Carnegie Mellon University

Introduction to Deep Learning for Engineers

Spring 2025, Deep Learning for Engineers
Jan 30, 2025, Sixth Session

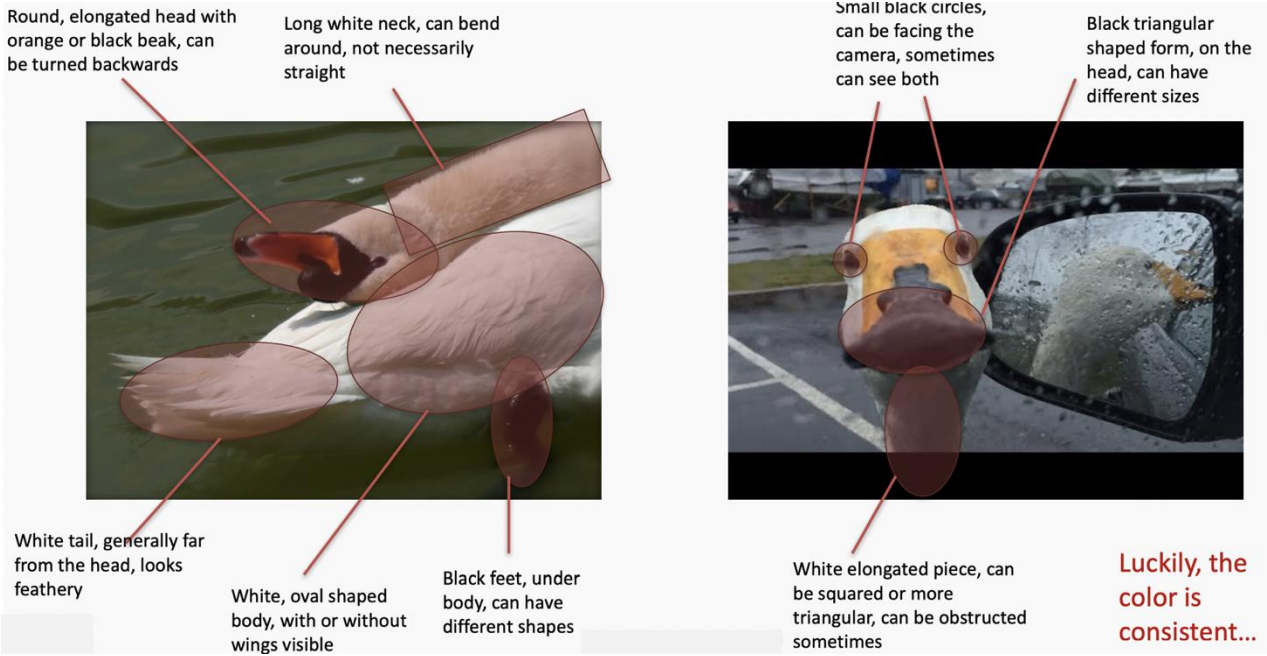
Amir Barati Farimani
Assistant Professor of Mechanical Engineering and Bio-Engineering
Carnegie Mellon University

Story so far:

1. **MLP** (Perceptron, Non-linear separability, Capacity, Depth vs number of Neurons)
2. **Universal Function Approximation**
3. **Empirical Risk Minimization** (Changing Integral to Summation with samples)
4. **Neural Networks Ingredients** (inputs, outputs, Loss functions, architectures)
5. **Optimization (**Gradient Descent**) and Backpropogation** (Chain rules & automatic differentiation)
6. **Design of F (x; w): Regularization** (weight Initialization, Drop out, Data Augmentation, etc.)
7. **Convolutional Neural Networks** (Automatic Feature Learners)

The idea behind convolution

The idea behind convolution



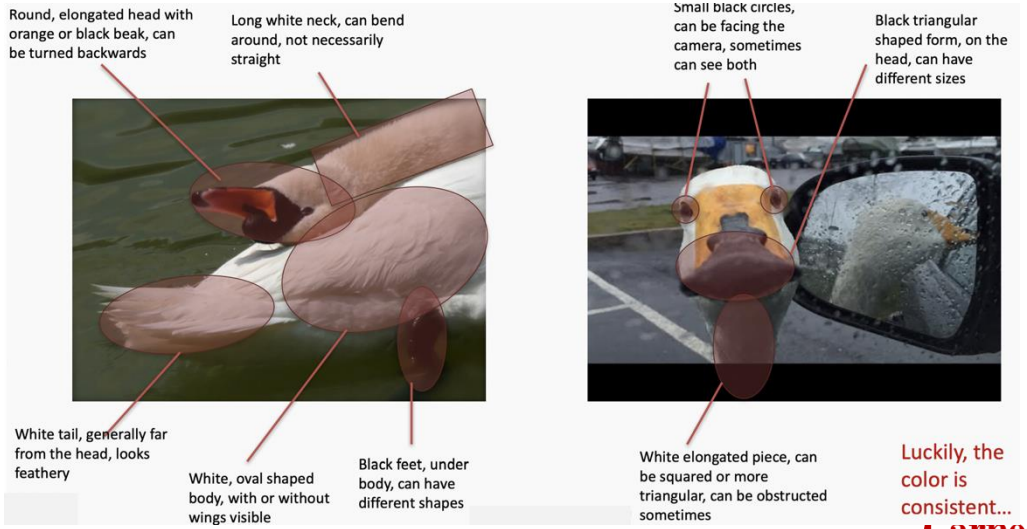
Common Features (Replicated)

Carnegie Mellon University

The idea behind convolution

- What if we learned the features to detect?
- We need a system that can do Representation Learning (or Feature Learning).

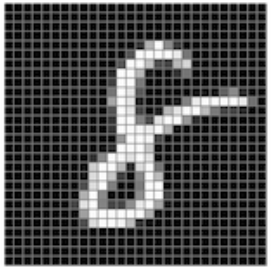
Representation Learning is a technique that allows a system to **automatically find relevant features for a given task**

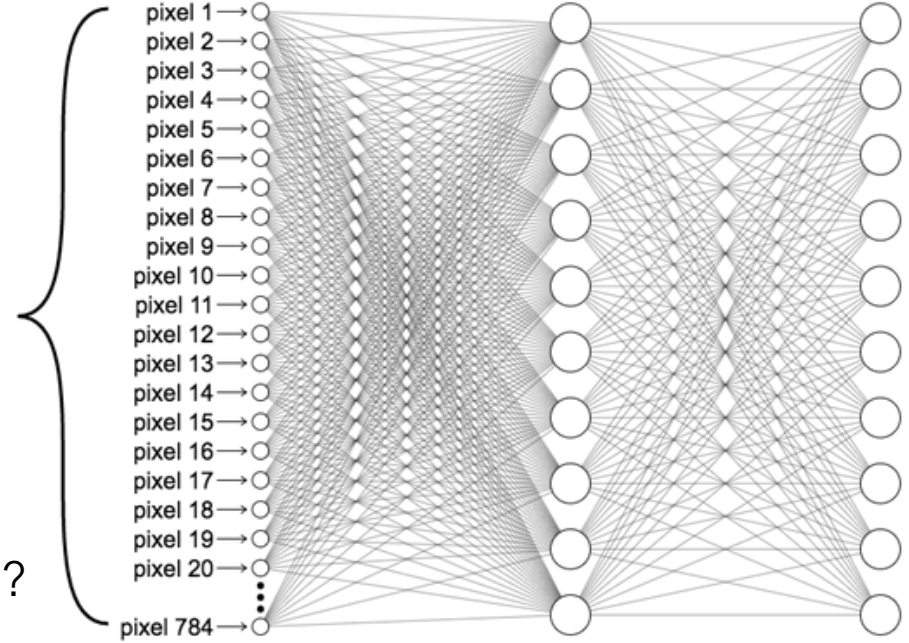


Carnegie Mellon University

The idea behind convolution

The amount of weights rapidly becomes unmanageable for large images. For a 224 x 224 pixel image with 3 color channels there are around 150,000 weights that must be trained! As a result, difficulties arise whilst training and overfitting can occur.





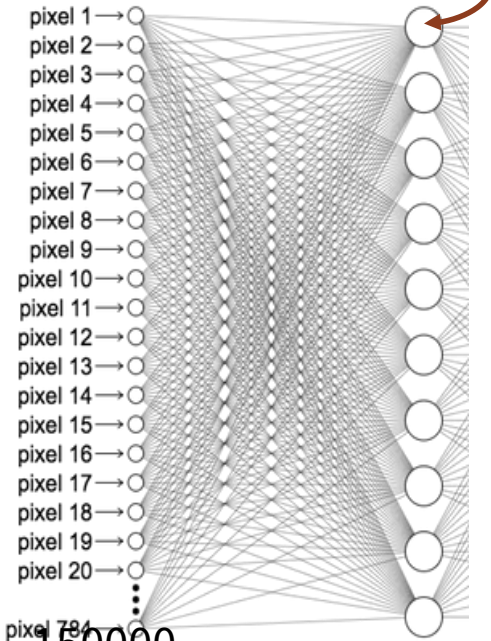
It is OK for 28*28 image

but how about 1024*1024 image?

The idea behind convolution: Scanning

224*224*3

150000 inputs, millions of weights



150000

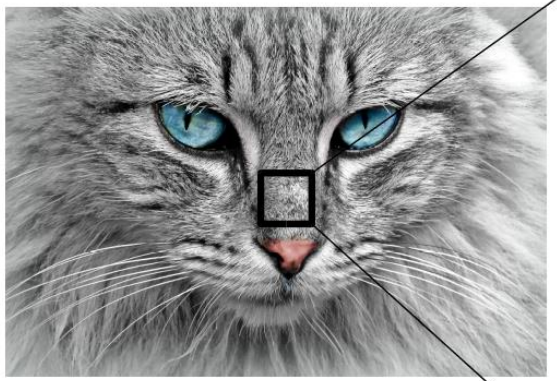
This node has 150000 incoming weights

Question: Can we have a more efficient way of learning the weights for the nodes and reduce the numbers?

flattening will loose the 2D features!

Carnegie Mellon University

What Computer See?



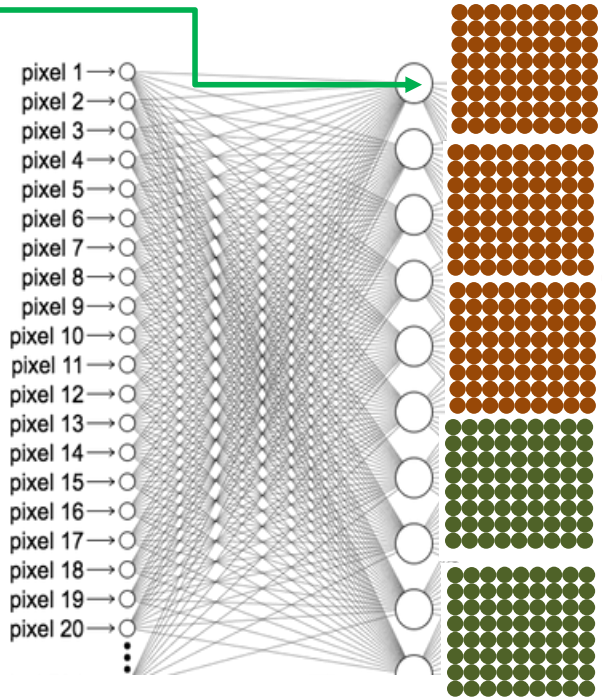
What we see

```
[[105 112 108 111 104  99 106  99  96 103 112 119 104  97  93  87]
 [ 91  98 102 106 104  79  98 103  99 105 123 136 110 105  94  85]
 [ 76  85  90 105 128 105  87  96  95  99 115 112 106 103  99  85]
 [ 99  81  81  93 120 131 127 100  95  98 102  99  96  93 101  94]
 [106  91  61  64  69  91  88  85 101 107 109  98  75  84  96  95]
 [114 108  85  55  55  69  64  54  64  87 112 129  98  74  84  91]
 [133 137 147 103  65  81  80  65  52  54  74  84 102  93  85  82]
 [128 137 144 140 109  95  86  70  62  65  63  63  60  73  86 101]
 [125 133 148 137 119 121 117  94  65  79  80  65  54  64  72  98]
 [127 125 131 147 133 127 126 131 111  96  89  75  61  64  72  84]
 [115 114 109 123 150 148 131 118 113 109 100  92  74  65  72  78]
 [ 89  93  90  97 108 147 131 118 113 114 113 109 106  95  77  80]
 [ 63  77  86  81  77  79 102 123 117 115 117 125 125 130 115  87]
 [ 62  65  82  89  78  71  80 101 124 126 119 101 107 114 131 119]
 [ 63  65  75  88  89  71  62  81 120 138 135 105  81  98 110 118]
 [ 87  65  71  87 106  95  69  45  76 130 126 107  92  94 105 112]
 [118  97  82  86 117 123 116  66  41  51  95  93  89  95 102 107]
 [164 146 112  80  82 120 124 104  76  48  45  66  88 101 102 109]
 [157 170 157 120  93  86 114 132 112  97  69  55  70  82  99  94]
 [130 128 134 161 139 100 109 118 121 134 114  87  65  53  69  86]
 [128 112  96 117 150 144 120 115 104 107 102  93  87  81  72  79]
 [123 107  96  86  83 112 153 149 122 109 104  75  80 107 112  99]
 [122 121 102  80  82  86  94 117 145 148 153 102  58  78  92 107]
 [122 164 148 103  71  56  78  83  93 103 119 139 102  61  69  84]]
```

