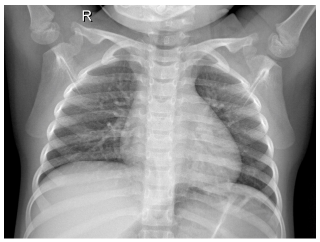
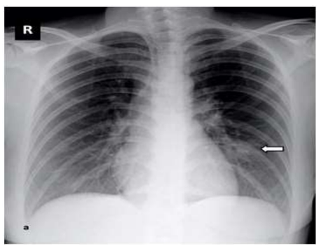
Convolutional Neural Networks

Traditional approach



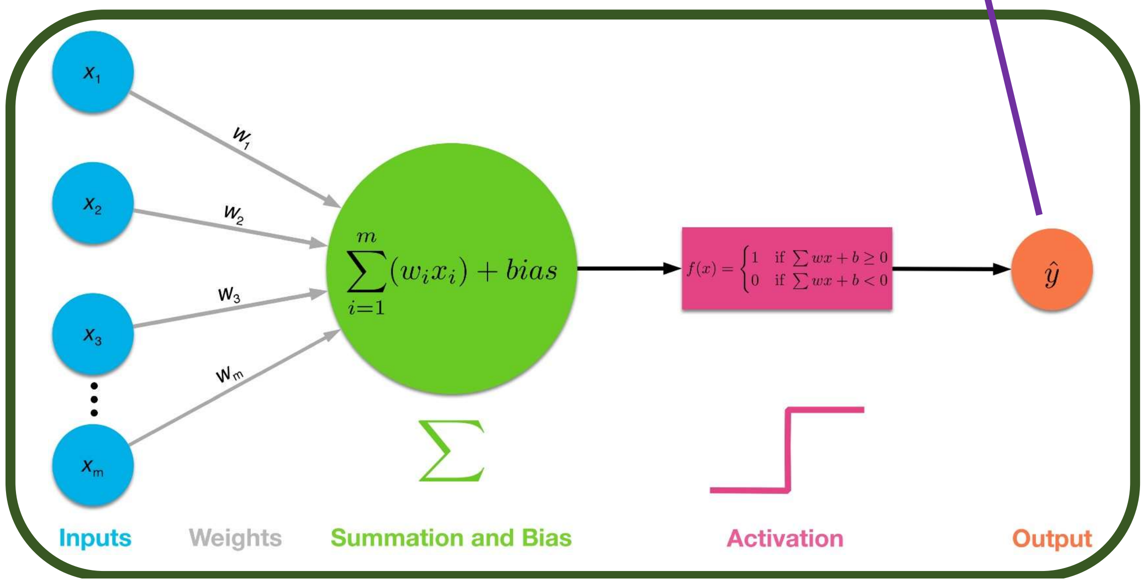


- Extract features, feed to the neural network

- Opacity Patterns
- Lesion Localization
- ……
- Pleural Effusion

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} = \begin{bmatrix} 2 \\ 1.5 \\ \dots \\ 8 \end{bmatrix}$$

0 – Healthy
1 – Corona

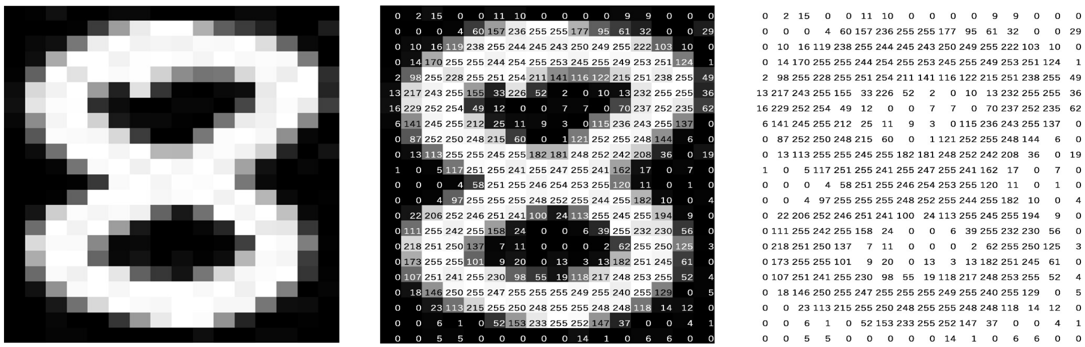


Wouldn't it be nice if we could feed the image directly ?

… and let the "learning process" figure out which features to extract ?

What is an Image?

- Data in the form of matrix (Rows and Columns) consisting of Pixels



https://mozanunal.com/images/pixel.png

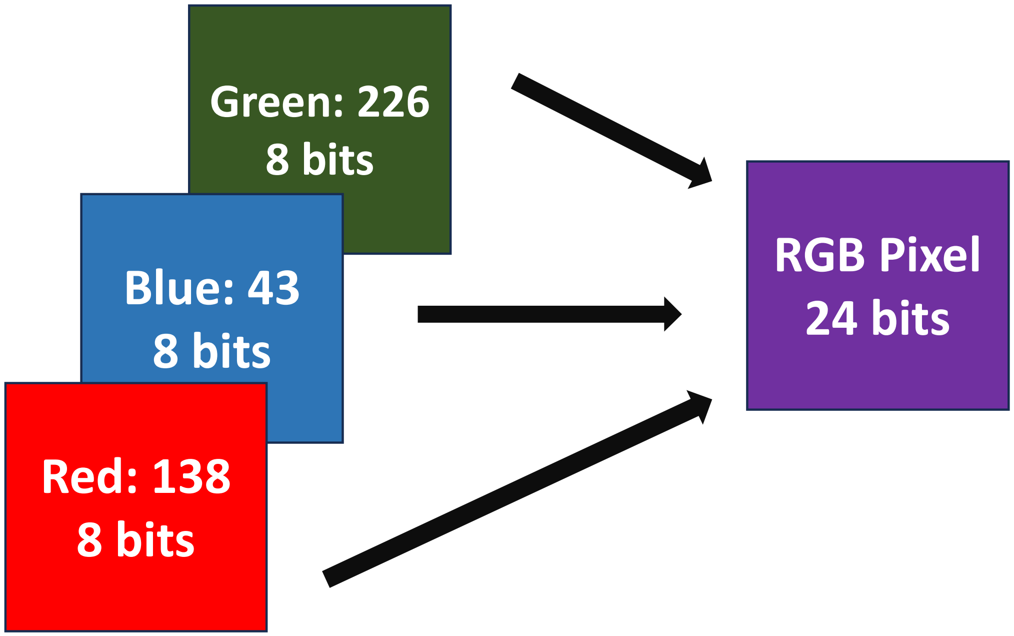
(a)
Color Image

(b)
Grayscale Image

(c)
Binary Image

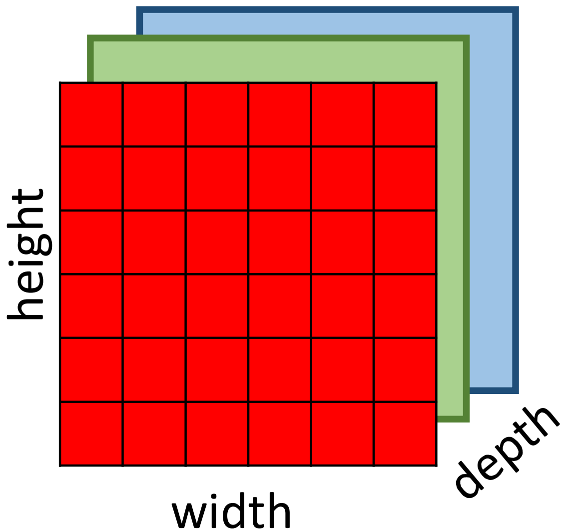
https://www.researchgate.net/profile/Sanskruti-Patel-2

