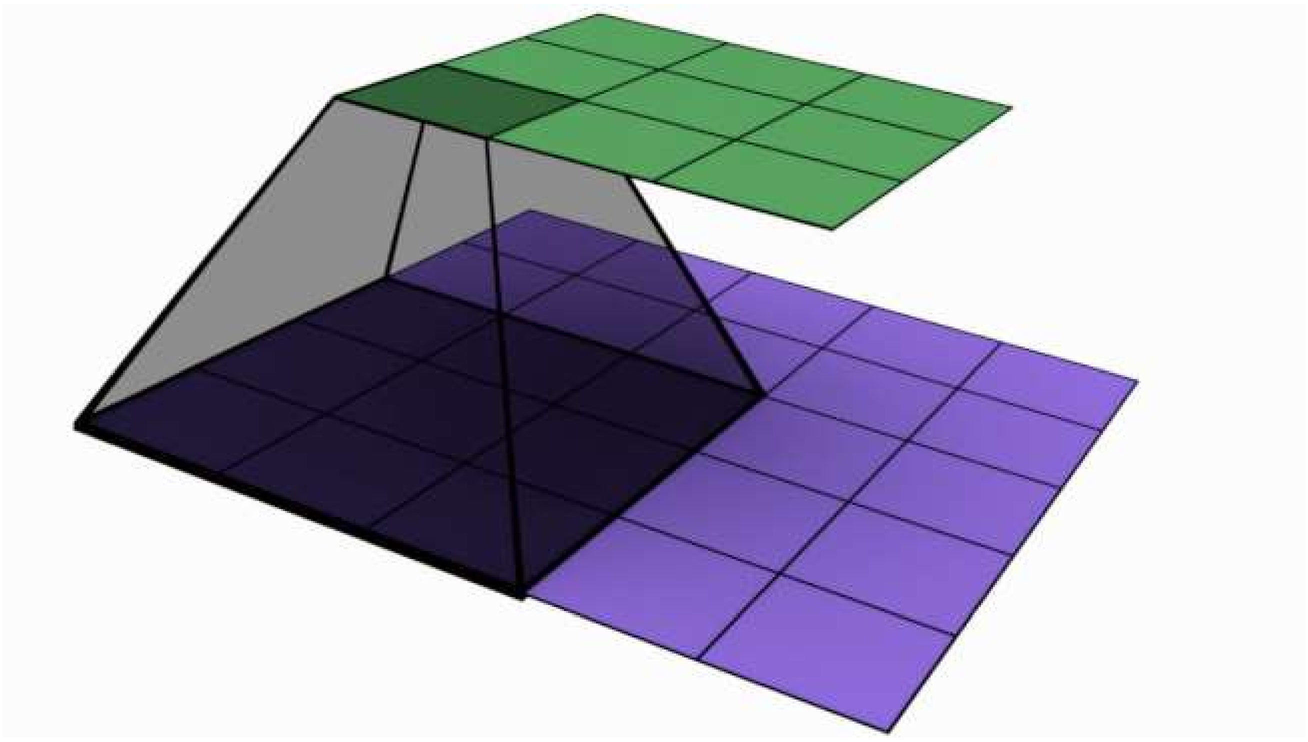
Session 2 - Exploring Neural Network Architectures

- Due Feb 3 by 9am
- Points 0
- Available after Jan 27 at 12am

Session 2 - Exploring Neural Architectures

- Concepts from Session 1
- Why do we add layers?
- Receptive Fields
- Convolution Mathematics
- Max pooling
    - Max pooling invariances
- Layer Count
- Kernels in Layers 1
- 3x3 is misleading
- Multi-Channel Convolution
- Out Network NOW
- Assignment

Concepts from Session 1



Convolving a 3x3 kernel on a 5x5 image/channel.

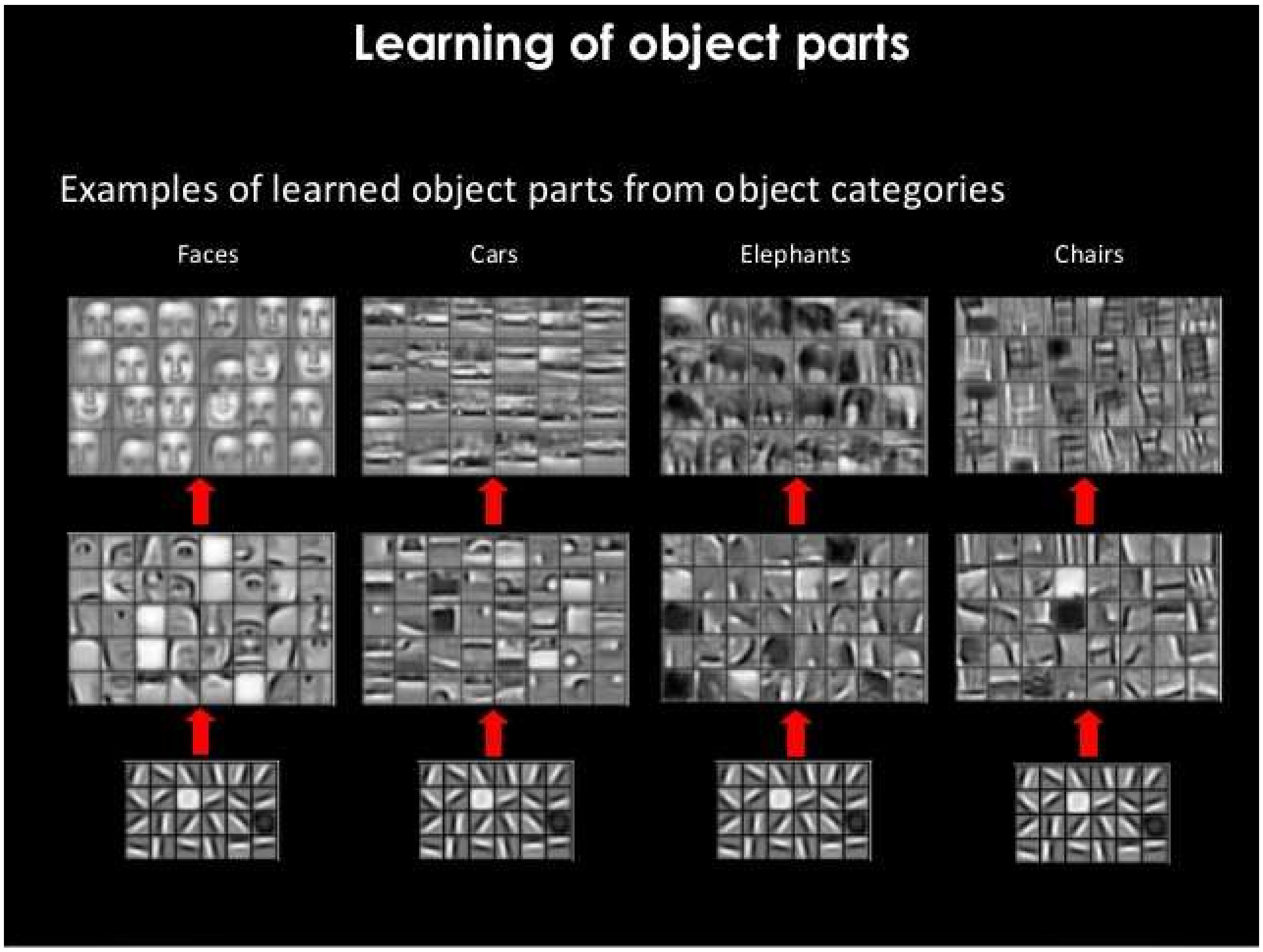
Here we see a 3x3 kernel convolving on an image/channel of size 5x5.

