Convolutional Neural Networks

A. M. Sadeghzadeh, Ph.D.

Sharif University of Technology
Computer Engineering Department (CE)
Data and Network Security Lab (DNSL)

March 6, 2023

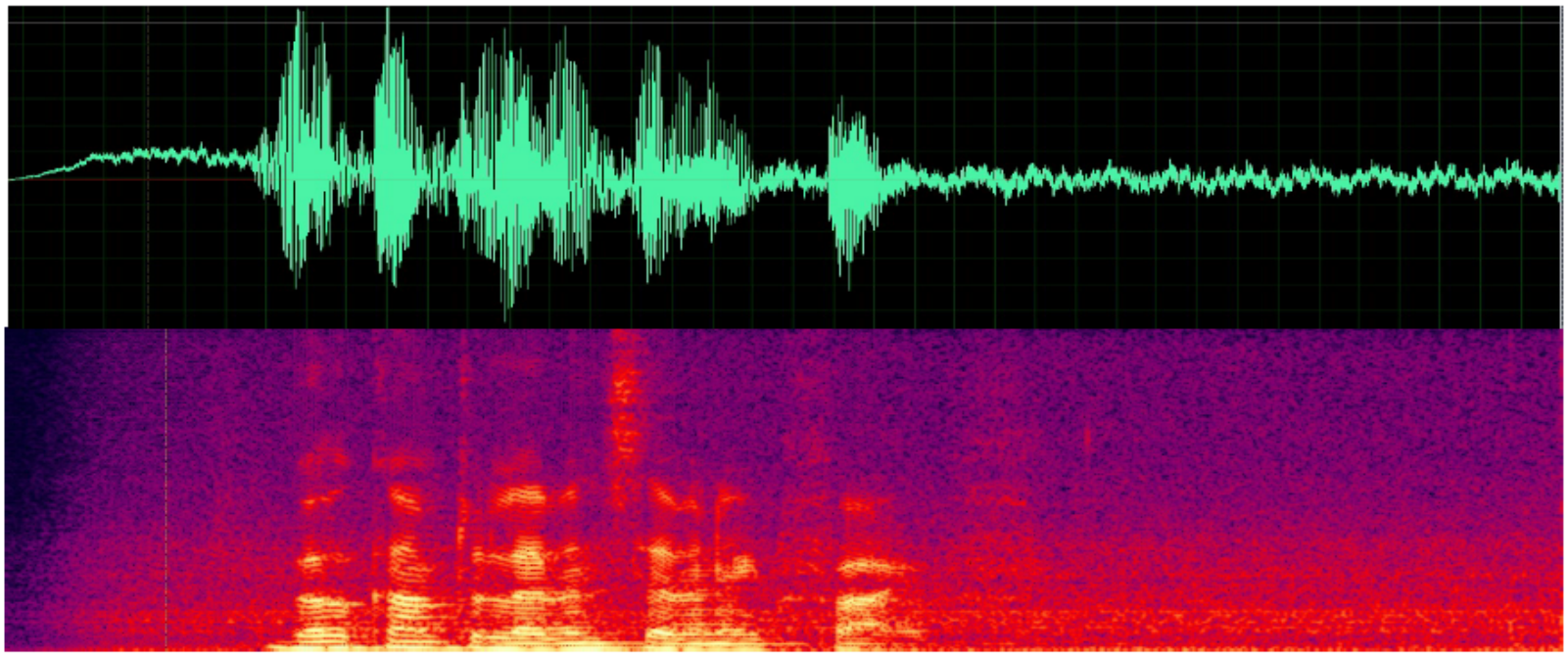
Most slides have been adapted from Bhiksha Raj, 11-785, CMU 2020, Justin Johnson, EECS 498-007, University of Michigan 2020, and Fei Fei Li, cs231n, Stanford 2017

Today's agenda

Recap

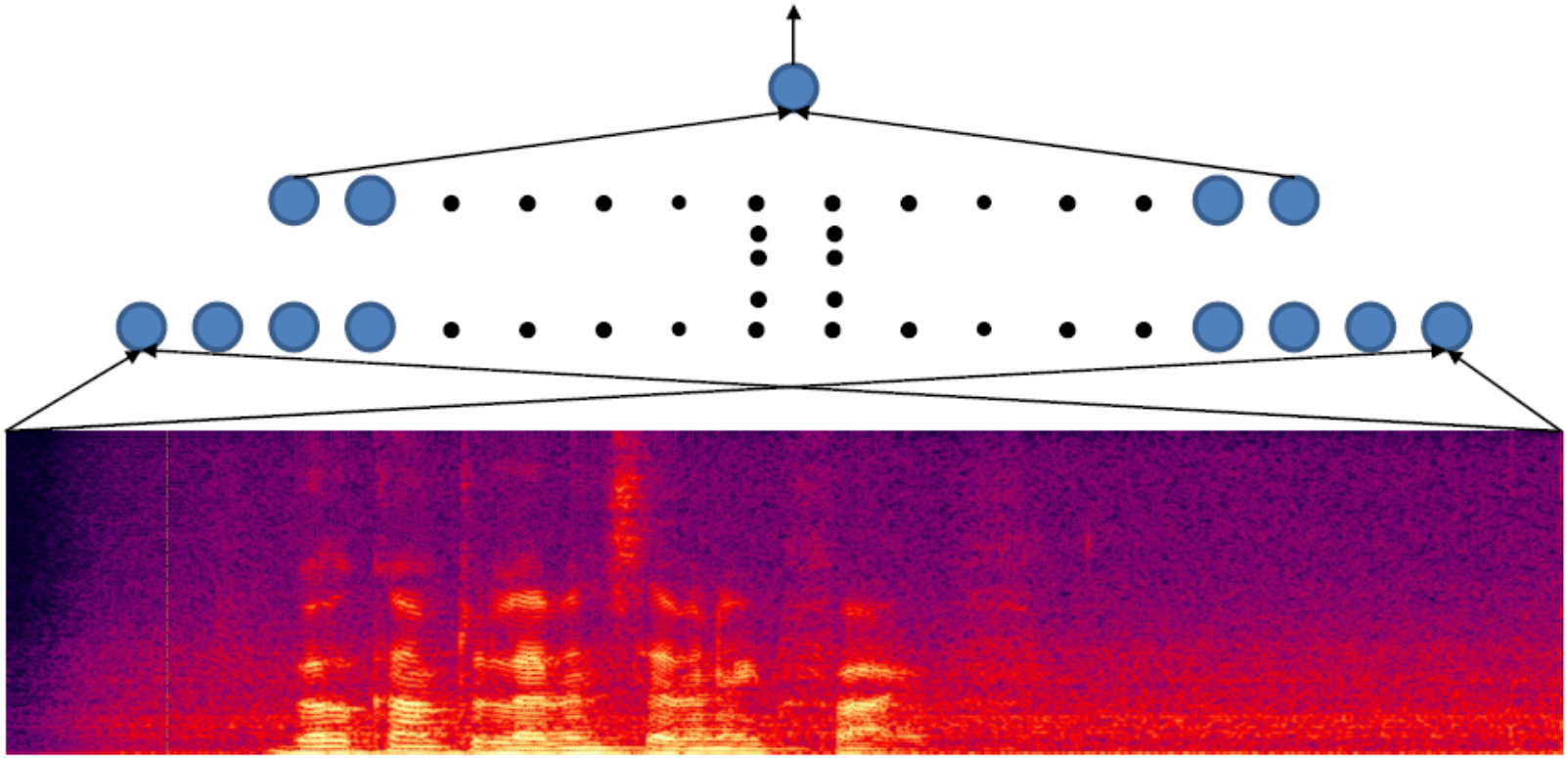
A new problem

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



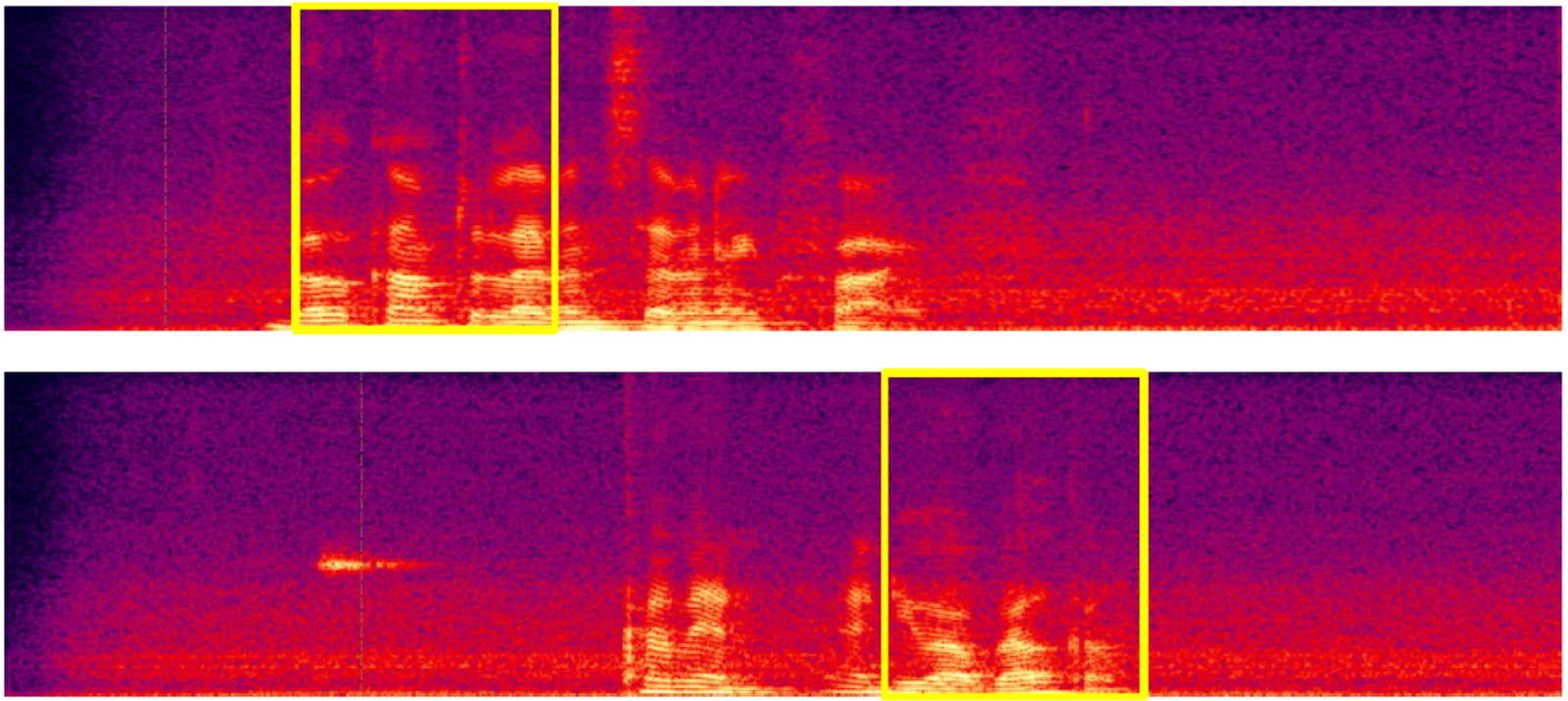
Finding a Welcome

- Trivial solution: Train an MLP for the entire recording



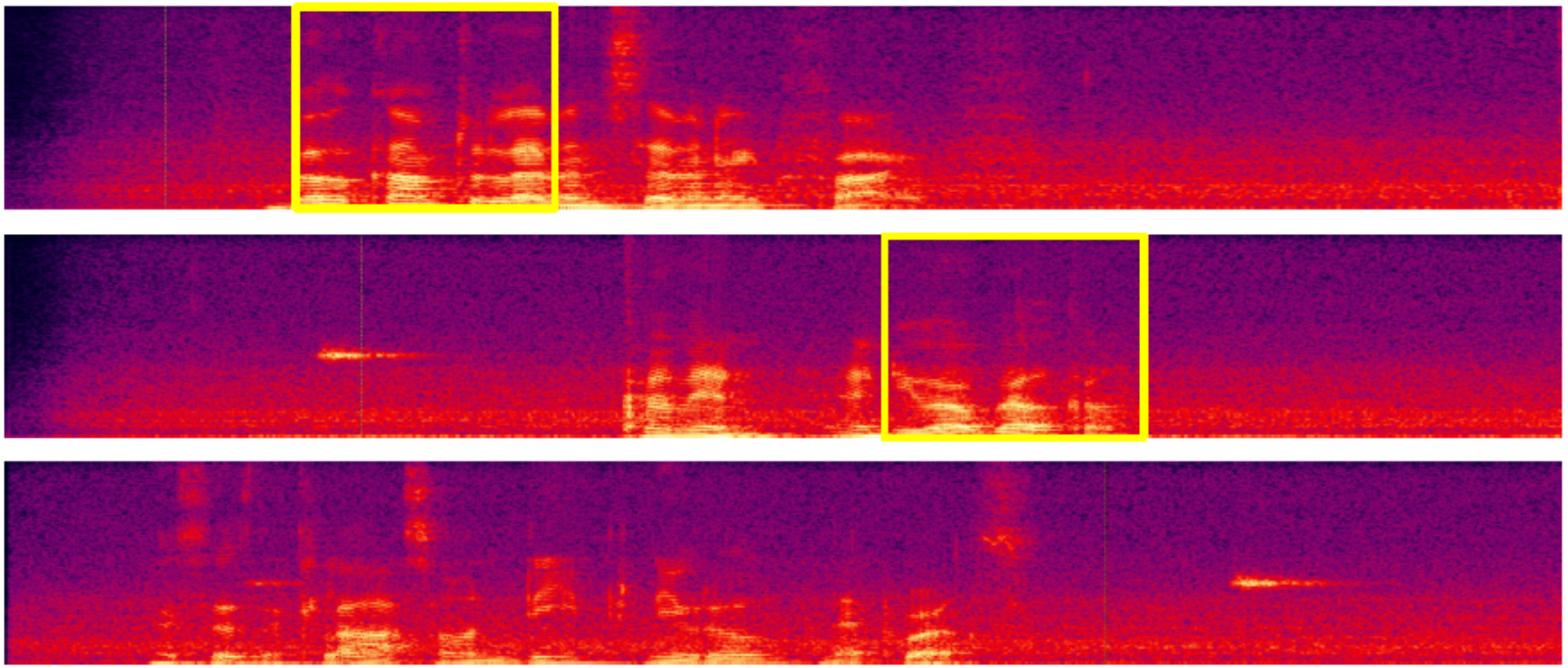
Finding a Welcome

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



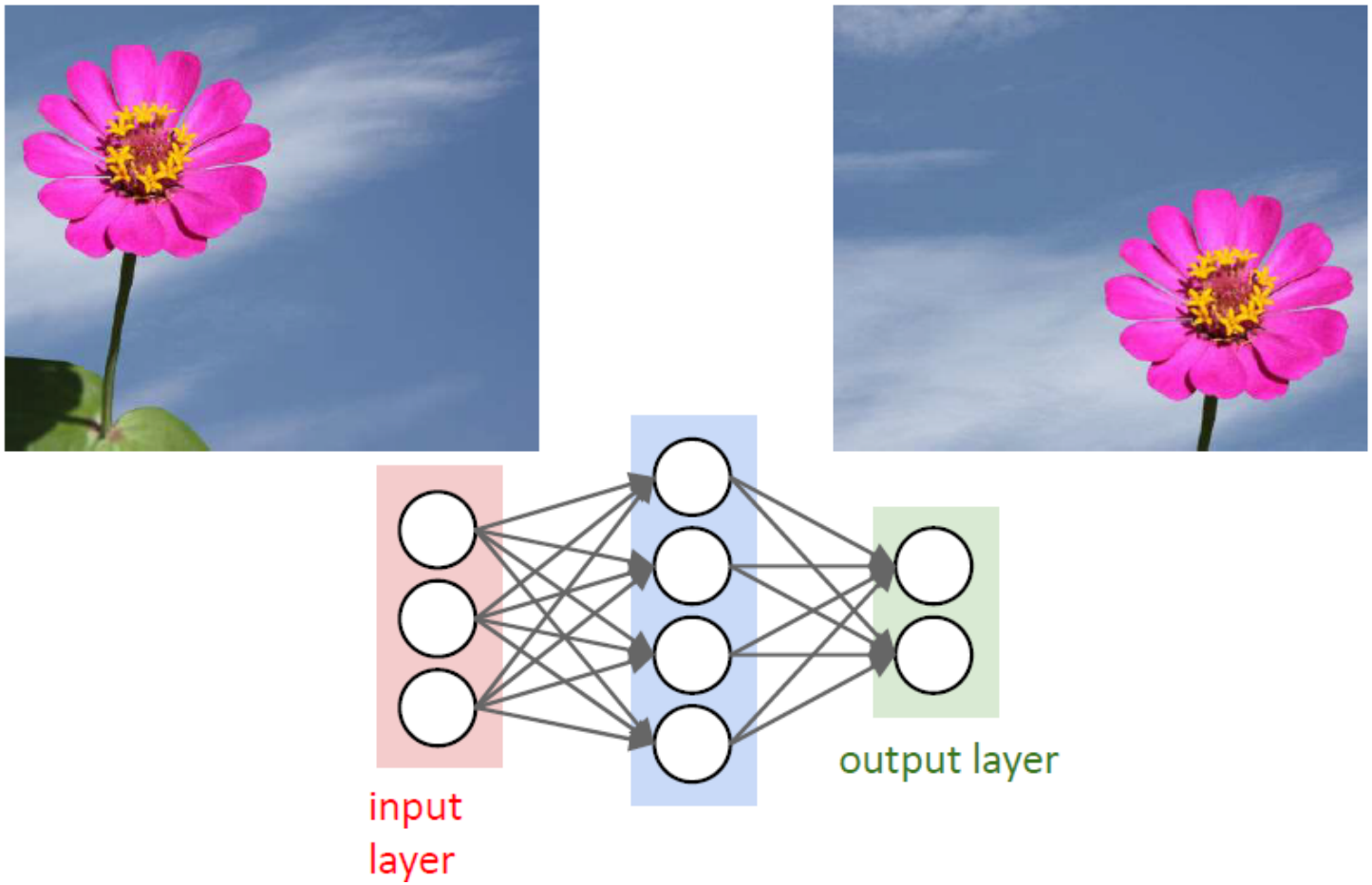
Finding a Welcome

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



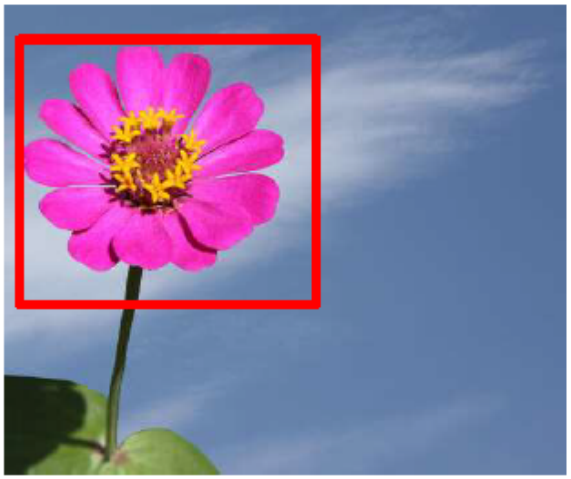
A problem

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



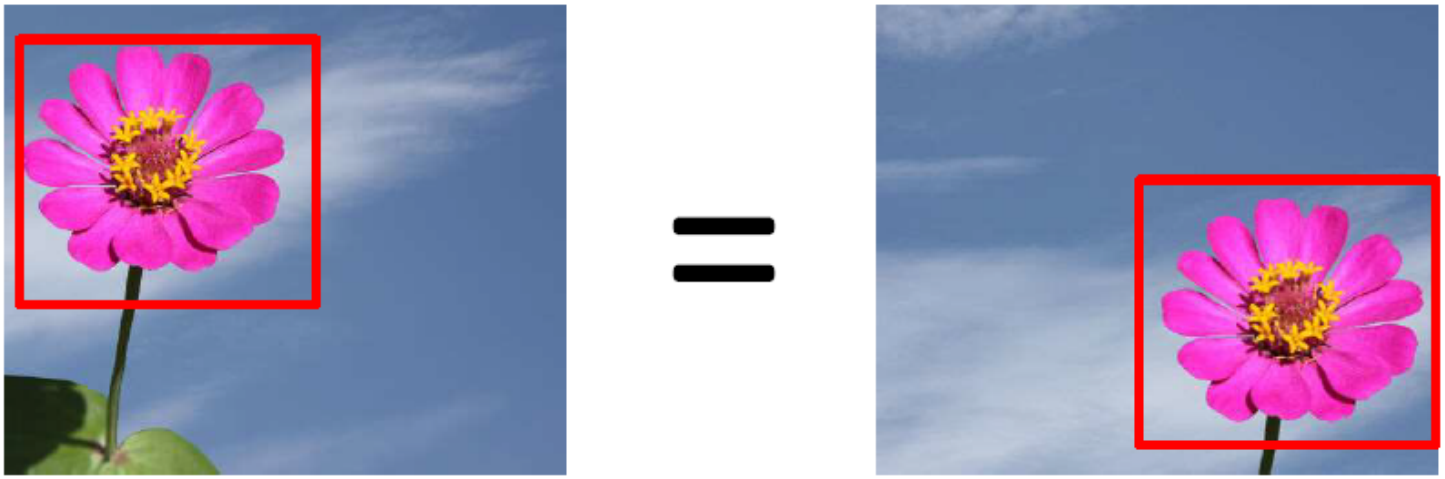
A problem

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



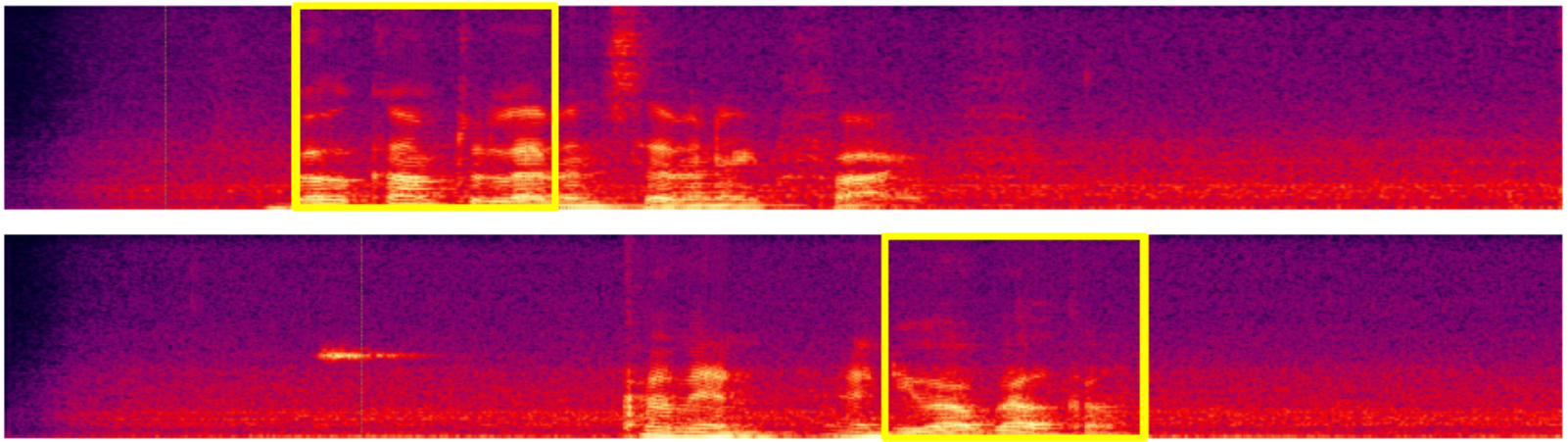
A problem

- 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**



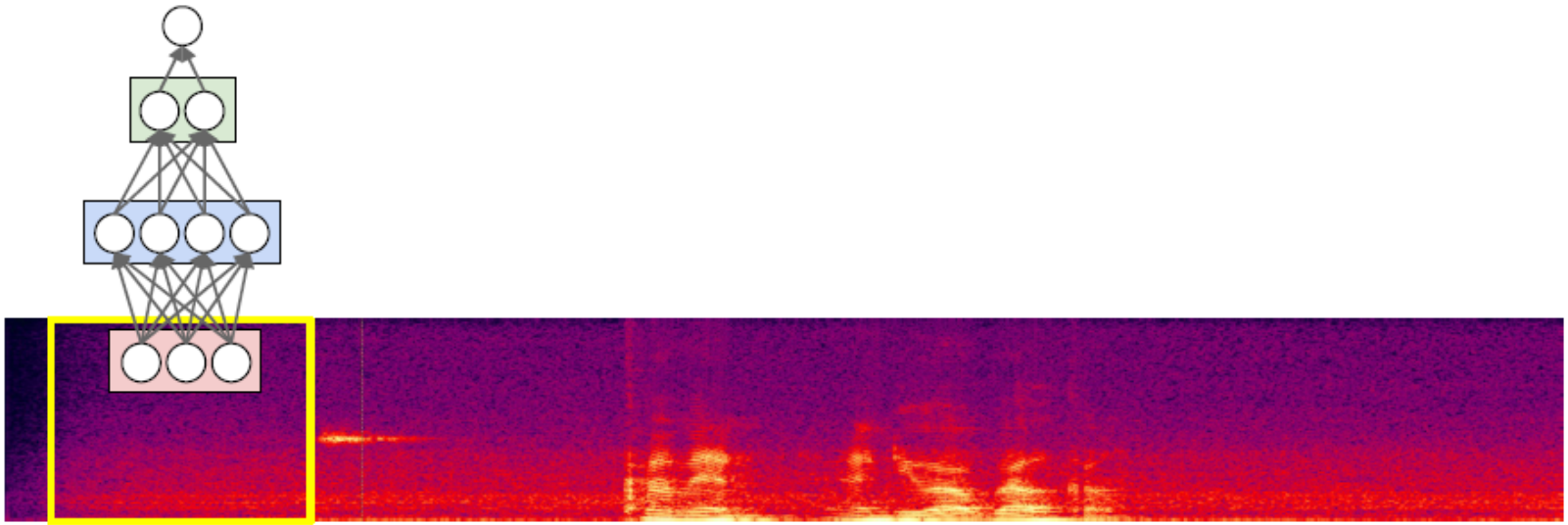
A problem

- 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**



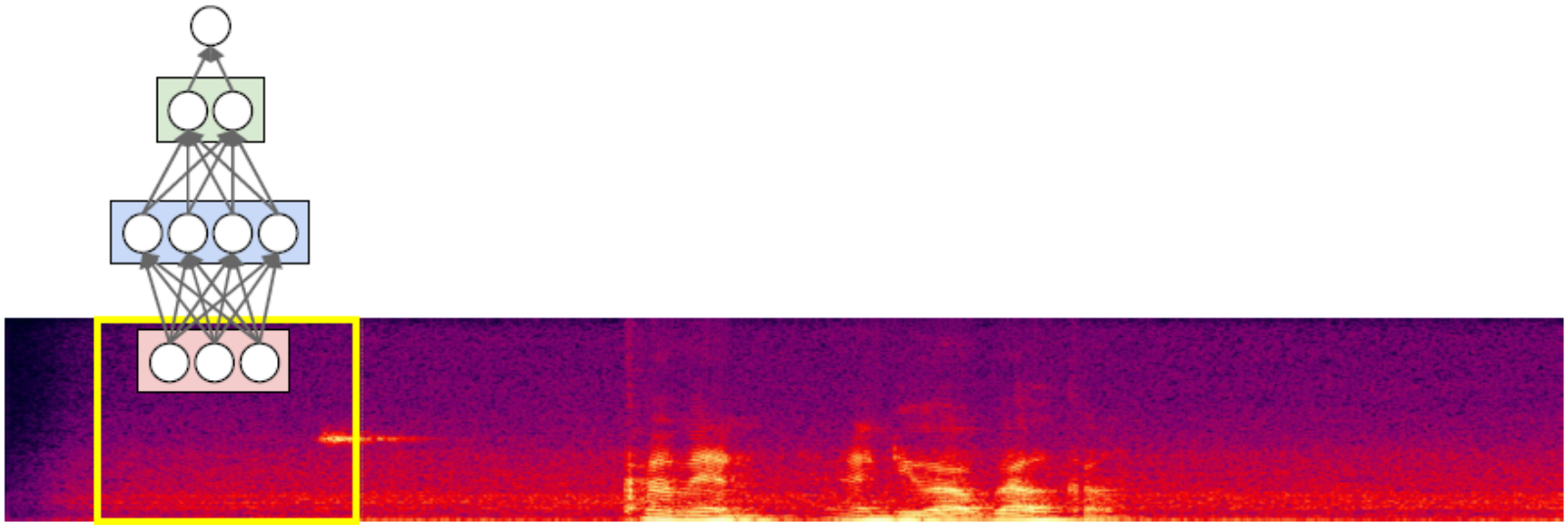
Solution: Scan

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



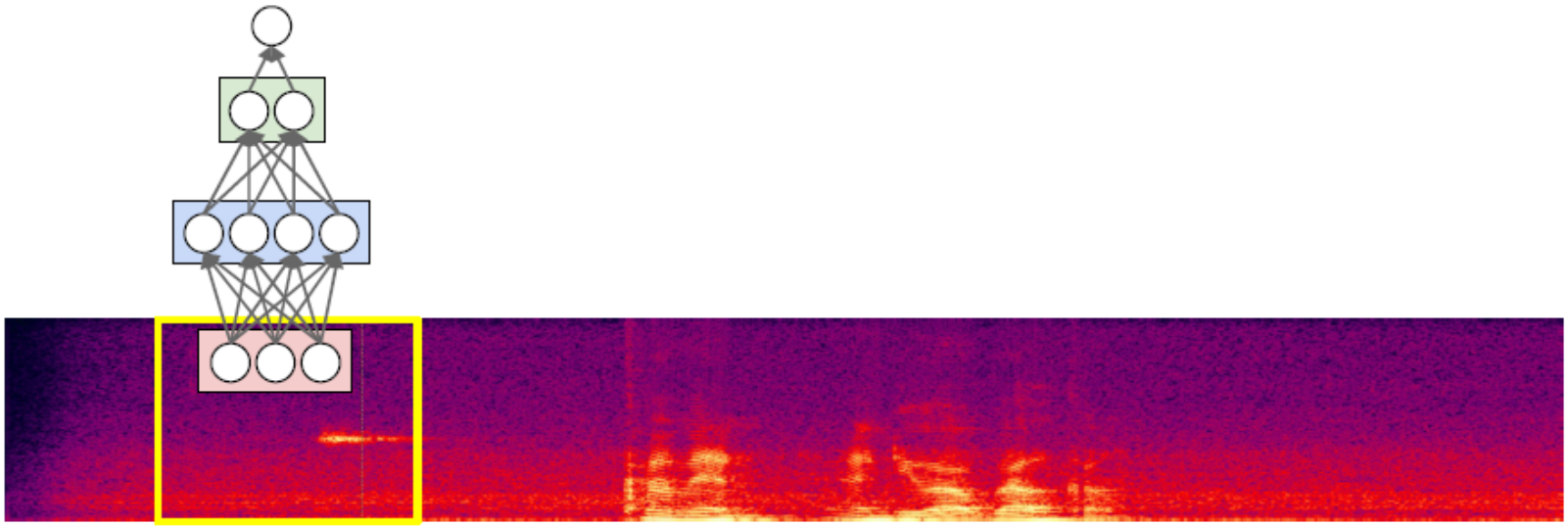
Solution: Scan

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



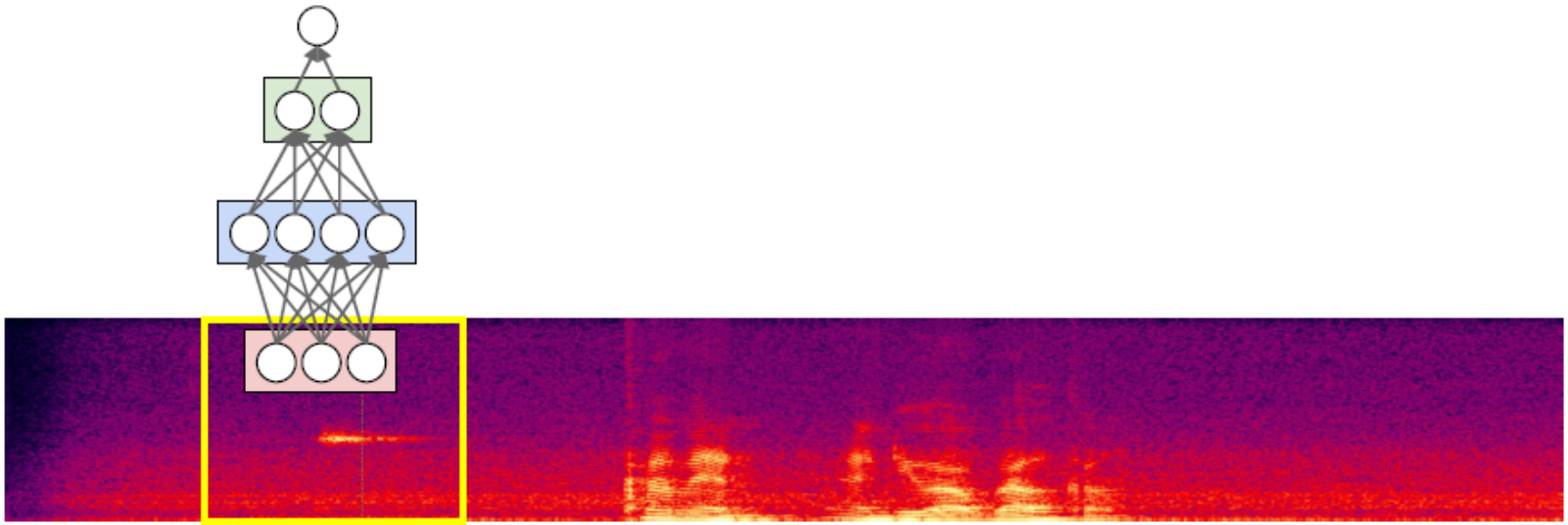
Solution: Scan

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



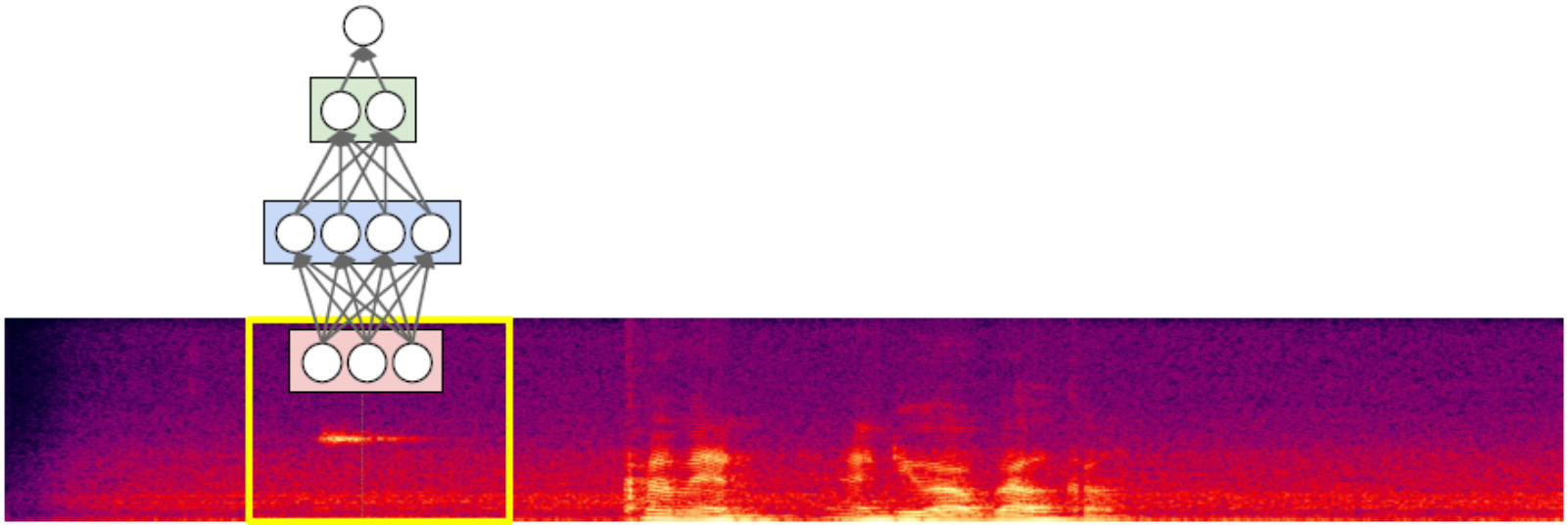
Solution: Scan

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



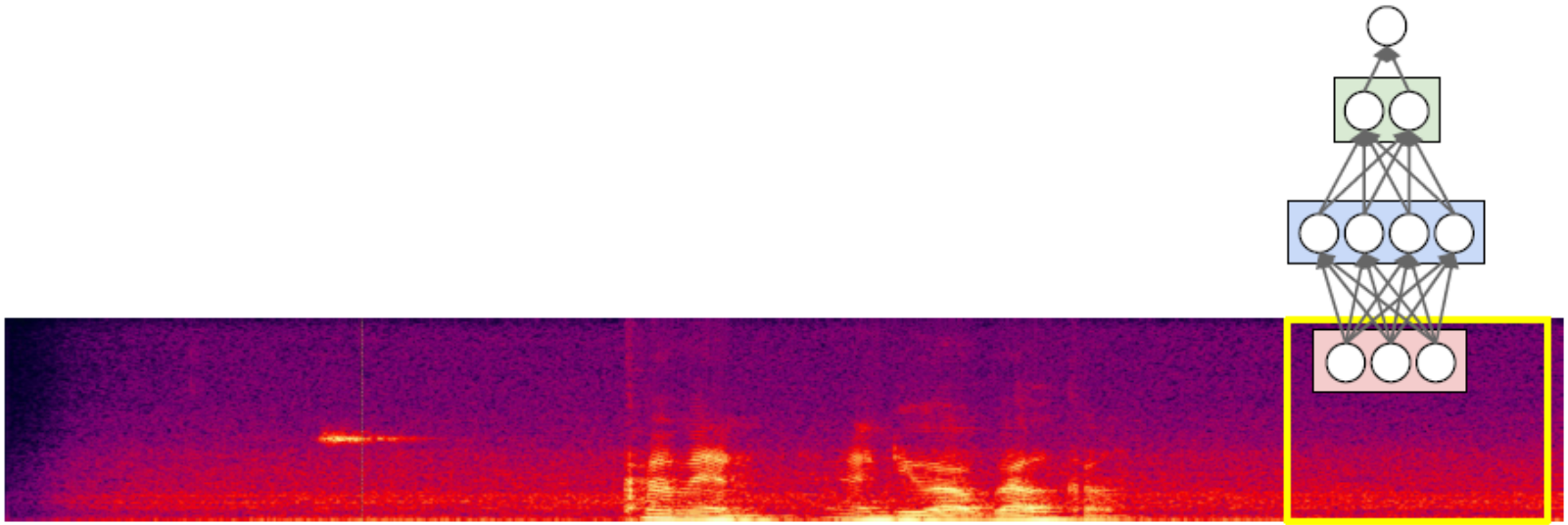
Solution: Scan

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



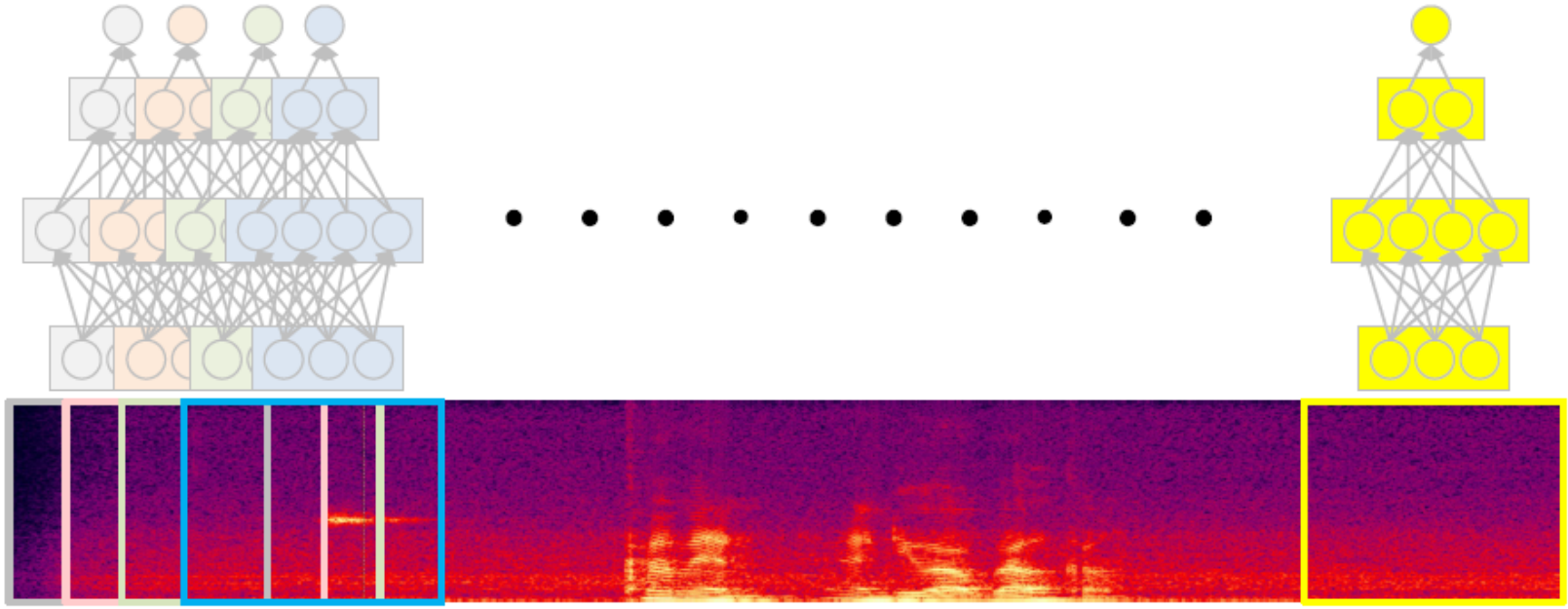
Solution: Scan

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



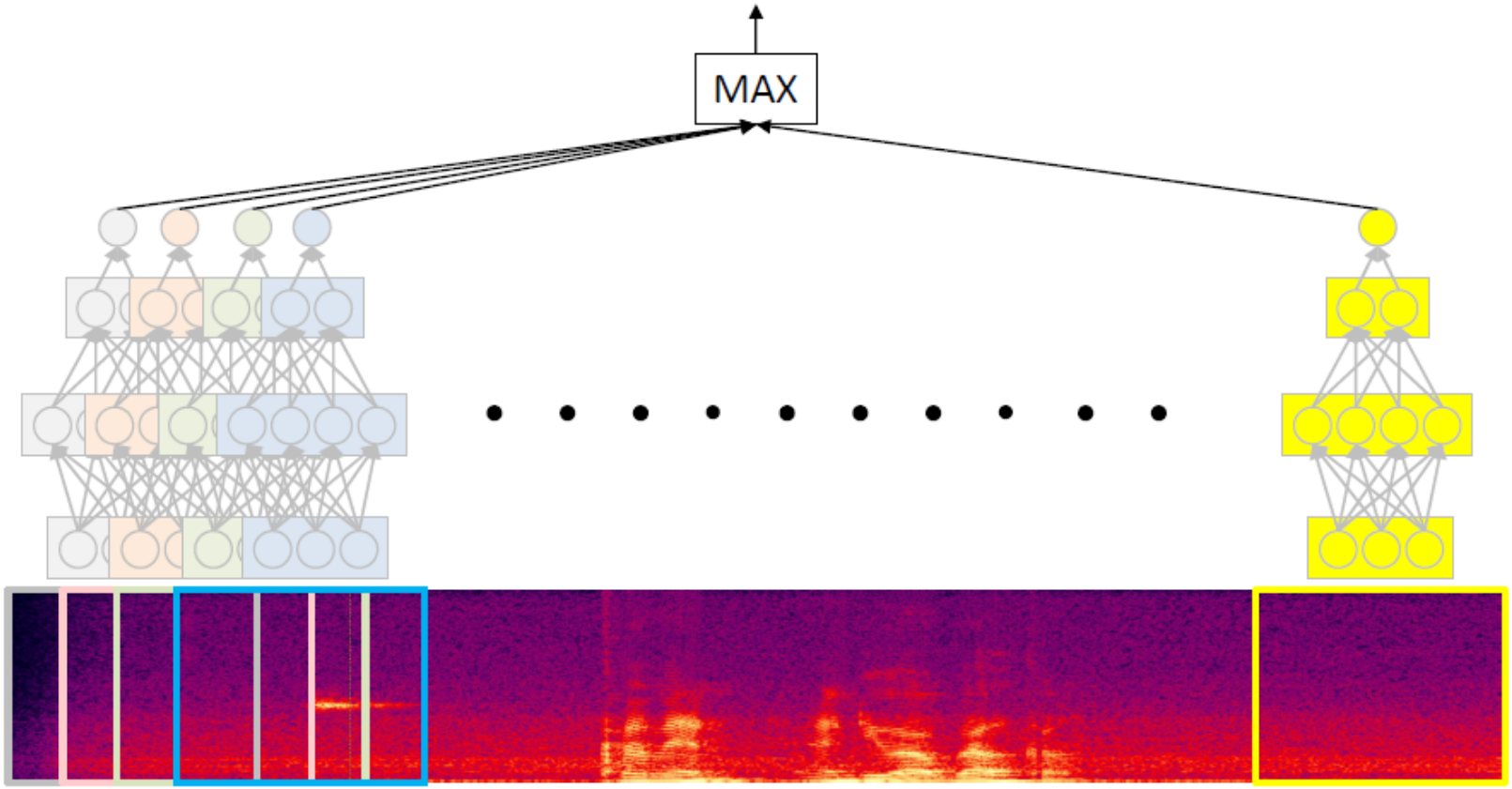
Solution: Scan

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



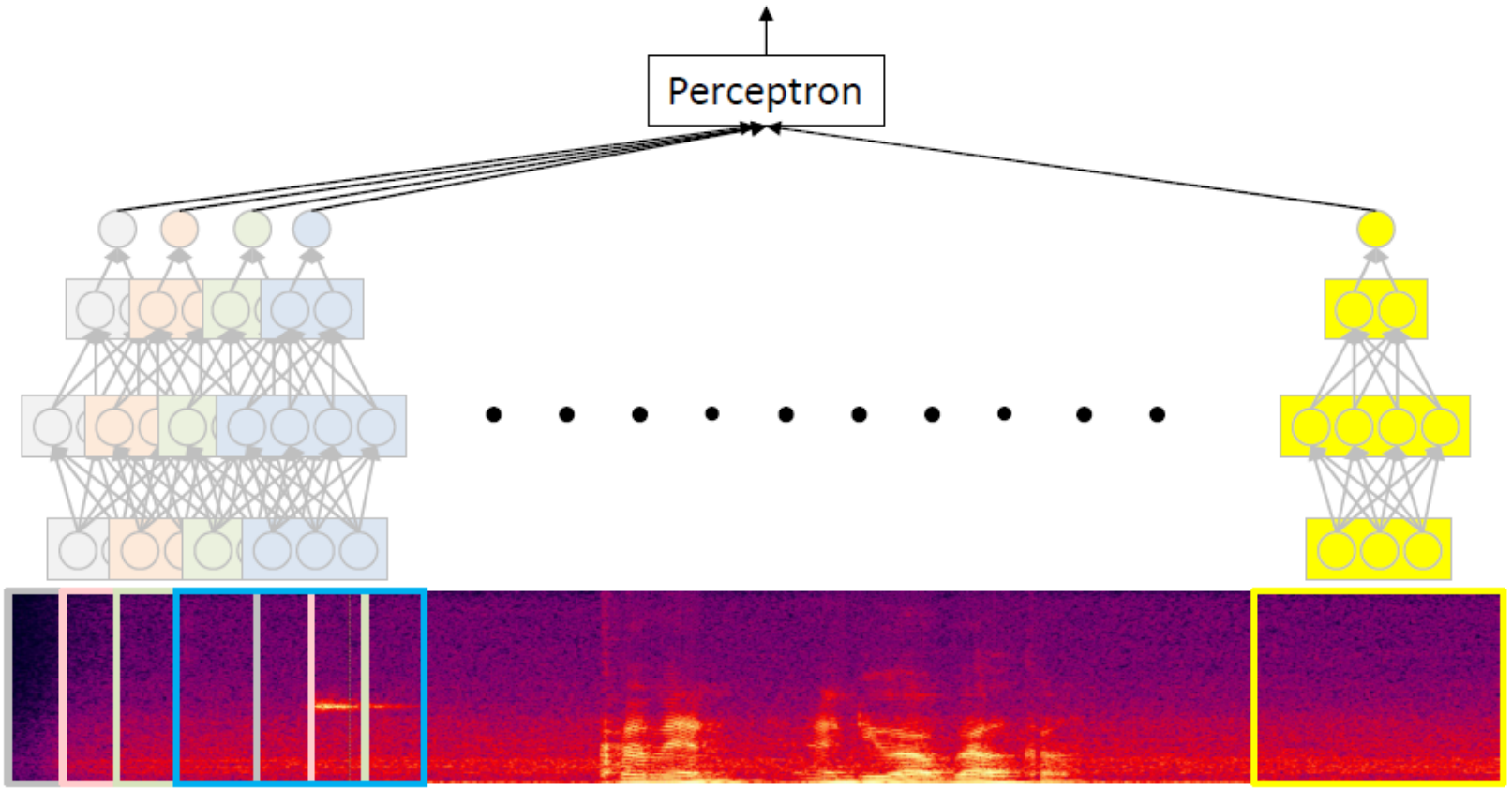
Solution: Scan

- "Does welcome occur in this recording?"
  - We have classified many "windows" individually
  - "Welcome" may have occurred in any of them
  - Maximum of all the outputs (Equivalent of Boolean OR)



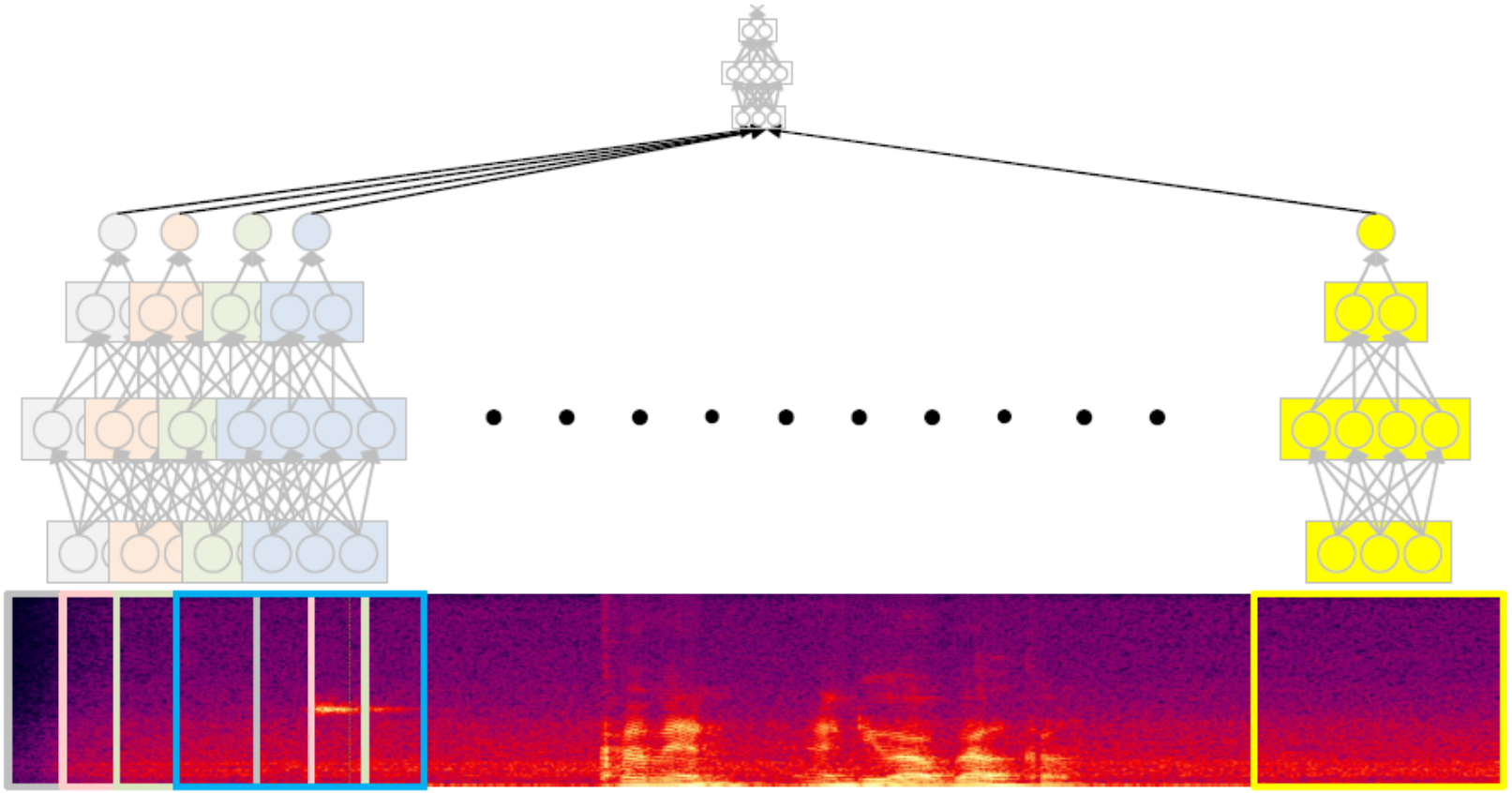
Solution: Scan

- "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
    - Adjacent windows can combine their evidence



