



Machine Intelligence

Dr. Arti Arya

Department of Computer Science & Engineering

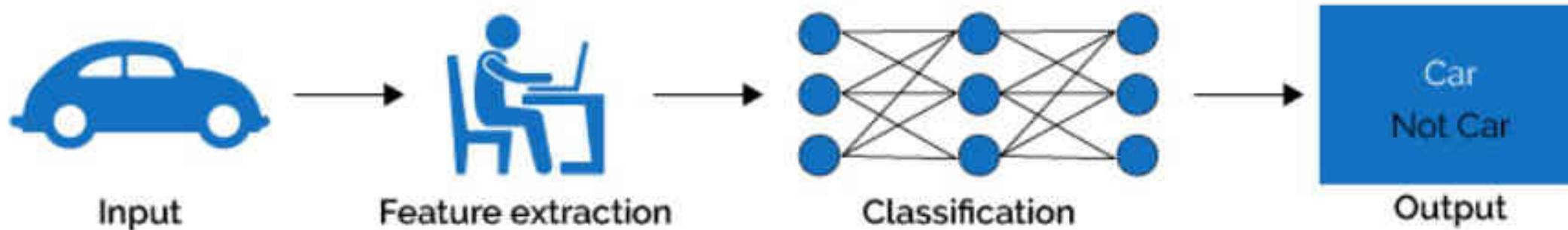
UE20CS302-Machine Intelligence

Disclaimer

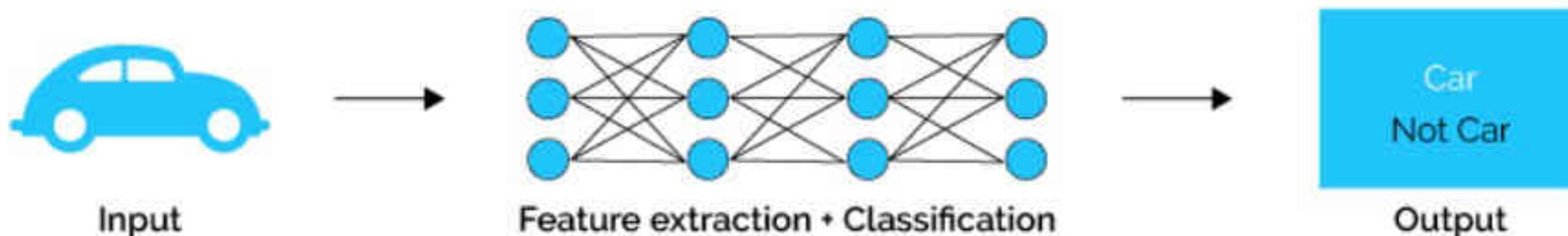


These slides are prepared from various resources from Internet and universities from India and Abroad.

Machine Learning



Deep Learning



Machine Intelligence

Unit 5 - Introduction to Deep Learning



- It sits under the artificial intelligence umbrella, either alongside machine learning or as a subset.
- The difference is that machine learning uses algorithms developed for specific tasks.
- Deep learning is more of a data representation based upon multiple layers of a matrix, where each layer uses output from the previous layer as input.

Unit 5 - Introduction to Deep Learning

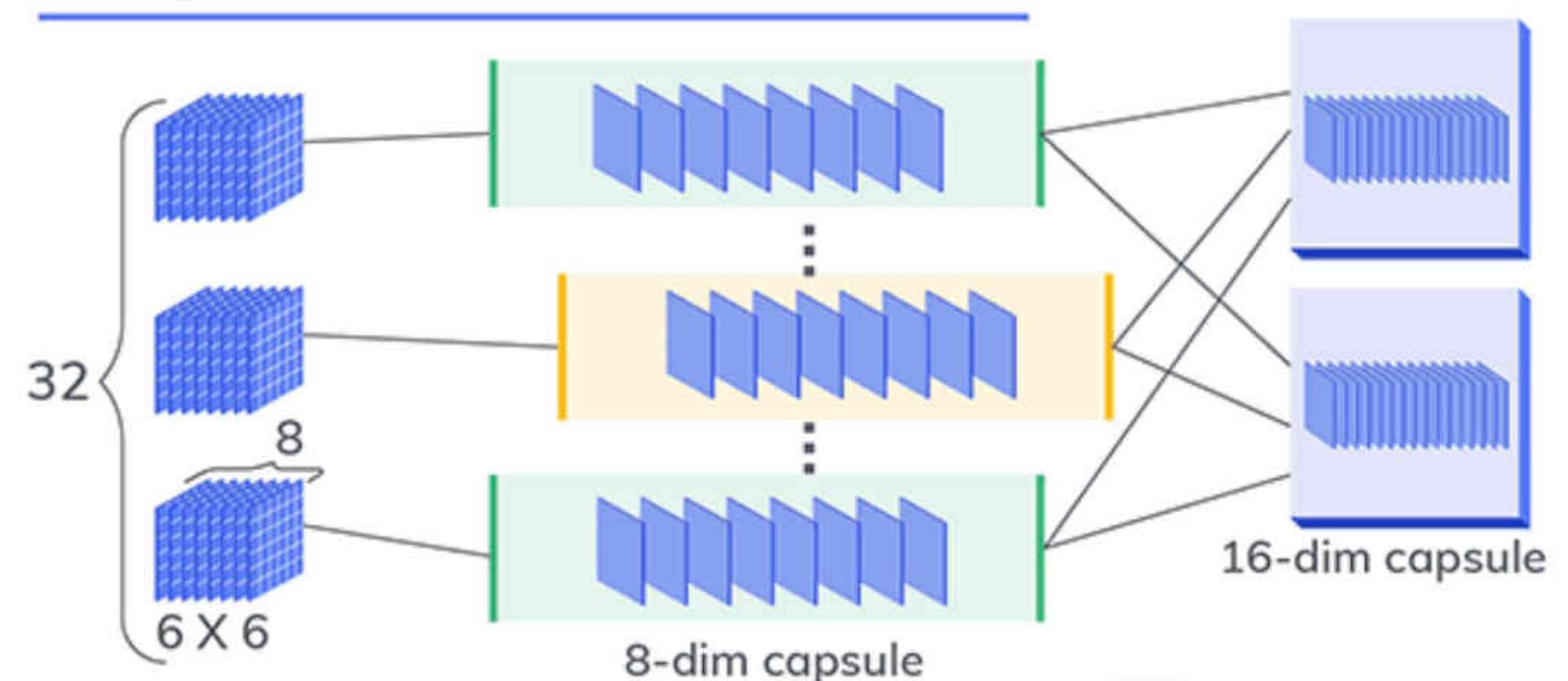
- Geoffrey Hinton, a British computer scientist and cognitive psychologist, also has been pushing the idea of capsule networks, which **stack up layers inside of layers of a neural network**, basically increasing the density of those layers.

I would highly recommend to read about capsule networks.

<https://towardsdatascience.com/capsule-neural-networks-part-2-what-is-a-capsule-846d5418929f>

But NOT a PART OF
SYLLABUS

Capsule Networks



Unit 5 - Introduction to Deep Learning

- Yet behind all of this there is no consensus about exactly how deep learning works, particularly as it moves from training to inferencing. Deep learning is more of mathematical distribution for complex behavior.
- To achieve that representation, and to shape it, there are a number of architectures being utilized. Deep neural networks and convolutional neural networks are the most common. Recurrent neural networks are being used, as well, which add the dimension of time.
- The downside of RNNs is the immense amount of processing, memory and storage requirements, which limits its use to large data centers.

Unit 5 - Introduction to Deep Learning

- Deep learning is a form of machine learning that enables computers to learn from experience and understand the world in terms of a hierarchy of concepts.
- A graph of these hierarchies can be many layers deep.
- Deep learning is implemented by the help of deep networks, which are nothing but neural networks with multiple hidden layers.
- “Deep learning, in part, is a black box,” said Michael Schuldenfrei, CTO at Optimal+.

- Deep learning algorithms typically need a lot more computational horsepower than machine learning.
- It uses convolution, pooling, and specific activation functions.
- Some of the techniques are similar to machine learning, some are different.”

Artificial Intelligence

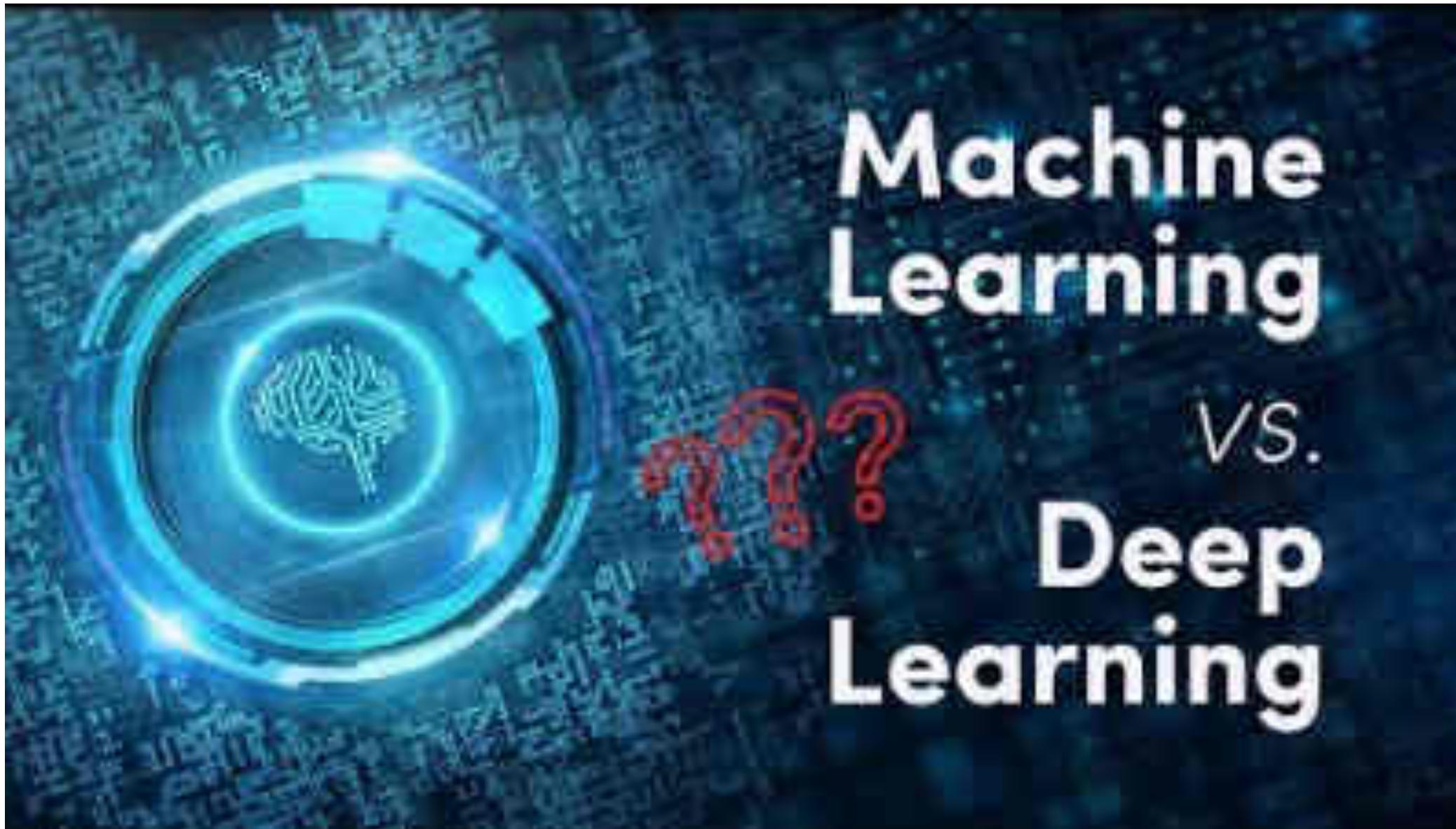
The theory and development of computer systems able to perform tasks normally requiring human intelligence

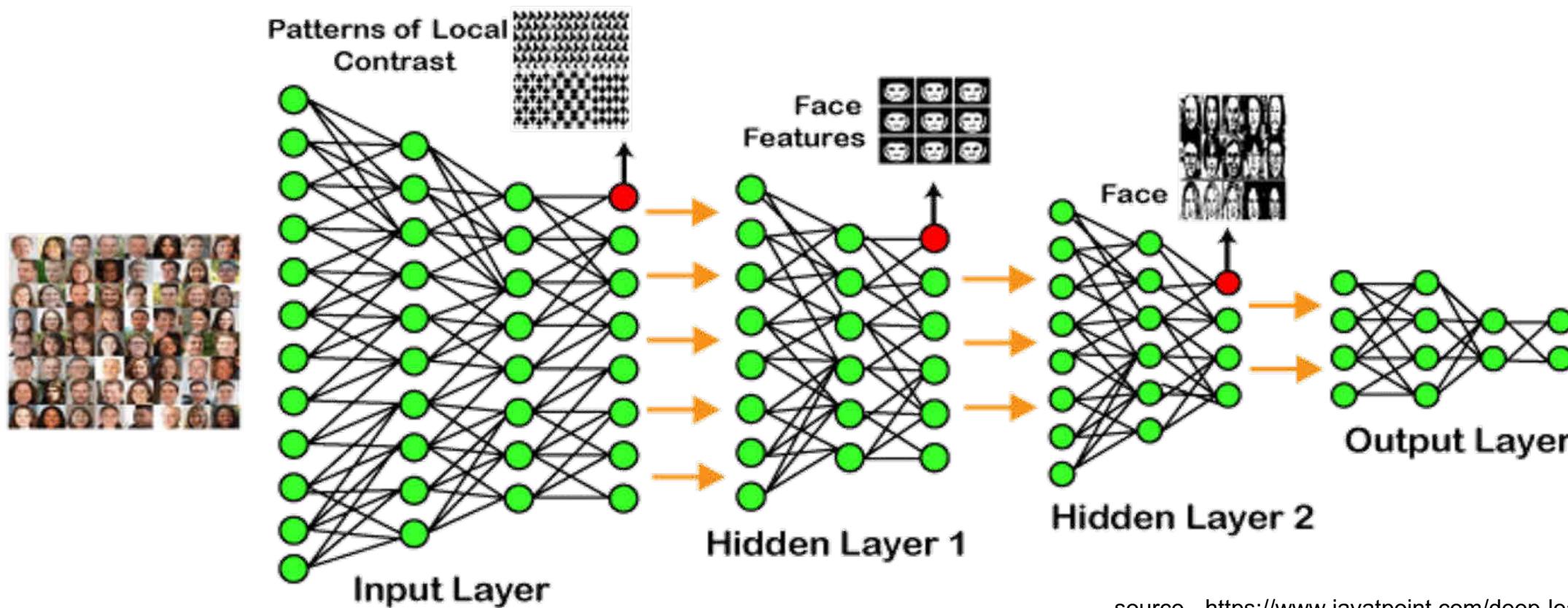
Machine Learning

Gives computers "the ability to learn without being explicitly programmed"

Deep Learning

Machine learning algorithms with brain-like logical structure of algorithms called artificial neural networks





source - <https://www.javatpoint.com/deep-learning>

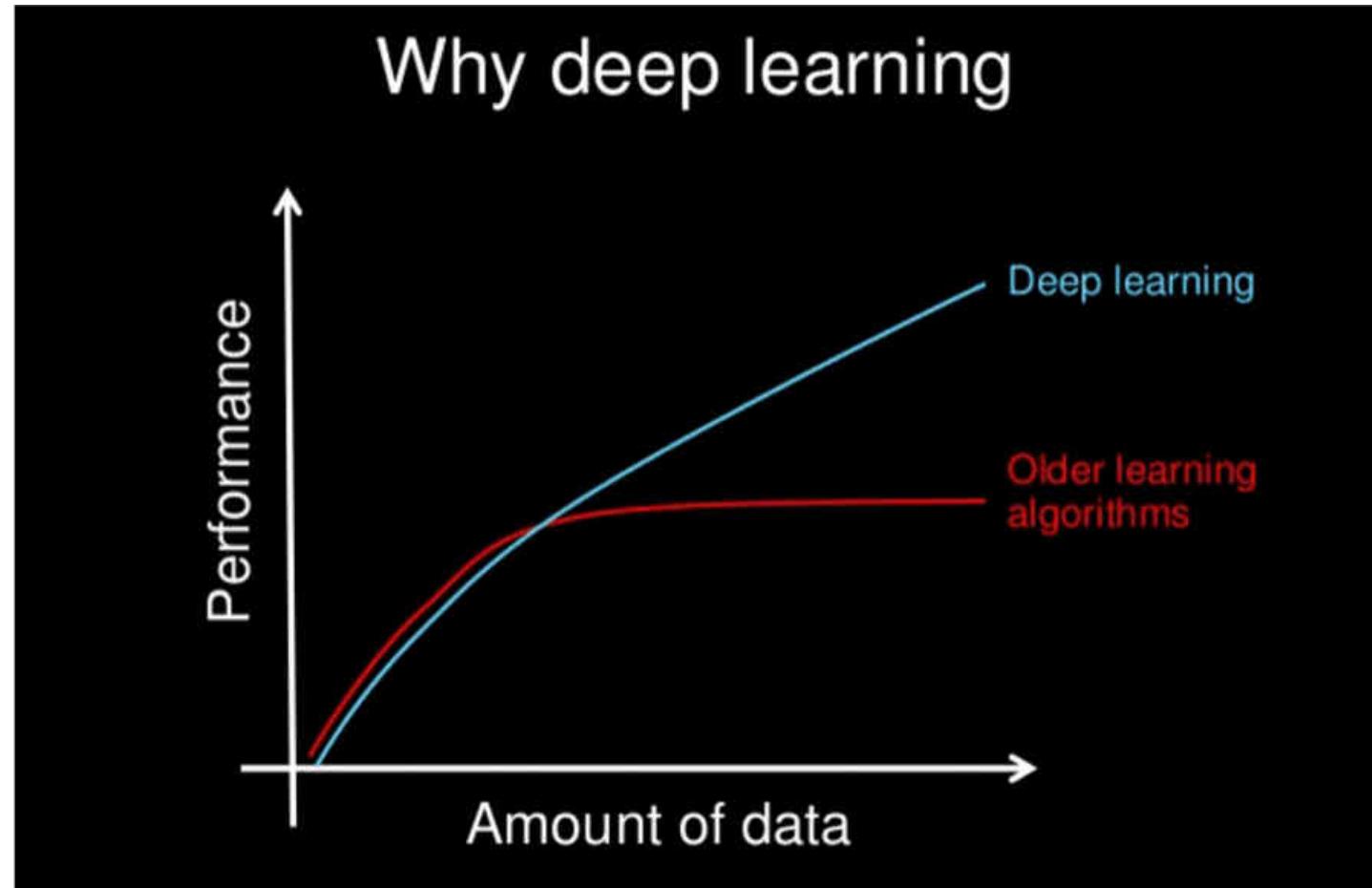
- Consider the raw data of images being provided to the first layer of the input layer - which determines the patterns of local contrast - it will differentiate on the basis of colors, luminosity, etc.
- Then the 1st hidden layer will determine the face feature, i.e., it will fixate on eyes, nose, and lips, etc.
- In the 2nd hidden layer, it determines the correct face after which it will be sent to the output layer

So, more hidden layers can be added to solve more complex problems

Machine Intelligence

Why Deep Learning ??

- When there is lack of domain understanding for feature introspection, Deep Learning techniques outshines others bcoz there is no worry about feature engineering



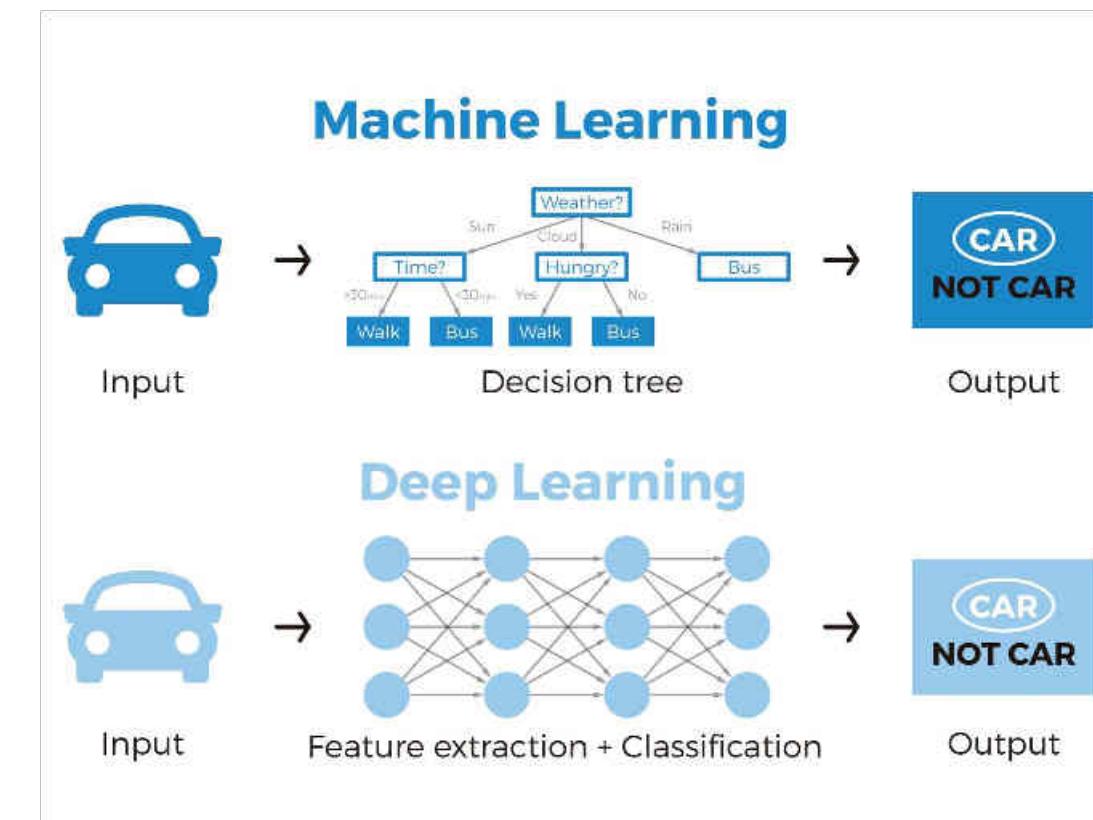
Source - <https://machinelearningmastery.com/what-is-deep-learning/>

- It is scalable - performance just keeps getting better as you feed them more data

Machine Intelligence

Deep Learning Vs Machine Learning

| Machine Learning | Deep Learning |
|--|---|
| Machine learning uses algorithms to parse data, learn from that data, and make informed decisions based on what it has learned | Deep learning structures algorithms in layers to create an “artificial neural network” that can learn and make intelligent decisions on its own |
| Can train on lesser training data | Requires large data sets for training |
| Takes less time to train | Takes longer time to train |
| Trains on CPU | Trains on GPU for proper training |
| The output is in numerical form for classification and scoring applications | The output can be in any form including free form elements such as free text and sound |
| Limited tuning capability for hyperparameter tuning | Can be tuned in various ways |



Machine Intelligence

Quiz!



Which of these are the reasons for Deep Learning taking off recently?

1. We have access to a lot more computational power.
2. Neural Networks are a brand new field.
3. We have access to a lot more data.
4. Deep learning has resulted in significant improvements in important applications such as online advertising, speech recognition, and image recognition.

Machine Intelligence

Quiz!



When an experienced deep learning engineer works on a new problem, they can usually use insight from previous problems to train a good model on the first try, without needing to iterate multiple times through different models.
True/False?

FALSE



THANK YOU

Dr. Arti Arya

Department of Computer Science & Engineering



PES
UNIVERSITY
ONLINE

MACHINE INTELLIGENCE

Sequence Learning Problems

Dr. Arti Arya
Department of Computer Sc. and Engg

MACHINE INTELLIGENCE

Sequence Learning Problems

Dr. Arti Arya
Department of Computer Sc. And Engg.

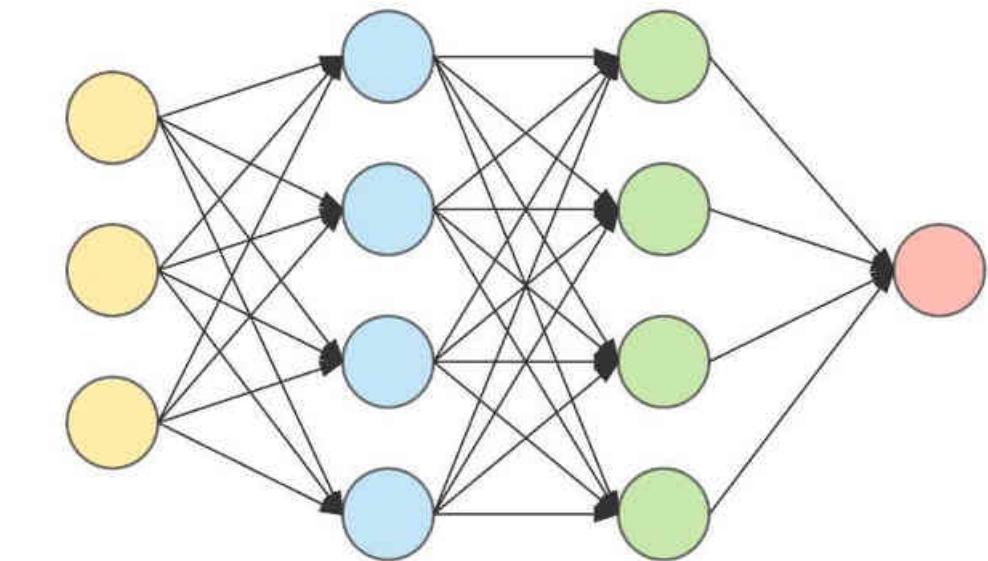
Disclaimer

These slides are prepared from various resources from Internet and universities from India and Abroad. Broadly adapted from slides by Prof. Mitesh M Khapra from IITM.

Machine Intelligence

Look at this kind of dataset....

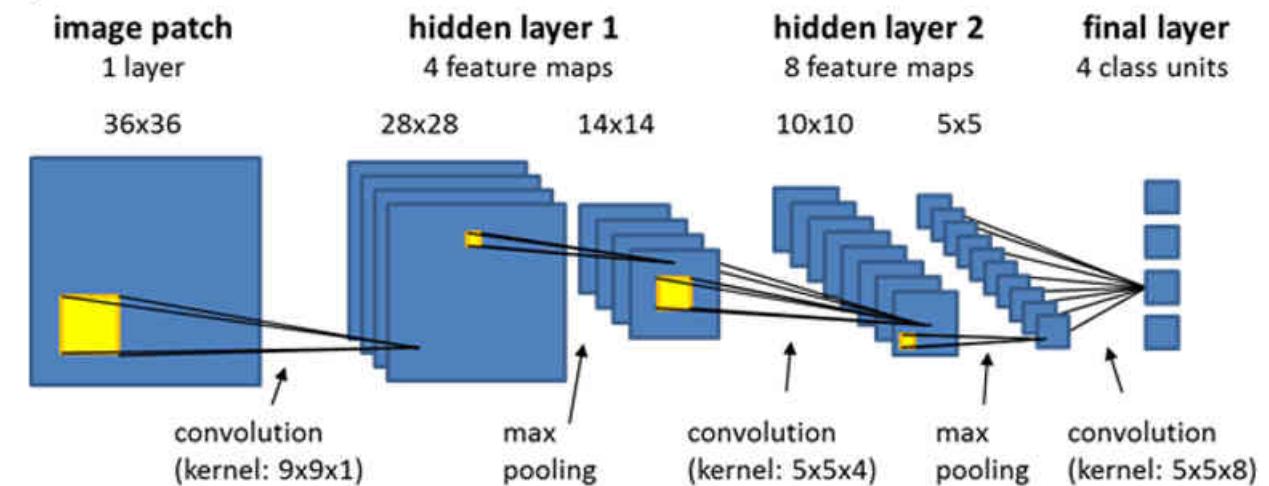
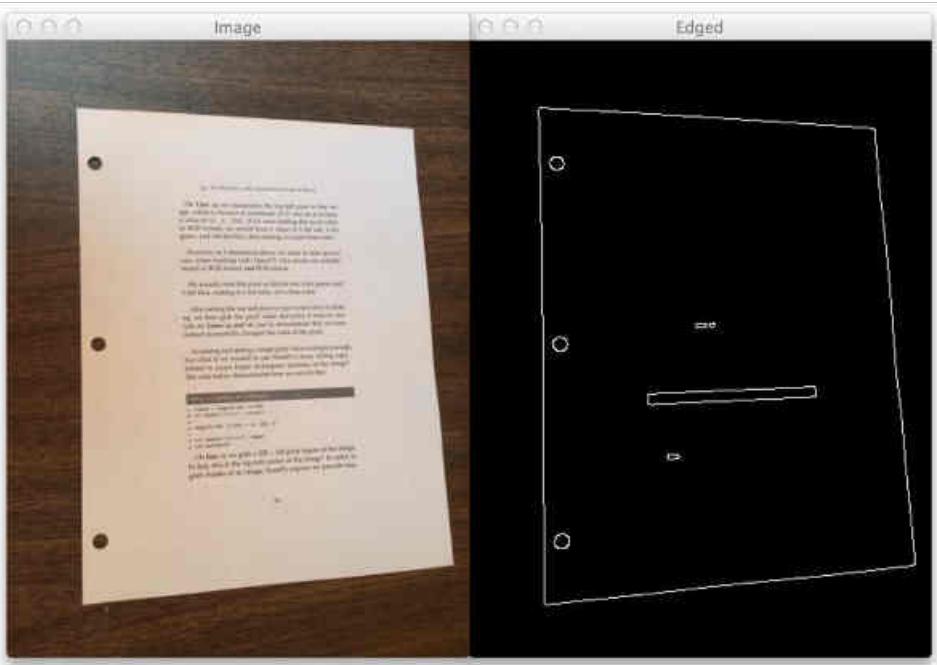
| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DIS | RAD | TAX | PTRATIO | B | LSTAT |
|---|---------|------|-------|------|-------|-------|-------|--------|-----|-------|---------|--------|-------|
| 0 | 0.00632 | 18.0 | 2.31 | 0.0 | 0.538 | 6.575 | 65.2 | 4.0900 | 1.0 | 296.0 | 15.3 | 396.90 | 4.98 |
| 1 | 0.02731 | 0.0 | 7.07 | 0.0 | 0.469 | 6.421 | 78.9 | 4.9671 | 2.0 | 242.0 | 17.8 | 396.90 | 9.14 |
| 2 | 0.02729 | 0.0 | 7.07 | 0.0 | 0.469 | 7.185 | 61.1 | 4.9671 | 2.0 | 242.0 | 17.8 | 392.83 | 4.03 |
| 3 | 0.03237 | 0.0 | 2.18 | 0.0 | 0.458 | 6.998 | 45.8 | 6.0622 | 3.0 | 222.0 | 18.7 | 394.63 | 2.94 |
| 4 | 0.06905 | 0.0 | 2.18 | 0.0 | 0.458 | 7.147 | 54.2 | 6.0622 | 3.0 | 222.0 | 18.7 | 396.90 | 5.33 |
| 5 | 0.02985 | 0.0 | 2.18 | 0.0 | 0.458 | 6.430 | 58.7 | 6.0622 | 3.0 | 222.0 | 18.7 | 394.12 | 5.21 |
| 6 | 0.08829 | 12.5 | 7.87 | 0.0 | 0.524 | 6.012 | 66.6 | 5.5605 | 5.0 | 311.0 | 15.2 | 395.60 | 12.43 |
| 7 | 0.14455 | 12.5 | 7.87 | 0.0 | 0.524 | 6.172 | 96.1 | 5.9505 | 5.0 | 311.0 | 15.2 | 396.90 | 19.15 |
| 8 | 0.21124 | 12.5 | 7.87 | 0.0 | 0.524 | 5.631 | 100.0 | 6.0821 | 5.0 | 311.0 | 15.2 | 386.63 | 29.93 |
| 9 | 0.17004 | 12.5 | 7.87 | 0.0 | 0.524 | 6.004 | 85.9 | 6.5921 | 5.0 | 311.0 | 15.2 | 386.71 | 17.10 |



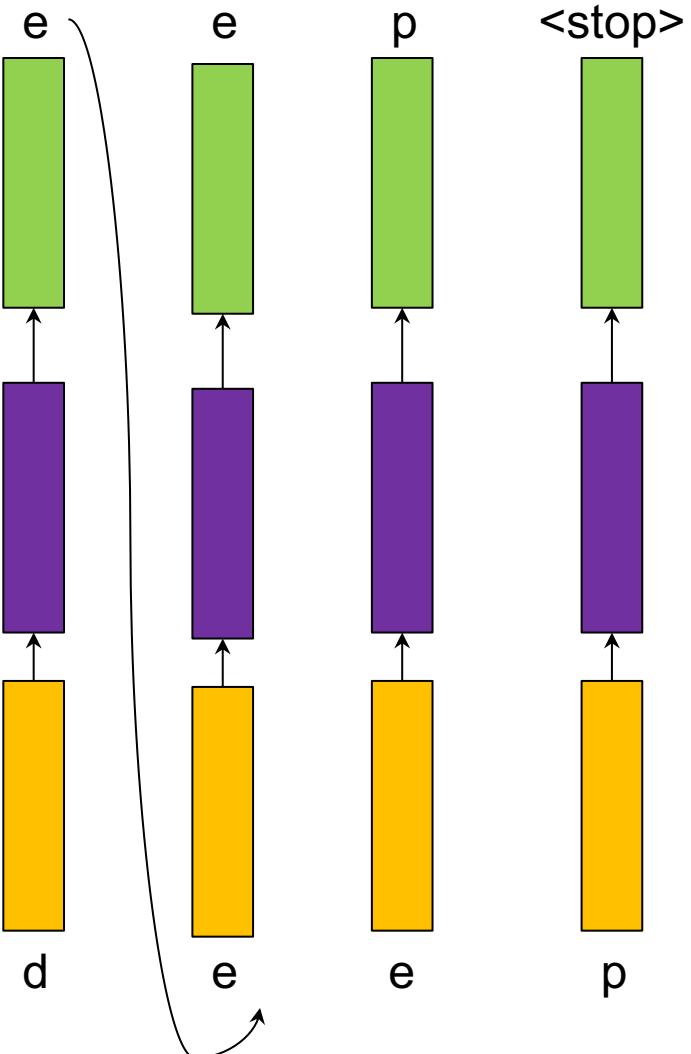
- Suppose we have to design a model that can predict LSTAT (ie it has learnt to predict LSTAT)
- So one can create a feed forward neural network with appropriate hyperparameters.

Machine Intelligence

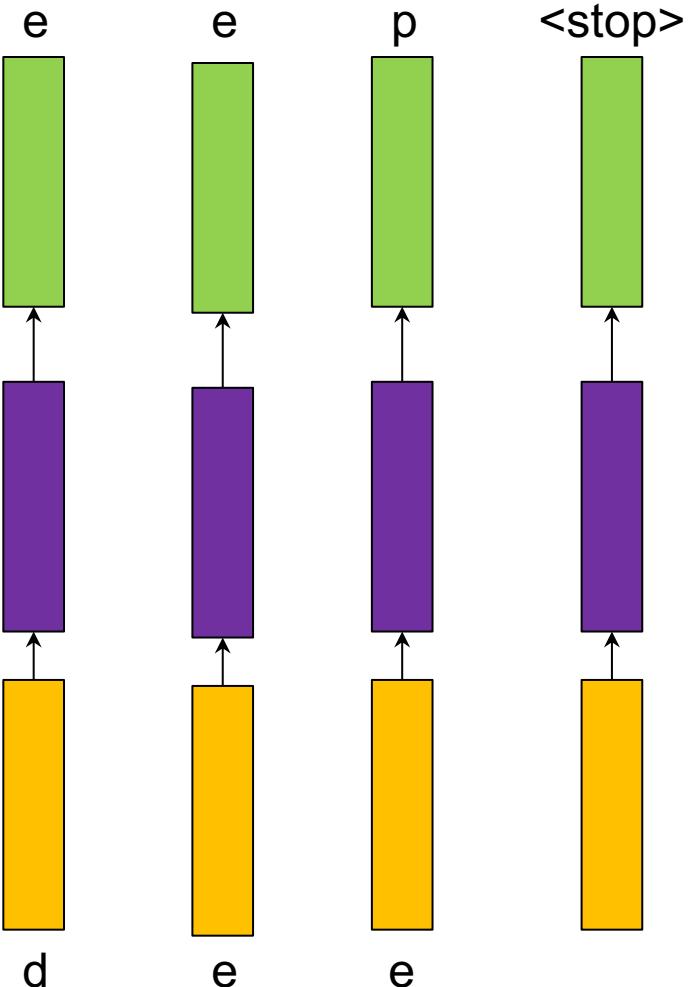
Now this one



- So if you have to create a network that will detect edges of a paper.....
- You can create a CNN with good hyperparameter



- In many applications the **input is not of a fixed size**
- Further successive inputs may not be independent of each other
- For example, consider the **task of auto completion**
- Say given the first character 'd', we want to predict next character 'e' and once predicted 'e' we want to predict the next character and so on.

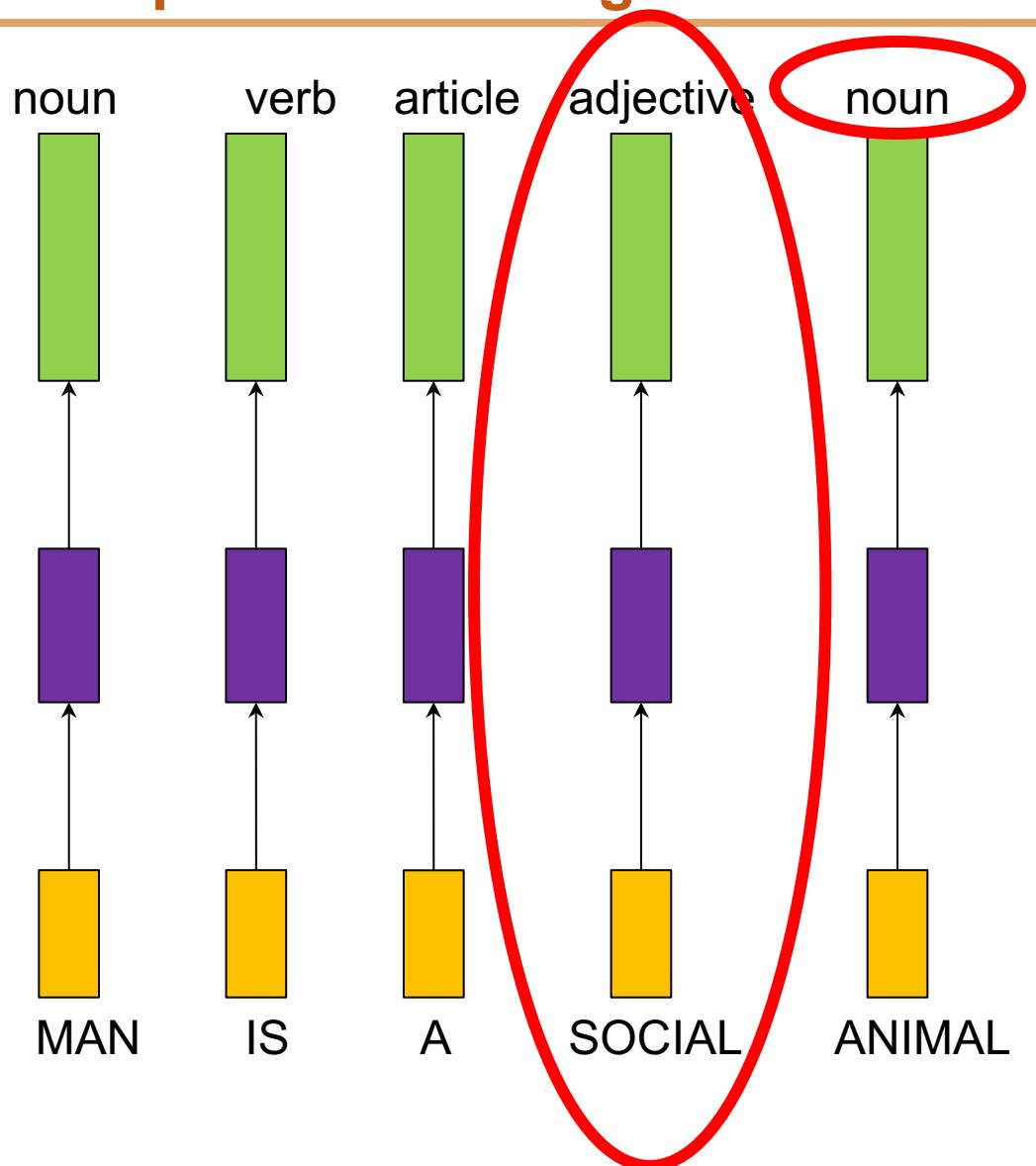


- Please note that, **successive inputs are not independent** (while predicting ‘e’ you would want to know what the previous input was in addition to the current input)
- Also, the **length of the inputs and the number of predictions** you need to make is **not fixed** (for example, “learn”, “deep”, “machine” have different number of characters)
- Also, **each network in the figure (orange-blue-green structure) is performing the same task** (input : character output : character)

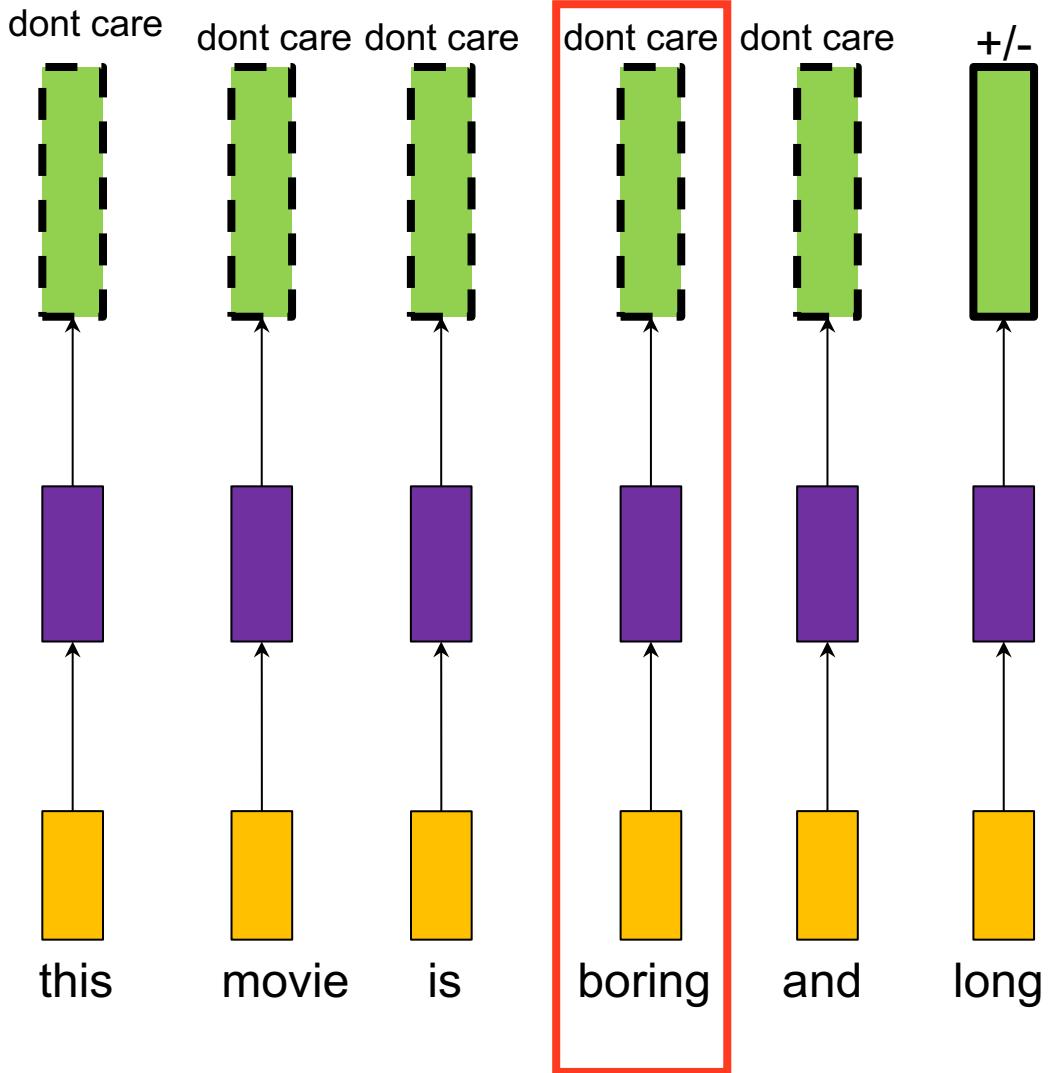
There are four basic sequence learning problems:

- Sequence prediction,
- Sequence generation,
- Sequence recognition, and
- Sequential decision making.

Sequence Learning Problem

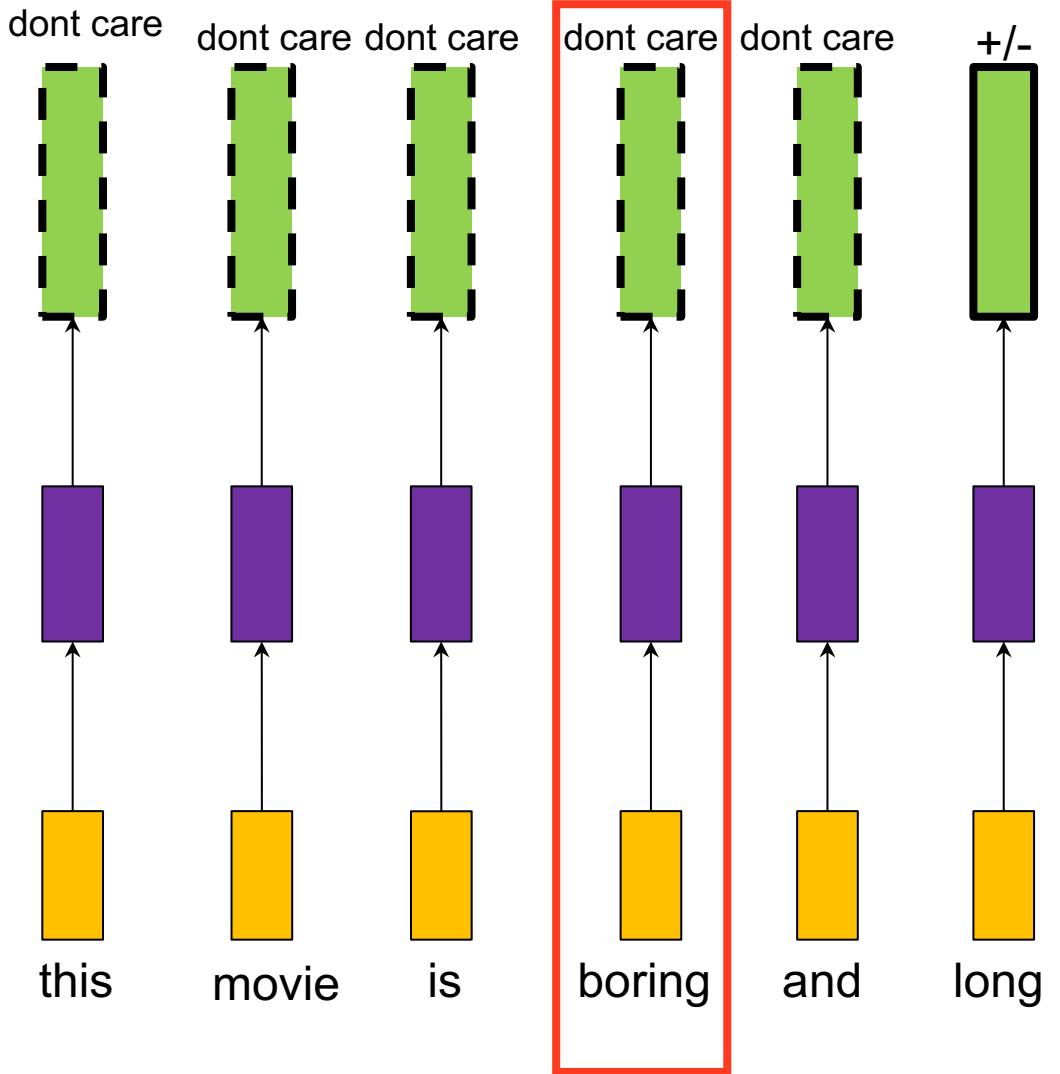


- Consider the **task of predicting the part of speech tag** (noun, adverb, adjective, verb) of each word in a sentence
- If we see an **adjective (social)**, then it is almost sure that **the next word should be a noun (animal)**
- Thus the **current output (noun)** depends on the **current input as well as the previous input**
- Further the size of the input is not fixed (sentences could have arbitrary number of words).
 - Here we are interested in producing an output at each time step
 - Each network is performing the same task (input : word, output : tag)



- Say, instead of producing an output at every stage, consider **the full input sequence and then produce an output.**
- Eg, in the task of predicting the **polarity of a movie**
- The prediction clearly does not depend only on the last word but also on some words which appear before

Sequence Learning Problem



- Just by considering the last word, one can't decide even by taking a word somewhere in the middle.
- In this case, the network is performing the same task at each step (input : word, output : +/-) but we don't care about intermediate outputs
- So such are sequence learning problems

- Consider a word '**bank**' in a sentence.
Bank could be a Verb (I can bank on him) or
a Noun(I had gone to a bank).
- In the case of **Noun**, the **previous word is an article** (a), from which we get to know that the following word is very unlikely to be a verb, it is going to be a Noun.
- So, even in this type of **ambiguous cases**, we look at the previous sequence of words(the context) and we are in a better position to make this decision

Some more Examples of Sequence Learning Problem

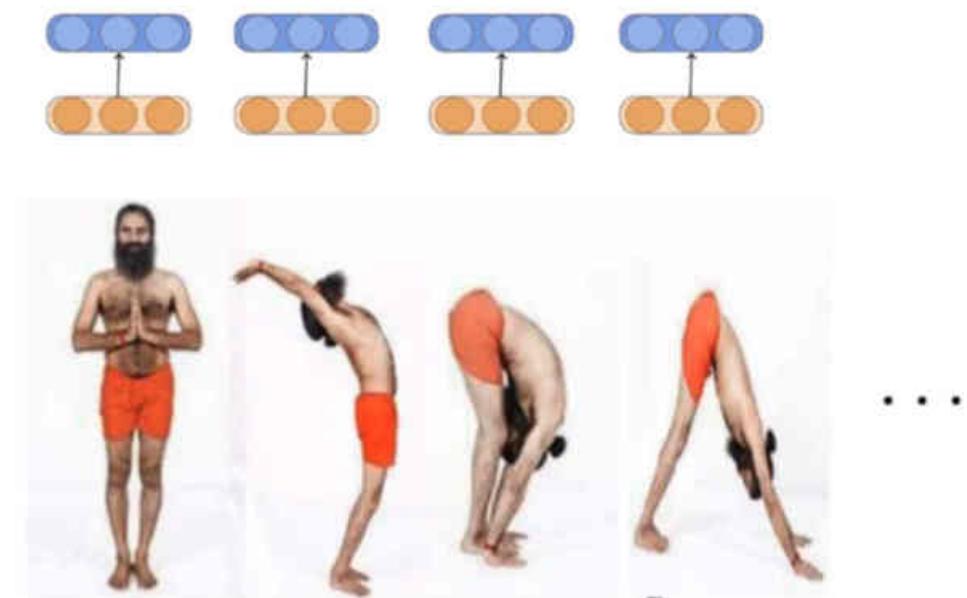
Speech Recognition —We can think of speech as a sequence of phonemes and we take the speech signal as the input and tries to map to phonemes in a language.

Another thing we can do is to look at the entire sequence of speech than say whether the person is speaking angrily, is happy/sad and etc.

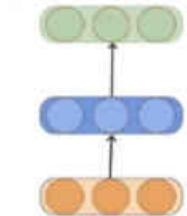
Some more Examples of Sequence Learning Problem

Video Labeling — A video is a sequence of frames

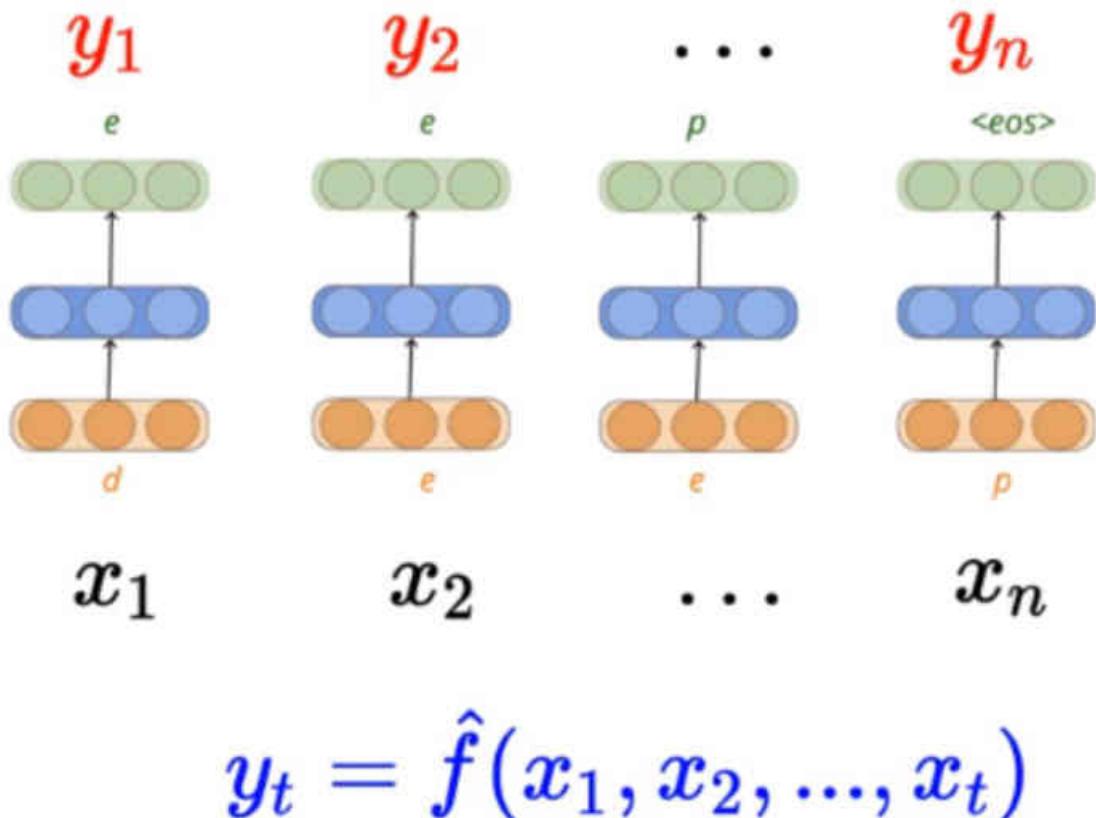
- So labeling of every frame in the video can be done;
- One can examine all the frames and give the label for the entire video.
- Input is again variable in this case as the no. of frames would be different for different videos.
- At every time step, we are taking in one frame as the input and in the end, we are assigning a label to the video.



Surya
Namaskar



How to model Sequence Learning Problem



- Let the output at the 1st-time step as y_1 ,
- Output at the 2nd-time step as y_2 and so on.
- So, in general, the output at the t^{th} time step as y_t (where 't' is the time step).

Clearly, y_t depends on all the previous input that we have seen so far.

It may not depend on all the previous inputs but at least it depends on some of the previous inputs apart from the current inputs.



THANK YOU

Dr. Arti Arya

Department of Computer Science & Engineering



MACHINE INTELLIGENCE

RNN- Recurrent Neural Networks

Mr. Vignesh Kamath

+

Inputs from Dr. Arti Arya

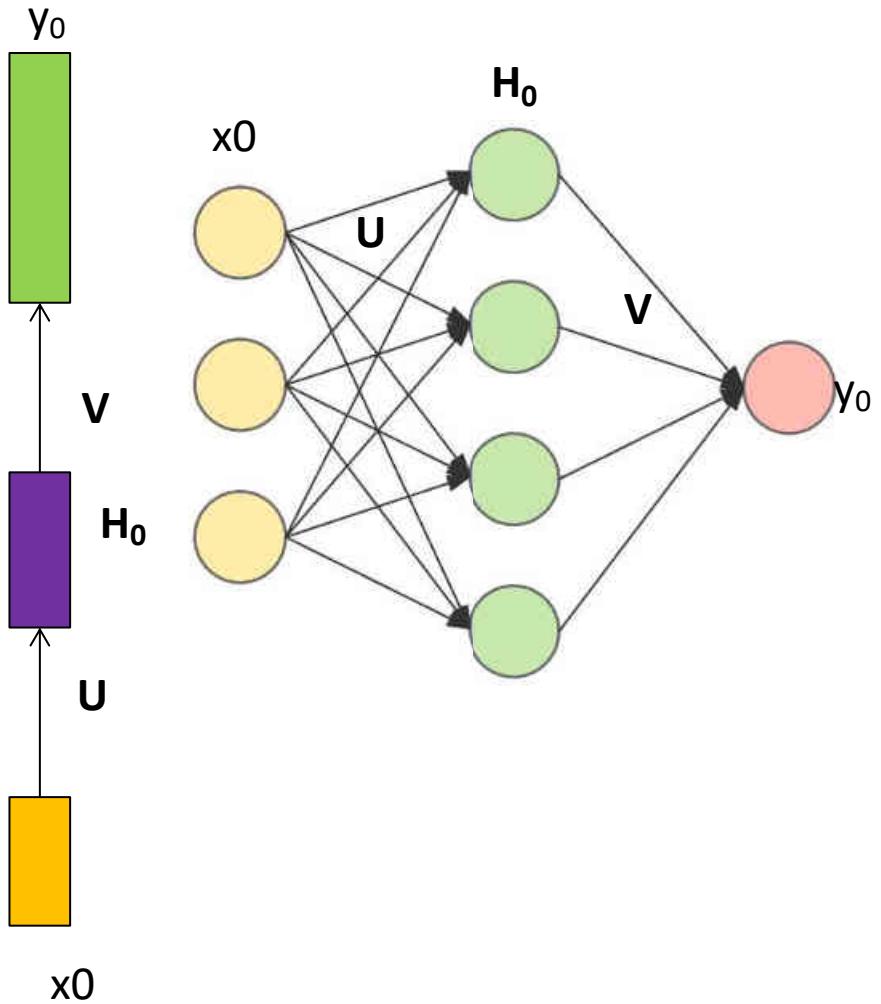
Department of Computer Science and Engineering

Disclaimer

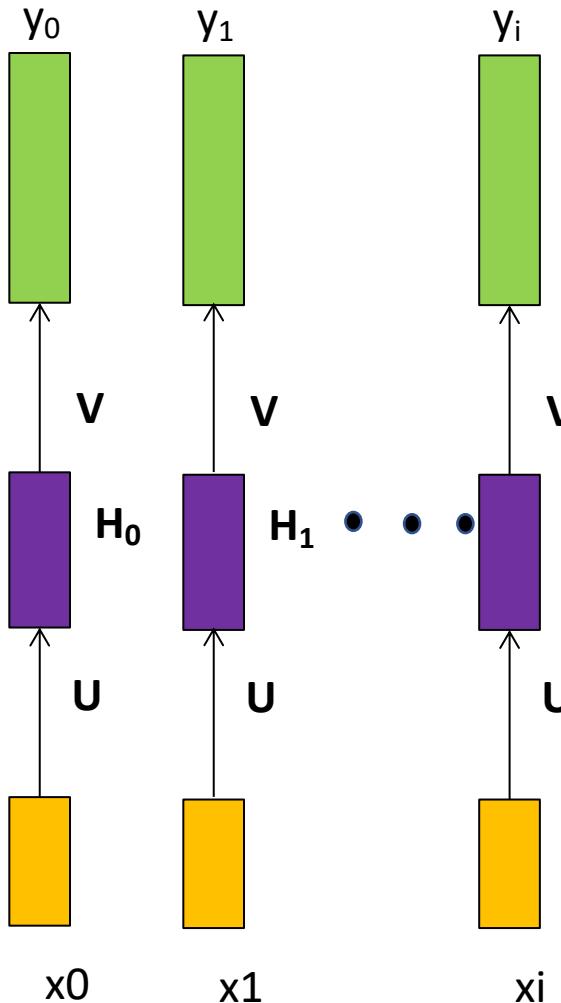
These slides are prepared from various resources from Internet and universities from India and Abroad and with the help from TAs . Broadly adapted from slides by Prof. Mitesh M Khapra from IITM

As we have seen Sequence learning Problems, so we want a model that can:

- Account for **dependencies between inputs**
- Account for **variable number of inputs**
- Make sure that the **function executed at each time step is the same**



- There is an input x , a hidden layer H and output y which is always function of input
- AND this is a simple ANN (RECALL)



- Clearly from the first network

$$H_0 = f(x_0 U + b)$$

$$y_0 = g(H_0 V + c)$$

If the **activation function** for the hidden layer is **tanh** and for **output layer** is **softmax** then,

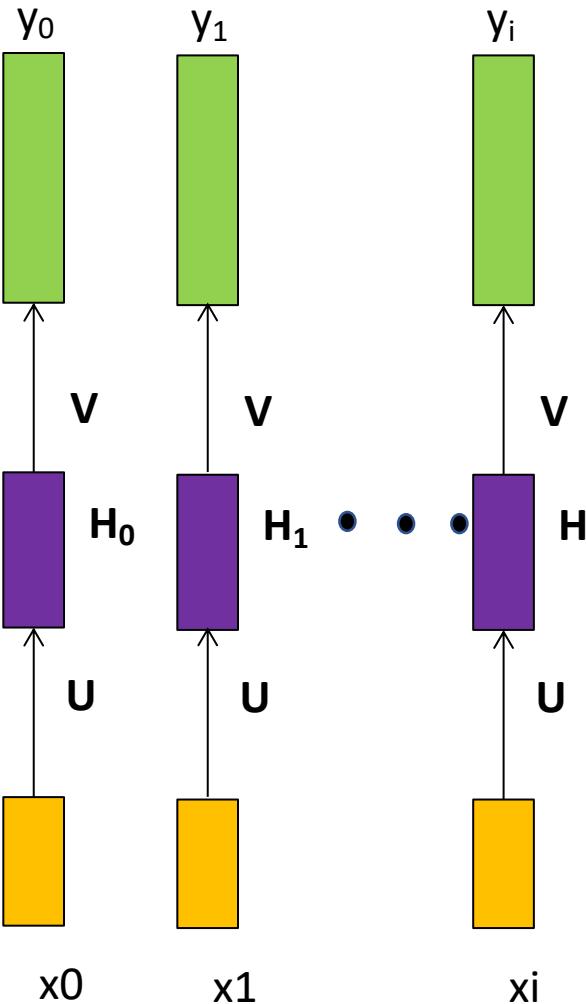
$$H_0 = \tanh(x_0 U + b)$$

$$y_0 = \text{softmax}(VH_0 + c)$$

- This will be the function that is executed at each time step i
- Since we want the same function to be executed at each timestep we should share the same network (which means same parameters)

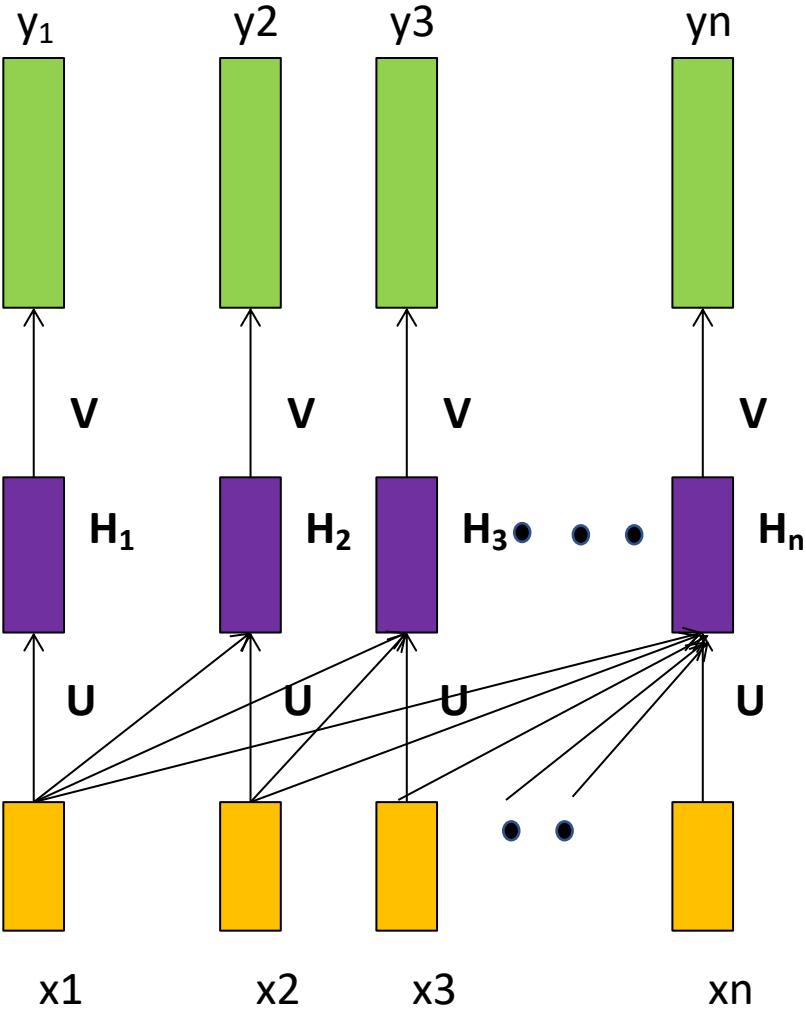
$$H_i = \tanh(x_i U + b)$$

$$Y_i = \text{softmax}(VH_i + c)$$



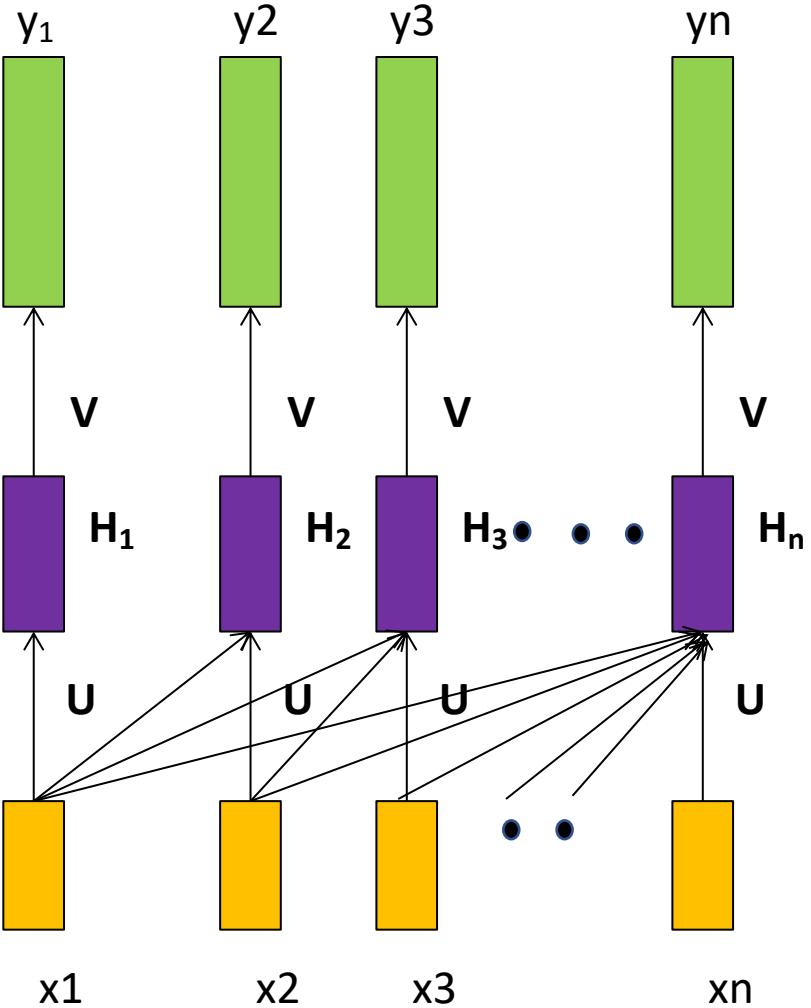
- This parameter sharing also ensures that the network becomes agnostic to the length (size) of the input.
- Now words of different sizes do not matter whether the word for autocomplete is “deep” or “deeper” or “deepest”.
- As we are simply computing the same function (with same parameters) at each time step, so **the number of time steps doesn't matter**
- It is just like creating **multiple copies of the network and executing them at each time step**.

How to account for dependencies between inputs



- Lets now look at the same network with following approach
 - At each time step, **feed all the previous inputs to the network**
 - Input x_1 at 1st time step, which predicts y_1 from x_1
 - At 2nd time step, the inputs are x_1 and x_2 and y_2 is predicted.
 - And so on

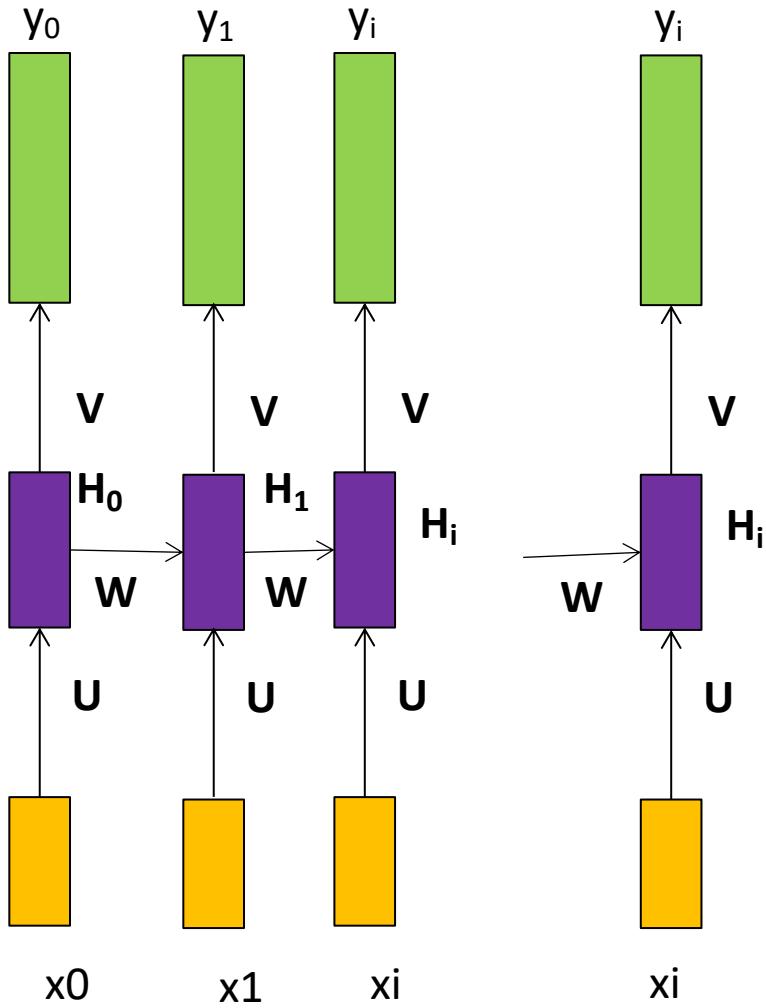
How to account for dependencies between inputs



- This method looks infeasiblewhy??
- *At the last time step, all the inputs are sequential inputs and it is just like a feedforward network.*
- Also,

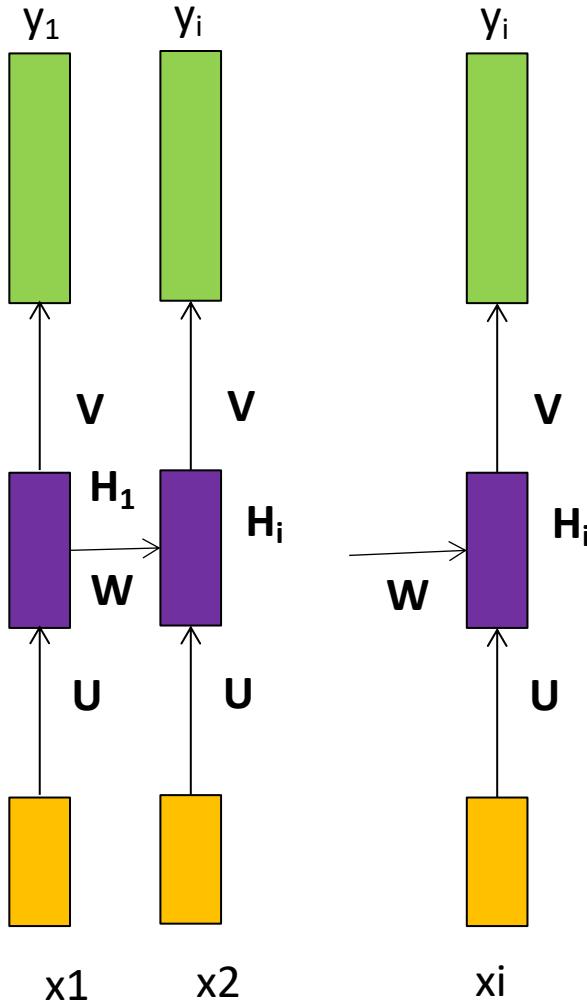
$$y_1 = \eta_1(x_1), y_2 = \eta_2(x_1, x_2), \dots, y_n = \eta_n(x_1, \dots, x_n)$$
- The network is now **sensitive to the length of the input sequence** as length of the sequence is changing at every time step.
- Eg, an i/p sequence of length 10 will require η_1, \dots, η_{10} functions whereas a sequence of length 100 will require $\eta_1, \dots, \eta_{100}$ functions.

How to account for dependencies between inputs



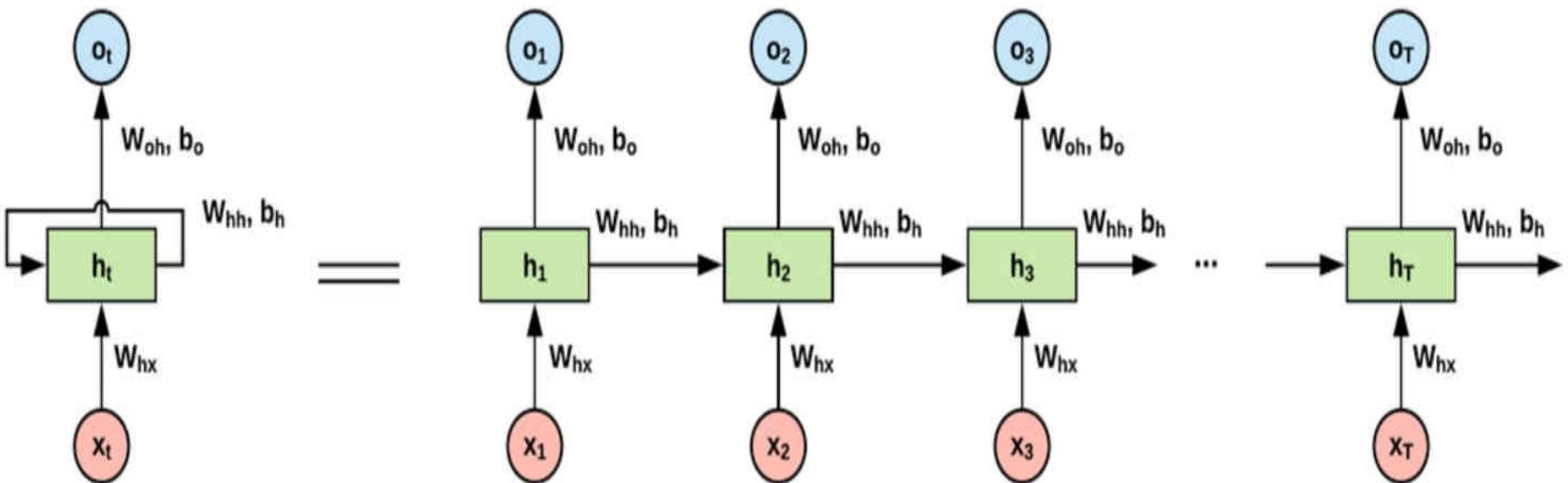
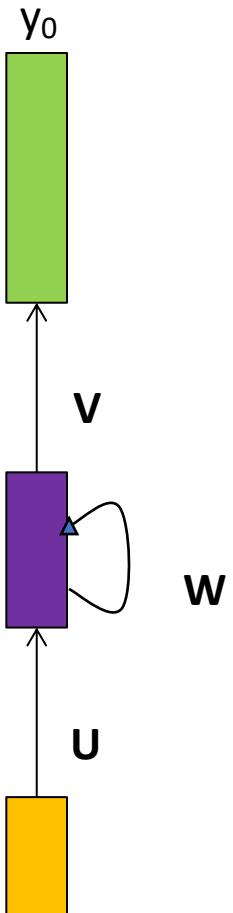
- The solution is to *add a recurrent connection in the network,*
- The function executed at each time step is
$$H_i = f(x_i U + W H_{i-1} + b)$$
$$y_i = g(H_i V + c)$$
- or
$$H_i = \tanh(x_i U + H_{i-1} W + b)$$
$$Y_i = \text{softmax}(V H_i + c)$$
- This solves all the problem,

How to account for dependencies between inputs



- The equation will remain same at all the time stamps and also the next time stamp is dependent on previous input.
- We can write the output y_i at time stamp i as function of U, V, W, x_i, H_{i-1}, b and c
- Where x_i is the current input and H_{i-1} is the hidden state of the previous input
- We set $h_{-1}=[0]$

The network can be compactly represented as



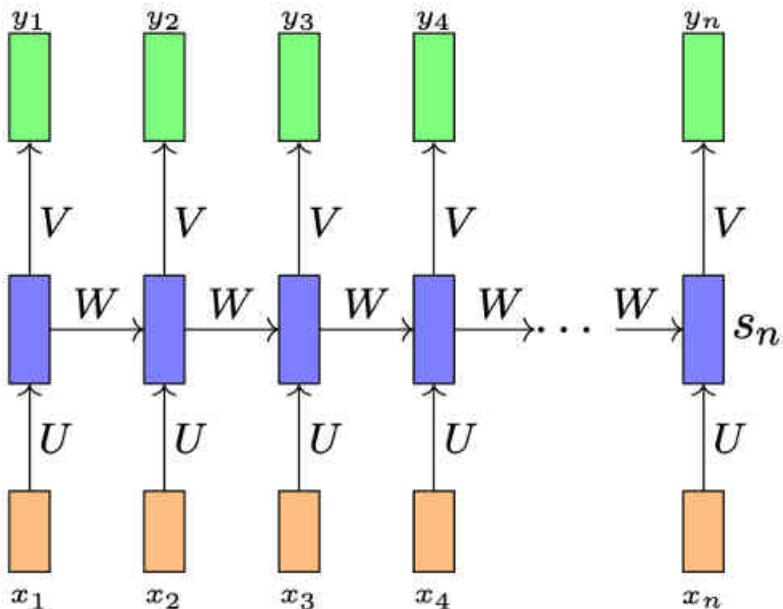
Sample RNN structure (Left) and its unfolded representation (Right)

x_0

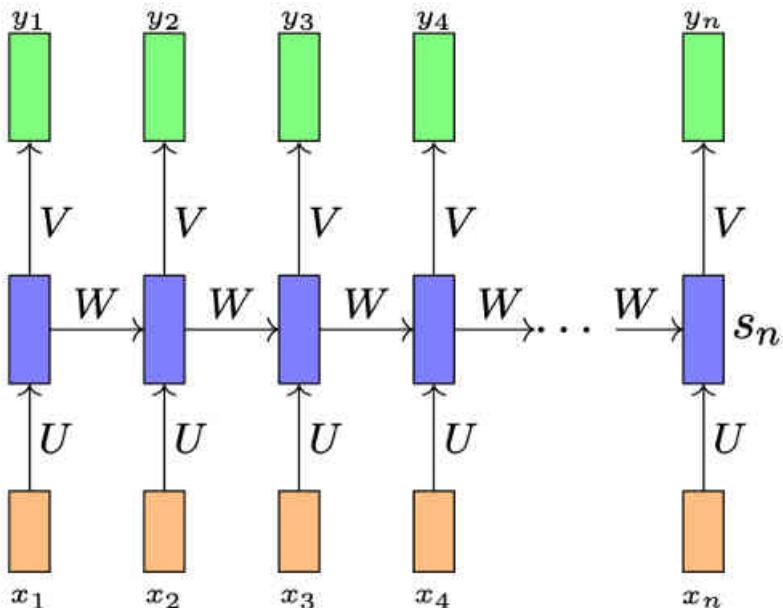
Source: <https://www.easy-tensorflow.com/tf-tutorials/recurrent-neural-networks/many-to-many>

Machine Intelligence

Final Idea



- So finally, we have input of the form $X=[x_1, x_2, \dots, x_n]$
- And a recurrent network with weight parameters U, W, V
- The following equation is computed at each time stamp to compute y_i
 - $H_i = \tanh(X_i U + H_{i-1} W + b)$
 - $Y_i = \text{softmax}(VH_i + c)$
- The parameters are shared at every time stamp



