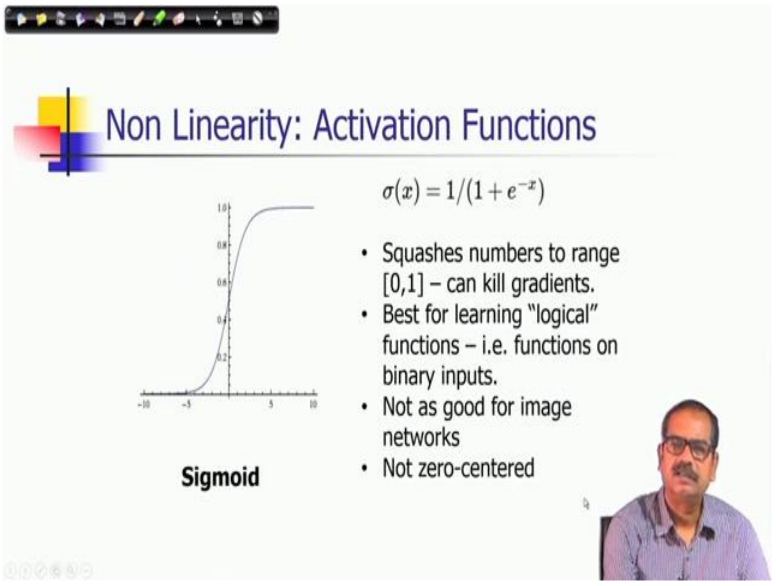
Computer Vision

Prof. Jayanta Mukhopadhyay
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 56
Deep Neural Architecture and Applications Part – II

We are discussing about Deep Neural Architecture. And in the last lecture we discussed how the deep computation is different from classical computations. And, then we have considered different types of supervised learning problems or different facets of supervised learning problem in the context of deep learning architecture.

(Refer Slide Time: 00:42)



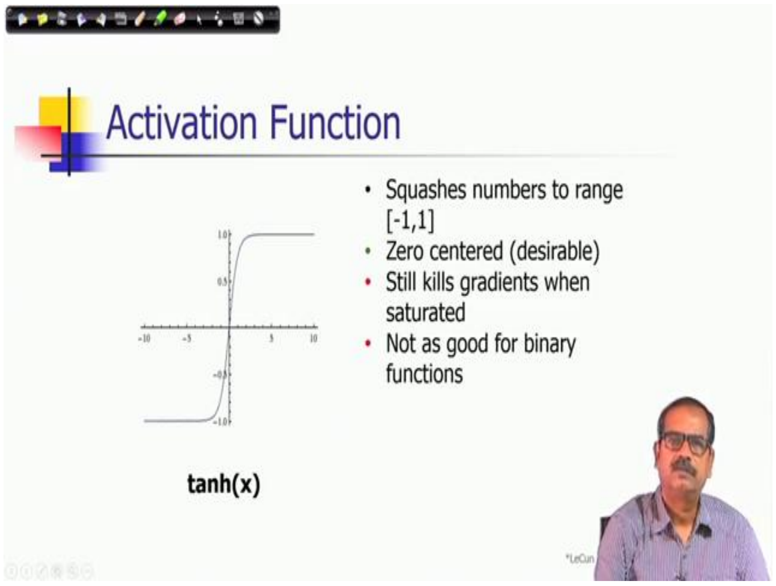
Non Linearity: Activation Functions

$\sigma(x) = 1/(1+e^{-x})$

- Squashes numbers to range [0,1] – can kill gradients.
- Best for learning "logical" functions – i.e. functions on binary inputs.
- Not as good for image networks
- Not zero-centered

Now, in the neural network there is a nonlinearity function, that we know that is used and those are also called activation functions. And, we have seen particularly the sigmoid function that is used in the artificial neural network very often. But so, the property of the sigmoid function is that it squashes numbers to range 0 to 1 And so, it can kill gradients as you can see it saturates when the value increases or value decreases and they are the gradient becomes almost 0. And, then it is best for learning logical functions that is functions on binary inputs, but it is not good for image networks and it is not 0 centered also

(Refer Slide Time: 01:23)

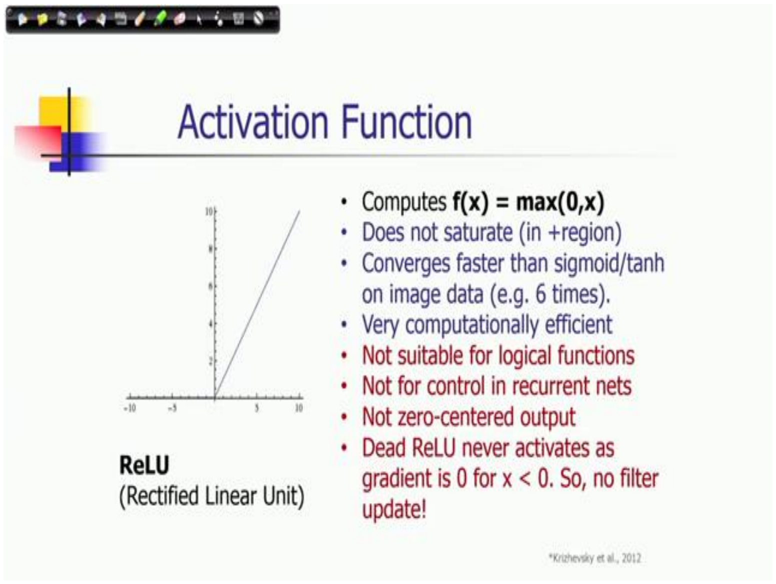


Activation Function

- Squashes numbers to range [-1,1]
- Zero centered (desirable)
- Still kills gradients when saturated
- Not as good for binary functions

So, there is another activation function which is called tan hyperbolic x. So, it is also ranging from -1 to 1 and it is also 0 centered, it is 0 centered which is desirable. And, still it has the problem of that it can kill gradients when it is saturated and it is not as good for binary functions.

(Refer Slide Time: 01:49)



Activation Function

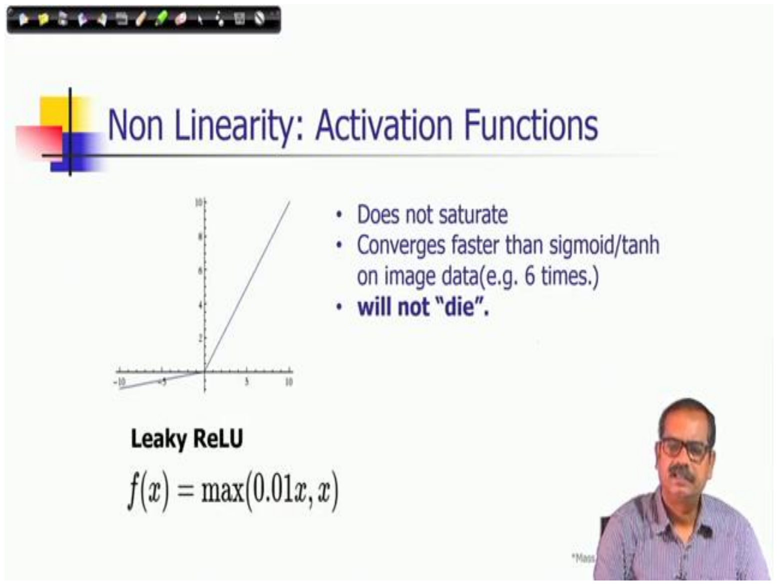
- Computes **f(x) = max(0,x)**
- Does not saturate (in +region)
- Converges faster than sigmoid/tanh on image data (e.g. 6 times).
- Very computationally efficient
- Not suitable for logical functions
- Not for control in recurrent nets
- Not zero-centered output
- Dead ReLU never activates as gradient is 0 for x < 0. So, no filter update!