Every purple pixel we see represents a value in the image or last channel. **Nearly always we'll have more than 1 channel in the last layer.** These values are generated from the multiplication of the values of the channel**s** in the last layer by the values of the kernels in the last layer. We can only change the kernel values if want to make changes to these channels.

The dark purple 3x3 moving box is our kernel (also called filter, 3x3 matrix, feature extractor, and convolution parameter). We initialize it (like any other parameter) randomly. We **do** have control over the values of these kernels, and that is what backpropagation would change, such that it becomes useful as a feature extractor for us.

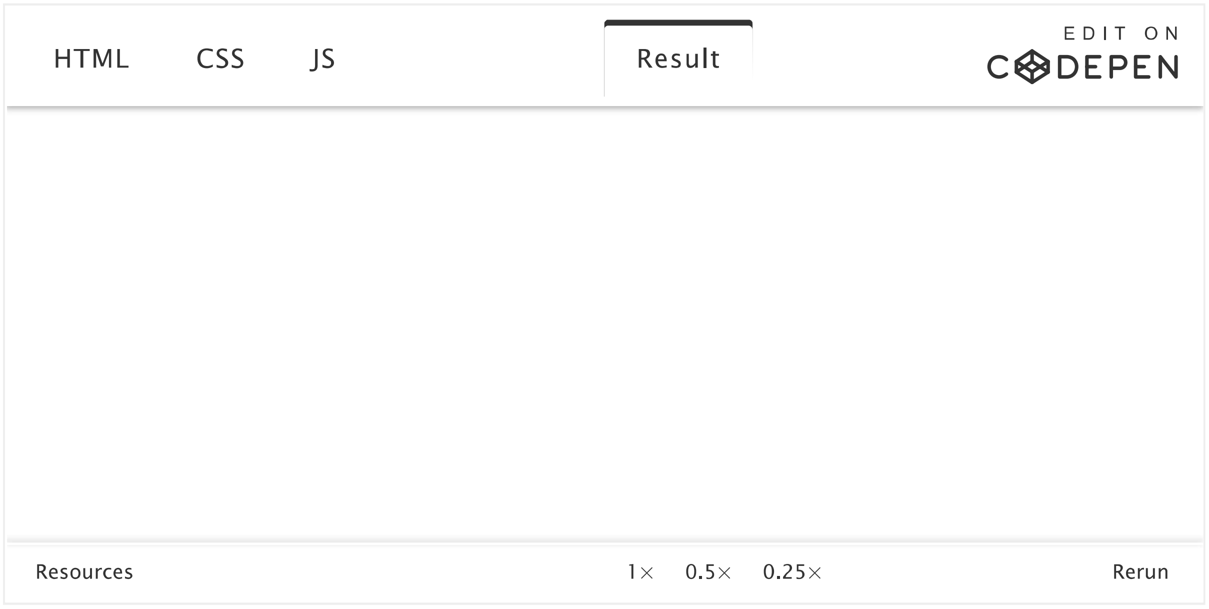
As you can see in the animation above, when we convolve on 5x5 by 3x3, we lose 2 pixels. This is not a concern for large channels, but changing the size of the channels is indeed an issue because it becomes difficult to track the channel size in a large network.

Why do we add layers?

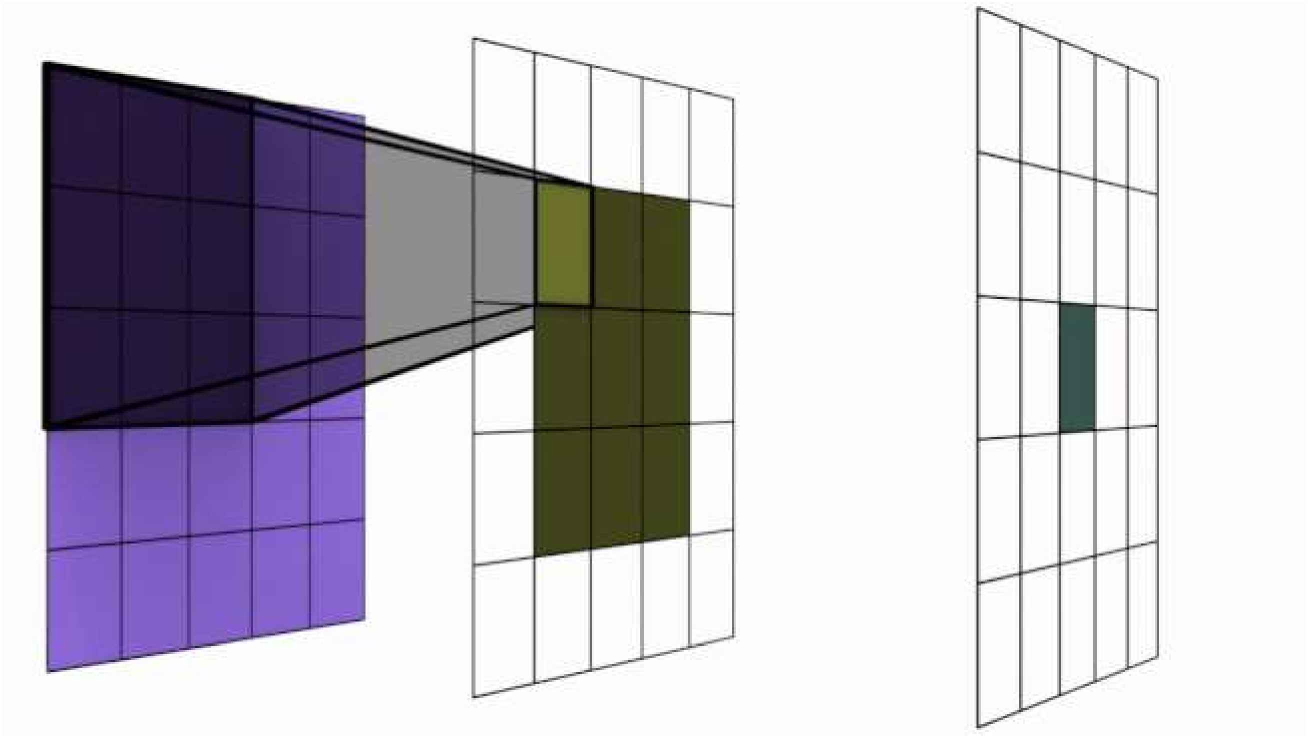


We have an objective (say detecting an object), and we can do that easily if we can detect the parts of the objects. Parts of the objects can be built from some patterns, and these patterns in turn can be made from textures. To make any kind of texture, we would need edges and gradients. We add layers to procedurally do exactly this. We expect that our first layers will be able to extract simple features like edges and gradients. The next layers would then build slightly complex features like textures, and patterns. Then later layers could build parts of objects, which can then be combined into objects. This can be seen in the image above.

As we progressively add layers, the receptive field of the network slowly increases. If we are using 3x3 kernels, then each pixel in the second layer has only "seen" (receptive field) 3x3 pixels. Before the network can take any decision, the whole image needs to be processed. We add layers to achieve this. Also, consider the fact that required or important edges and gradients can be made or seen within 11x11 pixels in an image of 400x400. But say, we were looking at a face, the parts of the face would take much more area (or the number of pixels).