Before proceeding let us look at the dimensions of the parameters carefully

$$x_i \in \mathbb{R}^n \quad (\text{n-dimensional input})$$

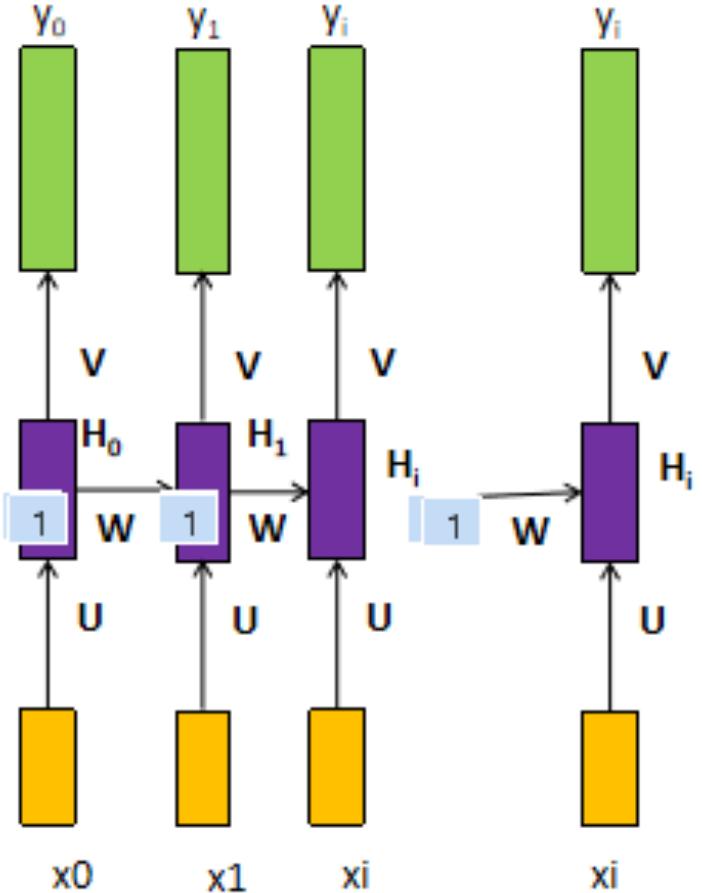
$$H_i \in \mathbb{R}^d \quad (d - \text{dimensional state})$$

$$y_i \in \mathbb{R}^k \quad (\text{say } k \text{ classes})$$

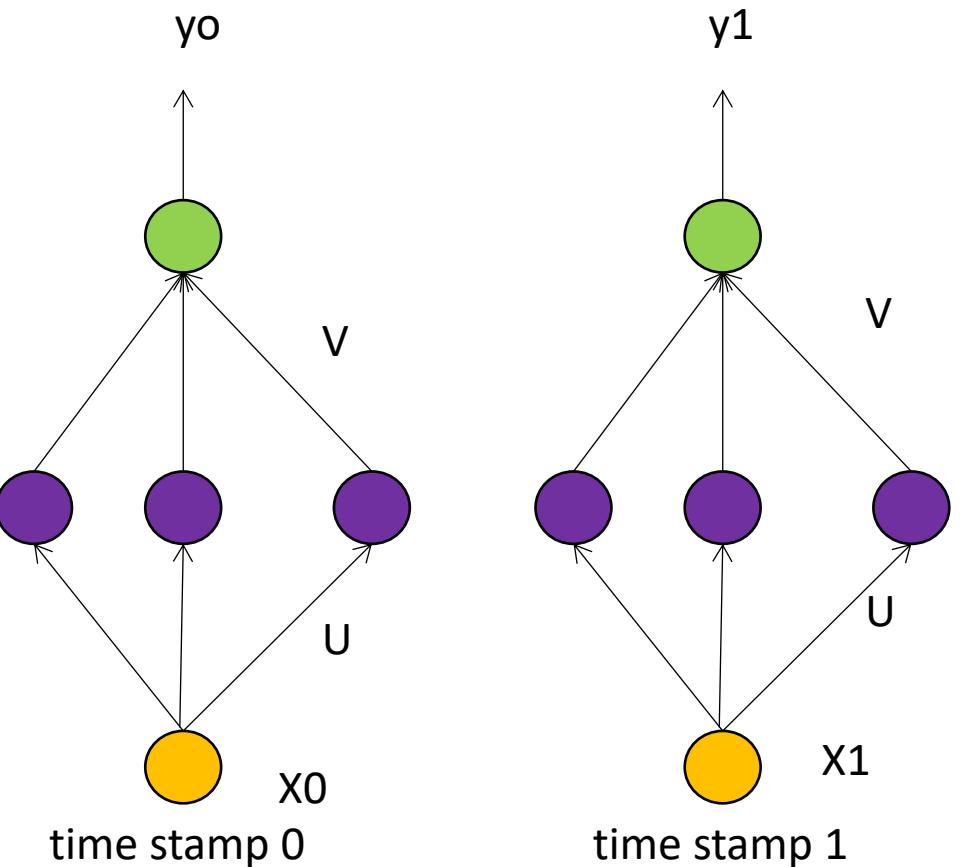
$$U \in \mathbb{R}^{n \times d}$$

$$V \in \mathbb{R}^{d \times k}$$

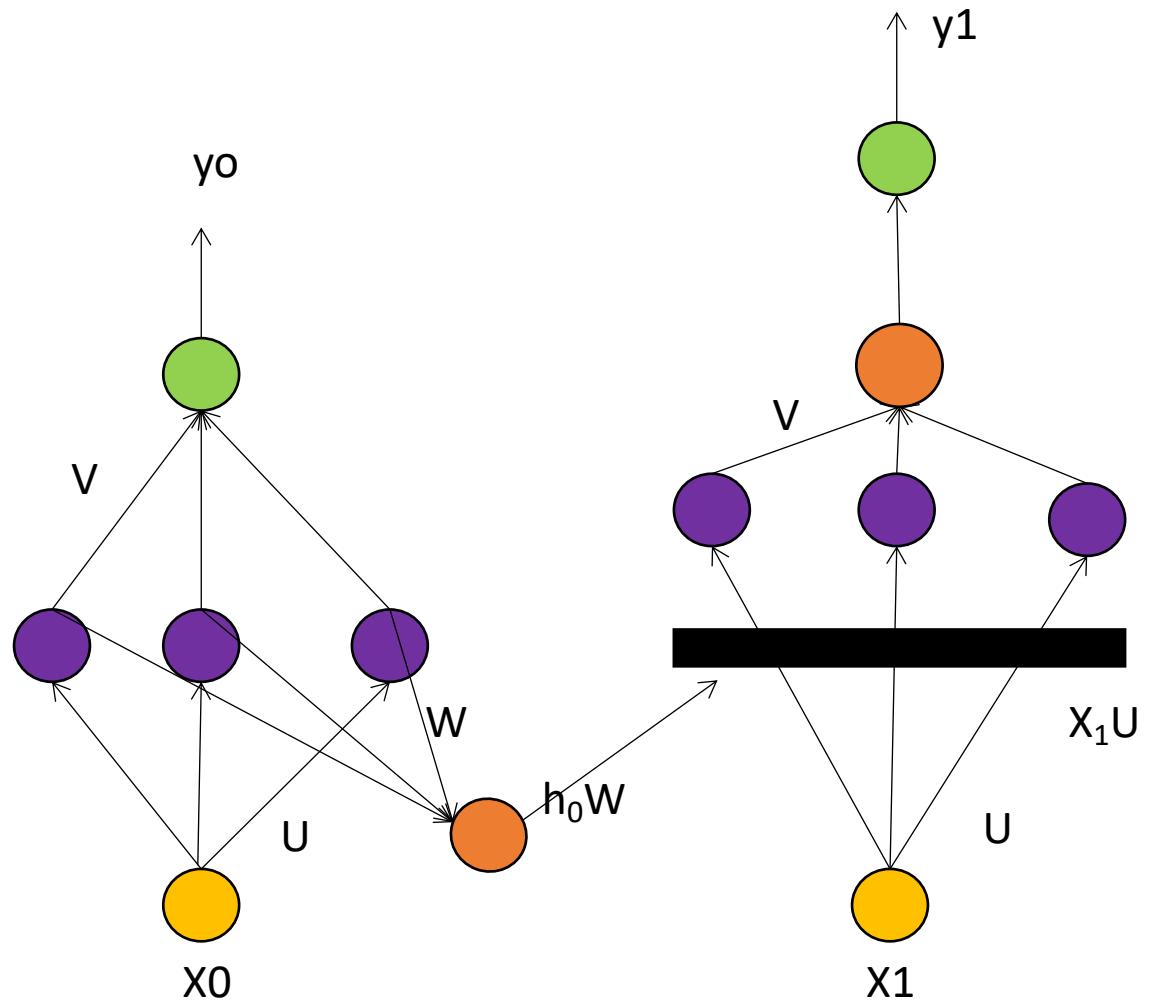
$$W \in \mathbb{R}^{d \times d}$$



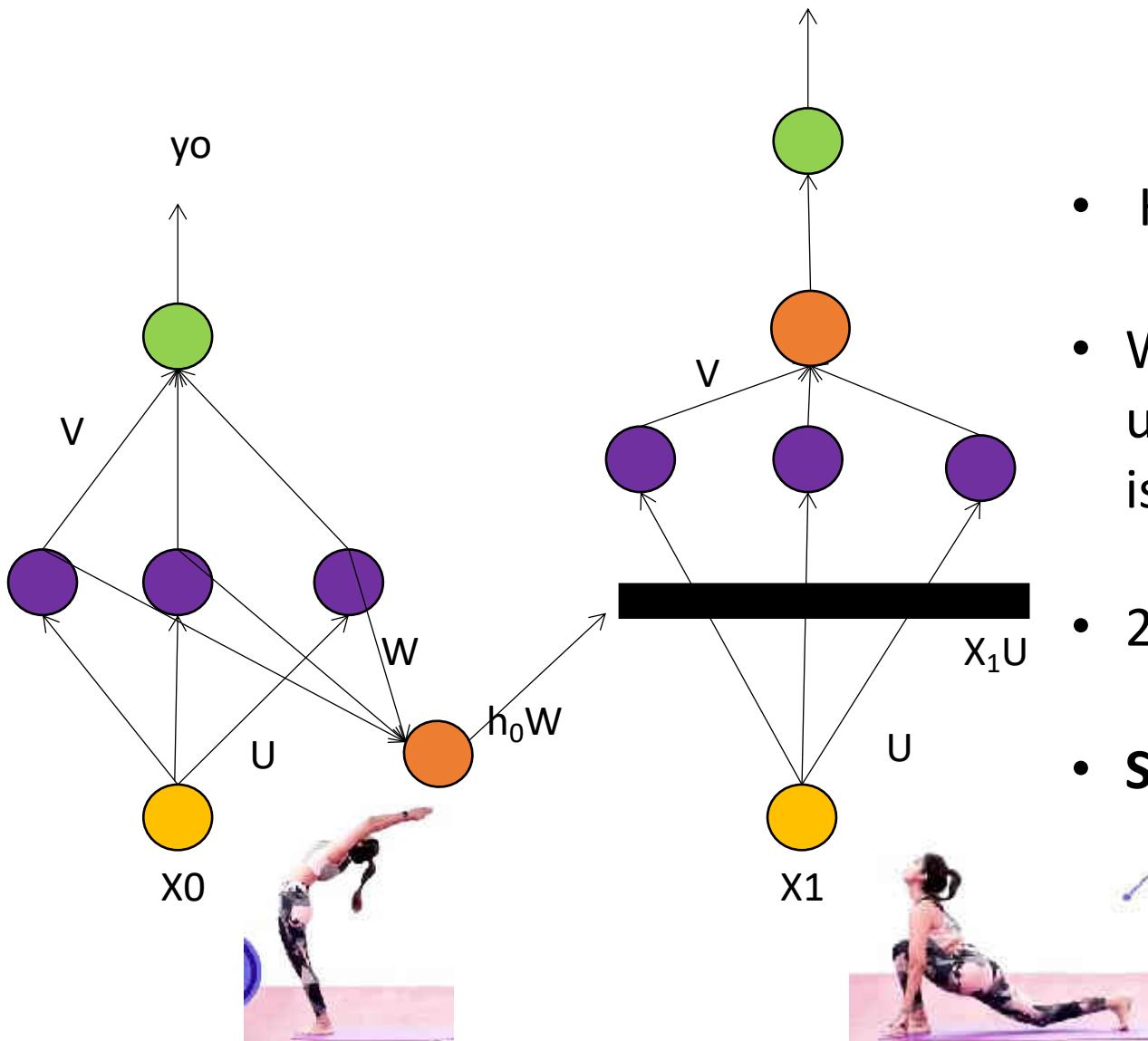
- Let us consider the first two time stamps.
- Consider the example of video of Suryanamaskar
- **let the video be sequence of images with dimension 28x28 pixels**



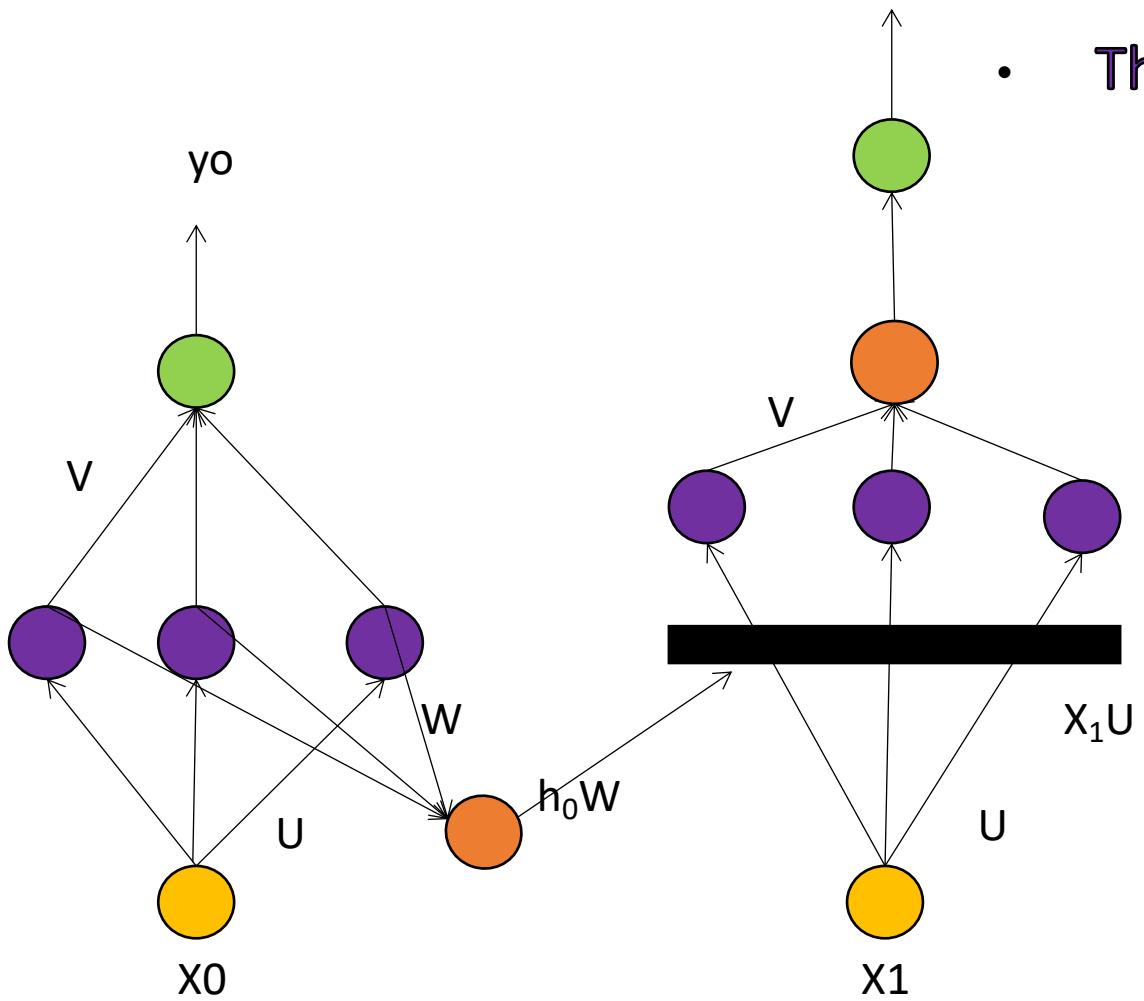
- Given figure represents network without recurrent connection
- when we add recurrent connection we get something like this



- Lets look at the **Input layer**
- We can say, $x_i \in \mathbb{R}^n$ so x_i is n-dimensional
- *Only x_i is n-dimensional not the entire X*
- Entire X will be $t \times n$ dimension, where t is the number of time stamp of that input



- Here x_i is an image and is **28 x28 px**
- We know from MNIST digit classification using ANN that the way to feed images to NN is by flattening them.
- $28 \times 28 = 784$
- So each $x_i \in \mathbb{R}^{784}$

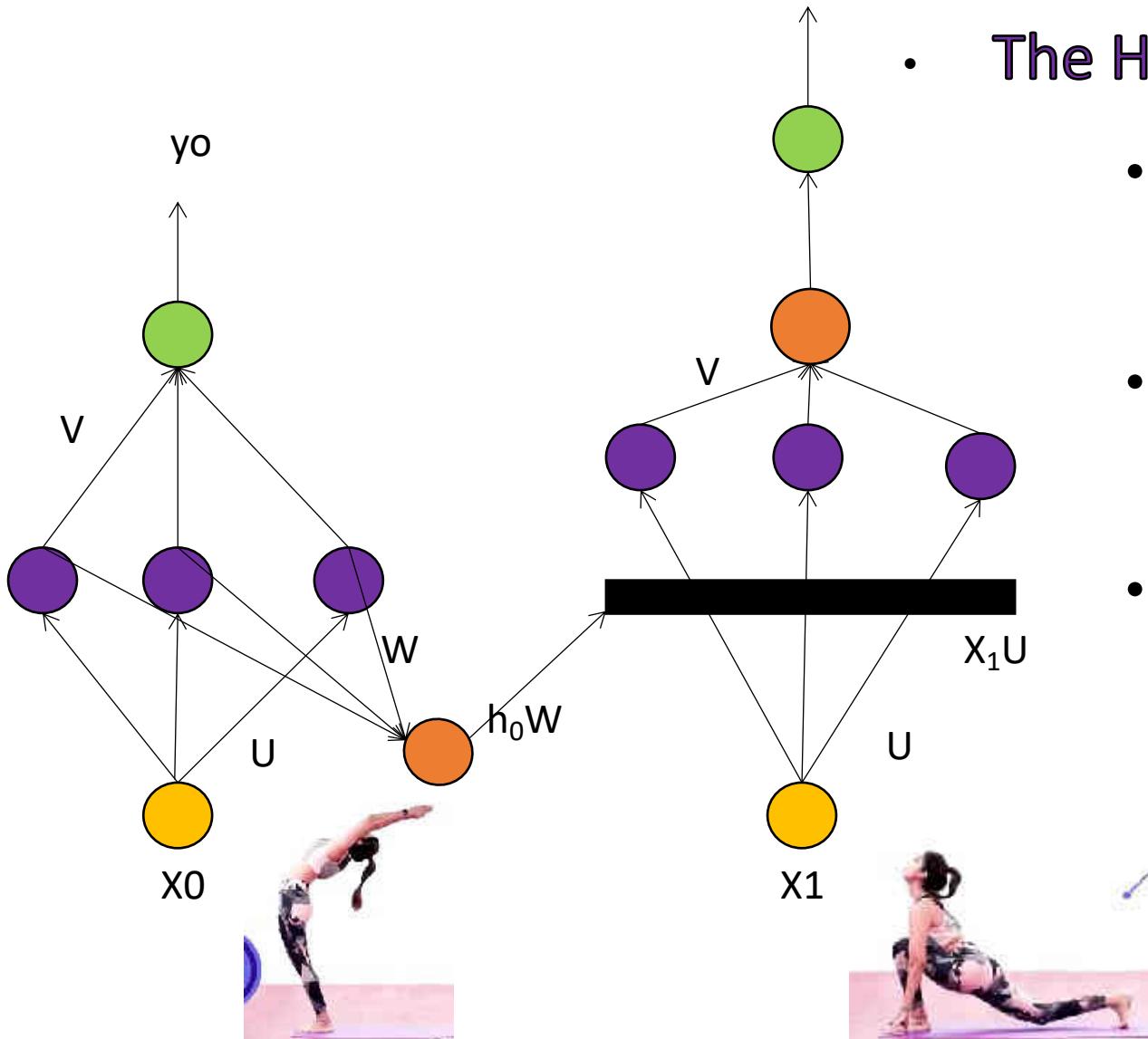


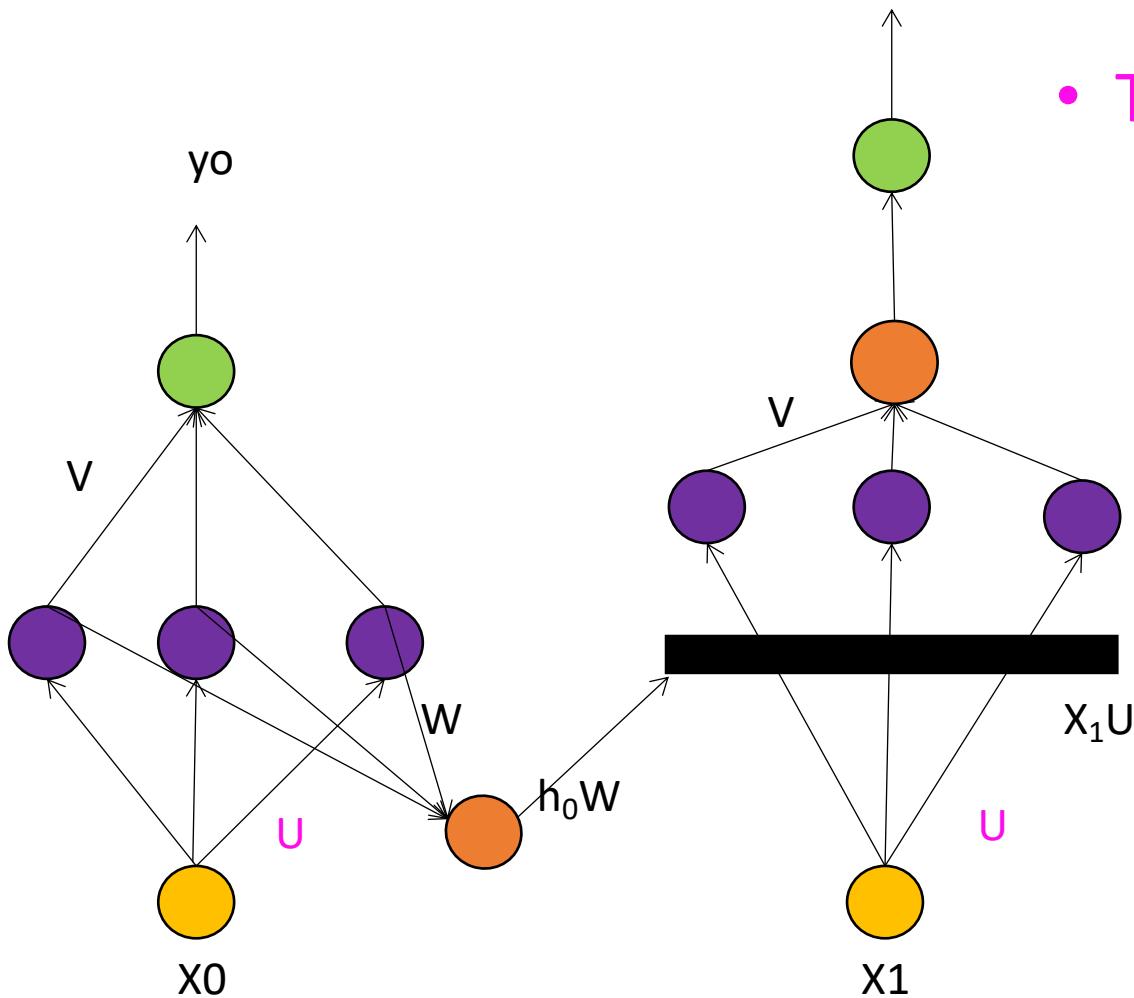
- **The Hidden Layer**

- Say, if we have d neurons in the hidden layer then the output will be a vector of $d - \text{dimension}$
- So $h_i \in \mathbb{R}^d$ so h_i is $d - \text{dimensional}$, where d is number of neurons in hidden layer

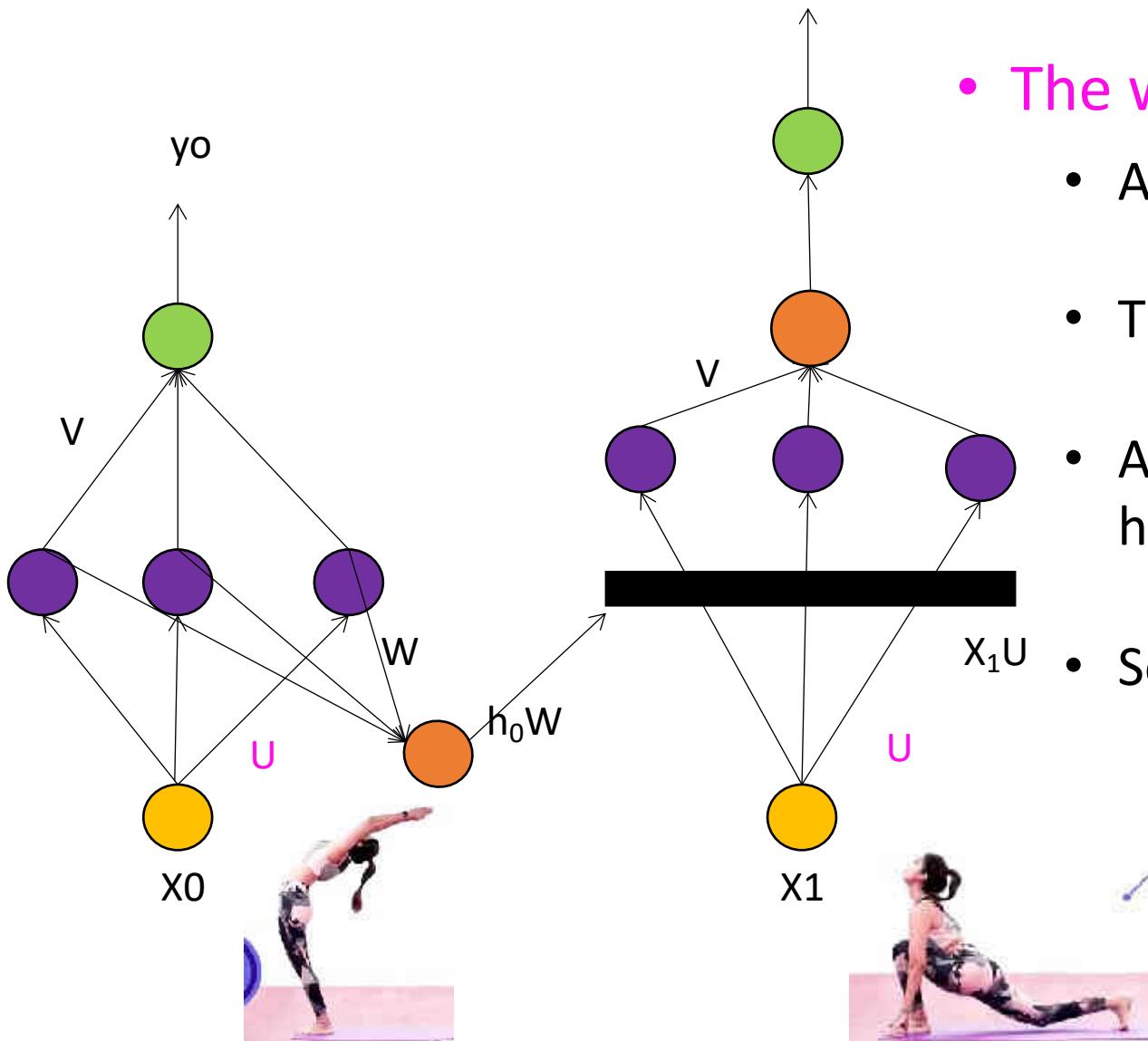
• The Hidden layer

- Say there are 200 neurons in hidden layer
- Then, the output h_i will also be a vector of dimension 200
- $h_i \in R^{200}$



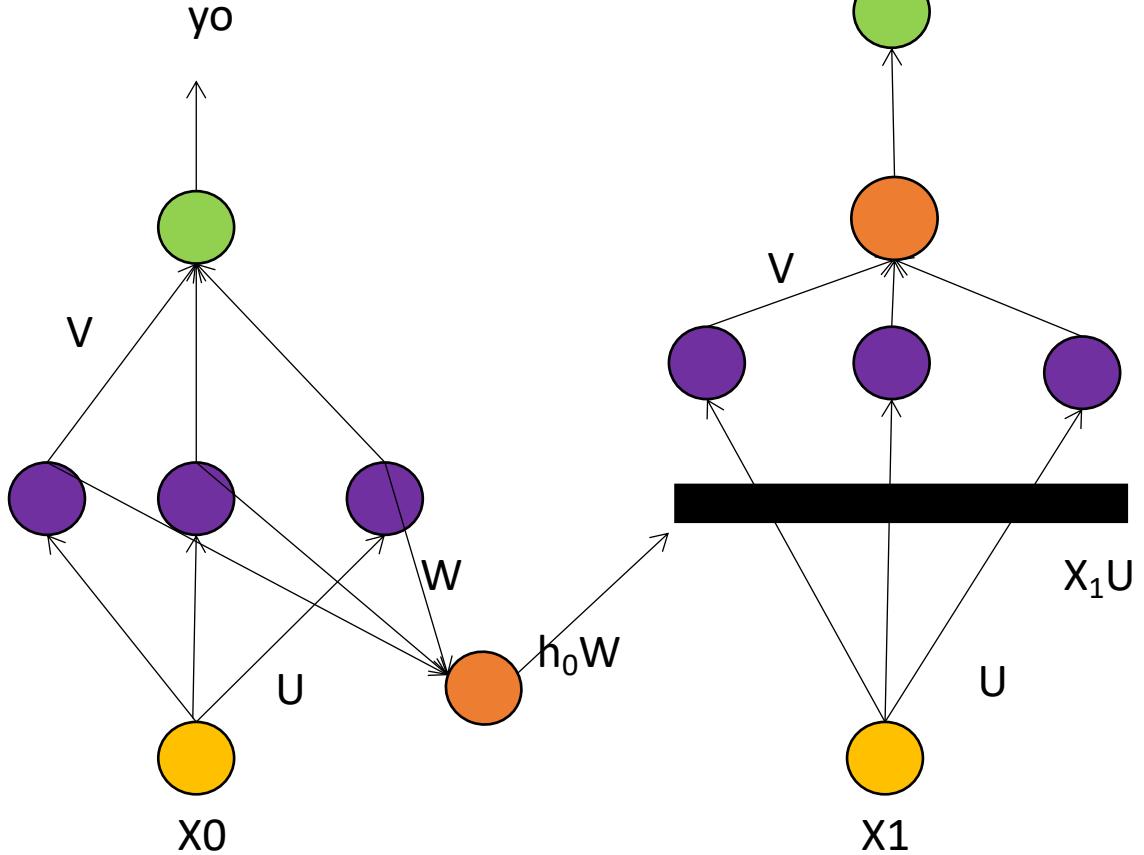


- The weight U
- If x_i is n – dim and there are d number of neurons in hidden layer , then
- $U = n \times d$ matrix

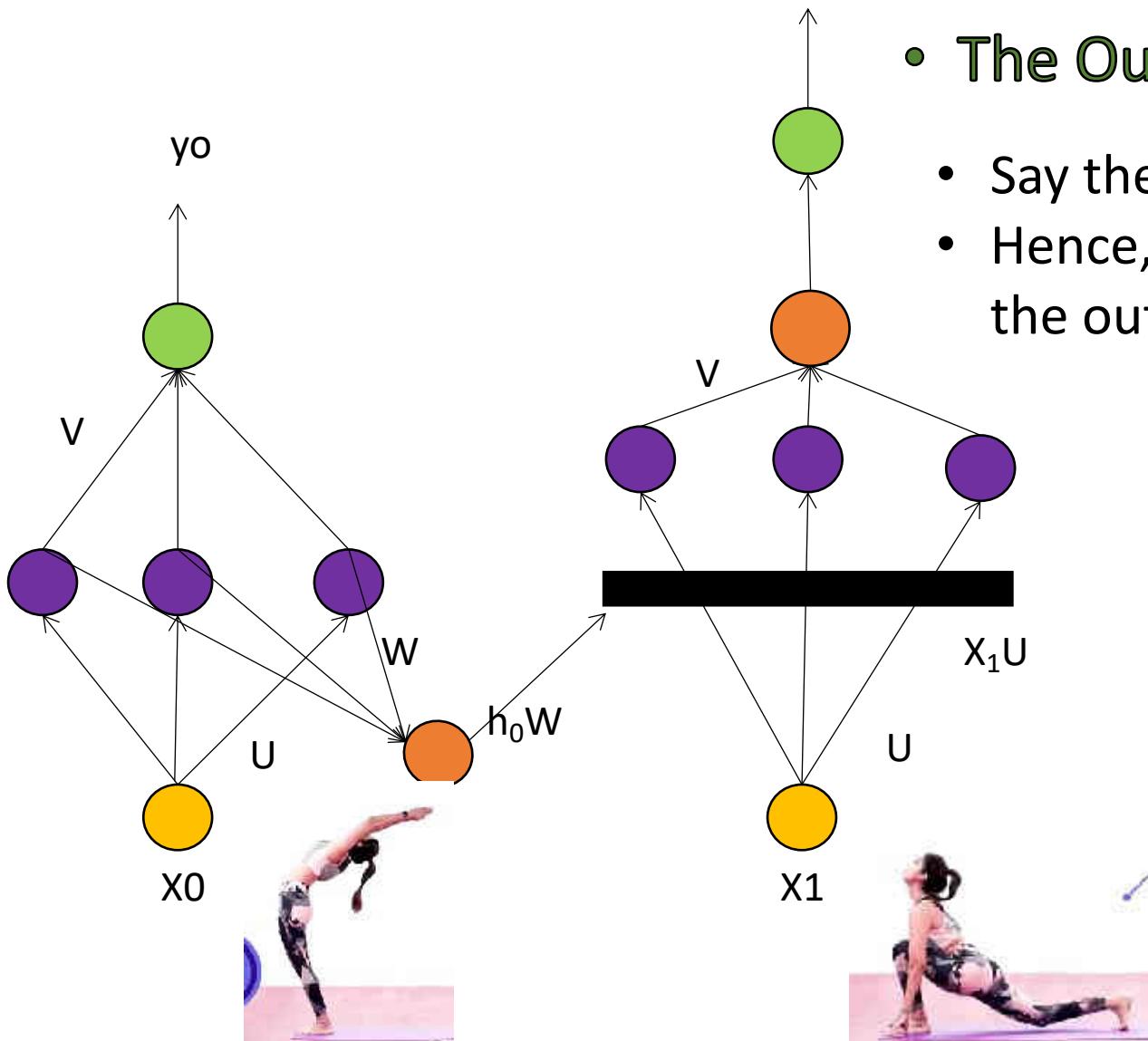


- The weight U
 - As $x_i \in \mathbb{R}^{784}$
 - That is, there are $n=784$ input neurons
 - And we considered 200 neurons in the hidden layer
 - So $U = 784 \times 200$ matrix

- The Output Layer

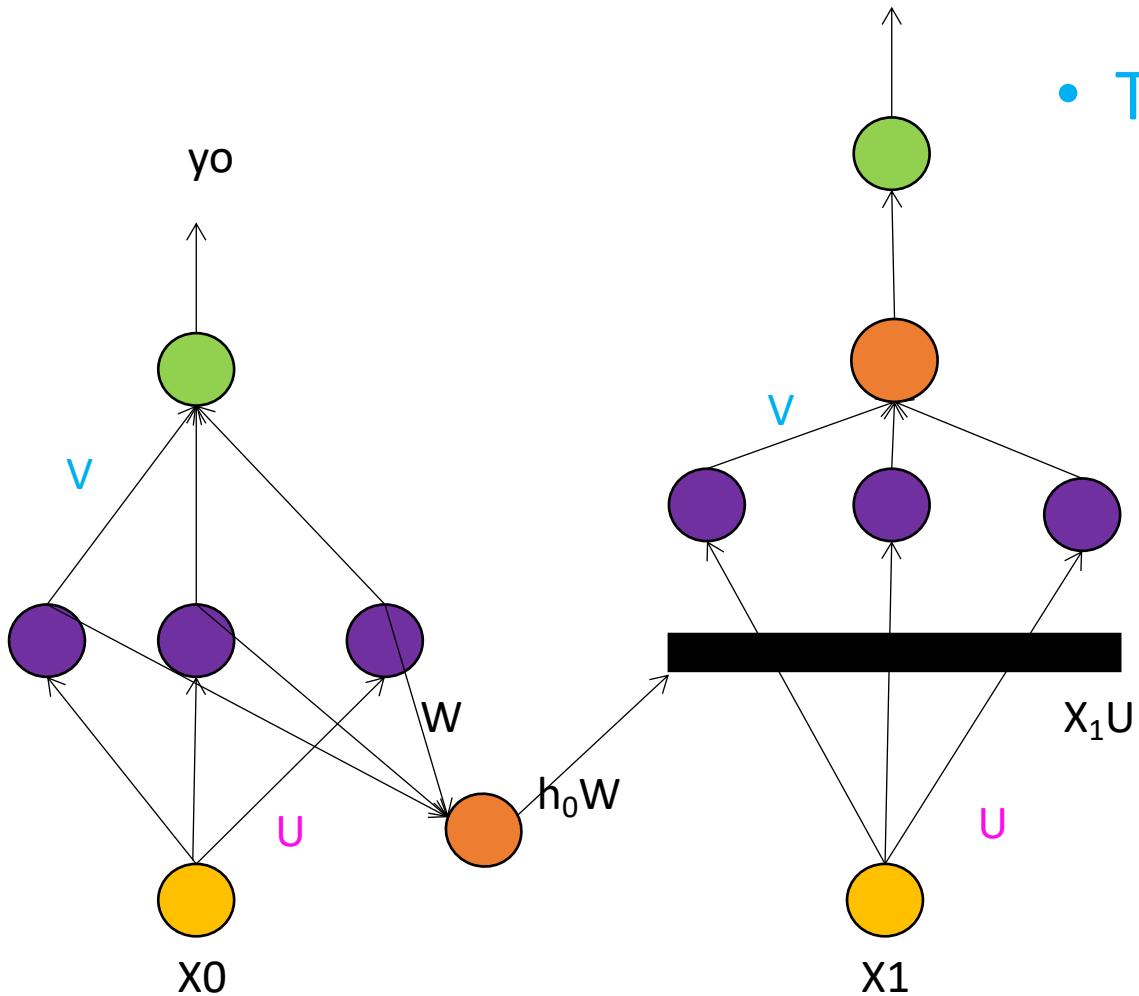


- Say for a task of classification, there are k classes
- so $y_i \in \mathbb{R}^k$ so y_i is k – dimensional, where k is number of classes



- **The Output Layer**

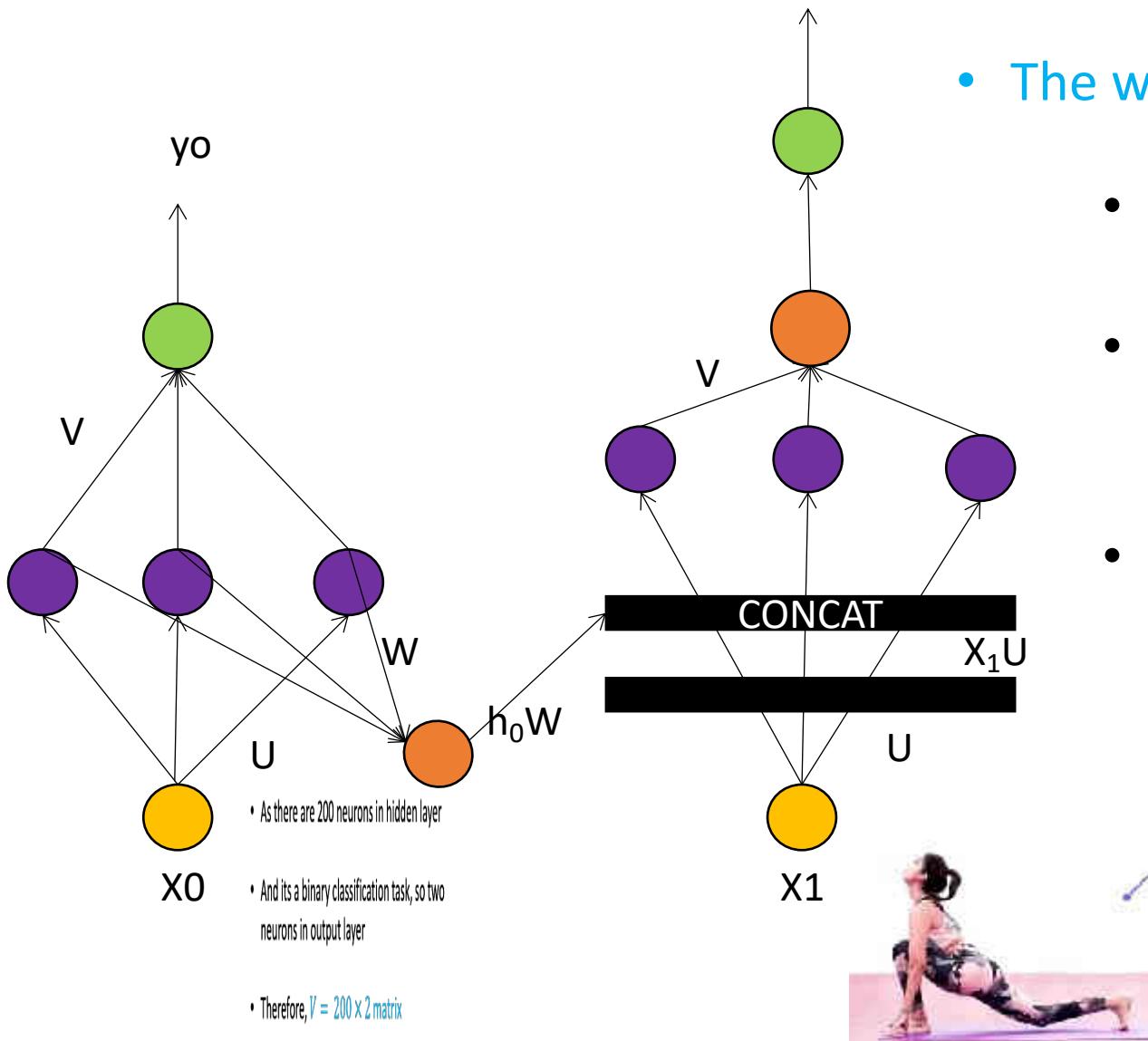
- Say the output is binary so $k= 2$
- Hence, $y_i \in \mathbb{R}^2$ and there will be 2 neurons in the output layer.



- The weight V

- If there are d neurons in hidden layer and there are k class labels. Then,

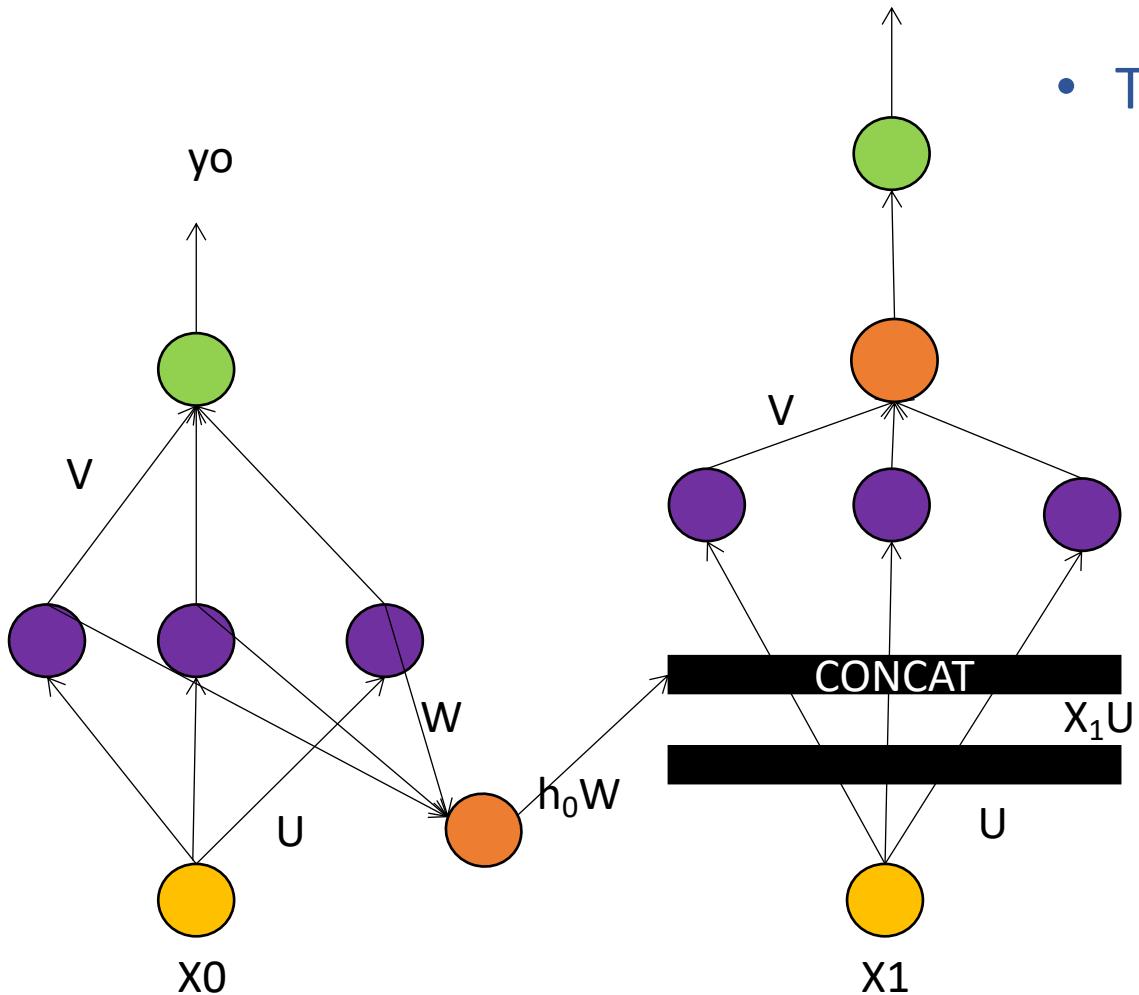
$$V = d \times k \text{ matrix}$$



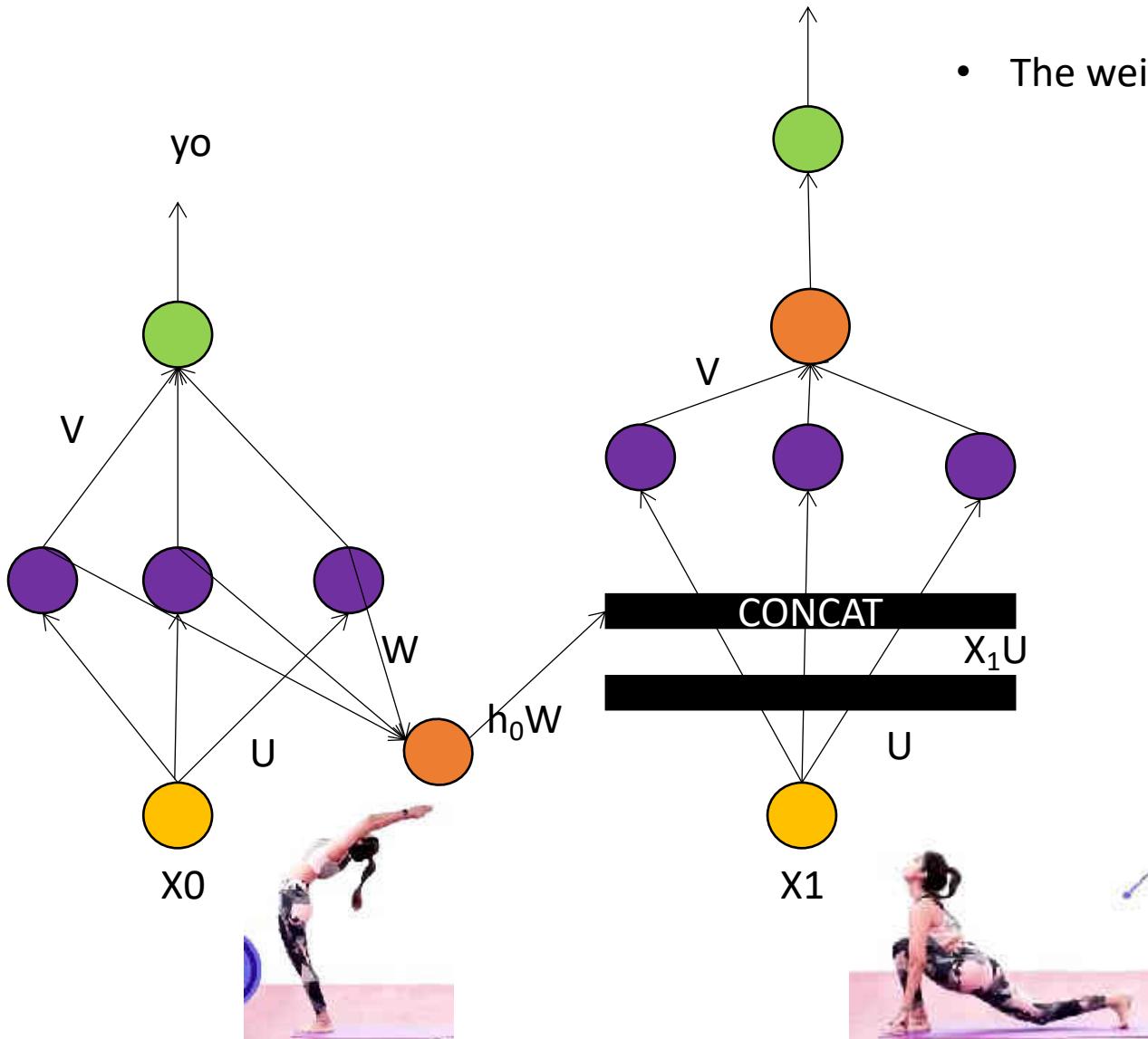
- The weight V

- As there are 200 neurons in hidden layer
- And its a binary classification task, so two neurons in output layer
- Therefore, $V = 200 \times 2$ matrix





- The weight W
- Now, there are d neurons in hidden layer ,and we know this will be added to a hidden layer output too
- So, $W = d \times d$ matrix



- The weight W
 - so output of h_0 is \mathbf{R}^{200}
 - we know $H_1 = \tanh(X_0U + H_0W)$
 - $\dim(x_0) = 1 \times 784$
 - $\dim(U) = 784 \times 200$
 - $\dim(x_0U) = 1 \times 200$
 - $\dim(H_0) = 1 \times 200$
 - Now $\dim(H_0W)$ should be 1×200
so that it can be added with x_0U
 - so $\dim(W) = 200 \times 200$

An RNN takes input of words each as a vector of length 'n', one hidden layer with $n/2$ neurons, one output layer 2 neurons, if total number of weights =161,(exclude bias). Find n

$$\text{Ans : } n(n/2) + n + (n/2)(n/2) = 161$$

$$n=14$$



THANK YOU

Dr. Arti Arya

Department of Computer Science & Engineering



PES
UNIVERSITY
ONLINE

MACHINE INTELLIGENCE

RNN- Recurrent Neural Networks

Mr. Vignesh Kamath

+

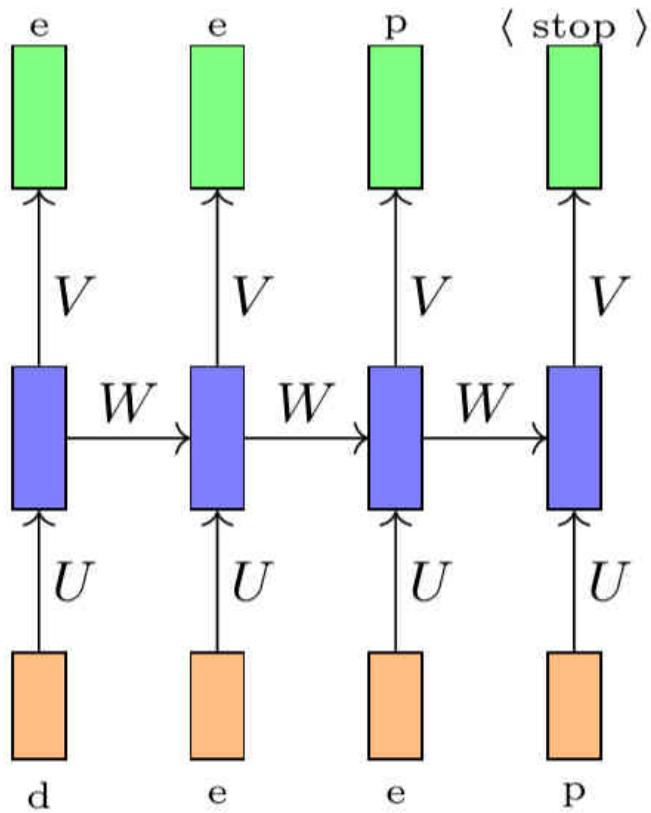
Dr. Arti Arya

Department of Computer Science and Engineering

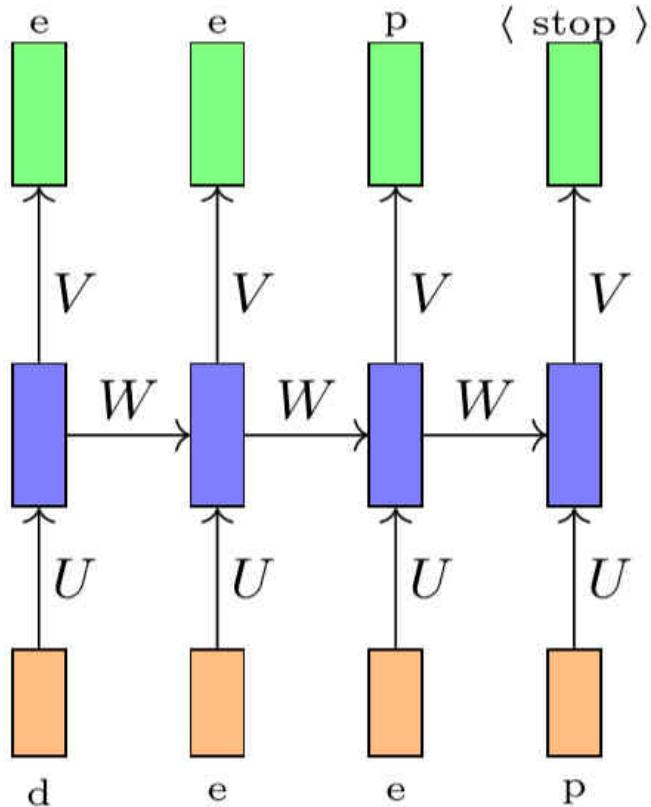
MACHINE INTELLIGENCE

RNN- Recurrent Neural Networks

Mr. Vighnesh Kamath + Dr. Arti Arya
Department of Computer Science

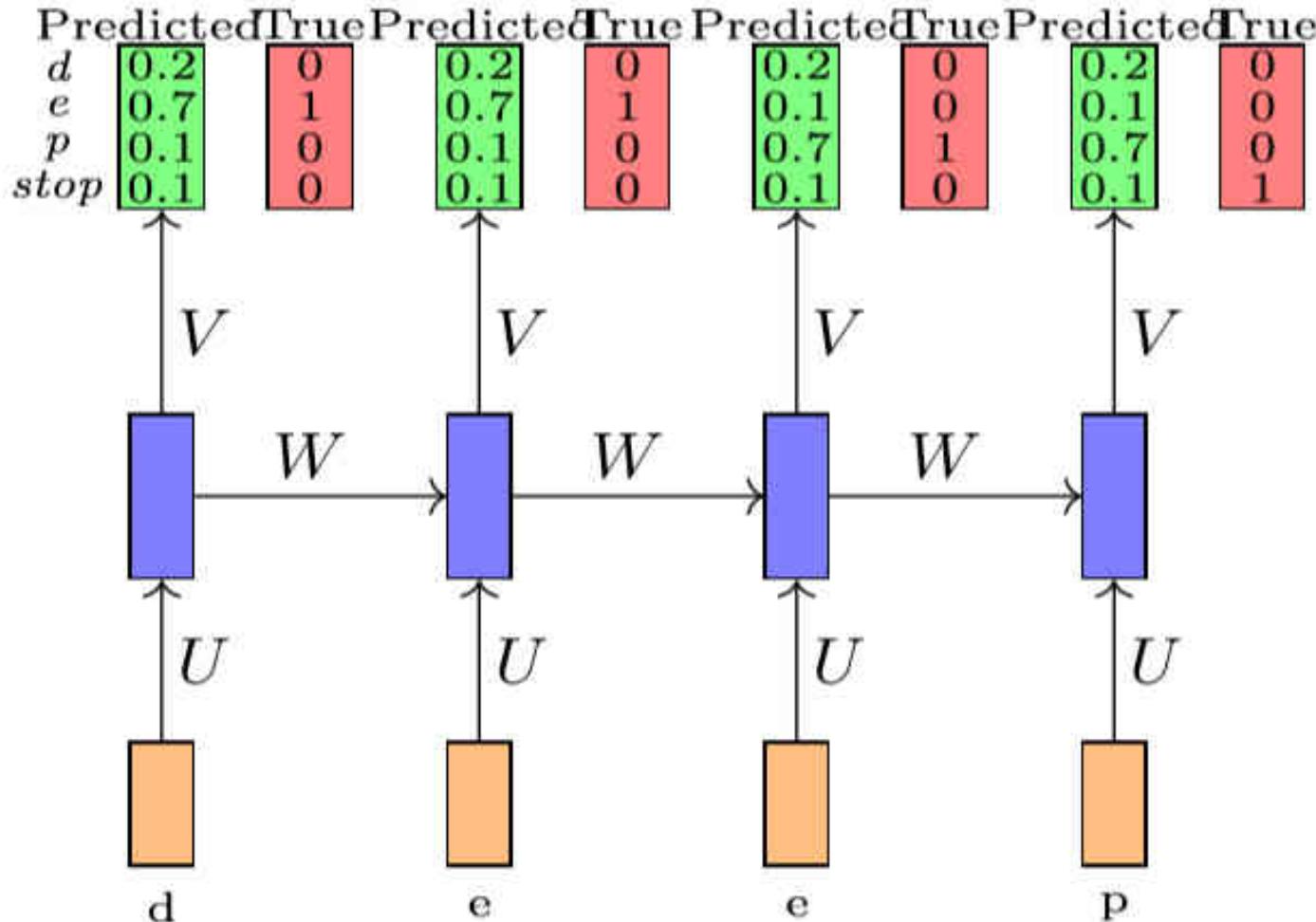


- Once again consider the **task of auto completion** (predicting the next character)
- Lets say, there are only 4 characters in the vocabulary (d, e, p, <stop>)
- At each time step, we predict one of these 4 characters
- What is a suitable output function for this task ?**
 - Softmax!!!!
- What is a suitable loss function for this task ?**
 - Cross Entropy!!!

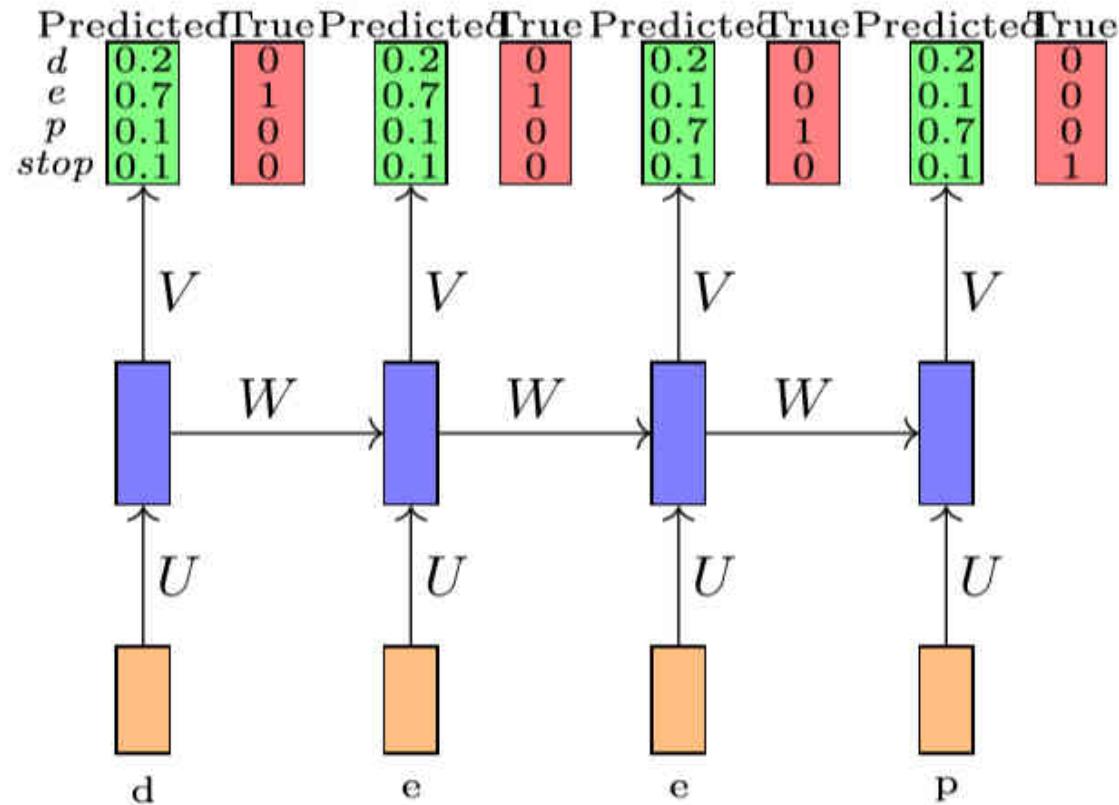


- Cross-entropy can be thought to calculate the total entropy between the distributions.
- Cross-entropy is a measure of the difference between two probability distributions for a given random variable or set of events.
- The cross-entropy between two probability distributions, such as Q from P, can be stated formally as:
$$H(P, Q)$$

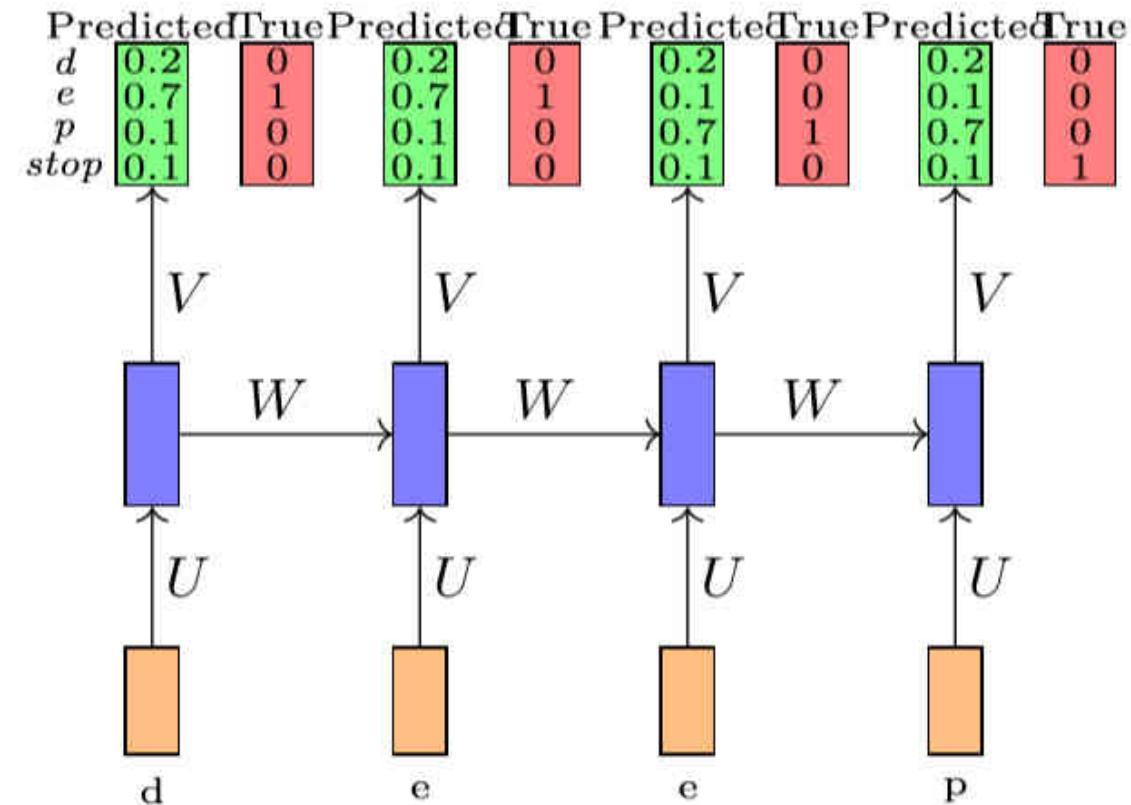
where $H()$ is the cross-entropy function, P may be the target distribution and Q is the approximation of the target distribution.



- Suppose we initialize U, V, W **randomly** and the network predicts the probabilities as shown.
- And the true probabilities are as shown.
- So here we have 2 questions:
- **What is the total loss(error) made by the model ?**
- **How do we back propagate this error and update (fine tune) the parameters $(\theta = \{U, V, W, b, c\})$ of the network ?**



- Considering the varying length for each sequential data, we also assume the parameters in each time step are the same across the whole sequential analysis.
- Otherwise it will be hard to compute the gradients. In addition, sharing the weights for any sequential length can generalize the model well.
- As for sequential labeling, we can use the maximum likelihood to estimate model parameters.
- In other words, we can minimize the negative log likelihood the objective function



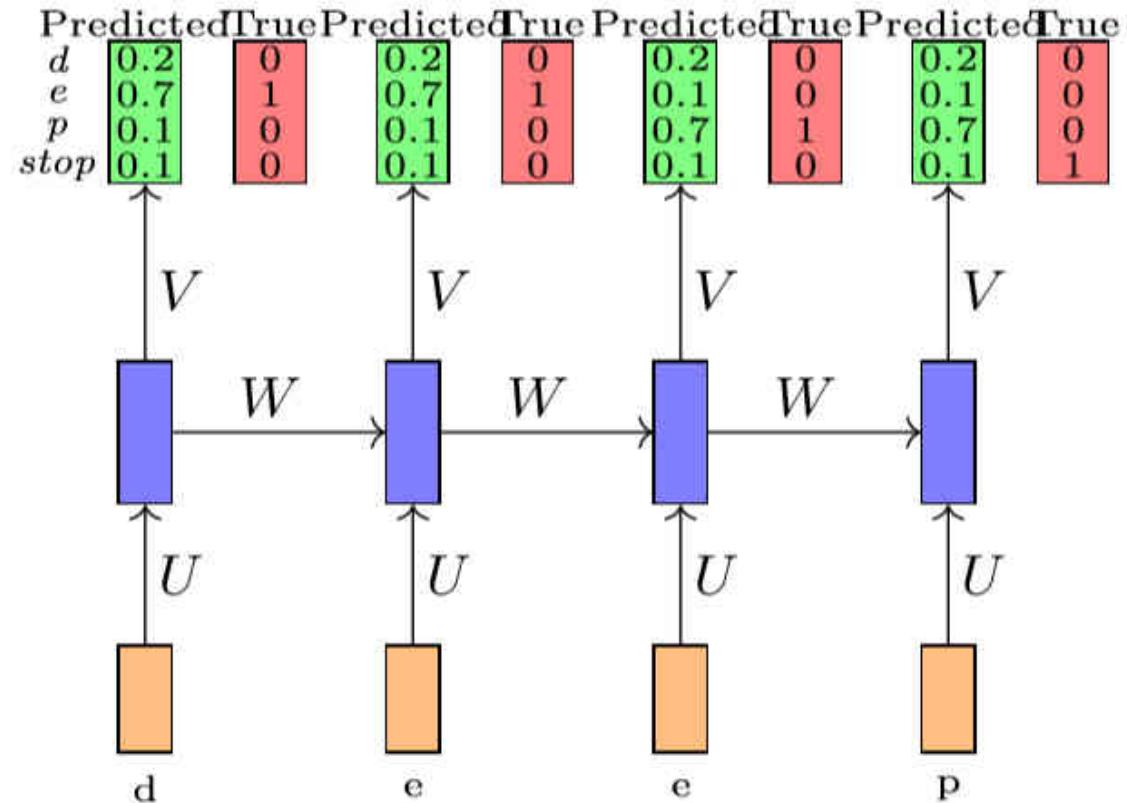
- Total loss(error), $\mathcal{L}(\theta)$ is simply the sum of the loss over all time-steps, $\mathcal{L}_t(\theta)$

$$\mathcal{L}(\theta) = \sum_{t=1}^T \mathcal{L}_t(\theta)$$

$$\mathcal{L}_t(\theta) = -y_i \log(\widehat{y}_{tc})$$

Where \widehat{y}_{tc} = predicted output of true character at time step t

T = number of timesteps



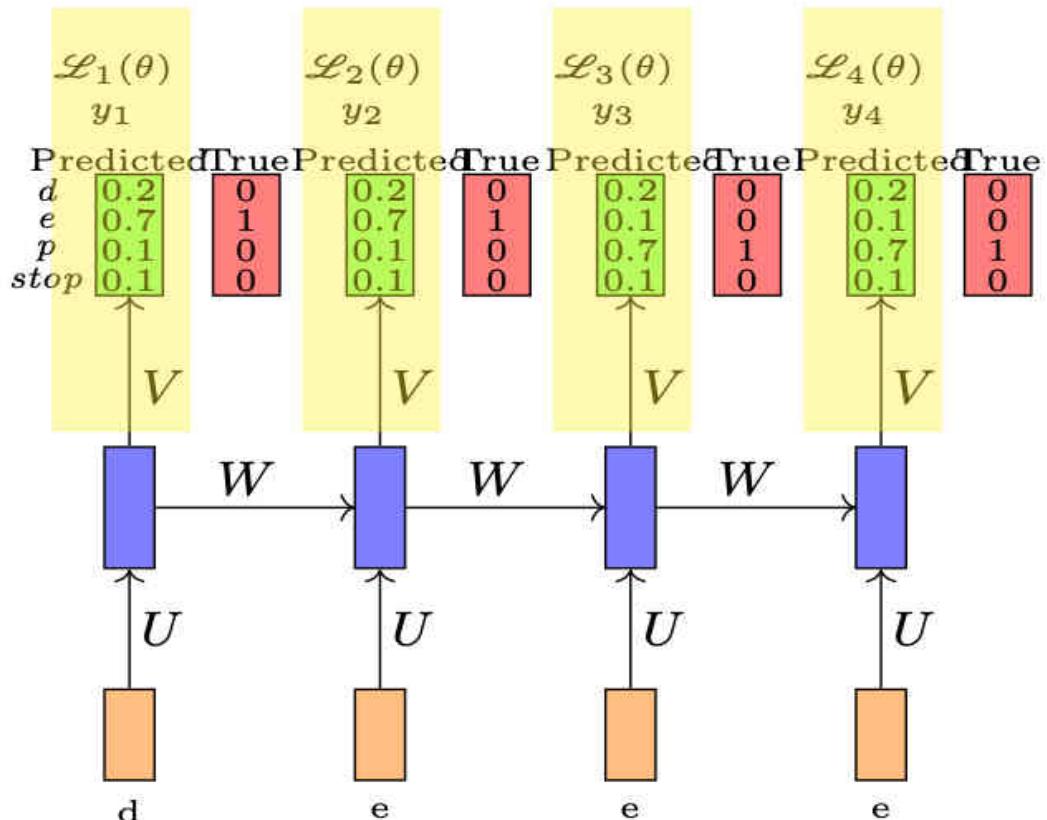
- Error can be minimized by finding optimal weights for the RNN U, W, V .
- So, We'll use gradient descent method.
- Calculate the gradient of the loss function(error) wrt all the weights, then update the weights according to this

$$V = V - \alpha \frac{\partial L}{\partial V}$$

$$W = W - \alpha \frac{\partial L}{\partial W}$$

$$U = U - \alpha \frac{\partial L}{\partial U}$$

Gradient of loss (error) wrt to V



- From the loss function \mathcal{L}
- So grad of \mathcal{L} wrt V** will be summation of grad of \mathcal{L}_i wrt V

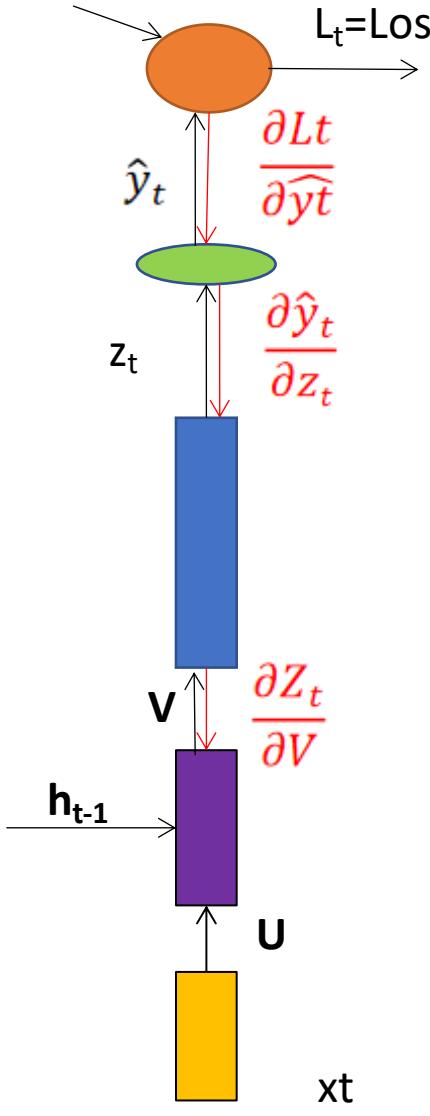
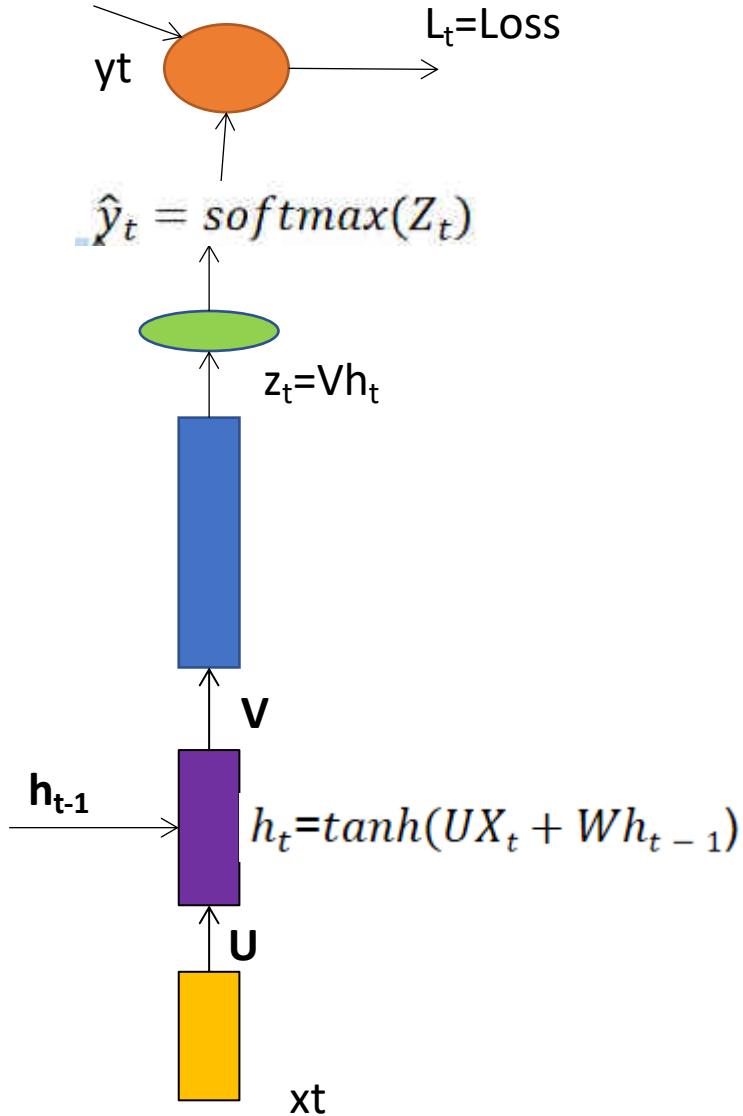
$$\frac{\partial L}{\partial V} = \frac{\partial L_0}{\partial V} + \frac{\partial L_1}{\partial V} + \frac{\partial L_2}{\partial V} + \dots + \frac{\partial L_{T-1}}{\partial V}$$

$$\frac{\partial \mathcal{L}(\theta)}{\partial V} = \sum_{t=1}^T \frac{\partial \mathcal{L}_t(\theta)}{\partial V}$$

- Each term is the summation and is the derivative of the loss w.r.t. the weights in the output layer . We have already done this in Backpropagation derivation in Unit 2.
- Let us consider a single layer , and it will be its summation over 0 to T-1 or 1 to T

Gradient of loss wrt to V

2 $L_0 = -y_0 \log(\hat{y}_0)$
 $\hat{y}_0 = \text{softmax}(h_0 V)$
 $h_0 = \tanh(U X_0 + W h_{\text{init}})$



- So for one time stamp, its shown in the figure
- So consider grad of L wrt to V , with help of chain rule, we will have the following
- solving this using chain rule we get

$$\frac{\partial L}{\partial V} = \sum_{j=0}^{T-1} (\hat{y}_j - y_j) \otimes h_j$$

- Similarly, we can compute gradient of loss wrt W

- so

$$\frac{\partial L}{\partial W} = \frac{\partial L_0}{\partial W} + \frac{\partial L_1}{\partial W} + \frac{\partial L_2}{\partial W} + \dots + \frac{\partial L_{T-1}}{\partial W}$$

-

or

$$\frac{\partial L}{\partial W} = \sum_{j=0}^{T-1} \frac{\partial L_j}{\partial W}$$

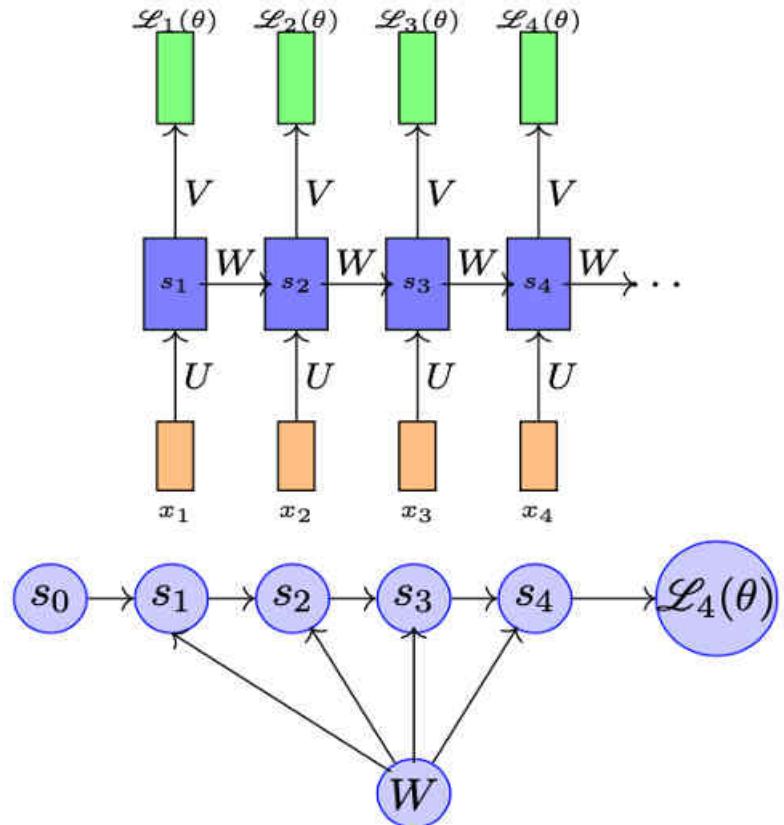
$$\frac{\partial \mathcal{L}(\theta)}{\partial W} = \sum_{t=1}^T \frac{\partial \mathcal{L}_t(\theta)}{\partial W}$$

- By the chain rule of derivatives, we

know that $\frac{\partial \mathcal{L}_t(\theta)}{\partial W}$ is obtained by summing gradients along all the paths from $L_t(\theta)$ to W

- What are the paths connecting $\mathcal{L}_t(\theta)$ to W ?

- Let us see this by considering $\mathcal{L}_4(\theta)$



$\mathcal{L}_4(\theta)$ depends on H_4

H_4 in turn depends on H_3 and W ,
 H_3 in turn depends on H_2 and W ,
 H_2 in turn depends on H_1 and W ,
 H_1 in turn depends on H_0 and W where H_0 is a constant starting state.

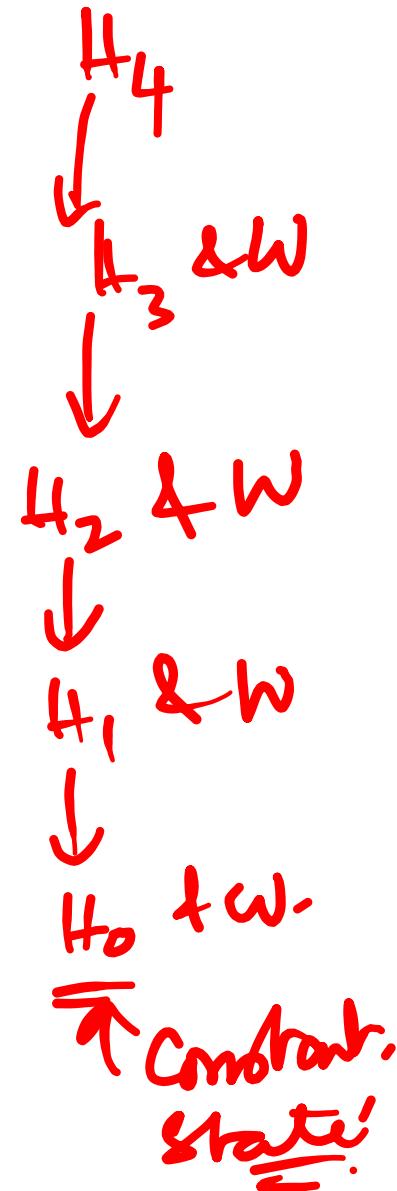
We've an ordered network here.

In ordered NW each state variable is computed one at a time in a specified order (manner).

So,

$$\frac{\partial L_4(\theta)}{\partial w} = \frac{\partial L_4(\theta)}{\partial h_4} \times \frac{\partial h_4}{\partial w}$$

↑ can be computed as computed in Backprop in ANN.



For $\frac{\partial H_4}{\partial W}$ → Since this is an ordered NW,
we can't compute $\frac{\partial H_4}{\partial W}$ simply by
considering H_3 Constant. ∵ it also depends
on W :

So, the total derivative has two parts

$(\frac{\partial^+ H_4}{\partial W})$ Explicit ✓
(treating all i/p as constant except W)

$(\frac{\partial^- H_4}{\partial W})$ Implicit ✓
(Summing over all indirect paths from H_4 to W).

$$\frac{\partial \tilde{H}_4}{\partial W} = \underbrace{\frac{\partial^+ H_4}{\partial W}}_{\text{Explicit}}$$

Explicit

$$\frac{\partial H_4}{\partial H_3} \cdot \underbrace{\frac{\partial H_3}{\partial W}}_{\text{Implicit}}$$

Implicit

$$= \frac{\partial^+ H_4}{\partial W} + \frac{\partial H_4}{\partial H_3} \left[\frac{\partial^+ H_3}{\partial W} + \frac{\partial H_3}{\partial H_2} \cdot \frac{\partial H_2}{\partial W} \right]$$

$$= \frac{\partial^+ H_4}{\partial W} + \frac{\partial H_4}{\partial H_3} \left[\frac{\partial^+ H_3}{\partial W} + \frac{\partial H_3}{\partial H_2} \left(\frac{\partial^+ H_2}{\partial W} + \frac{\partial H_2}{\partial H_1} \frac{\partial H_1}{\partial W} \right) \right].$$

✓ $\neq \frac{\partial^+ H_4}{\partial W} + \frac{\partial H_4}{\partial H_3} \cdot \frac{\partial^+ H_3}{\partial W} +$

$$\boxed{\frac{\partial H_4}{\partial H_3} \frac{\partial H_3}{\partial H_2} \cdot \frac{\partial^+ H_2}{\partial W} +}$$

$$\frac{\partial H_4}{\partial H_3} \frac{\partial H_3}{\partial H_2} \cdot \frac{\partial H_2}{\partial H_1} \frac{\partial H_1}{\partial W}$$

$$\frac{\partial H_4}{\partial W} = \left\{ \frac{\partial H_4}{\partial H_4} \cdot \frac{\partial^+ H_4}{\partial W} + \frac{\partial H_4}{\partial H_3} \frac{\partial^+ H_3}{\partial W} + \frac{\partial H_4}{\partial H_2} \cdot \frac{\partial^+ H_2}{\partial W} \right. \\ \left. + \frac{\partial H_4}{\partial H_1} \cdot \frac{\partial^+ H_1}{\partial W} \right\}$$

So,

$$\frac{\partial L_4(\theta)}{\partial W} = \frac{\partial L_4(\theta)}{\partial H_4} \cdot \frac{\partial H_4}{\partial W} \\ = \frac{\partial L_4(\theta)}{\partial H_4} \sum_{k=1}^4 \frac{\partial H_4}{\partial H_k} \cdot \frac{\partial^+ H_k}{\partial W}.$$

$$\frac{\partial L_t(\theta)}{\partial W} = \frac{\partial L_t(\theta)}{\partial H_t} \underbrace{\sum_{k=1}^t}_{\oplus} \frac{\partial H_t}{\partial H_k} \cdot \frac{\partial^+ H_k}{\partial W} \checkmark$$

This algo. is called BACKPROPAGATION

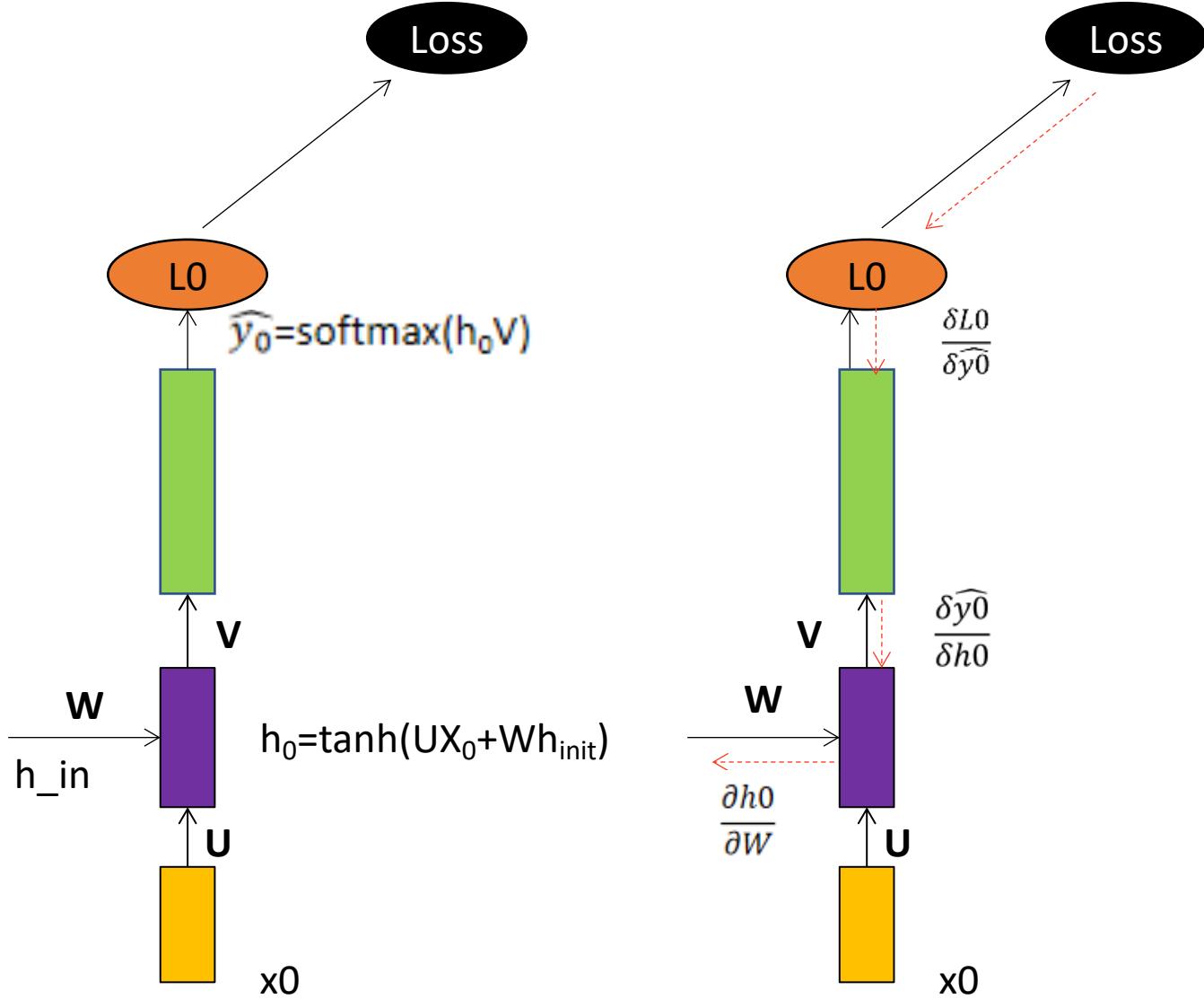
Through TIME (BPTT) as we

backpropagate thro' all time

steps.

