

Convolutional Neural Networks 2

Pavlos Protopapas

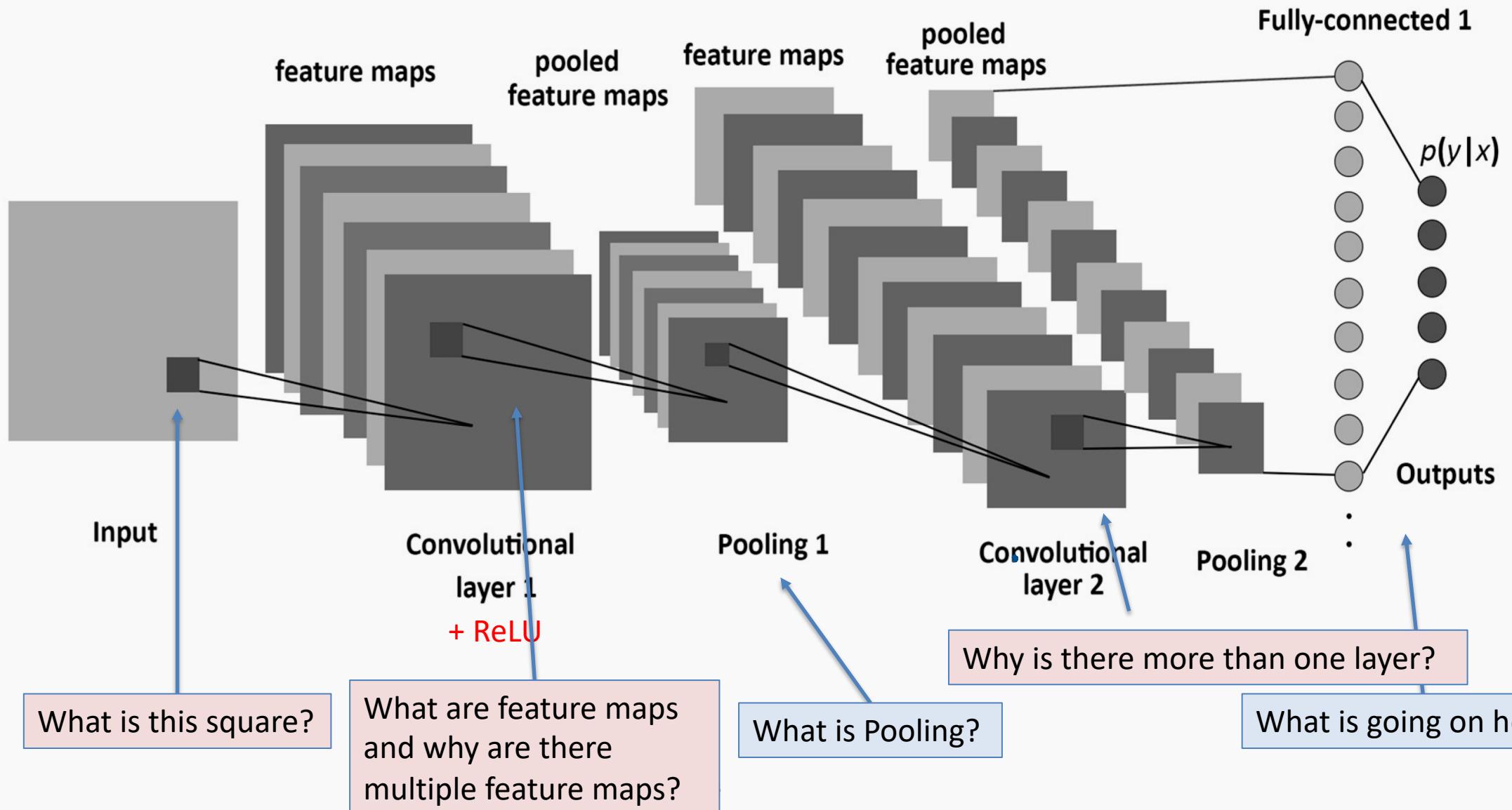
Outline

- Review/Questions
- What are the dimensions of filters, and how do we apply them from one layer to the next?
- Training CNNs
- What Activation Functions do we use?
- What is Pooling?
- Dense Layers

Outline

- **Review/Questions**
- *What are the dimensions of filters, and how do we apply them from one layer to the next?*
- Training CNNs
- What Activation Functions do we use?
- What is Pooling?
- Dense Layers

A Convolutional Network





Grayscale



*



$$\begin{bmatrix} 0.0625 & 0.125 & 0.0625 \\ 0.125 & 0.25 & 0.125 \\ 0.0625 & 0.125 & 0.0625 \end{bmatrix}$$

=

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 20 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

=

$$\begin{bmatrix} -3 & 0 & 3 \\ -6 & 0 & 6 \\ -3 & 0 & 3 \end{bmatrix}$$

=

$$\begin{bmatrix} -3 & -6 & -3 \\ 0 & 0 & 0 \\ 3 & 6 & 3 \end{bmatrix}$$

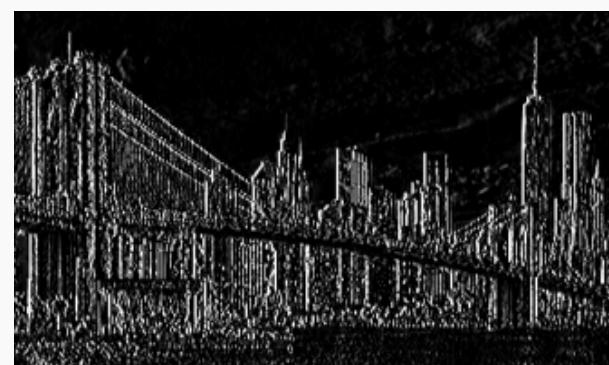
=



BLURRING



SHARPENING PROTOPAPAS

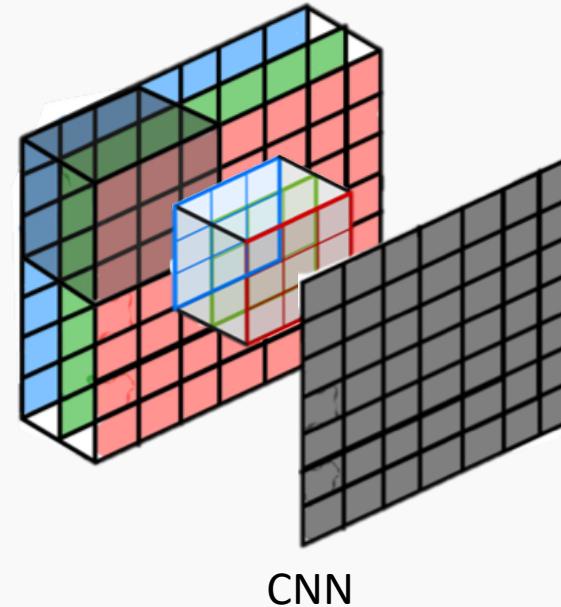
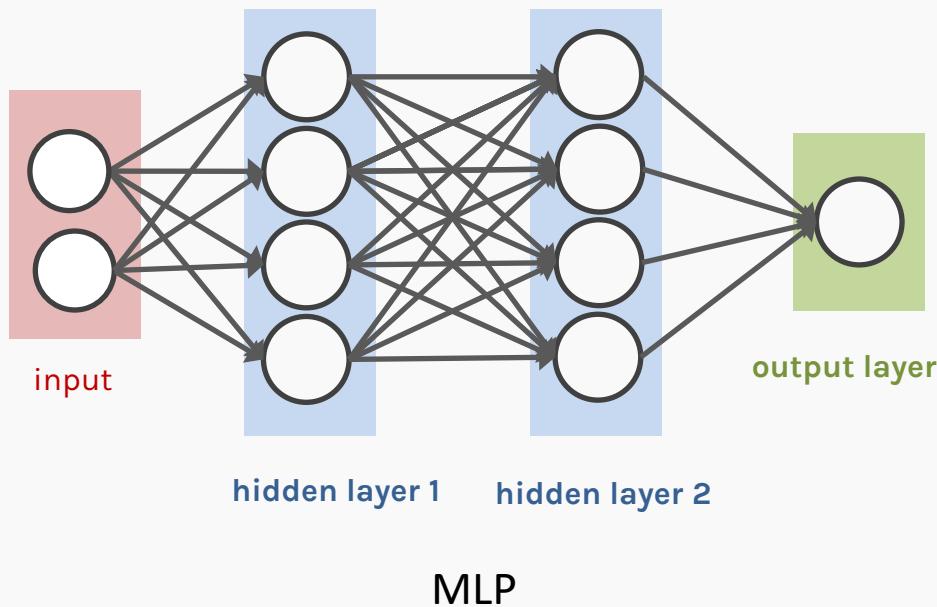


VERTICAL LINES



HORIZONTAL LINES

Basics of CNNs



- CNNs are composed of layers, but those layers are not fully connected: they have **filters**, sets of cube-shaped weights, that are applied throughout the image.
- Each 2D slice of a filter is called **kernel**.
- These filters introduce **translation invariance** and **parameter sharing**.
- How are they applied? **Convolution!**

Outline

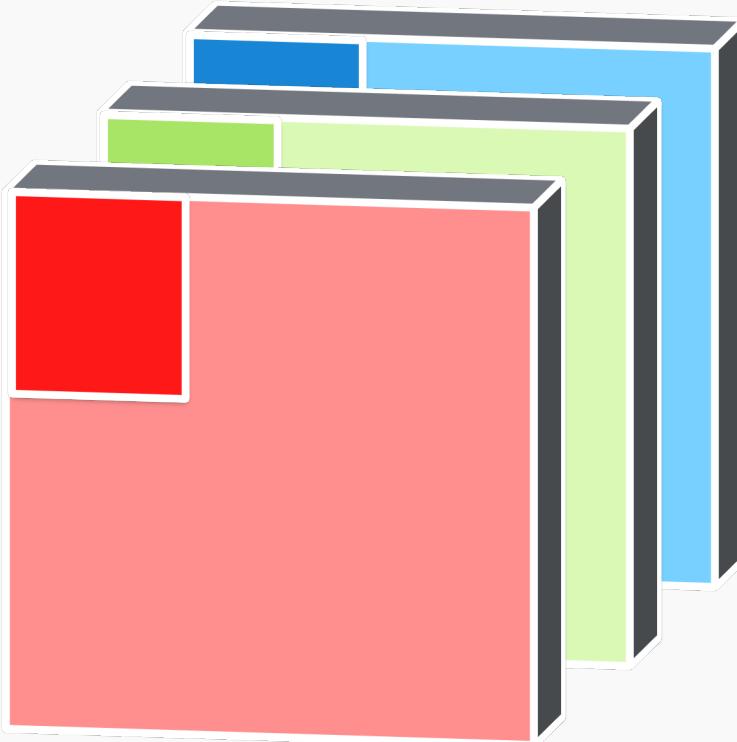
- Review/Questions
- **What are the dimensions of filters, and how do we apply them from one layer to the next?**
- Training CNNs
- What Activation Functions do we use?
- What is Pooling?
- Dense Layers

Example: A convolutional layer with one 3×3 filter that takes an 32×32 RGB image as input.

