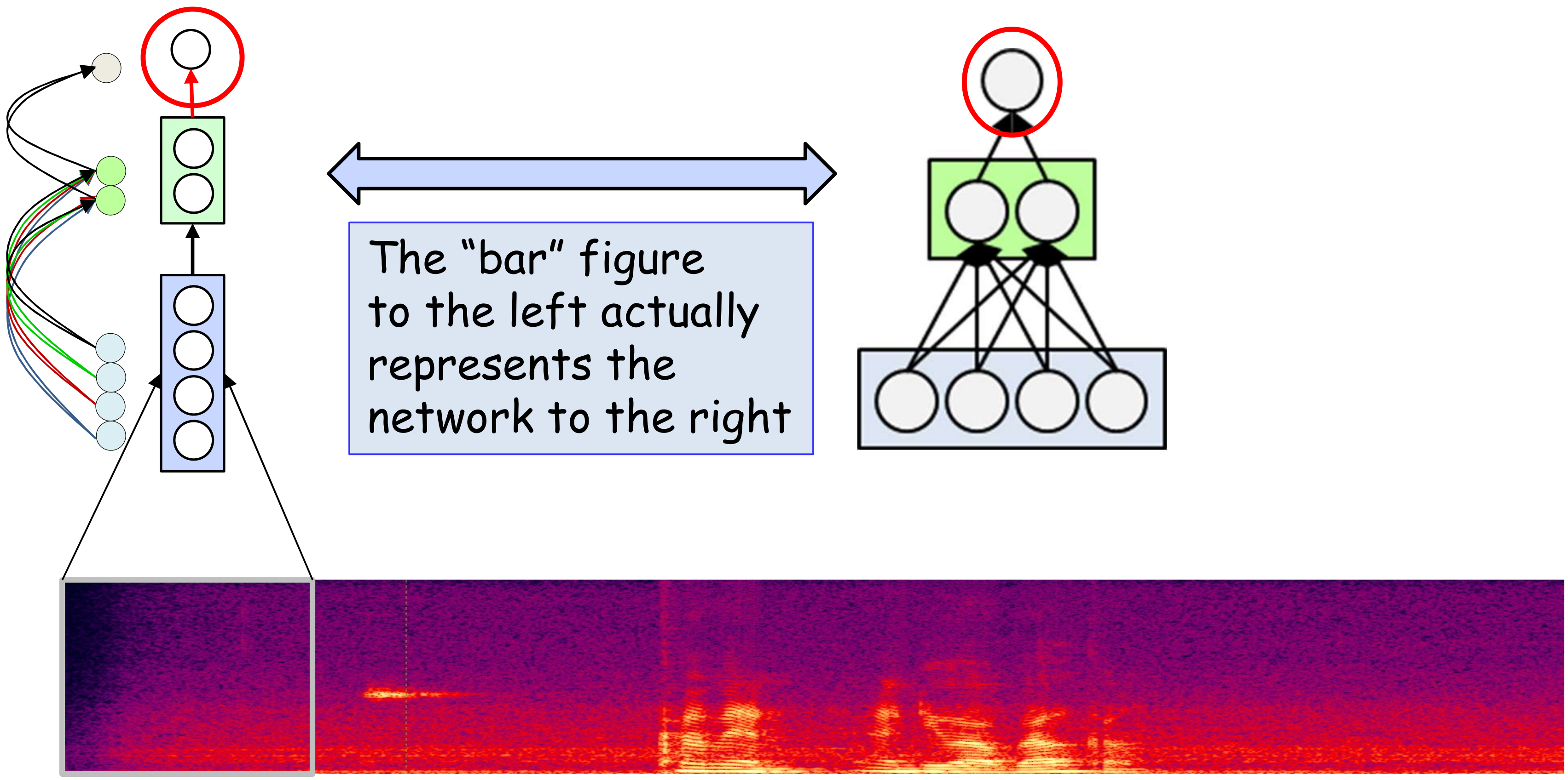
- At each location, each neuron computes a value based on its inputs
  - Which may either be the input image
  -

Scanning



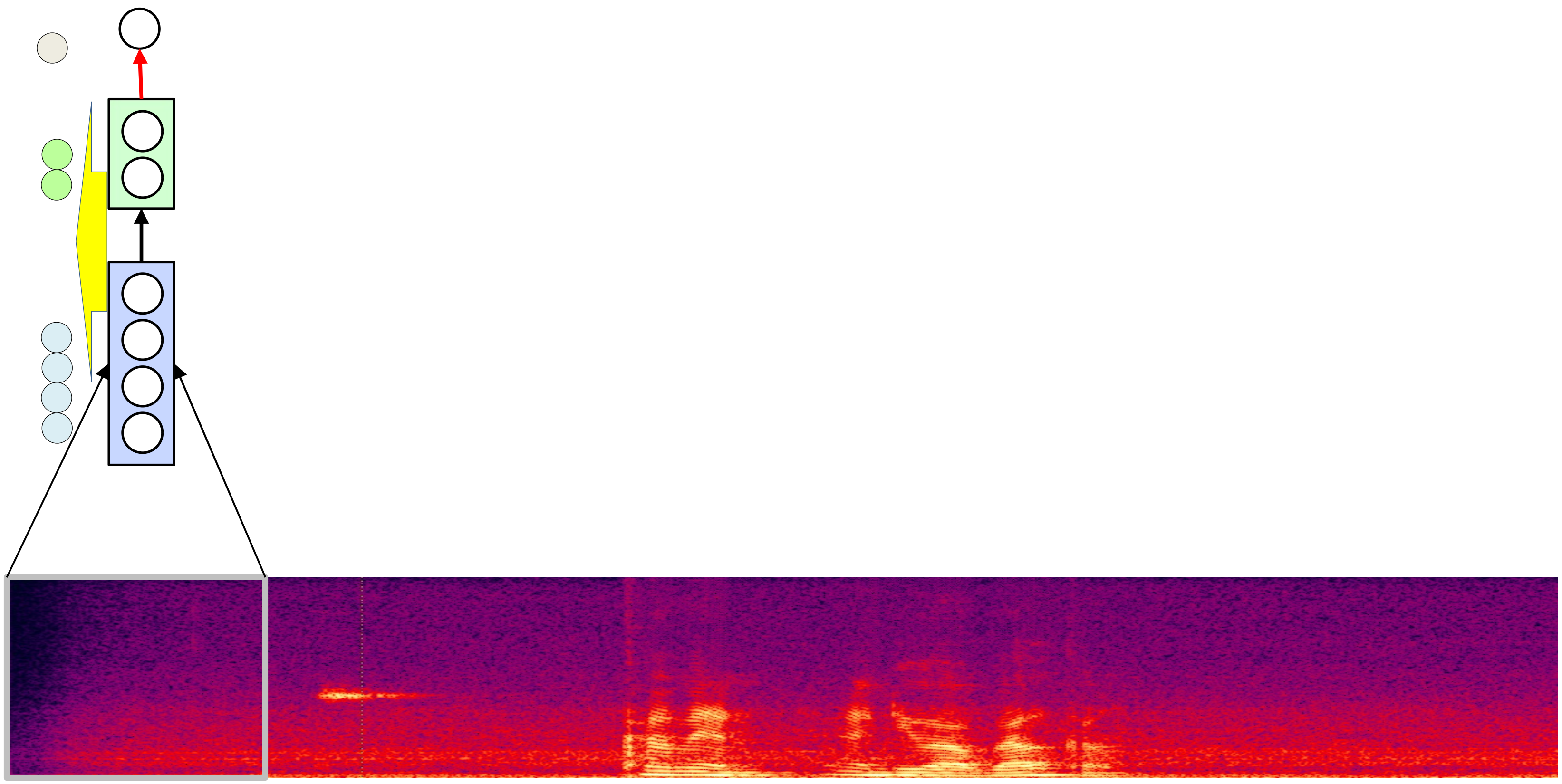
- At each location, each neuron computes a value based on its inputs
  - Which may either be the input image or the outputs of the previous layer

Scanning



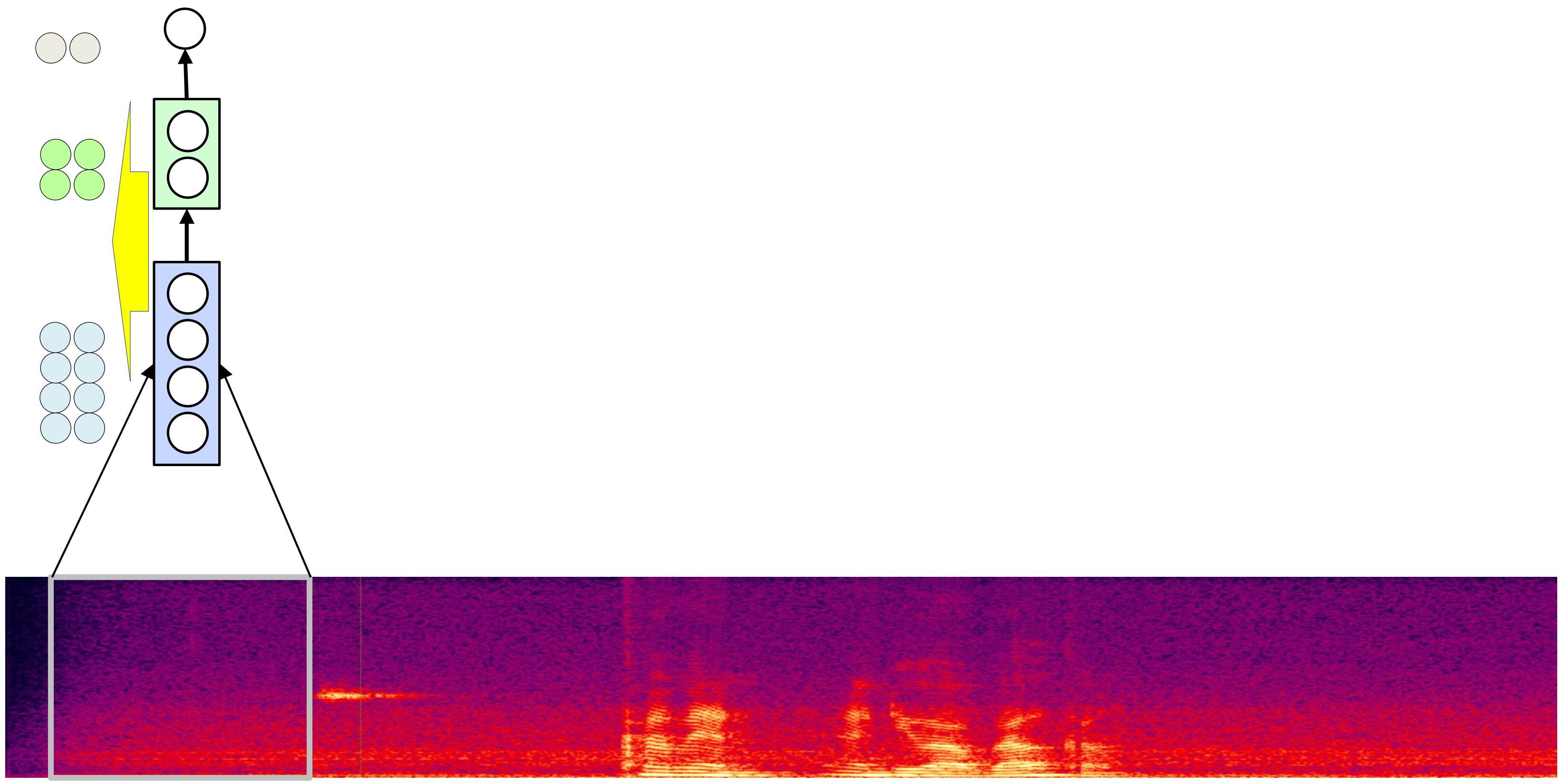
- At each location, each neuron computes a value based on its inputs
  - Which may either be the input image or the outputs of the previous layer

Scanning



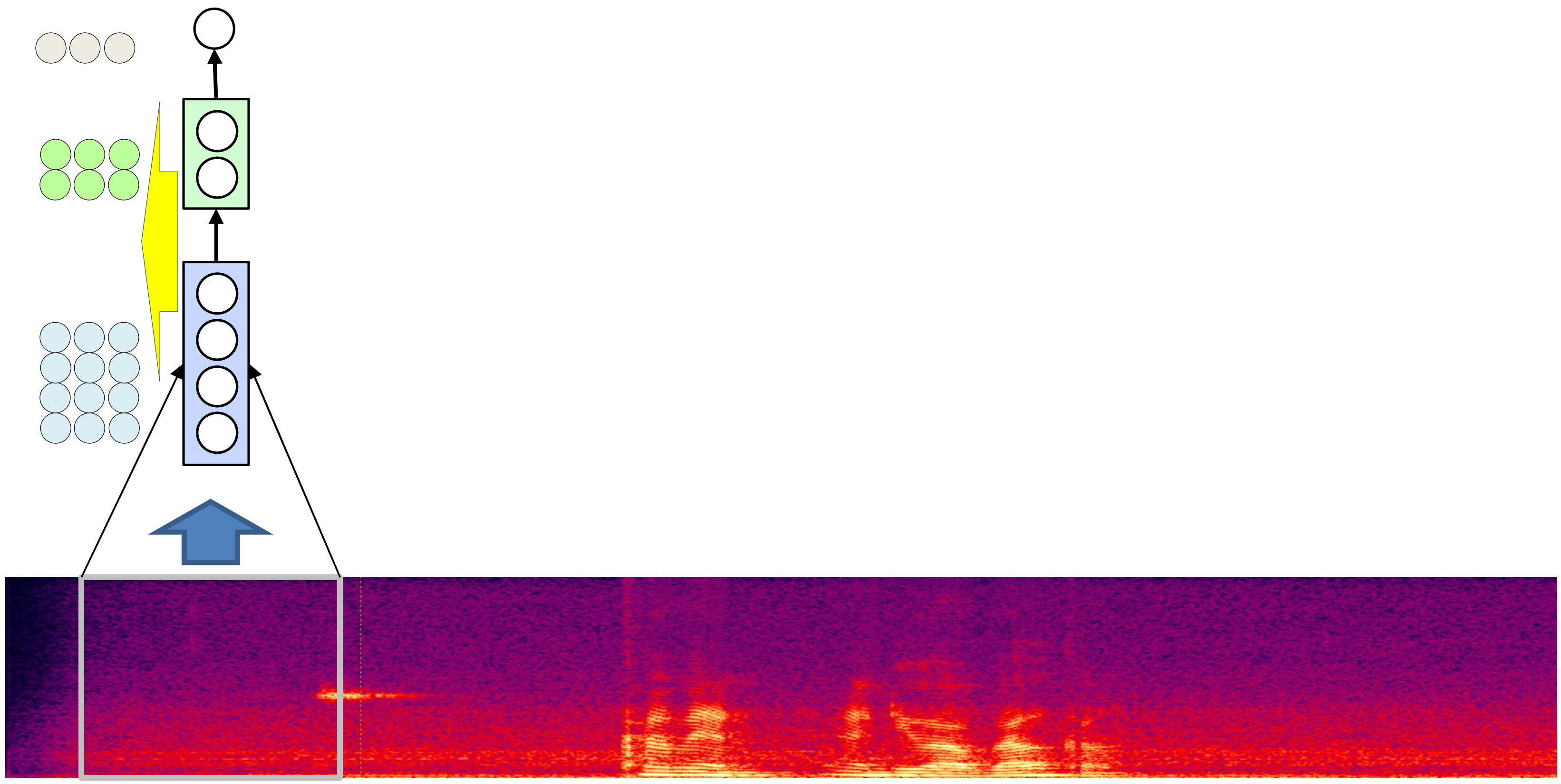
- At each location, each neuron computes a value based on its inputs
  - Which may either be the input image or the outputs of the previous layer

Scanning



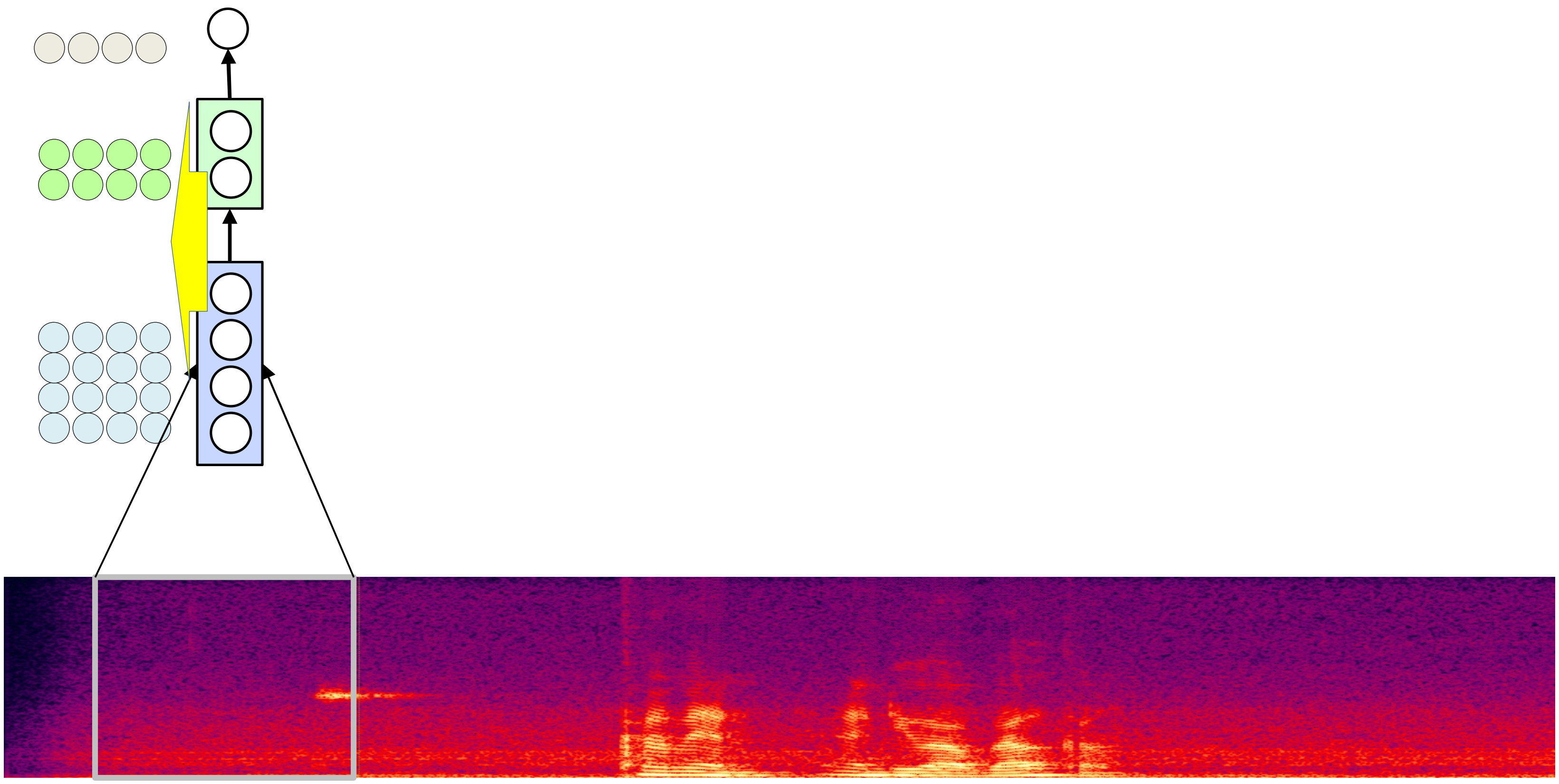
- The same sequence of computations is performed at each location
  - Producing similar sets of values
    - One value per neuron in each layer

Scanning



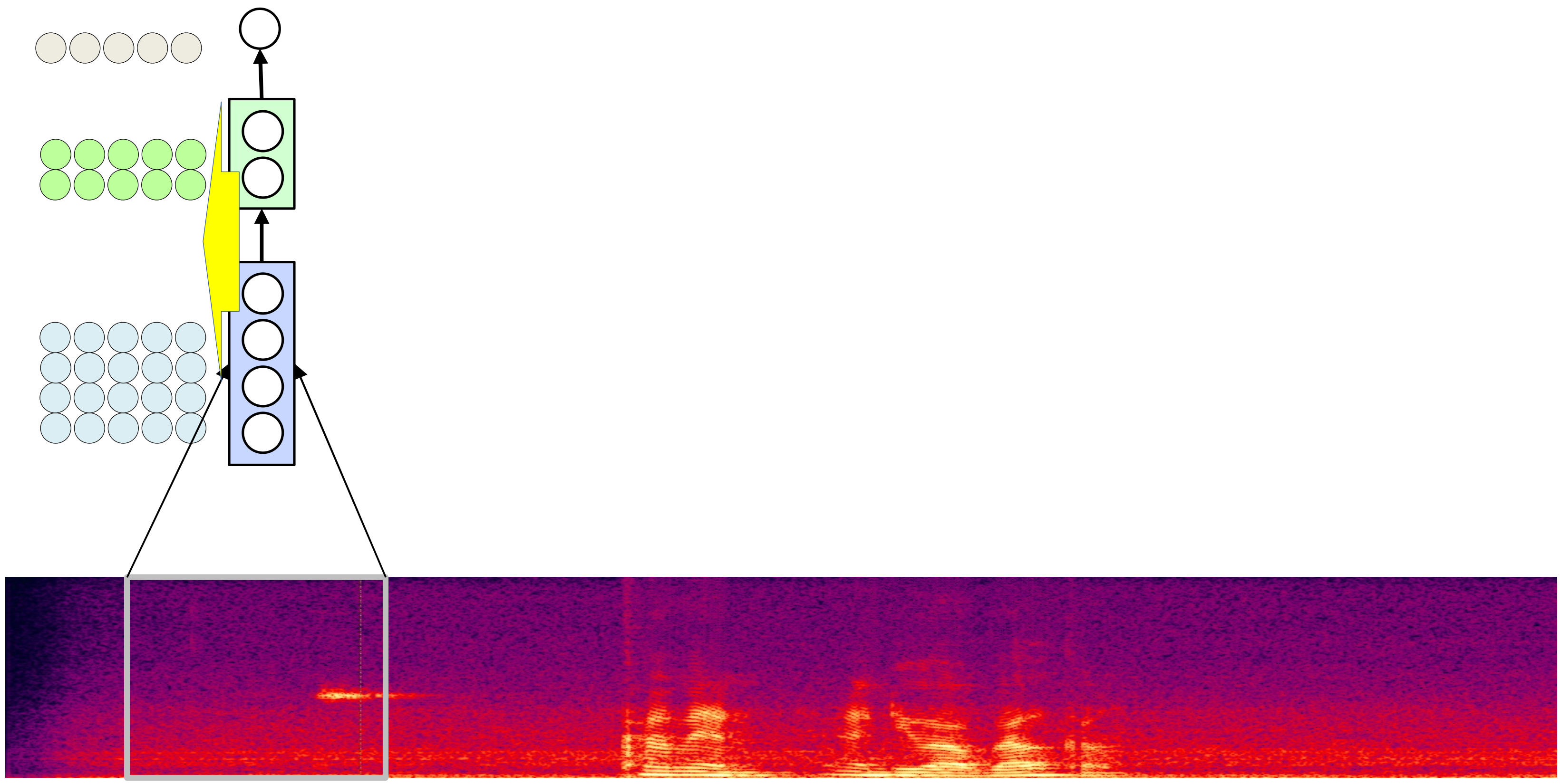
- The same sequence of computations is performed at each location
  – Producing similar sets of values
    - One value per neuron in each layer

Scanning



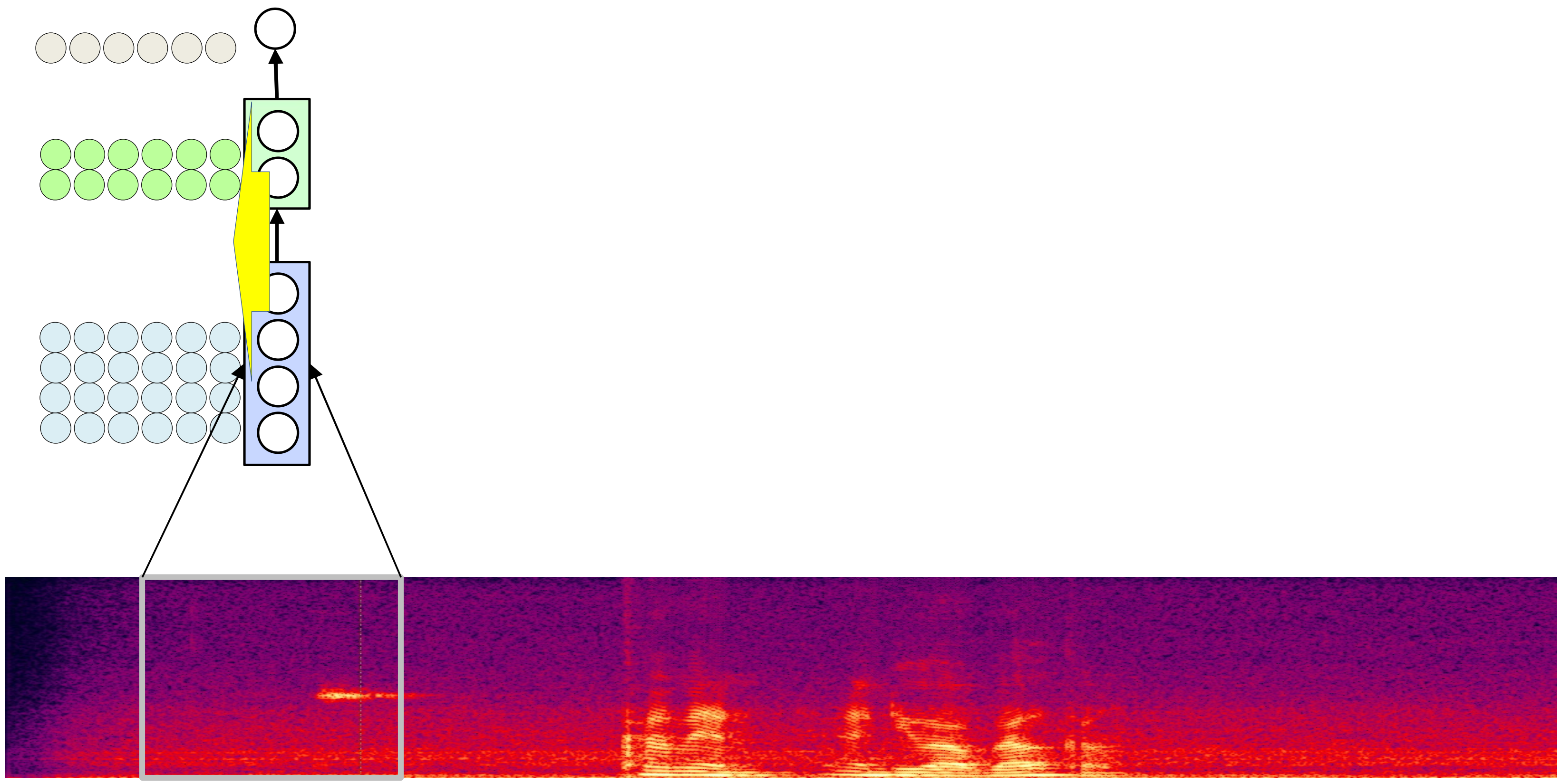
- The same sequence of computations is performed at each location
  - Producing similar sets of values
    - One value per neuron in each layer

Scanning



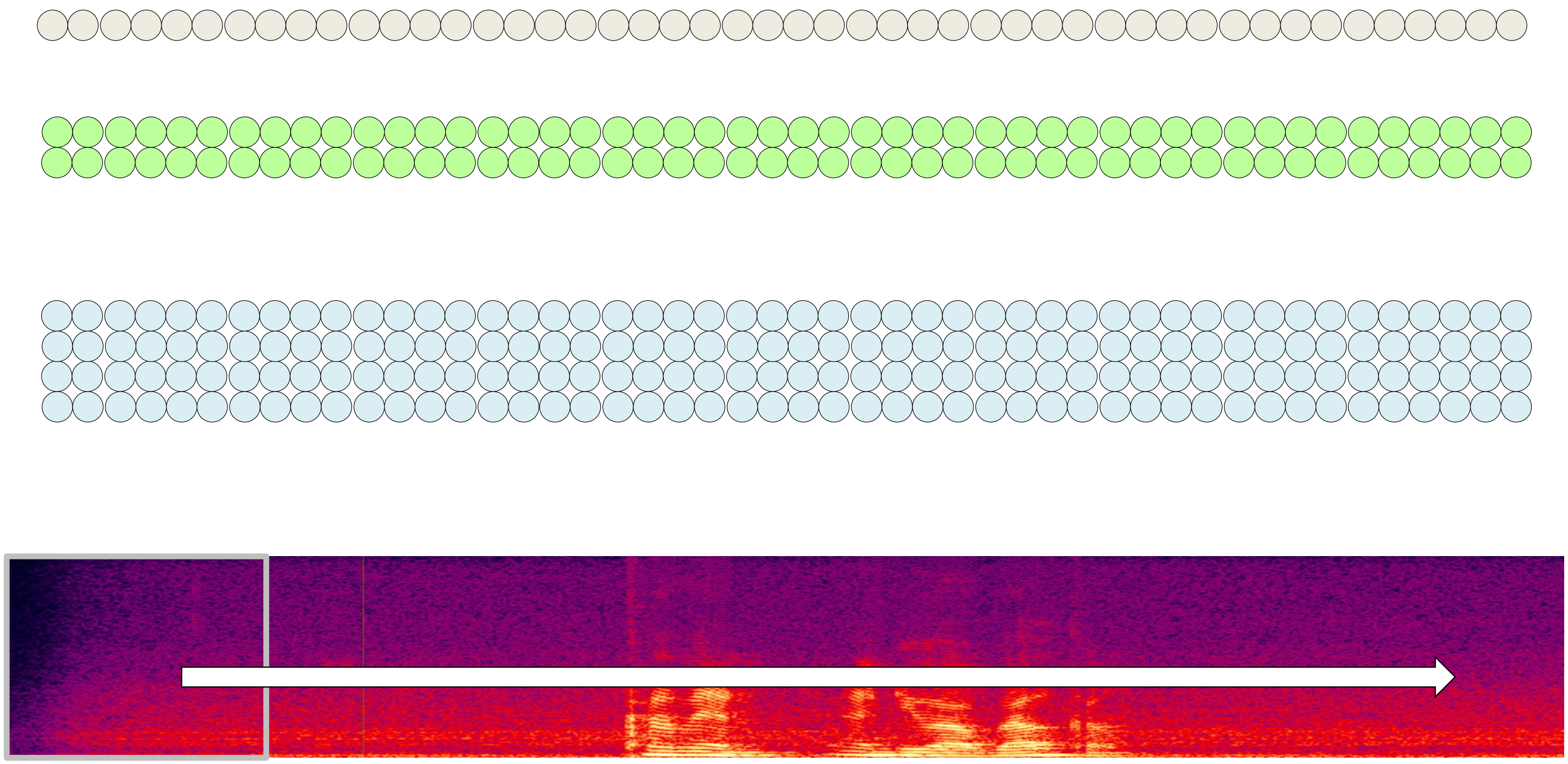
- The same sequence of computations is performed at each location
  - Producing similar sets of values
    - One value per neuron in each layer

Scanning



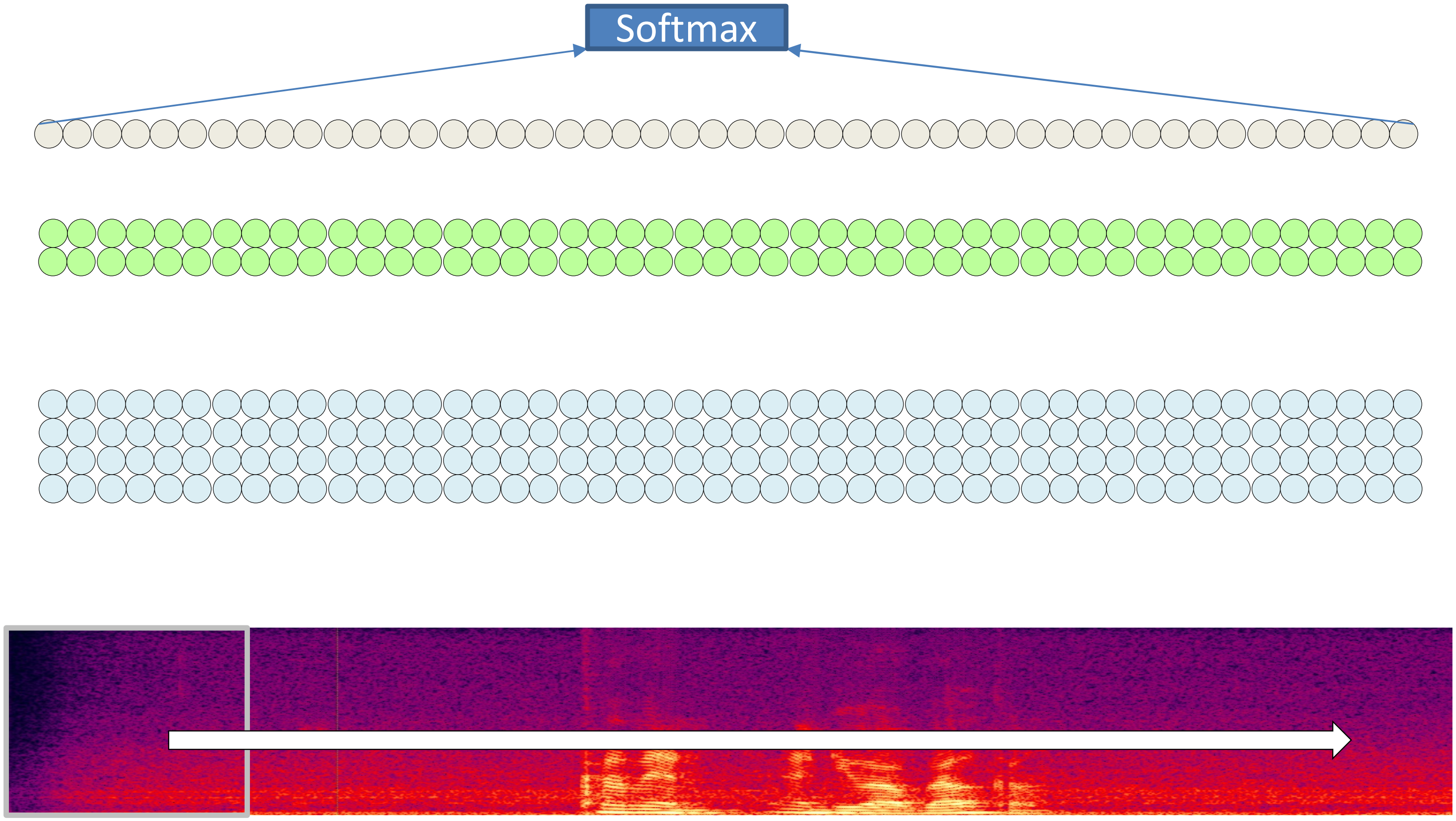
- The same sequence of computations is performed at each location
  - Producing similar sets of values
    - One value per neuron in each layer

Scanning the input



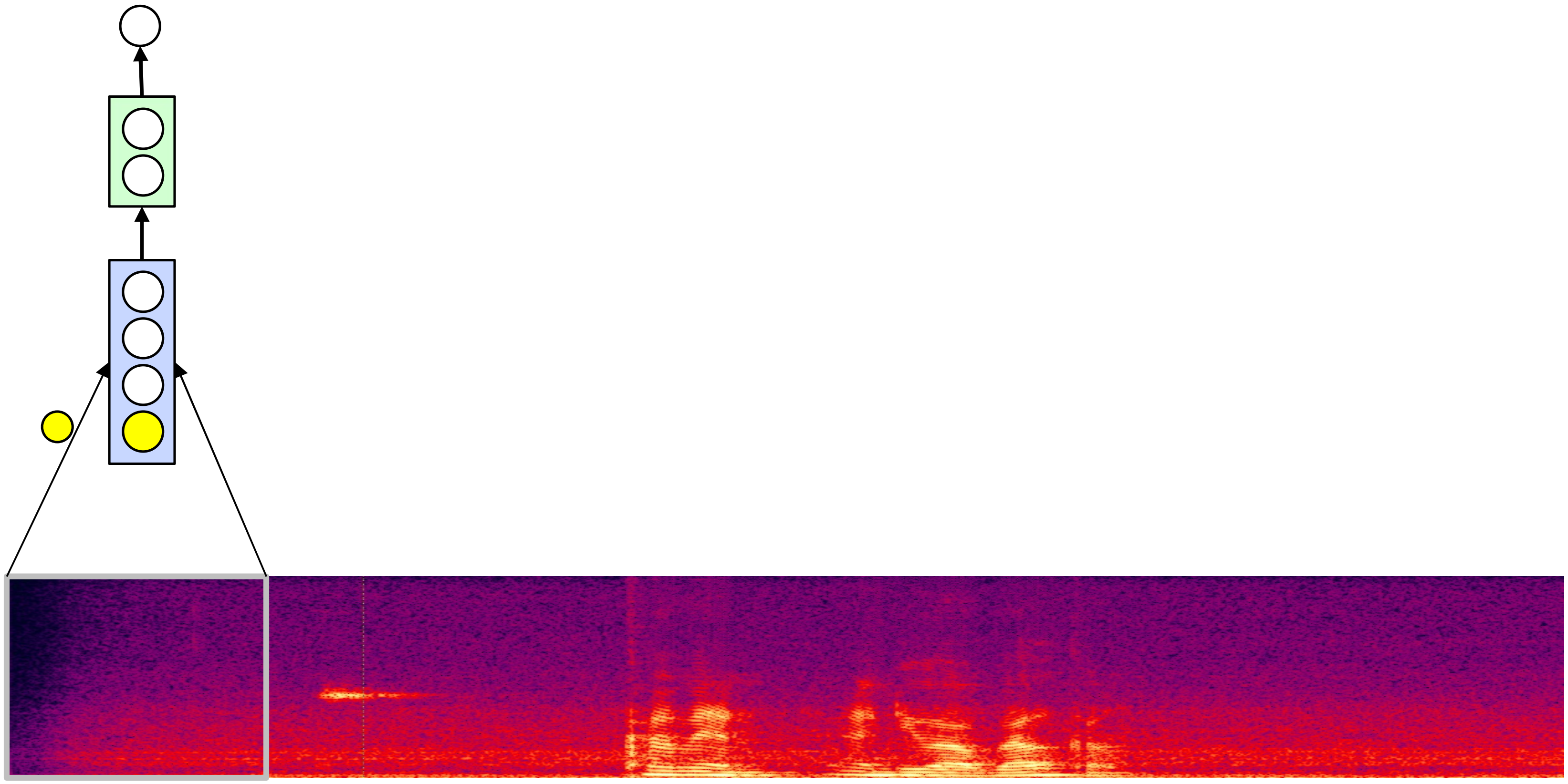
- We get a complete set of values (represented as a column) at each location evaluated by the MLP during the scan

Scanning the input



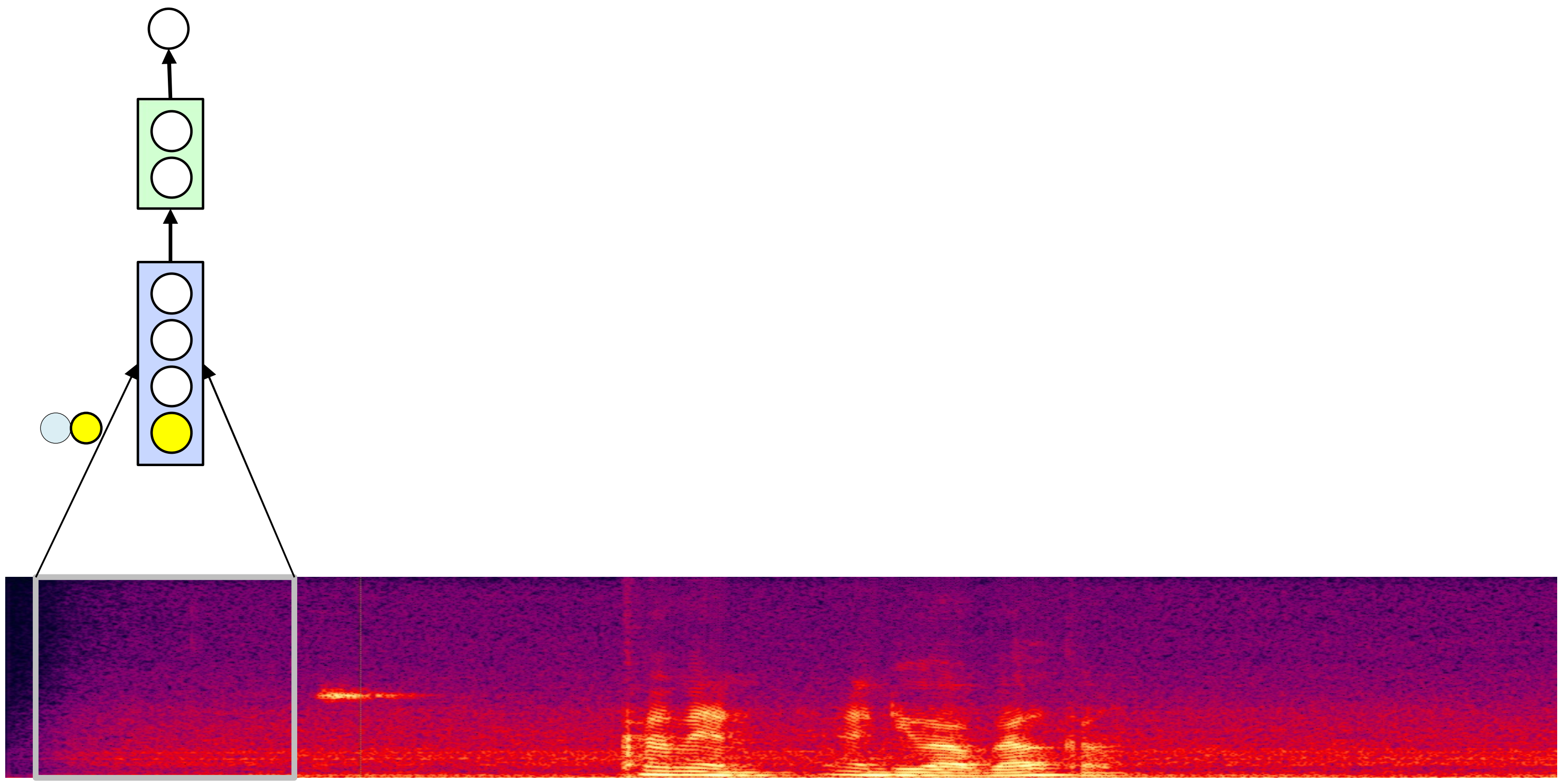
- We get a complete set of values (represented as a column) at each location evaluated by the MLP during the scan
  - Which we put through our final softmax to decide if the recording includes the word "Welcome"

Let's do it in a different order



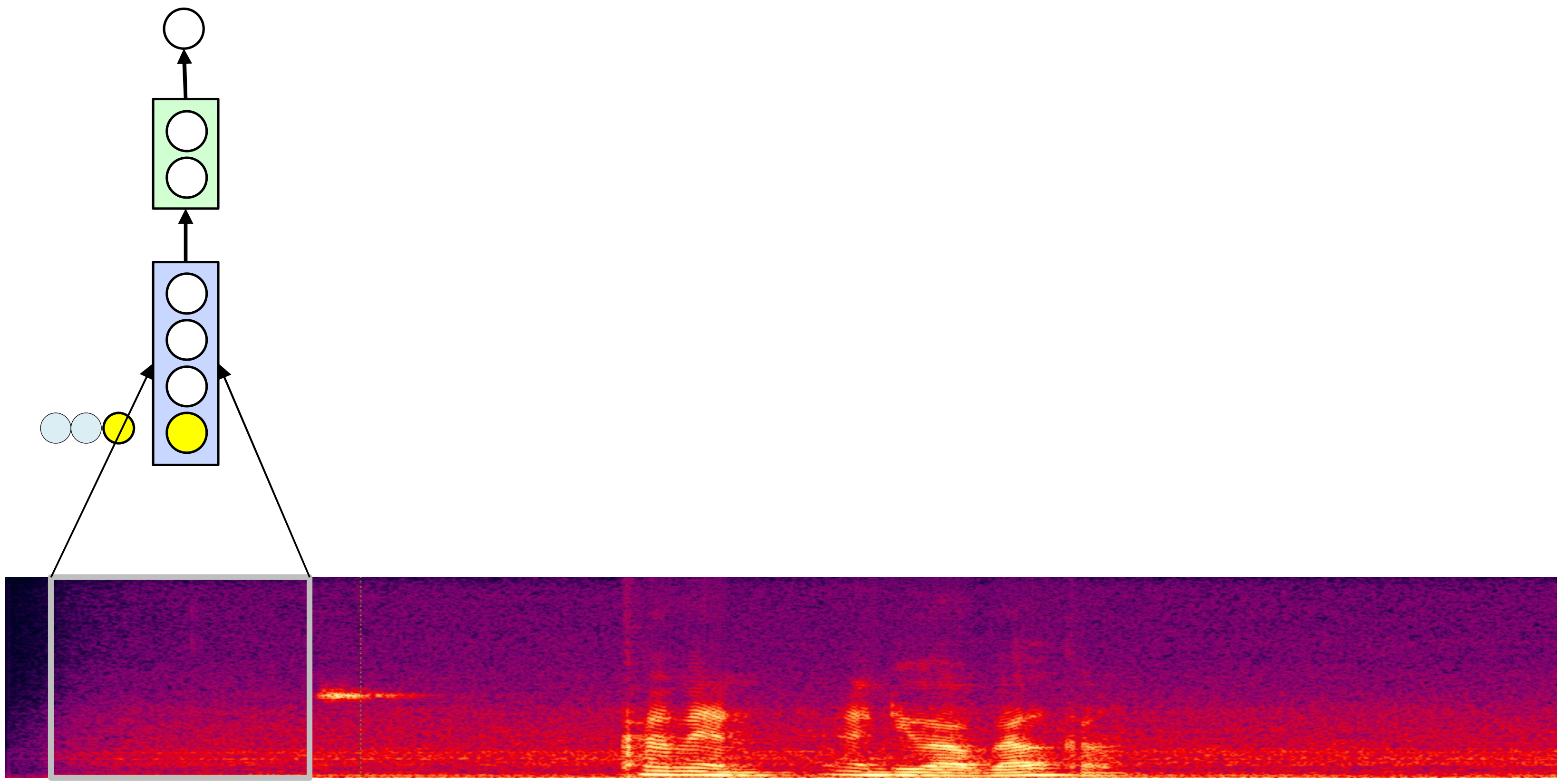
- Let us do the computation in a different order
- The first neuron evaluates each image first
  - "Scans" the input

Let’s do it in a different order



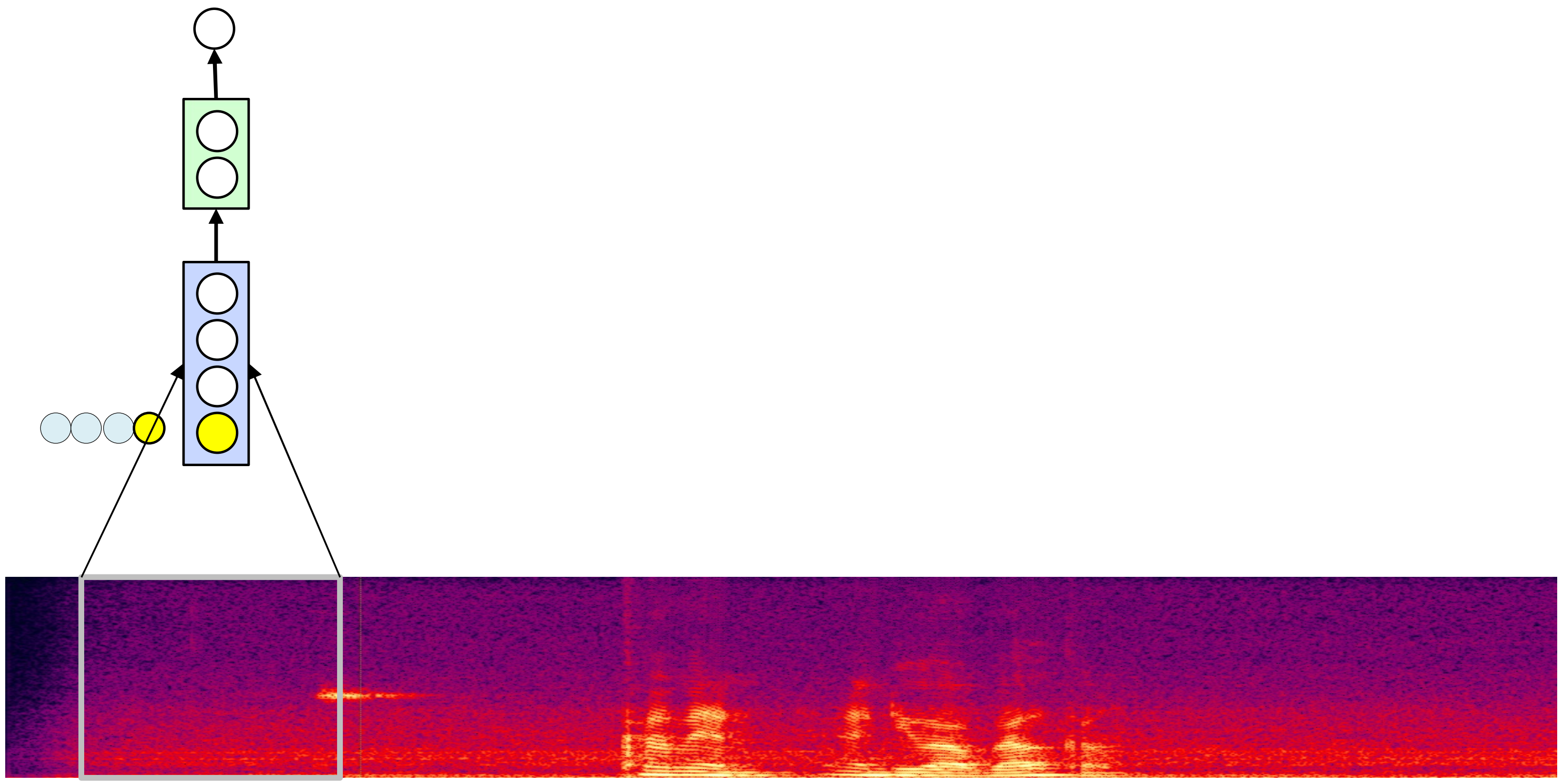
- Let us do the computation in a different order
- The first neuron evaluates each image first
  - “Scans” the input

Let’s do it in a different order



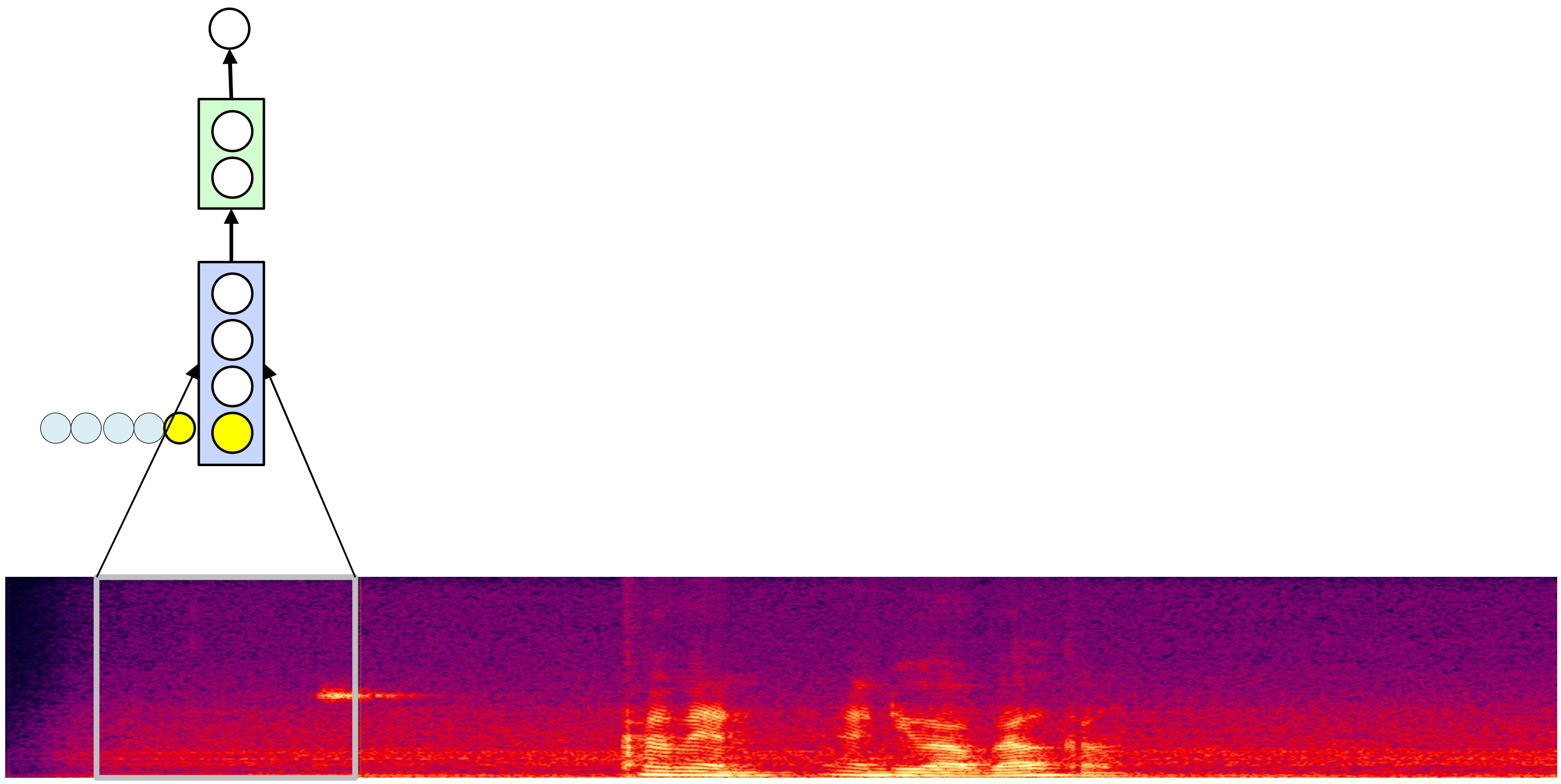
- Let us do the computation in a different order
- The first neuron evaluates each image first
  - “Scans” the input

Let's do it in a different order



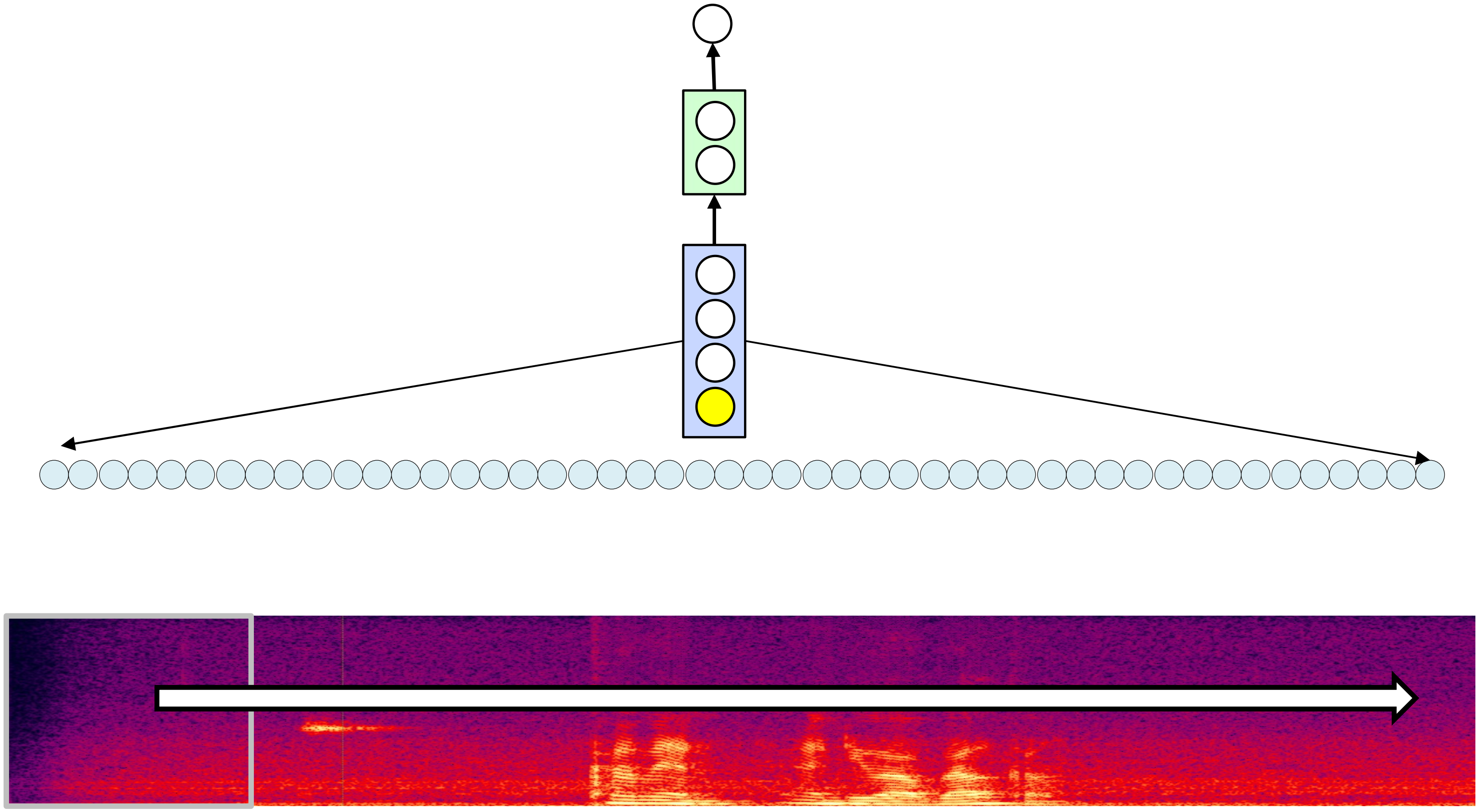
- Let us do the computation in a different order
- The first neuron evaluates each image first
  - "Scans" the input

Let’s do it in a different order



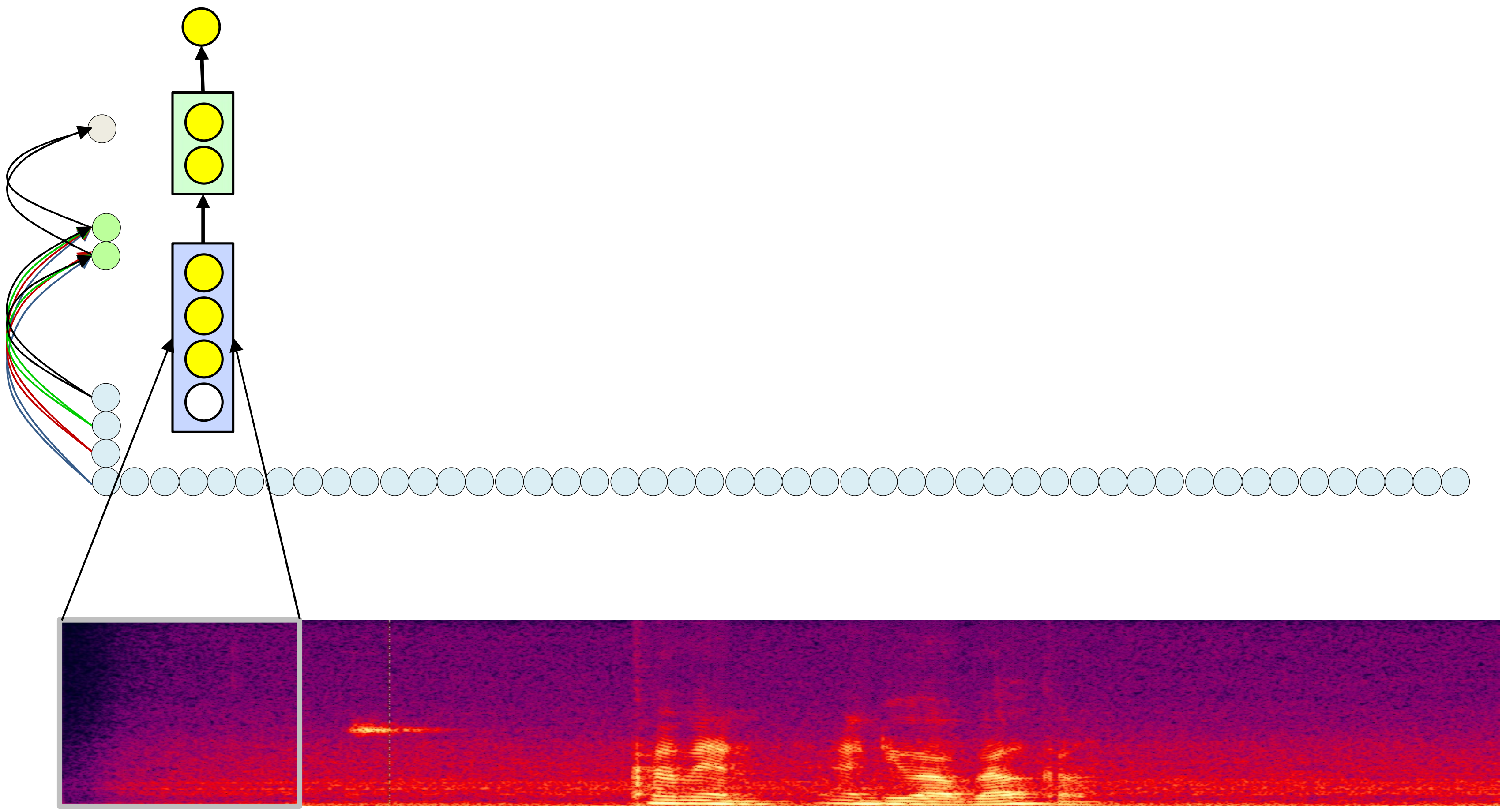
- Let us do the computation in a different order
- The first neuron evaluates each image first
  - “Scans” the input

Let’s do it in a different order



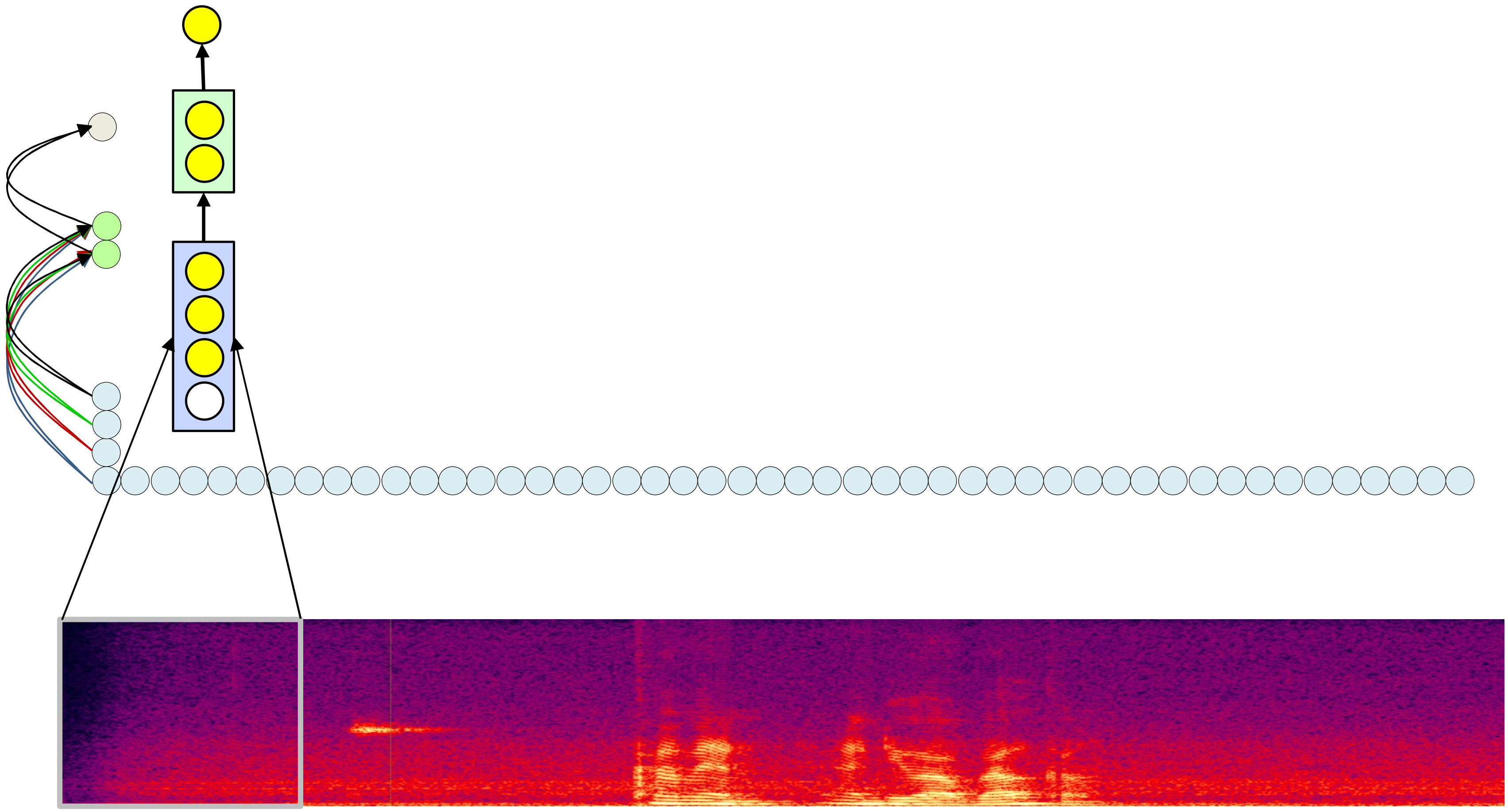
- Let us do the computation in a different order
- The first neuron evaluates each image first
  - “Scans” the input

Let's do it in a different order



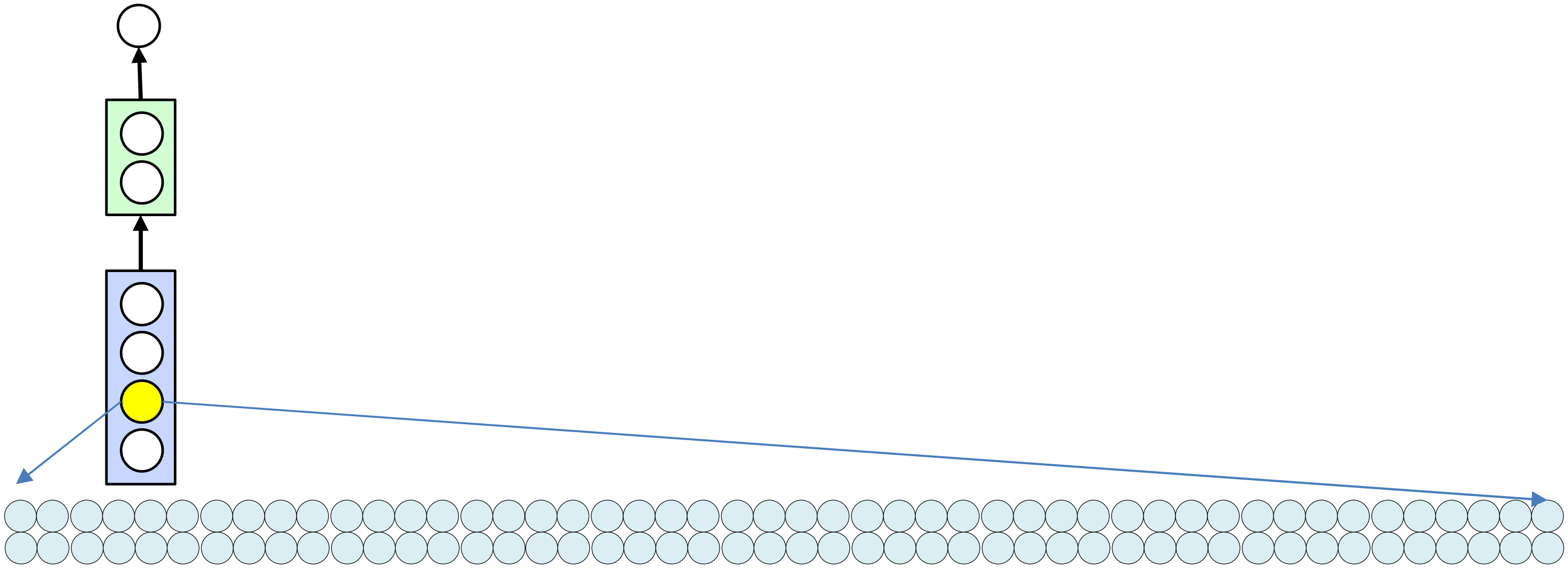
- Subsequently the rest of the neurons in the first layer operate on the first block
  - And the downstream layers as well
- Would the output of the MLP at the first block be different?

