Convolutional Neural Networks

Dr. Unchalisa Taetragool

Department of Computer Engineering, Faculty of Engineering
King Mongkut's University of Technology Thonburi

Convolutional Neural Network

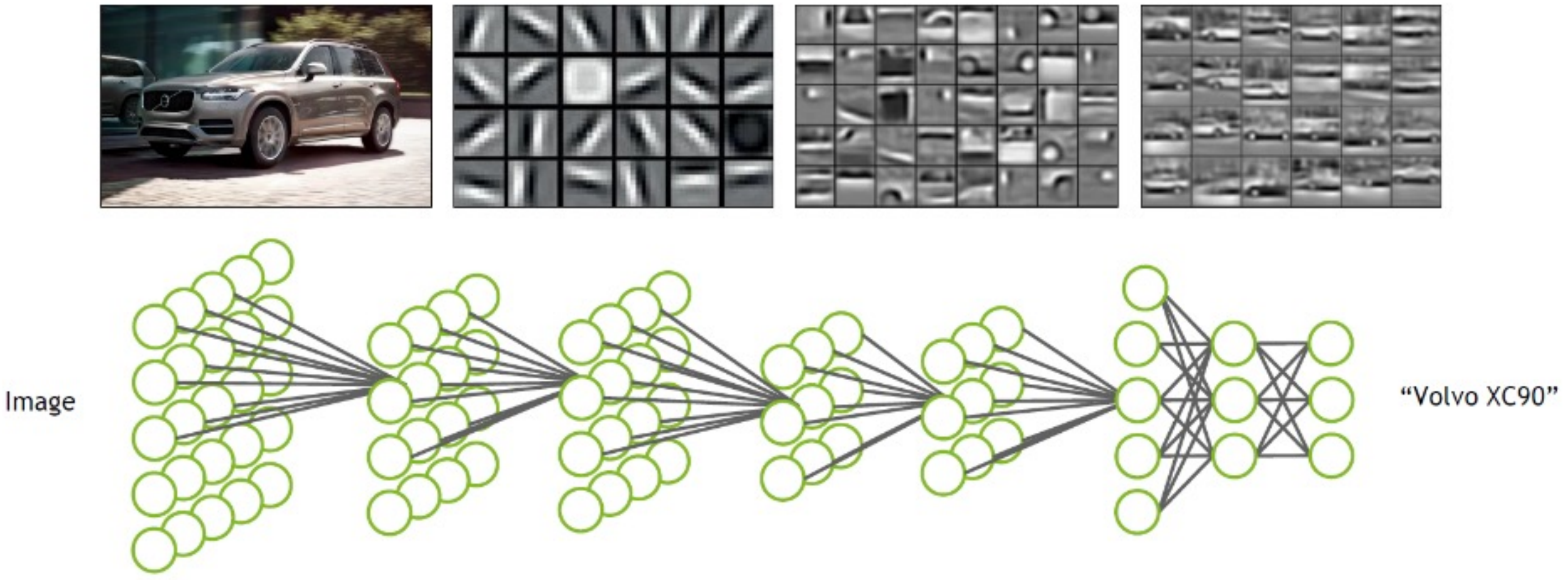


Image source: "Unsupervised Learning of Hierarchical Representations with Convolutional Deep Belief Networks" ICML 2009 & Comm. ACM 2011. Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Ng.