This particular function rectified linear unit, it is very popular in convolution neural network in different neural architecture. So, the functional form has been shown here that is you can see only positive half of the input it is linear and then for the negative half it is the value is 0. So, that there is a non-linearity and they discontinuity at the value 0 that is first order discontinuities there.

So, but the advantage is that it does not saturate in positive region. And, it converges faster than sigmoid and tan hyperbolic data for example, 6 times. And it is very computationally efficient, but it is not suitable for logical functions for representing modelling logical functions not for control in recurrent nets not 0 centered output.

And, if it is going into the red region which is the negative part then it is not going to activate the filters or activate the gradient. So, it stops there so, there is a dead zone in this case.

(Refer Slide Time: 03:00)



Non Linearity: Activation Functions

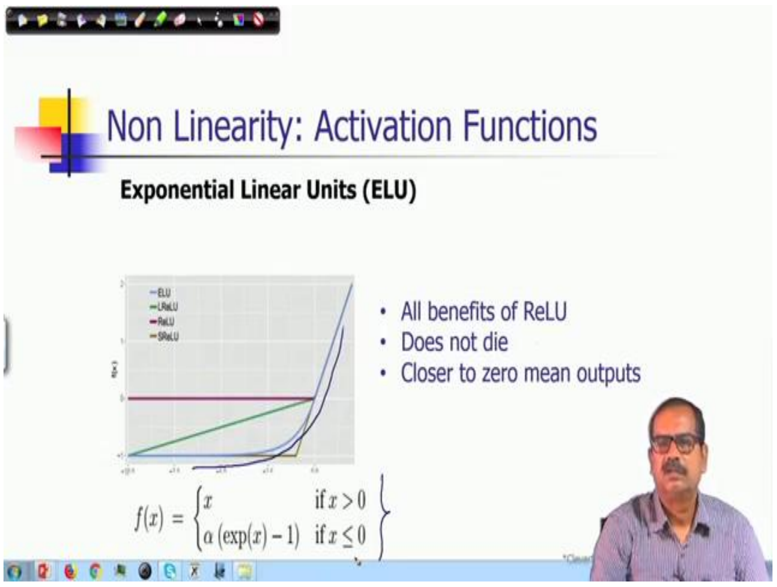
- Does not saturate
- Converges faster than sigmoid/tanh on image data(e.g. 6 times.)
- **will not "die".**

Leaky ReLU

$$f(x) = \max(0.01x, x)$$

So, to avoid that we can have leaky ReLU so, that you have very small increase in the negative part also the small gradient. So, which it has those advantage that it does not saturate, convergence faster and the gradients will not die.

(Refer Slide Time: 03:20)



Non Linearity: Activation Functions

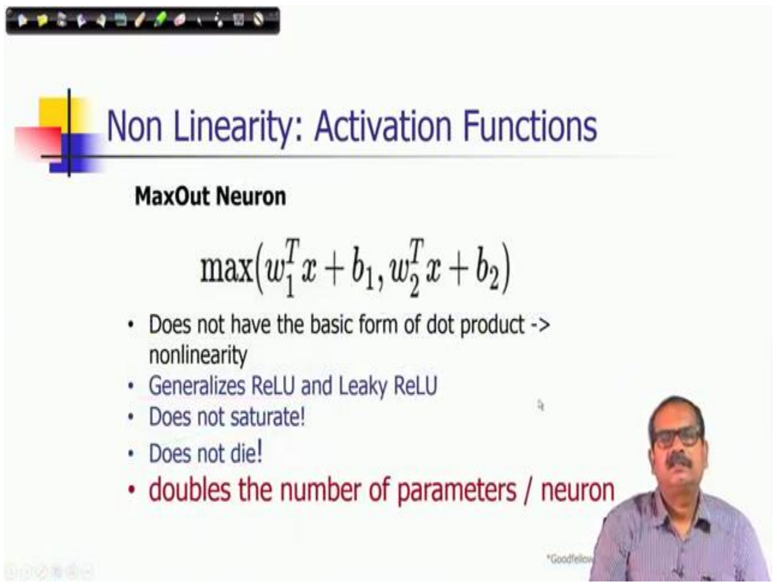
Exponential Linear Units (ELU)

- All benefits of ReLU
- Does not die
- Closer to zero mean outputs

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

Other kind of activation function like ReLU we have also exponential linear units. In this particular graph this blue graph is the exponential linear units which corresponds to this function. So, the benefits of all ReLU benefits are there and it is closer to 0 main outputs.

(Refer Slide Time: 03:47)



Non Linearity: Activation Functions

MaxOut Neuron

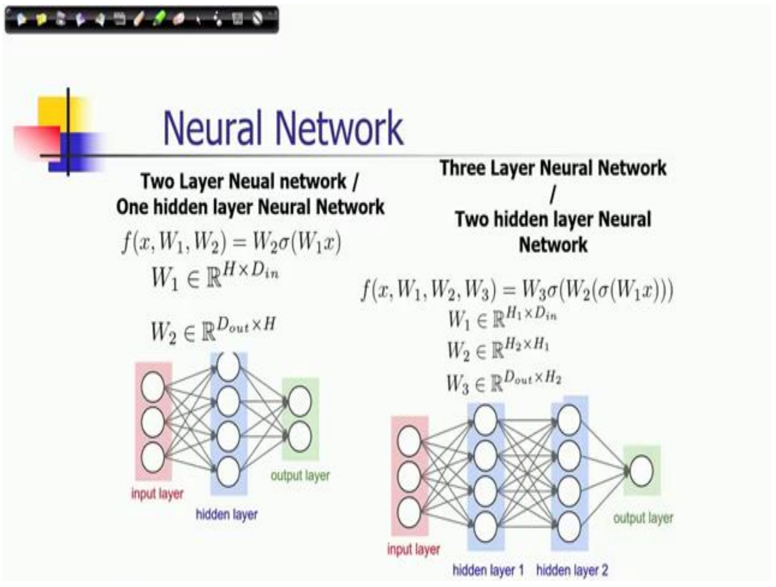
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

- Does not have the basic form of dot product -> nonlinearity
- Generalizes ReLU and Leaky ReLU
- Does not saturate!
- Does not die!
- doubles the number of parameters / neuron

*Goodfellow

Another non-linear activation function which is called max out neuron; here it is a maximum of two linear part, maximum of two linear combinations of inputs $w_1^T x + b_1$ and $w_2^T x + b_2$. So, it does not have basic form of dot product of course, it is non-linear and generalizes ReLU and leaky ReLU does not saturate does not die, problem is that it doubles the number of parameters per neural.

