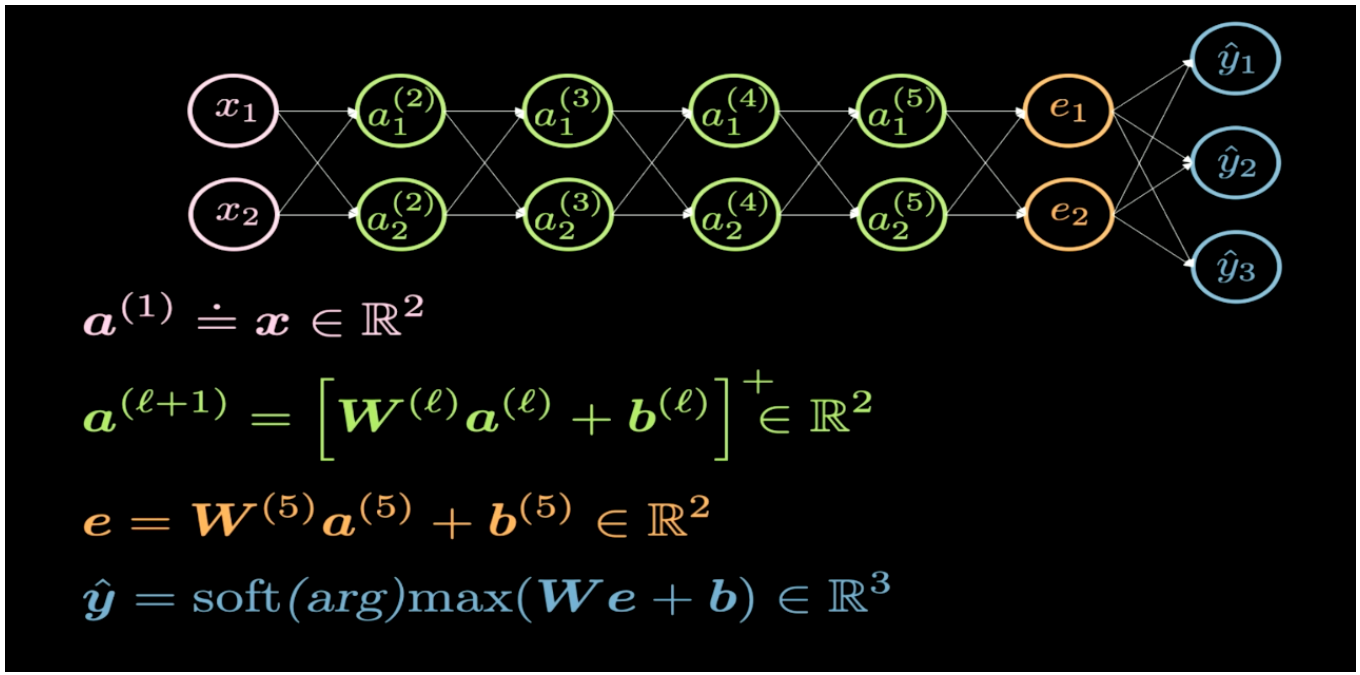
Deep Learning Refresher

Week 3

Tirtharaj Dash

Visualisation of Neurla Nets I

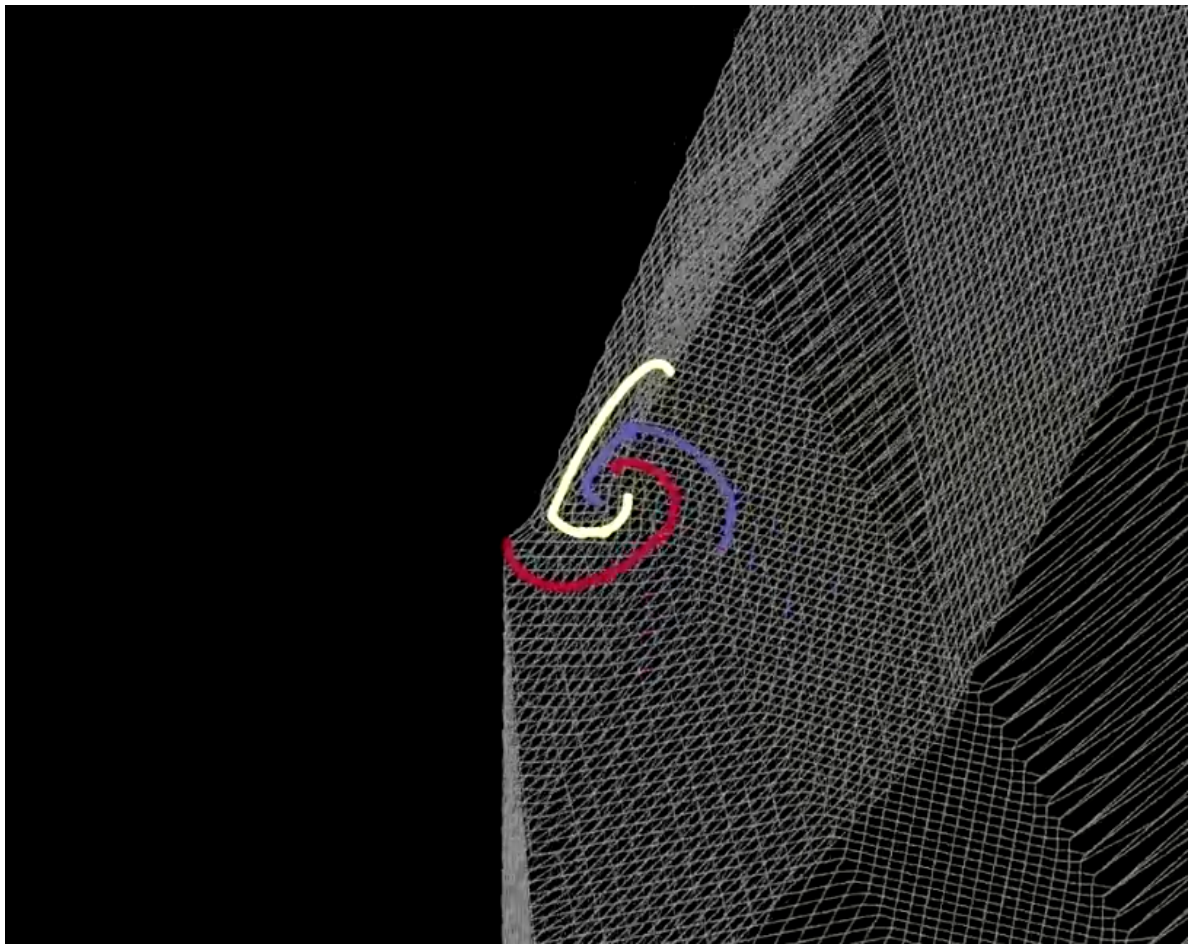
- Here is a multilayered network:



- The weights in each layer are: $[2 \times 2]$.
- Each layer outputs a transformed version of the input. This is easy to visualise.

Visualisation of Neurla Nets II

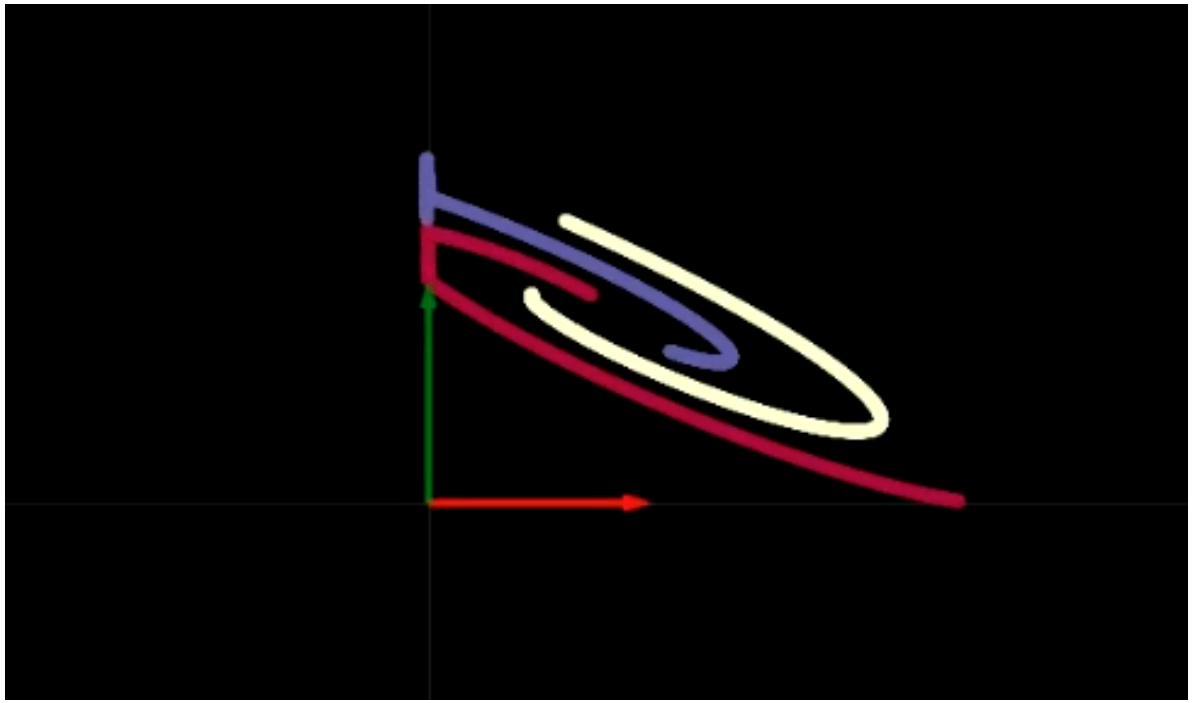
- The transformation in each layer is like folding our plane in some specific regions like this:



Visualisation of Neurla Nets III

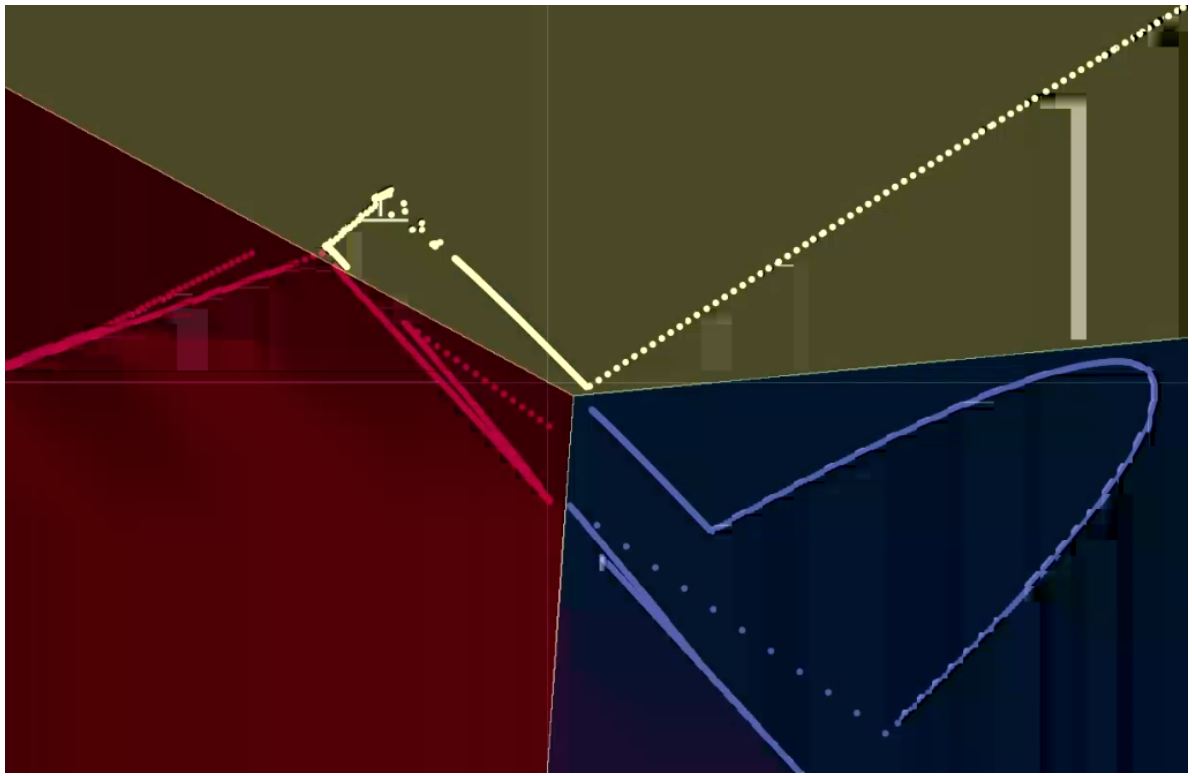
- This folding is very abrupt, this is because all the transformations are performed in the 2D layer.
- ReLU then converts the above transformation nonlinearly and this is what it looks like:

Visualisation of Neurla Nets IV



Visualisation of Neurla Nets V

- After multiple levels of transformations, we get a representation of data that is linearly separable:

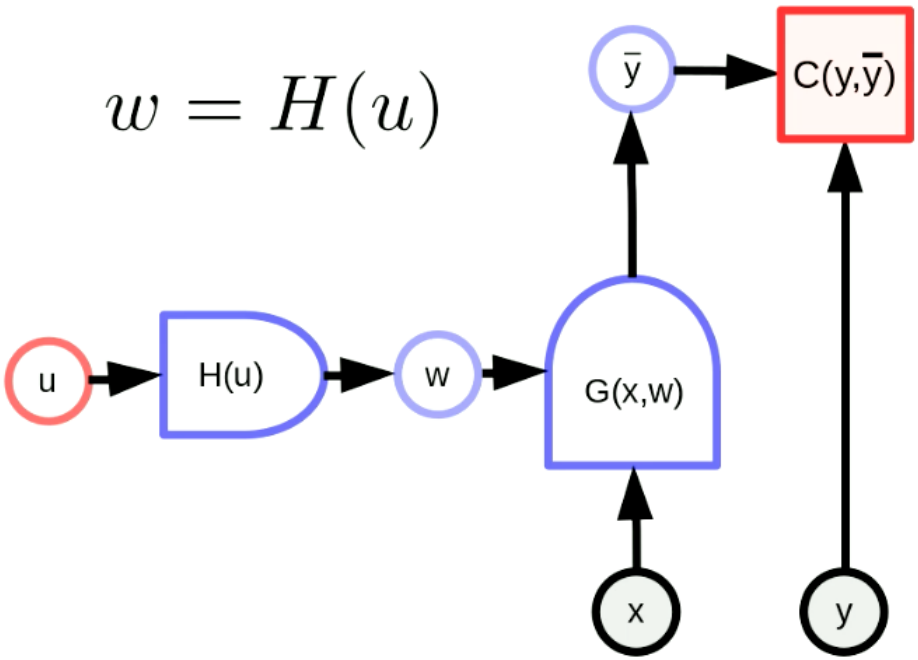


- As discussed in Week1, it will be easy for the network to train if the transformation was higher (> 2) dimensional.

