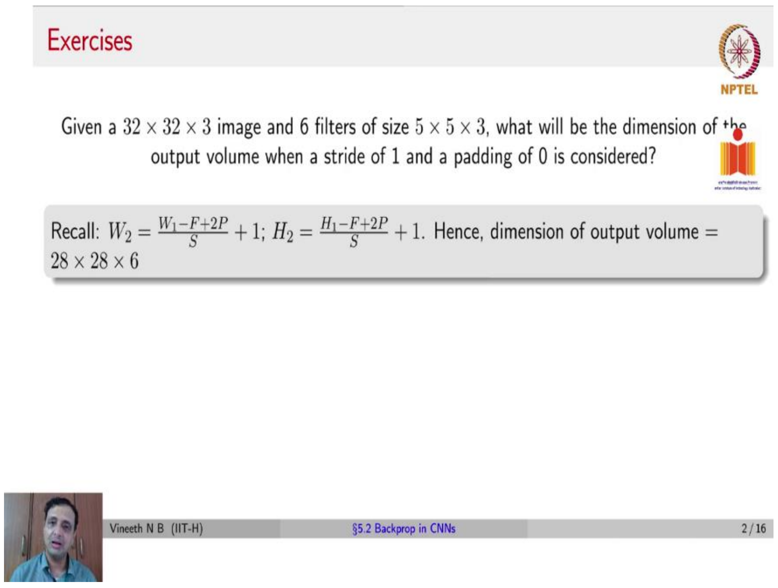
Deep Learning for Computer Vision
Professor Vineeth N Balasubramanian
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
Lecture No. 33
Backpropagation in CNNs

Having seen an introduction to Convolutional Neural Networks, we saw a couple of new layers that you can have in a neural network, a convolutional layer or cooling layer. Now, we ask the question with these new layers how does this affect backpropagation the way we saw it earlier?

(Refer Slide Time: 0:42)



Exercises

Given a $32 \times 32 \times 3$ image and 6 filters of size $5 \times 5 \times 3$, what will be the dimension of the output volume when a stride of 1 and a padding of 0 is considered?

Recall: $W_2 = \frac{W_1 - F + 2P}{S} + 1$; $H_2 = \frac{H_1 - F + 2P}{S} + 1$. Hence, dimension of output volume = $28 \times 28 \times 6$

Vineeth N B (IIT-H) §5.2 Backprop in CNNs 2 / 16

Before we go there, let us revisit our exercise from last class which was given a 32x32x3 image and 6 filters of size 5x5x3, what is the dimension of the output volume with a stride of 1 and a padding of 0? The answer is straight forward. $W_2 = \frac{W_1 - F + 2P}{S} + 1$. The similar formula for H_2 is simply plugging the values and you get 28x28x6 which is the depth of the output because of the number of filters. We did leave behind another question which was if the match pulling layer differentiable and how do you back propagate across it and this is something that we will visit at the end of this lecture.

(Refer Slide Time: 1:37)

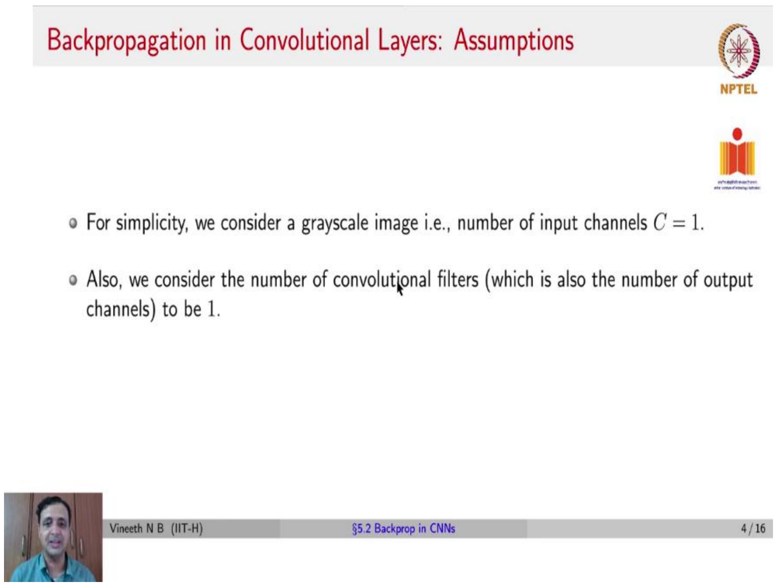
Acknowledgements

- This lecture's content is largely based on a similar lecture by Dhruv Batra at Georgia Tech (with a few adaptations)

Vineeth N B (IIT-H) §5.2 Backprop in CNNs 3 / 16

Let us now start and try to do backpropagation across a convolutional layer first. A large part of this lecture's content is adapted from Dhruv Batra's excellent lectures at Georgia Tech.

(Refer Slide Time: 1:51)

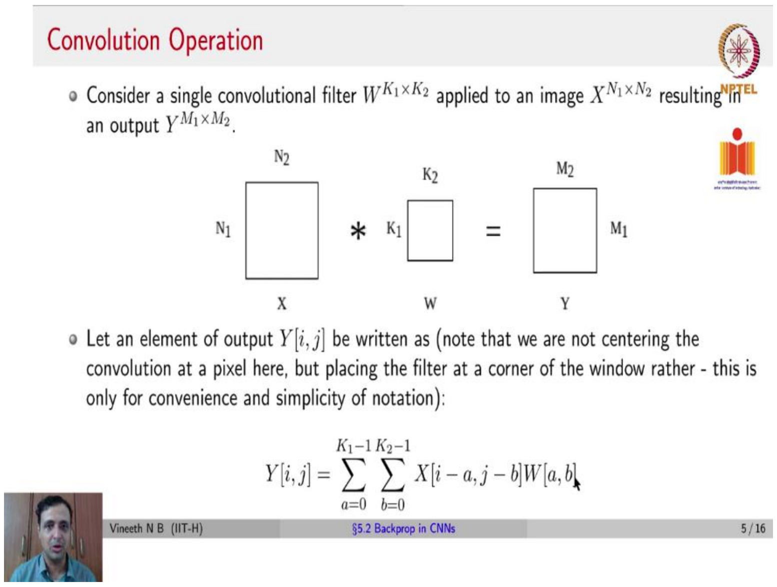

Backpropagation in Convolutional Layers: Assumptions

- For simplicity, we consider a grayscale image i.e., number of input channels $C = 1$.
- Also, we consider the number of convolutional filters (which is also the number of output channels) to be 1.

