

# **Deep Neural Networks**

## **Scanning for patterns**

### **(aka convolutional networks)**

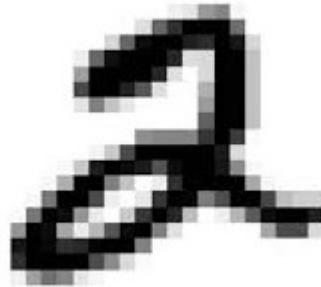
Bhiksha Raj

11-785, Spring 2020

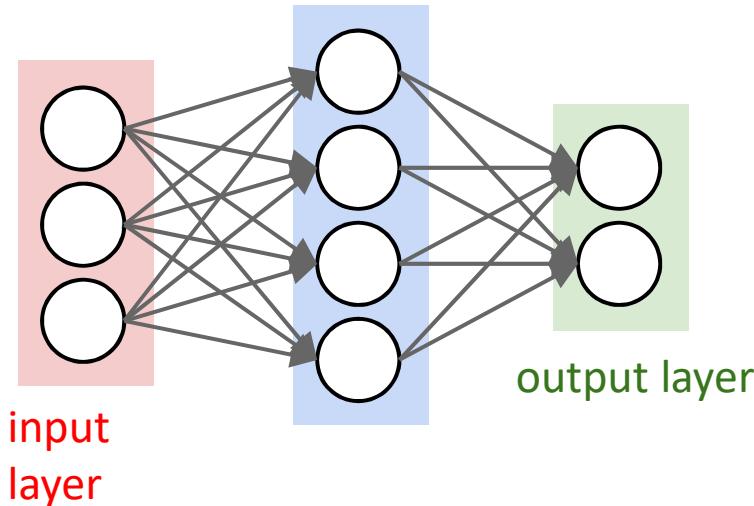
# **Story so far**

- **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

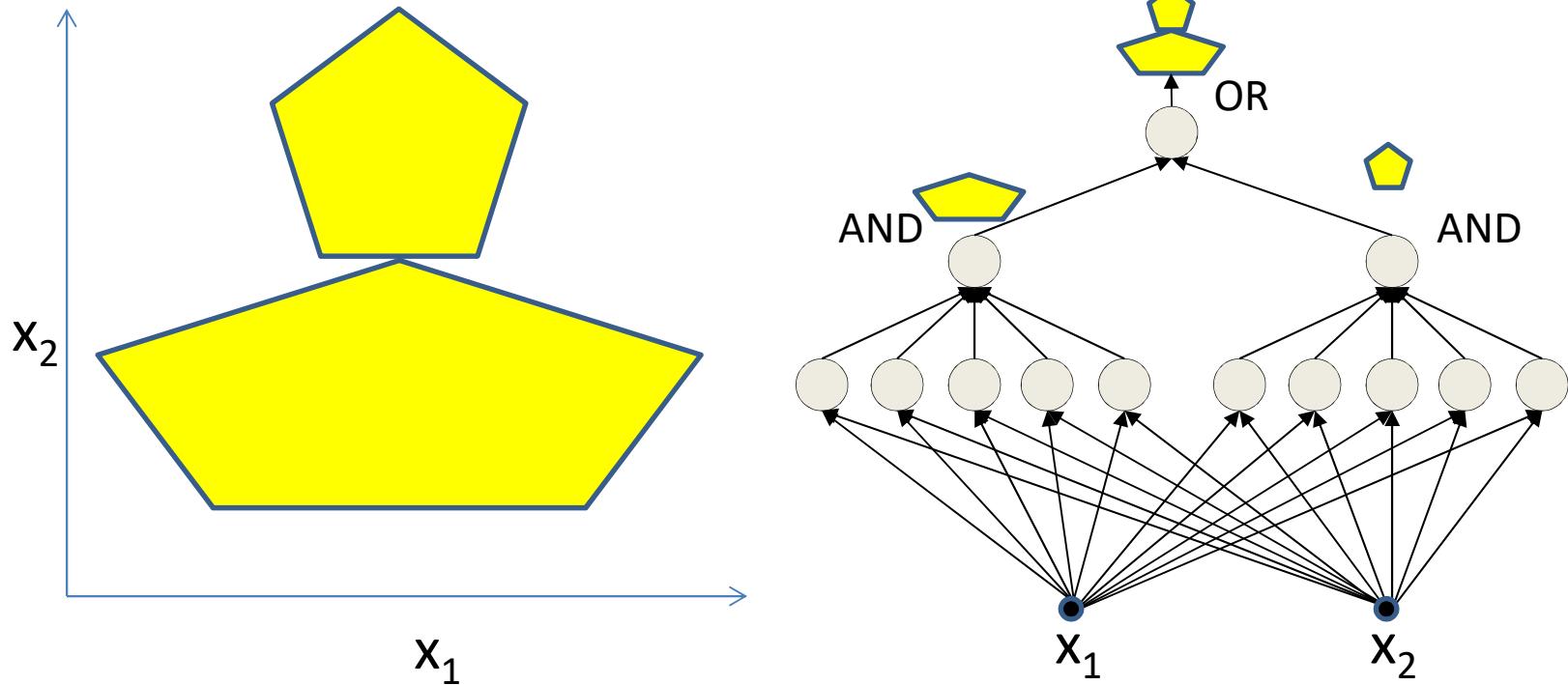


Or, more generally  
a vector input



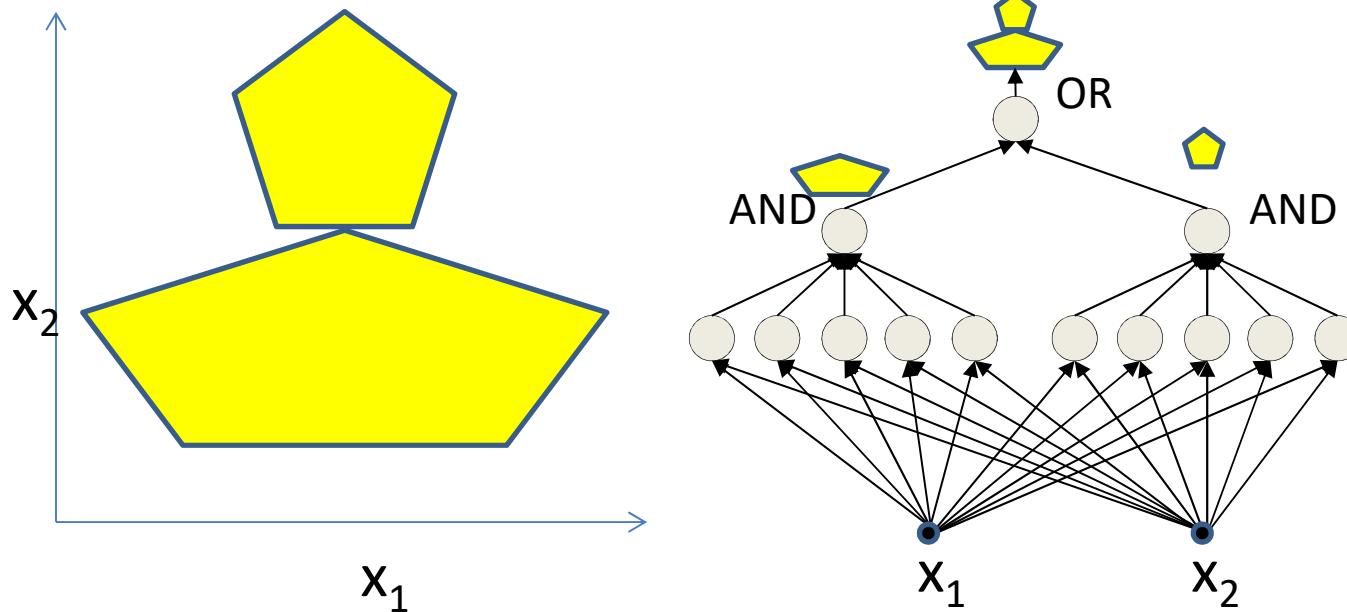
- Can recognize patterns in data
  - E.g. digits
  - Or any other vector data

# An important observation



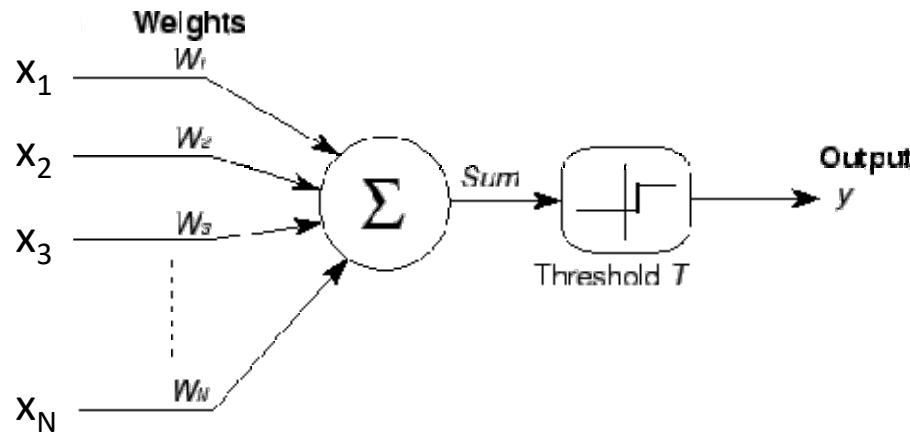
- The lowest layers of the network capture simple patterns
  - The linear decision boundaries in this example
- The next layer captures more complex patterns
  - The polygons
- The next one captures still more complex patterns..

# An important observation



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

# What do the neurons capture?

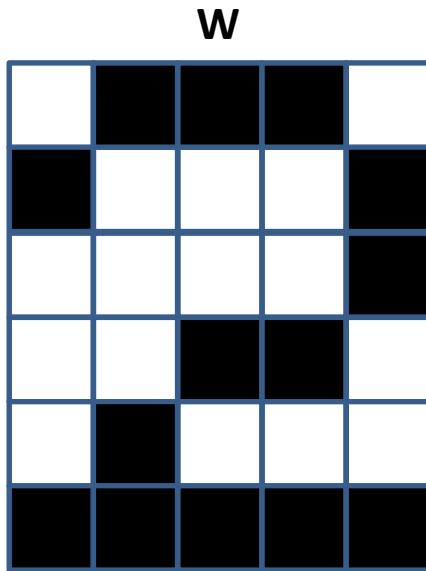


$$y = \begin{cases} 1 & \text{if } \sum_i w_i x_i \geq T \\ 0 & \text{else} \end{cases}$$

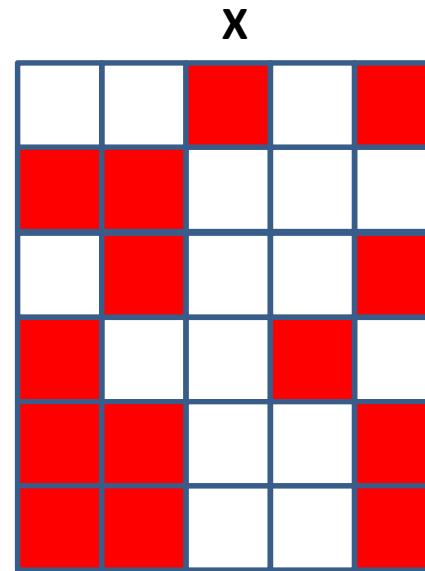
$$y = \begin{cases} 1 & \text{if } \mathbf{x}^T \mathbf{w} \geq T \\ 0 & \text{else} \end{cases}$$

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

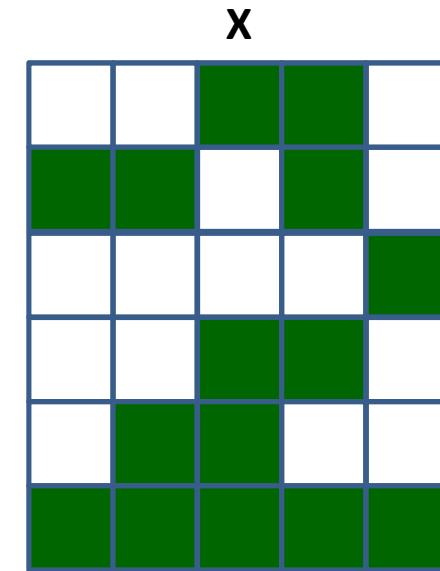
# The weights as a correlation filter



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



Correlation = 0.57

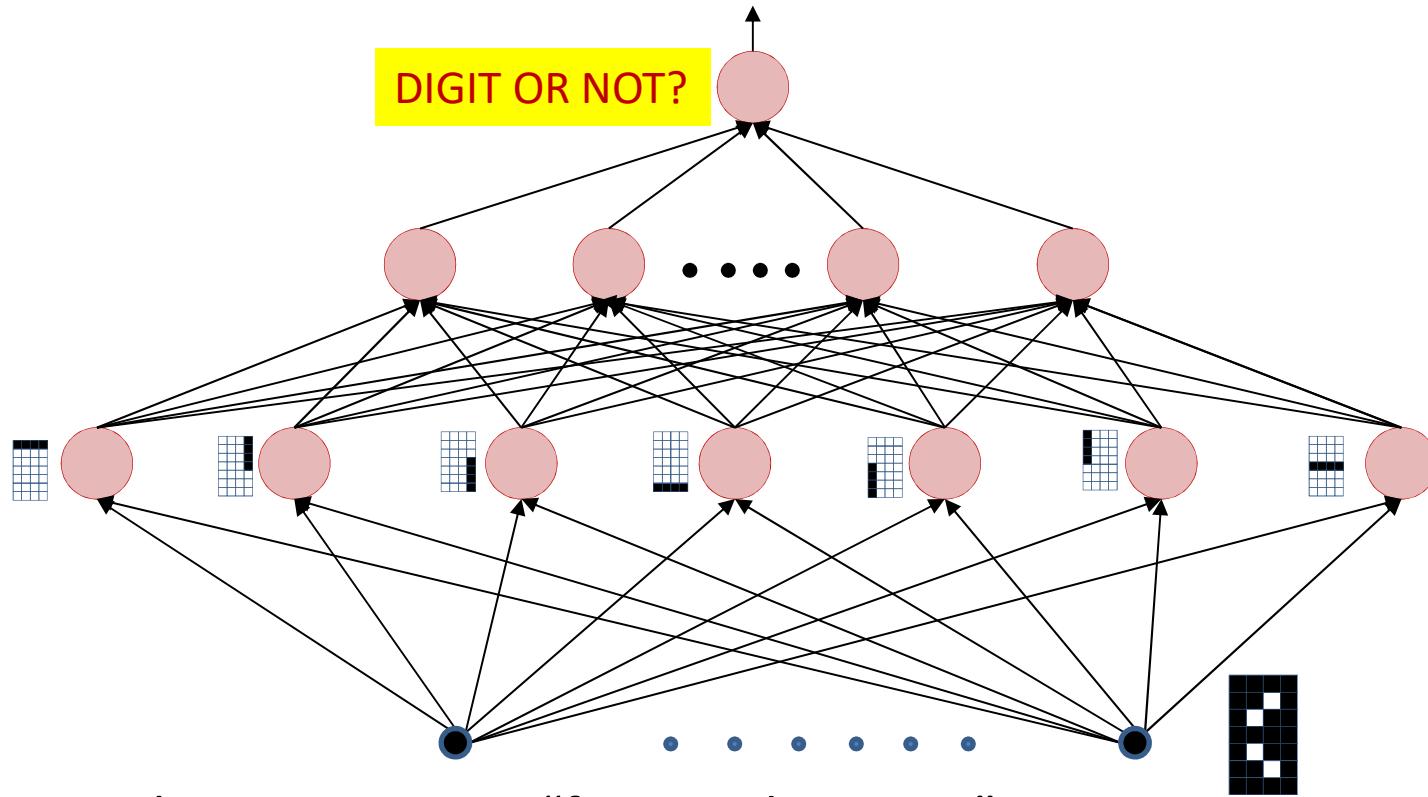


Correlation = 0.82



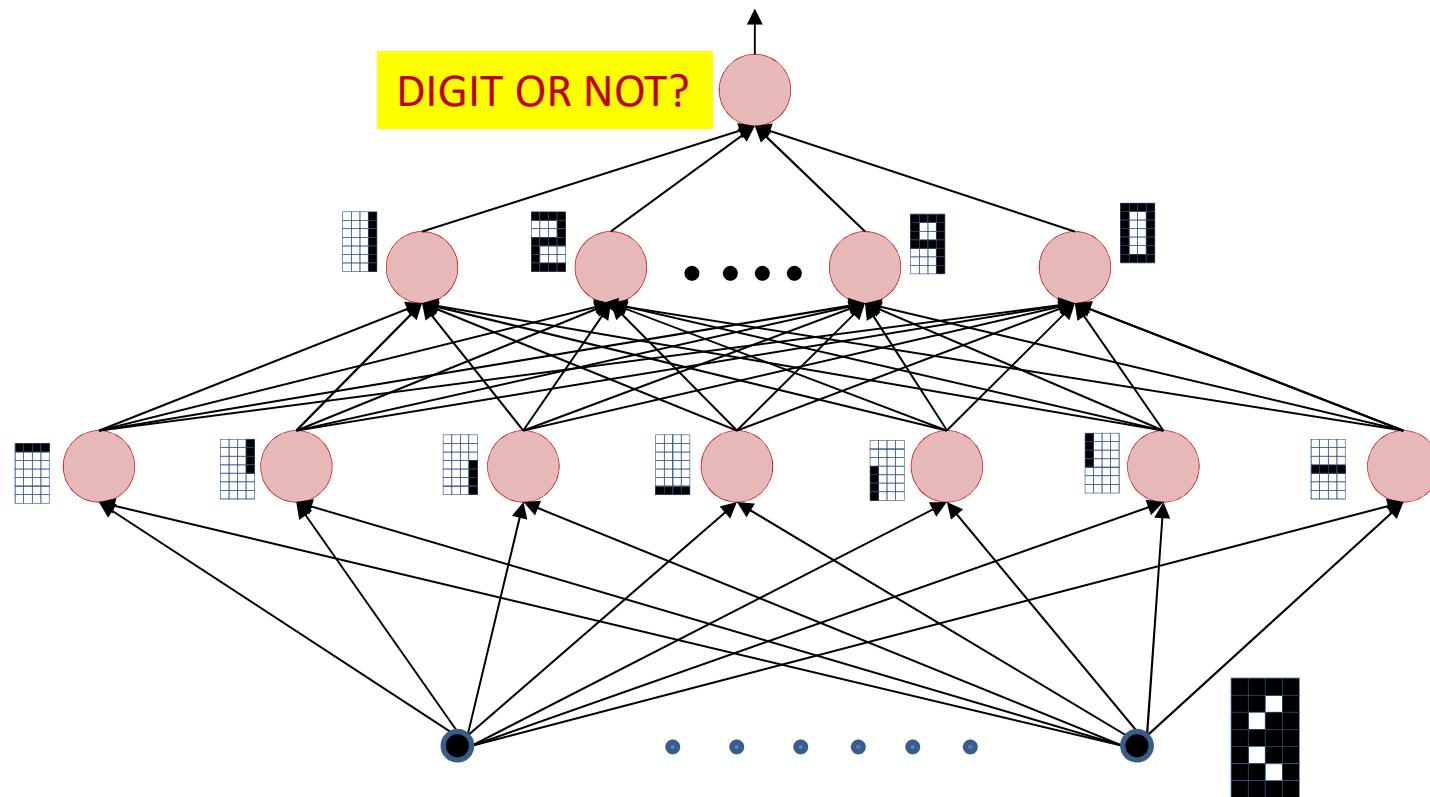
- The green pattern looks more like the weights pattern (black) than the red pattern
  - The green pattern is more *correlated* with the weights

# The MLP as a function over feature detectors



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

# Distributed representations: The MLP as a cascade of feature detectors



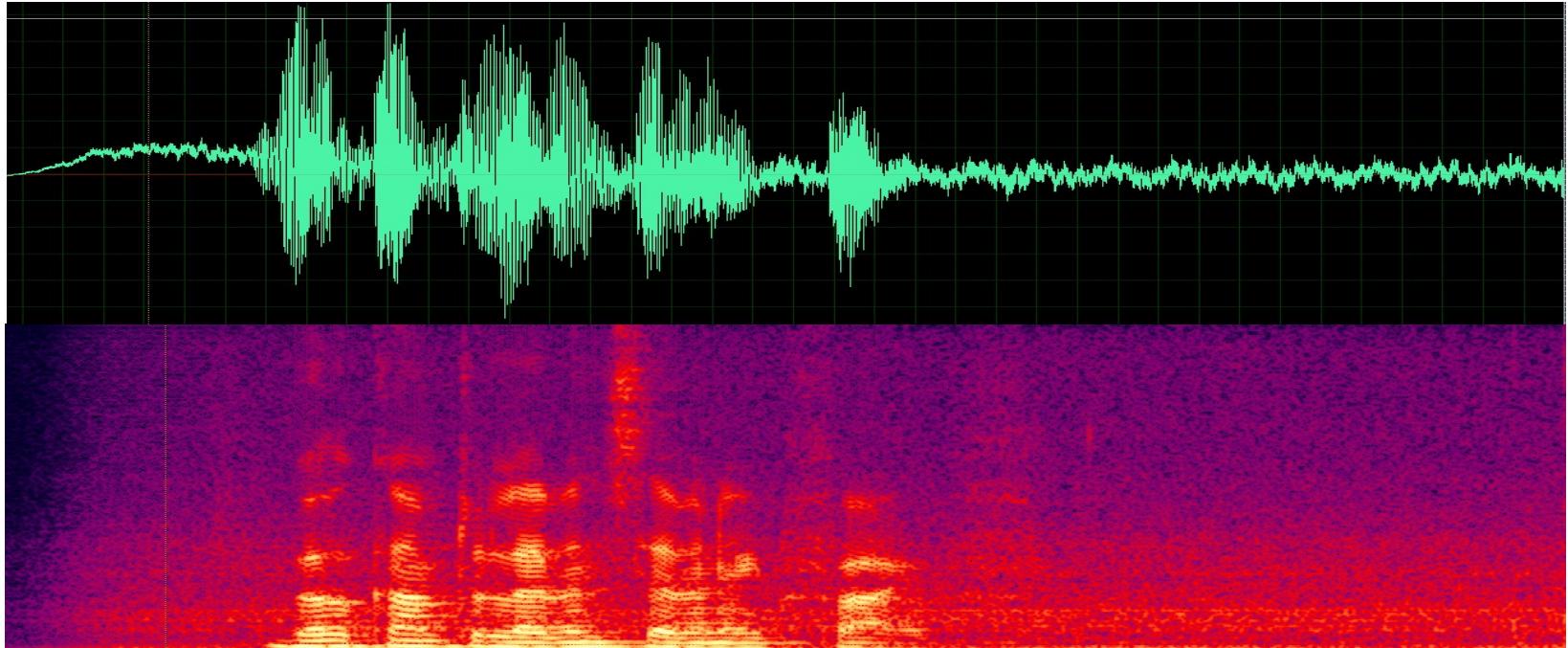
- The network is a cascade of feature detectors
  - Higher level neurons compose complex templates from features represented by lower-level neurons

# Story so far

- Perceptrons are correlation filters
  - They detect patterns in the input
- Layers in an MLP are detectors of increasingly complex patterns
  - Patterns of lower-complexity patterns
  - The representation of “acceptable” input patterns is distributed over the layers of the network
- MLP in classification
  - The network will fire if the combination of the detected basic features matches an “acceptable” pattern for a desired class of signal
    - E.g. Appropriate combinations of (Nose, Eyes, Eyebrows, Cheek, Chin) → Face
  - If the final complex pattern detected “matches” a desired pattern

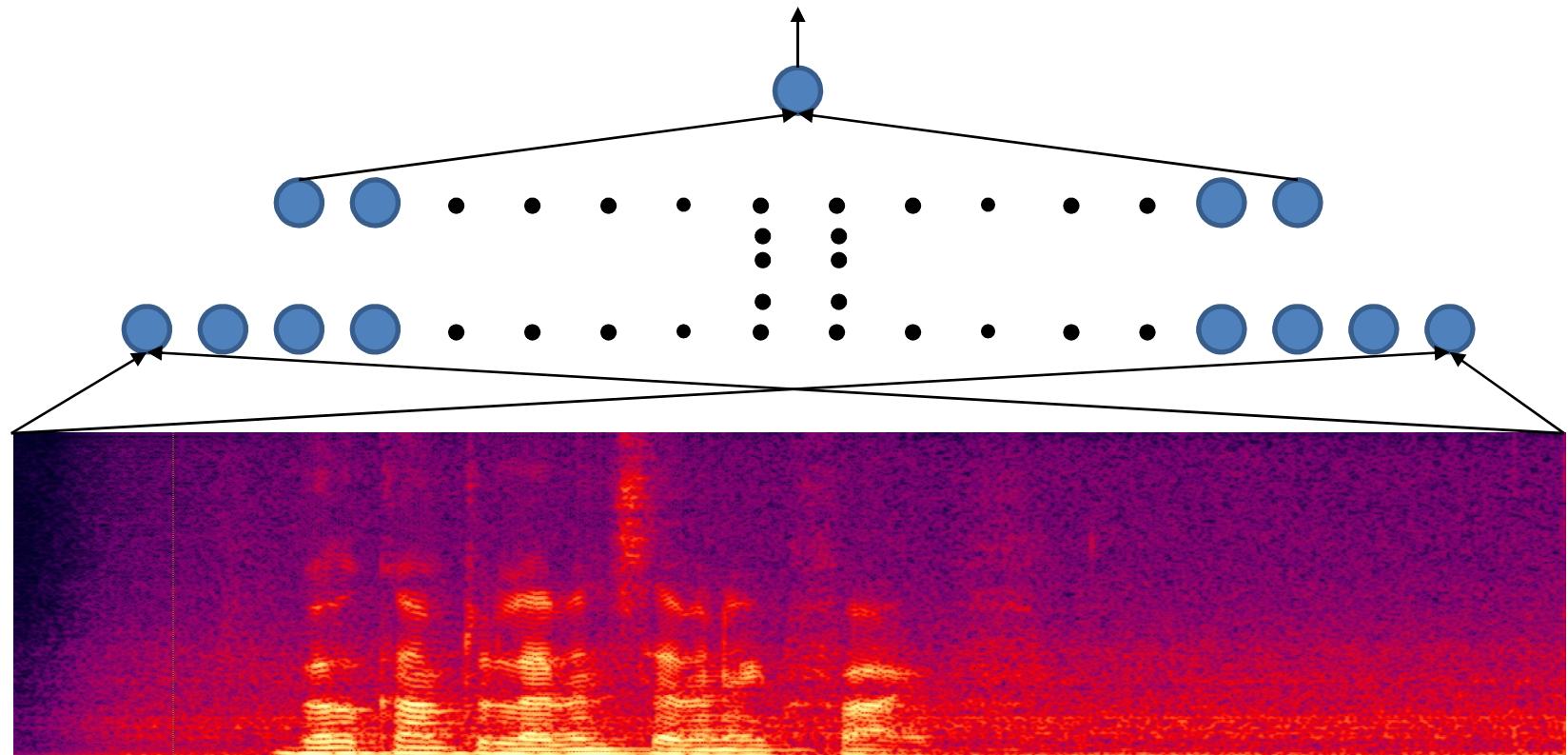
# Changing gears..

# A problem



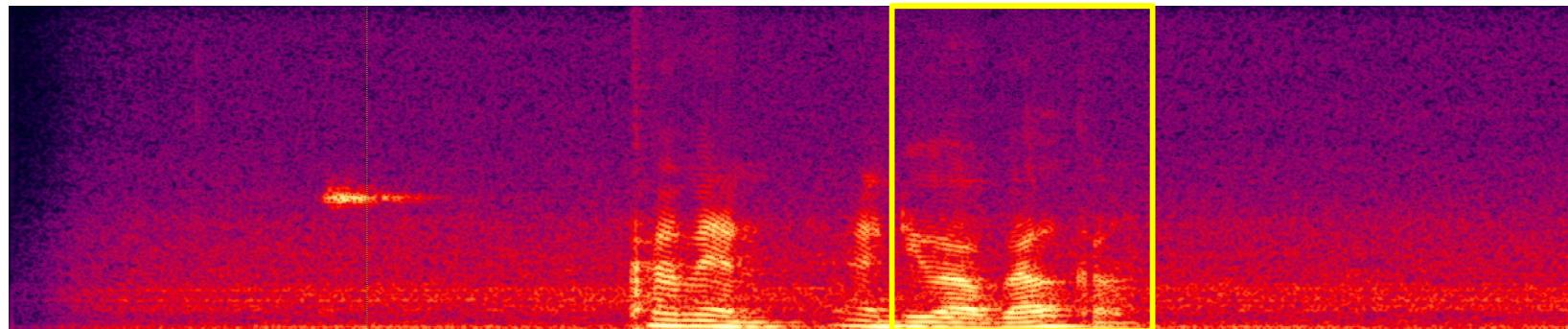
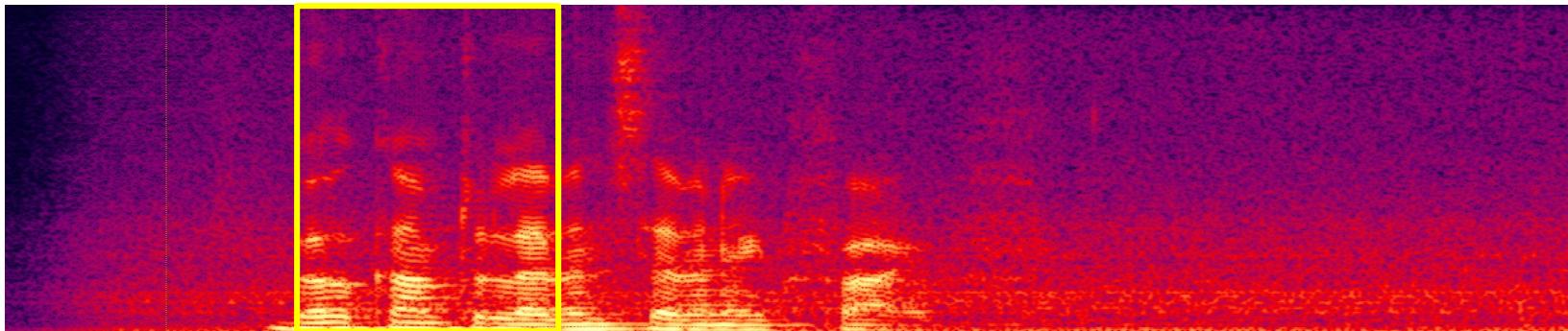
- Does this signal contain the word “Welcome”?
- Compose an MLP for this problem.
  - Assuming all recordings are exactly the same length..

# Finding a Welcome



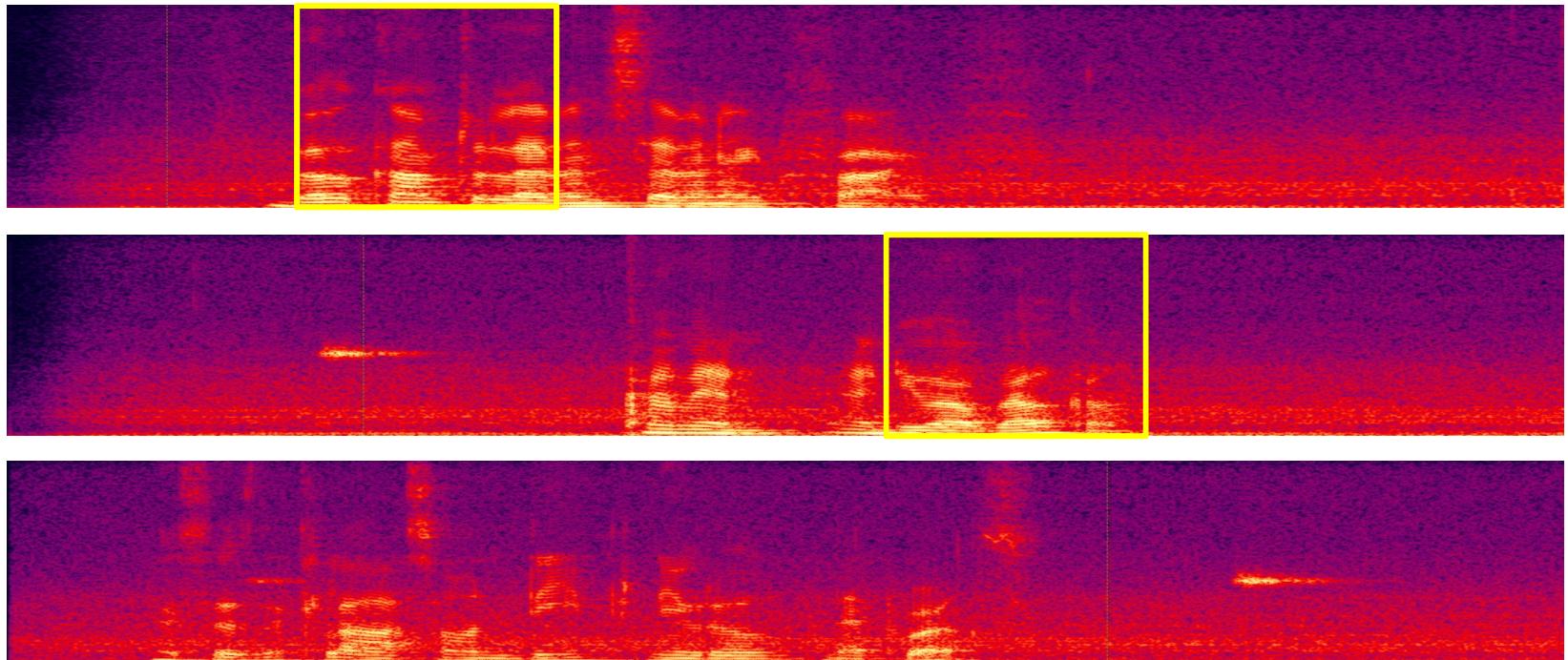
- Trivial solution: Train an MLP for the entire recording

# Finding a Welcome



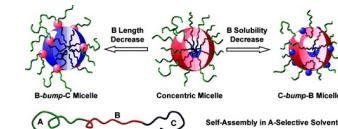
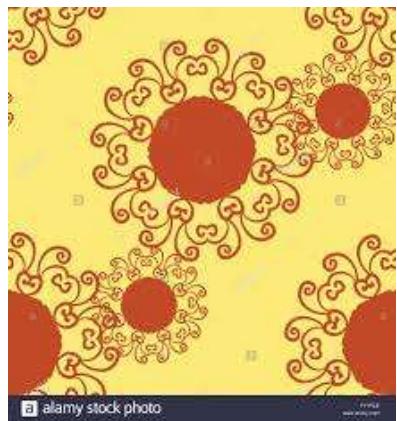
- Problem with trivial solution: Network that finds a “welcome” in the top recording will not find it in the lower one
  - Unless trained with both
  - Will require a very large network and a large amount of training data to cover every case

# Finding a Welcome



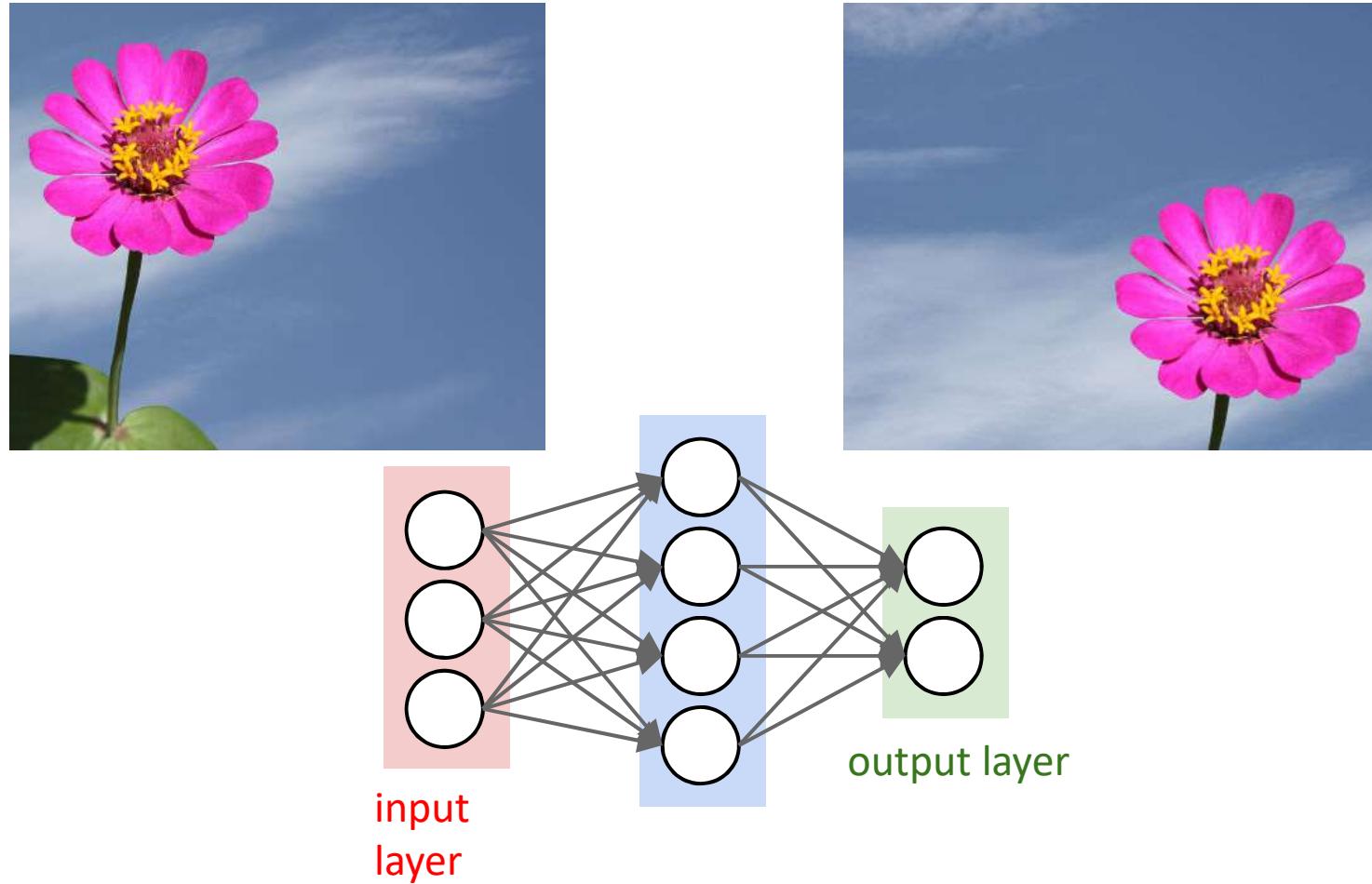
- Need a *simple* network that will fire regardless of the location of “Welcome”
  - and not fire when there is none

# Flowers



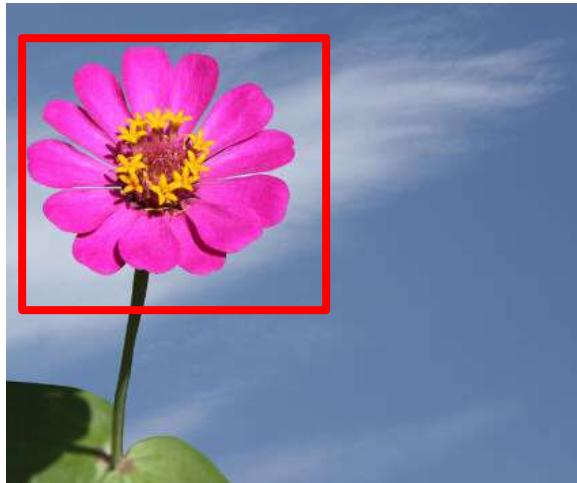
- Is there a flower in any of these images

# A problem



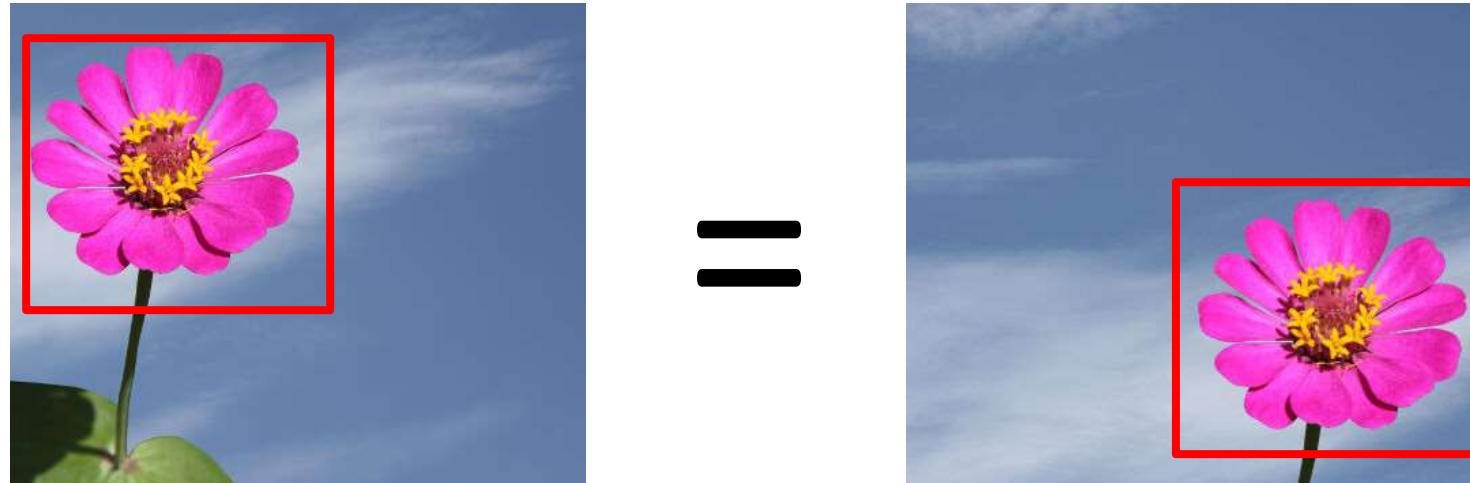
- Will an MLP that recognizes the left image as a flower also recognize the one on the right as a flower?

# A problem



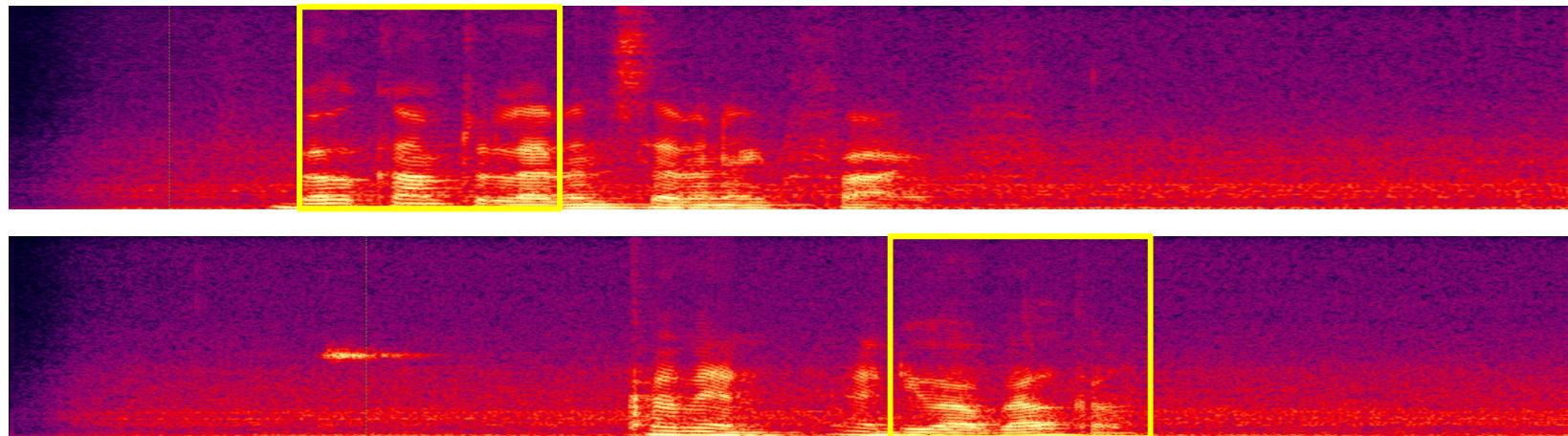
- Need a network that will “fire” regardless of the precise location of the target object

# The need for *shift invariance*



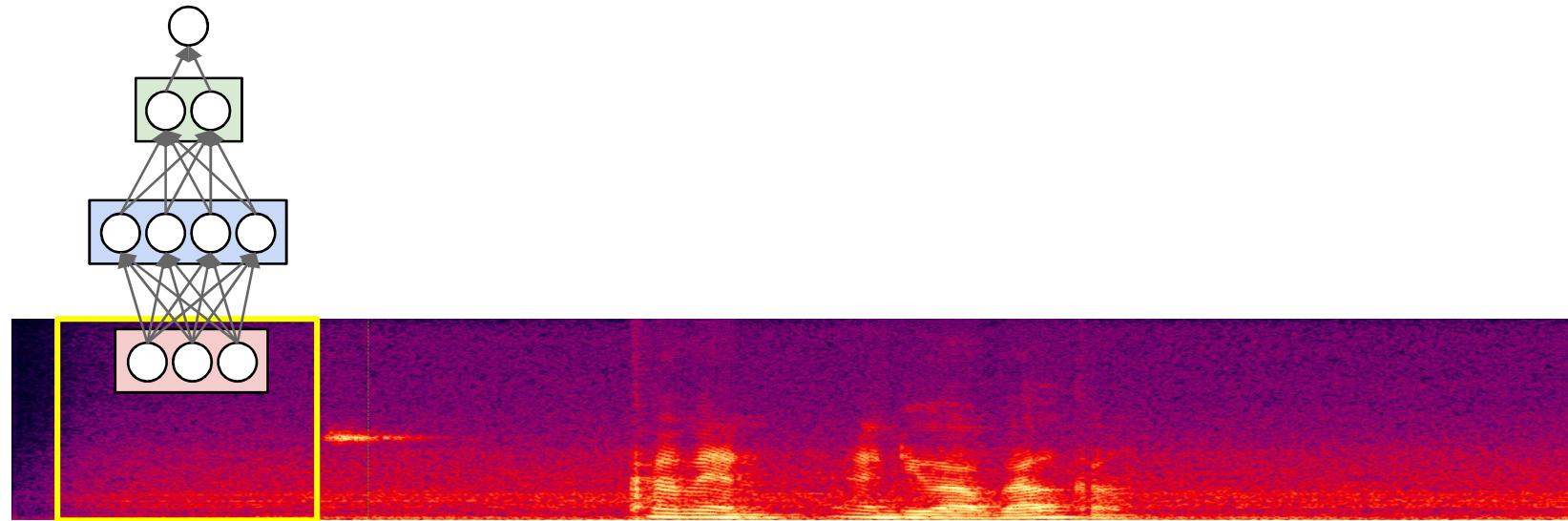
- In many problems the *location* of a pattern is not important
  - Only the presence of the pattern
- Conventional MLPs are sensitive to the location of the pattern
  - Moving it by one component results in an entirely different input that the MLP wont recognize
- Requirement: Network must be *shift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

# The need for *shift invariance*



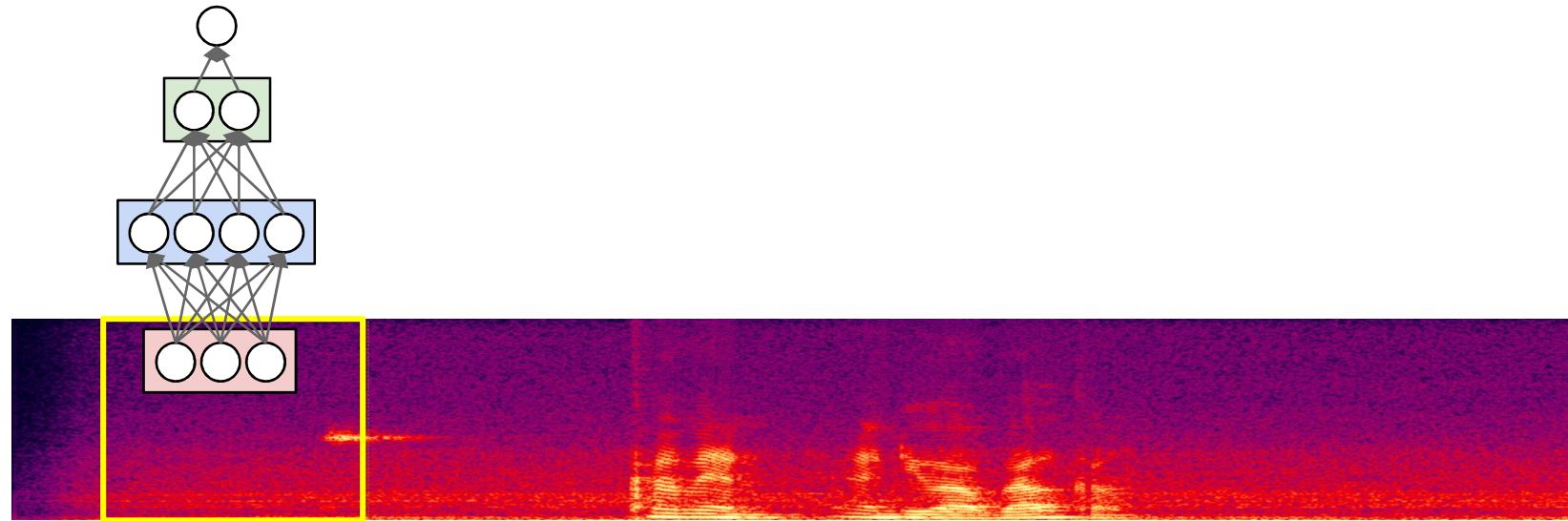
- In many problems the *location* of a pattern is not important
  - Only the presence of the pattern
- Conventional MLPs are sensitive to the location of the pattern
  - Moving it by one component results in an entirely different input that the MLP wont recognize
- Requirement: Network must be *shift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

# Solution: Scan



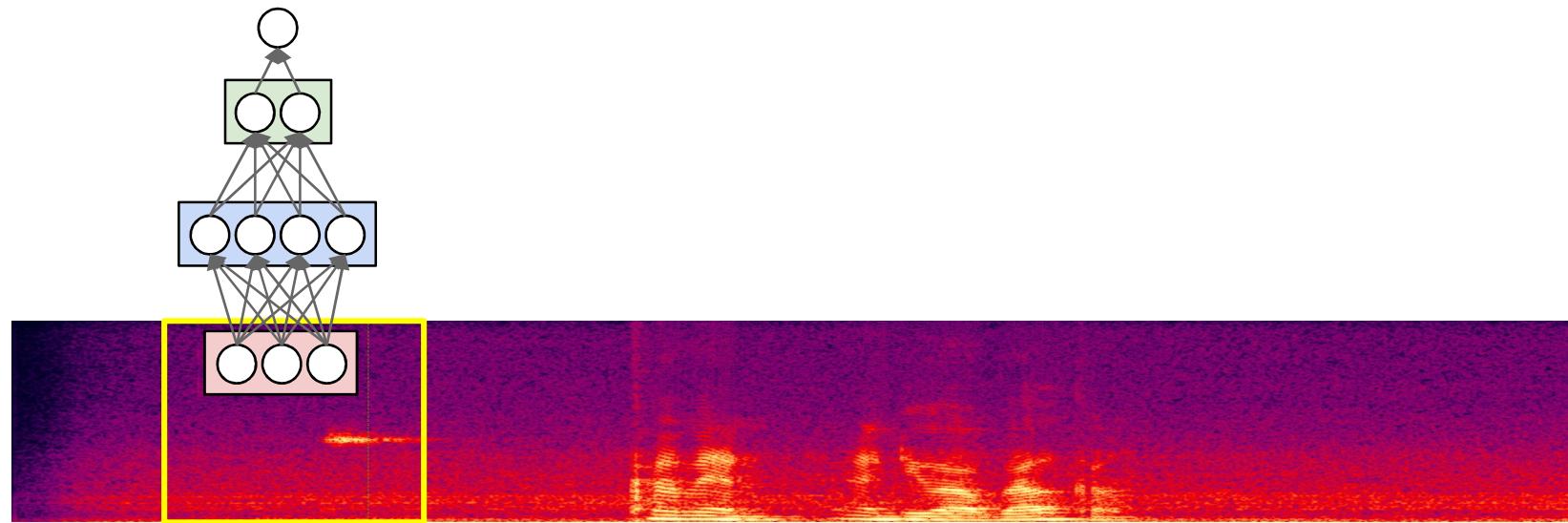
- Scan for the target word
  - The spectral time-frequency components in a “window” are input to a “welcome-detector” MLP

# Solution: Scan



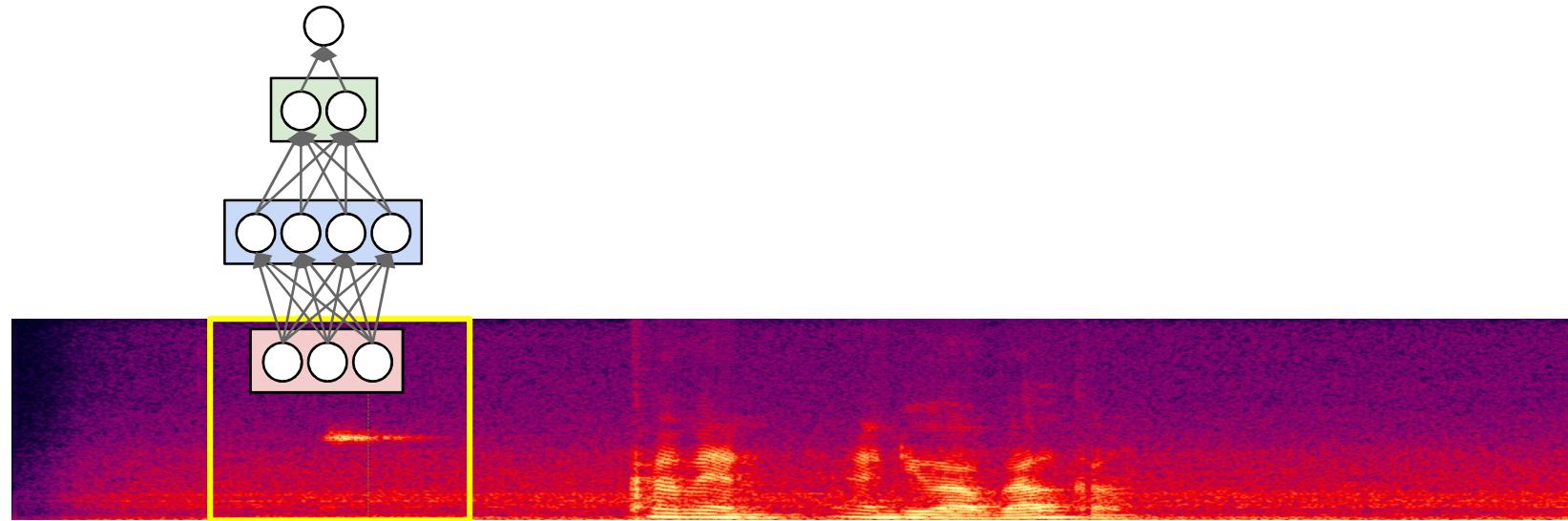
- Scan for the target word
  - The spectral time-frequency components in a “window” are input to a “welcome-detector” MLP

# Solution: Scan



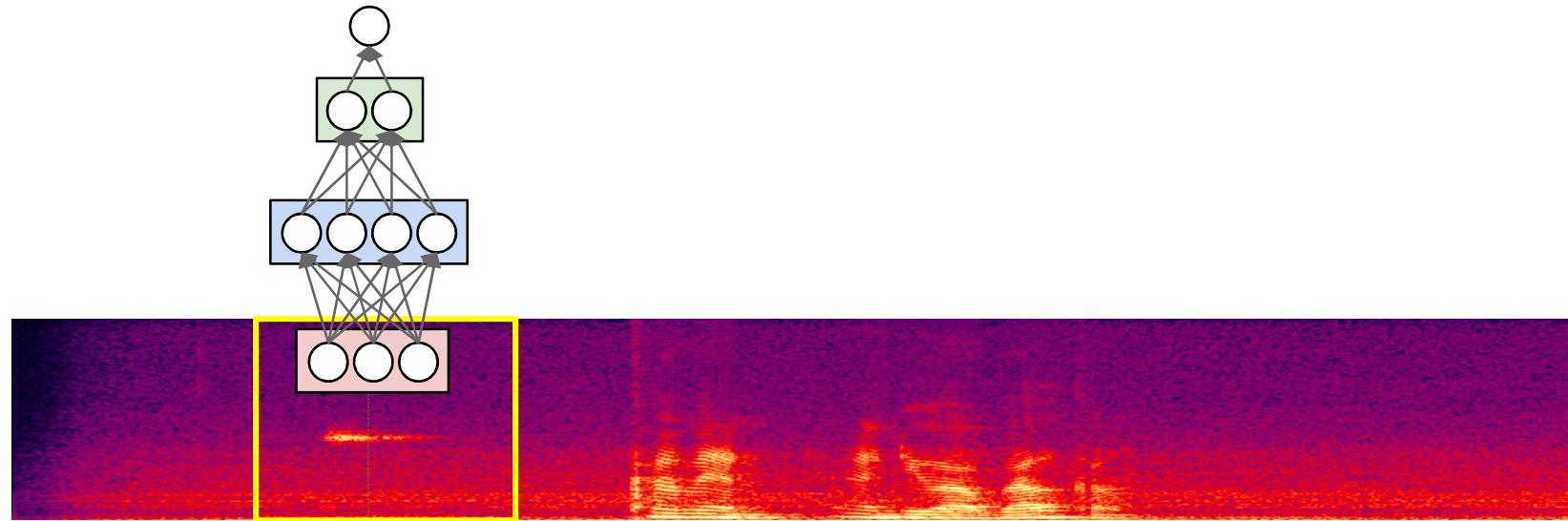
- Scan for the target word
  - The spectral time-frequency components in a “window” are input to a “welcome-detector” MLP

# Solution: Scan



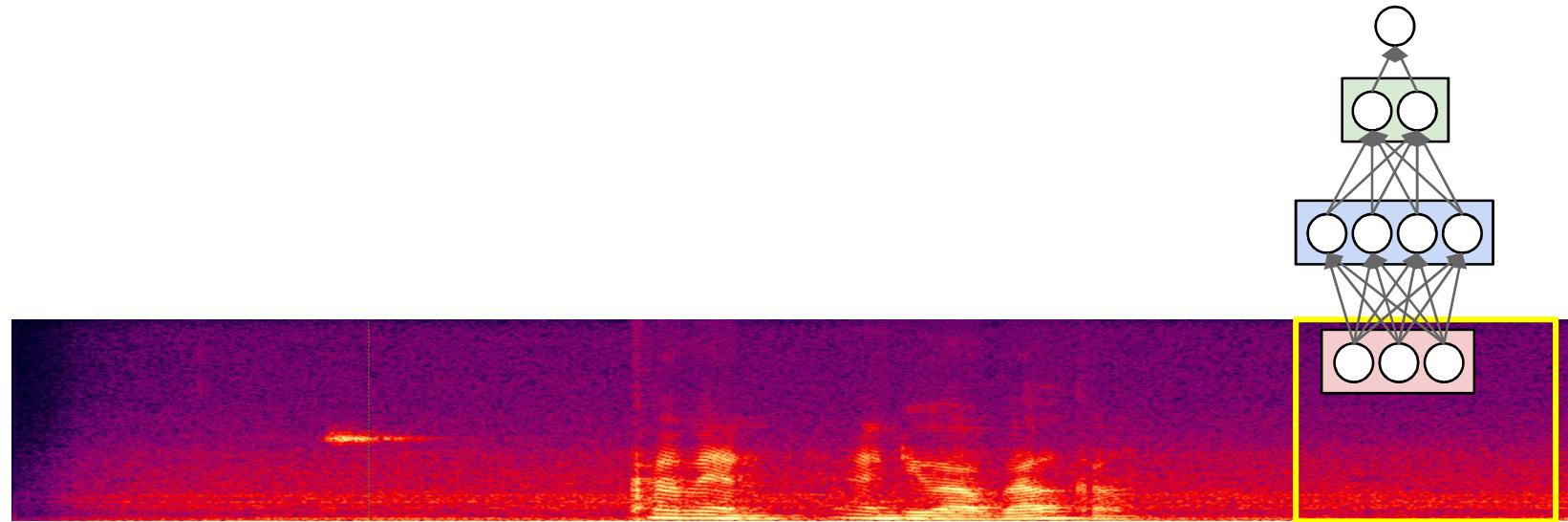
- Scan for the target word
  - The spectral time-frequency components in a “window” are input to a “welcome-detector” MLP

# Solution: Scan



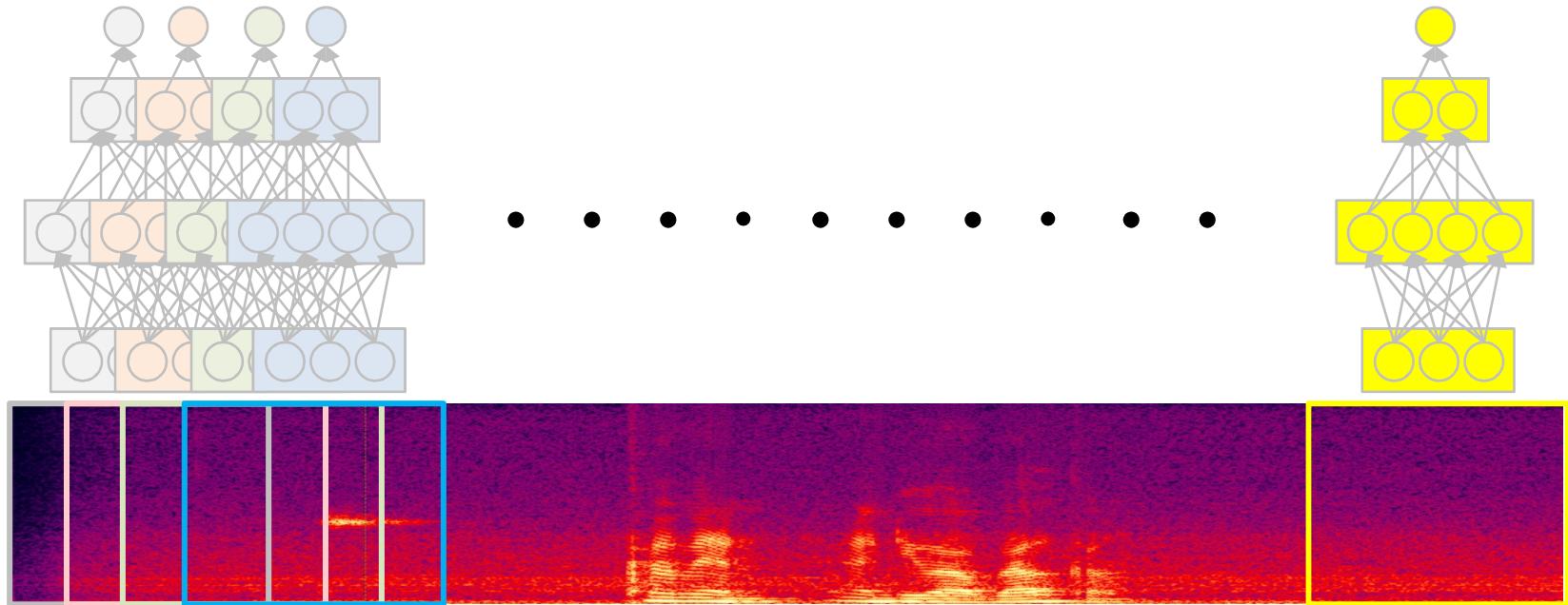
- Scan for the target word
  - The spectral time-frequency components in a “window” are input to a “welcome-detector” MLP

# Solution: Scan



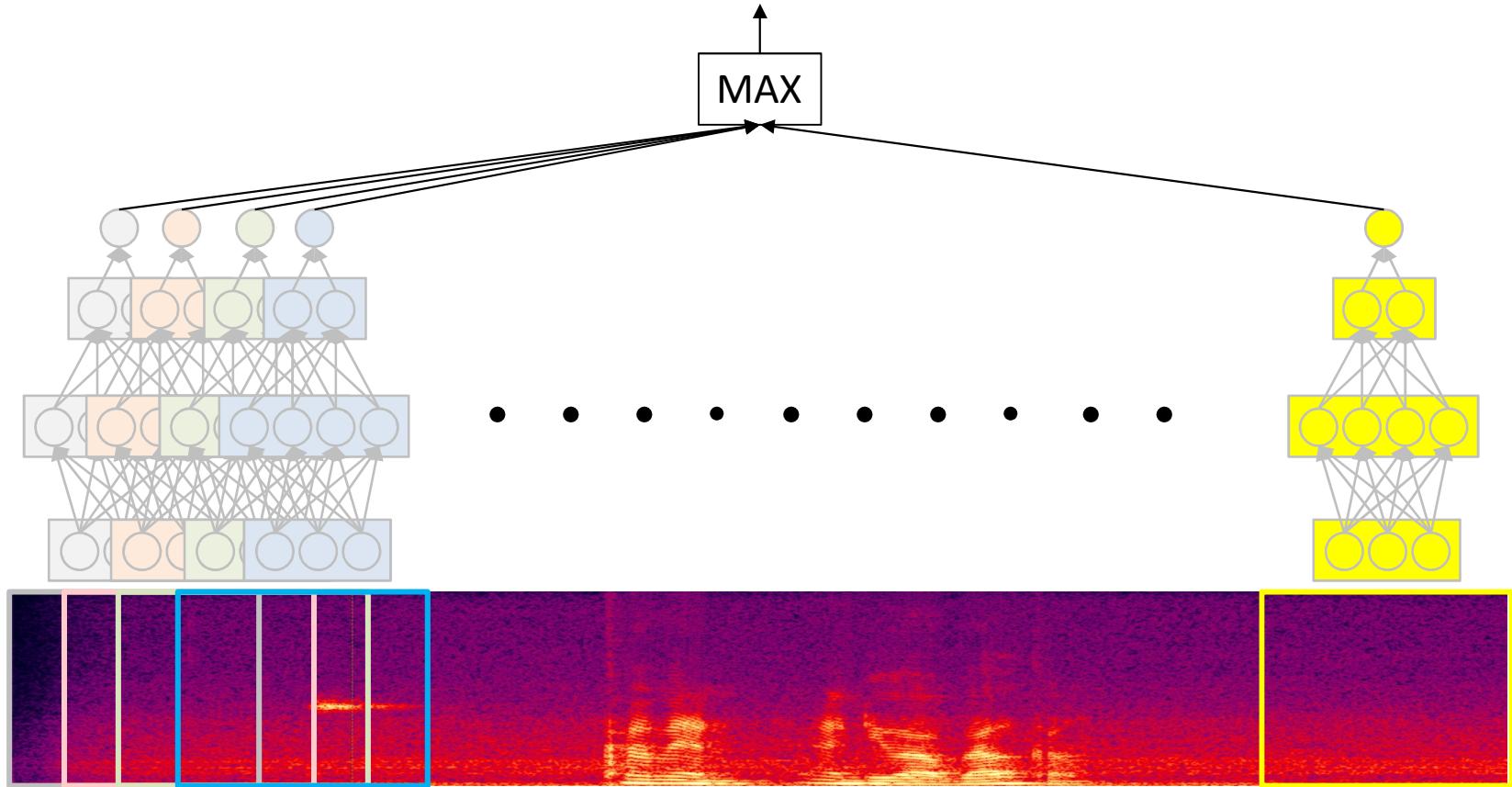
- Scan for the target word
  - The spectral time-frequency components in a “window” are input to a “welcome-detector” MLP

# Solution: Scan



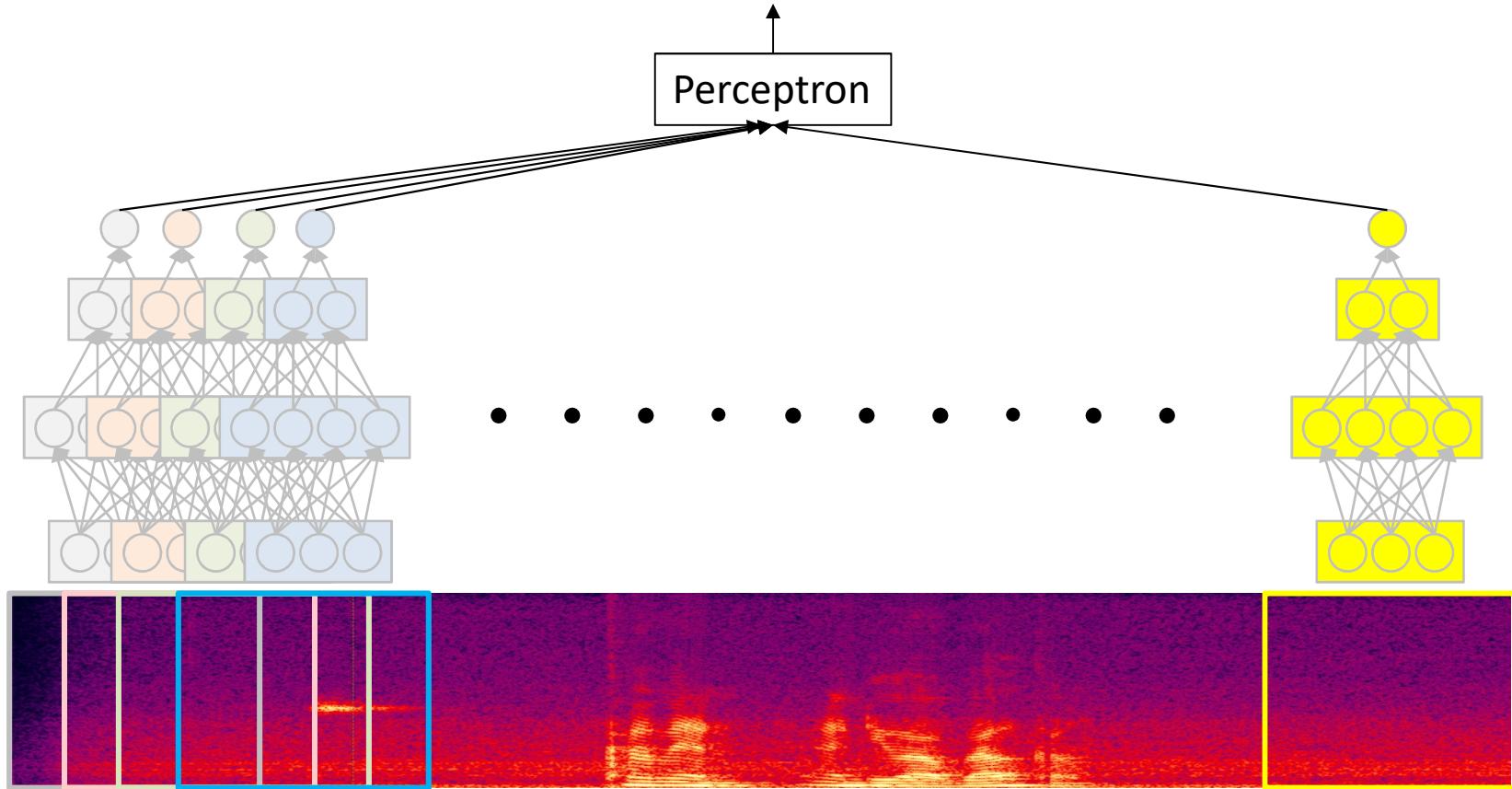
- “Does welcome occur in this recording?”
  - We have classified many “windows” individually
  - “Welcome” may have occurred in any of them

# Solution: Scan



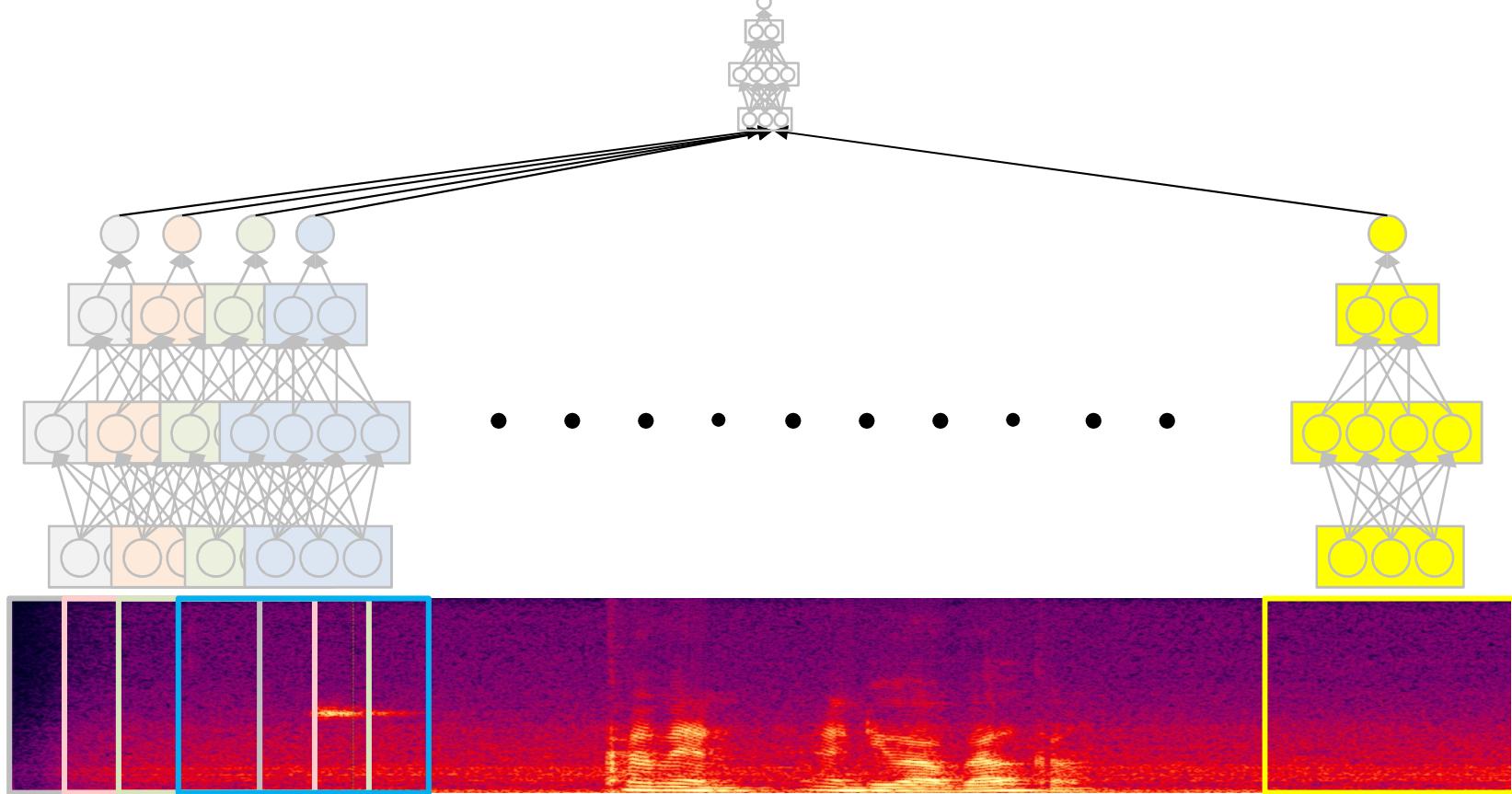
- “Does welcome occur in this recording?”
  - Maximum of all the outputs (Equivalent of Boolean OR)

# Solution: Scan



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

# Solution: Scan



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

# Scanning with an MLP

- $K$  = width of “patch” evaluated by MLP

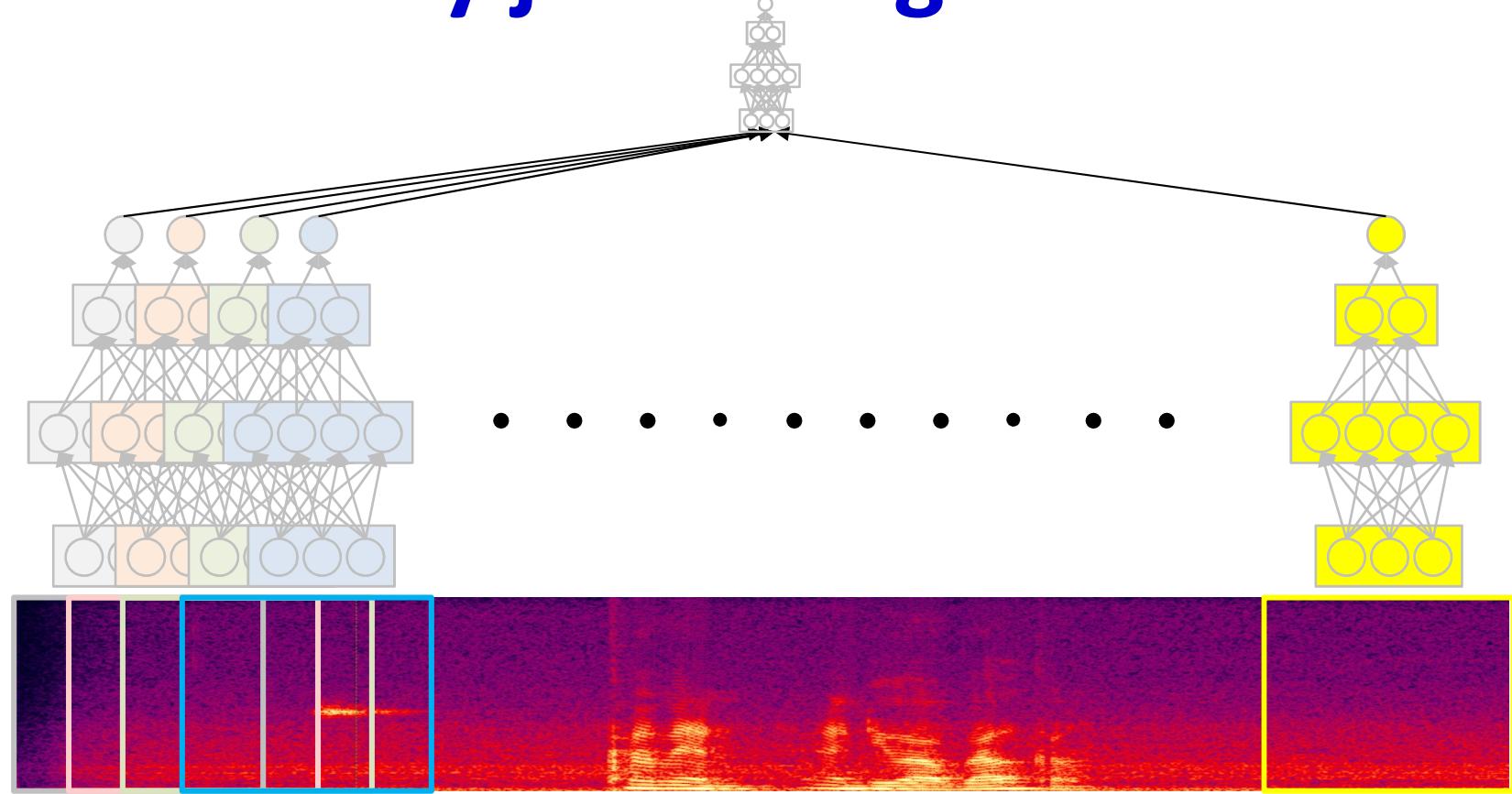
For  $t = 1:T-K+1$

$X_{\text{Segment}} = x(:, t:t+K-1)$

$y(t) = \text{MLP}(X_{\text{Segment}})$

$Y = \text{softmax}(y(1) \dots y(T-K+1))$

# 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

# Scanning with an MLP

- $K = \text{width of “patch” evaluated by MLP}$

For  $t = 1:T-K+1$

$X_{\text{Segment}} = x(:, t:t+K-1)$

$y(t) = \text{MLP}(X_{\text{Segment}})$

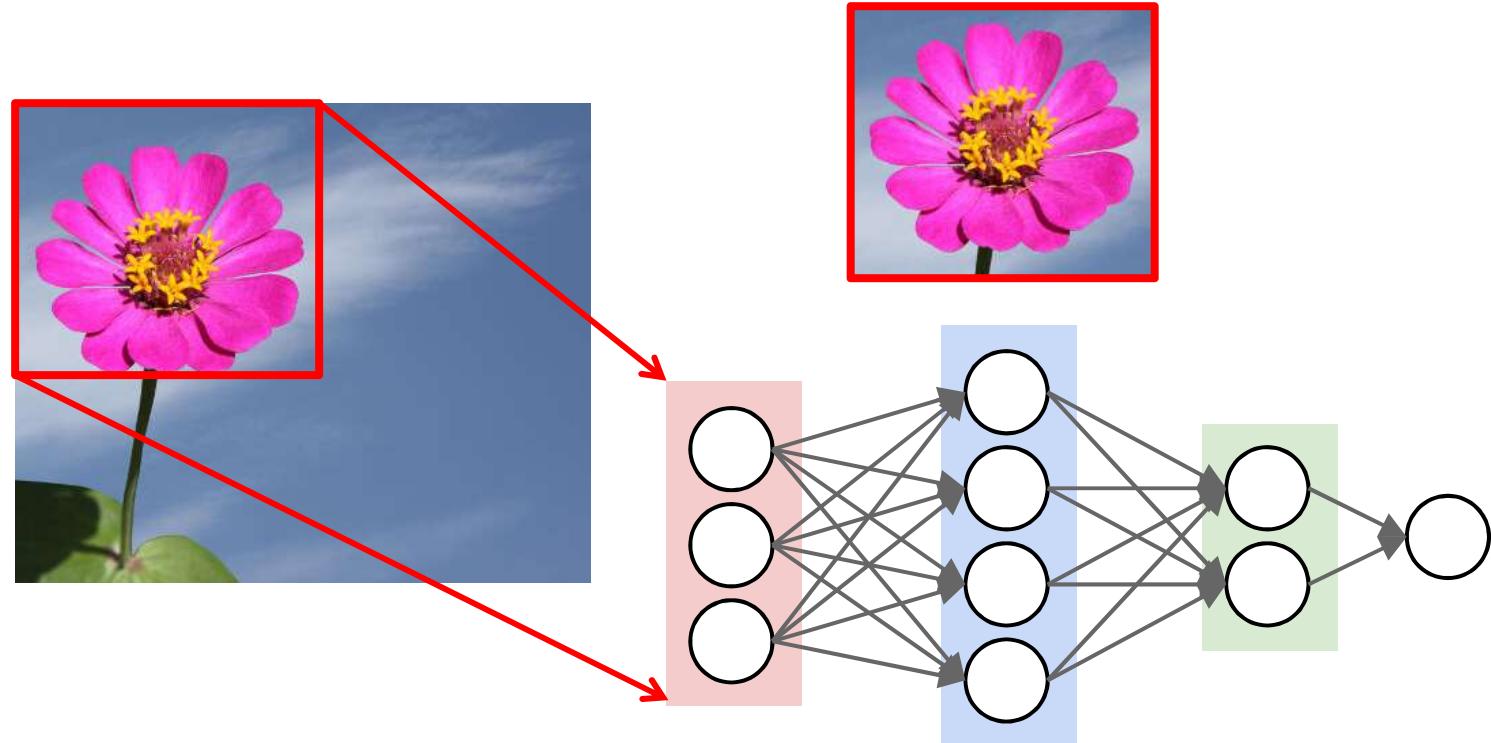
Just the final layer of the overall  
MLP

$Y = \text{softmax}(y(1) \dots y(T-K+1))$

# Scanning with an MLP

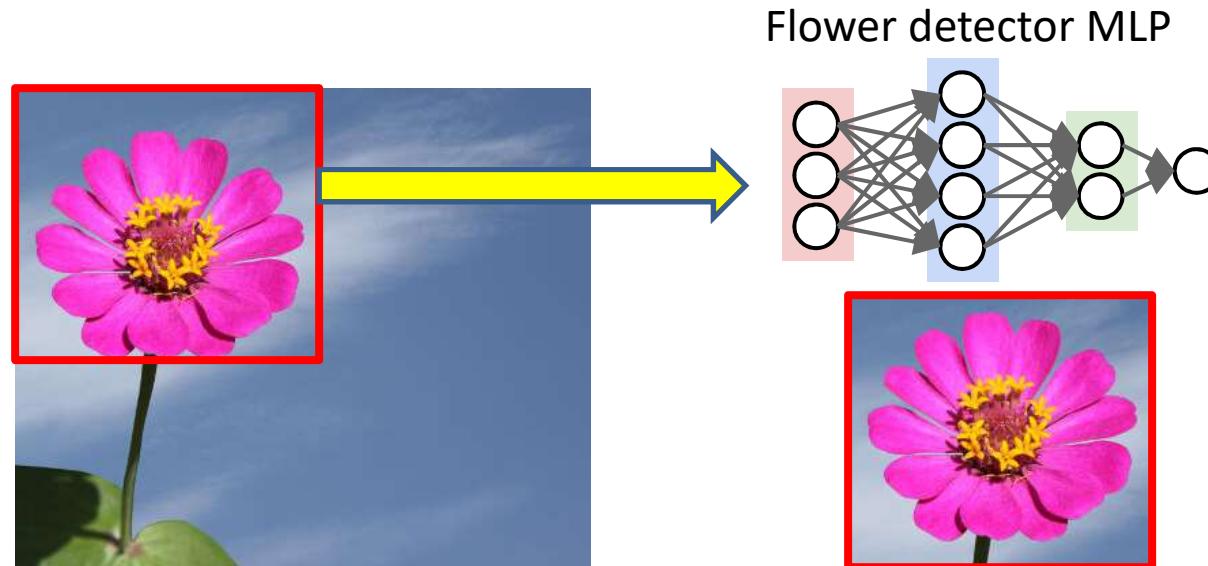
$Y = \text{giantMLP}(x)$

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



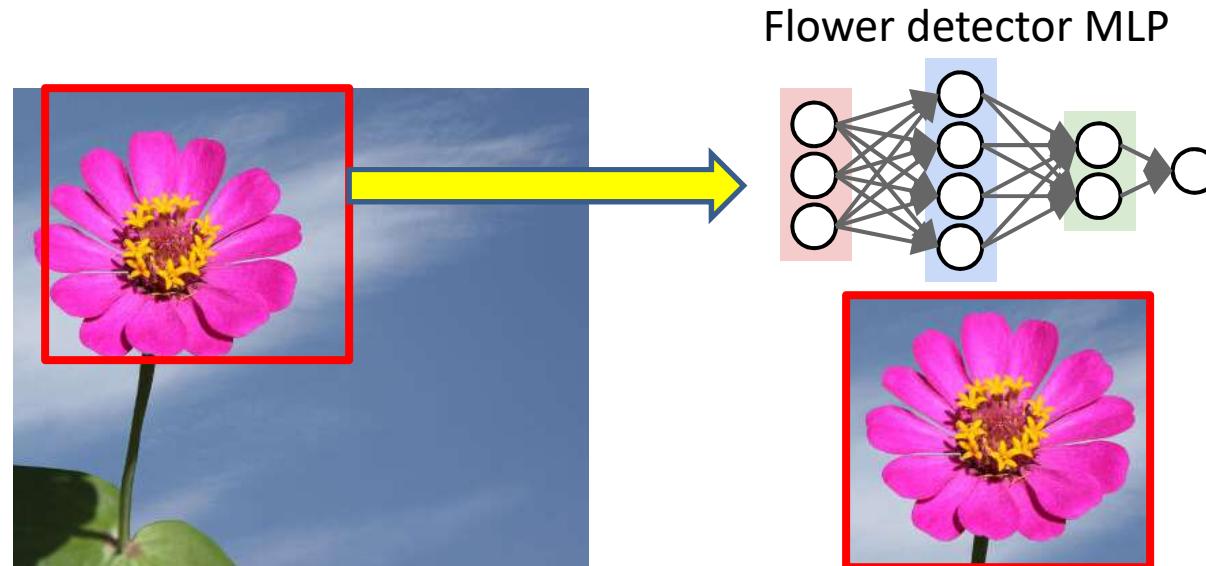
- *Scan* for the desired object
  - “Look” for the target object at each position