Parameter Transformation I

- The parameter vector w is output of a function. (Original paramter space is mapped into another space)
- A simple parameter transformation

$$w = H(u)$$



Parameter Transformation II

- w is output of H with paramter u.
- $G(x, w)$ is the network and $C(y, \bar{y})$ is the cost.
- The backprop for u and w are:

$$u \leftarrow u - \eta \frac{\partial C}{\partial \bar{y}} \frac{\partial G(x, w)}{\partial w} \frac{\partial H}{\partial u}$$

The dimensions of these terms should be consistent. These are:

- $u : [N_u \times 1]$
- $w : [N_w \times 1]$
- $\left(\frac{\partial H}{\partial u}\right)^{\mathsf{T}} : [N_u \times N_w]$
- $\left(\frac{\partial C}{\partial \bar{y}}\right)^{\mathsf{T}} : [N_y \times 1]$
- $\left(\frac{\partial G(x,w)}{\partial w}\right)^{\mathsf{T}} : [N_w \times N_y]$

Parameter Transformation III

- So to make things consistent, we have to move around these terms:

$$[N_u \times 1] = [N_u \times N_w] \times [N_w \times N_y] \times [N_y \times 1]$$

that is:

$$u \leftarrow u - \eta \left(\frac{\partial H}{\partial u}\right)^{\mathsf{T}} \left(\frac{\partial G(x,w)}{\partial w}\right)^{\mathsf{T}} \left(\frac{\partial C}{\partial \bar{y}}\right)^{\mathsf{T}}$$

We can combine the last two terms (as in lecture):

$$u \leftarrow u - \eta \left(\frac{\partial H}{\partial u}\right)^{\mathsf{T}} \left(\frac{\partial C}{\partial w}\right)^{\mathsf{T}}$$

Parameter Transformation IV

- Update to w: Now, since w is an output of H, which has the parameter u. Also, we have now the gradient of cost w.r.t u. So, what is the change in w due to our above change in u?
- It is:

$$w \leftarrow H\left(u - \eta \left(\frac{\partial H}{\partial u}\right)^{\mathsf{T}} \left(\frac{\partial C}{\partial w}\right)^{\mathsf{T}}\right)$$

approximation using Taylor series:

$$w \leftarrow H(u) - \eta H'(u) \left(\frac{\partial H}{\partial u}\right)^{\mathsf{T}} \left(\frac{\partial C}{\partial w}\right)^{\mathsf{T}}$$

or,

$$w \leftarrow w - \frac{\partial H}{\partial u} \left(\frac{\partial H}{\partial u}\right)^{\mathsf{T}} \left(\frac{\partial C}{\partial w}\right)^{\mathsf{T}}$$

A simple parameter sharing: Weight sharing

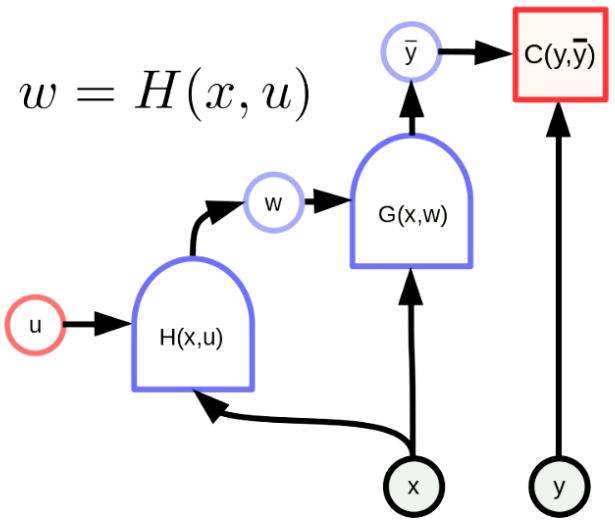
- $H(u)$ just replicates one component of u into multiple components of w.
- $H(u)$ is like a Y-branch to copy u_1 to w_1 and w_2.

$$w_1 = w_2 = u_1; w_3 = w_4 = u_2$$

- We force shared parameters to be equal, so the gradient w.r.t. to shared parameters will be summed in the backprop.
- For example the gradient of the cost function $C(y, \bar{y})$ with respect to u_1 will be the sum of the gradient of the cost function $C(y, \bar{y})$ with respect to w_1 and w_2.

Hypernetwork

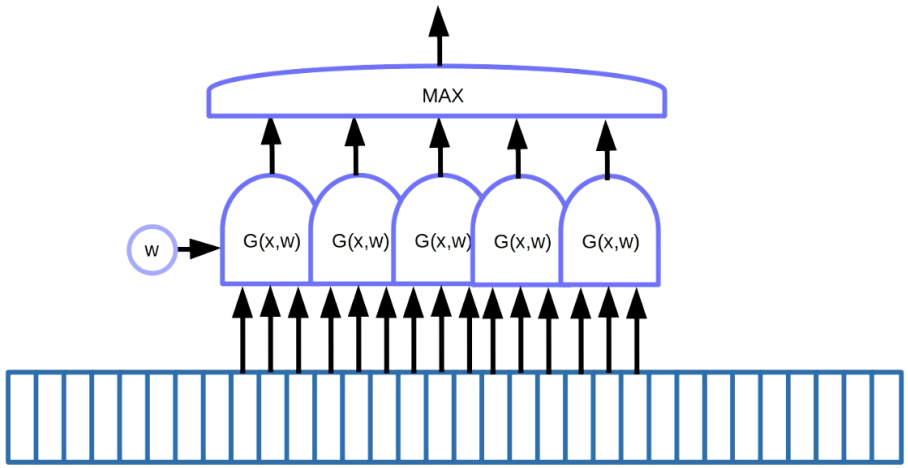
- A hypernetwork is a network where the weights of one network is the output of another network.



- The network G is dynamically configured by H network.
- The idea is that: Use a small network (H) to generate weights for a main (large) network (G).
- See this paper: `https://arxiv.org/pdf/1609.09106.pdf`

Motif detection in sequential data

- Motif detection means to find some motifs in sequential data like keywords in speech or text.
- Idea: Use a sliding window on data, which moves the weight-sharing function to detect a particular motif (i.e. a particular sound in speech signal), and the outputs (i.e. a score) goes into a maximum function.



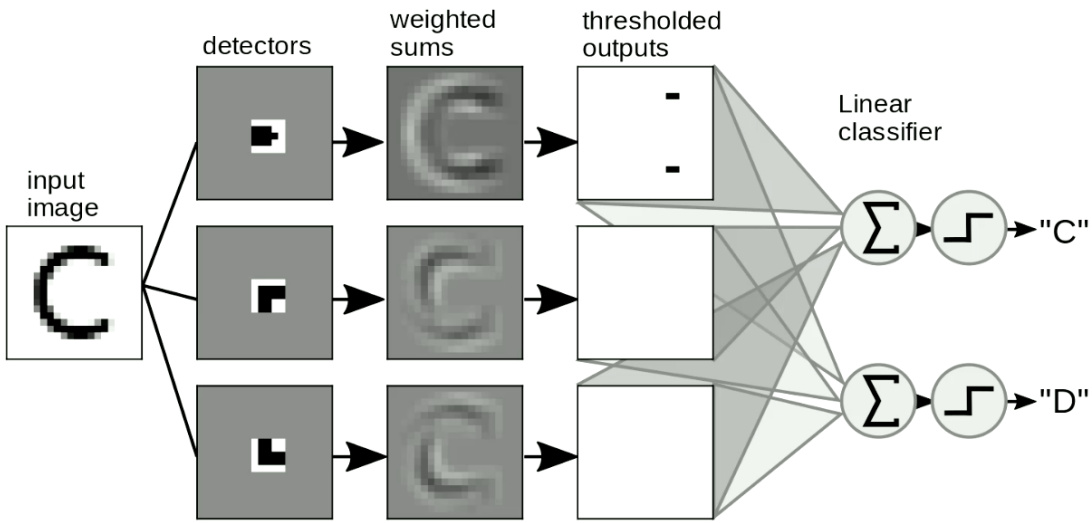
- There are 5 such functions $G()$s.
- For backprop, the five gradients are summed up to update w.

Motif detection in images I

- Slide a "templates" (or multiple templates) over an image to detect the shapes independent of position and distortion of the shapes.
- E.g. distinguish between two images: one of character 'C' and another 'D'.
- 'C' has two endpoints; 'D' has two corners.
- So there are three templates: endpoint template, corner templates.

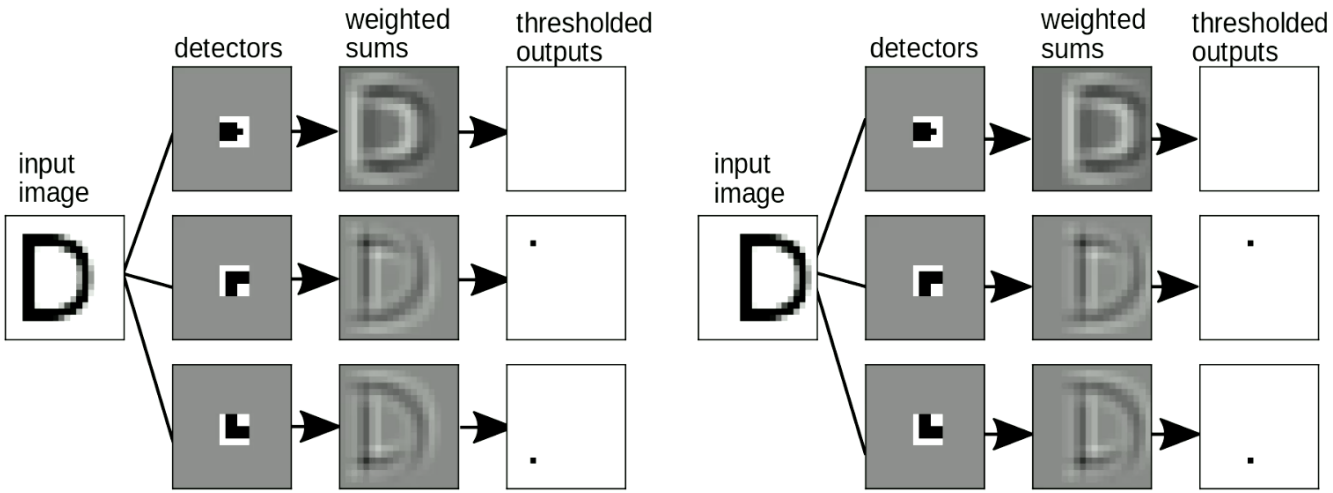
Motif detection in images II

- 'C' (image) has no corners but has endpoints so that output for 'C' is activated.



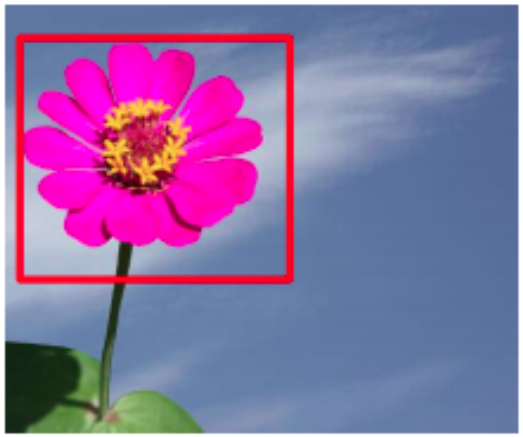
Motif detection in images III

- It is also important that our "template matching" should be shift-invariant: When we shift the input, the output (i.e. the letter detected) shouldn't change. This can be solved with weight sharing transformation.



Motif detection in images IV

Another example:



The output must be correct regardless of the precise location of the target object (here, flower).

What weight sharing does is a kind of scanning over the original image with the weight template.

Motif detection in images V

- This was the approach for long time: hand-crafted templates.
- Questions:
  - How to design 'templates' automatically?
  - How many templates to be designed?
  - What about rotations, noisy images etc?
  - Can we use neural nets to learn (or construct) these?
  - ...

Discrete Convolution I

Convolution:

- In 1-dimensional case:

$$y_i = \sum_j w_j x_{i-j}$$

- The i-th output is computed as the dot product between the reversed w and a window of the same size in x.
- To compute the full output, start the window at the beginning, shift this window by one entry each time and repeat until x is exhausted.

Discrete Convolution II

Cross-correlation:

- Most common deep learning libraries don't reverse w so it becomes cross-correlation:

$$y_i = \sum_j w_j x_{i+j}$$

- It is however a method of convention (with due respect to signal processing guys :-)) based on how you read w from memory.

Discrete Convolution III

Higher-dimensional convolution:

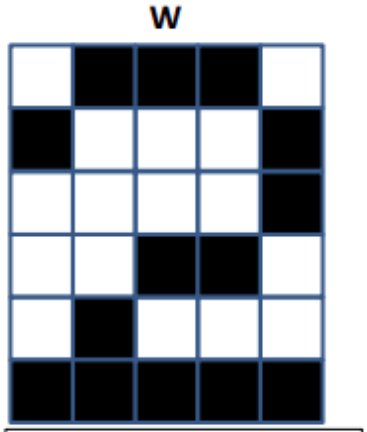
- For two dimensional inputs such as images, we make use of the two dimensional version of convolution:

$$y_{ij} = \sum_{kl} w_{kl} x_{i+k,j+l}$$

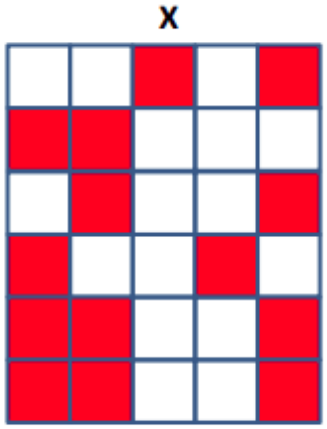
- Can be extended beyond two dimensions to three or four dimensions.
- Here w is called the convolution kernel.