Machine Intelligence

Grad of loss wrt to W(Pls look into the following slides if you wish to bcoz we have already discussed for L4(θ)

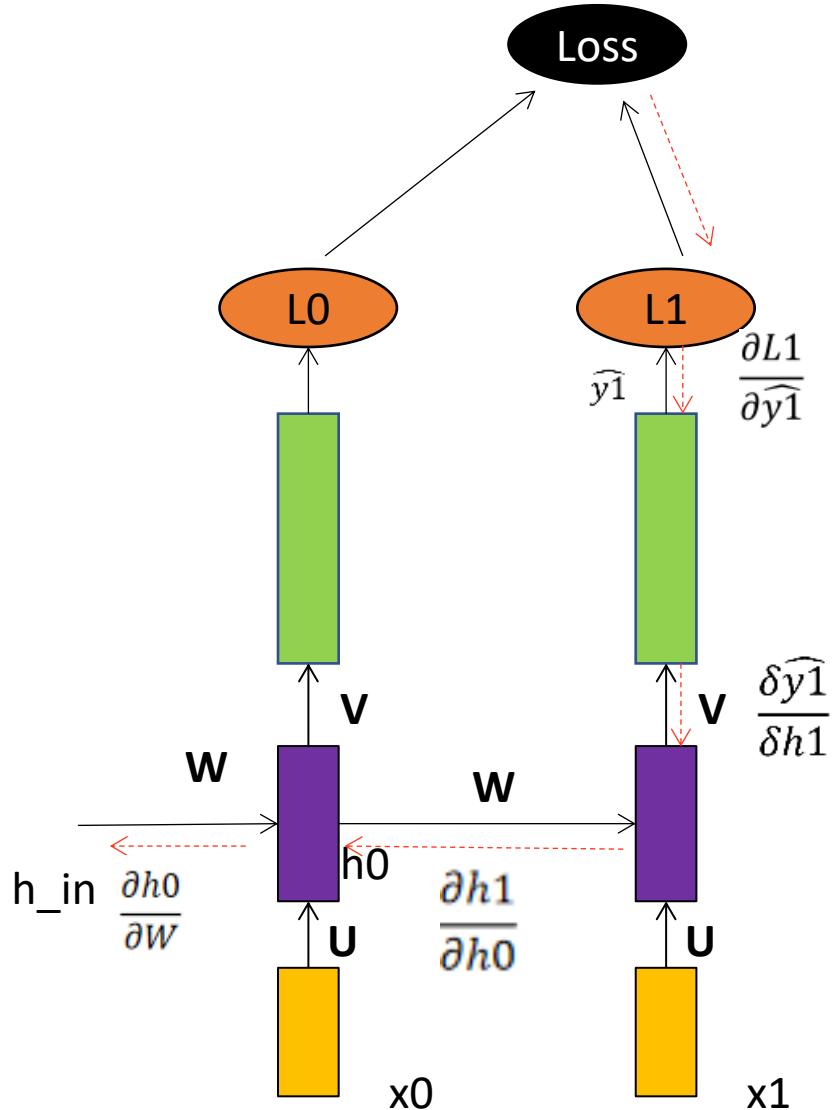


let us start with time stamp t=0

- 2: $L_0 = -y_0 \log(\hat{y}_0)$
- $\hat{y}_0 = \text{softmax}(h_0V)$
- $h_0 = \tanh(Ux_0 + Wh_{\text{init}})$

since L_0 and W are not directly related we will use chain rule

$$\frac{\partial L_0}{\partial W} = \frac{\partial L_0}{\partial \hat{y}_0} \frac{\partial \hat{y}_0}{\partial h_0} \frac{\partial h_0}{\partial W}$$

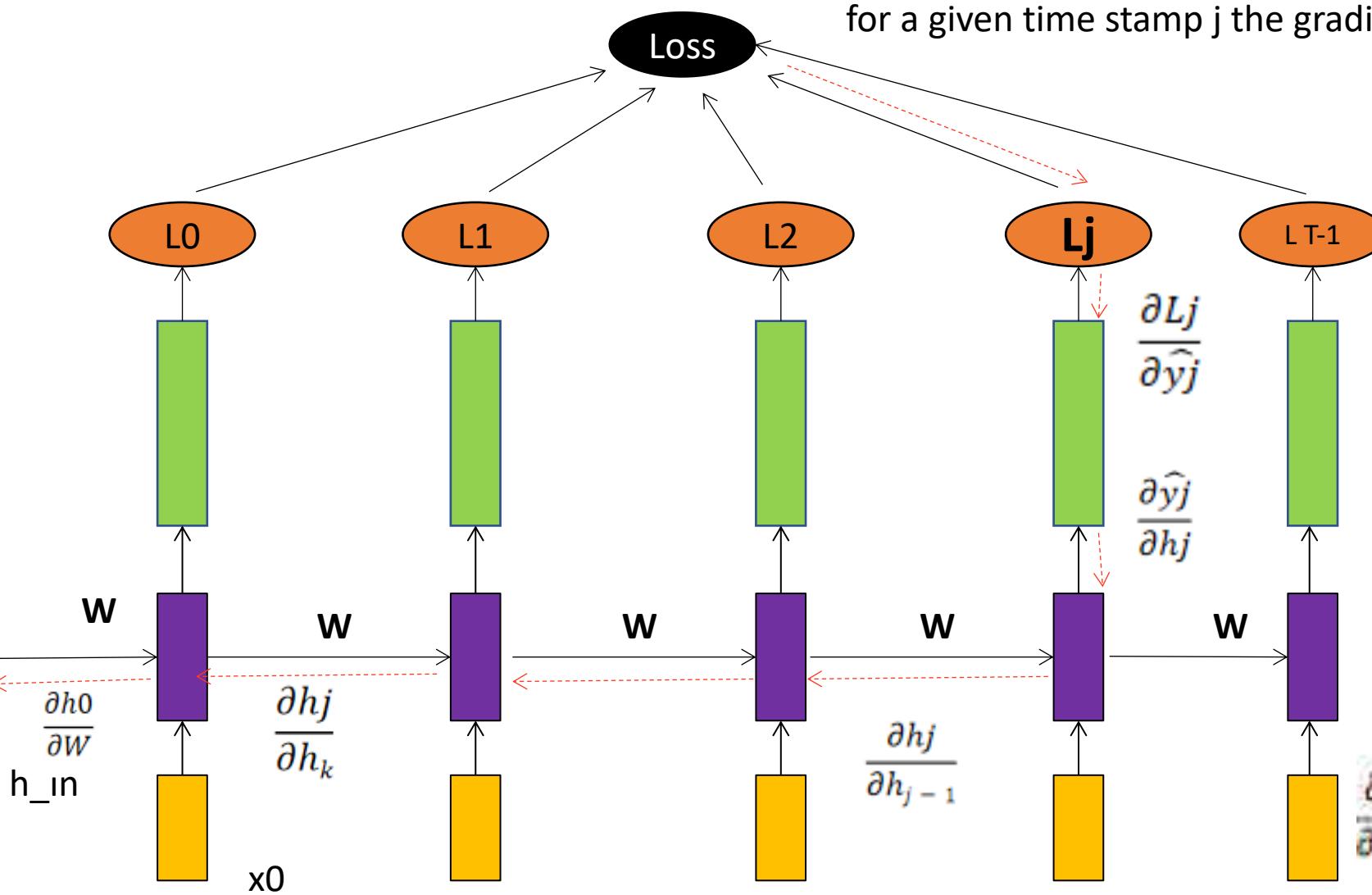


let us do for time stamp t=1

$$\frac{\partial L_1}{\partial W} = \frac{\partial L_1}{\partial \widehat{y}_1} \frac{\partial \widehat{y}_1}{\partial h_1} \frac{\partial h_1}{\partial W}$$

- but h_1 is not a independent variable its dependent on h_0
- Hence we continue our chain rule

$$\frac{\partial L_1}{\partial W} = \frac{\partial L_1}{\partial \widehat{y}_1} \frac{\partial \widehat{y}_1}{\partial h_1} \frac{\partial h_1}{\partial W} + \frac{\partial L_1}{\partial \widehat{y}_1} \frac{\partial \widehat{y}_1}{\partial h_1} \frac{\partial h_1}{\partial h_0} \frac{\partial h_0}{\partial W}$$



- This is known as Back propagation through time
- where your gradients flow not just till the current time step ,instead till the initial time step

we can write the generalized equation

$$\frac{\partial L_j}{\partial W} = \sum_{k=0}^j \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \left(\prod_{m=k+1}^j \frac{\partial h_m}{\partial h_{m-1}} \right) \frac{\partial h_k}{\partial W}$$

summing to all time step

$$\frac{\partial L}{\partial W} = \sum_{j=0}^{T-1} \sum_{k=0}^j \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \left(\prod_{m=k+1}^j \frac{\partial h_m}{\partial h_{m-1}} \right) \frac{\partial h_k}{\partial W}$$

Grad of loss wrt to U (Similar to W)

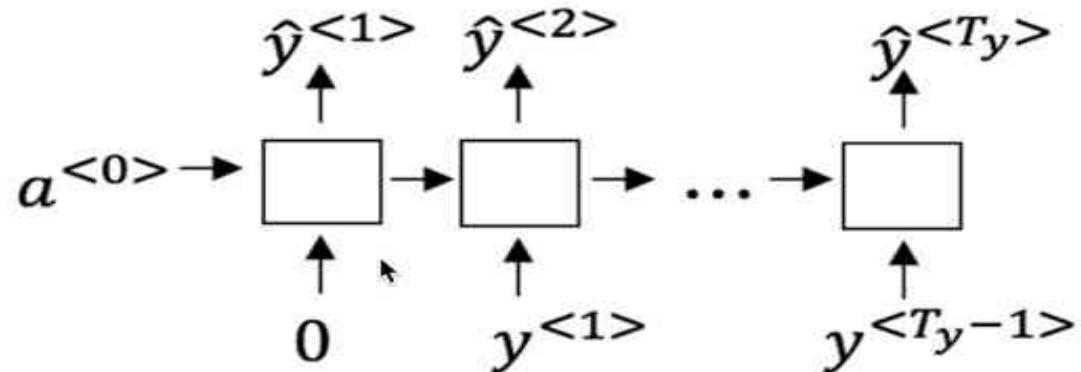
It will be same as W ,practice and trace the flow of gradients

$$\frac{\partial L}{\partial U} = \sum_{j=0}^{T-1} \sum_{k=0}^j (\hat{y}_j - y_j) \prod_{m=k+1}^j W^T \text{diag}(1 - \tanh^2(W h_{m-1} + U x_m)) \otimes x_k$$

Machine Intelligence

Quiz

You are training this RNN language model.



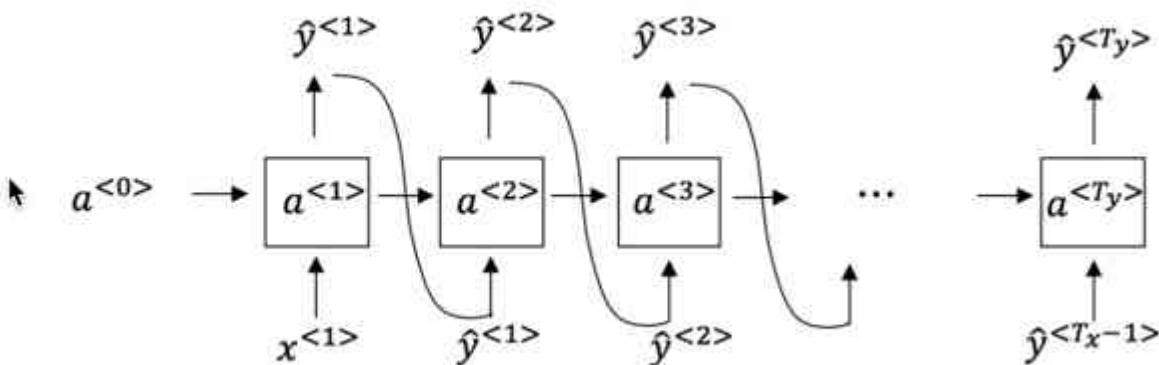
At the t^{th} time step, what is the RNN doing? Choose the best answer.

- Estimating $P(y^{<1>} , y^{<2>} , \dots , y^{<t-1>})$
- Estimating $P(y^{<t>})$
- Estimating $P(y^{<t>} | y^{<1>} , y^{<2>} , \dots , y^{<t-1>})$
- Estimating $P(y^{<t>} | y^{<1>} , y^{<2>} , \dots , y^{<t>})$

Machine Intelligence

Quiz

You have finished training a language model RNN and are using it to sample random sentences, as follows:



What are you doing at each time step i ?

- (i) Use the probabilities output by the RNN to pick the highest probability word for that time-step as $\hat{y}^{<i>}$. (ii) Then pass the ground-truth word from the training set to the next time-step.
- (i) Use the probabilities output by the RNN to randomly sample a chosen word for that time-step as $\hat{y}^{<i>}$. (ii) Then pass the ground-truth word from the training set to the next time-step.
- (i) Use the probabilities output by the RNN to pick the highest probability word for that time-step as $\hat{y}^{<i>}$. (ii) Then pass this selected word to the next time-step. 
- (i) Use the probabilities output by the RNN to randomly sample a chosen word for that time-step as $\hat{y}^{<i>}$. (ii) Then pass this selected word to the next time-step.



THANK YOU

Vighnesh Kamath

Department of Computer Science & Engineering

vighneshkamath43@gmail.com



PES
UNIVERSITY
ONLINE

MACHINE INTELLIGENCE

RNN- Recurrent Neural Networks

Vighnesh Kamath

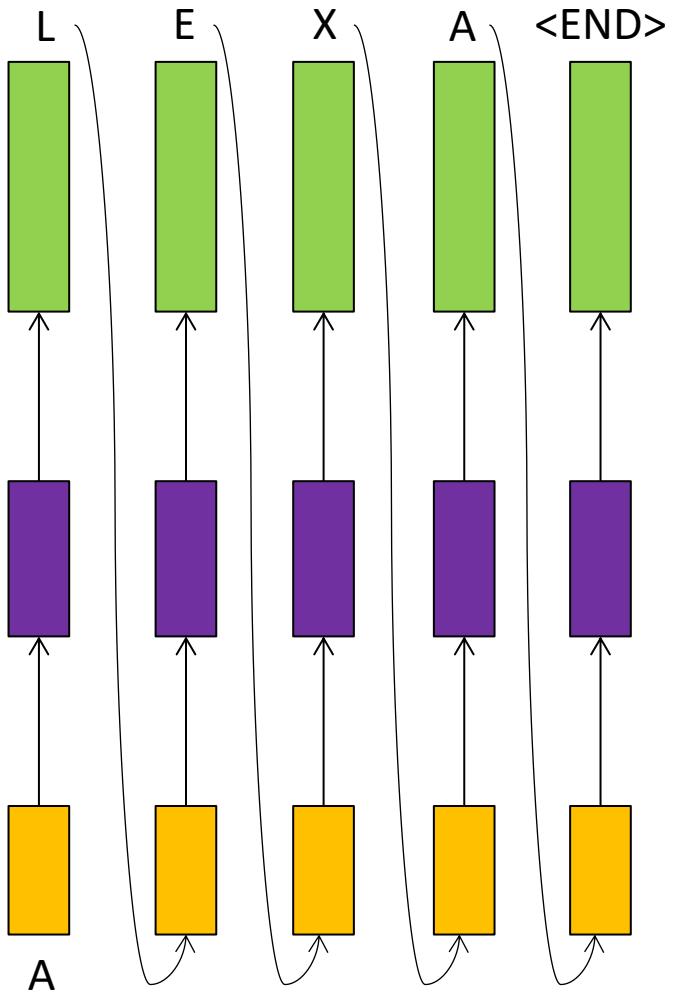
+

Dr. Arti Arya

Department of Computer Science and Engineering

Disclaimer

These slides are prepared from various resources from Internet and universities from India and Abroad and with the help from TAs . Broadly adapted from slides by Prof. Mitesh M Khapra from IITM

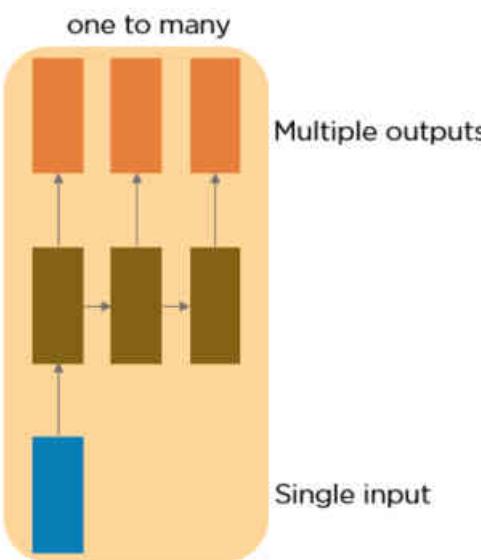
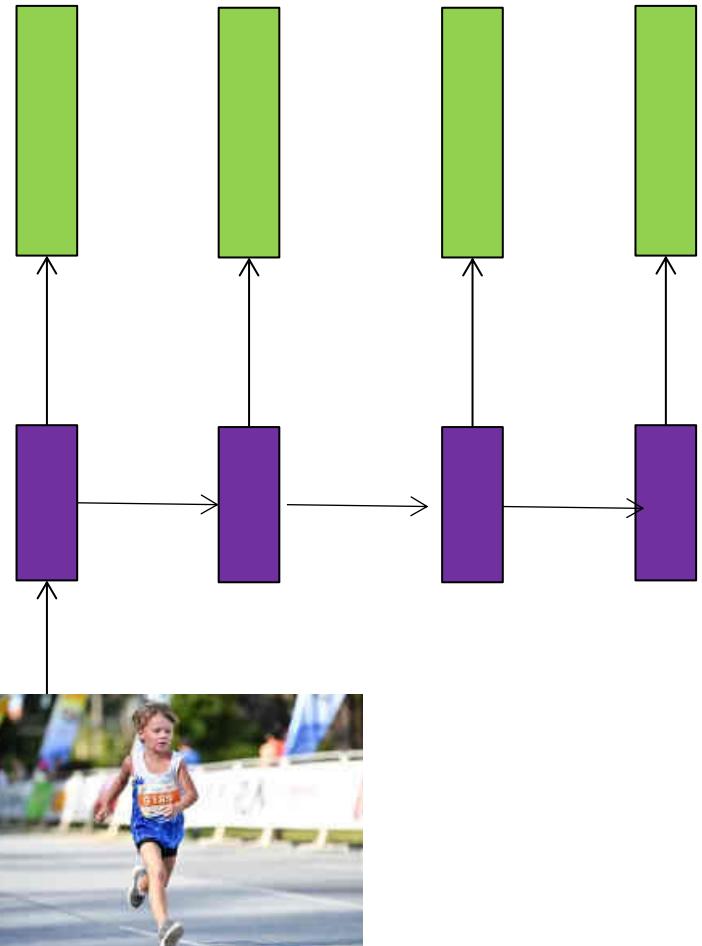


- In a one-to-one architecture, a single input is mapped to a single output, and the output from the time step t is fed as an input to the next time step $t+1$.
- Such architectures are known as One to One Architecture

This type of neural network
is known as the Vanilla
Neural Network

Consider the task of image captioning

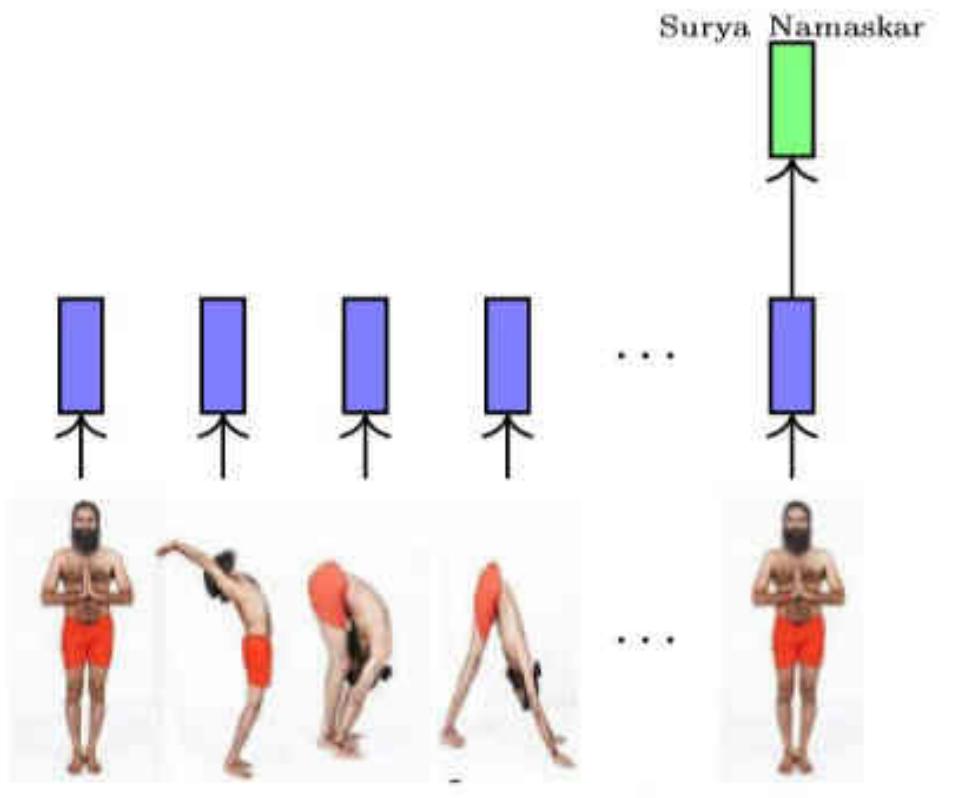
Boy is running <END>



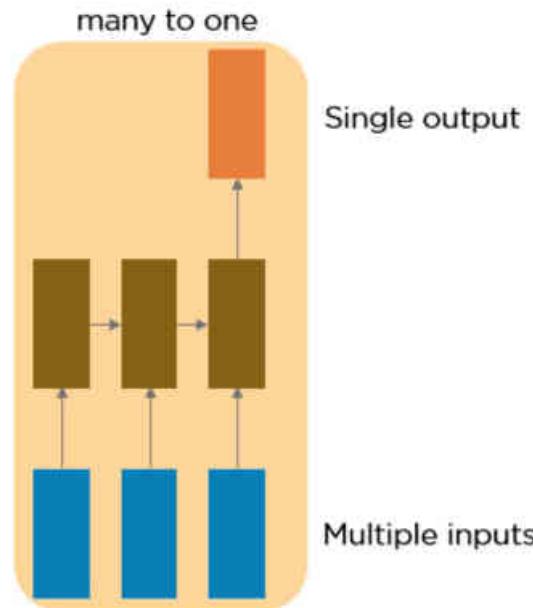
- A **single input** is mapped to **multiple hidden states** and **multiple output values**, which means RNN takes a **single input** and maps it to an **output sequence**.
- Although we have a single input value, we share the hidden states across time steps to predict the output.
- Such architectures are known as **One to Many Architecture**
- An example of this is the image caption or music generation.

Many-to-One

For example, classifying an action as **Suryanamaskar** or not Surya namaskar



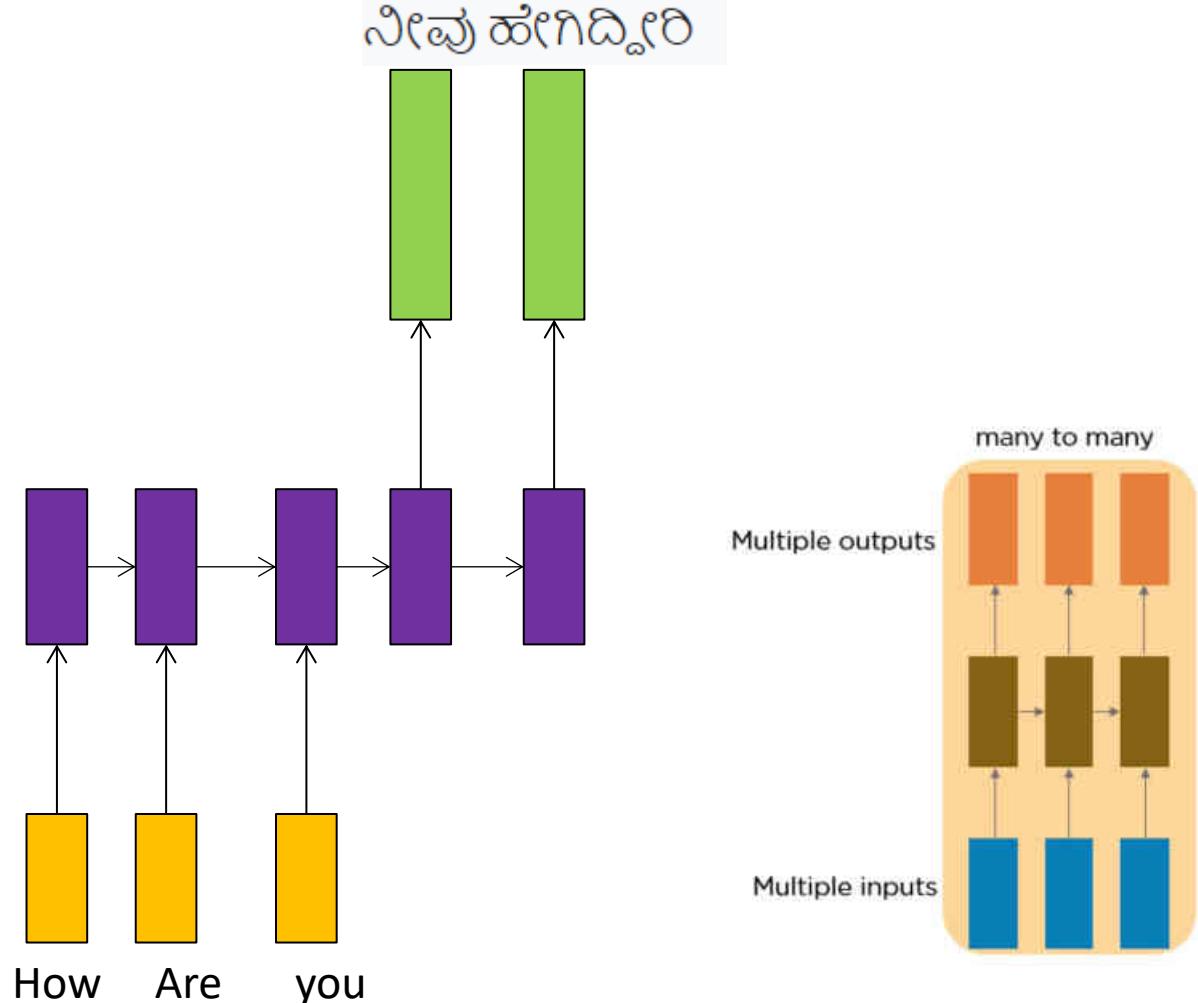
- The architecture accepts sequence of input and maps it to a single output value.
- Such architecture are known as **Many to One Architecture**



Sentiment analysis is a good example of this kind of network where a given sentence can be classified as expressing positive or negative sentiments.

Many-to-Many

Consider the task of language translation



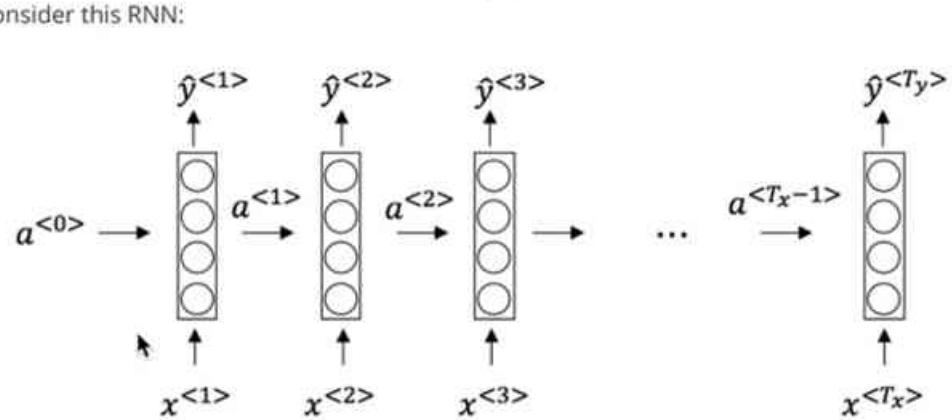
- Sentence in one language may not be of same size in another language
- Eg:
 - English: How are you
 - Kannada: ನೀವು ಹೇಗಿದ್ದೀರಿ
- For such problem, we design an architecture that accepts a sequence of **input of arbitrary length** and **maps to a sequence of output of arbitrary length**.

Such architecture is known as **Many to Many Architecture**

Machine translation,
Named Entity
recognition(NER)
are its examples.

Quick question

Consider this RNN:

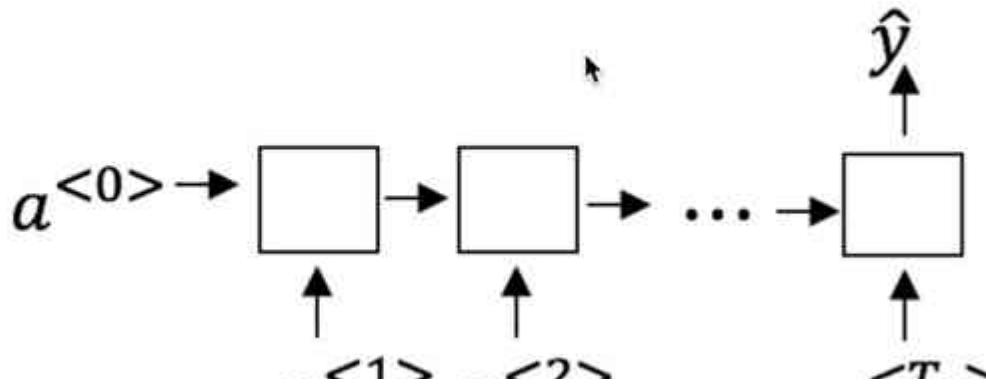


This architecture is appropriate when

1. $T_x = T_y$
2. $T_x < T_y$
3. $T_x > T_y$
4. $T_x = 1$

1
point

3. To which of these tasks would you apply a many-to-one RNN architecture? (Check all that apply).



- Speech recognition (input an audio clip and output a transcript)
- Sentiment classification (input a piece of text and output a 0/1 to denote positive or negative sentiment)
- Image classification (input an image and output a label)
- Gender recognition from speech (input an audio clip and output a label indicating the speaker's gender)

Two issues with RNN: Vanishing Gradient and Exploding Gradient

So far we have

- Understood what are Sequence Learning Problem
- And how to use RNN to solve sequence learning problem
- Also learnt how BPTT can help in updating the weights and thus reduces the error of the network.
- **But then comes the problem of Vanishing and exploding gradients problem**

Two issues with RNN: Vanishing Gradient and Exploding Gradient

Our new problem Vanishing and exploding gradients

While computing the derivative of loss wrt W and U, we saw that we have to traverse all the way back to first hidden state, as each hidden state at a specific time t_i is dependent on its previous hidden state at a time t_{i-1} .

Let us look at gradient loss L_2 wrt W which is given as:

$$\frac{\partial L_2}{\partial W} = \frac{\partial L_2}{\partial y_2} \frac{\partial y_2}{\partial h_2} \frac{\partial h_2}{\partial W}$$

if you look closely at this term we can't calculate the derivative pf h_2 wrt W directly as we know h_2 is dependent on h_1 and w ,which is in turn dependent on h_0 and w . Thus we need to calculate derivative wrt h_0 as well.

So for any loss L_j , we have to go back till the initial hidden step h_0 as each hidden state is dependent on its previous hidden state.

Two issues with RNN: Vanishing Gradient and Exploding Gradient

Let's look at gradient loss L_2 wrt W which is given as:

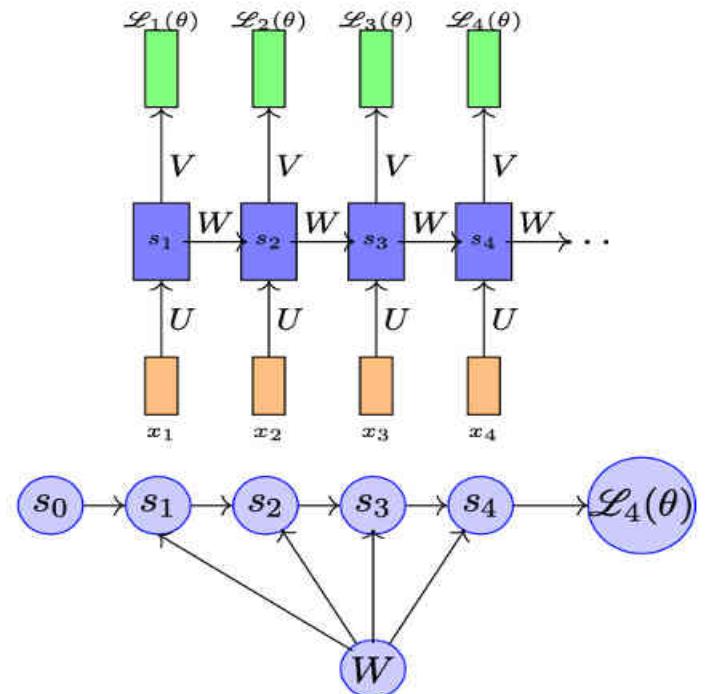
$$\frac{\partial h_2}{\partial W} = \frac{\partial h_2}{\partial h_1} \frac{\partial h_1}{\partial W} = \frac{\partial h_2}{\partial h_1} \frac{\partial h_1}{\partial h_0} \frac{\partial h_0}{\partial W}$$

$$\frac{\partial \mathcal{L}_2}{\partial W} = \frac{\partial \mathcal{L}_2}{\partial y_2} \frac{\partial y_2}{\partial h_2} \frac{\partial h_2}{\partial h_1} \frac{\partial h_1}{\partial h_0} \frac{\partial h_0}{\partial W}$$

$$\text{Or in general, } \frac{\partial h_t}{\partial h_k} = \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \dots \dots \frac{\partial h_{k+1}}{\partial h_k} = \prod_{j=k}^{t-1} \frac{\partial h_{j+1}}{\partial h_j}$$

$$\frac{\partial L_2}{\partial W} = \frac{\partial L_2}{\partial y_2} \frac{\partial y_2}{\partial h_2} \frac{\partial h_2}{\partial W}$$

Clearly, we can't calculate the derivative of h_2 wrt W directly as we know h_2 is dependent on h_1 and w , which is in turn dependent on h_0 and w . Thus we need to calculate derivative wrt h_0 as well.



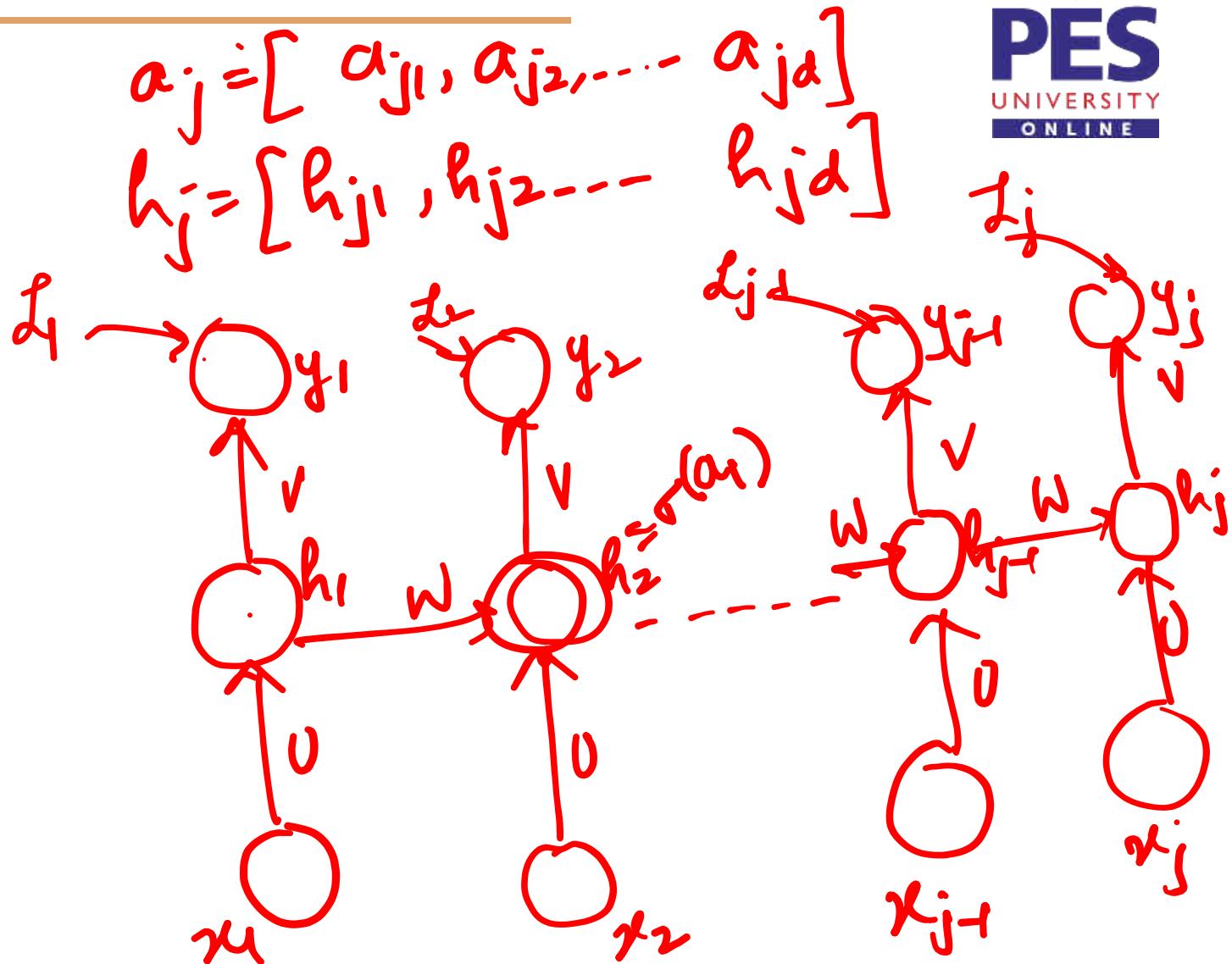
So we if need to calculate loss of any time step say L_j we need to go all the way back long to the initial hidden state h_0 as each hidden state is dependent on its previous hidden state.

Consider $\frac{\partial \ell_{ij}}{\partial h_{j-1}}$

$$a_j = W h_{j-1} + b$$

$$h_j = \sigma(a_j)$$

$$\begin{aligned}\frac{\partial \ell_{ij}}{\partial h_{j-1}} &= \frac{\partial \ell_{ij}}{\partial a_j} \times \frac{\partial a_j}{\partial h_{j-1}} \\ &= \sigma'(a_j) \times \frac{\partial a_j}{\partial h_{j-1}} \\ &= \sigma'(a_j) \times w \frac{\partial \ell_{ij}}{\partial h_{j-1}}\end{aligned}$$



$$= \sigma'(a_j) W$$

$$= \text{diag}(\sigma'(a_j)) W$$

$$\frac{\partial h_j}{\partial a_j} = \begin{bmatrix} \frac{\partial h_{j1}}{\partial a_{j1}} & \frac{\partial h_{j2}}{\partial a_{j1}} & \dots & \frac{\partial h_{jd}}{\partial a_{j1}} \\ \frac{\partial h_{j1}}{\partial a_{j2}} & \frac{\partial h_{j2}}{\partial a_{j2}} & \dots & \frac{\partial h_{jd}}{\partial a_{j2}} \\ \vdots & \vdots & \ddots & \frac{\partial h_{jd}}{\partial a_{jd}} \end{bmatrix}$$

If the magnitude of $\frac{\partial h_{ij}}{\partial a_{ji}}$ is small, then it $\frac{\partial h_{j-1}}{\partial a_{j-1}}$ will vanish & thus $\frac{\partial L_t}{\partial W}$ will vanish.

$$= \text{diag}(\sigma'(a_j))$$

We're interested in the magnitude of $\frac{\partial h_j}{\partial h_{j+1}}$. If it is small then, $\frac{\partial L_t}{\partial h_t}$ will vanish.

& thus $\frac{\partial L_t}{\partial w}$ will also vanish.

& if this magnitude is large then $\frac{\partial L_t}{\partial h_k}$ will explode & thus $\frac{\partial L_t}{\partial w}$ will explode.

Thus, $\|\frac{\partial h_j}{\partial a_j}\| = \|\text{diag}(\sigma'(a_j)) W\| \leq \|\text{diag}(\sigma'(a_j))\| \|W\|$

~~activation fn~~ $\because \sigma(a_j)$ is a bdd. function (Sigmoid or tanh)

so $\sigma'(a_j)$ is bdd

$\sigma'(a_j) \leq \gamma$ ($\gamma = \frac{1}{4}$, if σ is logistic)
& $\gamma = 1$, if σ is tanh func.

$$\left\| \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq \gamma \|w\| \leq \gamma \lambda \quad (\text{Considering } \|w\| \leq \lambda)$$

$$\alpha \left\| \frac{\partial h_t}{\partial h_k} \right\| = \left\| \prod_{j=k+1}^t \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq \prod_{j=k+1}^t \gamma \lambda \leq (\gamma \lambda)^{t-k}$$

If $\gamma \lambda < 1$, then gradient will vanish.

If $\gamma \lambda > 1$, then gradient will explode.

i.e when we initialize wts. of the n/w to very large numbers, the gradients will become large at every step.

While backpropagation, we multiply a large no. of partial deriv. together at every time step & it leads to \rightarrow exploding gradient.

One way to avoid this problem is to use truncated backpropagation (or gradient clipping) where we restrict the product to $\underbrace{\cdot \cdot \cdot}_{\sim} (< t-k)$ terms.

Machine Intelligence

Vanishing Gradient

While back propagating towards the initial hidden state, we lose information, and the RNN will not back propagate perfectly.

Remember $h_t = \tanh(Ux_t + Wh_{t-1})$? Every time we move backward, we compute the derivative of h_t .

THIS IS CALLED VANISHING GRADIENT PROBLEM

Derivative of \tanh is bounded to 1

Any two value b/n 0 and 1 when multiplied with each other gives us small number

usually initialize the weights of the network to a small number.

Thus, when we multiply the derivatives and weights while back propagating, we are essentially multiplying smaller numbers

So, when we multiply smaller numbers at every step while moving backward, our gradient becomes infinitesimally small and leads to a number that the computer can't handle;

Machine Intelligence

Vanishing Gradient

GRADIENT wrt hidden layer weights W

$$\frac{\partial L}{\partial W} = \sum_{j=0}^{T-1} \sum_{b=0}^j (\hat{y}_j - y_j) \prod_{m=k+1}^j W^T \text{diag}(1 - \tanh^2(W h_{m-1} + U x_m)) \otimes h_{k-1}$$

As you can observe, we are multiplying the ^{weights} and ^{activation function derivative} at every time step. Repeated multiplication of these two leads to a small number and causes the vanishing gradients problem.

The vanishing gradients problem occurs not only in RNN but also in other deep networks where we use **sigmoid or tanh** as the activation function.

So, to overcome this, we can use **ReLU** as an activation function instead of tanh.

Machine Intelligence

Exploding Gradient

Now can I suggest we keep the weights large initially ,so that this problem doesn't occur!!!!

Can we ??? NO!

when we initialize the weights of the network to a very large number, the gradients will become very large at every step.

While back propagating, we multiply a large number together at every time step, and it leads to infinity. This is called the **exploding gradient problem.**

OK! a new problem what is the solution to this!!!

gradient clipping !!
We can use gradient clipping to bypass the exploding gradient problem.

Machine Intelligence

Gradient clipping



In this method, we normalize the gradients according to a vector norm (say, L2) and clip the gradient value to a certain range.

For instance, if we set the threshold as 0.7, then we keep the gradients in the -0.7 to +0.7 range.

If the gradient value exceeds -0.7, then we change it to -0.7, and similarly, if it exceeds 0.7, then we change it to +0.7.

A bit of math again!!

Let's assume \hat{g} is the gradient of loss L with respect to W:

$$\hat{g} = \frac{\partial L}{\partial W}$$

First, we normalize the gradients using the L2 norm, that is, $||\hat{g}||$. If the normalized gradient exceeds the defined threshold, we update the gradient, as follows:

$$\hat{g} = \frac{\text{threshold}}{||\hat{g}||} \cdot \hat{g}$$

- Still we did not deal with vanishing gradient problem !!

We have a variant of the RNN called the **long short-term memory (LSTM)** network, which can solve the vanishing gradient problem effectively.



THANK YOU

Vighnesh Kamath

Department of Computer Science & Engineering

vighneshkamath43@gmail.com



PES
UNIVERSITY
ONLINE

MACHINE INTELLIGENCE

RNN- Recurrent Neural Networks

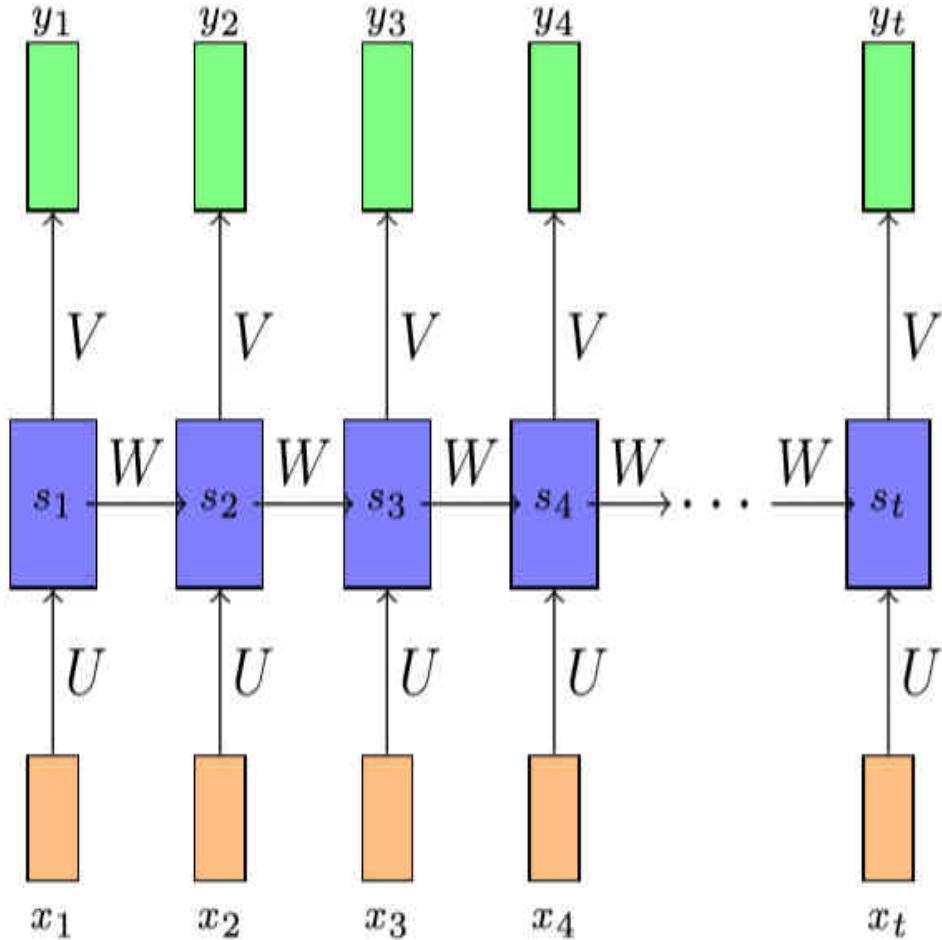
Dr. Arti Arya

Department of Computer Science and Engineering

Disclaimer

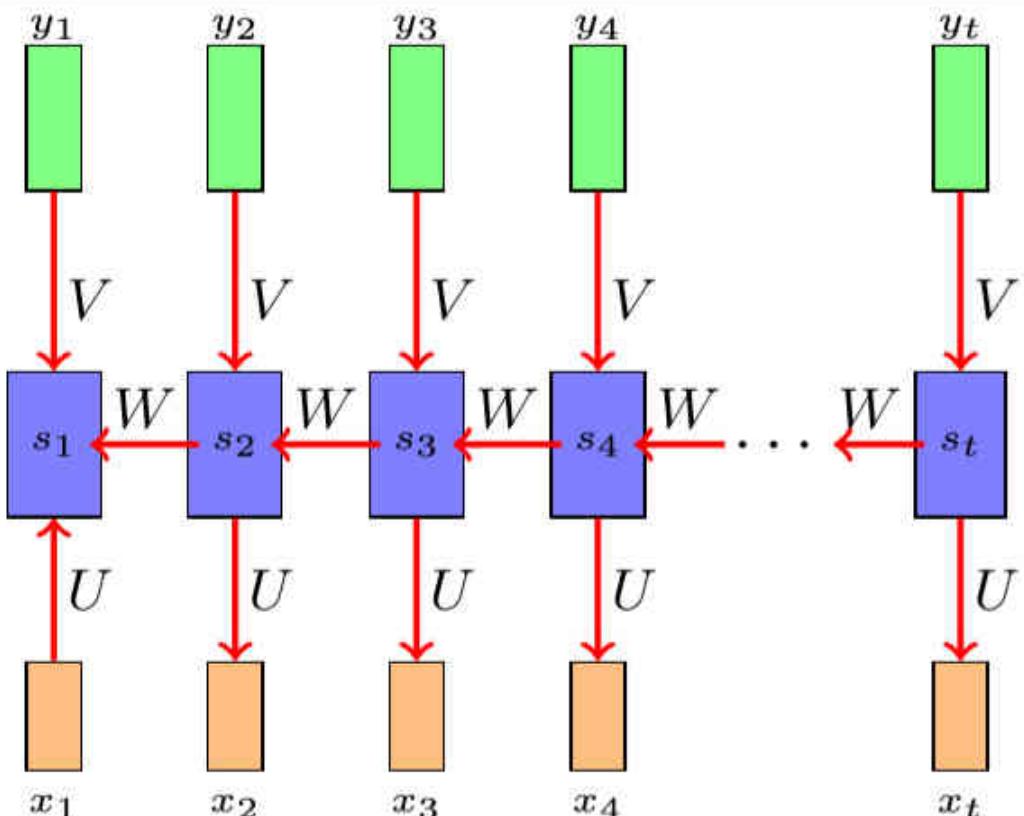
These slides are prepared from various resources from Internet and universities from India and Abroad and with the help from TAs . Broadly adapted from slides by Prof. Mitesh M Khapra from IITM

Selective Read, Selective Write and Selective Forget



- The state (s_i) of an RNN records information from all previous time steps
- At each new time step the **old information gets morphed by the current input**
- At this time step it not only has its own input but accumulated history from all the time steps
- One could imagine that after t steps the information stored at time step $t-k$ (for some $k < t$) gets completely morphed
- So much that it would be impossible to extract the original information stored at time step $t-k$

Selective Read ,Selective Write and Selective Forget



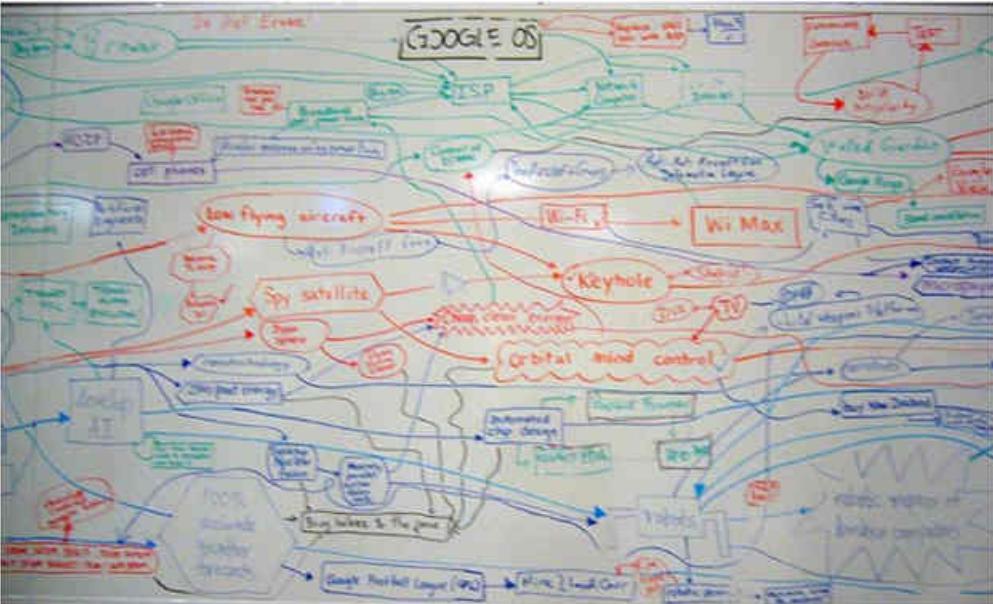
- A similar problem occurs when the information flows backwards (back propagation)
- It is very hard to assign the responsibility of the error caused at time step t to the events that occurred at time step $t-k$
- This responsibility is of course in the form of gradients and we studied the problem in backward flow of gradients
- We saw a formal argument for this while discussing vanishing gradients

Machine Intelligence

Selective Read ,Selective Write and Selective Forget



- Let us see an analogy for this
 - The analogy is a white board
 - We can think of the state as a fixed size memory
 - Compare this to a fixed size white board that you use to record information
 - At each time step (periodic intervals) we keep writing something to the board
 - Effectively at each time step we morph the information recorded till that time point
 - After many time steps it would be impossible to see how the information at time step $t-k$ contributed to the state at time step t
 - This happens whenever we do long derivations on white board that's why I use ppt...



- Continuing our whiteboard analogy, suppose we are interested in deriving an expression on the whiteboard
- We follow the following strategy at each time step
- **Selectively write** on the board
- **Selectively read** the already written content
- **Selectively forget** (erase) some content
- Let us look at each of these in detail

Selective Read ,Selective Write and Selective Forget

$$a = 1 \quad b = 3 \quad c = 5 \quad d = 11$$

Compute $ac(bd + a) + ad$

- say this the problem we want to compute

Say “board” can have only 3 statements at a time.

- ① ac
- ② bd
- ③ $bd + a$
- ④ $ac(bd + a)$
- ⑤ ad
- ⑥ $ac(bd + a) + ad$

Selective Read ,Selective Write and Selective Forget

$$a = 1 \quad b = 3 \quad c = 5 \quad d = 11$$

Compute $ac(bd + a) + ad$

Say “board” can have only 3 statements at a time.

① ac

② bd

③ $bd + a$

④ $ac(bd + a)$

⑤ ad

⑥ $ac(bd + a) + ad$

Let's say we write:

$$\begin{aligned} ac &= 5 \\ bd &= 33 \end{aligned}$$

We could also have written

$$a=1$$

$$c=5$$

$$a \times c = 5$$

$$b=3$$

$$d=11$$

$$b \times d = 33$$

- But wrote only two steps. **Why?**
- Bcoz the whiteboard has finite size and we are writing only the steps that are needed.
- And not write unnecessary things because one can run out of memory/space

So this is selective writing

Selective Read , Selective Write and Selective Forget

$$a = 1 \quad b = 3 \quad c = 5 \quad d = 11$$

Compute $ac(bd + a) + ad$

Say “board” can have only 3 statements at a time.

- | | | |
|---|-------------------|-------------------------------|
| ① | ac | |
| ② | bd | $ac=5$ |
| ③ | $bd + a$ | $bd=33$ |
| ④ | $ac(bd + a)$ | $bd+a=34$ |
| ⑤ | ad | $ac(bd+a)$ |
| ⑥ | $ac(bd + a) + ad$ | |

- Something is already stored on the whiteboard this is the state of the white board but we may **not need to read everything from the white board.**
- While writing one step we typically read some of the previous steps we have already written and then decide what to write next
- For example at Step 3, information from Step 2 is important
- In other words **we select what to read.**
- Now what has happened is we have exhausted the limit of our white board
- And we need to compute $ac(bd+a)$
- So we selectively erase
- Obviously **erase bd**
- So **we selectively forget some things**

Selective Read ,Selective Write and Selective Forget

- There are various other scenarios where we can motivate the need for selective write, read and forget
- Eg, the SVM derivation we did in Unit 2 we selectively wrote it
- Lets now ask you do the same derivation again without any resource
- Most of you will fail to do selective read because you have already done selective forget

Selective Read ,Selective Write and Selective Forget

Since the RNN also has a finite state size,
we need to figure out a way to allow it to
selectively read, write and forget

Such that during forward pass even if the information
gets morphed it gets morphed in principle manner .

Let's see how can we do it???



PROBLEM

The drawback of a recurrent neural network (RNN) is that **it will not retain information for a long time in memory. (Also it can't handle long input sequences)**

We know that an RNN stores sequences of information in its hidden state but when the input sequence is too long, **it cannot retain all the information in its memory due to the vanishing gradient problem.**

SOLUTION

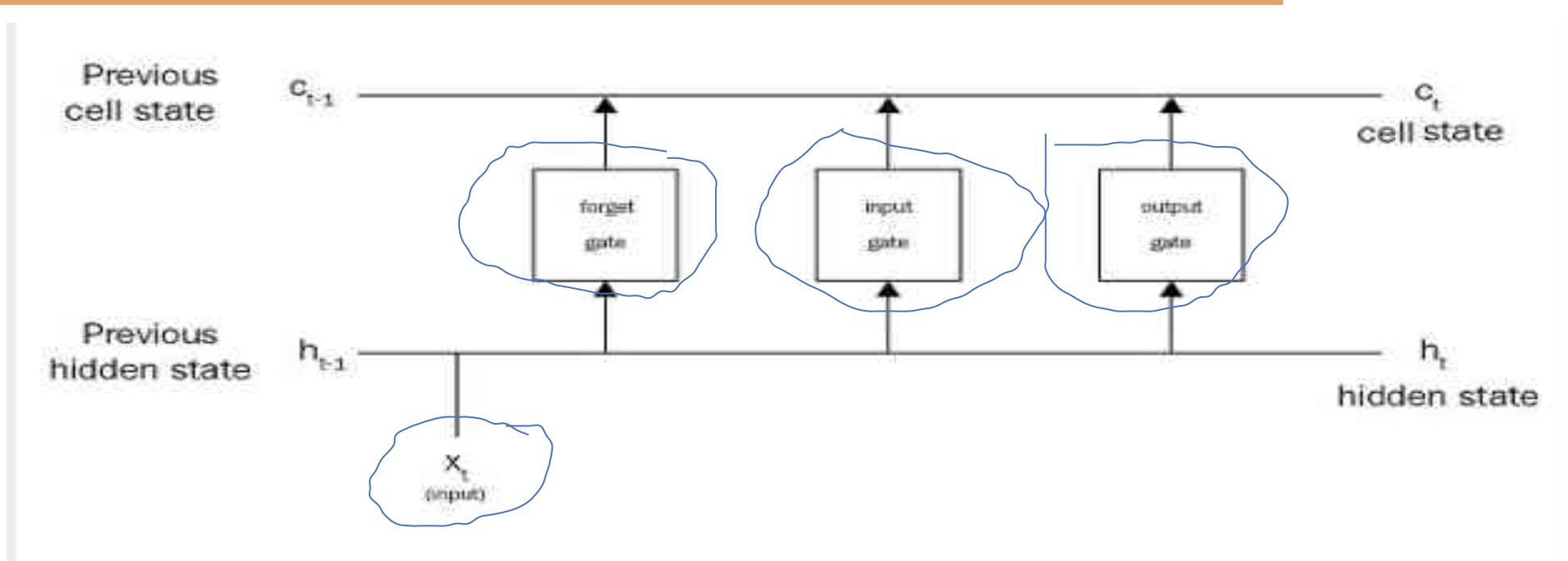
To combat this, we introduce a variant of RNN called a **long short-term memory (LSTM)** cell, which resolves the vanishing gradient problem by using a **special structure called a gate.**

Gates keep the information in memory as long as it is required. They learn what information to keep and what information to discard from the memory.

What makes LSTM cells so special? How do LSTM cells achieve long-term dependency?

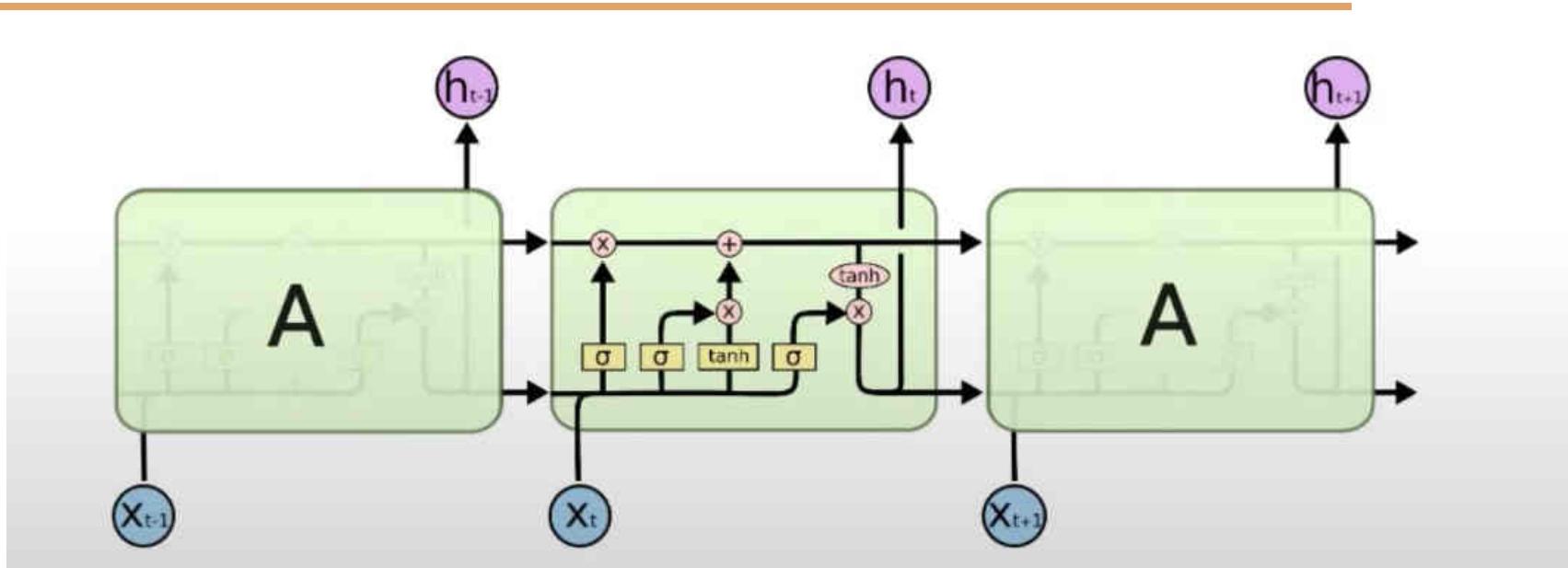
How does it know what information to keep and what information to discard from the memory?

This is all achieved by special structures called gates. A typical LSTM cell consists of three special gates called the **input gate, output gate, and forget gate**:



These three gates are responsible for deciding what information to add, output, and forget from the memory.

With these gates, an LSTM cell effectively keeps information in the memory only as long as required.



These three gates are responsible for deciding what information to add, output, and forget from the memory.

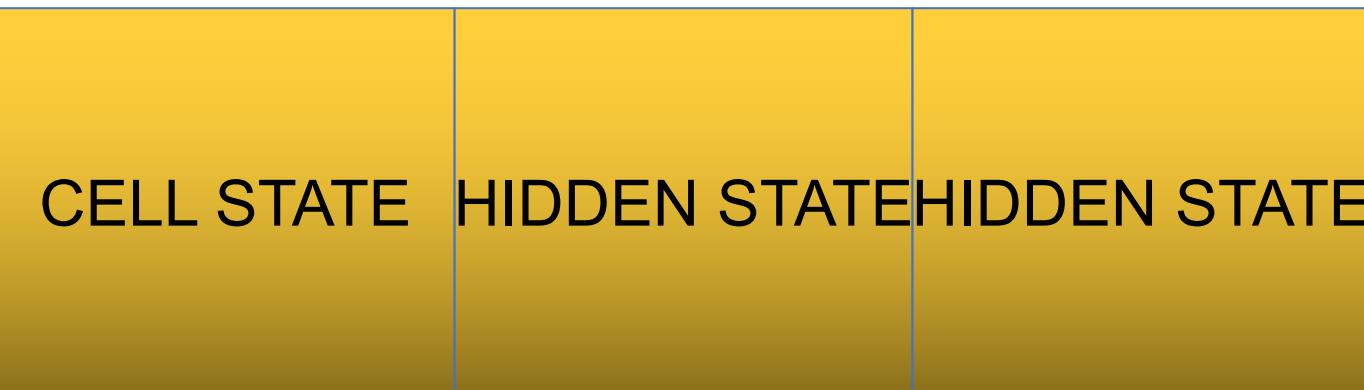
With these gates, an LSTM cell effectively keeps information in the memory only as long as required.

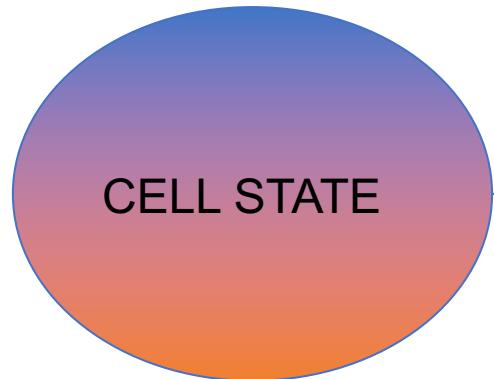
Machine Intelligence

LSTM

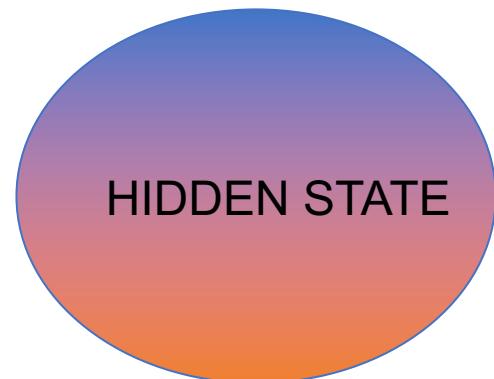
In an RNN cell, we used the **hidden state**,
for two purposes: one for **storing the information** and the other
for **making predictions**.

Unlike RNN, in the LSTM cell we break
the **hidden states** into two states, called the **cell state** and the
hidden state:





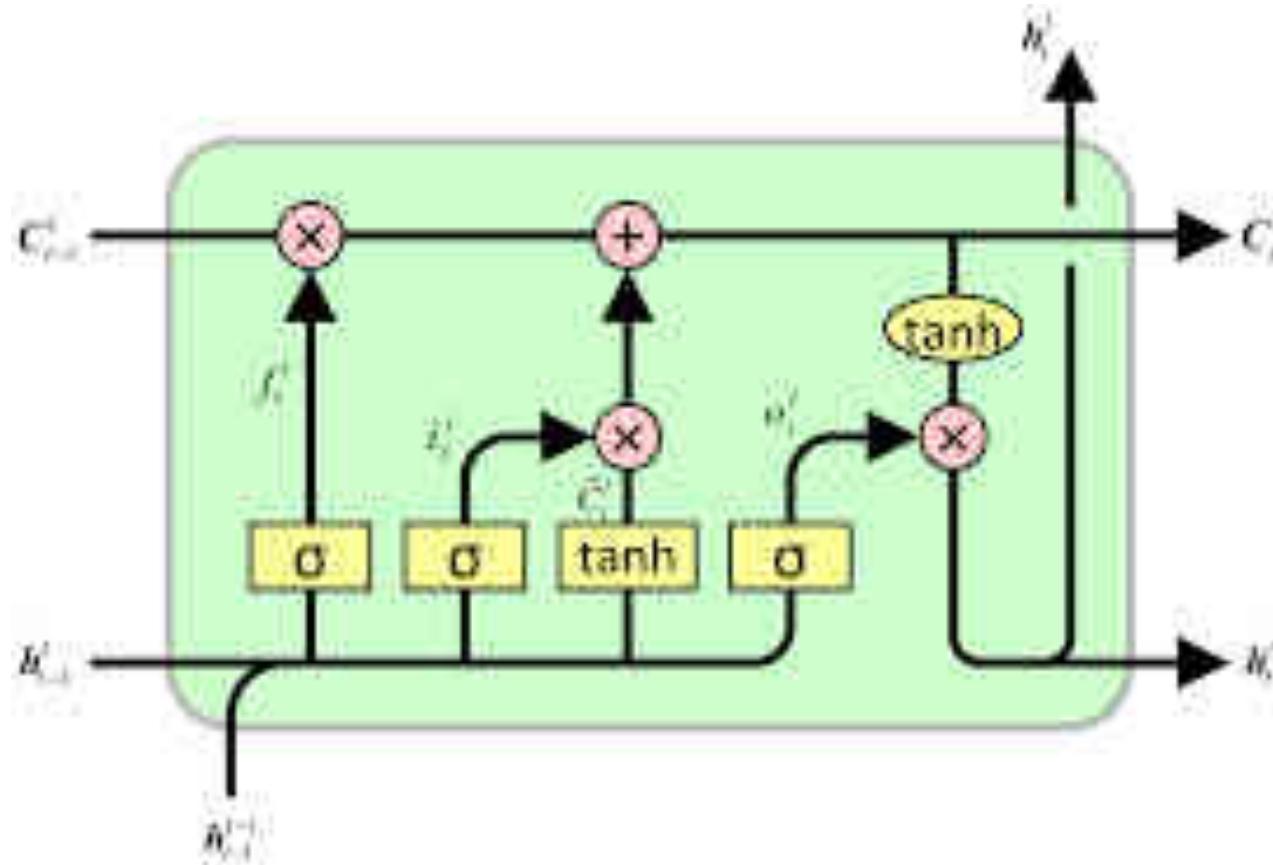
The cell state is also called internal memory and is where all the information will be stored



The hidden state is used for computing the output, that is, for making predictions

Machine Intelligence

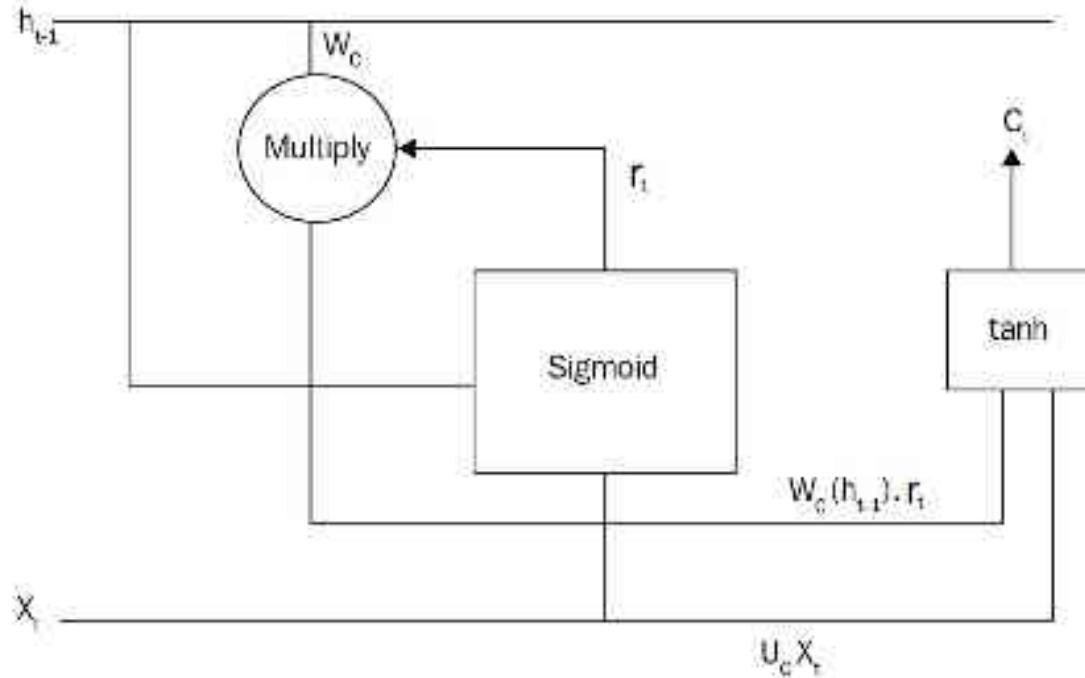
LSTM Architecture



- The following the the architecture of a LSTM cell
- LSTM implements this concept of Selective Read, Selective Write and Selective Forget using gates and solves problem vanishing gradients.
- However multiple gates lead to increase in number learnable parameters, and this is solved by one more modified architecture called GRU

Machine Intelligence

GRU Architecture

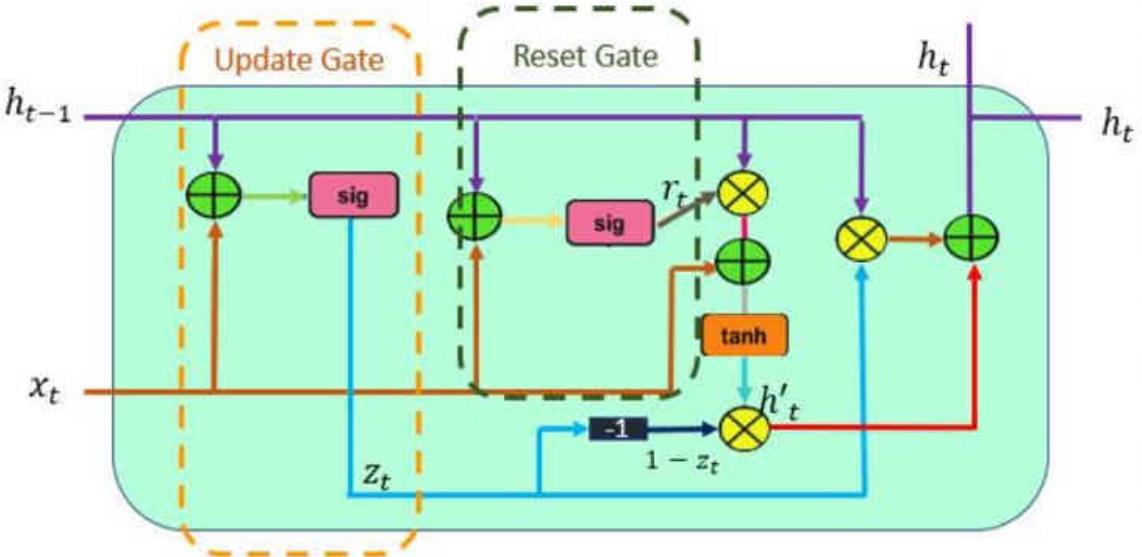


- The following is the architecture of a GRU cell
- GRU uses just two gates called **update gate and reset gate**
- There is however no concept behind decision to choose GRU over LSTM
- Its the domain of the project that favours which one to choose

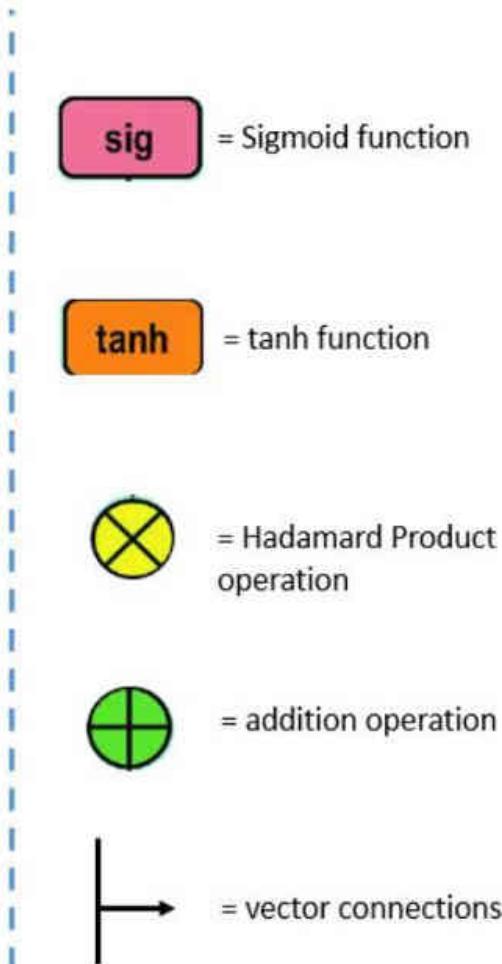
Machine Intelligence

GRU Architecture

The following is the architecture of a GRU cell



Gated Recurrent Network (GRU)

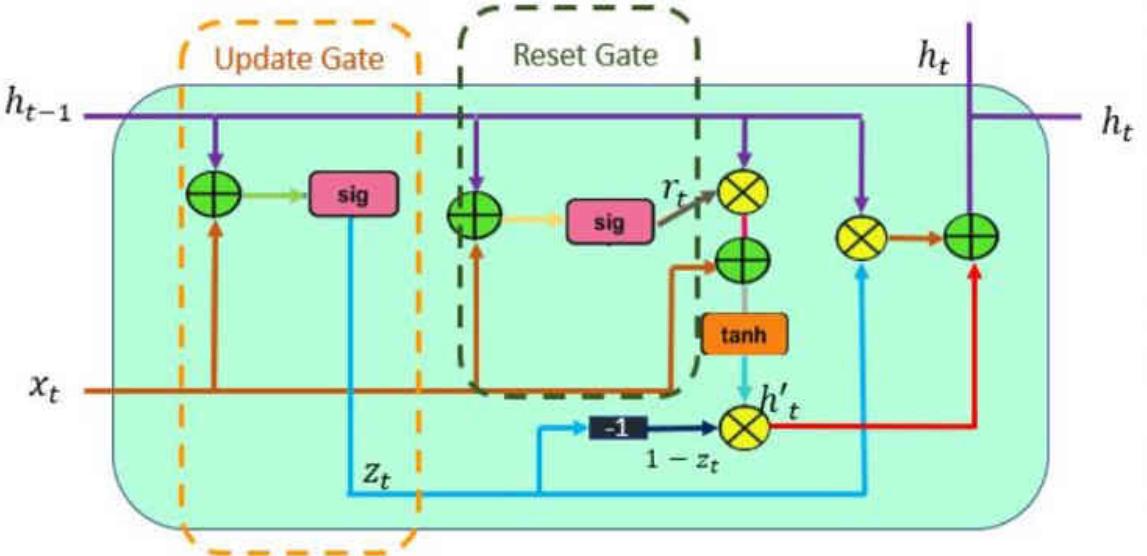


- GRU uses just two gates called **update gate and reset gate**
- There is however no concept behind decision to choose GRU over LSTM
- Its the domain of the project that favours which one to choose

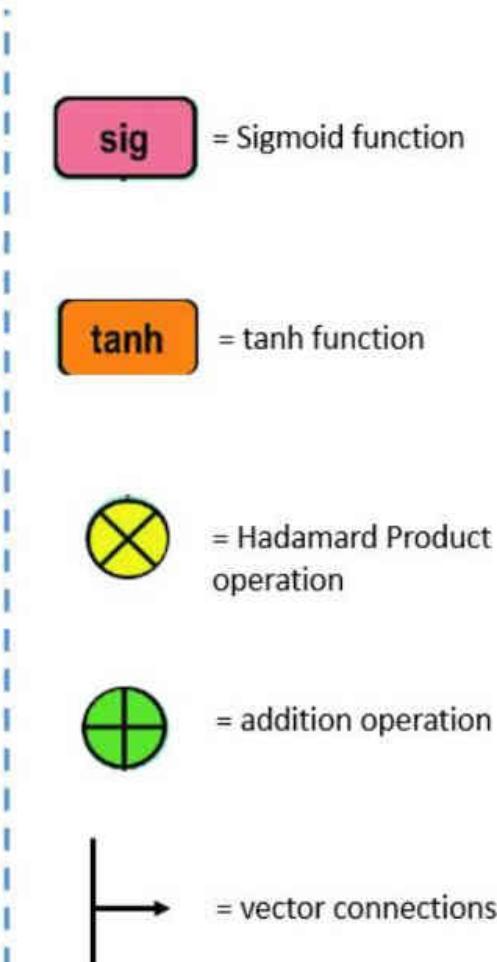
Unlike LSTM, GRU does not have **an output gate** and **combines the input and the forget gate into a single update gate**.

Machine Intelligence

GRU Architecture



Gated Recurrent Network (GRU)



- The update gate (z_t) is responsible for *determining the amount of previous information (prior time steps) that needs to be passed along the next state.*
- Here, x_t is the input vector served in the network unit.
- The reset gate (r_t) is used to decide *how much of the past information is needed to neglect.*

Unlike LSTM, GRU does not have **an output gate** and **combines the input and the forget gate into a single update gate.**

Machine Intelligence

Quiz

You are training an RNN, and find that your weights and activations are all taking on the value of NaN ("Not a Number"). Which of these is the most likely cause of this problem?

- Vanishing gradient problem.
- Exploding gradient problem.
- ReLU activation function $g(\cdot)$ used to compute $g(z)$, where z is too large.
- Sigmoid activation function $g(\cdot)$ used to compute $g(z)$, where z is too large.




THANK YOU

Vighnesh Kamath
+
Dr. Arti Arya

Department of Computer Science & Engineering



MACHINE INTELLIGENCE

Convolutional Neural Networks

Dr. Arti Arya

Department of Computer Science and
Engineering

Machine Intelligence

Declaration

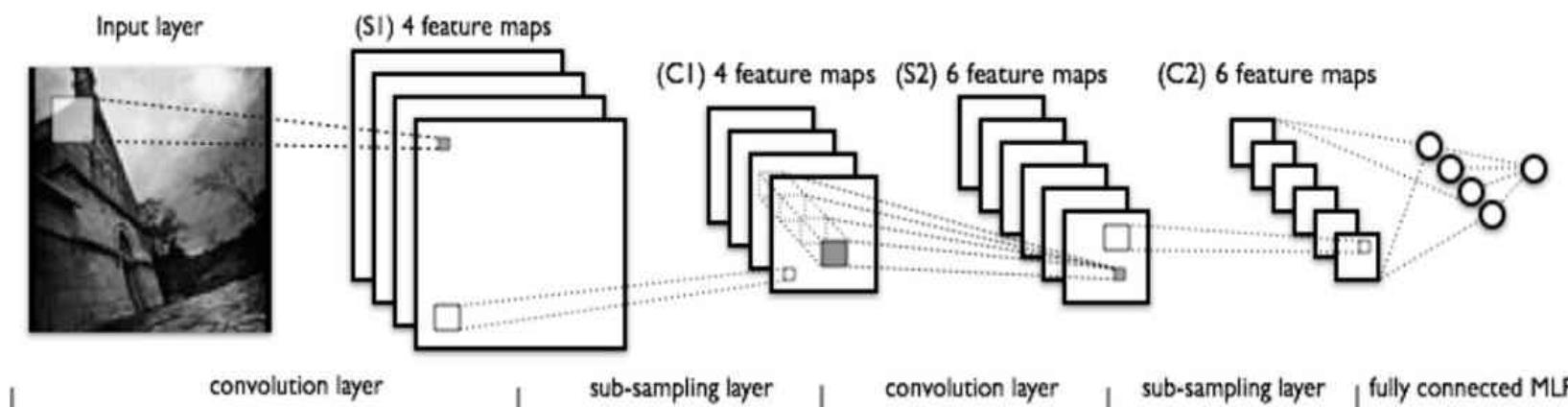


These slides are prepared from various resources from Internet and universities from India and Abroad. Broadly adapted from slides by Prof. Mitesh M Khapra from IITM.

What is a Convolution Neural Network?

ConvNets were initially developed in the neural network **image processing** community where they achieved break-through results in recognising an object from a pre-defined category.

A Convolutional Neural Network typically involves two operations, which can be thought of as feature extractors: **convolution** and **pooling**.



The Convolutional Neural Network architecture applied to image classification.

(Image adapted from <http://deeplearning.net/>)

The **output of this sequence of operations** is then typically sent as input to a **fully connected layer(s)** which is usually a multi-layer perceptron neural network (MLP).

What is a Convolution Neural Network?



For this image, suppose a classification task is to be performed.

Whether the image contains a bird or not?

This is a classic use case of CNN by which we perform image classification

Why don't we use neural networks for the same?

What is a Convolution Neural Network?

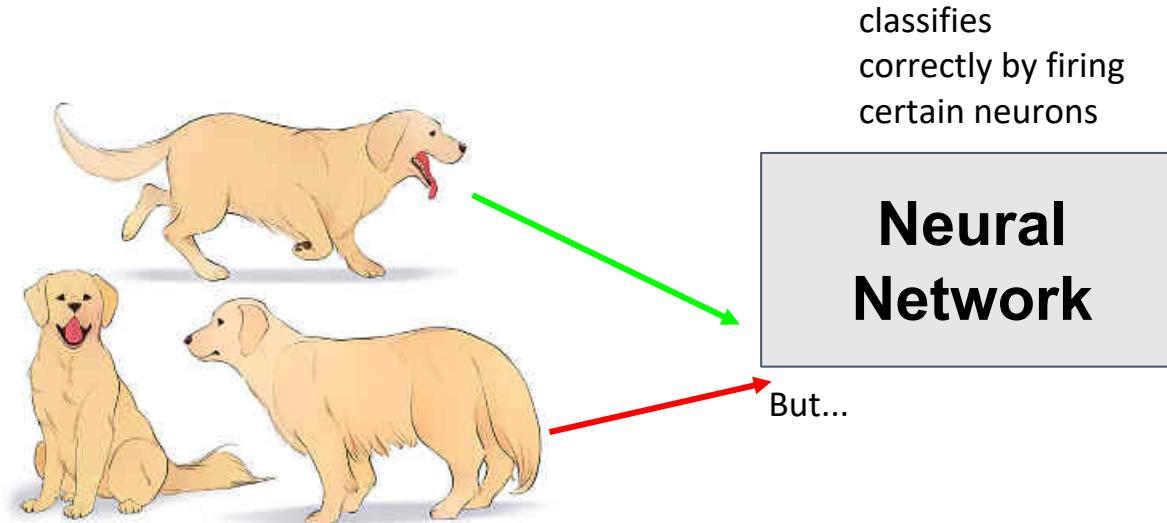
Why not Neural Networks for image classification task?

Reason 1: Images are Big

- Images used for computer vision are 224 x 224 or larger and they include **3 colour channels** so $224 \times 224 \times 3 = 150,528$ input features.
- As this propagates into further layers over a **million weights** are required in first layer alone. This would make it nearly impossible to train.

Reason 2: Positions can change

- If a network is trained to detect dogs, you'd want it to be able to detect a dog regardless of where it appears in the image.



But...

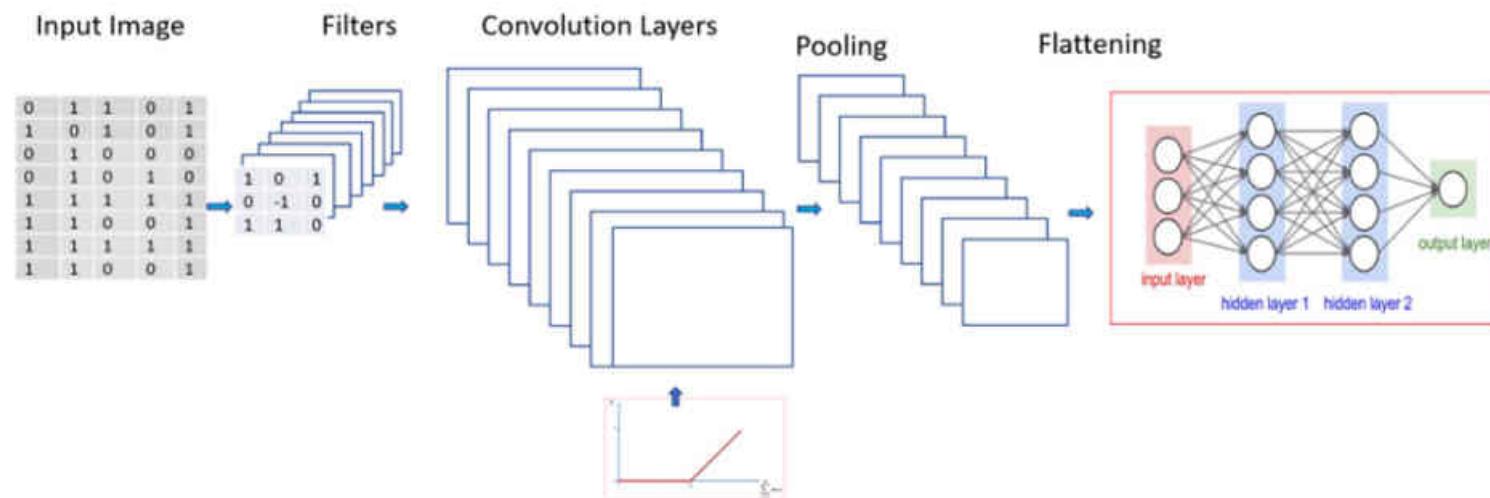
Sending input the same entity in different angles or positions, it **would not activate the same neurons**, so the network would react completely differently!

In such cases CNN can help us solve problems

What is a Convolution Neural Network?

What are Convolutional Neural Networks?

They're basically just neural networks that use **Convolutional layers**(Conv layers), which are based on the mathematical **operation of convolution**. Convolution layers consist of a set of filters.



What is a Convolution Neural Network?

