

Dhirubhai Ambani University

(Formerly known as DA-IICT)

Topic: Deep Learning: CNN

Course: Programming Lab

Course Code- PC503

Dr. Ankit Vijayvargiya

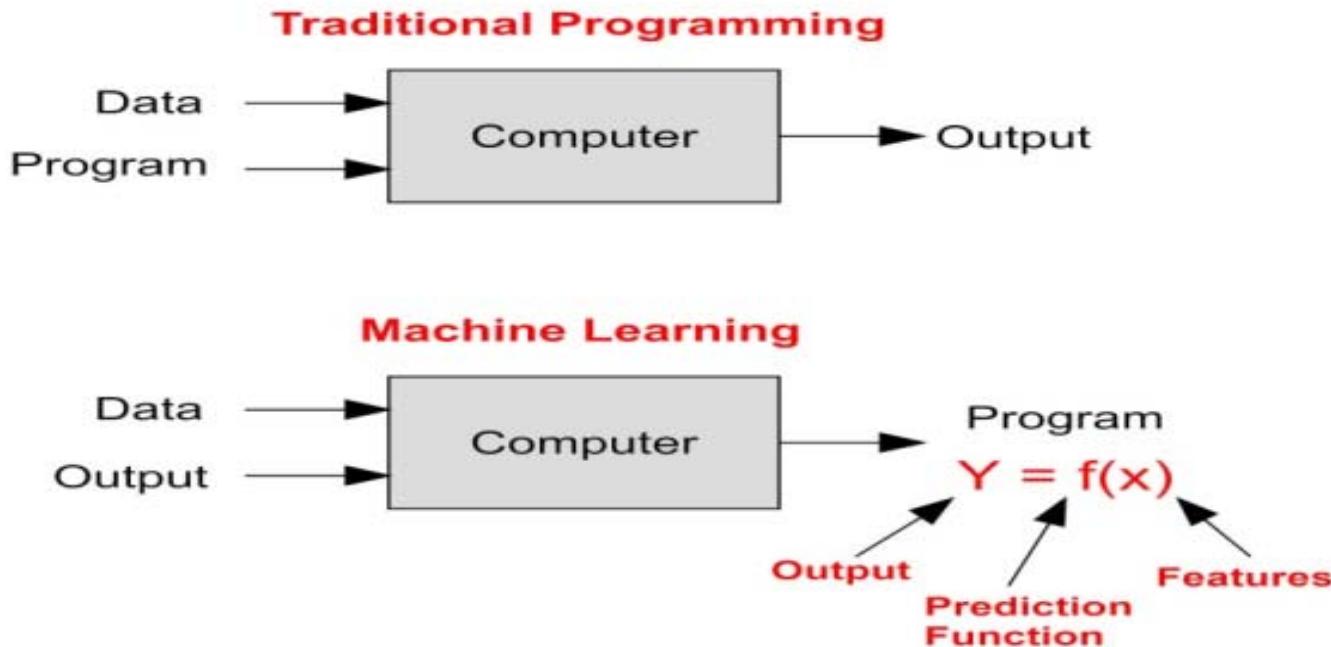
Assistant Professor

Room No. 4205, Faculty Block 4

Email: ankit_Vijayvargiya[at]dau.ac.in

Phone: 079-68261628(O), 7877709590(M)

Machine Learning (ML)



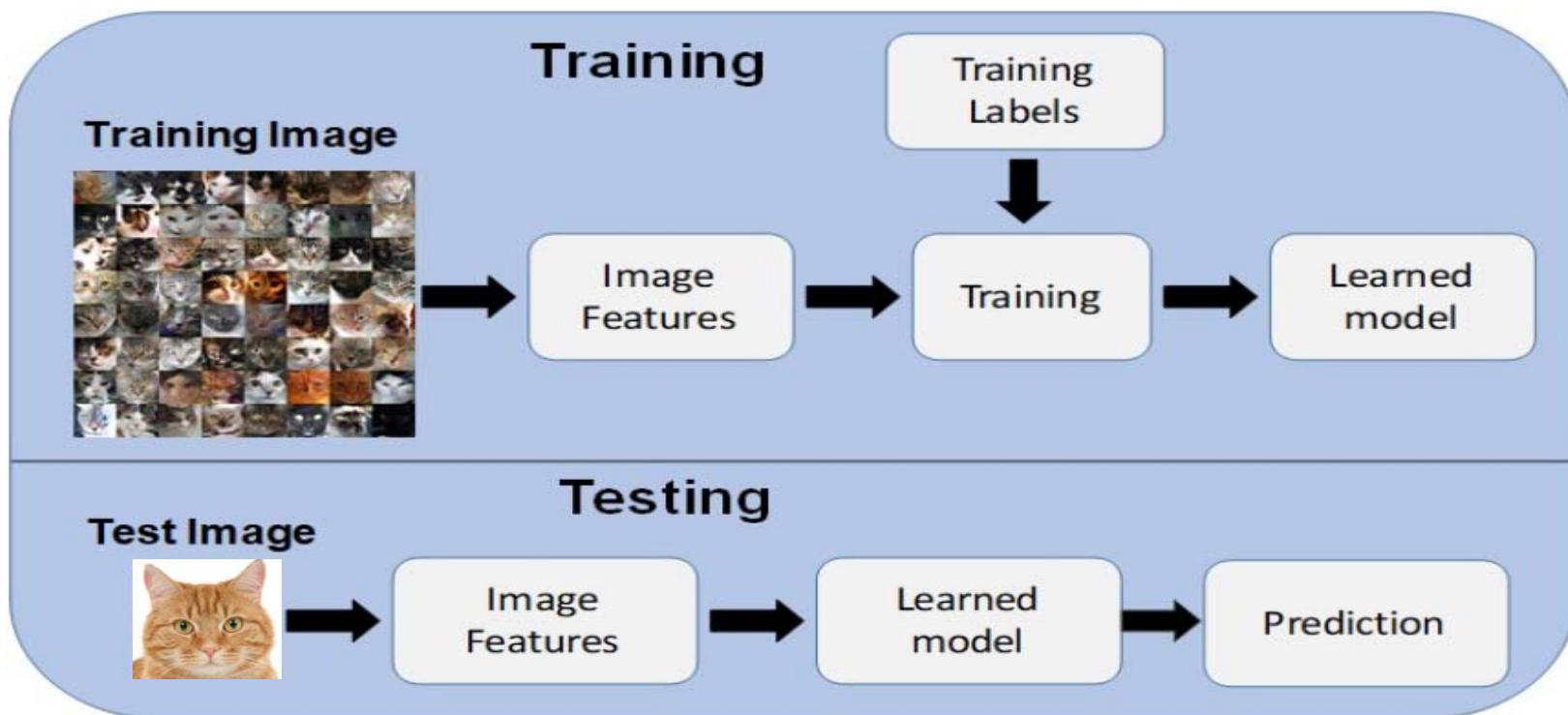
- Training: given a training set of labeled examples $(X_1, Y_1), \dots, (X_n, Y_n)$, estimate the prediction function $f(x)$ by minimizing the prediction error on the training set.
- Testing: apply f to a never before seen test example x and output the predicted value $Y = f(x)$.

Why ML?????????

