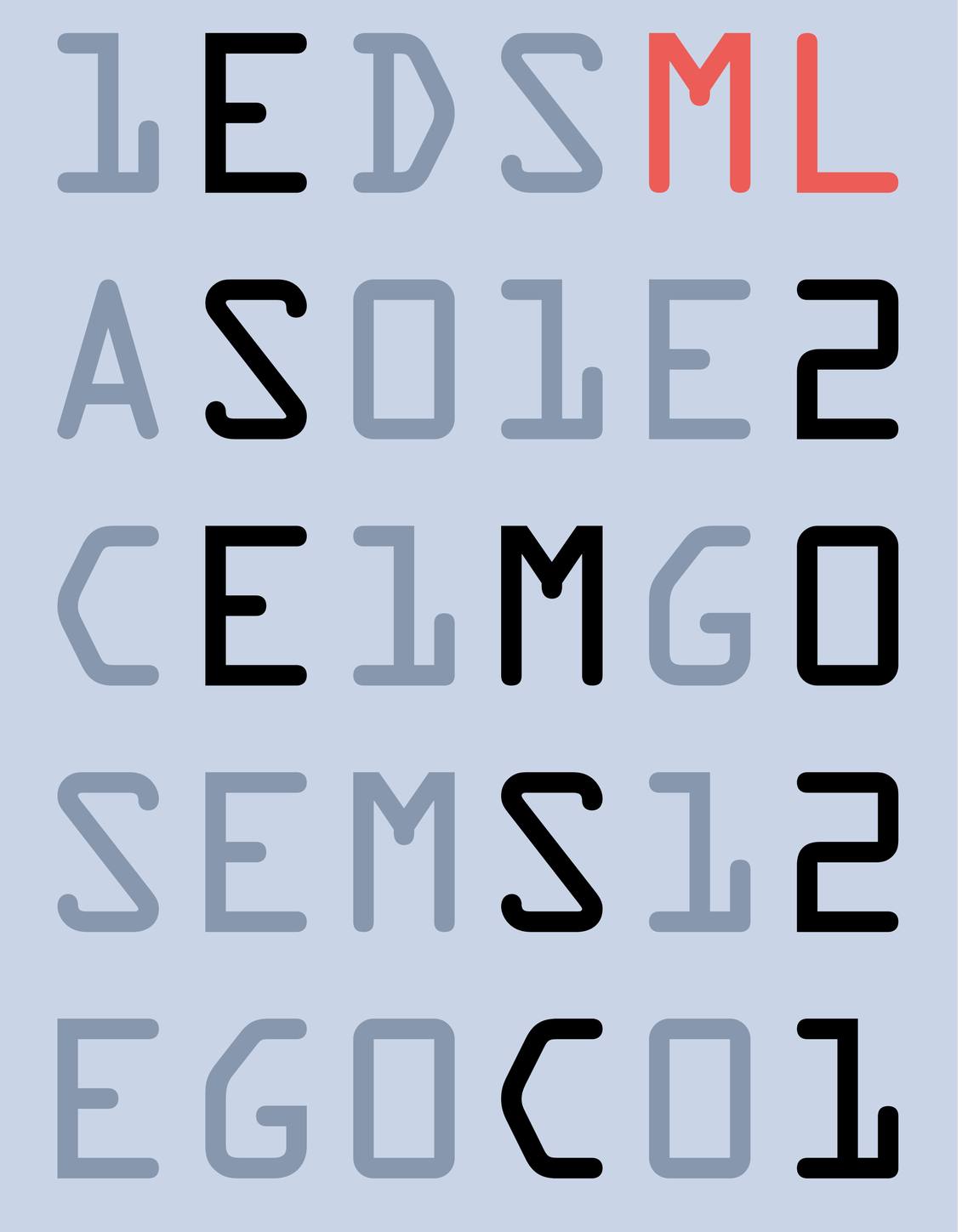


4-CNNs

Lluis Guasch

GTAs online:
Debbie

Objectives of the day:

- Understand what Convolutional Neural Networks (CNNs) are.
- Confident knowledge about calculations associated with the CNN parameters.
- Get familiar with classical CNN structures on well-known examples.

CNNs

1. Convolutional neural networks
2. Pooling and fully-connected layers
3. Examples & transposed convolutions
4. Transfer learning

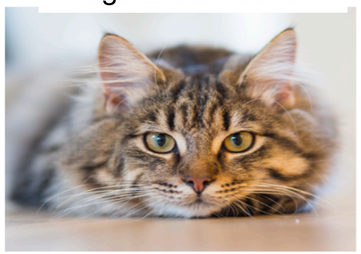
CNNs

1. **Convolutional neural networks**
2. Pooling and fully-connected layers
3. Examples & transposed convolutions
4. Transfer learning

CNNs are commonly (but not only) applied to computer vision problems:

computer vision: data interpretation using computers

Image Classification



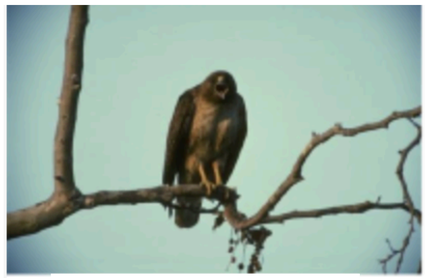
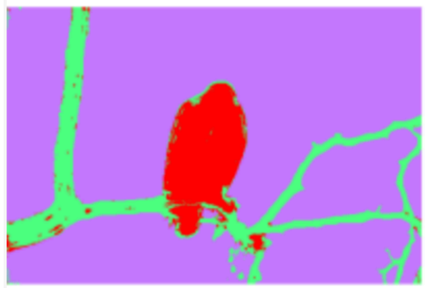
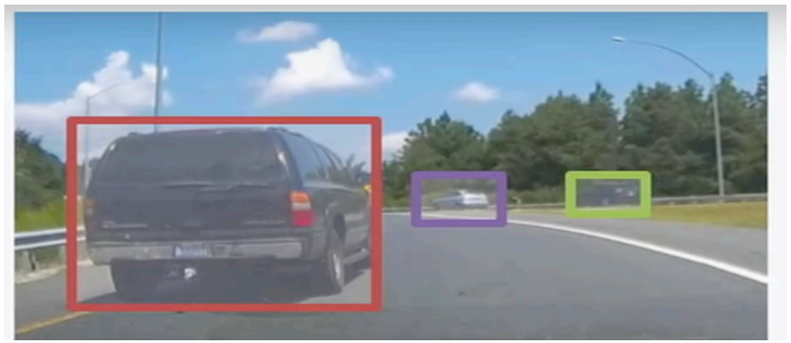


Image Segmentation

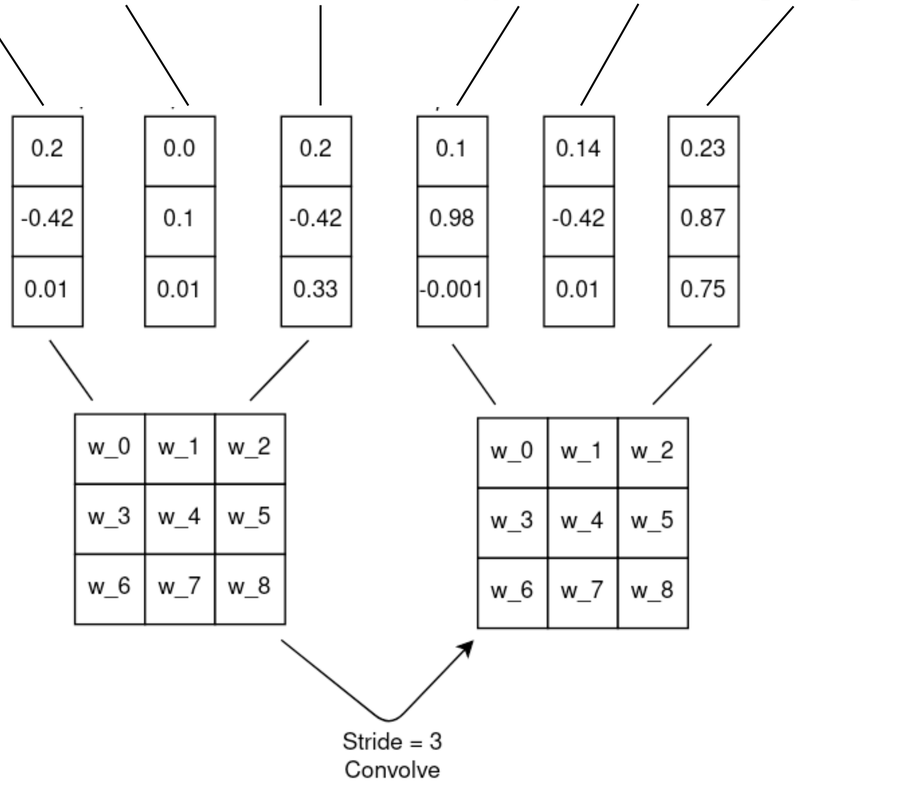




Object Localization (1 object) or Detection (several objects)

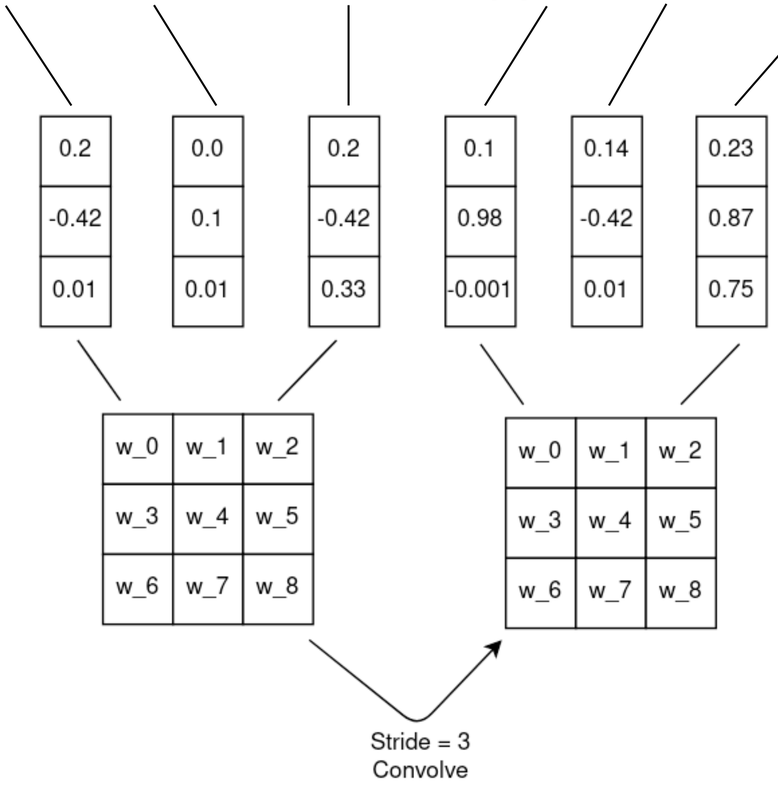
CNNs applied to Natural Language Processing problems:

convolutional neural networks applied to language



CNNs applied to Natural Language Processing problems:

convolutional neural networks applied to language



use tokenization to represent words as vectors (3-dimensional in this case)

MNIST dataset:

Data samples are images, and we want to create algorithms that can interpret these images as 0-9 digits.

But when we use **CNNs**, we do not want to vectorise the images, we want them as **2D images** directly.

MNIST dataset:

Data samples are images, and we want to create algorithms that can interpret these images as 0-9 digits.

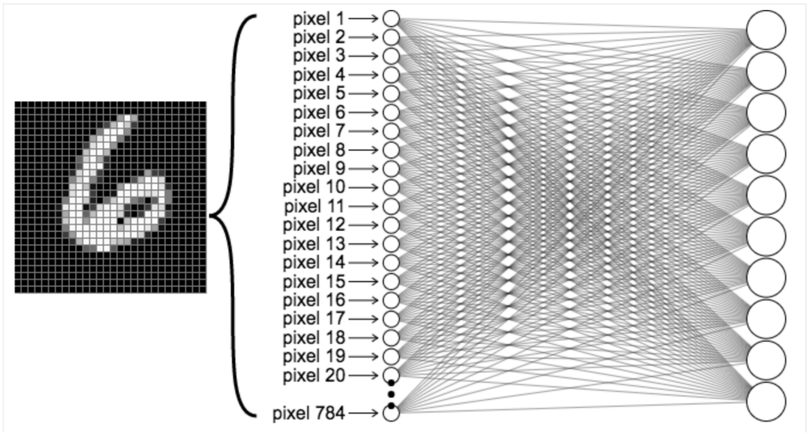
But when we use **CNNs**, we do not want to vectorise the images, we want them as **2D images** directly.

Will it be better to use 2D images as inputs to our networks? Why?

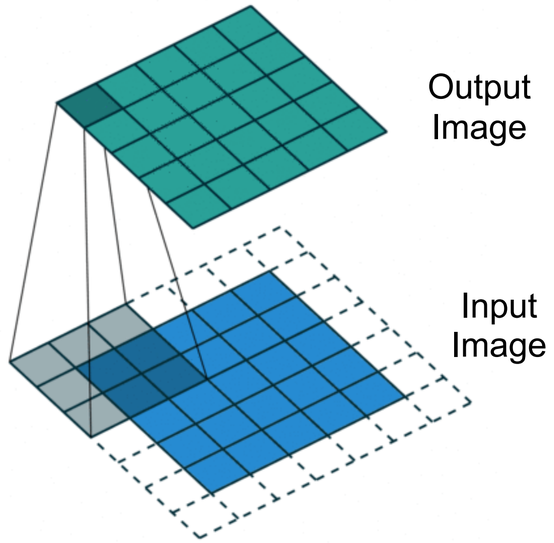
What are convolutional neural networks?

