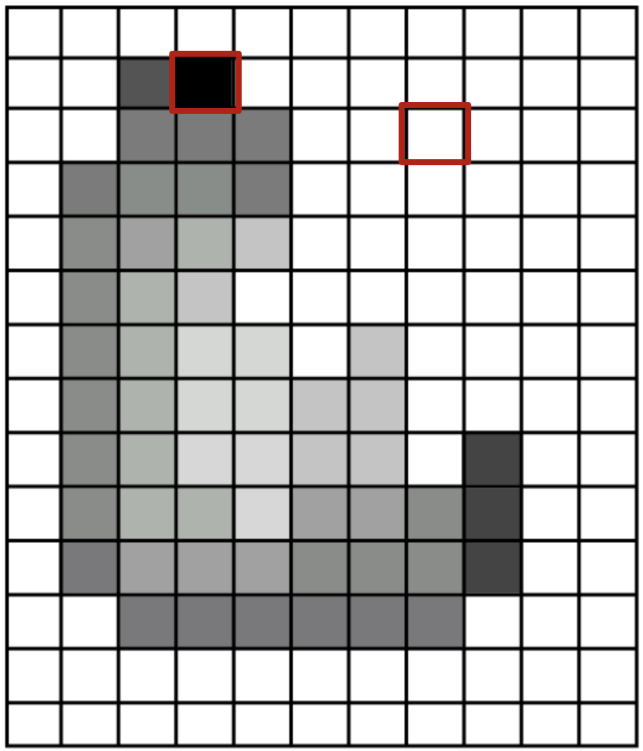
Topic 3: Deep Learning in Computer Vision

Image Representation: Matrices

- **Matrix**: Grayscale images



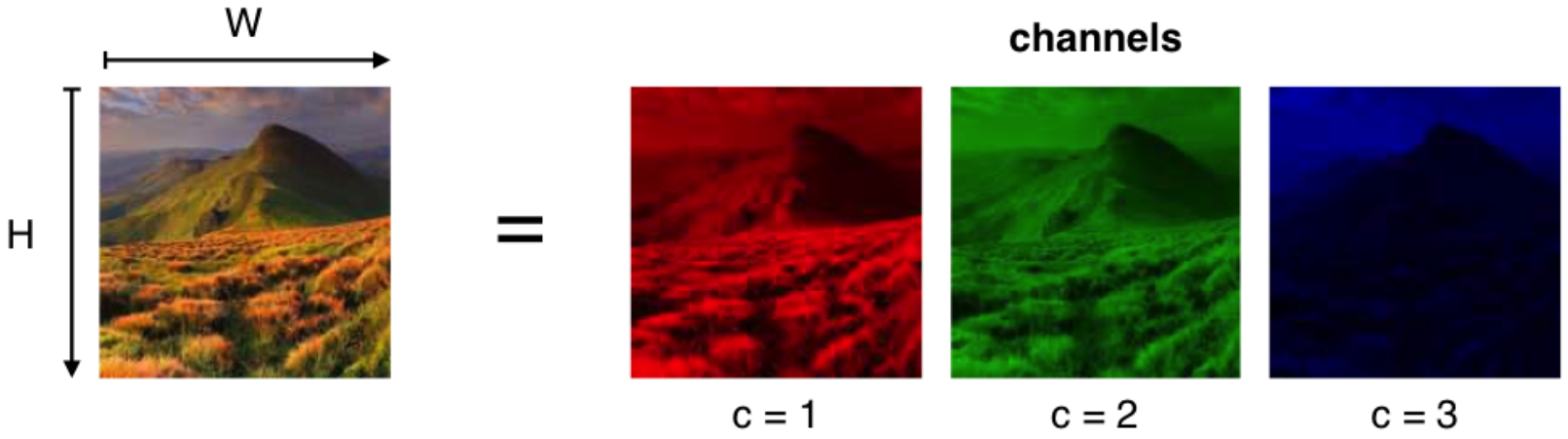
=

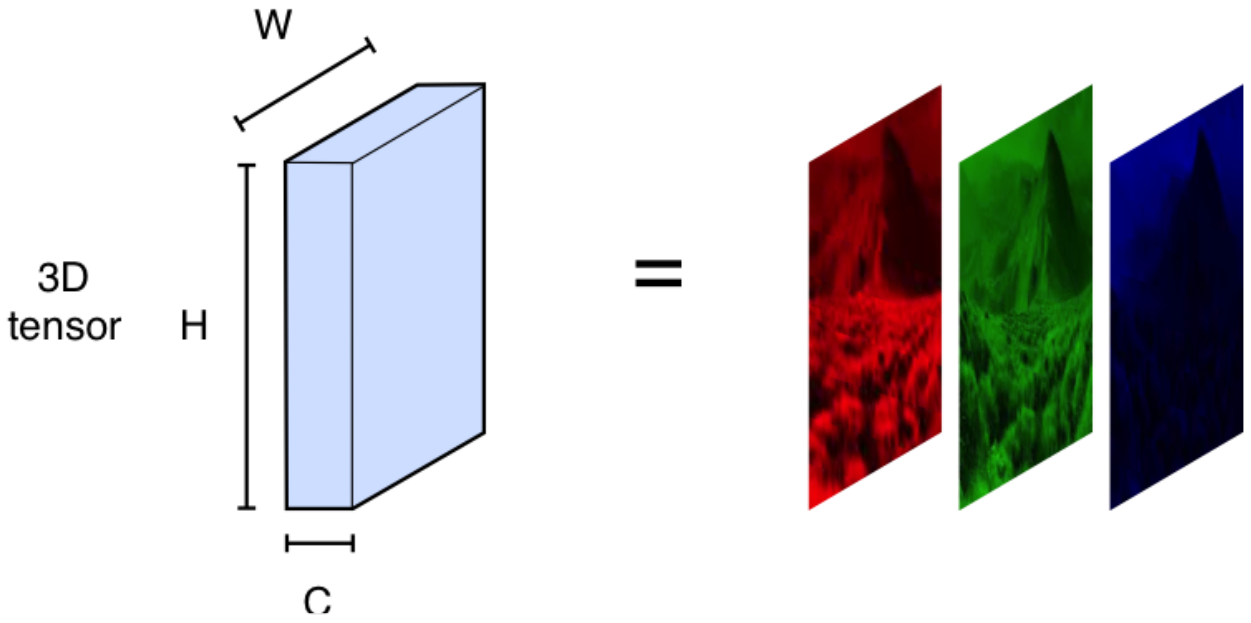
255	255	255	255	255	255	255	255	255	255	255
255	255	20	0	255	255	255	255	255	255	255
255	255	75	75	255	255	255	255	255	255	255
255	75	95	95	75	255	255	255	255	255	255
255	96	127	145	175	255	255	255	255	255	255
255	127	145	175	175	175	255	255	255	255	255
255	127	145	200	200	175	175	95	255	255	255
255	127	145	200	200	175	175	95	47	255	255
255	127	145	145	175	127	127	95	47	255	255
255	74	127	127	127	95	95	95	47	255	255
255	255	74	74	74	74	74	74	255	255	255
255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255

0 = black; 255 = white

Image Representation: Tensors

- **Tensor**: Most common form of representation



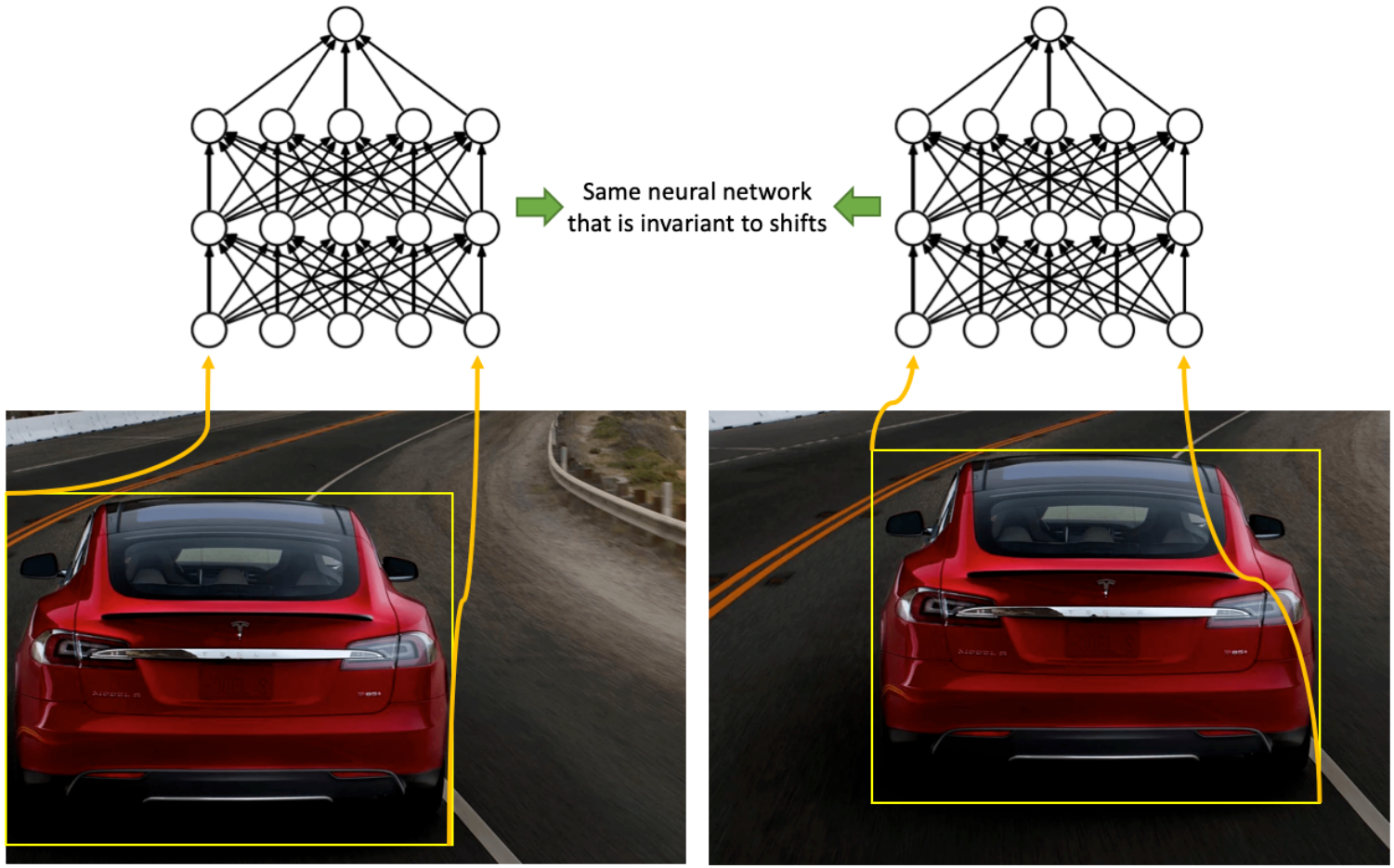


Computer Vision: An Collection of Exciting/Challenging Problems

- Object Detection
- Object Localization
- Object Tracking
- Segmentation
- Alignment

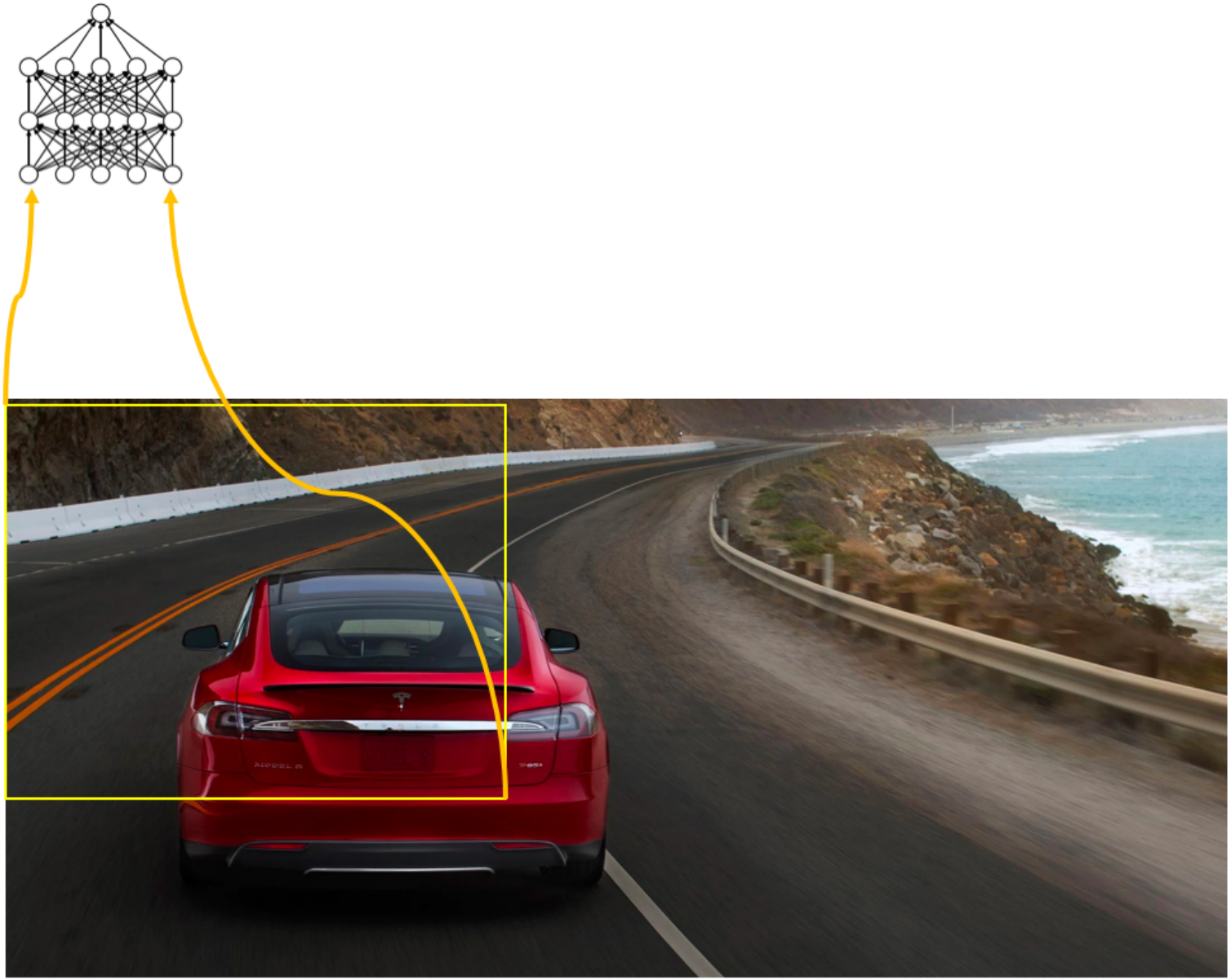
Object Detection with NNs: Need for Shift Invariance

- Will a neural network trained to detect a car in the left corner of the image, detect a car in a different location?
  - Conventional neural networks are sensitive to *shifts*.
  - Need one that detects the presence of car regardless of its location.



