

Deep Learning for Computer Vision
Professor. Vineeth N Balasubramanian
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
Lecture No. 32
Convolutional Neural Networks: An Introduction - Part 02

(Refer Slide Time: 0:14)



Pause and Ponder

- What is the connection between convolution and neural networks? Won't feedforward neural networks do?
- We will try to understand this by considering the task of "image classification"



Vineeth N B (IIT-H)

§5.1 Introduction to CNNs

18 / 37

Having reviewed convolution so far, let us try to ask the question, what is the connection between convolution and neural networks? If we want to use it on images, can't we use feed-forward neural networks? We will try to understand this more carefully by taking the example of image classification.

(Refer Slide Time: 0:44)

Traditional Machine Learning for Vision

- Traditional ML-based computer vision solutions involve static feature engineering from images (e.g. recall SIFT, LBP, HoG, etc)
- Though effective, static feature engineering was a bottleneck of pre-DL vision solutions

Static Feature Engineering (No Learning) Learning Weights of Classifier

Vineeth N B (IIT-H) §5.1 Introduction to CNNs 19 / 37

Let us try to understand how traditional machine learning was done earlier on computer vision. Given an image, traditional ML-based computer vision solutions for image classification would extract somewhat are known as handcrafted features from images, examples being SIFT, LBP, HoG, so on and so forth. Why are they called handcrafted? Because we came up with a procedure to extract those features, using some thought processes and heuristics. These were effective for quite a while.

But it was a bottleneck of scaling the solutions to more datasets and scaling the solutions to more real-world images. One would be given an image, you see the raw pixels, or extract edges, or extract features such as SIFT and HoG. It could be many others. This was known as Static Feature Engineering, where there was no machine learning or learning involved at all. These features were then provided to a classifier, such as a neural network, to finally give the output as say monument in this case.

(Refer Slide Time: 2:05)

Beyond Static Feature Engineering

- **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?

The diagram illustrates two parallel processes. The top process shows an input image of the Taj Mahal being processed by a single handcrafted kernel (edge detector) to produce a single feature map, which is then fed into a classifier. The bottom process shows the same input image being processed by multiple learned kernels (filters) to produce multiple feature maps, which are then fed into a classifier. A red arrow points to the feature maps in the bottom process with the text 'Learn these weights'.

Vineeth N B (IIT-H) §5.1 Introduction to CNNs 20 / 37

The question that we try to ask now is, instead of those handcrafted kernels, for edge detection for SIFT or any other method, can we learn meaningful kernels or filters to solve the problem? In addition to learning the weights or the parameters of the classifier? Rather, if you have an edge detector, which is a Laplacian of Gaussian, which has certain weights in its kernel, or any other kernel for that matter, can we automatically learn these weights as part of the learning algorithm, rather than separate these 2 as the first step is hand-engineered, and the second step being learning step. That is one of the first questions we ask.

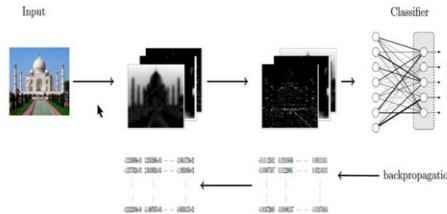
We could go even better and ask, can we learn multiple meaningful filters or kernels in addition to learning the weights of the classifier? Why just learn one, if we are learning, we may as well learn many of them.

(Refer Slide Time: 3:09)



Beyond Static Feature Engineering

- 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 backpropagation, discussed in the next lecture)



- Such a network is called a **Convolutional Neural Network!**



Vineeth N B (IIT-H)

§5.1 Introduction to CNNs



20 / 37

And in addition to that, can we also learn multiple layers of meaningful kernels and filters? We now know that convolution has some nice mathematical properties. And you can compose have a composition of convolution operations across different layers of a neural network. So, can we do that, in can we use that in some way to learn multiple layers of meaningful kernels or filters, before we learn the weights of the classifier or along with learning the weights of the classifier? This is possible. And this is possible by simply treating the kernels weights of the kernel coefficients as parameters and learning them, in addition to learning the weights of the final classifier.

So, you have to have you would have an input, you would have a convolution kernel, that gives you a certain output map, which becomes a second layer, you could have another convolution kernel, which would give you an output, which would give you the feature map at the third layer. And this can now be used as input to a final layer of fully connected neurons, which give us the final output. And we can use backpropagation to update these filter coefficients at each intermediate step. This network is called a convolution neural network.