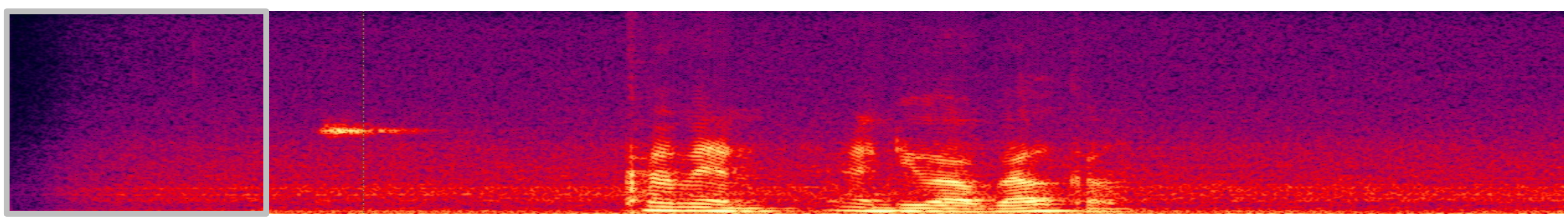
Let's do it in a different order



- Subsequently the rest of the neurons in the first layer operate on the first block
  - And the downstream layers as well
- Would the output of the MLP at the first block be different?
  - The fact that the first neuron has already evaluated the future blocks does not affect the output of that neuron, or the network itself, at the current block

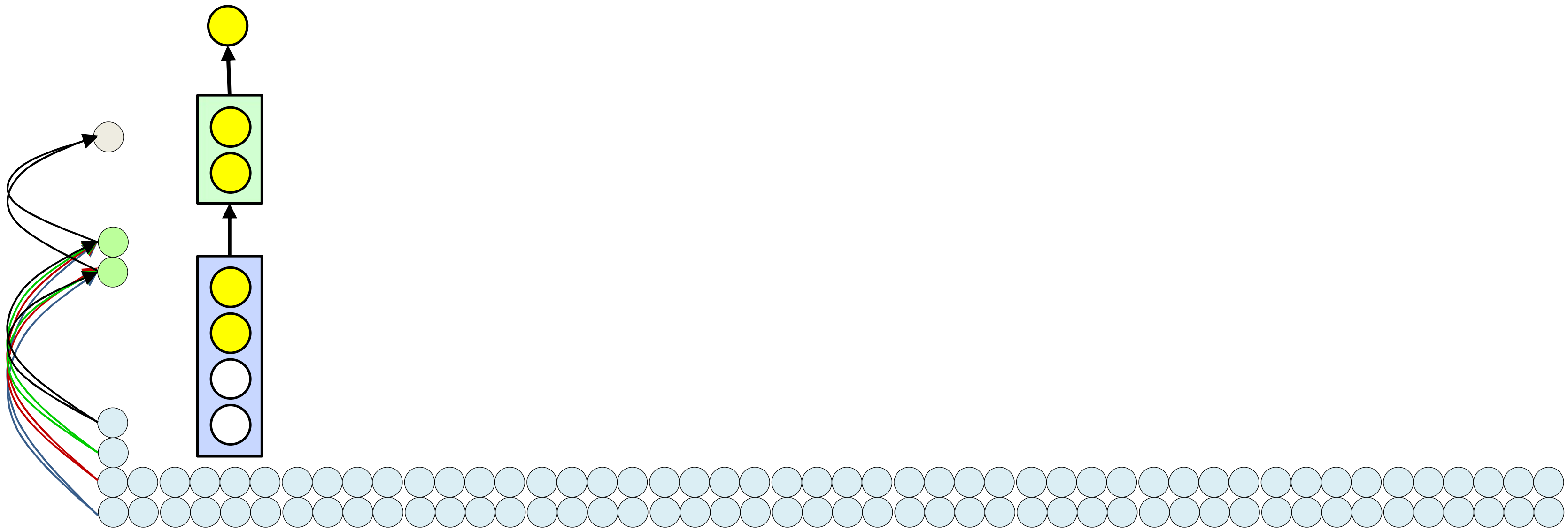
Let’s do it in a different order

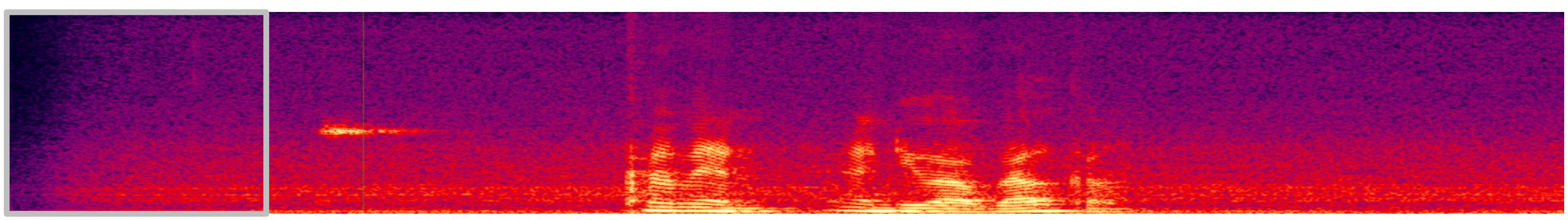




- What about now?
- The second neuron too has fully evaluated the entire input before the rest of the network evaluates the first block
  - This too should not change the output of the network for the first block

Let’s do it in a different order



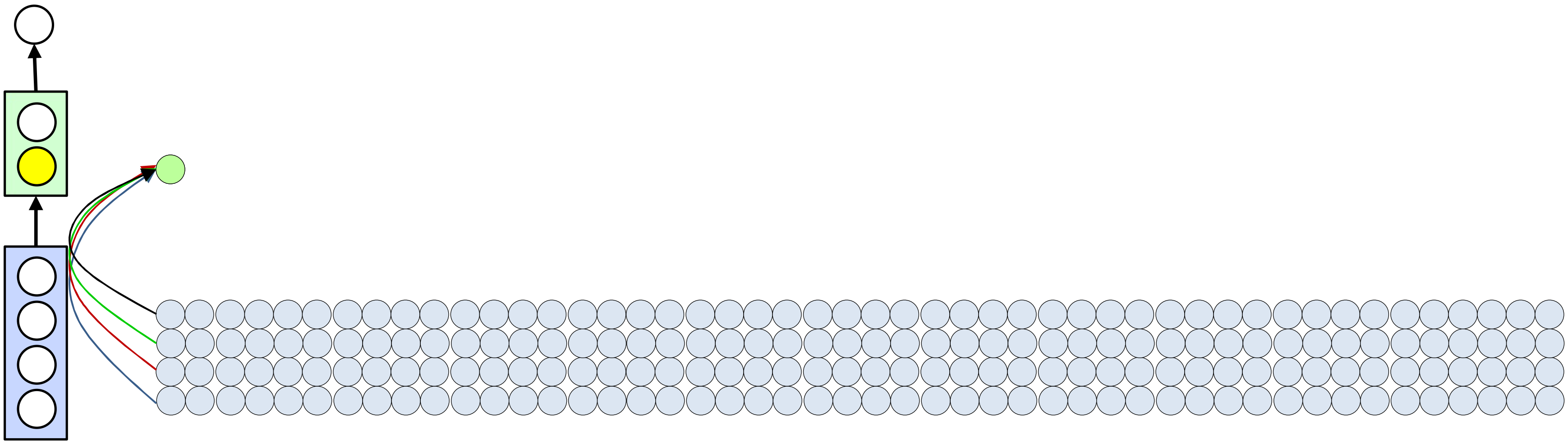


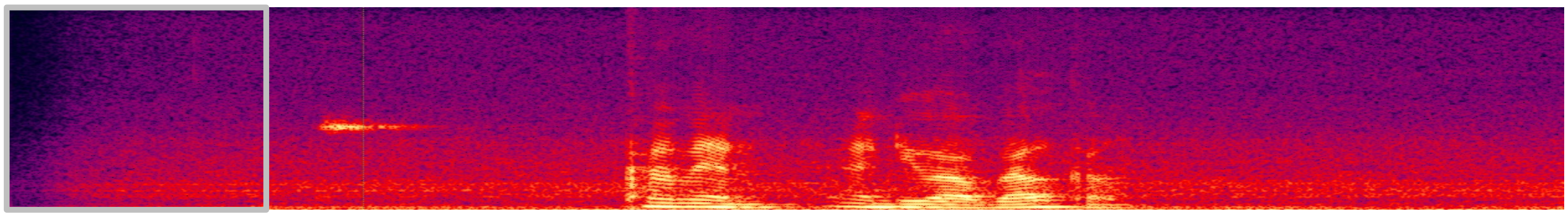
- What about now?
- The second neuron too has fully evaluated the entire input before the rest of the network evaluates the first block
  - This too should not change the output of the network for the first block

Let’s do it in a different order

- In fact if *all* of the neurons in the first layer fully evaluate the entire input before the rest of the network evaluates the first block, this will not change the output of the network at the first block

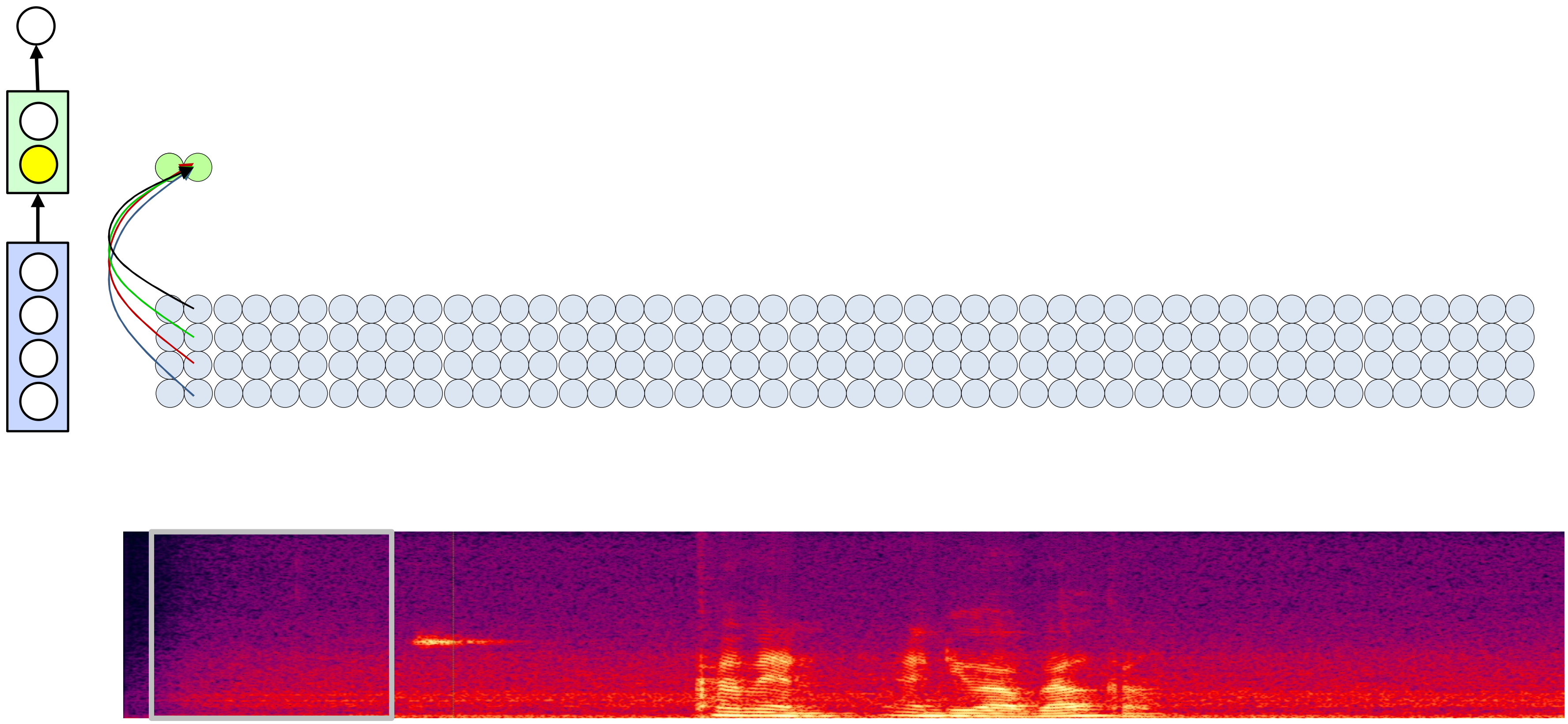
Let’s do it in a different order





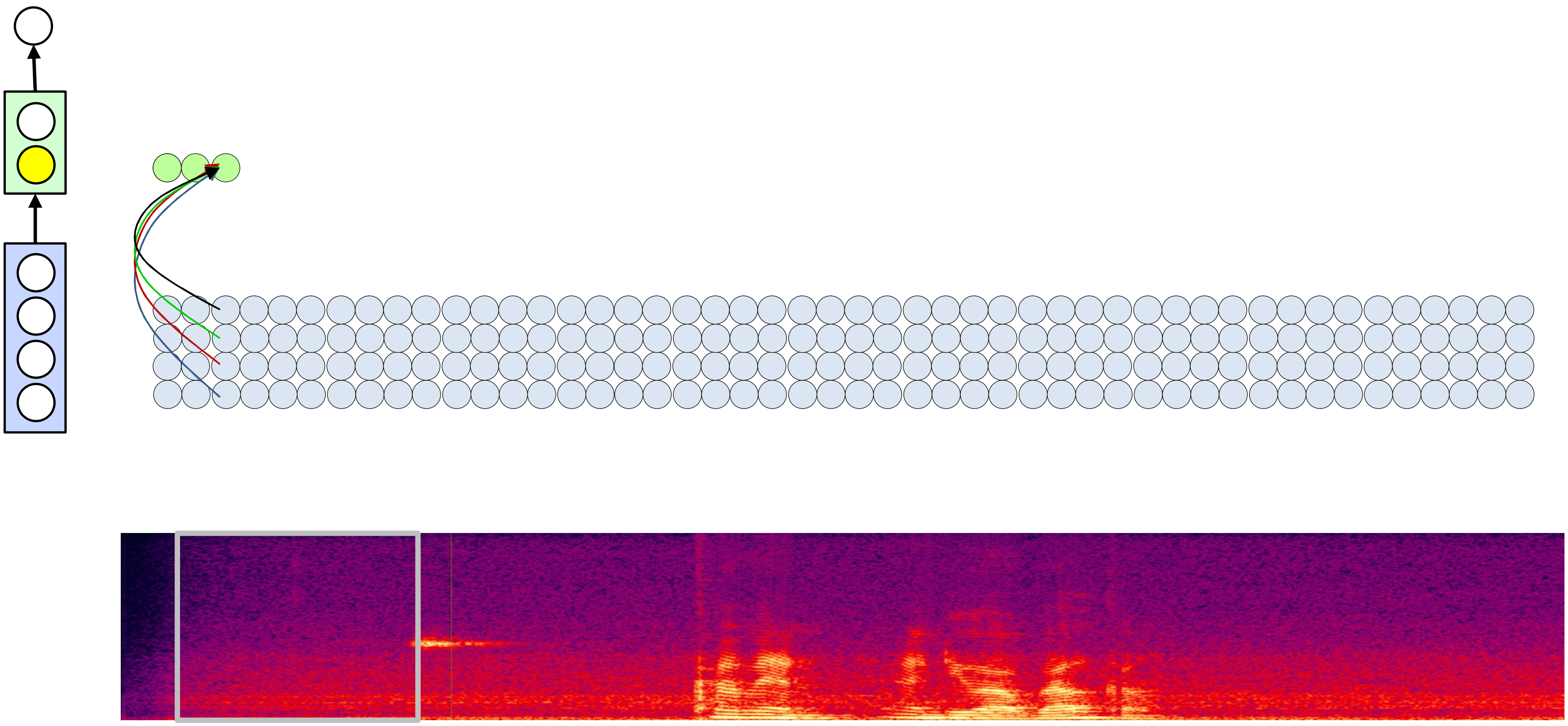
- But now, since the first layer neurons have already produced outputs for every location, each neuron in the second layer can go ahead and produce outputs for every position without waiting for the rest of the net
  - “Scan” the outputs of the first layer neurons

Let’s do it in a different order



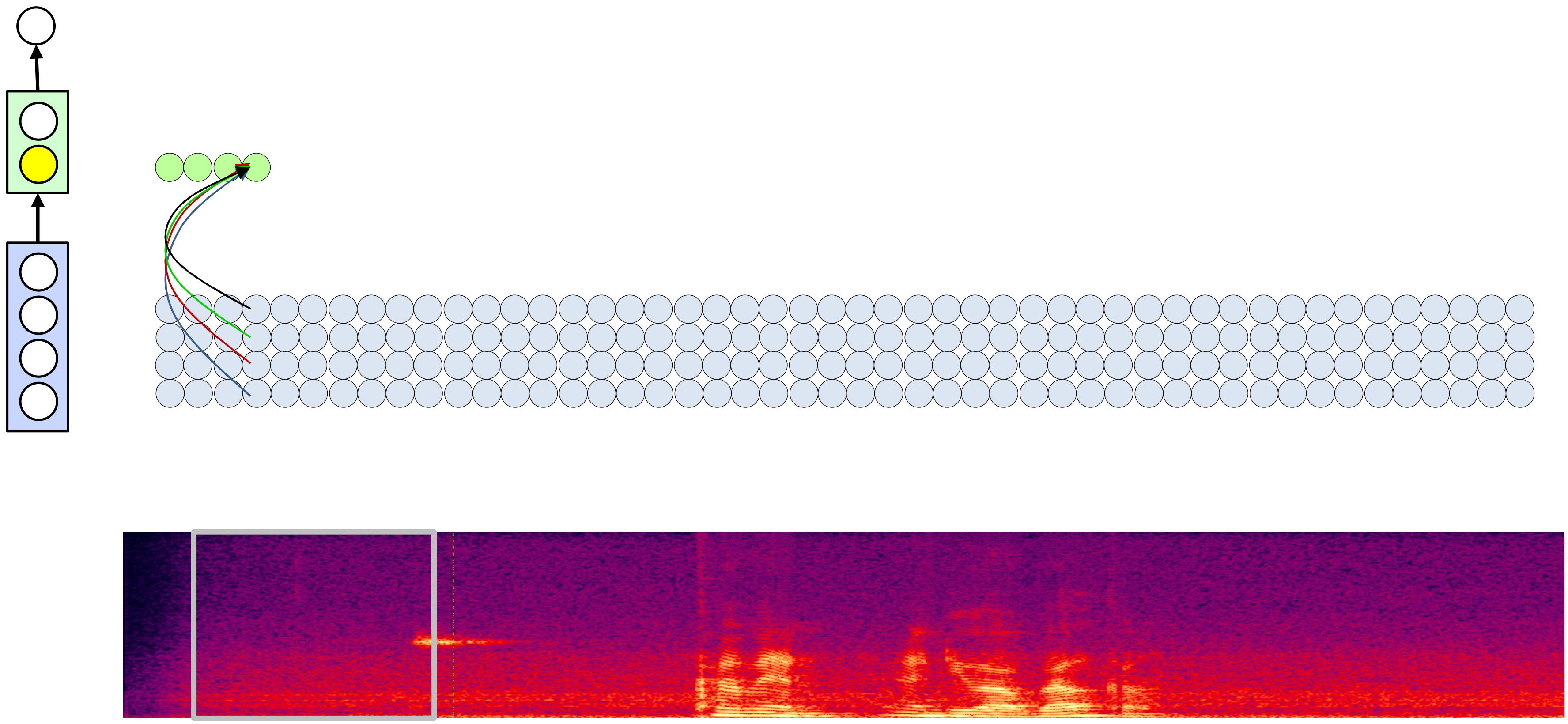
- But now, since the first layer neurons have already produced outputs for every location, each neuron in the second layer can go ahead and produce outputs for every position without waiting for the rest of the net
  - “Scan” the outputs of the first layer neurons

Let's do it in a different order



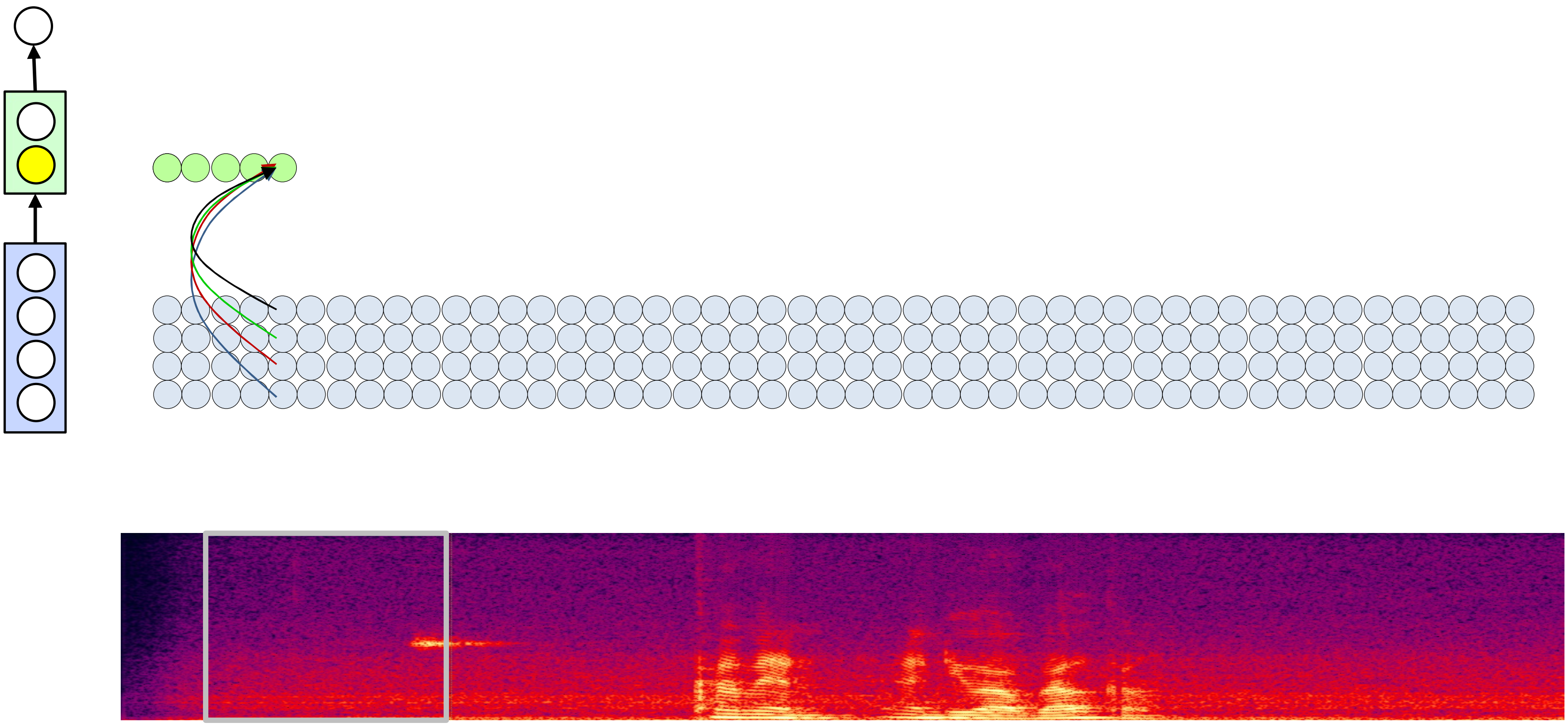
- But now, since the first layer neurons have already produced outputs for every location, each neuron in the second layer can go ahead and produce outputs for every position without waiting for the rest of the net
  – "Scan" the outputs of the first layer neurons

Let’s do it in a different order



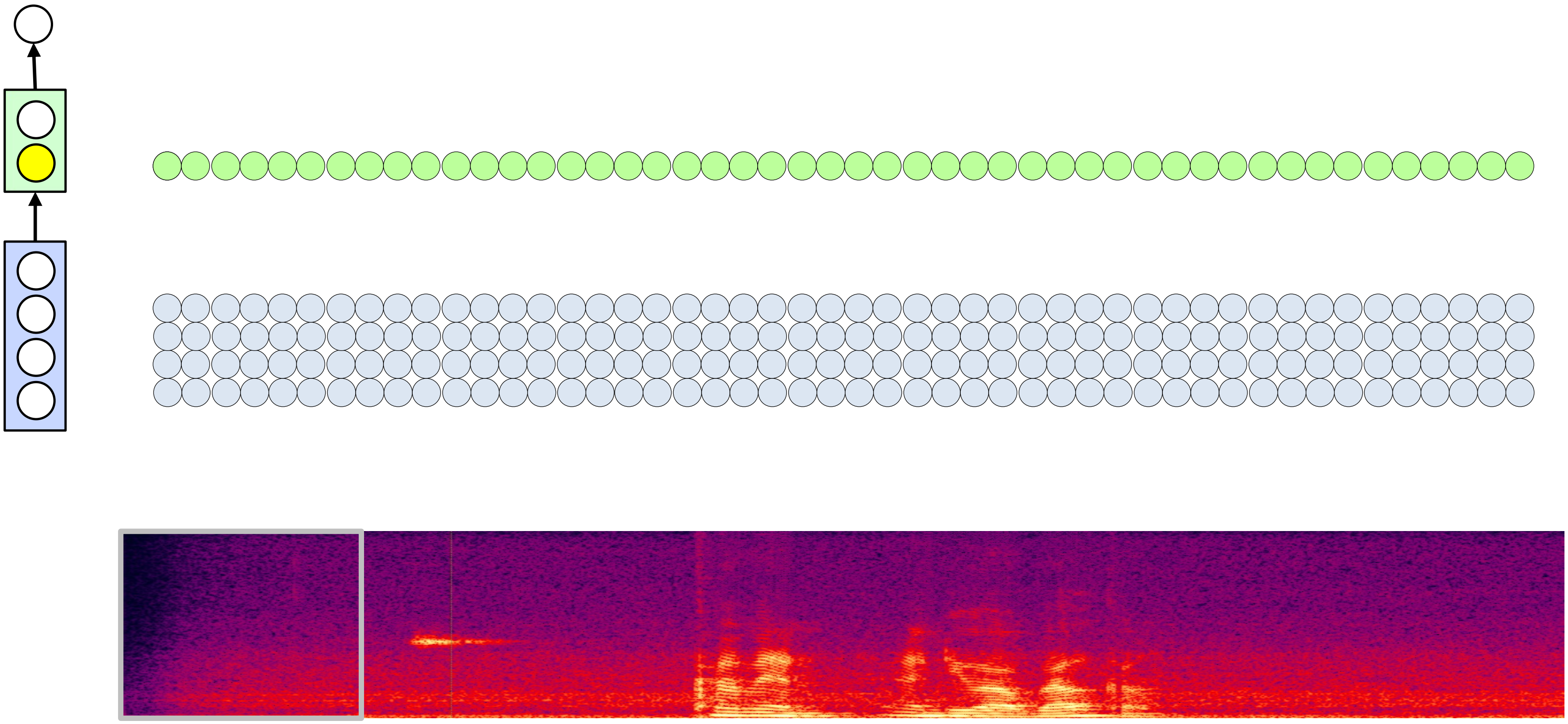
- But now, since the first layer neurons have already produced outputs for every location, each neuron in the second layer can go ahead and produce outputs for every position without waiting for the rest of the net
  - “Scan” the outputs of the first layer neurons

Let’s do it in a different order



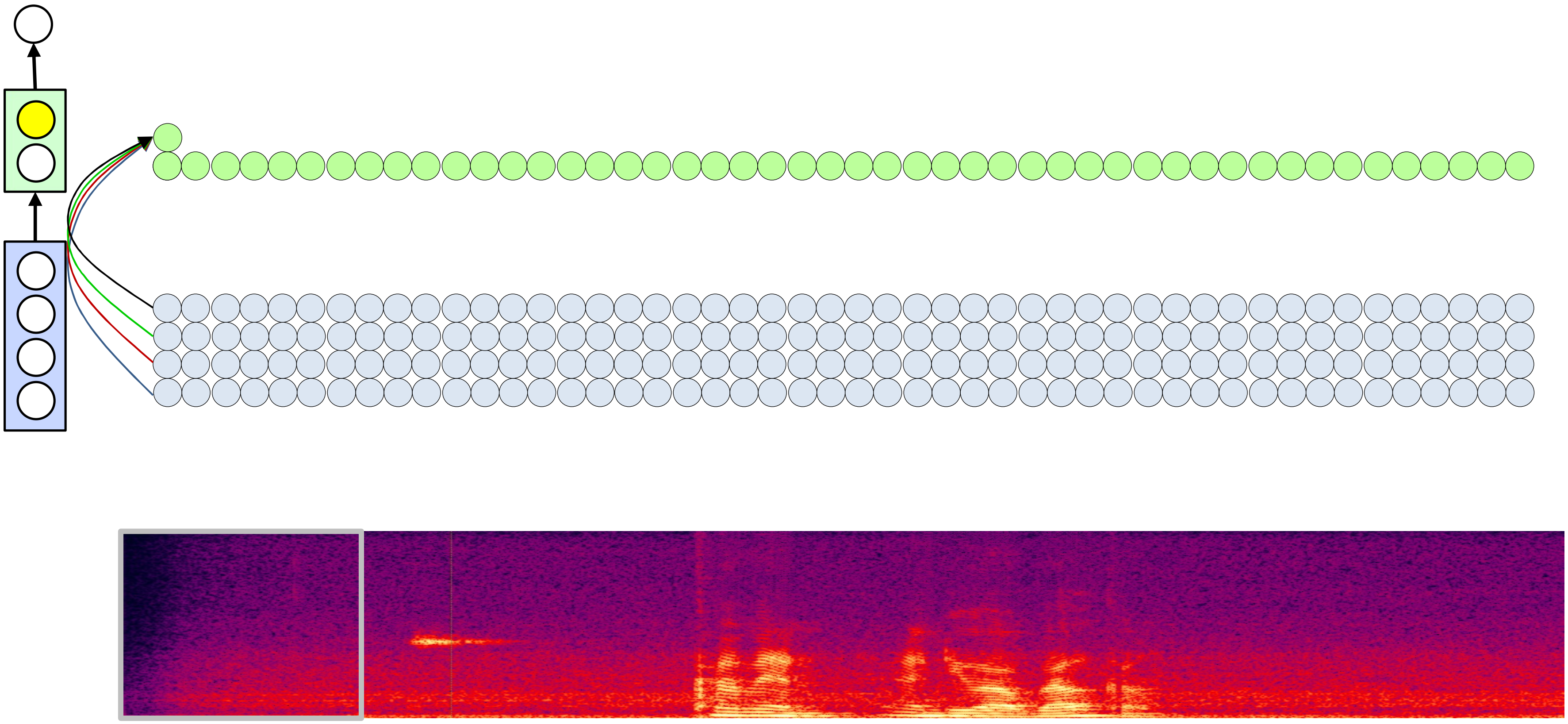
- But now, since the first layer neurons have already produced outputs for every location, each neuron in the second layer can go ahead and produce outputs for every position without waiting for the rest of the net
  - “Scan” the outputs of the first layer neurons

Let’s do it in a different order



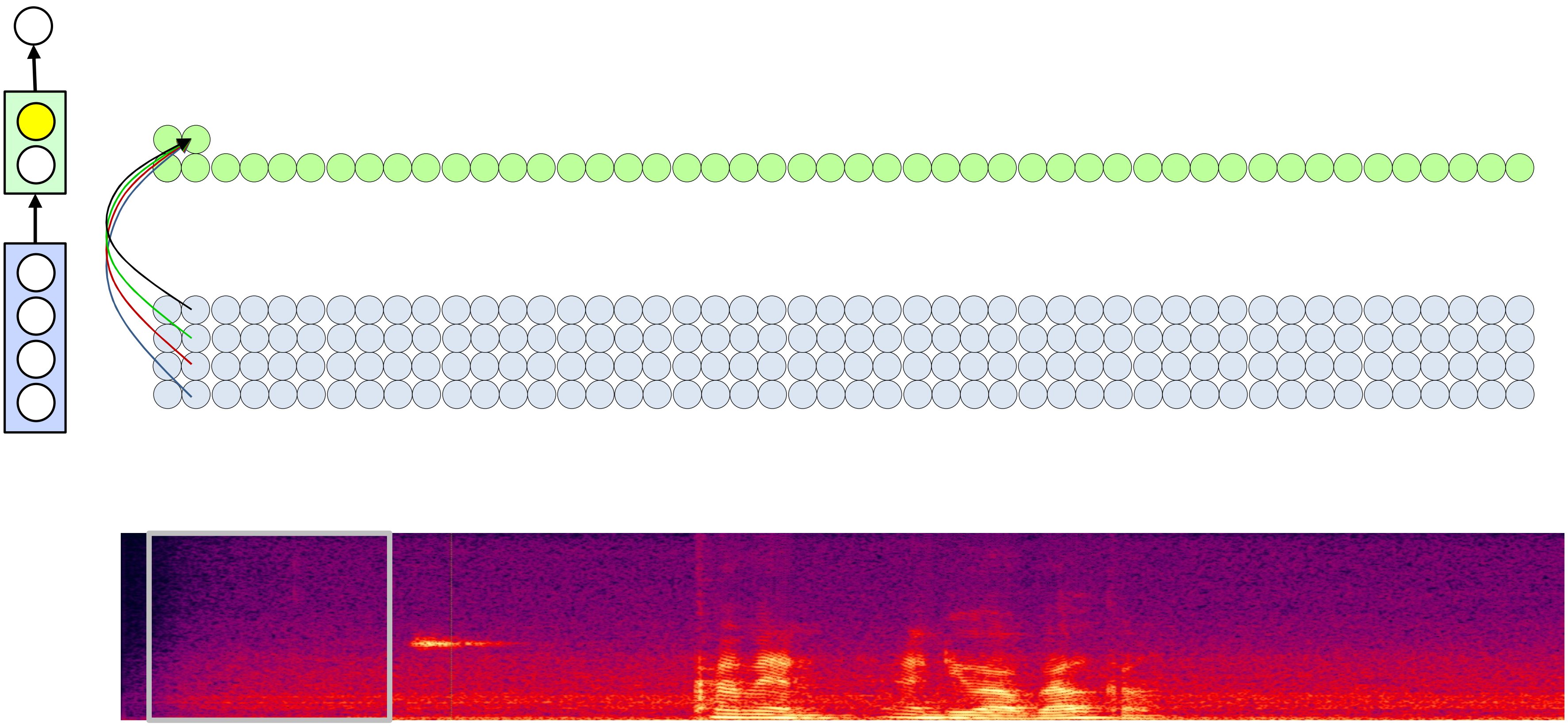
- But now, since the first layer neurons have already produced outputs for every location, each neuron in the second layer can go ahead and produce outputs for every position without waiting for the rest of the net
  - “Scan” the outputs of the first layer neurons

Let's do it in a different order



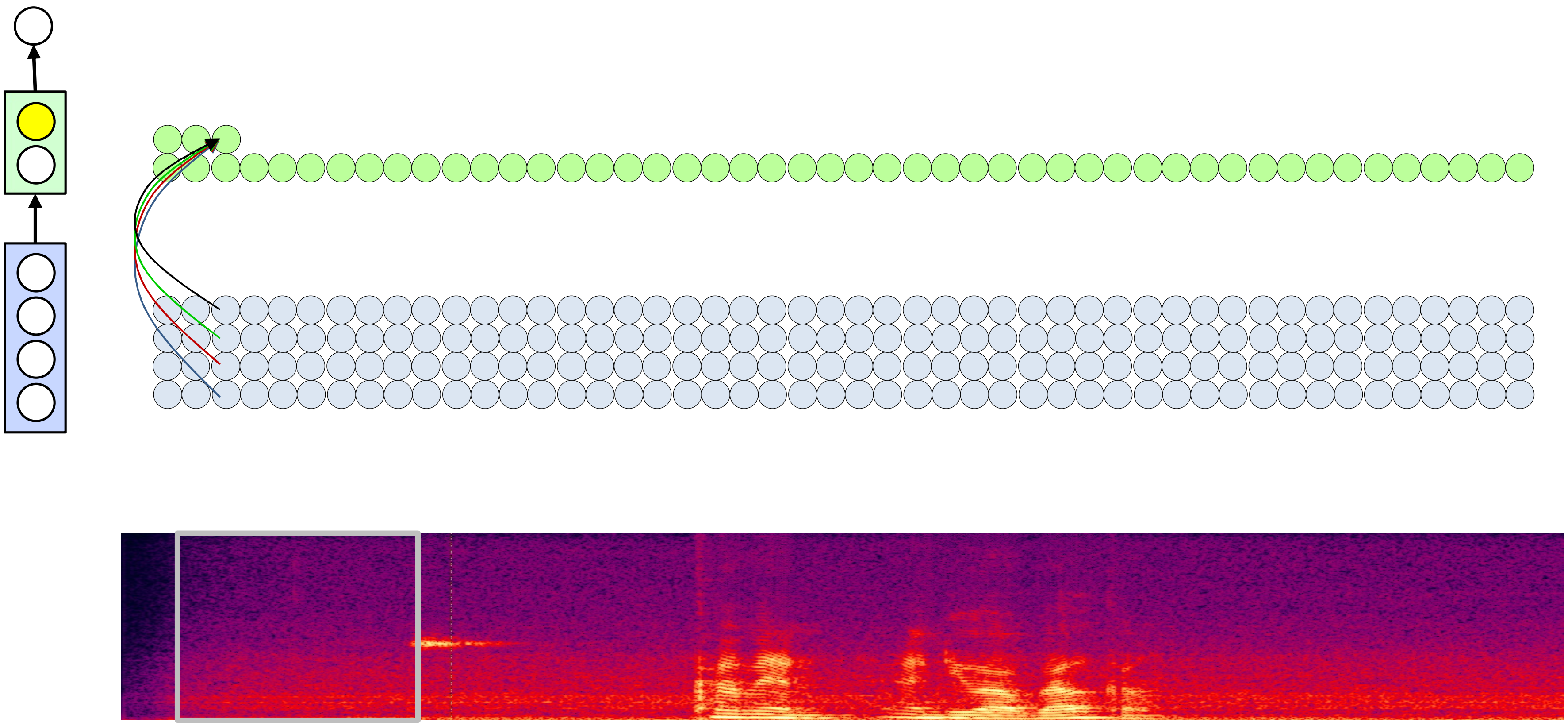
- But now, since the first layer neurons have already produced outputs for every location, each neuron in the second layer can go ahead and produce outputs for every position without waiting for the rest of the net
  - "Scan" the outputs of the first layer neurons

Let’s do it in a different order



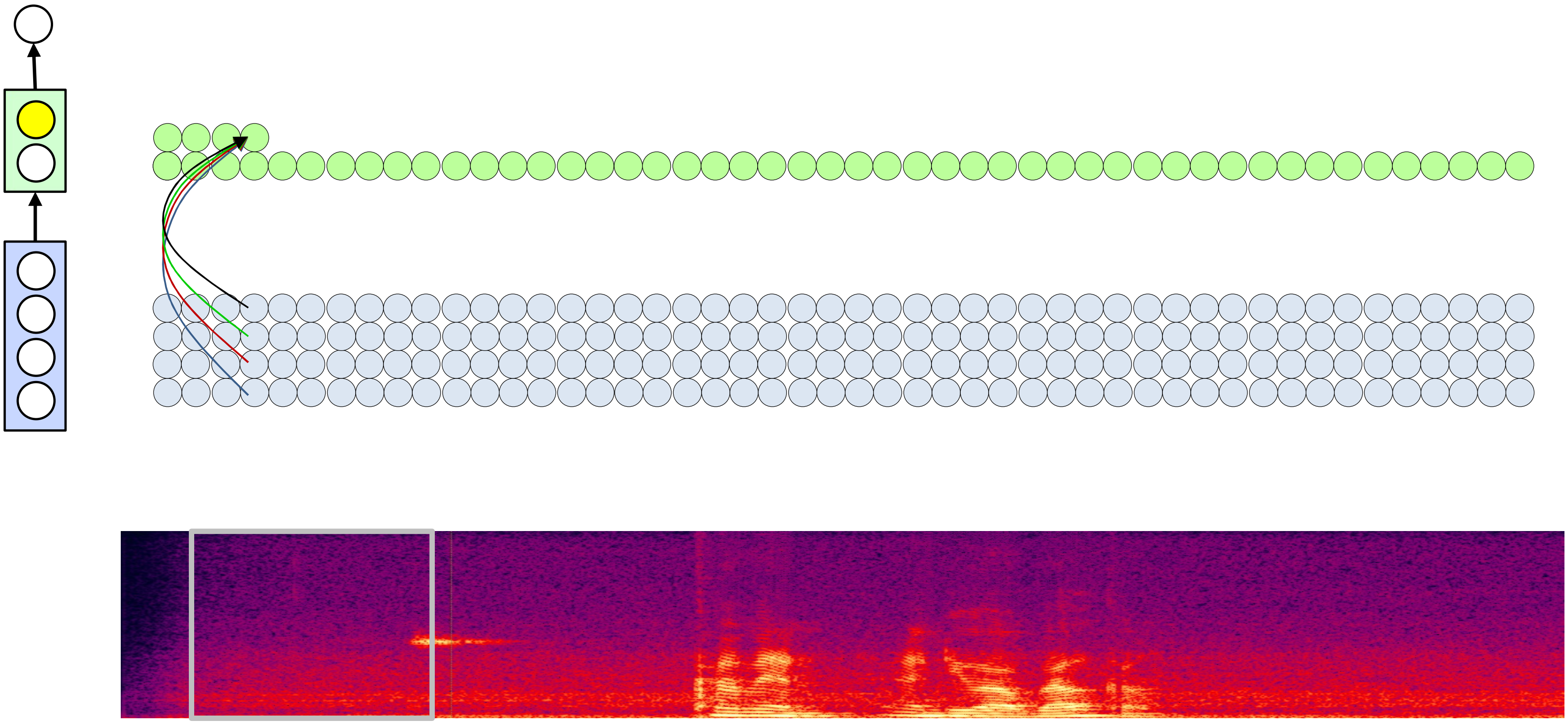
- But now, since the first layer neurons have already produced outputs for every location, each neuron in the second layer can go ahead and produce outputs for every position without waiting for the rest of the net
  - “Scan” the outputs of the first layer neurons

Let’s do it in a different order



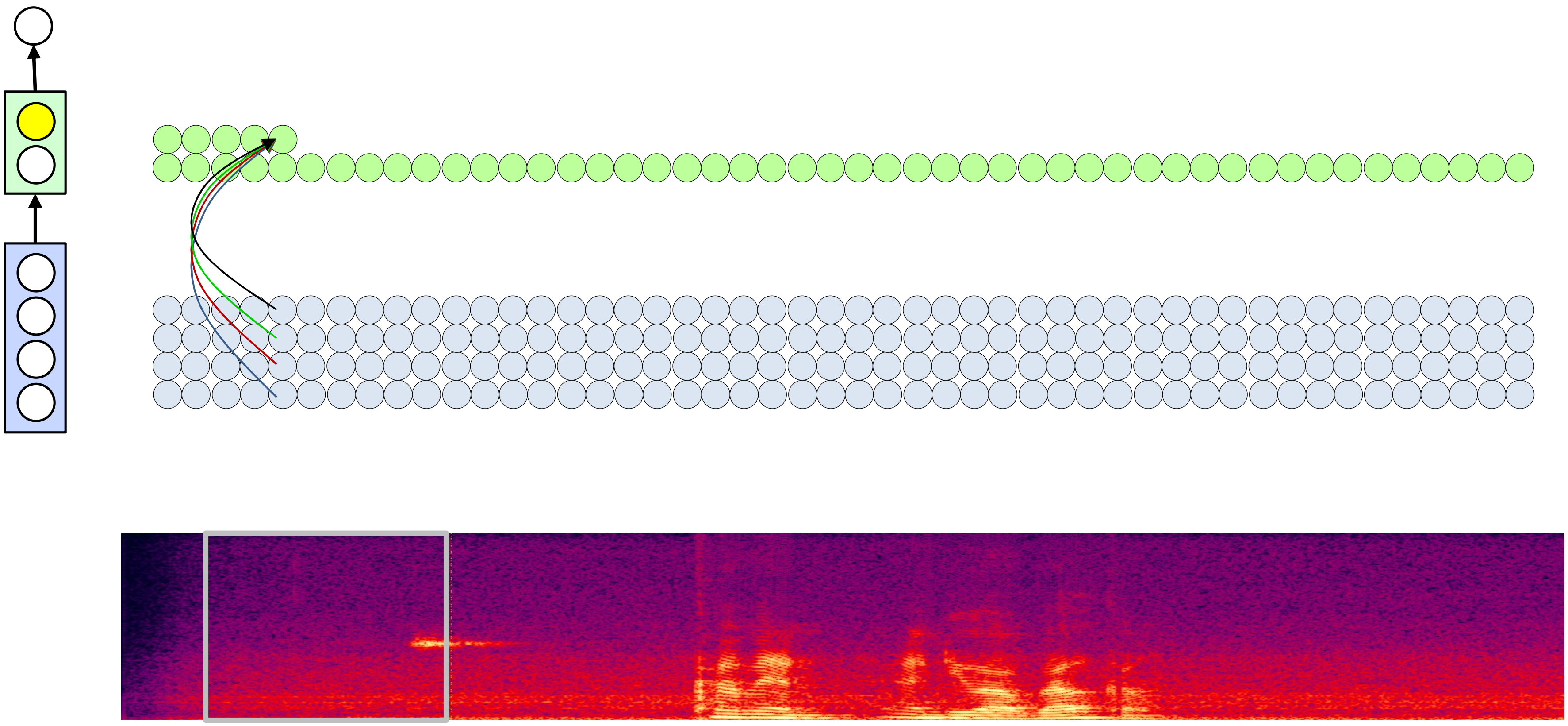
- But now, since the first layer neurons have already produced outputs for every location, each neuron in the second layer can go ahead and produce outputs for every position without waiting for the rest of the net
  - “Scan” the outputs of the first layer neurons

Let’s do it in a different order



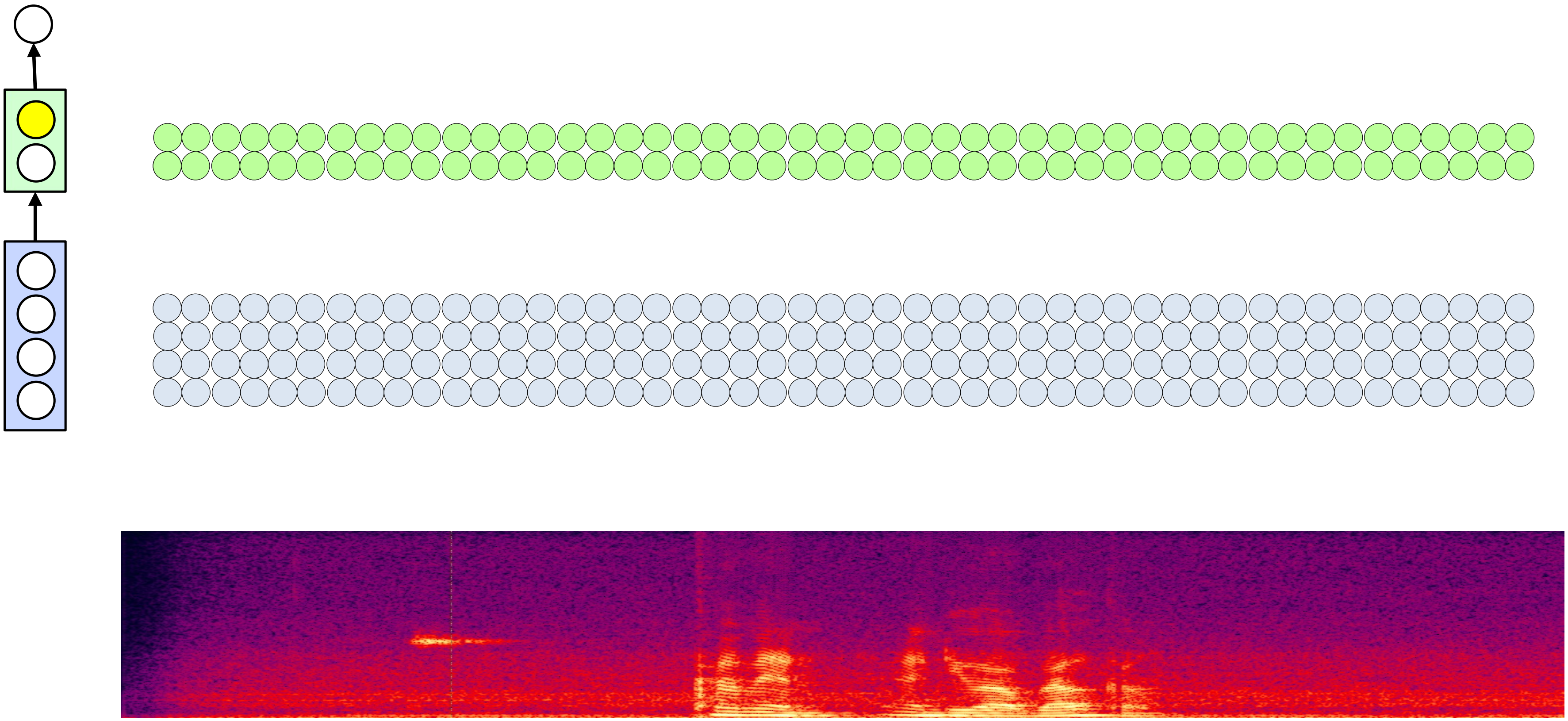
- But now, since the first layer neurons have already produced outputs for every location, each neuron in the second layer can go ahead and produce outputs for every position without waiting for the rest of the net
  - “Scan” the outputs of the first layer neurons

Let’s do it in a different order



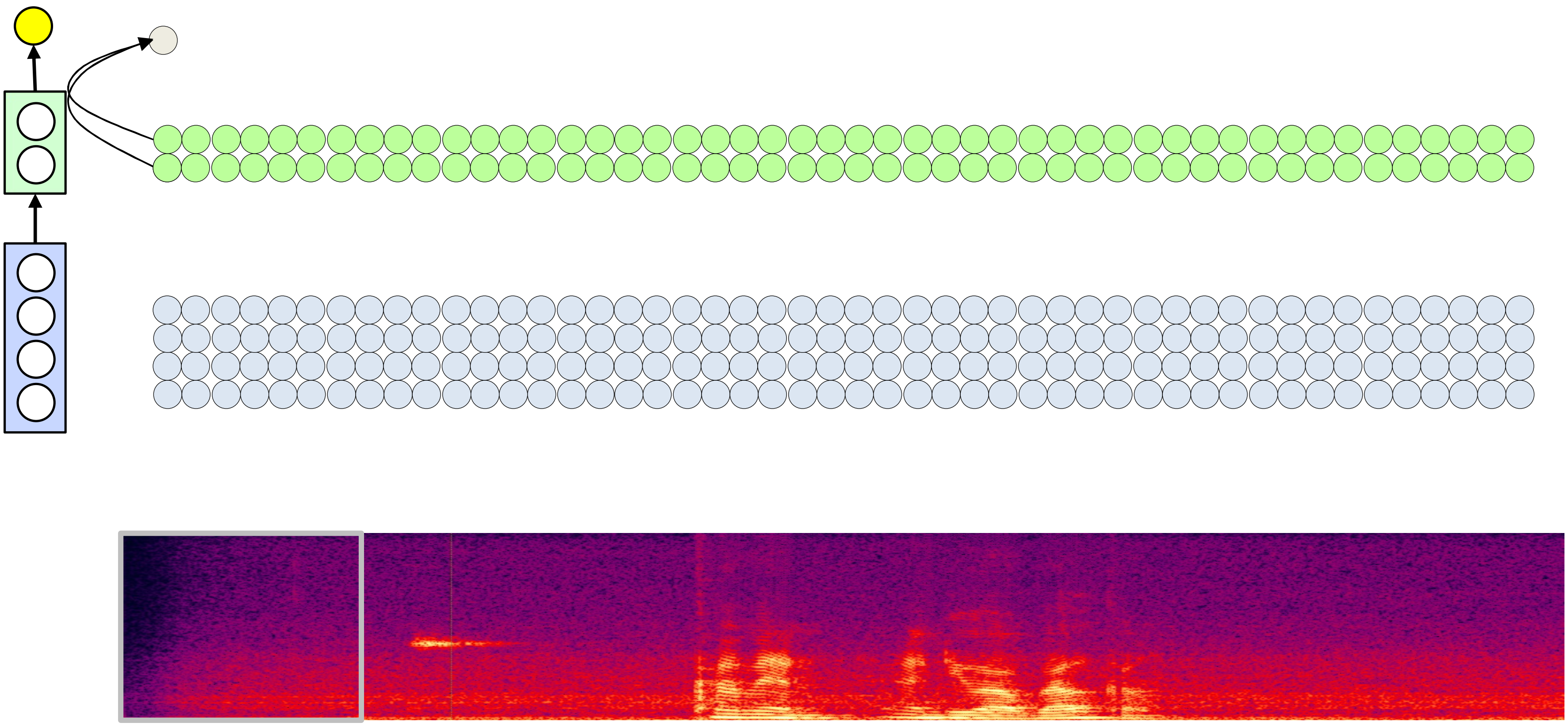
- But now, since the first layer neurons have already produced outputs for every location, each neuron in the second layer can go ahead and produce outputs for every position without waiting for the rest of the net
  - “Scan” the outputs of the first layer neurons

Let’s do it in a different order



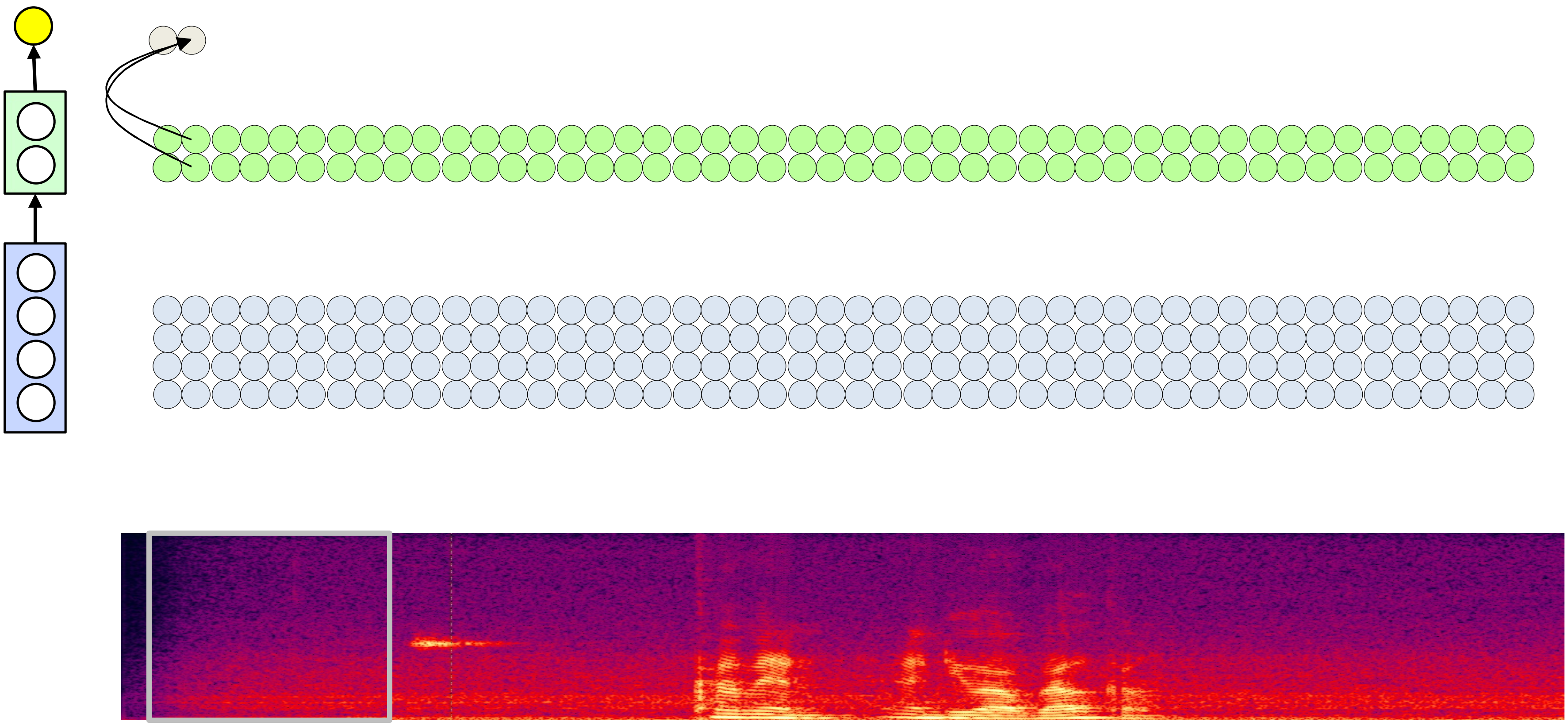
- But now, since the first layer neurons have already produced outputs for every location, each neuron in the second layer can go ahead and produce outputs for every position without waiting for the rest of the net
  - “Scan” the outputs of the first layer neurons

Let's do it in a different order



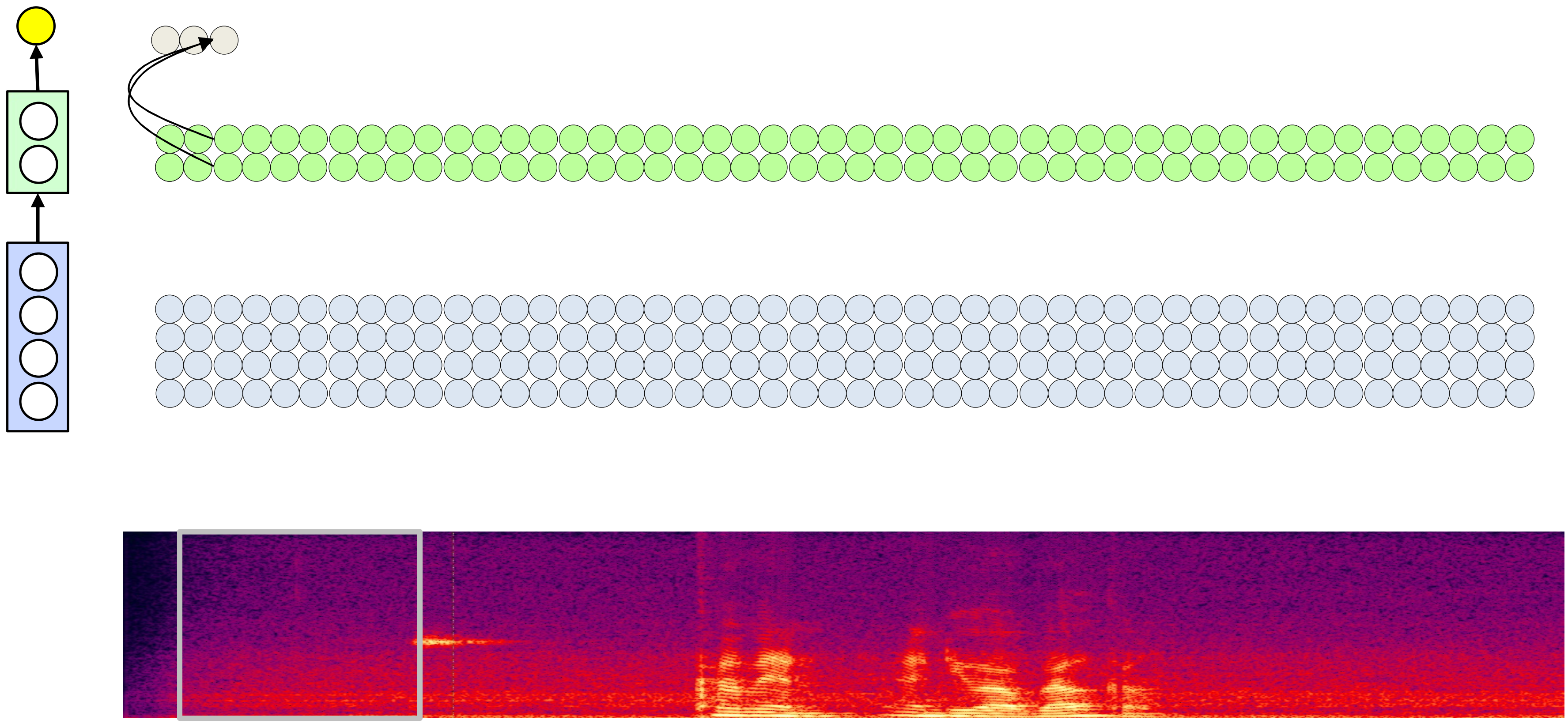
- At each position the output layer neurons can now operate on the outputs of the penultimate layer and produce the correct classification for the corresponding block!

Let's do it in a different order



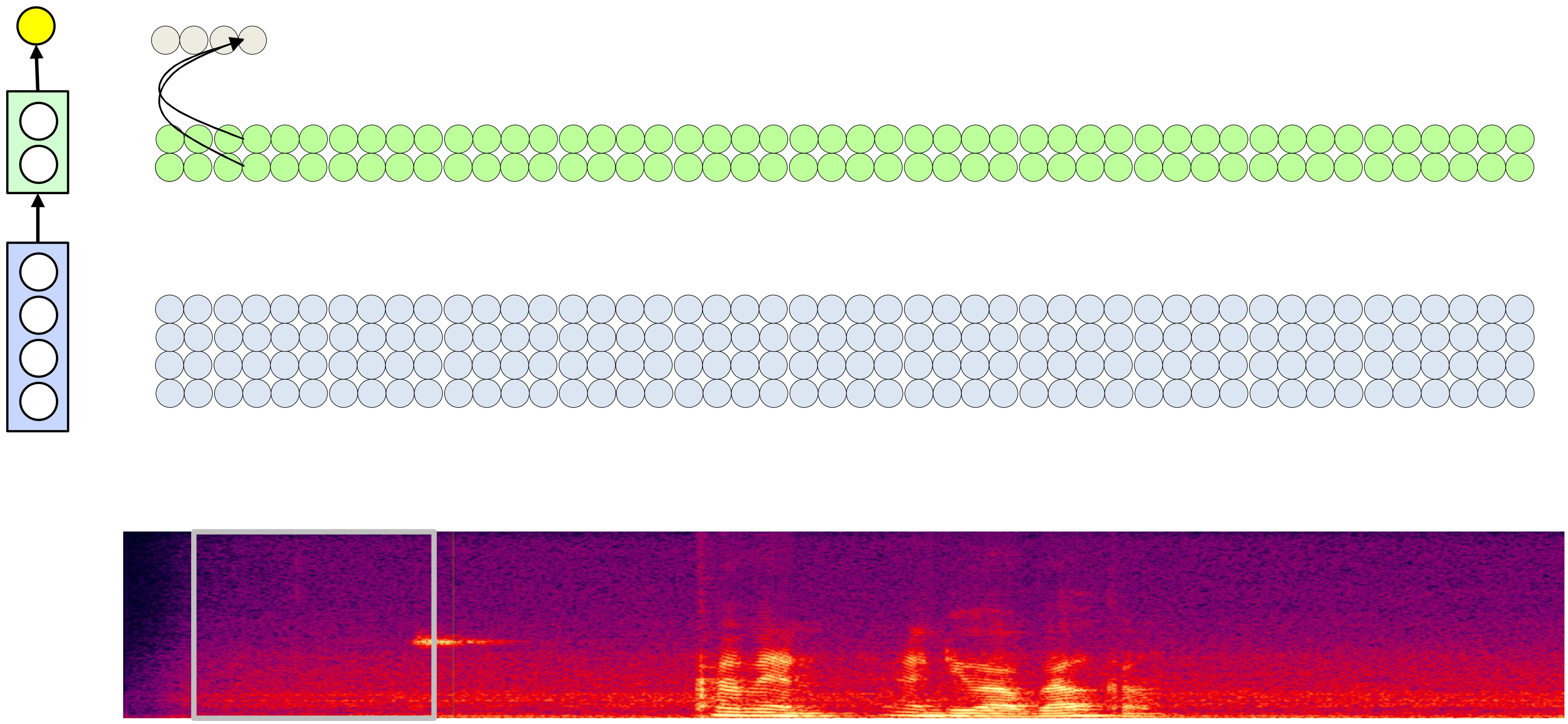
- At each position the output layer neurons can now operate on the outputs of the penultimate layer and produce the correct classification for the corresponding block!

Let’s do it in a different order



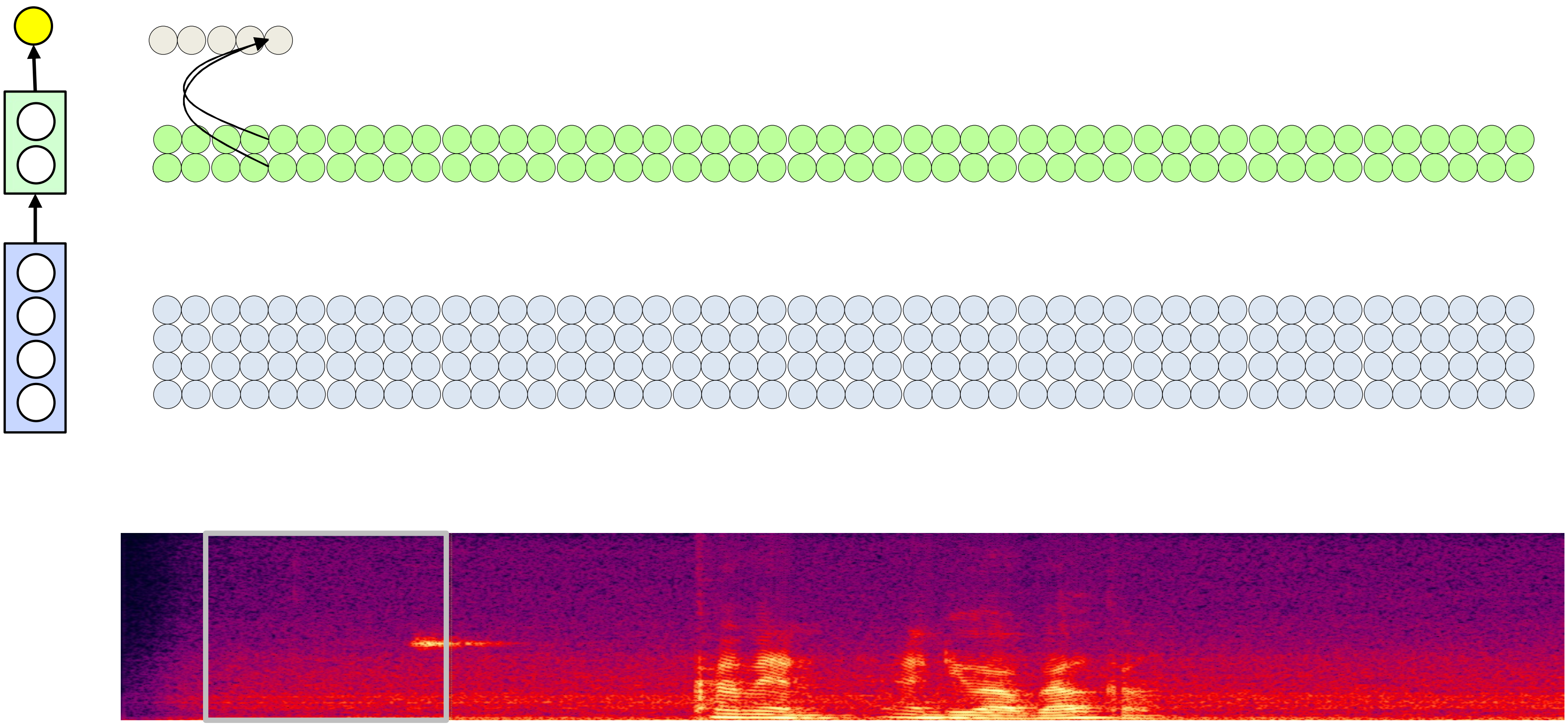
- At each position the output layer neurons can now operate on the outputs of the penultimate layer and produce the correct classification for the corresponding block!

Let's do it in a different order



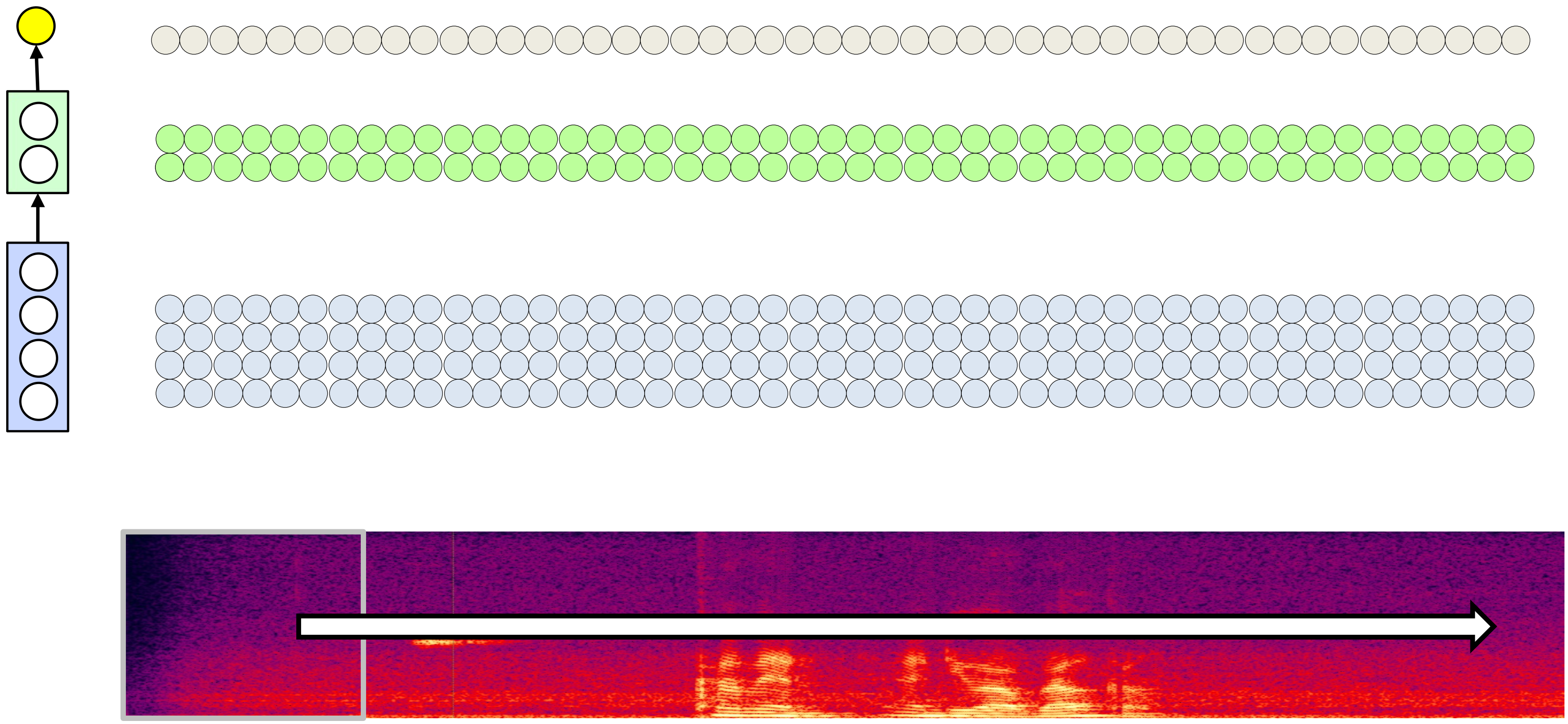
- At each position the output layer neurons can now operate on the outputs of the penultimate layer and produce the correct classification for the corresponding block!

Let's do it in a different order



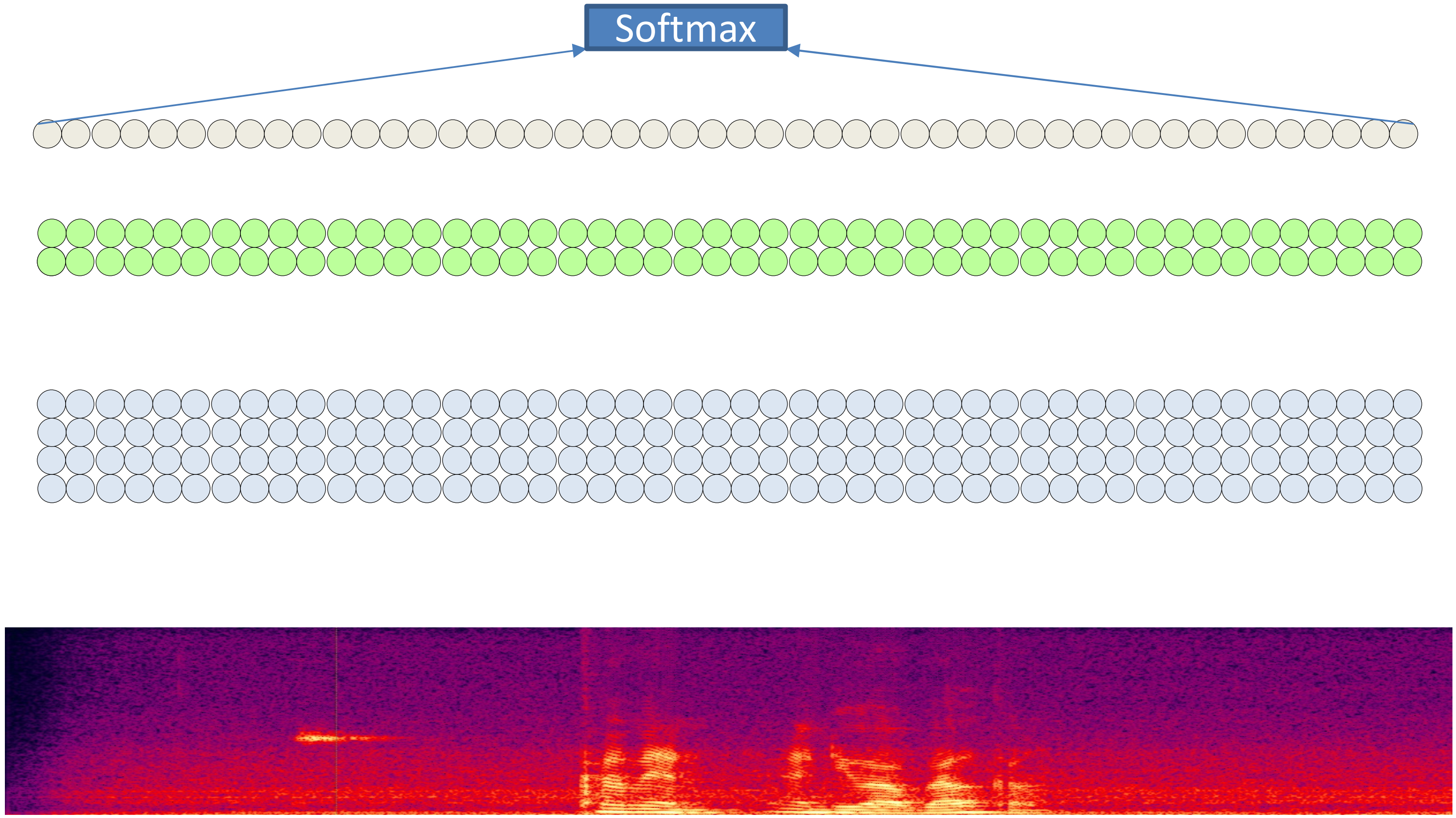
- At each position the output layer neurons can now operate on the outputs of the penultimate layer and produce the correct classification for the corresponding block!

