



دانشگاه کردستان
University of Kurdistan
زانکۆی کوردستان

**Department of Computer Engineering
University of Kurdistan**

Deep learning (Graduate level)

Convolutional Neural Networks (CNN)

By: Dr. Alireza Abdollahpouri

Deep Computer Vision

Our visual system is trained on images seen in 540 mln of years!



**“To know what is
where by looking.”**

Images are Numbers

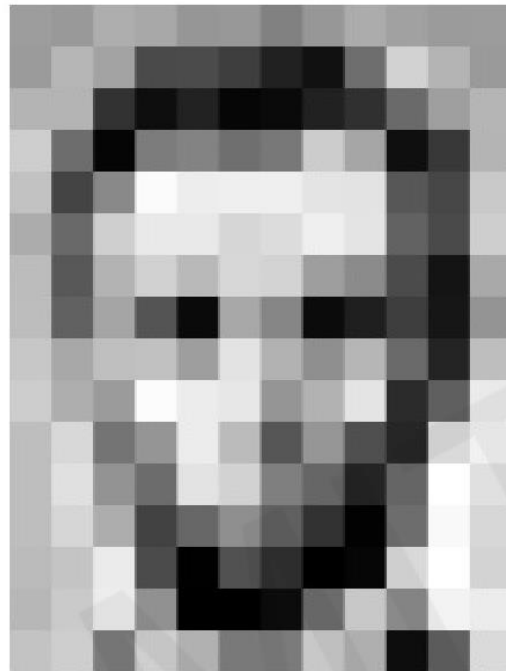
What I see



What a computer sees

08	02	22	97	38	15	00	40	00	75	04	05	07	78	52	12	50	77	91	08
49	49	99	40	17	81	18	57	40	87	17	40	98	43	69	48	04	56	42	00
81	49	31	73	55	79	14	29	93	71	40	47	53	88	30	03	49	13	36	65
52	70	95	23	04	60	11	42	69	24	68	56	01	32	56	71	37	02	36	91
22	31	16	71	51	67	63	89	41	92	36	54	22	40	40	28	66	33	13	80
24	47	32	60	99	03	45	02	44	75	33	53	78	36	84	20	35	17	12	50
32	98	81	28	64	23	67	10	26	38	40	67	59	54	70	66	18	38	64	70
47	26	20	68	02	62	12	20	95	63	94	39	63	08	40	91	66	49	94	21
24	55	58	05	66	73	99	26	97	17	78	78	96	83	14	88	34	89	63	72
21	36	23	09	75	00	74	44	20	45	35	14	00	61	33	97	34	31	33	95
78	17	53	28	22	75	31	47	15	94	03	80	04	62	16	14	09	53	56	92
16	39	05	42	96	35	31	47	55	58	88	24	00	17	54	24	36	29	85	57
86	56	00	48	55	71	89	07	05	44	44	37	44	60	21	58	51	54	17	58
19	80	81	68	05	94	47	69	28	73	92	13	86	52	17	77	04	89	55	40
04	52	08	83	97	35	99	16	07	97	57	32	16	26	26	79	33	27	98	66
88	36	68	87	57	62	20	72	03	46	33	67	46	55	12	32	63	93	53	69
04	42	16	73	38	25	39	11	24	94	72	18	08	46	29	32	40	62	76	36
20	69	36	41	72	30	23	88	34	62	99	69	82	67	59	85	74	04	36	16
20	73	35	29	78	31	90	01	74	31	49	71	48	86	81	16	23	57	05	54
01	70	54	71	83	51	54	69	16	92	33	48	61	43	52	01	89	19	47	48

Images are Numbers



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	83	17	110	210	180	154
180	180	50	14	34	6	10	83	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	95	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

What the computer sees

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	83	17	110	210	180	154
180	180	50	14	34	6	10	83	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	95	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

An image is just a matrix of numbers $[0,255]$!
i.e., $1080 \times 1080 \times 3$ for an RGB image

Grayscale images are **2D matrices of pixel brightness**

Color image: RGB 3 channels

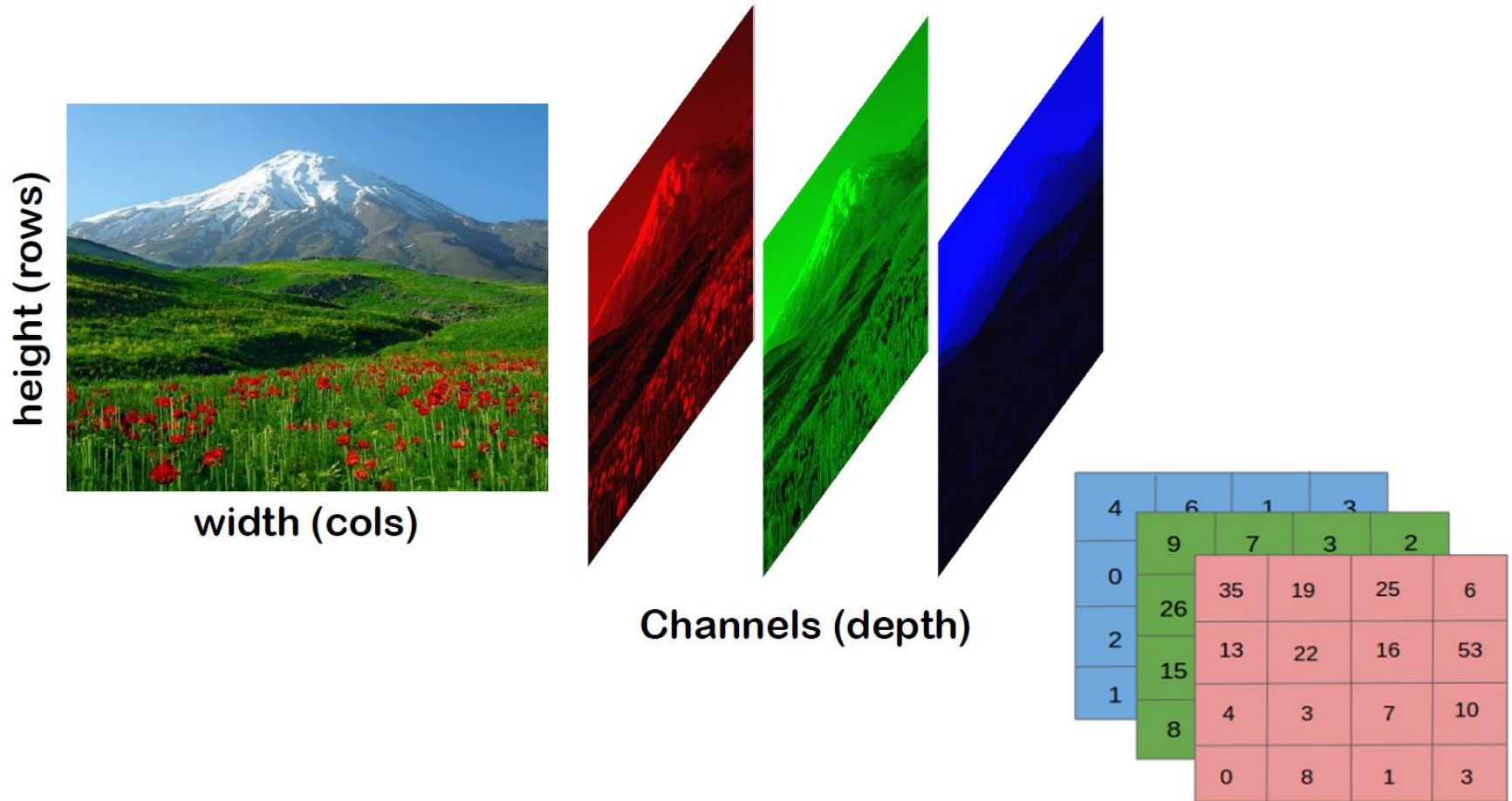
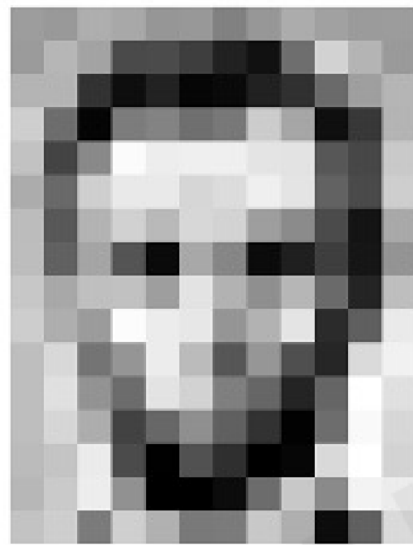


Image Classification task



Input Image



157	153	174	168	160	162	129	161	172	161	166	166
166	162	163	74	76	62	33	17	113	210	180	164
180	180	50	14	34	6	10	33	48	106	159	181
206	109	6	124	131	111	120	204	166	16	56	180
194	60	137	251	237	230	239	227	87	71	201	
172	106	207	233	233	214	220	239	228	98	74	206
188	88	179	208	186	216	211	168	138	76	20	169
189	97	166	84	10	148	134	11	31	63	22	148
199	168	191	193	198	227	178	143	182	106	96	190
200	174	193	232	236	231	149	178	228	43	96	234
190	216	116	148	236	187	86	160	73	38	218	241
190	234	147	108	227	210	137	102	96	101	206	234
190	214	173	66	103	143	96	63	3	108	249	216
187	196	236	76	1	81	47	0	6	217	266	211
183	202	237	145	0	0	12	108	208	138	243	236
196	209	123	207	177	121	123	200	173	13	96	218

Pixel Representation

classification

Lincoln

Washington

Jefferson

Obama

0.8

0.1

0.05

0.05

High-level Feature Detection

Let's identify key features in each image category



Nose,
Eyes,
Mouth



Wheels,
License Plate,
Headlights



Door,
Windows,
Steps

Manual Feature Extraction (challenges)

Viewpoint variation



Scale variation



Deformation



Occlusion



Illumination conditions



Background clutter



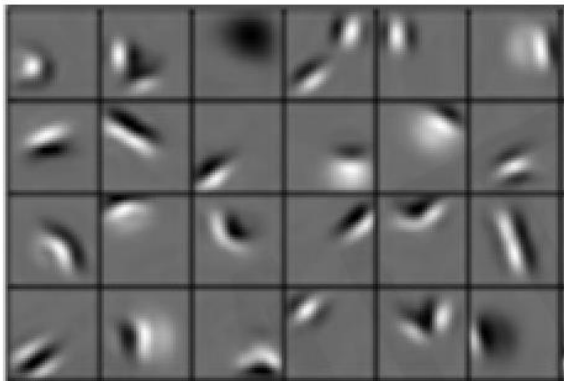
Intra-class variation



Learning Feature Representations

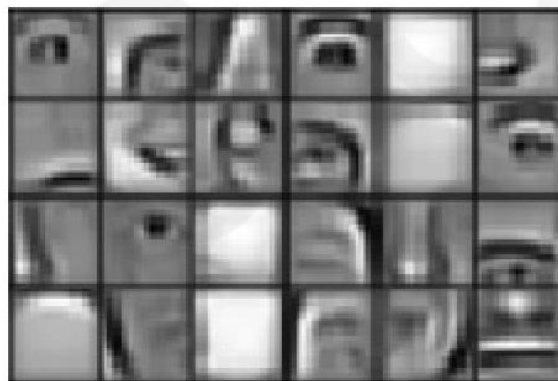
Can we learn a **hierarchy of features** directly from data instead of hand engineering?

Low level features



Edges, dark spots

Mid level features



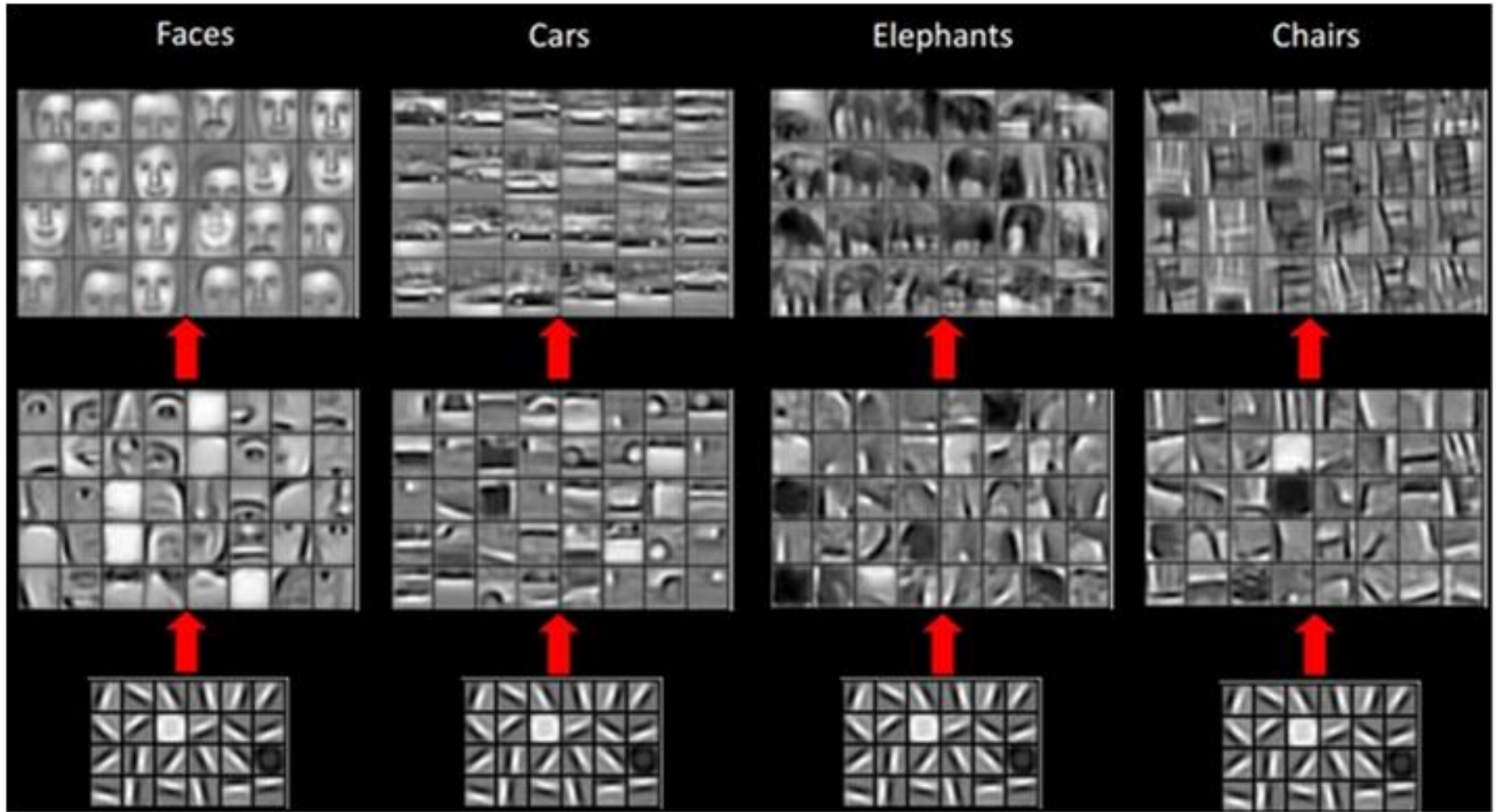
Eyes, ears, nose

High level features



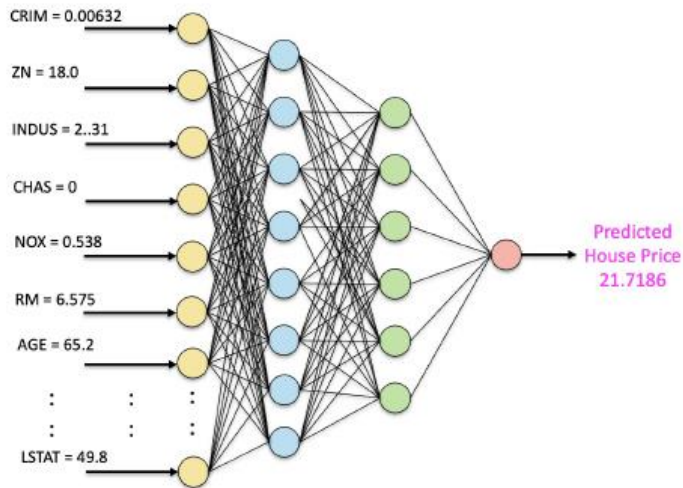
Facial structure

Learning Feature Representations



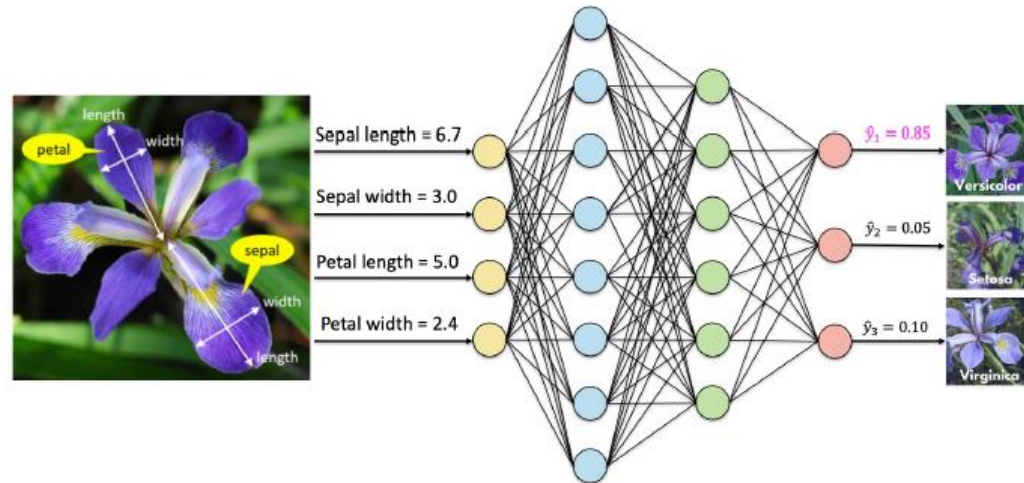
MLPs for Supervised learning tasks

Regression



- **Boston Housing Dataset**
 - 13 features and 506 records
 - A 3-Layer MLP (13-8-6-1)
 - Performance: RMSE = 3.97