Networks that use **convolutions** to process the data as it ‘travels’ through the network

FeedForward NN Approach on MNIST:
Spatial Organization is Lost

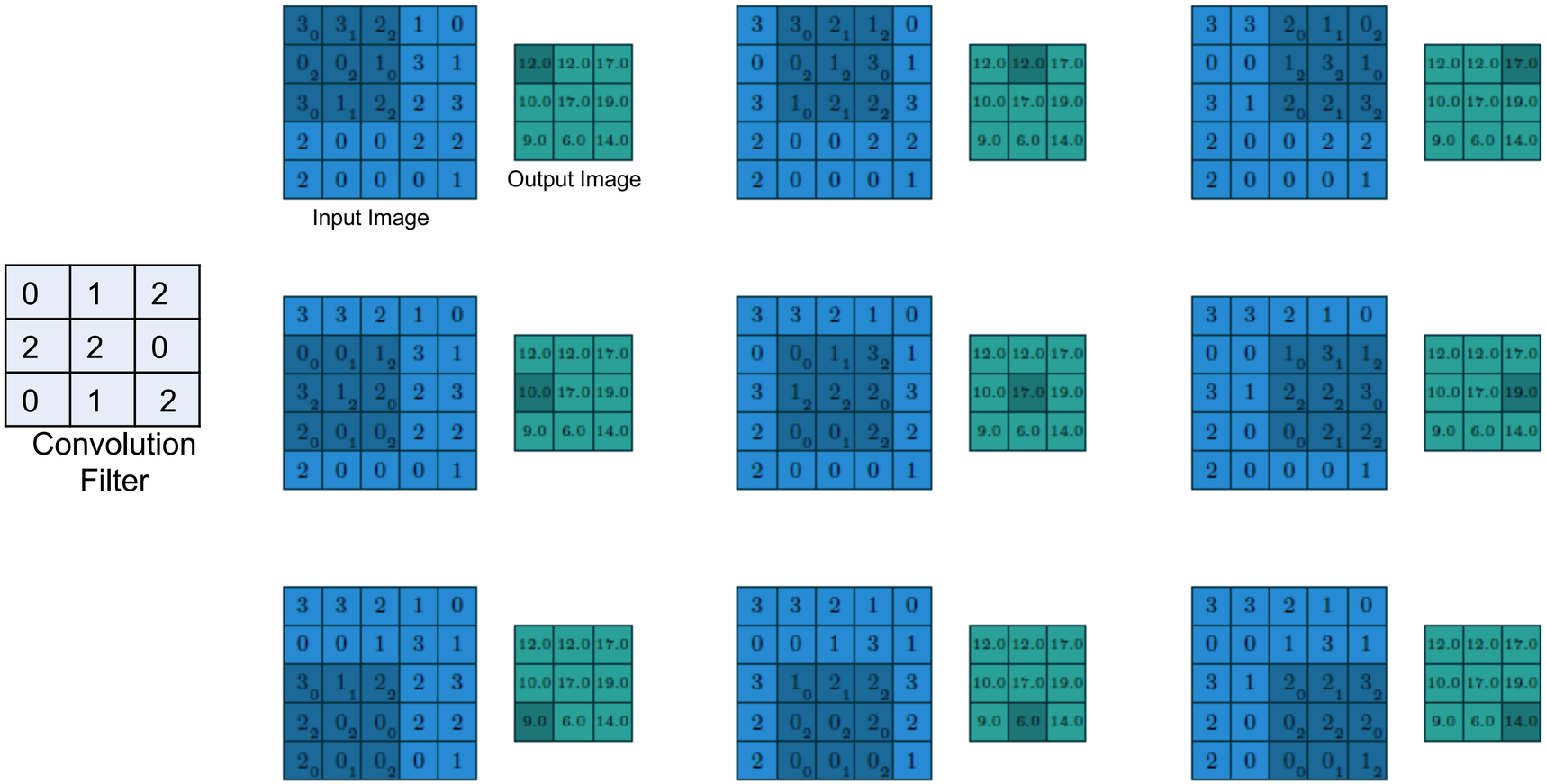


A CNN will take into account the spatial organization of the dataset



The grey 3x3 moving window is the convolutional filter

What is a convolution operation?



from Dumoulin & Visin, 2018

Exercise: calculate the output image (*tedious, I know*)

Input Image

4	2	3	3
2	1	2	5
3	2	0	3
1	2	3	1

*

Convolution Filter
(or Kernel)

2	-3	2
1	0	1
2	-3	3

=

Exercise: calculate the output image (*tedious, I know*)

Input Image

4	2	3	3
2	1	2	5
3	2	0	3
1	2	3	1

*

Convolution Filter (or Kernel)

2	-3	2
1	0	1
2	-3	3

=

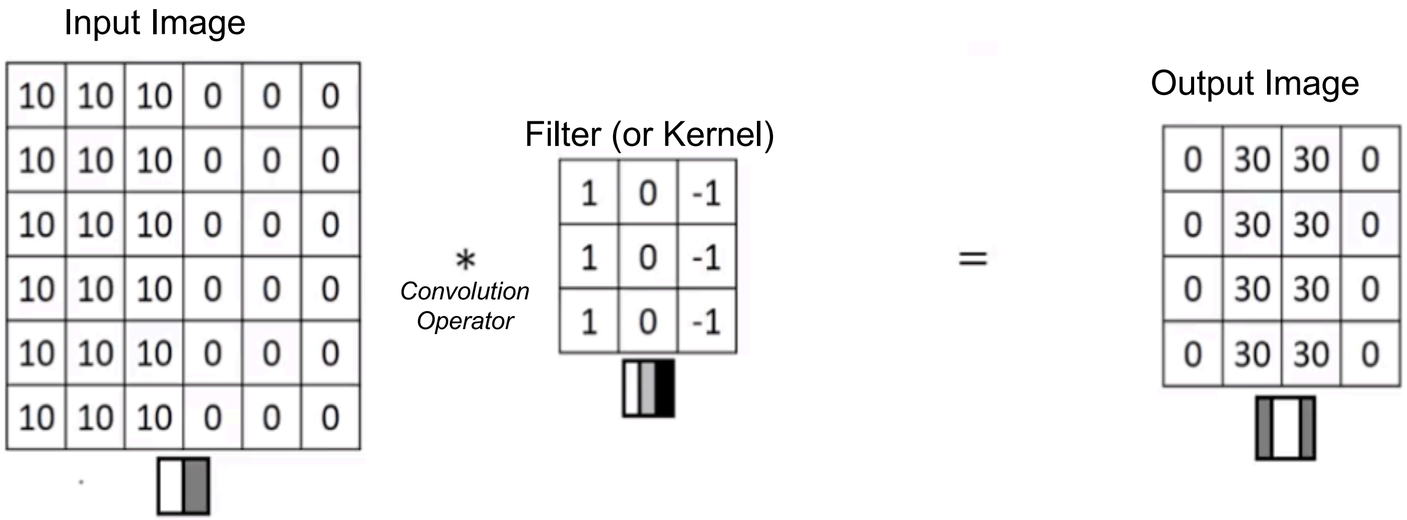
Output Image

12	20
13	9

Convolutional Filters

Filters (or **kernels**) are used by themselves on a variety of tasks.

For example for edge detection, we could design a filter like:



from Andrew Ng

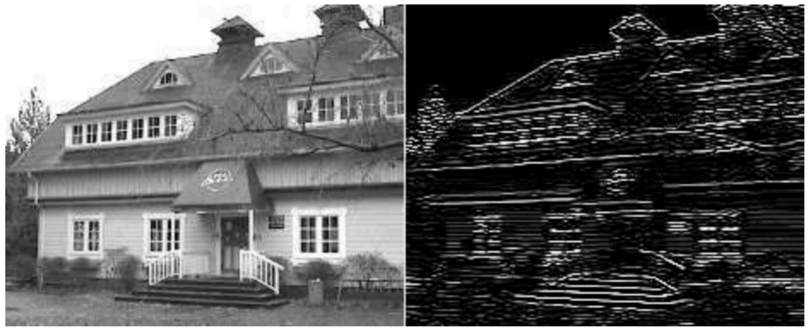
Edge detection and blurring:

Blur

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Horizontal Edges

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



https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

Convolutional Filters

Generic edges and Sobel filters:

Edges

-1	-1	-1
-1	8	-1
-1	-1	-1

Sobel Filters

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

Horizontal

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

Vertical

https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

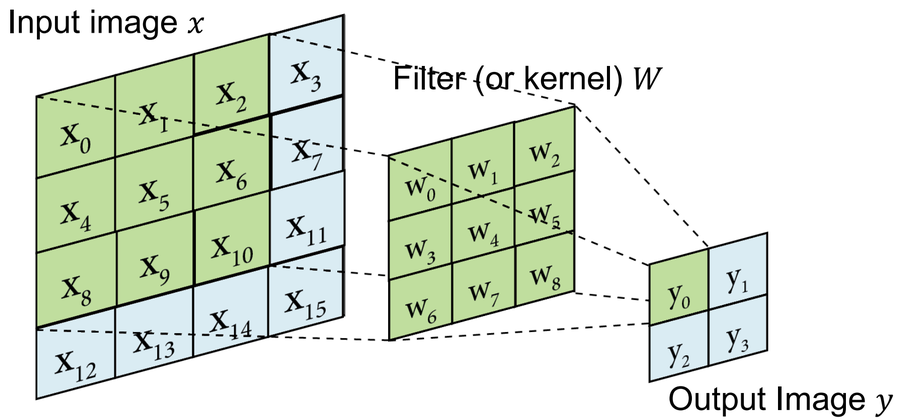
Convolutional Filters

Interactive convolutions:

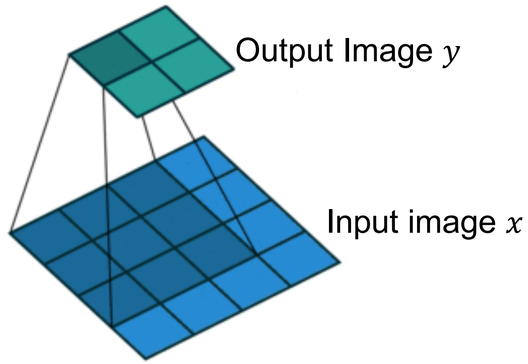
https://setosa.io/ev/image-kernels/

Convolutional Filters in CNNs

CNNs parametrise the filters to **optimise their coefficients**, which become the network trainable parameters



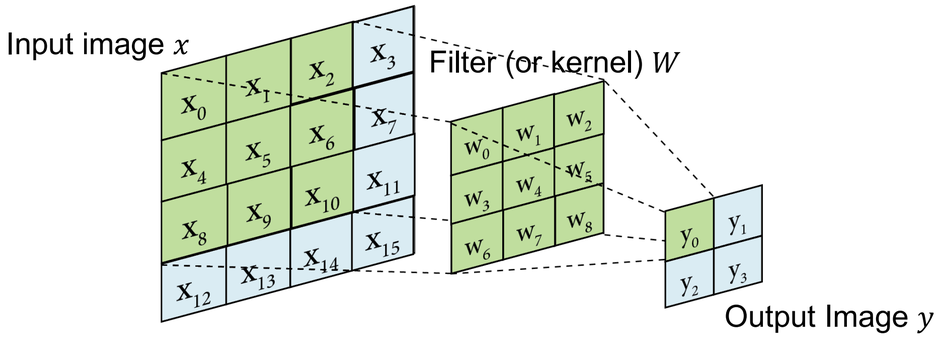
$$W * x = y$$



$$y_0 = w_0x_0 + w_1x_1 + w_2x_2 + w_3x_4 + w_4x_5 + w_5x_6 + w_6x_8 + w_7x_9 + w_8x_{10}$$

Convolutional Filters in CNNs

But convolutions are linear operations. As in the FNN case, we add **activations**:



$$y_0 = w_0x_0+w_1x_1+w_2x_2+w_3x_4+w_4x_5+w_5x_6+w_6x_8+w_7x_9+w_8x_{10}$$

A bias term can be added

$$y_0 = w_0x_0+w_1x_1+w_2x_2+w_3x_4+w_4x_5+w_5x_6+w_6x_8+w_7x_9+w_8x_{10} + \boldsymbol{b}$$

And a non-linear transformation is applied by an activation function g

$$y_0 = \boldsymbol{g}(w_0x_0+w_1x_1+w_2x_2+w_3x_4+w_4x_5+w_5x_6+w_6x_8+w_7x_9+w_8x_{10} + \boldsymbol{b})$$

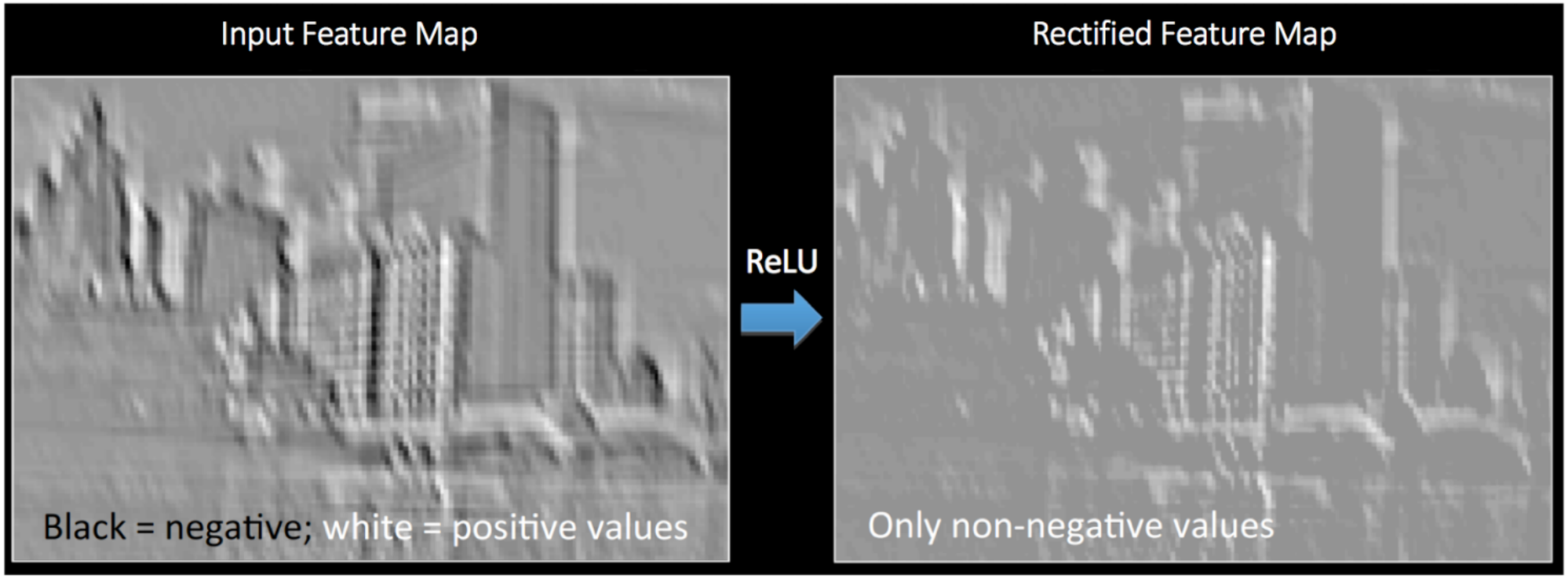
Convolutional Filters in CNNs

So, at each convolutional layer we perform a **convolution**:

https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

Convolutional Filters in CNNs

Followed by an **activation** (ReLU in this case):



https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

Filter Parameters

There are a few parameters that define how convolutions are computed:

Parameters

Input Image Size: $n \times n$

Filter (or Kernel) Size: f $\times f$

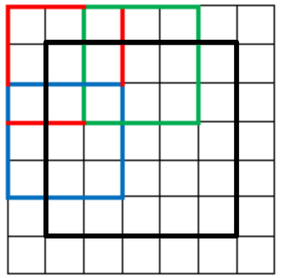
Padding: p

Stride : s

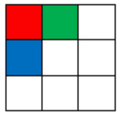
Output Image Size

$$\left(\frac{n+2p-f}{s}+1\right) \times \left(\frac{n+2p-f}{s}+1\right)$$

Input Image



Output Image

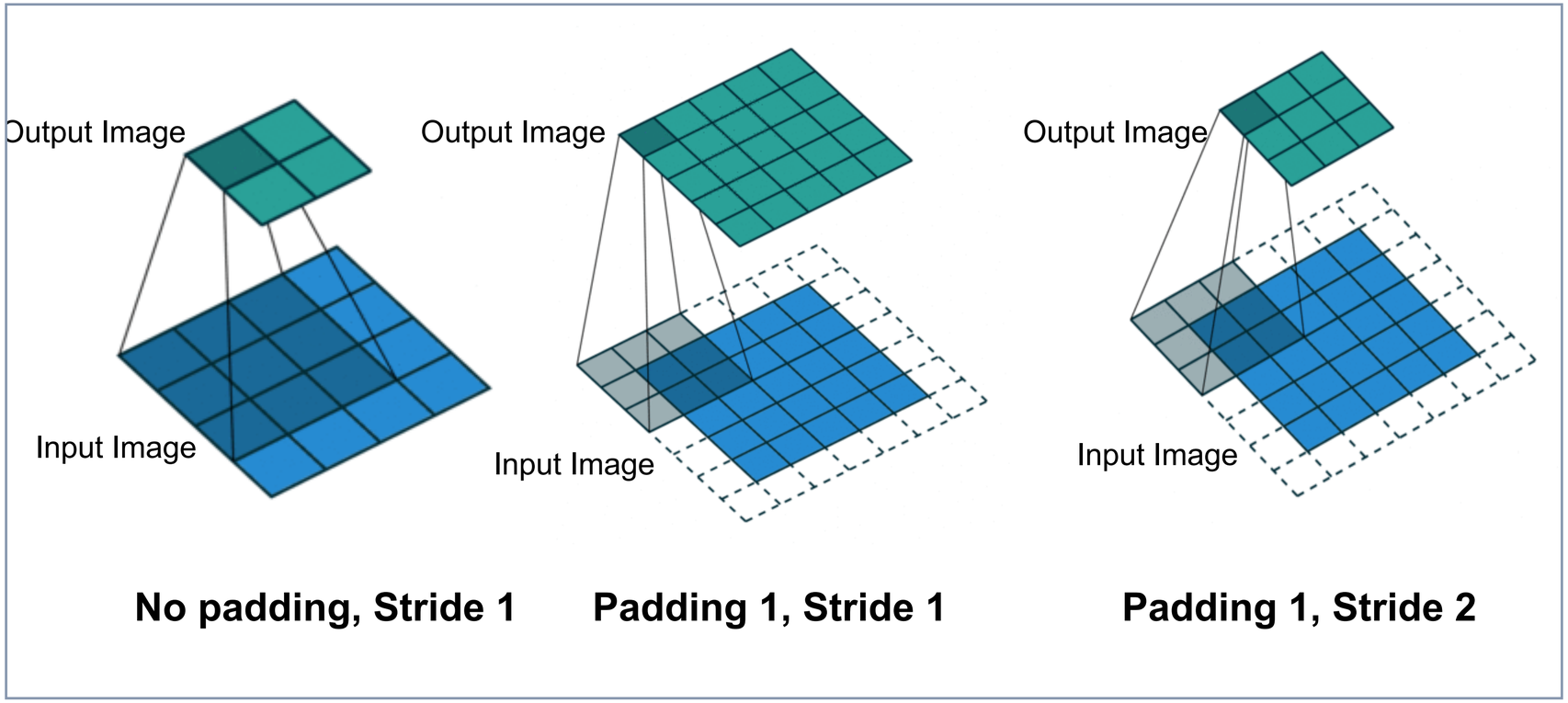


3x3!

$n = 5\ p = 1\ f = 3\ s = 2$

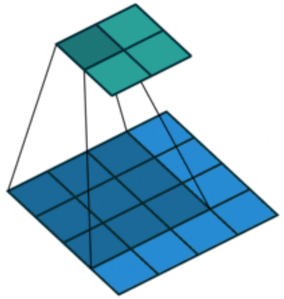
Filter Parameters

Padding and stride:

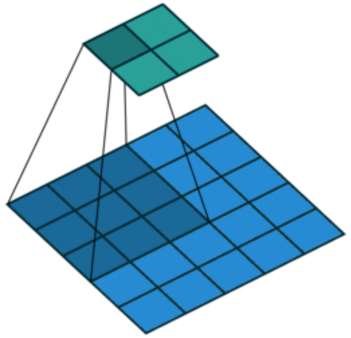


Filter Parameters

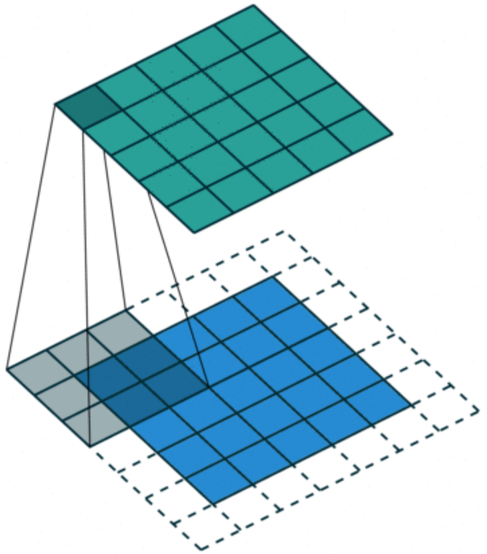
Padding and stride:



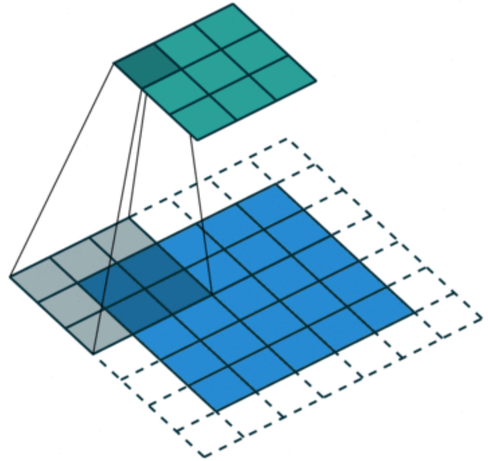
no padding
no strides



no padding
stride = 2



padding = 1
no strides

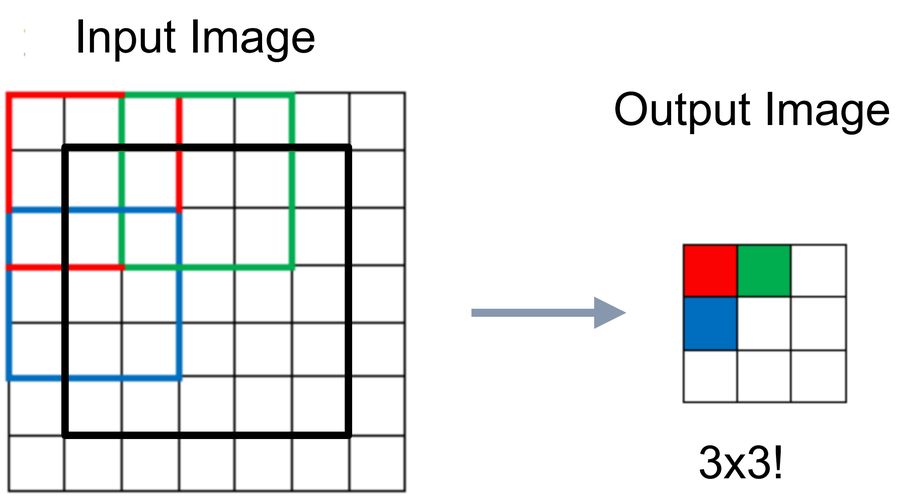


padding = 1
stride = 2

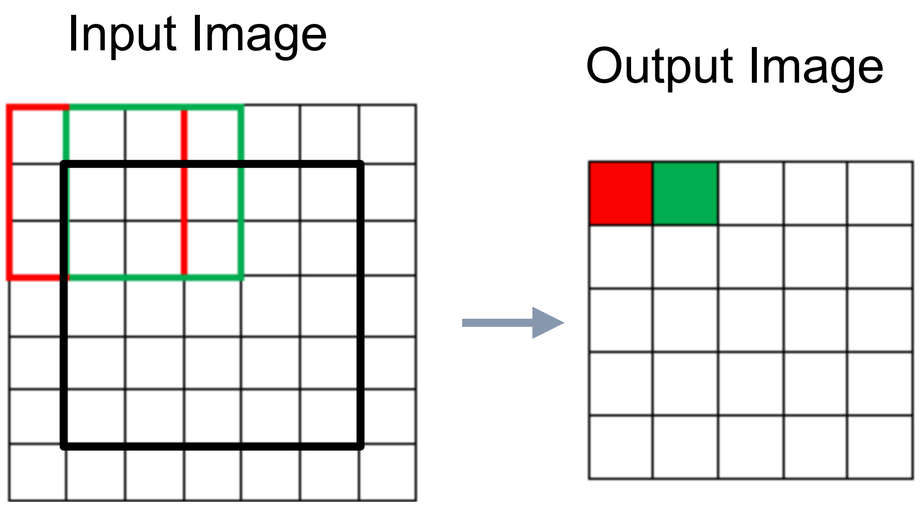
https://github.com/vdumoulin/conv_arithmetic

Filter Parameters

Stride and padding control the size of the output image:



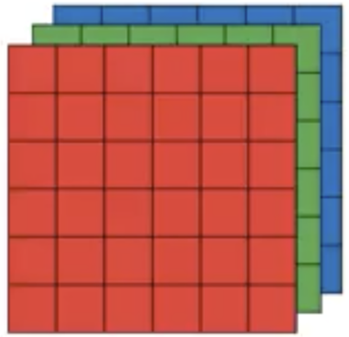
$n = 5\ p = 1\ f = 3\ s = 2$



$n = 5\ p = 1\ f = 3\ s = 1$

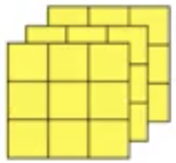
Multiple Input Channels

Often, images have **3 input channels** (**RGB**) or more. How do we convolve such 3-channel data?



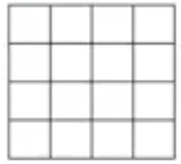
$6 \times 6 \times 3$

$*$



$3 \times 3 \times 3$

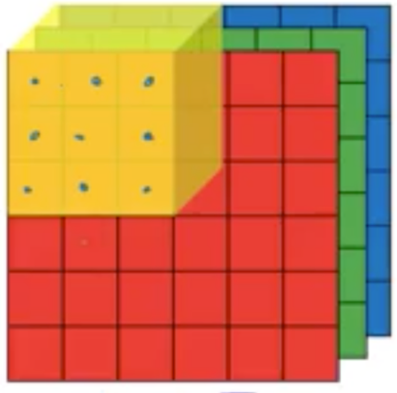
$=$



$4 \times 4 \times 1$

1 output channel

27 operations each time (28 if bias term)

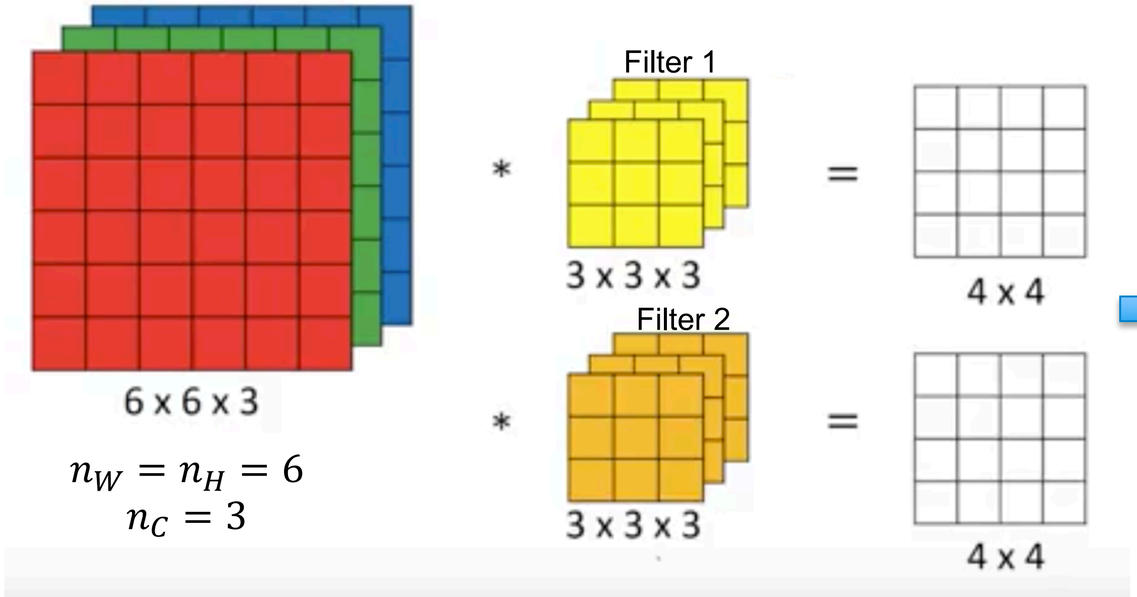


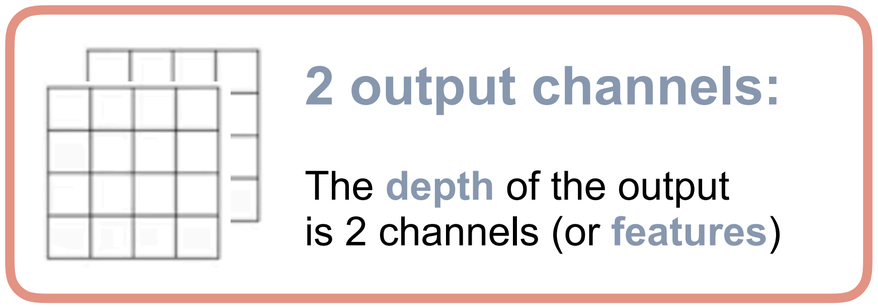
Each slice of the filter can detect different features in each of the 3 colour channels, for instance vertical edges in Red, horizontal edges in Green and vertical edges in Blue

https://www.youtube.com/watch?v=KTB_OFoAQcc&list=PLkDaE6sCZn6Gl29AoE31iwdVwSG-KnDzF&index=8

Multiple Output Channels

We can also decide that we want **multiple output channels**:



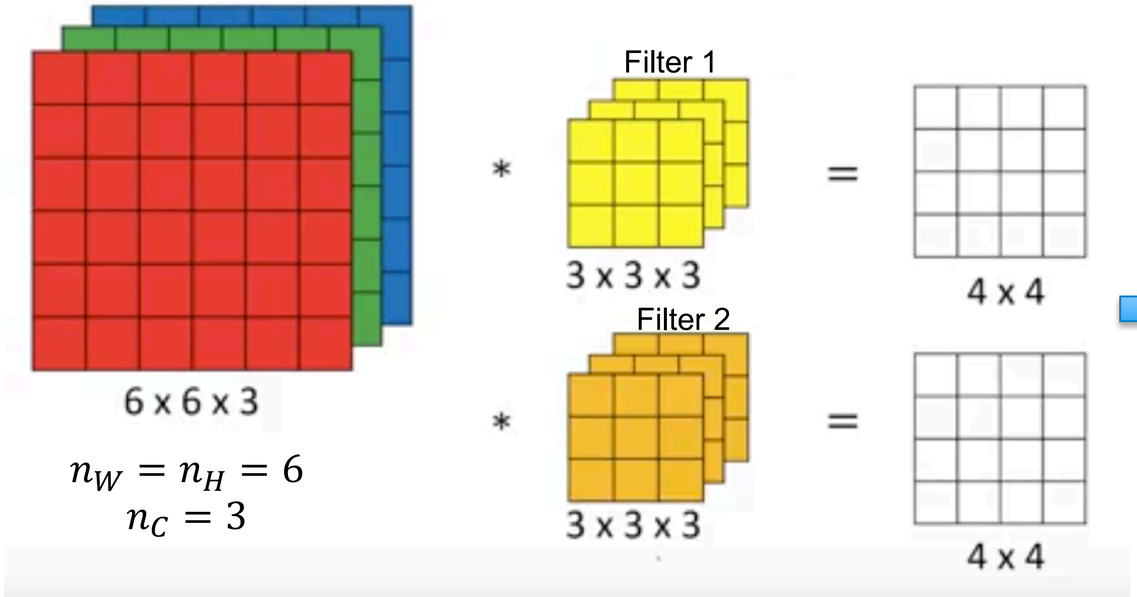


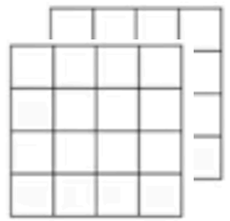
Exercise: *if my input Image is 50x50 with 10 channels, and if I apply 25 filters each of size 3x3x10 with a stride of 1, no padding, what is the size and number of channels of the output image?*

https://www.youtube.com/watch?v=KTB_OFoAQcc&list=PLkDaE6sCZn6Gl29AoE31iwdVwSG-KnDzF&index=8

Multiple Output Channels

We can also decide that we want **multiple output channels**:





2 output channels:

The **depth** of the output is 2 channels (or **features**)

Exercise: *if my input Image is 50x50 with 10 channels, and if I apply 25 filters each of size 3x3x10 with a stride of 1, no padding, what is the size and number of channels of the output image?*

Answer: *size is 48x48. with 25 channels*

https://www.youtube.com/watch?v=KTB_OFoAQcc&list=PLkDaE6sCZn6Gl29AoE31iwdVwSG-KnDzF&index=8

Training Weights

How many **weights** (filter coefficients + bias terms) need to be **trained**?