Let's do it in a different order



- At each position the output layer neurons can now operate on the outputs of the penultimate layer and produce the correct classification for the corresponding block!

Let's do it in a different order



- At each position the output layer neurons can now operate on the outputs of the penultimate layer and produce the correct classification for the corresponding block!
  - The final softmax will give us the correct answer for the entire input

Scanning with an MLP

- K = width of "patch" evaluated by MLP

```
For t = 1:T-K+1
   XSegment = x(:, t:t+K-1)
   y(t) = MLP(XSegment)

Y = softmax(y(1)..y(T-K+1))
```

Scanning with an MLP

```
for t = 1:T-K+1
    for l = 1:L  # layers operate at location t
        for j = 1:Dl
            if (l == 1) #first layer operates on input
                y(0,:,t) = x(:, t:t+K-1)
            end
            z(l,j,t) = b(l,j)  # bias
            for i = 1:Dl-1
                z(l,j,t) += w(l,i,j)y(l-1,i,t)
            y(l,j,t) = activation(z(l,j,t))

Y = softmax( y(L,:,1)..y(L,:,T-K+1) )
```

Scanning with an MLP

Over time

Over layers

```
for t = 1:T-K+1
    for l = 1:L # layers operate at location t
        for j = 1:Dl
            if (l == 1) #first layer operates on input
                y(0,:,t) = x(:, t:t+K-1)
            end
            z(l,j,t) = b(l,j)
            for i = 1:Dl-1
                z(l,j,t) += w(l,i,j)y(l-1,i,t)
            y(l,j,t) = activation(z(l,j,t))

Y = softmax( y(L,:,1)..y(L,:,T-K+1) )
```

Scanning with an MLP

Over time

Over layers

```
for t = 1:T-K+1
  for l = 1:L  # layers operate at location t
    for j = 1:D_l
      if (l == 1) #first layer operates on input
         y(0,:,t) = x(:, t:t+K-1)
      end
      z(l,j,t) = b(l,j)
      for i = 1:D_l-1
          z(l,j,t) += w(l,i,j)y(l-1,i,t)
      y(l,j,t) = activation(z(l,j,t))

Y = softmax(y(L,:,1)..y(L,:,T-K+1) )
```

Scanning with an MLP

Over layers

Over time

```
for l = 1:L          # layers operate at location t
  for t = 1:T-K+1
    for j = 1:D_l
       if (l == 1) #first layer operates on input
          y(0,:,t) = x(:, t:t+K-1)
       end
       z(l,j,t) = b(l,j)
       for i = 1:D_l-1
           z(l,j,t) += w(l,i,j)y(l-1,i,t)
       y(l,j,t) = activation(z(l,j,t))

Y = softmax(y(L,:,1)..y(L,:,T-K+1) )
```

Scanning with an MLP

```
for l = 1:L    # layers operate at location t
    for t = 1:T-K+1
        for j = 1:D_l
           if (l == 1) #first layer operates on input
              y(0,:,t) = x(:, t:t+K-1)
           end
           z(l,j,t) = b(l,j)
           for i = 1:D_l-1
               z(l,j,t) += w(l,i,j)y(l-1,i,t)
           y(l,j,t) = activation(z(l,j,t))

Y = softmax(y(L,:,1)..y(L,:,T-K+1) )
```

Scanning with an MLP

```
for t = 1:T-K+1
    for l = 1:L  # layers operate at location t
        if (l == 1) #first layer operates on input
            y(0, t) = x(:, t:t+K-1)
        end
        z(l,t) = W(l)y(l-1,t) + b(l)
        y(l,t) = activation(z(l,t))

Y = softmax( y(L,:,1)..y(L,:,T-K+1) )
```

Scanning with an MLP

Over time

Over layers

```
for t = 1:T-K+1
   for l = 1:L   # layers operate at location t
      if (l == 1) #first layer operates on input
         y(0, t) = x(:, t:t+K-1)
      end
      z(l,t) = W(l)y(l-1,t) + b(l)
      y(l,t) = activation(z(l,t))

Y = softmax( y(L,:,1)..y(L,:,T-K+1) )
```

Scanning with an MLP

Over time

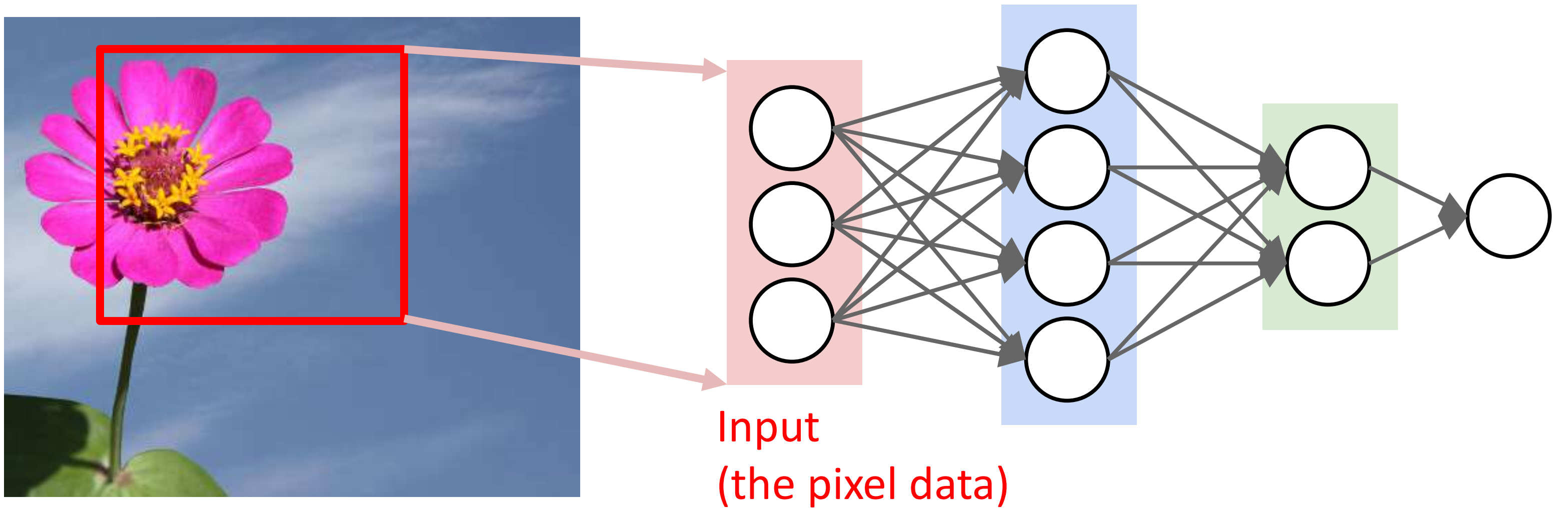
Over layers

```
for t = 1:T-K+1
    for l = 1:L   # layers operate at location t
        if (l == 1) #first layer operates on input
            y(0, t) = x(:, t:t+K-1)
        end
        z(l,t) = W(l)y(l-1,t) + b(l)
        y(l,t) = activation(z(l,t))

Y = softmax( y(L,:,1)..y(L,:,T-K+1) )
```

Scanning with an MLP: Vector notation

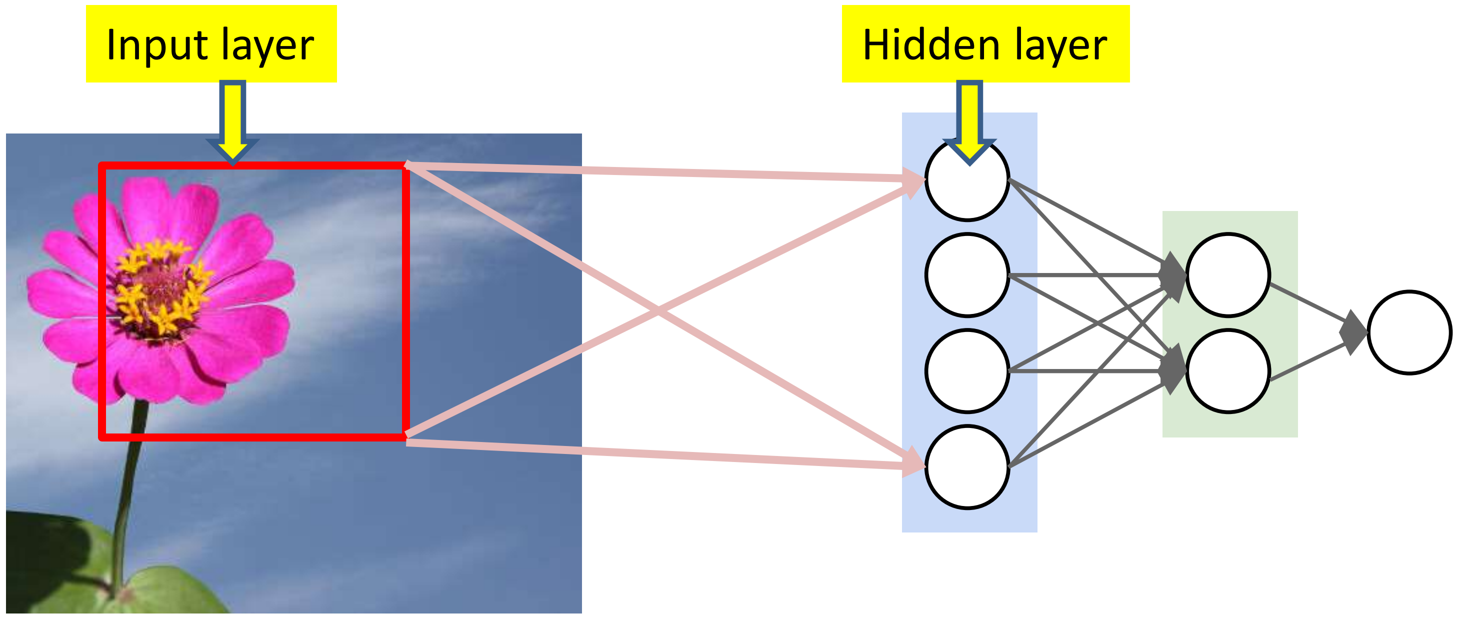
```
for l = 1:L   # layers operate at location t
    for t = 1:T-K+1
        if (l == 1) #first layer operates on input
            y(0, t) = x(:, t:t+K-1)
        end
        z(l,t) = W(l)y(l-1,t) + b(l)
        y(l,t) = activation(z(l,t))

Y = softmax( y(L,1)..y(L,T-K+1) )
```

Scanning in 2D: A closer look



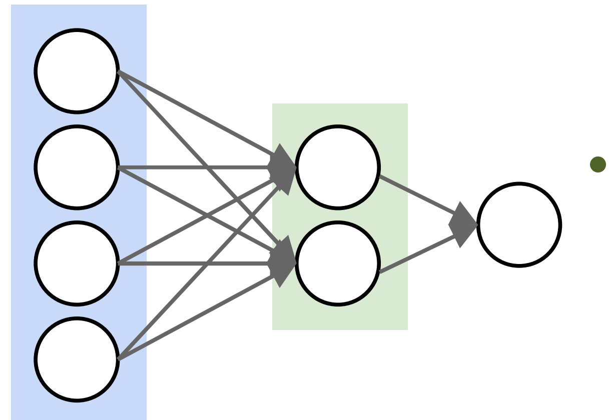
- *Scan* for the desired object
- At each location, the entire region is sent through an MLP

Scanning: A closer look



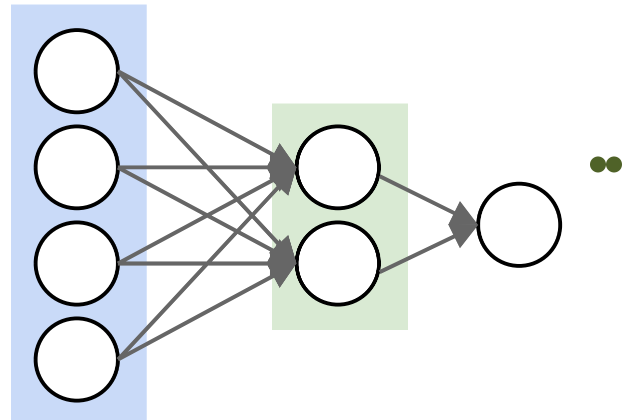
- The “input layer” is just the pixels in the image connecting to the hidden layer

Scanning: A closer look



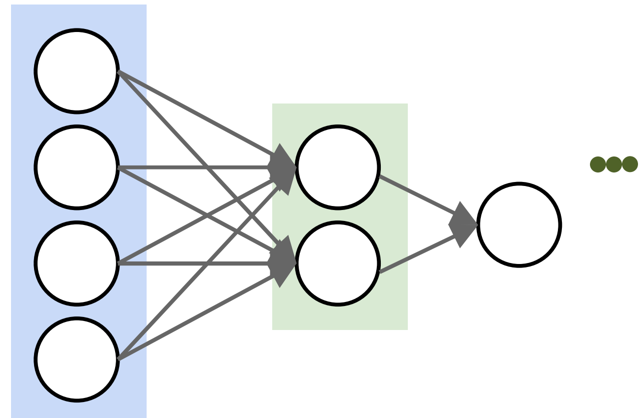
- Scanning: Analyze windows of pixels starting from top left, until the bottom right of the image
  - Produce an output for every window analyzed
  - Pass collection of outputs through a softmax

Scanning: A closer look



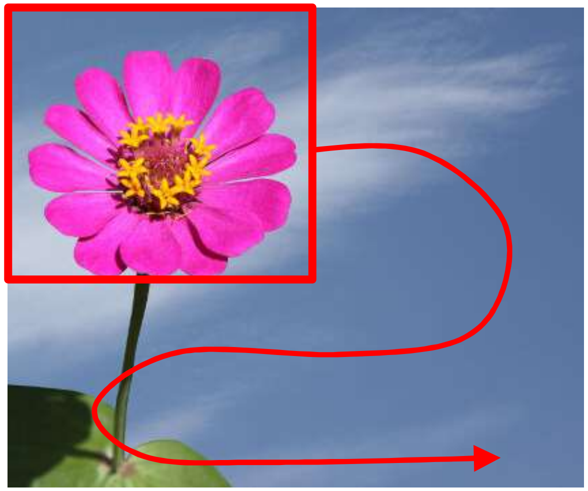
- Scanning: Analyze windows of pixels starting from top left, until the bottom right of the image
  - Produce an output for every window analyzed
  - Pass collection of outputs through a softmax

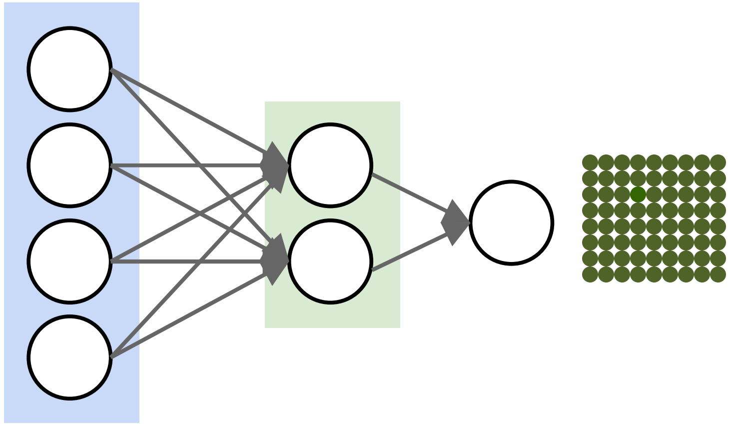
Scanning: A closer look



- Scanning: Analyze windows of pixels starting from top left, until the bottom right of the image
  - Produce an output for every window analyzed
  - Pass collection of outputs through a softmax

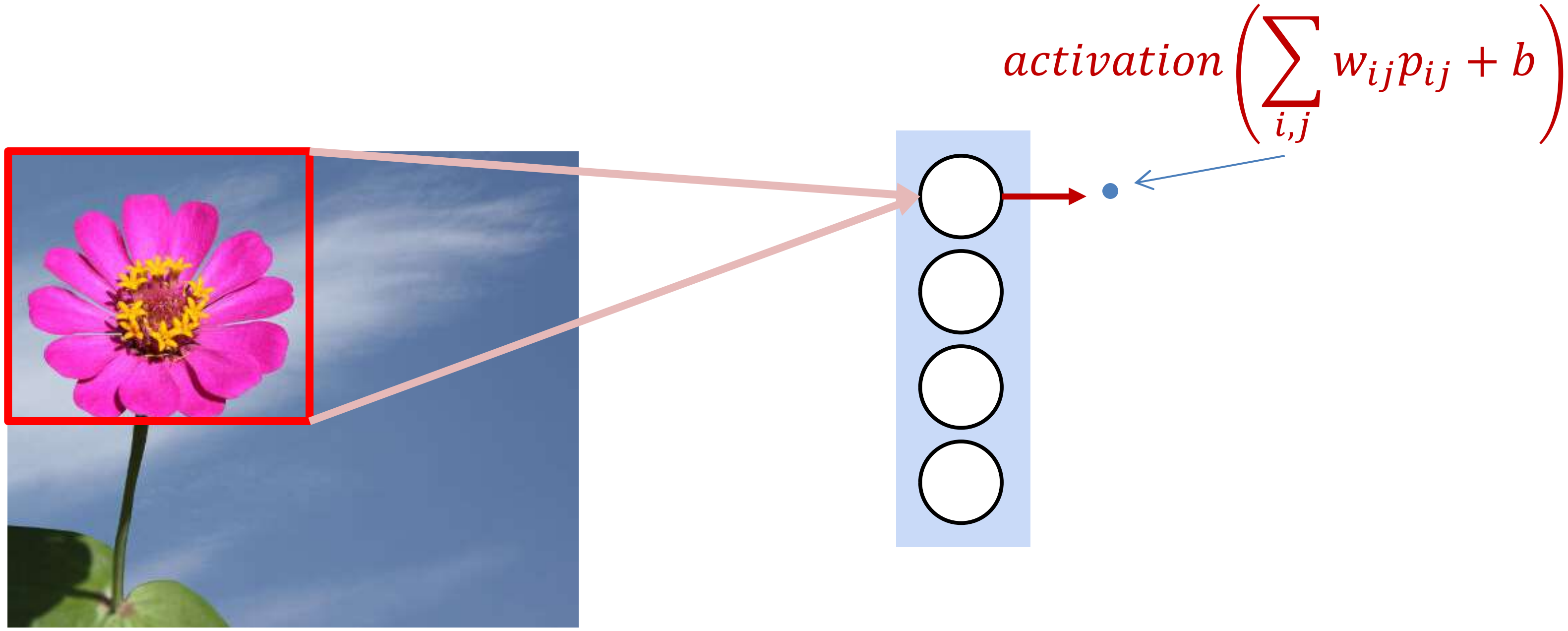
Scanning: A closer look





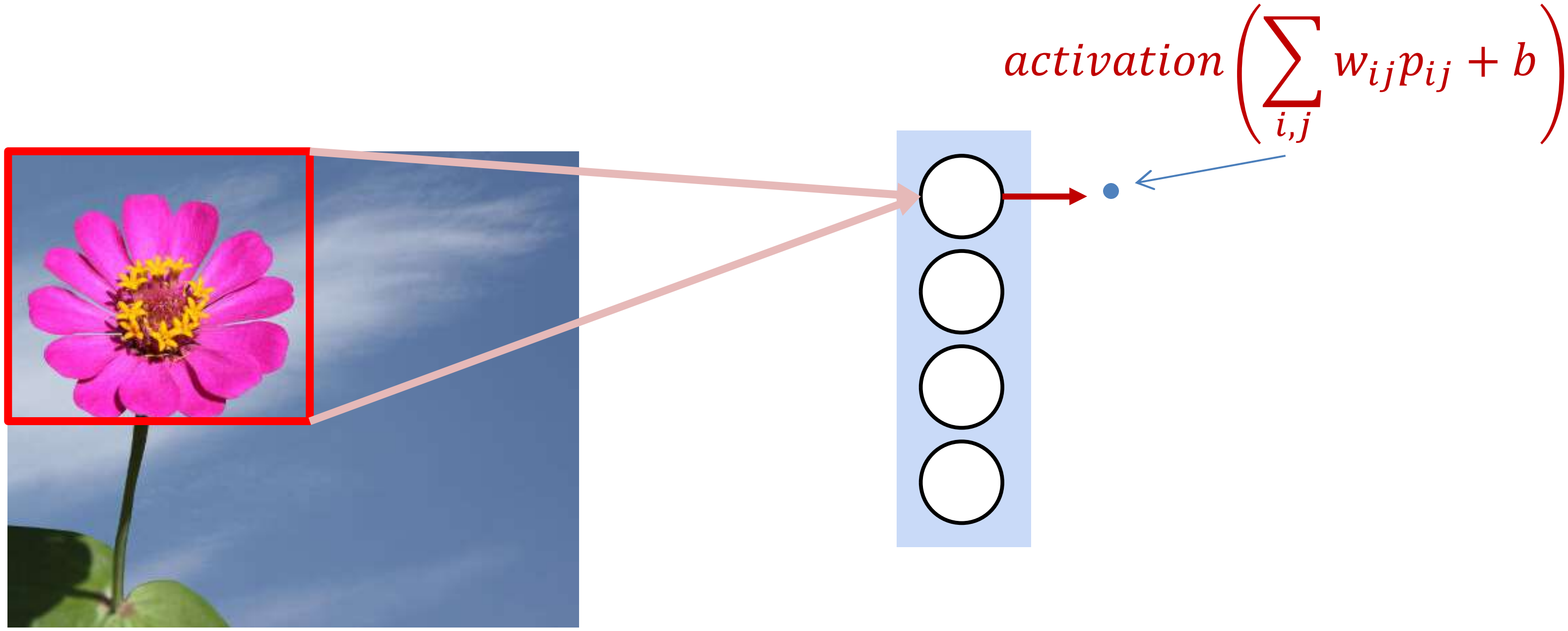
- Scanning: Analyze windows of pixels starting from top left, until the bottom right of the image
  - Produce an output for every window analyzed
  - Pass collection of outputs through a softmax

Scanning: A closer look



- Consider a single neuron in the first layer
  - At each position of the box, the neuron is evaluating a “window” of the picture at that location, as part of the classification for *that* region

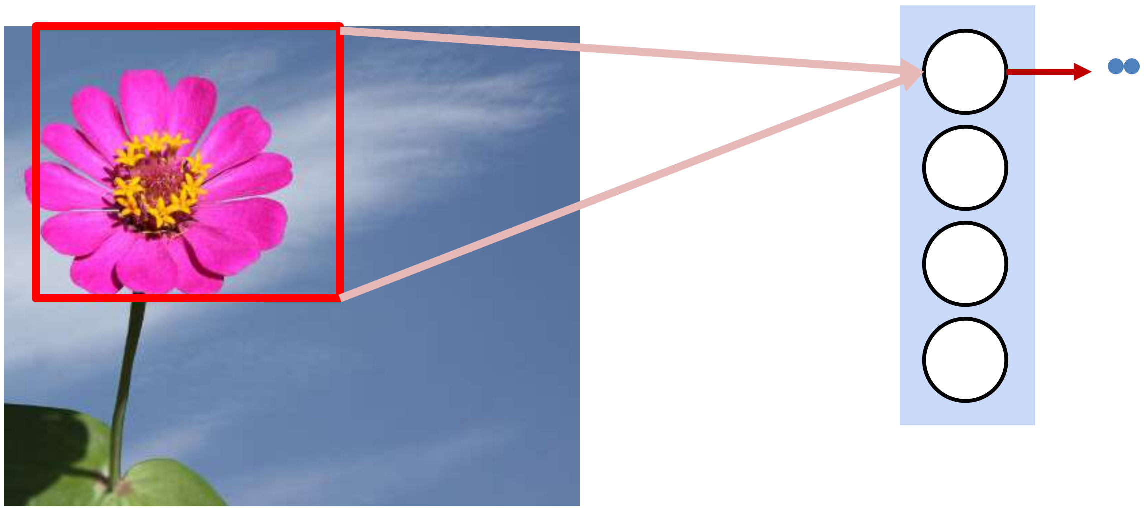
Scanning: A closer look



- Consider a single neuron in the first layer
  - At each position of the box, the neuron is evaluating a "window" of the picture at that location, as part of the classification for *that* region

- Let us compute the output of the first neuron for *all* the windows in the picture before computing the rest of the neurons
  - "Scanning" the image with just the neuron
  - We could arrange the outputs in correspondence to the original picture

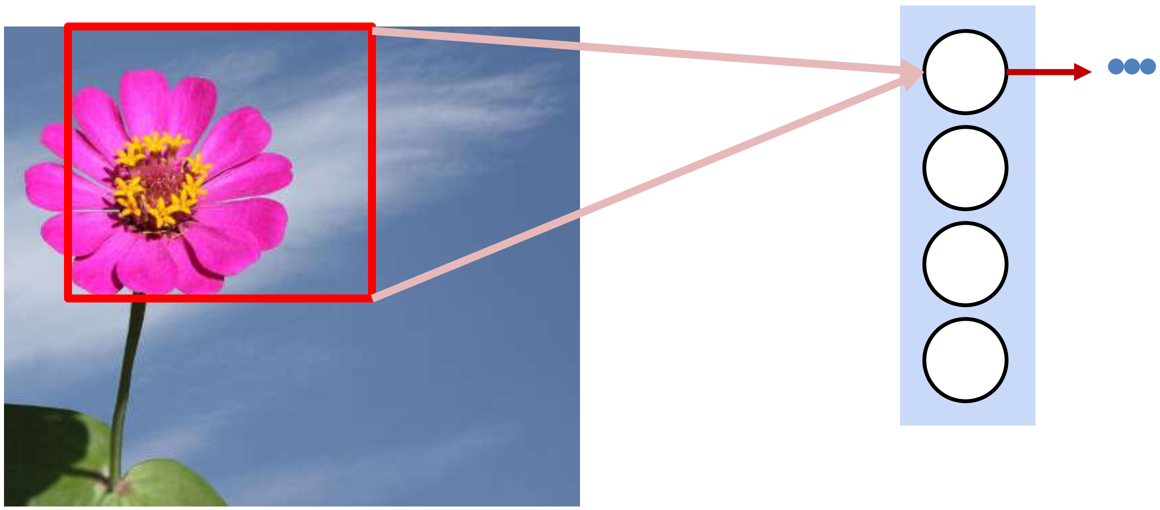
Scanning: A closer look



- Consider a single neuron in the first layer
  - At each position of the box, the neuron is evaluating a "window" of the picture at that location, as part of the classification for *that* region

- Let us compute the output of the first neuron for *all* the windows in the picture before computing the rest of the neurons
  - "Scanning" the image with just the neuron
  - We could arrange the outputs in correspondence to the original picture

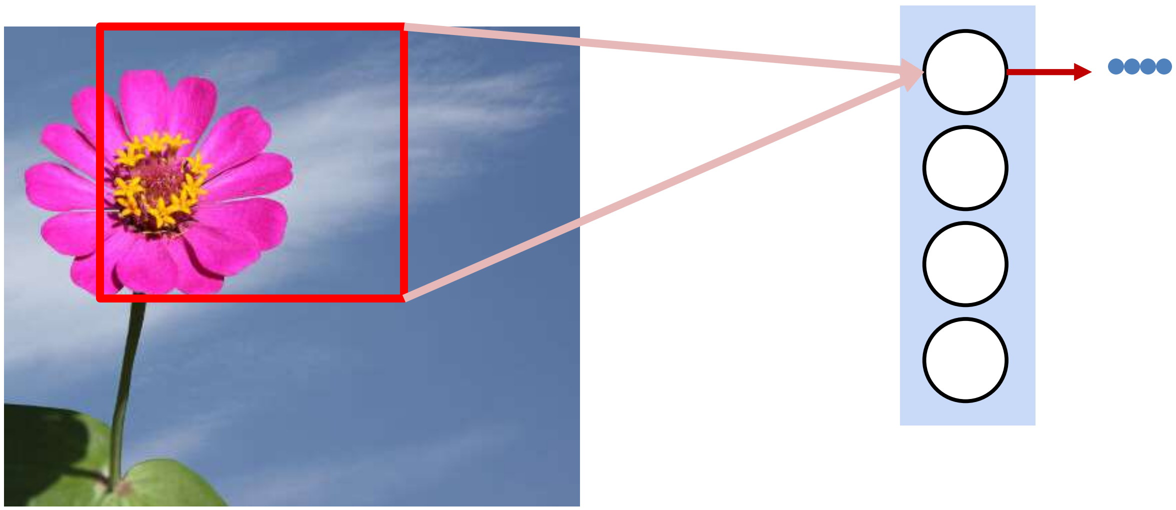
Scanning: A closer look



- Consider a single neuron in the first layer
  - At each position of the box, the neuron is evaluating a “window” of the picture at that location, as part of the classification for *that* region

- Let us compute the output of the first neuron for *all* the windows in the picture before computing the rest of the neurons
  - “Scanning” the image with just the neuron
  - We could arrange the outputs in correspondence to the original picture

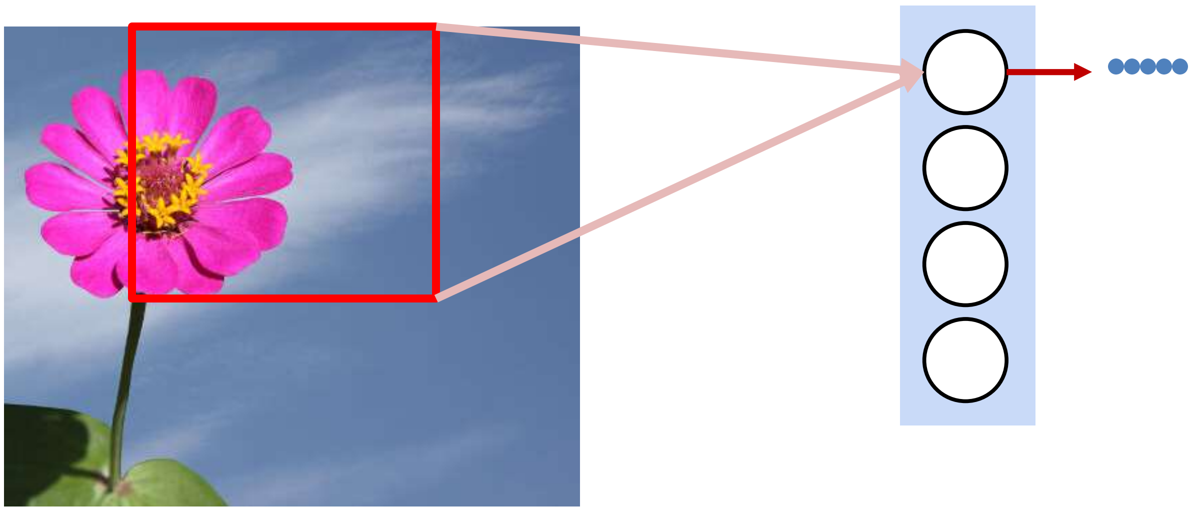
Scanning: A closer look



- Consider a single neuron in the first layer
  - At each position of the box, the neuron is evaluating a "window" of the picture at that location, as part of the classification for *that* region

- Let us compute the output of the first neuron for *all* the windows in the picture before computing the rest of the neurons
  - "Scanning" the image with just the neuron
  - We could arrange the outputs in correspondence to the original picture

Scanning: A closer look



- Consider a single neuron in the first layer
  - At each position of the box, the neuron is evaluating a “window” of the picture at that location, as part of the classification for *that* region
- Let us compute the output of the first neuron for *all* the windows in the picture before computing the rest of the neurons
  - “Scanning” the image with just the neuron
  - We could arrange the outputs in correspondence to the original picture

Scanning: A closer look

- Consider a single neuron in the first layer
  – At each position of the box, the neuron is evaluating a “window” of the picture at that location, as part of the classification for *that* region

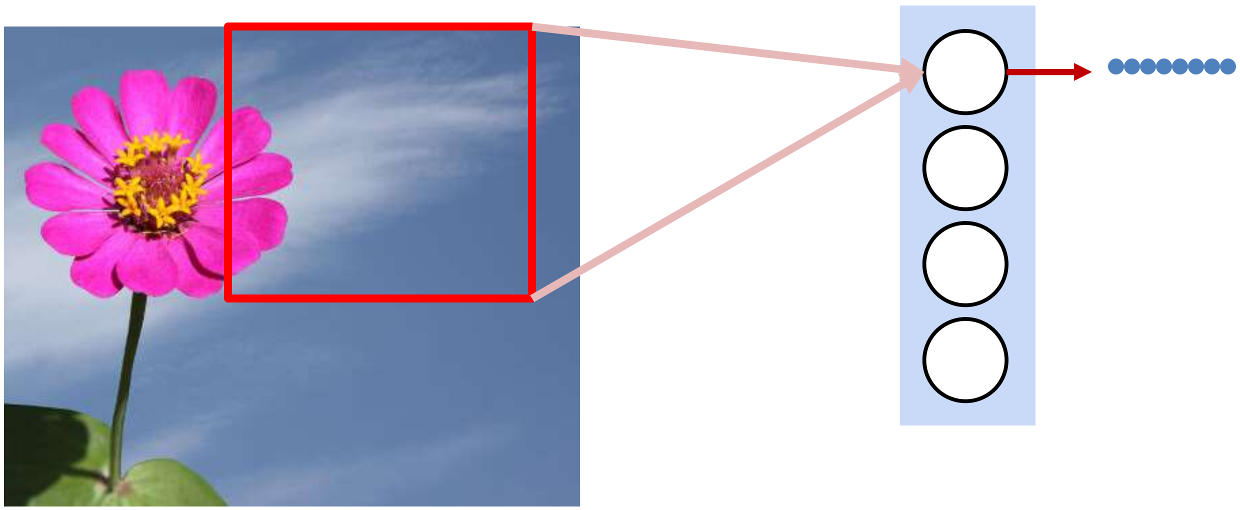
- Let us compute the output of the first neuron for *all* the windows in the picture before computing the rest of the neurons
  – “Scanning” the image with just the neuron
  – We could arrange the outputs in correspondence to the original picture

Scanning: A closer look

- Consider a single neuron in the first layer
  - At each position of the box, the neuron is evaluating a “window” of the picture at that location, as part of the classification for *that* region

- Let us compute the output of the first neuron for *all* the windows in the picture before computing the rest of the neurons
  - “Scanning” the image with just the neuron
  - We could arrange the outputs in correspondence to the original picture

Scanning: A closer look



- Consider a single neuron in the first layer
  - At each position of the box, the neuron is evaluating a "window" of the picture at that location, as part of the classification for *that* region

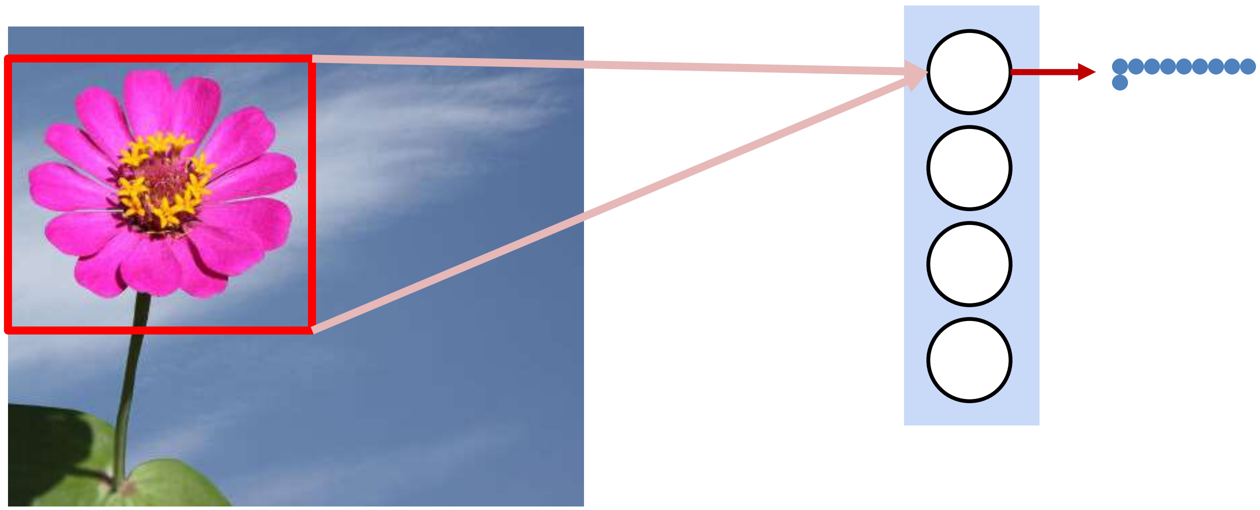
- Let us compute the output of the first neuron for *all* the windows in the picture before computing the rest of the neurons
  - "Scanning" the image with just the neuron
  - We could arrange the outputs in correspondence to the original picture

Scanning: A closer look

- Consider a single neuron in the first layer
  - At each position of the box, the neuron is evaluating a “window” of the picture at that location, as part of the classification for *that* region

- Let us compute the output of the first neuron for *all* the windows in the picture before computing the rest of the neurons
  - “Scanning” the image with just the neuron
  - We could arrange the outputs in correspondence to the original picture

Scanning: A closer look



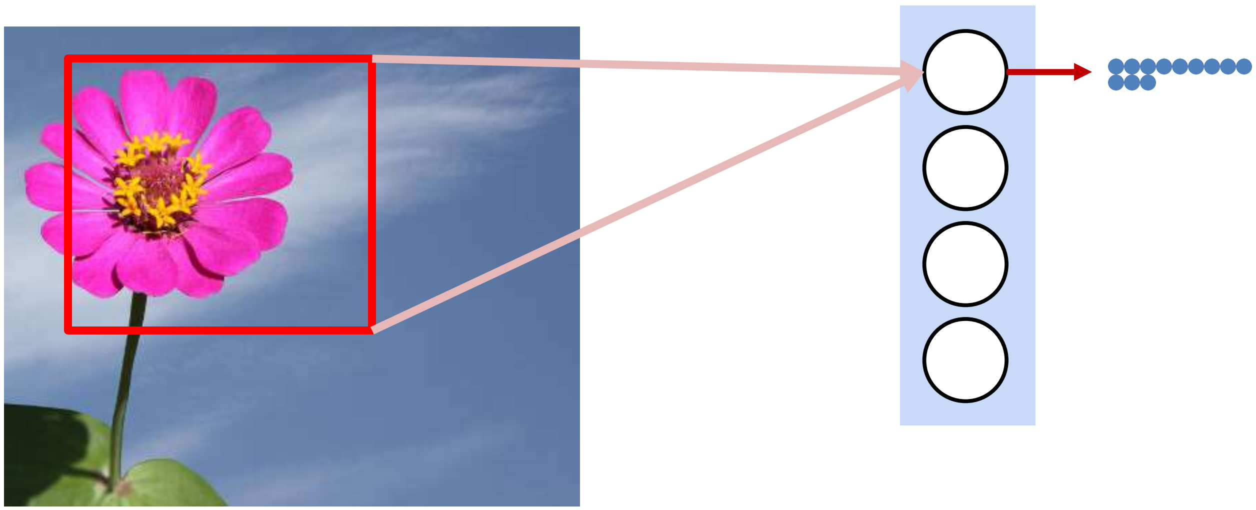
- Consider a single neuron in the first layer
  - At each position of the box, the neuron is evaluating a "window" of the picture at that location, as part of the classification for *that* region

- Let us compute the output of the first neuron for *all* the windows in the picture before computing the rest of the neurons
  - "Scanning" the image with just the neuron
  - We could arrange the outputs in correspondence to the original picture

Scanning: A closer look

- Consider a single neuron in the first layer
  - At each position of the box, the neuron is evaluating a “window” of the picture at that location, as part of the classification for *that* region
- Let us compute the output of the first neuron for *all* the windows in the picture before computing the rest of the neurons
  - “Scanning” the image with just the neuron
  - We could arrange the outputs in correspondence to the original picture

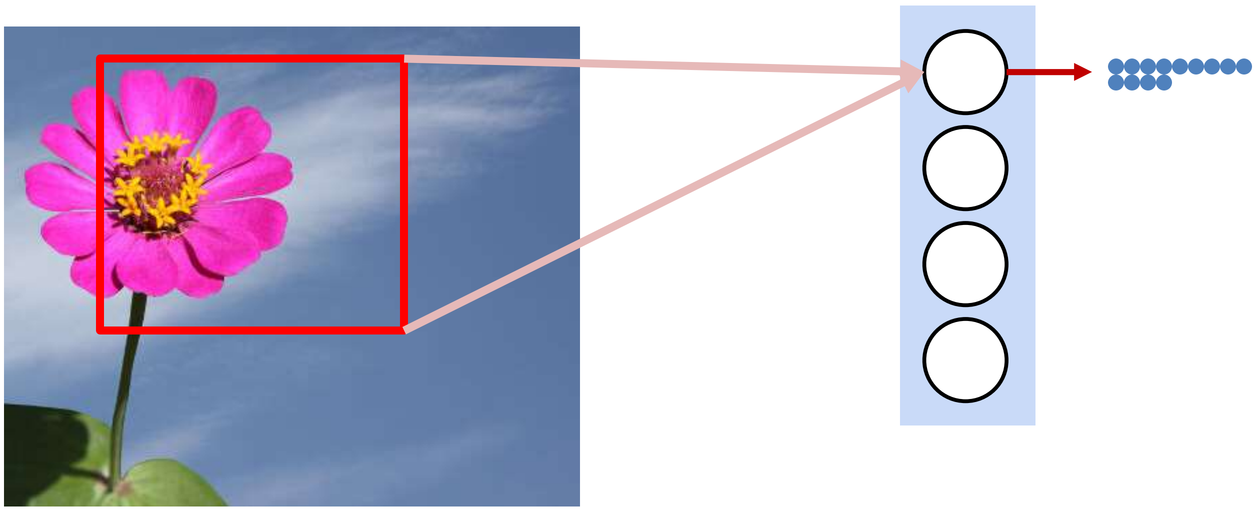
Scanning: A closer look



- Consider a single neuron in the first layer
  - At each position of the box, the neuron is evaluating a “window” of the picture at that location, as part of the classification for *that* region

- Let us compute the output of the first neuron for *all* the windows in the picture before computing the rest of the neurons
  - “Scanning” the image with just the neuron
  - We could arrange the outputs in correspondence to the original picture

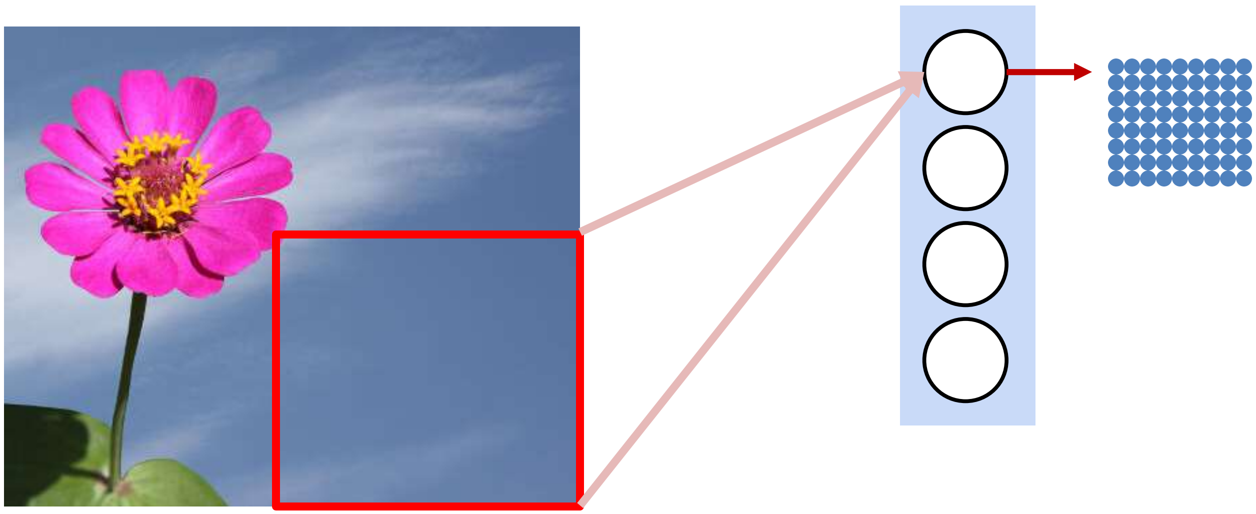
Scanning: A closer look



- Consider a single neuron in the first layer
  - At each position of the box, the neuron is evaluating a "window" of the picture at that location, as part of the classification for *that* region

- Let us compute the output of the first neuron for *all* the windows in the picture before computing the rest of the neurons
  - "Scanning" the image with just the neuron
  - We could arrange the outputs in correspondence to the original picture

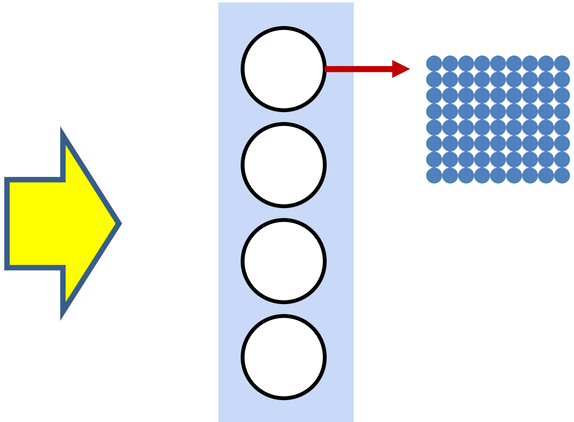
Scanning: A closer look



- Consider a single neuron in the first layer
  - At each position of the box, the neuron is evaluating a "window" of the picture at that location, as part of the classification for *that* region

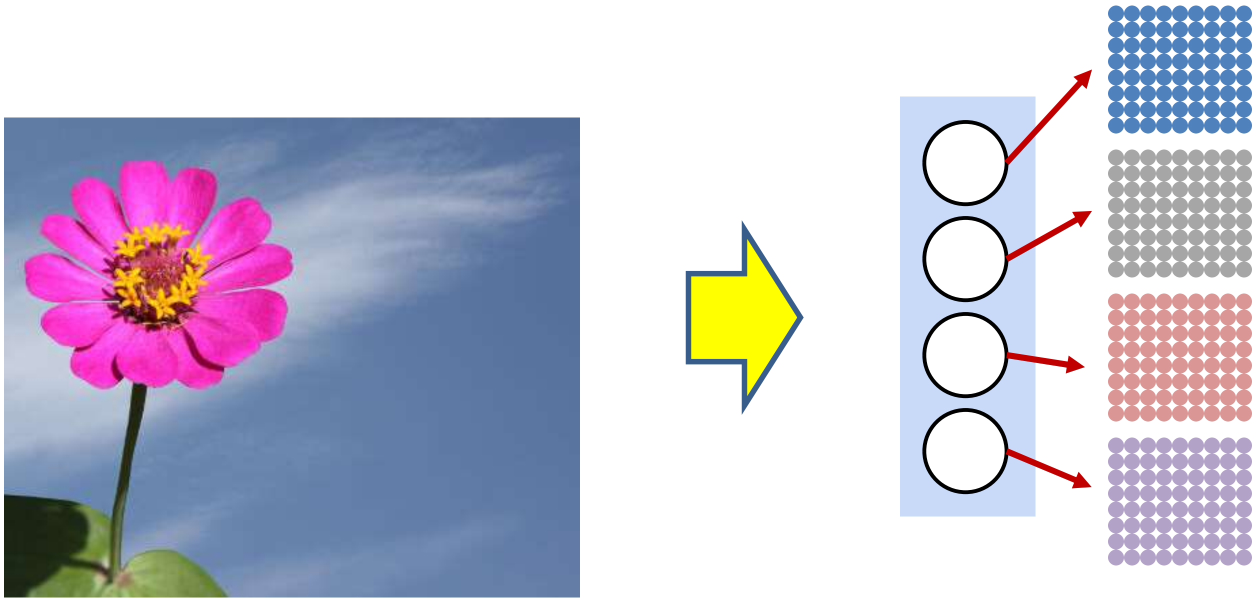
- Let us compute the output of the first neuron for *all* the windows in the picture before computing the rest of the neurons
  - "Scanning" the image with just the neuron
  - We could arrange the outputs in correspondence to the original picture

Scanning: A closer look



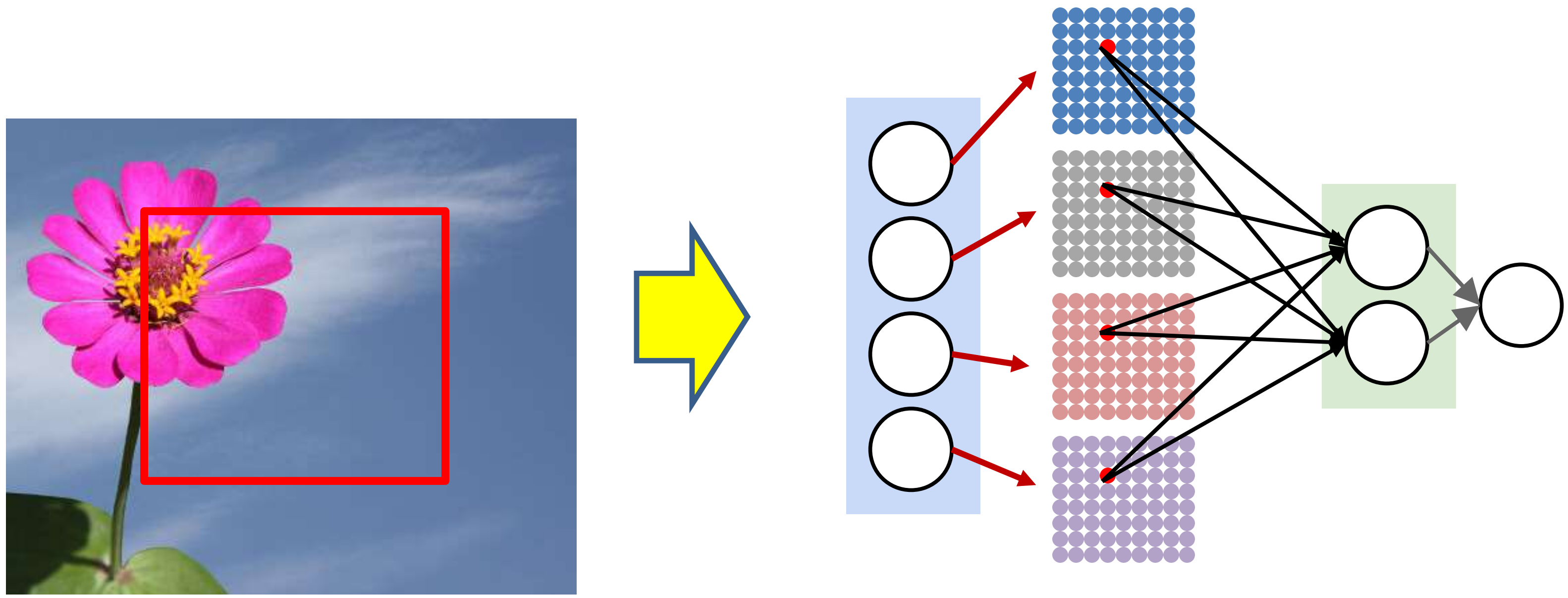
- Let us compute the output of the first neuron for *all* the windows in the picture before computing the rest of the neurons
- Eventually, we can arrange the outputs from the response at the scanned positions into a rectangle that's proportional in size to the original picture

Scanning: A closer look



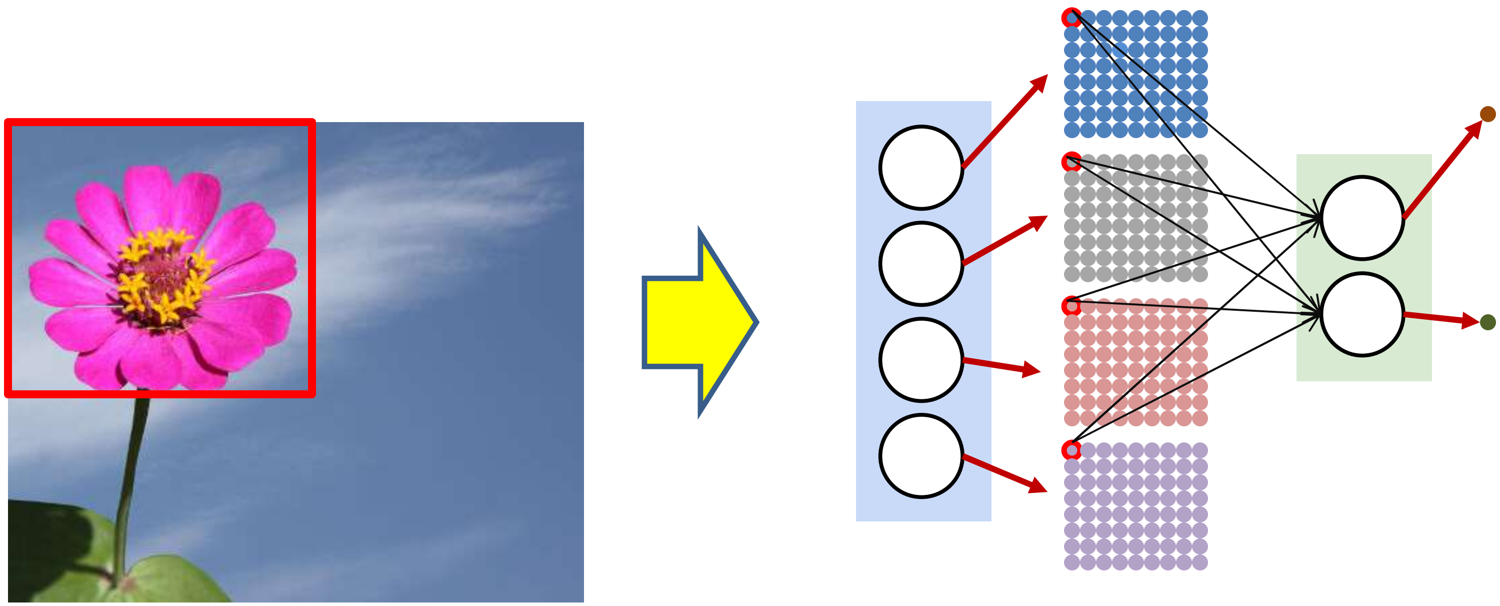
- We can repeat the process for each of the first-layer neurons
  - "Scan" the input with the neuron
  - Arrange the neuron's outputs from the scanned positions according to their positions in the original image

Scanning: A closer look



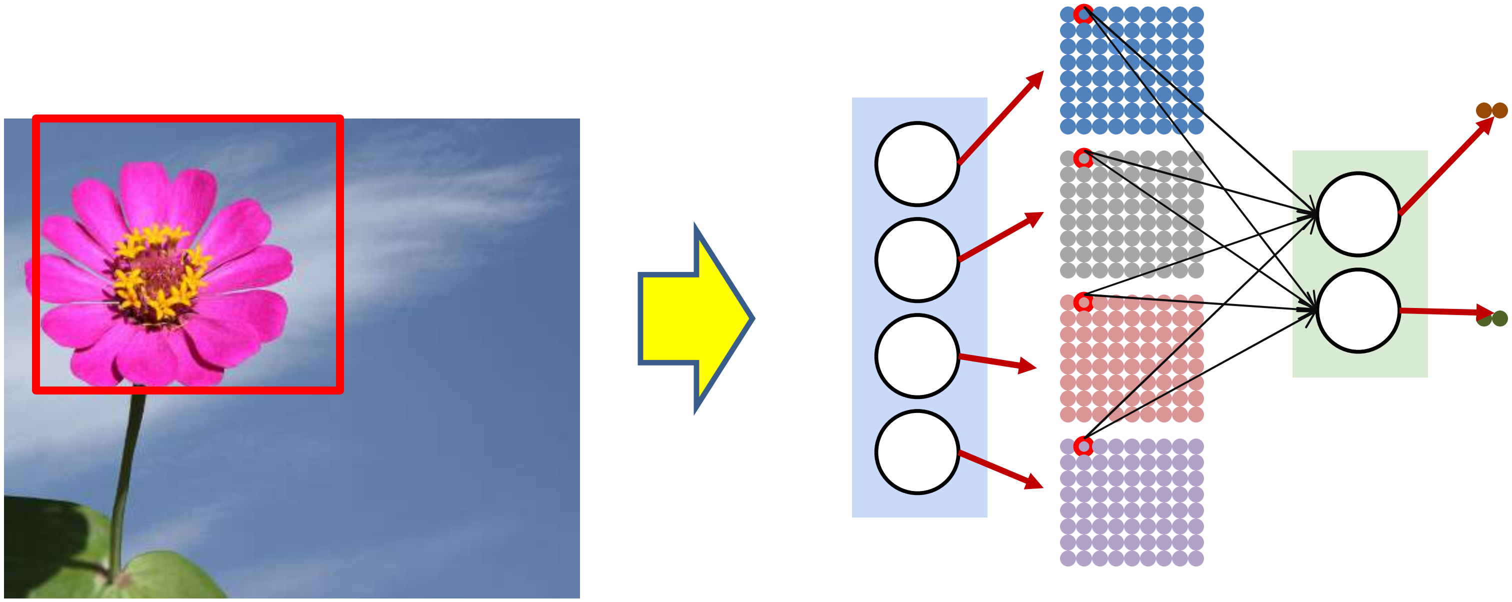
- To classify a specific “window” in the image, we send the first level activations from the positions corresponding to that position to the next layer

Scanning: A closer look



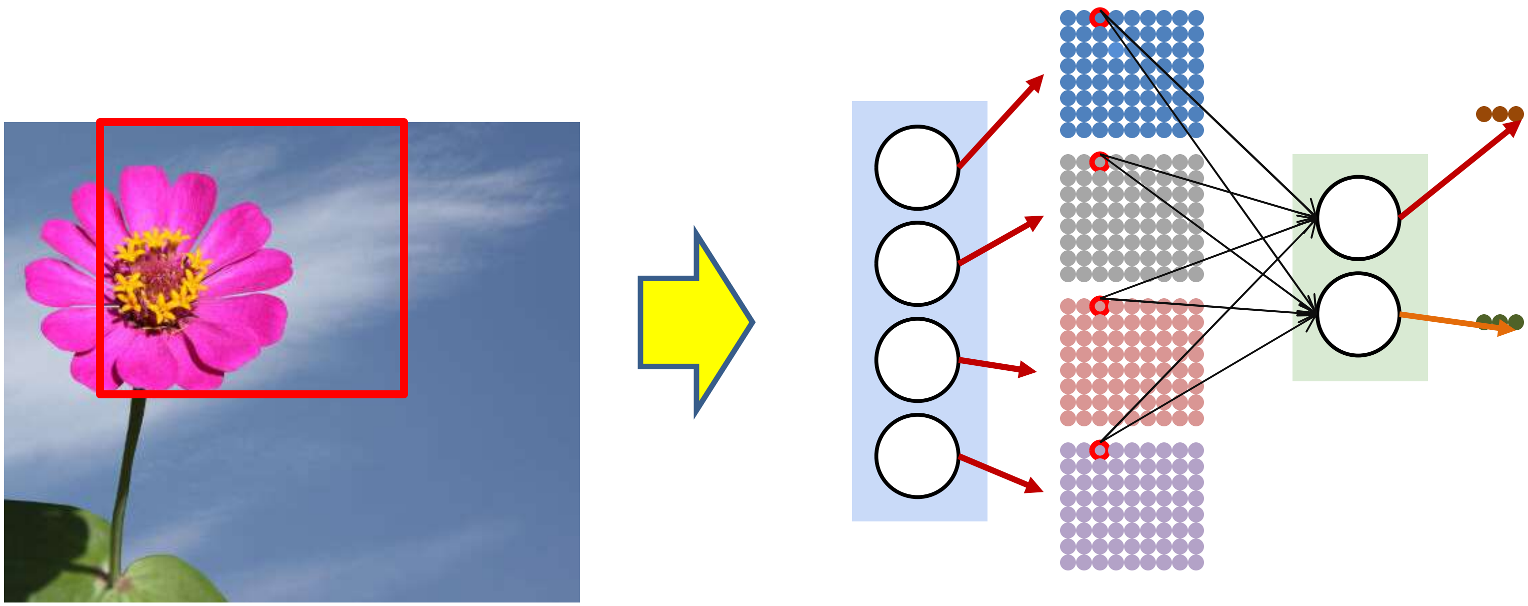
- We can recurse the logic
  - The second level neurons too can “**scan**” the rectangular outputs of the first-level neurons before computing subsequent layers
  - (Un)like the first level, they must jointly scan *multiple* “maps”
    - Each location in the output of the second level neuron considers the corresponding locations from the output maps of all the first-level neurons

Scanning: A closer look



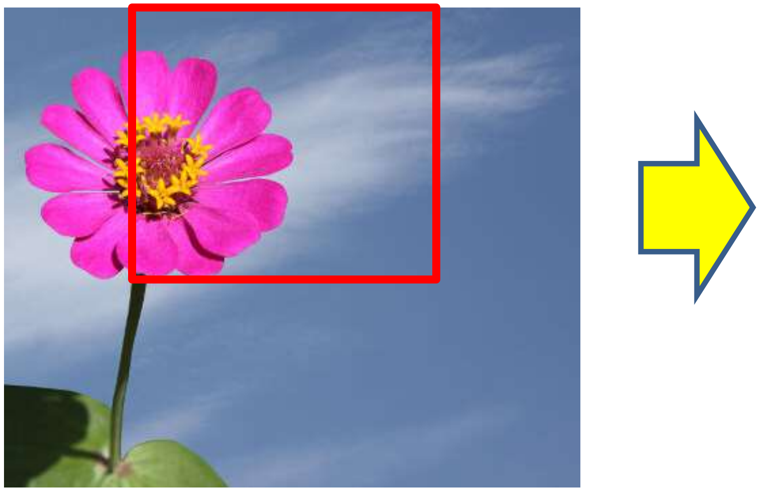
- We can recurse the logic
  - The second level neurons too can "**scan**" the rectangular outputs of the first-level neurons before computing subsequent layers
  - (Un)like the first level, they must jointly scan *multiple* "maps"
    - Each location in the output of the second level neuron considers the corresponding locations from the output maps of all the first-level neurons

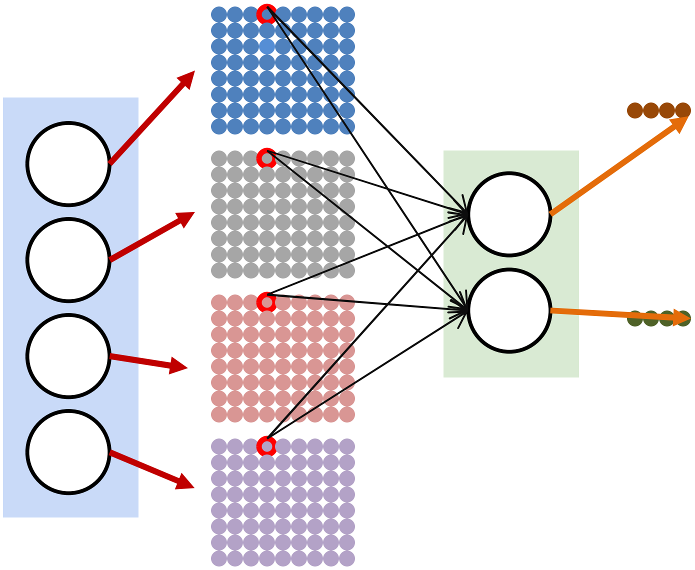
Scanning: A closer look



- We can recurse the logic
  - The second level neurons too can "**scan**" the rectangular outputs of the first-level neurons before computing subsequent layers
  - (Un)like the first level, they must jointly scan *multiple* "maps"
    - Each location in the output of the second level neuron considers the corresponding locations from the output maps of all the first-level neurons

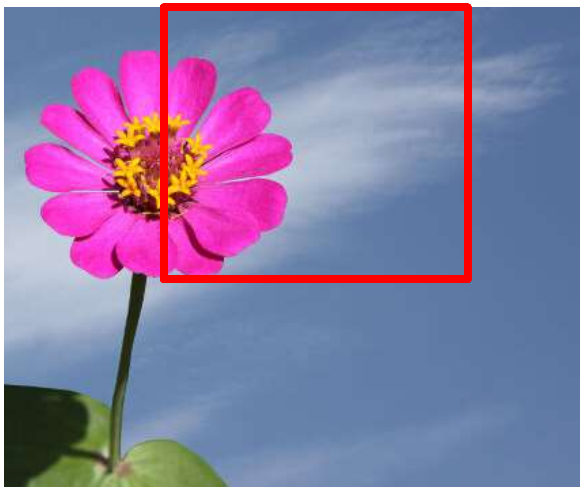
Scanning: A closer look

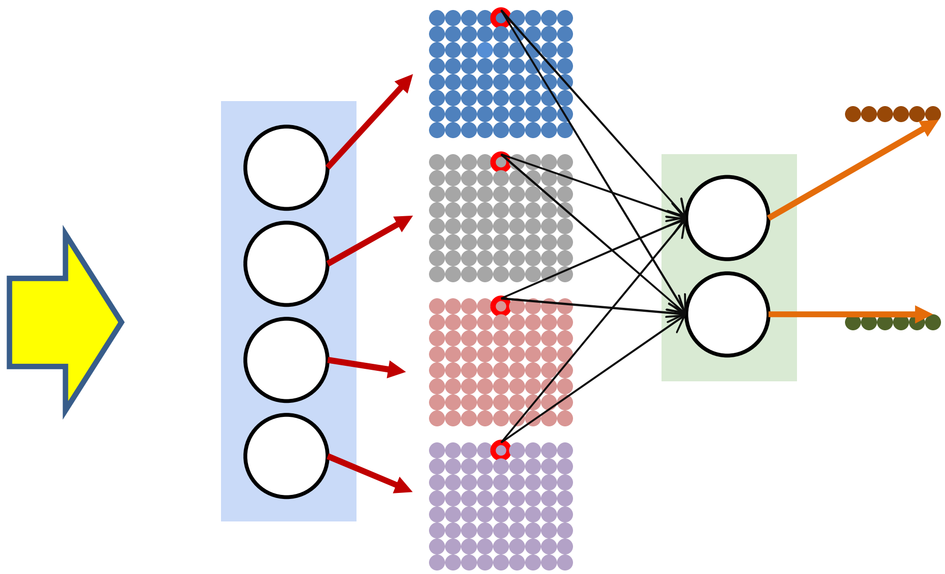




- We can recurse the logic
  - The second level neurons too can "**scan**" the rectangular outputs of the first-level neurons before computing subsequent layers
  - (Un)like the first level, they must jointly scan *multiple* "maps"
    - Each location in the output of the second level neuron considers the corresponding locations from the output maps of all the first-level neurons

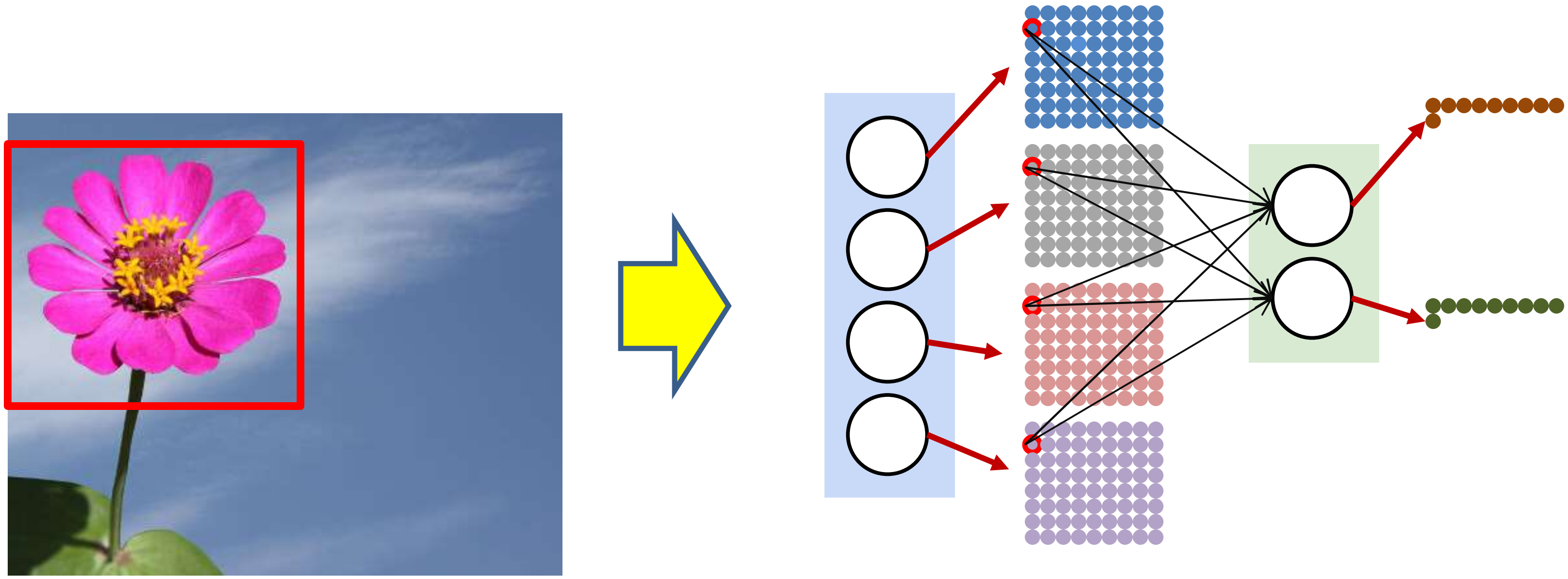
Scanning: A closer look





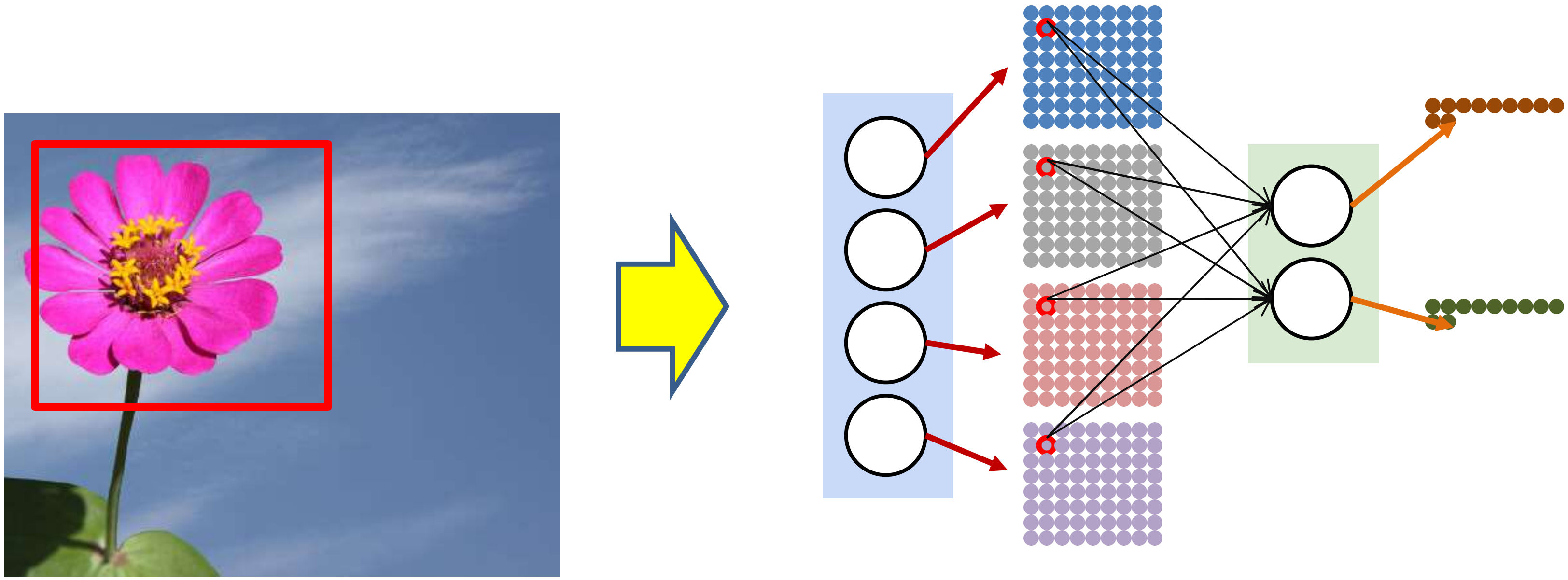
- We can recurse the logic
  - The second level neurons too can "**scan**" the rectangular outputs of the first-level neurons before computing subsequent layers
  - (Un)like the first level, they must jointly scan *multiple* "maps"
    - Each location in the output of the second level neuron considers the corresponding locations from the output maps of all the first-level neurons