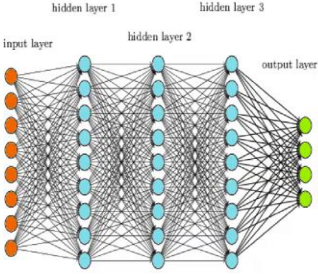
Why is that so? Because in each step, instead of taking a fully-connected bipartite graph to connect the first layer, the next layer, we are going to define a set of weights, which convolve on the input image and given output feature map and we are going to learn those weights using backpropagation.

(Refer Slide Time: 5:02)




Pause and Ponder

- Learning kernels/filters by treating them as parameters definitely is interesting
- But why not directly use flattened images with fully connected neural networks (or feedforward neural networks, FNNs) instead?





Vineeth N B (IIT-H) §5.1 Introduction to CNNs 21 / 37



Let us try to pause here and ask 1 or 2 questions again. So, learning filters seems interesting. But why not use flattened images with fully connected neural networks? Why do we want to complicate our lives and learn the filters? Why cannot we just take the raw pixels of the input image? Anyway, the neural network is going to learn the weights it should? Why should we? Why should that be a convolution operation in between? Why cannot the neural network simply take the pixels of the images as flatten them out and give them as the input to a neural network and let the layers learn whatever they should? Why cannot we do this?


(Refer Slide Time: 5:48)



Challenges of Applying FNNs to Images

On a reasonably *simple* dataset like MNIST, we can get about 2% error (or even better) using FNNs, but


- Ignores spatial (2-D) structure of input images – unroll each 28×28 image into a 784-D vector
 - Pixels that are spatially separate are treated the same way as pixels that are adjacent
- No obvious way for networks to learn same features (e.g. edges) at different places in the input image
- Can get computationally expensive for large images
 - For a 1MP color image with 20 neurons in the first hidden layer, how many weights in the first layer?
60 million!



MNIST Dataset

Credit: Steve Renals

Vineeth N B (IIT-H) §5.1 Introduction to CNNs 22 / 37



There are a few reasons let us try to evaluate them. If you take even a simple data set, such as what is known as the MNIST data set, so this is an example of the MNIST data set. The MNIST data set, as we mentioned earlier, is a data set for handwritten digits, which was used, which was used by the United States Postal Service in the 90s to automatically classify mail, based on zip code recognition from images.

So, if you use a feed-forward neural network without convolution, or without learning the filters on a data set such as MNIST, you get fairly good performance, close to about 98 percent, 90 percent performance with a simple feed-forward network. But there are certain limitations, and let us try to analyze them. The first limitation is this method ignores spatial correlations or the structures in images.



By taking pixels and flattening all of them into a single vector, we have removed the spatial relationships that exist in different parts of the image or different corners of the image. Spatially separate pixels are treated the same way, as adjacent pixels, we seem to be losing some important information that characterizes images. By doing this, that is the first concern.

The second concern here is, there is no obvious way for networks to learn the same features at different places in the input image. Unlike a few other machine learning problems in images, one is expected to recognize a cat on the image, whether the cat was located on the left top or the bottom right. If we simply flattened an image into a vector, a cat on the left top would have a very different vector representation, the dimensions in which the cat exists would be very different from a flattened vector when the cat was on the bottom right.

We are not enforcing the network to learn that a cat is a cat, irrespective of where it is in the image. And lastly, it can get computationally very expensive if you took an image, flattened it, and only used a fully connected feed-forward neural network. Why is that so? Let us take an example. If you had a 1-megapixel color image with 20 neurons in the first hidden layer, how many weights would you have in the first layer?

Think about it. And the answer is 60 million. 1 megapixel is 10^6 pixels. Color image, 3 channels, so that is 3 million pixels into 20 neurons, a fully connected graph is going to end up with 60 million weights in only the first layer, you now have to add layers probably for getting better performance and that is going to lead to an explosion in the number of weights.


(Refer Slide Time: 9:12)



How do Convolutional Neural Networks Solve these Challenges?

- Local receptive fields, in which hidden units are connected to local patches of the layer below, serve two purposes:
 - Capture local spatial relationships in pixels (which would not be captured by FNNs)
 - Greatly reduces number of parameters in the model
 - For a 1MP color image a filter size of $K_1 \times K_2$ in the first hidden layer, how many weights in a convolutional layer? $K_1 \times K_2$, compare with 60 million for FNNs on the previous slide!
- Weight sharing, which also serves two purposes:
 - Enables translation-invariance of neural network to objects in images
 - Reduces number of parameters in the model
- Pooling which condenses information from previous layer, serves two purposes:
 - Aggregates information, especially minor variations
 - Reduces size of output of a previous layer, which reduces number of computations in later layers

Credit: Steve Renals



Vineeth N B (IIT-H) | 5.1 Introduction to CNNs | 23 / 37

So, how do we overcome this? We overcome this by using the idea of convolution, where we know that a filter only operates on a local neighborhood in the original image. We call these local receptive fields. Rather the region of the input which is used for convolution is typically known as the receptive field. So, that receptive field is a local part of the image. So, each hidden unit of the next layer is connected to only 1 local part of the input image.