If I have 100 filters of size 4 x 4 x 3 (each with bias term) in layer l, what is the total number of weights (or degrees of freedom) to train for this layer?

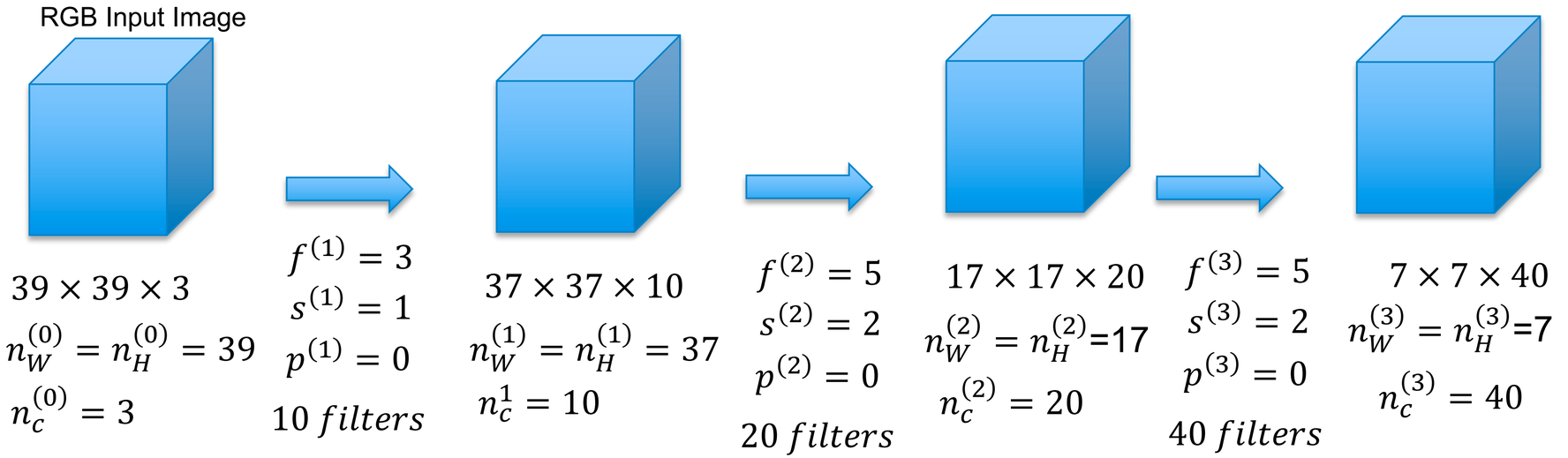
$$\boldsymbol{Number\ of\ filters} \times (\boldsymbol{size\ of\ filter} + \mathbf{1}\ \boldsymbol{bias\ term}) =$$

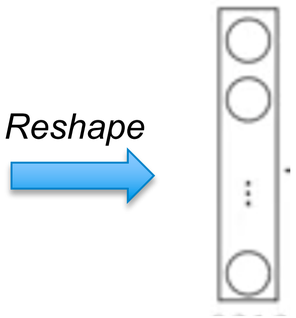
$$\mathbf{100} \times (\mathbf{4} \times \mathbf{4} \times \mathbf{3} + \mathbf{1}) = \ \mathbf{100} \times \mathbf{49} = \mathbf{4900}\ \boldsymbol{weights}$$

Note that the number of parameters is independent of the size of the input image, which was not the case for feed-forward neural networks!

Training Weights

Our first CNN example:





Full Convolution
(ie single neuron)

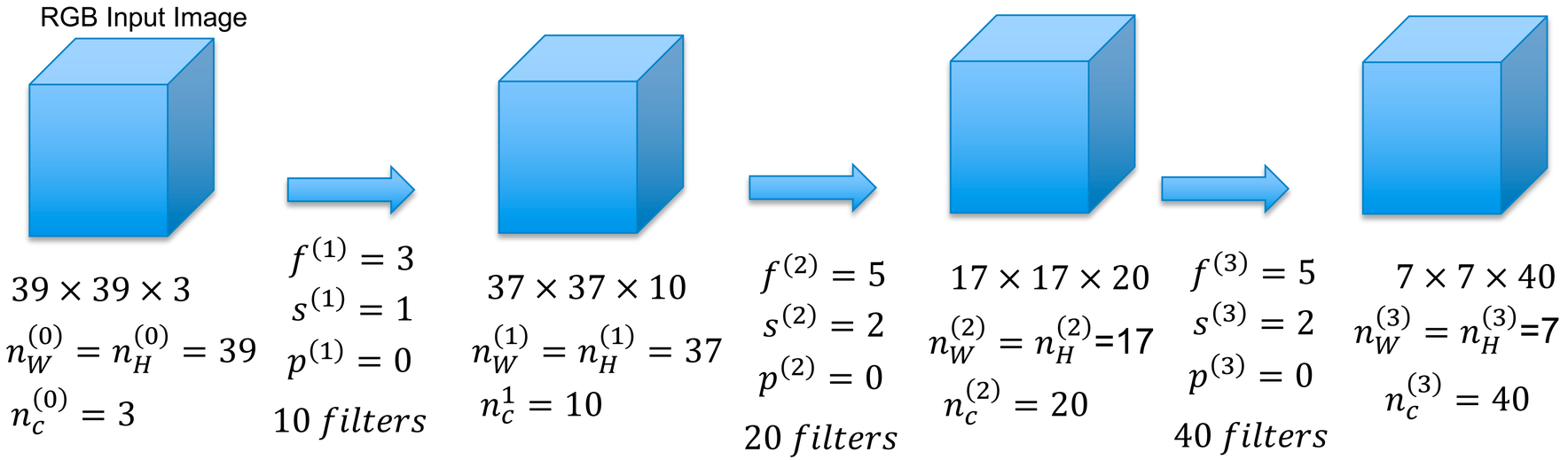
Probability of being a certain class

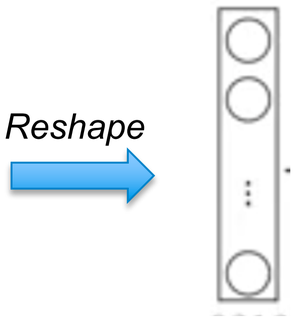
Exercise: how many weights does this network have (including bias terms)?

from Andrew Ng

Training Weights

Our first CNN example:





Full Convolution
(ie single neuron)

Probability of being a certain class

Answer:
=10x28+20x251+40x501+1961=
280+5020+20040+1961=**27301**

from Andrew Ng

PARAMETER COUNT

Degrees of freedom (or parameters) in a layer l:

Assume layer l has $n_c^{(l)}$ filters of size $f^{(l)}$, a padding $p^{(l)}$ and a stride $s^{(l)}$

Assume the input image has size $n_H^{(l-1)} \times n_W^{(l-1)} \times n_c^{(l-1)}$

and the output image has size $n_H^{(l)} \times n_W^{(l)} \times n_c^{(l)}$

The size of each filter is $f^{(l)} \times f^{(l)} \times n_c^{(l-1)}$ and the depth or number of output features is the number of filters $n_c^{(l)}$

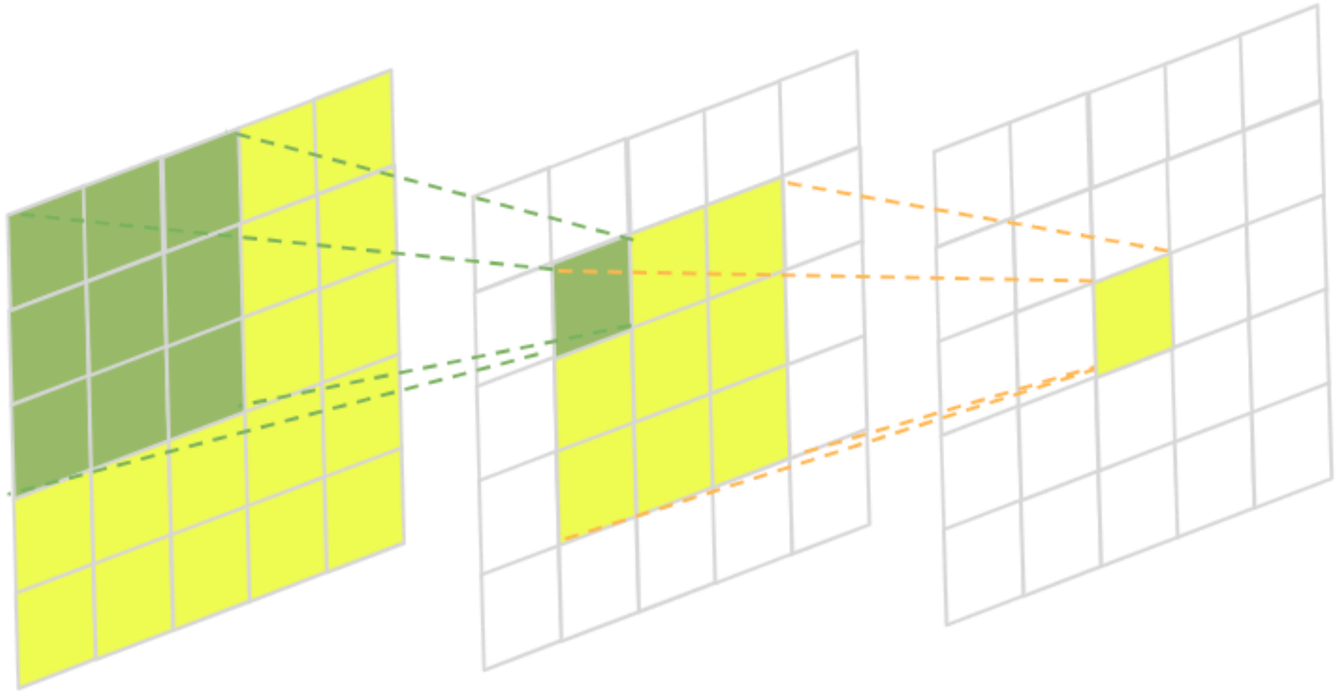
And we can generalize the formula $n_H^{(l)} = \frac{n_H^{(l-1)} + 2p^{(l)} - f^{(l)}}{s^{(l)}} + 1$ and $n_W^{(l)} = \frac{n_W^{(l-1)} + 2p^{(l)} - f^{(l)}}{s^{(l)}} + 1$

And the total number of degrees of freedom (or parameters) of layer l is:

$$\boldsymbol{Number\ of\ filters} \times (\boldsymbol{size\ of\ filter} + \mathbf{1}\ \boldsymbol{bias\ term}) = \boldsymbol{n_c^{(l)}} \times \left(\boldsymbol{f^{(l)}} \times \boldsymbol{f^{(l)}} \times \boldsymbol{n_c^{(l-1)}} + \mathbf{1}\right)$$

Receptive Field

The **receptive field** is defined as the region of the input that affects the output, and **it can be defined between adjacent or non-adjacent CNN layers**.

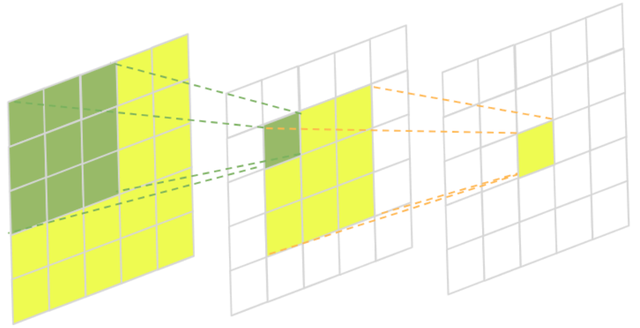


- The **green pixel** in the second layer has a receptive field shown as a **green region** in the first layer.
- The **yellow pixel** in the third layer has a receptive field shown as **yellow regions** in the second and first layers

https://www.baeldung.com/cs/cnn-receptive-field-size

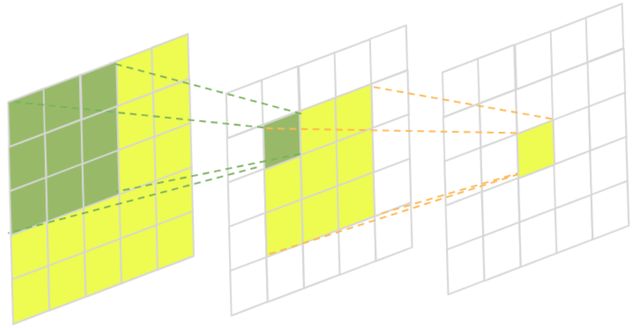
Receptive Field

Why are we interested in the receptive field in our network?



Receptive Field

Why are we interested in the receptive field in our network?