Vineeth N B (IIT-H) §5.2 Backprop in CNNs 4 / 16

To begin, let us assume a grayscale image, we assume that the number of input channels C=1. You should see that this one affects the derivation per se, it just makes it simpler for explaining. Also, we are going to assume the number of convolutional filters to be 1 which means the size of your output map will of the depth of your output map also be 1. But each convolutional will follow the same procedure to be able to computer its gradients.

(Refer Slide Time: 2:27)



Convolution Operation

- Consider a single convolutional filter $W^{K_1 \times K_2}$ applied to an image $X^{N_1 \times N_2}$ resulting in an output $Y^{M_1 \times M_2}$.

- Let an element of output $Y[i,j]$ be written as (note that we are not centering the convolution at a pixel here, but placing the filter at a corner of the window rather - this is only for convenience and simplicity of notation):

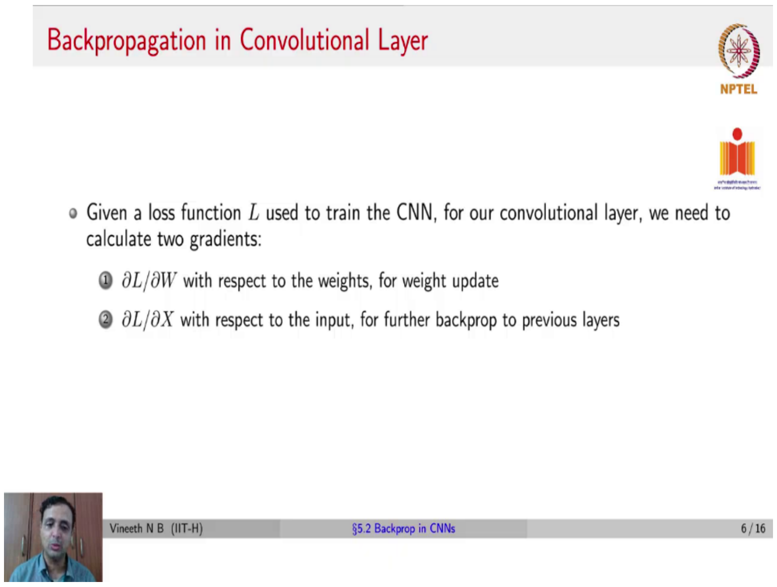
$$Y[i,j] = \sum_{a=0}^{K_1-1} \sum_{b=0}^{K_2-1} X[i-a, j-b] W[a,b]$$

Vineeth N B (IIT-H) §5.2 Backprop in CNNs 5 / 16

To start let us consider a single convolutional filter, let us assume the size of that filter to be K1xK2. This is applied to an image which is of size N1xN2 and you get an output Y which is of size M1xM2. Visually speaking, this is your space, you have an input X, filter W, and output Y. From the definition of convolution, we know that an element of the output Y, Y at ij. $Y[i,j] = \sum_{a=0}^{K_1-1} \sum_{b=0}^{K_2-1} X[i-a, j-b] W[a,b]$.

Note here that we are not placing the filer at the center of the input pixel but a corner. This does not matter, it is only to make this expression a little simpler to see. So, we would make the rest of the derivation a little simpler to understand.

(Refer Slide Time: 3:42)



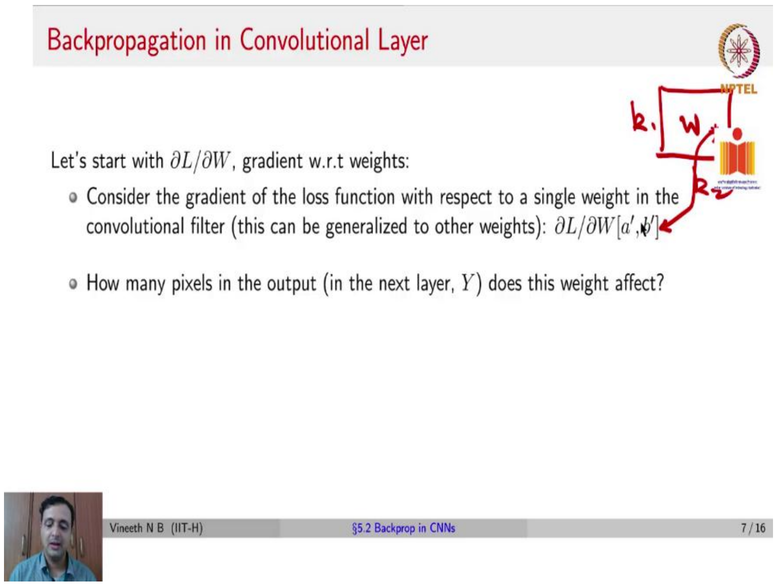
Backpropagation in Convolutional Layer

- Given a loss function L used to train the CNN, for our convolutional layer, we need to calculate two gradients:
  1. $\partial L / \partial W$ with respect to the weights, for weight update
  2. $\partial L / \partial X$ with respect to the input, for further backprop to previous layers

Vineeth N B (IIT-H) §5.2 Backprop in CNNs 6 / 16

We now are given a loss function L which is used to train a CNN. So, for our convolutional layer, there are two quantities that we have to compute. One is $\frac{\partial L}{\partial W}$, the gradient of the loss with respect to every weight in every filter and $\frac{\partial L}{\partial X}$ which is the loss with respect to every pixel of the input because that is necessary to backpropagate further to an earlier layer. We are going to try to derive the gradient for each of these quantities using the chain rule.