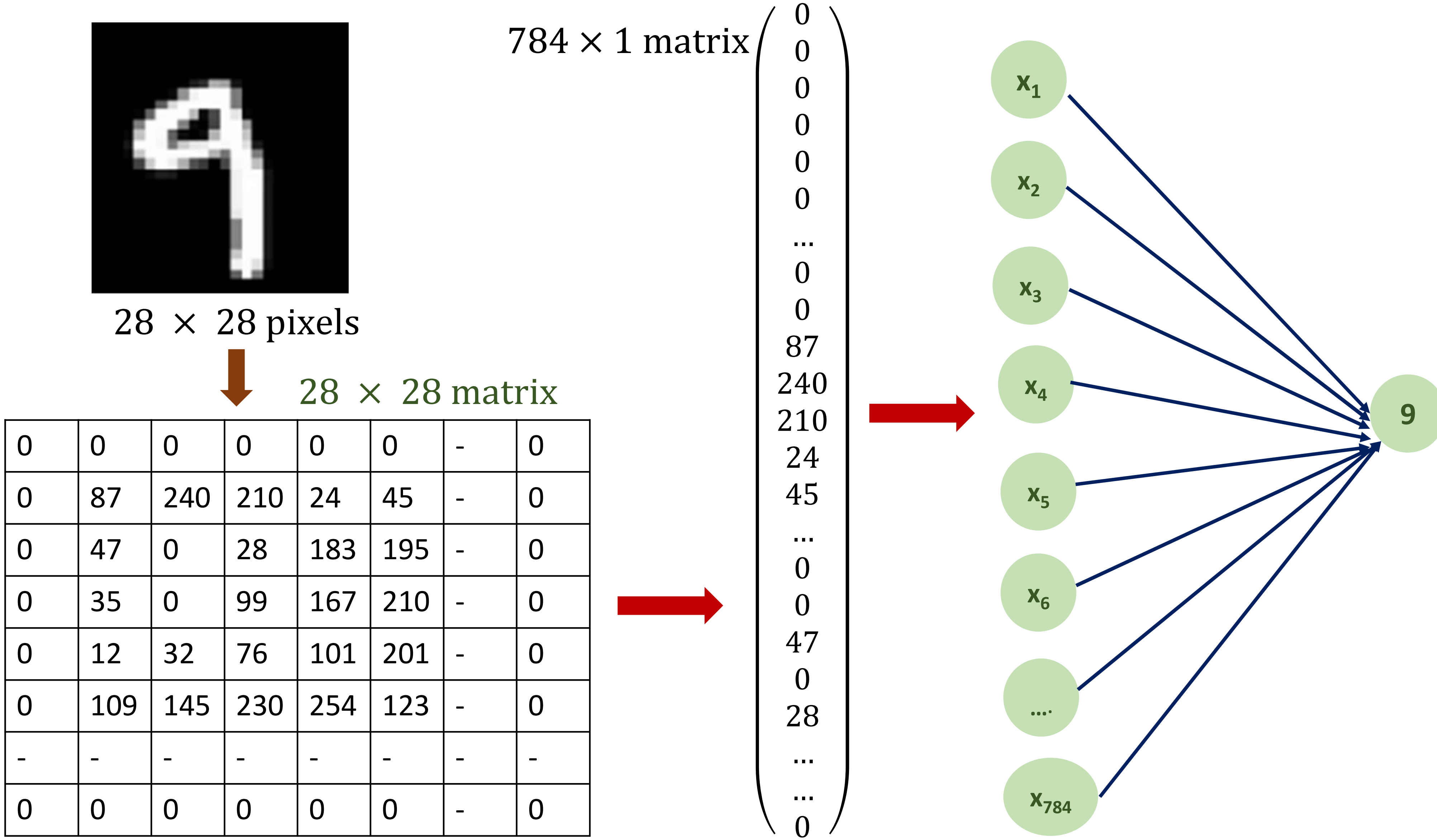
Color Image



Width × Height × Depth

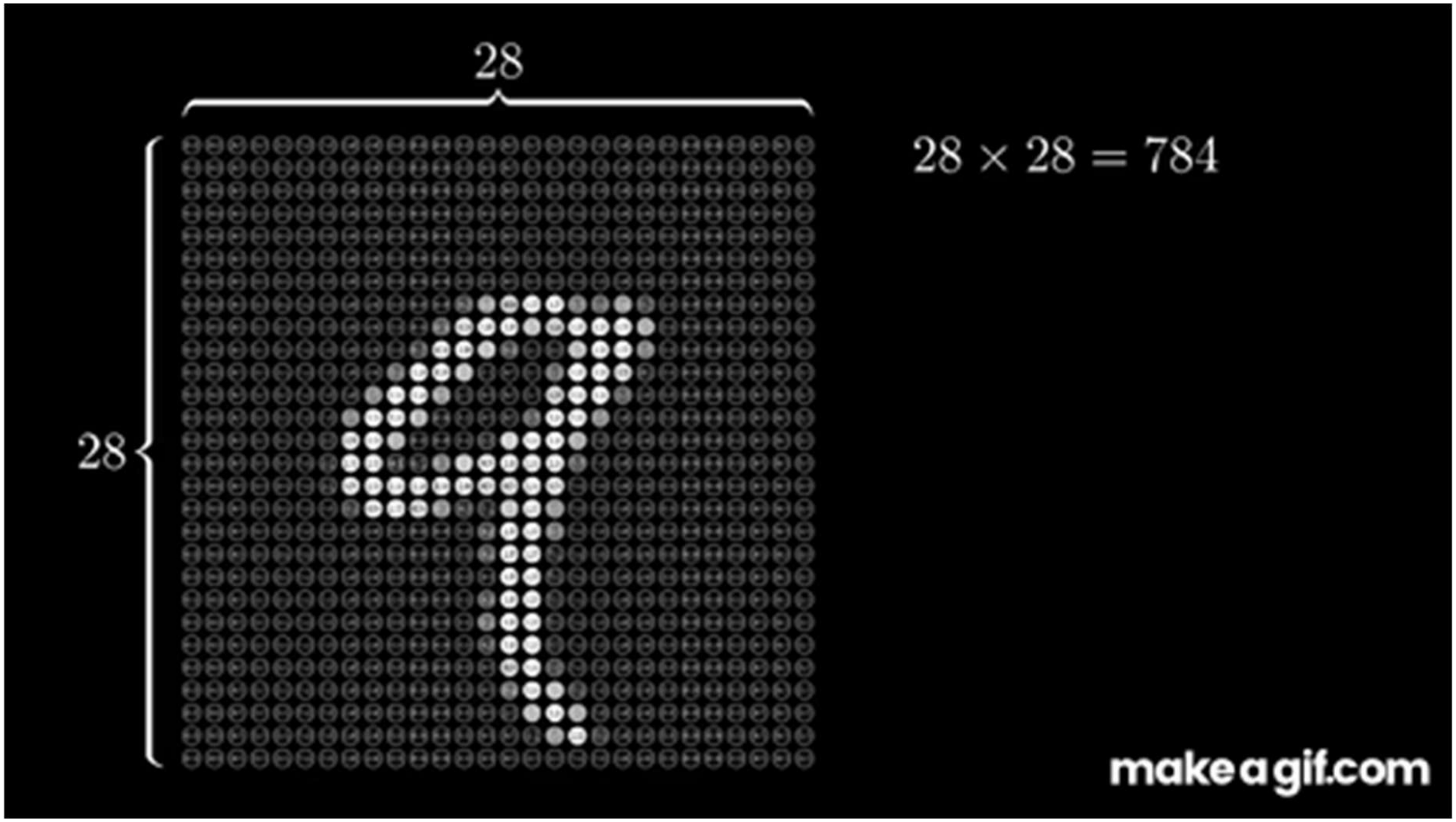
Depth: [Red, Green, Blue]