Solution: Scan

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



Scanning with an MLP

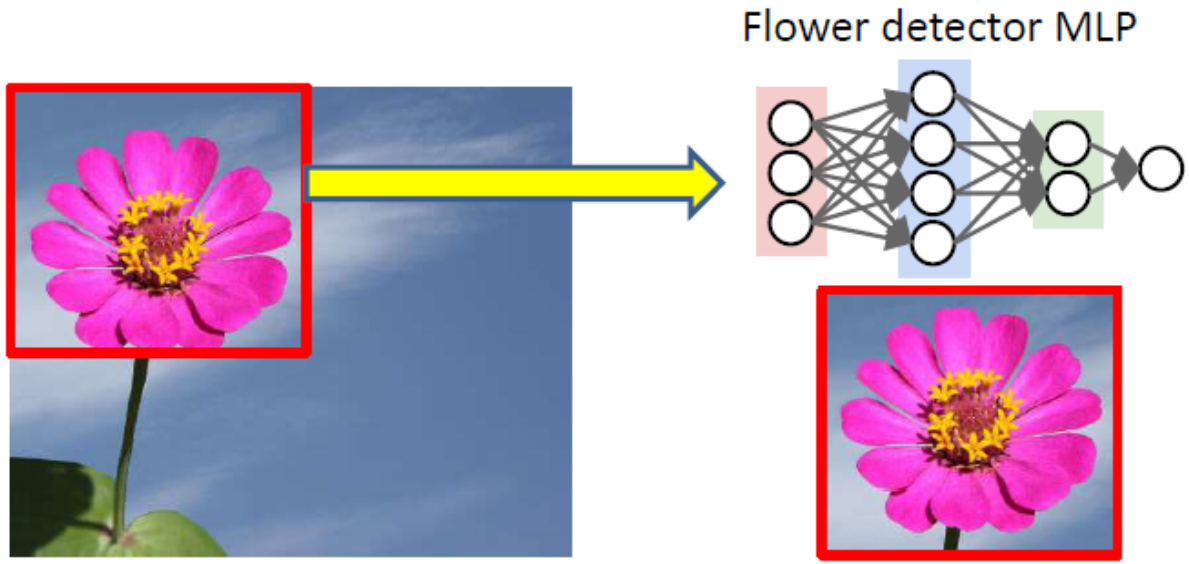
- K = width of "patch" evaluated by MLP

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

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

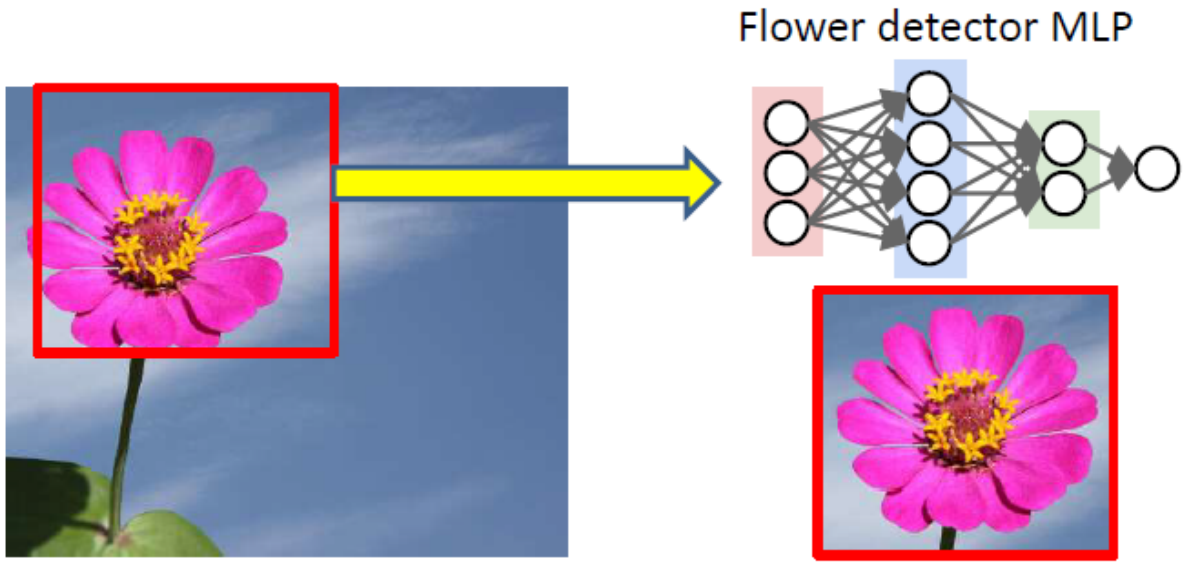
Scanning the picture to find a flower

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



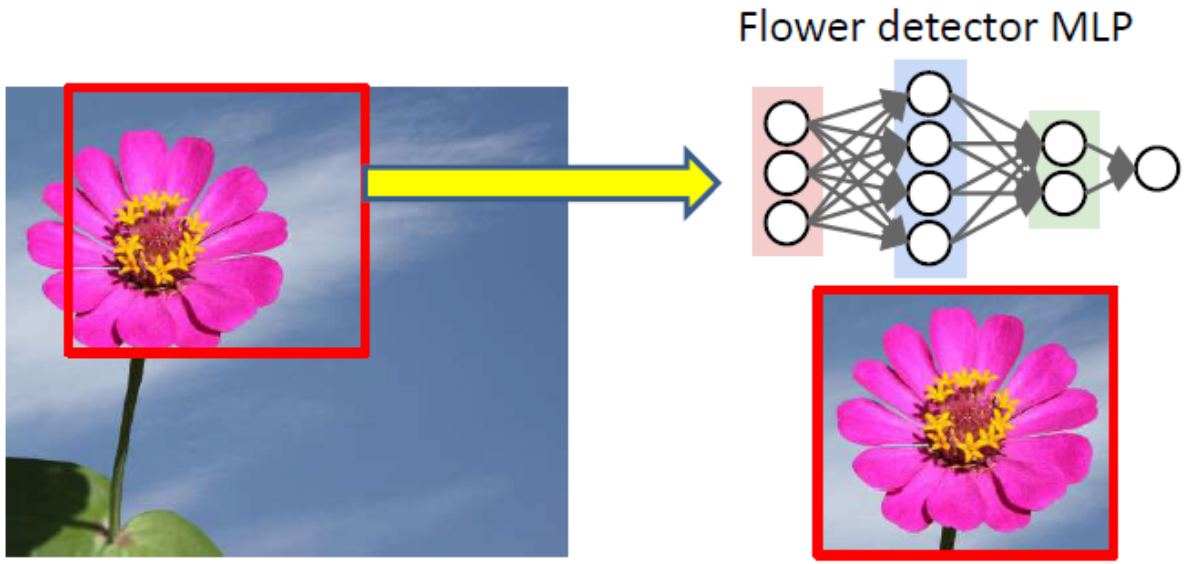
Scanning the picture to find a flower

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



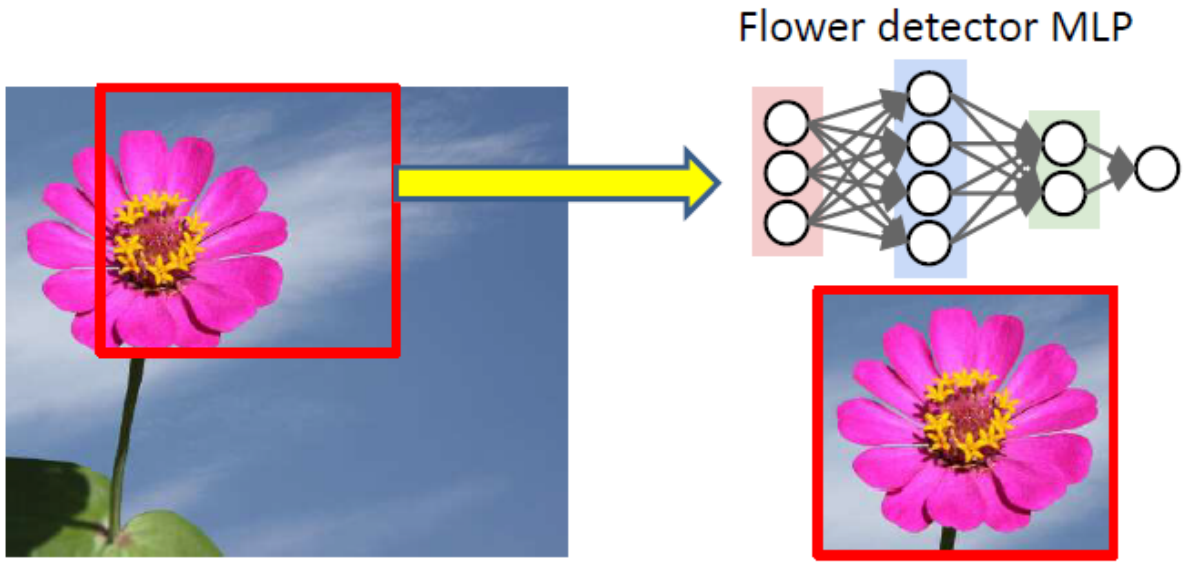
Scanning the picture to find a flower

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



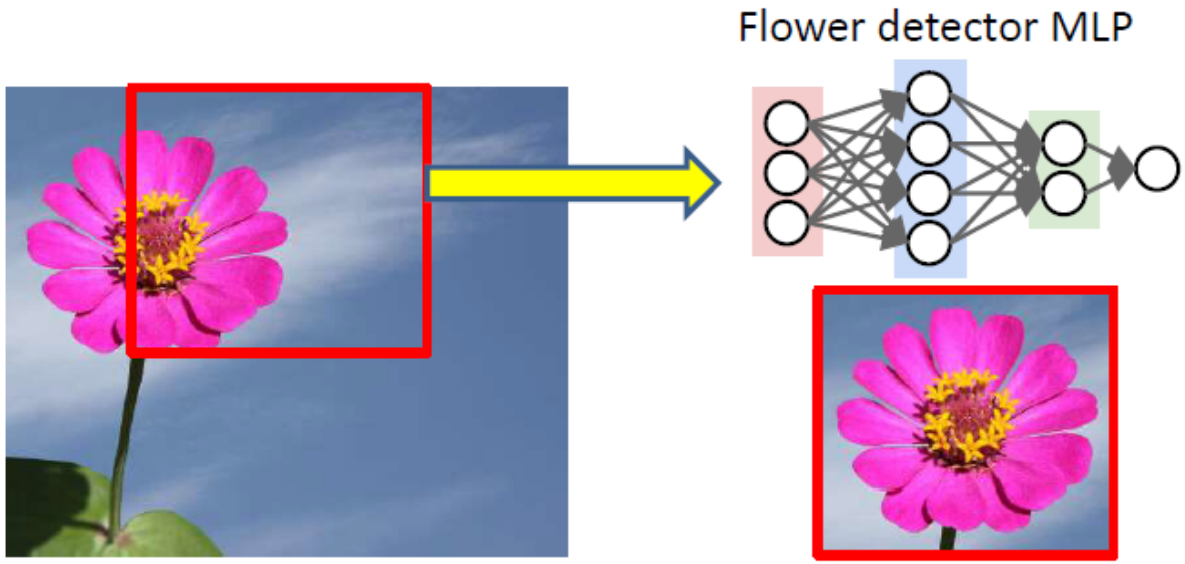
Scanning the picture to find a flower

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



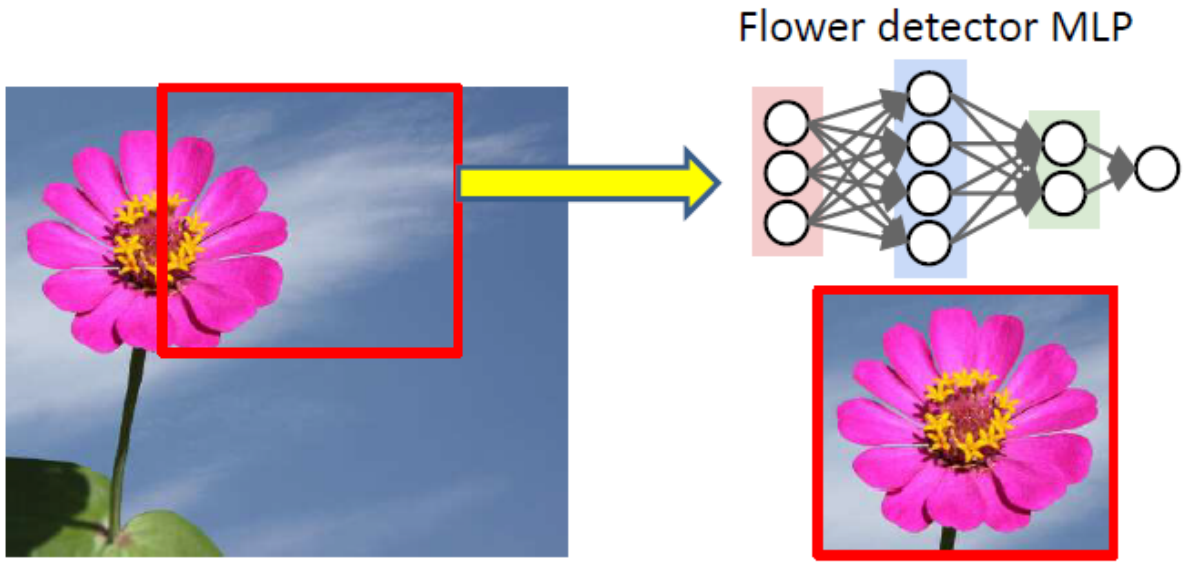
Scanning the picture to find a flower

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



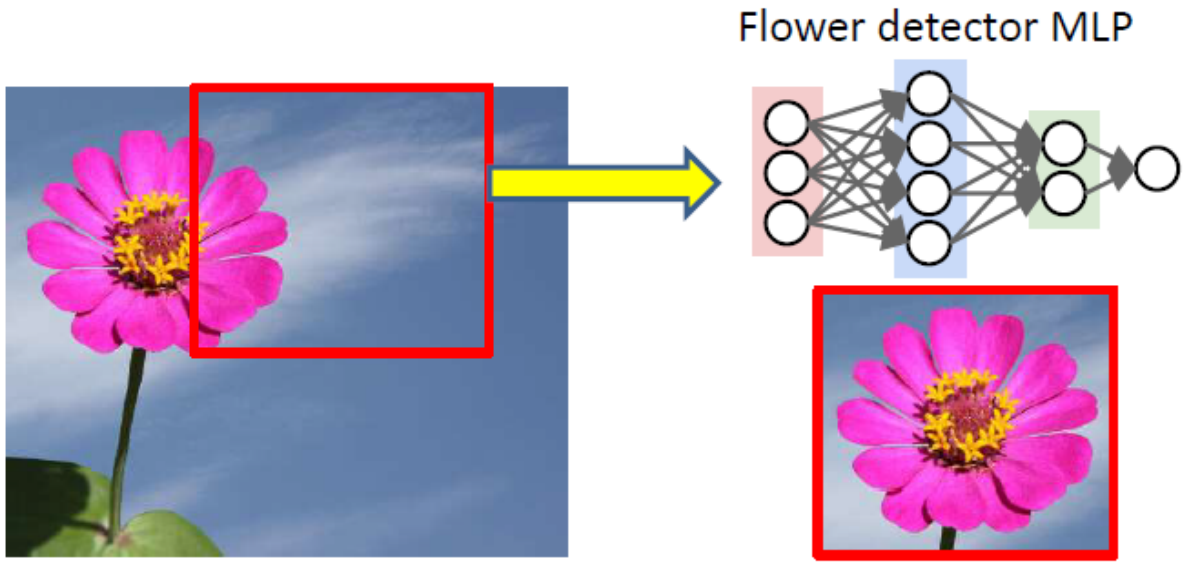
Scanning the picture to find a flower

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



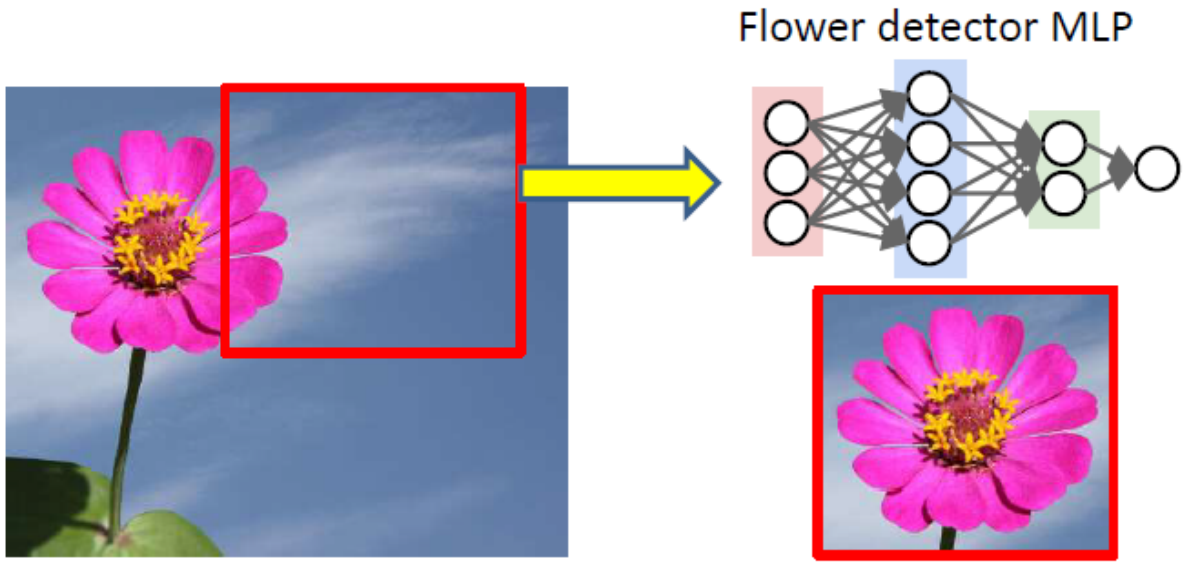
Scanning the picture to find a flower

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



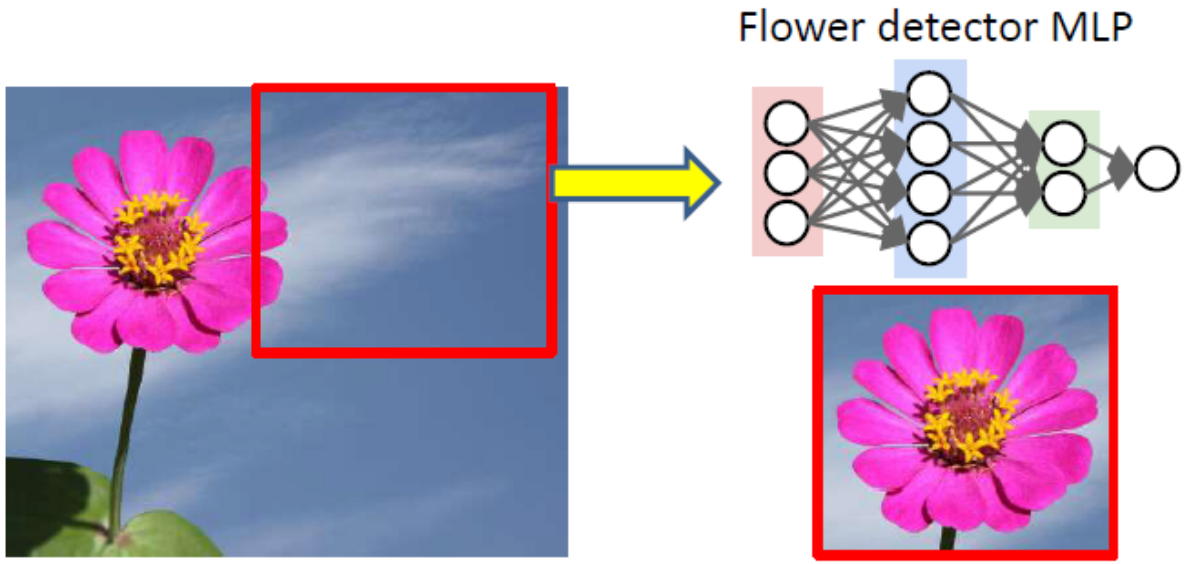
Scanning the picture to find a flower

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



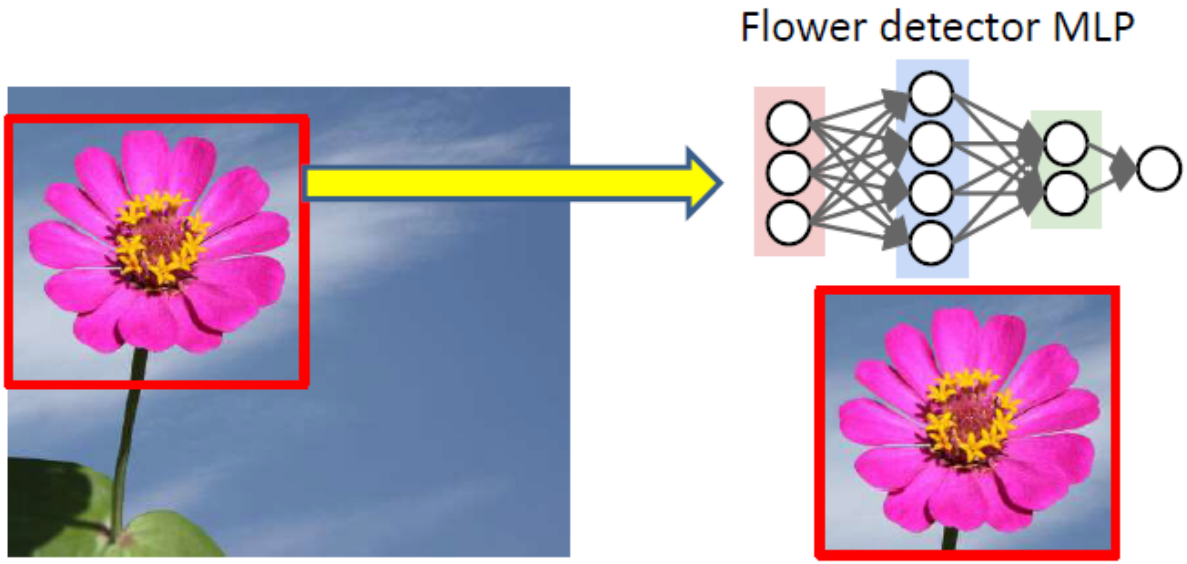
Scanning the picture to find a flower

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



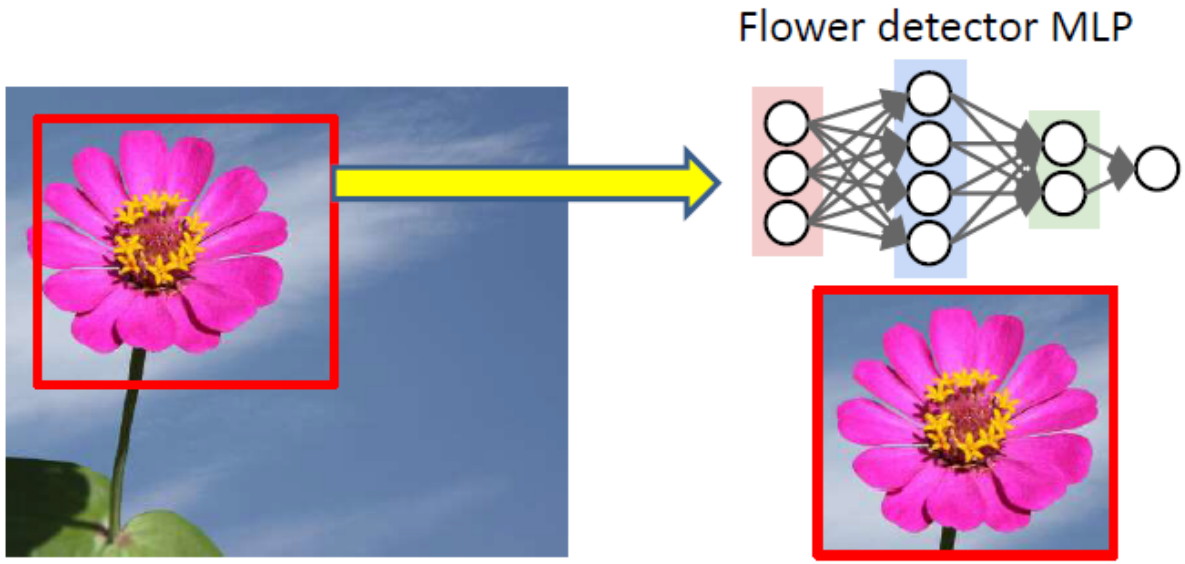
Scanning the picture to find a flower

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



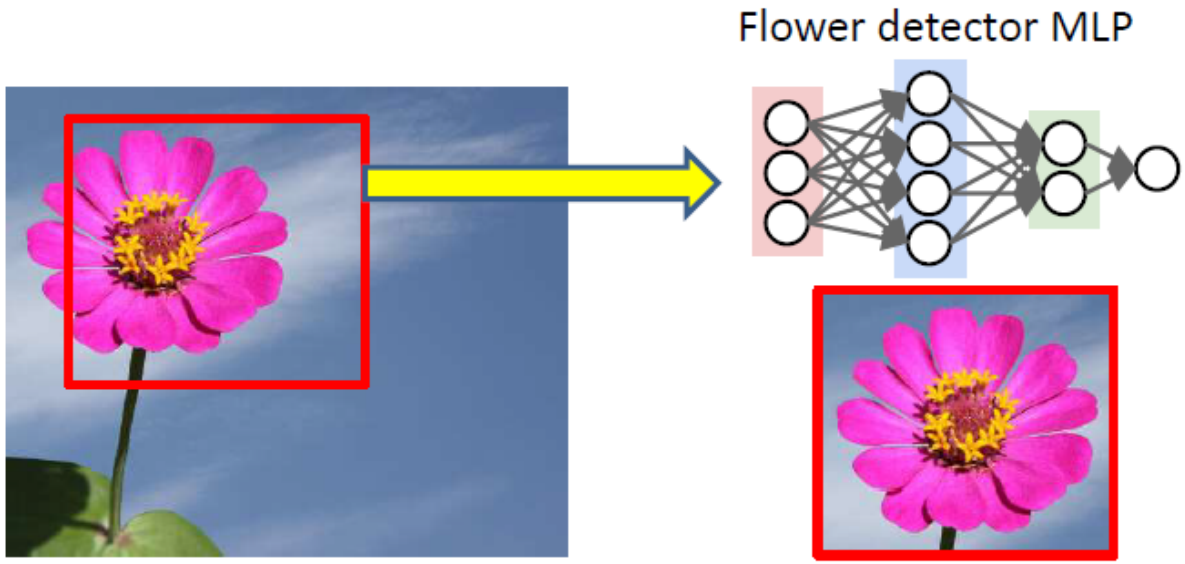
Scanning the picture to find a flower

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



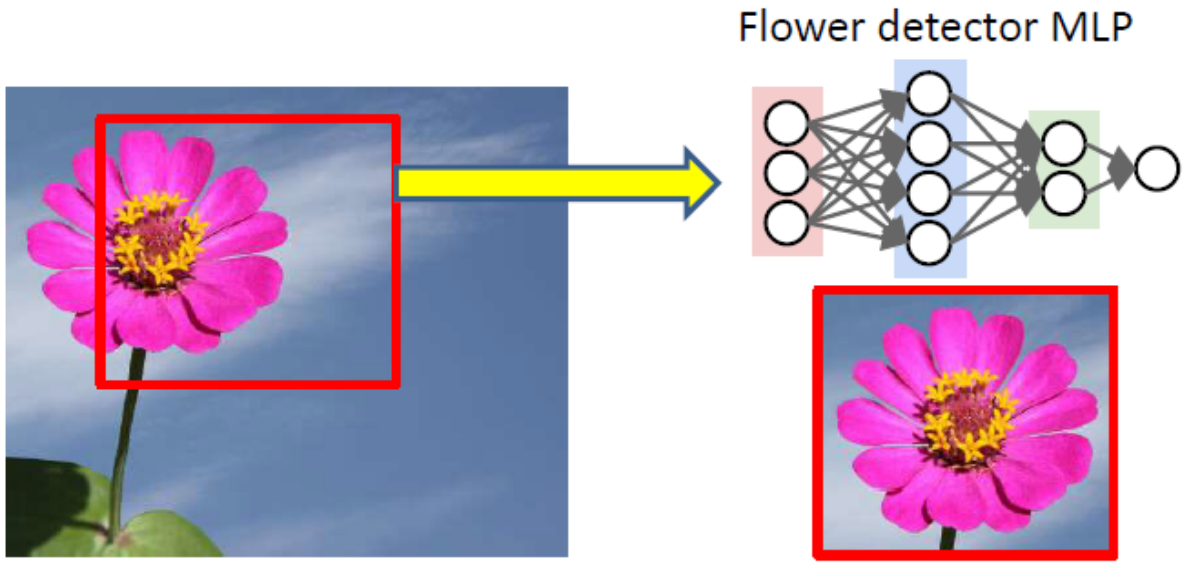
Scanning the picture to find a flower

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



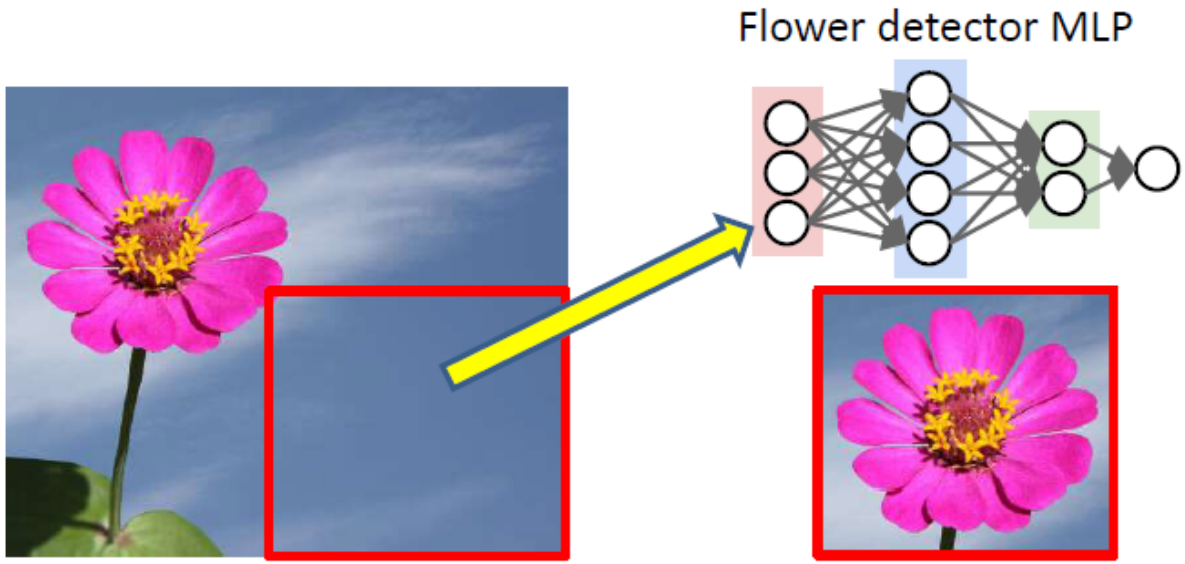
Scanning the picture to find a flower

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



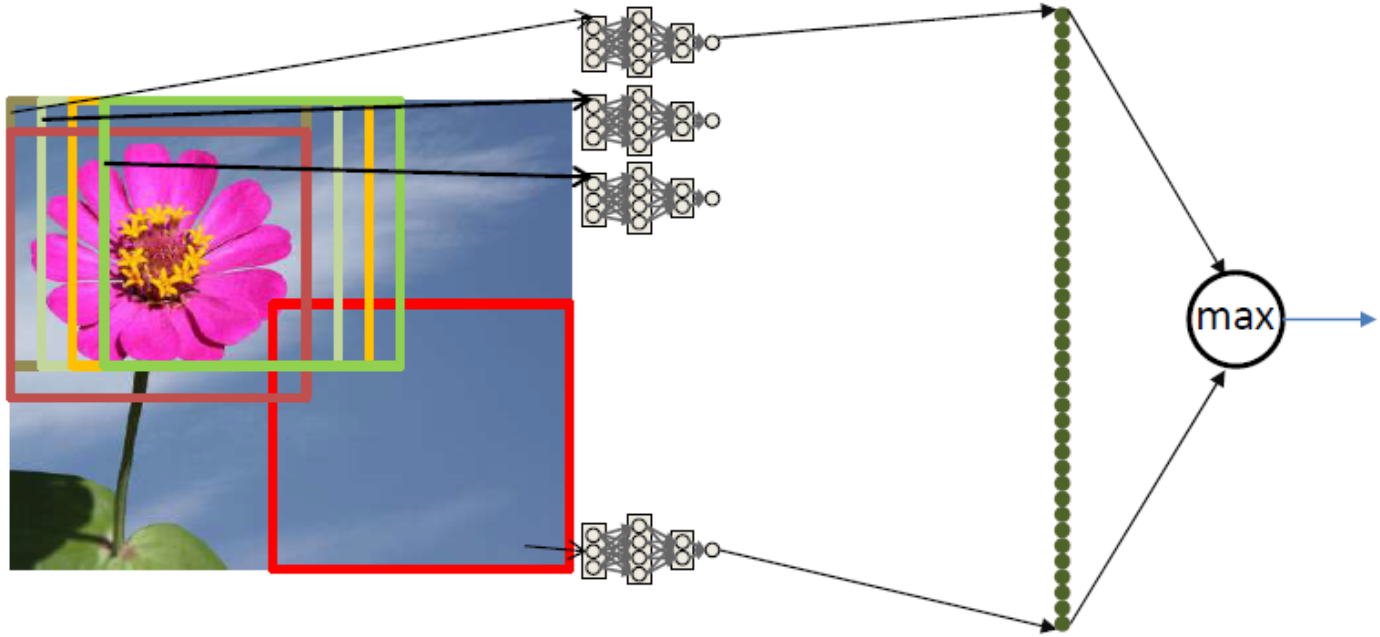
Scanning the picture to find a flower

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



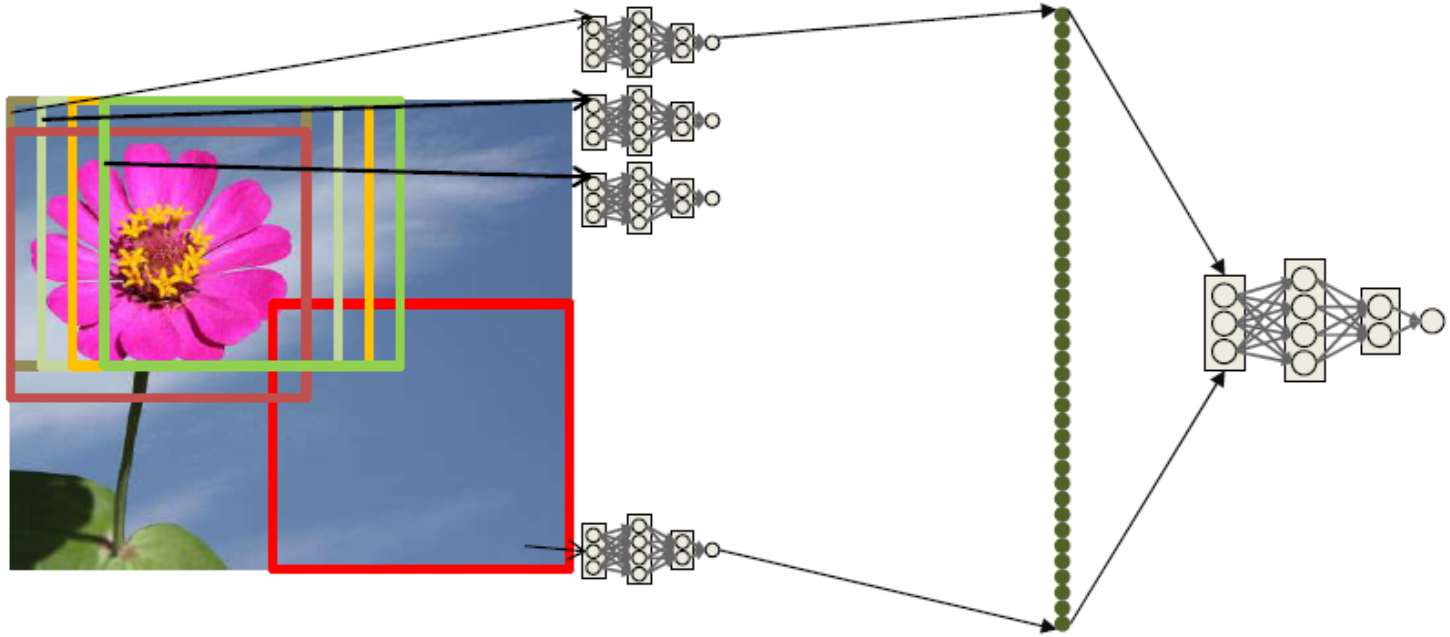
Scanning the picture to find a flower

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



Scanning the picture to find a flower

- Determine if any of the locations had a flower
  - 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
    - 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