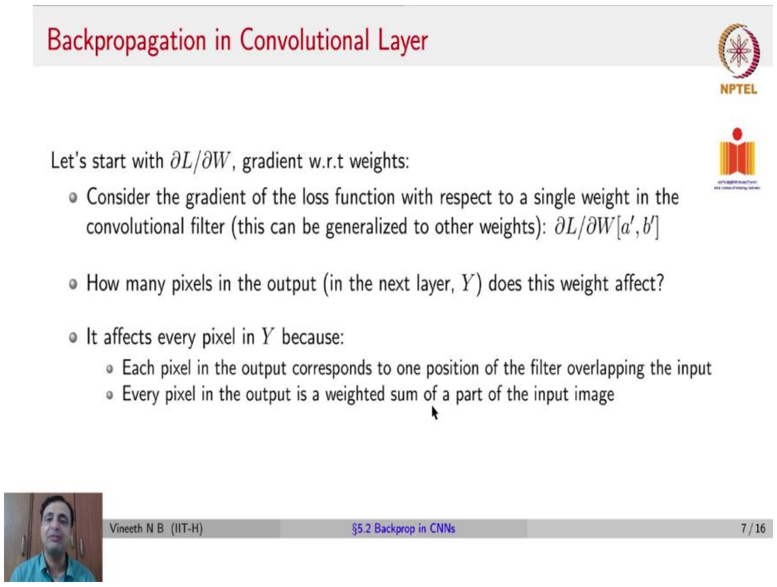
(Refer Slide Time: 4:23)



Backpropagation in Convolutional Layer

Let's start with $\partial L/\partial W$, gradient w.r.t weights:

- Consider the gradient of the loss function with respect to a single weight in the convolutional filter (this can be generalized to other weights): $\partial L/\partial W[a',b']$
- How many pixels in the output (in the next layer, Y) does this weight affect?



Backpropagation in Convolutional Layer

Let's start with $\partial L/\partial W$, gradient w.r.t weights:

- Consider the gradient of the loss function with respect to a single weight in the convolutional filter (this can be generalized to other weights): $\partial L/\partial W[a',b']$
- How many pixels in the output (in the next layer, Y) does this weight affect?
- It affects every pixel in Y because:
  - Each pixel in the output corresponds to one position of the filter overlapping the input
  - Every pixel in the output is a weighted sum of a part of the input image

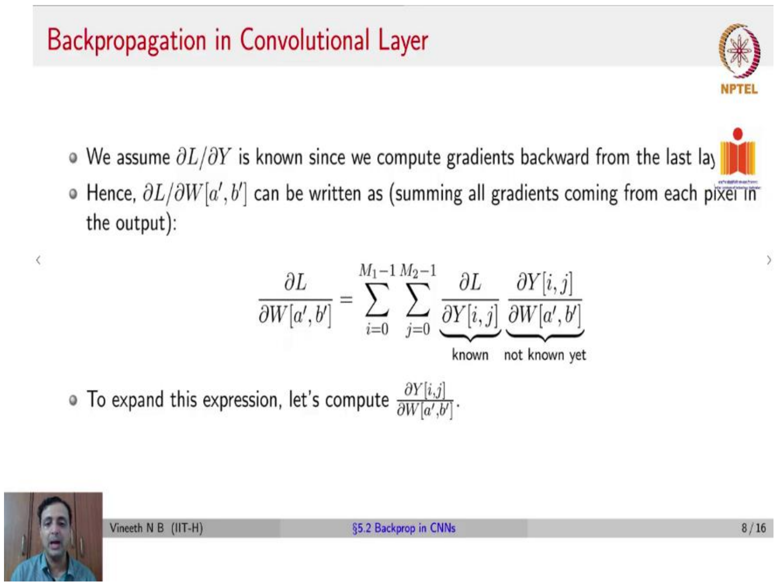
Let us start with $\frac{\partial L}{\partial W}$, the gradient with respect to the weights on the filter. To do this let us consider one particular weight in the convolutional filter. Let us say that you have $W[a', b']$ as one location on your filter. So, if you have your filter to be W which is K1xK2 dimensions, you have one of those values inside them to be $W[a', b']$. We now want to compute the gradient of the loss with respect to that particular weight and then this can be generalized to all weights in that filter.

To do this let us ask the question if you used a filter in a convolutional layer which is W, how many pixels and which pixels in the output or the next layer Y does that particular weigh value affect? Because remember the way backpropagation is done, all the values that a particular pixel impacts the next layer we have to accumulate all those gradients back into the gradient at that particular pixel. This is what we saw with feed-forward neural networks.

So, in this case, the question is if we took the weight at a prime be prime, what all pixels in the next layers map does that particular weight affect? The answer is every pixel in Y because every pixel in Y is obtained by convolution of the filter with the certain location in X. while a certain pixel in Y depends only on a few pixels in X, it does depend on every value in W which is the filter.

So, each pixel in the output corresponds to one position of the filter overlapping the input and every pixel in the output is a weighted sum of a part of the input image but the weight in the filter affects every pixel in the output.