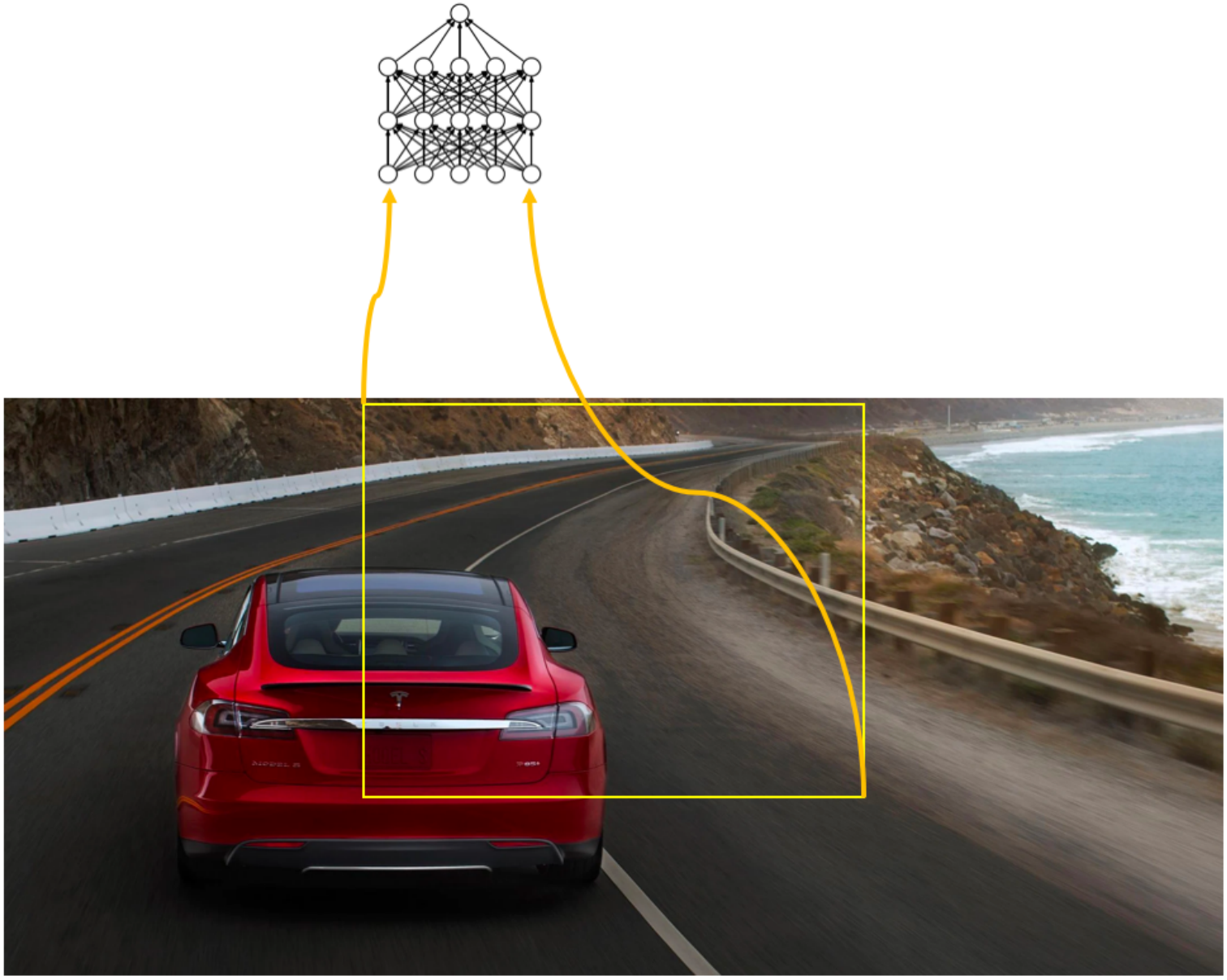
Solution: Scan the Image

- Scan for the target object at each location using a patch (kernel) of size $K \times K$.



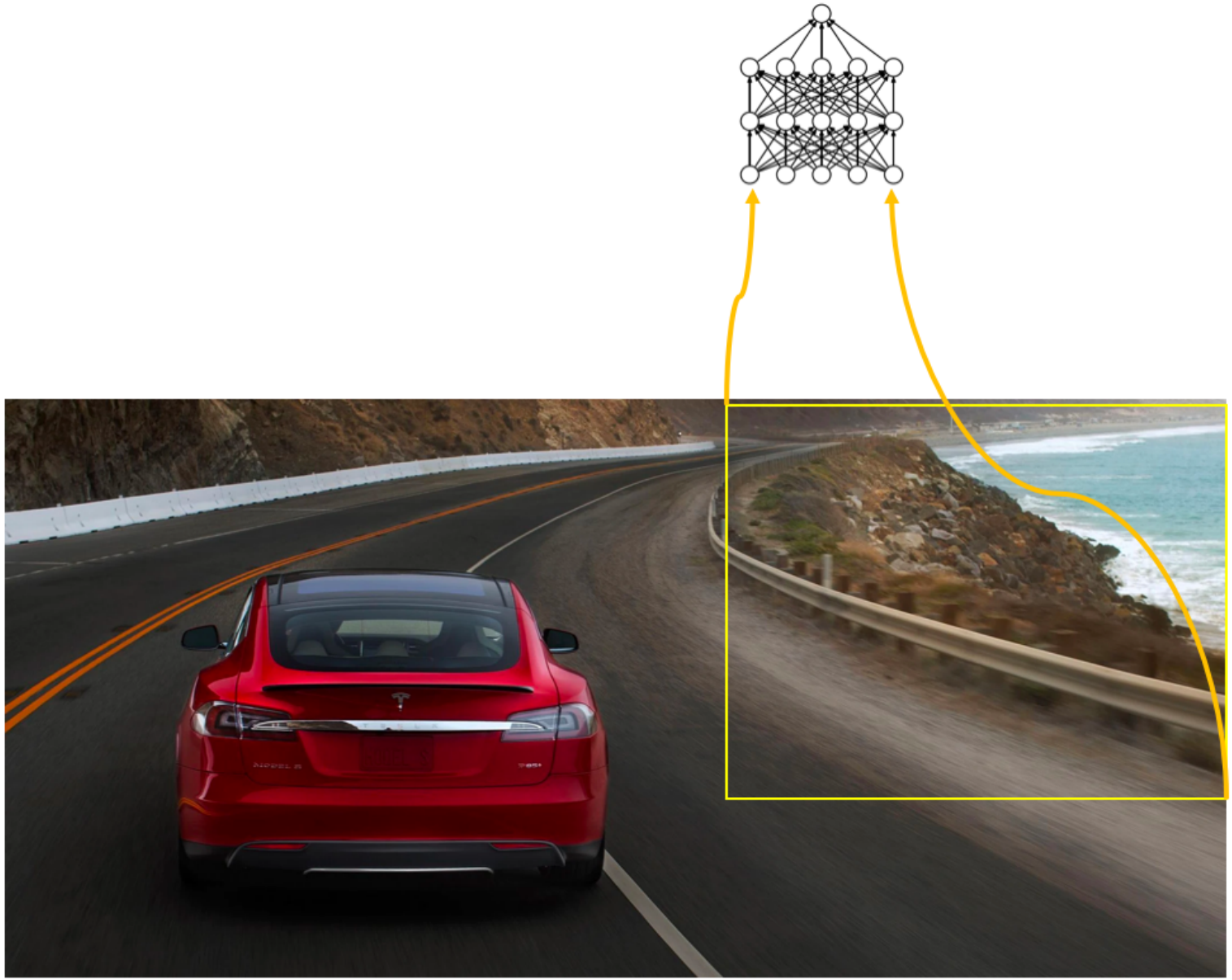
Solution: Scan the Image

- Scan for the target object at each location using a patch (kernel) of size $K \times K$.



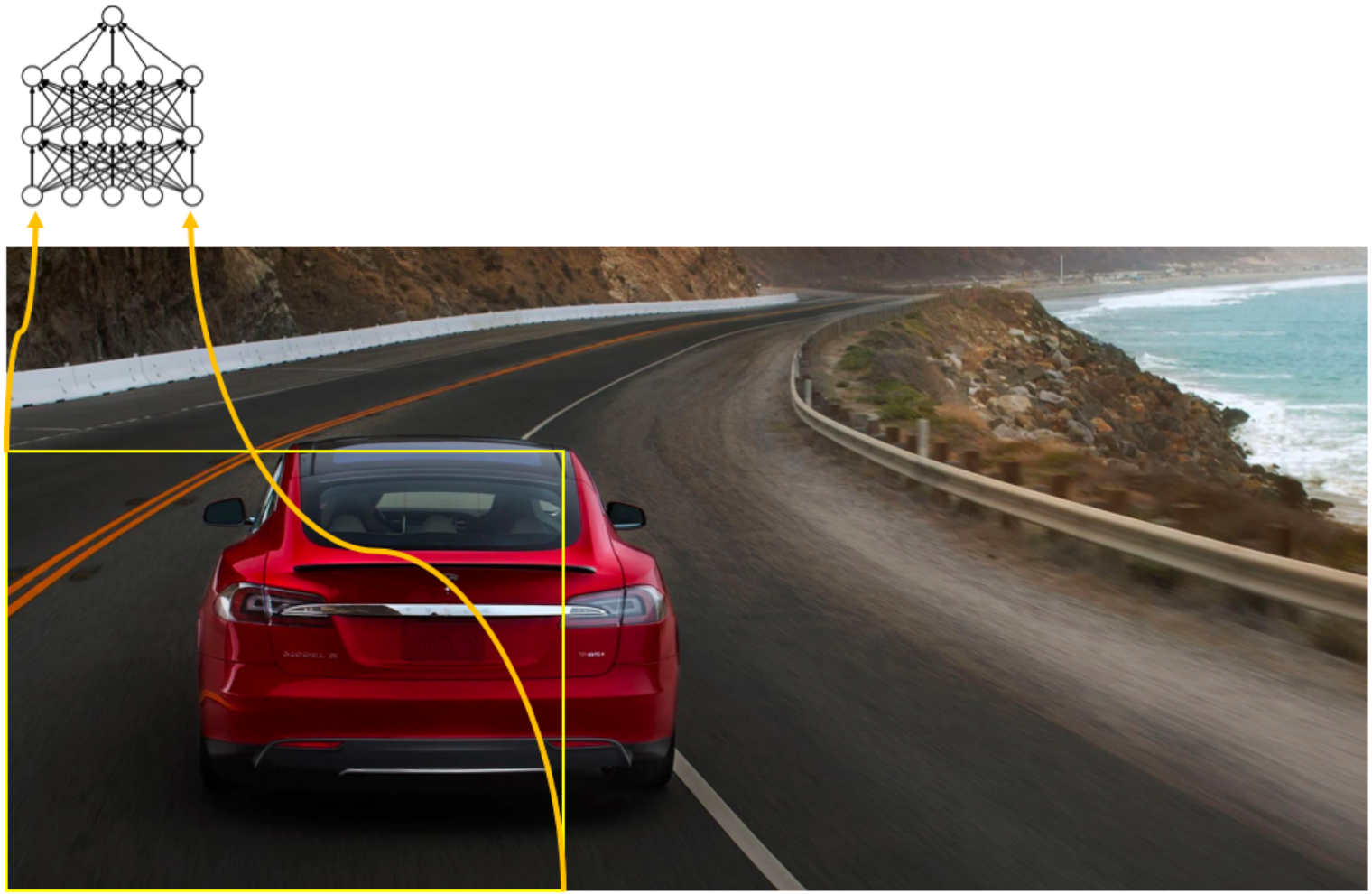
Solution: Scan the Image

- Scan for the target object at each location using a patch (kernel) of size $K \times K$.