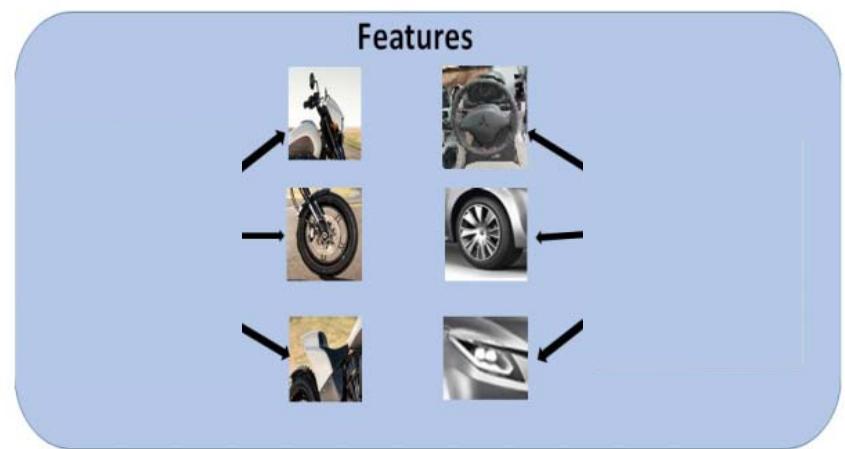
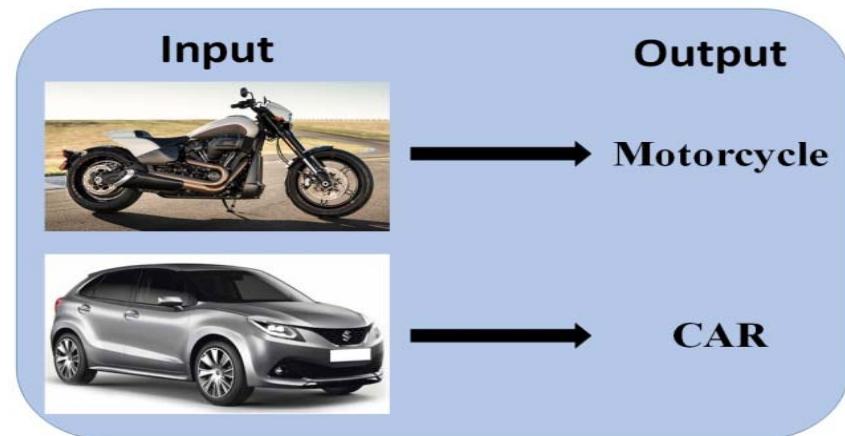
- Large amount of data available!
- Analysis of this data is not possible manually
- ML Algorithms manage the large amount of data
- Solve complex problems!



Steps of ML



Features



But still classification is not easy.....