Discrete Convolution IV

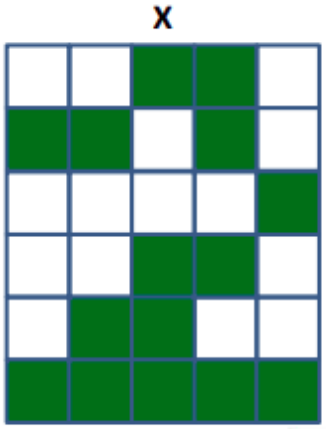
Example: Checking if an image is of the digit '2'



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



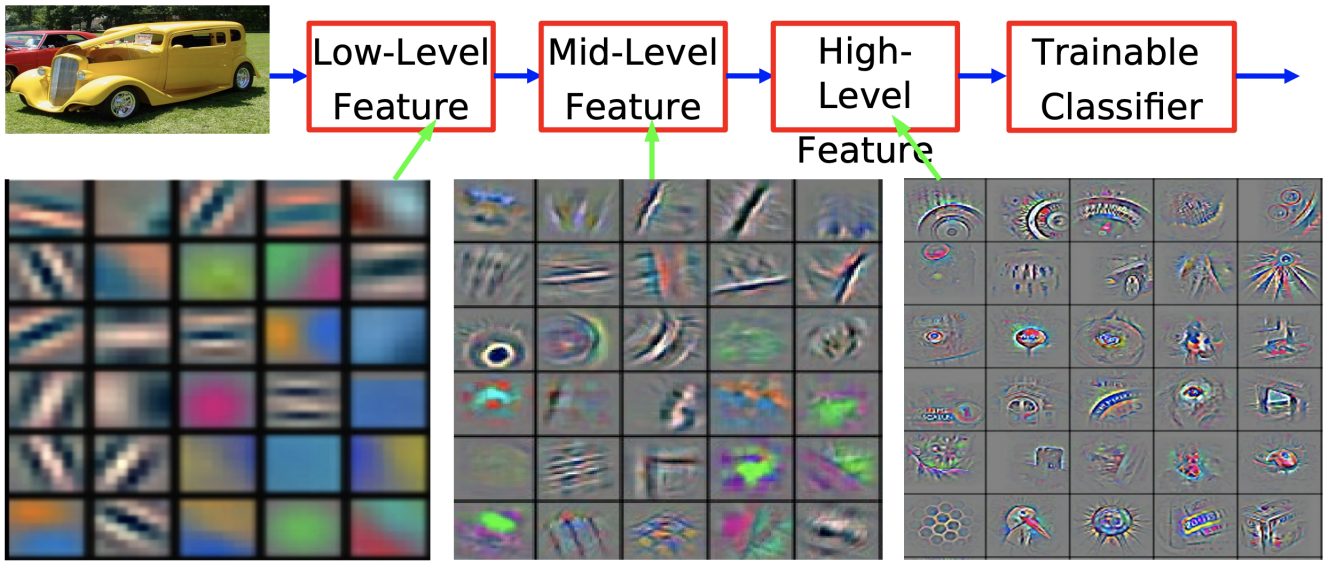
Correlation = 0.57



Correlation = 0.82

Deep Convolutional Neural Network I

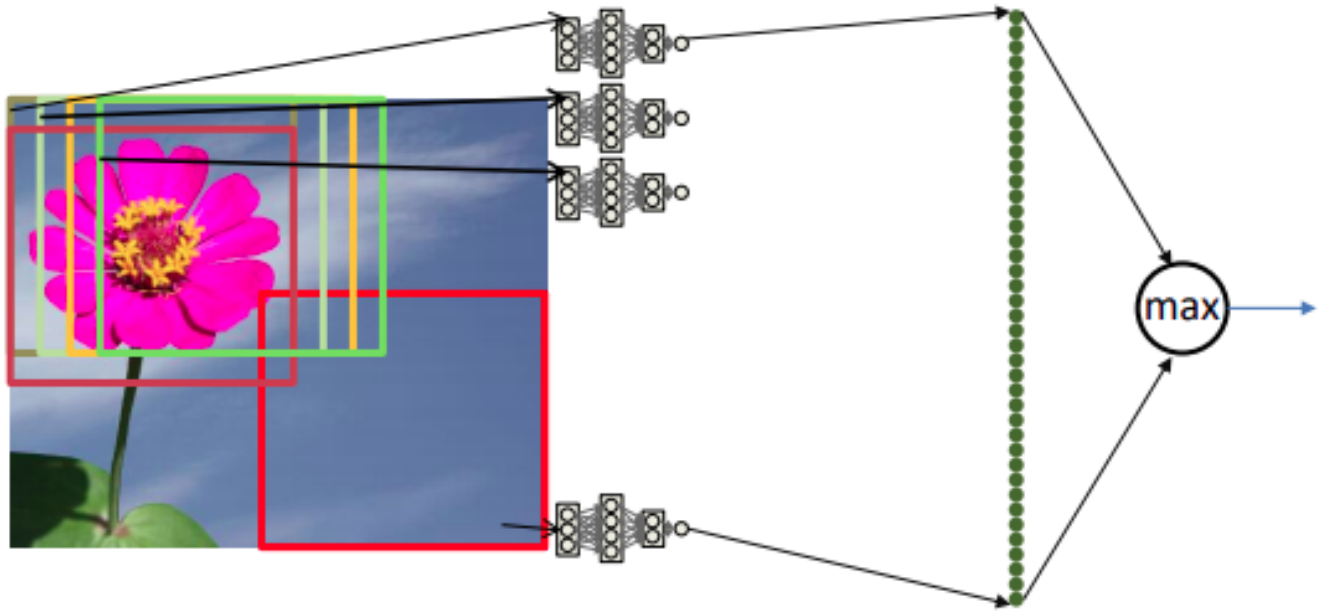
- DNNs: `repeat`(linear transofrmation, non-linearity)
- CNNs: `repeat`(convolution, non-linearity, [pooling])
- Reason for stacking such layers: build a hierarchical representation of the data.
- Why would we want to capture the hierarchical representation of the world? — Because the world we live in is compositional.



Deep Convolutional Neural Network II

Scanning an image to recognise a flower:

- This is how it is usually done:
  - We get one classification output per scanned location
  - The score output by the MLP
  - Look at the maximum value
  - Or, pass it through an MLP

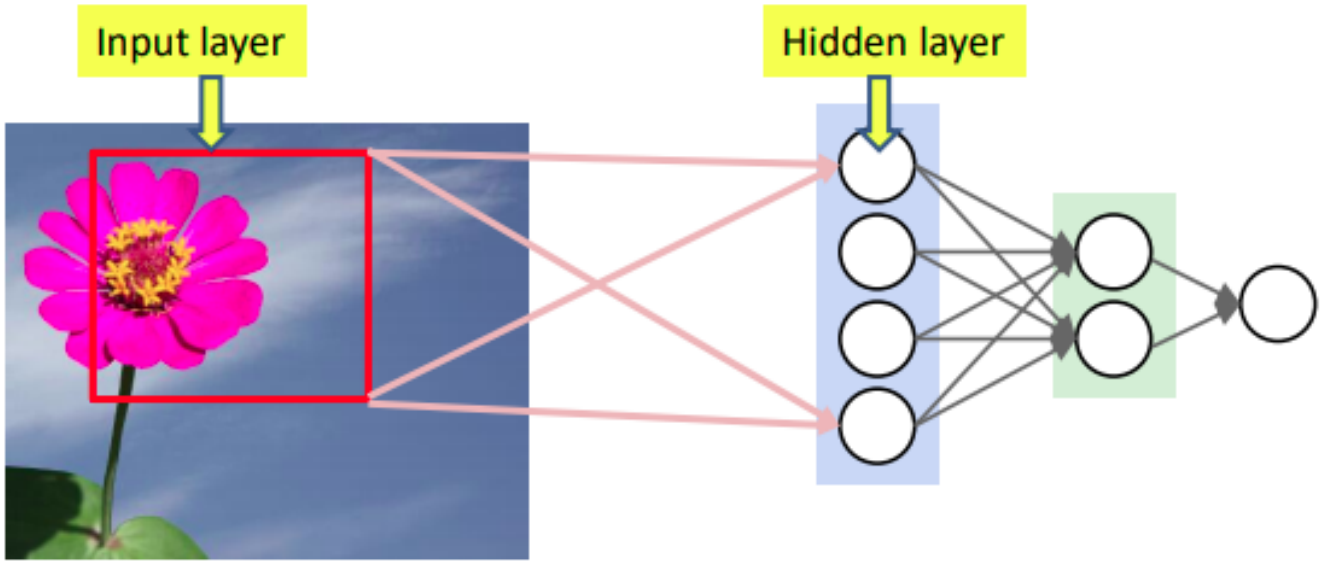


Deep Convolutional Neural Network III

- The entire operation can be viewed as a single giant network
  - Composed of many "subnets" (one per window)
  - With one key feature: all subnets are identical
  - These are shared parameter networks: all are searching for the same pattern
  - Any update of the parameters of one copy of the subnet must equally update all copies

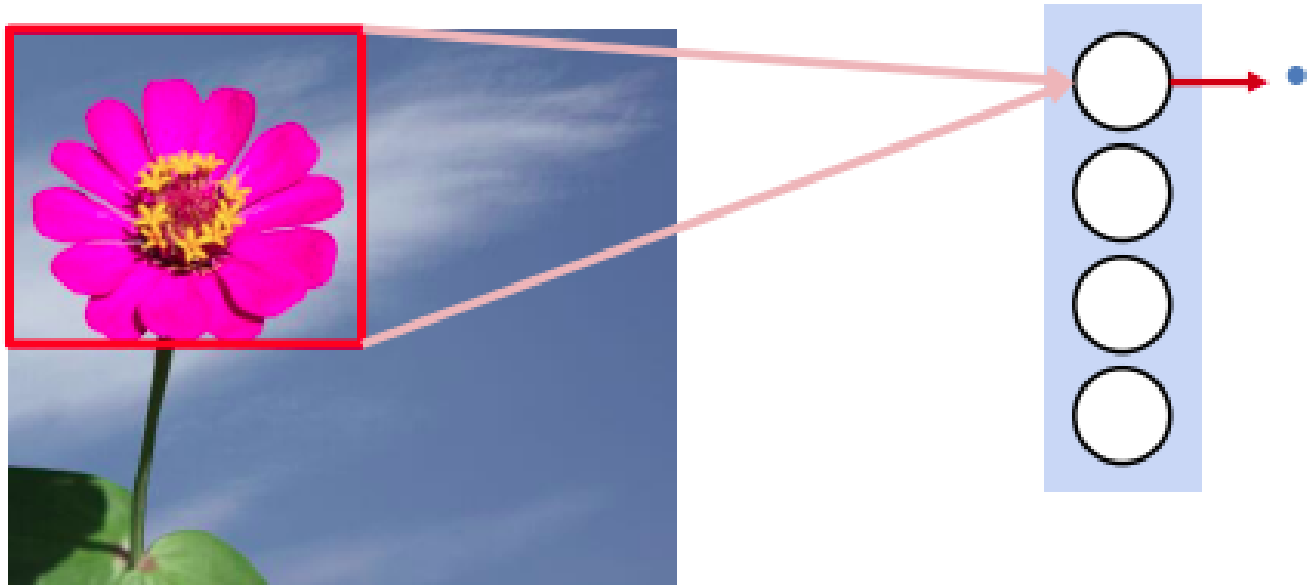
Deep Convolutional Neural Network IV

- If we closely look at any small network:

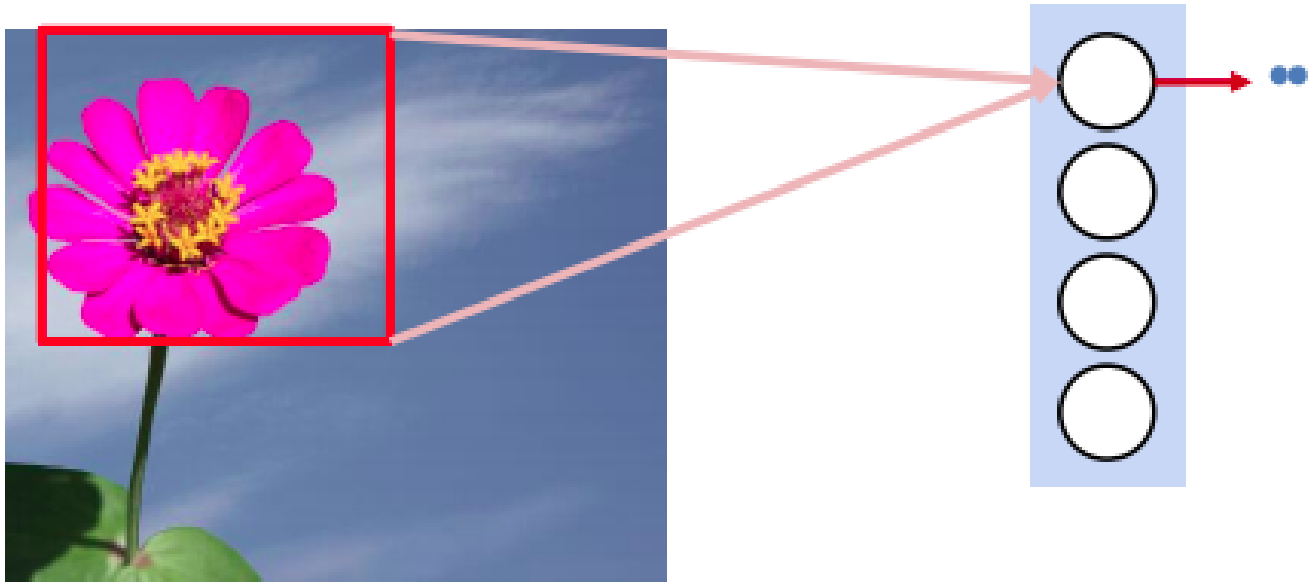


- At each position of the box, the perceptron is evaluating the part of the picture in the box as part of the classification for that region.
- Outputs are then arranged as we will see next.