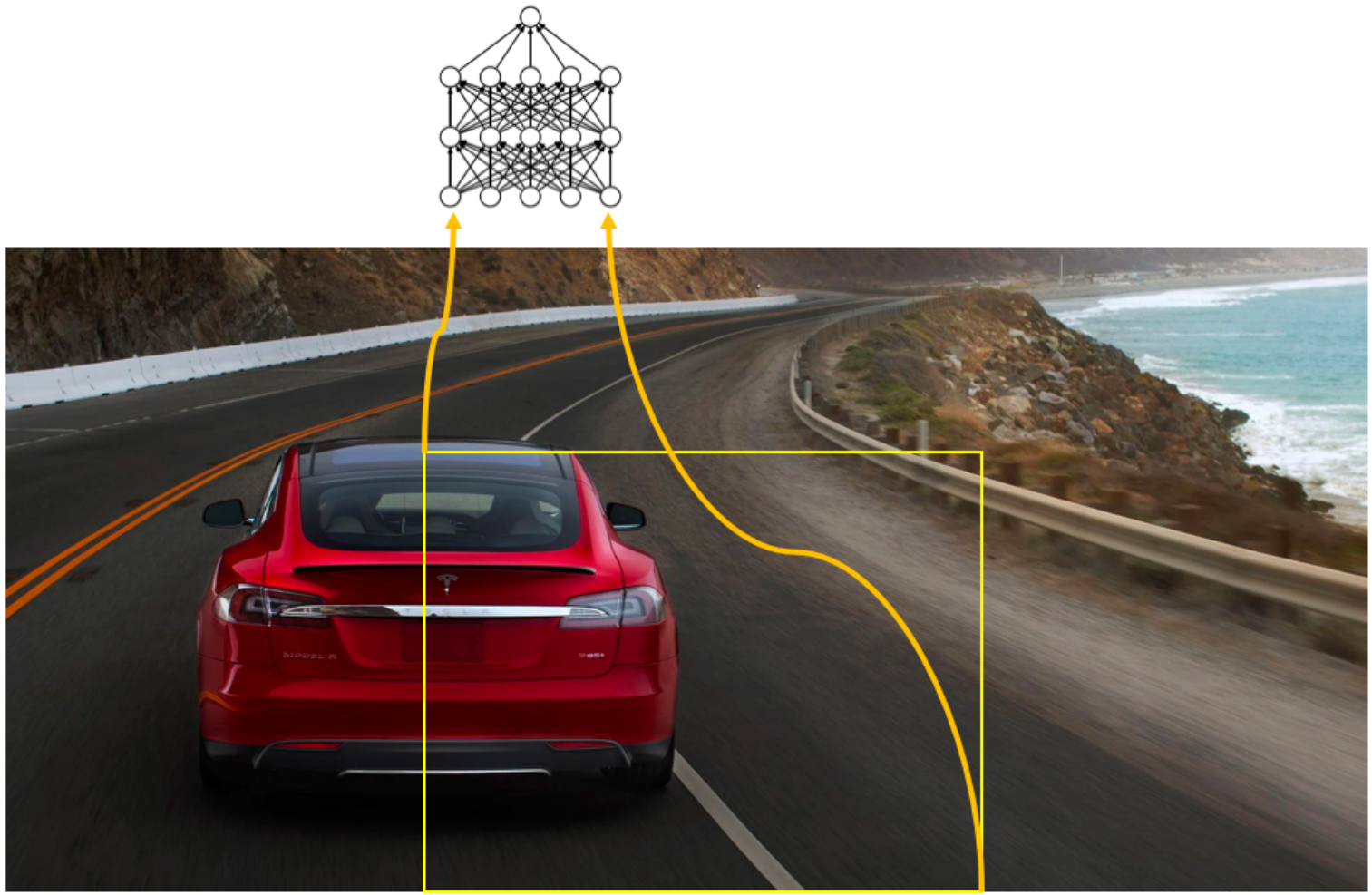
Solution: Scan the Image

- Scan for the target object at each location using a patch (kernel) of size $K \times K$.



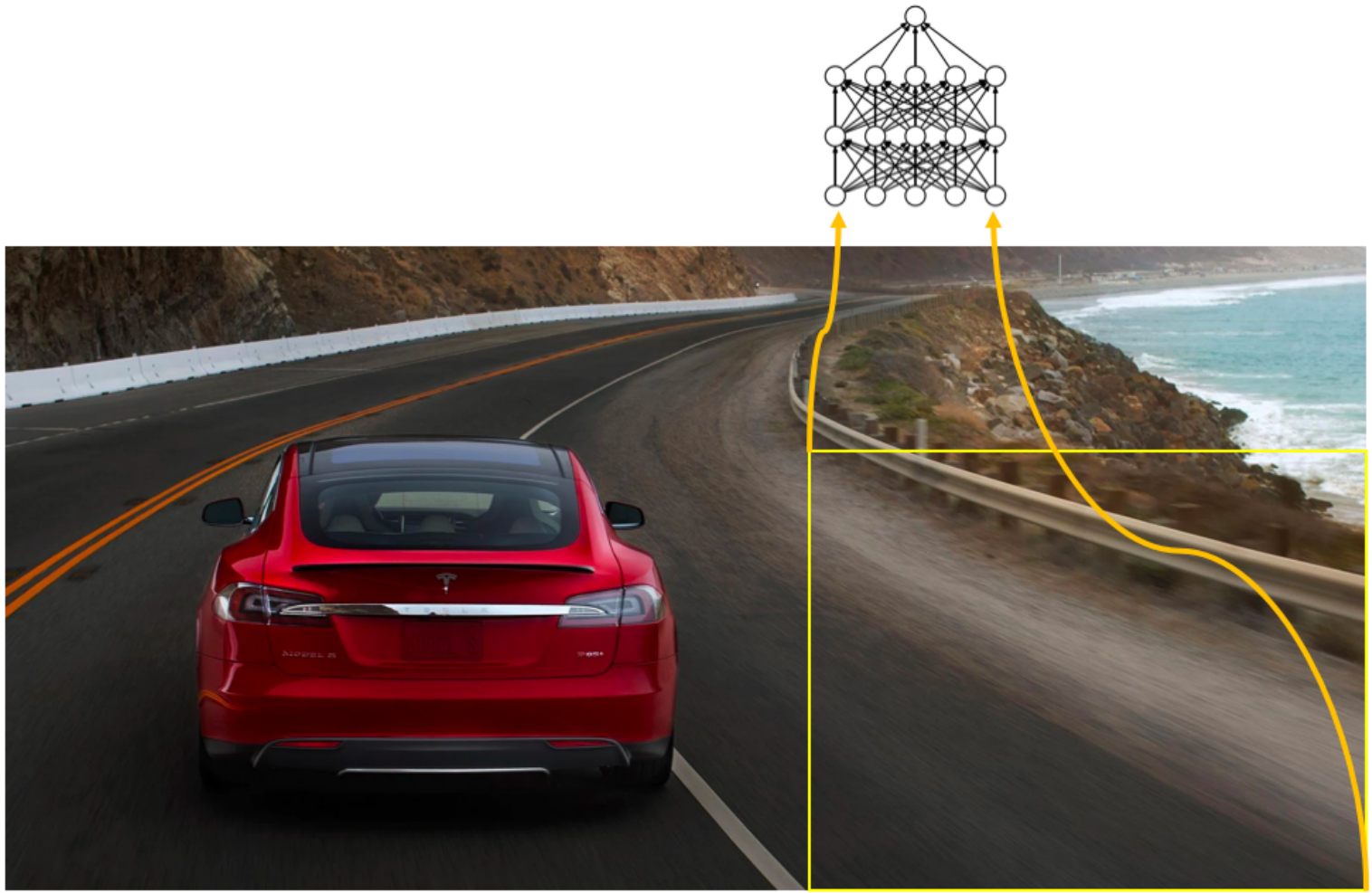
Solution: Scan the Image

- Scan for the target object at each location using a patch (kernel) of size $K \times K$.

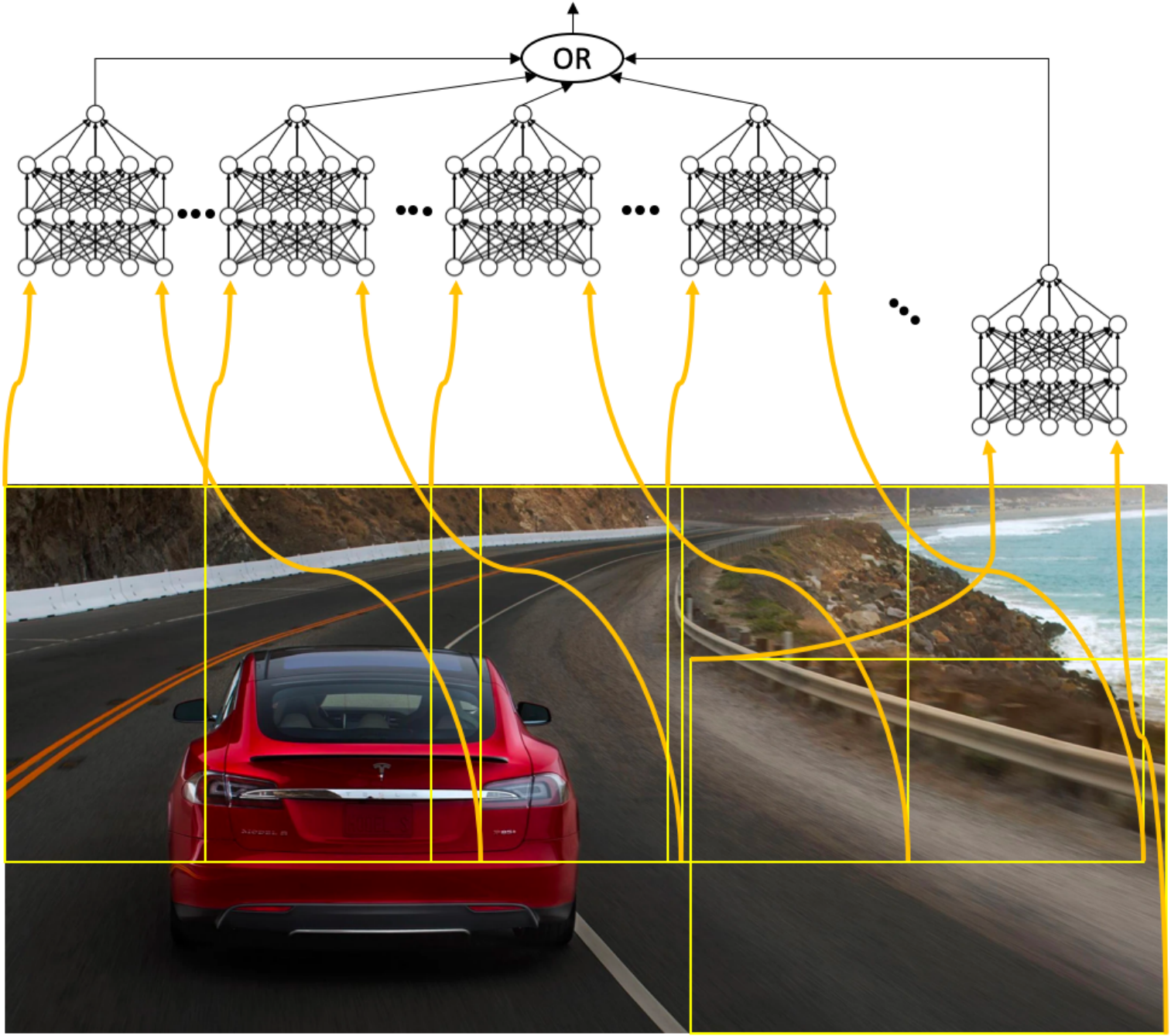


Solution: Scan the Image

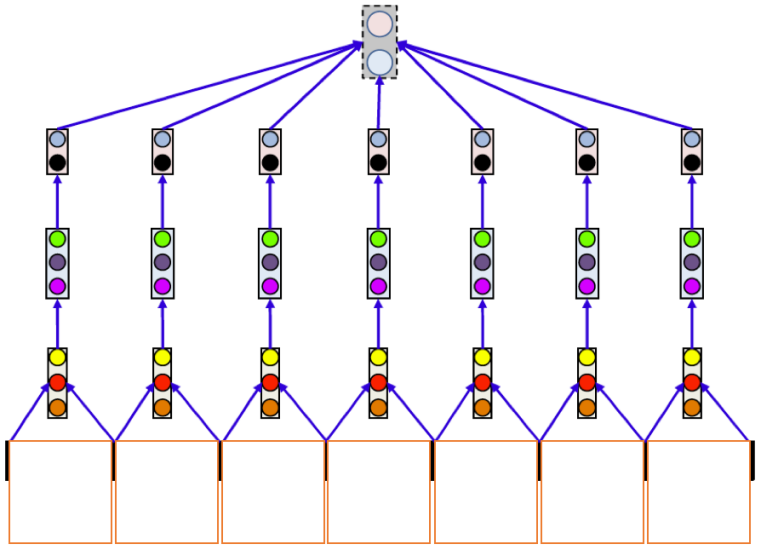
- Scan for the target object at each location using a patch (kernel) of size $K \times K$.



Combining all the scans, we get a Giant Neural Network!