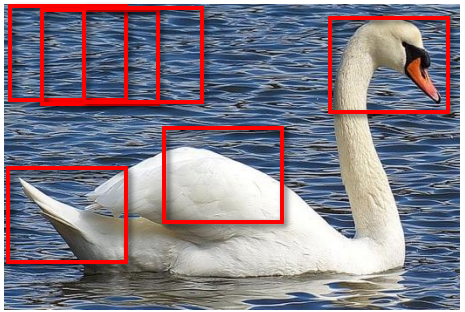
What computer sees

Carnegie Mellon University

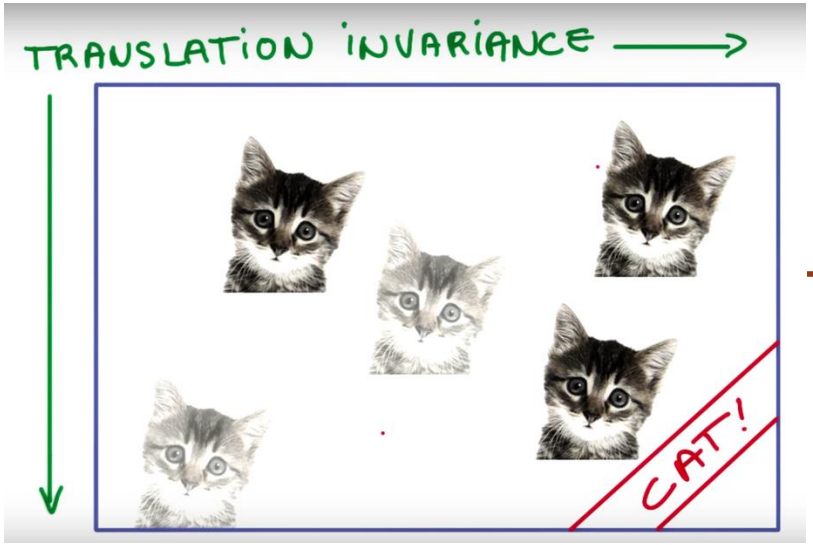
Let's see a node after input

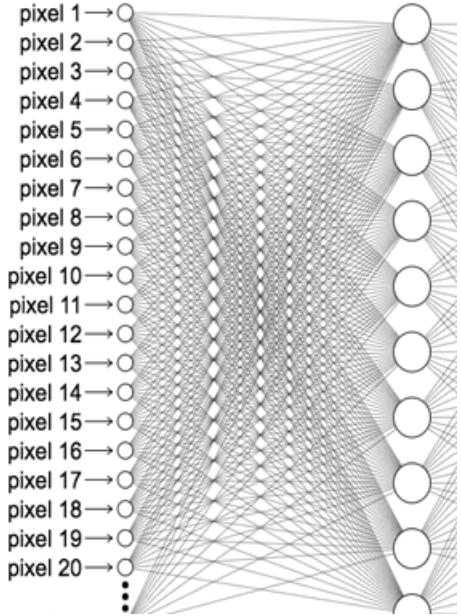


Instead

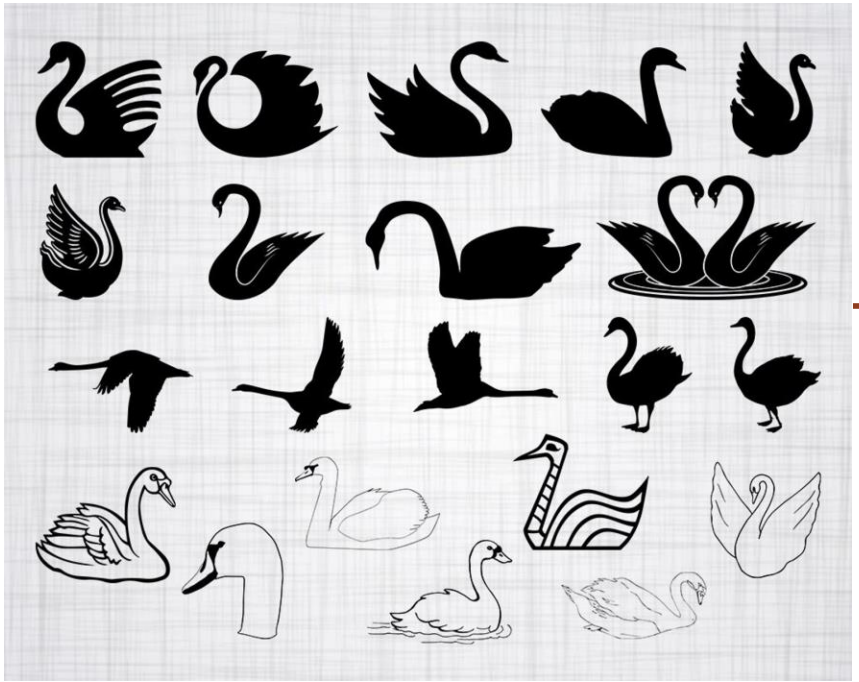


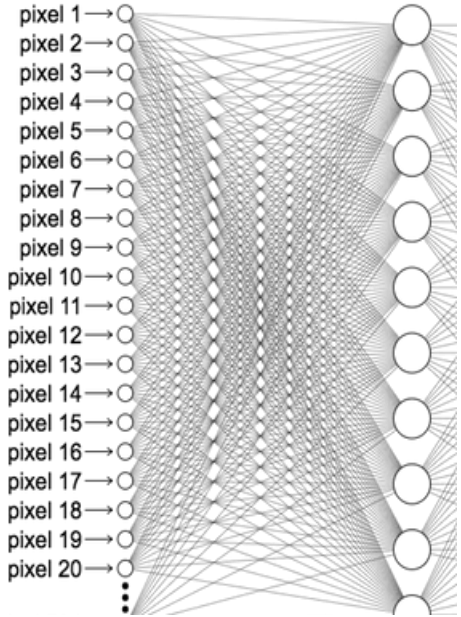
The idea behind convolution:
Translation challenge:



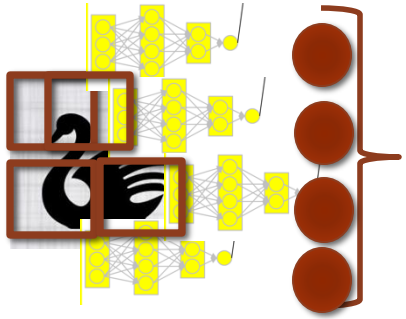


The idea behind convolution:
Rotation challenge:

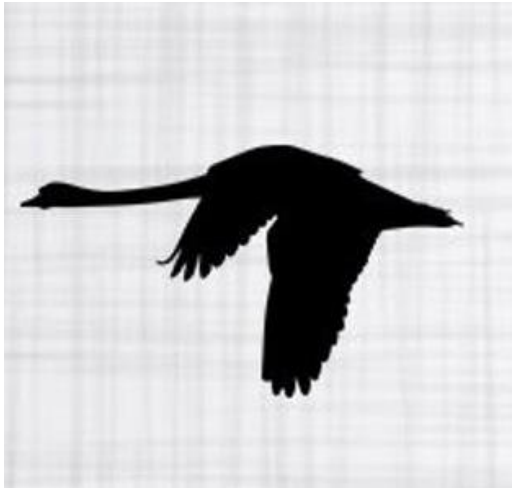




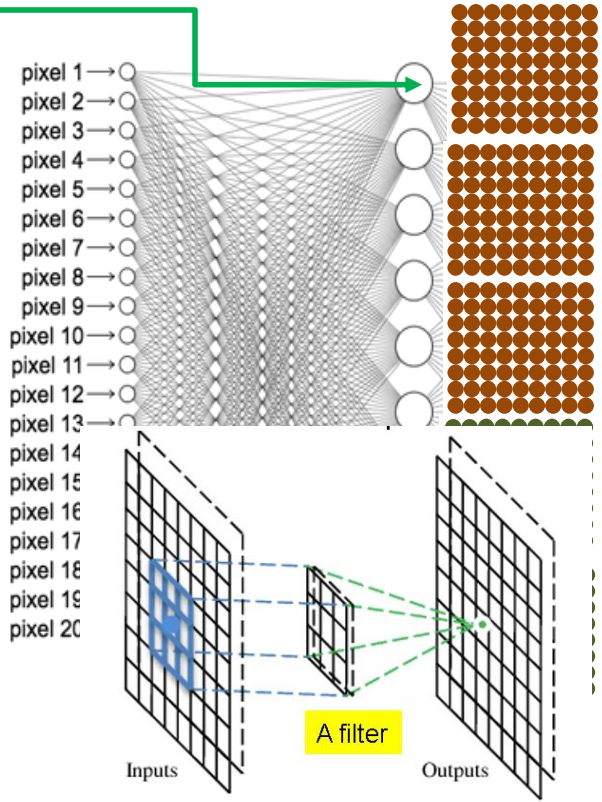
The idea behind convolution: Scanning



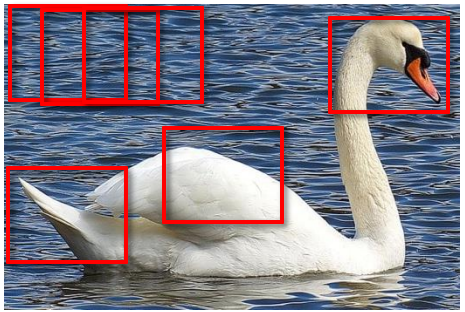
Softmax → Is there a Swan?



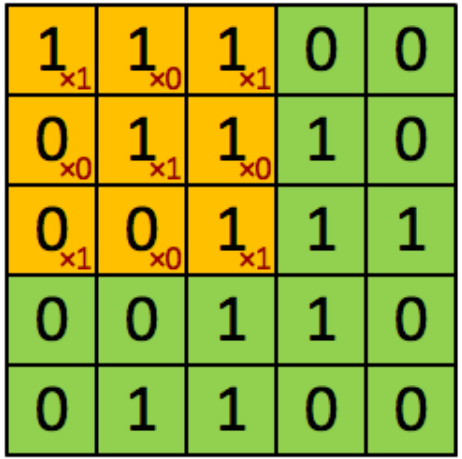
How can we do this?



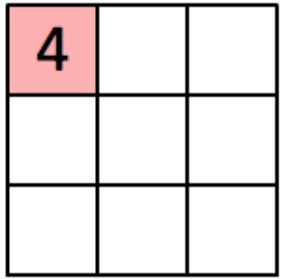
Instead



Scanner



Image



Convolved Feature

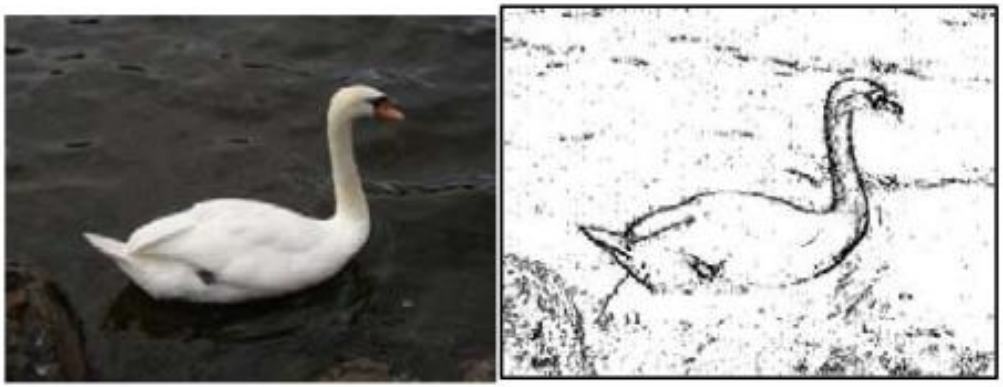
So what do we get as the output of convolved feature?

1_{x1}	1_{x0}	1_{x1}	0	0
0_{x0}	1_{x1}	1_{x0}	1	0
0_{x1}	0_{x0}	1_{x1}	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved Feature



Definition of Kernel/Filter/Scanner

Edge detection

Kernel

$$* \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} =$$

Sharpen

$$* \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} =$$

The idea of Filter

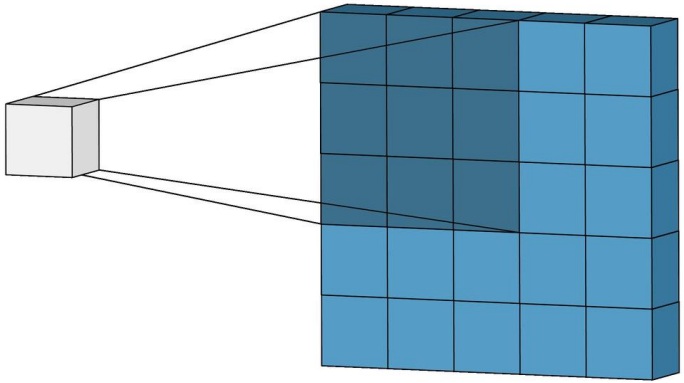
Convolution of an image with different filters can perform operations such as edge detection, blur and sharpen by applying filters.

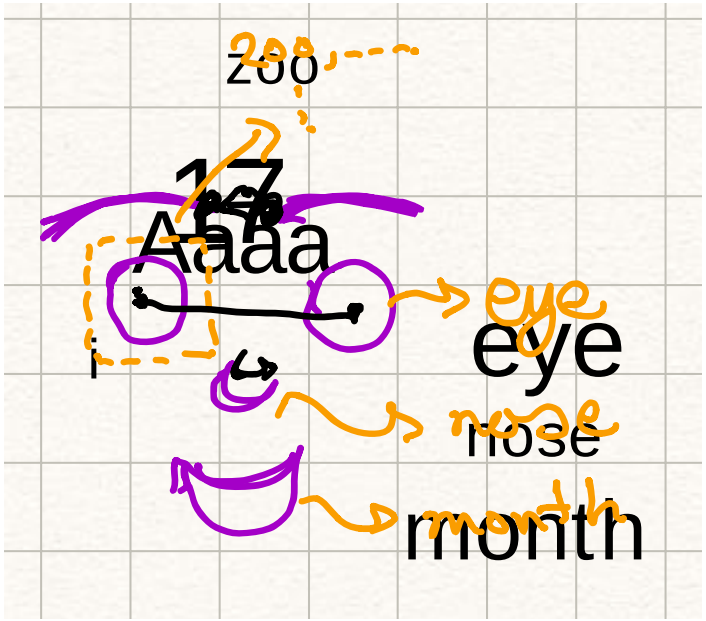
The example shows various convolution image after applying different types of filters (Kernels).