Input

`size=32x32`

`channels=3`

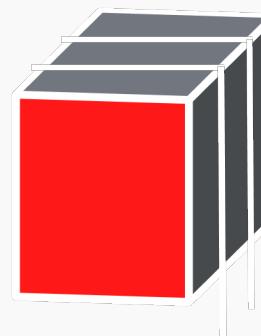


1 Filter

`size=3x3x3`

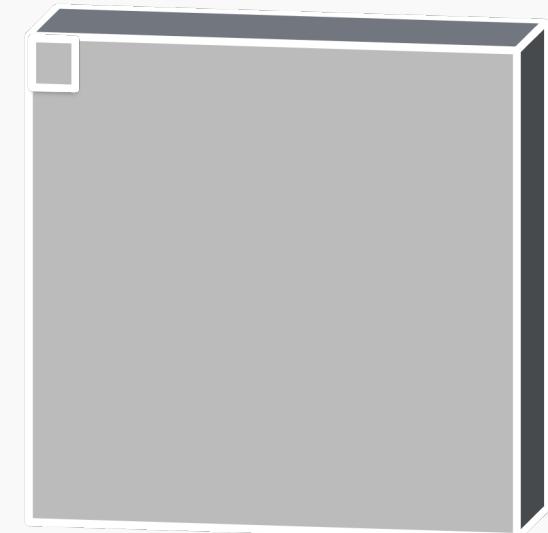
`stride = 1`

`padding = same`



Output

`size=32x32`



How many parameters does the layer have?

Example 2: A convolutional layer with 16 3x3 filters that takes 32x32 RGB image as input.

A convolutional layer with 16 3x3 filters that takes 32x32 RGB image as input.

- How many parameters does the layer have?

Example 2: A convolutional layer with 16 3x3 filters that takes 32x32 RGB image as input.

A convolutional layer with 16 3x3 filters that takes 32x32 RGB image as input.

- How many parameters does the layer have?

16

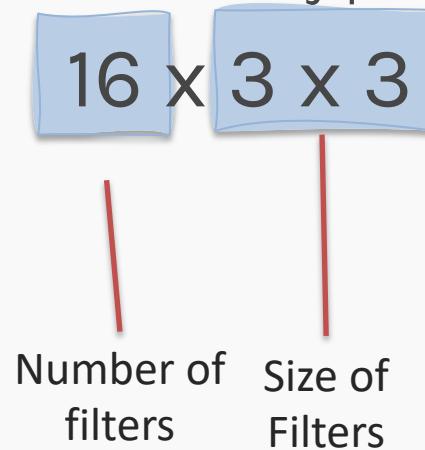


Number of
filters

Example 2: A convolutional layer with 16 3x3 filters that takes 32x32 RGB image as input.

A convolutional layer with 16 3x3 filters that takes 32x32 RGB image as input.

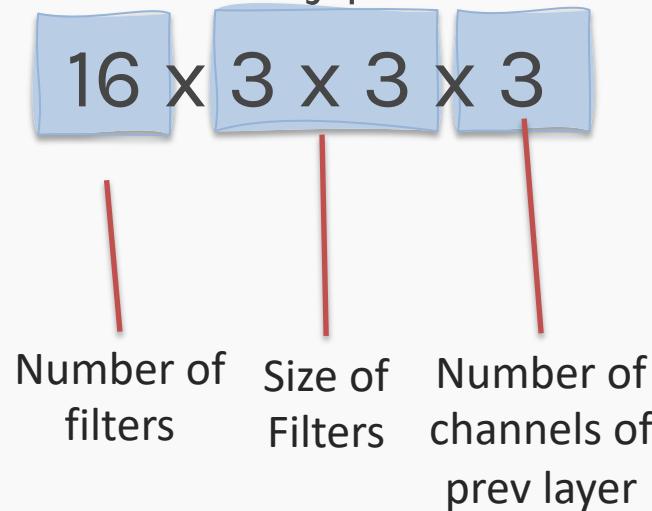
- How many parameters does the layer have?



Example 2: A convolutional layer with 16 3x3 filters that takes 32x32 RGB image as input.

A convolutional layer with 16 3x3 filters that takes 32x32 RGB image as input.

- How many parameters does the layer have?



Example 2: A convolutional layer with 16 3x3 filters that takes 32x32 RGB image as input.

A convolutional layer with 16 3x3 filters that takes 32x32 RGB image as input.

- How many parameters does the layer have?

$$16 \times 3 \times 3 \times 3 + 16$$

Number of filters Size of Filters Number of channels of prev layer Biases (one per filter)

The diagram illustrates the calculation of the number of parameters for a convolutional layer. It shows the formula $16 \times 3 \times 3 \times 3 + 16$. Below the formula, four red lines point from labels to specific parts of the expression: 'Number of filters' points to the first '16', 'Size of Filters' points to the three '3's, 'Number of channels of prev layer' points to the second '3', and 'Biases (one per filter)' points to the '+ 16' term.

Example 2: A convolutional layer with 16 3x3 filters that takes 32x32 RGB image as input.

A convolutional layer with 16 3x3 filters that takes 32x32 RGB image as input.

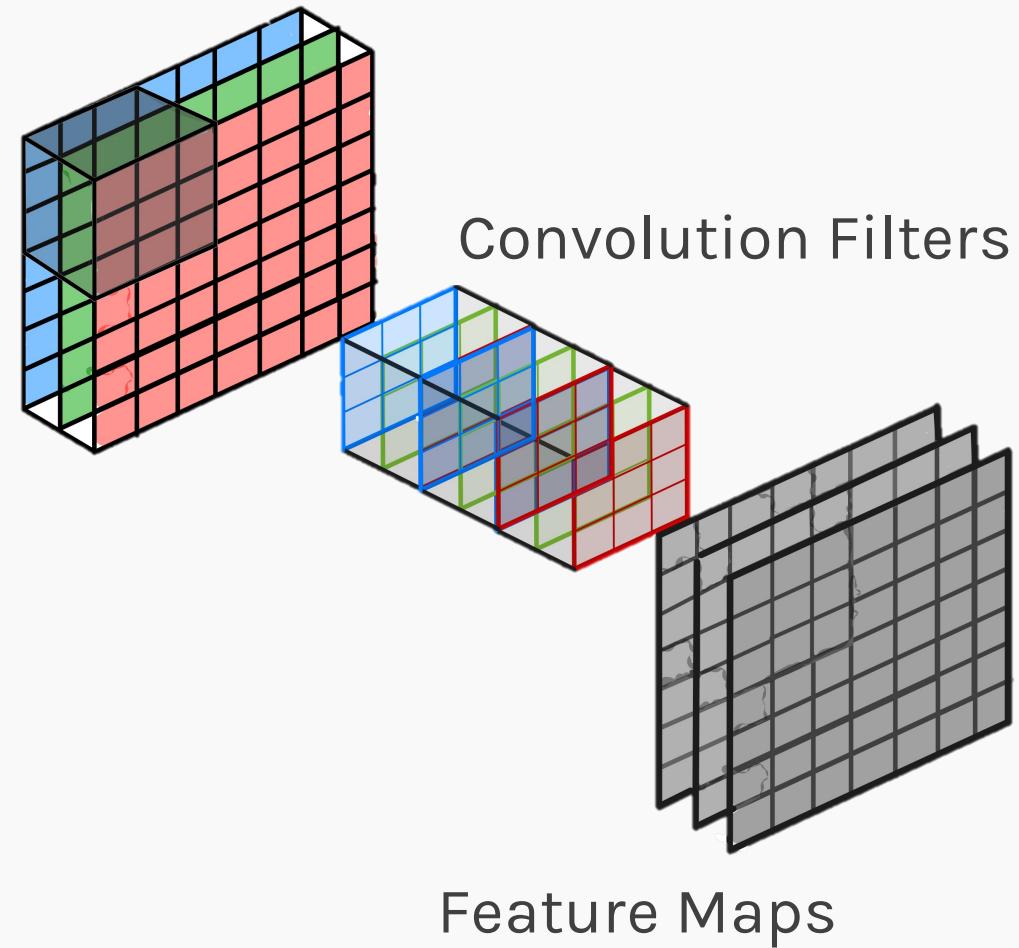
- How many parameters does the layer have?

$$16 \times 3 \times 3 \times 3 + 16 = 448$$

Number of filters Size of Filters Number of channels of prev layer Biases (one per filter)

Convolutional layers (cont.)

- **To be clear:** each filter is convolved with the entirety of the **3D input cube** but generates a **2D feature map**.
- Because we have multiple filters, we end up with a 3D output: **one 2D feature map per filter**.
- The feature map dimension can **change drastically** from one conv layer to the next: we can enter a layer with a $32 \times 32 \times 16$ input and exit with a $32 \times 32 \times 128$ output if that layer has 128 filters.



Outline

- Review/Questions
- What are the dimensions of filters, and how do we apply them from one layer to the next?
- **Training CNNs**
- What Activation Functions do we use?
- What is Pooling?
- Dense Layers

Training CNN

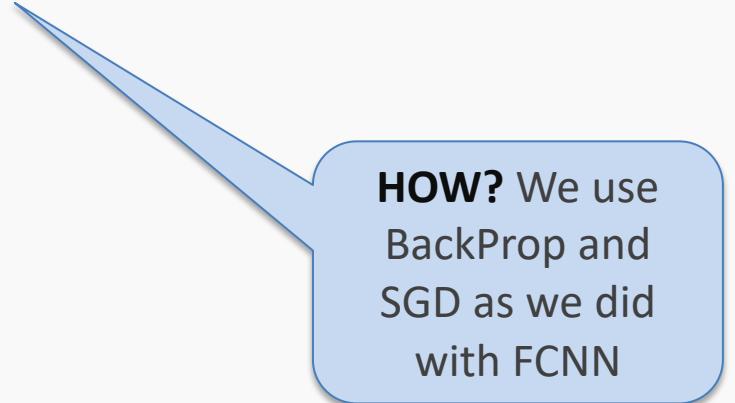
In a convolutional layer, we are basically applying multiple filters over the image to extract different features.