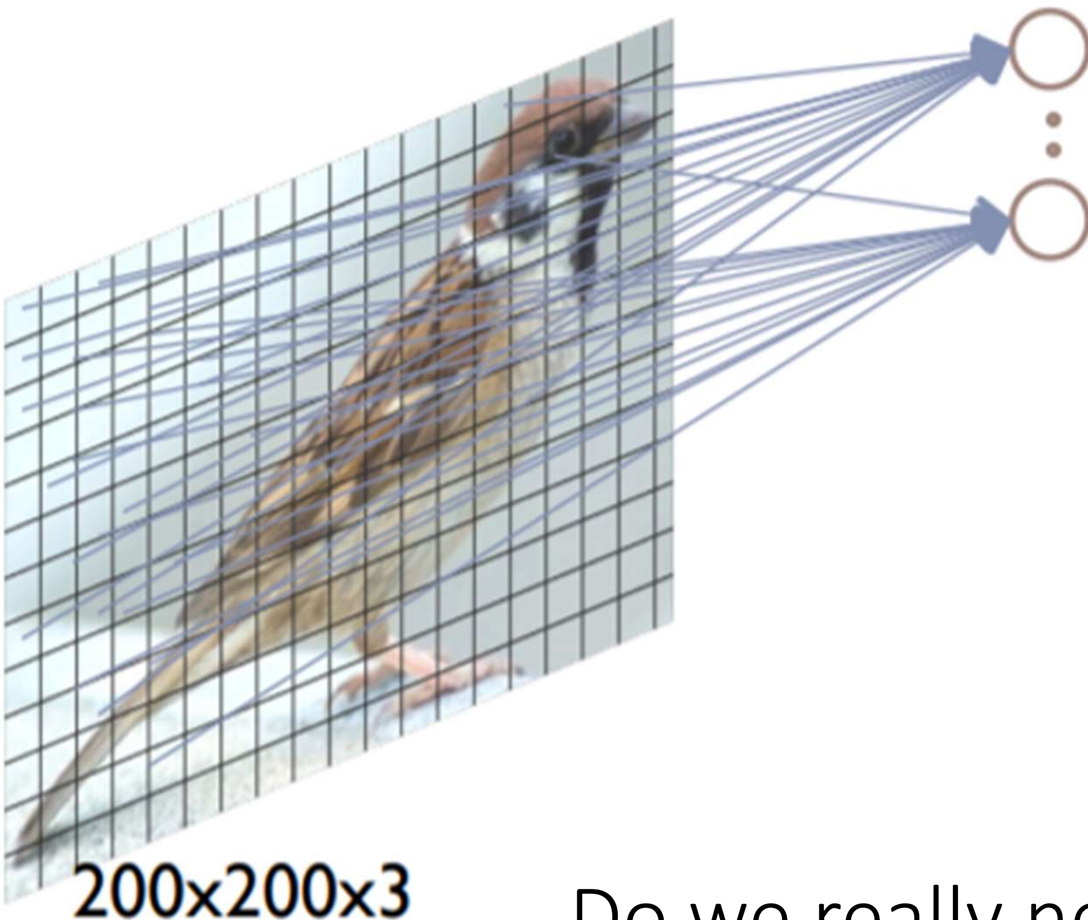
0	0	0	0	0	0	-	0
0	87	240	210	24	45	-	0
0	47	0	28	183	195	-	0
0	35	0	99	167	210	-	0
0	12	32	76	101	201	-	0
0	109	145	230	254	123	-	0
-	-	-	-	-	-	-	-
0	0	0	0	0	0	-	0

Flatten image, Feed to NN



https://www.youtube.com/watch?v=aircAruvnKk

For colour images?



- Input Layer: 120000
- # Hidden Units: 100000
- # Params to train: 12 billion
- Need huge training data to prevent overfitting
- Will not perform well, even if the image is shifted by one pixel
- No correlation exists among pixels

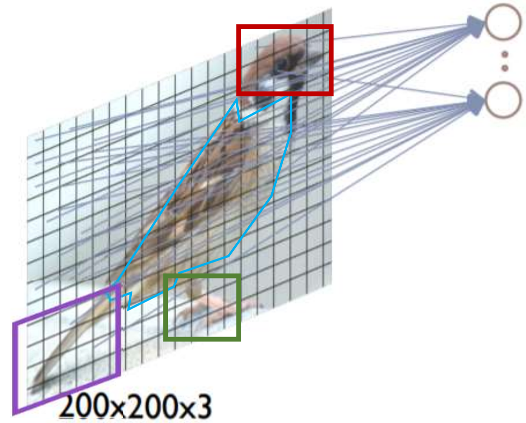
Do we really need full connectivity ?

Solutions?

- Huge number of parameters to train
- Need huge training data to prevent overfitting
- Will not perform well, even if the image is shifted by one pixel
- No correlation exists among pixels

- Reduce the number of parameters, # of input nodes
- Tolerate small shifts in where the pixels are in the image
- Take advantage of the correlations that we observe in complex images

Hierarchical Combination



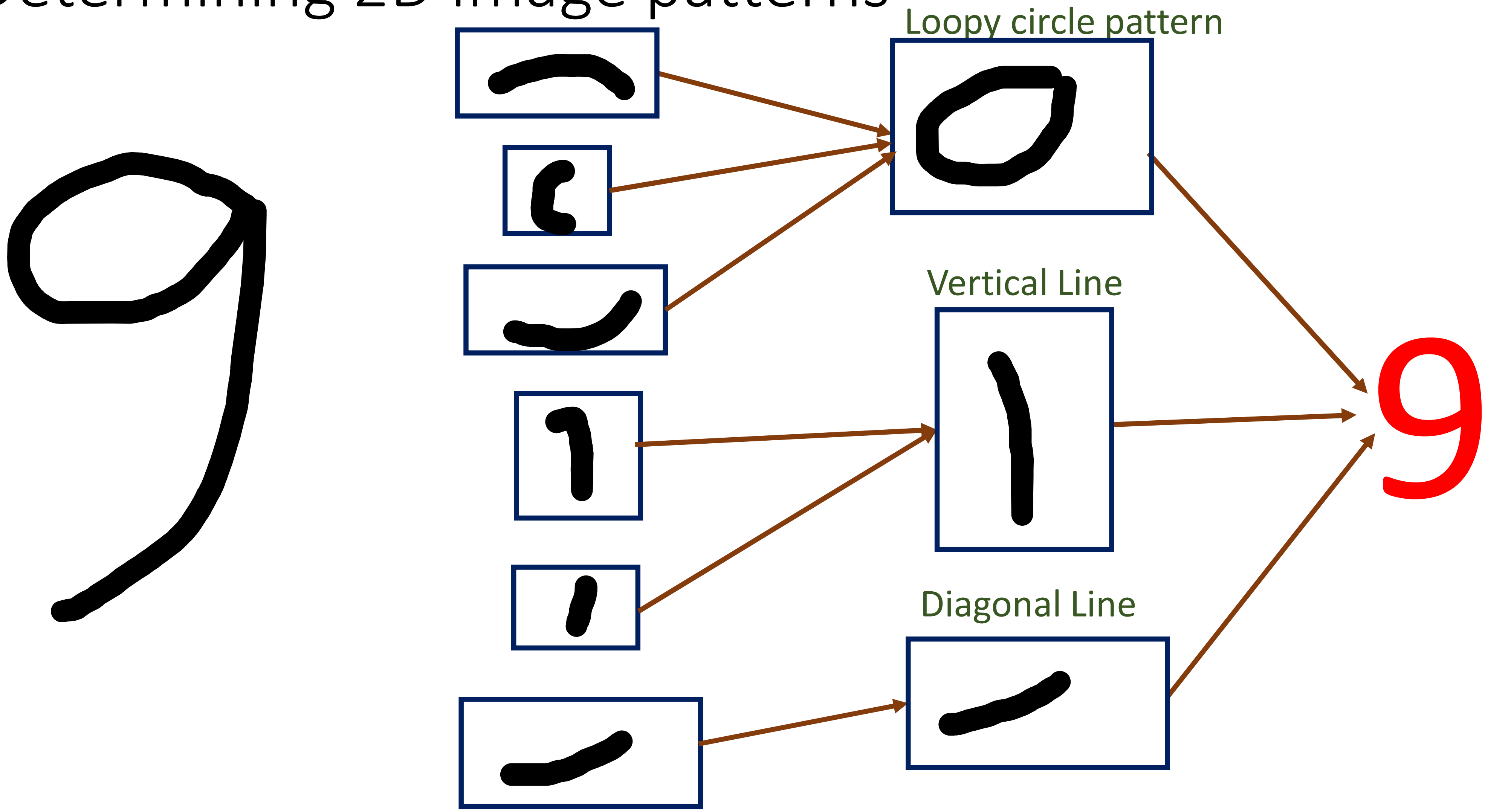
- Images are 2D.
- Assumption: Object image = combination of **2D image patterns**