Deep Convolutional Neural Network V



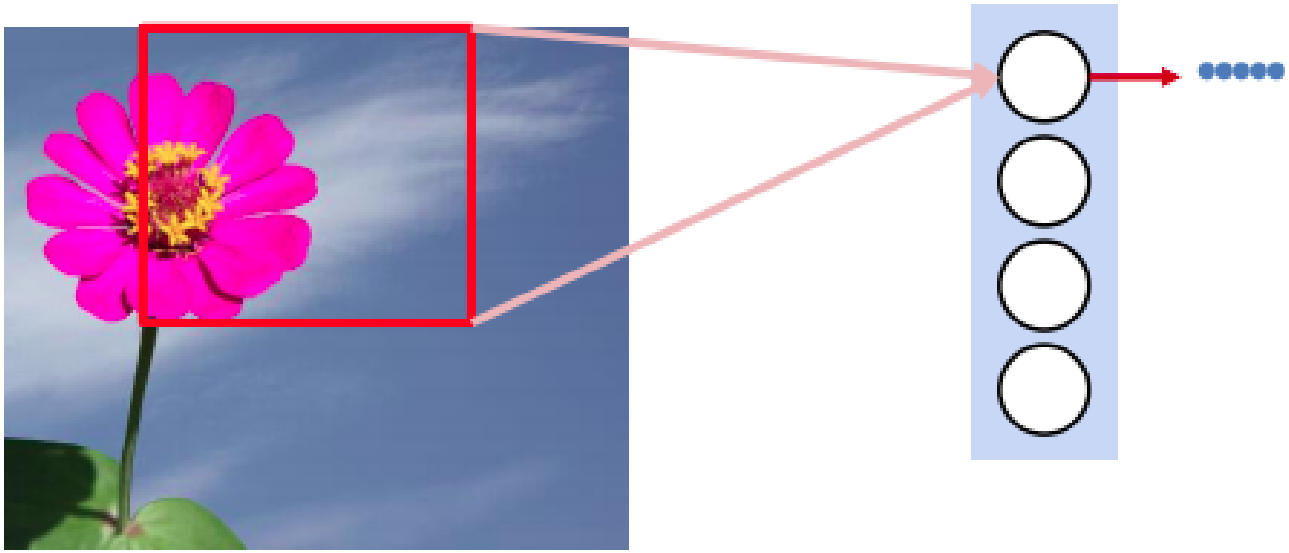
Deep Convolutional Neural Network VI



Deep Convolutional Neural Network VII

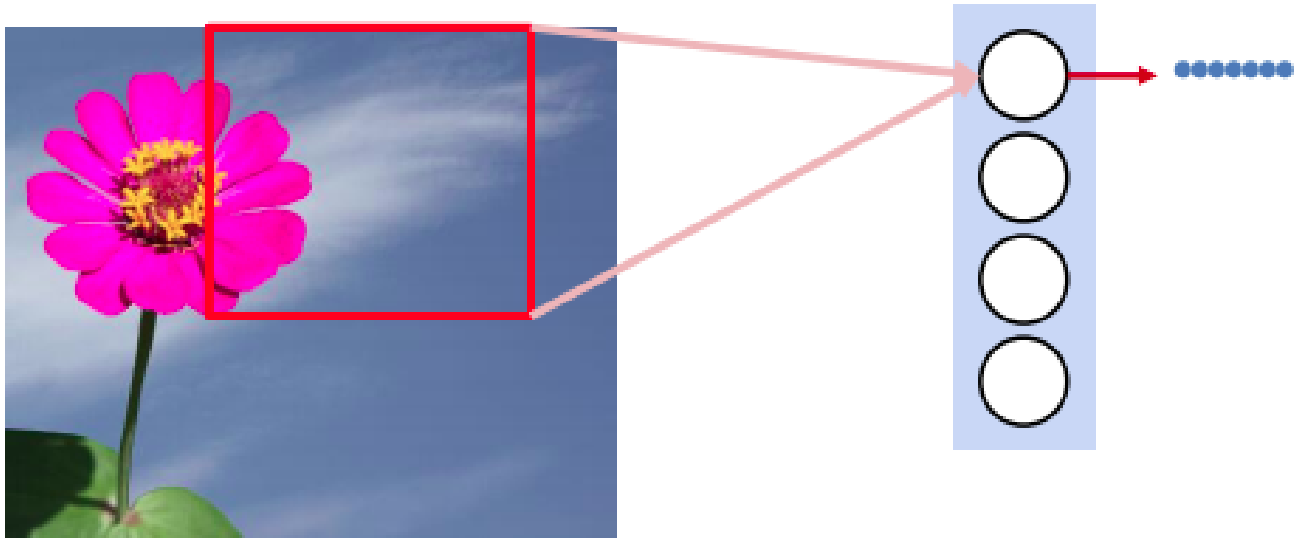
Deep Convolutional Neural Network VIII

Deep Convolutional Neural Network IX

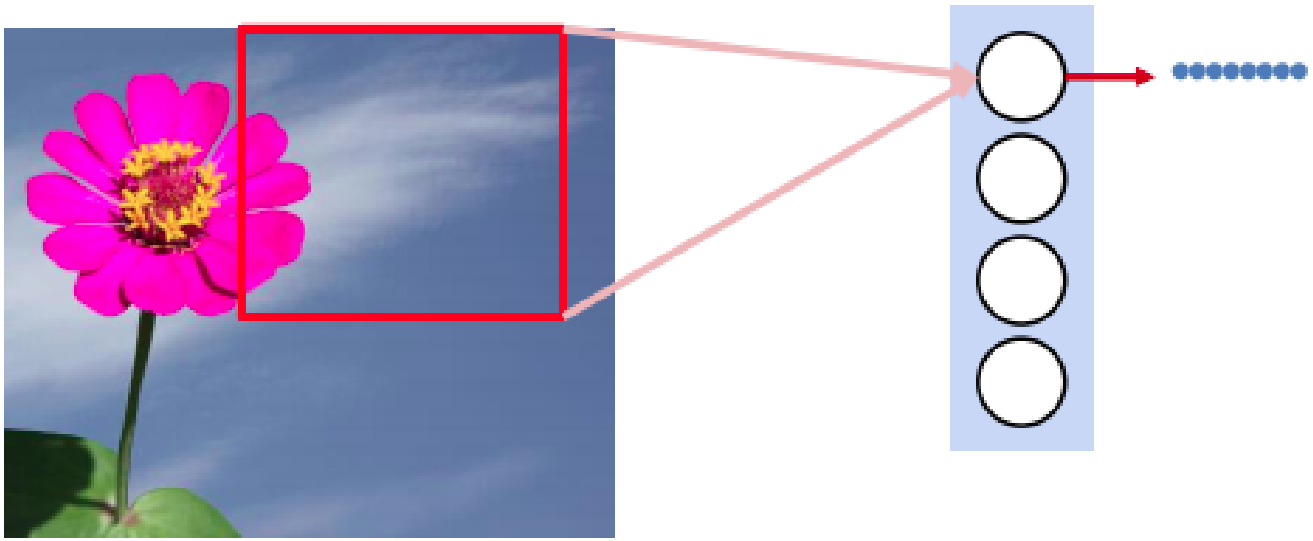


Deep Convolutional Neural Network X

Deep Convolutional Neural Network XI



Deep Convolutional Neural Network XII



Deep Convolutional Neural Network XIII

Deep Convolutional Neural Network XIV

Deep Convolutional Neural Network XV

Deep Convolutional Neural Network XVI

Deep Convolutional Neural Network XVII

Deep Convolutional Neural Network XVIII

Deep Convolutional Neural Network XIX

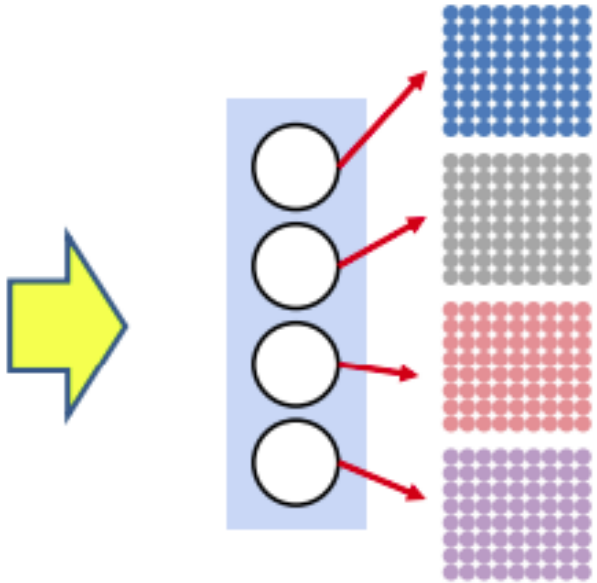
The same idea can be recursively applied:

- The second level neurons too are "scanning" the rectangular outputs of the first-level neurons
- But, they are jointly scanning **multiple** "pictures"

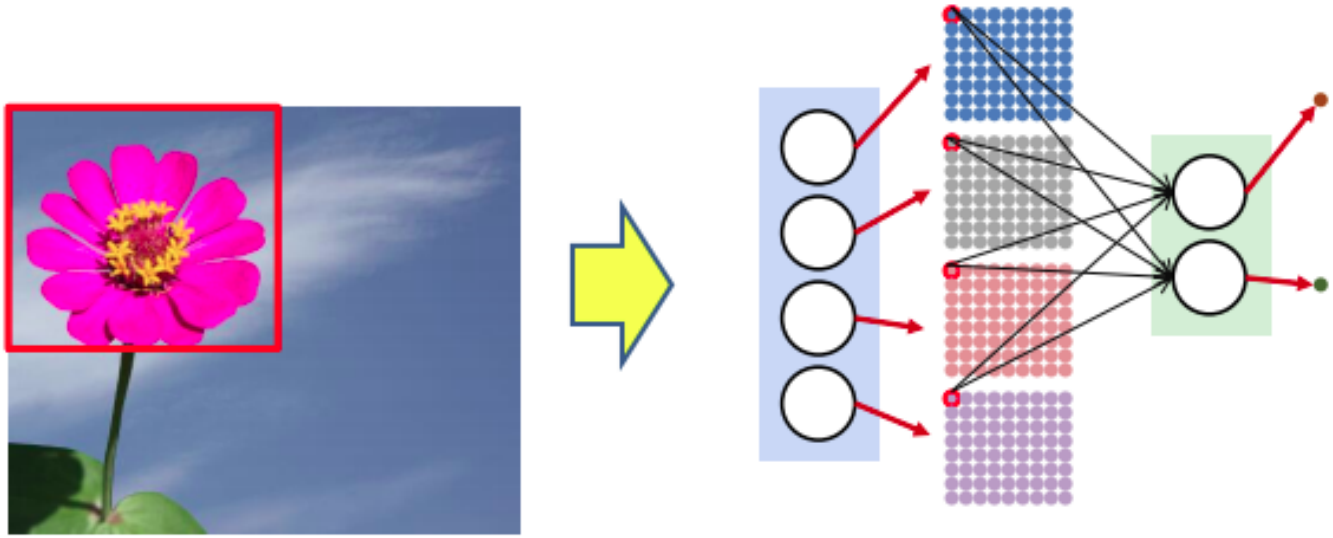
Deep Convolutional Neural Network XX

Each location in the output of the second level neuron considers the corresponding locations from the outputs of all the first-level neurons