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

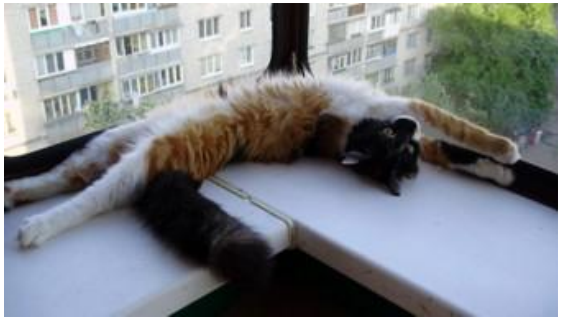
Carnegie Mellon University

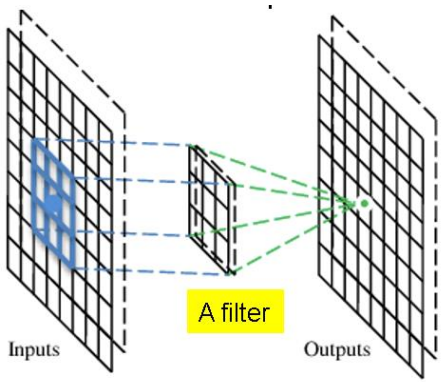
The idea of Filter



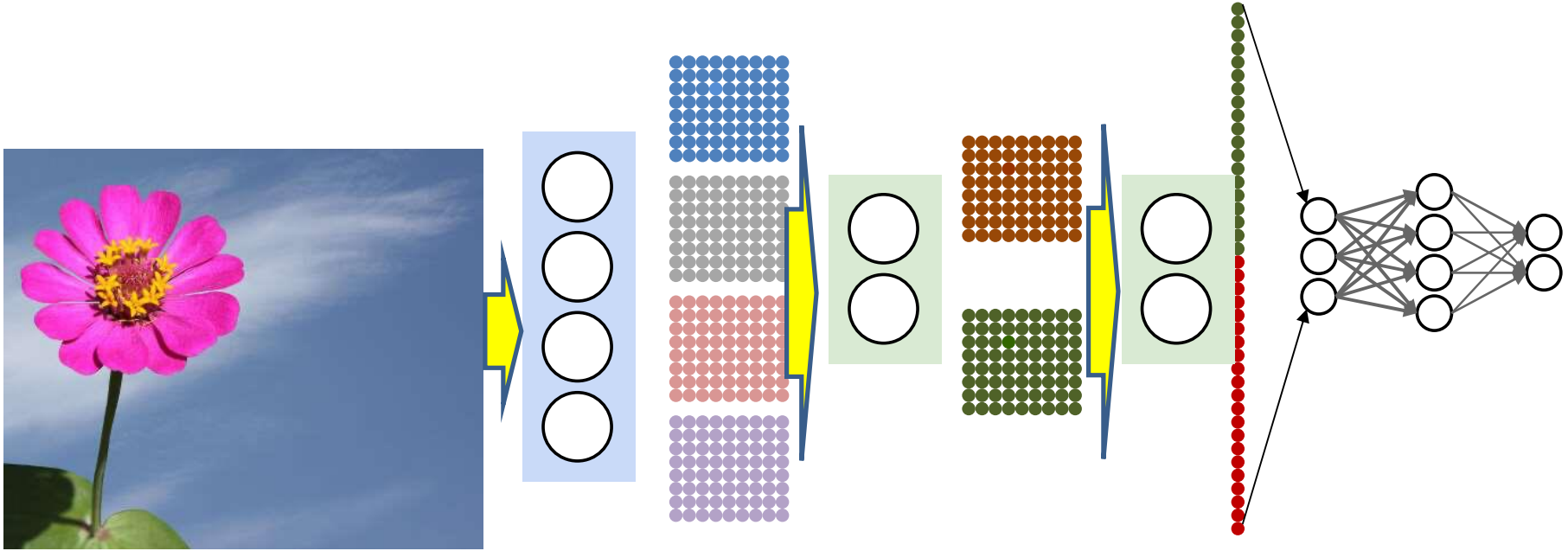


Translation-shift invariance





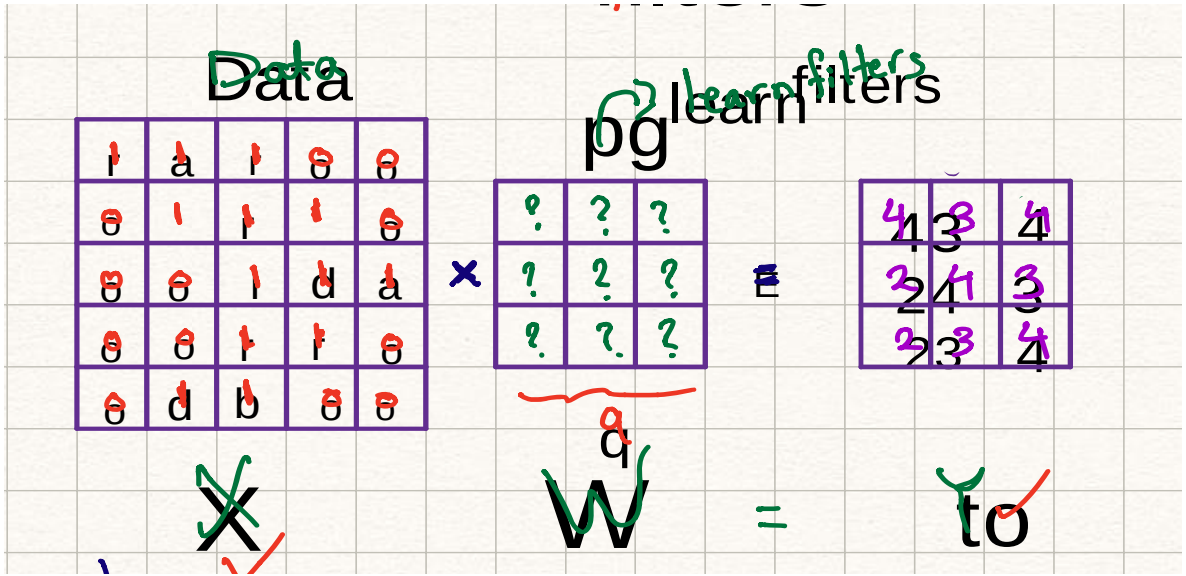
How do we notify what features we want?



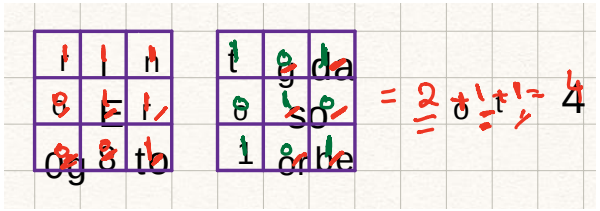
How Filters learn?

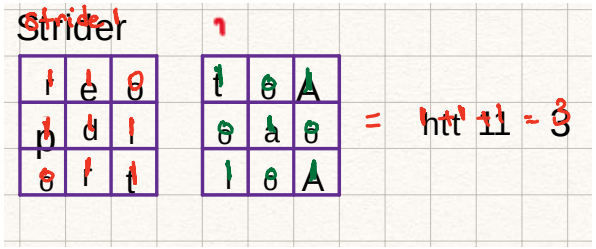
Input data

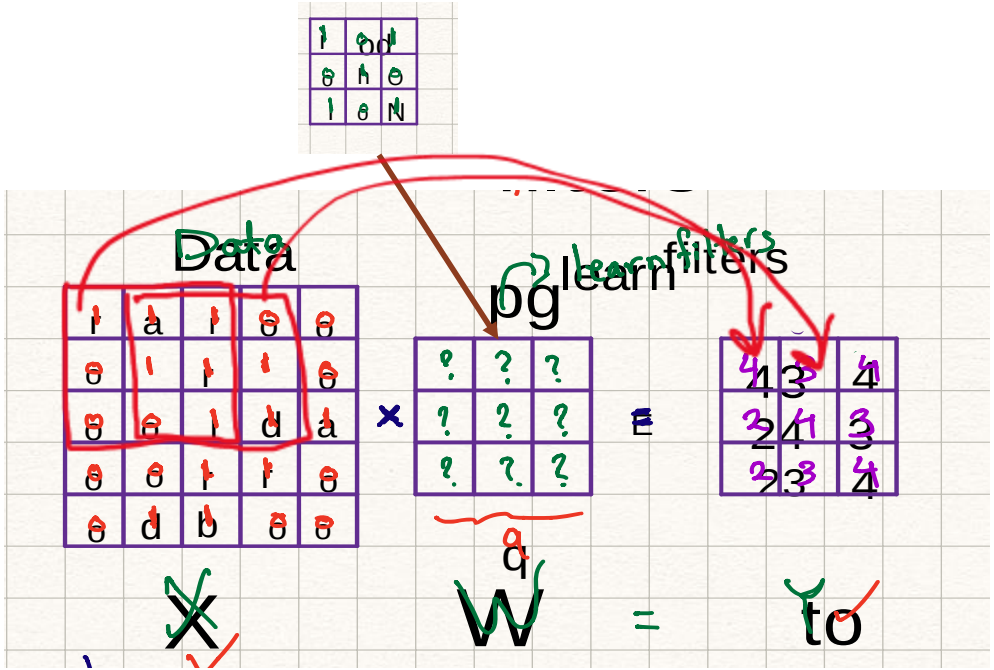
Desired pattern



How Filters work?







Some facts so far

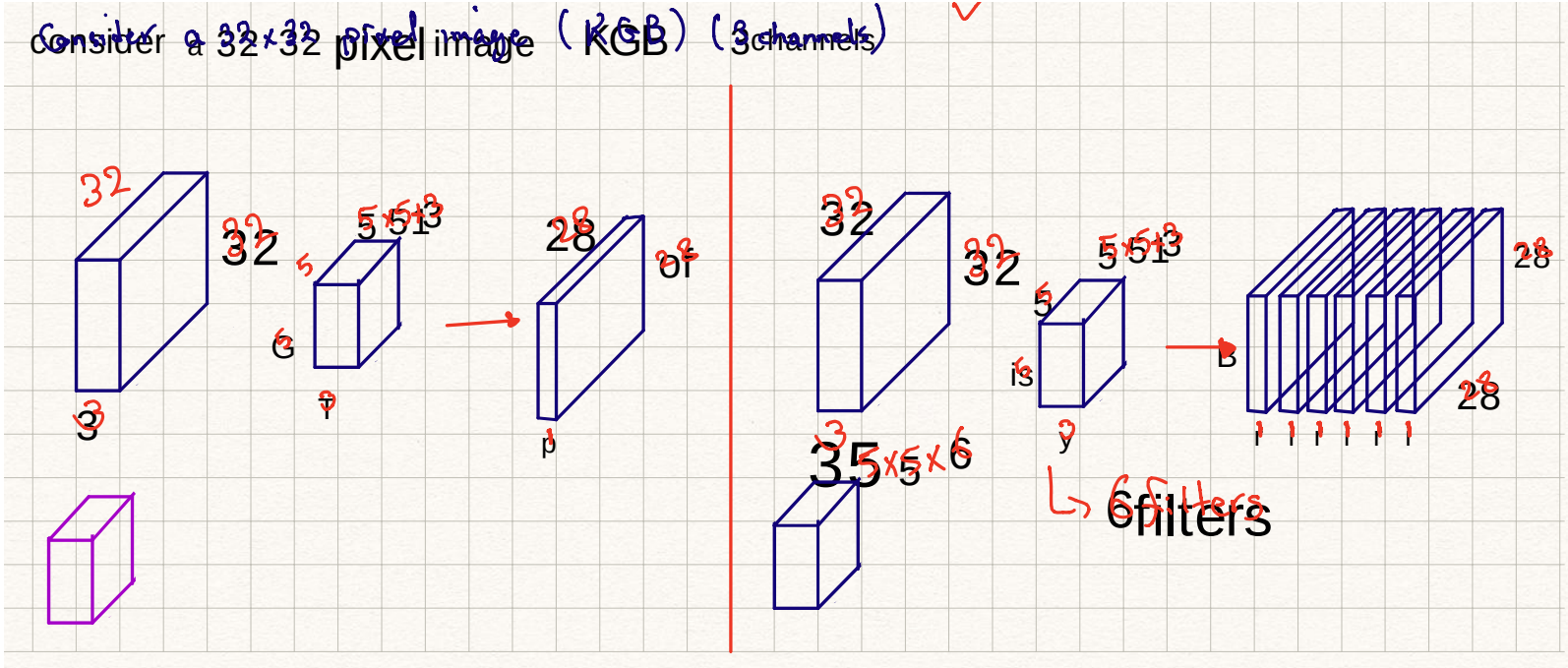
•Position-invariant pattern classification can be performed by scanning
– 1-D scanning forsound
– 2-D scanning for images
– 3-D and higher-dimensional scans for higher dimensional data

•Scanning is equivalent to composing a large network with repeating subnets

•The large network has shared subnets

•Learning in scanned networks: Backpropagation rules must be modified to combine gradients from parameters that share the same value
•– The principle applies in general for networks with shared parameters

How it works for Images?