hierarchical combination of **2D image patterns**

<u>Goal:</u>

1. Determine 2D image patterns [2D image patterns are smaller than image]
2. Map small image patterns → Large image patterns
3. Map larger image patterns → Target

Determining 2D image patterns

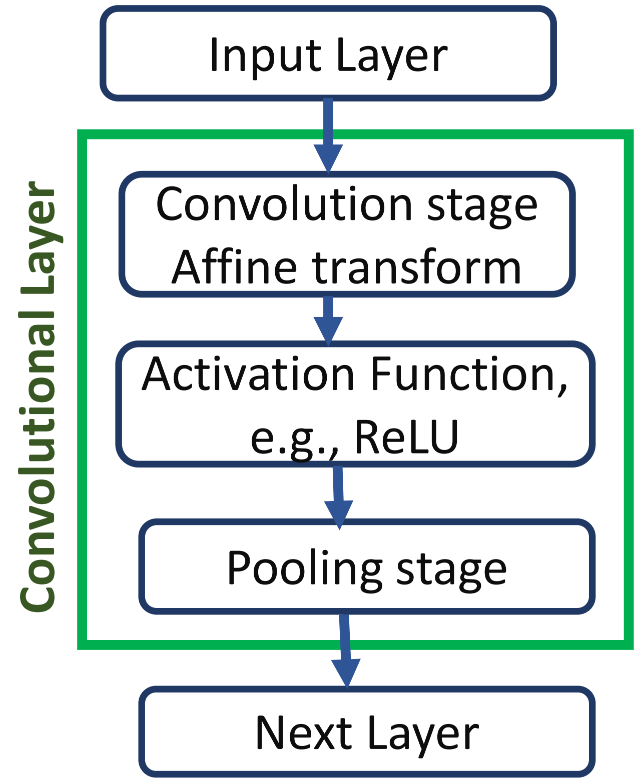


Convolutional Neural Networks (CNN)

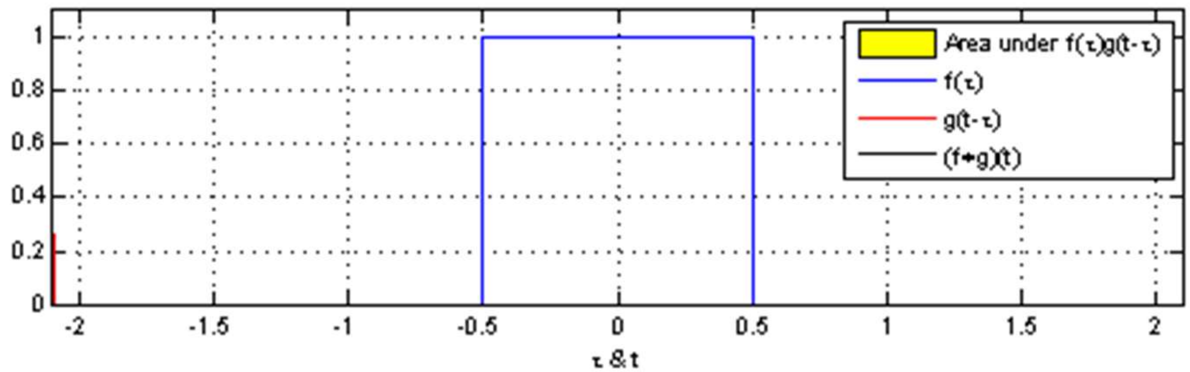
- ✓ CNN are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers

- Convolution is a mathematical operation on two functions (f and g), that produces a third function $(f * g)$ that expresses how the shape of one is modified by the other

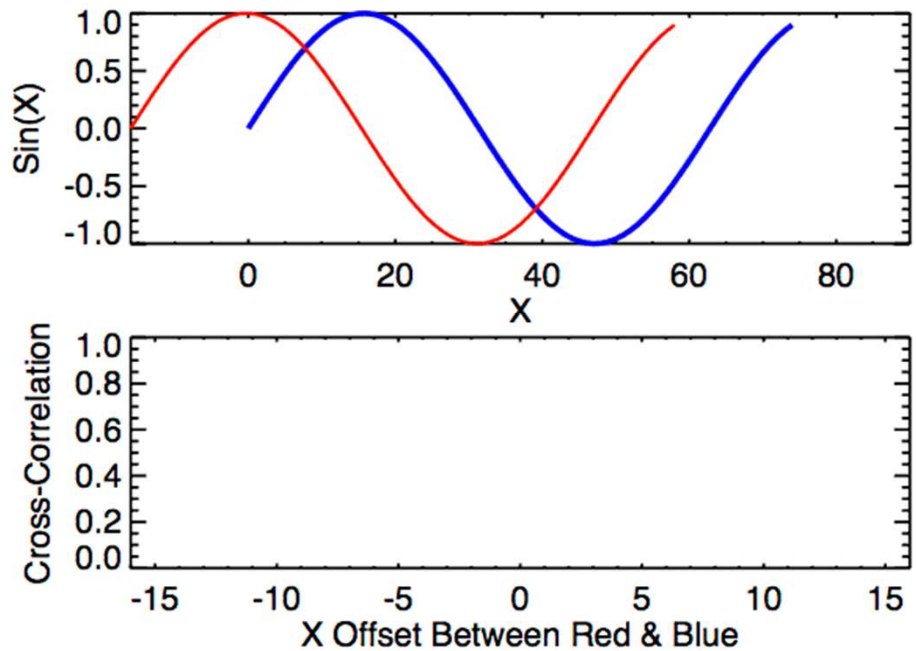
$$(\boldsymbol{f} * \boldsymbol{g})(\boldsymbol{t}) = \int_{-\infty}^{\infty} \boldsymbol{f}(\boldsymbol{\tau})\, \boldsymbol{g}(\boldsymbol{t}-\boldsymbol{\tau})\, \boldsymbol{d\tau}$$



Convolution



- Cross Correlation: It is a measure of similarity of two series as a function of the displacement of one relative to another.



Convolutional Stage

Step 1: Apply a **Filter** to the input Image

- A filter is just a smaller square that is commonly **N_{odd} pixels by N_{odd} pixels**

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

3 pixels by 3 pixels

1/9	1/9	1/9
1/9	-1/9	1/9
1/9	1/9	1/9

- ✓ Start with random values.
- ✓ The intensity of each pixel in the filter/kernel determined(learned) by Backpropagation.