But most importantly, [we are learning those filters!](#)

Training CNN

In a convolutional layer, we are basically applying multiple filters over the image to extract different features.

But most importantly, **we are learning those filters!**



HOW? We use BackProp and SGD as we did with FCNN

Training CNN

In a convolutional layer, we are basically applying multiple filters over the image to extract different features.

But most importantly, **we are learning those filters!**

One thing we're missing: non-linearity.

HOW? We use BackProp and SGD as we did with FCNN

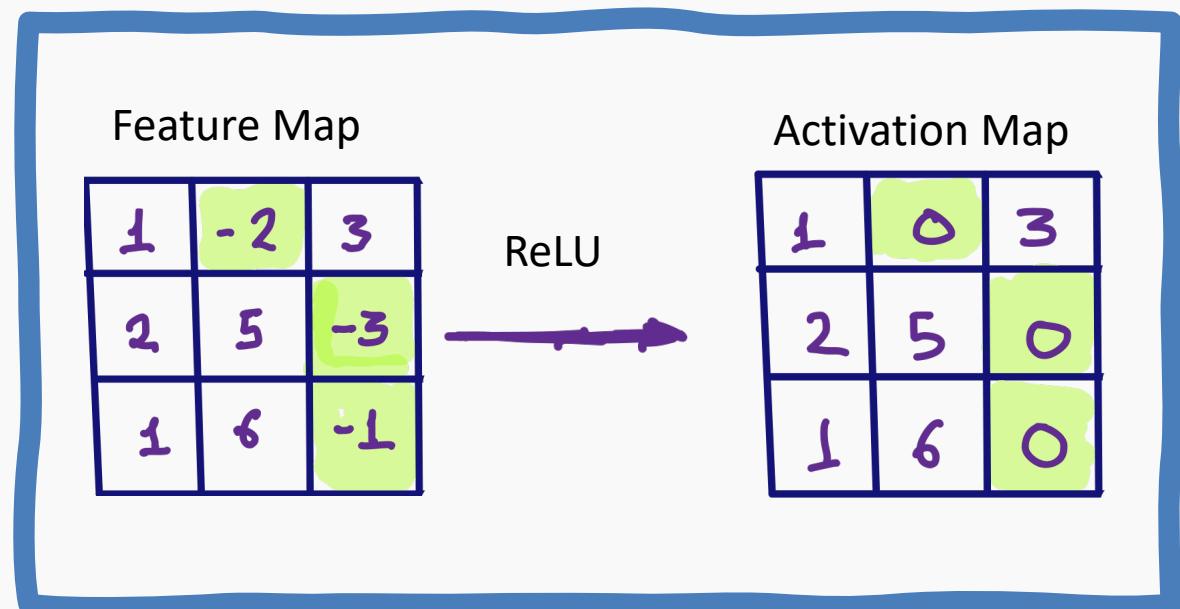
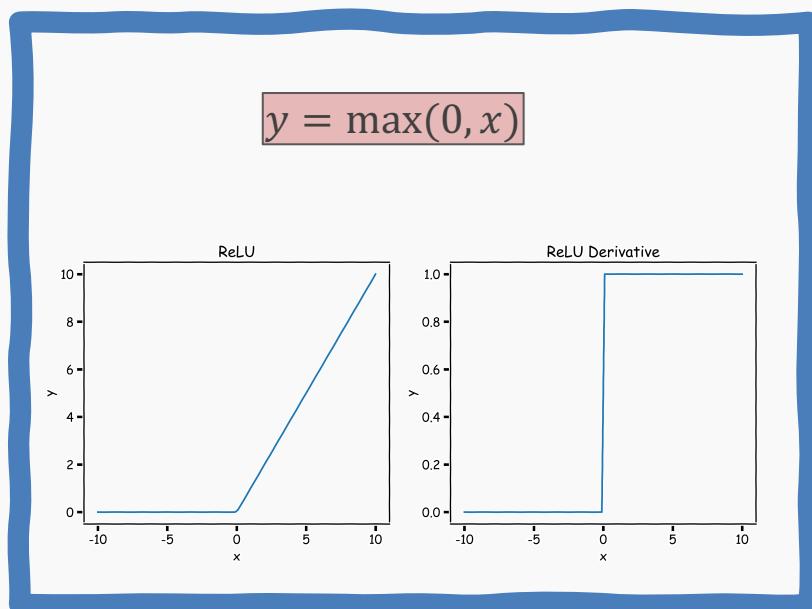
Outline

- Review/Questions
- What are the dimensions of filters, and how do we apply them from one layer to the next?
- Training CNNs
- **What Activation Functions do we use?**
- What is Pooling?
- Dense Layers

ReLU

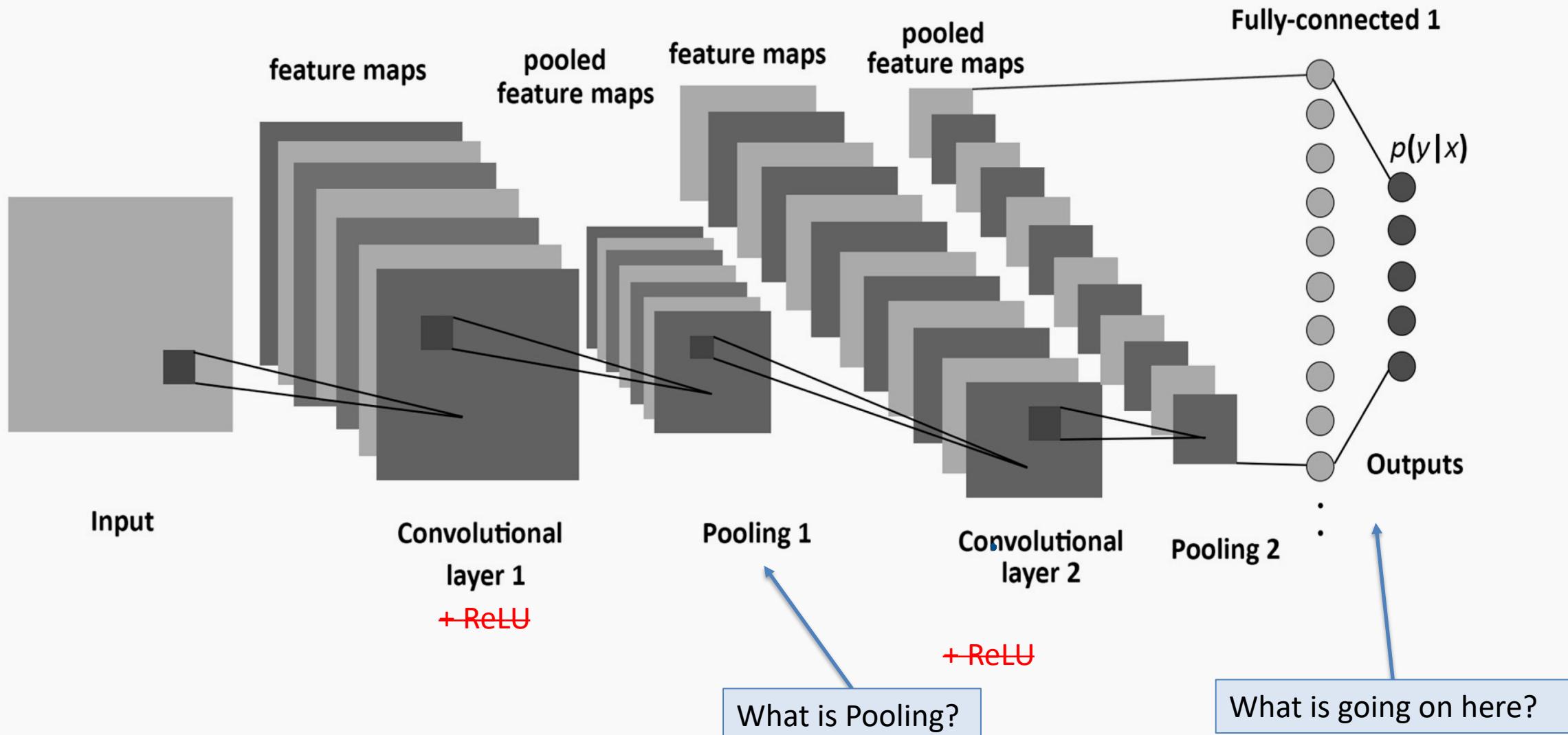
We apply non-linear activation **after** convolution as we did for FCNN.

The most successful non-linear activation function for CNNs is the **Rectified Non-Linear unit** (ReLU):



This combats the vanishing gradient problem occurring in sigmoid, it is easier to compute, and **generates sparsity**.

A Convolutional Network



Convolutional layers so far

Multiple parameters to define:

- number of filters
- size of kernels
- stride
- padding
- activation function to use