(Refer Slide Time: 6:47)



Backpropagation in Convolutional Layer

- We assume $\partial L/\partial Y$ is known since we compute gradients backward from the last lay
- Hence, $\partial L/\partial W[a',b']$ can be written as (summing all gradients coming from each pixel in the output):

$$\frac{\partial L}{\partial W[a',b']} = \sum_{i=0}^{M_1-1}\sum_{j=0}^{M_2-1} \underbrace{\frac{\partial L}{\partial Y[i,j]}}_{\text{known}} \underbrace{\frac{\partial Y[i,j]}{\partial W[a',b']}}_{\text{not known yet}}$$

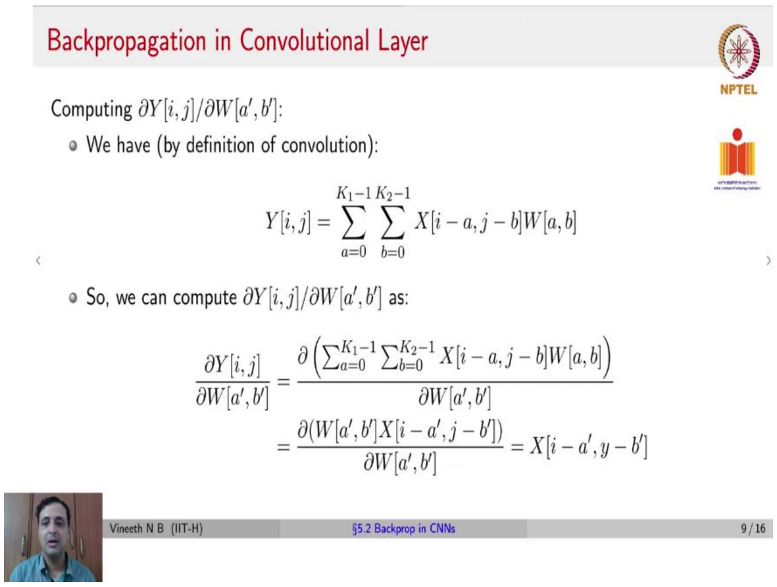
- To expand this expression, let's compute $\frac{\partial Y[i,j]}{\partial W[a',b']}$.

Vineeth N B (IIT-H) §5.2 Backprop in CNNs 8 / 16

Now, let us move forward we now write $\left(\frac{\partial L}{\partial W[a',b']} = \sum_{i=0}^{M_1-1}\sum_{j=0}^{M_2-1}\frac{\partial L}{\partial Y[i,j]}\frac{\partial Y[i,j]}{\partial W[a',b']}\right)$: $\frac{\partial L}{\partial W[a',b']}$ given by the summation of the entire dimension of Y, remember Y we said has a dimension of M1xM2. So, we have to some overall of these gradients where you have $\frac{\partial L}{\partial Y[i,j]}$ which is going to be the gradient of the loss until one particular pixel in that next layers map into $\frac{\partial Y[i,j]}{\partial W[a',b']}$

chain rule. $\frac{\partial L}{\partial Y}$ we assume is already known through backpropagation until that particular step. Our challenge now for the convolutional layer is to compute $\frac{\partial Y}{\partial W}$ in particular $\frac{\partial Y[i,j]}{\partial W[a',b']}$. That is the quantity that we now have to compute to compute this overall gradient of the loss with respect to $W[a',b']$.

(Refer Slide Time: 7:57)



Backpropagation in Convolutional Layer

Computing $\partial Y[i,j]/\partial W[a',b']$:

- We have (by definition of convolution):

$$Y[i,j] = \sum_{a=0}^{K_1-1} \sum_{b=0}^{K_2-1} X[i-a, j-b] W[a,b]$$

- So, we can compute $\partial Y[i,j]/\partial W[a',b']$ as:

$$\frac{\partial Y[i,j]}{\partial W[a',b']} = \frac{\partial \left(\sum_{a=0}^{K_1-1} \sum_{b=0}^{K_2-1} X[i-a,j-b] W[a,b] \right)}{\partial W[a',b']}$$
$$= \frac{\partial (W[a',b'] X[i-a',j-b'])}{\partial W[a',b']} = X[i-a', y-b']$$

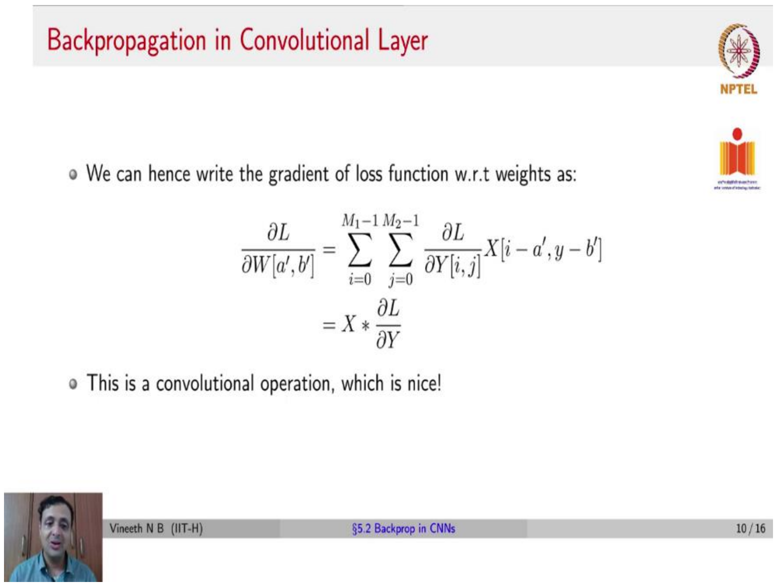
Vineeth N B (IIT-H) §5.2 Backprop in CNNs 9 / 16

Let us consider that particular quantity now which is $\frac{\partial Y[i,j]}{\partial W[a',b']}$. By definition of convolution once again we have $Y[i,j]$ is this follows from the standard definition of convolution this K1, K2 is the filter sizes and this your first principles definition of convolution using this we can now say $\frac{\partial Y[i,j]}{\partial W[a',b']}$ can be written as $\frac{\partial Y[i,j]}{\partial W[a',b']}$ which you now expand $Y[i,j]$ in terms of the entire output that you have from the first equation. So, you have the entire RHS of the first equation you put that in there.