(Refer Slide Time: 04:22)



Neural Network

Two Layer Neual network / One hidden layer Neural Network

$f(x, W_1, W_2) = W_2\sigma(W_1 x)$

$W_1 \in \mathbb{R}^{H \times D_{in}}$

$W_2 \in \mathbb{R}^{D_{out} \times H}$

Three Layer Neural Network / Two hidden layer Neural Network

$f(x, W_1, W_2, W_3) = W_3\sigma(W_2(\sigma(W_1 x)))$

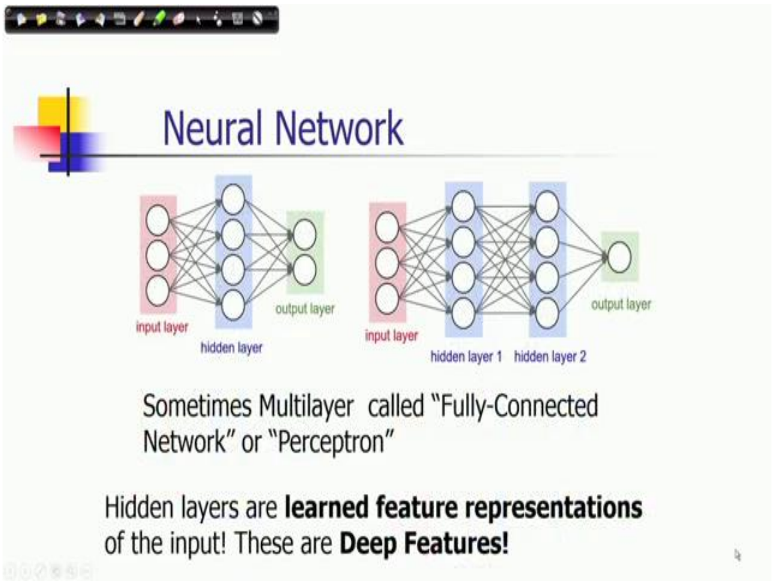
$W_1 \in \mathbb{R}^{H_1 \times D_{in}}$

$W_2 \in \mathbb{R}^{H_2 \times H_1}$

$W_3 \in \mathbb{R}^{D_{out} \times H_2}$

And, we have seen this is the standard architecture of two one hidden layer neural network or two layer neural network. And, another three layer neural network architecture.

(Refer Slide Time: 04:35)



Neural Network

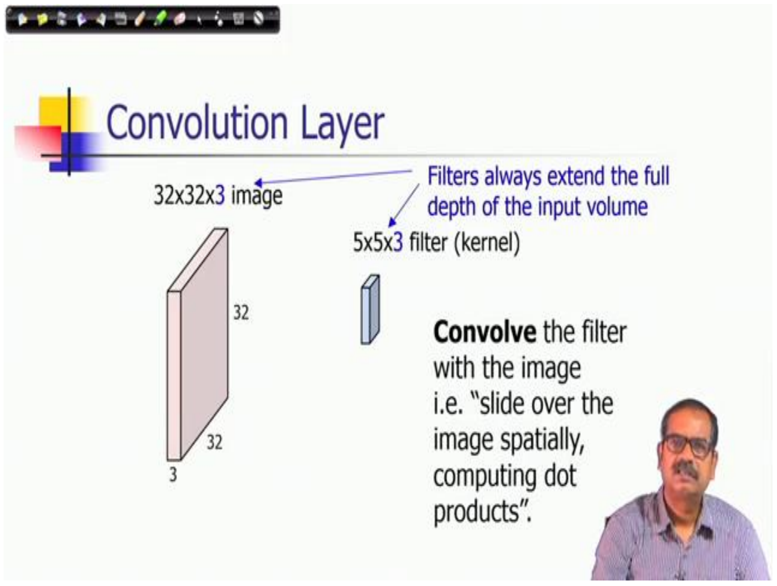
Sometimes Multilayer called "Fully-Connected Network" or "Perceptron"

Hidden layers are **learned feature representations** of the input! These are **Deep Features!**

And sometimes this is called multi layer or fully connected network or perceptron and hidden layers are learned feature representations of the input and these are deep features.

So, with this actually I have talked about general features of neural networks and also deep neural architectures. But, let me discuss the specific a specific deep neural architecture, which is very popular in various applications and which one is convolution neural network.

(Refer Slide Time: 05:05)



Convolution Layer

32x32x3 image

Filters always extend the full depth of the input volume

5x5x3 filter (kernel)

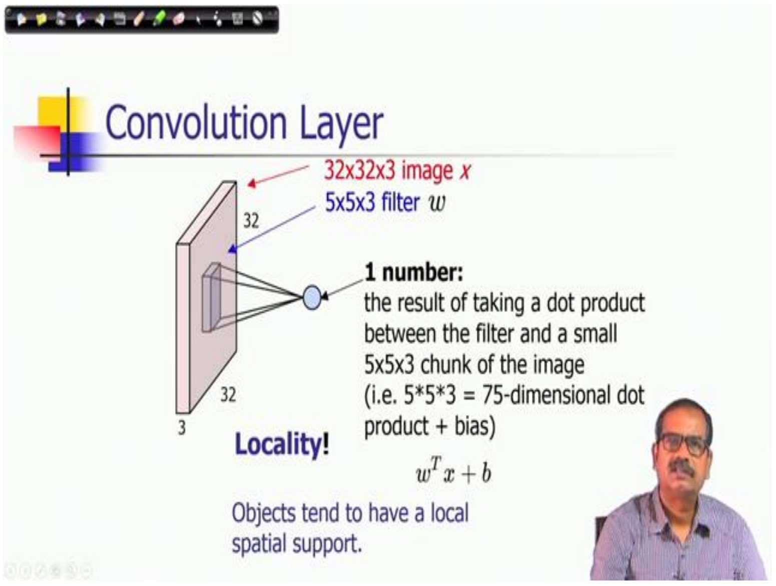
Convolve the filter with the image i.e. "slide over the image spatially, computing dot products".

In conventional neural network, the conventional layer that we need to understand that is a hidden layer. And, in the convolution layer what we have that input is an image as a $32 \times 32 \times 3$ image. That means, for example, it could be RGB 3 channel RGB image or it could be any other three components.

And there is a filter so, this filter performs the convolution operations over this image. And size of the filter sometimes it is called kernel also for example, in this size if here it is given a typical value say $5 \times 5 \times 3$. So, first thing the filter extend the full depth of the input volume. So, 3 is the depth in this example.

So, the filter has to span over the depth and then what it does it takes a dot product. So, I will explain that so, it convolves with respect to the image which means this filter slides over every point and computes convolution. That means, a dot products with the input with the filter weights and then produce that output at that at the central location of that central pixel of that mask.

(Refer Slide Time: 06:25)



Convolution Layer

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

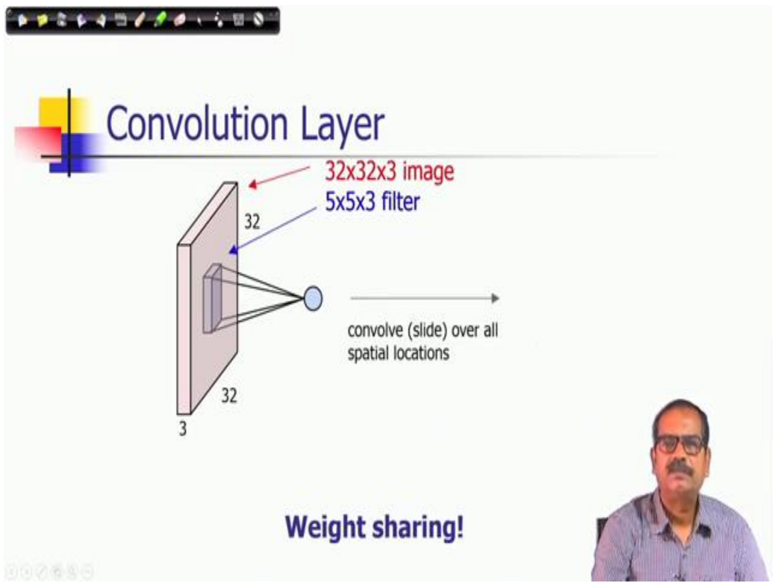
Locality!