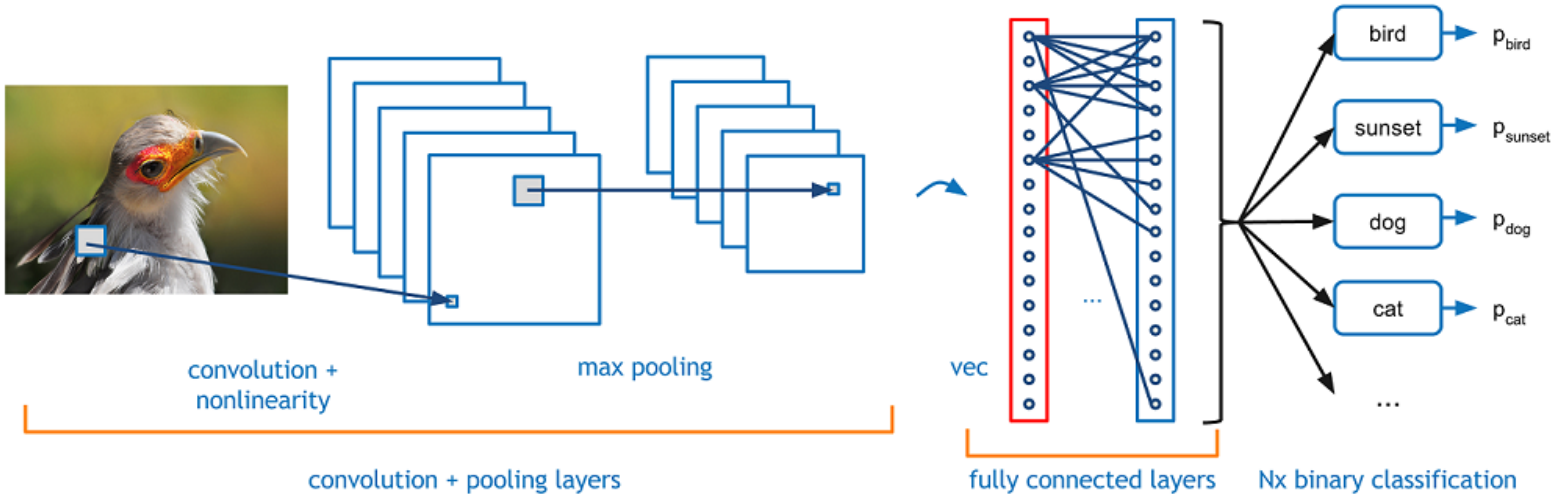
Facts about CNN

- **Automatic Feature Detection via Filters**
  - 1-D scanning for sound
  - 2-D scanning for images
  - 3-D and higher-dimensional scans for higher dimensional data

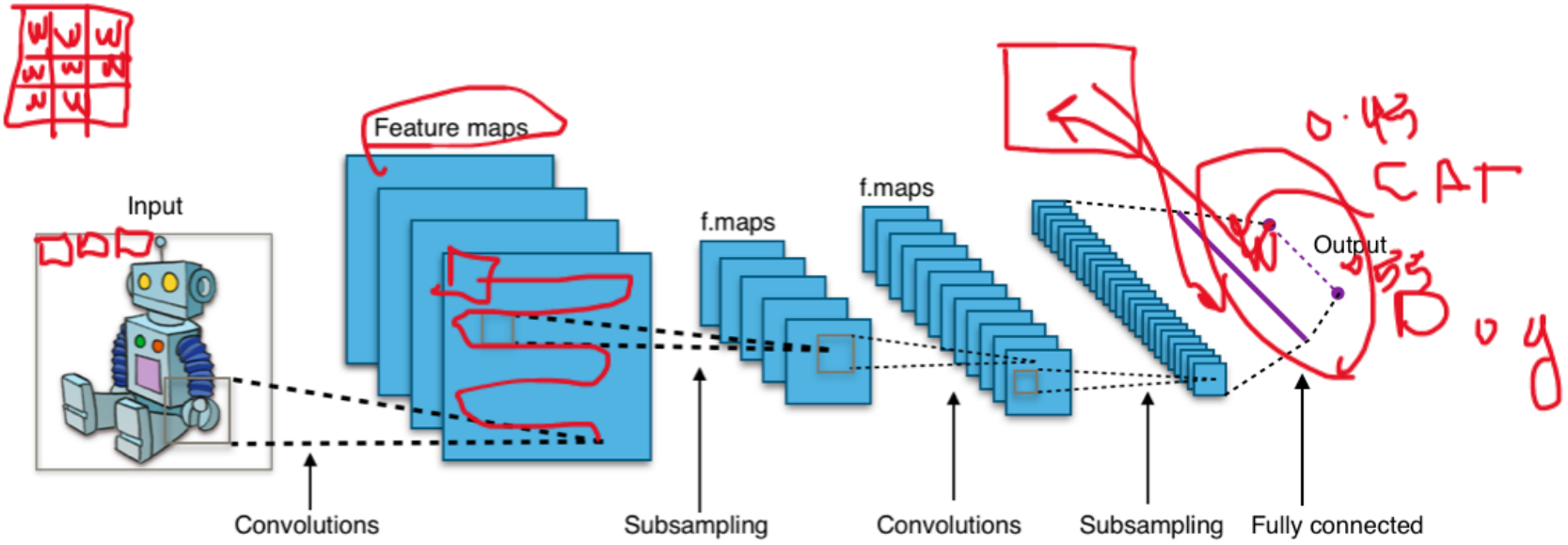
- **The Filters can solve translation-shift invariant problem**

- **Parameter (Weight) sharing can significantly reduce the size of Network**

CNN Overall Architecture

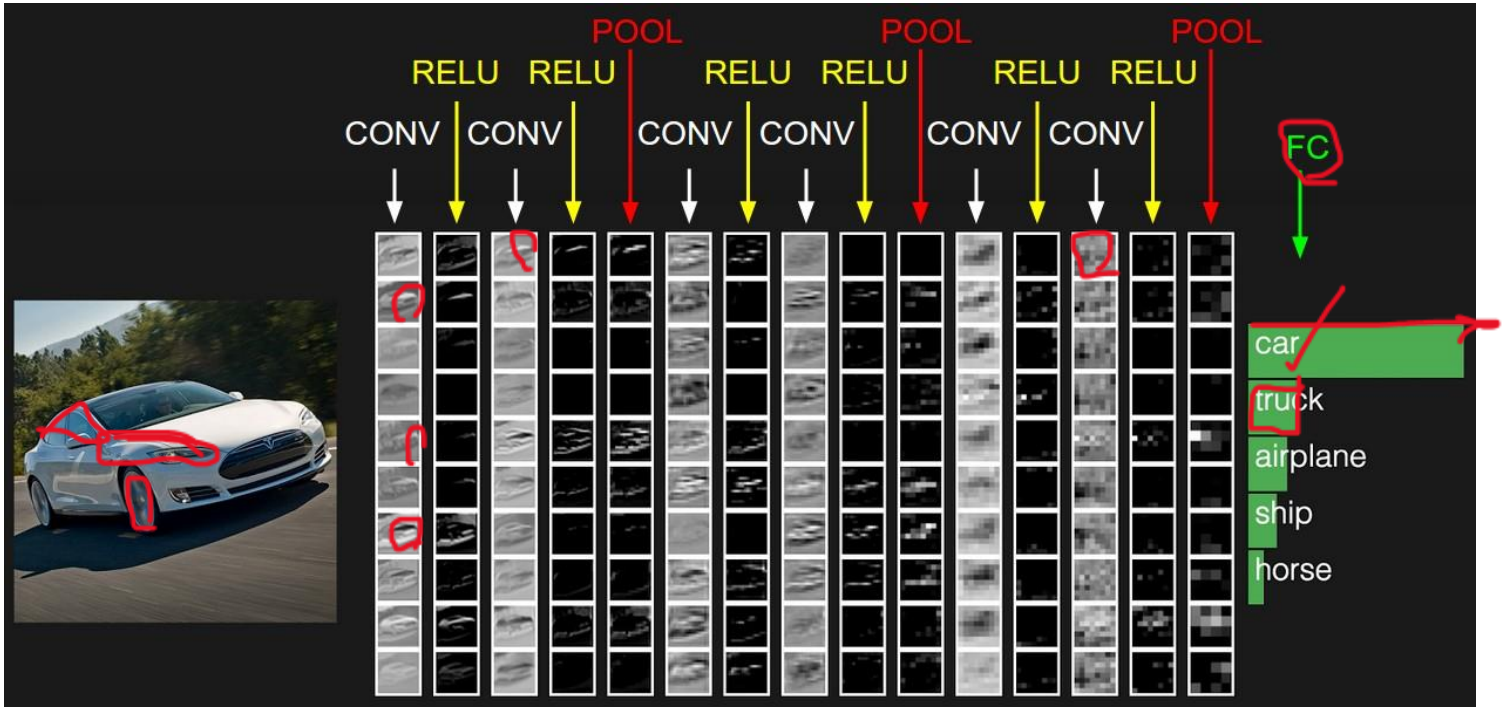


CNN Overall Architecture

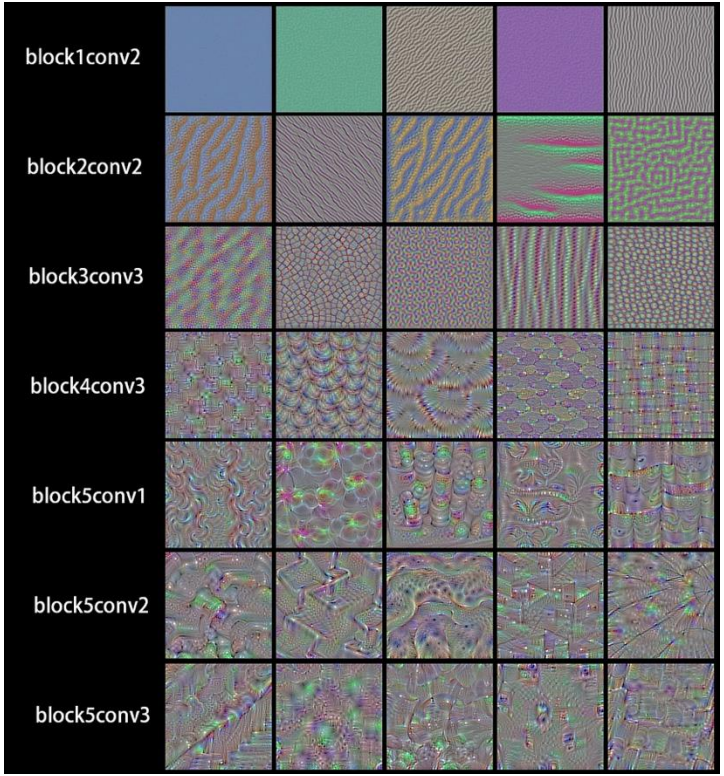


ReLU

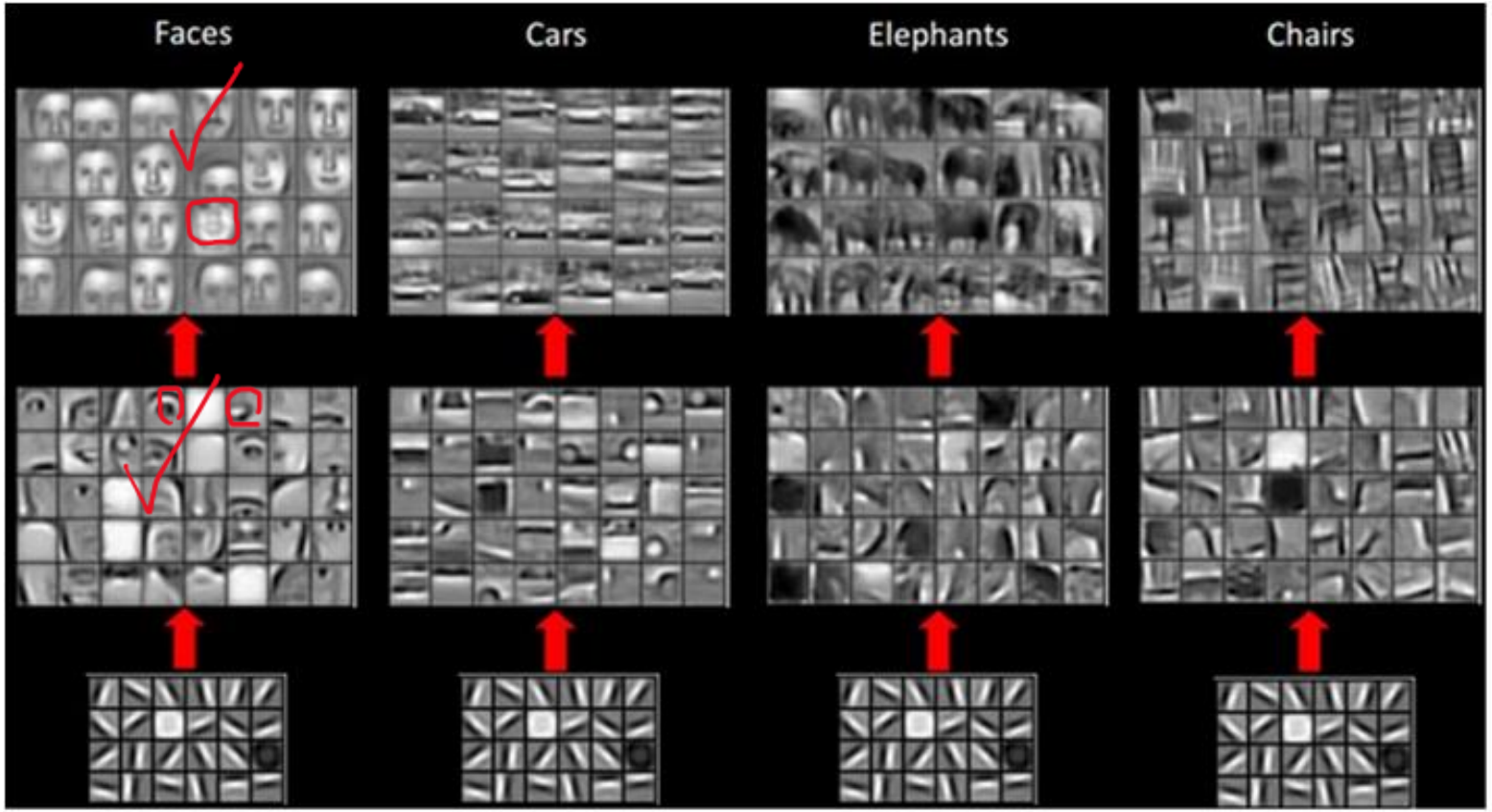
Feature Map (Convolution Layer)