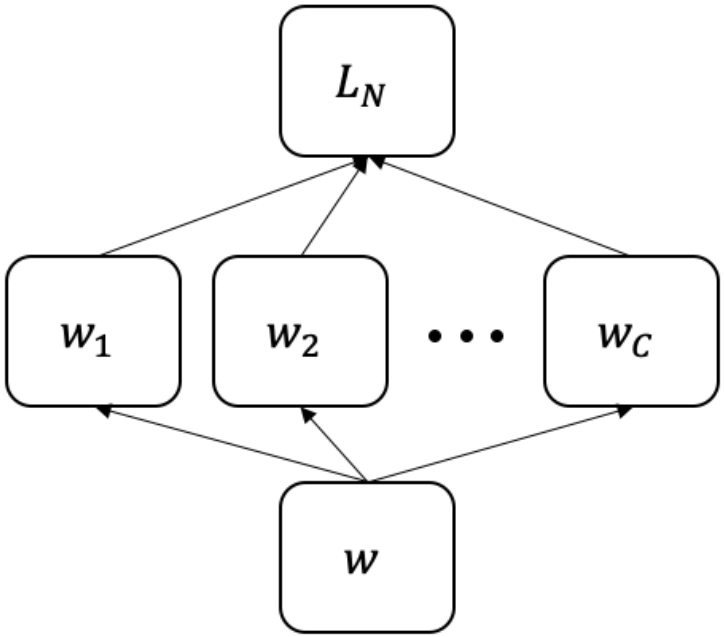
Note: This is a Shared Parameter Network!



```
def giant_nn(Img,K):
    W = width(Img)
    H = height(Img)
    C = num_channels(Img)
    for i = 1:W-K+1
        for j = 1:H-K+1
            ImgSegment = Img(1:C,i:i+K-1, j:j+K-1)
            y(i,j) = NN(ImgSegment)
    Y = OR(y(1,1), ..., y(W-K+1,H-K+1))
```

- Note that this is not the same as regular NN models.
- **Shared Parameter Networks**:
  - `NN()` uses the same weights for different locations
  - Block-structured weight matrix, with identical blocks for each kernel scan!
  - In other words, any parameter update of one scan must update equally for all scans.

Training Shared Parameter Networks



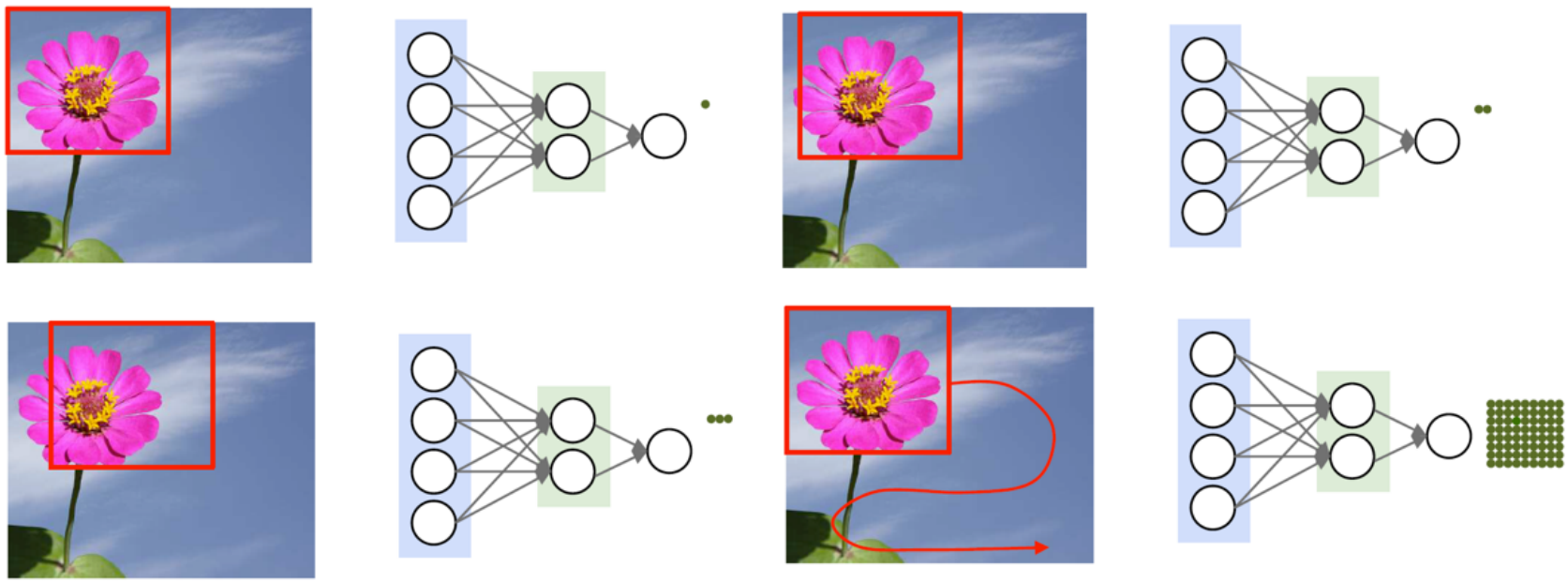
$$\frac{\partial L_N}{\partial w} = \sum_{c=1}^{C} \frac{\partial L_N}{\partial w_c} \cdot \frac{\partial w_c}{\partial w}$$

$$= \sum_{c=1}^{C} \frac{\partial L_N}{\partial w_c}.$$

- Say, we have L layers in `NN()` $\Rightarrow$ Let the weights be $W_1, \cdots, W_L$
- Let $\mathcal{S}$ denote the set of all edges with common value
- $\nabla_{w_{\mathcal{S}}} L_N = \dfrac{\partial L_N}{\partial w_{\mathcal{S}}} = \displaystyle\sum_{c \in \mathcal{S}} \frac{\partial L_N}{\partial w_c}.$
- Update step: $w_{\mathcal{S}} \leftarrow w_{\mathcal{S}} - \eta \nabla_{w_{\mathcal{S}}} L_N$,
- For every $(\ell, i, j) \in \mathcal{S}$, define $w^{\ell}_{i,j} = w_{\mathcal{S}}$.
- Continue until the stopping criterion is satisfied.

Take a Closer Look at Scanning...

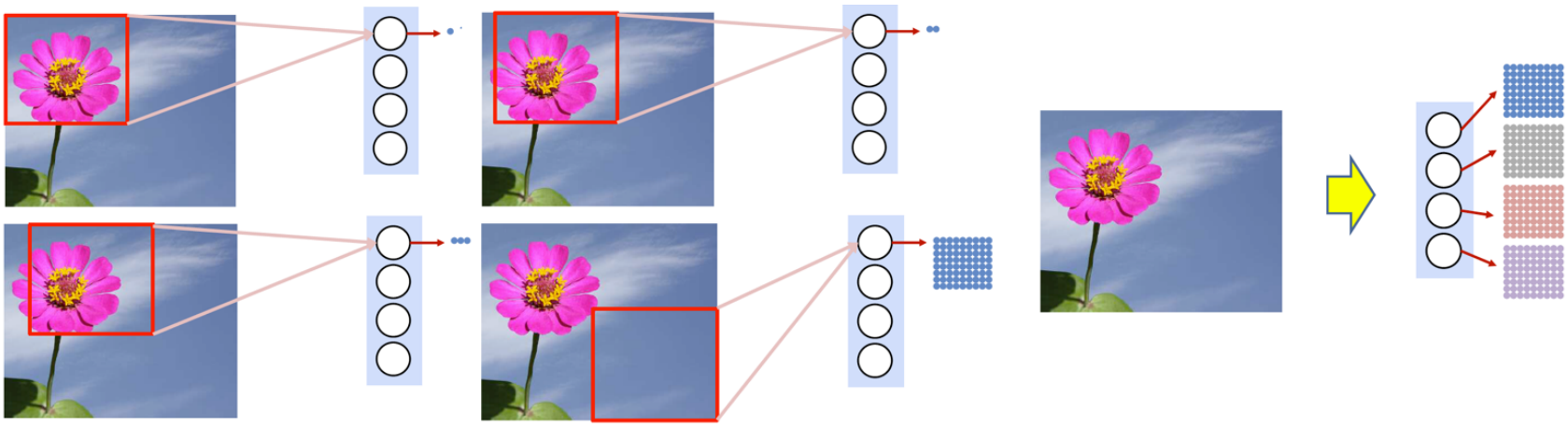
- Each scan passes the image segment through the entire network and produces an outcome
- Combining all the scans, we get a collection of outcomes, which are passed through OR (or softmax, or any other flat-NN) gate.



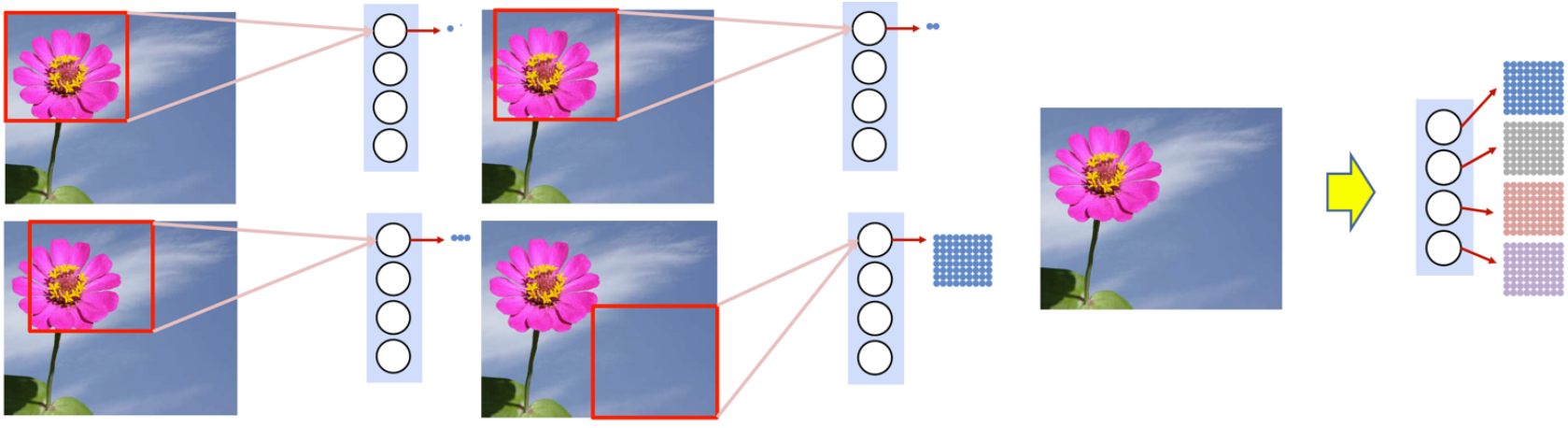
What if, we compute the outcome of one neuron for all scans?

A New Scanning Approach...

- Scan the entire image with one neuron
- Arrange the neuron's outputs from the scanned positions according to their positions in the original image

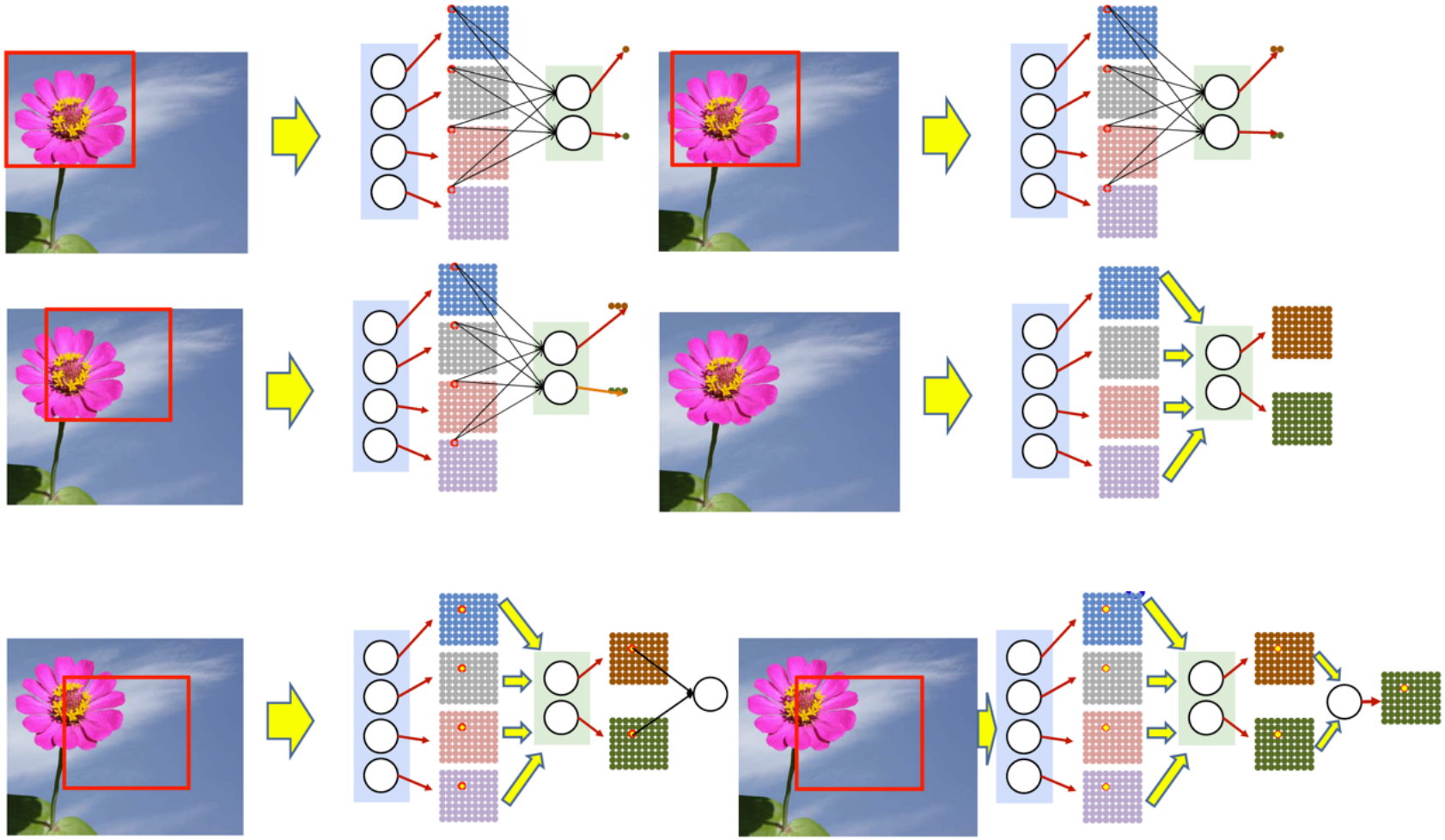


- Recurse the same logic across layers...