Receptive Fields & Attentions



Here we see our first layer as a 5x5 channel. We are convolving on this 5x5 channel with a kernel of size 3x3, and hence the resulting output resolution will be a channel with 3x3 pixels/values. We also 2 2 layers here.

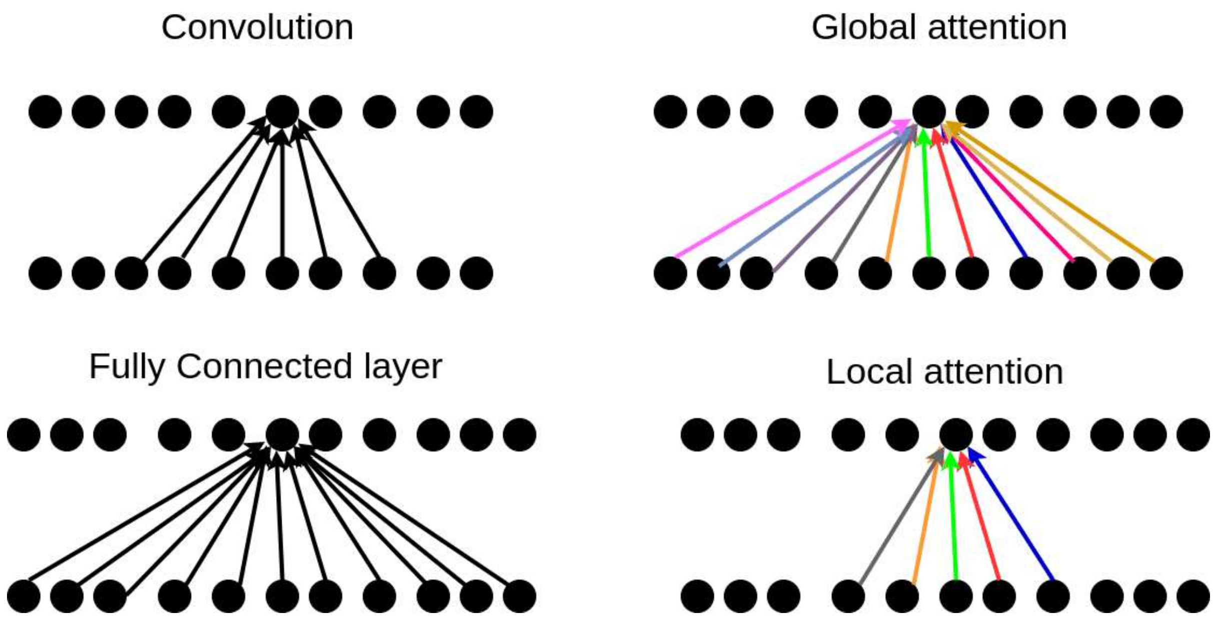
The receptive field in a Deep Neural Network (DNN) refers to the portion of the input space that a particular neuron is "sensitive" to, or in other words, the region of the input that influences the neuron's output.

In Convolutional Neural Networks (CNNs), the receptive field is determined by the spatial extent of the filters and the strides of the convolutional operations. Larger receptive fields lead to neurons that have a wider field of view, allowing them to capture more complex relationships in the input data.

To get the final output of 1 or 1x1, we could have used a 5x5 kernel directly. This means that using a 3x3 kernel twice is equivalent to using a 5x5 kernel. This also means that two layers of 3x3 have a resulting receptive field of 5x5.

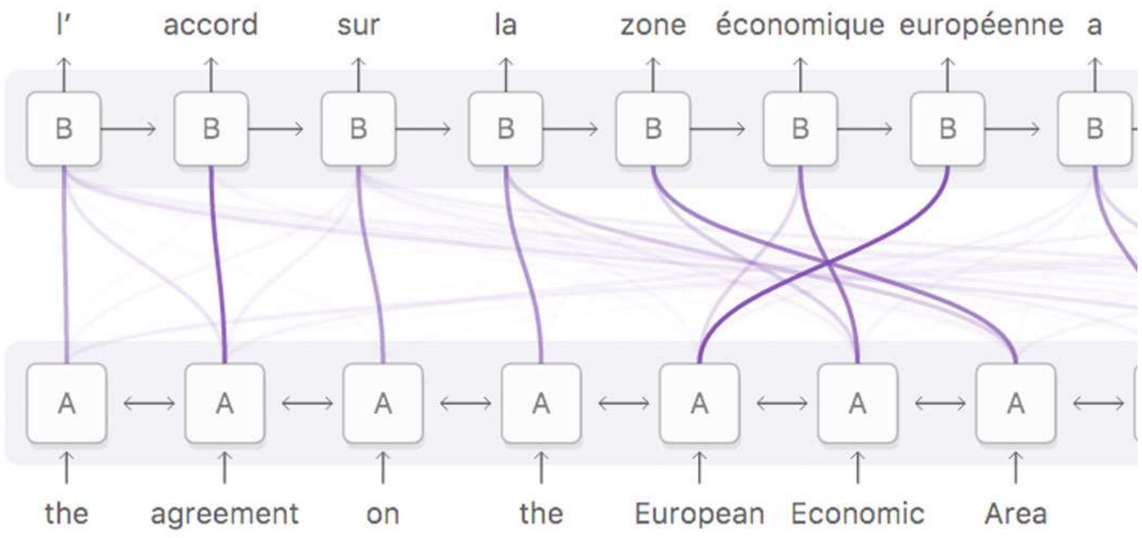
As we have discussed in class, we want the final global receptive field (at the final prediction layer or output layer) to be equal to the size of the image. This is important as the network needs to "see" the whole image before it can predict exactly what the image is all about.

This would mean that we need to add as many layers are required to reach the final receptive field equal to the size of the object. Since we have decided to consider the size of the object to be equal to the size of the image (for the first few sessions), our final receptive field is going to be the size of the image. (We know this is not true, images can have objects of any size, but we need to consider this restriction to build our concepts. Later we would work on what needs to be done to remove this restriction).

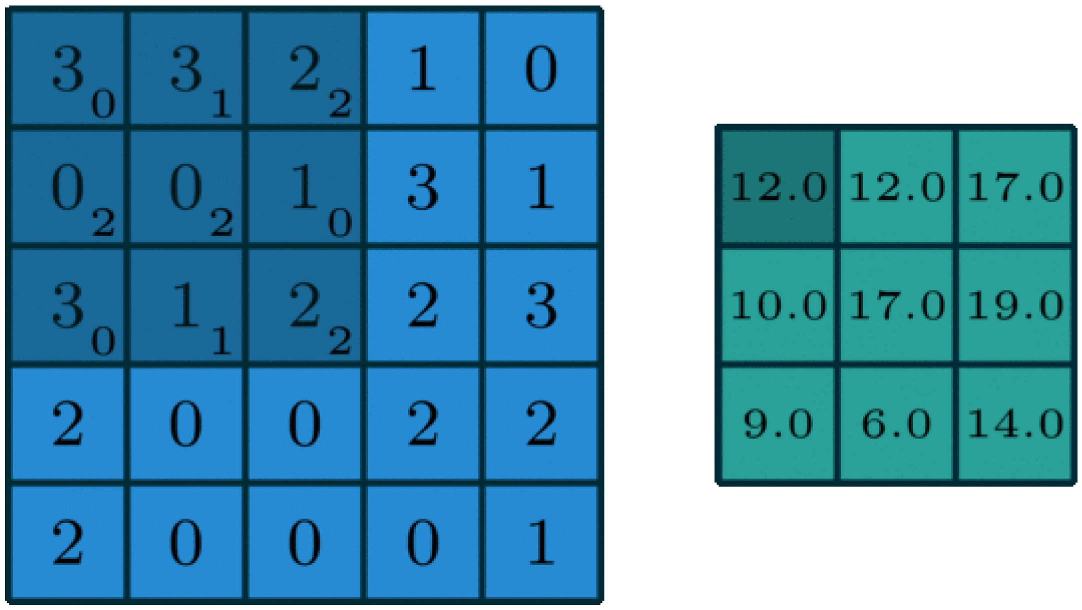


As we can see above, there are different kinds of attention spans or receptive fields. Convolution is, rather, a local operator (till the last layer). FC layer on the other hand has a full Receptive Field or Attention span. Full Attention is expensive, and local attention is limiting. We will solve this conundrum in 14-18 sessions.