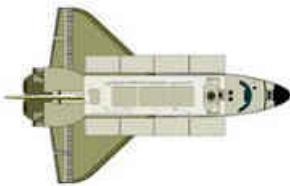
- Here element wise multiplication is happening
- If the **Blue matrix is an instance** we seem to get a smaller version of some abstracted version in the yellow box
- Is it **feature Reduction? ?**

Nope

- We can see that some neighbourhood information captured.

$$\begin{array}{|c|c|c|c|c|} \hline 7 & 2 & 3 & 3 & 8 \\ \hline 4 & 5 & 3 & 8 & 4 \\ \hline 3 & 3 & 2 & 8 & 4 \\ \hline 2 & 8 & 7 & 2 & 7 \\ \hline 5 & 4 & 4 & 5 & 4 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline \end{array} = \begin{array}{|c|c|c|c|c|} \hline & & & & \\ \hline & & & & \\ \hline & & 6 & & \\ \hline & & & & \\ \hline & & & & \\ \hline \end{array}$$

$$7 \times 1 + 4 \times 1 + 3 \times 1 + 2 \times 0 + 5 \times 0 + 3 \times 0 + 3 \times 1 + 3 \times 1 + 2 \times 1 = 6$$

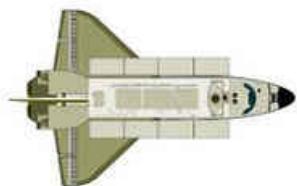
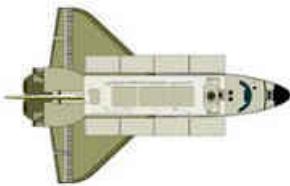


x_0

- Suppose we are tracking the position of an aeroplane using a laser sensor at discrete time intervals

Machine Intelligence

CNN



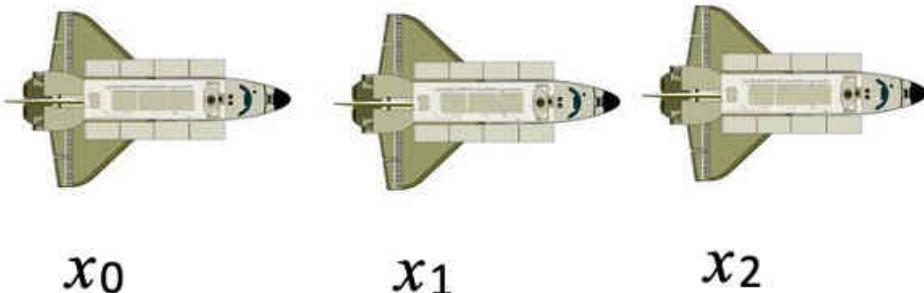
x_0

x_1

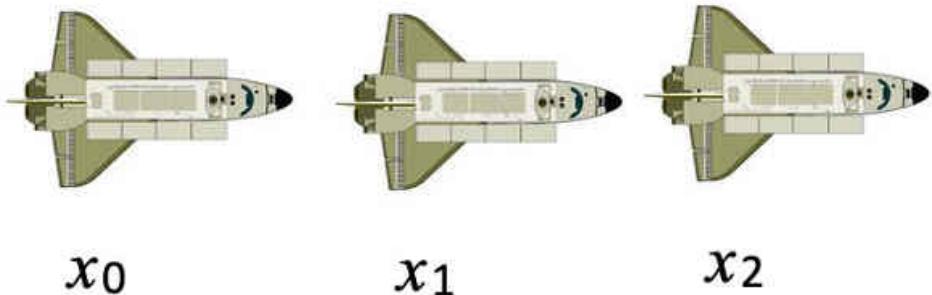
- Suppose we are tracking the position of an aeroplane using a laser sensor at discrete time intervals

Machine Intelligence

CNN

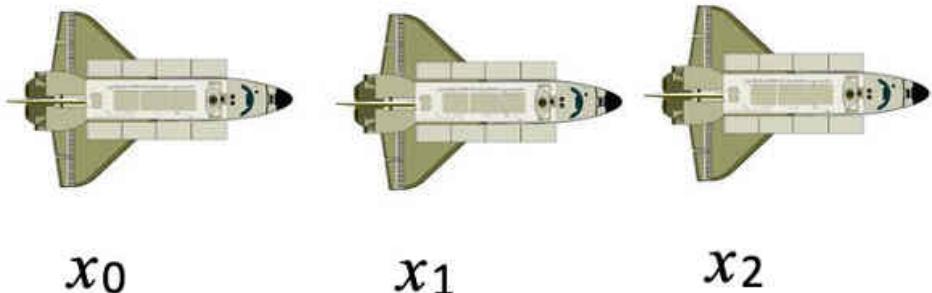


Suppose we are tracking the position of an aeroplane using a laser sensor at discrete time intervals



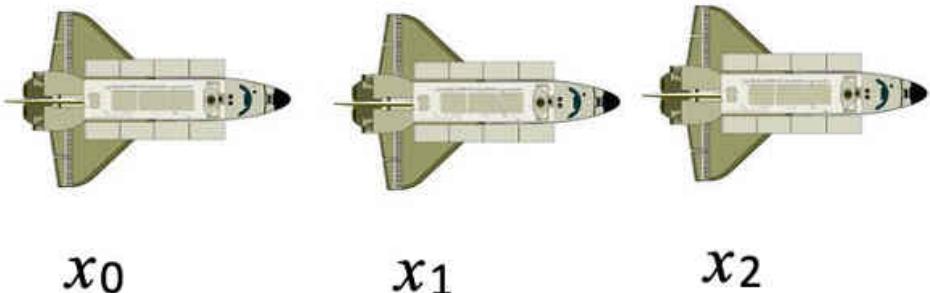
Suppose we are tracking the position of an aeroplane using a laser sensor at discrete time intervals

- Now suppose our sensor is noisy



Suppose we are tracking the position of an aeroplane using a laser sensor at discrete time intervals

- Now suppose our sensor is noisy
- To obtain a less noisy estimate we would like to average several measurements
- More recent measurements are more important so we would like to take a weighted average



Suppose we are tracking the position of an aeroplane using a laser sensor at discrete time intervals

$$s_t = \sum_{a=0}^{\infty} x_{t-a} w_a = (x * w)_t$$

Diagram illustrating the convolution operation:

- Input:** The sequence of input frames x_{t-a} for $a = 0, 1, 2, \dots, \infty$.
- Filter:** The filter w_a applied to each input frame.
- Convolution operator:** The process of applying the filter to the input sequence to produce the output s_t .

- Now suppose our sensor is noisy
- To obtain a less noisy estimate we would like to average several measurements
- More recent measurements are more important so we would like to take a weighted average

$$s_t = \sum_{a=0}^6 x_{t-a} w_a$$

- In practice, we would only sum over a small window

$$s_t = \sum_{a=0}^6 x_{t-a} w_a$$

- In practice, we would only sum over a small window
- The weight array (**w**) is known as the filter

Machine Intelligence

CNN

$$s_t = \sum_{a=0}^6 x_{t-a} w_a$$

| | w_{-6} | w_{-5} | w_{-4} | w_{-3} | w_{-2} | w_{-1} | w_0 |
|---|----------|----------|----------|----------|----------|----------|-------|
| w | 0.01 | 0.01 | 0.02 | 0.02 | 0.04 | 0.4 | 0.5 |
| x | 1.00 | 1.10 | 1.20 | 1.40 | 1.70 | 1.80 | 1.90 |
| s | 1.80 | | | | | | |

$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_2 w_{-4} + x_1 w_{-5} + x_0 w_{-6}$$

- In practice, we would only sum over a small window

- The weight array (**w**) is known as the filter

We just slide the filter over the input and

- compute the value of s_t based on a window around x_t

Machine Intelligence

CNN

$$s_t = \sum_{a=0}^6 x_{t-a} w_a$$

$w_{-6} w_{-5} w_{-4} w_{-3} w_{-2} w_{-1} w_0$

| | | | | | | | |
|---|------|------|------|------|------|-----|-----|
| w | 0.01 | 0.01 | 0.02 | 0.02 | 0.04 | 0.4 | 0.5 |
|---|------|------|------|------|------|-----|-----|

| | | | | | | | | | | | | |
|---|------|------|------|------|------|------|------|------|------|------|------|------|
| x | 1.00 | 1.10 | 1.20 | 1.40 | 1.70 | 1.80 | 1.90 | 2.10 | 2.20 | 2.40 | 2.50 | 2.70 |
|---|------|------|------|------|------|------|------|------|------|------|------|------|

| | | | | | | |
|---|------|------|--|--|--|--|
| s | 1.80 | 1.96 | | | | |
|---|------|------|--|--|--|--|

$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_2 w_{-4} + x_1 w_{-5} + x_0 w_{-6}$$

- In practice, we would only sum over a small window
 - The weight array (**w**) is known as the filter
- We just slide the filter over the input and
- compute the value of s_t based on a window around x_t

Machine Intelligence

CNN

$$s_t = \sum_{a=0}^6 x_{t-a} w_a$$

$w_{-6} w_{-5} w_{-4} w_{-3} w_{-2} w_{-1} w_0$

| | | | | | | | |
|---|------|------|------|------|------|-----|-----|
| w | 0.01 | 0.01 | 0.02 | 0.02 | 0.04 | 0.4 | 0.5 |
|---|------|------|------|------|------|-----|-----|

| | | | | | | | | | | | | |
|---|------|------|------|------|------|------|------|------|------|------|------|------|
| x | 1.00 | 1.10 | 1.20 | 1.40 | 1.70 | 1.80 | 1.90 | 2.10 | 2.20 | 2.40 | 2.50 | 2.70 |
|---|------|------|------|------|------|------|------|------|------|------|------|------|

s

| | | | | | |
|------|------|------|--|--|--|
| 1.80 | 1.96 | 2.11 | | | |
|------|------|------|--|--|--|

$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_2 w_{-4} + x_1 w_{-5} + x_0 w_{-6}$$

- In practice, we would only sum over a small window

- The weight array (**w**) is known as the filter

- We just slide the filter over the input and
- compute the value of s_t based on a window around x_t

Machine Intelligence

CNN

$$s_t = \sum_{a=0}^6 x_{t-a} w_a$$

| | w_{-6} | w_{-5} | w_{-4} | w_{-3} | w_{-2} | w_{-1} | w_0 |
|---|----------|----------|----------|----------|----------|----------|-------|
| w | 0.01 | 0.01 | 0.02 | 0.02 | 0.04 | 0.4 | 0.5 |
| x | 1.00 | 1.10 | 1.20 | 1.40 | 1.70 | 1.80 | 1.90 |
| s | 1.80 | 1.96 | 2.11 | 2.16 | 2.28 | | |

$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_2 w_{-4} + x_1 w_{-5} + x_0 w_{-6}$$

- In practice, we would only sum over a small window

- The weight array (**w**) is known as the filter

We just slide the filter over the input and

- compute the value of s_t based on a window around x_t

Machine Intelligence

CNN

$$s_t = \sum_{a=0}^6 x_{t-a} w_a$$

w

| | w_{-6} | w_{-5} | w_{-4} | w_{-3} | w_{-2} | w_{-1} | w_0 |
|--|----------|----------|----------|----------|----------|----------|-------|
| | 0.01 | 0.01 | 0.02 | 0.02 | 0.04 | 0.4 | 0.5 |

x

| | | | | | | | | | | | |
|------|------|------|------|------|------|------|------|------|------|------|------|
| 1.00 | 1.10 | 1.20 | 1.40 | 1.70 | 1.80 | 1.90 | 2.10 | 2.20 | 2.40 | 2.50 | 2.70 |
|------|------|------|------|------|------|------|------|------|------|------|------|

s

| | | | | | |
|------|------|------|------|------|------|
| 1.80 | 1.96 | 2.11 | 2.16 | 2.28 | 2.42 |
|------|------|------|------|------|------|

$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_2 w_{-4} + x_1 w_{-5} + x_0 w_{-6}$$

- In practice, we would only sum over a small window

- The weight array (**w**) is known as the filter

- We just slide the filter over the input and
- compute the value of s_t based on a window around x_t

- Here the input (and the kernel) is one dimensional

- Can we use a convolutional operation on a 2D input also?

What is a Convolution?

It is the **cross-channel sum** of the **element-wise multiplication** of a convolutional filter (kernel/mask) computed over a sliding window on an input tensor given a certain stride and padding, **plus a bias term**. The result is called a **feature map**.

$$1*2 - 1*2 - 1*3 + 0*1 + 2 = -1$$

$$1*2 - 1*2 - 1*1 + 0*-1 + 2 = 2$$

$$1*3 - 1*1 - 1*4 + 0*3 + 2 = 0$$

$$1*1 - (-1)*1 - 1*3 + 0*2 + 2 = 1$$

| | | |
|---|---|----|
| 2 | 2 | 1 |
| 3 | 1 | -1 |
| 4 | 3 | 2 |

Input matrix (3x3)

no padding

1 channel

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

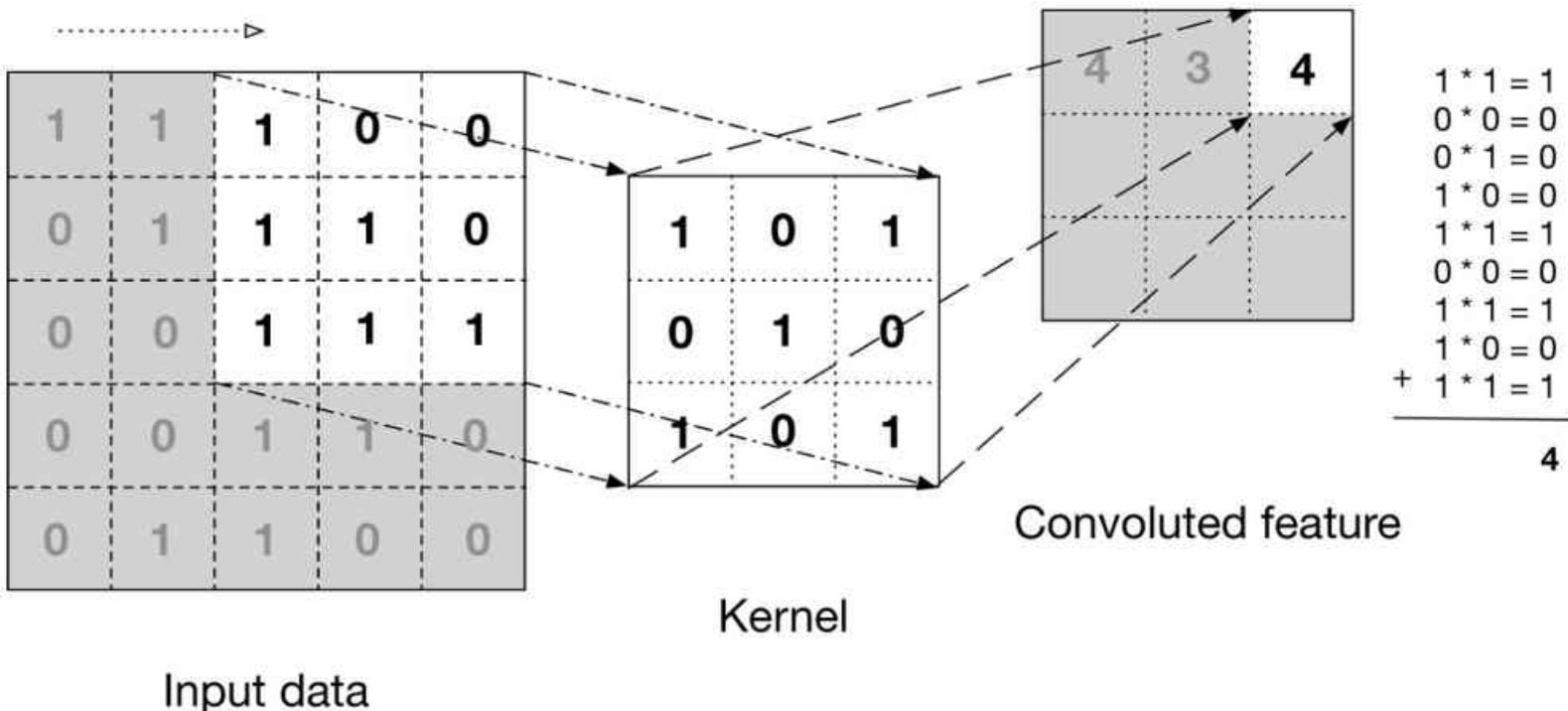
Kernel (2x2)
Stride 1
Bias = 2

| | |
|----|---|
| -1 | 2 |
| 0 | 1 |

Feature map (2x2)

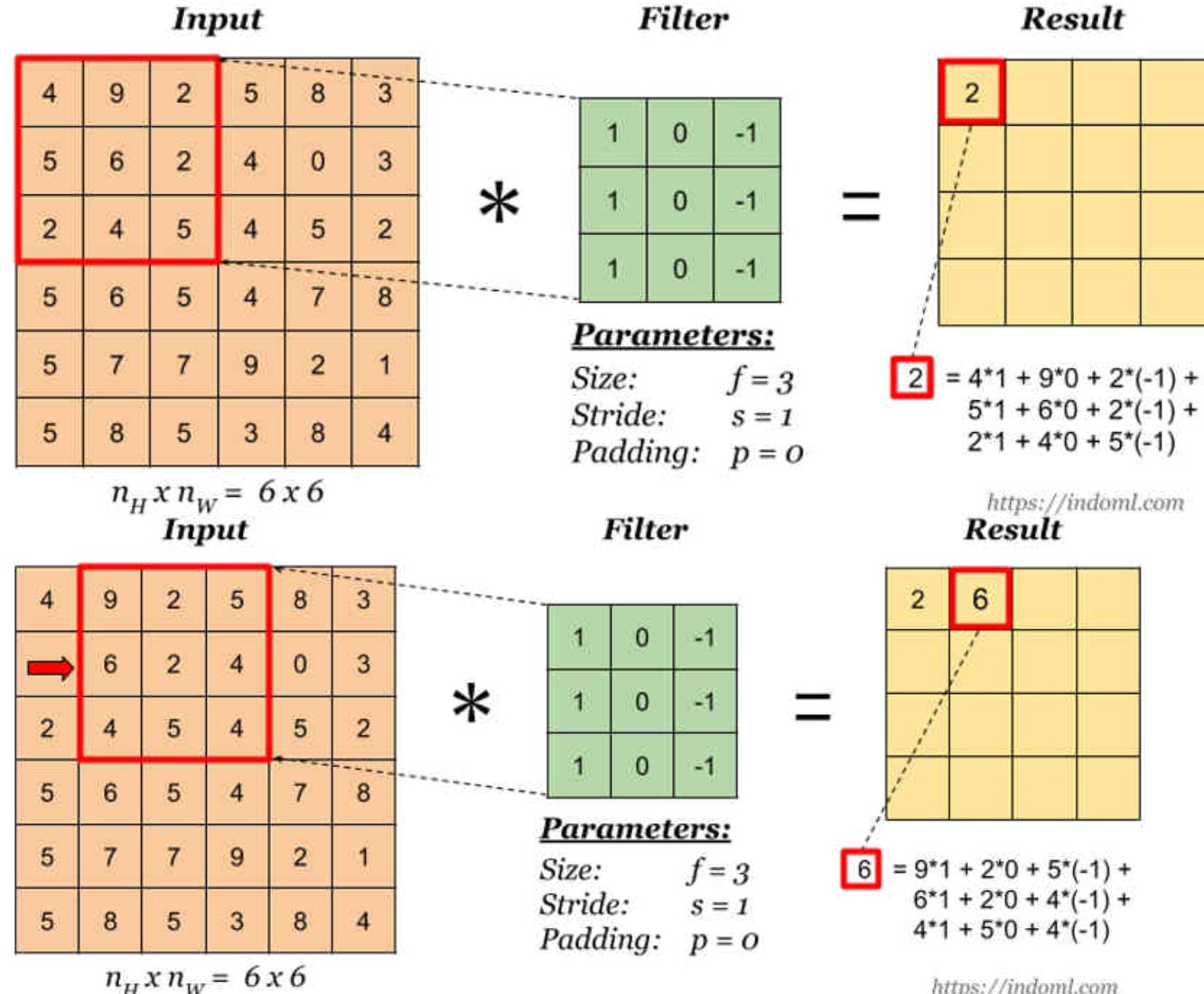
What is a Convolution?

Without Bias



We can use an input image and a filter to produce an output image by **convolving** the filter with the input image. This consists of

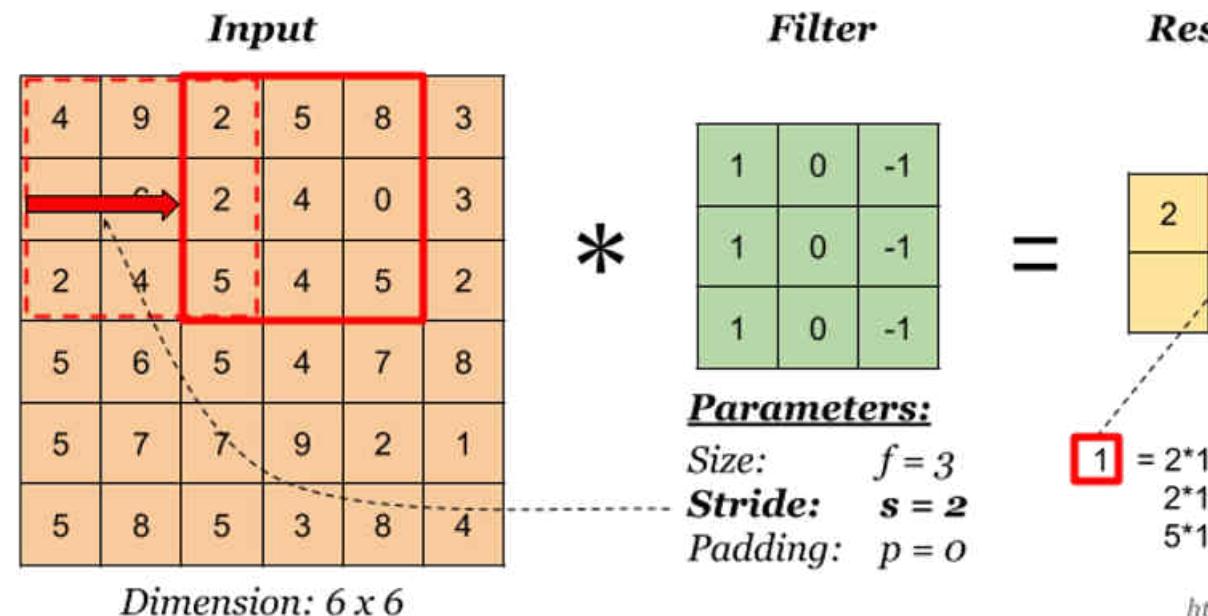
1. Overlaying the filter on top of the image at some location.
2. Performing **element-wise multiplication** between the values in the filter and their corresponding values in the image.
3. Summing up all the element-wise products. This sum is the output value for the **destination pixel** in the output image.
4. Repeating for all locations.



Machine Intelligence

Some terminologies

Stride: Stride governs how many cells the filter is moved in the input to calculate the next cell in the result.



<https://indoml.com>

Machine Intelligence

Some terminologies

- For a gray scale $(n \times n)$ image and $(f \times f)$ filter/kernel, the dimensions of the image resulting from a convolution operation is $(n - f + 1) \times (n - f + 1)$.

- For example, for an (8×8) image and (3×3) filter, the output resulting after convolution operation would be of size (6×6) . Thus, the image shrinks every time a convolution operation is performed.

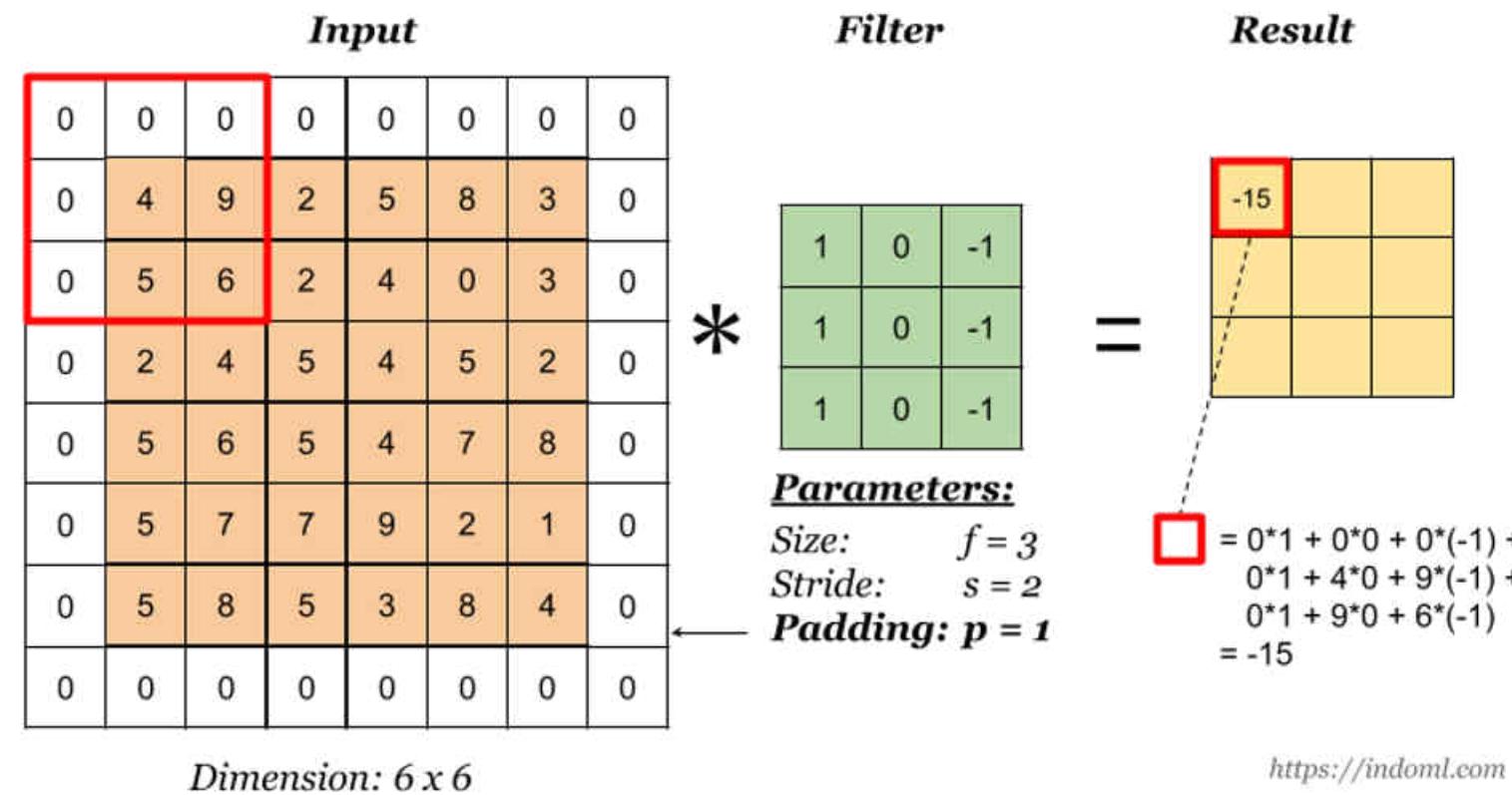
Machine Intelligence

Some terminologies

Padding works by extending the area (generally by adding 0 to the boundary) of which a convolutional neural network processes an image.

Padding has the following benefits:

- It allows to use a CONV layer without necessarily shrinking the height and width of the volumes.
- This is important for building deeper networks, since otherwise the height/width would shrink as we go to deeper layers.



Machine Intelligence

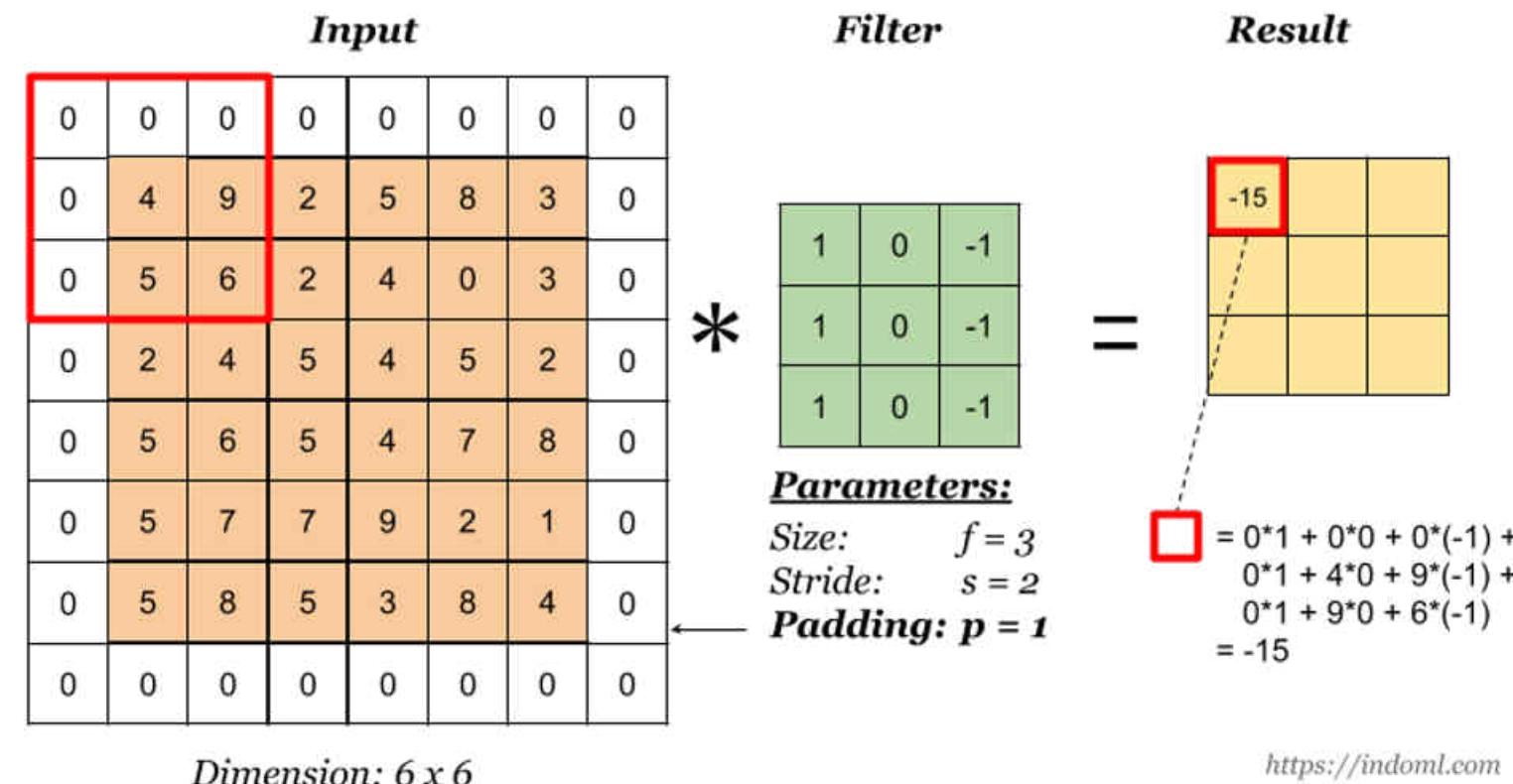
Some terminologies

- This prevents shrinking as, if p = number of layers of zeros added to the border of the image, then our $(n \times n)$ image becomes $(n + 2p) \times (n + 2p)$ image after padding.
- So, applying convolution-operation (with $(f \times f)$ filter) **outputs** $(n + 2p - f + 1) \times (n + 2p - f + 1)$ images.
- For example, adding one layer of padding to an (8×8) image and using a (3×3) filter we would get an (8×8) output after performing convolution operation.

Machine Intelligence

Some terminologies

- It helps us **keep more of the information at the border of an image**. Without padding, very few values at the next layer would be affected by pixels as the edges of an image.



What is a Convolution? Padding

Consider the i/p matrix of the previous slide. The **pixel in the corner will only get covered once** but **the middle pixel will get covered more than once** which basically means that there is more info on that middle pixel so these are the two main downsides

1. Shrinking outputs
2. Loosing information on corners of the image

Padding is the **amount of pixels added to an image/input when it is being processed by the kernel of a CNN.**

For example, if the padding in a CNN is set to zero, then every pixel value that is added will be of value zero. If, however, the zero padding is set to one, there will be a one pixel border added to the image with a pixel value of zero.

What is a Convolution? Padding

| | | | | | | |
|---|-----|-----|-----|-----|-----|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 105 | 102 | 100 | 97 | 96 | |
| 0 | 103 | 99 | 103 | 101 | 102 | |
| 0 | 101 | 98 | 104 | 102 | 100 | |
| 0 | 99 | 101 | 106 | 104 | 99 | |
| 0 | 104 | 104 | 104 | 100 | 98 | |

Image Matrix

| Kernel Matrix | | |
|---------------|----|----|
| 0 | -1 | 0 |
| -1 | 5 | -1 |
| 0 | -1 | 0 |

$$\begin{aligned} & 0 * 0 + 0 * -1 + 0 * 0 \\ & + 0 * -1 + 105 * 5 + 102 * -1 \\ & + 0 * 0 + 103 * -1 + 99 * 0 = 320 \end{aligned}$$

| | | | |
|-----|--|--|--|
| 320 | | | |
| | | | |
| | | | |
| | | | |
| | | | |

Output Matrix

Convolution with horizontal and vertical strides = 1

What is a Convolution? Stride=2

| | | | | | | | |
|---|-----|-----|-----|-----|-----|-----|-----|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 105 | 102 | 100 | 97 | 96 | | |
| 0 | 103 | 99 | 103 | 101 | 102 | 101 | |
| 0 | 101 | 98 | 104 | 102 | 100 | 99 | 101 |
| 0 | 99 | 101 | 106 | 104 | 99 | | |
| 0 | 104 | 104 | 104 | 100 | 98 | 101 | |

Kernel Matrix

| | | |
|----|----|----|
| 0 | -1 | 0 |
| -1 | 5 | -1 |
| 0 | -1 | 0 |

| | | |
|-----|--|--|
| 320 | | |
| | | |
| | | |
| | | |

$$\begin{aligned} & 0 * 0 + 0 * -1 + 0 * 0 \\ & + 0 * -1 + 105 * 5 + 102 * -1 \\ & + 0 * 0 + 103 * -1 + 99 * 0 = 320 \end{aligned}$$

Convolution with horizontal and
vertical strides = 2

What is a Convolution? Multi Channel

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

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

Output

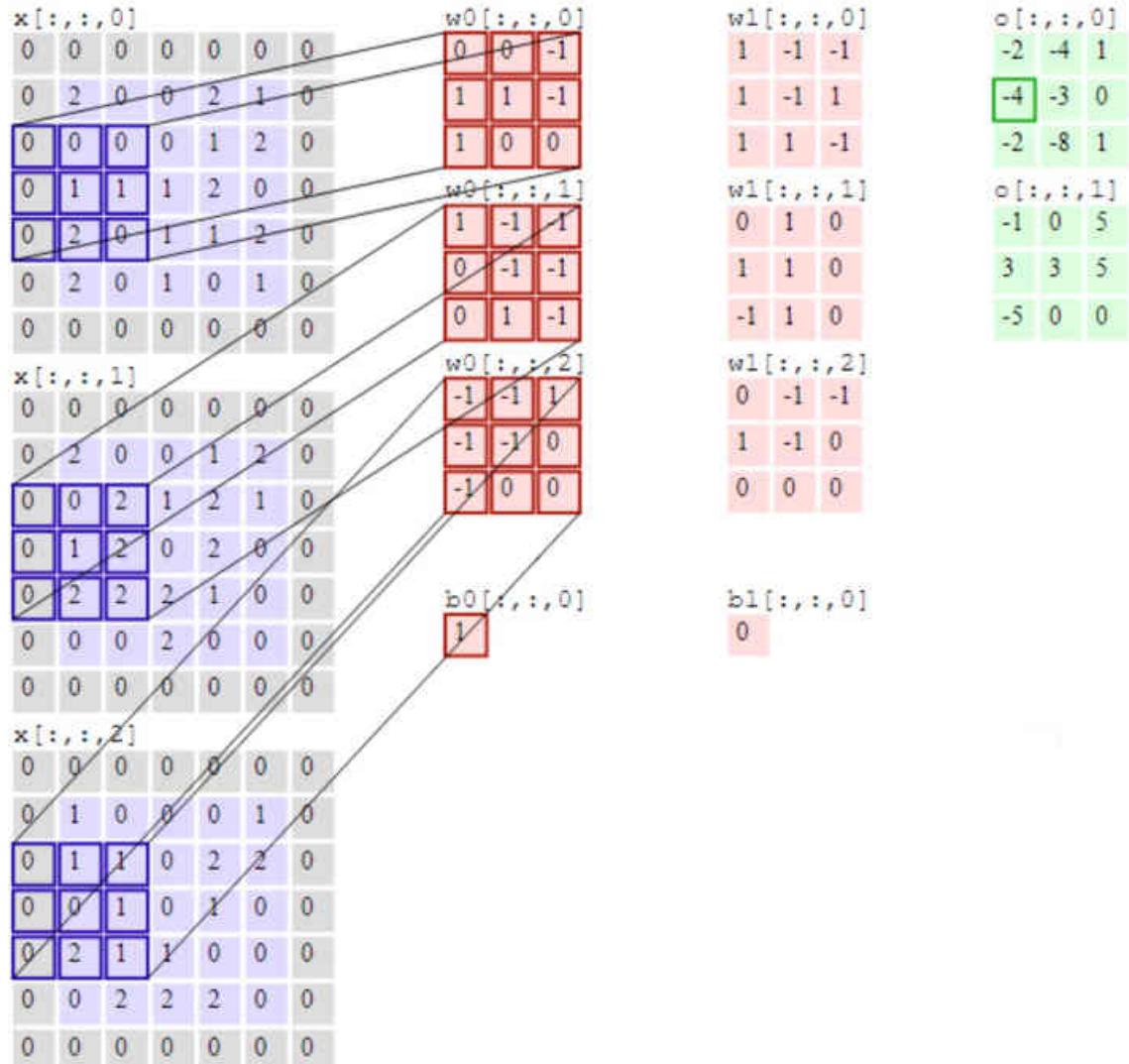
Machine Intelligence

What is a Convolution? Multi Channel

CONV layer requires four hyperparameters:

1. Number of filters K,
2. their spatial extent F,
3. the stride S,
4. the amount of zero padding P.

- The input volume is of size $W_1=5$, $H_1=5$, $D_1=3$, and the CONV layer parameters are $K=2, F=3, S=2, P=1$.
- That is, we have two filters of size 3×3 , and they are applied with a stride of 2. Therefore, the output volume size has spatial size $(5 - 3 + 2)/2 + 1 = 3$.
- The visualization below iterates over the output activations (green), and shows that each element is computed by elementwise multiplying the highlighted input (blue) with the filter (red), summing it up



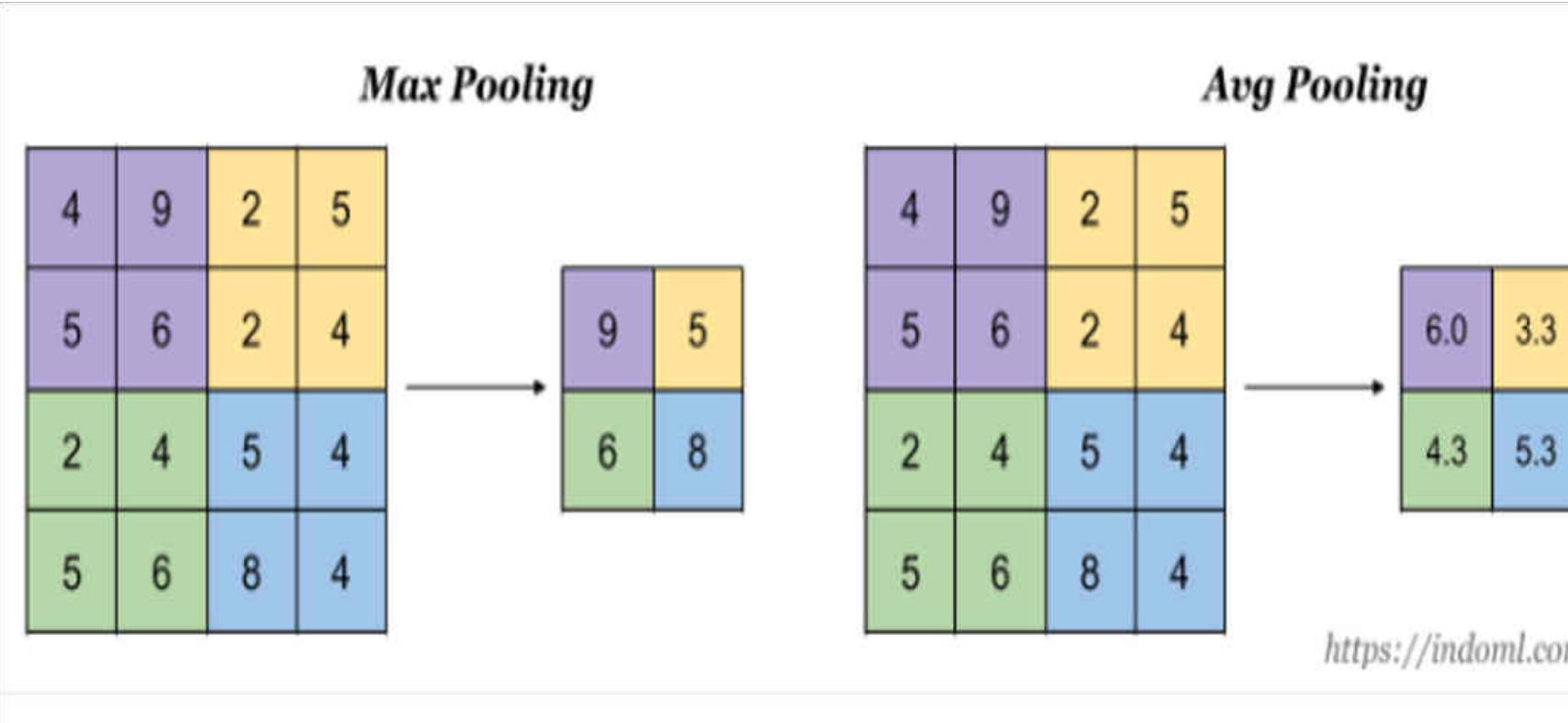
What is Pooling or downsampling?

- Is a process of applying some operation over regions in the **input feature map** and **extracting some representative value** for each of the analyzed regions.

- Two most common pooling operations are:
 - **Max-pool**
 - **Average-pool**

Machine Intelligence

CNN



What is a Convolution? Multi Channel

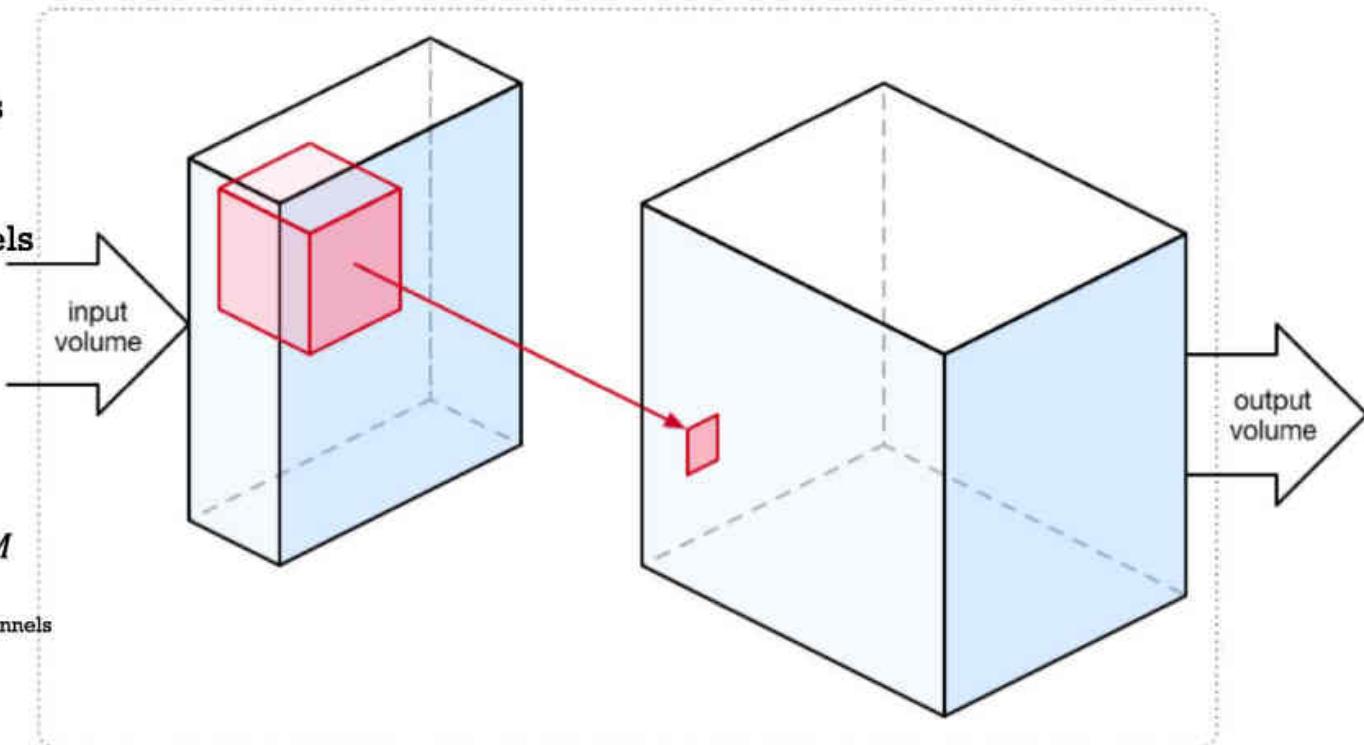
N: Number of input channels
W: Width of the kernel
H: Height of the kernel
M: Number of output channels

$$\text{Kernel size} = N * W * H$$

$$\#Params = M * N * W * H + M$$

256 convolutions of kernel (3,3) on 256 input channels

$$256 * 256 * 3 * 3 = \sim 0.5M$$



Convolution computations are:

- Independent (across filters and within filter)
- Simple (multiplication and sums)

Strengths of CNN

- The main strengths of CNNs are to provide **an efficient dense network which performs the prediction or identification etc. efficiently.**
- CNNs are the most popular topic in the pool of deep learning, which is indeed very vast, and **this is usually because of the ConvNets.**
- Immense datasets are applied to CNNs, it is even considered that **larger the data, greater the accuracy will result**, otherwise operations such as transfer learning shall be applied to expand the data.
- The power of CNN is to detect distinct features from images all by itself, without any actual human intervention.
- The most popular dataset that CNN picks the features from are the Cats and Dogs dataset where each feature is picked automatically and the pictures are classified as dogs or cats.
- They are easily parallelizable also.

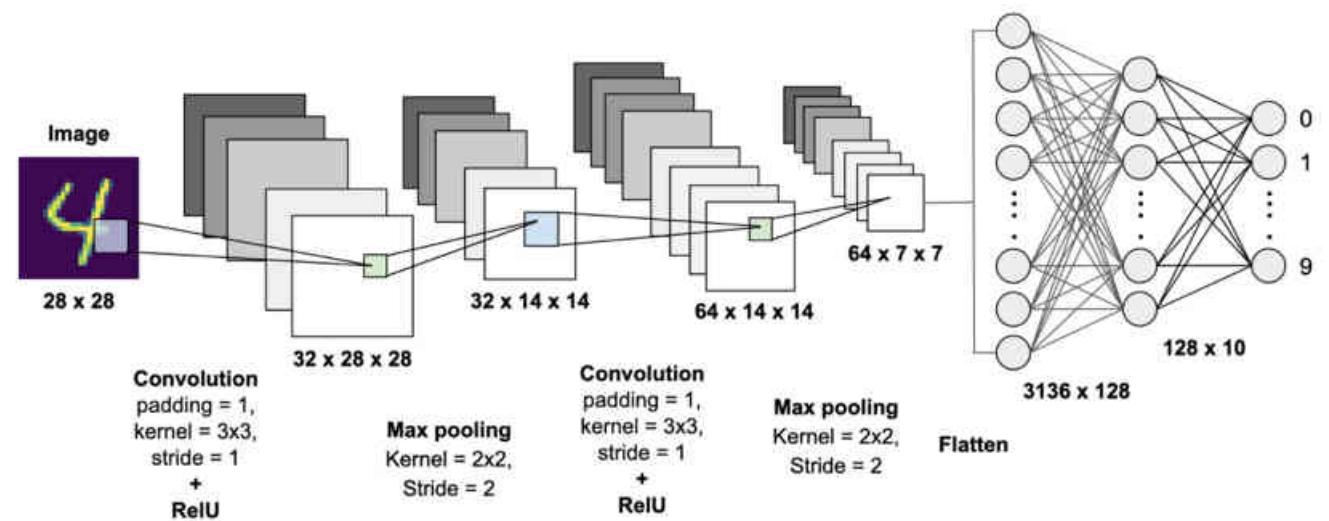
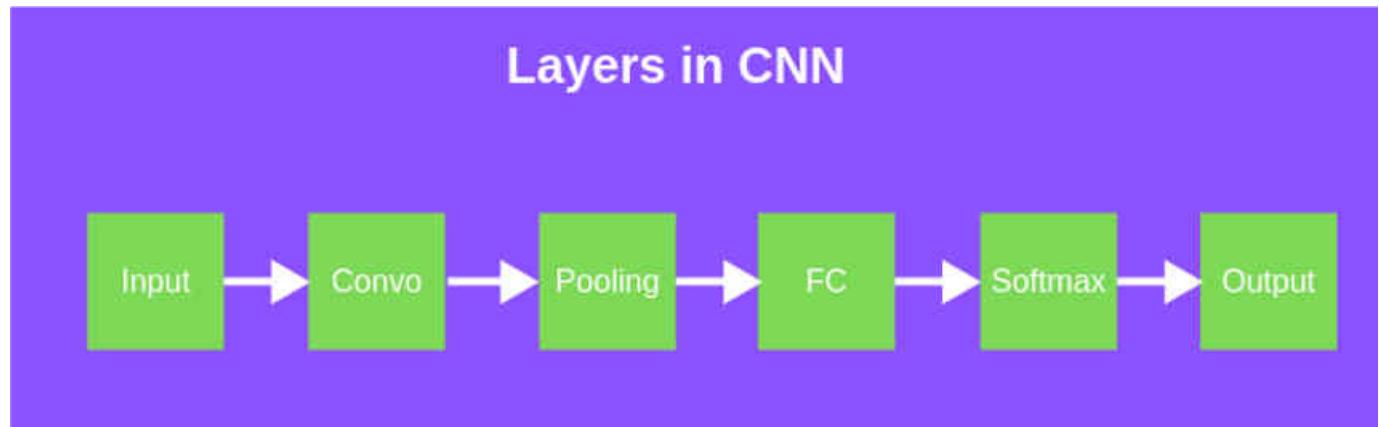
Machine Intelligence

CNN

Layers in CNN

There are five different layers in CNN

- Input layer
- Convo layer (Convo + ReLU)
- Pooling layer
- Fully connected(FC) layer
- Softmax/logistic layer
- Output layer

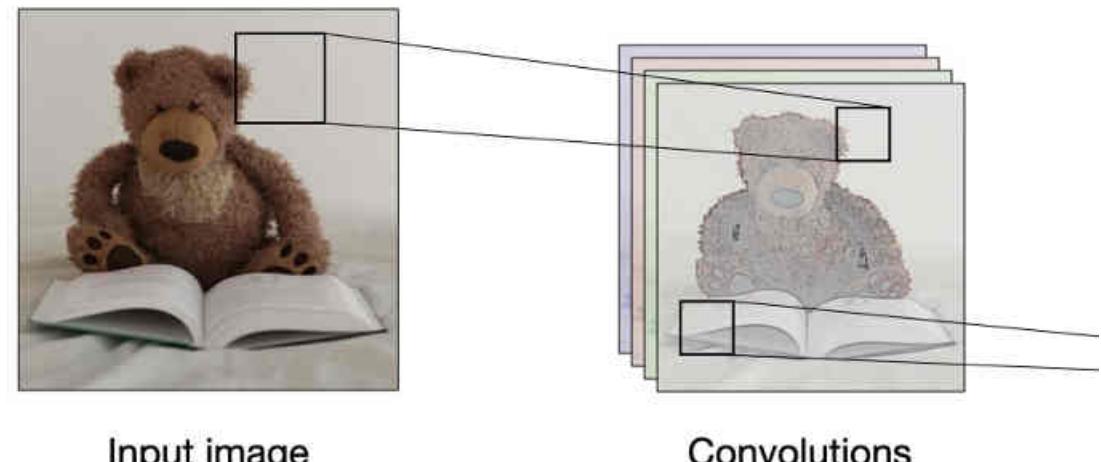


Machine Intelligence

CNN

Input layer: Contains input image, suppose it is of dimensions 28×28 it needs to be reshaped into a single column. $28 \times 28 = 784$, so we convert it to 784×1 before feeding it to input. If there are m training examples then dimension of input will be $(784, m)$

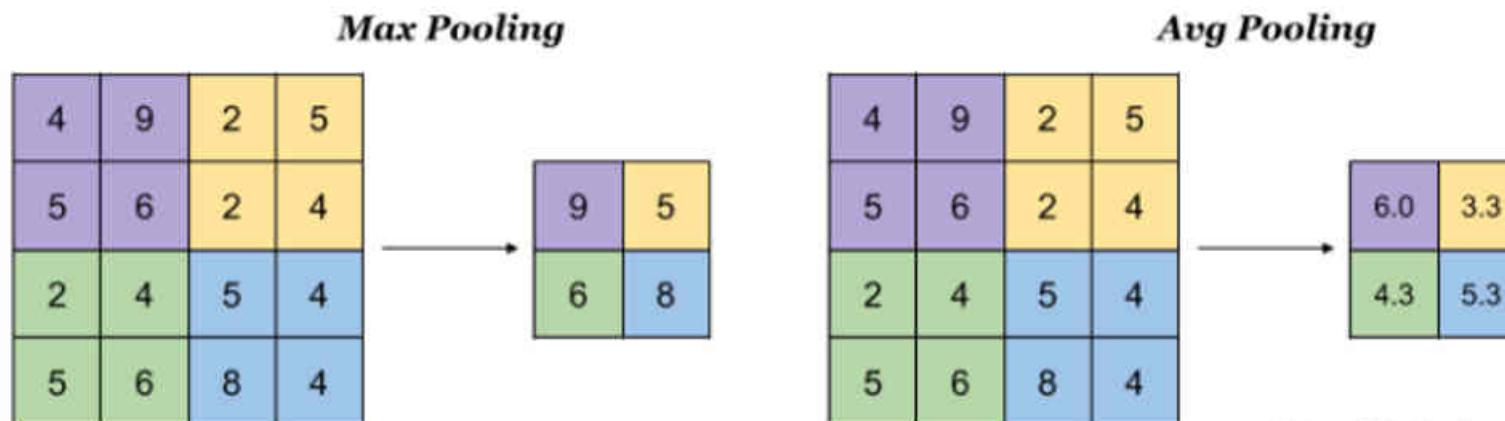
Convo layer : Perform convolution operation and results in feature extraction.



Machine Intelligence

CNN

- **Pooling layer:** Pooling layer is used to reduce the size of the representations and to speed up calculations, as well as to make some of the features it detects a bit more robust.
- Sample types of pooling are max pooling and avg pooling.



<https://indoml.com>

- The convolution layers are used to help the computer determine features that could be missed in simply flattening an image into its pixel values. The convolution layers are typically split into two sections, convolutions and pooling.

Machine Intelligence

CNN

Fully Connected Layer(FC)

Fully connected layer involves **weights, biases, and neurons**. It connects neurons in one layer to neurons in another layer. It is used to classify images between different category by training.

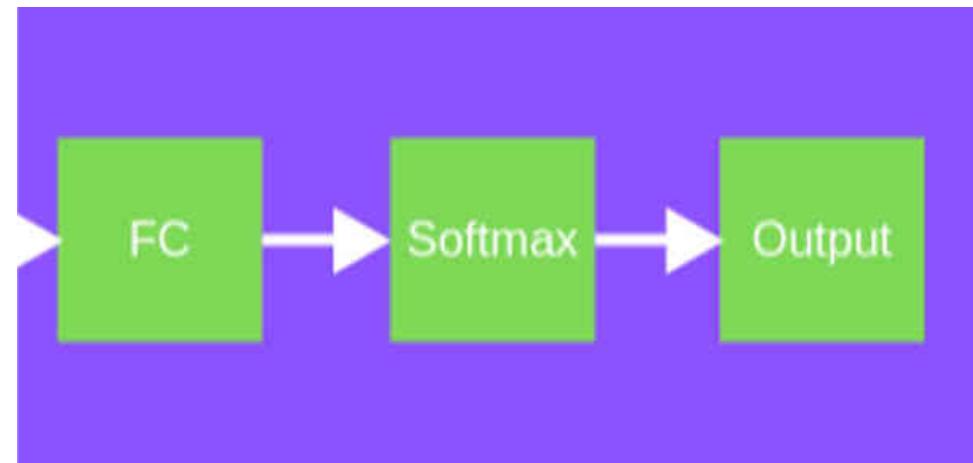
Softmax / Logistic Layer

Softmax or Logistic layer is the **last layer of CNN**. It resides at the end of FC layer. Logistic is used for binary classification and softmax is for multi-classification.

Output Layer

Output layer contains the label which is in the form of **one-hot encoded**.

*Note that this is the general flow of a CNN



Machine Intelligence

One-hot-encoding

One-hot-encoding:

- One needs to **prepare the data in specific ways** before fitting a machine learning model.
- One good example is to use a **one-hot encoding on categorical data**.
- Suppose, In the “color” variable example, there are 3 categories and therefore 3 binary variables are needed.
- A “1” value is placed in the binary variable for the color and “0” values for the other colors.

| | 1 | red, | green, | blue |
|---|----|------|--------|------|
| 1 | 1, | 0, | 0 | |
| 2 | 0, | 1, | 0 | |
| 3 | 0, | 0, | 1 | |

One hot encoding is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction.

For which purpose Convolutional Neural Network is used?

✓ Mainly to process and analyse digital images, with some success cases involving processing voice and natural language.

✗ Mainly to process and analyse financial models, predicting future trends.

✗ It is a multi purpose alghorithm that can be used for Supervised Learning.

✗ It is a multi purpose alghorithm that can be used for Unsupervised Learning.

CNN has some components and parameters which works well with images. That's why it's mainly used to analyse and predict images.

Why Convolutions?

- In neural network usage, "dense" connections connect all inputs.
- But if you consider images used for computer vision are 224×224 or larger and they include **3 colour channels** so $224 \times 224 \times 3 = 150,528$ input features!
- As this propagates into further layers over a **million weights** are required in first layer alone. In this case there is no sharing and connections are made to all inputs.
- By contrast, a CNN is "sparse" because only the local "patch" of pixels is connected, instead using all pixels as an input.
- Let's look into this with an example image,

Why Convolutions?

- In a Convolution Neural Network, the approach is as shown in this image,
 - **Parameter Sharing:**
 - **Sparsity of Connections:**

PARAMETER SHARING:

- A feature detector, for example a vertical edge detector, that's useful in one part of the image is probably useful in another part of the image.
 - That is **if we apply a 3×3 filter** for detecting vertical edges at one part of an image, we can then **apply the same 3×3 filter** at **another position** in the image .

$$\begin{array}{|c|c|c|c|c|c|} \hline
 10 & 10 & 10 & 0 & 0 & 0 \\ \hline
 10 & 10 & 10 & 0 & 0 & 0 \\ \hline
 10 & 10 & 10 & 0 & 0 & 0 \\ \hline
 10 & 10 & 10 & 0 & 0 & 0 \\ \hline
 10 & 10 & 10 & 0 & 0 & 0 \\ \hline
 10 & 10 & 10 & 0 & 0 & 0 \\ \hline
 \end{array}
 \begin{array}{|c|c|c|} \hline
 1 & 0 & -1 \\ \hline
 1 & 0 & -1 \\ \hline
 1 & 0 & -1 \\ \hline
 \end{array}
 =
 \begin{array}{|c|c|c|c|} \hline
 0 & 30 & 30 & 0 \\ \hline
 0 & 30 & 30 & 0 \\ \hline
 0 & 30 & 30 & 0 \\ \hline
 0 & 30 & 30 & 0 \\ \hline
 \end{array}$$

- Each of these feature detectors, can use the **same parameters** in a lot of different positions in the input image in order to **detect a vertical edge or some other feature**.
 - This is also true for low-level features like edges as well as for higher-level features like maybe detection of the eye that indicates a face.

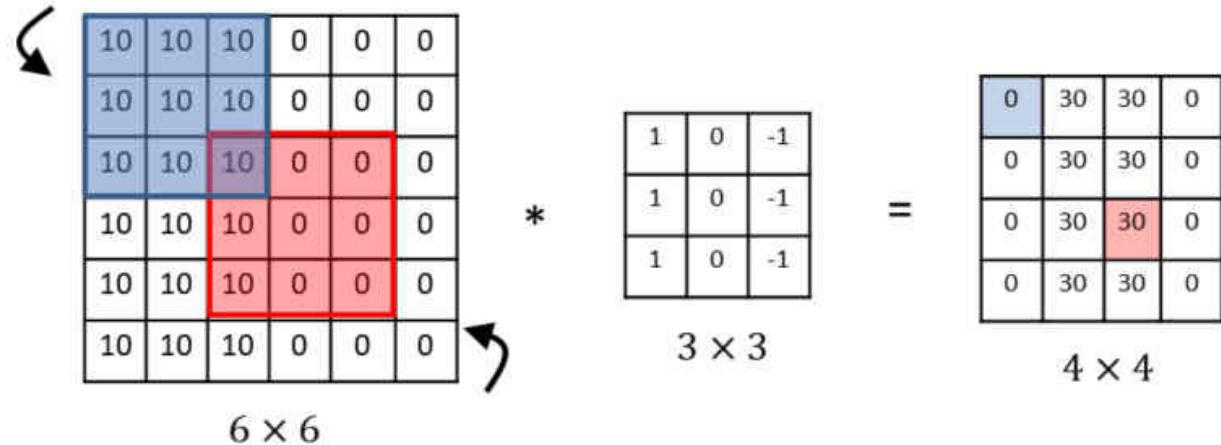
Why Convolutions?

- In a Convolution Neural Network, the approach is as shown in this image,
- Parameter Sharing:**
- Sparsity of Connections:**

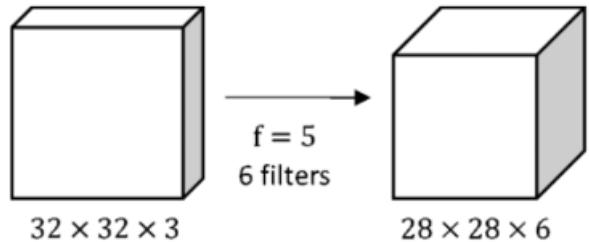
Sparsity of Connections:

Convnets are able to have relatively few parameters due to sparse connections.

- Eg, If we look at the zero (blue-upper left): it was computed by a 3×3 convolution and it depends only on this 3×3 blue input grid of cells. So, this output unit on the right is connected only to 9 out of these 36 (6×6) input features.
- In particular, the rest of these pixel values, **do not have any effect on that output**. As an another example, this red output depends only on 9 input features, and as if only those 9 input features are connected to this output.

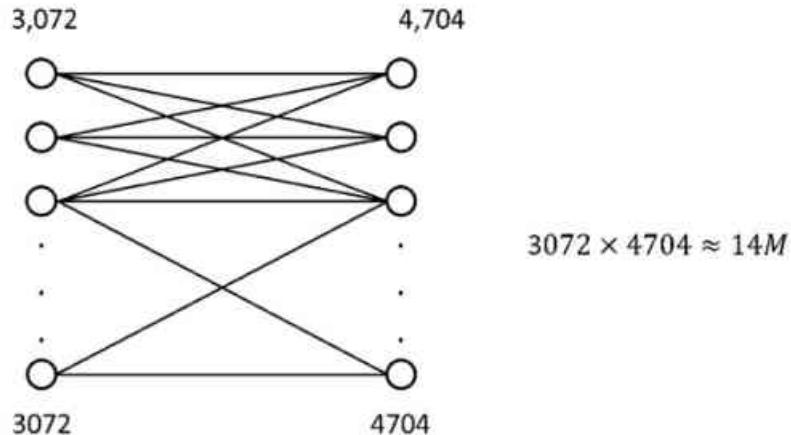


Why Convolutions?



Let's say that we have a $32 \times 32 \times 3$ dimensional image. And let's say we use 6, 5×5 filters ($f=5$), so this gives $28 \times 28 \times 6$ dimensional output.
 $((n-f+1) \times (n-f+1))$

Total # of parameters = $5 \times 5 + 1 = 26$ parameters per kernel
 So, for 6 filters = $26 \times 6 = 156$ parameters



Convolution also allows working with inputs of variable sizes

Motivation for using convolution networks

1. Convolution leverages three important ideas to improve ML systems:
 1. Sparse interactions
 2. Parameter sharing
 3. Equivariant representations
2. Convolution also allows for working with inputs of variable size

Examples of 2D Convolution applied to images



$$\begin{matrix} & 1 & 1 & 1 \\ * & 1 & 1 & 1 \\ & 1 & 1 & 1 \end{matrix} =$$

Examples of 2D Convolution applied to images



$$* \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} =$$



blurs the image

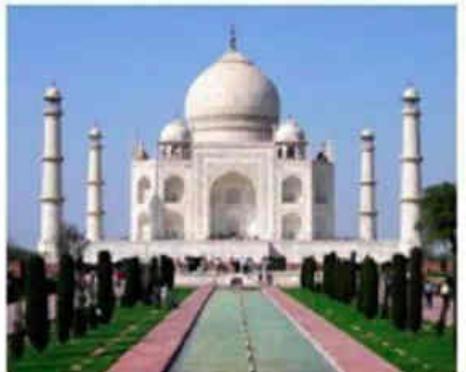
Feature map

Examples of 2D Convolution applied to images



$$\begin{matrix} * & \begin{matrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{matrix} & = \end{matrix}$$

Examples of 2D Convolution applied to images



$$\begin{matrix} * & \begin{matrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{matrix} & = \end{matrix}$$



sharpens the image

Feature map

Examples of 2D Convolution applied to images



$$\begin{matrix} * & \begin{matrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{matrix} & = \end{matrix}$$

Examples of 2D Convolution applied to images



$$* \begin{matrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{matrix} =$$



detects the edges

Feature map

Feature maps:

Apply **filters** or **feature detectors** to the input image to generate the **feature maps**.

Feature detectors or filters help **identify different features** present in an image like edges, vertical lines, horizontal lines, bends, etc.

QUIZ

What is the difference between CNN and ANN?

✖ CNN uses a more simpler algorithm than ANN.

✓ CNN has one or more layers of convolution units, which receives its input from multiple units.

✖ They complete each other, so in order to use ANN, you need to start with CNN.

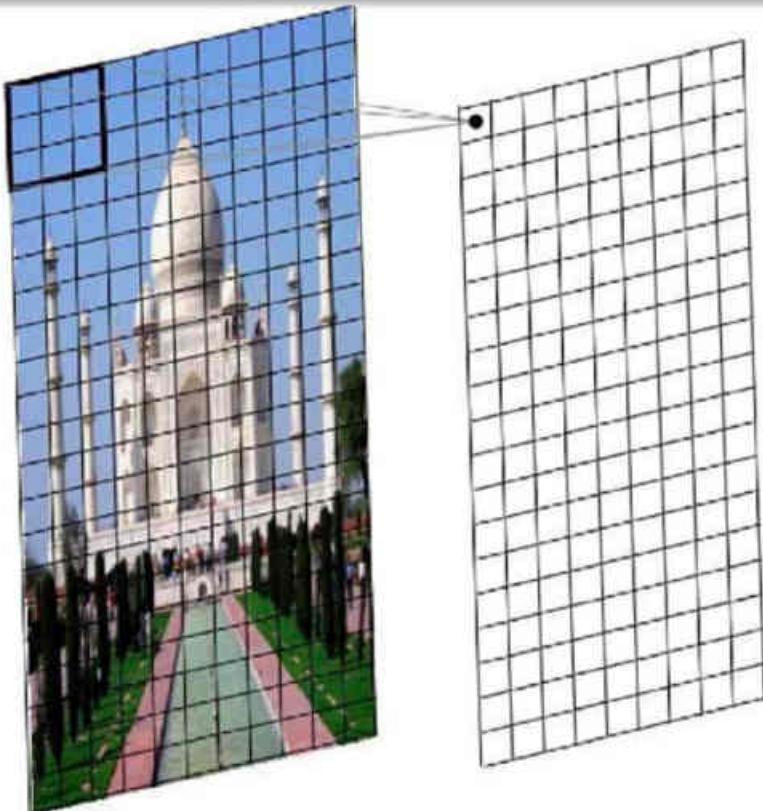
✖ CNN is a easiest way to use Neural Networks.

The only difference is the Convolutional component, which is what makes CNN good in analysing and predict data like images. The other steps are the same.

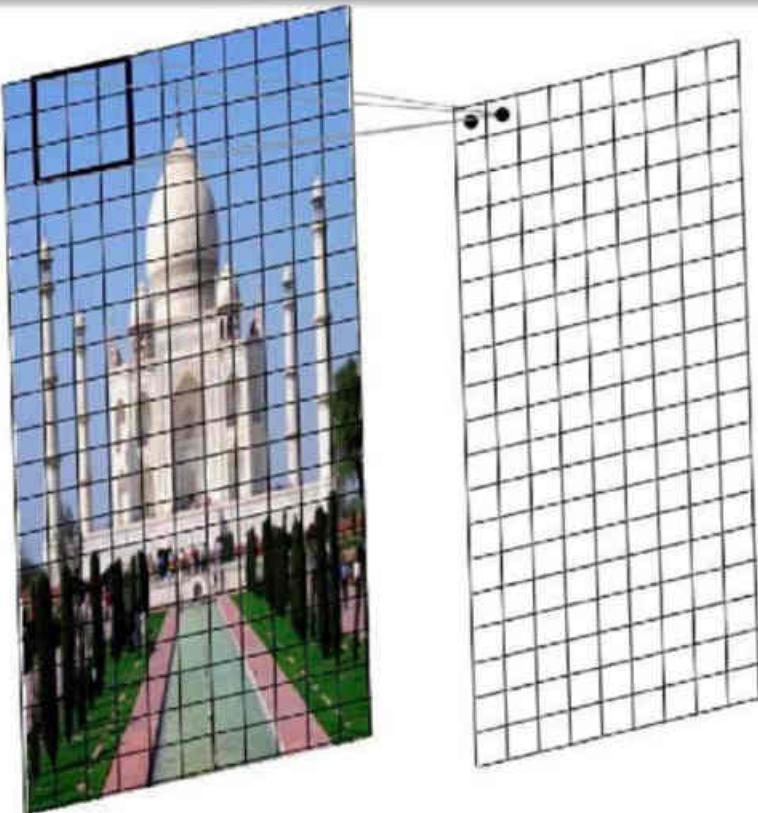
Working example of 2D Convolution



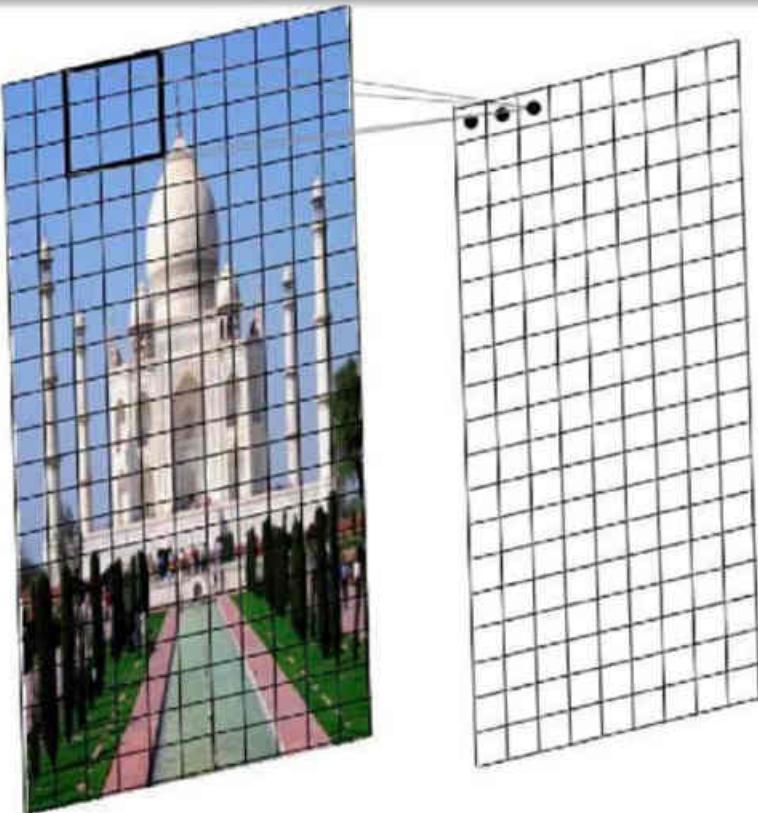
- We just slide the kernel over the input image



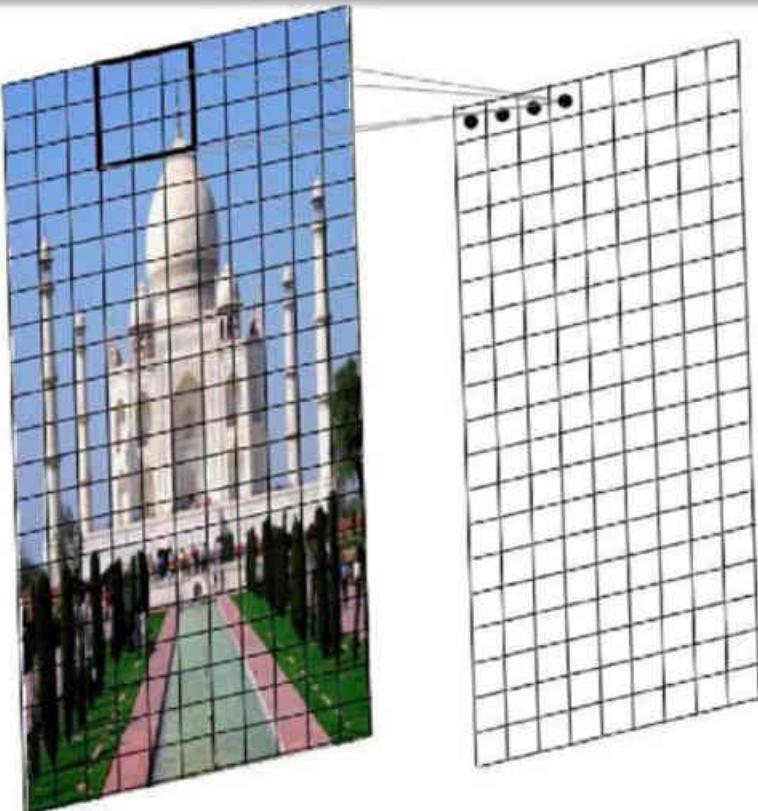
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



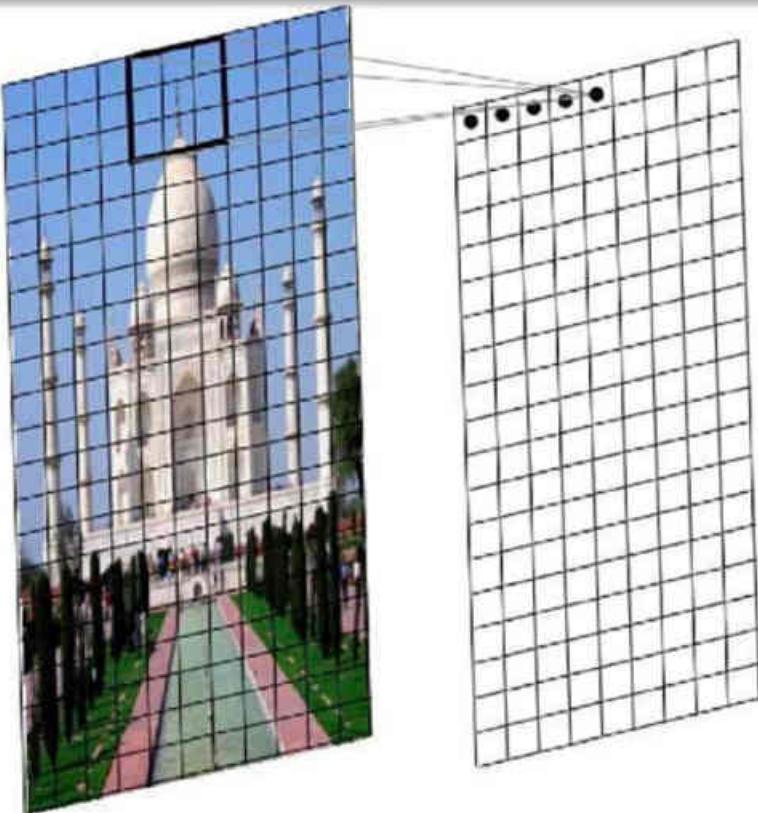
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



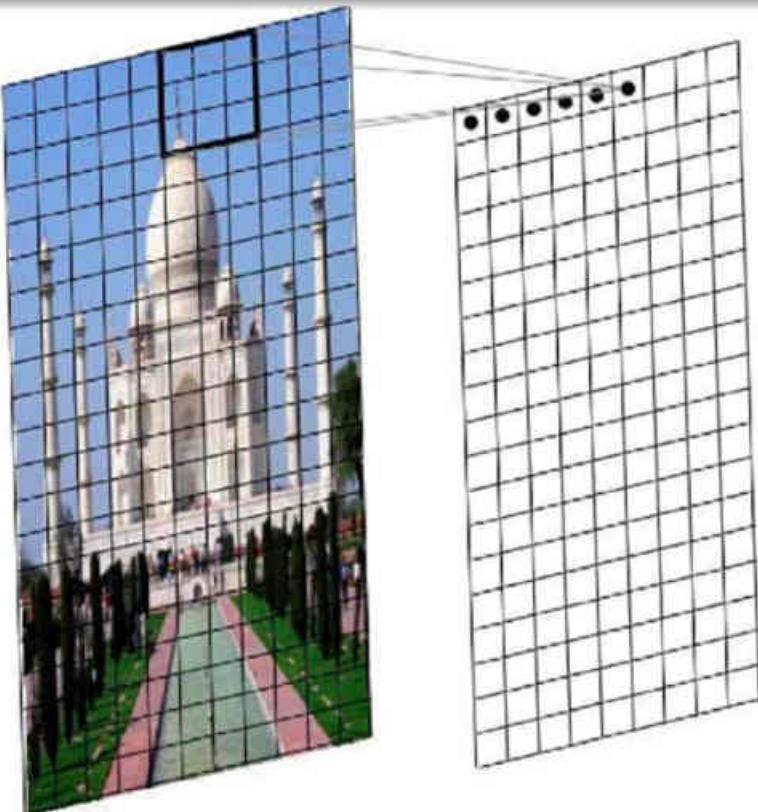
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



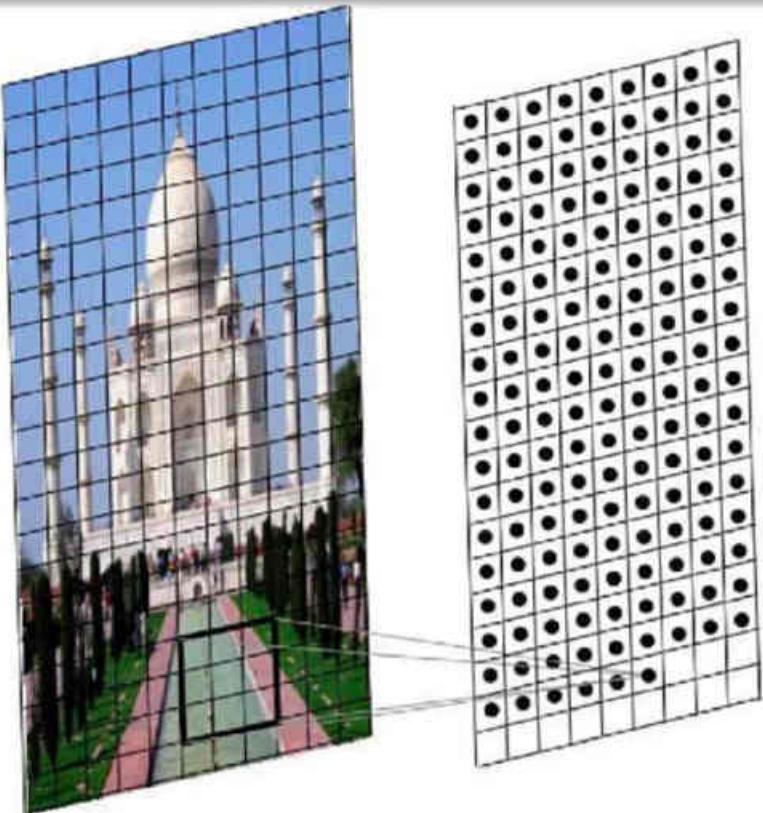
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



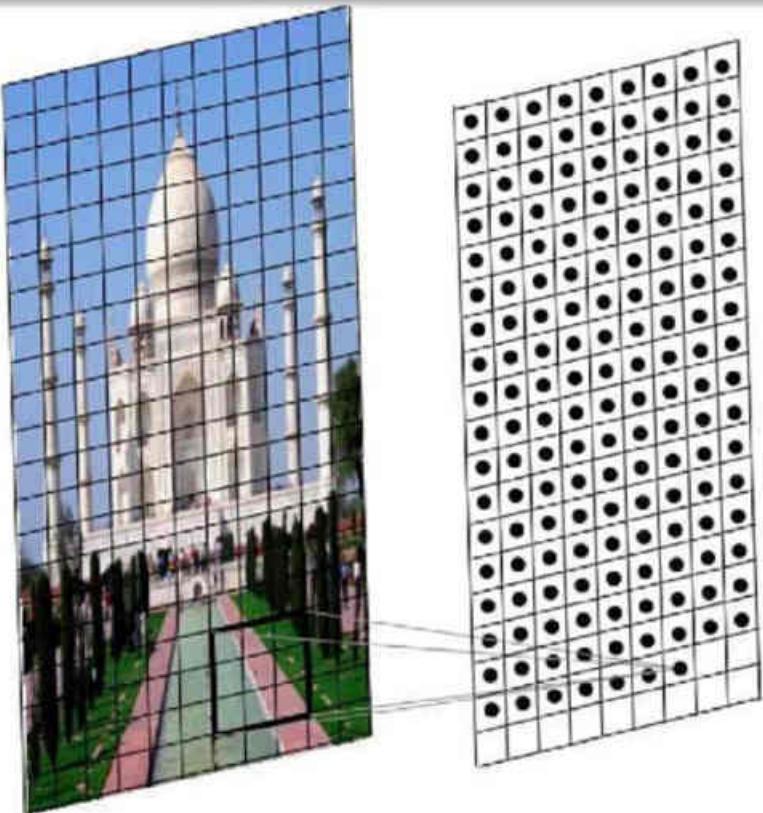
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



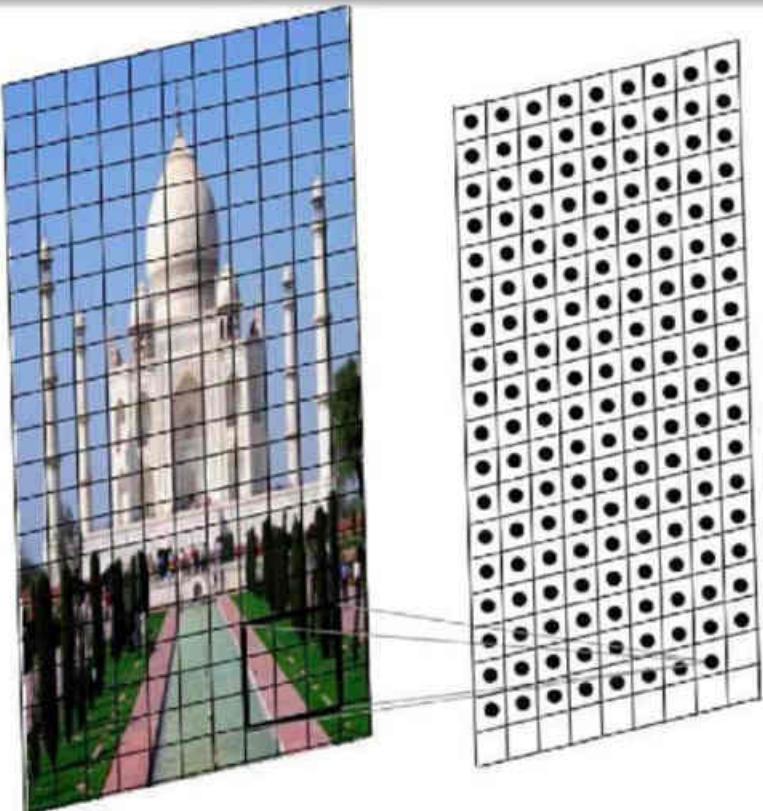
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



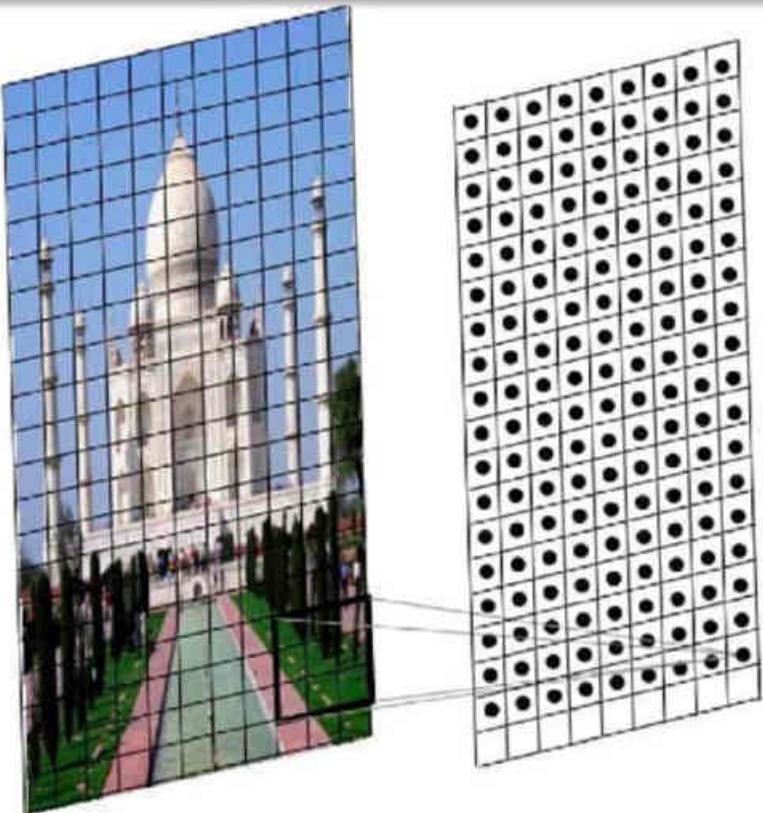
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



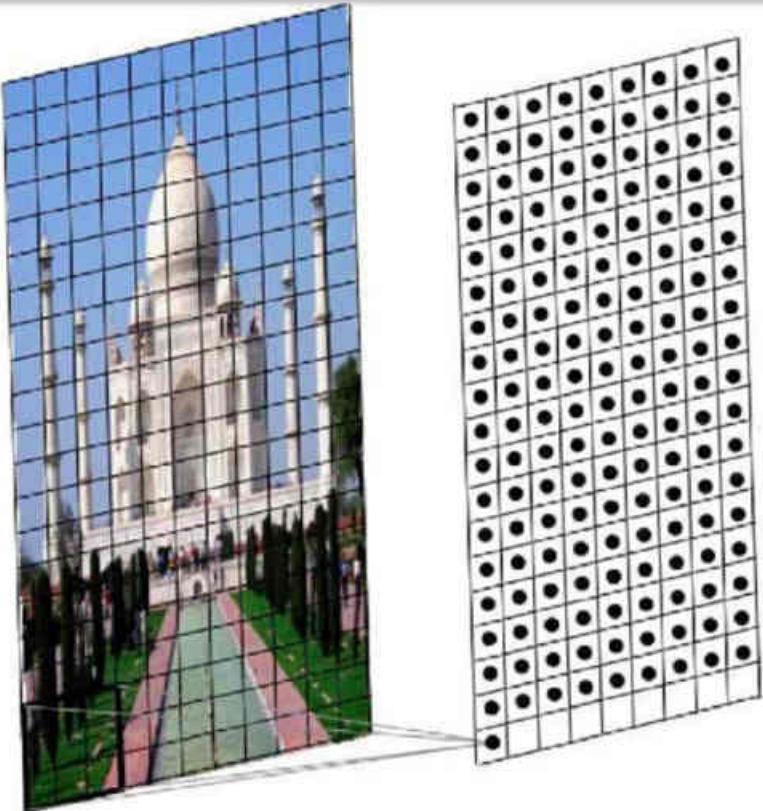
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