Now, in all these summations here is only one summation one term here that is going to depend on $W[a',b']$. Remember in one convolution, one particular filter value is converted with only 1 pixel in the input in that single convolution operation when you move the filter to the next location, it may contribute something else in the other location but in one single convolution operation, one value in the filter is multiplied by only 1 input pixel and that term now is going to be $W[a',b']$ into $X[i-a',y-b']$.

Every other term in this double summation when you differentiate with respect to $W[a',b']$ will become 0 because it does not depend on $W[a',b']$. This quantity trivially becomes $X[i-a',y-b']$. Let us see how to use this.

(Refer Slide Time: 9:47)



Backpropagation in Convolutional Layer

- We can hence write the gradient of loss function w.r.t weights as:

$$\frac{\partial L}{\partial W[a',b']} = \sum_{i=0}^{M_1-1}\sum_{j=0}^{M_2-1} \frac{\partial L}{\partial Y[i,j]} X[i-a', y-b']$$
$$= X * \frac{\partial L}{\partial Y}$$

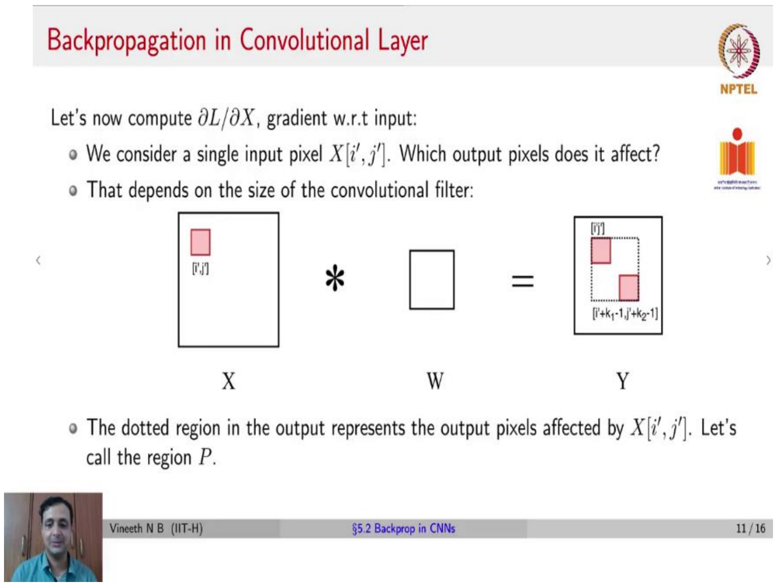
- This is a convolutional operation, which is nice!

Vineeth N B (IIT-H) §5.2 Backprop in CNNs 10 / 16

Let us plug this back end we now have $\frac{\partial L}{\partial W[a',b']}$ equal to summation over all the pixels in Y, $\frac{\partial L}{\partial Y}$ for every pixel in Y into the second term which we just compute on the previous slide which turns out to be $X[i - a', y - b']$ look carefully this is now the convolution of X and $\frac{\partial L}{\partial Y}$ which is beautiful. $\left(\frac{\partial L}{\partial W[a',b']} = \sum_{i=0}^{M_1-1}\sum_{j=0}^{M_2-1}\frac{\partial L}{\partial Y[i,j]}X[i-a', y-b']\right)$

$\frac{\partial L}{\partial Y}$ remember is also going to have the dimensions of Y it also that gradient that set of gradients will look like an MH because you have a gradient of the loss with respect to every pixel in Y. So, that is going to form one matrix and X is a matrix by itself the convolution of X and $\frac{\partial L}{\partial Y}$ is $\frac{\partial L}{\partial W[a',b']}$. So, you can compute this as a convolution in the back-propagation step.

(Refer Slide Time: 10:52)



Backpropagation in Convolutional Layer

Let's now compute $\partial L/\partial X$, gradient w.r.t input:

- We consider a single input pixel $X[i', j']$. Which output pixels does it affect?
- That depends on the size of the convolutional filter:

- The dotted region in the output represents the output pixels affected by $X[i', j']$. Let's call the region P.

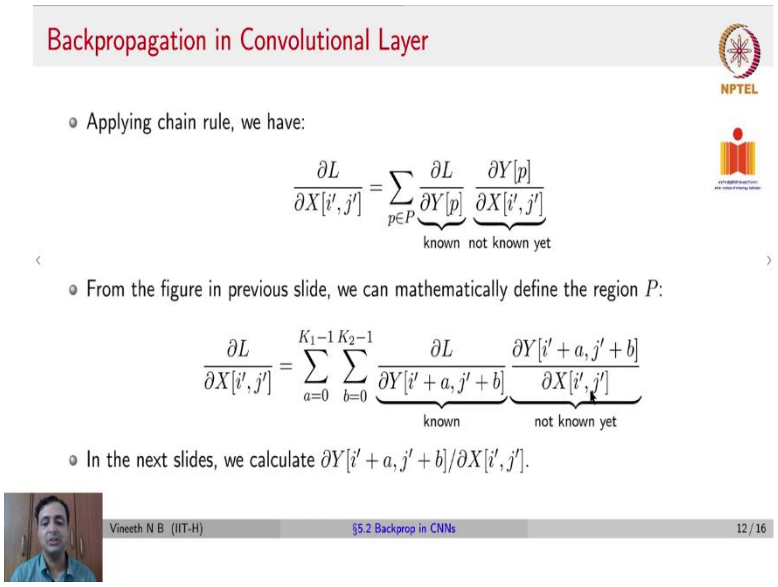
Vineeth N B (IIT-H) §5.2 Backprop in CNNs 11 / 16

Let us now move on to the next quantity which is $\frac{\partial L}{\partial X}$, gradient with respect to inputs of that layer of that Convolutional layer. Let us once again consider a single input pixel $X[i', j']$. Let us again ask the question which pixels in Y would be affected by this particular pixel in X? If we see it visually we can say that given an X and let us say there is 1 pixel given by this red square which is at $[i', j']$ when it is convolved with W you would now have this it would affect this entire range of pixels that go from $[i', j']$ to $[i' + K_1 - 1, j' + K_2 - 1]$.