Image Recognition: Simple Example

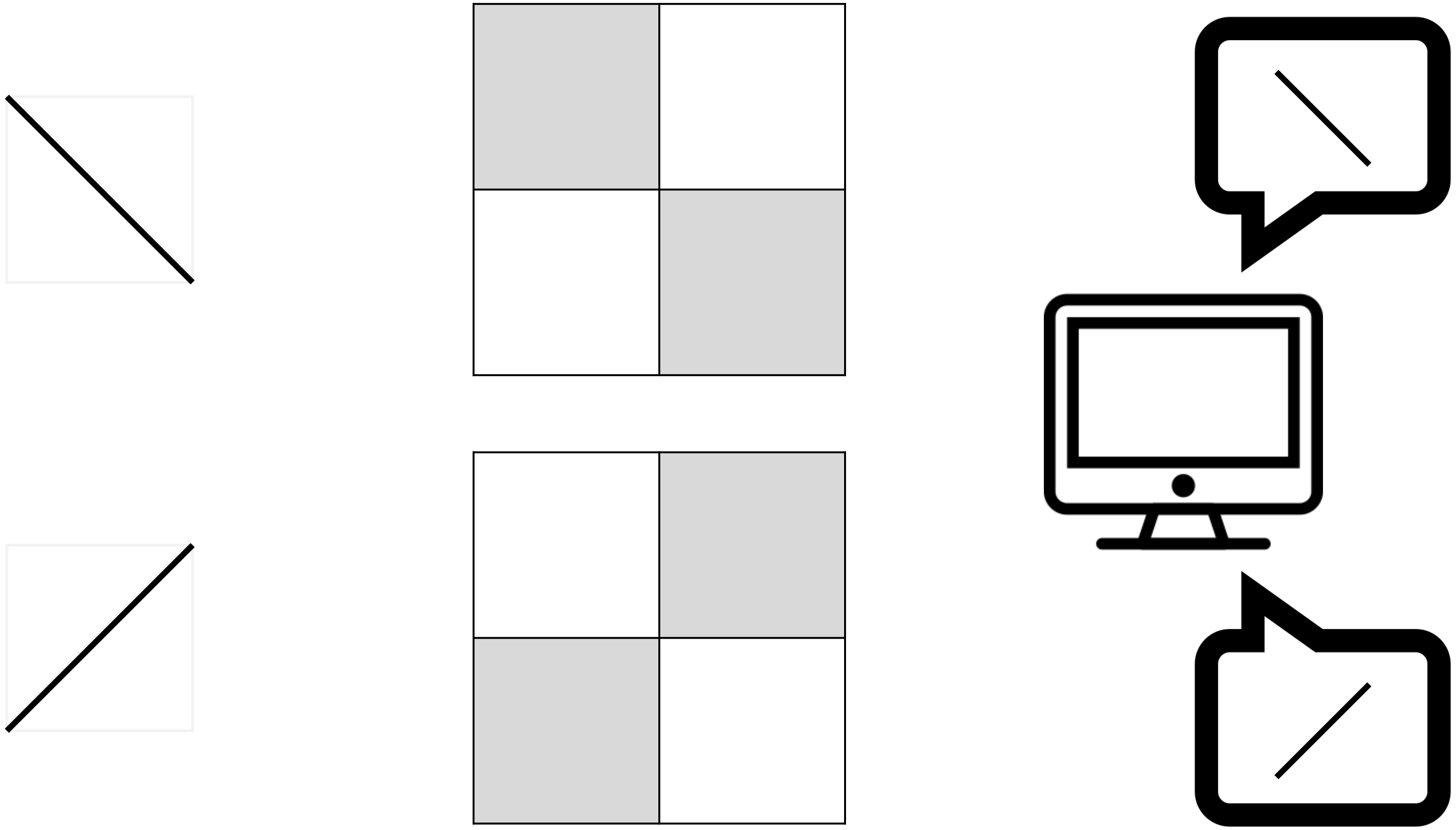
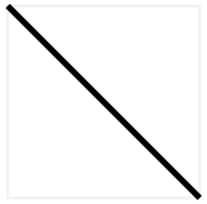
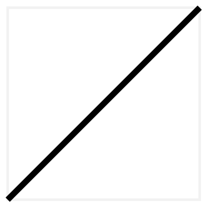


Image Recognition: Simple Example



1	-1
-1	1

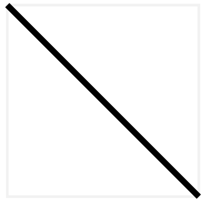
1	-1	-1	1



-1	1
1	-1

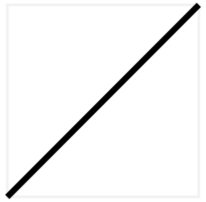
-1	1	1	-1

Image Recognition: Simple Example



+	+	+	+
1	-1	-1	1
+1	-1	-1	+1

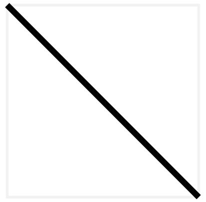
= 0



+	+	+	+
-1	1	1	-1
-1	1	1	-1

= 0

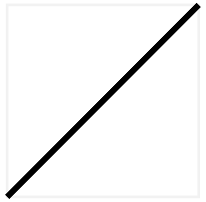
Image Recognition: Simple Example



1	-1
-1	1

1	-1	-1	1

TRY



-1	1
1	-1

-1	1	1	-1

How can a computer find the filters?