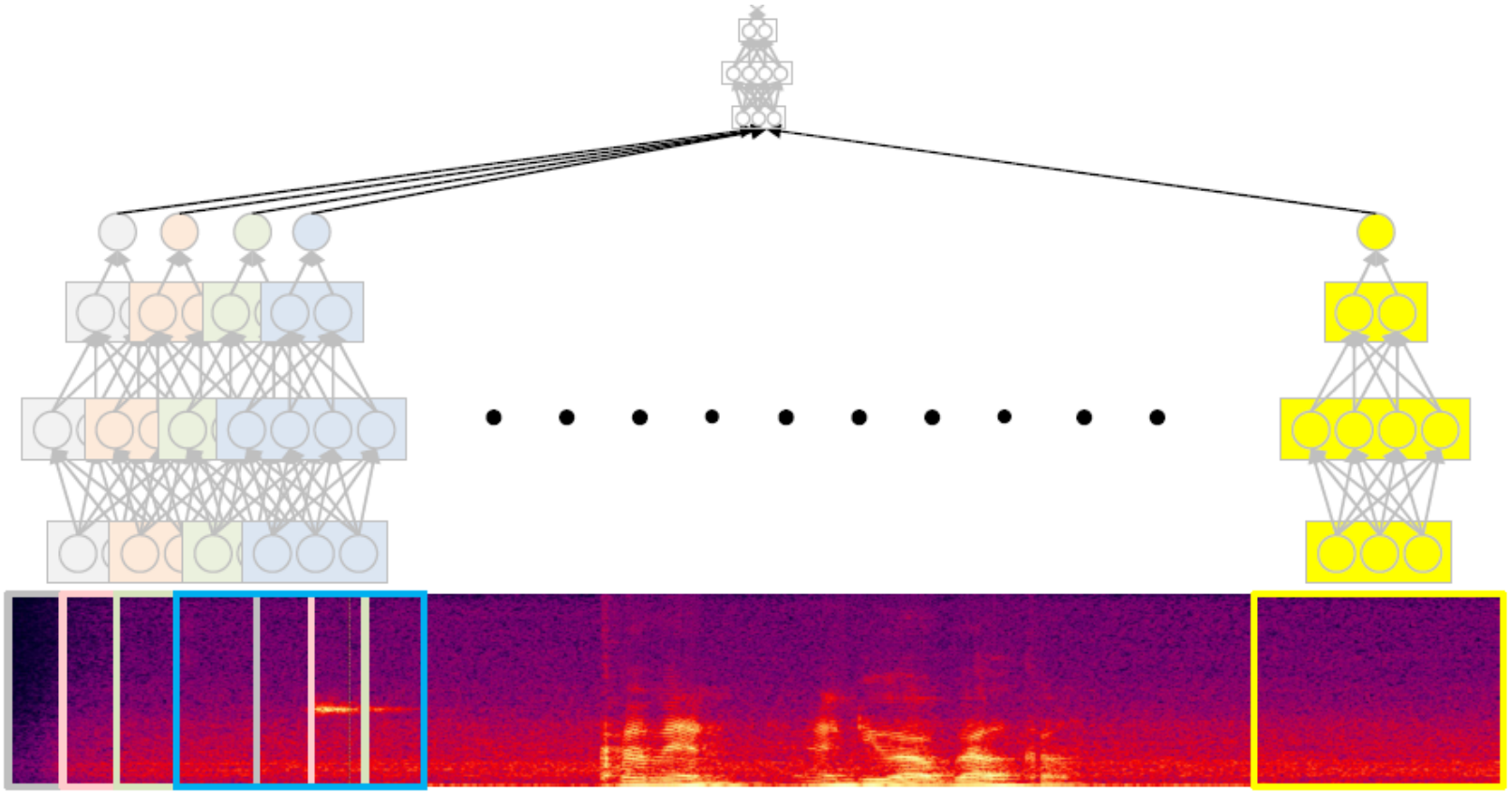
```
For i = 1:W-K+1
    For j = 1:H-K+1
        ImgSegment = Img(i:i+K-1, j:j+K-1)
        y(i,j) = MLP(ImgSegment)

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

Convolutional Neural Networks

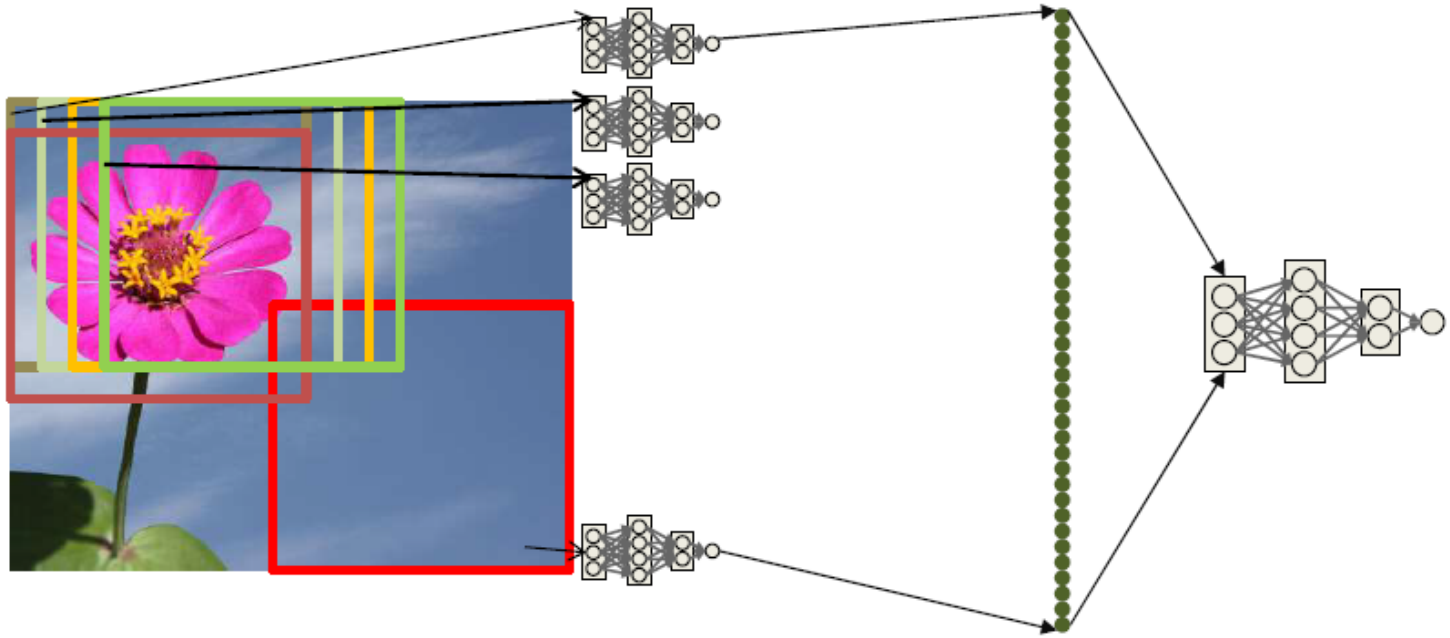
Its just a giant network with common subnets

- The entire operation can be viewed as one **giant network**
  - With many subnetworks, one per window
  - Restriction: **All subnets are identical and have an identical set of parameters**
    - This is what we call a shared parameter network
- The network is shift-invariant!



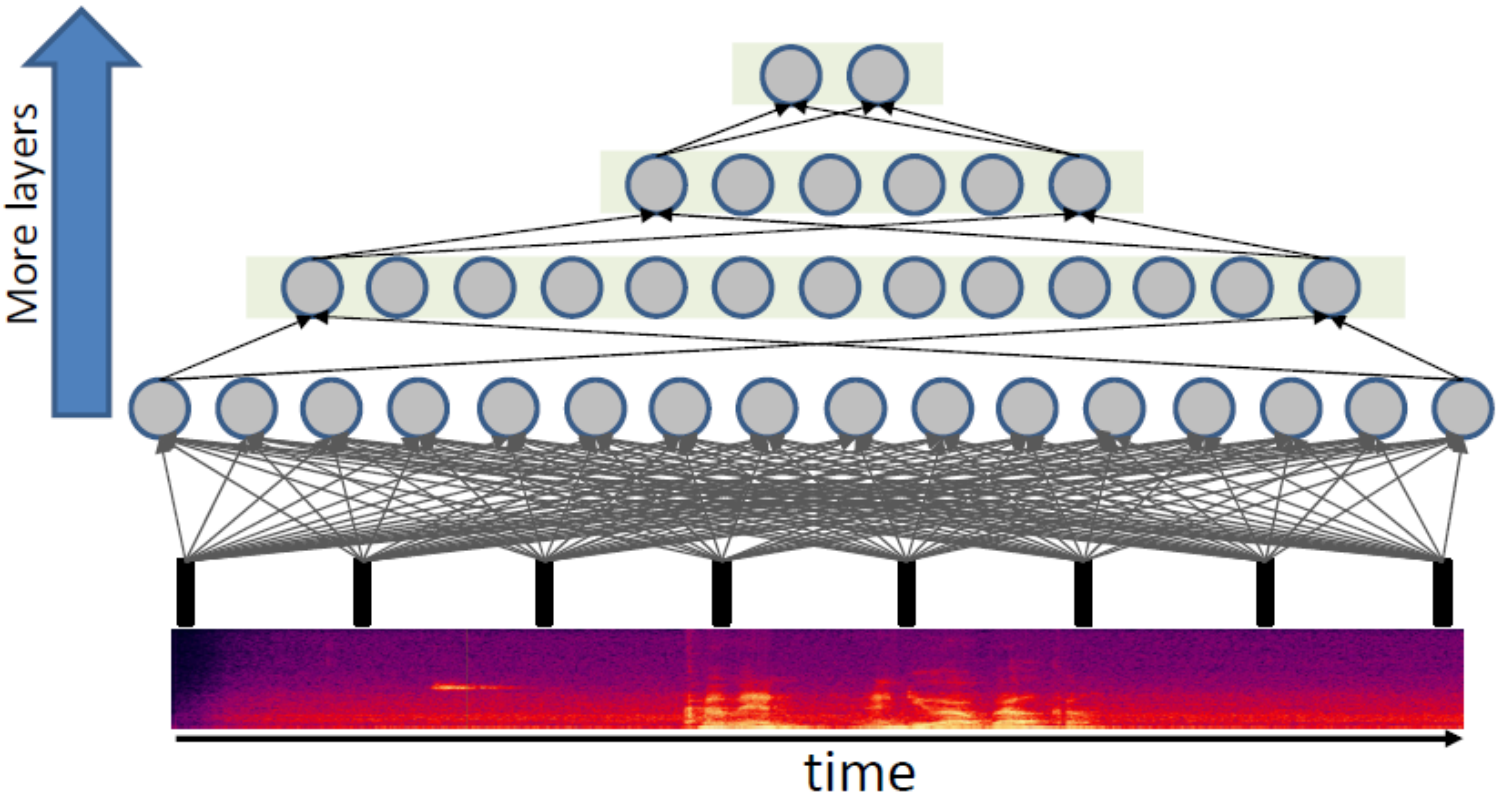
Its just a giant network with common subnets

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



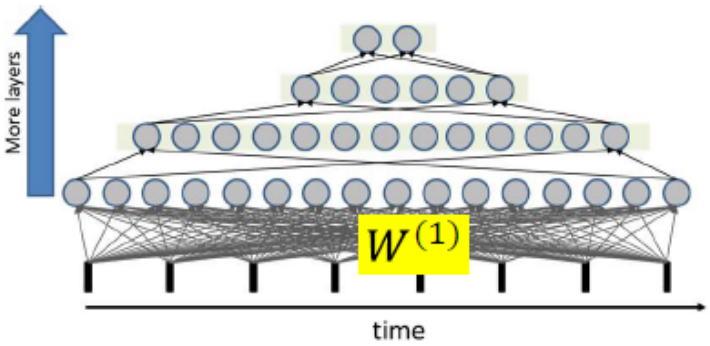
Regular networks vs. scanning networks

- 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

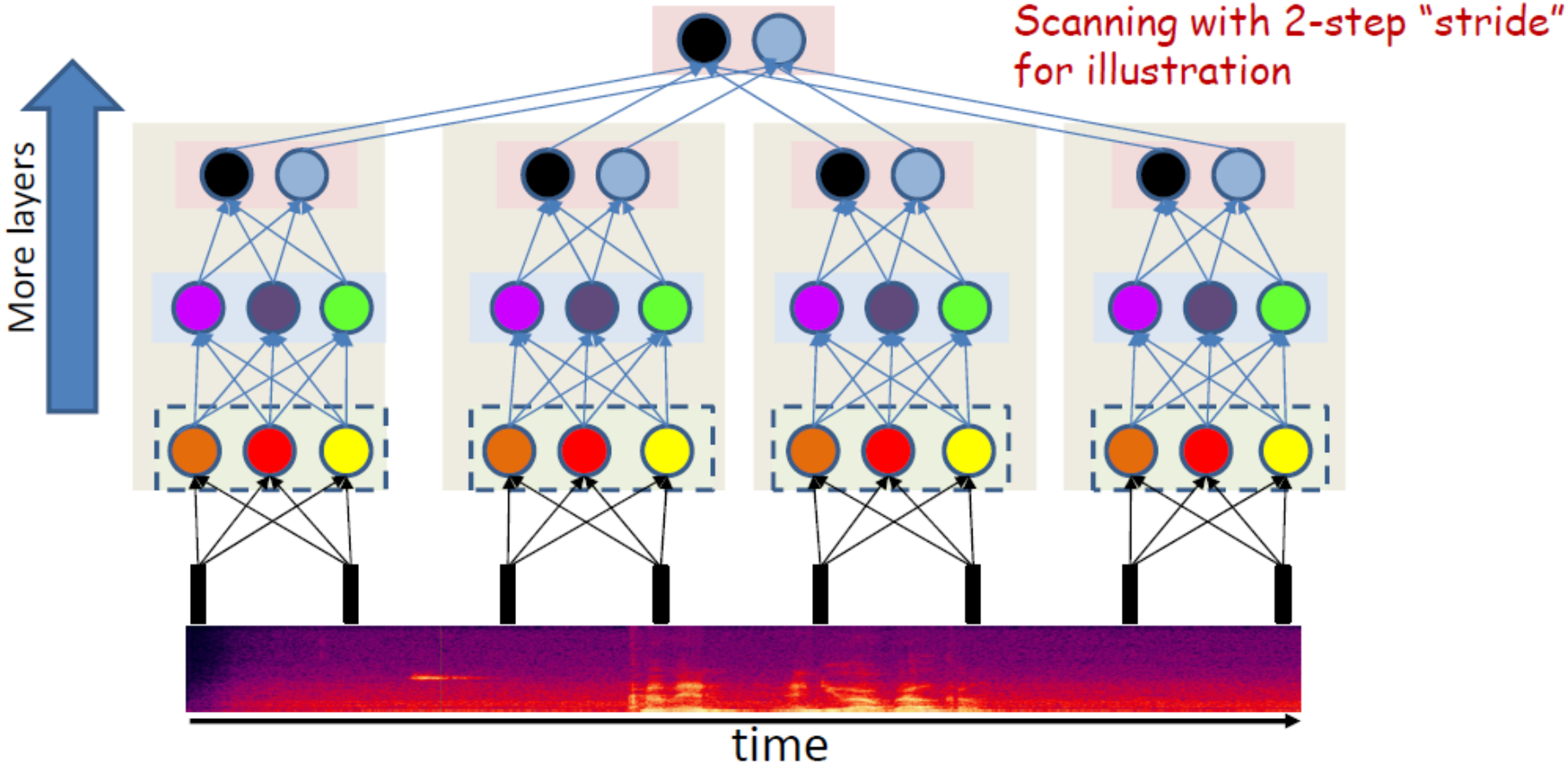
- Consider the first layer
  - Assume N inputs and M outputs
- The weights matrix is a full $M \times N$ matrix
  - Requiring MN unique parameters



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

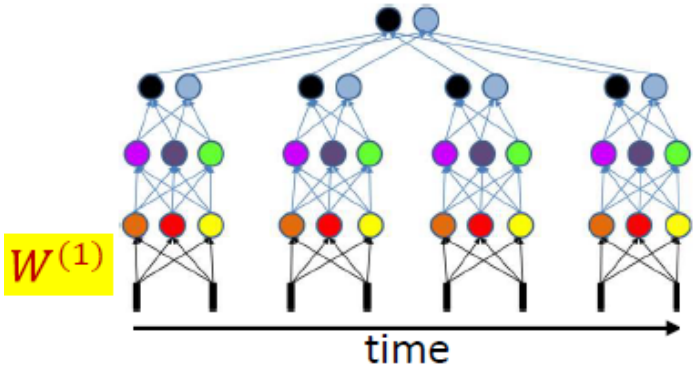
Scanning networks

- 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

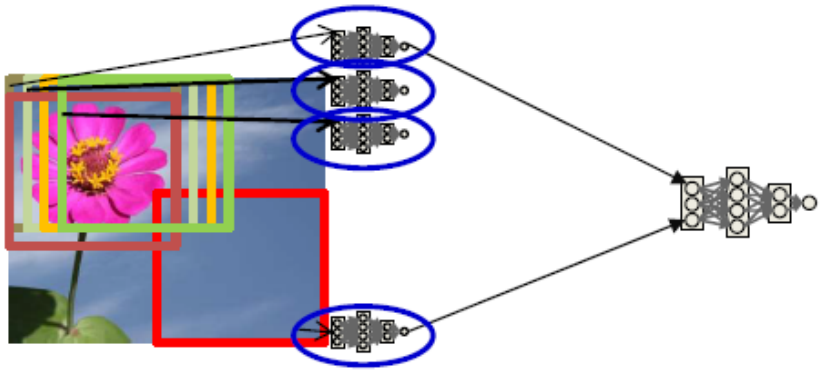
- In a scanning MLP each neuron is connected to a subset of neurons in the previous layer
  - The weights matrix is sparse
  - The weights matrix is block structured with identical blocks
  - **The network is a shared parameter model**
  - **Also, far fewer parameters**

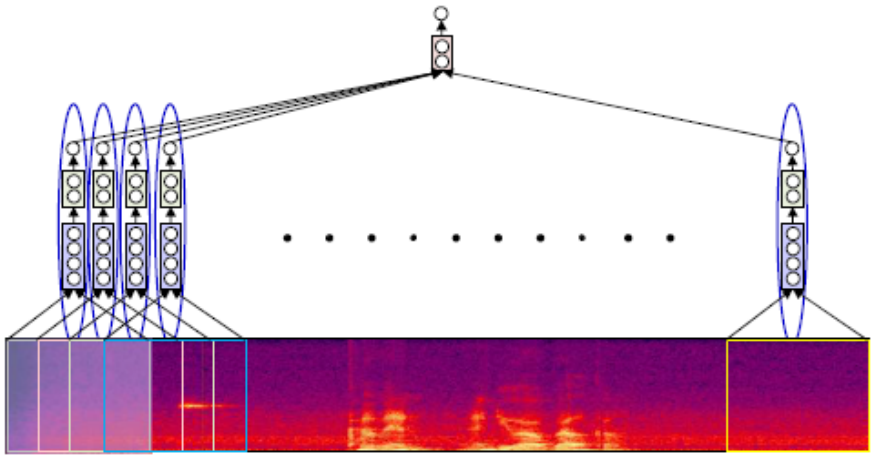


$$W^{(1)} = \begin{bmatrix} w_{11} & w_{12} & 0 & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ w_{21} & w_{22} & 0 & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ w_{31} & w_{32} & 0 & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & 0 & w_{11} & w_{12} & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & w_{21} & w_{22} & 0 & 0 & 0 & \vdots & \vdots \\ 0 & 0 & w_{31} & w_{32} & 0 & 0 & 0 & \vdots & \vdots \\ 0 & 0 & 0 & 0 & w_{11} & w_{12} & 0 & 0 & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & w_{31} & w_{32} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \\ . \\ . \\ X_N \end{bmatrix}$$

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





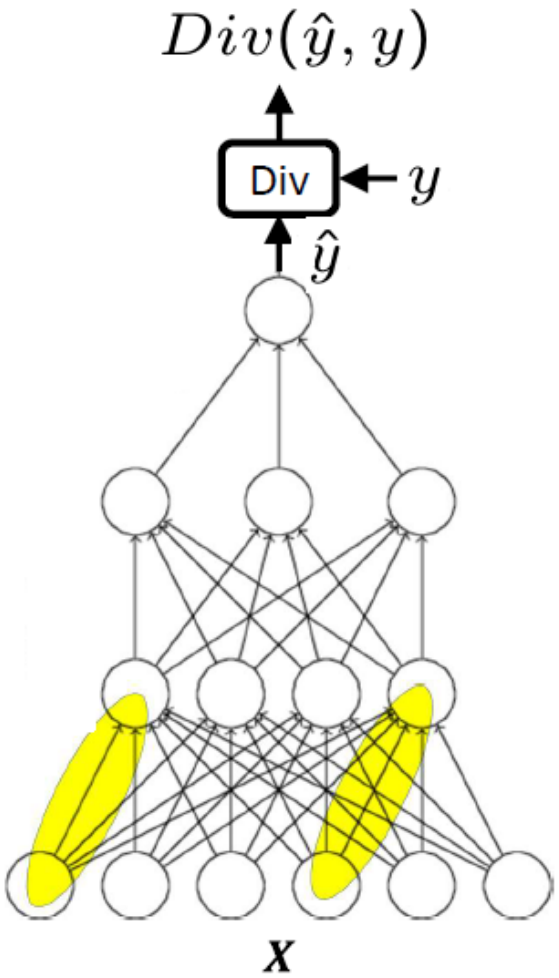
Learning in shared parameter networks

- Consider a simple network with shared weights

$$w_{ij}^{k} = w_{mn}^{\ell} = w^{s}$$

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

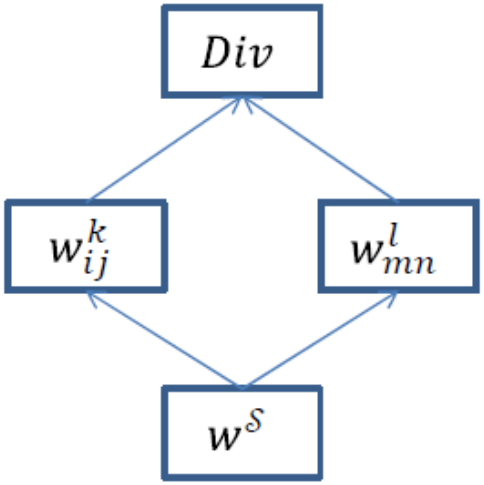
- For any training instance X, a small perturbation of w^{s} perturbs both w_{ij}^{k} and w_{mn}^{ℓ} identically
  - Each of these perturbations will individually influence the loss $Div(\hat{y}, y)$



Computing the divergence of shared parameters

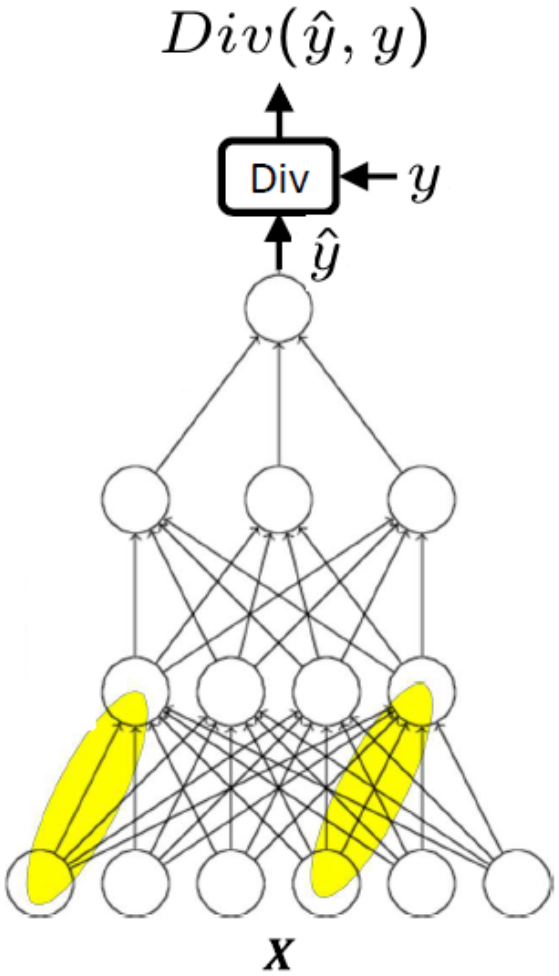
- Each of the individual terms can be computed via backpropagation

Influence diagram



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

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

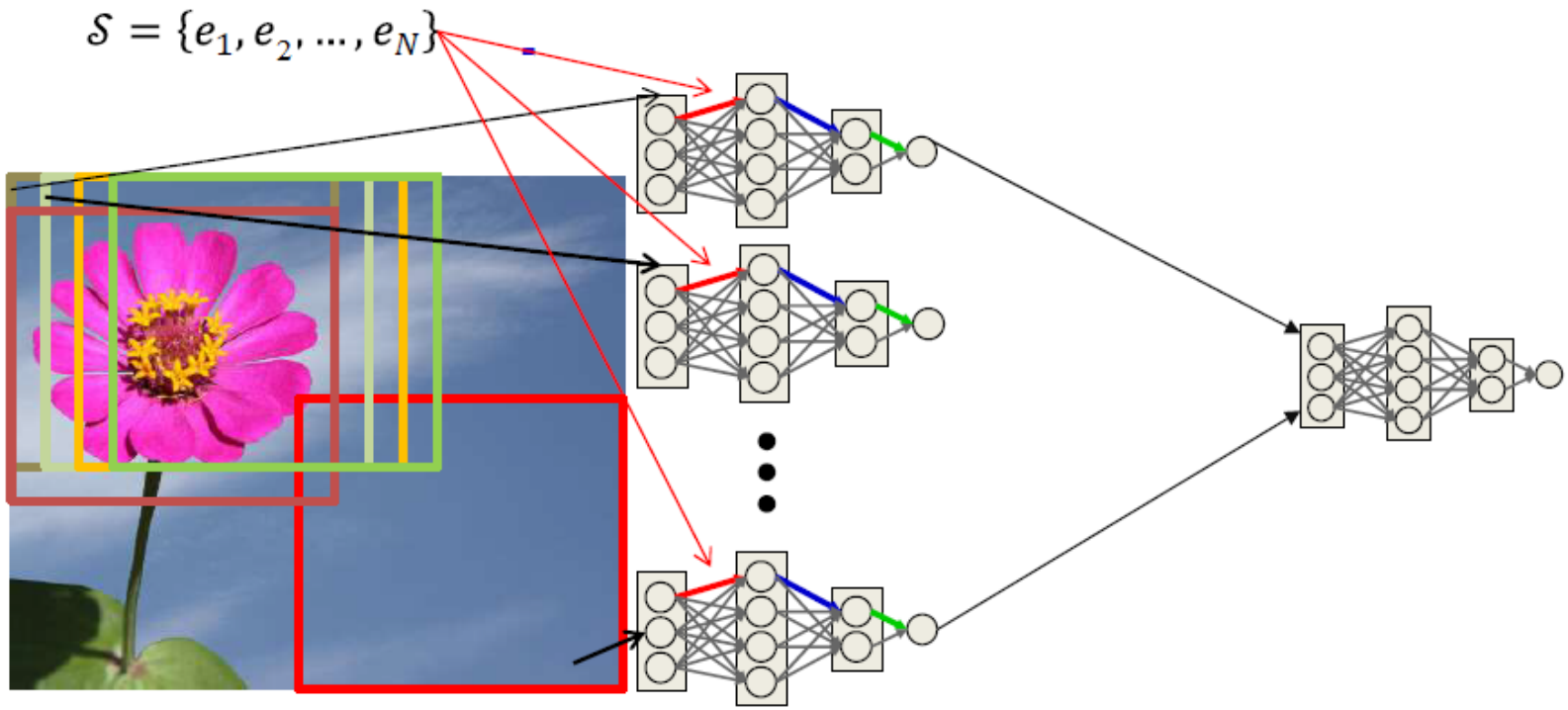


Computing the divergence of shared parameters

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

$$\frac{d\,Div}{d\,w^{\mathcal{S}}} = \sum_{i=1}^{N} \frac{\partial Div}{\partial w^{e_i}}$$

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



Training networks with shared parameters

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

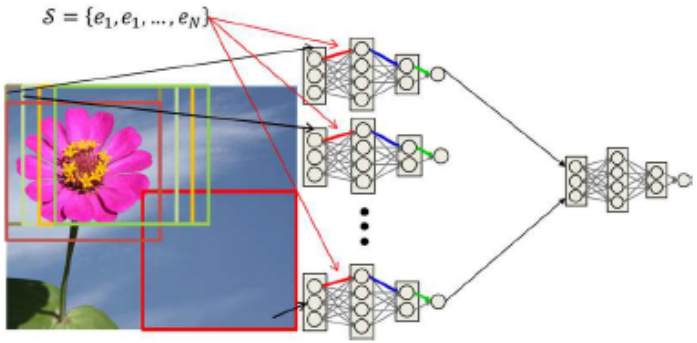
$$\nabla_{\mathcal{S}} Loss = \frac{dLoss}{dw^{\mathcal{S}}}$$

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

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

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

- Until $Loss$ has converged



63

Training networks with shared parameters

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

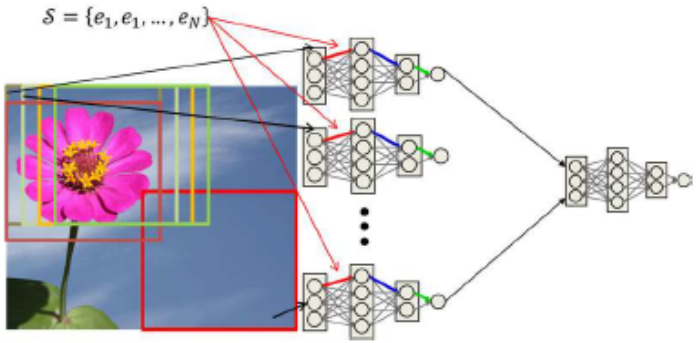
$$\nabla_{\mathcal{S}} Loss = \frac{dLoss}{dw^{\mathcal{S}}}$$

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

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

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

- Until $Loss$ has converged



64

Training networks with shared parameters

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

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

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

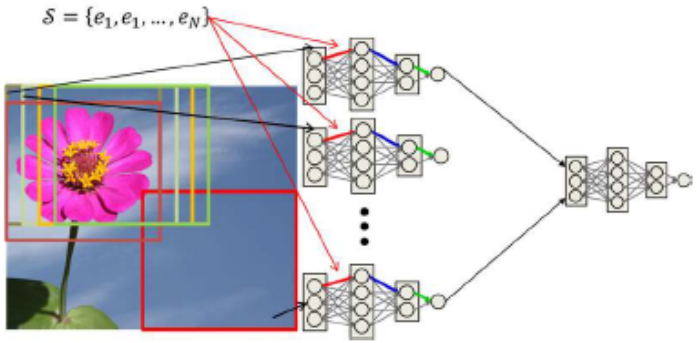
$$\nabla_{\mathcal{S}} Loss = \frac{dLoss}{dw^{\mathcal{S}}}$$

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

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

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

- Until $Loss$ has converged



65

Training networks with shared parameters

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

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

Computed by Backprop

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

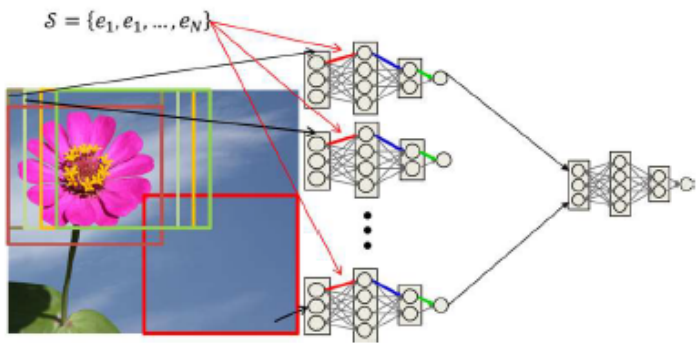
$$\nabla_{\mathcal{S}} Loss = \frac{dLoss}{dw^{\mathcal{S}}}$$

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