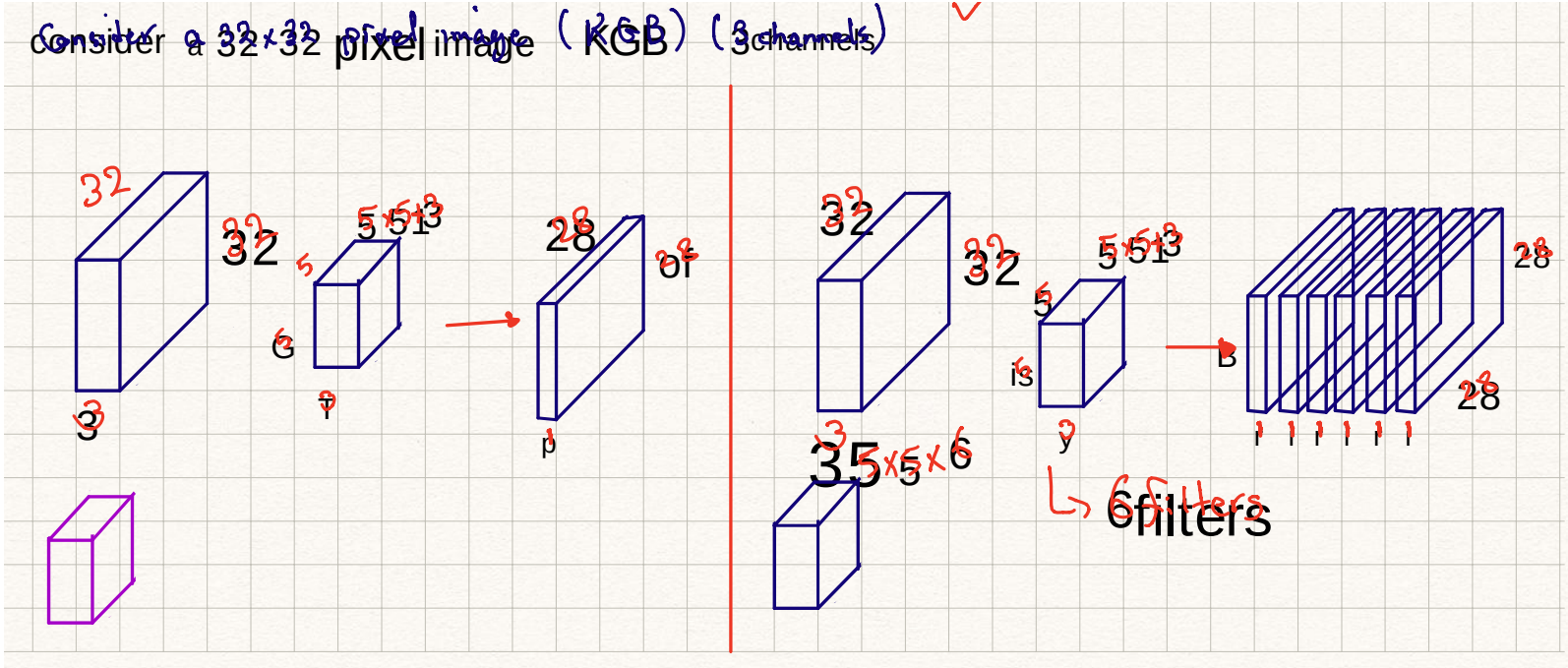
Feature Map


block1conv2
block2conv2
block3conv3
block4conv3
block5conv1
block5conv2
block5conv3

Feature Map



How it works for Images?



CNNs are Automatic Feature Detectors

Sharing Weights

Feature Detection

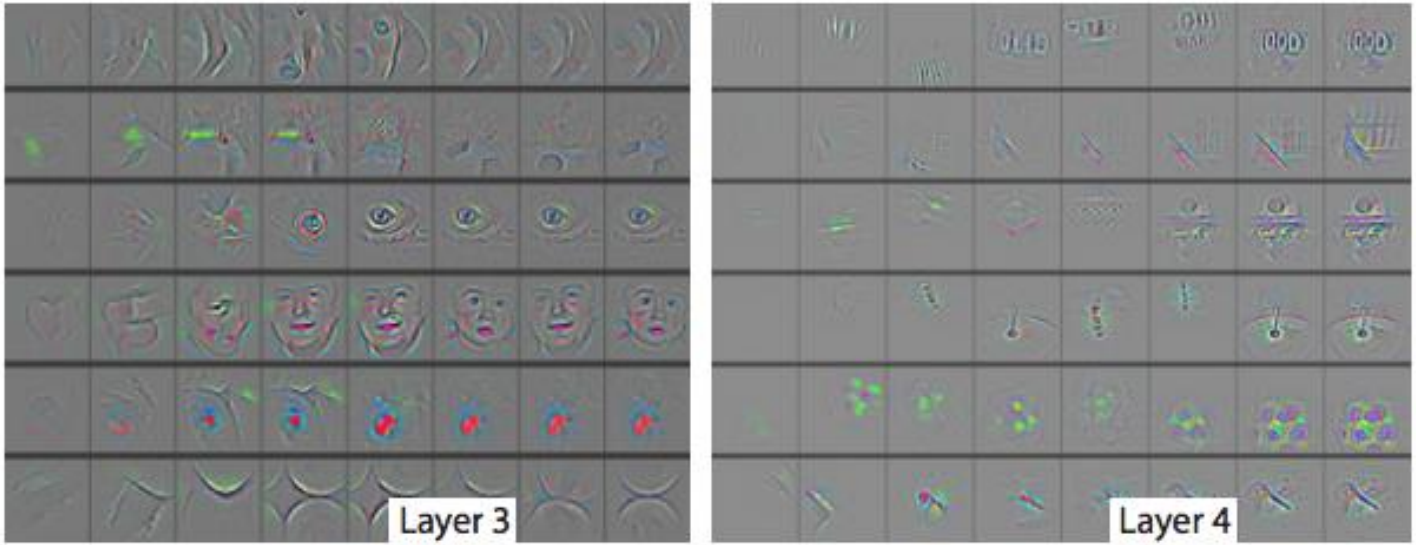
Spatial Local Features

Translation In-variant

Each filter searches for a particular feature at different image locations (translation invariance)

Who decides these features?

The network itself while **training** learns the filter weights and bias terms.

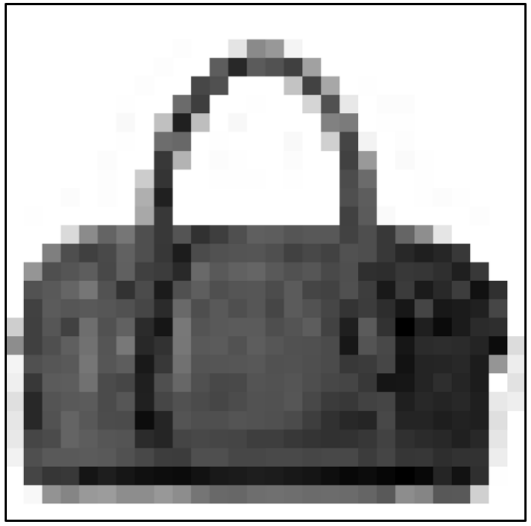


volution of randomly chosen subset of model features at **training epochs** 1,2,5,10,20,30,40,64.

Visualizing and Understanding Convolutional Networks,
Matthew D. Zeiler and Rob Fergus, ECCV 2014

Activation map/layer=feature map

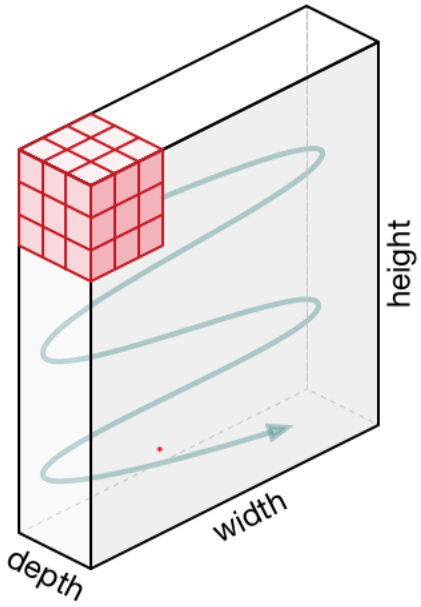
Input Image



Output From Conv2D

(Feature Maps after ReLU Processing)

How it works for Images?



0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...	...	...	...	...	...	...

Input Channel #1 (Red)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...	...	...	...	...	...	...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...	...	...	...	...	...	...

Input Channel #3 (Blue)