The attention span for FC layers is 100%, and that allows it to get weighted information from any of the earlier neurons



Convolution Mathematics



We can see above that our 3x3 kernel has these values:

0 1 2

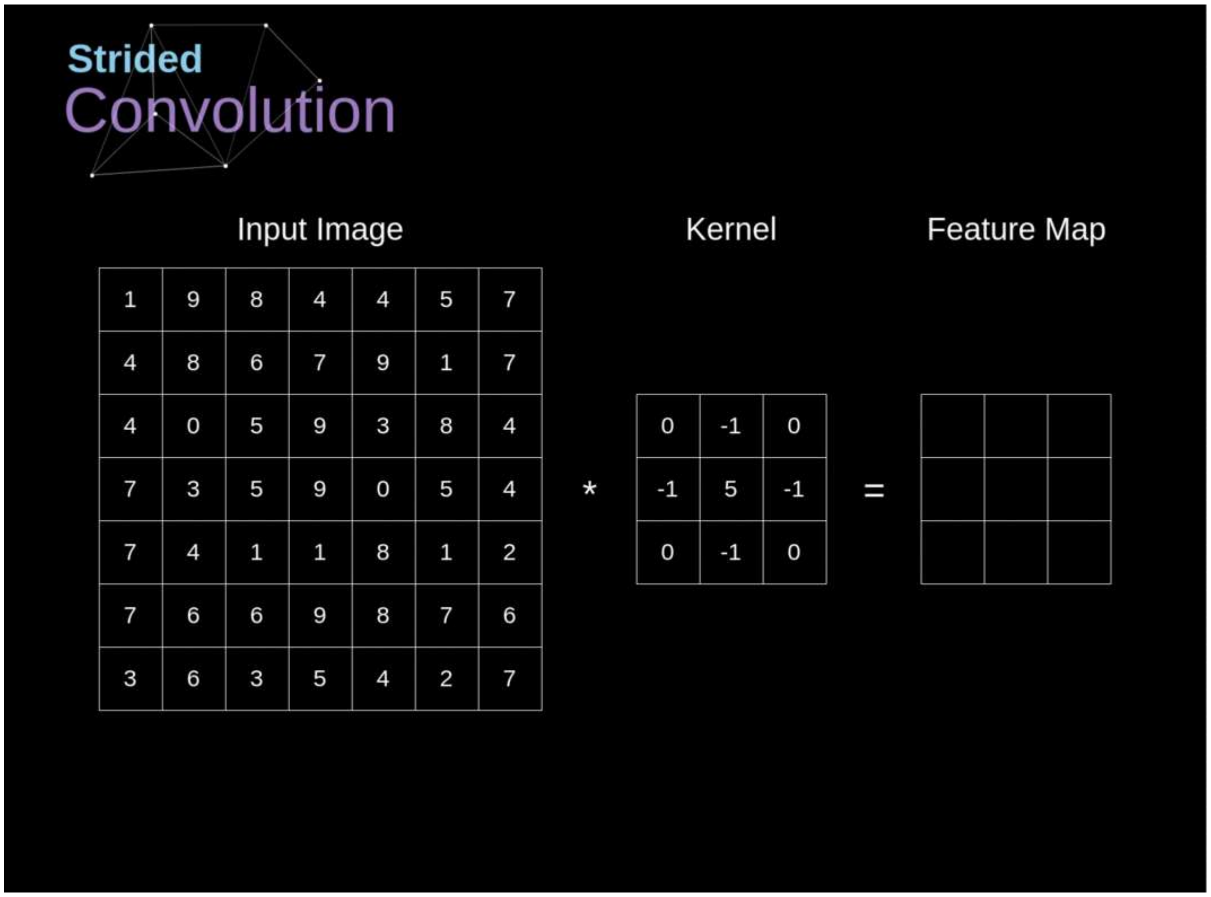
2 2 0

0 1 2

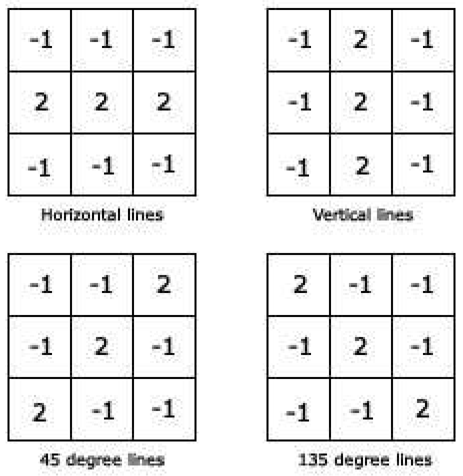
Whenever our kernel is stopping on a 3x3 area, we are looking at 9 multiplications and the sum of the resulting 9 multiplications being passed on to the output (green) channel as shown in the image above.

The values in the output channel can be considered as the "confidence" of finding a particular feature. The higher the value, the higher the confidence, and the lower (or more negative) the value, the "higher" the confidence in the non-existence of the feature.

A Strided convolution would look like this:



Some examples of edge detectors would be:



-1	-1	-1
2	2	2
-1	-1	-1

Horizontal lines

-1	2	-1
-1	2	-1
-1	2	-1

Vertical lines

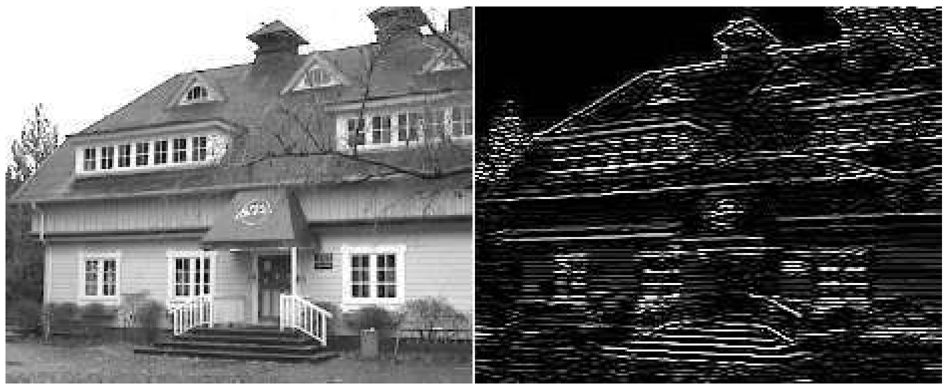
-1	-1	2
-1	2	-1
2	-1	-1

45 degree lines

2	-1	-1
-1	2	-1
-1	-1	2

135 degree lines

When we use the *horizontal edge detector kernel* with the values shown above, we get the following result:



Let's look at this through some numbers. Let us look at how a vertical edge would look like in an image:

0.2 0.2 **0.9** 0.2 0.5

0.1 0.1 **0.9** 0.3 0.2

0.0 0.2 **0.8** 0.1 0.1

0.2 0.3 **0.9** 0.1 0.2

0.1 0.1 **0.9** 0.3 0.2

The values shown in **bold** represent a vertical line in the image.