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

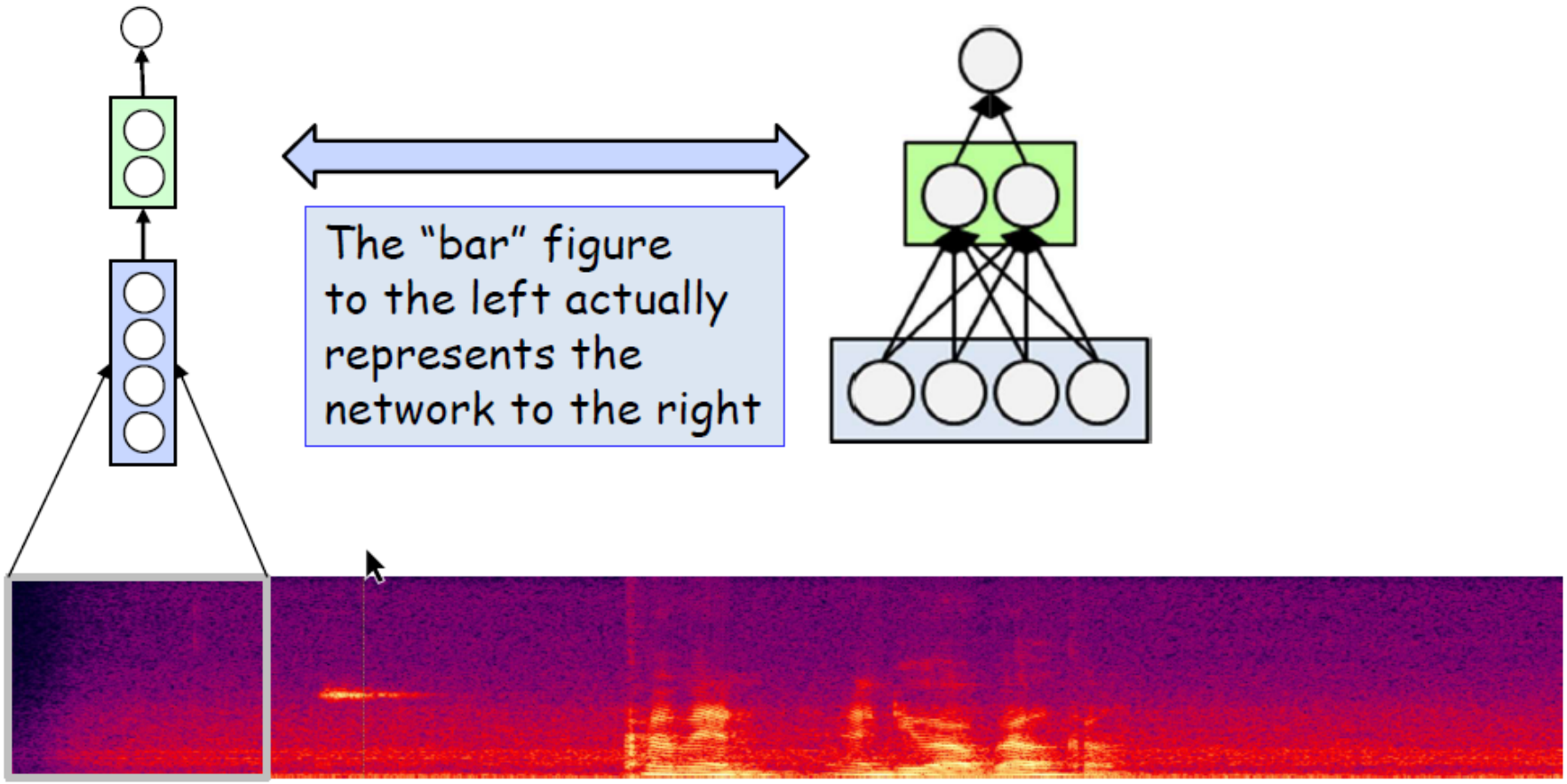
$$w_{i,j}^{(k)} = w^{\mathcal{S}}$$

- Until $Loss$ has converged

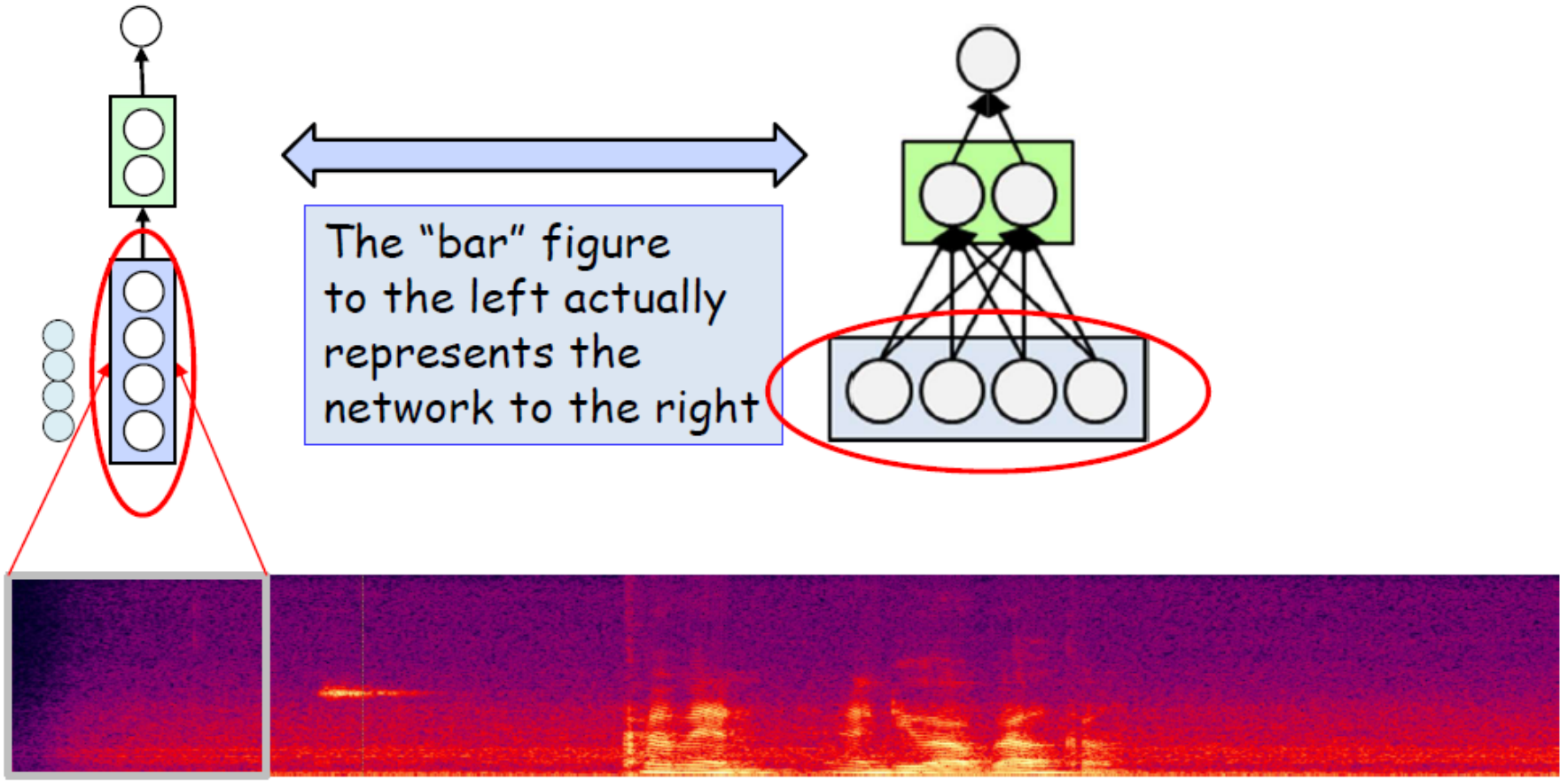


66

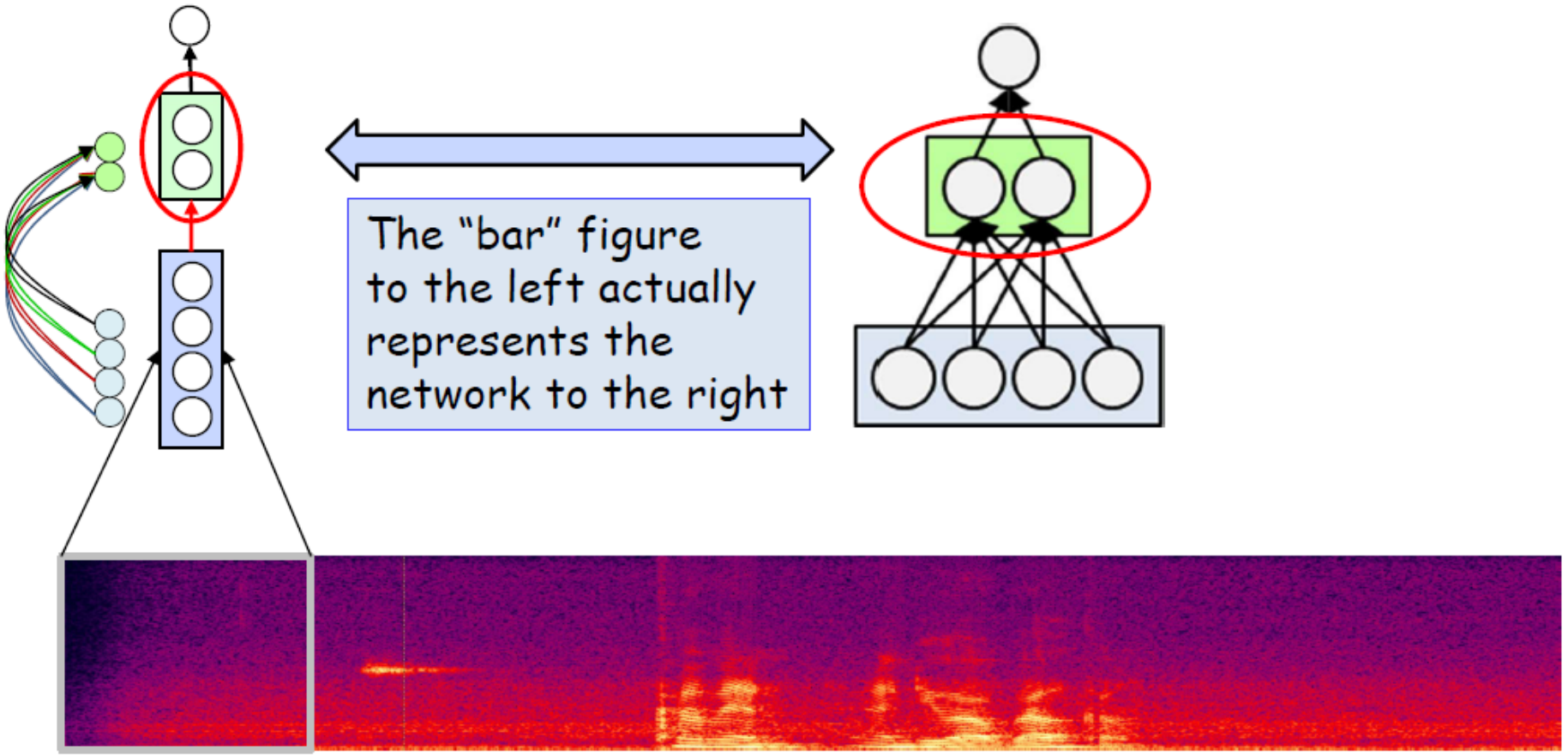
Let's do it in a different order



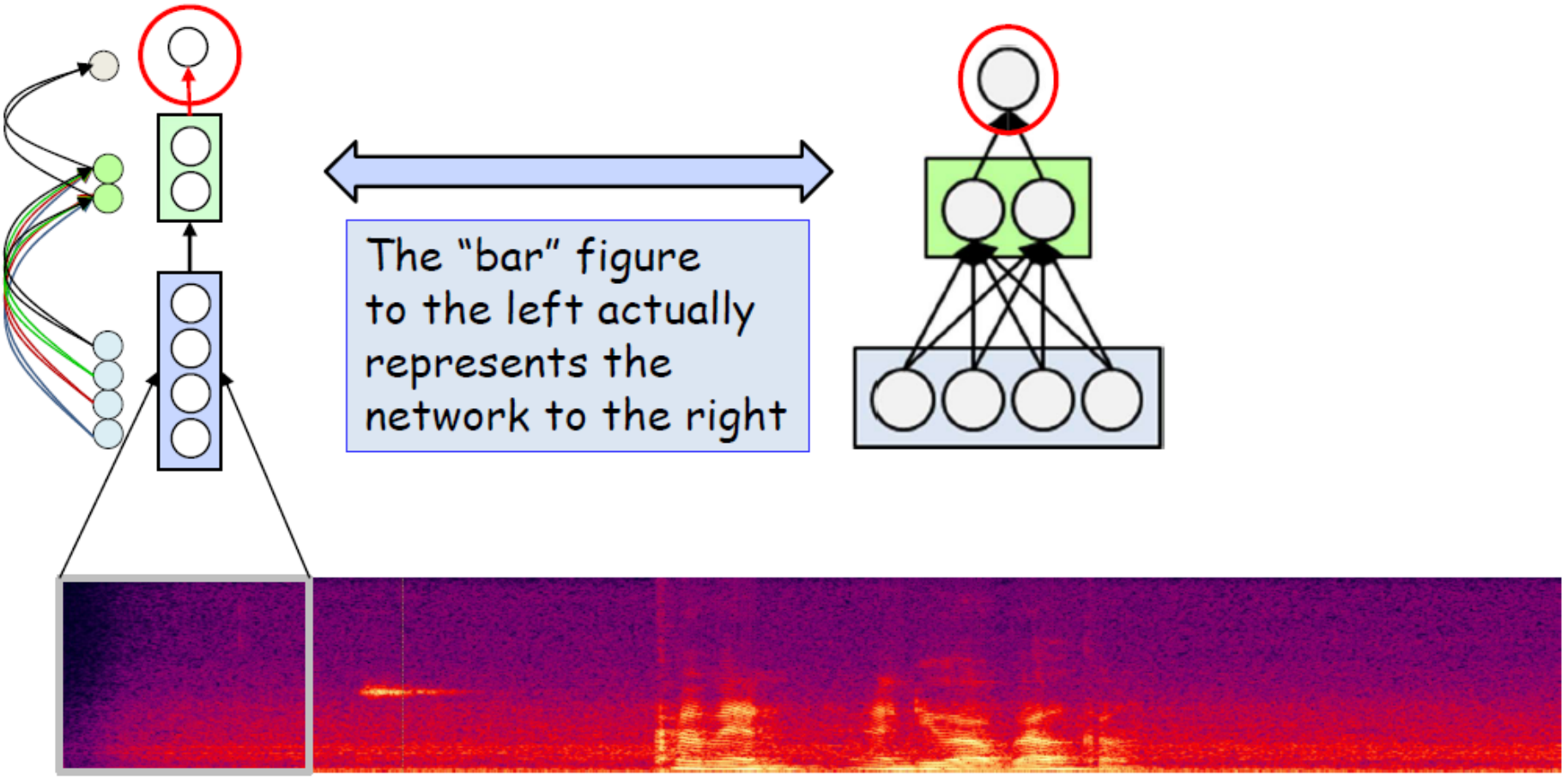
Let's do it in a different order



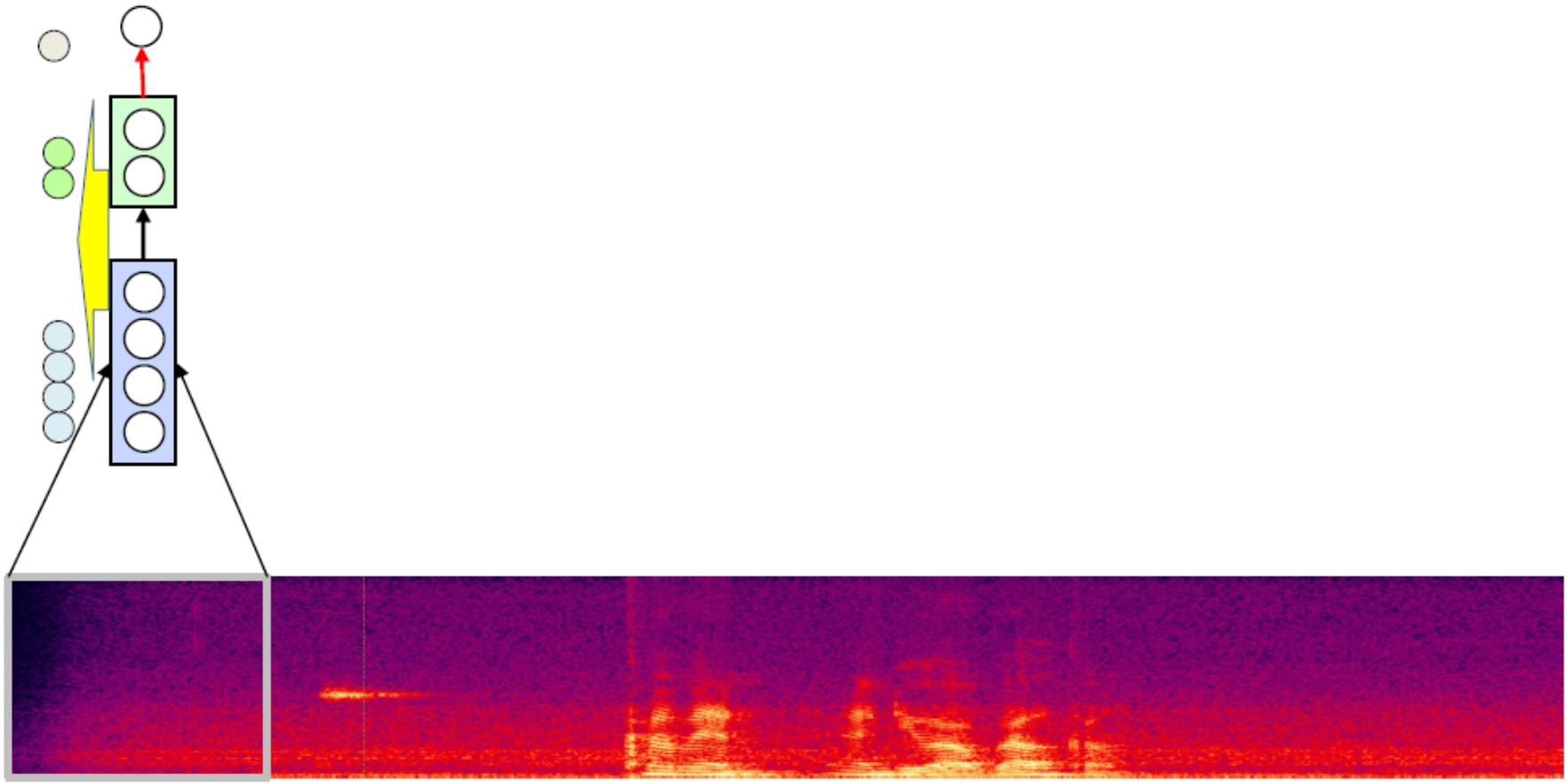
Let's do it in a different order



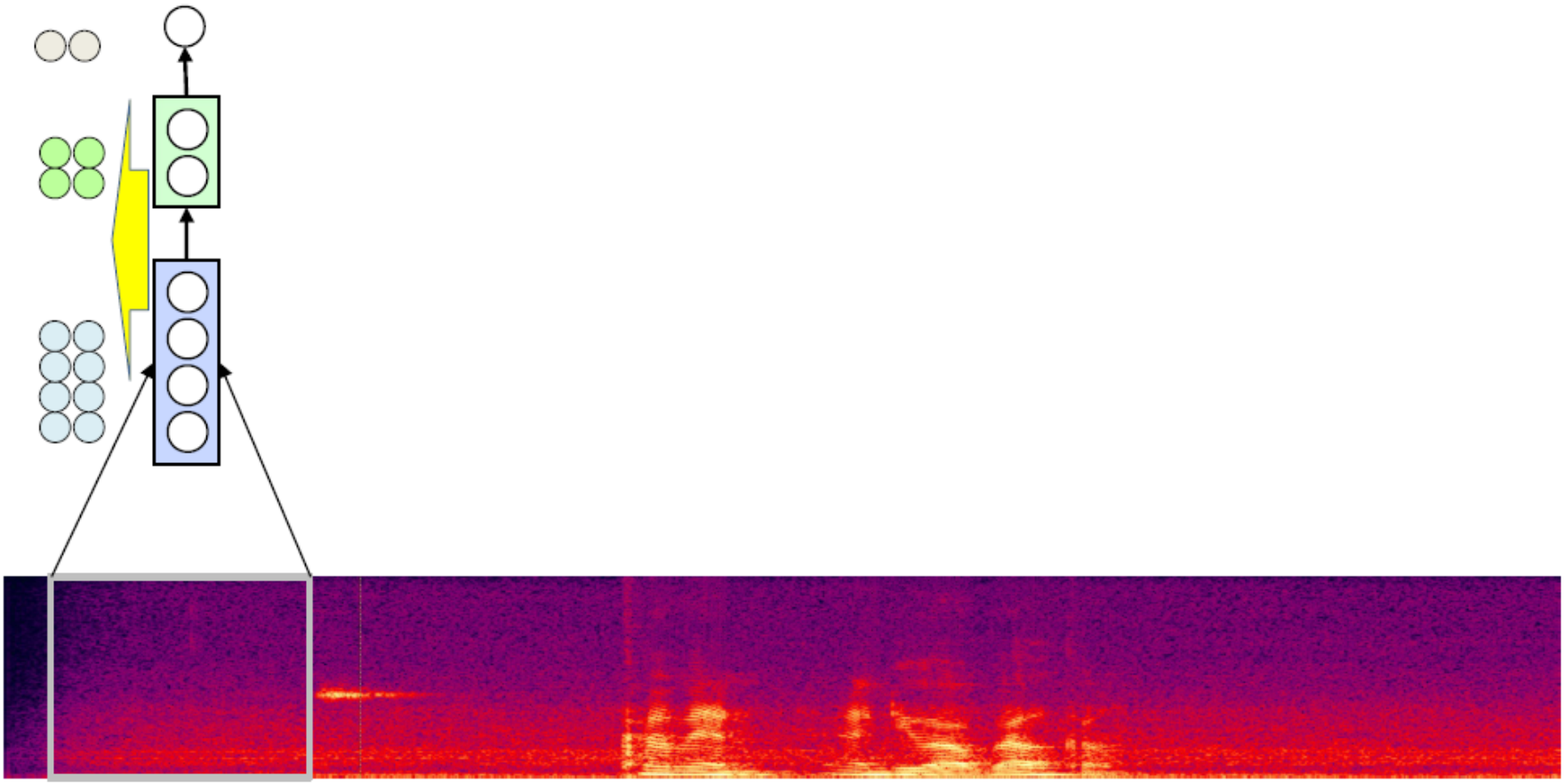
Let's do it in a different order



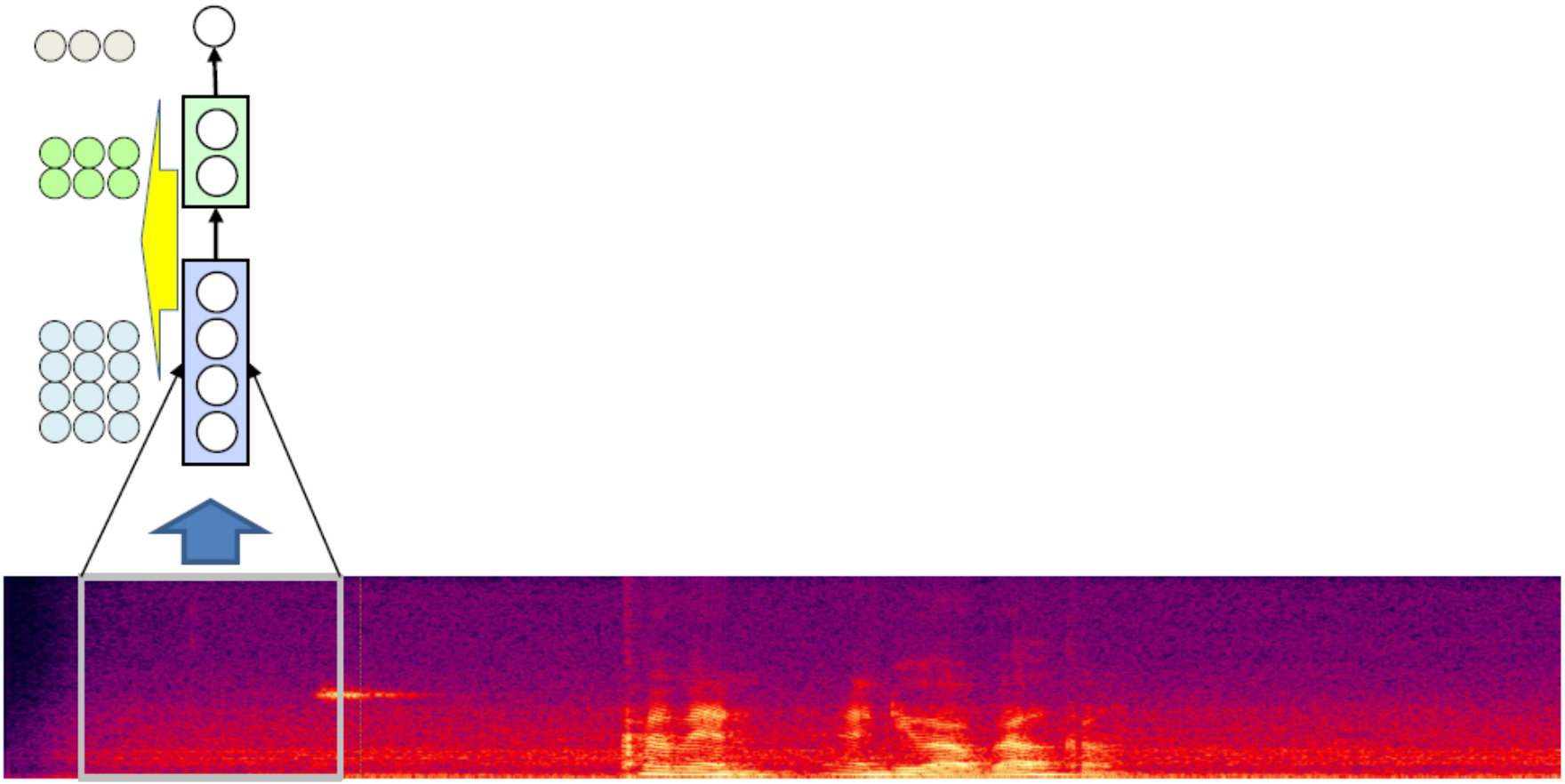
Let's do it in a different order



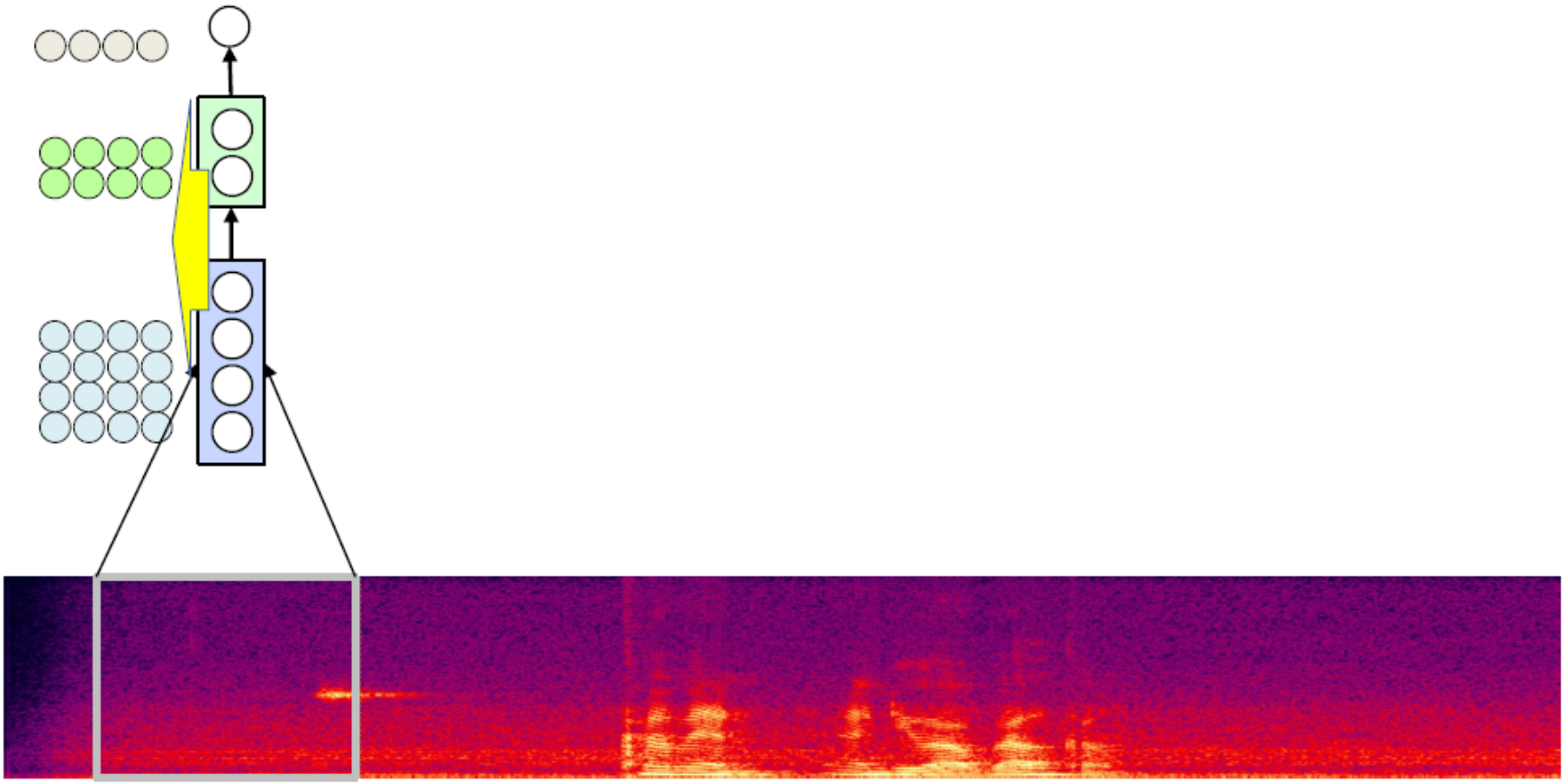
Let's do it in a different order



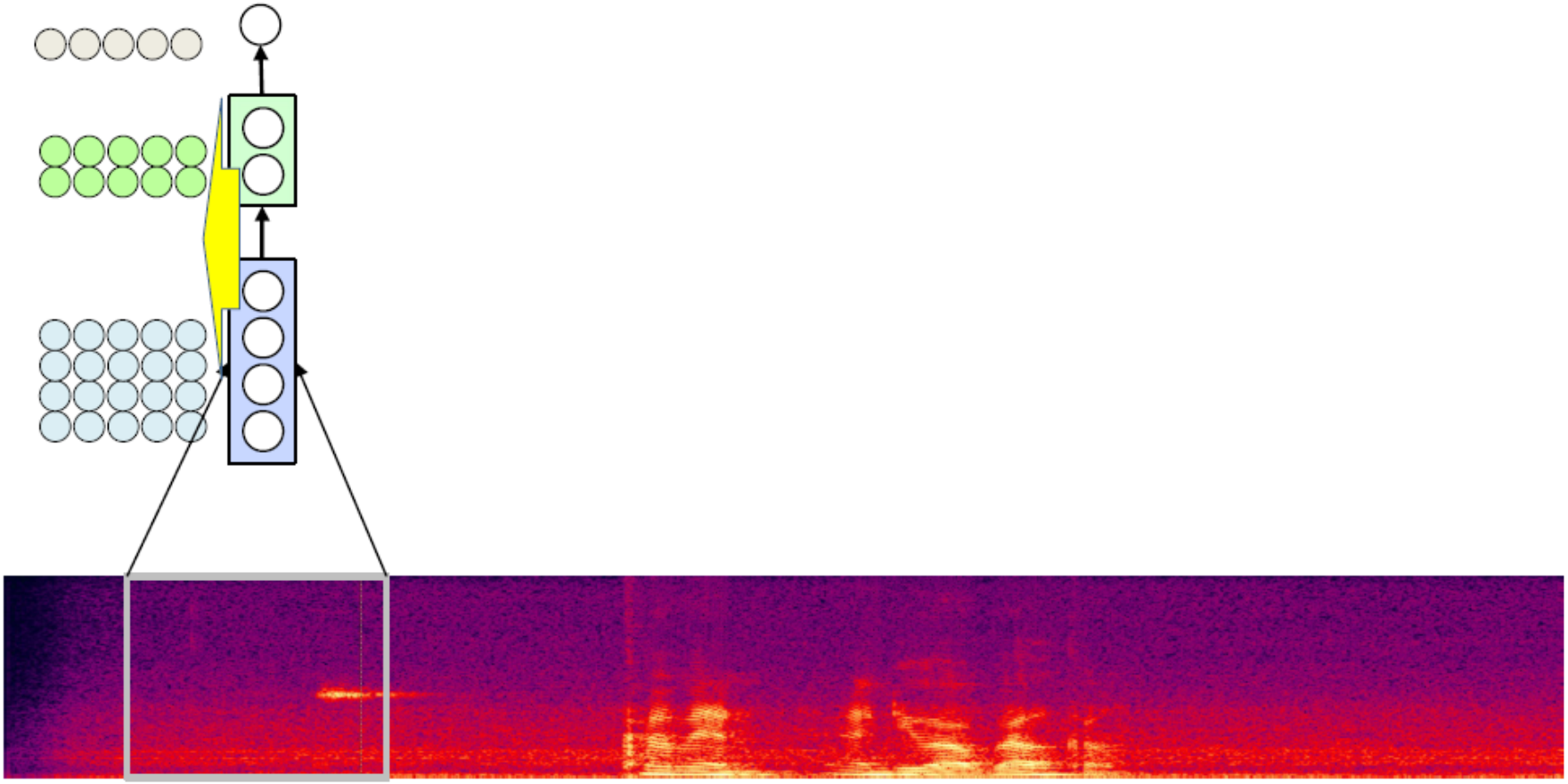
Let's do it in a different order



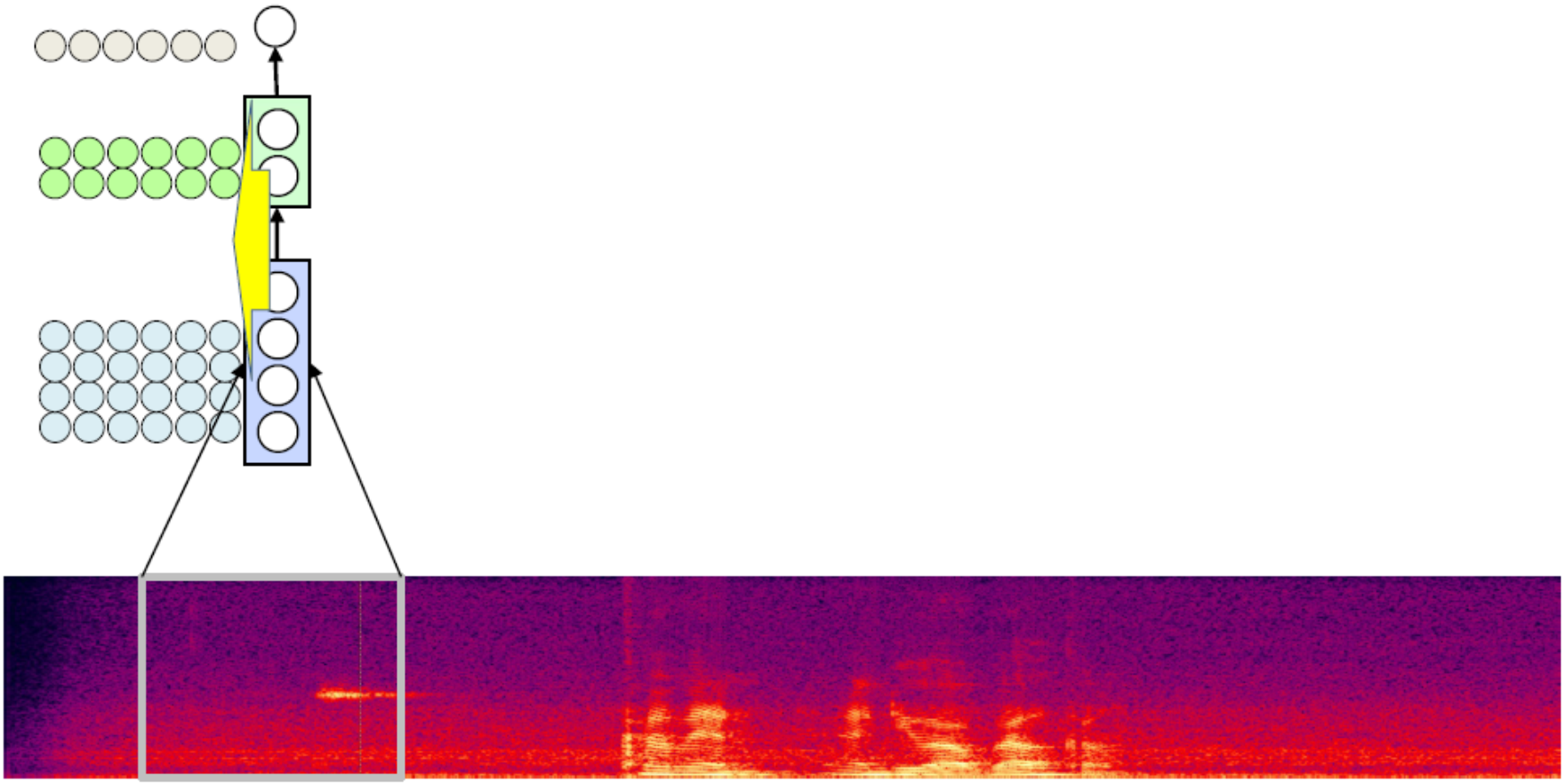
Let's do it in a different order



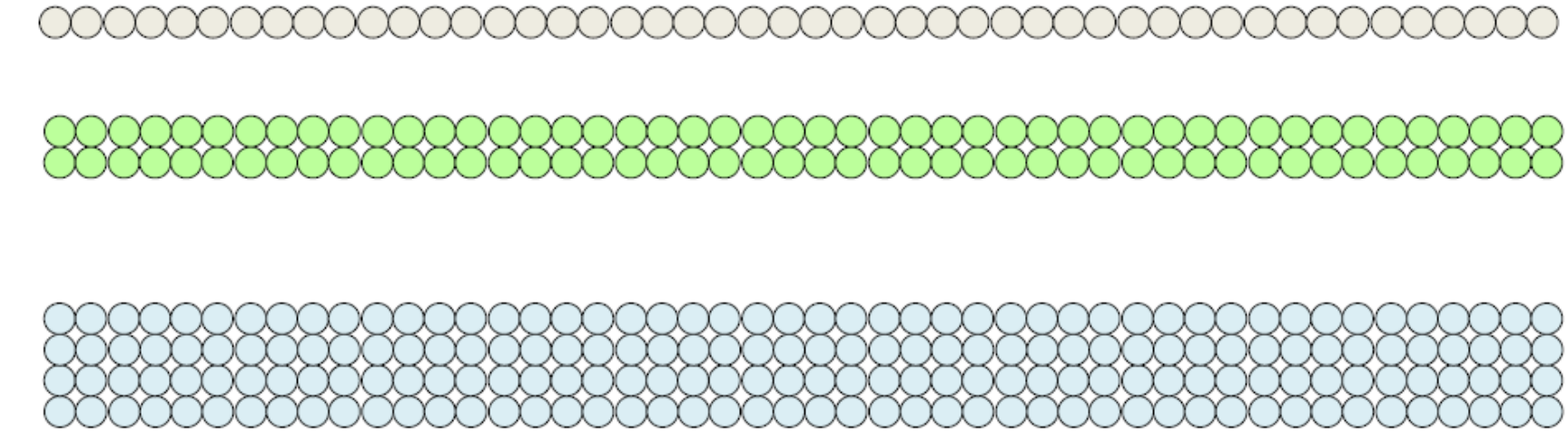
Let's do it in a different order

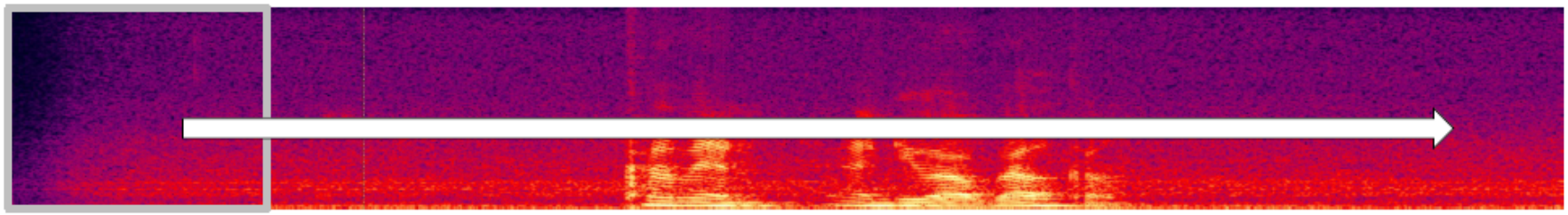


Let's do it in a different order

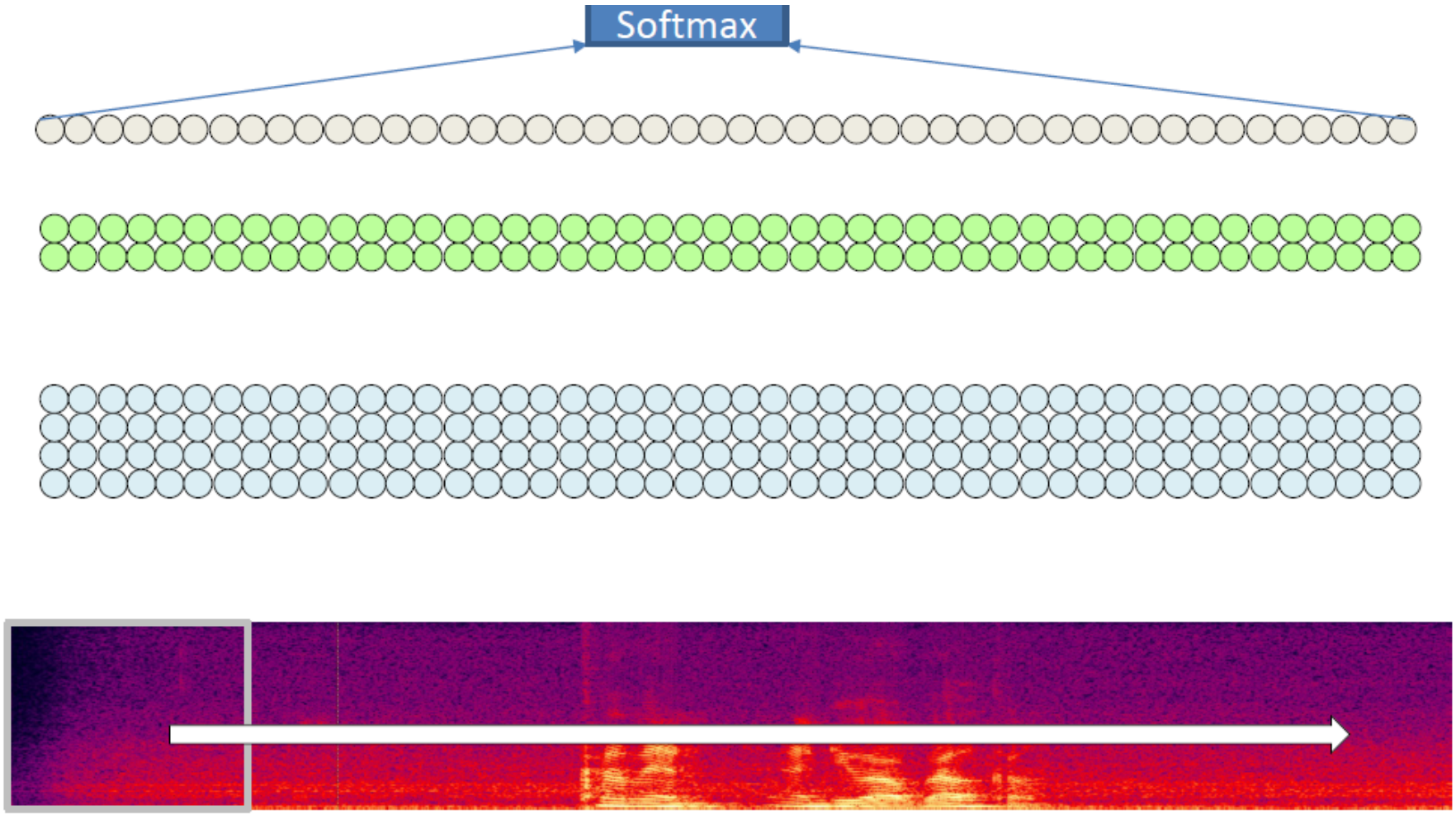


Let's do it in a different order

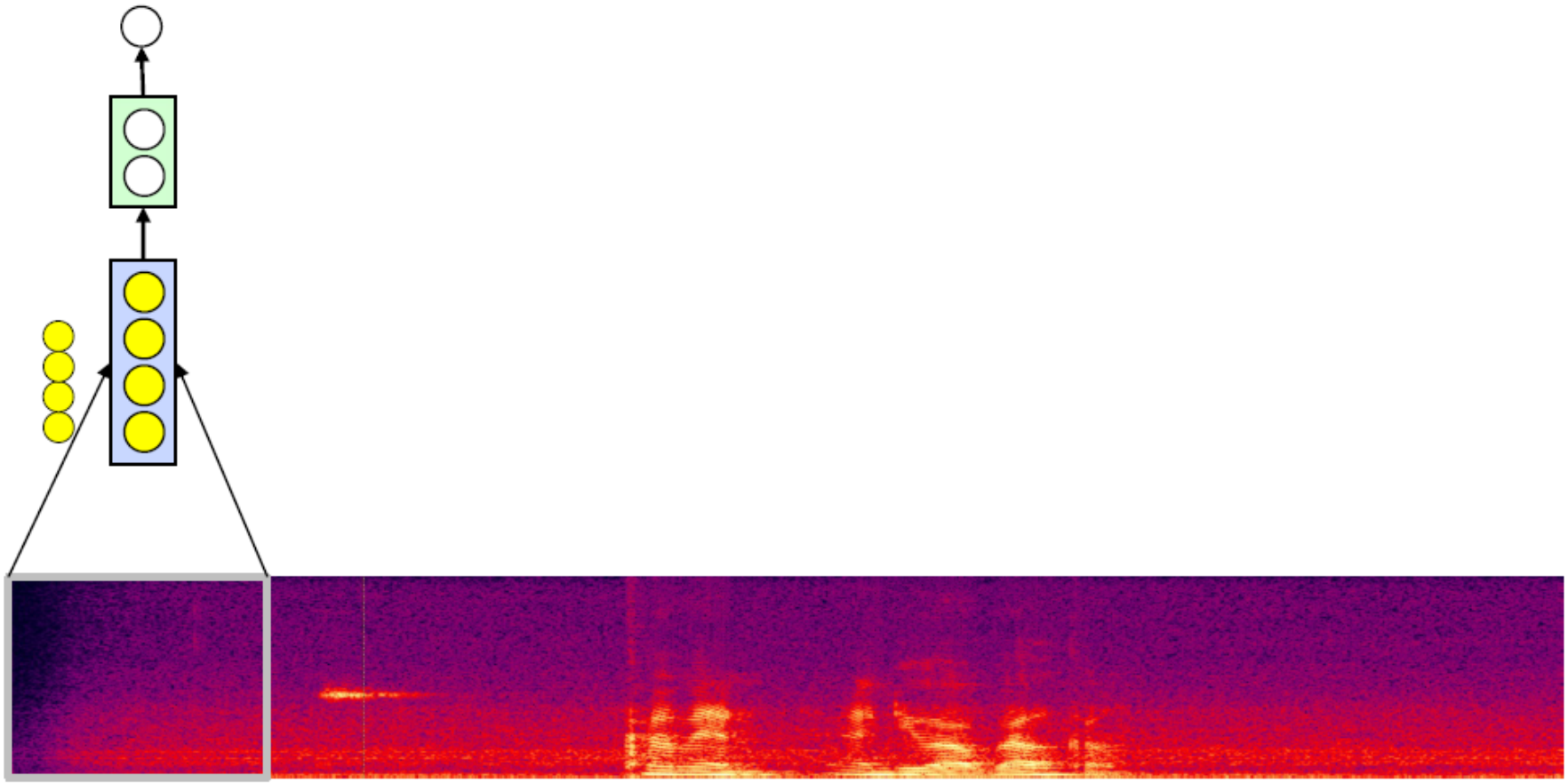




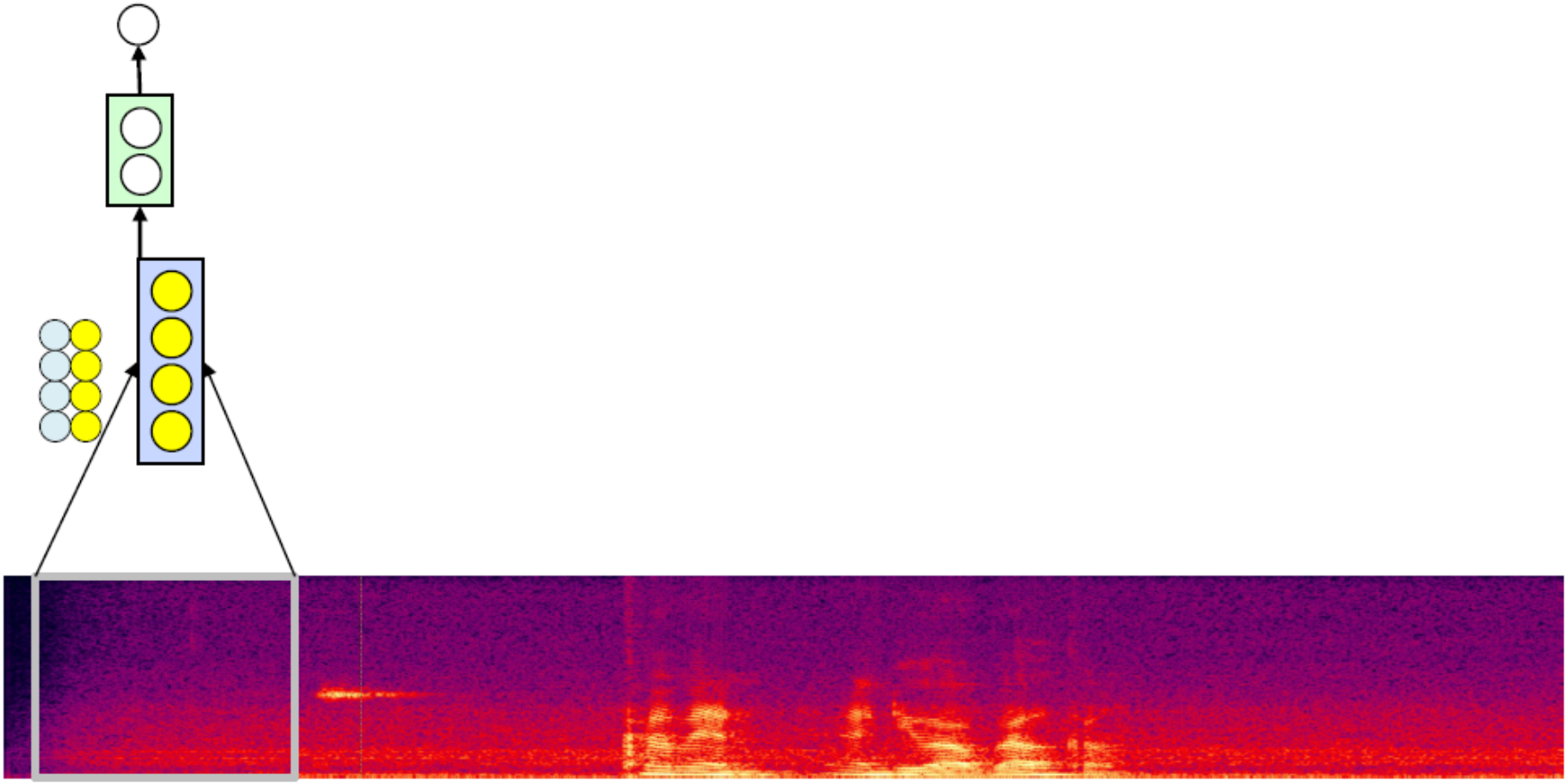
Let's do it in a different order



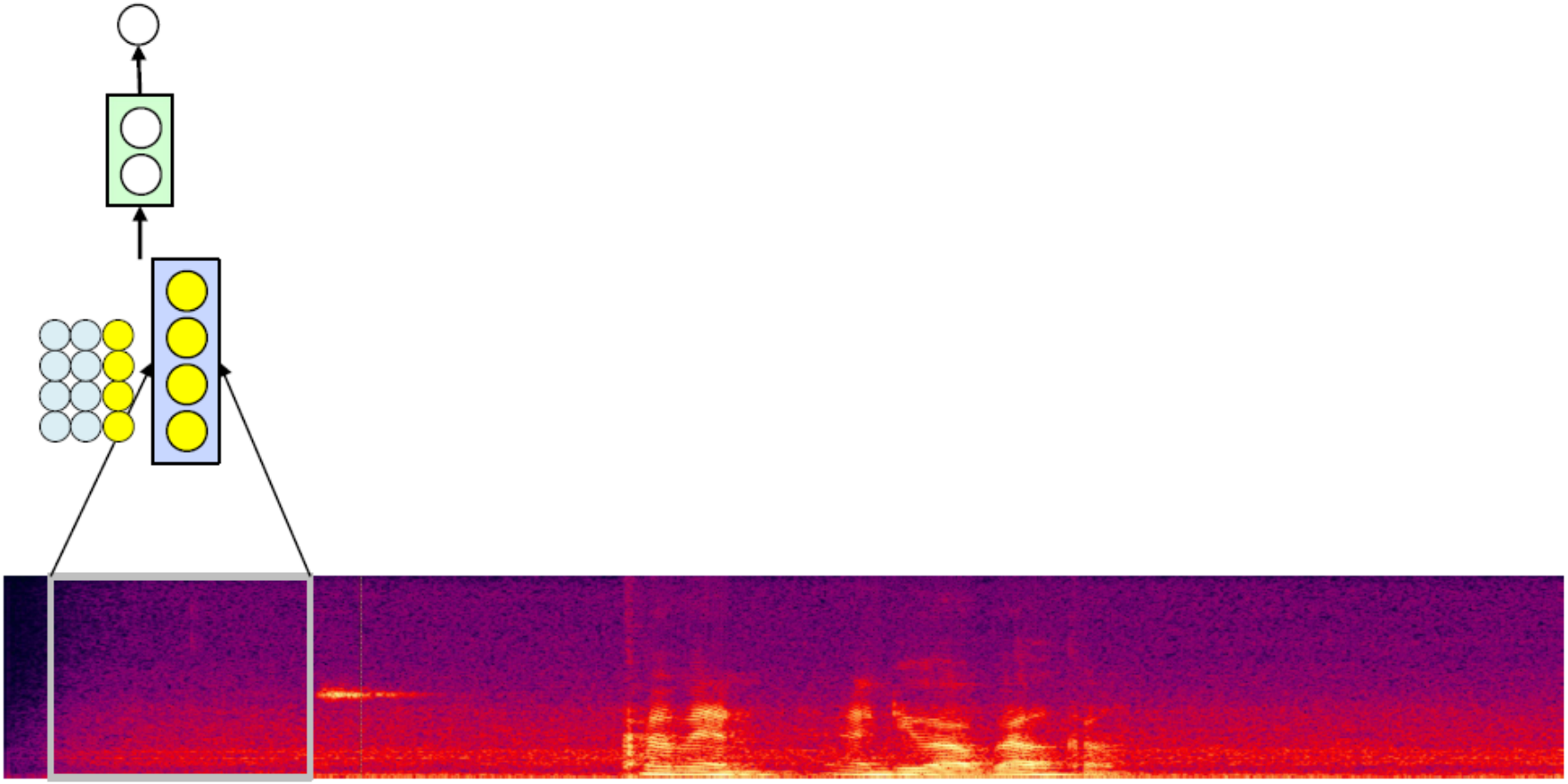
Let's do it in a different order



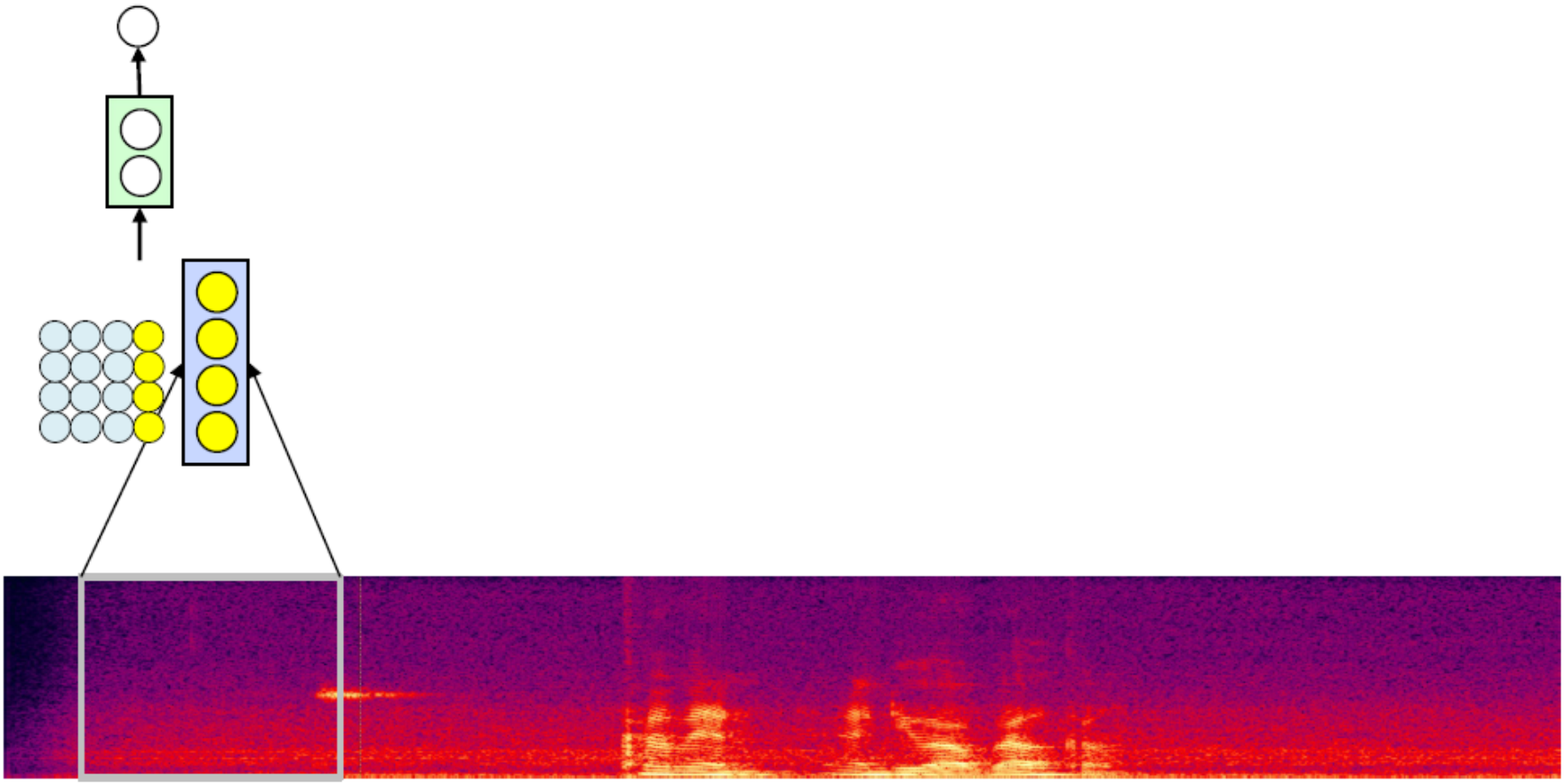
Let's do it in a different order



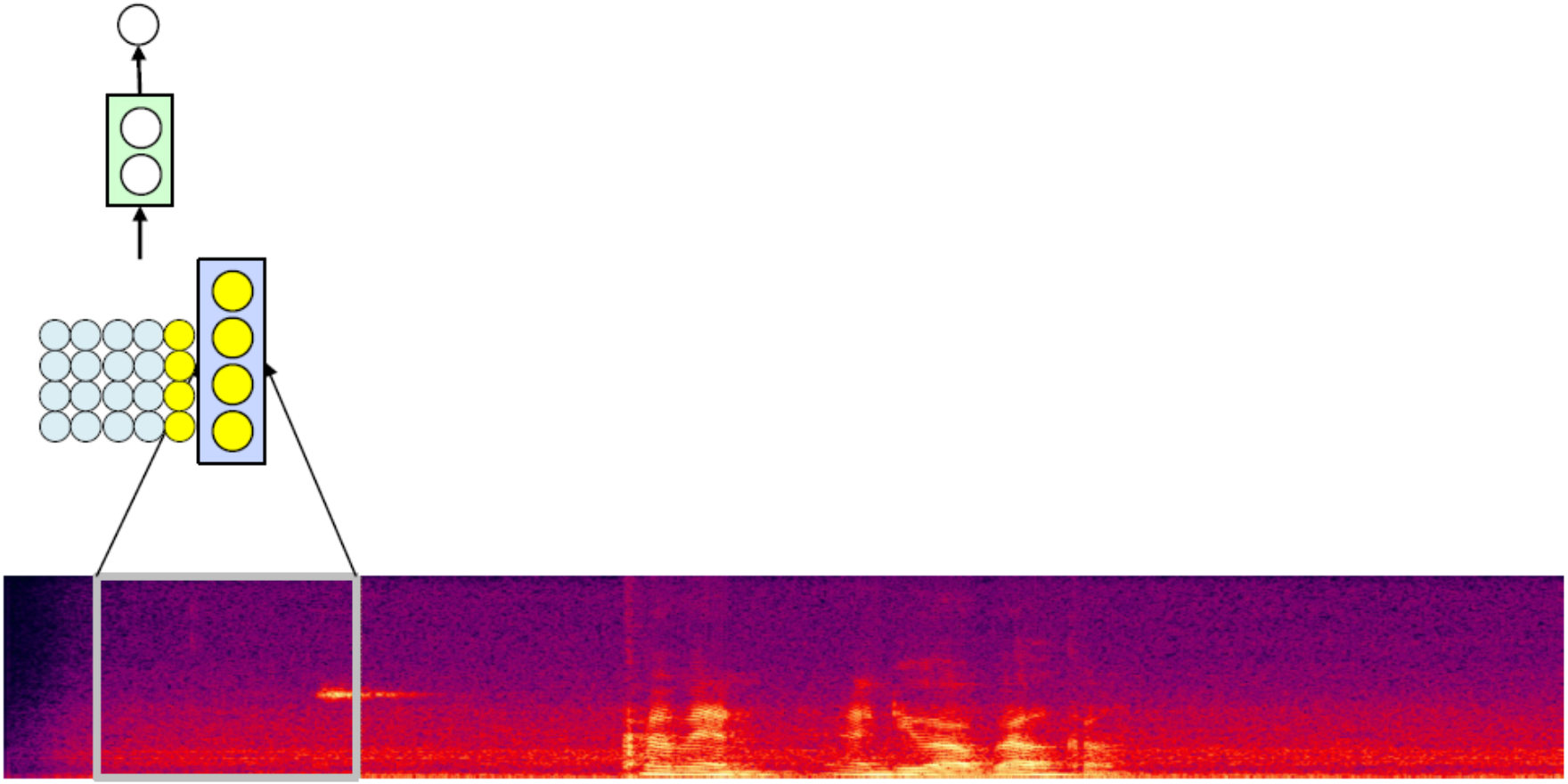
Let's do it in a different order



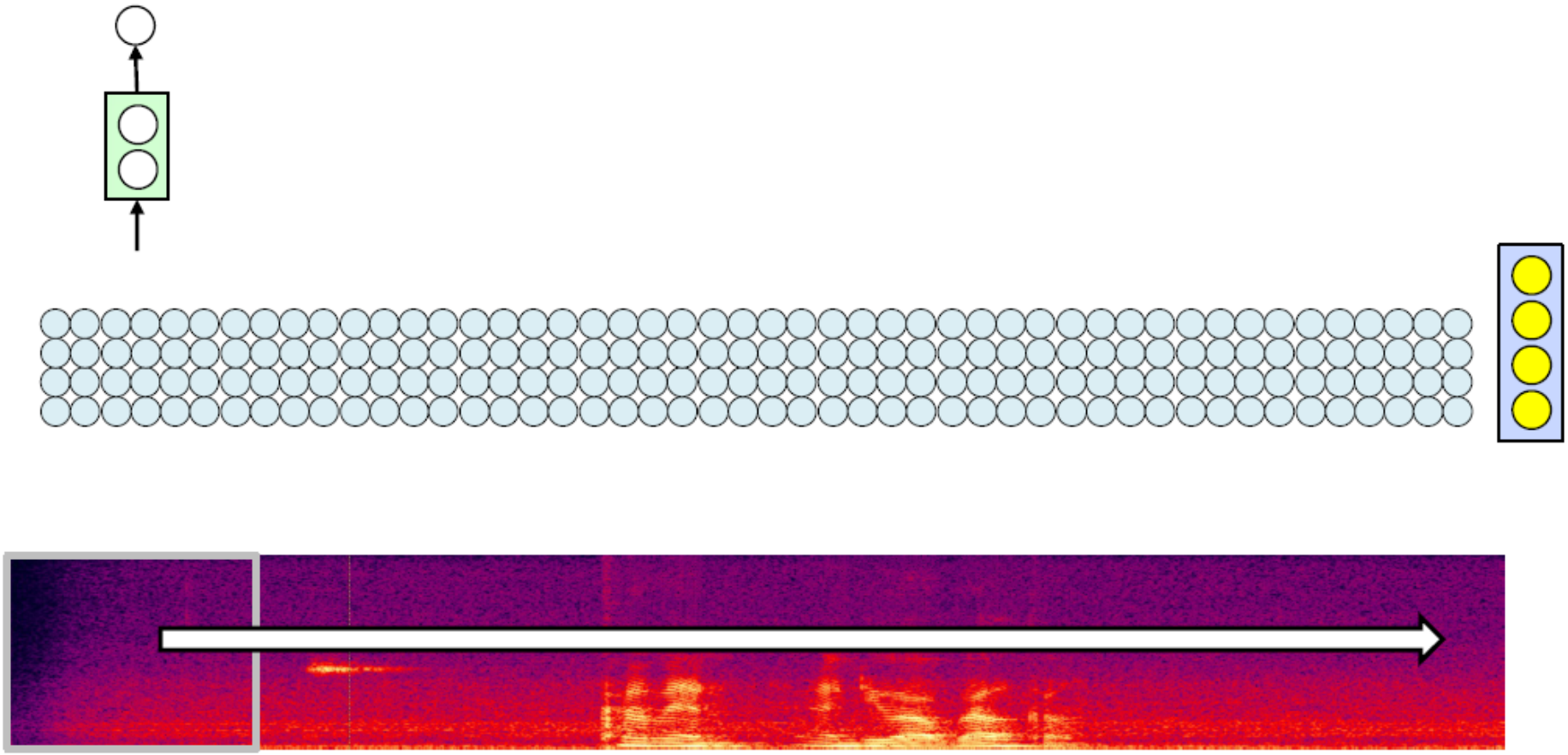
Let's do it in a different order



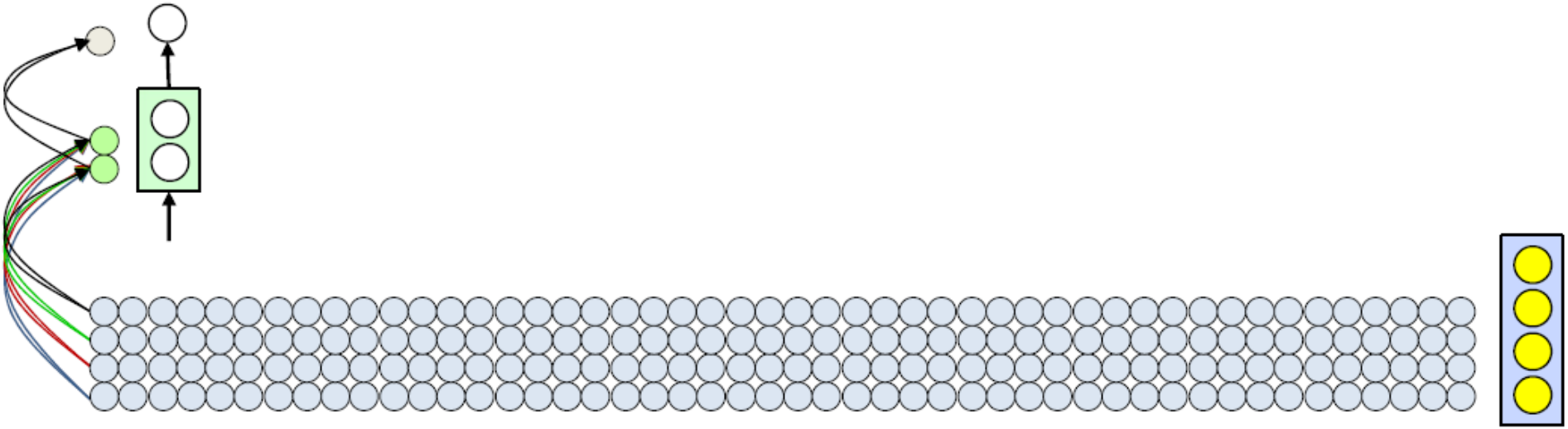
Let's do it in a different order

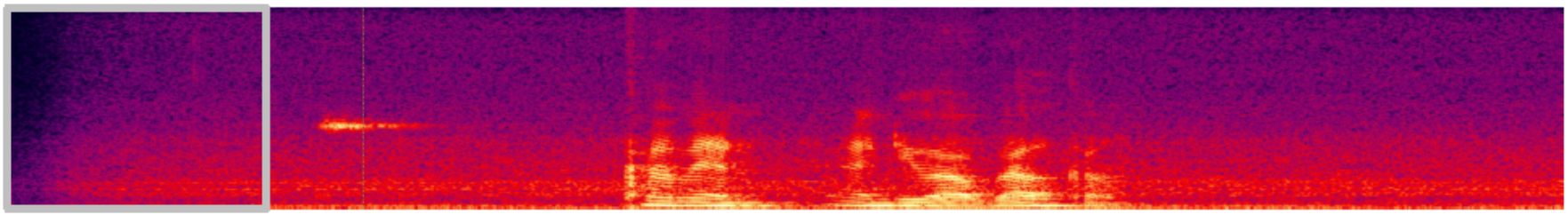


Let's do it in a different order

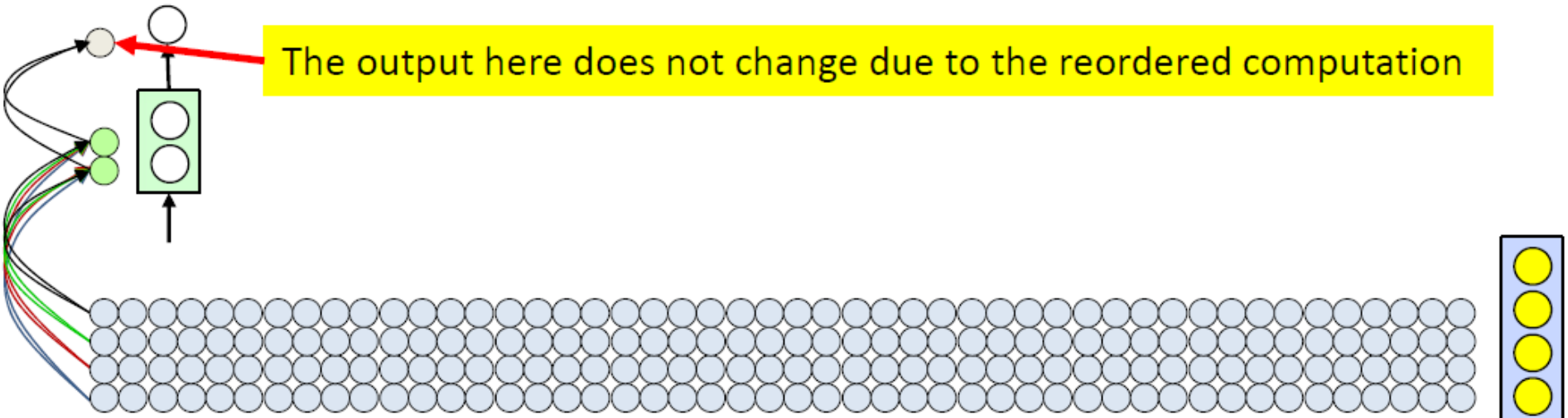


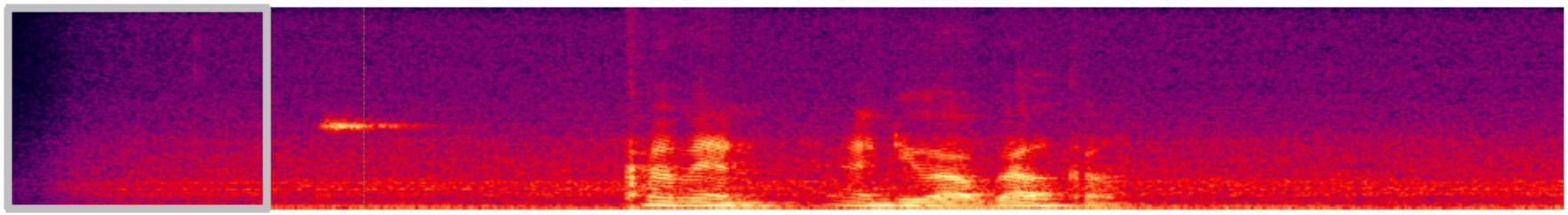
Let's do it in a different order



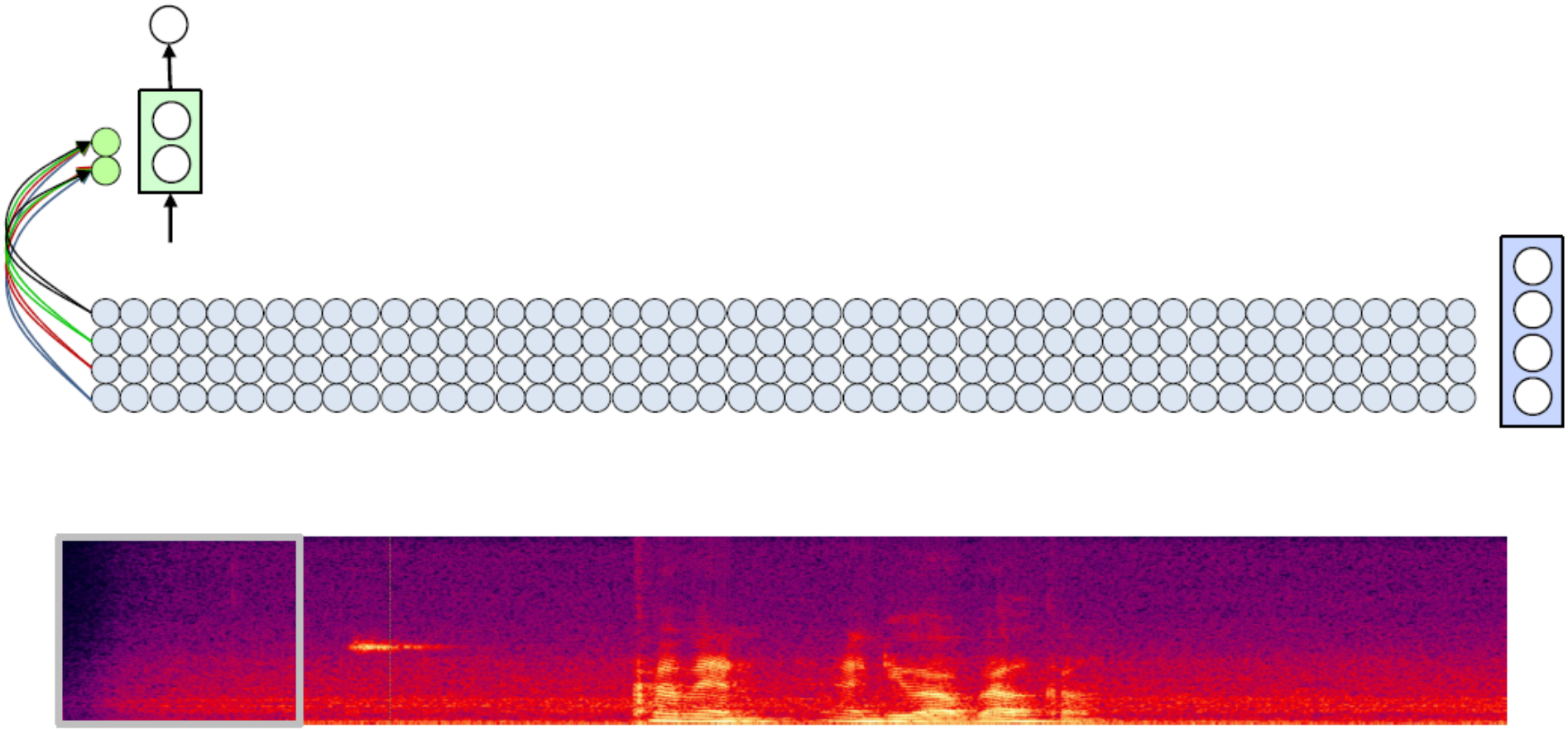


Let's do it in a different order

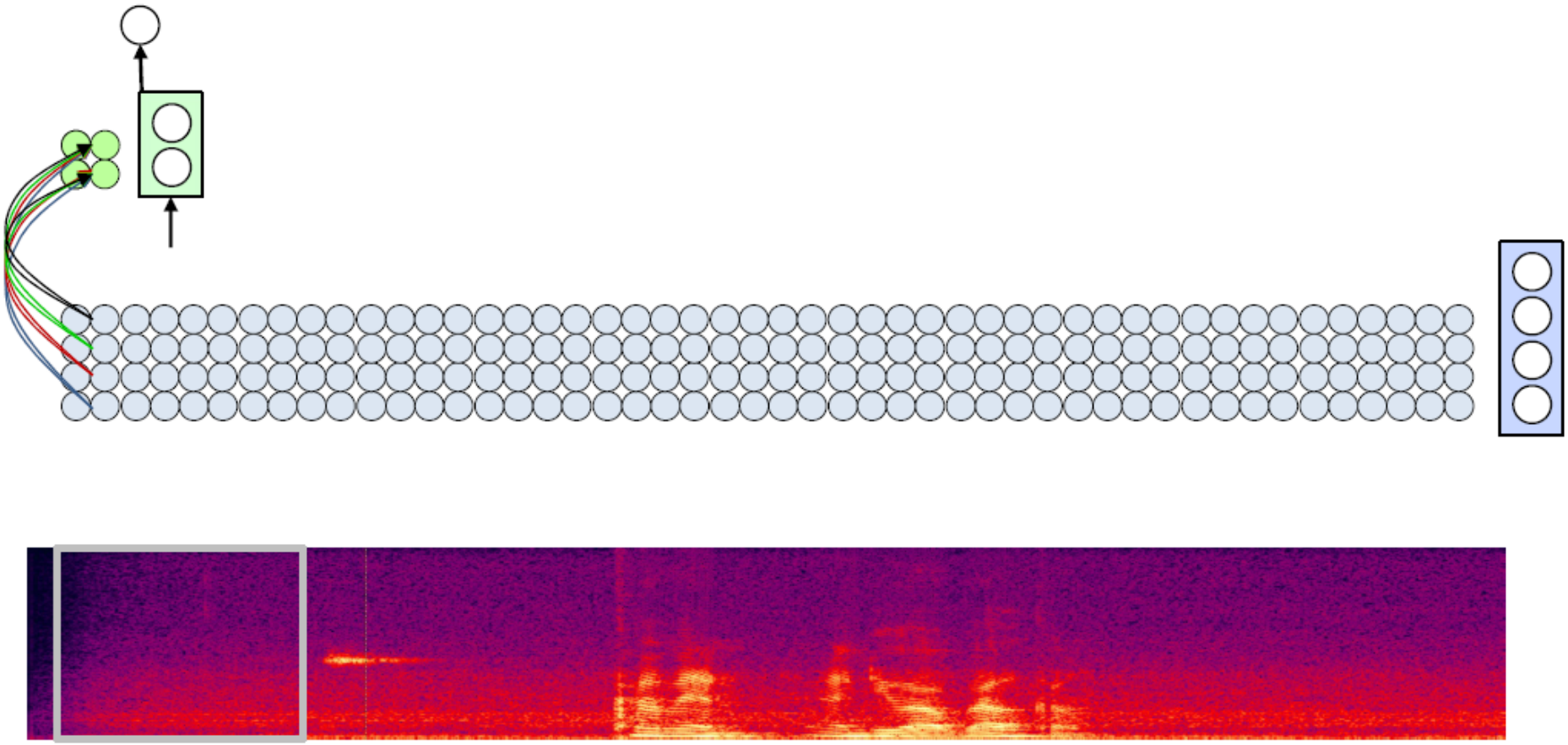




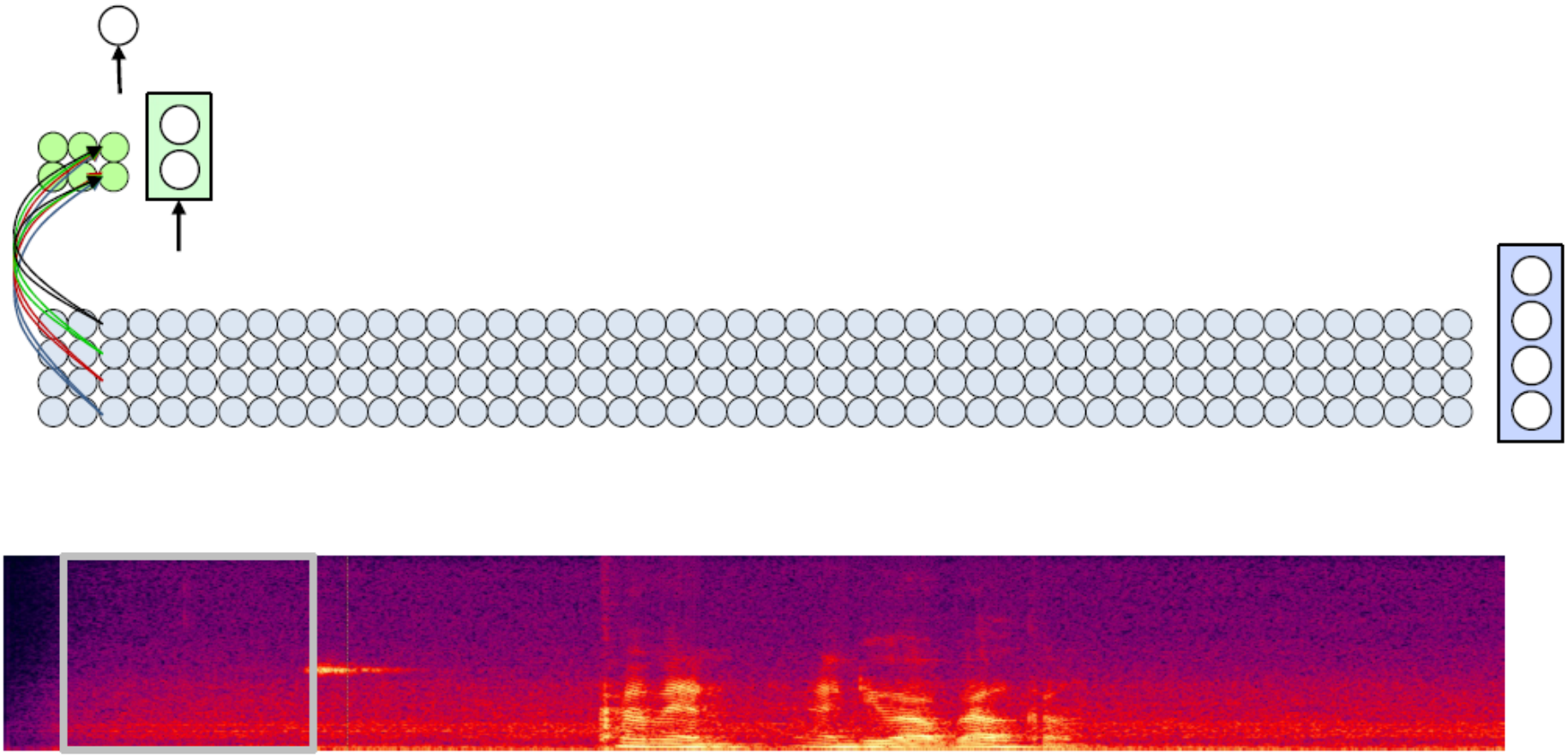
Let's do it in a different order



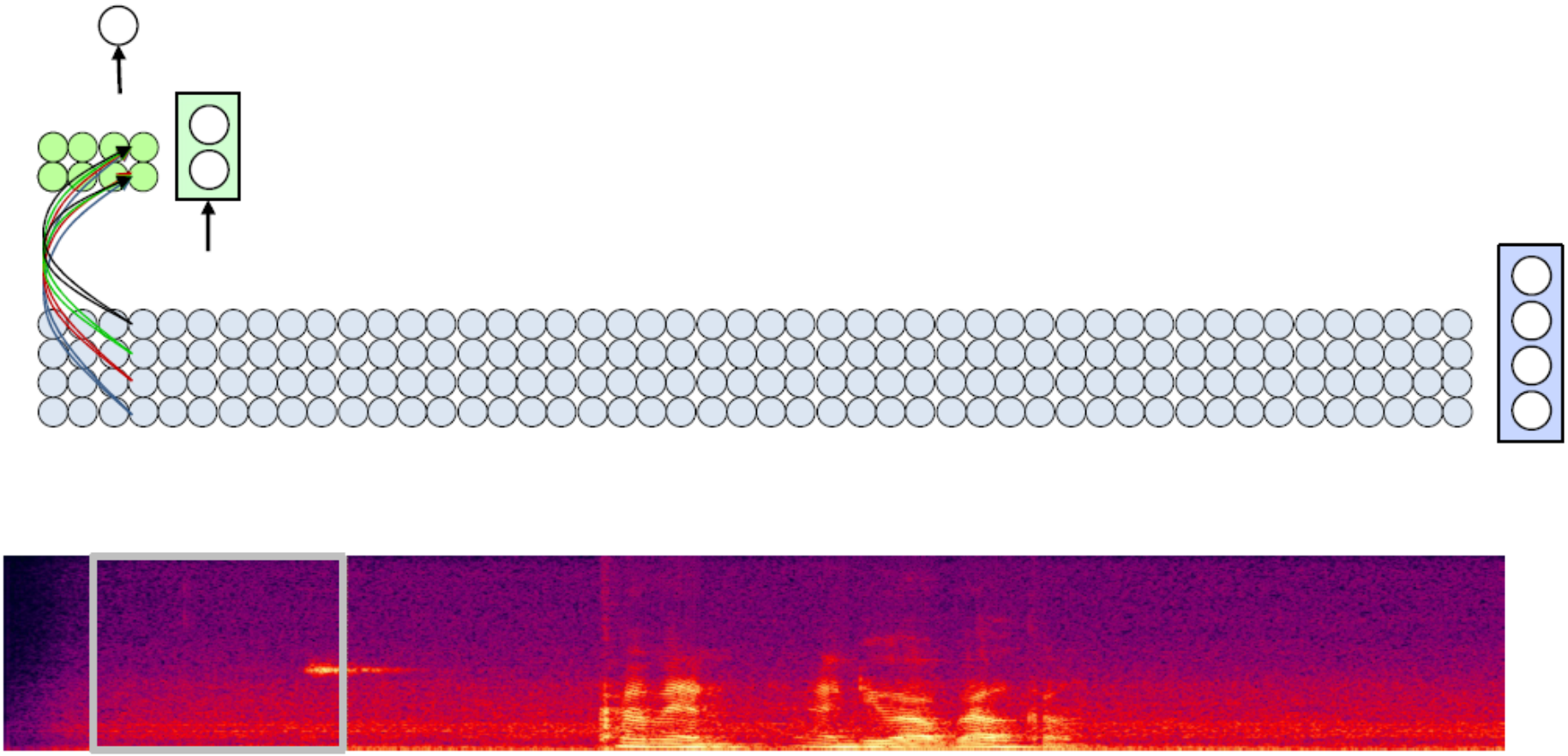
Let's do it in a different order



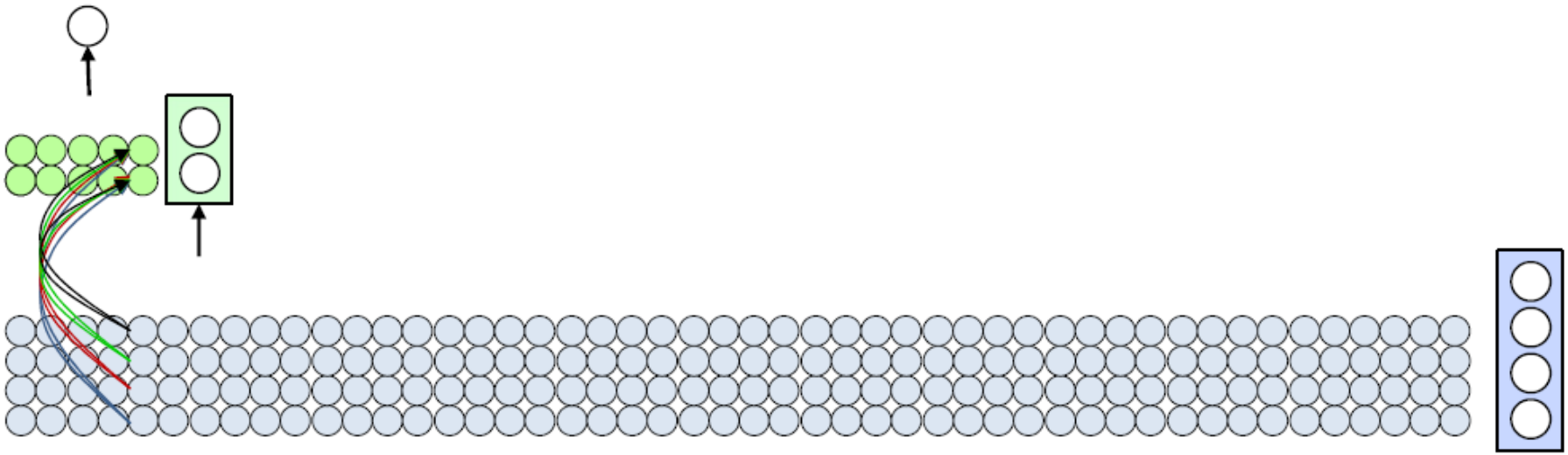
Let's do it in a different order

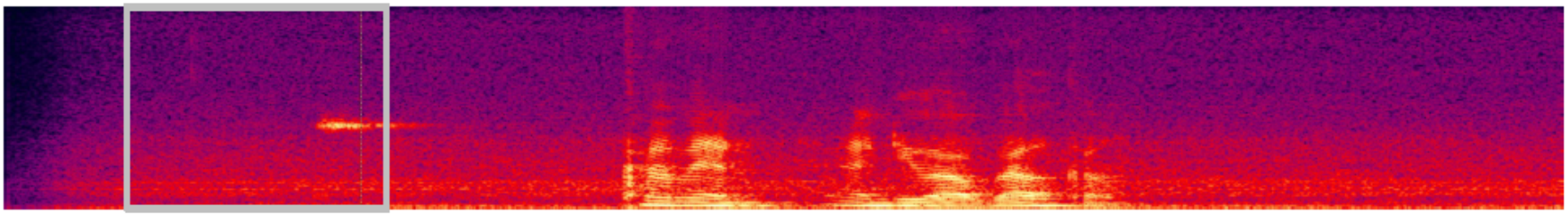


Let's do it in a different order

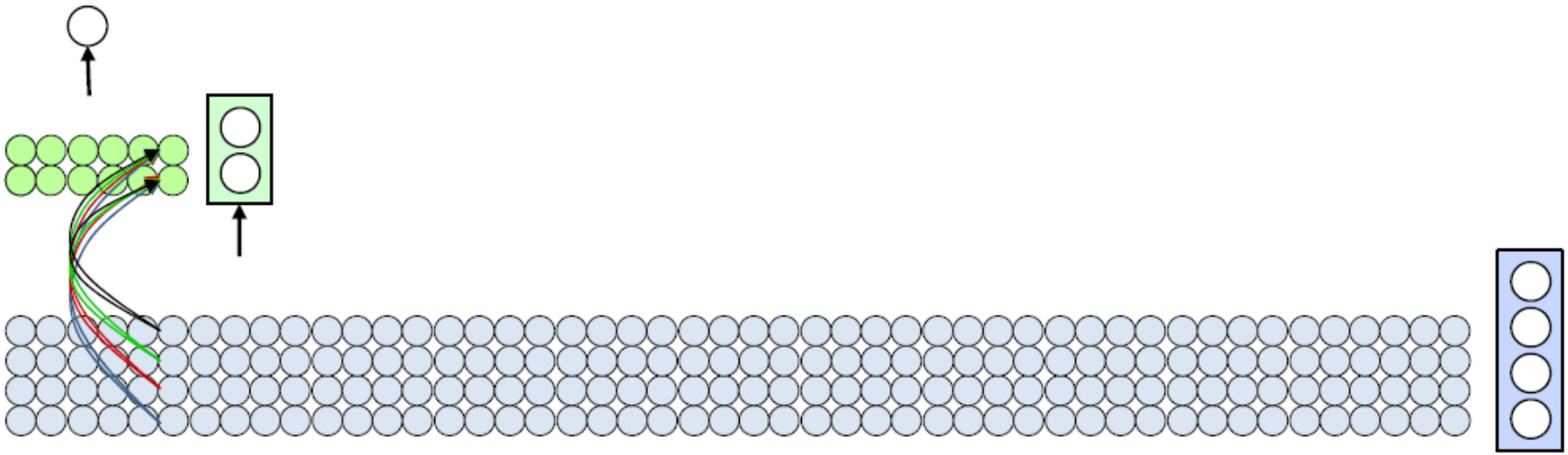


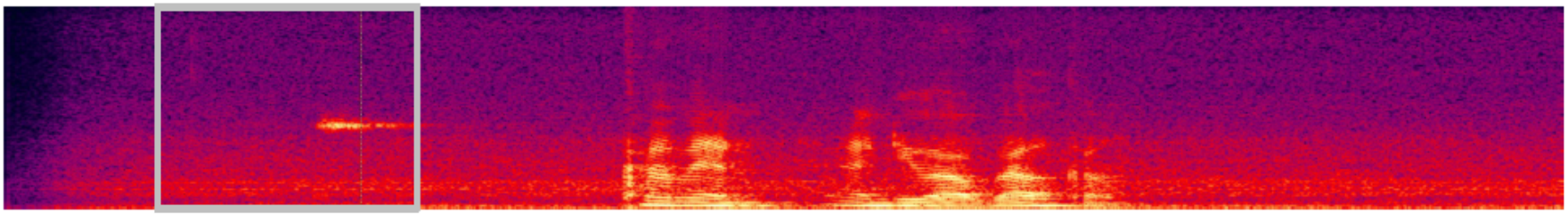
Let's do it in a different order



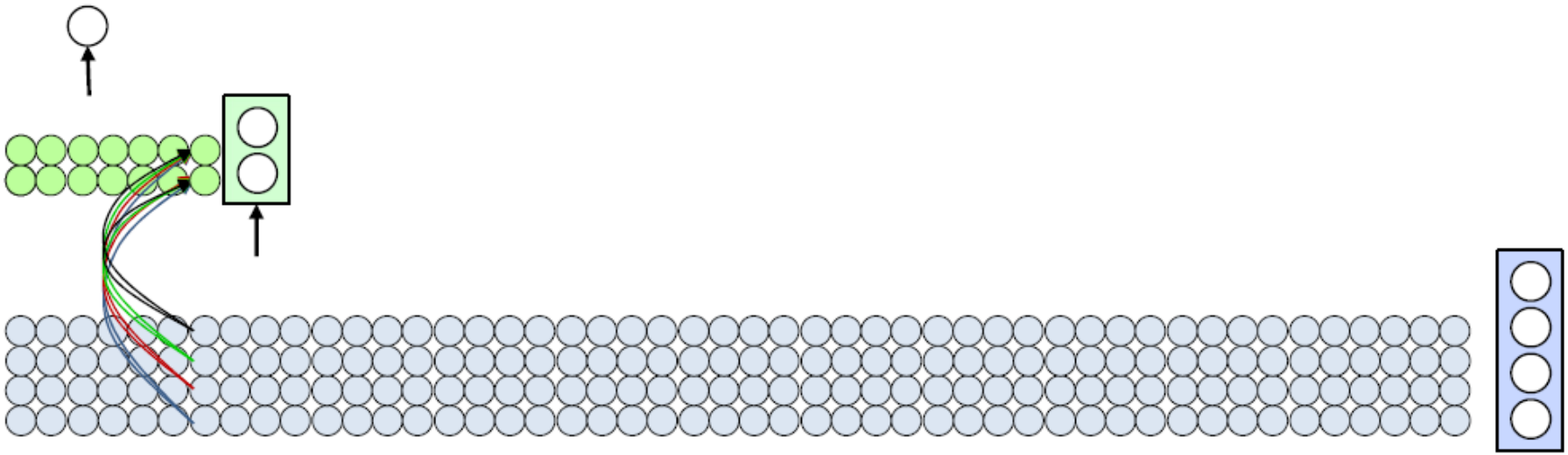


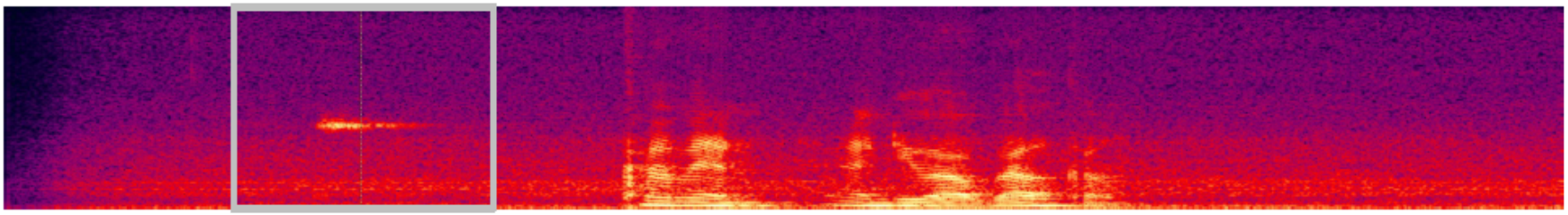
Let's do it in a different order



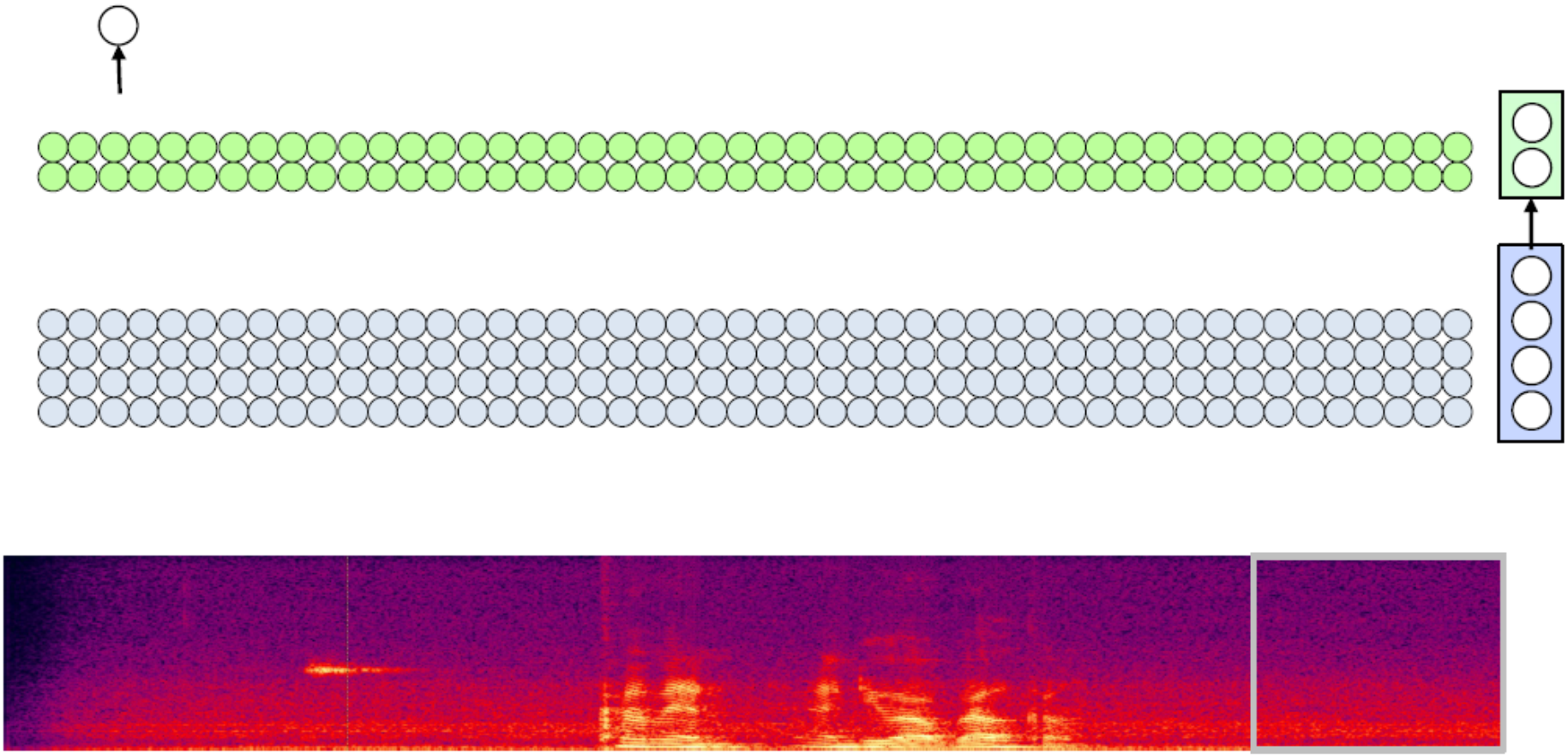


Let's do it in a different order

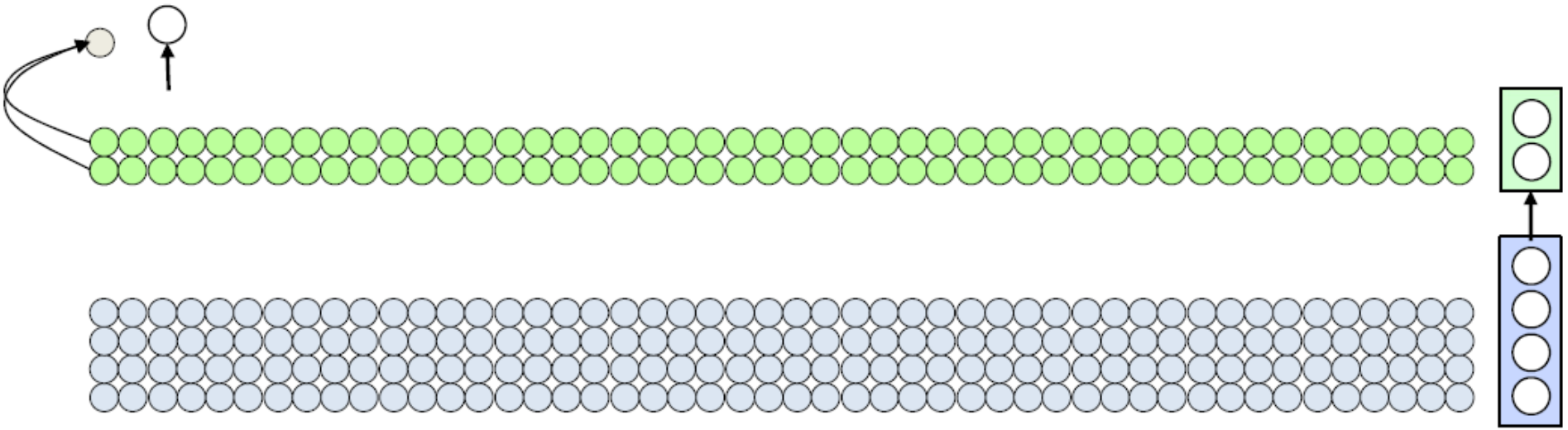


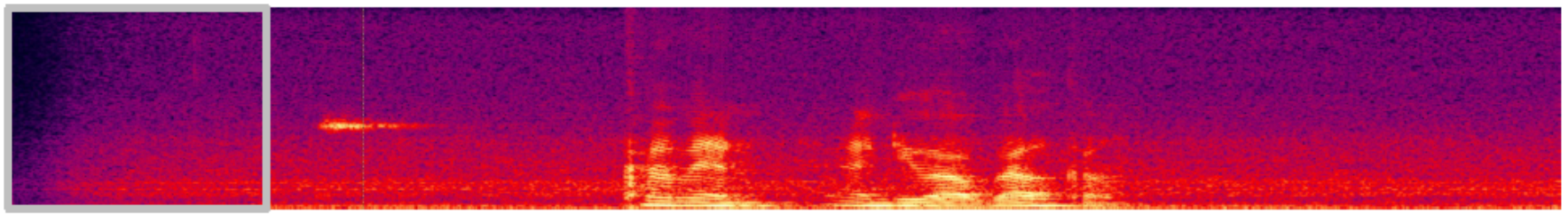


Let's do it in a different order

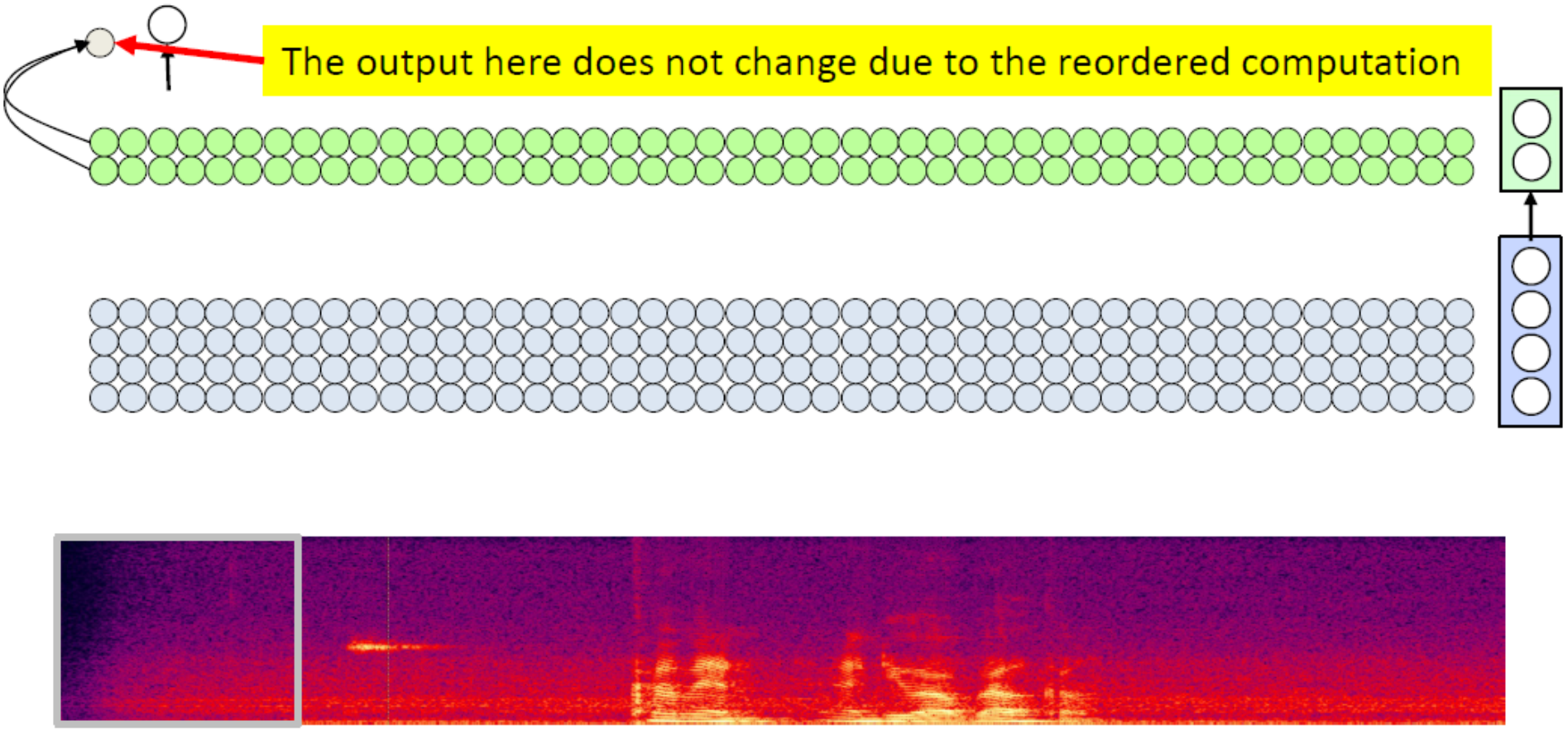


Let's do it in a different order

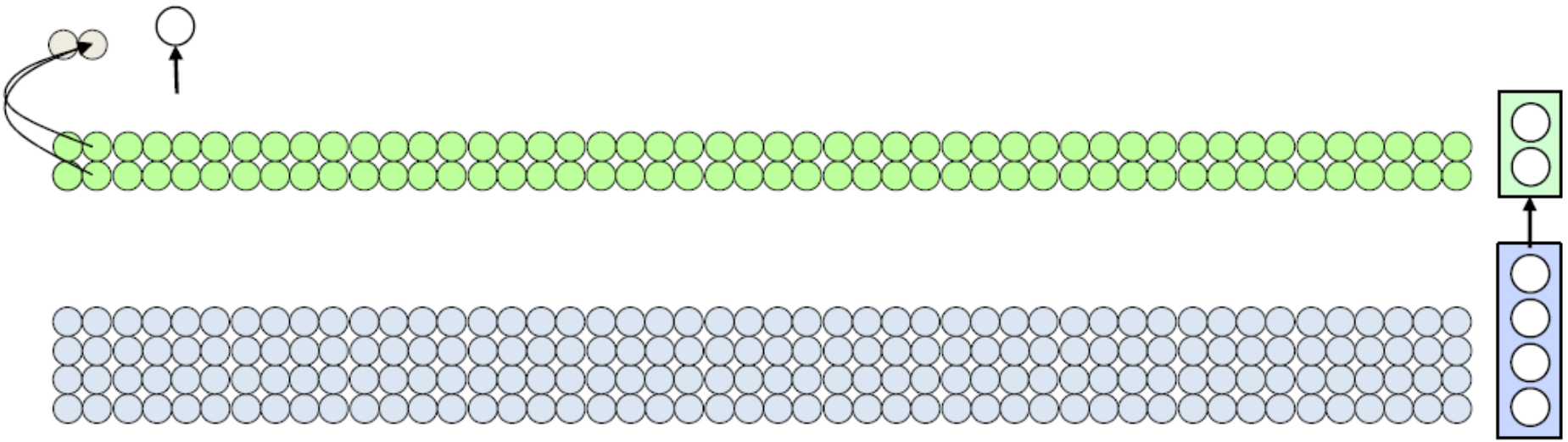


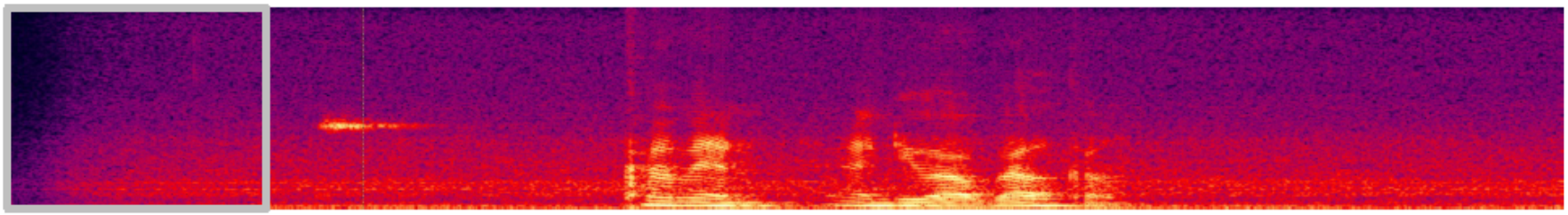


Let's do it in a different order

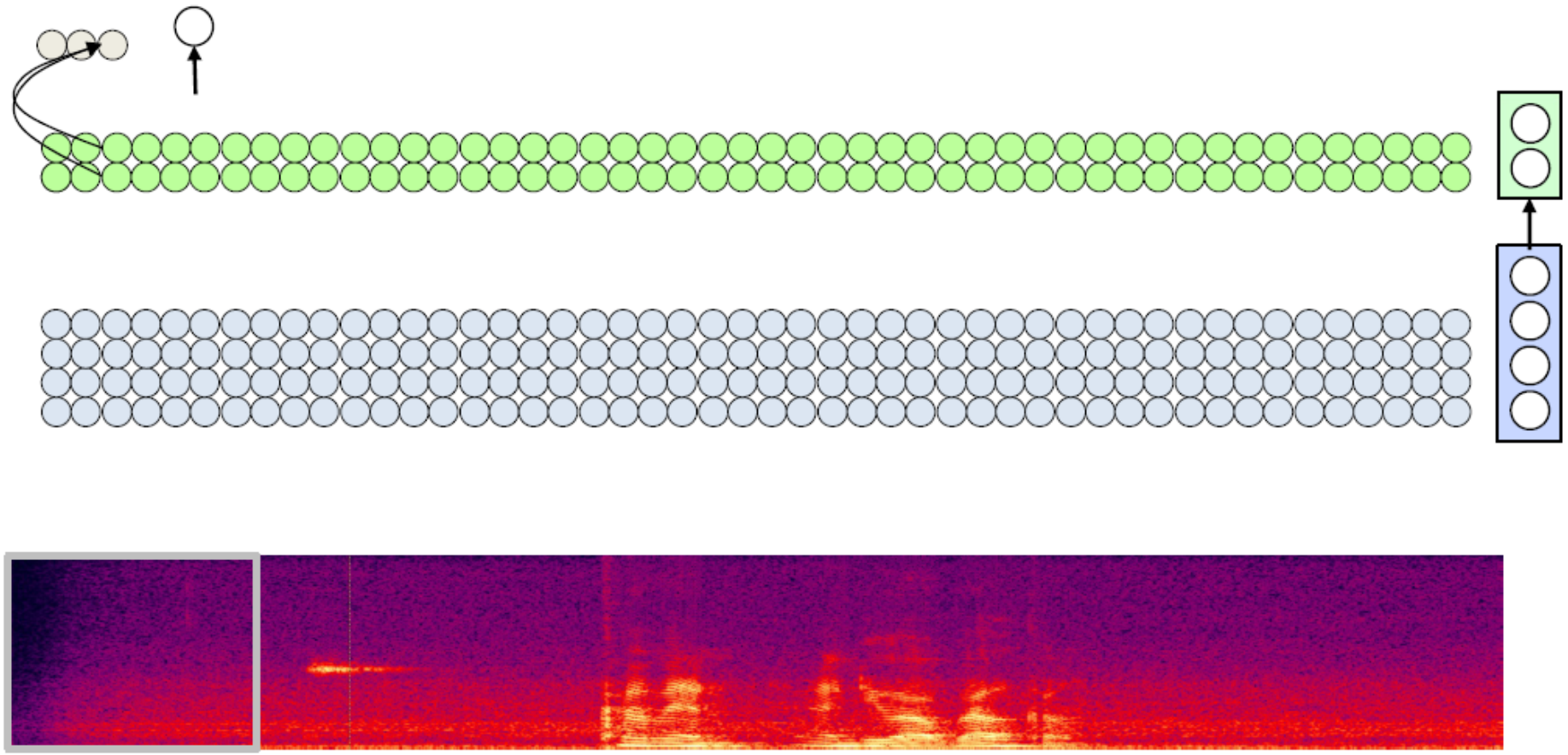


Let's do it in a different order

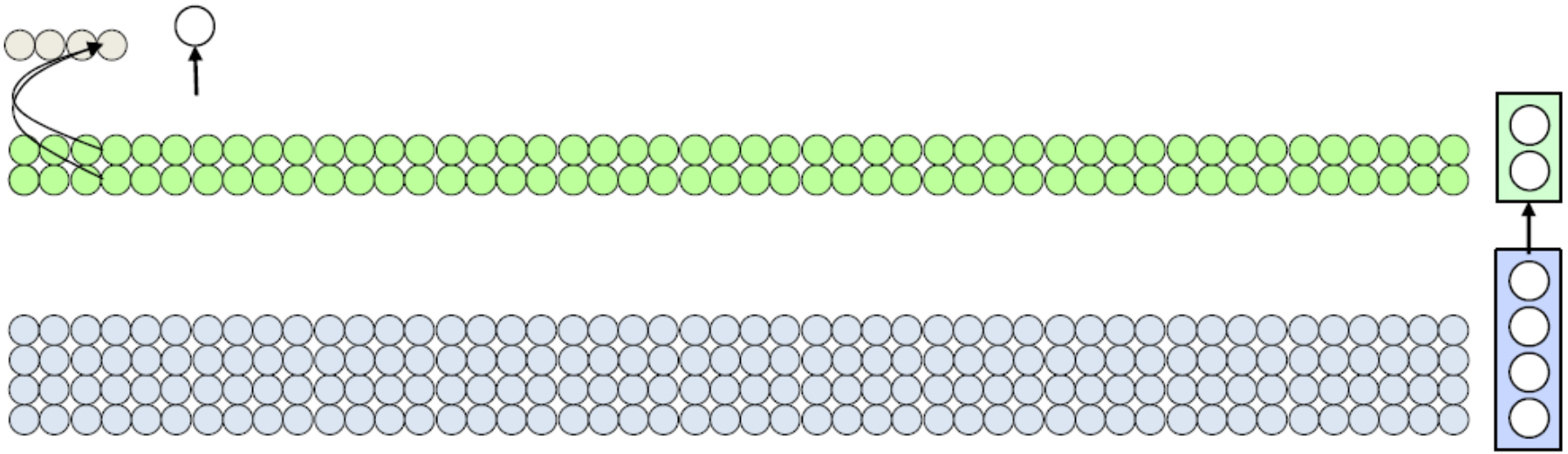


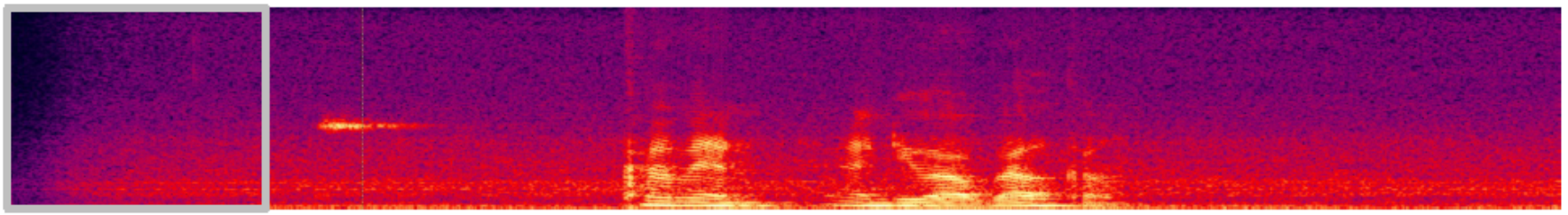


Let's do it in a different order

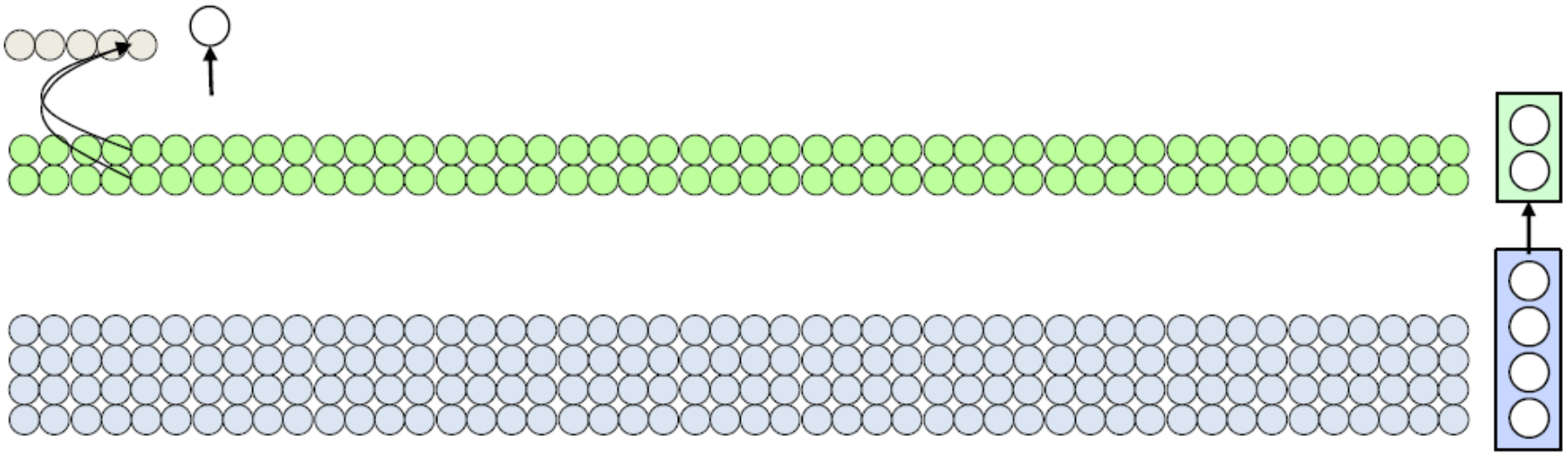


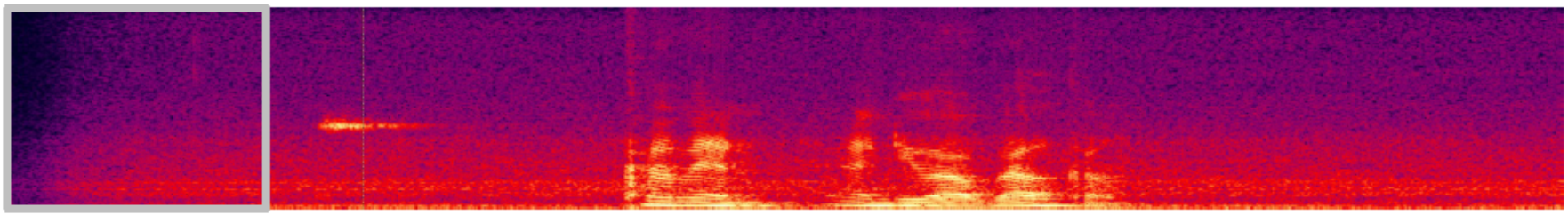
Let's do it in a different order



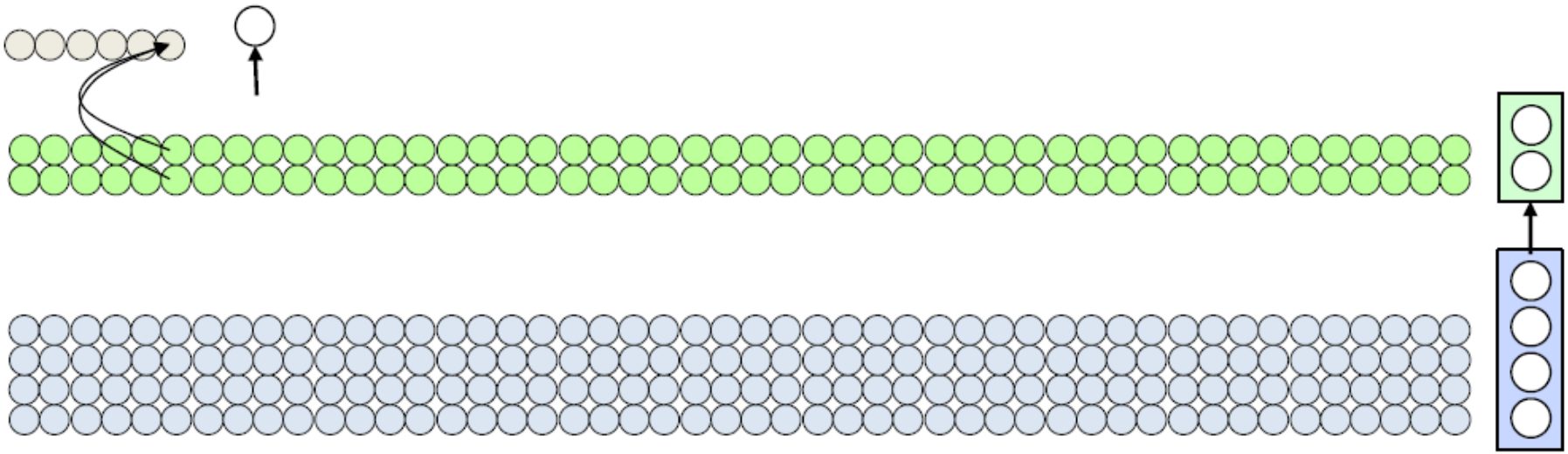


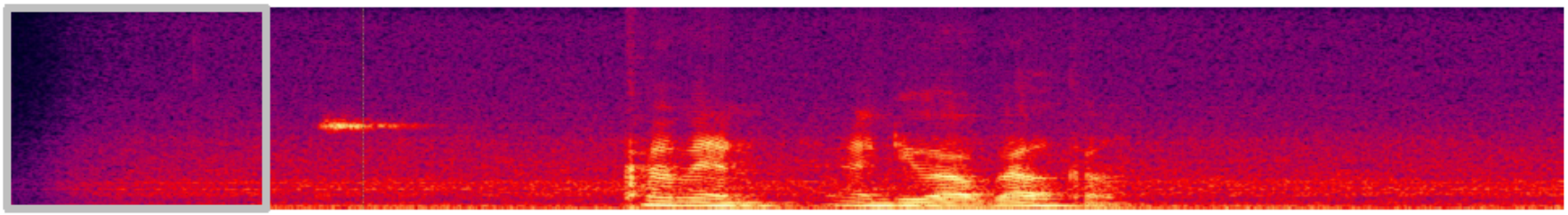
Let's do it in a different order



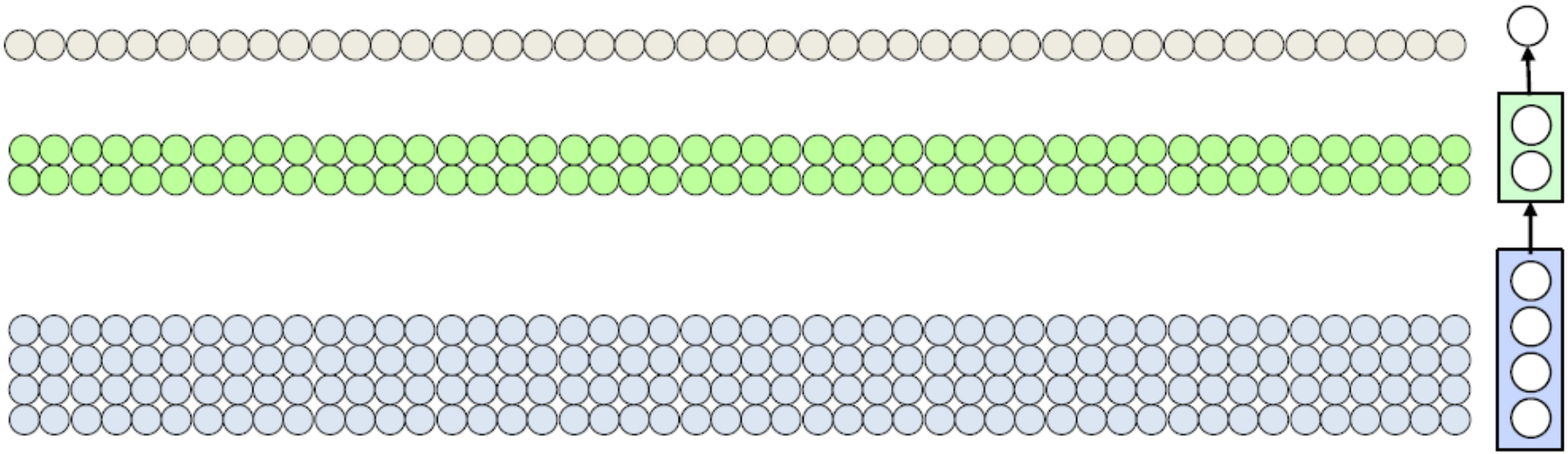


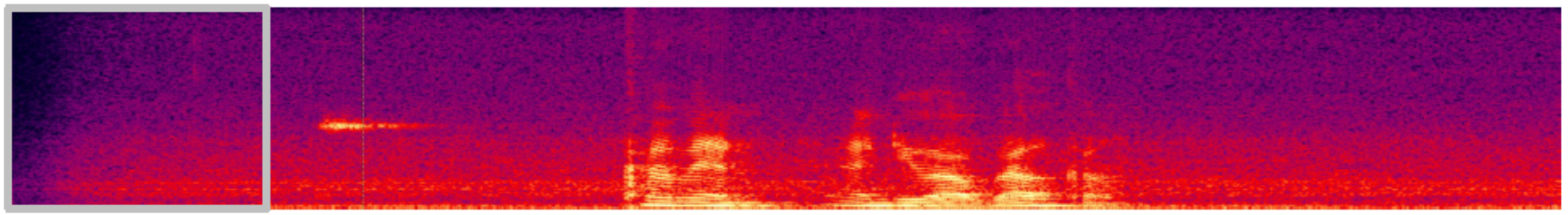
Let's do it in a different order



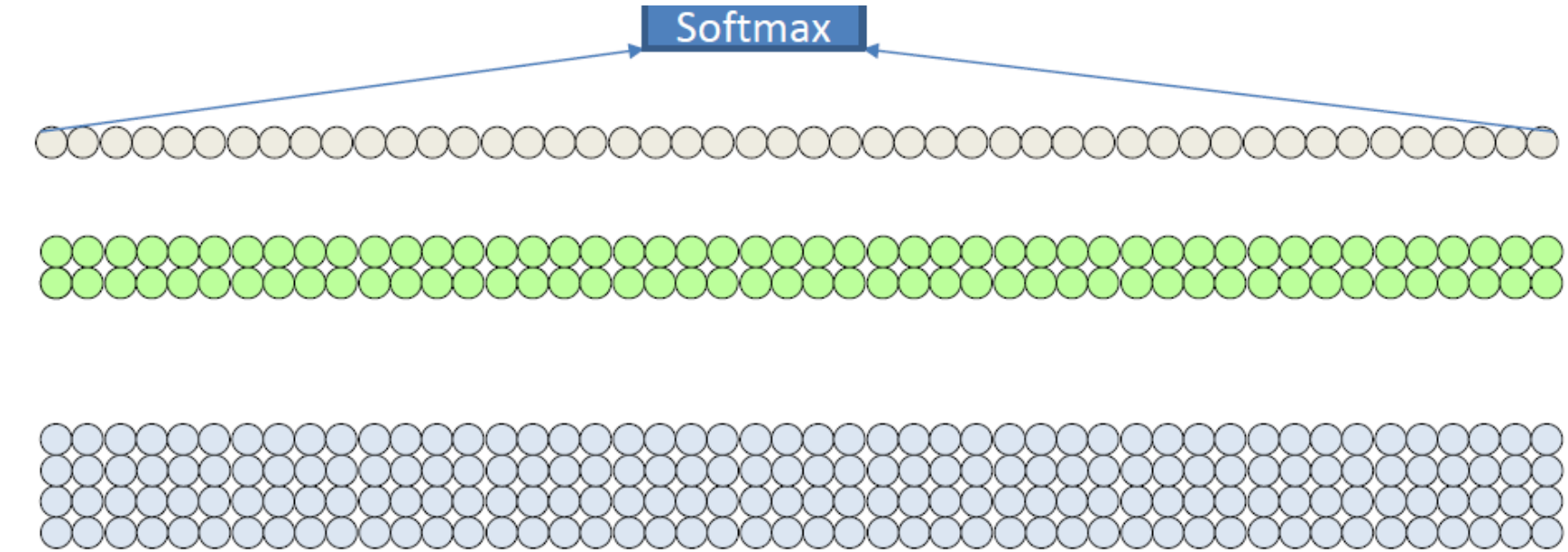


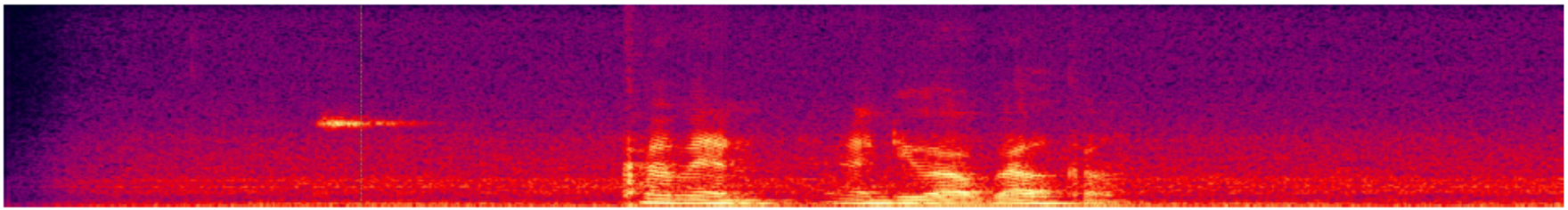
Let's do it in a different order





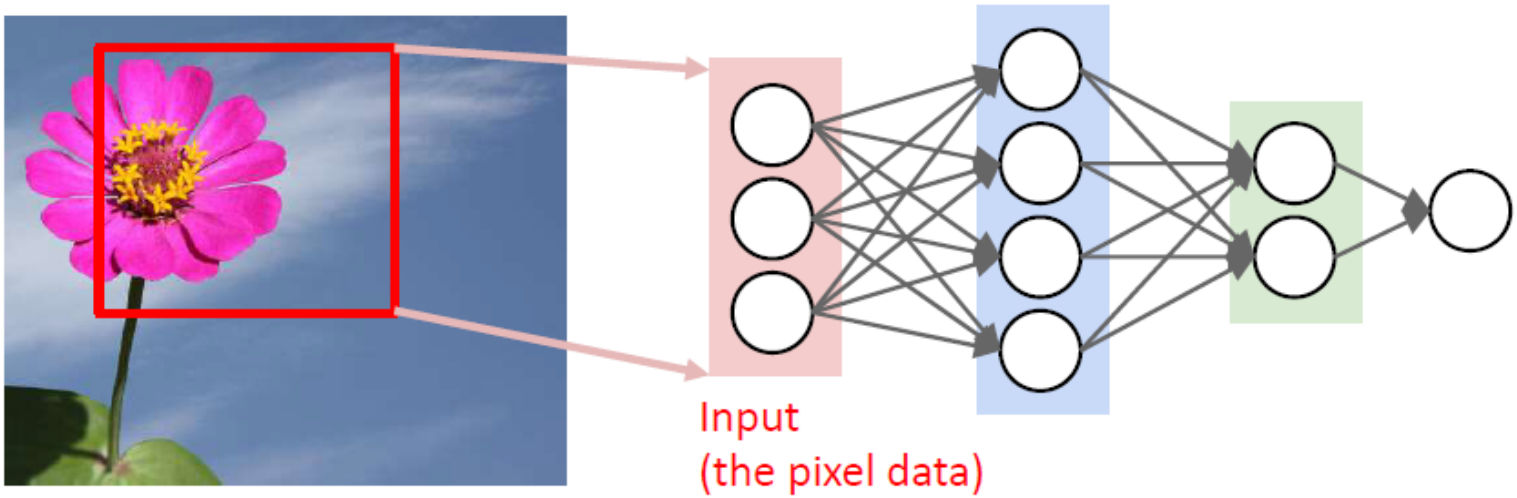
Let's do it in a different order





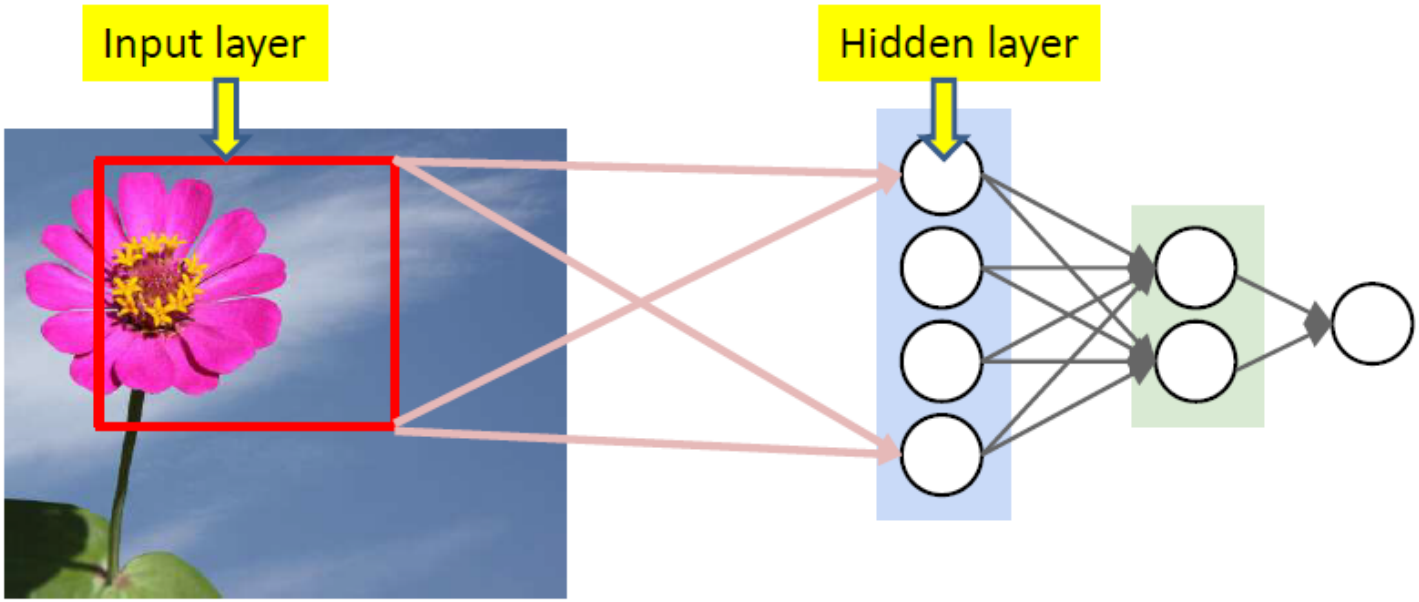
Scanning in 2D: A closer look

- Scan for the desired object
- At each location, the entire region is sent through an MLP
- The “input layer” is just the pixels in the image connecting to the hidden layer



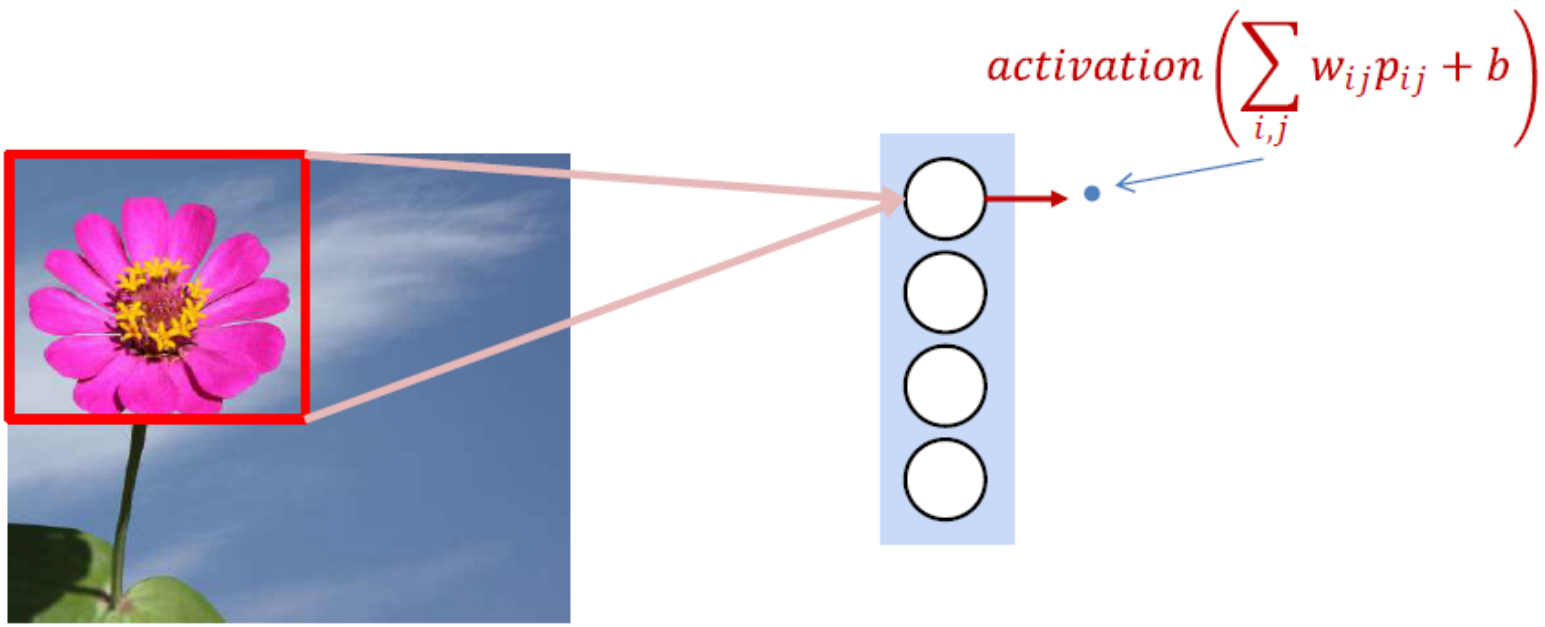
Scanning in 2D: A closer look

- Scan for the desired object
- At each location, the entire region is sent through an MLP
- The “input layer” is just the pixels in the image connecting to the hidden layer



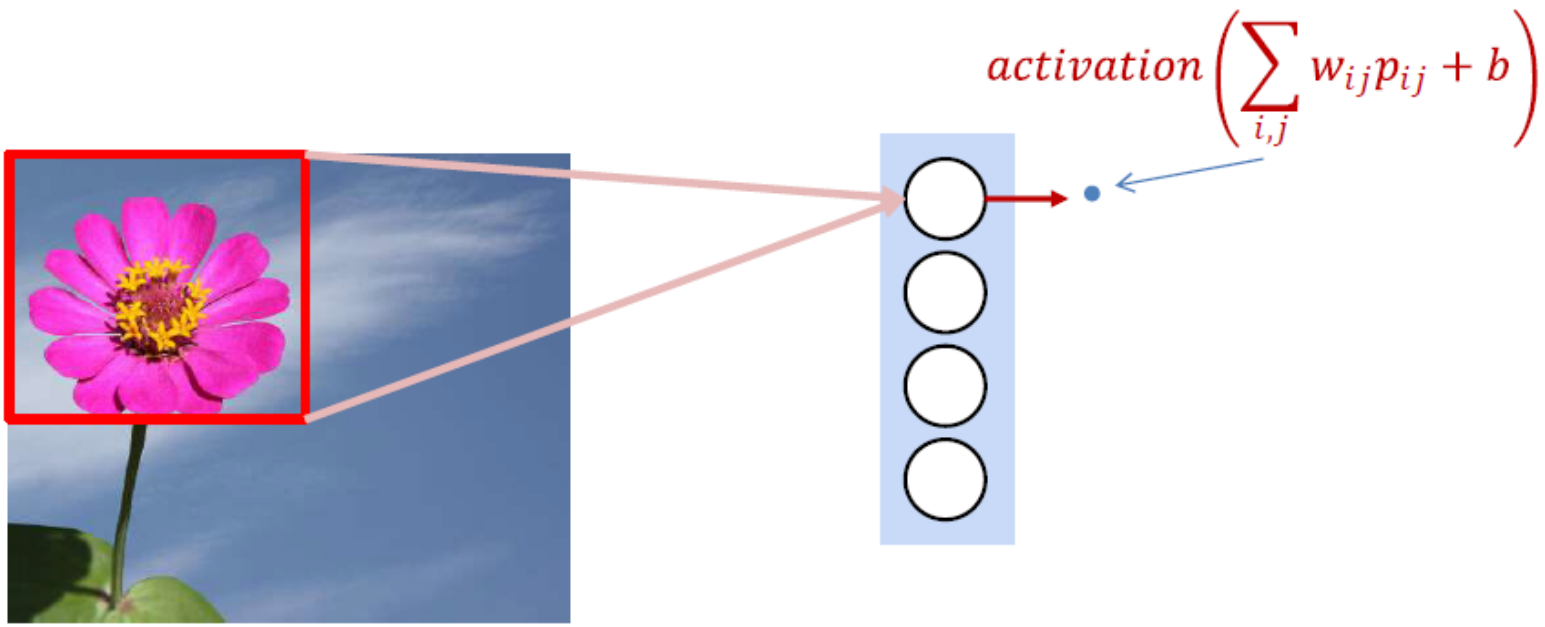
Scanning in 2D: A closer look

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



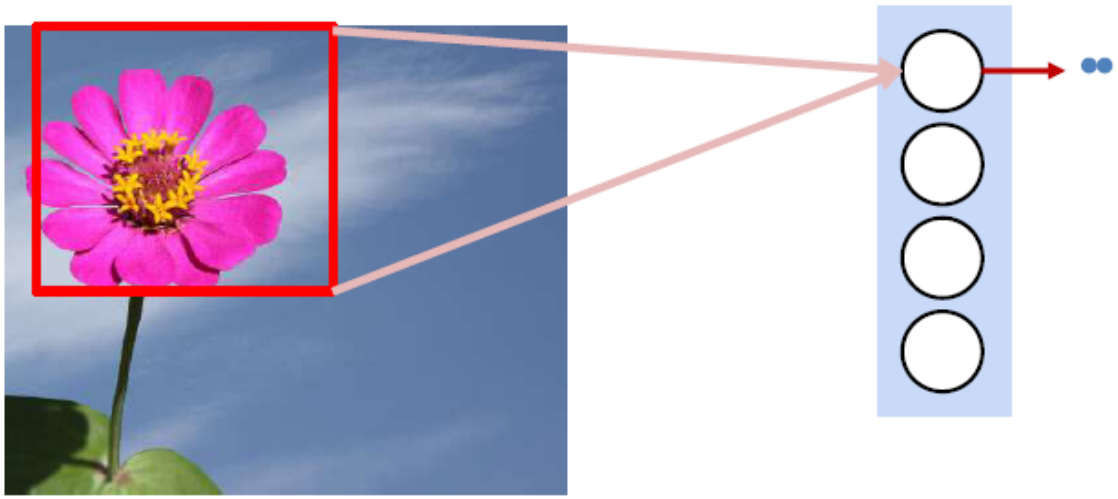
Scanning in 2D: A closer look

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



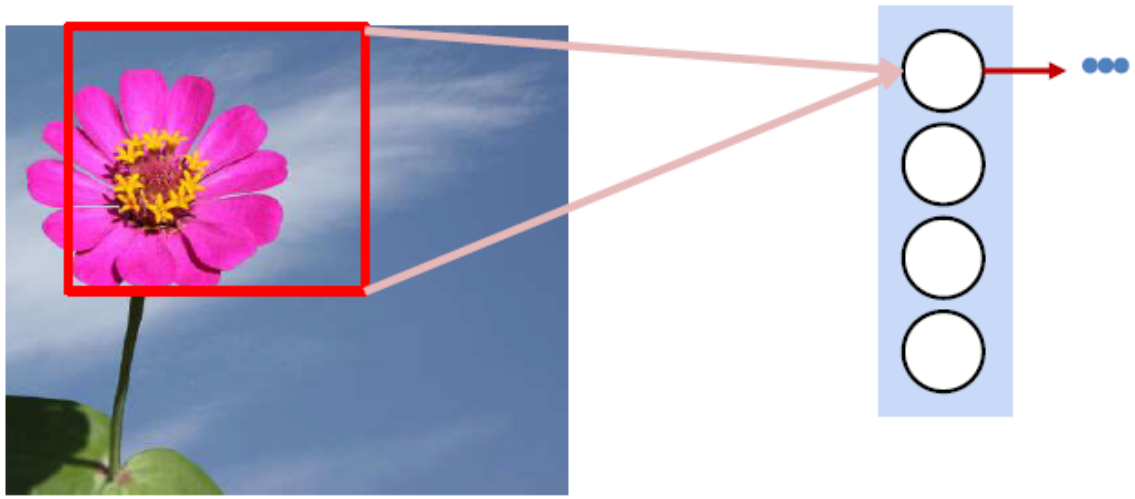
Scanning in 2D: A closer look

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



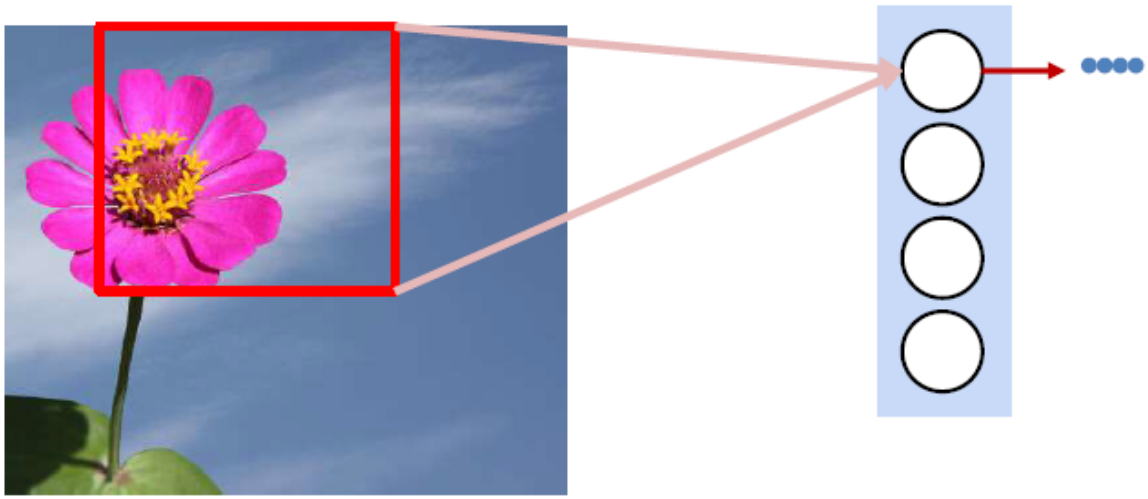
Scanning in 2D: A closer look

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



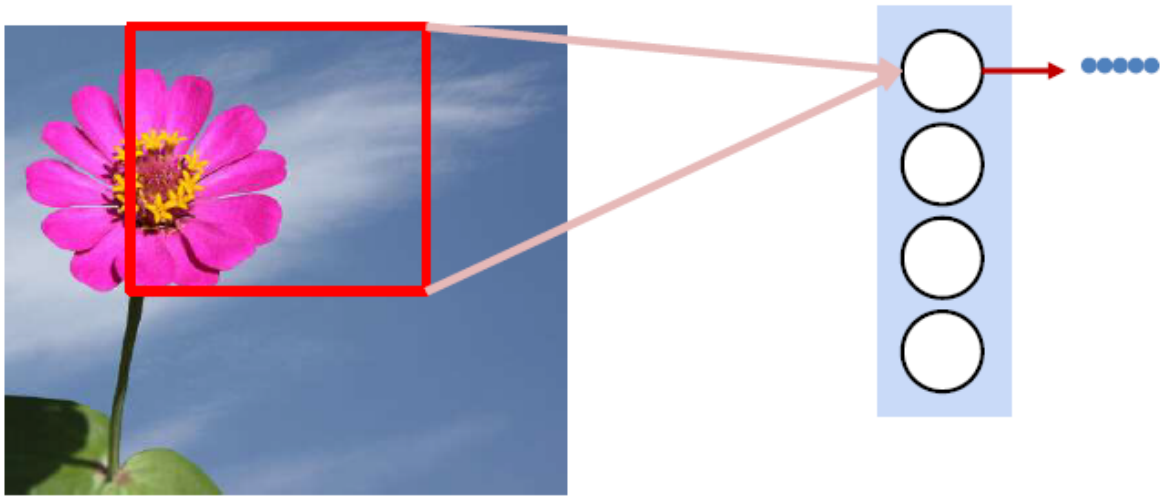
Scanning in 2D: A closer look

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



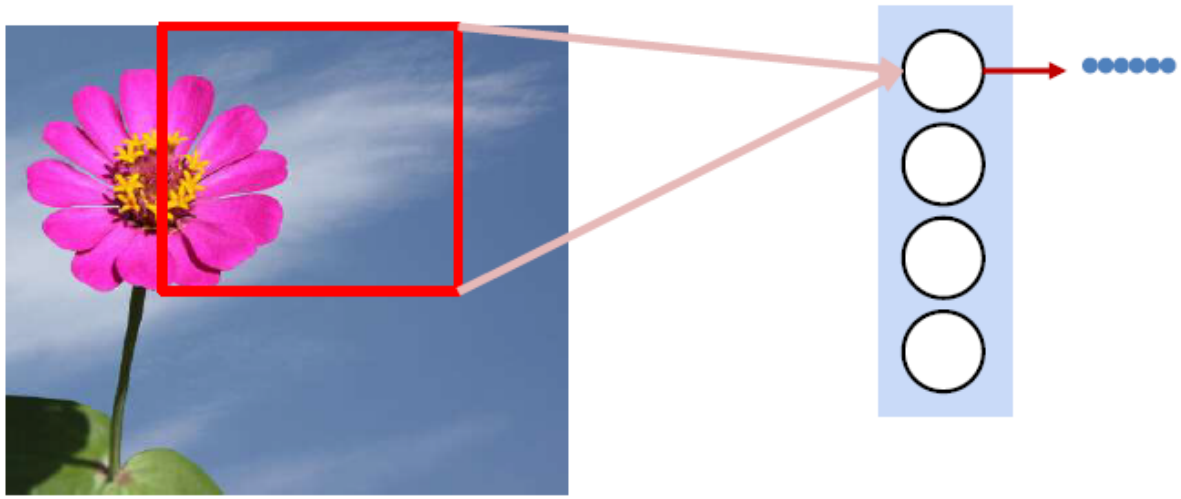
Scanning in 2D: A closer look

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



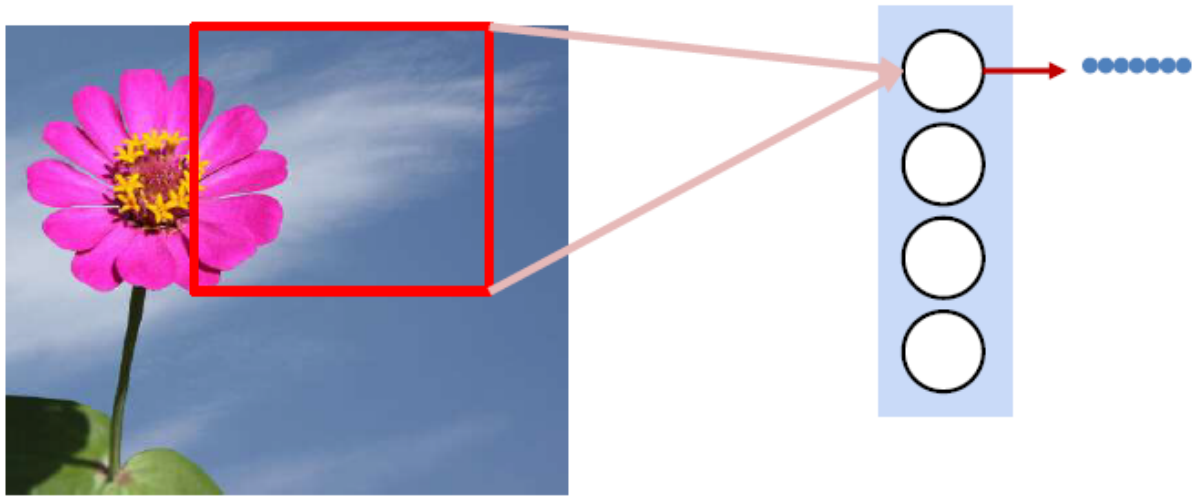
Scanning in 2D: A closer look

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



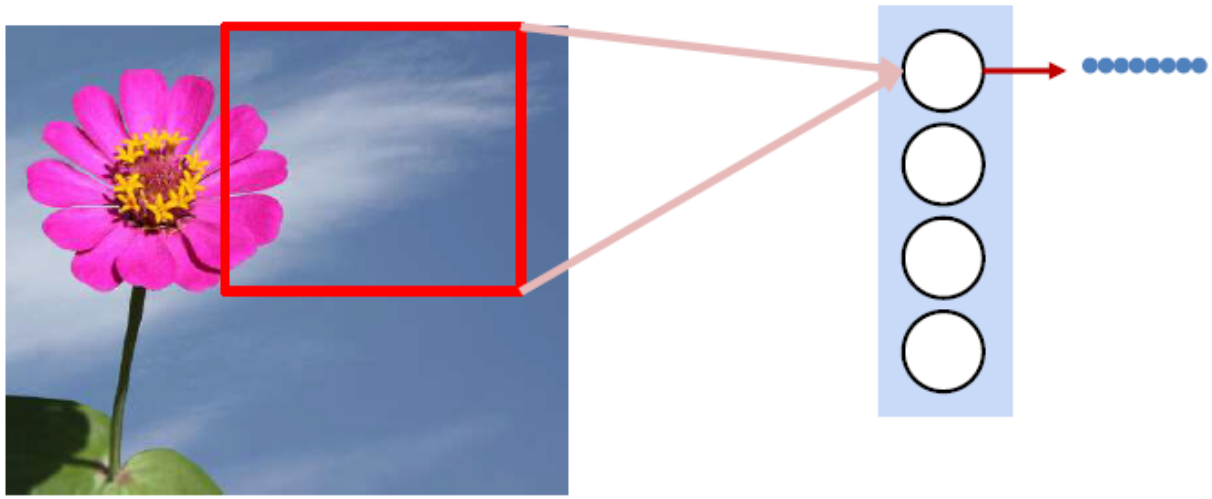