Because the receptive field will determine what are the hyperparameters I need to ensure full receptive field on my inputs:

- filter **size** & **stride**
- **number** of convolutional **layers**

1. Convolutional neural networks
2. **Pooling and fully-connected layers**
3. Examples & transposed convolutions
4. Transfer learning

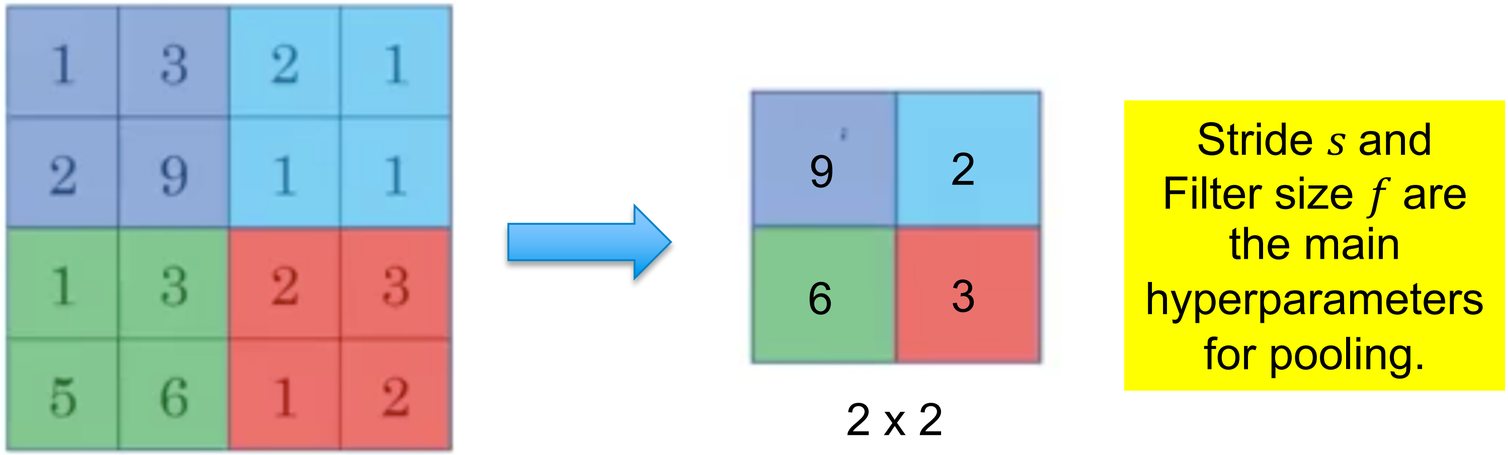
CNN Layers

Three main types of layers in CNNs:

- **Convolutional** layers (CONV)
- **Pooling** layers (POOL)
- **Fully-connected** layers (FC)

Pooling layers **down-sample** by collecting feature information in patches

Example: **max pooling**



4 x 4

Here $s = 2$ and $f = 2$ Usually $f = s$

Pooling makes the input representations (feature dimension) smaller and more manageable for the next layer, Max Pooling is used because it may be interesting to keep the high values for the activation of the next layer as they may characterize some important features. Pooling reduces the number of parameters and computations in the network, therefore controlling overfitting.

We have already seen fully-connected layers in FFNs:

Transform for example 32x32x3 image into 10x1 vector.

1. Reshape image into 1D vector x of dimension 32x32x3= 3072
2. Apply full convolution through matrix W

x is 3072x1

Wx

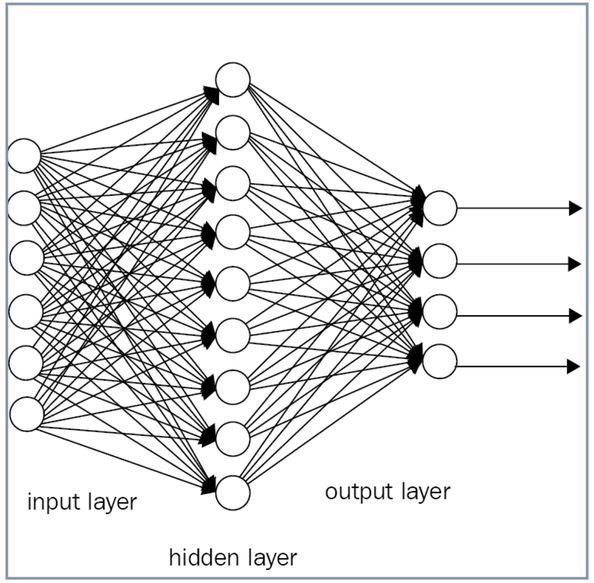
W (or θ) is 10x3072
if no bias term

g non-linear activation function

$g(Wx)$

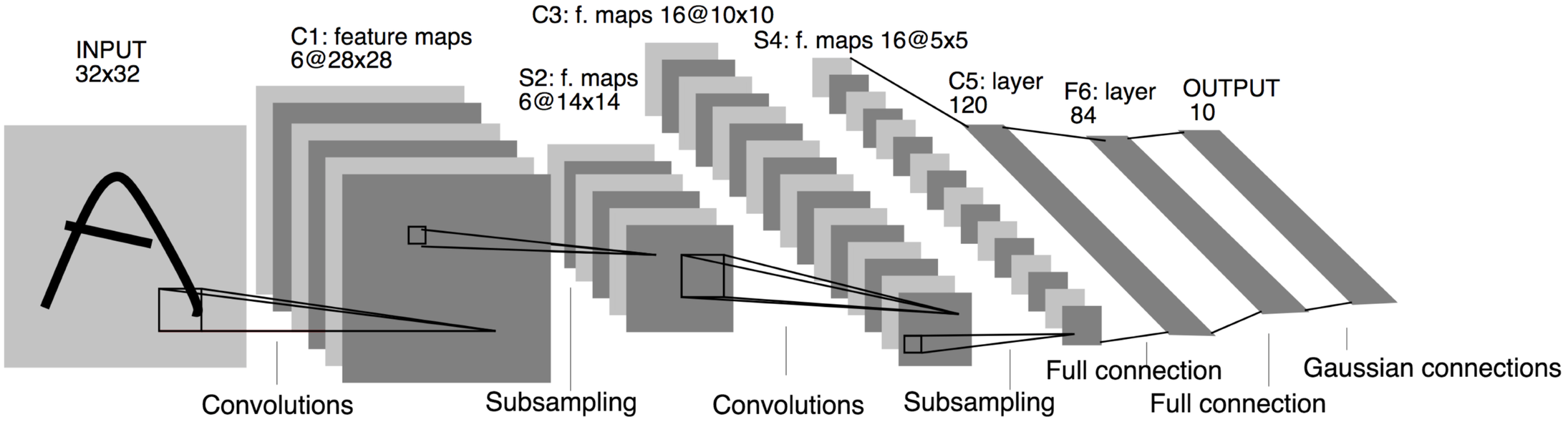
Wx is 10x1

FC layers

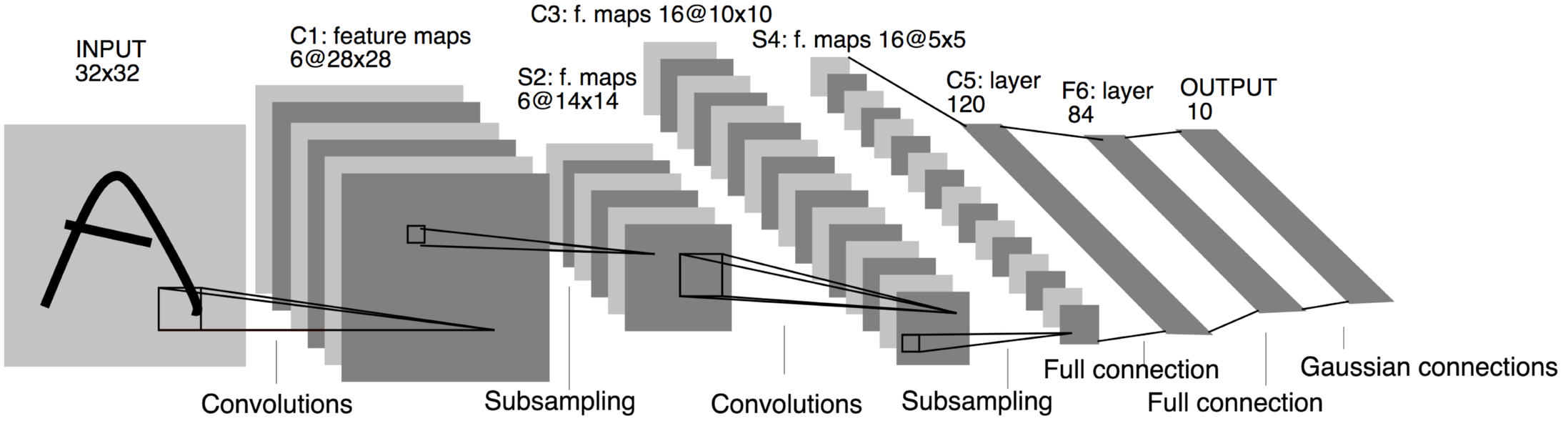


Fully connected layers often have many weights as they connect every neuron in input layer to every neuron in output layer.

LeNet-5 architecture from Yann LeCunn applied to MNIST:

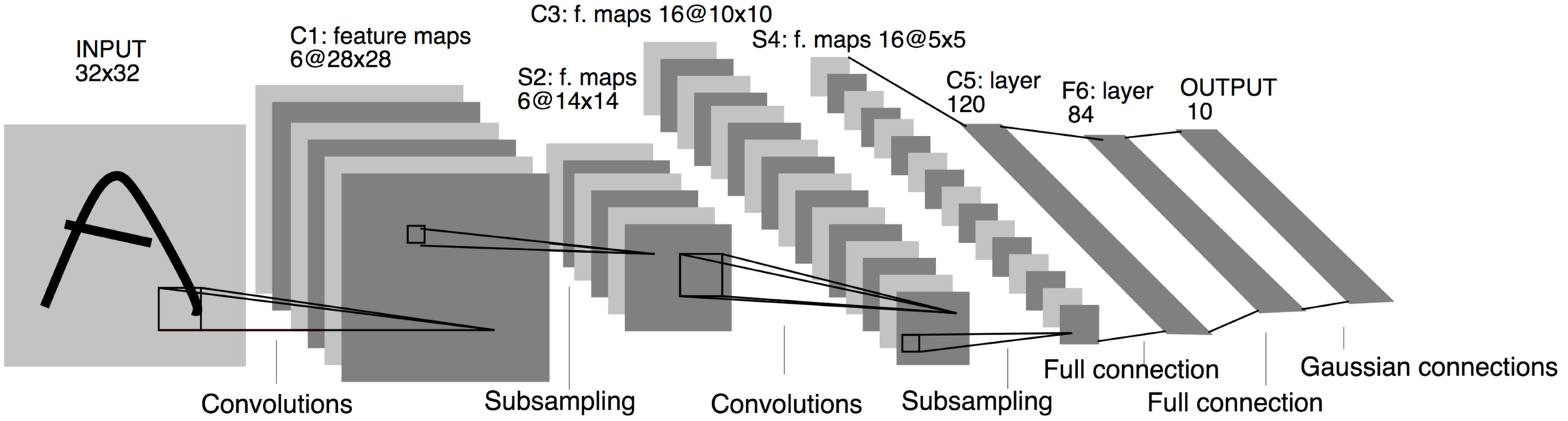


LeNet-5 architecture from Yann LeCunn applied to MNIST:



How many trainable parameters do we have connecting layer S4 and C5?

LeNet-5 architecture from Yann LeCunn applied to MNIST:

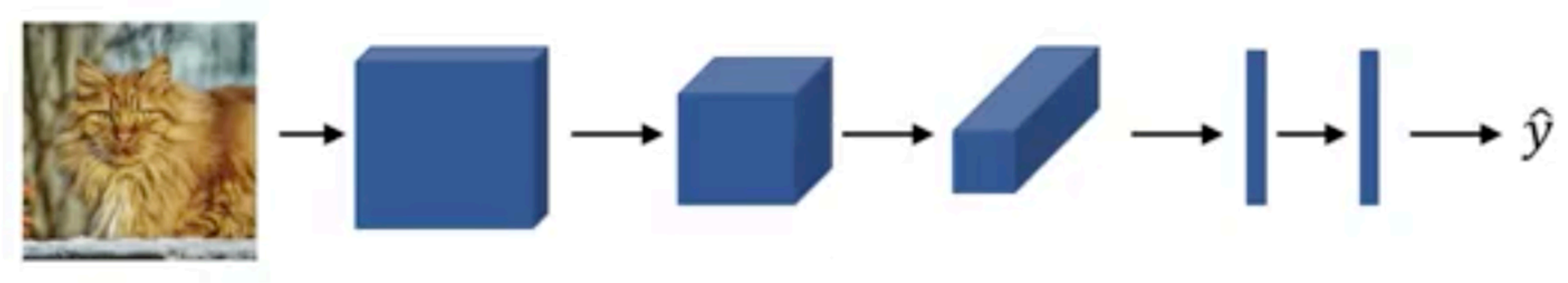


How many trainable parameters do we have connecting layer S4 and C5?

16 x 5 x 5 x 120 + 120 = 48,120

CNN parameters are optimised using a loss function like we did for FNNs:

Example of cat image identification. Training set $\left(x^{(1)}, y^{(1)}\right) \ldots \left(x^{(m)}, y^{(m)}\right)$



$$Cost\ Function\ \mathrm{J} = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}_i, y_i)\ \ \text{(mean of cost functions for each data point)}$$

The value of m depends whether batch, minibatch or stochastic gradient descent is used

CNN parameters are optimised using a loss function like we did for FNNs:

$$Cost\ Function\ \mathrm{J} = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}_i, y_i)\ \ (\text{mean of cost functions for each data point})$$

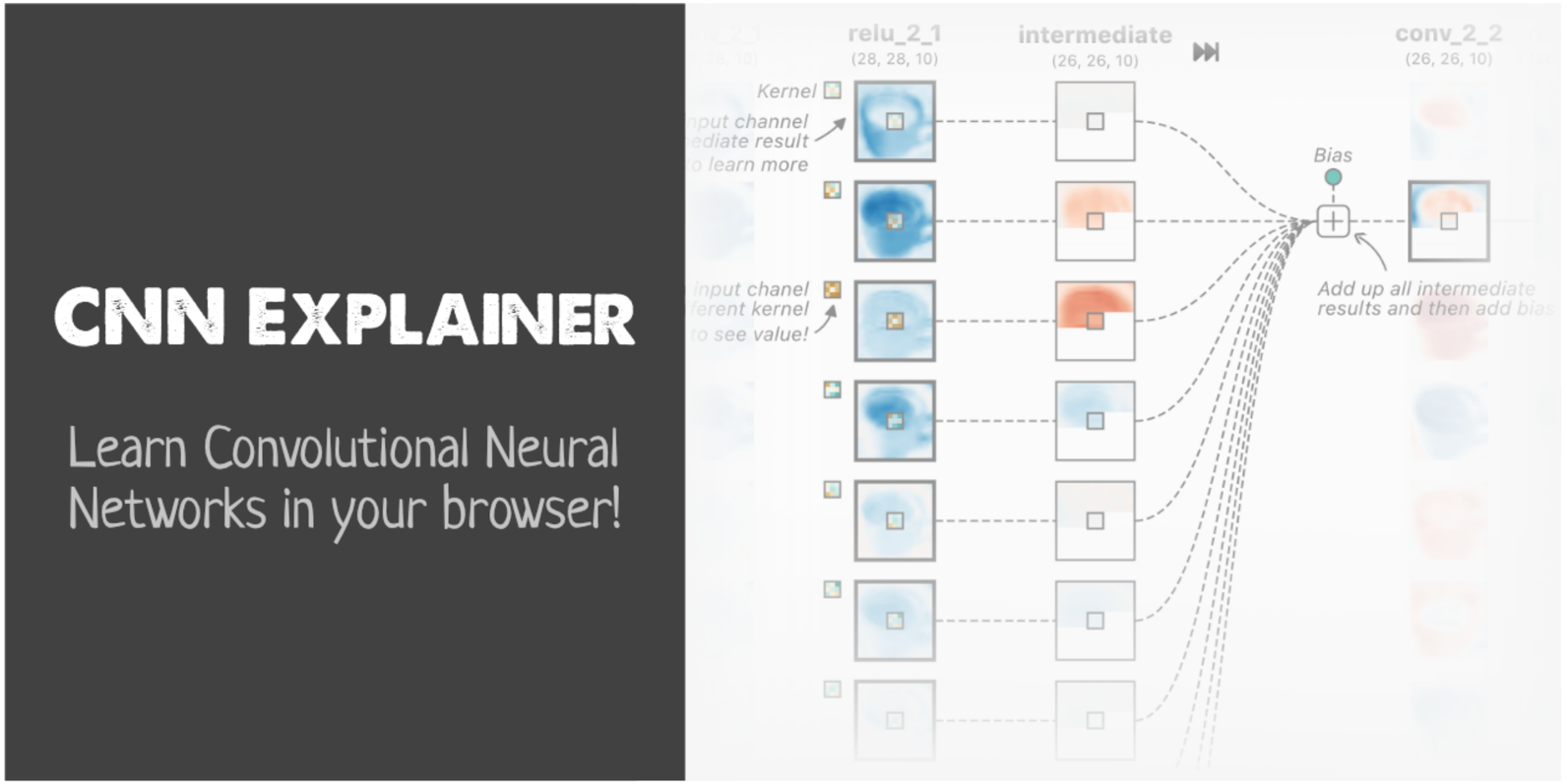
The value of m depends whether batch, minibatch or stochastic gradient descent is used

Let's make sure this is clear:

The **Loss** (or Cost function) is calculated at every iteration as an **average** of all the losses corresponding to each data sample that have seen the same exact network (**in a minibatch**). This will generate a single value of the loss, than we will then use to diagnose the performance of our network; for example, plotting its value at every iteration —or at every epoch— to make sure it decreases as expected (for both training and validation sets).