Let us define are vertical kernel as:

-1 2 -1

-1 2 -1

-1 2 -1

After convolution, the values that we get are:

-2.0 4.3 -2.3

-1.7 4.1 -2.1

-1.7 4.1 -2.1

We can clearly see in this example that the central vertical values in the 3x3 output layer are above, the detection of the vertical line. **Not only we have detected the vertical line, but we are also passing on an image/channel which shows a vertical line**.

Assuming that we are working on an image that has a size of 400x400x3, **how many layers would we need to add to achieve a full receptive field** (or reach 1x1)?

We saw in the last lecture, that we gain a receptive field of 2 every time we add a new layer. Let's refresh

Image Size 7x7 > Each pixel "knows about itself", i.e. RF = 1

Channel Size 5x5 post convolution with 3x3. Each Pixel "knows" about 3x3 pixels, i.e. RF = 3x3

Channel Size 3x3 post convolution with 3x3. Each pixel **"locally knows"** about 3x3 pixels, i.e. Local-RF = 3x3. But indirectly **"globally knows"** about 5x5. i.e. GlobalRF = 5x5

We see a pattern here. Every time we add a layer GRF increases by 2.

Hence, crudely, we need to add 200 layers to reach a GRF of 400!

Now, these are insanely large numbers of layers. We can do much better than this!

GPU Processing

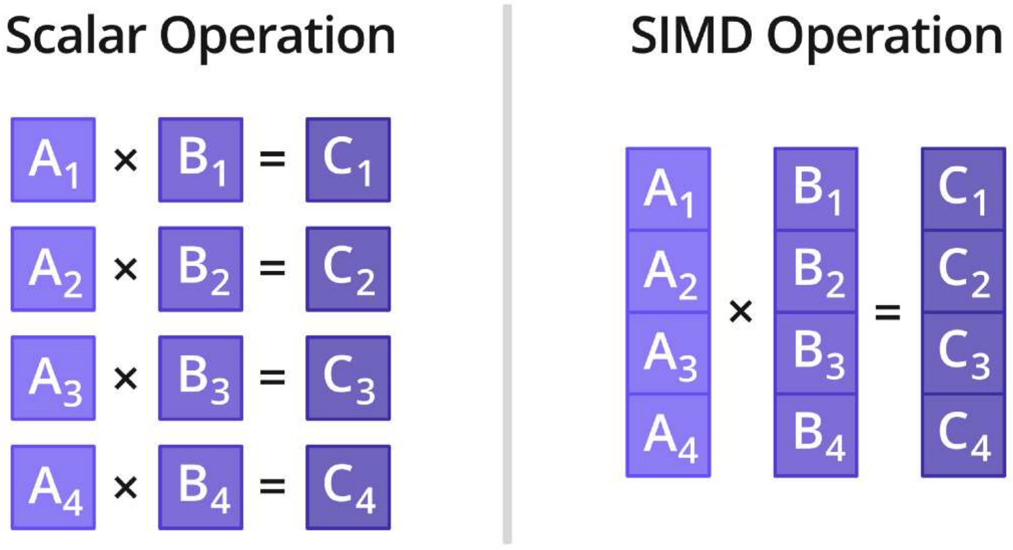
While we are at this matrix below, let's rearrange this a bit (heavily)

0.2	0.2	0.9	0.2	0.5
0.1	0.1	0.9	0.3	0.2
0.0	0.2	0.8	0.1	0.1
0.2	0.3	0.9	0.1	0.2
0.1	0.1	0.9	0.3	0.2

Below is our rearrangement (that happens inside a GPU). Please spend some time thinking about what happened here:

0.2	0.2	0.9	0.1	0.1	0.9	0.0	0.2	0.8
0.2	0.9	0.2	0.1	0.9	0.3	0.2	0.8	0.1
0.9	0.2	0.5	0.9	0.3	0.2	0.8	0.1	0.1
0.1	0.1	0.9	0.0	0.2	0.8	0.2	0.3	0.9
0.1	0.9	0.3	0.2	0.8	0.1	0.3	0.9	0.1
0.9	0.3	0.2	0.8	0.1	0.1	0.9	0.1	0.2
0.0	0.2	0.8	0.2	0.3	0.9	0.1	0.1	0.9
0.2	0.8	0.1	0.3	0.9	0.1	0.1	0.9	0.3
0.8	0.1	0.1	0.9	0.1	0.2	0.9	0.3	0.2

Modern Graphics Processing Units use something called SIMD (Single Instruction Multiple Data). This is different from how CPUs perform processing (Scalar Operations)



But why are we learning about these things?

Well, for a GPU, doing all of those convolutions is a single step! Let's see below:

-1	2	-1	-1	2	-1	-1	2	-1	
0.2	0.2	0.9	0.1	0.1	0.9	0.0	0.2	0.8	-2.0
0.2	0.9	0.2	0.1	0.9	0.3	0.2	0.8	0.1	4.3
0.9	0.2	0.5	0.9	0.3	0.2	0.8	0.1	0.1	-2.3
0.1	0.1	0.9	0.0	0.2	0.8	0.2	0.3	0.9	-1.7
0.1	0.9	0.3	0.2	0.8	0.1	0.3	0.9	0.1	4.1
0.9	0.3	0.2	0.8	0.1	0.1	0.9	0.1	0.2	2.1
0.0	0.2	0.8	0.2	0.3	0.9	0.1	0.1	0.9	-1.7
0.2	0.8	0.1	0.3	0.9	0.1	0.1	0.9	0.3	4.1
0.8	0.1	0.1	0.9	0.1	0.2	0.9	0.3	0.2	-2.1

GPUs spend a little time rearranging the channel in the form above, and perform this multiplication and sum in single steps (that's where this 3x3 kernel size optimization is coming from!).

Receptive Field Calculations

We commented that we need 200 layers to reach our target receptive field, and we can do better! Well, MaxPooling is one operator that helps us in doing that.

Before we work with MaxPooling, let's look at the RF Formula.

Beautiful RF Article on Distill (https://distill.pub/2019/computing-receptive-fields/).

First the formula for calculating the output channel size (n_{out})

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

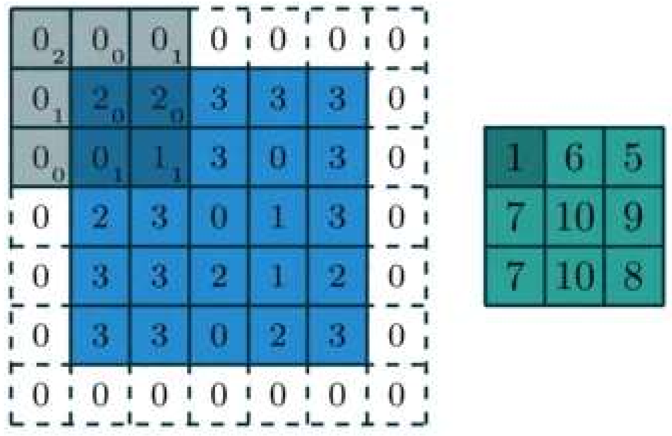
n_{in}:	number of input features
n_{out}:	number of output features
k:	convolution kernel size
p:	convolution padding size
s:	convolution stride size

We know that when we convolve on a 7x7 by 3x3, the output size is 5x5. Let's check it out with this formula

$$n_{out} = \frac{(n_{in}+2p-k)}{s} + 1 = \frac{(7+2\cdot 0-3)}{1} + 1 = 5$$

Let's talk about this s (stride).