Scanning in 2D: A closer look

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



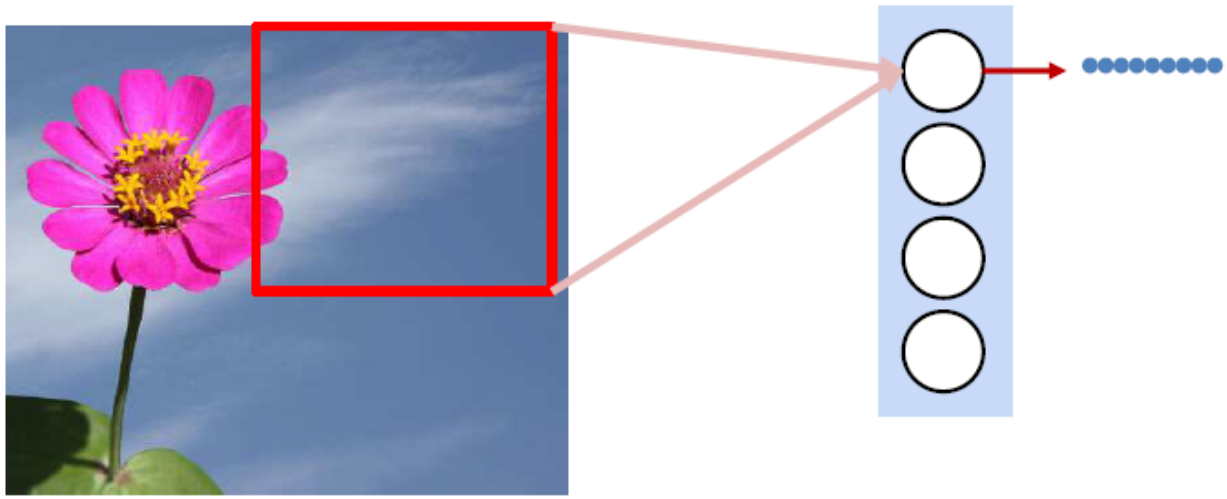
Scanning in 2D: A closer look

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



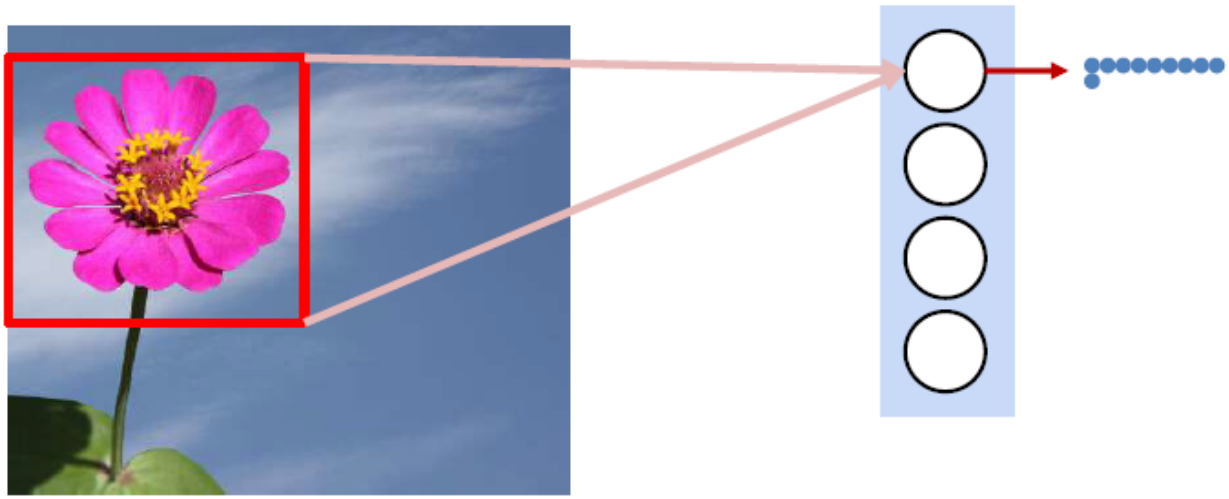
Scanning in 2D: A closer look

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



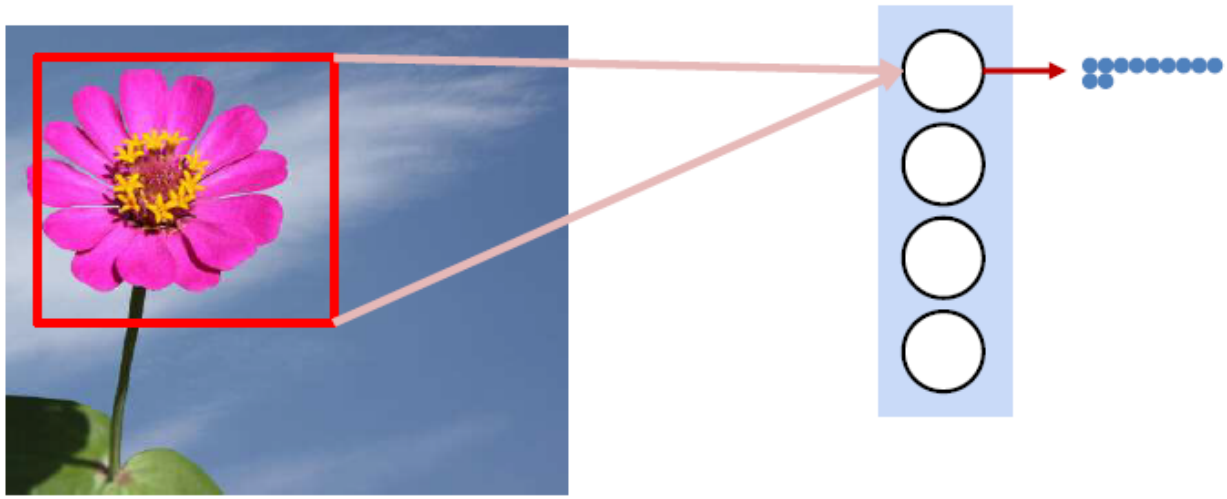
Scanning in 2D: A closer look

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



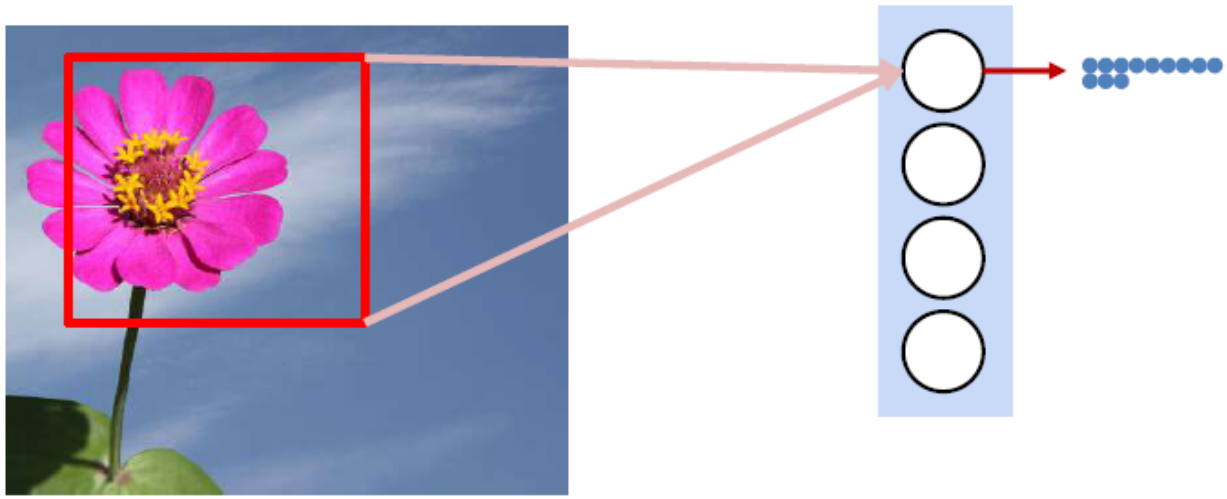
Scanning in 2D: A closer look

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



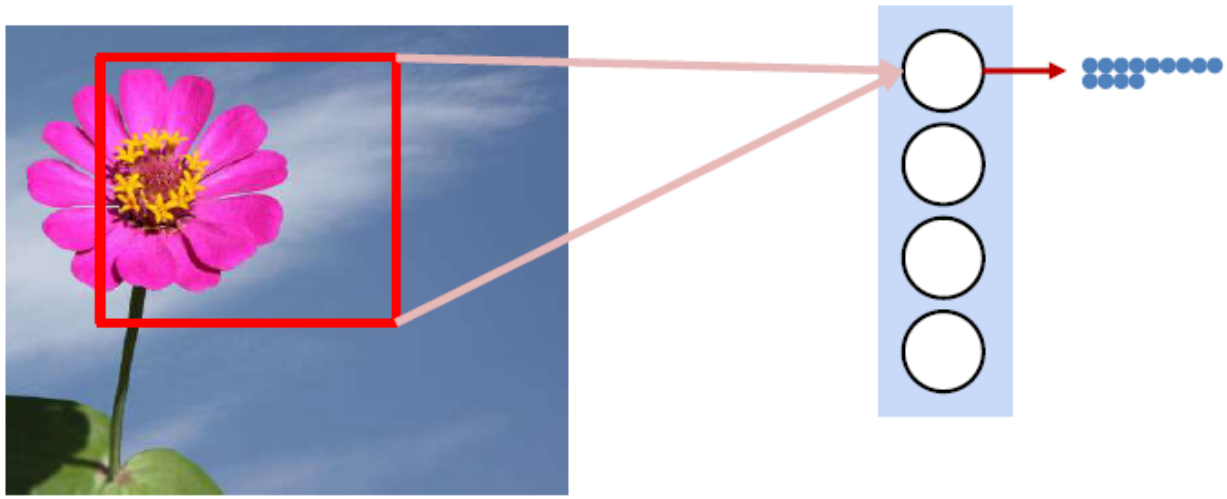
Scanning in 2D: A closer look

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



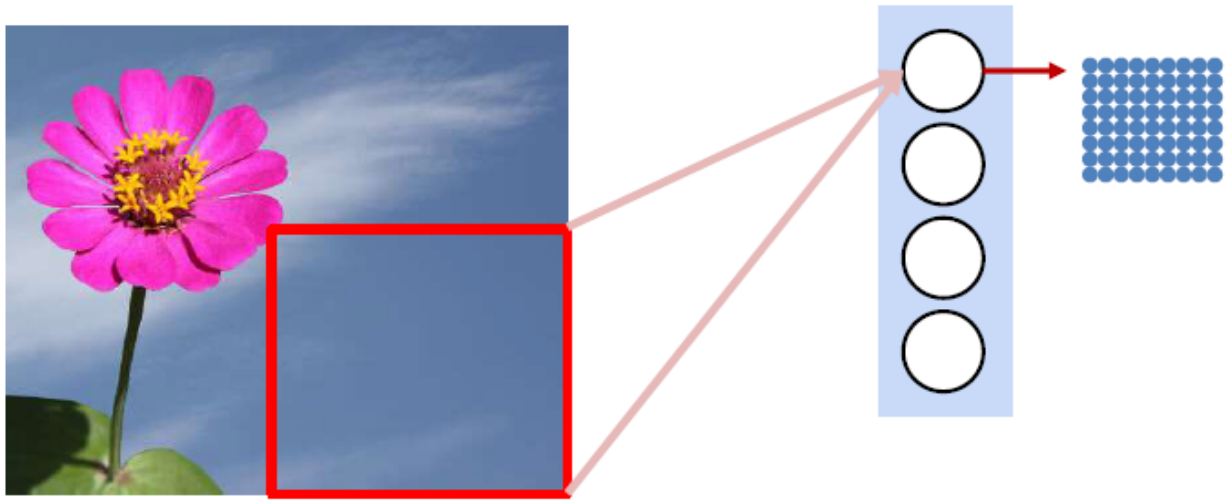
Scanning in 2D: A closer look

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



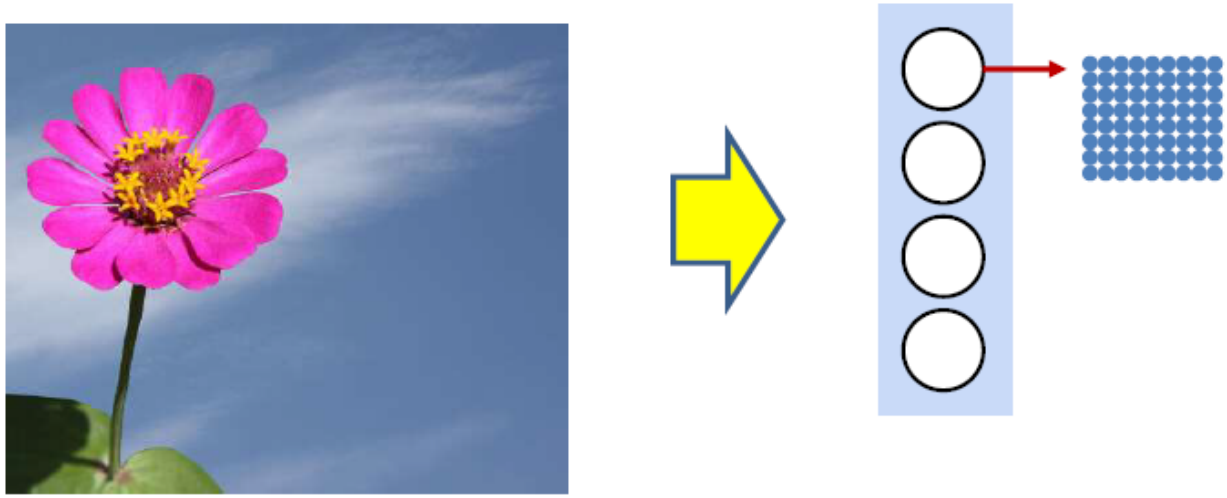
Scanning in 2D: A closer look

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



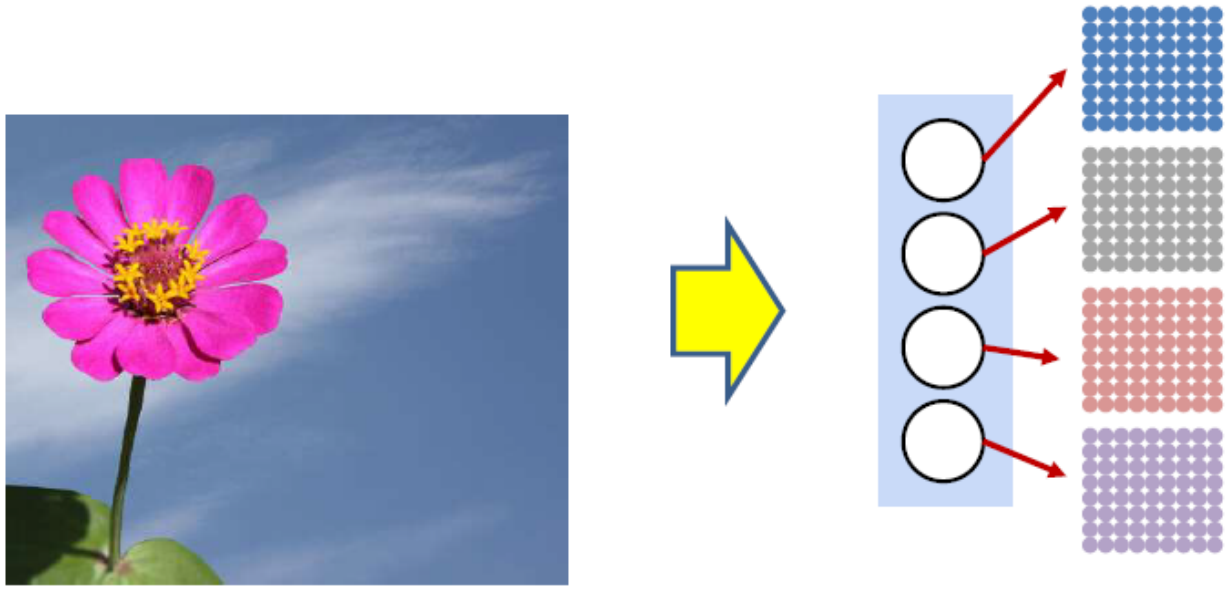
Scanning in 2D: A closer look

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



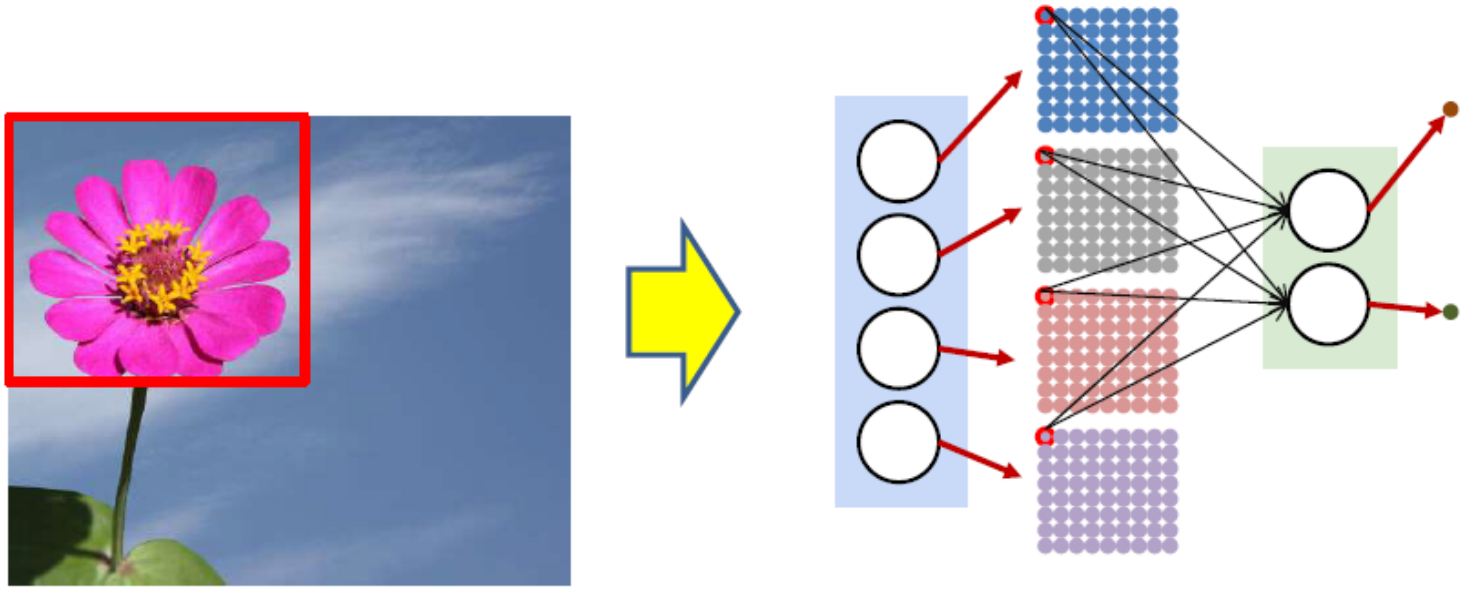
Scanning in 2D: A closer look

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



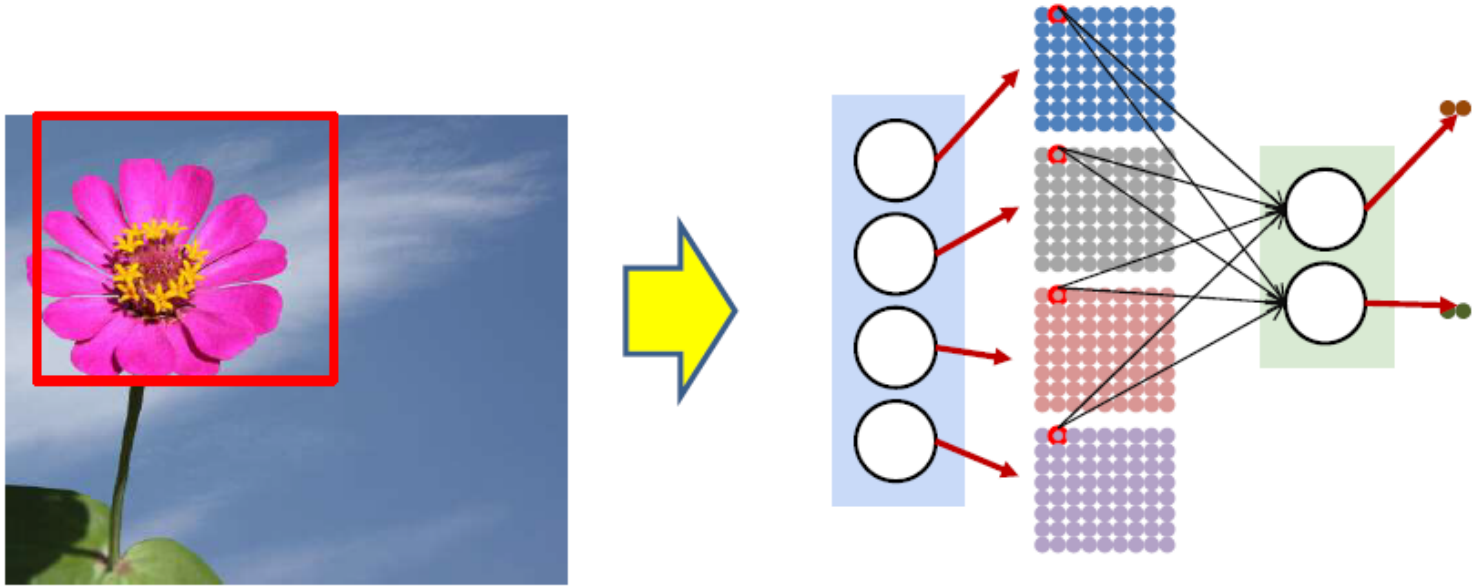
Scanning in 2D: A closer look

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



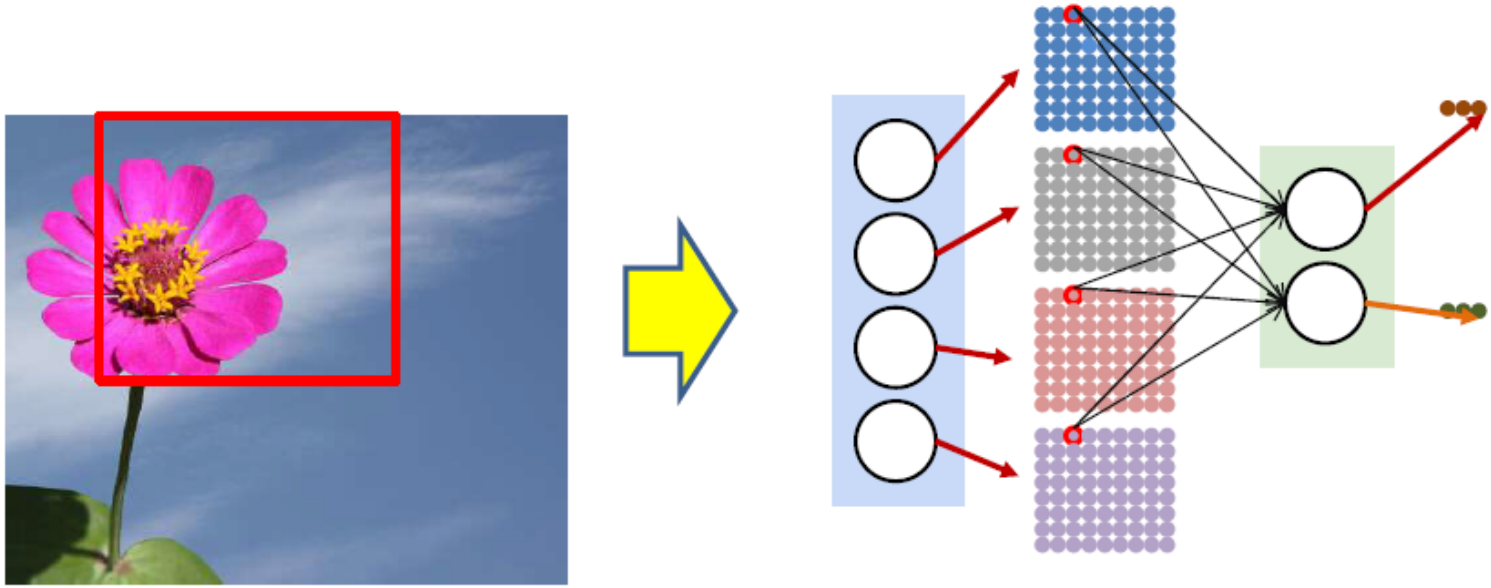
Scanning in 2D: A closer look

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



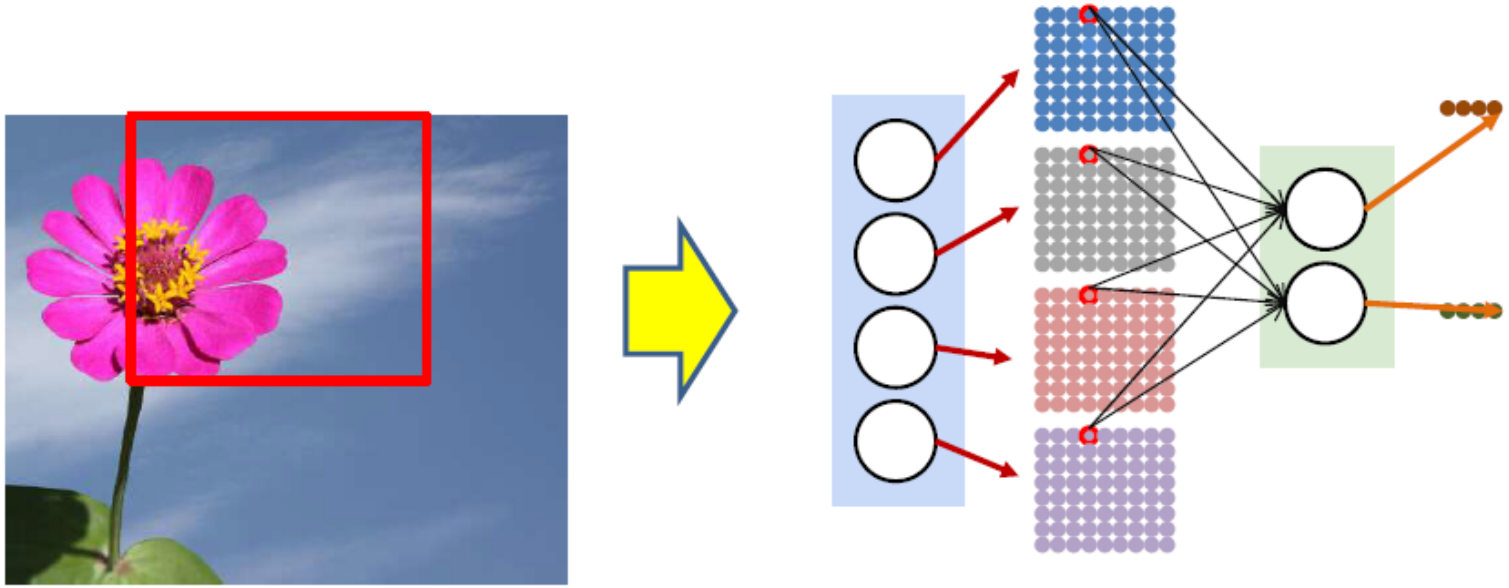
Scanning in 2D: A closer look

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



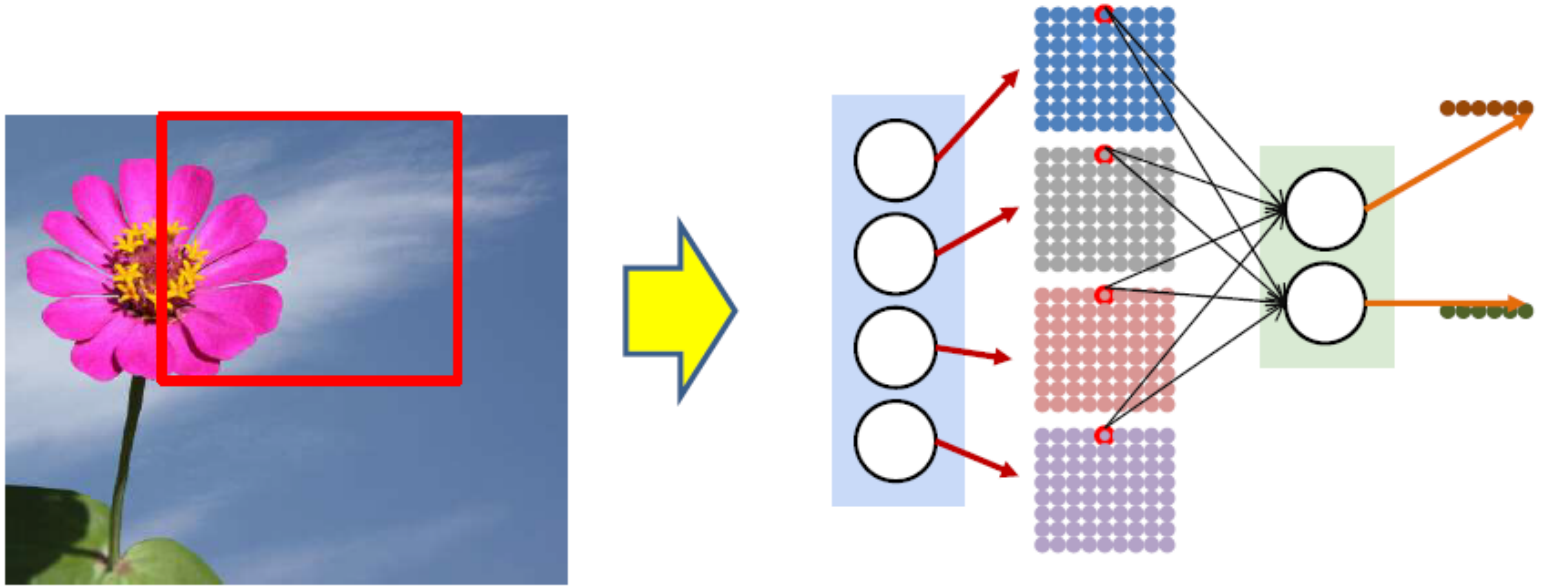
Scanning in 2D: A closer look

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



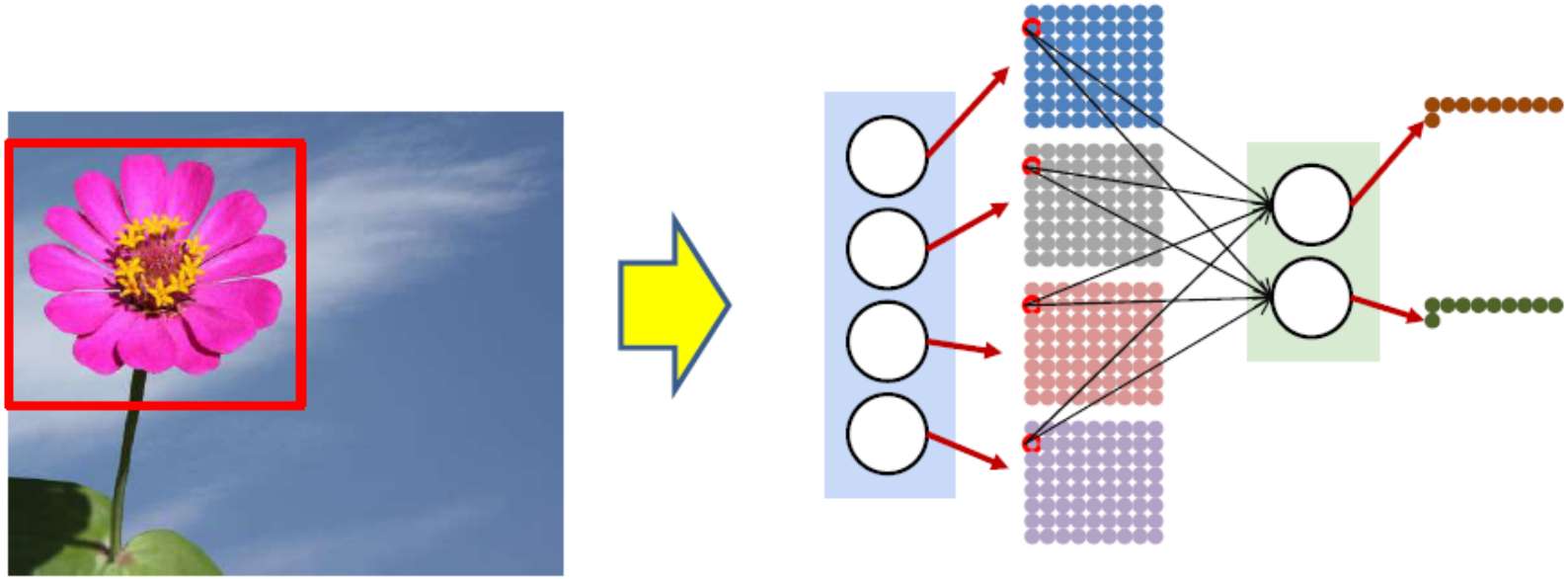
Scanning in 2D: A closer look

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



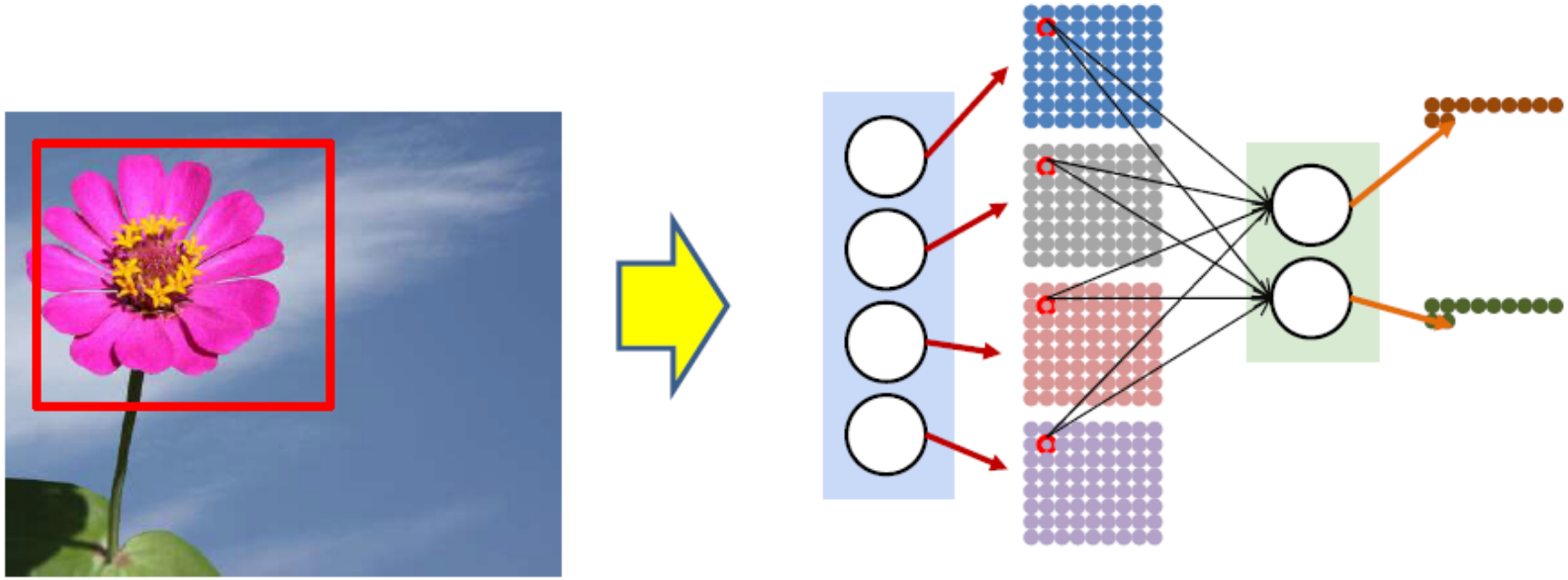
Scanning in 2D: A closer look

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



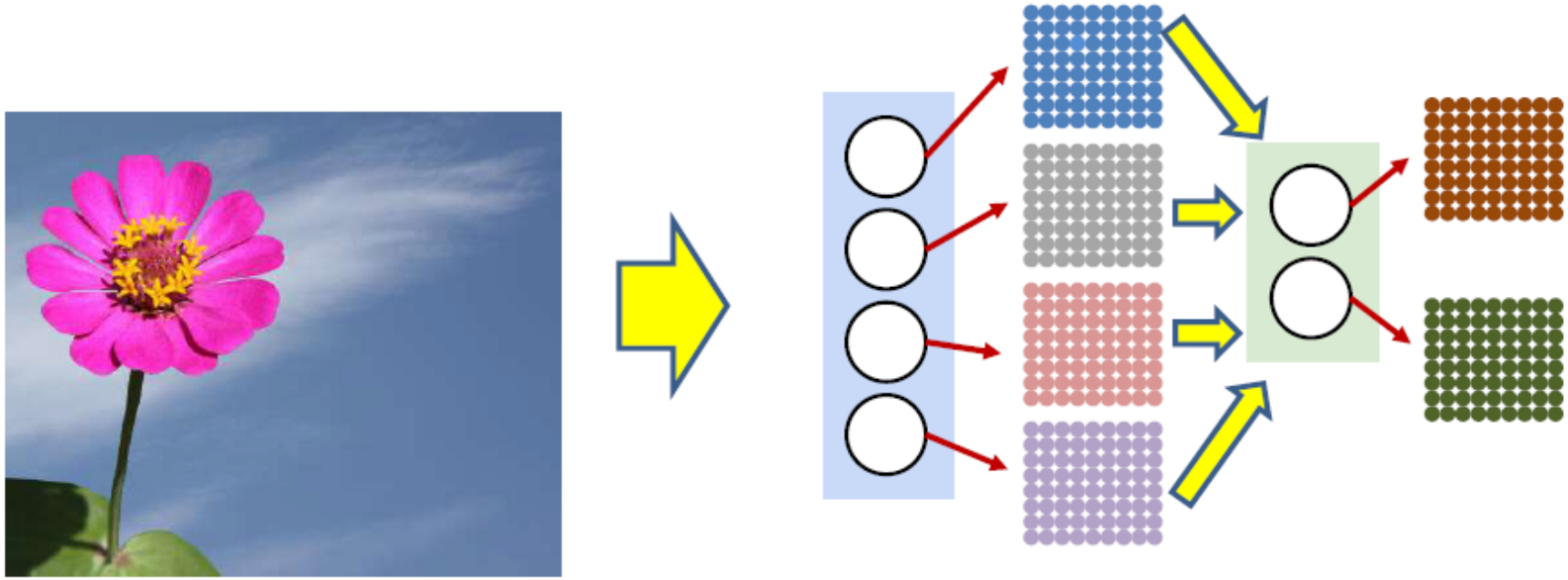
Scanning in 2D: A closer look

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



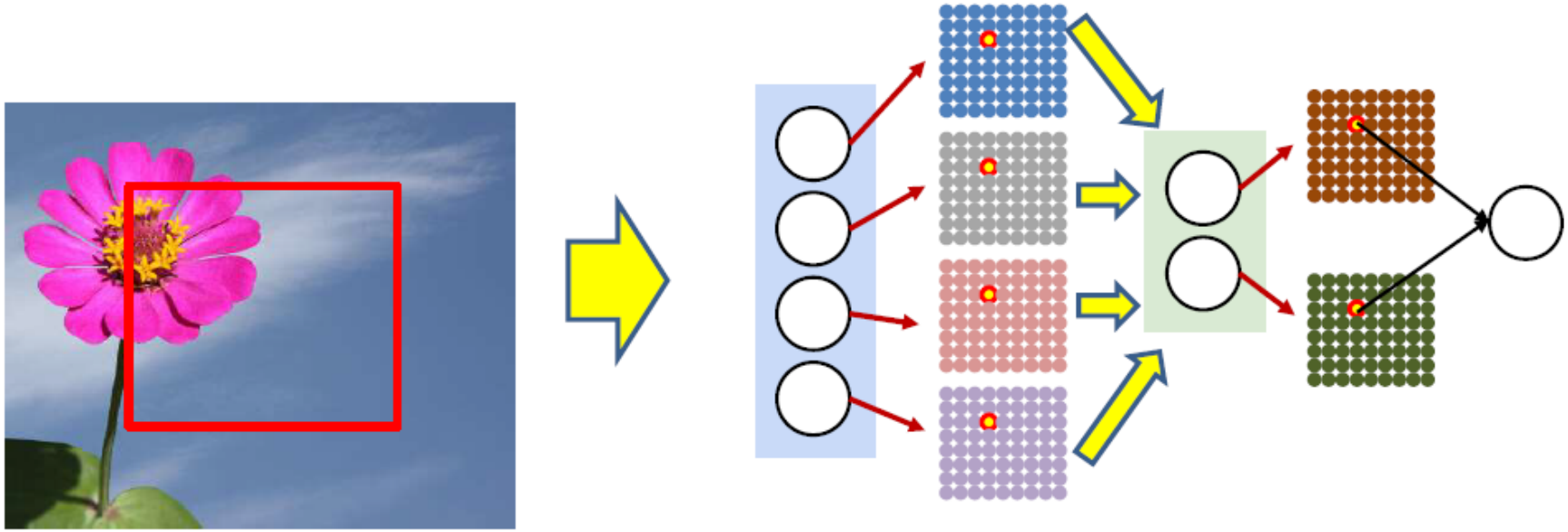
Scanning in 2D: A closer look

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



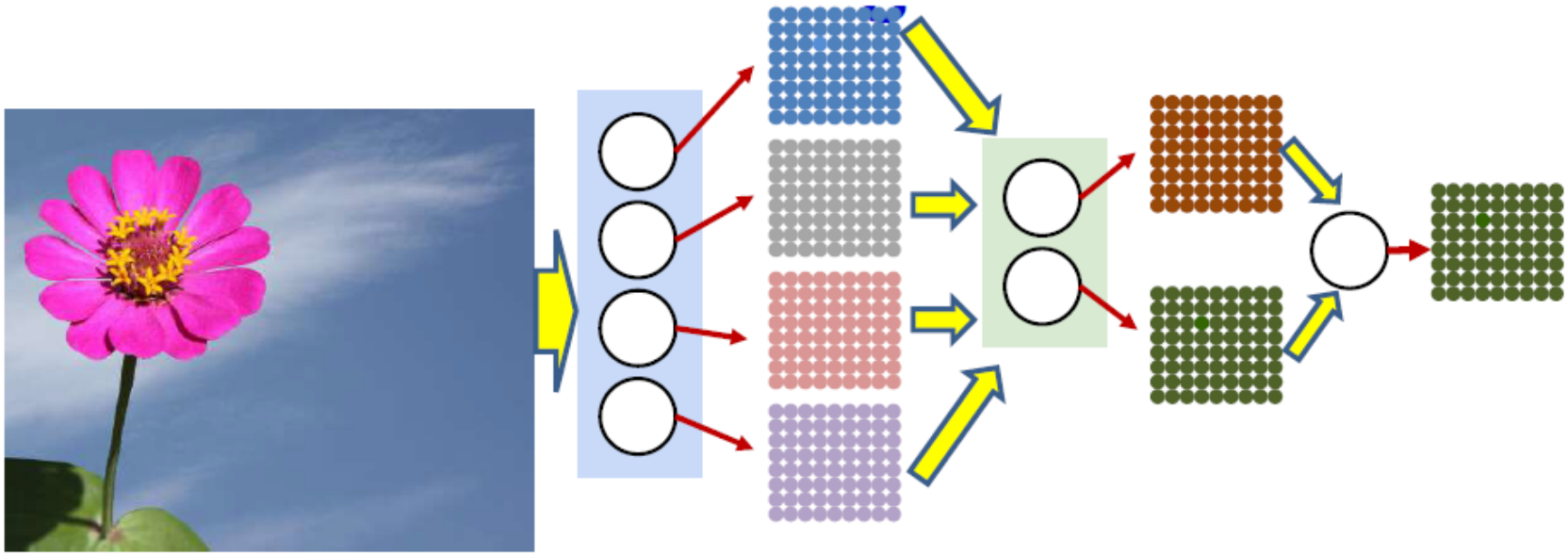
Scanning in 2D: A closer look

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



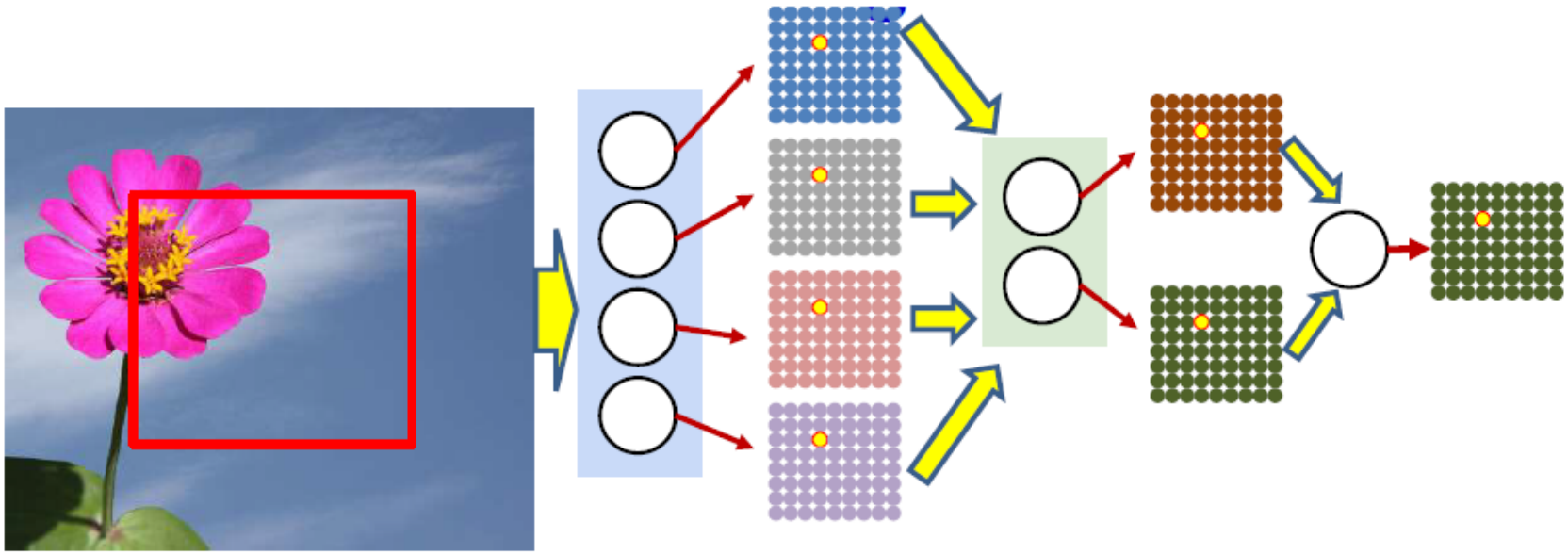
Detecting a picture anywhere in the image

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



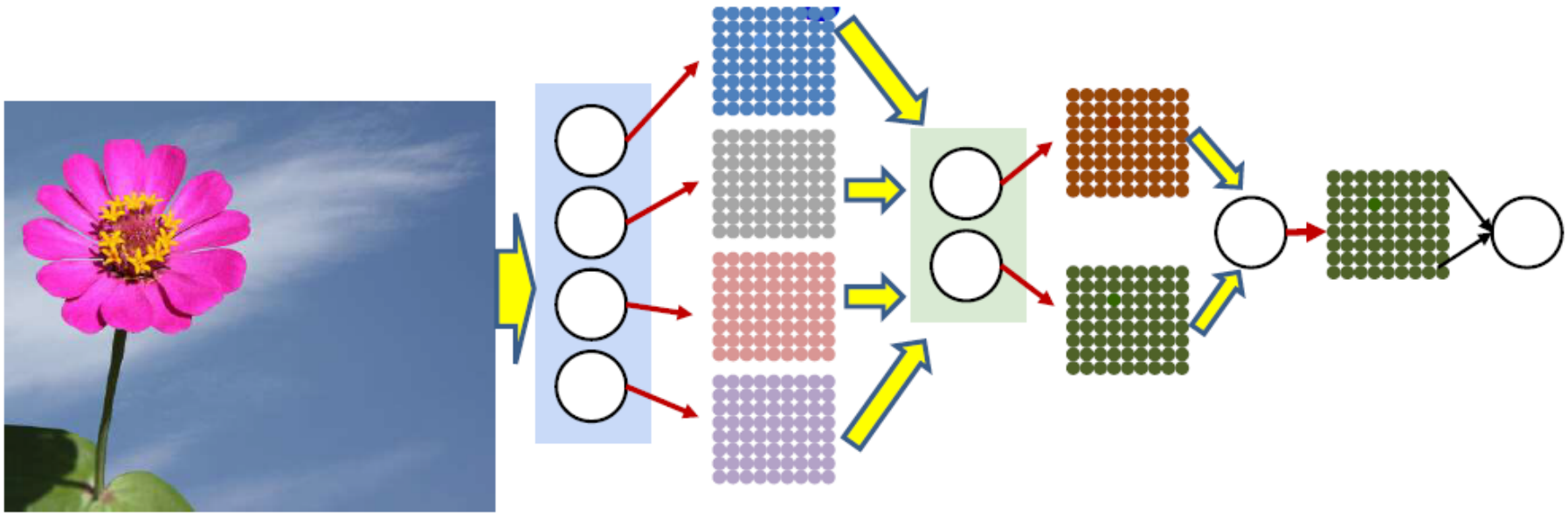
Detecting a picture anywhere in the image

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



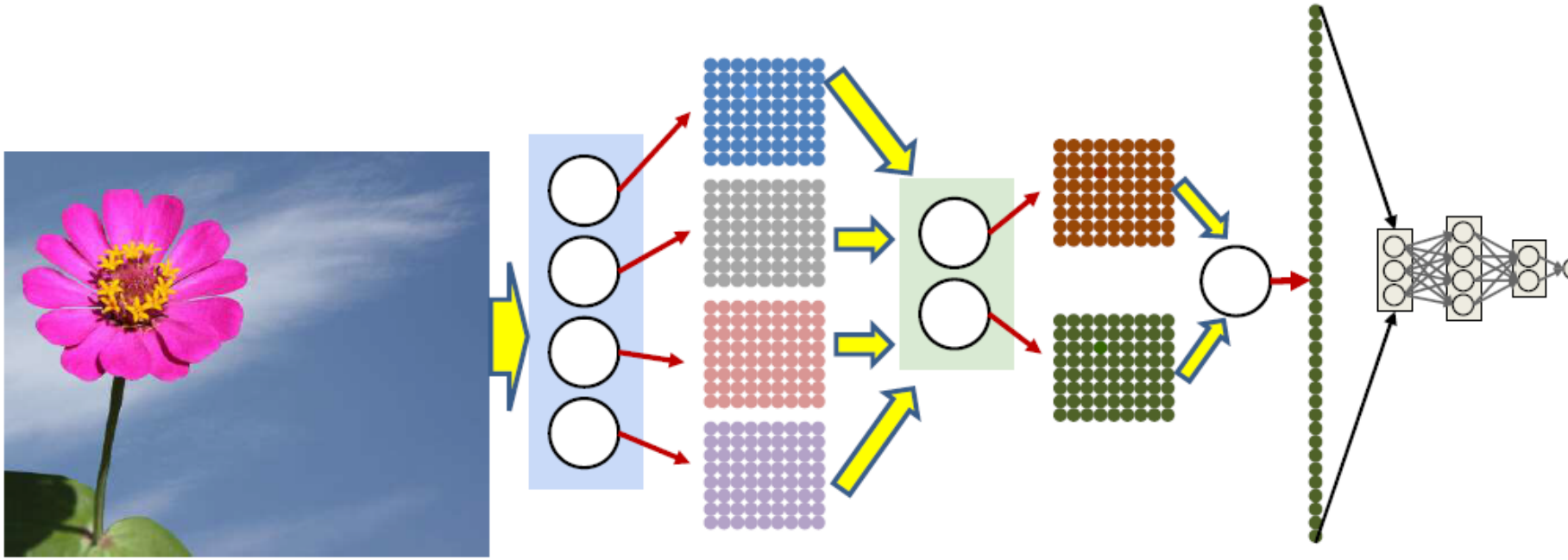
Detecting a picture anywhere in the image

- Is there a flower in the picture?
- The entire output map can be sent into a final **max** to detect a flower in the full picture
  - Or a softmax, or a full MLP. . .



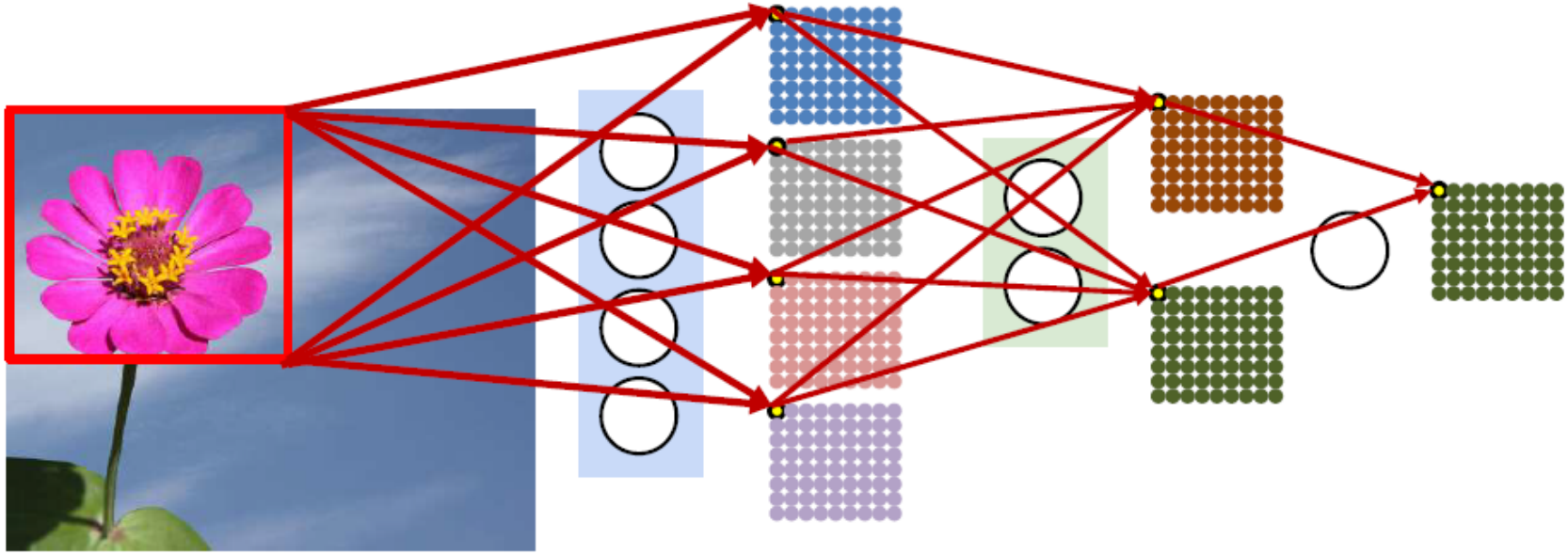
Detecting a picture anywhere in the image

- 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**



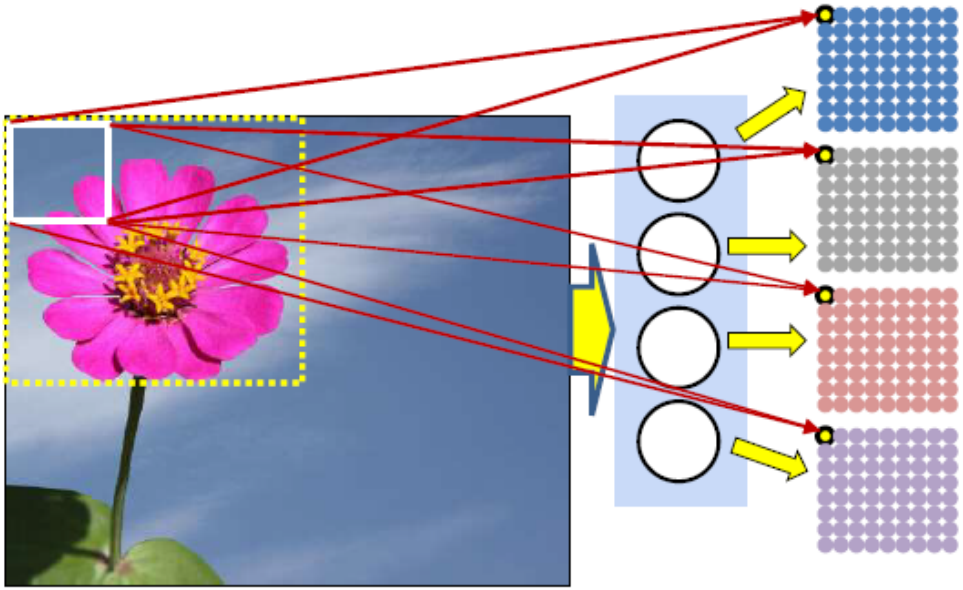
The behavior of the layers

- The first layer neurons “look” at the entire “window” to extract window level features
  - Subsequent layers only perform classification over these window-level features
- We don’t know the size of flower!
  - **Distributing the scan**



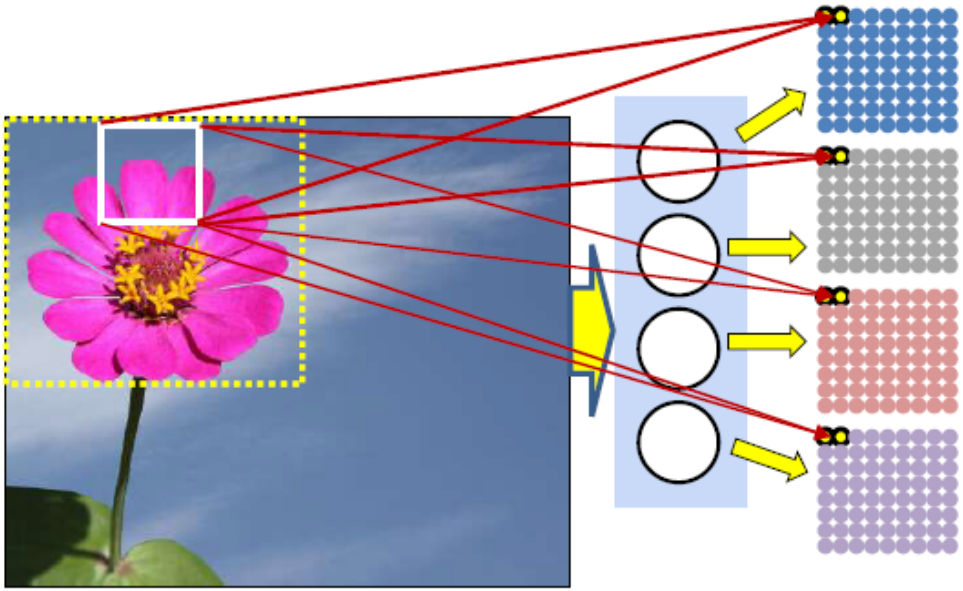
Distributing the scan

- Distribution forces localized patterns in lower layers
  - **More generalizable**
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates windows of outputs from the first layer
- The higher layer implicitly learns the arrangement of sub-patterns that represents the larger pattern (the flower in this case)



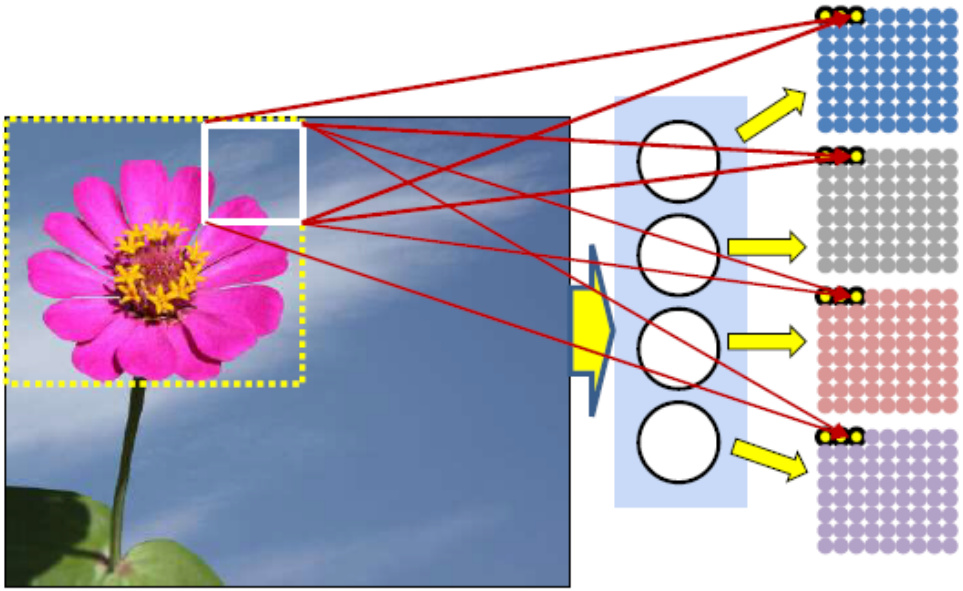
Distributing the scan

- Distribution forces localized patterns in lower layers
  - **More generalizable**
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates windows of outputs from the first layer
- The higher layer implicitly learns the arrangement of sub-patterns that represents the larger pattern (the flower in this case)



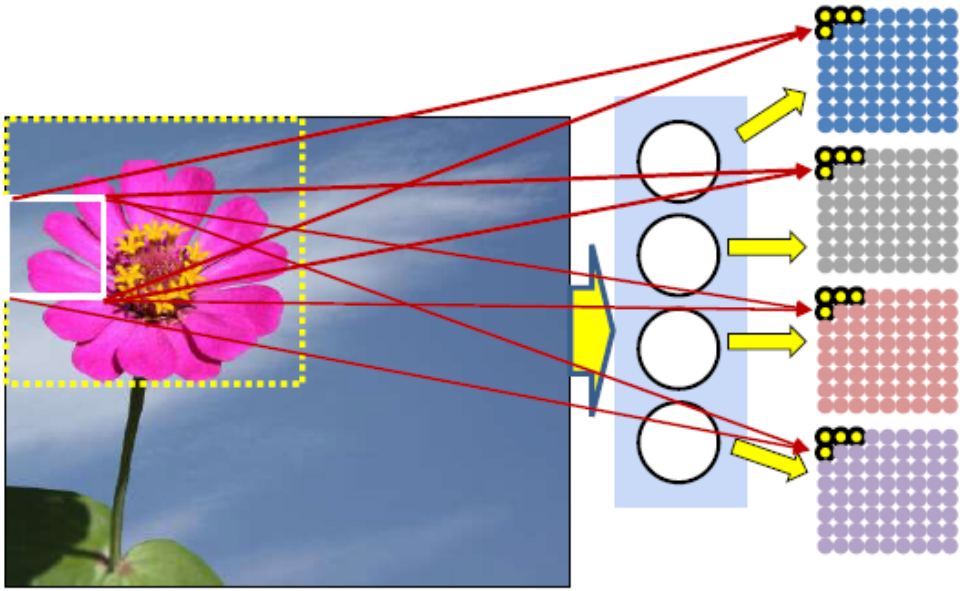
Distributing the scan

- Distribution forces localized patterns in lower layers
  - **More generalizable**
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates windows of outputs from the first layer
- The higher layer implicitly learns the arrangement of sub-patterns that represents the larger pattern (the flower in this case)



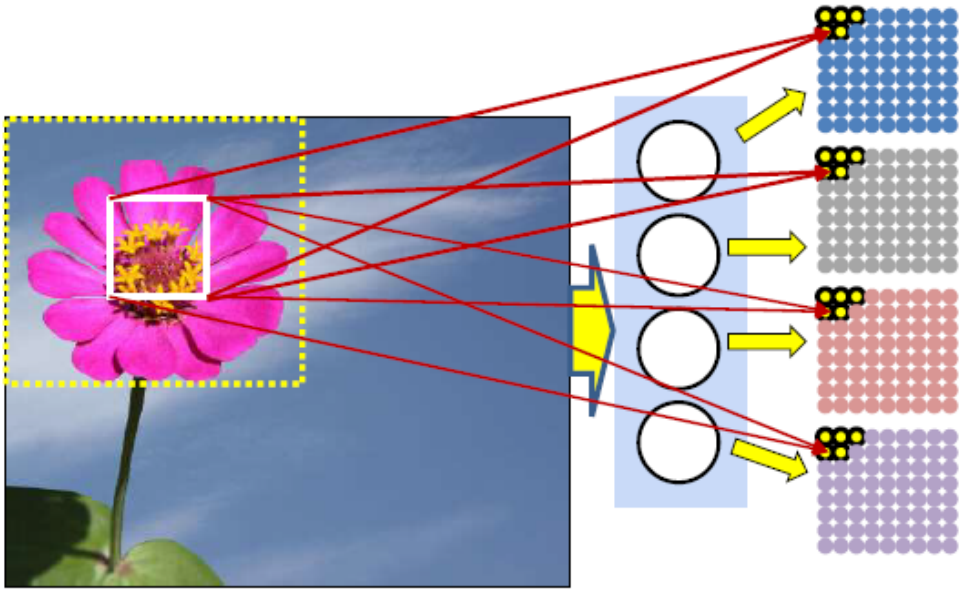
Distributing the scan

- Distribution forces localized patterns in lower layers
  - **More generalizable**
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates windows of outputs from the first layer
- The higher layer implicitly learns the arrangement of sub-patterns that represents the larger pattern (the flower in this case)



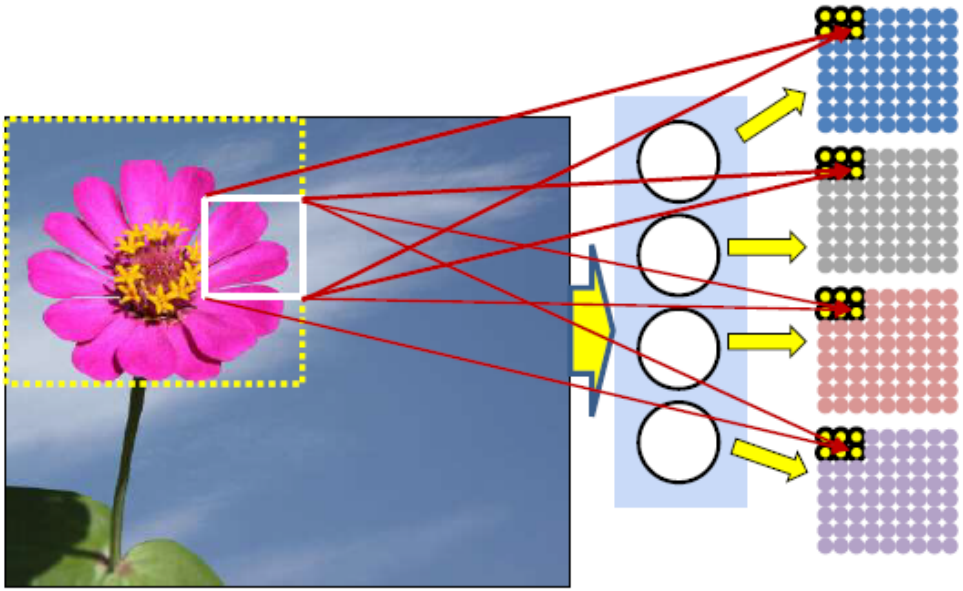
Distributing the scan

- Distribution forces localized patterns in lower layers
  - **More generalizable**