To apply the Filter to the input image, we overlay the Filter onto the image

Convolutions

-1 1	1 1	1 1	1	-1
-1 1	1 -1	-1 1	1	-1
-1 1	1 1	1 1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

$* \; 1/9$

1	1	1
1	-1	1
1	1	1

$=$

-0.11		

$$(-1*1+1*1+1*1+(-1)*1+1*(-1)+(-1)*1+(-1)*1+1*1+1*1)/9=-1/9 = -\ 0.11$$

Convolutions

-1	1 1	1 1	1 1	-1
-1	1 1	-1 -1	1 1	-1
-1	1 1	1 1	1 1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

$*$ 1/9

1	1	1
1	-1	1
1	1	1

=

-0.11	1	

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

*

1/9	1/9	1/9
1/9	-1/9	1/9
1/9	1/9	1/9

=

-0.11	1	-0.11
-0.55	0.11	-0.33
-0.33	0.33	-0.33
-0.22	-0.11	-0.22
-0.33	-0.33	-0.33

Feature Map

- ✓ Filters are nothing but the feature detectors. A 2-D map is created where certain features appear in the input.
- ✓ Filters are location invariant. They can detect the loopy patterns in any location of the image.

Filters

- Different values of the filter matrix will produce different Feature Maps for the same input image -– **Multiple filters, multiple 'feature maps'**

- The Convolution operation captures the local dependencies in the original image

Padding: Valid and Same

- The size of the output is smaller than the input. To maintain the dimension of output as in input, we use padding. Padding is a process of adding zeros to the input matrix symmetrically

0	0	0	0	0	0	0
0	-1	1	1	1	-1	0
0	-1	1	-1	1	-1	0
0	-1	1	1	1	-1	0
0	-1	-1	-1	1	-1	0
0	-1	-1	-1	1	-1	0
0	-1	-1	1	-1	-1	0
0	-1	1	-1	-1	-1	0
0	0	0	0	0	0	0

*

1/9	1/9	1/9
1/9	-1/9	1/9
1/9	1/9	1/9

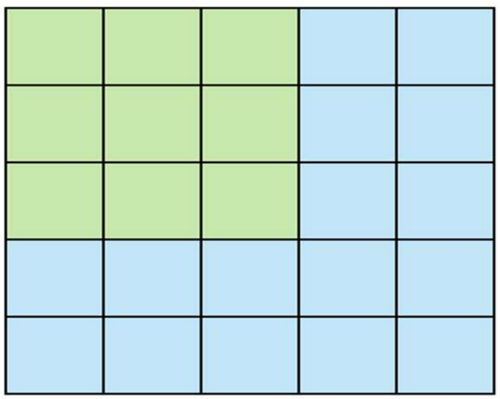
=

0.22	-0.22	0.22	-0.22	0.22
0.22	-0.11	1	-0.11	0.22
0	-0.55	0.11	-0.33	0.22
-0.22	-0.33	0.33	-0.33	0
-0.44	-0.22	-0.11	-0.22	-0.22
-0.22	-0.33	-0.33	-0.33	-0.22
0	0	-0.22	0	-0.22

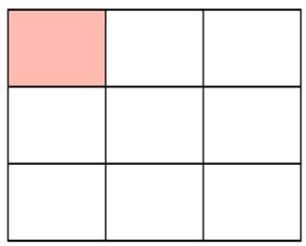
- Valid: no padding
- Same: Pad so that the output size is same as the input size

Stride

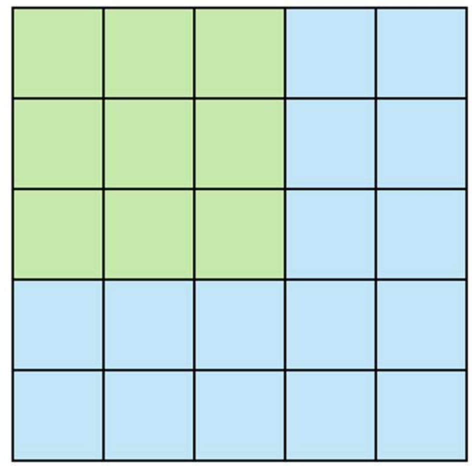
- Stride denotes how many steps we are moving in each step during convolution.