Let's observe this image below:



We can see here, that our image size is 7x7 (and not 5x5) because we added a padding of 1.

Our kernel stride here is 2. And the output that we get is 3x3. Let's check out the formula:

$$n_{out} = \frac{(n_{in}+2p-k)}{s} + 1 = \frac{(5+2\cdot 1-3)}{2} + 1 = \frac{4}{2} + 1 = 3$$

Our formula works!

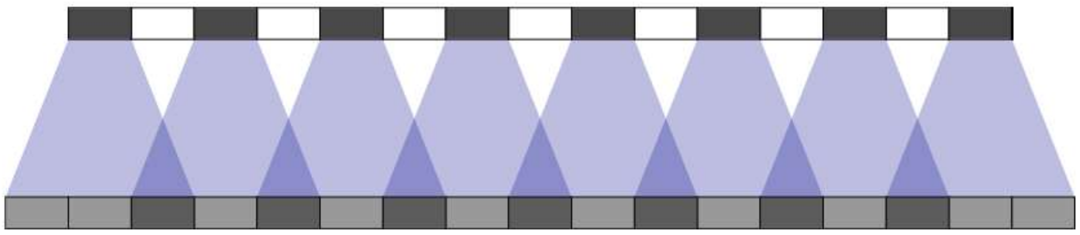
Strides

So, when we add strides, our output channel size reduces drastically. We can make some observations here:

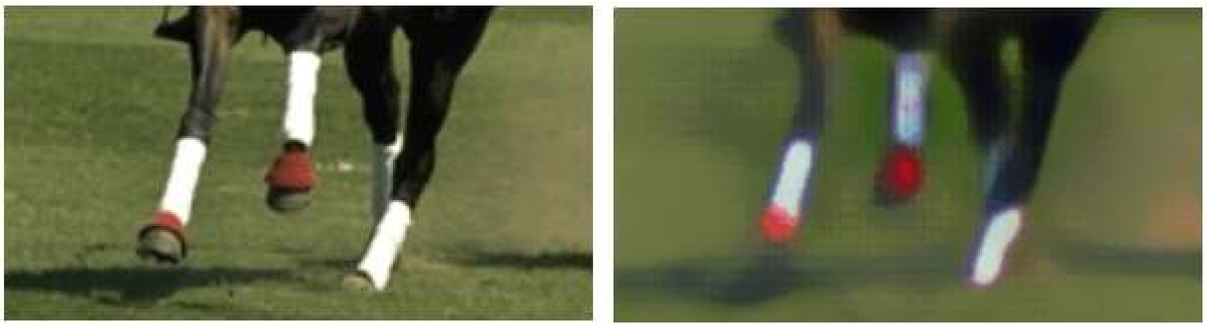
Our output channel has a smaller size

The operation must take less processing time

Compared to the stride of 1, in the stride of 2, we are spending "less time" with the pixels. (Compared to the stride of 1, where we read a pixel 9 times, with a stride of 2, we read a pixel only 4 times max **and read surrounding pixels twice only!**).



So strides larger than 1 are fast, but compromise on the quality of data read. Its output **literally** is a diffused image as shown below:



This is going to cause something called "checkerboard" issues later on.

In Convolution, we saw that we take 9-pixel values and do "something" about it.

Let's think about what all we can do with these 9 values:

Get 9 parameters and then multiply 9-pixel values

Add all of the 9-pixel values and calculate the average

Find the maximum of these 9 values and use it

Find the minimum of these 9 values and use it.

And so on...

The point is that we can decide to do whatever we want to do with these values and if it makes sense, we use this new process.

RF Formula

Finally! The formula for calculating RF

$$n_{out} = \frac{(n_{in} + 2p - k)}{s} + 1$$

Above is the formula for calculating the size of the output channel

The formula for RF is:

$$r_{out} = r_{in} + (k - 1) \cdot j_{in}$$

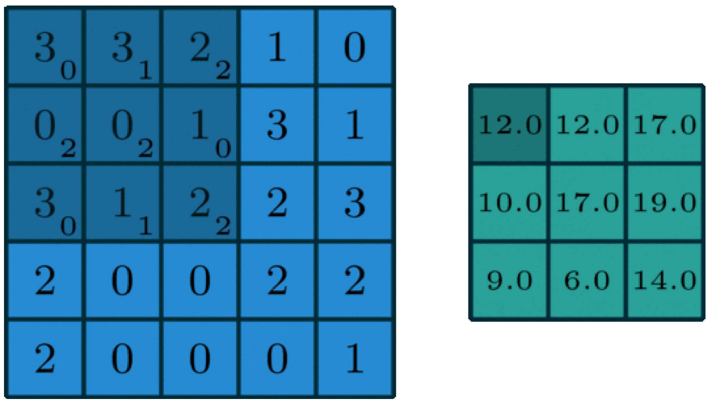
This j_{in} is going to trouble us, so let's delve into it:

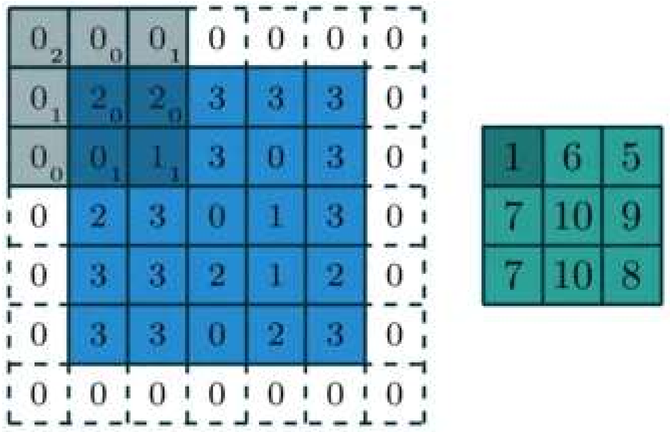
$$j_{out} = j_{in} \cdot s$$

j_{in} *(jump or representation power of the pixels)(also notice it's called j-in and not j-out)* is basically how many strides in total we have taken till now.

In 10 Convolutions, let's say, we have taken a stride of 2, 4 times, then the total stride taken is 8, and our j_{in} is 8.

Let's look at these two images:





In the left image, 3 pixels represent 5 pixels.

In the right image, 3 pixels represent 7 pixels!

So the kernel that convolved here had the same local receptive field of 3x3, but the pixels now represent receptive fields of the case, where they were convolved by 5x5! This is where the jump parameter comes it.

The j_{in} parameter was, say 1, while this convolution happened (ref: right image). Hence the RF based on the formula is:

$$r_{out} = r_{in} + (k-1) \cdot j_{in} = 1 + (3-1)\,.1 = 3$$

Hence, nothing changed "during" this convolution, but:

$$j_{out} = j_{in} \cdot s = 1 \cdot 2 = 2$$

j_{out} has changed, which is j_{in} for the next convolution. So if we had normal convolution in the next step with k = 3, and s = 1, the RF would be:

$$r_{out} = r_{in} + (k-1) \cdot j_{in} = 3 + (3-1) \cdot 2 = 7!$$

Quick RF Calculations

$$n_{out} = \frac{(n_{in} + 2p - k)}{s} + 1$$

$$r_{out} = r_{in} + (k-1) \cdot j_{in}$$

$$j_{out} = j_{in} \cdot s$$

n_{in}	r_{in}	j_{in}	s	p	n_{out}	r_{out}	j_{out}
32	1	1	1	1	32	3	1
32	3	1	**2**	0	15	5	2
15	5	**2**	1	0	13	9	2
13	9	2	**2**	1	7	13	4
7	13	**4**	1	1	7	21	4
7	21	4	**2**	0	3	29	8
3	29	**8**	1	0	1	45	8

Max Pooling

Max Pooling is a simple solution to our problem of reducing the number of layers. And it does it by increasing the Receptive Field as well!