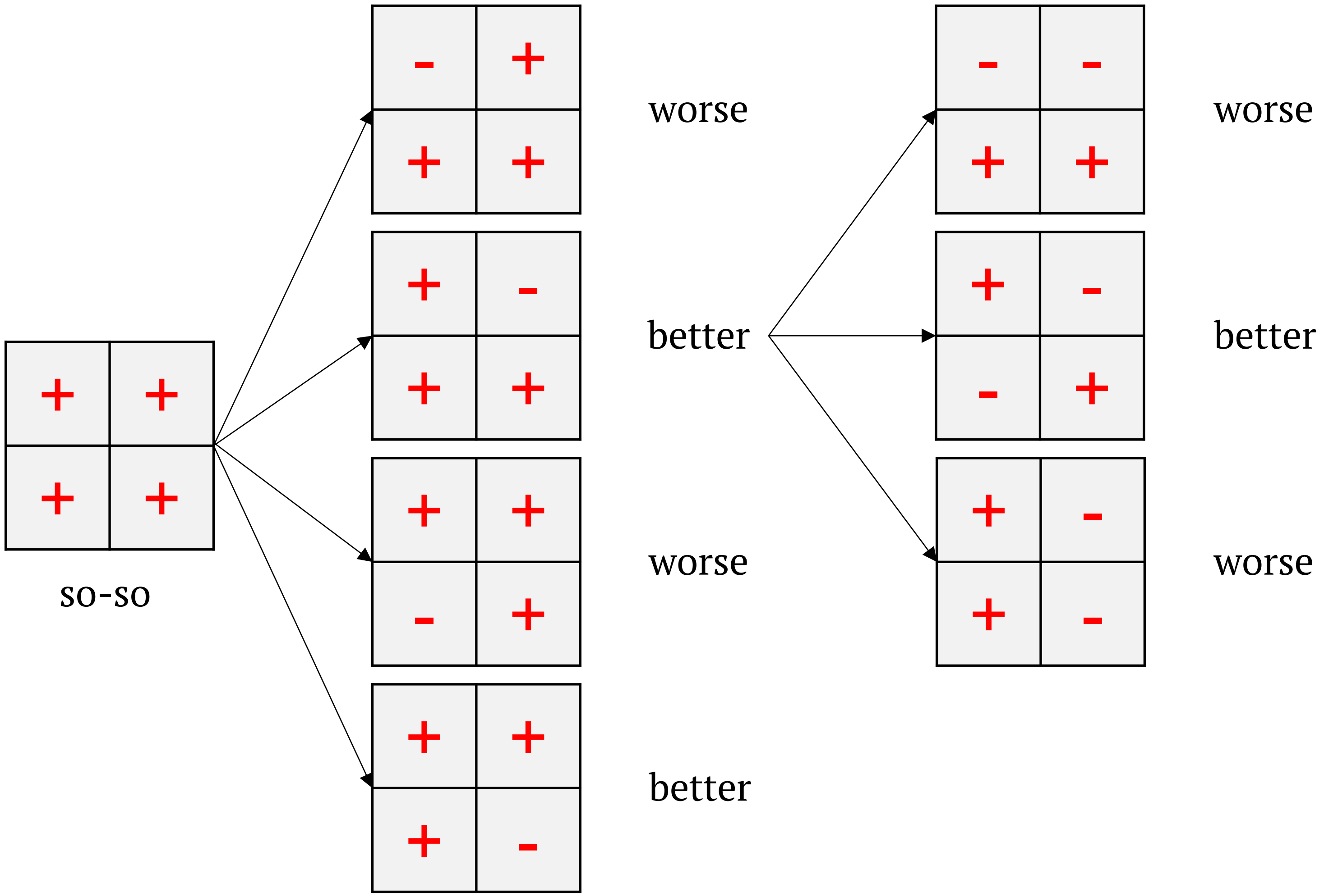
$$
\begin{bmatrix} + & + \\ + & + \end{bmatrix}
\quad
\begin{bmatrix} - & + \\ + & + \end{bmatrix}
\quad
\begin{bmatrix} + & - \\ + & + \end{bmatrix}
\quad
\begin{bmatrix} + & + \\ - & + \end{bmatrix}
$$

$$
\begin{bmatrix} + & + \\ + & - \end{bmatrix}
\quad
\begin{bmatrix} - & - \\ + & + \end{bmatrix}
\quad
\begin{bmatrix} - & + \\ - & + \end{bmatrix}
\quad
\begin{bmatrix} + & + \\ - & - \end{bmatrix}
$$

and more …

Find the best filters from the 16 choices

How can a computer find the filters?



How can a computer find the filters?

0.5	1.2
0.7	1.0

0.6	0.9
0.5	1.1

0.9	-0.7
-0.6	1.1

and so on...