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates windows of outputs from the first layer
- The higher layer implicitly learns the arrangement of sub-patterns that represents the larger pattern (the flower in this case)



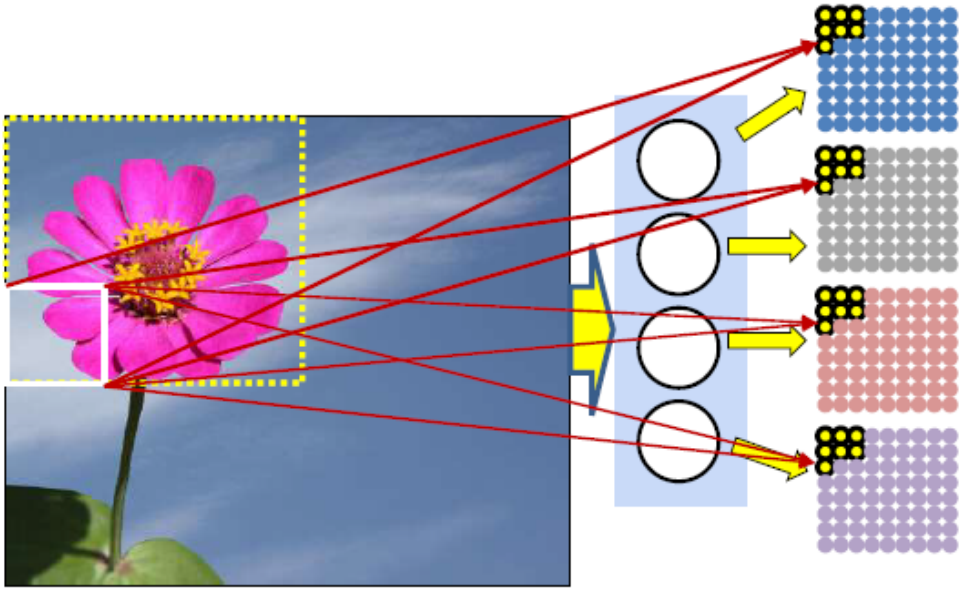
Distributing the scan

- Distribution forces localized patterns in lower layers
  - **More generalizable**
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates windows of outputs from the first layer
- The higher layer implicitly learns the arrangement of sub-patterns that represents the larger pattern (the flower in this case)



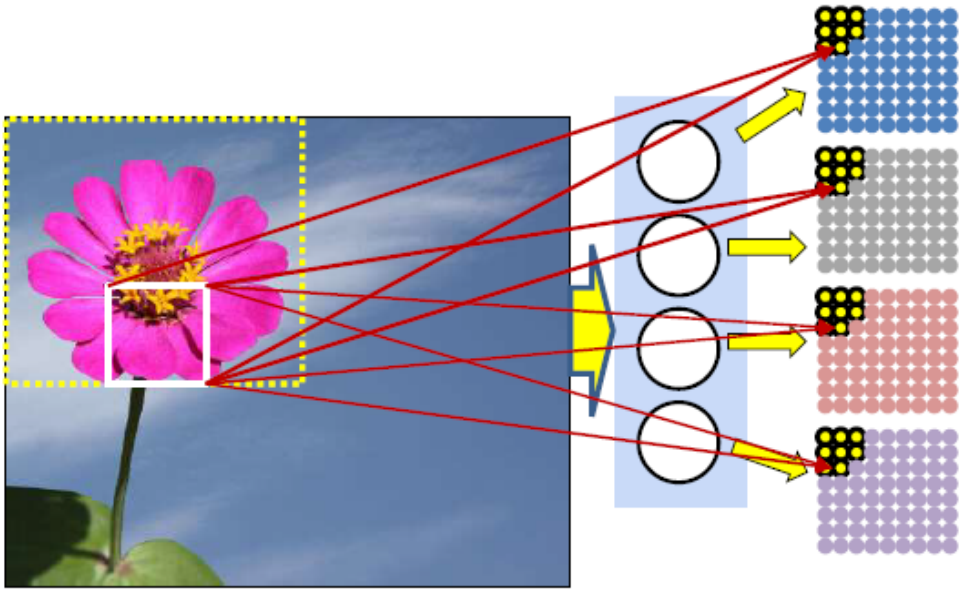
Distributing the scan

- Distribution forces localized patterns in lower layers
  - **More generalizable**
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates windows of outputs from the first layer
- The higher layer implicitly learns the arrangement of sub-patterns that represents the larger pattern (the flower in this case)



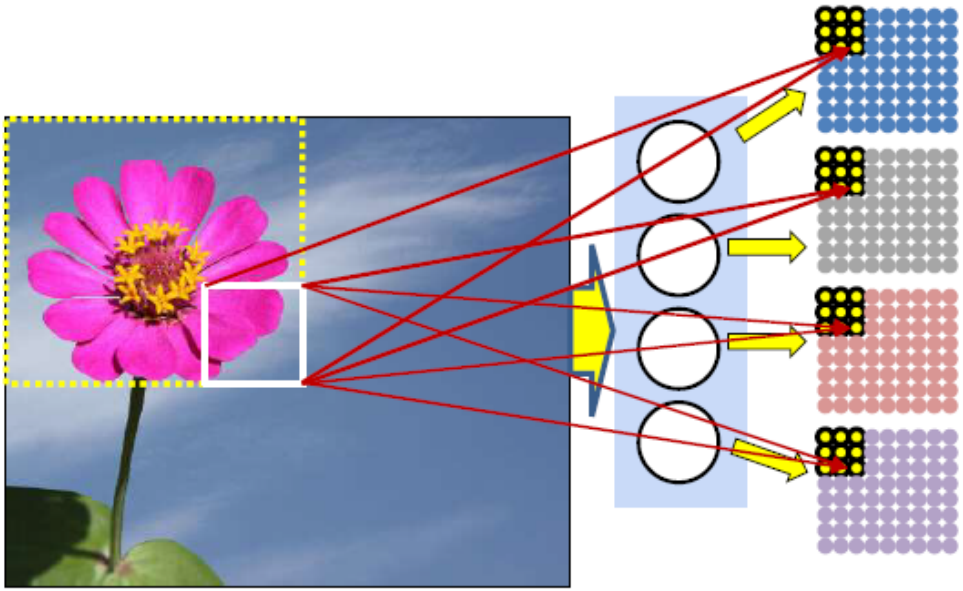
Distributing the scan

- Distribution forces localized patterns in lower layers
  - **More generalizable**
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates windows of outputs from the first layer
- The higher layer implicitly learns the arrangement of sub-patterns that represents the larger pattern (the flower in this case)



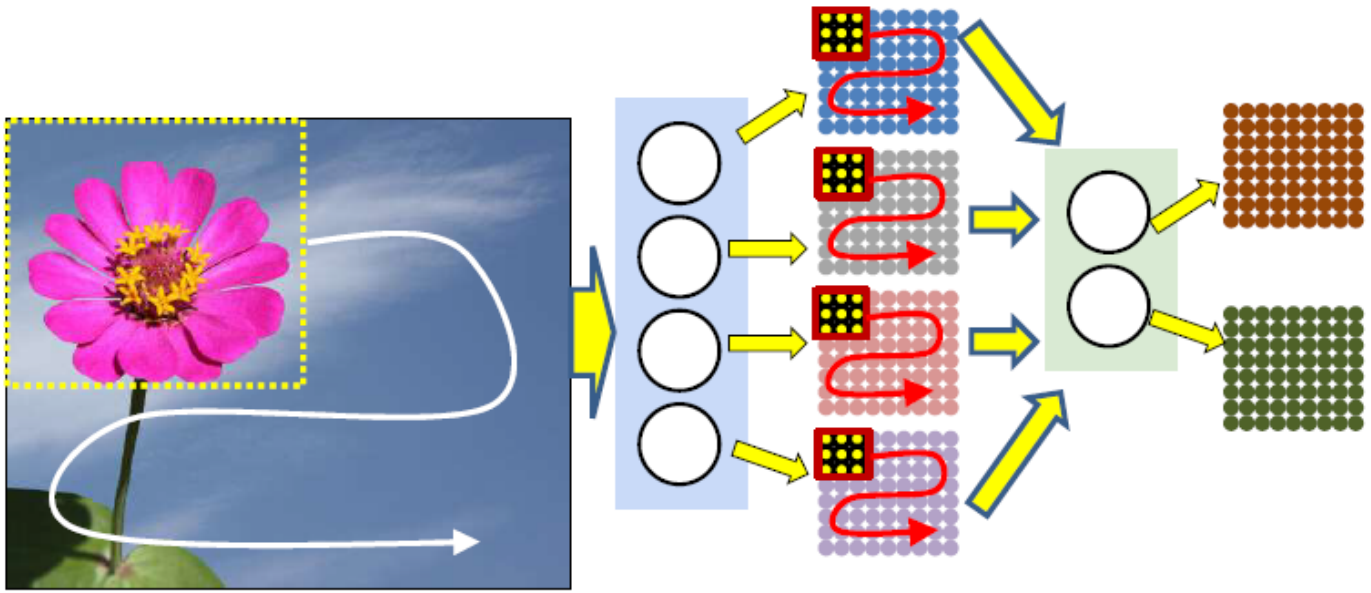
Distributing the scan

- Distribution forces localized patterns in lower layers
  - **More generalizable**
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates windows of outputs from the first layer
- The higher layer implicitly learns the arrangement of sub-patterns that represents the larger pattern (the flower in this case)



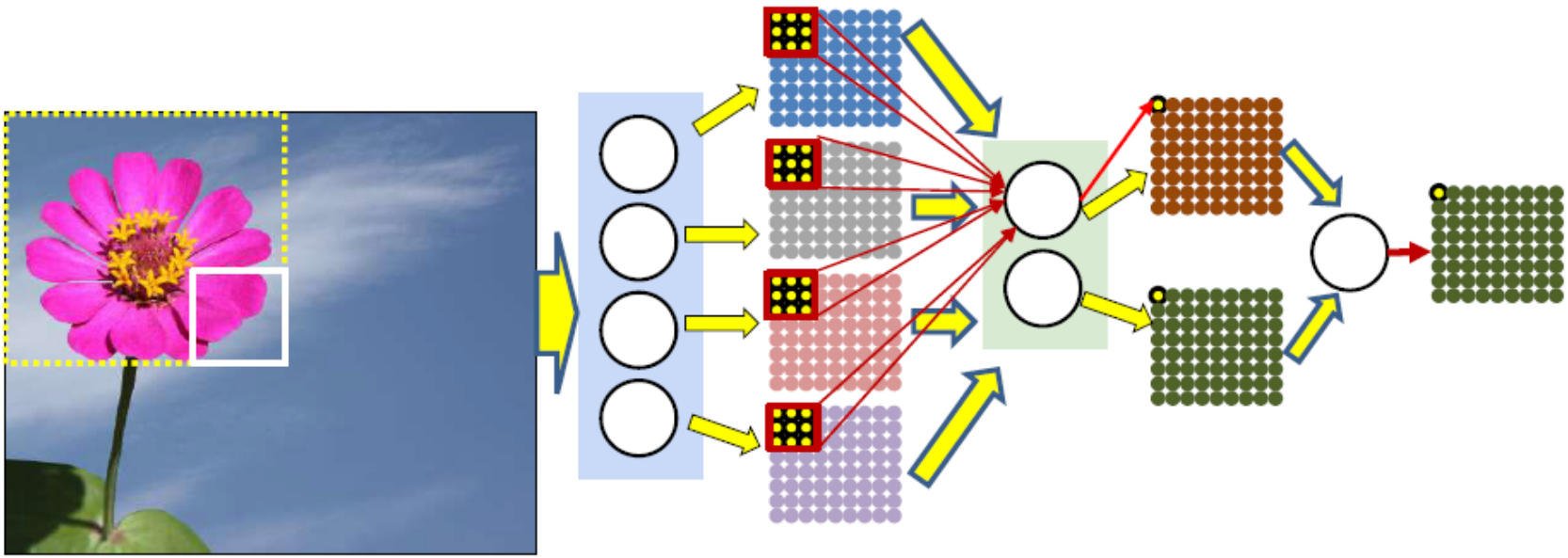
Distributing the scan

- Distribution forces localized patterns in lower layers
  - **More generalizable**
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates windows of outputs from the first layer
- The higher layer implicitly learns the arrangement of sub-patterns that represents the larger pattern (the flower in this case)



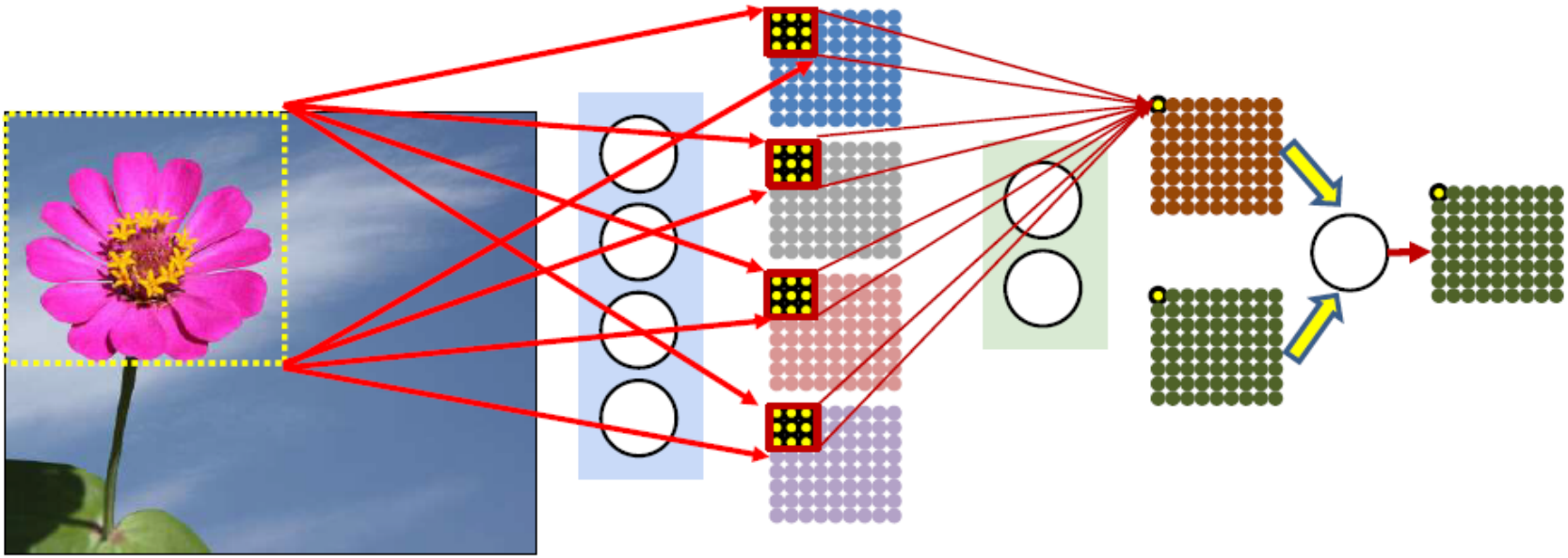
Distributing the scan

- Distribution forces localized patterns in lower layers
  - **More generalizable**
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates windows of outputs from the first layer
- The higher layer implicitly learns the arrangement of sub-patterns that represents the larger pattern (the flower in this case)



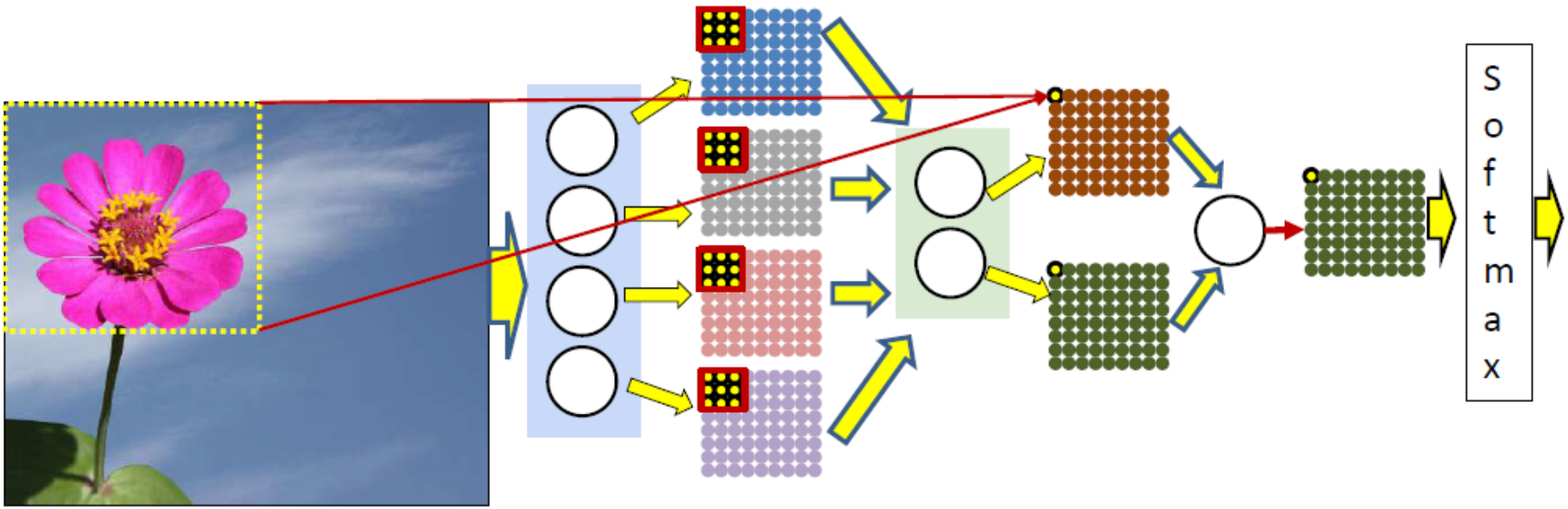
Distributing the scan

- Distribution forces localized patterns in lower layers
  - **More generalizable**
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates windows of outputs from the first layer
- The higher layer implicitly learns the arrangement of sub-patterns that represents the larger pattern (the flower in this case)



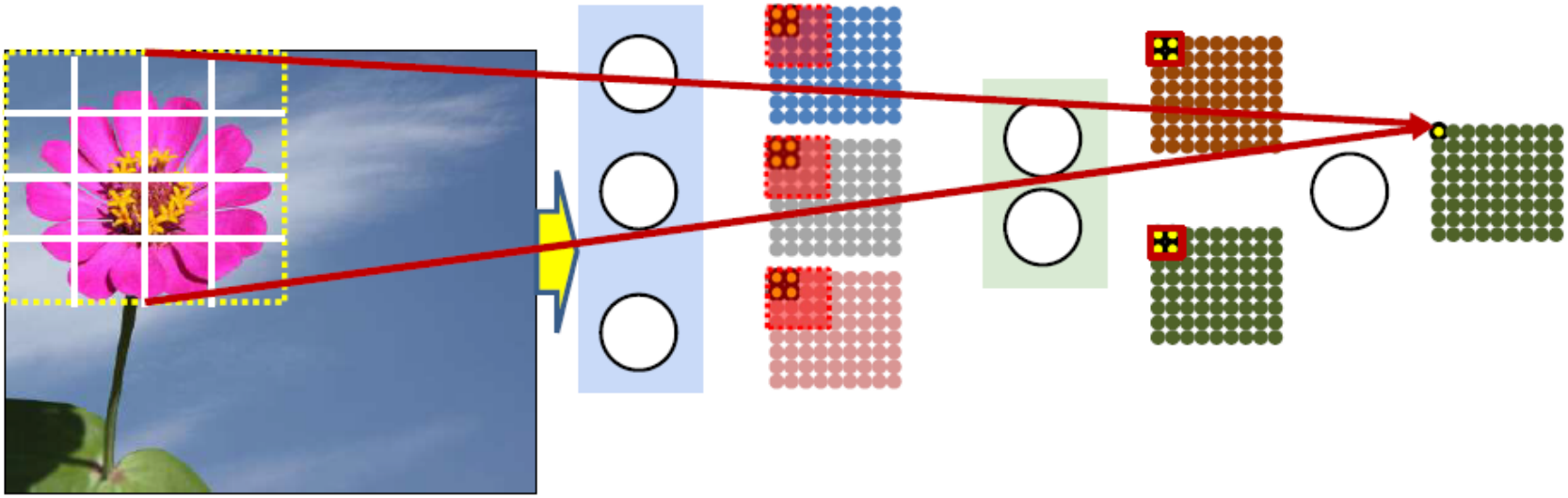
Distributing the scan

- Distribution forces localized patterns in lower layers
  - **More generalizable**
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates windows of outputs from the first layer
- The higher layer implicitly learns the arrangement of sub-patterns that represents the larger pattern (the flower in this case)



This logic can be recursed

- Building the pattern over 3 layers



What is a convolution

- Scanning an image with a “filter”
  - Note: a filter is really just a perceptron, with weights and a bias

Example 5x5 image with binary pixels

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Example 3x3 filter

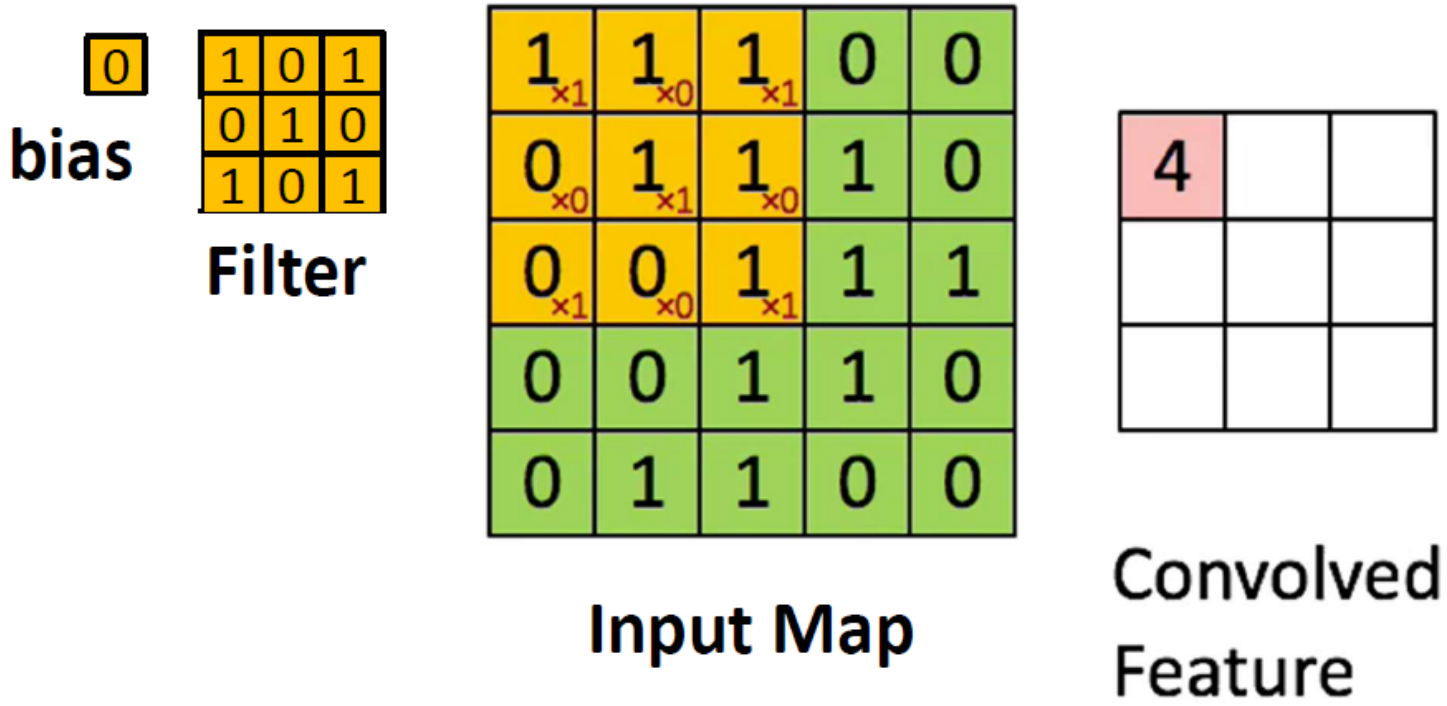
1	0	1
0	1	0
1	0	1

bias

0

What is a convolution

- Scanning an image with a “filter”
  - At each location, the filter and the underlying map values are multiplied component wise, and the products are added along with the bias



What is a convolution

- Scanning an image with a “filter”
  - The filter may proceed by more than 1 pixel at a time
  - E.g. with a “stride” of two pixels per shift

Example 5x5 image with binary pixels

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Example 3x3 filter

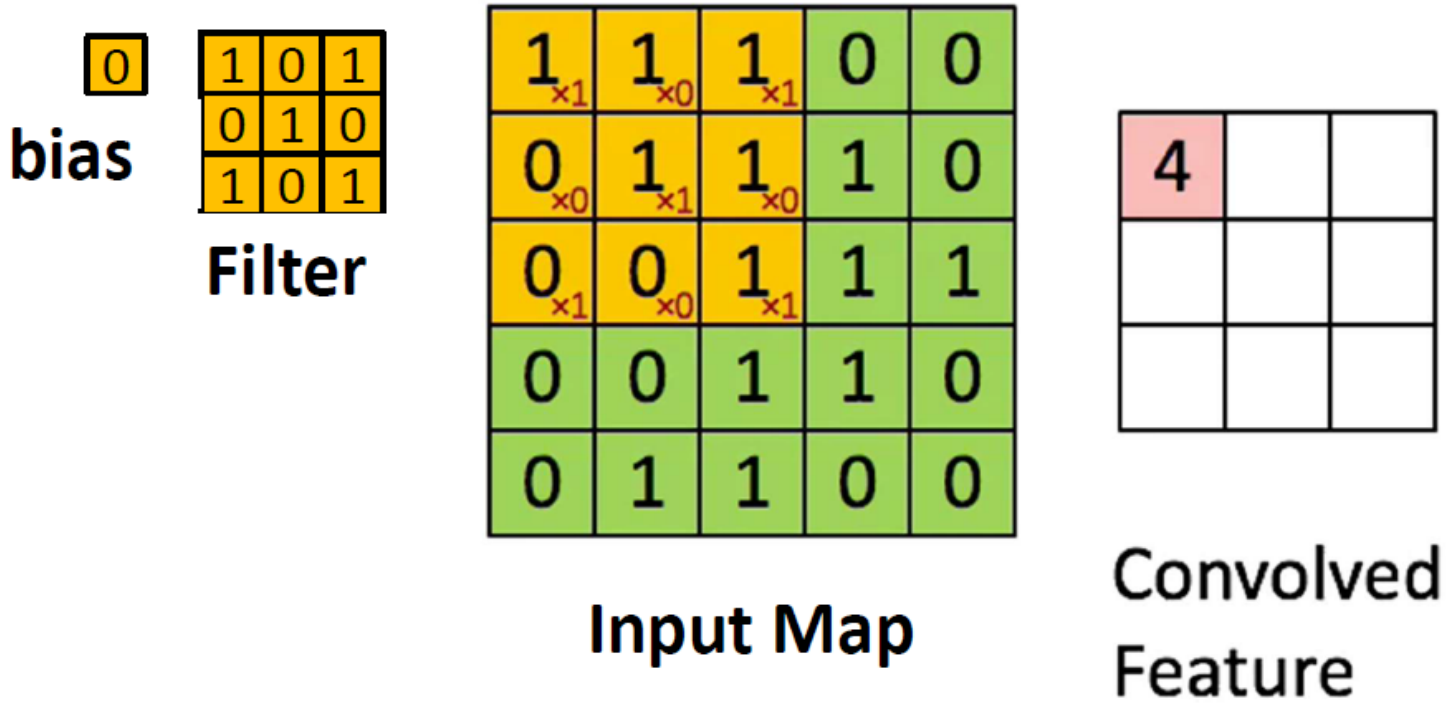
1	0	1
0	1	0
1	0	1

bias

0

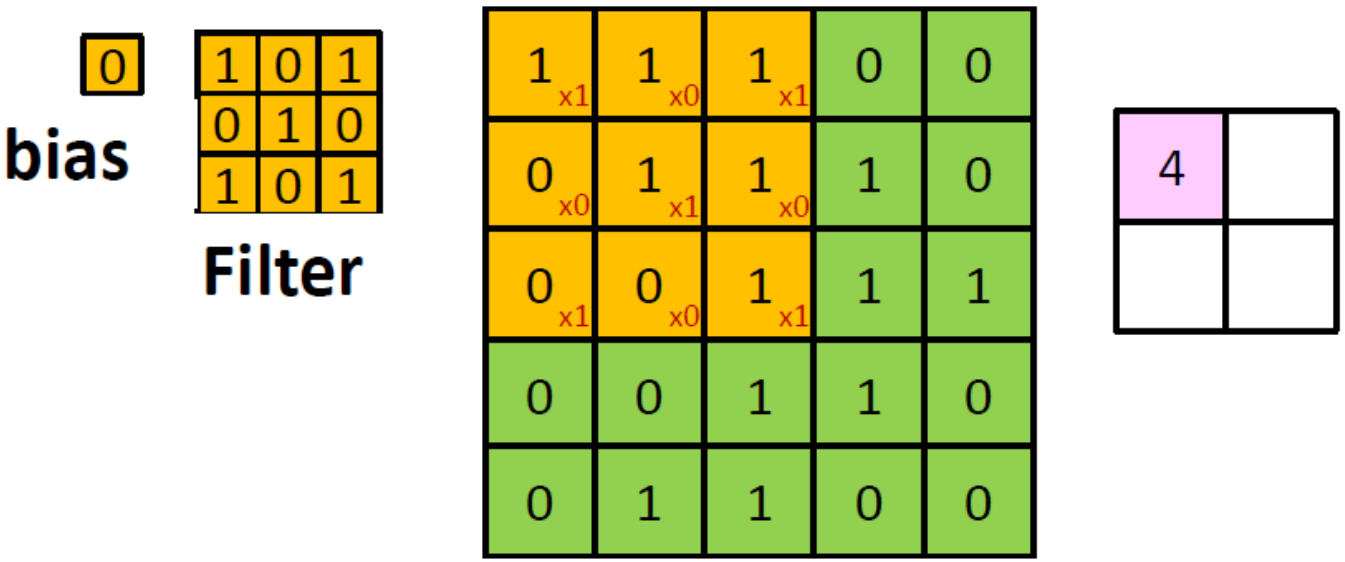
What is a convolution

- Scanning an image with a “filter”
  - The filter may proceed by more than 1 pixel at a time
  - E.g. with a “stride” of two pixels per shift



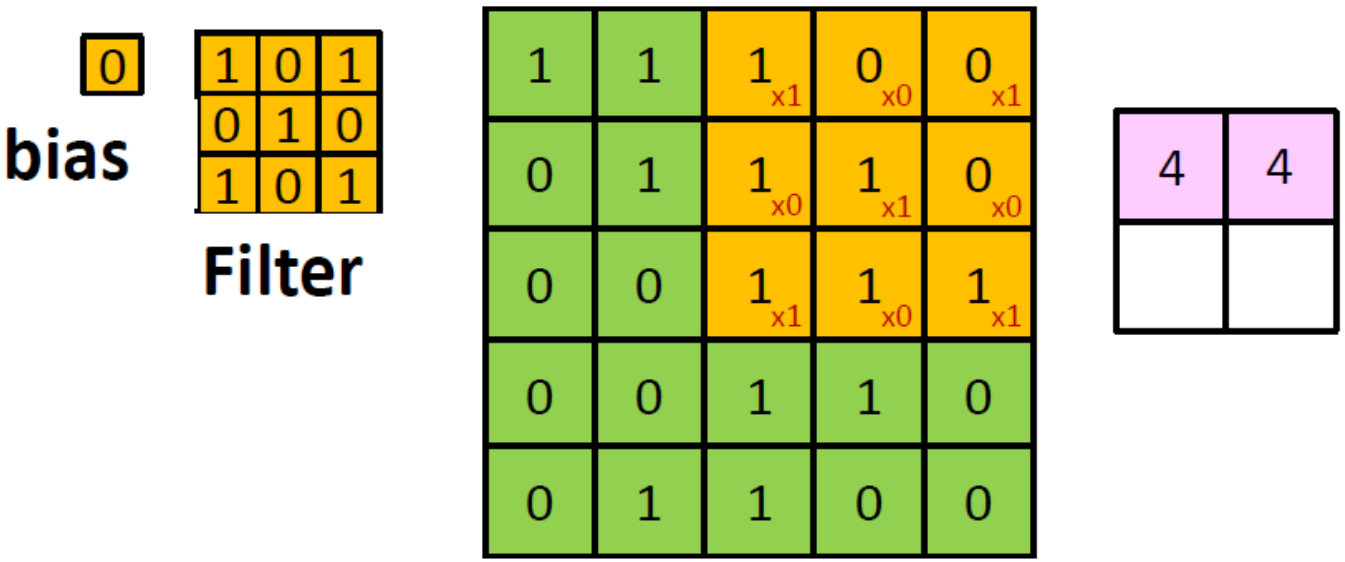
What is a convolution

- Scanning an image with a “filter”
  - The filter may proceed by more than 1 pixel at a time
  - E.g. with a “stride” of two pixels per shift



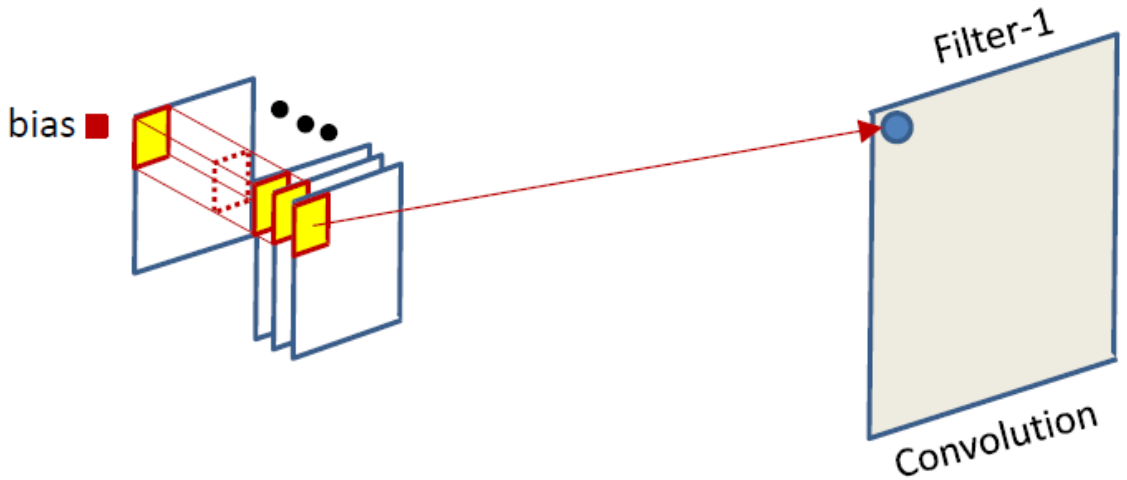
What is a convolution

- Scanning an image with a “filter”
  - The filter may proceed by more than 1 pixel at a time
  - E.g. with a “stride” of two pixels per shift



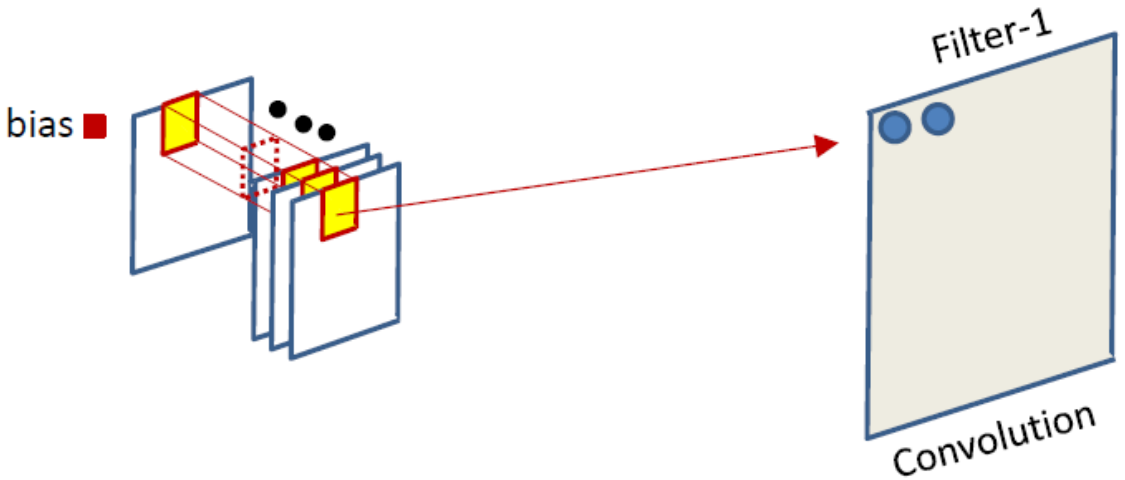
The “cube” view of input

- The computation of the convolutional map at any location sums the convolutional outputs at all planes



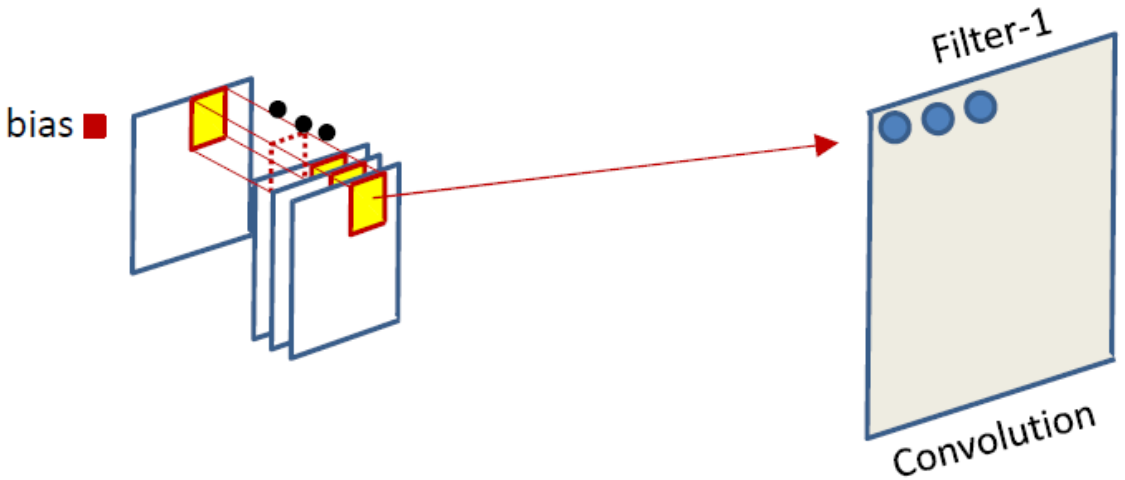