This serves a few purposes. Firstly, it captures spatial relationships because if you do convolution of a filter with a patch, you are capturing the 2D spatial relationships in that region. And such relationships may not be captured by feed-forward neural networks effectively, it greatly reduces the number of parameters in the model. Because now you only need to connect to a local region rather than have to have a fully connected graph where every neuron in the first layer is connected to every neuron in the second layer.


So, now if we take the same example, if you had a 1-megapixel colored image, and let us assume now that you have a filter size of $K_1 \times K_2$ that you want to learn in that first hidden layer, how many weights would you have now? The number of weights would simply be the size of the filter itself. As we said, when you use convolution in a neural network, the filter coefficients themselves become the weights. So in this case, the number of weights would be $K_1 \times K_2$ compare with the 60 million that we talked about feedforward networks on the previous slide.

Secondly, convolution neural networks introduce a concept of what is known as weight sharing. Weight sharing is, you take a filter, whether you convolve it with the top right, or the bottom left. You use the same values in the filter, you do not change the filter weights when you convolve with the top left, and when you convolve with the bottom right, which is what you would have done if you flattened out an image and used a fully connected layer to estimate the weights in that particular layer. This has 2 purposes too.

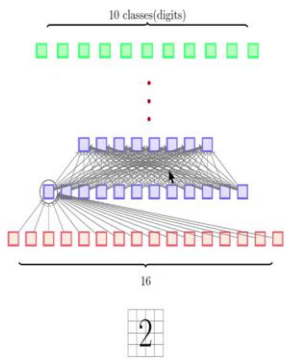
Firstly, as we just mentioned, it enables translation invariance of the neural network to objects in images. So as I said, whether the cat was on the left top or the bottom right, it is the same weights that are operating on them. So, it would recognize a cat be it in whichever part of the image. Secondly, it reduces the number of parameters in the model, as we just say. A third thing that CNNs also do is known as pooling. Pooling serves to condense information from the previous layer. This also serves 2 purposes.

One, by condensing information, it aggregates information, especially minor variations that are kind of average out to get a common output in the next layer. Secondly, pooling ends up subsampling an original image into a smaller image in the next layer. And by doing that, the next layer, when convolution is applied on that next layer, the number of convolutions you have to do reduces, which reduces the number of computations in later layers of the CNN.


(Refer Slide Time: 13:11)

 **Local Receptive Fields**

- This is what a regular feedforward neural network will look like
- There are many dense connections here
- All 16 input neurons are contributing to computation of h_{11}
- Let us contrast this to what happens in case of convolution




Vineeth N B (IIT-H) §5.1 Introduction to CNNs 24 / 37



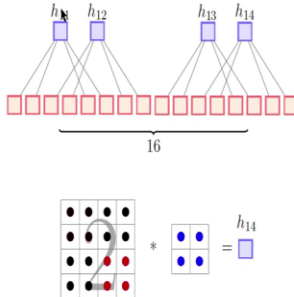
Let us look at each of these in more detail. Now. Let us start with local receptive fields. So to recap, CNNs have local receptive fields, weight sharing, and pooling. Let us review each of them in sequence. Let us start with the local receptive field. So, if we used a regular feed-forward neural network, it would look something like this. So, your input is these red neurons. Here, your input is a handwritten digit image.

Let us assume that you have because there are 16 pixels here, you have 16 input neurons. And then the blue squares here are the neurons in intermediate layers. And finally, on the output layer, you have 10 green neurons, which correspond to the 10 digits that you have. You can see here that every input neuron is connected to each hidden layer neuron here in the first hidden layer. So, that is a dense connection. So, and if you are now connected to all the input pixels in the first hidden layer, and so on, this becomes a very dense collection of weights, as we just mentioned. Let us contrast this with what happens if we do convolution.


(Refer Slide Time: 14:27)

 **Local Receptive Fields**

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



Vineeth N B (IIT-H) | §5.1 Introduction to CNNs | 25 / 37



So, if you observe here in this figure, once again, the input is 16 input neurons because that is the size of the input image. But now, only pixels 1, 2, 5, and 6. So, these 4 neurons here are the ones that contribute to the first convolution, pixels 3 and 4, and 7 and 8 do not contribute to that first convolution. So, only pixels 1, 2, 5 and 6, contribute to the computation of h_{11} which is the first pixel in the hidden layer.