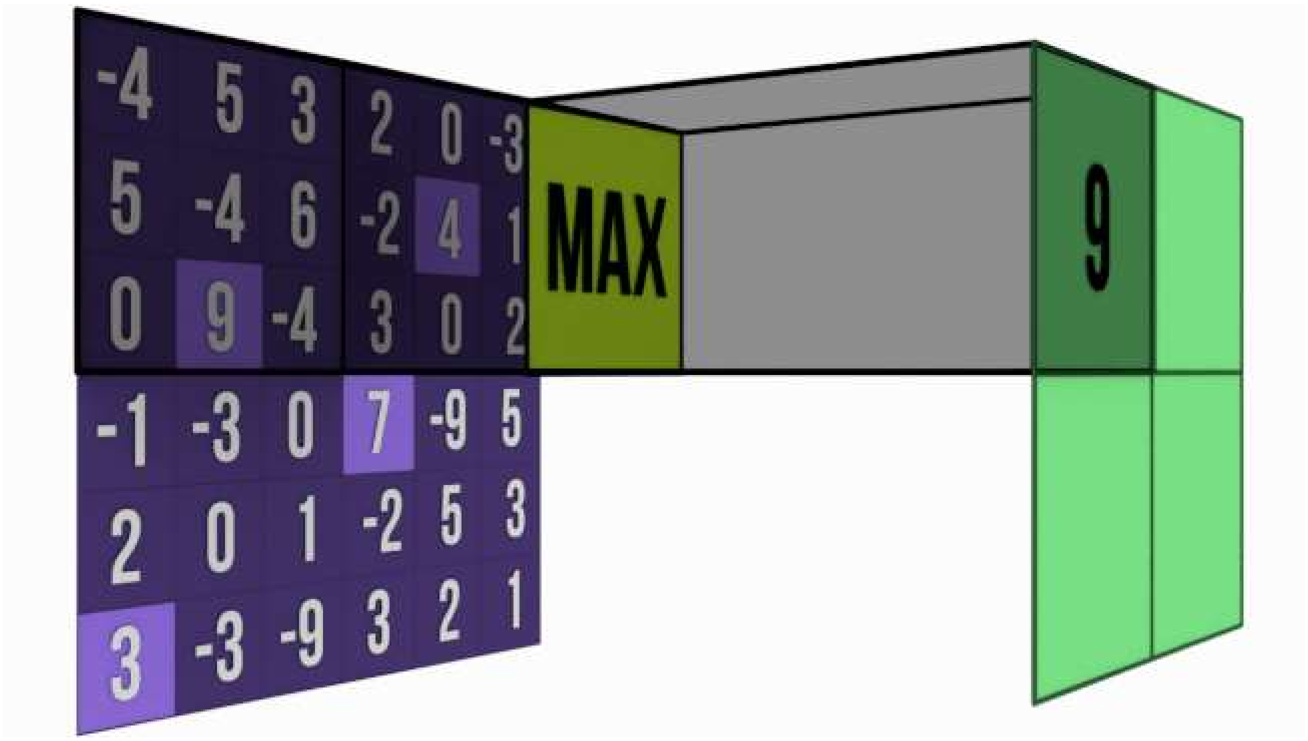
From above we can notice the following benefits of strided-operations:

- Accelerated RF increase
- Channel size reduction
  - reduction of GPU RAM usage
  - reduction of GPU Compute Requirement
  - increase in batch size (total images that we can process now)
  - the capability of running models on edge devices with low RAM and Compute
  - *Shift Invariance*
  - *Rotational Invariance*
  - *Scale Invariance*

The downside of using too much of strided convolution is:

- too much loss of data
- and blurry channel creation.

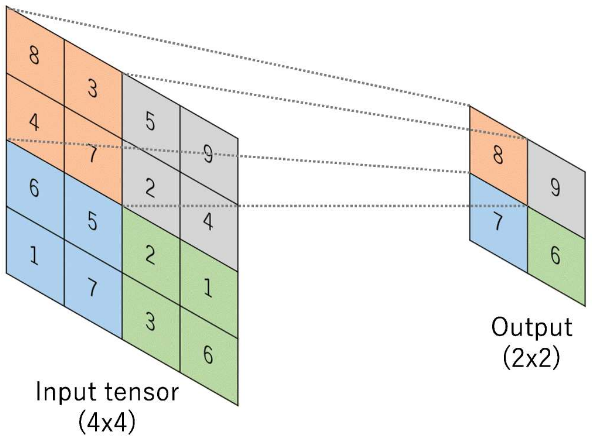
We can further reduce the computation requirement by using **Max Pooling (https://www.newkidscenter.com/images/10415642/children-drawing.jpg)** !



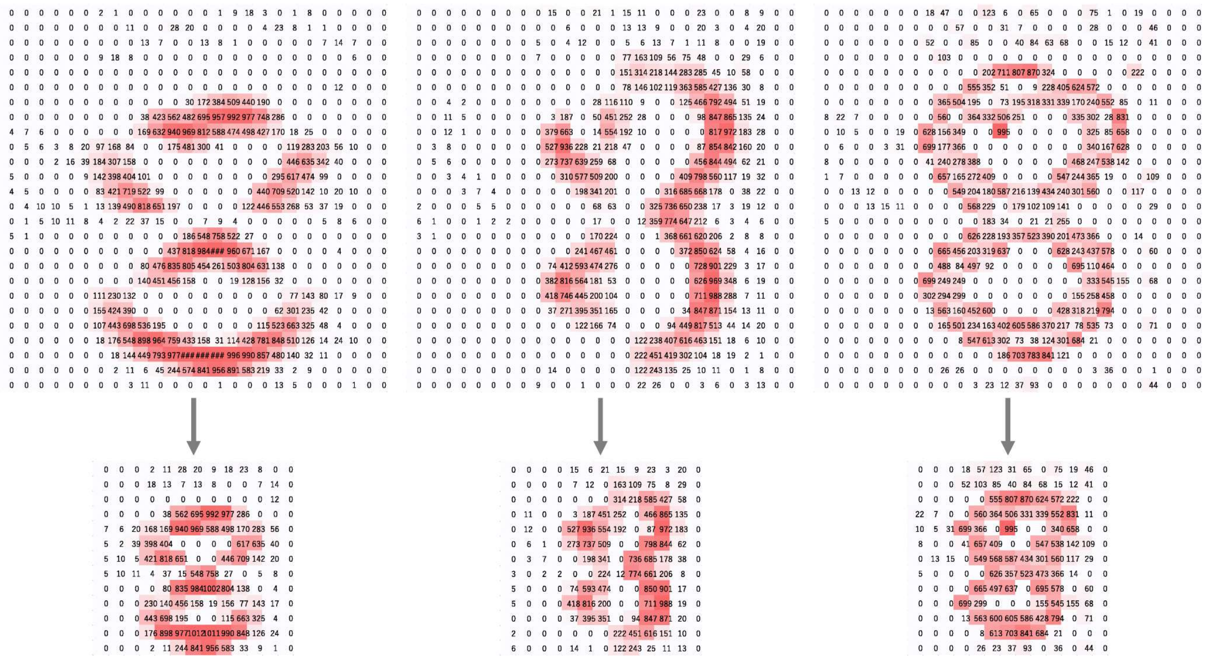
Above we are using a 3x3 MaxPooling. **We nearly always use 2x2 MaxPooling**, and above image is for demonstration.