Objects tend to have a local spatial support.

So, this is the example that you have $5 \times 5 \times 3$ filter w and it is placed within a pixel of image. And then it takes a dot product and then produces a value and this value is the output at that at that point. So, for example, in this case you have $5 \times 5 \times 3$ chunk of the image; that means, 75 dimensional dot product you are doing and there.

In the neuron as you and as you know that it is the dot product and in the neuron there is a bias term. So, then you add the bias term and that could be also the output of the convolution layer. So, one of the feature of this convolution is that it is the locality which means that objects tend to have a local spatial support. So, this could be exploited using convolution.

(Refer Slide Time: 07:22)

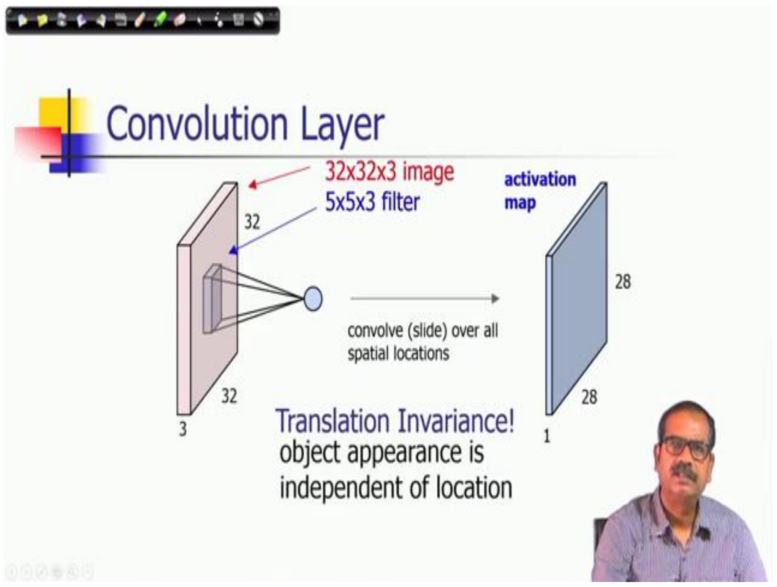


Convolution Layer

Weight sharing!

And the other feature is that weight sharing.

(Refer Slide Time: 07:27)



Convolution Layer

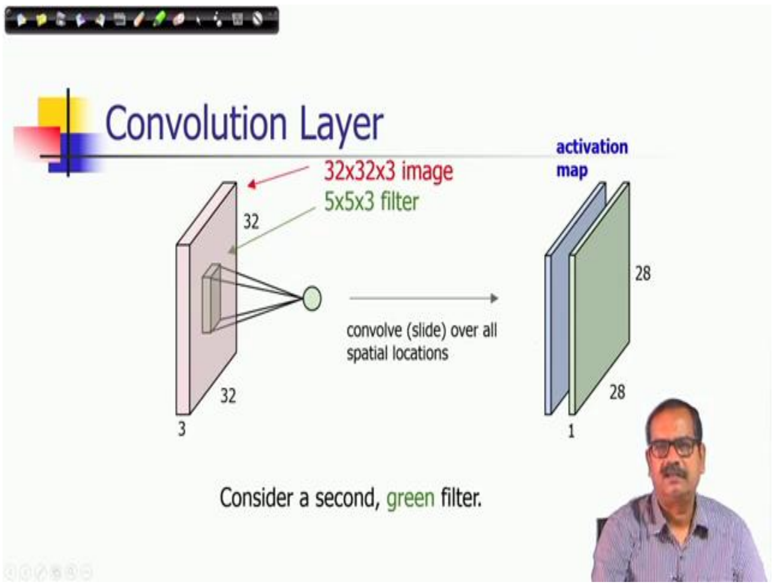
Translation Invariance!
object appearance is independent of location

Since, it will be your convolving over all special locations so, the same weights are used. And finally, you are generating the output. So, if we consider that boundary conditions that only pixels, which are fully embedded the convolution mask which is which is fully embedded within the image at those pixels sites. You are only considering output from those pixels sites then the size of the activation map get reduced.

As you can see the filter is of size 5×5 ; that means, along height and along width you have to leave out two pixels each and at each edges. Because, those are not those pixels are not fully embedded. So, finally, size would be 28×28 and along depth wise also other planes are not coming into picture because they are not fully embedded. Only the central pixels convolutions for the central plane those are only considered here. Because those for those pixels only this pool embedding is available.

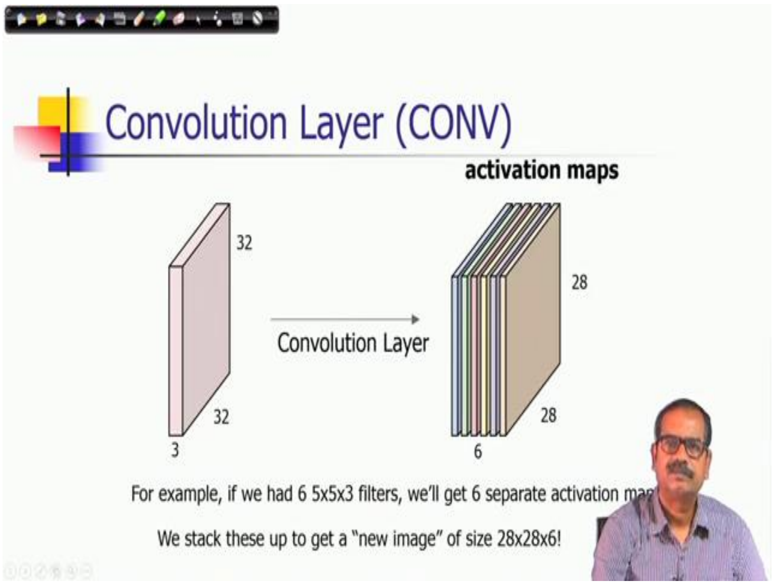
Say another feature of this computation that it is translation invariance which means the object appearance is independent of location.

(Refer Slide Time: 08:43)


Convolution Layer
32x32x3 image
5x5x3 filter
32
32
3
activation map
28
28
1
convolve (slide) over all spatial locations
Consider a second, green filter.

Now, you consider that there are other filters. So, there is another second say green filter of the same size, but its weights could be different. And, it also produced another we call it channel output. So, it also produce another output.

(Refer Slide Time: 08:59)

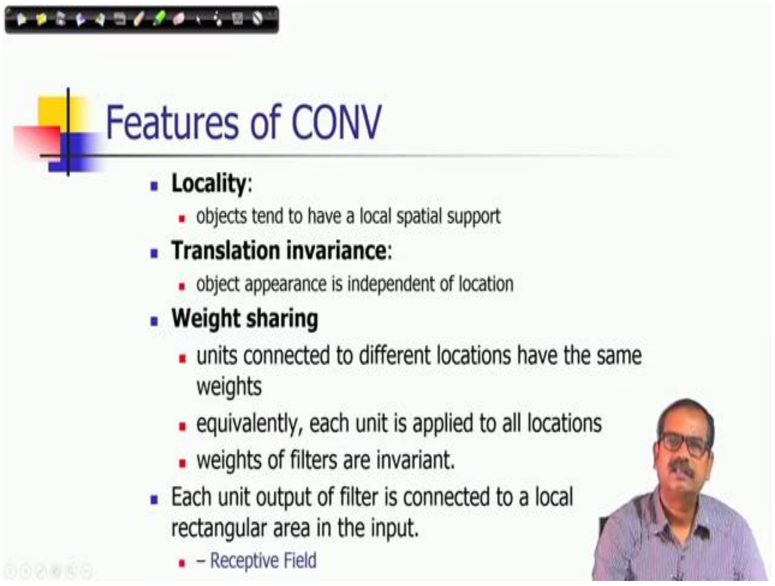

Convolution Layer (CONV)

For example, if we had 6 5x5x3 filters, we'll get 6 separate activation maps

We stack these up to get a "new image" of size 28x28x6!