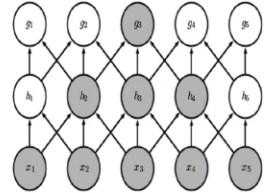
Similarly, as you move forward, you have a similar number of operations for each of your pixels in the hidden layer. How does this help? The connections now are much sparser. This sparse connectivity automatically reduces the number of weights, you have to learn, as we just mentioned. Importantly, we are also taking advantage of the structure of the image. As you can see, here, it is 1, 2, 5, and 6, that is part that participates in the h_{11} output, and 1, 2, 5, 6 are spatially correlated as compared to 1, 2, 3, 4, 7, 8 for instance. That is another advantage of the local receptive field.

(Refer Slide Time: 15:46)



Local Receptive Fields

- But is sparse connectivity really a 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



Vineeth N B (IIT-H)


§5.1 Introduction to CNNs

26 / 37

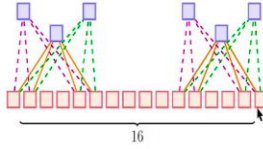
One question you could ask here is, but is sparse connectivity, really a good thing? Would not it be better to have full connections? Assuming we have infinite compute bandwidth and storage bandwidth and let the neural network learn what it should? By having sparse connections are we not disallowing certain pixels to interact and contribute to the output of the next layer? The answer is not really, we do not lose information through this process. Because as we go through the depth of such a neural network, if you look at the 2 highlighted neurons x_1 and x_5 .

Here, they may not interact, when we consider the first hidden layer, but they may start interacting to later hidden layers indirectly because x_1 contributes to h_2 ; x_1, x_2, x_3 contribute to h_2 , and h_2 and h_4 , where h_4 gets input from x_5 , contribute in the successive layer, and indirectly x_1 and x_5 interact, or the depth of the neural network. So, we are not losing out on this information, as long as we have a few layers in the neural network.


(Refer Slide Time: 17:07)

**Weight Sharing**

- Consider the following network; do we want the kernel weights to be different for different parts of the image?
- Not really. We would want the filter to respond to an object or an artefact in an image in the same way irrespective of where it is located in the image
⇒ **translation-invariance**
- We can have as many different kernels to capture different kinds of artifacts, but each one is intended to give the same response on all parts of the image
- This is called **weight sharing**



Vineeth N B (IIT-H) §5.1 Introduction to CNNs 27 / 37



The second is weight sharing. Let us try to understand this with an example. Again, let us say that pixels 1, 2, 5, 6 are connected through a set of weights to that first-pixel hidden neuron in the first hidden layer. And similarly, the last 4 pixels, are certain numbers 15, 16, 11, and 12. They are connected to the last pixel of that hidden neuron. We would like to ask the question, do we want these kernel weights to be different? Do we want them to be a different set of weights here and a different set of weights here?

The answer is no. We would want that filter to respond to an object or an artifact in the image, in the same way, irrespective of where it is present in the image. So, this is known as translation invariance. So, we want our neural network to be invariant to the translation of an object in an image from one position to another position. Secondly, we can also have different kernels to capture a different kinds of artifacts. So, it is not required that you must have 1 kernel for the first half first, top left part, and another kernel for the bottom right part, you have the same kernel, and you have another kernel to capture another kind of an artifact, maybe one of them will capture the value for a person, another of them will capture the nose of a person.

This idea of having the same weights for all parts of your input, which is deep, which comes as a default from convolution is known as weight sharing. This also reduces the number of parameters.

(Refer Slide Time: 18:59)

Convolutional Neural Network

- A typical CNN looks as follows:

Input Convolution Layer 1 Pooling Layer Convolution Layer 2 Pooling Layer FC 1(120)FC 2(84)Output(10)

$S = 1, F = 5, K = 6, P = 0, \text{Param} = 150$

$S = 1, F = 2, K = 6, P = 0, \text{Param} = 0$

$S = 1, F = 5, K = 16, P = 0, \text{Param} = 2400$

$S = 1, F = 2, K = 6, P = 0, \text{Param} = 0$

Param = 48120 Param = 10164 Param = 850



- It has alternate convolution and pooling layers
- What do pooling layers do?