# Solution: Scan



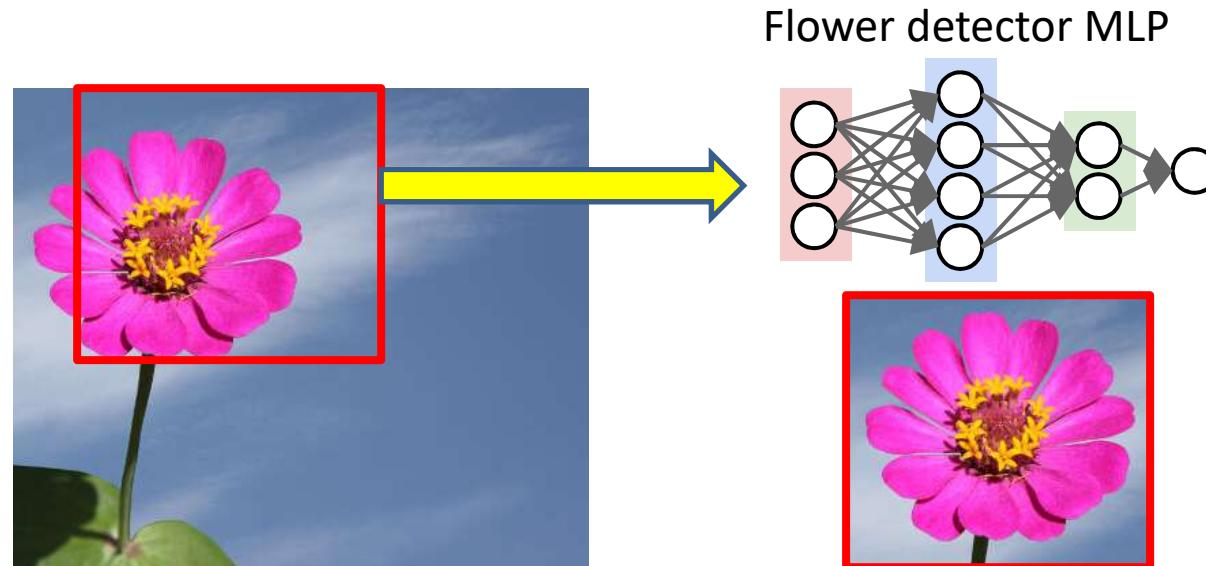
- *Scan* for the desired object

# Solution: Scan



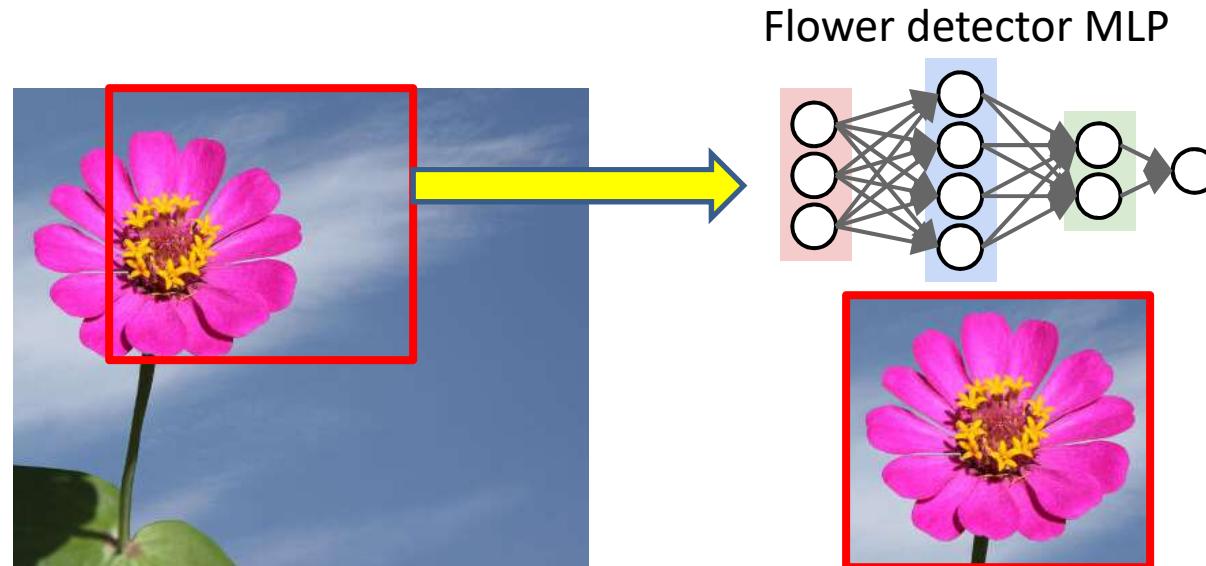
- *Scan* for the desired object

# Solution: Scan



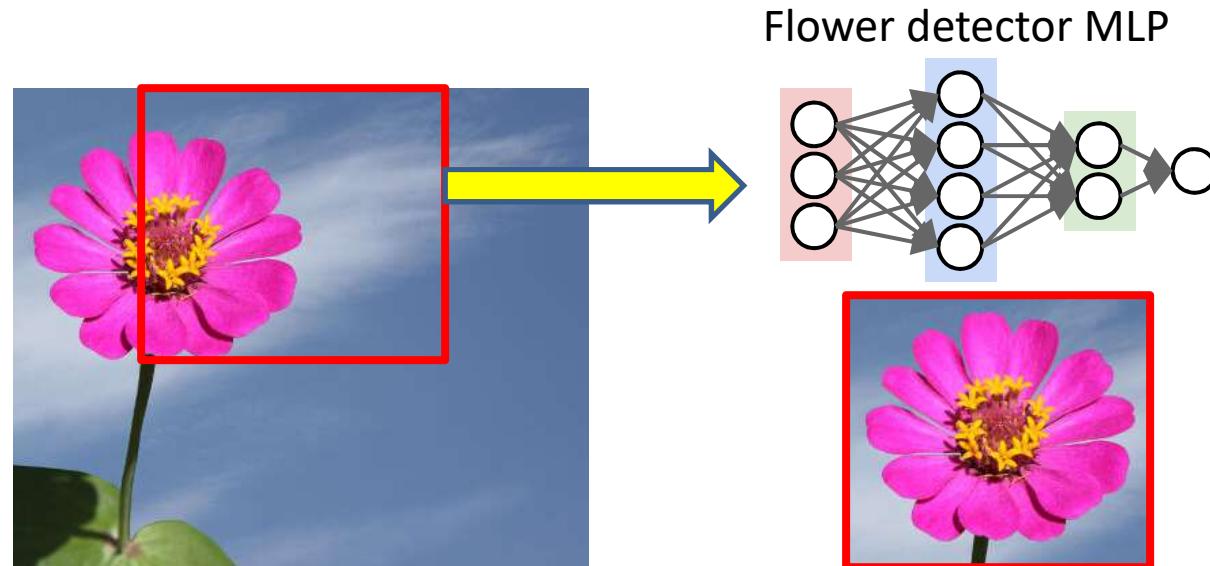
- *Scan* for the desired object

# Solution: Scan



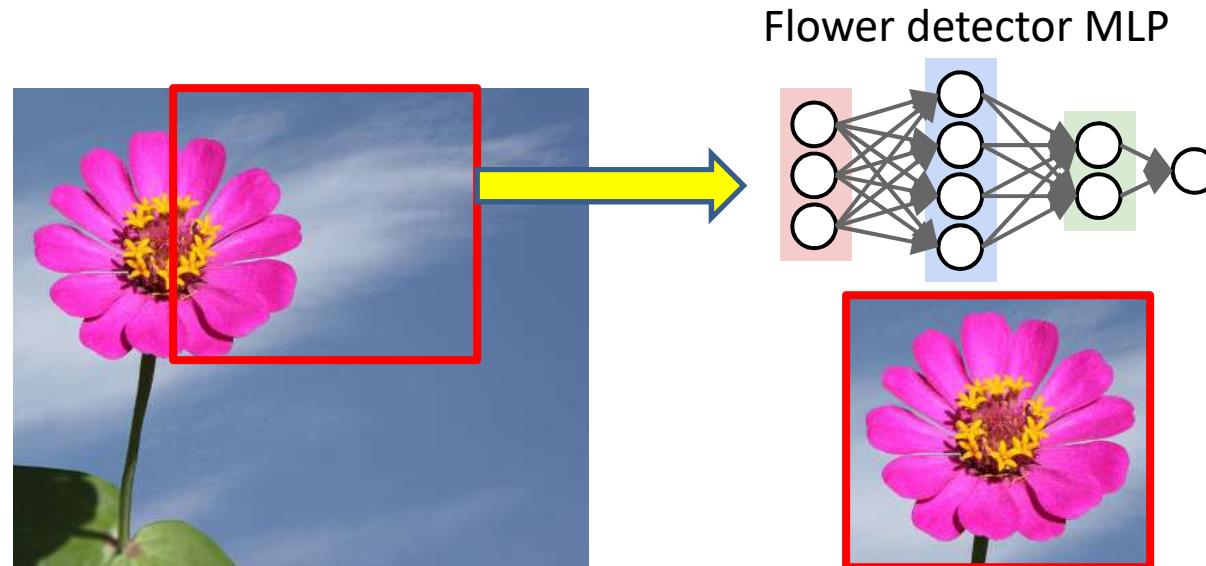
- *Scan* for the desired object

# Solution: Scan



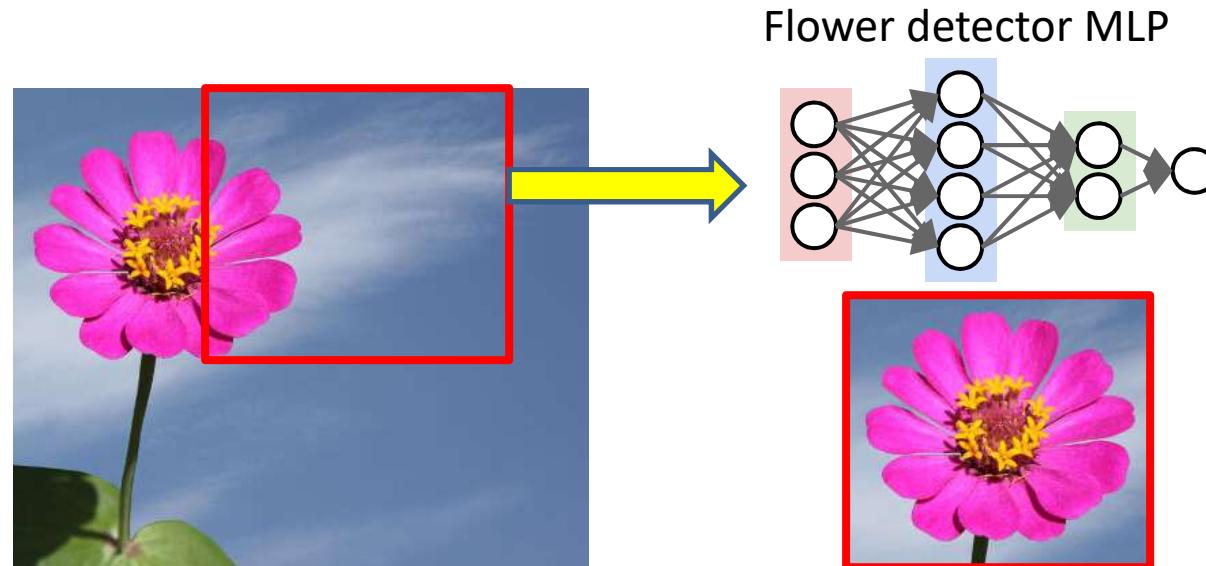
- *Scan* for the desired object

# Solution: Scan



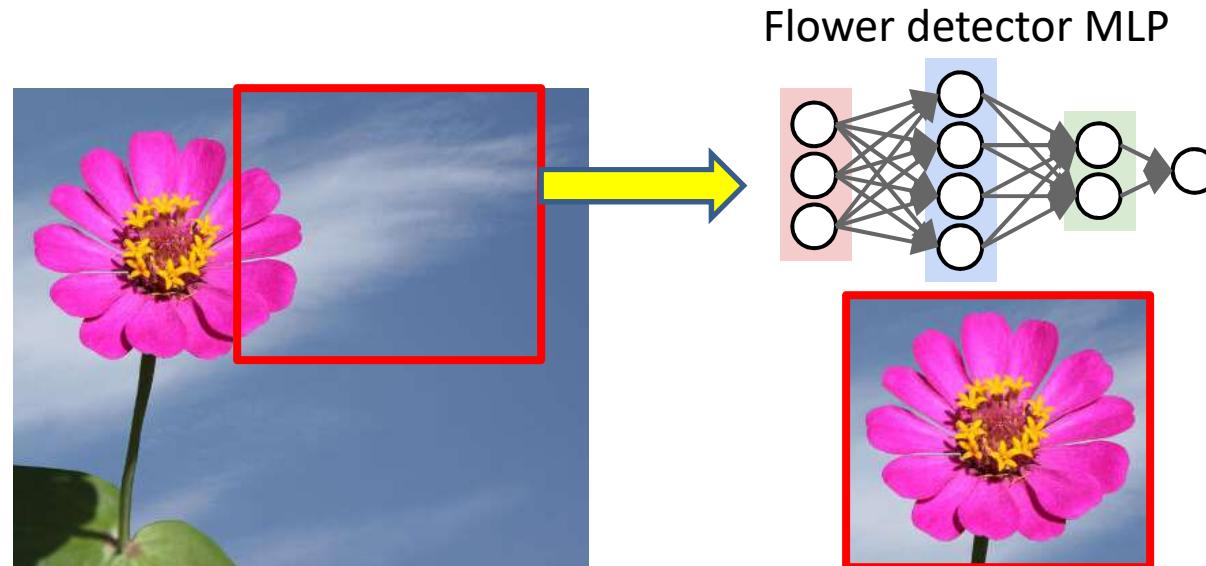
- *Scan* for the desired object

# Solution: Scan



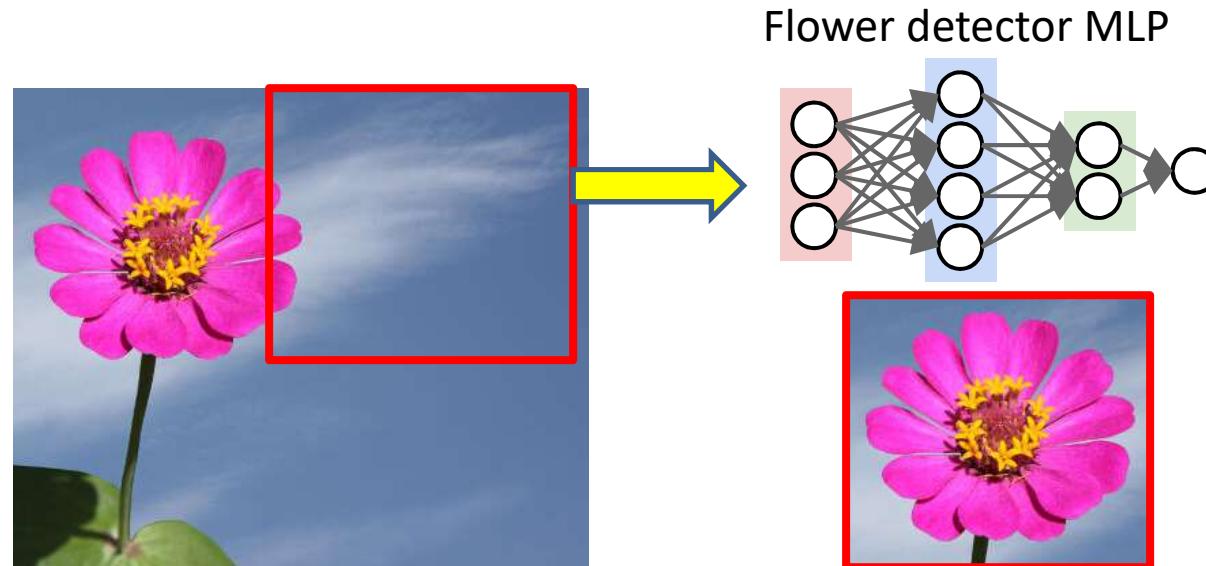
- *Scan* for the desired object

# Solution: Scan



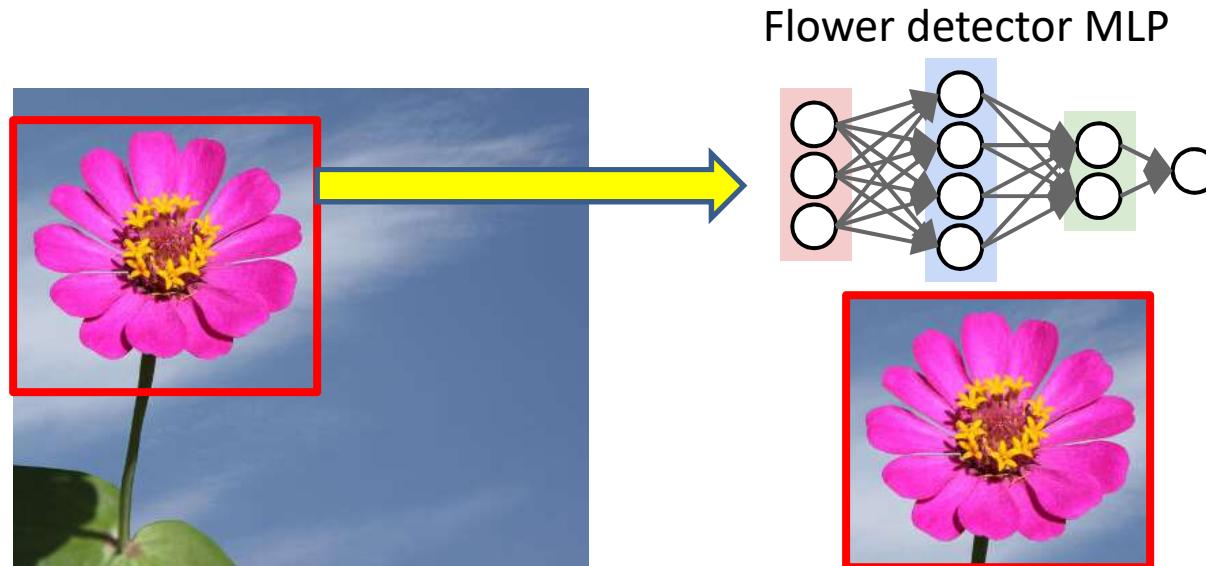
- *Scan* for the desired object

# Solution: Scan



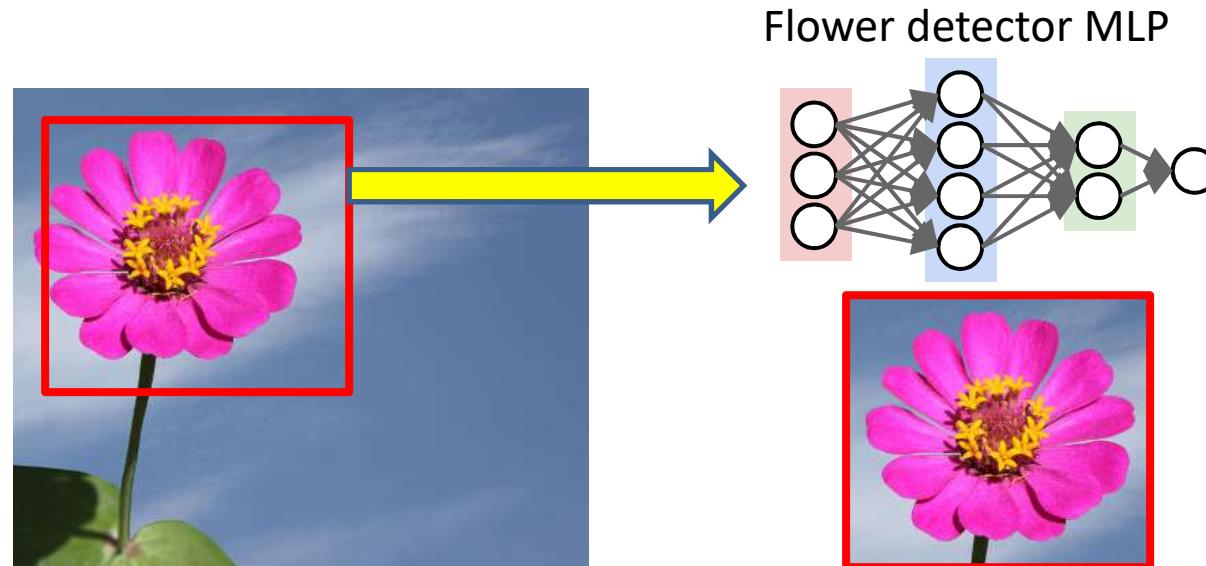
- *Scan* for the desired object

# Solution: Scan



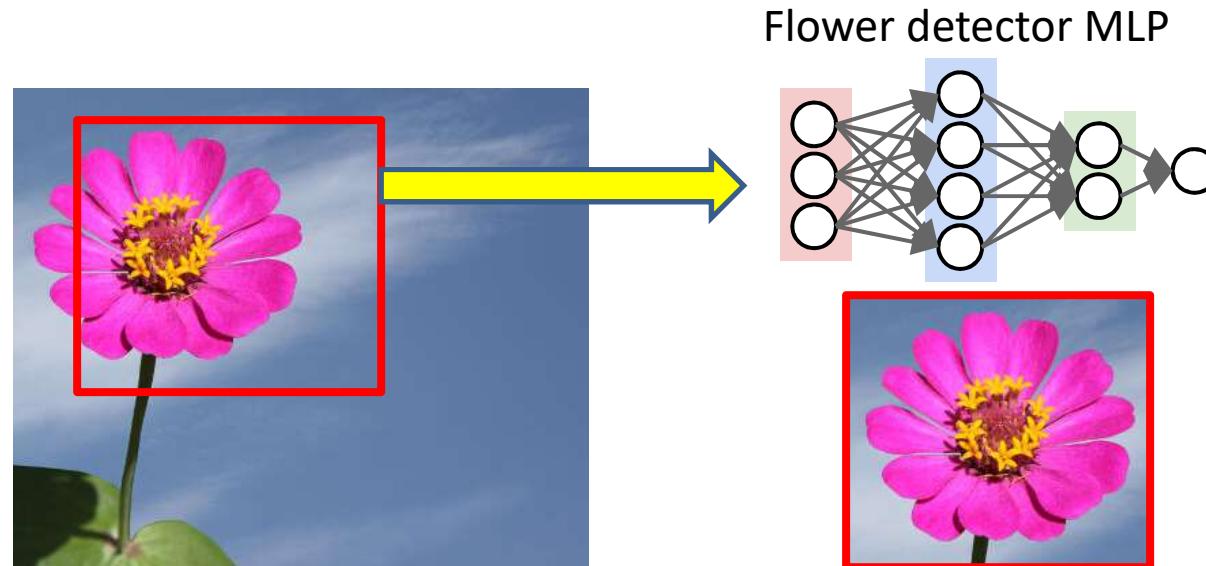
- *Scan* for the desired object

# Solution: Scan



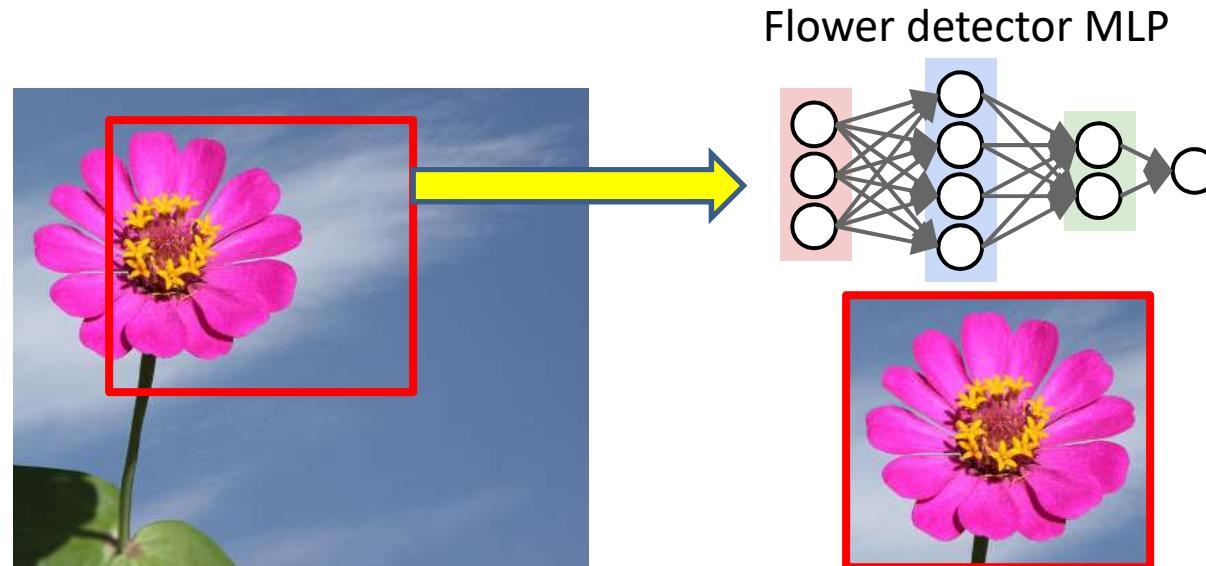
- *Scan* for the desired object

# Solution: Scan



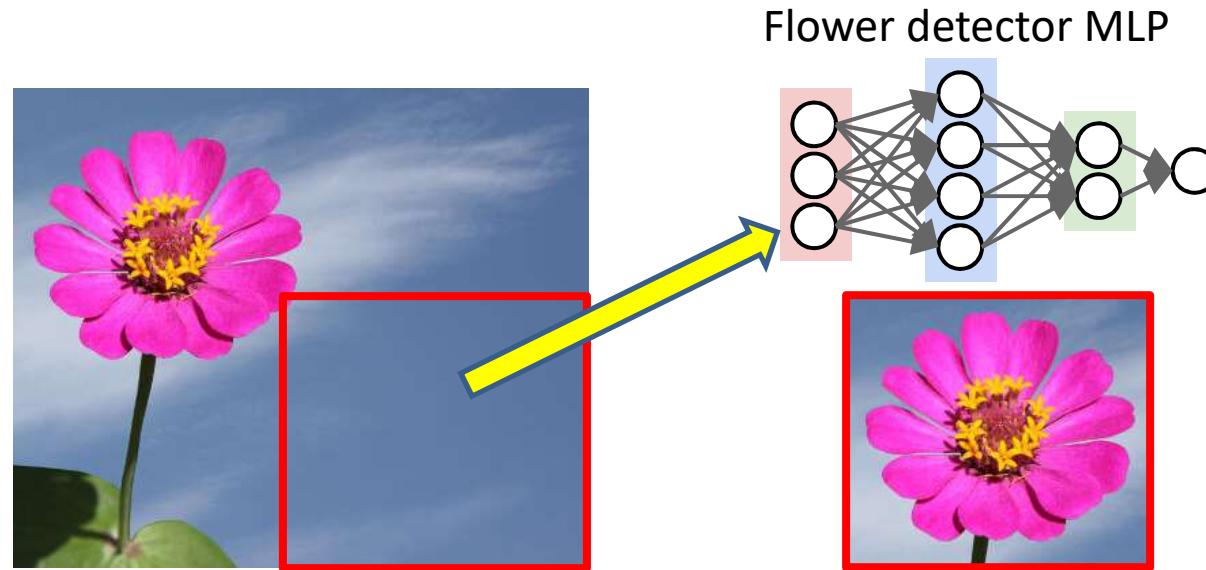
- *Scan* for the desired object

# Solution: Scan



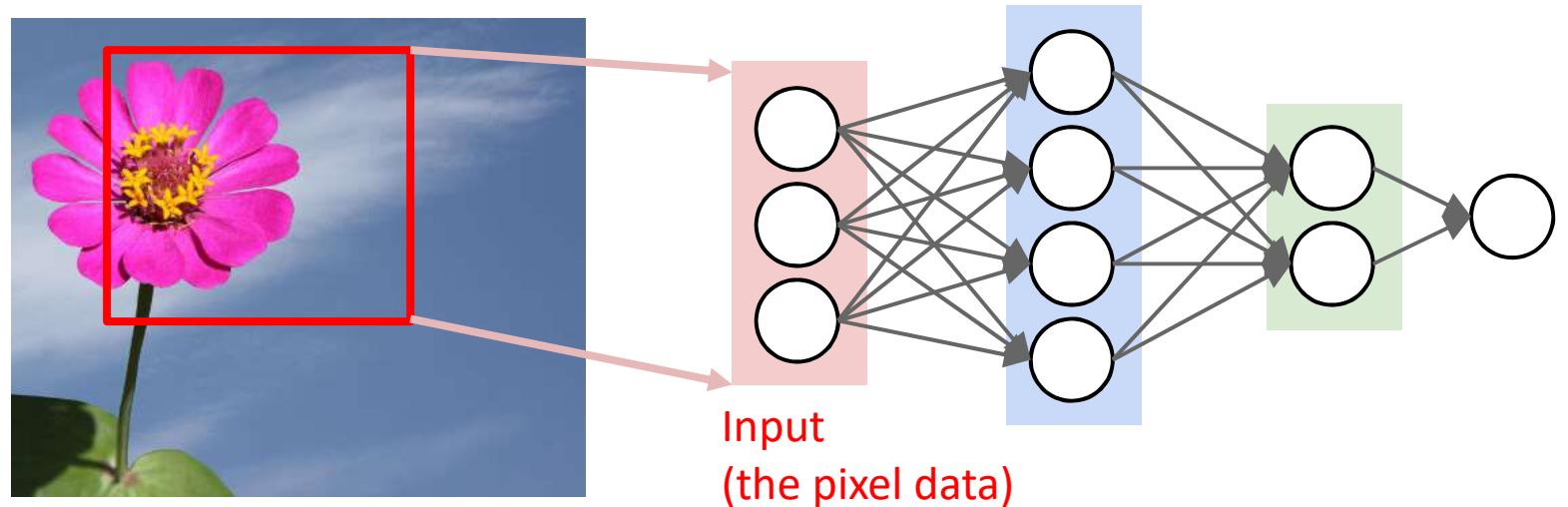
- *Scan* for the desired object

# Solution: Scan



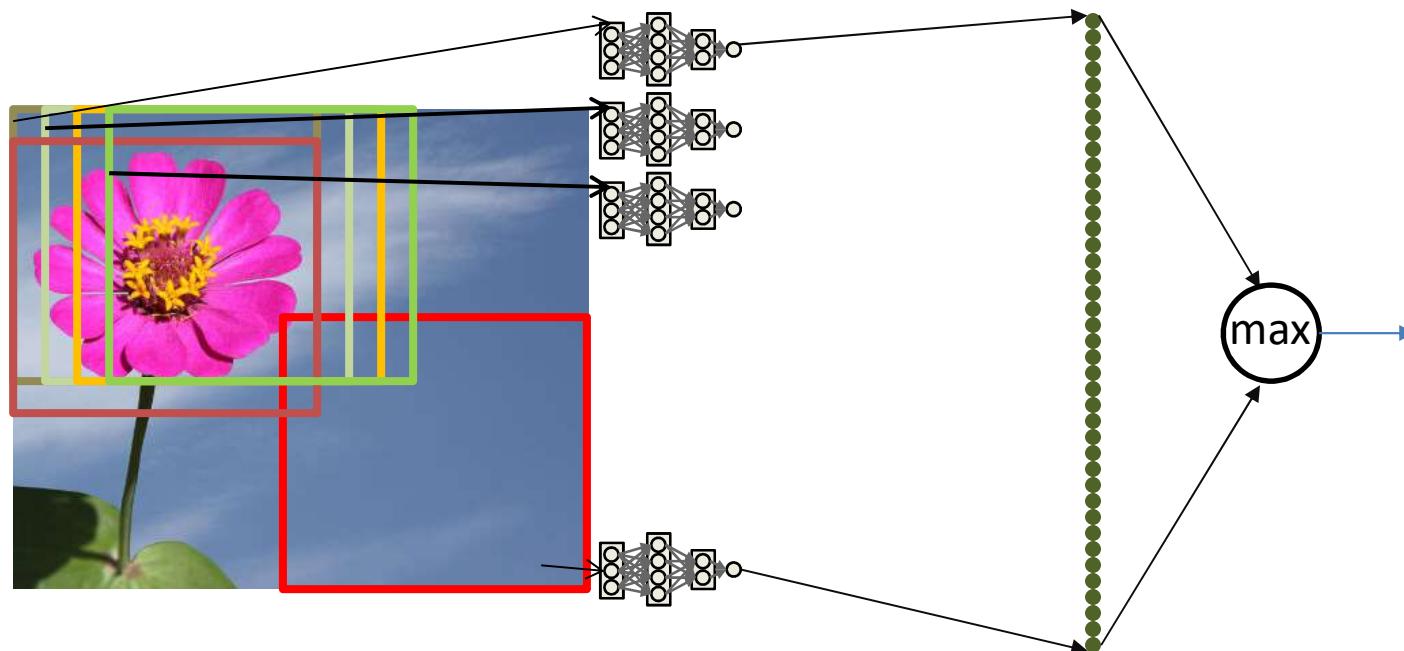
- *Scan* for the desired object

# Scanning



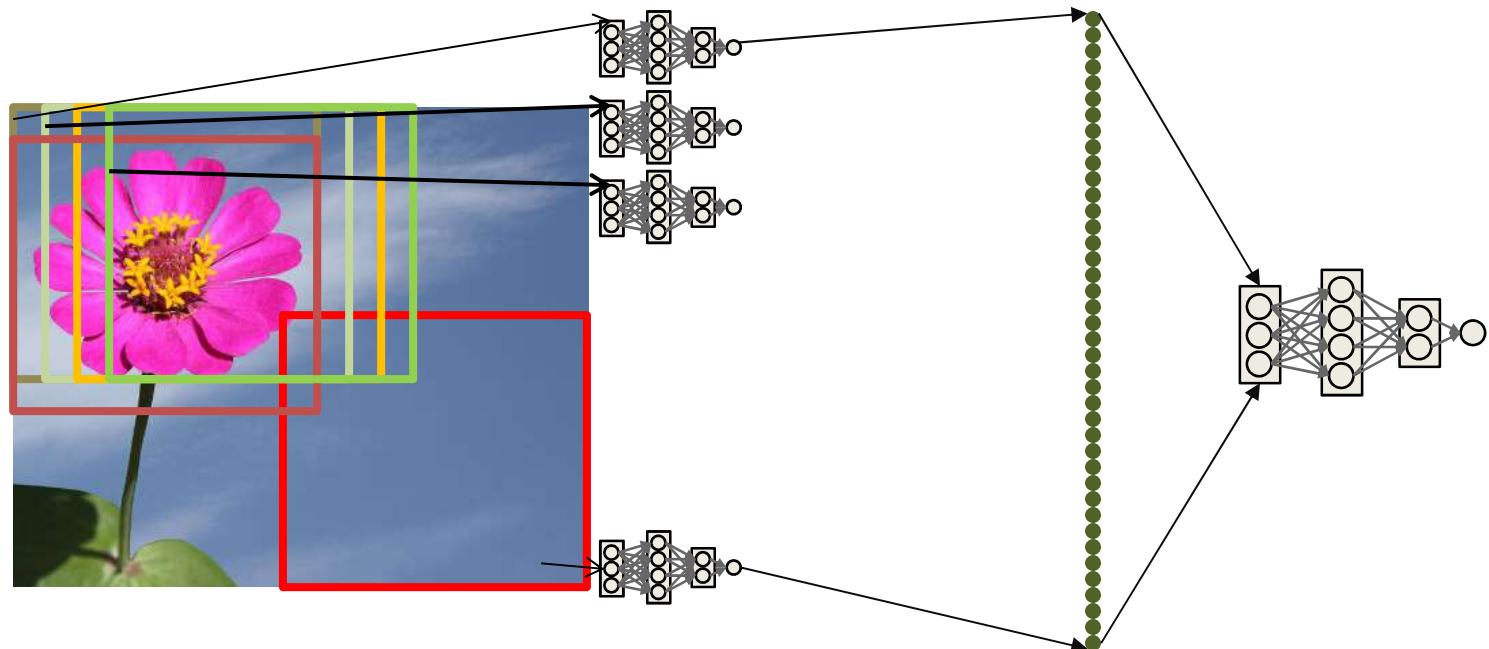
- *Scan* for the desired object
- At each location, the entire region is sent through an MLP

# Scanning the picture to find a flower



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

- $K \times K$  = size of “patch” evaluated by MLP
- $W$  is width of image
- $H$  is height of image

For  $i = 1 : W - K + 1$

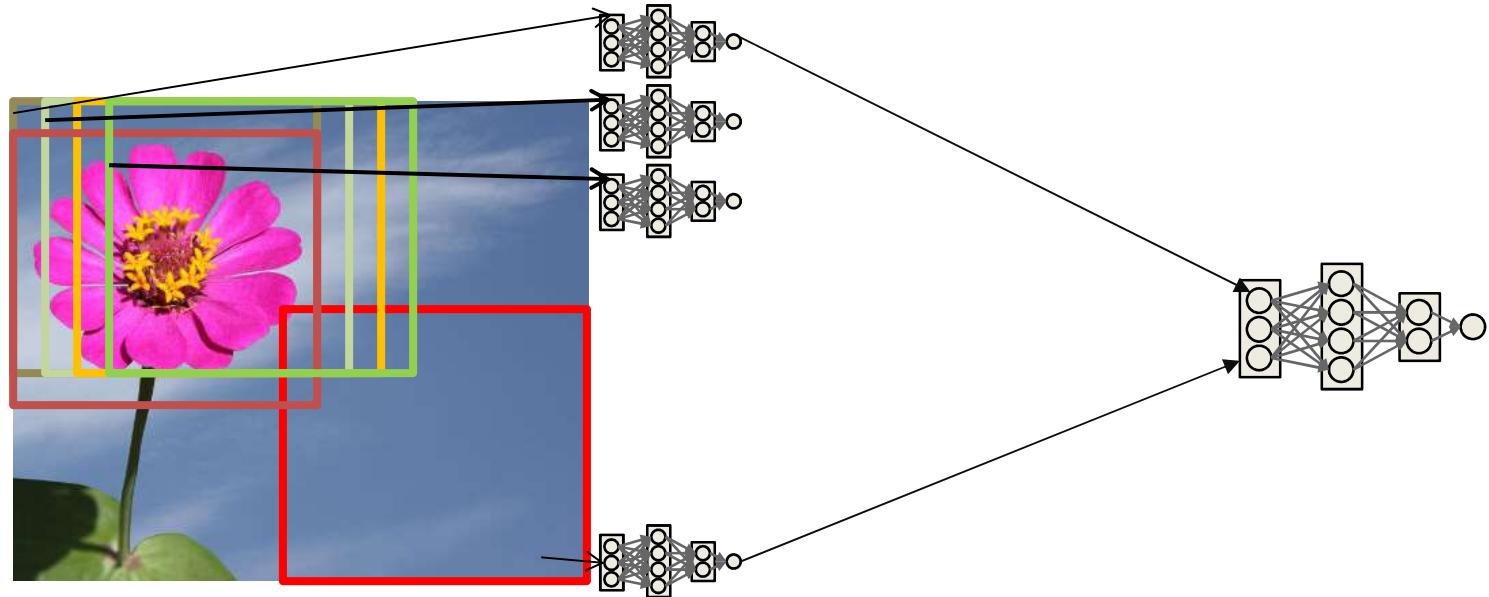
    For  $j = 1 : H - K + 1$

        ImgSegment = Img( $i : i + K - 1, j : j + K - 1$ )

$y(i, j) = \text{MLP}(\text{ImgSegment})$

$\hat{Y} = \text{softmax}(y(1, 1) \dots y(W - K + 1, H - K + 1))$

# 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

# Scanning with an MLP

- $K \times K$  = 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( $i : i + K - 1, j : j + K - 1$ )

$y(i, j) = \text{MLP}(\text{ImgSegment})$

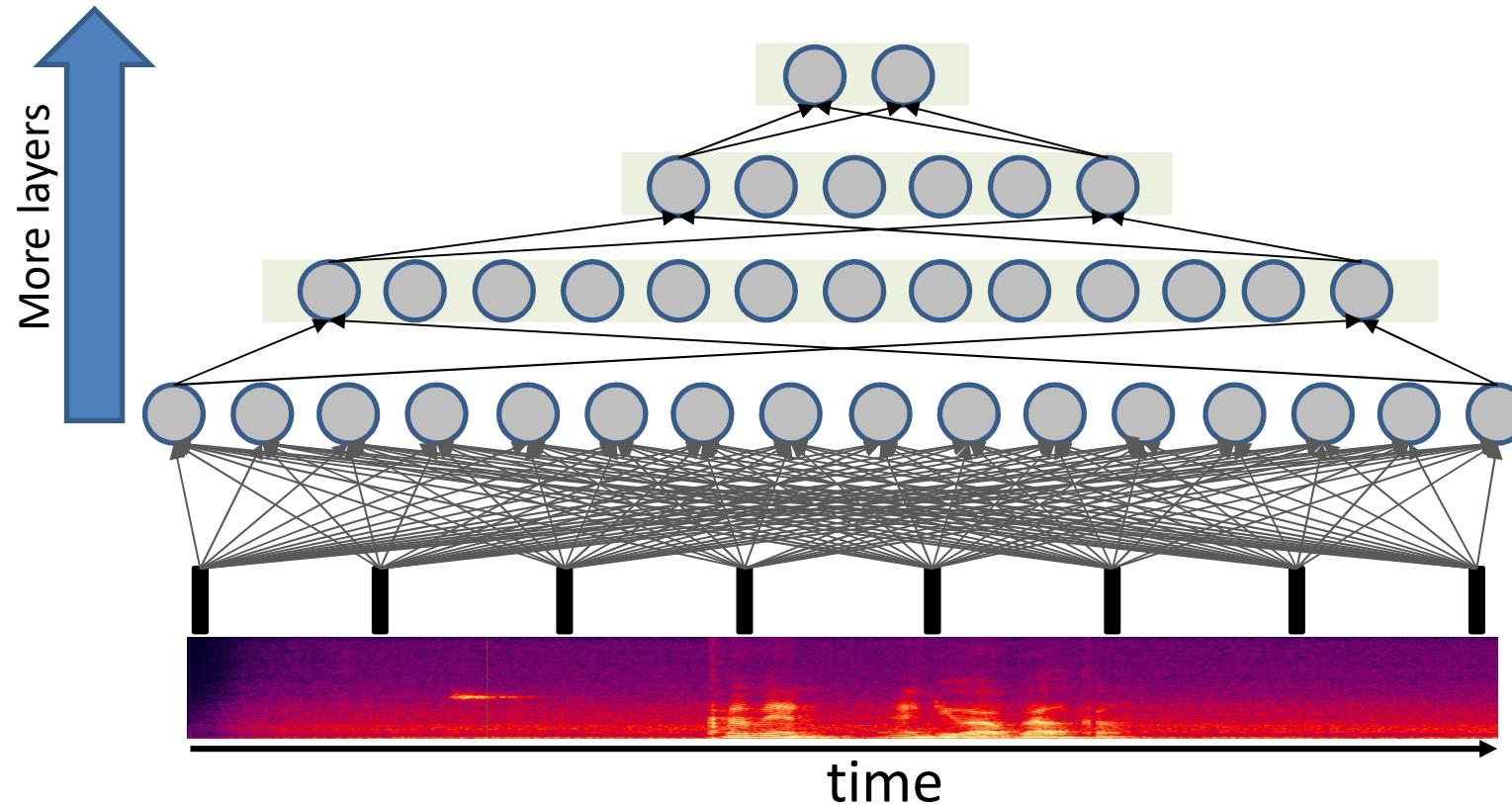
Just the final layer of the overall  
MLP

$\mathbb{Y} = \text{softmax}(\mathbb{y}(1, 1) \dots \mathbb{y}(W - K + 1, H - K + 1))$

# Scanning with an MLP

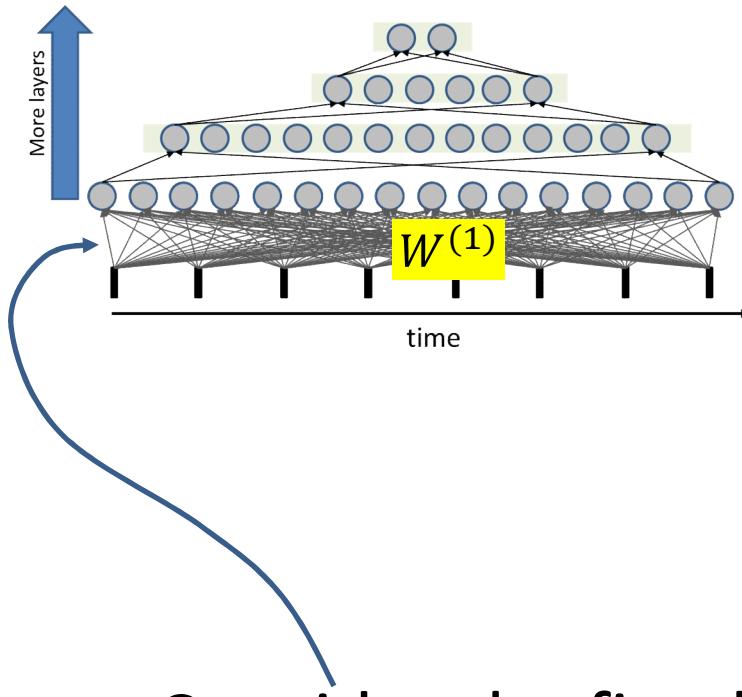
```
Y = giantMLP(img)
```

# 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

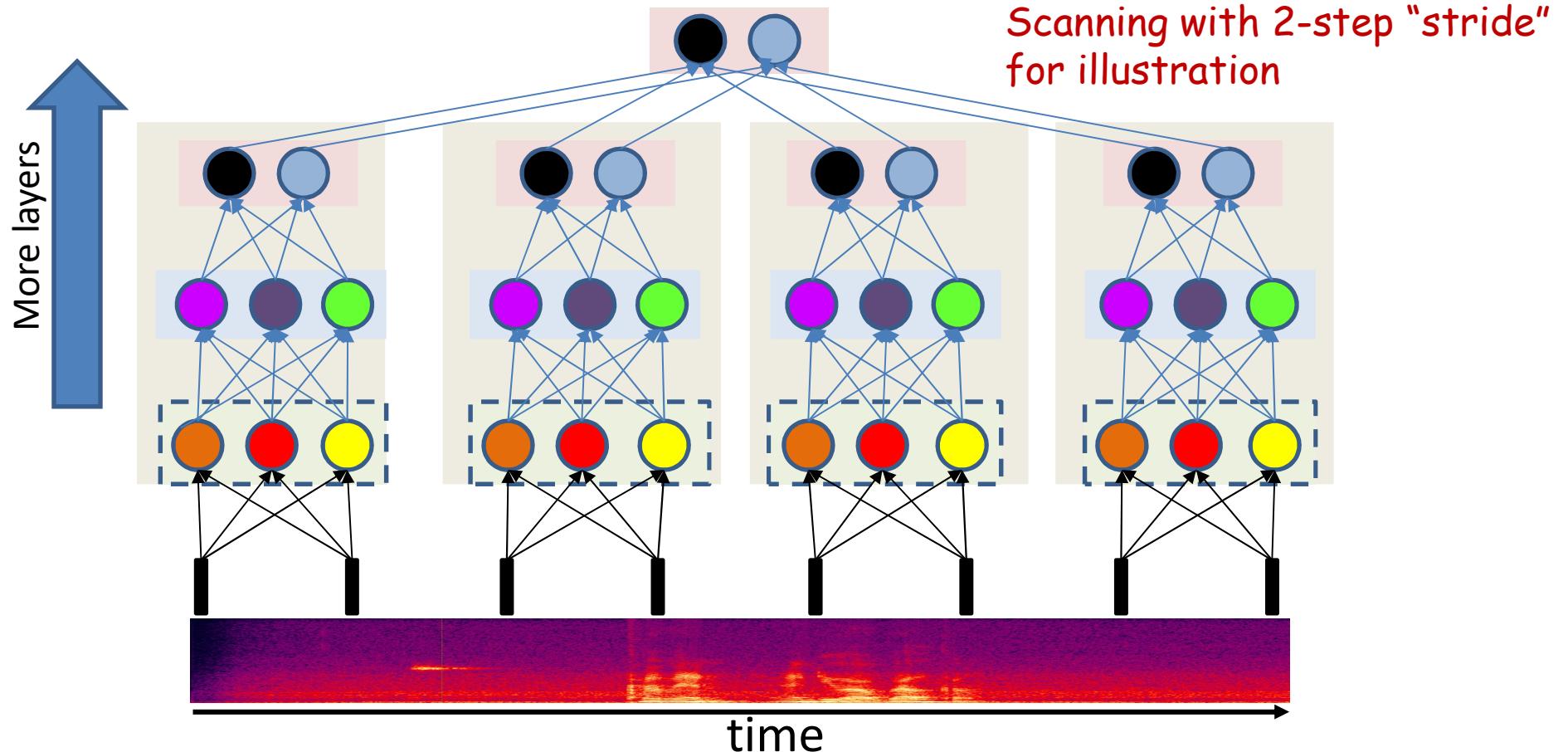
# Regular network



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

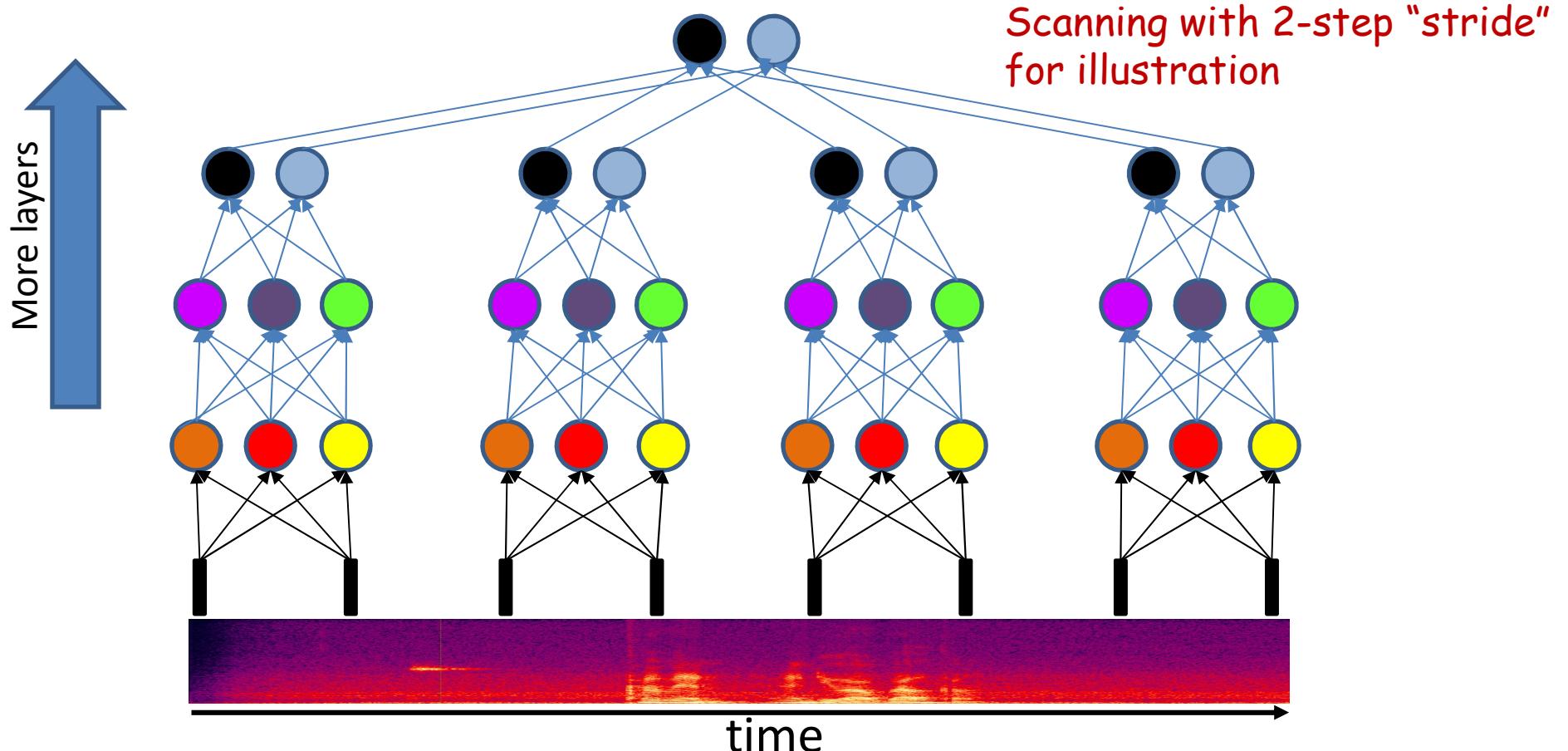
- Consider the first layer
  - Assume  $N$  inputs and  $M$  outputs
- The weights matrix is a full  $N \times M$  matrix
  - Requiring  $NM$  unique parameters

# Scanning networks



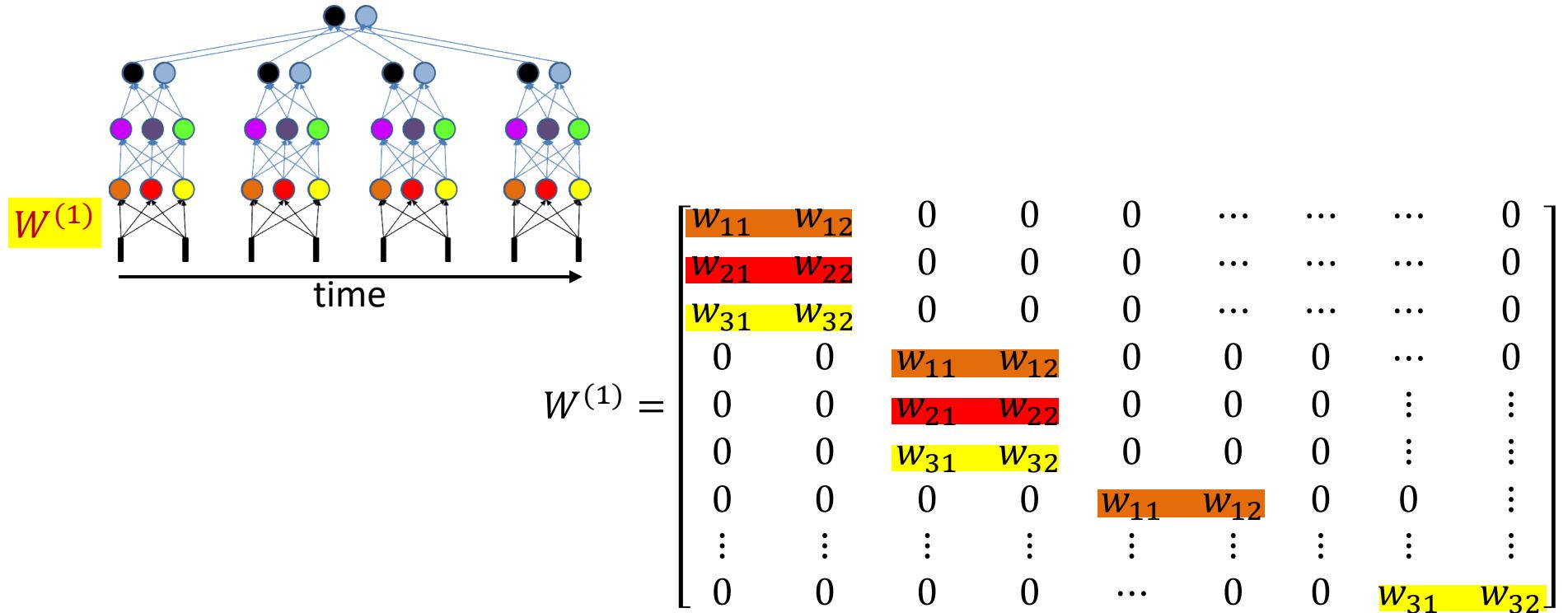
- In a **scanning MLP** each neuron is connected to a subset of neurons in the previous layer
  - The weights matrix is sparse
  - The weights matrix is block structured **with identical blocks**

# Scanning networks



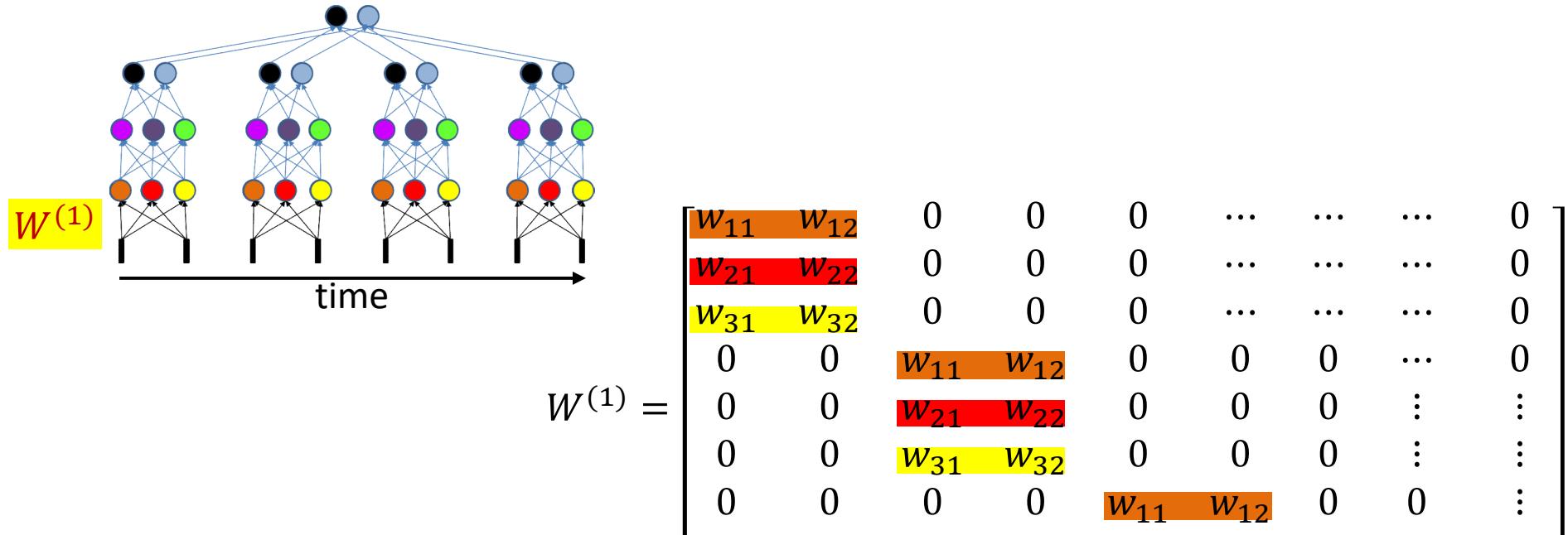
- In a **scanning MLP** each neuron is connected to a subset of neurons in the previous layer
  - The weights matrix is sparse
  - The weights matrix is block structured **with identical blocks**

# Scanning networks



- 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**

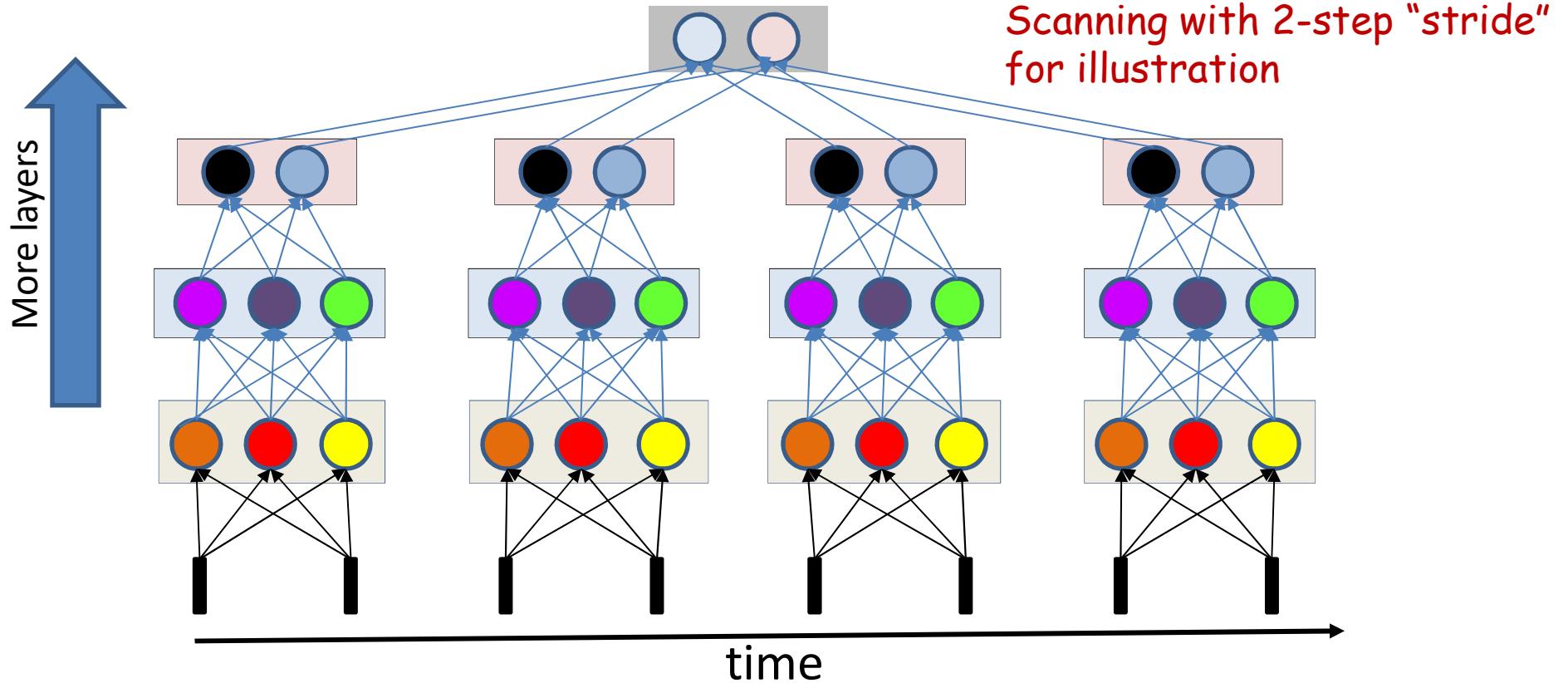
# Scanning networks



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

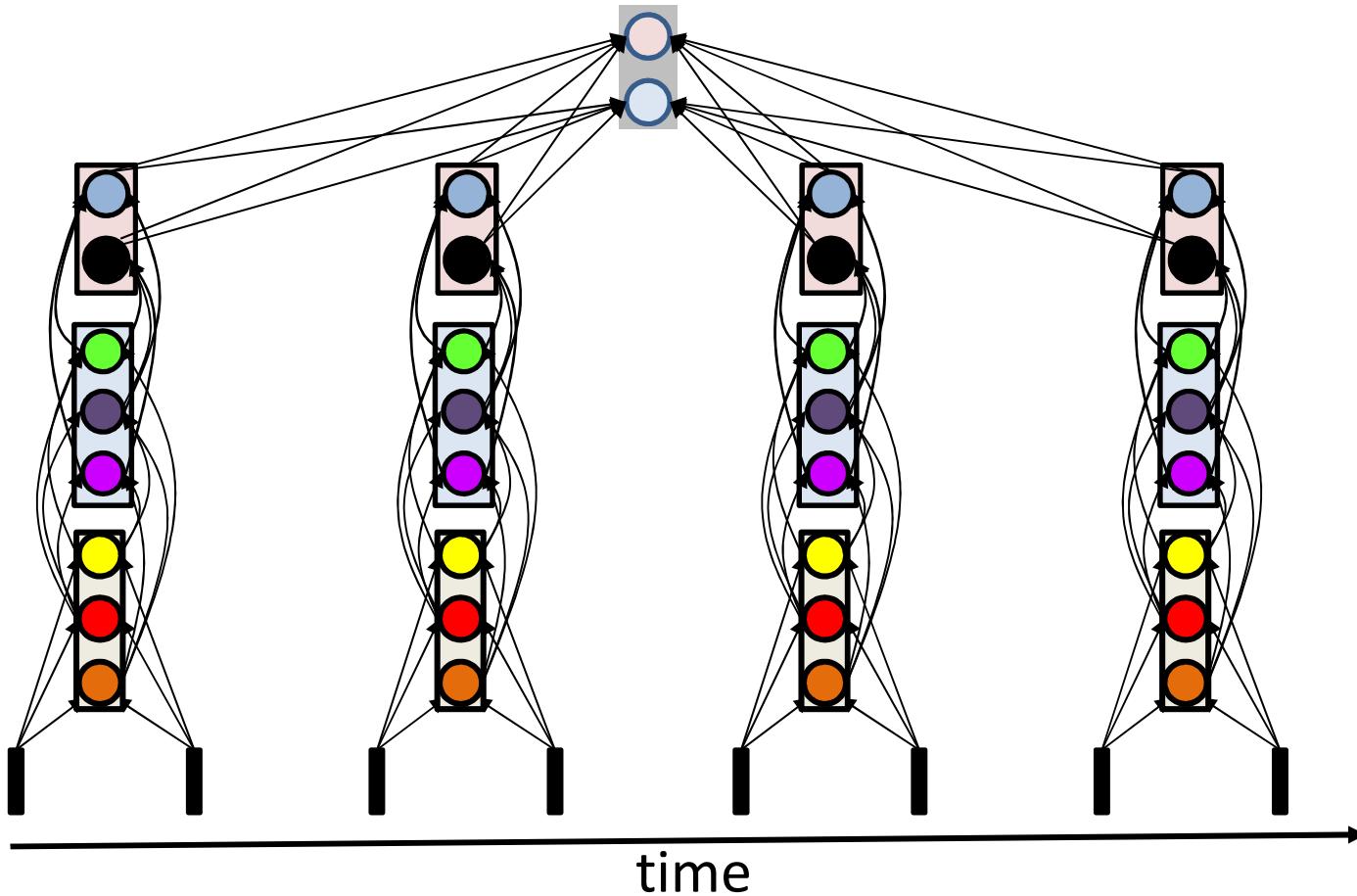
- In a **scanning MLP** each neuron is connected to a subset of neurons in the previous layer
  - The weights matrix is sparse
  - The weights matrix is block **structured with identical blocks**
  - *The network is a shared-parameter model*
- *Also, far fewer parameters (we return to this topic shortly)*

# Scanning networks



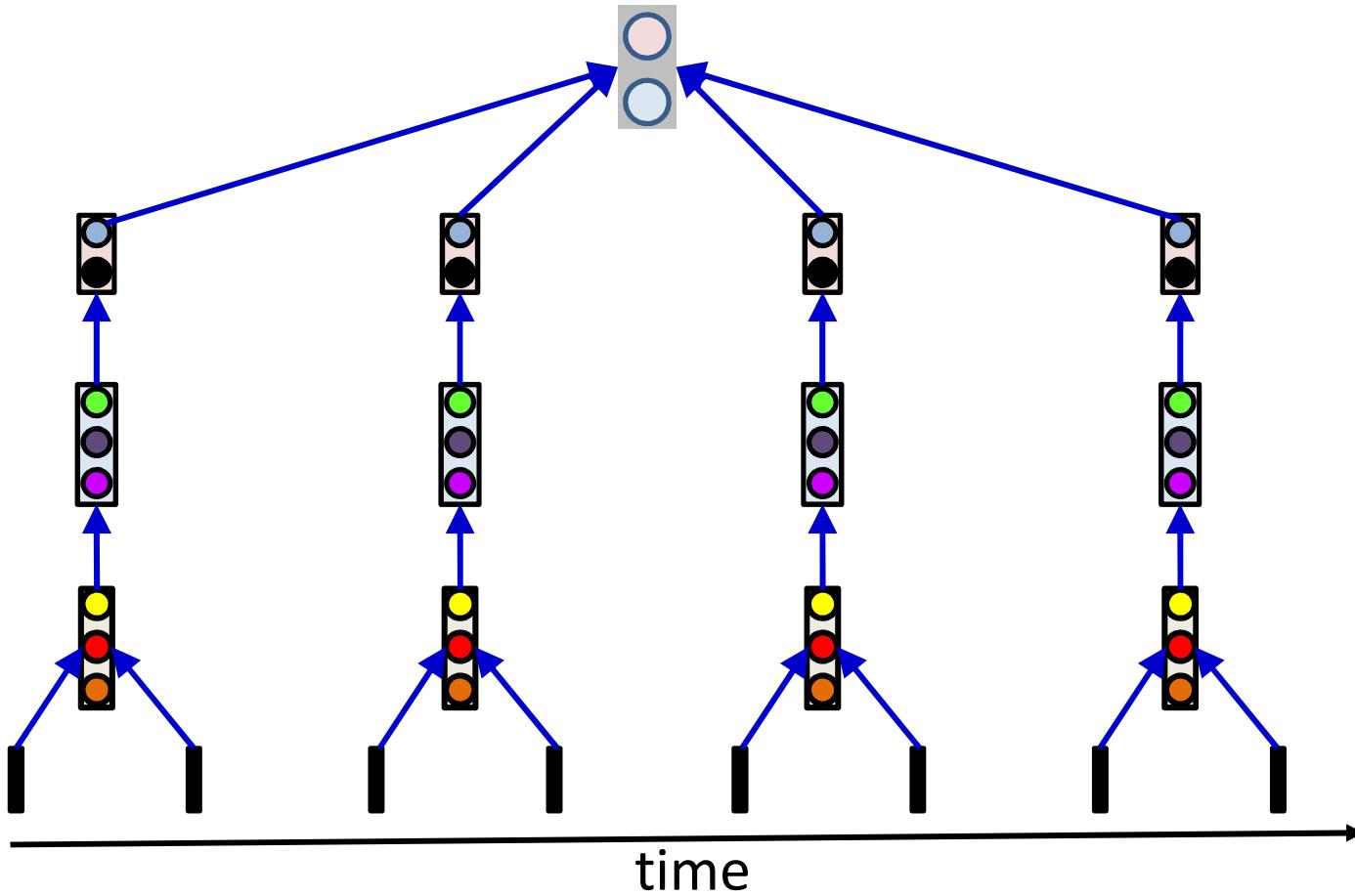
- Modifying the visualization for intuition..
  - Will still be the same network

# Scanning networks



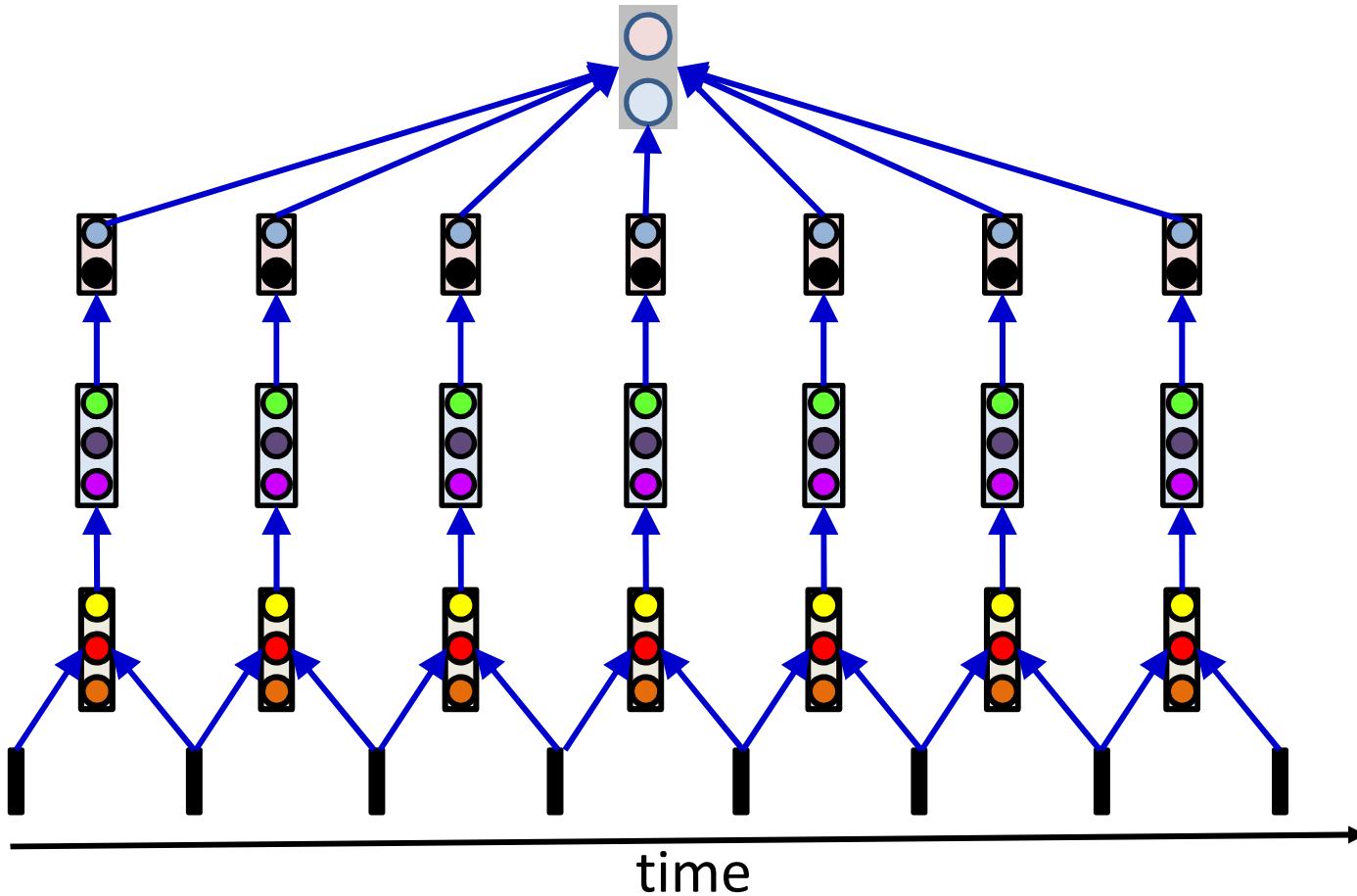
- A modified drawing
  - Indicates progression of time/space
    - The progression of “bars” of neurons is indicative of time
    - Note: bars at the lowest level are also *vectors* of inputs
  - More appropriate
    - Since vertical bars are vectors

# 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,  $N \times M$  connections

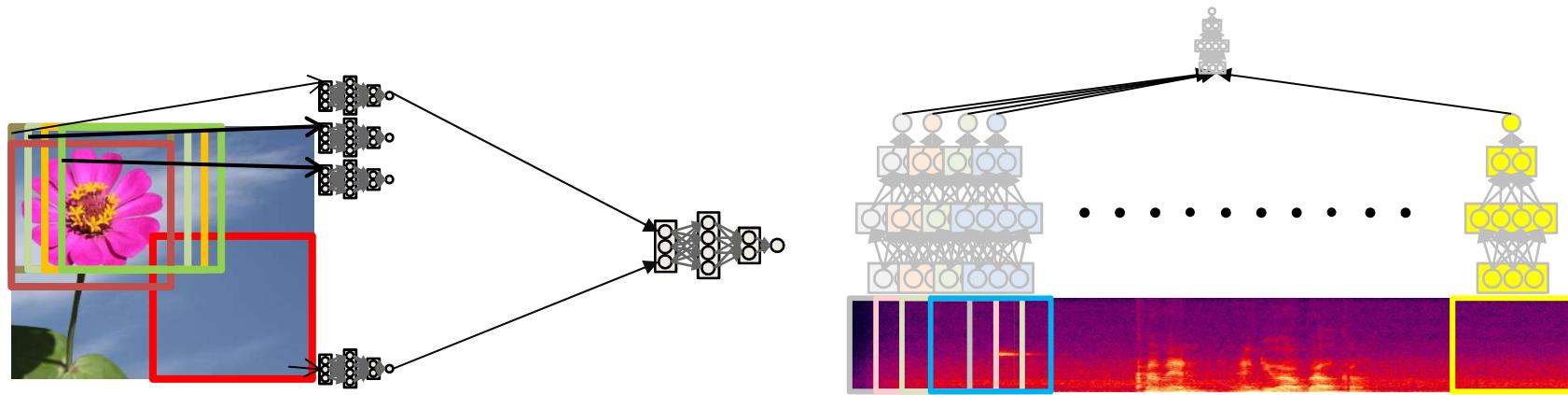
# Scanning networks



- 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

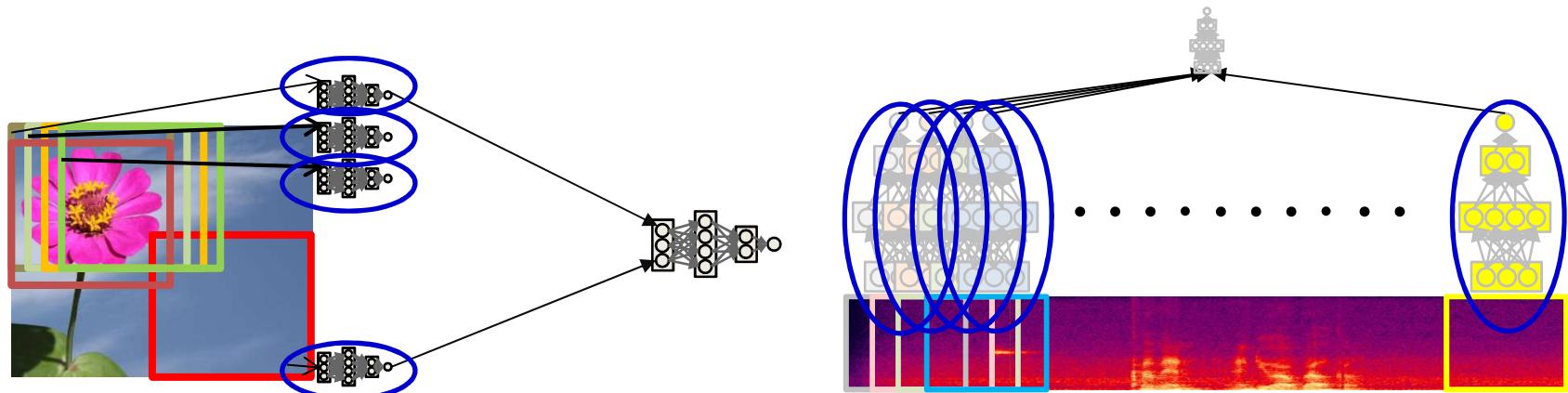
Visualizing scanning with a stride of 1

# Training the network



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

# Training the network: constraint



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

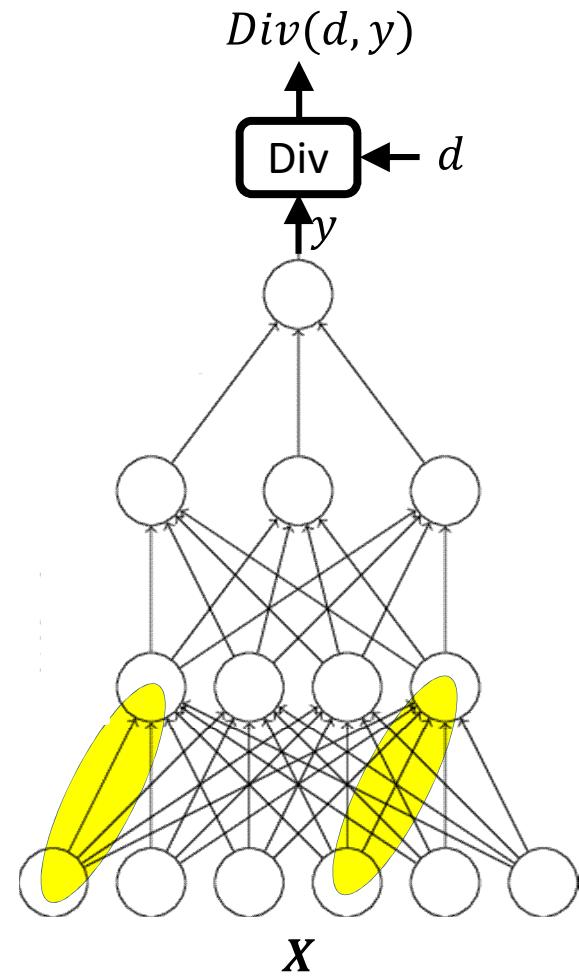
# Learning in shared parameter networks

- Consider a simple network with shared weights

$$w_{ij}^k = w_{mn}^l = w^s$$

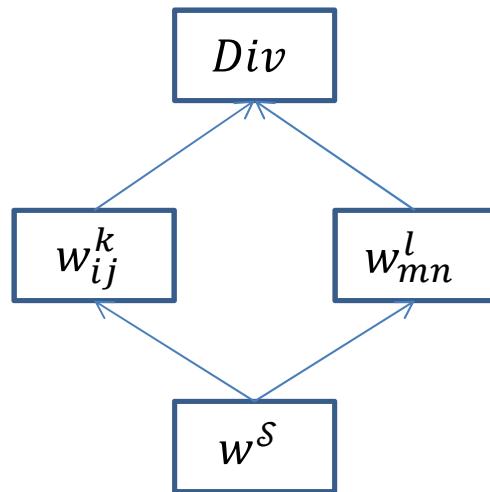
- A weight  $w_{ij}^k$  is required to be identical to the weight  $w_{mn}^l$

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

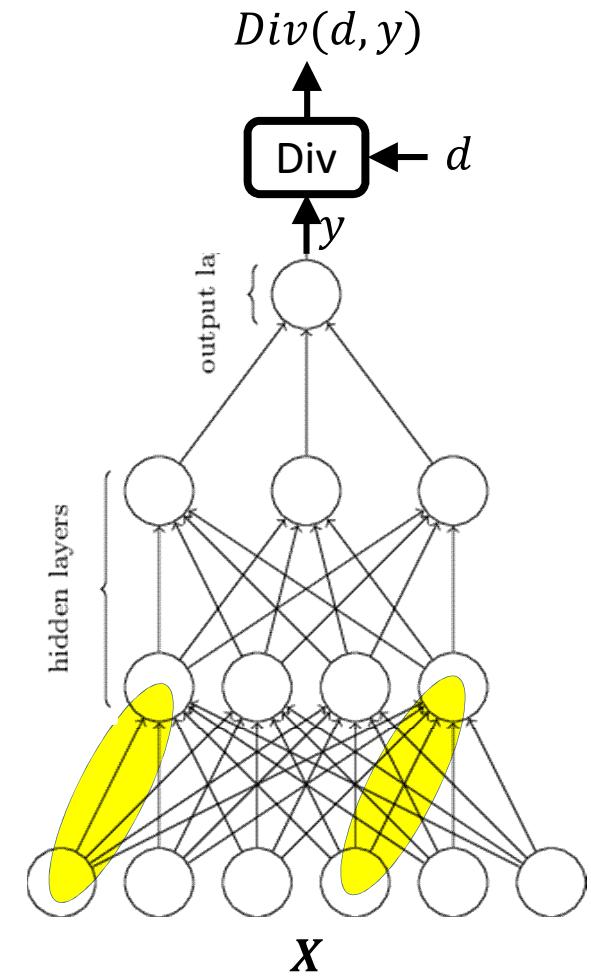


# Computing the divergence of shared parameters

Influence diagram

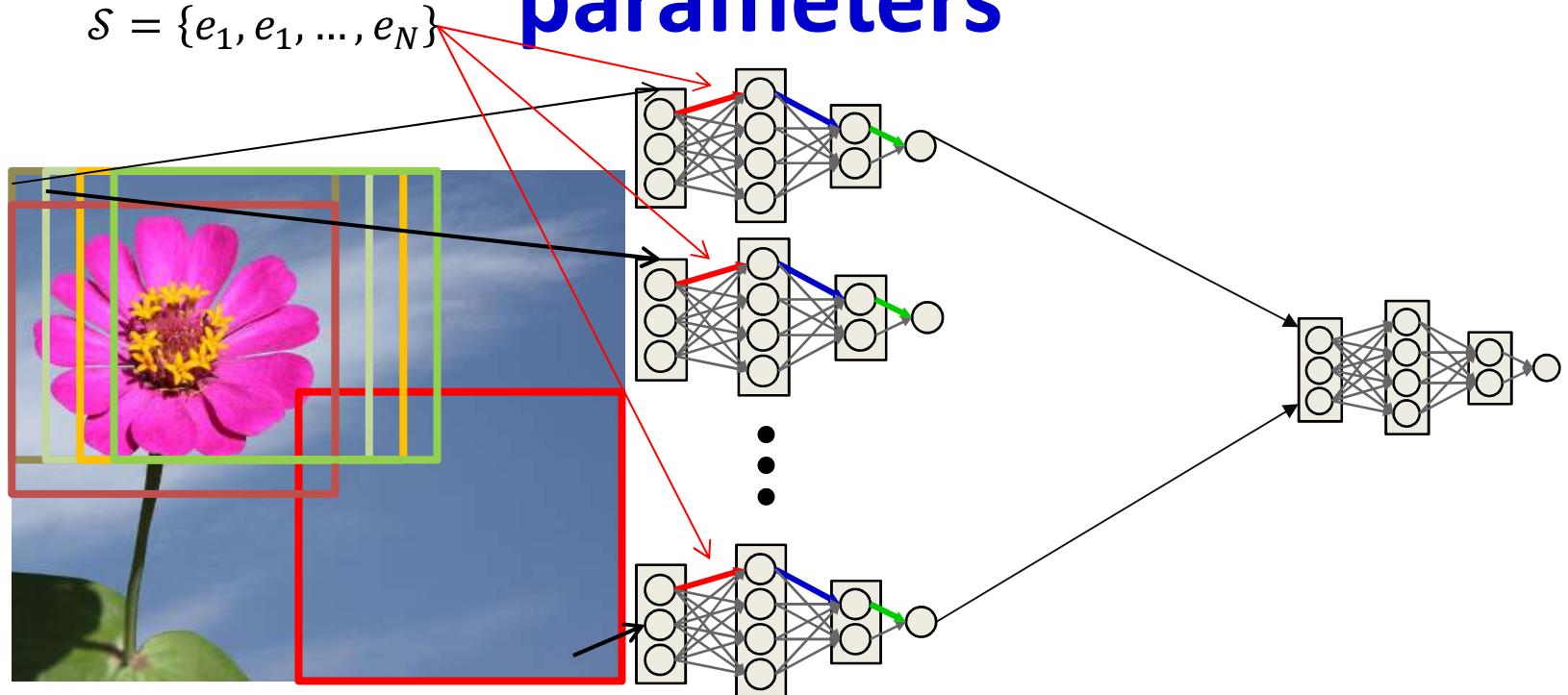


$$\begin{aligned}\frac{dDiv}{dw^s} &= \frac{\partial Div}{\partial w_{ij}^k} \frac{dw_{ij}^k}{dw^s} + \frac{\partial Div}{\partial w_{mn}^l} \frac{dw_{mn}^l}{dw^s} \\ &= \frac{\partial Div}{\partial w_{ij}^k} + \frac{\partial Div}{\partial w_{mn}^l}\end{aligned}$$



- 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:

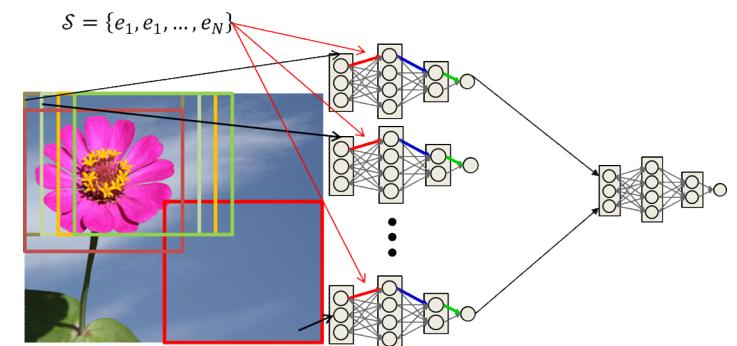
$$\nabla_{\mathcal{S}} Loss = \frac{dLoss}{dw^{\mathcal{S}}}$$

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

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

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

- Until  $Loss$  has converged



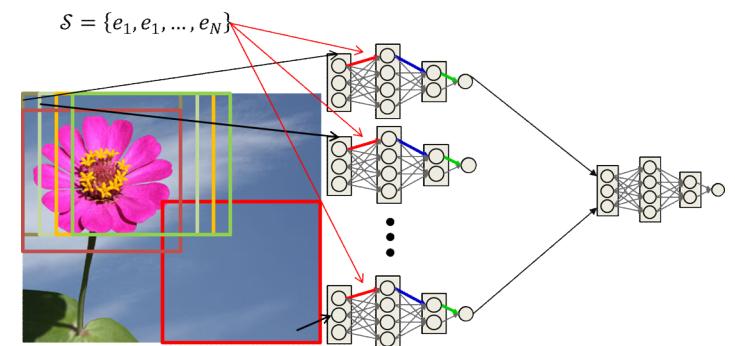
# Training networks with shared parameters

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

- 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

- For every training instance  $X$ 
  - For every set  $\mathcal{S}$ :

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

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

- $\nabla_{\mathcal{S}} \text{Loss} += \nabla_{\mathcal{S}} \text{Div}$

- Compute:

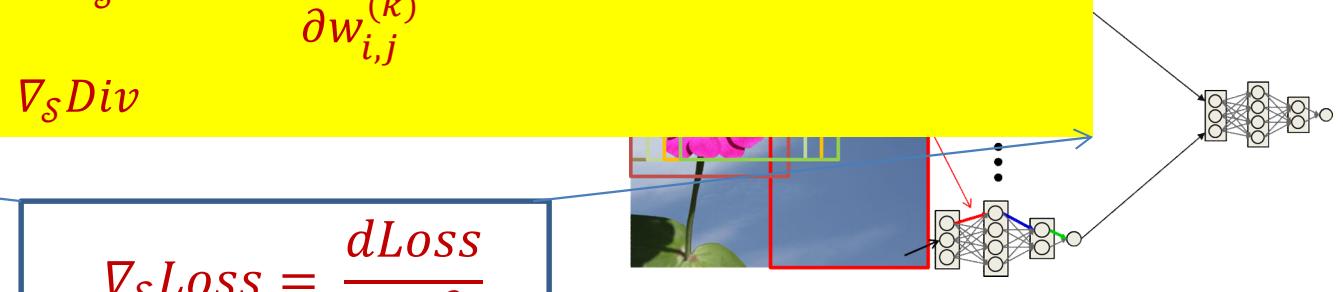
$$\nabla_{\mathcal{S}} \text{Loss} = \frac{d \text{Loss}}{d w^{\mathcal{S}}}$$

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

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

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

- Until  $\text{Loss}$  has converged



# Training networks with shared parameters

- For every training instance  $X$

- For every set  $\mathcal{S}$ :

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

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

Computed by Backprop

- $\nabla_{\mathcal{S}} \text{Loss} += \nabla_{\mathcal{S}} \text{Div}$

- Compute:

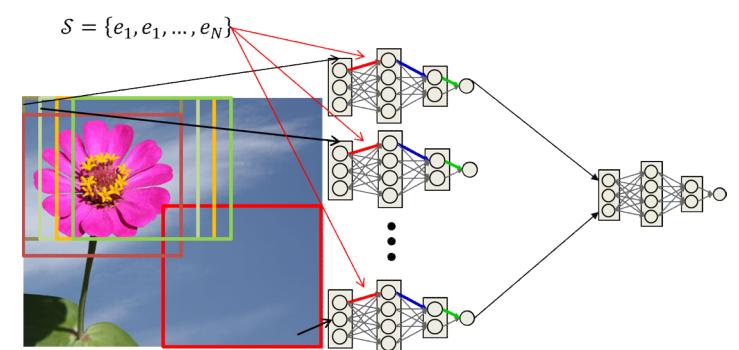
$$\nabla_{\mathcal{S}} \text{Loss} = \frac{d \text{Loss}}{d w^{\mathcal{S}}}$$

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

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

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

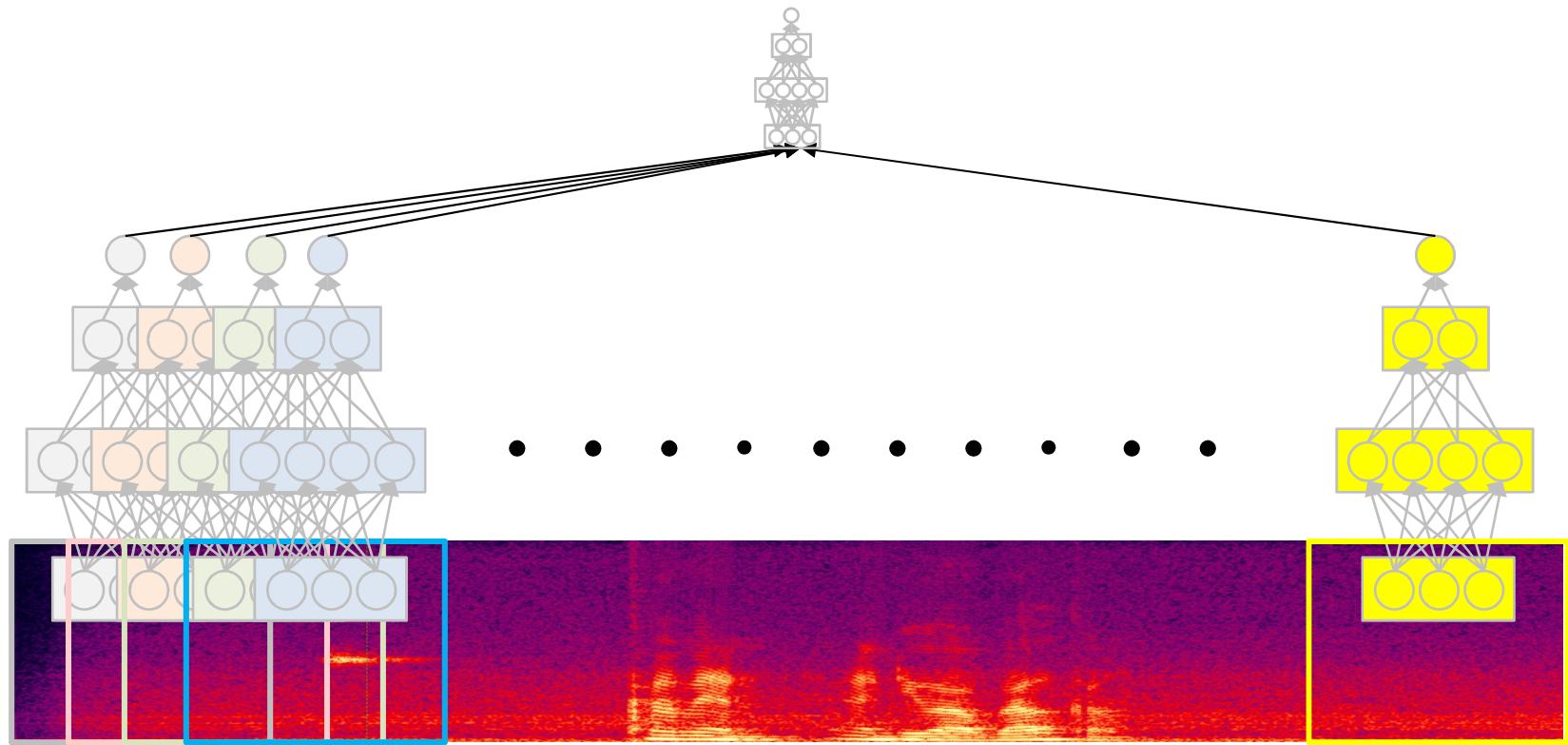
- Until  $\text{Loss}$  has converged



# Story so far

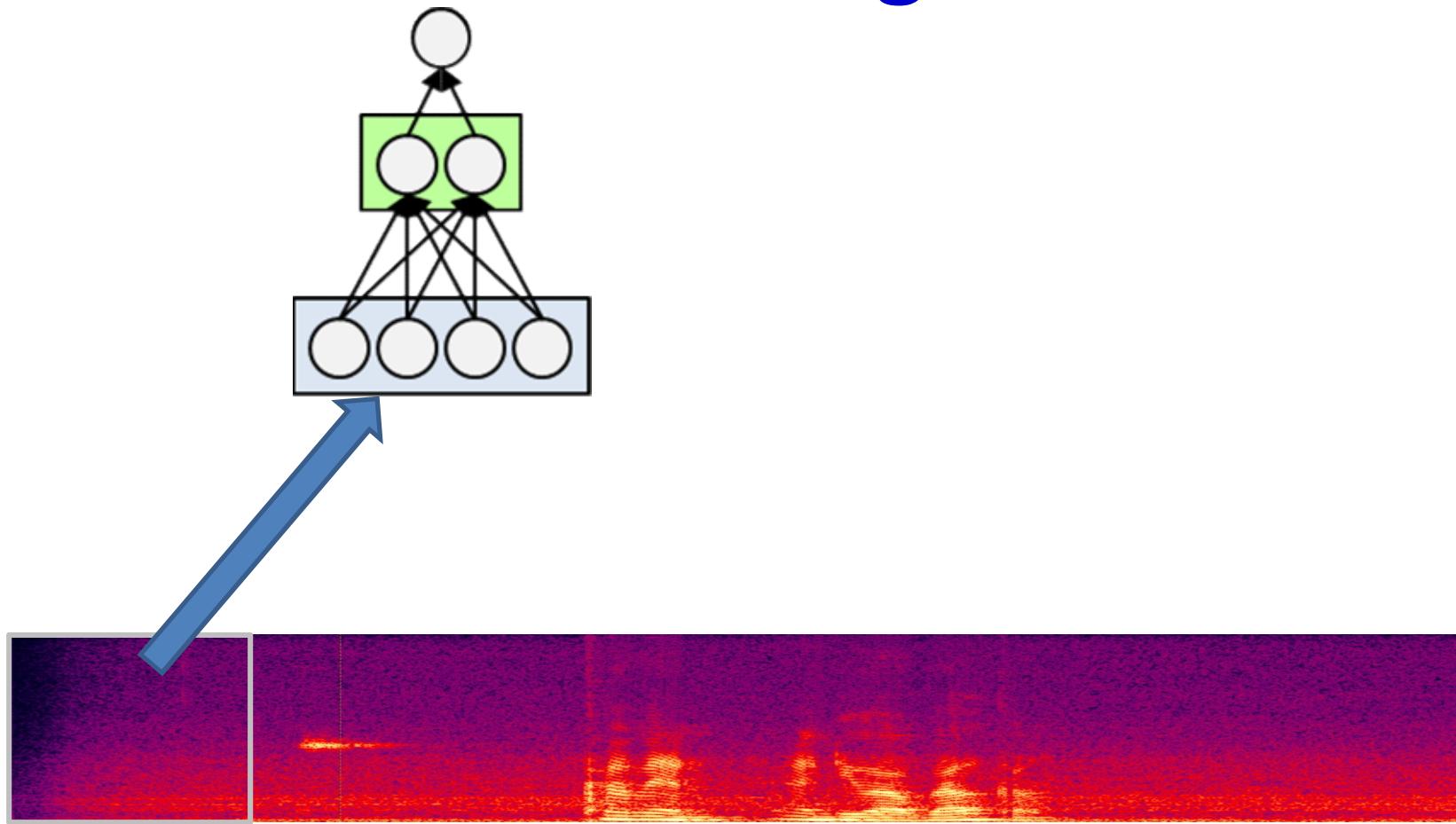
- Position-invariant pattern classification can be performed by scanning
  - 1-D scanning for sound
  - 2-D scanning for images
  - 3-D and higher-dimensional scans for higher dimensional data
- Scanning is equivalent to composing a large network with repeating subnets
  - The large network has shared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



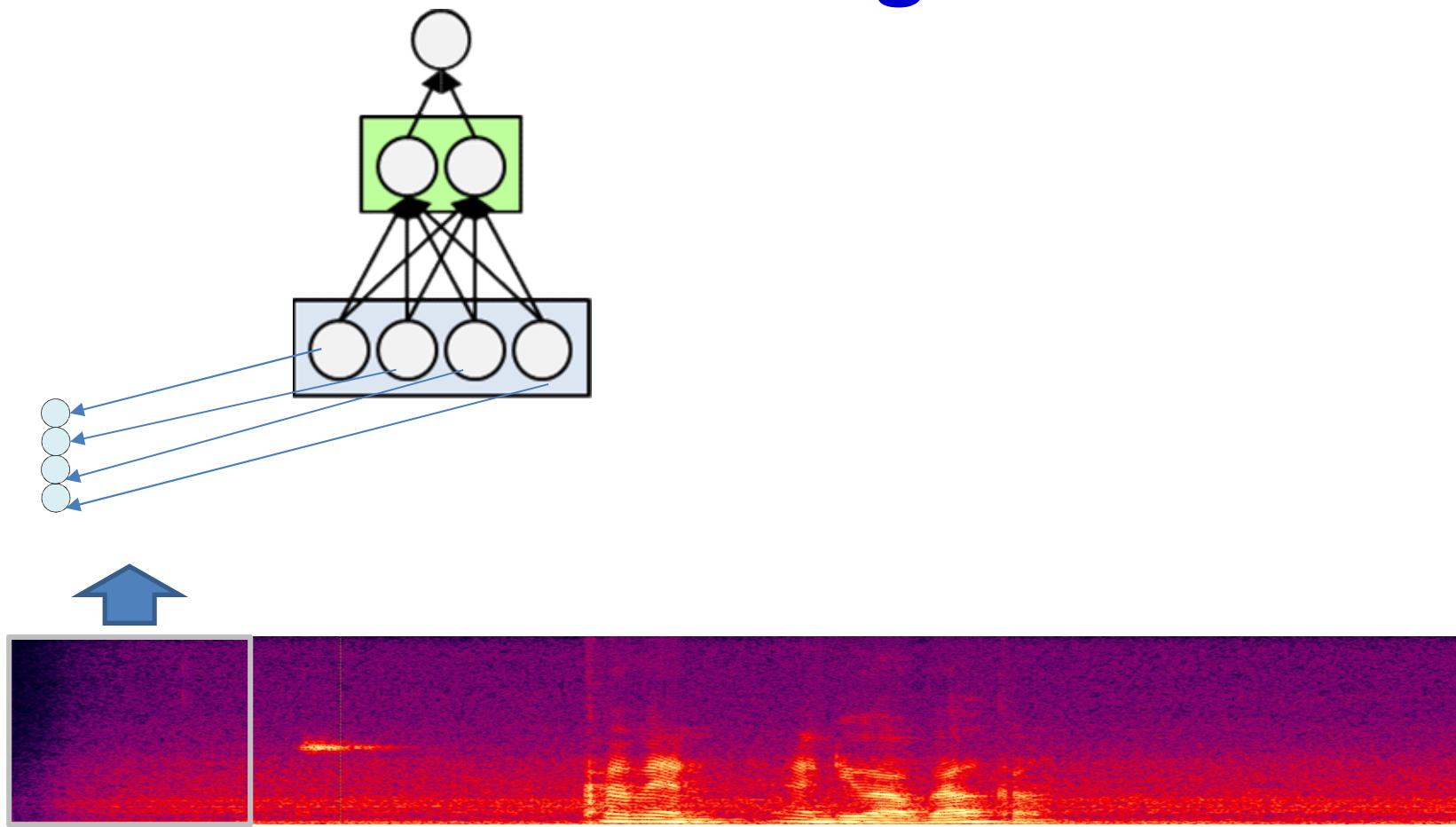
- The entire MLP operates on each “window” of the input

# 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

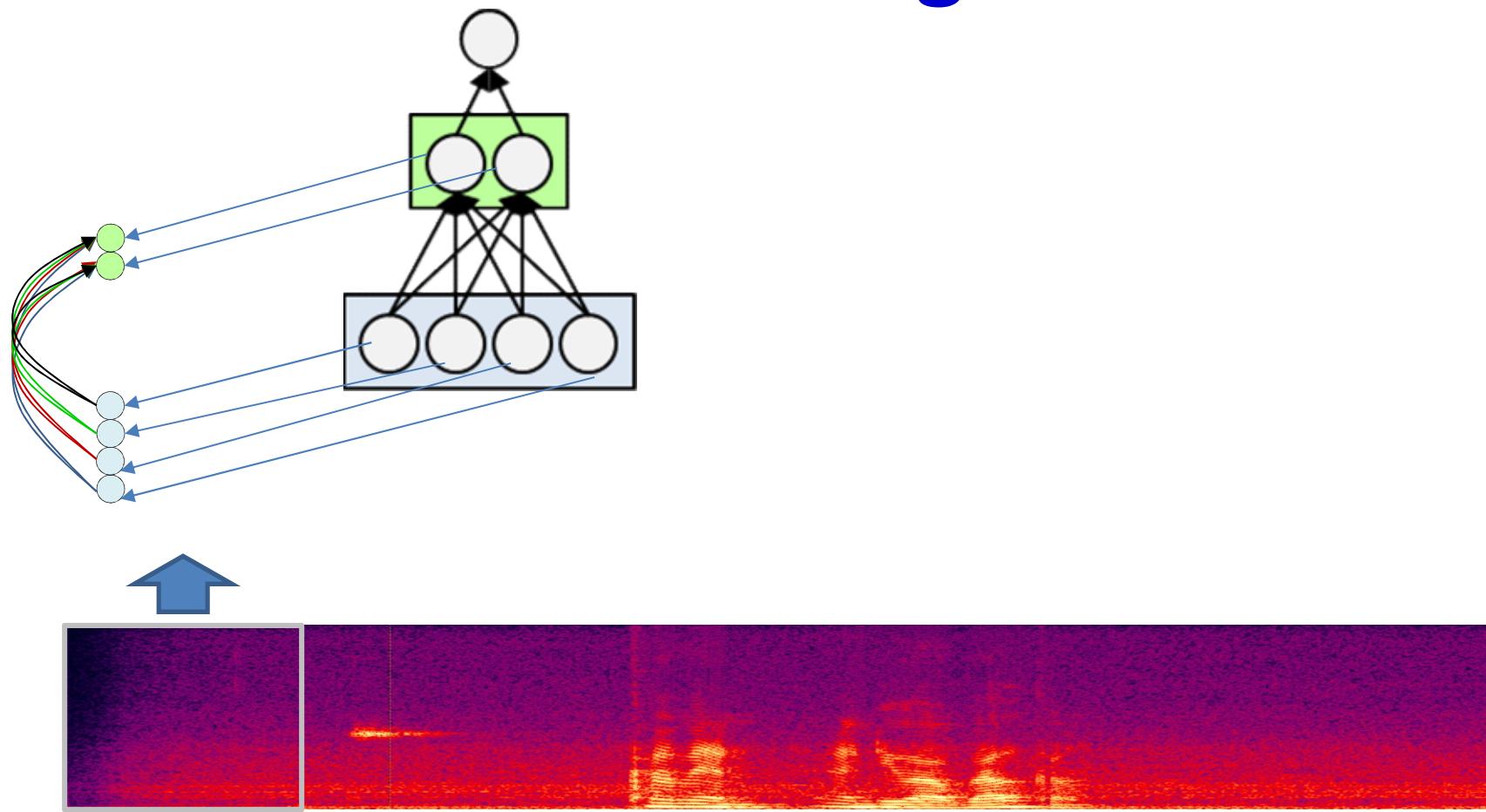
# Scanning



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

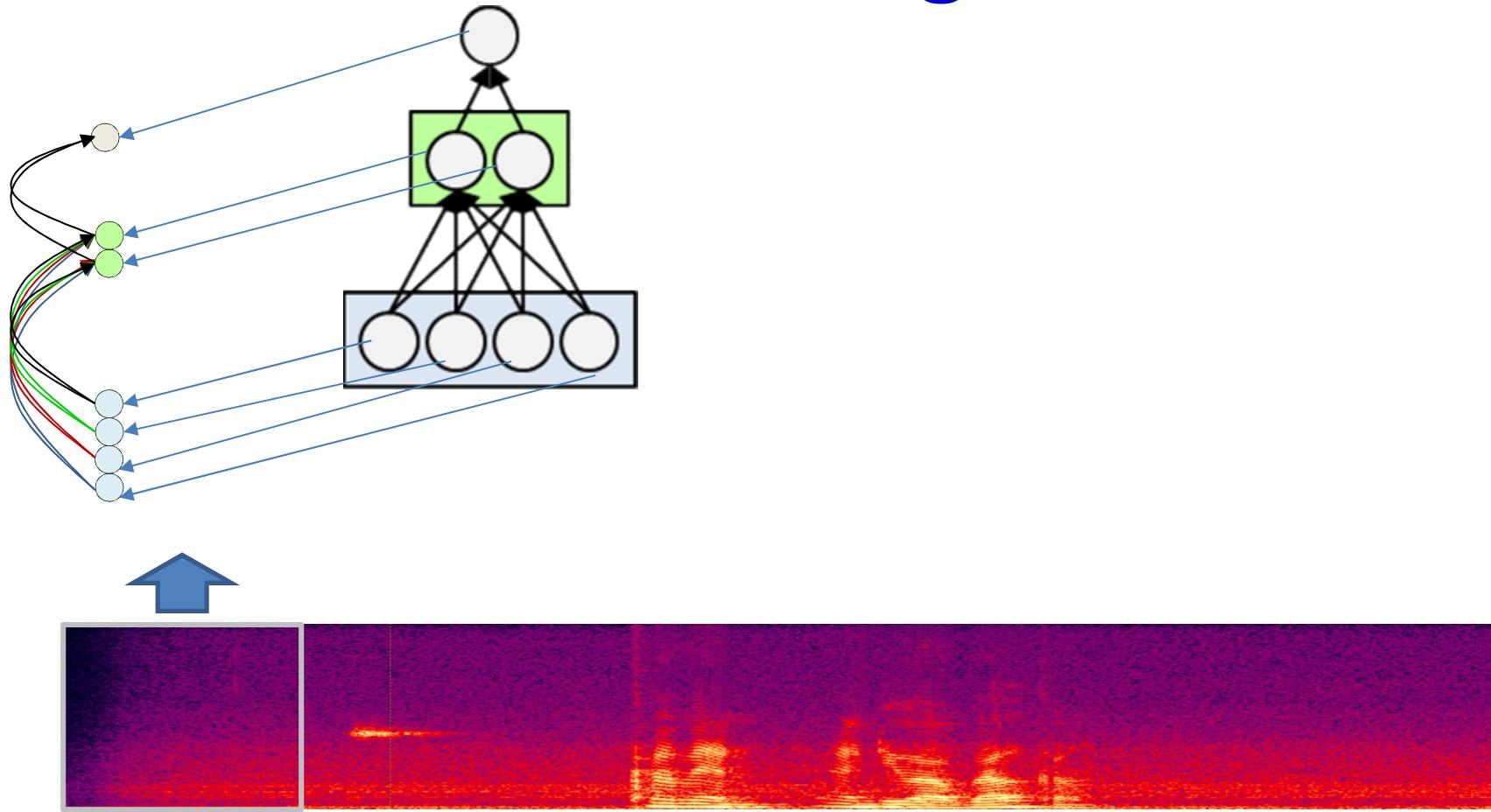
—

# Scanning



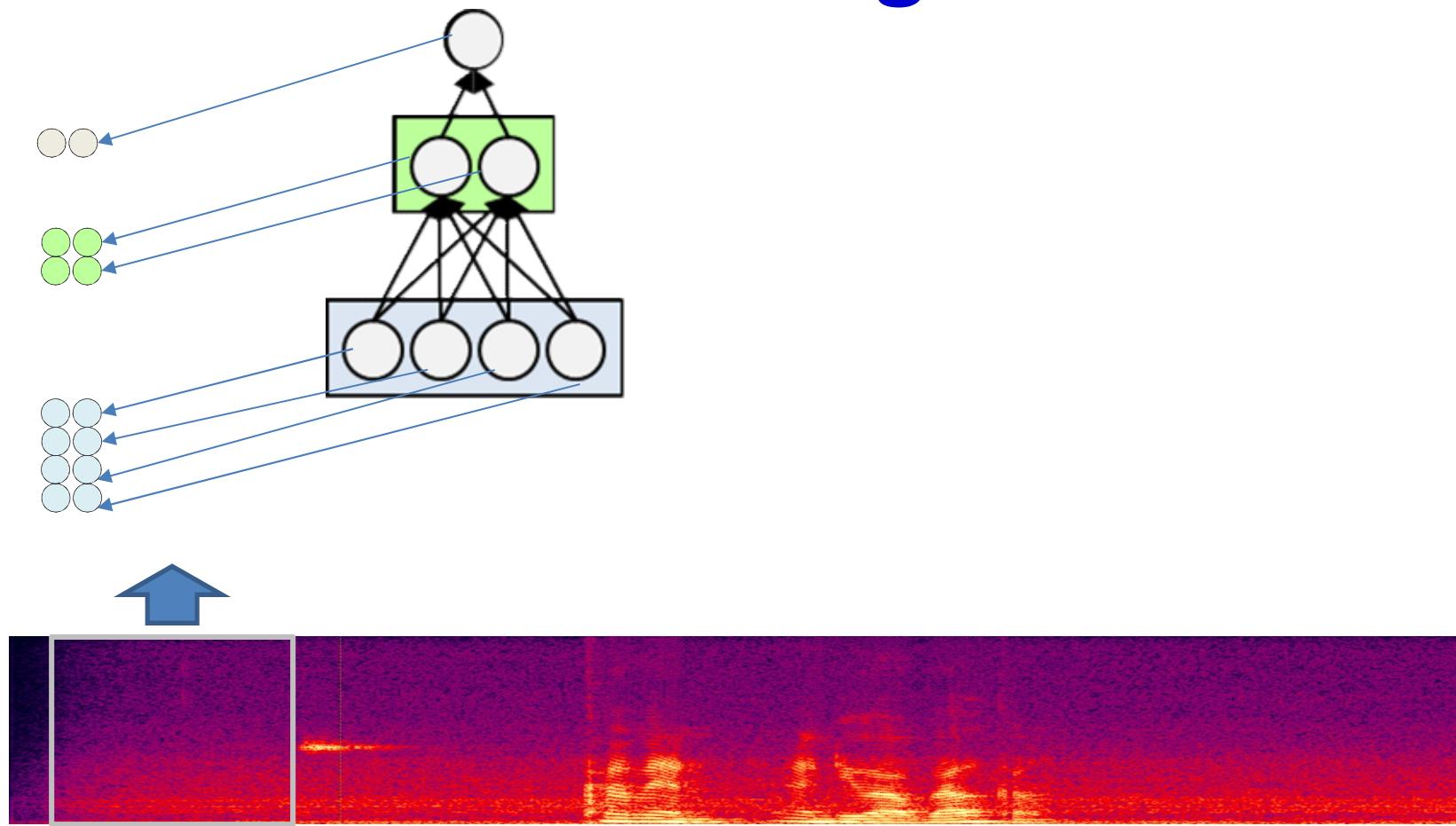
- At each location, each neuron computes a value based on its inputs
  - Which may either be the input image or the outputs of the previous layer

# Scanning



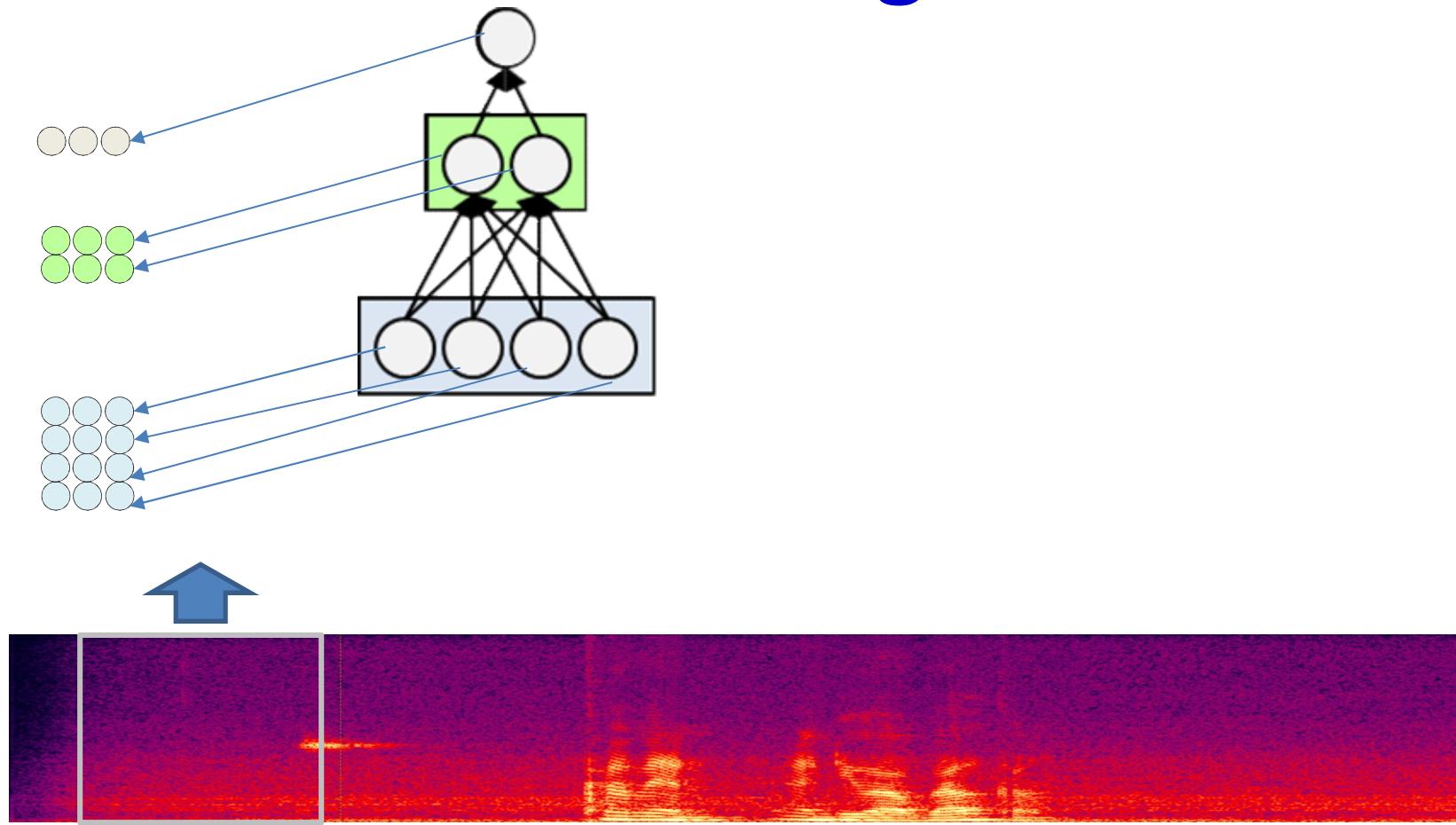
- At each location, each neuron computes a value based on its inputs
  - Which may either be the input image or the outputs of the previous layer

# Scanning



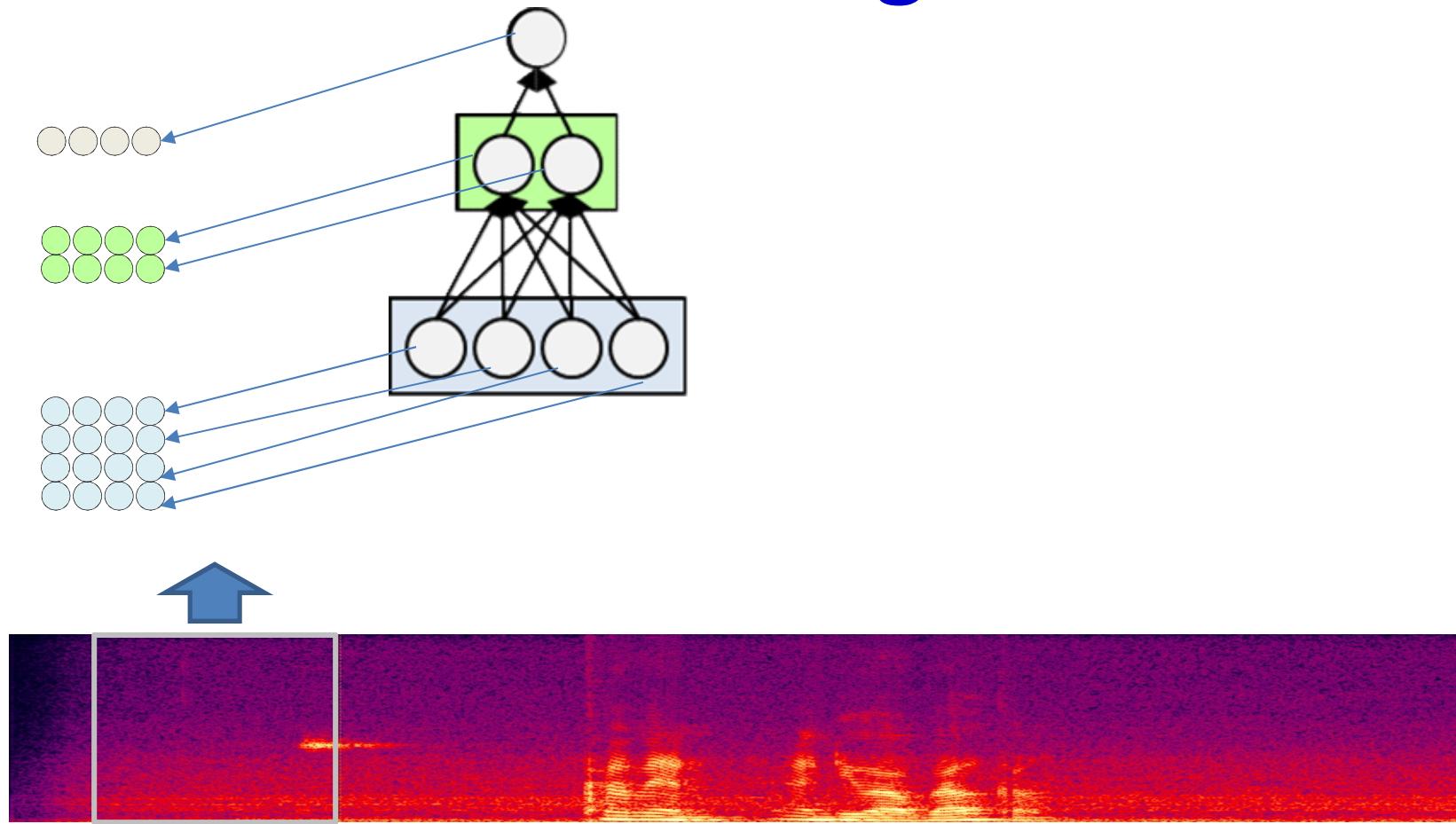
- The same sequence of computations is performed at each location
  - Producing similar sets of values
    - One value per neuron in each layer

# Scanning



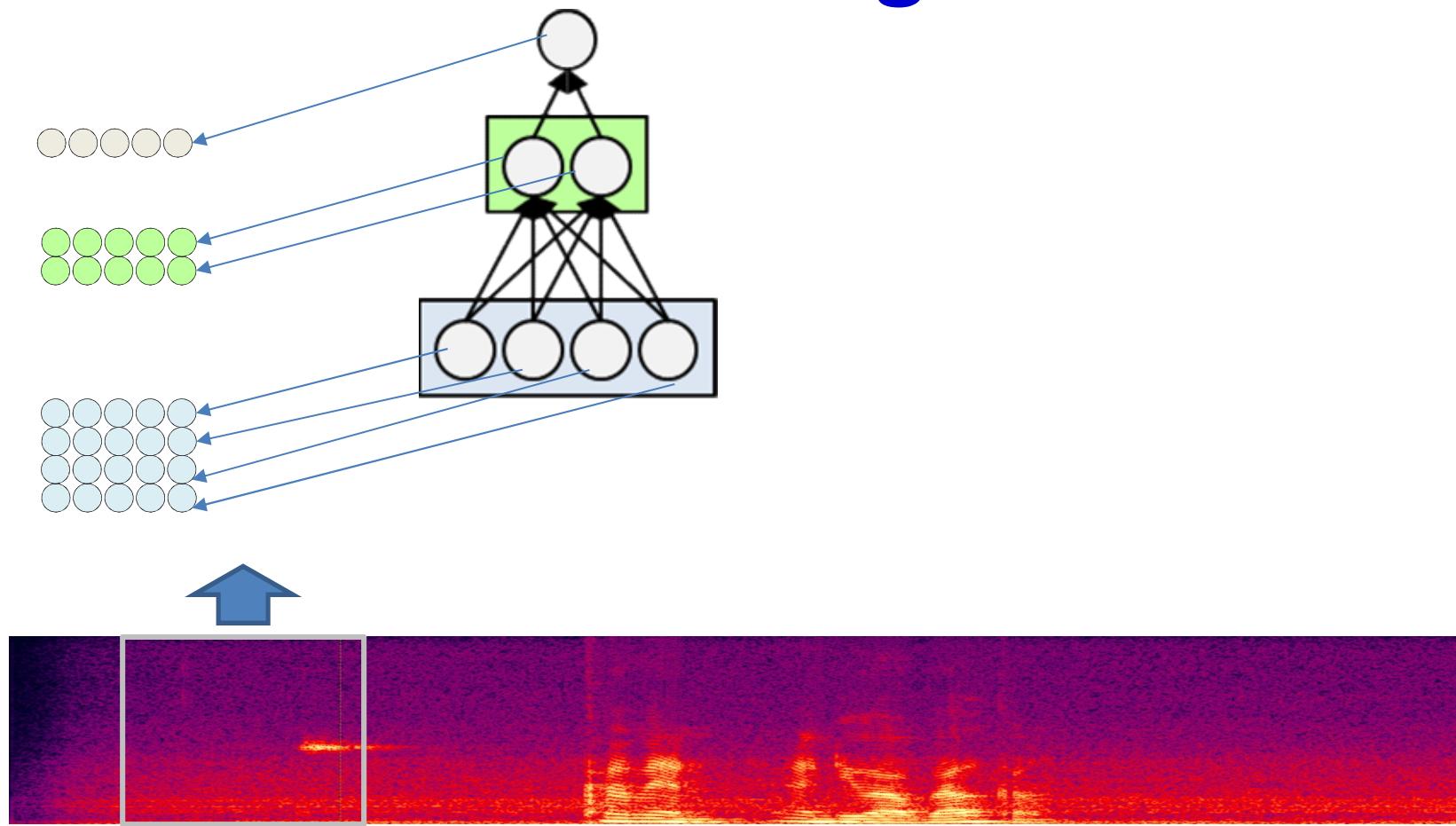
- The same sequence of computations is performed at each location
  - Producing similar sets of values
    - One value per neuron in each layer

# Scanning



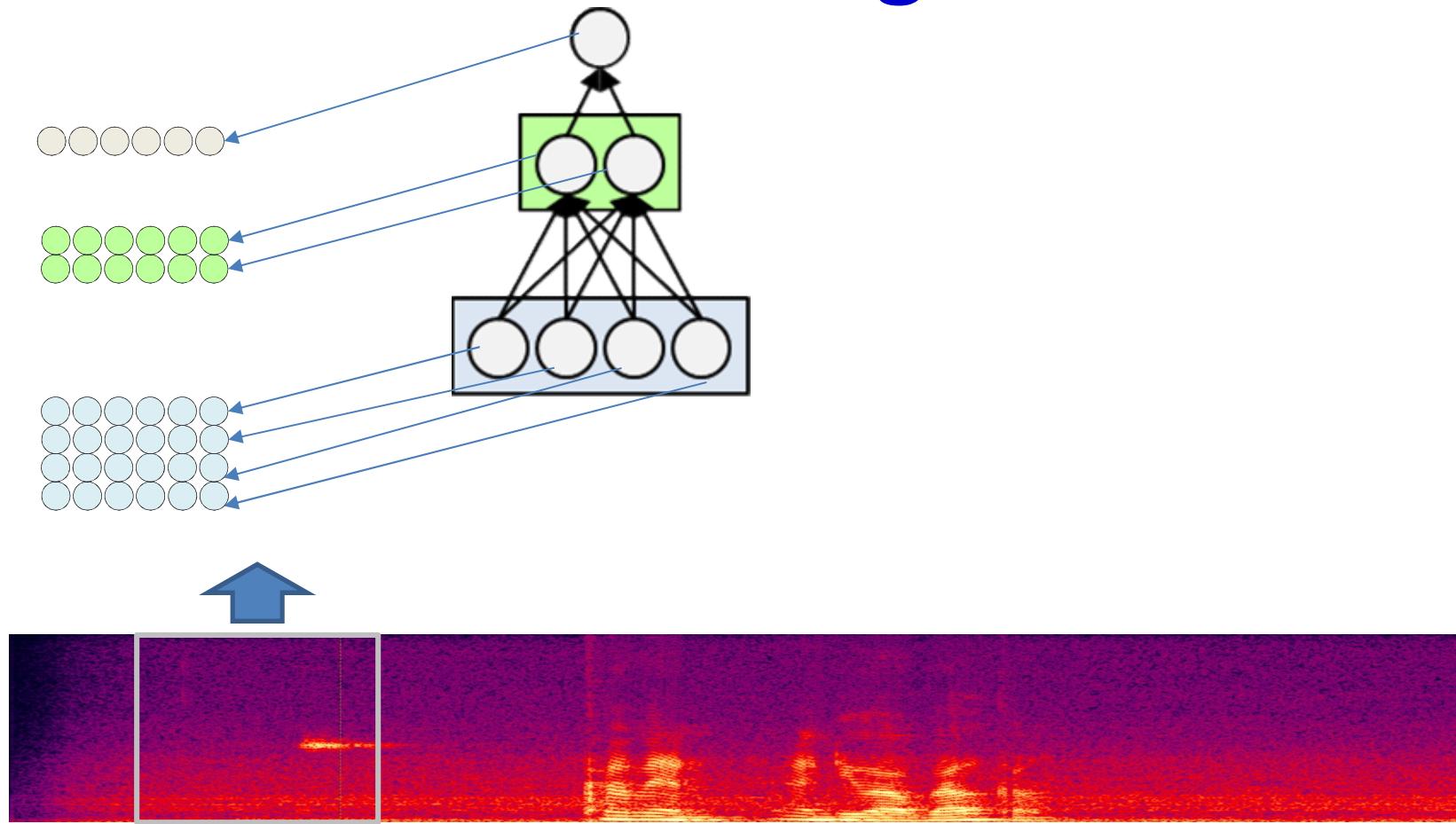
- The same sequence of computations is performed at each location
  - Producing similar sets of values
    - One value per neuron in each layer

# Scanning



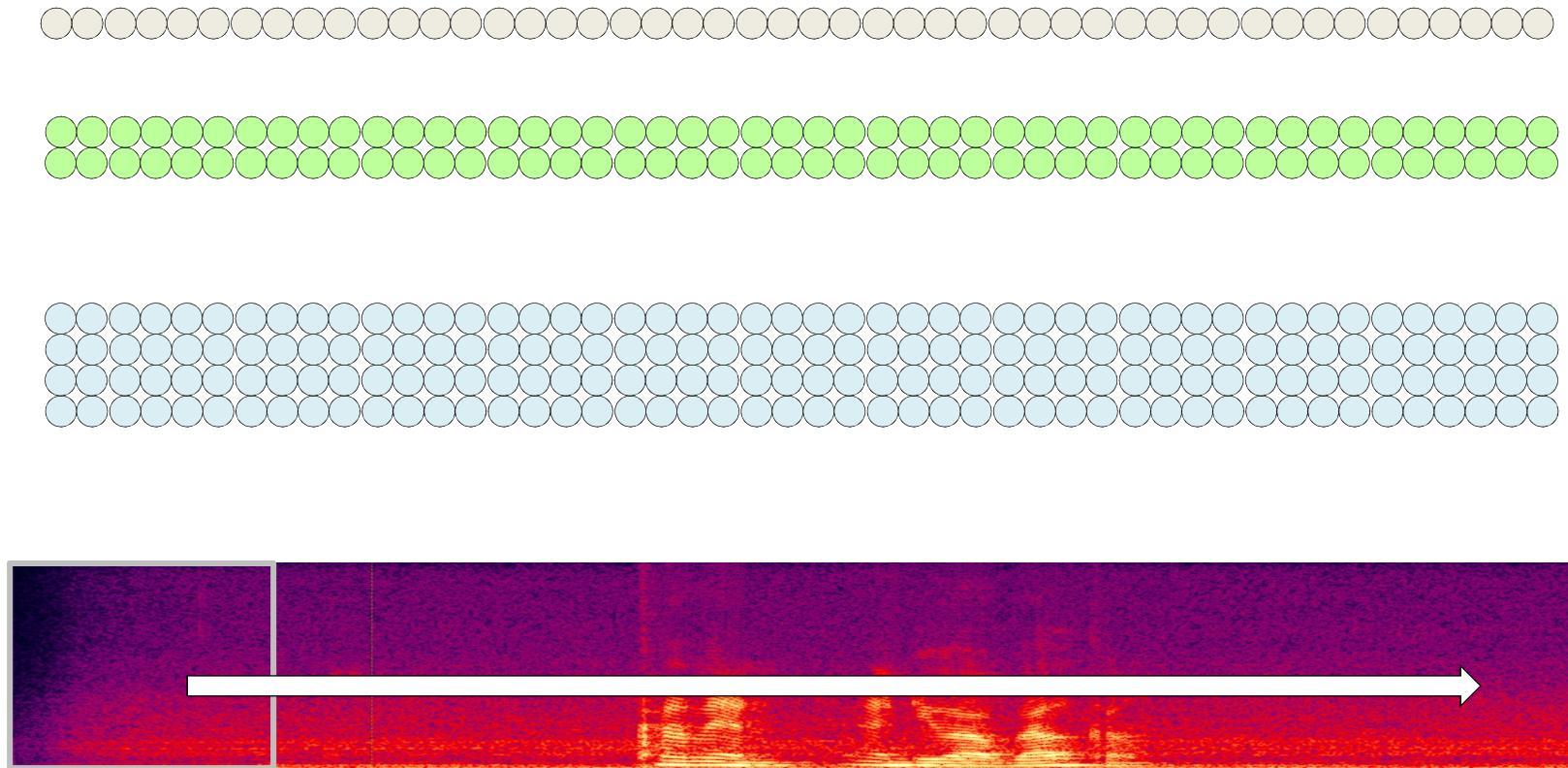
- The same sequence of computations is performed at each location
  - Producing similar sets of values
    - One value per neuron in each layer

# Scanning



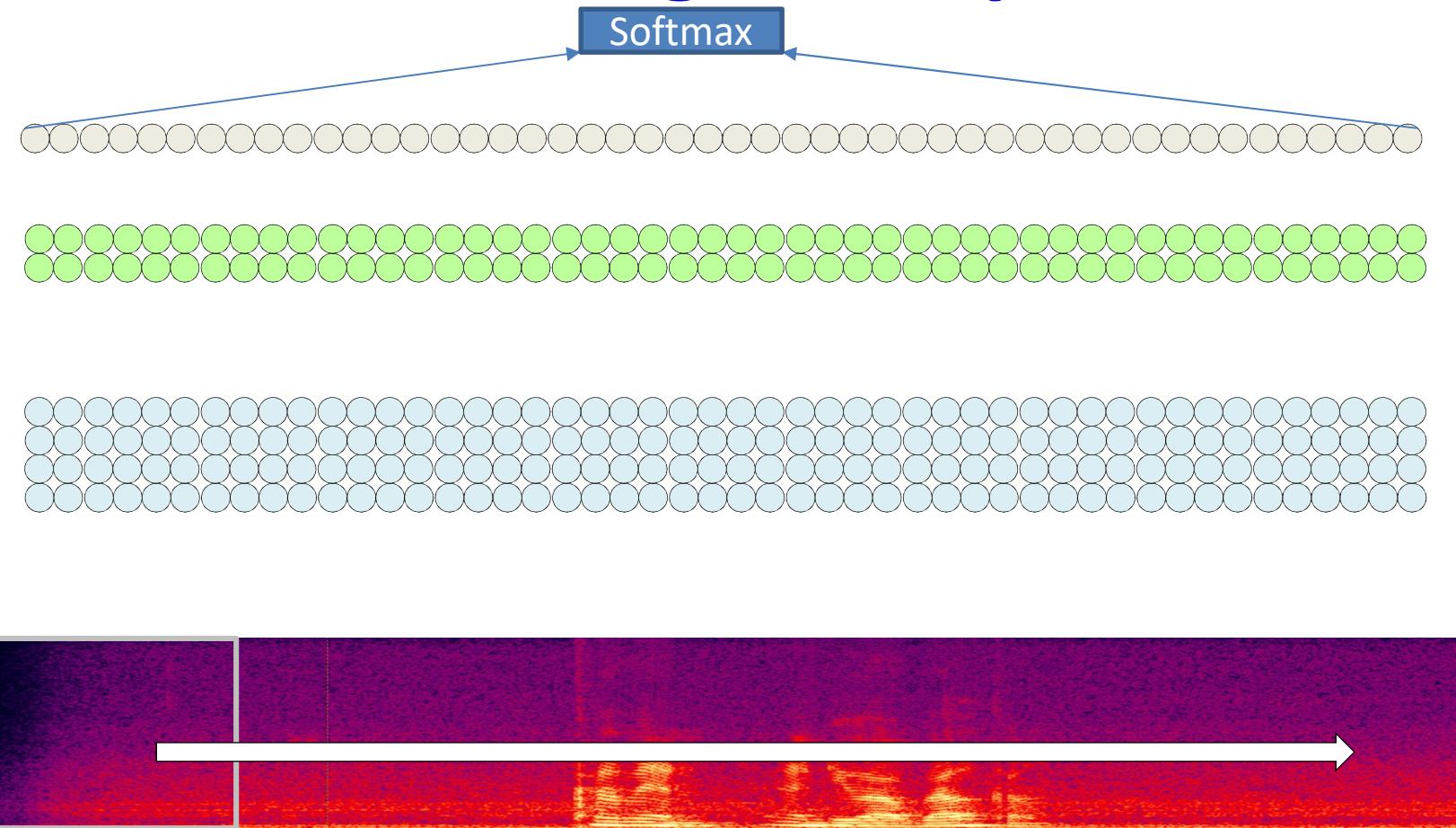
- The same sequence of computations is performed at each location
  - Producing similar sets of values
    - One value per neuron in each layer

# Scanning the input



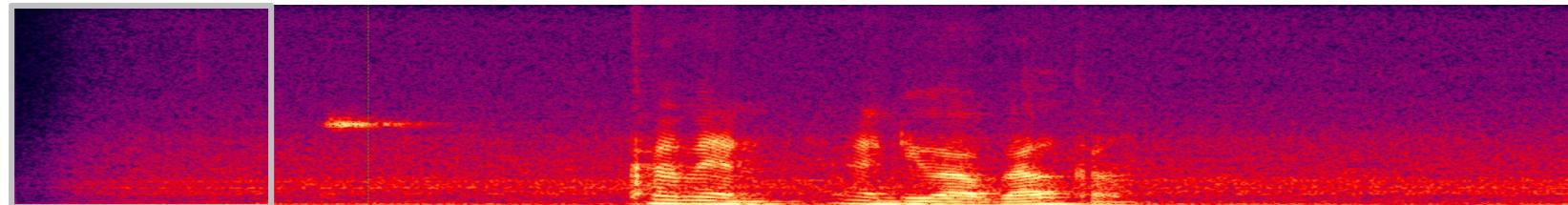
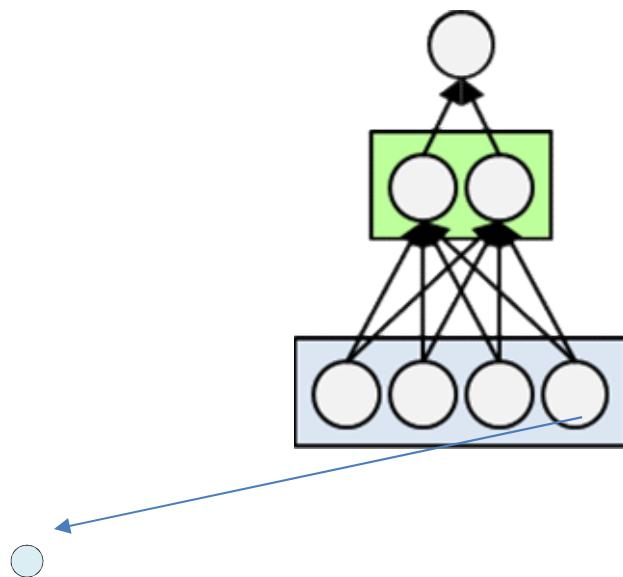
- We get a complete set of values (represented as a column) at each location evaluated by the MLP during the scan

# Scanning the input



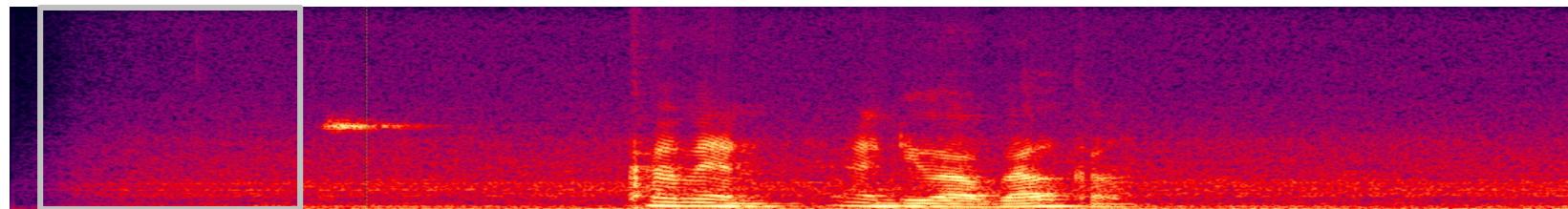
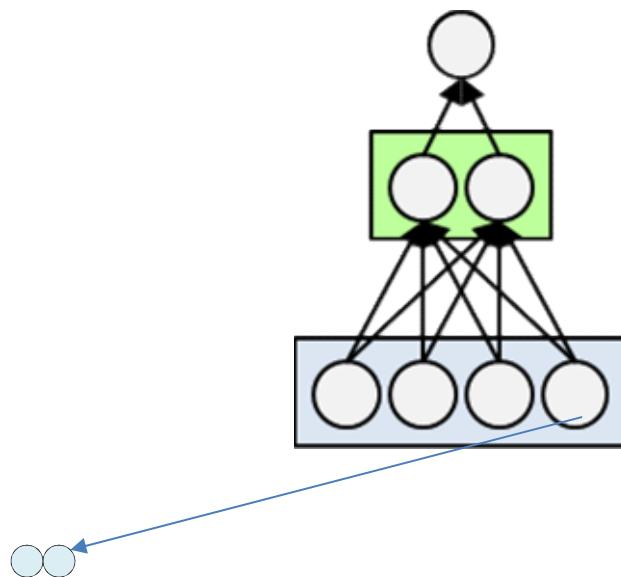
- We get a complete set of values (represented as a column) at each location evaluated by the MLP during the scan
  - Which we put through our final softmax to decide if the recording includes the word “Welcome”

# Lets do it in an different order



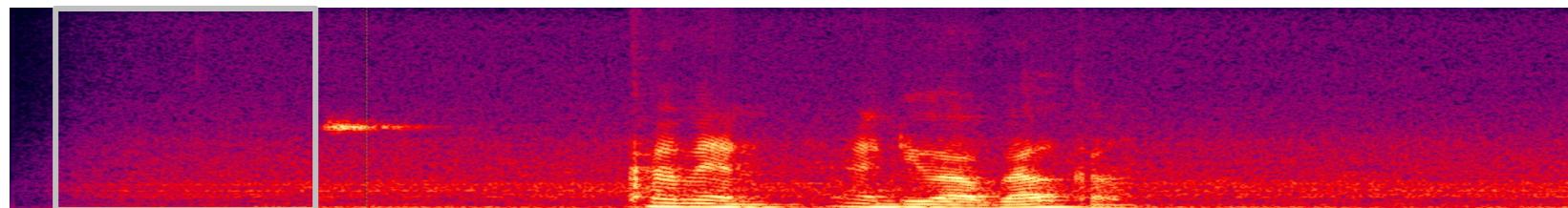
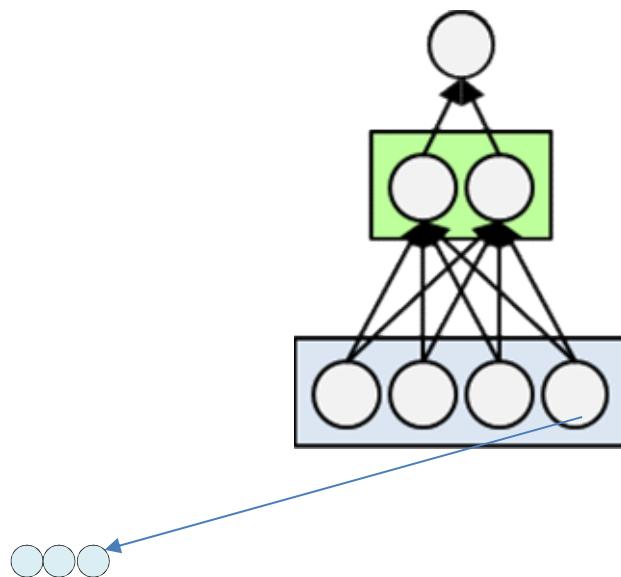
- Let us do the computation in a different order
- The first neuron evaluates each image first
  - “Scans” the input

# Lets do it in an different order



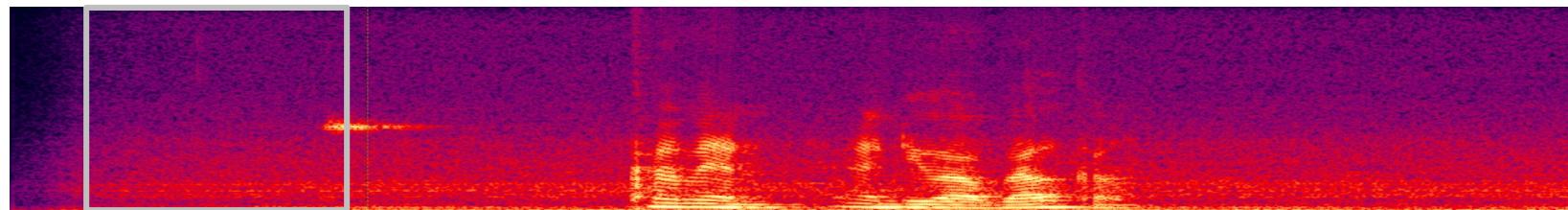
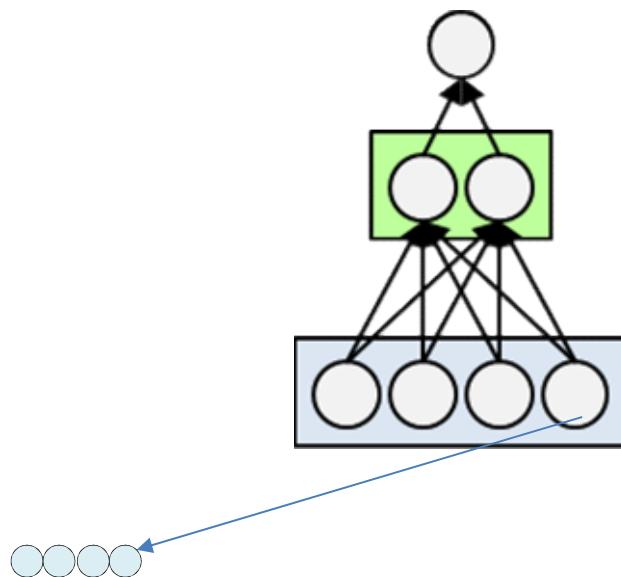
- Let us do the computation in a different order
- The first neuron evaluates each image first
  - “Scans” the input

# Lets do it in an different order



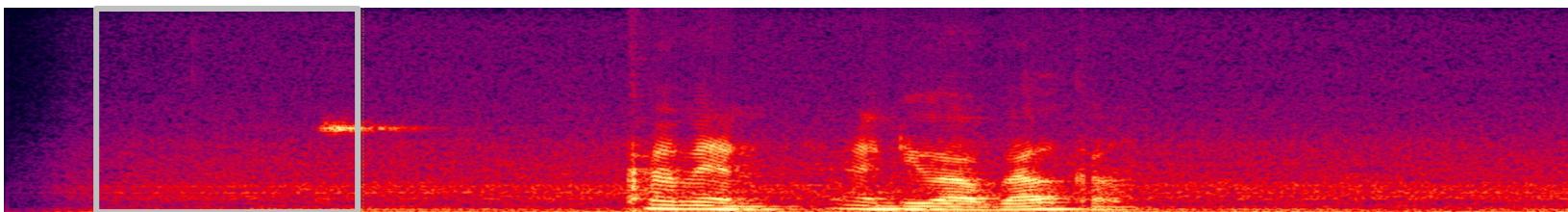
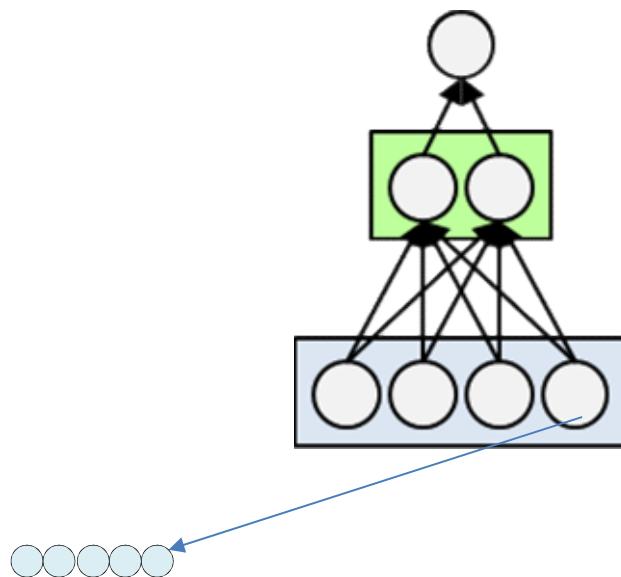
- Let us do the computation in a different order
- The first neuron evaluates each image first
  - “Scans” the input

# Lets do it in an different order



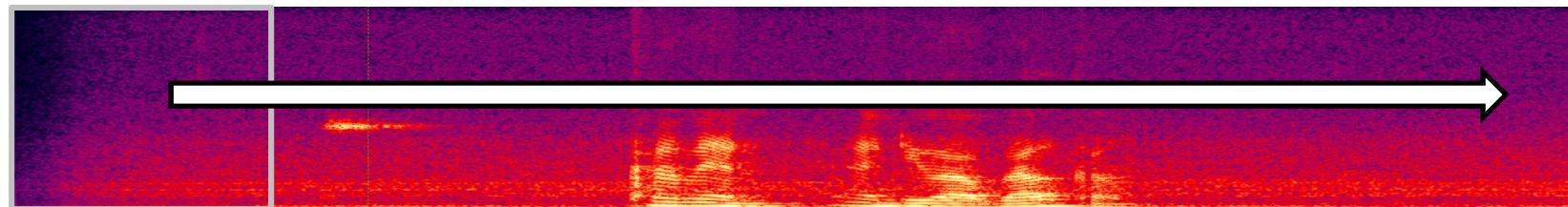
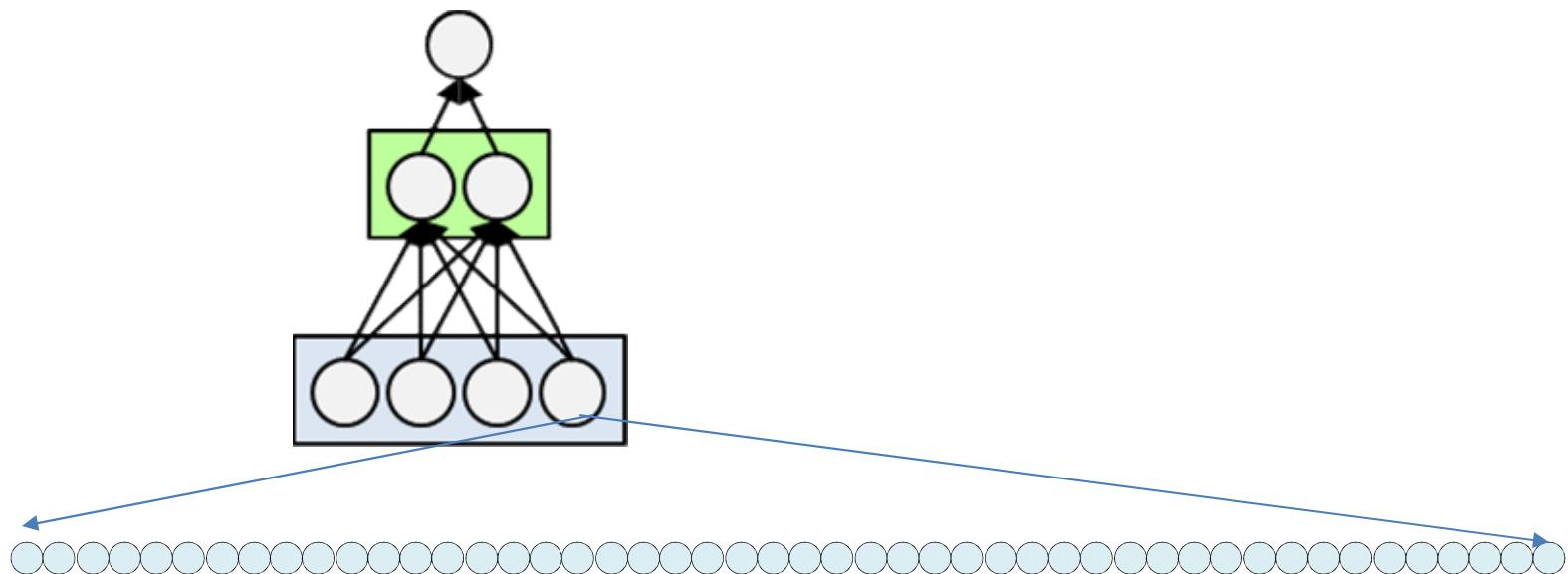
- Let us do the computation in a different order
- The first neuron evaluates each image first
  - “Scans” the input

# Lets do it in an different order



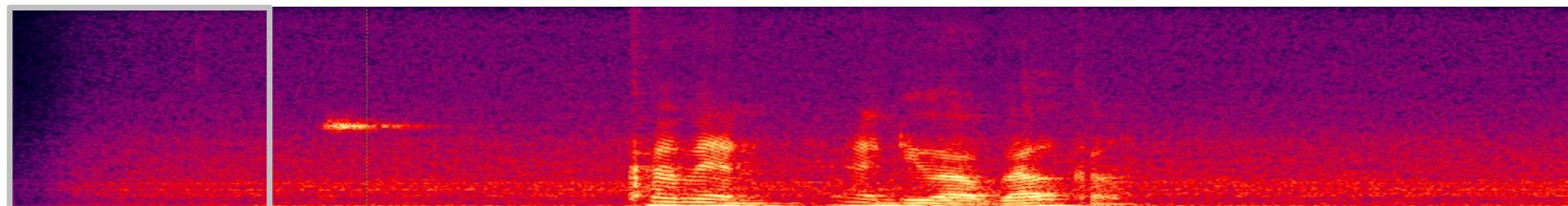
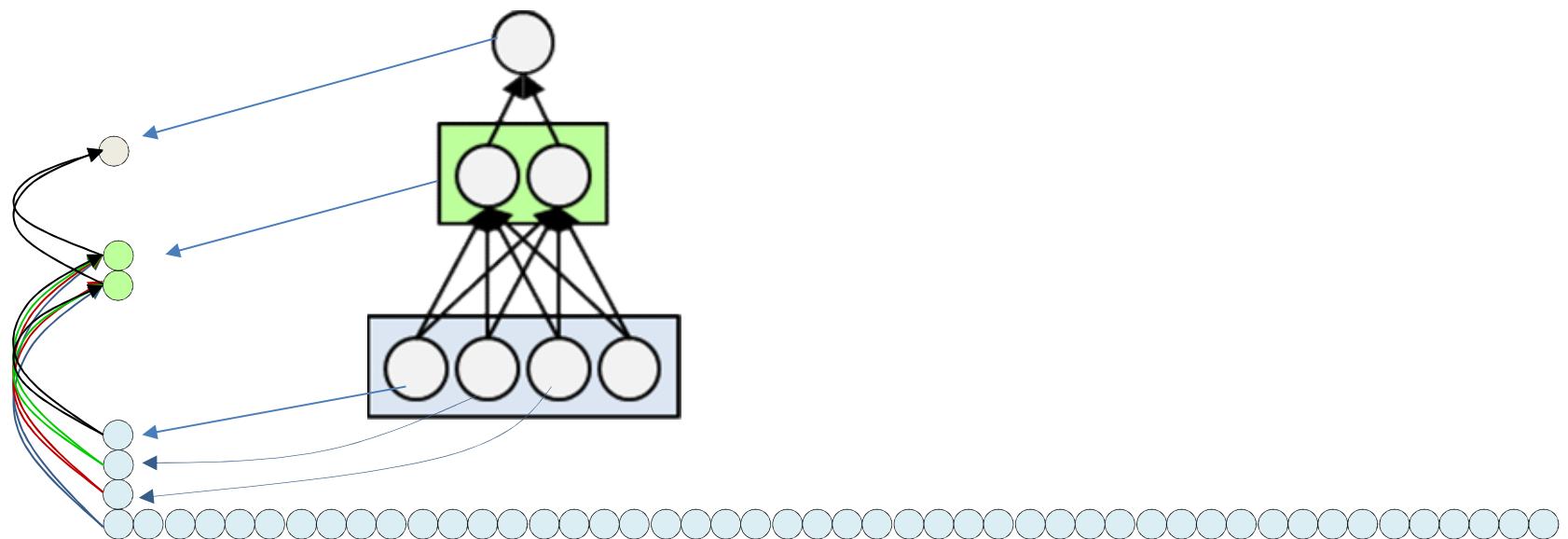
- Let us do the computation in a different order
- The first neuron evaluates each image first
  - “Scans” the input

# Lets do it in an different order



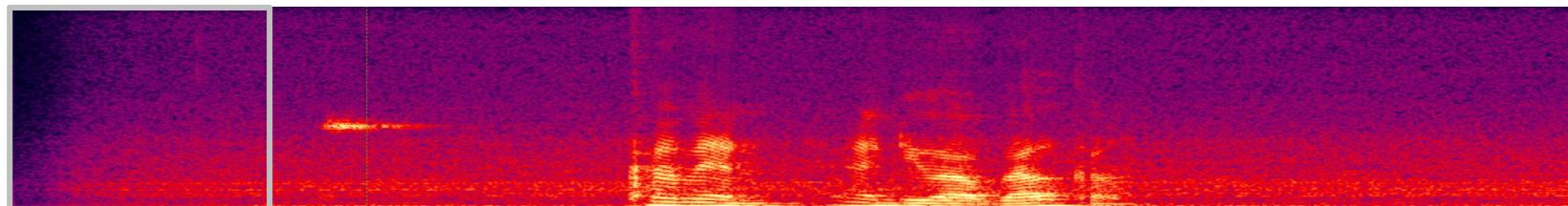
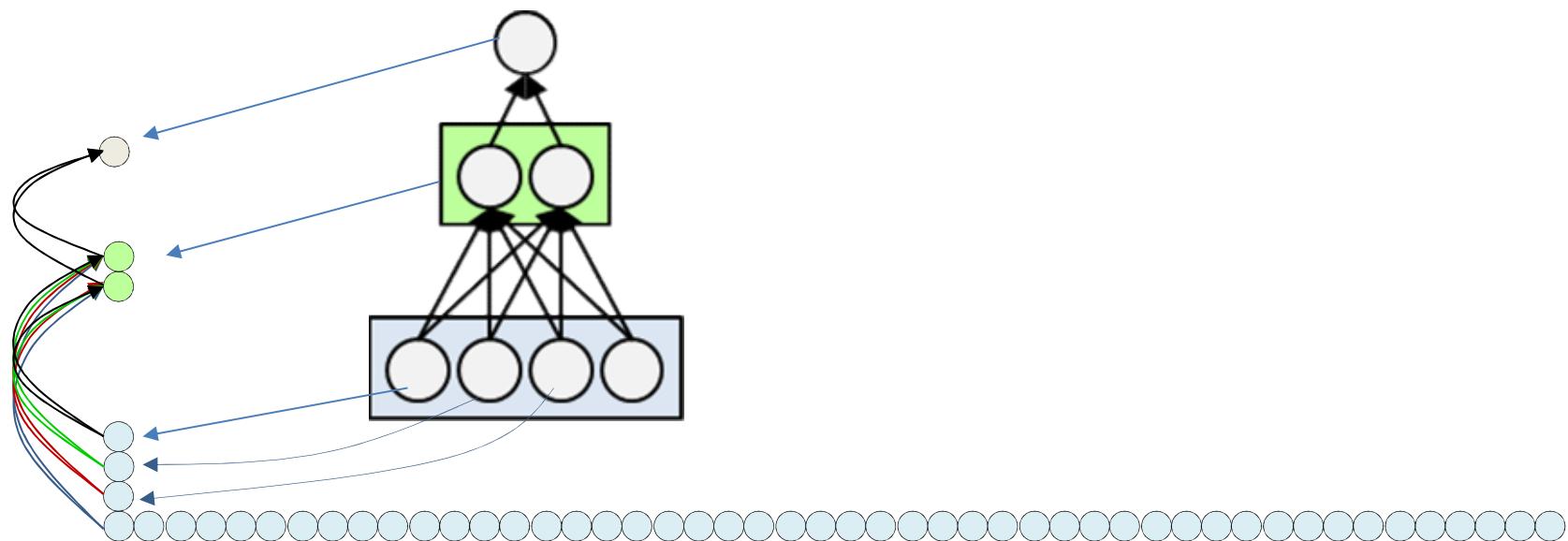
- Let us do the computation in a different order
- The first neuron evaluates each image first
  - “Scans” the input

# Lets do it in an different order



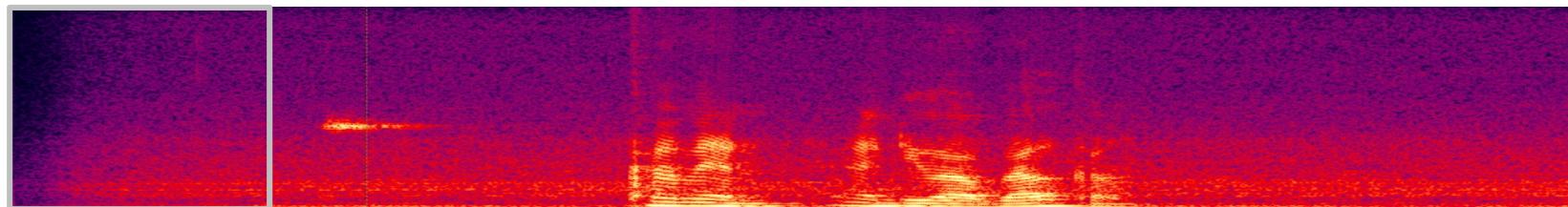
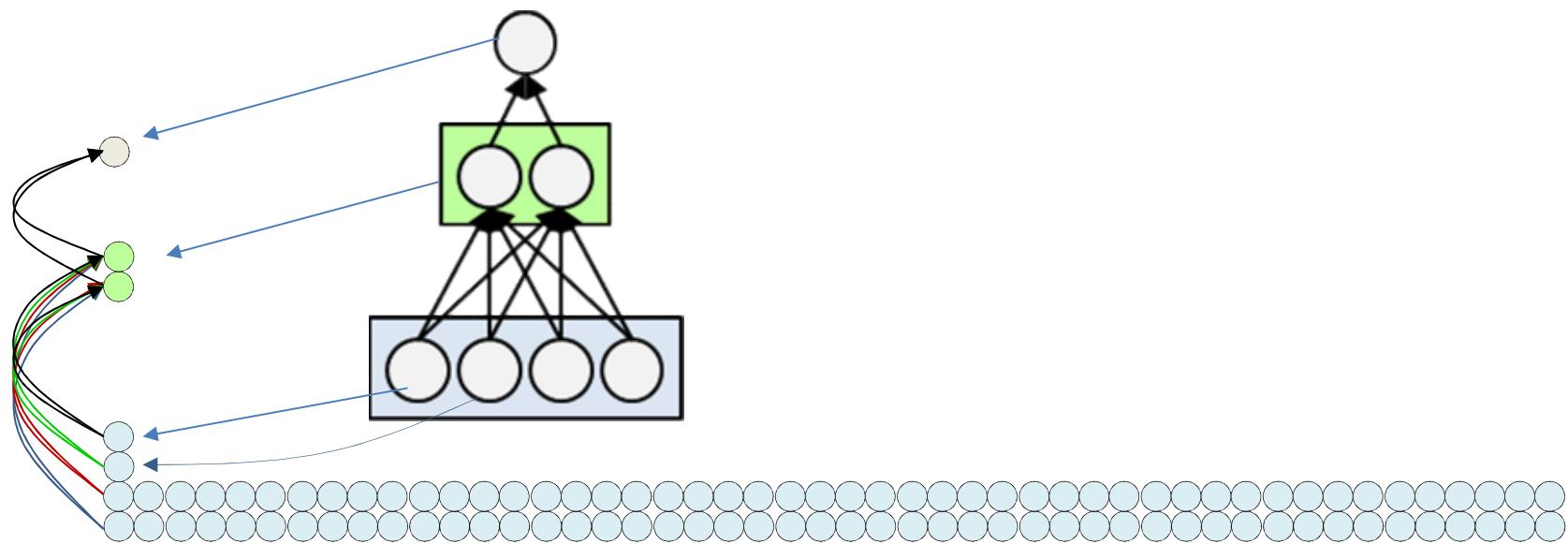
- Subsequently the rest of the neurons in the first layer operate on the first block
  - And the downstream layers as well
- Would the output of the MLP at the first block be different?