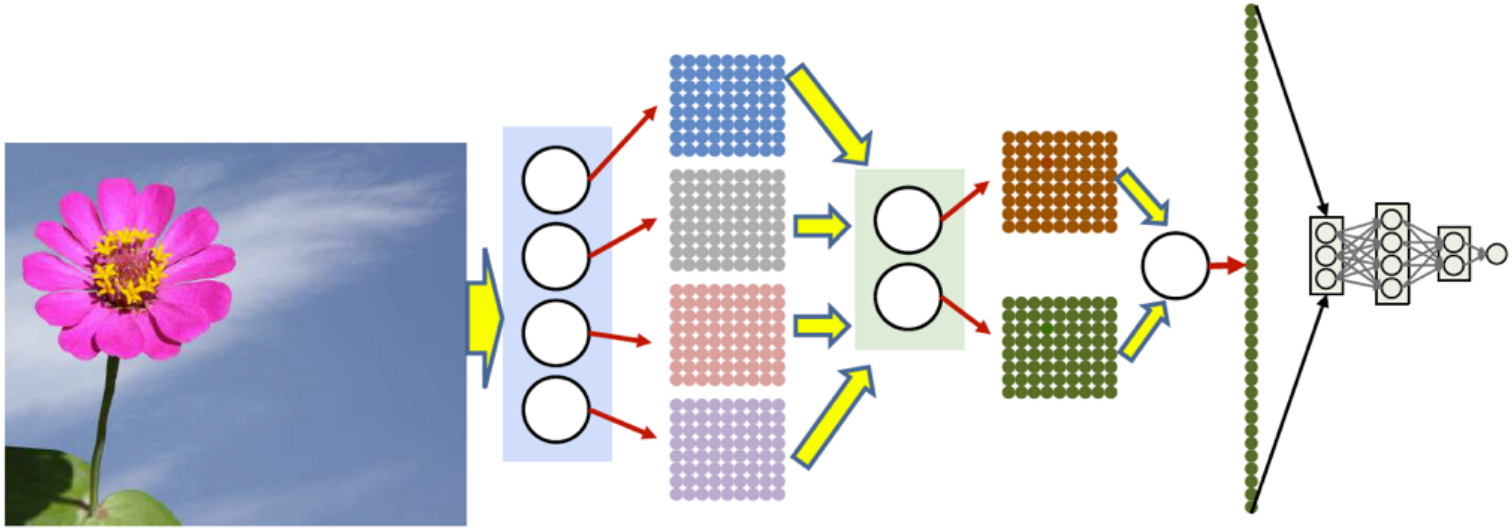
A New Scanning Approach (cont...)

- Subsequent layers jointly scan multiple maps of the previous layer.
- Final layer $\Rightarrow$ A map that detects flower at each location in the image.

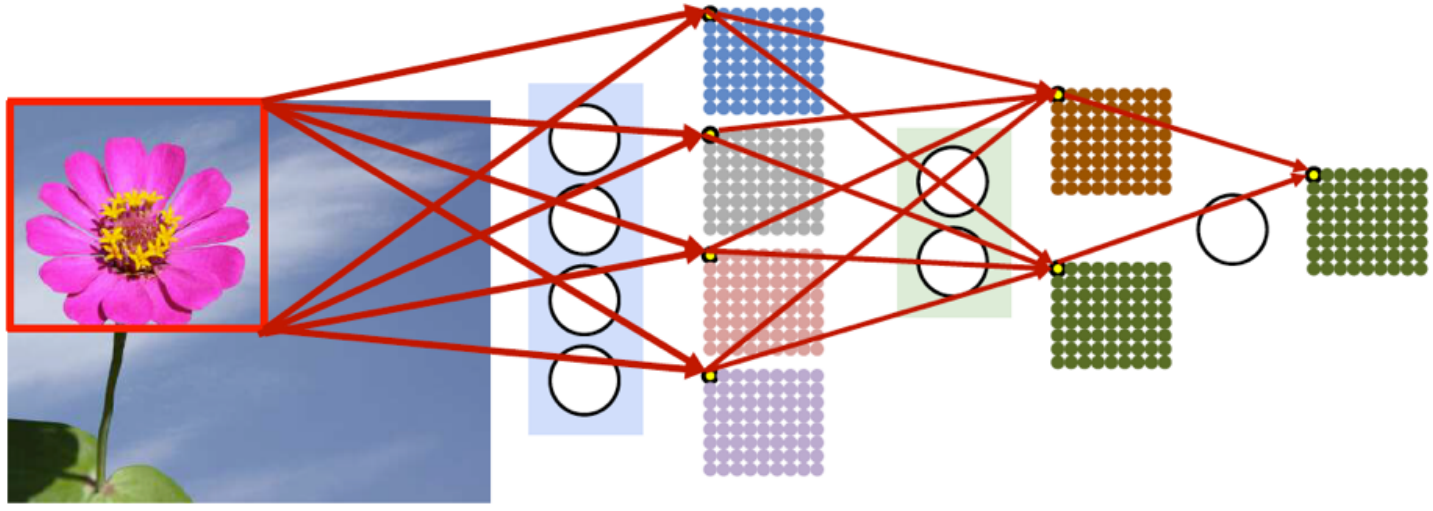


A New Scanning Approach (cont...)

- Goal: Detect the flower, not locate it!
- Flatten the final map and pass it into a softmax layer, or another NN.

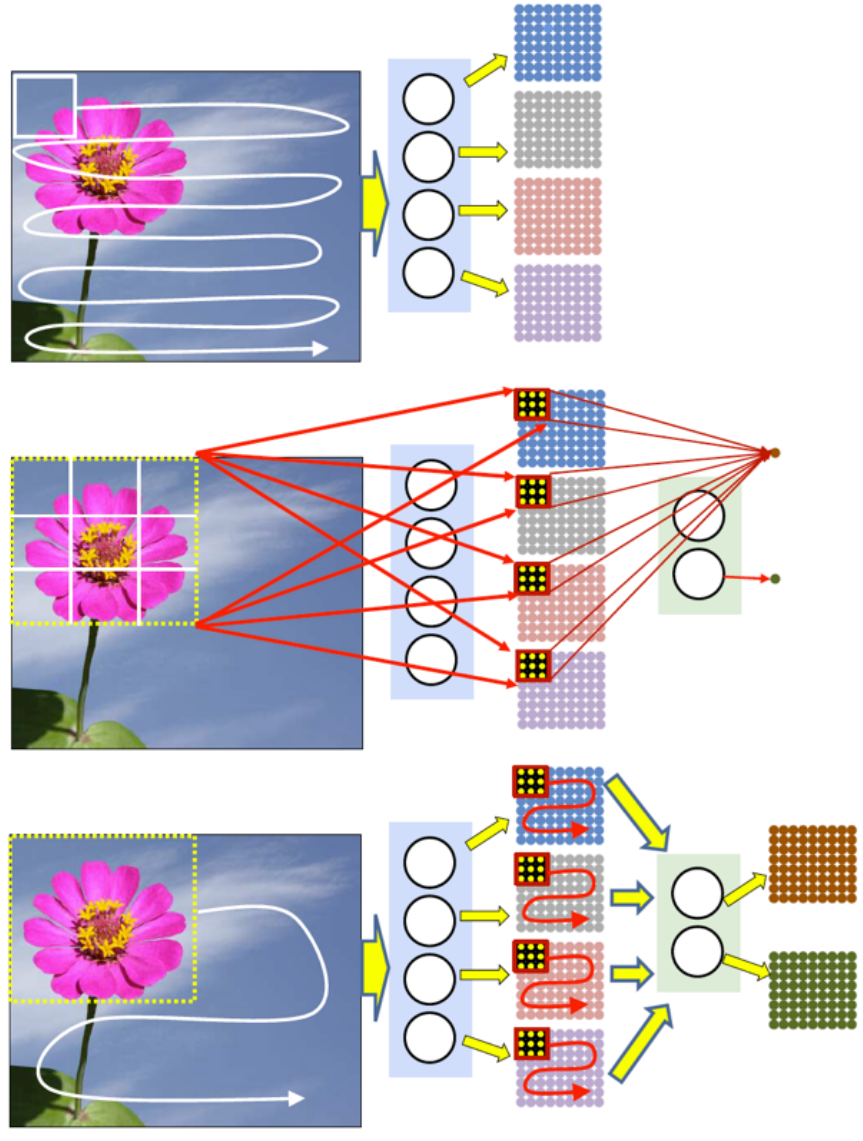


Is this scan sufficient?



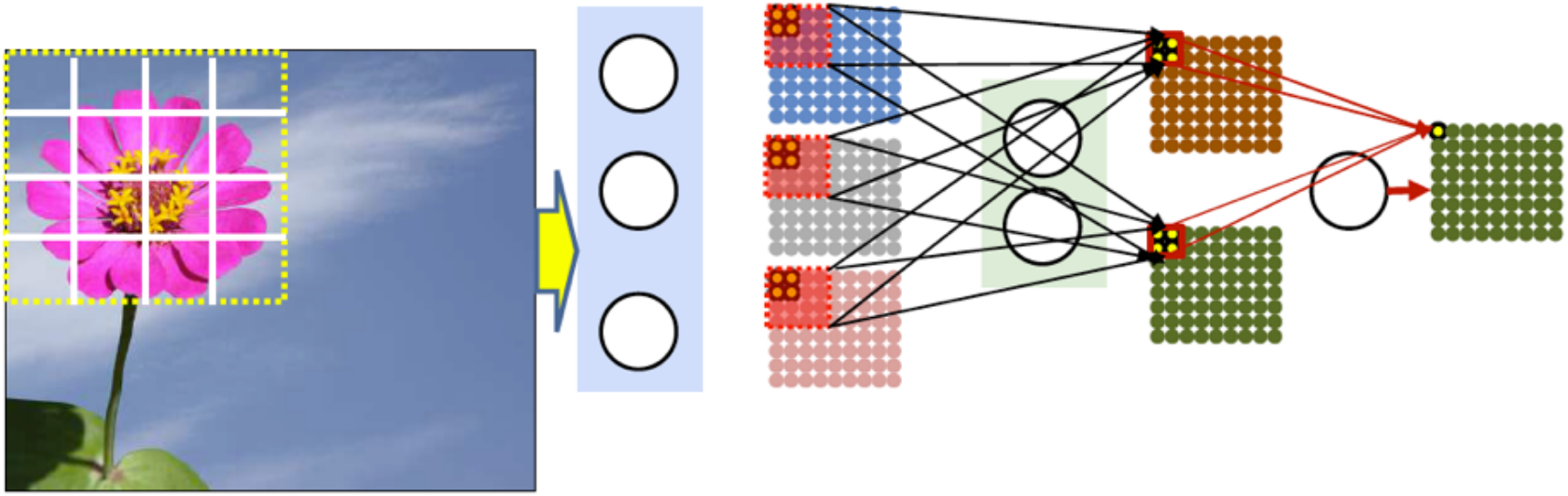
A New Scanning Approach (cont...)

- Distribute the scan $\Rightarrow$ Subsequent layers evaluate blocks of outputs from previous layers
- Now, we can evaluate larger windows
  - Windows distributed over layers $\Rightarrow$ Useful to learn larger patterns
- Jointly scan all first-layer maps $\Rightarrow$ Output of second layer represents a scan of full-sized input window



Convolutional Neural Network (CNNs)

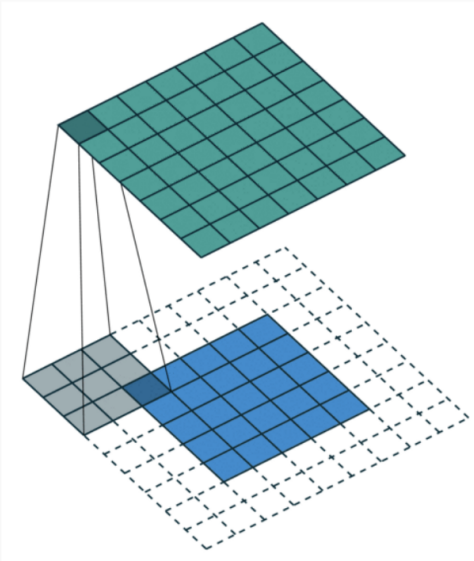
Use the same logic recursively across multiple layers, we obtain a convolutional neural network!