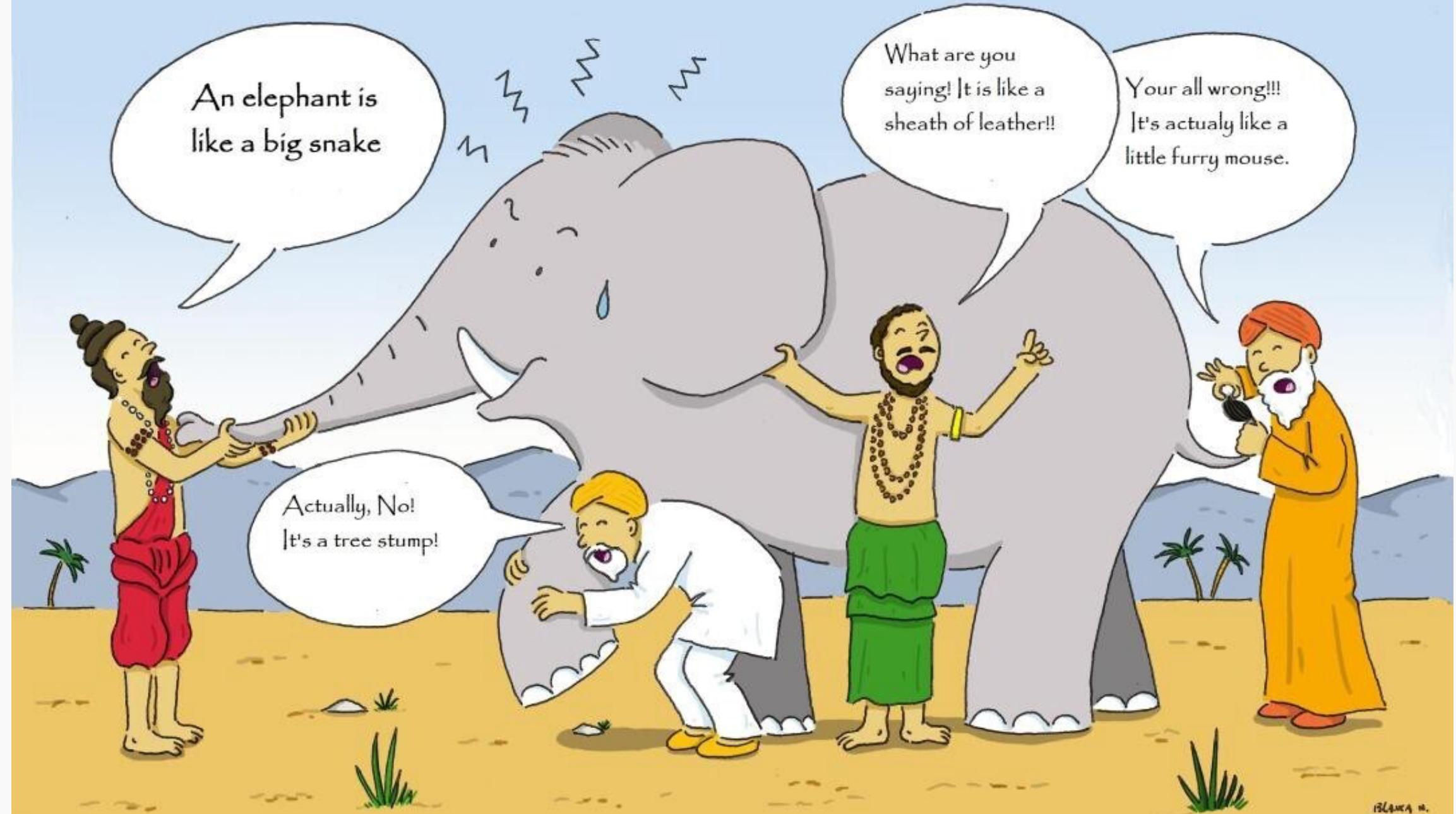
Convolutional layers so far

Multiple parameters to define:

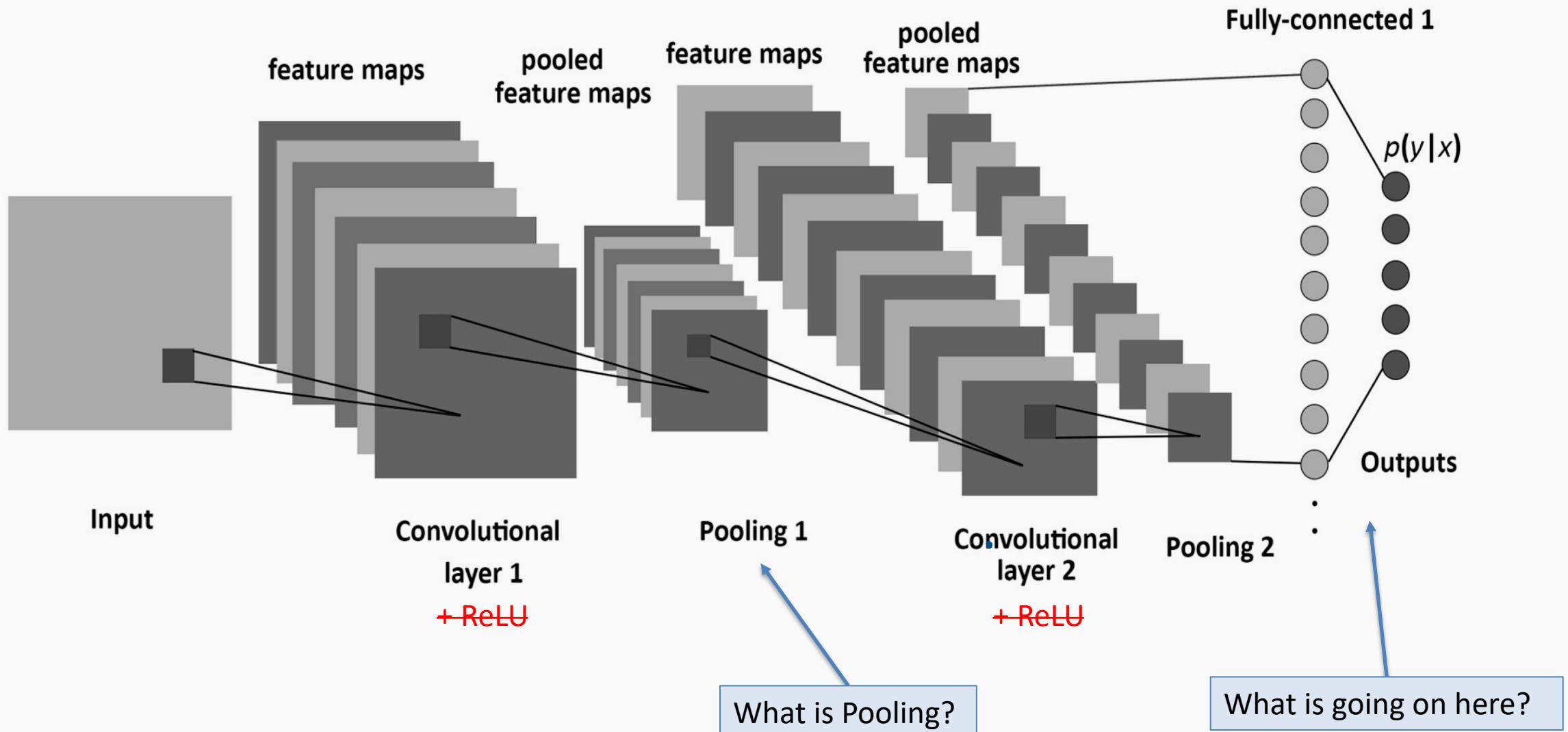
```
number of output filters in the convolution
height and width of the 2D convolution window
"valid" means no padding.
"same" results in padding with zeros evenly
set the activation.
Default activation is 'linear'

tf.keras.layers.Conv2D(
    filters, kernel_size, strides=(1, 1), padding='valid',
    data_format=None, dilation_rate=(1, 1), groups=1, activation=None,
    use_bias=True, kernel_initializer='glorot_uniform',
    bias_initializer='zeros', kernel_regularizer=None,
    bias_regularizer=None, activity_regularizer=None, kernel_constraint=None,
    bias_constraint=None, **kwargs
)
```

strides of the convolution along the height and width



A Convolutional Network



Outline

- Review/Questions
- What are the dimensions of filters, and how do we apply them from one layer to the next?
- Training CNNs
- What Activation Functions do we use?
- **What is Pooling?**
- Dense Layers

Pooling

A **pooling** layer is a new layer added after the convolutional layer. Specifically, it is added after a nonlinearity (e.g. ReLU) has been applied to the feature maps*.

The pooling layer operates upon each activation map separately to create a new set of the same number of pooled feature maps.

Example:

1	1	2	5
5	7	7	8
3	1	1	0
1	2	3	4

max pool with 2x2 window
and stride 2

7	

* Maxpooling could be applied before ReLU.

Pooling

A **pooling** layer is a new layer added after the convolutional layer. Specifically, it is added after a nonlinearity (e.g. ReLU) has been applied to the feature maps.

The pooling layer operates upon each activation map separately to create a new set of the same number of pooled feature maps.

Example:

1	1	2	5
5	7	7	8
3	1	1	0
1	2	3	4

max pool with 2x2 window
and stride 2

7	8

Pooling

A **pooling** layer is a new layer added after the convolutional layer. Specifically, it is added after a nonlinearity (e.g. ReLU) has been applied to the feature maps.

The pooling layer operates upon each activation map separately to create a new set of the same number of pooled feature maps.

Example:

1	1	2	5
5	7	7	8
3	1	1	0
1	2	3	4

max pool with 2x2 window
and stride 2

7	8
3	

Pooling

A **pooling** layer is a new layer added after the convolutional layer. Specifically, it is added after a nonlinearity (e.g. ReLU) has been applied to the feature maps.

The pooling layer operates upon each activation map separately to create a new set of the same number of pooled feature maps.

Example:

1	1	2	5
5	7	7	8
3	1	1	0
1	2	3	4

max pool with 2x2 window
and stride 2

7	8
3	4

Pooling

A **pooling** layer is a **new** layer added after the convolutional layer. Specifically, it is added after a nonlinearity (e.g. ReLU) has been applied to the feature maps*.

The pooling layer operates upon each feature map separately to create a new set of the same number of pooled feature maps.

Pooling involves selecting:

- A pooling **operation**, much like a filter, to be applied to feature maps: e.g. max, mean, median.
- The **size** of the pooling operation.
- The **stride**.

* Maxpooling could be applied before ReLU.

Pooling

Pooling involves selecting:

- A pooling **operation**, much like a filter, to be applied to feature maps: e.g. max, mean, median.
- The **size** of the pooling operation.
- The **stride**.

The size of the pooling operator must be smaller than the size of the feature map; specifically, it is almost always 2×2 applied with a stride of 2 using **max pooling**.

Pooling

Pooling involves selecting:

- A pooling **operation**, much like a filter, to be applied to feature maps: e.g. max, mean, median.
- The **size** of the pooling operation.
- The **stride**.

The size of the pooling operator must be smaller than the size of the feature map; specifically, it is almost always 2×2 applied with a stride of 2 using **max pooling**.

Invariant to small, “**local transitions**”

Face detection: enough to check the presence of eyes, not their precise location

Reduces input size of the **final fully connected layers (more later)**

No learnable parameters

Pooling: more examples with stride 2x2

1	1	2	5
5	7	7	8
3	1	1	0
1	2	3	4

max pool with 2x2 window
and stride 2x2

7	8
3	4

1	1	2	5
5	7	7	8
3	1	1	0
1	2	3	4

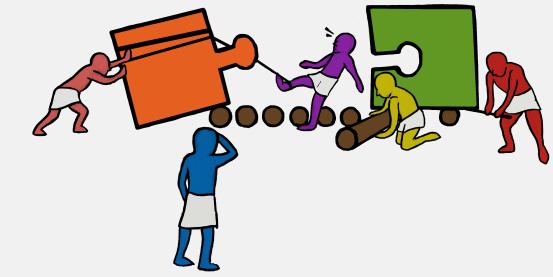
mean pool with 2x2 window
and stride 2x2

3.5	5.5
1.75	2

Exercise: Pooling mechanics

The aim of this exercise is to understand the tf.keras implementation of average and max pooling:

- implement Max Pooling by building a model with a single MaxPooling2D layer
- Next, implement Average Pooling by building a model with a single AvgPooling2D layer
- Use the helper code to visualize the output
- Use the hint we provide



How far the pooling window moves for each pooling step

```
tf.keras.layers.MaxPool2D(  
    pool_size=(2, 2), strides=None, padding='valid', data_format=None,  
    **kwargs  
)
```

Window-size over which to take the maximum

Channels first or channels last

"valid" means no padding, "same" results in padding

This block contains a snippet of Python code for a MaxPool2D layer. Red arrows point from the explanatory text below to specific parts of the code: one arrow points to the 'pool_size' parameter, another to the 'padding' parameter, and a third to the 'data_format' parameter. The text 'Channels first or channels last' is aligned with the 'data_format' parameter, and the text 'Window-size over which to take the maximum' is aligned with the 'pool_size' parameter. The text 'valid' is highlighted in green, with a red arrow pointing to the explanatory text at the bottom right.