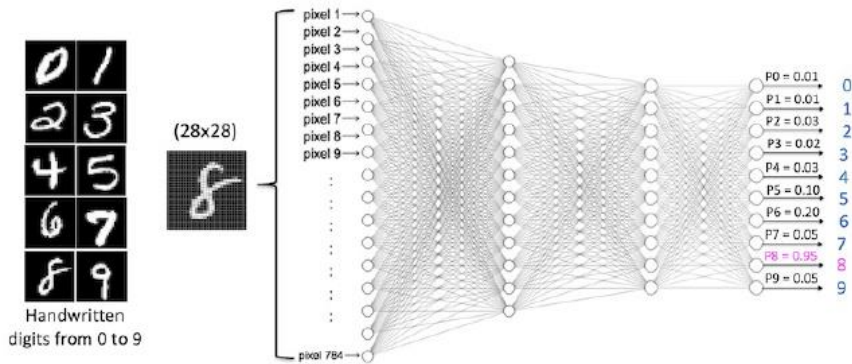
Classification



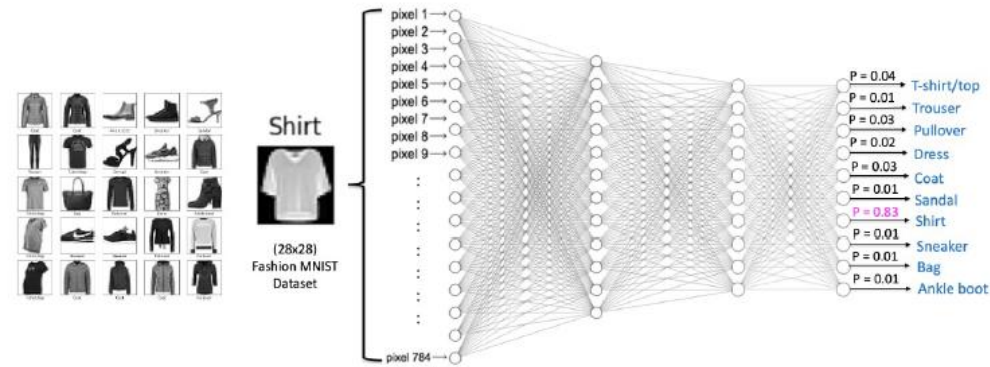
- **Iris Flower Dataset**
 - 4 features and 150 records
 - A 3-Layer MLP (4-25-15-3)
 - Performance: 98.7% Accuracy

MLPs for Grayscale Image Classifications

Handwritten Digits Recognition



Fashion Image Classification



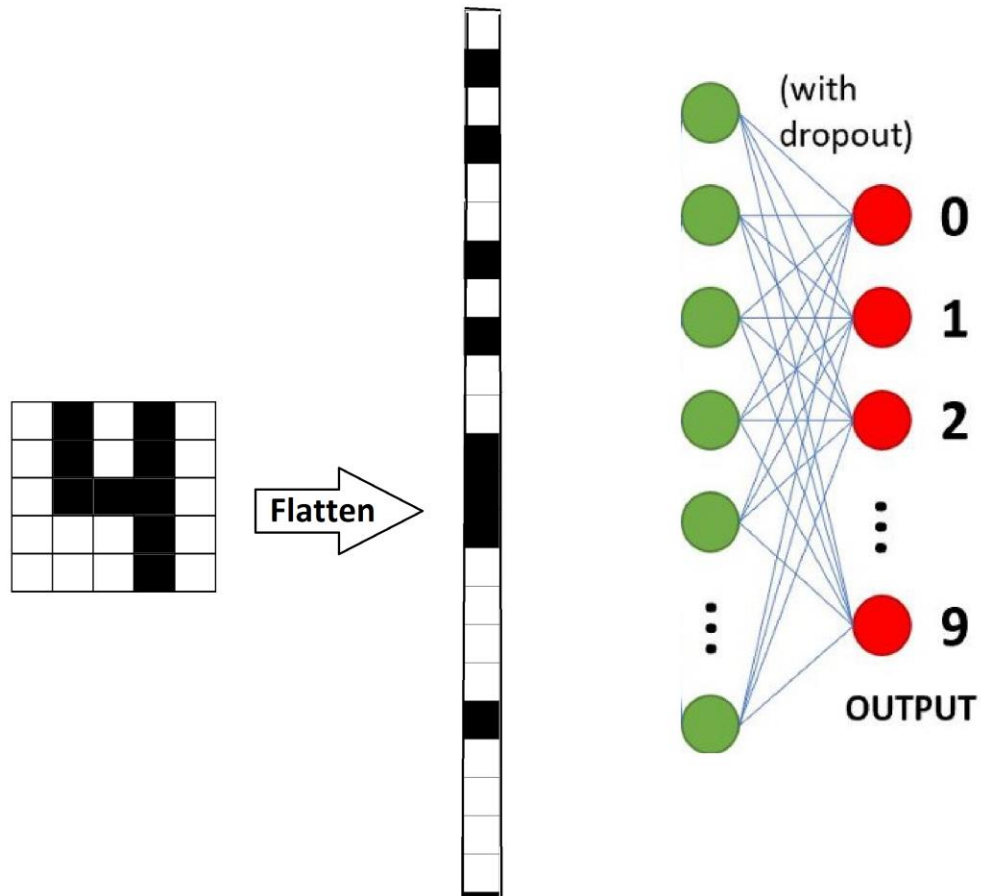
- **MNIST Dataset**

- 70,000 28x28 Grayscale Images
- 3-Layer MLP (784-128-64-10)
- Performance:** 98.36% Accuracy
- No. of Parameters:** 109.386K

- **Fashion MNIST Dataset**

- 70,000 28x28 Grayscale Images
- 3-Layer MLP (784-128-64-10)
- Performance:** 84.18% Accuracy
- No. of Parameters:** 109.386K

Traditional Method



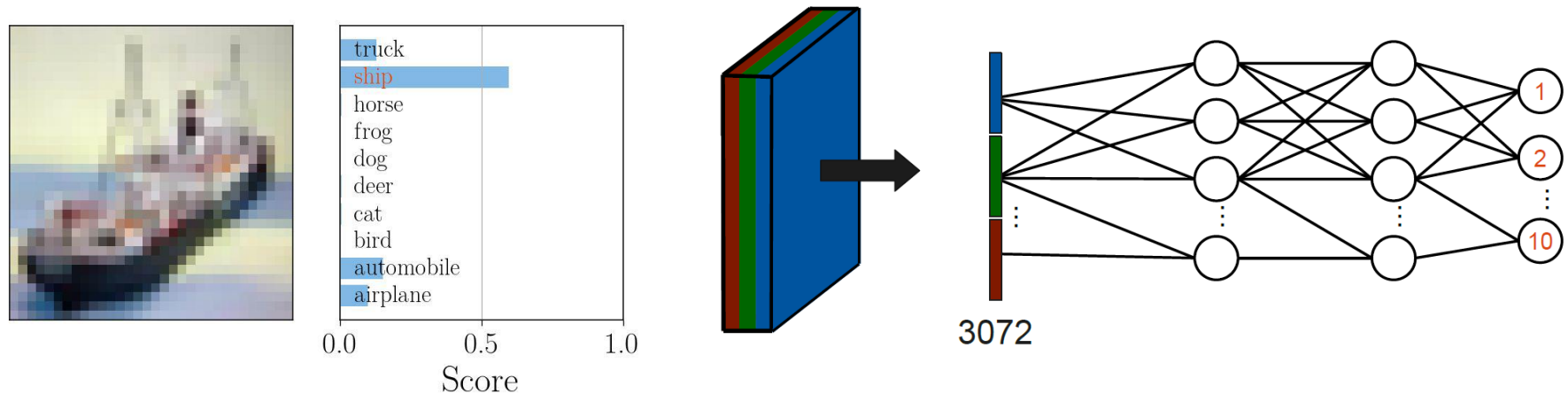
CIFAR-10 Classification Task

70,000 color images with 10 classes

Input $\mathbf{x} = (x_1, x_2, \dots, x_{3072})$, for $32 \times 32 \times 3 = 3072$ input features

Output $\mathbf{y} = (y_1, y_2, \dots, y_{10})$, one for each class

- frog, airplane, automobile, bird, cat, deer, dog, horse, ship, truck

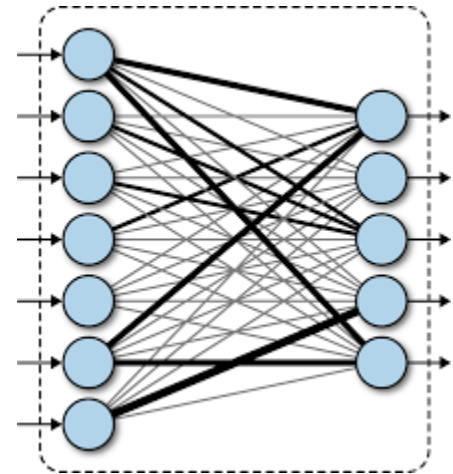


Traditional Method

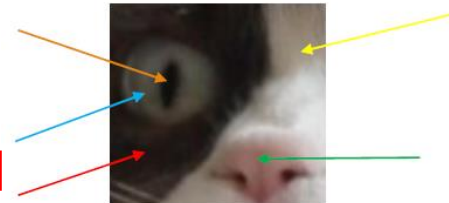
Previous DNNs use fully-connected layers
Connect **all** the neurons between the layers

Drawbacks:

- (-) **Large number of parameters**
- Easy to be over-fitted
- Large memory consumption
- (-) Does not enforce any structure, e.g., **No Spatial information**
- In many applications, local features are important, e.g., images



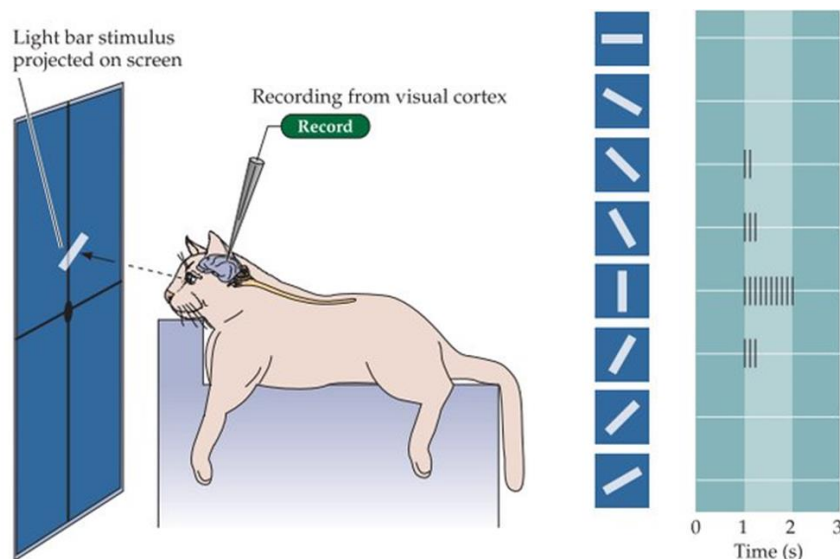
Neighboring
variables are
locally correlated



Hubel and Wiesel's experiment

Hubel and Wiesel's experiments on cats' visual cortex influenced the intuition behind CNN models.

- They discovered that certain neurons in the visual cortex were sensitive to edges and lines.
- Different neurons responded to specific orientations of edges, regardless of their position in the visual field.

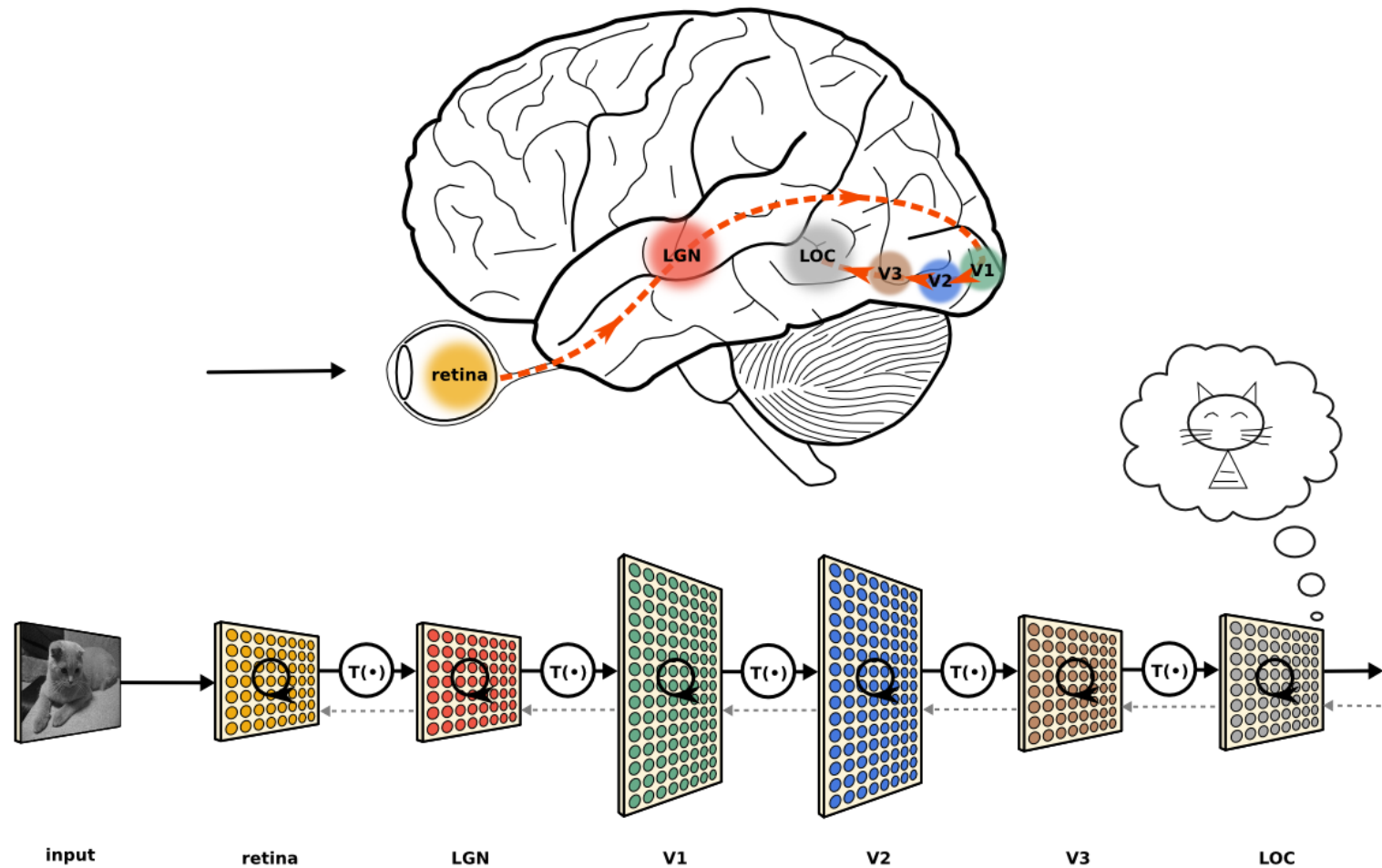


Hubel and Wiesel's experiment

- Their ground-breaking research led to the discovery of specialized cells in the visual cortex called "**simple cells**" and "**complex cells**."
- Simple cells responded selectively to specific orientations of lines or edges.
- Complex cells responded to more complex visual stimuli, such as moving lines or gratings

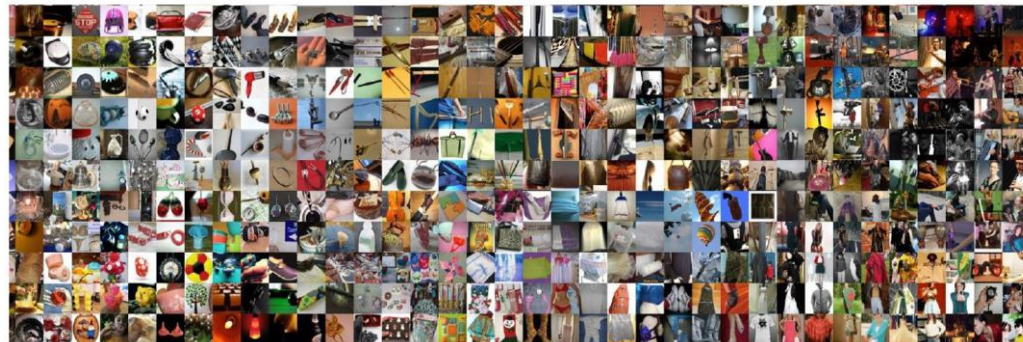
These findings led to the development of CNNs, which mimic the hierarchical processing of visual information in the brain.