And, if there are 6 such filters, you have 6 such activation maps six separate activation maps. And you can stack this to get a new image of size $28 \times 28 \times 6$. So, 6 number of convolutional filters.

(Refer Slide Time: 09:20)

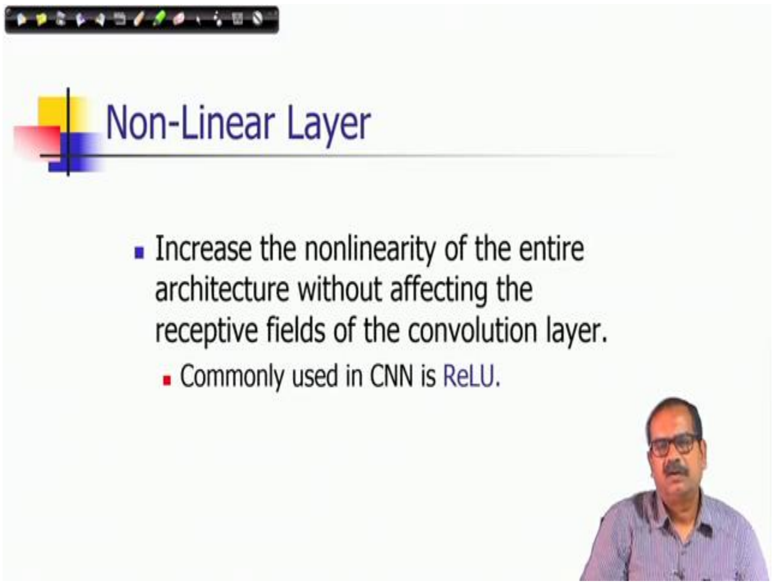

Features of CONV

- **Locality:**
  - objects tend to have a local spatial support
- **Translation invariance:**
  - object appearance is independent of location
- **Weight sharing**
  - units connected to different locations have the same weights
  - equivalently, each unit is applied to all locations
  - weights of filters are invariant.
- Each unit output of filter is connected to a local rectangular area in the input.
  - – Receptive Field

So, the features of CONV layer or convolution layer we can consider that is locality, which is the feature of having local spatial support for objects. Then translation invariance where object appearance could be independent of location; and weight sharing which means the units could be connected to different locations having the same weight.

And equivalent each unit is applied to all locations and weights of filters are invariant. This keeps the number of parameters small or convolution neural network. Of course, since there are many such layers finally, total number of parameters will be quite large. Then each unit output or filter is connected to a local rectangular area in the input and that is considered as the receptive field.

(Refer Slide Time: 10:13)

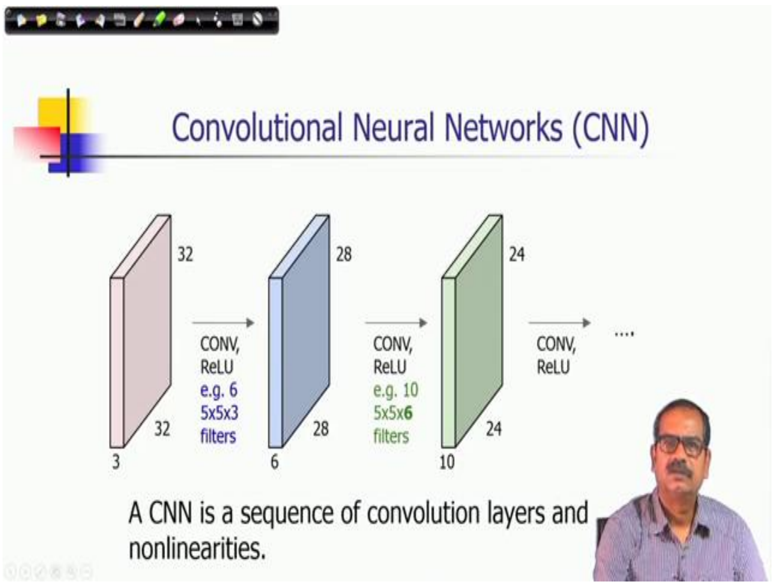


Non-Linear Layer

- Increase the nonlinearity of the entire architecture without affecting the receptive fields of the convolution layer.
  - Commonly used in CNN is ReLU.

So, as there is a non-linear function in each neuron. So, after convolution layer we consider the that the non-linearity at every point itself is considered as a layered response from the system. Whose input is the output of the convolution layer then it goes through the non-linear stages at every pixel. So, it is the point wise non-linearity and it increases the non-linearity of the inter architecture without affecting the respective fields of convolutional layer, which generalizes the model further.

(Refer Slide Time: 10:50)



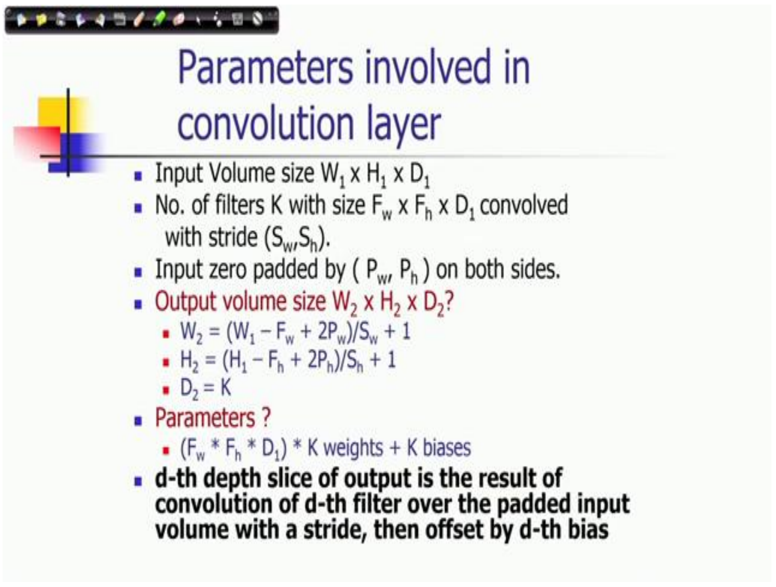
Convolutional Neural Networks (CNN)

32
32
3
CONV, ReLU e.g. 6 5x5x3 filters
28
28
6
CONV, ReLU e.g. 10 5x5x6 filters
24
24
10
CONV, ReLU
....

A CNN is a sequence of convolution layers and nonlinearities.

And, ReLU is commonly used that I have mentioned. So, what is a CNN it is a sequence of convolution layers and nonlinearity. So, you have a this is your input you have the convolutional layer and also the ReLU this is one layer. Then another layer of convolutional layer and ReLU second layer and it goes on. So, CNN is sequence of such convolution layers and nonlinearities.