Why is it like this, why is it not centered at $[i', j']$? Because that is the way we define convolution in this context for simplicity. If we had defined convolution as going from $-{}^{K_1}/_{2}$ to plus $-{}^{K_1}/_{2}$ it would have been centered but because this is the way we defined convolution the indices went from 0 to $K_1 - 1$ and b to $K_2 - 1$. You would now have these pixels here represented by this dotted square which are the pixels that would be affected by $[i', j']$.

Now, we call this region P and we now know that the gradients of the loss with respect to these pixels will be influencing the gradient of the loss with respect to this pixel. All other pixels in Y do not have a contribution from this pixel and hence those gradients of L with respect to Y do not matter to us.

(Refer Slide Time: 12:44)



Backpropagation in Convolutional Layer

- Applying chain rule, we have:

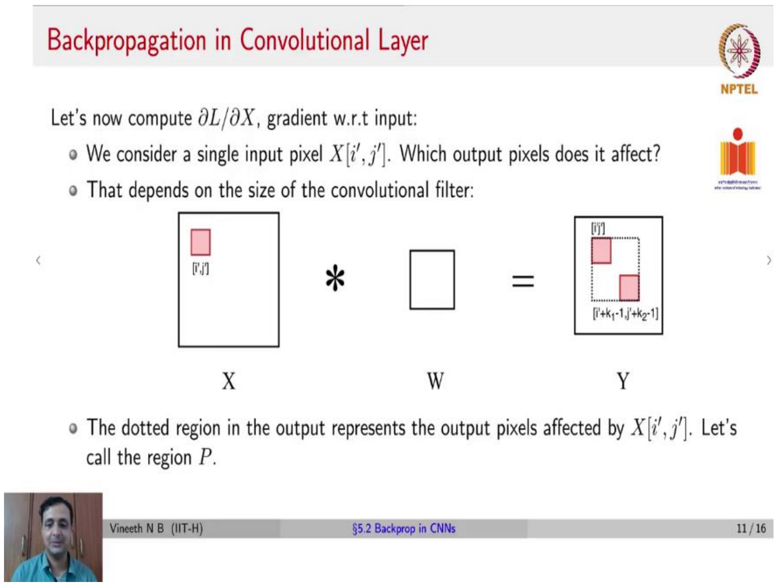
$$\frac{\partial L}{\partial X[i',j']} = \sum_{p \in P} \underbrace{\frac{\partial L}{\partial Y[p]}}_{\text{known}} \underbrace{\frac{\partial Y[p]}{\partial X[i',j']}}_{\text{not known yet}}$$

- From the figure in previous slide, we can mathematically define the region P:

$$\frac{\partial L}{\partial X[i',j']} = \sum_{a=0}^{K_1-1} \sum_{b=0}^{K_2-1} \underbrace{\frac{\partial L}{\partial Y[i'+a,j'+b]}}_{\text{known}} \underbrace{\frac{\partial Y[i'+a,j'+b]}{\partial X[i',j']}}_{\text{not known yet}}$$

- In the next slides, we calculate $\partial Y[i'+a,j'+b]/\partial X[i',j']$.

Vineeth N B (IIT-H) §5.2 Backprop in CNNs 12 / 16



Backpropagation in Convolutional Layer

Let's now compute $\partial L/\partial X$, gradient w.r.t input:

- We consider a single input pixel $X[i',j']$. Which output pixels does it affect?
- That depends on the size of the convolutional filter:

[i',j']
*
=
[i',j']
[i'+k$_1$-1,j'+k$_2$-1]
X
W
Y

- The dotted region in the output represents the output pixels affected by $X[i',j']$. Let's call the region P.

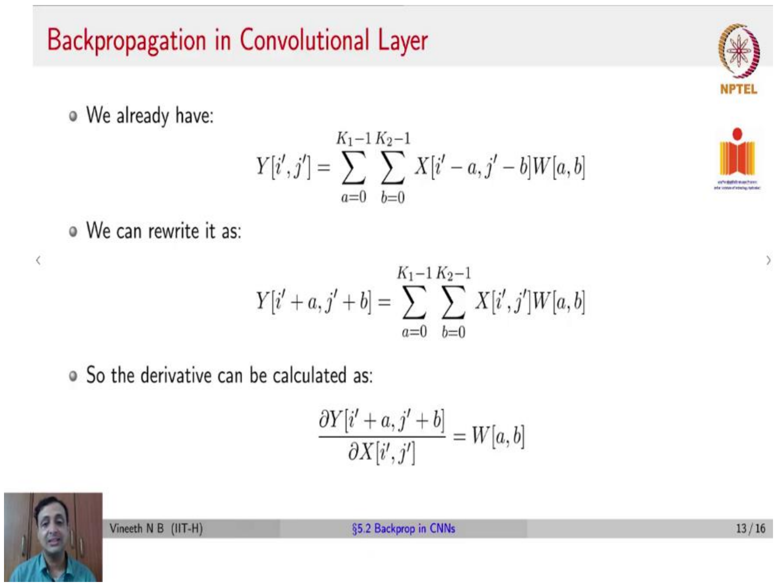
Vineeth N B (IIT-H) §5.2 Backprop in CNNs 11 / 16

Let us now say $\frac{\partial L}{\partial X[i',j']} = \sum_{p \in P}, \frac{\partial L}{\partial Y[p]} \frac{\partial Y[p]}{\partial X[i',j']}$. From the figure in the previous slide, we can now define that region P as going from 0 to $K_1 - 1$, b going from 0 to $K_2 - 1$. ∂L at a particular pixel $Y[i' + a, j' + b]$ and the derivative of $Y[i' + a, j' + b]$ with respect to $X[i', j']$ rather, if you see the previous slide we are taking each pixel here and adding them and all I am saying is that we want a gradient of a particular pixel in this region P with respect to this pixel at in X that is what we are writing in the summation.