Hierarchical visual processing in the brain



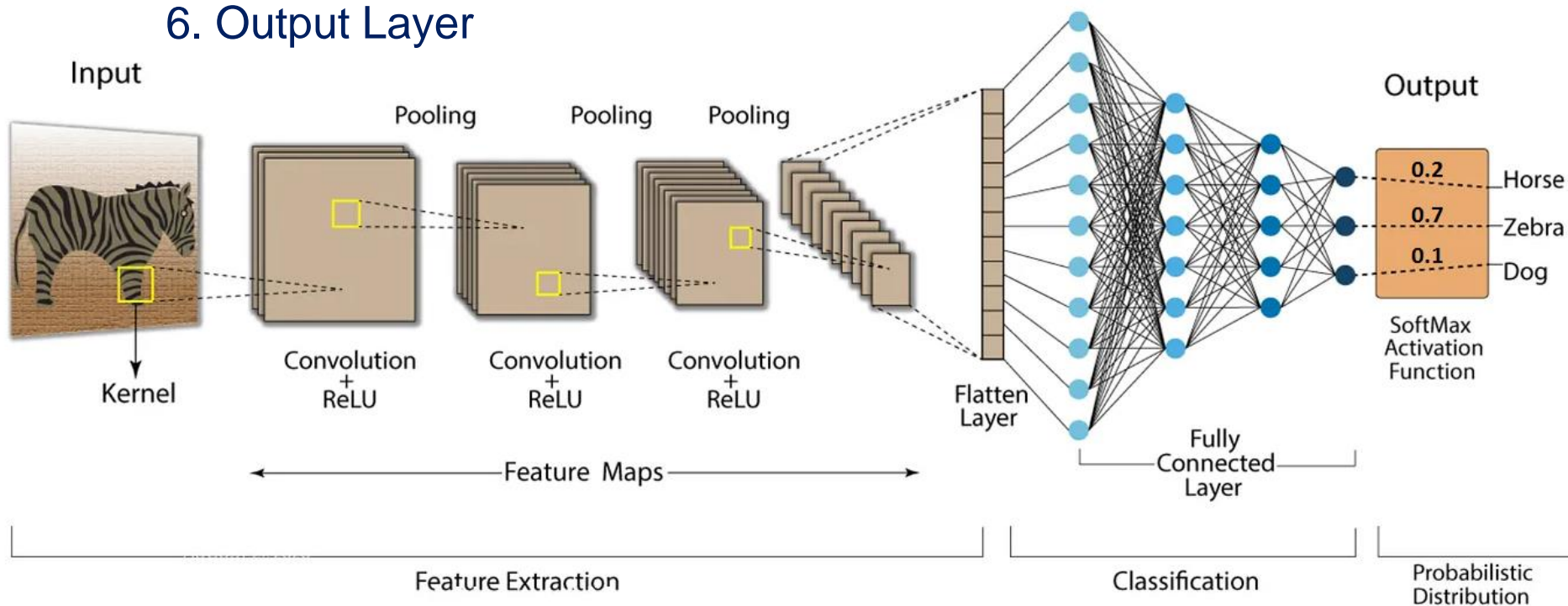
ImageNet and AlexNet (2012)

- ImageNet is a large-scale image dataset with over 14 million annotated images across over 2000 categories.
- The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is an annual image classification competition based on ImageNet started in 2010.
- In 2012, AlexNet, a deep 8-layer CNN developed by Krizhevsky et al., won ILSVRC with a 16.4% error rate, significantly outperforming traditional computer vision methods.
- Since AlexNet, ILSVRC error rates have continued to drop each year with newer CNN architectures.



Convolutional Neural Networks (CNN)

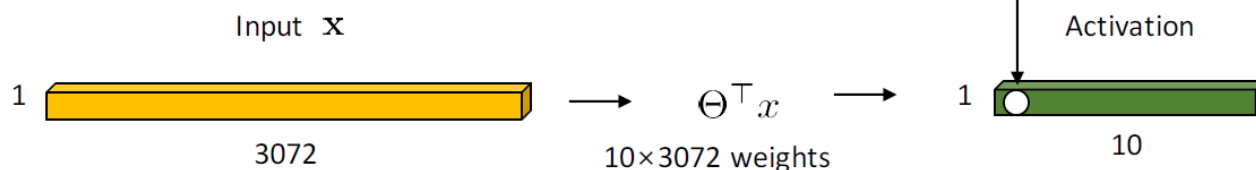
1. Input Layer (Height x Width x Depth)
2. Convolutional Layers
3. Activation Function (ReLU)
4. Pooling Layers
5. Fully Connected Layers
6. Output Layer



Convolutional layer

Fully-connected layer

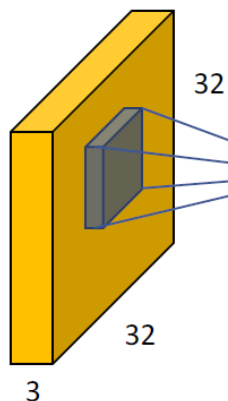
- $32 \times 32 \times 3$ image \rightarrow stretch to 3072×1



The result of taking a dot product between a row of Θ^T and the input

Convolution layer

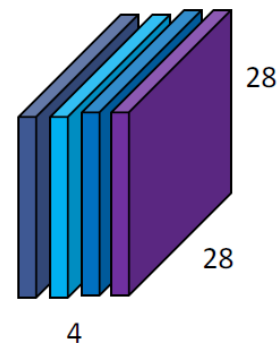
$32 \times 32 \times 3$ image



If there are **four** $5 \times 5 \times 3$ filters

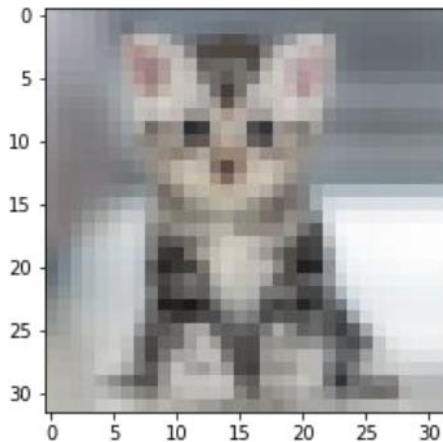
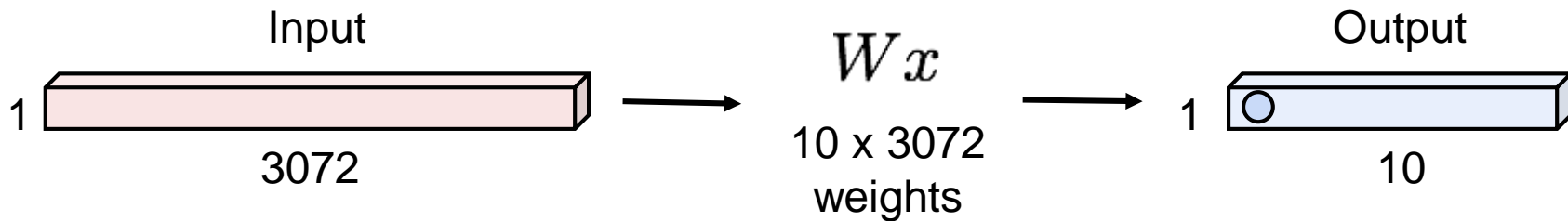
Convolve (slide) over all spatial locations

4 separate activation maps



Traditional Method

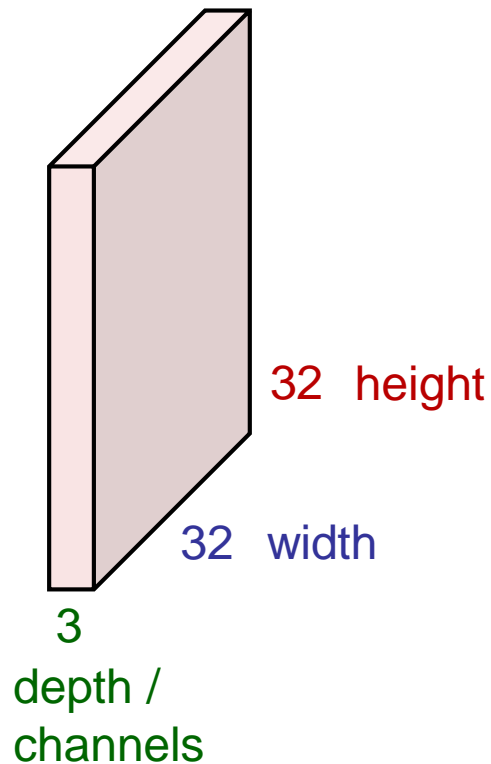
32x32x3 image -> stretch to 3072 x 1



→ Cat

Convolutional layer

3x32x32 image



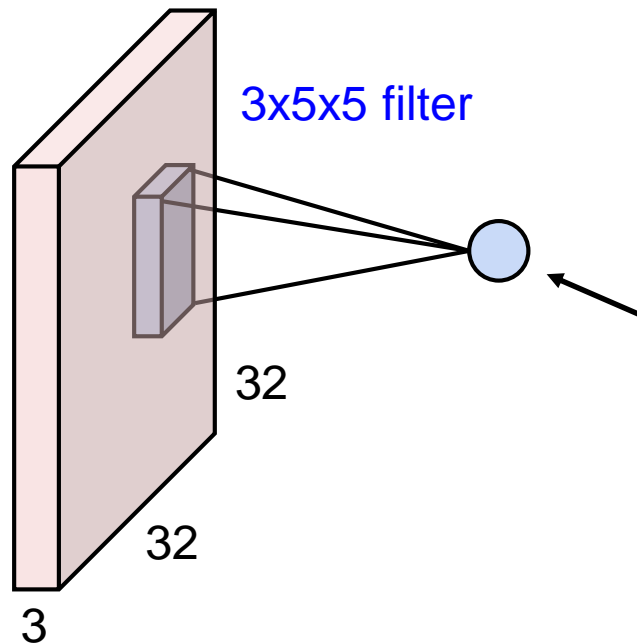
3x5x5 filter



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

Convolutional layer

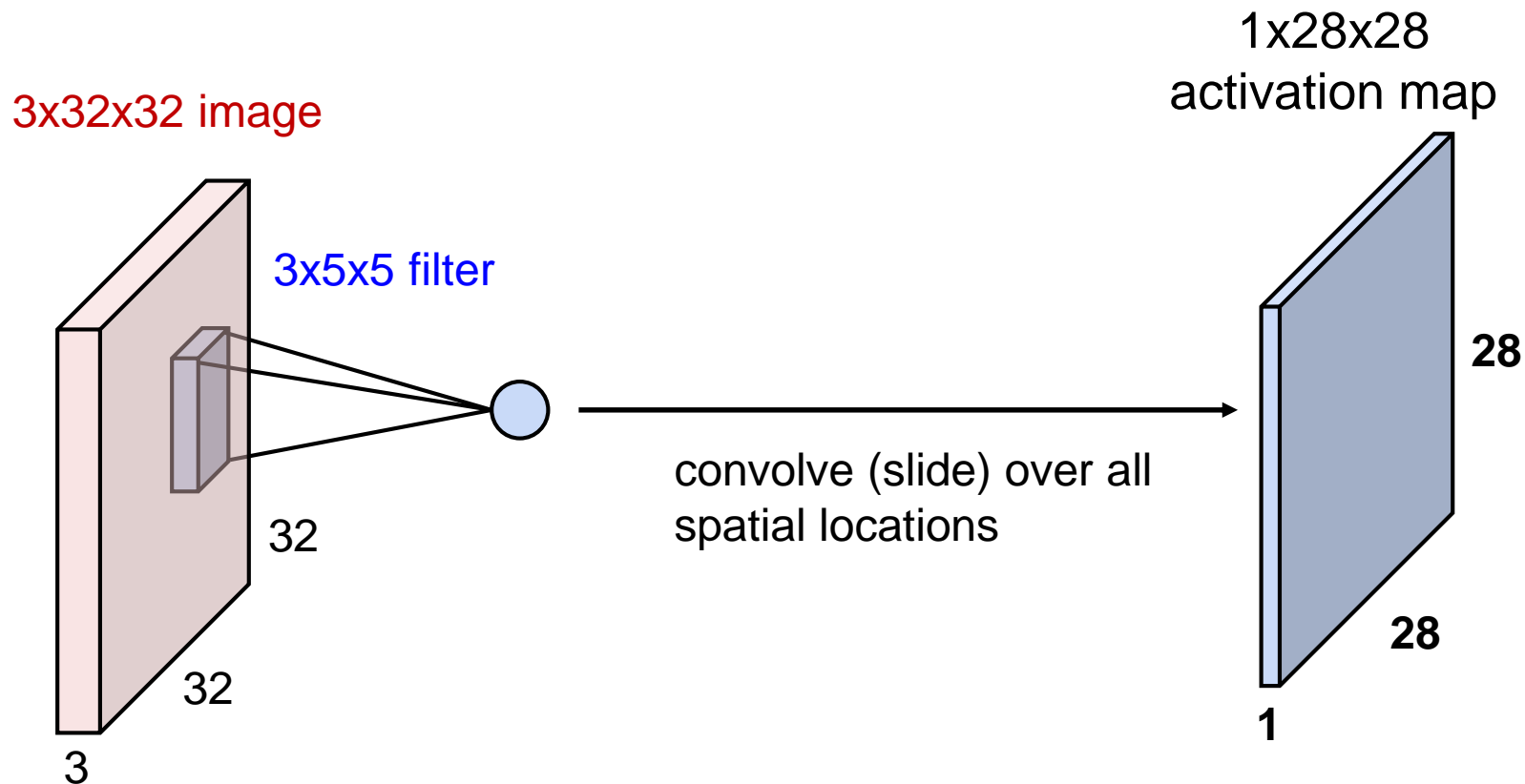
3x32x32 image



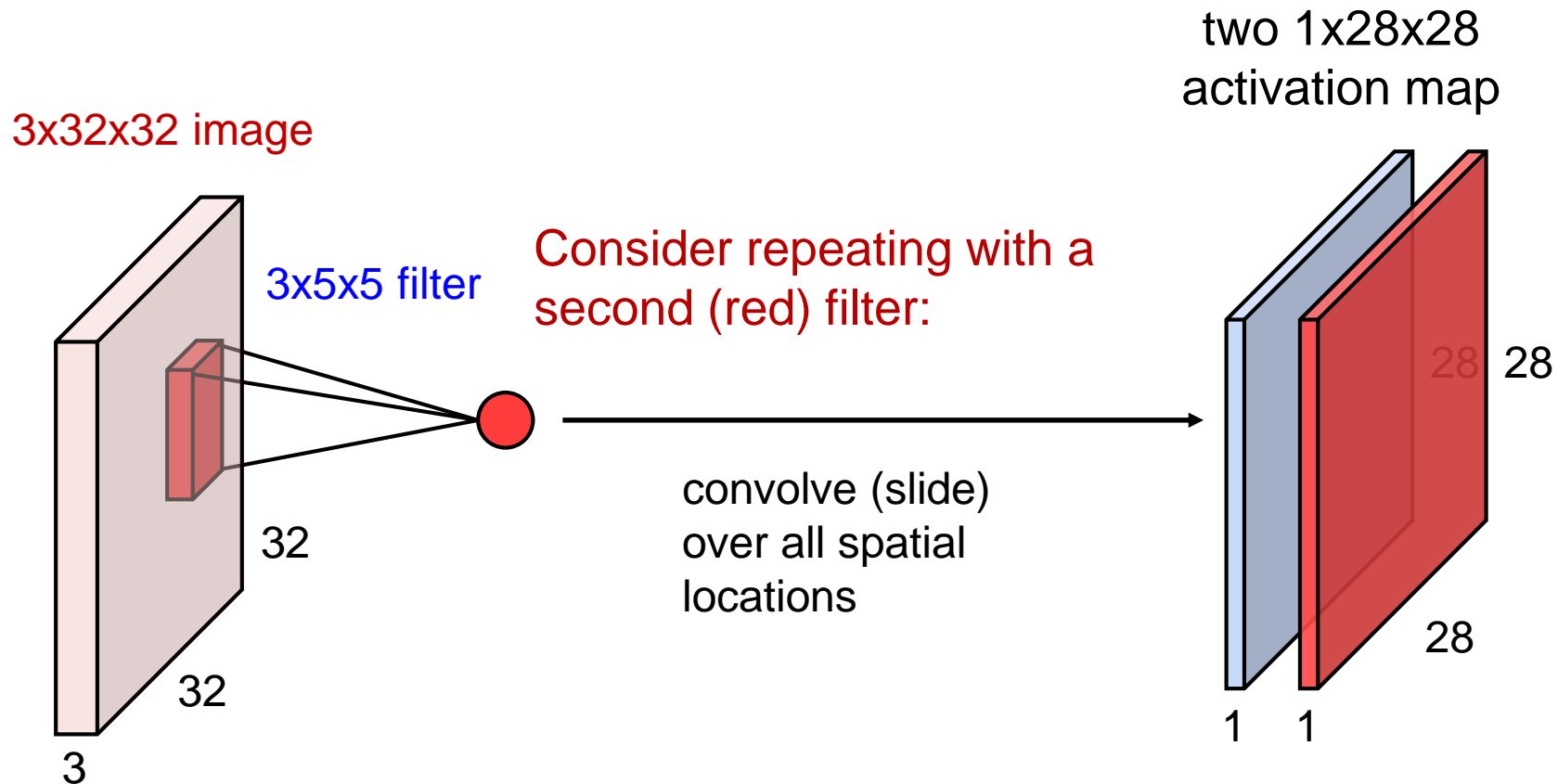
1 number :
the result of taking a dot product between the
filter and a small 3x5x5 chunk of the image