Scanning: A closer look



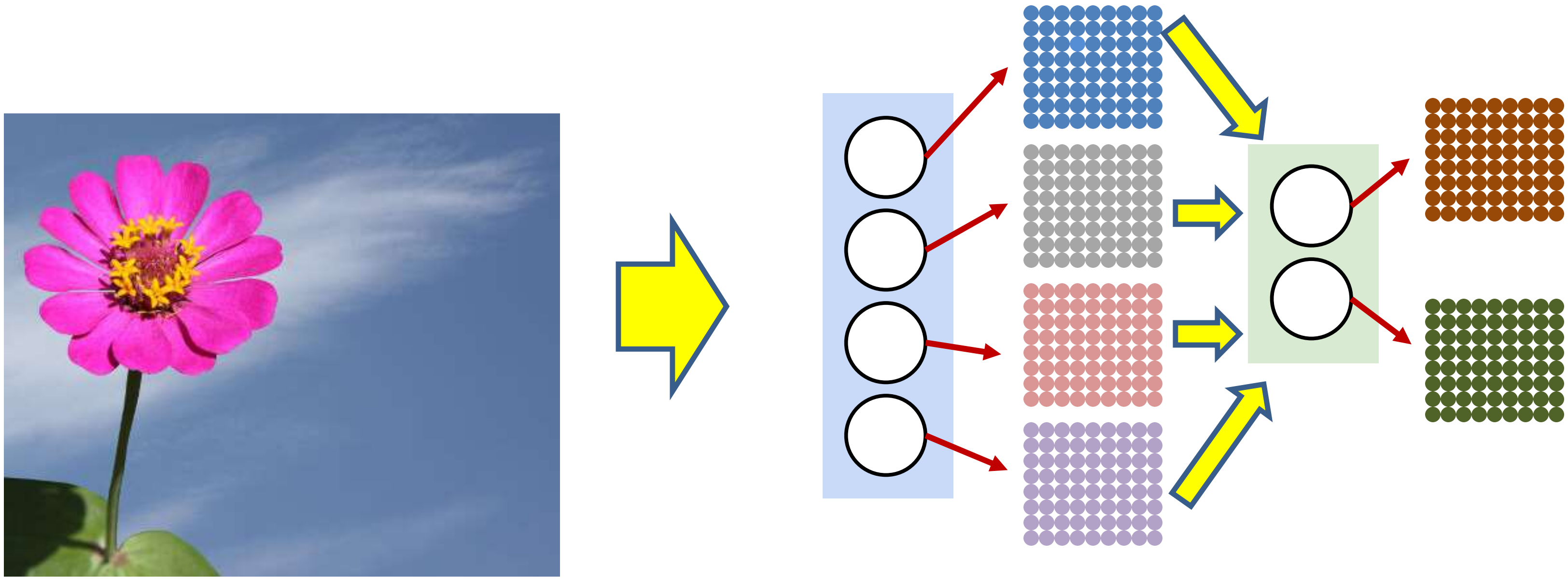
- We can recurse the logic
  - The second level neurons too can "**scan**" the rectangular outputs of the first-level neurons before computing subsequent layers
  - (Un)like the first level, they must jointly scan *multiple* "maps"
    - Each location in the output of the second level neuron considers the corresponding locations from the output maps of all the first-level neurons

Scanning: A closer look



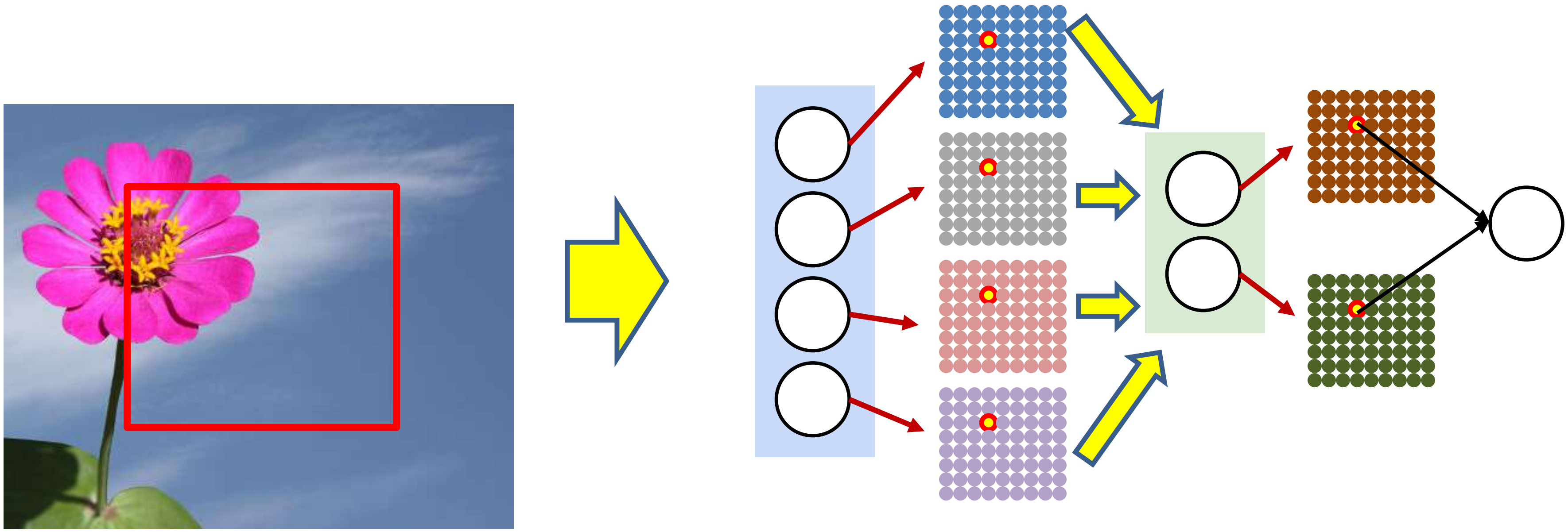
- We can recurse the logic
  - The second level neurons too can "**scan**" the rectangular outputs of the first-level neurons before computing subsequent layers
  - (Un)like the first level, they must jointly scan *multiple* "maps"
    - Each location in the output of the second level neuron considers the corresponding locations from the output maps of all the first-level neurons

Scanning: A closer look



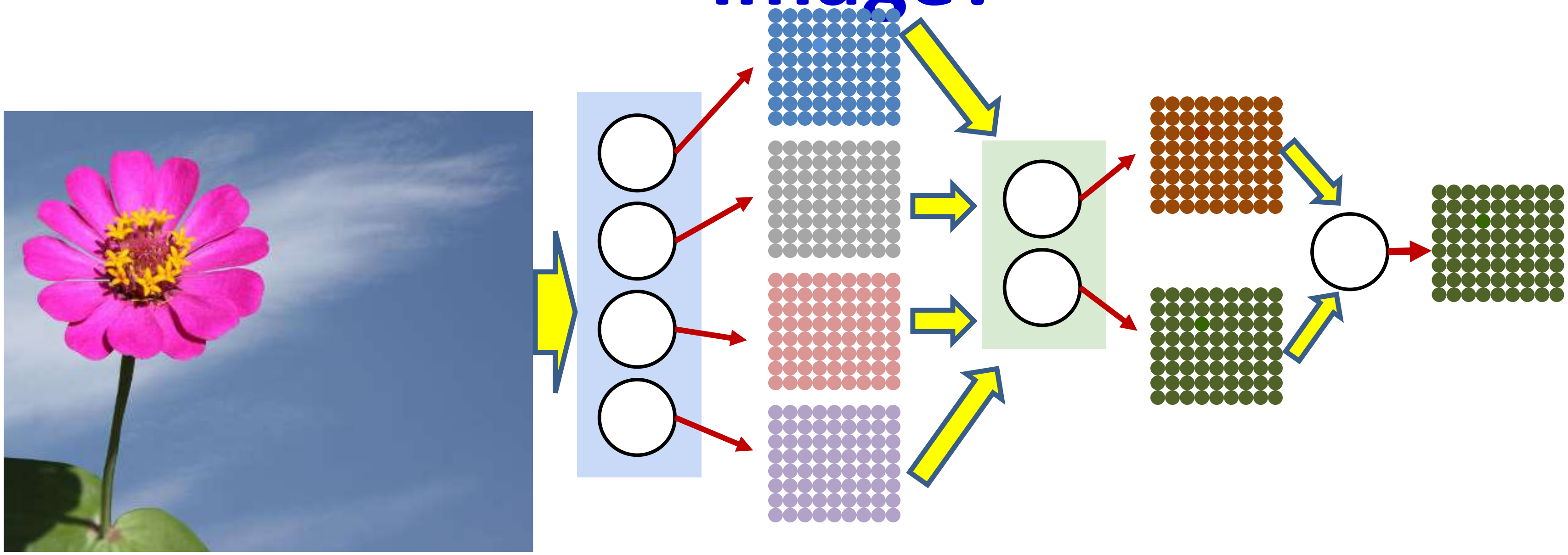
- We can recurse the logic
  - The second level neurons too can “**scan**” the rectangular outputs of the first-level neurons before computing subsequent layers
  - (Un)like the first level, they must jointly scan *multiple* “maps”
    - Each location in the output of the second level neuron considers the corresponding locations from the output maps of all the first-level neurons

Scanning: A closer look



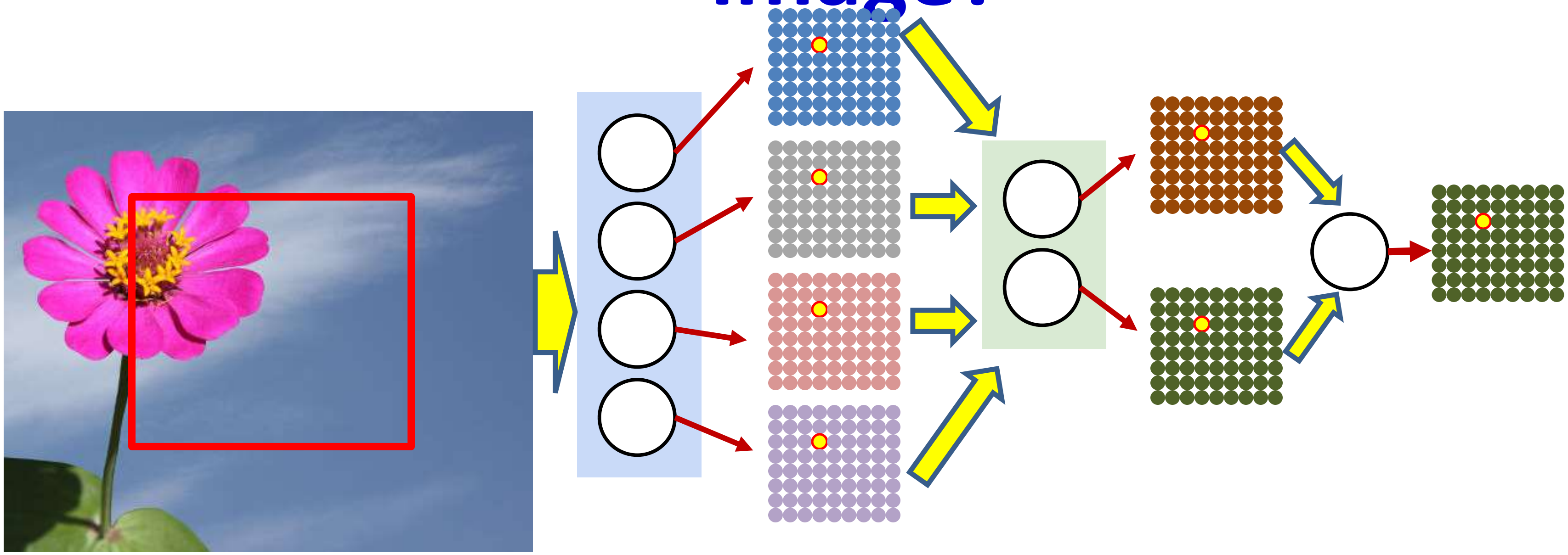
- To detect a picture ***at any location*** in the original image, the output layer must consider the corresponding outputs of the last hidden layer

Detecting a picture anywhere in the image?



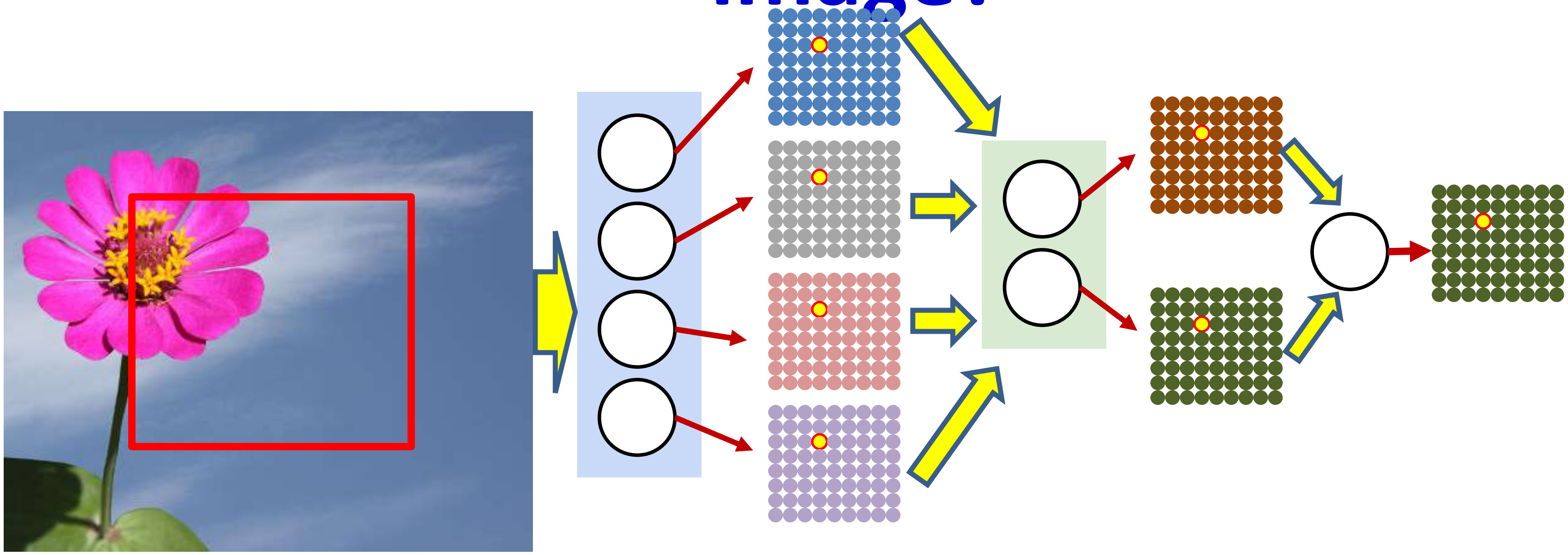
- Recursing the logic, we can create a map for the neurons in the next layer as well
  - The map is a flower detector for each location of the original image

Detecting a picture anywhere in the image?



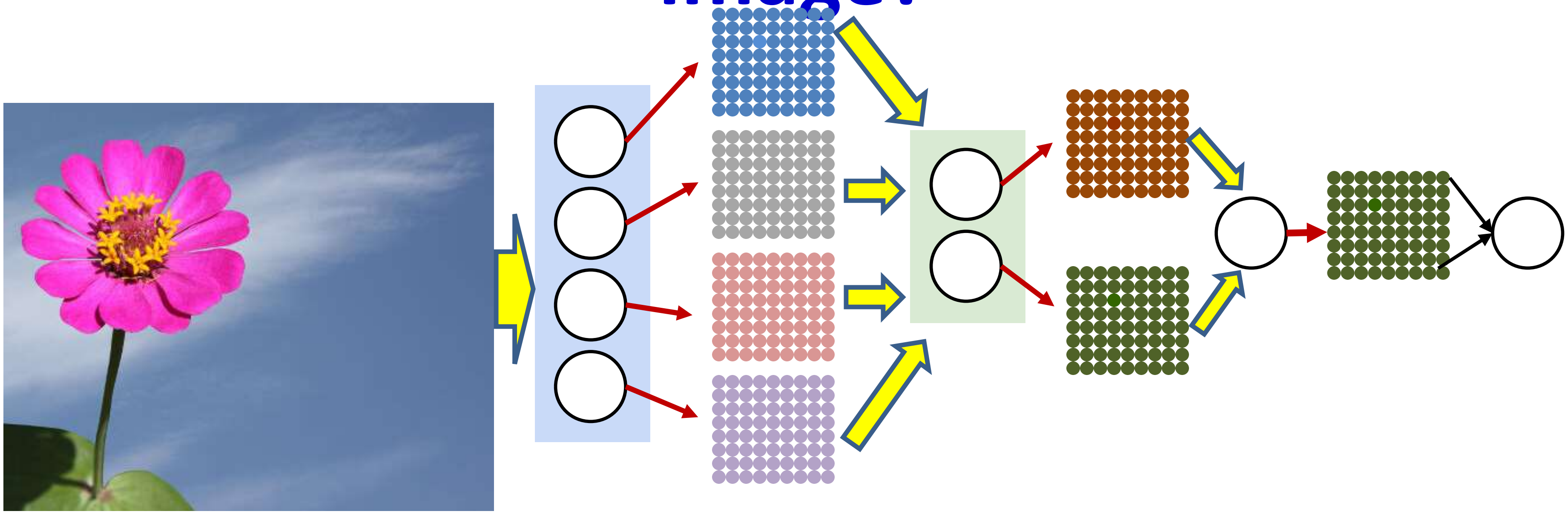
- To detect a picture ***at any location*** in the original image, only need to consider the corresponding location of the output map

Detecting a picture anywhere in the image?



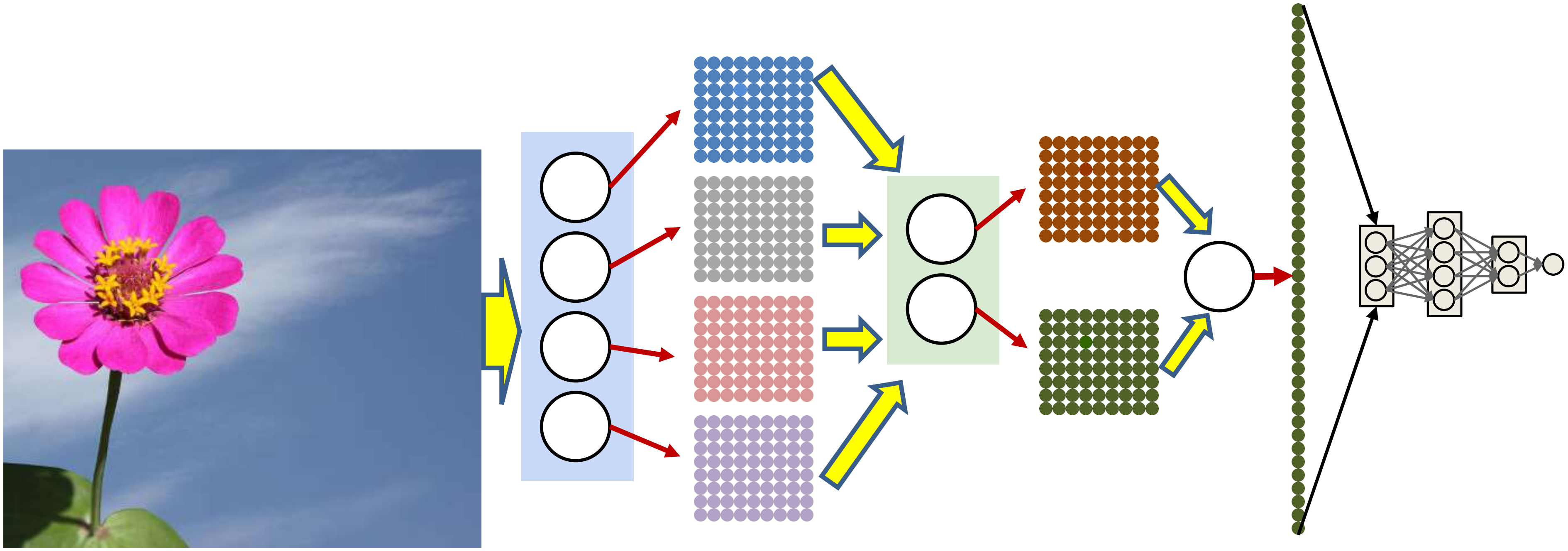
- To detect a picture ***at any location*** in the original image, only need to consider the corresponding location of the output map
- Actual problem? Is there a flower in the image
  - Not "detect the location of a flower"

Detecting a picture anywhere in the image?



- Is there a flower in the picture?
- The entire output map can be sent into a final "max" to detect a flower in the full picture
  - 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

Scanning with an MLP

- `KxK` = size of "patch" evaluated by MLP
- `W` is width of image
- `H` is height of image

```
for x = 1:W-K+1
  for y = 1:H-K+1
    ImgSegment = Img(*, x:x+K-1, y:y+K-1)
    Y(x,y) = MLP(ImgSegment)

Y = softmax( Y(1,1)..Y(W-K+1,H-K+1) )
```

Scanning with an MLP

```
for x = 1:W-K+1
  for y = 1:H-K+1
     for l = 1:L  # layers operate on vector at (x,y)
        for j = 1:D_l
           if (l == 1) #first layer operates on input
              Y(0,:,x,y) = Img(1:C, x:x+K-1, y:y+K-1)
           end
           z(l,j,x,y) = b(l,j)
           for i = 1:D_l-1
               z(l,j,x,y) += w(l,i,j)Y(l-1,i,x,y)
           Y(l,j,x,y) = activation(z(l,j,x,y))

Y = softmax( Y(L,:,1,1)..Y(L,:,W-K+1,H-K+1) )
```

Scanning with an MLP

```
for x = 1:W-K+1
  for y = 1:H-K+1
    for l = 1:L  # layers operate on vector at (x,y)
      for j = 1:Dl
        if (l == 1) #first layer operates on input
          Y(0,:,x,y) = Img(1:C, x:x+K-1, y:y+K-1)
        end
        z(l,j,x,y) = b(l,j)
        for i = 1:Dl-1
          z(l,j,x,y) += w(l,i,j)Y(l-1,i,x,y)
        Y(l,j,x,y) = activation(z(l,j,x,y))

Y = softmax( Y(L,:,1,1)..Y(L,:,W-K+1,H-K+1) )
```

Scanning with an MLP

```
for l = 1:L
  for x = 1:W-K+1
    for y = 1:H-K+1
      for j = 1:D_l
        if (l == 1) #first layer operates on input
           Y(0,:,x,y) = Img(1:C, x:x+K-1, y:y+K-1)
        end
        z(l,j,x,y) = b(l,j)
        for i = 1:D_l-1
           z(l,j,x,y) += w(l,i,j)Y(l-1,i,x,y)
        Y(l,j,x,y) = activation(z(l,j,x,y))

Y = softmax( Y(L,:,1,1)..Y(L,:,W-K+1,H-K+1) )
```

Scanning with an MLP

```
for l = 1:L
  for x = 1:W-K+1
    for y = 1:H-K+1
      for j = 1:D_l
        if (l == 1) #first layer operates on input
          Y(0,:,x,y) = Img(1:C, x:x+K-1, y:y+K-1)
        end
        z(l,j,x,y) = b(l,j)
        for i = 1:D_l-1
          z(l,j,x,y) += w(l,i,j)Y(l-1,i,x,y)
        Y(l,j,x,y) = activation(z(l,j,x,y))

Y = softmax( Y(L,:,1,1)..Y(L,:,W-K+1,H-K+1) )
```

Scanning with an MLP

```
for x = 1:W-K+1
  for y = 1:H-K+1
     for l = 1:L  # layers operate on vector at (x,y)
        if (l == 1) #first layer operates on input
            Y(0,x,y) = Img(1:C, x:x+K-1, y:y+K-1)
        end
        z(l,x,y) = W(l)Y(l-1,x,y) + b(l)
        Y(l,x,y) = activation(z(l,x,y))

Y = softmax( Y(L,1,1)..Y(L,W-K+1,H-K+1) )
```

Scanning with an MLP

```
for x = 1:W-K+1
  for y = 1:H-K+1
    for l = 1:L  # layers operate on vector at (x,y)
      if (l == 1) #first layer operates on input
          Y(0,x,y) = Img(1:C, x:x+K-1, y:y+K-1)
      end
      z(l,x,y) = W(l)Y(l-1,x,y) + b(l)
      Y(l,x,y) = activation(z(l,x,y))

Y = softmax( Y(L,1,1)..Y(L,W-K+1,H-K+1) )
```

Scanning with an MLP

```
for l = 1:L         # layers
  for x = 1:W-K+1
     for y = 1:H-K+1
        if (l == 1) #first layer operates on input
            Y(0,x,y) = Img(1:C, x:x+K-1, y:y+K-1)
        end
        z(l,x,y) = W(l)Y(l-1,x,y) + b(l)
        Y(l,x,y) = activation(z(l,x,y))

Y = softmax( Y(L,1,1)..Y(L,W-K+1,H-K+1) )
```

Reordering the computation: Vector notation

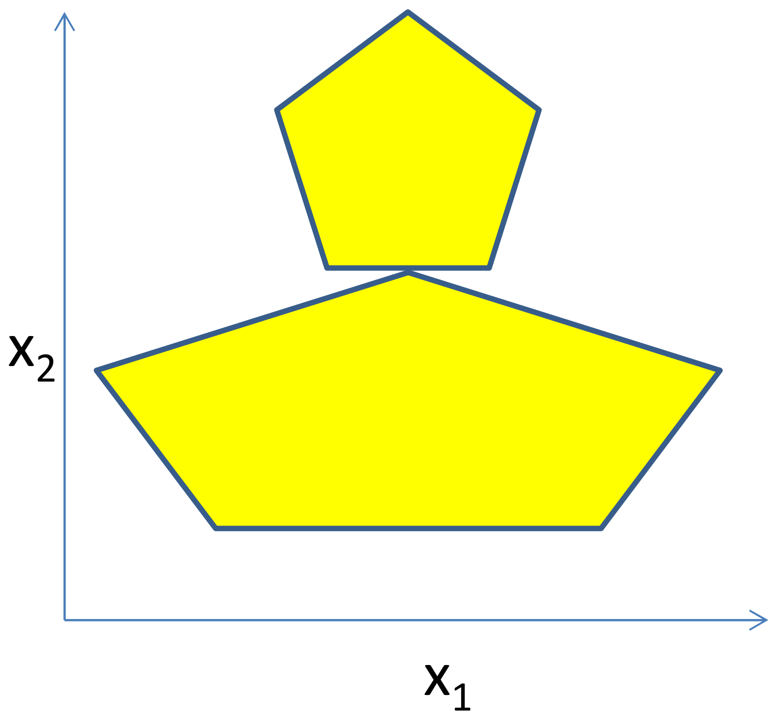
```
for l = 1:L  # layers operate on vector at (x,y)
   for x = 1:W-K+1
      for y = 1:H-K+1
         if (l == 1) #first layer operates on input
             Y(0,x,y) = Img(1:C, x:x+K-1, y:y+K-1)
         end
         z(l,x,y) = W(l)Y(l-1,x,y) + b(l)
         Y(l,x,y) = activation(z(l,x,y))

Y = softmax( Y(L,1,1)..Y(L,W-K+1,H-K+1) )
```

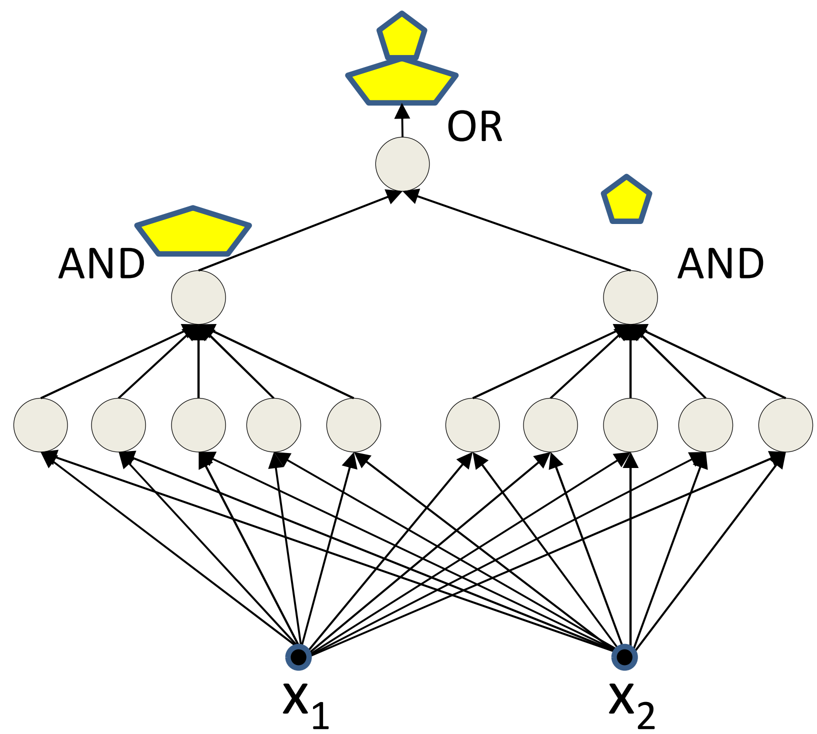
Story so far

- Position-invariant pattern classification can be performed by scanning the input for a target pattern
  - Scanning is equivalent to composing a large network with shared subnets

- The operations in scanning the input with a full network can be equivalently reordered as
  - scanning the input with individual neurons in the first layer to produce scanned "maps" of the input
  - Jointly scanning the "map" of outputs by all neurons in the previous layers by neurons in subsequent layers

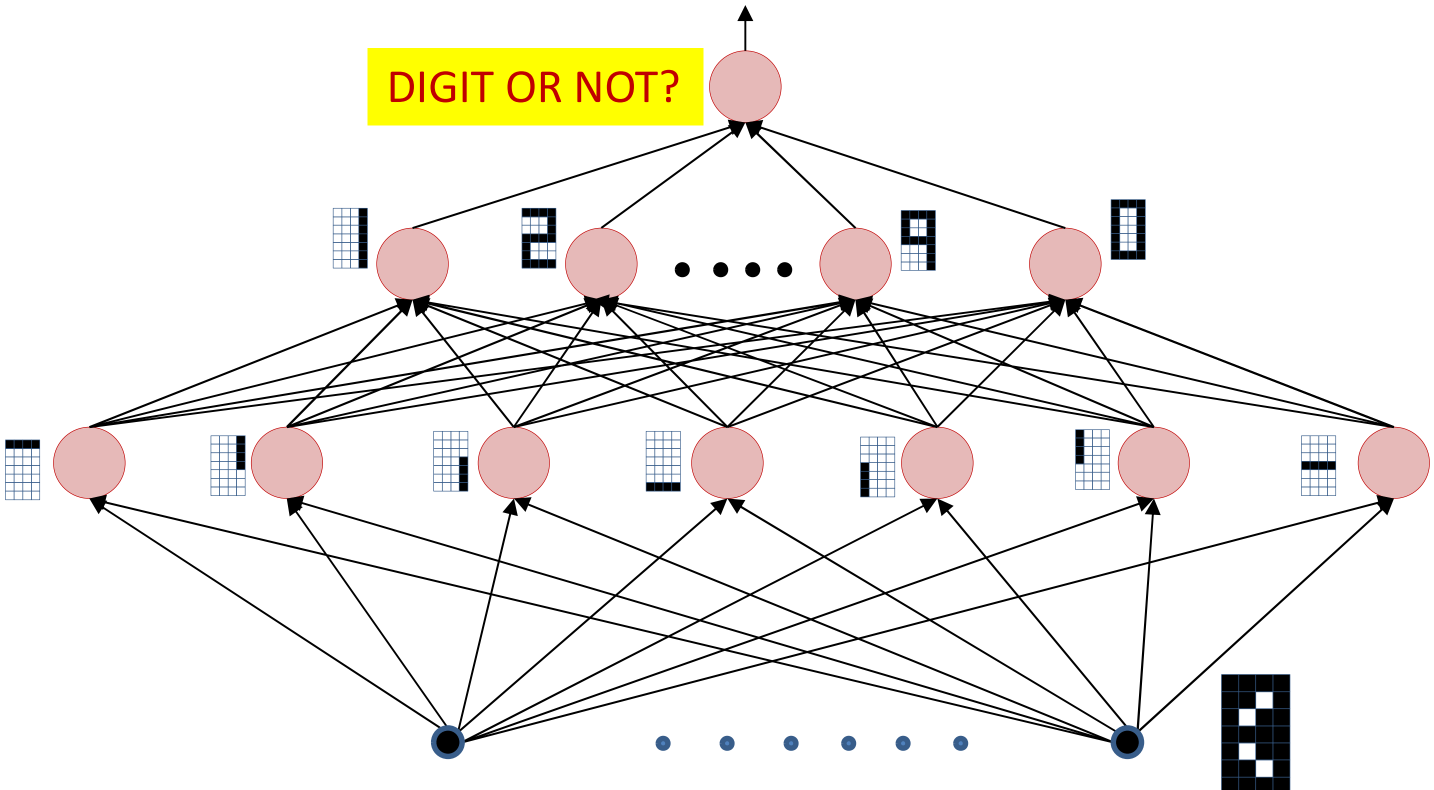
Recall: What does an MLP model?





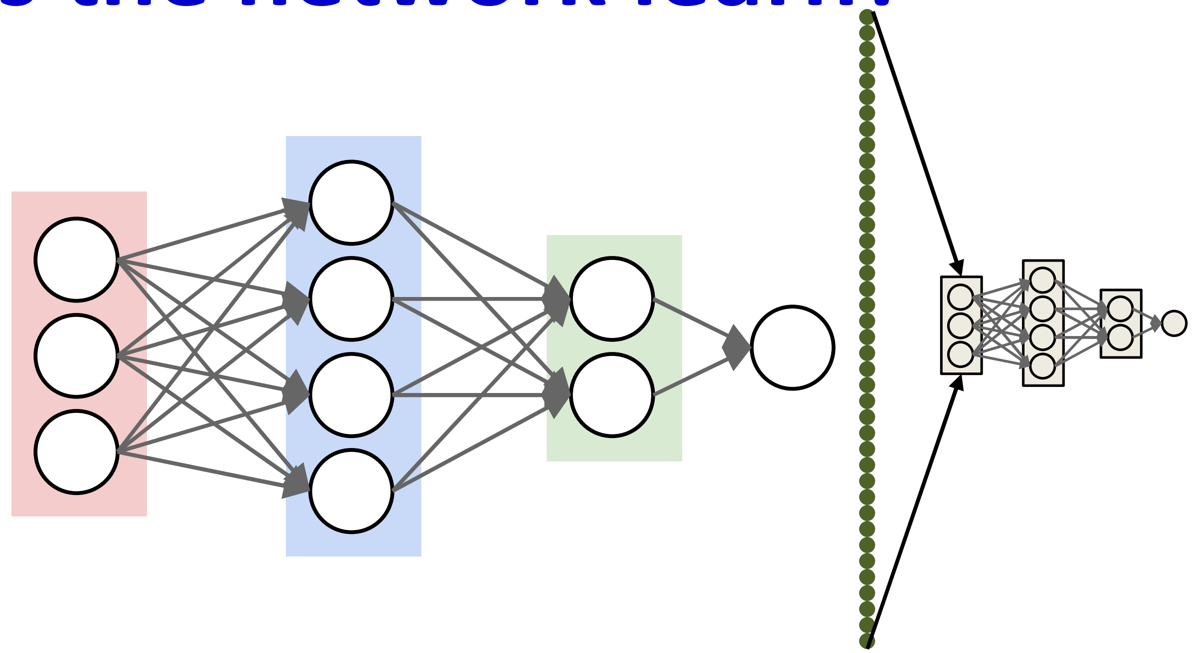
- The lowest layers of the network capture simple patterns
  - The linear decision boundaries in this example
- The next layer captures more complex patterns
  - The polygons
- The next one captures still more complex patterns..

How does an MLP represent patterns



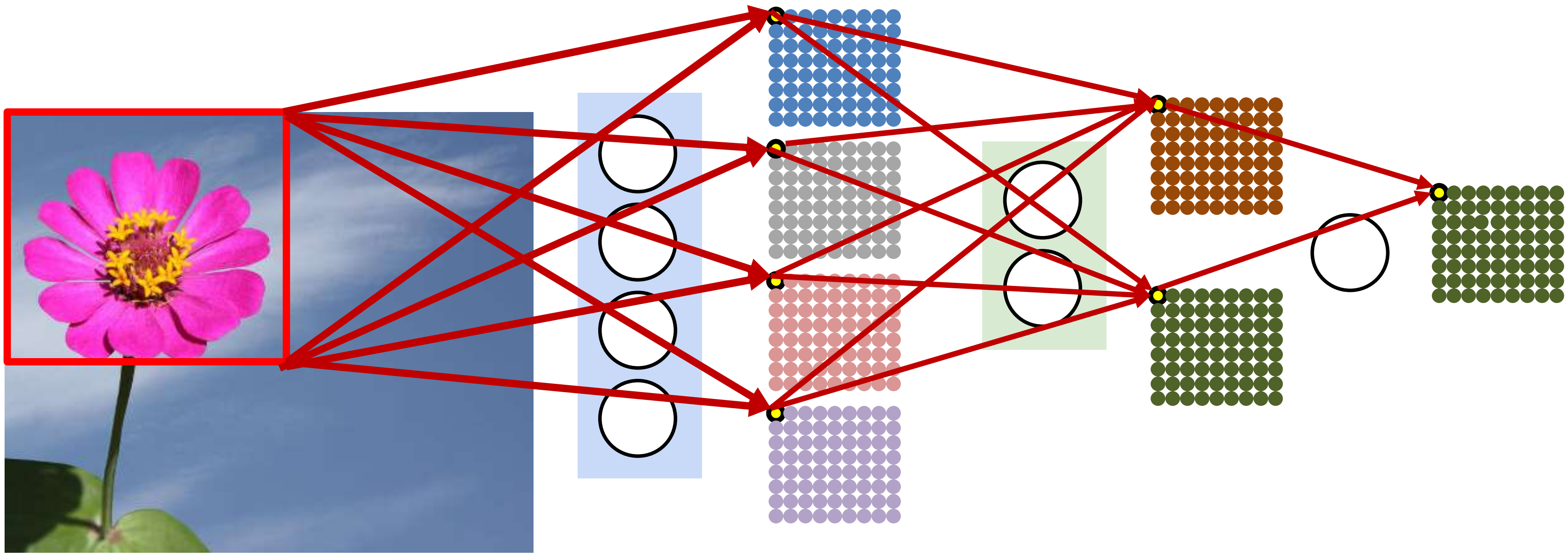
- **The neurons in an MLP *build up* complex patterns from simple pattern hierarchically**
  - Each layer learns to "detect" simple combinations of the patterns detected by earlier layers
- Ideally must encourage such hierarchical learning
  - More data/parameter efficient

Returning to our problem: What does the network learn?



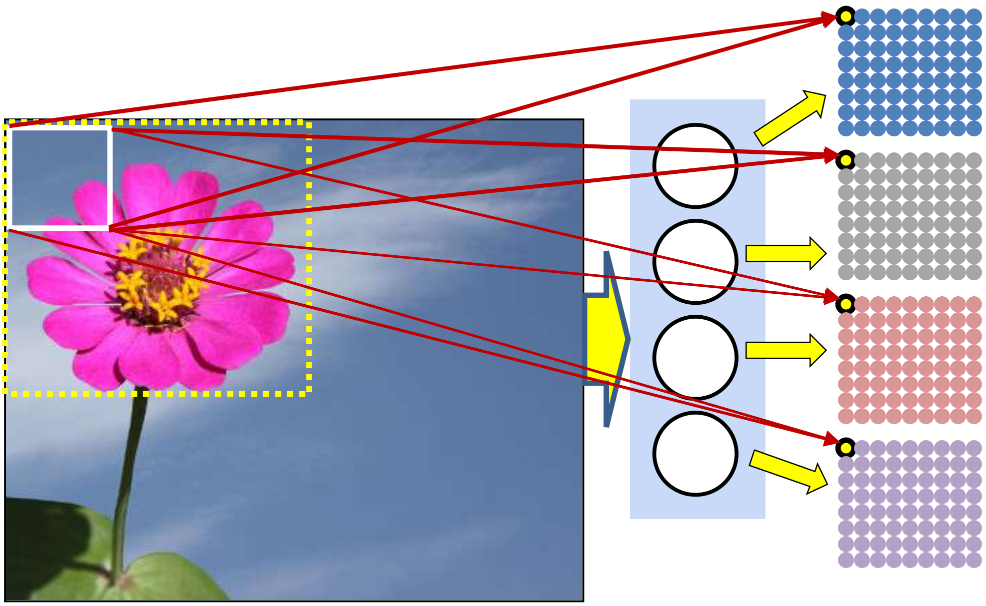
- The entire MLP looks for a flower-like pattern at each location

The behavior of the layers



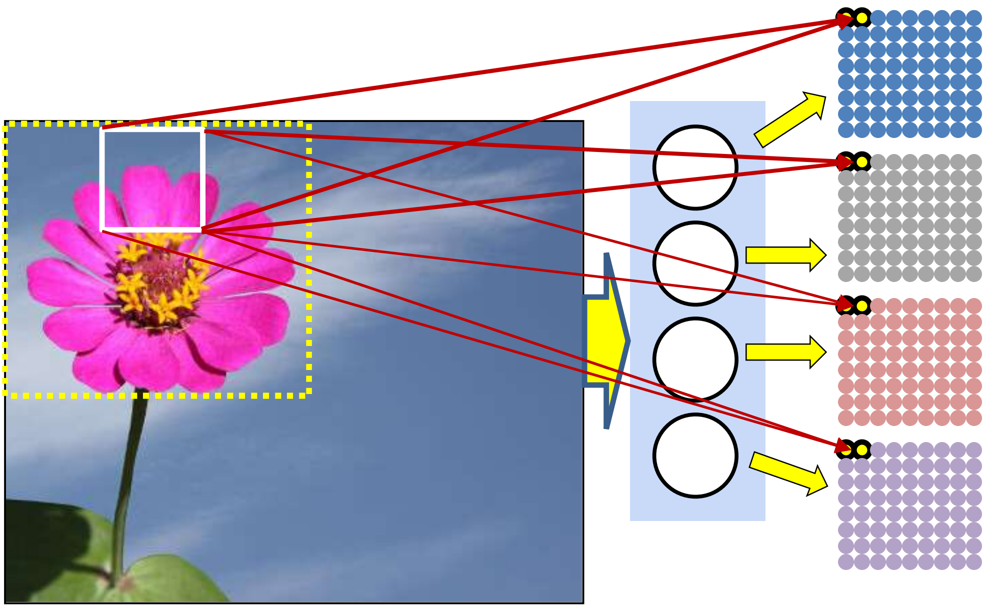
- The first layer neurons "look" at the entire "window" to extract window-level features
  - Subsequent layers only perform classification over these window-level features
- **The first layer neurons is responsible for evaluating the entire *window* of pixels**
  - **Subsequent layers only look at a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

Distributing the scan



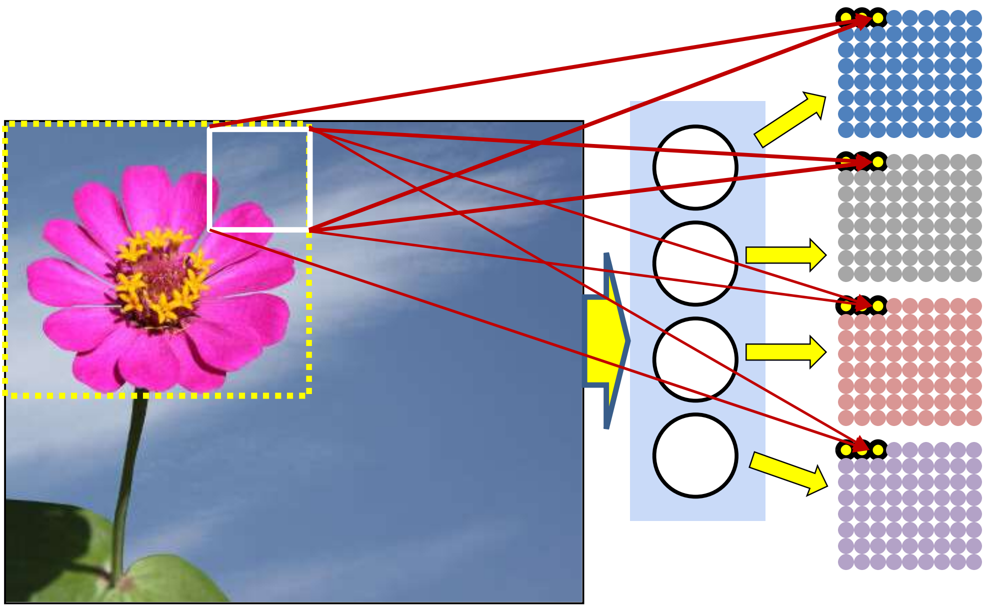
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels

Distributing the scan



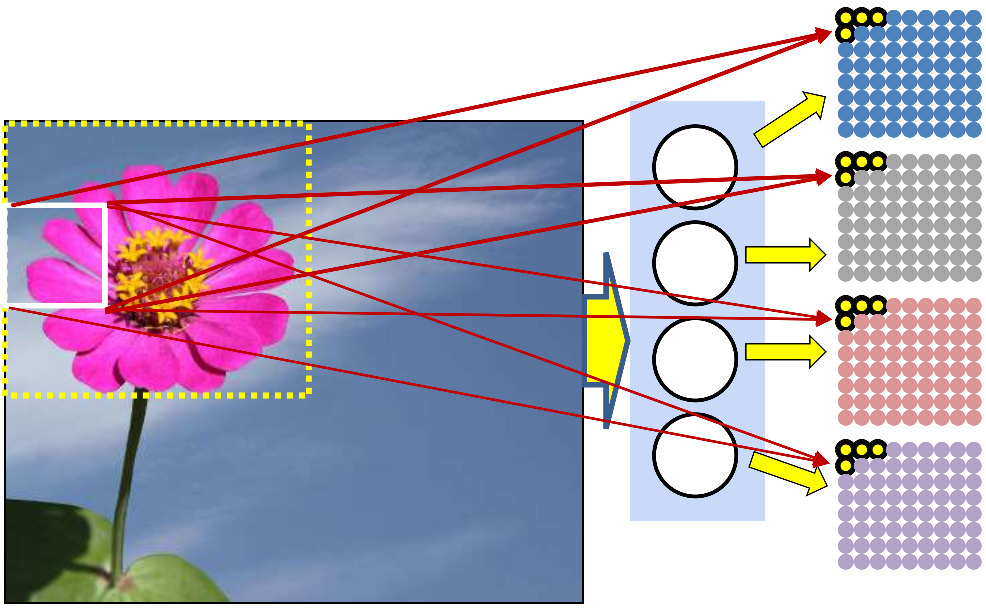
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels

Distributing the scan



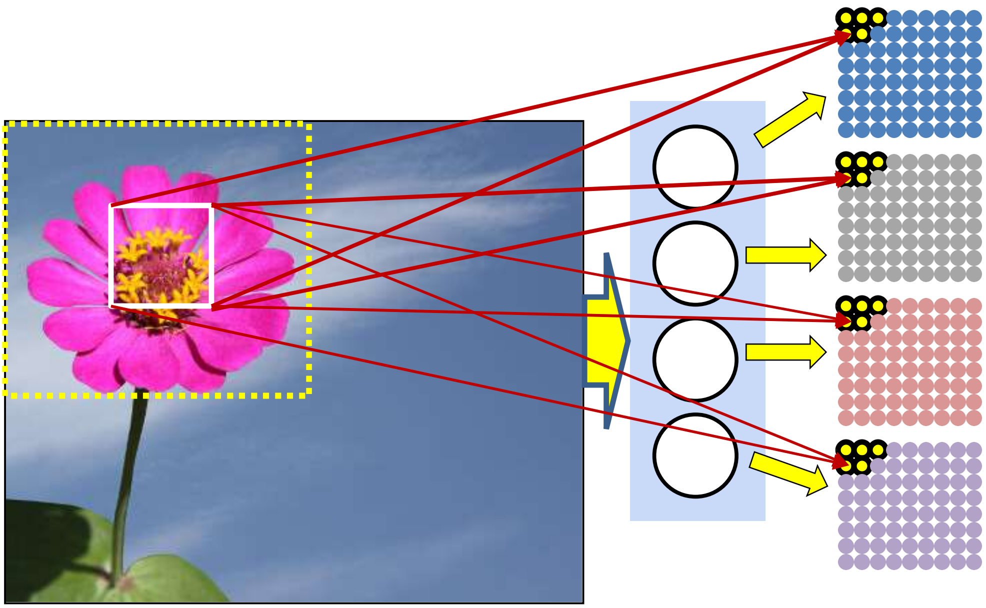
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels

Distributing the scan



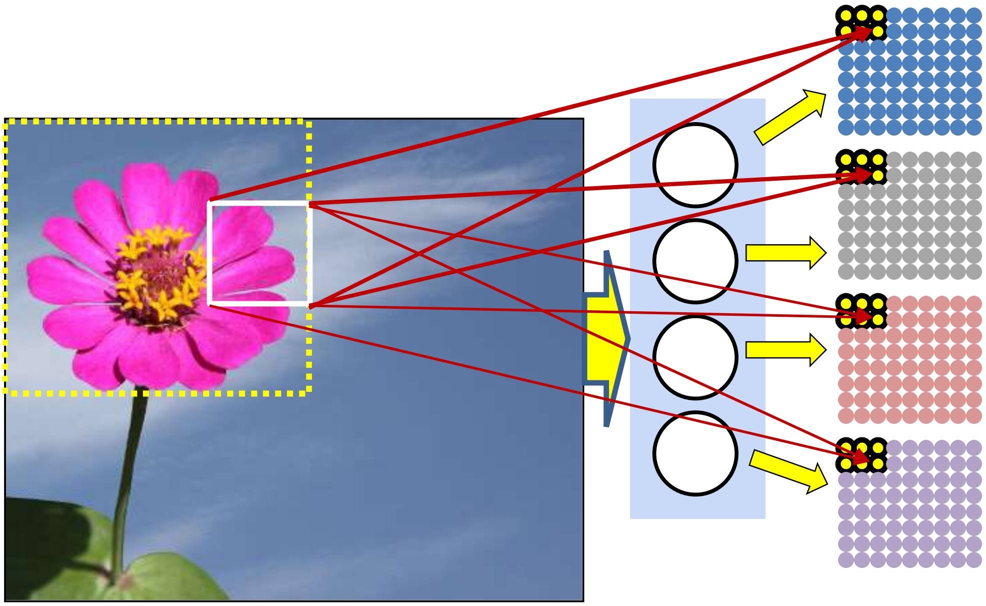
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels

Distributing the scan



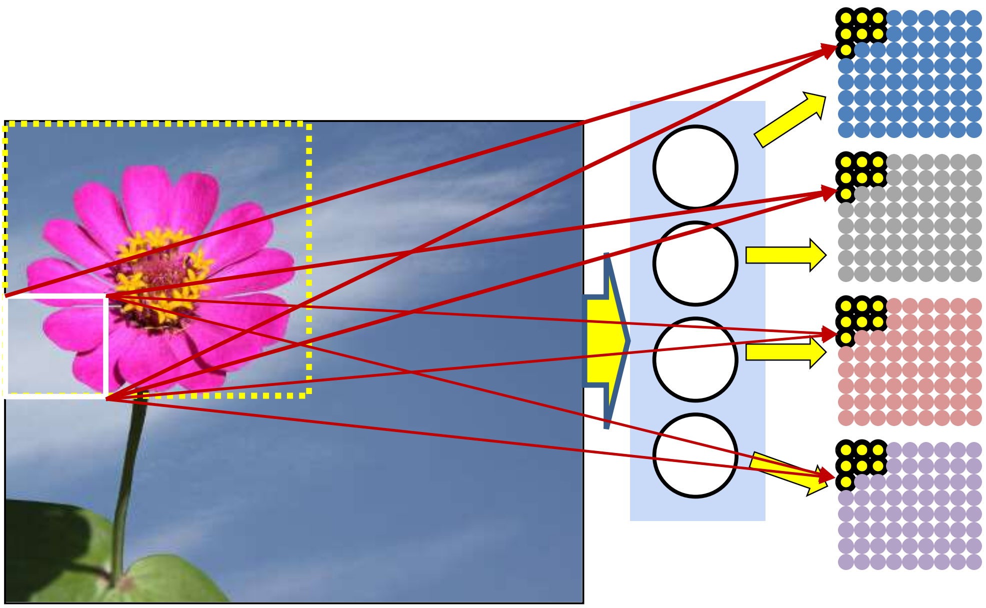
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels

Distributing the scan



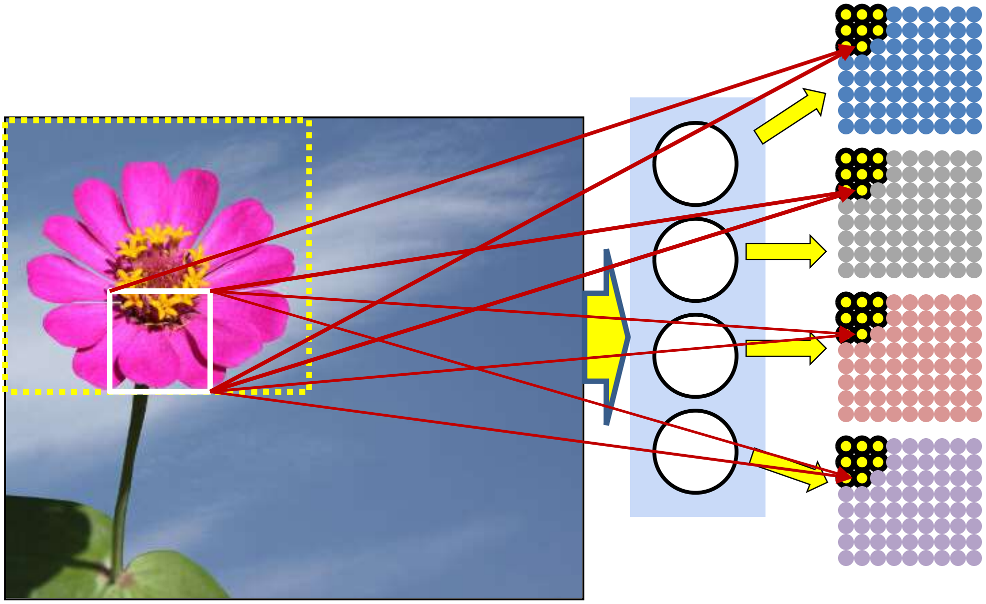
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels

Distributing the scan



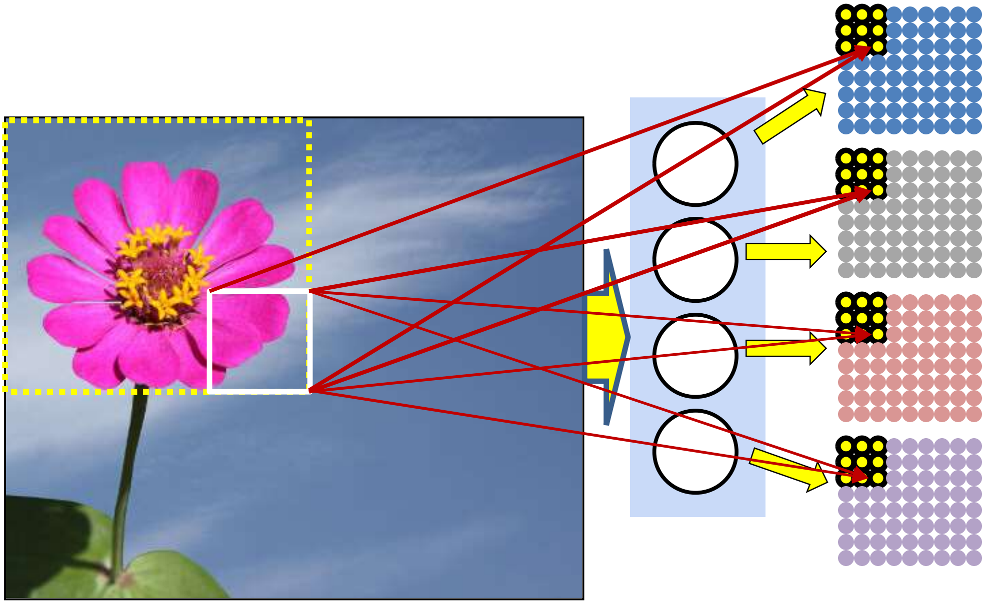
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  – The first layer evaluates smaller blocks of pixels

Distributing the scan



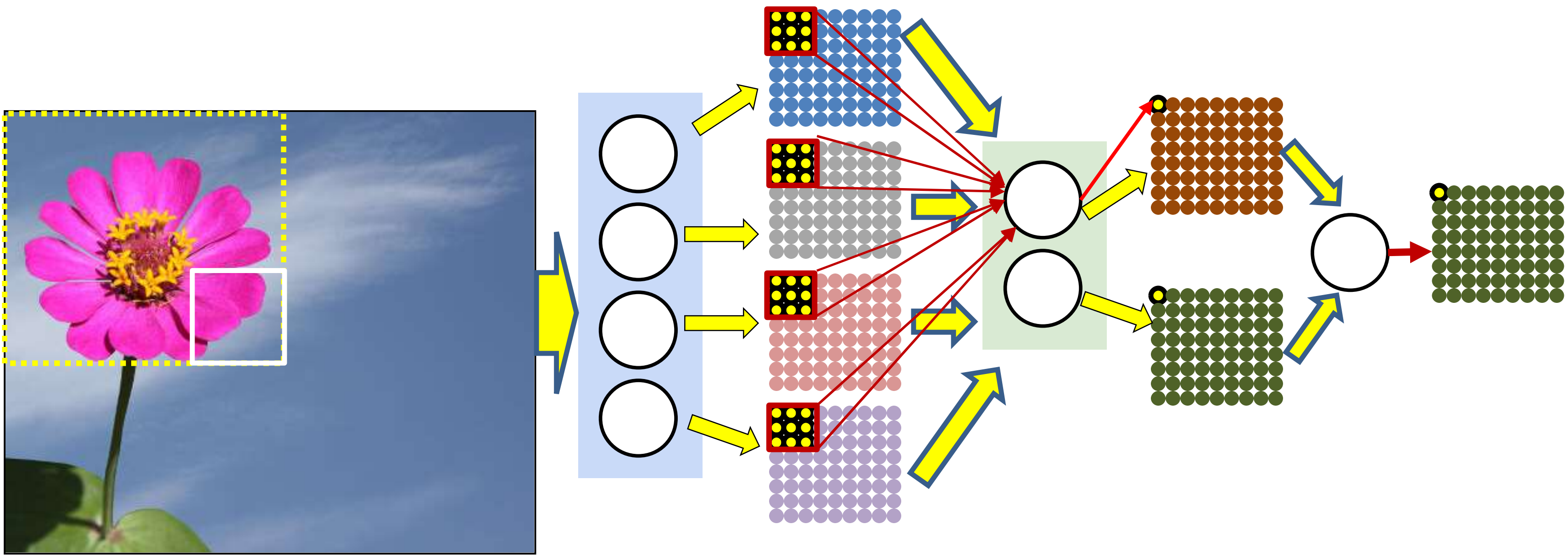
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels

Distributing the scan



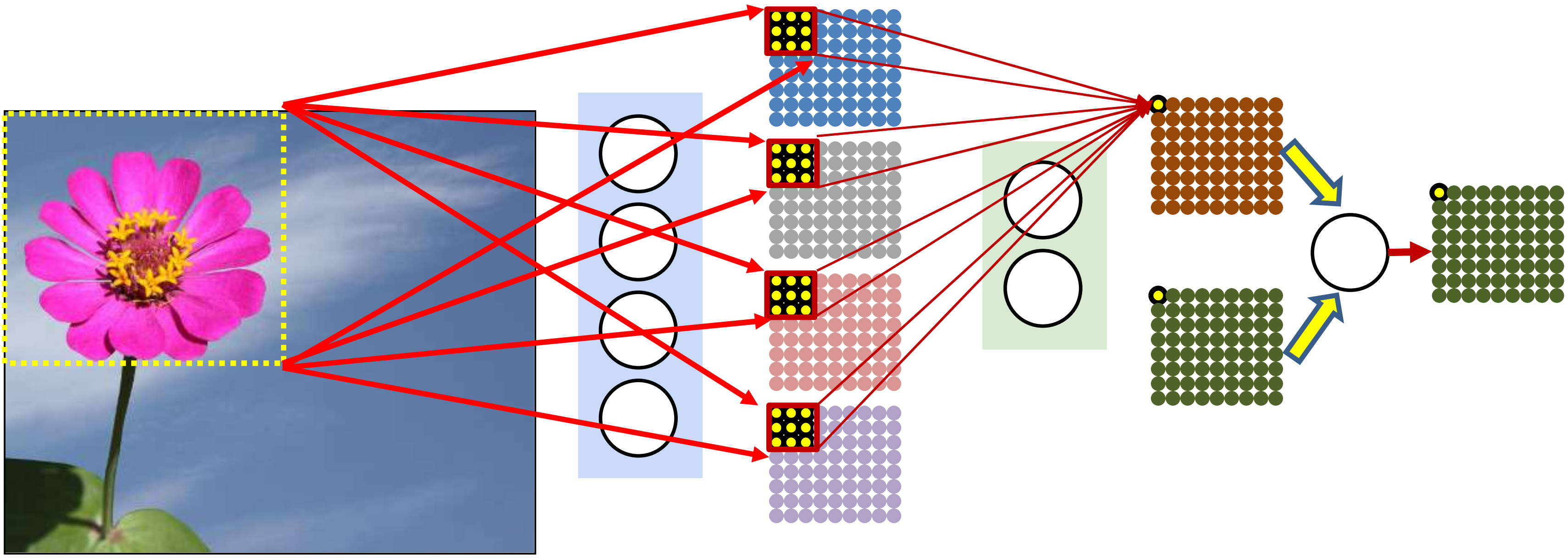
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels

Distributing the scan



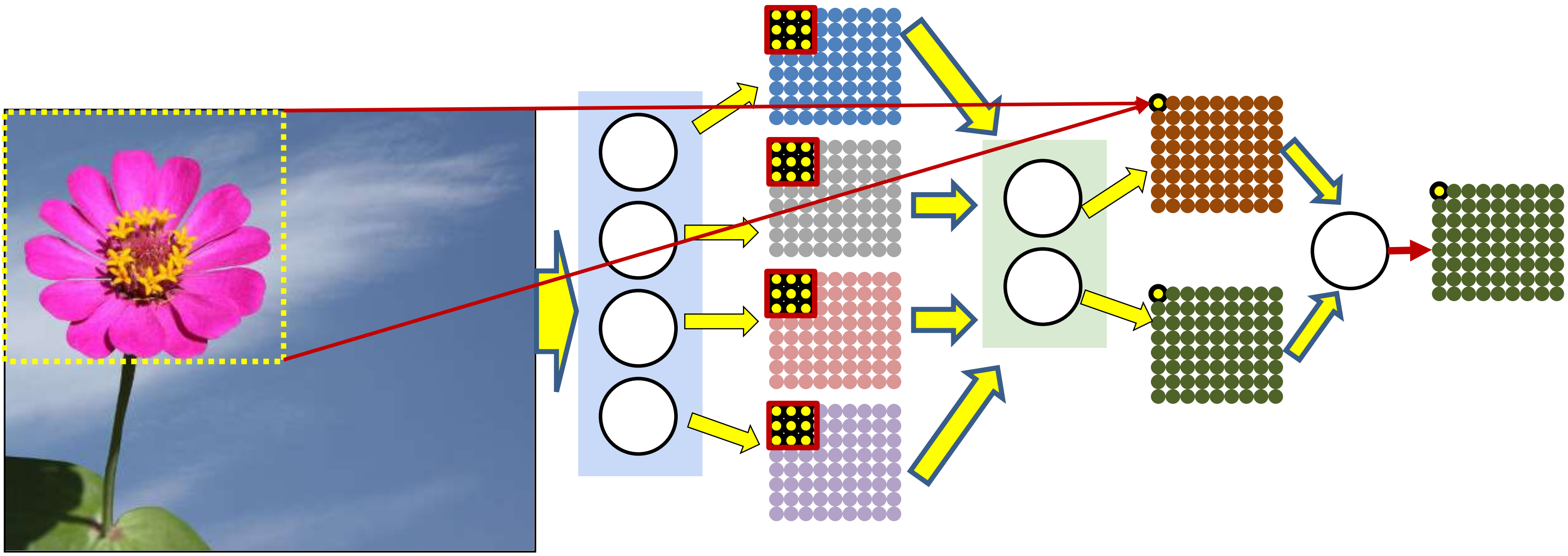
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates blocks of outputs from the first layer

Distributing the scan



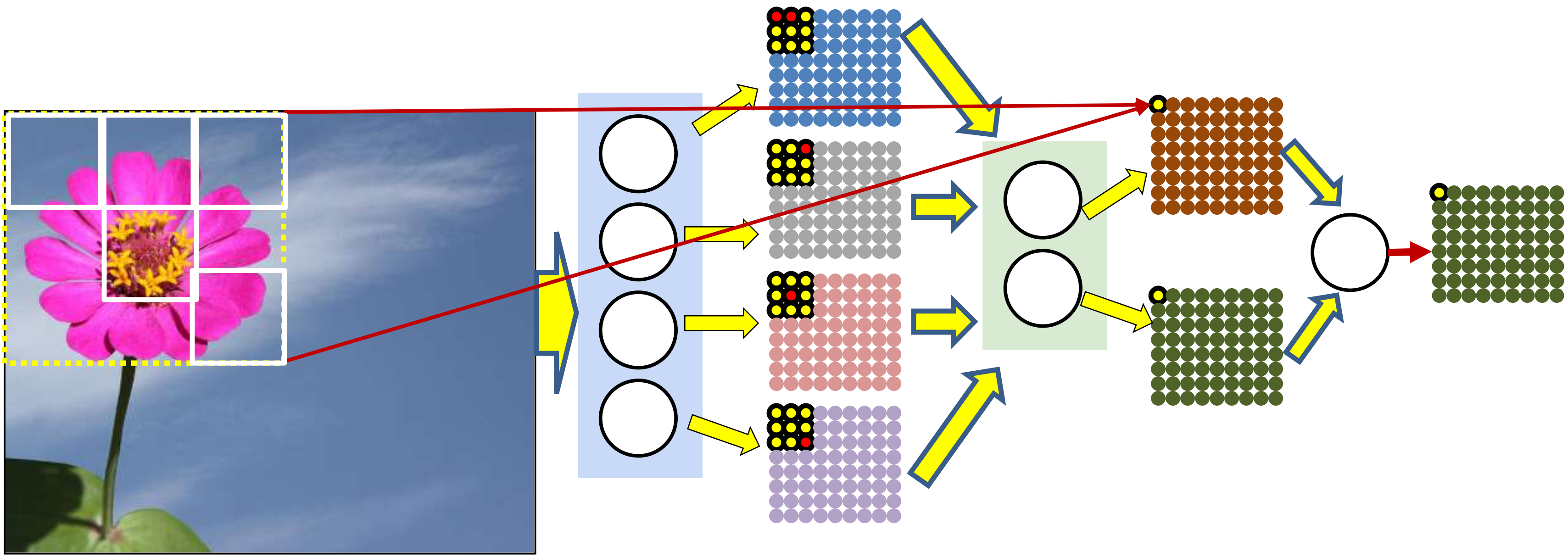
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  – The first layer evaluates smaller blocks of pixels
  – The next layer evaluates the windows of outputs from the first layer

Distributing the scan



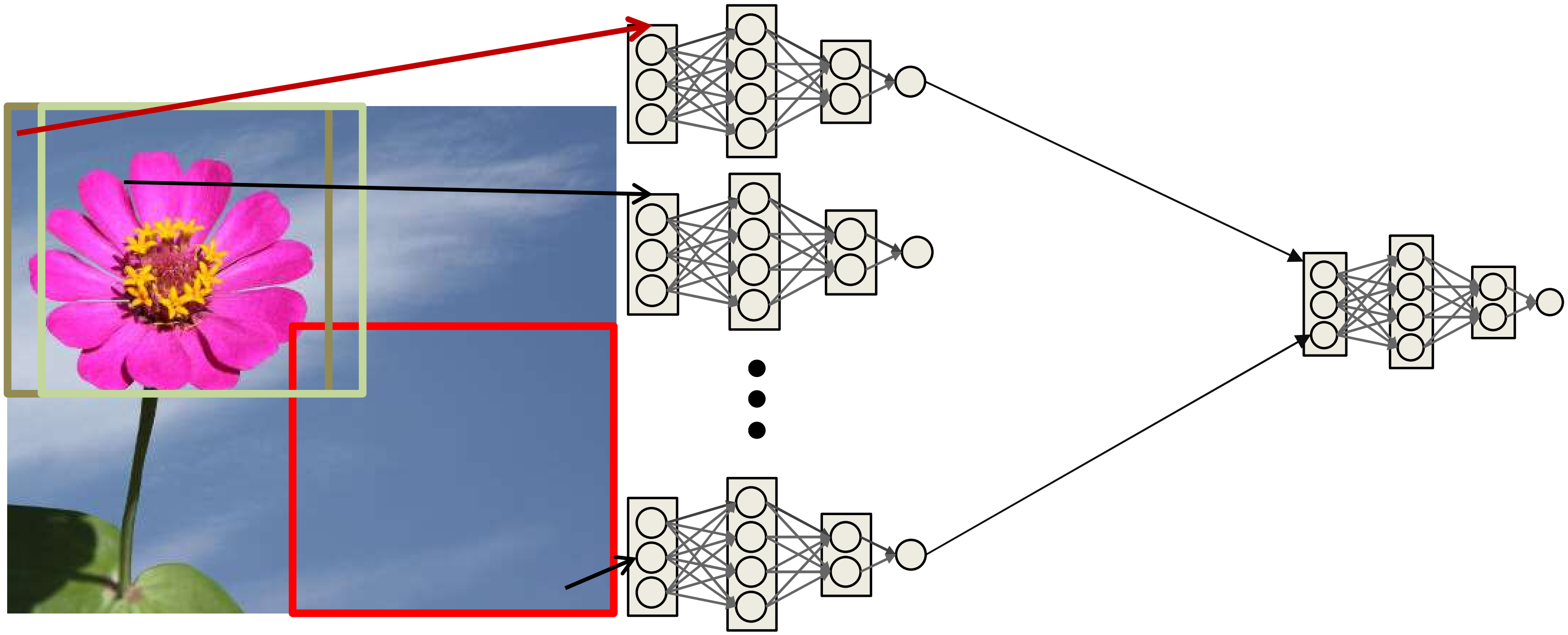
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates windows of outputs from the first layer
  - This effectively evaluates the larger window of the original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

Distributing the scan



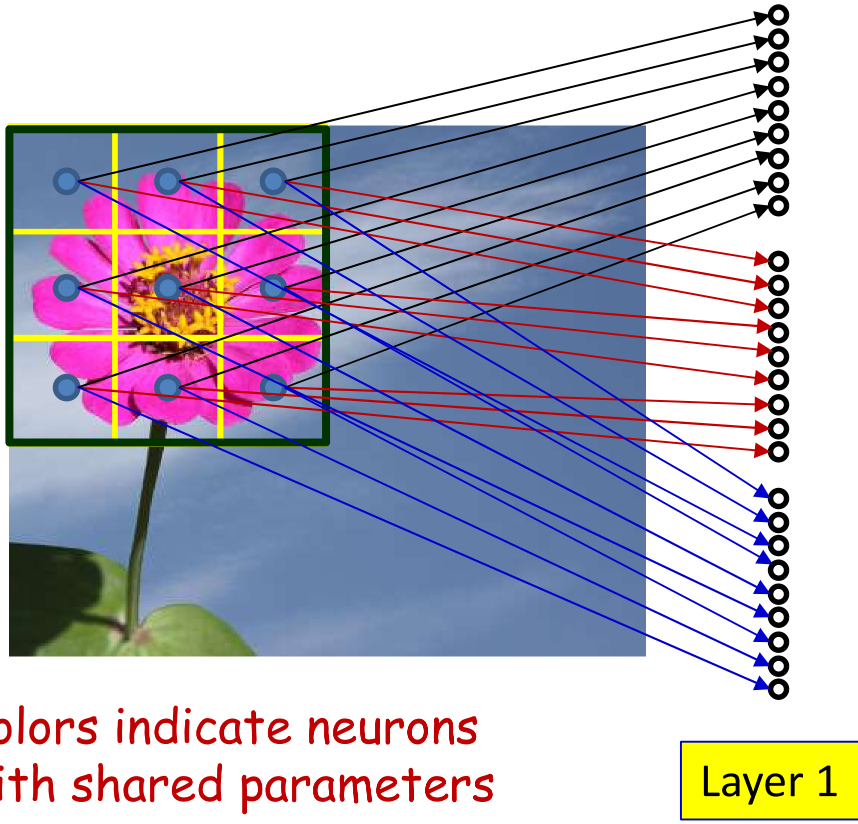
- The window has been *distributed* over two layers
- The higher layer implicitly learns the *arrangement* of sub patterns that represents the larger pattern (the flower in this case)

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



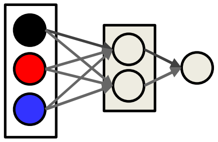
- With a minor modification…

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



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

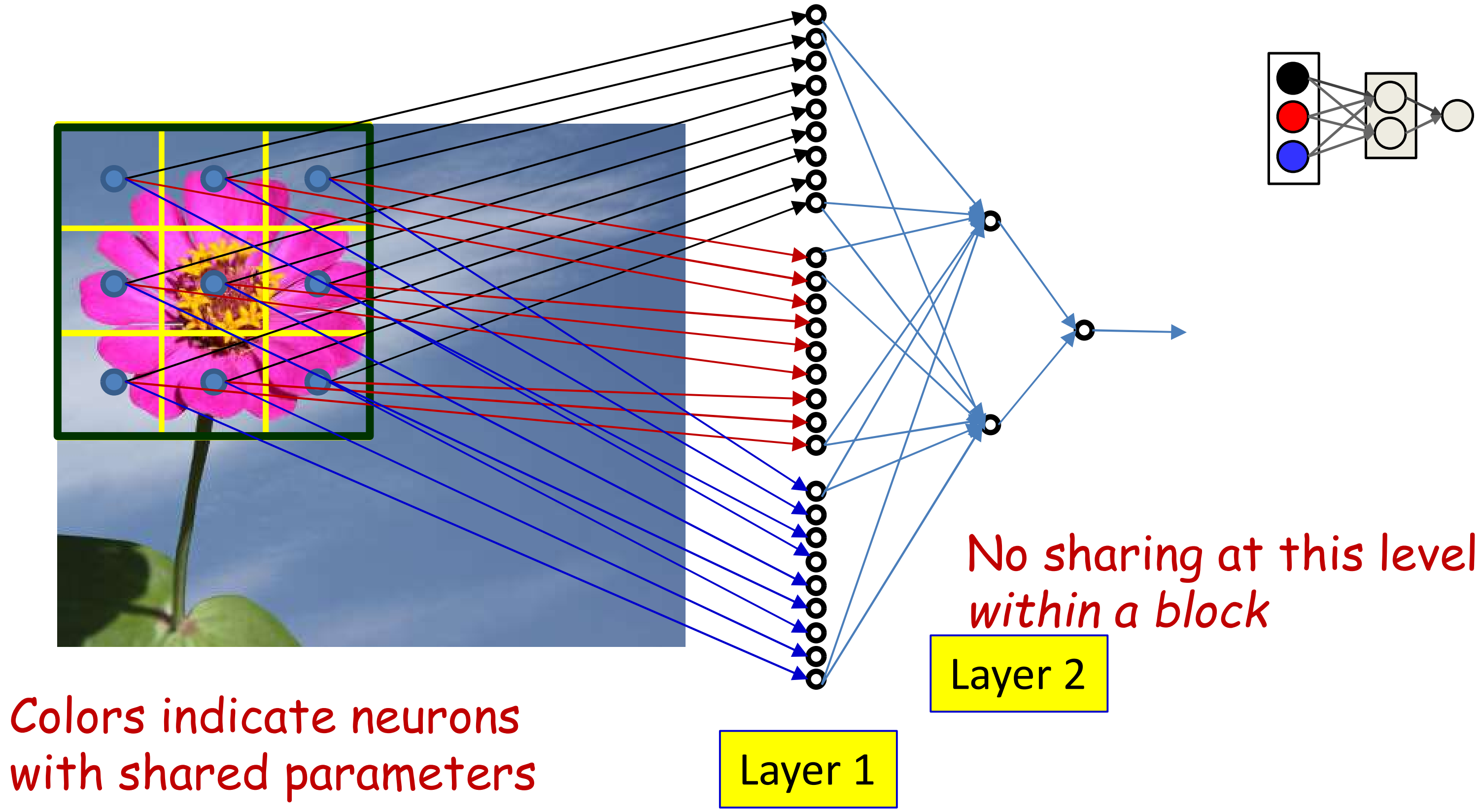
The pattern of weights going out of any cell is identical to that from any other cell.