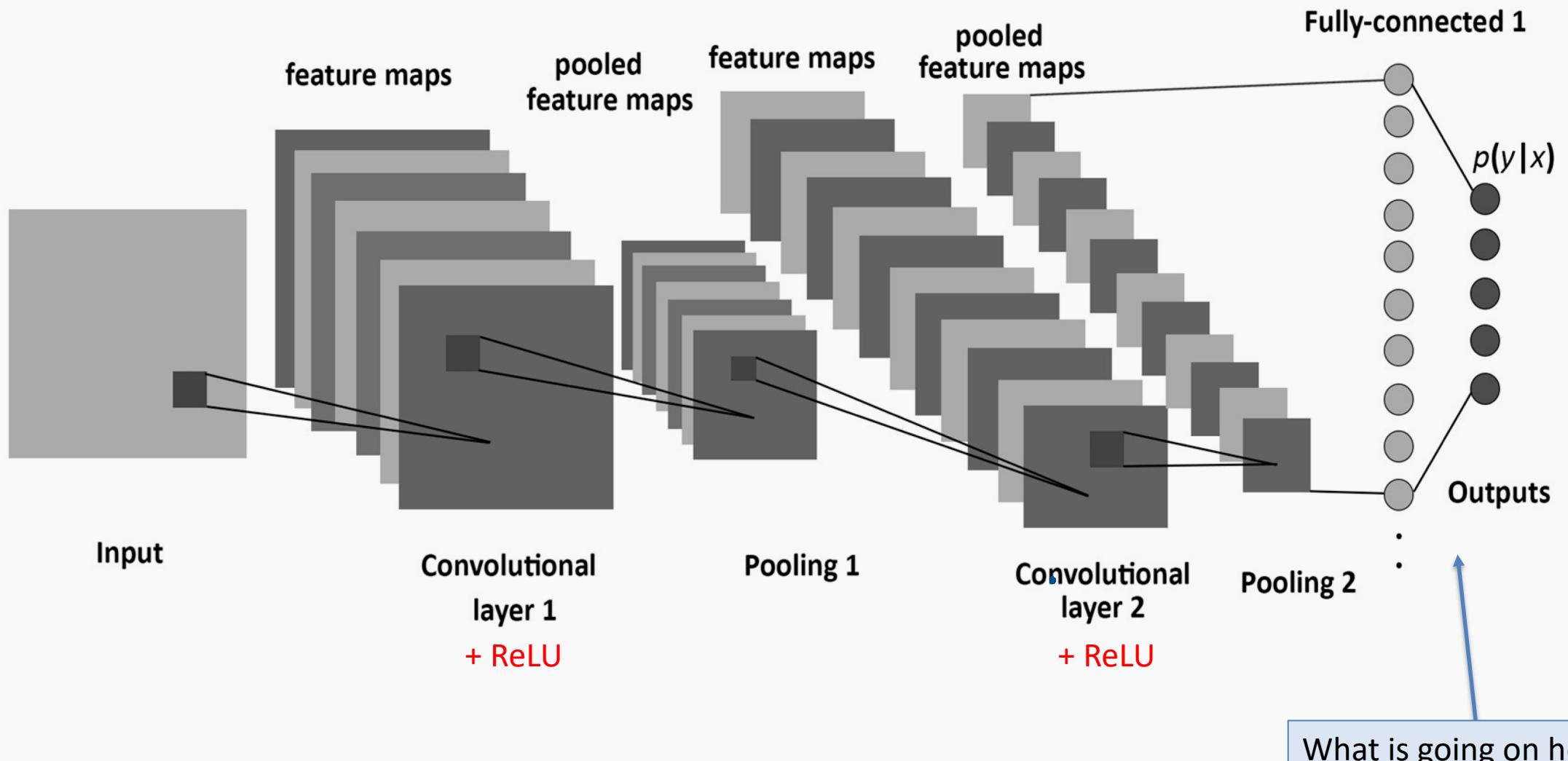


Outline

- Review/Questions
- What are the dimensions of filters, and how do we apply them from one layer to the next?
- Training CNNs
- What Activation Functions do we use?
- What is Pooling?
- **Dense Layers**



A Convolutional Network



What do CNN layers learn?

- Each CNN layer learns features of increasing complexity.

What do CNN layers learn?

- Each CNN layer learns features of increasing complexity.
- The first layers learn **basic feature detection filters**: edges, corners, etc.

What do CNN layers learn?

- Each CNN layer learns features of increasing complexity.
- The first layers learn **basic feature detection filters**: edges, corners, etc.
- The middle layers learn filters that detect **parts of objects**. For faces, they might learn to respond to eyes, noses, etc.

What do CNN layers learn?

- Each CNN layer learns features of increasing complexity.
- The first layers learn **basic feature detection filters**: edges, corners, etc.
- The middle layers learn filters that detect **parts of objects**. For faces, they might learn to respond to eyes, noses, etc.
- The last layers have higher representations: they learn to recognize **full objects**, in different shapes and positions.

WHEN YOU CREATE A CONVOLUTIONAL NEURAL NETWORK

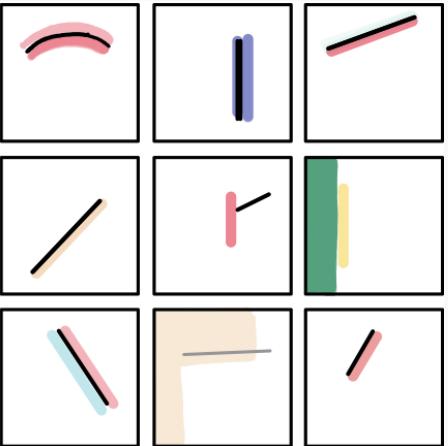


**AND YOU REALIZE THAT YOU DON'T
HAVE A GPU TO TRAIN IT**

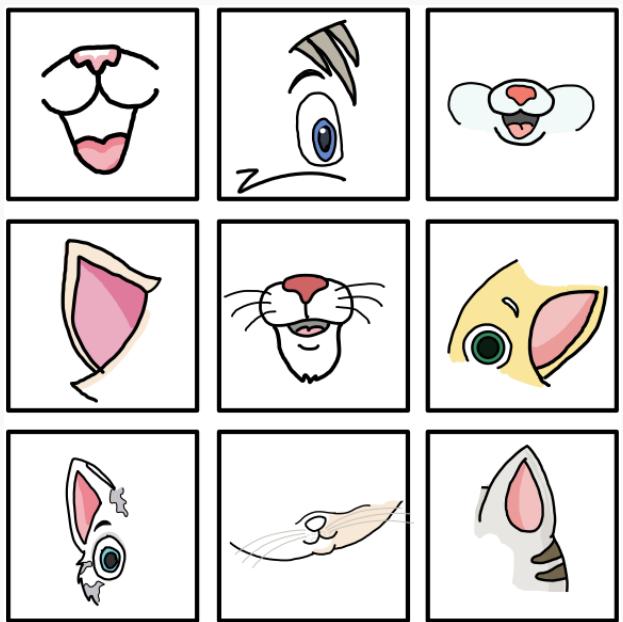
CATS



Layer 1



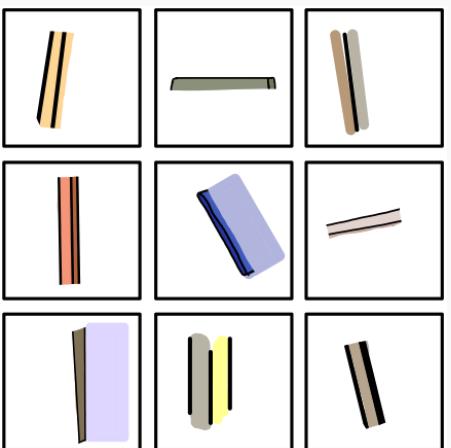
Layer 2



CHAIRS



PROTOPAPAS



Faces



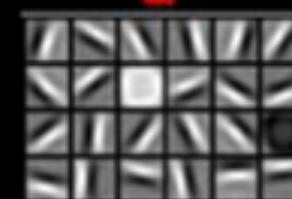
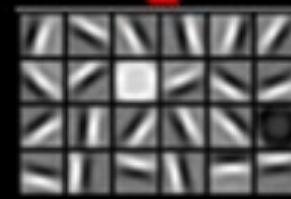
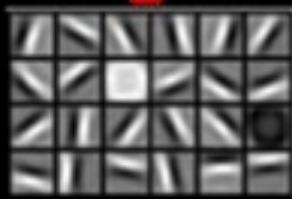
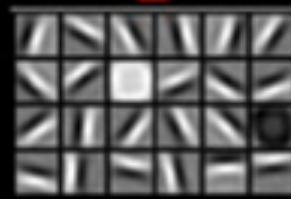
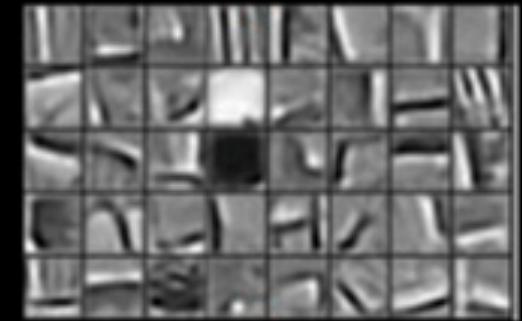
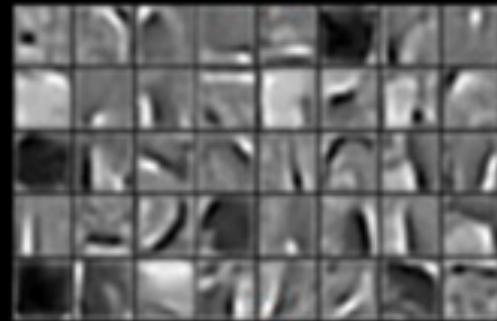
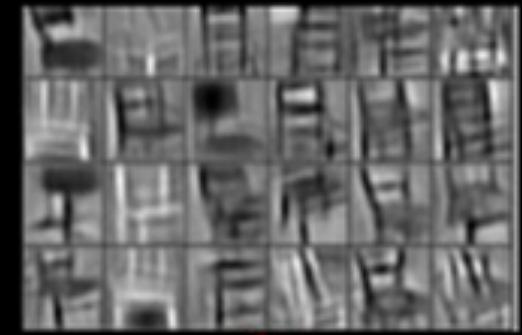
Cars