# Lets do it in an different order



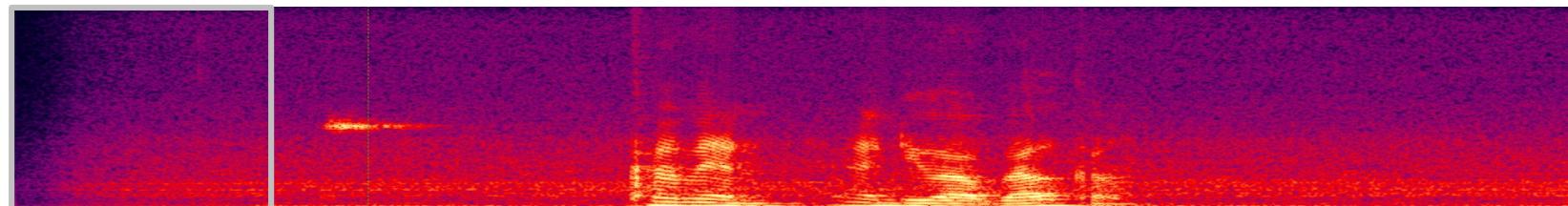
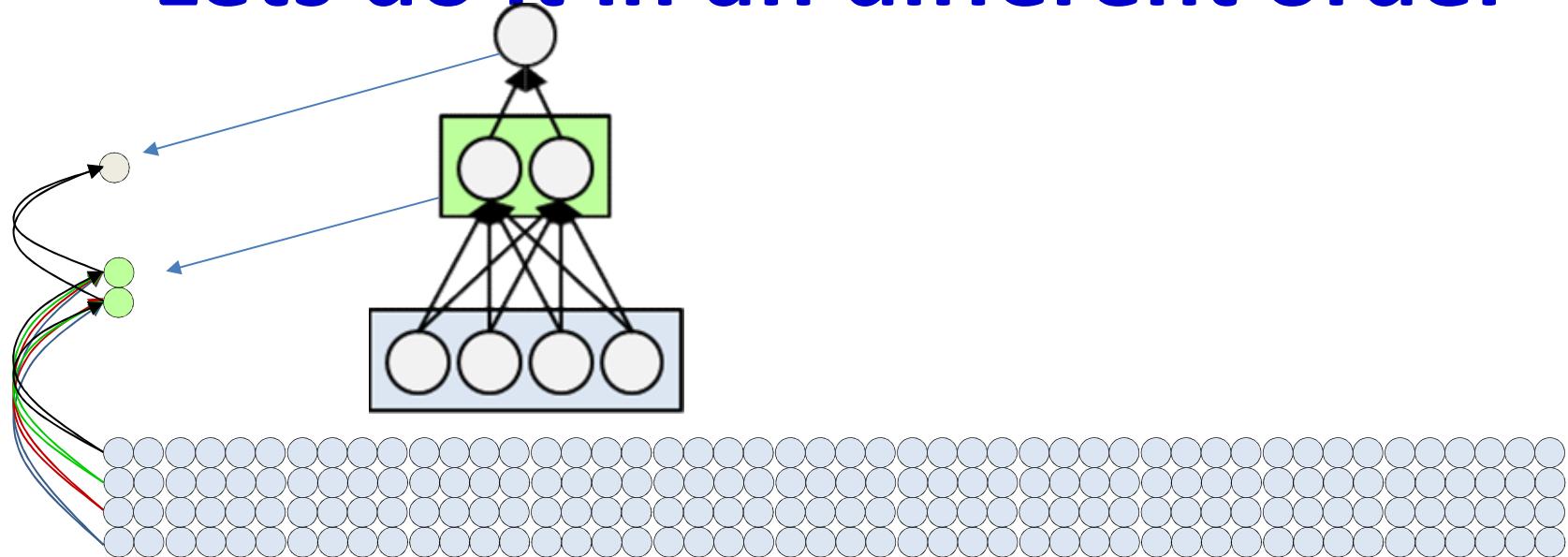
- Subsequently the rest of the neurons in the first layer operate on the first block
  - And the downstream layers as well
- Would the output of the MLP at the first block be different?
  - The fact that the first neuron has already evaluated the future blocks does not affect the output of that neuron,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

# Lets do it in an different order



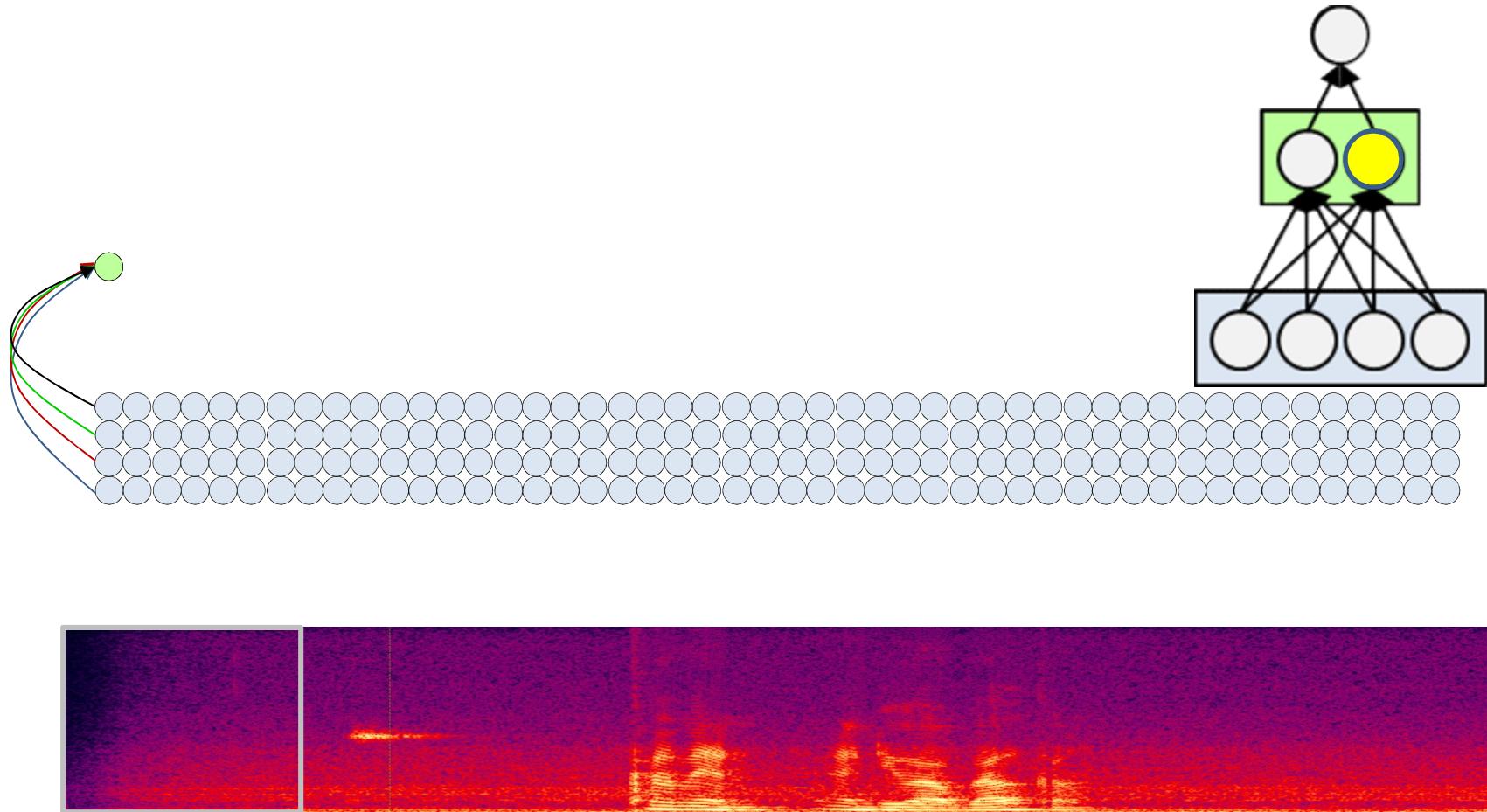
- What about now?
- The second neuron too has fully evaluated the entire input before the rest of the network evaluates the first block
  - This too should not change the output of the network for the first block

# Lets do it in an different order



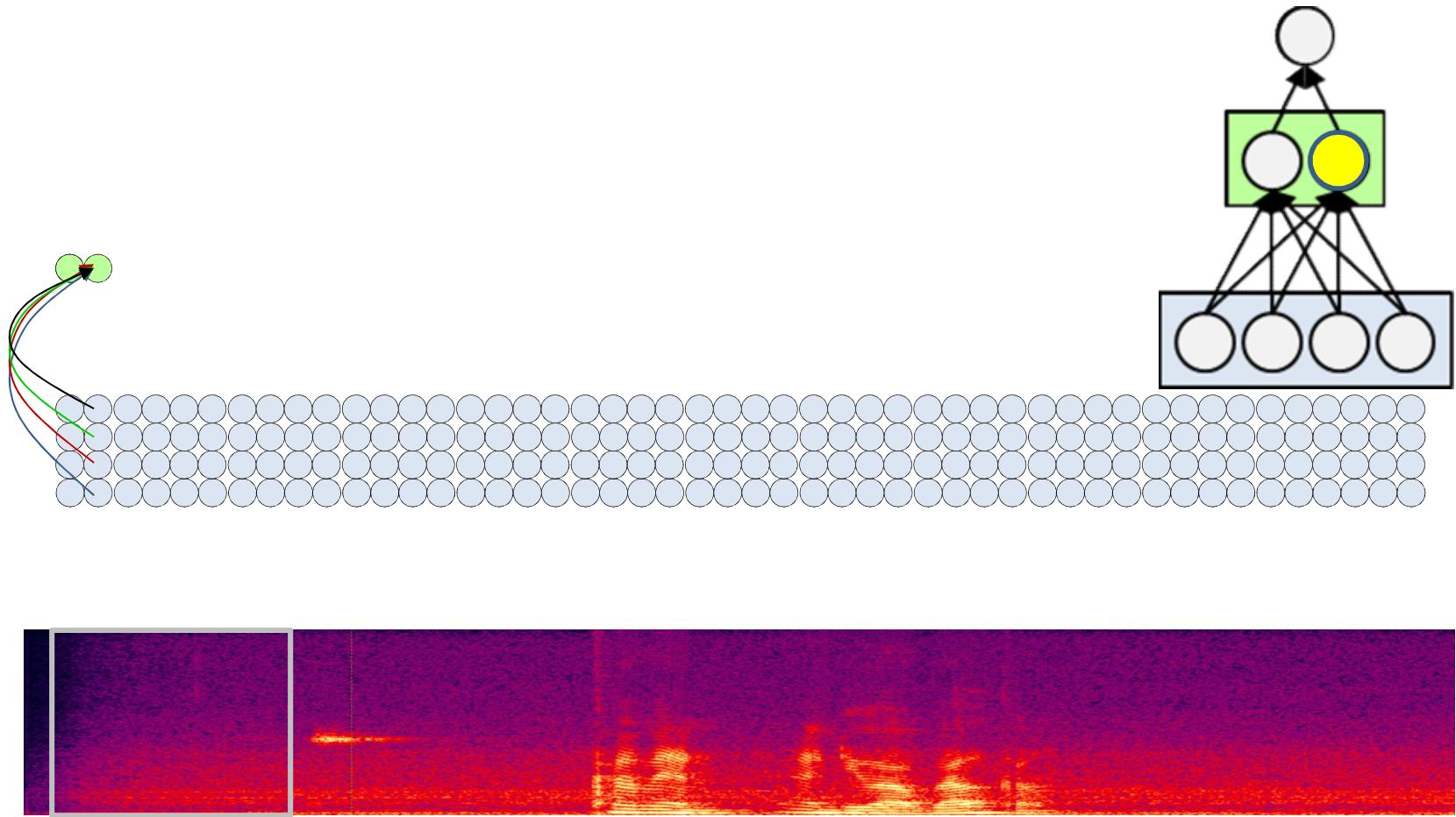
- In fact if *all* of the neurons in the first layer fully evaluate the entire input before the rest of the network evaluates the first block, this will not change the output of the network at the first block

# Lets do it in an different order



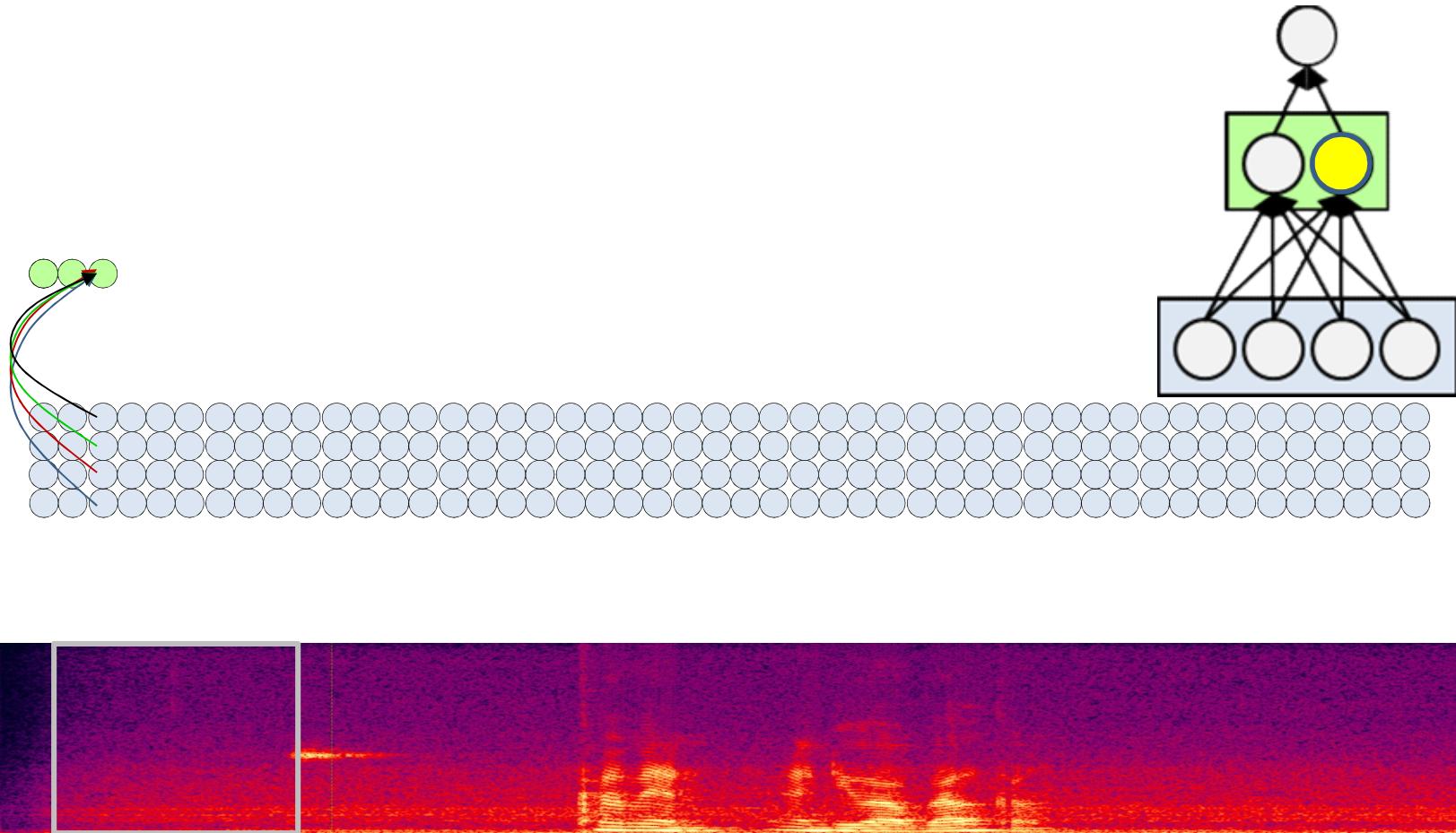
- But now, since the first layer neurons have already produced outputs for every location, each neuron in the second layer can go ahead and produce outputs for every position without waiting for the rest of the net
  - “Scan” the outputs of the first layer neurons

# Lets do it in an different order



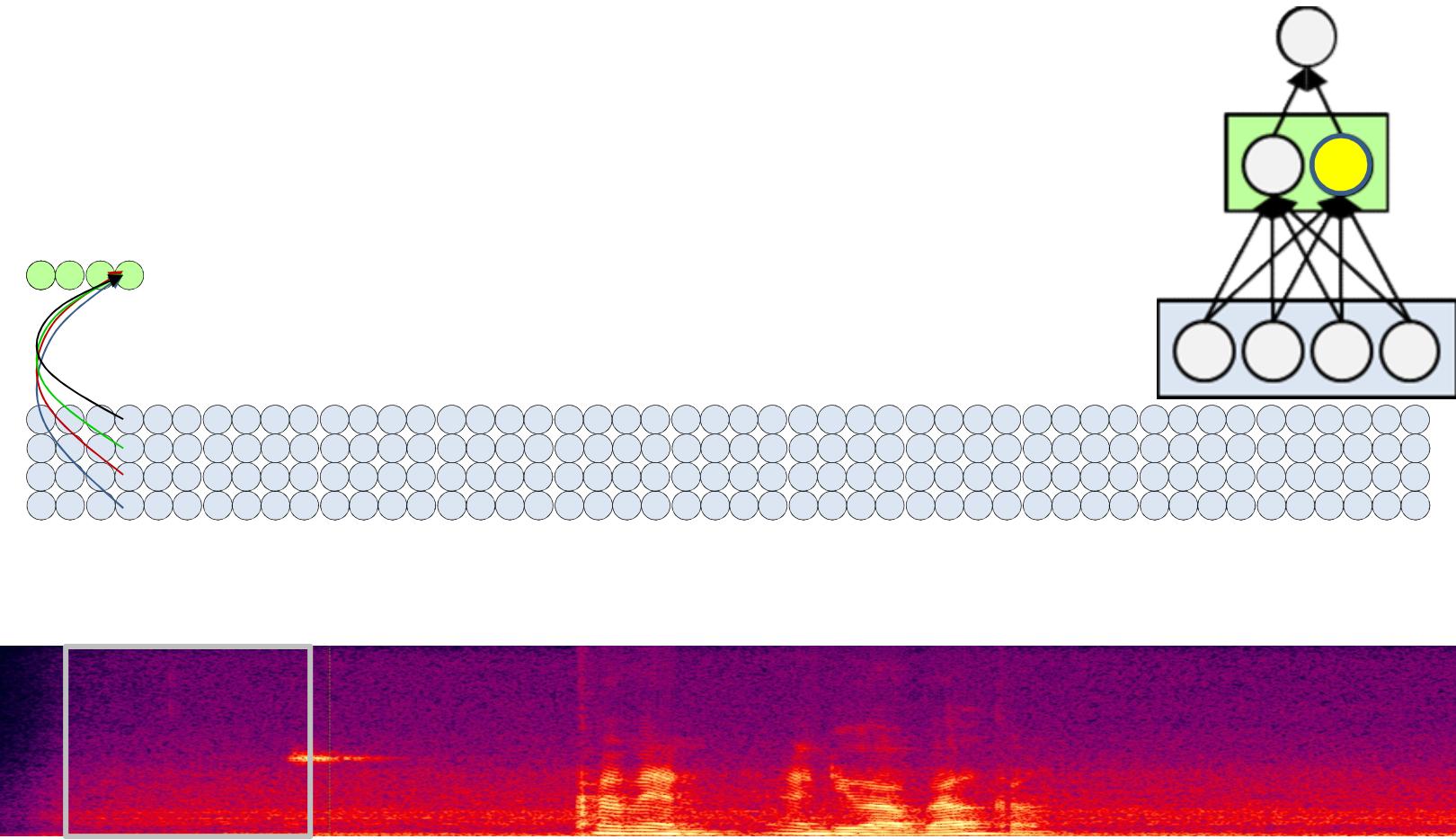
- But now, since the first layer neurons have already produced outputs for every location, each neuron in the second layer can go ahead and produce outputs for every position without waiting for the rest of the net
  - “Scan” the outputs of the first layer neurons

# Lets do it in an different order



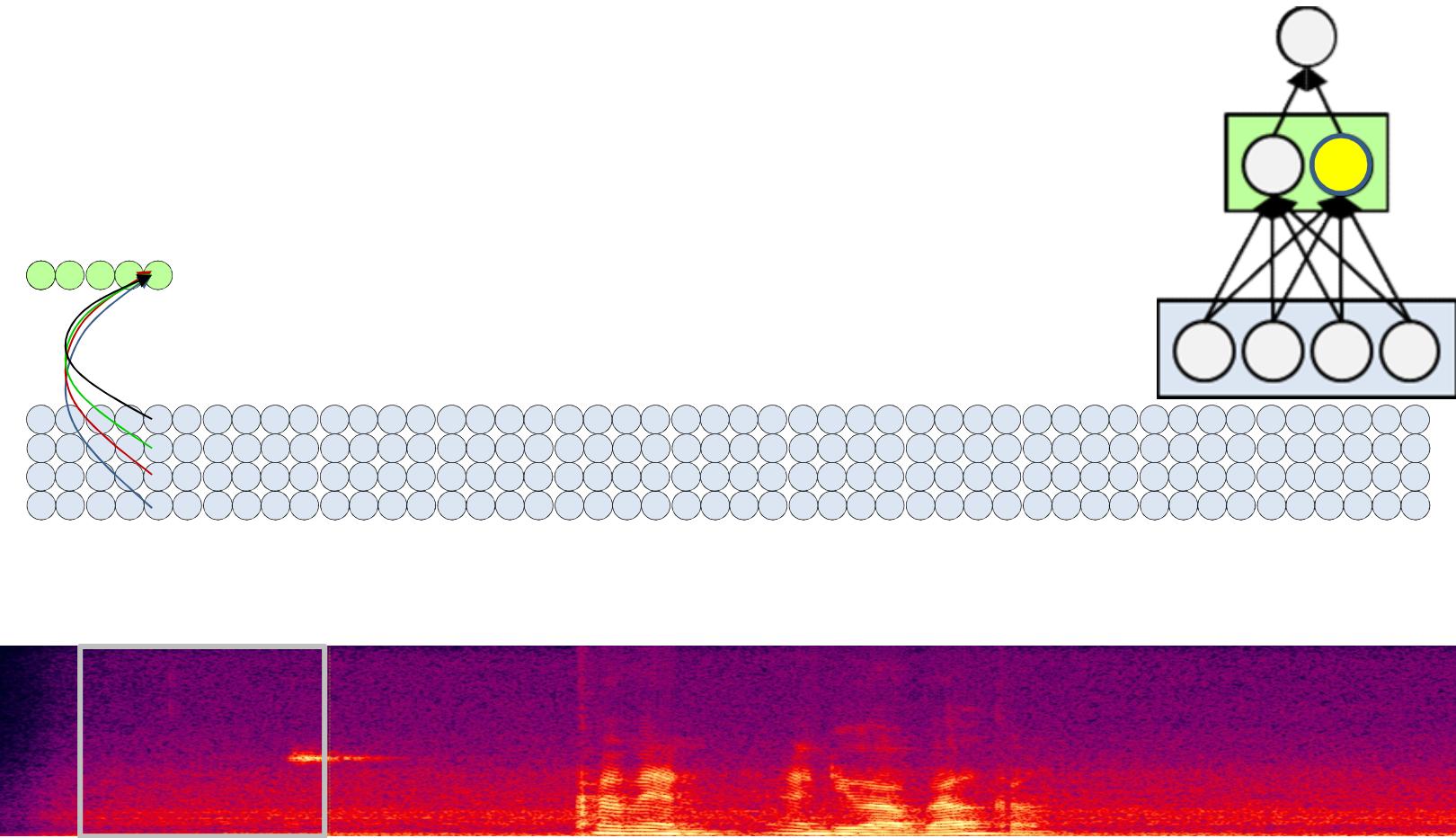
- But now, since the first layer neurons have already produced outputs for every location, each neuron in the second layer can go ahead and produce outputs for every position without waiting for the rest of the net
  - “Scan” the outputs of the first layer neurons

# Lets do it in an different order



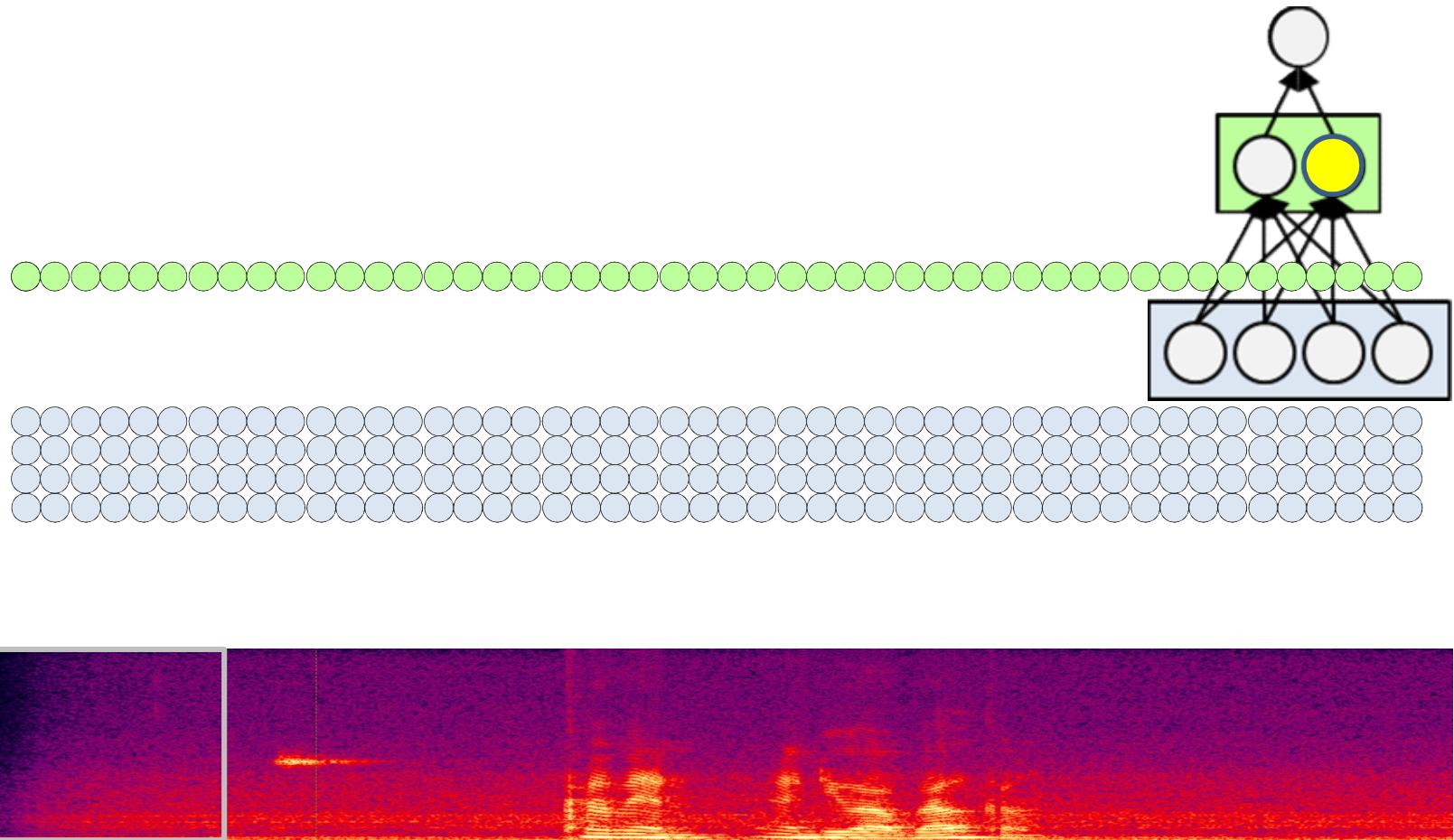
- But now, since the first layer neurons have already produced outputs for every location, each neuron in the second layer can go ahead and produce outputs for every position without waiting for the rest of the net
  - “Scan” the outputs of the first layer neurons

# Lets do it in an different order



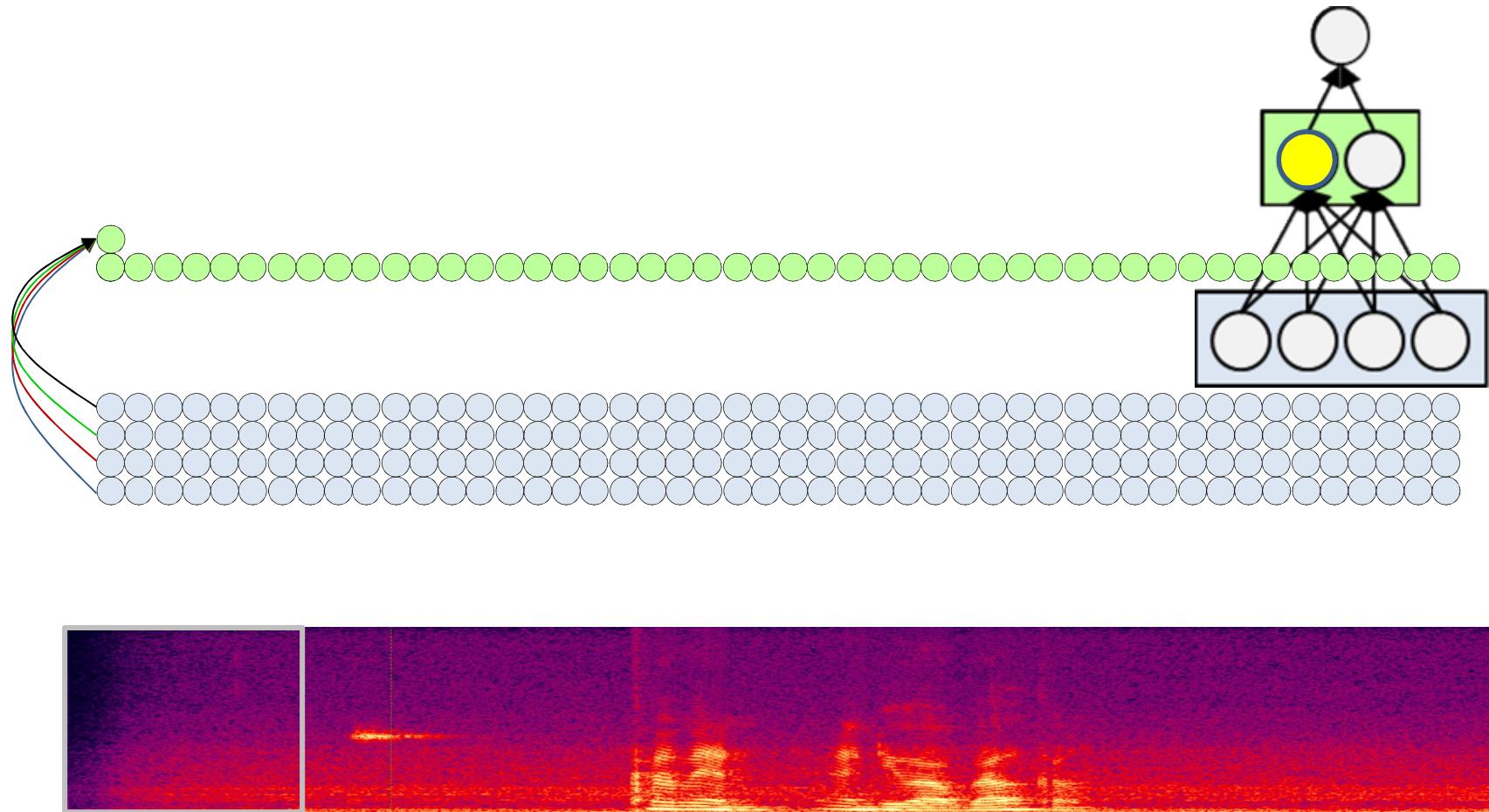
- But now, since the first layer neurons have already produced outputs for every location, each neuron in the second layer can go ahead and produce outputs for every position without waiting for the rest of the net
  - “Scan” the outputs of the first layer neurons

# Lets do it in an different order



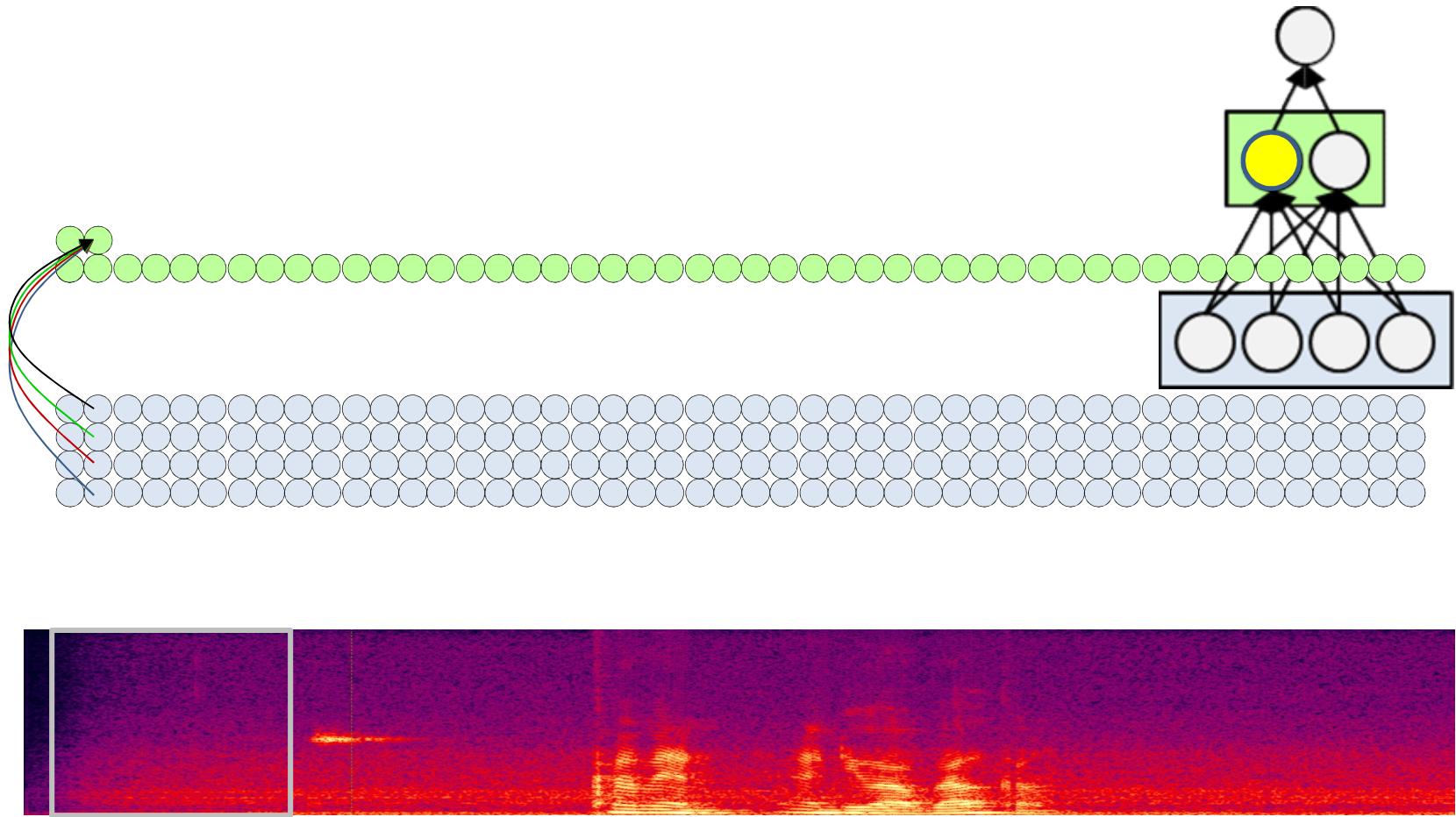
- But now, since the first layer neurons have already produced outputs for every location, each neuron in the second layer can go ahead and produce outputs for every position without waiting for the rest of the net
  - “Scan” the outputs of the first layer neurons

# Lets do it in an different order



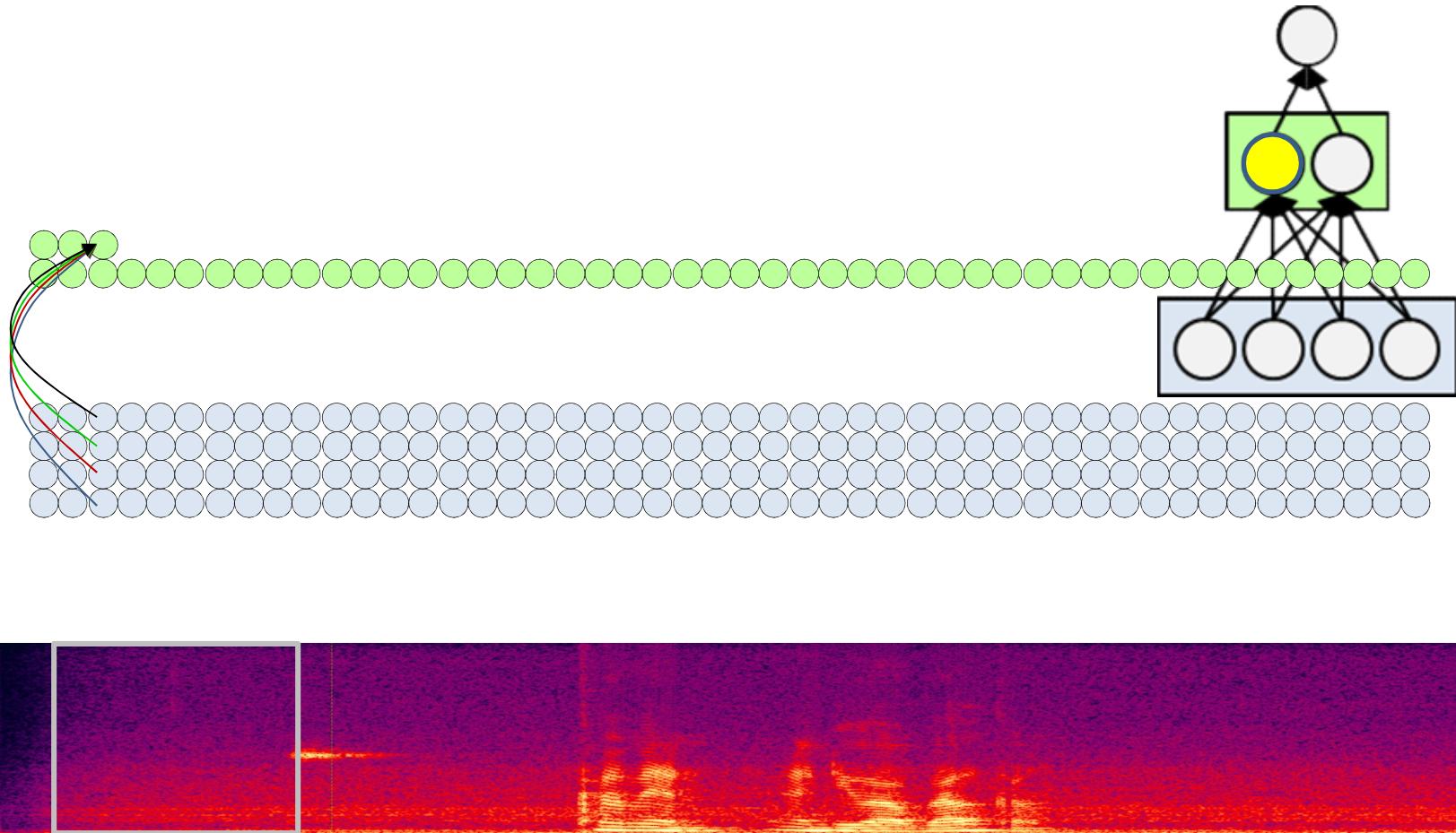
- But now, since the first layer neurons have already produced outputs for every location, each neuron in the second layer can go ahead and produce outputs for every position without waiting for the rest of the net
  - “Scan” the outputs of the first layer neurons

# Lets do it in an different order



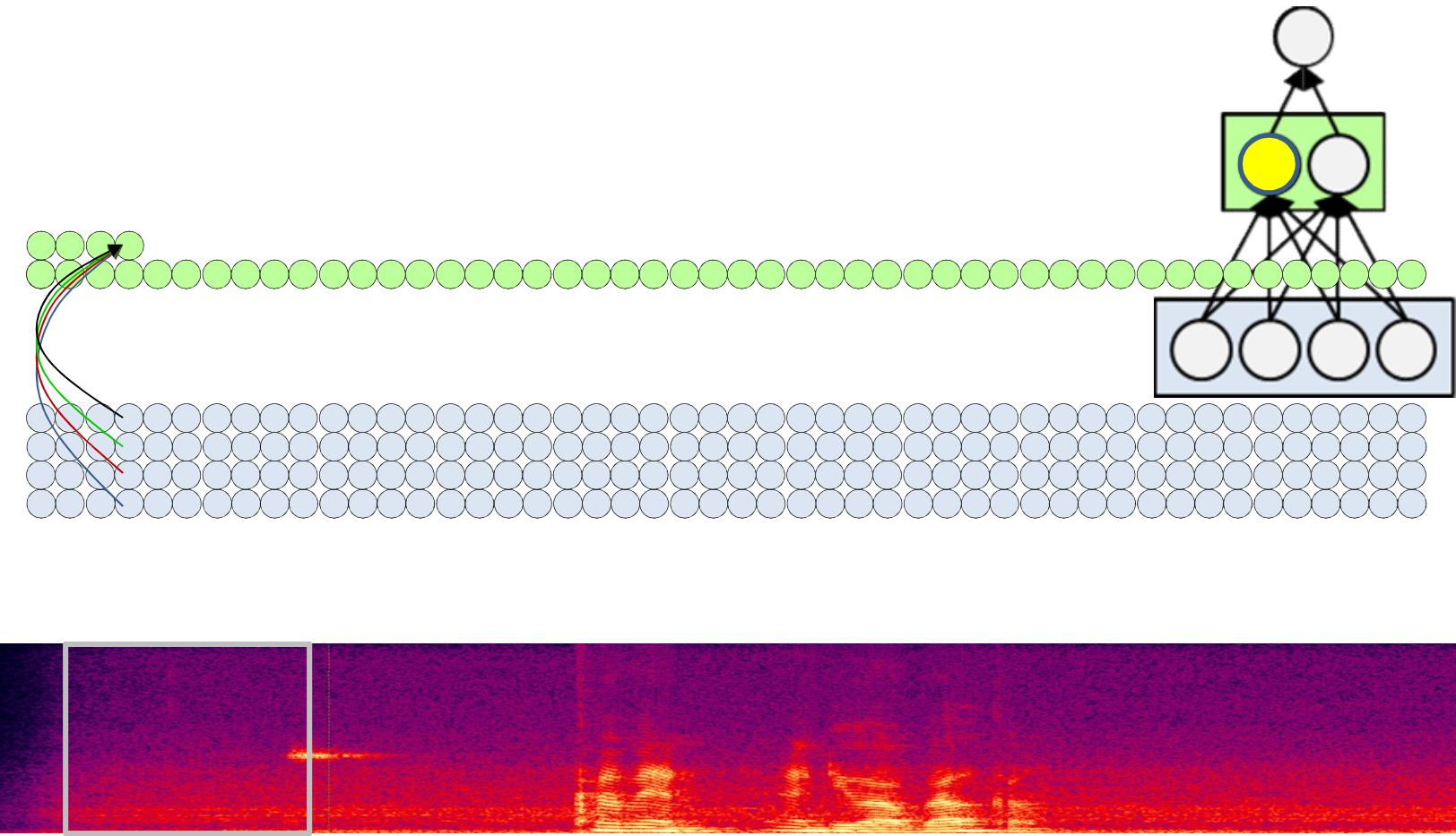
- But now, since the first layer neurons have already produced outputs for every location, each neuron in the second layer can go ahead and produce outputs for every position without waiting for the rest of the net
  - “Scan” the outputs of the first layer neurons

# Lets do it in an different order



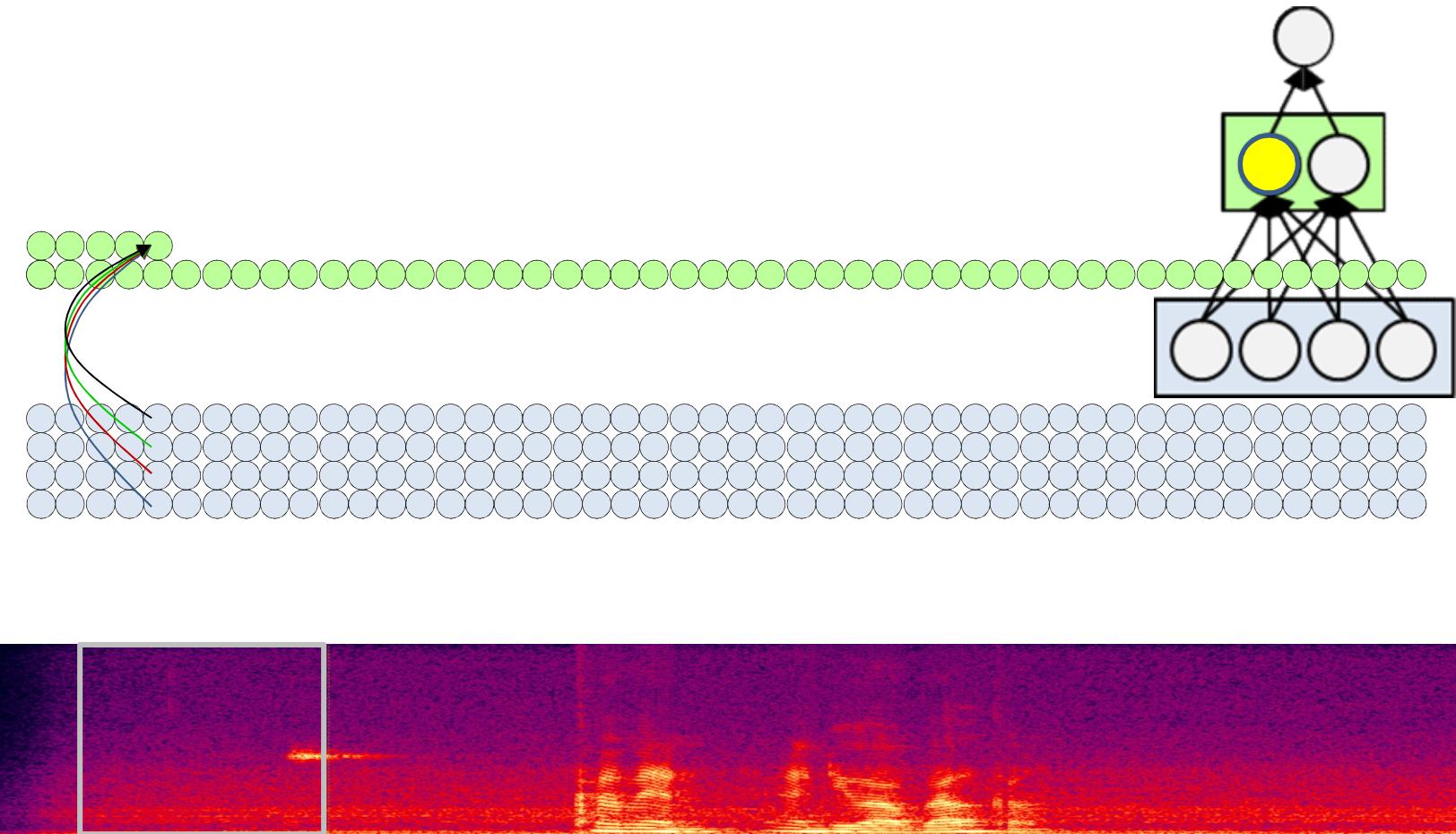
- But now, since the first layer neurons have already produced outputs for every location, each neuron in the second layer can go ahead and produce outputs for every position without waiting for the rest of the net
  - “Scan” the outputs of the first layer neurons

# Lets do it in an different order



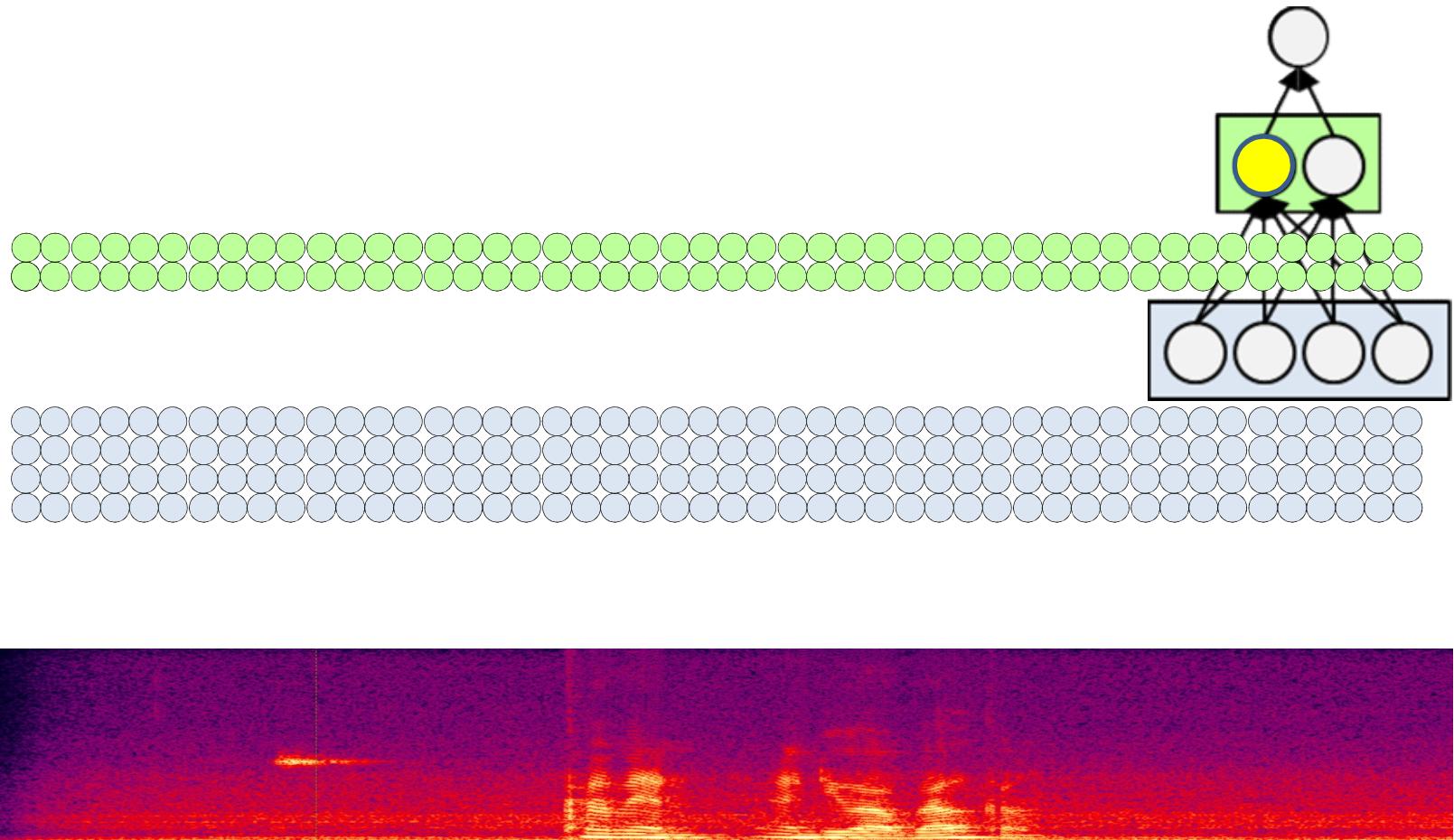
- But now, since the first layer neurons have already produced outputs for every location, each neuron in the second layer can go ahead and produce outputs for every position without waiting for the rest of the net
  - “Scan” the outputs of the first layer neurons

# Lets do it in an different order



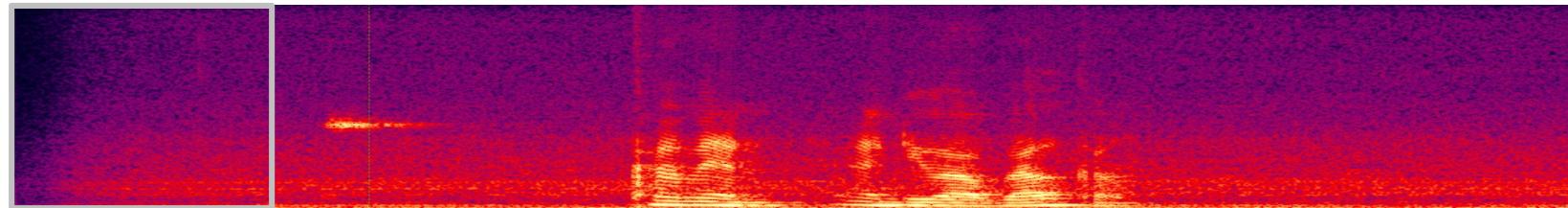
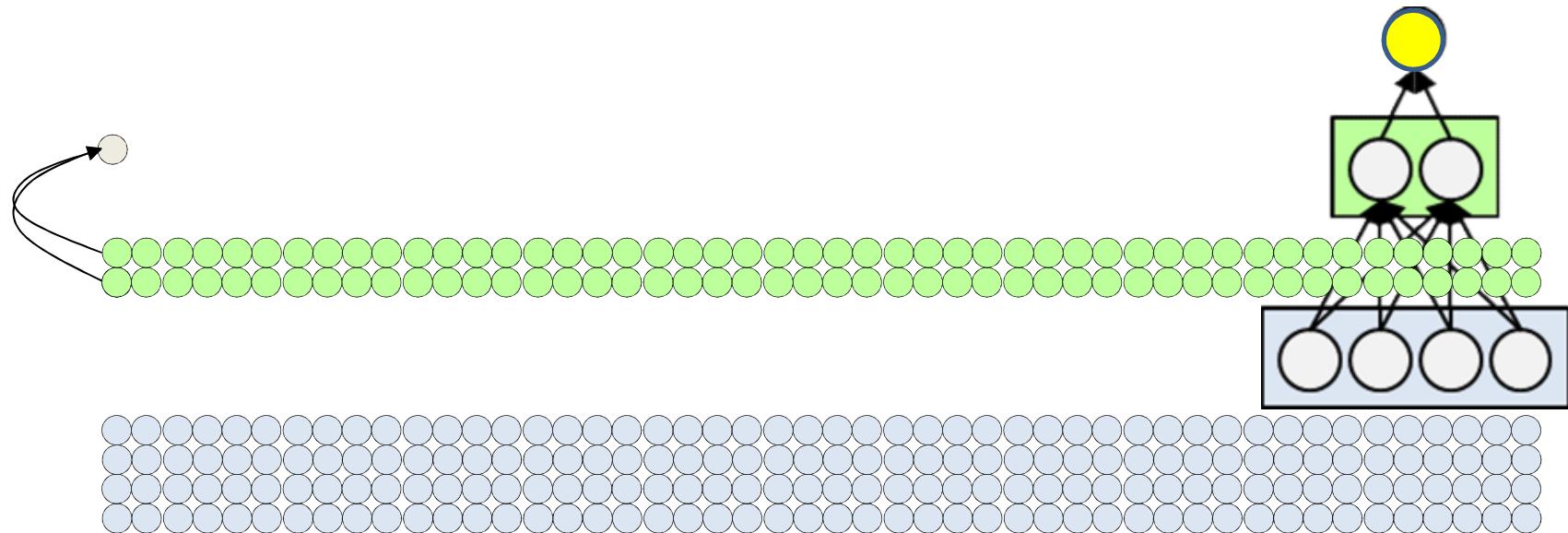
- But now, since the first layer neurons have already produced outputs for every location, each neuron in the second layer can go ahead and produce outputs for every position without waiting for the rest of the net
  - “Scan” the outputs of the first layer neurons

# Lets do it in an different order



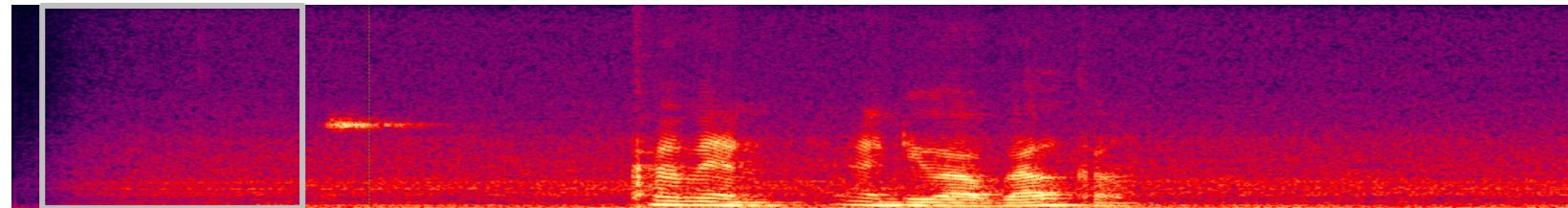
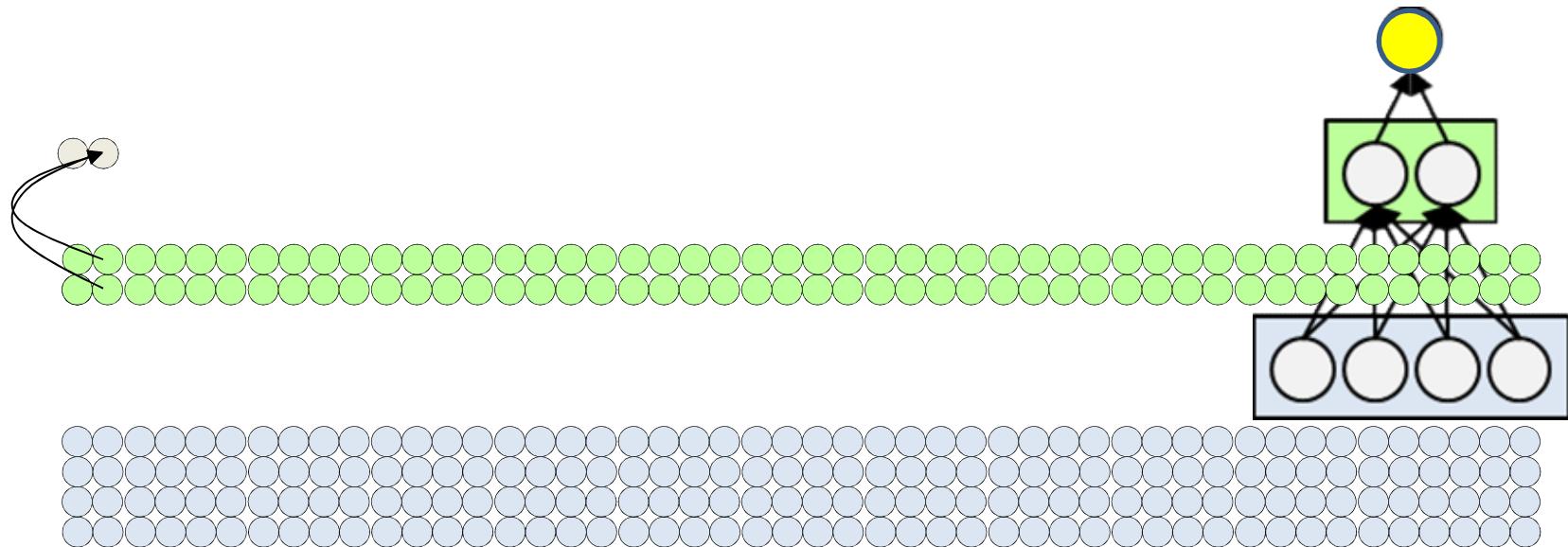
- But now, since the first layer neurons have already produced outputs for every location, each neuron in the second layer can go ahead and produce outputs for every position without waiting for the rest of the net
  - “Scan” the outputs of the first layer neurons

# Lets do it in an different order



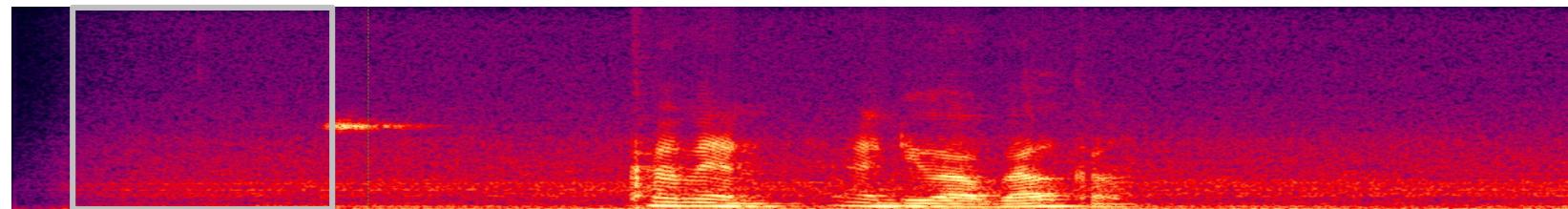
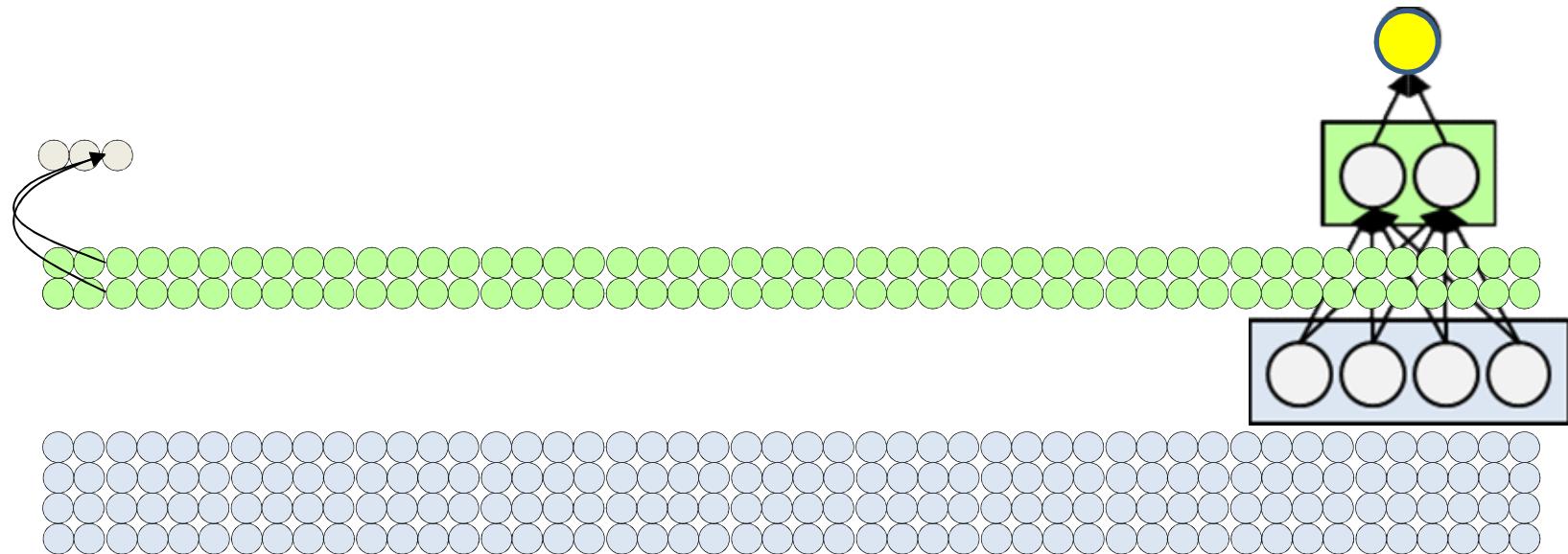
- At each position the output layer neurons can now operate on the outputs of the penultimate layer and produce the correct classification for the corresponding block!

# Lets do it in an different order



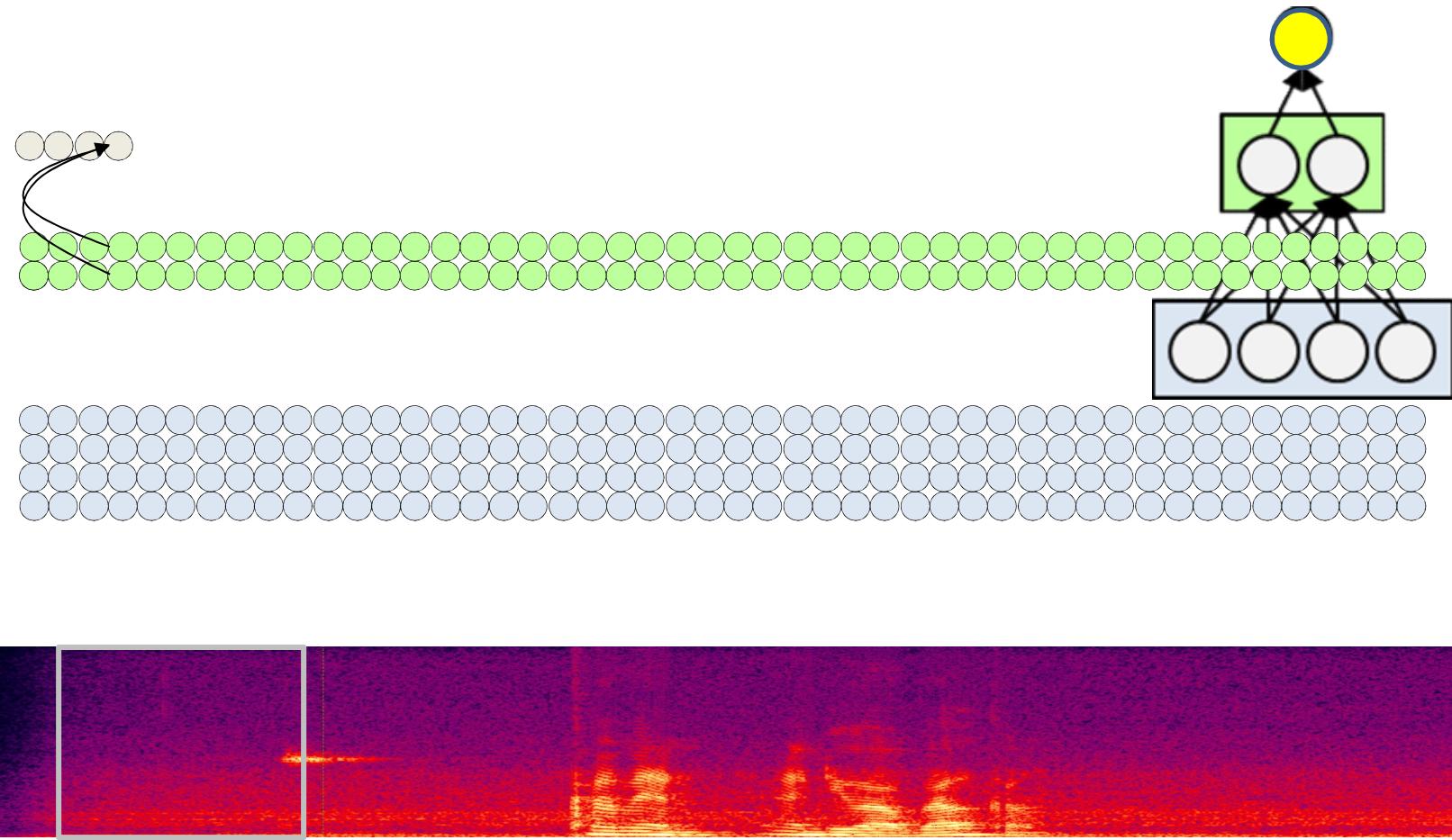
- At each position the output layer neurons can now operate on the outputs of the penultimate layer and produce the correct classification for the corresponding block!

# Lets do it in an different order



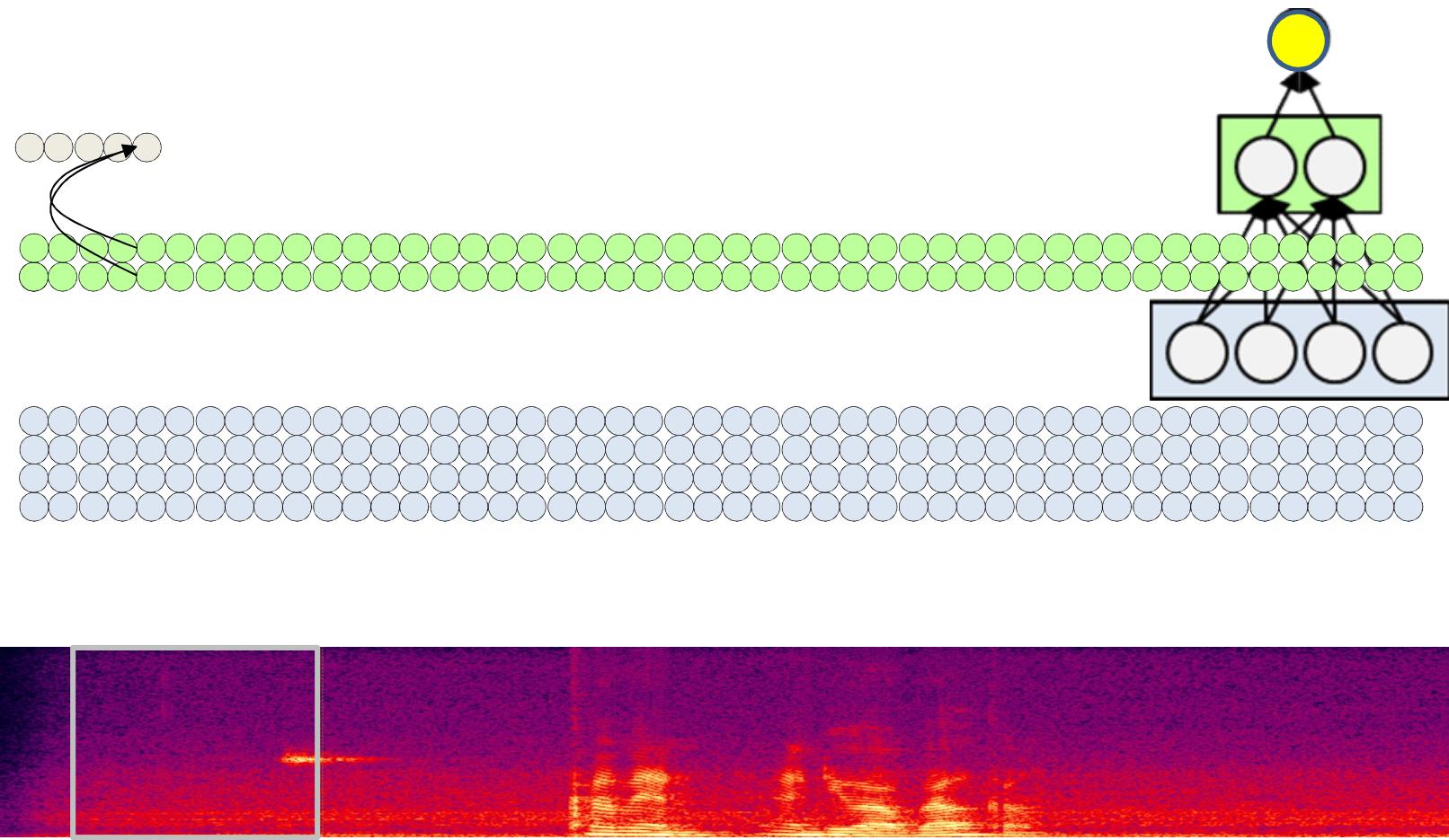
- At each position the output layer neurons can now operate on the outputs of the penultimate layer and produce the correct classification for the corresponding block!

# Lets do it in an different order



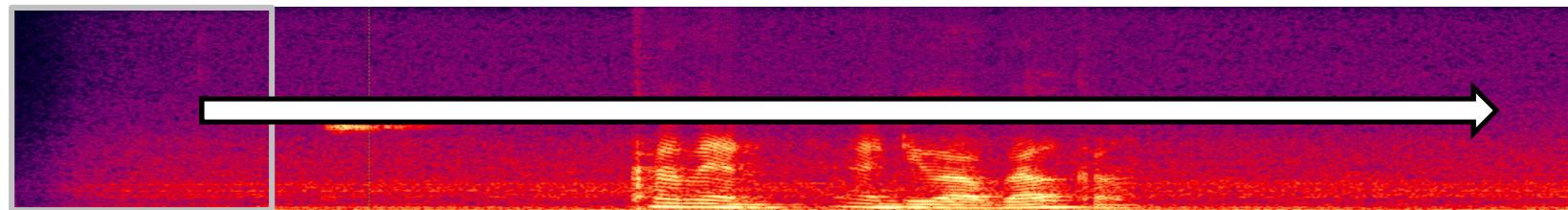
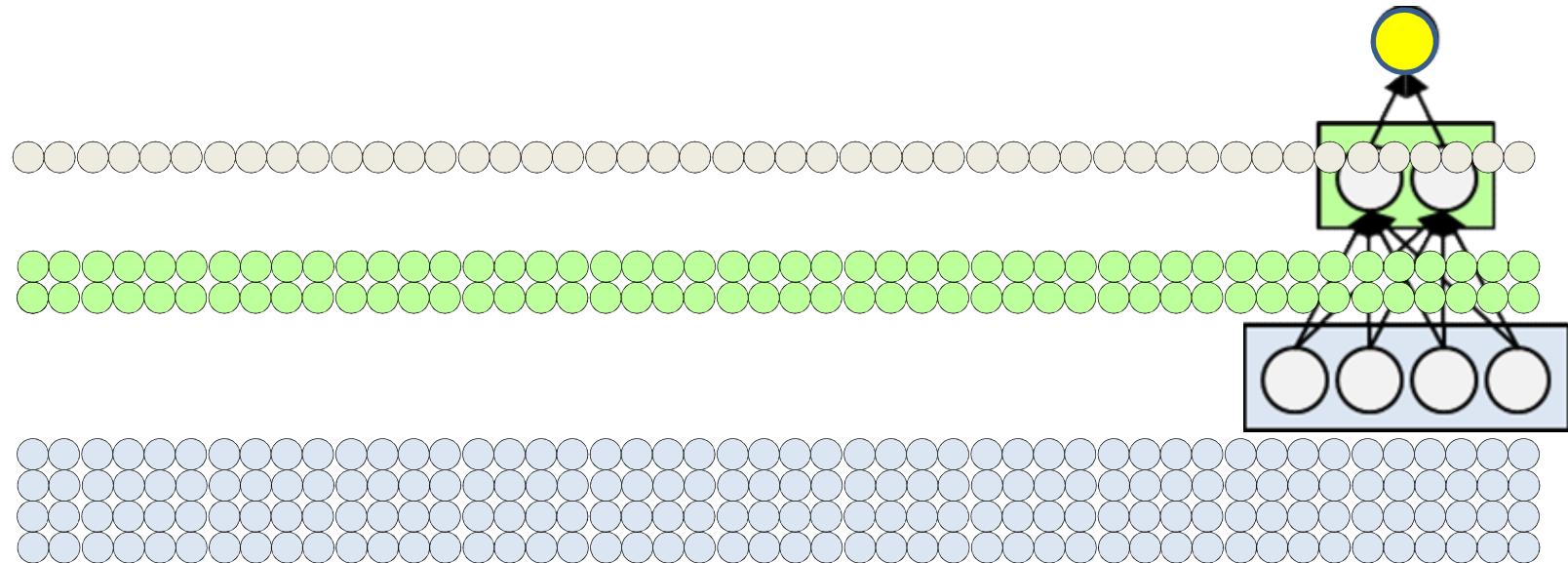
- At each position the output layer neurons can now operate on the outputs of the penultimate layer and produce the correct classification for the corresponding block!

# Lets do it in an different order



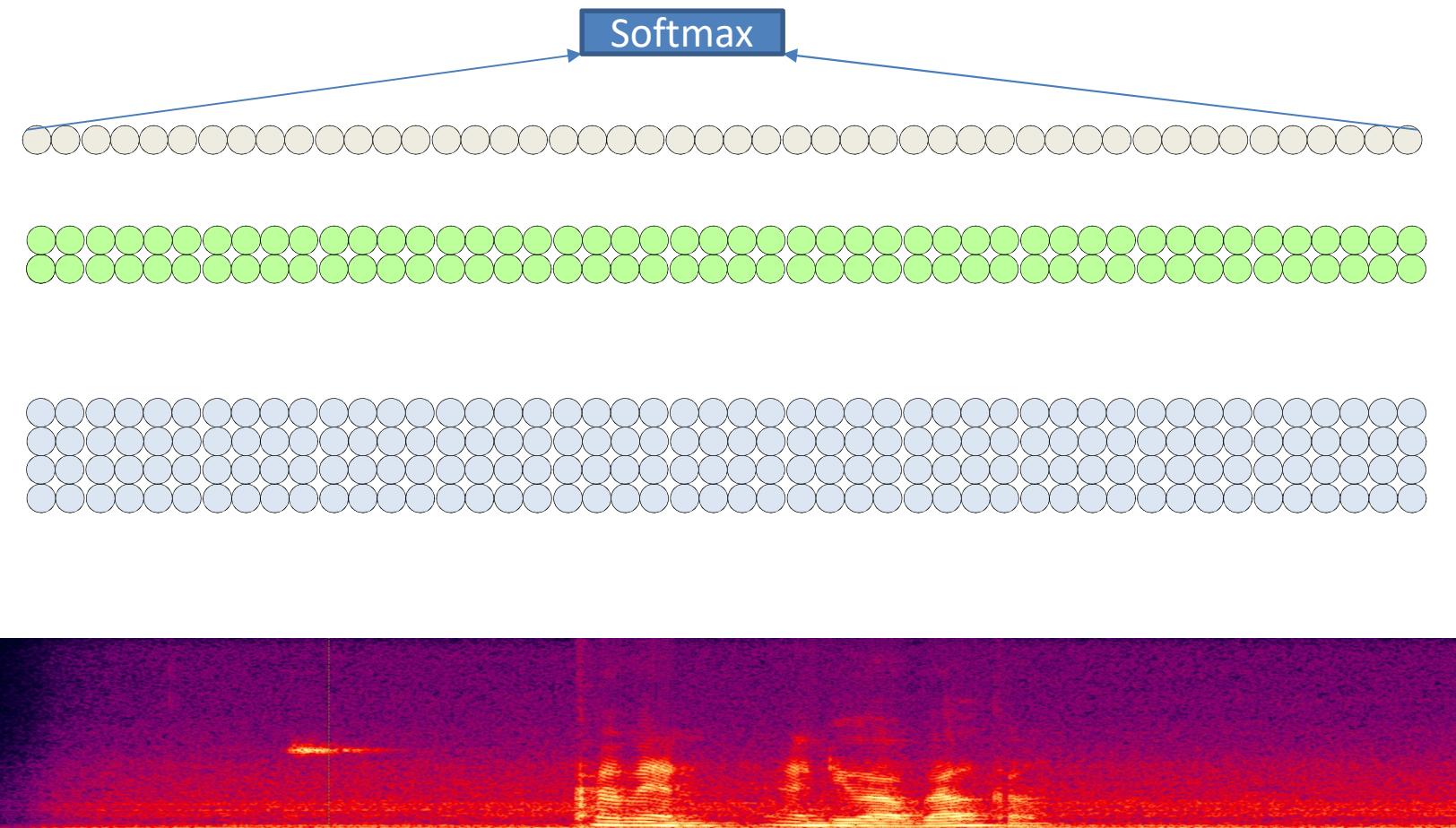
- At each position the output layer neurons can now operate on the outputs of the penultimate layer and produce the correct classification for the corresponding block!

# Lets do it in an 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!

# Lets do it in an 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 <sup>117</sup>

# Scanning with an MLP

- $K = \text{width of “patch” evaluated by MLP}$

For  $t = 1:T-K+1$

$X_{\text{Segment}} = x(:, t:t+K-1)$

$y(t) = \text{MLP}(X_{\text{Segment}})$

$Y = \text{softmax}(y(1) \dots y(T-K+1))$

# Scanning with MLP

```
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) = 0
            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))
        end
    end
Y = softmax( y(L,:,1) .. y(L,:,T-K+1) )
```

# Scanning with MLP

```
for t = 1:T-K+1           Over time
    for l = 1:L             Over layers
        # 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) = 0
            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))
        end
    end
Y = softmax( y(L,:,1) .. y(L,:,T-K+1) )
```

# Scanning with MLP

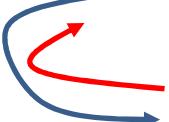
```
for t = 1:T-K+1
    for l = 1:L # layers operate at location t
        for j = 1:D_l
            if (l == 1) #first layer operates on input
                y(0,:,t) = x(:, t:t+K-1)
            end
            z(l,j,t) = 0
            for i = 1:D_{l-1}
                z(l,j,t) += w(l,i,j)y(l-1,i,t)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

# Scanning with MLP

```
for l = 1:L    # layers operate at location t
    for j = 1:Dl
        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,j,t) = 0
            for i = 1:Dl-1
                z(l,j,t) += w(l,i,j)y(l-1,i,t)
            end
            y(l,j,t) = activation(z(l,j,t))
        end
    end
Y = softmax(y(L,:,1) .. y(L,:,T-K+1))
```

Over layers

Over time



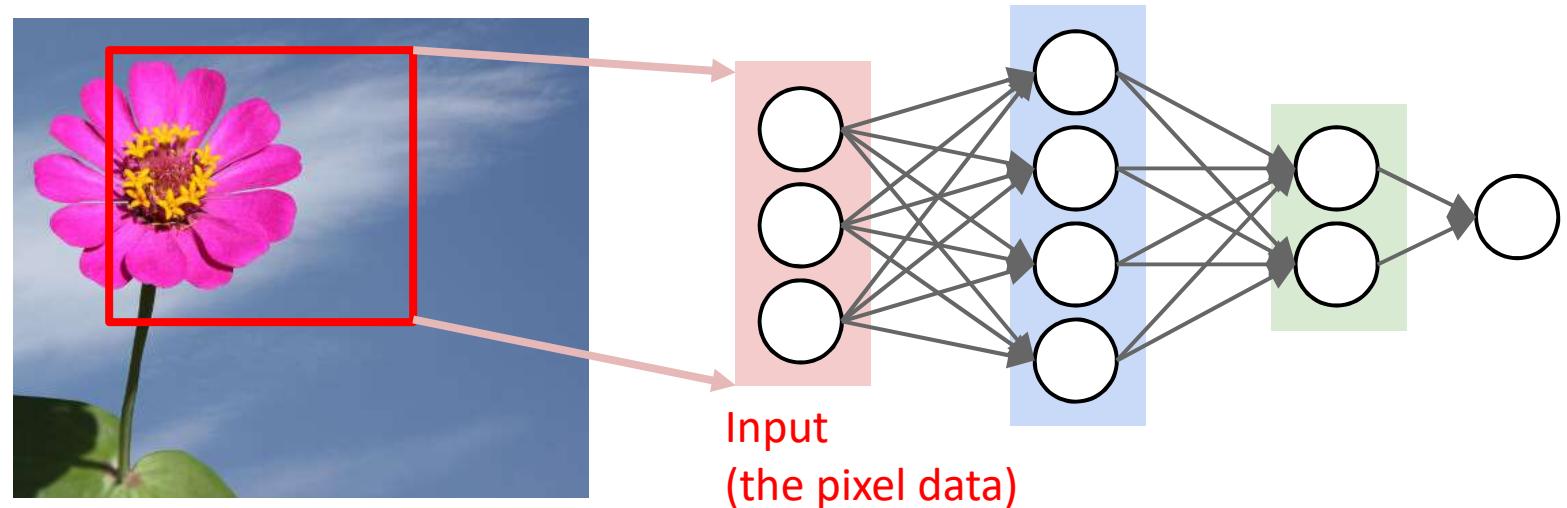
# Scanning with MLP