Chihuahua or Muffin



Sheepdog or Mop



Barn owl or Apple

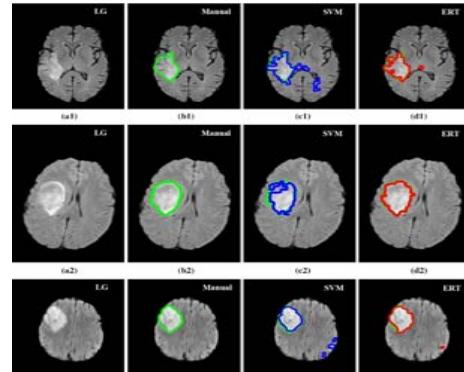


Labradoodle or Fried-chicken

Applications



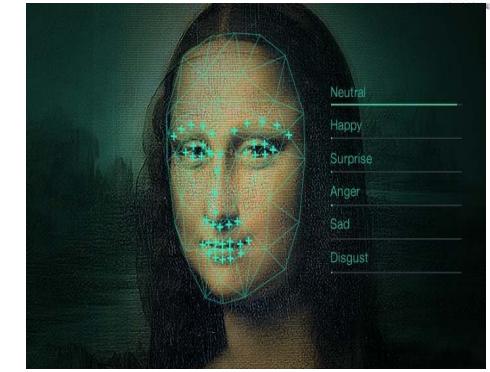
Healthcare



Disease Diagnosis



Robotics



Sentiment Analysis



Self Driving Cars



Price Prediction

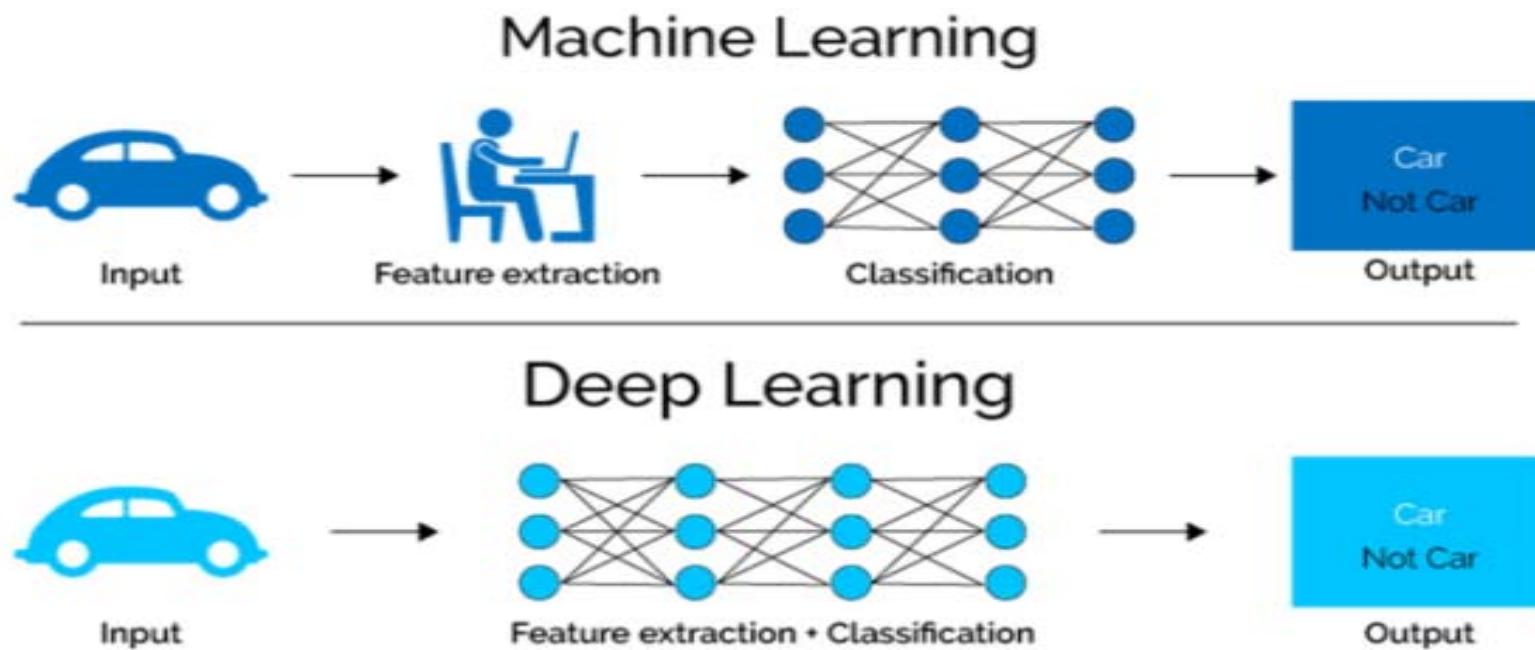


Engineering



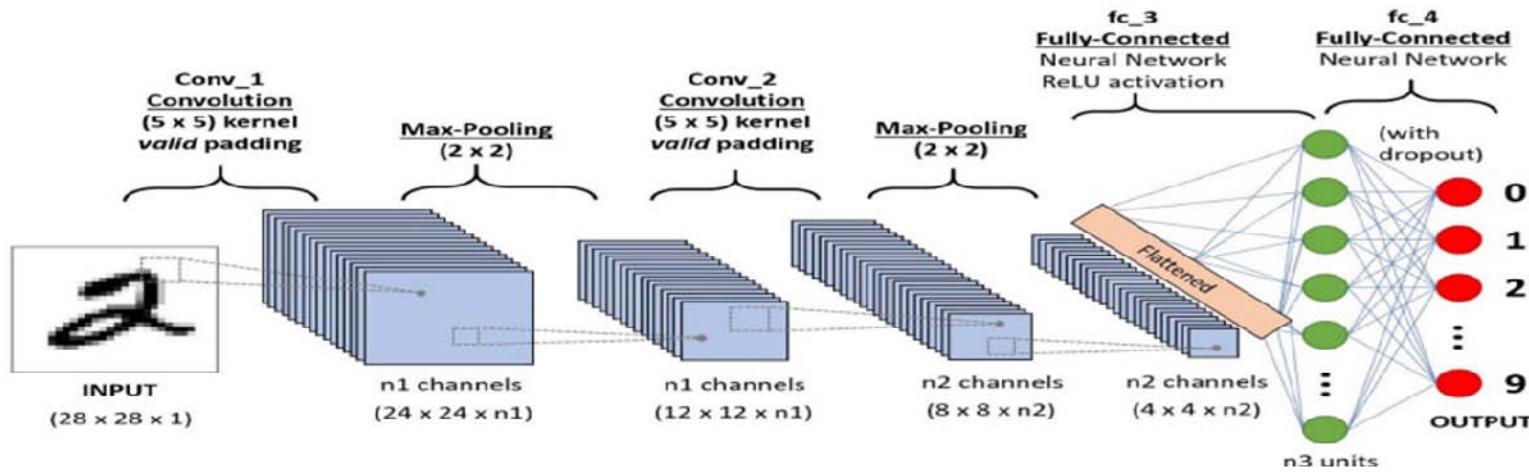
Digital Marketing

What does Deep Learning offer....

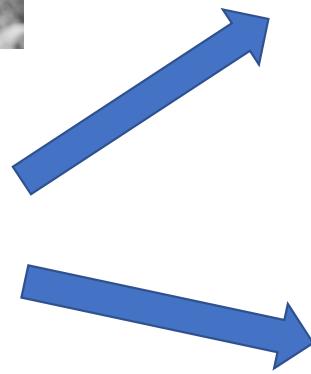


Convolutional Neural Network: Image Classification

- ❖ How the Computer Reads an Image?
- ❖ What is the Fully Connected Network?
- ❖ Why not Fully Connected Networks for Image Recognition?
- ❖ What is Convolutional Neural Network?
- ❖ How do Convolutional Neural Networks work?



How computer reads an image

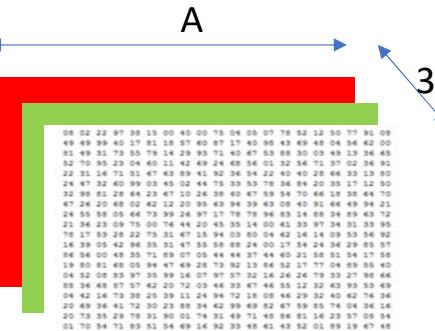


```

08 02 22 97 38 15 00 40 00 75 06 05 07 78 52 12 50 77 91 08
49 49 99 40 17 81 18 57 40 87 17 40 98 43 49 45 04 56 42 00
81 49 31 73 55 79 14 29 93 71 40 67 58 88 30 03 49 23 36 65
52 70 95 23 04 60 11 42 69 24 68 54 01 32 54 71 37 02 36 91
22 31 16 71 51 47 63 89 41 92 36 54 22 40 40 28 64 33 13 80
24 47 32 60 99 03 45 09 44 75 33 53 78 36 84 20 35 27 12 50
32 98 81 28 64 23 47 10 26 38 40 67 59 34 70 64 18 38 64 70
67 26 20 48 02 62 12 20 95 43 96 39 43 08 40 93 44 49 94 21
24 55 58 05 64 73 99 26 97 17 78 78 94 83 14 88 34 89 63 72
21 34 23 09 75 00 76 44 20 45 35 14 00 41 33 97 34 31 33 95
78 17 53 25 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92
14 39 05 42 94 35 31 47 55 58 88 20 00 17 54 24 36 29 85 57
86 54 00 48 35 71 89 07 05 44 44 37 44 40 21 58 51 54 17 58
19 80 81 68 05 94 47 49 28 73 92 13 84 52 17 77 04 89 55 40
04 52 08 83 97 35 99 14 07 97 57 32 14 26 78 33 27 93 66
88 34 48 87 57 62 20 72 03 46 33 47 46 55 12 32 63 93 53 69
04 42 26 73 38 25 39 11 24 94 72 18 08 46 29 32 40 42 76 36
20 49 36 41 72 30 23 88 34 42 99 69 82 47 59 85 74 04 36 16
20 73 35 29 78 31 90 01 74 31 49 71 68 86 81 16 23 97 05 54
01 70 54 71 83 51 54 69 14 92 33 48 61 43 52 01 89 19 67 48
    
```



Channels

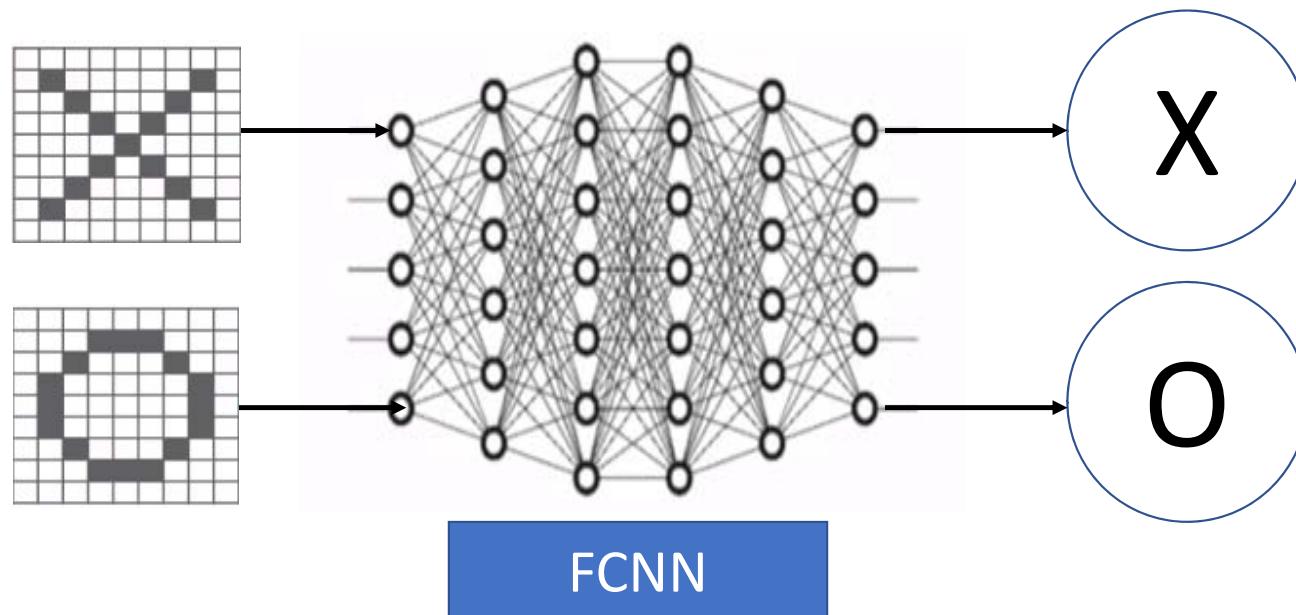


Pixels value of RGB

B: Rows | A: Columns | 3:

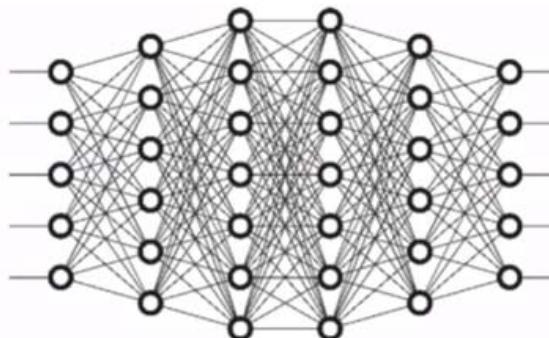
219

Fully Connected Neural Network



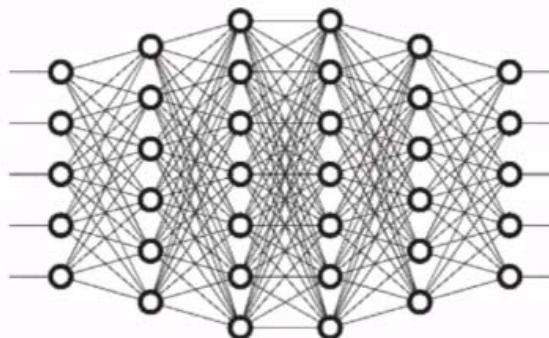
Why not fully connected networks

Image with
 $28 \times 28 \times 3$
Pixels
(MNIST)



*Number of weights in
the first hidden layer
will be 2352*

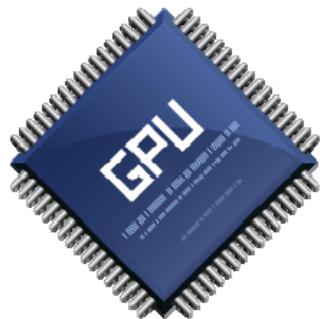
Image with
 $200 \times 200 \times 3$
pixels



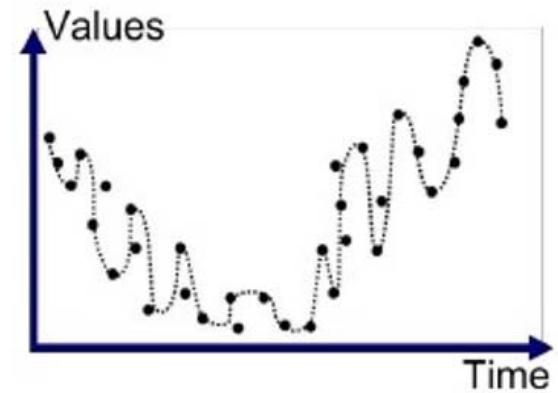
*Number of weights in
the first hidden layer
will be 120,000*

Why not fully connected networks

High computational Resources

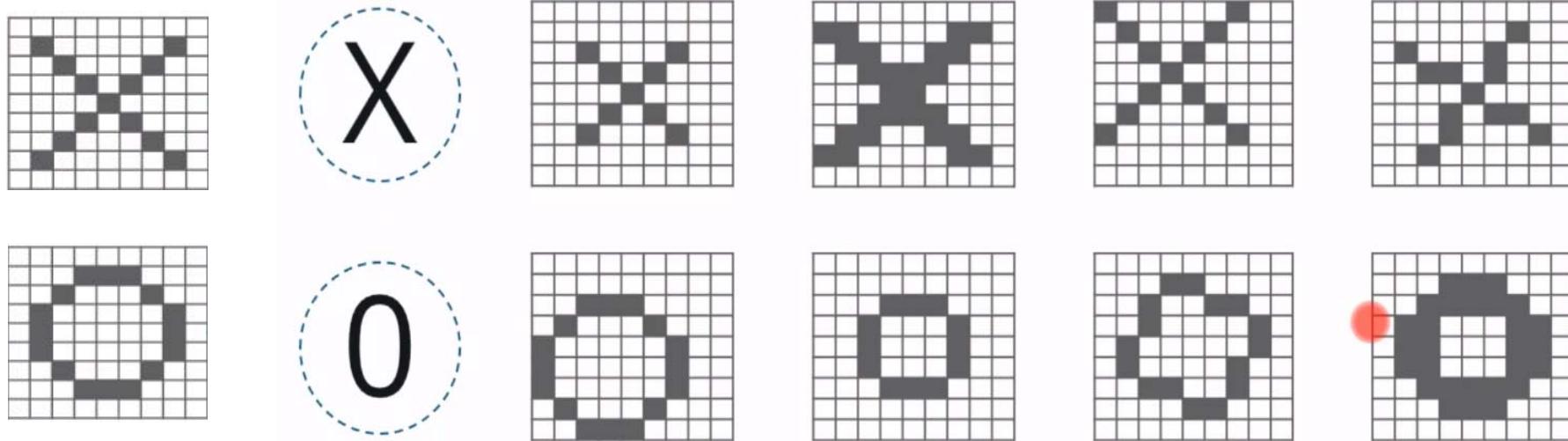


Overfitting



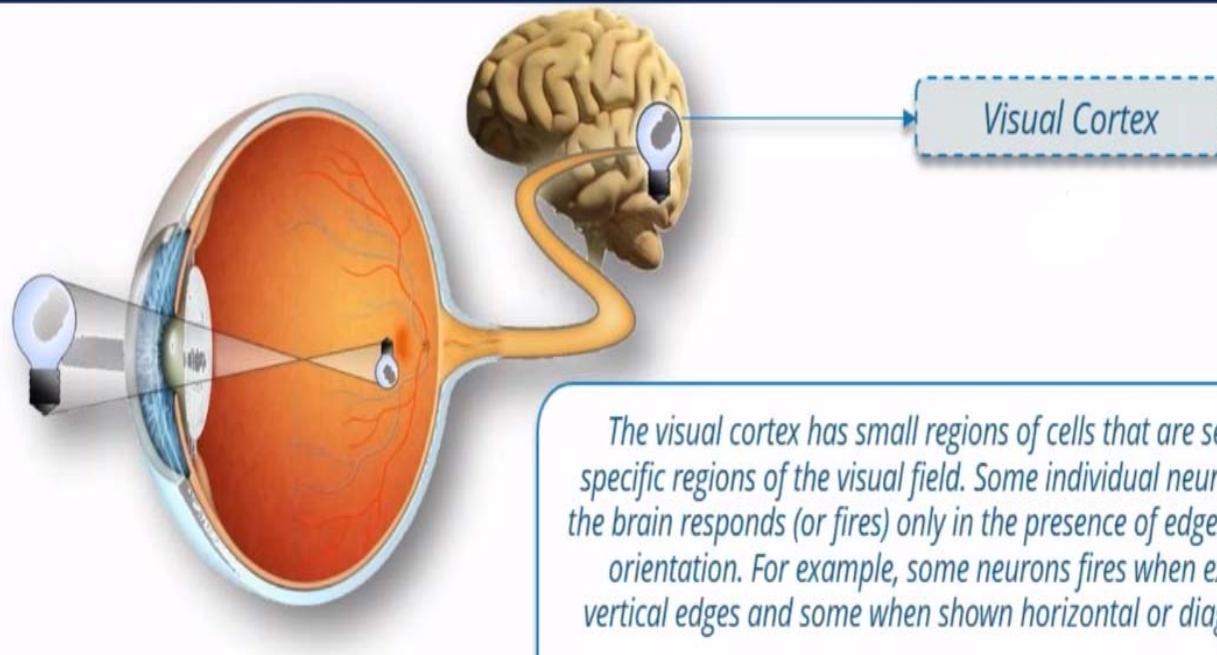
Why not fully connected networks

Here we will have some problems because X and O images won't always have the same images. There can be certain deformations. Consider the diagram below