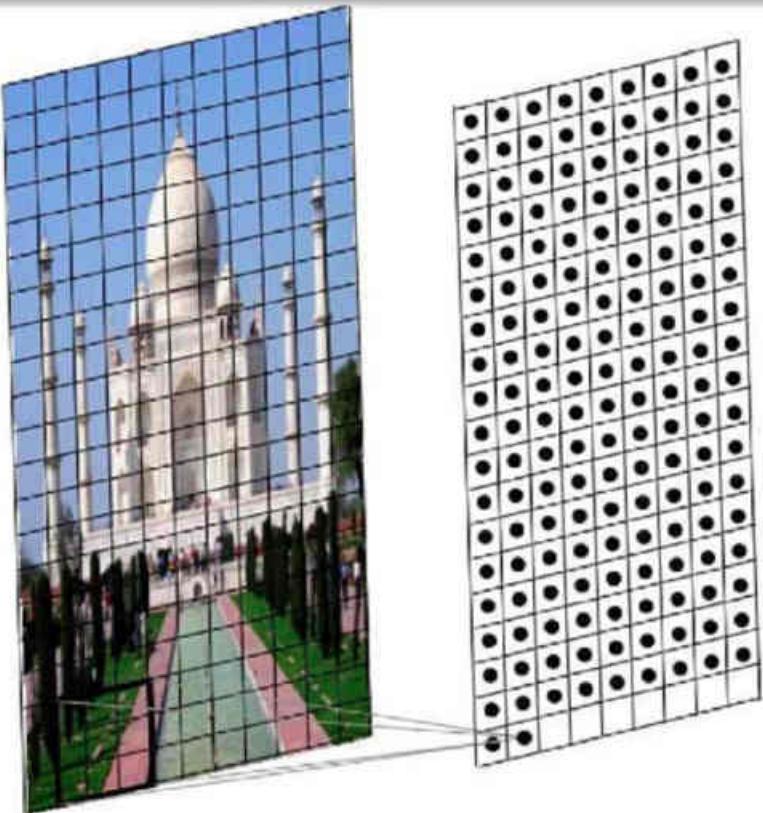
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



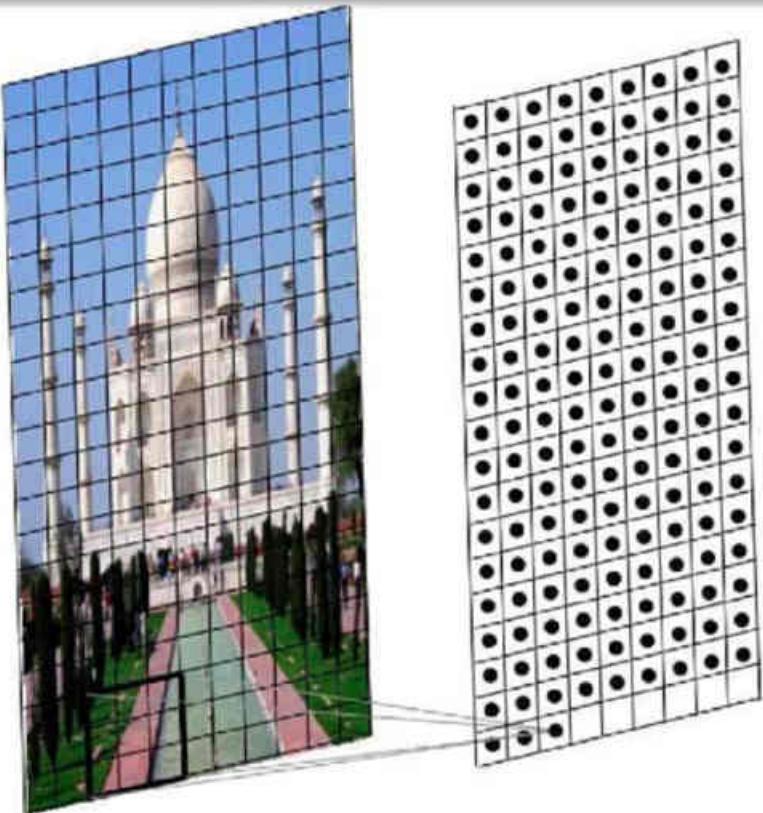
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



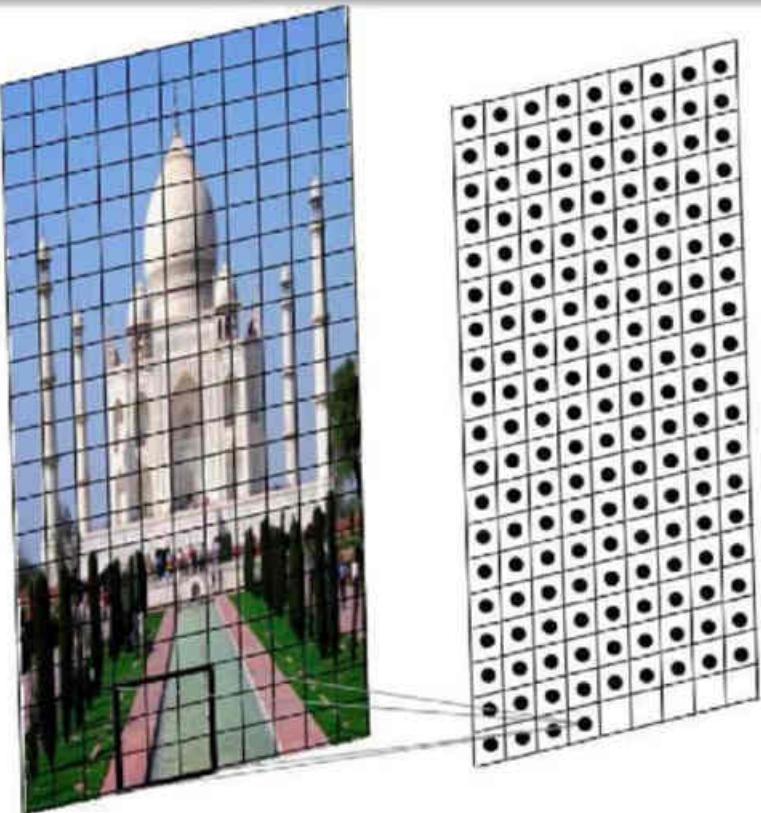
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



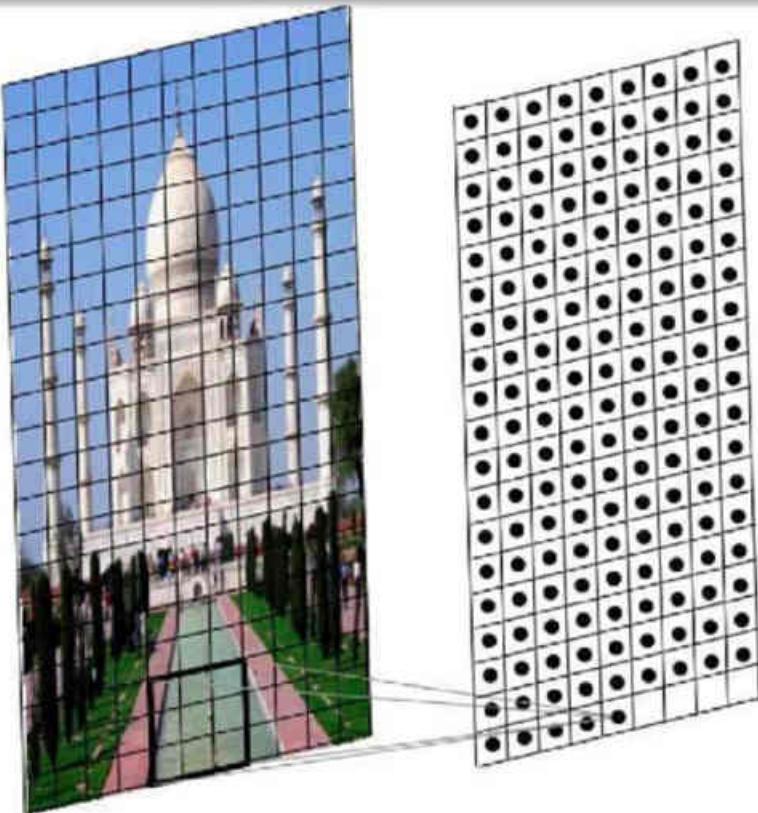
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



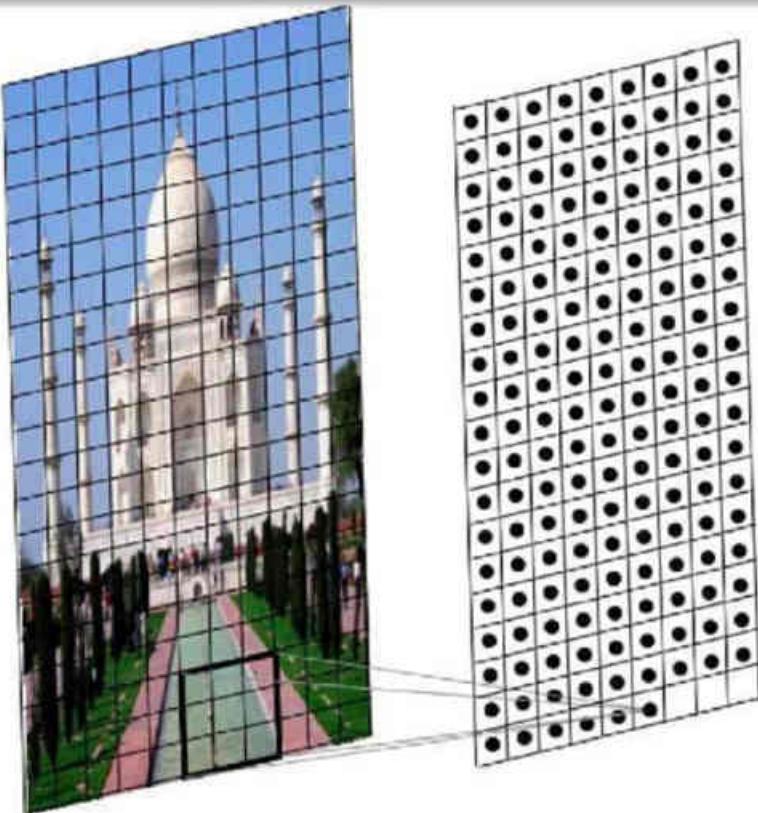
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



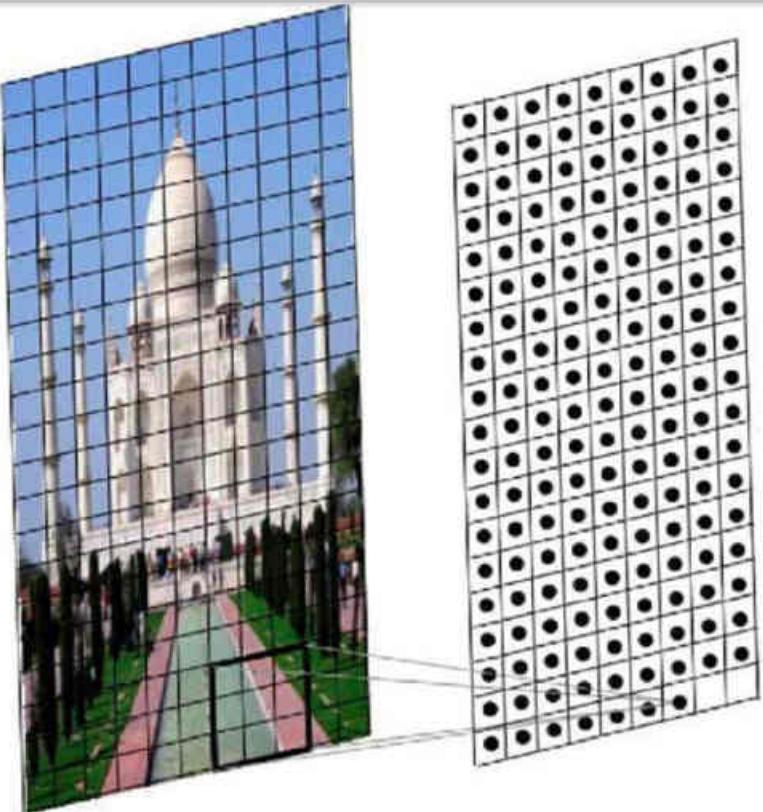
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



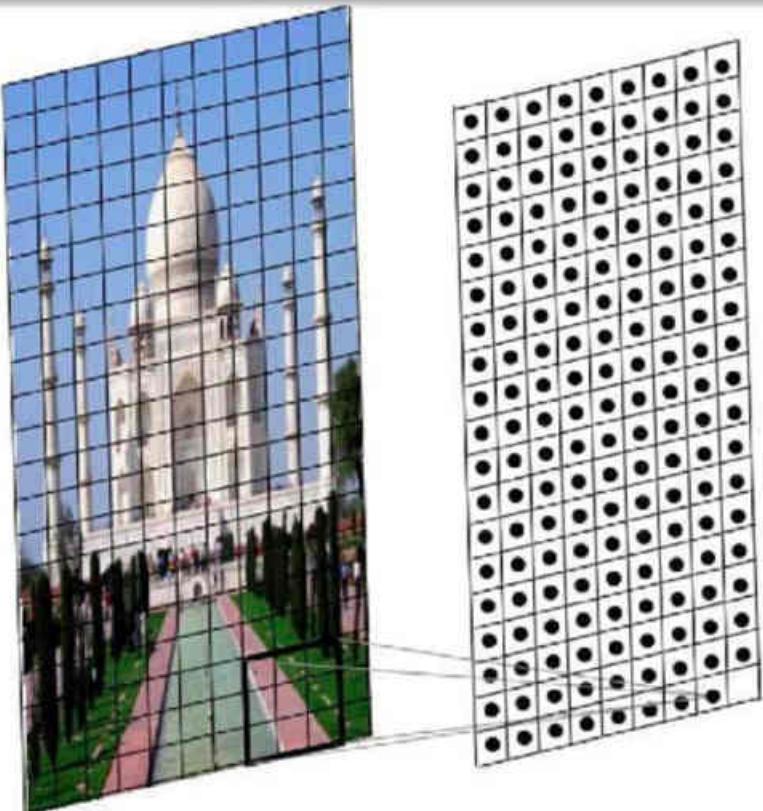
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



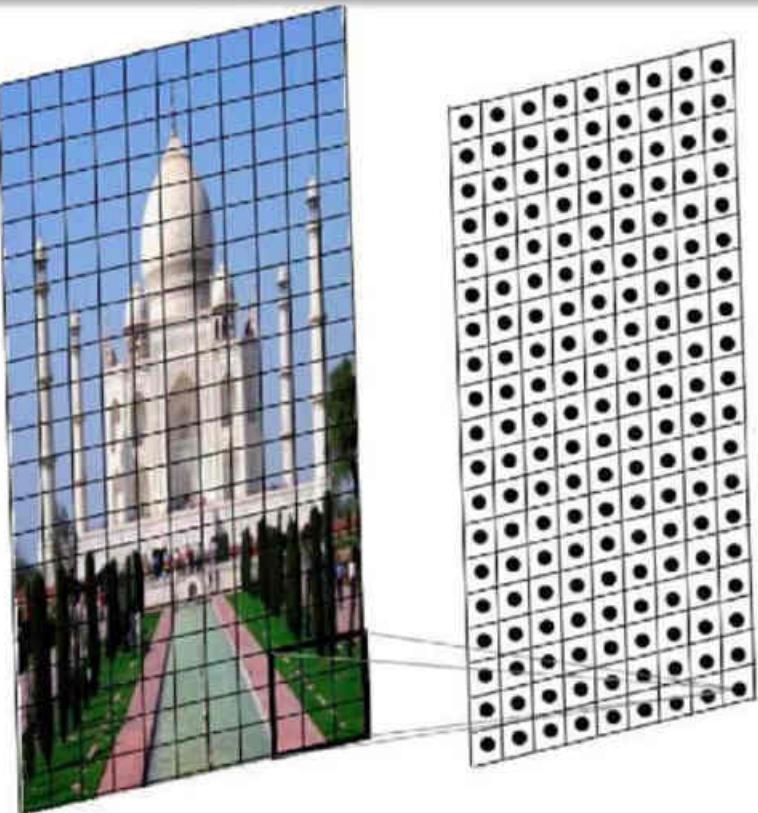
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



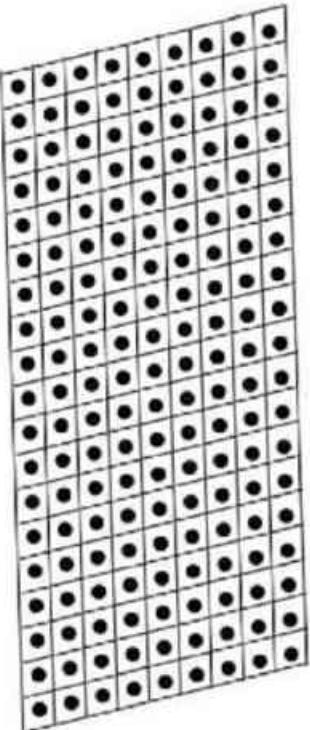
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output
- The resulting output is called a feature map.



- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output
- The resulting output is called a feature map.
- We can use multiple filters to get multiple feature maps.

Question

- In the 1D case, we slide a one dimensional filter over a one dimensional input

Question

- In the 1D case, we slide a one dimensional filter over a one dimensional input

| | | | | | | |
|---|---|---|---|---|---|---|
| A | B | C | B | A | B | C |
|---|---|---|---|---|---|---|

Question

- In the 1D case, we slide a one dimensional filter over a one dimensional input

| | | | | | | |
|---|---|---|---|---|---|---|
| A | B | C | B | A | B | C |
|---|---|---|---|---|---|---|

Question

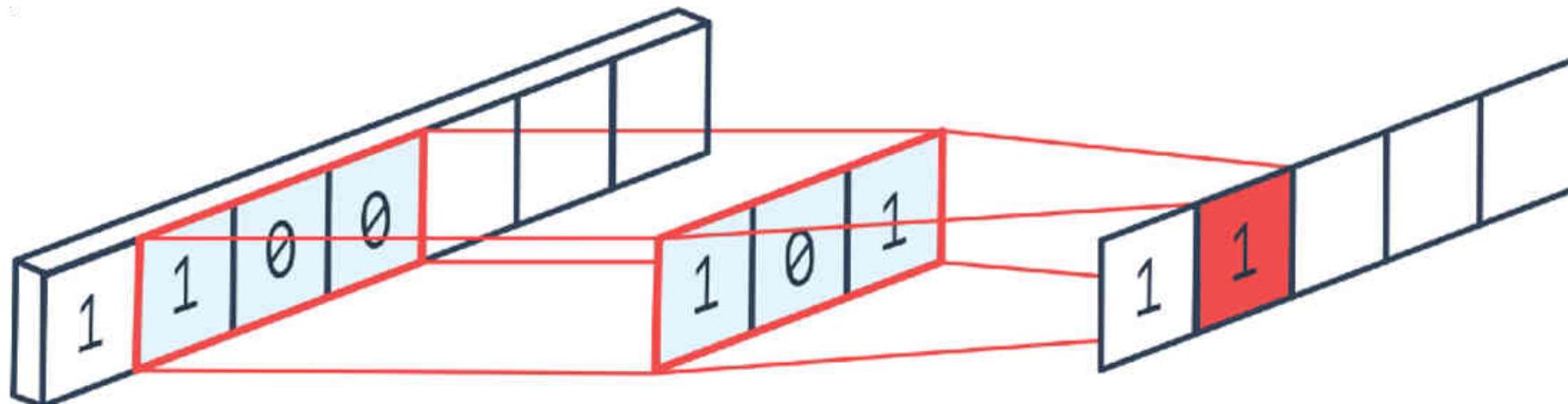
- In the 1D case, we slide a one dimensional filter over a one dimensional input

| | | | | | | |
|---|---|---|---|---|---|---|
| A | B | C | B | A | B | C |
|---|---|---|---|---|---|---|

Question

- In the 1D case, we slide a one dimensional filter over a one dimensional input

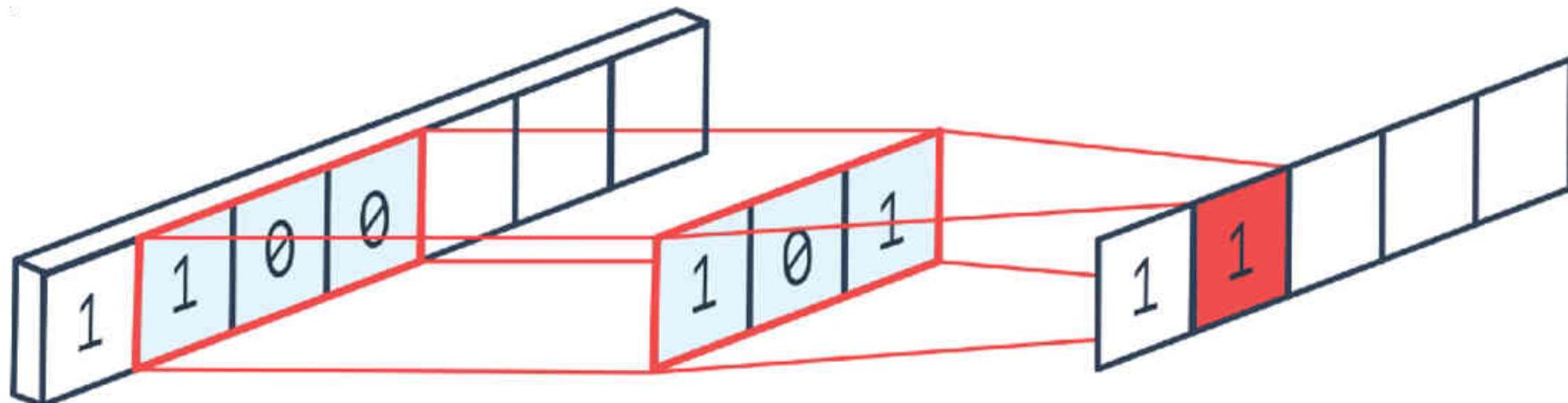
| | | | | | | |
|---|---|---|---|---|---|---|
| A | B | C | B | A | B | C |
|---|---|---|---|---|---|---|



Question

- In the 1D case, we slide a one dimensional filter over a one dimensional input

| | | | | | | |
|---|---|---|---|---|---|---|
| A | B | C | B | A | B | C |
|---|---|---|---|---|---|---|



Question

- In the 1D case, we slide a one dimensional filter over a one dimensional input
- In the 2D case, we slide a two dimensional filter over a two dimensional output

Question

- In the 1D case, we slide a one dimensional filter over a one dimensional input
- In the 2D case, we slide a two dimensional filter over a two dimensional output

| | | | |
|---|---|---|---|
| a | b | c | d |
| e | f | g | h |
| i | j | k | l |

Question

- In the 1D case, we slide a one dimensional filter over a one dimensional input
- In the 2D case, we slide a two dimensional filter over a two dimensional output

| | | | |
|---|---|---|---|
| a | b | c | d |
| e | f | g | h |
| i | j | k | l |

Question

- In the 1D case, we slide a one dimensional filter over a one dimensional input
- In the 2D case, we slide a two dimensional filter over a two dimensional output

| | | | |
|---|---|---|---|
| a | b | c | d |
| e | f | g | h |
| i | j | k | l |

Question

- In the 1D case, we slide a one dimensional filter over a one dimensional input
- In the 2D case, we slide a two dimensional filter over a two dimensional output

| | | | |
|---|---|---|---|
| a | b | c | d |
| e | f | g | h |
| i | j | k | l |

Question

- In the 1D case, we slide a one dimensional filter over a one dimensional input
- In the 2D case, we slide a two dimensional filter over a two dimensional output

| | | | |
|---|---|---|---|
| a | b | c | d |
| e | f | g | h |
| i | j | k | l |

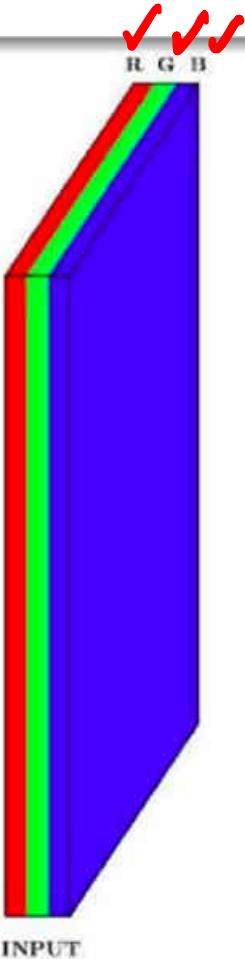
Question

- In the 1D case, we slide a one dimensional filter over a one dimensional input
- In the 2D case, we slide a two dimensional filter over a two dimensional output

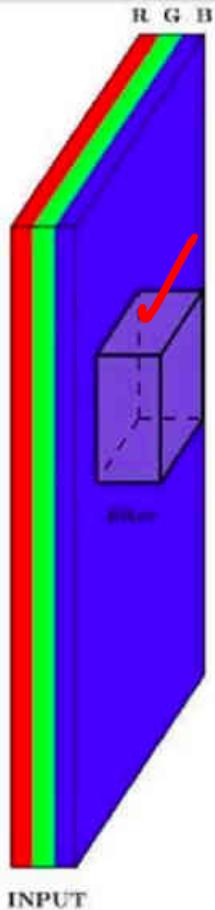
| | | | |
|---|---|---|---|
| a | b | c | d |
| e | f | g | h |
| i | j | k | l |

Question

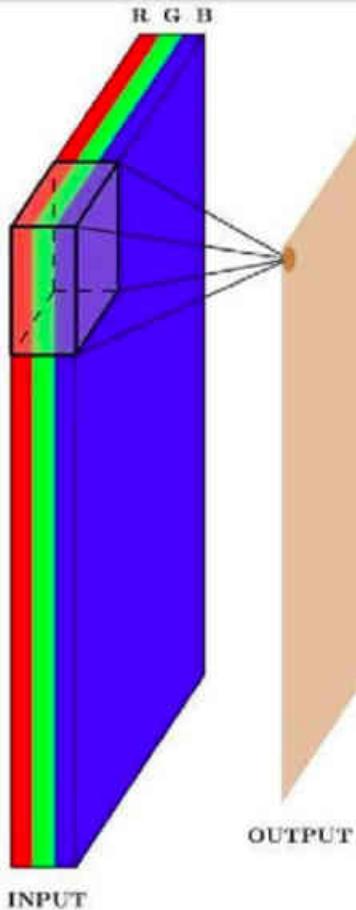
- In the 1D case, we slide a one dimensional filter over a one dimensional input
- In the 2D case, we slide a two dimensional filter over a two dimensional output
- What would happen in the 3D case?



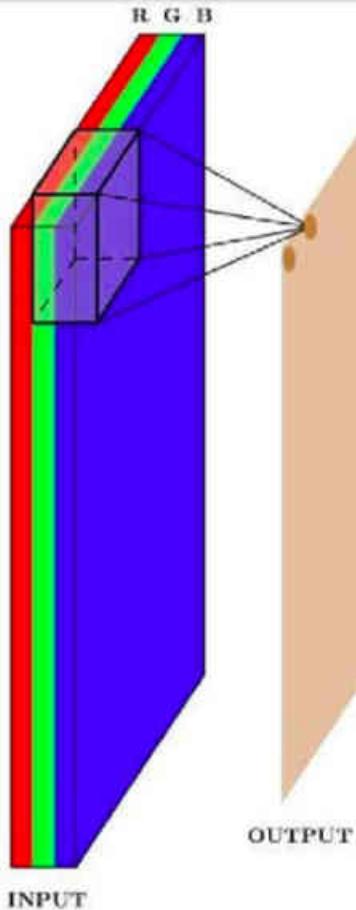
- What would a 3D filter look like?



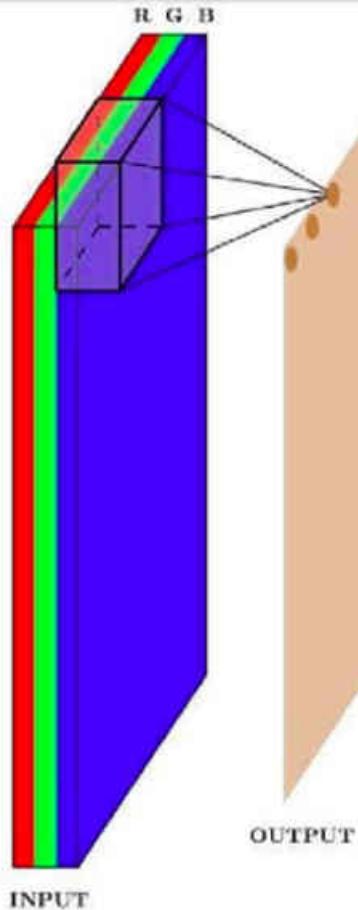
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume



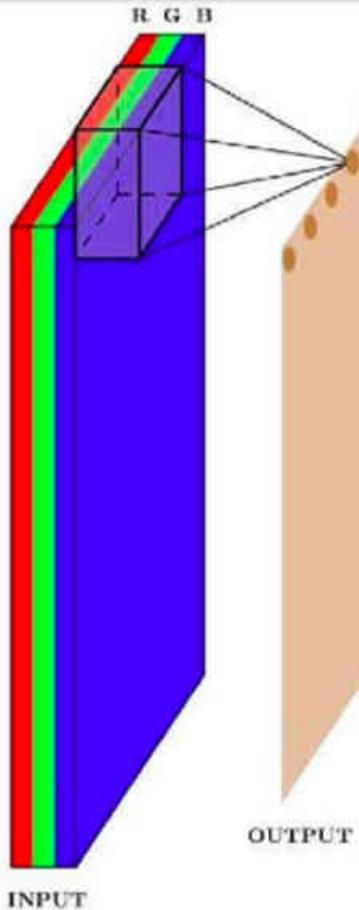
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



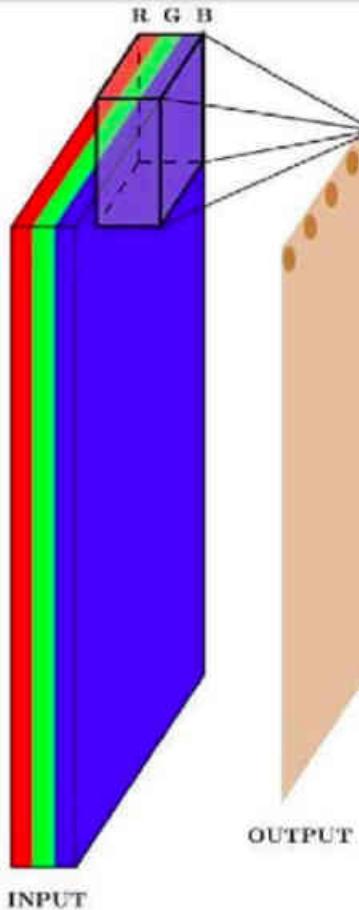
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



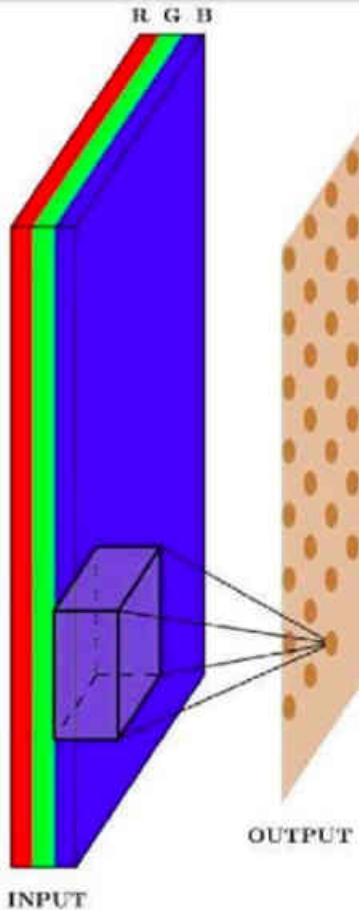
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



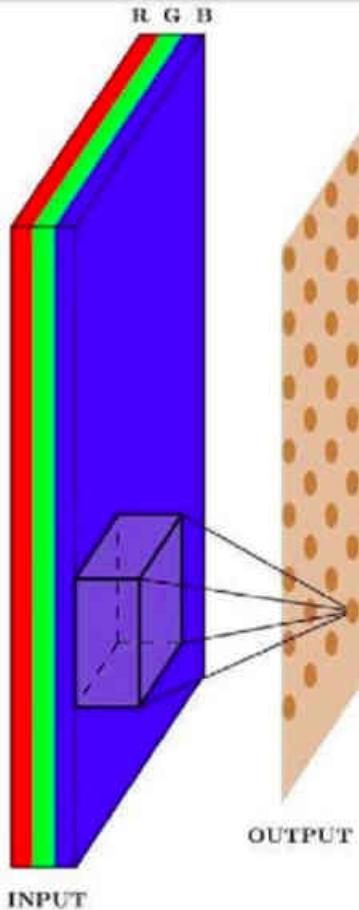
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



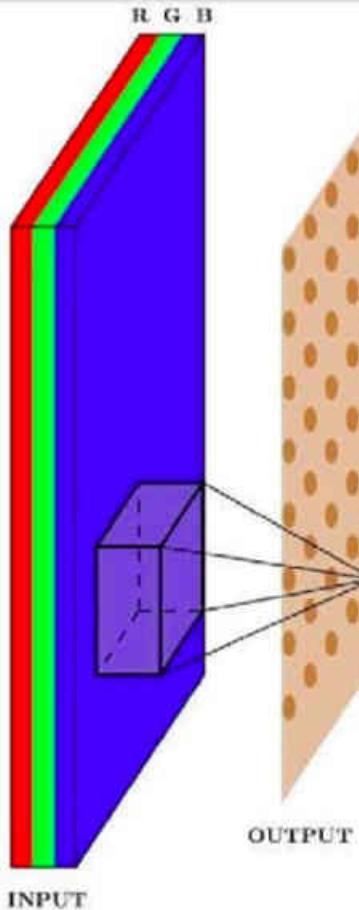
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



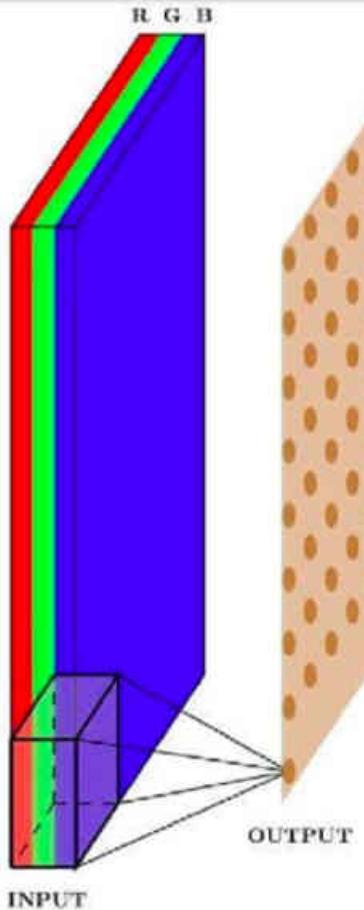
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



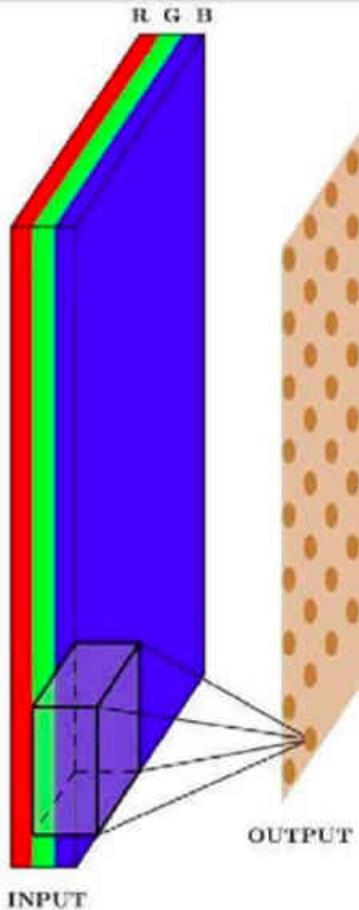
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



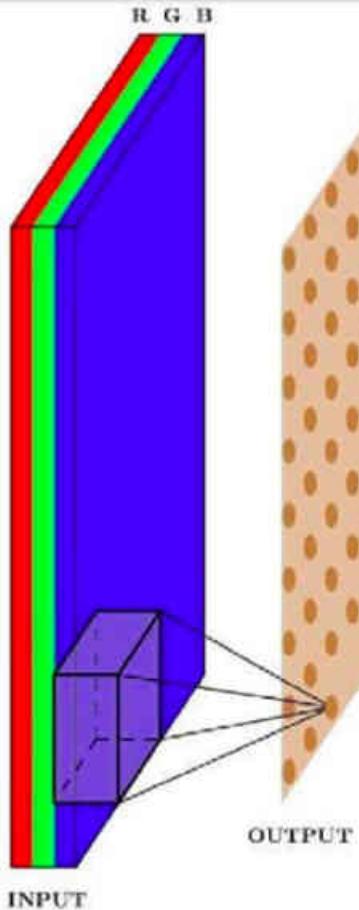
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



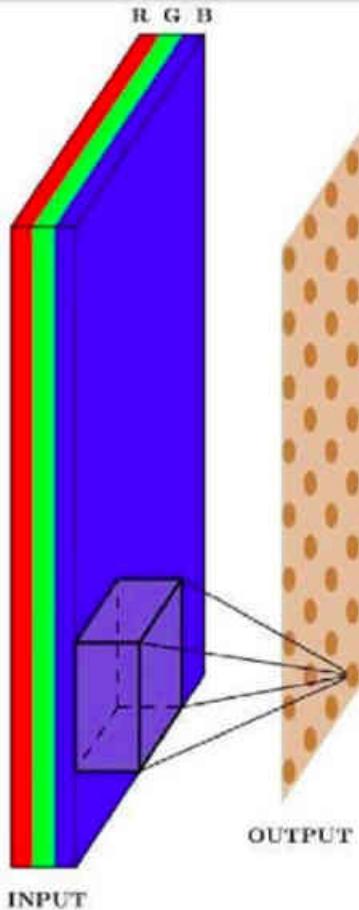
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



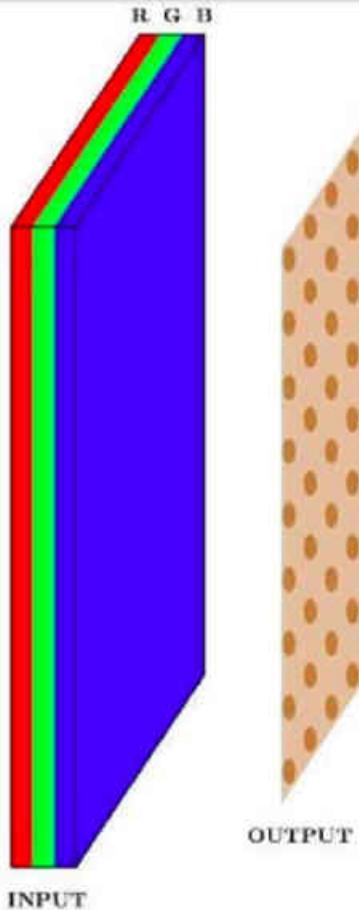
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



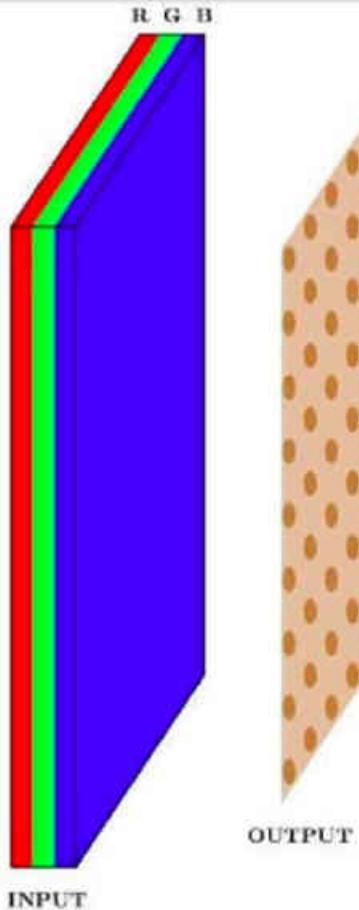
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



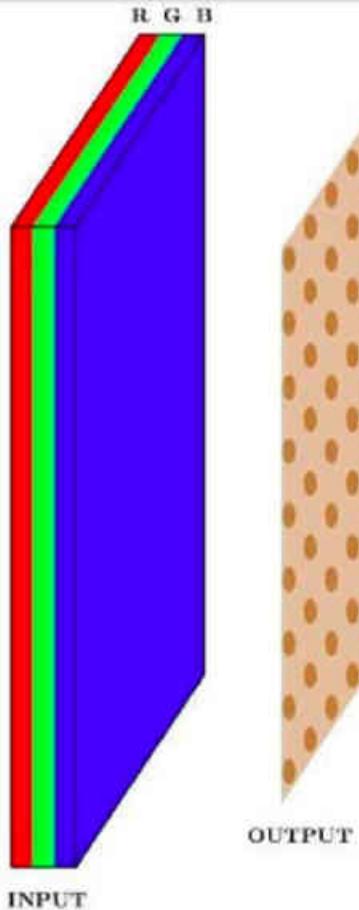
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation
- Note that in this lecture we will assume that the filter always extends to the depth of the image
- In effect, we are doing a 2D convolution operation on a 3D input (because the filter moves along the height and the width but not along the depth)

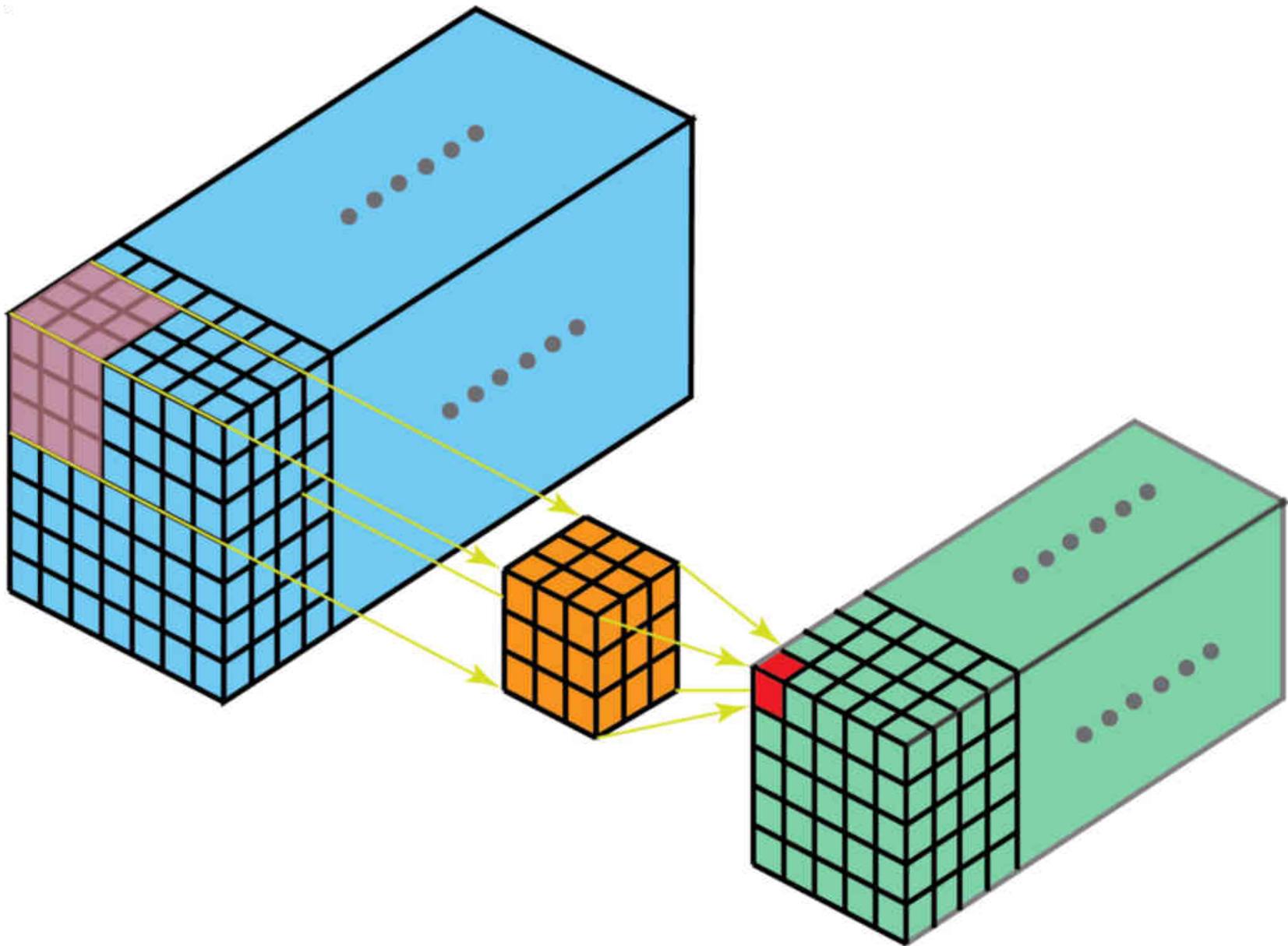


- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation
- Note that in this lecture we will assume that the filter always extends to the depth of the image
- In effect, we are doing a 2D convolution operation on a 3D input (because the filter moves along the height and the width but not along the depth)
- As a result the output will be 2D (only width and height, no depth)



- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation
- Note that in this lecture we will assume that the filter always extends to the depth of the image
- In effect, we are doing a 2D convolution operation on a 3D input (because the filter moves along the height and the width but not along the depth)
- As a result the output will be 2D (only width and height, no depth)
- Once again we can apply multiple filters to get multiple feature maps

CNN: Summarising the 3D case: convolution along depth



- So far we have not said anything explicit about the dimensions of the

- So far we have not said anything explicit about the dimensions of the
 - ➊ inputs

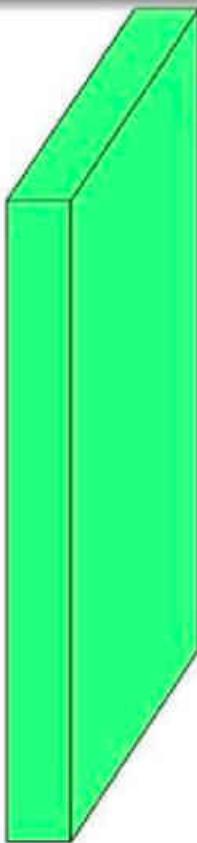
- So far we have not said anything explicit about the dimensions of the
 - ➊ inputs
 - ➋ filters

- So far we have not said anything explicit about the dimensions of the
 - ➊ inputs
 - ➋ filters
 - ➌ outputs

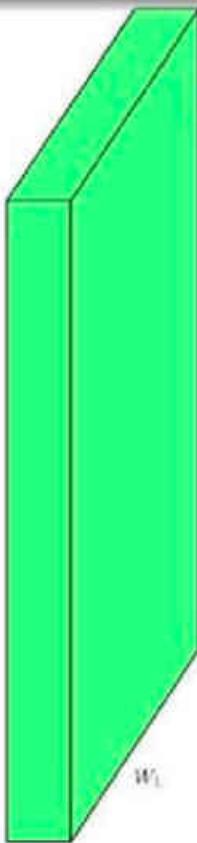
- So far we have not said anything explicit about the dimensions of the
 - ➊ inputs
 - ➋ filters
 - ➌ outputs

and the relations between them

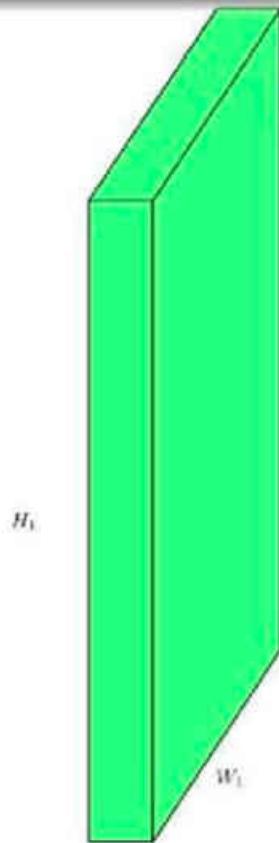
- So far we have not said anything explicit about the dimensions of the
 - ➊ inputs
 - ➋ filters
 - ➌ outputsand the relations between them
- We will see how they are related but before that we will define a few quantities



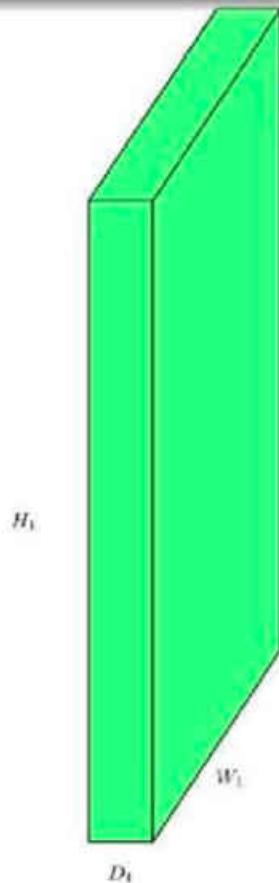
- We first define the following quantities



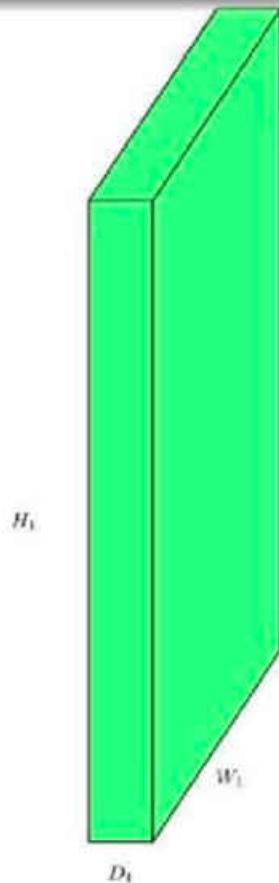
- We first define the following quantities
- Width (W_1),



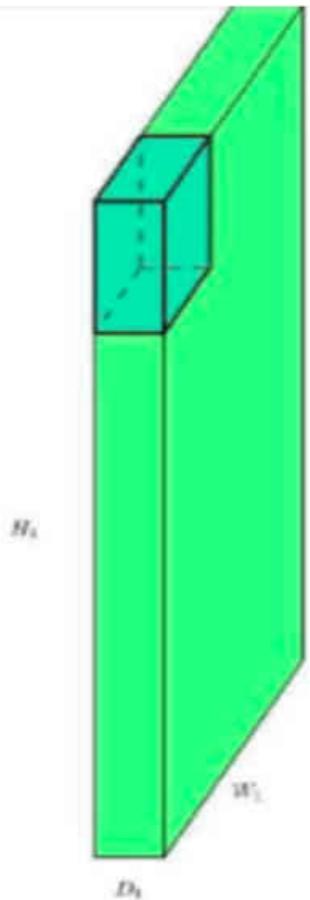
- We first define the following quantities
- Width (W_1), Height (H_1)



- We first define the following quantities
- Width (W_1), Height (H_1) and Depth (D_1) of the original input

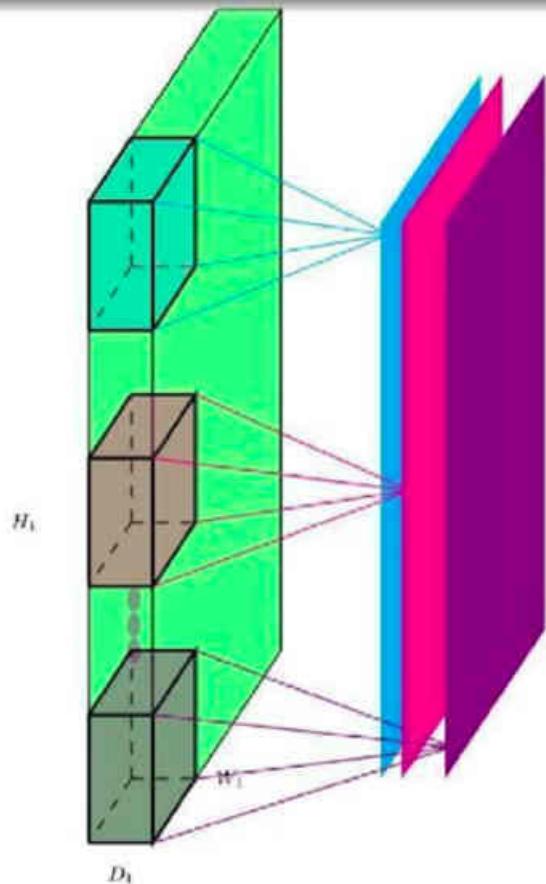


- We first define the following quantities
- Width (W_1), Height (H_1) and Depth (D_1) of the original input
- The Stride S (We will come back to this later)

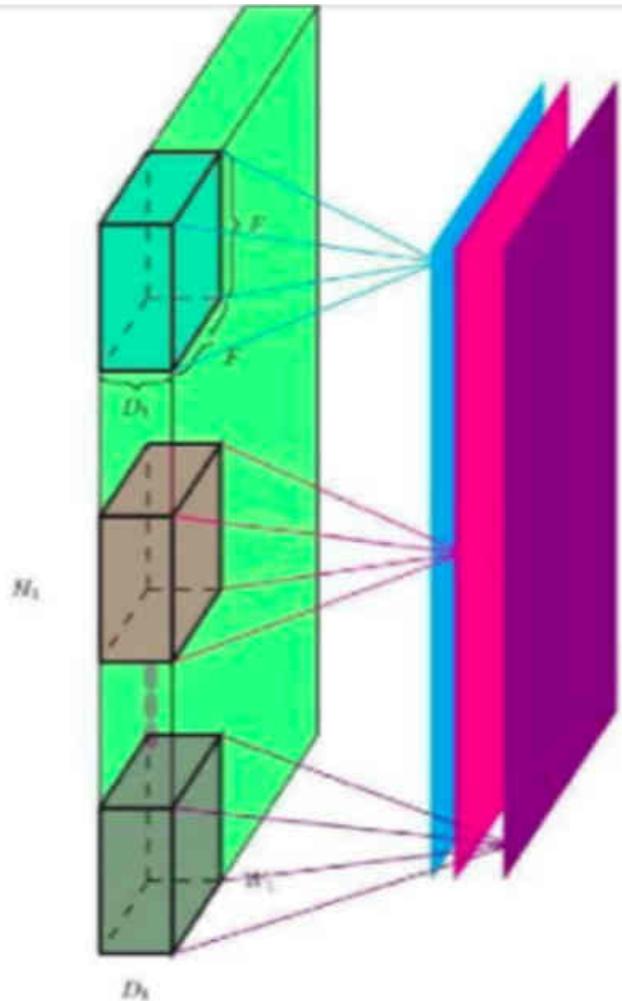


- We first define the following quantities
- Width (W_1), Height (H_1) and Depth (D_1) of the original input
- The Stride S (We will come back to this later)

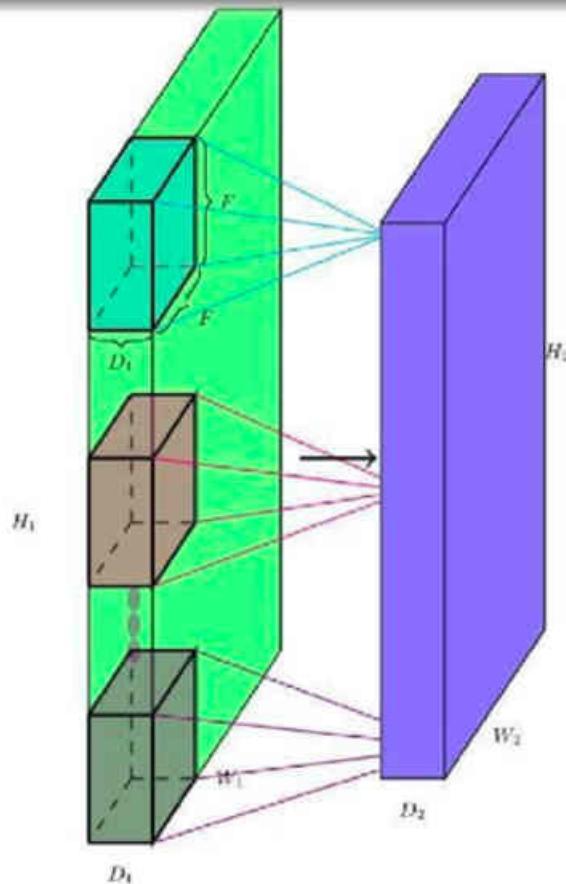
Remember that almost all filters are square. This is called spatial dimension of filters and is denoted as 'f'.



- We first define the following quantities
- Width (W_1), Height (H_1) and Depth (D_1) of the original input
- The Stride S (We will come back to this later)
- The number of filters K

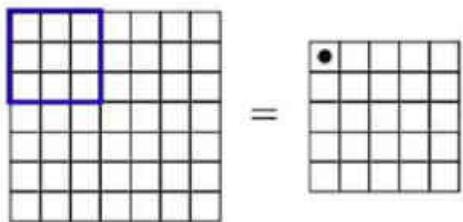


- We first define the following quantities
- Width (W_1), Height (H_1) and Depth (D_1) of the original input
- The Stride S (We will come back to this later)
- The number of filters K
- The spatial extent (F) of each filter (the depth of each filter is same as the depth of each input)

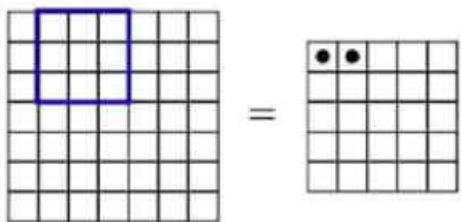


- We first define the following quantities
- Width (W_1), Height (H_1) and Depth (D_1) of the original input
- The Stride S (We will come back to this later)
- The number of filters K
- The spatial extent (F) of each filter (the depth of each filter is same as the depth of each input)
- The output is $W_2 \times H_2 \times D_2$ (we will soon see a formula for computing W_2 , H_2 and D_2)

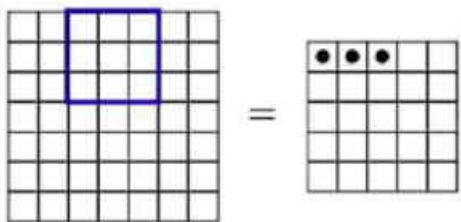
- Let us compute the dimension (W_2, H_2) of the output



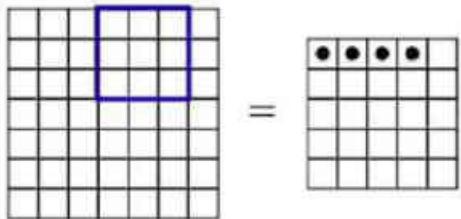
- Let us compute the dimension (W_2, H_2) of the output



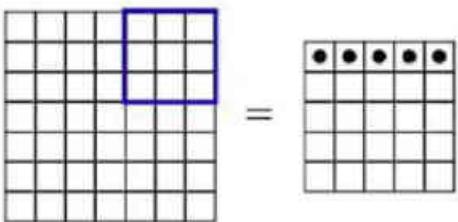
- Let us compute the dimension (W_2, H_2) of the output



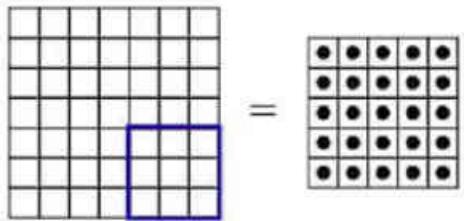
- Let us compute the dimension (W_2, H_2) of the output



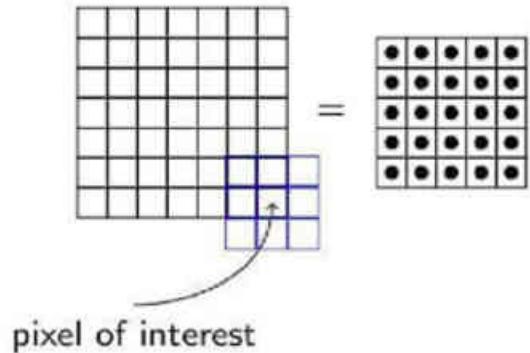
- Let us compute the dimension (W_2, H_2) of the output



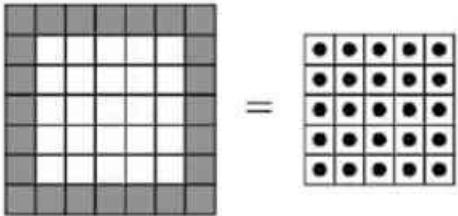
- Let us compute the dimension (W_2, H_2) of the output



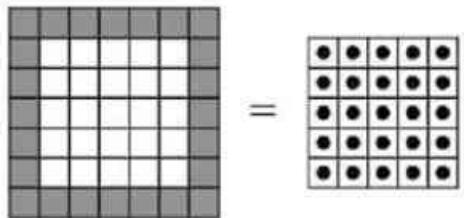
- Let us compute the dimension (W_2, H_2) of the output



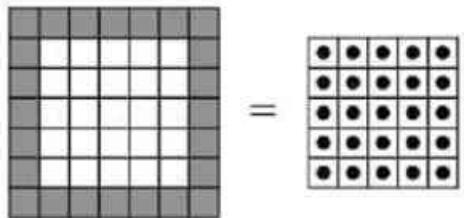
- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary



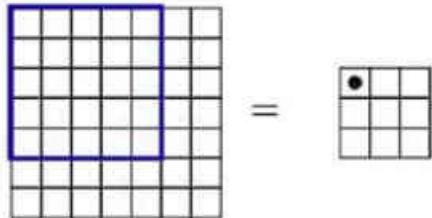
- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)



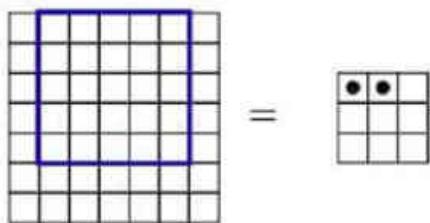
- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input



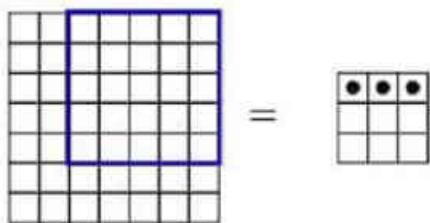
- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input
- As the size of the kernel increases, this becomes true for even more pixels



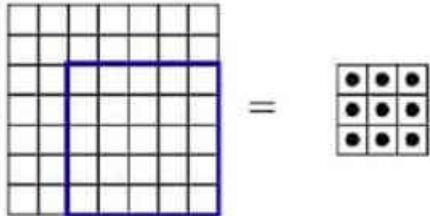
- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input
- As the size of the kernel increases, this becomes true for even more pixels
- For example, let's consider a 5×5 kernel



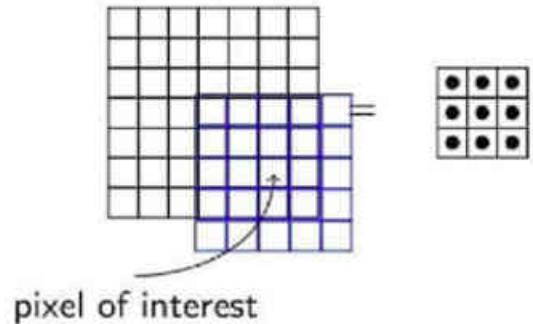
- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input
- As the size of the kernel increases, this becomes true for even more pixels
- For example, let's consider a 5×5 kernel
- We have an even smaller output now



- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input
- As the size of the kernel increases, this becomes true for even more pixels
- For example, let's consider a 5×5 kernel
- We have an even smaller output now

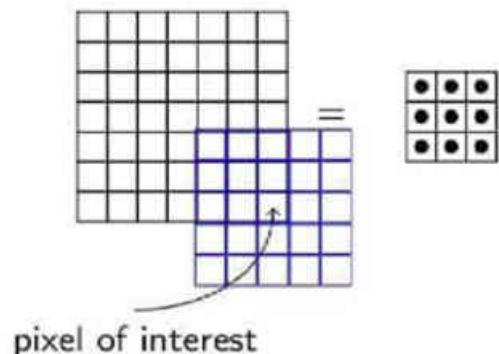


- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input
- As the size of the kernel increases, this becomes true for even more pixels
- For example, let's consider a 5×5 kernel
- We have an even smaller output now



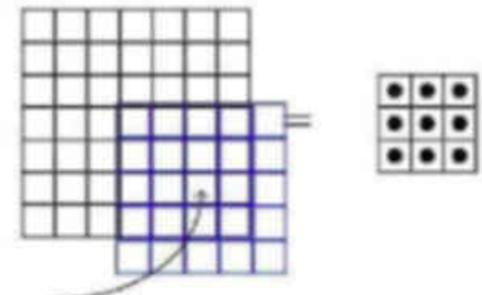
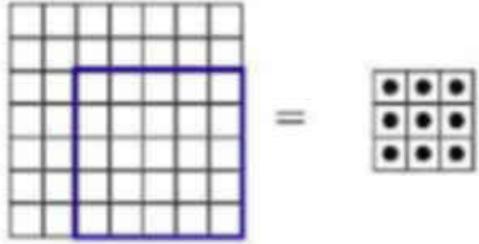
pixel of interest

- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input
- As the size of the kernel increases, this becomes true for even more pixels
- For example, let's consider a 5×5 kernel
- We have an even smaller output now

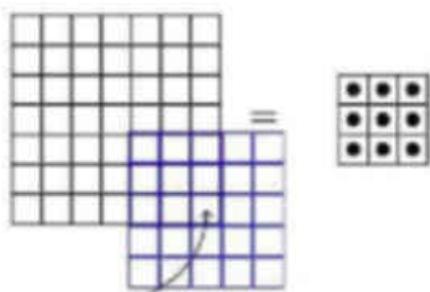


pixel of interest

- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input
- As the size of the kernel increases, this becomes true for even more pixels
- For example, let's consider a 5×5 kernel
- We have an even smaller output now



pixel of interest



pixel of interest

- Once the kernel traverses through all the pixels, some parts of the kernel tends to extend outside the image leading to the loss of information
- We have a problem all around the edge
 - We have a few choices here
 - We can ignore these pixels
 - We would of course lose information in these areas
 - Or what else can **we do is padding**

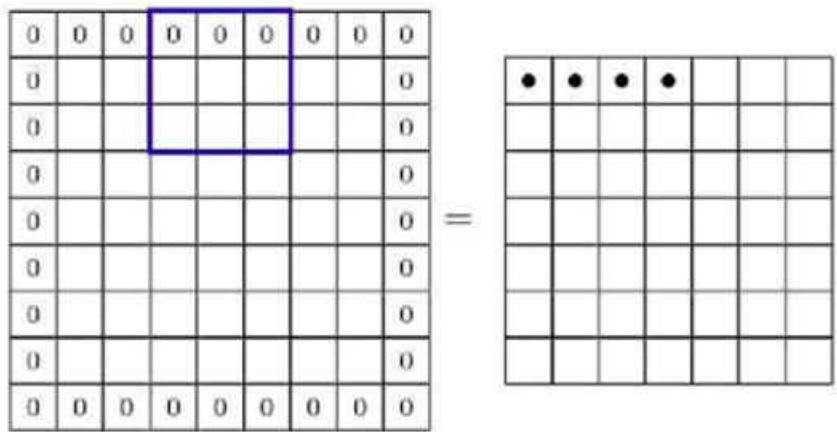
-
- What if we want the output to be of same size as the input?
 - We can use something known as padding

- What if we want the output to be of same size as the input?
 - We can use something known as padding
 - Pad the inputs with appropriate number of 0 inputs so that you can now apply the kernel at the corners
 - Let us use pad $P = 1$ with a 3×3 kernel

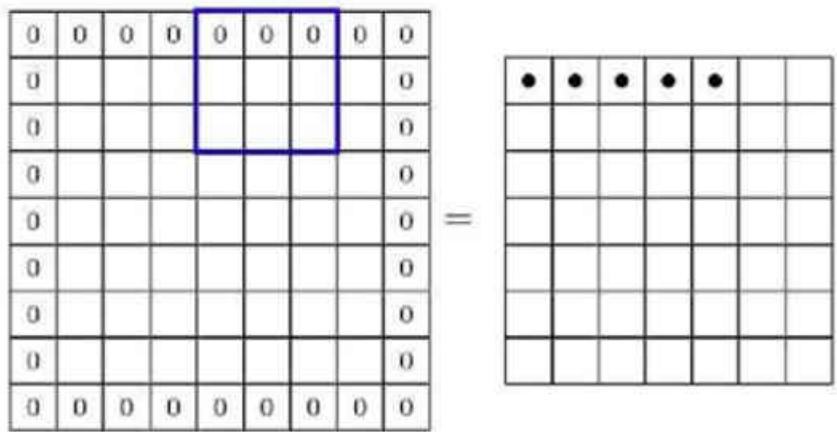
- What if we want the output to be of same size as the input?
 - We can use something known as padding
 - Pad the inputs with appropriate number of 0 inputs so that you can now apply the kernel at the corners
 - Let us use pad $P = 1$ with a 3×3 kernel
 - This means we will add one row and one column of 0 inputs at the top, bottom, left and right

$$\begin{array}{|c|c|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & & & & & & & & & 0 \\ \hline 0 & & & & & & & & & 0 \\ \hline 0 & & & & & & & & & 0 \\ \hline 0 & & & & & & & & & 0 \\ \hline 0 & & & & & & & & & 0 \\ \hline 0 & & & & & & & & & 0 \\ \hline 0 & & & & & & & & & 0 \\ \hline 0 & & & & & & & & & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array} = \begin{array}{|c|c|c|c|c|c|c|c|c|c|} \hline \bullet & \bullet & \bullet & & & & & & & \\ \hline & & & & & & & & & \\ \hline & & & & & & & & & \\ \hline & & & & & & & & & \\ \hline & & & & & & & & & \\ \hline & & & & & & & & & \\ \hline & & & & & & & & & \\ \hline & & & & & & & & & \\ \hline & & & & & & & & & \\ \hline \end{array}$$

- What if we want the output to be of same size as the input?
- We can use something known as padding
- Pad the inputs with appropriate number of 0 inputs so that you can now apply the kernel at the corners
- Let us use pad $P = 1$ with a 3×3 kernel
- This means we will add one row and one column of 0 inputs at the top, bottom, left and right

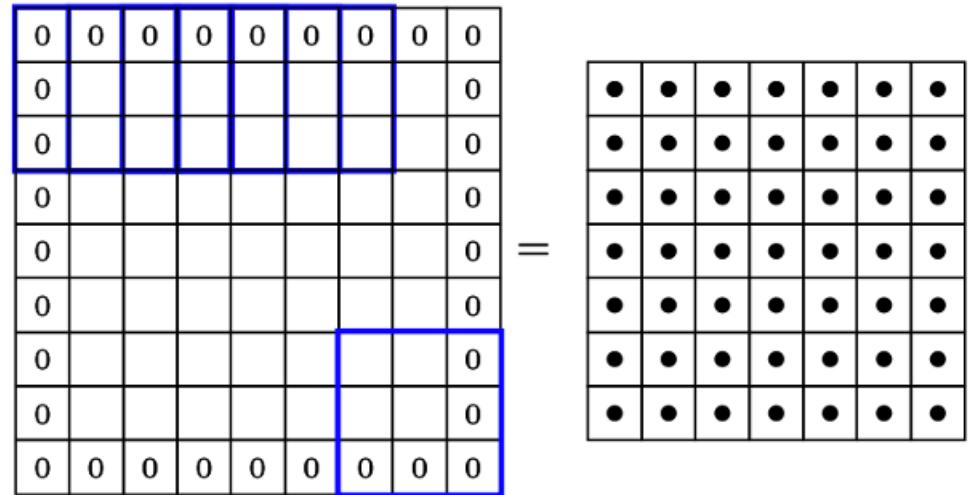


- What if we want the output to be of same size as the input?
- We can use something known as padding
- Pad the inputs with appropriate number of 0 inputs so that you can now apply the kernel at the corners
- Let us use pad $P = 1$ with a 3×3 kernel
- This means we will add one row and one column of 0 inputs at the top, bottom, left and right



A diagram illustrating a convolution operation. On the left, there is a 10x10 input matrix with values 0 throughout. A 3x3 kernel is applied to the input, highlighted by a blue border. The result of the convolution is shown on the right, which is a 7x7 matrix with a single dot at its center.

- What if we want the output to be of same size as the input?
- We can use something known as padding
- Pad the inputs with appropriate number of 0 inputs so that you can now apply the kernel at the corners
- Let us use pad $P = 1$ with a 3×3 kernel
- This means we will add one row and one column of 0 inputs at the top, bottom, left and right



- What if we want the output to be of same size as the input?
- We can use something known as padding
- Pad the inputs with appropriate number of 0 inputs so that you can now apply the kernel at the corners
- Let us use pad $P = 1$ with a 3×3 kernel
- This means we will add one row and one column of 0 inputs at the top, bottom, left and right

We now have,

$$W_2 = W_1 - F + 2P + 1$$

$$H_2 = H_1 - F + 2P + 1$$

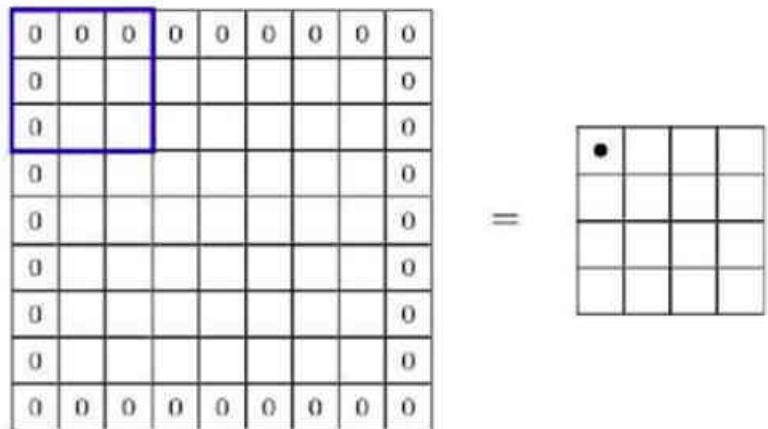
We will refine this formula further

$$W_2 = W_1 - F + 2P + 1$$

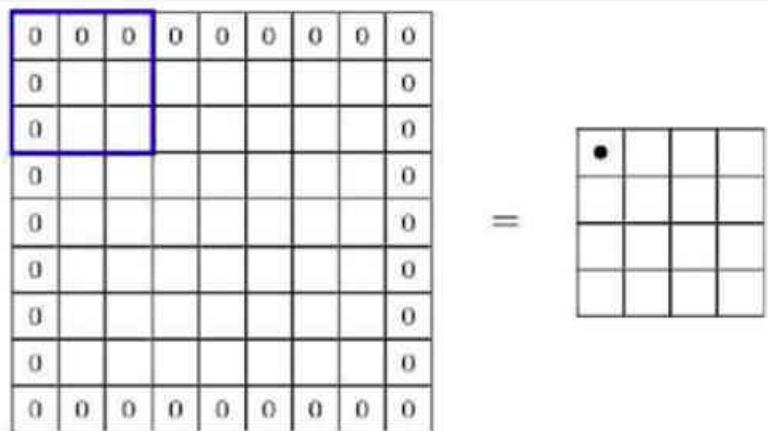
$$\text{Since } W_2 = W_1$$

- $F = 2P + 1$
- $F - 1 = 2P$
- $P = (F - 1) / 2$

-
- What does the stride S do?
 - It defines the intervals at which the filter is applied (here $S = 2$)



- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



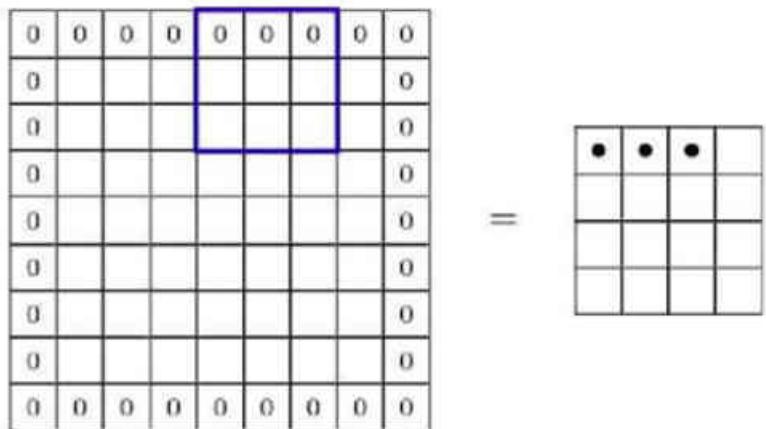
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions

We can see that the step size (stride) is 2

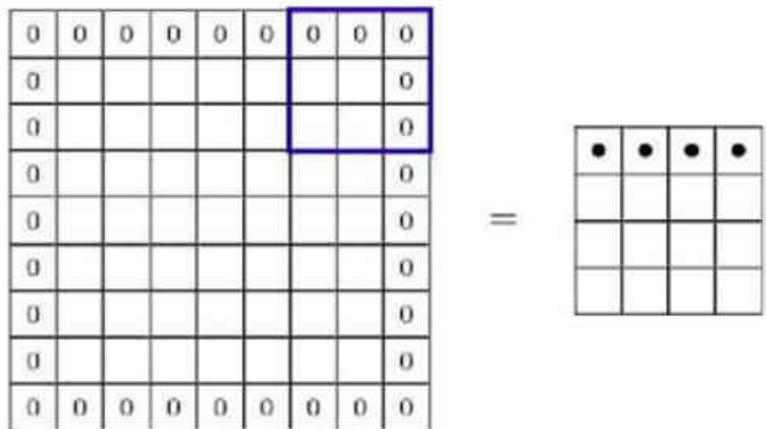
So given the input dimensions, stride and the padding what is the output size:

$$O = \frac{I - F + 2P}{S} + 1$$

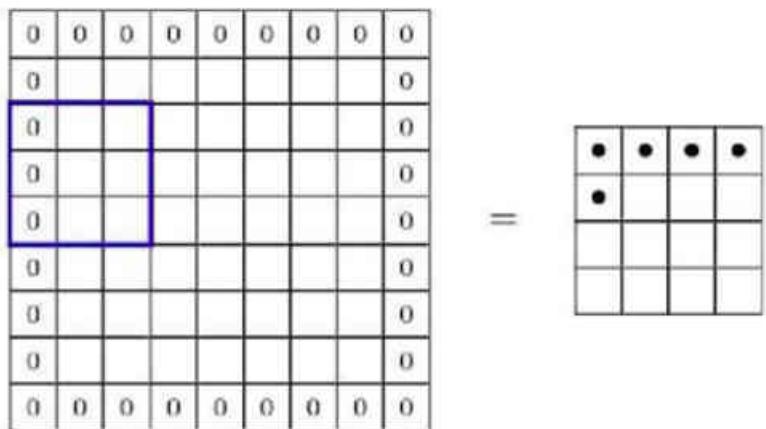
The one is outside due to floor and ceiling reasons



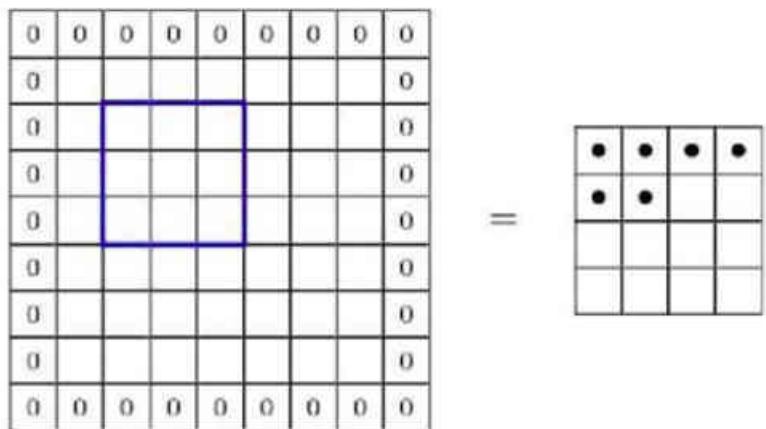
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



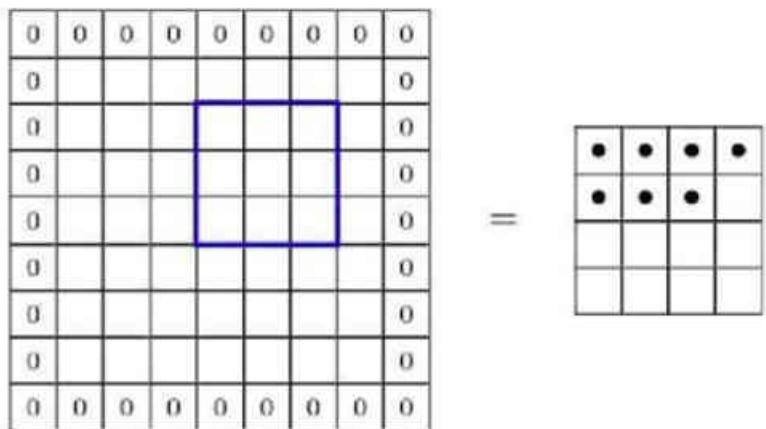
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



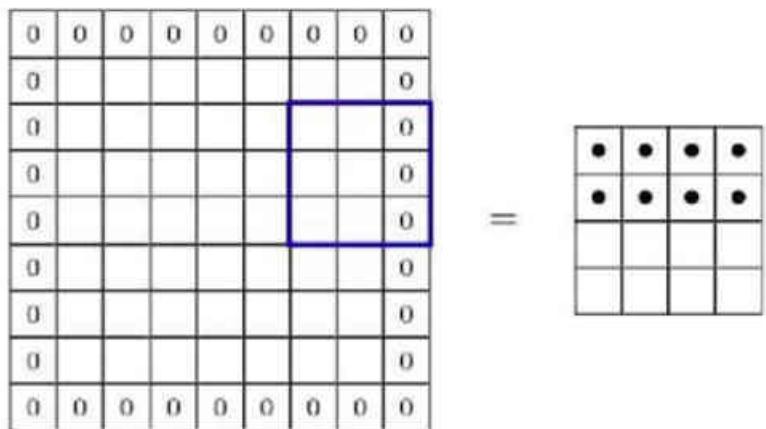
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



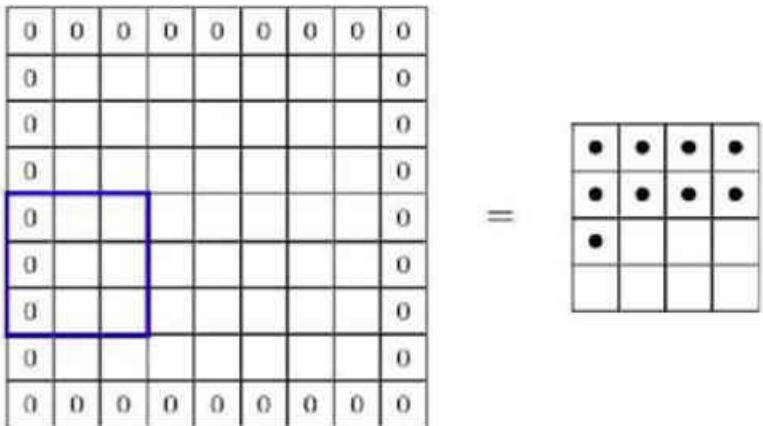
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



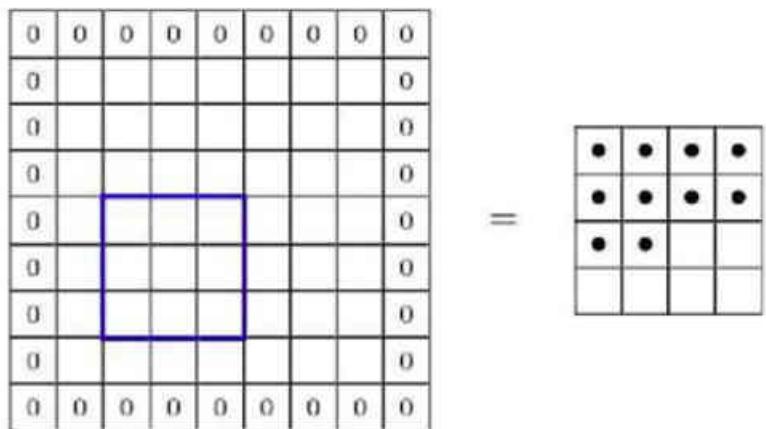
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



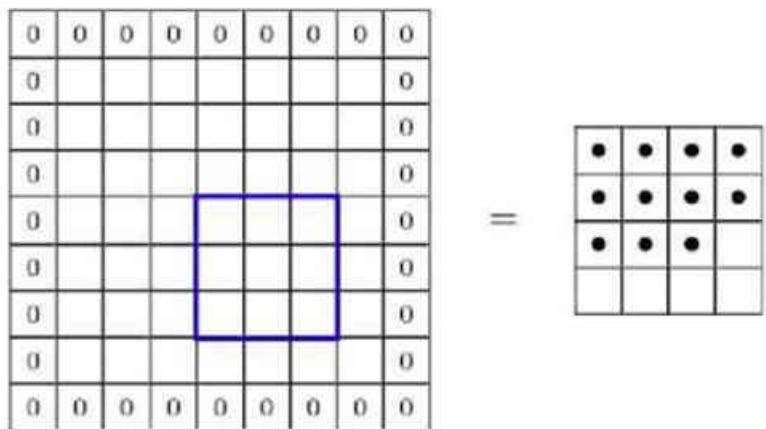
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



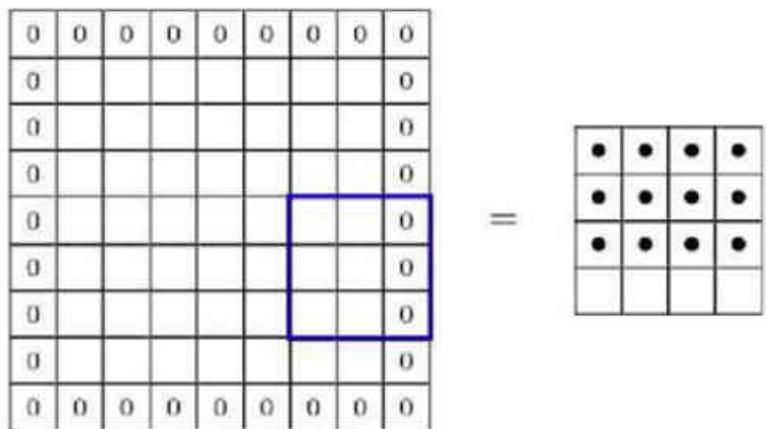
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



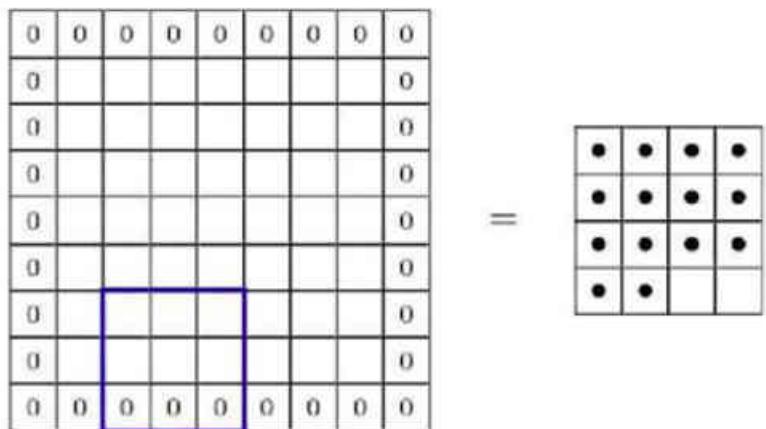
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & & & & & & & & 0 \\ 0 & & & & & & & & 0 \\ 0 & & & & & & & & 0 \\ 0 & & & & & & & & 0 \\ 0 & & & & & & & & 0 \\ 0 & & & & & & & & 0 \\ 0 & & & & & & & & 0 \\ 0 & & & & & & & & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix} = \begin{matrix} \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & & & \end{matrix}$$

- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller 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 |

$$= \begin{array}{|c|c|c|c|} \hline \bullet & \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet & \bullet \\ \hline \end{array}$$

- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions

So what should our final formula look like,

$$\checkmark W_2 = \frac{W_1 - F + 2P}{S} + 1 \quad \checkmark$$

$$\checkmark H_2 = \frac{H_1 - F + 2P}{S} + 1 \quad \checkmark$$

References

1. [MIT 6.S191 \(2020\): Convolutional Neural Networks - YouTube](#)
1. [Deep Learning\(CS7015\): Lec 11.3 Convolutional Neural Networks - YouTube- Mitesh M Khapra](#)
1. [Convolutional Neural Network - YouTube- Ahlad Kumar](#)
1. [Lecture Collection | Convolutional Neural Networks for Visual Recognition \(Spring 2017\) - YouTube](#)
1. [CNNs, Part 1: An Introduction to Convolutional Neural Networks - victorzhou.com](#)



Thank You

Dr. Arti Arya
artiarya@pes.edu



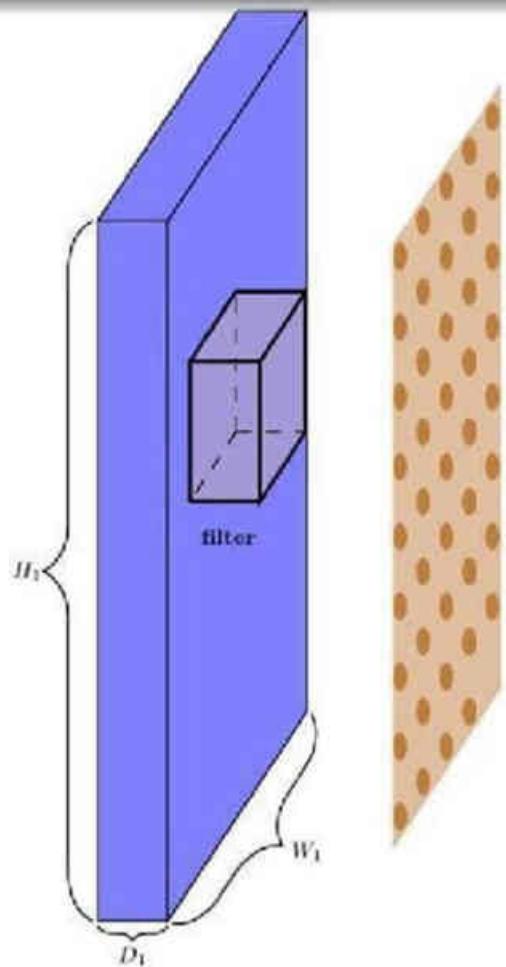
MACHINE INTELLIGENCE

Dimensions and Parameter Calculations in CNNs

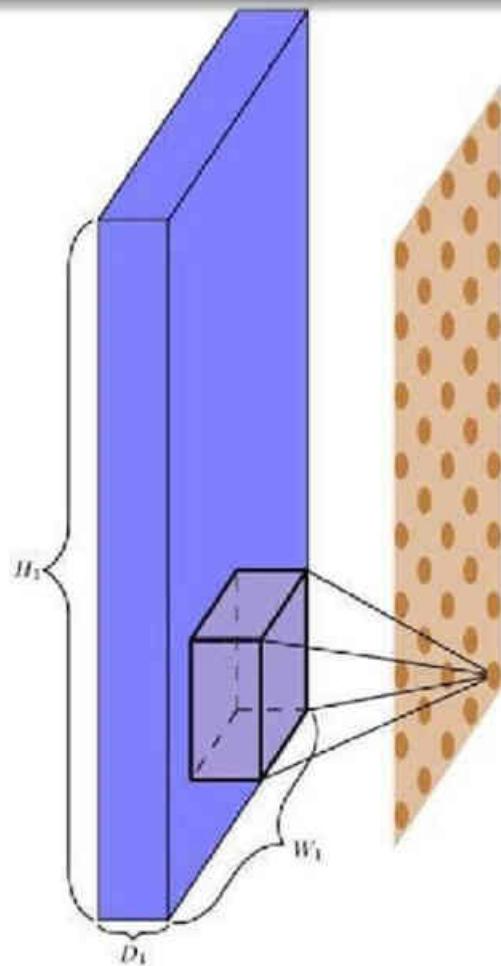
Dr. Arti Arya

Disclaimer

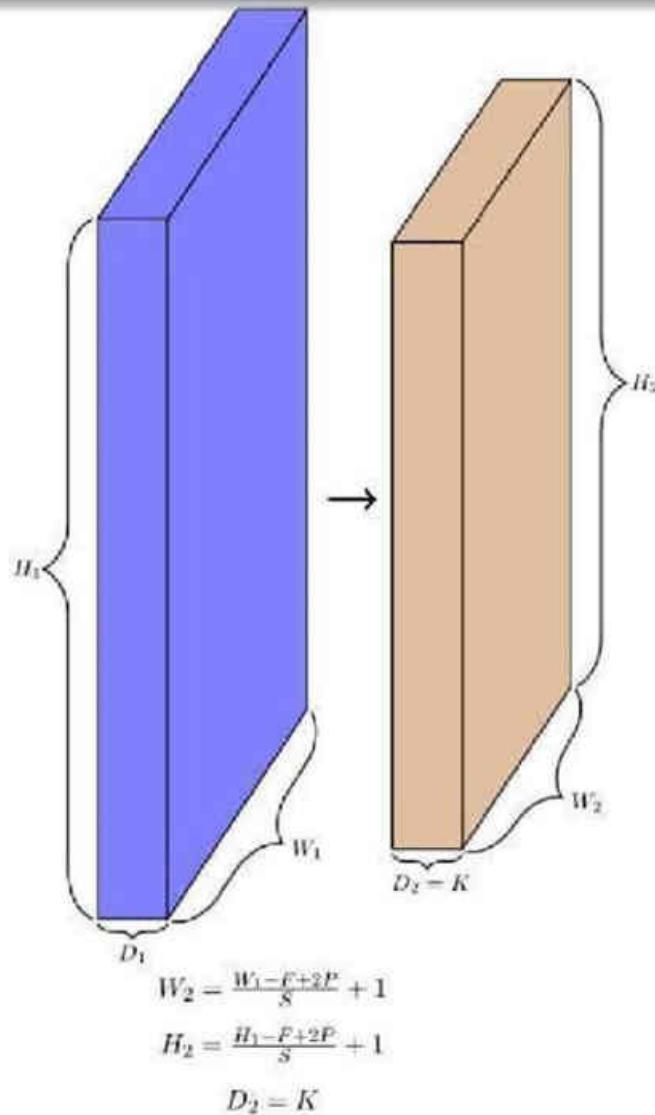
These slides are prepared from various resources from Internet and universities from India and Abroad. Broadly adapted from slides by Prof. Mitesh M Khapra from IITM.