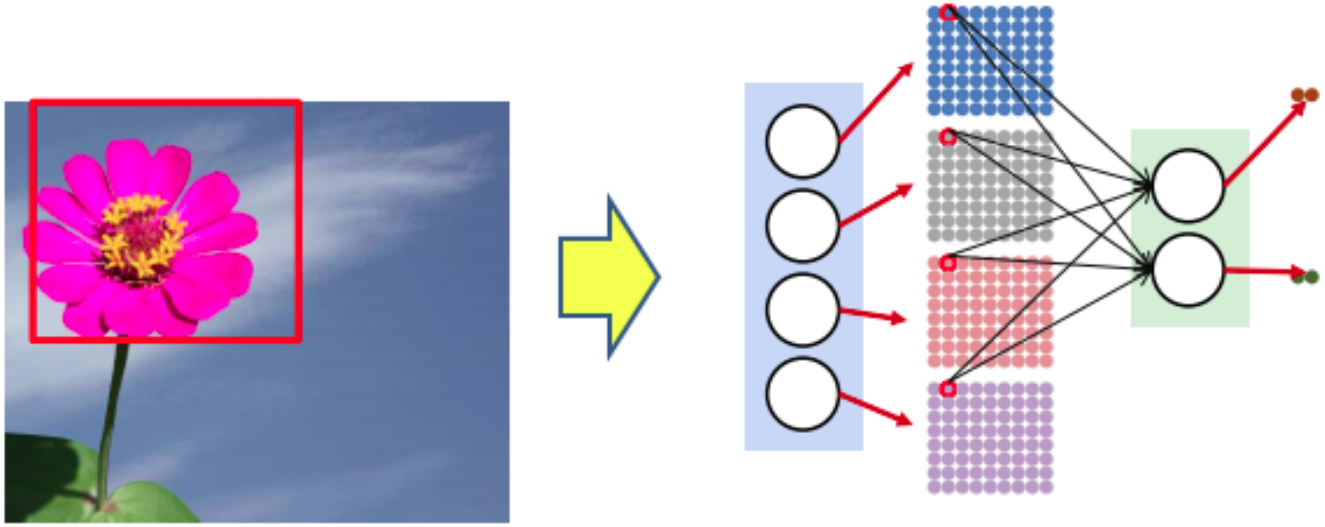
Deep Convolutional Neural Network XXI



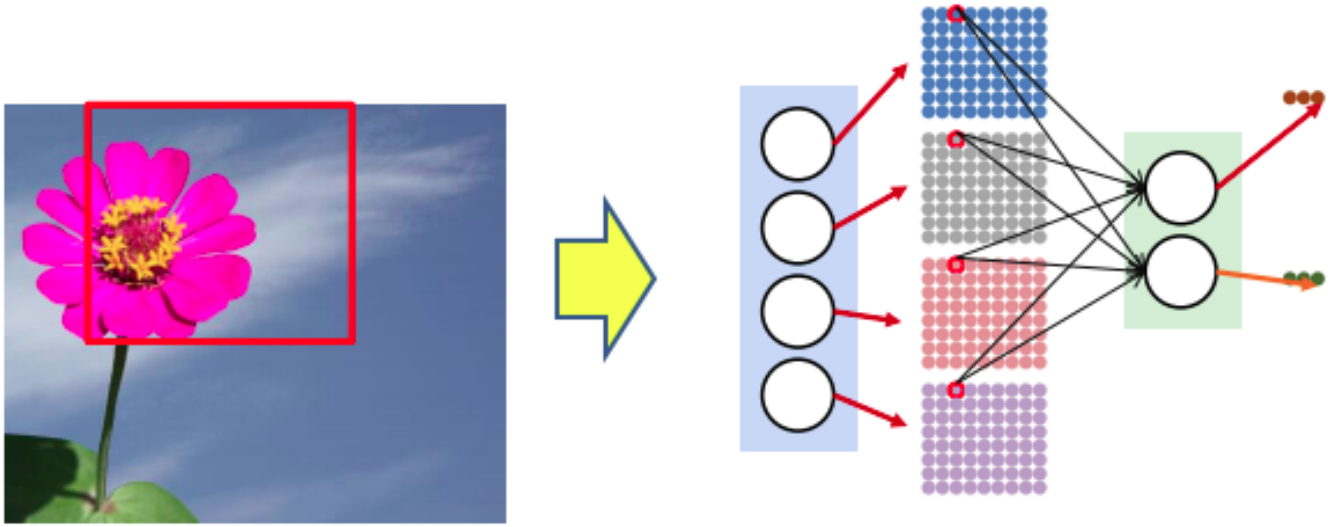
Deep Convolutional Neural Network XXII



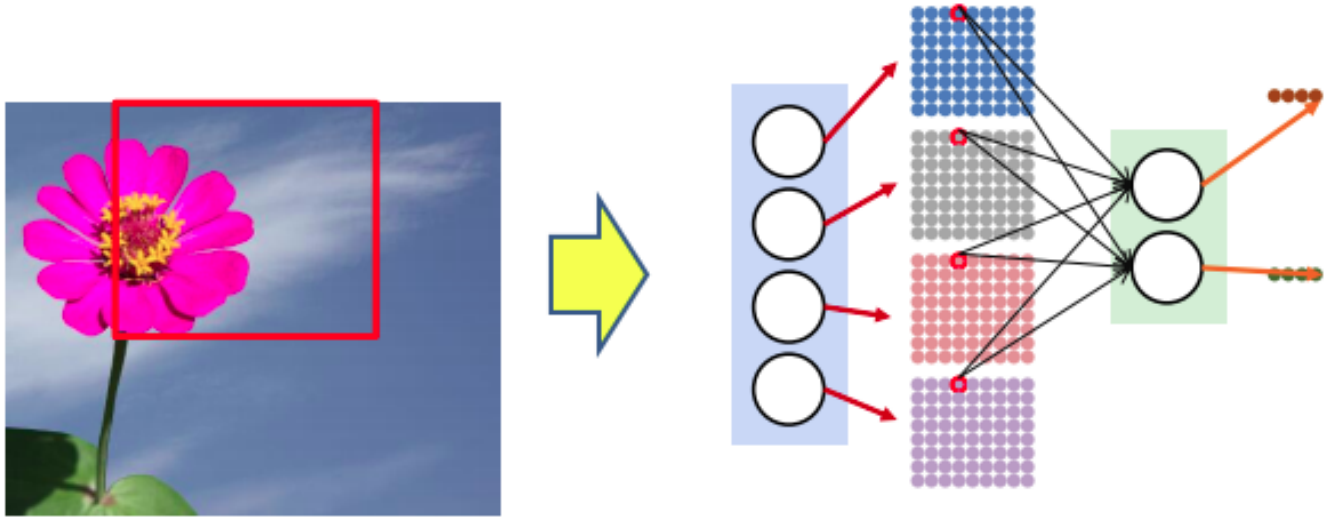
Deep Convolutional Neural Network XXIII



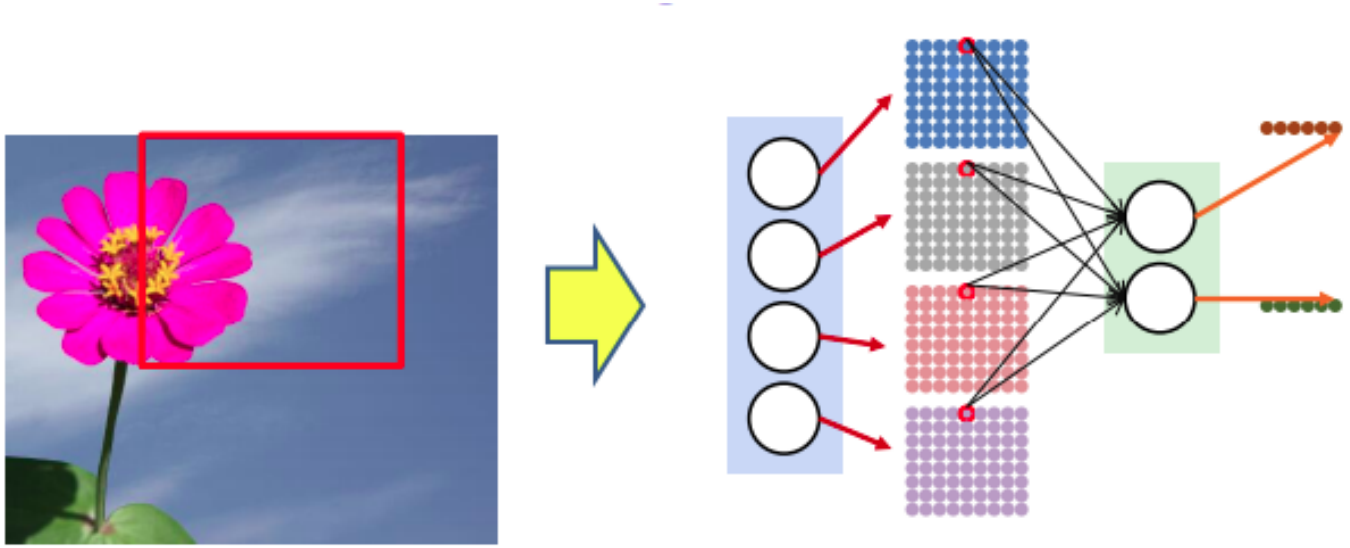
Deep Convolutional Neural Network XXIV



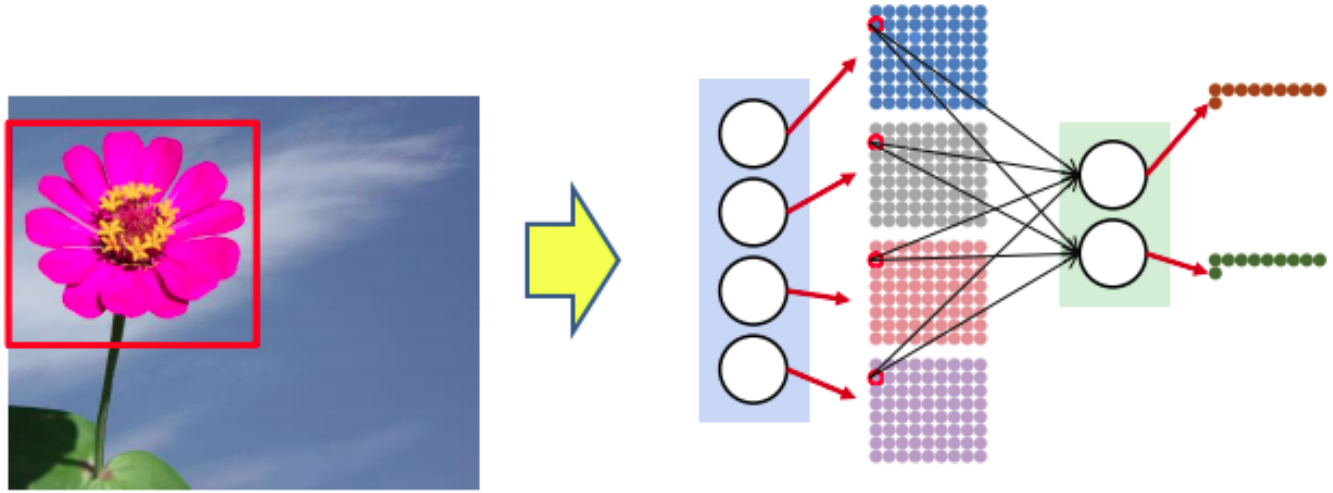
Deep Convolutional Neural Network XXV



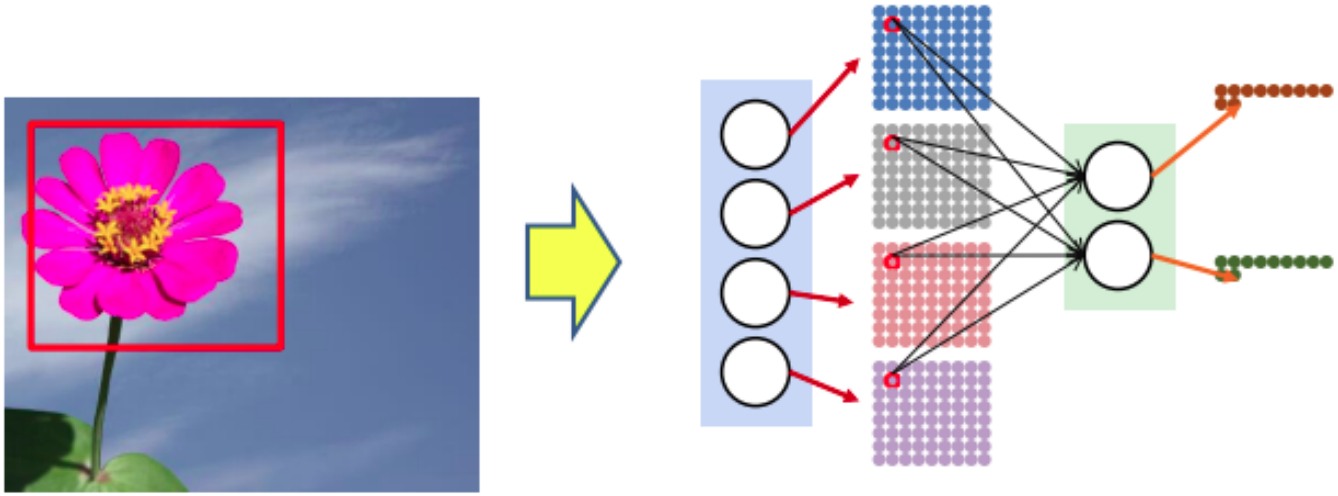
Deep Convolutional Neural Network XXVI



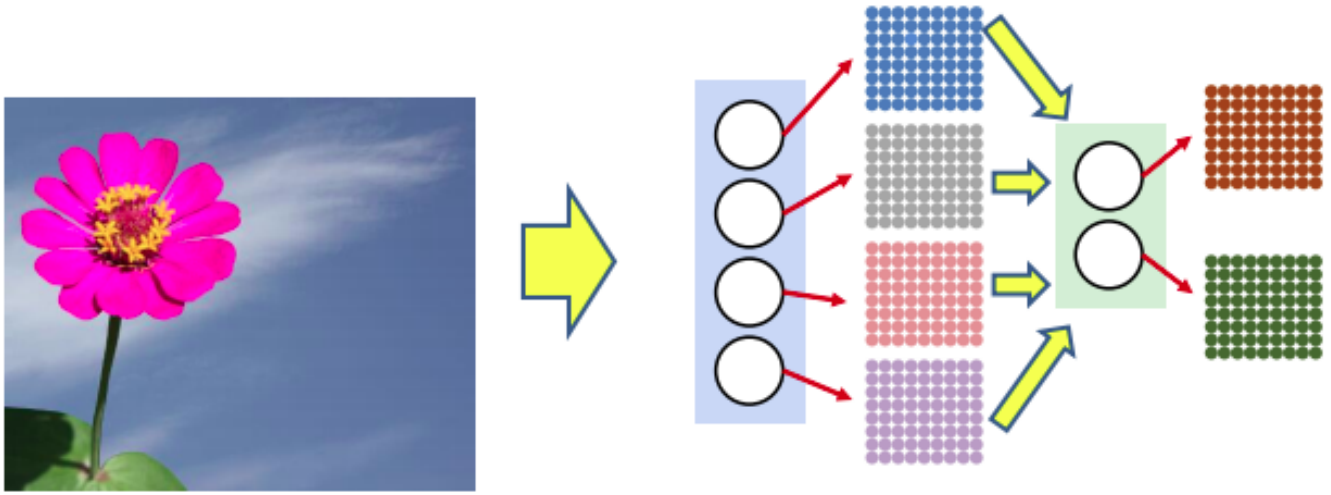
Deep Convolutional Neural Network XXVII



Deep Convolutional Neural Network XXVIII

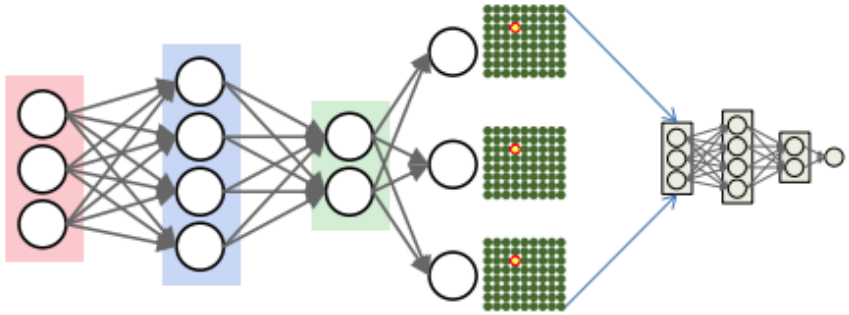


Deep Convolutional Neural Network XXIX



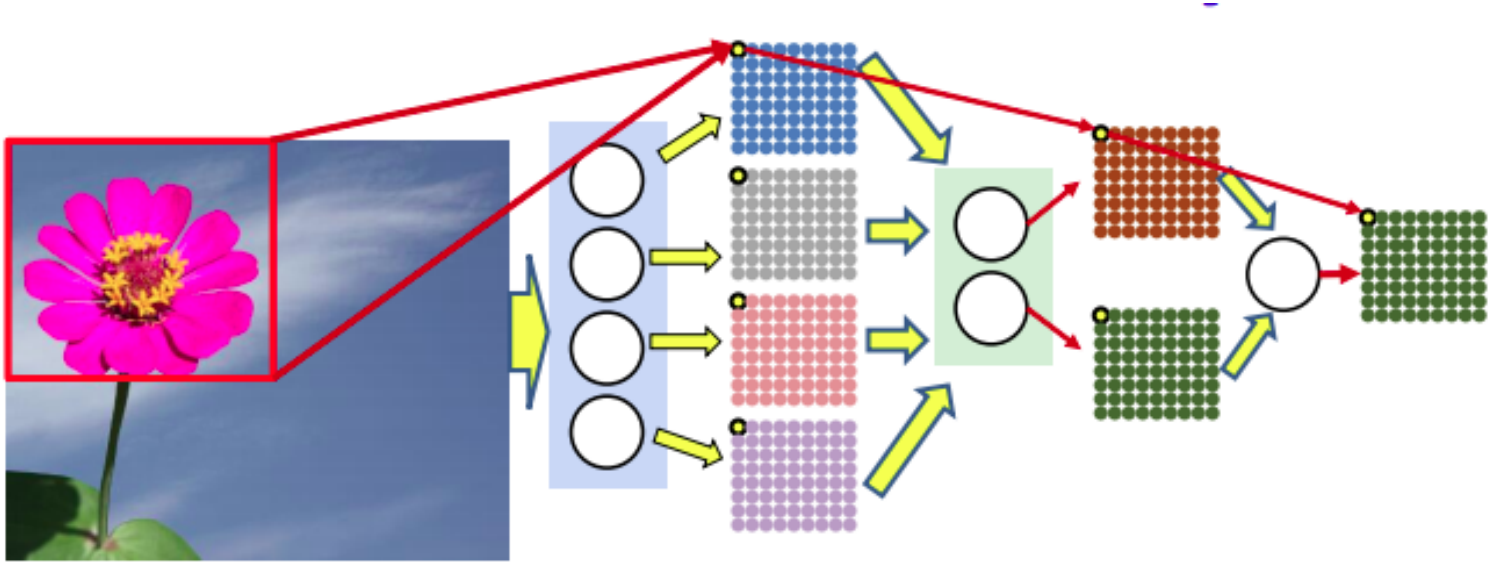
Deep Convolutional Neural Network XXX

- Is there a flower in the picture?
  - The output of the almost-last layer is also a grid/picture
  - The entire grid can be sent into a final neuron that performs a logical "OR" to detect a picture
    - Finds the max output from all positions
  - In more general sense: At each location, the net searches for a flower
    - The entire map of outputs is sent through a follow-up perceptron (or MLP) to determine if there really is a flower in the picture