Colors indicate neurons with shared parameters

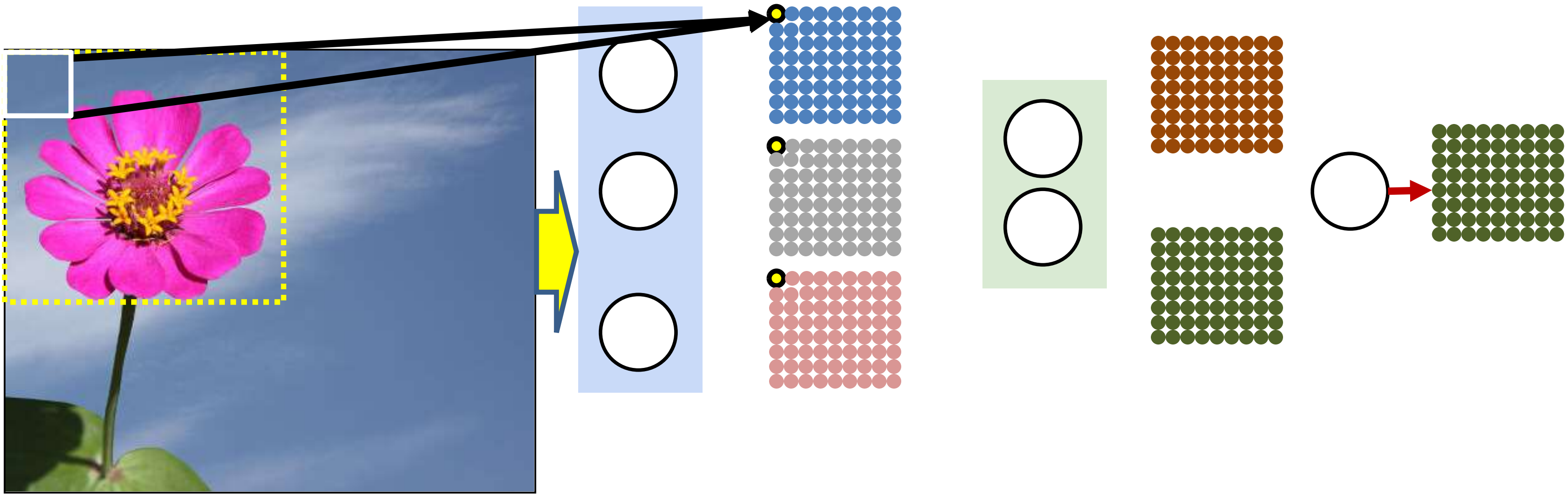
- The network that analyzes individual blocks is now itself a shared parameter network..

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



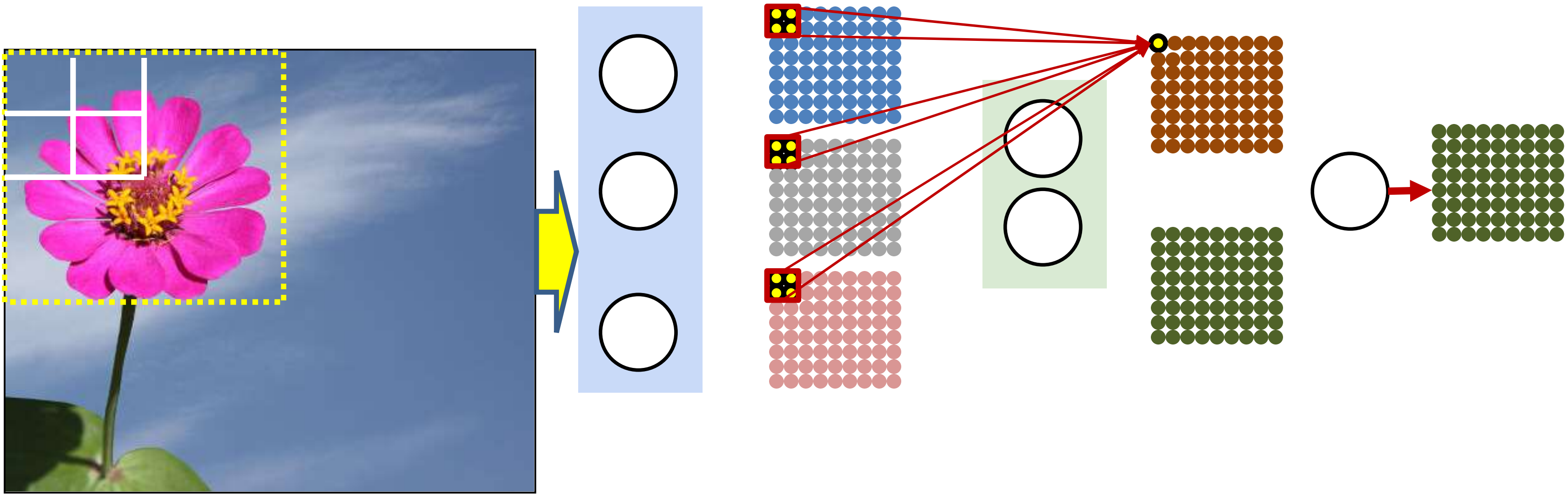
- The network that analyzes individual blocks is now itself a shared parameter network..

This logic can be recursed



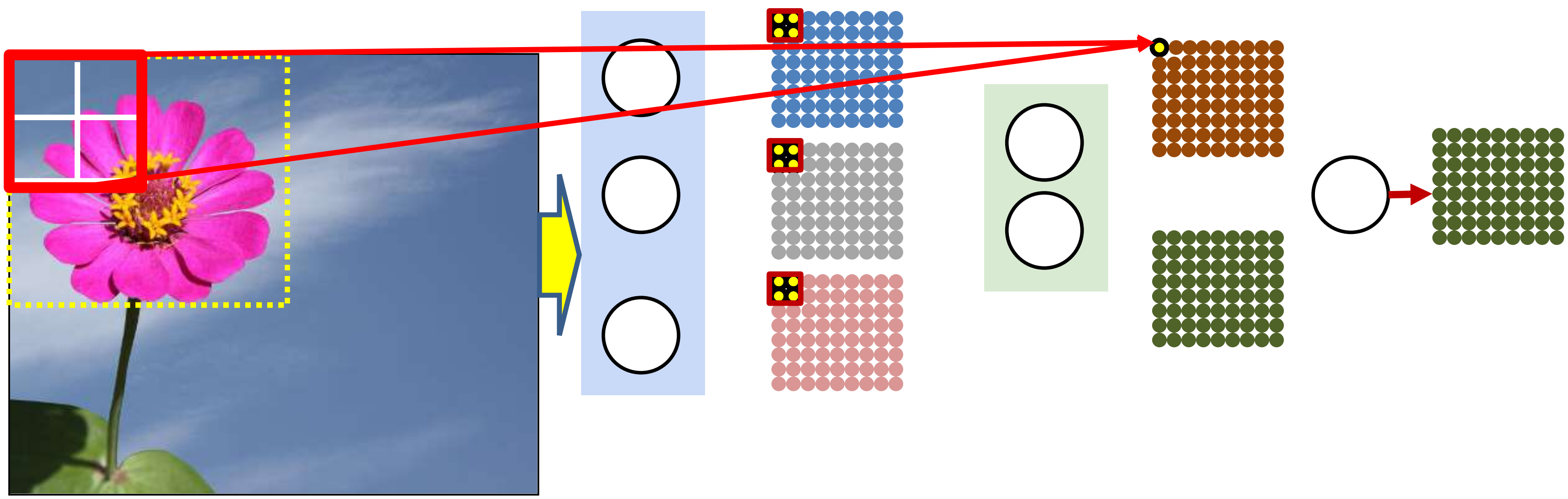
- Building the pattern over 3 layers

This logic can be recursed



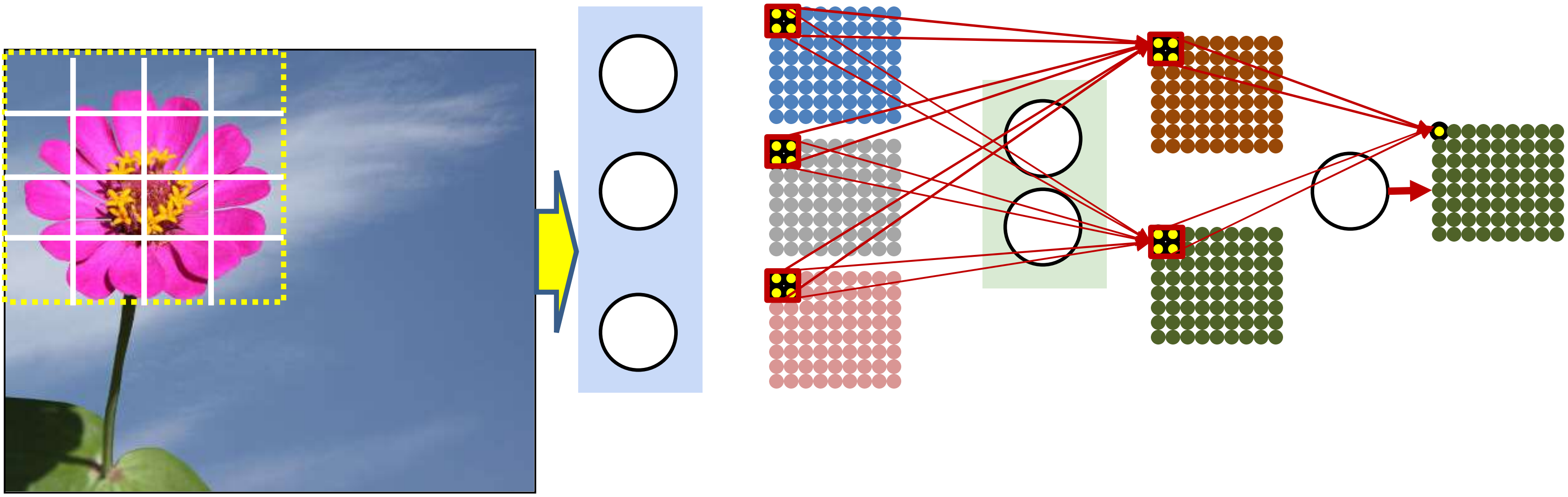
- Building the pattern over 3 layers

This logic can be recursed



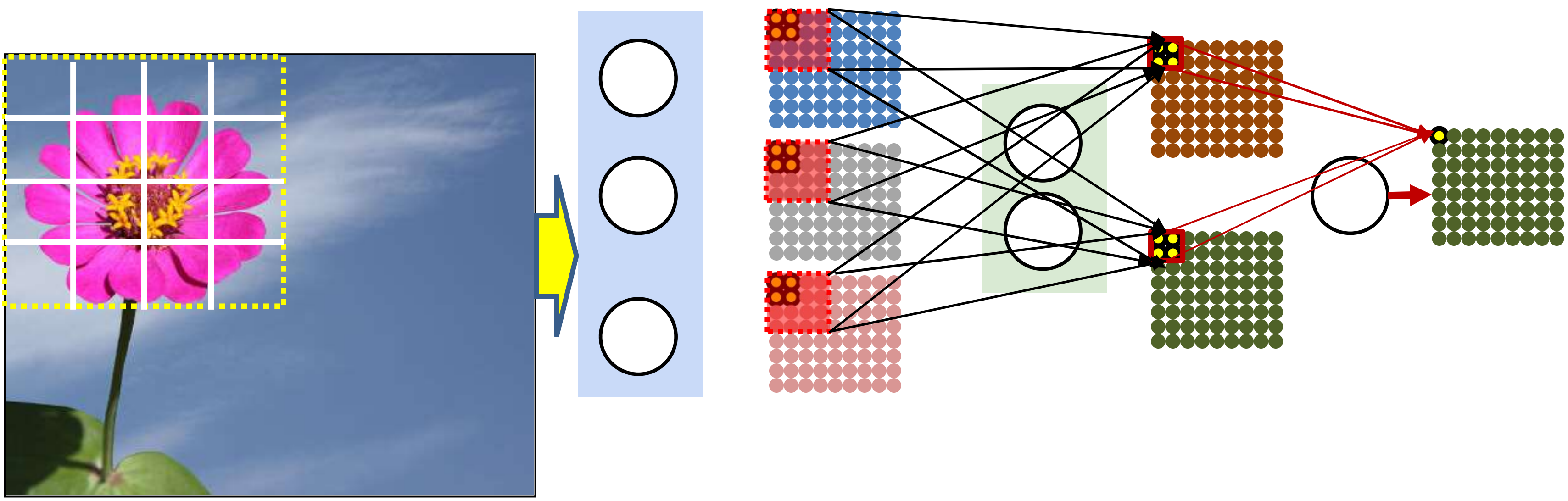
- Building the pattern over 3 layers

This logic can be recursed



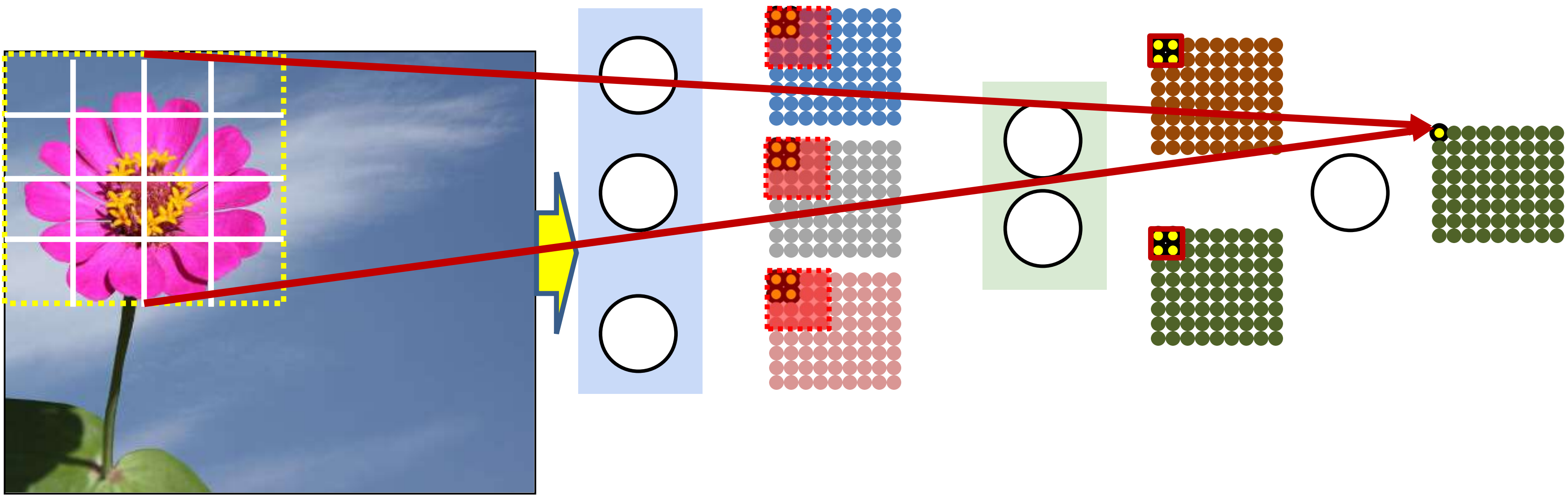
- Building the pattern over 3 layers

This logic can be recursed



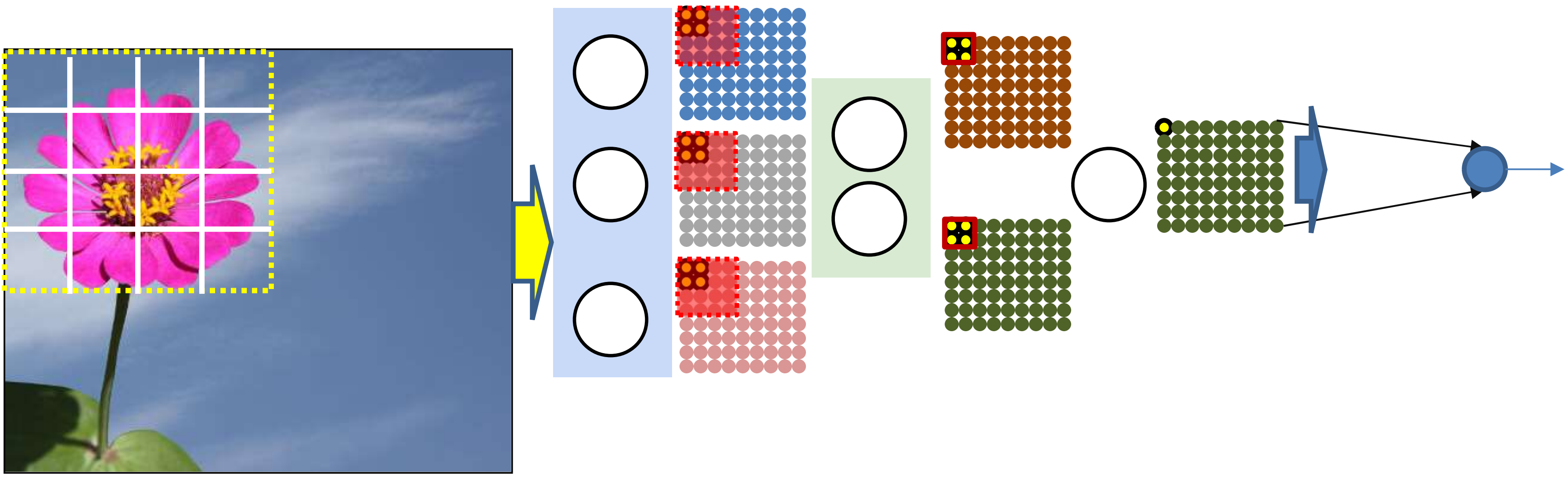
- Building the pattern over 3 layers

This logic can be recursed



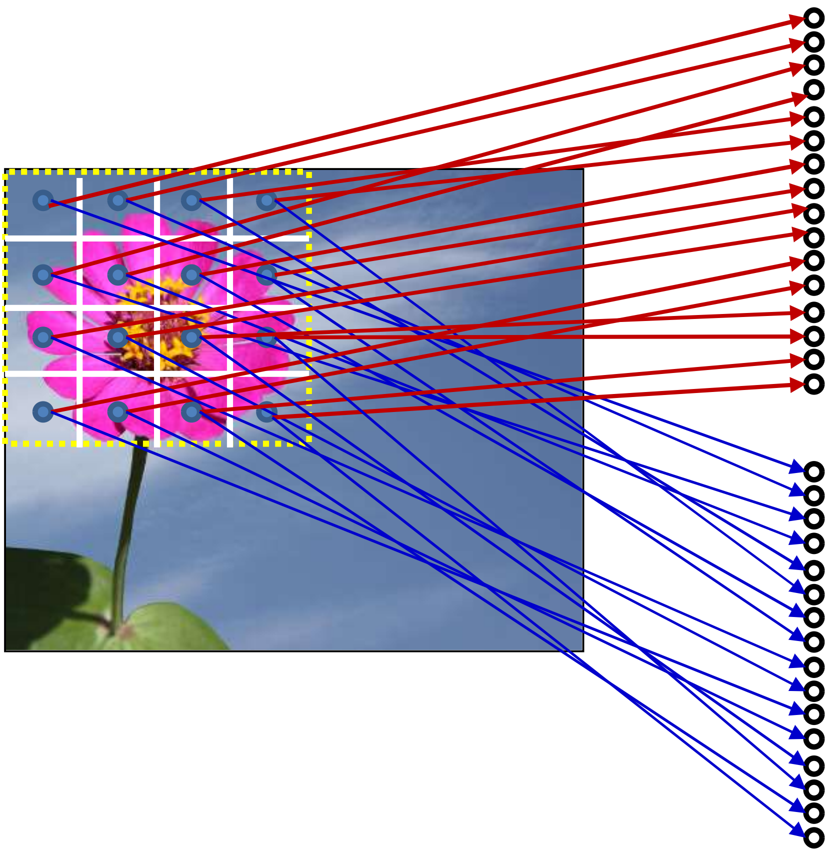
- Building the pattern over 3 layers

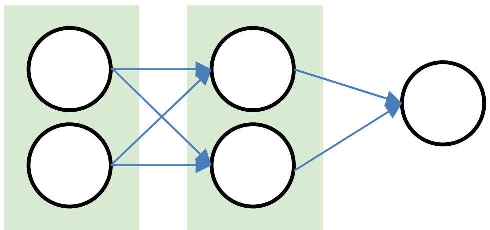
Does the picture have a flower



- 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

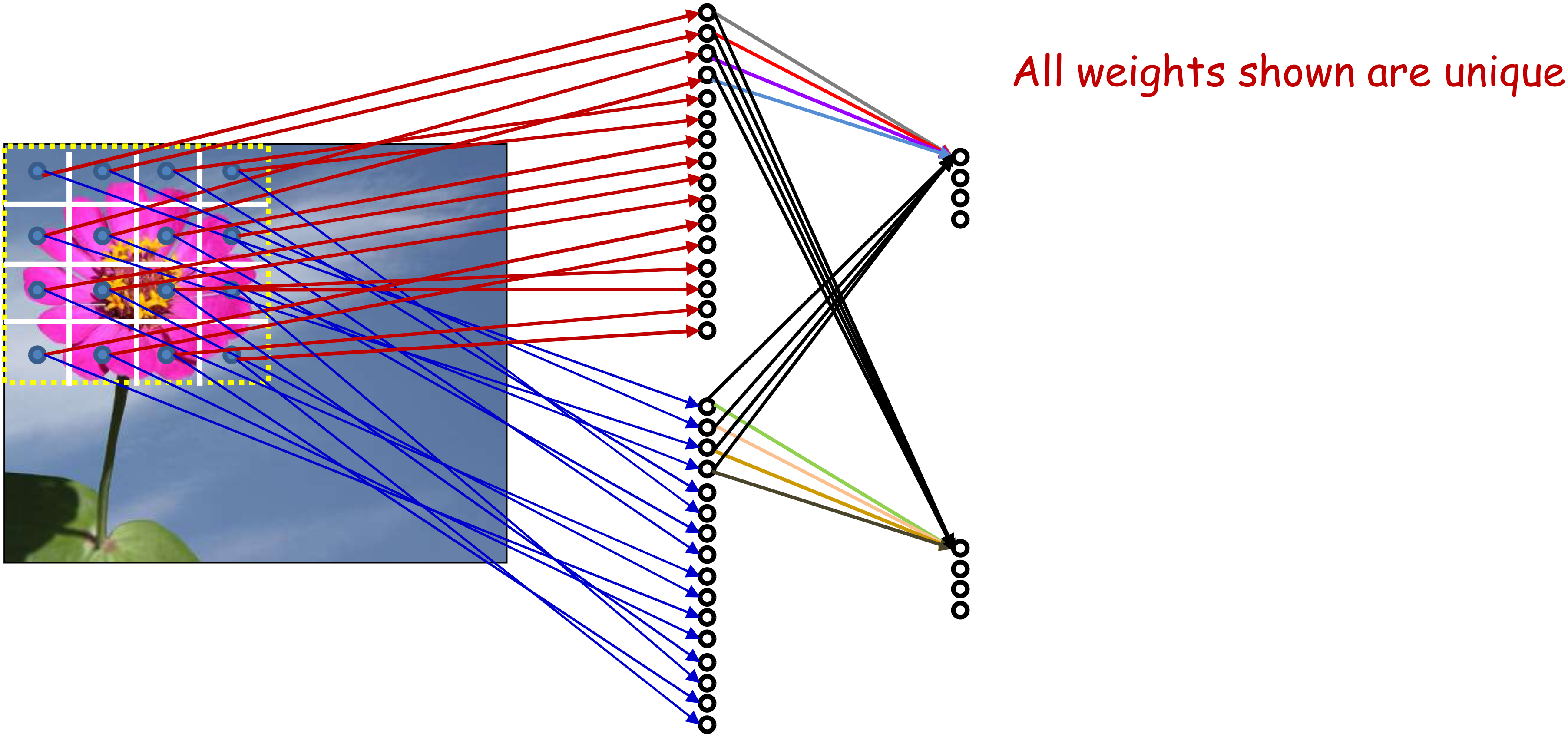




Showing a simpler 2x2x1 network to fit on the slide

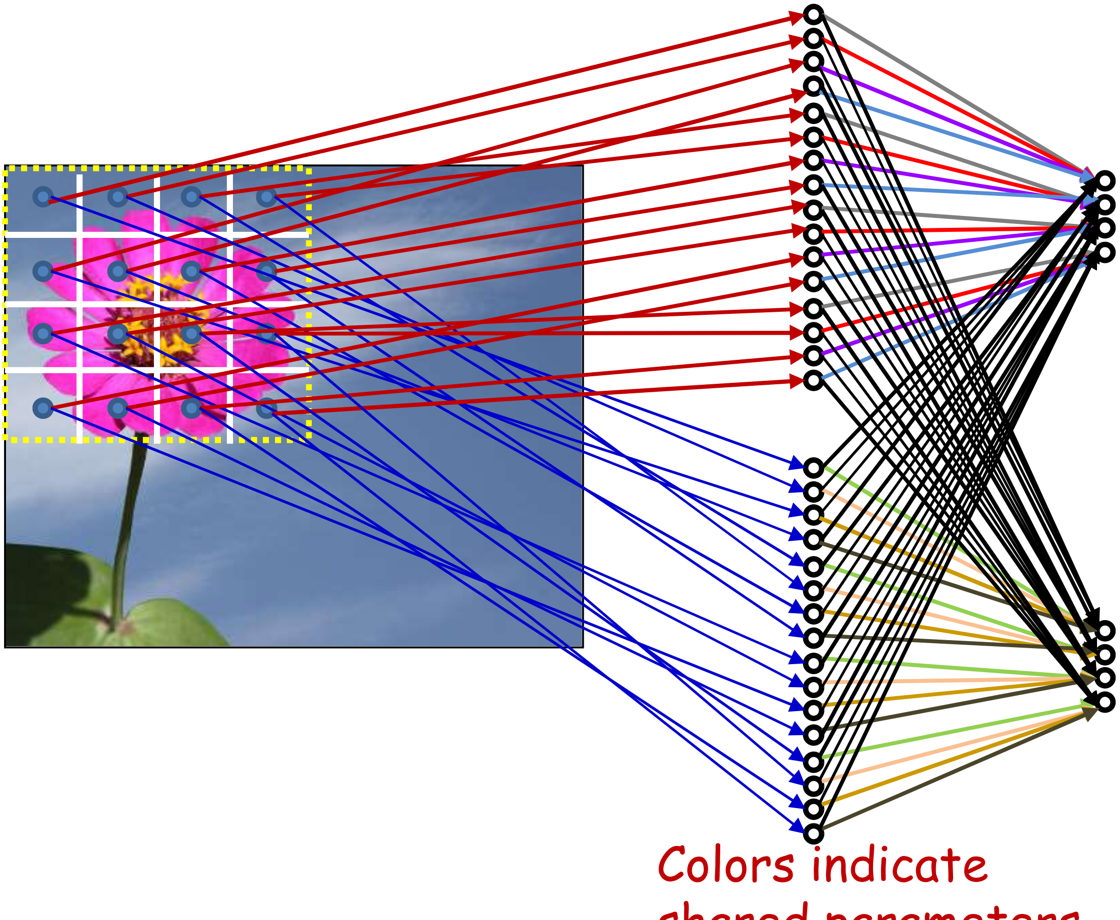
- Building the pattern over 3 layers

The 3-layer shared parameter net



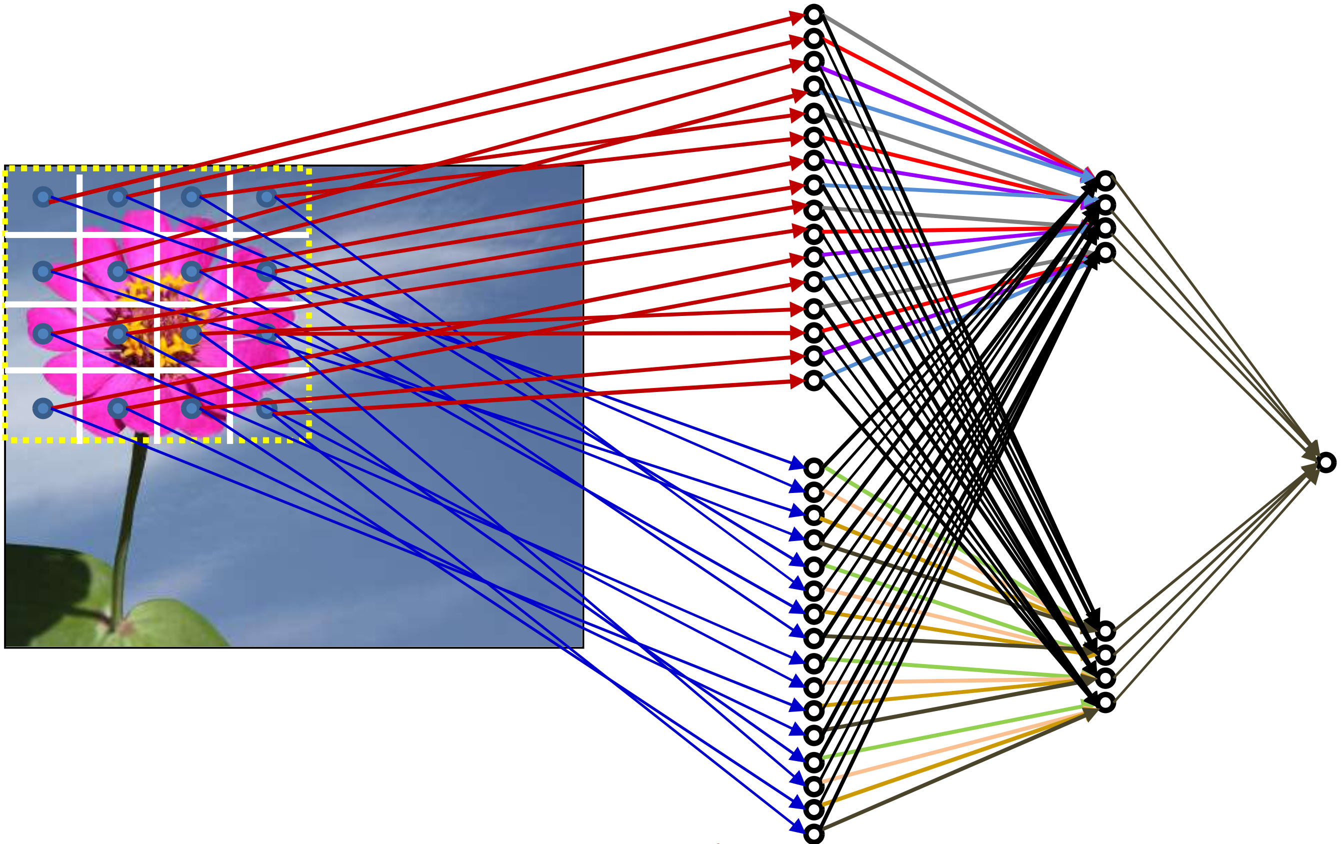
- Building the pattern over 3 layers

The 3-layer shared parameter net



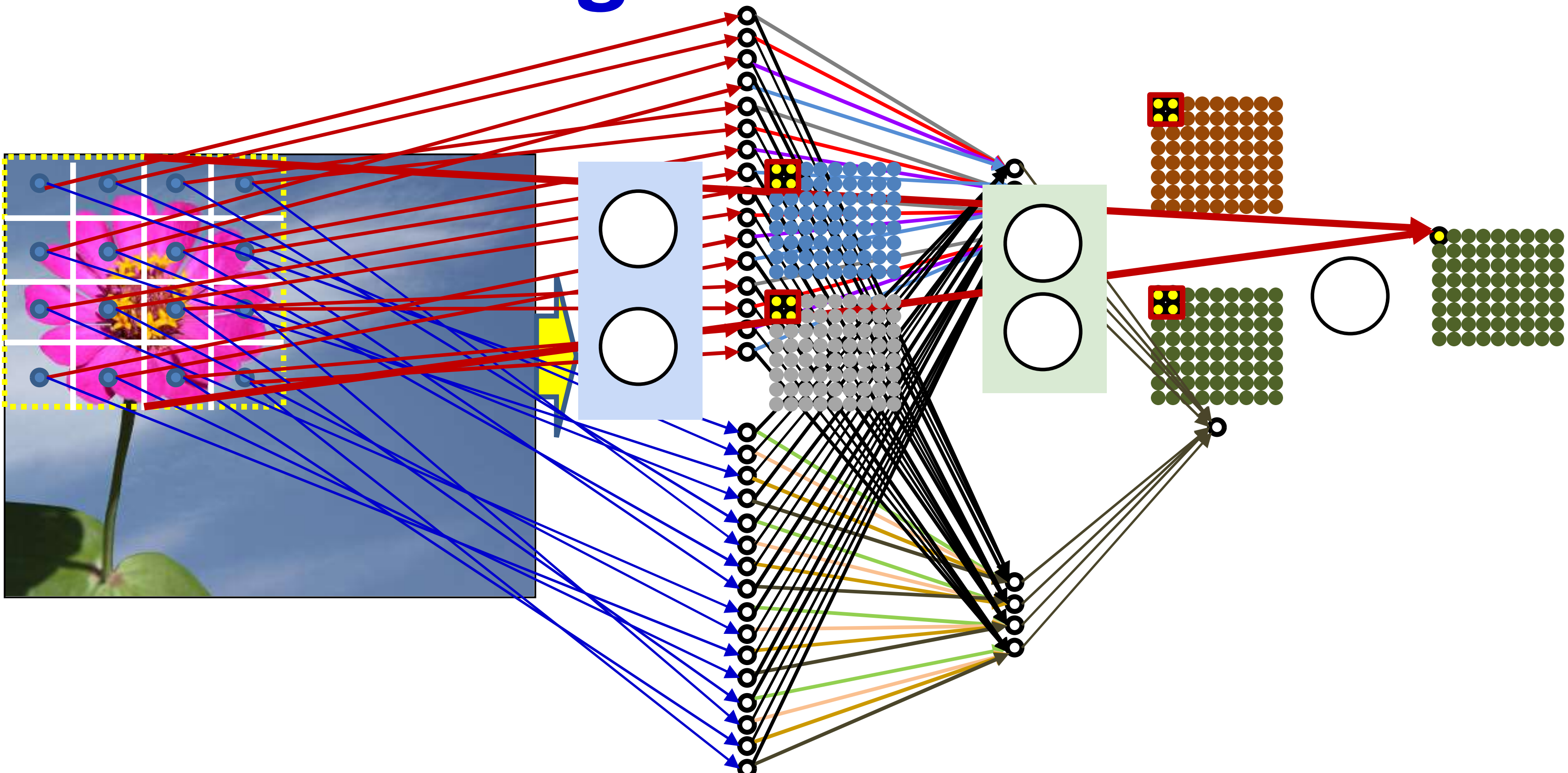
- Building the pattern over 3 layers

The 3-layer shared parameter net



- Building the pattern over 3 layers

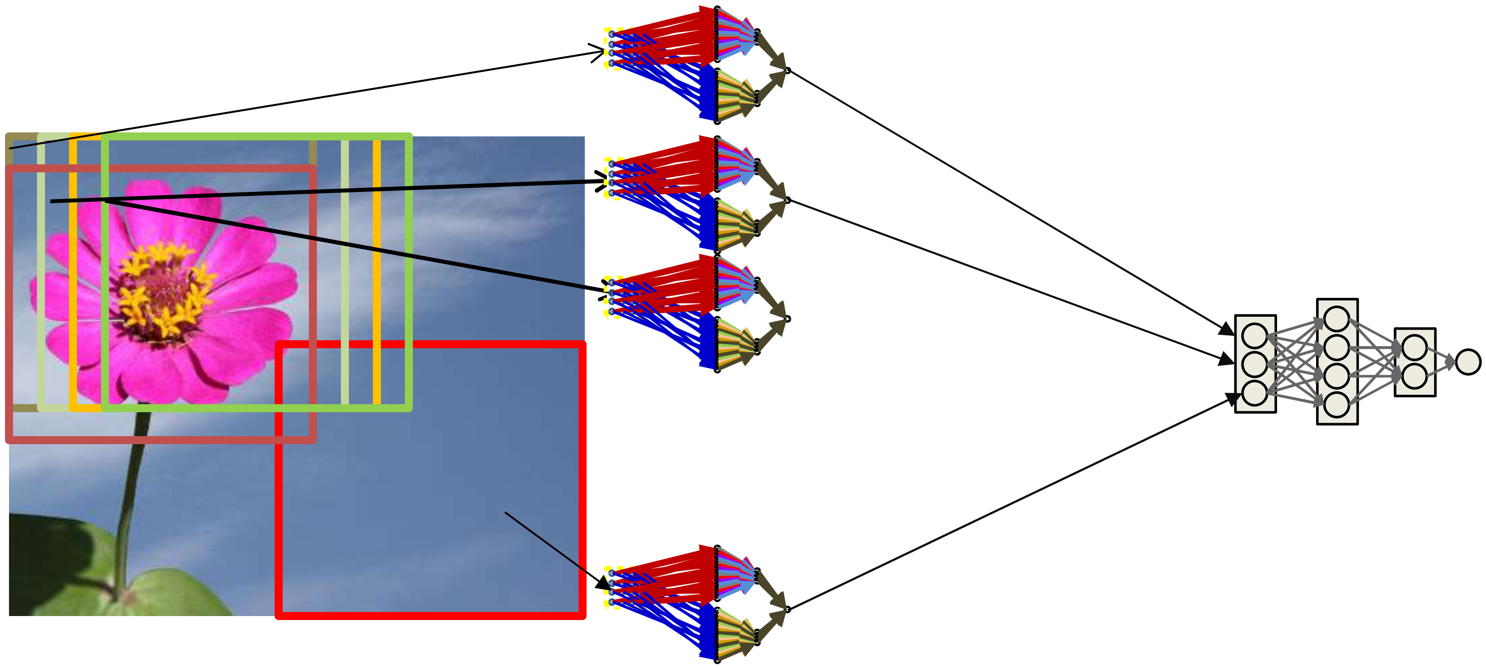
This logic can be recursed



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

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

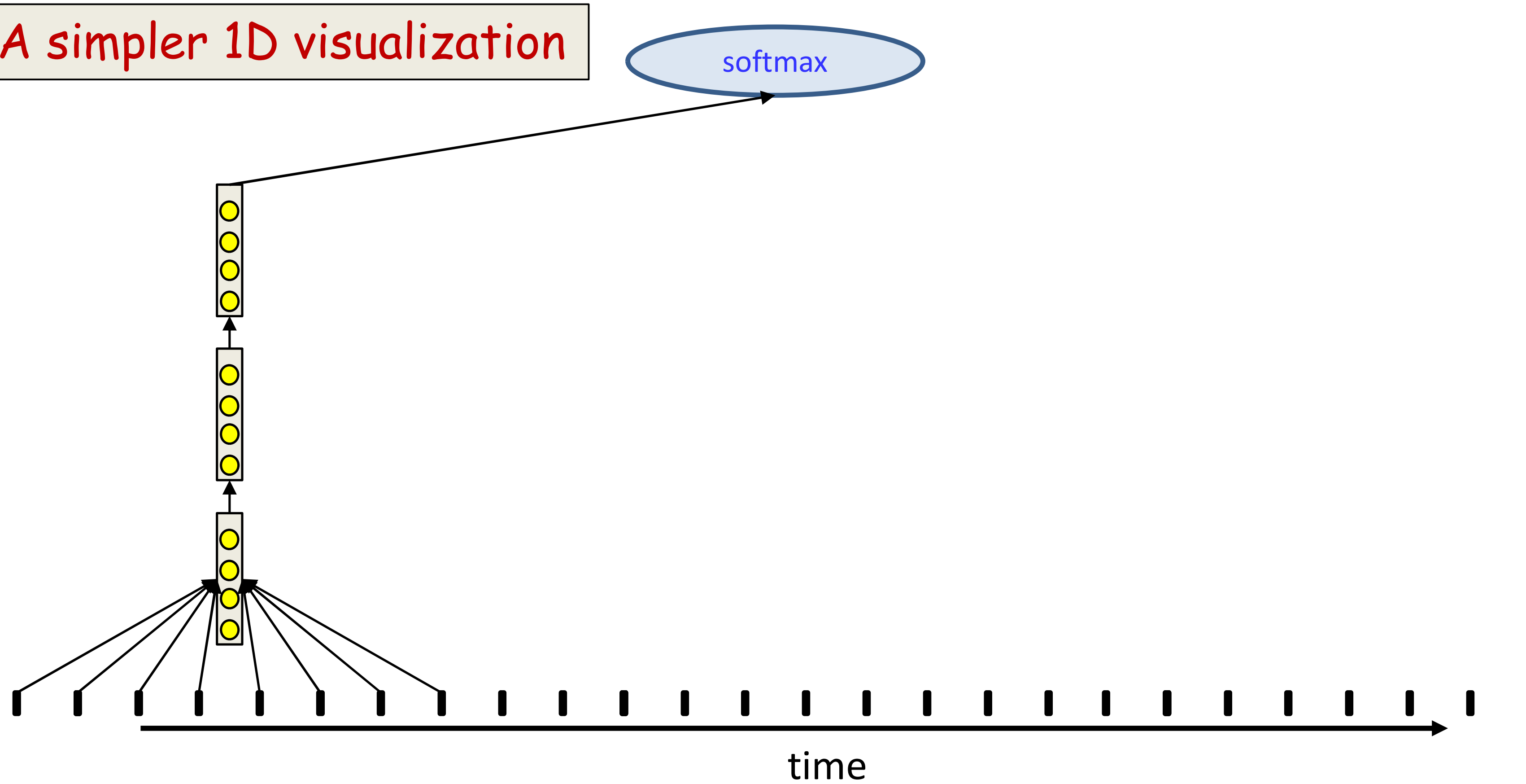
Using hierarchical build-up of features



- 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

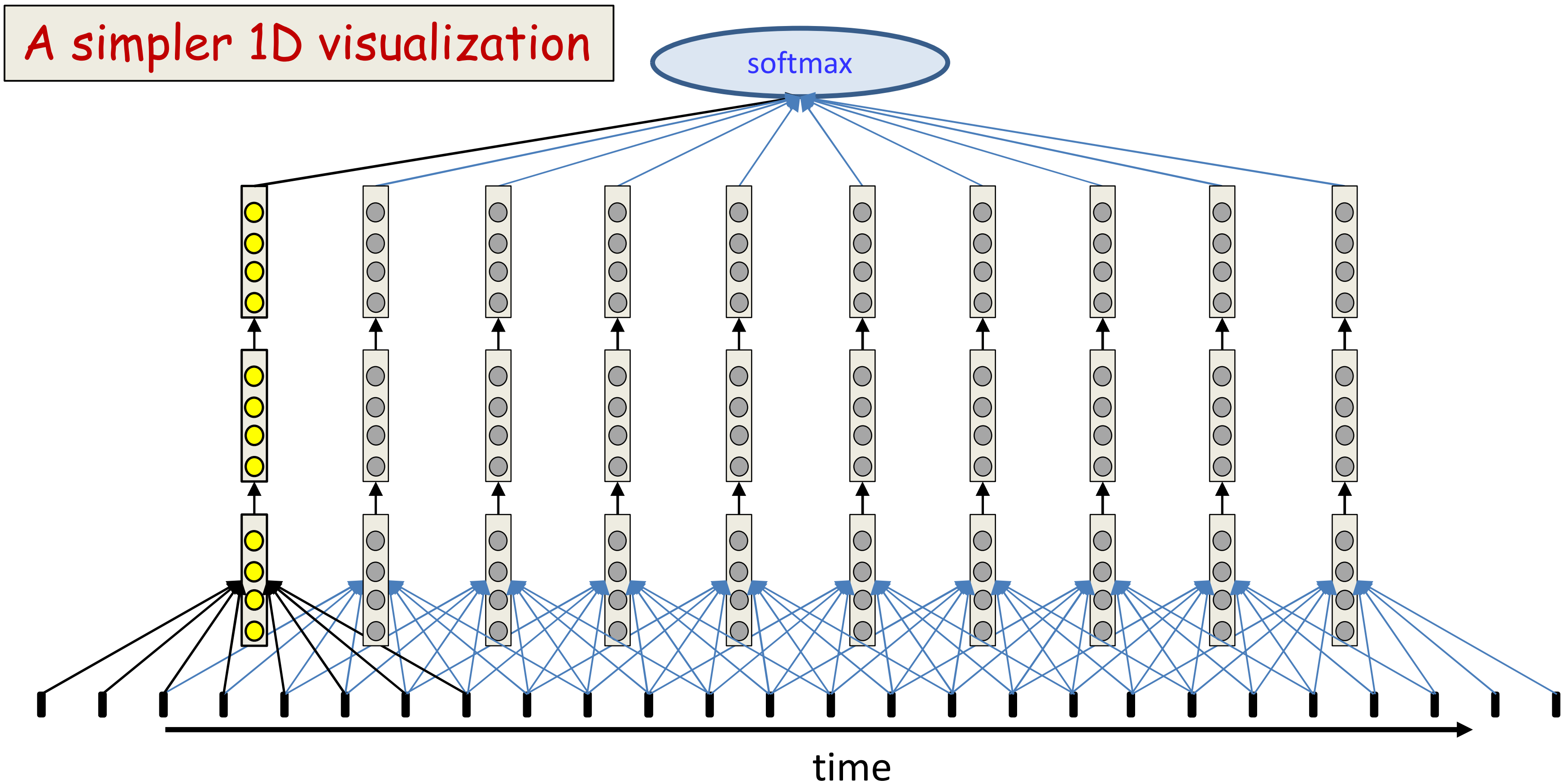
Scanning without distribution

A simpler 1D visualization



- ***Non-distributed*** scan of 8-time-step wide patterns with a stride of two time steps

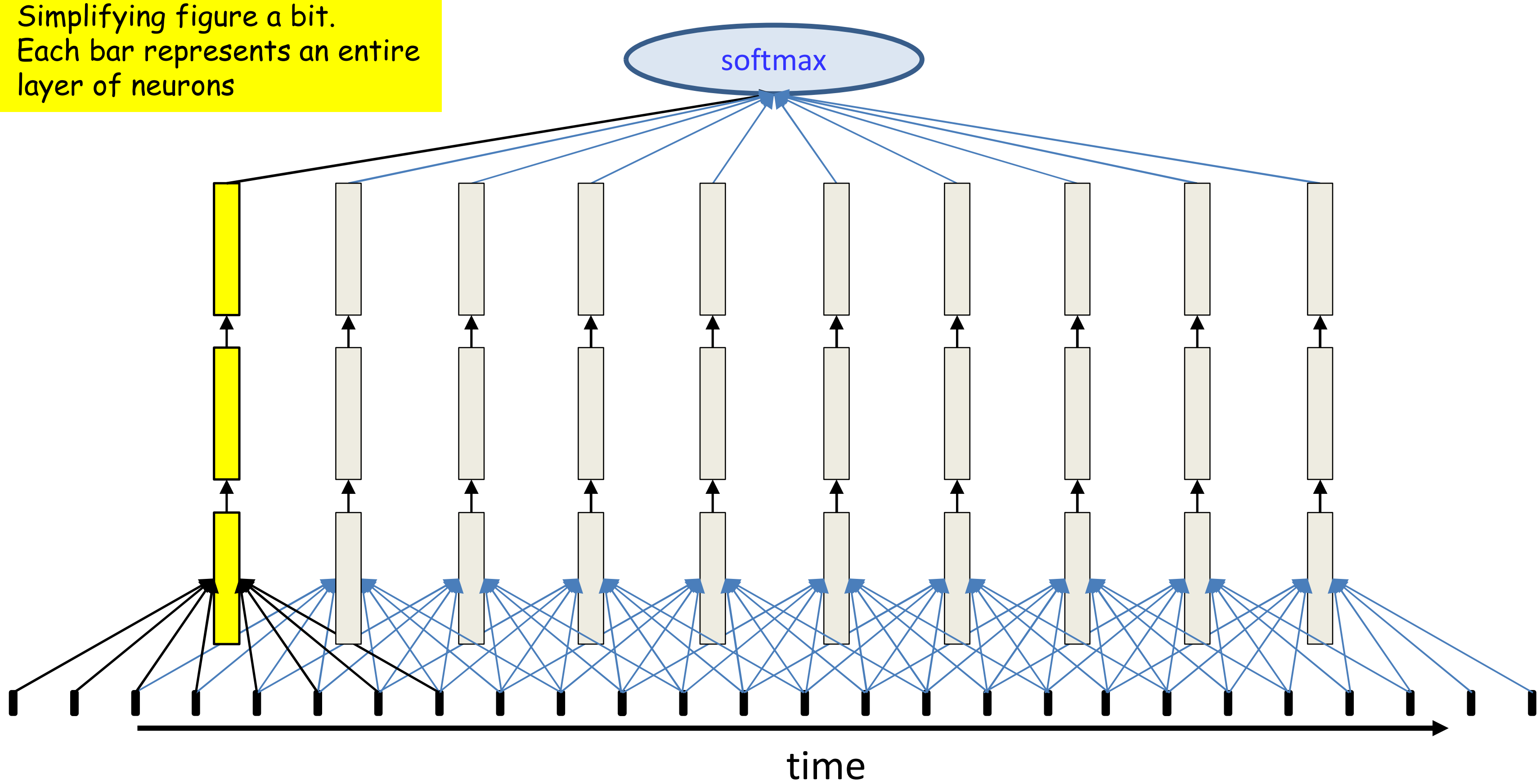
Scanning without distribution



- ***Non-distributed*** scan of 8-time-step wide patterns with a stride of two time steps
- Each column (scanning net) operates independently of every other column
  - No computation is shared across columns

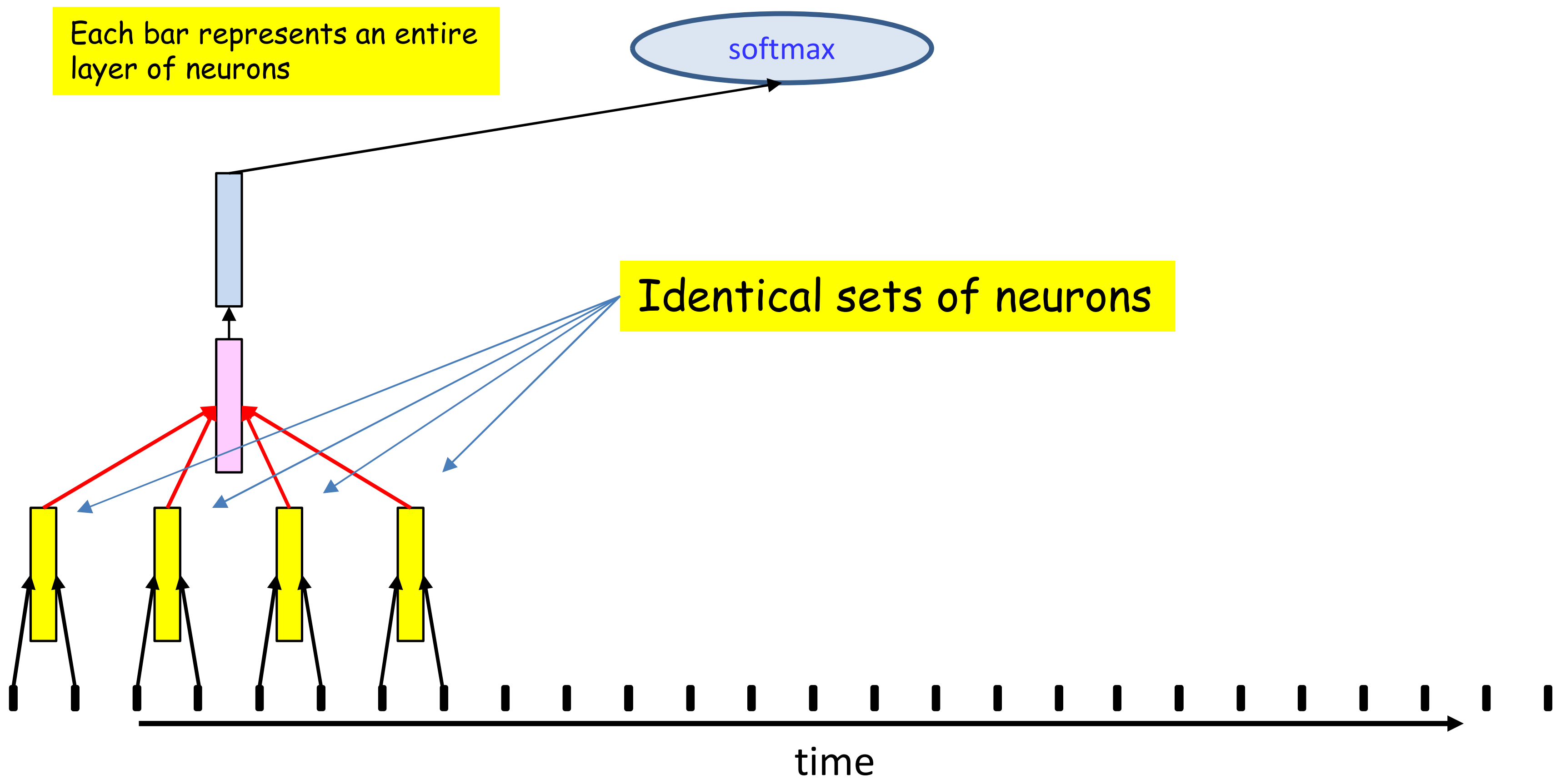
Scanning without distribution

Simplifying figure a bit.
Each bar represents an entire layer of neurons



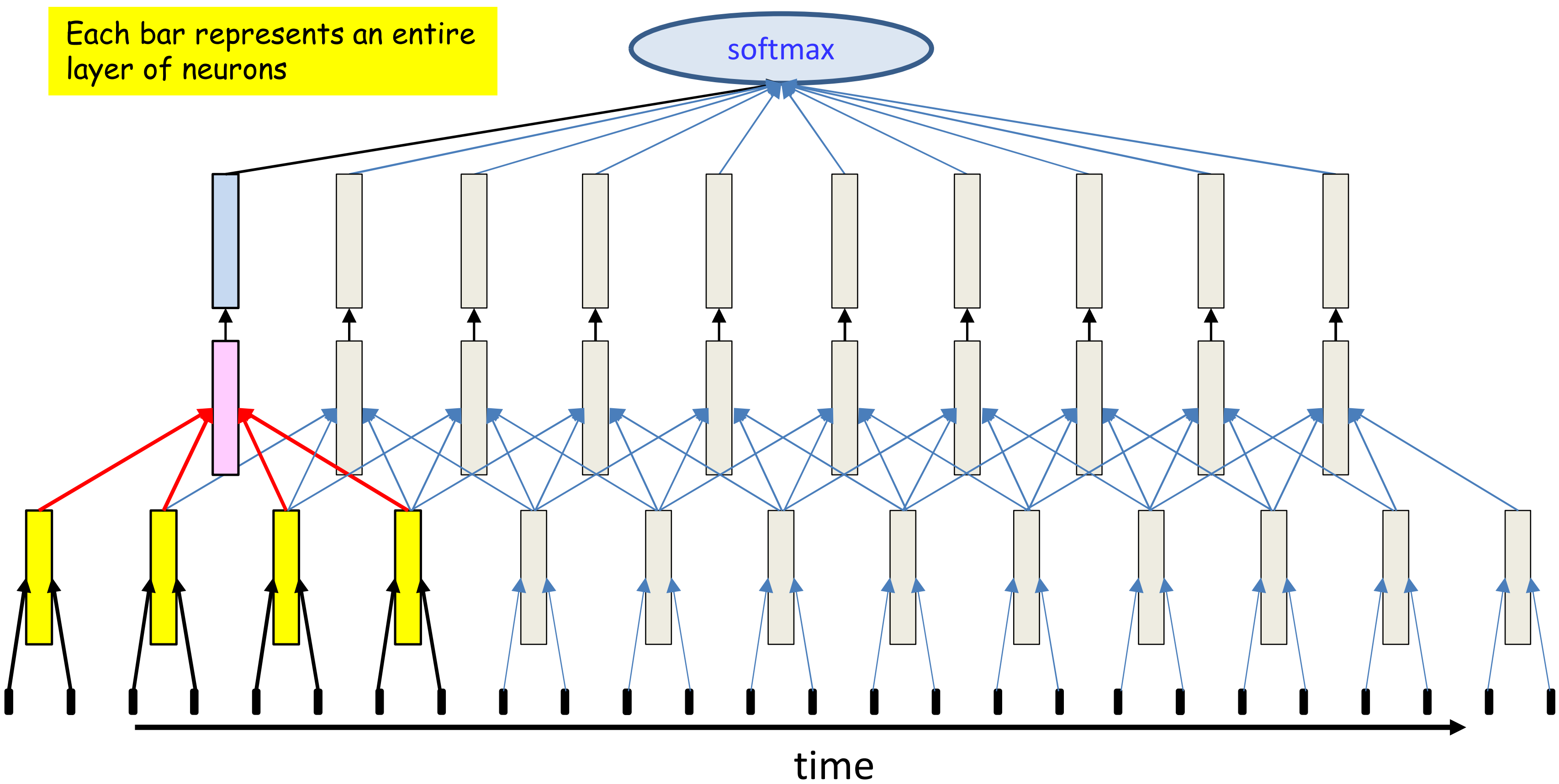
- ***Non-distributed*** scan of 8-time-step wide patterns with a stride of two time steps

Distributed scanning



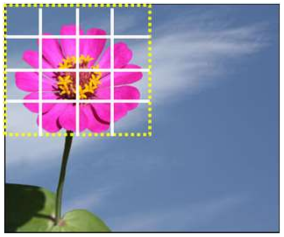
- Scan of 8-time-step wide patterns with a stride of two time steps ***distributed over two layers***

Distributed scanning



- Scan of 8-time-step wide patterns with a stride of two time steps ***distributed over two layers***
  - **At each position higher level neurons *reuse* some of the computations performed at the previous step(s)!**

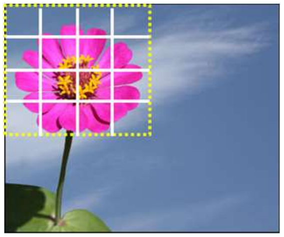
Scanning with an MLP (2D) (without distribution)



- `KxK` = size of "window" evaluated by MLP
- `W` is width of image
- `H` is height of image

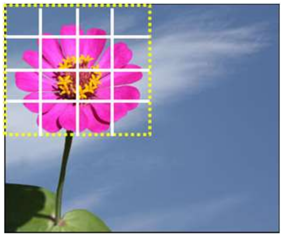
```
for x = 1:W-K+1
  for y = 1:H-K+1
    ImgSegment = Img(*, x:x+W-1, y:y+W-1)
    Y(x,y) = MLP(ImgSegment)

Y = softmax( Y(1,1)..Y(W-K+1,H-K+1) )
```

Scanning with an MLP (2D) (without distribution)



```
for x = 1:W-K+1
    for y = 1:H-K+1
       for l = 1:L  # layers
           for j = 1:Dl
               Compute z(l,j,x,y) [not expanded]
               Y(l,j,x,y) = activation(z(l,j,x,y))

Y = softmax( Y(L,:,1,1)..Y(L,:,W-K+1,H-K+1) )
```

Reordering the computation (without distribution)



```
for l = 1:L  # layers
    for x = 1:W-K+1
        for y = 1:H-K+1
            for j = 1:D_l
                for (l==1)
                    Segment = Y(0,1:C,x:x+K-1,y:y+K-1)
                else
                    Segment = Y(l-1,1:D_l-1,x,y)
                Compute z(l,j,x,y) [from Segment]
                Y(l,j,x,y) = activation(z(l,j,x,y))

Y = softmax( Y(L,:,1,1)..Y(L,:,W-K+1,H-K+1) )
```

Reordered scanning with distribution

Each layer now scans the output maps from the previous layer in windows of $K_l x K_l$

```
for l = 1:L  # layers
    for x = 1:W_{l-1}-K_l+1
        for y = 1:H_{l-1}-K_l+1
            for j = 1:D_l
                Segment = Y(l-1,1:D_l,x:x+K_l-1,y:y+K_l-1)
                Compute z(l,j,x,y) from Segment
                Y(l,j,x,y) = activation(z(l,j,x,y))

Y = softmax( Y(L,:,1,1)..Y(L,:,W-K+1,H-K+1) )
```

Reordered scanning with distribution

```
Y(0,:,:,:) = Image
for l = 1:L  # layers operate on vector at (x,y)
  for x = 1:W_{l-1}-K_l+1
    for y = 1:H_{l-1}-K_l+1
      for j = 1:D_l
        z(l,j,x,y) = b(l,j)
        for i = 1:D_{l-1}
          for x′ = 1:K_l
            for y′ = 1:K_l
              z(l,j,x,y) += w(l,i,j,x′,y′)
                     Y(l-1,i,x+x′-1,y+y′-1)
        Y(l,j,x,y) = activation(z(l,j,x,y))

Y = softmax( Y(L,:,1,1)..Y(L,:,W-K+1,H-K+1) )
```

Reordered scanning with distribution

```
Y(0,:,:,:) = Image
for l = 1:L  # layers operate on vector at (x,y)
    for x = 1:W_{l-1}-K_l+1
        for y = 1:H_{l-1}-K_l+1
            for j = 1:D_l
                z(l,j,x,y) = b(l,j)
                for i = 1:D_{l-1}
                    for x′ = 1:K_l
                        for y′ = 1:K_l
                            z(l,j,x,y) += w(l,i,j,x′,y′)
                                   Y(l-1,i,x+x′-1,y+y′-1)
                Y(l,j,x,y) = activation(z(l,j,x,y))

Y = softmax( Y(L,:,1,1)..Y(L,:,W-K+1,H-K+1) )
```

This operation is a "convolution"

“Convolutional Neural Network” (aka scanning with an MLP)

```
Y(0,:,:,:) = Image
for l = 1:L  # layers operate on vector at (x,y)
    for x = 1:W_{l-1}-K_l+1
        for y = 1:H_{l-1}-K_l+1
            for j = 1:D_l
                z(l,j,x,y) = 0
                for i = 1:D_{l-1}
                    for x' = 1:K_l
                        for y' = 1:K_l
                            z(l,j,x,y) += w(l,i,j,x',y')
                                          Y(l-1,i,x+x'-1,y+y'-1)
                Y(l,j,x,y) = activation(z(l,j,x,y))

Y = softmax( Y(L,:,1,1)..Y(L,:,W-K+1,H-K+1) )
```

Convolutional neural net: Vector notation

The weight W(l,j)**is now a 3D D_{l-1}xK_lxK_l tensor (assuming square receptive fields)**

The product in blue is a tensor inner product with a scalar output

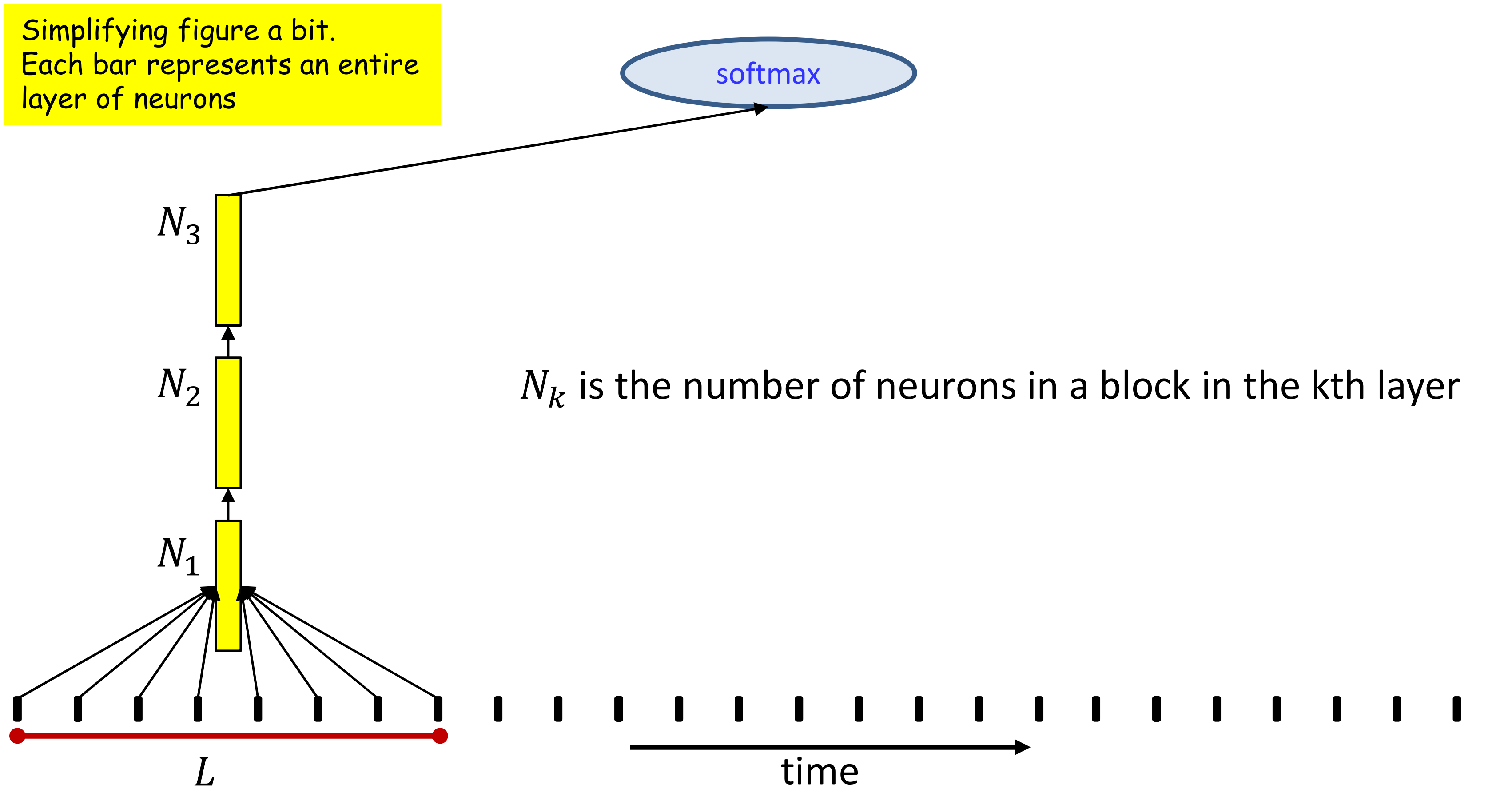
```
Y(0) = Image
for l = 1:L  # layers operate on vector at (x,y)
  for x = 1:W_{l-1}-K_l+1
    for y = 1:H_{l-1}-K_l+1
      for j = 1:D_l
        segment = Y(l-1,:,x:x+K_l-1,y:y+K_l-1) #3D tensor
        z(l,j,x,y) = W(l,j).segment + b(l,j)
                     #tensor inner prod.
        Y(l,j,x,y) = activation(z(l,j,x,y))

Y = softmax( Y(L) )
```

Why distribute?