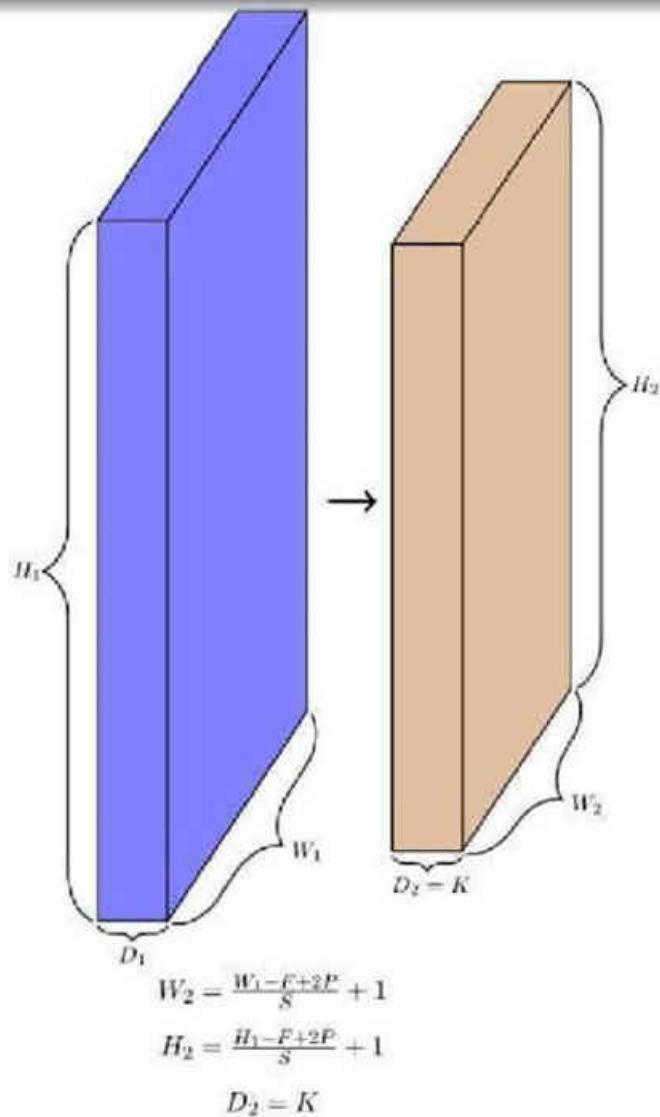
- Finally, coming to the depth of the output.



- Finally, coming to the depth of the output.
- Each filter gives us one 2D output.



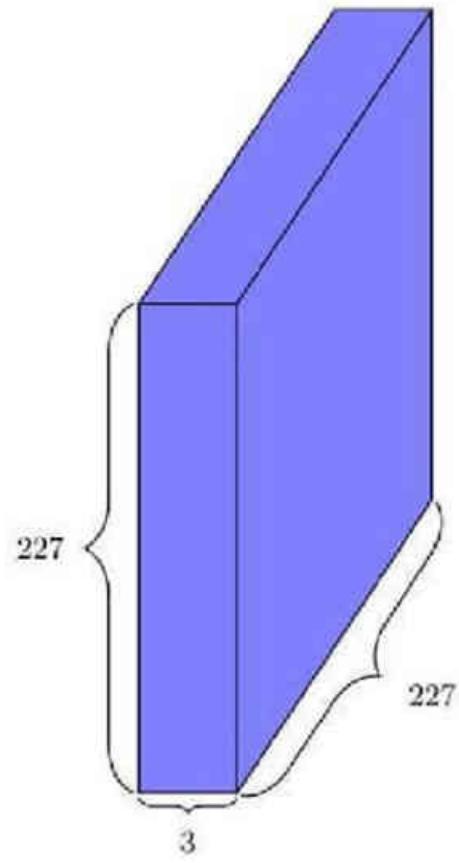
- Finally, coming to the depth of the output.
- Each filter gives us one 2D output.
- K filters will give us K such 2D outputs



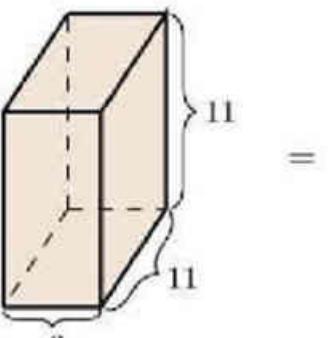
- Finally, coming to the depth of the output.
- Each filter gives us one 2D output.
- K filters will give us K such 2D outputs
- We can think of the resulting output as $K \times W_2 \times H_2$ volume
- Thus $D_2 = K$

Example 1

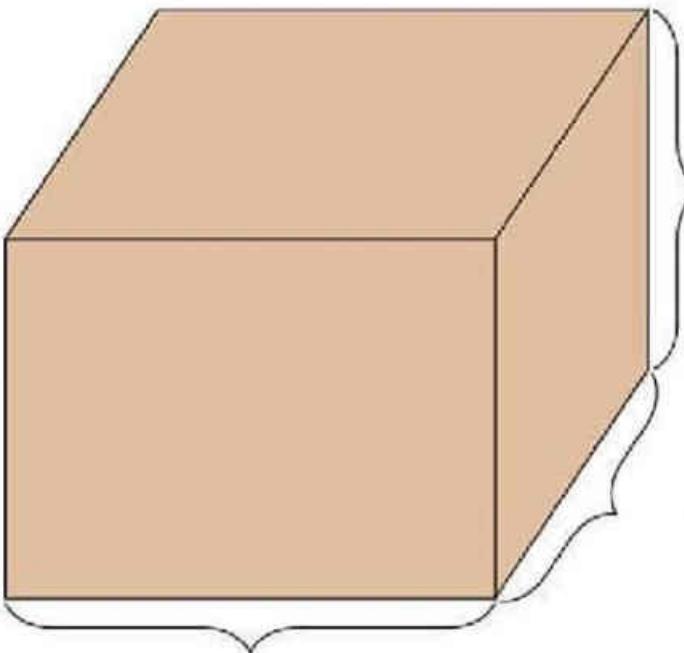
Let us do a few exercises



*



=



227

227

3

96 filters

Stride = 4

Padding = 0

$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$

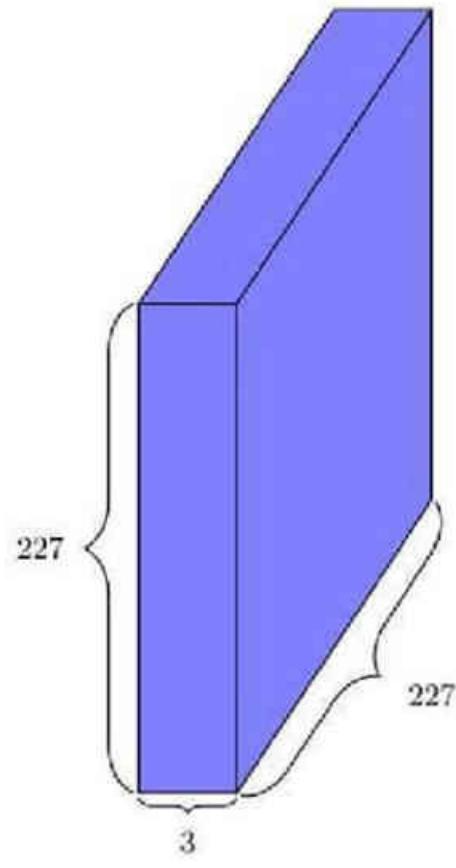
$$H_2 = ?$$

$$W_2 = ?$$

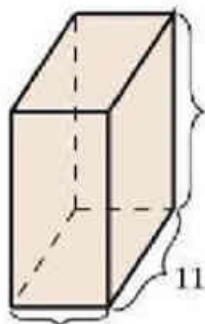
$$D_2 = ?$$

Example 1

Let us do a few exercises



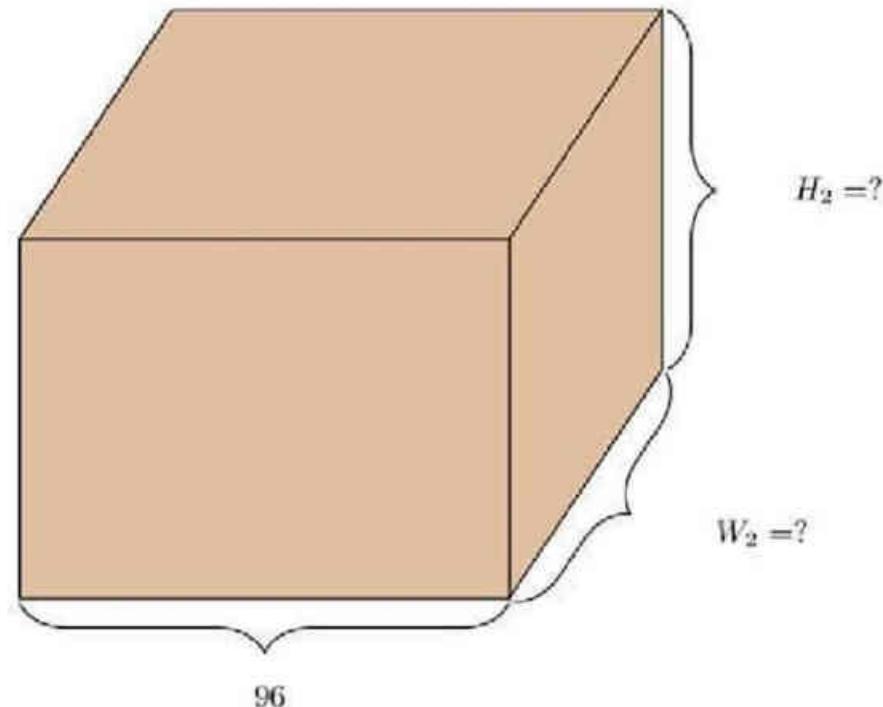
*



A diagram of a convolutional layer kernel. It is a 3D cube with a height of 11, a width of 11, and a depth of 3. The depth dimension is labeled "3". The height dimension is labeled "11". The width dimension is labeled "11".

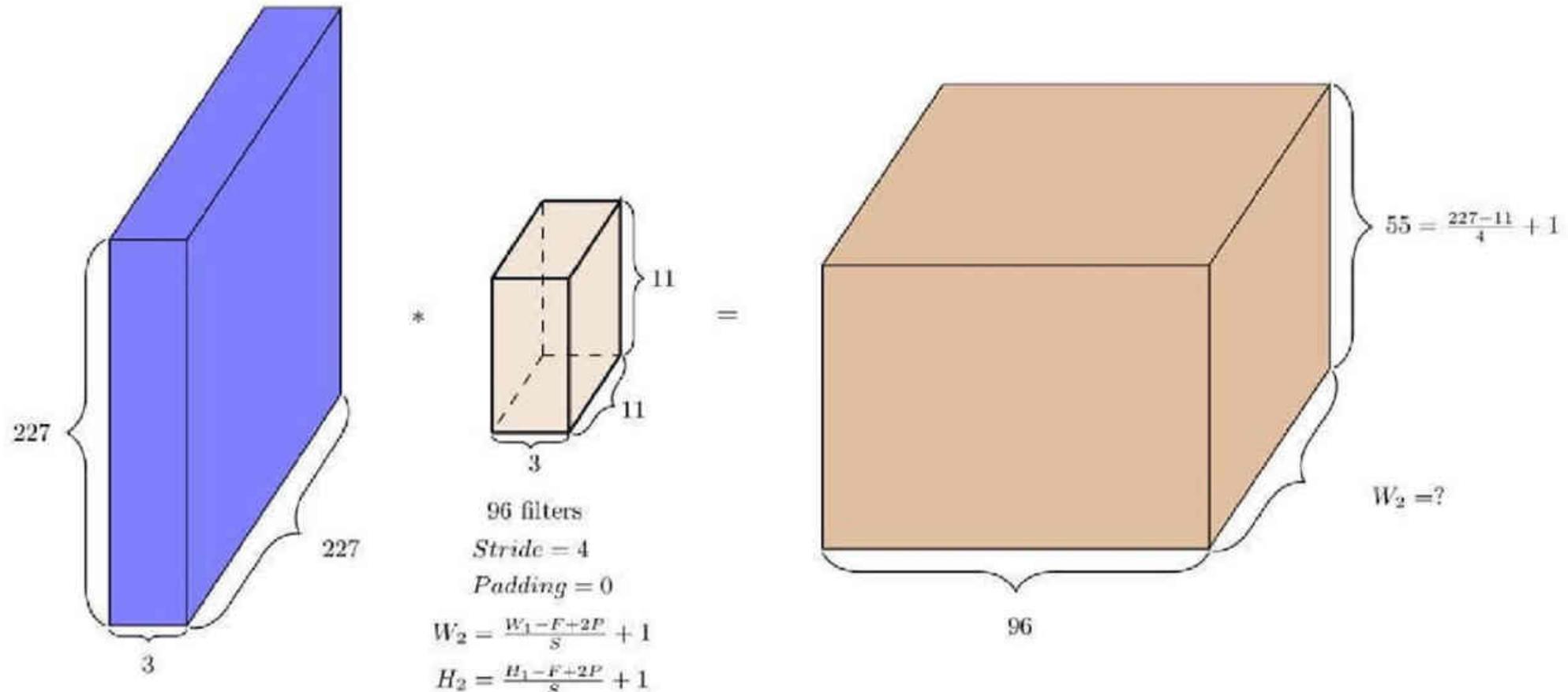
96 filters
Stride = 4
Padding = 0

$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$
$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$



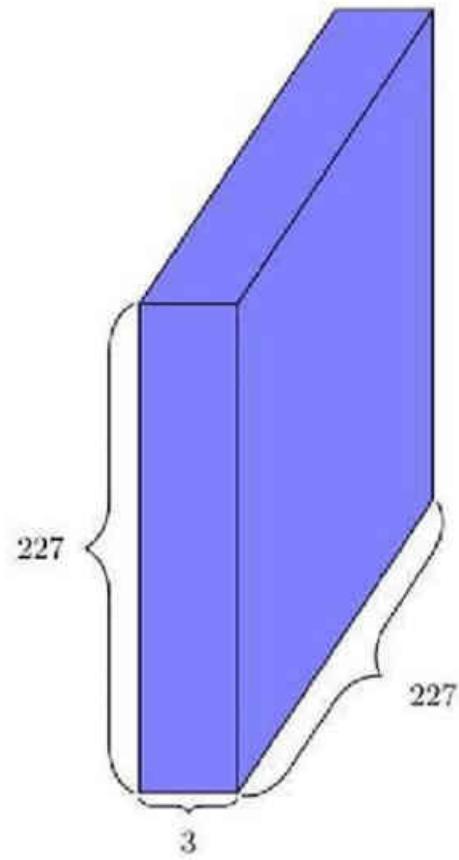
Example 1

Let us do a few exercises

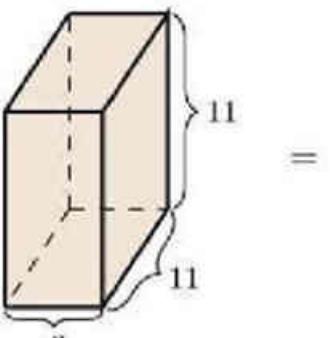


Example 1

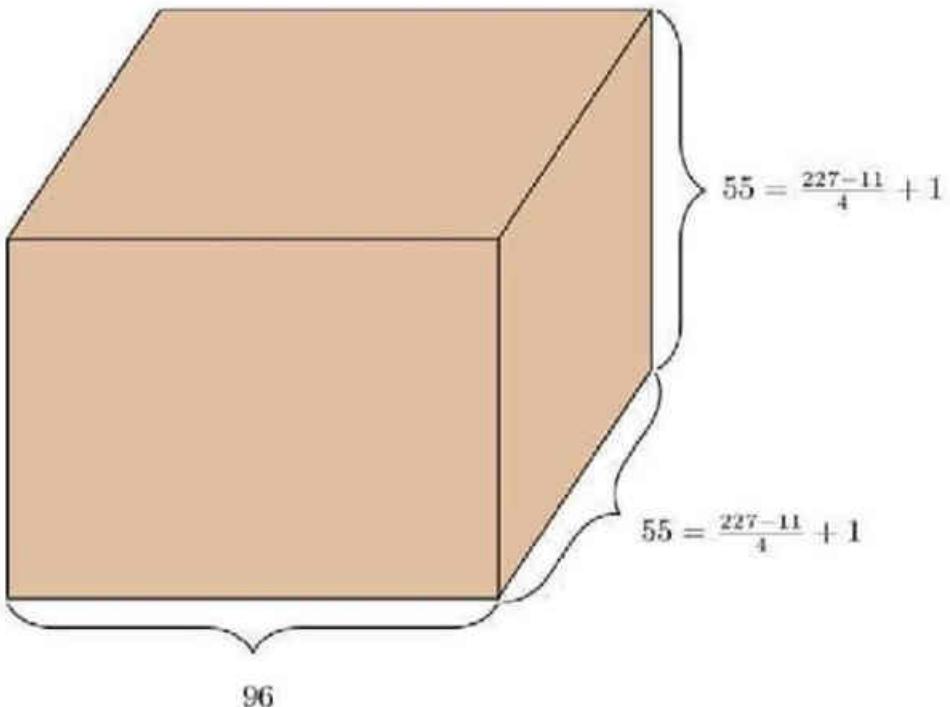
Let us do a few exercises



*



=



227

227

3

96 filters

Stride = 4

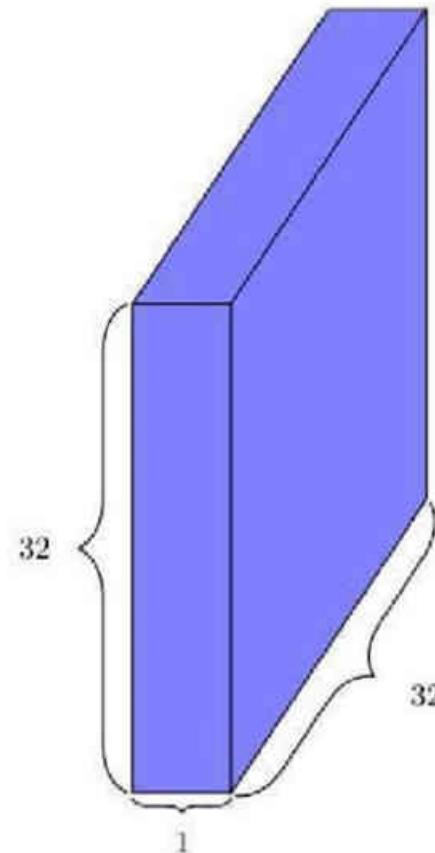
Padding = 0

$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

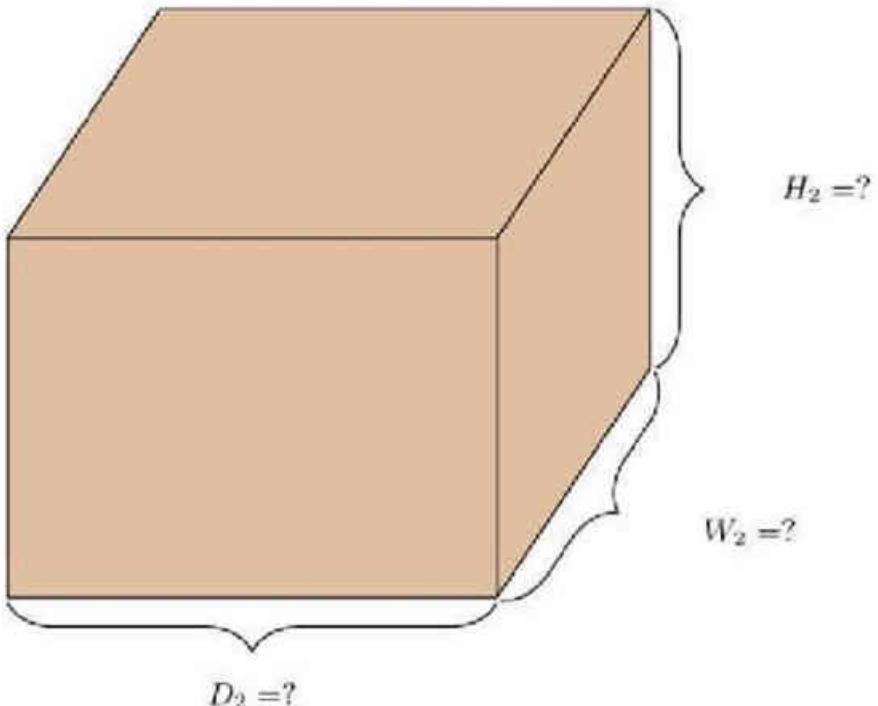
$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$

Example 2

Let us do a few exercises

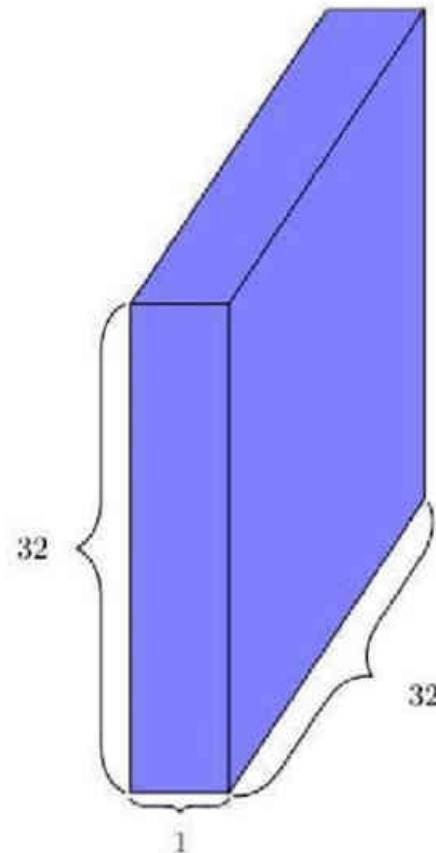


$$* \quad \begin{array}{c} \text{Input tensor: } H_1 = 32, W_1 = 32, D_1 = 1 \\ \text{Filter size: } F = 5 \\ \text{Stride: } S = 1 \\ \text{Padding: } P = 0 \\ W_2 = \frac{W_1 - F + 2P}{S} + 1 \\ H_2 = \frac{H_1 - F + 2P}{S} + 1 \end{array}$$



Example 2

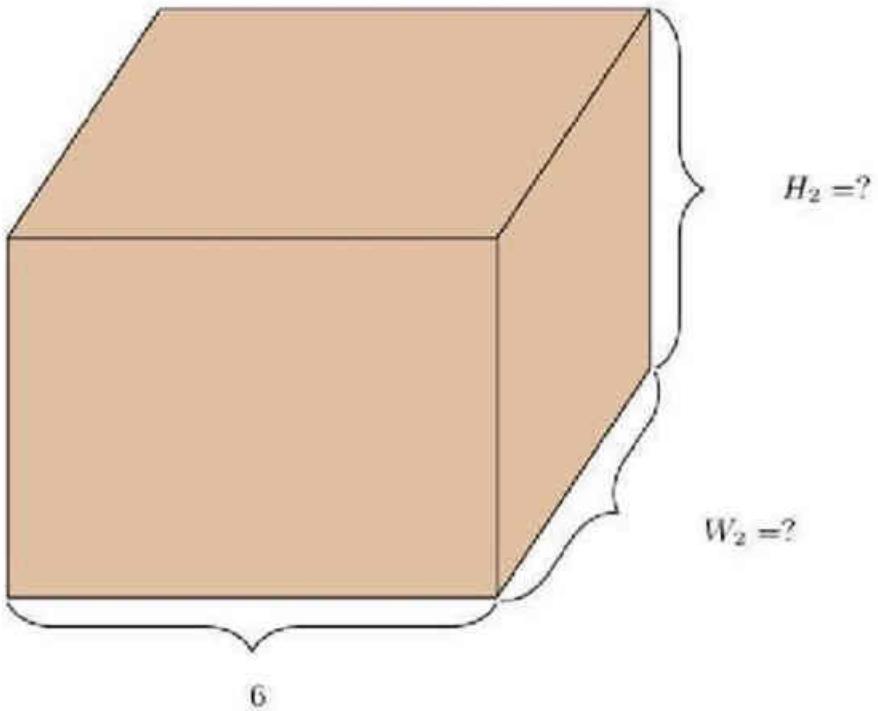
Let us do a few exercises



$$\begin{matrix} * \\ \text{6 filters} \\ \text{Stride} = 1 \\ \text{Padding} = 0 \end{matrix}$$

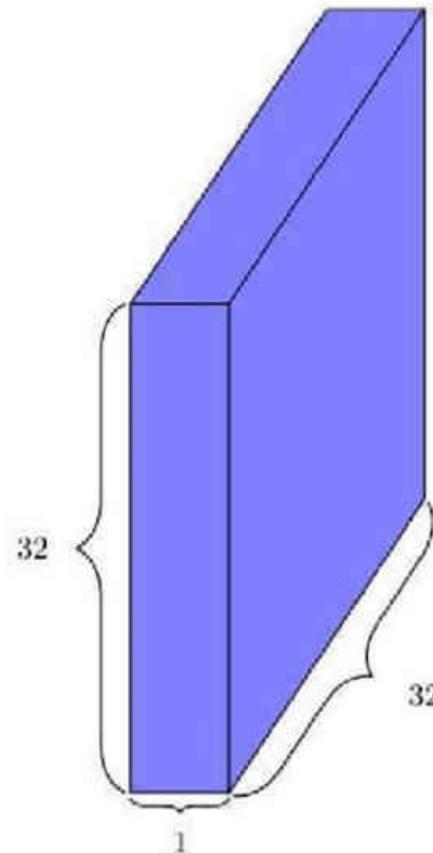
$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$



Example 2

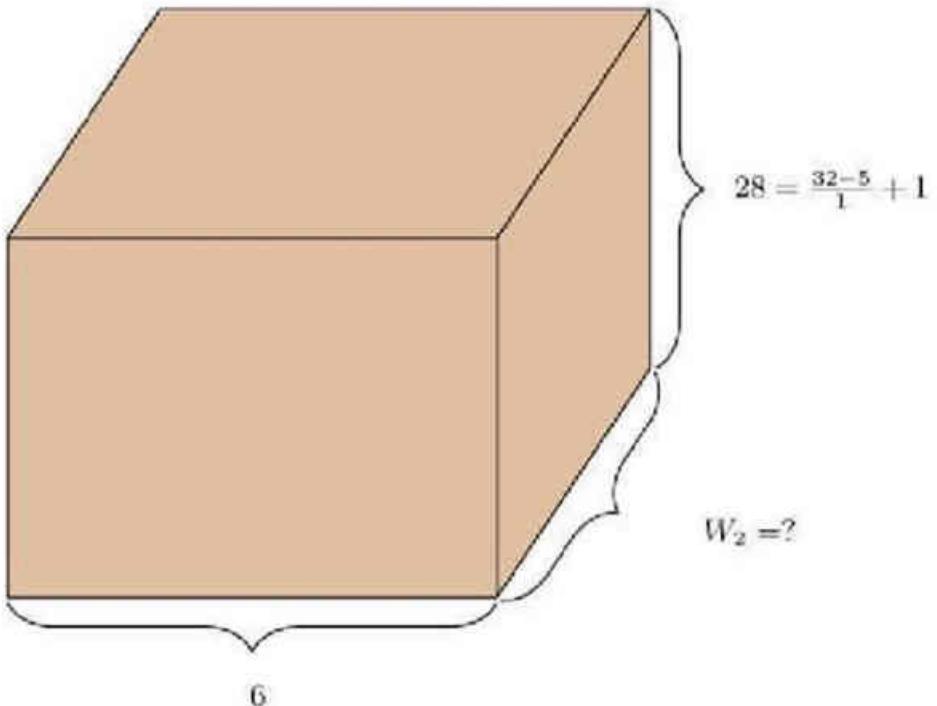
Let us do a few exercises



$$\begin{matrix} * \\ \text{6 filters} \\ \text{Stride} = 1 \\ \text{Padding} = 0 \end{matrix}$$

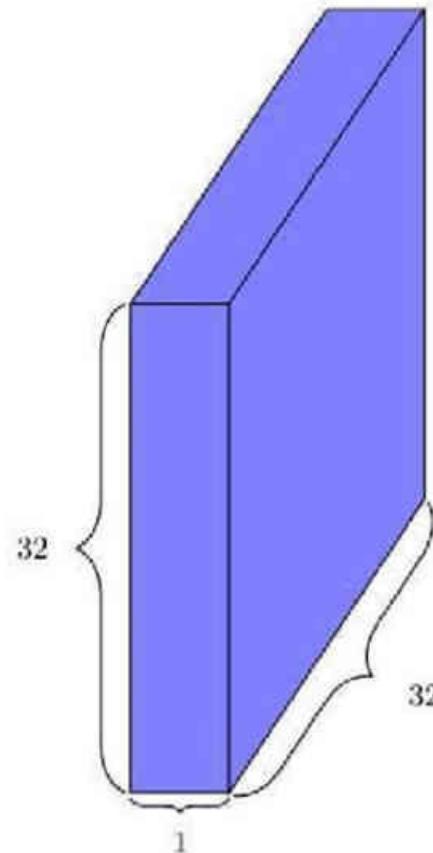
$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$



Example 2

Let us do a few exercises

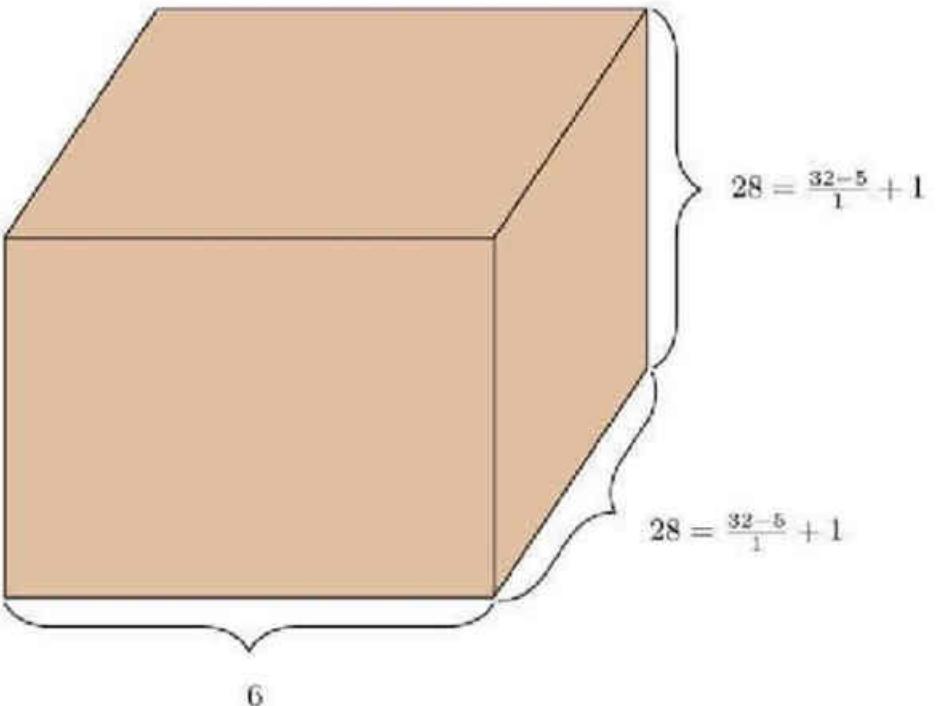


$$* \quad \begin{array}{c} \text{5} \\ \text{1} \\ \text{5} \end{array} =$$

6 filters
Stride = 1
Padding = 0

$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$



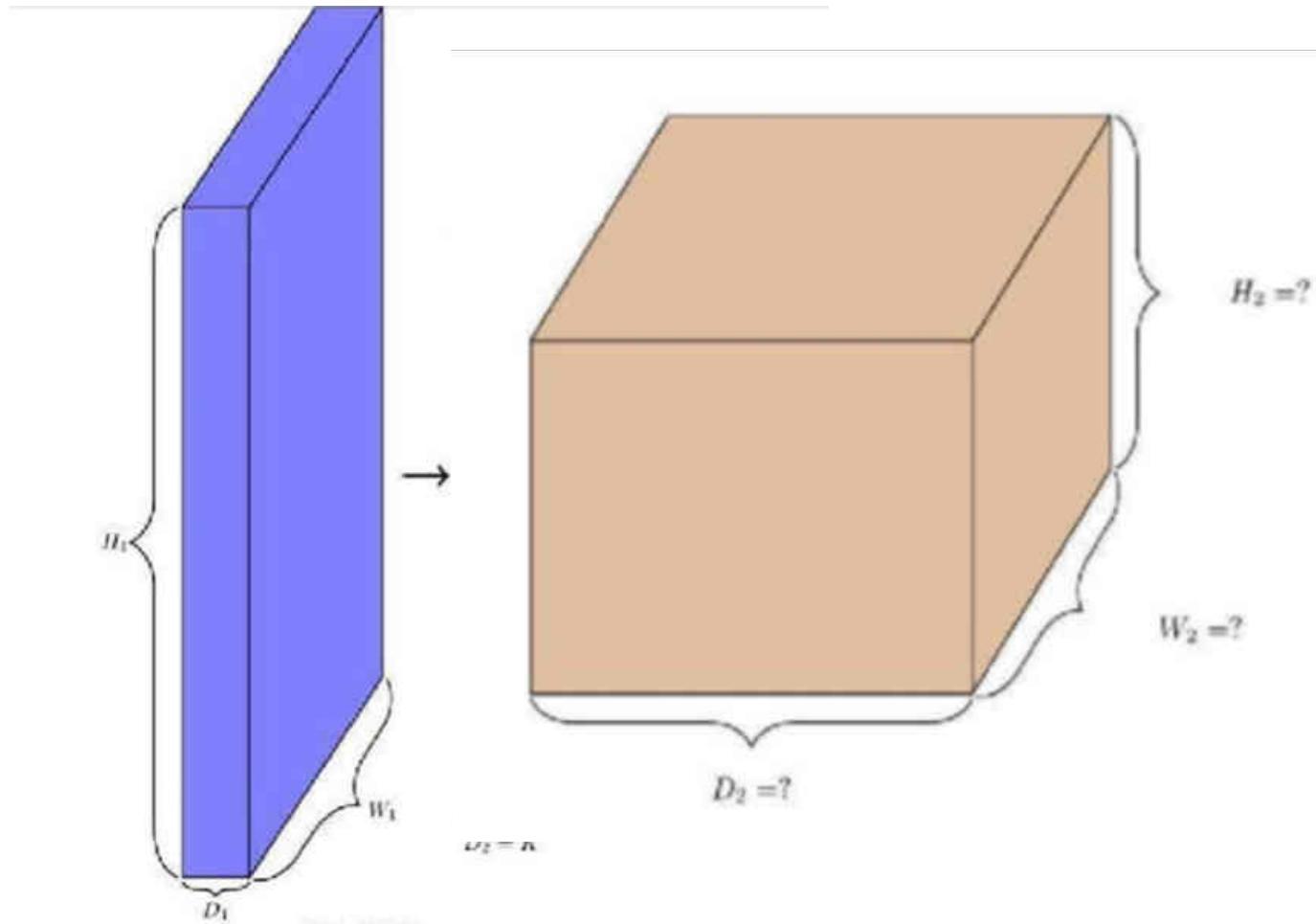
Take-away from the last few slides

- Consider an input image of dimensions **width W1, Height H1 and Depth = D1**
- When this input image is passed through a kernel/ filter with **stride as S** and **padding as P**
- Then we can say that the image after passing through the filter will be of the dimensions **width W2, Height H2 and Depth D2** which is calculated by

$$W2 = \frac{W1 - \text{Width of filter} + 2P}{S} + 1$$

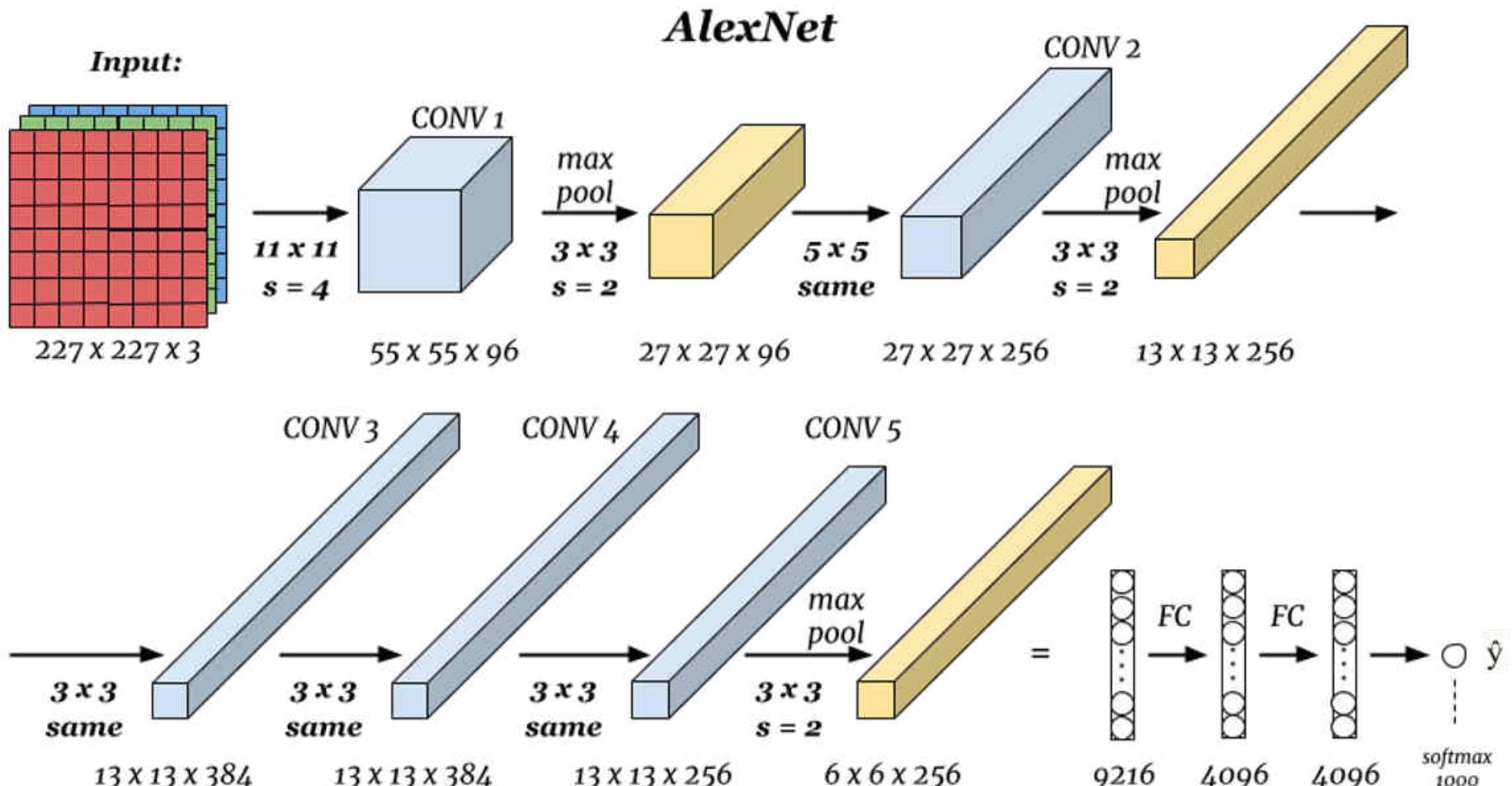
$$H2 = \frac{H1 - \text{Height of filter} + 2P}{S} + 1$$

D2 = No. of filters



CNN Numericals using AlexNet

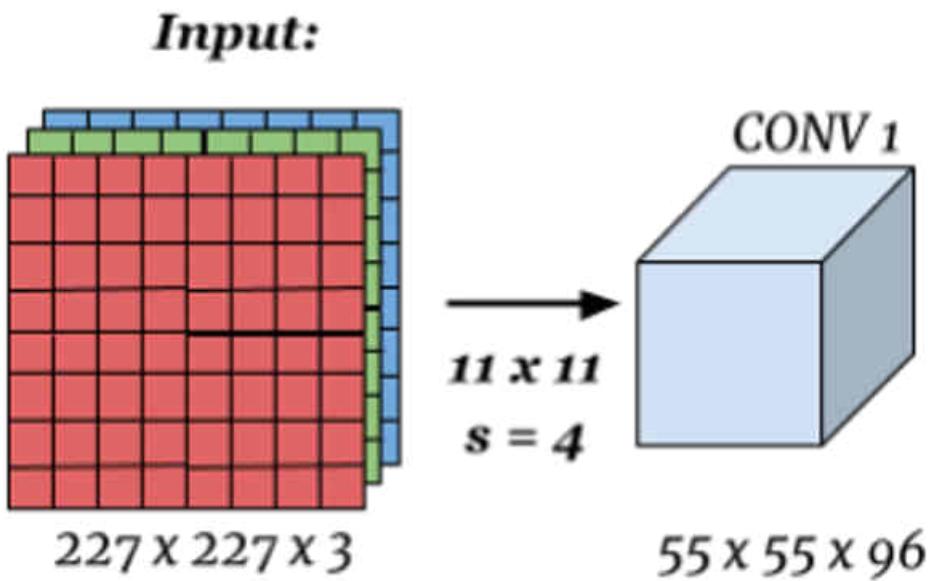
AlexNet is another classic CNN architecture, designed by **Alex Krizhevsky** in collaboration with Ilya Sutskever and Geoffrey Hinton, who was Krizhevsky's Ph.D. advisor. Let's consider this in order to understand dimension calculations.



AlexNet: Dimension Calculations

| | Input Image | Filter | Formula | Output after passing through filter |
|--------|-------------|--------|--|-------------------------------------|
| Height | 227 | 11 | $H2 = \frac{H1 - \text{Height of filter} + 2P}{S} + 1$ | $\frac{(227-11)}{4} + 1 = 55$ |
| Width | 227 | 11 | $W2 = \frac{W1 - \text{Width of filter} + 2P}{S} + 1$ | $\frac{(227-11)}{4} + 1 = 55$ |
| Depth | 3 | 1 | $D2 = \text{No. of filters}$ | 96 |

↑
96 Kernels or filters



Putting things into perspective

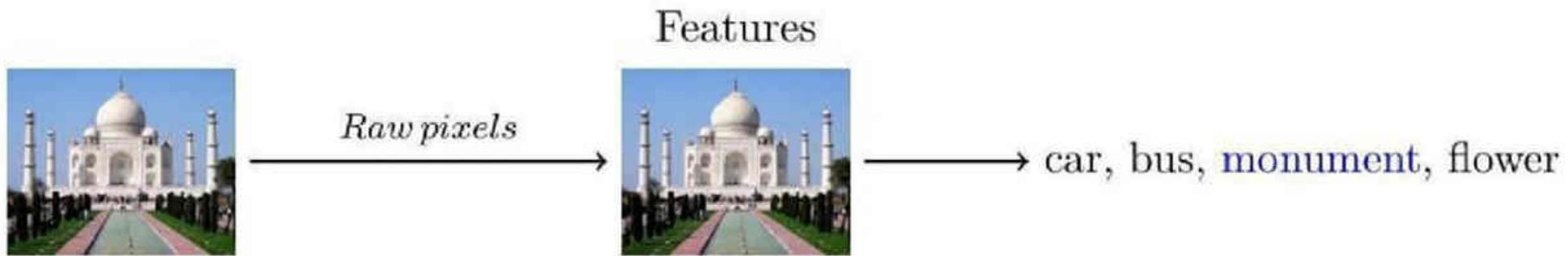
- What is the connection between this operation (convolution) and neural networks?

Putting things into perspective

- What is the connection between this operation (convolution) and neural networks?
- We will try to understand this by considering the task of “image classification”









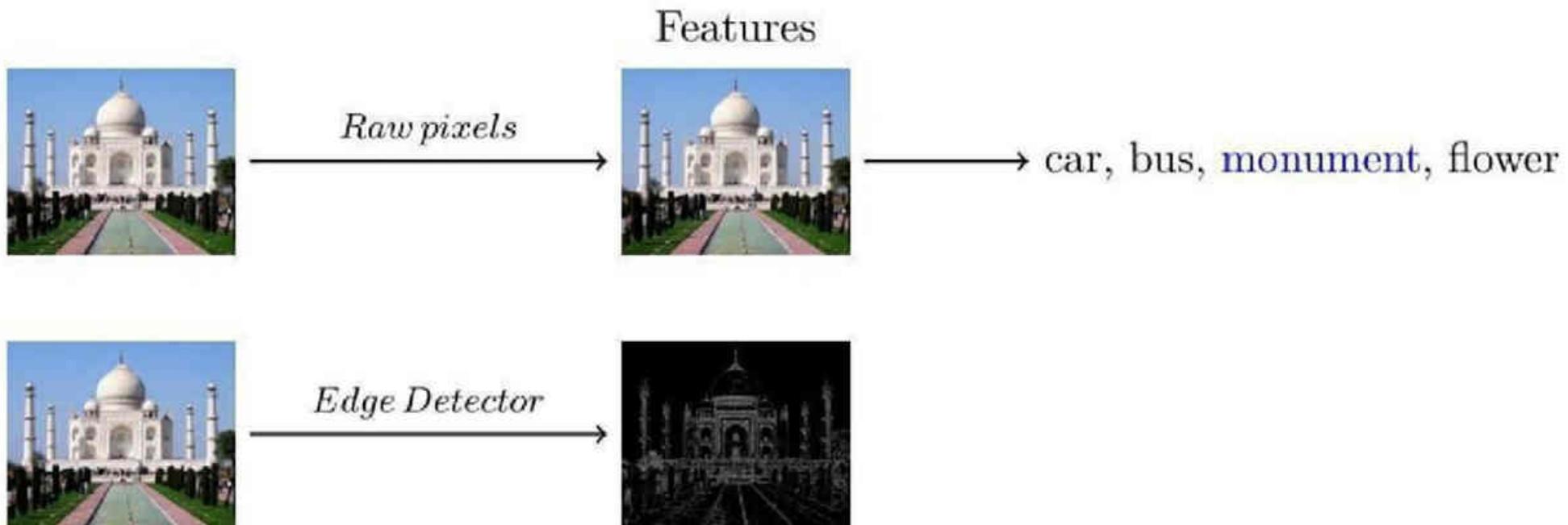
Raw pixels

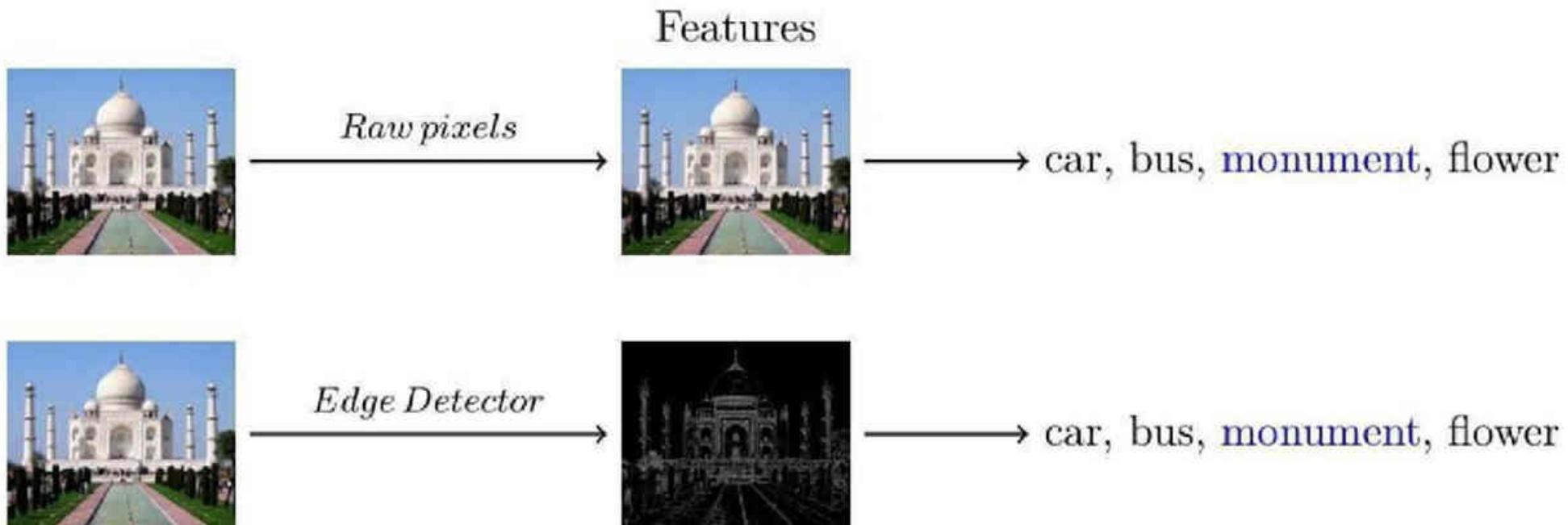
Features

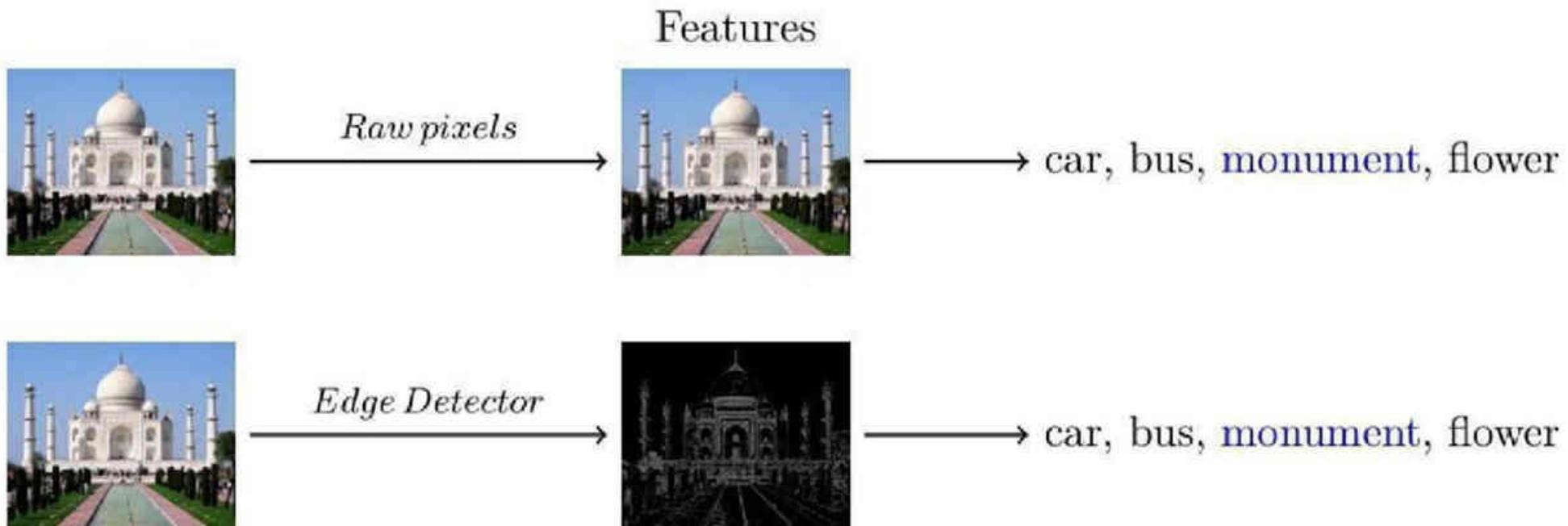


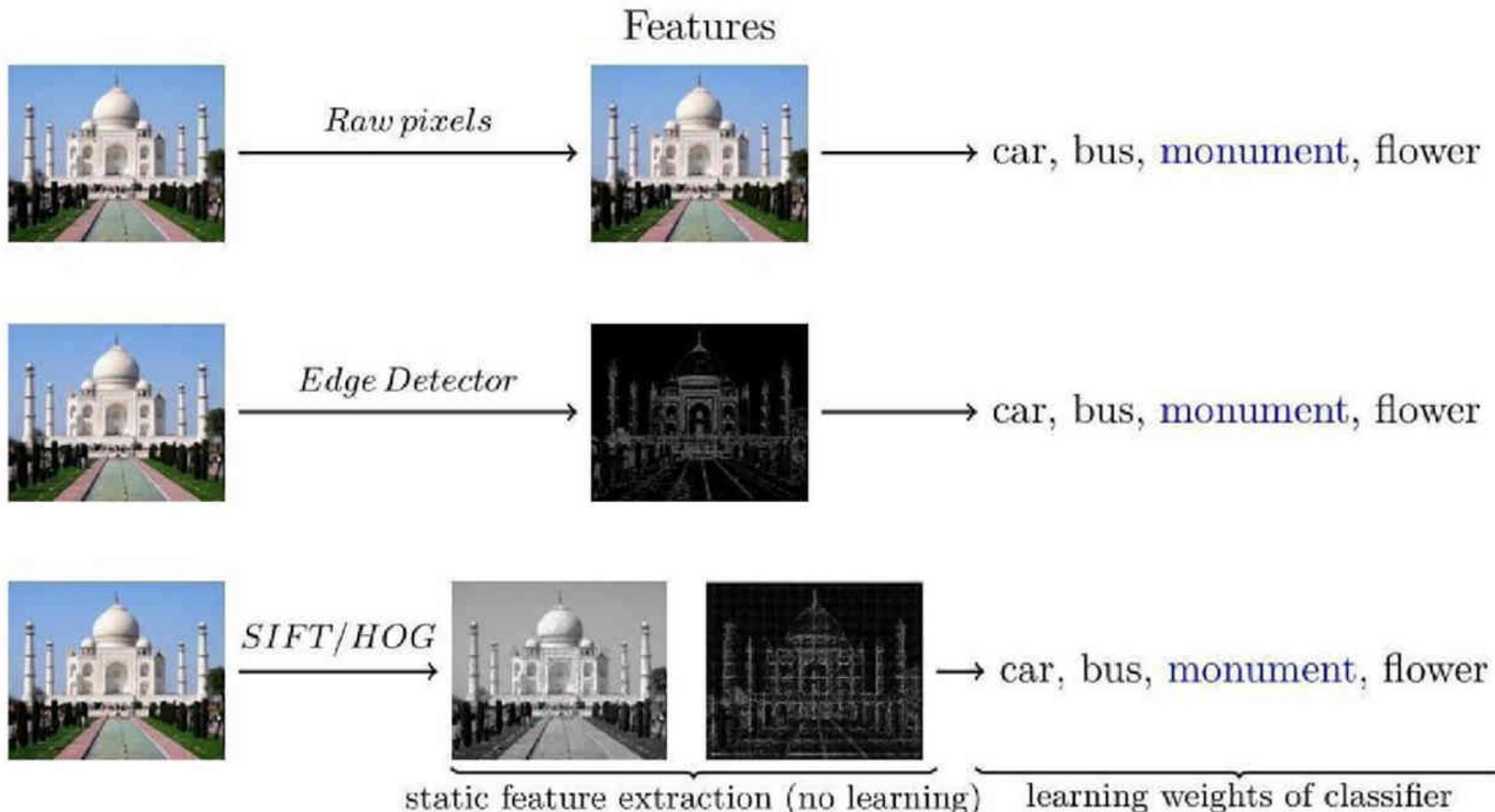
→ car, bus, **monument**, flower



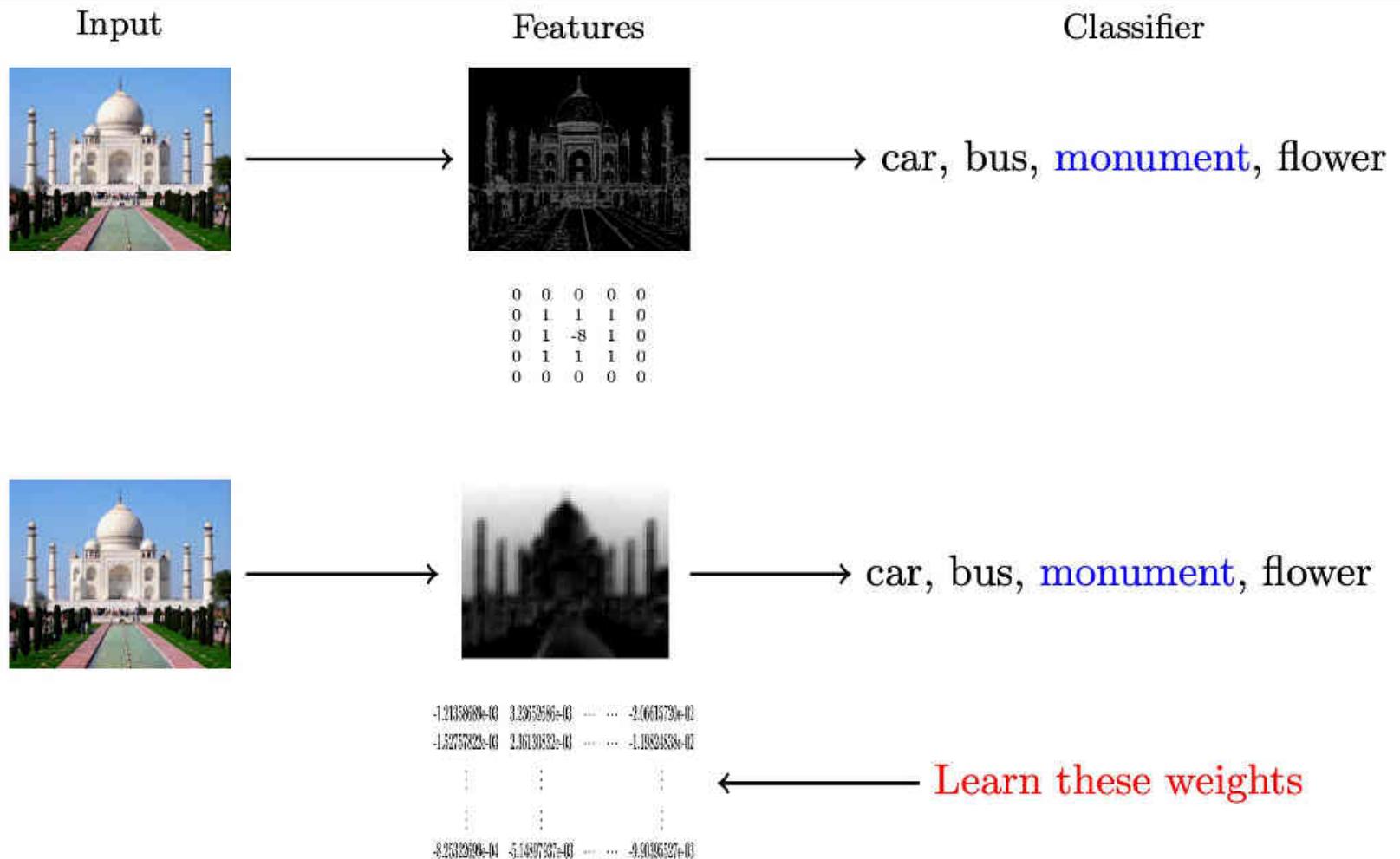




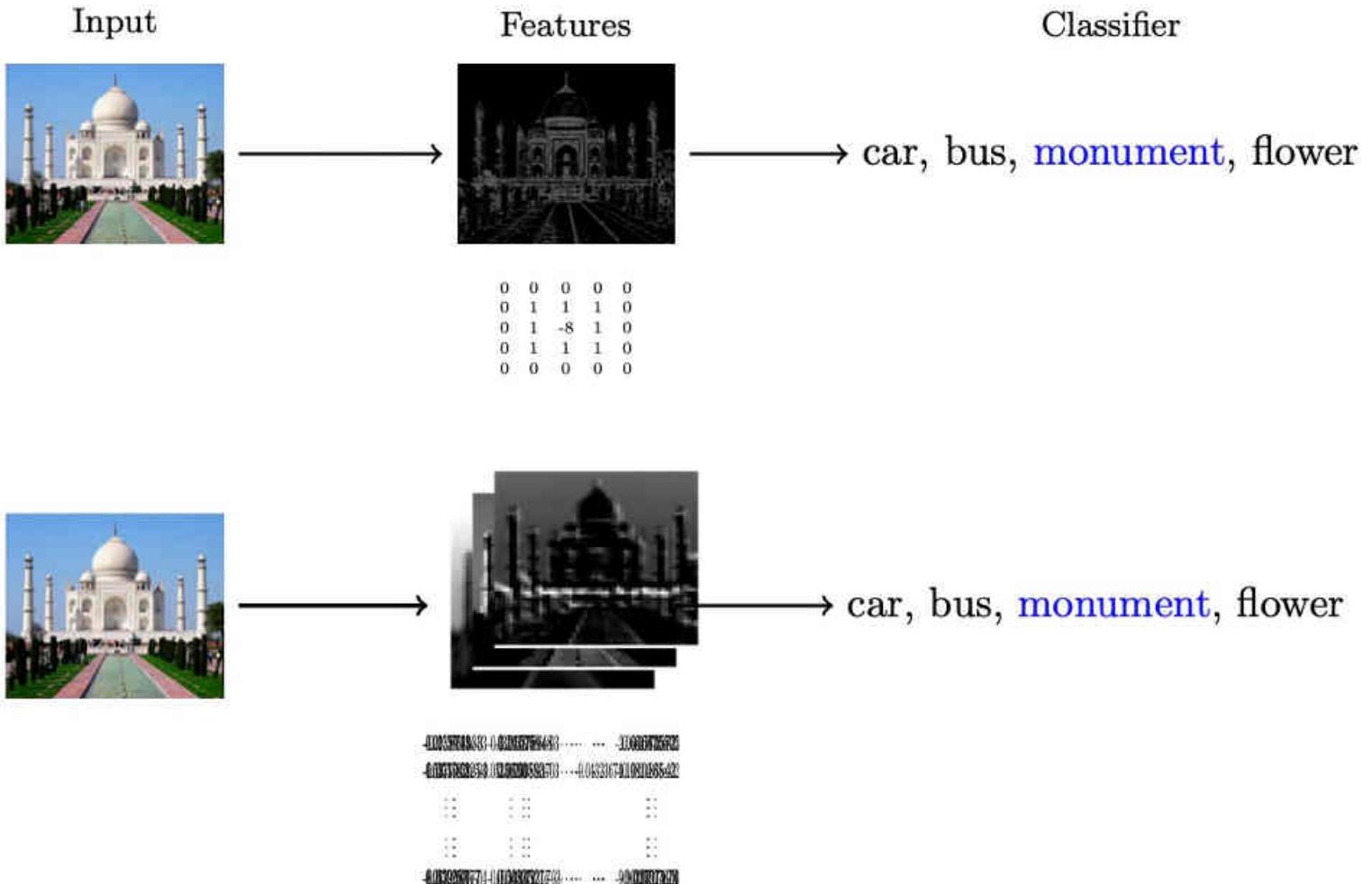




CNN

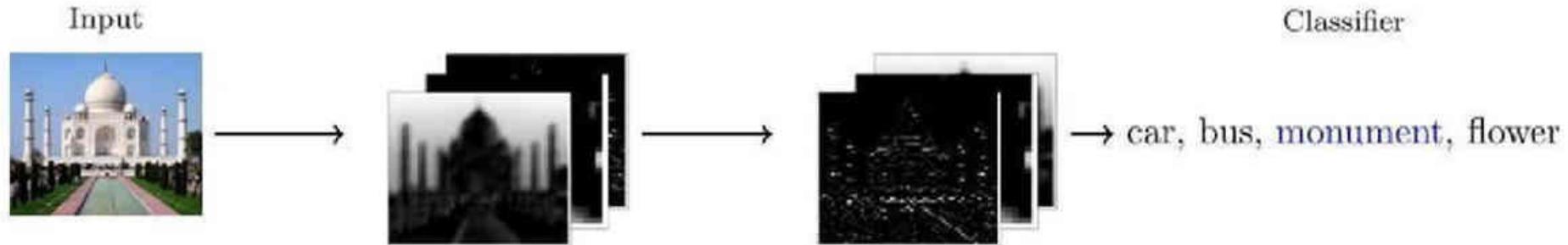


- Instead of using handcrafted kernels such as edge detectors **can we learn meaningful kernels/filters in addition to learning the weights of the classifier?**

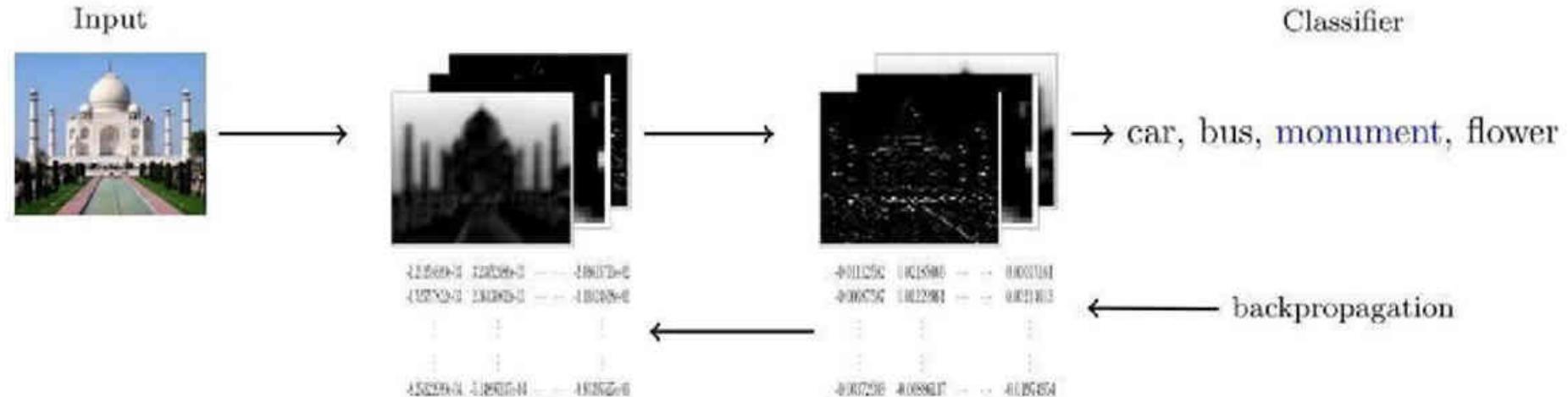


- Even better: Instead of using handcrafted kernels (such as edge detectors) can we learn **multiple** meaningful kernels/filters in addition to learning the weights of the classifier?

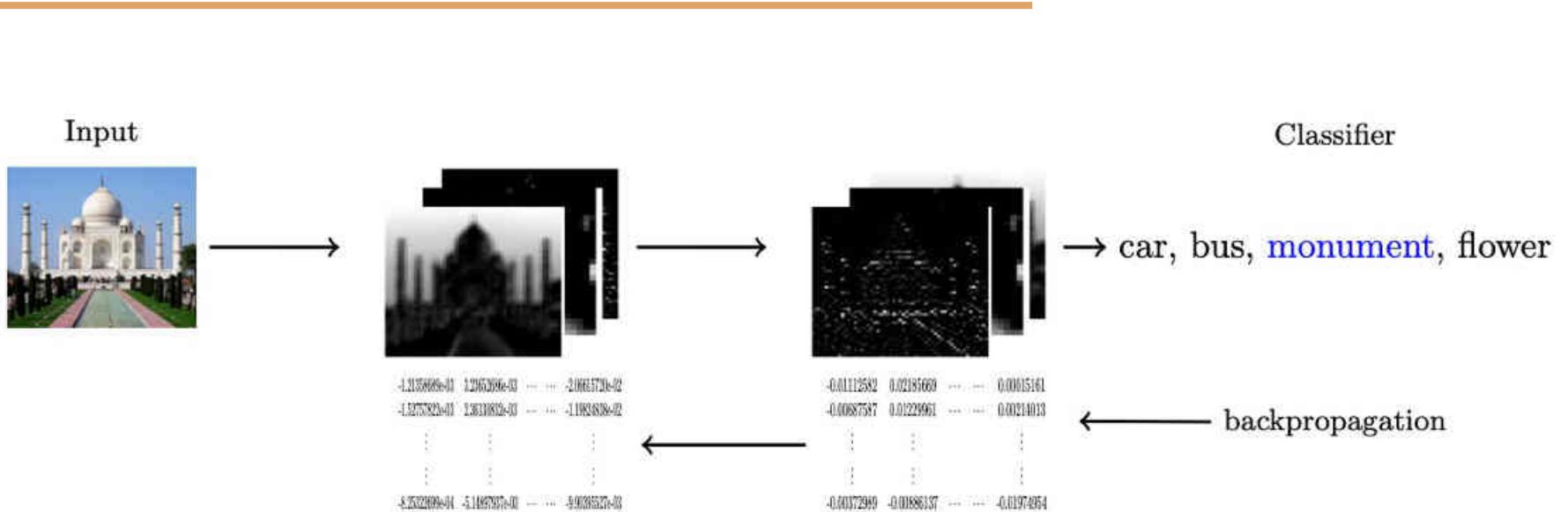
- Can we learn multiple **layers** of meaningful kernels/filters in addition to learning the weights of the classifier?



- Can we learn multiple **layers** of meaningful kernels/filters in addition to learning the weights of the classifier?
- Yes, we can !



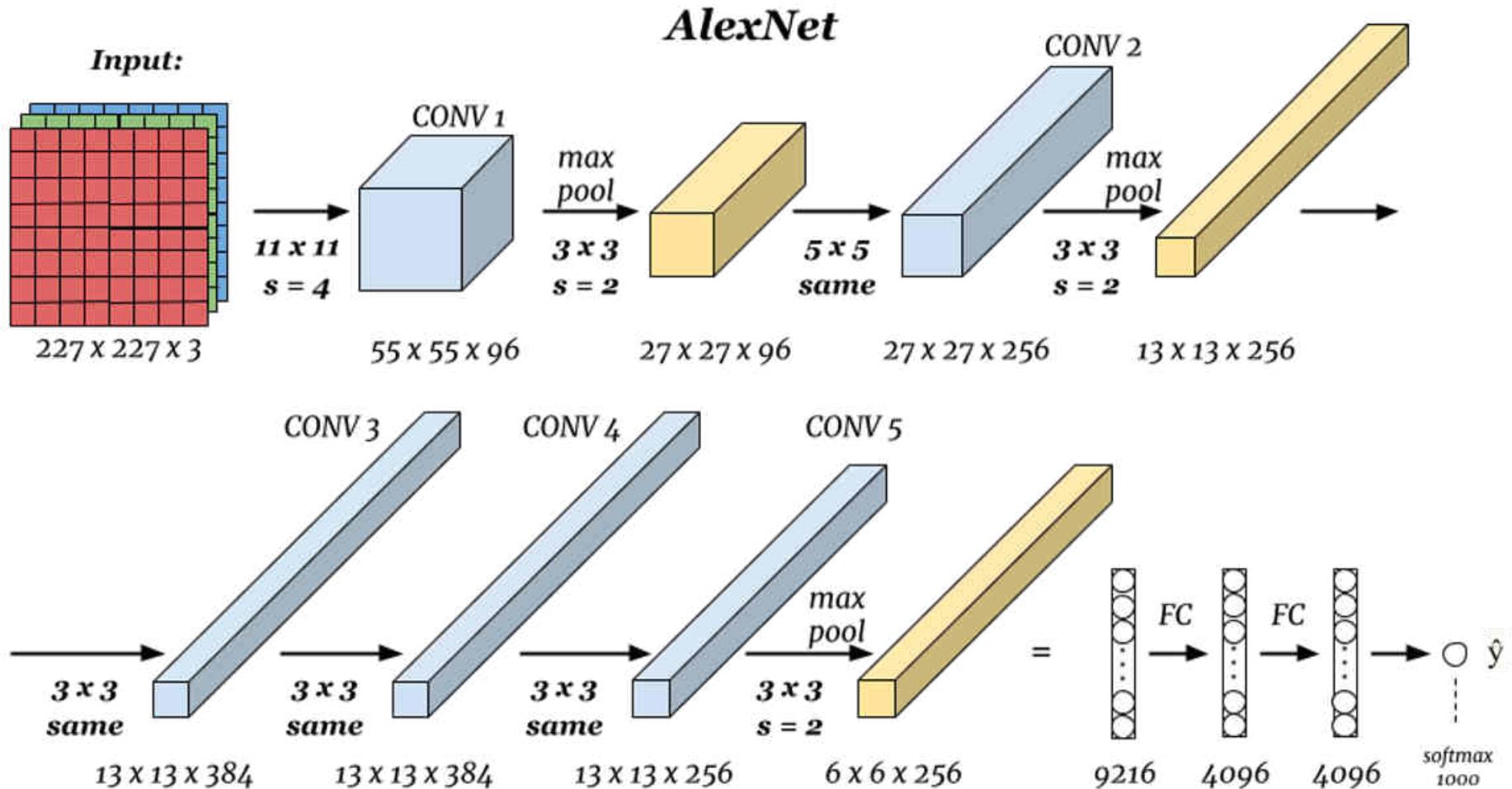
- Can we learn multiple **layers** of meaningful kernels/filters in addition to learning the weights of the classifier?
- Yes, we can !
- Simply by treating these kernels as parameters and learning them in addition to the weights of the classifier (using back propagation)



- Can we learn multiple **layers** of meaningful kernels/filters in addition to learning the weights of the classifier?
- Yes, we can !
- Simply by treating these kernels as parameters and learning them in addition to the weights of the classifier (using back propagation)
- Such a network is called a Convolutional Neural Network.

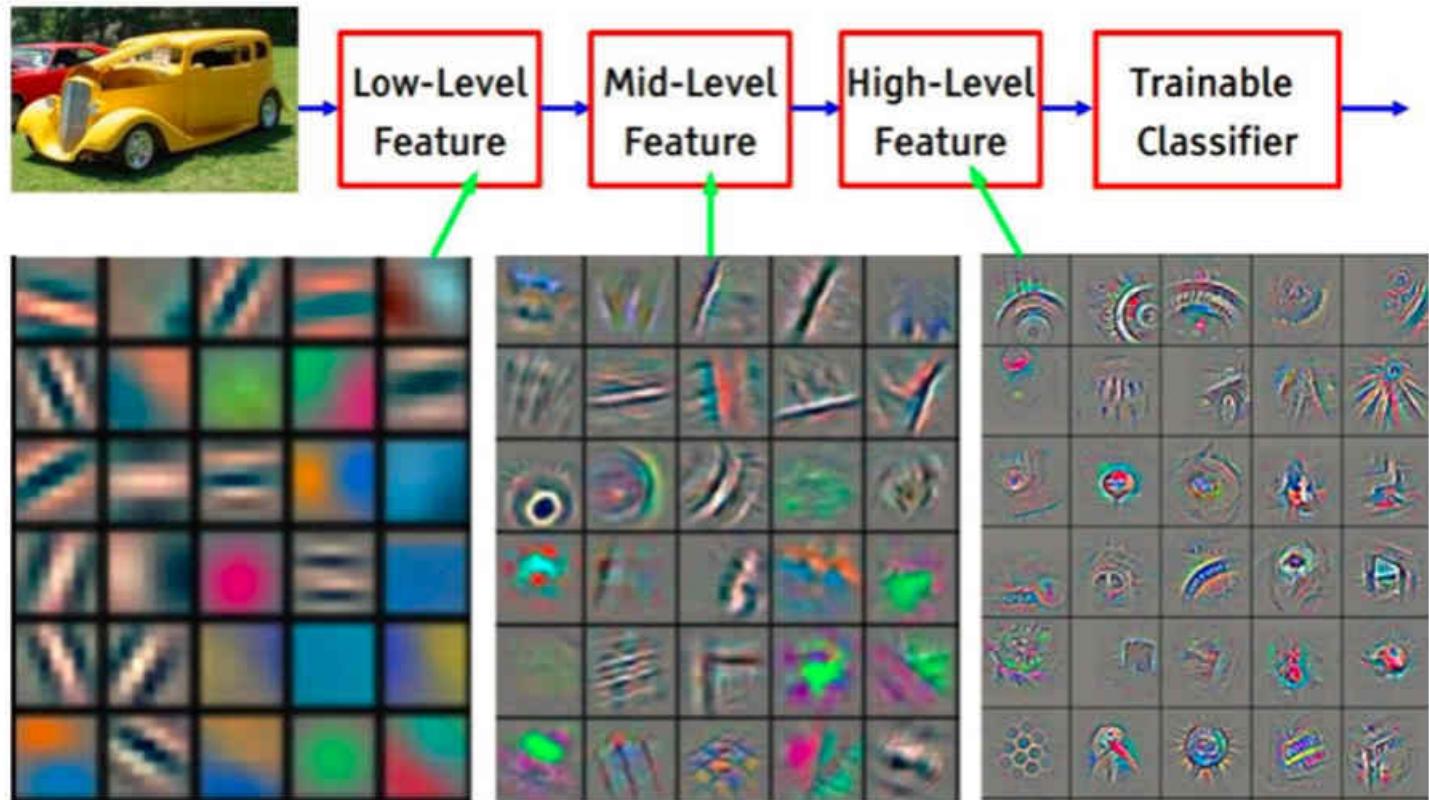
Why do we stack layers?

Let's revisit AlexNet and try to understand the intuition behind stacking layers and feature detection

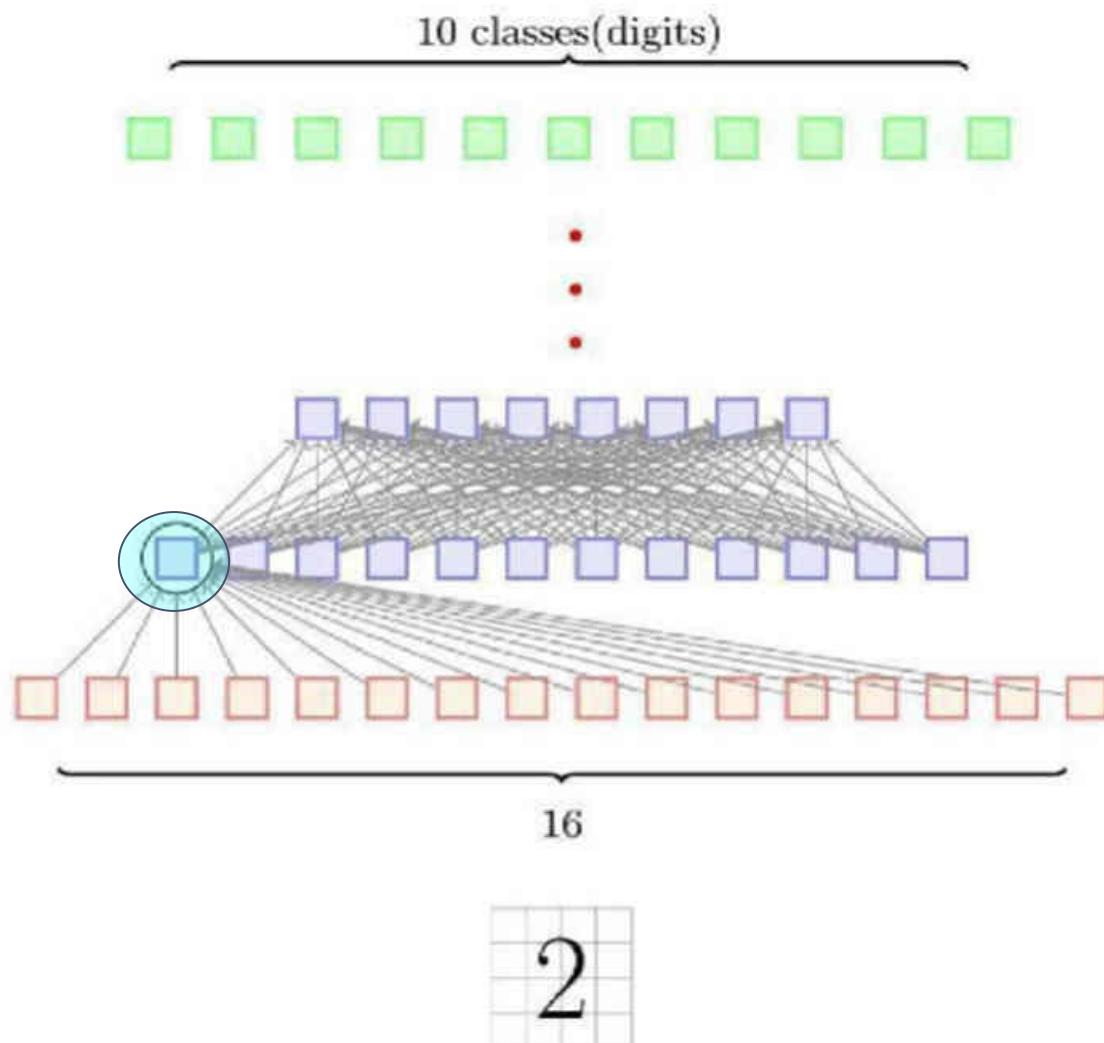


Why do we stack layers?