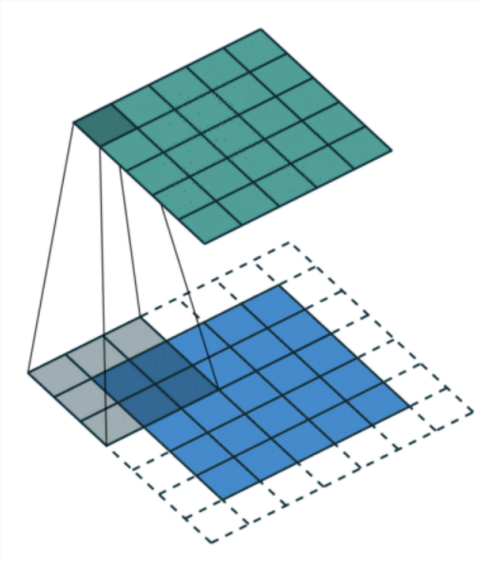
```
def cnn(Img,L,K):
    W = width(Img)
    H = height(Img)
    C = num_channels(Img)
    for l = 1:L
        Compute W_{l-1}, H_{l-1}, K_l, D_l, D_{l-1}
        for x = 1:W_{l-1} − K_l + 1
            for y = 1:H_{l-1} − K_l + 1
                Segment = Y(l-1, 1:D_{l-1}, x:x+K_l-1, y:y+K_l-1)
                for j = 1:D_l
                    Compute z(l,j,x,y) from Segment
                    Y(l,j,x,y) = activation(z(l,j,x,y))
    Y = softmax(Y(L,:,1,1), ···, Y(L,:,W+K-1,H-K+1))
```

Padding

- Sometimes, vital information is present on the corners of previous layers' maps.
- Large kernels will ignore information present in the corners of the activation maps.
- Consequence: Severe drop in accuracy in deeper networks
- **Solution:** Padding
  - *Zero Padding:* Append zeros
  - *Replication Padding:* Replicate the pixel value in the closest neighboring position.
  - *Reflective Padding:* Replicate the pixel value in the position at the same distance from closest neighboring position, but on the opposite side.



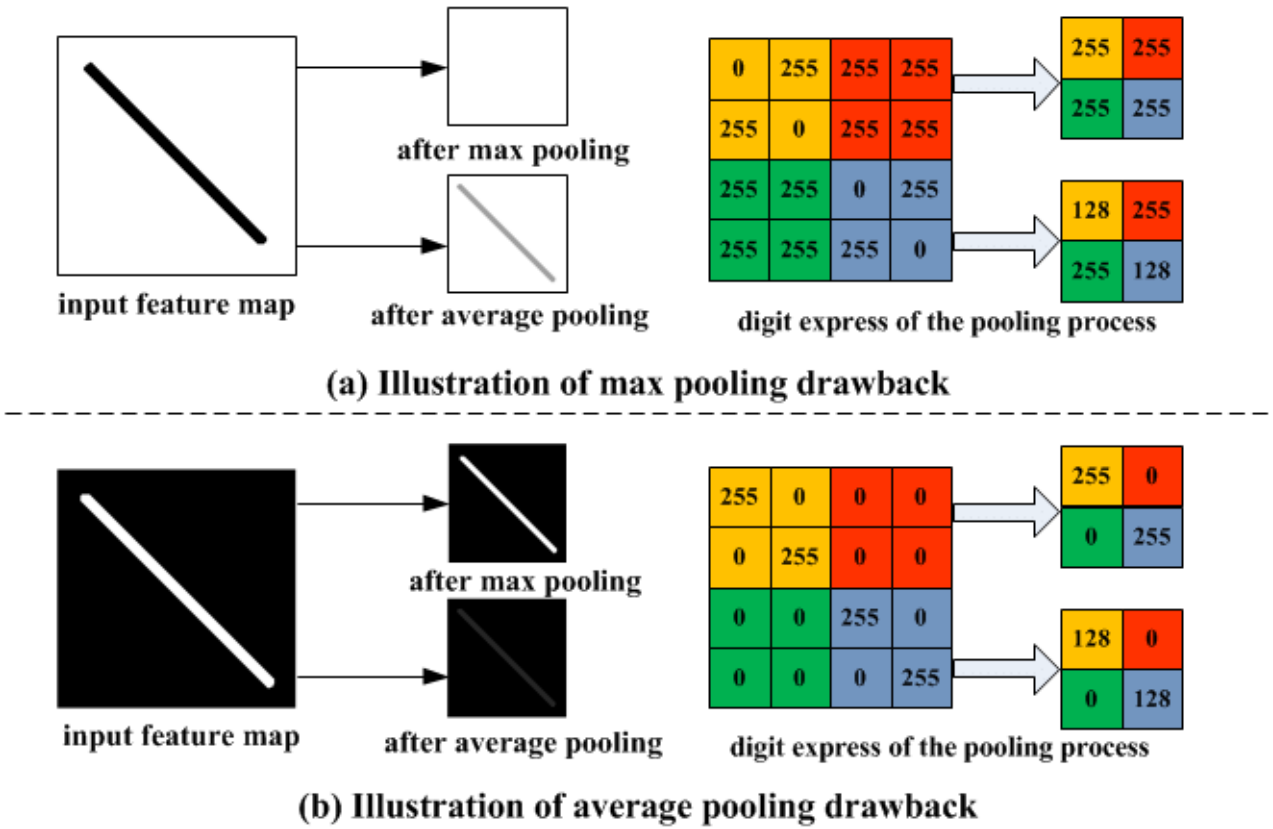
Full padding. Introduces zeros such that all pixels are visited the same amount of times by the filter. Increases size of output.



Same padding. Ensures that the output has the same size as the input.

Robustness to Jitter: Pooling

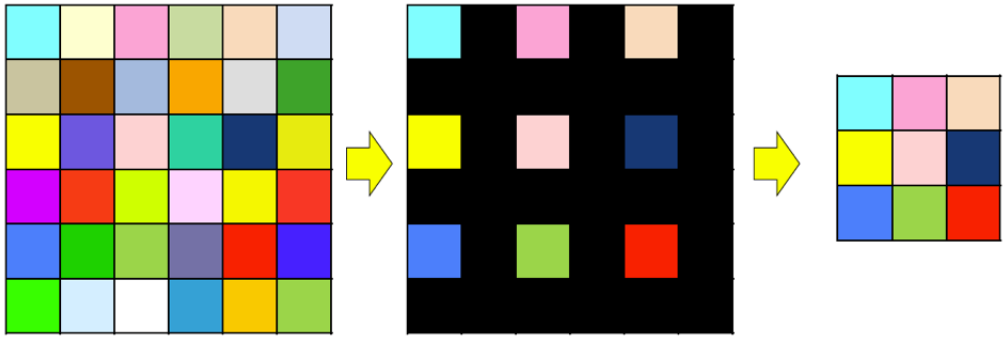
- ***Average Pooling:*** Compute the average of all entries in the kernel
- ***Max Pooling:*** Compute the maximum of all entries in the kernel
- Strong push towards completely ignoring pooling operation[1]



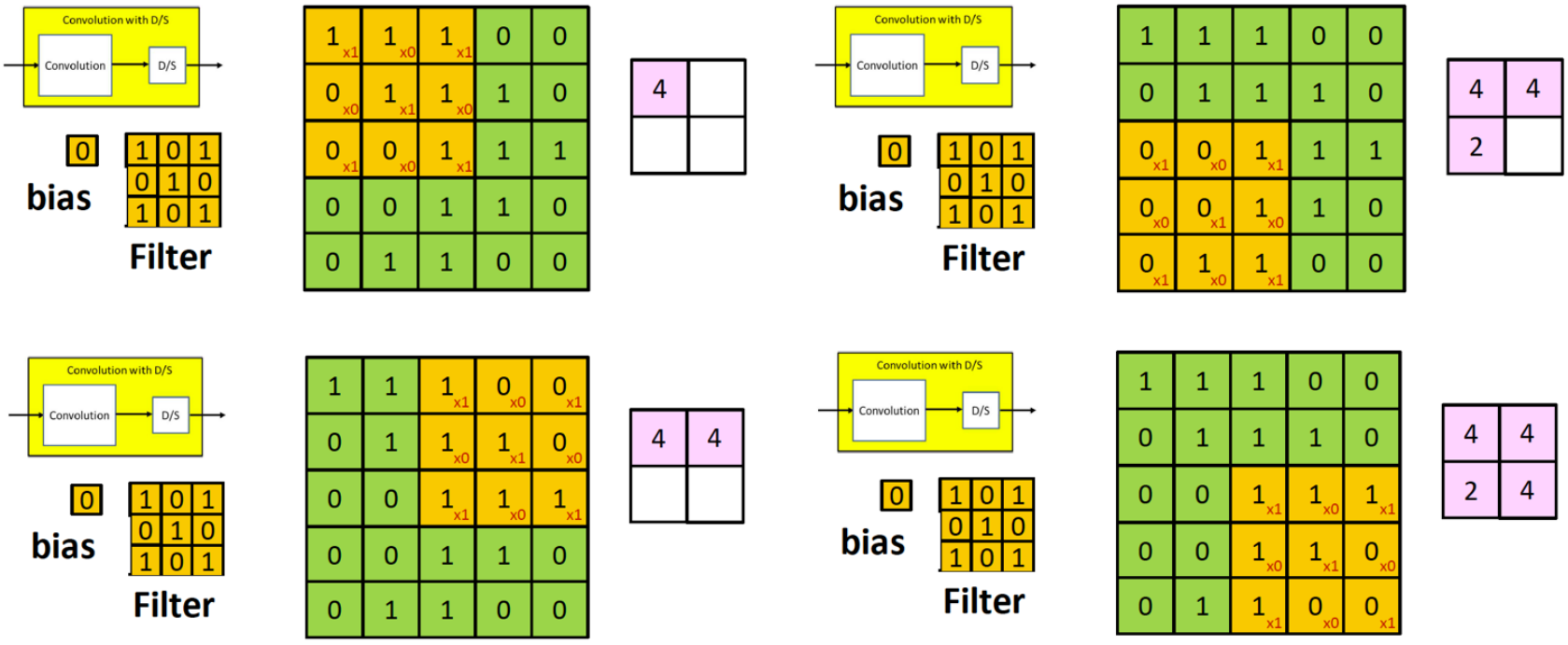
[1] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller, "Striving for Simplicity: The All Convolutional Net," arXiv preprint:1412.6806, 2014.

Need for Downsampling: Striding

- Downsampling: Reduces the size of map by a factor of s in every direction.

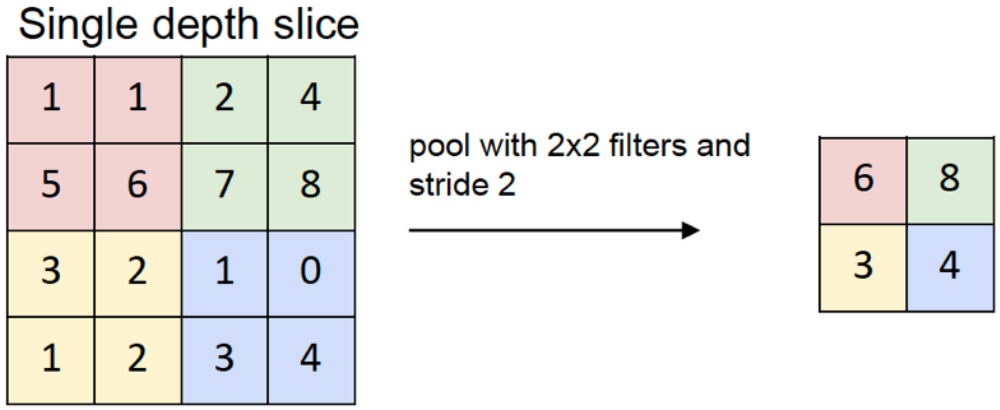


- Convolution + Downsampling $\Rightarrow$ Strides (More efficient method!)

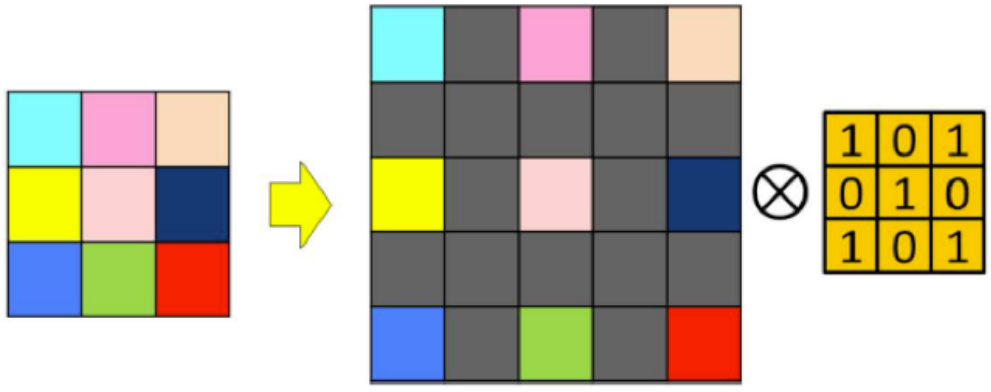


Striding (cont...)

- Pooling + Downsampling $\Rightarrow$ Strides (More efficient method!)