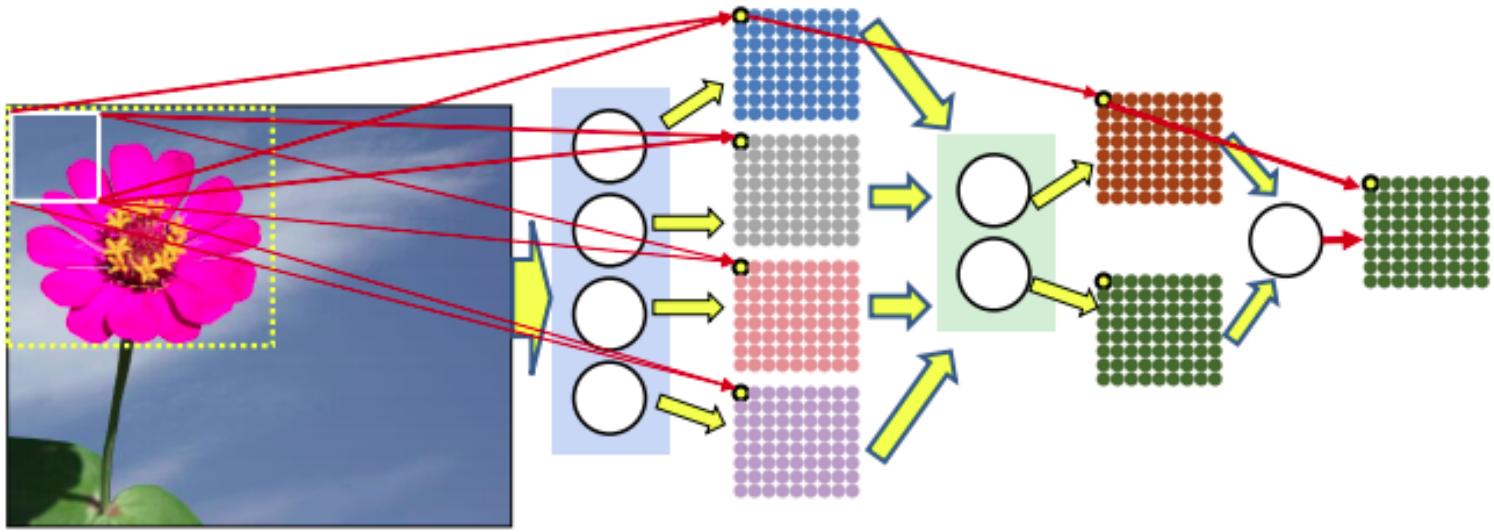


Deep Convolutional Neural Network XXXI

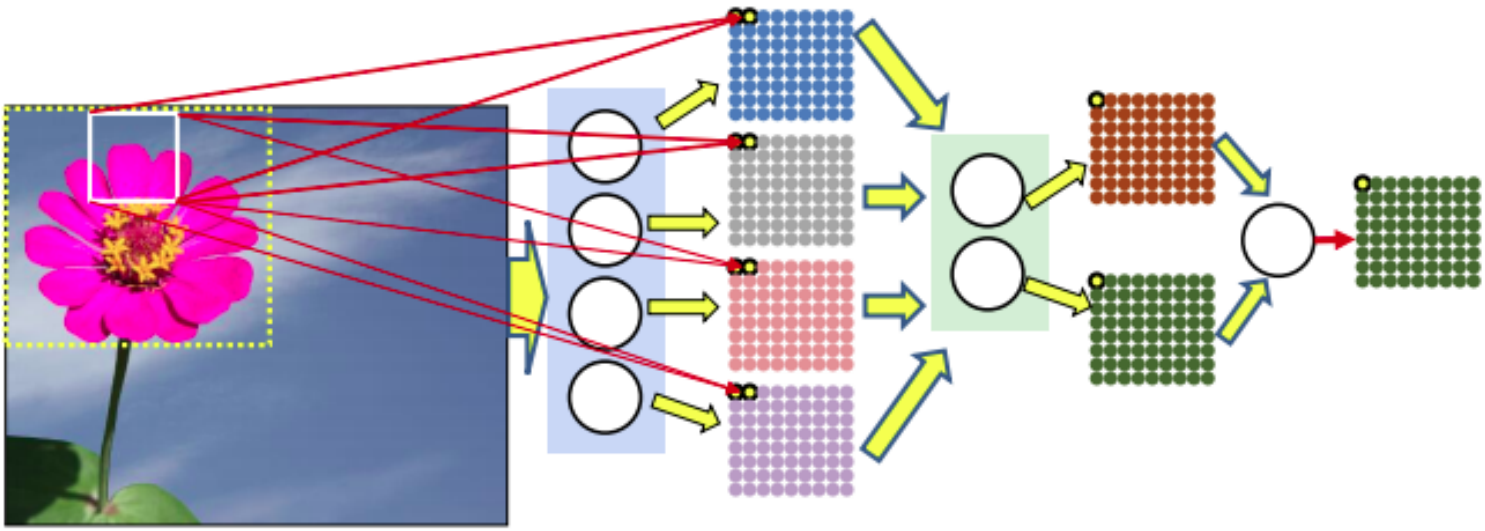
- Scan was actually distributed:



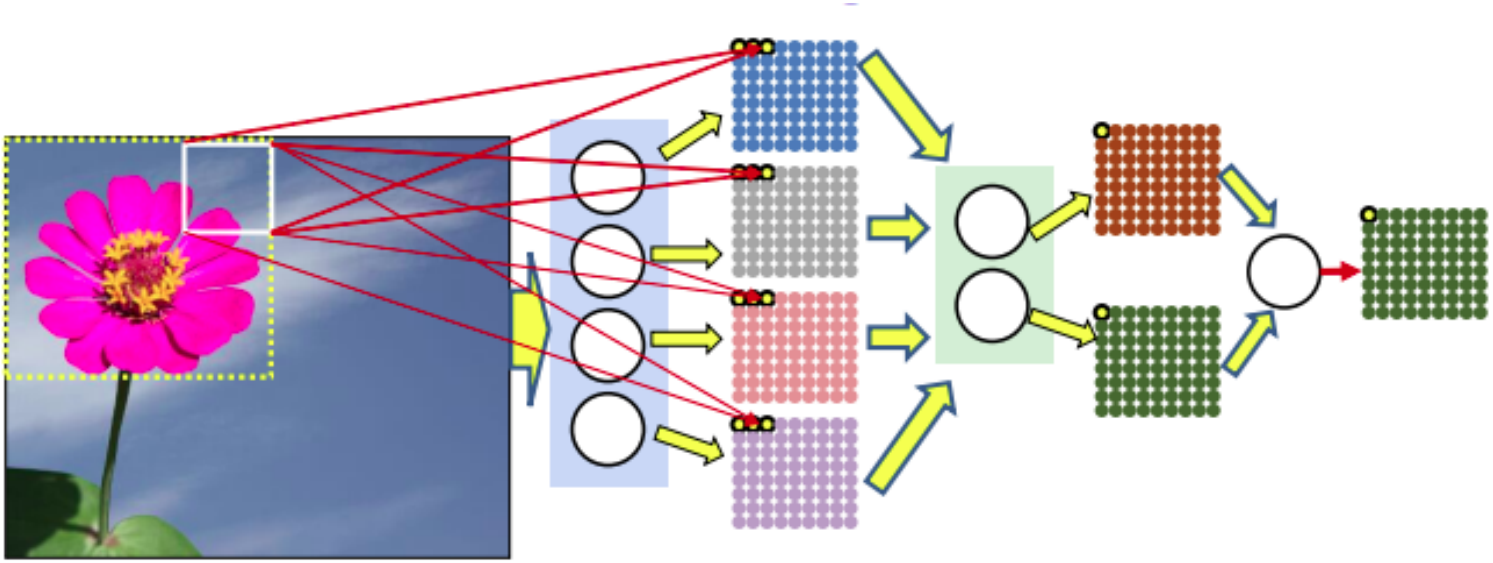
Deep Convolutional Neural Network XXXII



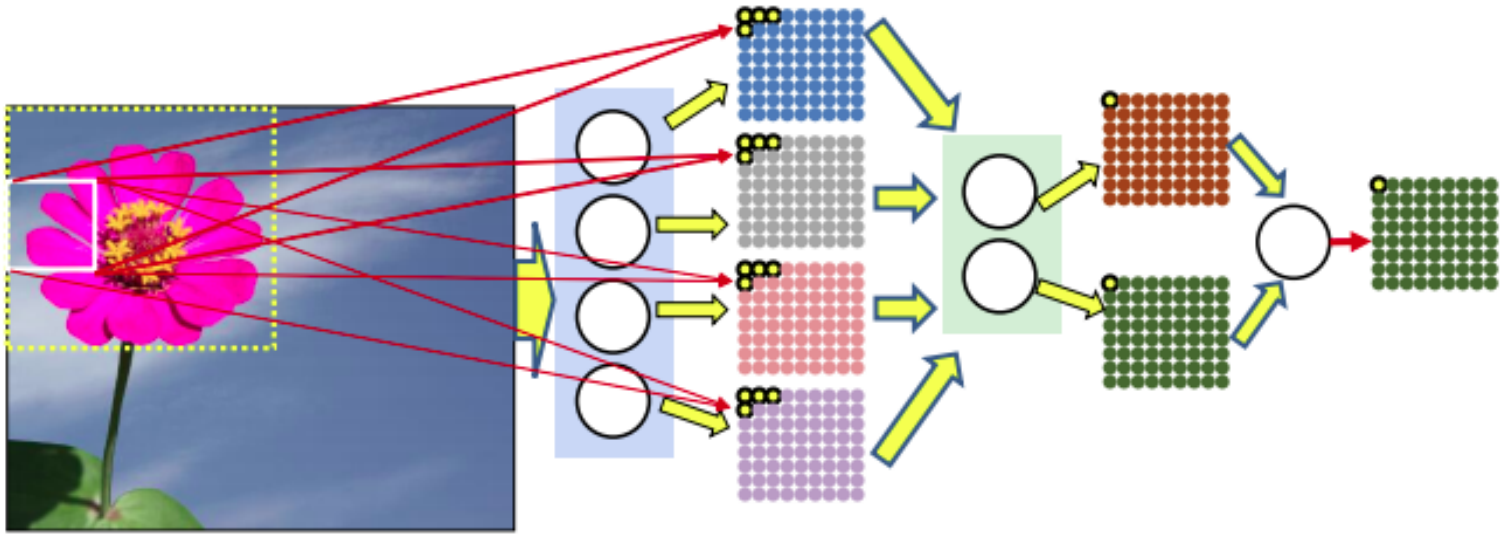
Deep Convolutional Neural Network XXXIII



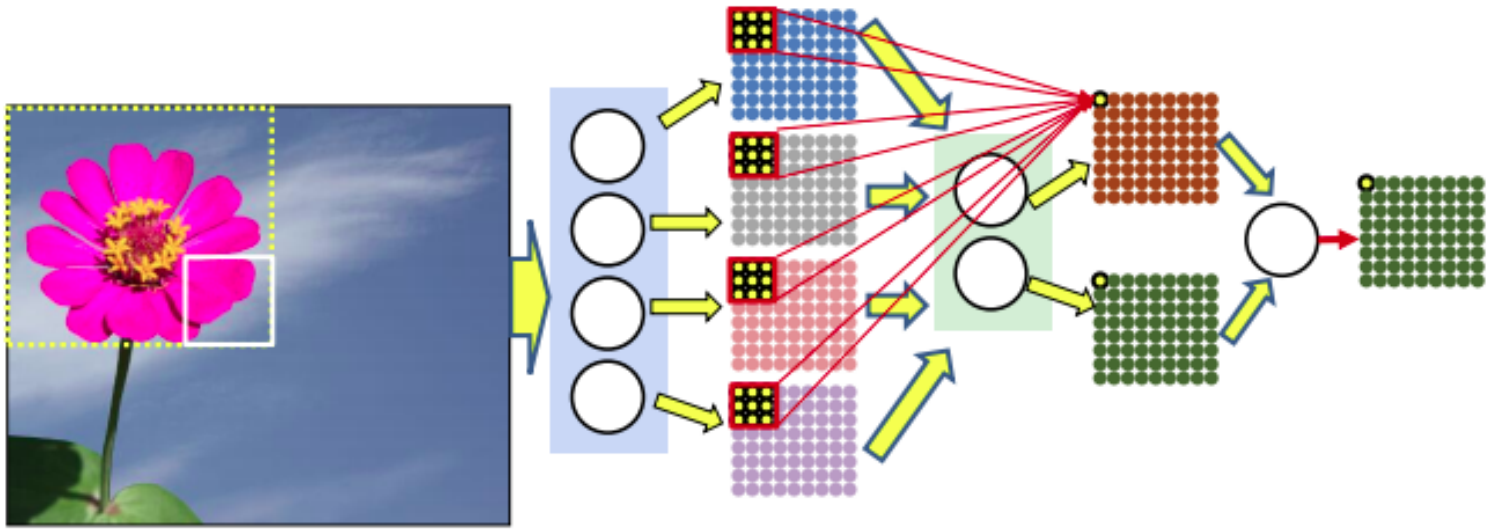
Deep Convolutional Neural Network XXXIV



Deep Convolutional Neural Network XXXV



Deep Convolutional Neural Network XXXVI



Deep Convolutional Neural Network XXXVII

Generic CNN architecture I

- Normalisation
  - Adjusting whitening (optional)
  - Subtractive methods e.g. average removal, high pass filtering
  - Divisive: local contrast normalisation, variance normalisation
- Filter banks:
  - Increase dimensionality
  - Projection on overcomplete basis
  - Edge detections
- Non-linearities
  - Sparsification
  - Typically Rectified Linear Unit (ReLU)
- Handling jitter: Pooling
  - Aggregating over a feature map
  - Max pooling
  - L_p-norm pooling
  - ...