Gradient Descent

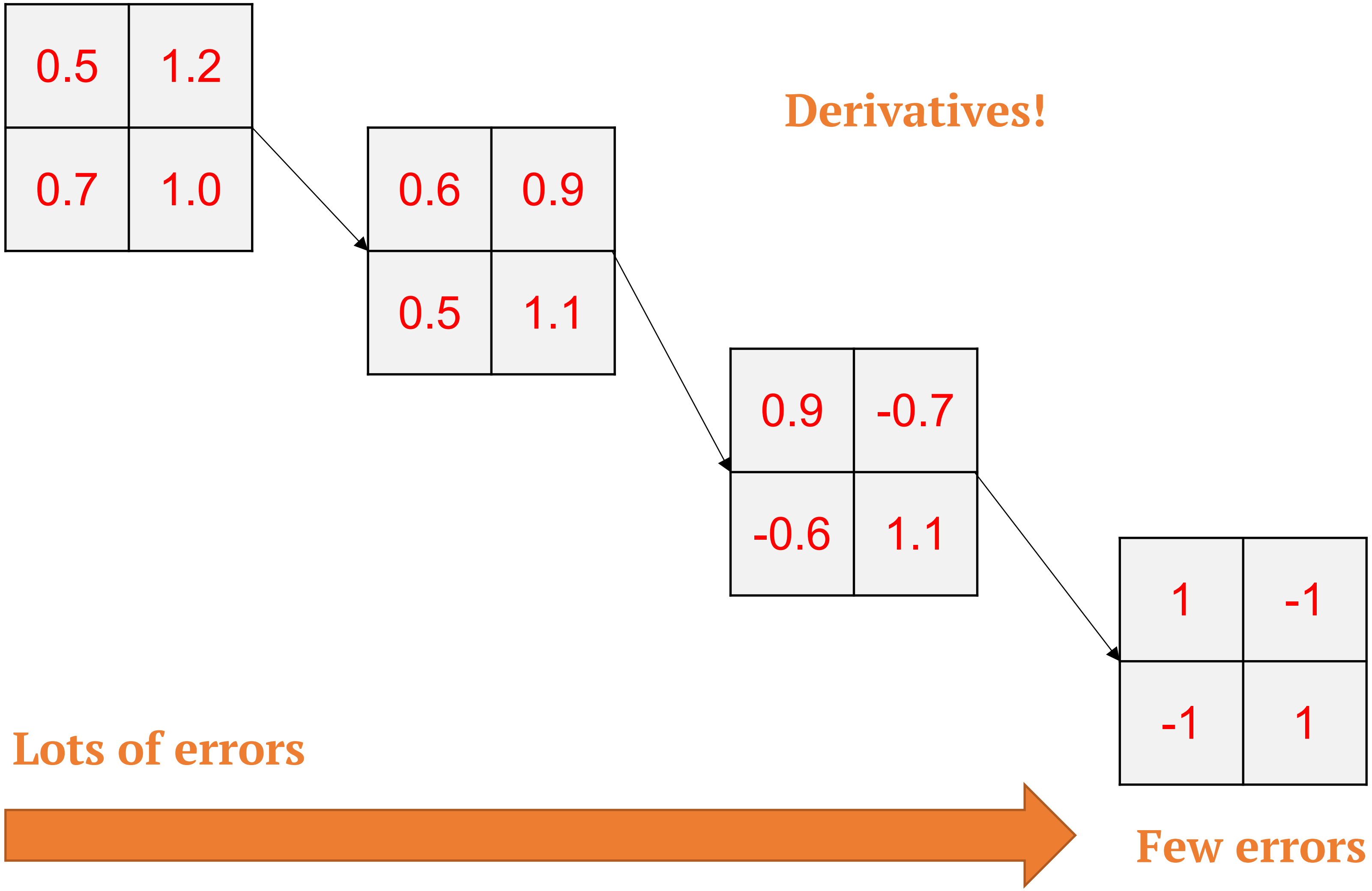
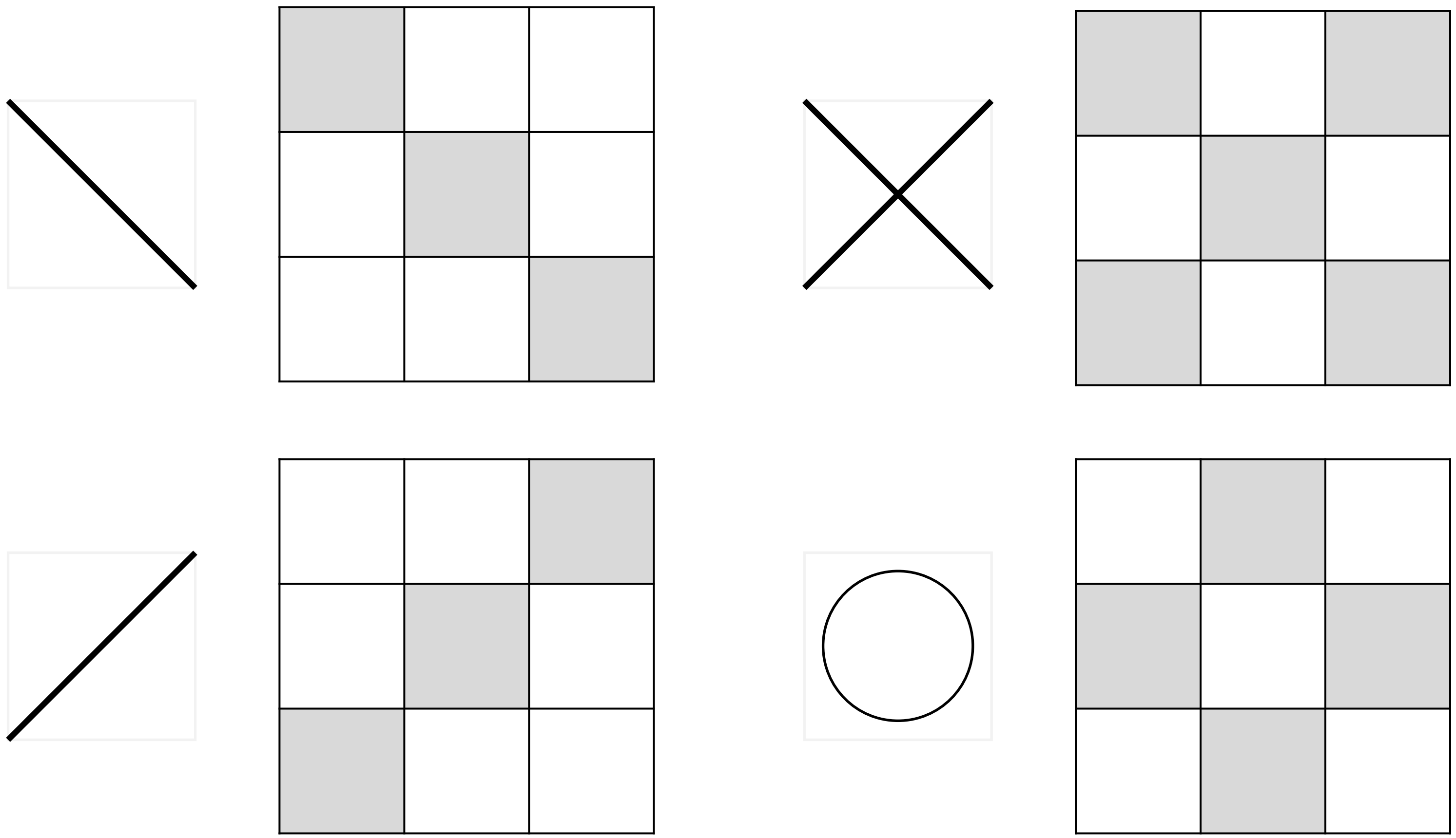