- Distribution forces *localized* patterns in lower layers
  - More generalizable
- Fewer computations
  - Reusable computations from lower layers

- Number of parameters…

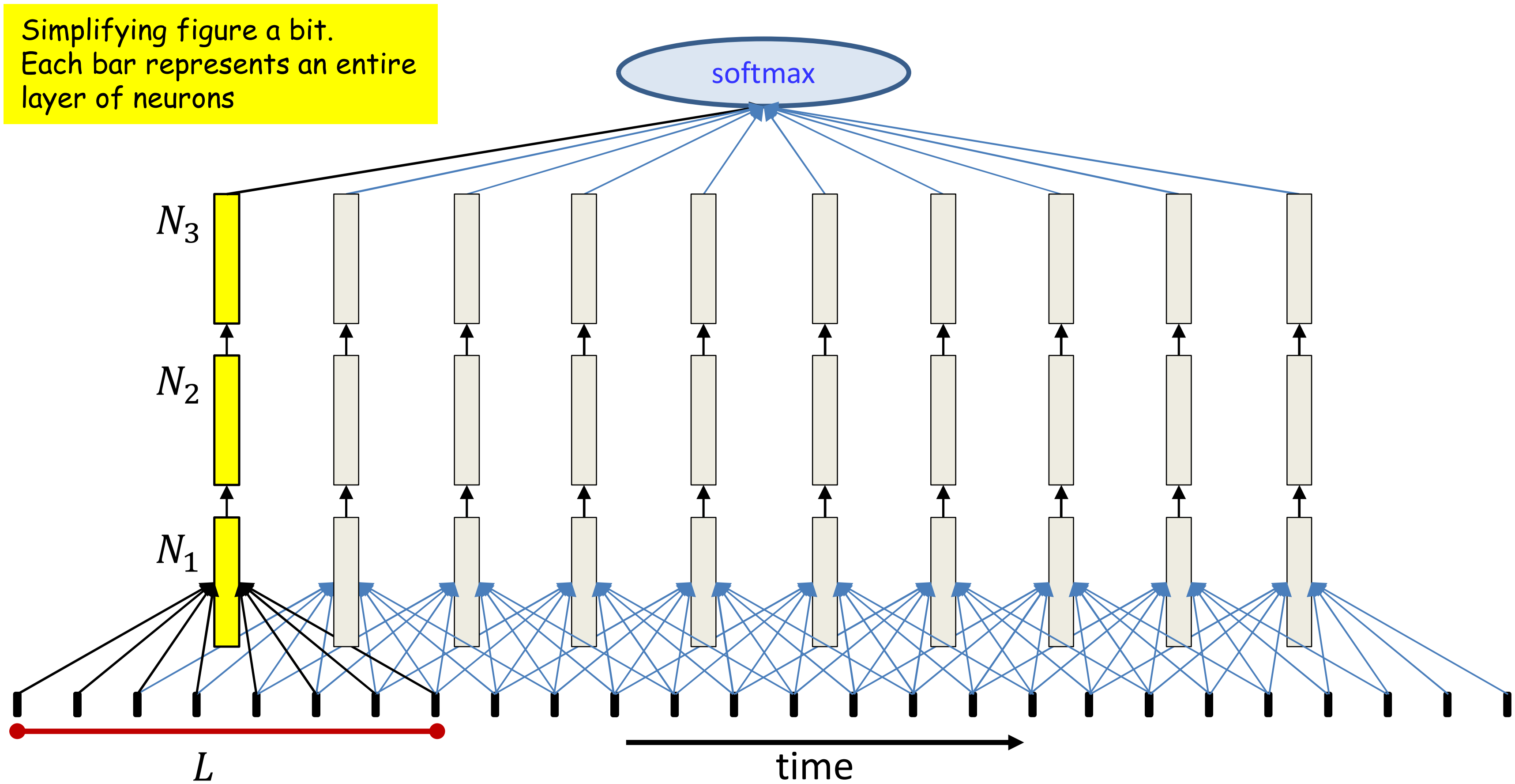
Scanning without distribution



N_k is the number of neurons in a block in the kth layer

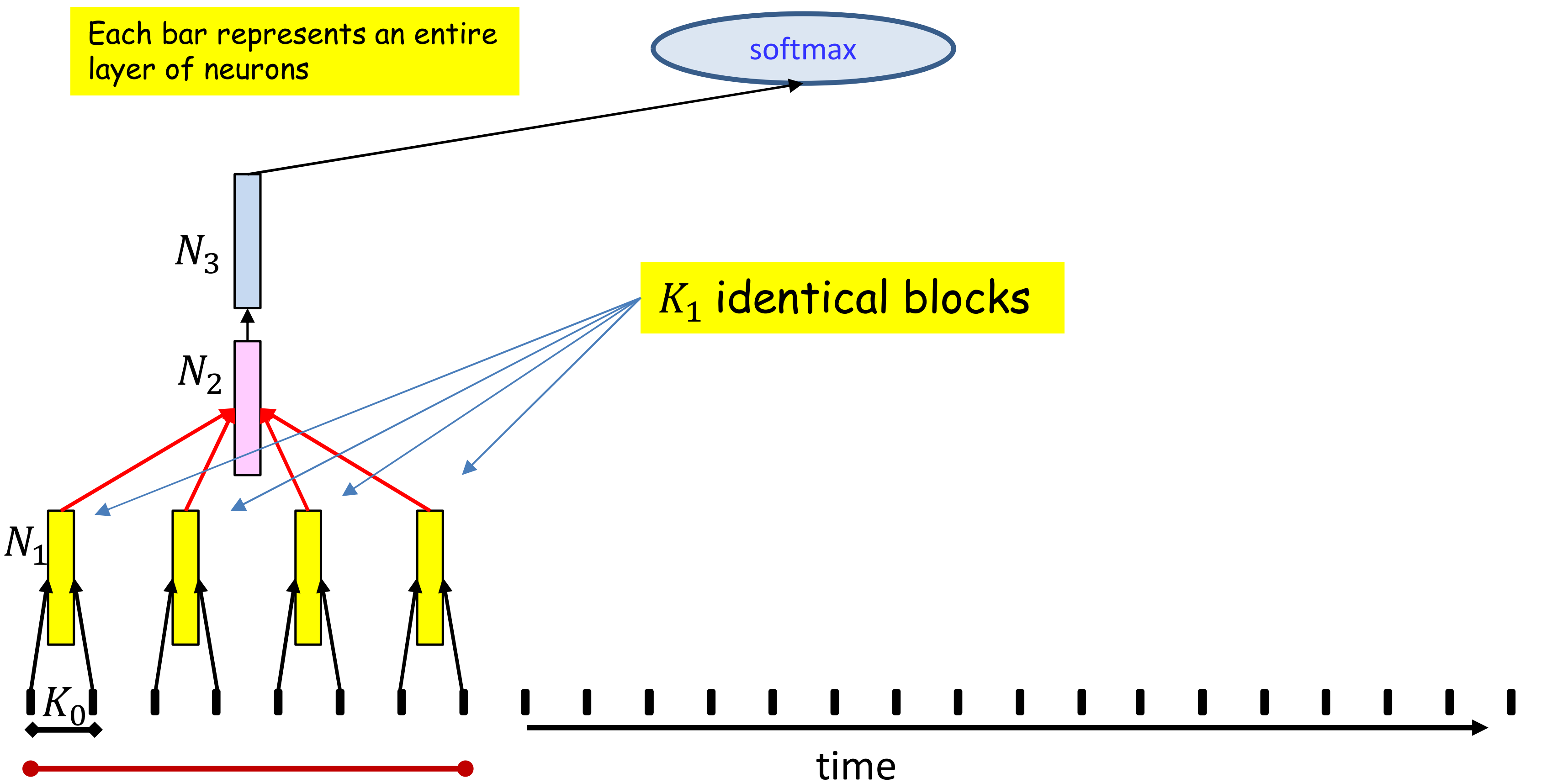
- Total parameters: $8DN_1 + N_1N_2 + N_2N_3$
  - D is dimensionality of input
  - More generally: $LDN_1 + N_1N_2 + N_2N_3$
  - Ignoring bias terms in computation
- Only need to count parameters for *one* column, since other columns are identical

Scanning without distribution



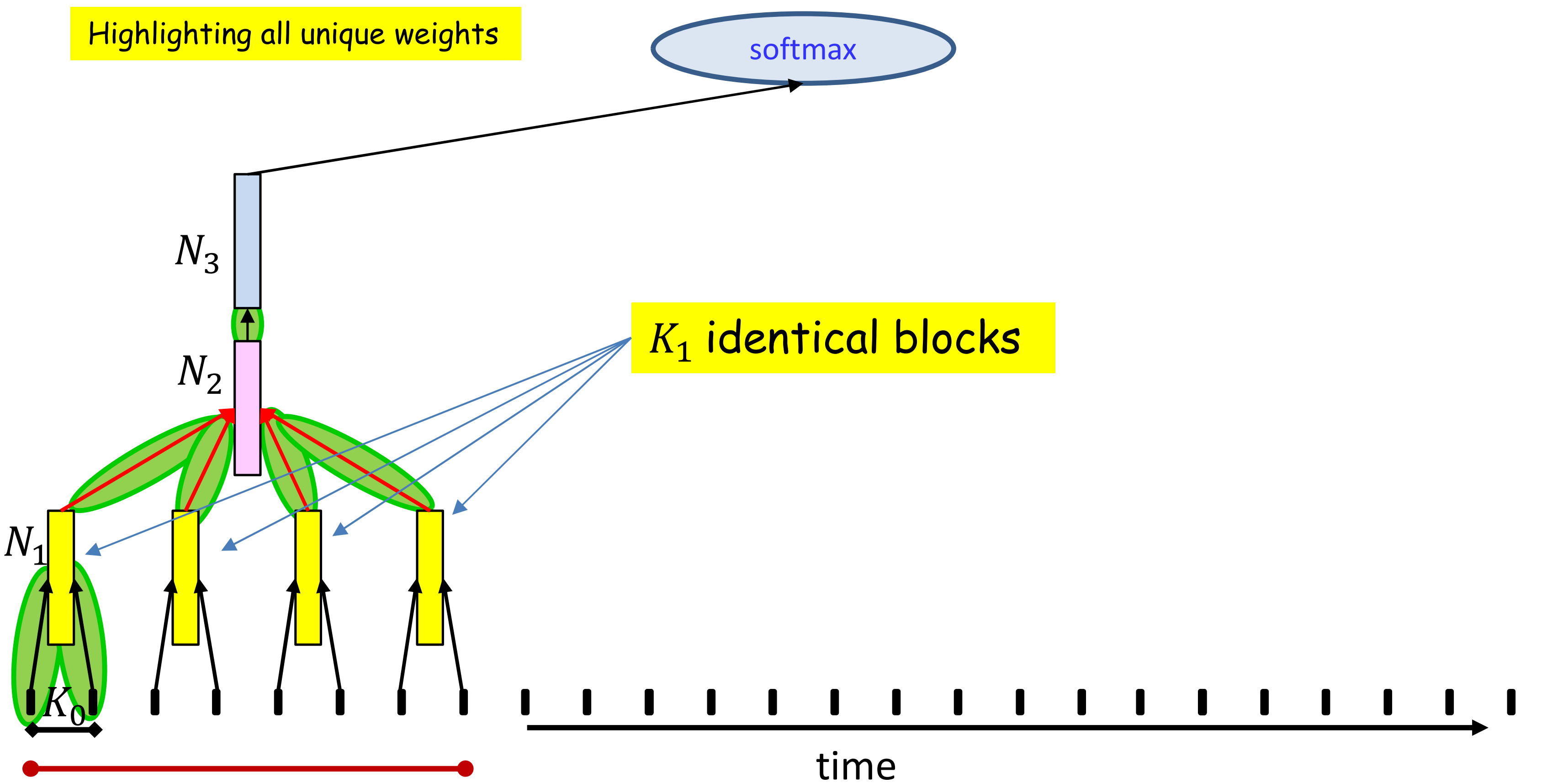
- Total parameters: $8DN_1 + N_1N_2 + N_2N_3$
  - D is dimensionality of input
  - More generally: $LDN_1 + N_1N_2 + N_2N_3$
  - Ignoring bias terms in computation
- Only need to count parameters for *one* column, since other columns are identical

Distributed scanning



- Total parameters: $2DN_1 + 4N_1N_2 + N_2N_3$
  - More generally: $K_0DN_1 + K_1N_1N_2 + N_2N_3$
  - **Fewer parameters than a non-distributed net with identical number of neurons**

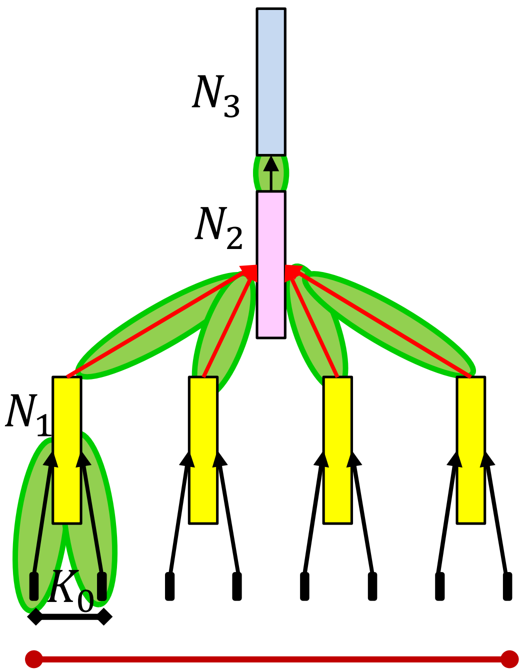
Distributed scanning



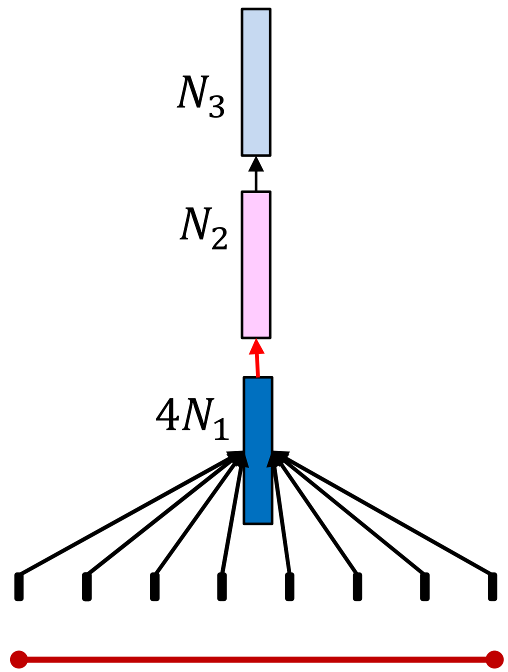
- Total parameters: $2DN_1 + 4N_1N_2 + N_2N_3$
  - More generally: $K_0DN_1 + K_1N_1N_2 + N_2N_3$
  - **Fewer parameters than a non-distributed net with identical number of neurons**

Distributed vs non-distributed scanning

Highlighting all unique weights for distributed scan

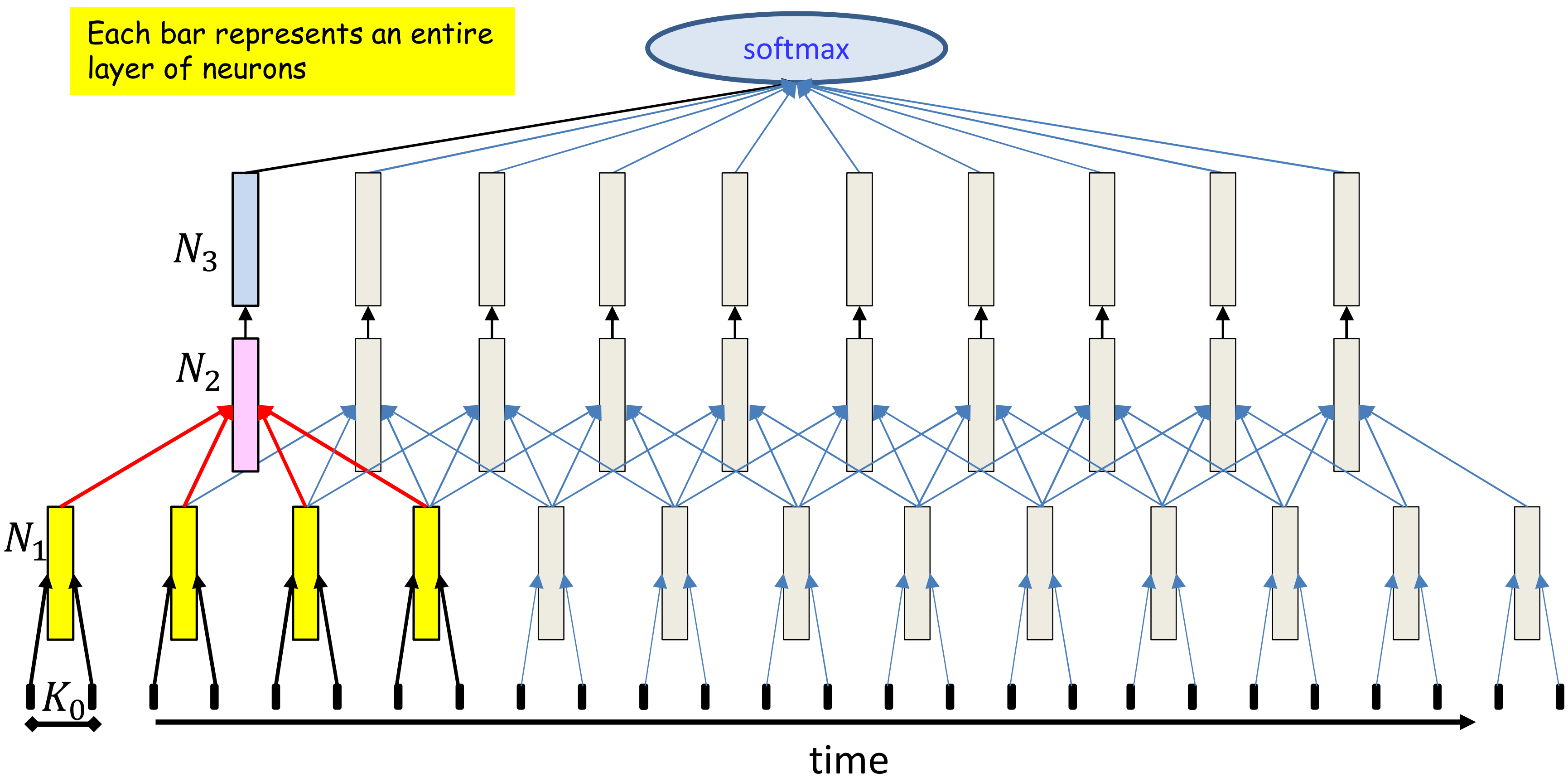


Equivalent non-distributed net has
$8D(4N_1) + 4N_1N_2 + N_2N_3$
parameters (not including bias terms)



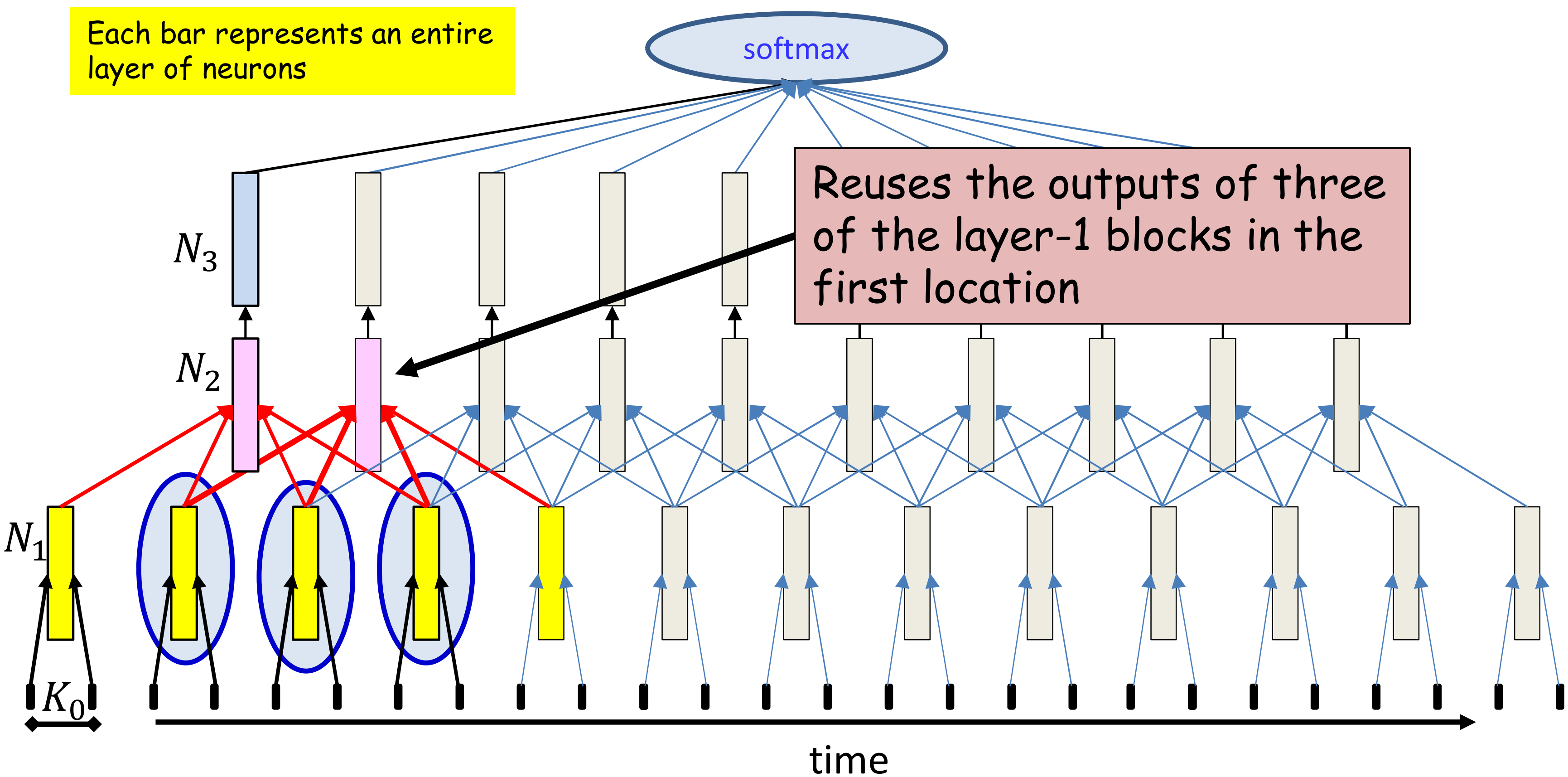
- Total parameters: $2DN_1 + 4N_1N_2 + N_2N_3$
  - More generally: $K_0DN_1 + K_1N_1N_2 + N_2N_3$
  - **Fewer parameters than a non-distributed net with identical number of neurons**

Distributed scanning



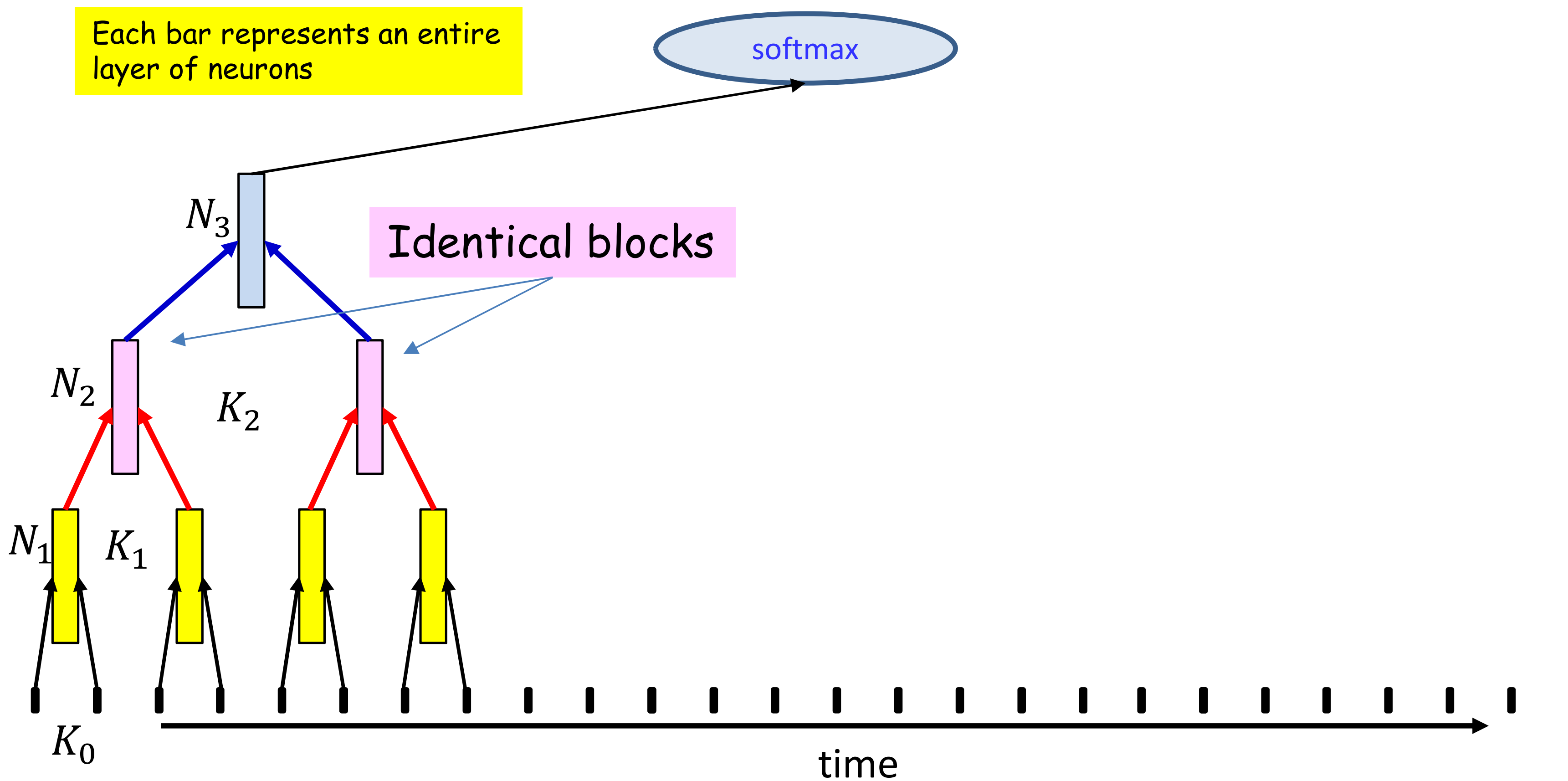
- Total parameters: $2DN_1 + 4N_1N_2 + N_2N_3$
  - More generally: $K_0DN_1 + K_1N_1N_2 + N_2N_3$
  - **Fewer parameters than a non-distributed net with identical number of neurons**

Distributed scanning



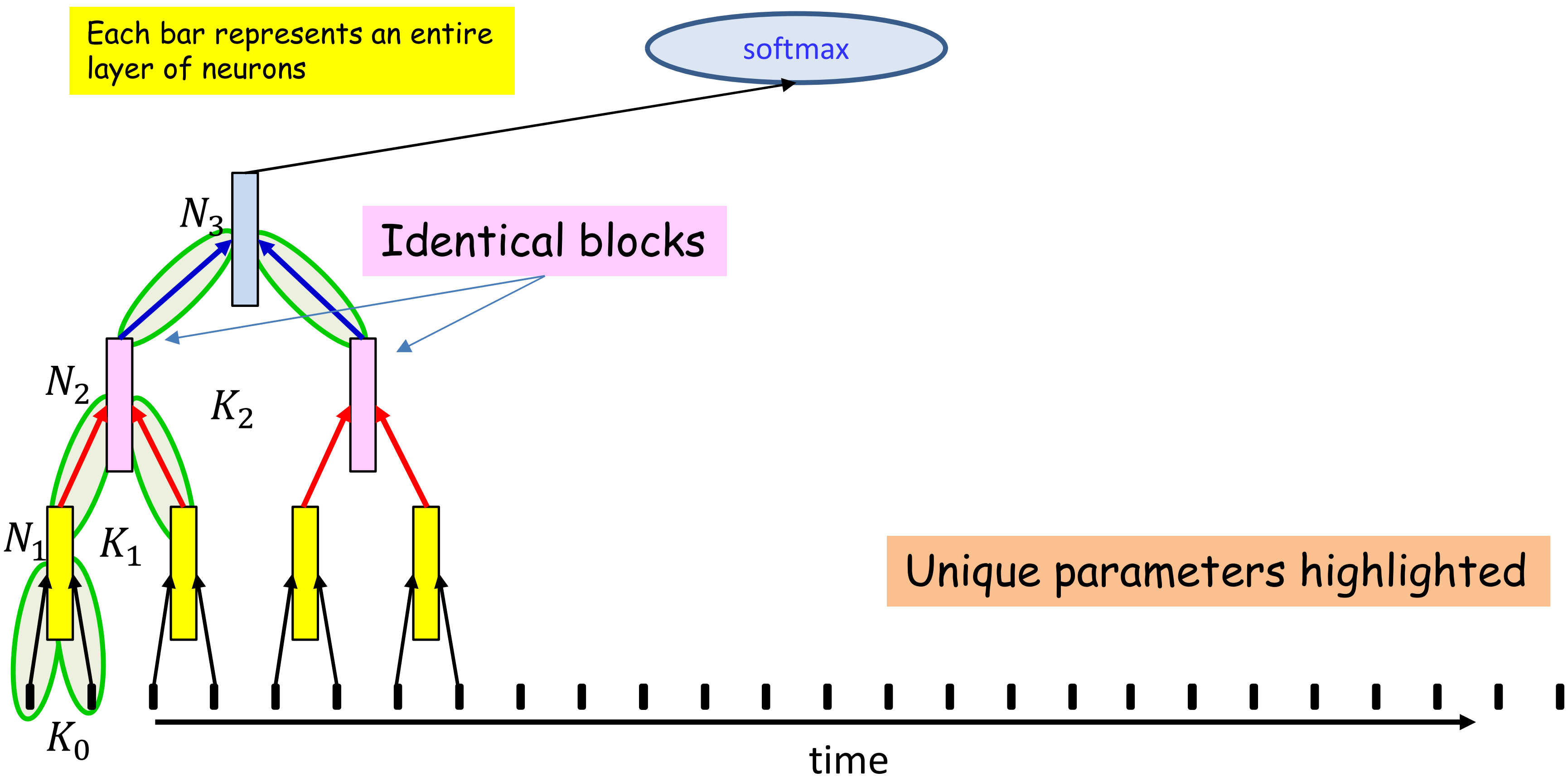
- Total parameters: $2DN_1 + 4N_1N_2 + N_2N_3$
  - More generally: $K_0DN_1 + K_1N_1N_2 + N_2N_3$
  - **Fewer parameters than a non-distributed net with identical number of neurons**
  - ***Large additional benefit from the fact that scans at neighboring positions share the computation of lower-level blocks!***

Distributed scanning: 3 levels



- Total parameters: $2DN_1 + 2N_1N_2 + 2N_2N_3$
  - More generally: $K_0DN_1 + K_1N_1N_2 + K_2N_2N_3$
  - **Far fewer parameters than non-distributed scan with network with identical no. of neurons**

Distributed scanning: 3 levels

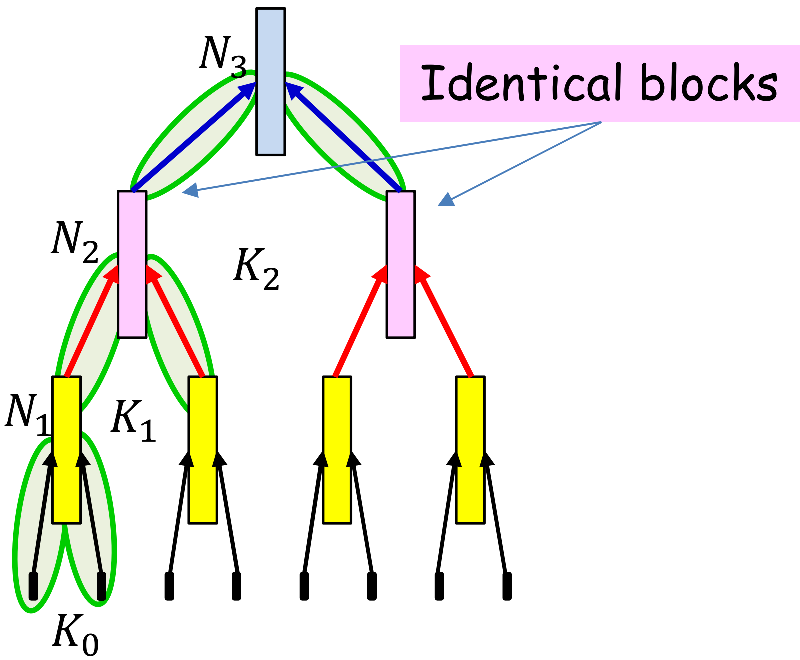


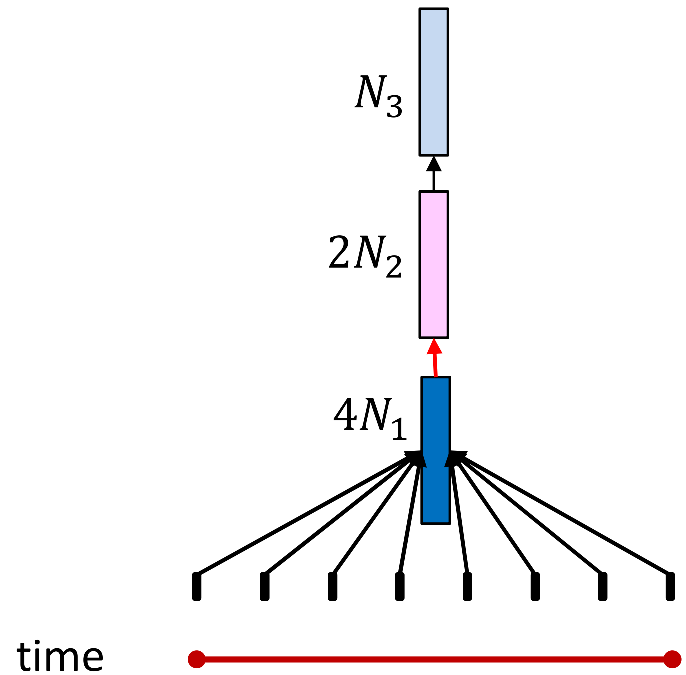
- Total parameters: $2DN_1 + 2N_1N_2 + 2N_2N_3$
  - More generally: $K_0DN_1 + K_1N_1N_2 + K_2N_2N_3$
  - **Far fewer parameters than non-distributed scan with network with identical no. of neurons**

Distributed scanning: 3 levels

Each bar represents an entire layer of neurons

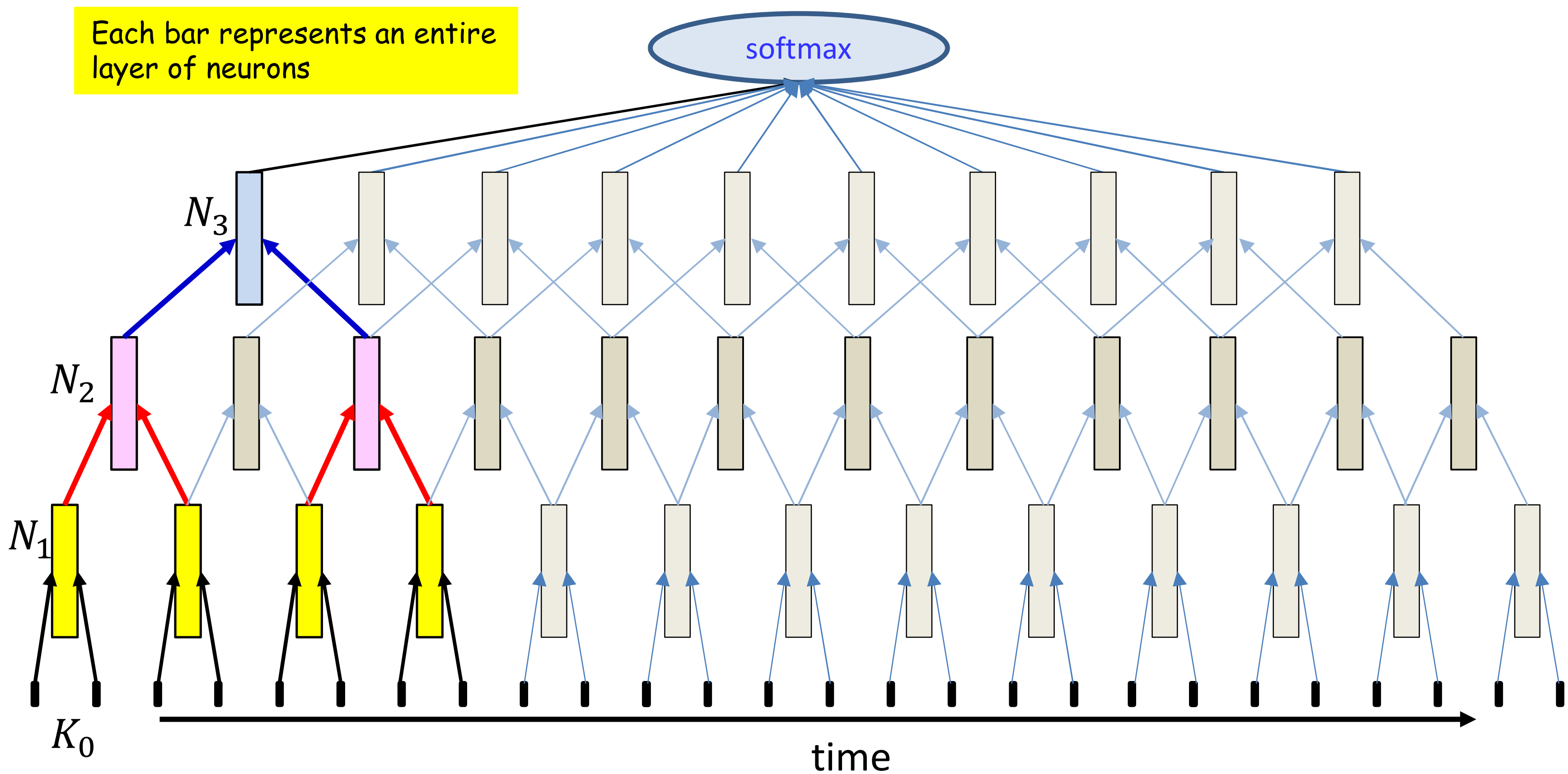
Equivalent non-distributed net has
$8D(4N_1) + (4N_1)(2N_2) + (2N_2)N_3$
neurons





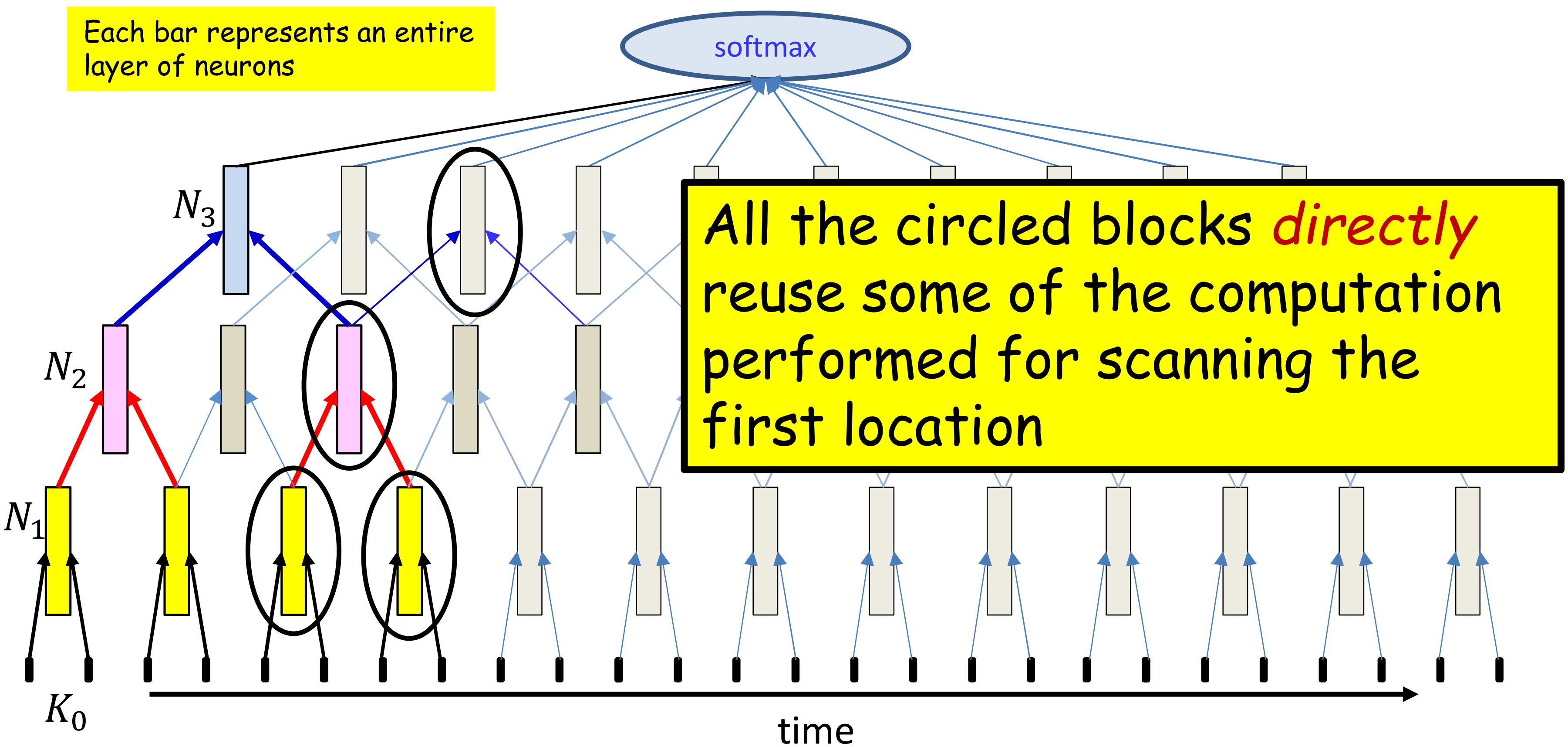
- Total parameters: $2DN_1 + 2N_1N_2 + 2N_2N_3$
  - More generally: $K_0DN_1 + K_1N_1N_2 + K_2N_2N_3$
  - **Far fewer parameters than non-distributed scan with network with identical no. of neurons**

Distributed scanning: 3 levels



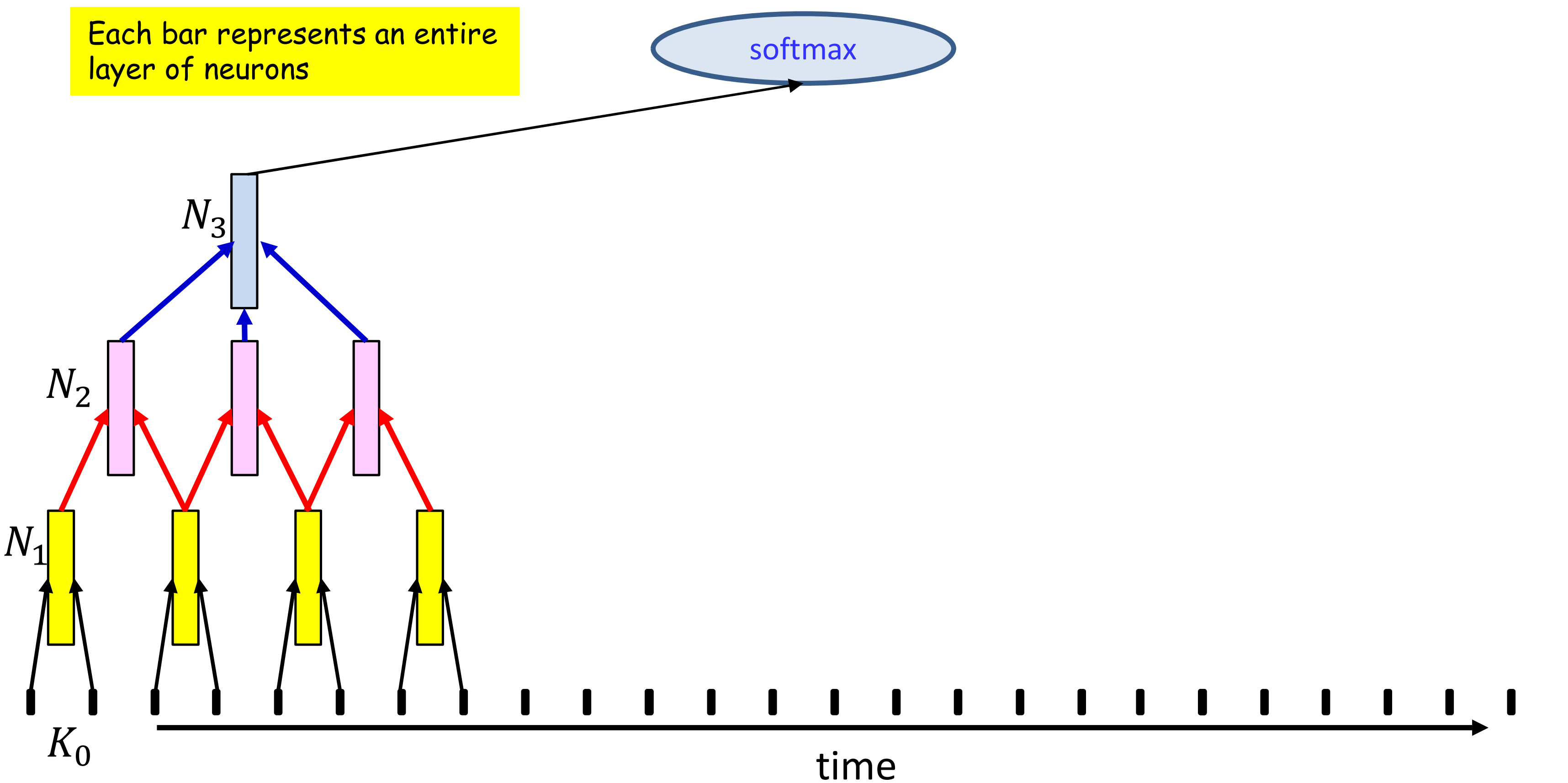
- Total parameters: $2DN_1 + 2N_1N_2 + 2N_2N_3$
  - More generally: $K_0DN_1 + K_1N_1N_2 + K_2N_2N_3$
  - **Far fewer parameters than non-distributed scan with network with identical no. of neurons**

Distributed scanning: 3 levels



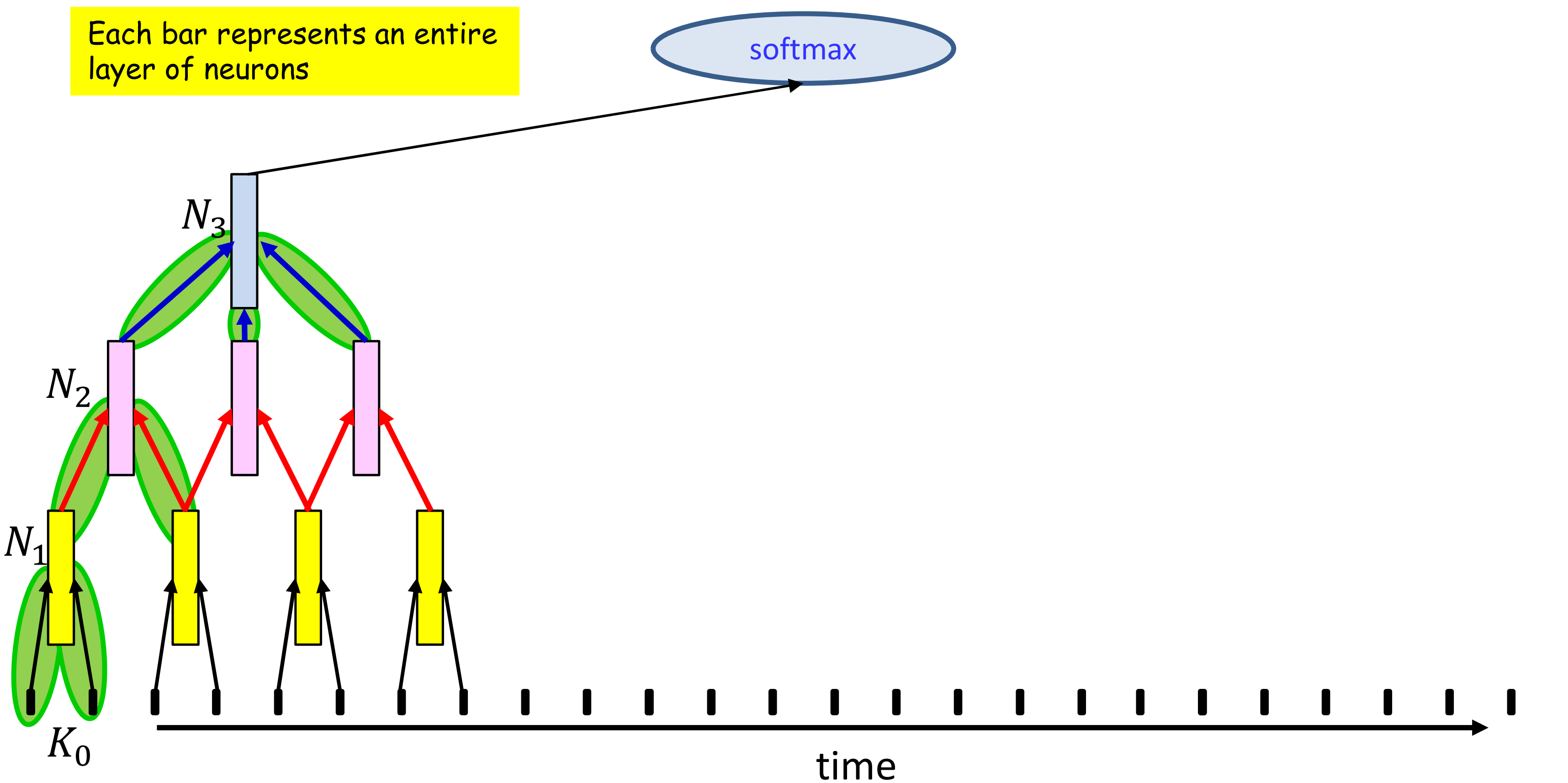
- Total parameters: $2DN_1 + 2N_1N_2 + 2N_2N_3$
  - More generally: $K_0DN_1 + K_1N_1N_2 + K_2N_2N_3$
  - **Far fewer parameters than non-distributed scan by network with identical no. of neurons**
  - **Large additional gains from reuse of computation!!**

Distributed scanning



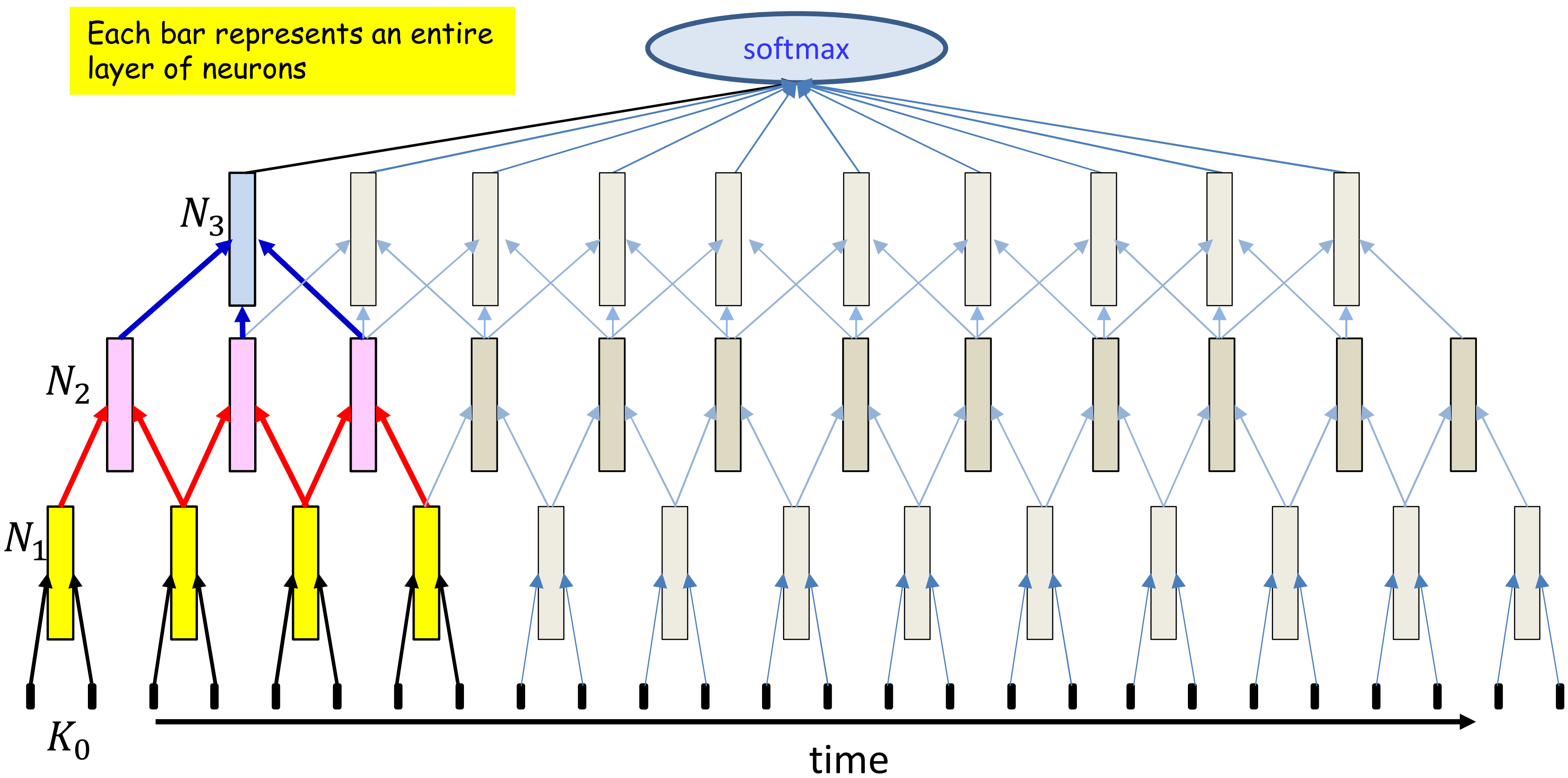
- Total parameters: $2DN_1 + 2N_1N_2 + 3N_2N_3$
  - More generally: $K_0DN_1 + K_1N_1N_2 + K_2N_2N_3$
  - **Will have fewer parameters than a non-distributed structure with identical numbers of neurons**

Distributed scanning



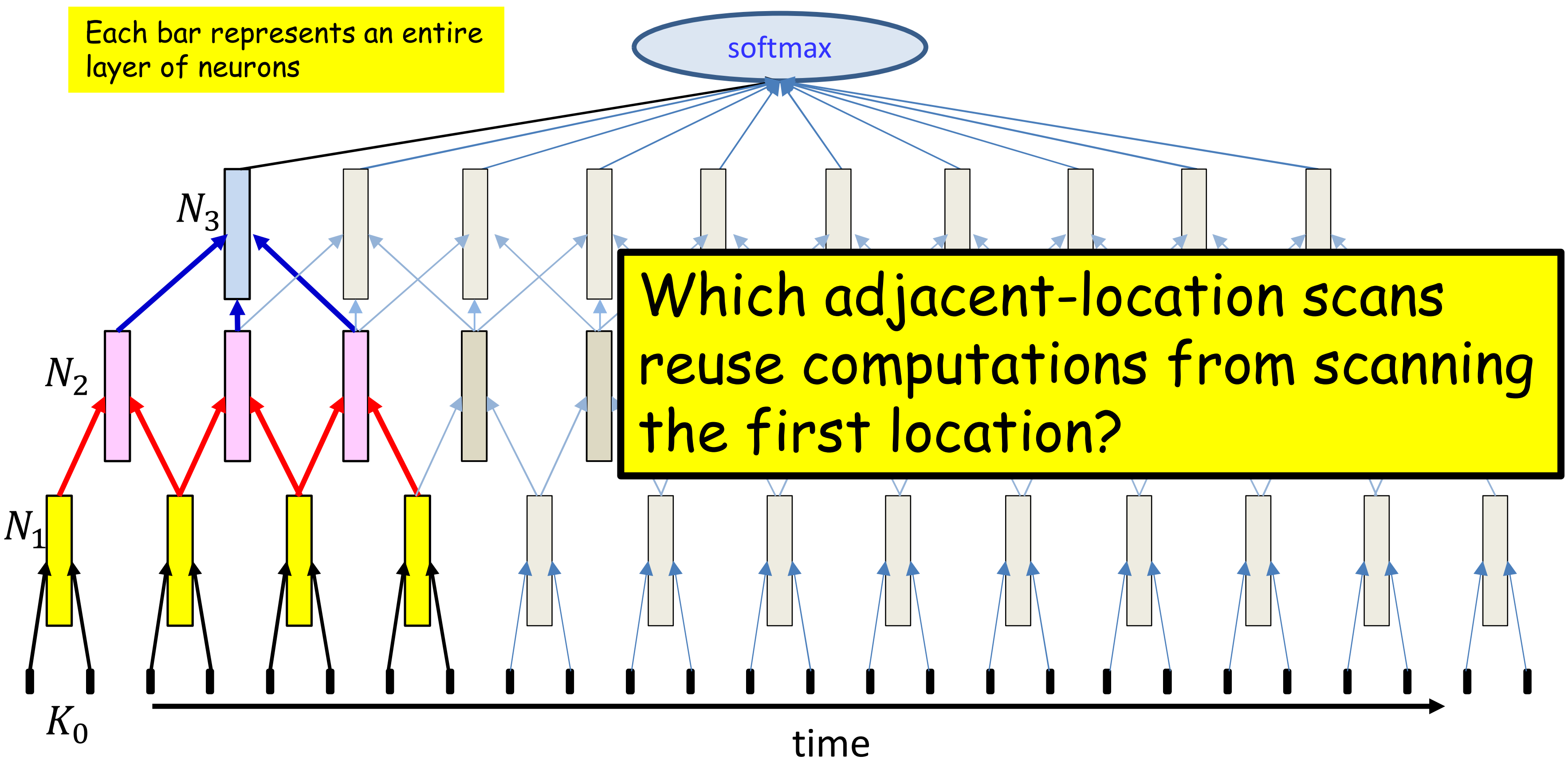
- Total parameters: $2DN_1 + 2N_1N_2 + 3N_2N_3$
  - More generally: $K_0DN_1 + K_1N_1N_2 + K_2N_2N_3$
  - **Will have fewer parameters than a non-distributed structure with identical numbers of neurons**

Distributed scanning



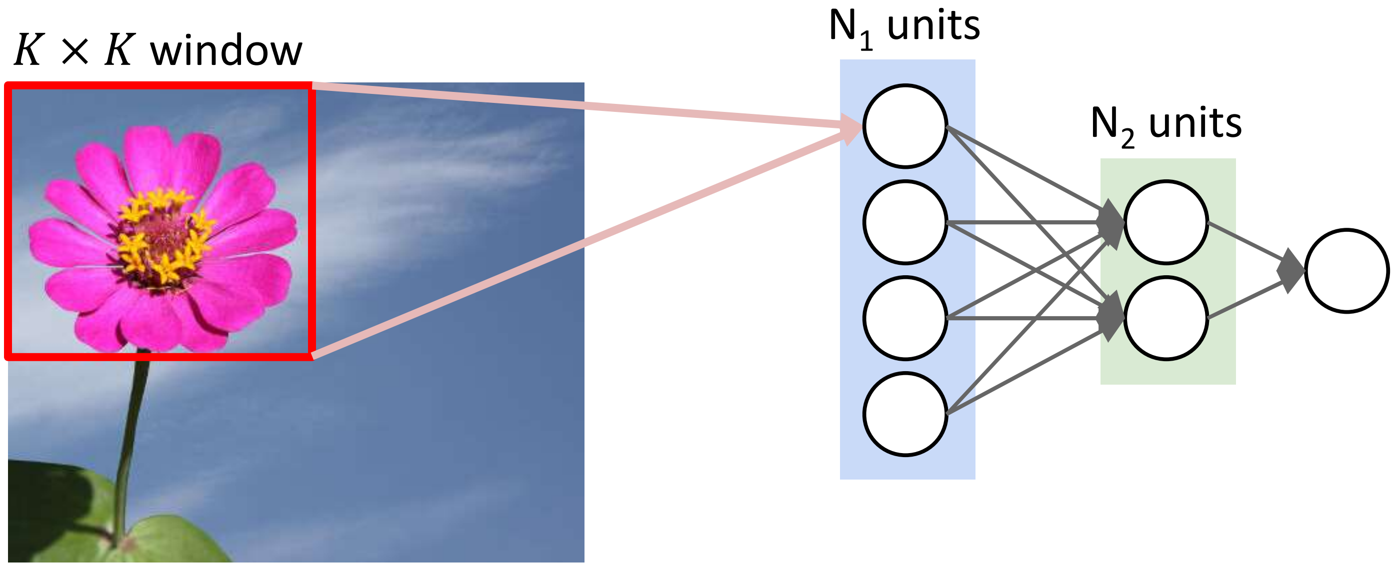
- Total parameters: $2DN_1 + 2N_1N_2 + 3N_2N_3$
  - More generally: $K_0DN_1 + K_1N_1N_2 + K_2N_2N_3$
  - **Will have fewer parameters than a non-distributed structure with identical numbers of neurons**

Distributed scanning



- Total parameters: $2DN_1 + 2N_1N_2 + 3N_2N_3$
  - More generally: $K_0DN_1 + K_1N_1N_2 + K_2N_2N_3$
  - **Will have fewer parameters than a non-distributed structure with identical numbers of neurons**
  - **Also benefits much more from shared computation in the scans of adjacent locations**

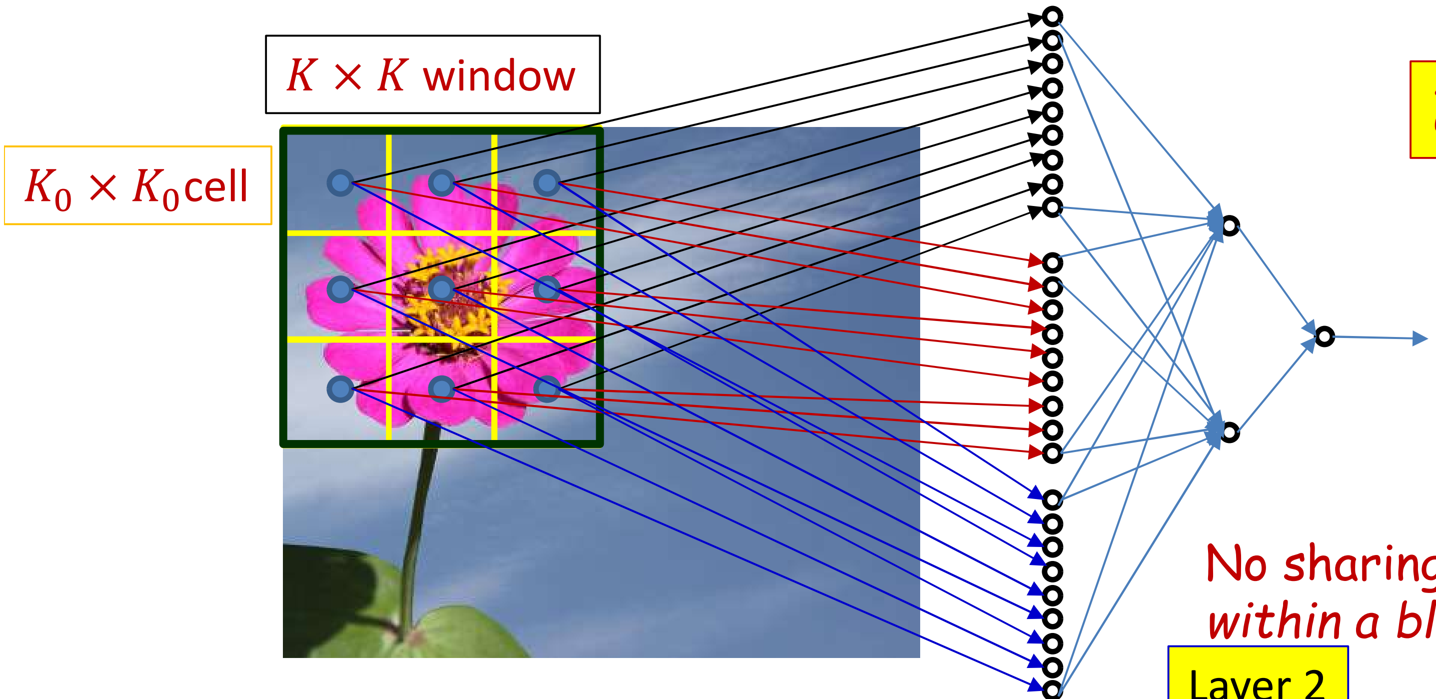
Images: *Undistributed* network



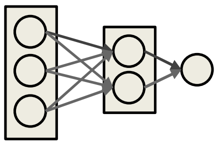
- Only need to consider what happens in *one* block
  - All other blocks are scanned by the same net
- $(K^2 + 1)N_1$ weights in first layer
- $(N_1 + 1)N_2$ weights in second layer
  - $(N_{i-1} + 1)N_i$ weights in subsequent i^{th} layer
- Total parameters: $\mathcal{O}(K^2N_1 + N_1N_2 + N_2N_3 \ldots)$
  - Ignoring the bias term