Vineeth N B (IIT-H) §5.1 Introduction to CNNs 28 / 37

Let us now try to see how our CNN looks in completion. Here is a sample CNN convolution neural network, given an input, you have a convolution on layer 1, which can have many filters. So stride is 1, size of the filter $F=5 \times 5$, $K=6$. That is the number of filters. So, the number of feature maps so the depth of the output in that first layer, P is equal to 0 padding is equal to 0. And the total number of parameters that you see here is 150, parameters 150 here. Then you have something called a pooling layer, and then a convolution layer, then a pooling layer.

And then you have what is known as FC layers or fully connected layers. Let us now try to understand what these pooling layers in between are. So, you can see here that most CNNs have alternate convolution and pooling layers, at least the initial CNN that was designed had it this way. These days, there are other options, more layers that you can fit in into an architecture of CNN, which we will revisit later in this week's lectures. Let us now try to understand what a pooling layer does?

(Refer Slide Time: 20:13)



Pooling Layer

Input * 1 filter =

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

maxpool
2x2 filters (stride 2)

8	4
7	5


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

maxpool
2x2 filters (stride 1)

8	8	4
8	8	5
7	6	5

- **Pooling** is a parameter-free down sampling operation
- Instead of **Max Pooling**, we can also do **Average Pooling**, L_2 **Pooling**, etc
- Other notable mentions: Mixed Pooling (combines max and average pooling), Spatial Pyramid Pooling, Spectral Pooling - we'll see some of these in later lectures

Vineeth N B (IIT-H) §5.1 Introduction to CNNs 29 / 37



A pooling layer is a parameter-free downsampling operation, subsampling, or a downsampling operation, and it is parameter-free, with no weights involved, you do not need to learn any weights in this particular layer. Let us see an example of one such pooling operation. So, you can see here that this could be a feature map, or it could be an output of a convolution step, which is a feature map in one of the hidden layers, which have switches of size, say 4x4 if we now try to do what is known as max pooling with stride, 2, here is how it would look.

See here, that you get 1, 4, 5, and 8, you do what is known as max pooling, which means you take the maximum value from that, which is 8, and you put that here. Then you do a stride 2, which means you skip 1 pixel in between and then go to the next 1. Once again, you take this 2x2 window, the max value, there is 4, and you put the 4 here. If you see down, we have the same max pool operation, same 2 cross 2 filters, but now with a stride of 1.

So, when you stride 1, you go to the immediate neighbor, and you do not skip a step and go to pixels later. So in this case, you can have 8 as the max pool pixel here. And similarly, when you go to the next step, the maximum value, here again, is 8, and you put 8 in the corresponding output in that layer. Let us complete this operation. Similarly, you go to the next step, and you have to stride you're to go to the next row in stride 1, you now have to still complete that row.


And you keep repeating this over every step, all that you have to do is in that 2 by 2 window that you are looking no weights involved, simply take the max element and put it in that position in

the next layer. As you can see, this turns out to become a sub-sampling operation, which means it would reduce the size of whatever image or feature map that you had here to a smaller size based on your choice of filters, and choice of stride.

In this example, we saw what is known as max pooling, where you take the max value in a 2 cross 2 window. But you have other kinds of pooling operations to where you can take you can do what is known as average pooling, or L2 pooling. Average pooling is where you take the average of values in a given 2x2 window. L2 pooling is where you take the L2 norm of all the weights in a given 2x2 window, and so on.

There are many more cousins of pooling, such as mixed pooling, which combines max and average pooling, that is spatial pyramid pooling, spectral pooling, so on and so forth. And we will visit some of these as we go forward to later lectures in this course.

(Refer Slide Time: 23:19)



Other Variants of Convolution: Dilated Convolution

- Introduces another parameter to convolutional layer called **dilation rate**
- Controls spacing between values in a kernel
- Figure shows 3×3 kernel with dilation rate 2
- Notice that dilated rate 1 is standard convolution
- A subtle difference between dilated convolution and standard convolution with stride > 1 , what is it?

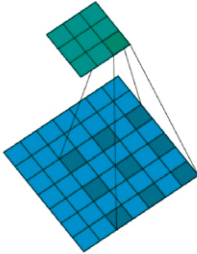


Image Credit: Vincent Dumoulin



Vineeth N B (IIT-H)

§5.1 Introduction to CNNs

30 / 37


In addition to these 3 operations, that is local receptive fields, weight sharing, and pooling. There are also variants that one can introduce in these layers that we discussed specifically, in the convolution layer of a convolution neural network. Let us see a few of these variants. Before we close this lecture. A popular variant is known as dilated convolution, where there is a new parameter introduced, known as dilation rate, which as the name says, controls the spacing between values in a kernel. So, when you apply a kernel on an input image, you change the spacing in how the values are applied, or convert the kernel with respect to the input.