Image Recognition: More Complex Example

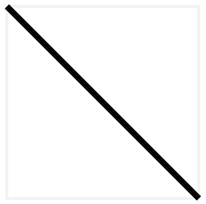


How can a computer find the filters?

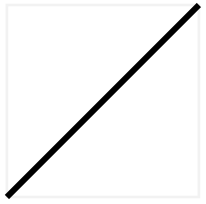
+ 1	- -1	+ 1
- -1	+ 1	- -1
+ 1	- -1	+ 1

Previous Knowledge

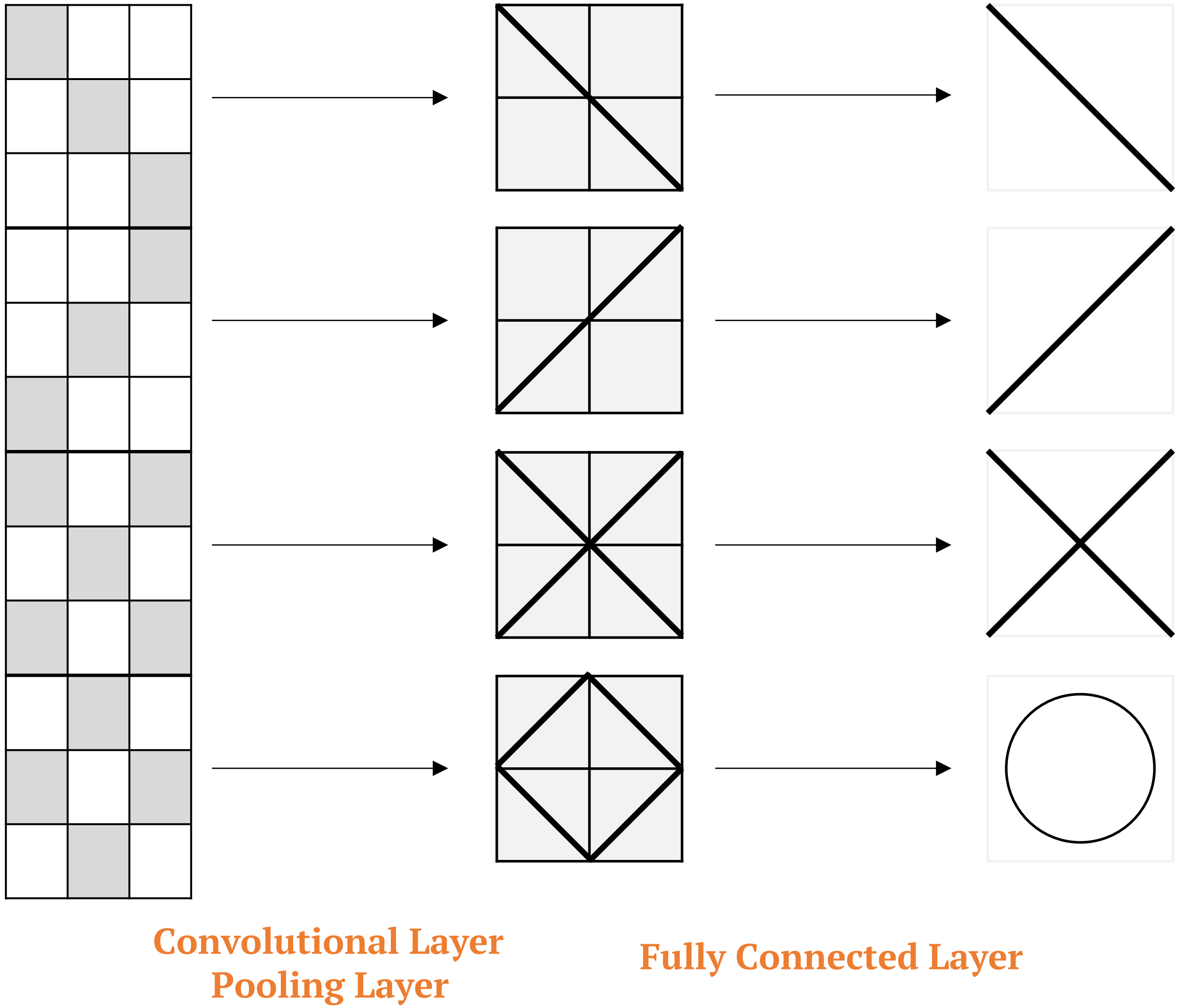
1	-1
-1	1