$$w^T x + b$$

Convolution Layer

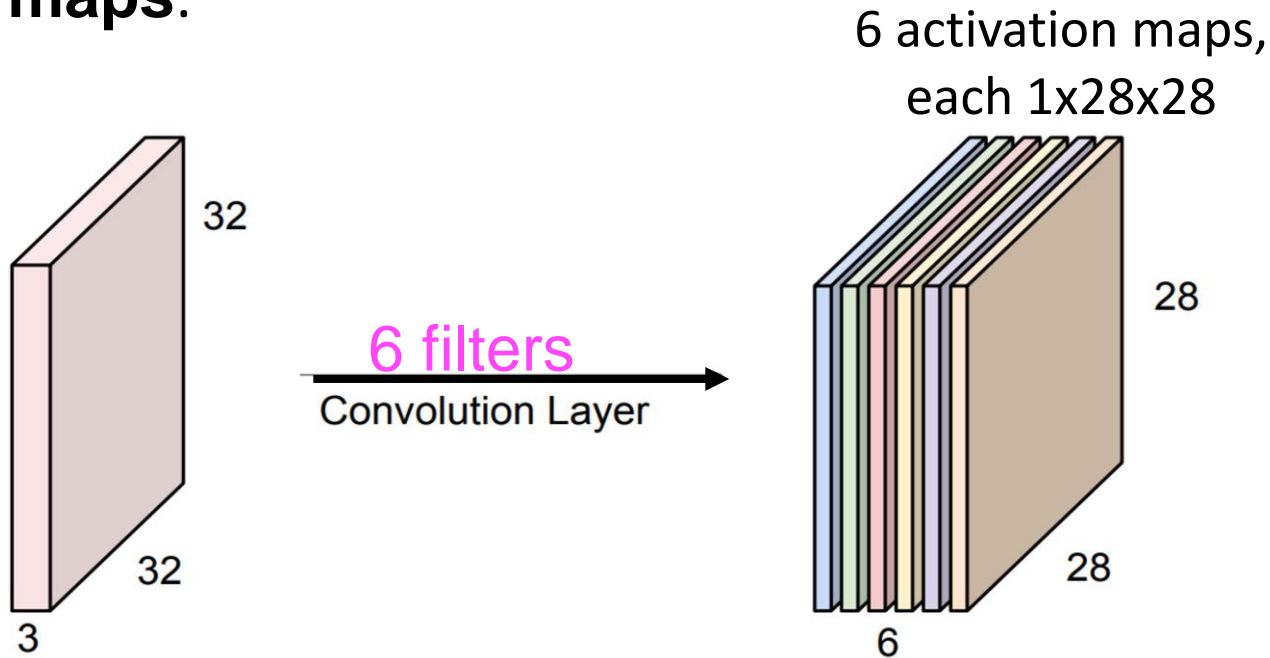


Convolutional layer



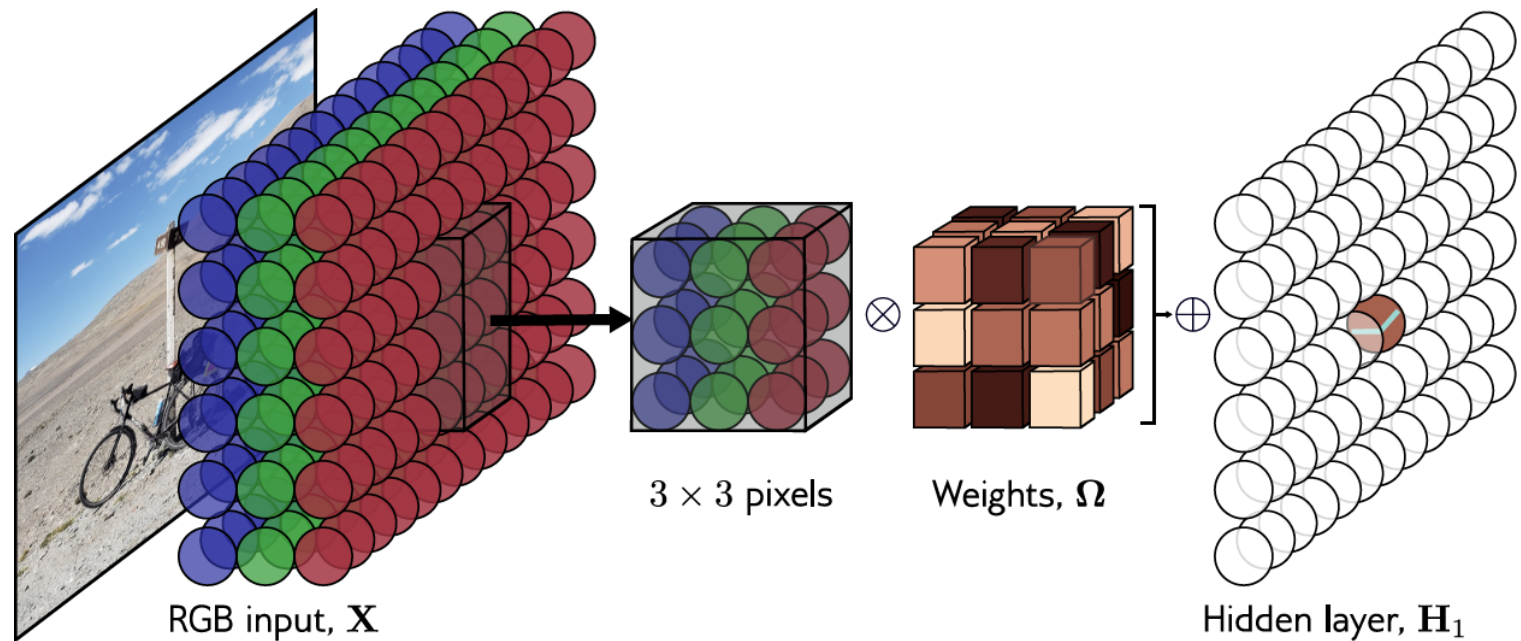
Multiple 3D Filters

- For example, if we had 6 $5 \times 5 \times 3$ filters, we would get 6 **separate** feature maps:



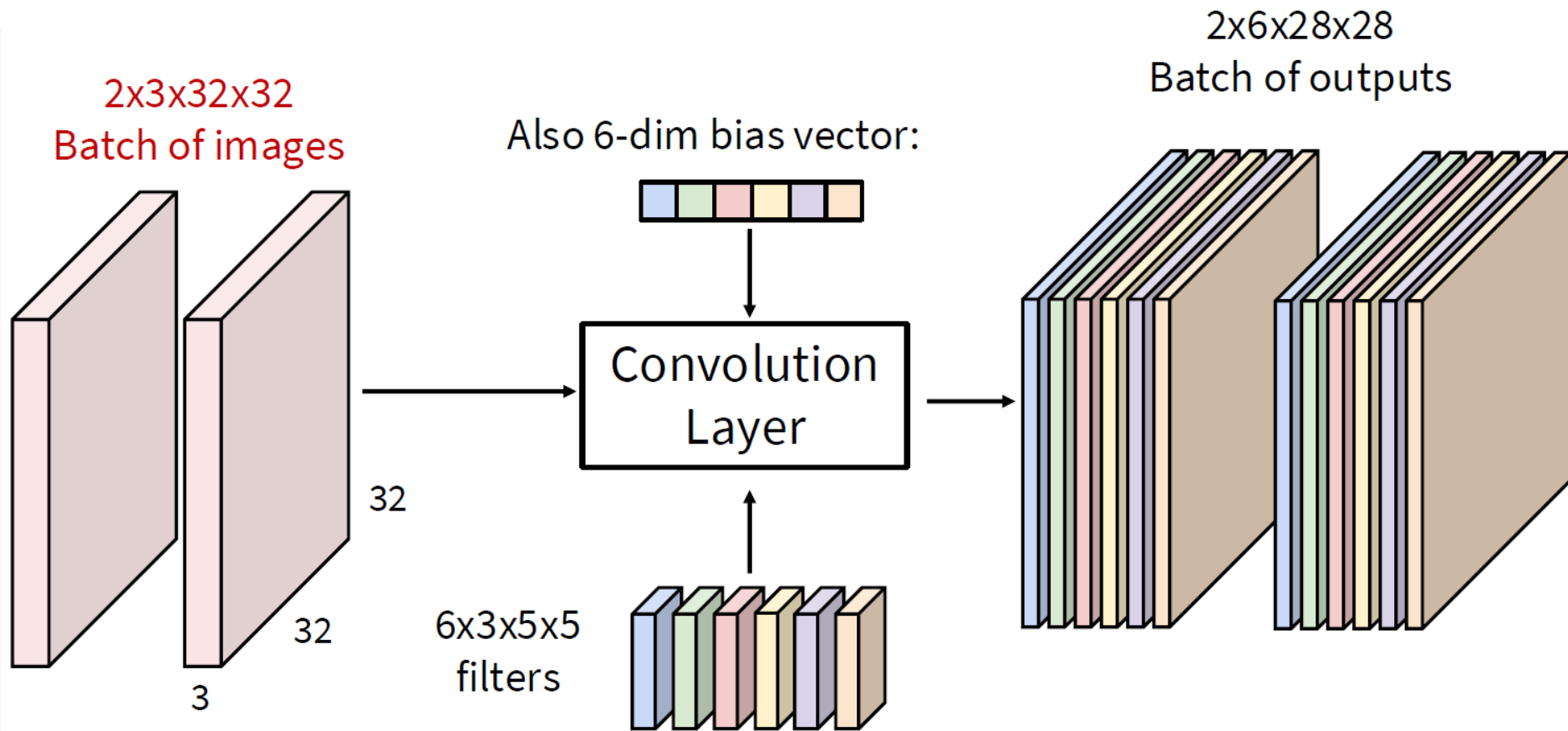
We stack these up to get 6 feature maps of size $28 \times 28 \times 6$ as output of this convolutional layer

Channels in 2D convolution

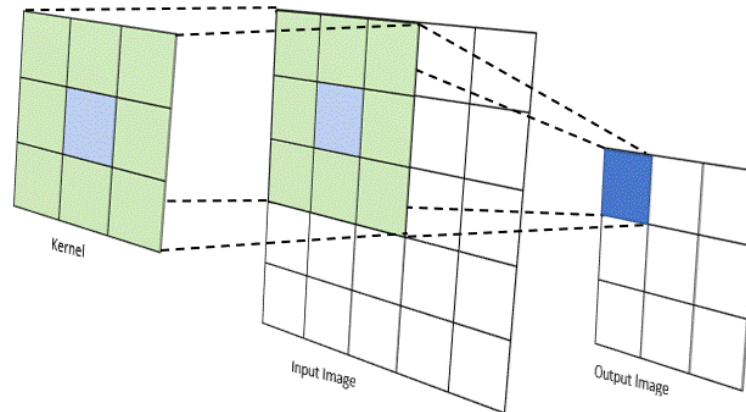


**Kernel size, stride, dilation
all work as you would
expect**

Convolution Layer



Convolutional layer



Input

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Filter / Kernel

1	0	1
0	1	0
1	0	1

x

=

Feature map

4		

Convolution Operation

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved
Feature

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image

4	3	

Convolved
Feature

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image

4	3	4

Convolved
Feature

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image

4	3	4
2		

Convolved
Feature

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image

4	3	4
2	4	

Convolved
Feature

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image

4	3	4
2	4	3

Convolved
Feature

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image

4	3	4
2	4	3
2		

Convolved
Feature

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image

4	3	4
2	4	3
2	3	

Convolved
Feature

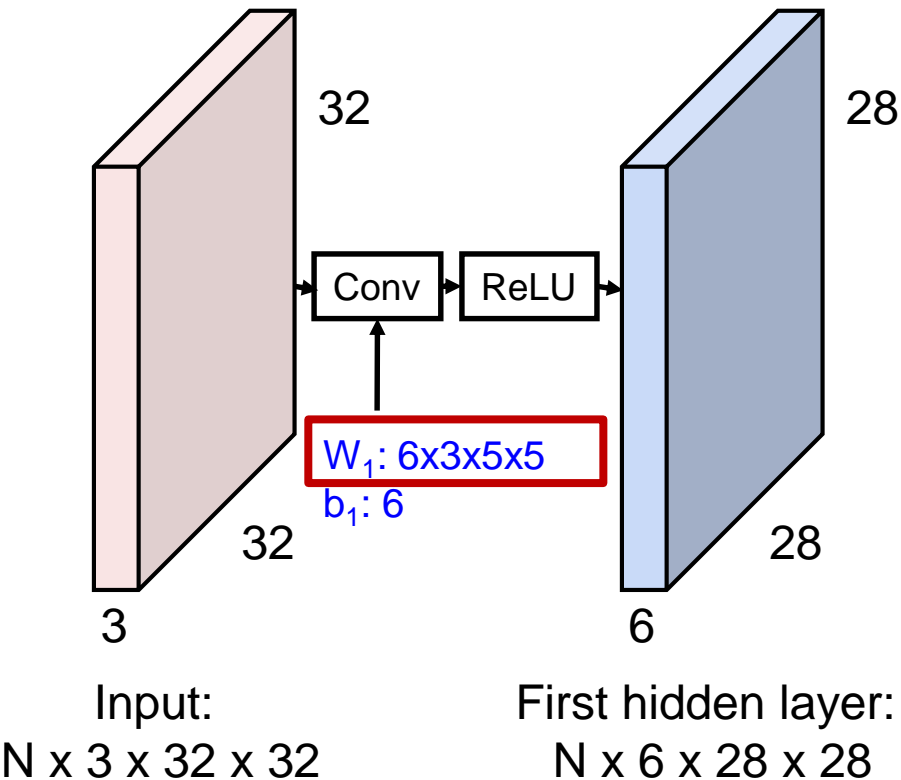
1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image

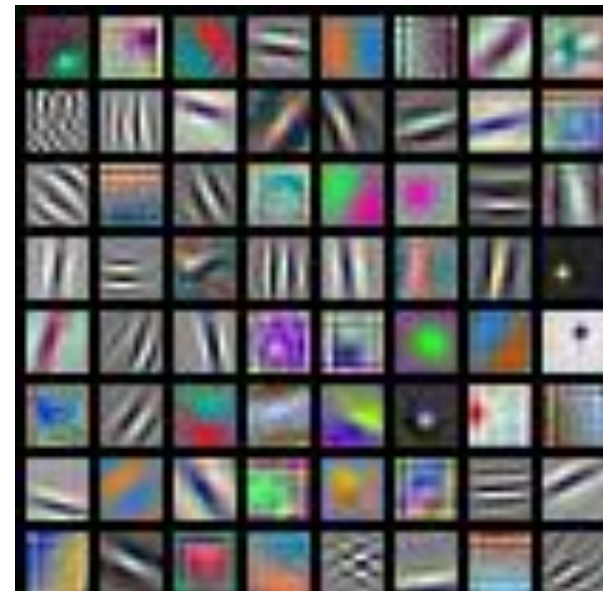
4	3	4
2	4	3
2	3	4

Convolved
Feature

What do convolutional filters learn?