CNN Visualisation

https://poloclub.github.io/cnn-explainer/



1. Convolutional neural networks
2. Pooling and fully-connected layers
3. **Examples & transposed convolutions**
4. Transfer learning

DATASETS

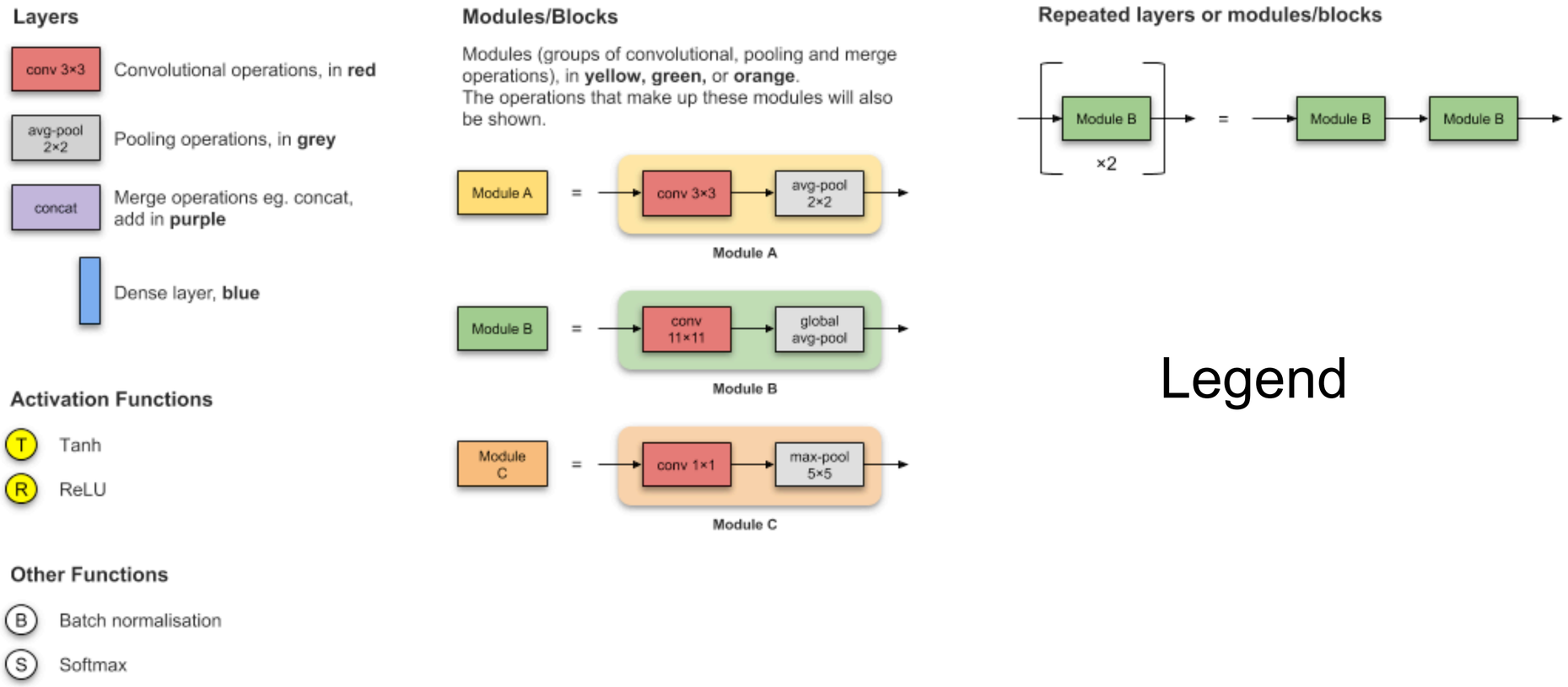
Commonly used datasets in ML:

very deep networks work best with large datasets

- **MNIST**: images of handwritten digits [60k training & 10k test images]
- **CIFAR-10** & **CIFAR-100**: RGB natural images [50k training & 10k test images]

- **Imagenet**: > 15 milion images in 20k classes.

Commonly used CNN network architectures:

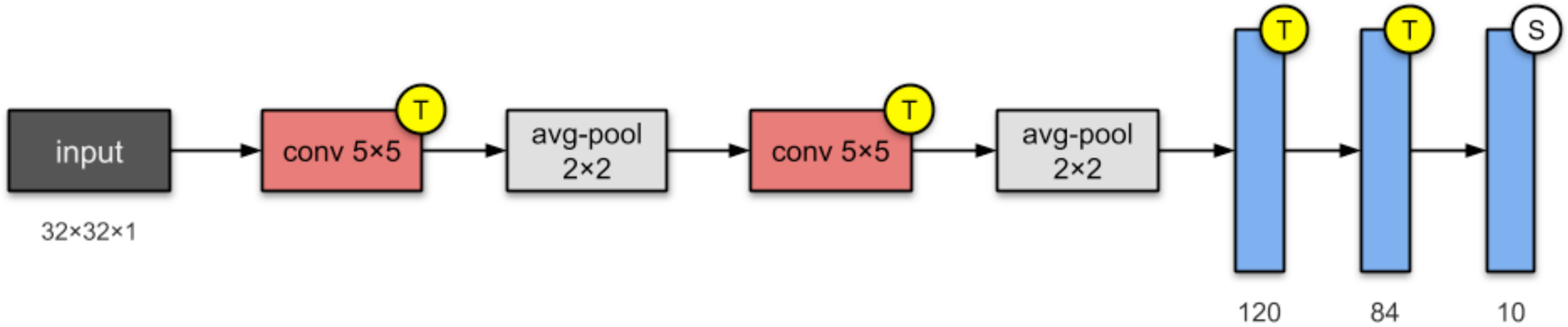


https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d

Commonly used CNN network architectures: LeNet-5 (1998)

- first CNN
- 1998!

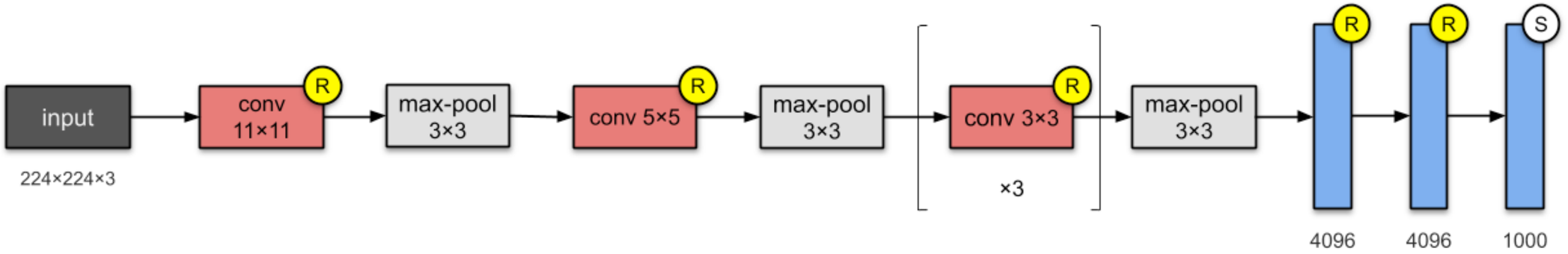
2 convolutions
+
3 fully-connected layers



https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d

Commonly used CNN network architectures: AlexNet (2012)

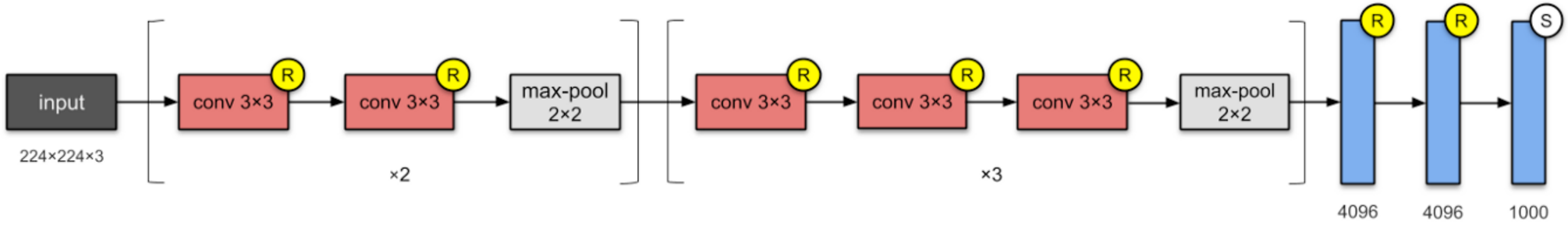
- introduced ReLU activations
- Dropout



https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d

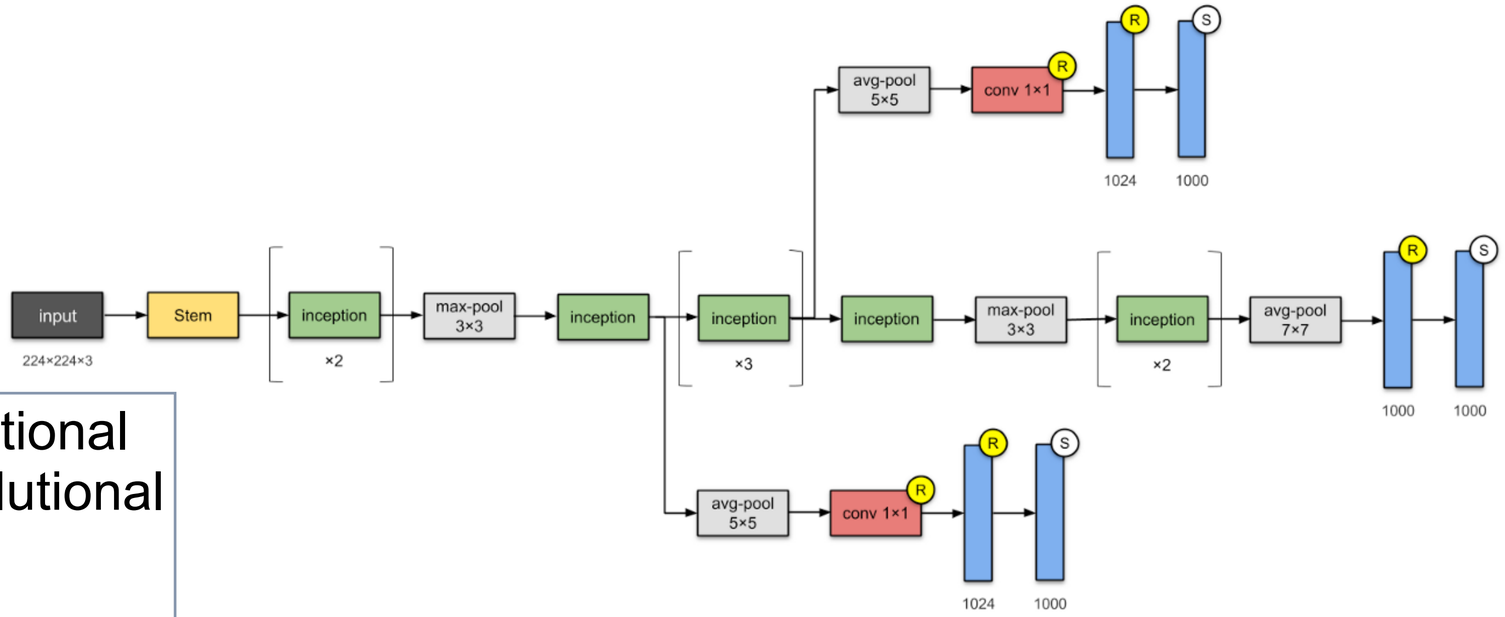
Commonly used CNN network architectures: VGG-16 (2014)

- Introduced very deep networks (2x deeper than AlexNet)
- More layers (16 or 19 for VGG-19) but smaller filters

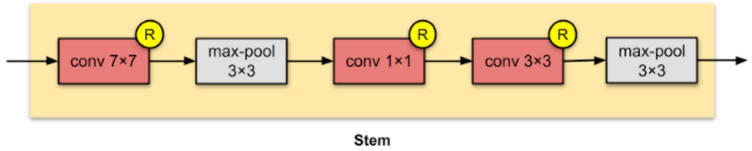


https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d

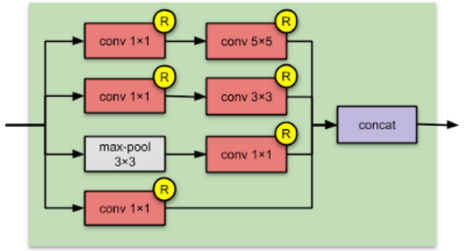
Commonly used CNN network architectures: Inception-v1 (2014)



- Stack modules containing convolutional layers (rather than stacking convolutional layers).
- Stem and Inception modules were named in later versions - **network within a network concept**



Stem

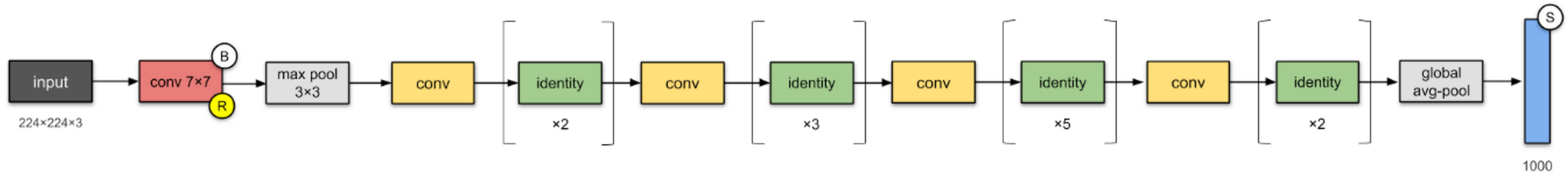


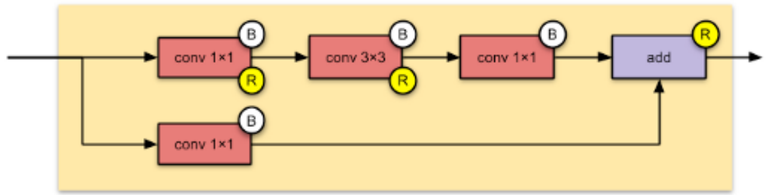
Inception module

https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d

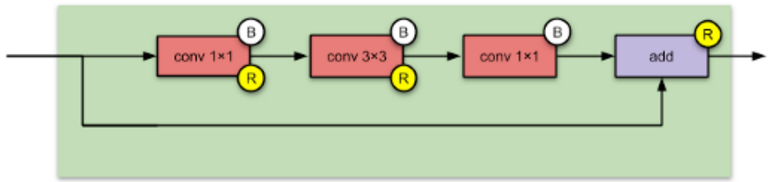
Commonly used CNN network architectures: ResNET-50 (2015)