So let us see an example here of a 3x3 kernel that is green in color, the blue is the input image with a dilation rate of 2. Let us see how it works. You can see now that when you convolve you are taking not you are not placing a 3x3 kernel directly on a 3x3 window of the input, produce spacing things out, and taking your placing the 3x3 kernel on that spaced out image.

One could also say this is equivalent to subsampling your original image and then applying by a dilation rate and then applying the same standard convolution. Both of them turn out to be equivalent, but we are doing it in 1 step. Now, remember, we already said sampling and interpolation can be interpreted as conventional operations. We just taking advantage of that now using a dilated convolution. So, that is the animation of how dilated convolution looks.

You can notice now, that when the dilation rate is 1, it becomes standard convolution. There is a subtle difference between dilated convolution and standard convolution with say stride 2 what is it? When you do standard convolution with saying stride 2, the convolution is still with the original neighbors neighboring pixels in the image, just that the next convolution goes 2 pixels further, but a dilated convolution, each convolution itself is dilated and sees a larger neighborhood in the input image.


(Refer Slide Time: 25:54)



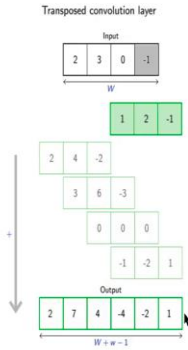
Other Variants of Convolution: Transpose Convolution

- Allows for learnable upsampling
- Also known as Deconvolution (bad) or Upconvolution
- Traditionally, we could achieve upsampling through interpolation or similar rules
- Why not allow the network to learn the rules by itself?
- Let us see a 1D example

Credit: Francois Fleuret



Vineeth N B (IIT-H) §5.1 Introduction to CNNs 31 / 37



Another popular variant of convolution that is used while training neural networks or CNNs is known as transpose convolution. As the name suggests, transpose convolution allows upsampling of an input. What do we mean? In traditional convolution, you have an input image,

you take a filter, and if you do not pad, you are going to get an output that is smaller in size. We are going to transpose that now.

We are going to see if given an input image, and if you convolve can you get a bigger image without doing anything else? And that process is what is known as transpose convolution. It is sometimes also called deconvolution or convolution. Although it is not technically deconvolution, deconvolution means a different thing in a signal processing context, although certain articles call such an operation deconvolution.

Traditionally, we would achieve upsampling using interpolation or similar rules. We saw this in the first week of lectures, that one could take something like a tenth kernel, and convolve that with an image to get a higher resolution image, or up to sample the image. We try to now ask, why should we use the tenth kernel? Why cannot we learn the weights of that interpolation kernel? Let us try to see how this transpose convolution is done using an example.

So, you have an input. Let us see a 1D example just to keep things simple and be able to easily explain, let us consider a 1D example, which has a dimension of 4 (2, 3, 0, -1) are the values in this input. Let us consider the kernel to also be 1D, which is 1, 2, -1. Now we want to find out how do we get an input which is larger than the output, which is larger than the input, let us see how we do it.

We take the 1, 2, -1, and convolve it with the first value of the input. And that gives you 2, 4, -2. Then we slide 1, 2, -1 to the second value of the input. And we now get 3, 6, -3. We further slide 1, 2 -1 to the next location and input get 0, 0, 0. And finally, we get -1, -2, 1. Now in each of these locations, we add up the values that we get, as we convert this filter with each location of the input.

So, your final output would look like 2, $(4+3=)7$, $(-2+6+0=) 4$, $(-3-1=) -4$, $(0-2=) -2$, and 1. This gives you the output which is now larger than the input. And you could do the same thing, even in 2 dimensions to make an image larger. We will see examples of where this is used a little later in this course.

(Refer Slide Time: 29:08)

Other Variants of Convolution: Transpose Convolution

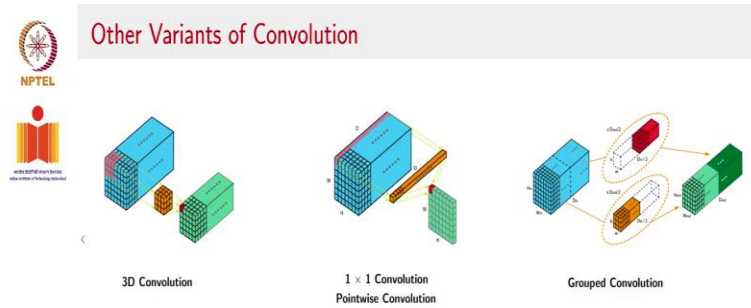
Upsampling 2×2 input to a 4×4 output Upsampling 2×2 input to a 5×5 output

GIF Credit: Vincent Dumoulin
Vineeth N B (IIT-H) §5.1 Introduction to CNNs 32 / 37

Let us understand this with a visual illustration. See, you can see 2 examples here, 1 of upsampling, a 2×2 input to a 4×4 output, and another upsampling for 2×2 input to a 5×5 output. In these images. The blue image in the middle is the 2×2 input. And the green image is the output. The shaded regions show the filter and how they are applied. So, you can see here that you first apply the 3×3 filter on just 1 pixel of your input and that gives you one of the values in the output.