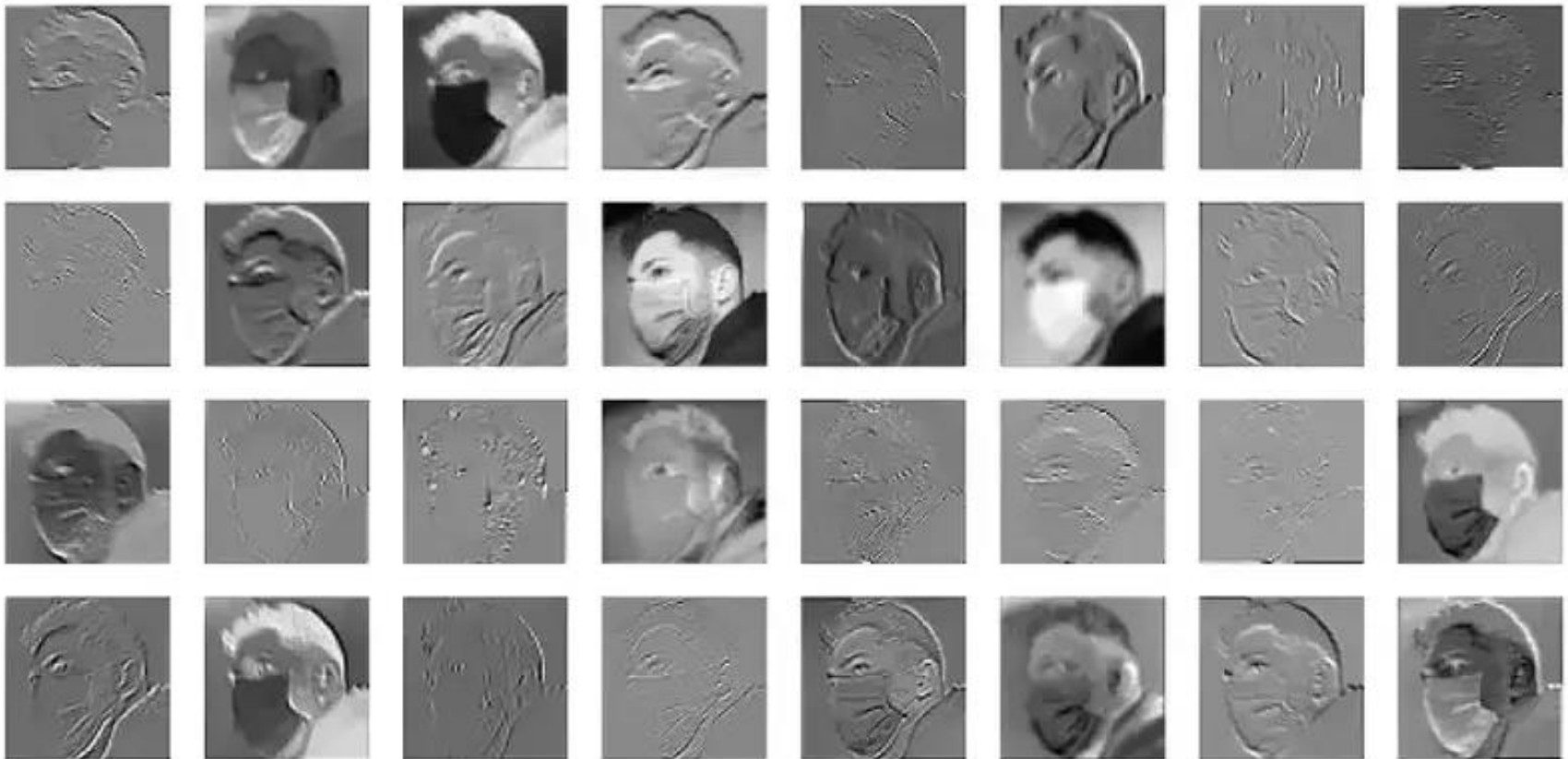
First-layer conv filters: local image templates
(Often learns oriented edges, opposing colors)



AlexNet: 64 filters, each $3 \times 11 \times 11$

Convolution filters

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



Filters



$$\begin{matrix} * & \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} & = \end{matrix}$$



blurs the image



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

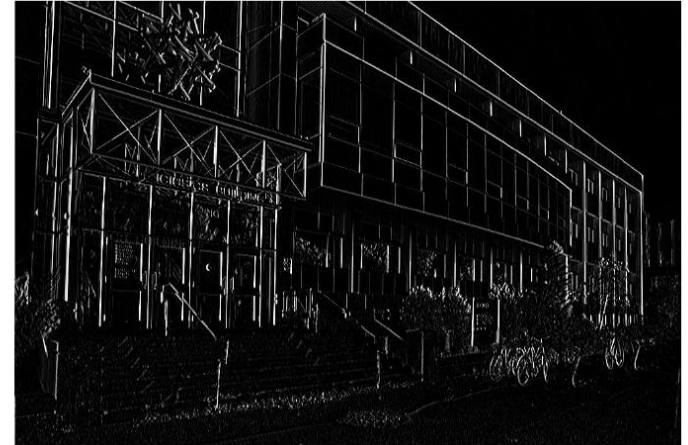


sharpens the image

Filters



$$\star \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \longrightarrow$$



10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

Input Image

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

Kernel

=

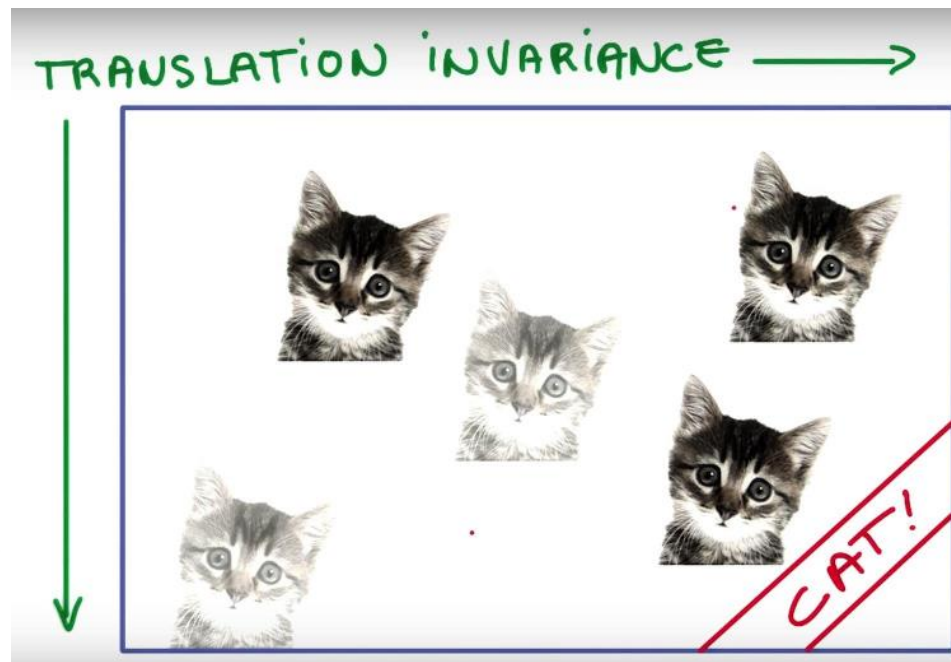
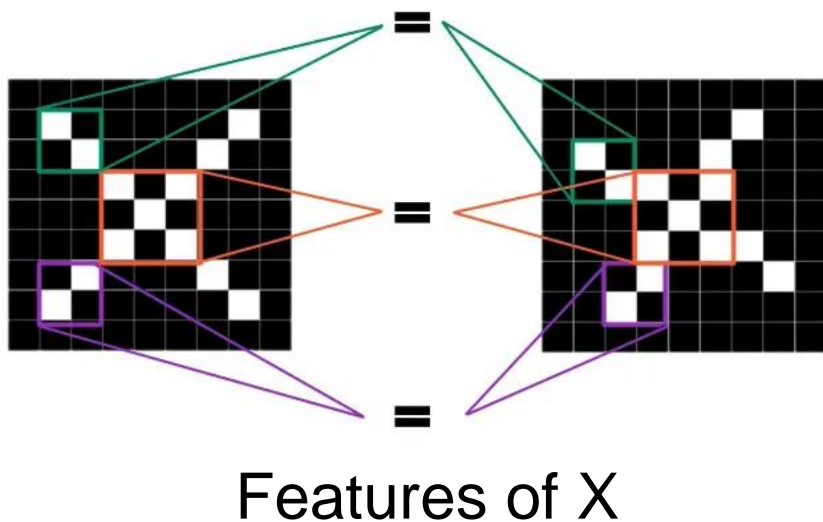
0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

Output
Feature Map

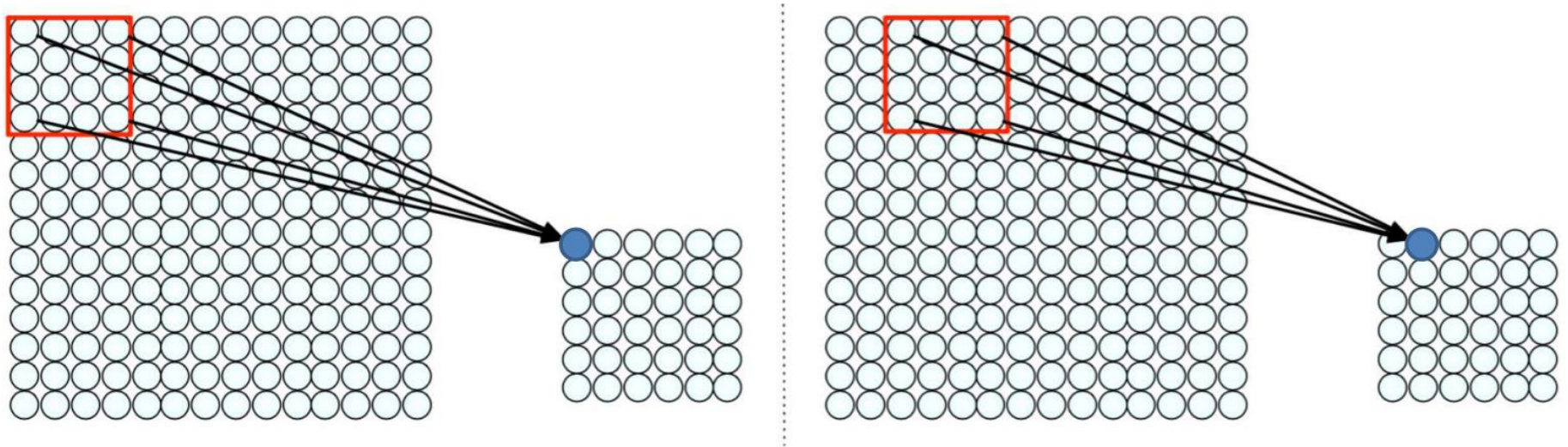
Vertical Edge Detection

Translation invariance

When input is changed spatially (translated or shifted), the corresponding output to recognize the object should **not be changed**



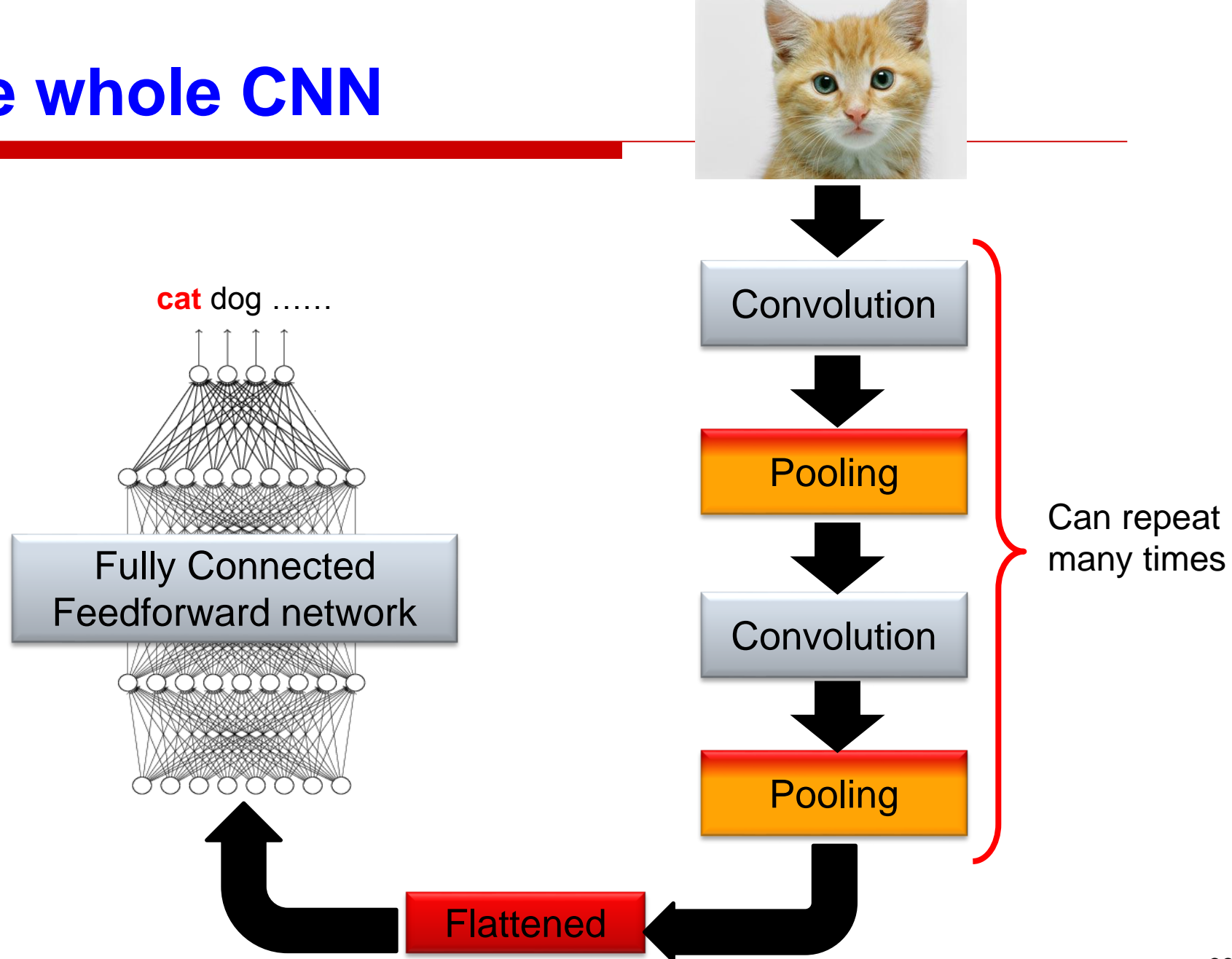
Using Spatial Structure/Information



2) Slide the patch window across the image.

*Different **weights (filters)** detect different features*

The whole CNN



CNN – Main components

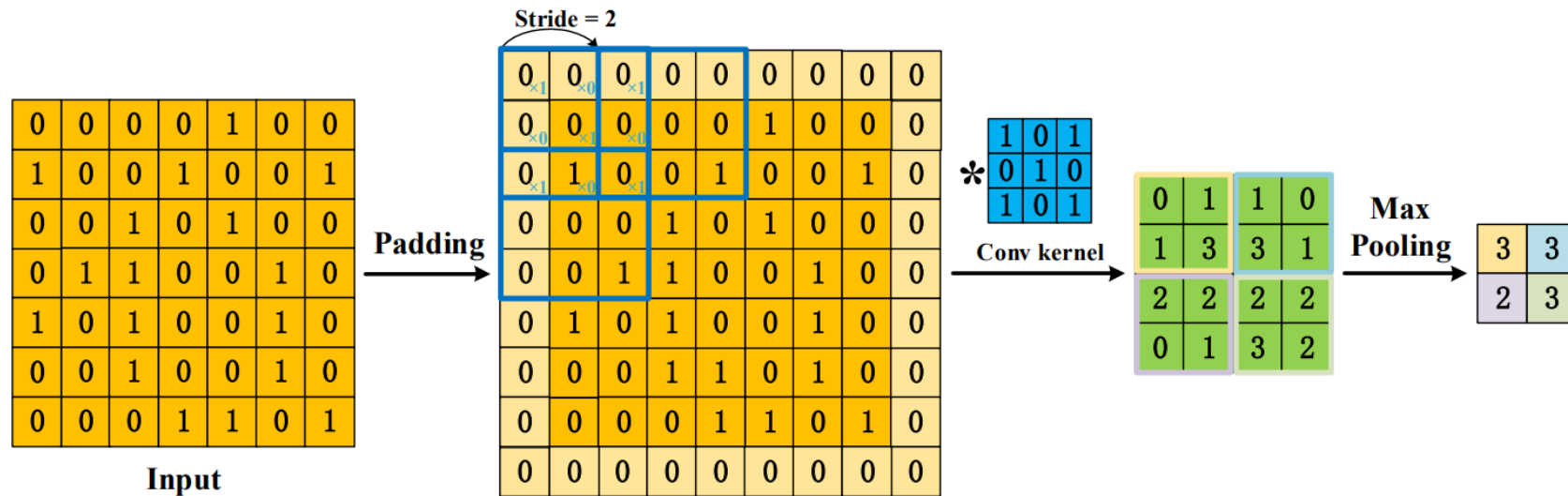
1. To build a CNN model, four components are typically needed (Li et al. 2020).