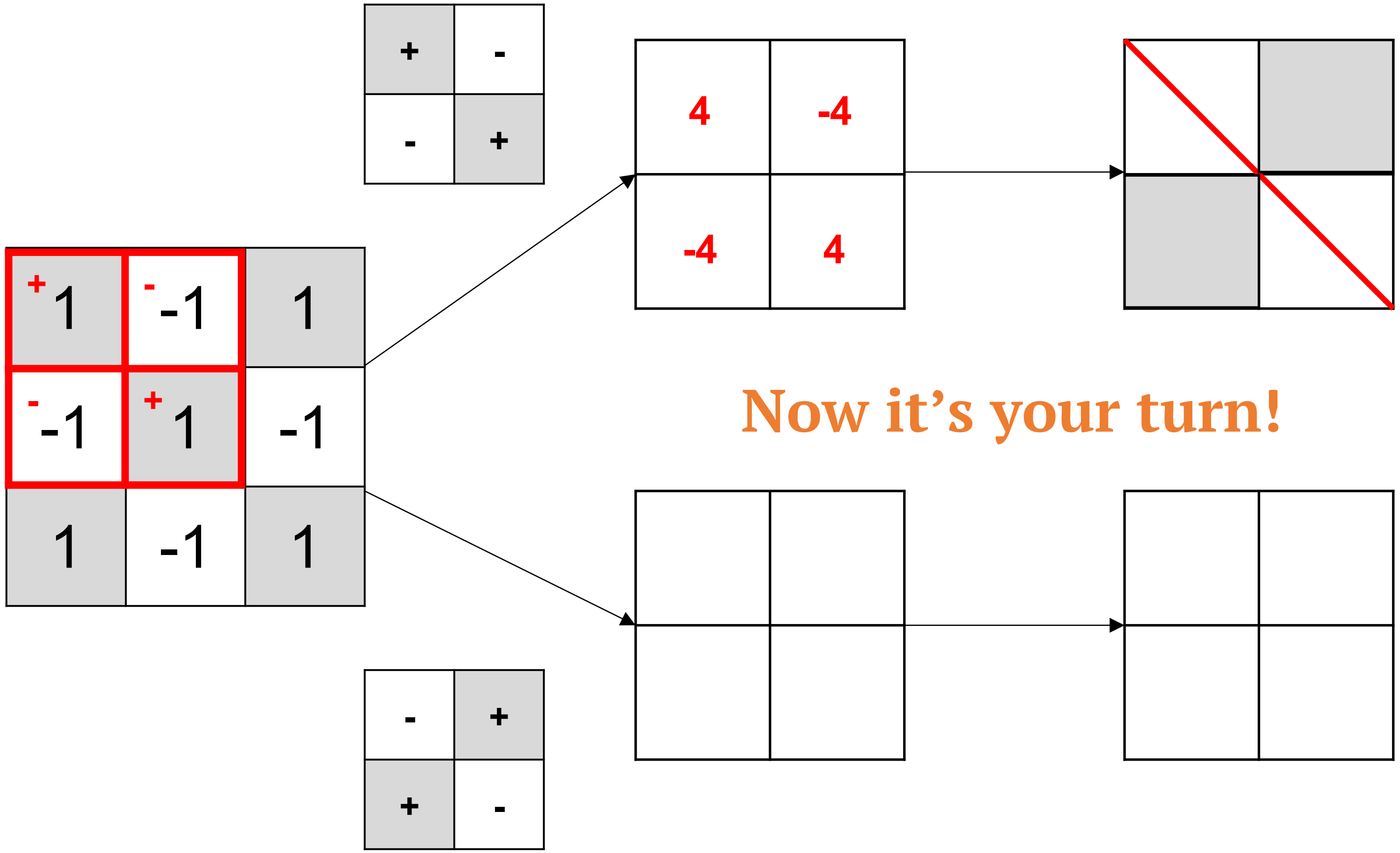
-1	1
1	-1



Convolutional Neural Network

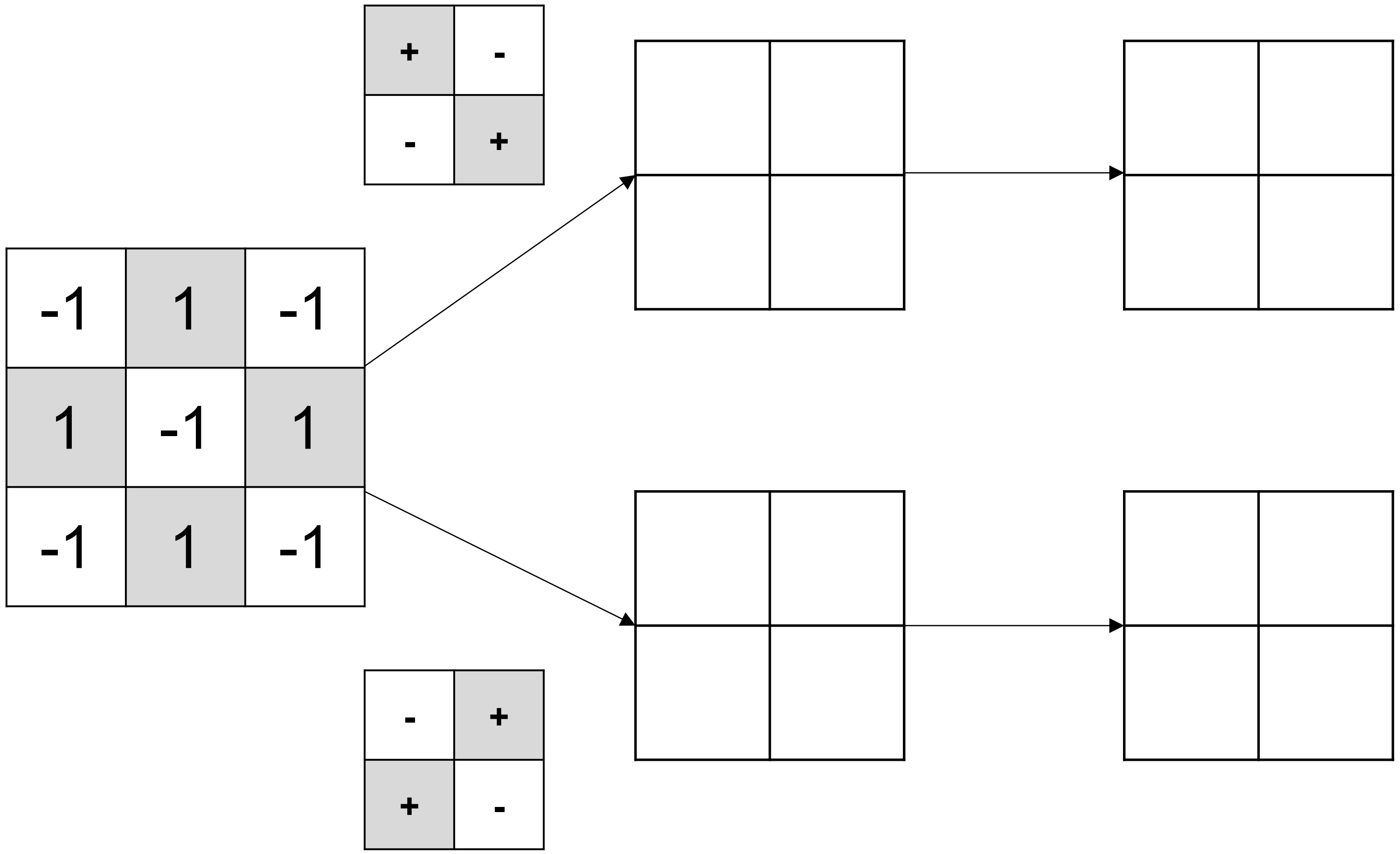


Convolutional Neural Network