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2

0	1	1
0	1	0
1	-1	1

Kernel Channel #3

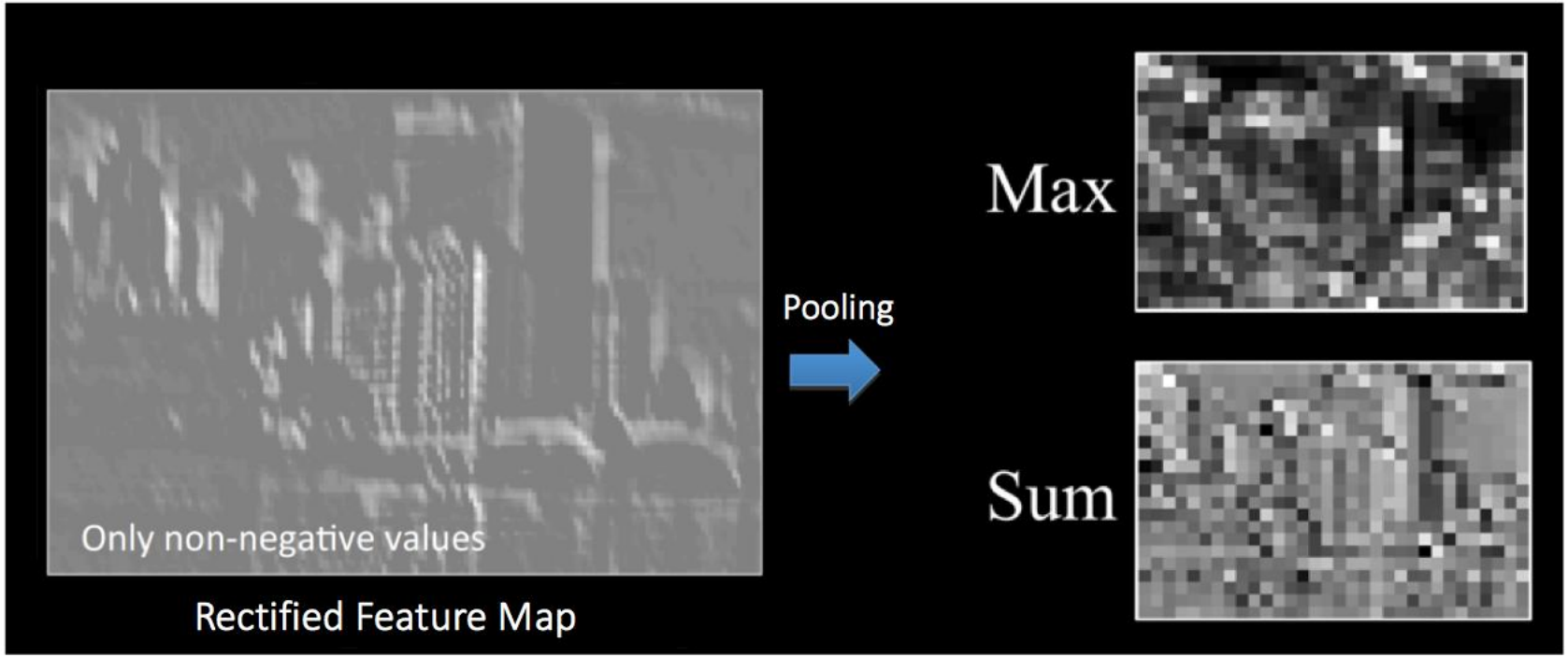
$308 + (-498) + 164 + 1 = -25$

Bias = 1

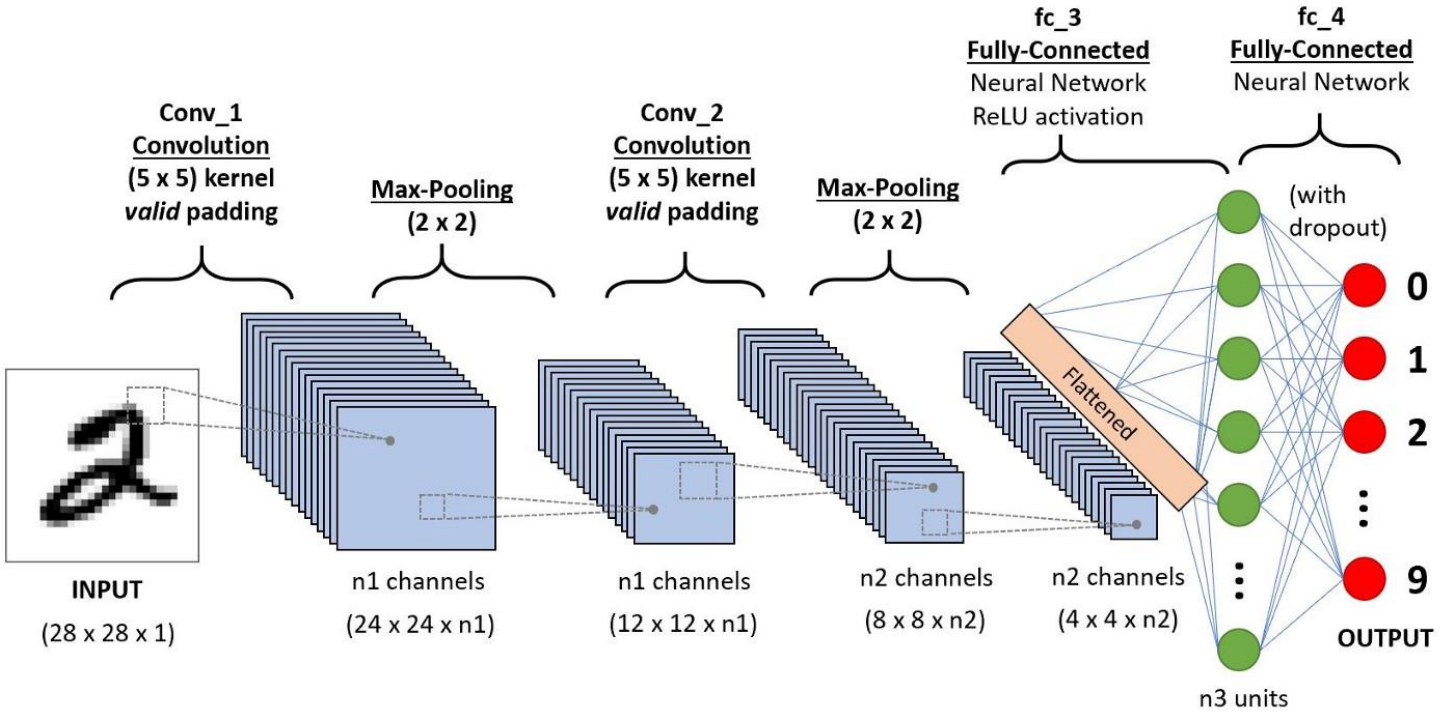
Output

-25				...
				...
				...
				...
...	...	...	...	...

Pooling (Down sampling)

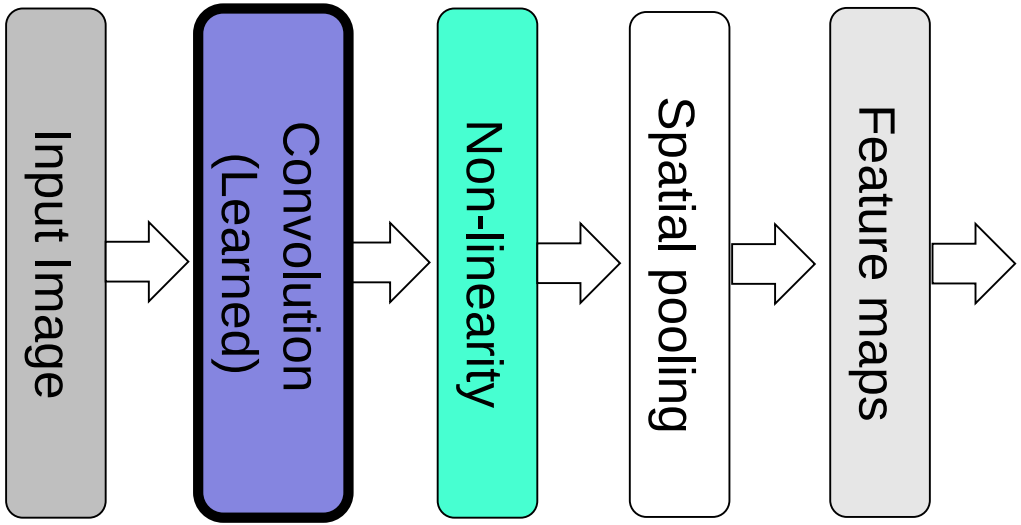


CNN Overall Architecture



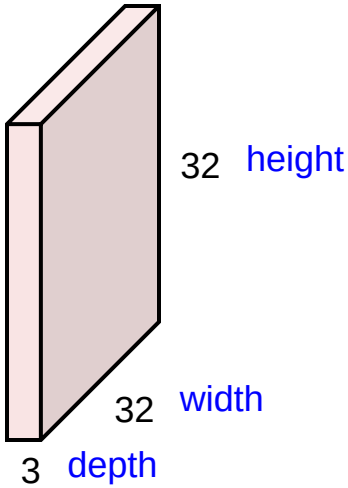
Carnegie Mellon University

CNN Components



Convolution Layer

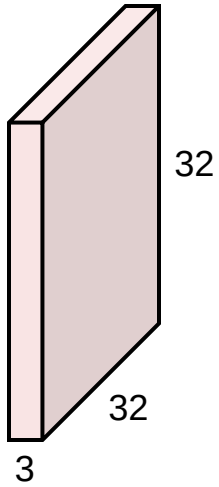
32x32x3 image -> preserve spatial structure



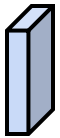
Carnegie Mellon University

Convolution Layer

32x32x3 image



5x5x3 filter

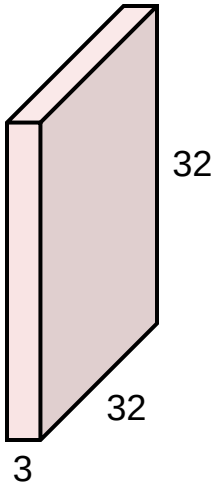


Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

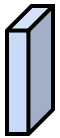
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image

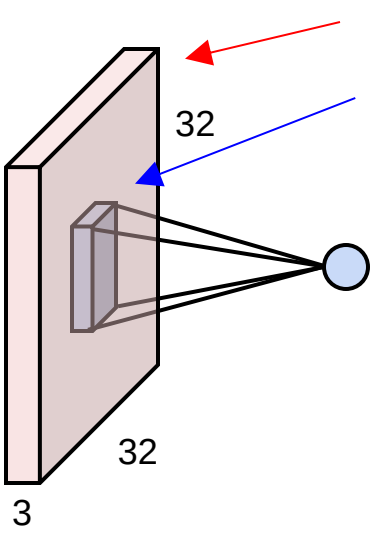


5x5x3 filter



Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Convolution Layer

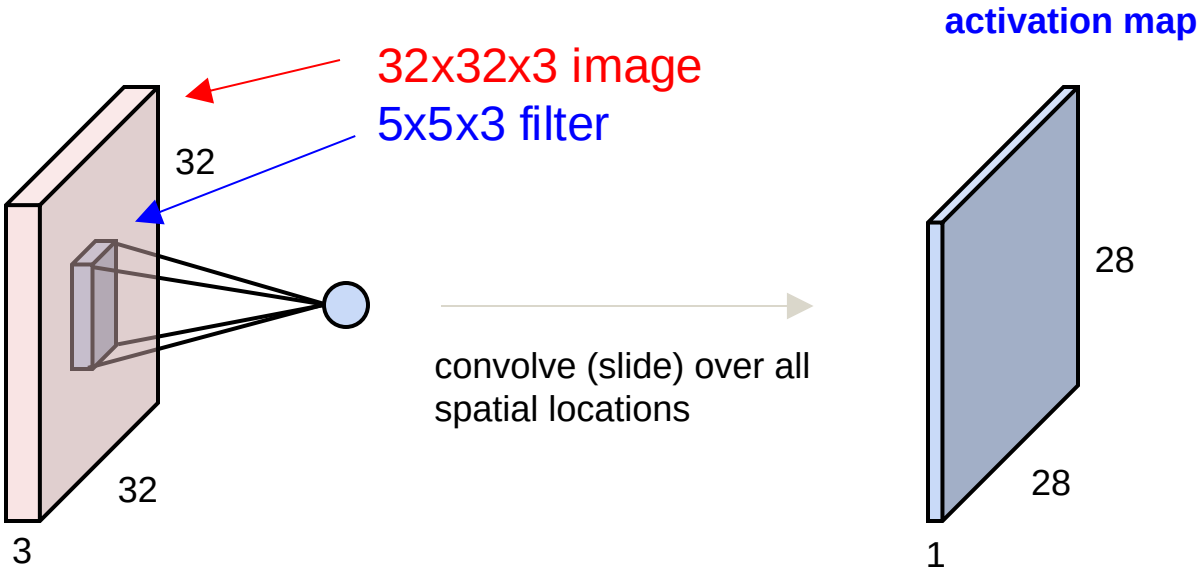


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Carnegie Mellon University

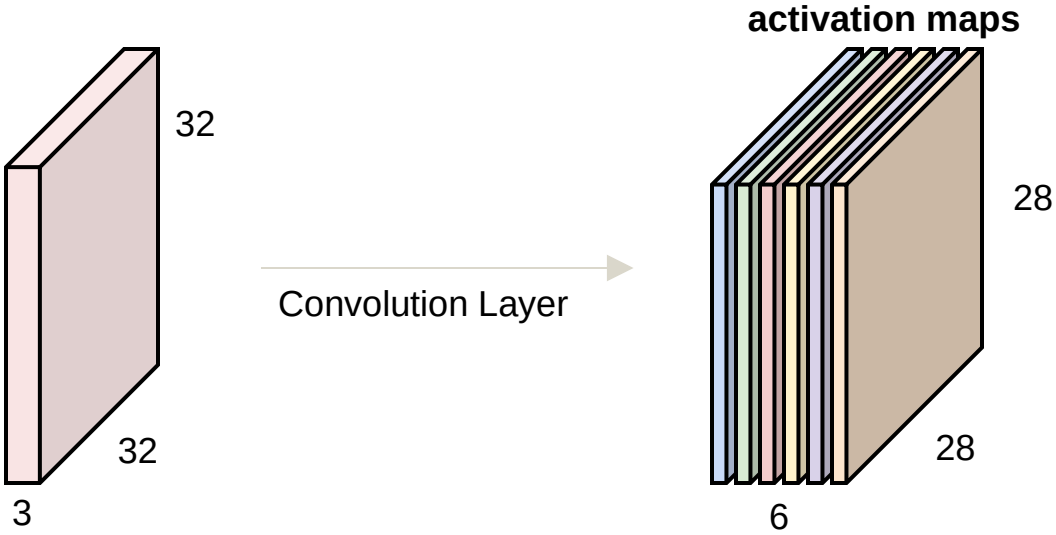
Convolution Layer



Convolution Layer

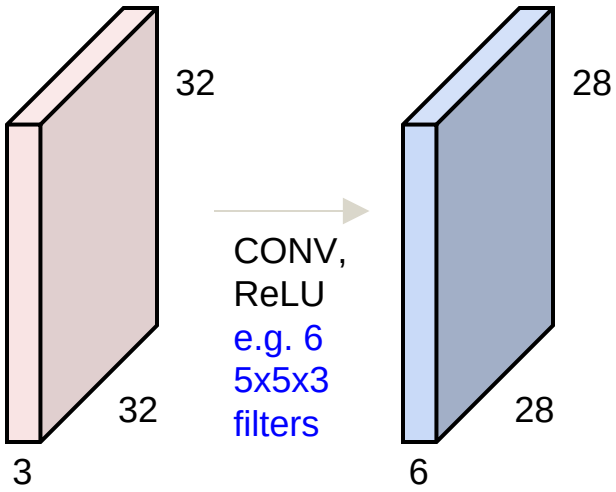
consider a second, green filter

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

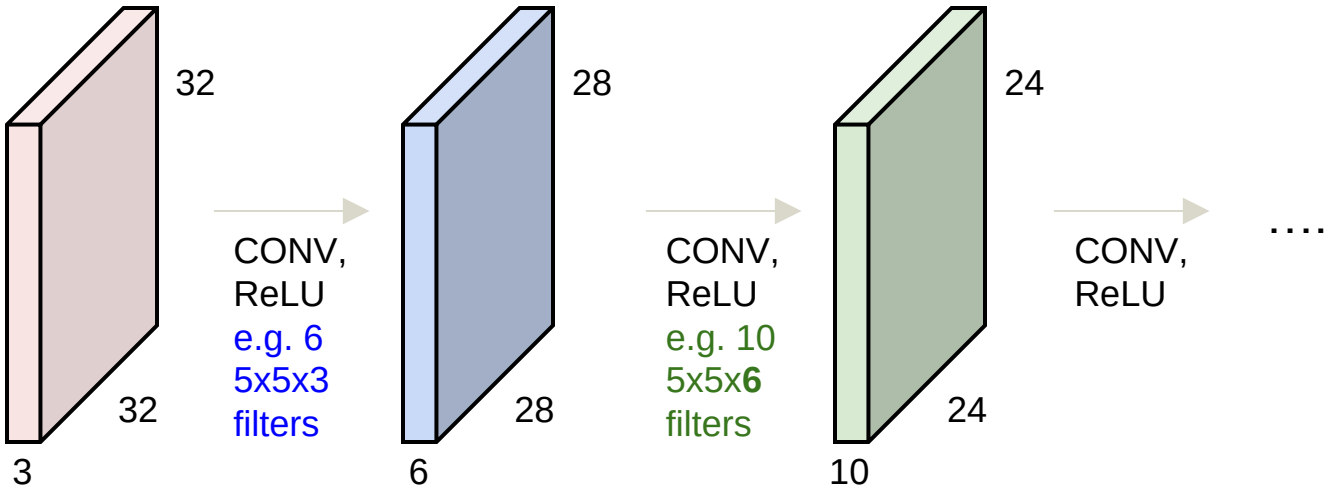


We stack these up to get a "new image" of size 28x28x6!