Convolution The outputs of convolution can be called feature maps.

Padding Padding enlarges the input with zero value.

Stride For controlling the density of convolving, stride used.

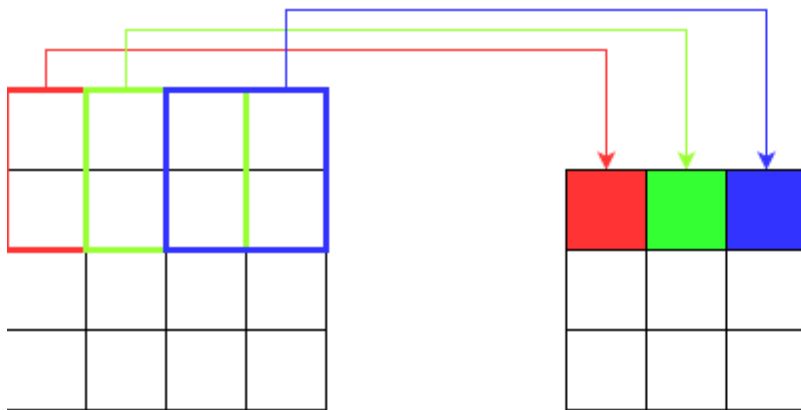
Pooling As a result, Pooling (down-sampling) such as max pooling and average pooling obviates large number of features in feature map.



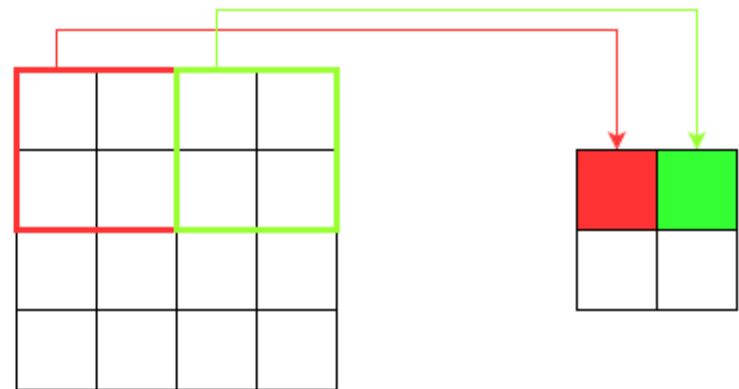
Strides

- Stride is the number of pixels shifts over the input matrix.
- When the stride is 1 then we move the filters to 1 pixel at a time. When the stride is 2 then we move the filters to 2 pixels at a time and so on.

Stride = 1

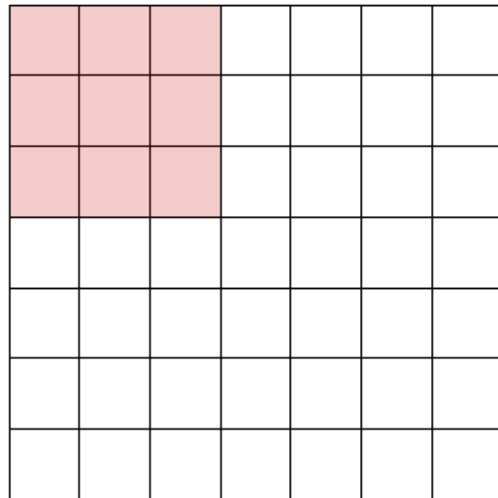


Stride = 2

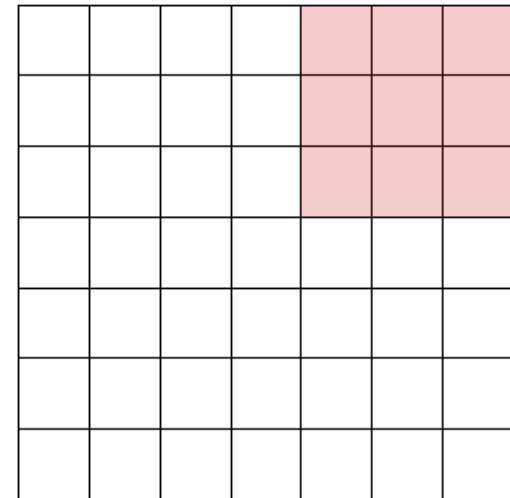
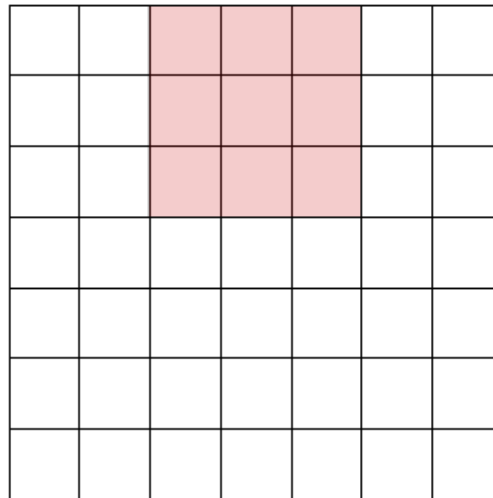


Strides

stride = 2



stride = 2



Padding

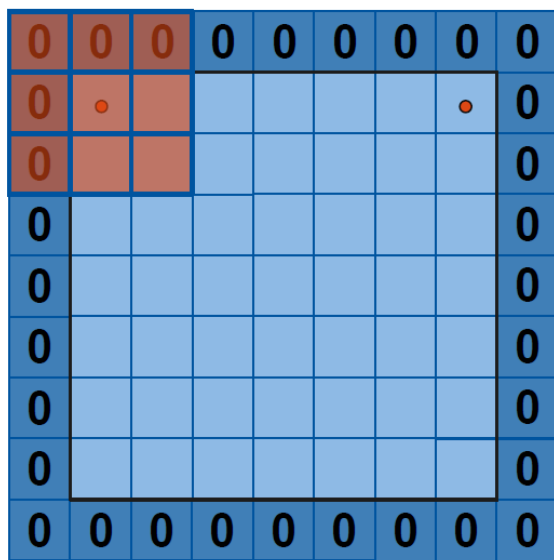
- Sometimes filter does not fit perfectly fit the input image. We have two options:
- Pad the picture with zeros (zero-padding) so that it fits
- Drop the part of the image where the filter did not fit. This is called valid padding which keeps only valid part of the image.

Padding

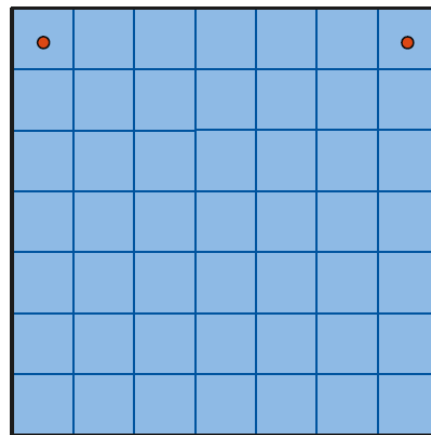
Add zeros around image borders to conserve the spatial extent of the input.

Prevents fast shrinking of the input data (image)

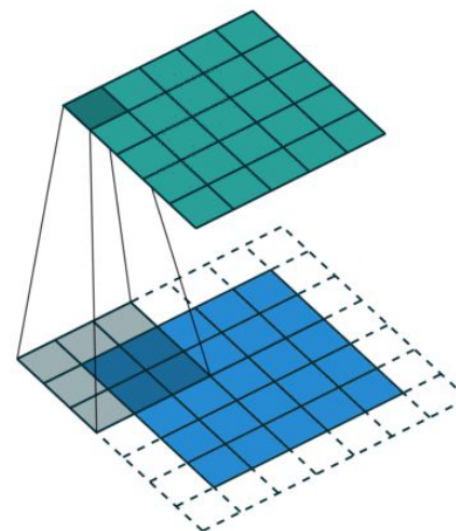
- **Example:** Convolution with 3 x 3 filter and padding



7



7



Padding

- In practice: Common to **zero pad** the border
 - Used to control the output filter size

9

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

7×7 input (spatially)
Zero pad 1 pixel border
Assume 3×3 filter
Applied with **stride 3**

→ **3×3 output**

9

Size of the Output

If you have a stride of 1 and if you set the size of zero padding to

$$\text{Zero Padding} = \frac{(K - 1)}{2}$$

where K is the filter size, then the input and output volume will always have the same spatial dimensions.

The formula for calculating the output size for any given conv layer is

$$O = \frac{(W - K + 2P)}{S} + 1$$

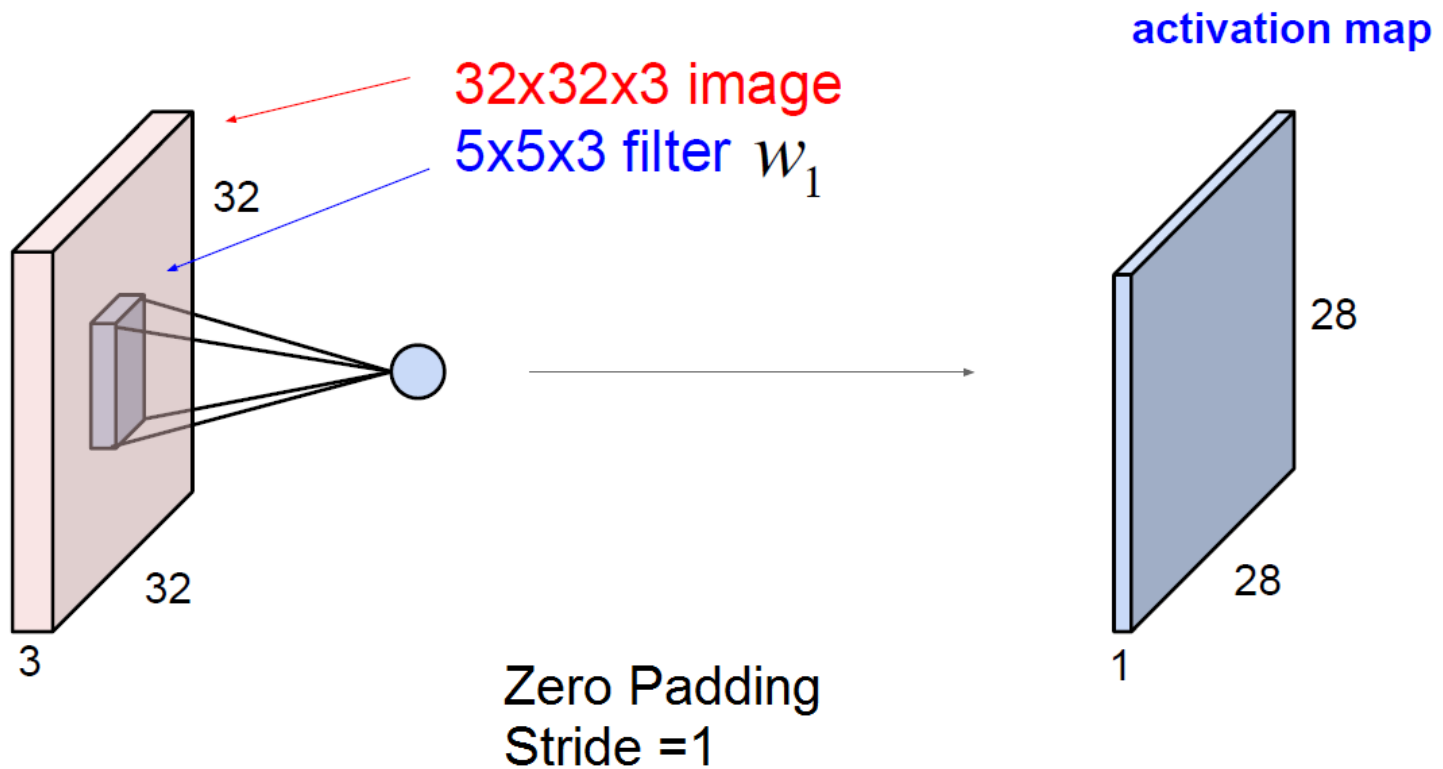
where O is the output height/length, W is the input height/length, K is the filter size, P is the padding, and S is the stride.



Size of the Output

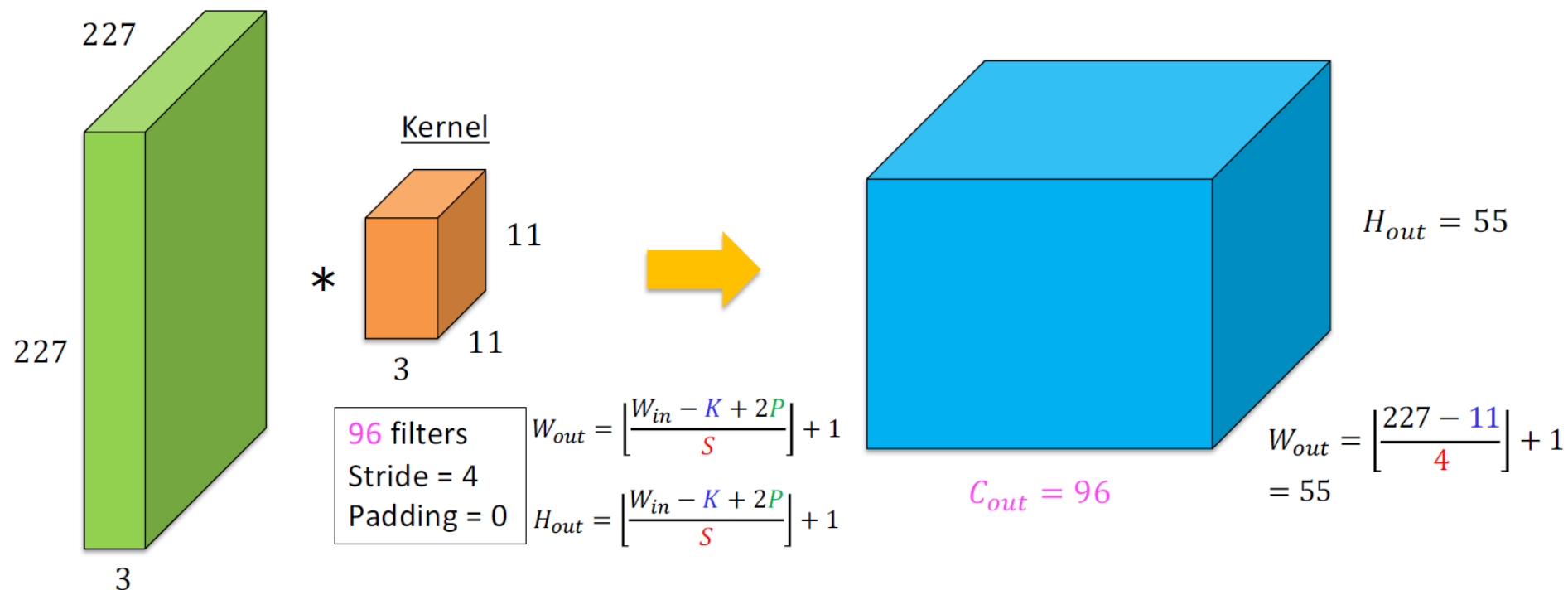
$$O = \frac{(W - K + 2P)}{S} + 1$$

$$\frac{32 - 5 + 2 \cdot 0}{1} + 1 = 28$$



Size of the Output

Work out output dimensions for the following setting:



Convolution with PyTorch

```
self.conv1 = nn.Conv2d(1, 16, kernel_size=5, stride=1,  
padding=0)
```

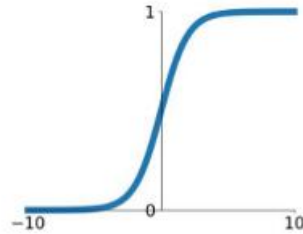
```
self.conv2 = nn.Conv2d(16, 8, kernel_size=5, stride=1,  
padding=0)
```



Non-linearity

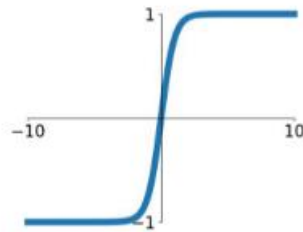
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



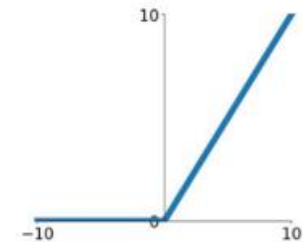
tanh

$$\tanh(x)$$



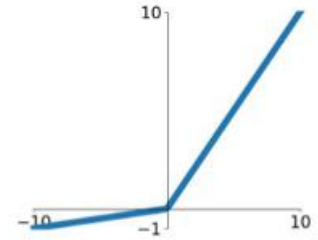
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

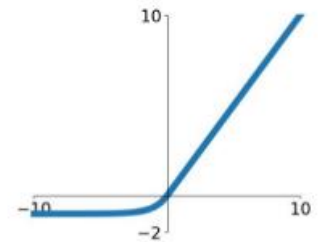


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$




Non Linearity (ReLU)

- ReLU stands for Rectified Linear Unit for a non-linear operation. The output is $f(x) = \max(0, x)$.
- Why ReLU is important?
- ReLU's purpose is to introduce non-linearity in our ConvNet. Since, the real world data would want our ConvNet to learn would be non-negative linear values.
- There are other non linear functions such as tanh or sigmoid can also be used instead of ReLU.
- Most of the data scientists uses ReLU since performance wise ReLU is better than other two.