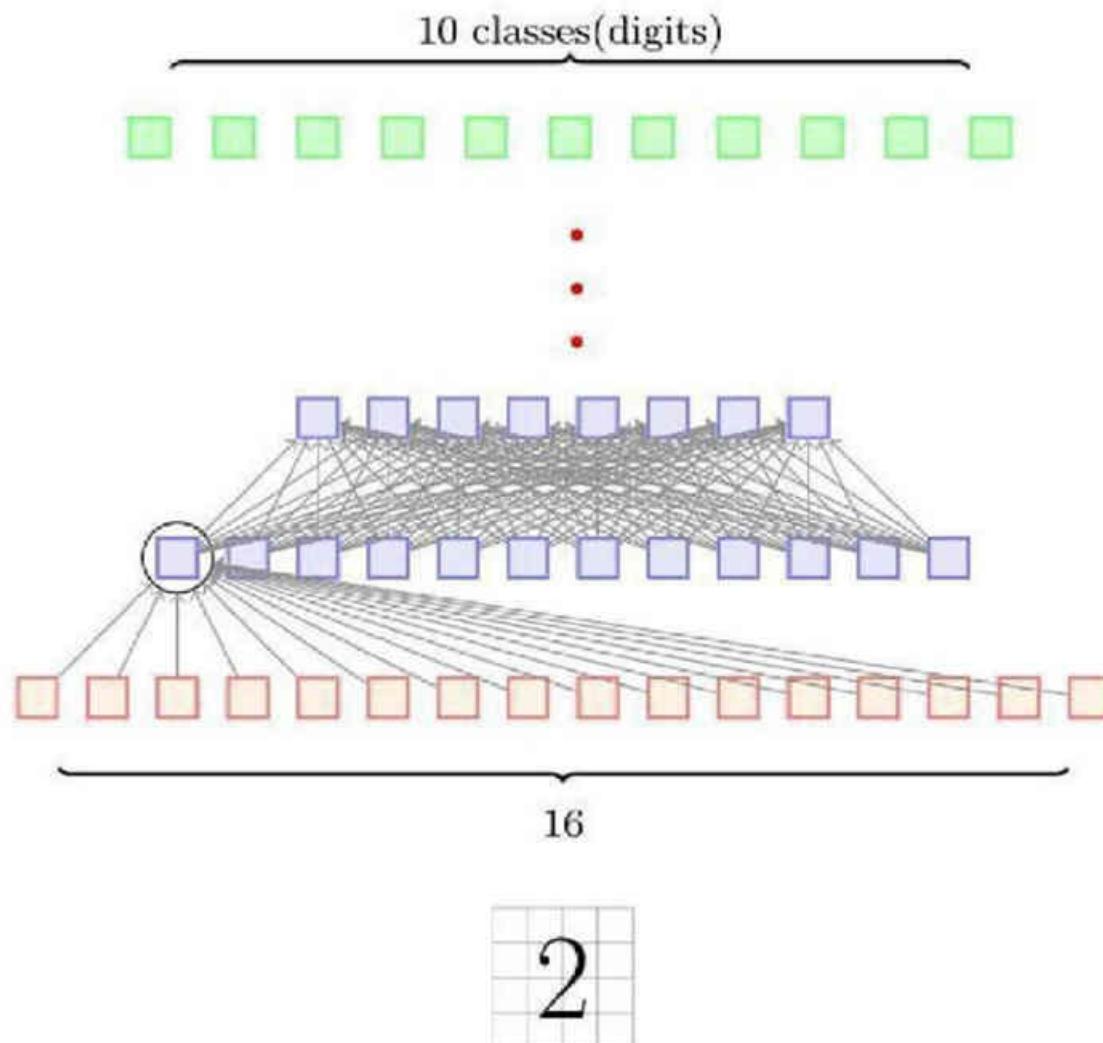
- On stacking the layers CNN learns to extract more and more complex features.
- Initially they extract simple features like edges and progressively capture high level features
- We also know that images progressively shrink after each layer size(O) < size (I)
- Filters at each layer remain the same and find larger compressed/ informed patterns
- Input is weighted sum from previous layers



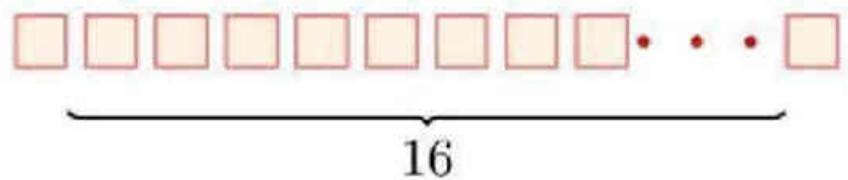
CNN

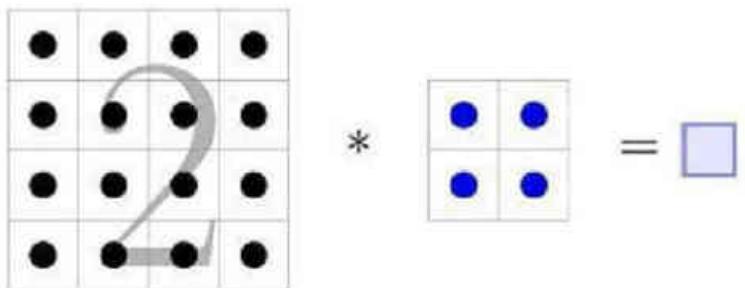


- This is what a regular feed-forward neural network will look like
- There are many dense connections here
- For example all the 16 input neurons are contributing to the computation of h_{11}

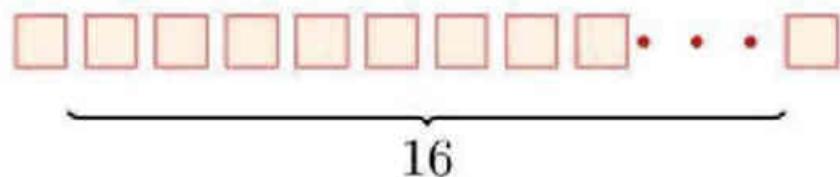


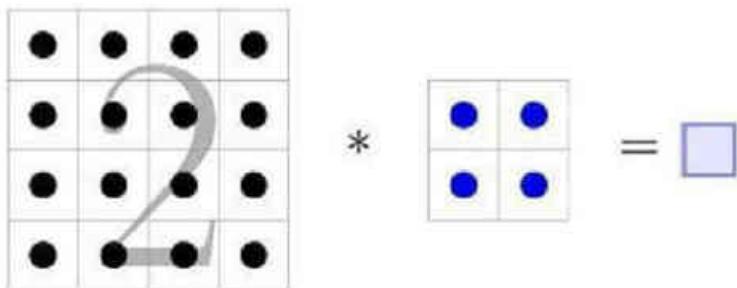
- This is what a regular feed-forward neural network will look like
- There are many dense connections here
- For example all the 16 input neurons are contributing to the computation of h_{11}
- Contrast this to what happens in the case of convolution



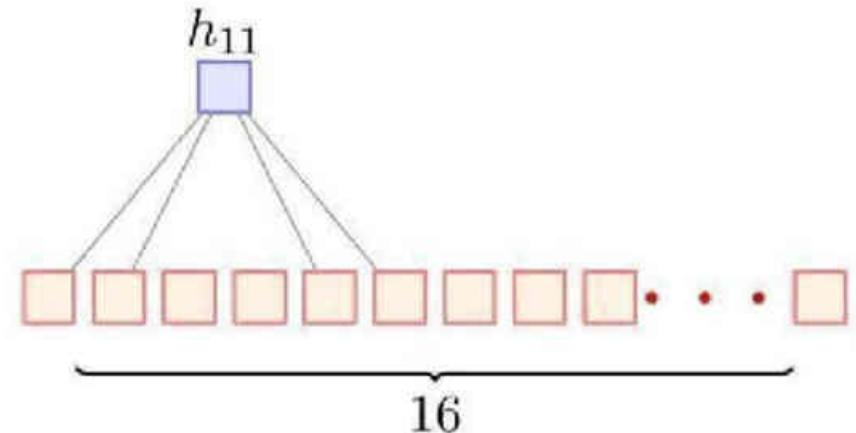

$$\begin{matrix} \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \end{matrix} \quad * \quad \begin{matrix} \bullet & \bullet \\ \bullet & \bullet \end{matrix} = \square$$

- Only a few local neurons participate in the computation of h_{11}

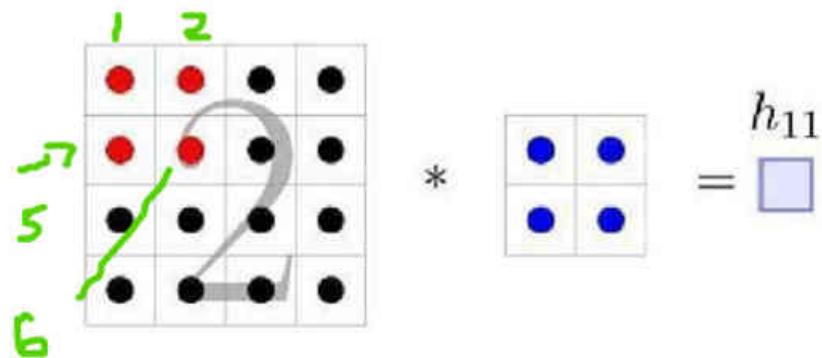




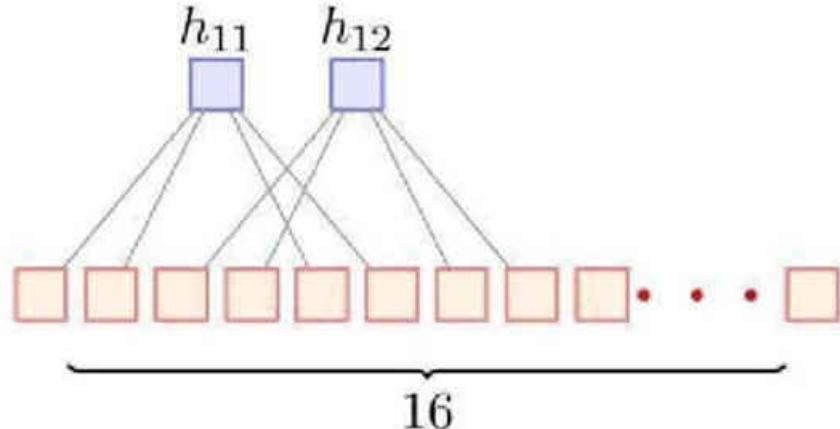
A diagram illustrating a convolution operation. On the left, a 5x5 input grid with a highlighted 3x3 receptive field is shown. In the center, a multiplication symbol (*) is placed between the input grid and a 3x3 kernel containing four blue dots. To the right of the multiplication symbol is an equals sign (=) followed by a purple square, representing the output unit.



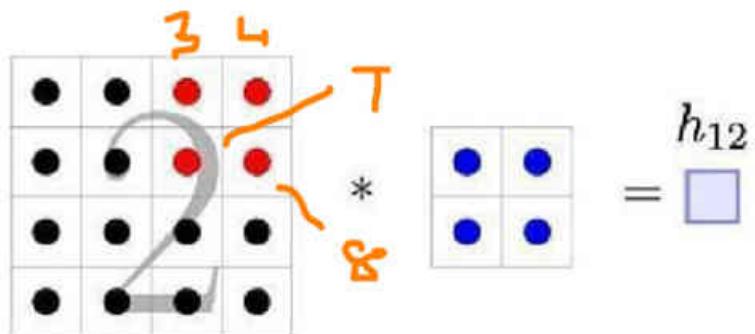
- Only a few local neurons participate in the computation of h_{11}
- For example, only pixels 1, 2, 5, 6 contribute to h_{11}

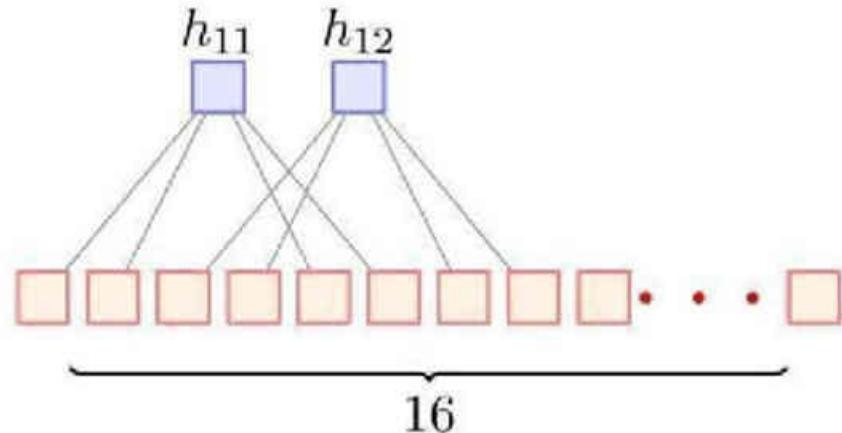


A diagram illustrating a convolution step. An input grid of size 5x5 is multiplied by a kernel of size 3x3. The input grid has red and black dots. The kernel has blue dots. The result is a single value h_{11} , represented by a purple square.

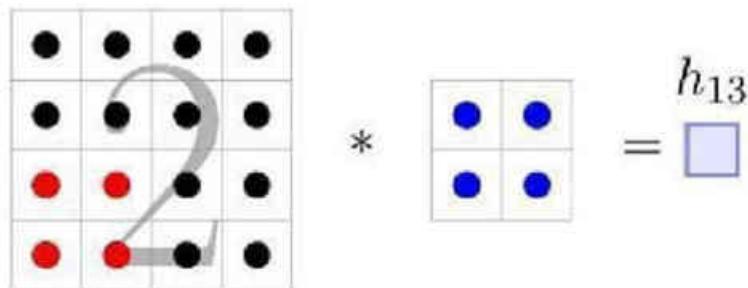


- Only a few local neurons participate in the computation of h_{11}
- For example, only pixels 1, 2, 5, 6 contribute to h_{11}



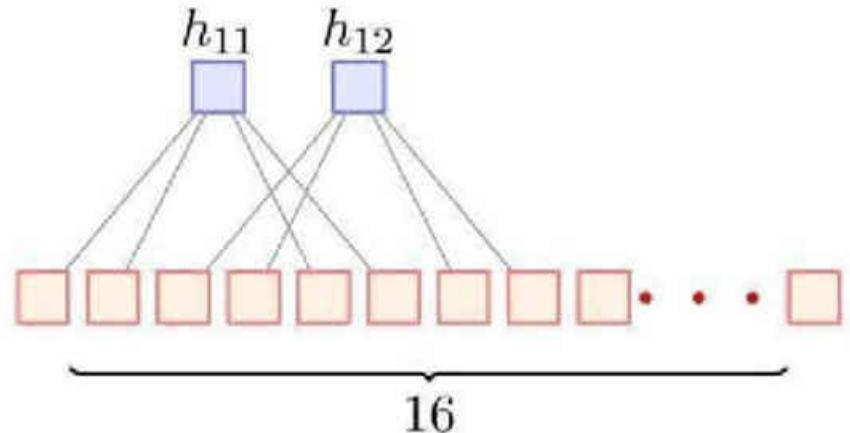


- Only a few local neurons participate in the computation of h_{11}
- For example, only pixels 1, 2, 5, 6 contribute to h_{11}

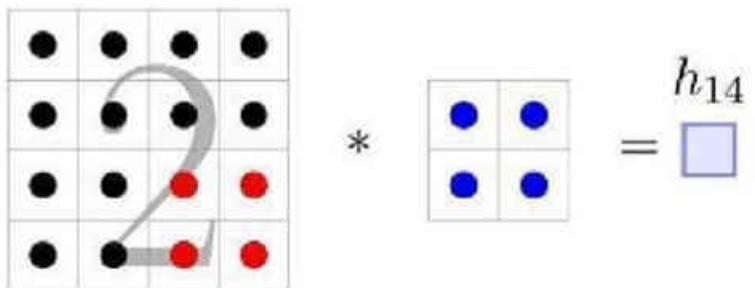


A 5x5 input grid with black and red dots. A 3x3 kernel with blue dots is applied to it. A circled area highlights a 3x3 receptive field. The result of the convolution step is h_{13} , represented by a blue square.

9, 10 ,13, 14

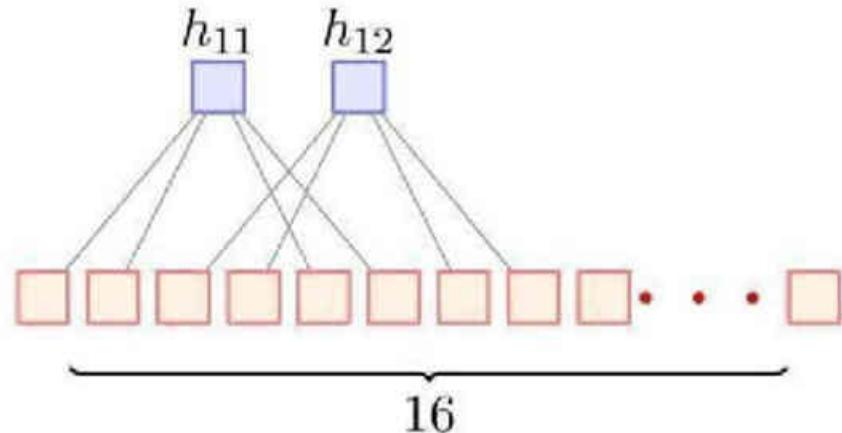


- Only a few local neurons participate in the computation of h_{11}
- For example, only pixels 1, 2, 5, 6 contribute to h_{11}

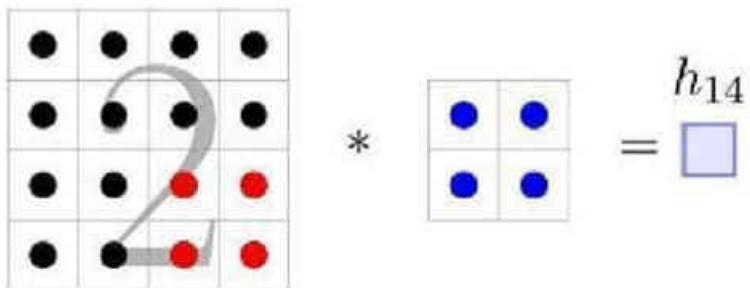


A 5x5 input grid with black dots at positions (1,1), (1,2), (1,3), (1,4), (2,1), (2,2), (2,3), (2,4), (3,1), (3,2), (3,3), (3,4), (4,1), (4,2), (4,3), (4,4) and red dots at positions (4,5), (5,5). A 3x3 kernel with blue dots at all positions is applied to the input. A circled '2' is shown near the input grid. The result is labeled h_{14} in a blue square.

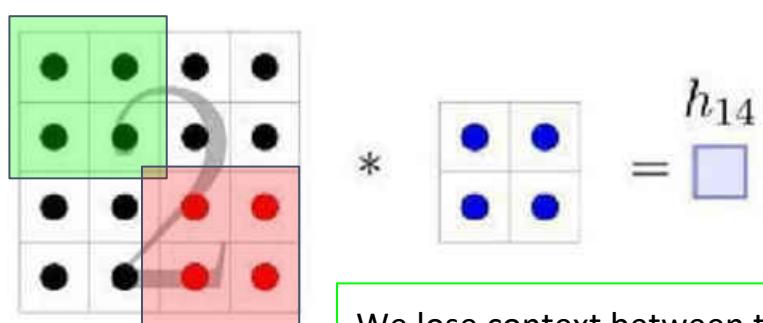
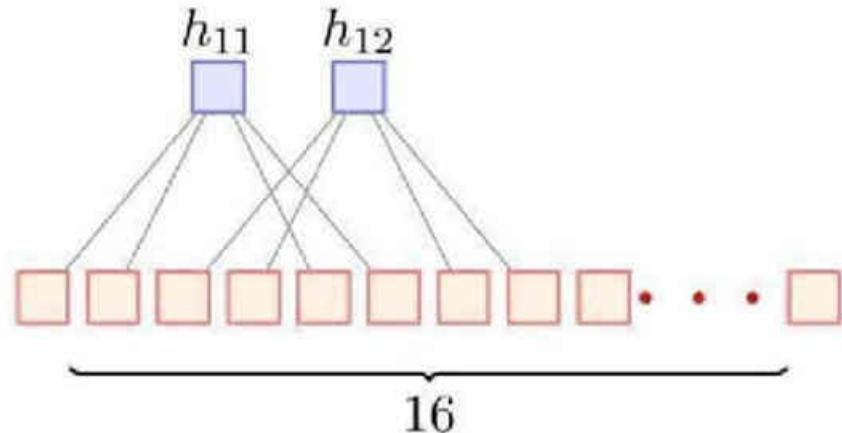
11,12, 15, 16



- Only a few local neurons participate in the computation of h_{11}
- For example, only pixels 1, 2, 5, 6 contribute to h_{11}
- The connections are much sparser

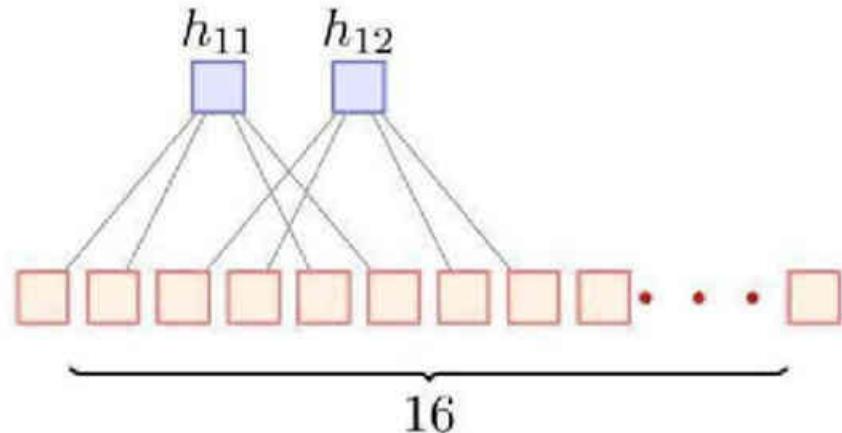


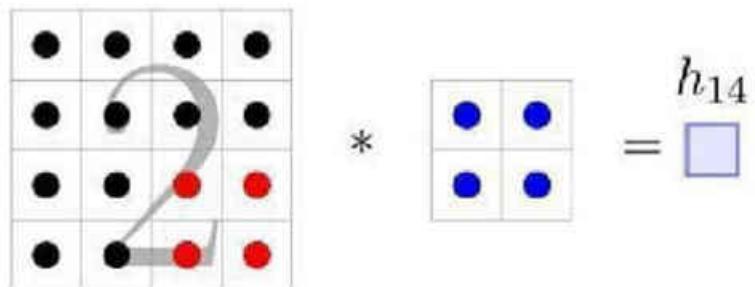
The diagram illustrates a convolution operation. On the left is a 3x3 input kernel with values: black, black, black; black, black, black; red, red, red. A circled '2' is highlighted in the bottom-right corner. To its right is a multiplication symbol (*). Next is a 3x3 weight kernel with values: blue, blue, blue; blue, blue, blue; blue, blue, blue. A circled '2' is highlighted in the top-left corner. To its right is an equals sign (=). Finally, there is a blue square labeled h_{14} .



We lose context between the **green** and the **red** set of points- interactions between neighbours are more interesting

- Only a few local neurons participate in the computation of h_{11}
- For example, only pixels 1, 2, 5, 6 contribute to h_{11}
- The connections are much sparser
- We are taking advantage of the structure of the image(interactions between neighboring pixels are more interesting)
- This **sparse connectivity** reduces the number of parameters in the model





The diagram illustrates the computation of output unit h_{14} . An input image patch (3x3 grid with black and red dots) is multiplied (*) by a convolutional kernel (3x3 grid with blue dots). The result is shown as a small square labeled h_{14} .

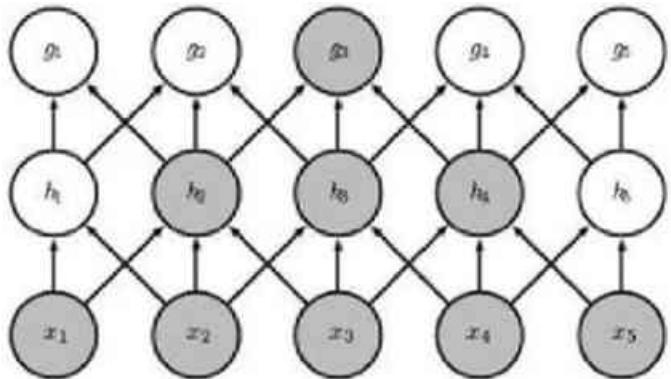
- Only a few local neurons participate in the computation of h_{11}
- For example, only pixels 1, 2, 5, 6 contribute to h_{11}
- The connections are much sparser
- We are taking advantage of the structure of the image (interactions between neighboring pixels are more interesting)
- This **sparse connectivity** reduces the number of parameters in the model

- But is sparse connectivity really good thing ?

* Goodfellow-et-al-2016

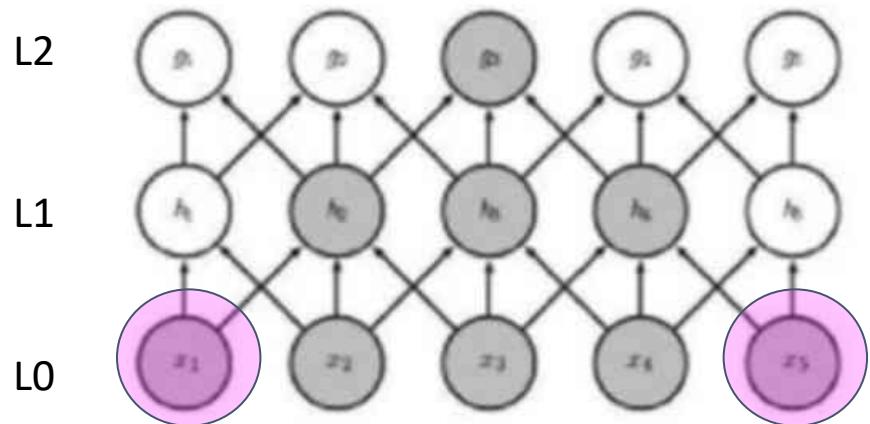
- But is sparse connectivity really good thing ?
- Aren't we losing information (by losing interactions between some input pixels)

* Goodfellow-et-al-2016



- But is sparse connectivity really good thing ?
- Aren't we losing information (by losing interactions between some input pixels)
- Well, not really

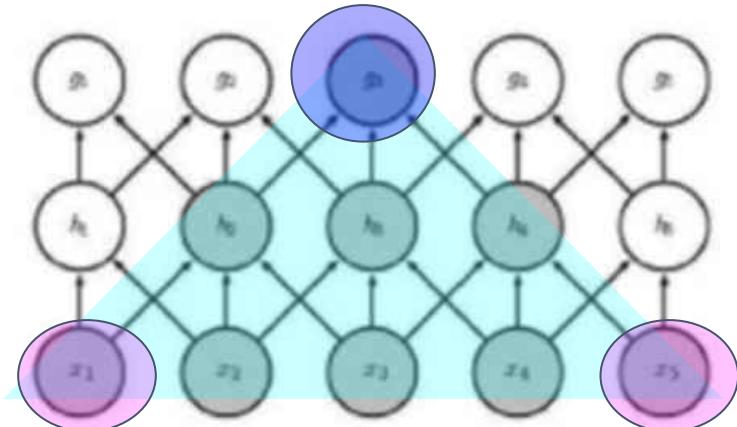
* Goodfellow-et-al-2016



- But is sparse connectivity really good thing ?
- Aren't we losing information (by losing interactions between some input pixels)
 - Well, not really
 - The two highlighted neurons (x_1 & x_5)* do not interact in *layer 1*

* Goodfellow-et-al-2016

Neurons in L0 indirectly contribute to calculate g_3 (in dark blue)



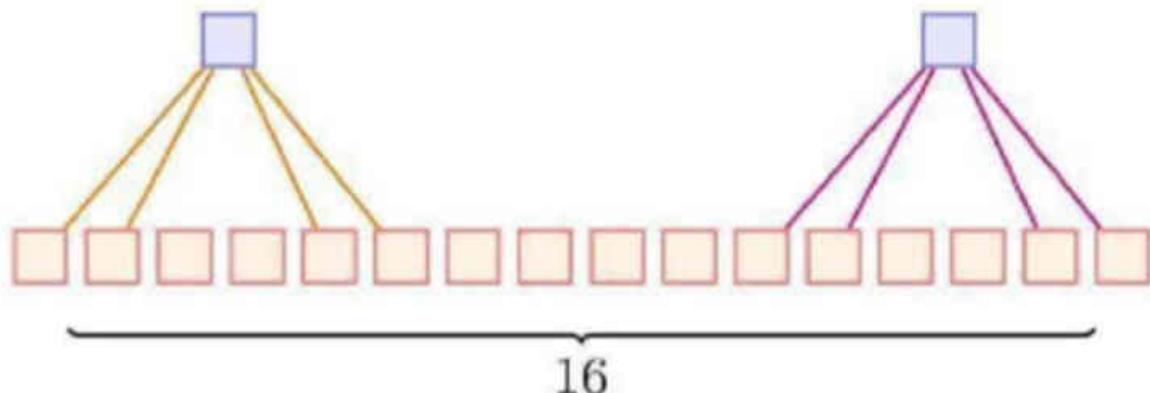
Weighted Sum

- But is sparse connectivity really good thing ?
- Aren't we losing information (by losing interactions between some input pixels)
- Well, not really
- The two highlighted neurons (x_1 & x_5)* do not interact in *layer 1*
- But they indirectly contribute to the computation of g_3 and hence interact indirectly

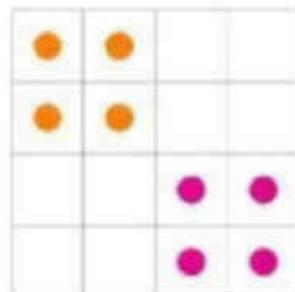
* Goodfellow-et-al-2016

- Another characteristic of CNNs is **weight sharing**

- Another characteristic of CNNs is **weight sharing**
- Consider the following network



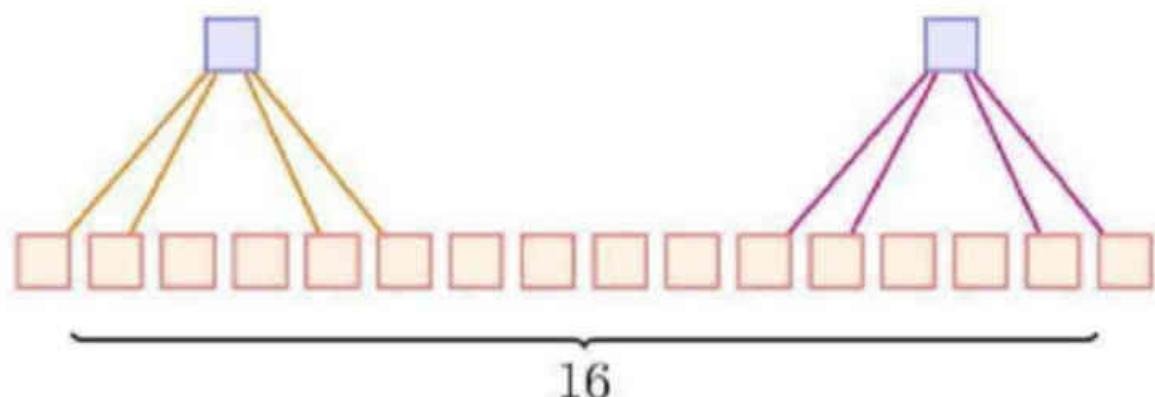
- Kernel 1
- Kernel 2



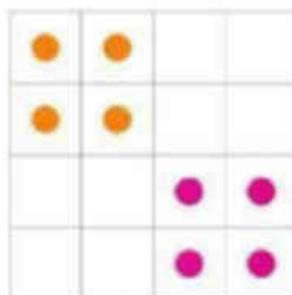
4x4 Image

- Another characteristic of CNNs is **weight sharing**
- Consider the following network

CNN

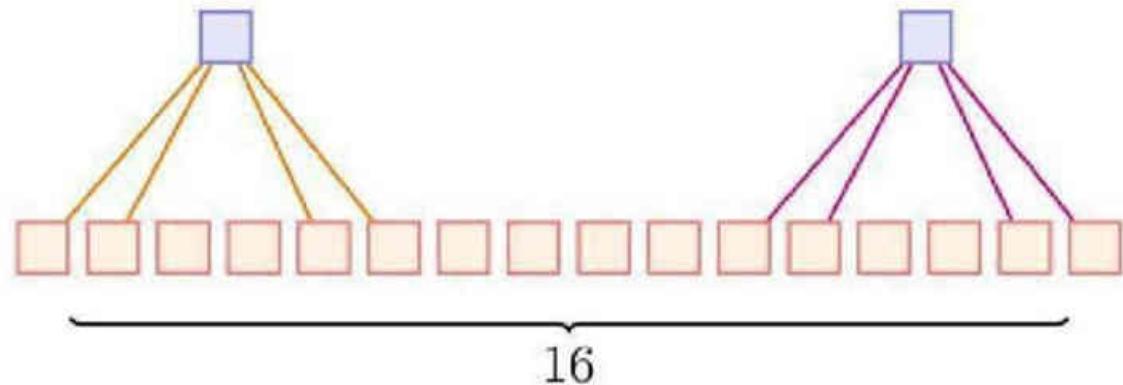


- Kernel 1
- Kernel 2

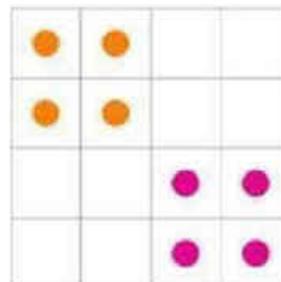


4x4 Image

- Another characteristic of CNNs is **weight sharing**
- Consider the following network
- Do we want the kernel weights to be different for different portions of the image?

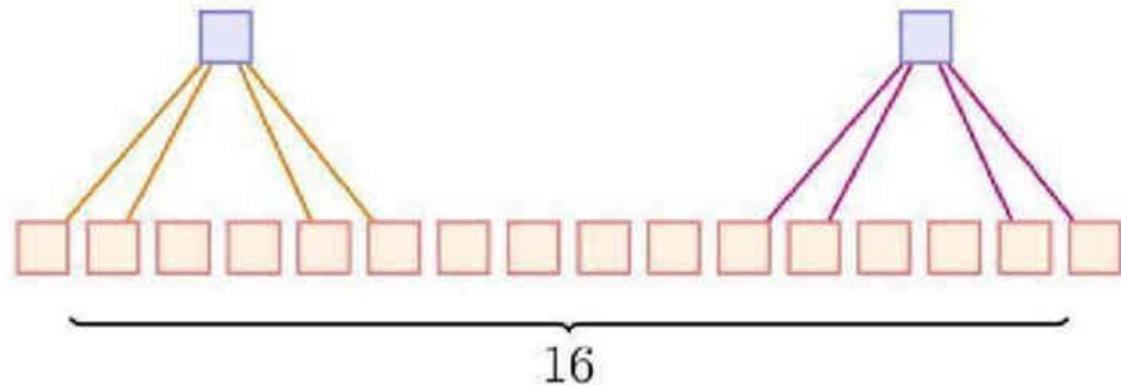


- Kernel 1
- Kernel 2

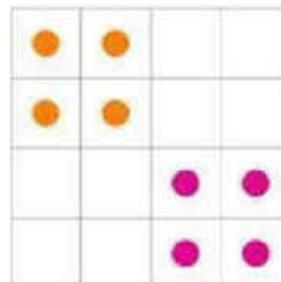


4x4 Image

- Another characteristic of CNNs is **weight sharing**
- Consider the following network
- Do we want the kernel weights to be different for different portions of the image?
- Imagine that we are trying to learn a kernel that detects edges



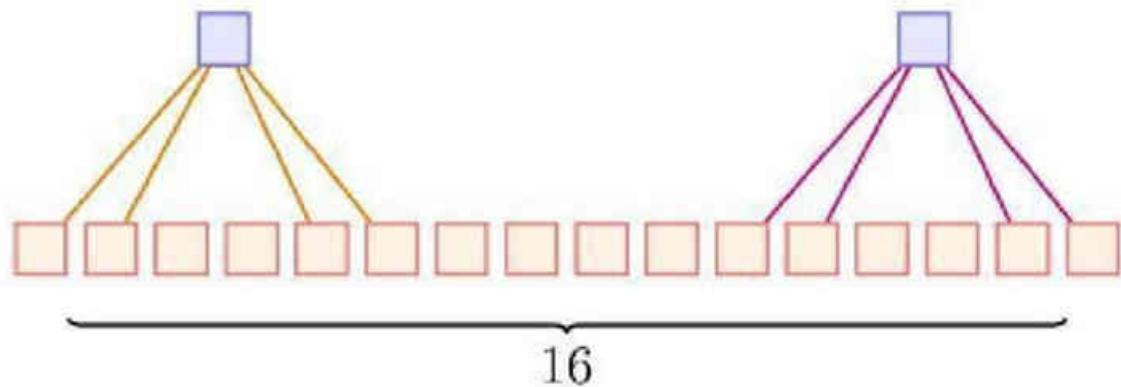
- Kernel 1
- Kernel 2



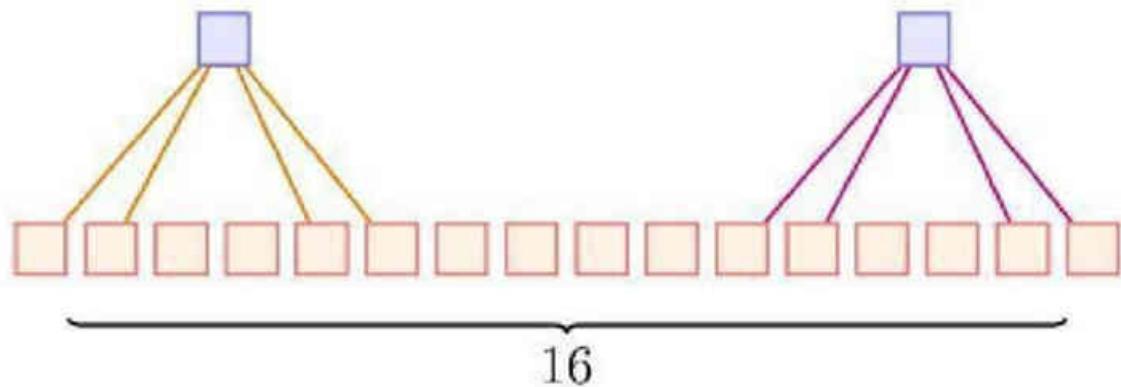
4x4 Image

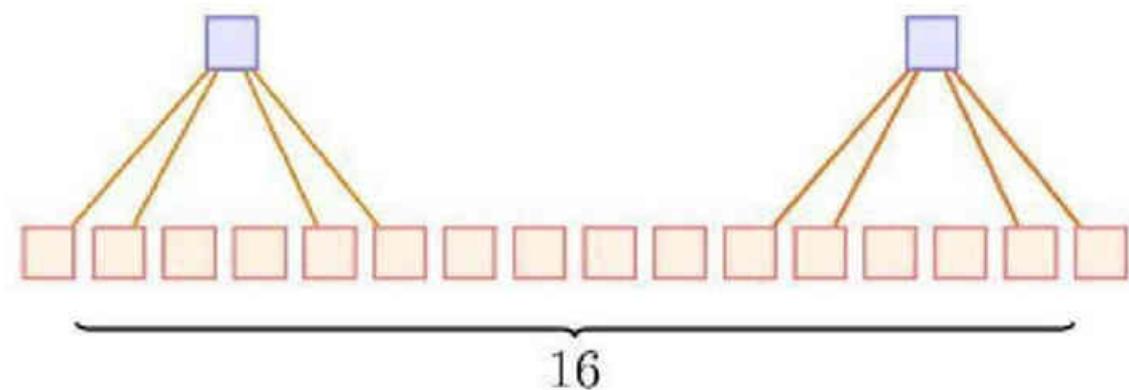
- Another characteristic of CNNs is **weight sharing**
- Consider the following network
- Do we want the kernel weights to be different for different portions of the image?
- Imagine that we are trying to learn a kernel that detects edges
- Shouldn't we be applying the same kernel at all the portions of the image?

- In other words shouldn't the *orange* and *pink* kernels be the same

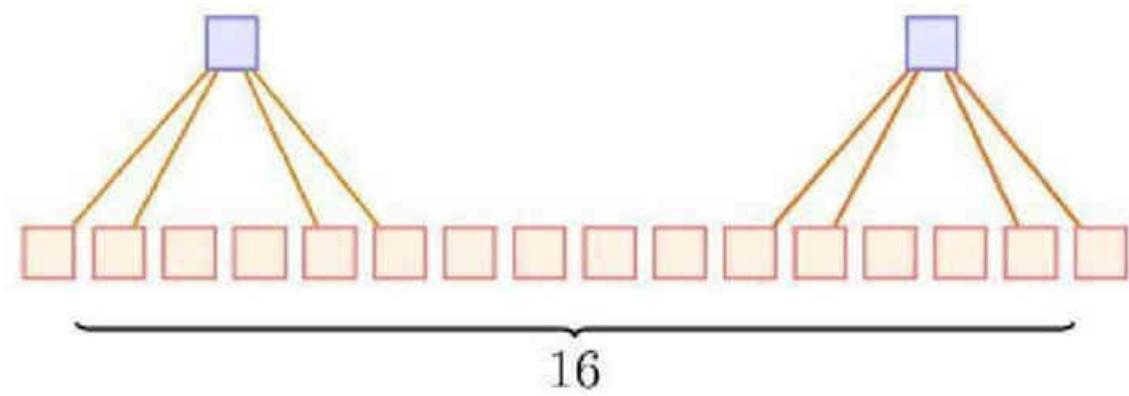


- In other words shouldn't the *orange* and *pink* kernels be the same
- Yes, indeed

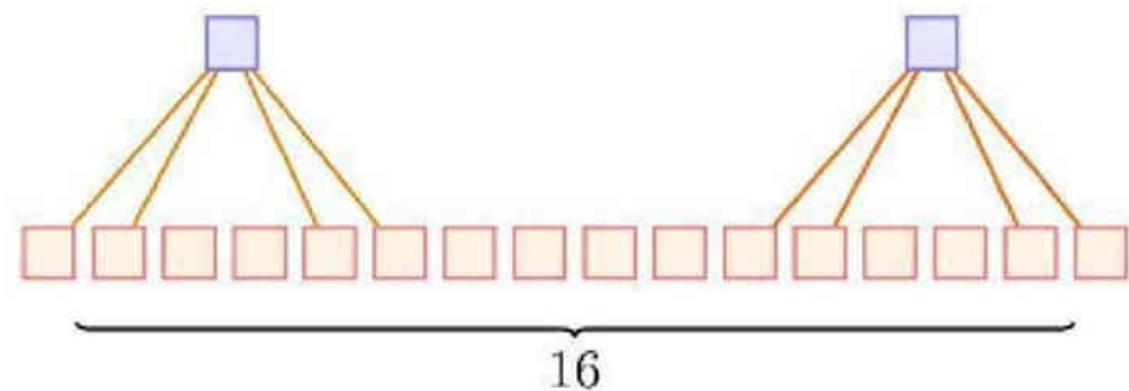




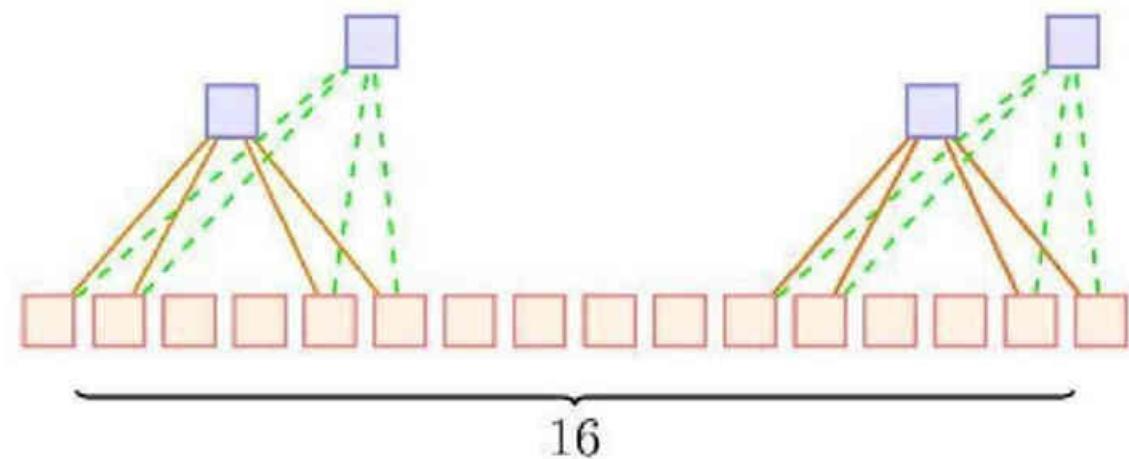
- In other words shouldn't the *orange* and *pink* kernels be the same
- Yes, indeed
- This would make the job of learning easier(instead of trying to learn the same weights/kernels at different locations again and again)



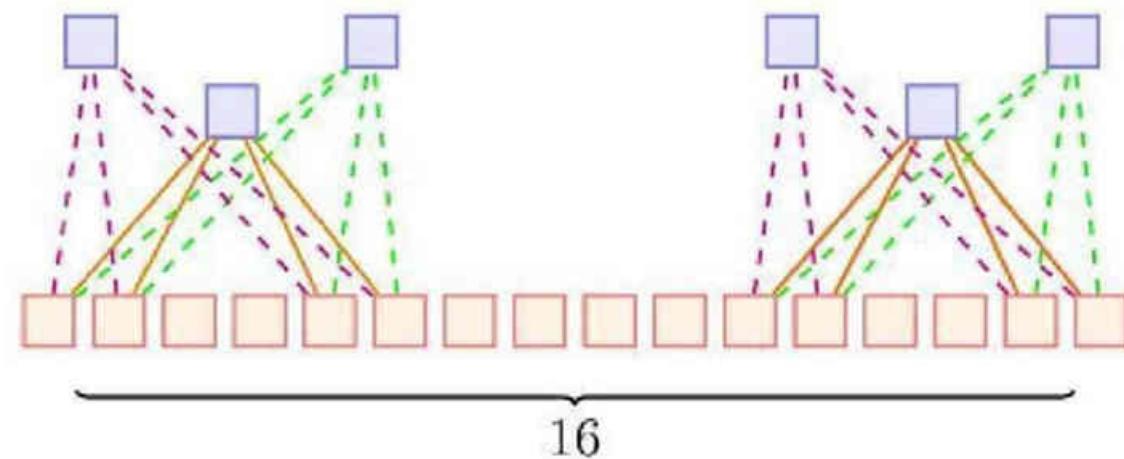
- In other words shouldn't the *orange* and *pink* kernels be the same
- Yes, indeed
- This would make the job of learning easier(instead of trying to learn the same weights/kernels at different locations again and again)
- But does that mean we can have only one kernel?



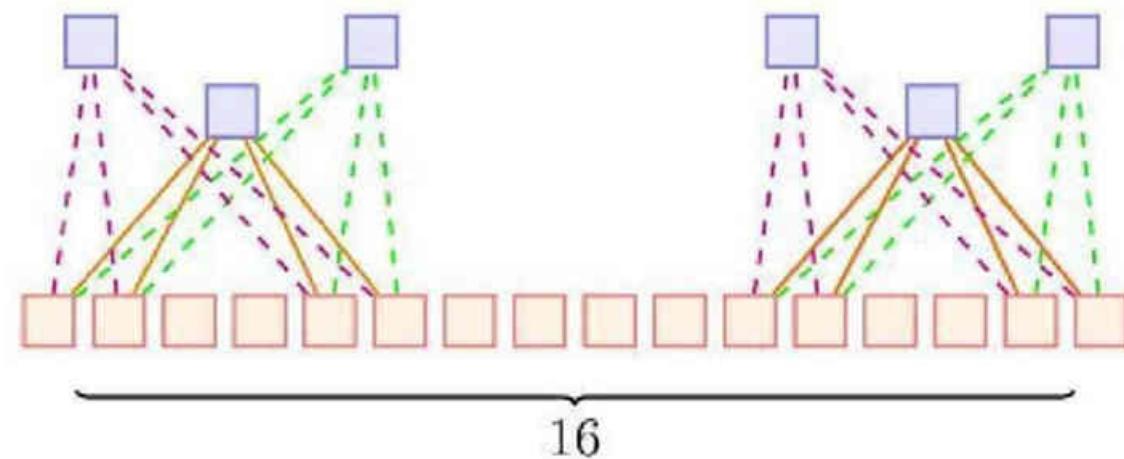
- In other words shouldn't the *orange* and *pink* kernels be the same
- Yes, indeed
- This would make the job of learning easier(instead of trying to learn the same weights/kernels at different locations again and again)
- But does that mean we can have only one kernel?
- No, we can have many such kernels but the kernels will be shared by all locations in the image



- In other words shouldn't the *orange* and *pink* kernels be the same
- Yes, indeed
- This would make the job of learning easier(instead of trying to learn the same weights/kernels at different locations again and again)
- But does that mean we can have only one kernel?
- No, we can have many such kernels but the kernels will be shared by all locations in the image



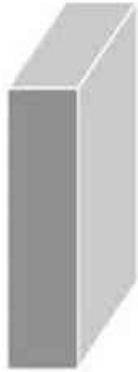
- In other words shouldn't the *orange* and *pink* kernels be the same
- Yes, indeed
- This would make the job of learning easier(instead of trying to learn the same weights/kernels at different locations again and again)
- But does that mean we can have only one kernel?
- No, we can have many such kernels but the kernels will be shared by all locations in the image



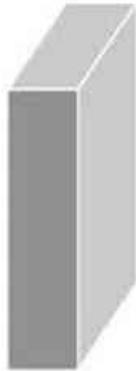
- In other words shouldn't the *orange* and *pink* kernels be the same
- Yes, indeed
- This would make the job of learning easier(instead of trying to learn the same weights/kernels at different locations again and again)
- But does that mean we can have only one kernel?
- No, we can have many such kernels but the kernels will be shared by all locations in the image
- This is called “weight sharing”

- So far, we have focused only on the convolution operation

- So far, we have focused only on the convolution operation
- Let us see what a full convolutional neural network looks like



Input

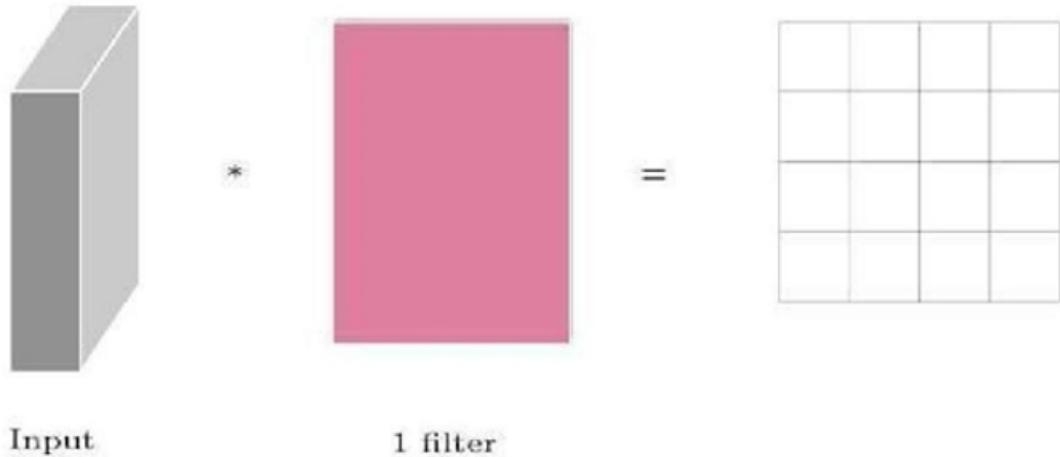


Input

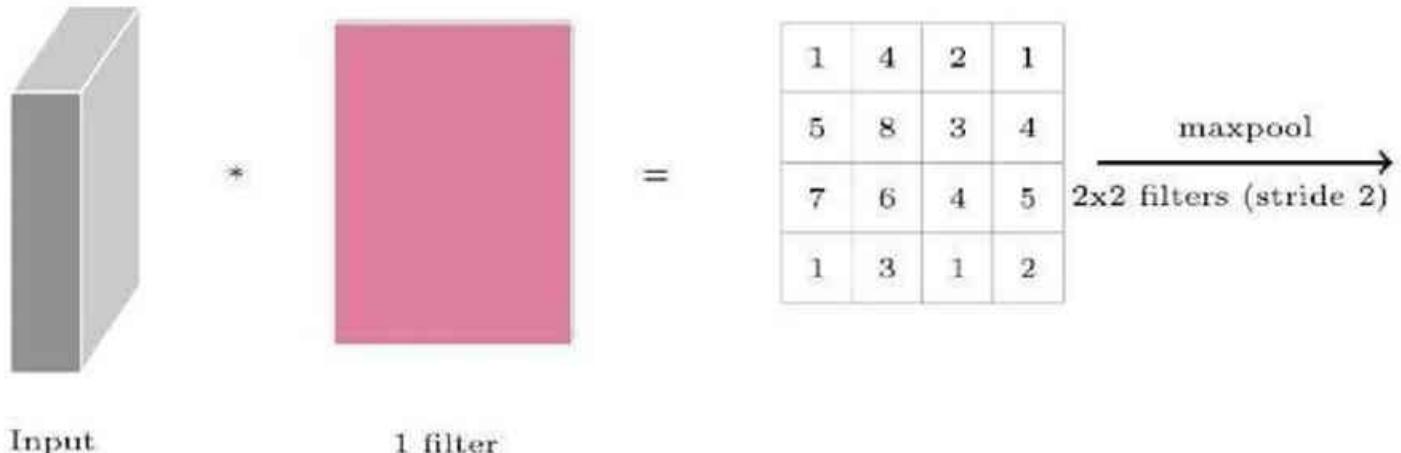


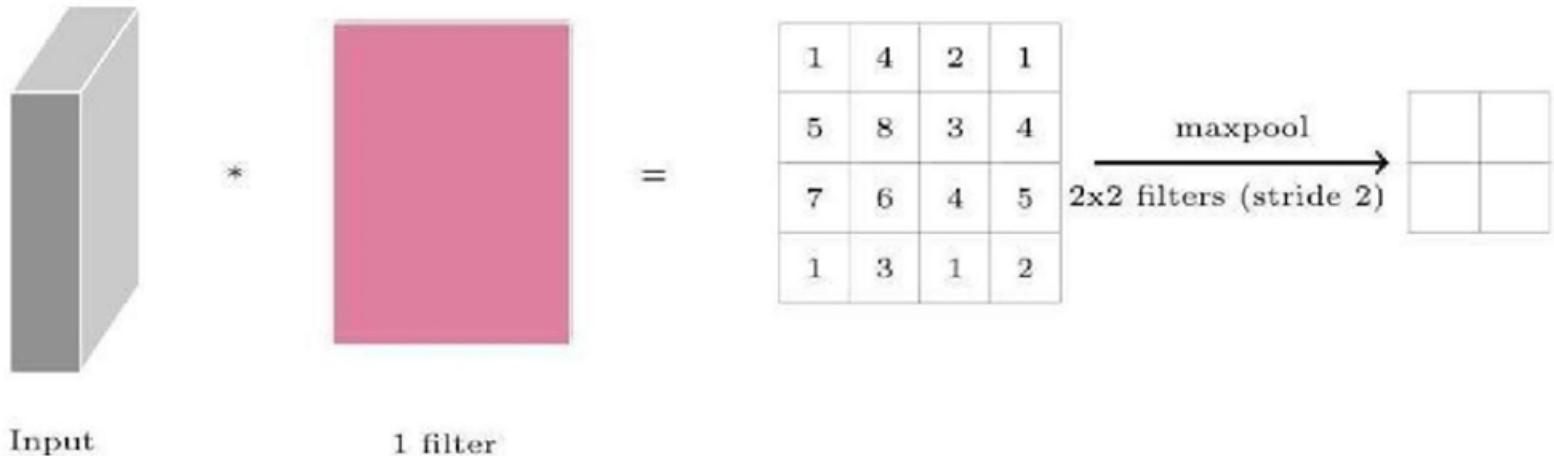
*

1 filter

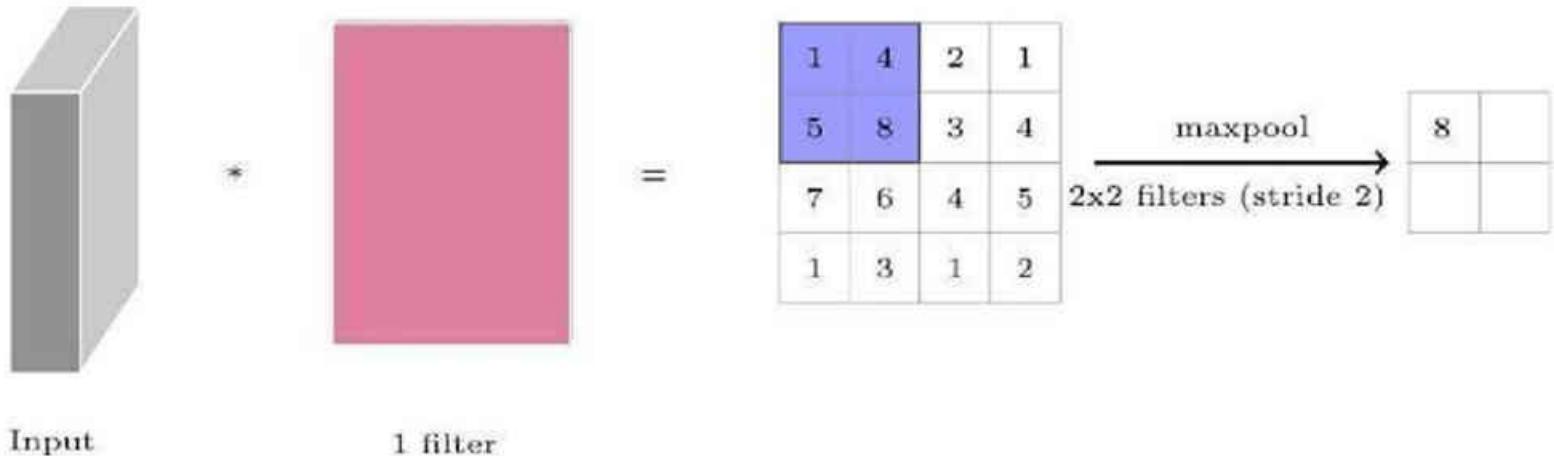


$$\text{Input} \quad * \quad \text{1 filter} = \begin{array}{|c|c|c|c|} \hline 1 & 4 & 2 & 1 \\ \hline 5 & 8 & 3 & 4 \\ \hline 7 & 6 & 4 & 5 \\ \hline 1 & 3 & 1 & 2 \\ \hline \end{array}$$

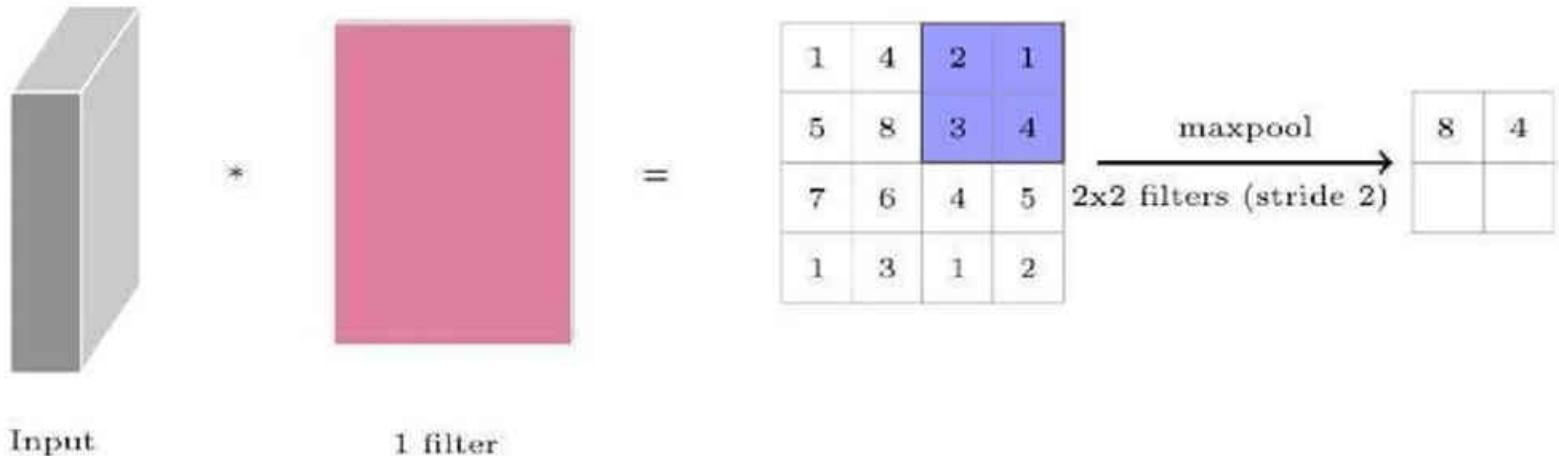




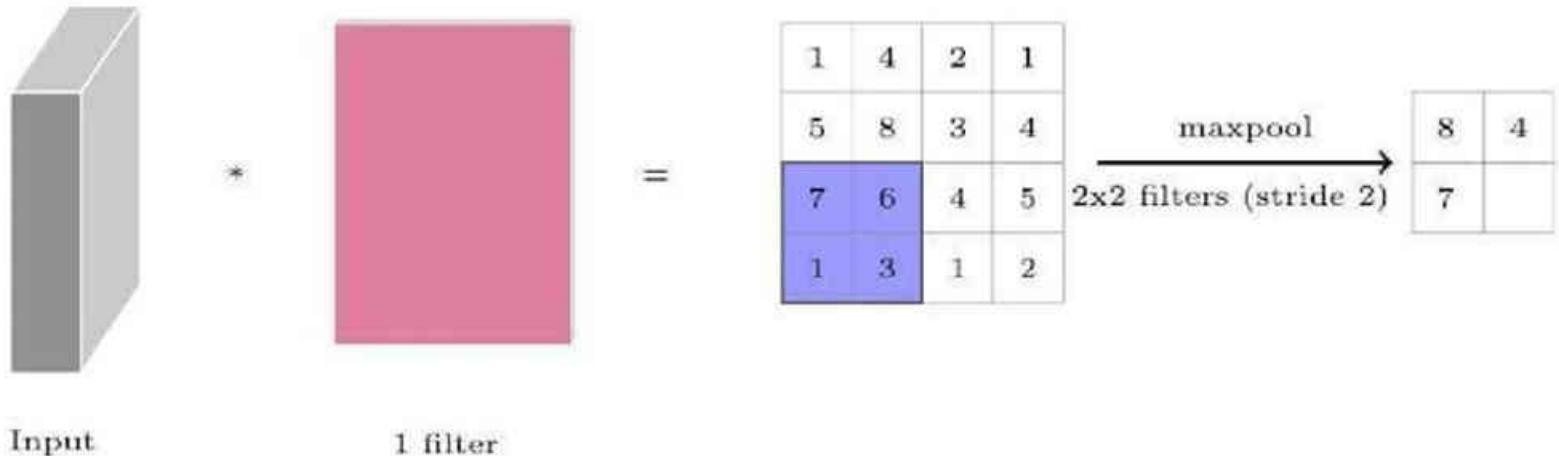
CNN



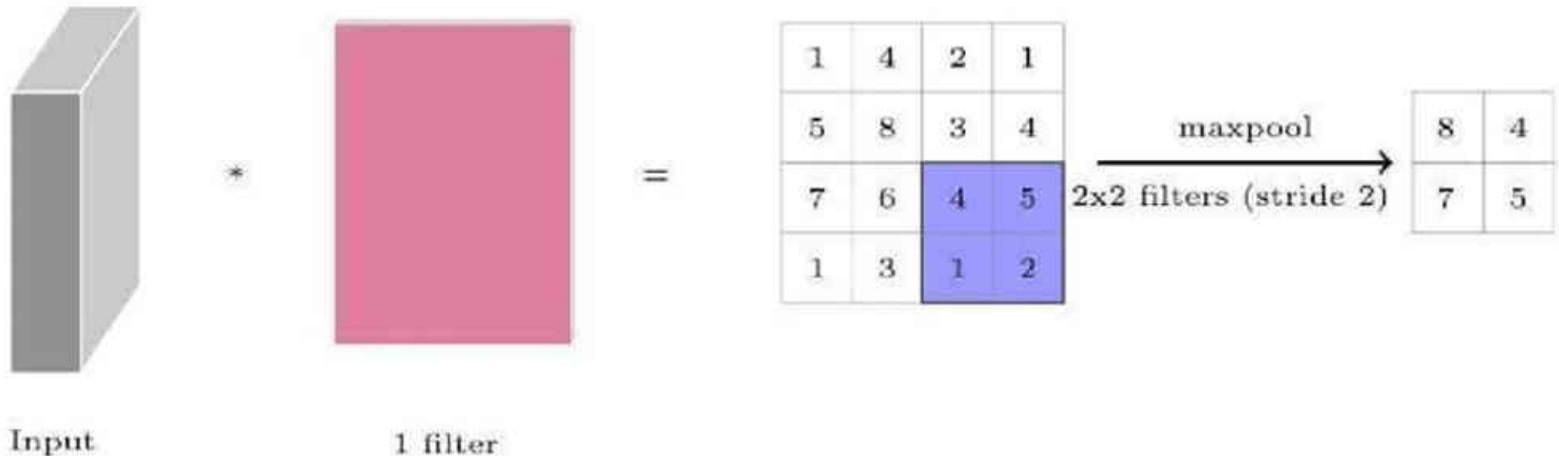
CNN



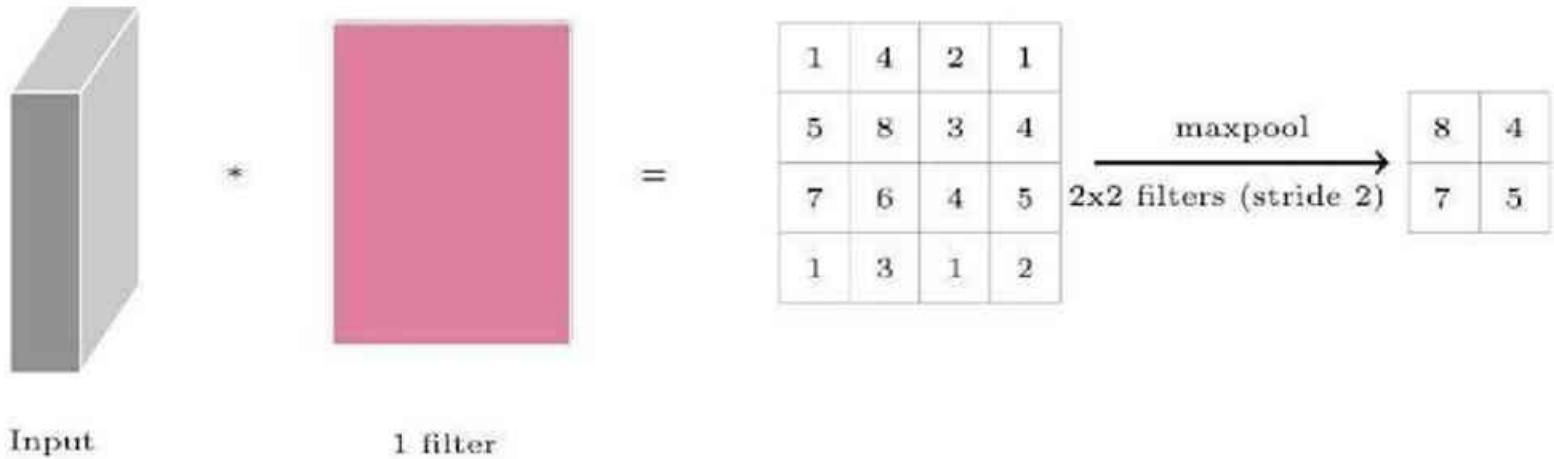
CNN



CNN

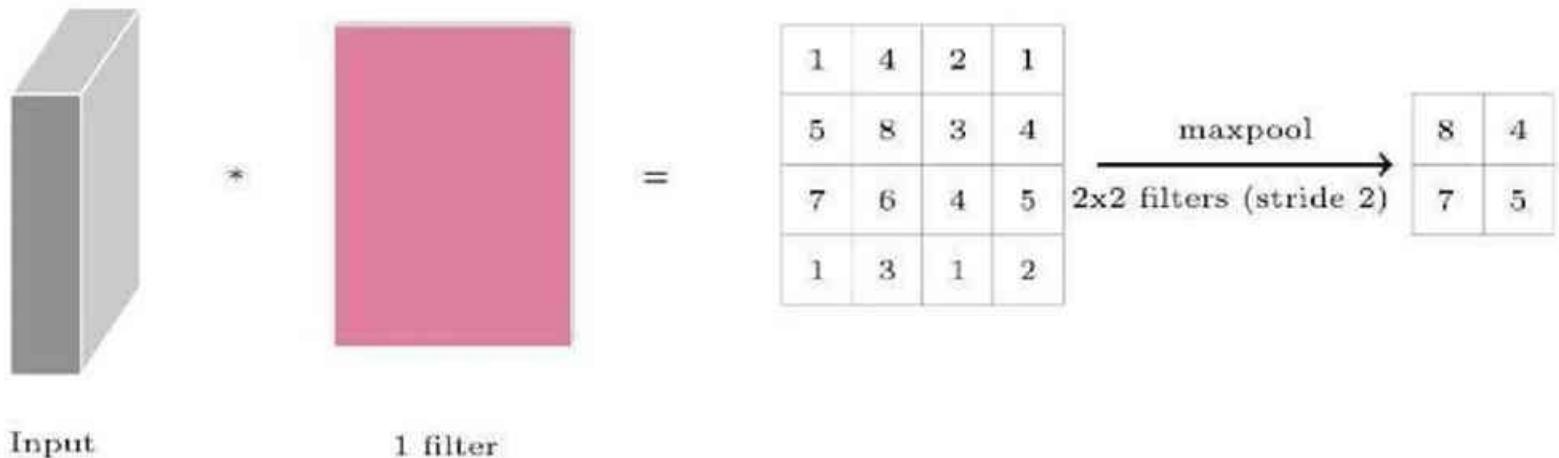


CNN



| | | | |
|---|---|---|---|
| 1 | 4 | 2 | 1 |
| 5 | 8 | 3 | 4 |
| 7 | 6 | 4 | 5 |
| 1 | 3 | 1 | 2 |

CNN

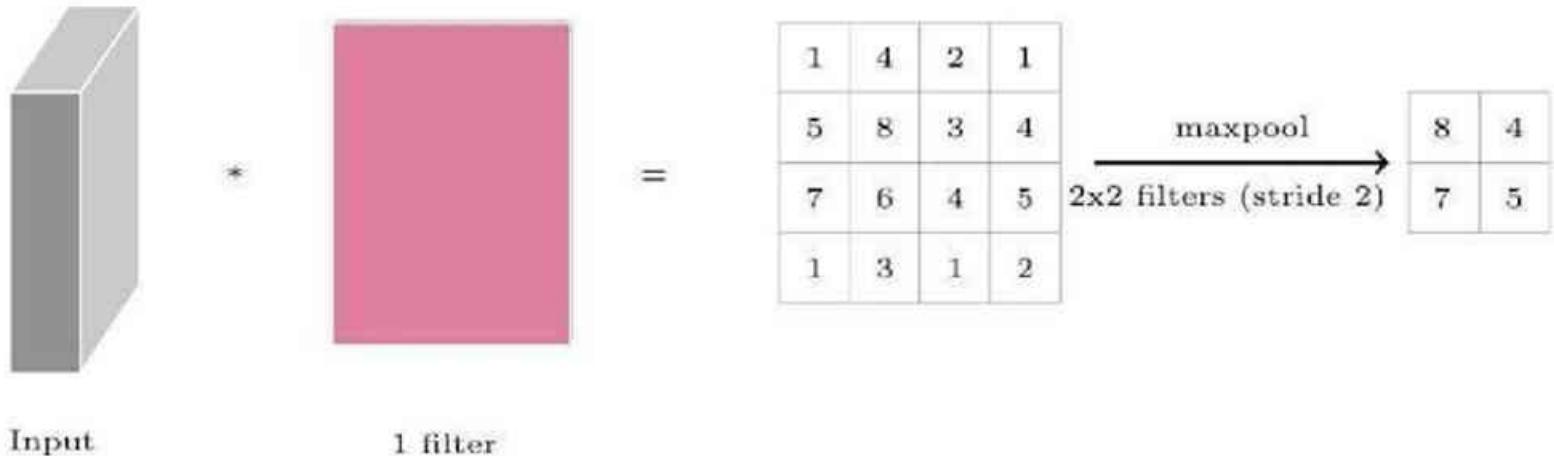


maxpool →

2x2 filters (stride 1)

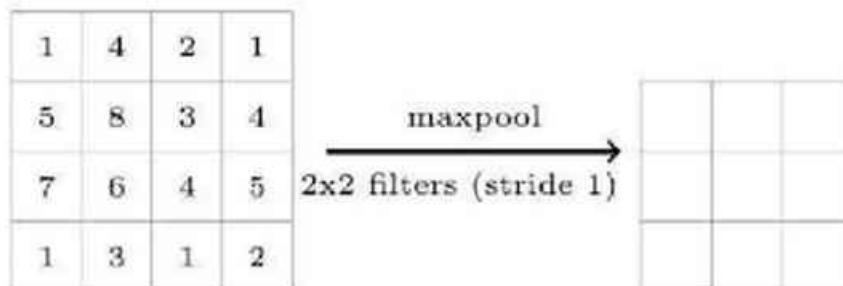
| | | | |
|---|---|---|---|
| 1 | 4 | 2 | 1 |
| 5 | 8 | 3 | 4 |
| 7 | 6 | 4 | 5 |
| 1 | 3 | 1 | 2 |

CNN

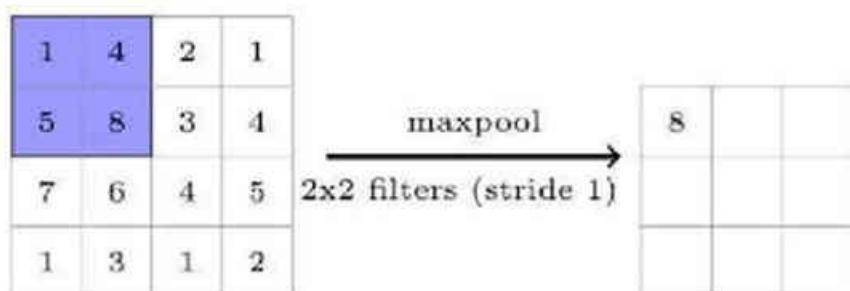
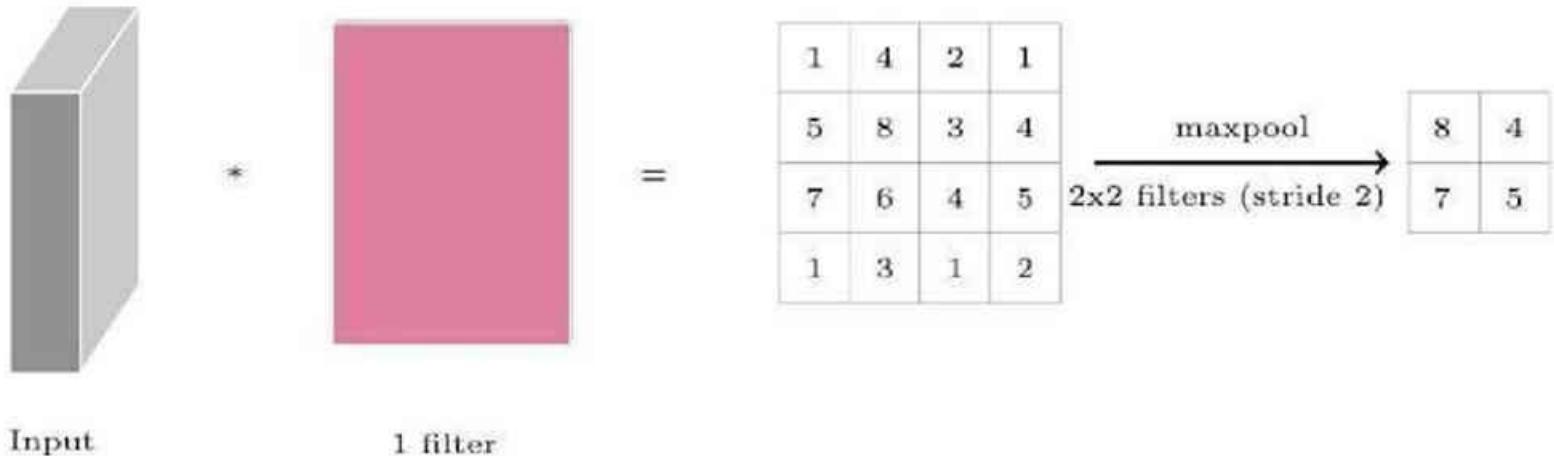


Input

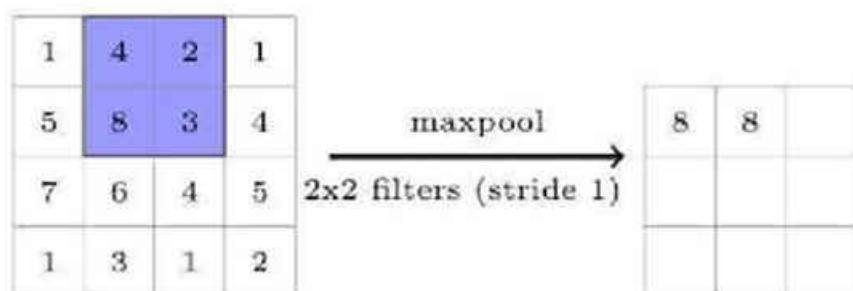
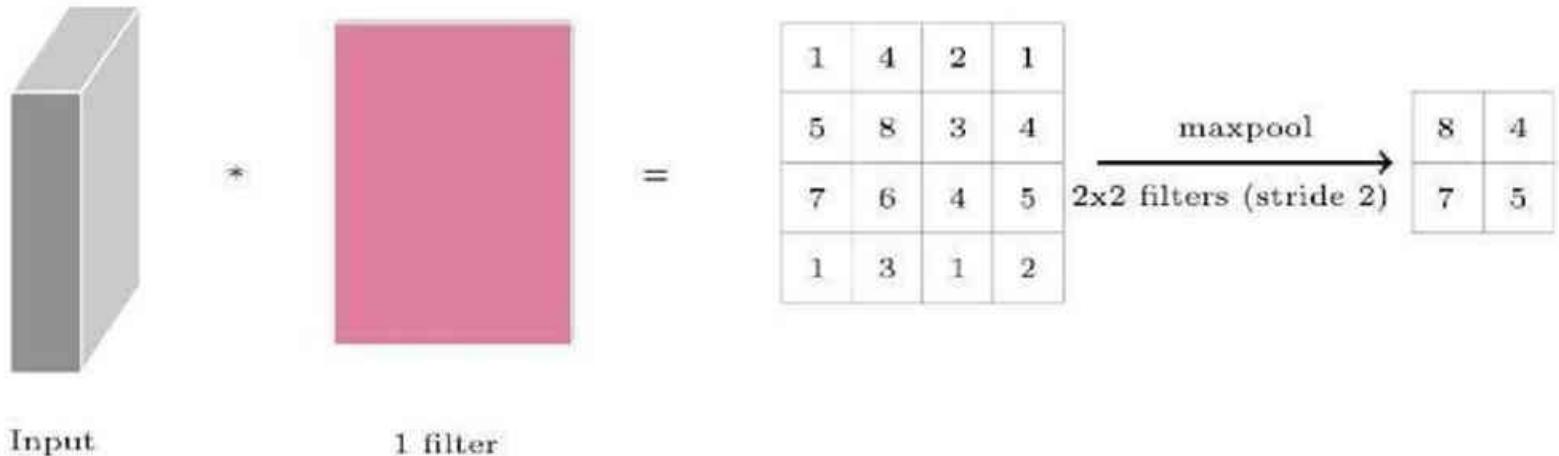
1 filter



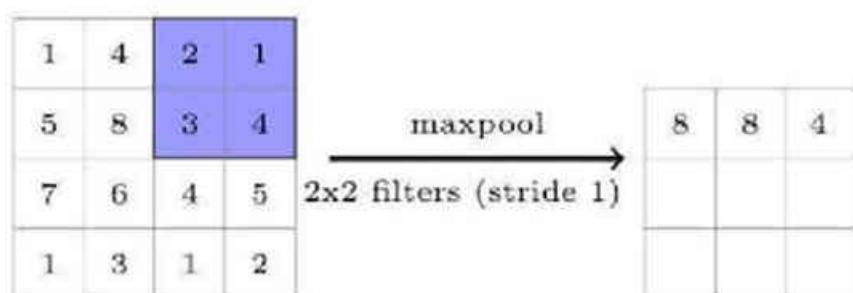
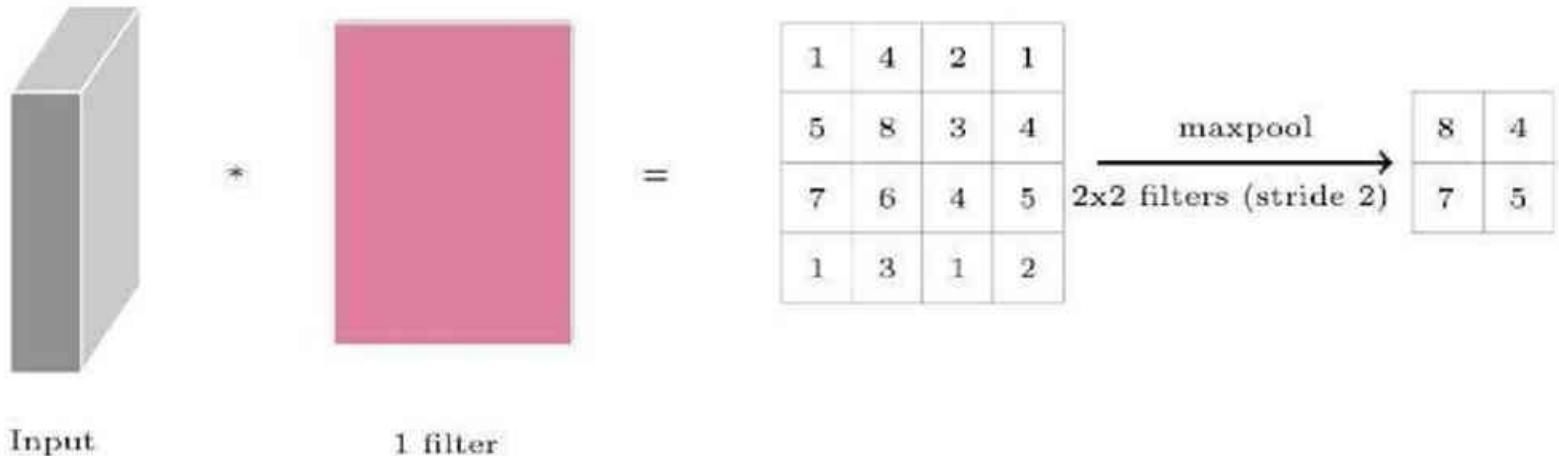
CNN



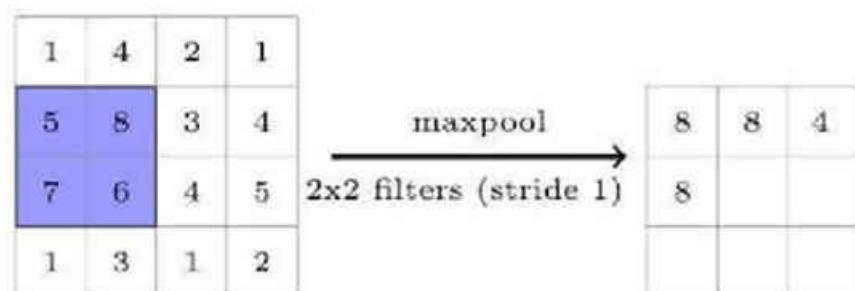
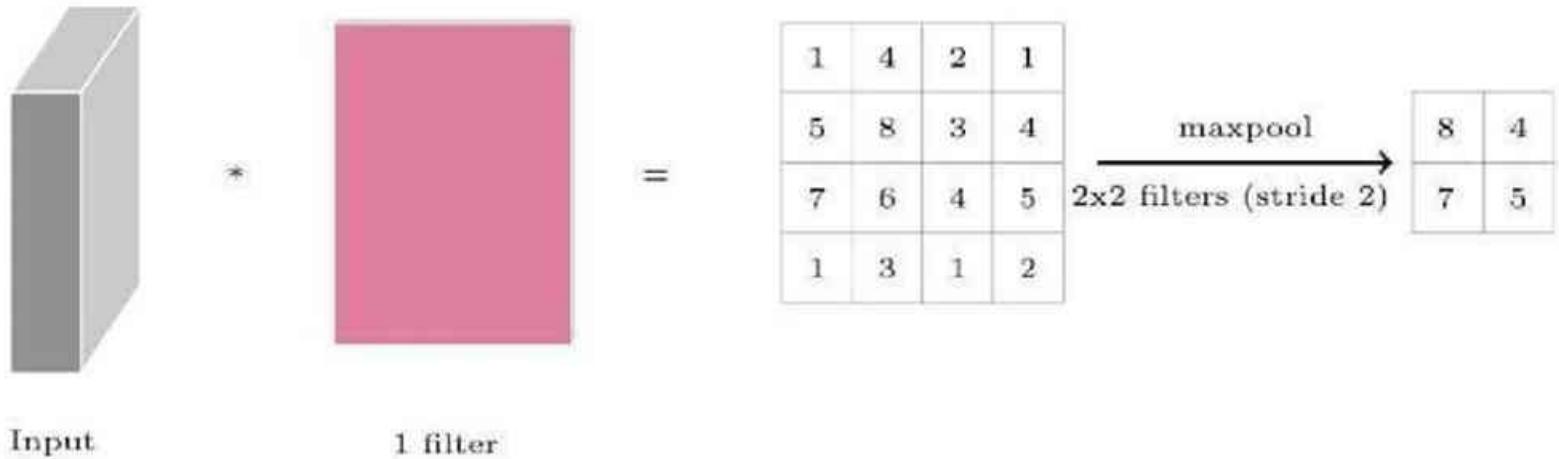
CNN