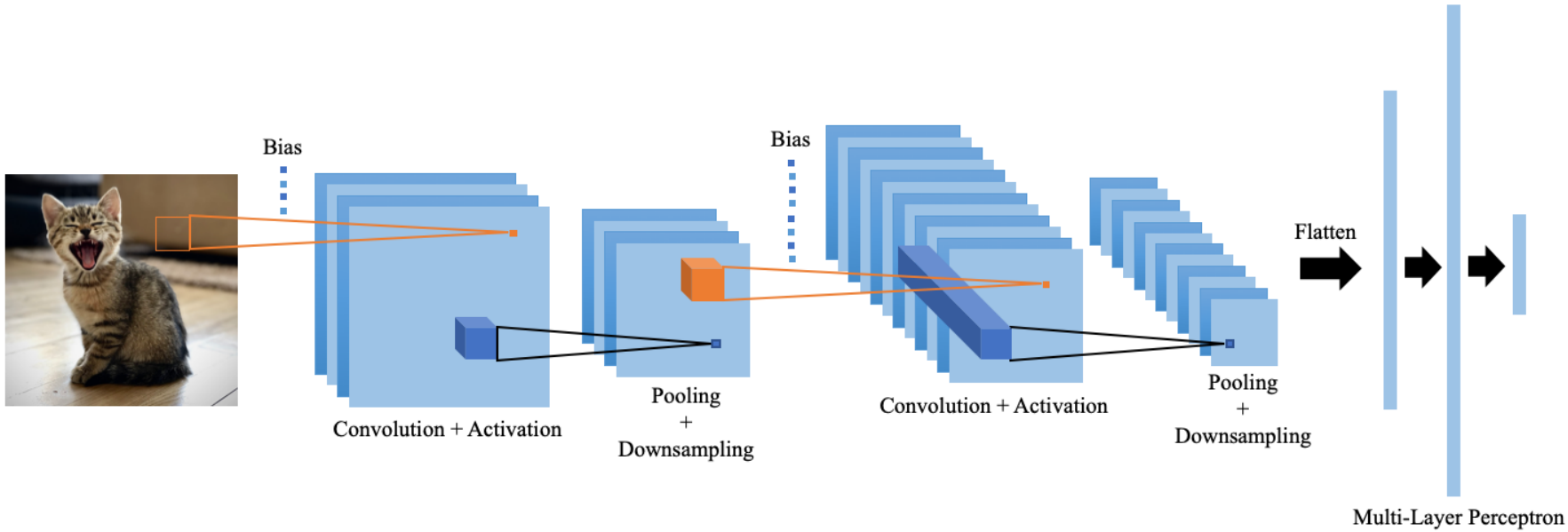


- Upsampling: Increases the size of map by a factor of s in every direction.
  - Typically used after a conv layer
  - Never in conjunction with pooling
  - Not in the final layer...



A General CNN Architecture

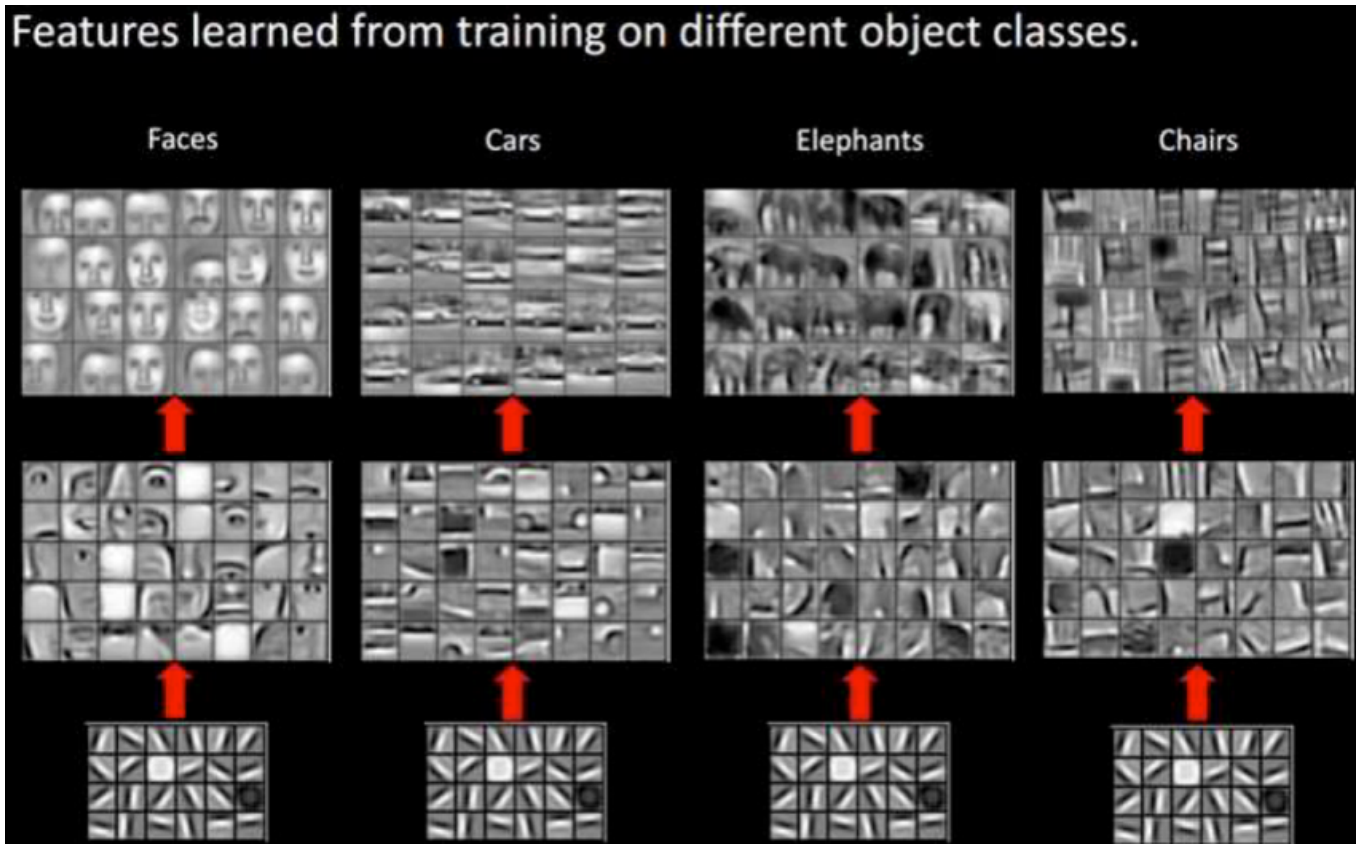
Combining all the feature maps, we obtain...



- **Model parameters updated during training:** Filter weights, bias variables and perceptron weights.
- **Layer Dimensions:** $N \times N$ input + $K \times K$ kernel + s stride = $\left(1 + \left\lfloor \frac{N-K}{s} \right\rfloor\right) \times \left(1 + \left\lfloor \frac{N-K}{s} \right\rfloor\right)$ output – no padding case
- Vectorizing convolution operation demands inner product of *tensors*.
- **Note:** No specific order between convolution layers and downsampling layers.
- For any CONV layer, there is an FC layer that implements the same forward function. The converse is also true. (Ref. https://cs231n.github.io/convolutional-networks/)

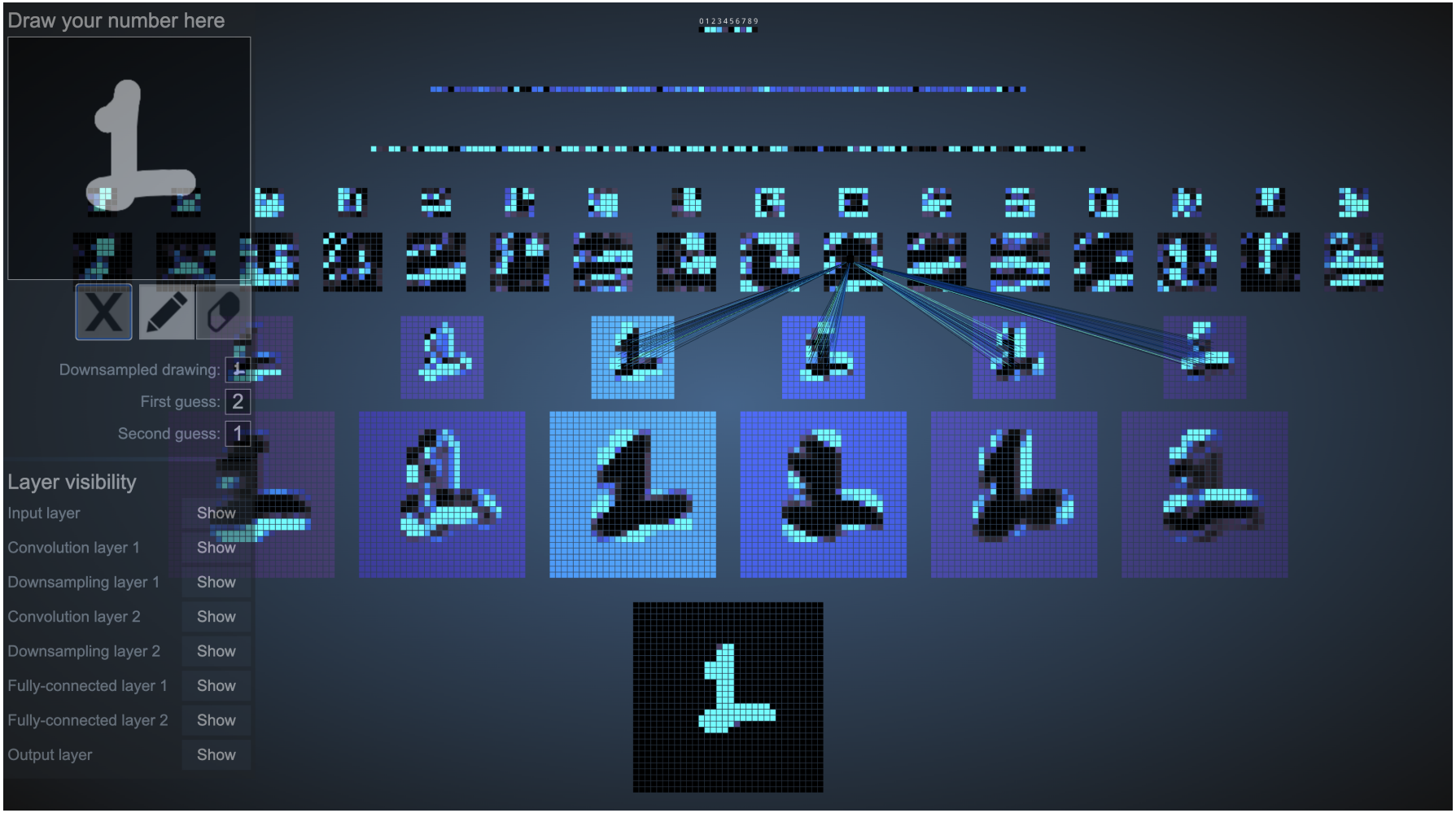
Receptive Fields of CNNs

- Receptive field is the pattern of input image, as seen by each neuron (filter).
- Layer 1: Simple arrangement of weights
- Higher Layers: Non-trivial exercise
  - What patterns in the input does a given neuron respond to?
  - Set the output of neuron to 1 and learn the input via backpropagation.



Illustrative Example: Visualizing Receptive Fields of CNNs[2]

- Layer 1: Convolutional layer (6 filters) + Downsampler
- Layer 2: Convolutional layer (16 filters) + Downsampler
- Layer 3: Flatten + Neural Network + Softmax Output

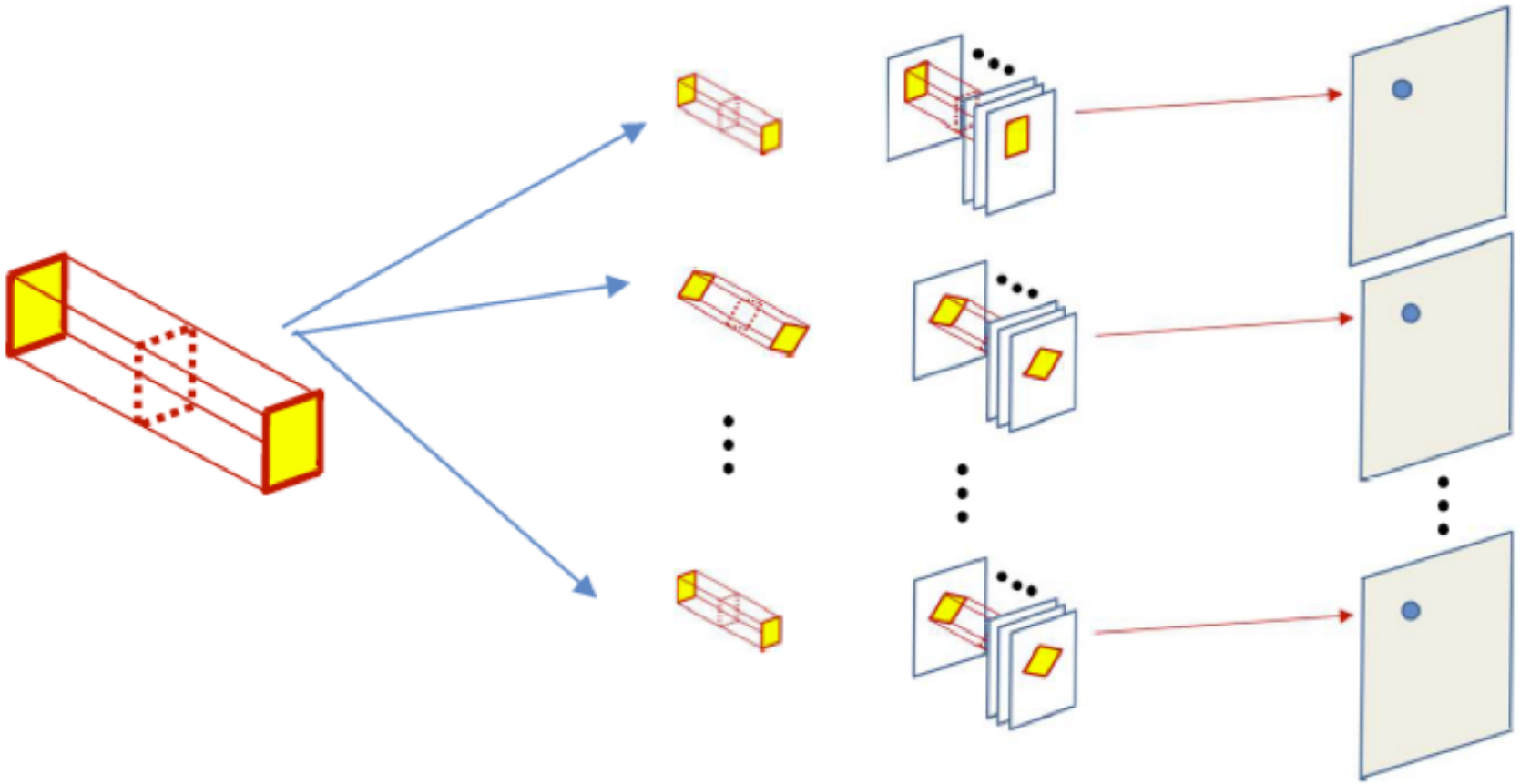


[2] Source: A. W. Harley, "An Interactive Node-Link Visualization of Convolutional Neural Networks," in ISVC, pages 867-877, 2015, Available: http://www.cs.cmu.edu/ aharley/vis/