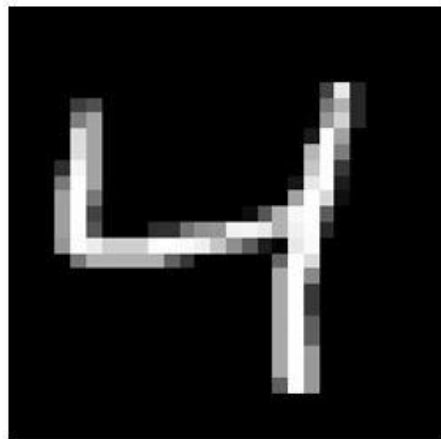


Non Linearity (ReLU)

1	14	-9	4
-2	-20	10	6
-3	3	11	1
2	54	-2	80



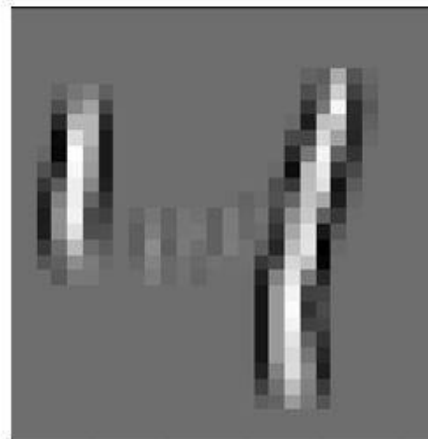
1	14	0	4
0	0	10	6
0	3	11	1
2	54	0	80



Image

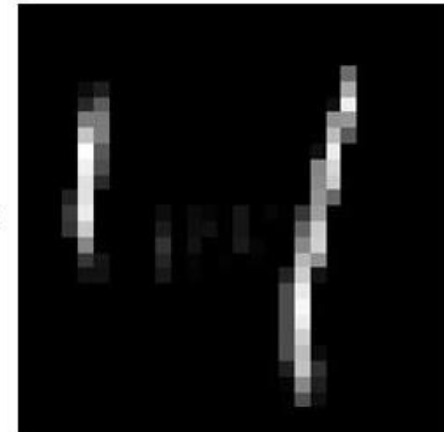
*  =

Kernel



Output

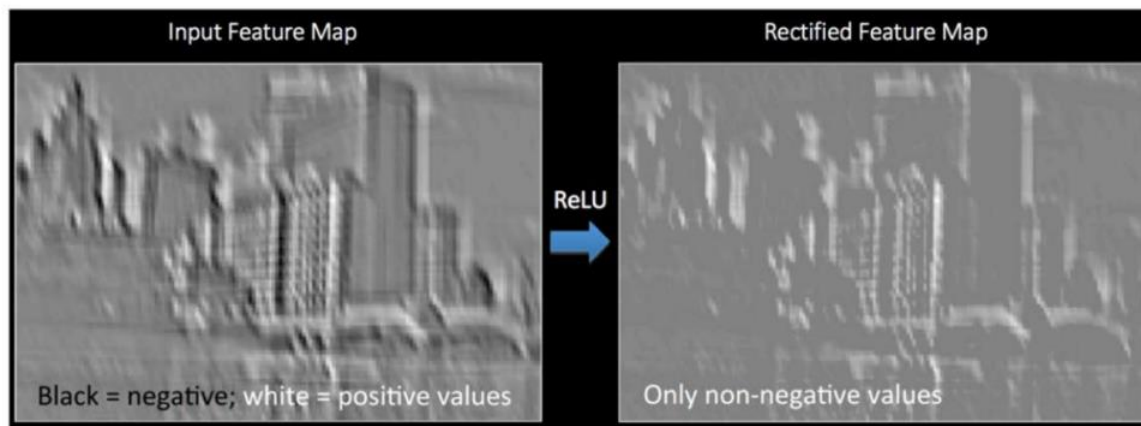
 ReLU =



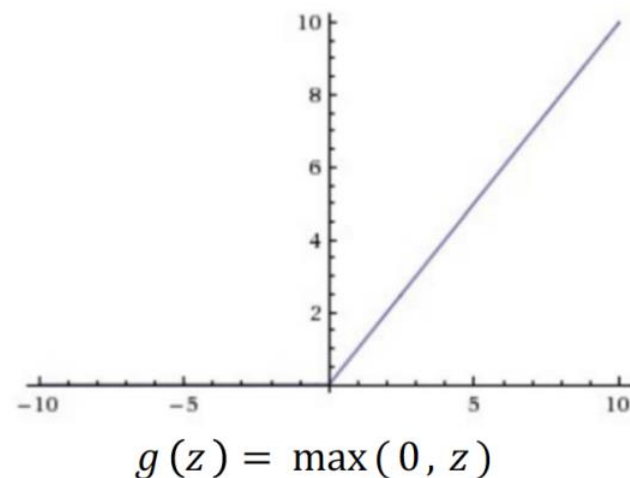
Output

Non Linearity (ReLU)

- Apply after every convolution operation (i.e., after convolutional layers)
- ReLU: pixel-by-pixel operation that replaces all negative values by zero. **Non-linear operation!**



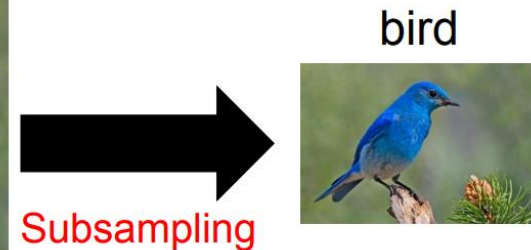
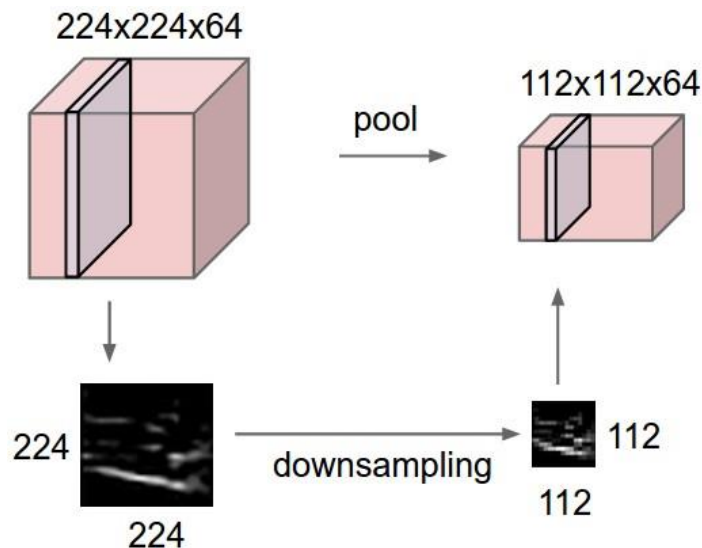
Rectified Linear Unit (ReLU)



Pooling Layers

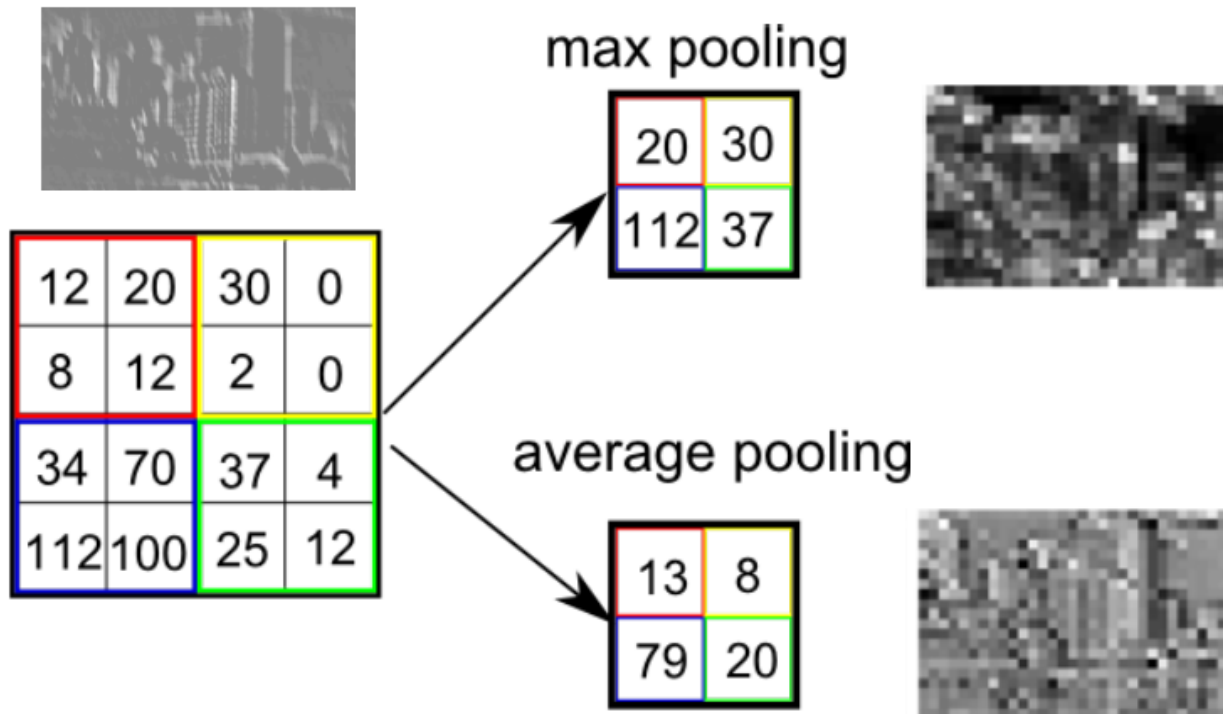
Pooling (or subsampling)

- Make the representations smaller (**will not change the object**)
- (+) Reduce number of parameters and computation