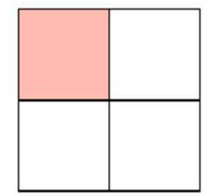
Stride 1



Feature Map



Stride 2

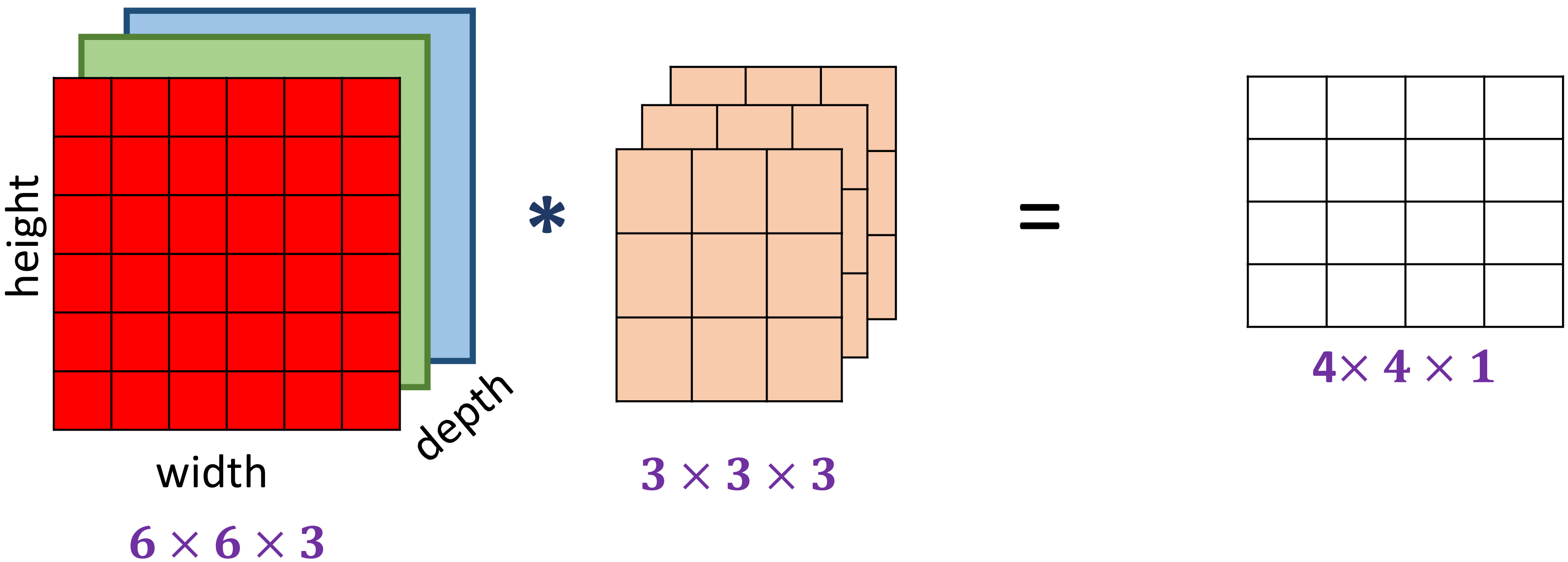


Feature Map

Solutions Obtained

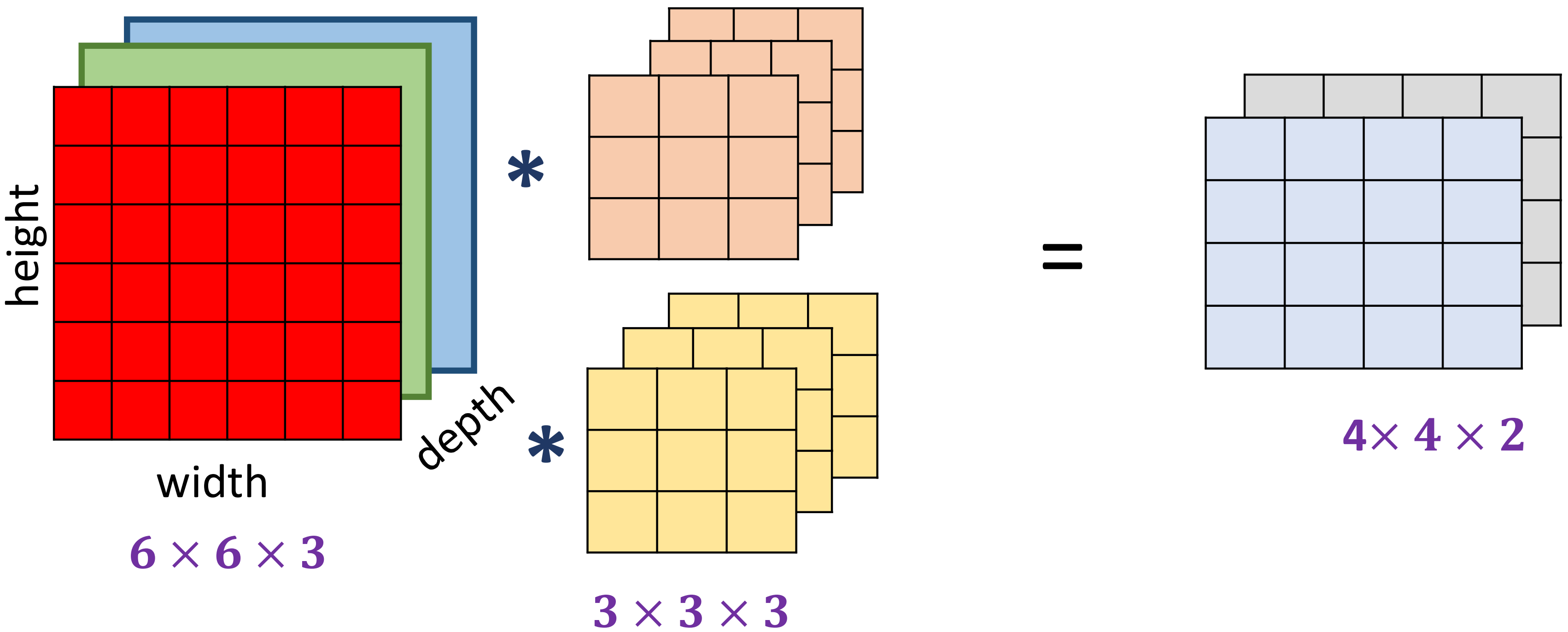
- In feed forward neural network, every output unit interacts with every input unit.
- CNN, typically have sparse connectivity (sparse weights). Each member of the kernel is used at every position of the input.
- This is accomplished by making the kernel smaller than the input.
- Reduce the number of parameters, where instead of learning a separate set of parameters for every location, we learn only one set – Parameter Sharing.
- Hyperparameters: Filter size, and number of filters.

Convolutions on RGB images



If you have 10 filters that are 3 x 3 x 3 in one layer of a neural network, how many parameters does that layer have?

Convolutions on RGB images



Summary of convolutions

f^l = filter size

p^l = padding

s^l =stride

$n_c^{[l]}$ = number of filters

Layer l is a convolution layer

- Input: $n_H^{[l-1]} \times n_w^{[l-1]} \times n_c^{[l-1]}$: $28 \times 28 \times 3$
- Each filter is: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$: $5 \times 5 \times 3$
- Weights: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$: $5 \times 5 \times 3 \times 10$
- Output Size: $n_H^{[l]} \times n_w^{[l]} \times n_c^{[l]}$

  : $24 \times 24 \times 10$ [with valid padding]

  : $28 \times 28 \times 10$ [with same padding]

$$n_{H/w}^{[l]} = \left\lfloor \frac{n_{H/w}^{[l-1]} + 2p^l - f^l}{s} + 1 \right\rfloor$$

$$= \left\lfloor \frac{28 + 2 \times 0 - 5}{1} + 1 \right\rfloor = 24$$

Convolutional Neural Networks (CNN)

❑ First Stage: Convolution

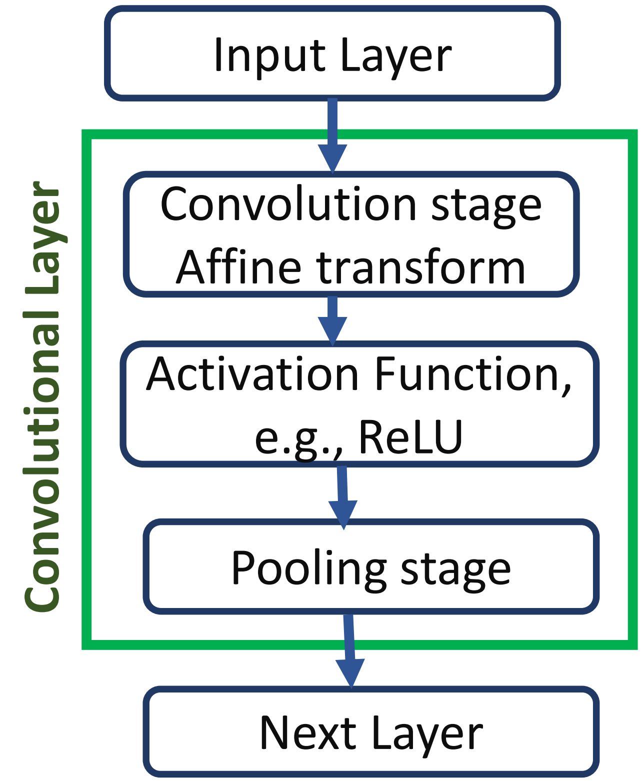
- The layer performs several convolutions in parallel to produce a set of pre-activations

❑ Second Stage: Non-linearity

- Each pre-activation is run through a nonlinear activation function

❑ Third Stage: Pooling

- Retaining the statistics of the data

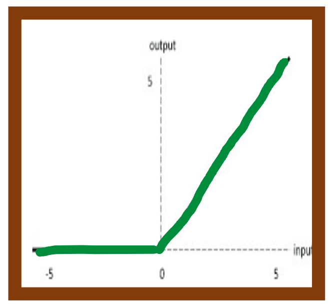


Activation Function

Feature Map

-0.11	1	-0.11
-0.55	0.11	-0.33
-0.33	0.33	-0.33
-0.22	-0.11	-0.22
-0.33	-0.33	-0.33

ReLU Activation



y = max(0, x)

Feature Map, Post ReLU

0	1	0
0	0.11	0
0	0.33	0
0	0	0
0	0	0

Pooling

- Pooling is used to reduce the spatial volume of input image after convolution
- **Motivation**: We care about presence of features, not their exact location !
  - Dimensionality Reduction
  - Prevents overfitting