(Refer Slide Time: 11:23)



Parameters involved in convolution layer

- Input Volume size $W_1 \times H_1 \times D_1$
- No. of filters K with size $F_w \times F_h \times D_1$ convolved with stride (S_w, S_h).
- Input zero padded by (P_w, P_h) on both sides.
- Output volume size $W_2 \times H_2 \times D_2$?
  - $W_2 = (W_1 - F_w + 2P_w)/S_w + 1$
  - $H_2 = (H_1 - F_h + 2P_h)/S_h + 1$
  - $D_2 = K$
- Parameters ?
  - $(F_w * F_h * D_1) * K$ weights + K biases
- **d-th depth slice of output is the result of convolution of d-th filter over the padded input volume with a stride, then offset by d-th bias**

So, let us also see what are the parameters involved in convolutional layer. Consider your input is of size $W_1 \times H_1 \times D_1$. So, in my previous example we have taken $32 \times 32 \times 3$

that was in size of the input. So, value of W_1 was 32, H_1 was 32 and D_1 is 3. And considered there are K, number of filters in the previous example this value of K was 6; we considered 6 filters. And size of the filter is $F_w \times F_h \times D_1$.

So, in the previous example we have taken $5 \times 5 \times 3$ filters, which means F_w was 5; F_h was 5 and D_1 is kept 3. So, you note that the depth wise the filter has to have the same number of channels what you have in the input or same number of depth slices, but is there in the input. Because the idea of convolution layer is that the filter should be totally embedded within the input image.

And that to at the central slice and that is why the depth has to be the same. And they need to provide only single channel output. And then there is another parameter called stride. So, it is considering that the point where you are performing convolutions. So, how the point could be separated in the grid? So, if they are adjacent then stride is 1; that was the previous case.

Suppose you leave 1 out of them then the stride would be 2. And also input can could be 0 padded on both sides to keep the size same. So, if you would like to include also the edge pixels or boundary pixels of the image, then pad the input to 0 and then perform this computation. And it would produce the output volume size of $W_2 \times H_2 \times 2$ So, what should be this output volume size? Now these values are all related with the values of input sizes.

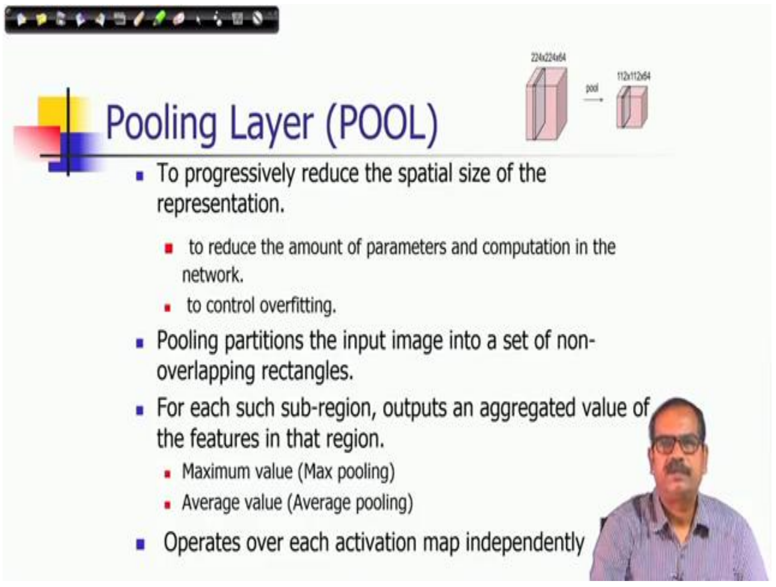
Like values of in W_1, H_1, D_1;; F_w, F_h, D_1, D 1 is not very important here, but W_1 and H_1 F_w, F_h, S_w, S_h and P_w, P_h. For all those values they will determine what should be the size of your output. For example consider the value of W_2 and as this is the expressions. So, it has taken care of striding it has taken care of 0 padding, it has taken care of the size of the filter and complete embedding of the filters within the input.

So, you observe that the depth number of depth slices should be the number of filters K here. And what could be the number of parameters? Since the parameters involves say the weights of the filters and since there are the size of the filter is $F_w \times F_h \times D_1$. So, that many number of weights should be there.

So, it is $F_w \times F_h \times D_1$. And if there are K filters you have to multiply with K. So, we are assuming here that in the convolution layer all the filters of same size and all the filters of same size. So, the parameters as you can see $F_w * F_h * D_1 * K$ that many weights and each neuron could have a bias so; you can have K biases.

So, your summary of this particular operations is that d th depth slice of output is the result of convolution of d th filter of the padded input volume with a stride. And then offset by d th bias.

(Refer Slide Time: 15:28)



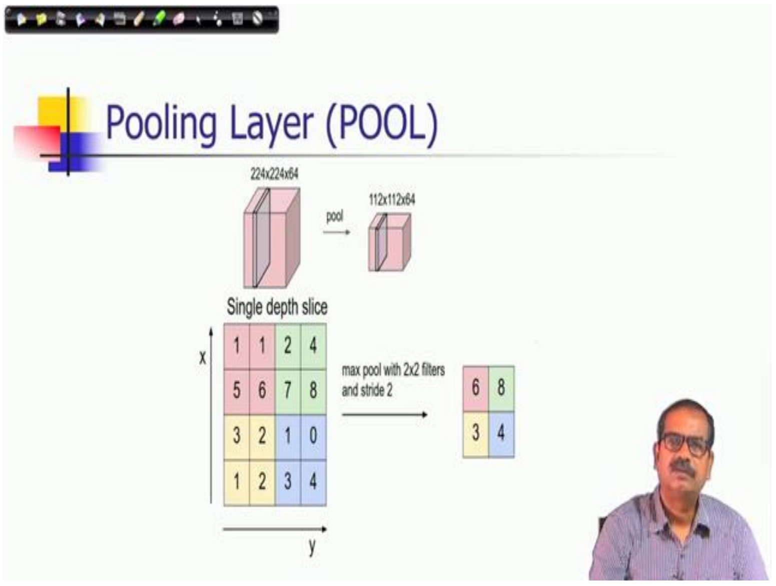
Pooling Layer (POOL)

- To progressively reduce the spatial size of the representation.
  - to reduce the amount of parameters and computation in the network.
  - to control overfitting.
- Pooling partitions the input image into a set of non-overlapping rectangles.
- For each such sub-region, outputs an aggregated value of the features in that region.
  - Maximum value (Max pooling)
  - Average value (Average pooling)
- Operates over each activation map independently

There is another kind of layer in CNN, which is called pooling layer. So, in the pooling layer it tries to progressively reduce the spatial size of the representation that is it tasks. So, that it can reduce the amount of parameters and also computation in the network it controls overfitting. And pooling partitions input image into a set of non overlapping rectangles.

So, for each sub region outputs an aggregated value of the features in that region. There are two types of aggregation one is max pooling, which considers maximum value. The other one is average pooling which takes average value. So, it operates over each activation map independently.