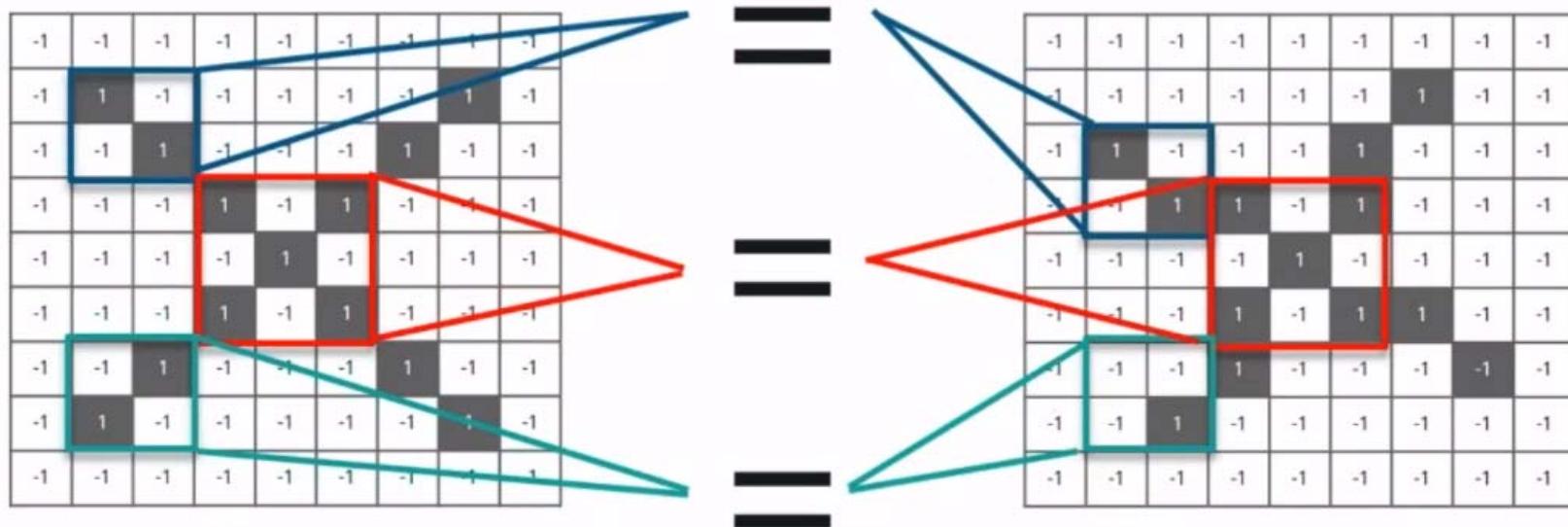
What is Convolutional Neural Network

Convolutional Neural Network (CNN) is a type of feed-forward artificial neural network in which the Connectivity pattern between its neurons is inspired by the organization of the animal visual cortex.



How CNN works????

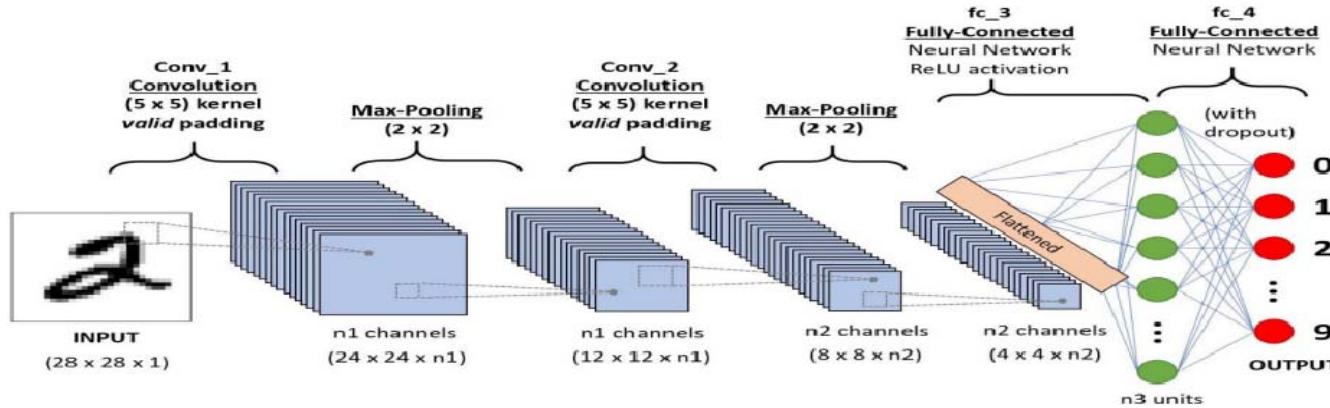
CNN compares the images piece by piece. The pieces that it looks for are called features. By finding rough feature matches in roughly the same position in two images, CNN gets a lot better at seeing similarity than the whole-image matching scheme.



Layers of CNN

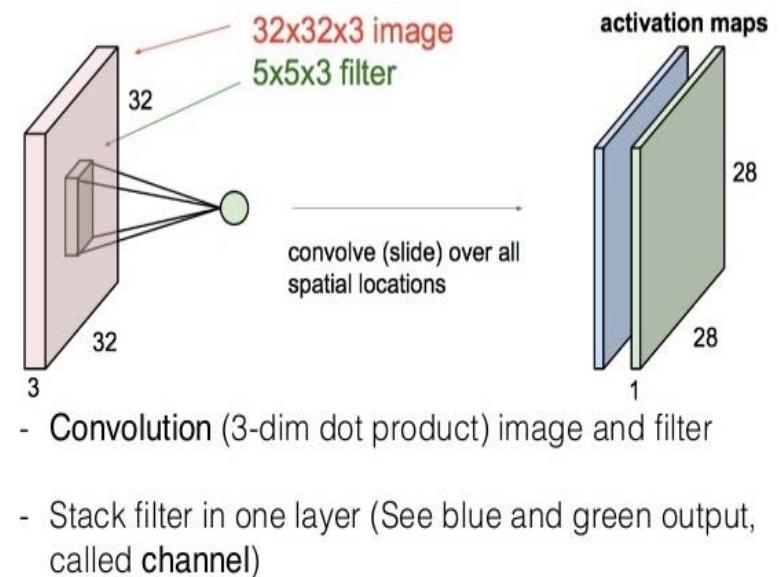
A Convolutional Neural Network consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of:

- Convolutional Layer
- Pooling Layer
- Flattening
- Fully Connected Layer



Convolutional Layer

- The primary purpose of the convolution layer is to extract the features from the input image.
- They detect patterns regardless of their location in the input image.
- Reduce the number of parameters compared to fully connected layers by sharing weights (filters).
- Focus on small regions of the image, preserving spatial relationships between pixels.
- Convolutional layers can handle high-dimensional input data like large images effectively.



Convolutional Layer

Convolution operation on a $M \times N \times 3$ image matrix with a $3 \times 3 \times 3$ Kernel



| | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... |
| 0 | 156 | 155 | 156 | 158 | 158 | 158 | ... |
| 0 | 153 | 154 | 157 | 159 | 159 | 159 | ... |
| 0 | 149 | 151 | 155 | 158 | 159 | 159 | ... |
| 0 | 146 | 146 | 149 | 153 | 158 | 158 | ... |
| 0 | 145 | 143 | 143 | 148 | 158 | 158 | ... |
| ... | ... | ... | ... | ... | ... | ... | ... |

Input Channel #1 (Red)

| | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... |
| 0 | 167 | 166 | 167 | 169 | 169 | 169 | ... |
| 0 | 164 | 165 | 168 | 170 | 170 | 170 | ... |
| 0 | 160 | 162 | 166 | 169 | 170 | 170 | ... |
| 0 | 156 | 156 | 159 | 163 | 168 | 168 | ... |
| 0 | 155 | 153 | 153 | 158 | 168 | 168 | ... |
| ... | ... | ... | ... | ... | ... | ... | ... |

Input Channel #2 (Green)

| | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... |
| 0 | 163 | 162 | 163 | 165 | 165 | 165 | ... |
| 0 | 160 | 161 | 164 | 166 | 166 | 166 | ... |
| 0 | 156 | 158 | 162 | 165 | 166 | 166 | ... |
| 0 | 155 | 155 | 158 | 162 | 167 | 167 | ... |
| 0 | 154 | 152 | 152 | 157 | 167 | 167 | ... |
| ... | ... | ... | ... | ... | ... | ... | ... |