2-D version

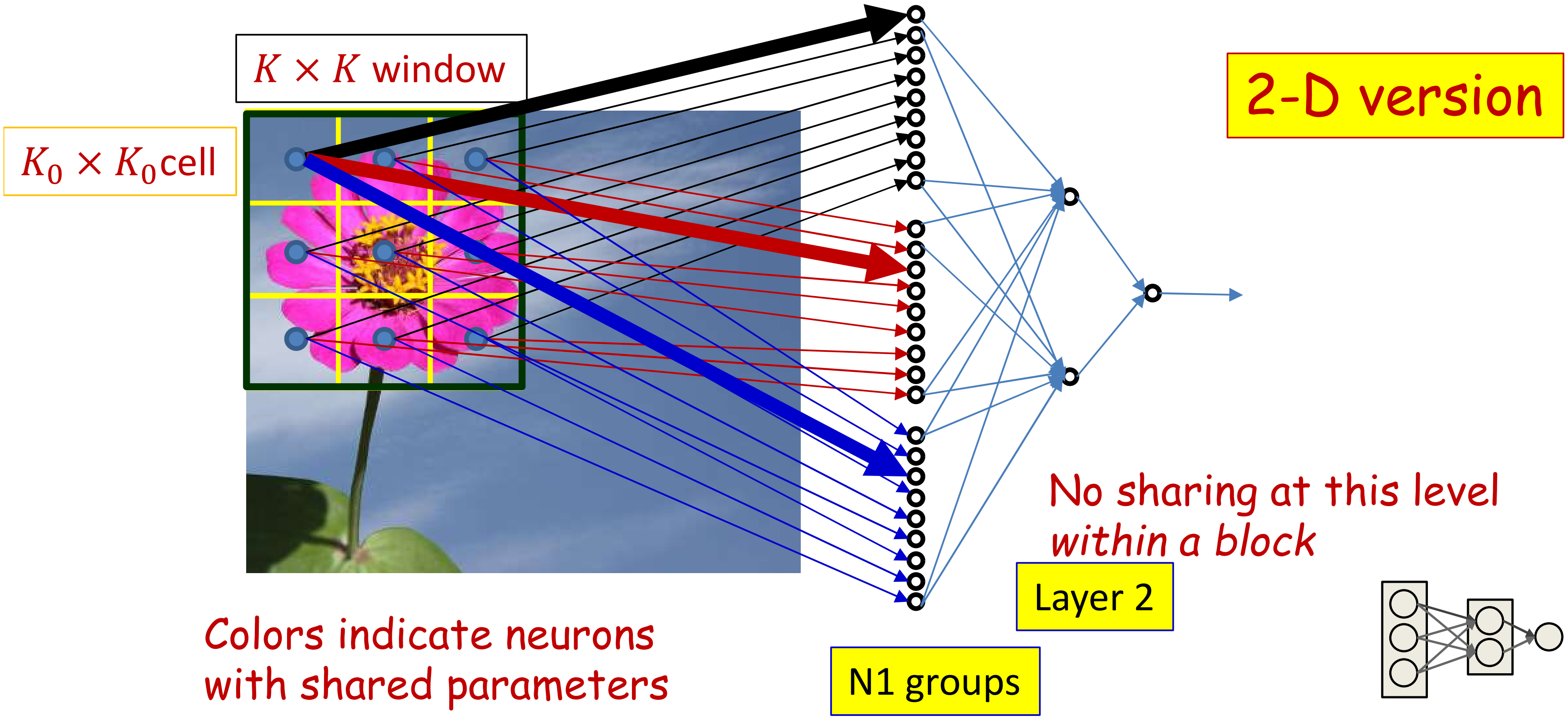
When distributed over 2 layers



Colors indicate neurons
with shared parameters

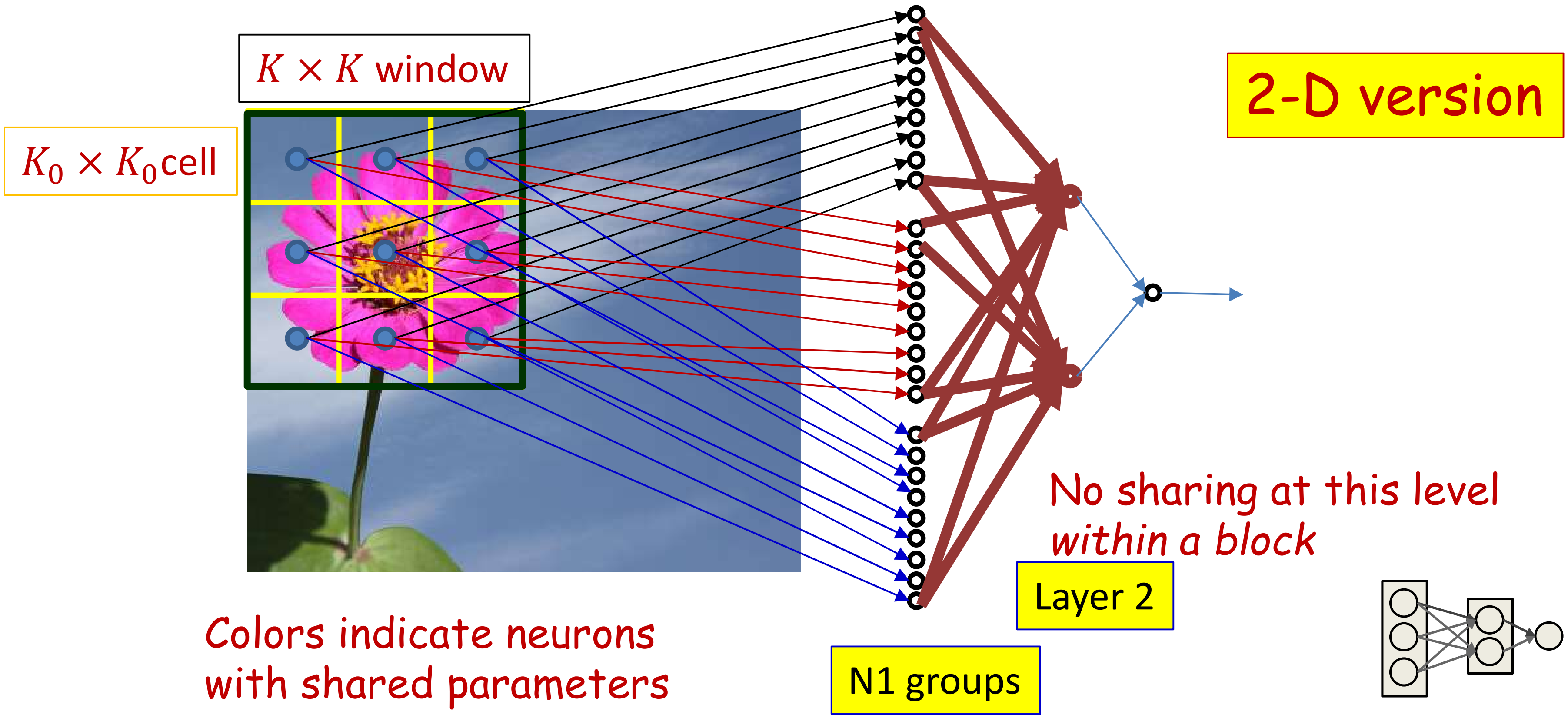


When distributed over 2 layers



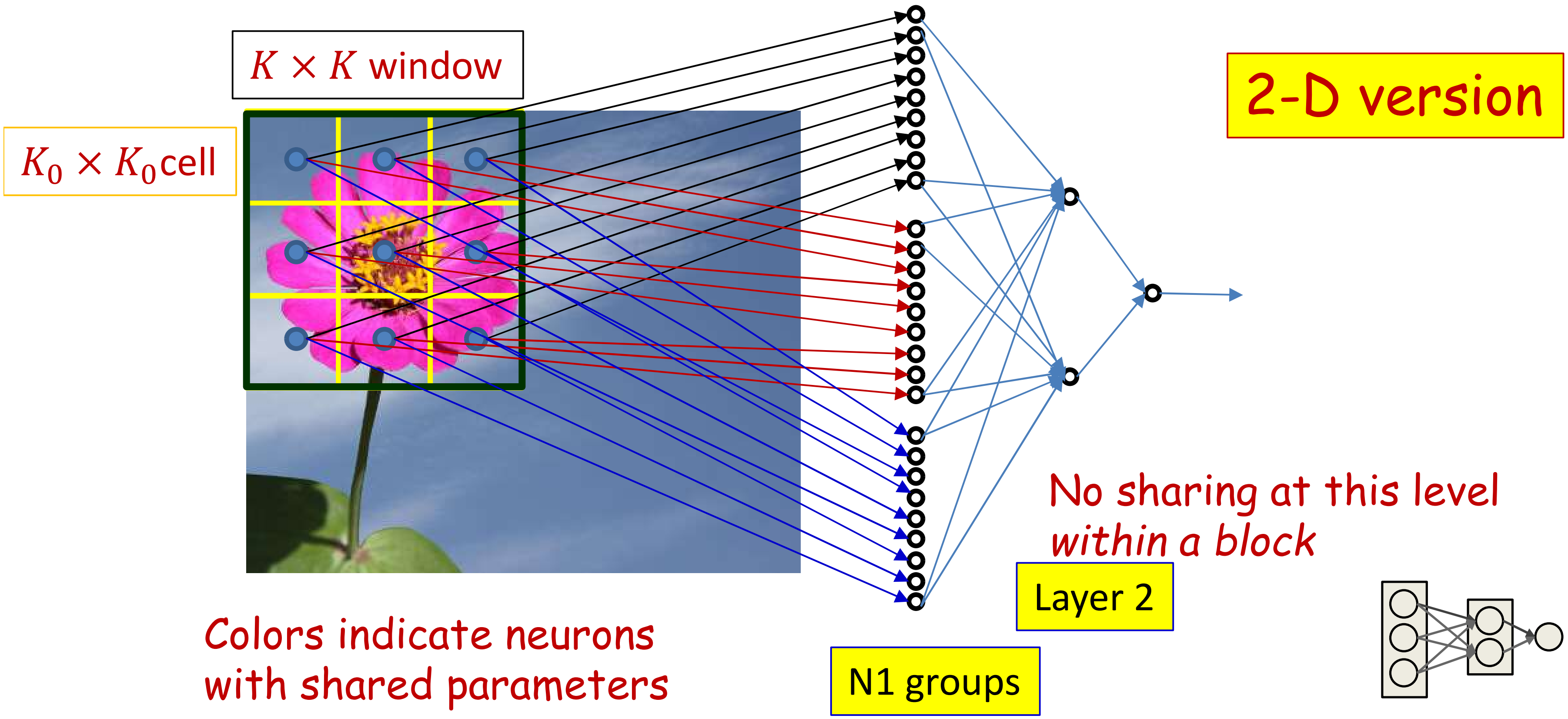
- First layer: N_1 lower-level units, each looks at K_0^2 pixels
  - $N_1(K_0^2 + 1)$ weights

When distributed over 2 layers



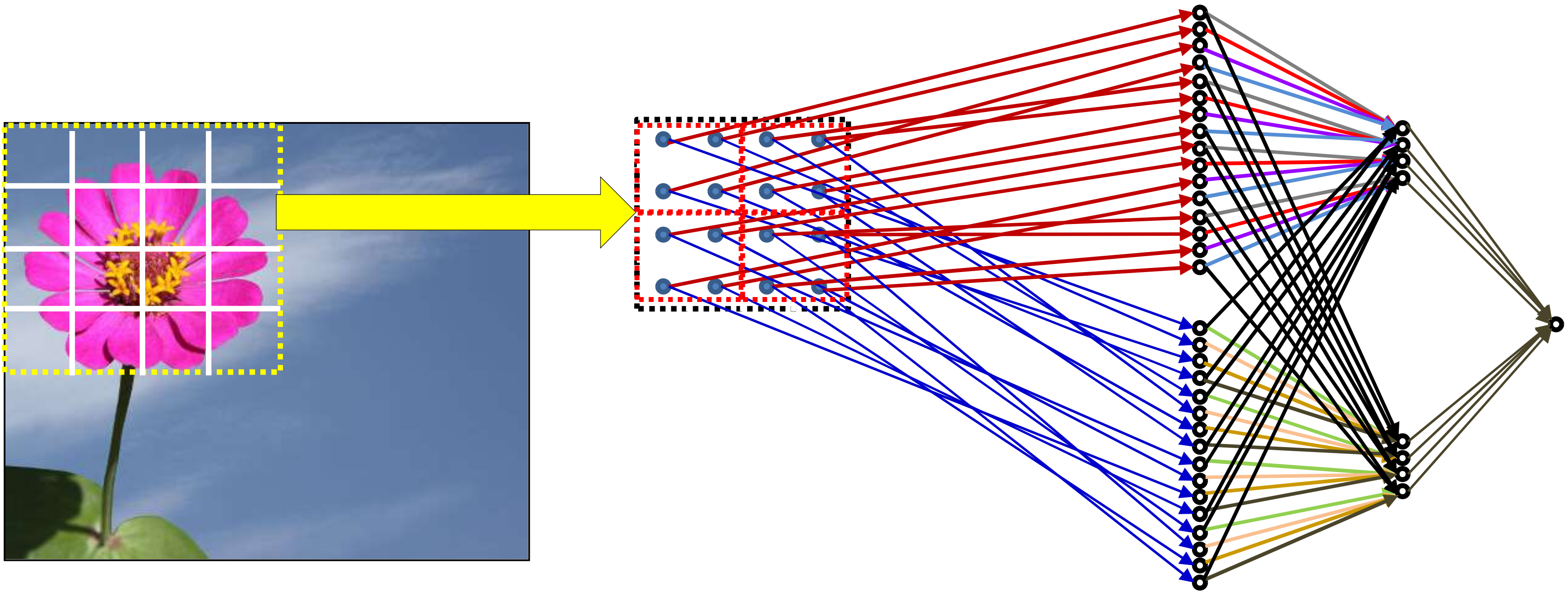
- First layer: N_1 lower-level units, each looks at K_0^2 pixels
  - $N_1(K_0^2 + 1)$ weights
- Second layer needs $(K_1^2 N_1 + 1)N_2$ weights

When distributed over 2 layers



- First layer: N_1 lower-level units, each looks at K_0^2 pixels
  - $N_1(K_0^2 + 1)$ weights
- Second layer needs $(K_1^2 N_1 + 1)N_2$ weights
- Subsequent layers needs $N_{i-1}N_i$ when distributed over 2 layers only
  - Total parameters: $\mathcal{O}(K_0^2 N_1 + K_1^2 N_1 N_2 + N_2 N_3 \ldots)$

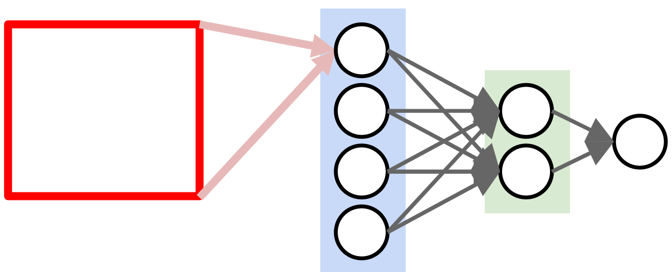
When distributed over 3 layers



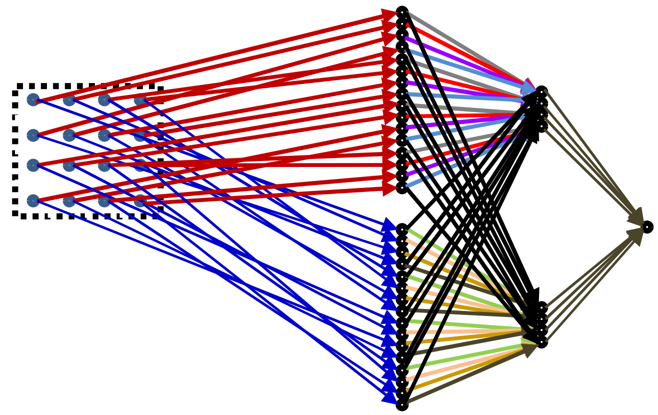
- First layer: N_1 lower-level (groups of) units, each looks at K_0^2 pixels
  - $N_1(K_0^2 + 1)$ weights
- Second layer: N_2 (groups of) units looking at groups of $K_1 \times K_1$ connections from each of N_1 first-level neurons
  - $(K_1^2 N_1 + 1)N_2$ weights
- Third layer:
  - $(K_2^2 N_2 + 1)N_3$ weights
- Subsequent layers need $N_{i-1}N_i$ neurons
  - Total parameters: $\mathcal{O}(K_0^2 N_1 + K_1^2 N_1 N_2 + K_2^2 N_2 N_3 + \cdots)$

Comparing Number of Parameters

Conventional MLP, not distributed



Distributed (3 layers)



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

- $\mathcal{O}(K_0^2 N_1 + K_1^2 N_1 N_2 + K_2^2 N_2 N_3 + \cdots)$
- Here, let $K = 16$, $K_0 = 4$, $K_1 = 4$, $N_1 = 4$, $N_2 = 2$, $N_3 = 1$
- Total 64+128+2 = 194 weights

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
  - Significant gains from shared computation
- **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**

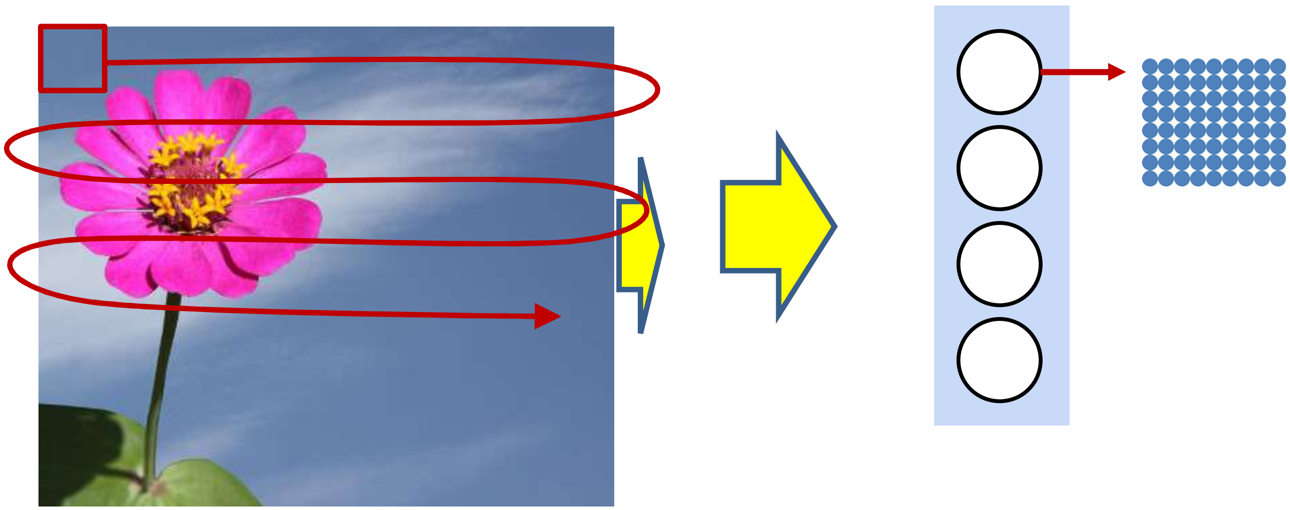
Story so far

- Position-invariant pattern classification can be performed by scanning the input for a target pattern
  - Scanning is equivalent to composing a large network with shared subnets
- The operations in scanning the input with a full network can be equivalently reordered as
  - scanning the input with individual neurons in the first layer to produce scanned "maps" of the input
  - Jointly scanning the "map" of outputs by all neurons in the previous layers by neurons in subsequent layers
- The scanning block can be distributed over multiple layers of the network
  - Results in significant reduction in the total number of parameters

Some final touches

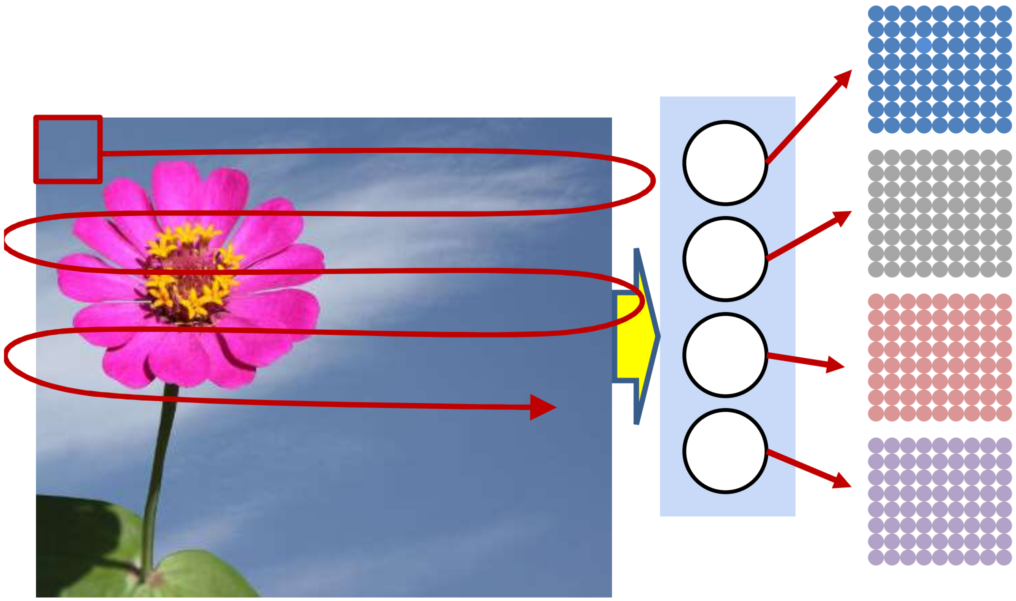
- Terminology
  - Filters and receptive fields
- Shrinking the maps
  - Scanning with strides
- Accounting for jitter
  - Pooling

Hierarchical composition: A different perspective



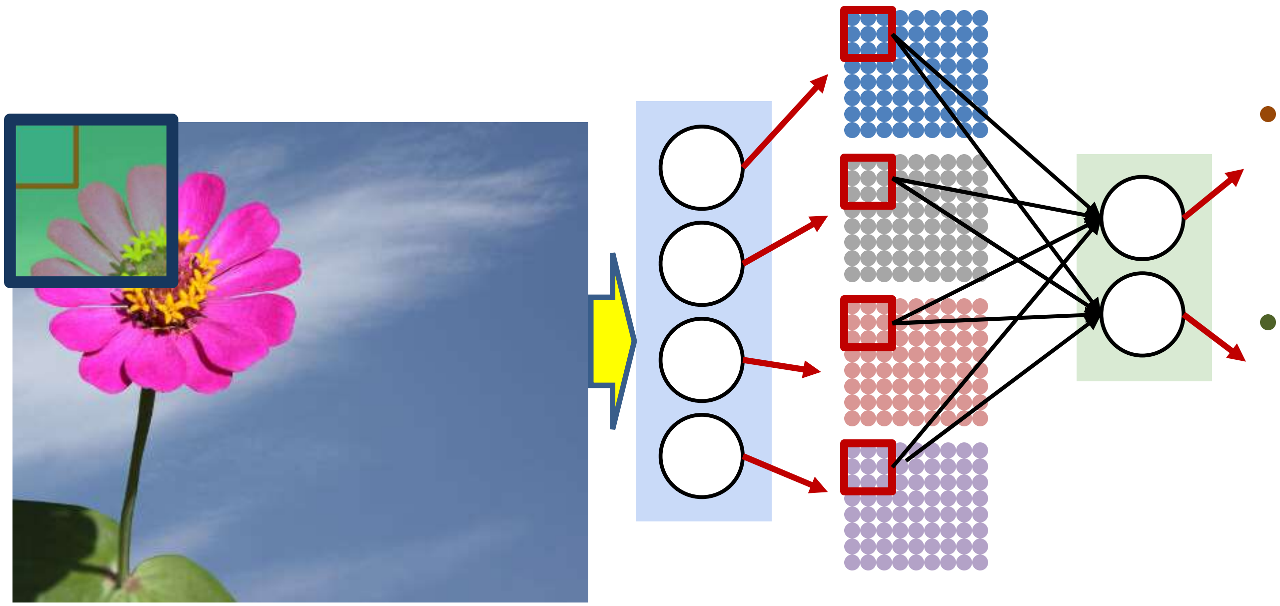
- The entire operation can be redrawn as before as maps of the entire image
- Each neuron scans and “redraws” the input with some features enhanced
  - The specific features that the neuron detects

Building up patterns



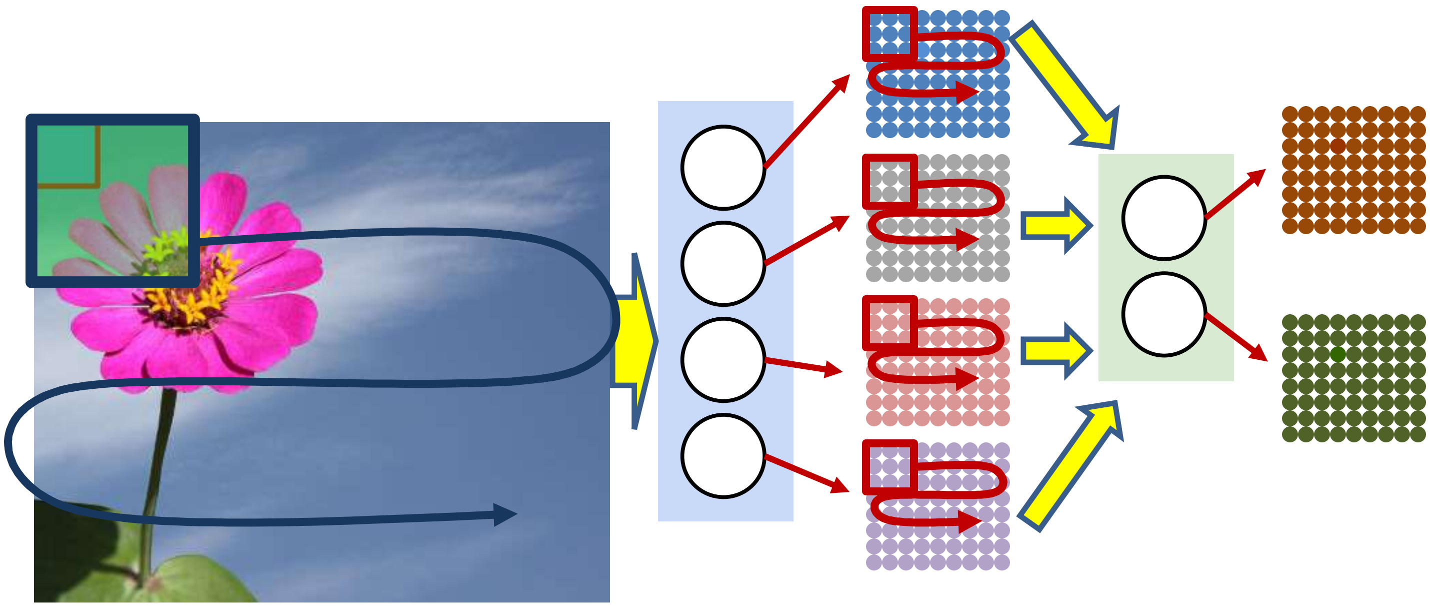
- The first layer looks at small *sub* regions of the input image
  - Sufficient to detect, say, petals
    - And enhances those

The higher-level neurons



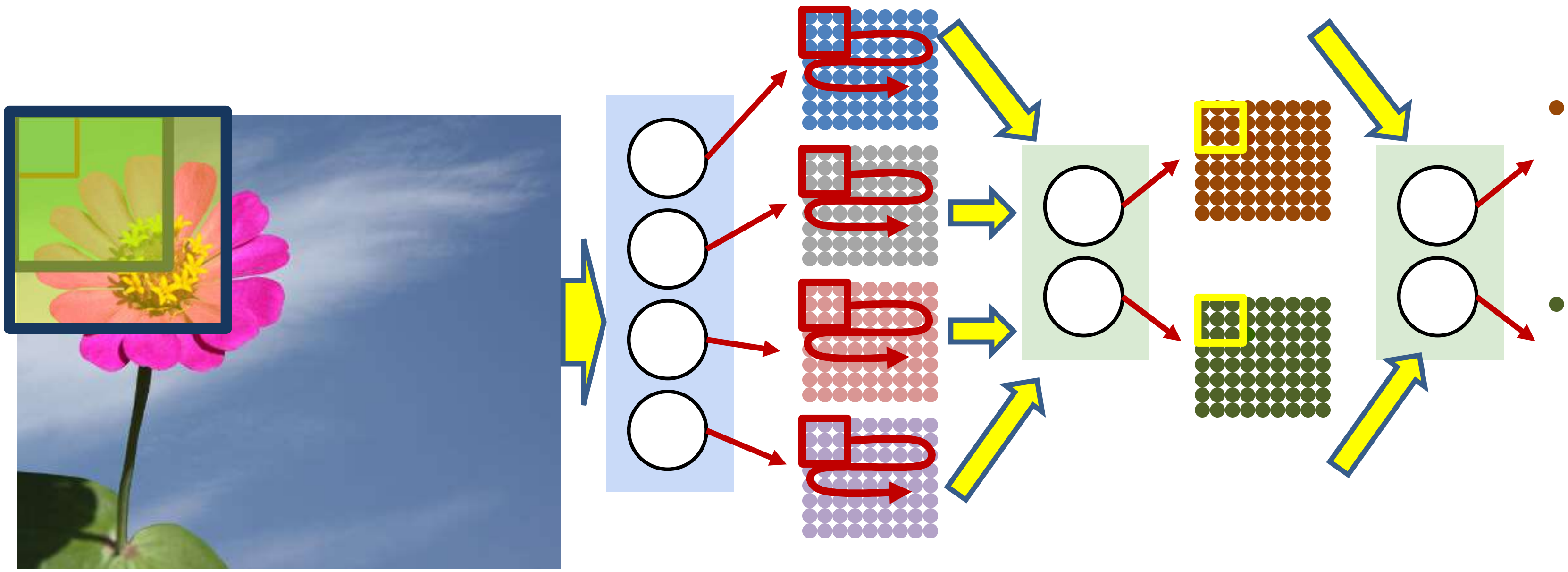
- The first layer looks at *sub* regions of the main image
  - Sufficient to detect, say, petals
- The second layer looks at *regions* of the output of the first layer
  - To put the petals together into a part of a flower
  - This corresponds to looking at a larger region of the original input image

The higher-level neurons



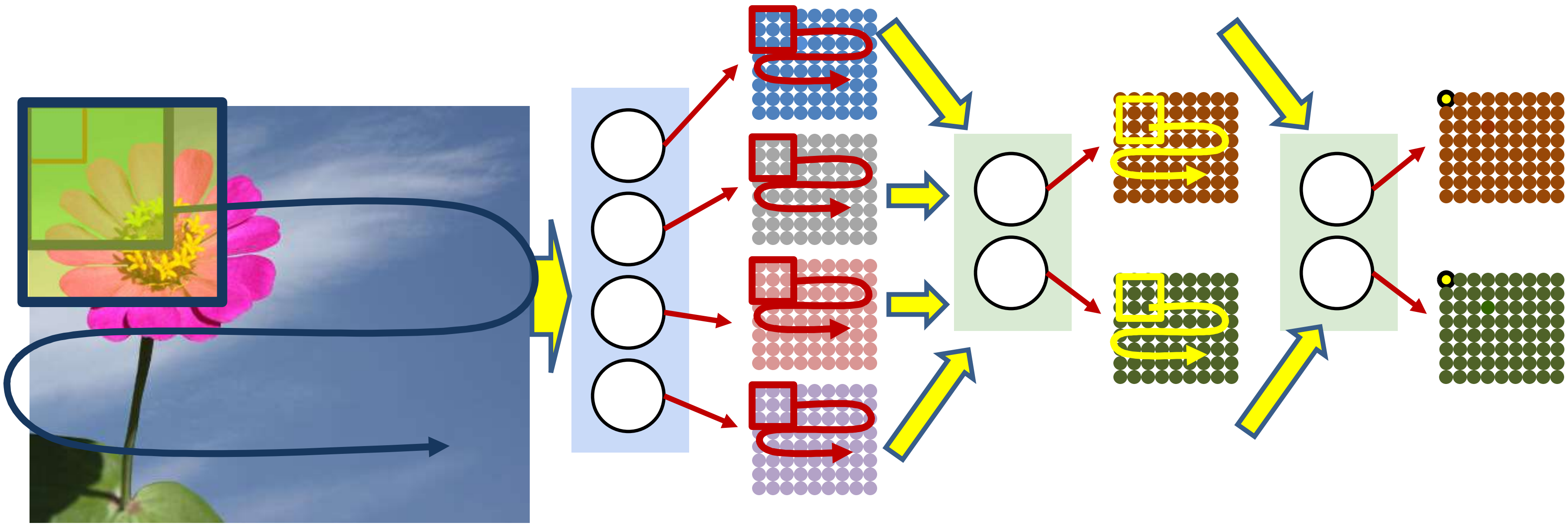
- The first layer looks at *sub* regions of the main image
  - Sufficient to detect, say, petals
- The second layer looks at *regions* of the output of the first layer
  - To put the petals together into a part of a flower
  - This corresponds to looking at a larger region of the original input image

Still-higher level neurons



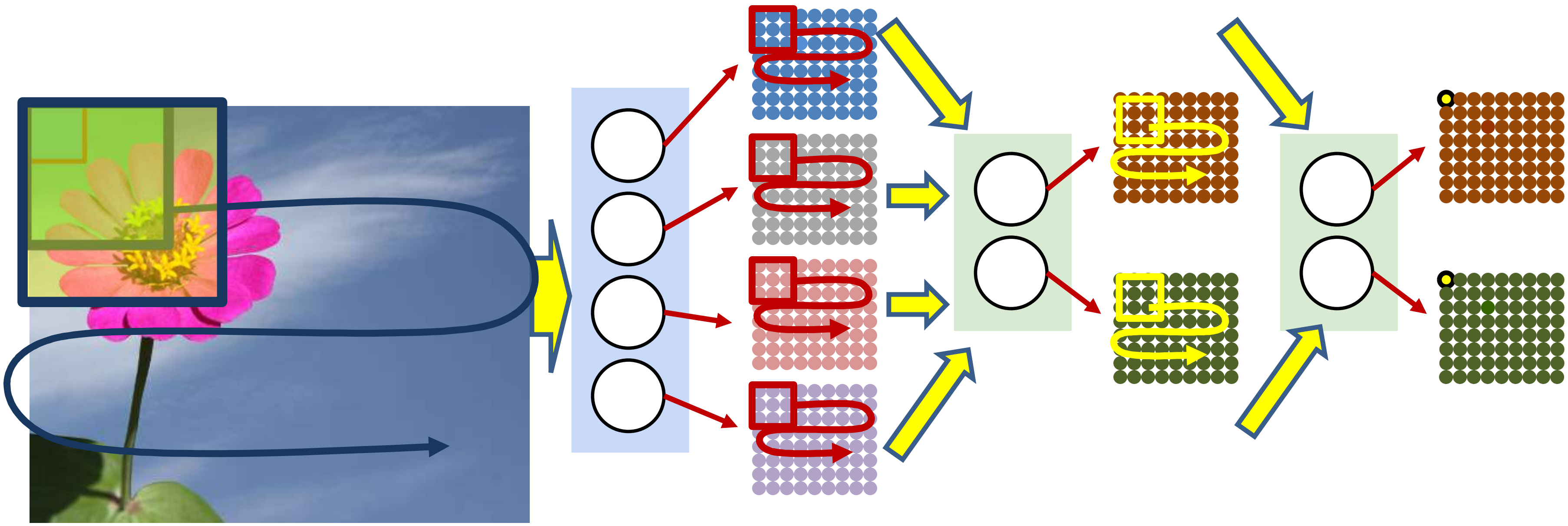
- The first layer looks at *sub* regions of the main image
  - Sufficient to detect, say, petals
- The second layer looks at *regions* of the output of the first layer
  - To put the petals together into a part of a flower
  - This corresponds to looking at a larger region of the original input image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

Still-higher level neurons



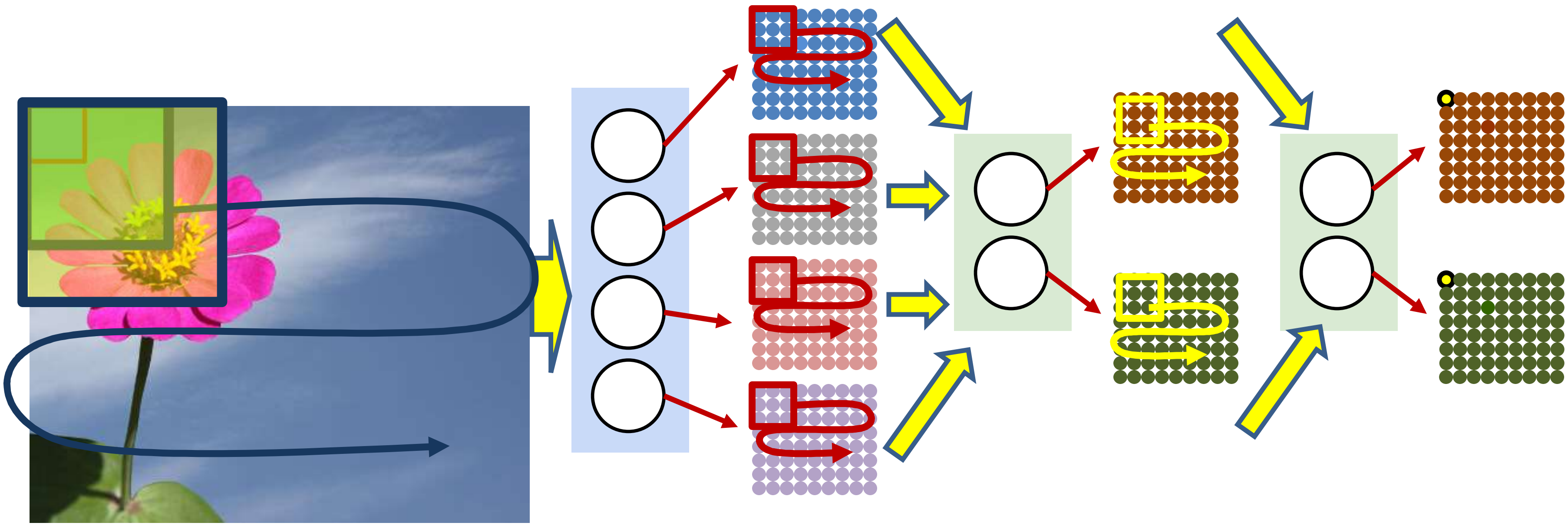
- The first layer looks at *sub* regions of the main image
  - Sufficient to detect, say, petals
- The second layer looks at *regions* of the output of the first layer
  - To put the petals together into a flower
  - This corresponds to looking at a larger region of the original input image
- We may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

Terminology: Filters



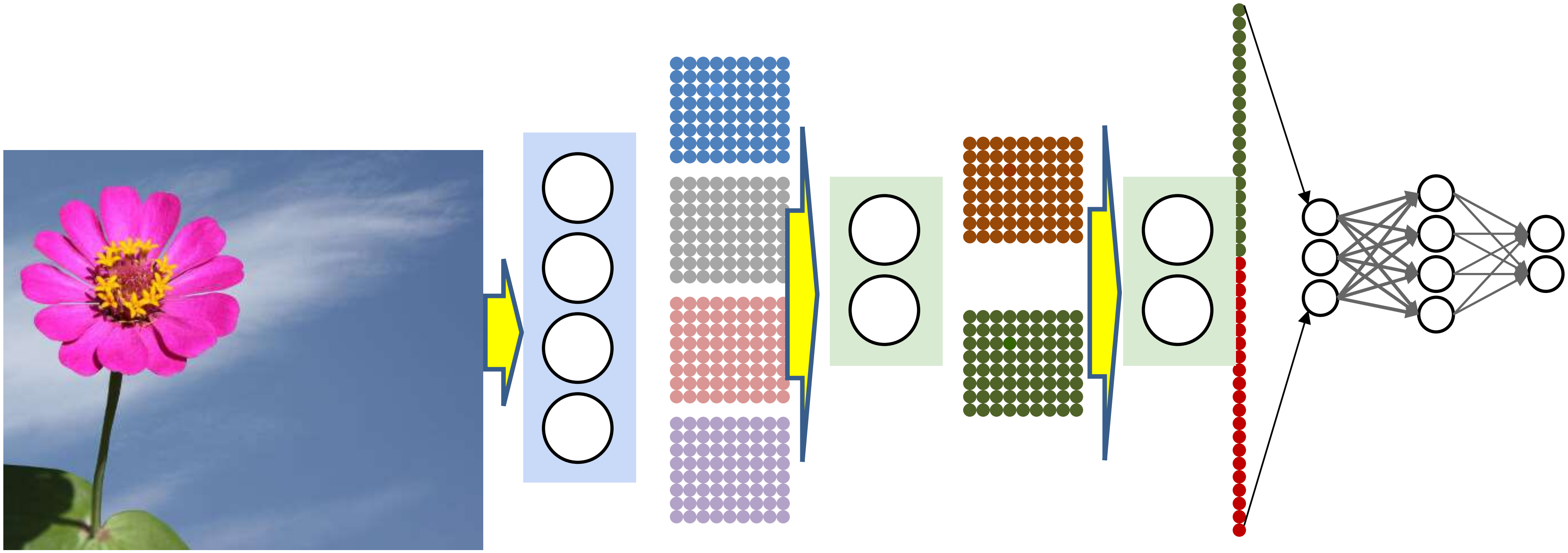
- Each of the scanning neurons is generally called a "filter"
  – Each filter scans for a pattern in the map it operates on

Terminology: Receptive fields



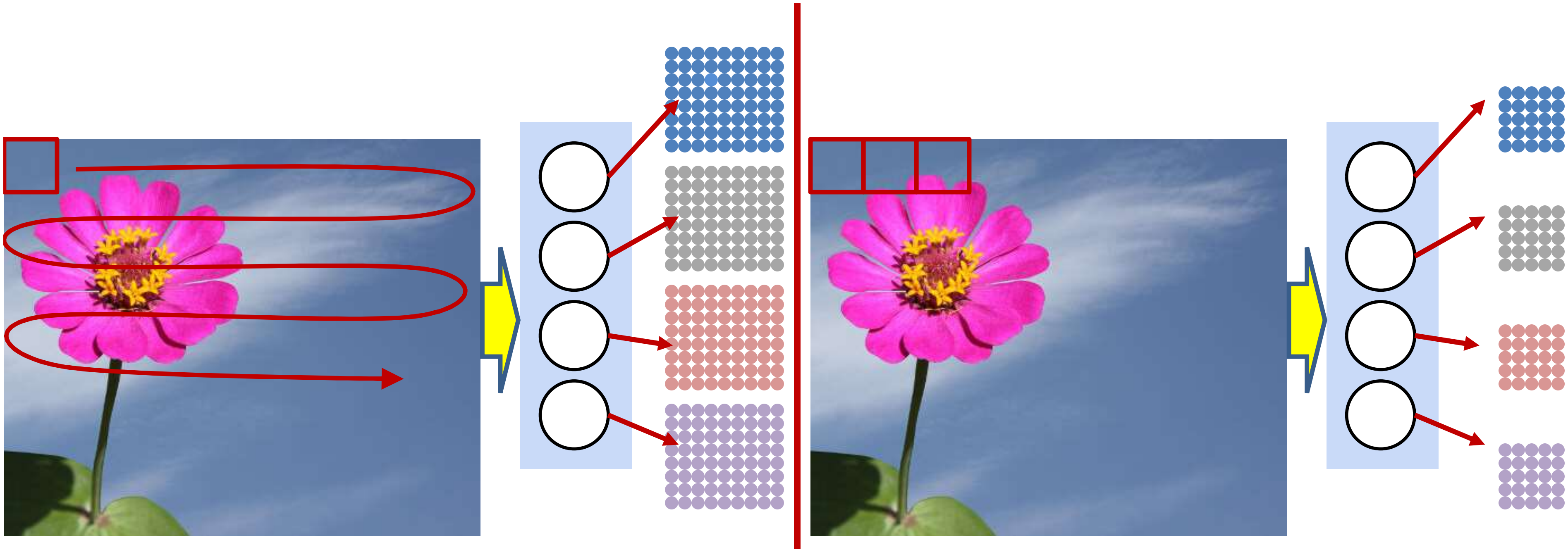
- The pattern in the *input* image that each neuron responds to is its "Receptive Field"
  - The squares show the *sizes* of the receptive fields for the first, second and third-layer neurons
- The actual receptive field for a first layer neuron is simply its arrangement of weights
- For the higher layer neurons, 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"

Terminology: "Flattening"



- The rectangular maps of the neurons in the final layer of the scanning network will generally be reorganized into a vector before passing them to the final softmax or MLP
- This restructuring of the maps is often called "flattening"

Modification 1: Convolutional "Stride"



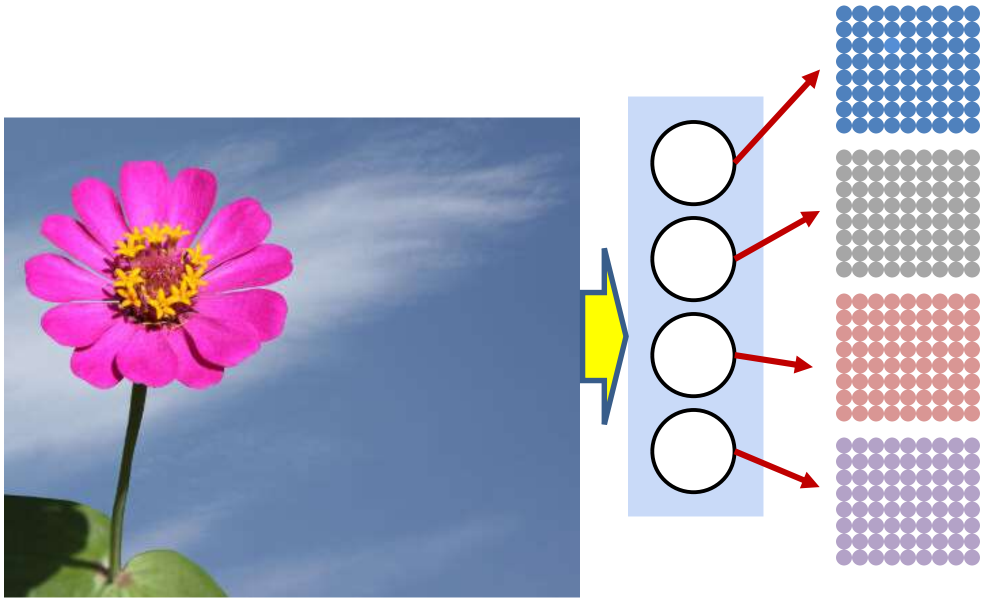
- The scans of the individual "filters" may advance by more than one pixel at a time
  - The "stride" may be greater than 1
  - Effectively increasing the granularity of the scan
    - Saves computation, sometimes at the risk of losing information
- This will result in a reduction of the size of the resulting maps
  - They will shrink along each axis by a factor equal to the stride
  - To prevent guaranteed loss of information by the shrinking, the number of output maps (neurons) must be S^2 the number of input maps, where S is the stride
- This can happen at any layer

CNN with strides

The weight W(l,j)**is now a 3D $D_{l-1}\times K_l \times K_l$ tensor (assuming square receptive fields)**

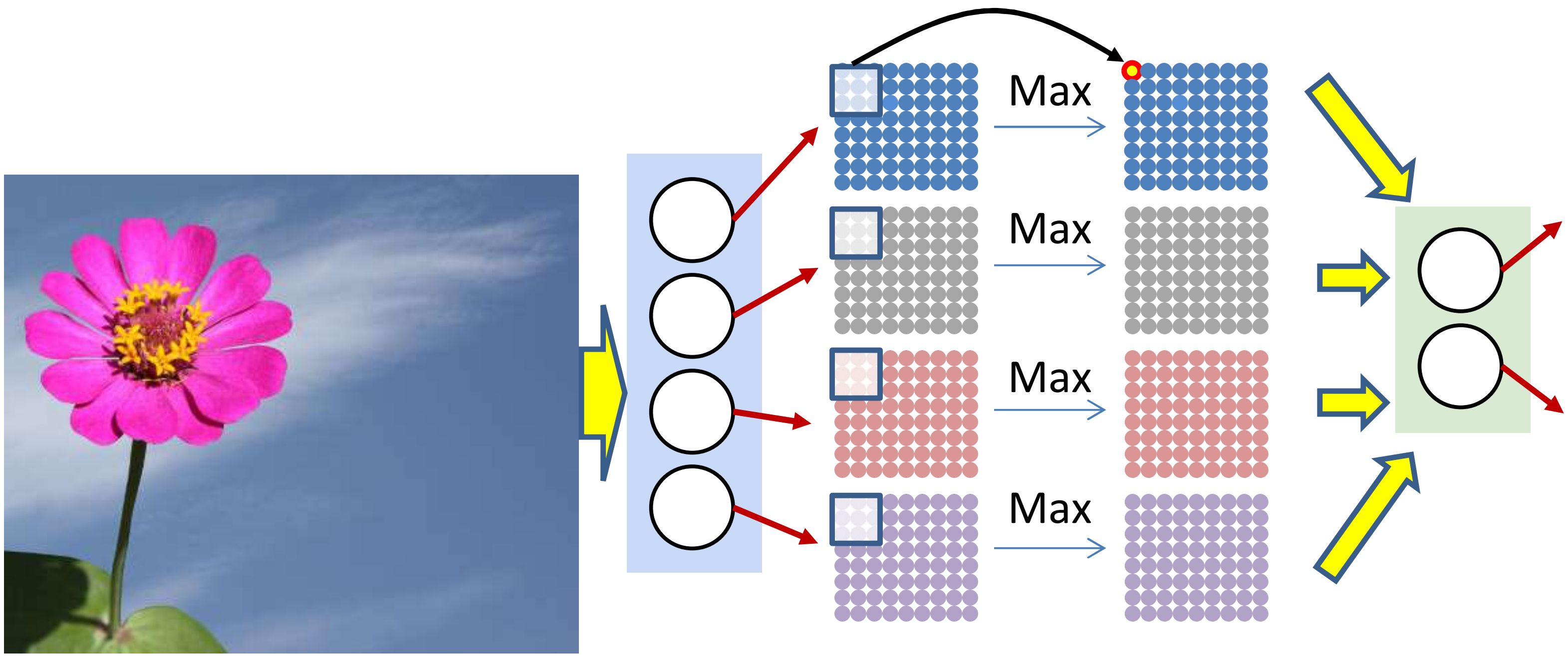
```
Y(0) = Image
for l = 1:L  # layers operate on vector at (x,y)
  for j = 1:D_l
    m = 1
    for x = 1:stride:W_{l-1}-K_l+1
       n = 1
       for y = 1:stride:H_{l-1}-K_l+1
          segment = Y(l-1,:,x:x+K_l-1,y:y+K_l-1) #3D tensor
          z(l,j,m,n) = W(l,j).segment #tensor inner prod.
          Y(l,j,m,n) = activation(z(l,j,m,n))
          n++
        m++

Y = softmax( Y(L) )
```

Modification 2: Accounting for jitter



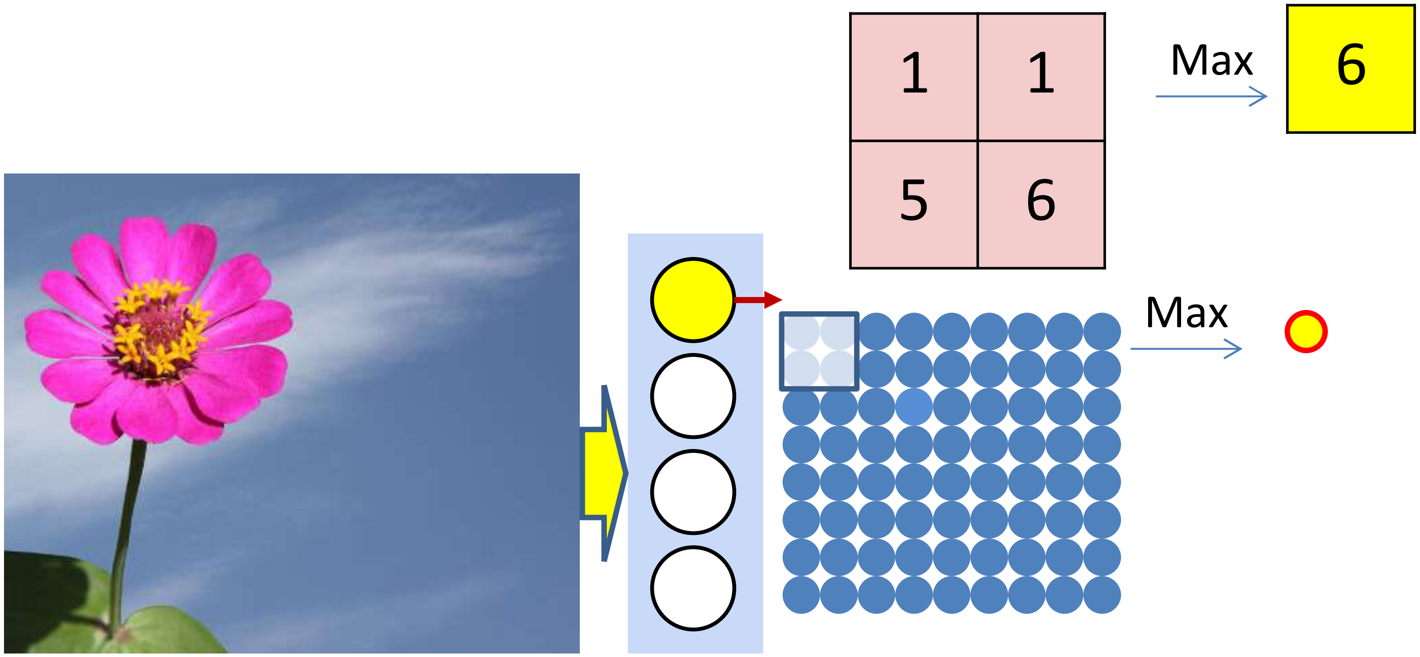
- We would like to account for some jitter in the first-level patterns
  - If a pattern shifts by one pixel, is it still a petal?

Accounting for jitter



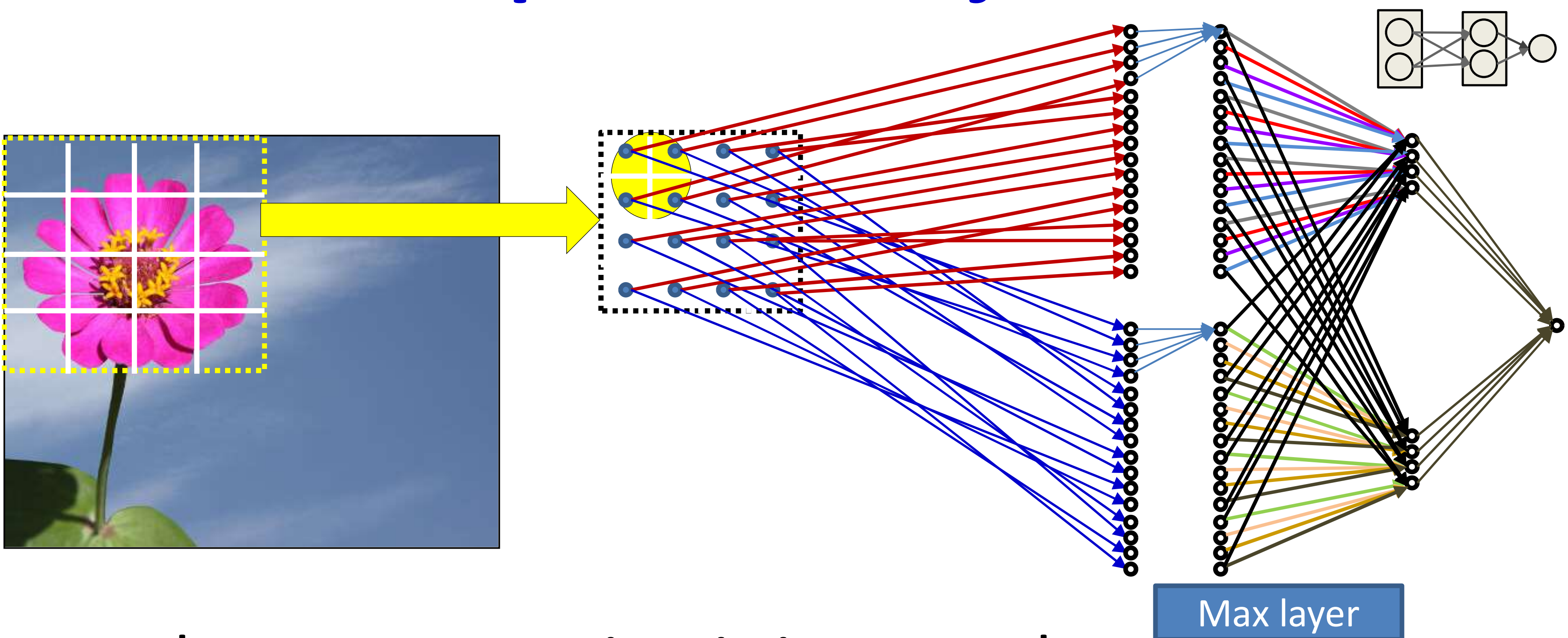
- We would like to account for some jitter in the first-level patterns
  - If a pattern shifts by one pixel, is it still a petal?
  - A small jitter is acceptable
    - Replace each value by the maximum of the values within a small region around it
      - ***Max filtering or Max pooling***

Max filter



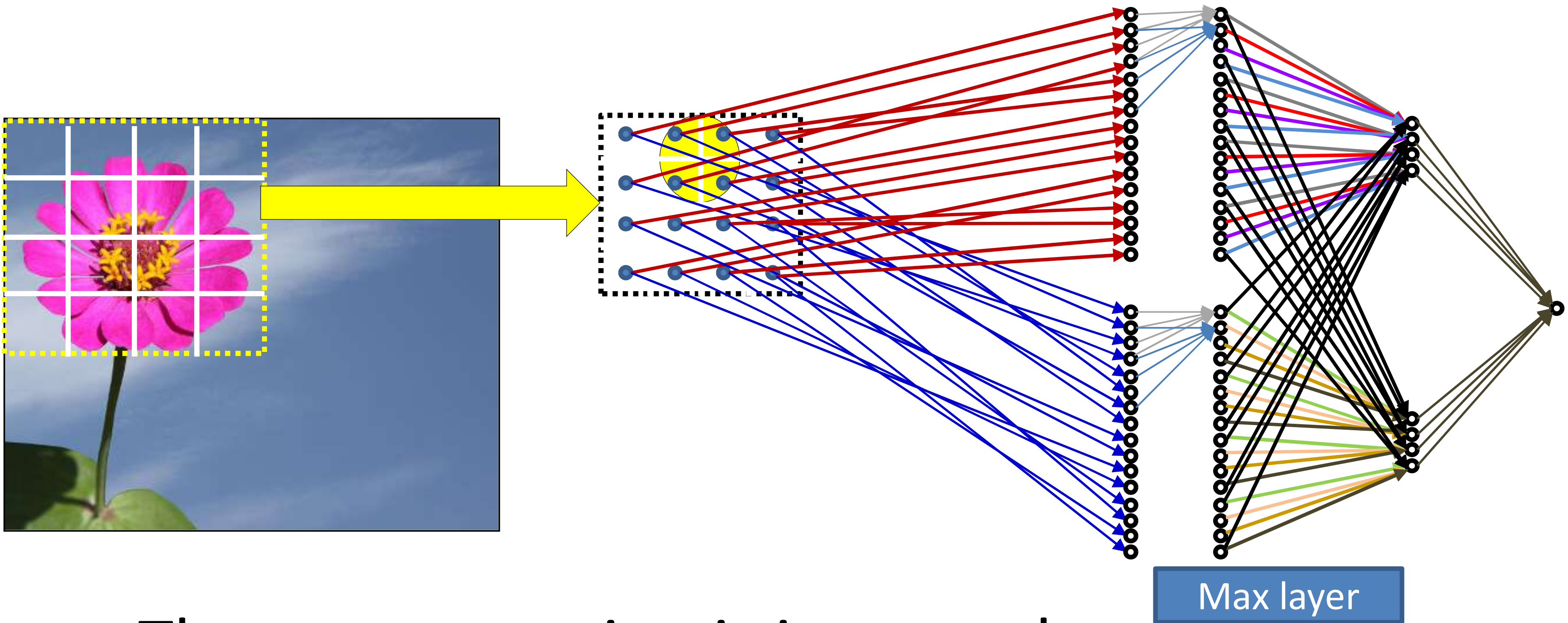
- We would like to account for some jitter in the first-level patterns
  - If a pattern shifts by one pixel, is it still a petal?
  - A small jitter is acceptable
    - Replace each value by the maximum of the values within a small region around it
      - ***Max filtering*** **or** ***Max pooling***

The max operation is just a neuron



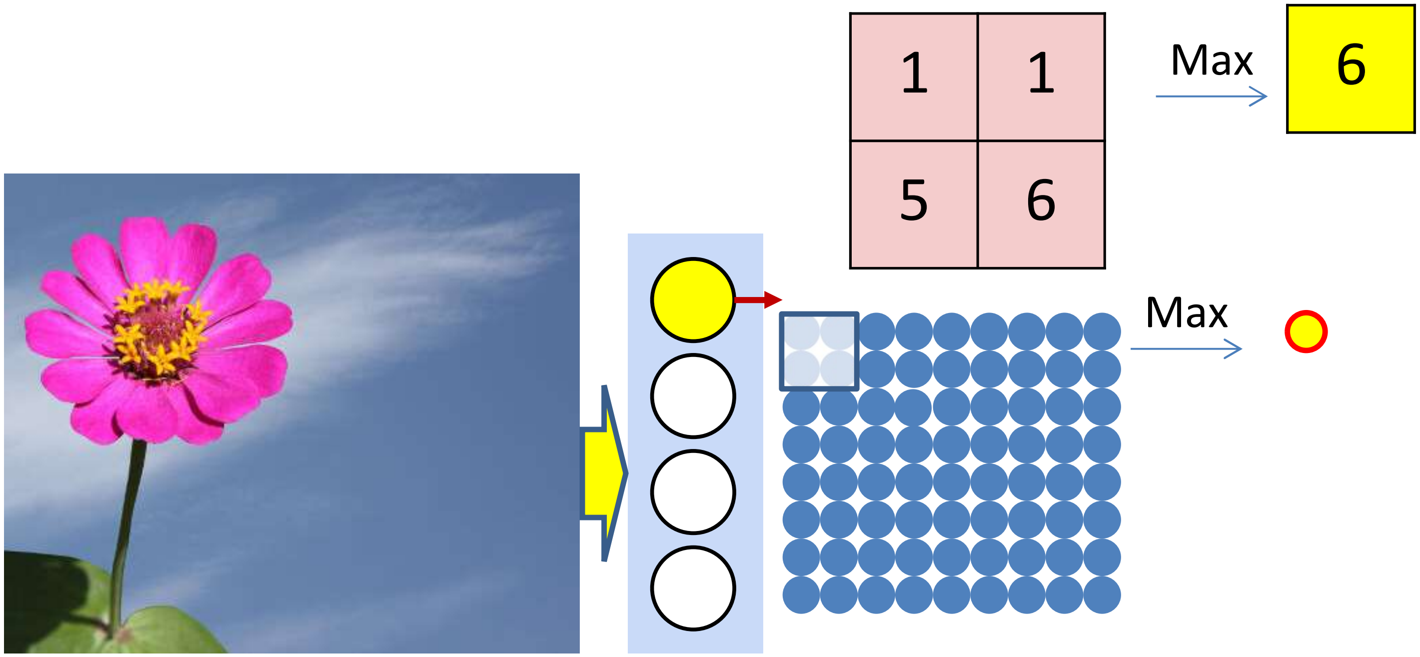
- The max operation is just another neuron
- Instead of applying an activation to the weighted sum of inputs, each neuron just computes the maximum over all inputs

The max operation is just a neuron



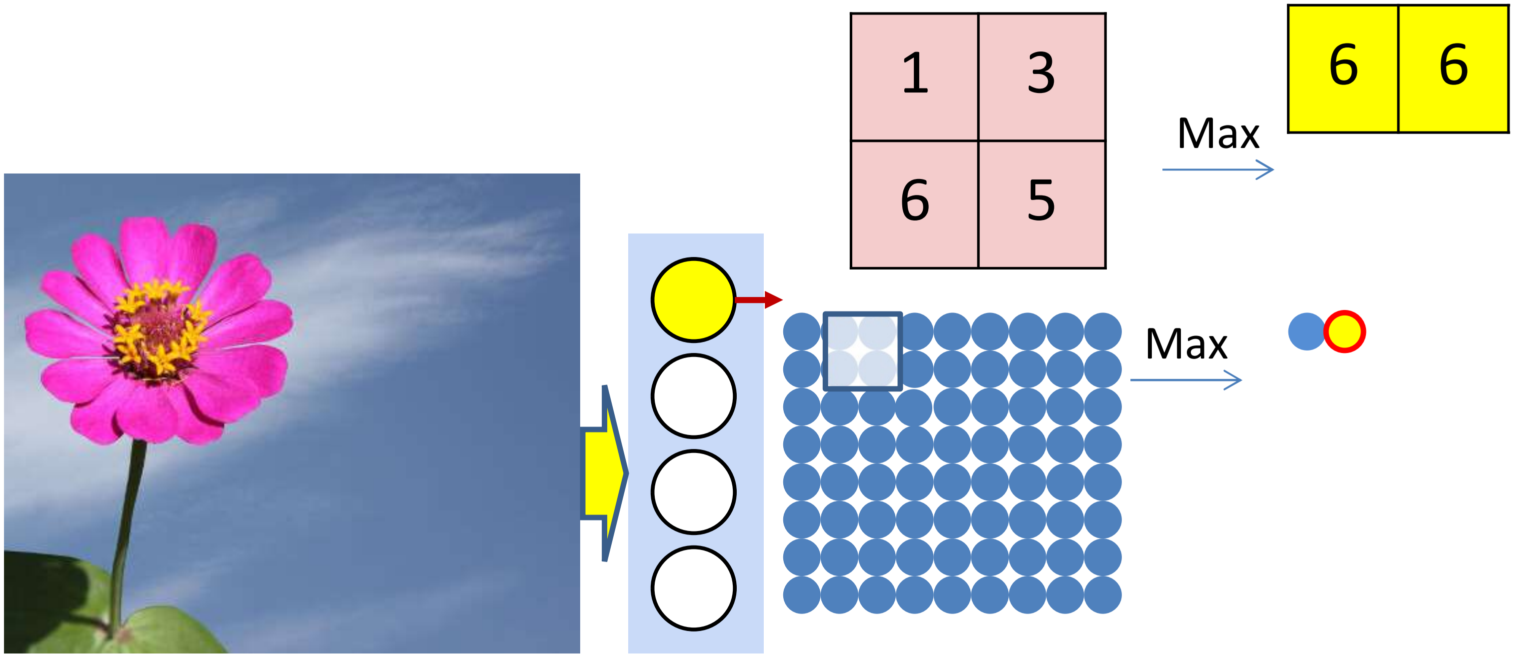
- The max operation is just another neuron
- Instead of applying an activation to the weighted sum of inputs, each neuron just computes the maximum over all inputs

Scanning with the max



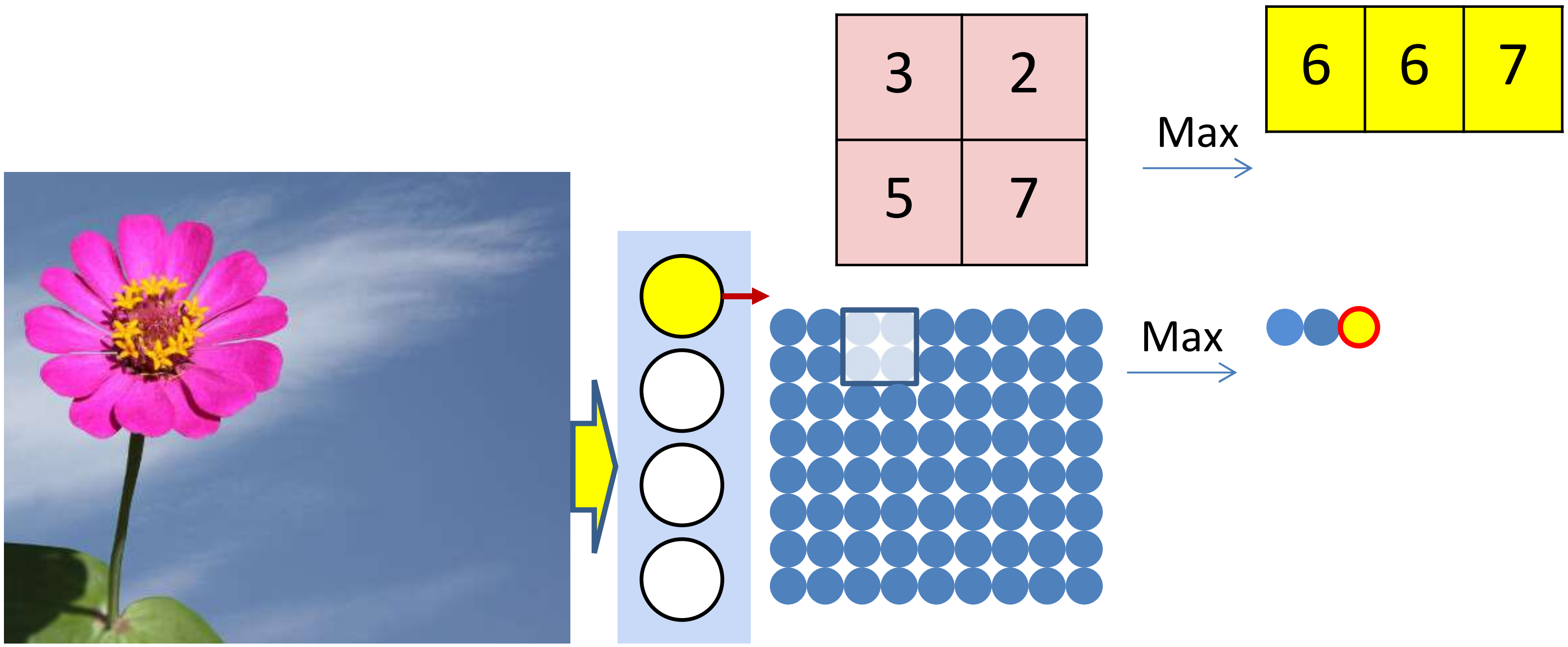
- The max filtering can also be performed as a scan

Scanning with the max



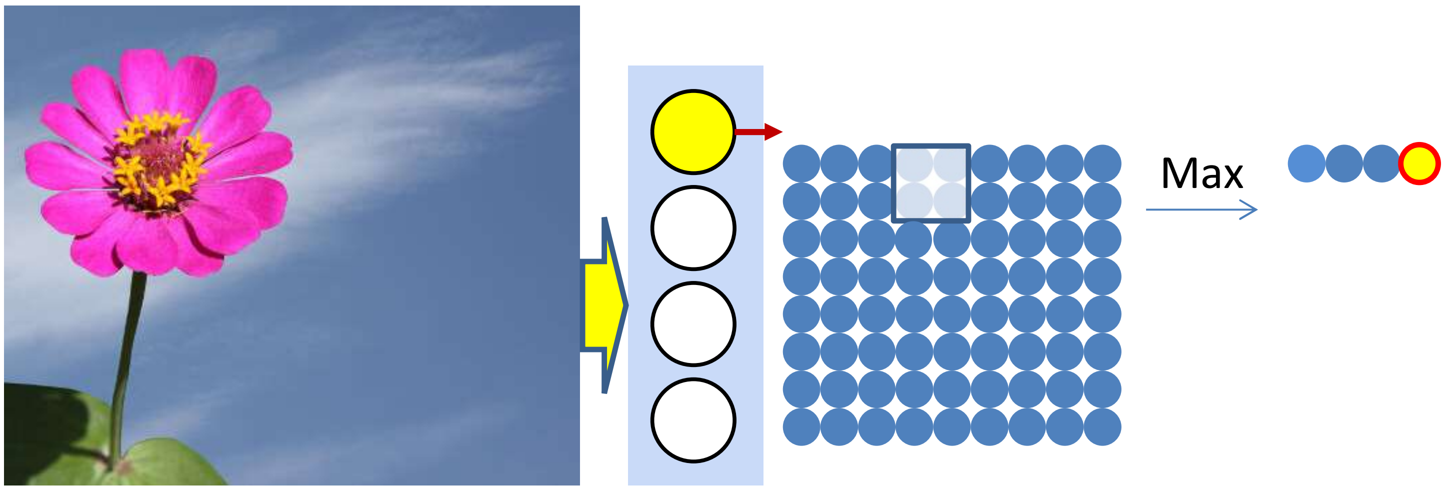
- The "max filter" operation too "scans" the picture

Scanning with the max



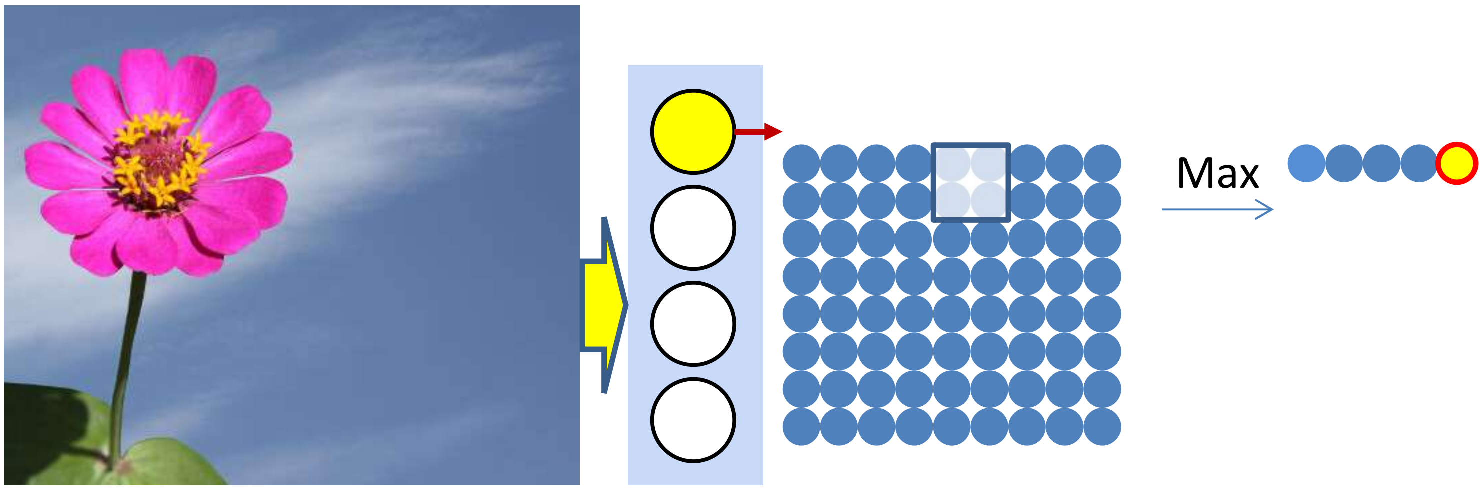
- The “max filter” operation too “scans” the picture