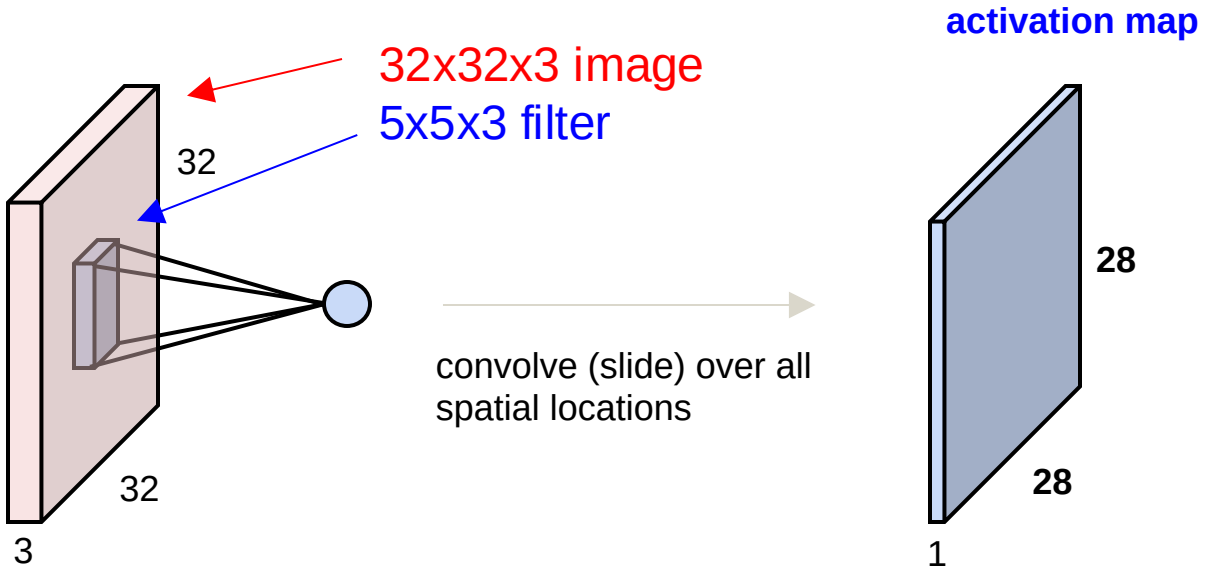
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



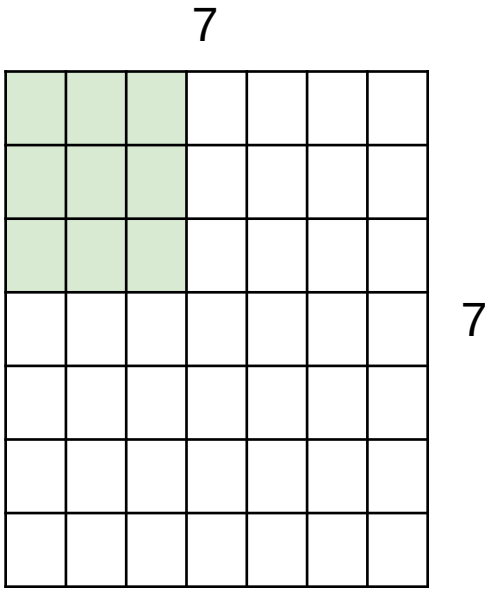
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



A closer look at spatial dimensions:

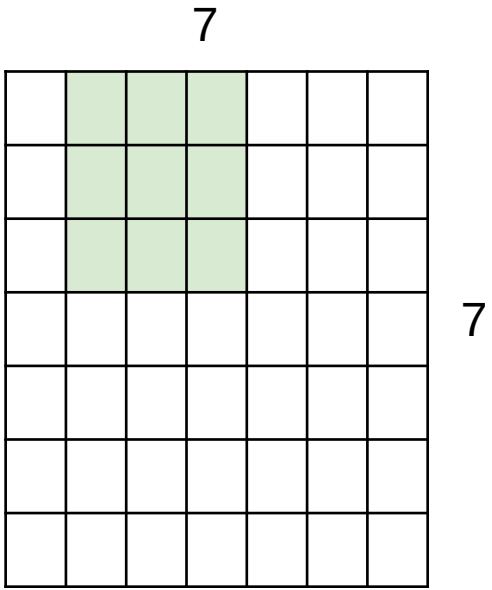


A closer look at spatial dimensions:



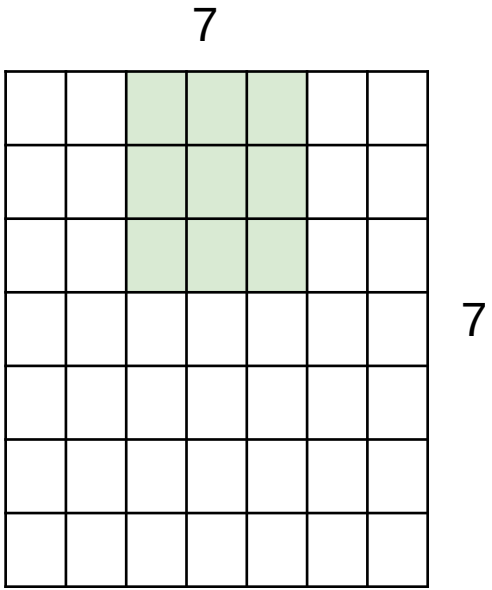
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



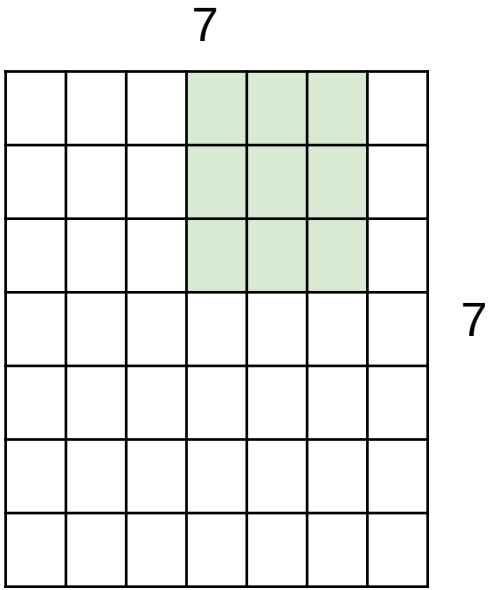
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



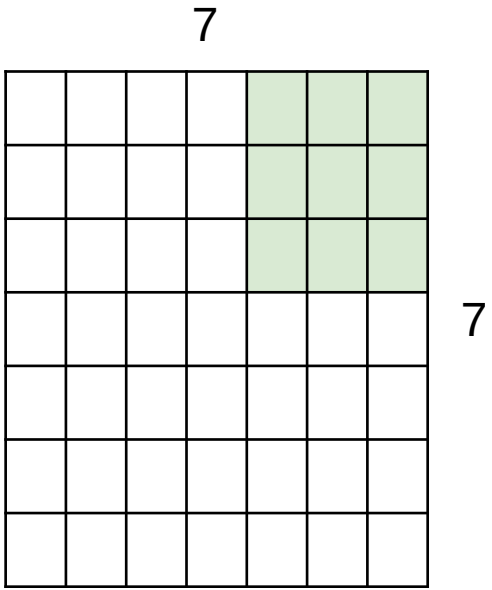
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

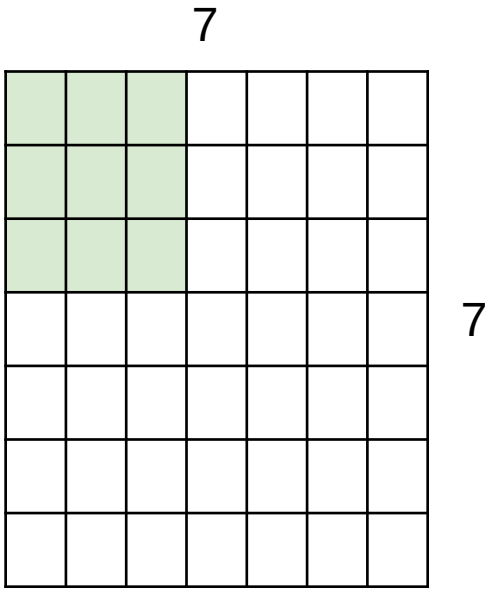
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

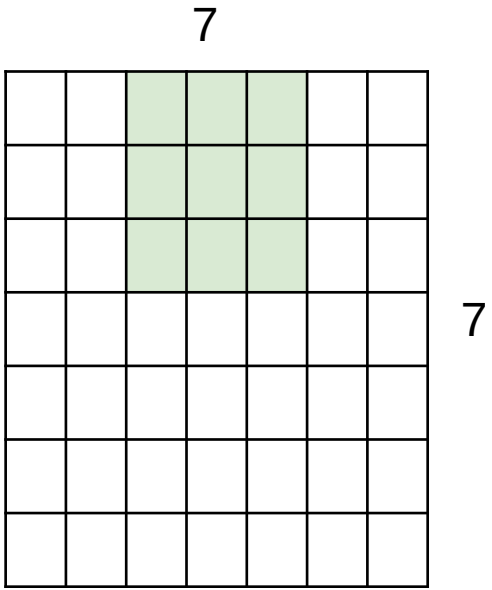
=> 5x5 output

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

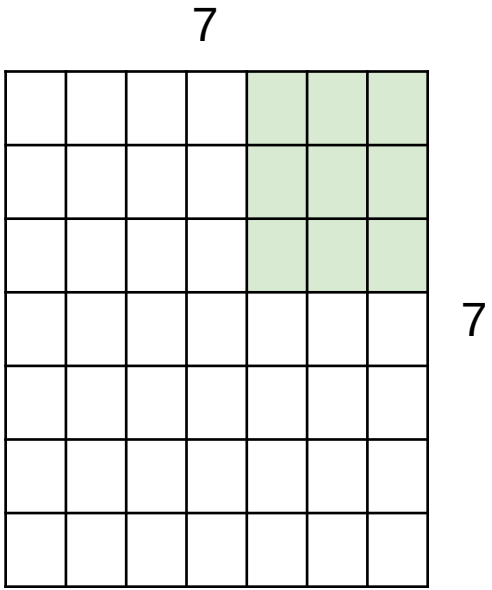
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

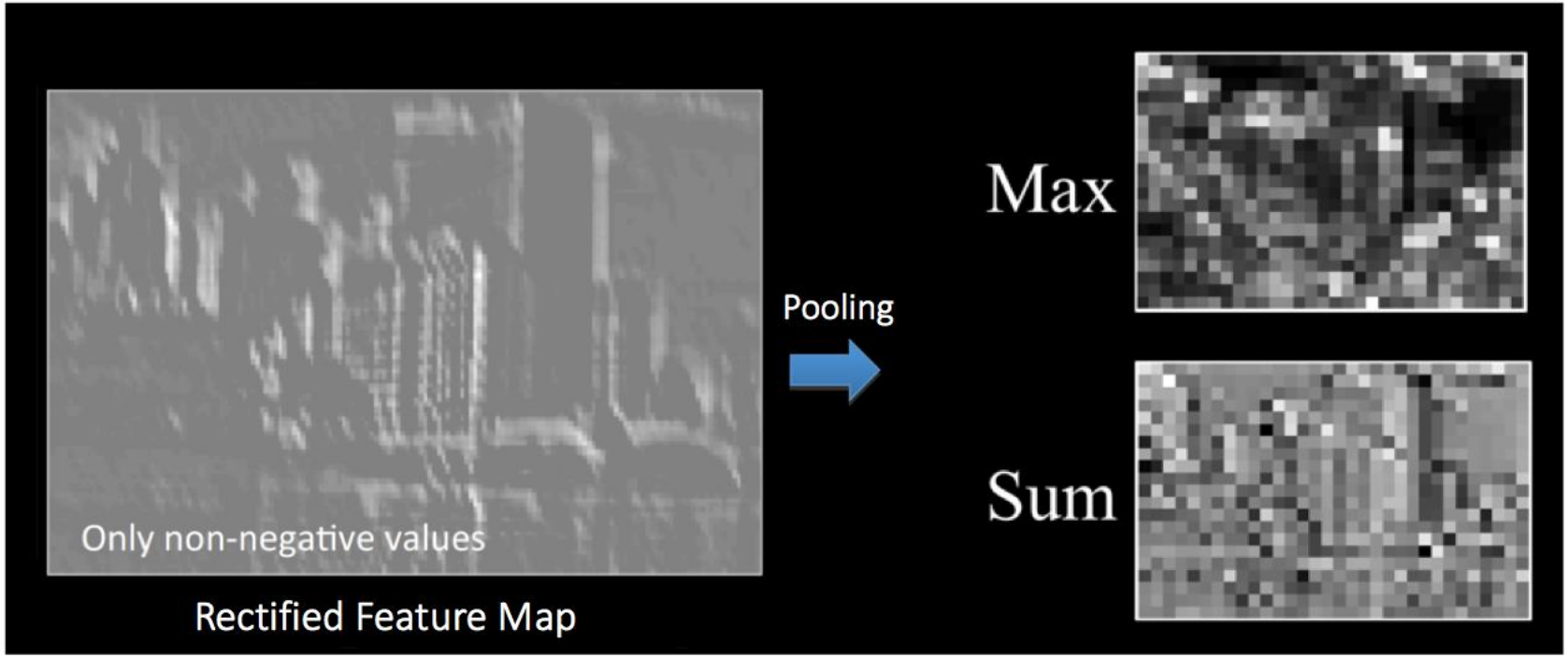
Carnegie Mellon University

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

Pooling (Down sampling)



Pooling (Down sampling)