The left quantity here is known because we assume that all the ∂Y until ∂Y are known until that time we are only worried about how to compute the back prop across a single convolutional layer. If there were more convolutional layers the same procedures would be applied iteratively.

But the second quantity here is currently not known to us $\frac{\partial Y[i'+a,j'+b]}{\partial X[i',j']}$. Let us try to compute this on the next slide.

(Refer Slide Time: 14:17)



Backpropagation in Convolutional Layer

- We already have:

$$Y[i',j'] = \sum_{a=0}^{K_1-1}\sum_{b=0}^{K_2-1} X[i'-a,j'-b]W[a,b]$$

- We can rewrite it as:

$$Y[i'+a,j'+b] = \sum_{a=0}^{K_1-1}\sum_{b=0}^{K_2-1} X[i',j']W[a,b]$$

- So the derivative can be calculated as:

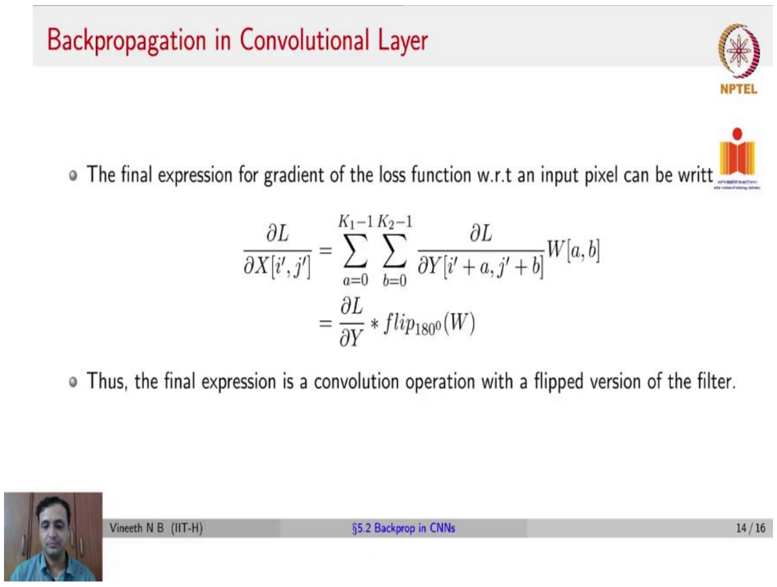
$$\frac{\partial Y[i'+a,j'+b]}{\partial X[i',j']} = W[a,b]$$

Vineeth N B (IIT-H) §5.2 Backprop in CNNs 13 / 16

From the definition of convolution once again we have $Y[i',j']$ given by this equation this comes again comes from the basic definition of convolution. So, if this was the definition of $X[i',j']$ then the definition of $Y[i'+a,j'+b]$, which is going to be another pixel location will be replaced wherever $i'$is thereby $i'+a$, replace where ever j' is thereby $j'+b$. So, $i'+a$ a becomes i', $j'+b-b$ becomes j' and $W[a,b]$ stays as it is.

Now, the $\frac{\partial Y[i'+a,j'+b]}{\partial X[i',j']} = W[a,b]$ for a particular choice of a and b, for a different choice of a and b it would be that term in the summation as we said the last for the previous derivative $\frac{\partial L}{\partial W}$ also, all the other terms in the summation would not affect the derivative with respect to 1 particular a, b. So, this is the quantity that we have for ∂Y at a particular pixel location with respect to $\partial X[i',j']$. Let us now plug this back

(Refer Slide Time: 15:45)



Backpropagation in Convolutional Layer

- The final expression for gradient of the loss function w.r.t an input pixel can be writt

$$\frac{\partial L}{\partial X[i',j']} = \sum_{a=0}^{K_1-1}\sum_{b=0}^{K_2-1} \frac{\partial L}{\partial Y[i'+a,j'+b]} W[a,b]$$
$$= \frac{\partial L}{\partial Y} * flip_{180^0}(W)$$

- Thus, the final expression is a convolution operation with a flipped version of the filter.

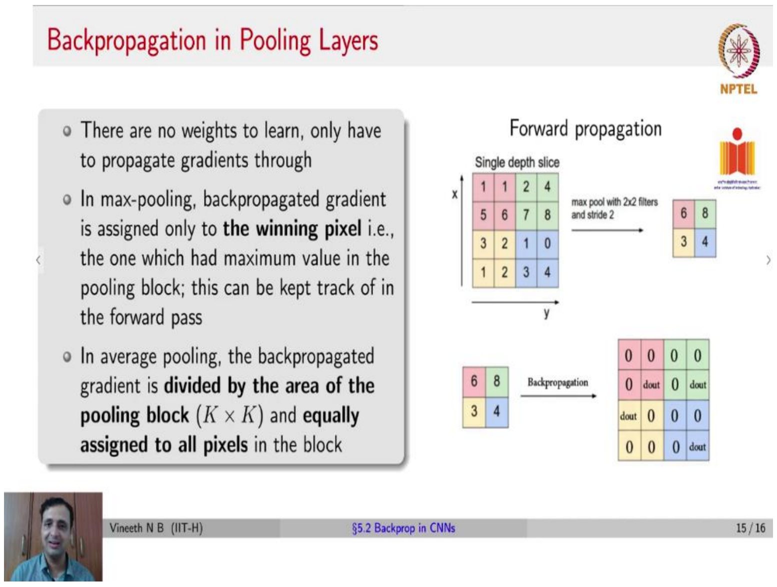
Vineeth N B (IIT-H) §5.2 Backprop in CNNs 14 / 16