- Popularised **skip-connections** & **batch normalisation**
- Deep:152 CNN layers





Conv block



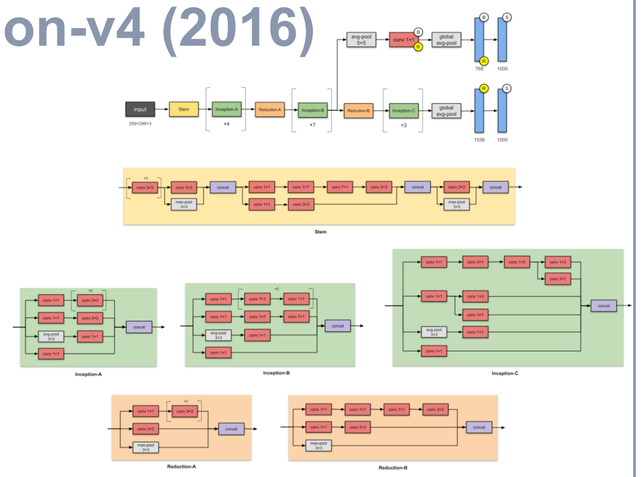
Identity block

https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d

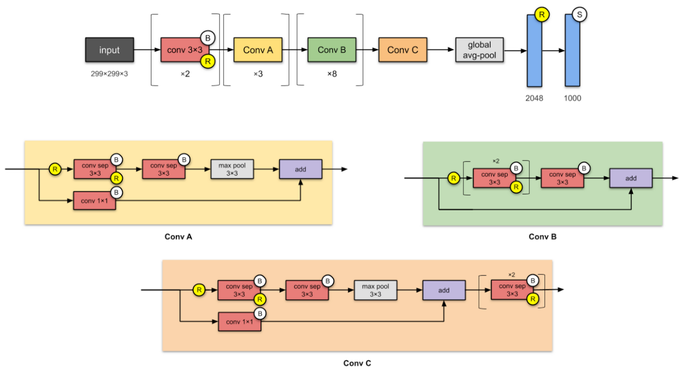
CNN ARCHITECTURES

… and many others

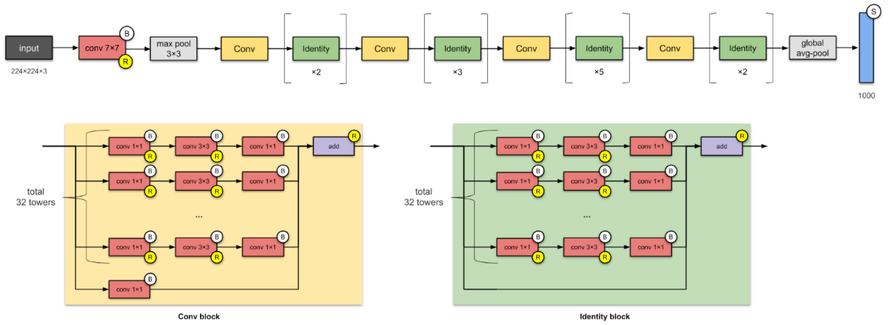
Inception-v4 (2016)



Xception (2016)



ResNEXT-50 (2017)



https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d

Performance and characteristics

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
VGG16	528 MB	0.713	0.901	138,357,544	23
InceptionV3	92 MB	0.779	0.937	23,851,784	159
ResNet50	98 MB	0.749	0.921	25,636,712	-
Xception	88 MB	0.790	0.945	22,910,480	126
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
ResNeXt50	96 MB	0.777	0.938	25,097,128	-

The top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation dataset.

Depth refers to the topological depth of the network. This includes activation layers, batch normalization layers etc.

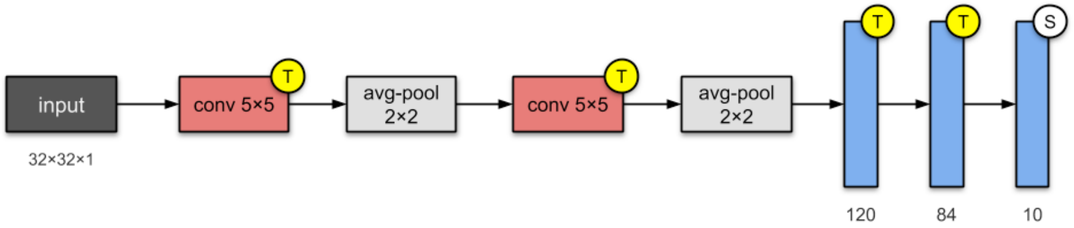
https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d

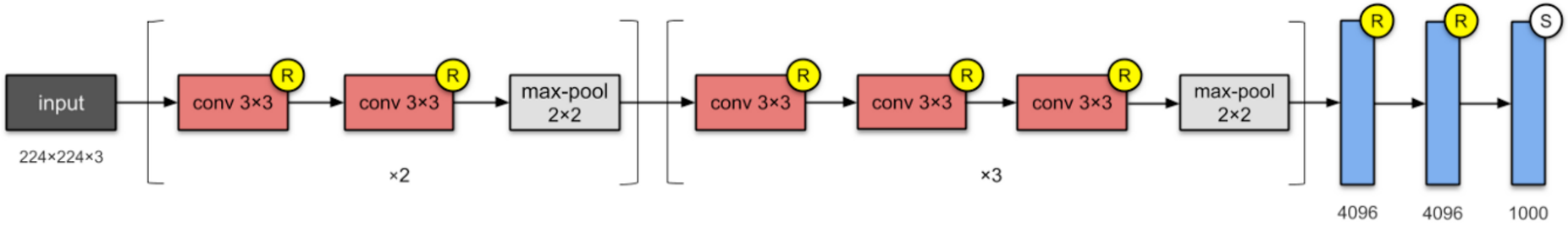
But modern networks (not CNNs) are much larger:

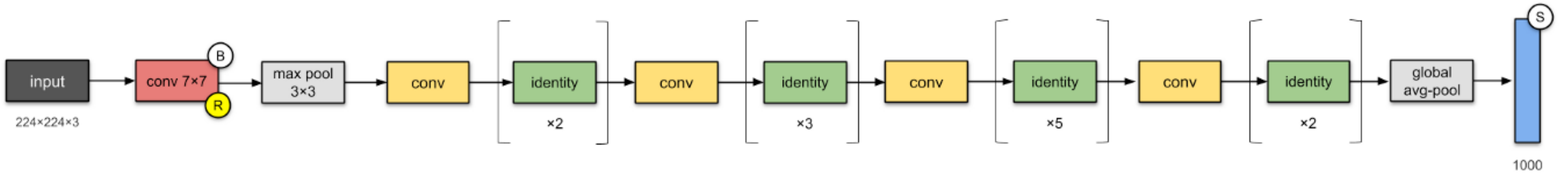
- DALL·E-2 has **3.5 billion parameters**
- GPT-3 has **175 billion parameters** (largest network to date)

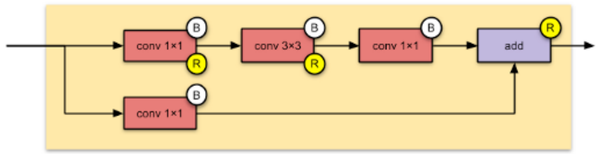
CNN UPSAMPLING

What are the outputs of the CNNs we have seen so far?

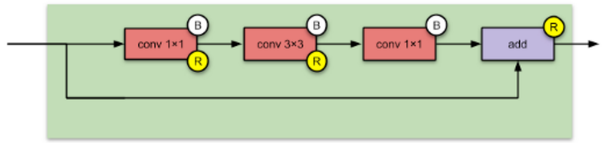








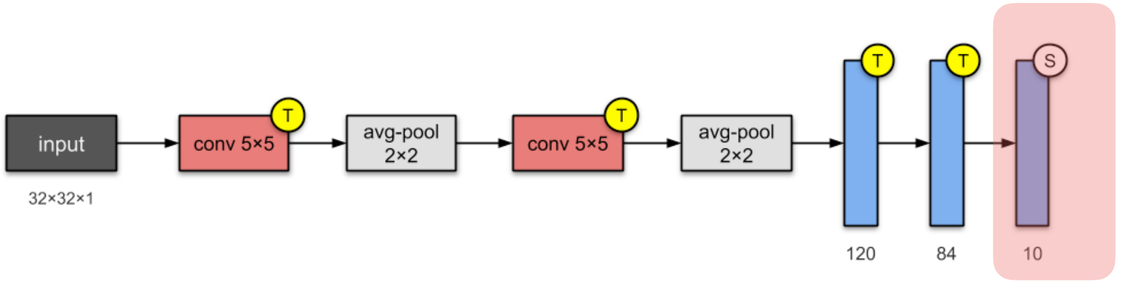
Conv block

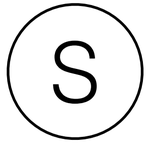


Identity block

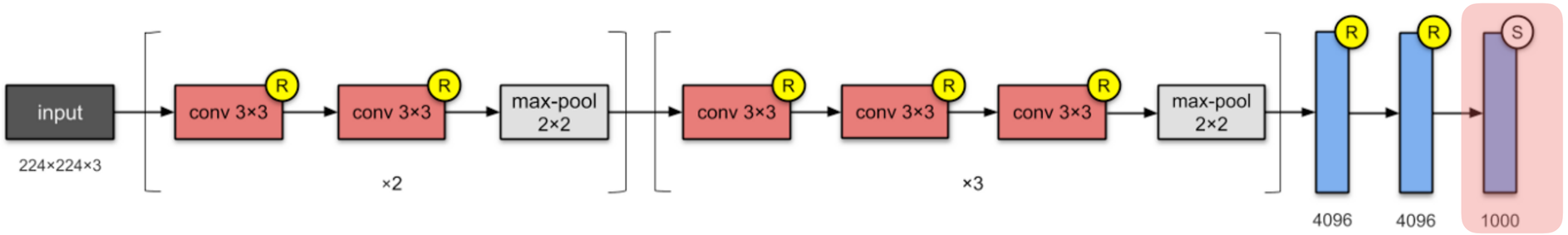
CNN UPSAMPLING

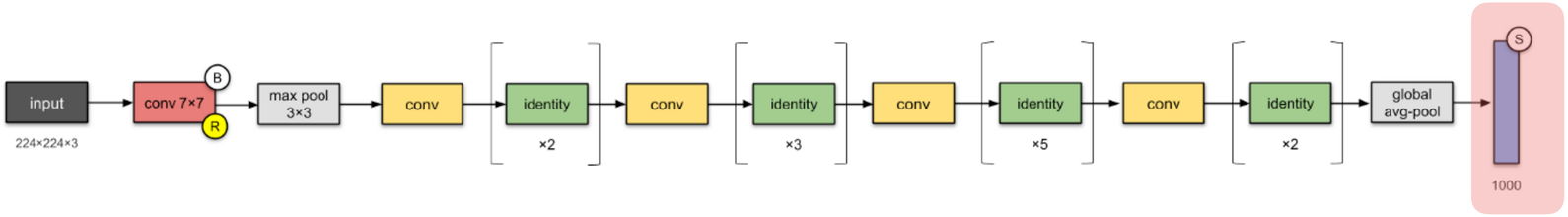
What are the outputs of the CNNs we have seen so far?

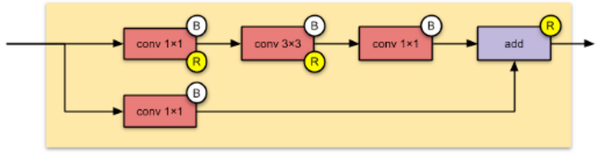




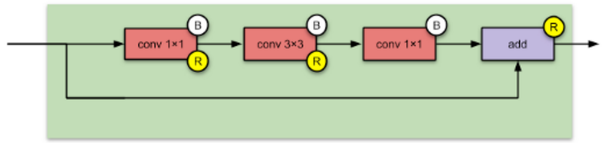
softmax







Conv block

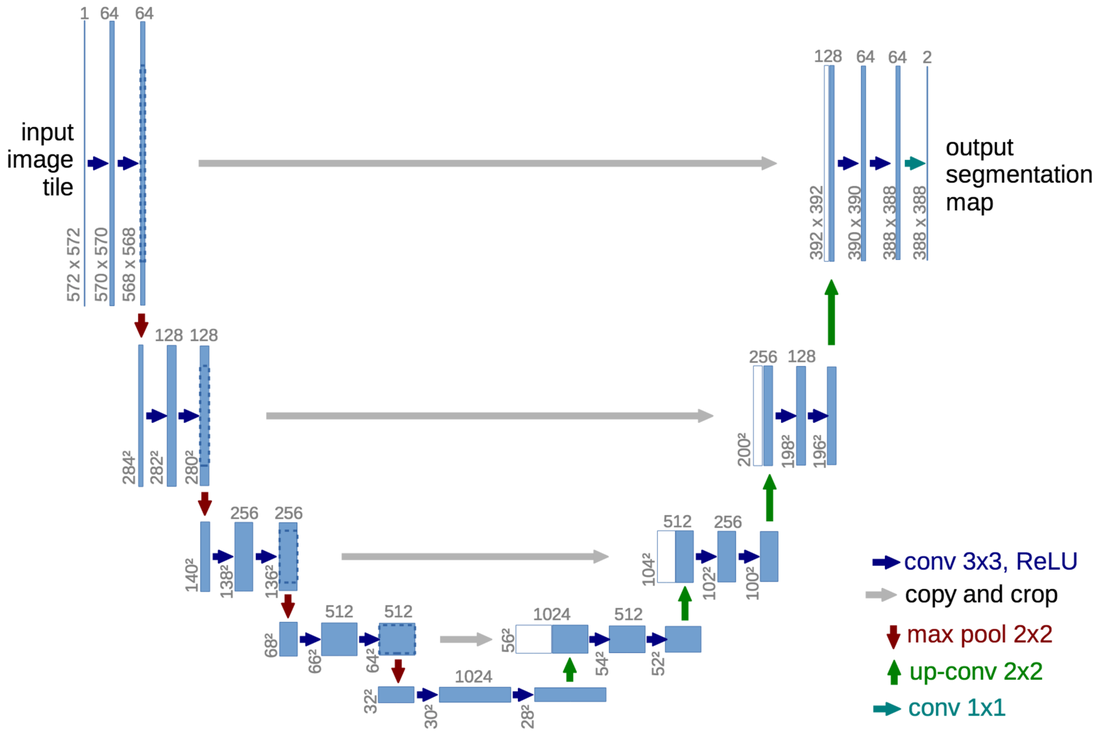


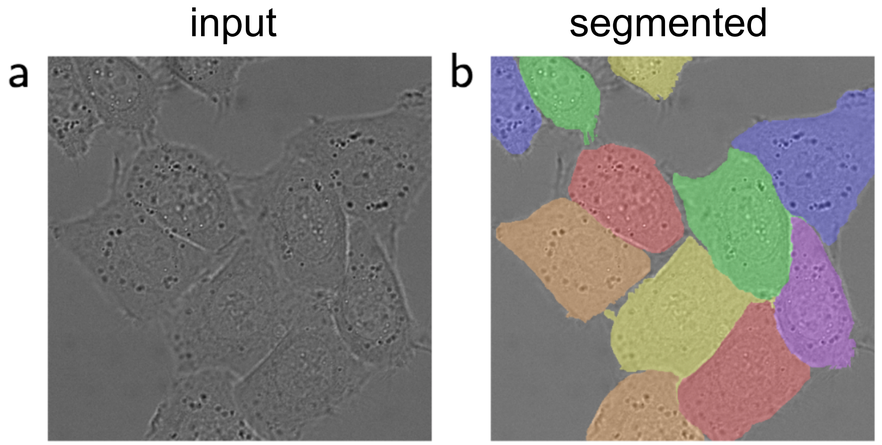
Identity block

CNN UPSAMPLING

But CNNs have other applications. In the field of computer vision, a very common architecture is the **U-Net** which is a type of **convolutional autoencoder** (we will see them next week).