Similarly, you move the filter on locations, and in this case, the last 2 values of the filter get multiplied by 2 locations. In the 2×2 input, and that gives you the value in the second location at the 4×4 output. In the 5×5 output scenario, you leave gaps in the 2×2 image. And so your values are going to be slightly different here. But you see this sliding over. And as you move that 3×3 filter over different parts of the 2×2 image, you get different values for your, for each pixel in your output image.

(Refer Slide Time: 30:34)



Credit: Illarion Khlestov, Chi-Feng Wang



Vineeth N B (IIT-H)

§5.1 Introduction to CNNs

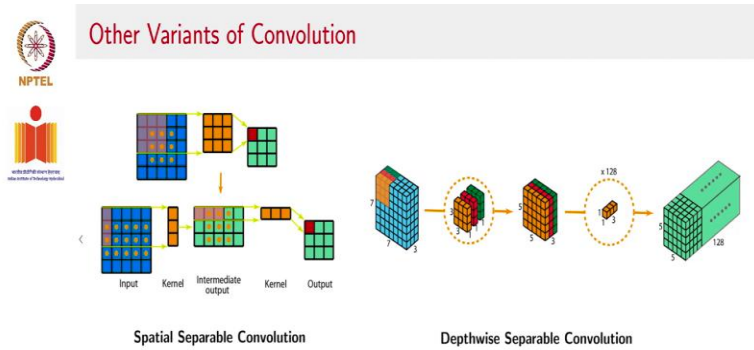
33 / 37

There are other kinds of convolution that are also possible. Some of these are something that we will see in detail when we talk about their applications in different contexts. But let us briefly review them now. 3D convolution, as we already mentioned, is where you do 3D convolution, and you get the output also to be a 3D volume. In the scenario that we talked about in this lecture, we did have our filter to be 3D, but our output feature map was 2D, and we made it 3D by taking many filters or many feature maps.

But what if we ensure that the output is a 3D volume, by the nature of convolution itself, that is 3D convolution. We also will talk about a 1x1 convolution where you only convolve along with the depth of a volume and that becomes 1 single value in the output. So, an input of $W \times H \times B$ becomes $W \times H$, because you convolve the entire depth along 1 particular pixel with a depth kernel and you get one scalar in that particular location.

You can also have grouped convolution where different filters convolve with different depths in your input. So, if you had about say 100 channels in your input, 1 set of filters could interact with the first 10 channels, then another with the next 10 channels, so on and so forth, that is grouped convolution.

(Refer Slide Time: 32:13)



Credit: Chi-Feng Wang

Vineeth N B (IIT-H)

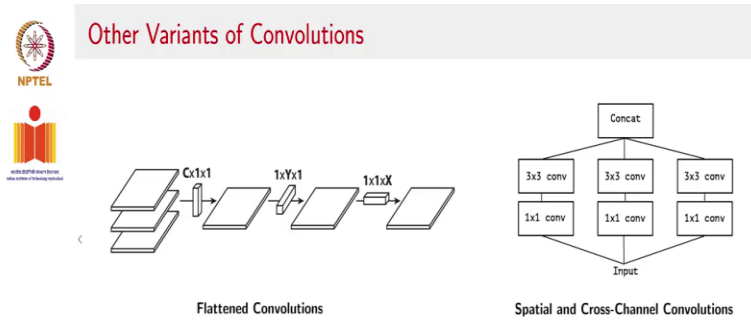
§5.1 Introduction to CNNs

34 / 37

We can also have separable convolution, the way we saw it in the first week, the way we could separate convolution in 2 dimensions as convolution along 1 dimension, followed by convolution along the second dimension. Recall that we said that this could help reduce computations, which is what we talked about here. So given an input, if you originally had a 3x3 kernel, to get us the 3x3 output, you first have a column kernel, get an intermediate output, then have a row kernel and then you get your final output. This is just a visualization of separable convolutions that we saw earlier in this course.

We could also have what is known as Depthwise Separable Convolution, were given an input volume, you have filters in 3 different channels for 3 different channels. And you can go through each of them individually to get 3 different feature maps. And then you run a depth convolution to compress all of them into a single feature map. Now, this can be repeated for different filters to get multiple filters to feature maps in this next layer.