Input Channel #3 (Blue)

| | | |
|----|----|----|
| -1 | -1 | 1 |
| 0 | 1 | -1 |
| 0 | 1 | 1 |

Kernel Channel #1



308

| | | |
|---|----|----|
| 1 | 0 | 0 |
| 1 | -1 | -1 |
| 1 | 0 | -1 |

Kernel Channel #2



-498

| | | |
|---|----|---|
| 0 | 1 | 1 |
| 0 | 1 | 0 |
| 1 | -1 | 1 |

Kernel Channel #3



164

+

$+ 1 = -25$

Bias = 1

Output

| | | | |
|-----|-----|-----|-----|
| -25 | | | ... |
| | | | ... |
| | | | ... |
| | | | ... |
| ... | ... | ... | ... |

228

Convolutional Layer

Convolution operation on a $M \times N \times 3$ image matrix with a $3 \times 3 \times 3$ Kernel

| | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... |
| 0 | 156 | 155 | 156 | 158 | 158 | 158 | ... |
| 0 | 153 | 154 | 157 | 159 | 159 | 159 | ... |
| 0 | 149 | 151 | 155 | 158 | 159 | 159 | ... |
| 0 | 146 | 146 | 149 | 153 | 158 | 158 | ... |
| 0 | 145 | 143 | 143 | 148 | 158 | 158 | ... |
| ... | ... | ... | ... | ... | ... | ... | ... |

Input Channel #1 (Red)

| | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... |
| 0 | 167 | 166 | 167 | 169 | 169 | 169 | ... |
| 0 | 164 | 165 | 168 | 170 | 170 | 170 | ... |
| 0 | 160 | 162 | 166 | 169 | 170 | 170 | ... |
| 0 | 156 | 156 | 159 | 163 | 168 | 168 | ... |
| 0 | 155 | 153 | 153 | 158 | 168 | 168 | ... |
| ... | ... | ... | ... | ... | ... | ... | ... |

Input Channel #2 (Green)

| | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... |
| 0 | 163 | 162 | 163 | 165 | 165 | 165 | ... |
| 0 | 160 | 161 | 164 | 166 | 166 | 166 | ... |
| 0 | 156 | 158 | 162 | 165 | 166 | 166 | ... |
| 0 | 155 | 155 | 158 | 162 | 167 | 167 | ... |
| 0 | 154 | 152 | 152 | 157 | 167 | 167 | ... |
| ... | ... | ... | ... | ... | ... | ... | ... |

Input Channel #3 (Blue)

| | | |
|----|----|----|
| -1 | -1 | 1 |
| 0 | 1 | -1 |
| 0 | 1 | 1 |

Kernel Channel #1



310

+

| | | |
|---|----|----|
| 1 | 0 | 0 |
| 1 | -1 | -1 |
| 1 | 0 | -1 |

Kernel Channel #2



-170

| | | |
|---|----|---|
| 0 | 1 | 1 |
| 0 | 1 | 0 |
| 1 | -1 | 1 |

Kernel Channel #3

+

325

+ 1 = 466

Bias = 1

| | | | | | |
|-----|-----|-----|-----|-----|-----|
| -25 | 466 | | | | ... |
| | | | | | ... |
| | | | | | ... |
| | | | | | ... |
| ... | ... | ... | ... | ... | ... |

Output

Problem Associated with the Convolutional Layer

There are two main problems associated with convolutional layer:

- Reducing the dimension of image: Here we use only one convolutional layer for convolution but if we pass the image through the network and get convolved by more filters then the size of the image will be reduced and may be useless.
- The edges features role is very less comparing to the other features.

Zero Padding

- Zero padding is a technique that allows us to preserve the original input size.
- In the zero padding we add a border of pixel all with values zero around the edges of the input images.
- When we use the zero padding then the dimension of the input size will be increase.
- After convolution, the dimension of the image will be same as the input image.

| | | | | | | |
|---|---|---|---|---|---|---|
| o | o | o | o | o | o | o |
| o | 1 | 1 | 1 | 0 | 0 | 0 |
| o | 0 | 1 | 1 | 1 | 0 | 0 |
| o | 0 | 0 | 1 | 1 | 1 | 0 |
| o | 0 | 1 | 1 | 1 | 0 | 0 |
| o | 0 | 1 | 1 | 0 | 0 | 0 |
| o | o | o | o | o | o | o |

Let us take a example of input image size is 5X5. If we add zero around the edges of the input image then the input image size will be 7X7. Feature detector size is 3X3. Then the output image size will be same as input image size(5X5).

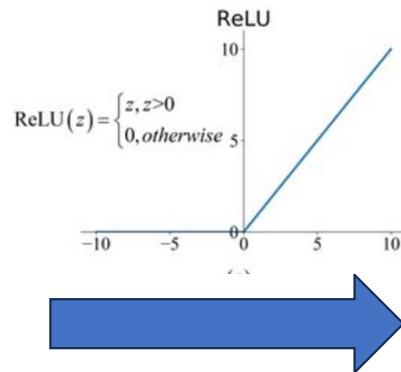
$$R = \frac{M - P}{Stride} + 1 = \frac{7 - 3}{1} + 1 = 5 \quad (1)$$

$$S = \frac{N - Q}{Stride} + 1 = \frac{7 - 3}{1} + 1 = 5 \quad (2)$$

Activation (non-linear) Function

- Replace all negative pixel values in the feature map by zero.
- The purpose of ReLu is to introduce non-linearity in CNN, since most of the real world data would be non-linear
- Other non-linear functions such as tanh (-1,1) or sigmoid (0,1), can also be used instead of ReLU (0,input).

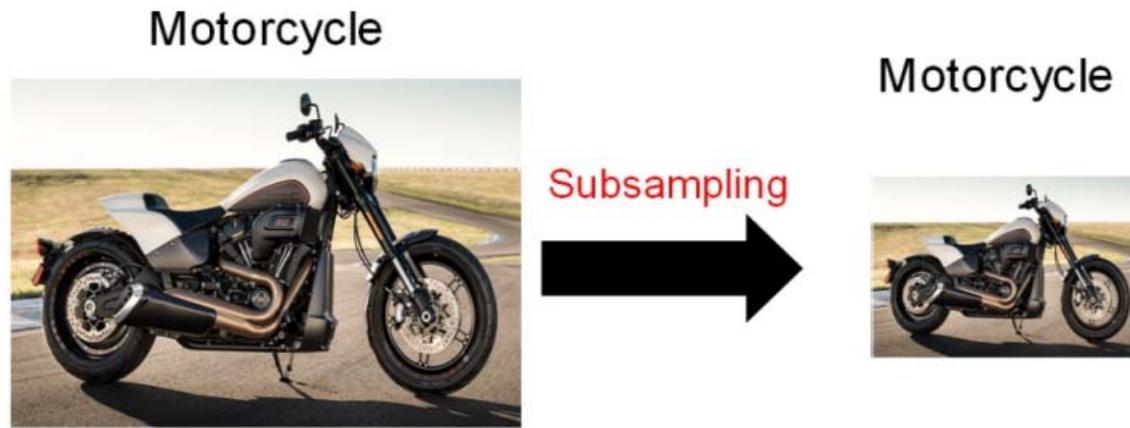
| | | |
|------|------|------|
| -249 | -91 | -37 |
| 250 | -134 | 101 |
| 27 | 61 | -153 |



| | | |
|-----|----|-----|
| 0 | 0 | 0 |
| 250 | 0 | 101 |
| 27 | 61 | 0 |

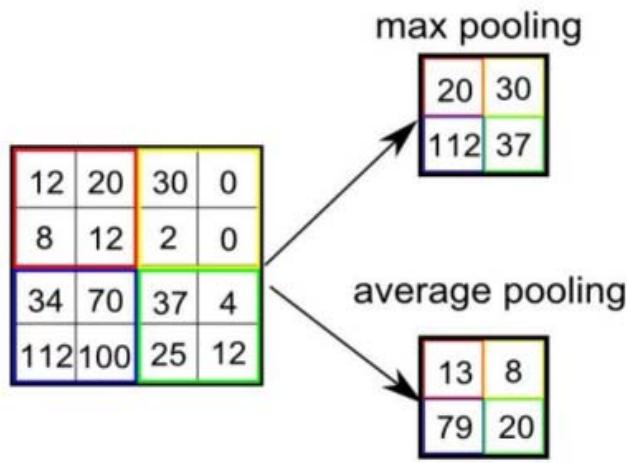
Pooling Layer

- Pooling layer function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network.
- Pooling layer operates on each feature map independently.