```
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) = 0
            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))
        end
    end
Y = softmax(y(L,:,1) .. y(L,:,T-K+1))
```

# Scanning with MLP: Vector notation

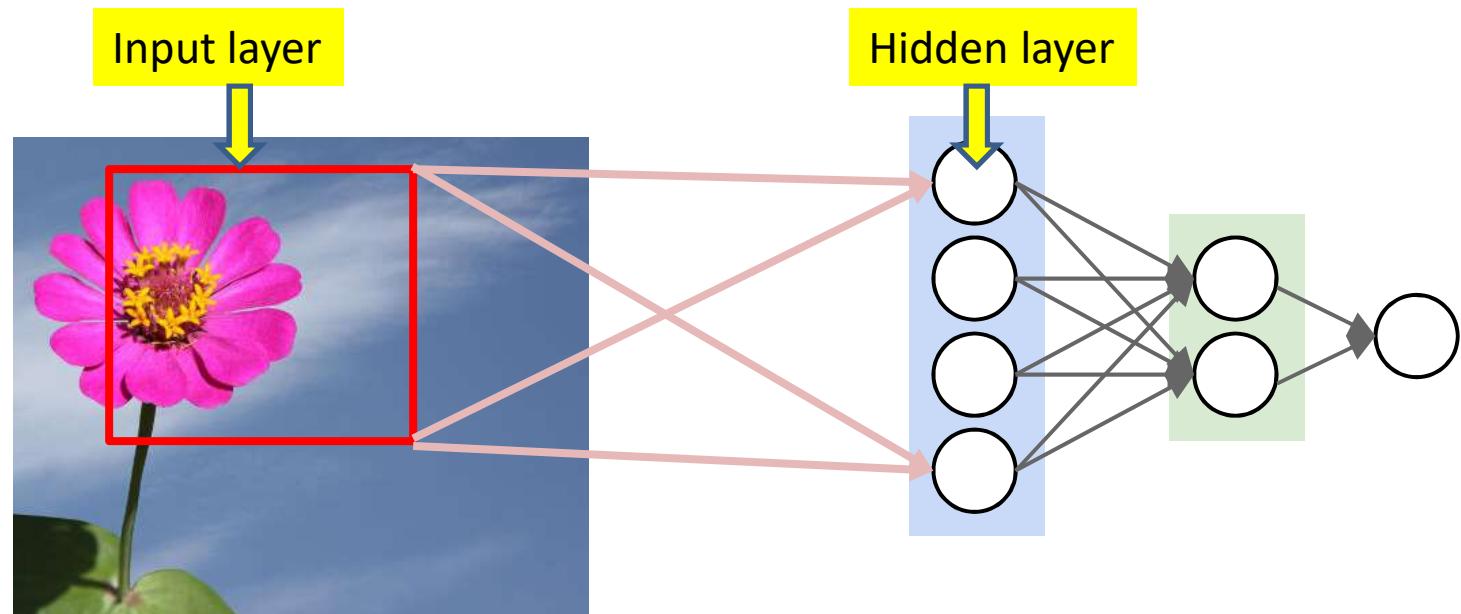
```
for l = 1:L      # layers operate at location t
    for t = 1:T-K+1
        if (l == 1) #first layer operates on input
            y(0, t) = x(:, t:t+K-1)
        end
        z(l, t) = w(l)y(l-1, t)
        y(l, t) = activation(z(l, t))
    end
Y = softmax( y(L, 1) .. y(L, T-K+1) )
```

# Scanning in 2D: A closer look



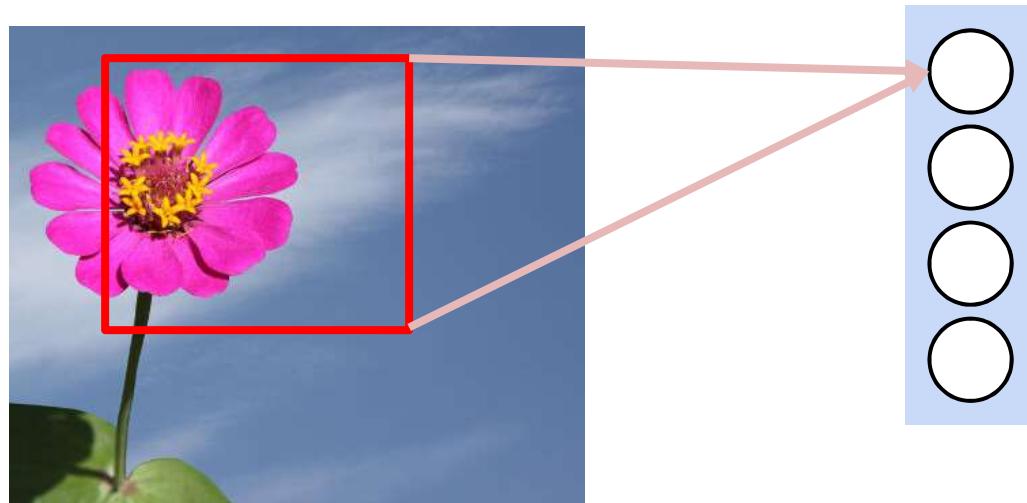
- *Scan for the desired object*
- At each location, the entire region is sent through an MLP

# Scanning: A closer look



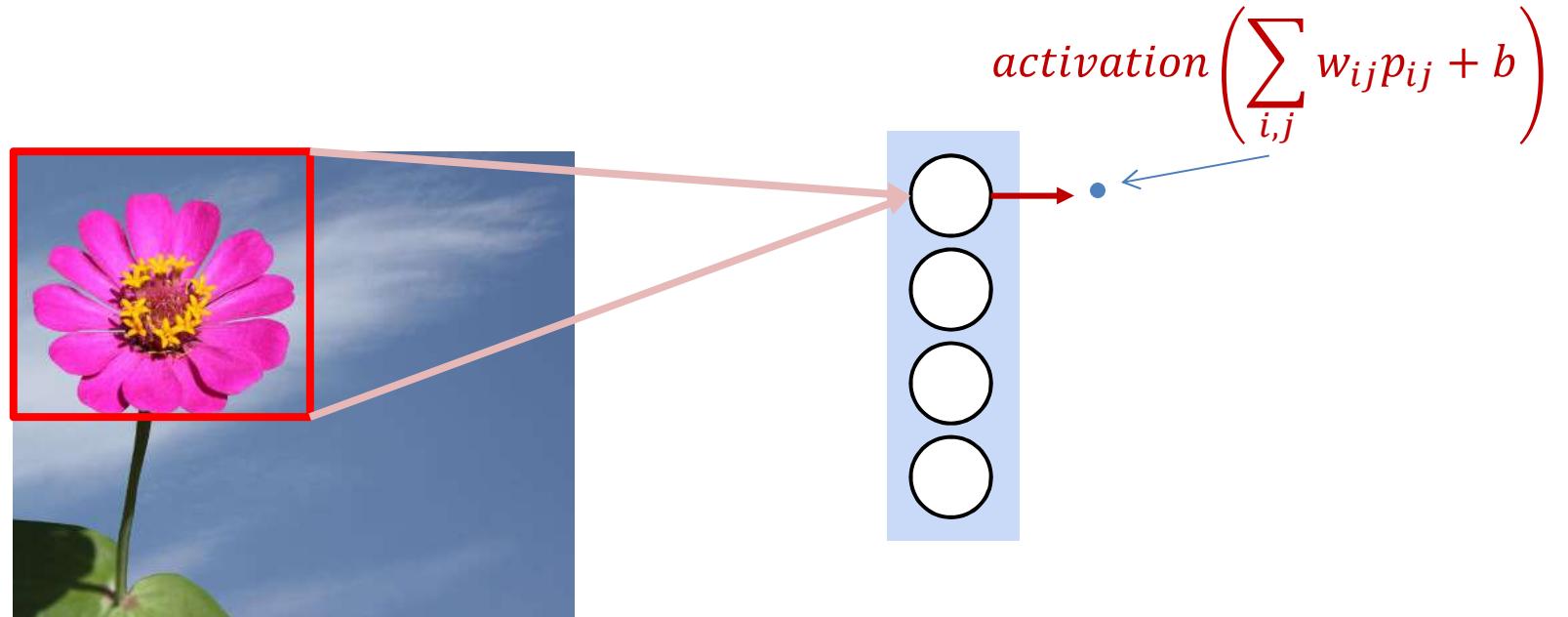
- The “input layer” is just the pixels in the image connecting to the hidden layer

# Scanning: A closer look



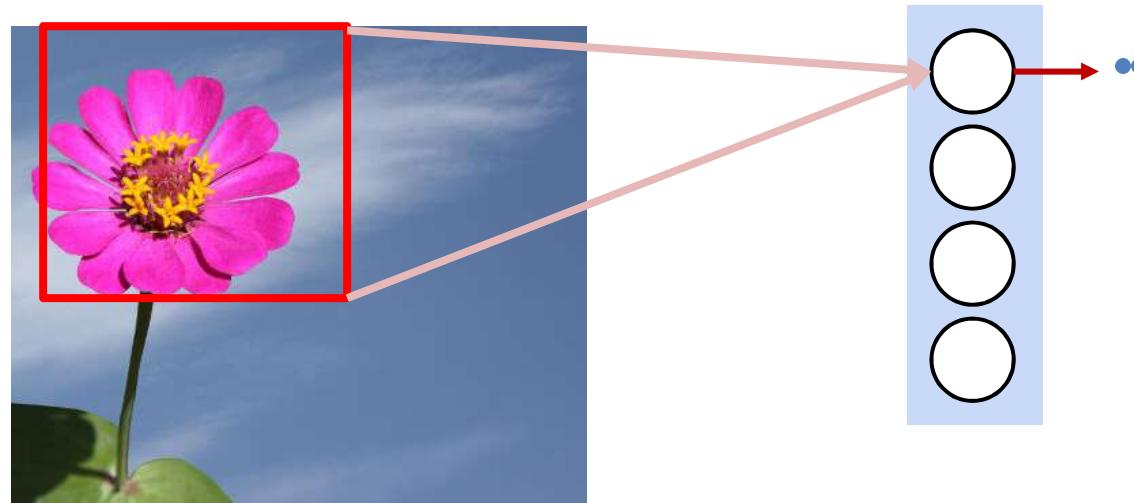
- Consider a single neuron

# Scanning: A closer look



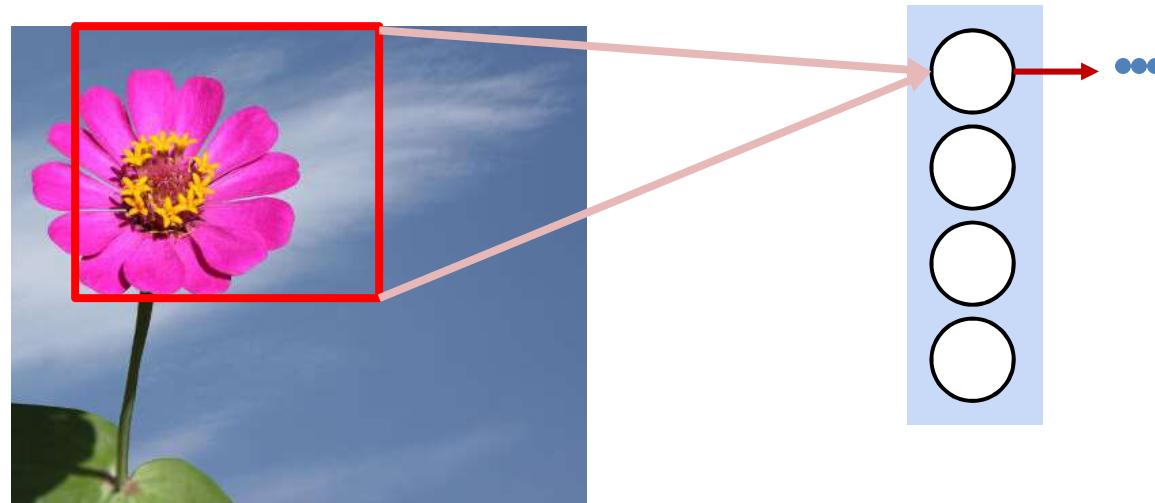
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the part of the picture in the box as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture

# Scanning: A closer look



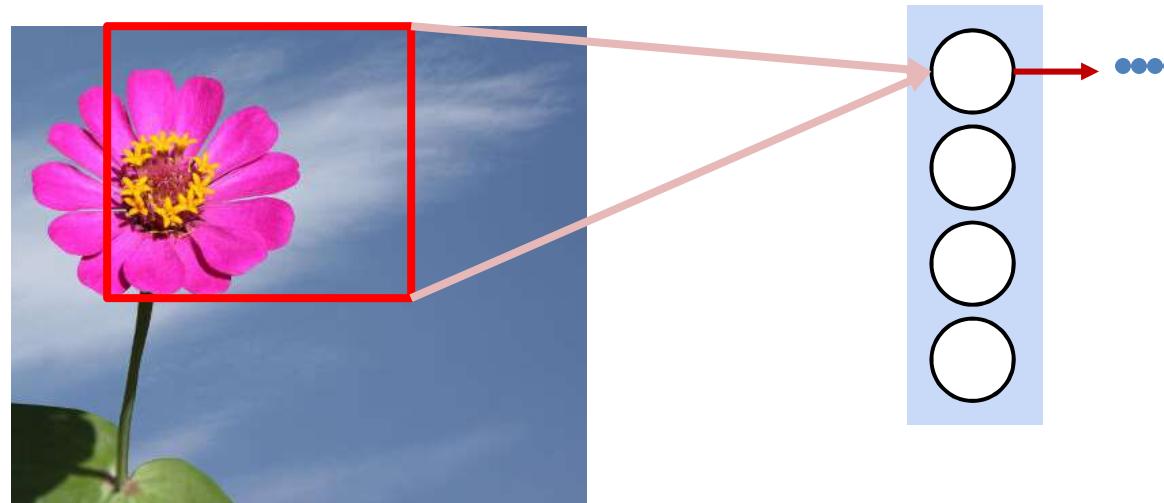
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture

# Scanning: A closer look



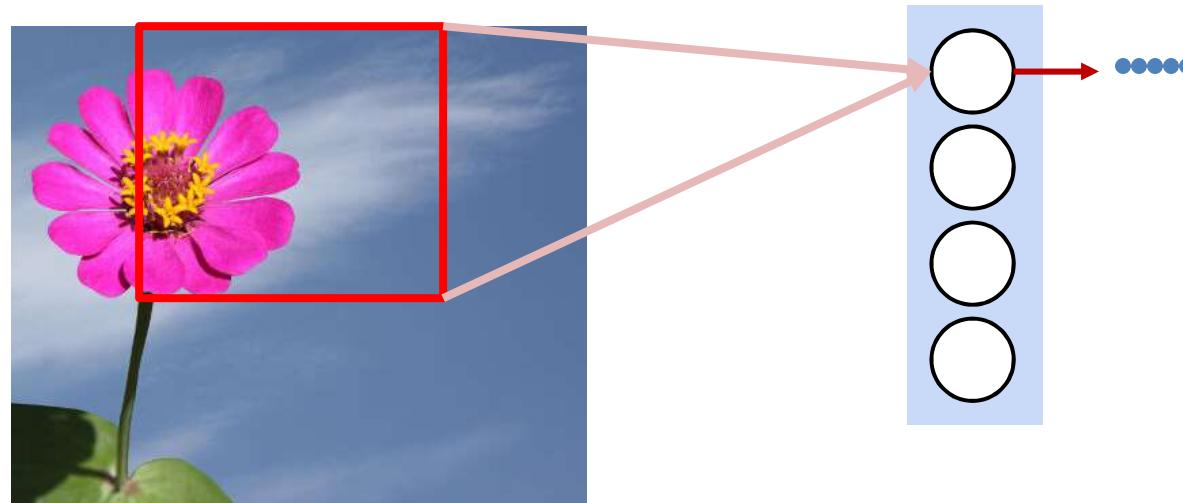
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture

# Scanning: A closer look



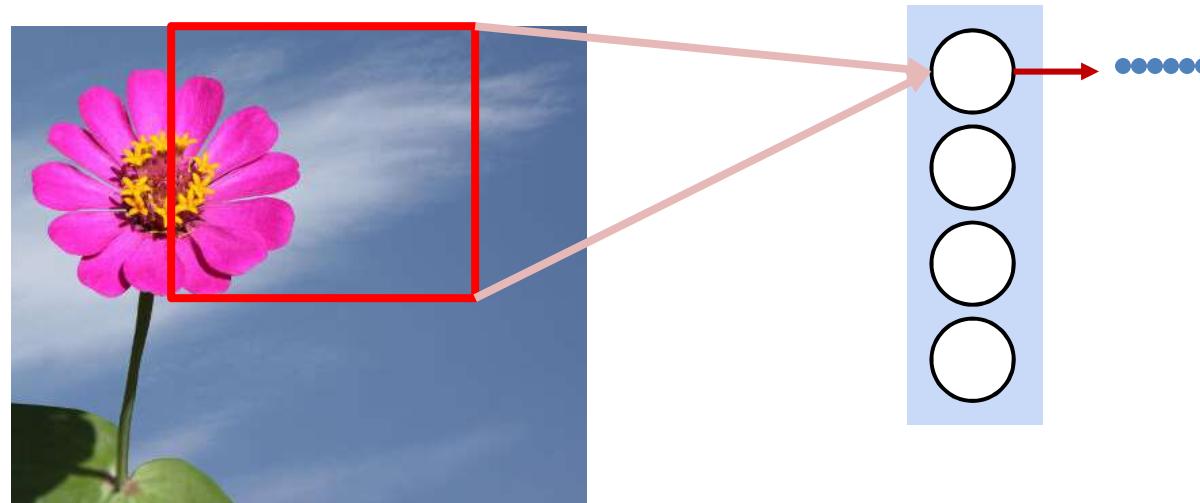
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture

# Scanning: A closer look



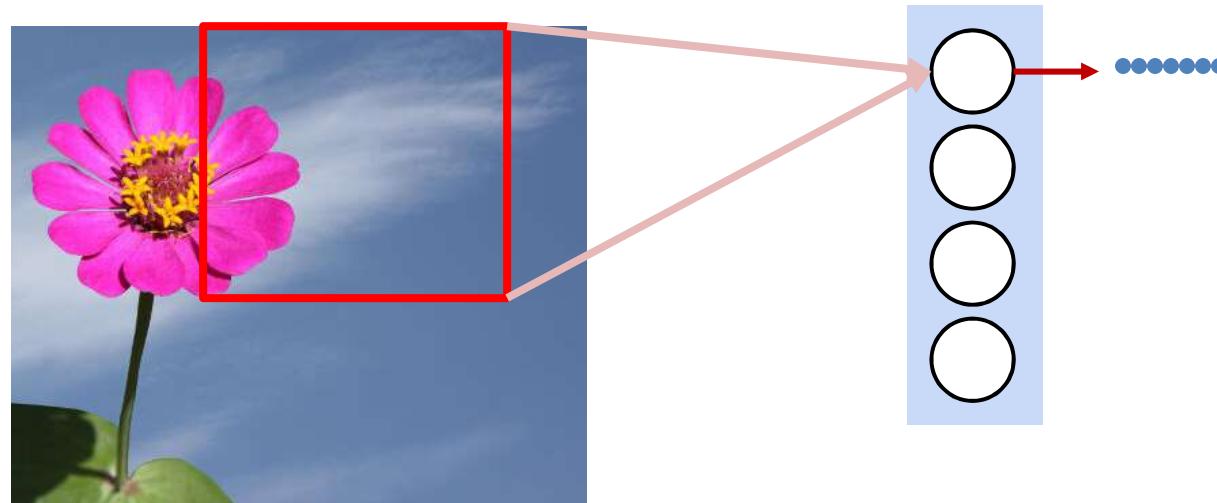
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture

# Scanning: A closer look



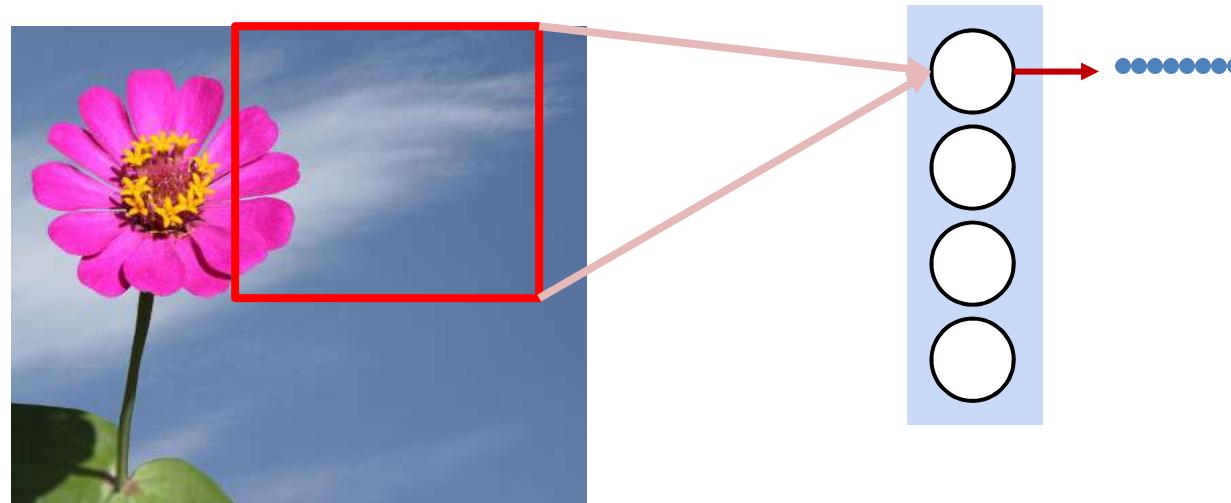
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture

# Scanning: A closer look



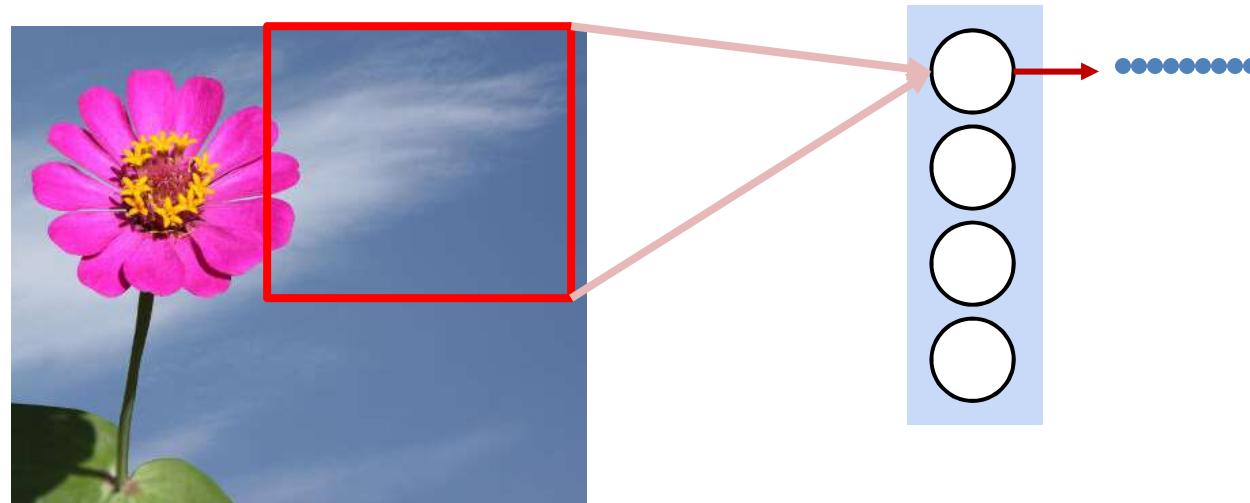
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture

# Scanning: A closer look



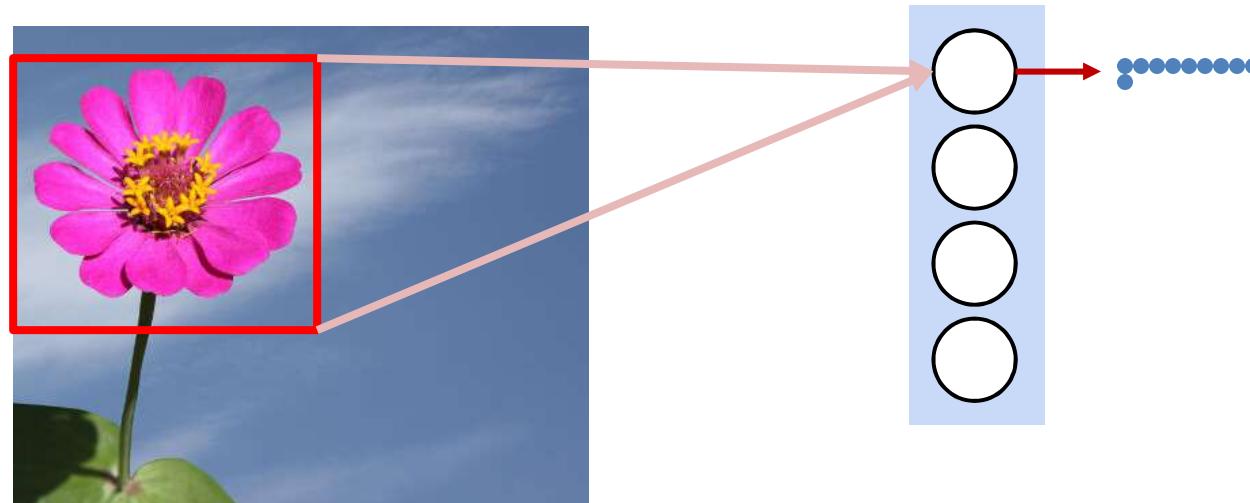
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture

# Scanning: A closer look



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

# Scanning: A closer look



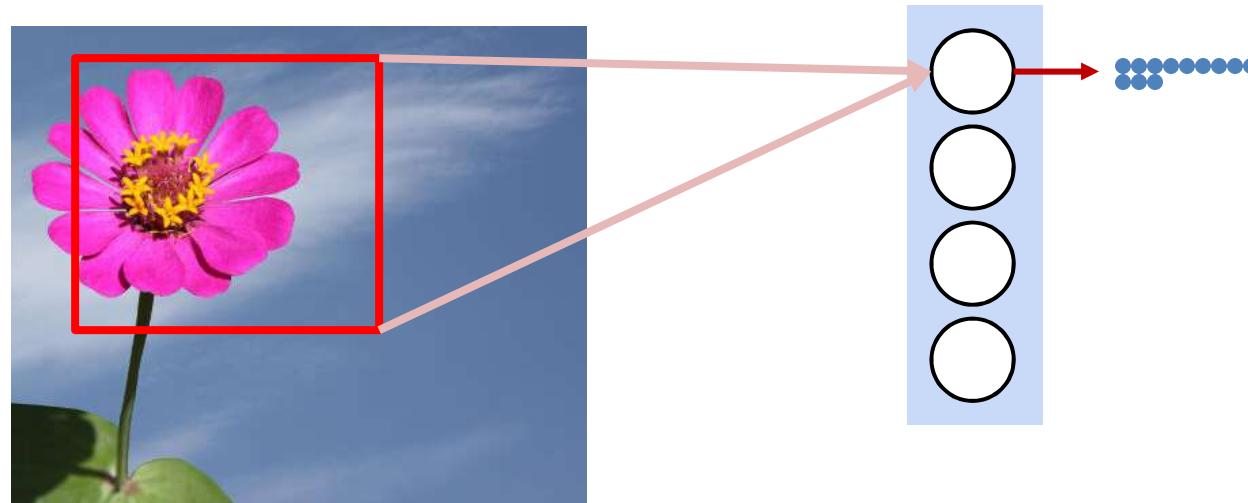
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture

# Scanning: A closer look



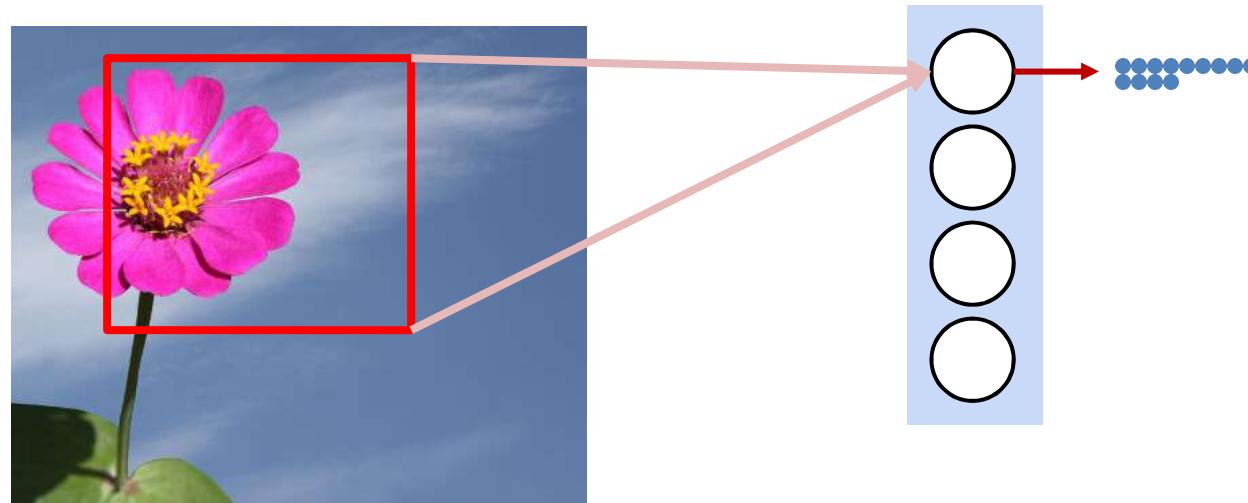
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture

# Scanning: A closer look



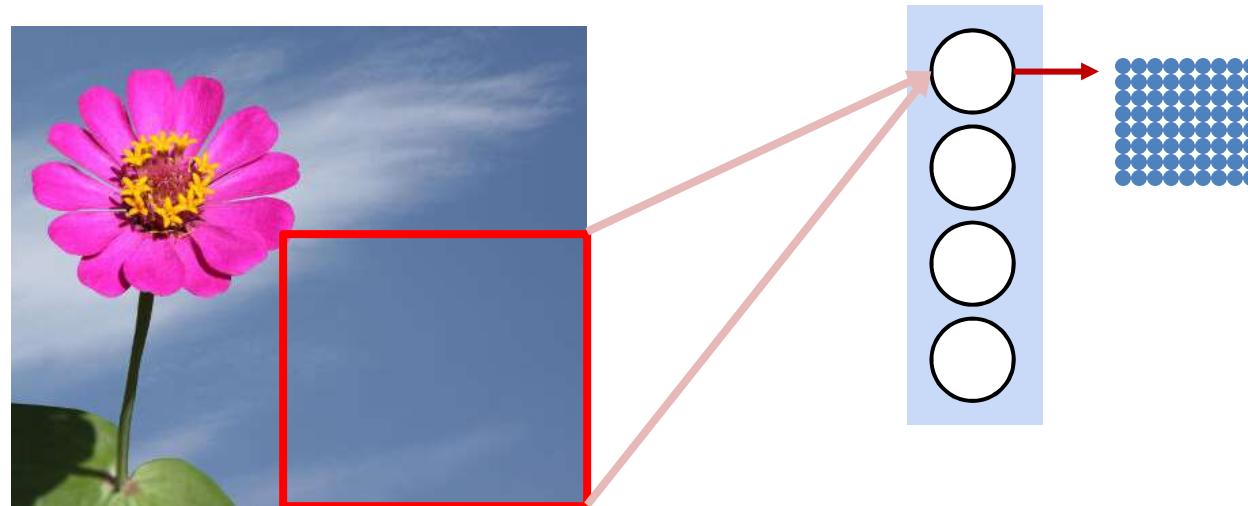
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture

# Scanning: A closer look



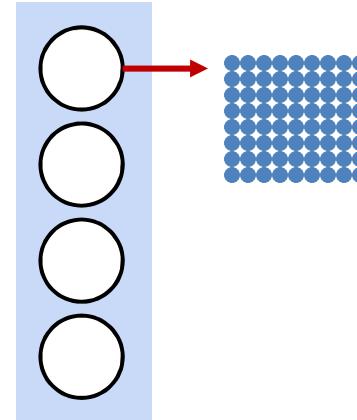
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture

# Scanning: A closer look



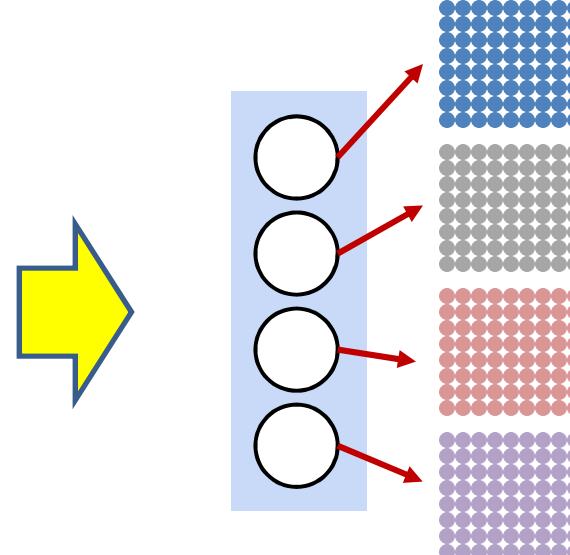
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture
- Eventually, we can arrange the outputs from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

# Scanning: A closer look



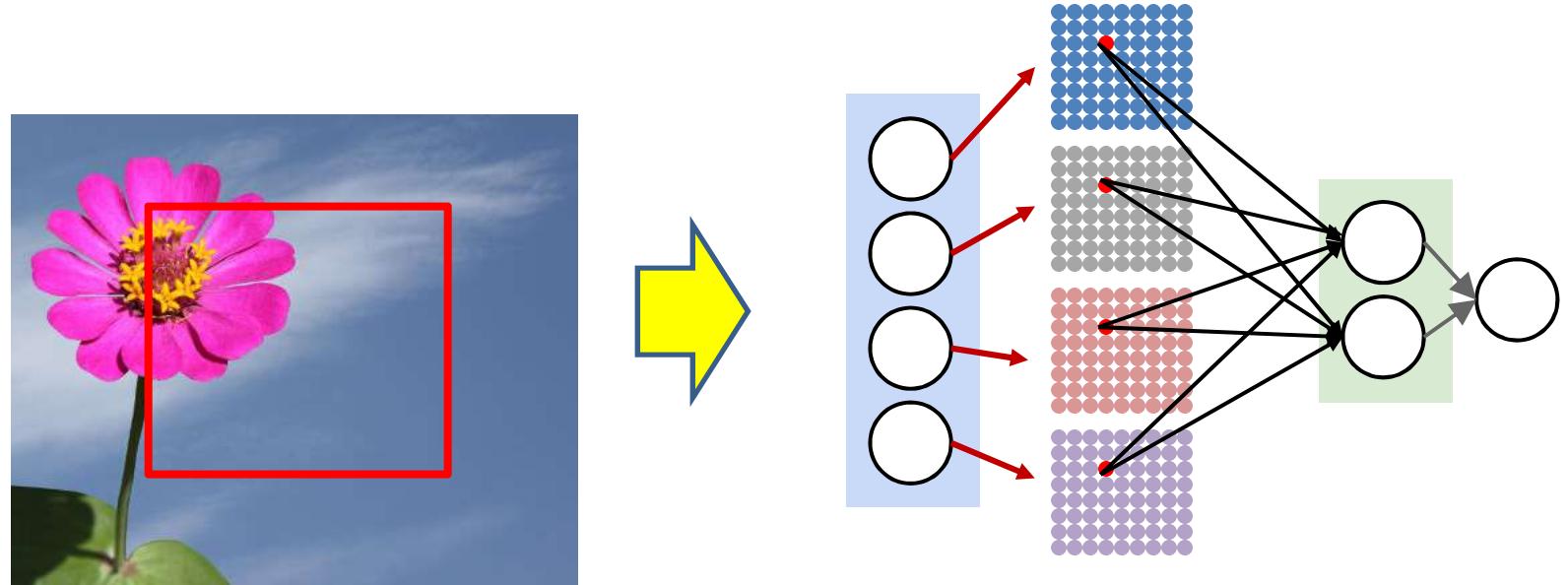
- Consider a single perceptron
- At each position of the box, the perceptron is evaluating the picture as part of the classification for *that* region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture
- Eventually, we can arrange the outputs from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

# Scanning: A closer look



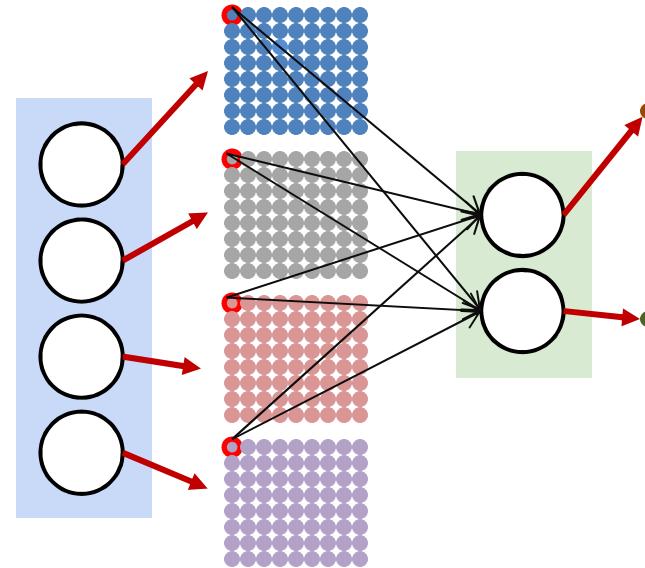
- Similarly, each first-layer perceptron's outputs from the scanned positions can be arranged as a rectangular pattern

# Scanning: A closer look



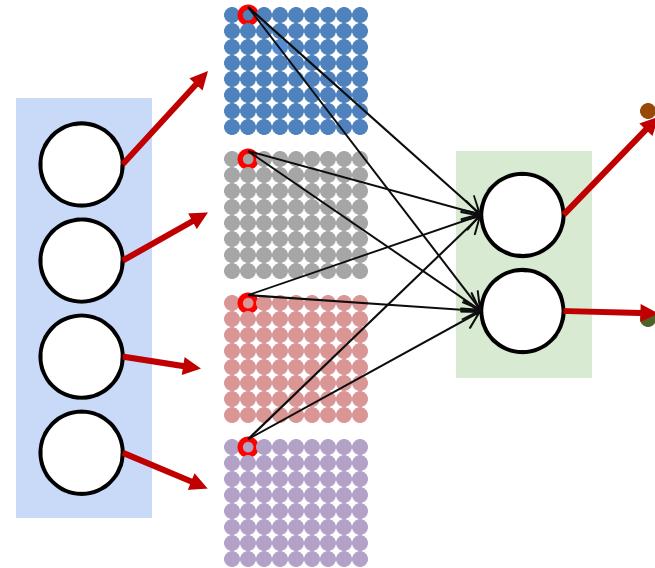
- To classify a specific “patch” in the image, we send the first level activations from the positions corresponding to that position to the next layer

# Scanning: A closer look



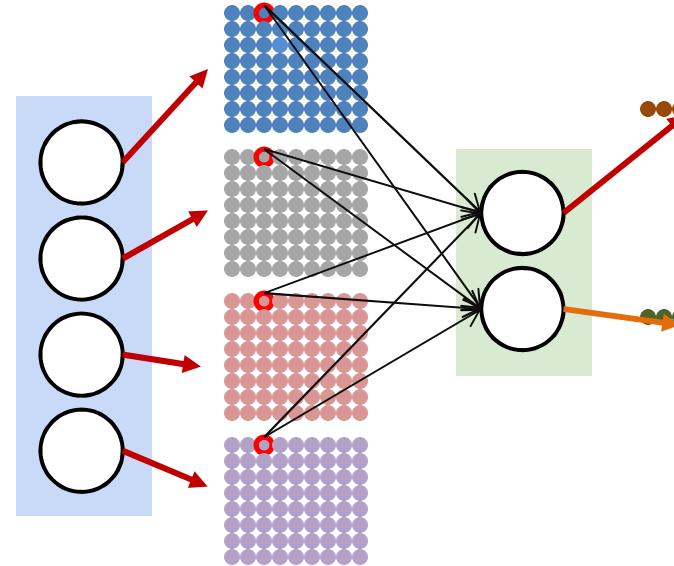
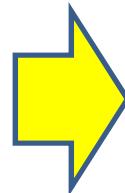
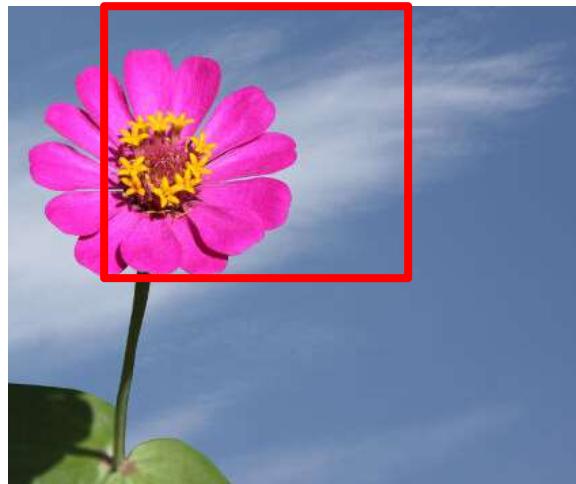
- We can recurse the logic
  - The second level neurons too are “scanning” the rectangular outputs of the first-level neurons
  - (Un)like the first level, they are jointly scanning *multiple* “pictures”
    - Each location in the output of the second level neuron considers the corresponding locations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

# Scanning: A closer look



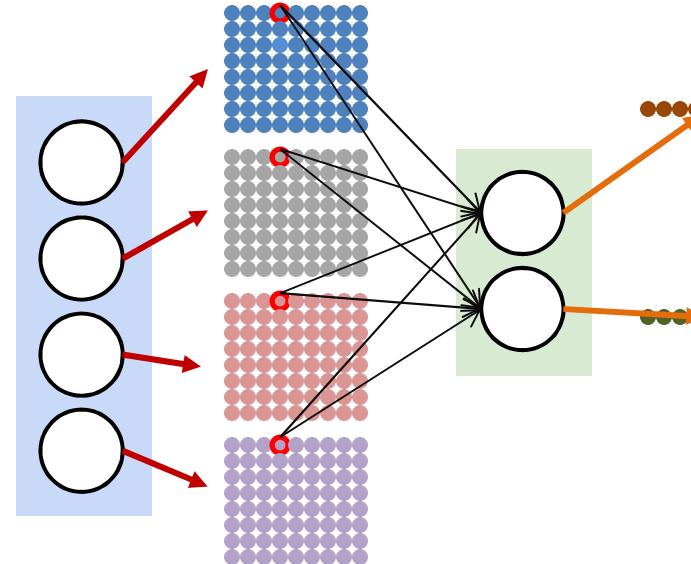
- We can recurse the logic
  - The second level neurons too are “scanning” the rectangular outputs of the first-level neurons
  - (Un)like the first level, they are jointly scanning *multiple* “pictures”
    - Each location in the output of the second level neuron considers the corresponding locations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

# Scanning: A closer look



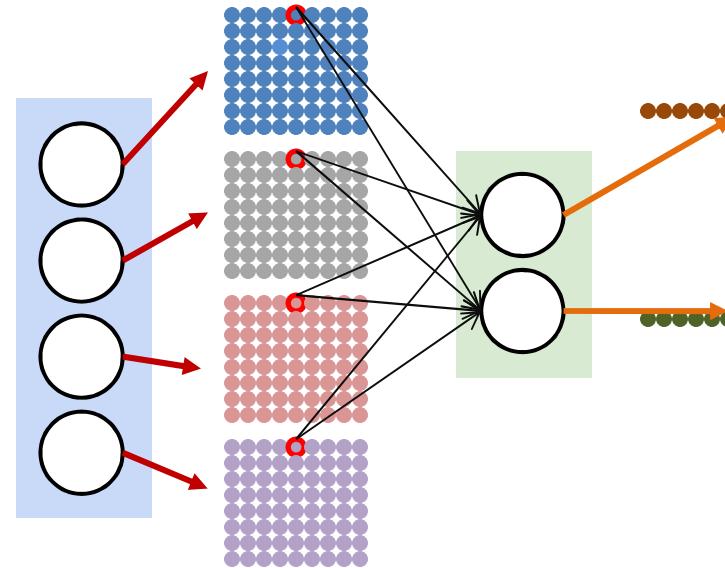
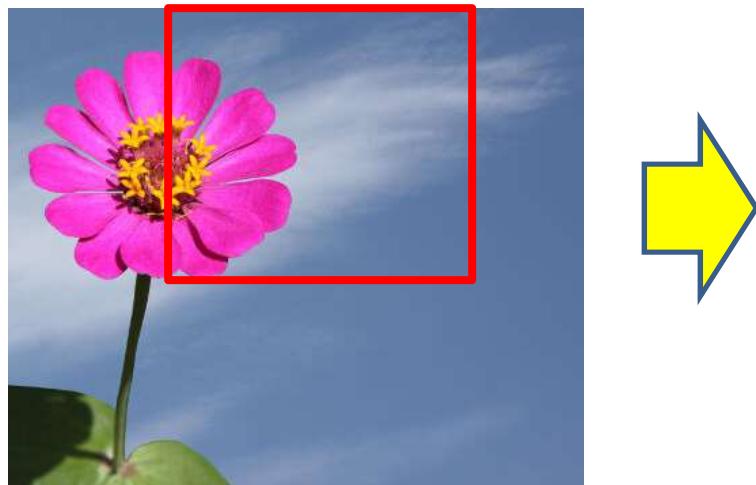
- We can recurse the logic
  - The second level neurons too are “scanning” the rectangular outputs of the first-level neurons
  - (Un)like the first level, they are jointly scanning *multiple* “pictures”
    - Each location in the output of the second level neuron considers the corresponding locations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

# Scanning: A closer look



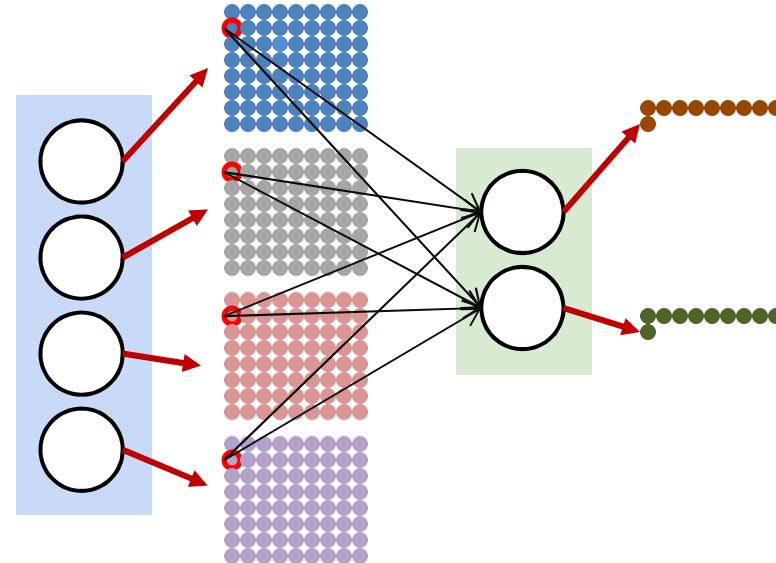
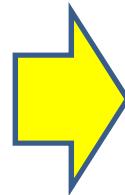
- We can recurse the logic
  - The second level neurons too are “scanning” the rectangular outputs of the first-level neurons
  - (Un)like the first level, they are jointly scanning *multiple* “pictures”
    - Each location in the output of the second level neuron considers the corresponding locations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

# Scanning: A closer look



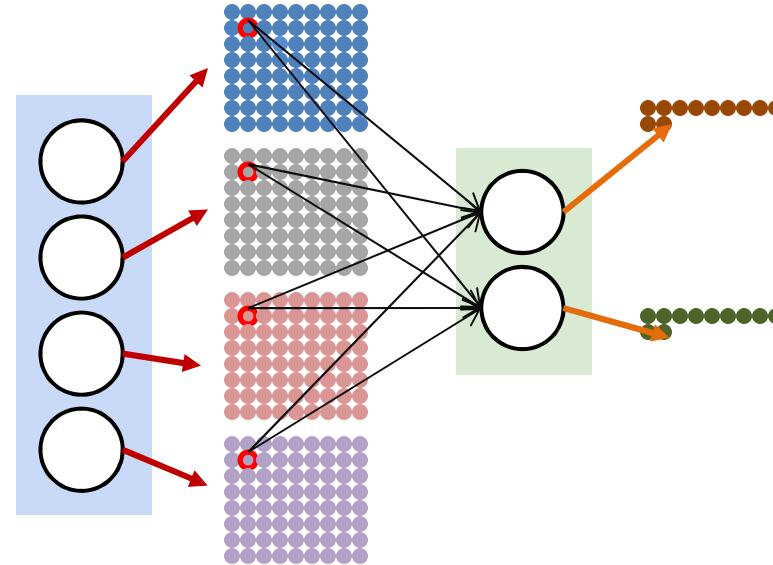
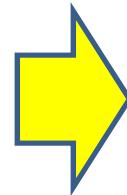
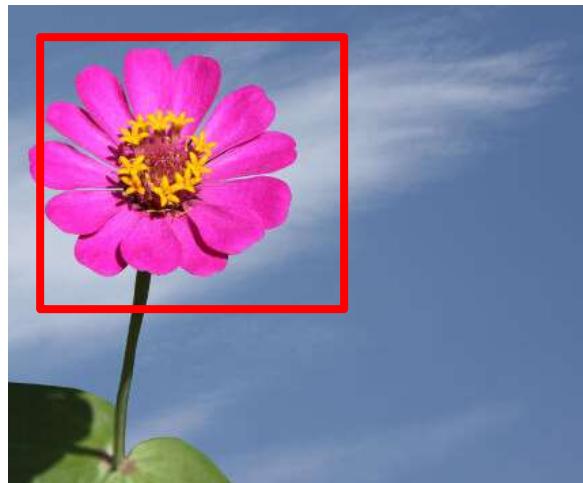
- We can recurse the logic
  - The second level neurons too are “scanning” the rectangular outputs of the first-level neurons
  - (Un)like the first level, they are jointly scanning *multiple* “pictures”
    - Each location in the output of the second level neuron considers the corresponding locations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

# Scanning: A closer look



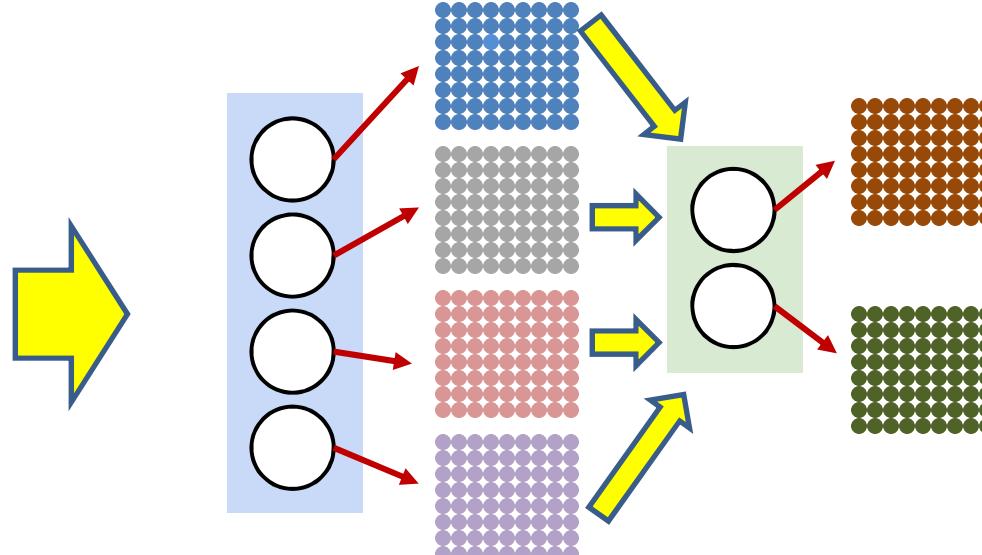
- We can recurse the logic
  - The second level neurons too are “scanning” the rectangular outputs of the first-level neurons
  - (Un)like the first level, they are jointly scanning *multiple* “pictures”
    - Each location in the output of the second level neuron considers the corresponding locations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

# Scanning: A closer look



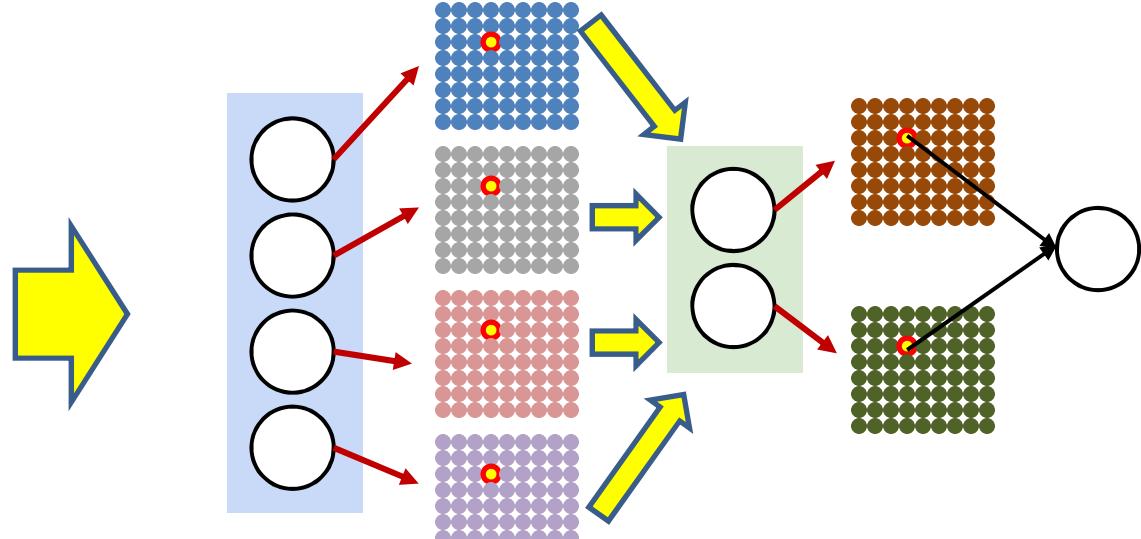
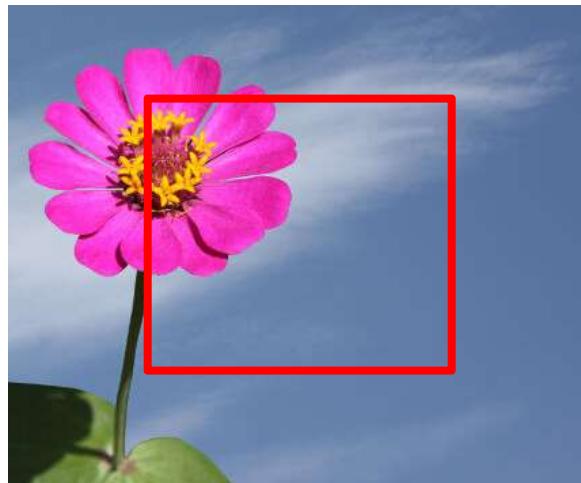
- We can recurse the logic
  - The second level neurons too are “scanning” the rectangular outputs of the first-level neurons
  - (Un)like the first level, they are jointly scanning *multiple* “pictures”
    - Each location in the output of the second level neuron considers the corresponding locations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

# Scanning: A closer look



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

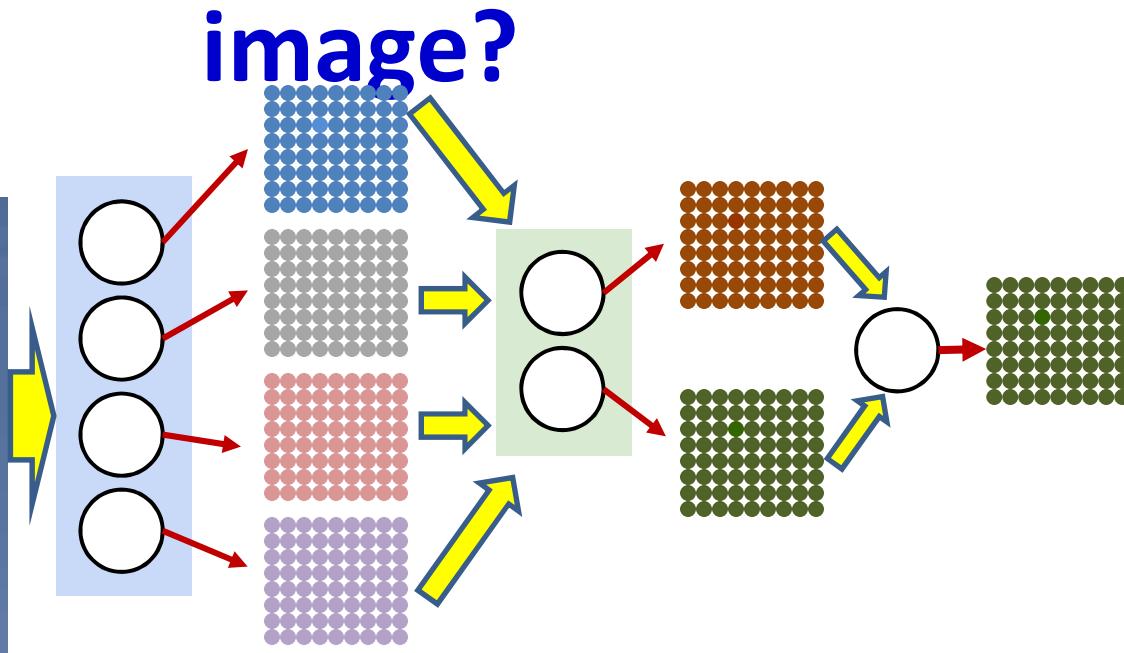
# Scanning: A closer look



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

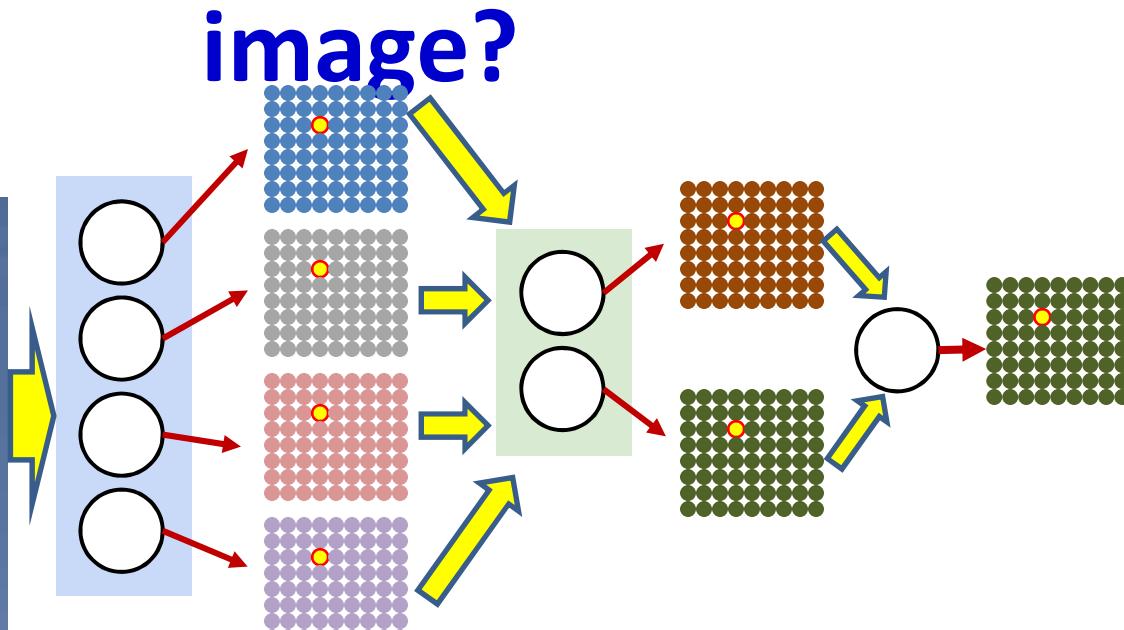
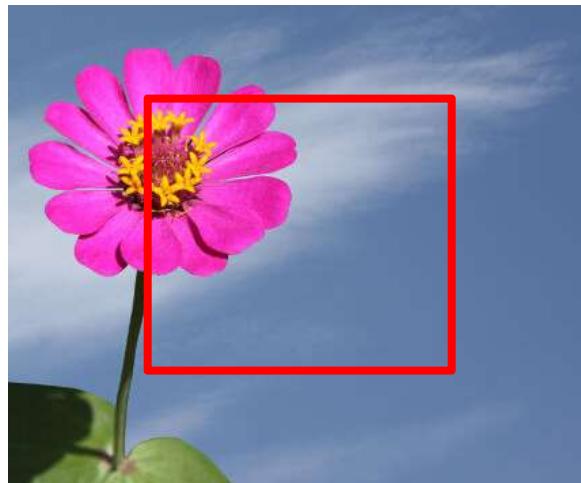
# Detecting a picture anywhere in the

## image?



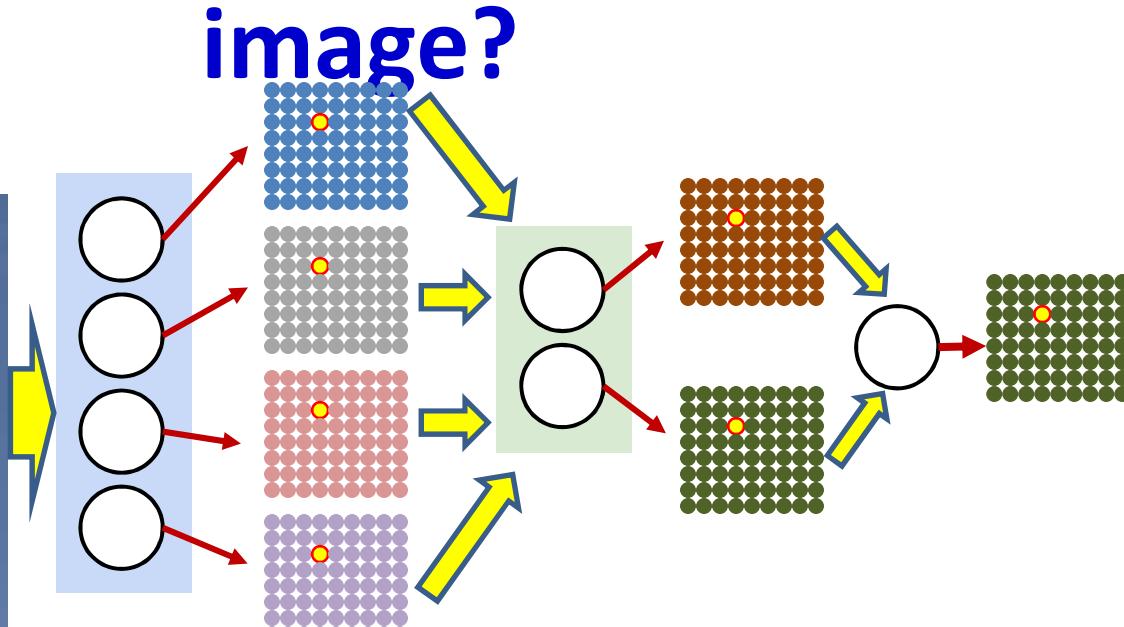
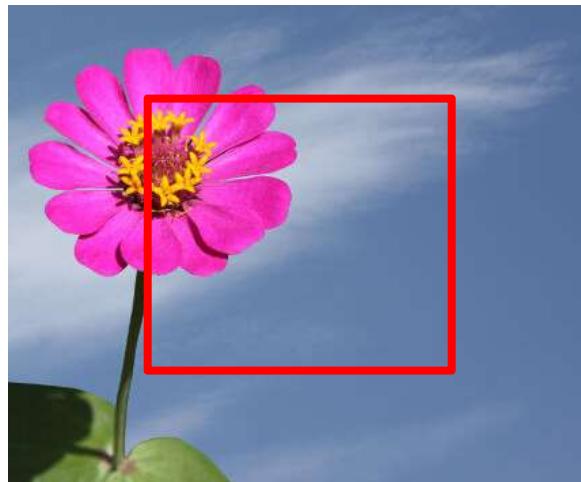
- Recursing the logic, we can create a map for the neurons in the next layer as well
  - The map is a flower detector for each location of the original image

# Detecting a picture anywhere in the image?



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

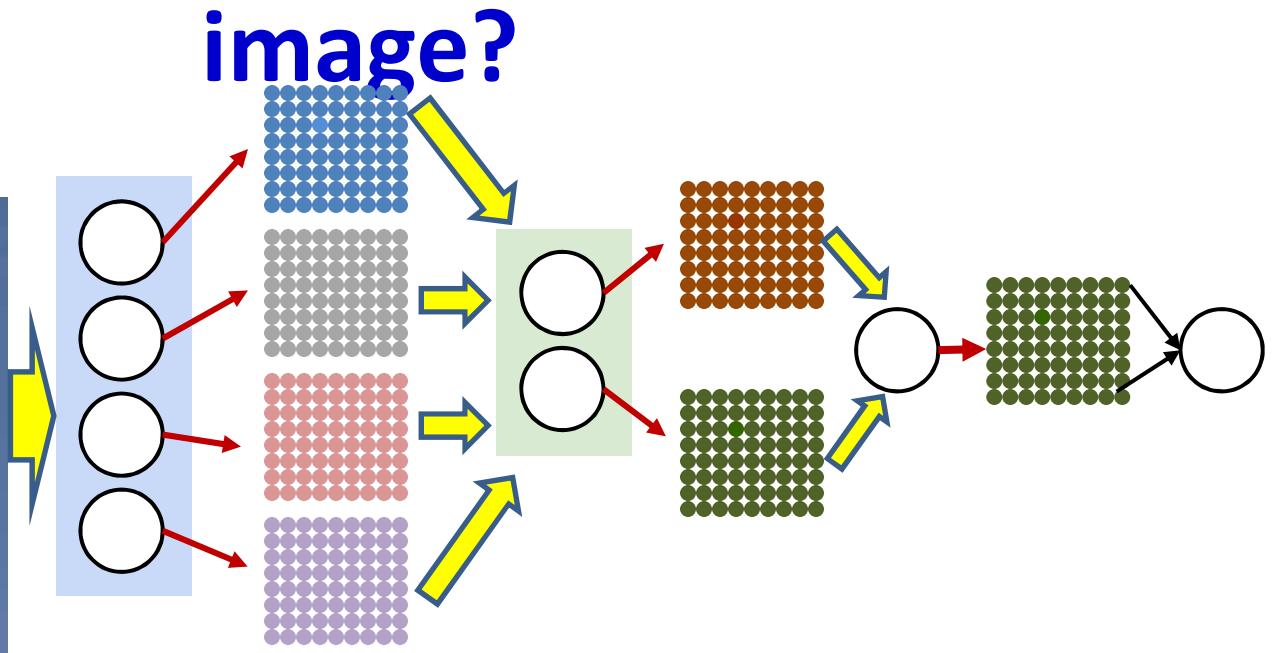
# Detecting a picture anywhere in the image?



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

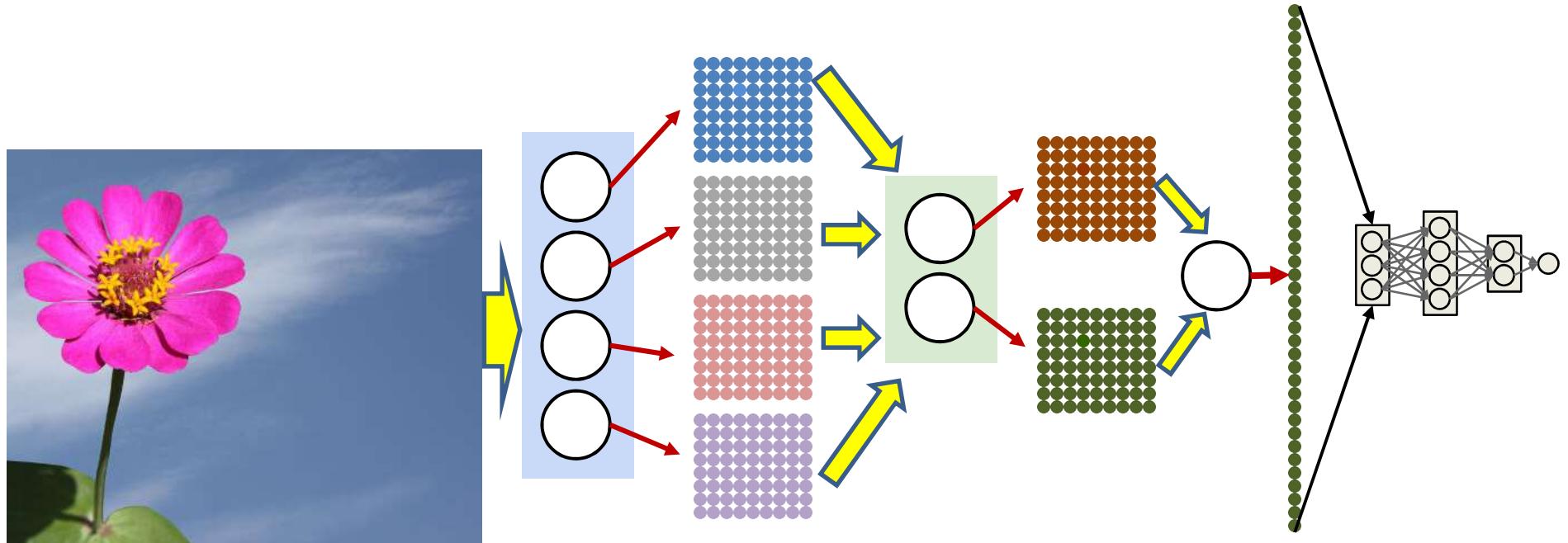
# Detecting a picture anywhere in the

## image?



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

# Detecting a picture in the image



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

# Scanning with an MLP

- $K \times K$  = 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 MLP

```
for x = 1:W-K+1
    for y = 1:H-K+1
        # First layer operates on the input
        # Unwrap WxW patch at (x,y) into a D0x1 vector
        ImgSegment = Img(1:C, x:x+K-1, y:y+K-1)
        Y(0,:,x,y) = ImgSegment
        for l = 1:L  # layers operate on vector at (x,y)
            for j = 1:Dl
                z(l,j,x,y) = 0
                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 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) = 0
                for i = 1:D_{l-1}
                    z(l,j,x,y) += w(l,i,j)Y(l-1,i,x,y)
                end
                Y(l,j,x,y) = activation(z(l,j,x,y))
            end
        end
    end
end

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

# Scanning with MLP

```
for x = 1:W-K+1
    for y = 1:H-K+1
        for l = 1:L
            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) = 0
                for i = 1:Dl-1
                    z(l,j,x,y) += w(l,i,j)Y(l-1,i,x,y)
                end
                Y(l,j,x,y) = activation(z(l,j,x,y))
            end
        end
    end
end

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

# layers operate on vector at (x,y)

# Scanning with MLP

```
for l = 1:L
    for j = 1:Dl
        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,j,x,y) = 0
                for i = 1:Dl-1
                    z(l,j,x,y) += w(l,i,j)Y(l-1,i,x,y)
                end
                Y(l,j,x,y) = activation(z(l,j,x,y))
            end
        end
    end
end

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

# Scanning with MLP

```
for l = 1:L
    for j = 1:Dl
        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,j,x,y) = 0
                for i = 1:Dl-1
                    z(l,j,x,y) += w(l,i,j)Y(l-1,i,x,y)
                end
                Y(l,j,x,y) = activation(z(l,j,x,y))
            end
        end
    end
end

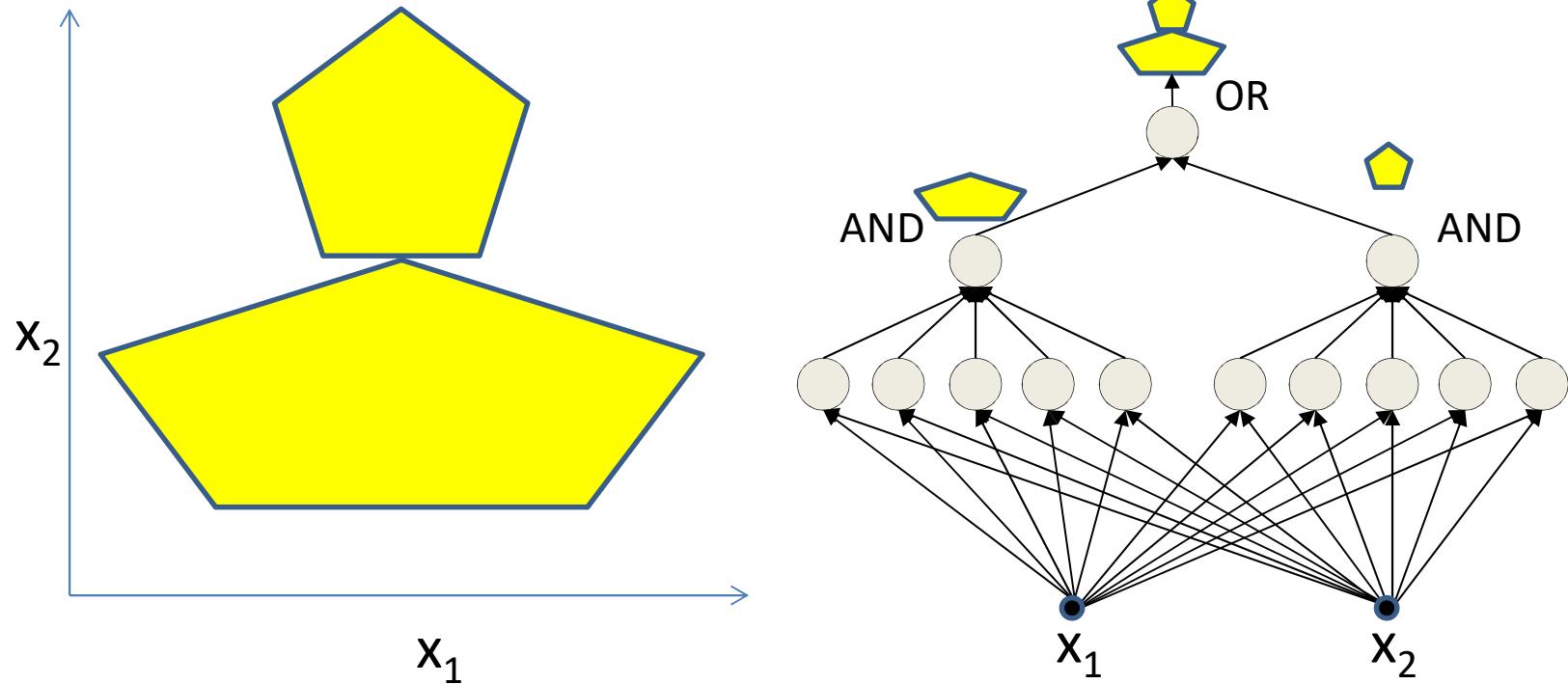
Y = softmax( Y(L,:,1,1) .. Y(L,:,W-K+1,H-K+1) )
```