Beyond Shift Invariance

CNNs are shift-invariant!
But, are they rotation, scale, or reflection invariant?

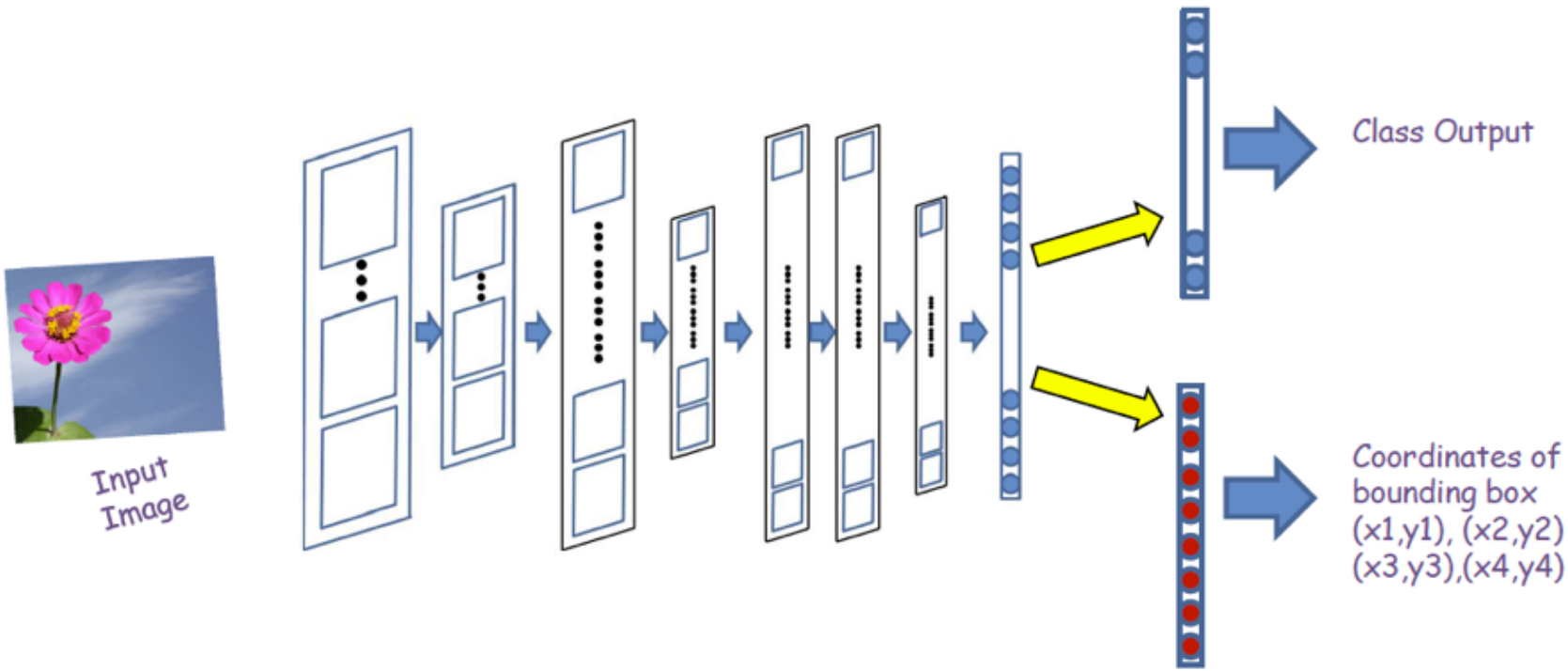
- Transform invariance can be achieved by introducing a finite set of neurons based on transformed kernels.
- Gradients flow back through transformations to update filter weights.



Bounding Boxes

This is multi-task learning – a joint detection + localization problem!

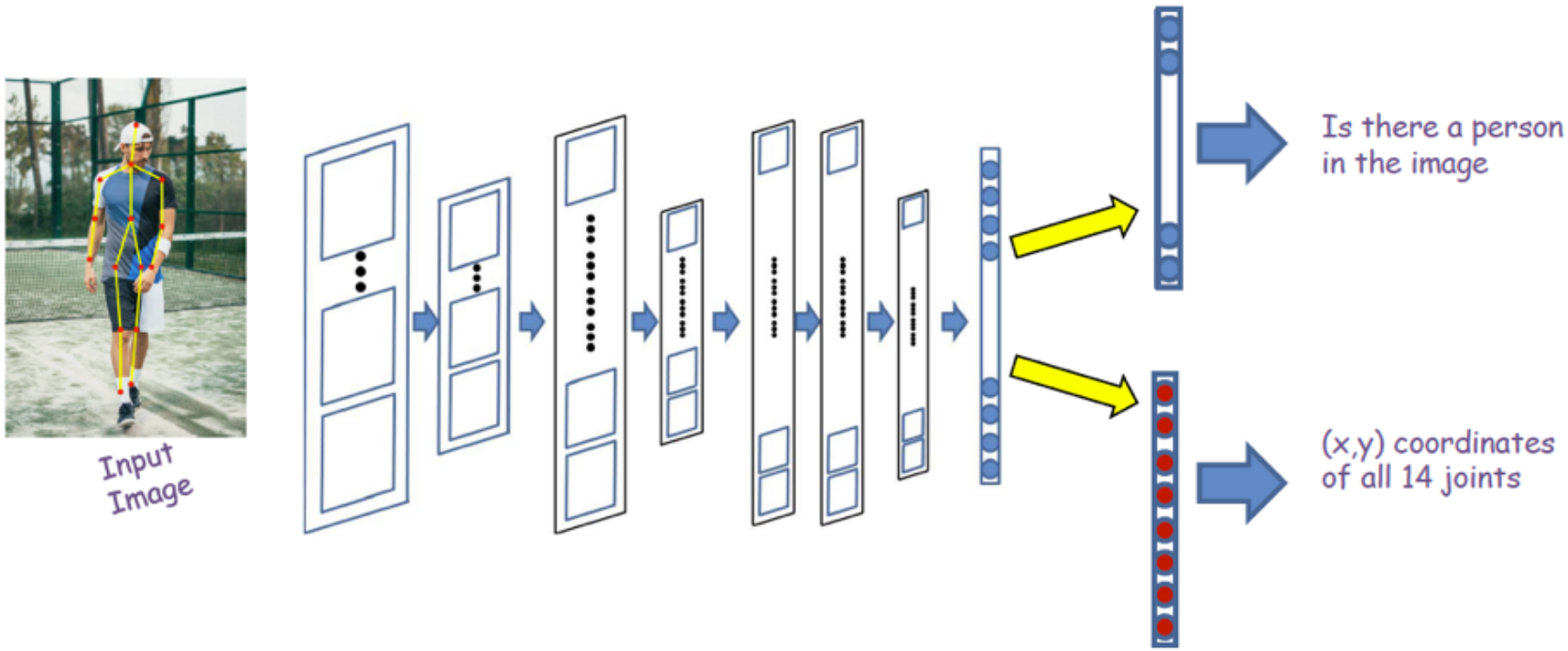
- Same architecture with two separate output layers.
  - One for predicting the class label
  - The other predicts the corners of the bounding box.
  - Loss function: Sum of cross-entropy loss of detector and L2-loss for bounding box predictor.



Pose Estimation

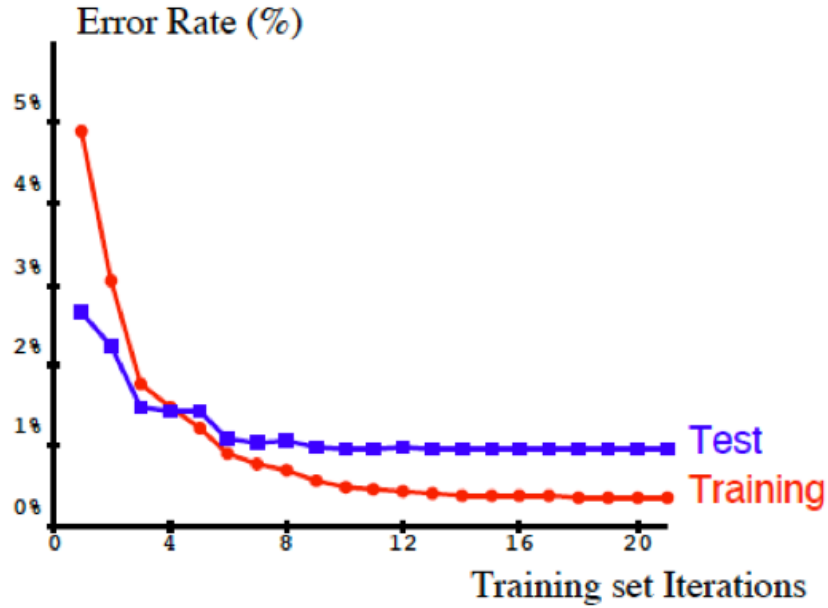
This is multi-task learning – a joint detection + pose estimation problem!

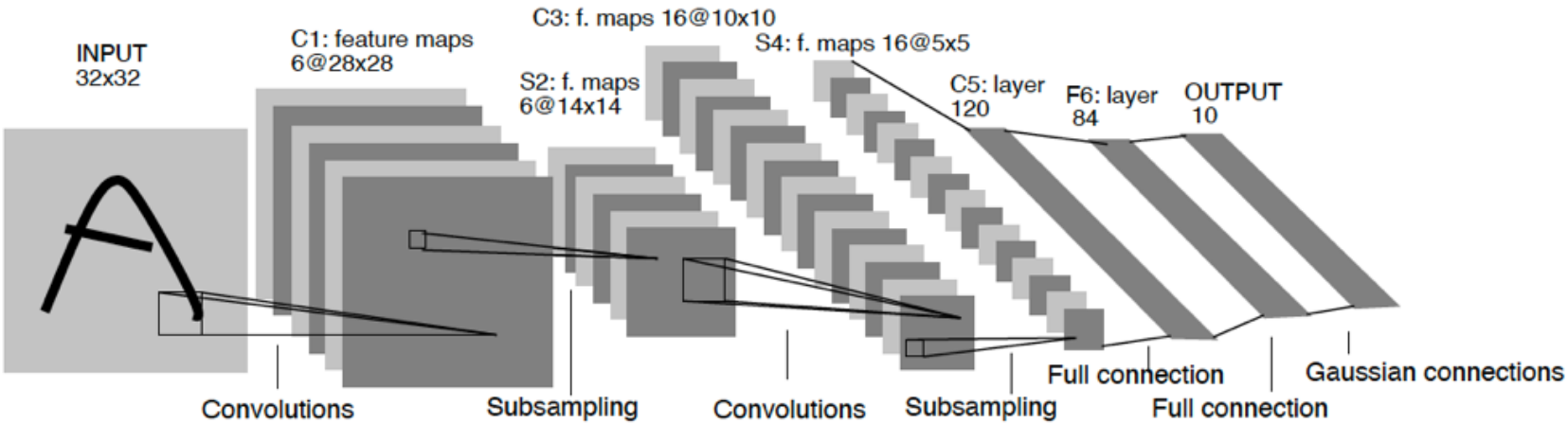
- Same architecture with two separate output layers.
  - One for predicting the class label
  - The other predicts the location of nodes in the stick model.
  - Loss function: Sum of cross-entropy loss of detector and L2-loss for stick-model predictor.



Evolution of CNN Architectures: LeNeT[3]

- Yann LeCun's architectures from 1990s
- **LeNet:** Best known in that era.
- Error rate: 1% on MNIST data





[3] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, 1998.