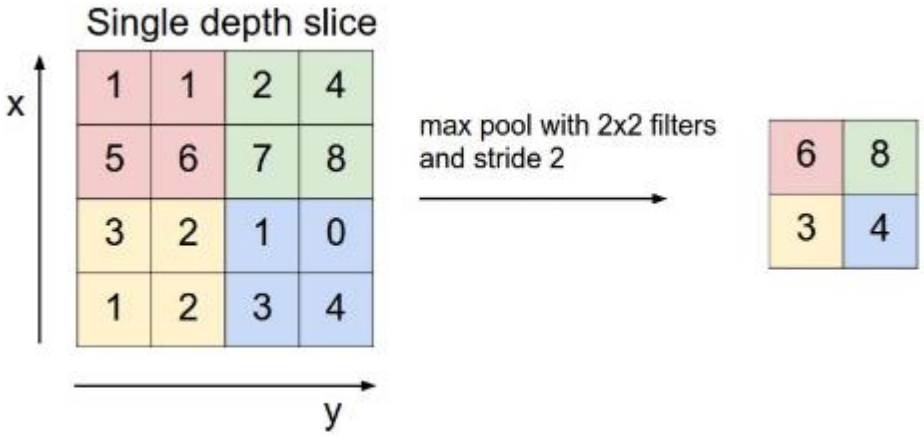


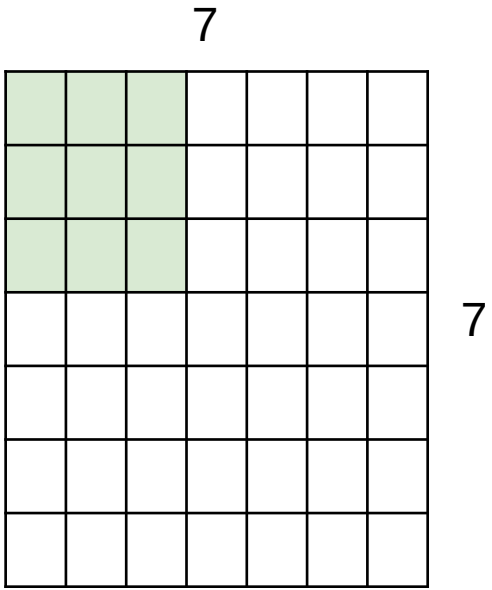
Fig. 2: F=2 S=2 Max Pooling

Max Pooling

Reduces dimensionality of each feature map, but retains the most important information

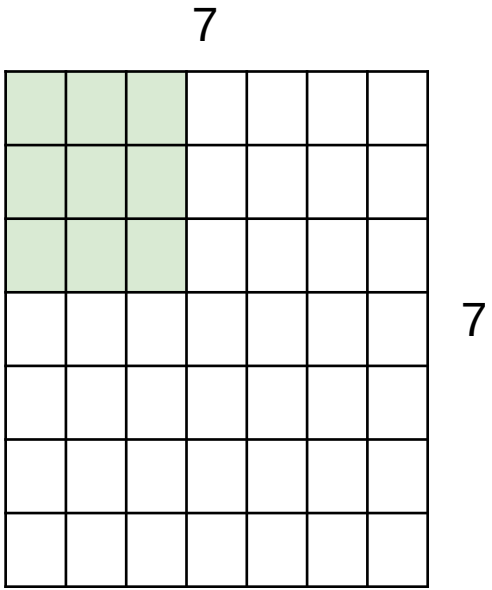
Reduced number of parameters reduces computation, memory reads, storage requirements and over-fitting to training data
Makes the network invariant to small transformations in input image, as max pooled value over local neighborhood won't change on small distortions

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

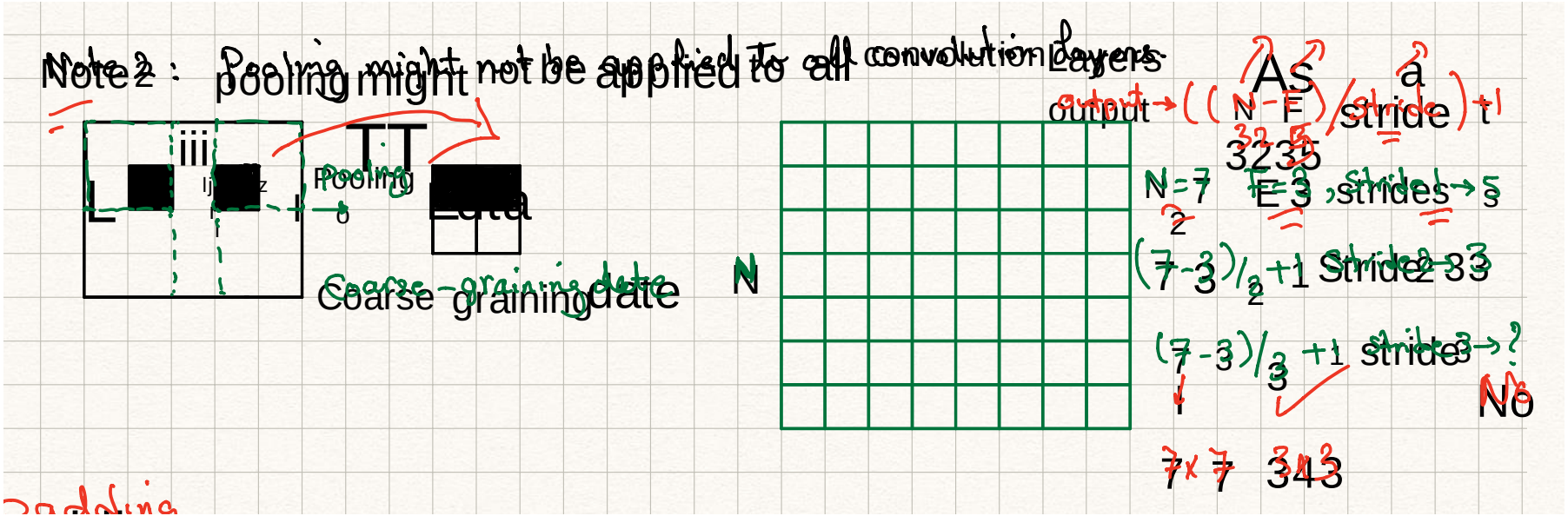
A closer look at spatial dimensions:



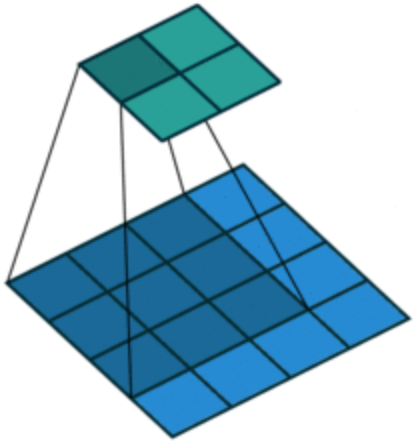
7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

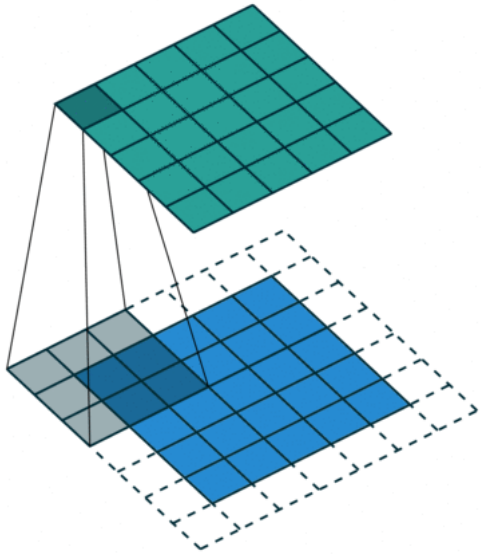
Padding Philosophy



Convolution with padding



4x4 input. 3x3 filter. Stride = 1.
2x2 output.



5x5 input. 3x3 filter. Stride = 1.
5x5 output.

Animation source: https://github.com/vdumoulin/conv_arithmetic

Padding

PADDING

MOTIVATION

We have a dimensional constraint on CNN: the dimension of an output layer is $(N - F)/S + 1$, where N is the dimension of the input, F is the dimension of the filter, and S is the stride. Under this constraint, certain inputs cannot be performed CNN. For instance, if we have $N = 7, F = 3, S = 3$, the output dimension is non-integer. Therefore, we need a tool to fix dimension disagreement.

ZERO-PADDING

For spatial rearrangement, we add zeros outside the original data set. For instance, if our original data set is $\begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}$,

after zero-padding once, the resulting matrix is $\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$.

If we apply zero padding once to the sample dimension mentioned in the motivation section, $N = 9$ after padding and CNN can be performed on the padded data.

Padding

OUTPUT DIMENSION

The output dimension of a padding layer in which the dimension of the input being N, the dimension of the filter being F, the stride being S and padding being P:

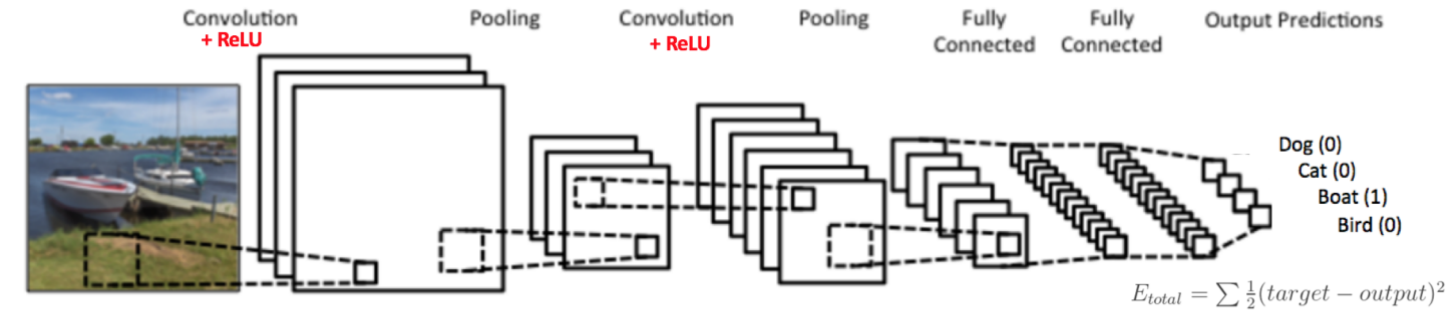
$$\text{dim} = (2P + N - F)/S + 1$$

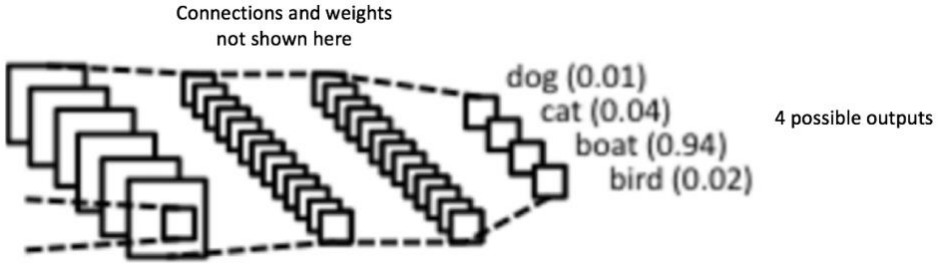
And we can analyze input and output on a Conv Layer:

1. Accepts a volume of size $W_1 \times H_1 \times D_1$
2. Takes two hyperparameters:
   (a) number of filters K
   (b) spatial extent F
   (c) stride S
   (d) amount of zero padding P
3. Produces a volume of size $W_2 \times H_2 \times D_2$:
   (a) $W_2 = (W_1 - F + 2P)/S + 1$
   (b) $H_2 = (H_1 - F + 2P)/S + 1$
   (c) $D_2 = K$
4. With parameter sharing, it introduces FFD1 weights per filter, for a total of $(F \cdot F \cdot D1) \cdot K$ weights and K biases

Note: A common setting of the hyperparameters is $\{F = 3, S = 1, P = 1\}$

Putting it together





Training Procedure

- **Step1:** We initialize all filters and parameters / weights with random values

- **Step2:** The network takes a training image as input, goes through the forward propagation step (convolution, ReLU and pooling operations along with forward propagation in the Fully Connected layer) and finds the output probabilities for each class.
  - Lets say the output probabilities for the boat image above are [0.2, 0.4, 0.1, 0.3]
  - Since weights are randomly assigned for the first training example, output probabilities are also random.

https://towardsdatascience.com/

Training Procedure

- **Step3:** Calculate the total error at the output layer (summation over all 4 classes)
  - **Total Error = ∑ ½ (target probability – output probability) ²**

- **Step4:** Use Backpropagation to calculate the *gradients* of the error with respect to all weights in the network and use *gradient descent* to update all filter values / weights and parameter values to minimize the output error.
  - The weights are adjusted in proportion to their contribution to the total error.
  - When the same image is input again, output probabilities might now be [0.1, 0.1, 0.7, 0.1], which is closer to the target vector [0, 0, 1, 0].
  - This means that the network has *learnt* to classify this particular image correctly by adjusting its weights / filters such that the output error is reduced.
  - Parameters like number of filters, filter sizes, architecture of the network etc. have all been fixed before Step 1 and do not change during training process – only the values of the filter matrix and connection weights get updated.

https://towardsdatascience.com/

Training Procedure

•**Step5:** Repeat steps 2-4 with all images in the training set.

The above steps *train* the ConvNet – this essentially means that all the weights and parameters of the ConvNet have now been optimized to correctly classify images from the training set.

When a new (unseen) image is input into the ConvNet, the network would go through the forward propagation step and output a probability for each class (for a new image, the output probabilities are calculated using the weights which have been optimized to correctly classify all the previous training examples). If our training set is large enough, the network will (hopefully) generalize well to new images and classify them into correct categories.

https://towardsdatascience.com/

Classification: ImageNet Challenge top-5 error