What CNNs are good for? I

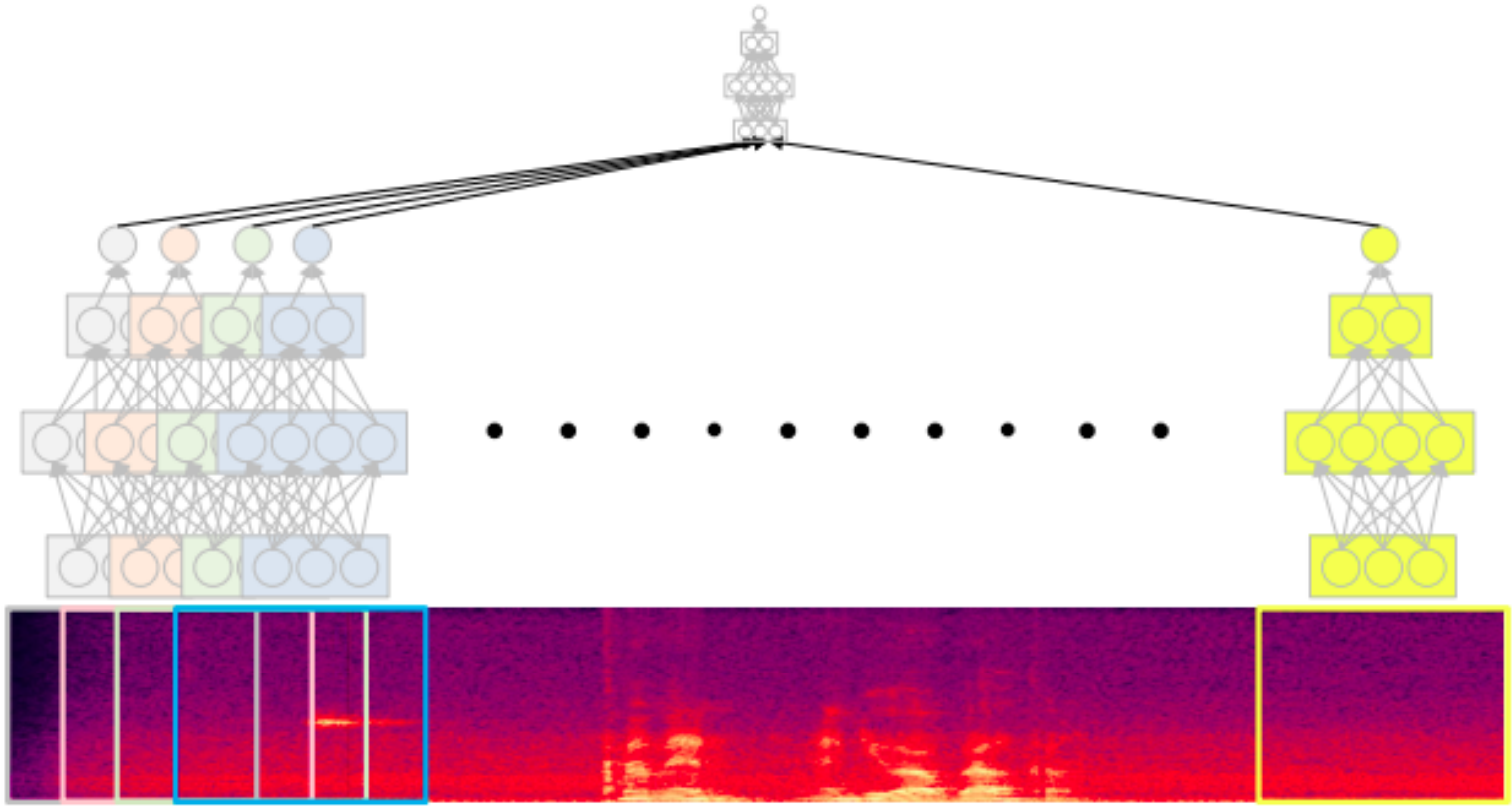
- Locality:
  - There is a strong local correlation between values.
  - Nearby pixels same color; similarity decreases as they move distant
  - The local correlations can help us detect local features (CNNs do this).
  - If we feed the CNN with permuted pixels, it will not perform well at recognizing the input images, while FC will not be affected.
  - The local correlation justifies local connections.
- Stationarity:
  - The features are essential and can appear anywhere on the image (shared weight and pooling)
  - Moreover, statistical signals are uniformly distributed, which means we need to repeat the feature detection for every location on the input image.

What CNNs are good for? II

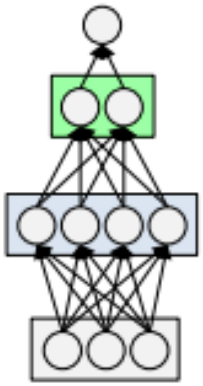
- Compositionality:
  - The natural images are compositional, meaning the features compose an image in a hierarhical manner.
  - Justifies use of multiple layers
- Success in various application domains: videos, images, texts, and speech recognition.

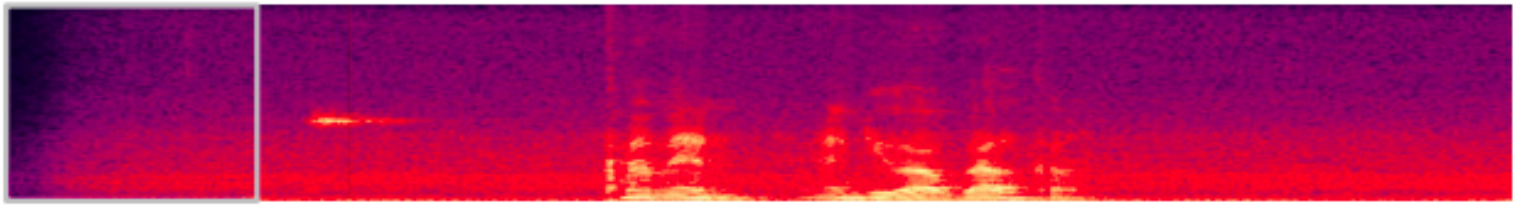
A simple case-study: Speech Recognition I

The spectrographic time-frequency components are fed as input to the 1D CNN.

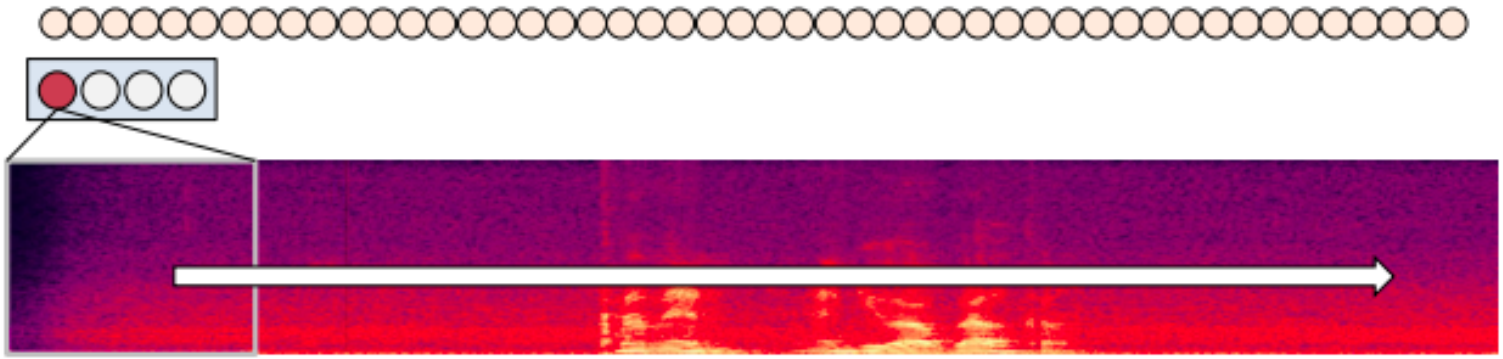


A simple case-study: Speech Recognition II

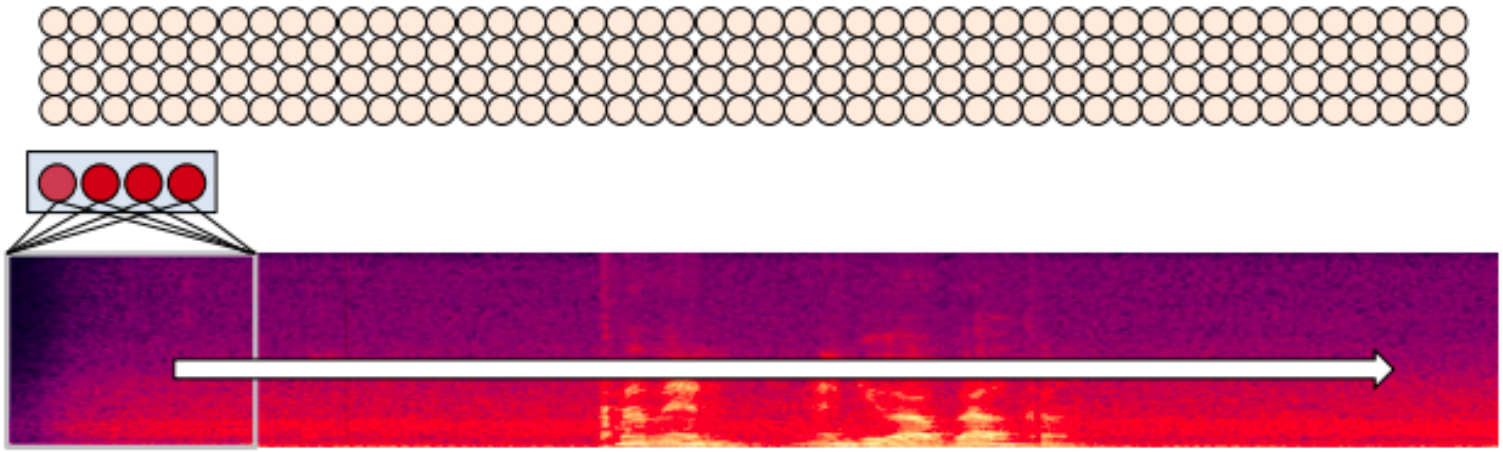




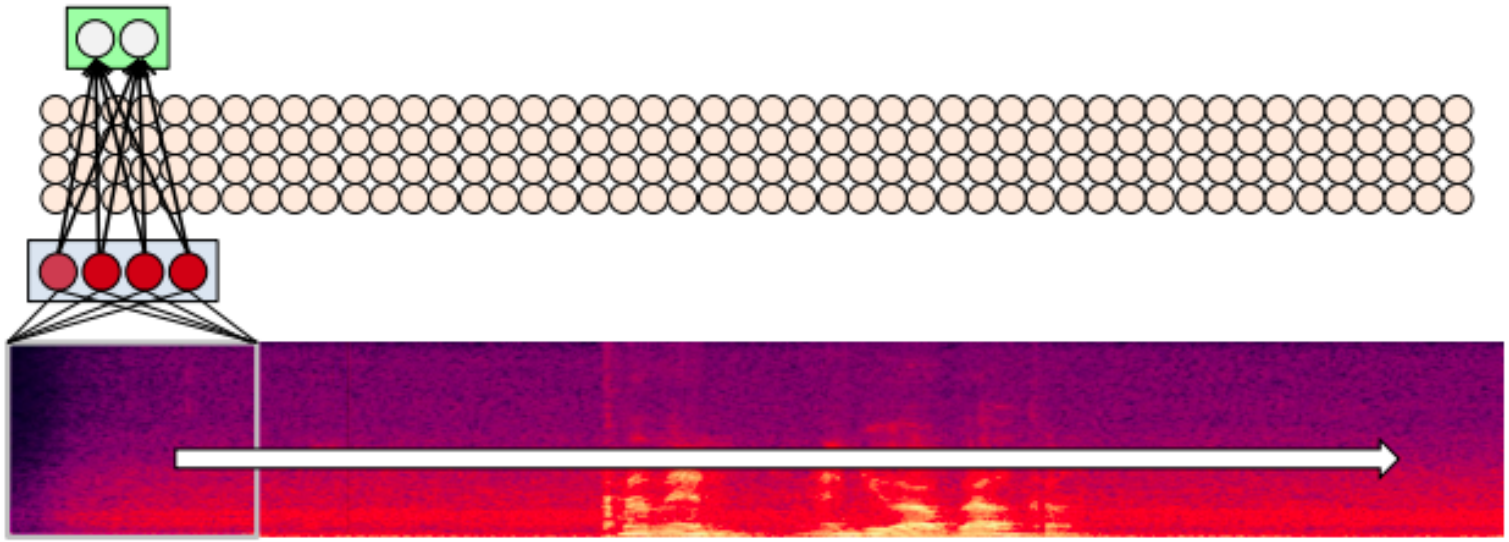
A simple case-study: Speech Recognition III



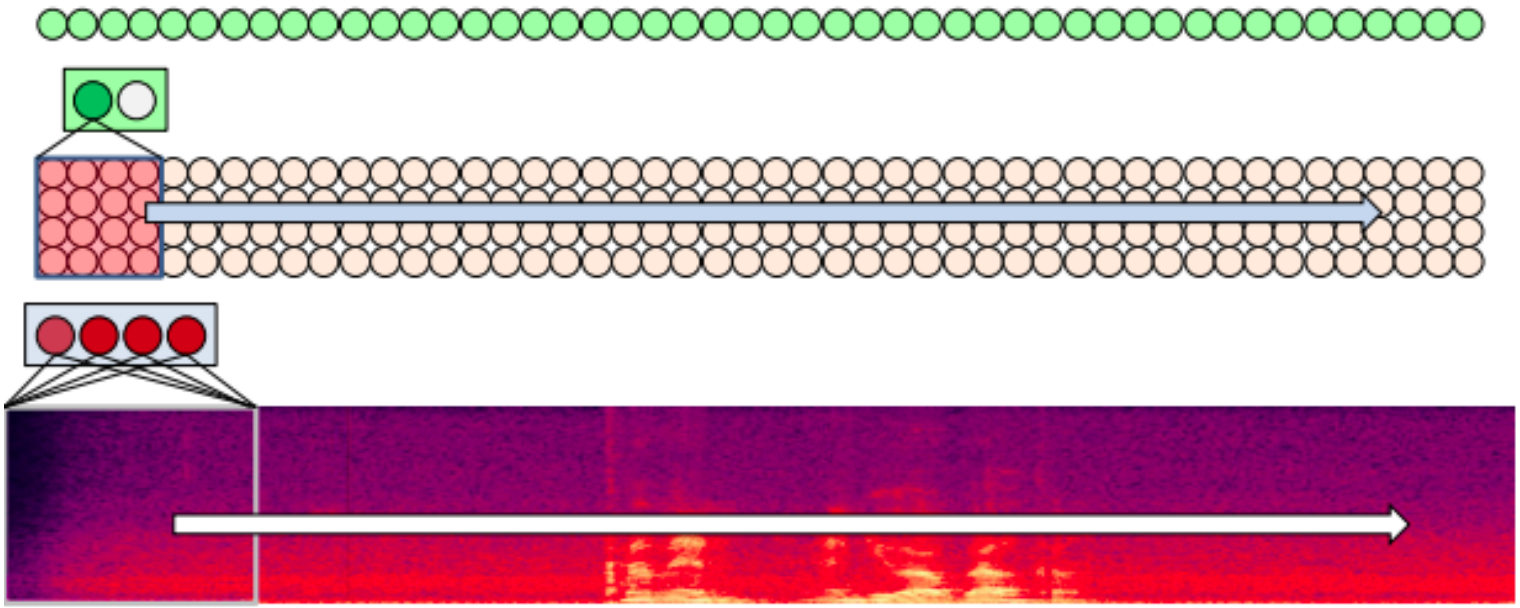
A simple case-study: Speech Recognition IV