Subsampling pixels will not change the object.

Pooling Layer



Different Types of Pooling:

- Max Pooling
- mean Pooling
- Sum Pooling
- Min Pooling
- Median Pooling

The most common approach used in pooling is Max Pooling.

Pooling Layer

Output after passing through pooling layer.

The basic role of the pooling layer is to shrink the size of our image matrix.

Here we have converted a 7x7 matrix to a 4x4 matrix.

Since we took 3 features in the beginning, we have 3 outputs after the pooling layer.

Now next we have to stack up all the layers.

| | | | | | | |
|------|------|------|------|------|------|------|
| 0.77 | 0 | 0.11 | 0.33 | 0.55 | 0 | 0.33 |
| 0 | 1.00 | 0 | 0.33 | 0 | 0.11 | 0 |
| 0.11 | 0 | 1.00 | 0 | 0.11 | 0 | 0.55 |
| 0.33 | 0.33 | 0 | 0.55 | 0 | 0.33 | 0.33 |
| 0.55 | 0 | 0.11 | 0 | 1.00 | 0 | 0.11 |
| 0 | 0.11 | 0 | 0.33 | 0 | 1.00 | 0 |
| 0.33 | 0 | 0.55 | 0.33 | 0.11 | 0 | 1.77 |

| | | | | | | |
|------|------|------|------|------|------|------|
| 0.33 | 0 | 0.11 | 0 | 0.11 | 0 | 0.33 |
| 0 | 0.55 | 0 | 0.33 | 0 | 0.55 | 0 |
| 0.11 | 0 | 0.55 | 0 | 0.55 | 0 | 0.11 |
| 0 | 0.33 | 0 | 1.00 | 0 | 0.33 | 0 |
| 0.11 | 0 | 0.55 | 0 | 0.55 | 0 | 0.11 |
| 0 | 0.55 | 0 | 0.33 | 0 | 0.55 | 0 |
| 0.33 | 0 | 0.11 | 0 | 0.11 | 0 | 0.33 |

| | | | | | | |
|------|------|------|------|------|------|------|
| 0.33 | 0 | 0.11 | 0.33 | 0.11 | 0 | 0.33 |
| 0 | 0.11 | 0 | 0.33 | 0 | 1.00 | 0 |
| 0.33 | 0 | 0.11 | 0 | 1.00 | 0 | 0.11 |
| 0.33 | 0.33 | 0 | 0.55 | 0 | 0.33 | 0.33 |
| 0.77 | 0 | 1.00 | 0 | 0.11 | 0 | 0.33 |
| 0 | 1.00 | 0 | 0.33 | 0 | 0.11 | 0 |
| 0.33 | 0 | 0.11 | 0.33 | 0.33 | 0 | 0.33 |



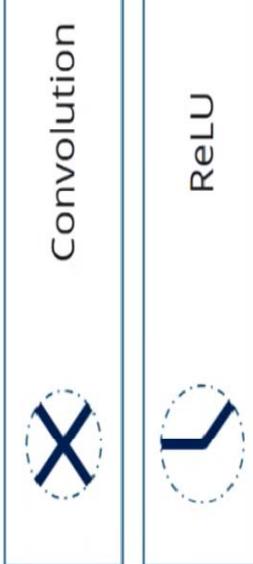
| | | | |
|------|------|------|------|
| 1.00 | 0.33 | 0.55 | 0.33 |
| 0.33 | 1.00 | 0.33 | 0.55 |
| 0.55 | 0.33 | 1.00 | 0.11 |
| 0.33 | 0.55 | 0.11 | 0.77 |

| | | | |
|------|------|------|------|
| 0.55 | 0.33 | 0.55 | 0.33 |
| 0.33 | 1.00 | 0.55 | 0.11 |
| 0.55 | 0.55 | 0.55 | 0.11 |
| 0.33 | 0.11 | 0.11 | 0.33 |

| | | | |
|------|------|------|------|
| 0.33 | 0.55 | 1.00 | 0.77 |
| 0.55 | 0.55 | 1.00 | 0.33 |
| 1.00 | 1.00 | 0.11 | 0.55 |
| 0.77 | 0.33 | 0.55 | 0.33 |

Stacking up the layers

| | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|
| -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 | -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 |



Input Image

| | | | | | | |
|------|------|------|------|------|------|------|
| 0.77 | 0 | 0.11 | 0.33 | 0.55 | 0 | 0.33 |
| 0 | 1.00 | 0 | 0.33 | 0 | 0.11 | 0 |
| 0.11 | 0 | 1.00 | 0 | 0.11 | 0 | 0.55 |
| 0.33 | 0.33 | 0 | 0.55 | 0 | 0.33 | 0.33 |
| 0.55 | 0 | 0.11 | 0 | 1.00 | 0 | 0.11 |
| 0 | 0.11 | 0 | 0.33 | 0 | 1.00 | 0 |
| 0.33 | 0 | 0.55 | 0.33 | 0.11 | 0 | 1.77 |

| | | | | | | |
|------|------|------|------|------|------|------|
| 0.33 | 0 | 0.11 | 0 | 0.11 | 0 | 0.33 |
| 0 | 0.55 | 0 | 0.33 | 0 | 0.55 | 0 |
| 0.11 | 0 | 0.55 | 0 | 0.55 | 0 | 0.11 |
| 0 | 0.33 | 0 | 1.00 | 0 | 0.33 | 0 |
| 0.11 | 0 | 0.55 | 0 | 0.55 | 0 | 0.11 |
| 0 | 0.55 | 0 | 0.33 | 0 | 0.55 | 0 |
| 0.33 | 0 | 0.11 | 0 | 0.11 | 0 | 0.33 |

| | | | | | | |
|------|------|------|------|------|------|------|
| 0.33 | 0 | 0.11 | 0.33 | 0.11 | 0 | 0.77 |
| 0 | 0.71 | 0 | 0.33 | 0 | 1.00 | 0 |
| 0.55 | 0 | 0.11 | 0 | 1.00 | 0 | 0.33 |
| 0.33 | 0.33 | 0 | 0.66 | 0 | 0.33 | 0.33 |
| 0.11 | 0 | 1.00 | 0 | 0.33 | 0 | 0.33 |
| 0.77 | 0 | 0.33 | 0.33 | 0.33 | 0 | 0.33 |

7X7 matrices

The diagram shows the 7x7 matrices being processed by two parallel layers: Pooling and 4x4 matrices. The Pooling layer is represented by a blue triangle pointing right, and the 4x4 matrices layer is represented by a blue triangle pointing right.