(Refer Slide Time: 16:13)

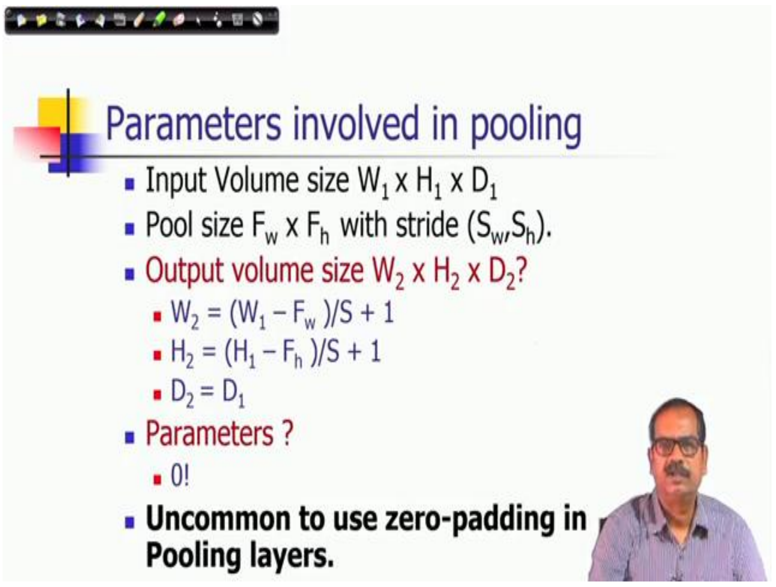


Pooling Layer (POOL)

So, this is an example of pooling layer, you can see that the input was $224\times224\times64$ doing max pooling with 2×2 filters and stride 2 your living one sample in between. That is a stride 2 and that is how the input size output size half of the input size. So, you have 224×224 output would be 100×112, but the number of depth remains the same.

And the example here it is showing max pooling first we have partition the input into 2×2 . Rectangles into rectangular sizes blocks and from there you are choosing the maximum value. And you are replacing you are forming your one pixel for each block.

(Refer Slide Time: 17:01)



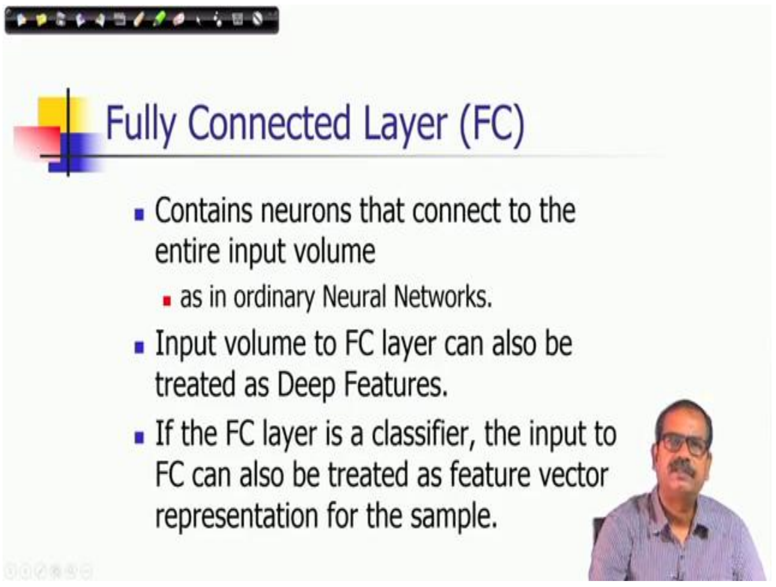
Parameters involved in pooling

- Input Volume size $W_1 \times H_1 \times D_1$
- Pool size $F_w \times F_h$ with stride (S_w, S_h).
- Output volume size $W_2 \times H_2 \times D_2$?
  - $W_2 = (W_1 - F_w)/S + 1$
  - $H_2 = (H_1 - F_h)/S + 1$
  - $D_2 = D_1$
- Parameters ?
  - 0!
- **Uncommon to use zero-padding in Pooling layers.**

So, parameters which are involved in pooling let us observe that. So, here input volume size is once again $W_1 \times H_1 \times D_1$. And pool size could be $F_w \times F_h$ with stride (S_w, S_h). So, output volume size is $W_2 \times H_2 \times D_2$ and that is related with input parameters. So, W_2 would be $\frac{W_1 - F_w}{S_w} + 1$. So, that is why that is what you are doing. And H_2 is $\frac{H_1 - F_h}{S_h} + 1$. So, that is that is the H_2 and your number of depth remains the same as the input.

What about number of parameters as you can see? There is no weight no bias nothing you are it is without when without specifying any things you can perform this computation. So, actually there is no parameter involved in this operation. It is just the computations that is providing you the output of the here. It does not depend upon any parameter. And, it is very uncommon to use zero-padding in a pooling layer.

(Refer Slide Time: 18:12)

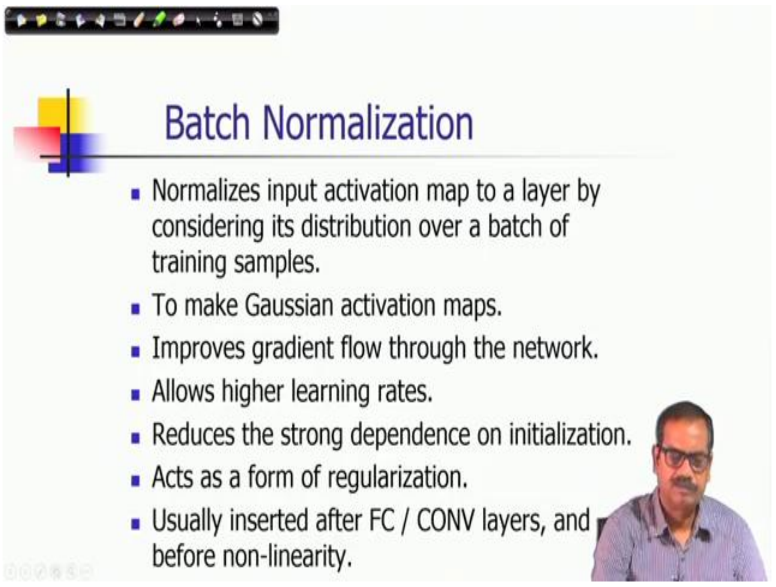


Fully Connected Layer (FC)

- Contains neurons that connect to the entire input volume
  - as in ordinary Neural Networks.
- Input volume to FC layer can also be treated as Deep Features.
- If the FC layer is a classifier, the input to FC can also be treated as feature vector representation for the sample.

Then we will discuss about the fully connected layer which is also coupled with the convolutional layers or CNNs. So, it contains neurons that connect to the entire input volume that is why it is fully connected as it is there in ordinary neural networks. And input volume to FC layer can also be treated as deep features. So, it is either it could be deep features or it could be the feature representation itself for a classifier. So, these are the two options for FC layer.

(Refer Slide Time: 18:42)



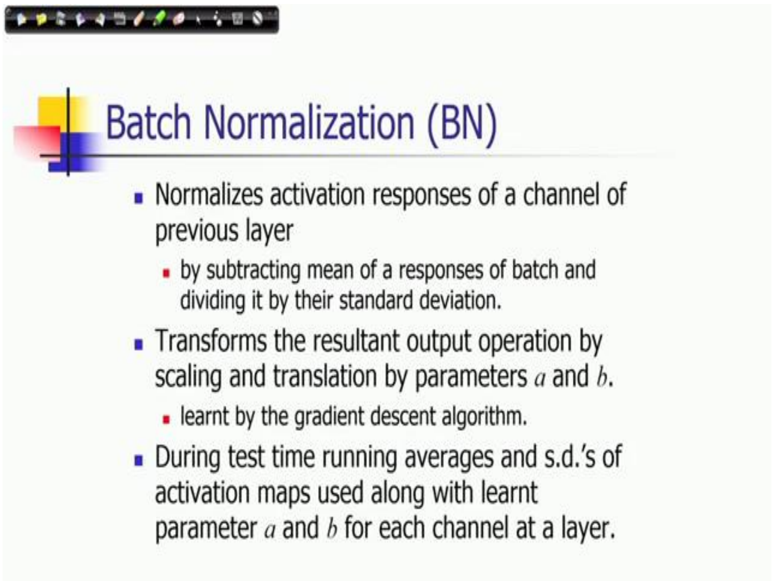
Batch Normalization

- Normalizes input activation map to a layer by considering its distribution over a batch of training samples.
- To make Gaussian activation maps.
- Improves gradient flow through the network.
- Allows higher learning rates.
- Reduces the strong dependence on initialization.
- Acts as a form of regularization.
- Usually inserted after FC / CONV layers, and before non-linearity.