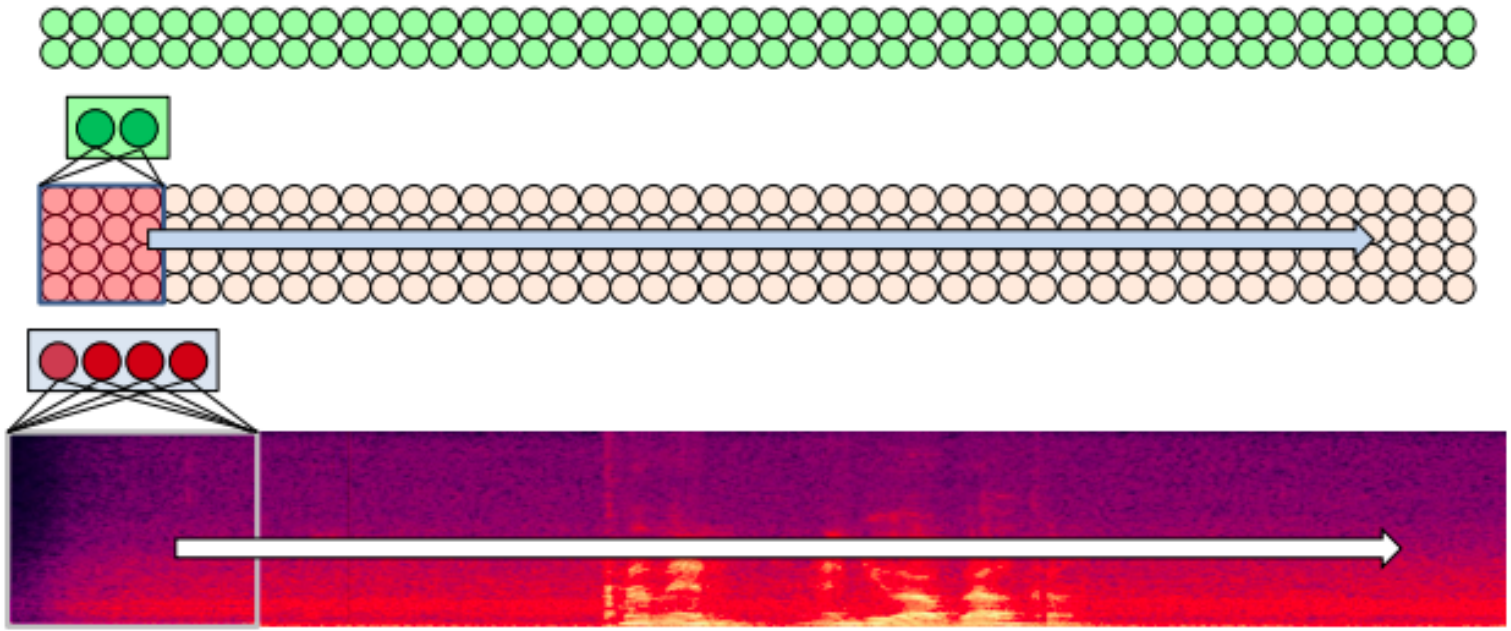
A simple case-study: Speech Recognition V



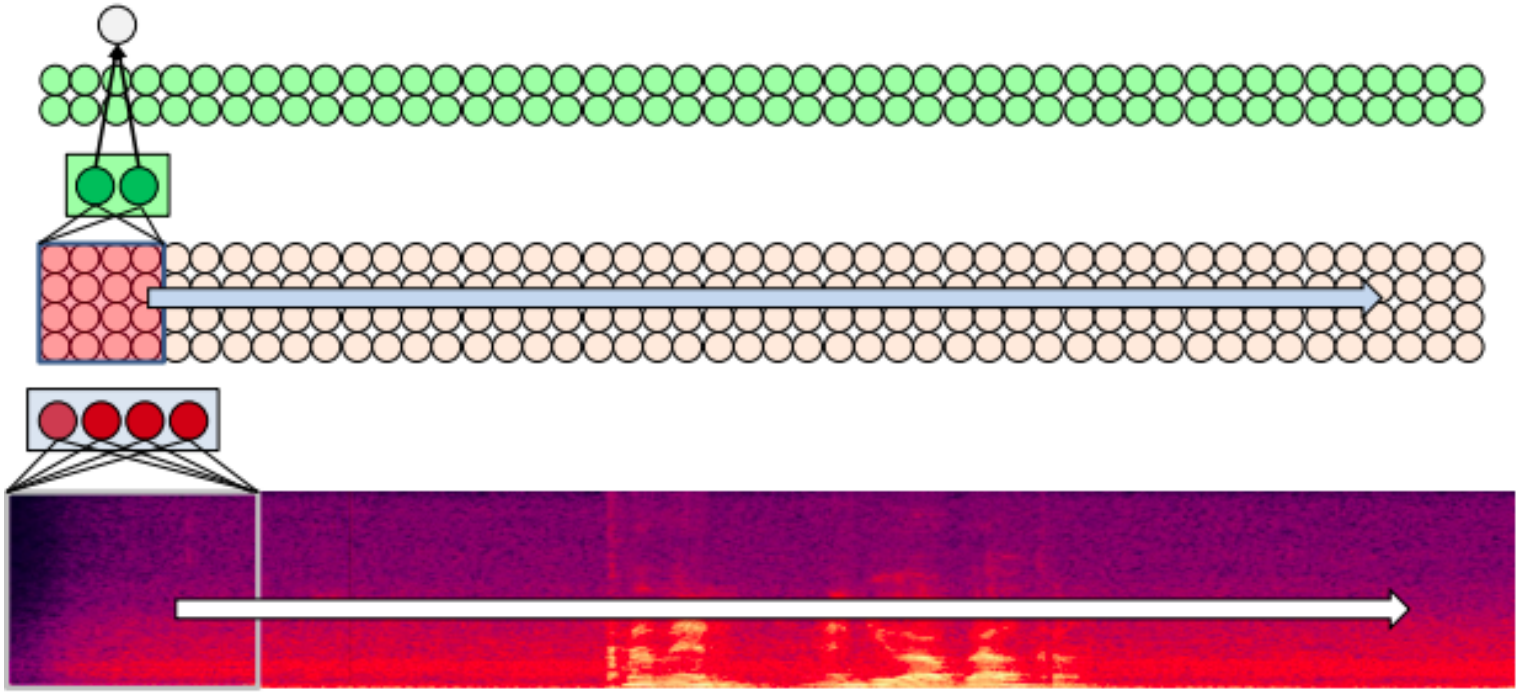
A simple case-study: Speech Recognition VI



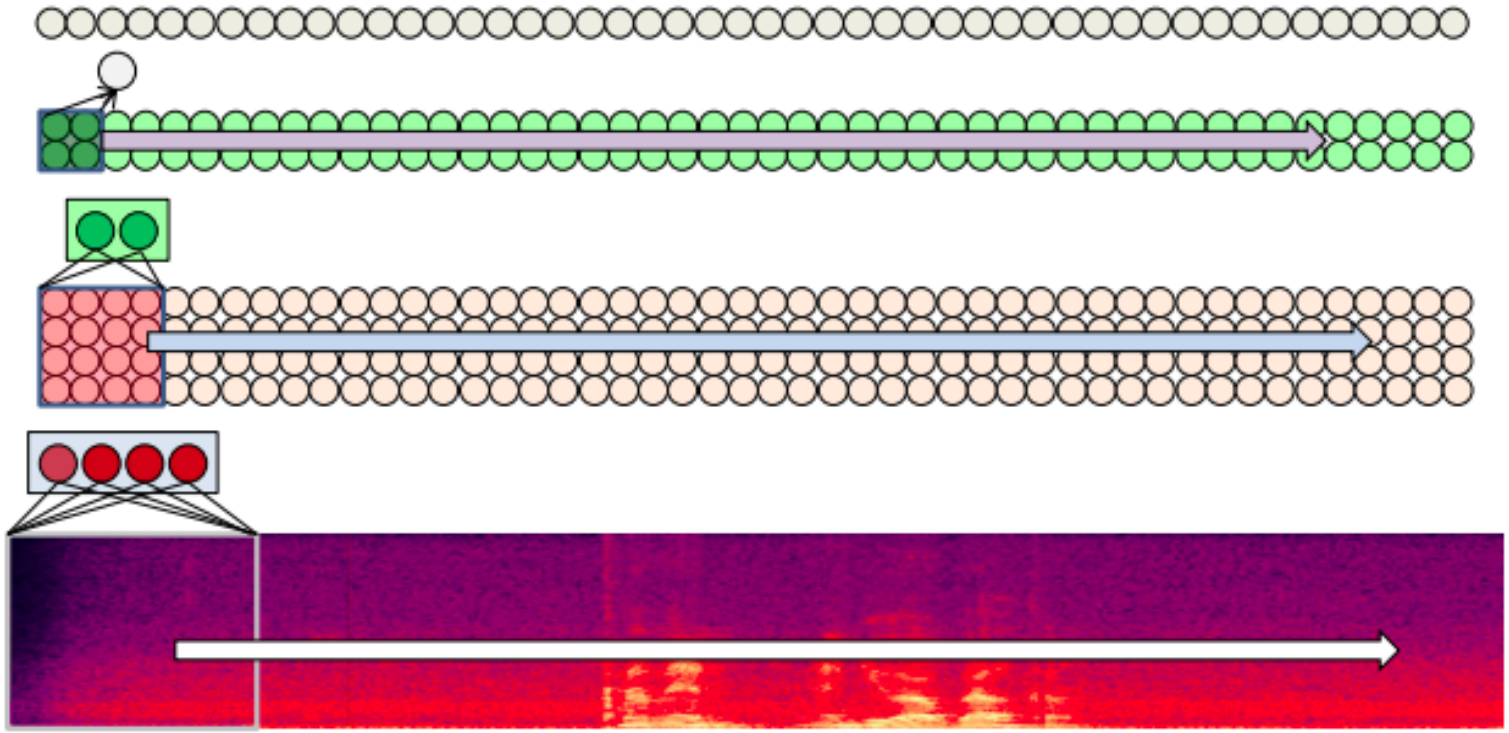
A simple case-study: Speech Recognition VII



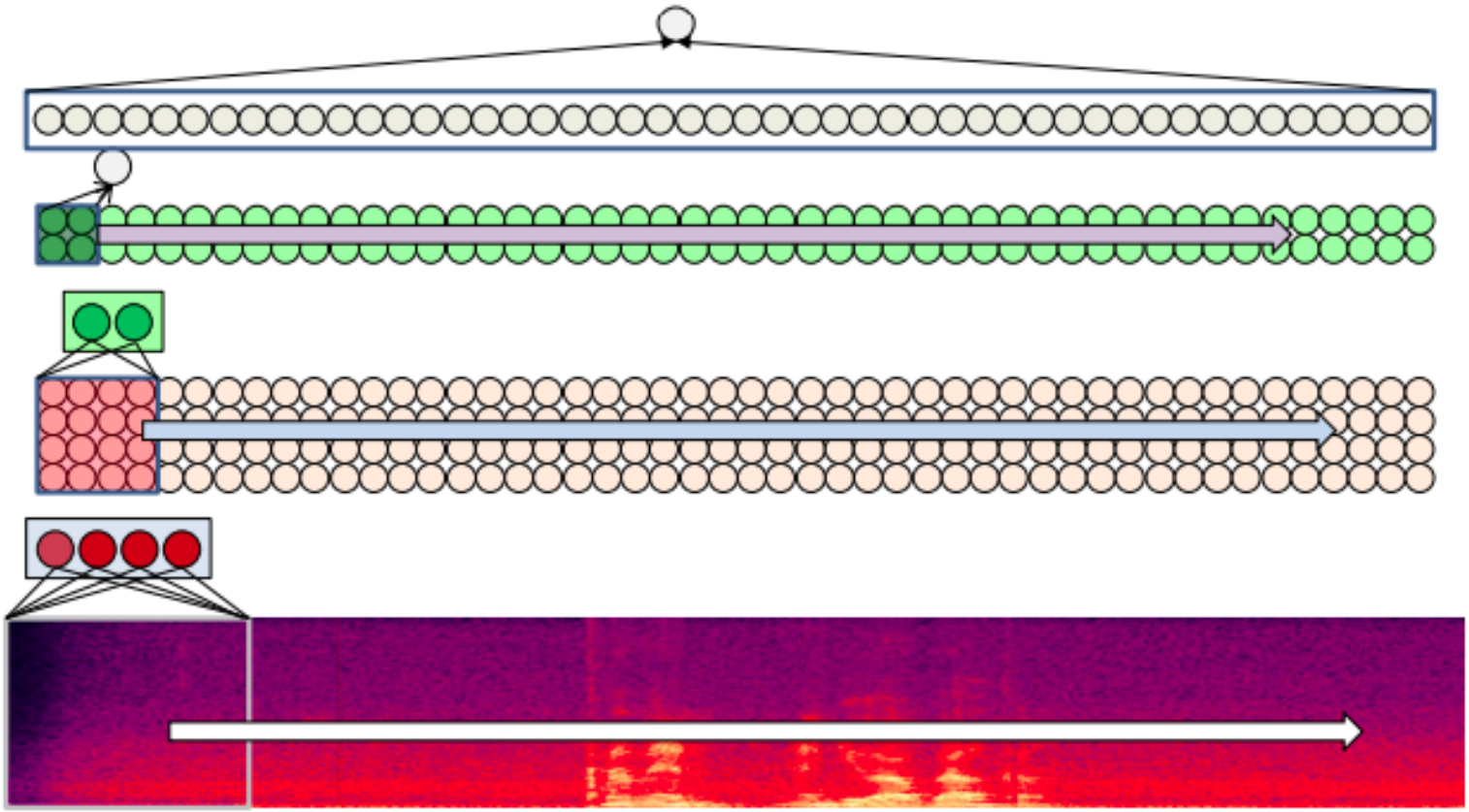
A simple case-study: Speech Recognition VIII



A simple case-study: Speech Recognition IX



A simple case-study: Speech Recognition X



A simple case-study: Speech Recognition XI

E.g. A final perceptron (or MLP) to aggregate evidence: "Does this recording have the target word?"

Homework

- Understand the basic blocks of initial CNN architectures for digit recognition: LeNet5 (`http://yann.lecun.com/exdb/lenet/`)
- Understand VGGNet architecture
- Understand ResNet architecture: mainly, the roles of residual connections