Elephants



Chairs



Faces



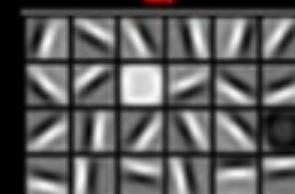
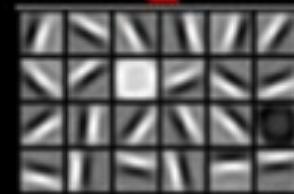
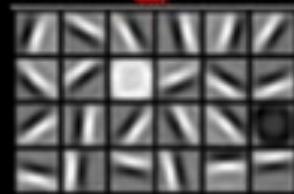
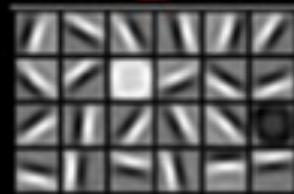
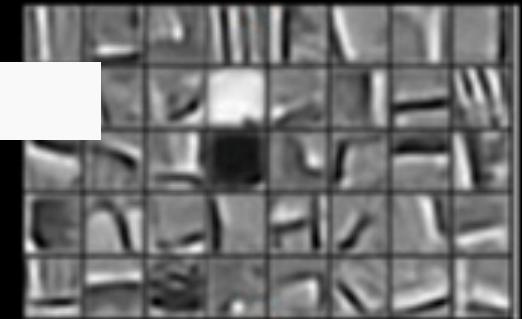
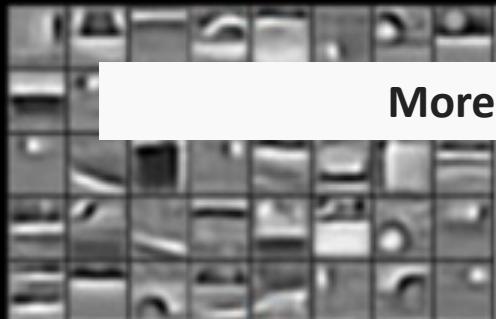
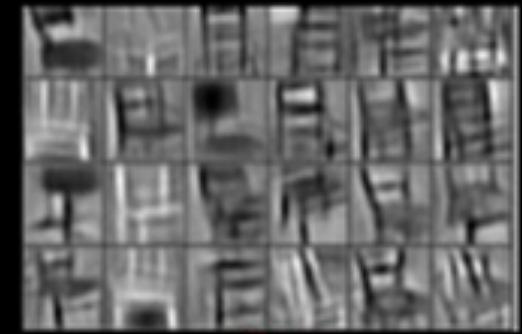
Cars



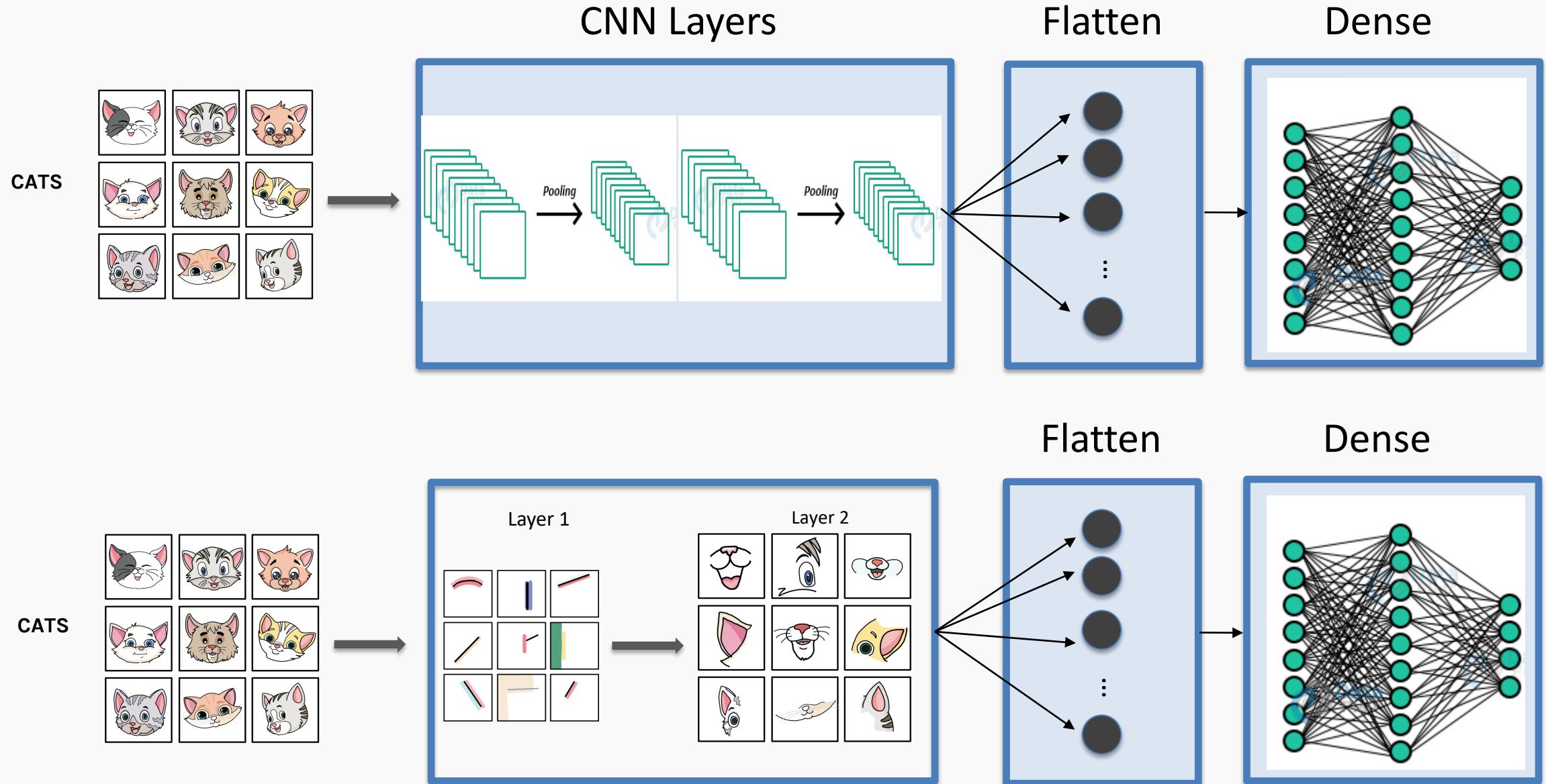
Elephants



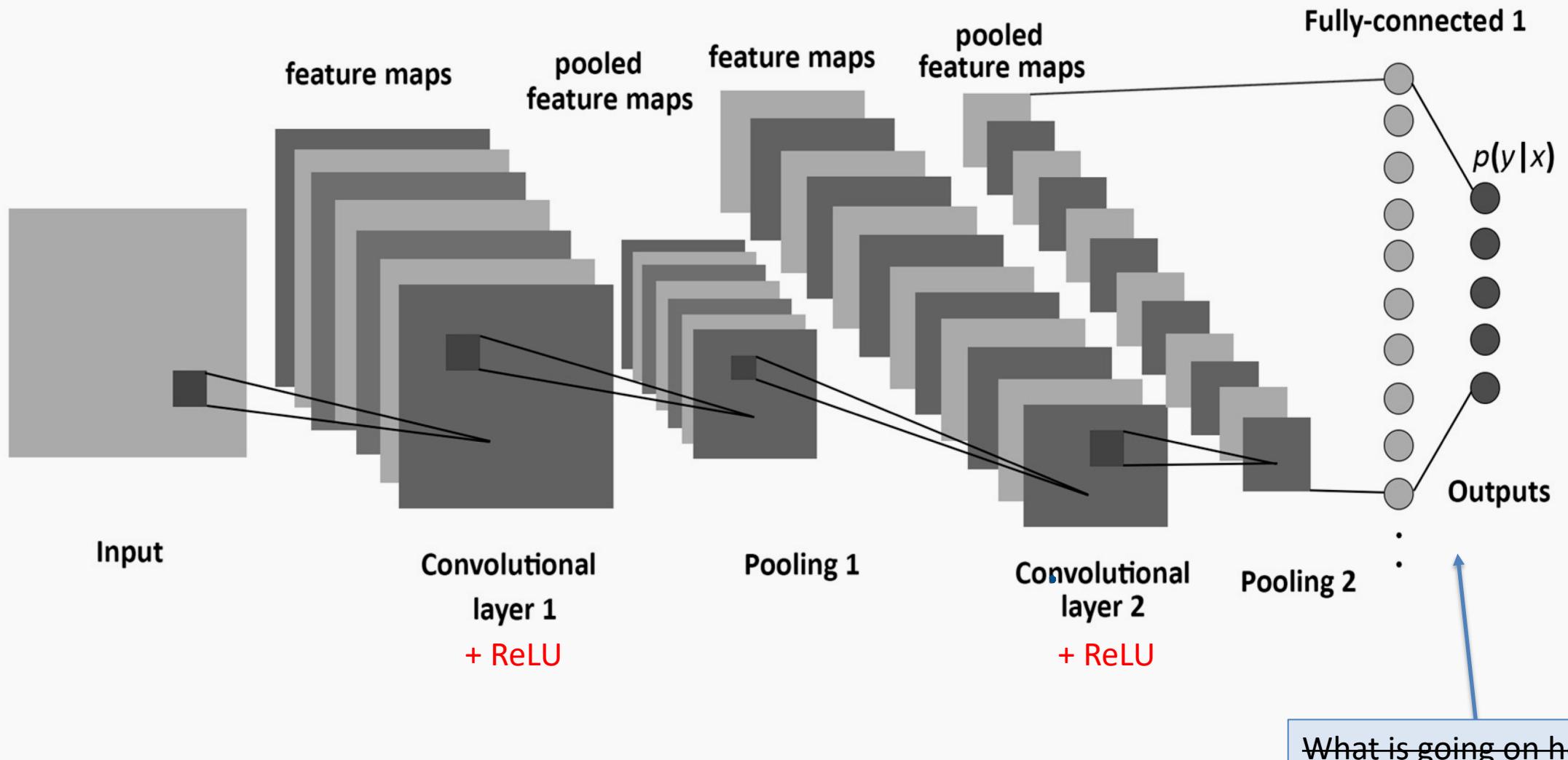
Chairs



More on this in Lecture 8



A Convolutional Network



Building a CNN

A convolutional neural network is built by stacking layers, typically of 3 types:

Convolutional
Layers

Pooling Layers

Fully connected
Layers

Building a CNN

Convolutional Layers

Action

- Apply filters to extract features
- Filters are composed of small kernels, learned
- One bias per filter
- Apply activation function on every value of feature map

Parameters

- Number of filters
- Size of kernels (W and H only, D is defined by input cube)
- Activation function
- Stride
- Padding

I/O

- Input: previous set of feature maps: 3D cuboid
- Output: 3D cuboid, one 2D map per filter

Building a CNN

A convolutional neural network is built by stacking layers, typically of 3 types:

Convolutional
Layers

Pooling Layers

Fully connected
Layers

Building a CNN

Pooling Layers

Action

- Reduce dimensionality
- Extract maximum or average of a region
- Sliding window approach

Parameters

- Stride
- Size of window

I/O

- Input: previous set of feature maps, 3D cuboid
- Output: 3D cuboid, one 2D map per filter, reduced spatial dimensions

Building a CNN

A convolutional neural network is built by stacking layers, typically of 3 types:

Convolutional
Layers

Pooling Layers

Fully connected
Layers

Building a CNN

Fully connected Layers

Action

- Aggregate information from final feature maps
- Generate final classification, regression, segmentation, etc

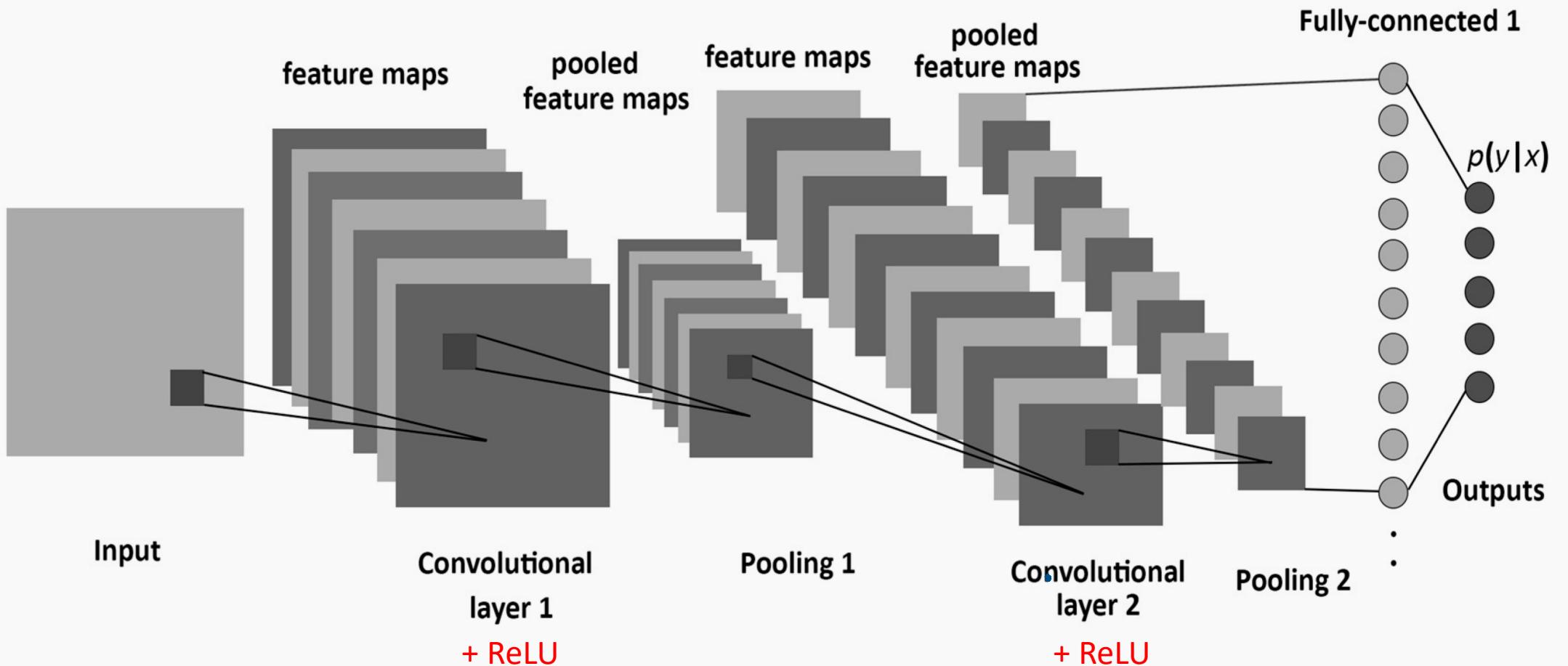
Parameters

- Number of nodes
- Activation function: usually changes depending on role of the layer. If aggregating info, use ReLU. If producing final classification, use Softmax. If regression use linear

I/O

- Input: **FLATTENED** previous set of feature maps
- Output: Probabilities for each class or simply prediction for regression \hat{y}

A Convolutional Network

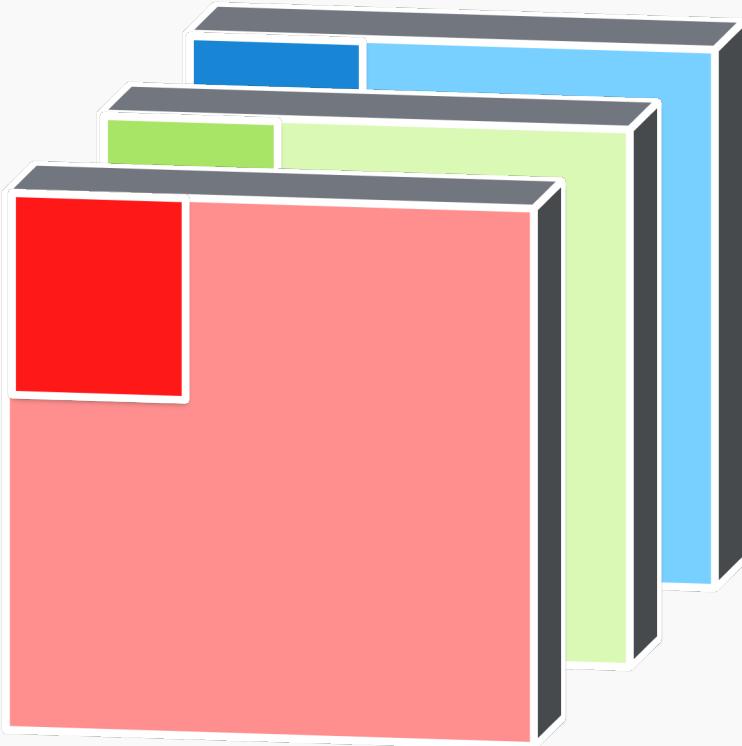


Examples

- Let **C** be a CNN with the following disposition:
 - **Input:** 32x32x3 images
 - **Conv1:** 8 3x3 filters, stride 1, padding=same
 - **Conv2:** 16 5x5 filters, stride 2, padding=same
 - **Flatten layer** (explained in the next few slides)
 - **Dense1:** 512 nodes
 - **Dense2:** 4 nodes
- How many parameters does this network have?

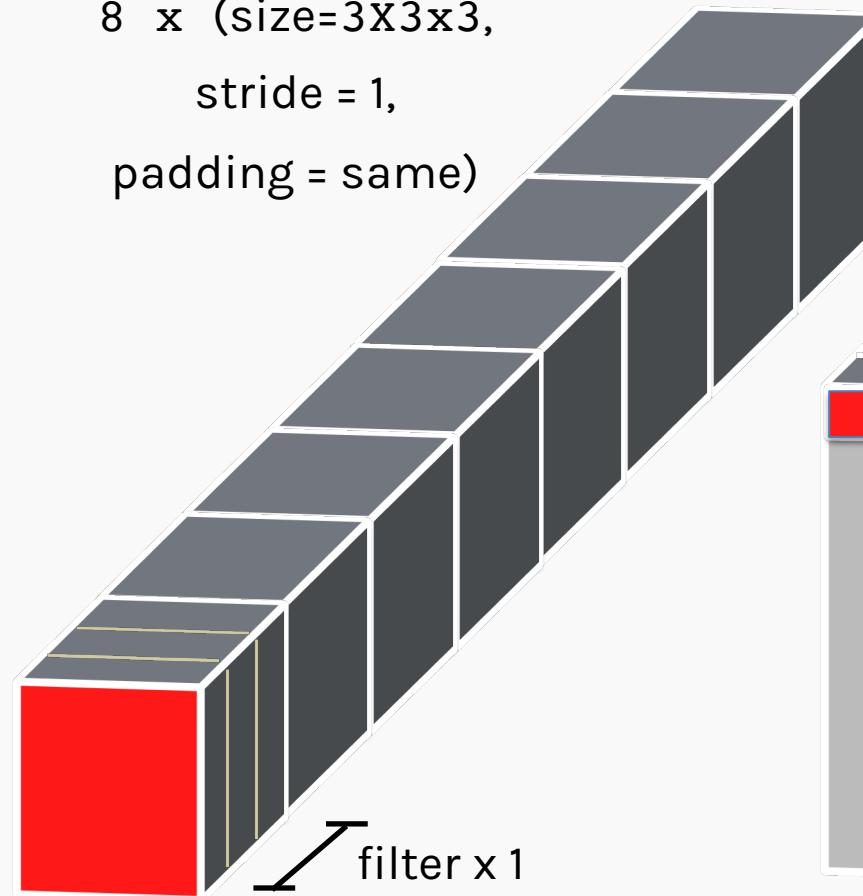
Input

size=32x32
channels=3



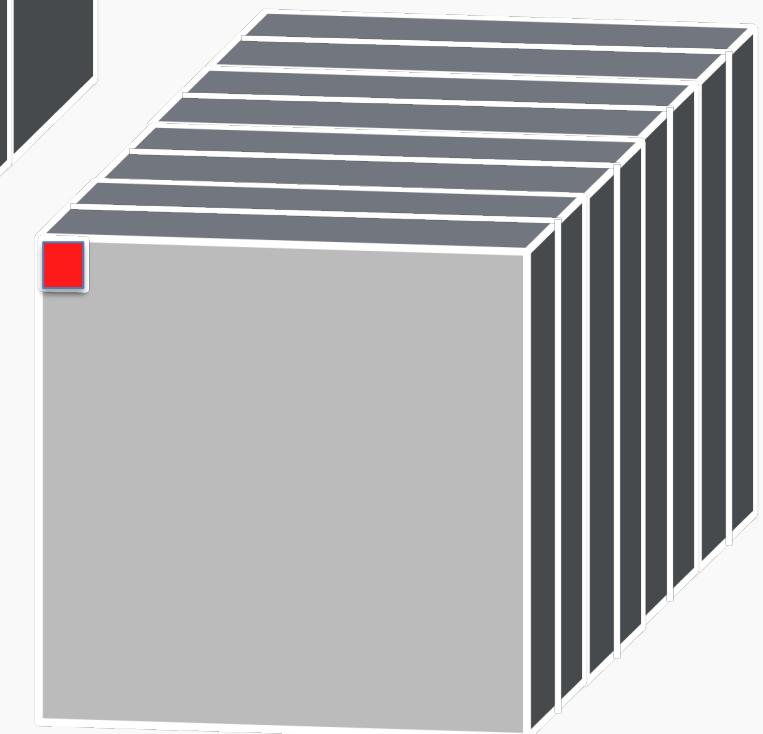
Filter

8 x (size=3x3x3,
stride = 1,
padding = same)