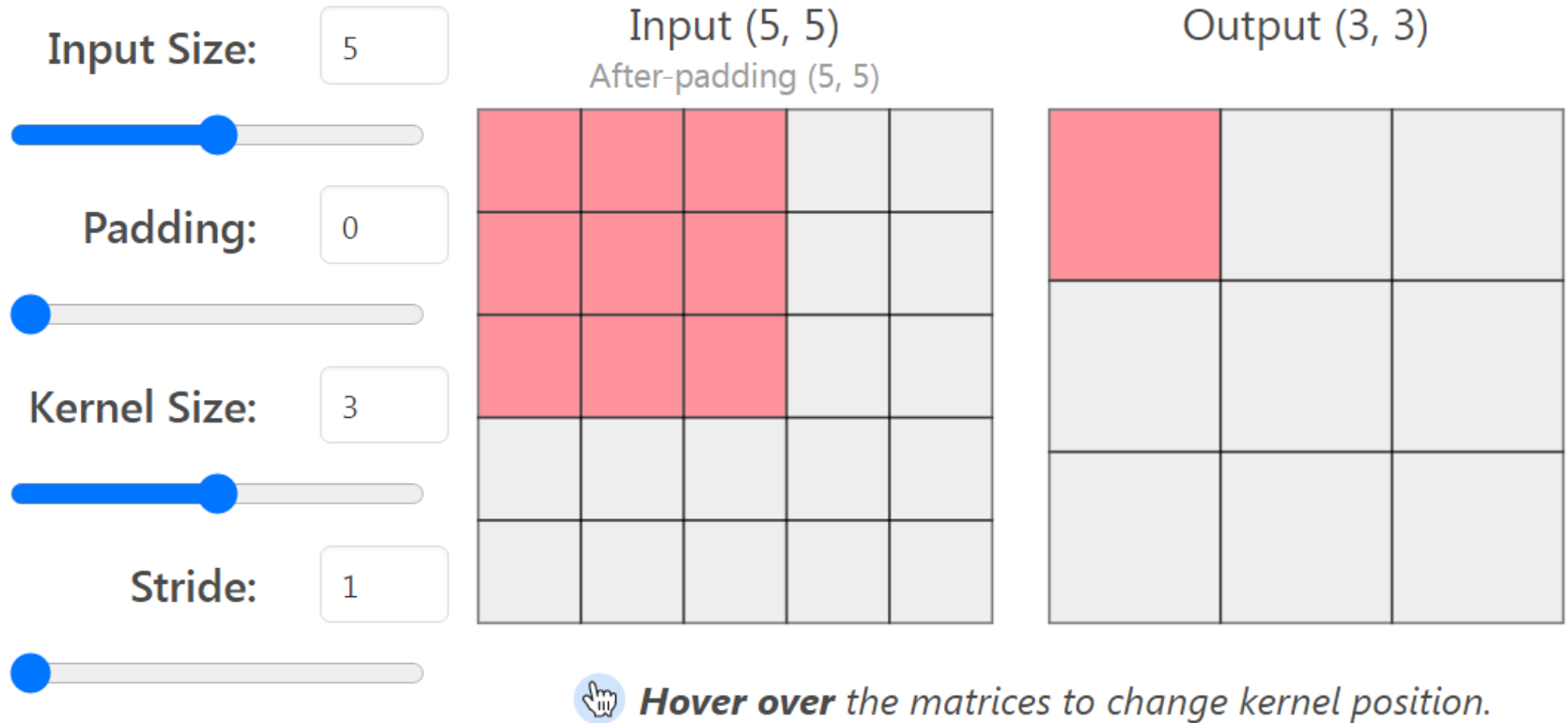


Pooling Layer - Downsampling

- Max pooling and average pooling
 - With 2×2 filters and stride 2

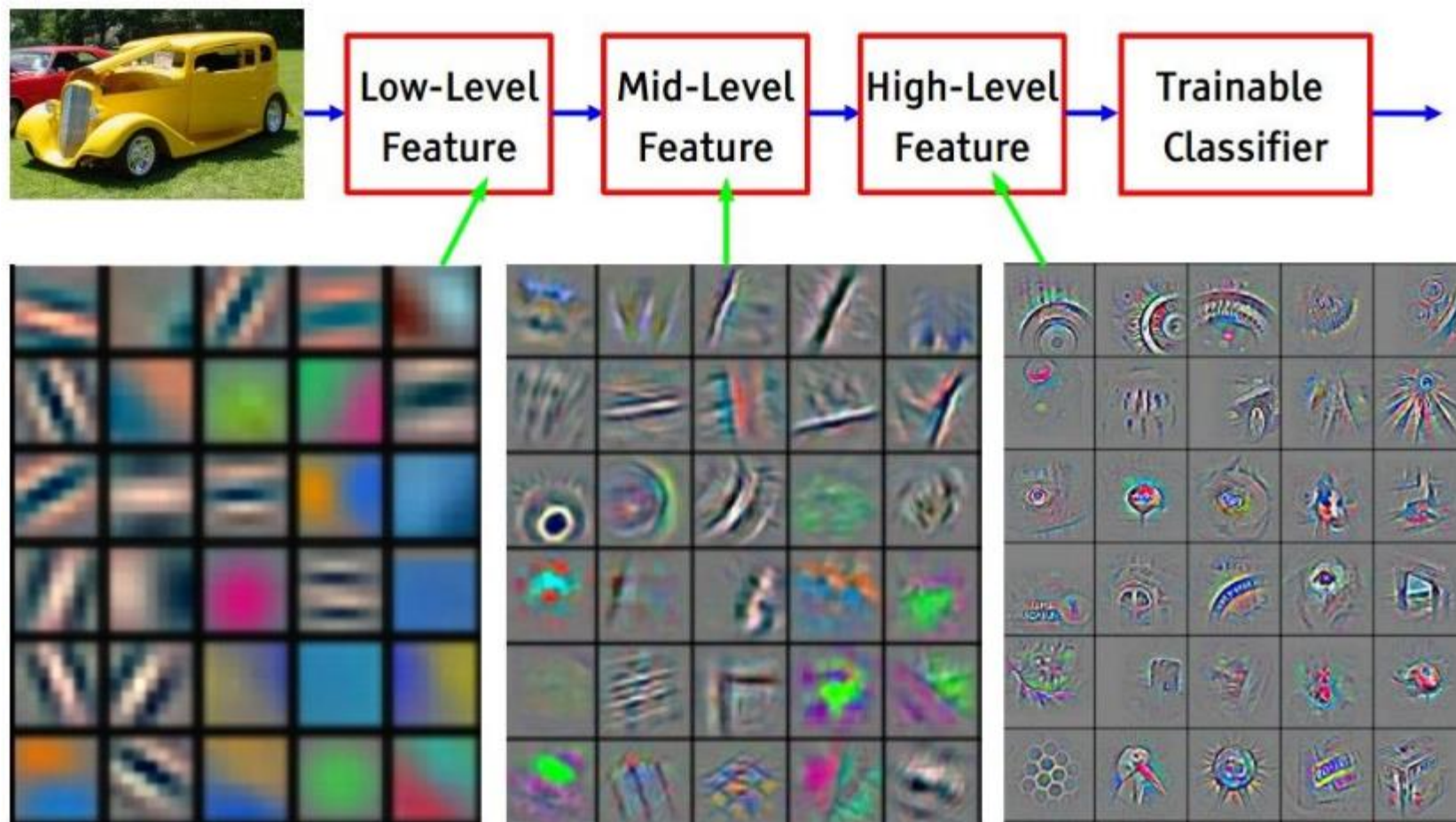


Understanding Hyper-parameters



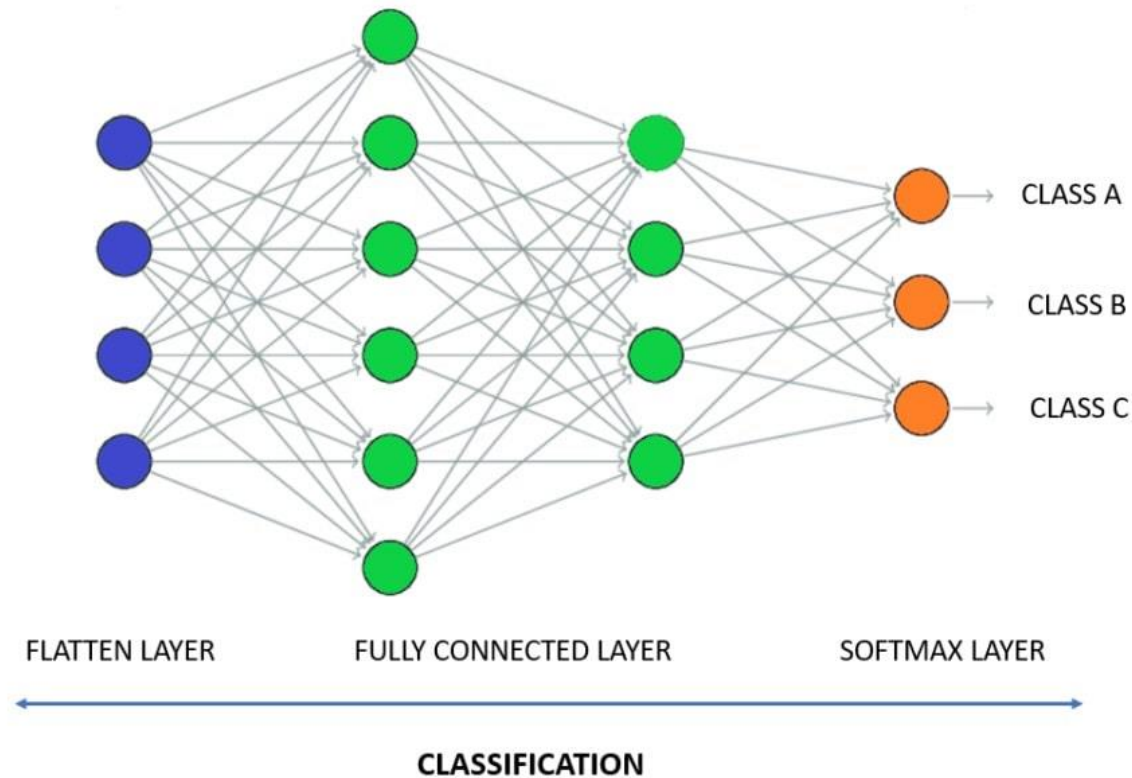
<https://poloclub.github.io/cnn-explainer/>

Visualization of CNNs layers



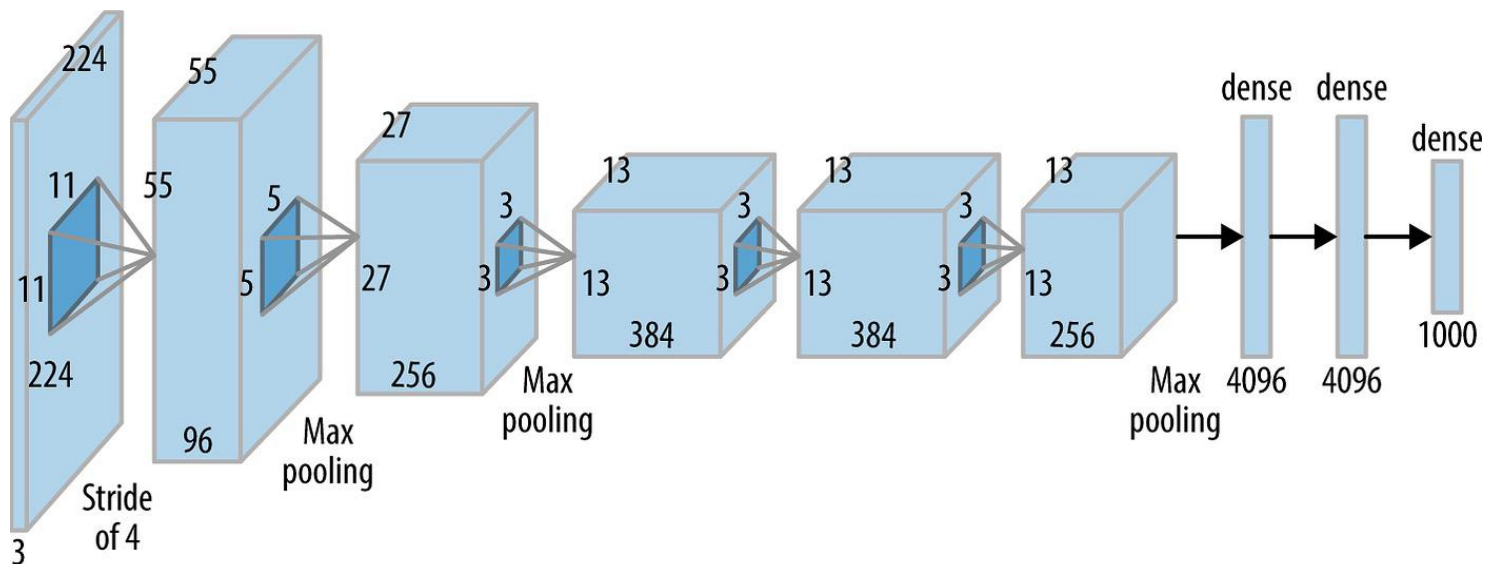
Fully Connected Layer

The layer we call as FC layer, we flattened our matrix into vector and feed it into a fully connected layer like neural network.

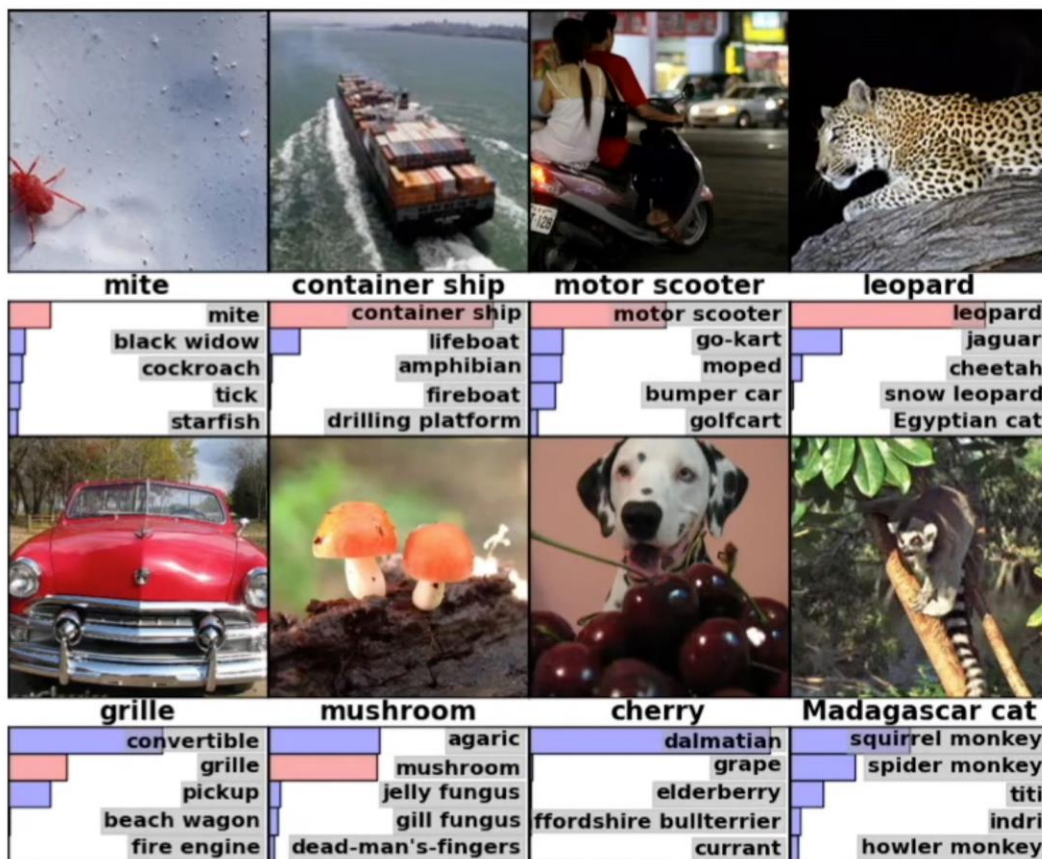


AlexNet Architecture

- **8 layers** (5 conv layers and 3 fully connected layers)
- CONV1-CONV2-POOL-CONV3-POOL-CONV4-CONV5-POOL-FC-FC-FC
- The ReLU non-linearity is applied to the output of every layers
- Response normalization layer follow the first and second conv layers
- Overlapped 3x3 Max Pooling and Dropout in FC layers
- 60M Parameters



ImageNet Dataset (ILSVRC Challenge)



- About 1.2 million images and 1000 classes

- Image resolution: RGB-Color Images with 244x244x3

- Accuracy is measured as top-5 performance.

Correct prediction if the true label Matches one of the top 5 predictions of model

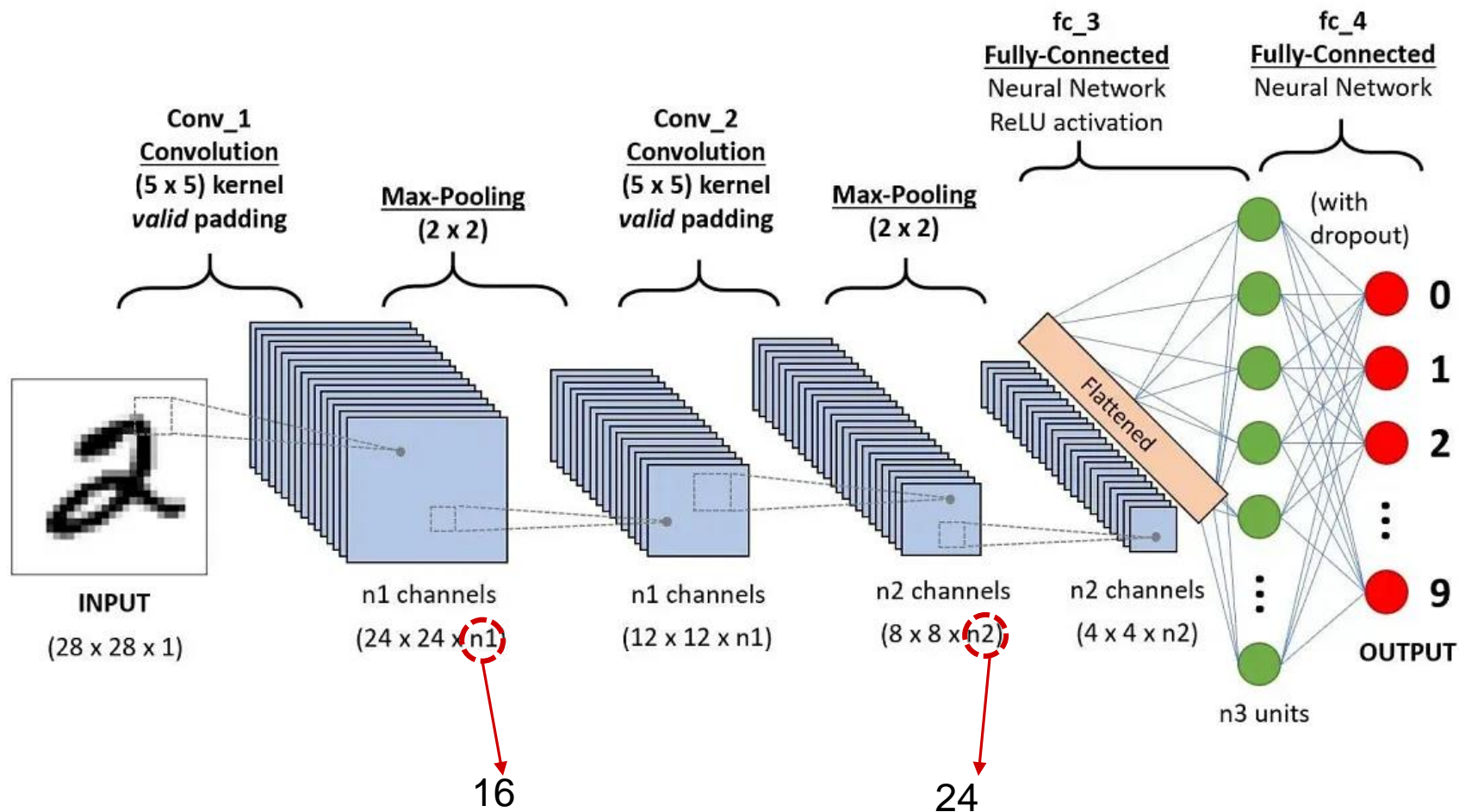
Complexity of AlexNet

- Convolutional layers cumulatively contain about 90-95% of computation, **only about 5% of the parameters**
- **Fully-connected layers contain about 95% parameters**
- Trained with SGD
 - On two NVIDIA GTX 580 3GB GPUs
 - For about 1 week

	Activation shape	Activation size	# parameters
Input image	227 x 227 x 3	154587	0
Conv 1	55 x 55 x 96 ($f=11$ $s=4$ $p=0$)	290400	34944
Pool 1	27 x 27 x 96 ($f=3$ $s=2$)	69984	0
Conv 2	27 x 27 x 256 ($f=5$ $s=1$ $p=2$)	186624	614,656
Pool 2	13 x 13 x 256 ($f=3$ $s=2$)	43264	0
Conv 3	13 x 13 x 384 ($f=3$ $s=1$ $p=1$)	64896	885,120
Conv 4	13 x 13 x 384 ($f=3$ $s=1$ $p=1$)	64896	1,327,488
Conv 5	13 x 13 x 256 ($f=3$ $s=1$ $p=1$)	43264	884,992
Pool 5	6 x 6 x 256 ($f=3$ $s=2$)	9216	0
FC 3	4096 x 1	4096	37,748,737
FC 4	4096 x 1	4096	16,777,217
Softmax	1000 x 1	1000	4096001



CNN for MIST Dataset



CNN for MIST Dataset

Dimension Flow:

Input: $1 \times 28 \times 28$ (grayscale MNIST images)

After conv1: $16 \times 24 \times 24$ ($28-5+1=24$)

After maxpool1: $16 \times 12 \times 12$ ($24/2=12$)

After conv2: $24 \times 8 \times 8$ ($12-5+1=8$)

After maxpool2: $24 \times 4 \times 4$ ($8/2=4$)

Flatten: $24 \times 4 \times 4 = 384$ features

fc1: $384 \rightarrow 64$

fc2: $64 \rightarrow 10$ (digit classes)

CNN for MIST Dataset

Convolutional Layers:

```
self.conv1 = nn.Conv2d(1, 16, kernel_size=(5, 5), stride=(1, 1))  
self.conv2 = nn.Conv2d(16, 24, kernel_size=(5, 5), stride=(1, 1))
```

Fully Connected Layers:

```
self.fc1 = nn.Linear(in_features=384, out_features=64, bias=True)  
self.fc2 = nn.Linear(in_features=64, out_features=10, bias=True)
```

Regularization

```
self.dropout = nn.Dropout(p=0.5, inplace=False)
```



A clear blue sky with several fluffy white clouds. The clouds are scattered, with a larger cluster in the upper center and a few smaller ones towards the bottom left and right.

Questions