There are various operations, which are required for efficient training and testing of CNN or for efficient computations of CNN effective computation of CNN. One of the operation is known as batch normalization. So, it tries to condition the input and also the intermediate responses. So, what it does? It normalizes input activation map to a layer by considering its distribution over a batch of training samples.

Consider for example, you can apply your Gaussian activation maps Gaussian model there. Using batch normalization you can improve the gradient flow through the network and it allows also higher learning rates. So, your convergence becomes faster. And it is, it reduces the strong dependence on initialization this also acts as a form of regularization of the network. And, batch normalization is usually inserted after fully connected or convolution layers and before non-linearity.

(Refer Slide Time: 19:46)



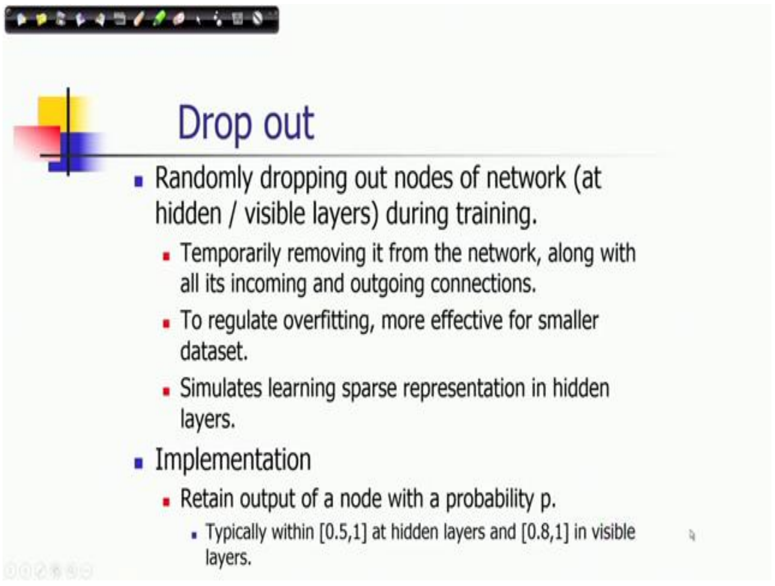
Batch Normalization (BN)

- Normalizes activation responses of a channel of previous layer
  - by subtracting mean of a responses of batch and dividing it by their standard deviation.
- Transforms the resultant output operation by scaling and translation by parameters a and b.
  - learnt by the gradient descent algorithm.
- During test time running averages and s.d.'s of activation maps used along with learnt parameter a and b for each channel at a layer.

So, let me elaborate a little bit about this computation. As I mentioned it normalizes activation responses of a channel a previous layer. And how does this normalization work? It subtracts mean of responses of a batch and divides it by their standard deviation. And transform the resultant output operation by scaling and translation by parameters a and b. In fact, this is also learned by the gradient descent algorithm.

So, during test time running averages and standard deviations of activation maps are used with the learned parameter for each channel at a layer.

(Refer Slide Time: 20:25)



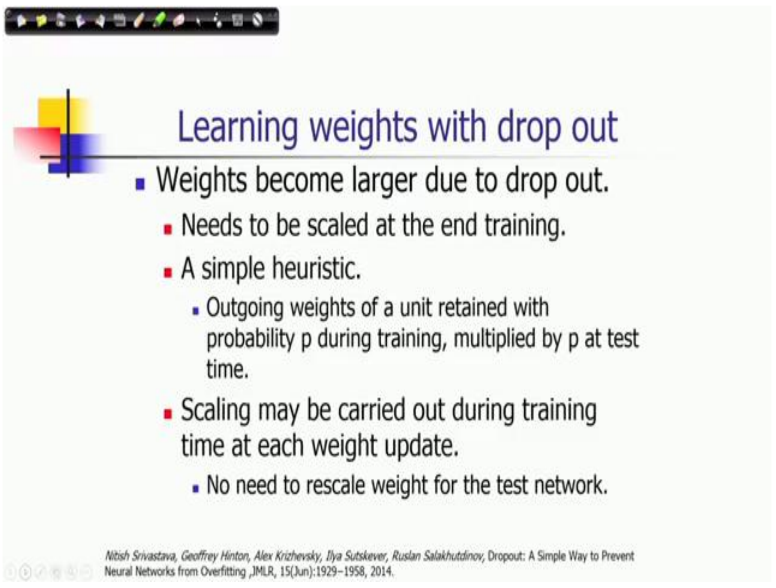
Drop out

- Randomly dropping out nodes of network (at hidden / visible layers) during training.
  - Temporarily removing it from the network, along with all its incoming and outgoing connections.
  - To regulate overfitting, more effective for smaller dataset.
  - Simulates learning sparse representation in hidden layers.
- Implementation
  - Retain output of a node with a probability p.
    - Typically within [0.5,1] at hidden layers and [0.8,1] in visible layers.

So, that is how the batch normalization is carried out in CNNs. There is another operations which is also quite common and which improves the generalization of the model and that is called dropout. What it does? It randomly dropout nodes of network at hidden or visible layers, it could be hidden and visible layer during training.

So, dropout means it temporarily removes that node from the network along with all its incoming and outgoing connections. And, it regulates overfitting and which is more effective for smaller dataset it simulates learning sparse representation in hidden layers. So, implementation of dropout to could be in this way that, you can consider that you can retain an output of a node with a probability p. And, typically the value lies between 0.5 to 1 at hidden layers and 0.8 to 1 in the visible layers which means input or output layers.

(Refer Slide Time: 21:32)



Learning weights with drop out

- Weights become larger due to drop out.
  - Needs to be scaled at the end training.
  - A simple heuristic.
    - Outgoing weights of a unit retained with probability p during training, multiplied by p at test time.
  - Scaling may be carried out during training time at each weight update.
    - No need to rescale weight for the test network.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, Dropout: A Simple Way to Prevent Neural Networks from Overfitting ,JMLR, 15(Jun):1929−1958, 2014.

So, weight becomes larger due to drop out that is one of the effect of dropping out that whatever weight you compute, you learn now the values could be actual value would be larger effect would be larger and you get large weight. So, you need to scale down that weight at the end of the training. And, there is a simple heuristic that if you are outgoing weights of unit is retained with probability p during training.

Then you should multiply by p at the test time or you can consider these operations during training time itself at each weight update. And, then you do not required to do it on during testing. So, you can use the same weight as you learn. So, with this let me take a break. And, we will continue this discussion in the next lecture, where we will be discussing different types of convolutional neural networks, different kinds of architectures in my next lecture.

Thank you very much.