We have $\frac{\partial L}{\partial X[i',j']}$ to be friends all the pixels in the region P $\frac{\partial L}{\partial Y}$ at one of those pixels into $W[a,b]$ for this choice of a and b. Once again, this becomes interesting. This looks like the definition of cross-correlation. It is the cross-correlation of $\frac{\partial L}{\partial Y}$ with W, the filter. Rather, we can say it is the convolution of the filter flipped with $\frac{\partial L}{\partial Y}$.

We know that cross-correlation and convolution are it differ by that flip of the filter and that become by a flip by 180 degrees which is a flip in 2 directions and that is what now we have here, $\frac{\partial L}{\partial Y}$ into a double convolution with a double flipped filter. This is interesting again because even now the chain rule across the convolutional layer to compute $\frac{\partial L}{\partial X}$ can be computed as a convolution.

Why does this matter? Recall our discussion of image frequencies in the first week. We said that convolution can be evaluated through very efficient methods such as fast Fourier transform. You could use all that to efficiently compute these gradients during backpropagation in this step.

(Refer Slide Time: 17:22)



Backpropagation in Pooling Layers

- There are no weights to learn, only have to propagate gradients through
- In max-pooling, backpropagated gradient is assigned only to **the winning pixel** i.e., the one which had maximum value in the pooling block; this can be kept track of in the forward pass
- In average pooling, the backpropagated gradient is **divided by the area of the pooling block** $(K \times K)$ and **equally assigned to all pixels** in the block

Now, we are felt with one question, so that defines, that concludes how backpropagation is done across a convolutional layer. There are only 2 quantities W and X. We are now still a felt with how do you backpropagate across a pooling layer? As we said before, a pooling layer is parameter-free. It does not have any weights. There are no weights to updates, no gradients. The only gradients we have to compute are the gradients with respect to X in the previous layer to allow the gradients to propagate through to earlier layers.

How do we do this? If we have max pooling, let us take this visual example on the right. So, you can see here that these red, green, yellow, and blue regions are max pooled in the forward pass, 1156, the max value is 6. 2478, the max value is 8. 3212, the max value is 3. 1034, the max value is 4. So, in the forward propagation, for every 2x2 region, you take the max element and put it here and then you take it further through later layers.

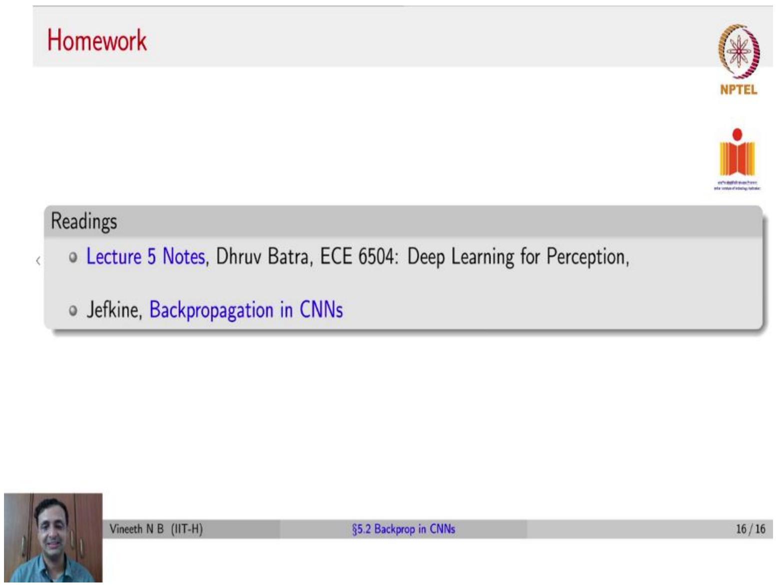
In the back-prop step, in the same layer, you keep track of what was the winning position in each 2x2 window. For example, we know that 6 came from this position of the first two crosses 2 windows. So, whatever gradient we have at 6 when we backpropagate goes to that location in the previous layer, backpropagation is drawn the other way here but this is the previous layer, what you see on the right here is the previous layer.

The full gradient at 6, remember we assume that when we backpropagate, the gradients until this step for each of this pixel is known to us, we only have to see how to backpropagate those gradients across the pooling layer. All these gradients go to this location, all these gradients go to this location, all these gradients go to this location because that was the position of the winning neuron and similarly for this to this.

What happens if we did not use max-pooling? If we use say, average pooling? If we used average pooling, you would take whatever gradient came into one particular location say the position of 6, distribute hat gradient equally to each of these 4 locations in case of 2x2 pooling.

If you did 3x3 pooling, you would divide that among 9 pixels. In this case, we have 2x2 pooling, so whatever gradient you have at 6 will be divided into 4 and that gradient would be given to each of these pixels in the 2x2 neighborhood that led to 6 and that is how backpropagation is done across pooling layers.

(Refer Slide Time: 20:23)



Homework

Readings

- Lecture 5 Notes, Dhruv Batra, ECE 6504: Deep Learning for Perception,
- Jefkine, Backpropagation in CNNs

Vineeth N B (IIT-H) §5.2 Backprop in CNNs 16 / 16

That concludes our discussion of backpropagation in CNNs. For more details please read the lecture notes of Dhruv Batra on the same topic or a very nice write-up by Jefkine on Backpropagation in CNNs.