downsampling, upsampling, and skip-connections

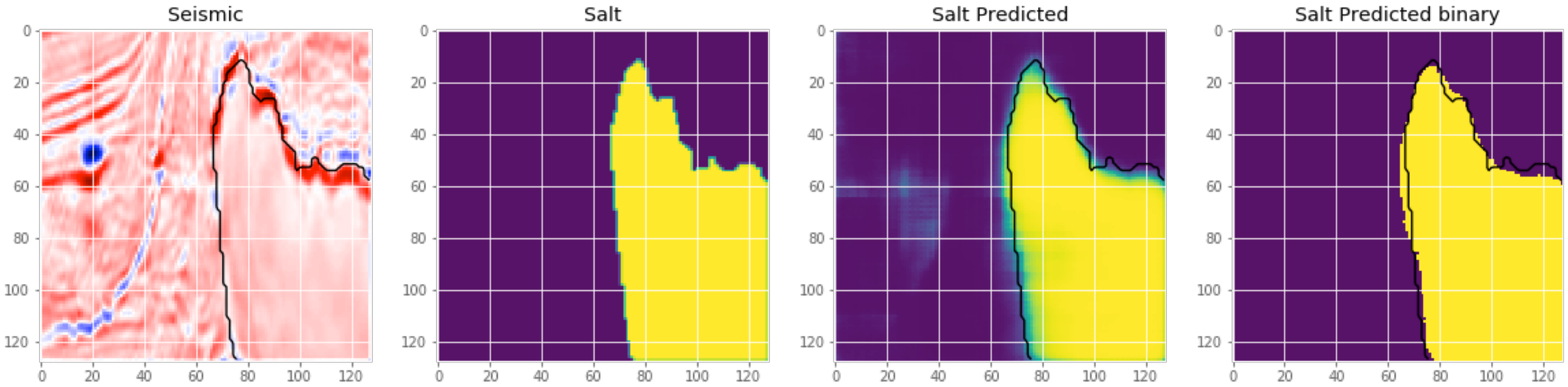




https://arxiv.org/pdf/1505.04597.pdf

Example of applying U-Nets to seismic data:

semantic segmentation of seismic data



https://www.kaggle.com/c/tgs-salt-identification-challenge

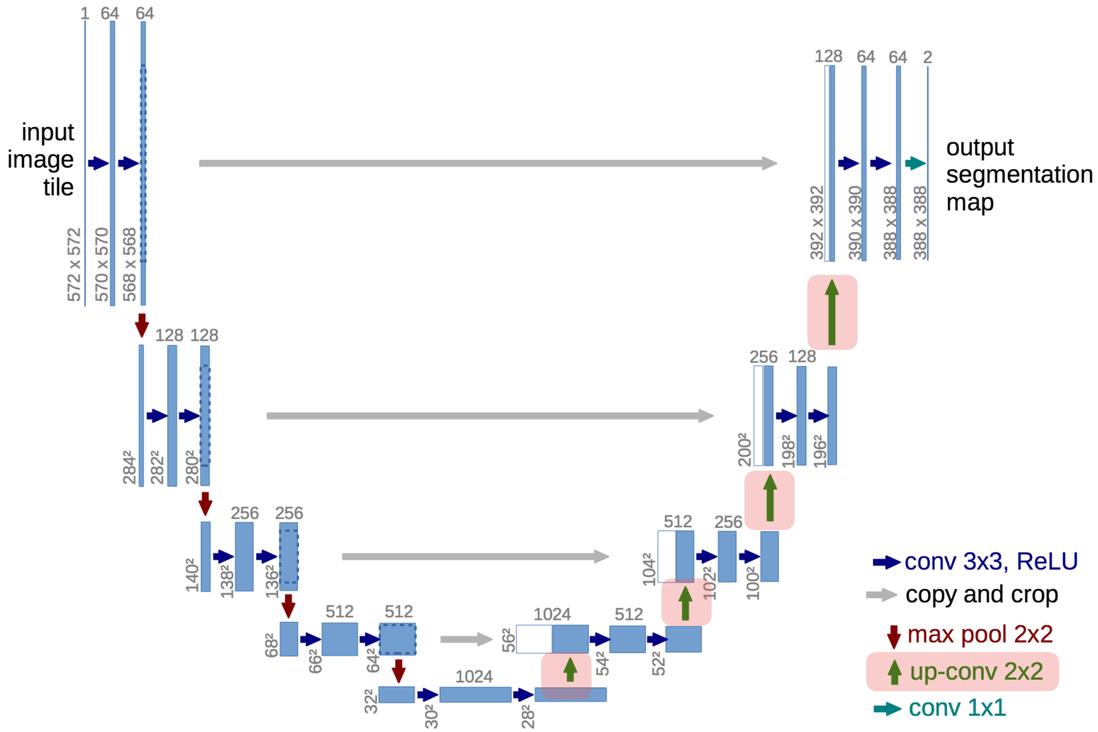
CNN UPSAMPLING

An important operation we perform when we want networks to output images is upscaling.

Convolutions, and maxpool operations downsample data.

How do we upsample it back again?

downsampling, **upsampling**, and skip-connections



Methods for upsampling data:

Nearest Neighbor

1	2
3	4

→

1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

Input: 2 x 2 Output: 4 x 4

"Bed of Nails"

1	2
3	4

→

1	0	2	0
0	0	0	0
3	0	4	0
0	0	0	0

Input: 2 x 2 Output: 4 x 4

Max Pooling
Remember which element was max!

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

Input: 4 x 4

→

5	6
7	8

Output: 2 x 2

→ . . . → Rest of the network

Max Unpooling
Use positions from pooling layer

1	2
3	4

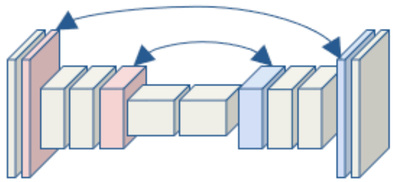
Input: 2 x 2

→

0	0	2	0
0	1	0	0
0	0	0	0
3	0	0	4

Output: 4 x 4

Corresponding pairs of downsampling and upsampling layers



CNN Transposed Convolutions

Transposed convolutions (also called up convolutions, deconvolutions, …)

CNN Transposed Convolutions

Transposed convolutions (also called up convolutions, ~~deconvolutions~~, …)

Ground Truth

¼ Sized Input

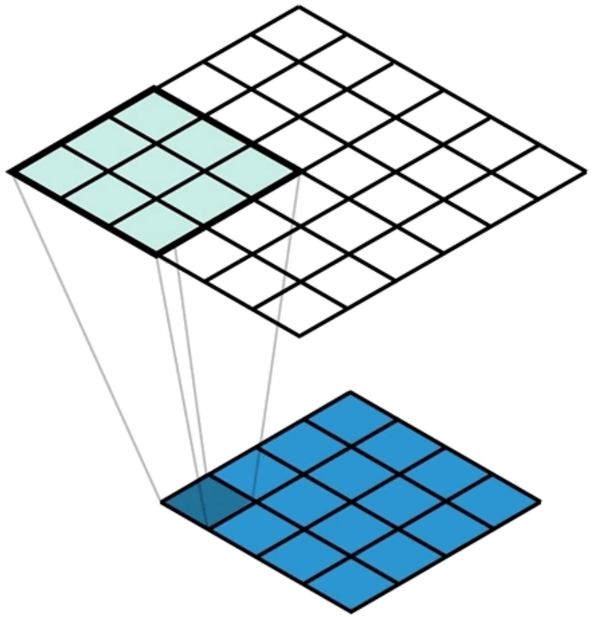
Bicubic

Super Resolution Network

https://medium.com/apache-mxnet/transposed-convolutions-explained-with-ms-excel-52d13030c7e8

CNN Transposed Convolutions

How do transpose convolutions work:



Where we accumulate values over the cells (pixels) outputs.

https://medium.com/apache-mxnet/transposed-convolutions-explained-with-ms-excel-52d13030c7e8

CNN Transposed Convolutions

How do trasnpose convolutions work:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1																						
2			Input							Kernel						Output						
3																						
4																1	2	3	3	2	1	
5			1	1	1	1				1	1	1				2	4	6	6	4	2	
6			1	1	1	1				1	1	1				3	6	9	9	6	3	
7			1	1	1	1				1	1	1				3	6	9	9	6	3	
8			1	1	1	1										2	4	6	6	4	2	
9																1	2	3	3	2	1	
10																						

Note that we are using **padding** here, otherwise the output would be 2x2

https://medium.com/apache-mxnet/transposed-convolutions-explained-with-ms-excel-52d13030c7e8

How do trasnpose convolutions work:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
1																									
2				Input									Kernel						Output						
3																									
4		0	0	0	0	0	0	0	0																
5		0	0	0	0	0	0	0	0										1	5	11	14	8	3	
6		0	0	1	3	2	1	0	0				1	2	3				1	6	15	18	12	3	
7		0	0	1	3	3	1	0	0				0	1	0				4	13	21	21	15	11	
8		0	0	2	1	1	3	0	0				2	1	2				5	17	28	27	25	11	
9		0	0	3	2	3	3	0	0										4	7	9	12	8	6	
10		0	0	0	0	0	0	0	0										6	7	14	13	9	6	
11		0	0	0	0	0	0	0	0																

Note that we are using **padding** here, otherwise the output would be 2x2

https://medium.com/apache-mxnet/transposed-convolutions-explained-with-ms-excel-52d13030c7e8

CNN Transposed Convolutions

The parameters of the transpose convolution kernels are trainable parameters and they can have different stride and padding values.

CONV2D

CLASS `torch.nn.Conv2d(`*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *padding_mode='zeros'*, *device=None*, *dtype=None*`)` [SOURCE]

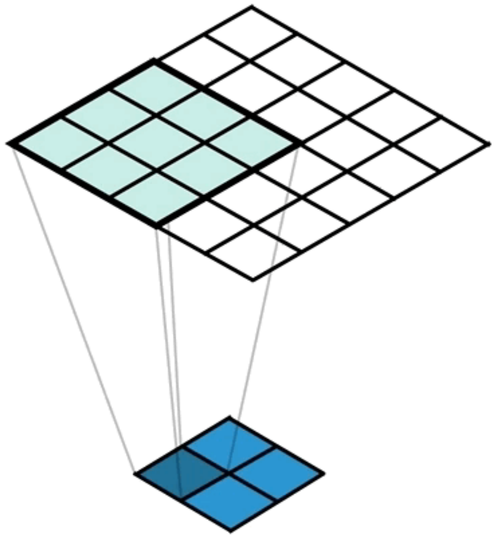
CONVTRANSPOSE2D

CLASS `torch.nn.ConvTranspose2d(`*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *output_padding=0*, *groups=1*, *bias=True*, *dilation=1*, *padding_mode='zeros'*, *device=None*, *dtype=None*`)` [SOURCE]

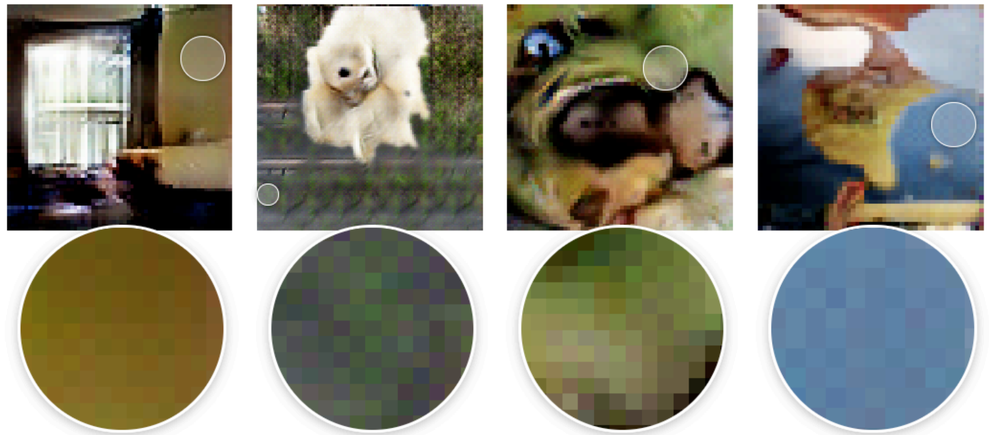
https://medium.com/apache-mxnet/transposed-convolutions-explained-with-ms-excel-52d13030c7e8

CNN Transposed Convolutions

Transpose convolutions can lead to **checkerboard** imprints on the outputs:



stride causes uneven accumulation that follows a regular pattern



This effect can be reduced by using extra convolutional layers, or by changing the stride values

https://distill.pub/2016/deconv-checkerboard/

1. Convolutional neural networks
2. Pooling and fully-connected layers
3. Examples & transposed convolutions
4. **Transfer learning**

What is transfer learning and why is it useful?

> *Transfer learning and domain adaptation refer to the situation where what has been learned in one setting … is exploited to improve generalization in another setting*
>
> *Deep learning, Goodfellow et al, 2016*

- The most well-known CNN designs are **available** on-line and have been successfully **trained** on very large number of images (millions).
- In many applications we often work with a relatively **small number of images**.
- The idea of transfer learning is to use an existing trained CNN model which tries to solve a problem of similar nature and **tailor the model** to our particular application?

Two main strategies in transfer learning:

- Add one (or a few layers) or retrain the last layers of a pre-trained network:

  This strategy assumes that the filters of most of the network do a good job at extracting data features we can use. The last layers, then, act as a final fine-tunning to capture the specific features of our data.

- Retrain the whole network with small learning rates:

  This strategy assumes that as a whole, the network captures data features well, and it only needs a bit of a ‘nudge’ to adapt the network parameters to our particular problem. In this case, we want to keep the underlying abstraction that the network does at different scales, but fine-tune it to our problem.

SUMMARY

- CNNs mostly (but not only) used in **computer vision** problems with 2-D or 3-D data. Common tasks are: image classification, image segmentation, object localization/detection.
- Convolutional neural networks exploit **local context** in the data. Deeper layers tend to represent more abstract data features.
- Transfer Learning: **not limited to CNN** architectures, but common method to benefit from networks trained on large datasets.