# 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)
            Y(l,x,y) = activation(z(l,x,y))
    end
end

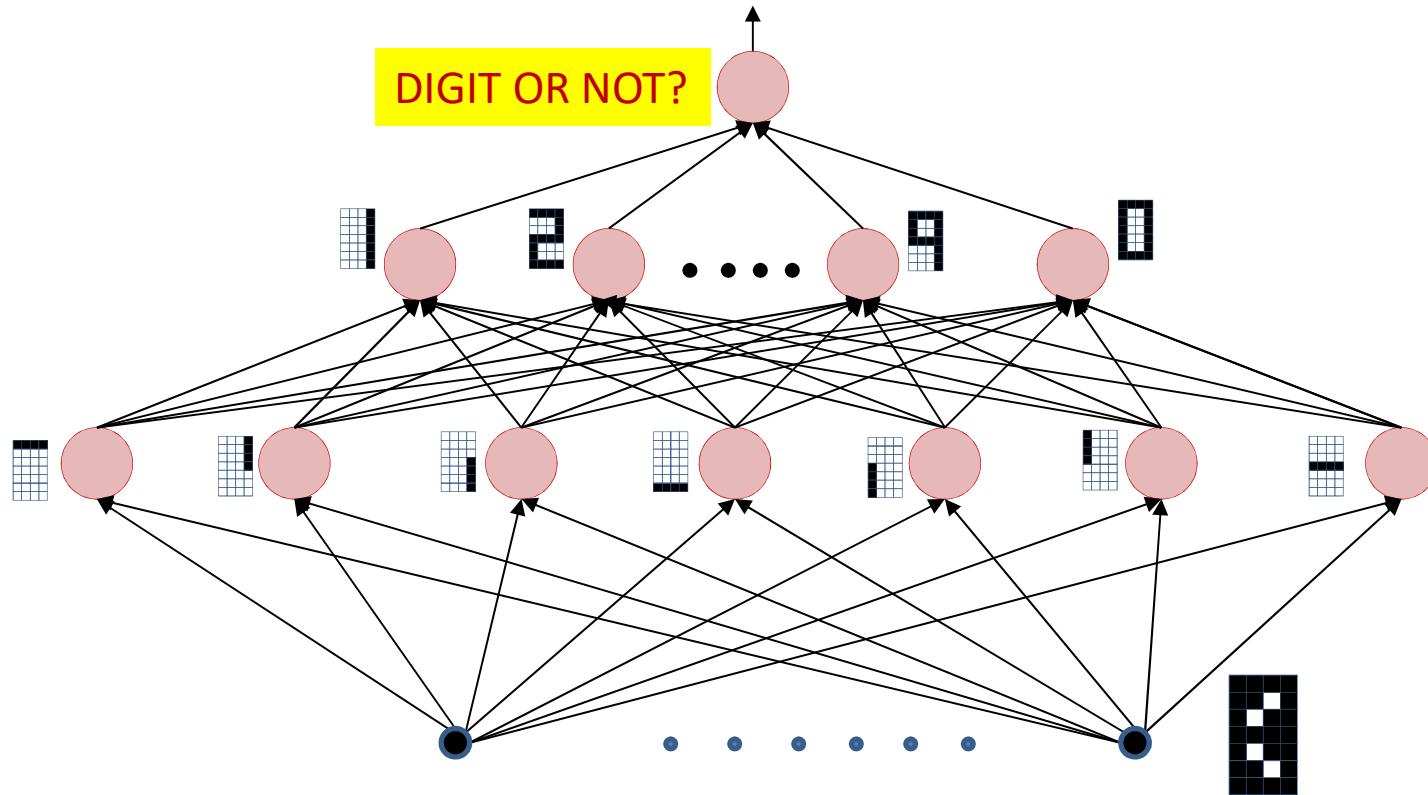
Y = softmax( Y(L,1,1) .. Y(L,W-K+1,H-K+1) )
```

# Recall: What does an MLP learn?



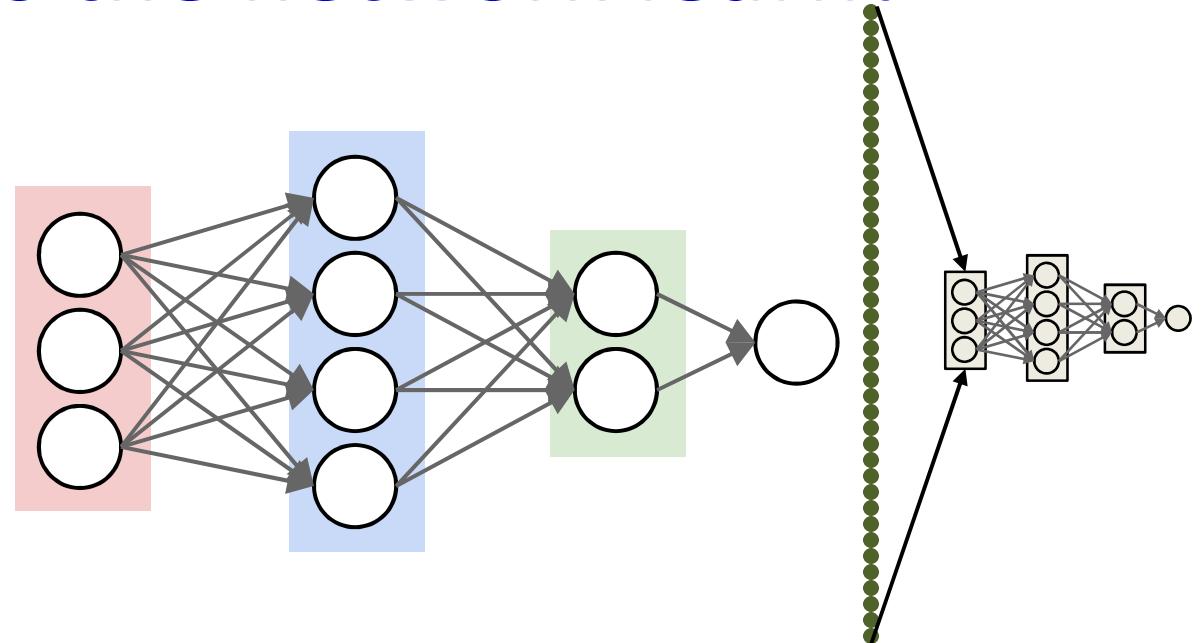
- The lowest layers of the network capture simple patterns
  - The linear decision boundaries in this example
- The next layer captures more complex patterns
  - The polygons
- The next one captures still more complex patterns..

# Recall: How does an MLP represent patterns



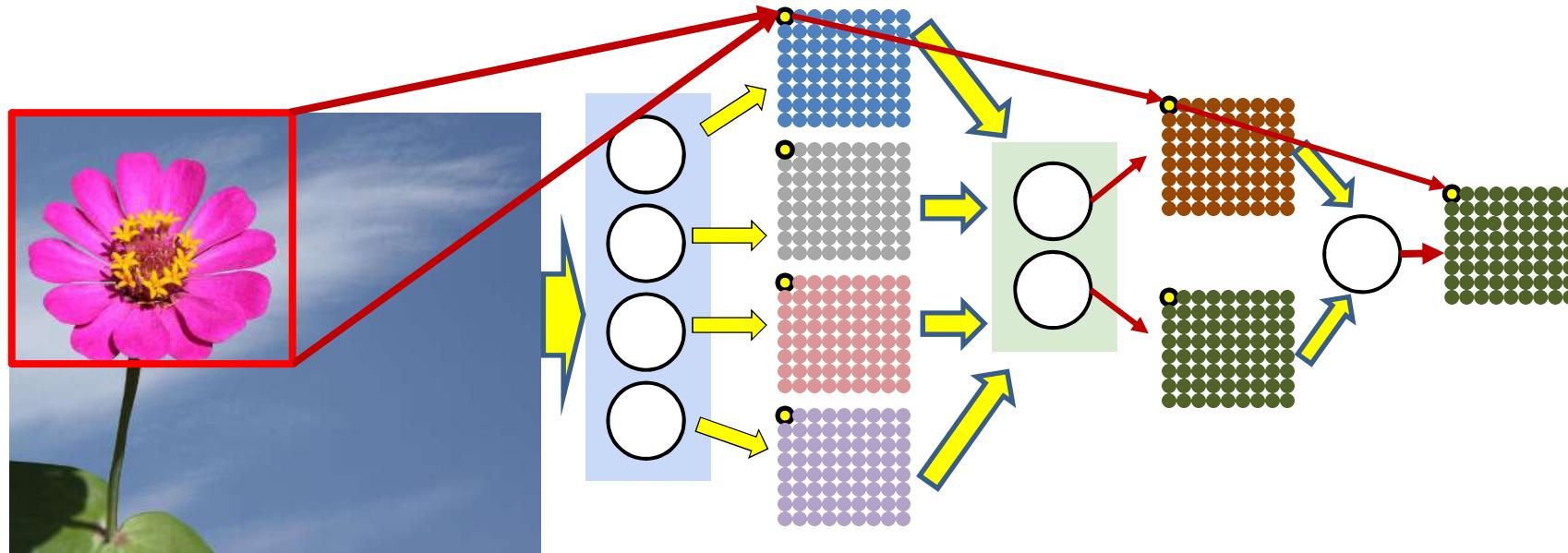
- The neurons in an MLP *build up* complex patterns from simple pattern hierarchically
  - Each layer learns to “detect” simple combinations of the patterns detected by earlier layers

# Returning to our problem: What does the network learn?



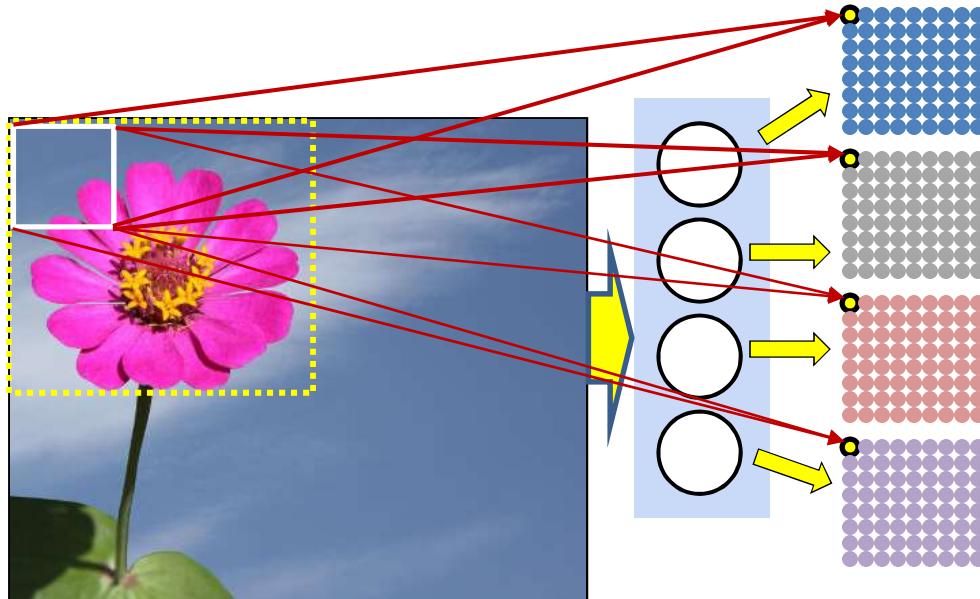
- The entire MLP looks for a flower-like pattern at each location

# The behavior of the layers



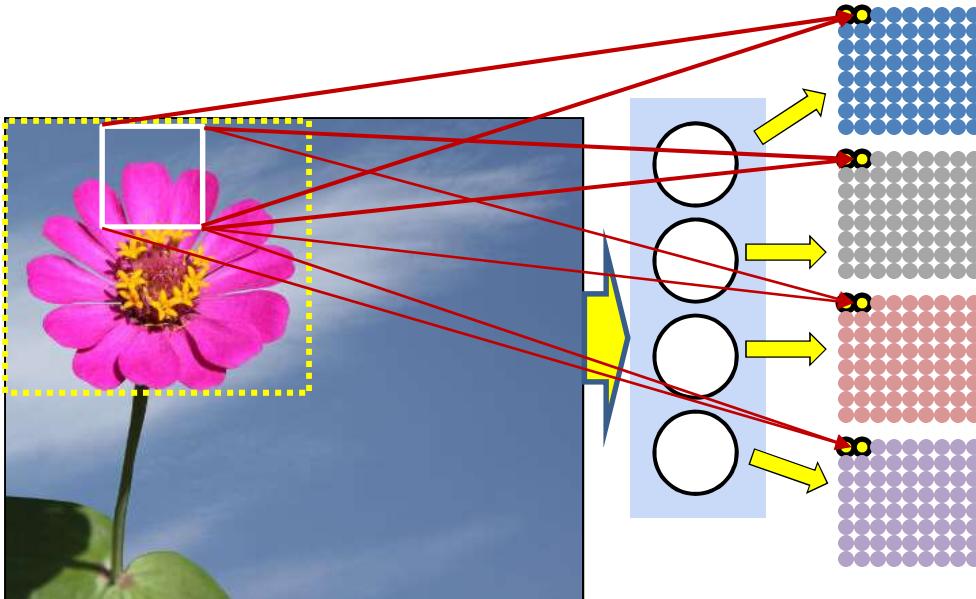
- The first layer neurons “look” at the entire “block” to extract block-level features
  - Subsequent layers only perform classification over these block-level features
- **The first layer neurons is responsible for evaluating the *entire block of pixels***
  - **Subsequent layers only look at a *single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

# Distributing the scan



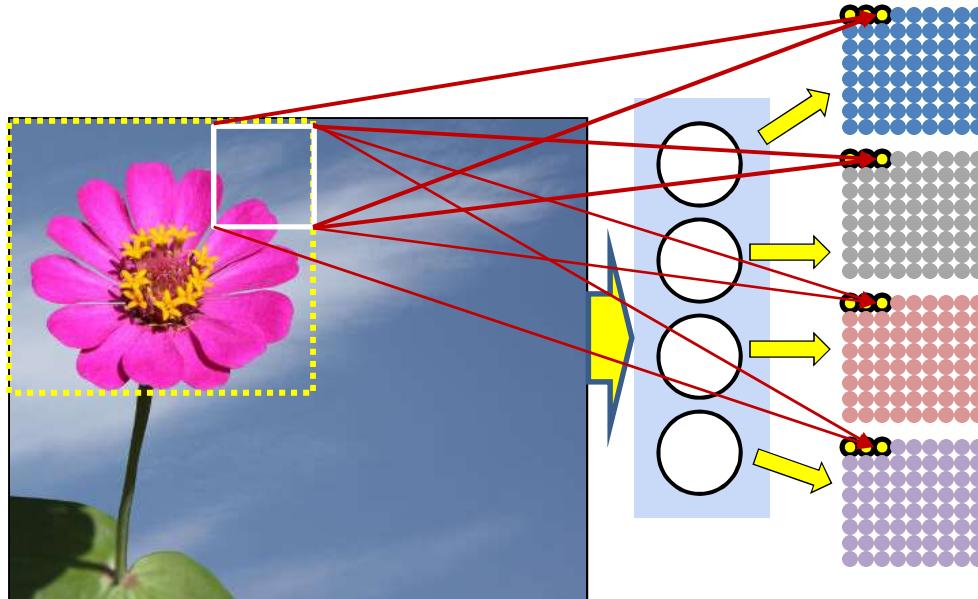
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels

# Distributing the scan



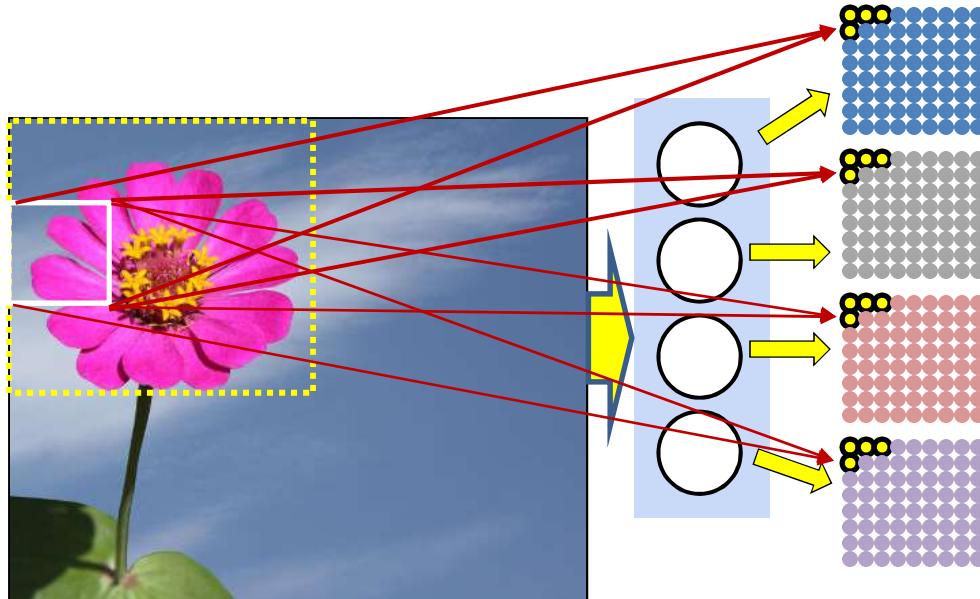
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels

# Distributing the scan



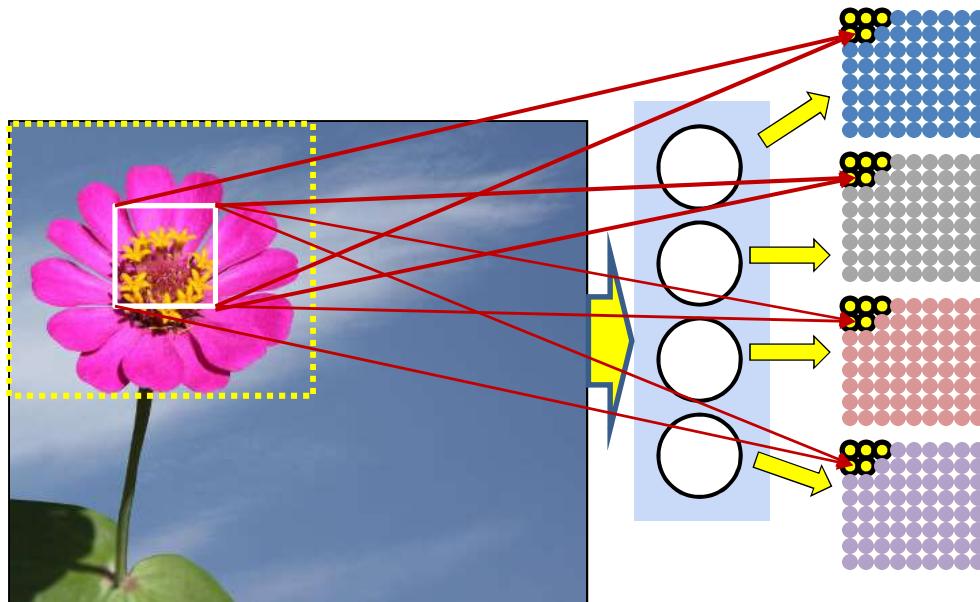
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels

# Distributing the scan



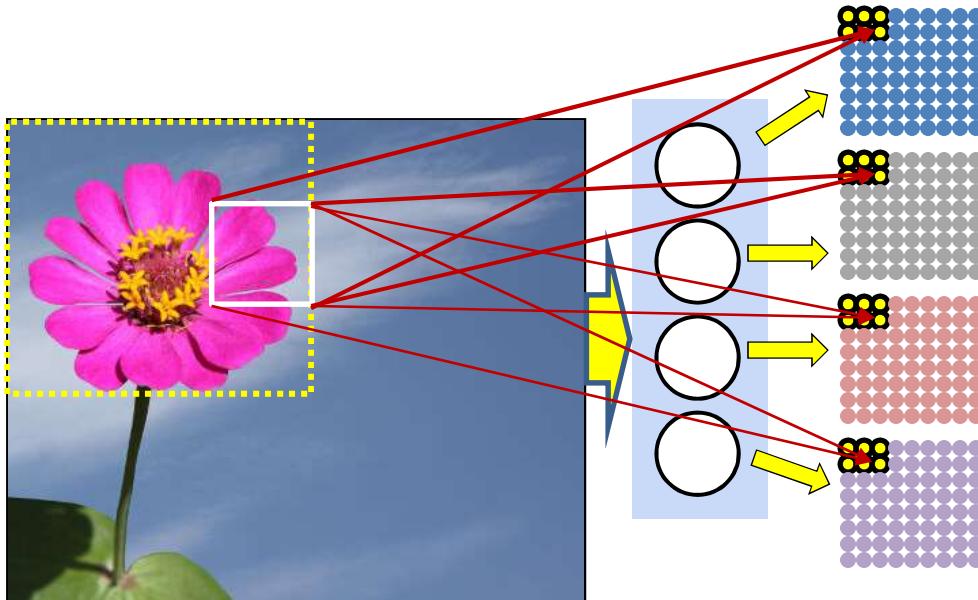
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels

# Distributing the scan



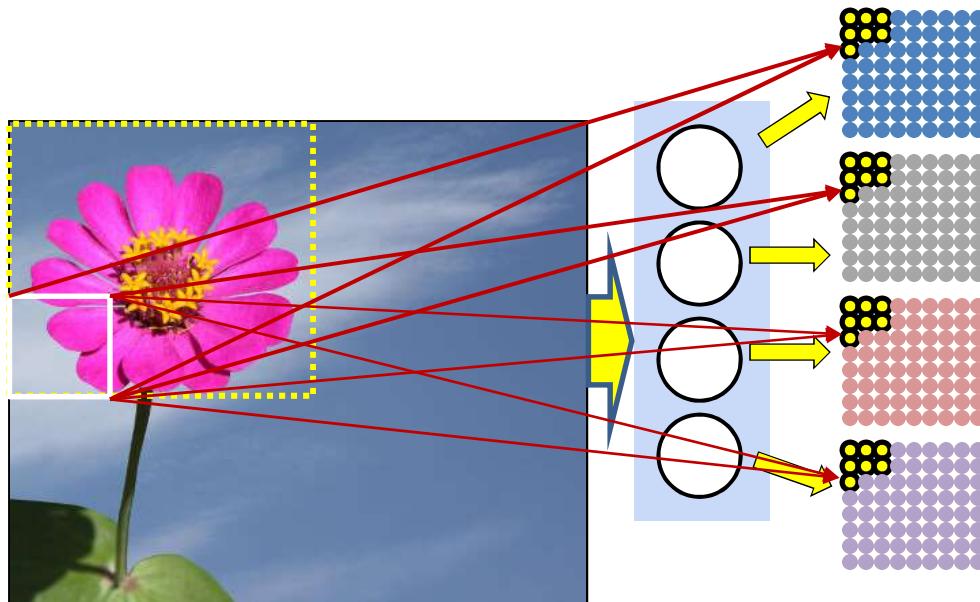
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels

# Distributing the scan



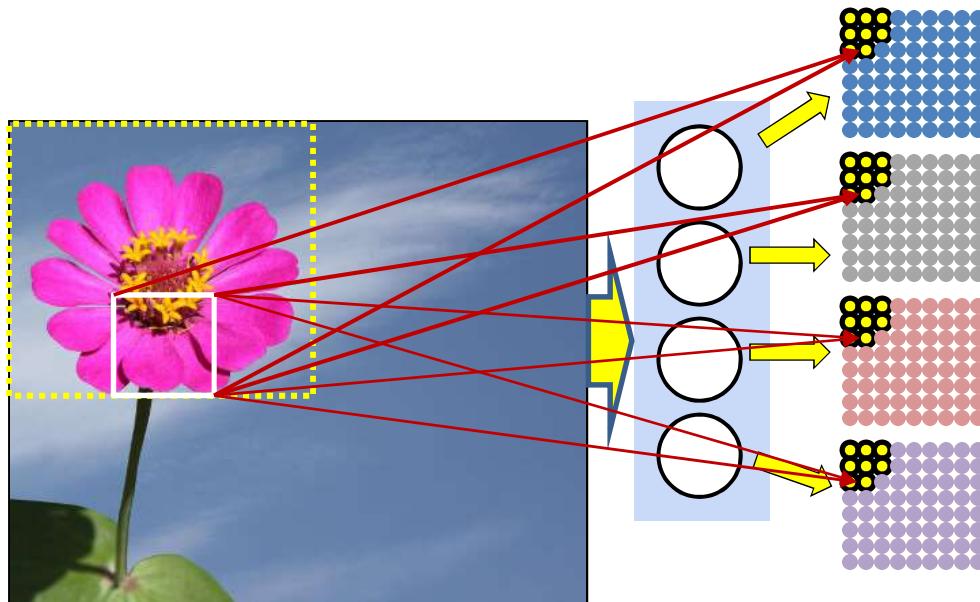
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels

# Distributing the scan



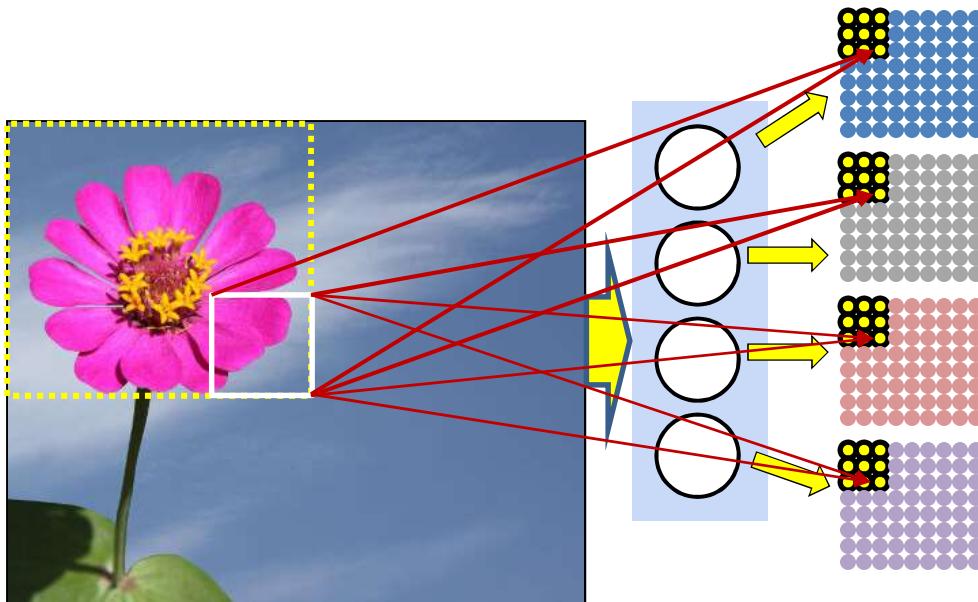
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels

# Distributing the scan



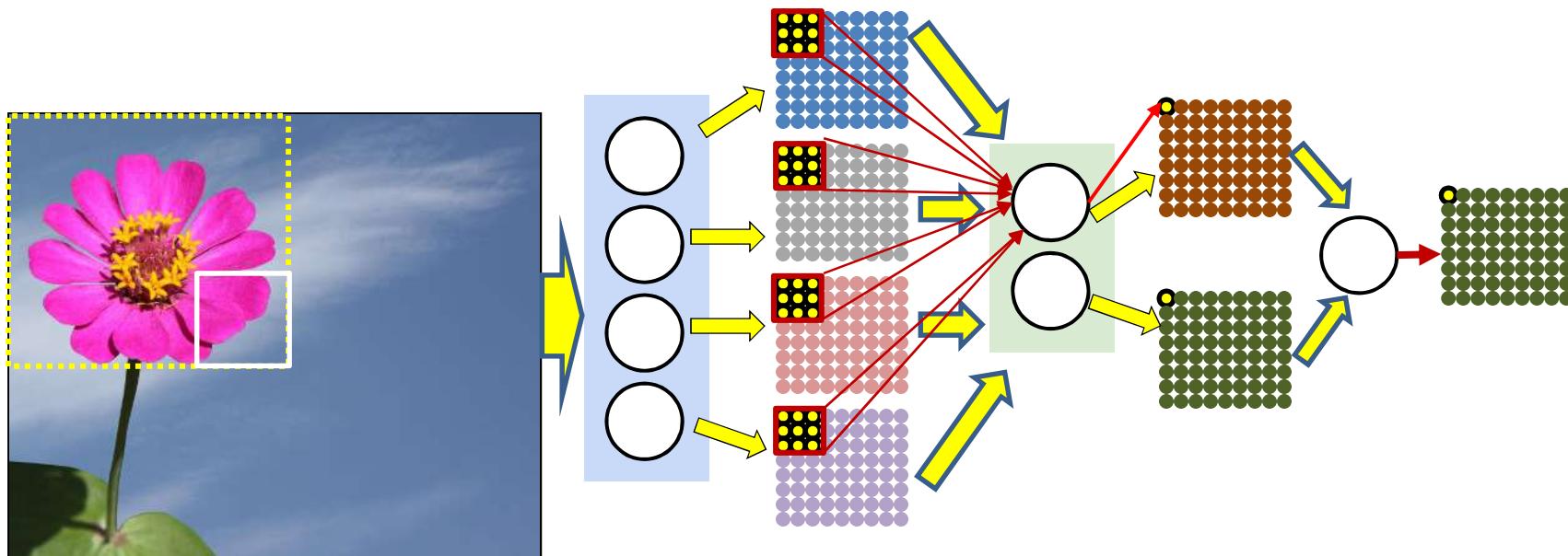
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels

# Distributing the scan



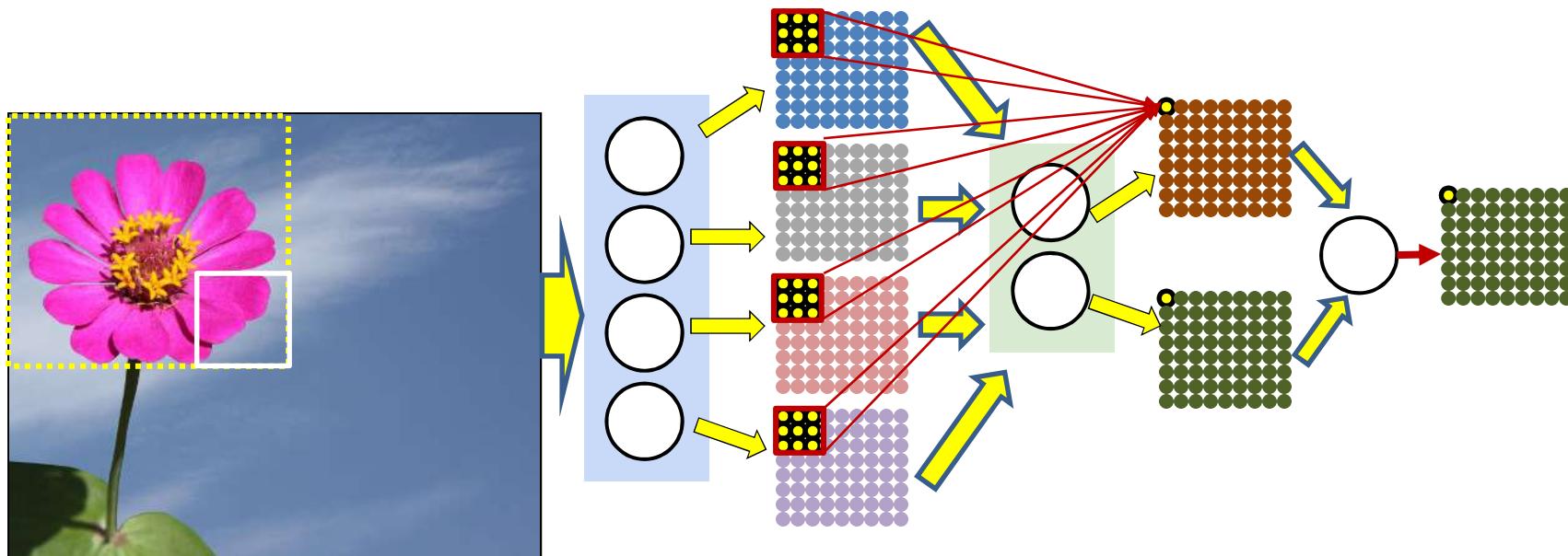
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels

# Distributing the scan



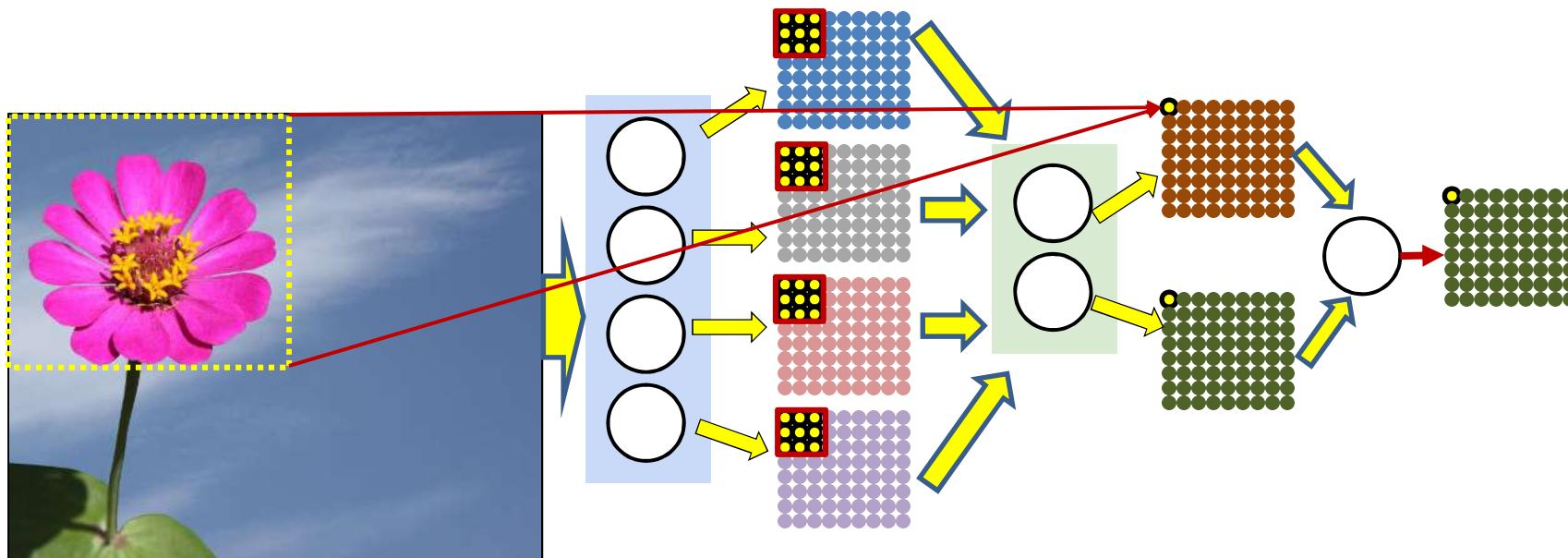
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates blocks of outputs from the first layer

# Distributing the scan



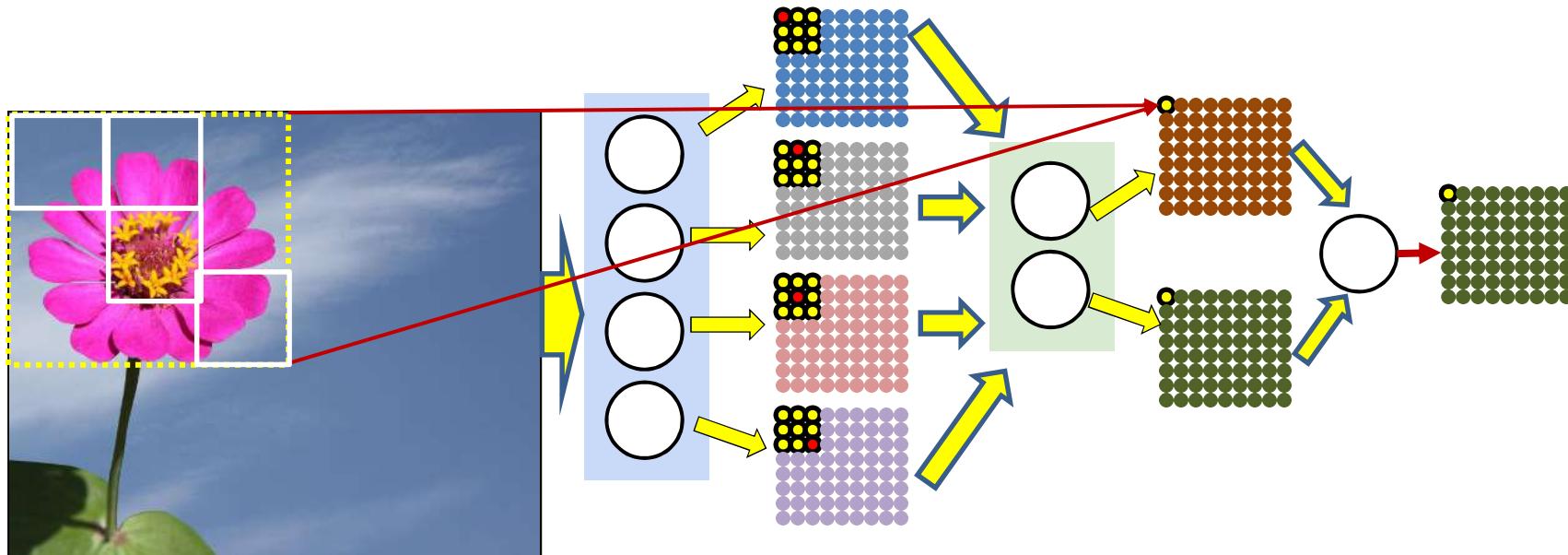
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates blocks of outputs from the first layer

# Distributing the scan



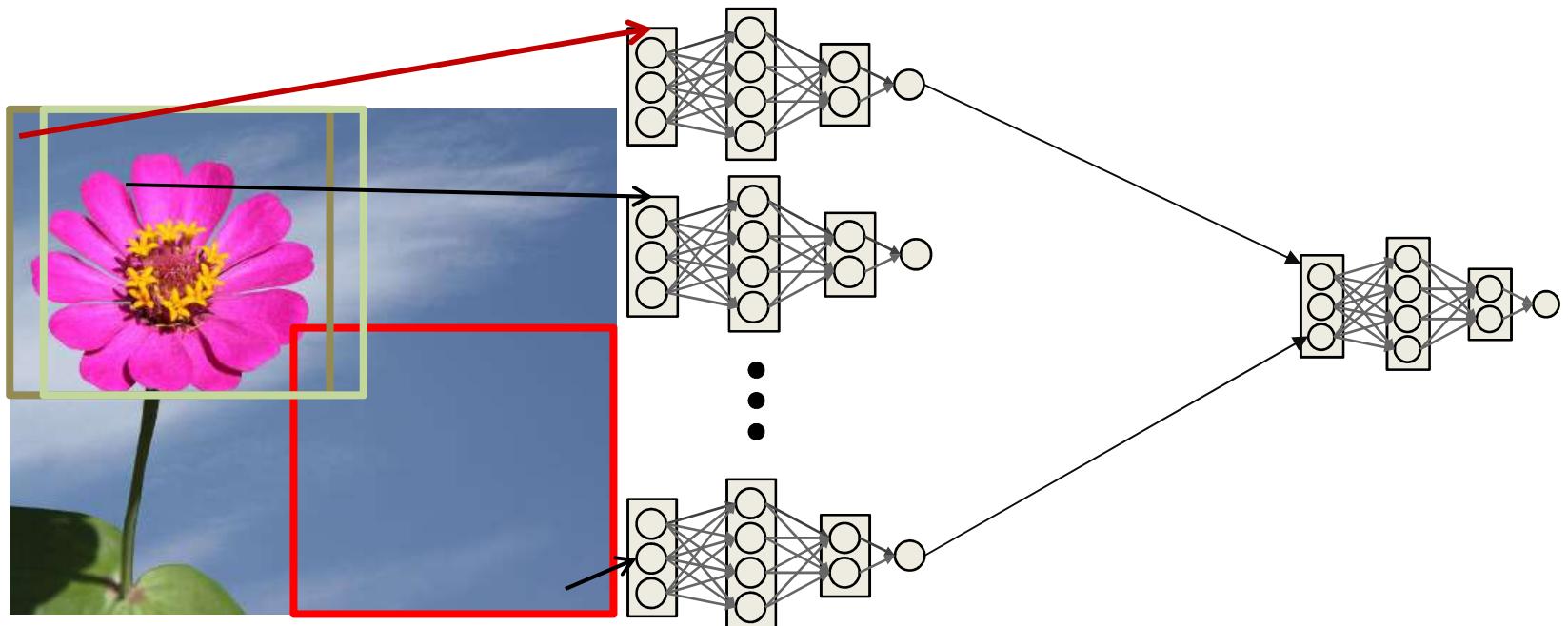
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates blocks of outputs from the first layer
  - This effectively evaluates the larger block of the original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

# Distributing the scan



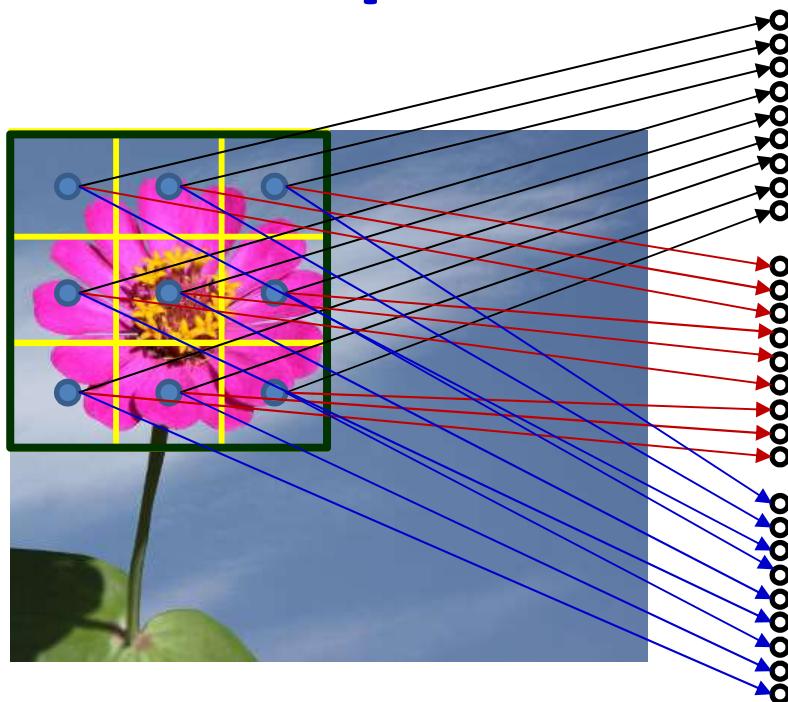
- The higher layer implicitly learns the *arrangement* of sub patterns that represents the larger pattern (the flower in this case)

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



- With a minor modification...

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

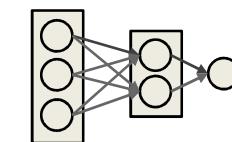


Colors indicate neurons  
with shared parameters

Layer 1

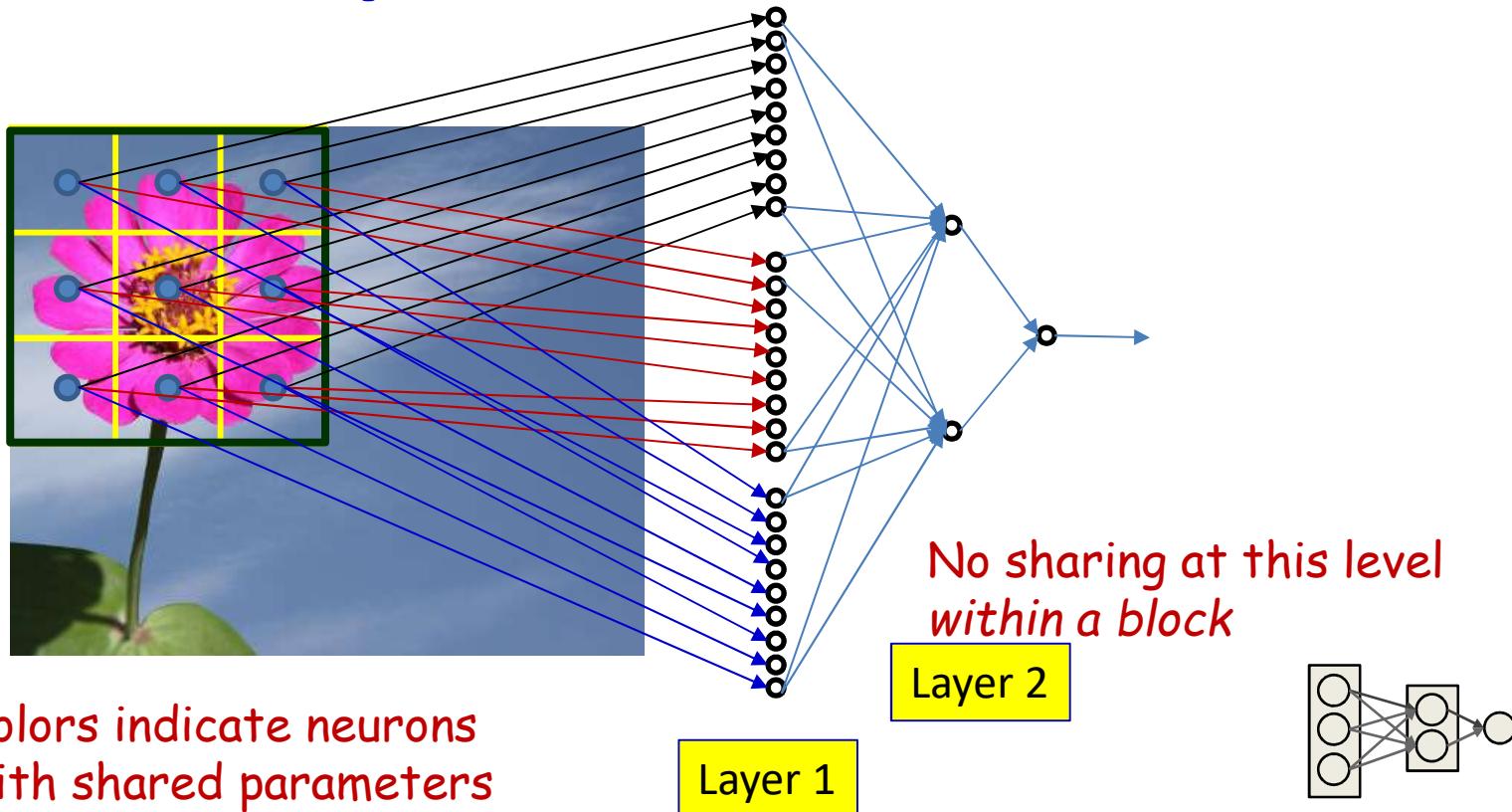
Each arrow represents an entire set  
of weights over the smaller cell

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



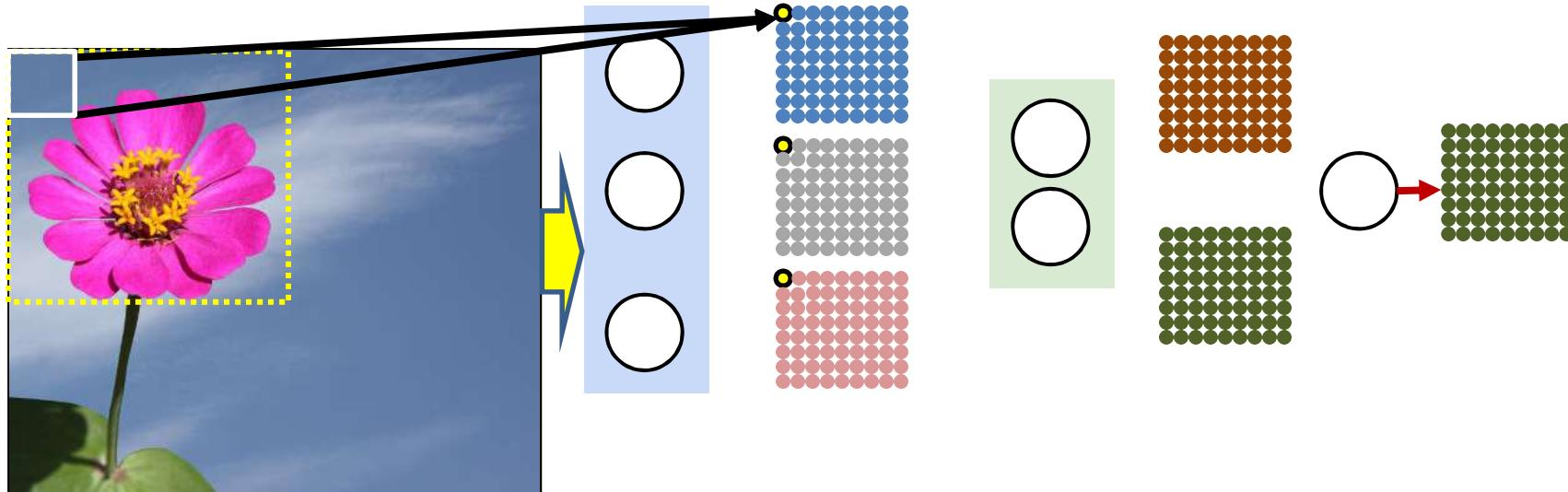
- The network that analyzes individual blocks is now itself a shared parameter network..

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



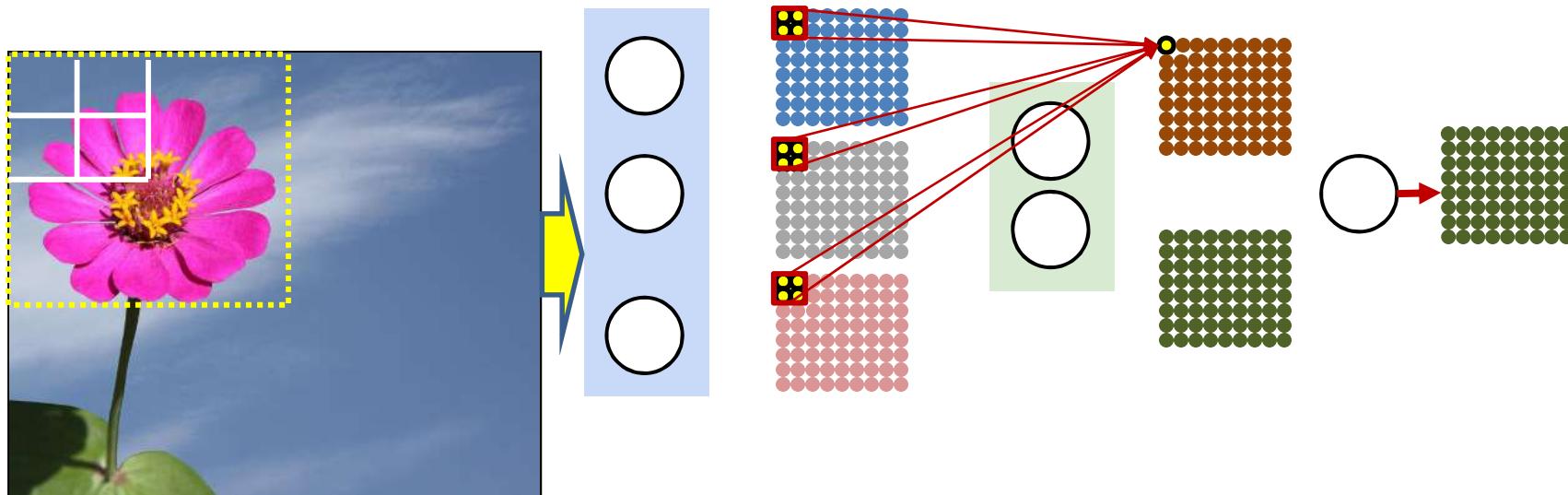
- The network that analyzes individual blocks is now itself a shared parameter network..

# This logic can be recursed



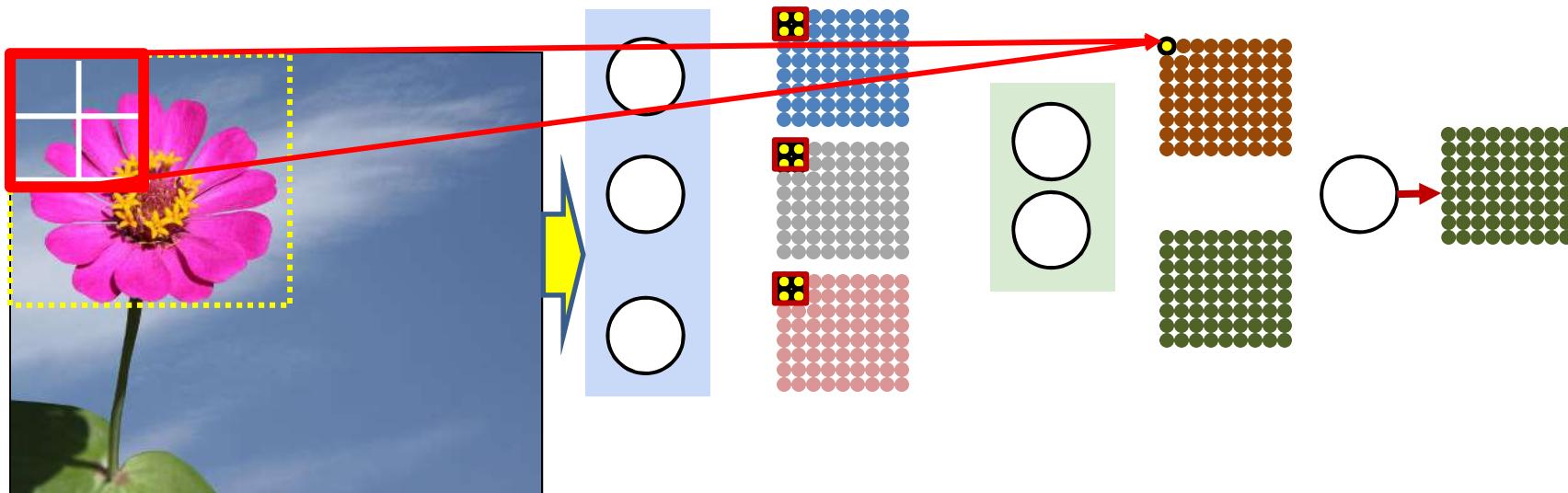
- Building the pattern over 3 layers

# This logic can be recursed



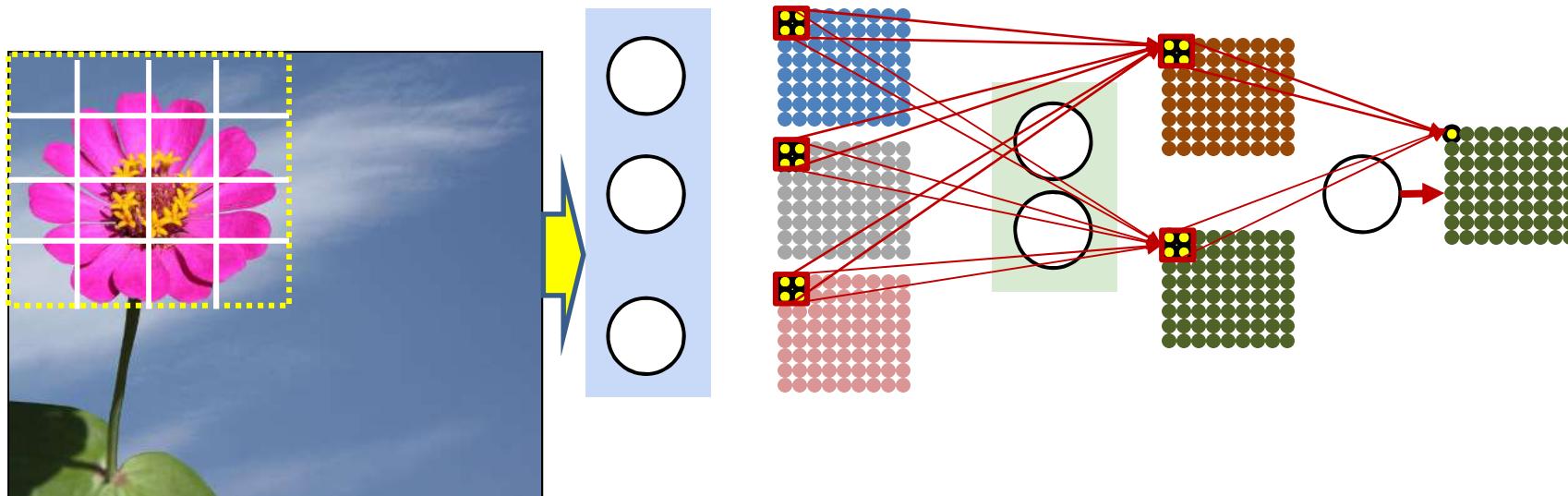
- Building the pattern over 3 layers

# This logic can be recursed



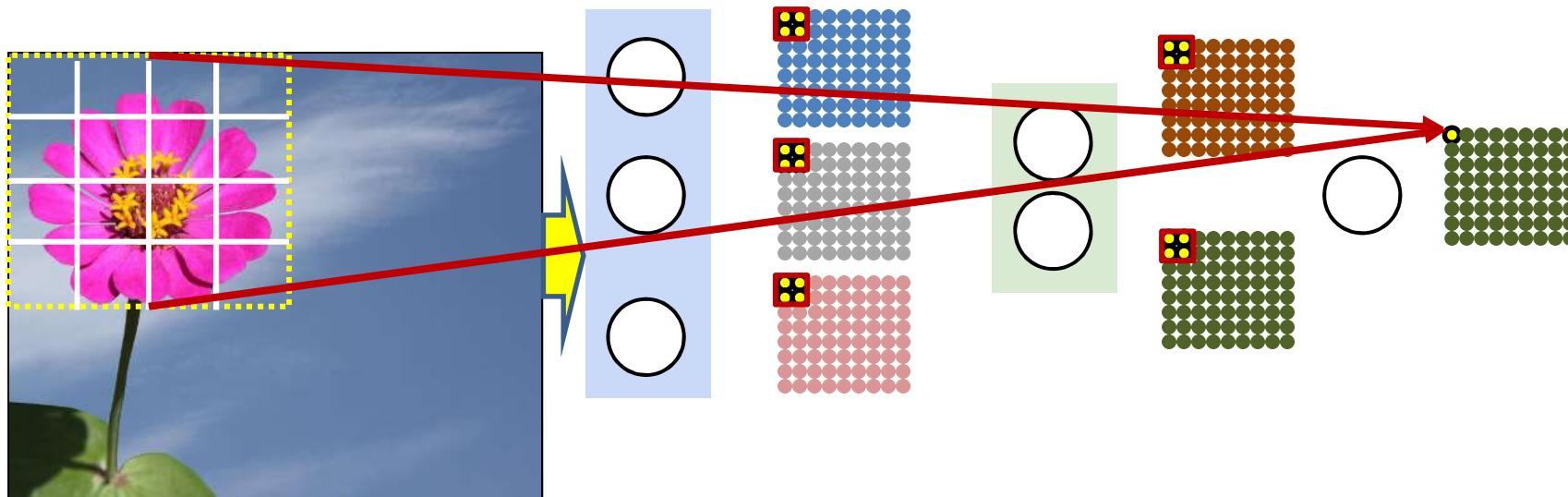
- Building the pattern over 3 layers

# This logic can be recursed



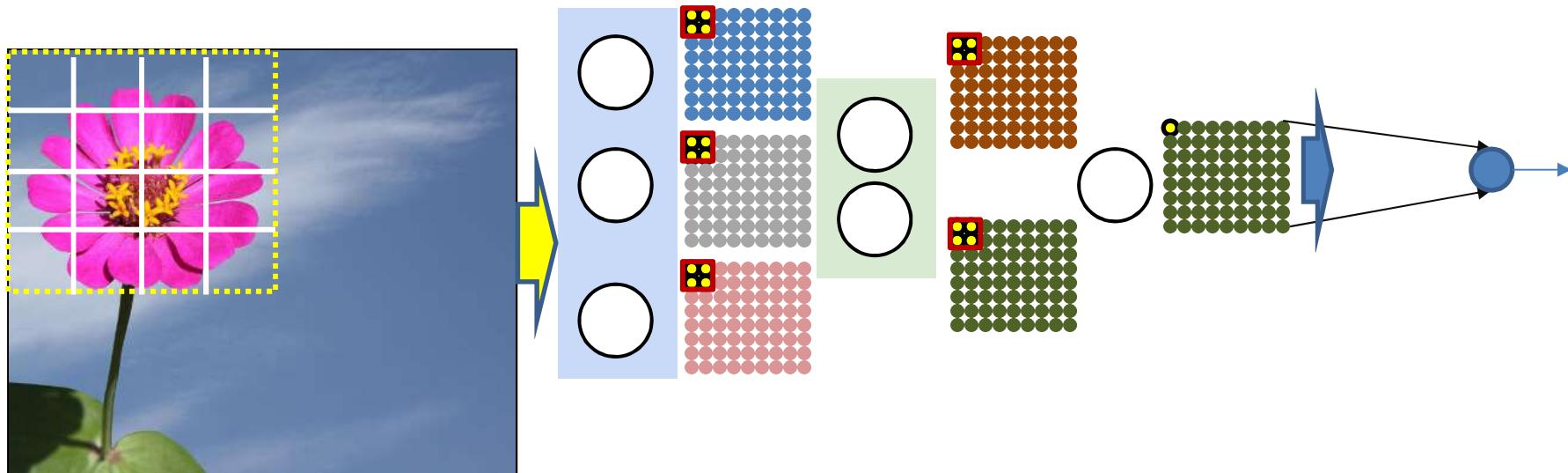
- Building the pattern over 3 layers

# This logic can be recursed



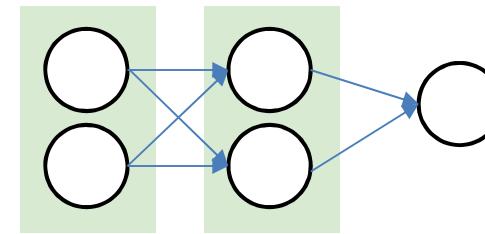
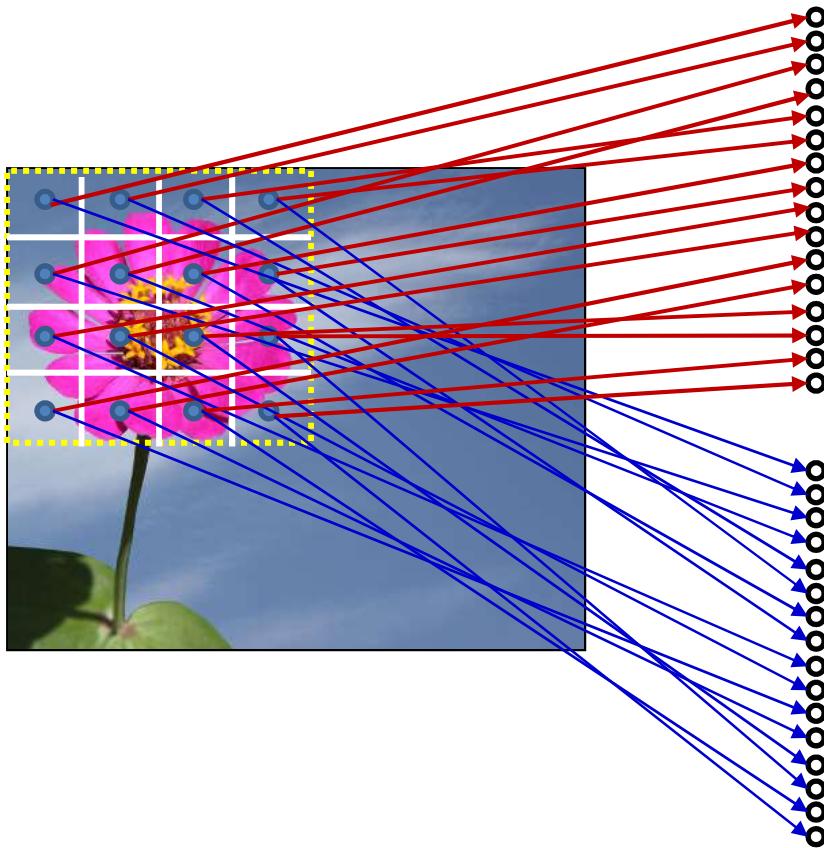
- Building the pattern over 3 layers

# Does the picture have a flower



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

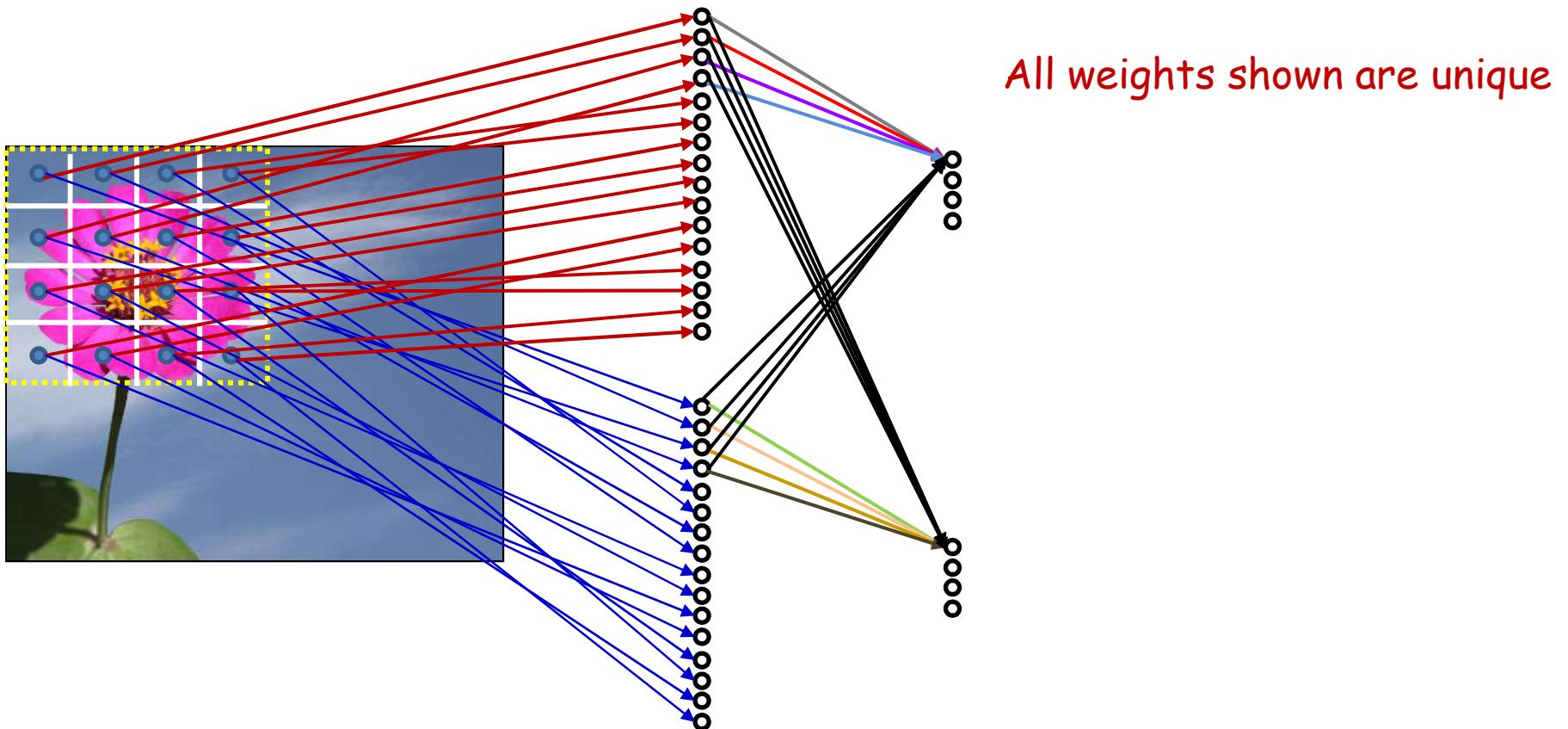
# The 3-layer shared parameter net



Showing a simpler 2x2x1 network to fit on the slide

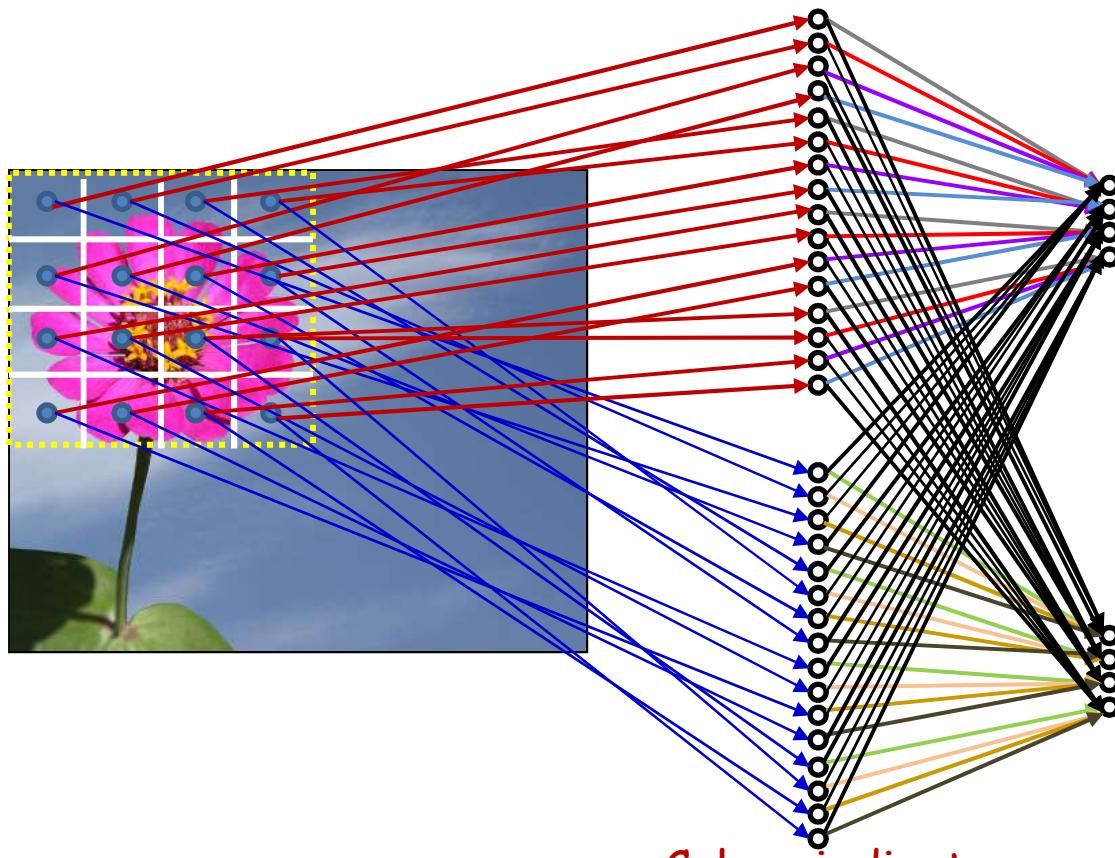
- Building the pattern over 3 layers

# The 3-layer shared parameter net



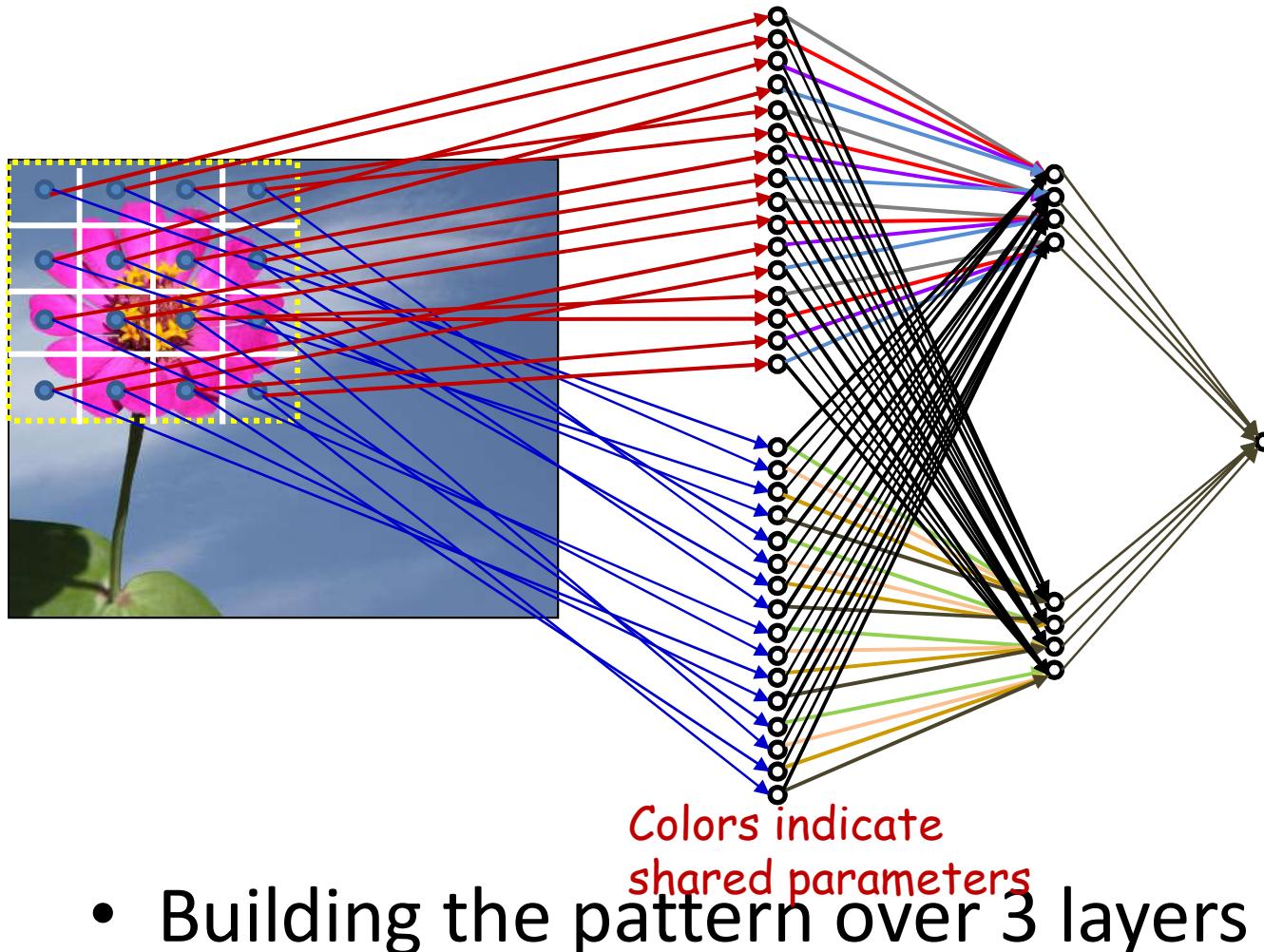
- Building the pattern over 3 layers

# The 3-layer shared parameter net

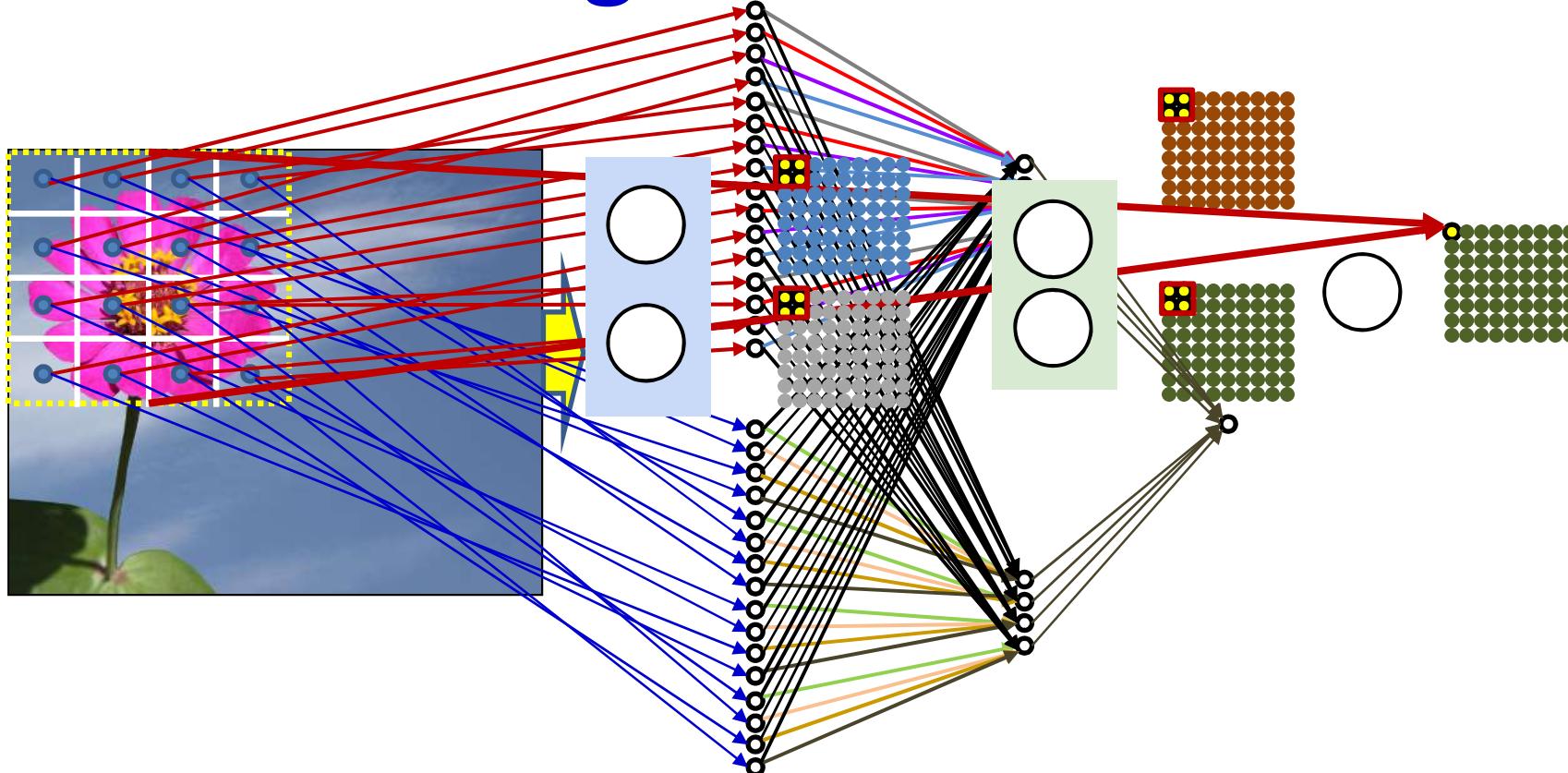


- Building the pattern over 3 layers

# The 3-layer shared parameter net



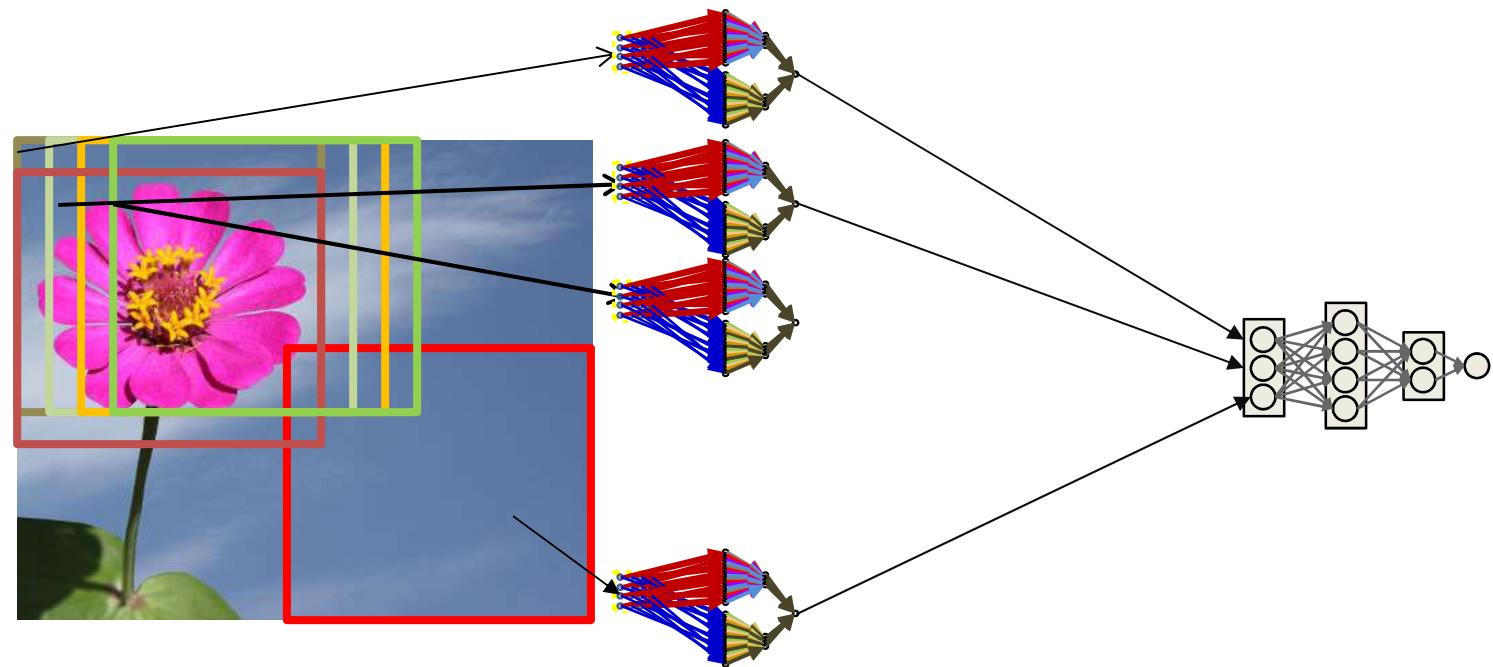
# This logic can be recursed



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*

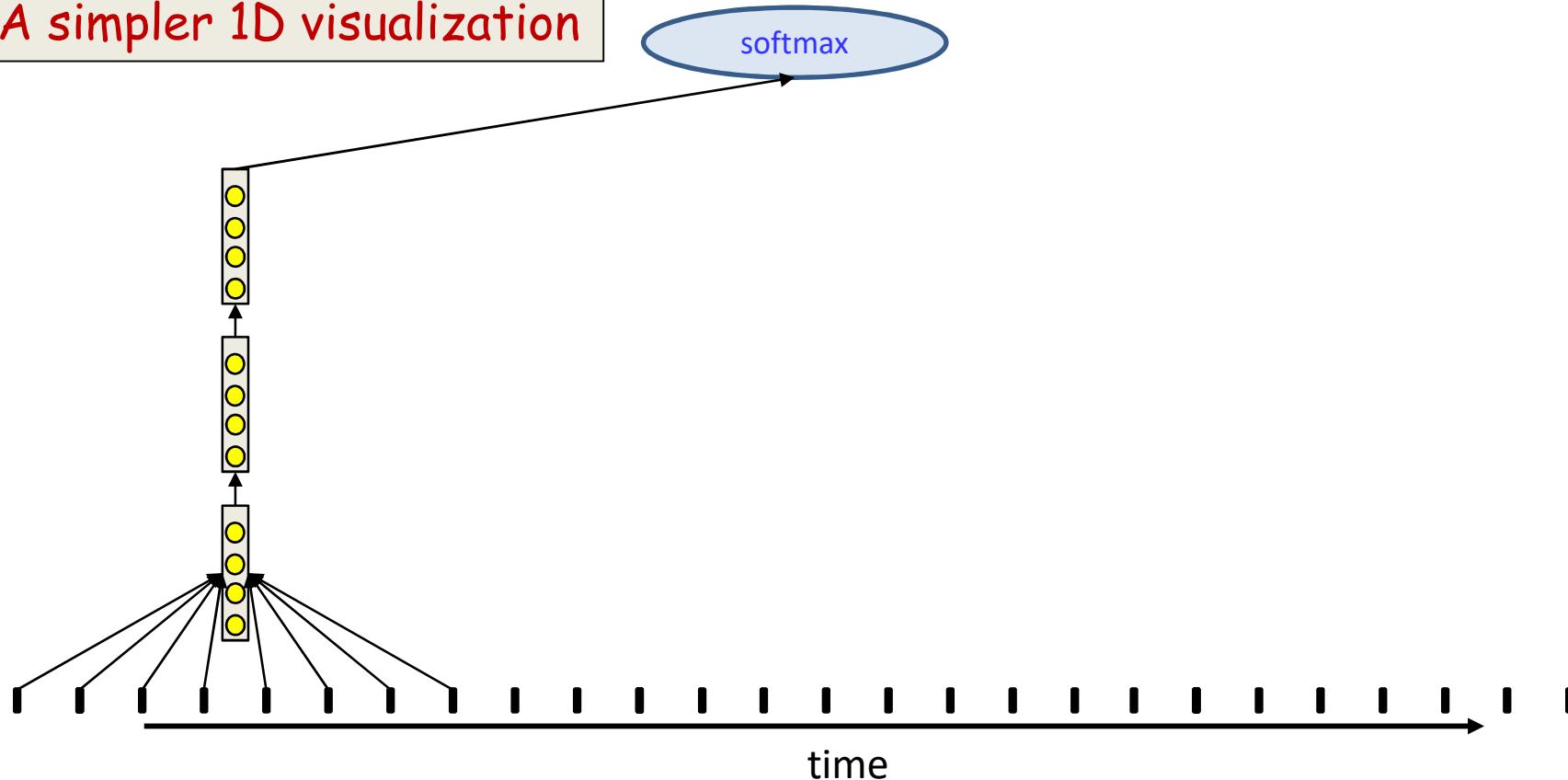
# Using hierarchical build-up of features



- 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

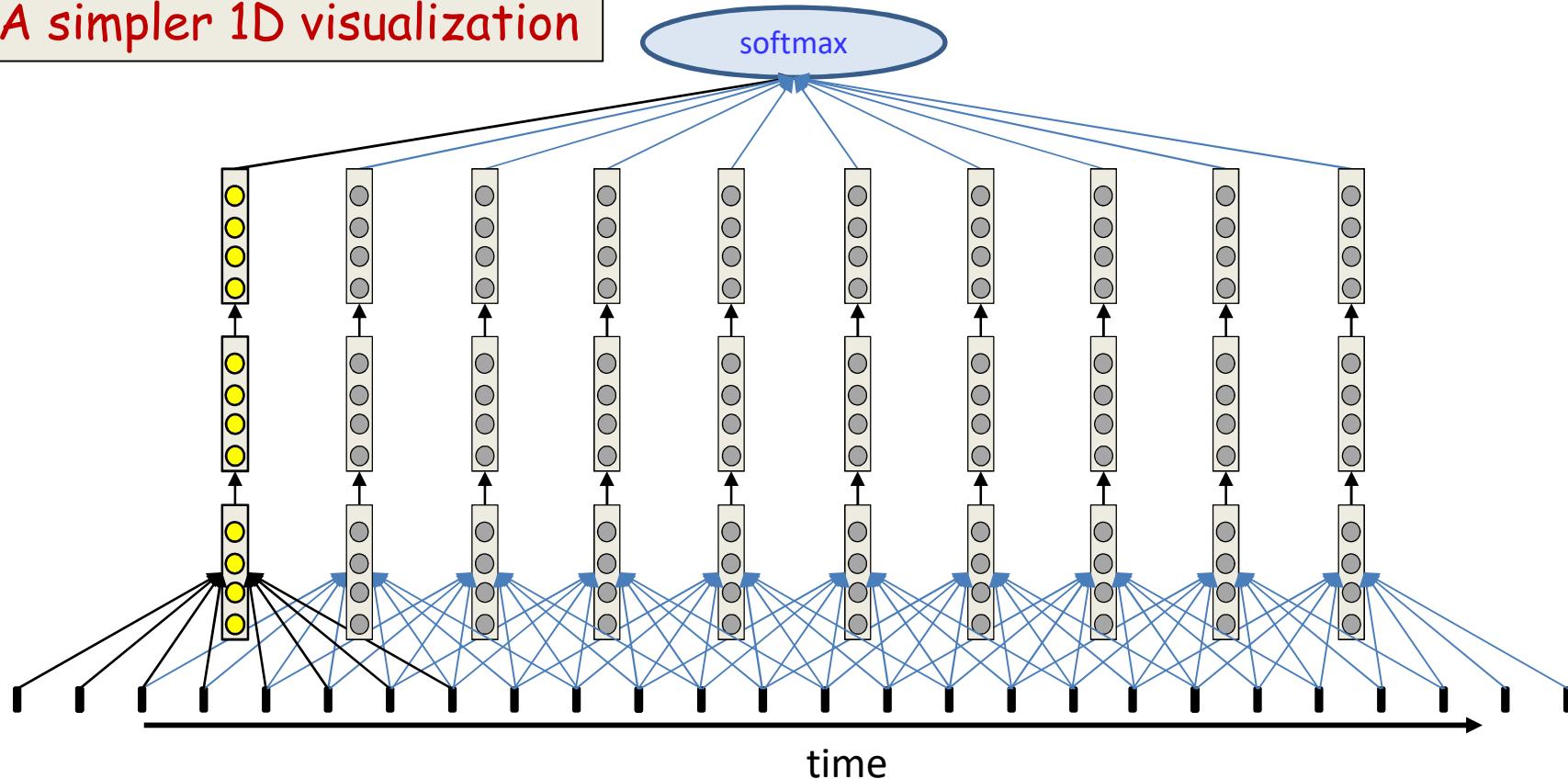
A simpler 1D visualization



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

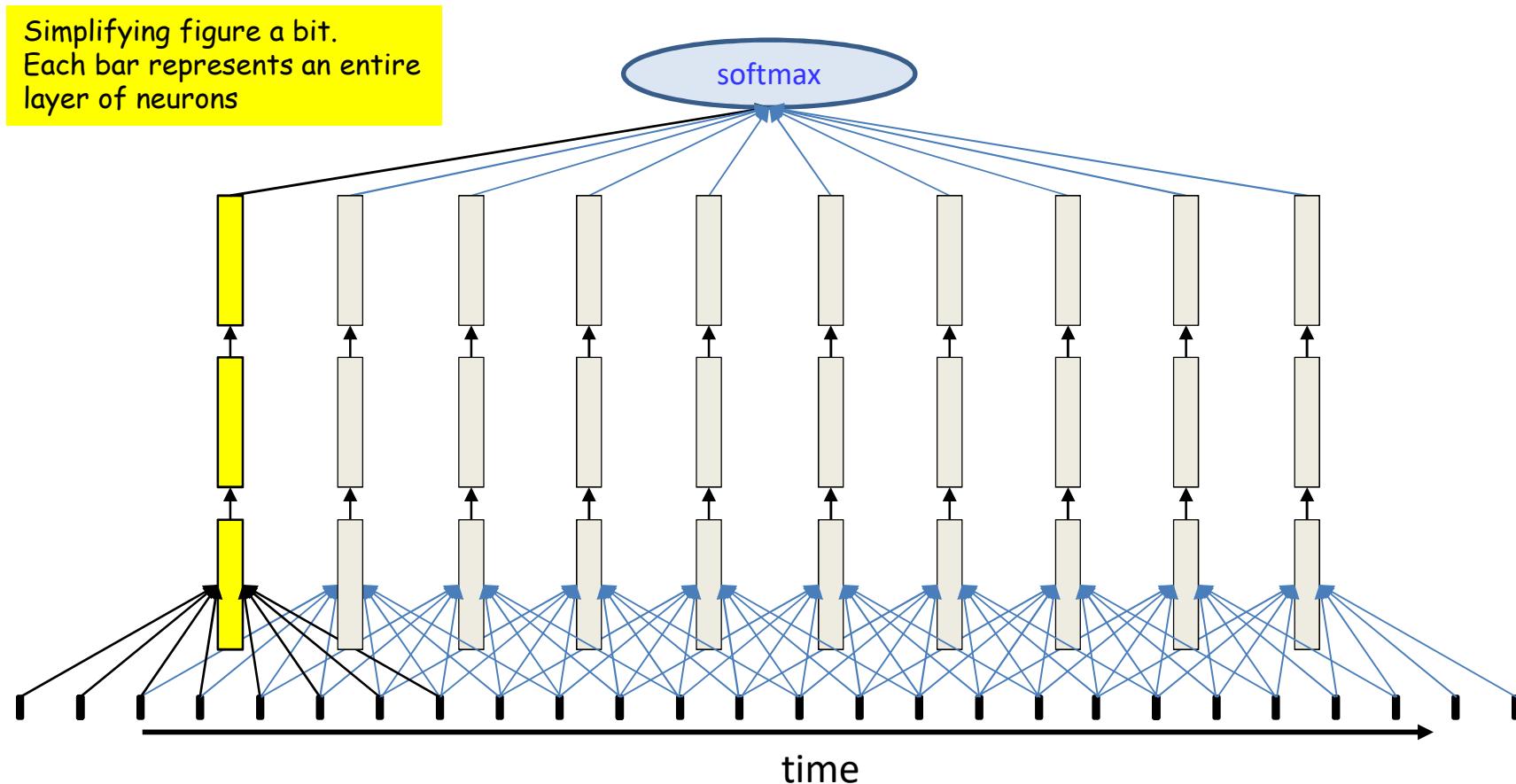
# Scanning without distribution

A simpler 1D visualization



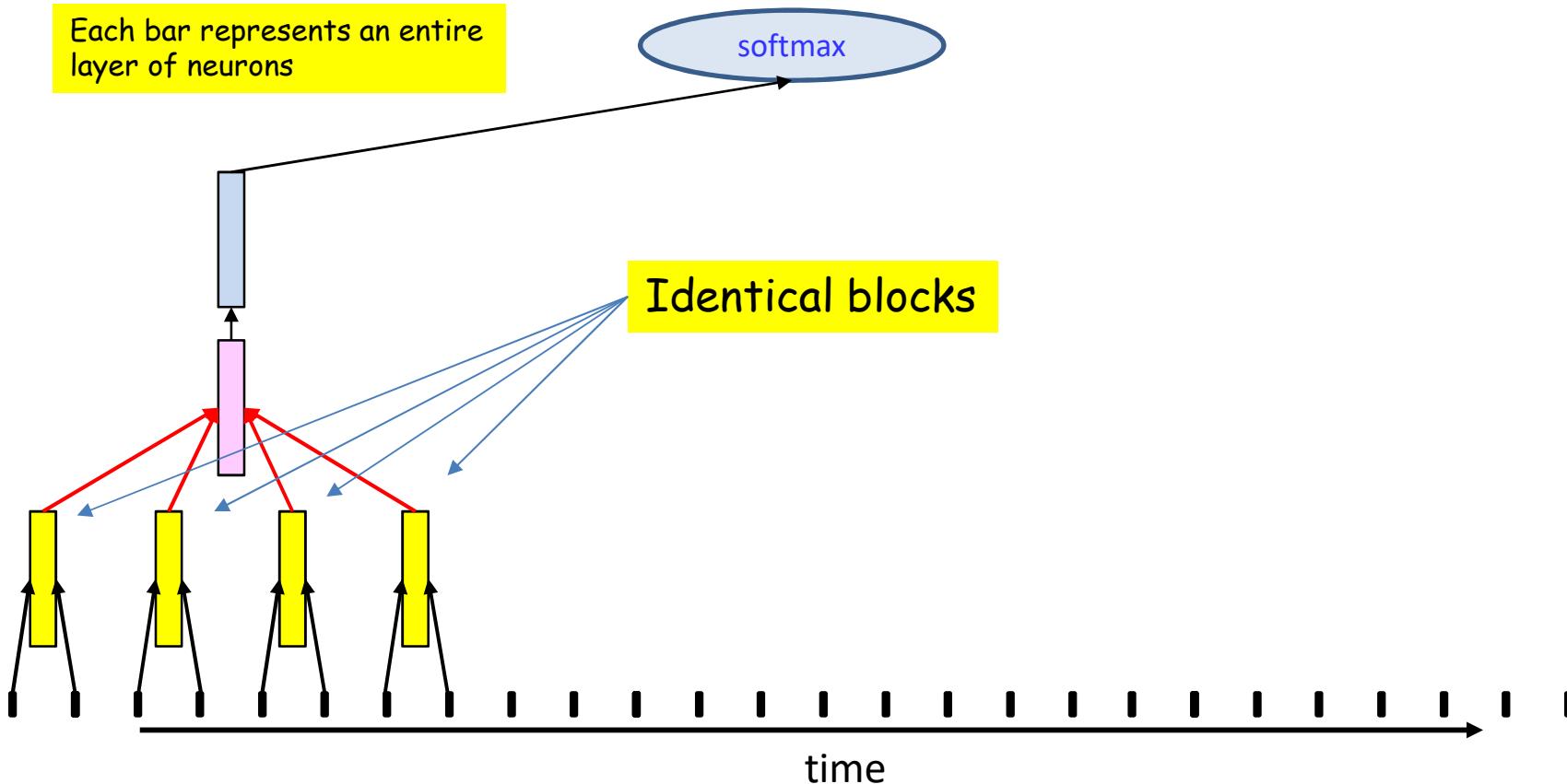
- ***Non-distributed*** scan of 8-time-step wide patterns with a stride of two time steps

# Scanning without distribution



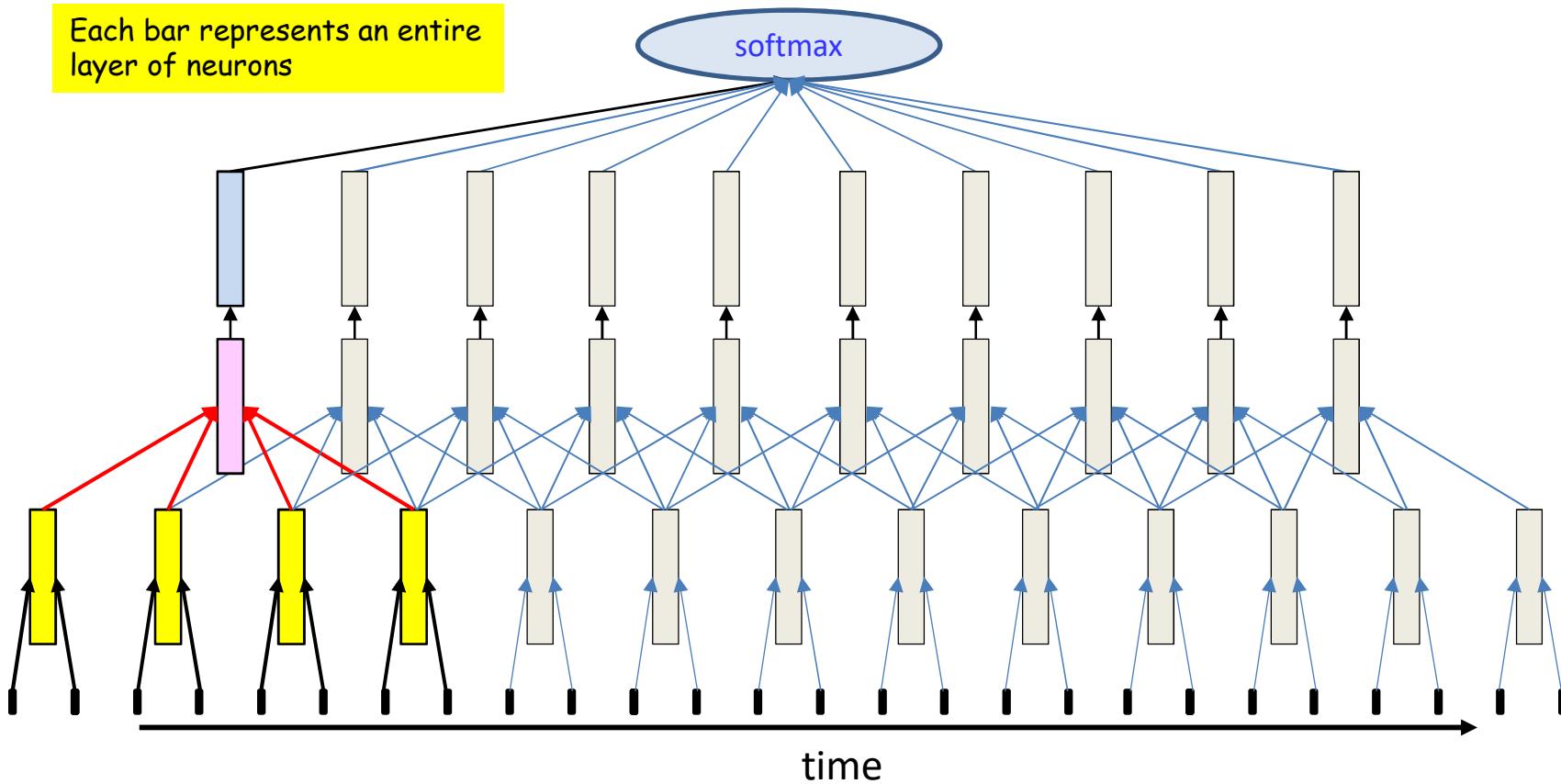
- ***Non-distributed*** scan of 8-time-step wide patterns with a stride of two time steps

# Distributed scanning



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

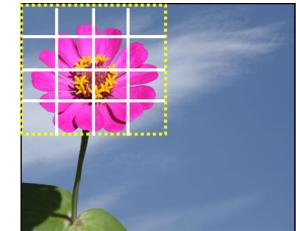
# Distributed scanning



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

# Scanning with an MLP

- $K \times K$  = 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+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

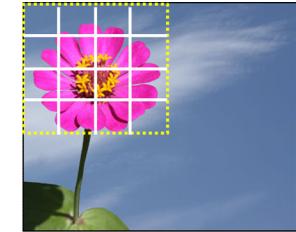
```
for x = 1:W-K+1  
    for y = 1:H-K+1
```

```
        for l = 1:L  # layers  
            for j = 1:Dl
```

**Compute  $z(l, j, x, y)$  [not expanded]**

$Y(l, j, x, y) = \text{activation}(z(l, j, x, y))$

$Y = \text{softmax}(\ Y(L, :, 1, 1) \dots Y(L, :, W-K+1, H-K+1) \ )$



# Reordering the computation

```
for l = 1:L # layers  
    for j = 1:Dl
```

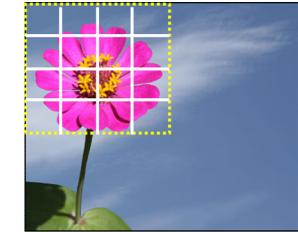
```
        for x = 1:W-K+1
```

```
            for y = 1:H-K+1
```

Compute  $z(l, j, x, y)$  [not expanded]

$Y(l, j, x, y) = \text{activation}(z(l, j, x, y))$

$Y = \text{softmax}(\ Y(L, :, 1, 1) \dots Y(L, :, W-K+1, H-K+1) \ )$



# Reordering the computation

```
for l = 1:L  # layers
    for j = 1:Dl
        for x = 1:Wl-1-Kl+1
            for y = 1:Hl-1-Kl+1
                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) )
```

Each layer's map is now a different size: Maps scan progressively by  $K_l$  in each layer

# Reordering the computation

```
Y(0,:,:,:, :) = Image
for l = 1:L  # layers operate on vector at (x,y)
    for j = 1:Dl
        for x = 1:Wl-1-Kl+1
            for y = 1:Hl-1-Kl+1
                z(l,j,x,y) = 0
                for i = 1:Dl-1
                    for x' = 1:Kl
                        for y' = 1:Kl
                            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) )
```

# Reordering the computation