Scanning with the max



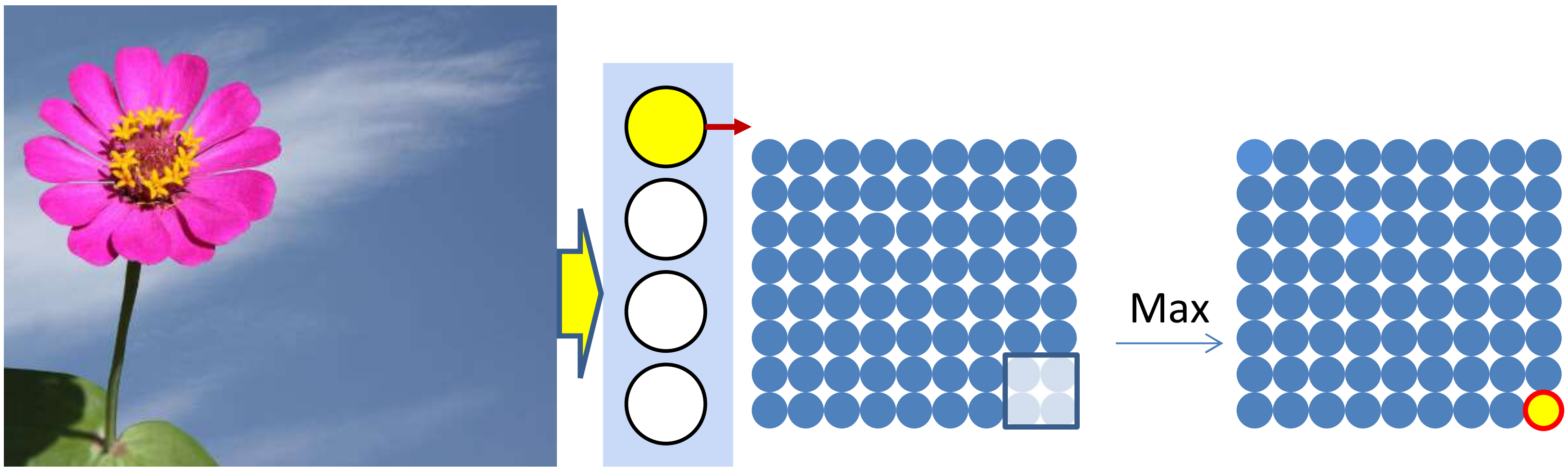
- The “max filter” operation too “scans” the picture

Scanning with the max



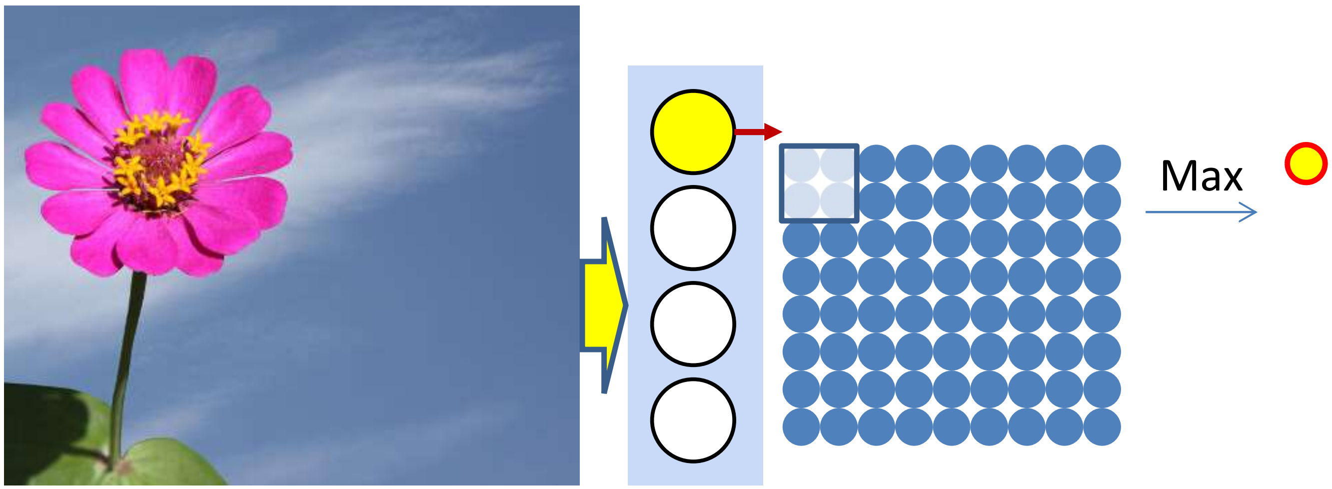
- The "max filter" operation too "scans" the picture

Scanning with the max



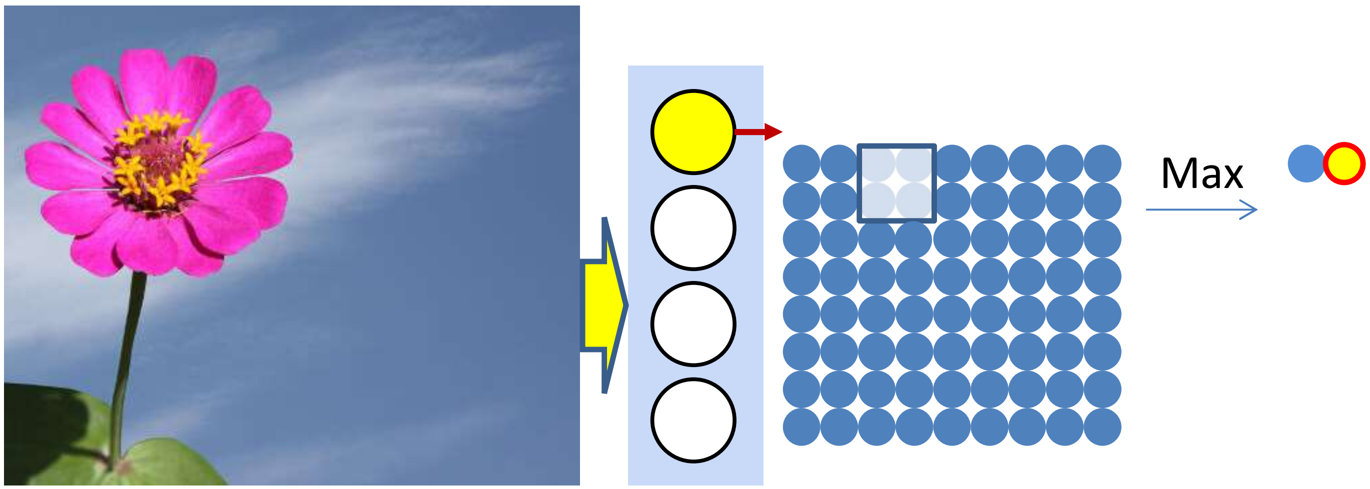
- The “max filter” operation too “scans” the picture

Max pooling "Strides"



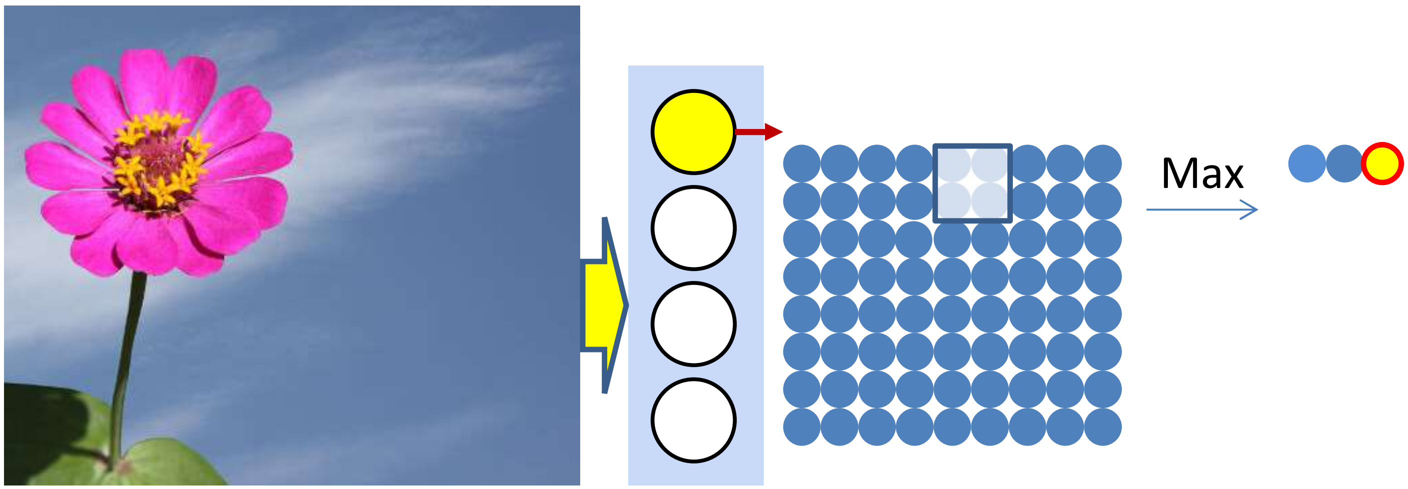
- The "max" operations may "stride" by more than one

Max pooling "Strides"



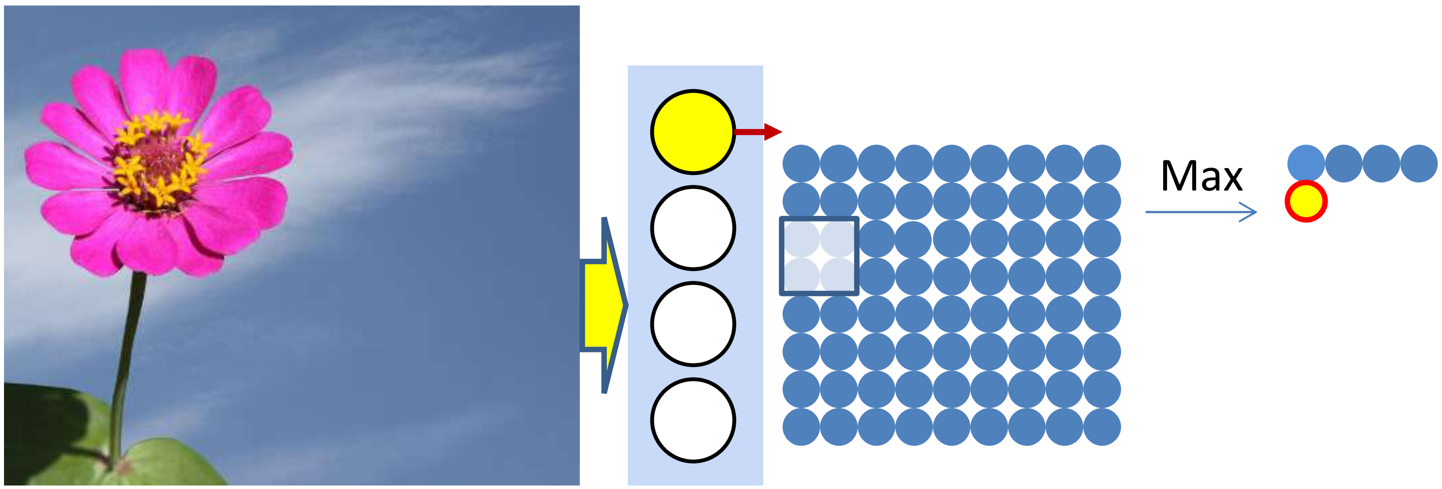
- The "max" operations may "stride" by more than one

Max pooling "Strides"



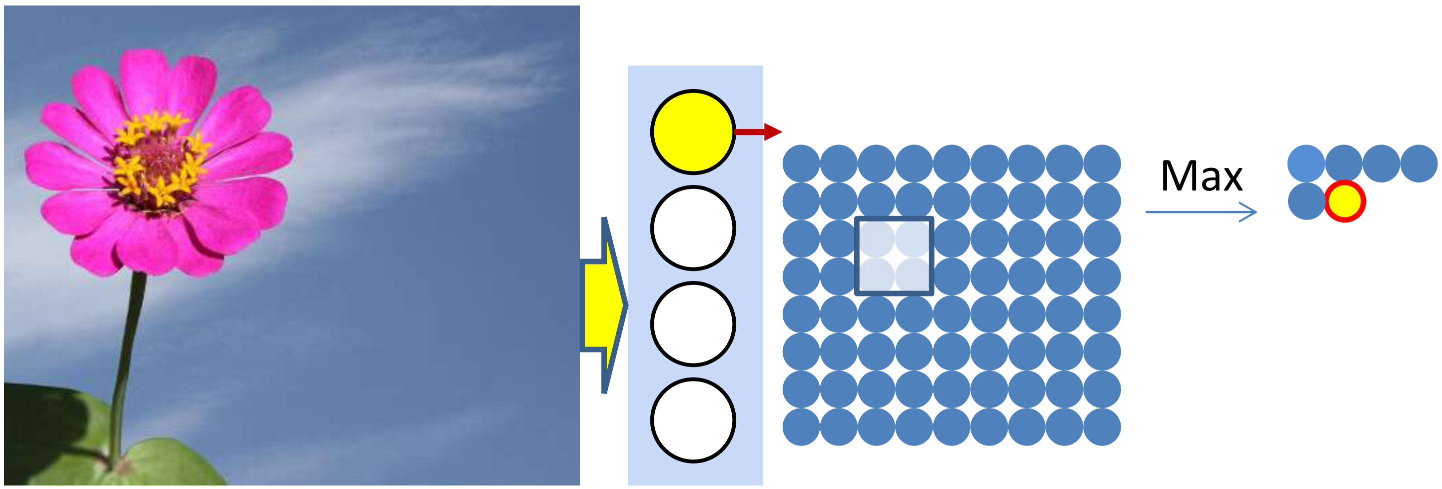
- The "max" operations may "stride" by more than one

Max pooling "Strides"



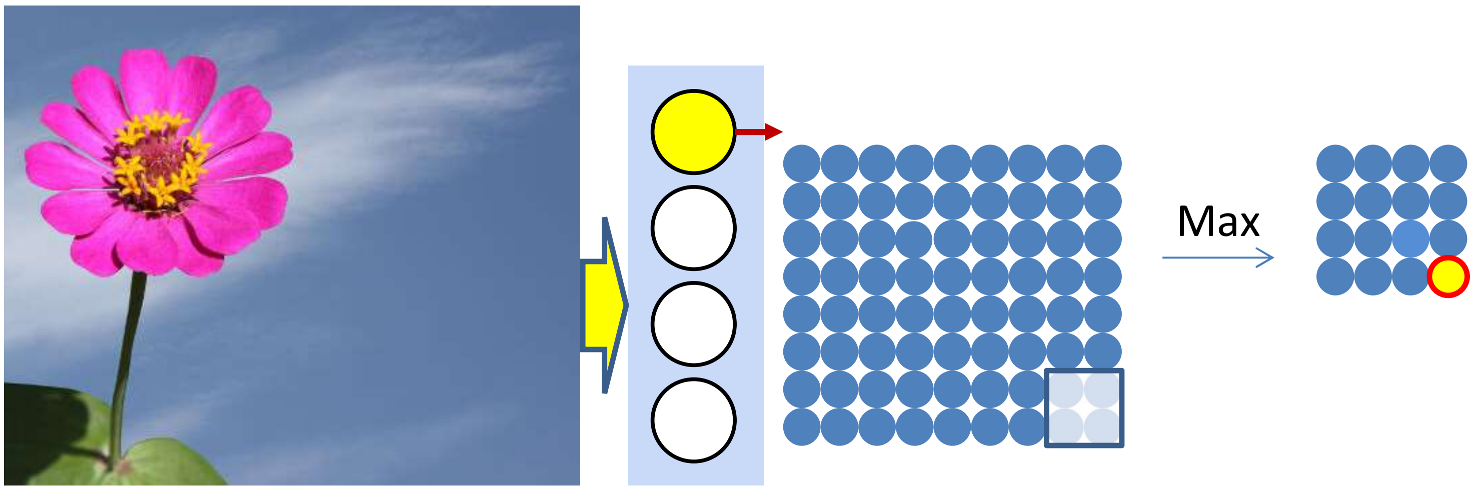
- The "max" operations may "stride" by more than one

Max pooling "Strides"



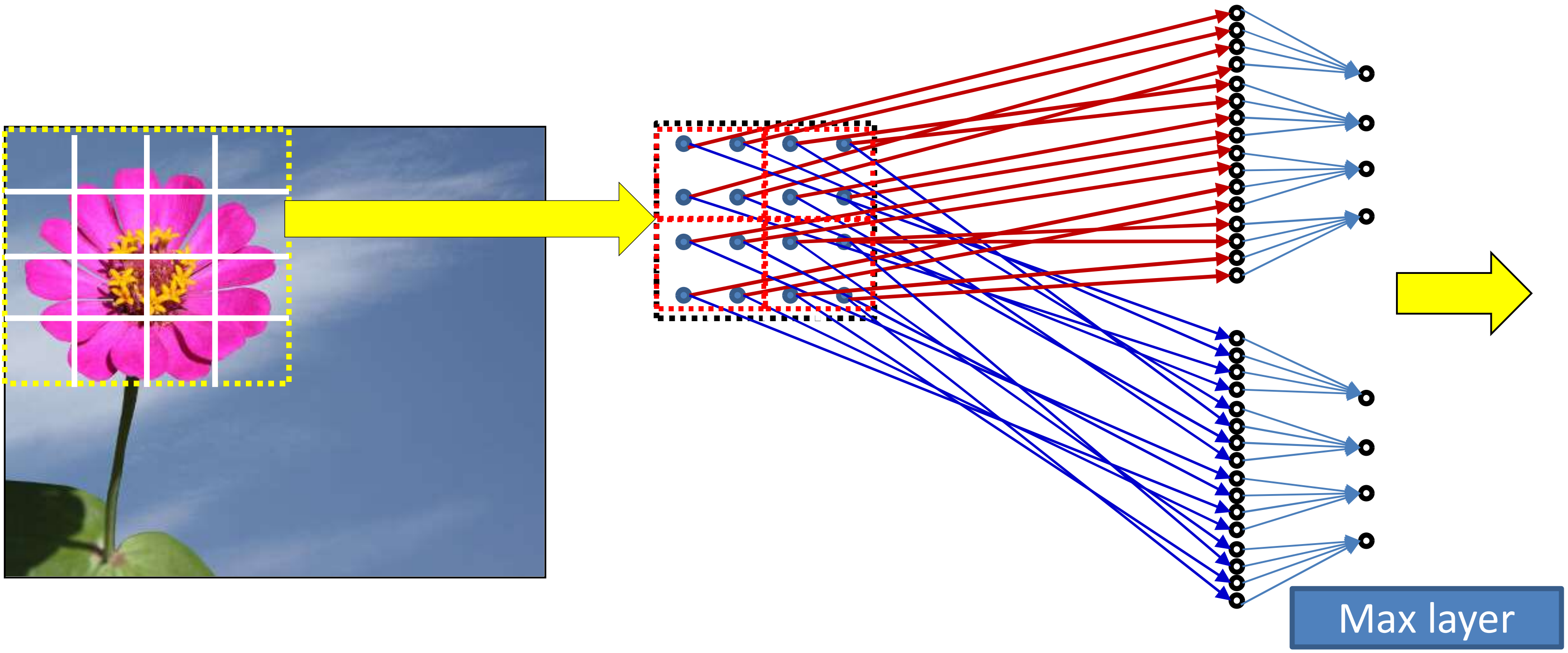
- The "max" operations may "stride" by more than one

Max pooling "Strides"



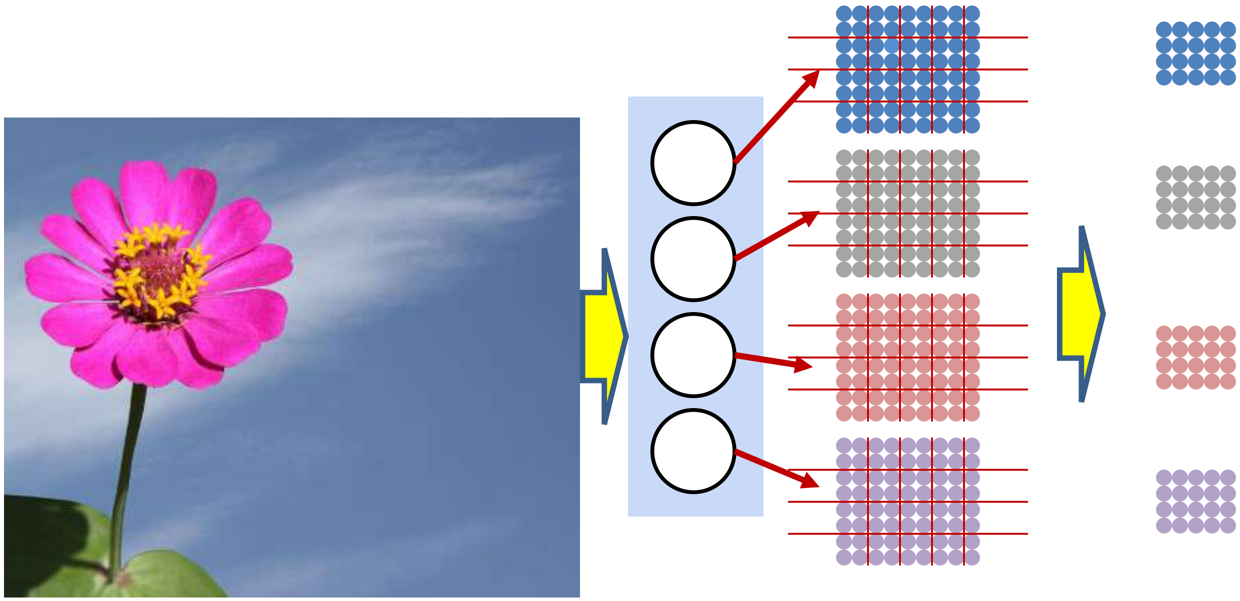
- The "max" operations may "stride" by more than one
  - This will result in a *shrinking* of the map
  - The operation is usually called "pooling"
    - Pooling a number of outputs to get a single output
    - When stride is greater than 1, also called "Down sampling"

Shrinking with a max



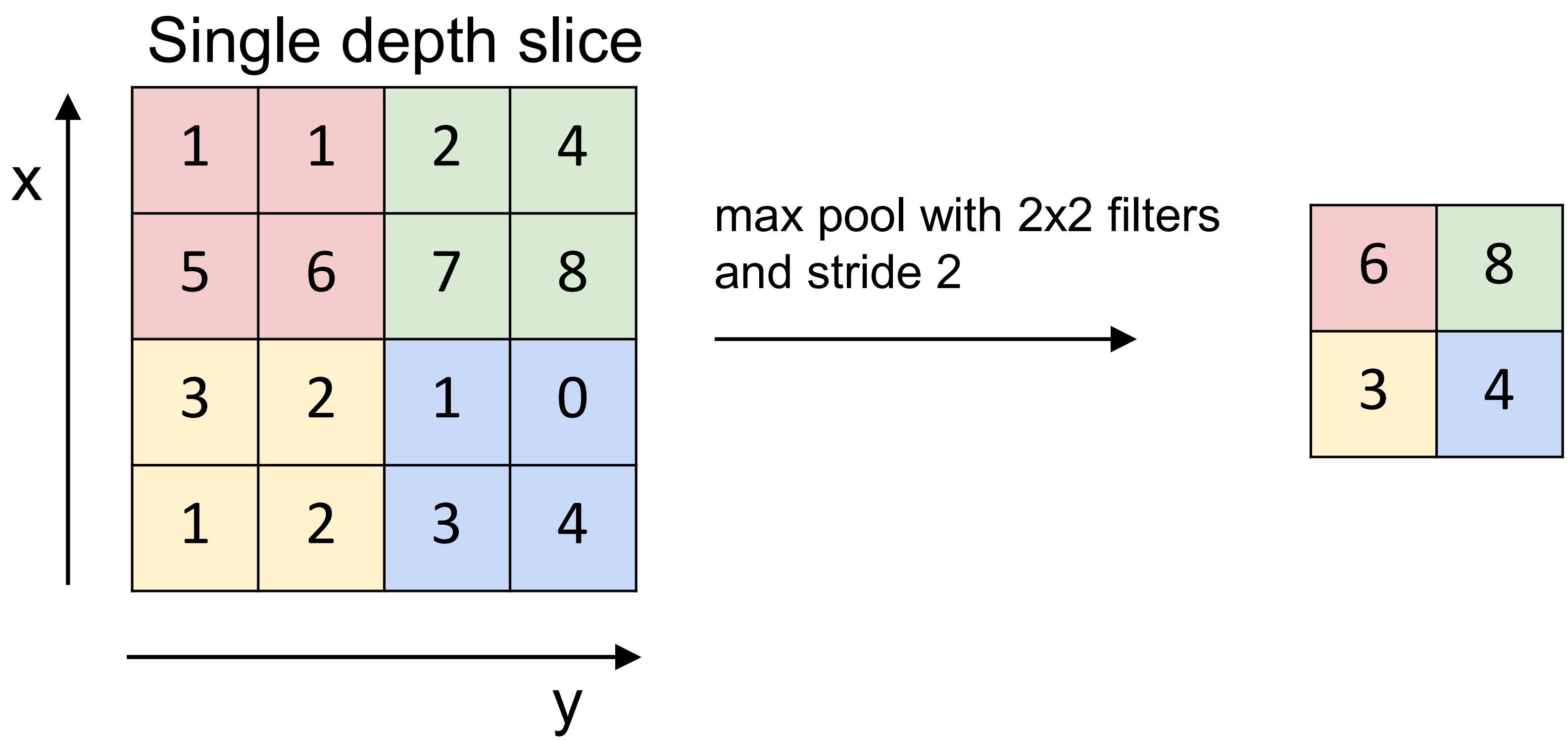
- In this example we *shrank* the image after the max
  - Adjacent “max” operators did not overlap
  - The stride was the size of the max filter itself

Non-overlapped strides



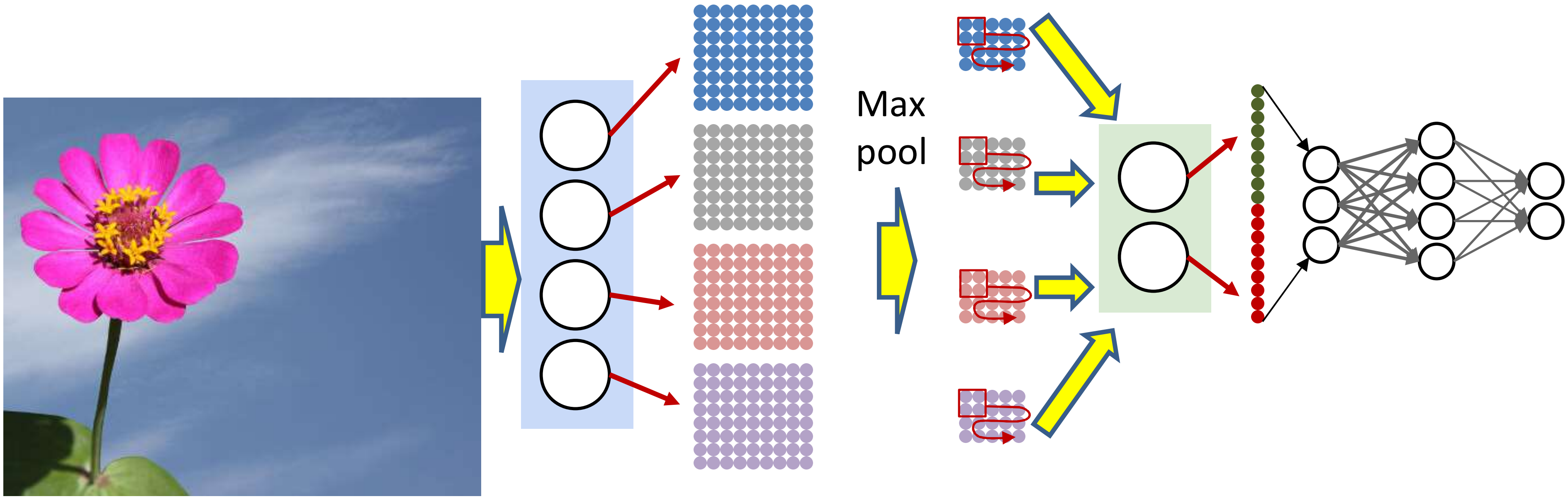
- Non-overlapping strides: Partition the output of the layer into blocks
- Within each block only retain the *highest* value
  - If you detect a petal anywhere in the block, a petal is detected..

Max Pooling



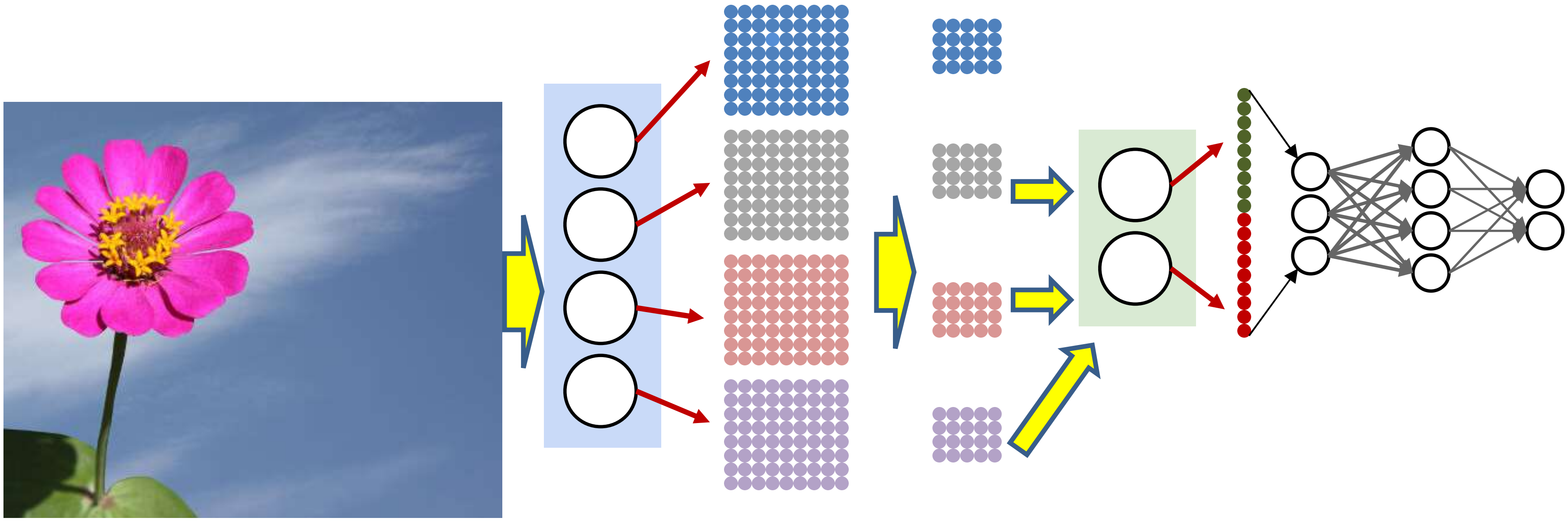
Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

Higher layers



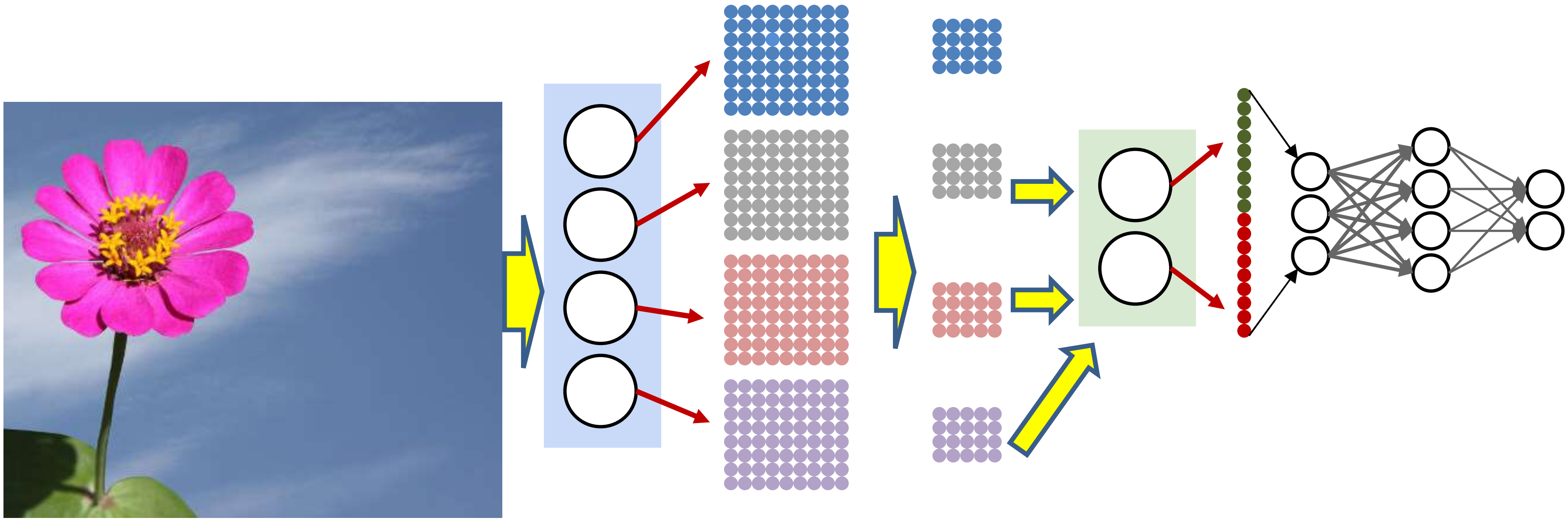
- The next layer works on the *max-pooled* maps

The overall structure



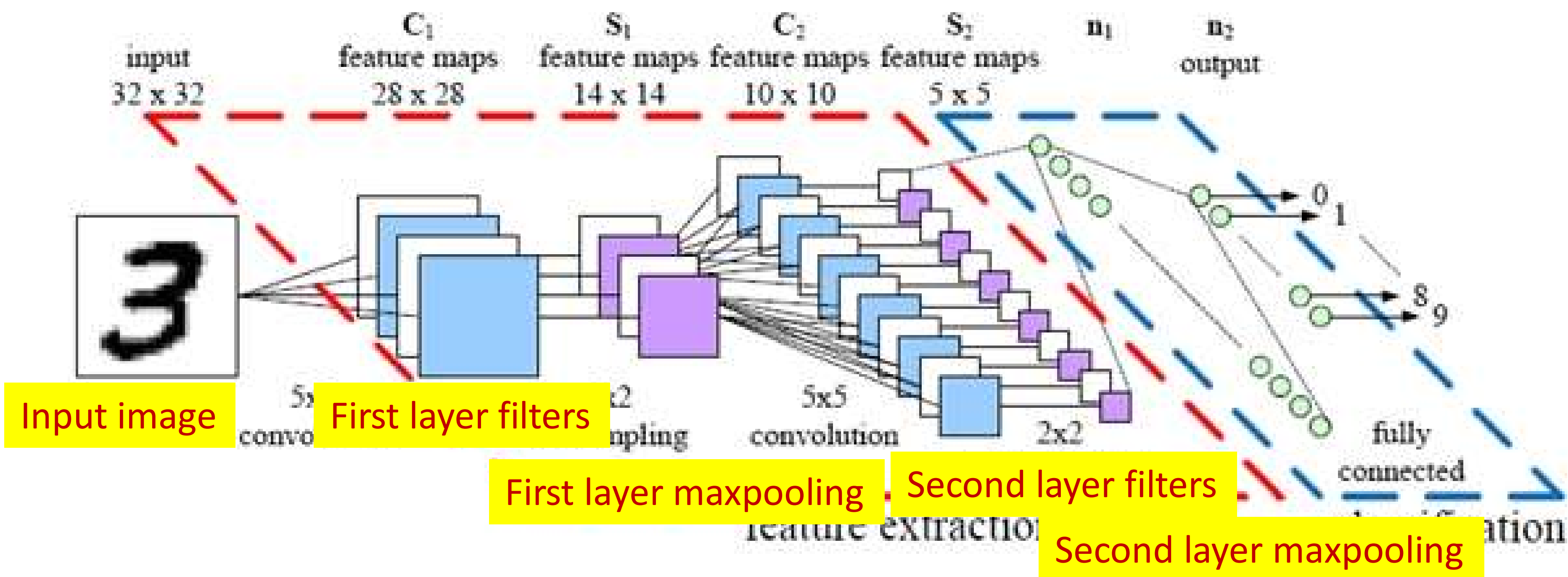
- We can have many layers of “convolution” (scanning) followed by max pooling (and reduction) before the final MLP
  - Not every convolutional layer needs to be followed by max pooling

The overall structure

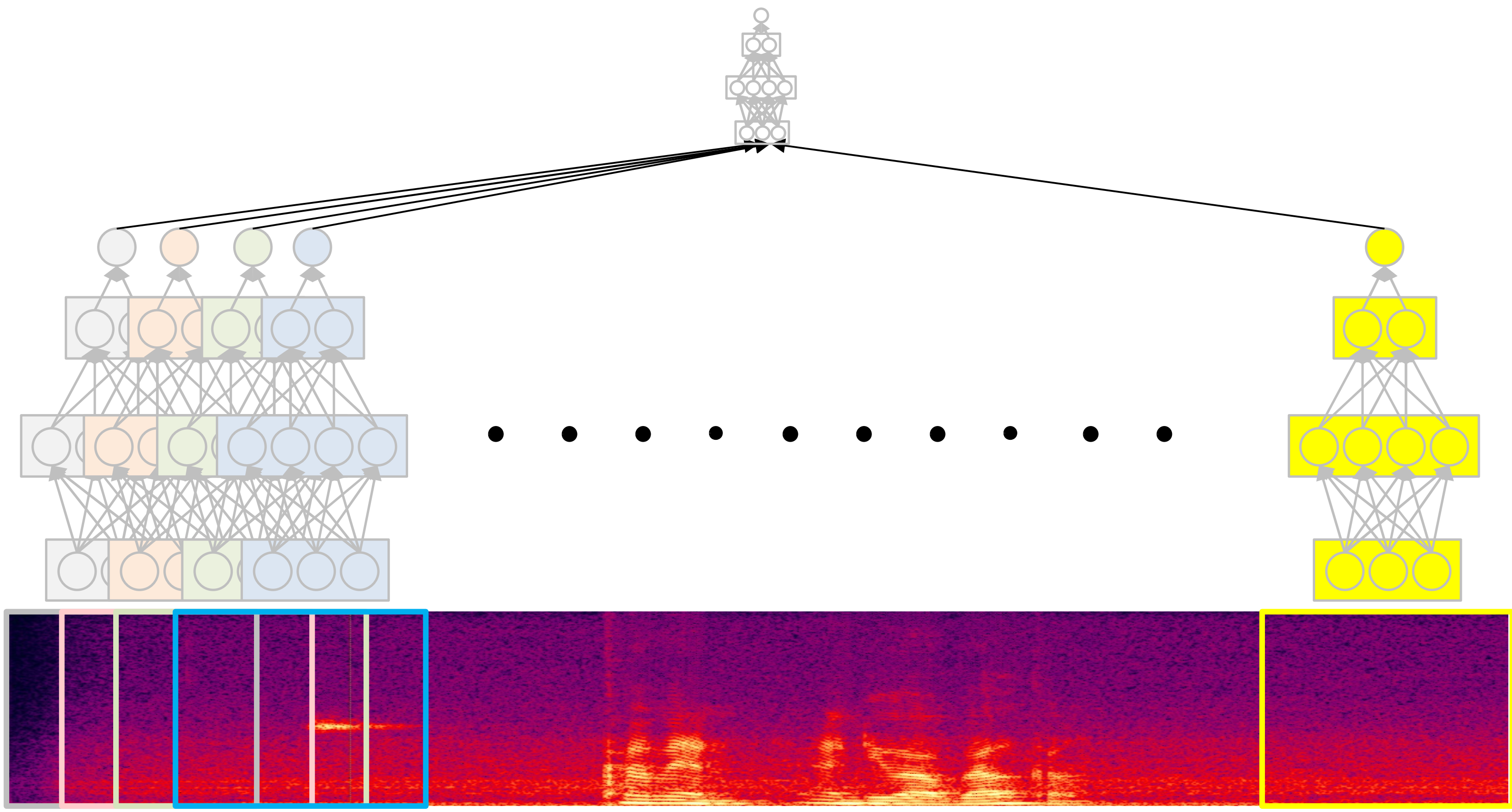


- This entire structure is called a ***Convolutional Neural Network***

Convolutional Neural Network

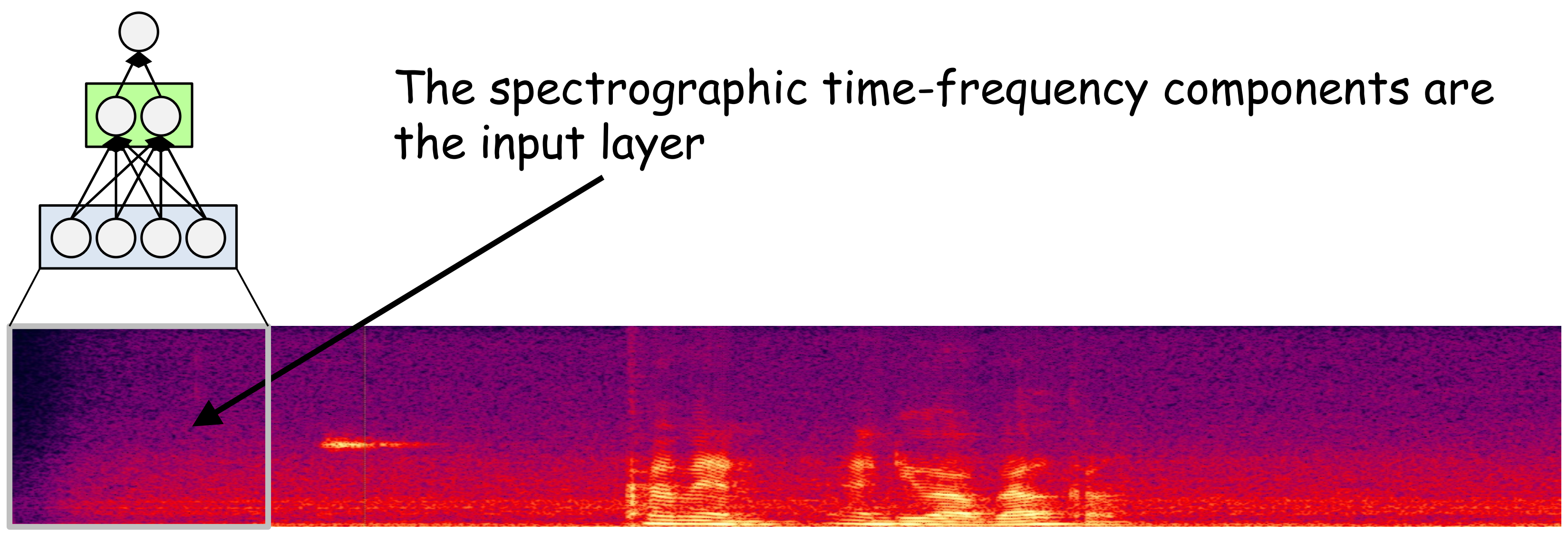


1-D convolution



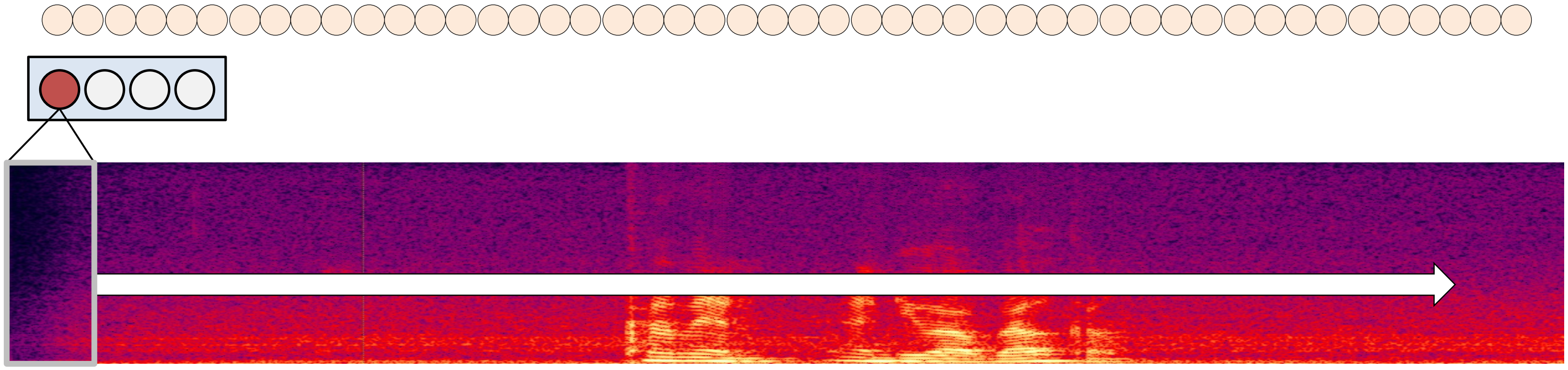
- The 1-D scan version of the convolutional neural network is the *time-delay neural network*
  – Used primarily for speech recognition

1-D scan version



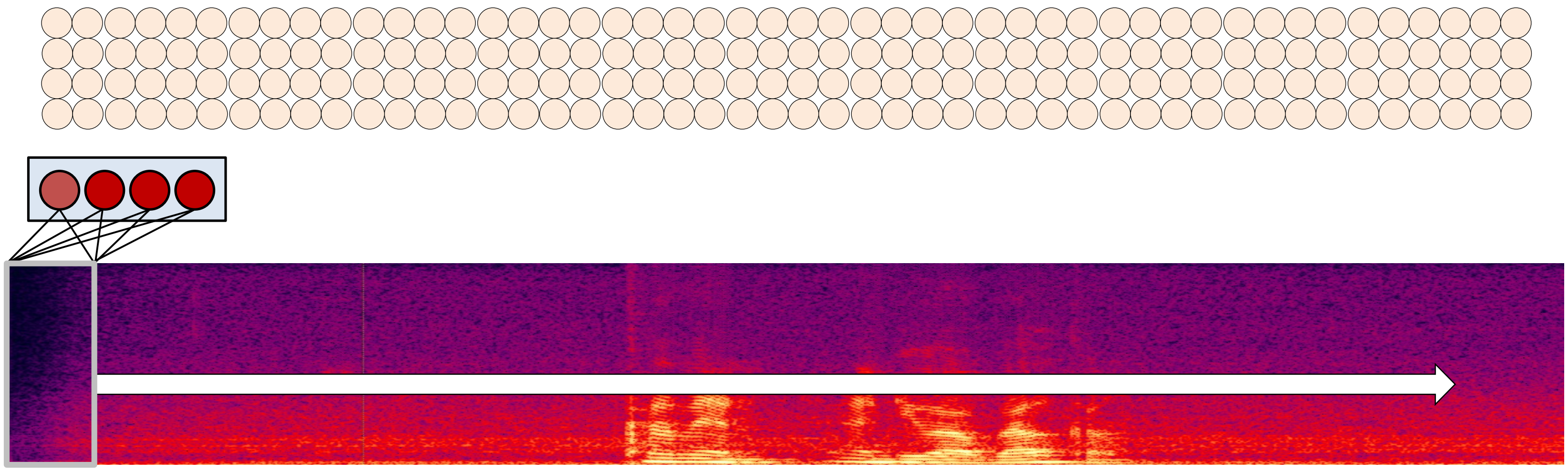
- The 1-D scan version of the convolutional neural network

1-D scan version



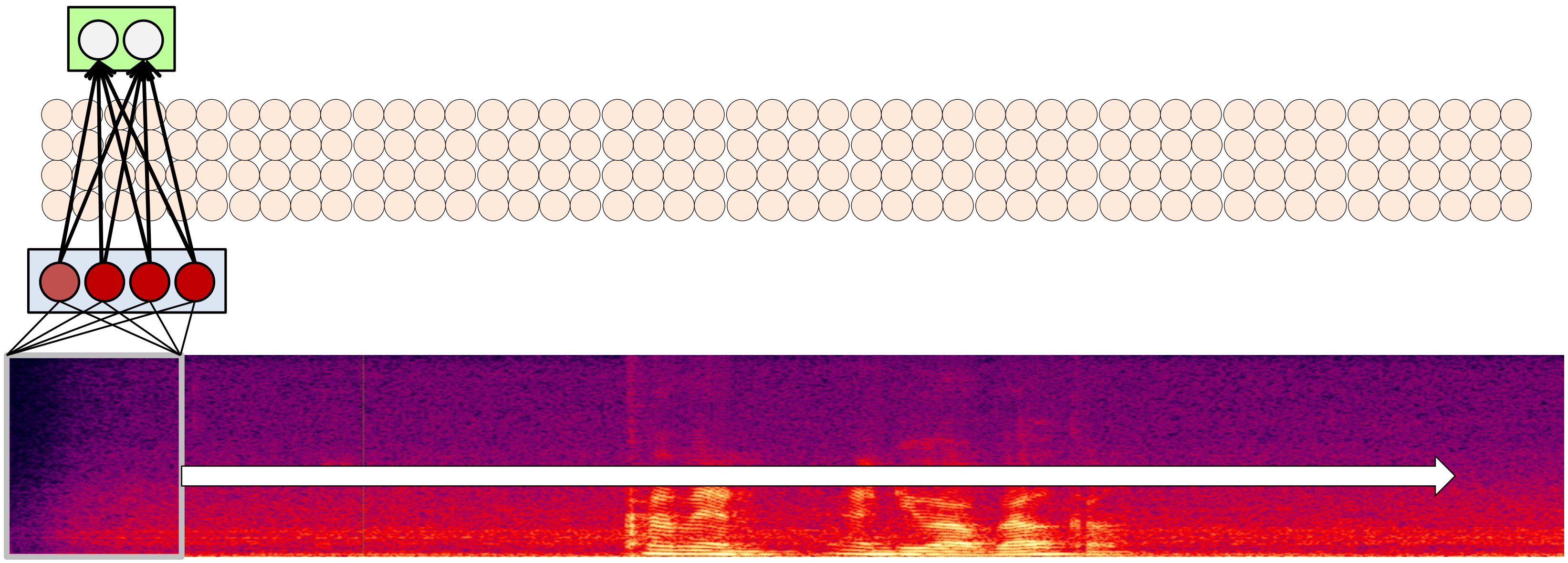
- The 1-D scan version of the convolutional neural network

1-D scan version



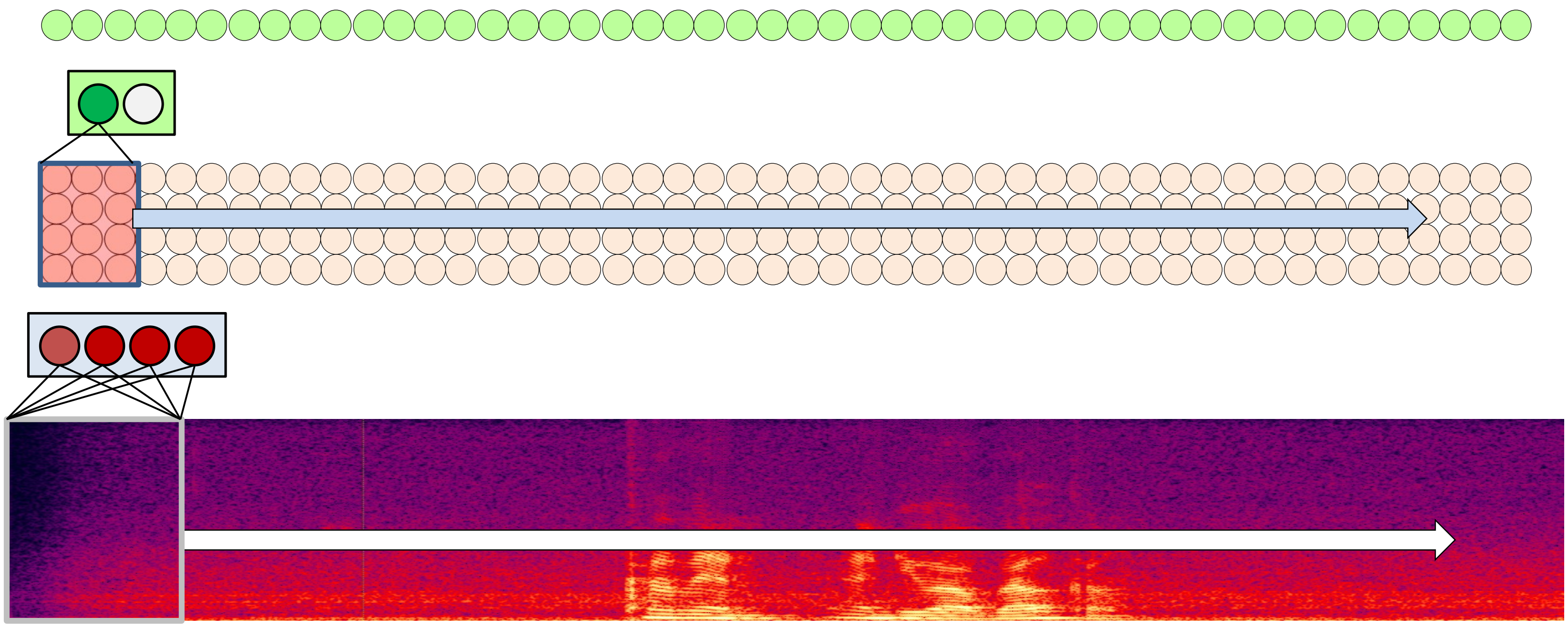
- The 1-D scan version of the convolutional neural network

1-D scan version



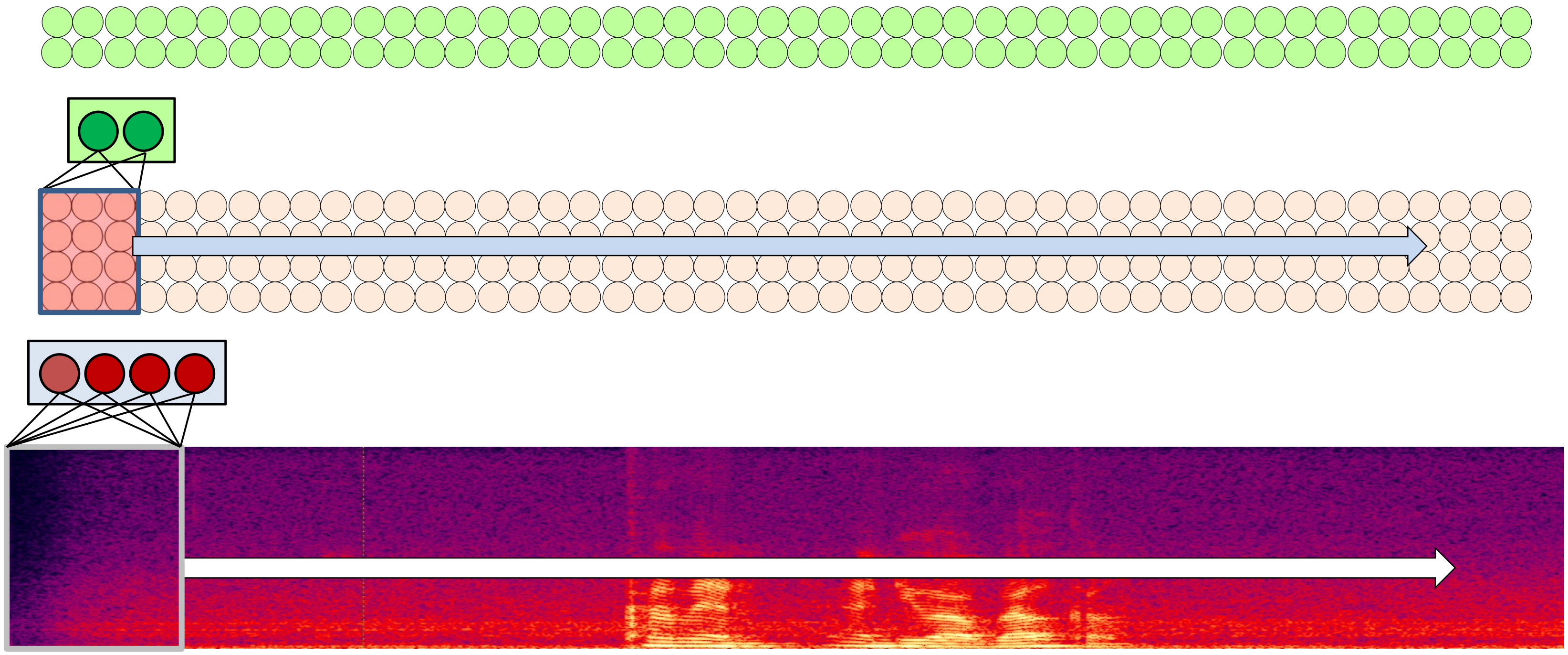
- The 1-D scan version of the convolutional neural network
  - Max pooling optional
    - Not generally done for speech

1-D scan version



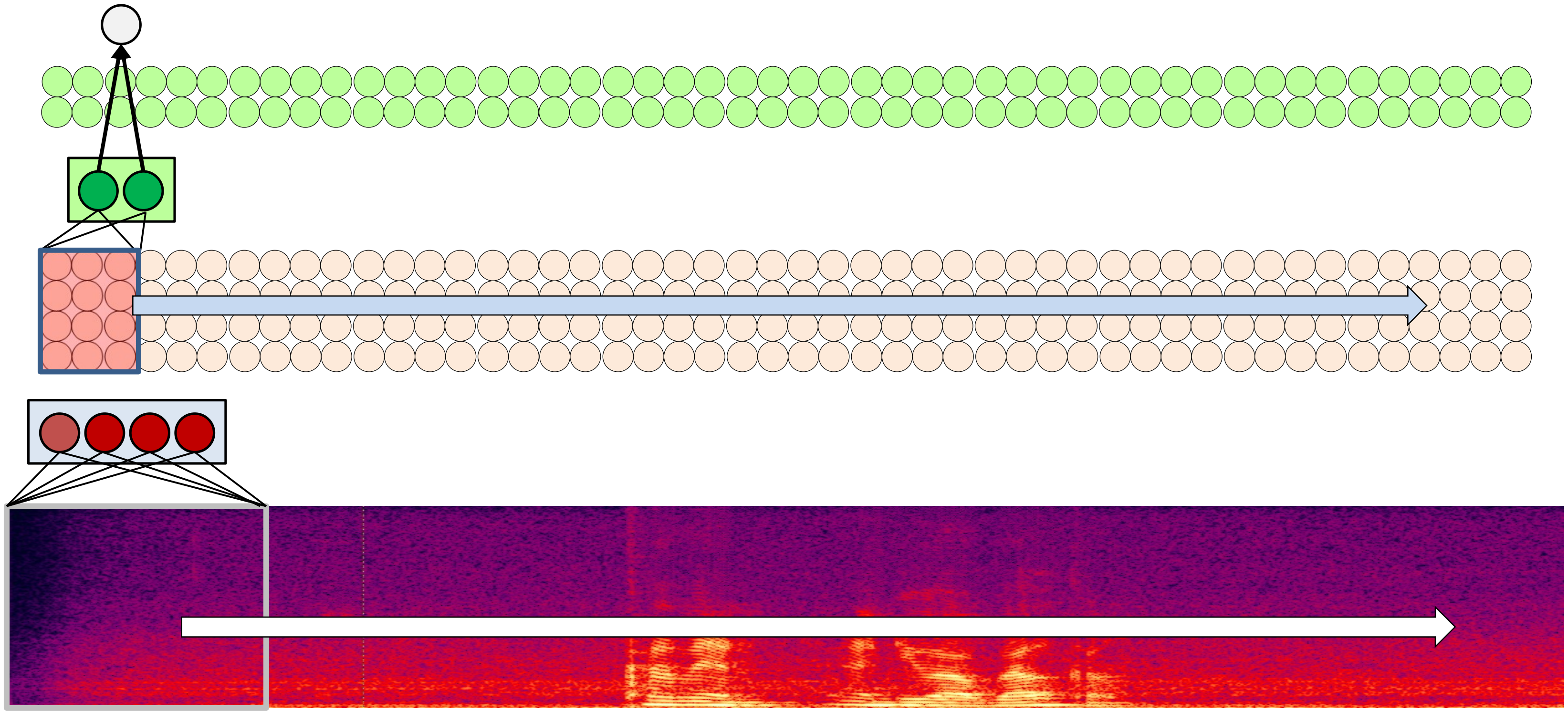
- The 1-D scan version of the convolutional neural network
  - Max pooling optional
    - Not generally done for speech

1-D scan version



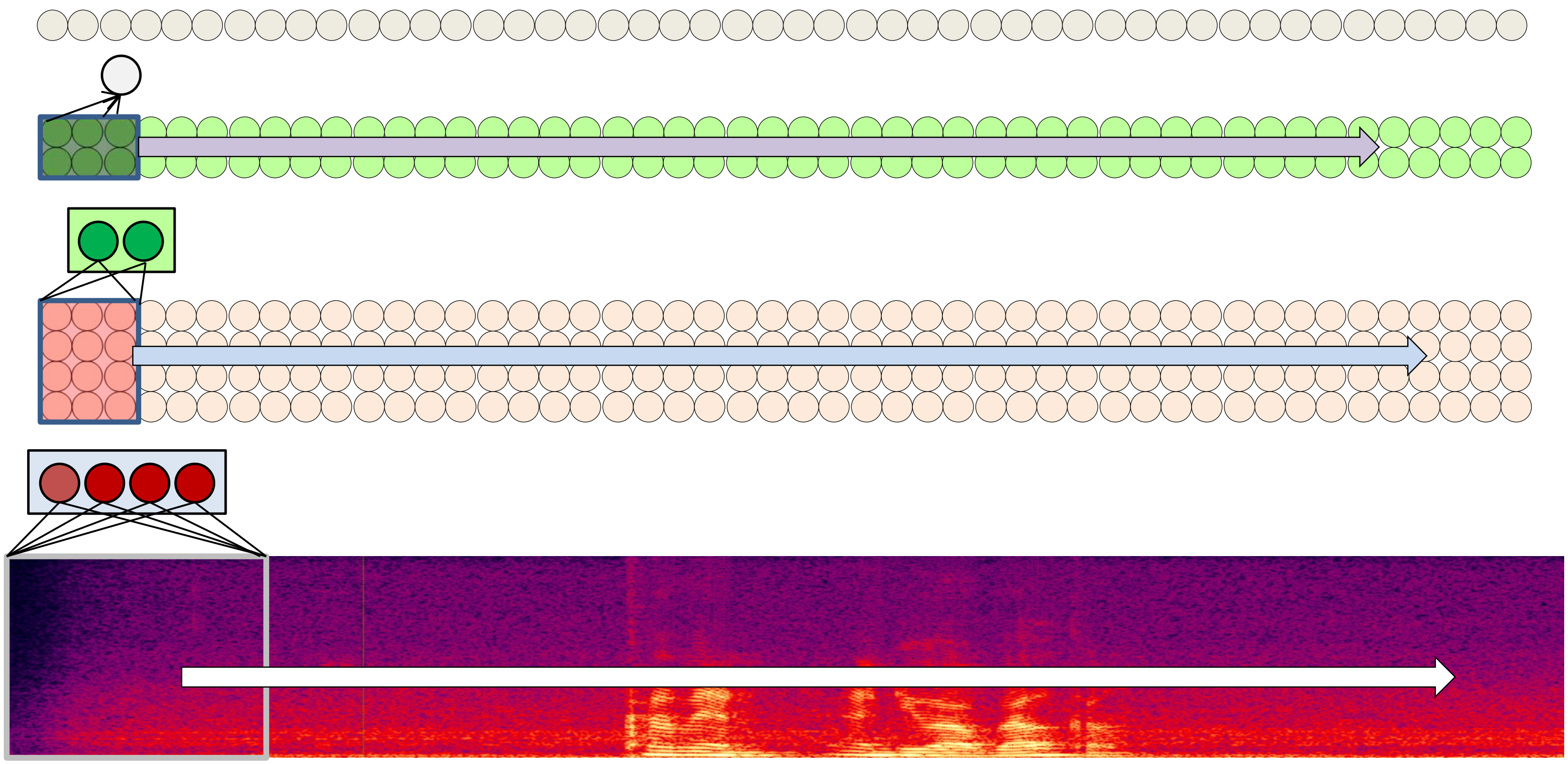
- The 1-D scan version of the convolutional neural network
  - Max pooling optional
    - Not generally done for speech

1-D scan version



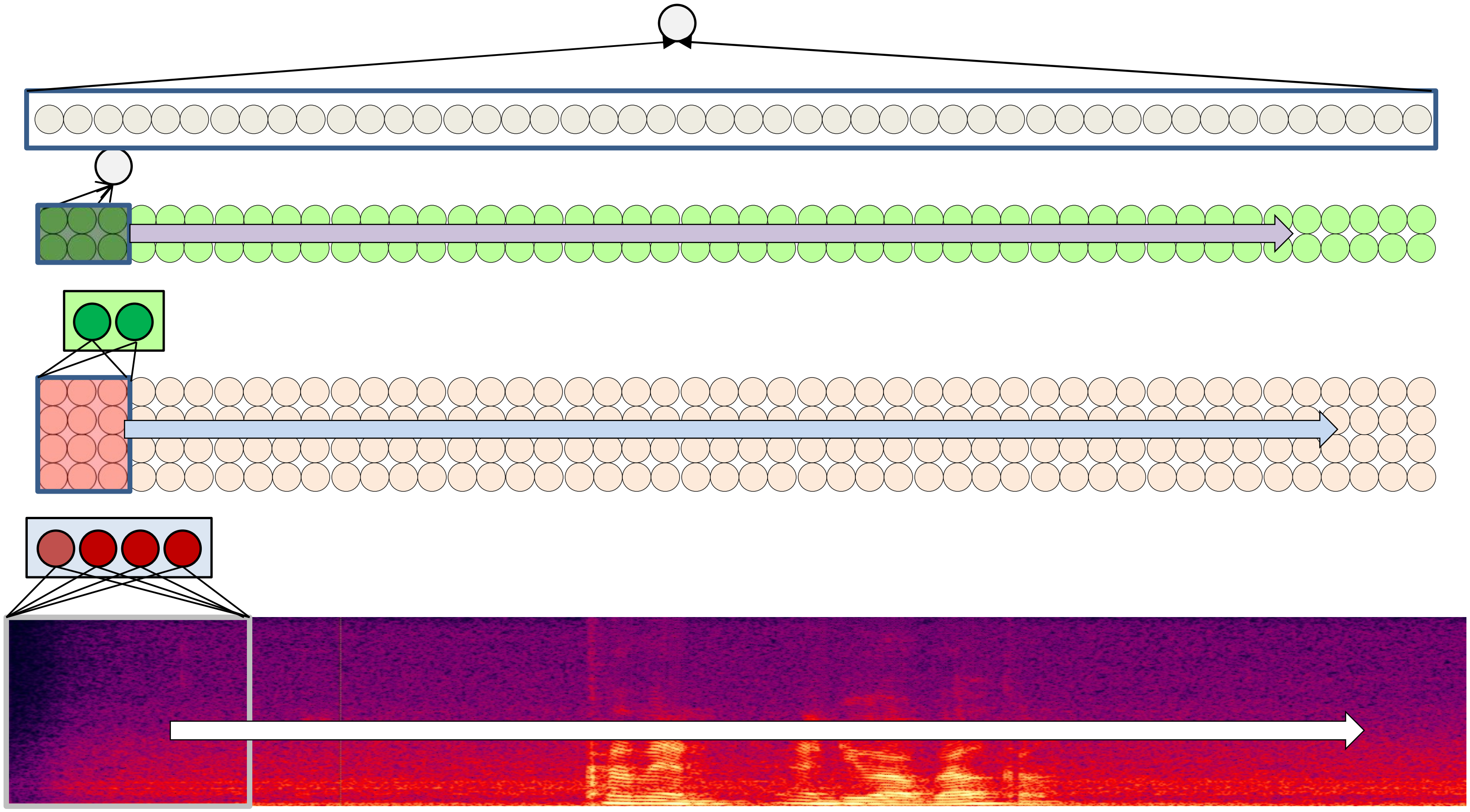
- The 1-D scan version of the convolutional neural network
  - Max pooling optional
    - Not generally done for speech

1-D scan version



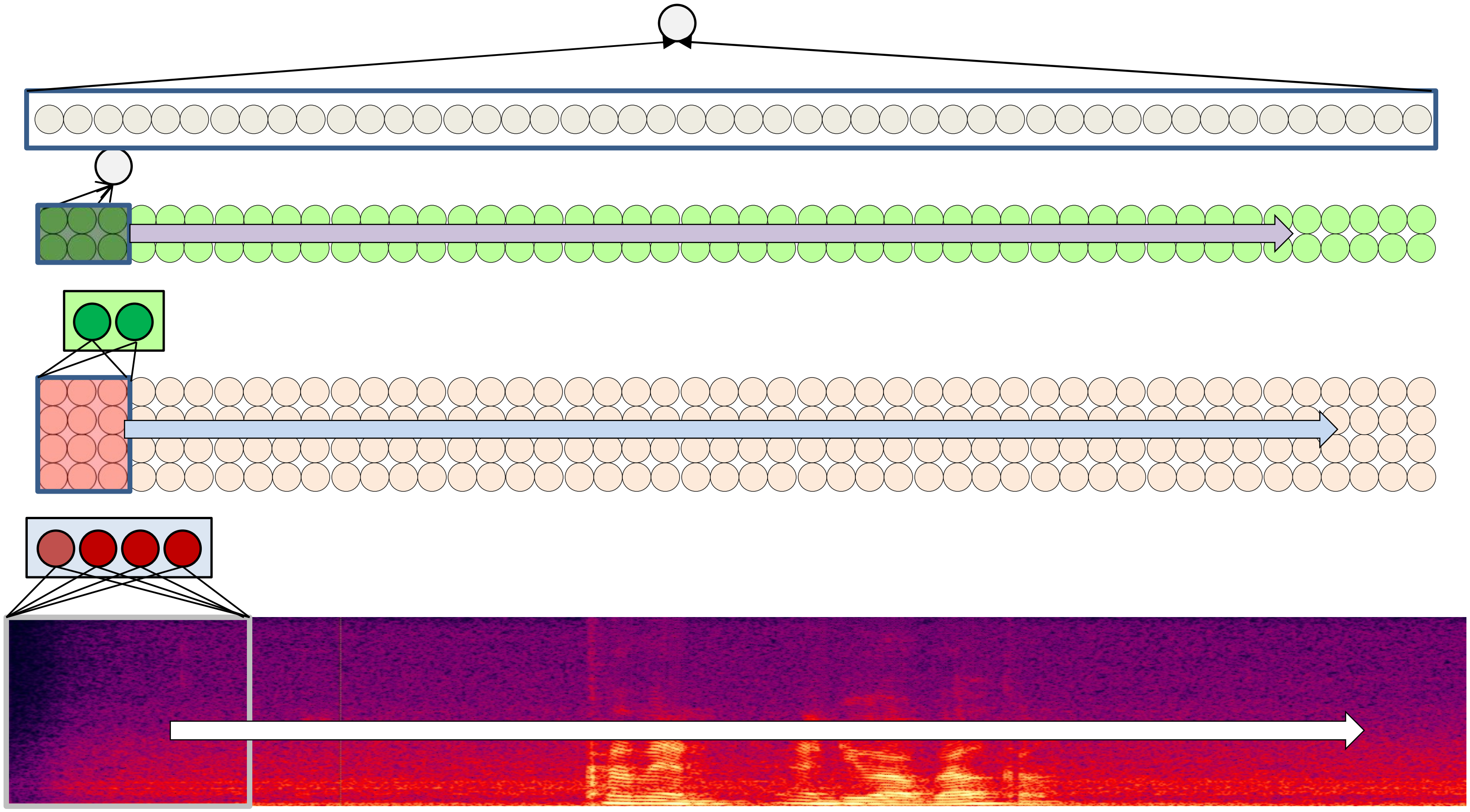
- The 1-D scan version of the convolutional neural network
  - Max pooling optional
    - Not generally done for speech

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
- The scanned "block" can be distributed over multiple layers for efficiency
- 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