0	1	0
0	0.11	0
0	0.33	0
0	0	0
0	0	0

Max Pooling

1	**1**
0.33	**0.33**
0.33	**0.33**
0	**0**

Average Pooling

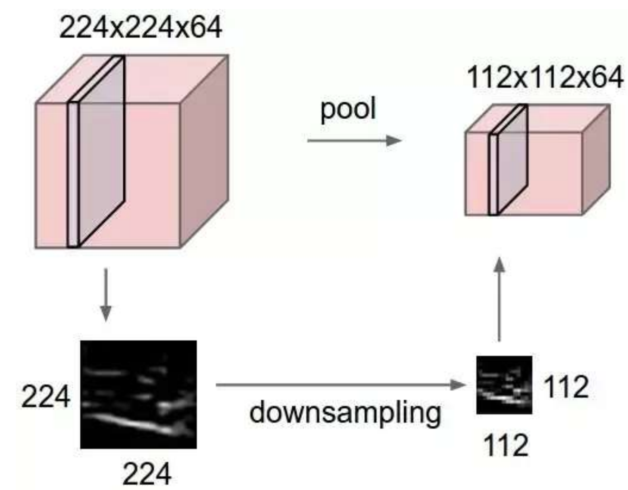
0.28	0.28
0.22	0.22
0.16	0.16
0	0

Summary of Pooling

Hyperparameters:

Note: Pooling does not learn any parameters

- ✓ f: filter size
- ✓ s: stride
- ✓ Max or Average pooling



$$n_H \times n_w \times n_c \xrightarrow{\text{Pooling}} \left[\frac{n_H - f}{s} + 1\right] \times \left[\frac{n_W - f}{s} + 1\right] \times n_c$$

Summary

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

*

Loopy pattern filter

1/9	1/9	1/9
1/9	-1/9	1/9
1/9	1/9	1/9

-0.11	1	-0.11
-0.55	0.11	-0.33
-0.33	0.33	-0.33
-0.22	-0.11	-0.22
-0.33	-0.33	-0.33

ReLU Activation

0	1	0
0	0.11	0
0	0.33	0
0	0	0
0	0	0

Max Pooling

1	1
0.33	0.33
0.33	0.33
0	0

Summary

1	1	1	-1	-1
1	-1	1	-1	-1
1	1	1	-1	-1
-1	-1	1	-1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1
1	-1	-1	-1	-1

*

Loopy pattern filter

1/9	1/9	1/9
1/9	-1/9	1/9
1/9	1/9	1/9

1	-0.11	-0.11
0.11	-0.33	0.33
0.33	-0.33	-0.33
-0.11	-0.55	-0.33
-0.55	-0.33	-0.55