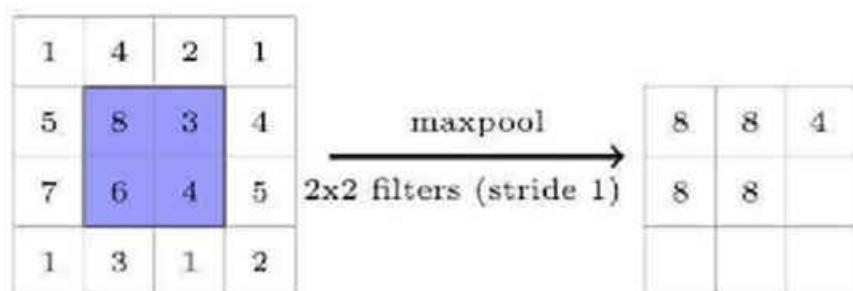
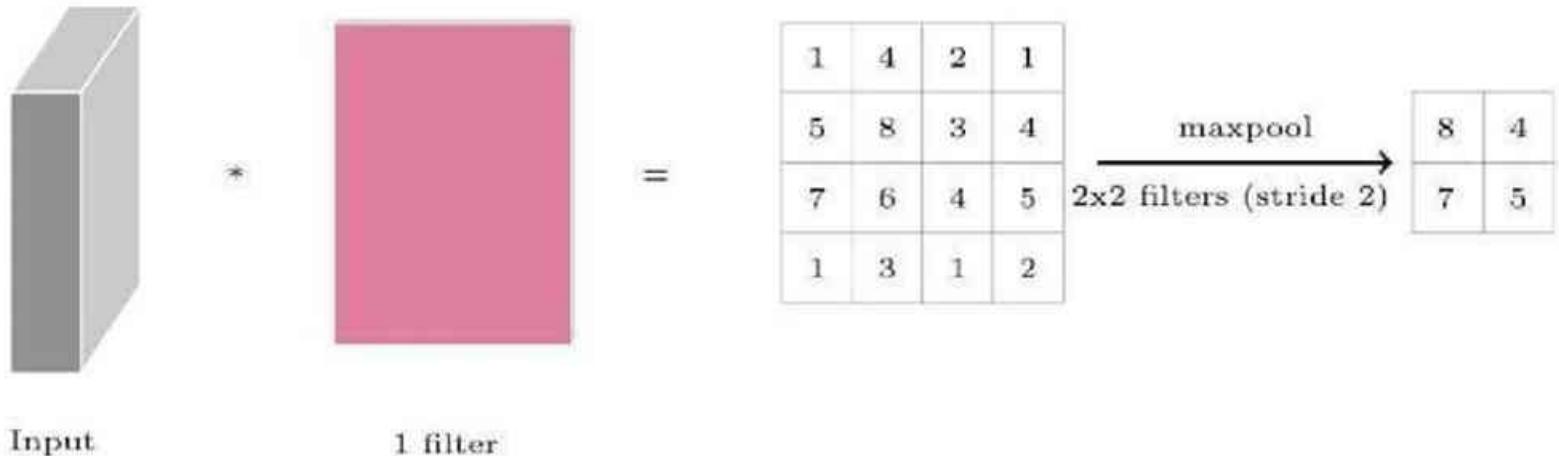


CNN

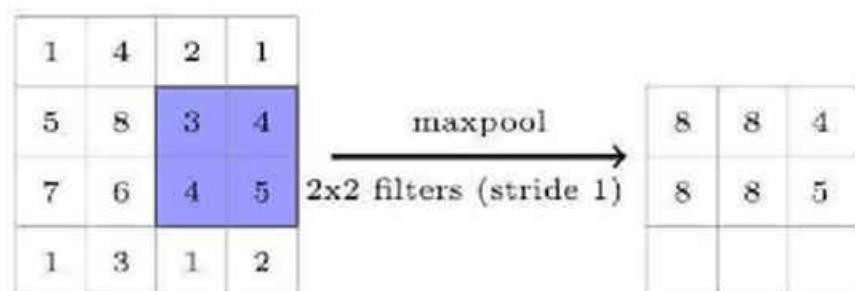
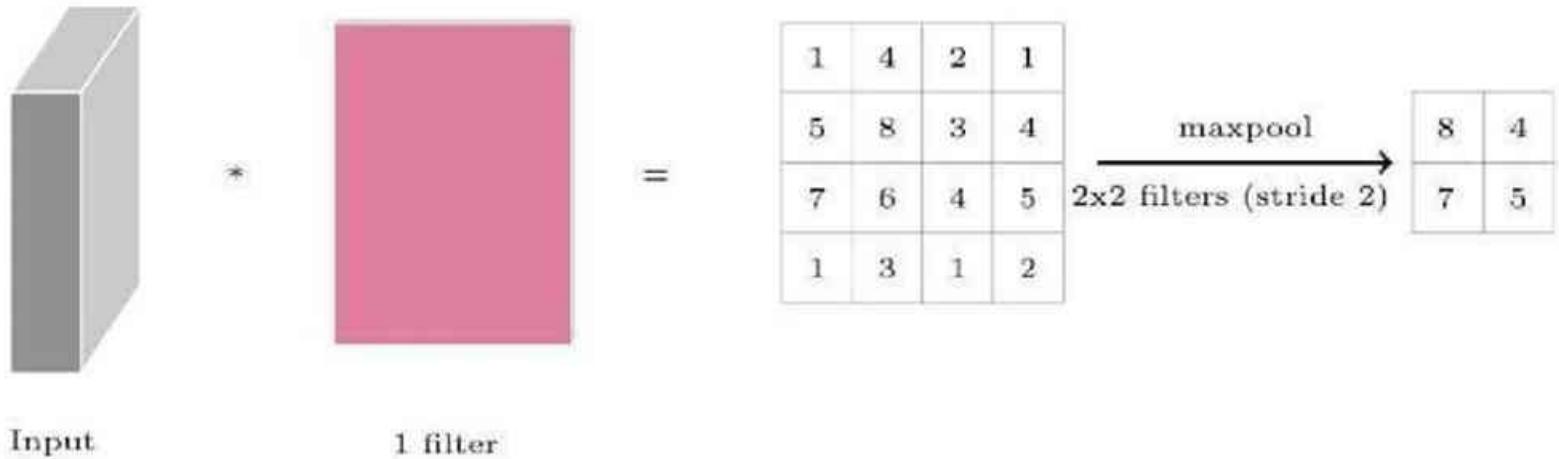


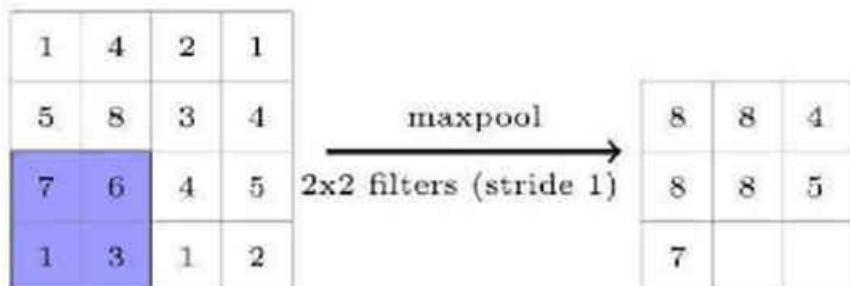
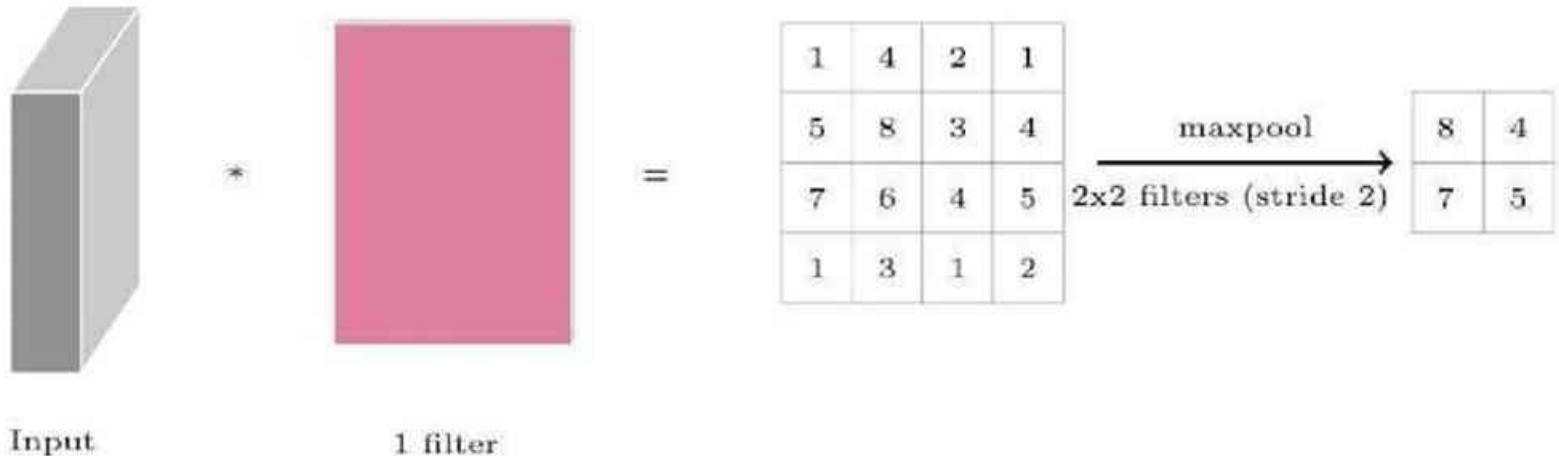
CNN

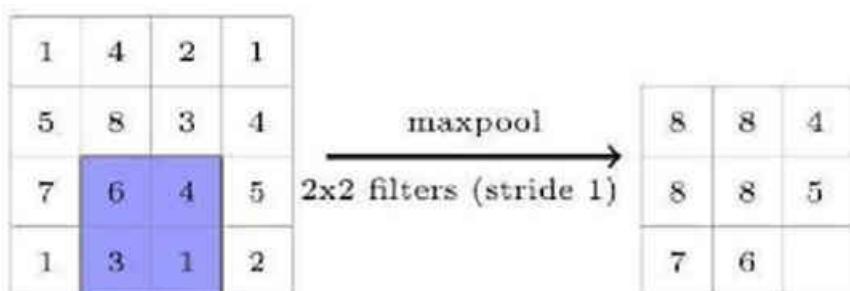
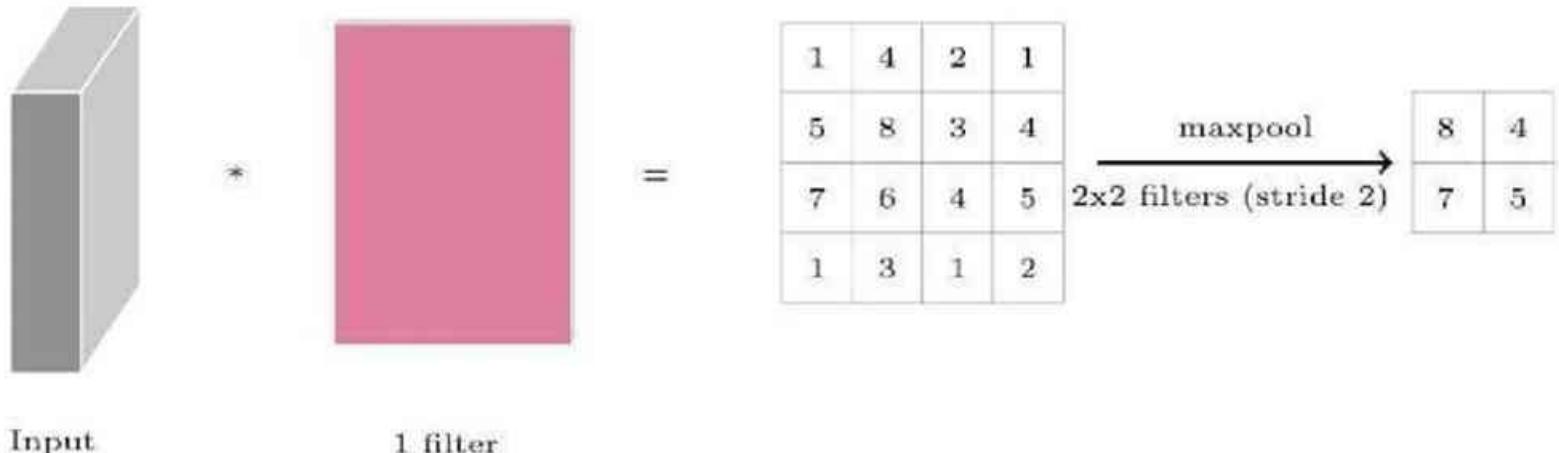




CNN

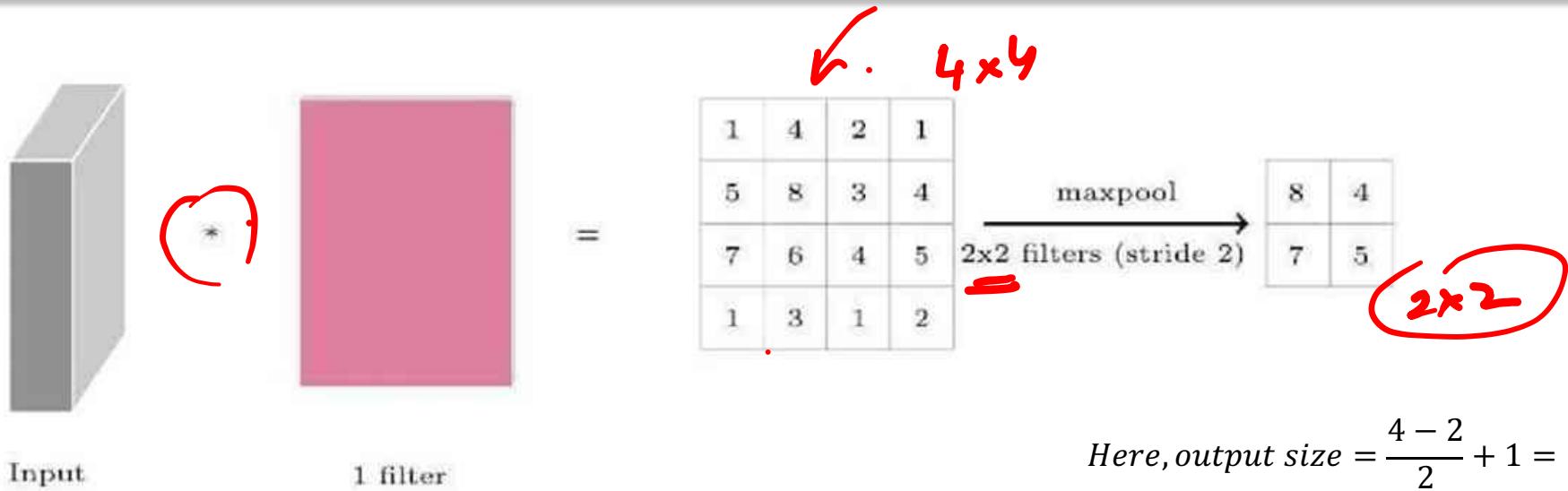






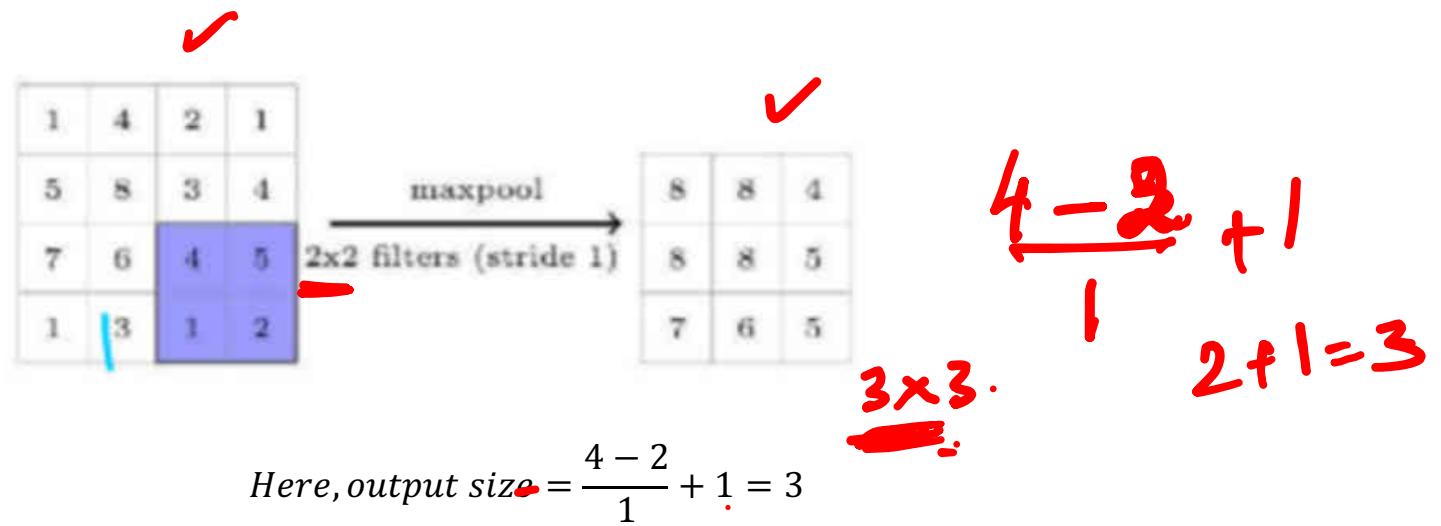
CNN

$$\frac{4-2}{2} + 1 = 1+1=2$$



Input size has been reduced by performing max pooling in this case

$$\checkmark \quad \text{Output} = \frac{\text{size of input} - \text{Pool filter size}}{\text{stride}} + 1$$



How many parameters do we have in the max pool layers

- Zilch – It does not need to learn anything
- So what will be the output size $O = \frac{I - P_s}{S} + 1$

O = Size (width) of output image.

I = Size (width) of input image.

S = Stride of the convolution operation.

P_s = Pool size.

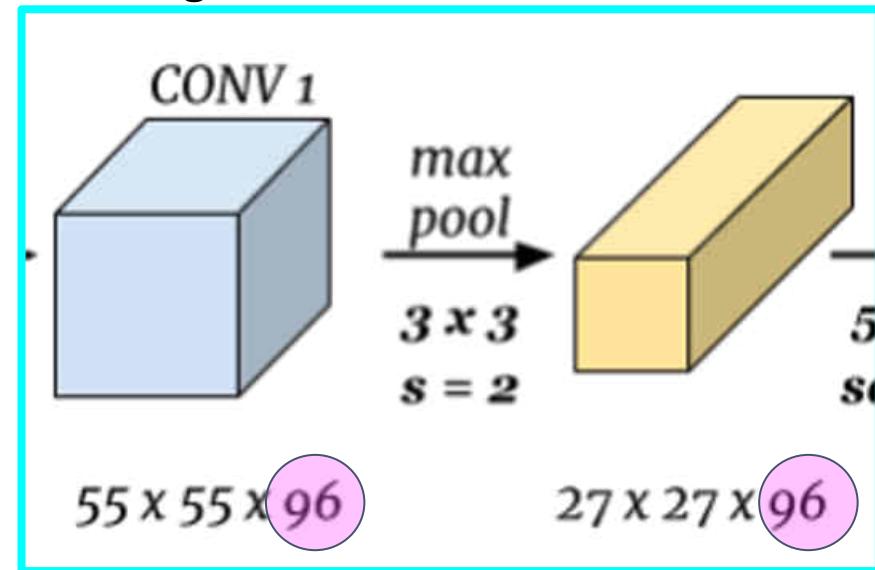
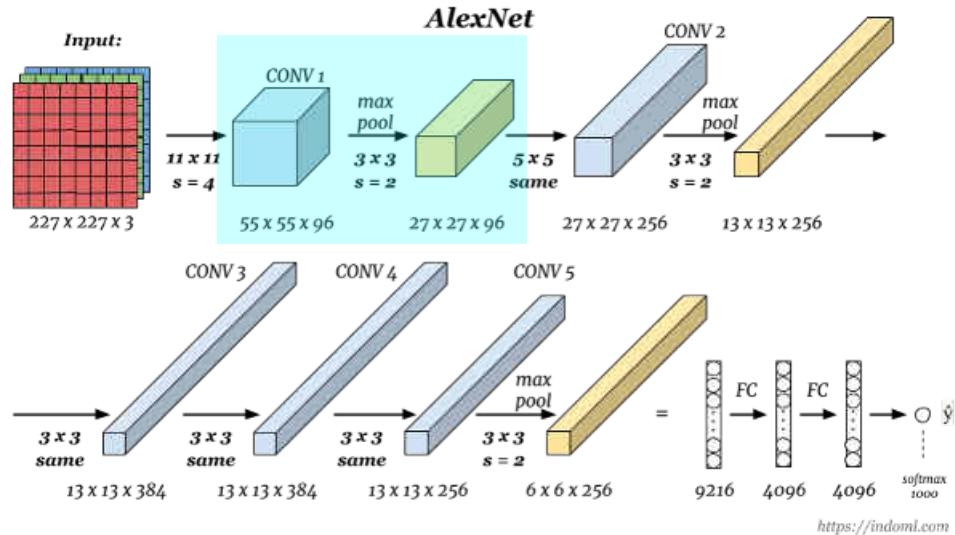
- So is the max taken across all the kernels
- No, It is taken in the same spatial dimension
- That's the reason why we have $k(\text{maxpool}) = k(\text{prev_conv_layer})$

The size (O) of the output image is given by

$$O = \frac{I - P_s}{S} + 1$$

Numericals

Let's revisit AlexNet and work out some numericals from this image



$$\text{Output size} = \frac{\text{Input size} - \text{Pool filter size}}{\text{Stride}} + 1$$

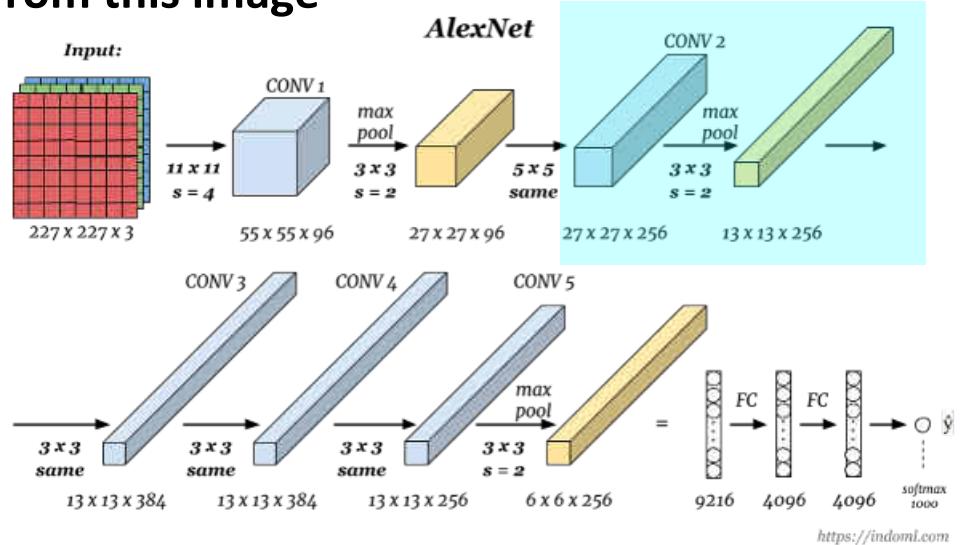
Here let us look into the first max pooling layer
 And the depth = depth of the previous convolution layer = 96
 Output Size = $(55-3) + 1$

$$= \frac{26}{2} + 1 = 27$$

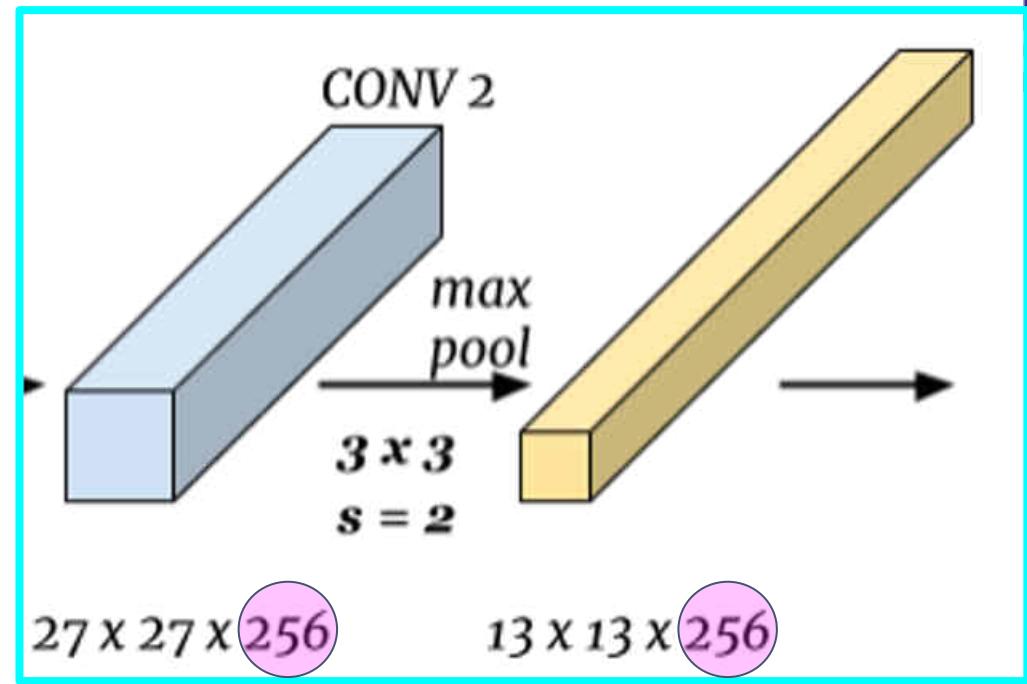
So the size is **$27 \times 27 \times 96$**

Numericals

Let's revisit AlexNet and work out some numericals from this image



$$\text{Output size} = \frac{\text{Input size} - \text{Pool filter size}}{\text{Stride}} + 1$$



Here let us look into the second max pooling layer
 And the depth = depth of the previous convolution layer = 256
 Output Size = $\frac{(27-3)}{2} + 1$

$$= \frac{24}{2} + 1 = 12 + 1 = 13$$

So the size is **13 x 13 x 256**

Calculating number of parameters

W_c = Number of weights of the Conv Layer.

B_c = Number of biases of the Conv Layer.

P_c = Number of parameters of the Conv Layer.

K = Size (width) of kernels used in the Conv Layer.

N = Number of kernels.

C = Number of channels of the input image.

$$W_c = K^2 \times C \times N$$

$$B_c = N$$

$$P_c = W_c + B_c$$

Numericals

Parameter calculation:

$$W_c = K^2 \times C \times N$$

$$B_c = N$$

$$P_c = W_c + B_c$$

W_c = Number of weights of the Conv Layer.

B_c = Number of biases of the Conv Layer.

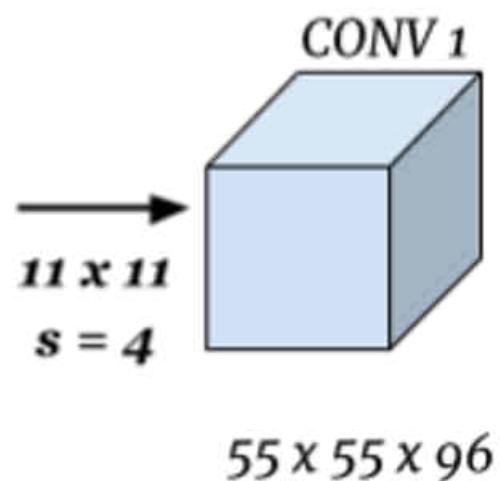
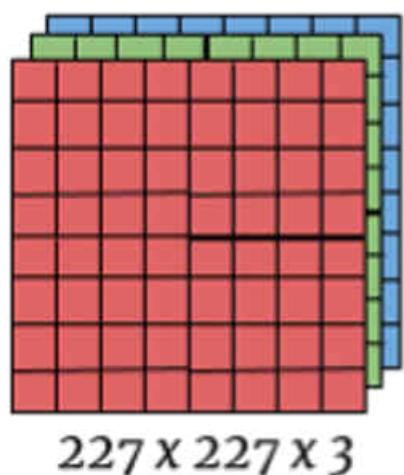
P_c = Number of parameters of the Conv Layer.

K = Size (width) of kernels used in the Conv Layer.

N = Number of kernels.

C = Number of channels of the input image.

Input:



$$W_c = (11 \times 11) \times 3 \times 96 = 34,848$$

$$B_c = N = 96$$

Total number of parameters $P_c = W_c + B_c$

$$P_c = 34,848 + 96 = 34,944$$

Numericals

Parameter calculation:

$$W_c = K^2 \times C \times N$$

$$B_c = N$$

$$P_c = W_c + B_c$$

W_c = Number of weights of the Conv Layer.

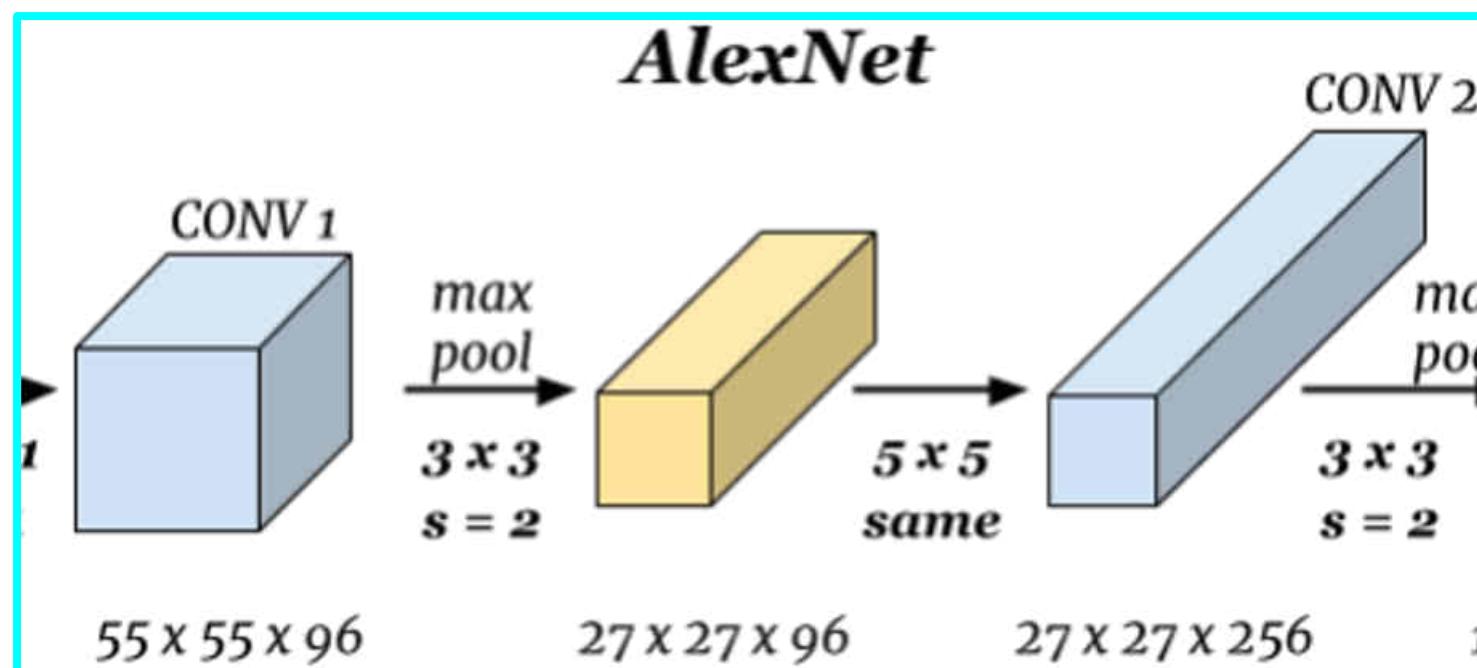
B_c = Number of biases of the Conv Layer.

P_c = Number of parameters of the Conv Layer.

K = Size (width) of kernels used in the Conv Layer.

N = Number of kernels.

C = Number of channels of the input image.



$$W_c = (5 \times 5) \times 96 \times 256 = 614,400$$

$$B_c = N = 256$$

Total number of parameters $P_c = W_c + B_c$

$$P_c = 614,400 + 256 = 614,656$$

Numerical on parameter calculation

Parameter calculation:

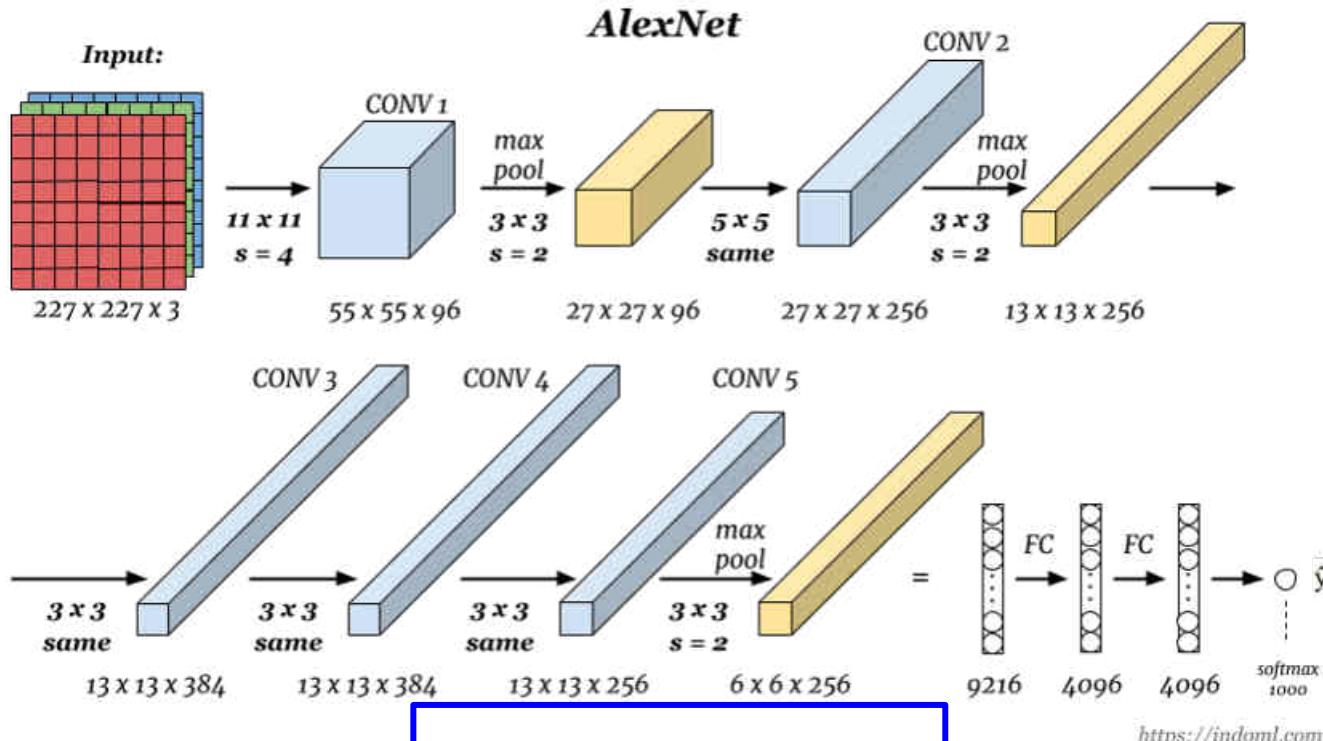
Calculate the parameters to the given information:

Convolution Layer 1 has dimensions of $55 \times 55 \times 96$ which on being passed through a max pool layer of 3×3 filter size and stride as 2 leads to Convolution layer 2 with dimensions $27 \times 27 \times 256$, on using a kernel of dimensions 5×5

Calculate the **number of parameters in the convolution layers** (which means you will have to calculate Number of weights and Number of biases of the convolution Layer)

Numericals

Parameter calculation: The rest of the network is an exercise to understand better



$$W_c = K^2 \times C \times N$$

$$B_c = N$$

$$P_c = W_c + B_c$$

| Layer Name | Tensor Size | Weights | Biases | Parameters |
|--------------|---------------------------|------------|--------|-------------------|
| Input Image | $227 \times 227 \times 3$ | 0 | 0 | 0 |
| Conv-1 | $55 \times 55 \times 96$ | 34,848 | 96 | 34,944 |
| MaxPool-1 | $27 \times 27 \times 96$ | 0 | 0 | 0 |
| Conv-2 | $27 \times 27 \times 256$ | 614,400 | 256 | 614,656 |
| MaxPool-2 | $13 \times 13 \times 256$ | 0 | 0 | 0 |
| Conv-3 | $13 \times 13 \times 384$ | 884,736 | 384 | 885,120 |
| Conv-4 | $13 \times 13 \times 384$ | 1,327,104 | 384 | 1,327,488 |
| Conv-5 | $13 \times 13 \times 256$ | 884,736 | 256 | 884,992 |
| MaxPool-3 | $6 \times 6 \times 256$ | 0 | 0 | 0 |
| FC-1 | 4096×1 | 37,748,736 | 4,096 | 37,752,832 |
| FC-2 | 4096×1 | 16,777,216 | 4,096 | 16,781,312 |
| FC-3 | 1000×1 | 4,096,000 | 1,000 | 4,097,000 |
| Output | 1000×1 | 0 | 0 | 0 |
| Total | | | | 62,378,344 |

Pooling gives translation invariance

- They are useful as small changes in the location of the feature in the input detected by the convolutional layer will result in a **pooled feature map with the feature in the same location.**
- This capability added by pooling is called the model's invariance to local translation.



Advantages of the pooling layer

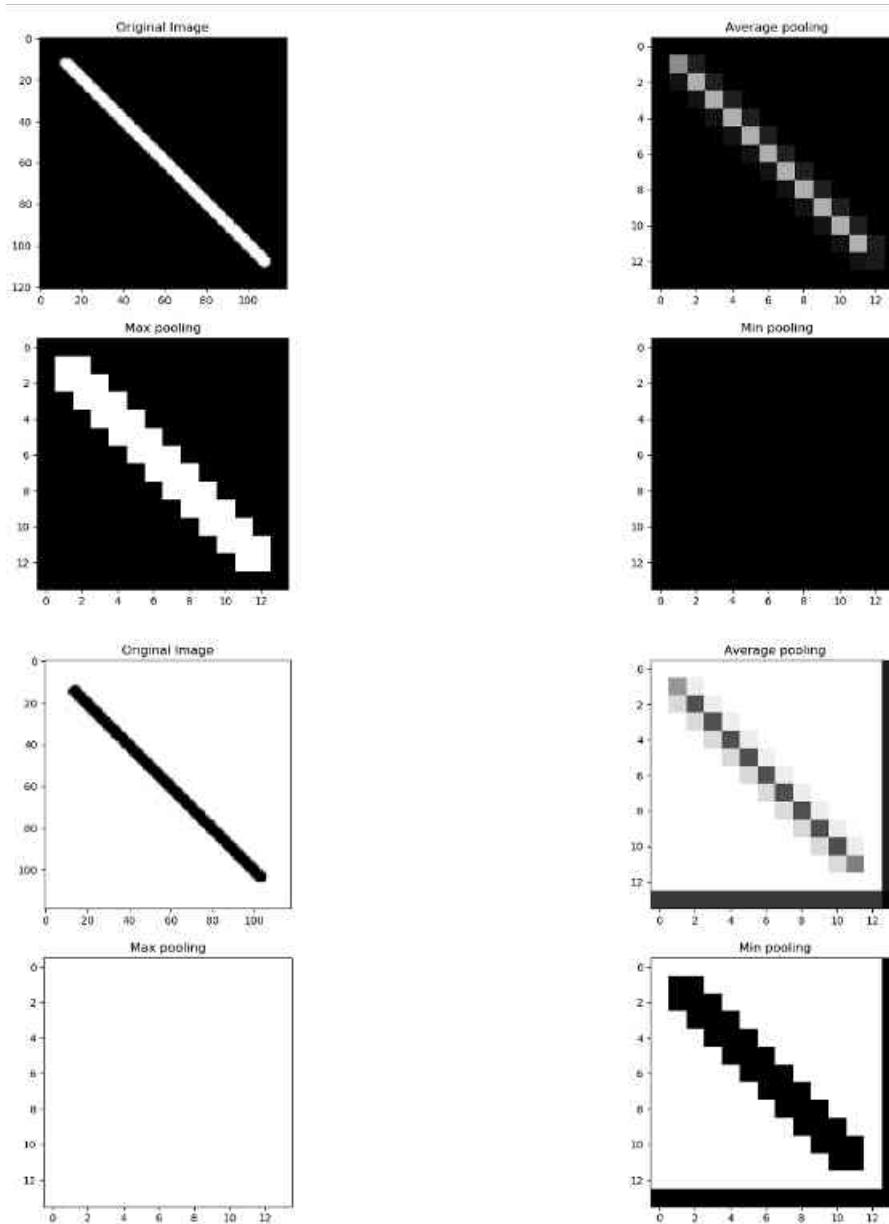
- Reduces the number of training parameters and computation cost, thus **control overfitting**.
- Adding **translation-invariance** to the feature maps
- To **save memory** during training: less memory allocation in the GPUs

Is the pooling layer trained during backpropagation?

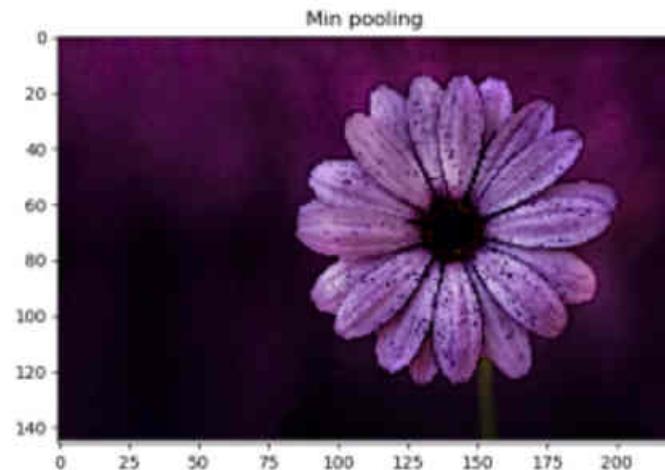
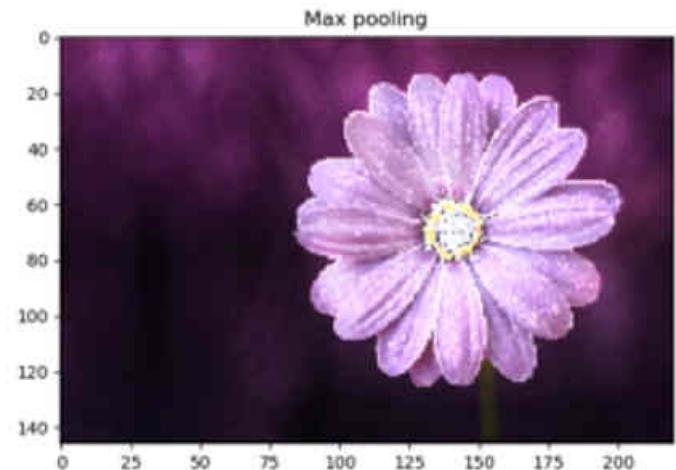
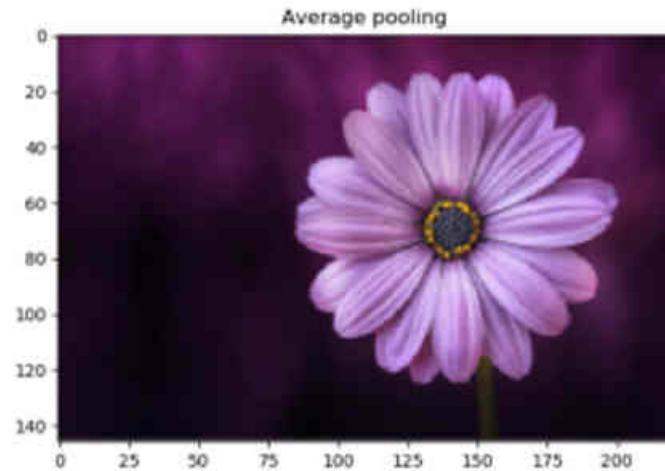
- The pooling layer is **not trained** during the backpropagation of gradients because the output volume of data depends on the values of the input volume of data.
- Pooling layers **does not have parameters which affect the back propagation.**
- **Adding pooling layers does not change the number of trainable parameters** (parameters that affect backpropagation include bias and weights in convolution filters).

When to use which pooling method?

- We cannot say that a particular pooling method is better over other generally. The choice of pooling operation is made based on the data at hand.
- **Average pooling method** smooths out the image and hence the sharp features may not be identified when this pooling method is used.
- **Max pooling** selects the brighter pixels from the image. It is useful when the background of the image is dark and we are interested in only the lighter pixels of the image. For example: in MNIST dataset, the digits are represented in white color and the background is black. So, max pooling is used.
- Similarly, **min pooling** is used in the other way round.



On a image when applied different pooling techniques



QUIZ

What is the role of pooling layers in a CNN?

Allow flexibility in the location of certain features in the image, therefore allowing non-rigid or rotated or scaled objects to be recognized,

they reduce the output image size.

they allow unions of features to be computed

they reduce the output image size.

QUIZ

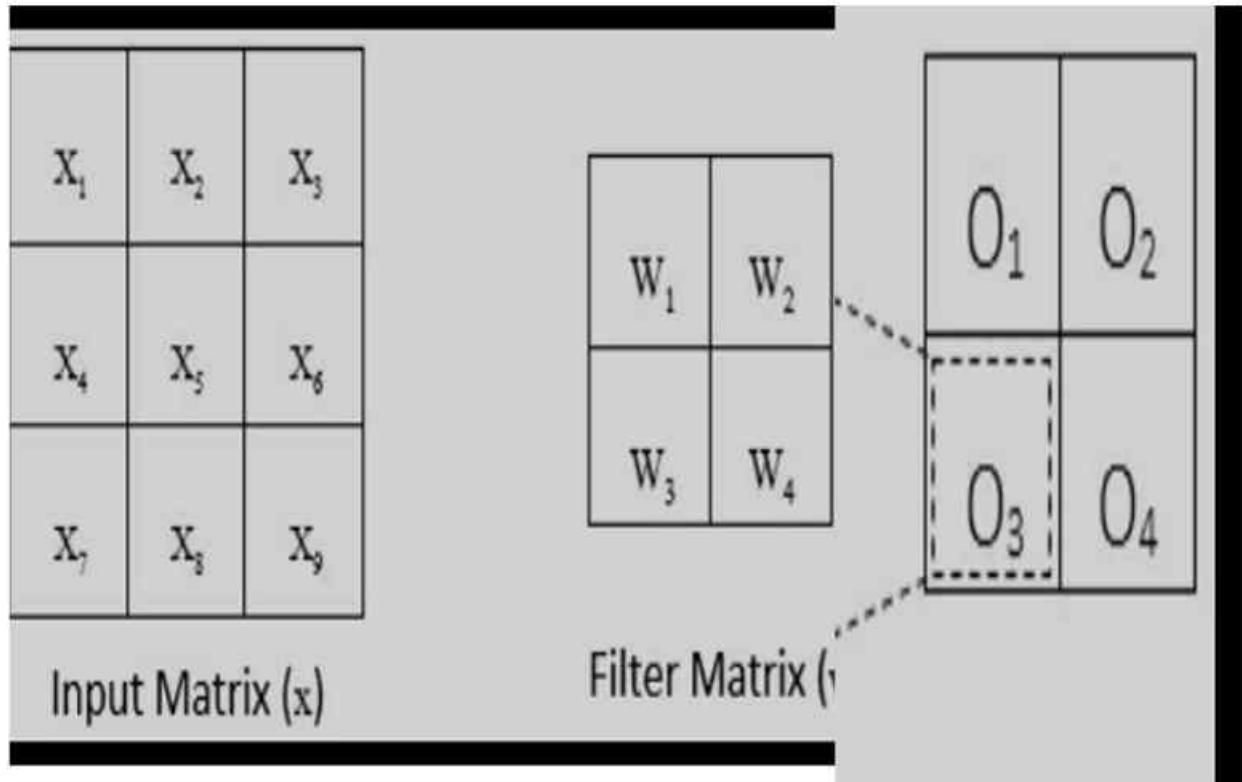
What is the role of pooling layers in a CNN?

Allow flexibility in the location of certain features in the image, therefore allowing non-rigid or rotated or scaled objects to be recognized,

they reduce the output image size.

they allow unions of features to be computed

CNN



Forward Propagation

[\(2\) Forward and Back Propagation over a CNN... code from Scratch!!](#)
[LinkedIn](#)

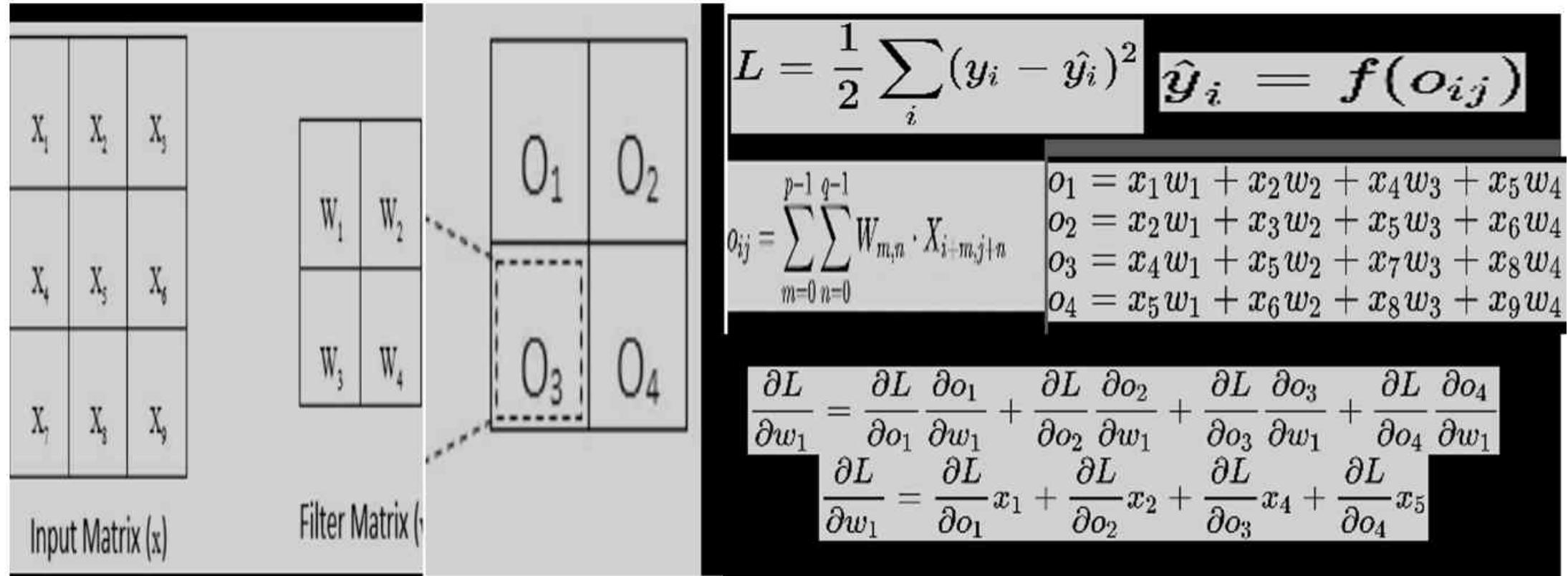
$$\begin{aligned} o_1 &= x_1 w_1 + x_2 w_2 + x_4 w_3 + x_5 w_4 \\ o_2 &= x_2 w_1 + x_3 w_2 + x_5 w_3 + x_6 w_4 \\ o_3 &= x_4 w_1 + x_5 w_2 + x_7 w_3 + x_8 w_4 \\ o_4 &= x_5 w_1 + x_6 w_2 + x_8 w_3 + x_9 w_4 \end{aligned}$$

$$o_{ij} = \sum_{m=0}^{p-1} \sum_{n=0}^{q-1} W_{m,n} \cdot X_{i+m, j+n}$$

$$\hat{y}_i = f(o_{ij})$$

$$L = \frac{1}{2} \sum_i (y_i - \hat{y}_i)^2$$

CNN



Backpropagation

Batch normalization

Machine Intelligence

Batch Normalization

- **Normalization is a pre-processing technique used to standardize data.**
- In other words, having different sources of data inside the same range. Not normalizing the data before training can cause problems in our network, making it drastically harder to train and decrease its learning speed.
- Let's suppose, we want to predict a fair price for each car based on competitors' data.
- We have two features per car, the age in years and the total amount of kilometers it has been driven for.
- These can have very different ranges, ranging from 0 to 30 years, while distance could go from 0 up to 1000s of kilometers.
- We don't want features to have these differences in ranges, as the value with the higher range might bias our models into giving them inflated importance.

- The other technique used to normalize data is forcing the data points to have a mean of 0 and a standard deviation of 1, using the following formula.

$$x_{normali} = \frac{x - m}{s}$$

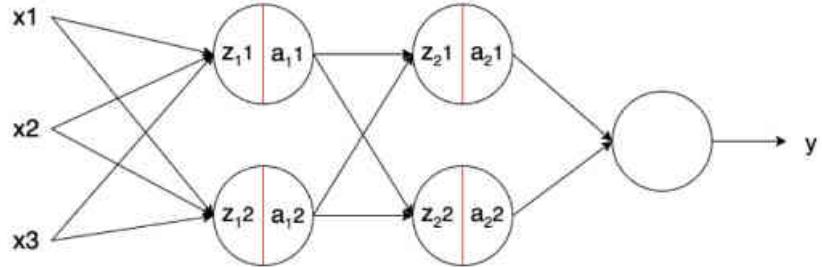
- x is the data point to normalize.
- m is the mean of the data set.
- s the standard deviation of the data set.
- Now, each data point mimics a standard normal distribution. Having all the features on this scale, none of them will have a bias, and therefore, our models will learn better.
- We use this to normalize batches of data inside the network itself.

- Batch Norm is a normalization technique done between **the layers of a Neural Network instead of in the raw data.**
- It is done along **mini-batches instead of the full data set**. It serves to speed up training and use higher learning rates, making learning easier.

$$z^N = \left(\frac{z - m_z}{s_z} \right)$$

- m_z is the **mean of the neurons output**.
- s_z is the **standard deviation of the neurons output**.

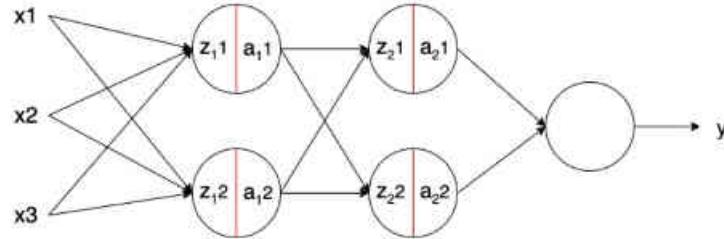
Batch normalization



- Usually, a neuron without Batch Norm would be computed as follows:

$$z = g(w, x) + b; \quad a = f(z)$$

- g is the linear transformation of the neuron.
- w is the weights of the neuron.
- b is the bias of the neurons.
- a is the activation function.
- The model learns the parameters w and b .



- Adding Batch Norm, it looks:

$$z = g(w, x); \quad z^N = \left(\frac{z - m_z}{s_z} \right) \cdot \gamma + \beta; \quad a = f(z^N)$$

- z^N is the output of Batch Norm.
- m_z is the mean of the neurons' output.
- s_z is the standard deviation of the output of the neurons.
- gamma and beta are the learning parameters of Batch Norm.
- Note that the bias (b) of the neurons is removed.
- This is because as we subtract the mean m_z , any constant over the values of, such as b , can be ignored as it will be subtracted by itself.

$$z = g(w, x); \quad z^N = \left(\frac{z - m_z}{s_z} \right) \cdot \gamma + \beta; \quad a = f(z^N)$$

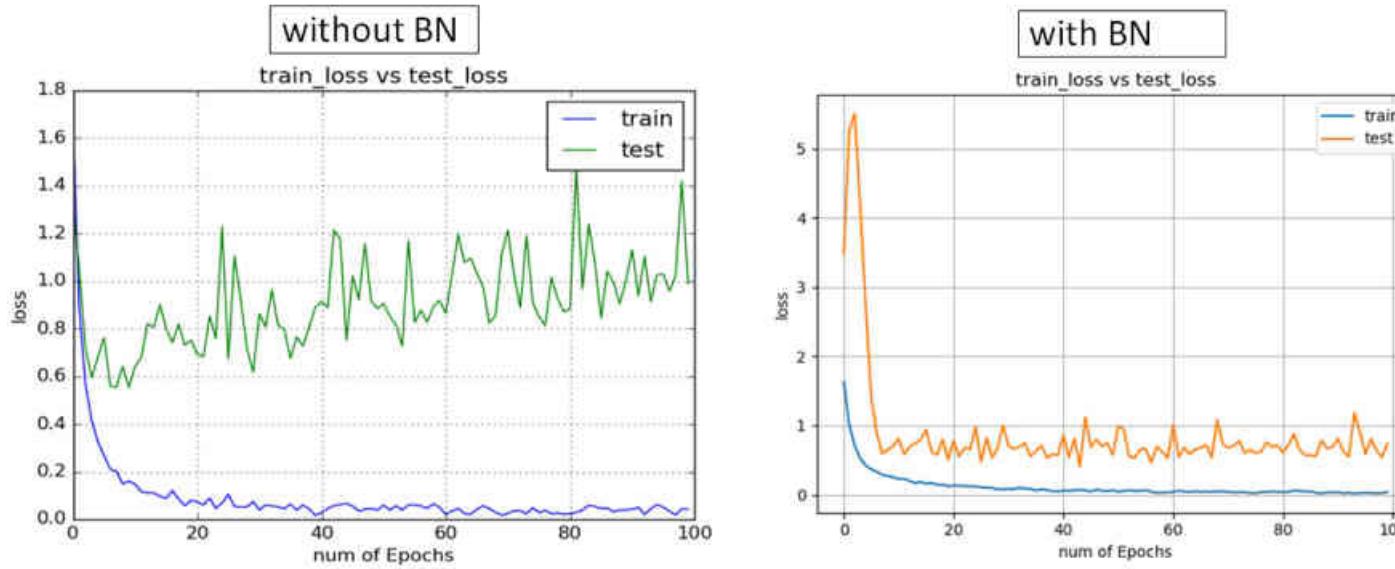
- The parameters γ & β **shift the mean and standard deviation** respectively.
- Thus, the outputs of Batch Norm over a layer results in a distribution with a mean beta and standard deviation gamma.
- These values are learned over epochs and the other learning parameters, such as the weights of the neurons, aiming to decrease the loss of the model.

Batch normalization in CNN

- Batch Norm **works in a very similar way in CNNs as it does in feed-forward networks**. Although we could do it in the same way as before, we have to follow the convolutional property.
- **In convolutions, we have shared filters that go along the feature maps of the input** (in images, the feature map is generally the height and width).
- **These filters are the same on every feature map**. It is then reasonable to normalize the output, in the same way, sharing it over the feature maps.
- In other words, this means that the **parameters used to normalize are calculated along with each feature map**.
- In a regular Batch Norm, each feature would have a different mean and standard deviation. Here, **each feature map will have a single mean and standard deviation**, used on all the features it contains.

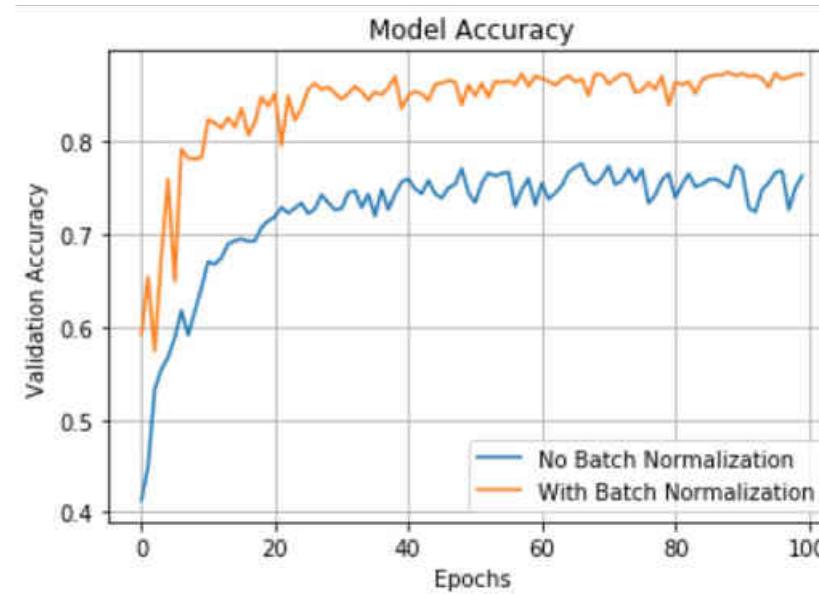
to address the vanishing/exploding gradients problems,
and more generally the problem that the distribution of each layer's inputs changes
during training, as the parameters of the previous layers change

Loss with and without Batch normalization



As we can see from the above curve, use of batch normalization gave the similar loss value of close to zero for training data but the test / validation loss improves, learning gets stabilized and it converges to lower error value.

Model accuracy affected by Batch normalization



We can see that the validation accuracy increases when Batch normalization is applied.

Dropouts: An analogy for it

Why are we learning Dropouts?

CNN can be implemented as a feed forward network with dropouts- so let's look at what is dropouts

- Let's say you were the only person at your company who knows about finance.
- But if every morning you tossed a coin to decide whether you will go to work or not, then **your coworkers will need to adapt**.
- Some days you might not be at work but finance tasks still need to get done, so they can't rely only at you.
- So what's the plan to get the work done?



The finance guy tosses a coin to decide whether to go to work or not



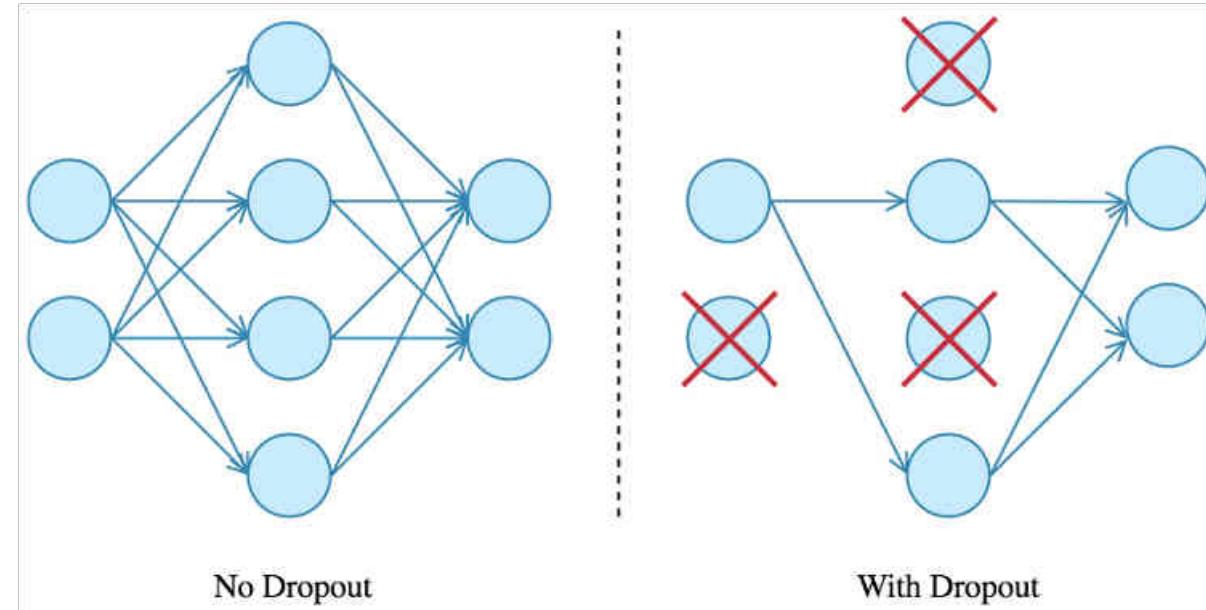
The person on getting heads: goes to work tails: **doesn't go to work**

The idea of dropout was put forward by Nitish Srivastava and Geoffrey Hinton. The link to their paper is:

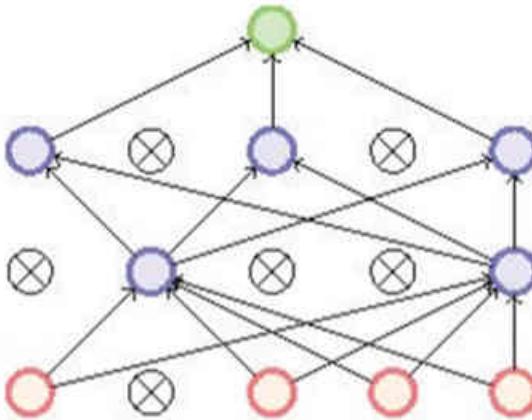
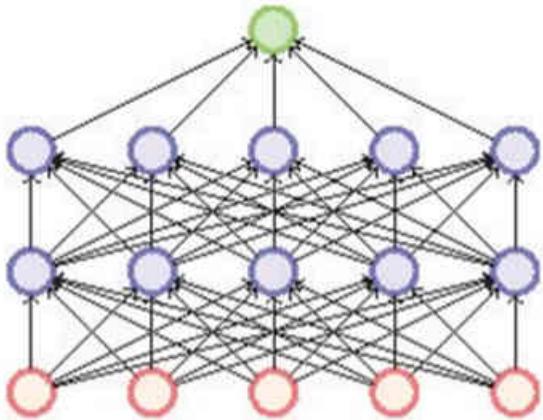
<https://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf>

Dropouts: Real-life analogy

- Your coworkers will need to learn about finance and this expertise needs to be spread out between various people.
- The workers need to cooperate with several other employees, not with a fixed set of people.
- This makes the company **much more resilient overall, increasing the quality and skill set of the employees.**
- **If we assume each employee in the company as neurons and look into the concepts of dropout we will understand it better**

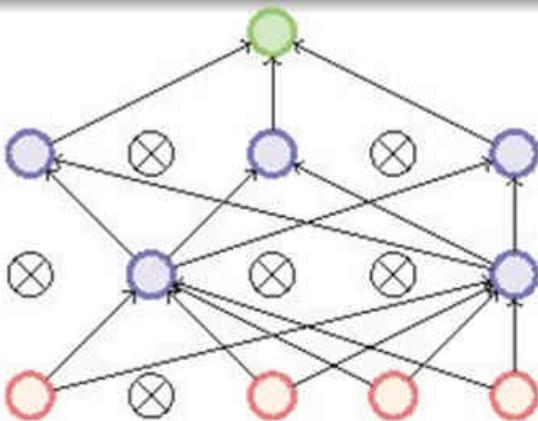
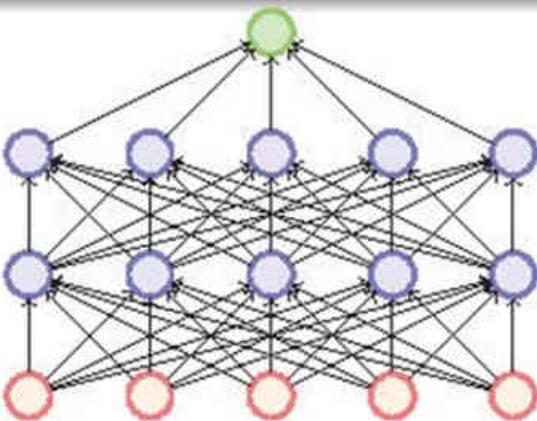


Dropouts



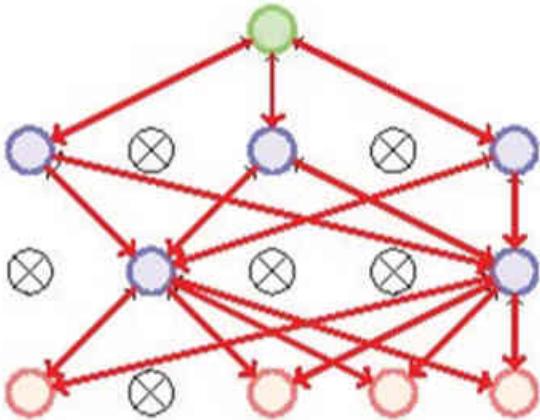
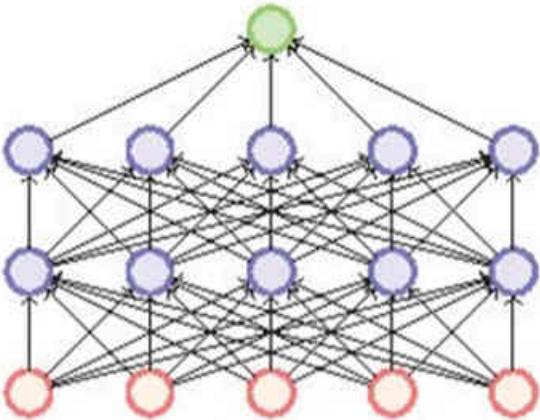
- Dropout refers to dropping out units
- Temporarily remove a node and all its incoming/outgoing connections resulting in a thinned network
- Each node is retained with a fixed probability (typically $p = 0.5$) for hidden nodes and $p = 0.8$ for visible nodes

Dropouts



- Suppose a neural network has n nodes
- Using the dropout idea, each node can be retained or dropped
- For example, in the above case we drop 5 nodes to get a thinned network
- Given a total of n nodes, what are the total number of thinned networks that can be formed? 2^n
- Of course, this is prohibitively large and we cannot possibly train so many networks
- **Trick:** (1) Share the weights across all the networks
(2) Sample a different network for each training instance
- Let us see how?

Dropouts



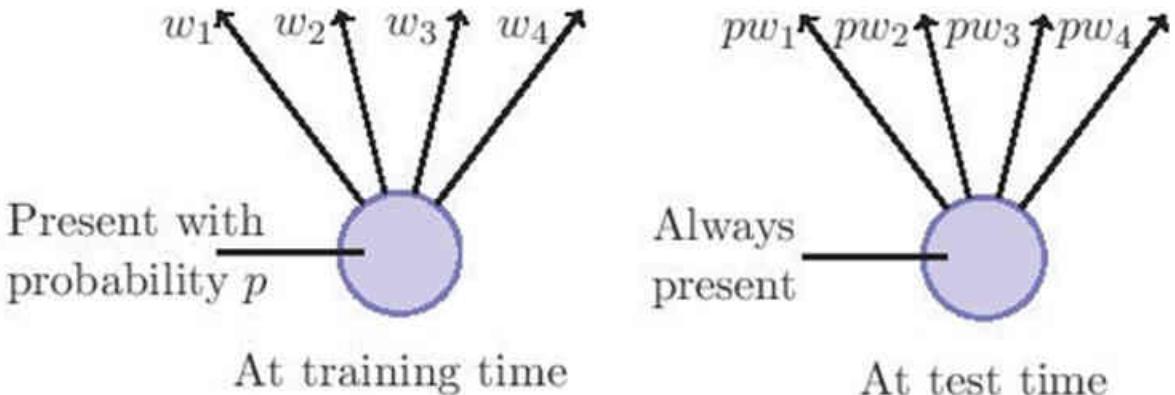
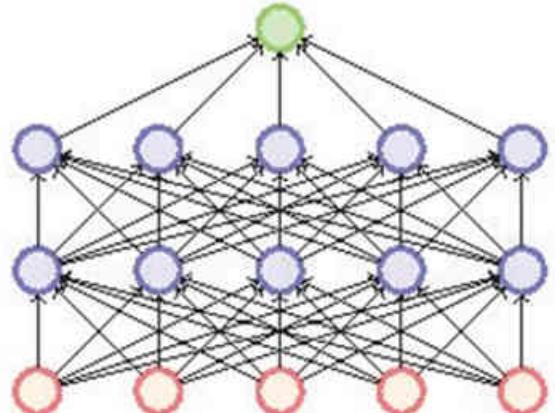
- We initialize all the parameters (weights) of the network and start training
- For the first training instance (or mini-batch), we apply dropout resulting in the thinned network
- We compute the loss and backpropagate
- Which parameters will we update? Only those which are active

Dropouts



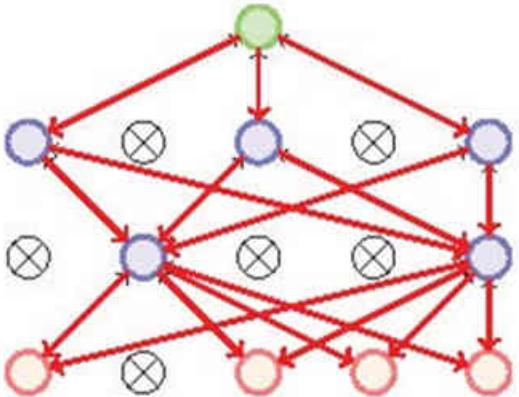
- For the second training instance (or mini-batch), we again apply dropout resulting in a different thinned network
- We again compute the loss and backpropagate to the active weights
- If the weight was active for both the training instances then it would have received two updates by now
- If the weight was active for only one of the training instances then it would have received only one update by now
- Each thinned network gets trained rarely (or even never) but the parameter sharing ensures that no model has untrained or poorly trained parameters

Dropouts



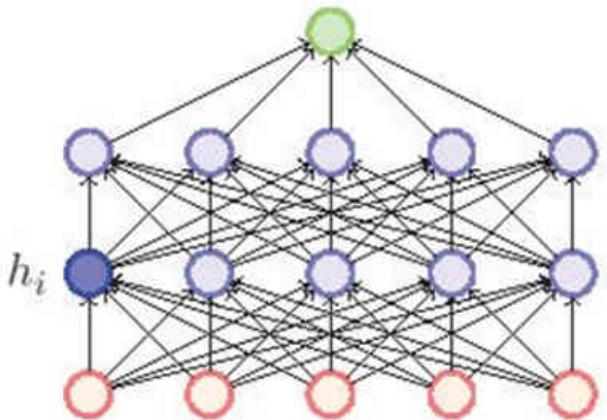
- What happens at test time?
- Impossible to aggregate the outputs of 2^n thinned networks
- Instead we use the full Neural Network and scale the output of each node by the fraction of times it was on during training

Dropouts



- Dropout essentially applies a masking noise to the hidden units
- Prevents hidden units from co-adapting
- Essentially a hidden unit cannot rely too much on other units as they may get dropped out any time
- Each hidden unit has to learn to be more robust to these random dropouts

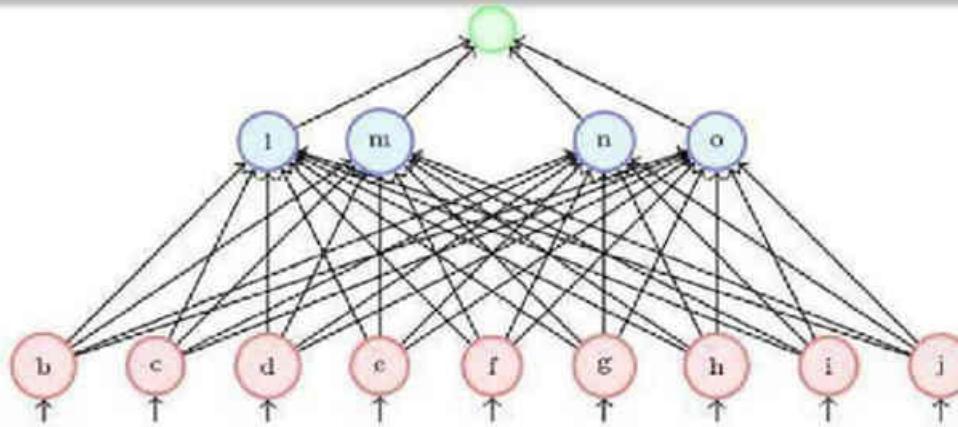
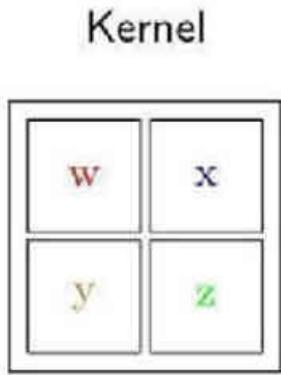
Dropouts



- Here is an example of how dropout helps in ensuring redundancy and robustness
- Suppose h_i learns to detect a face by firing on detecting a nose
- Dropping h_i then corresponds to erasing the information that a nose exists
- The model should then learn another h_i which redundantly encodes the presence of a nose
- Or the model should learn to detect the face using other features

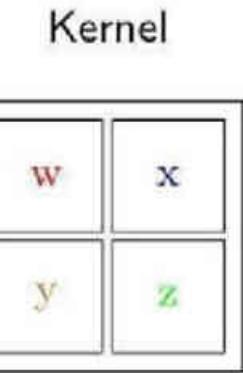
CNN: Training of CNNs

| Input | | |
|-------|---|---|
| b | c | d |
| e | f | g |
| h | i | j |

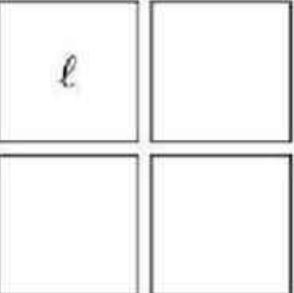


- A CNN can be implemented as a feedforward neural network

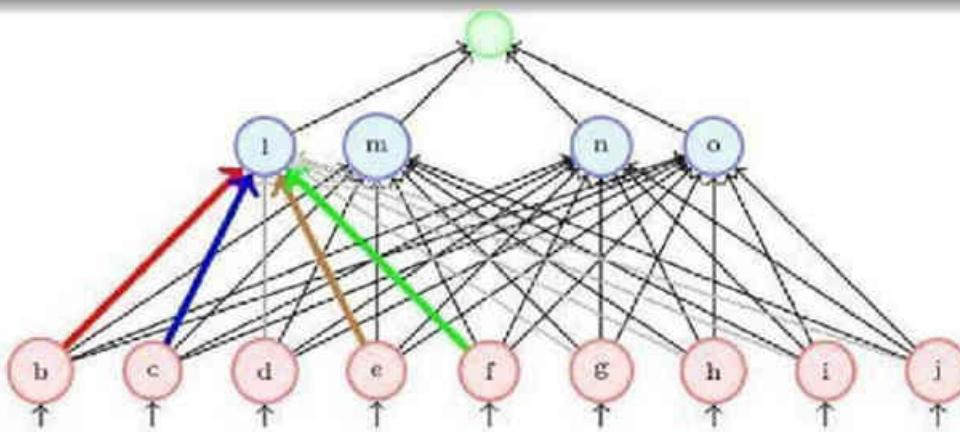
| Input | | |
|-------|---|---|
| b | c | d |
| e | f | g |
| h | i | j |



Output

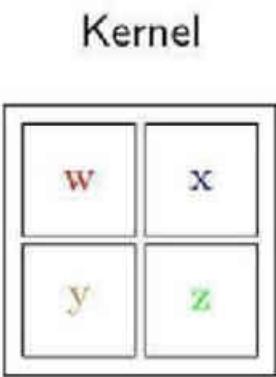


| | |
|--------|--|
| ℓ | |
| | |

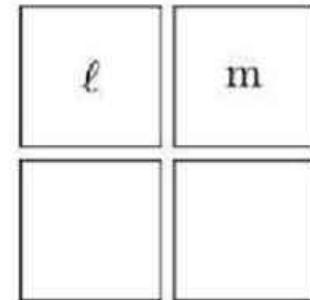


- A CNN can be implemented as a feedforward neural network
- wherein only a few weights(in color) are active

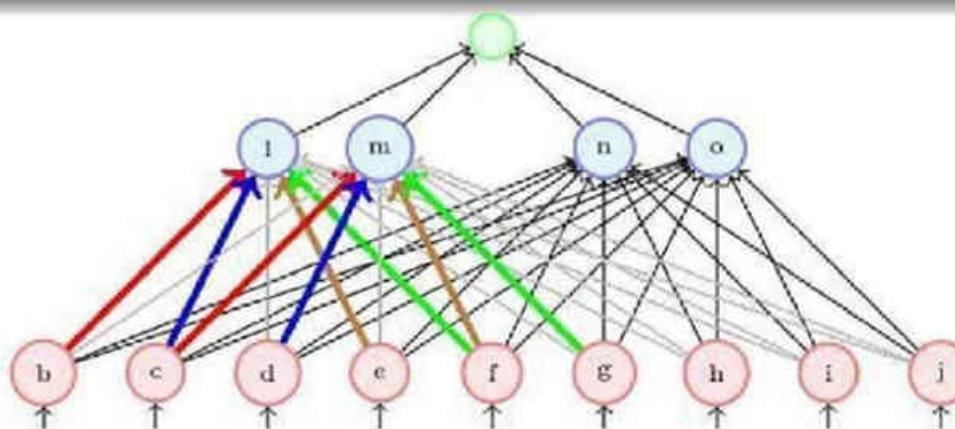
| Input | | |
|-------|---|---|
| b | c | d |
| e | f | g |
| h | i | j |



Output



| | |
|--------|-----|
| ℓ | m |
| | |



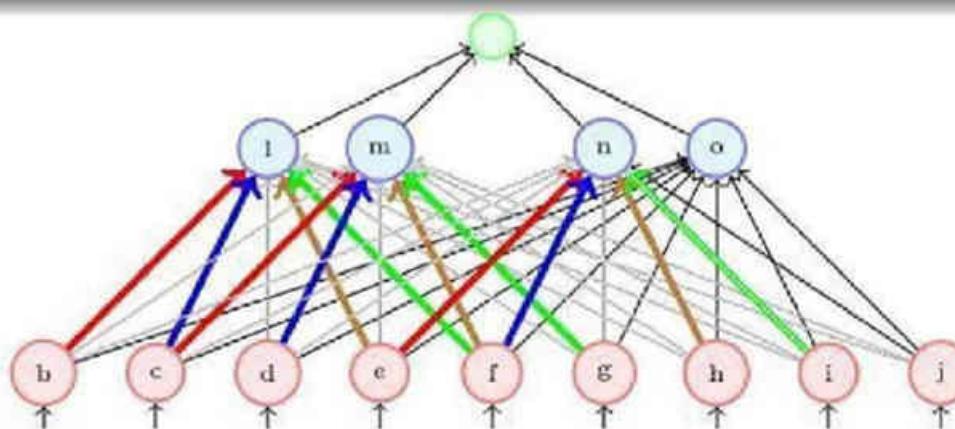
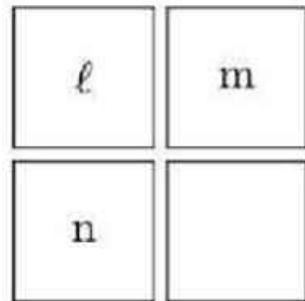
- A CNN can be implemented as a feedforward neural network
- wherein only a few weights(in color) are active
- the rest of the weights (in gray) are zero

CNN

| Input | | |
|-------|---|---|
| b | c | d |
| e | f | g |
| h | i | j |

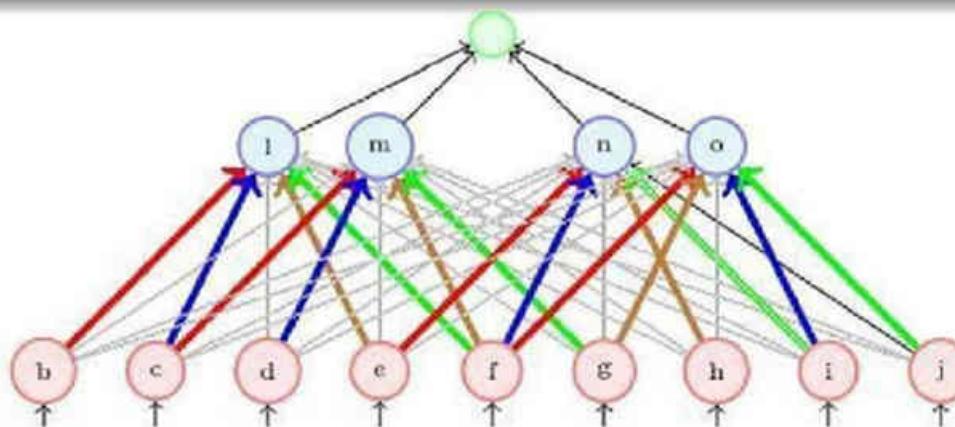
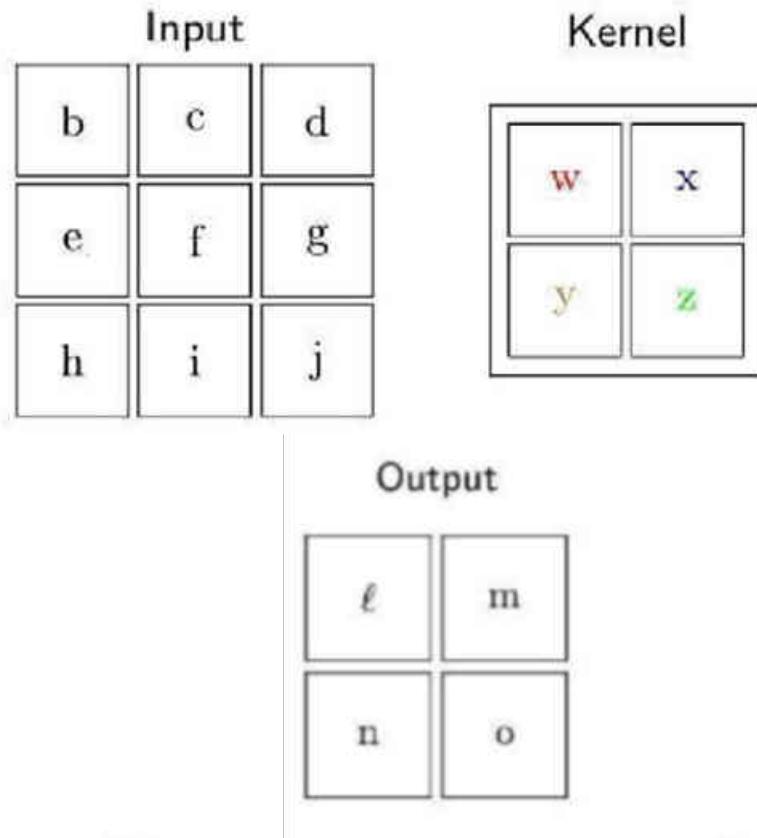
Kernel

Output



- A CNN can be implemented as a feedforward neural network
 - wherein only a few weights(in color) are active
 - the rest of the weights (in gray) are zero

CNN



- We can thus train a convolution neural network using backpropagation by thinking of it as a feedforward neural network with sparse connections

- A CNN can be implemented as a feedforward neural network
- wherein only a few weights(in color) are active
- the rest of the weights (in gray) are zero

References

1. [MIT 6.S191 \(2020\): Convolutional Neural Networks - YouTube](#)
1. [Deep Learning\(CS7015\): Lec 11.3 Convolutional Neural Networks - YouTube- Mitesh M Khapra](#)
1. [Convolutional Neural Network - YouTube- Ahlad Kumar](#)
1. [Lecture Collection | Convolutional Neural Networks for Visual Recognition \(Spring 2017\) - YouTube](#)
1. [CNNs, Part 1: An Introduction to Convolutional Neural Networks - victorzhou.com](#)



THANK YOU

Dr. Arti Arya
artiarya@pes.edu