The “cube” view of input

- The computation of the convolutional map at any location sums the convolutional outputs at all planes



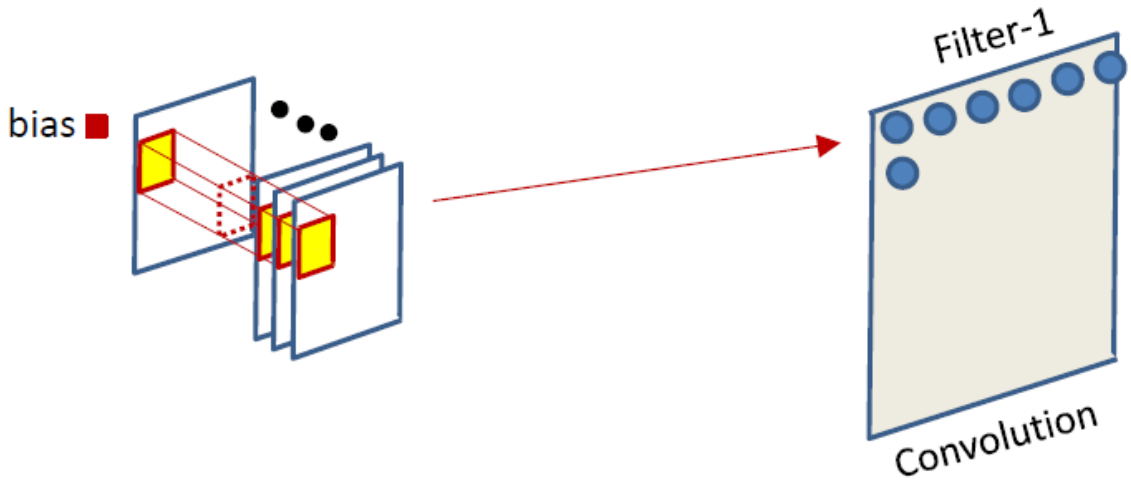
The “cube” view of input

- The computation of the convolutional map at any location sums the convolutional outputs at all planes



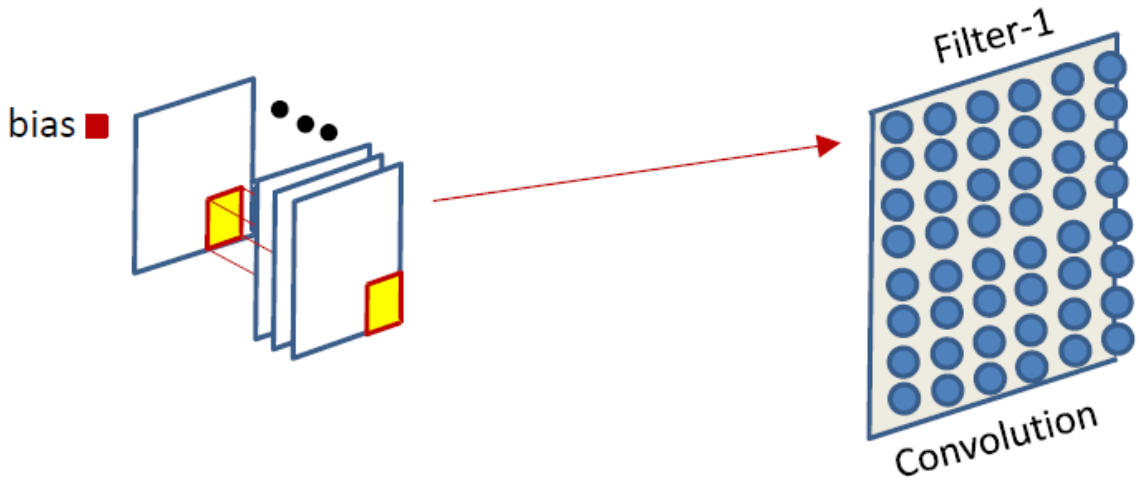
The “cube” view of input

- The computation of the convolutional map at any location sums the convolutional outputs at all planes



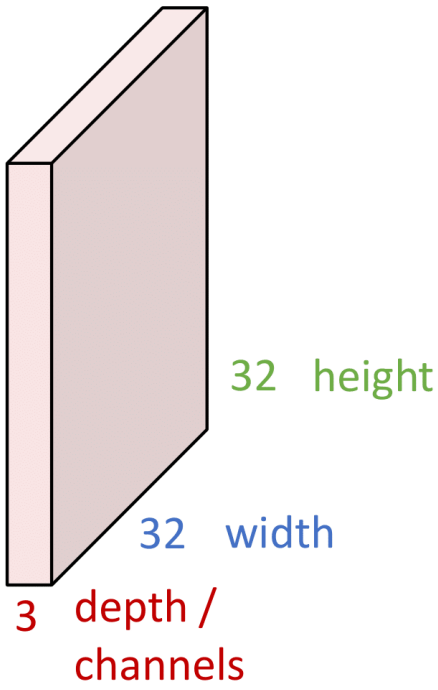
The “cube” view of input

- The computation of the convolutional map at any location sums the convolutional outputs at all planes

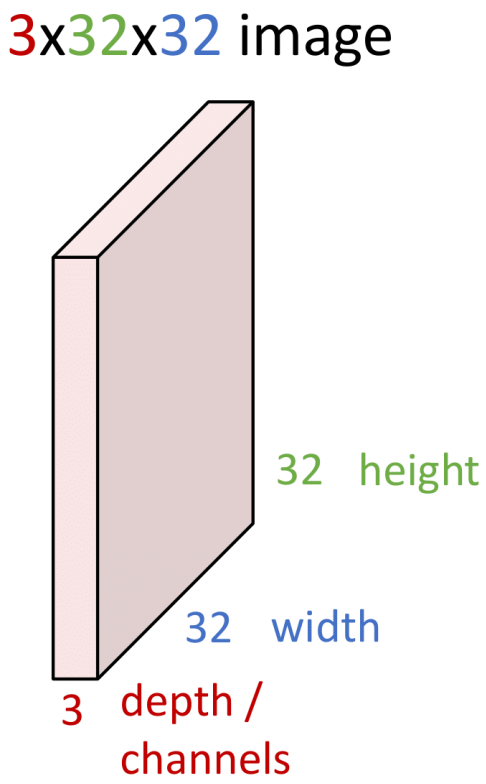


Convolution Layer

3x32x32 image: preserve spatial structure

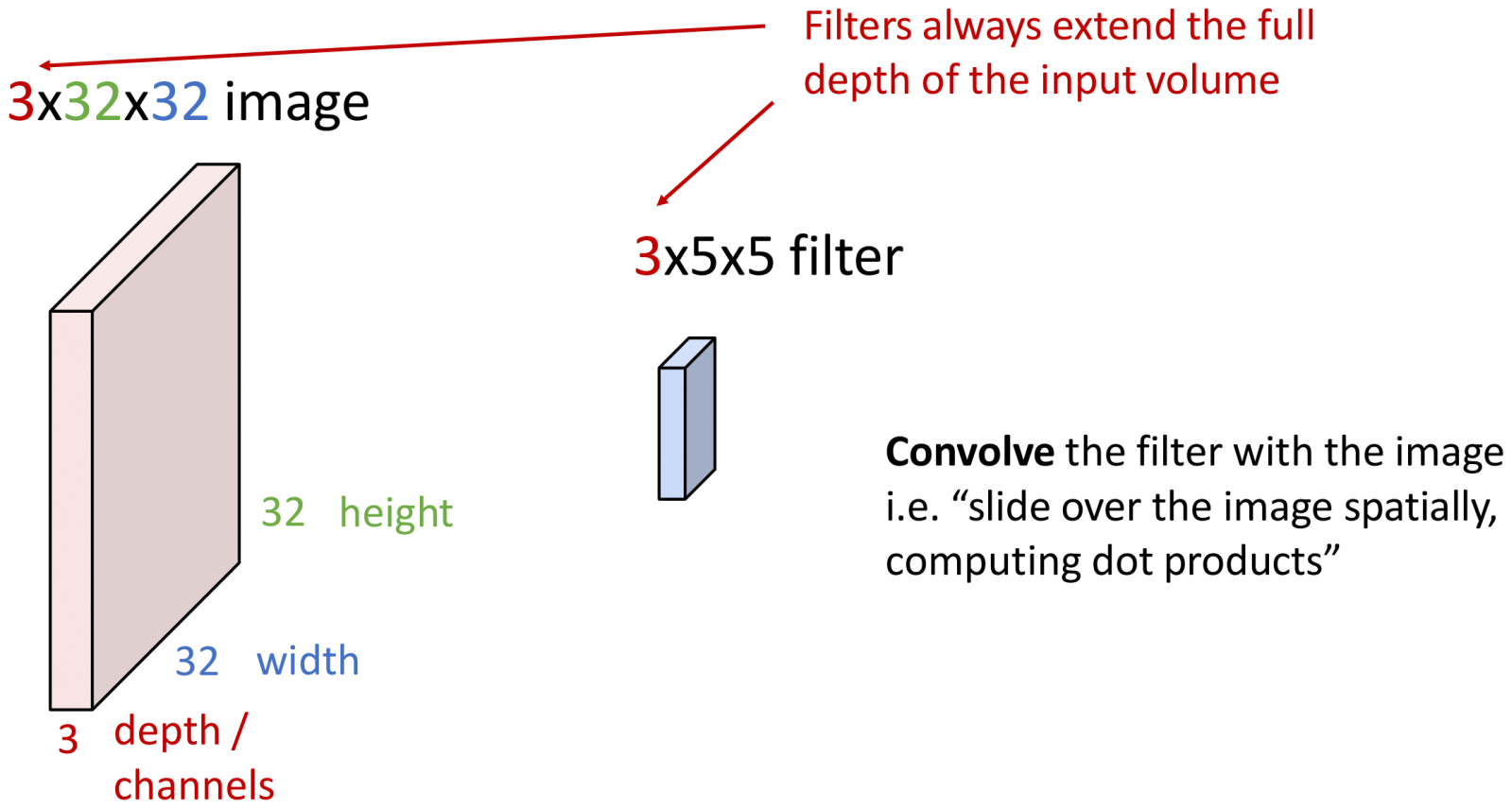


Convolution Layer

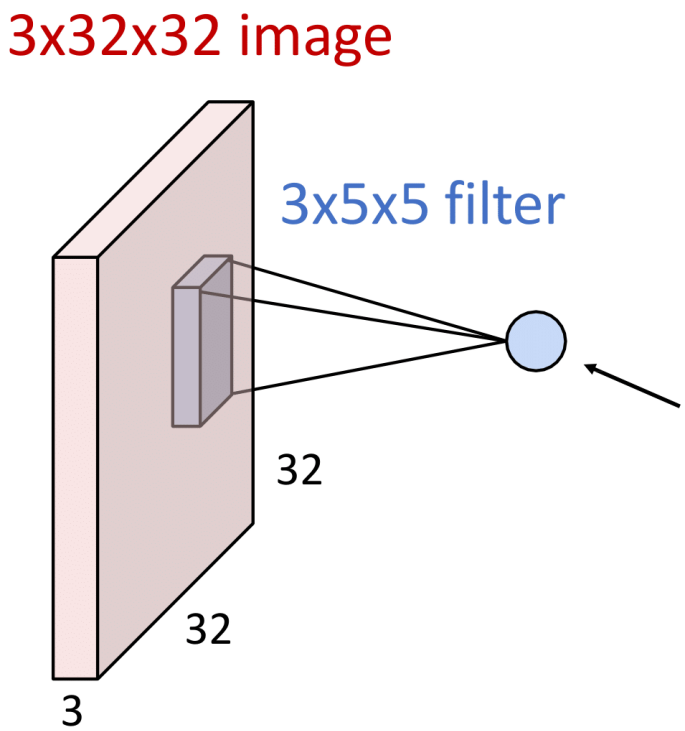


Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Convolution Layer



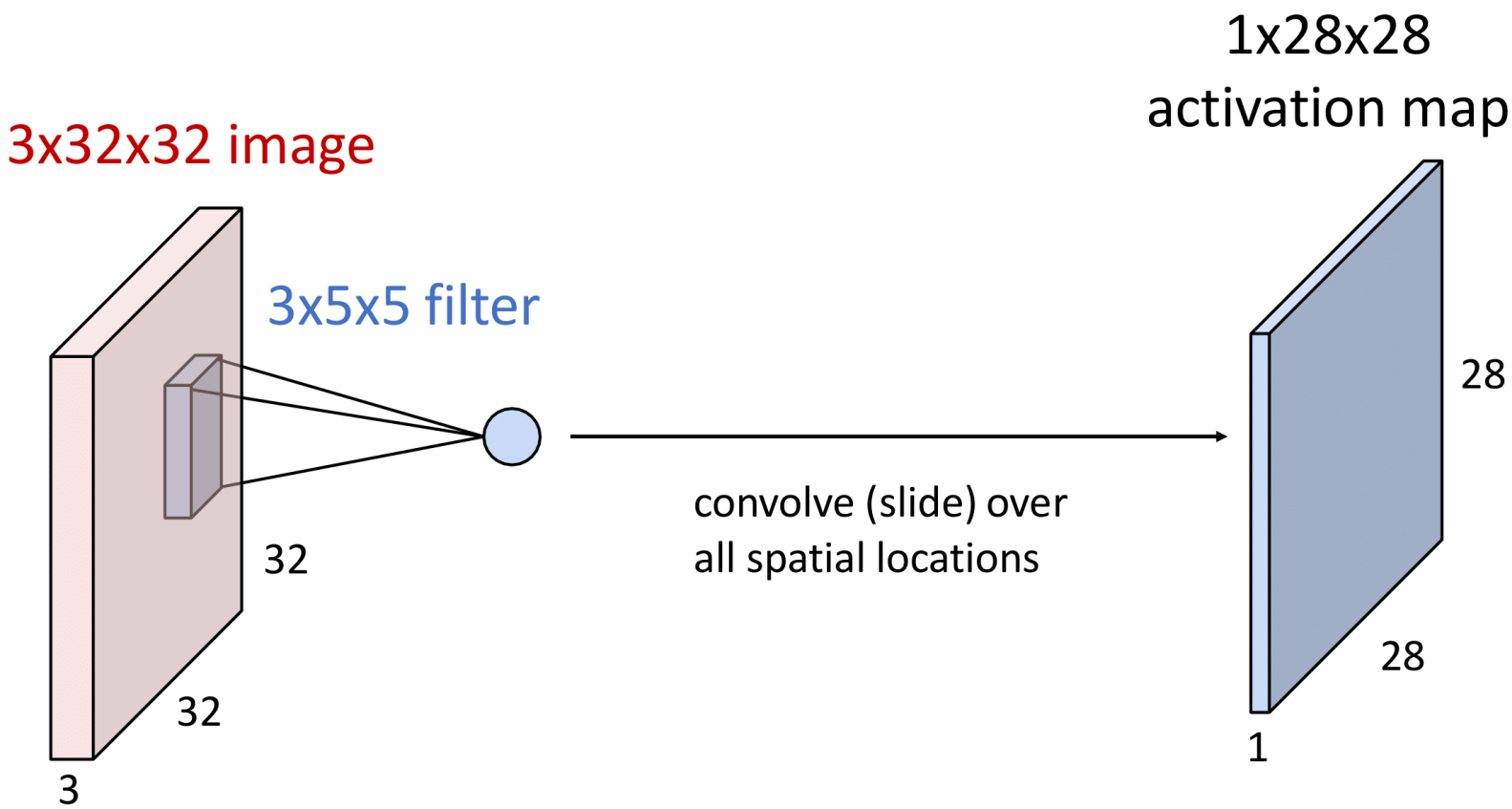
Convolution Layer



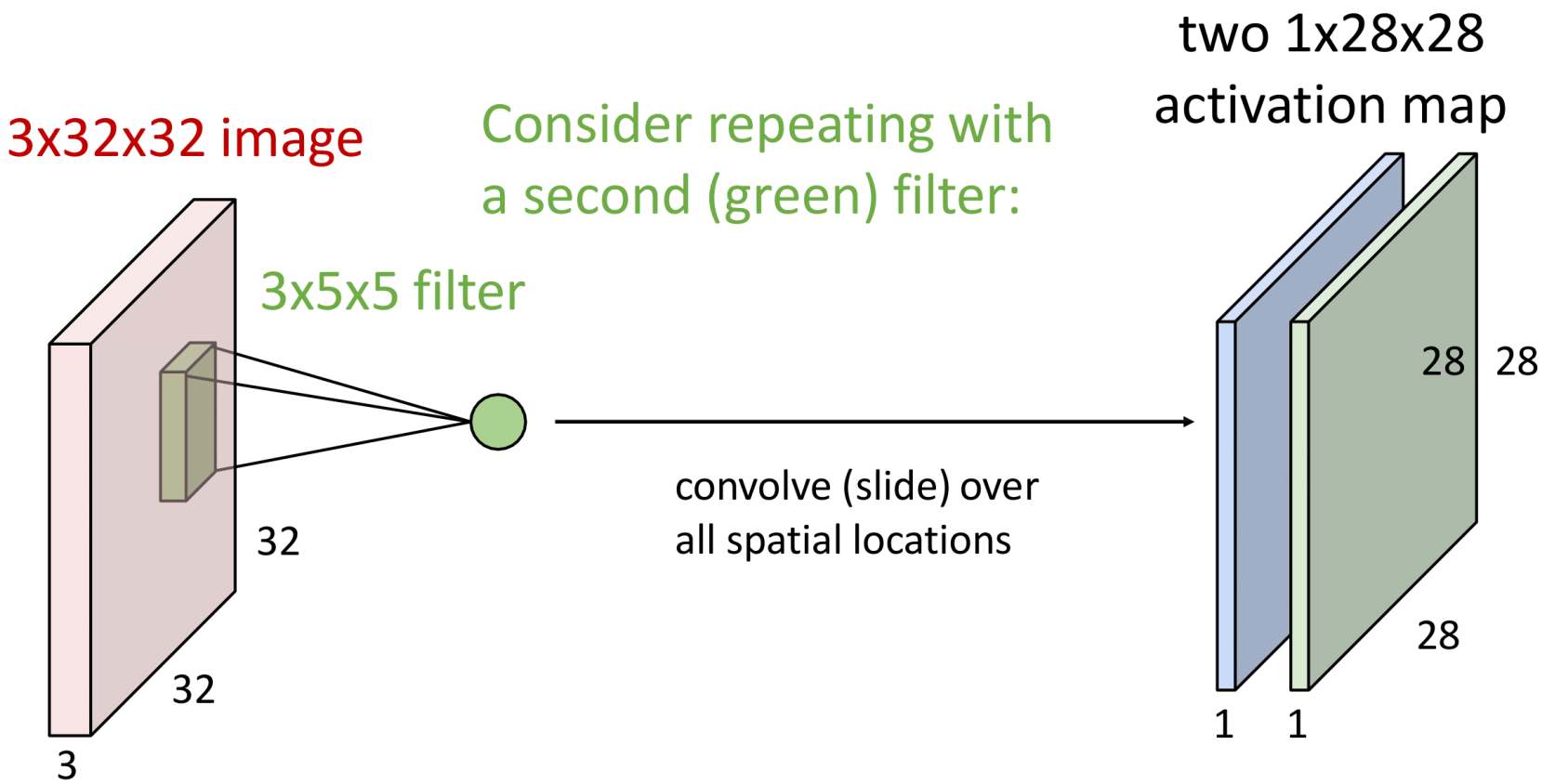
1 number:
the result of taking a dot product between the filter and a small 3x5x5 chunk of the image
(i.e. 3*5*5 = 75-dimensional dot product + bias)

$$w^T x + b$$

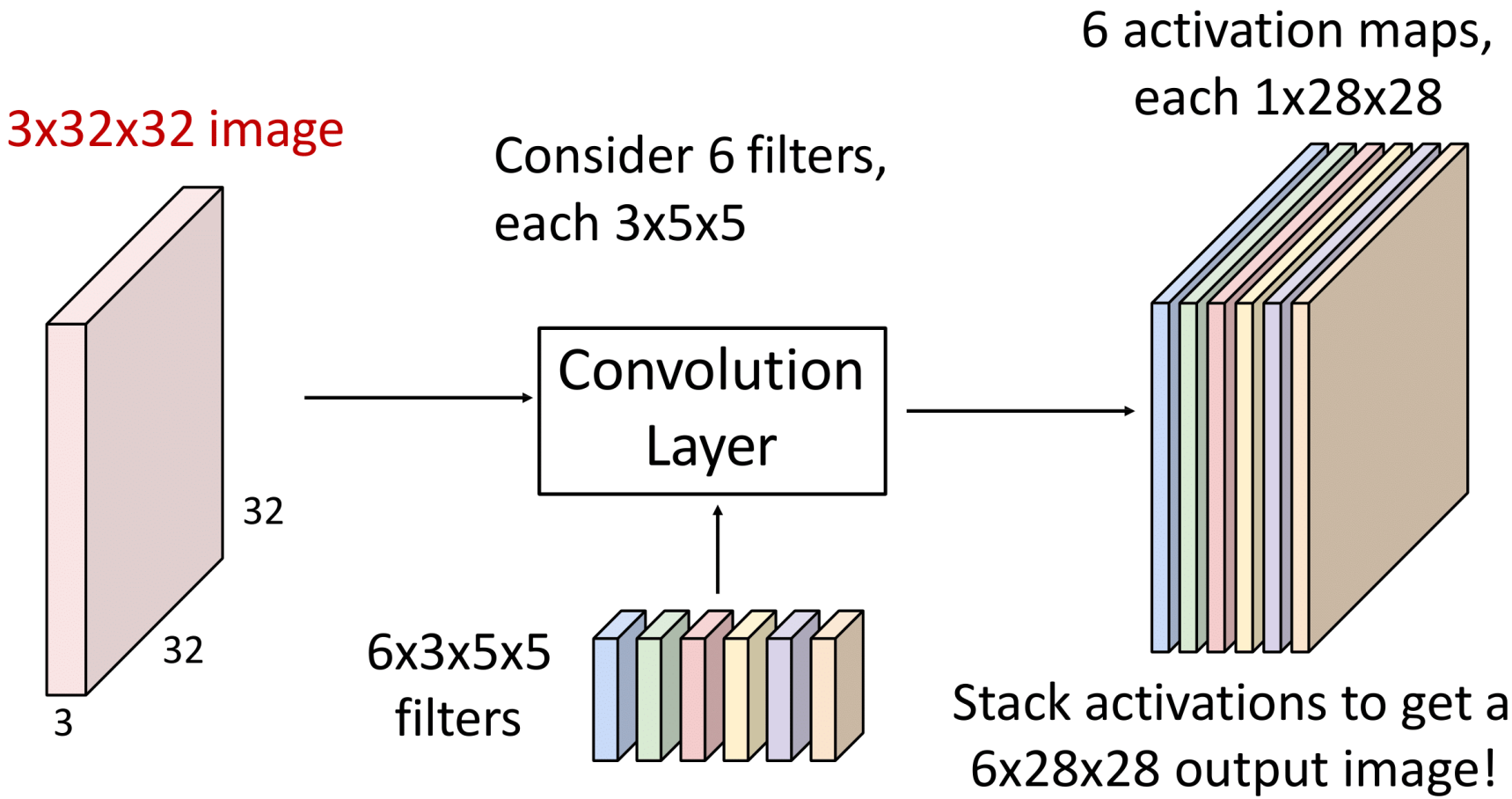
Convolution Layer



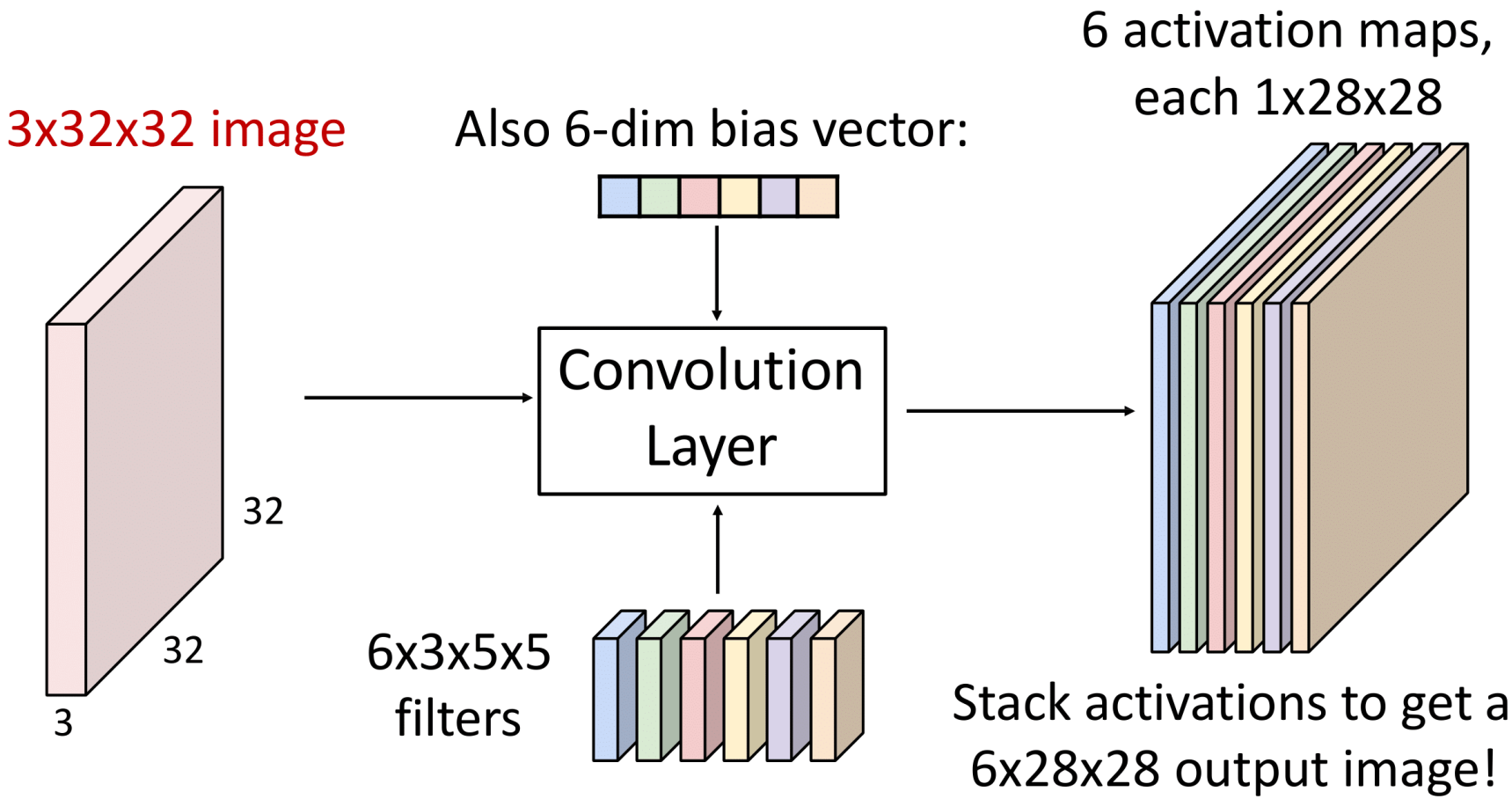
Convolution Layer



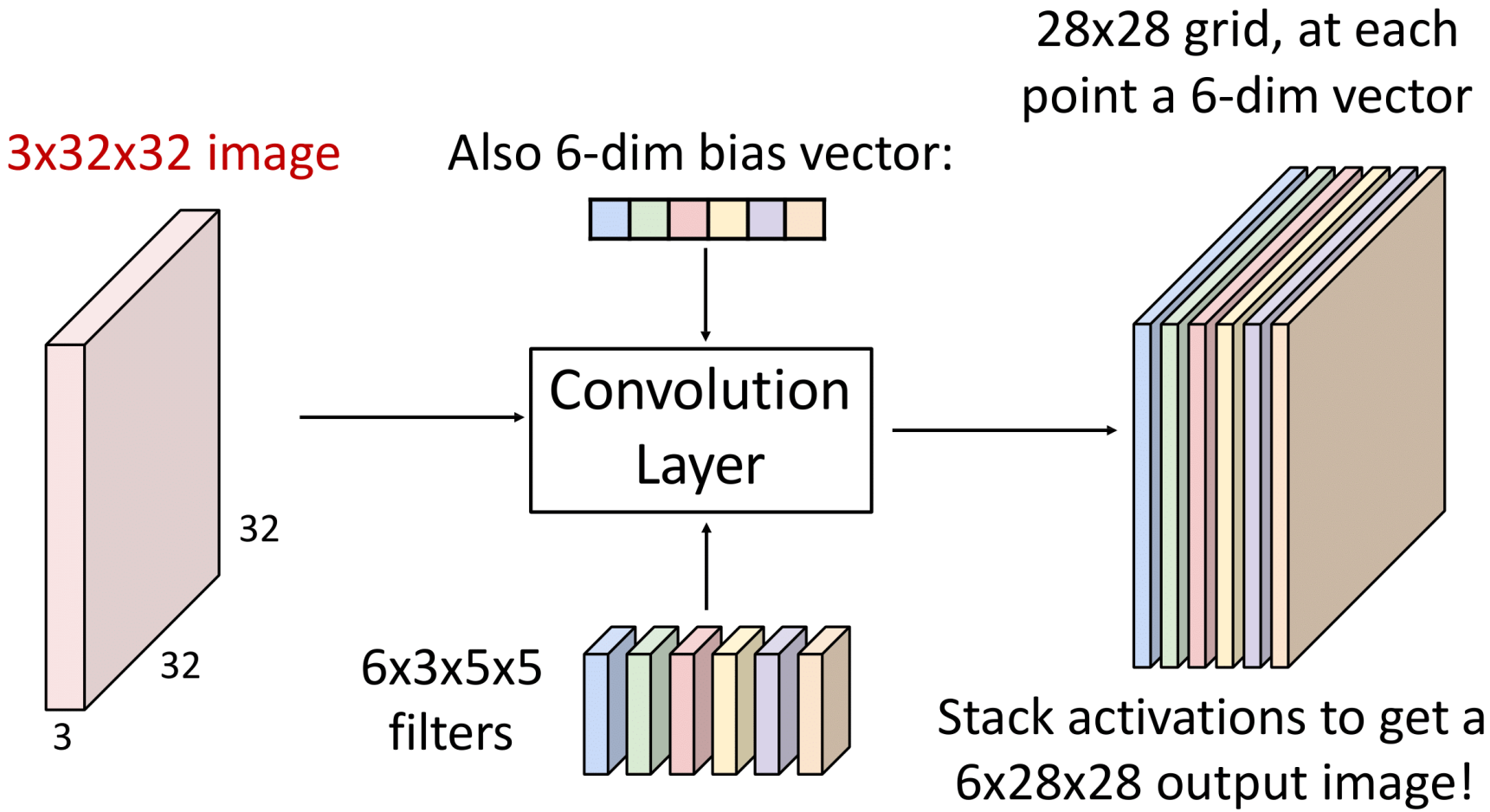
Convolution Layer



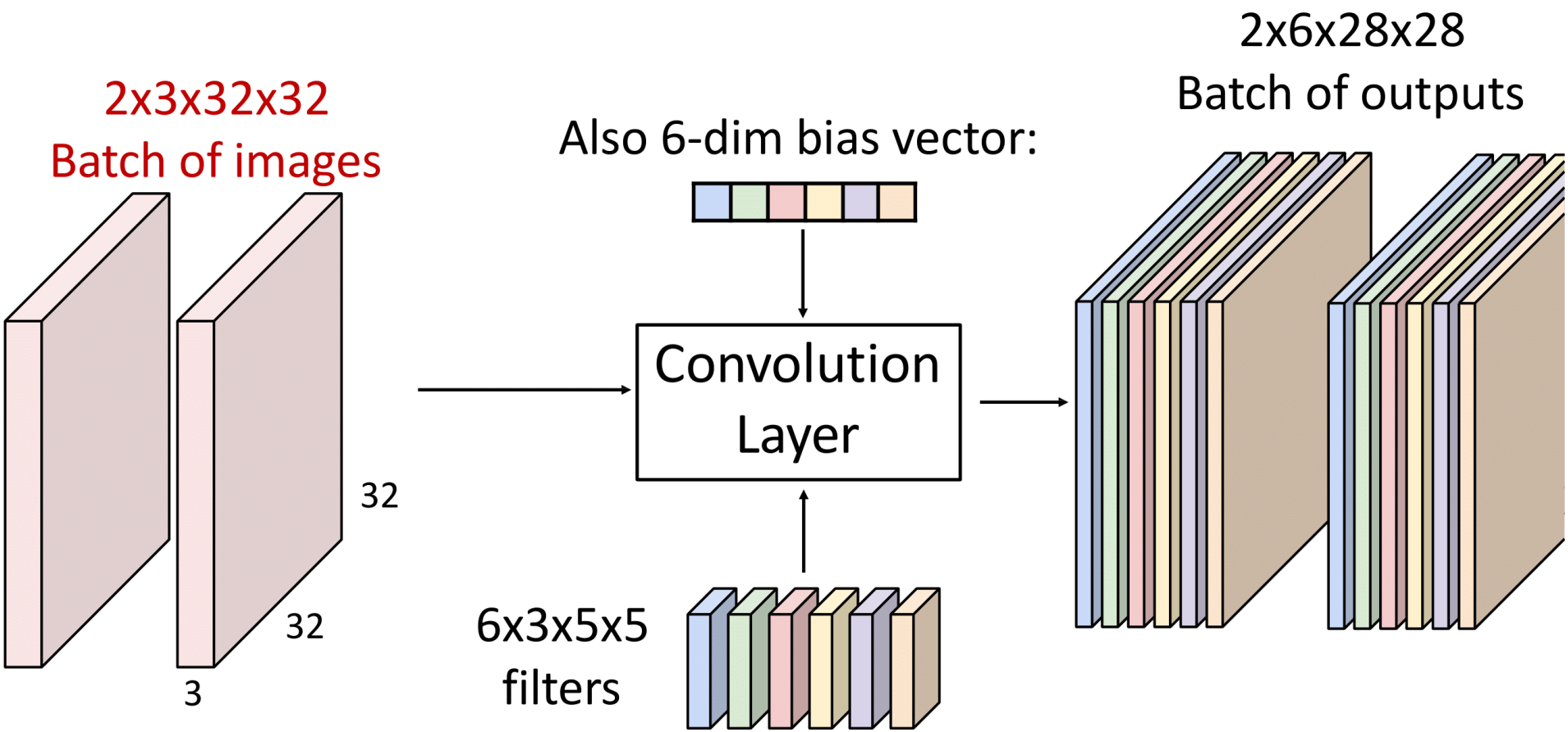
Convolution Layer



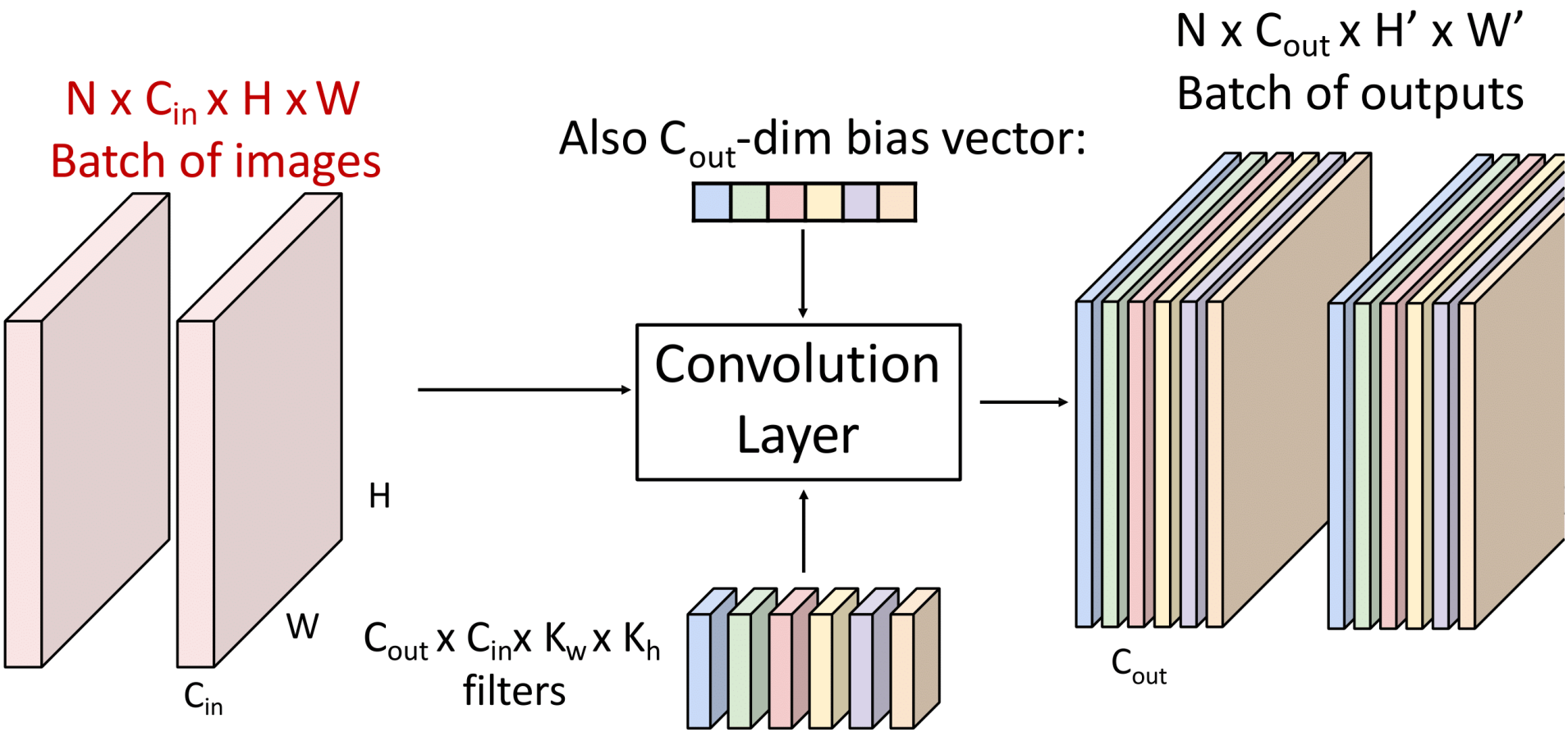
Convolution Layer



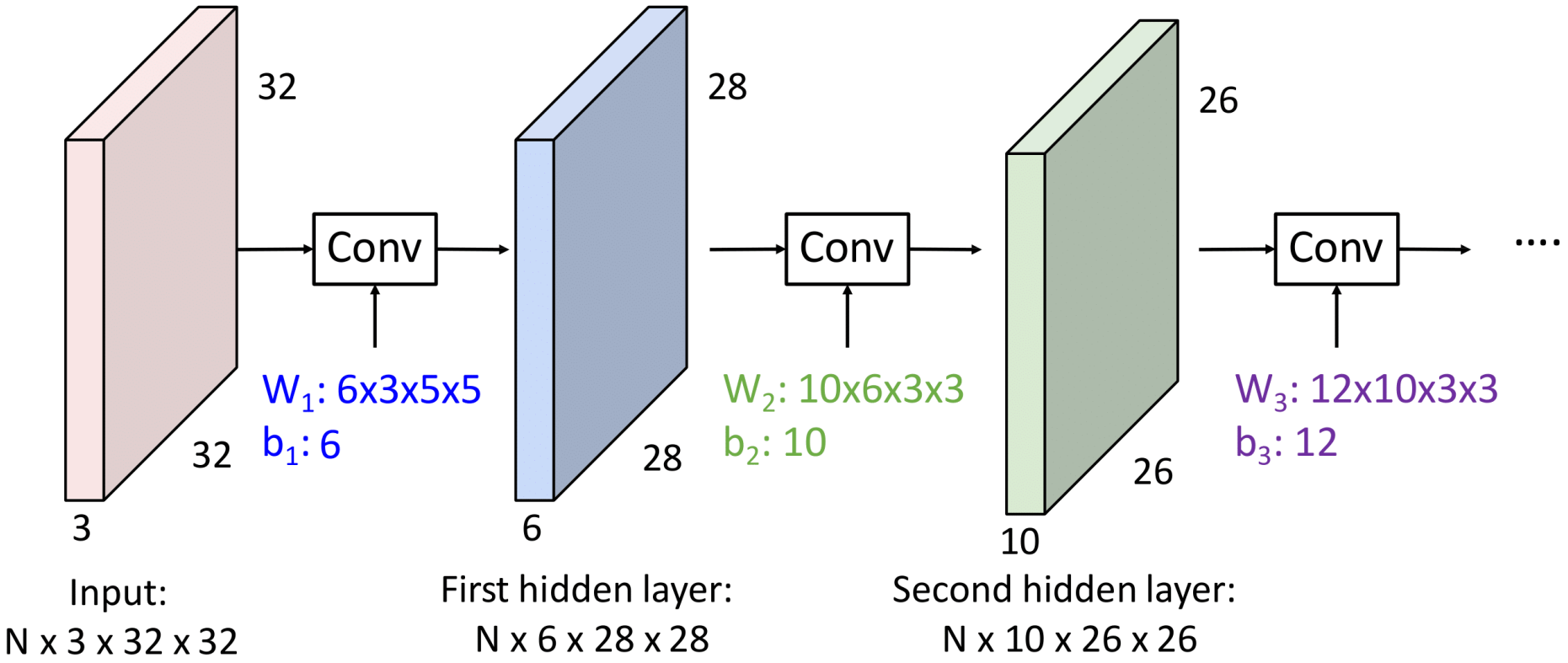
Convolution Layer



Convolution Layer

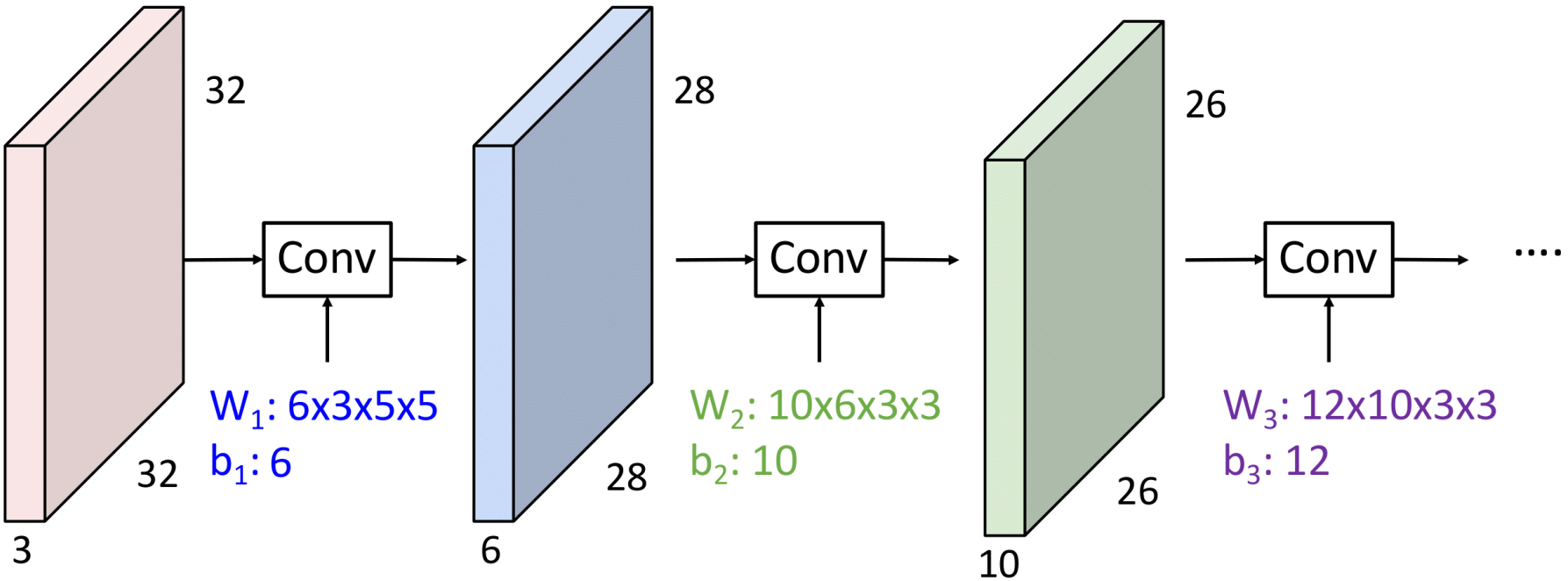


Stacking convolutions



Stacking convolutions

Q: What happens if we stack two convolution layers?



Input:
N x 3 x 32 x 32

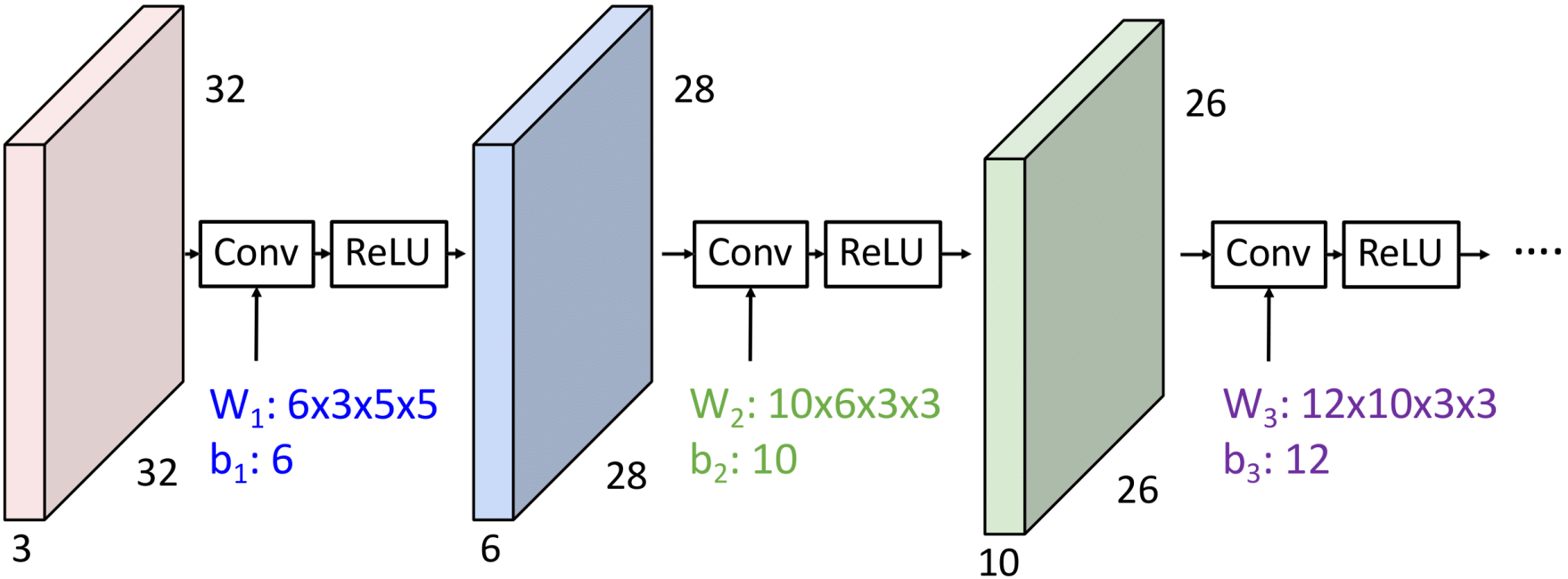
First hidden layer:
N x 6 x 28 x 28

Second hidden layer:
N x 10 x 26 x 26

Stacking convolutions

Q: What happens if we stack two convolution layers? (Recall $y=W_2W_1x$ is a linear classifier)

A: We get another convolution!

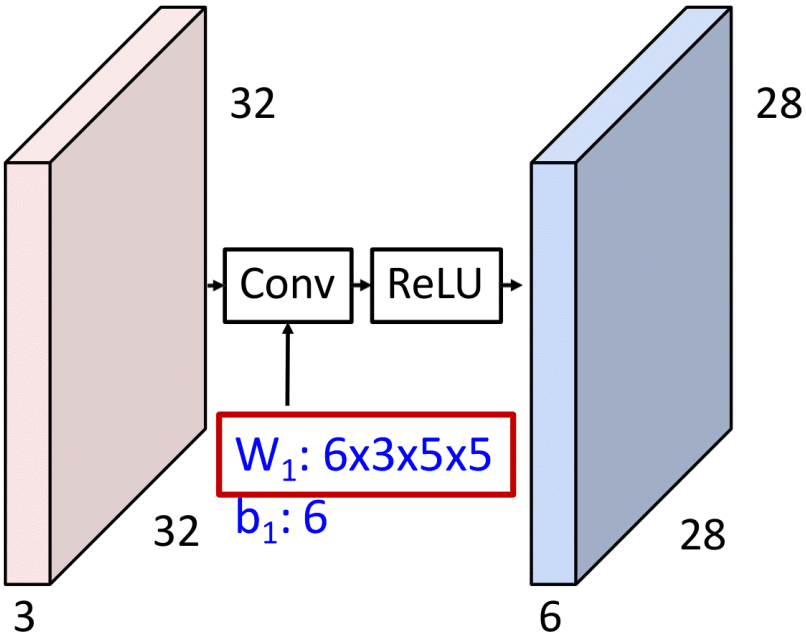


Input:
N x 3 x 32 x 32

First hidden layer:
N x 6 x 28 x 28

Second hidden layer:
N x 10 x 26 x 26

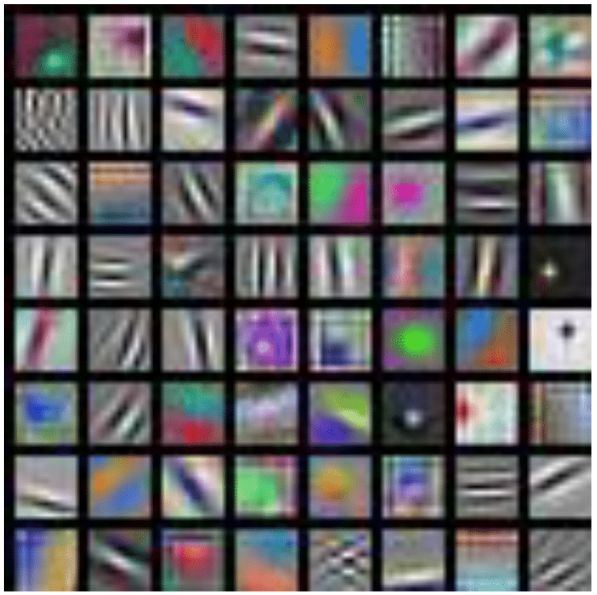
What do convolutional filters learn?



Input:
N x 3 x 32 x 32

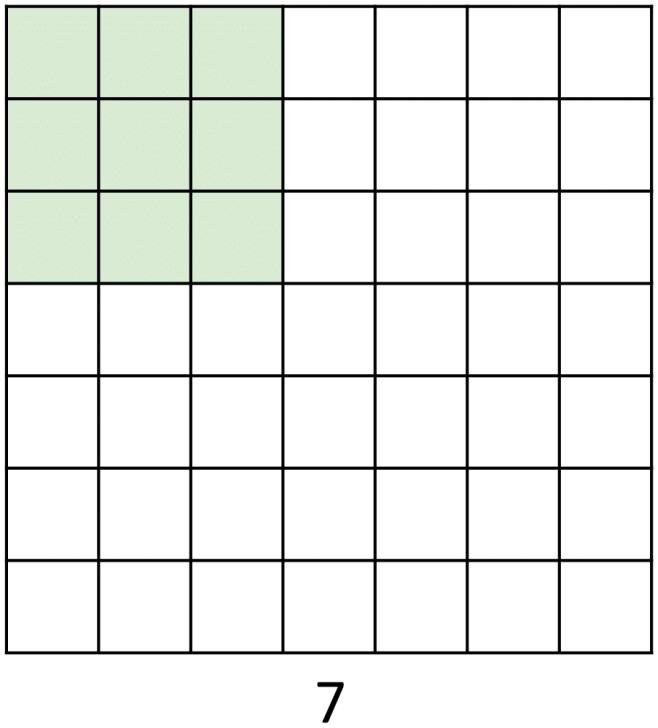
First hidden layer:
N x 6 x 28 x 28

First-layer conv filters: local image templates
(Often learns oriented edges, opposing colors)



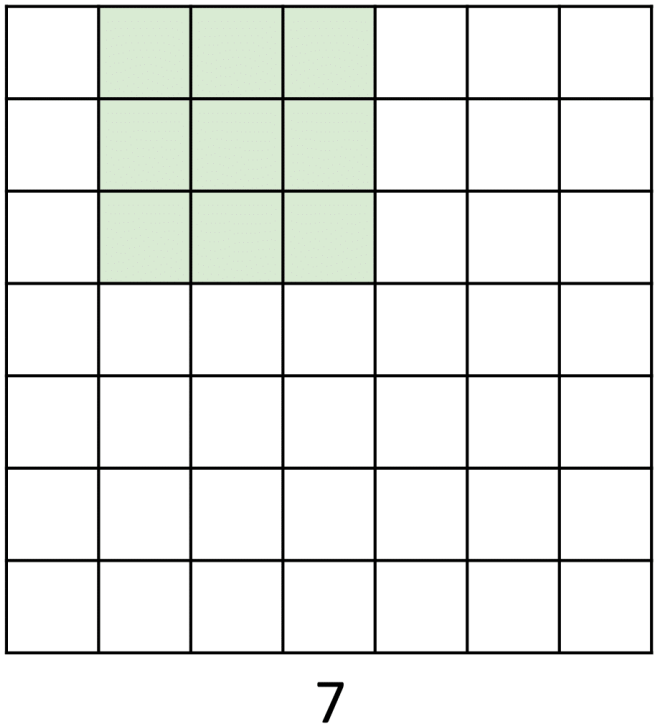
AlexNet: 64 filters, each 3x11x11

A closer look at spatial dimensions



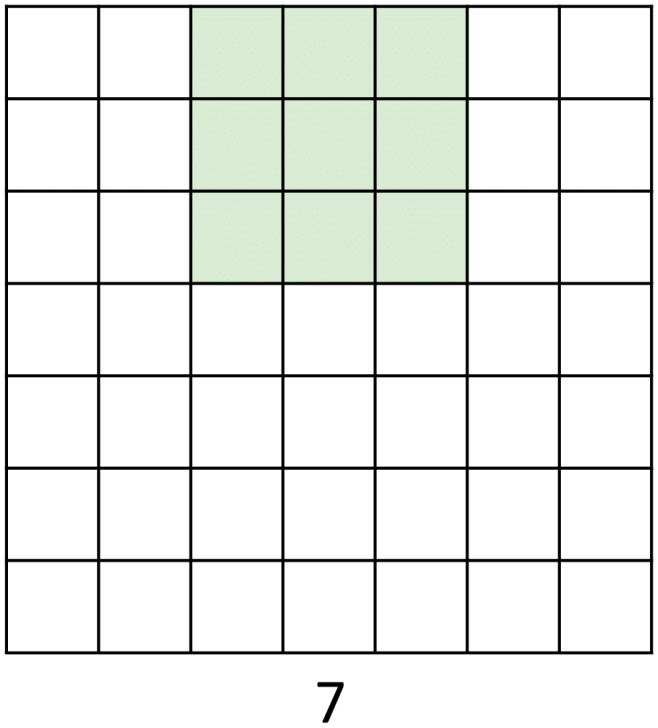
Input: 7x7
Filter: 3x3

A closer look at spatial dimensions



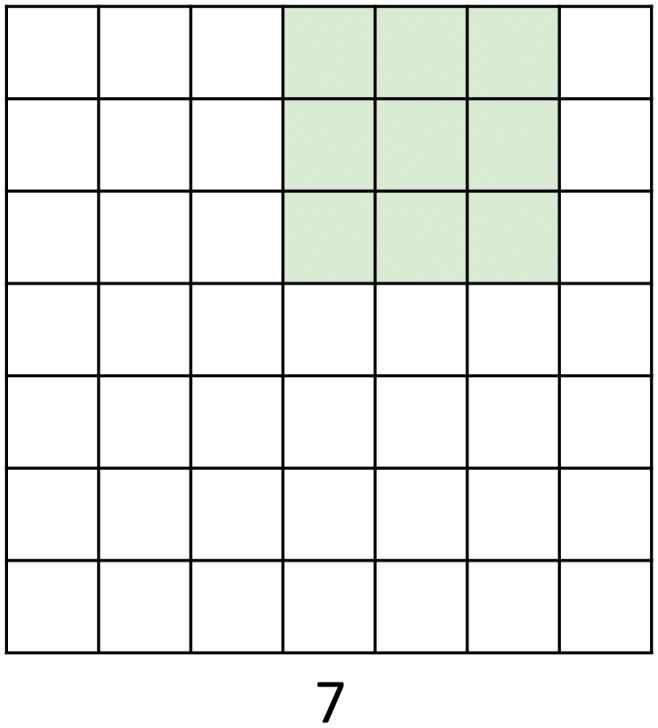
Input: 7x7
Filter: 3x3

A closer look at spatial dimensions



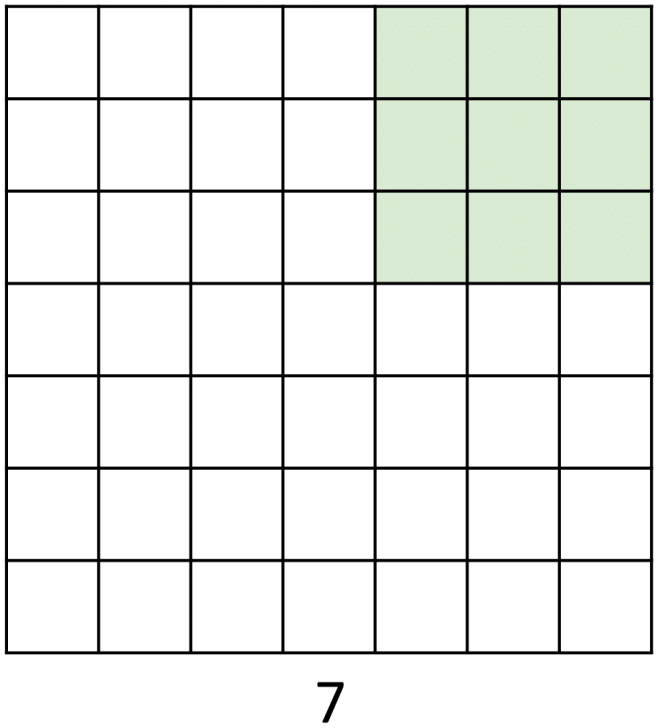
Input: 7x7
Filter: 3x3

A closer look at spatial dimensions



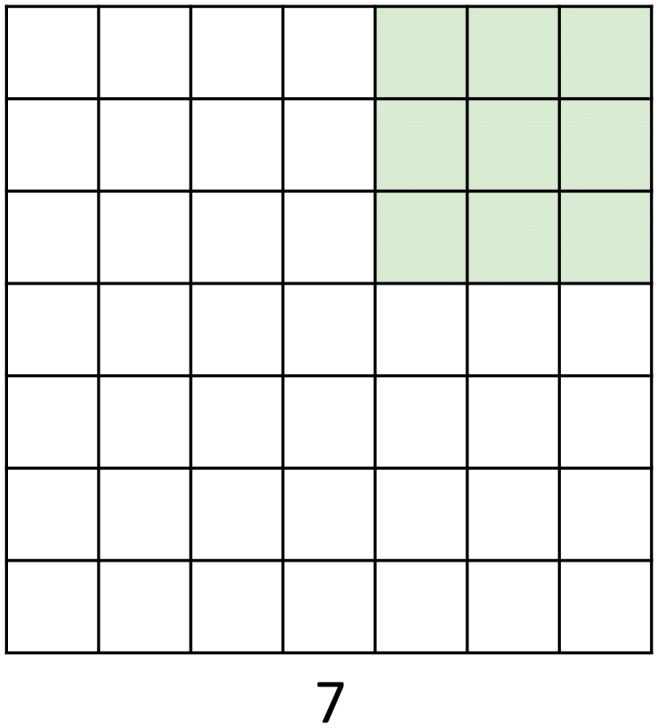
Input: 7x7
Filter: 3x3

A closer look at spatial dimensions



Input: 7x7
Filter: 3x3
Output: 5x5

A closer look at spatial dimensions



Input: 7x7
Filter: 3x3
Output: 5x5

In general:
Input: W
Filter: K
Output: W – K + 1

Problem: Feature maps "shrink" with each layer!

A closer look at spatial dimensions

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

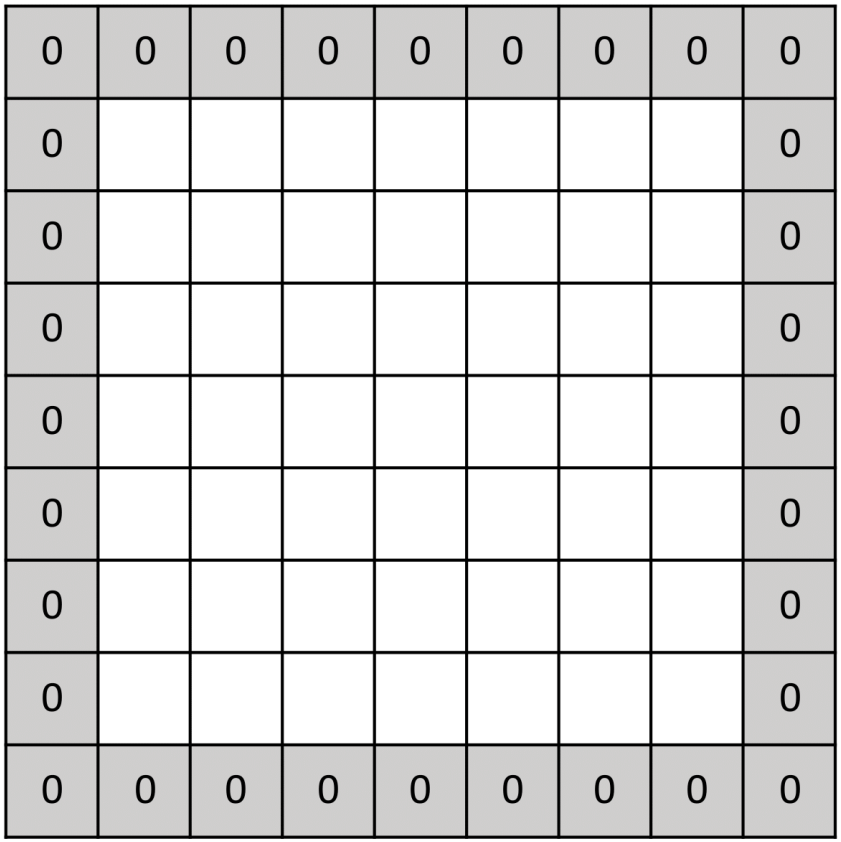
Input: 7x7
Filter: 3x3
Output: 5x5

In general:
Input: W
Filter: K
Output: W – K + 1

Problem: Feature maps "shrink" with each layer!

Solution: **padding**
Add zeros around the input

A closer look at spatial dimensions



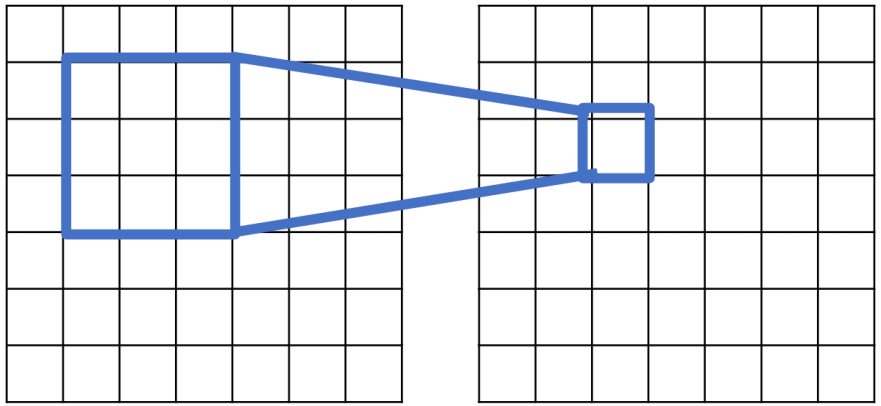
Input: 7x7
Filter: 3x3
Output: 5x5

In general:
Input: W
Filter: K
Padding: P
Output: W – K + 1 + 2P

Very common:
Set P = (K – 1) / 2 to make output have same size as input!

Receptive Fields

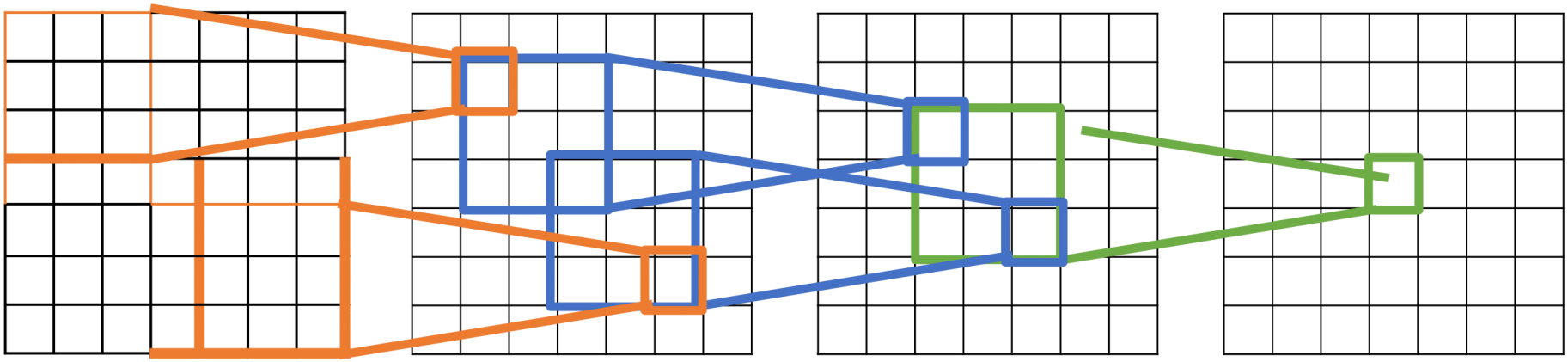
For convolution with kernel size K, each element in the output depends on a K x K **receptive field** in the input



Input Output

Receptive Fields

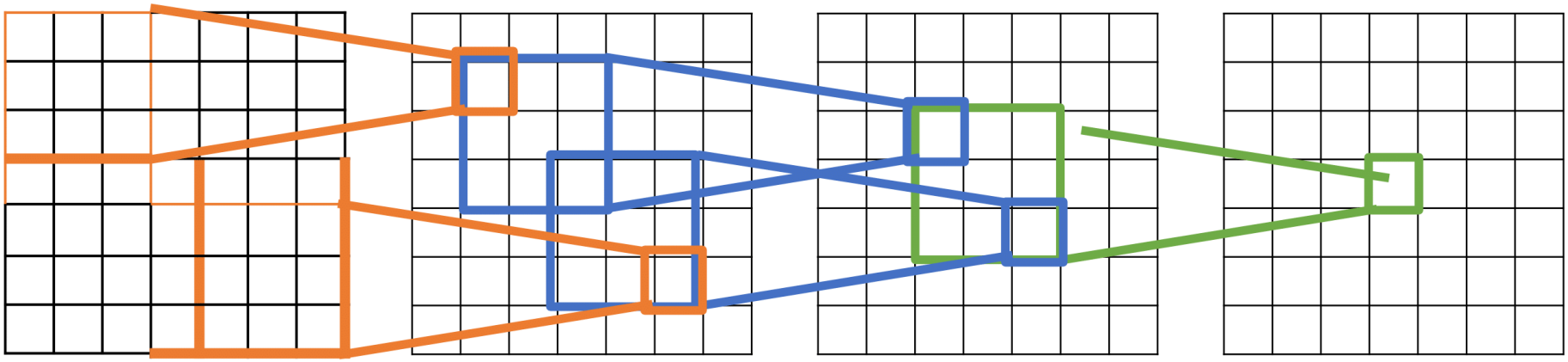
Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)



Be careful – ”receptive field in the input” vs “receptive field in the previous layer”
Hopefully clear from context!

Receptive Fields

Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)



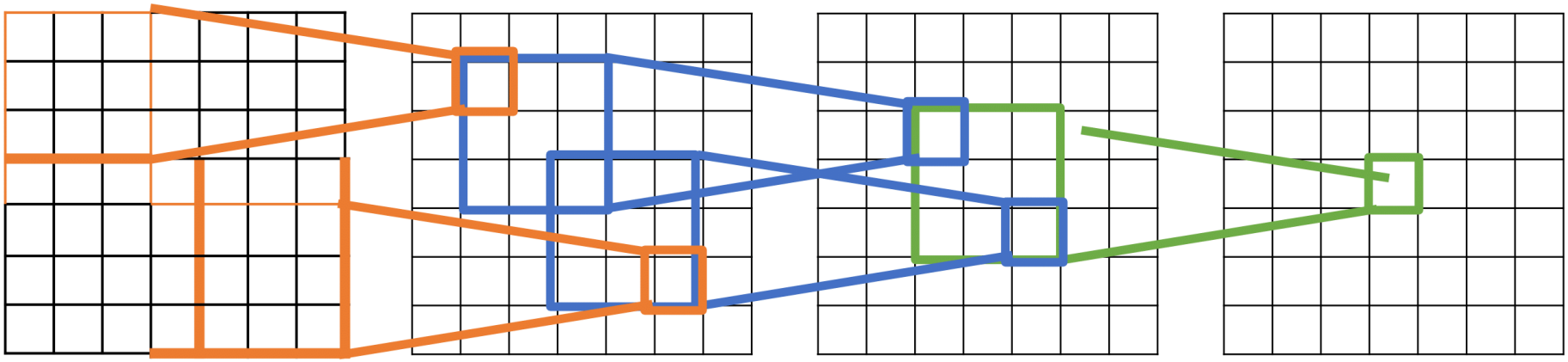
Input

Problem: For large images we need many layers for each output to “see” the whole image image

Output

Receptive Fields

Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)



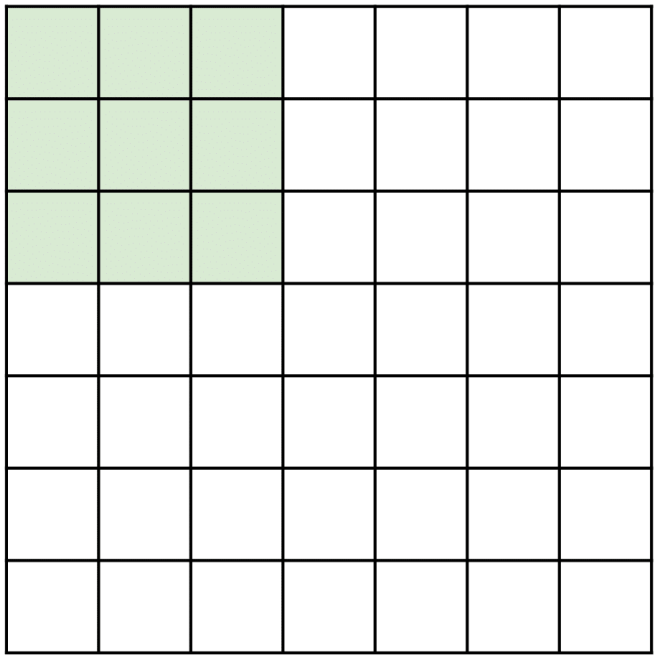
Input

Output

Problem: For large images we need many layers for each output to “see” the whole image image

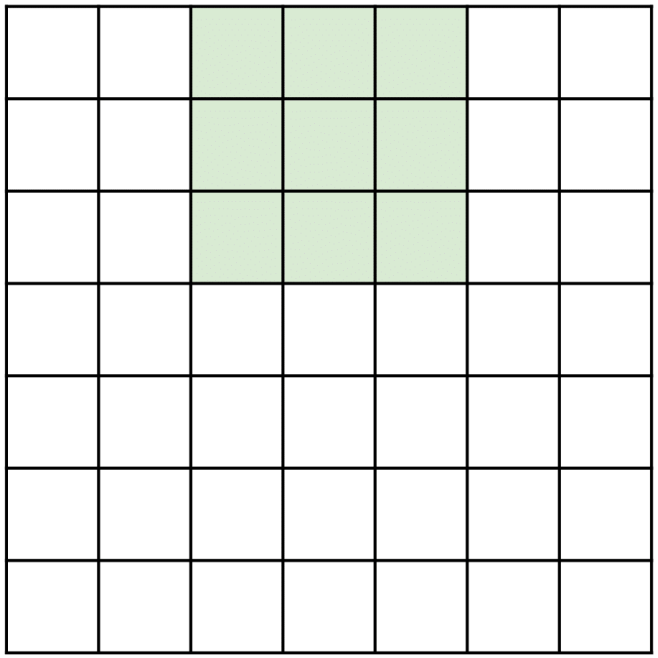
Solution: Downsample inside the network

Strided Convolution



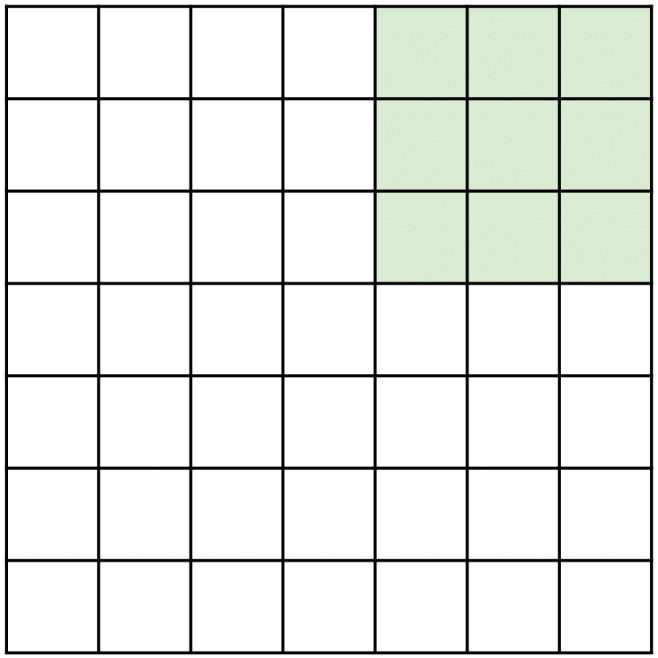
Input: 7x7
Filter: 3x3
Stride: 2

Strided Convolution



Input: 7x7
Filter: 3x3
Stride: 2

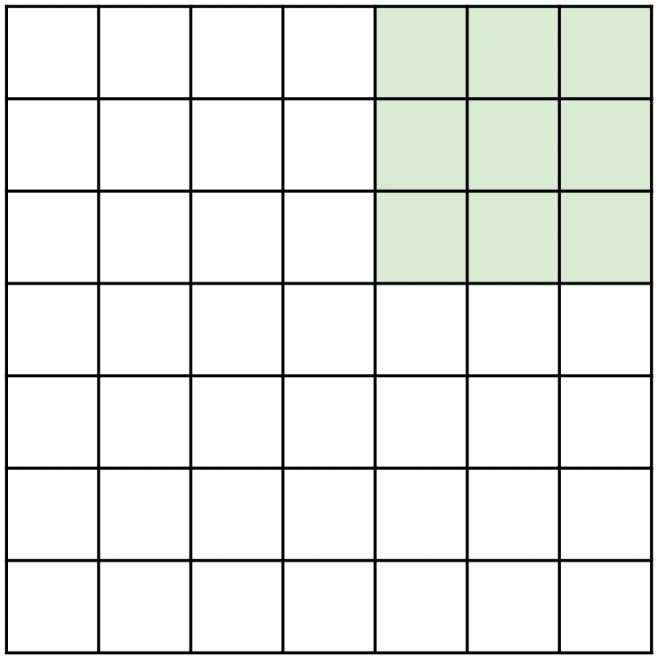
Strided Convolution



Input: 7x7
Filter: 3x3
Stride: 2

Output: 3x3

Strided Convolution



Input: 7x7
Filter: 3x3