ReLU Activation

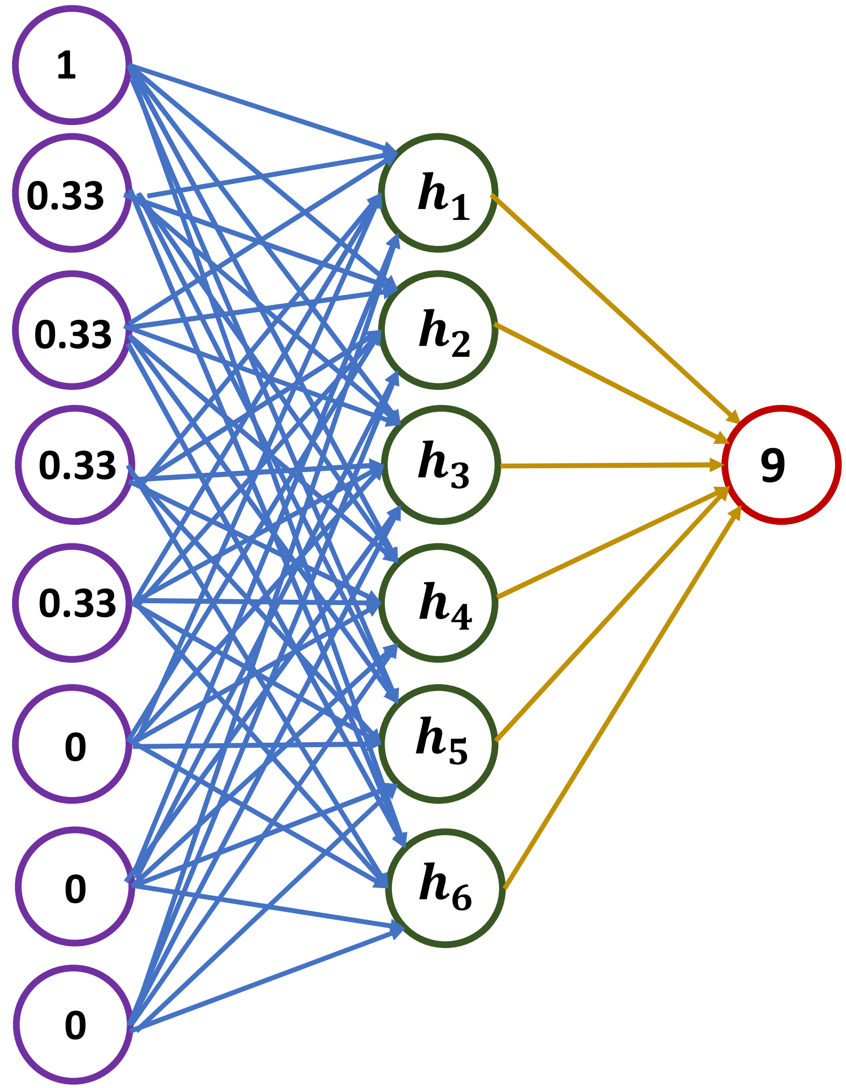
1	0	0
0.11	0	0.33
0.33	0	0
0	0	0
0	0	0

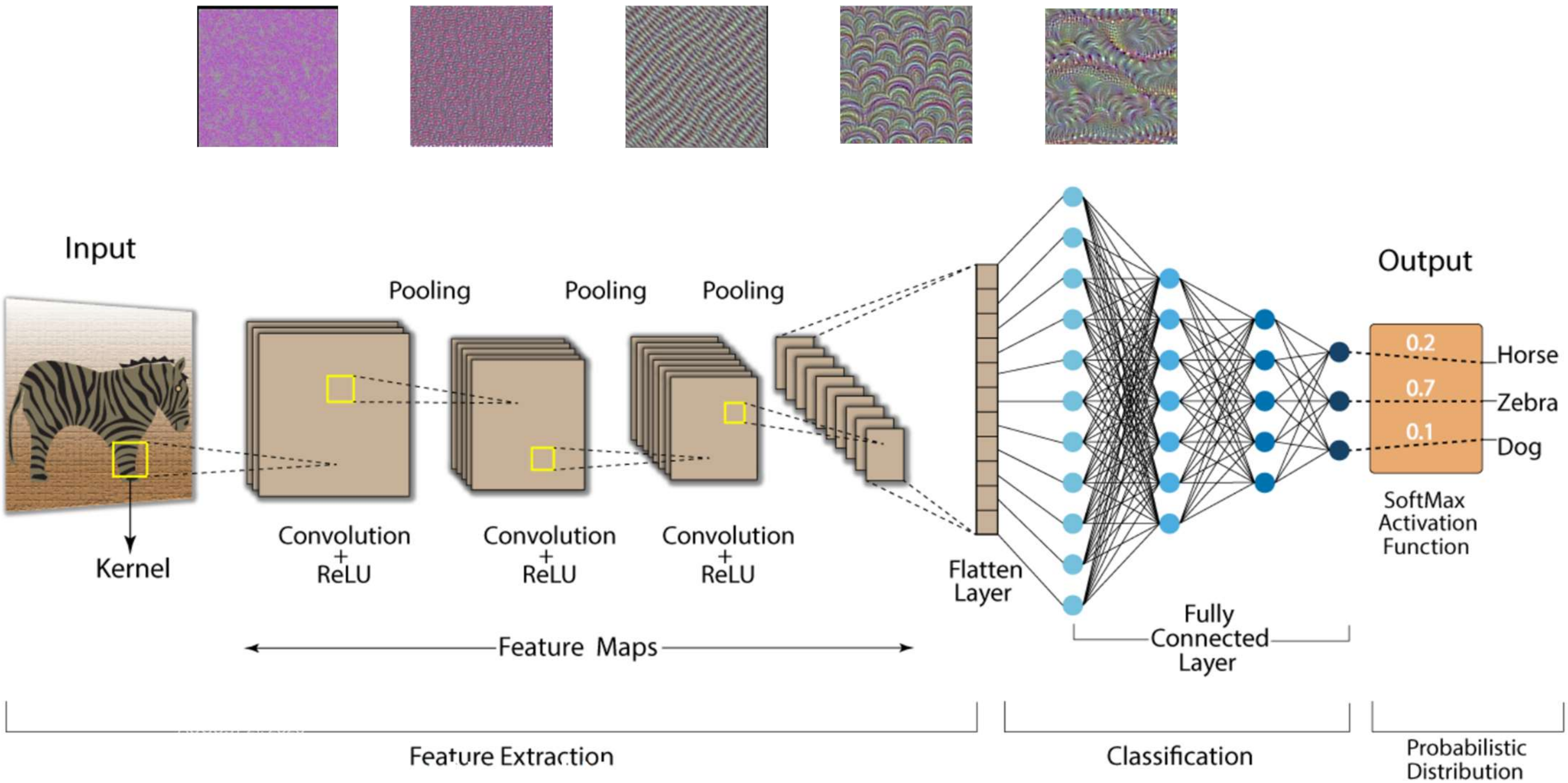
Max Pooling

1	0.33
0.33	0.33
0.33	0
0	0

Classification

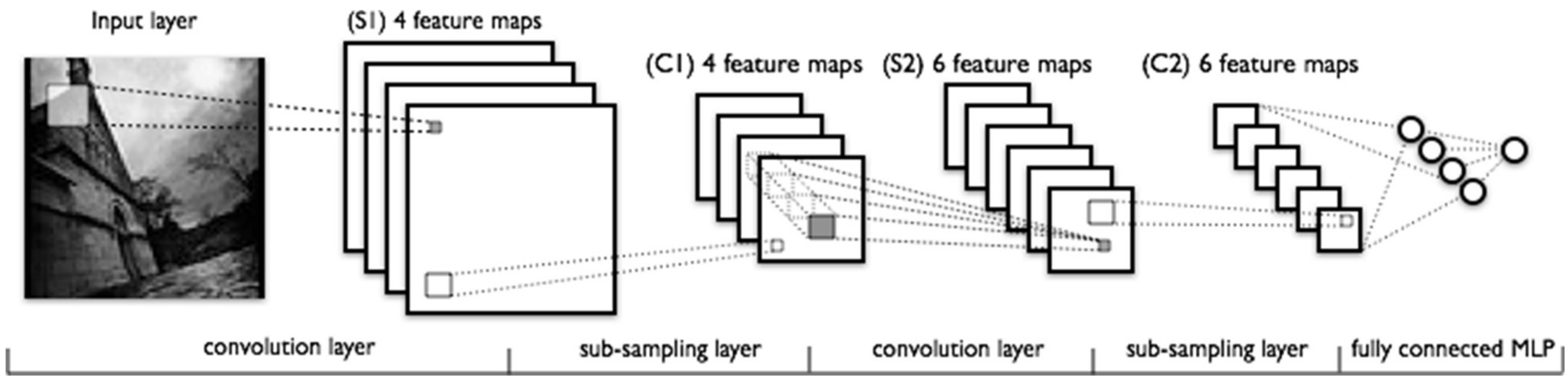
1	0.33
0.33	0.33
0.33	0
0	0





Convolution Neural Network Architecture

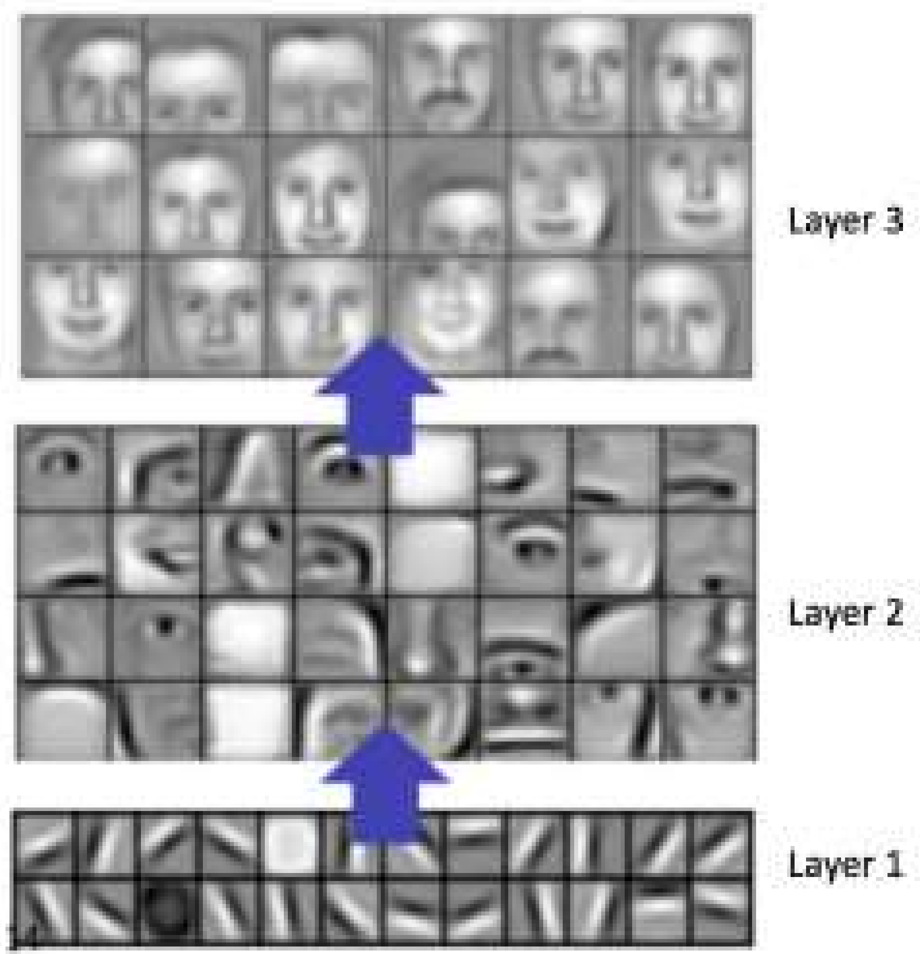
Convolutional Neural Networks (CNN)



- Learn features in input image through convolution
- Introduce non-linearity through activation function (real-world data is non-linear!)
- Reduce dimensionality and preserve spatial invariance with pooling
- Fully connected layer uses these features for classifying input image

Learned Features

Learn **hierarchy of features** directly from data (rather than hand-engineering them)



High Level features
Facial Structures

Mid Level features
Eyes, Ears, Nose

Low Level features
Edges, dark spots

http://web.eecs.umich.edu/~honglak/icml09-ConvolutionalDeepBeliefNetworks.pdf

Summary: Layer Patterns

INPUT → [[CONV -> RELU]*N → POOL?]*M → [FC → RELU]*K → CLASSIFIER

* -> Repetition

? -> Optional

<u>Common ConvNet Architectures:</u>

- INPUT → FC → CLASSIFIER
- INPUT → CONV → RELU → FC → CLASSIFIER
- INPUT → [CONV → RELU → POOL]*2 → FC → RELU → CLASSIFIER
- INPUT → [CONV → RELU → CONV → RELU → POOL]*3 → [FC → RELU]*2 → CLASSIFIER