```
Y(0,:,:,:, :) = Image
for l = 1:T # layers operate on vector at (x,y)
    for j =
        for x = 1:W_{l-1}-K_l+1
            for y = 1:H_{l-1}-K_l+1
                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 Network” (aka scanning with an MLP)

```
Y(0,:,:,:, :) = Image
for l = 1:L  # layers operate on vector at (x,y)
    for j = 1:D_l
        for x = 1:W_{l-1}-K_l+1
            for y = 1:H_{l-1}-K_l+1
                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  $D_{l-1} \times K_l \times K_l$  tensor (assuming square receptive fields)

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

$\mathbf{Y}(0) = \text{Image}$

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

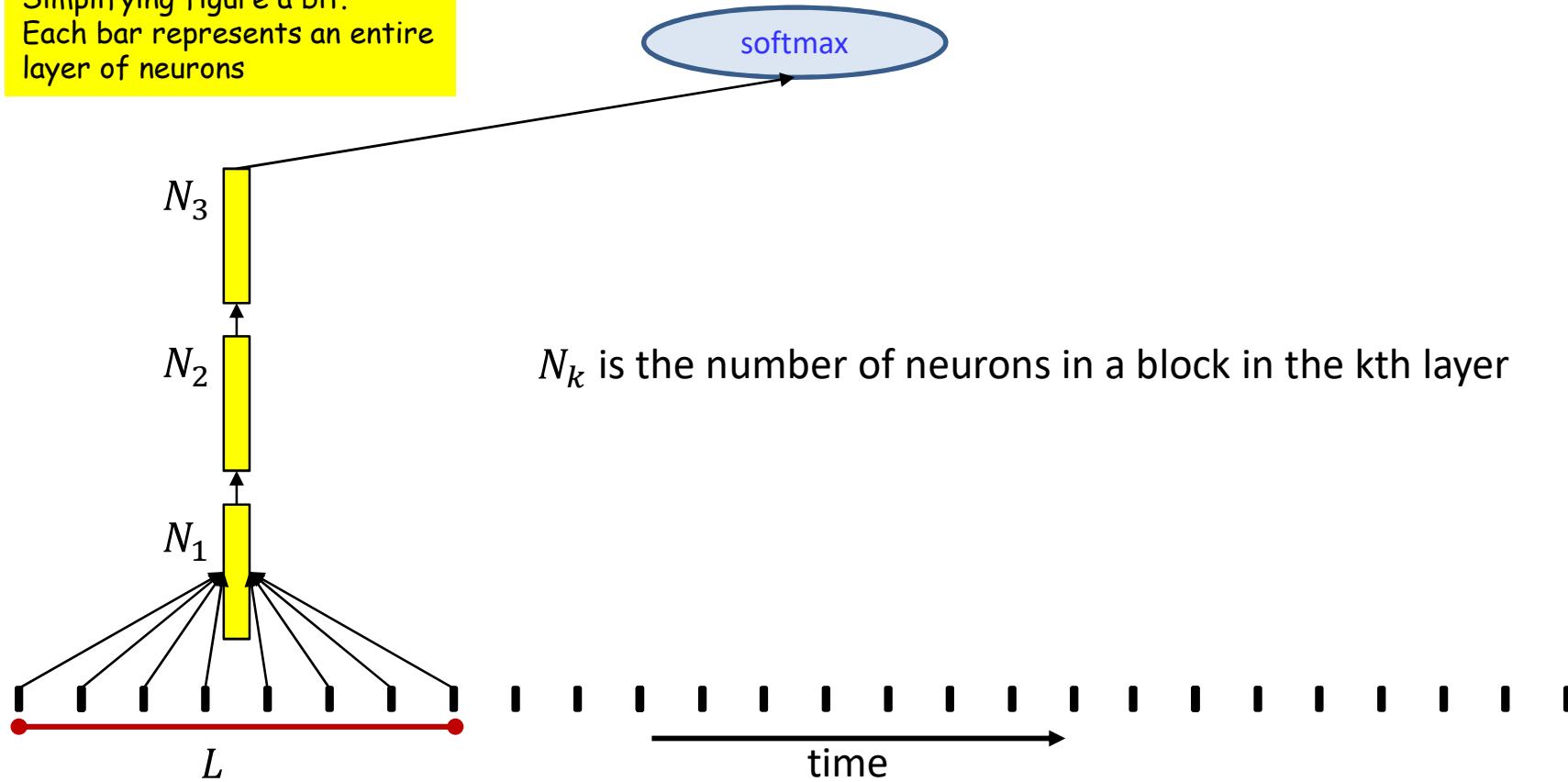
$\mathbf{Y} = \text{softmax}(\mathbf{Y}(L))$

# Why distribute?

- Distribution forces *localized* patterns in lower layers
  - More generalizable
- Number of parameters...

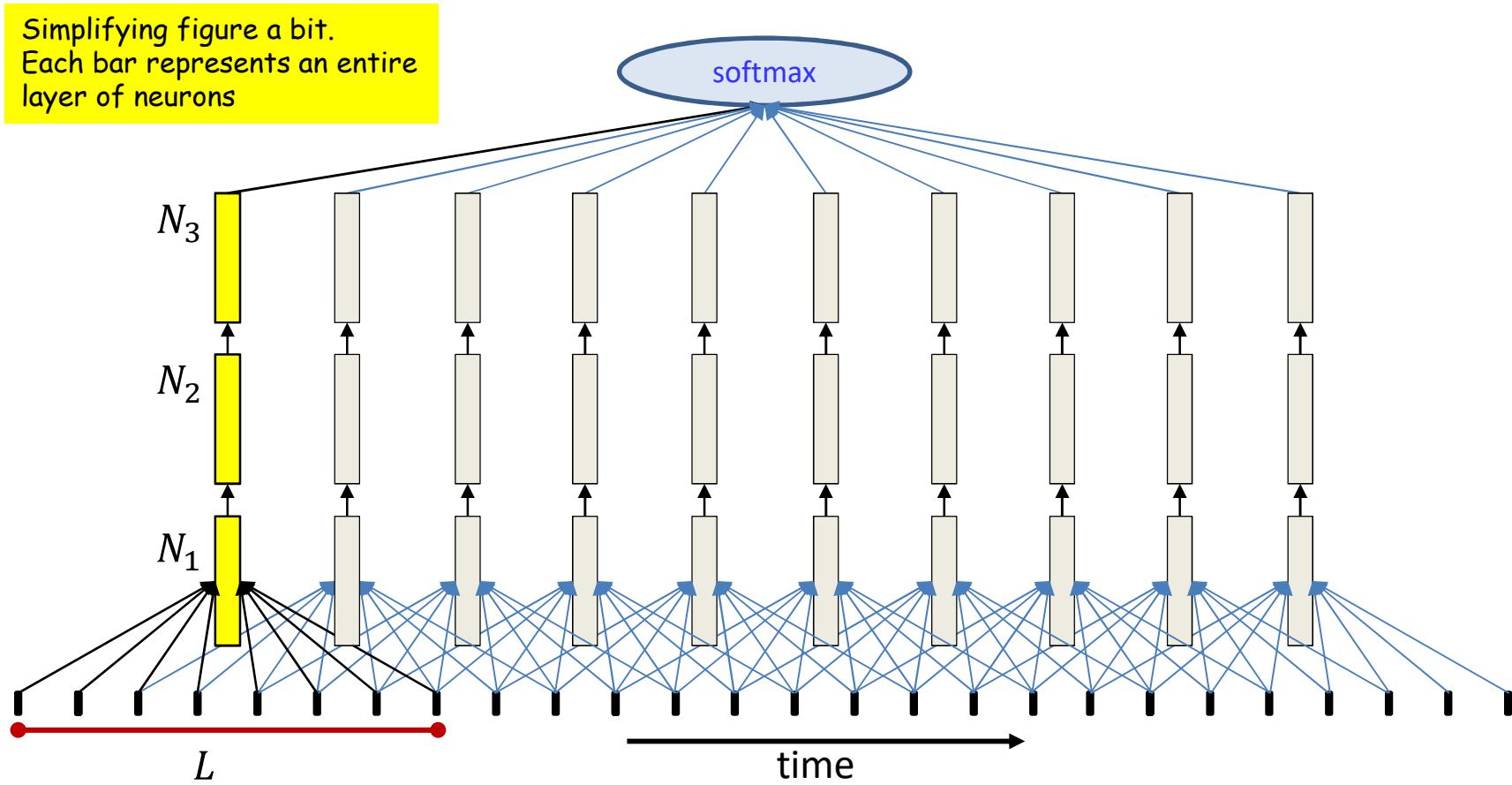
# Scanning without distribution

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



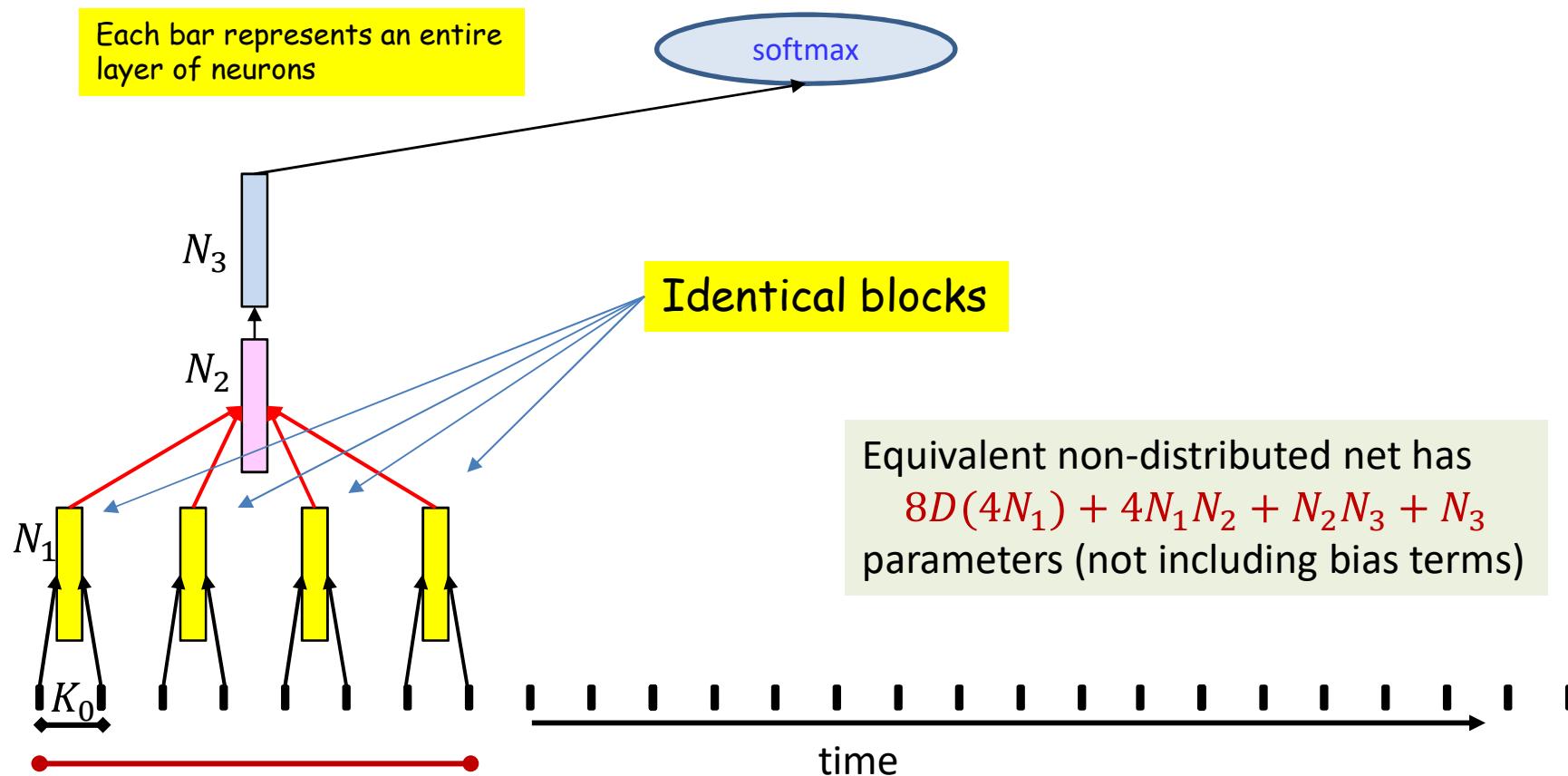
- Total parameters:  $8DN_1 + N_1N_2 + N_2N_3 + N_3$ 
  - $D$  is dimensionality of input
  - More generally:  $LDN_1 + N_1N_2 + N_2N_3 + N_3$
  - Ignoring bias terms in computation
- Only need to count parameters for *one* column, since other columns are identical

# Scanning without distribution



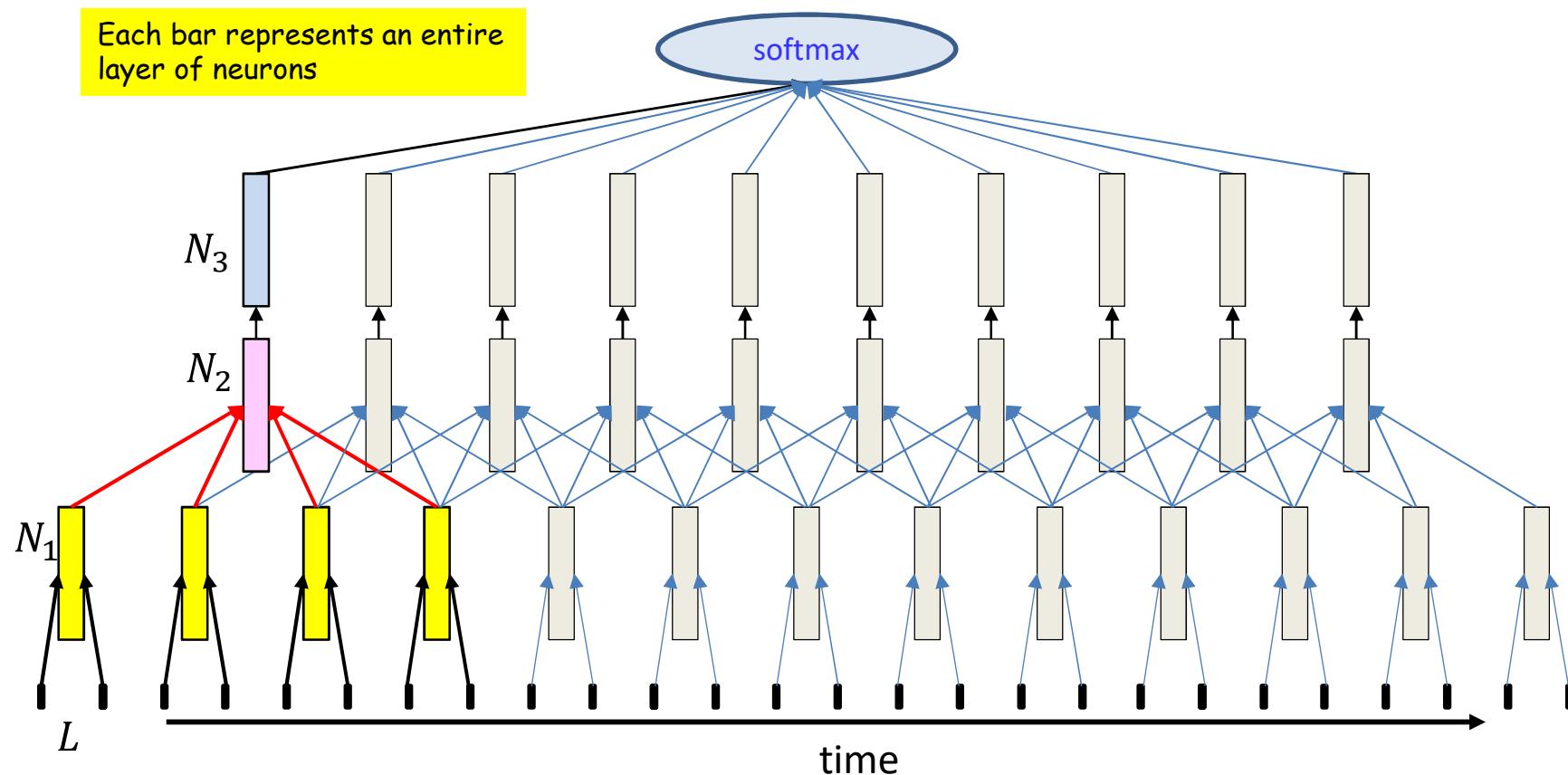
- Total parameters:  $8DN_1 + N_1N_2 + N_2N_3 + N_3$ 
  - $D$  is dimensionality of input
  - More generally:  $LDN_1 + N_1N_2 + N_2N_3 + N_3$
  - Ignoring bias terms in computation
- Only need to count parameters for *one* column, since other columns are identical

# Distributed scanning



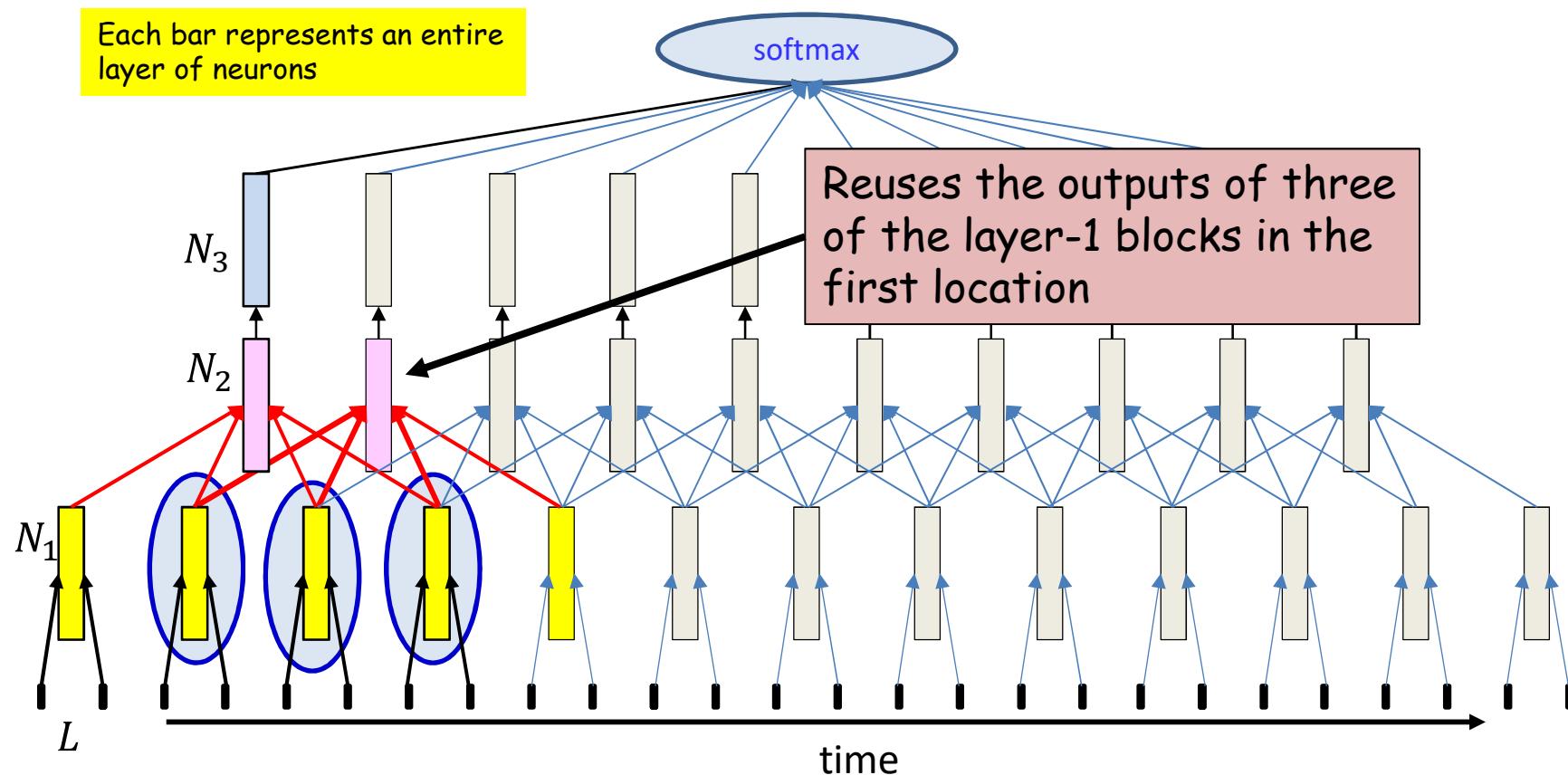
- Total parameters:  $2DN_1 + 4N_1N_2 + N_2N_3 + N_3$ 
  - More generally:  $K_0DN_1 + K_1N_1N_2 + N_2N_3 + N_3$
  - **Fewer parameters than a non-distributed net with identical number of neurons**

# Distributed scanning



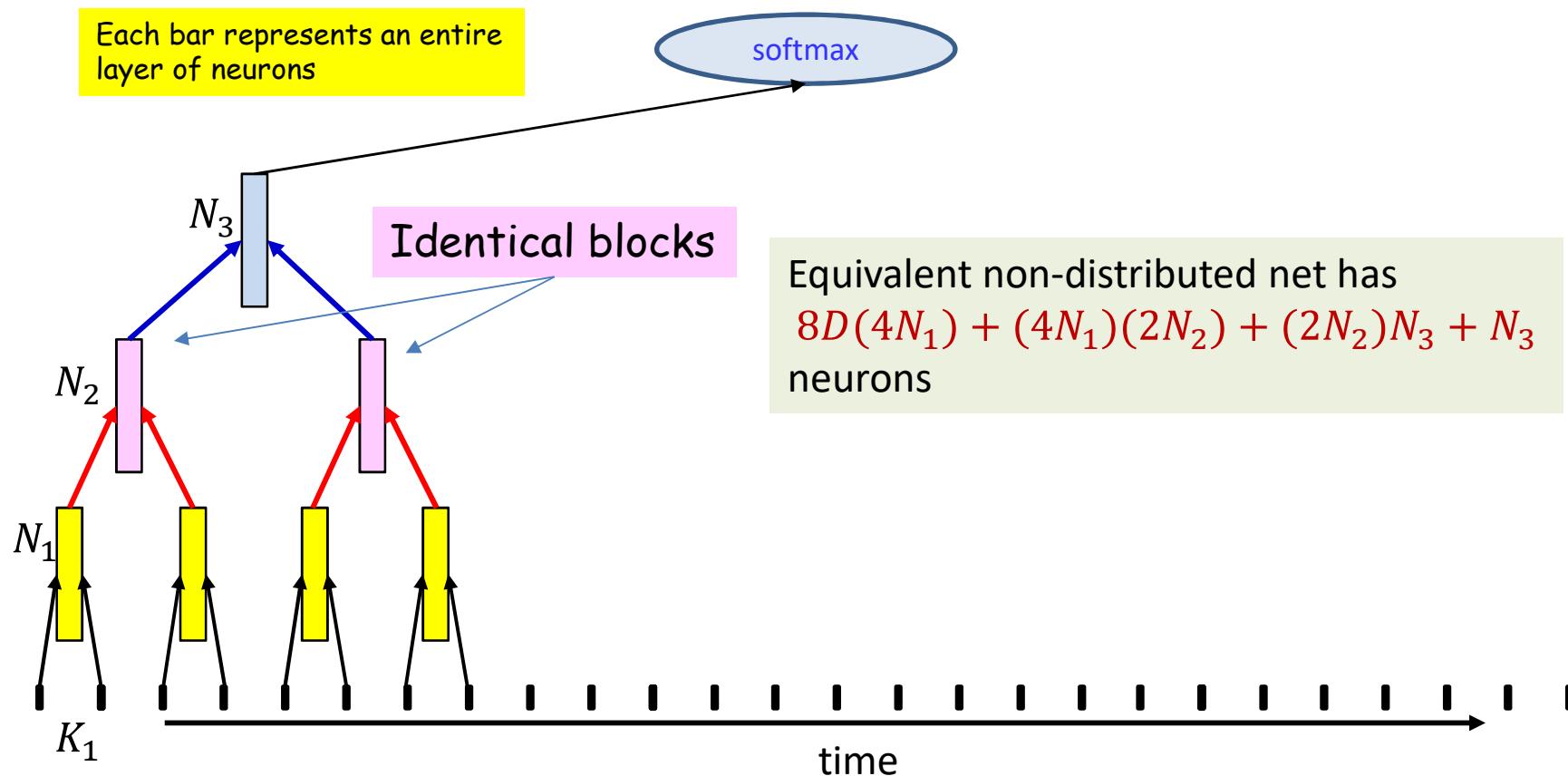
- Total parameters:  $2DN_1 + 4N_1N_2 + N_2N_3 + N_3$ 
  - More generally:  $K_0DN_1 + K_1N_1N_2 + N_2N_3 + N_3$
  - Fewer parameters than a non-distributed net with identical number of neurons

# Distributed scanning



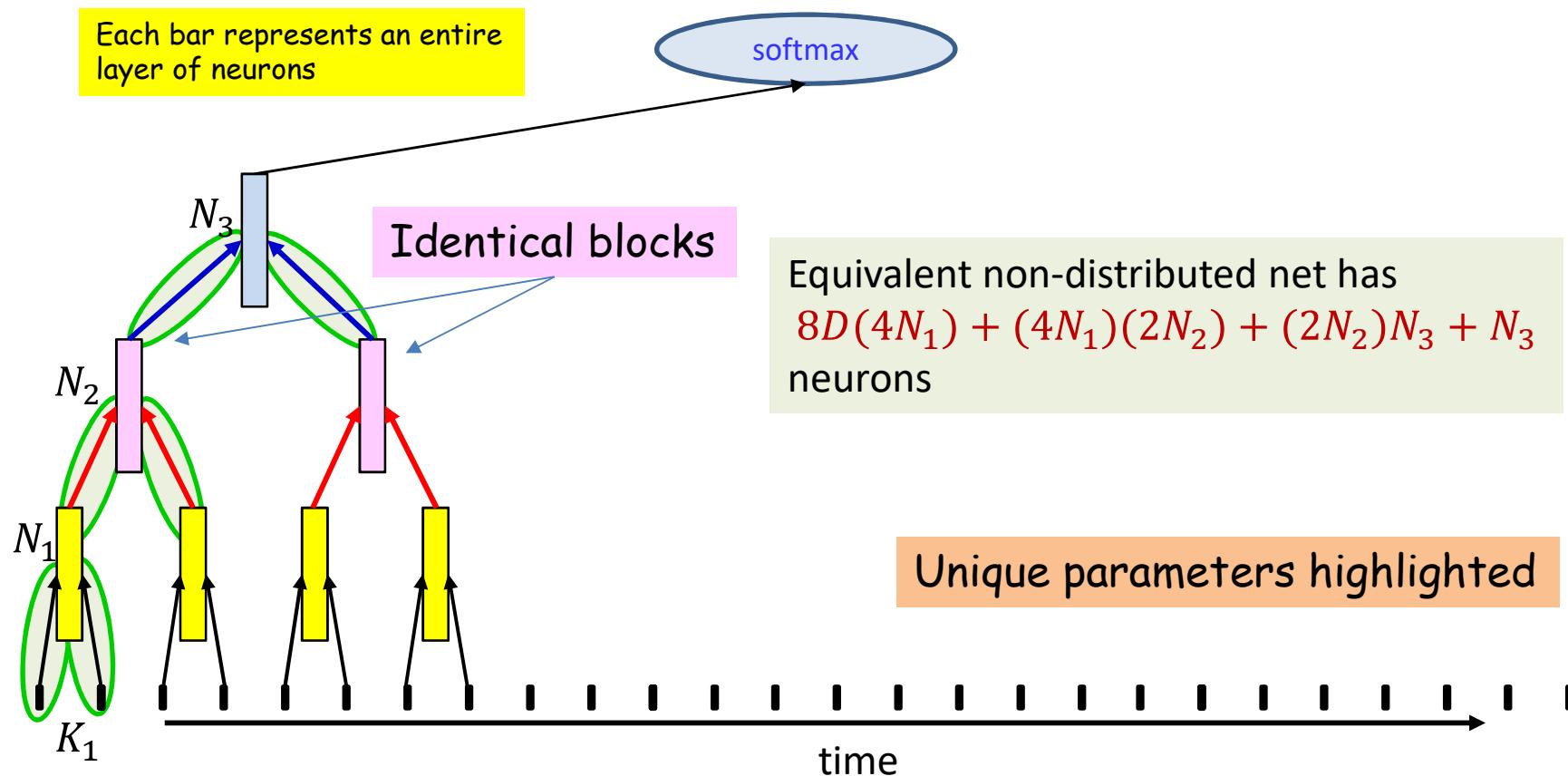
- Total parameters:  $2DN_1 + 4N_1N_2 + N_2N_3 + N_3$ 
  - More generally:  $K_0DN_1 + K_1N_1N_2 + N_2N_3 + N_3$
  - Fewer parameters than a non-distributed net with identical number of neurons
  - *Large additional benefit from the fact that scans at neighboring positions share the computation of lower-level blocks!*

# Distributed scanning: 3 levels



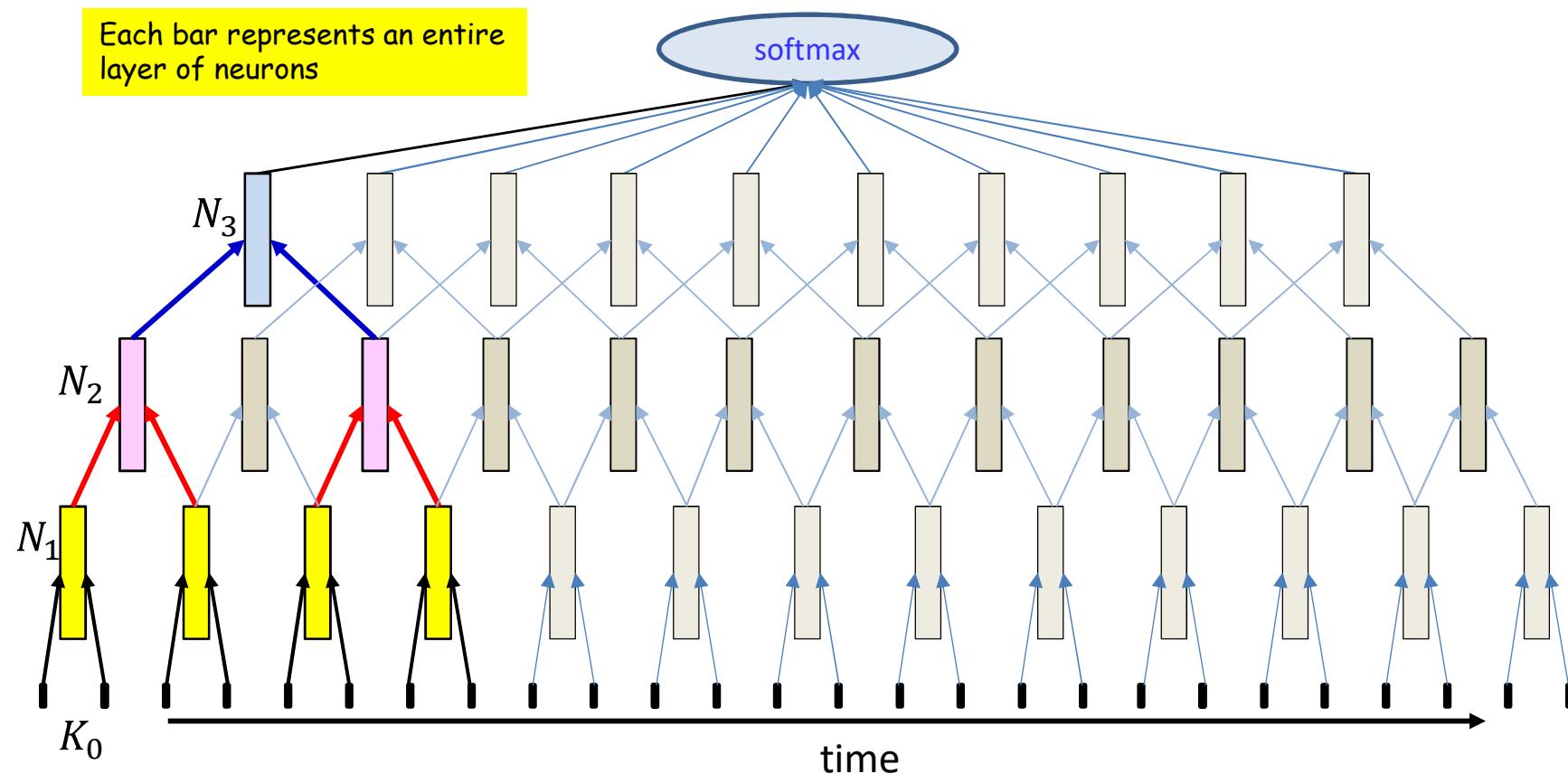
- Total parameters:  $2DN_1 + 2N_1N_2 + 2N_2N_3 + N_3$ 
  - More generally:  $K_0DN_1 + K_1N_1N_2 + K_2N_2N_3 + N_3$
  - **Far fewer parameters than non-distributed scan with network with identical no. of neurons**

# Distributed scanning: 3 levels



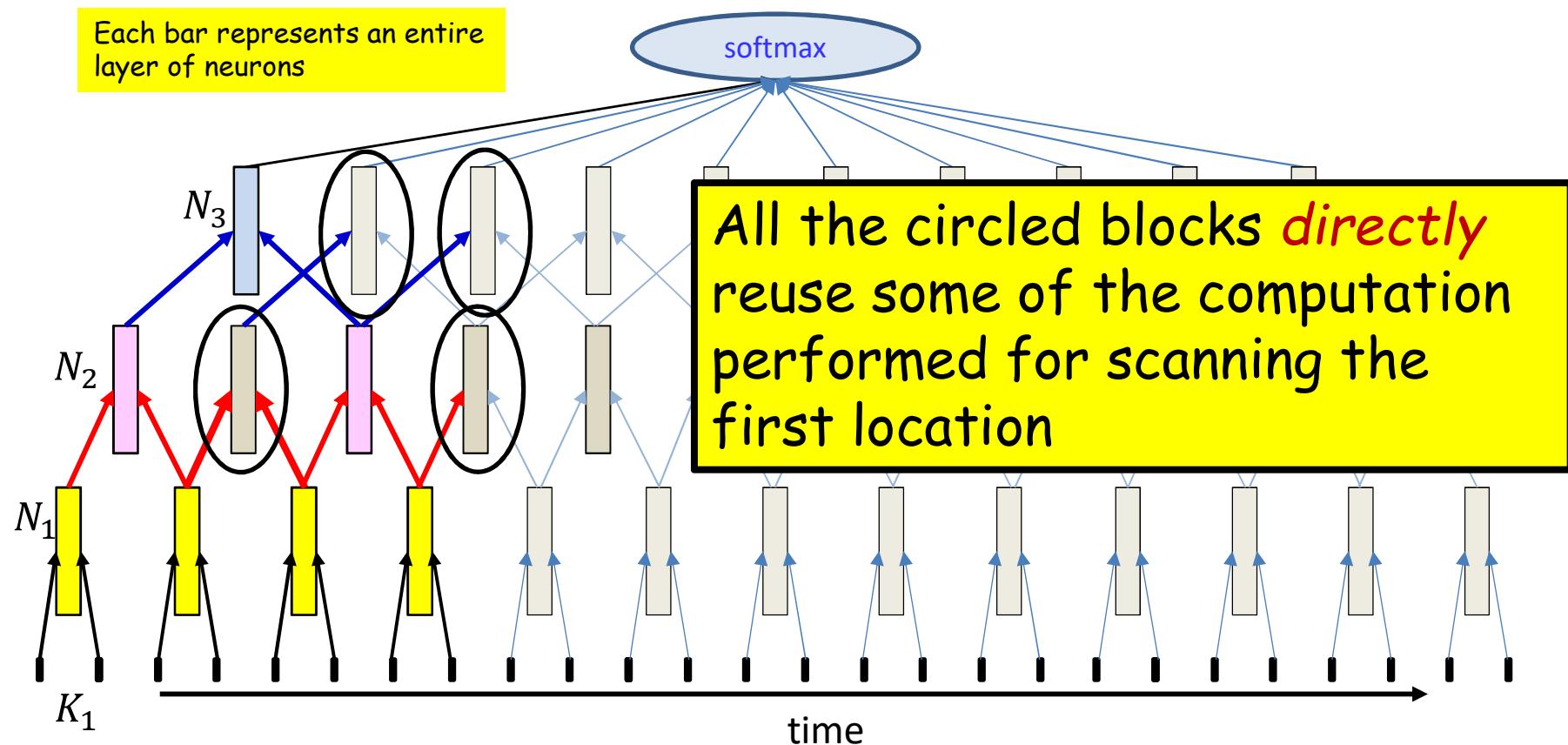
- Total parameters:  $2DN_1 + 2N_1N_2 + 2N_2N_3 + N_3$ 
  - More generally:  $K_0DN_1 + K_1N_1N_2 + K_2N_2N_3 + N_3$
  - Far fewer parameters than non-distributed scan with network with identical no. of neurons

# Distributed scanning: 3 levels



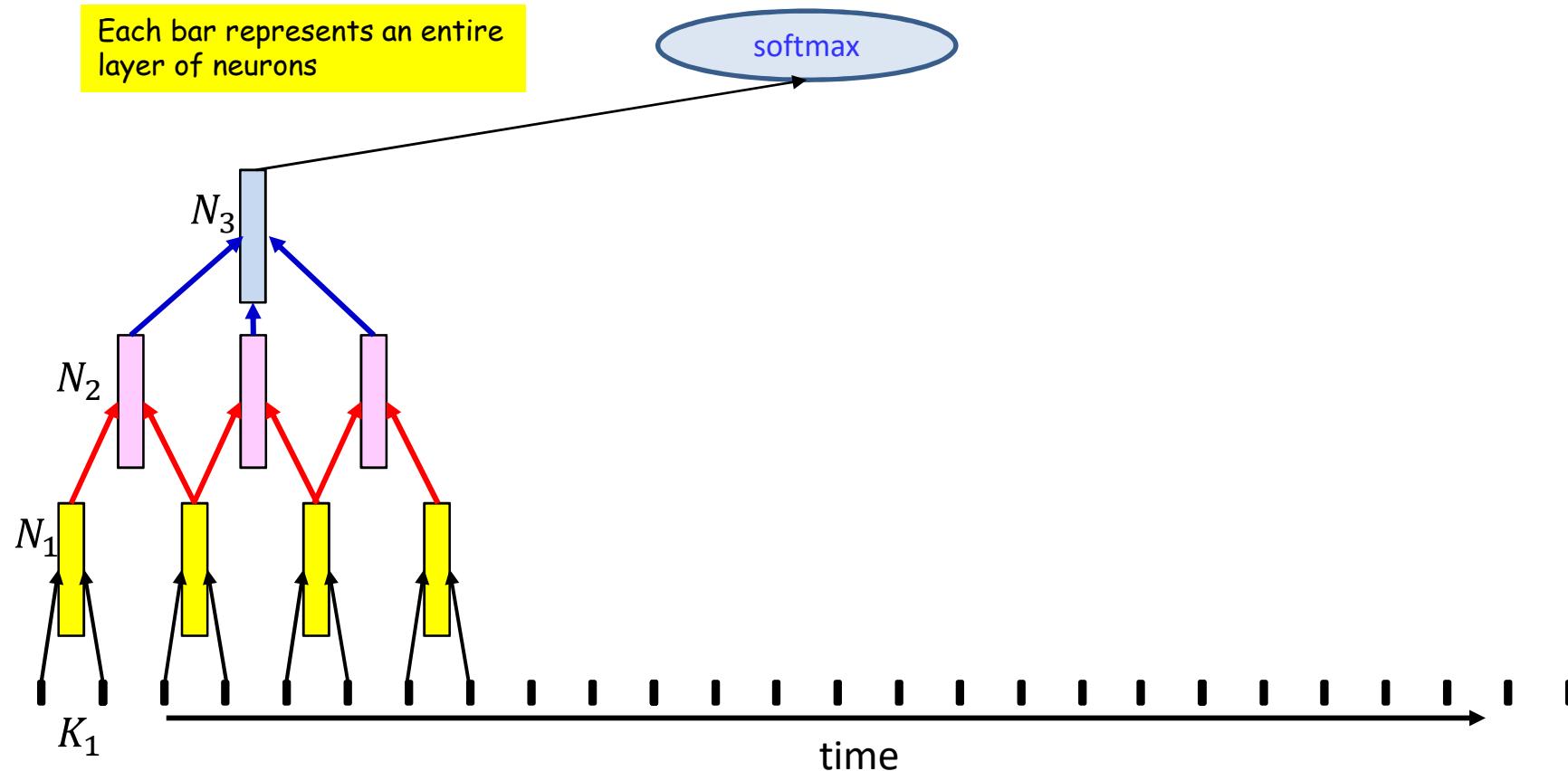
- Total parameters:  $2DN_1 + 2N_1N_2 + 2N_2N_3 + N_3$ 
  - More generally:  $K_0DN_1 + K_1N_1N_2 + K_2N_2N_3 + N_3$
  - **Far fewer parameters than non-distributed scan with network with identical no. of neurons**

# Distributed scanning: 3 levels



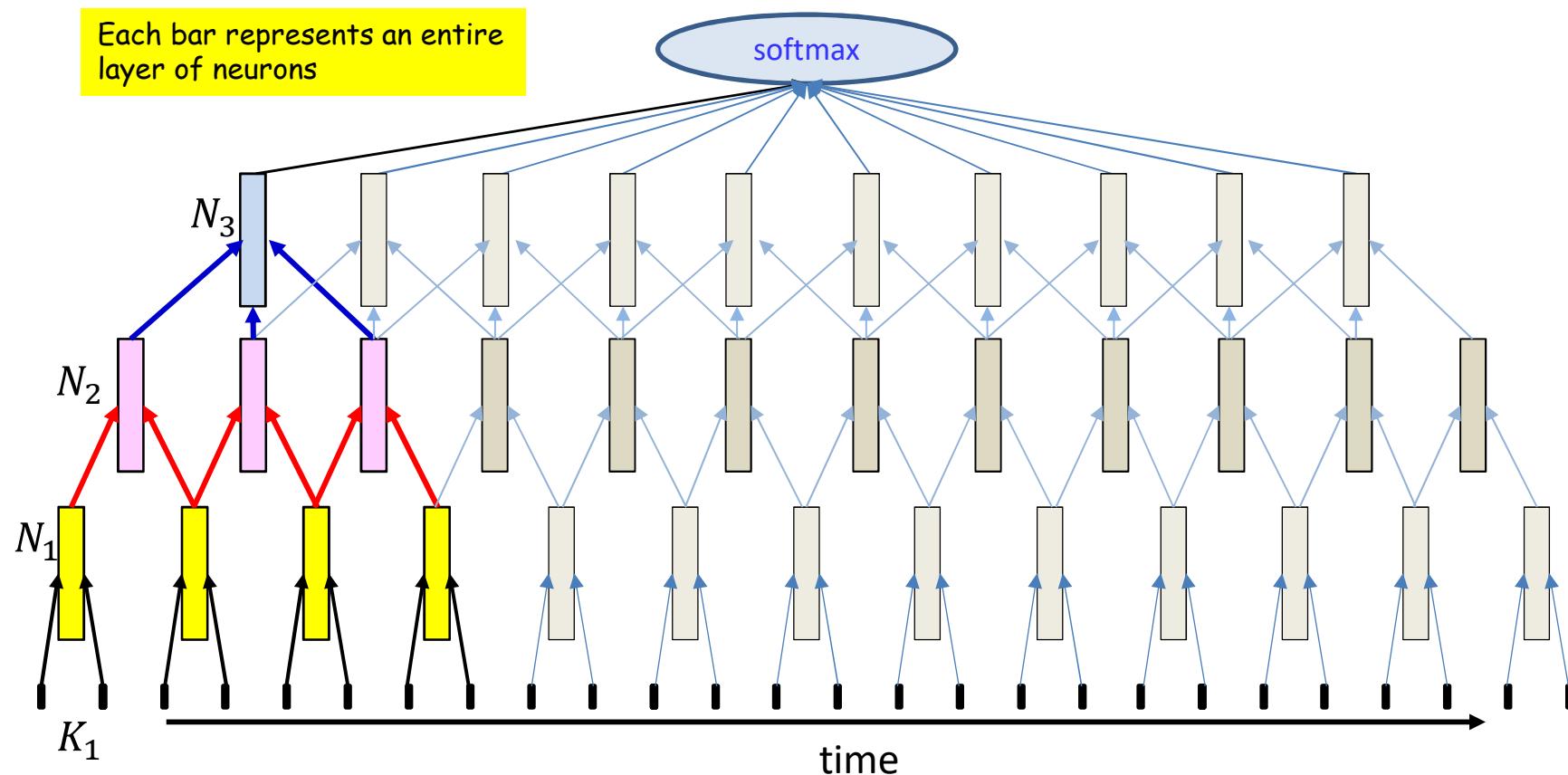
- Total parameters:  $2DN_1 + 2N_1N_2 + 2N_2N_3 + N_3$ 
  - More generally:  $K_1DN_1 + K_2N_1N_2 + (\frac{L}{K_1K_2})N_2N_3 + N_3$
  - Far fewer parameters than non-distributed scan by network with identical no. of neurons
  - Large additional gains from reuse of computation!!

# Distributed scanning



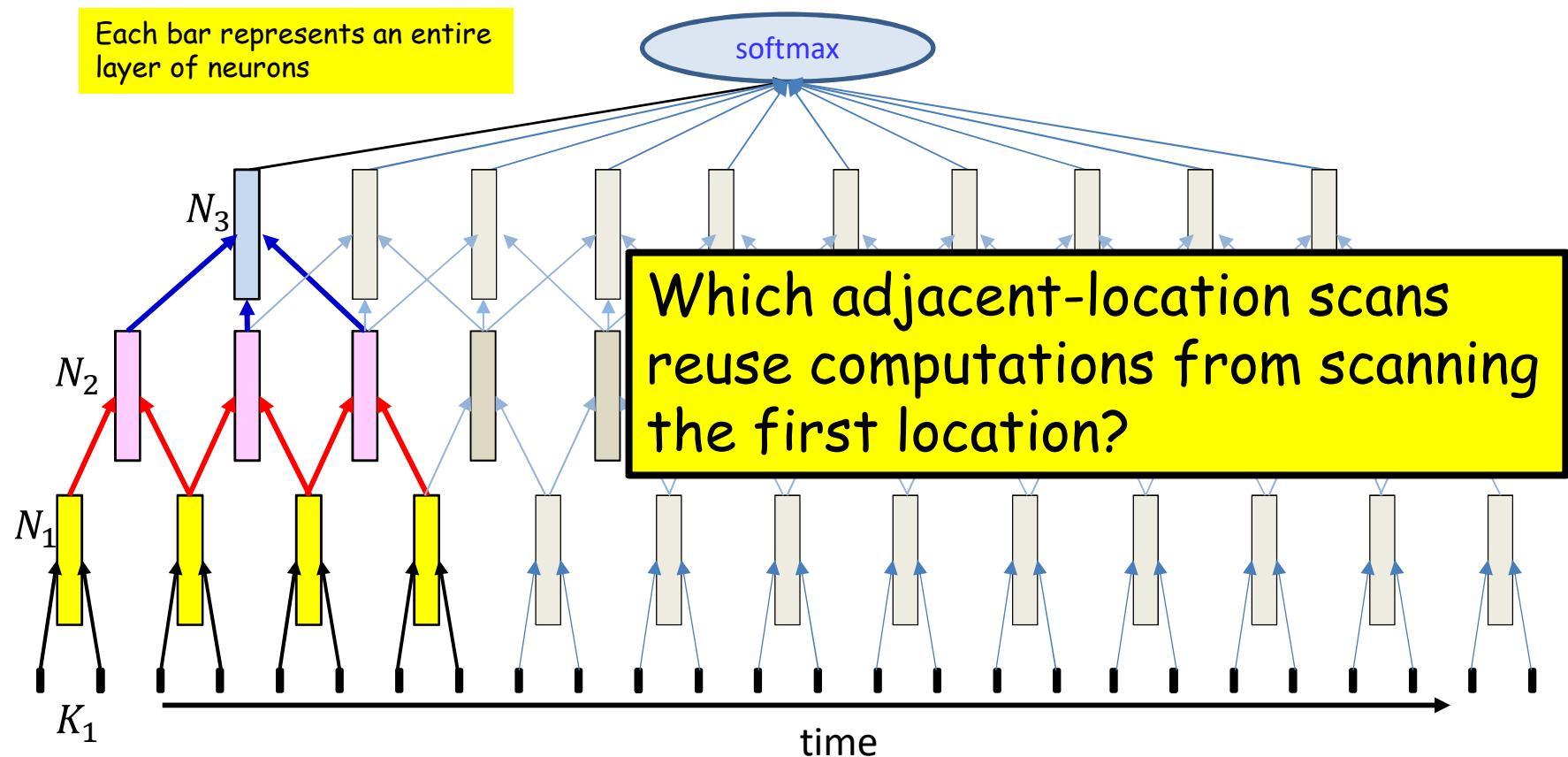
- Total parameters:  $2DN_1 + 2N_1N_2 + 3N_2N_3 + N_3$ 
  - More generally:  $K_1DN_1 + K_2N_1N_2 + K_3N_2N_3 + N_3$
  - **Will have fewer parameters than a non-distributed structure with identical numbers of neurons**

# Distributed scanning



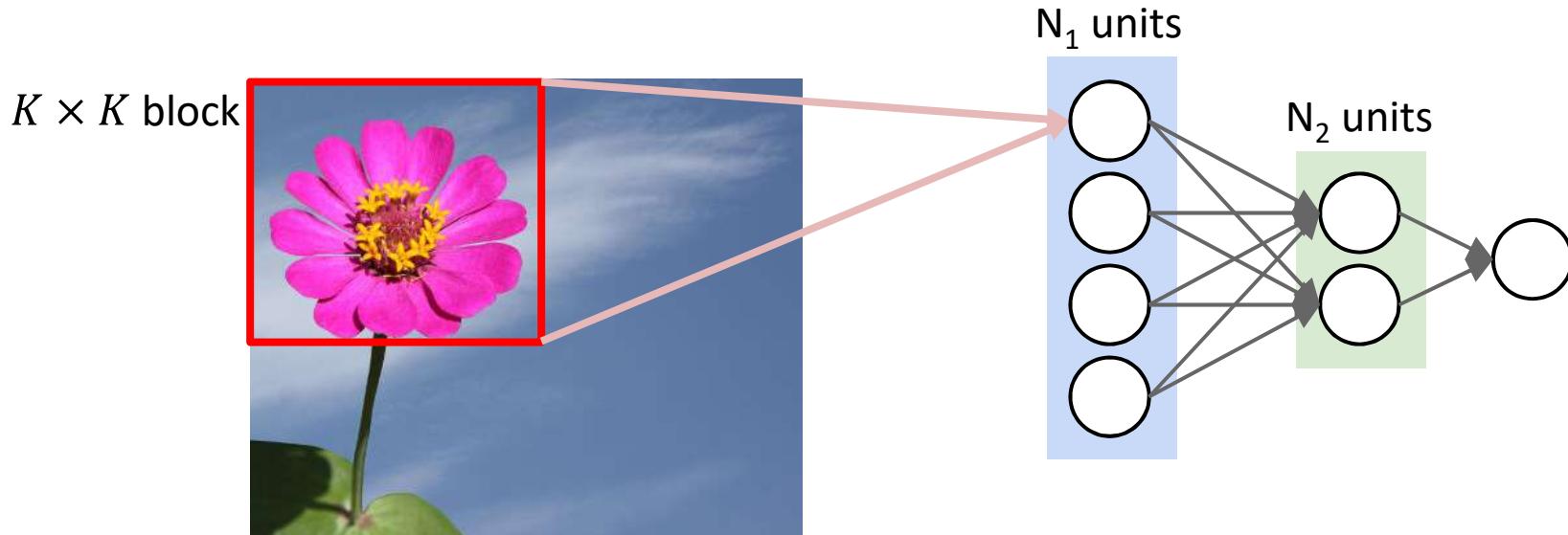
- Total parameters:  $2DN_1 + 2N_1N_2 + 3N_2N_3 + N_3$ 
  - More generally:  $K_1DN_1 + K_2N_1N_2 + K_3N_2N_3 + N_3$
  - **Can end up being more parameters than for non-distributed scanning**

# Distributed scanning



- Total parameters:  $2DN_1 + 2N_1N_2 + 3N_2N_3 + N_3$ 
  - More generally:  $K_1DN_1 + K_2N_1N_2 + K_3N_2N_3 + N_3$
  - **Can end up being more parameters than for non-distributed scanning**
  - **But still benefit much more from shared computation in the scans of adjacent locations**

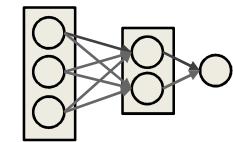
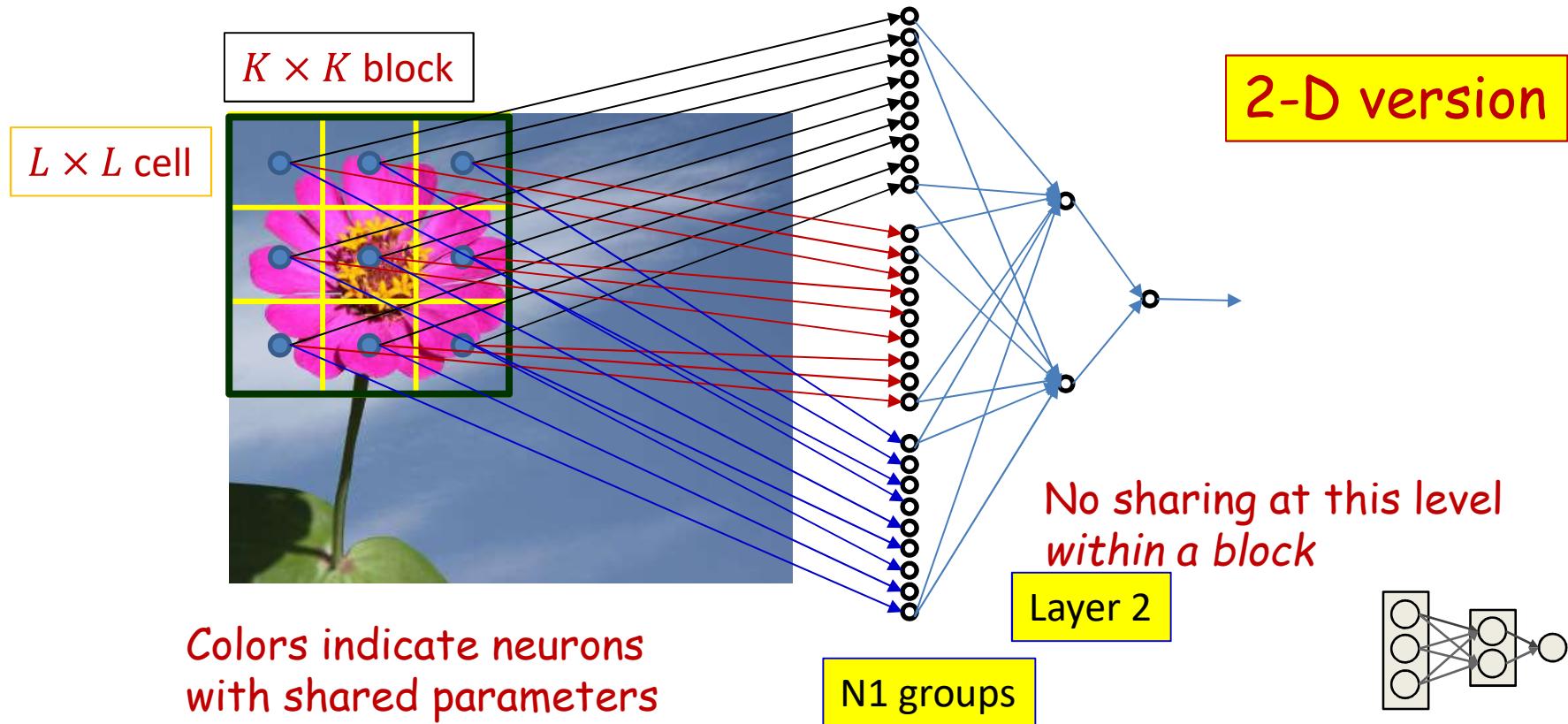
# Parameters in *Undistributed* network



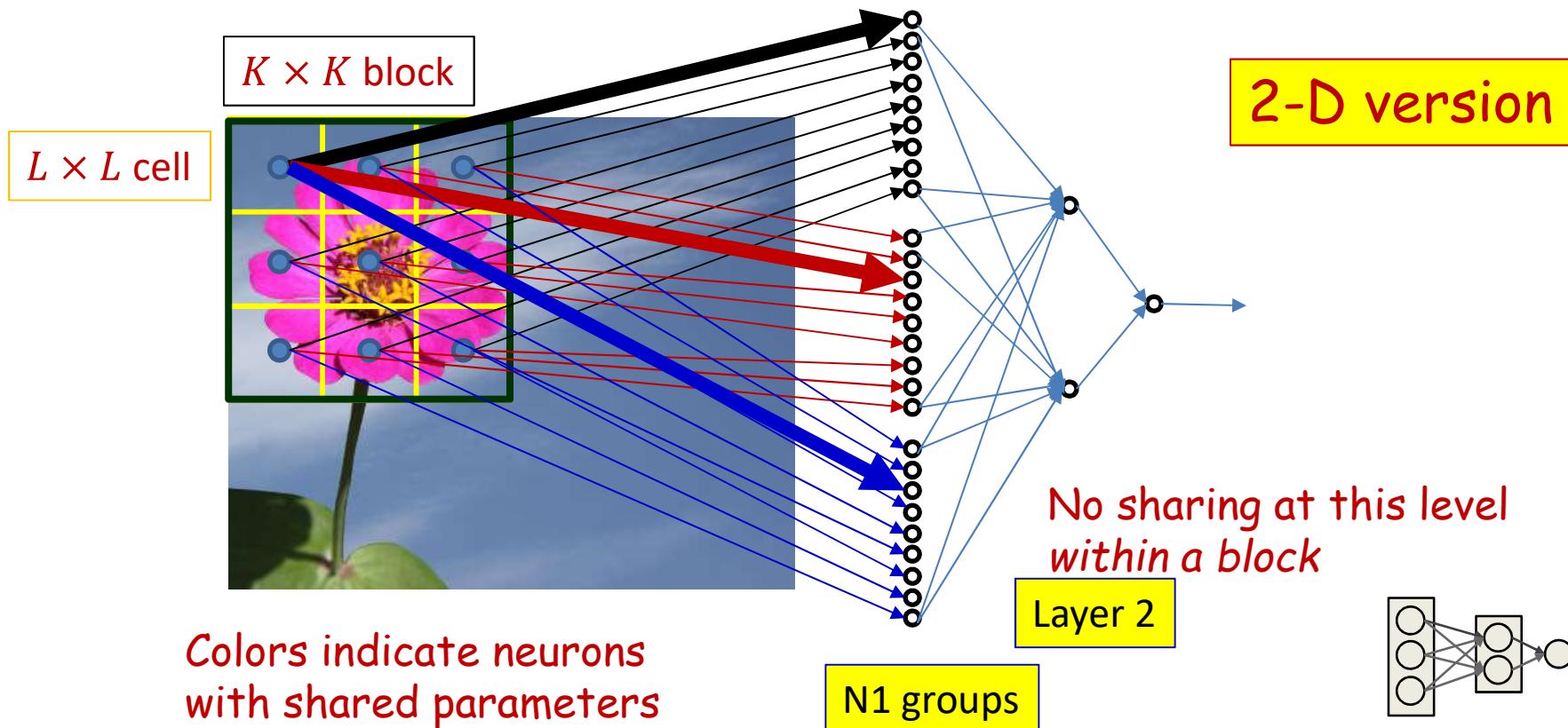
- Only need to consider what happens in *one* block
  - All other blocks are scanned by the same net
- $(K^2 + 1)N_1$  weights in first layer
- $(N_1 + 1)N_2$  weights in second layer
  - $(N_{i-1} + 1)N_i$  weights in subsequent  $i^{\text{th}}$  layer
- Total parameters:  $\mathcal{O}(K^2N_1 + N_1N_2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

2-D version

# When distributed over 2 layers

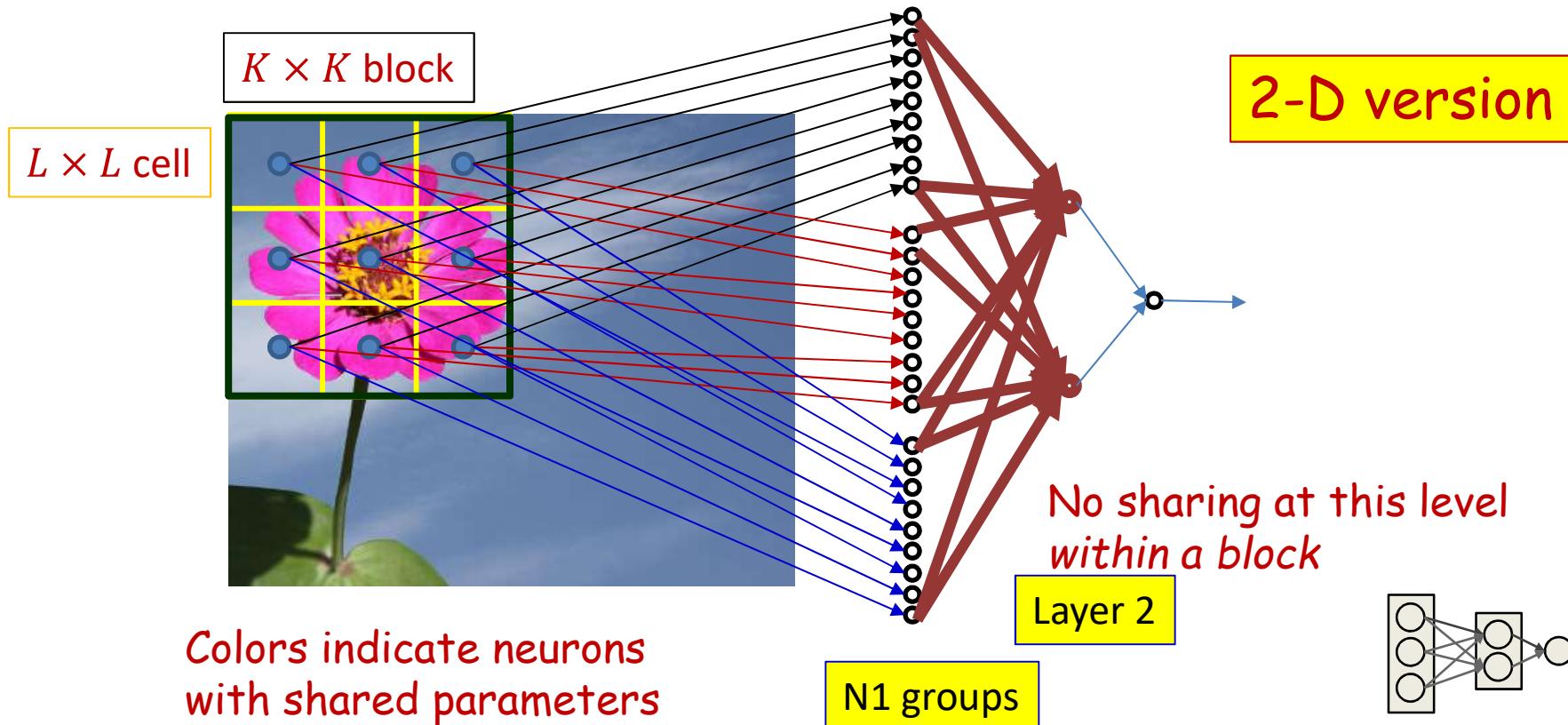


# When distributed over 2 layers



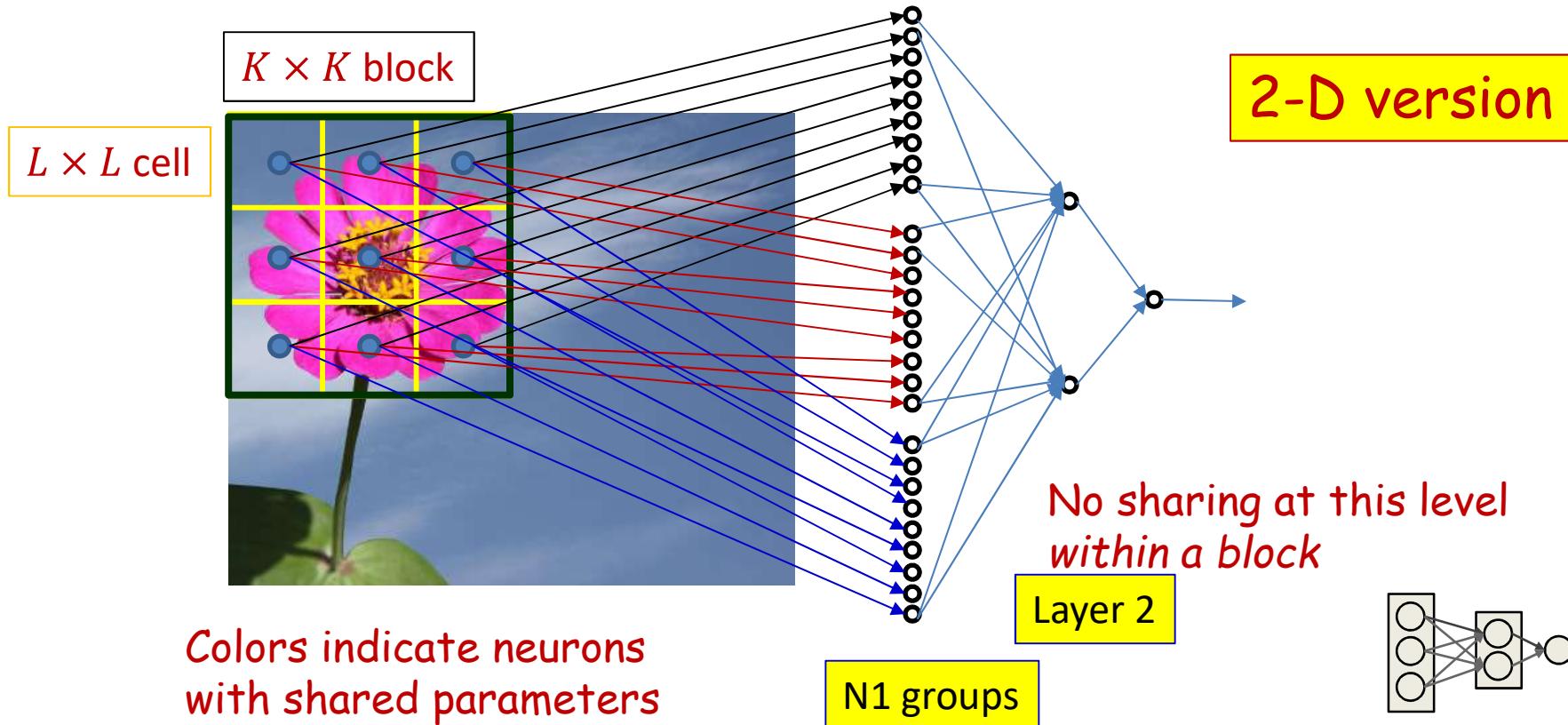
- First layer:  $N_1$  lower-level units, each looks at  $L^2$  pixels
  - $N_1(L^2 + 1)$  weights

# When distributed over 2 layers



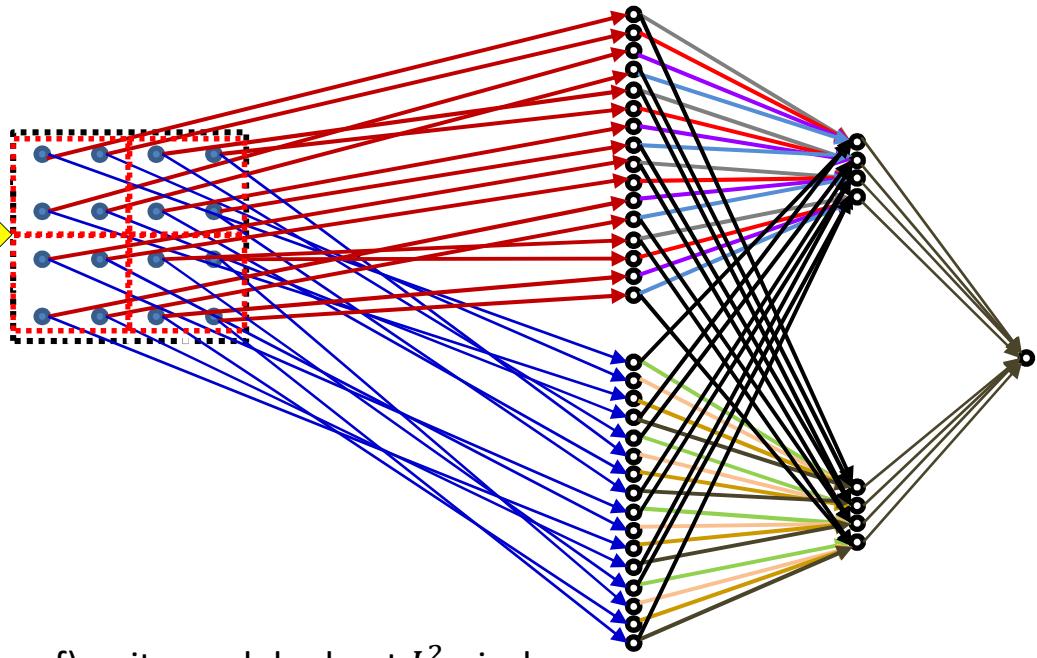
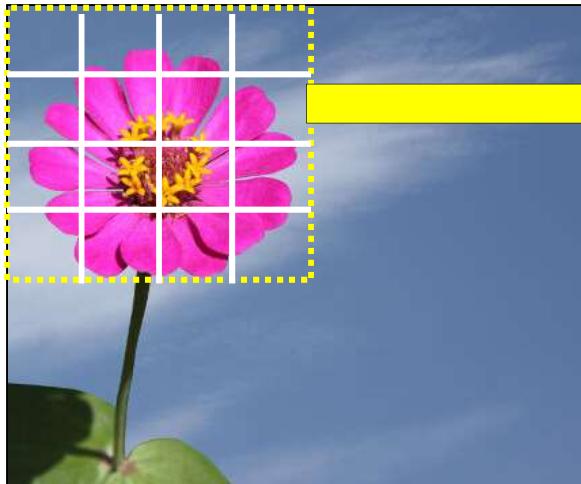
- First layer:  $N_1$  lower-level units, each looks at  $L^2$  pixels
  - $N_1(L^2 + 1)$  weights
- Second layer needs  $(\left(\frac{K}{L}\right)^2 N_1 + 1)N_2$  weights

# When distributed over 2 layers



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

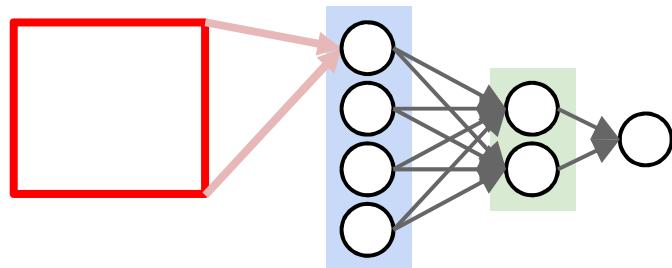
# When distributed over 3 layers



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

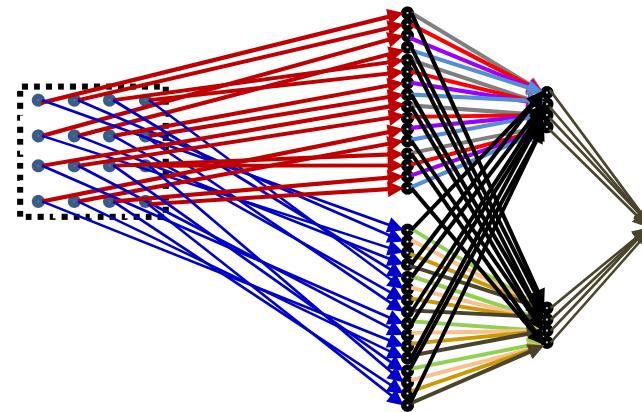
# Comparing Number of Parameters

Conventional MLP, not distributed



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

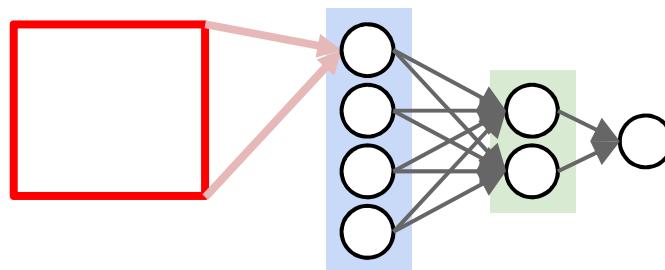
Distributed (3 layers)



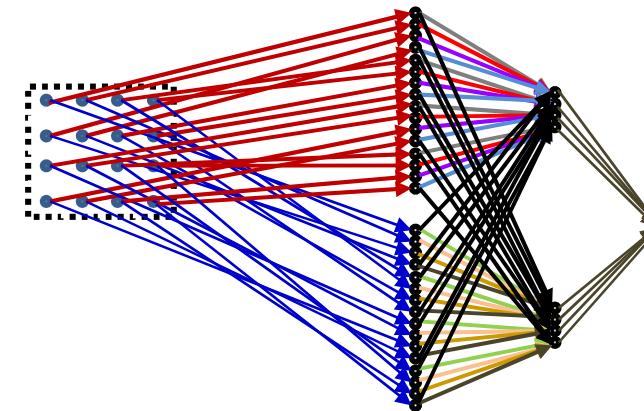
- $\mathcal{O}\left(L_1^2N_1 + L_2^2N_1N_2 + \left(\frac{K}{L_1L_2}\right)^2 N_2N_3 + \dots\right)$
- Here, let  $K = 16, L_1 = 4, L_2 = 4, N_1 = 4, N_2 = 2, N_3 = 1$
- Total  $64+128+8 = 160$  weights

# Comparing Number of Parameters

Conventional MLP, not distributed



Distributed (3 layers)



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

These terms dominate..

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

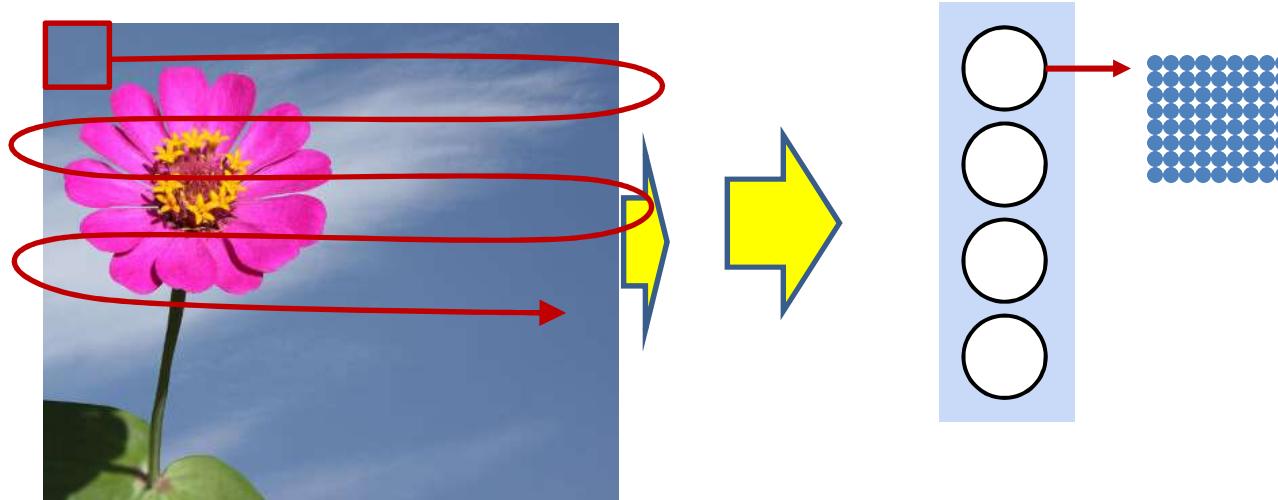
# Why distribute?

- 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

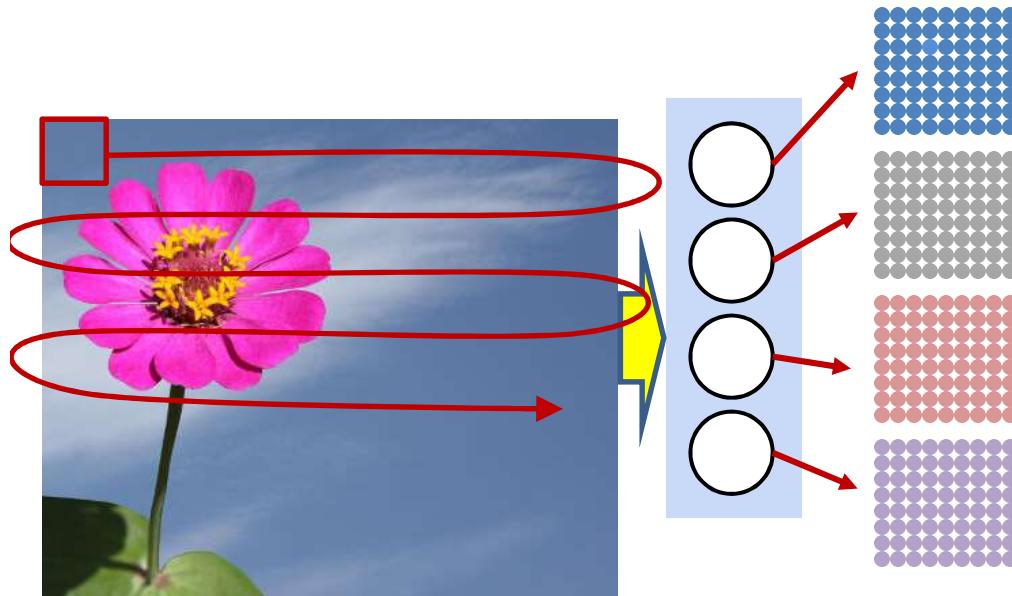
- Position-invariant pattern classification can be performed by scanning the input for a target pattern
  - Scanning is equivalent to composing a large network with shared subnets
- The operations in scanning the input with a full network can be equivalently reordered as
  - scanning the input with individual 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

# Hierarchical composition: A different perspective



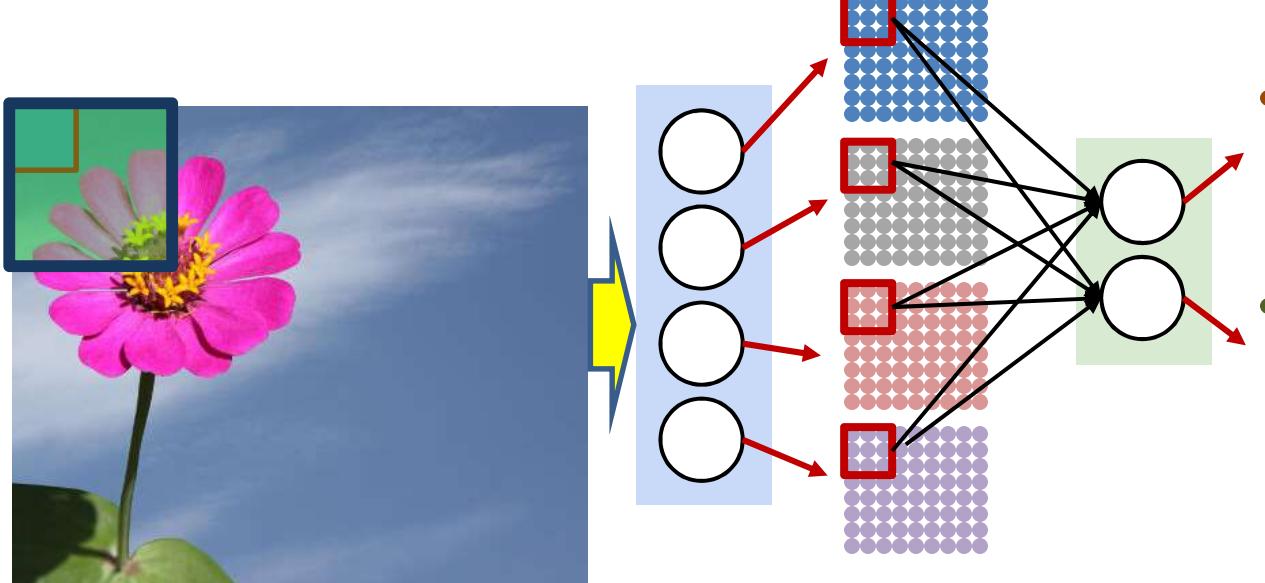
- The entire operation can be redrawn as before as maps of the entire image

# Building up patterns



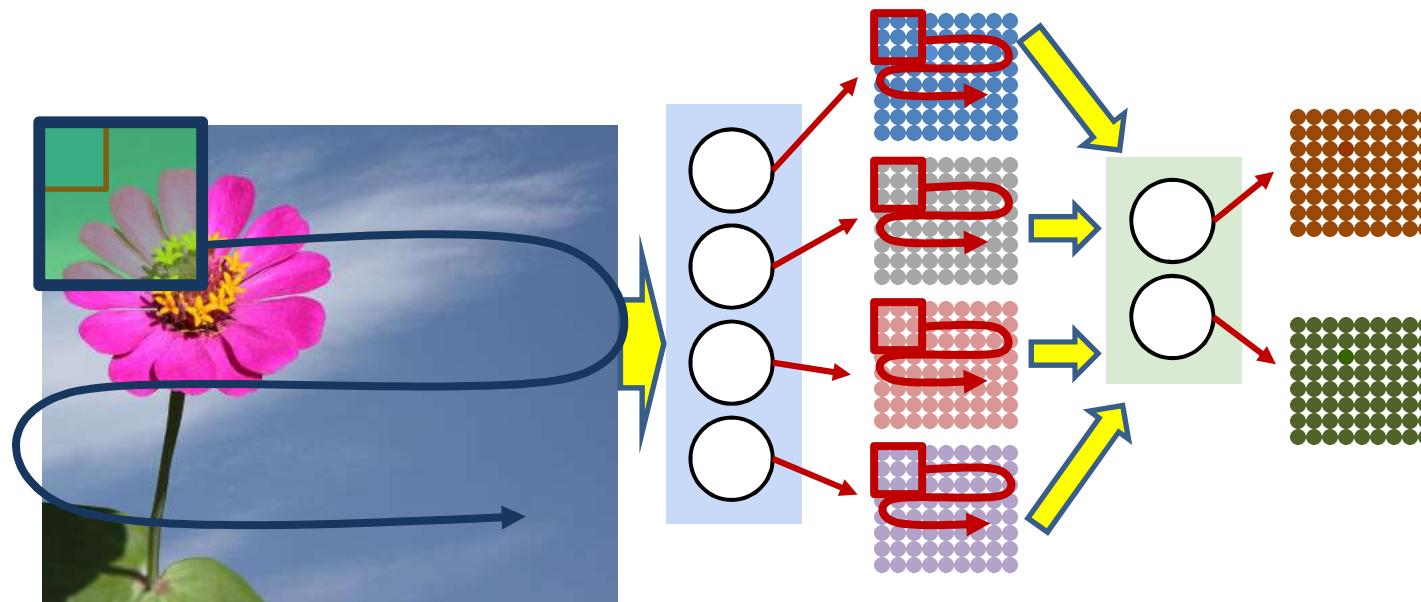
- The first layer looks at small *sub* regions of the main image
  - Sufficient to detect, say, petals

# Some modifications



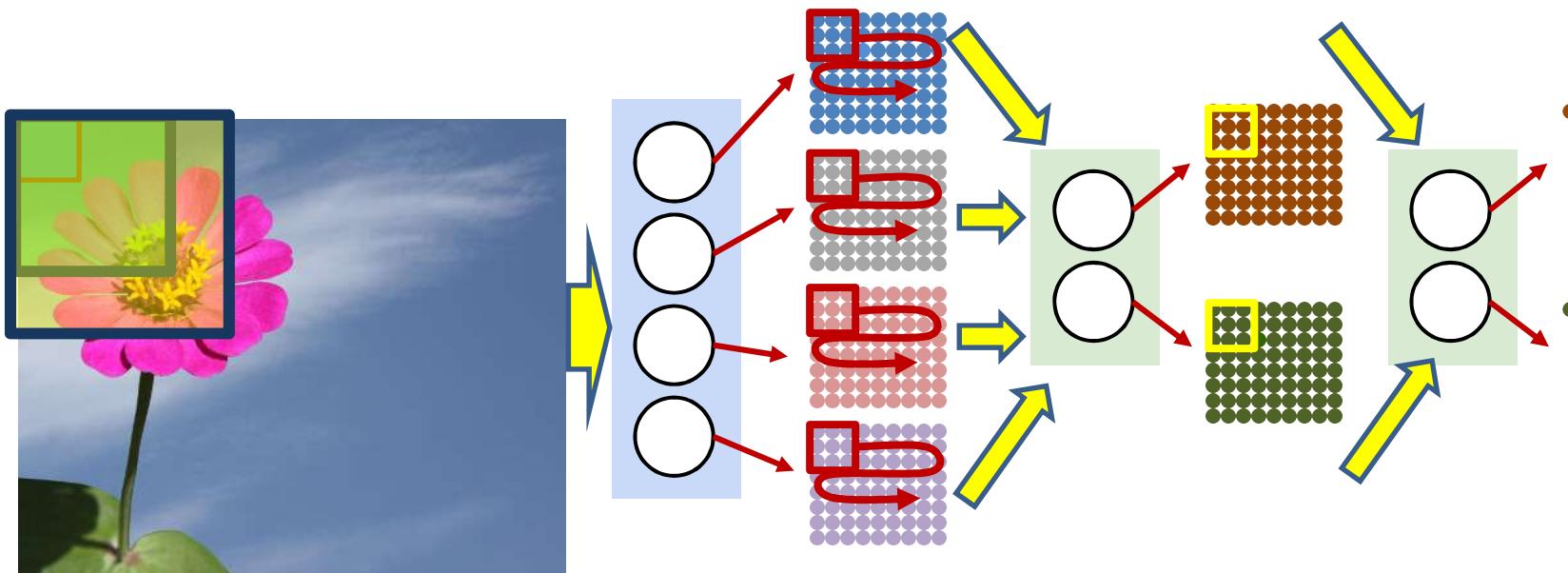
- The first layer looks at *sub* regions of the main image
  - Sufficient to detect, say, petals
- The second layer looks at *regions* of the output of the first layer
  - To put the petals together into a flower
  - This corresponds to looking at a larger region of the original input image

# Some modifications



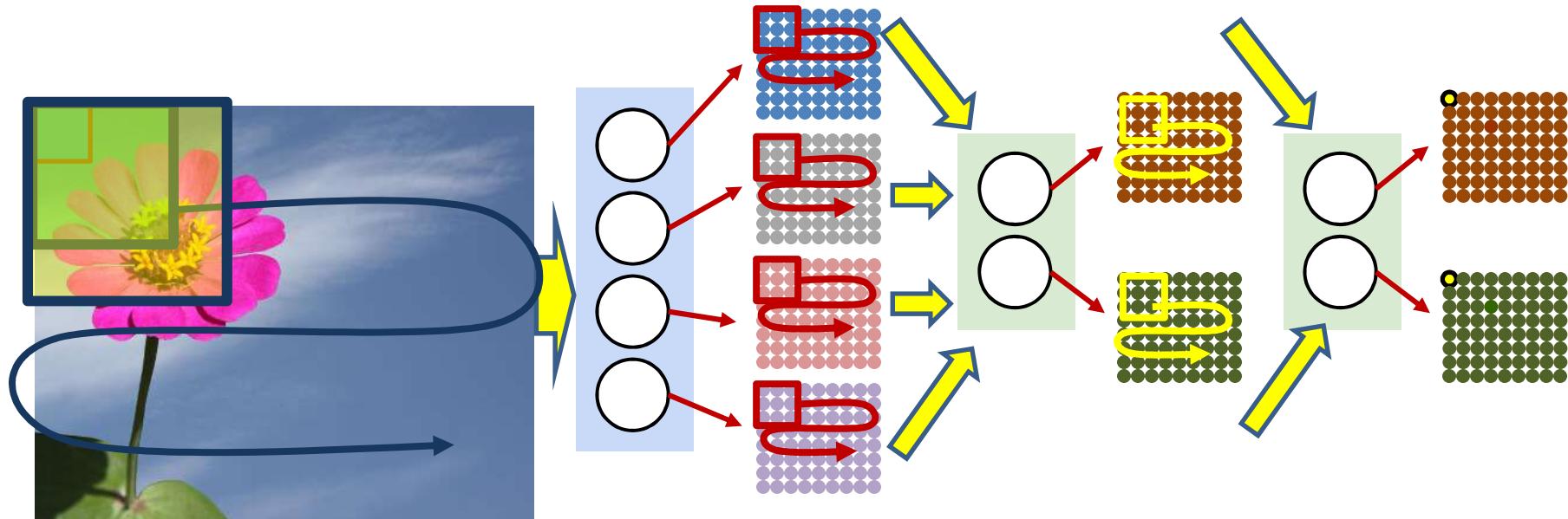
- The first layer looks at *sub* regions of the main image
  - Sufficient to detect, say, petals
- The second layer looks at *regions* of the output of the first layer
  - To put the petals together into a flower
  - This corresponds to looking at a larger region of the original input image

# Some modifications



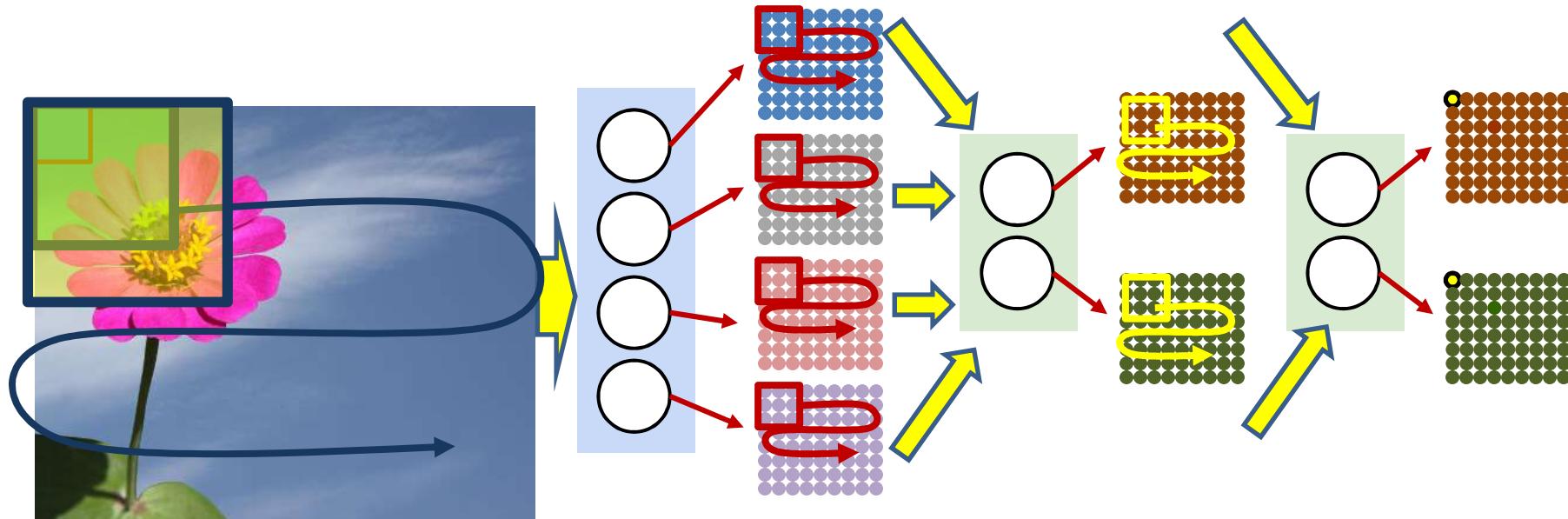
- The first layer looks at *sub* regions of the main image
  - Sufficient to detect, say, petals
- The second layer looks at *regions* of the output of the first layer
  - To put the petals together into a flower
  - This corresponds to looking at a larger region of the original input image
- We may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

# Some modifications



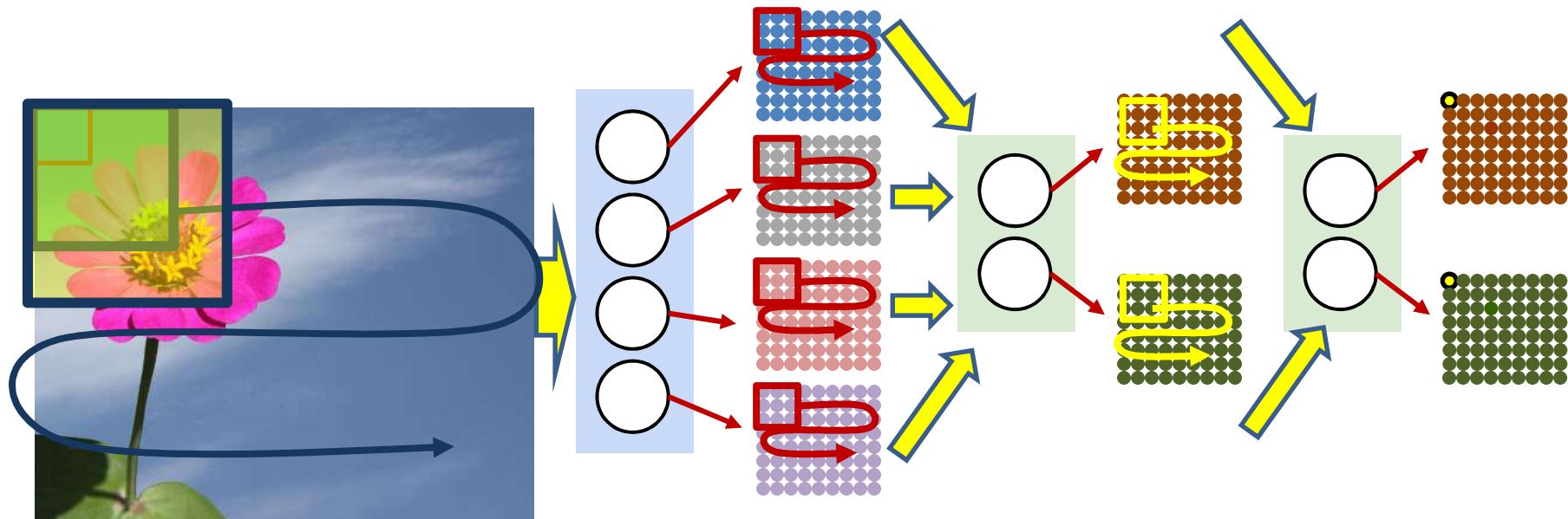
- The first layer looks at *sub* regions of the main image
  - Sufficient to detect, say, petals
- The second layer looks at *regions* of the output of the first layer
  - To put the petals together into a flower
  - This corresponds to looking at a larger region of the original input image
- We may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

# Terminology



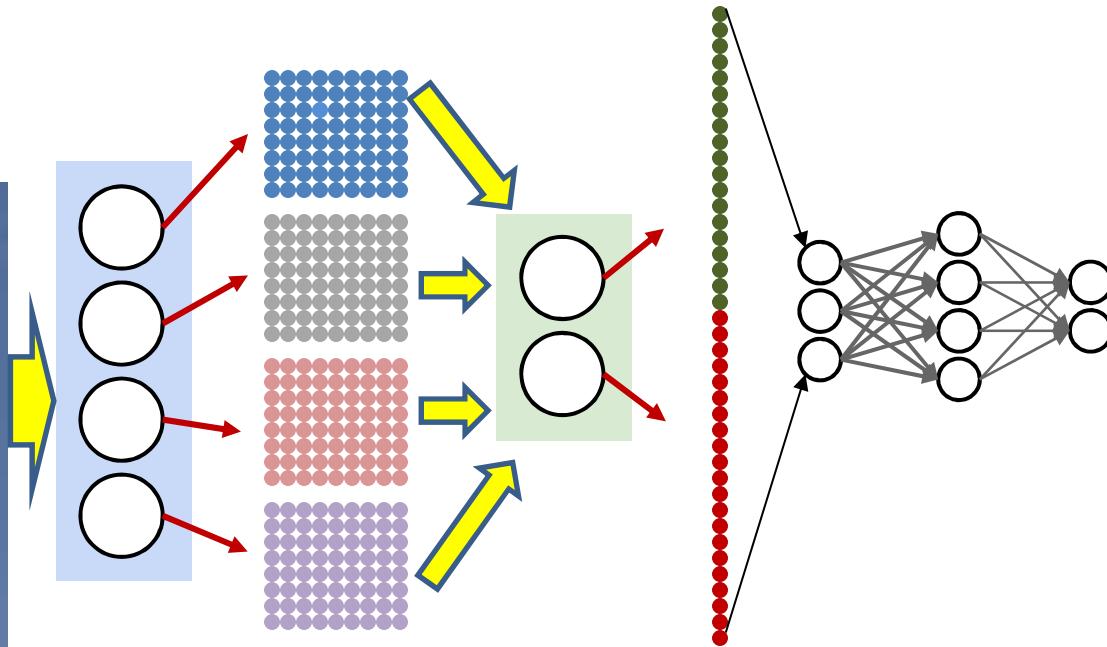
- Each of the scanning neurons is generally called a “filter”
  - Its really a correlation filter as we saw earlier
  - Each filter scans for a pattern in the map it operates on

# Terminology



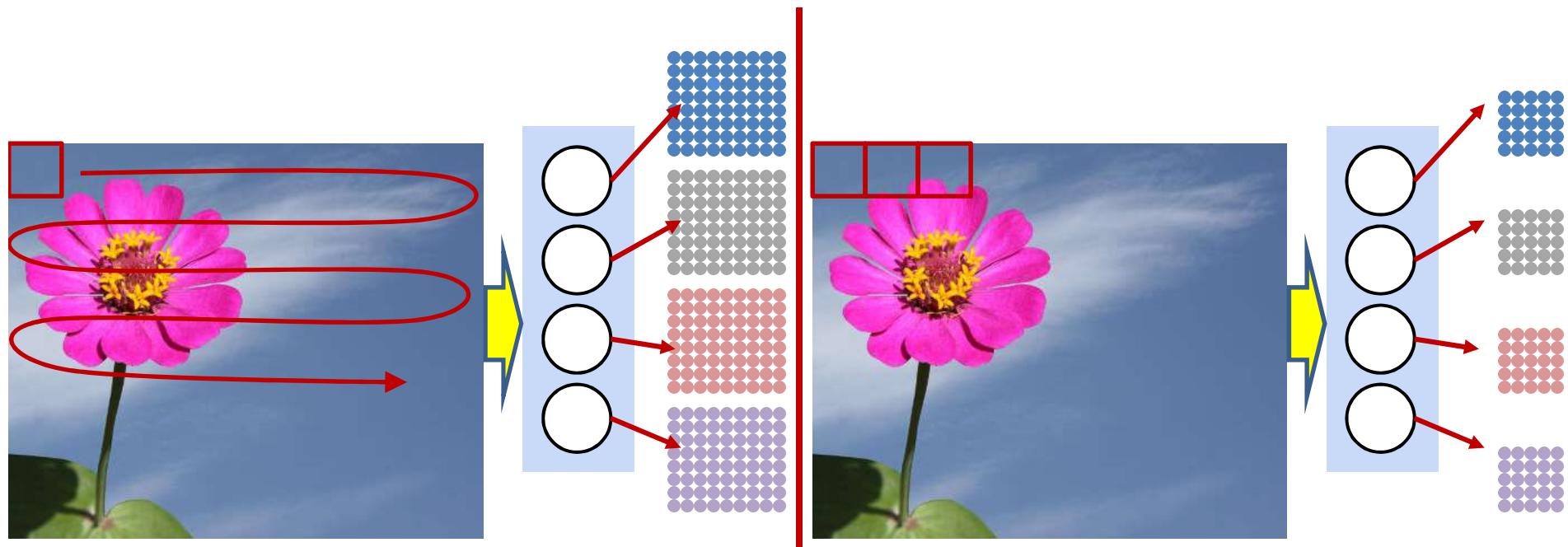
- The pattern in the *input* image that each filter sees is its “Receptive Field”
  - The squares show the *sizes* of the receptive fields for the first, second and third-layer neurons
- The actual receptive field for a first layer filter is simply its arrangement of weights
- For the higher level filters,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

# Some modifications



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

# Modification 1: Convolutional “Stride”



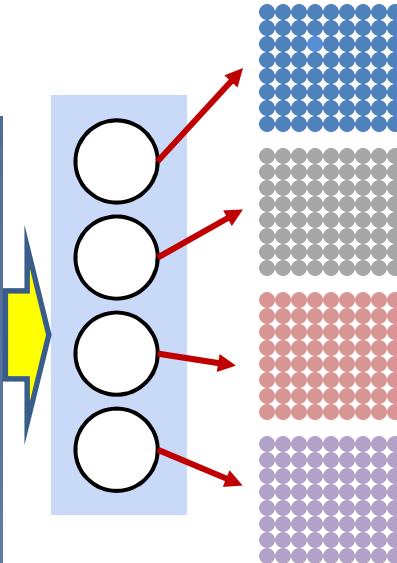
- The scans of the individual “filters” may advance by more than one pixel at a time
  - The “stride” may be greater than 1
  - Effectively increasing the granularity of the scan
    - Saves computation, sometimes at the risk of losing information
- This will result in a reduction of the size of the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

# Convolutional neural net

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

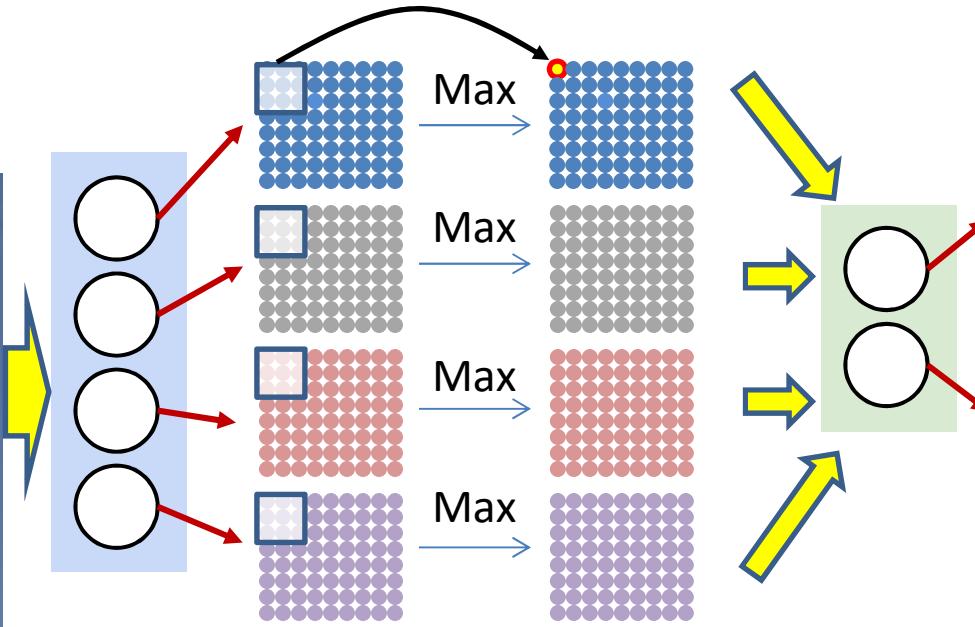
```
Y(0) = Image
for l = 1:L  # layers operate on vector at (x,y)
    for j = 1:D_l
        m = 1
        for x = 1:stride:W_{l-1}-K_l+1
            n = 1
            for y = 1:stride:H_{l-1}-K_l+1
                segment = Y(l-1, :, x:x+K_l-1, y:y+K_l-1) #3D tensor
                z(l, j, m, n) = W(l,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

# Accounting for jitter



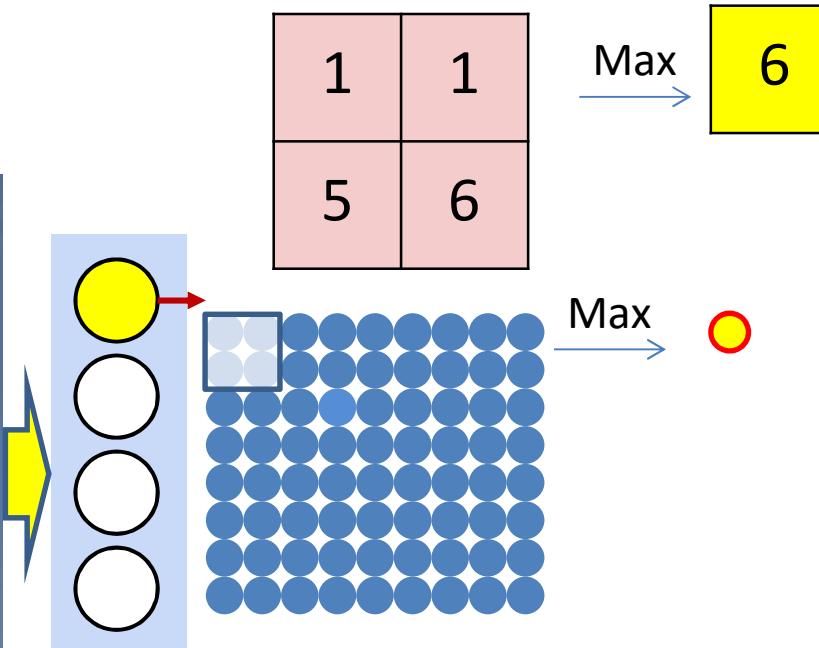
- We would like to account for some jitter in the first-level patterns
  - If a pattern shifts by one pixel, is it still a petal?