The effect of MP can be observed below:

a



b

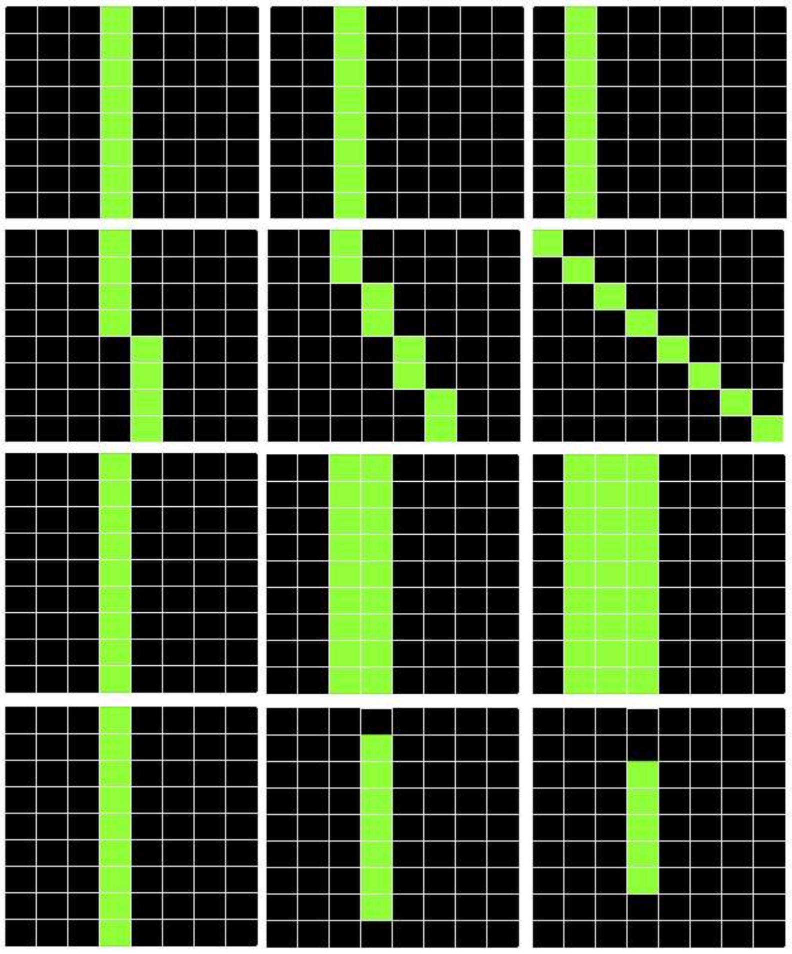


Some rules that we need to follow:

- MaxPooling or Strided Convolution MUST be used in a limited way (you'll see 3 or at max 4 usage per model)
- They must be AS FAR AS AWAY FROM THE PREDICTION LAYER
- Prefer MP if computation is not an issue. If accuracy is paramount, go for Strided Convolution.

Max Pooling & Strided Convolution Invarianes

Apply MaxPooling mentally to each row once, then twice, and convince yourself, that the output is the same!



Layer Counts After Max Pooling

400 | 398 | 396 | 394 | 392 | 390 | MP (2x2)
195 | 193 | 191 | 189 | 187 | 185 | MP (2x2)
92 | 90 | 88 | 86 | 84 | 82 | MP (2x2)
41 | 39 | 37 | 35 | 33 | 31 | MP (2x2)
15 | 13 | 11| 9 | 7 | 5 | 3 | 1

By using MaxPooling we have reduced the layer count from 200 to 27. That's much better.

Assignment

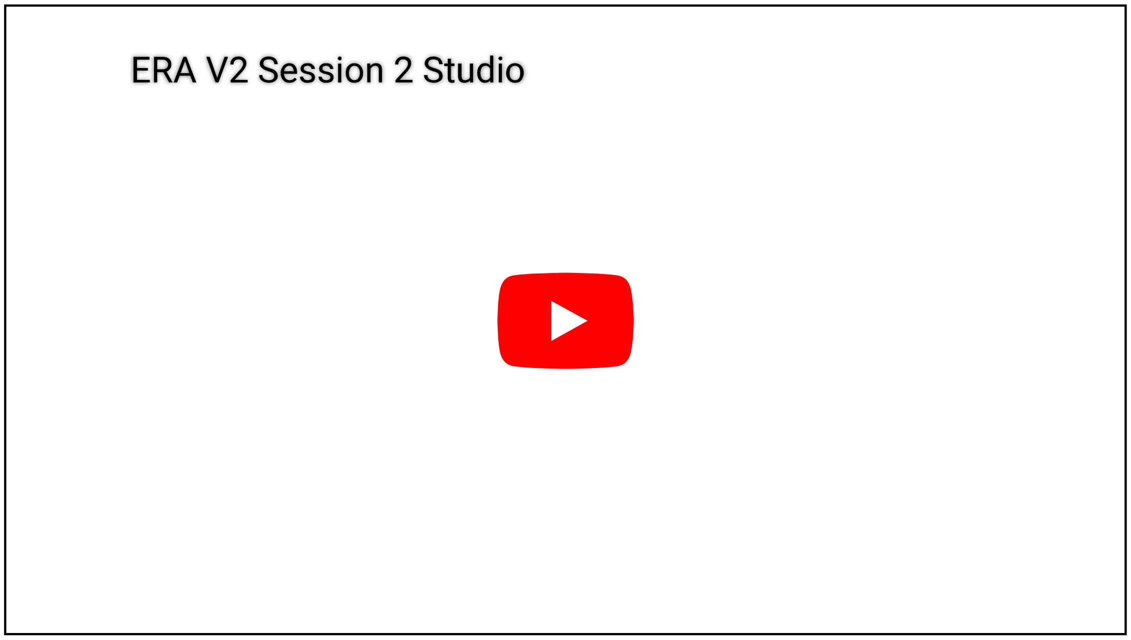
Let's write our first **Code (https://colab.research.google.com/drive/1zH-xrj_h2XIFeuVtz3CE82sYeQeE8fO_?usp=sharing)** .

1. Open the link above
2. Duplicate this file to your Collaboratory
3. Go through the contents of the file carefully:
    1. Annotate every function, class, and line carefully. You must add comments to all the cells to explain exactly what that cell does (the quality of comments will add to the score)
    2. in the cell where the main model is defined:
        1. fill in the blank for the r_in, n_in, etc.
    3. run each cell one by one
    4. experiment

   5. Once you're done with your experiments, then attempt S2 Assignment
4. You'll have 45 minutes to answer the questions asked about the code you just commented on and annotate
5. You'll be running the code once/twice during these 45 minutes (before attempting the Assignment, try and play around in another duplicate file)
6. Read the Assignment Questions carefully before attempting the questions
7. Negative Score for BAD file name

VIDEOS

STUDIO



GOOGLE MEET

ERA V2 Session 2 GM