(Refer Slide Time: 33:36)



Credit: Illarion Khlestov



Vineeth N B (IIT-H)

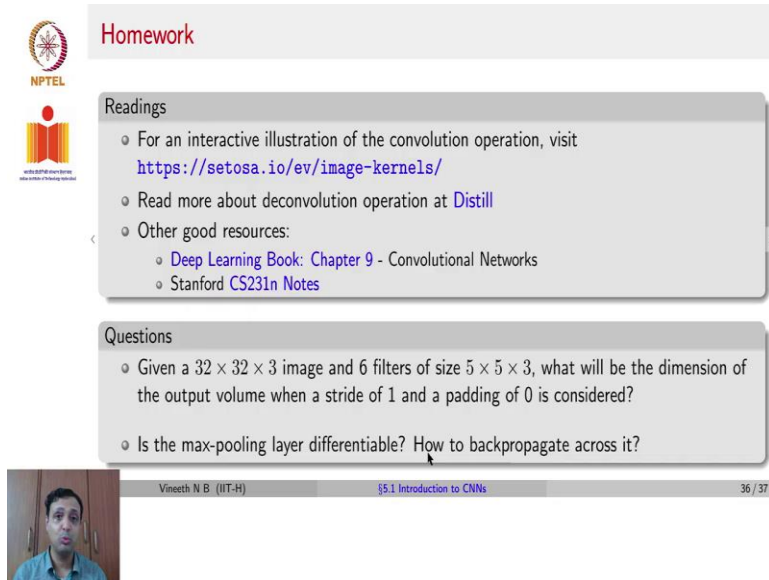
§5.1 Introduction to CNNs

35 / 37

We can also look at what is known as flattened convolutions which are very similar to separable convolutions where you first have a filter which is along 1 of the dimensions, then have a filter which is along another dimension, and then have a third filter along a third dimension. In some sense, this is a combination of separable convolution and depthwise convolution, this is just the general paradigm for that.

And finally, we can also have spatial and cross-channel convolutions where we can do convolutions in parallel and then concatenate them to get an output. We will see examples of several of these types of convolutions in later lectures this week, in the context of how they are used.

(Refer Slide Time: 34:27)



The screenshot shows a slide titled "Homework" with two main sections: "Readings" and "Questions".

Readings

- For an interactive illustration of the convolution operation, visit <https://setosa.io/ev/image-kernels/>
- Read more about deconvolution operation at [Distill](#)
- Other good resources:
 - Deep Learning Book: Chapter 9 - Convolutional Networks
 - Stanford CS231n Notes

Questions



- 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?
- Is the max-pooling layer differentiable? How to backpropagate across it?

At the bottom of the slide, there is a small video feed of a man speaking, and a footer with the text "Vineeth N B (IIT-H) §5.1 Introduction to CNNs 36 / 37".

In conclusion of this lecture, here are your recommended readings. For an interactive illustration of convolution, please see this link. For a very nice discussion of the deconvolution operation, please see this link at distill.com. Other good resources are chapter 9 of the deep learning book and certain notes are the CS231n course. A couple of questions for you to take away given a $32 \times 32 \times 3$, image and 6 filters of size $5 \times 5 \times 3$. What is the dimension of the output with a stride of 1 and a padding of 0?


Work it up. Another question here is, is the max-pooling layer differentiable? How do you backpropagate across it? Think about these questions at the end of this lecture.

(Refer Slide Time: 35:23)



References

- David Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". In: *International Journal of Computer Vision* 60 (Nov. 2004), pp. 91–110.
- Navneet Dalal and Bill Triggs. "Histograms of oriented gradients for human detection". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)* 1 (2005), 886–893 vol. 1.
- Svetlana Lazebnik, Cordelia Schmid, and J. Ponce. "Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories". In: vol. 2. Feb. 2006, pp. 2169–2178.
- Kaiming He et al. "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition". In: *Lecture Notes in Computer Science* (2014), 346–361.
- Dingjun Yu et al. "Mixed Pooling for Convolutional Neural Networks". In: Oct. 2014, pp. 364–375.
- Oren Rippel, Jasper Snoek, and Ryan P. Adams. "Spectral Representations for Convolutional Neural Networks". In: *NIPS*. 2015.
- Nal Kalchbrenner et al. "Neural Machine Translation in Linear Time". In: *ArXiv abs/1610.10099* (2016).



Vineeth N B (IIT-H) §5.1 Introduction to CNNs 37 / 37

And here are some references.