Output

(size=32x32,
channels = 8)



How many parameters does the layer have if I want to use 8 filters?

n_filters x filter_volume + biases = total number of params

$$8 \times (3 \times 3 \times 3) + 8 = 224$$

Input

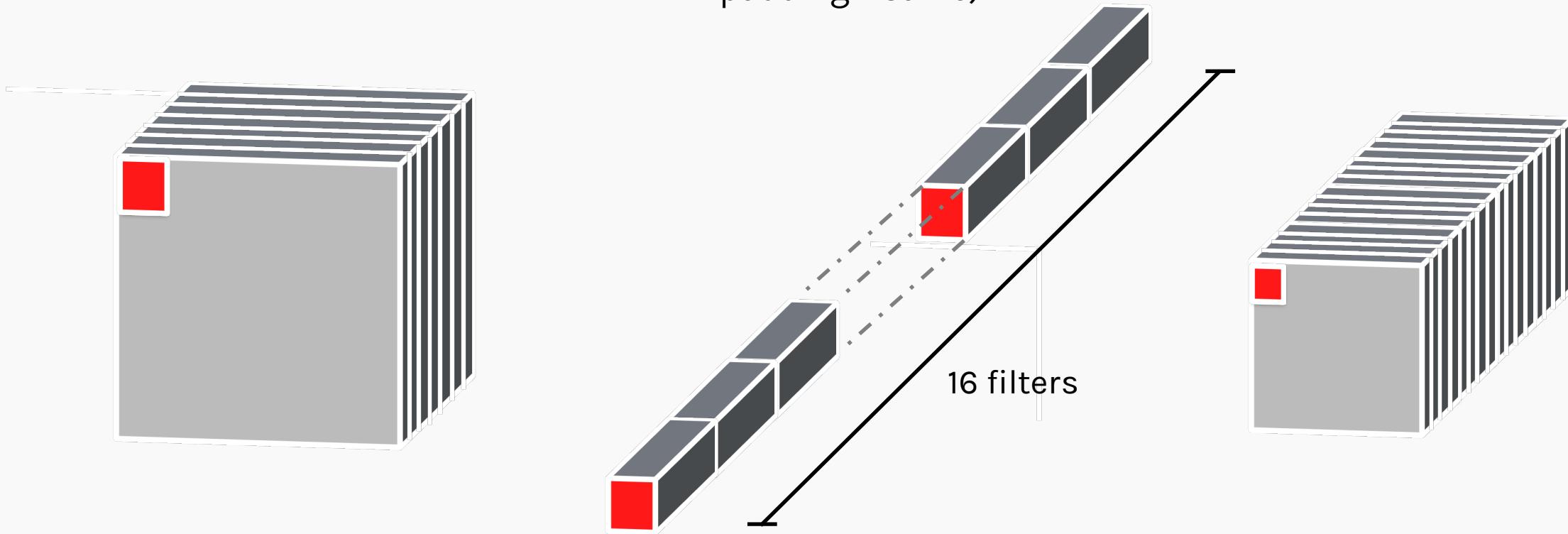
(size=32X32,
channels=8)

Filter

16 x (size=5X5X8,
stride = 2,
padding = same)

Output

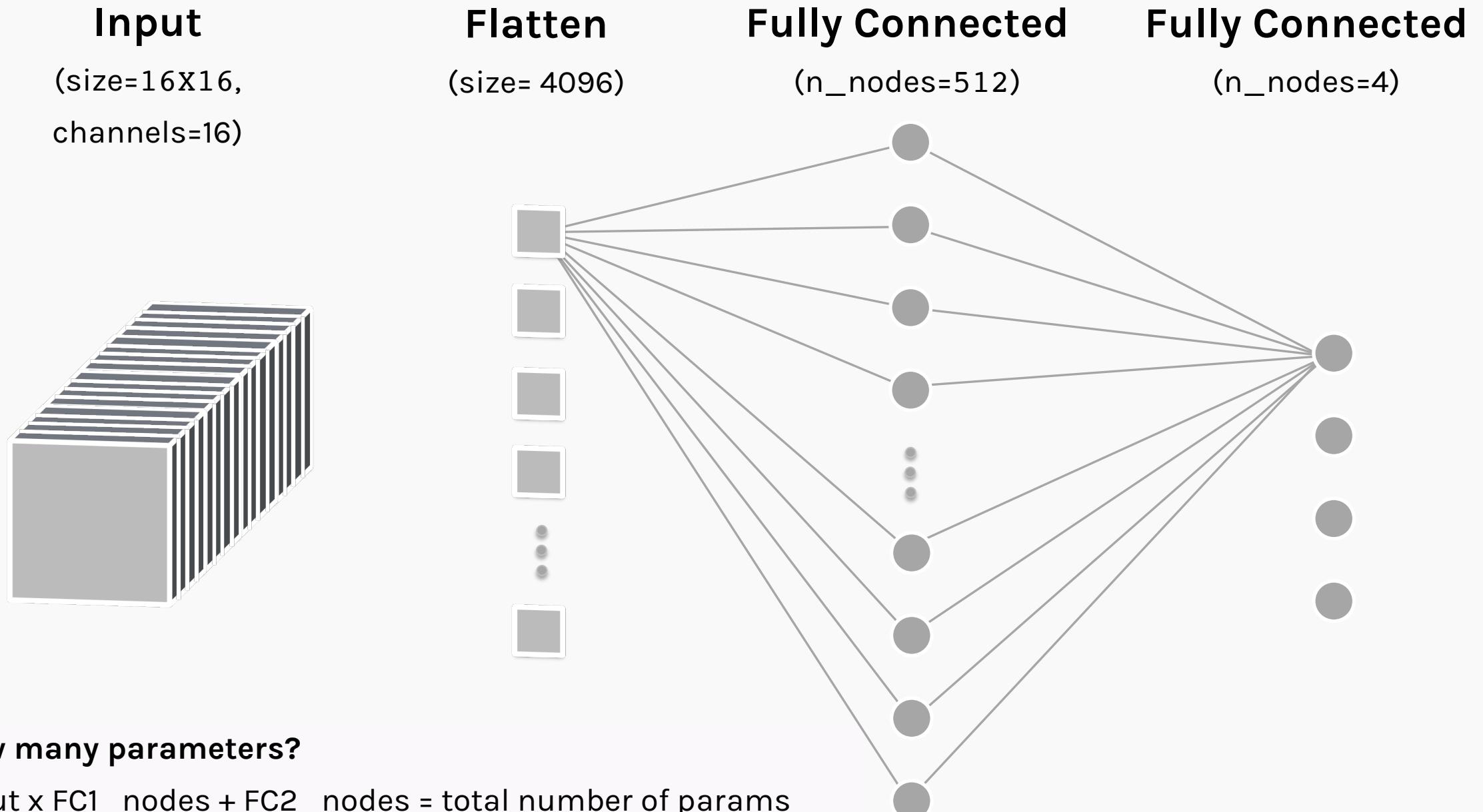
(size=16X16 ,
channels=16)



How many parameters does the layer have if I want to use 16 filters?

n_filters x filter_volume + biases = total number of params

$$16 \times (5 \times 5 \times 8) + 16 = 3216$$



How many parameters?

input x FC1_nodes + FC2_nodes = total number of params

$$(16 \times 16 \times 16) \times 512 + 512 + 512 \times 4 + 4 = 2,099,716$$

Examples

- Let **C** be a CNN with the following disposition:
 - **Input:** 32x32x3 images
 - **Conv1:** 8 3x3 filters, stride 1, padding=same
 - **Conv2:** 16 5x5 filters, stride 2, padding=same
 - **Flatten layer** (explained in the next few slides)
 - **Dense1:** 512 nodes
 - **Dense2:** 4 nodes
- How many parameters does this network have?

$$(8 \times 3 \times 3 \times 3 + 8) + (16 \times 5 \times 5 \times 8 + 16) + (16 \times 16 \times 16 \times 512 + 512) + (512 \times 4 + 4)$$

Conv1

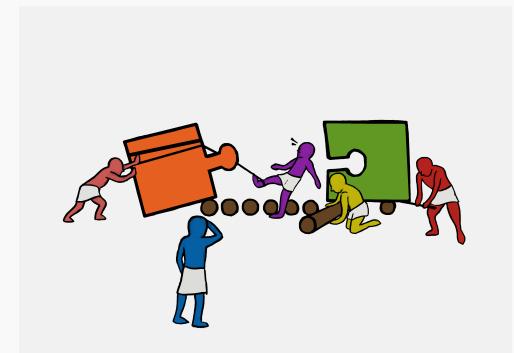
Conv2

Dense1

Dense2

Exercise: Performance measure

- The aim of this exercise is to compare average and max pooling by measuring accuracy and number of parameters for the classification of MNIST digits
- Build three MNIST classification models, one with no pooling, one with average pooling, and one with max pooling, and train them with similar hyper-parameters
- Compute the number of parameters and the accuracy on the test set for each model



Model Type	Test Accuracy	Test Loss	Number of Parameters
Without pooling	0.8884	0.4061	303314
With avg pooling	0.8996	0.3618	29138
With max pooling	0.9116	0.2936	29138