| | | | |
|------|------|------|------|
| 1.00 | 0.33 | 0.55 | 0.33 |
| 0.33 | 1.00 | 0.33 | 0.55 |
| 0.55 | 0.33 | 1.00 | 0.11 |
| 0.33 | 0.55 | 0.11 | 0.77 |

| | | | |
|------|------|------|------|
| 0.55 | 0.33 | 0.55 | 0.33 |
| 0.33 | 1.00 | 0.55 | 0.11 |
| 0.55 | 0.55 | 0.55 | 0.11 |
| 0.33 | 0.11 | 0.11 | 0.33 |

| | | | |
|------|------|------|------|
| 0.33 | 0.55 | 1.00 | 0.77 |
| 0.55 | 0.55 | 1.00 | 0.33 |
| 1.00 | 1.00 | 0.11 | 0.55 |
| 0.77 | 0.33 | 0.55 | 0.33 |

4X4 matrices

The diagram shows the 4x4 matrices being processed by two parallel layers: Pooling and 2x2 matrices. The Pooling layer is represented by a blue triangle pointing right, and the 2x2 matrices layer is represented by a blue triangle pointing right. The 2x2 matrices are then flattened into a single vertical column.

| | |
|------|------|
| 1 | 0.55 |
| 0.55 | 1.00 |

| | |
|------|------|
| 1 | 0.55 |
| 0.55 | 0.55 |

| | |
|------|------|
| 0.55 | 1.00 |
| 1.00 | 0.55 |

2X2 matrices

| | | | |
|------|------|------|------|
| 1.00 | 0.55 | 0.55 | 0.55 |
| 0.55 | 1.00 | 1.00 | 1.00 |
| 0.55 | 0.55 | 1.00 | 1.00 |
| 0.55 | 0.55 | 0.55 | 1.00 |

Flatten

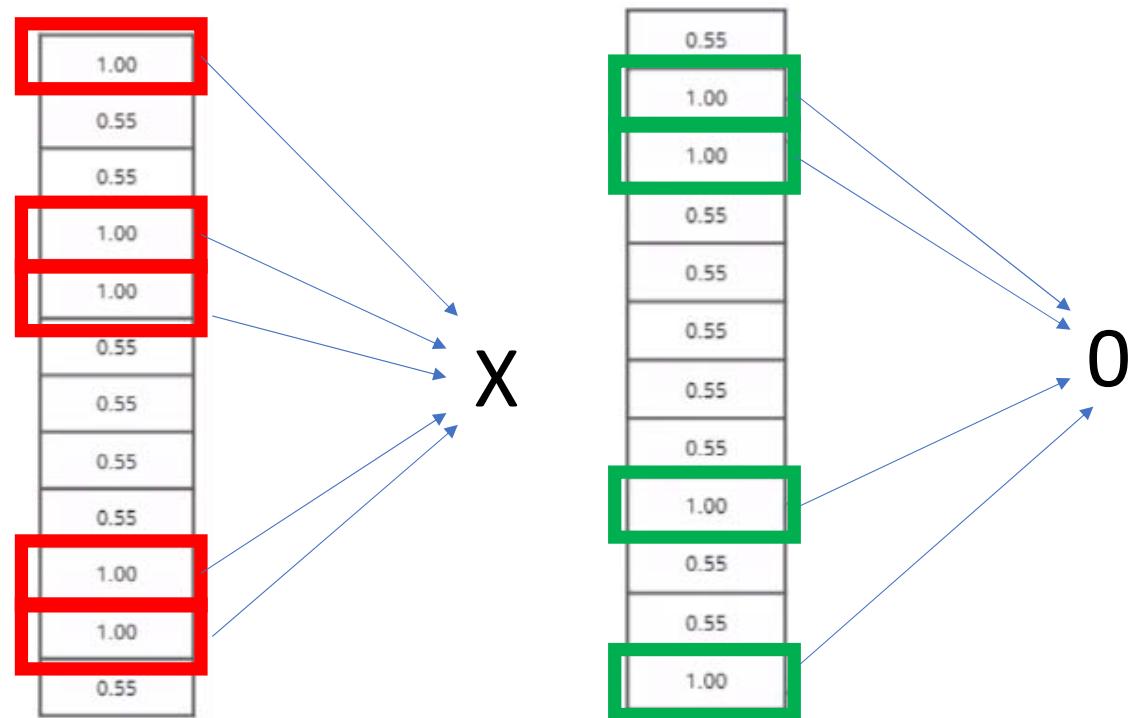
Output

When we feed in, 'X' and 'O'. Then there will be some element that will be high.

Now if we have an input image with 1st, 4th, 5th, 10th, 11th value high we can say that the image is X

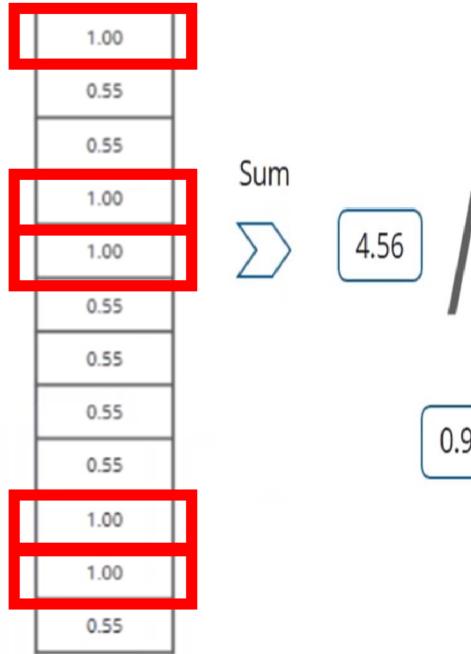
Similarly, if we have an image with 2nd, 3rd, 9th and 12th value high, we can say that it is O

This completes the training of our model. Now let's see the results on some unseen images.

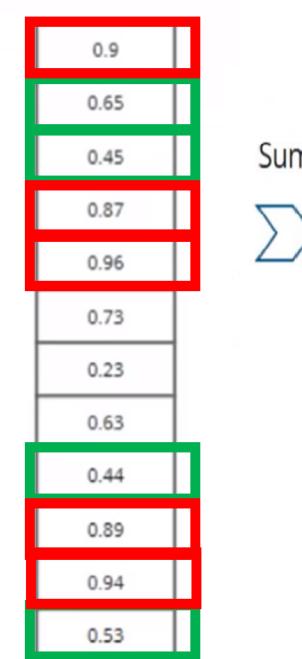


Compare the input vector with X and O

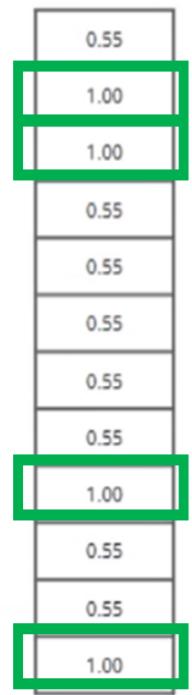
Let's compare with the list of 'X' and 'O'



Vector for 'X'

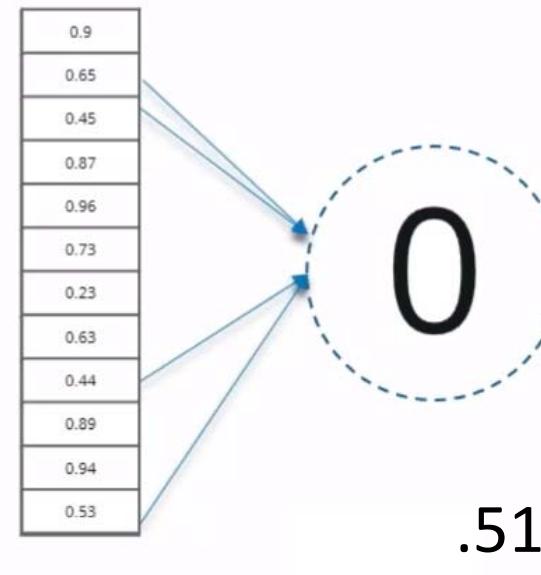


Input Image



Vector for 'O'

Result



The new input image is classified as 'X'