# Accounting for jitter



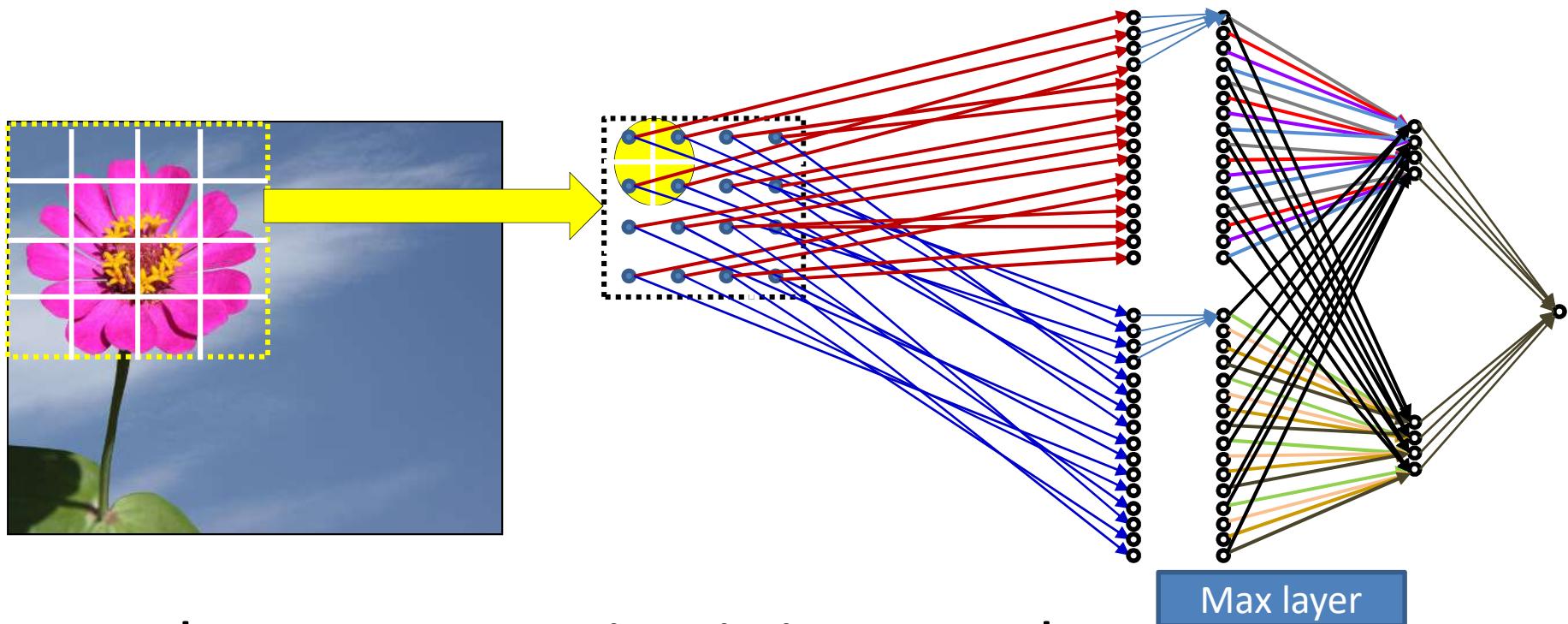
- We would like to account for some jitter in the first-level patterns
  - If a pattern shifts by one pixel, is it still a petal?
  - A small jitter is acceptable
    - Replace each value by the maximum of the values within a small region around it
      - *Max filtering or Max pooling*

# Accounting for jitter



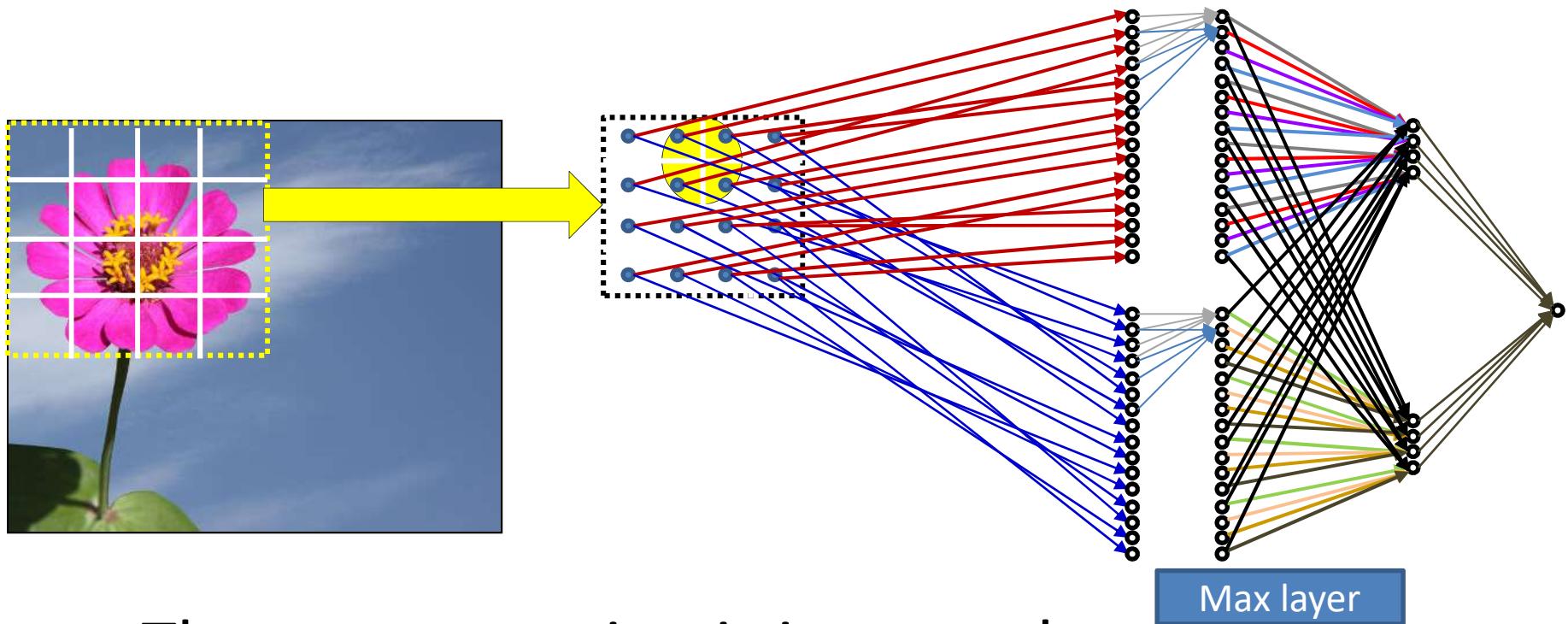
- We would like to account for some jitter in the first-level patterns
  - If a pattern shifts by one pixel, is it still a petal?
  - A small jitter is acceptable
    - Replace each value by the maximum of the values within a small region around it
      - *Max filtering or Max pooling*

# The max operation is just a neuron



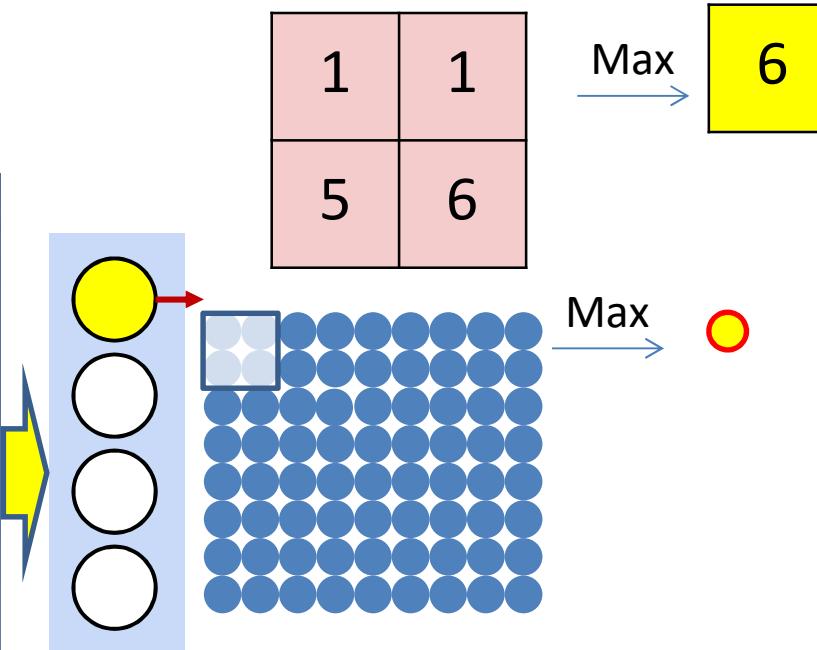
- The max operation is just another neuron
- Instead of applying an activation to the weighted sum of inputs, each neuron just computes the maximum over all inputs

# The max operation is just a neuron



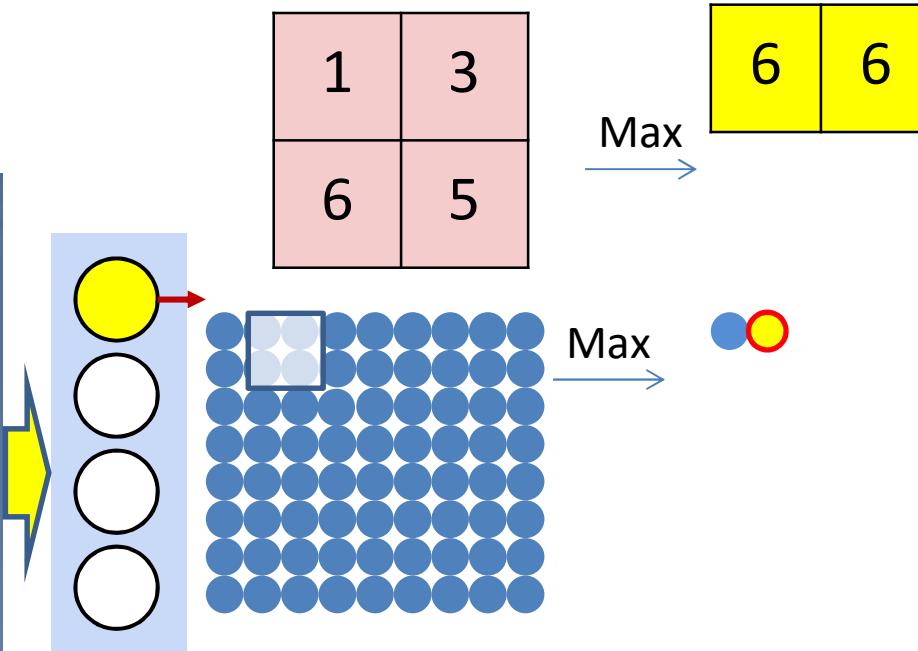
- The max operation is just another neuron
- Instead of applying an activation to the weighted sum of inputs, each neuron just computes the maximum over all inputs

# Accounting for jitter



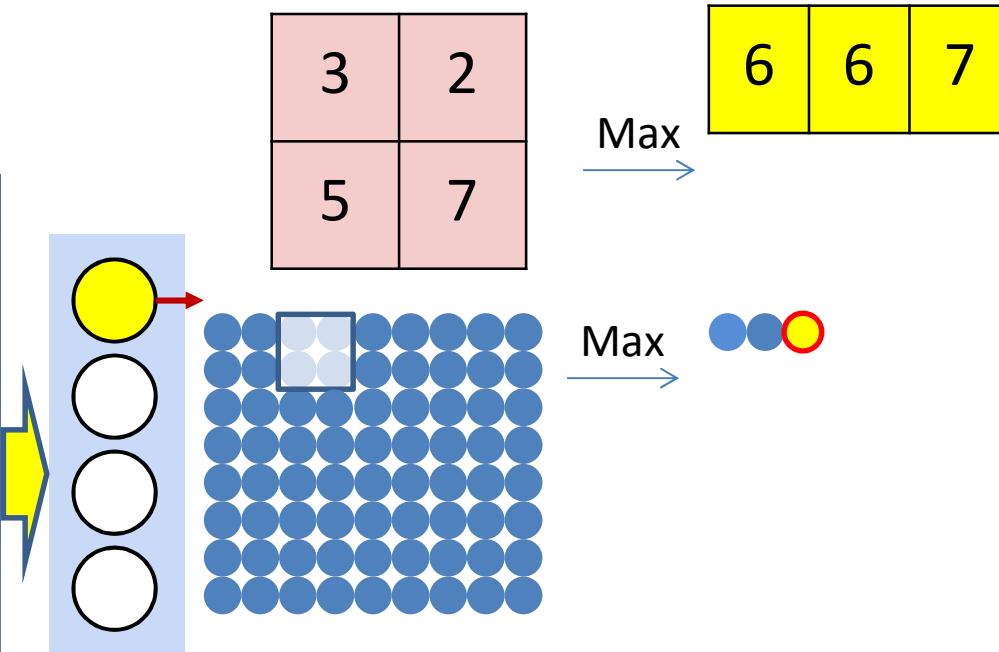
- The max filtering can also be performed as a scan

# Accounting for jitter



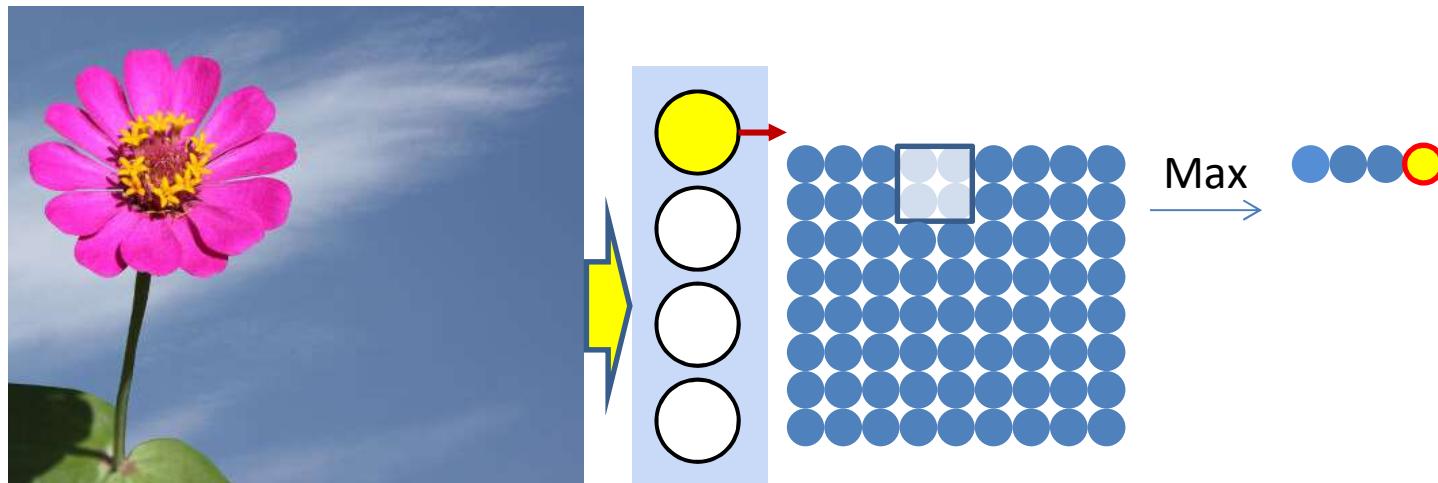
- The “max filter” operation too “scans” the picture

# Accounting for jitter



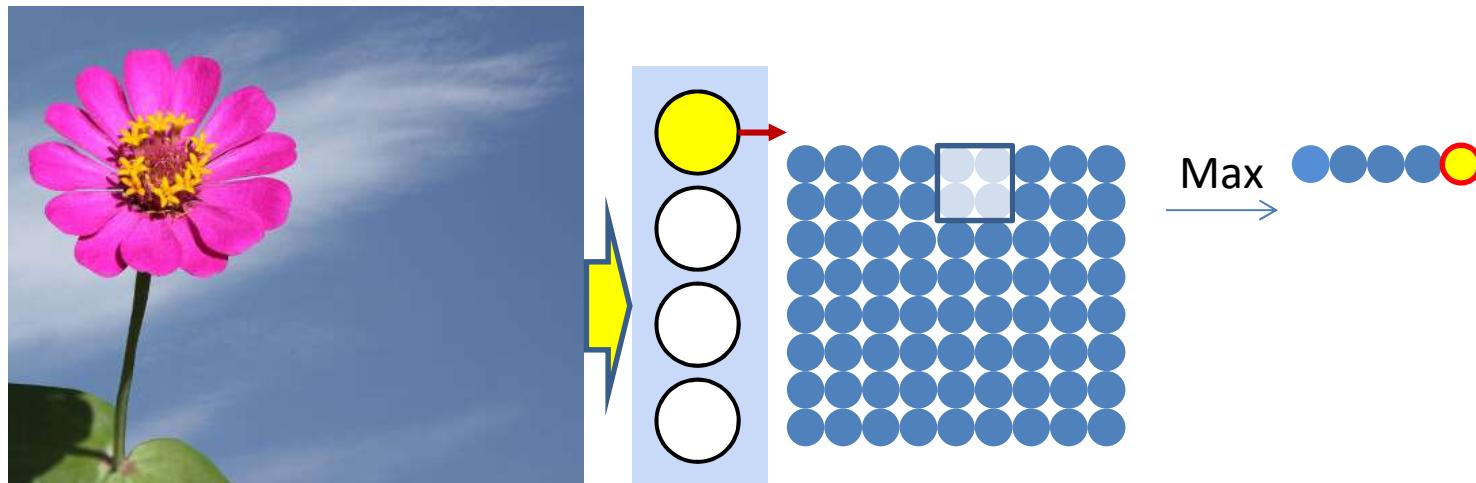
- The “max filter” operation too “scans” the picture

# Accounting for jitter



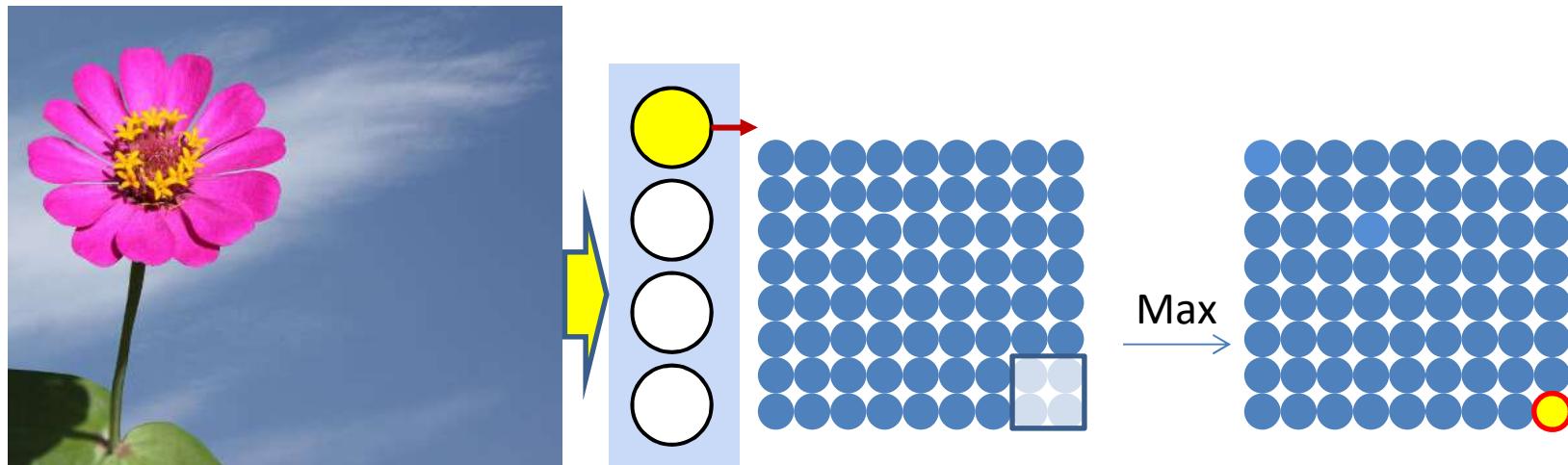
- The “max filter” operation too “scans” the picture

# Accounting for jitter



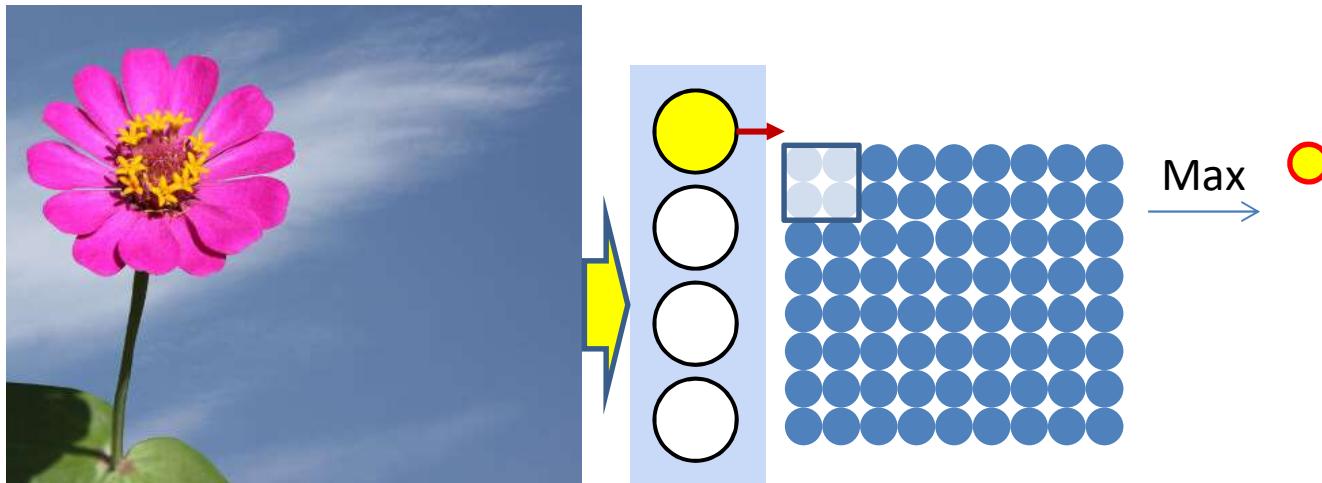
- The “max filter” operation too “scans” the picture

# Accounting for jitter



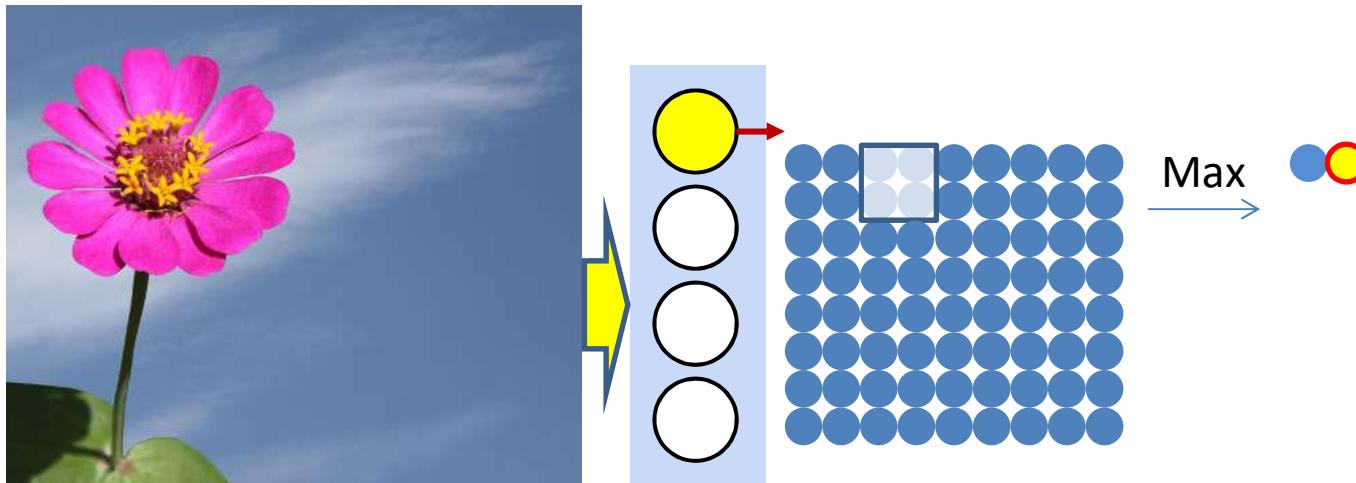
- The “max filter” operation too “scans” the picture

# Max pooling “Strides”



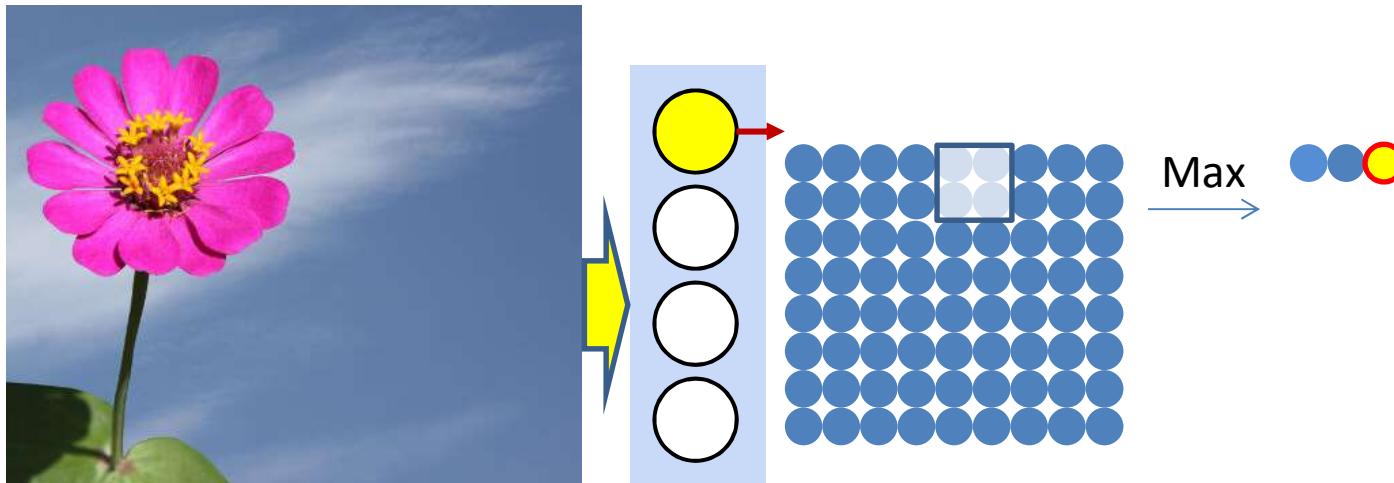
- The “max” operations may “stride” by more than one pixel

# Max pooling “Strides”



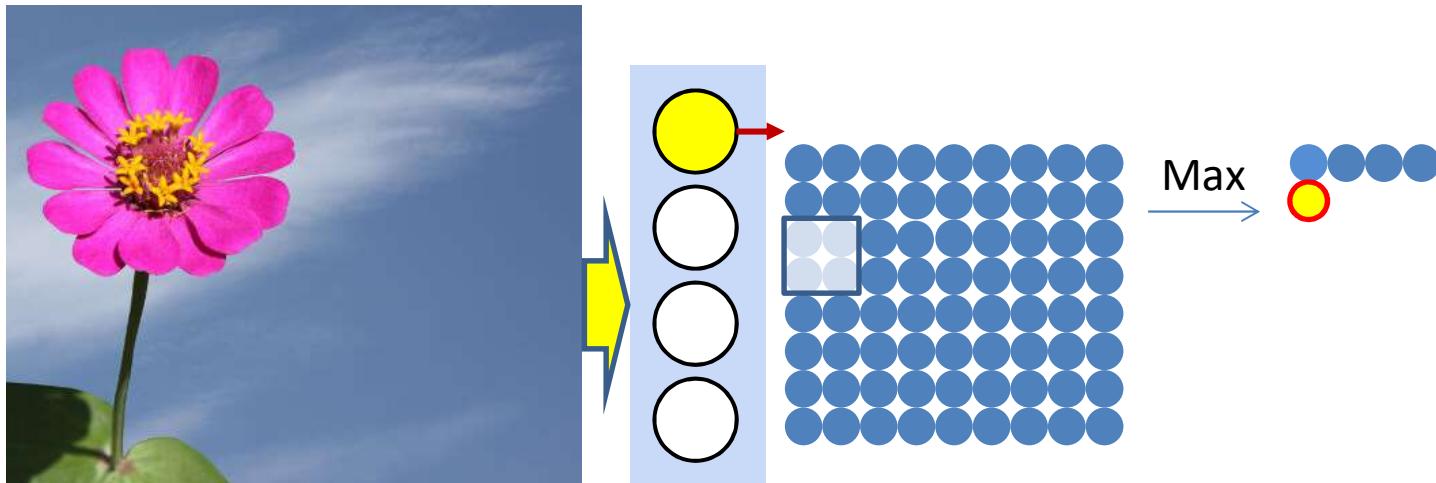
- The “max” operations may “stride” by more than one pixel

# Max pooling “Strides”



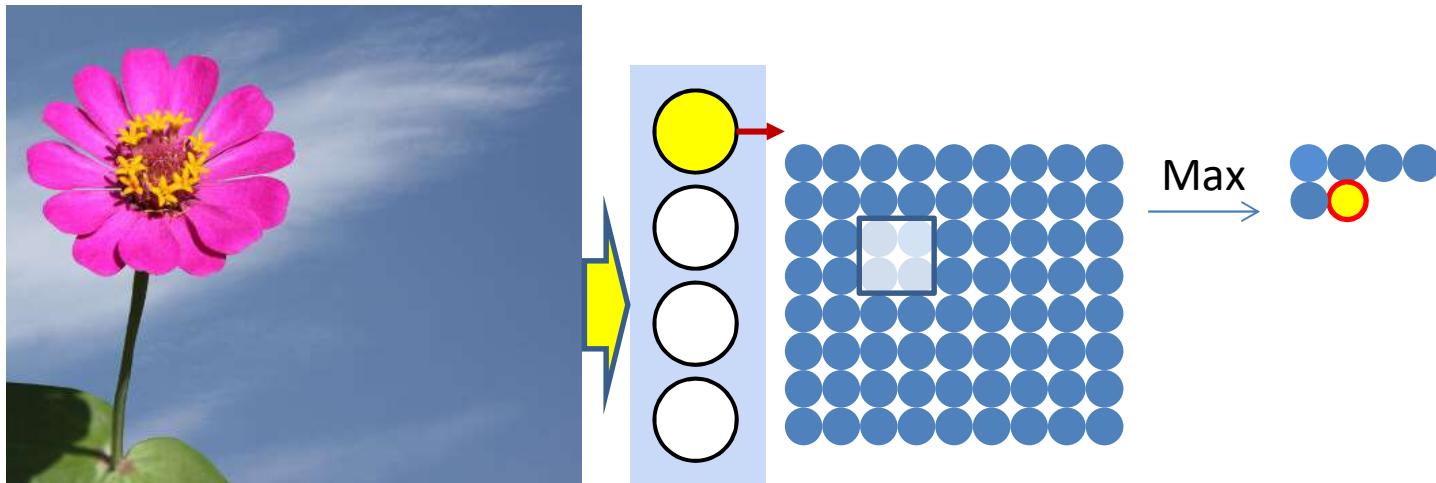
- The “max” operations may “stride” by more than one pixel

# Max pooling “Strides”



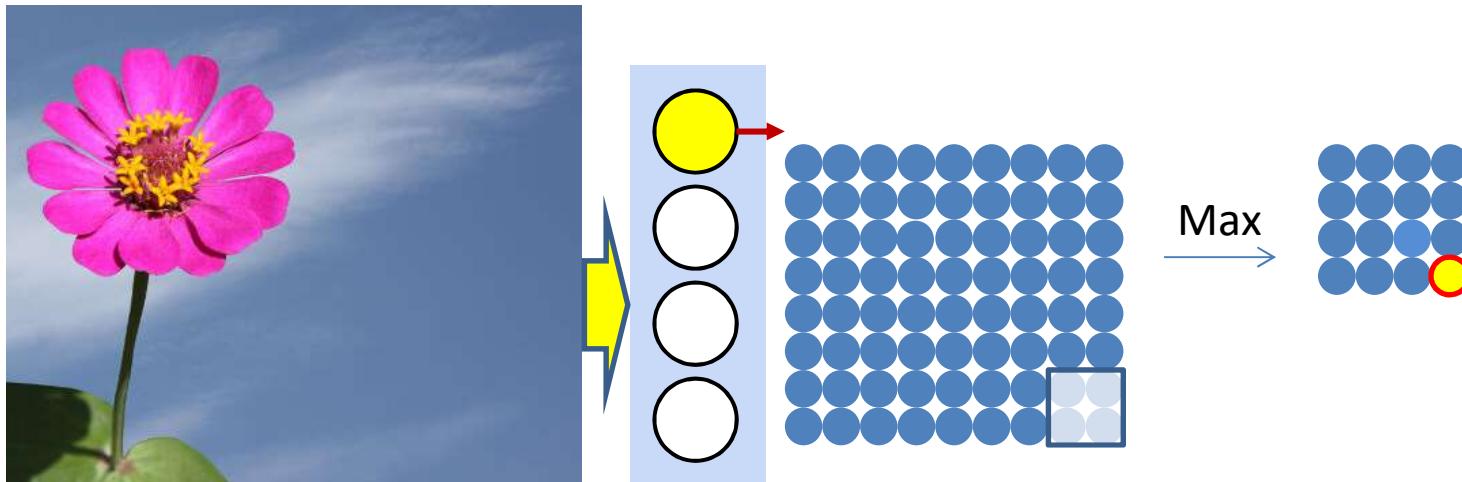
- The “max” operations may “stride” by more than one pixel

# Max pooling “Strides”



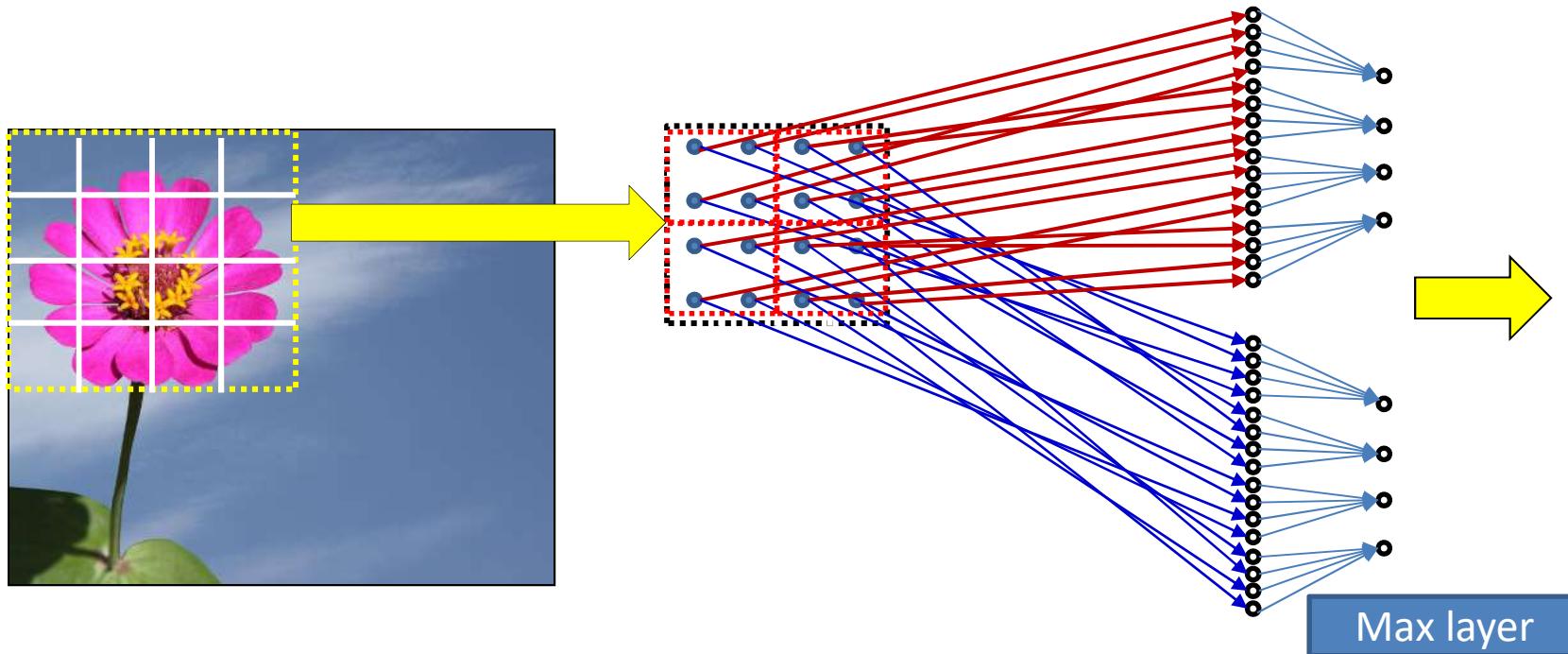
- The “max” operations may “stride” by more than one pixel

# Max pooling “Strides”



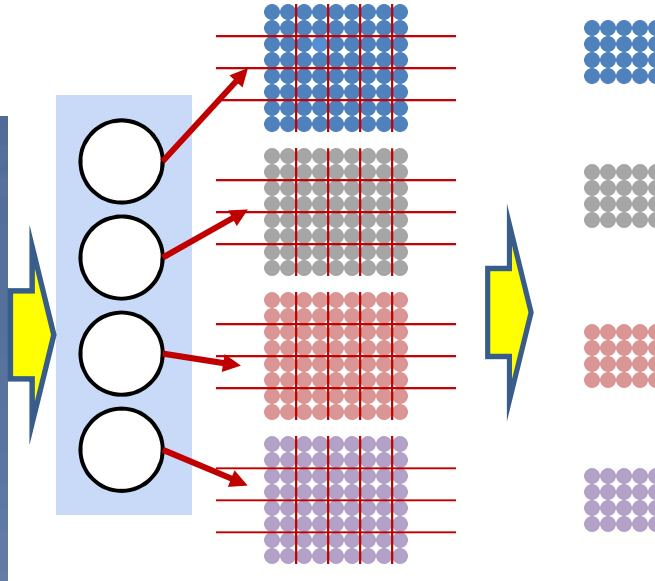
- The “max” operations may “stride” by more than one pixel
  - This will result in a *shrinking* of the map
  - The operation is usually called “pooling”
    - Pooling a number of outputs to get a single output
    - When stride is greater than 1, also called “Down sampling”

# Shrinking with a max



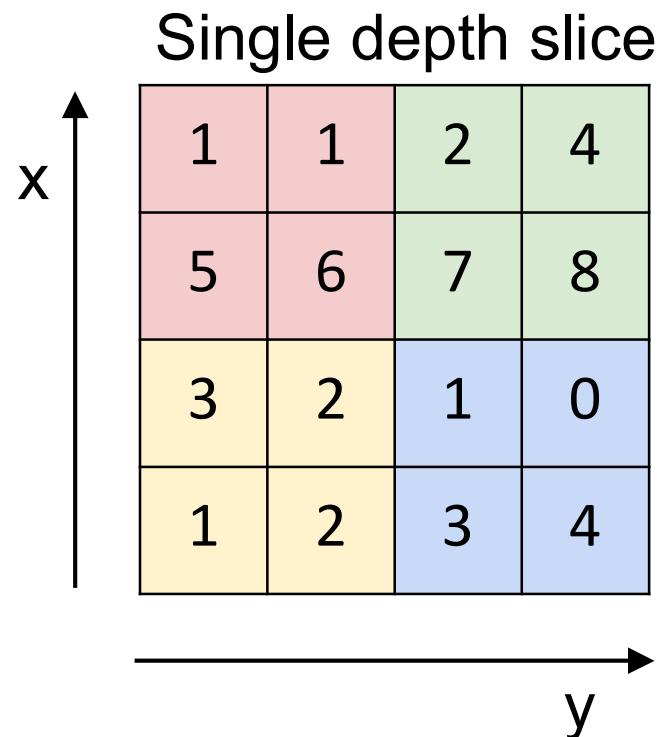
- In this example we *shrank* the image after the max
  - Adjacent “max” operators did not overlap
  - The stride was the size of the max filter itself

# Non-overlapped strides



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

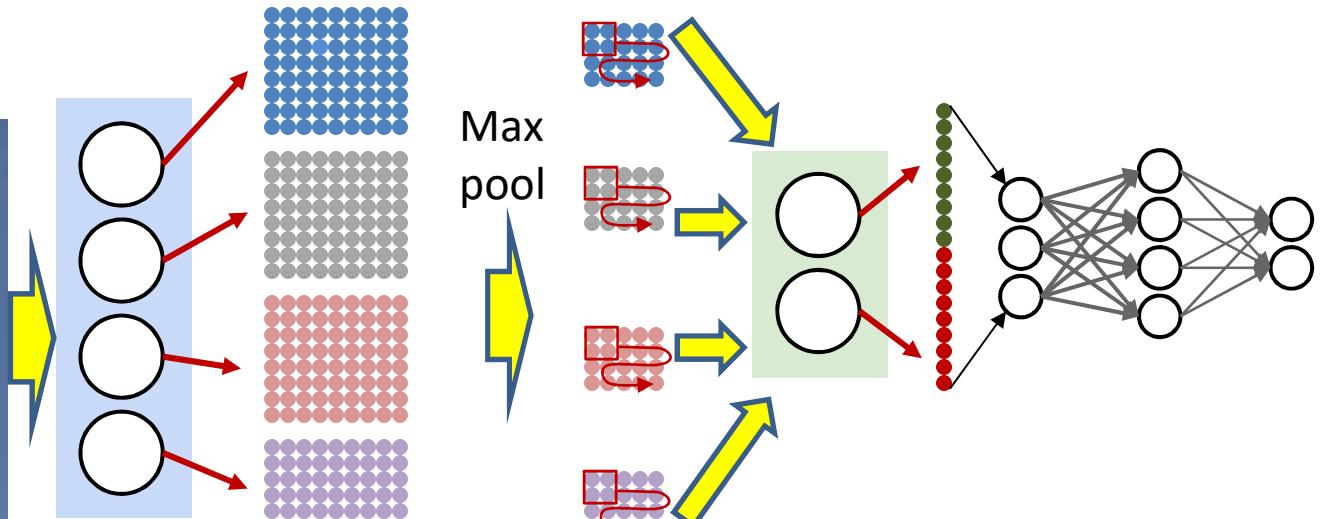
# Max Pooling



max pool with 2x2 filters  
and stride 2

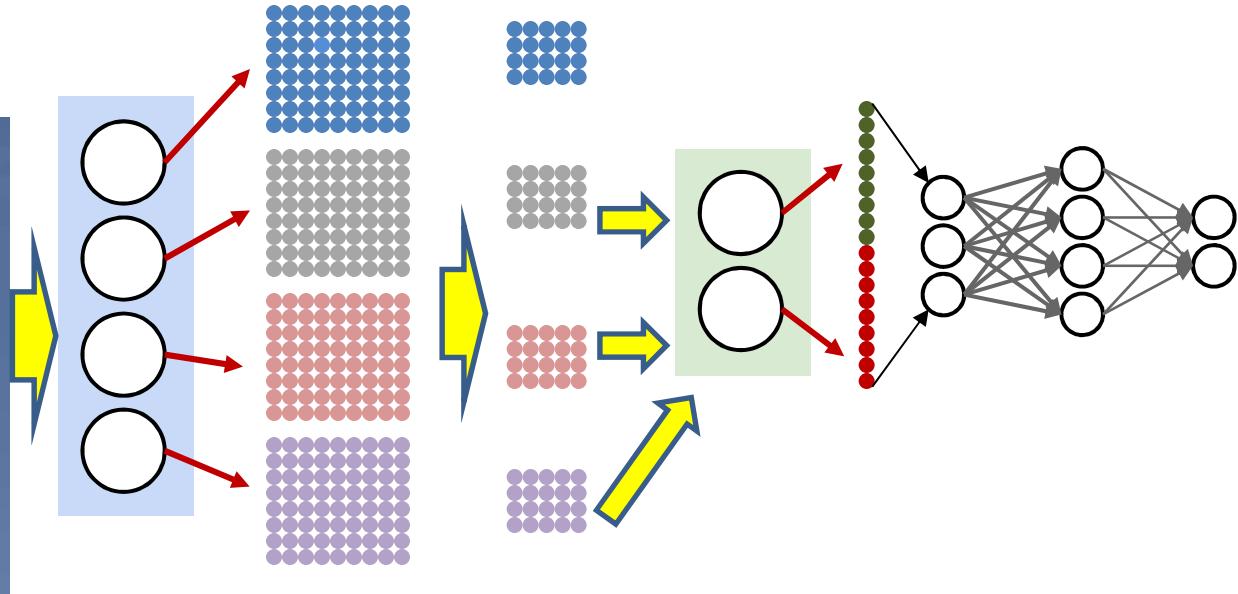
6	8
3	4

# Higher layers



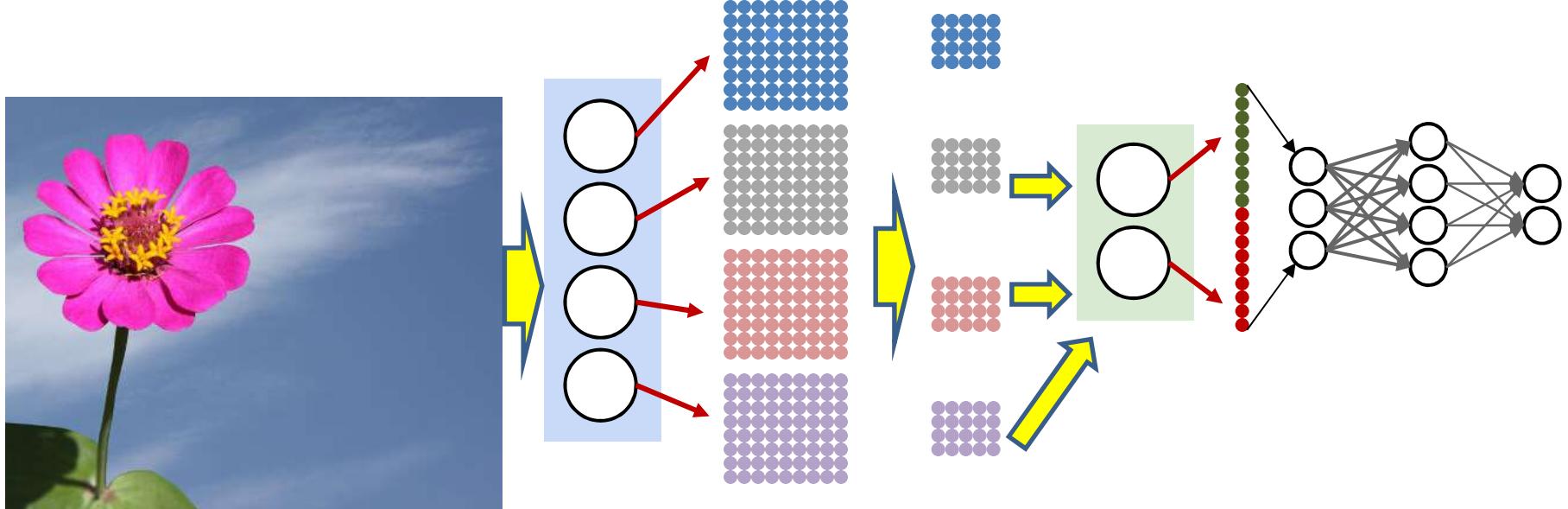
- The next layer works on the *max-pooled* maps

# The overall structure



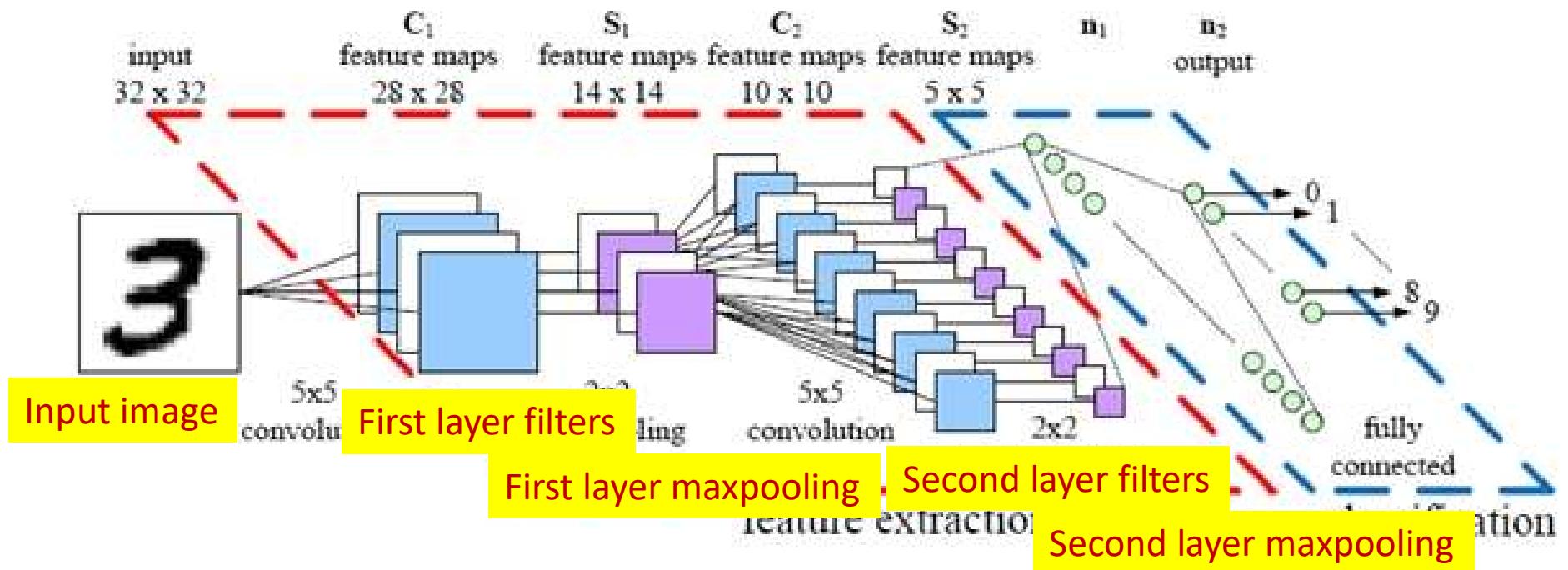
- In reality we can have many layers of “convolution” (scanning) followed by max pooling (and reduction) before the final MLP
  - The individual perceptrons at any “scanning” or “convolutional” layer are called “filters”
    - They “filter” the input image to produce an output image (map)
  - The individual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

# The overall structure

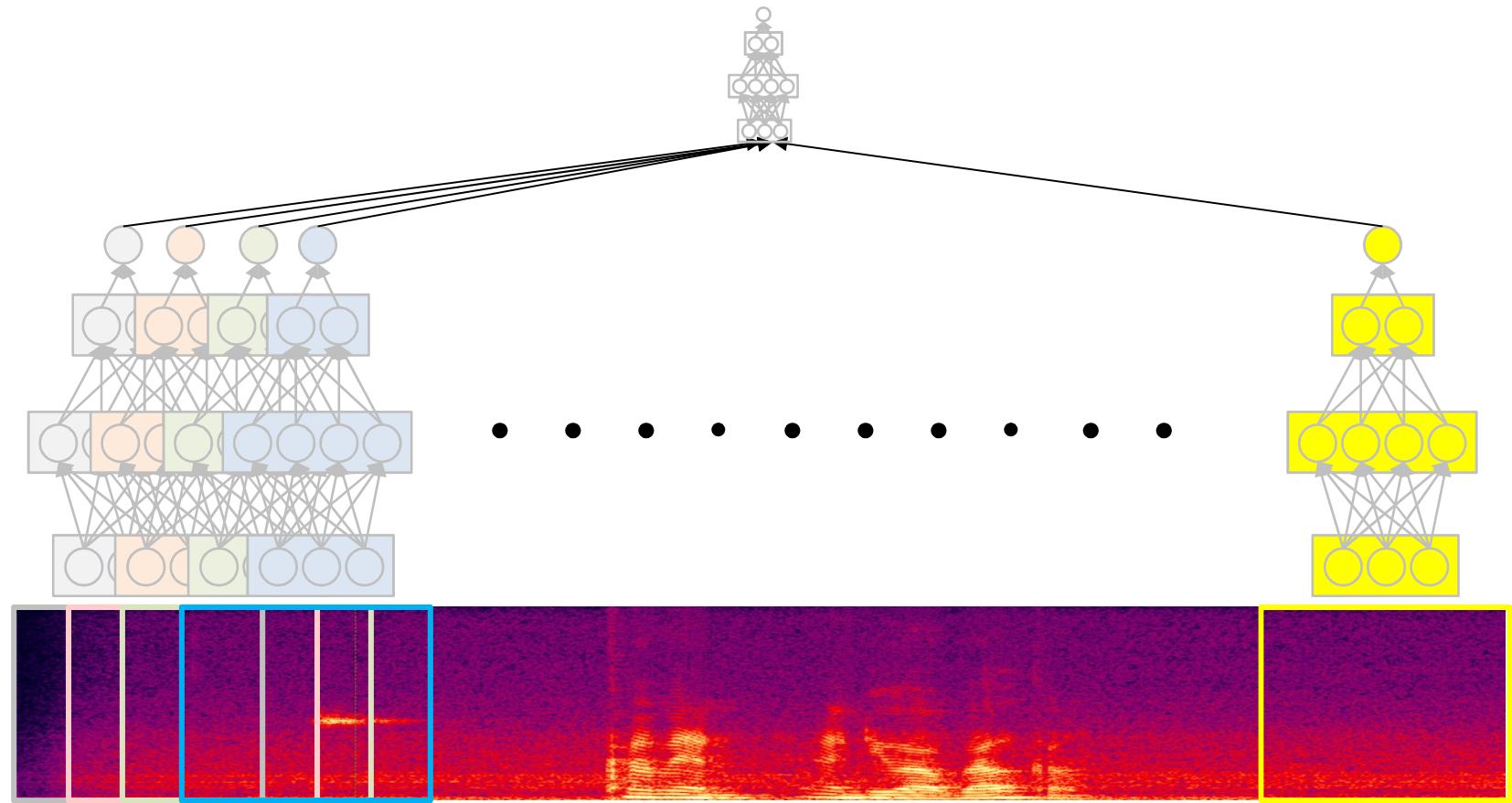


- This entire structure is called a ***Convolutional Neural Network***

# Convolutional Neural Network

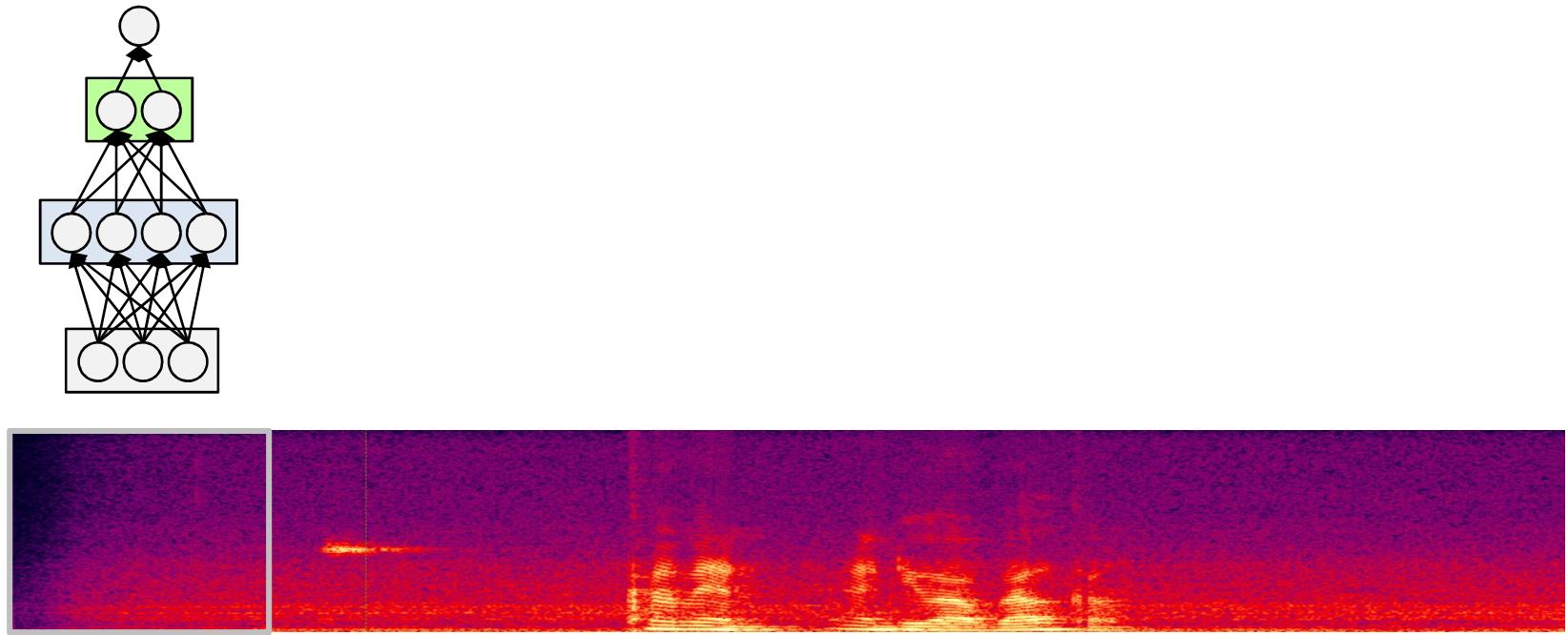


# 1-D convolution



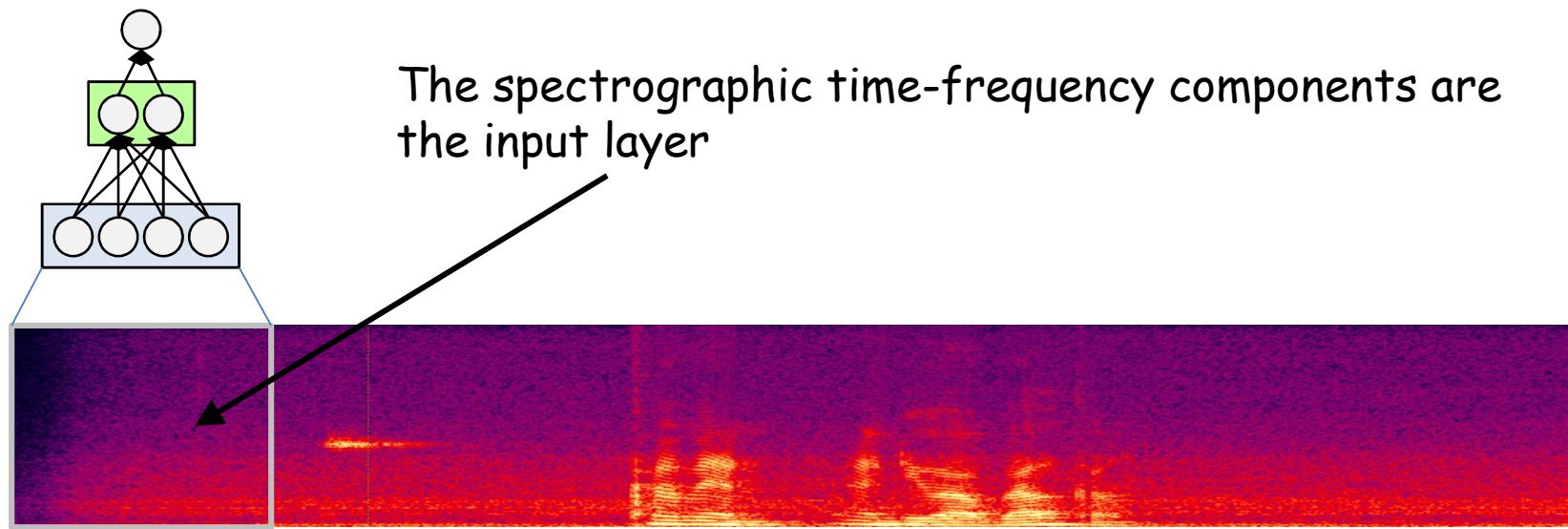
- The 1-D scan version of the convolutional neural network is the *time-delay neural network*
  - Used primarily for speech recognition

# 1-D scan version



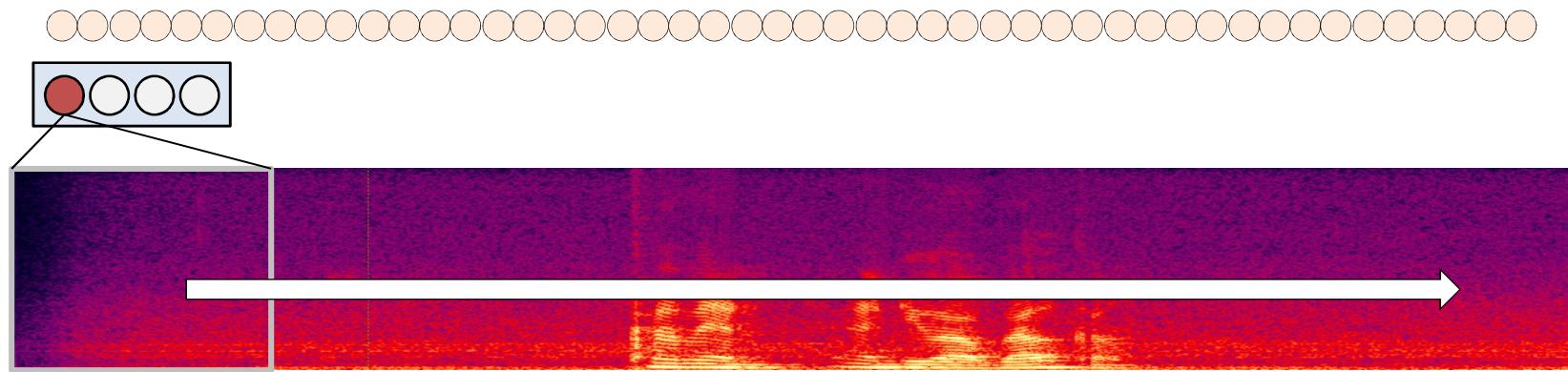
- The 1-D scan version of the convolutional neural network

# 1-D scan version



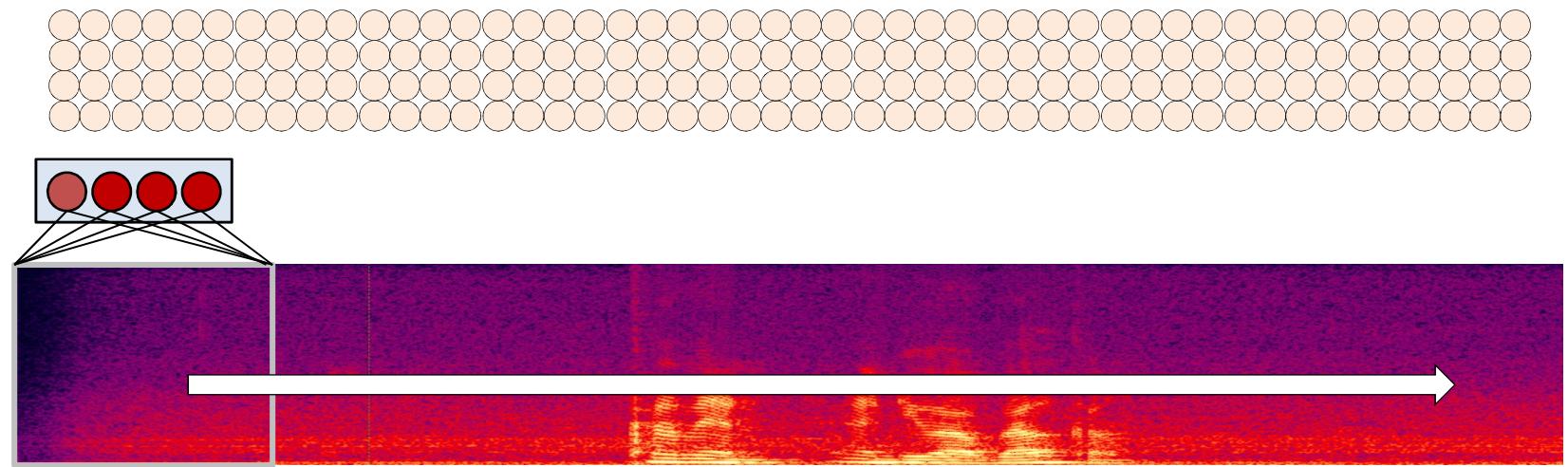
- The 1-D scan version of the convolutional neural network

# 1-D scan version



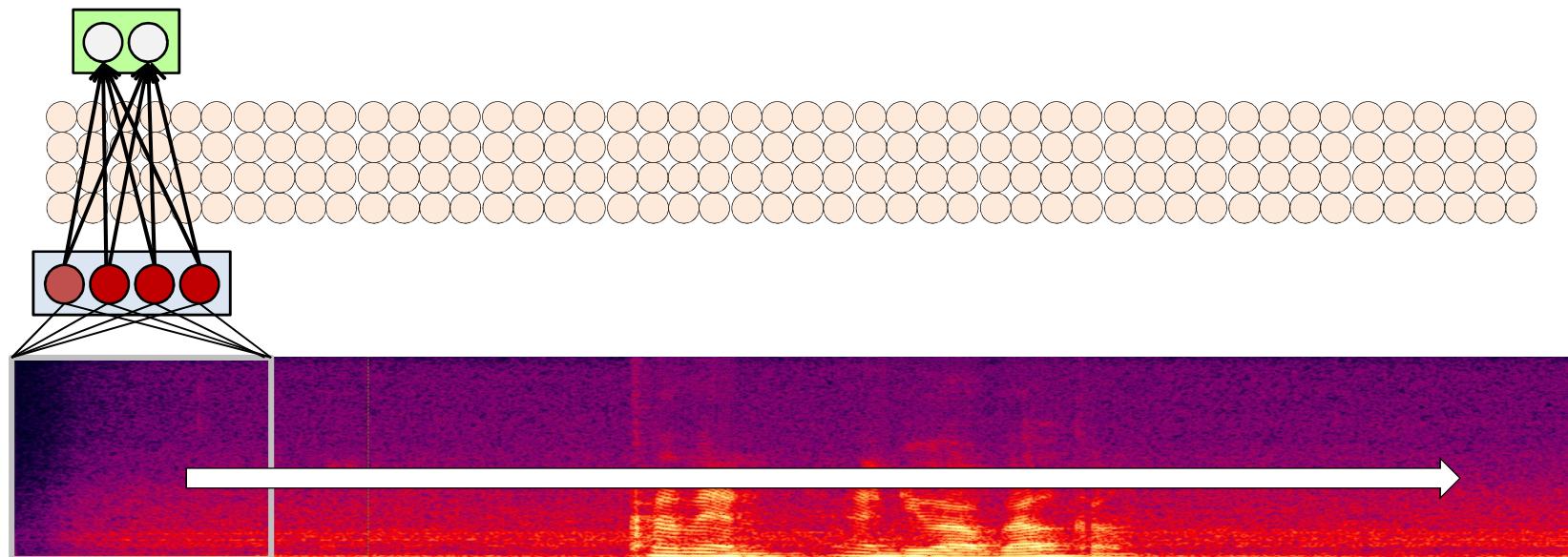
- The 1-D scan version of the convolutional neural network

# 1-D scan version



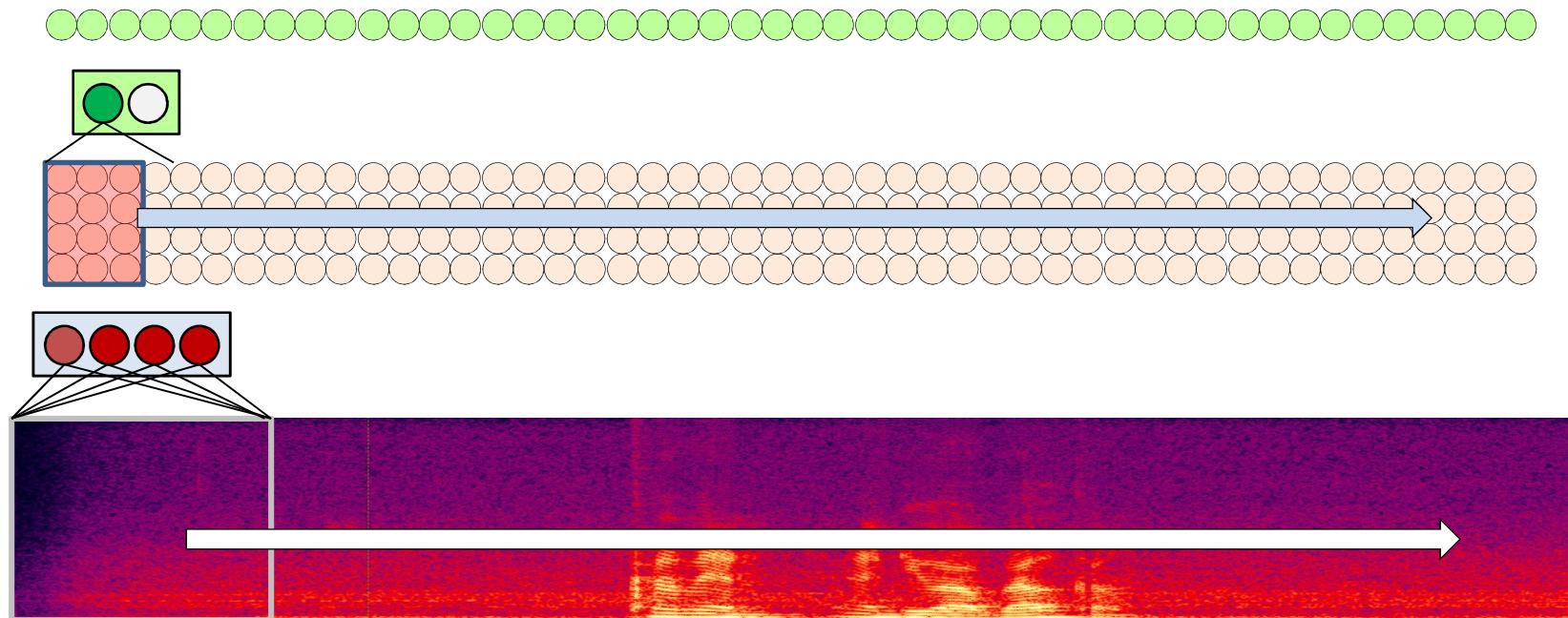
- The 1-D scan version of the convolutional neural network

# 1-D scan version



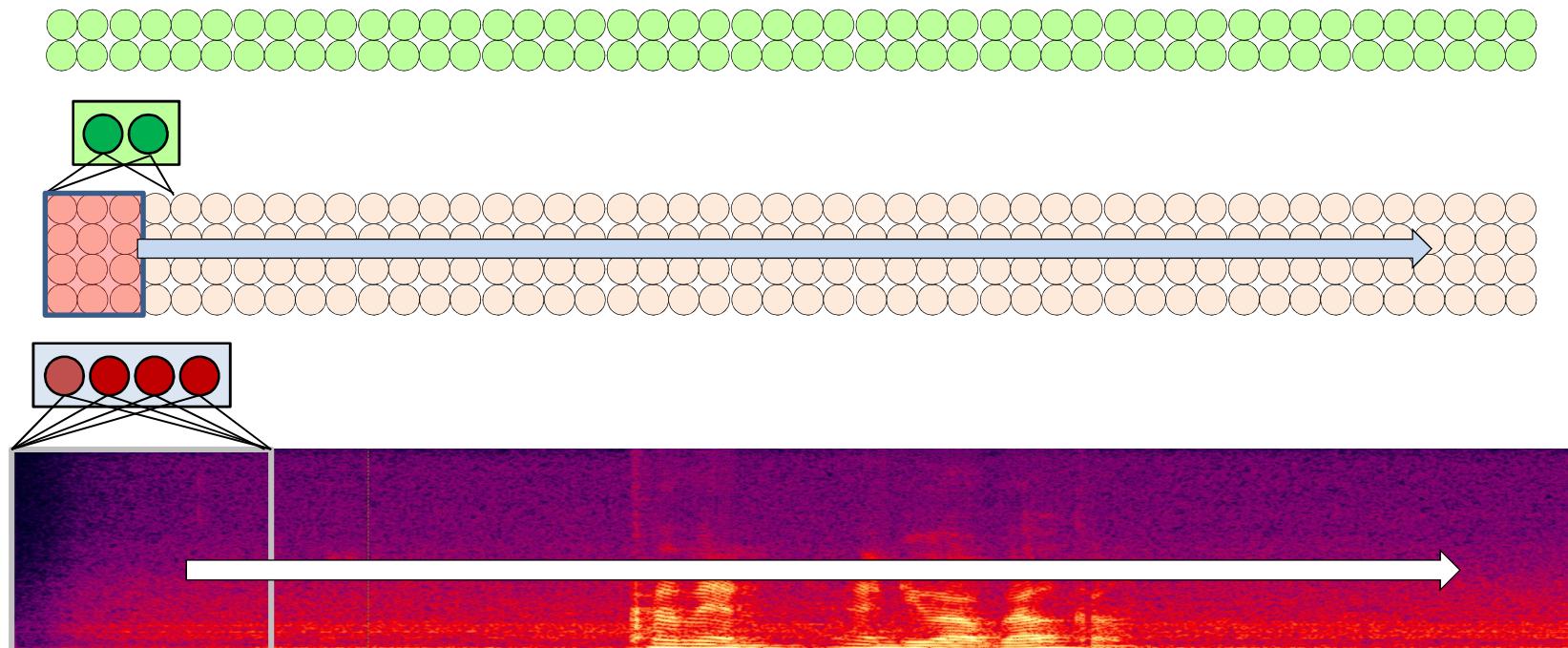
- The 1-D scan version of the convolutional neural network
  - Max pooling optional
    - Not generally done for speech

# 1-D scan version



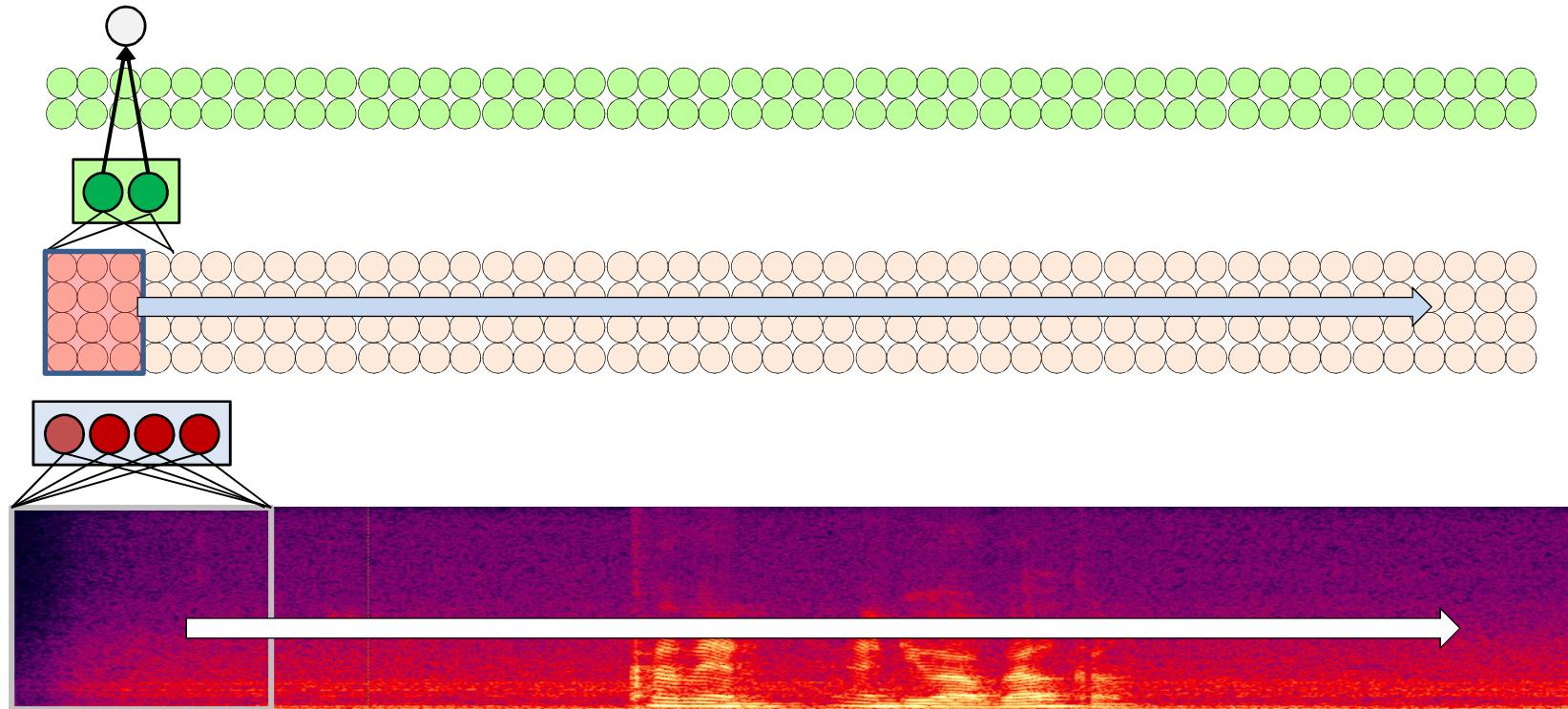
- The 1-D scan version of the convolutional neural network
  - Max pooling optional
    - Not generally done for speech

# 1-D scan version



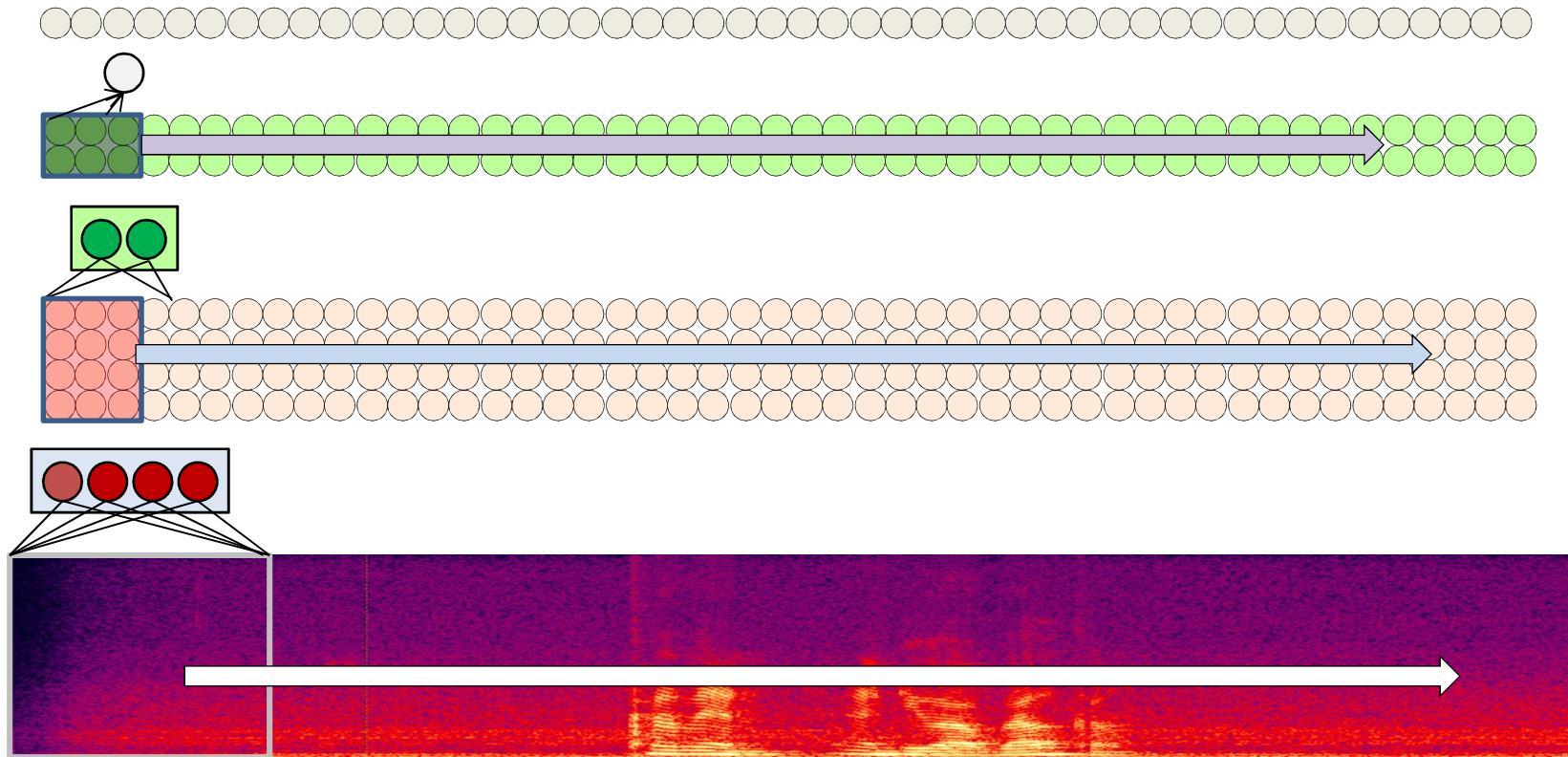
- The 1-D scan version of the convolutional neural network
  - Max pooling optional
    - Not generally done for speech

# 1-D scan version



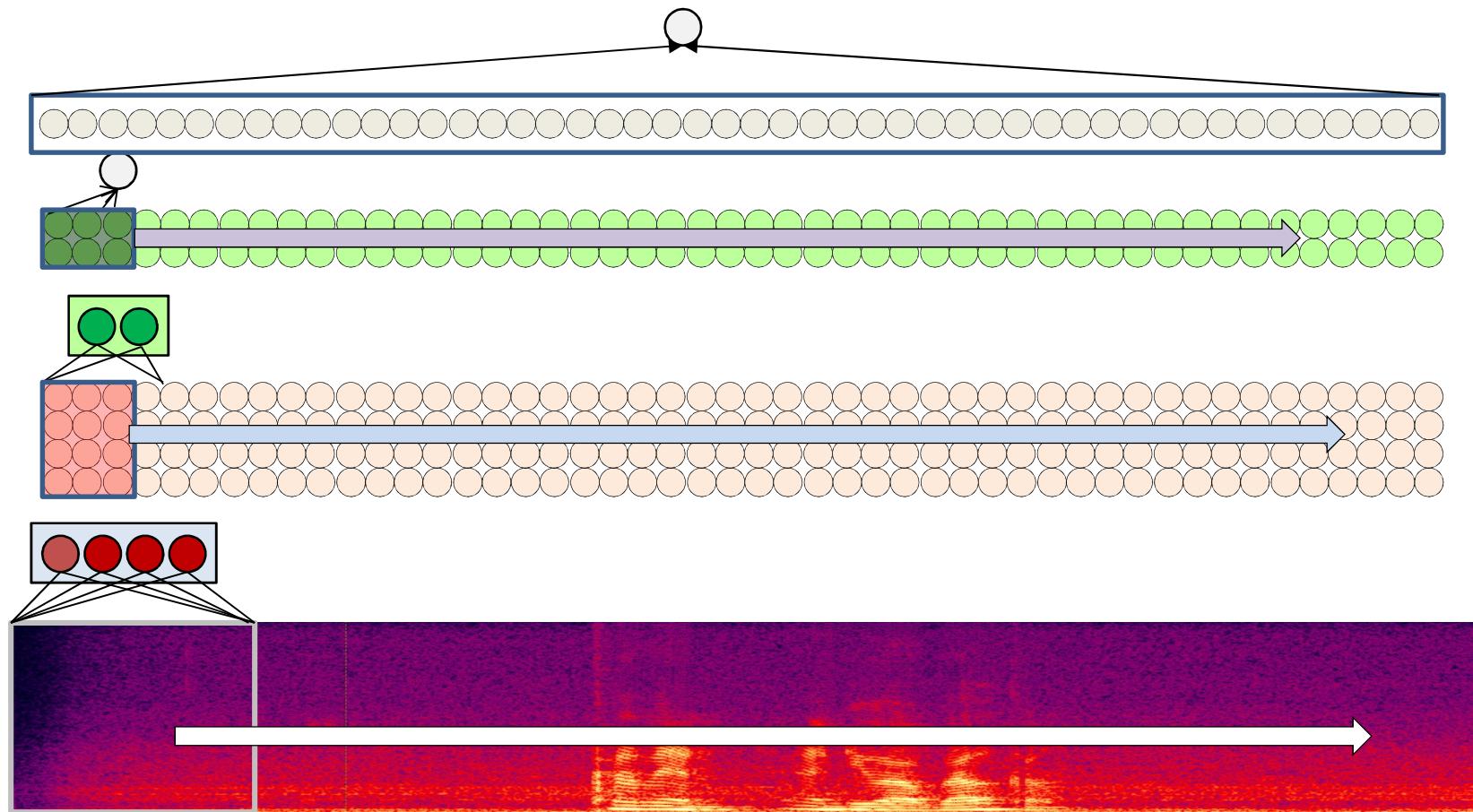
- The 1-D scan version of the convolutional neural network
  - Max pooling optional
    - Not generally done for speech

# 1-D scan version



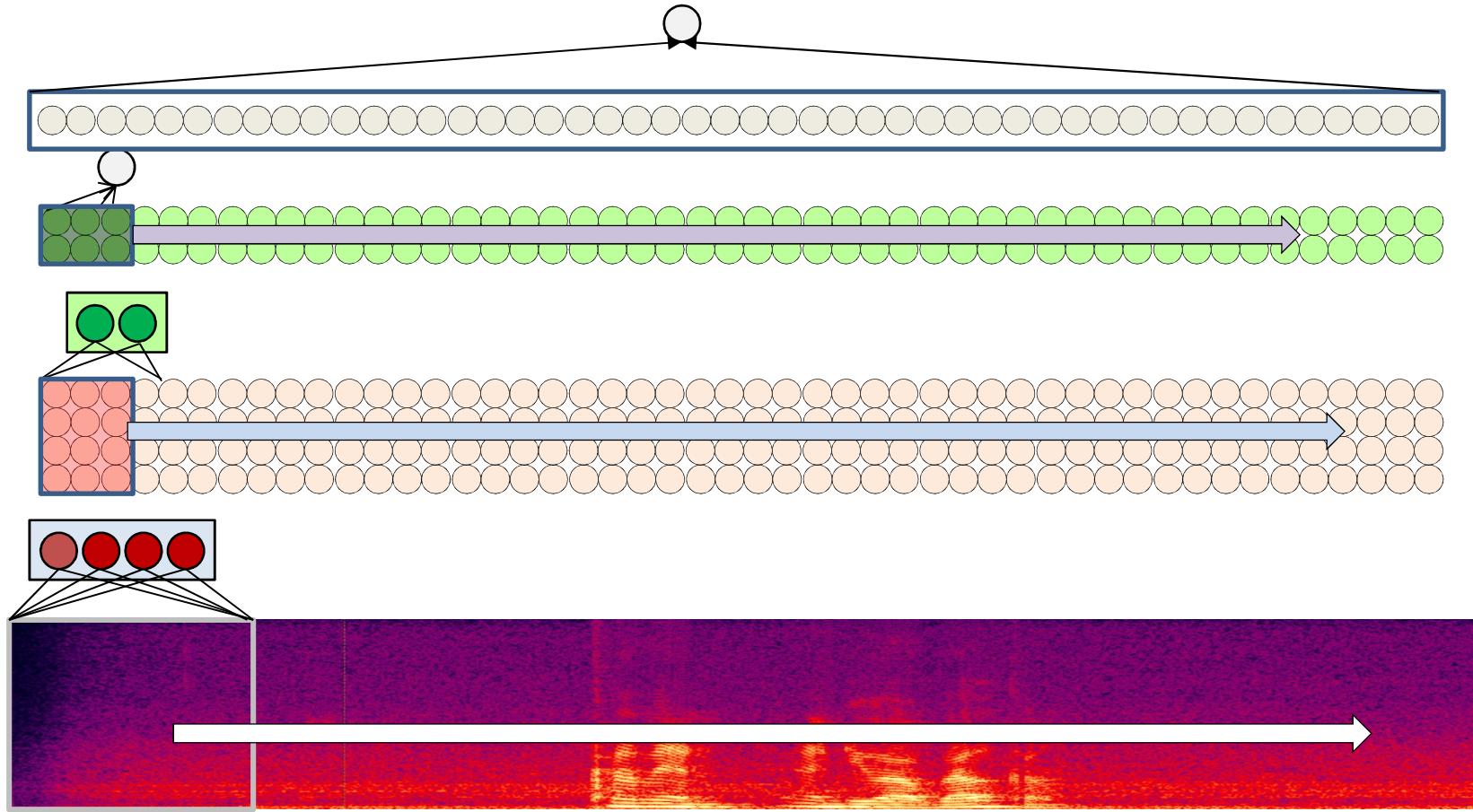
- The 1-D scan version of the convolutional neural network
  - Max pooling optional
    - Not generally done for speech

# 1-D scan version



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

# Time-Delay Neural Network



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

# Story so far

- Neural networks learn patterns in a hierarchical manner
  - Simple to complex
- Pattern classification tasks such as “does this picture contain a cat” are best performed by scanning for the target pattern
- Scanning for patterns can be viewed as classification with a large shared-parameter network
- Scanning an input with a network and combining the outcomes is equivalent to scanning with individual neurons
  - First level neurons scan the input
  - Higher-level neurons scan the “maps” formed by lower-level neurons
  - A final “decision” layer (which may be a max, a perceptron, or an MLP) makes the final decision
- 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