Stride: 2

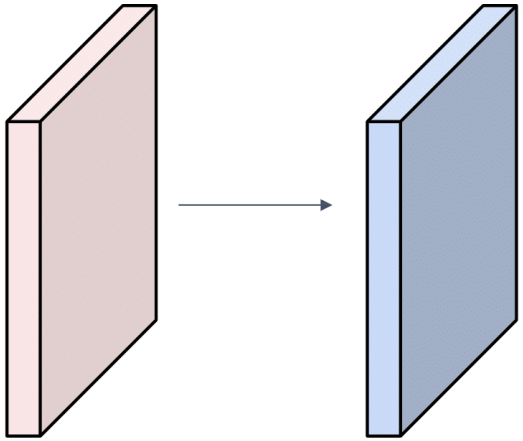
Output: 3x3

In general:
Input: W
Filter: K
Padding: P
Stride: S
Output: (W – K + 2P) / S + 1

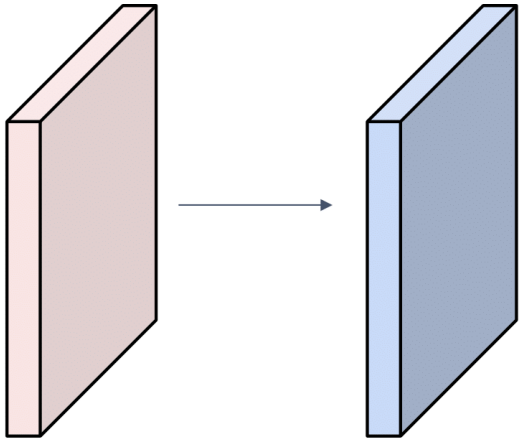
Convolution example

Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

Output volume size: ?



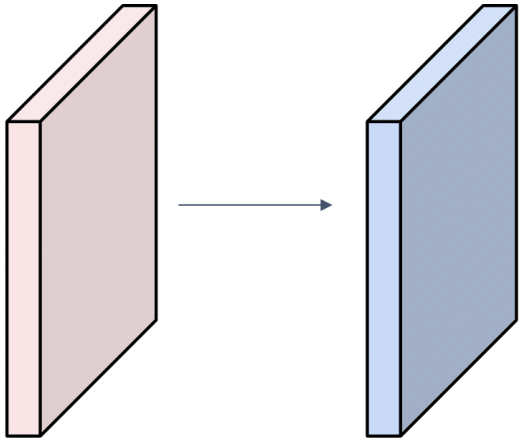
Convolution example



Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
10 x 32 x 32

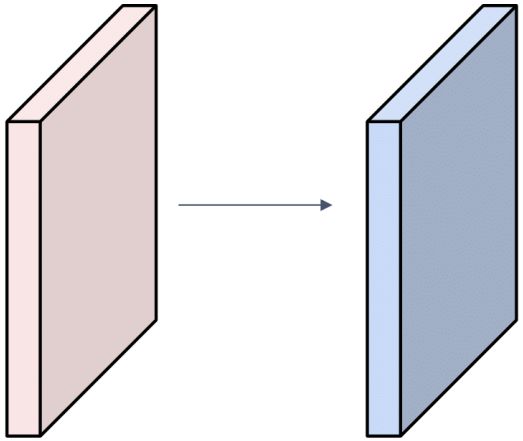
Convolution example



Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

Output volume size: 10 x 32 x 32
Number of learnable parameters: ?

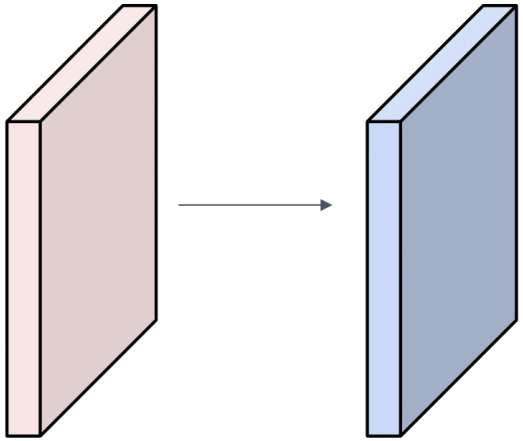
Convolution example



Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

Output volume size: 10 x 32 x 32
Number of learnable parameters: **760**
Parameters per filter: 3*5*5 + 1 (for bias) = 76
10 filters, so total is 10 * 76 = **760**

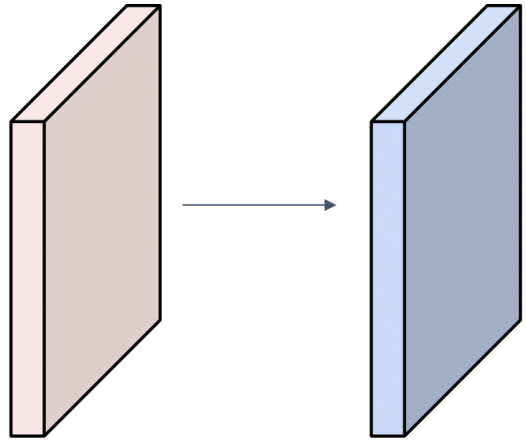
Convolution example



Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

Output volume size: 10 x 32 x 32
Number of learnable parameters: 760
Number of multiply-add operations: ?

Convolution example

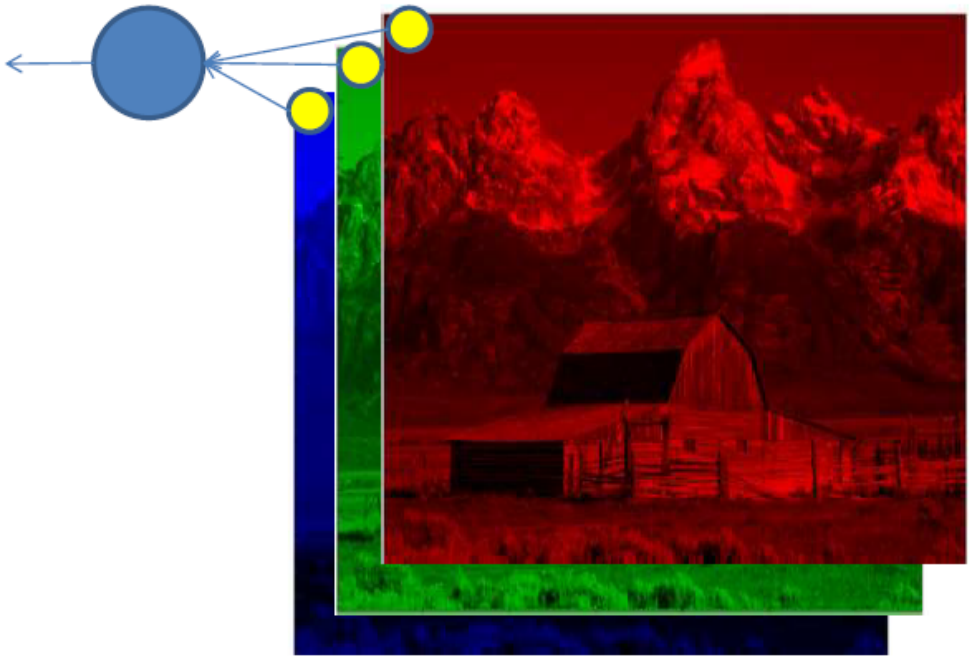


Input volume: **3** x 32 x 32
10 **5x5** filters with stride 1, pad 2

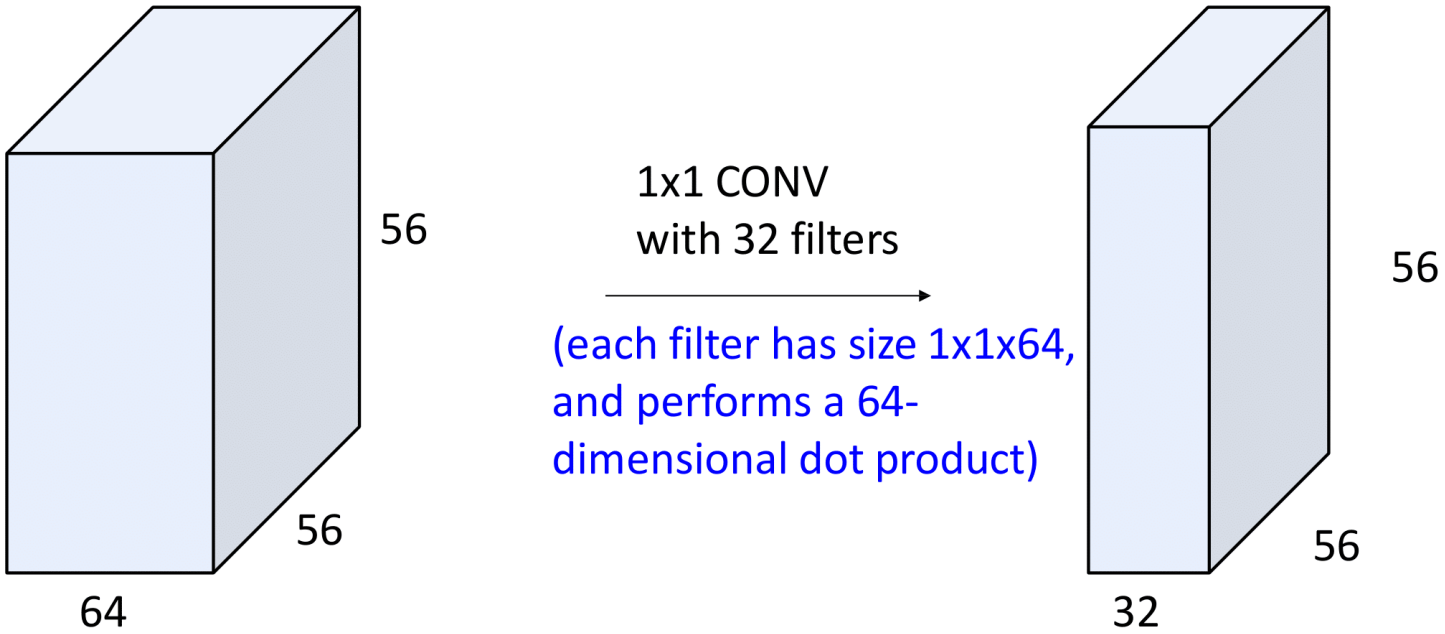
Output volume size: **10 x 32 x 32**
Number of learnable parameters: 760
Number of multiply-add operations: **768,000**
10*32*32 = 10,240 outputs; each output is the inner product of two **3**x**5x5** tensors (75 elems); total = 75*10240 = **768K**

The 1x1 filter

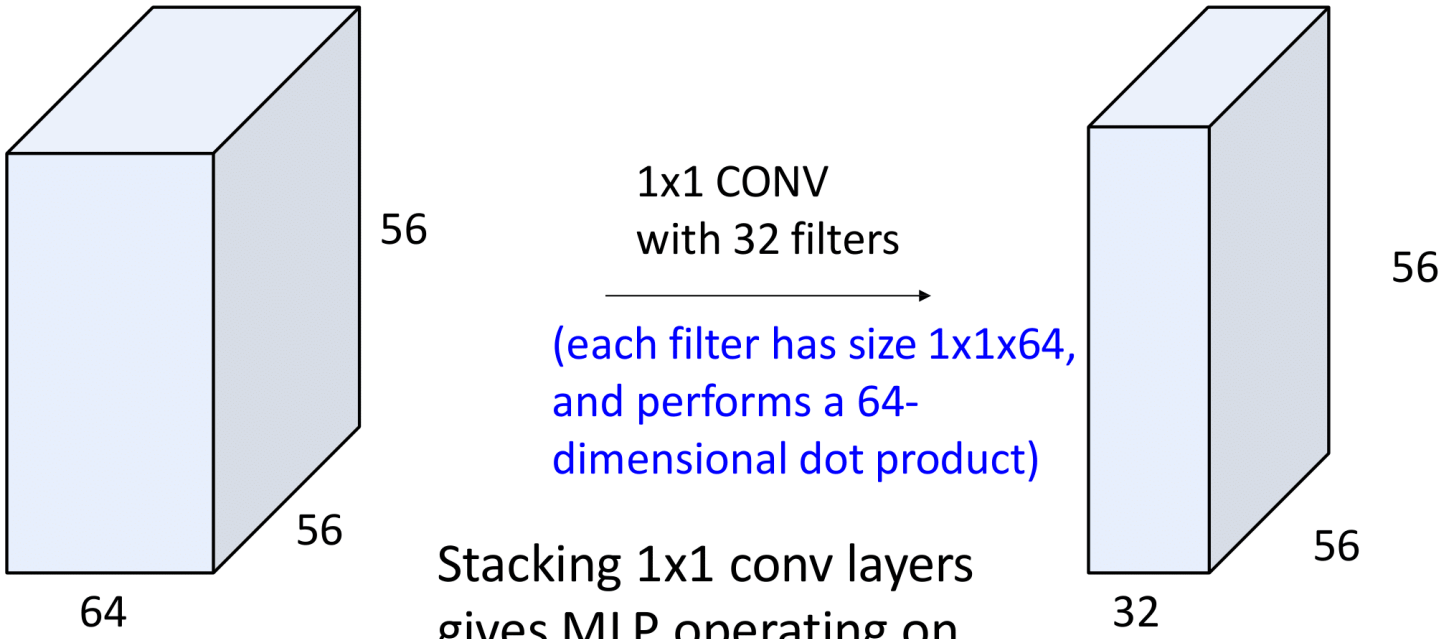
- A 1x1 filter is simply a perceptron that operates over the depth of the stack of maps, but has no spatial extent
  - Takes one pixel from each of the maps (at a given location) as input



Example: 1x1 convolution



Example: 1x1 convolution



Stacking 1x1 conv layers gives MLP operating on each input position

Lin et al, “Network in Network”, ICLR 2014

Convolution summary

Input: C_{in} x H x W

Hyperparameters:

- **Kernel size**: K_H x K_W
- **Number filters**: C_{out}
- **Padding**: P
- **Stride**: S

Weight matrix: C_{out} x C_{in} x K_H x K_W

giving C_{out} filters of size C_{in} x K_H x K_W

Bias vector: C_{out}

Output size: C_{out} x H' x W' where:

- H' = (H – K + 2P) / S + 1
- W' = (W – K + 2P) / S + 1

Convolution summary

Input: C_{in} x H x W

Hyperparameters:

- **Kernel size**: K_H x K_W
- **Number filters**: C_{out}
- **Padding**: P
- **Stride**: S

Weight matrix: C_{out} x C_{in} x K_H x K_W
giving C_{out} filters of size C_{in} x K_H x K_W

Bias vector: C_{out}

Output size: C_{out} x H' x W' where:

- H' = (H – K + 2P) / S + 1
- W' = (W – K + 2P) / S + 1

Common settings:

$K_H = K_W$ (Small square filters)

P = (K – 1) / 2 ("Same" padding)

C_{in}, C_{out} = 32, 64, 128, 256 (powers of 2)

K = 3, P = 1, S = 1 (3x3 conv)

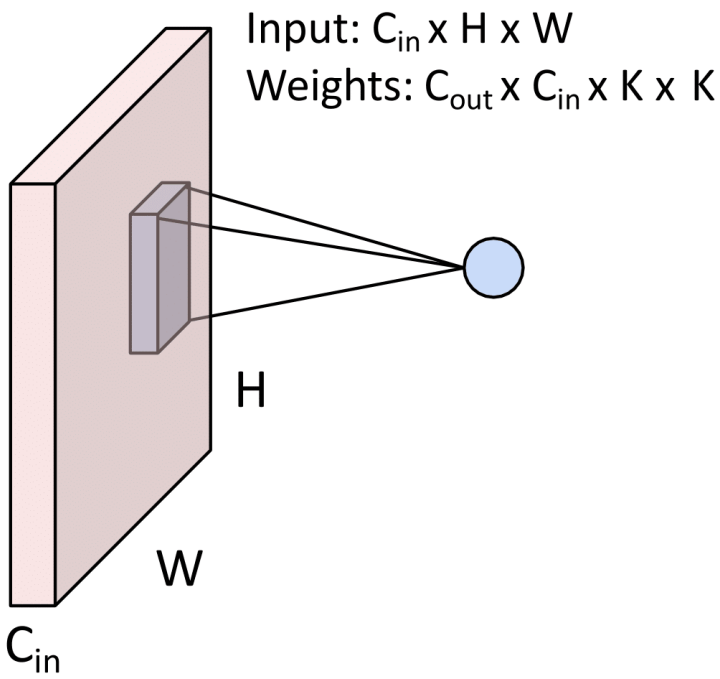
K = 5, P = 2, S = 1 (5x5 conv)

K = 1, P = 0, S = 1 (1x1 conv)

K = 3, P = 1, S = 2 (Downsample by 2)

Other types of convolution

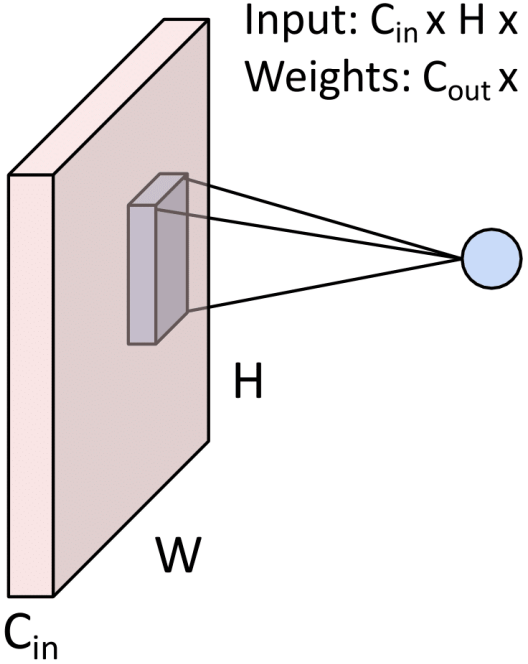
So far: 2D Convolution



Other types of convolution

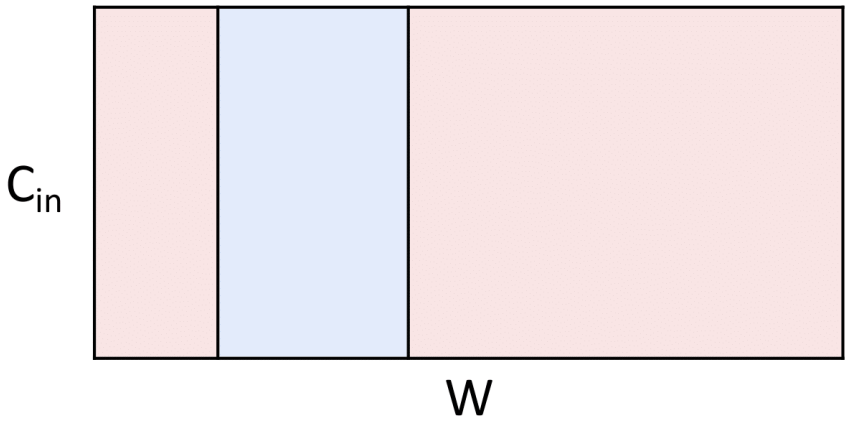
So far: 2D Convolution

Input: C_{in} x H x W
Weights: C_{out} x C_{in} x K x K



1D Convolution

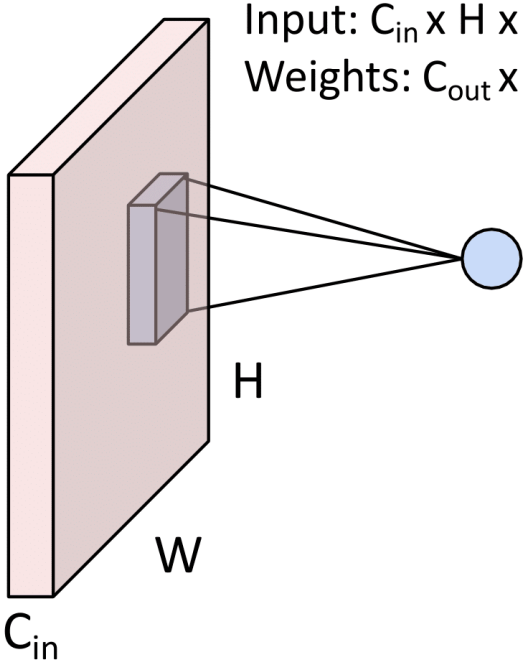
Input: C_{in} x W
Weights: C_{out} x C_{in} x K



Other types of convolution

So far: 2D Convolution

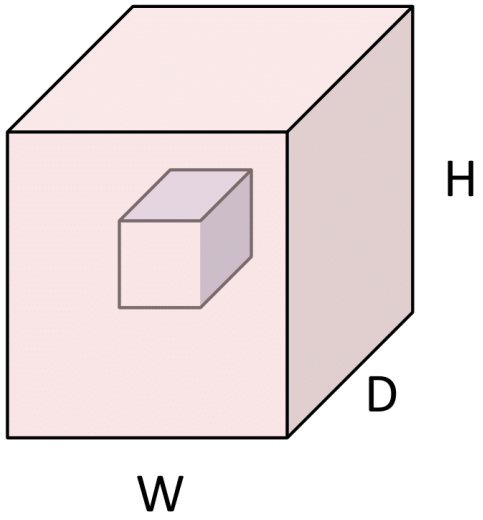
Input: C_{in} x H x W
Weights: C_{out} x C_{in} x K x K



3D Convolution

Input: C_{in} x H x W x D
Weights: C_{out} x C_{in} x K x K x K

C_{in}-dim vector
at each point
in the volume



PyTorch Convolution Layers

Conv2d

CLASS `torch.nn.Conv2d(`*`in_channels`*`, `*`out_channels`*`, `*`kernel_size`*`, `*`stride=1`*`, `*`padding=0`*`, `*`dilation=1`*`, `*`groups=1`*`, `*`bias=True`*`, `*`padding_mode='zeros'`*`)` [SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

PyTorch Convolution Layers

Conv2d

CLASS `torch.nn.Conv2d(`*in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros'*`)` [SOURCE]

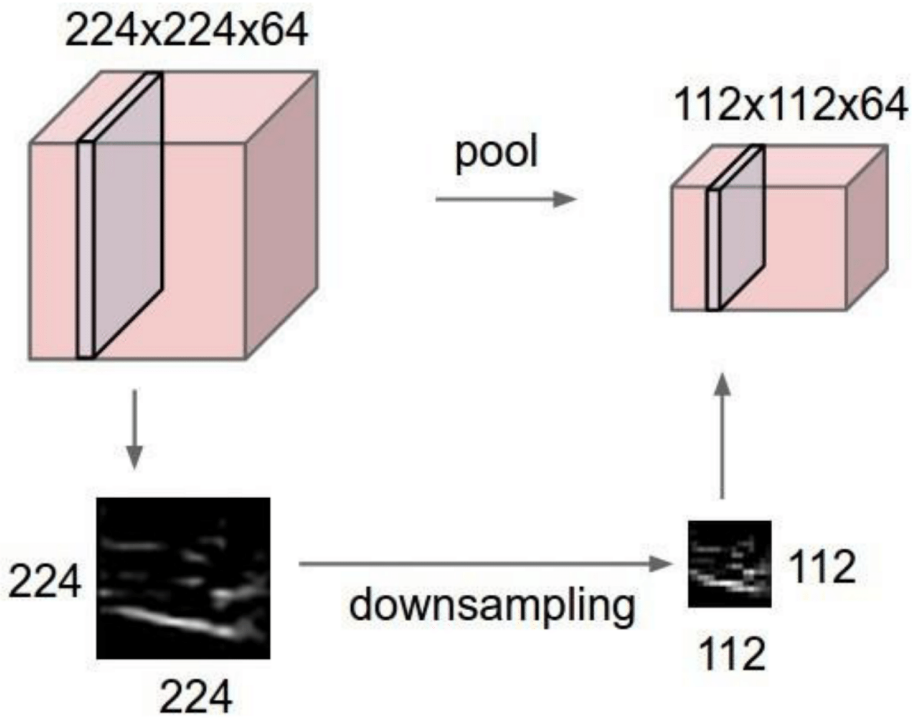
Conv1d

CLASS `torch.nn.Conv1d(`*in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros'*`)` [SOURCE]

Conv3d

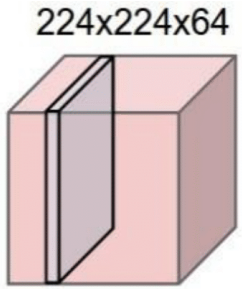
CLASS `torch.nn.Conv3d(`*in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros'*`)` [SOURCE]

Pooling Layers: Another way to downsample

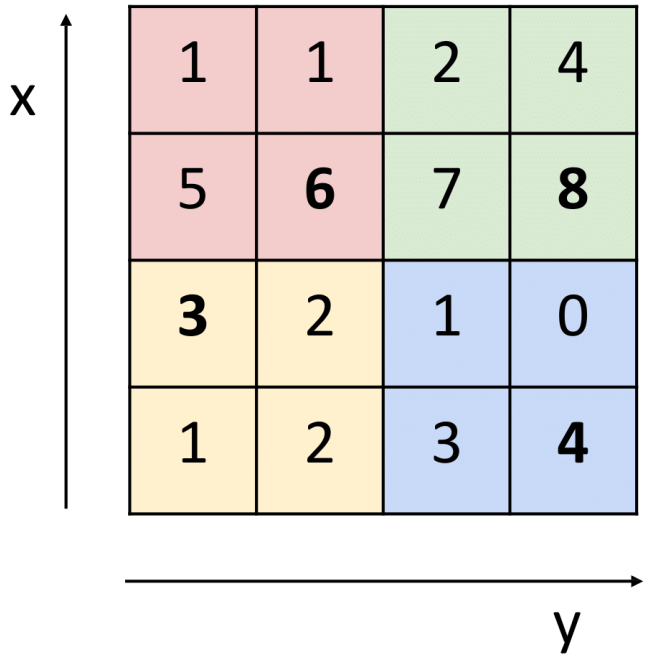


Hyperparameters:
Kernel Size
Stride
Pooling function

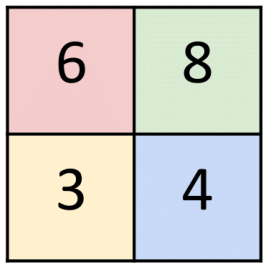
Max Pooling



Single depth slice



Max pooling with 2x2 kernel size and stride 2



Introduces **invariance** to small spatial shifts

No learnable parameters!

Pooling Summary

Input: C x H x W

Hyperparameters:

- Kernel size: K
- Stride: S
- Pooling function (max, avg)

Output: C x H' x W' where

- H' = (H – K) / S + 1
- W' = (W – K) / S + 1

Learnable parameters: None!

Common settings:

max, K = 2, S = 2

max, K = 3, S = 2 (AlexNet)

References

- Deep Learning, Ian Goodfellow, MIT Press, Ch. 9