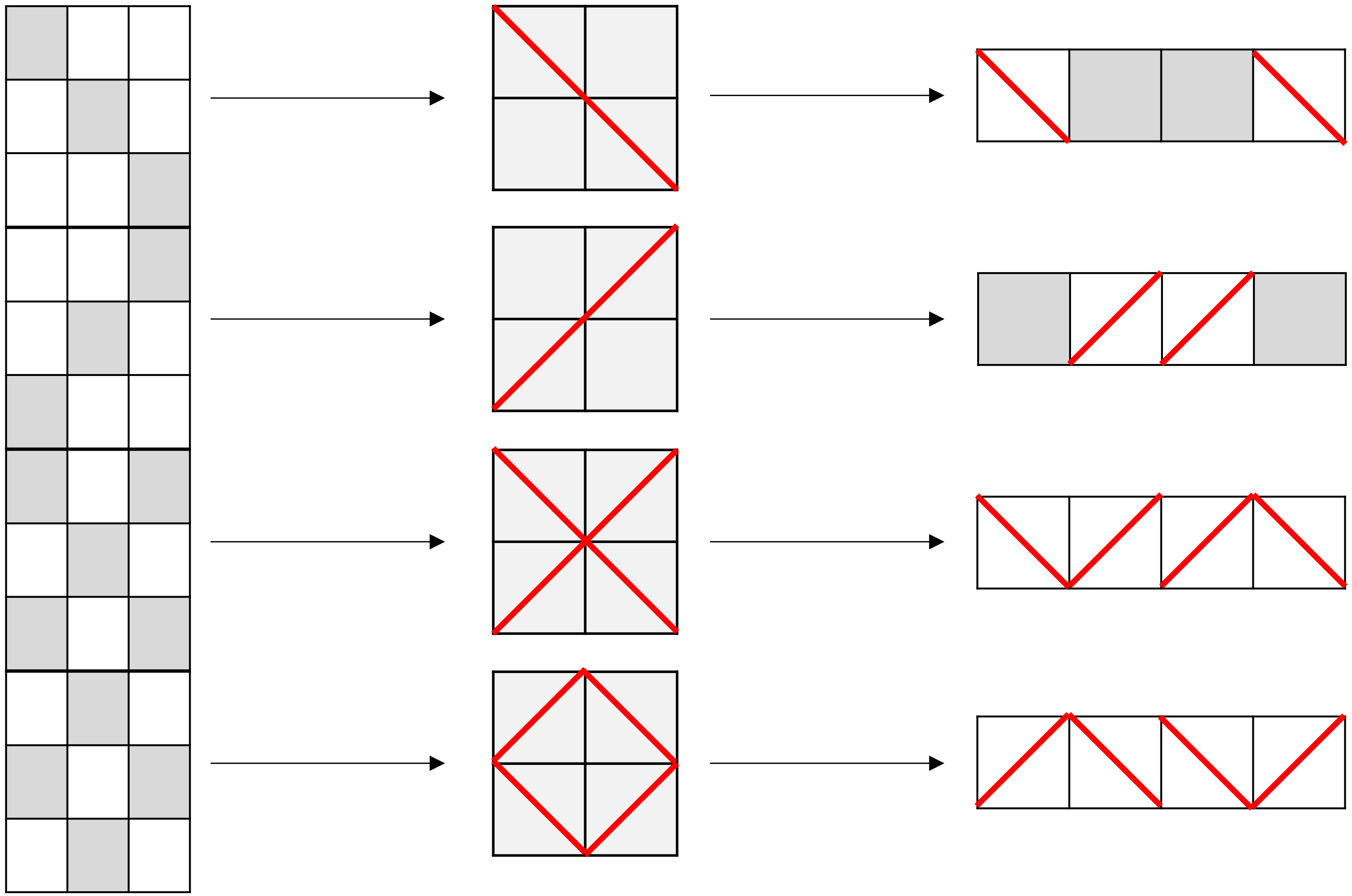


Now it's your turn!

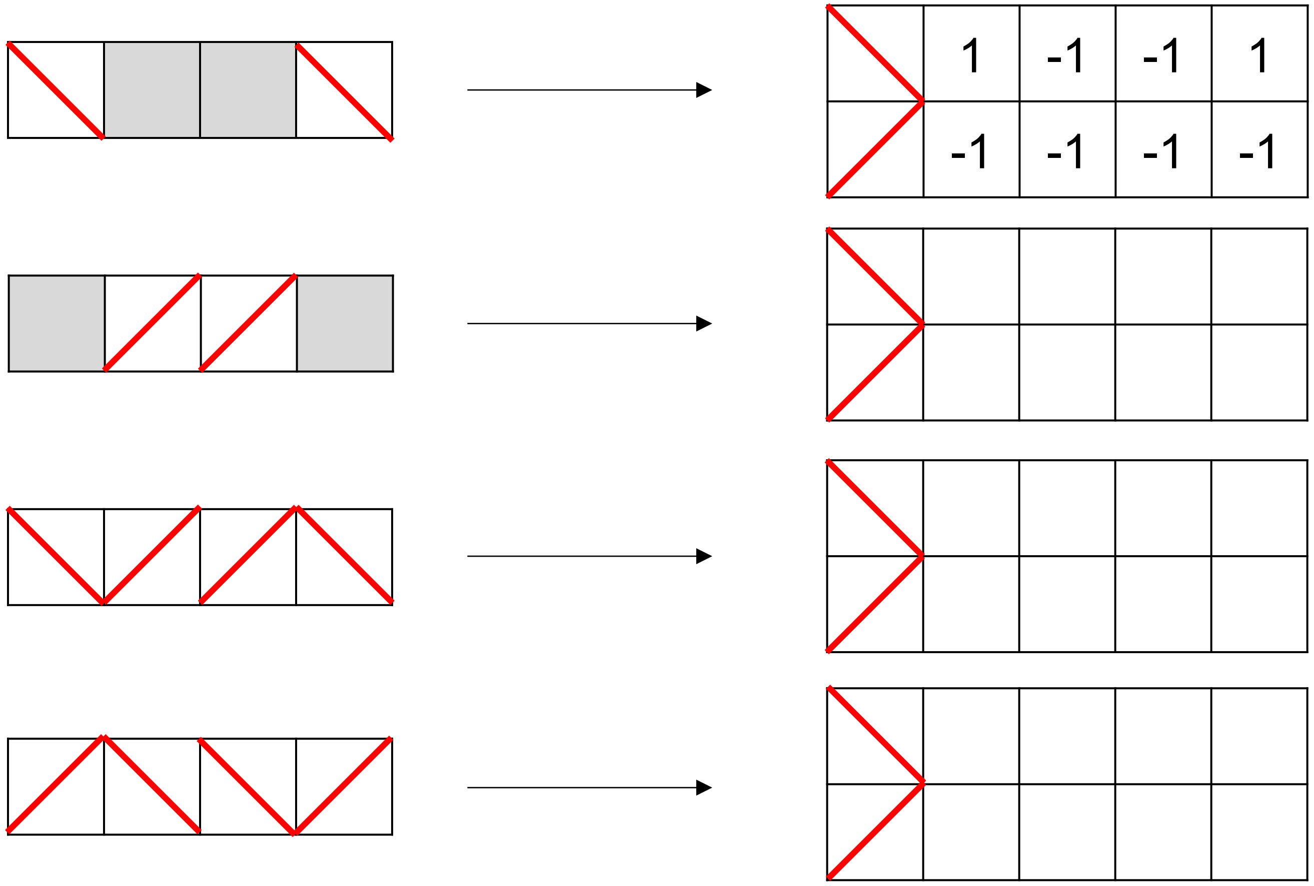
Convolutional Neural Network



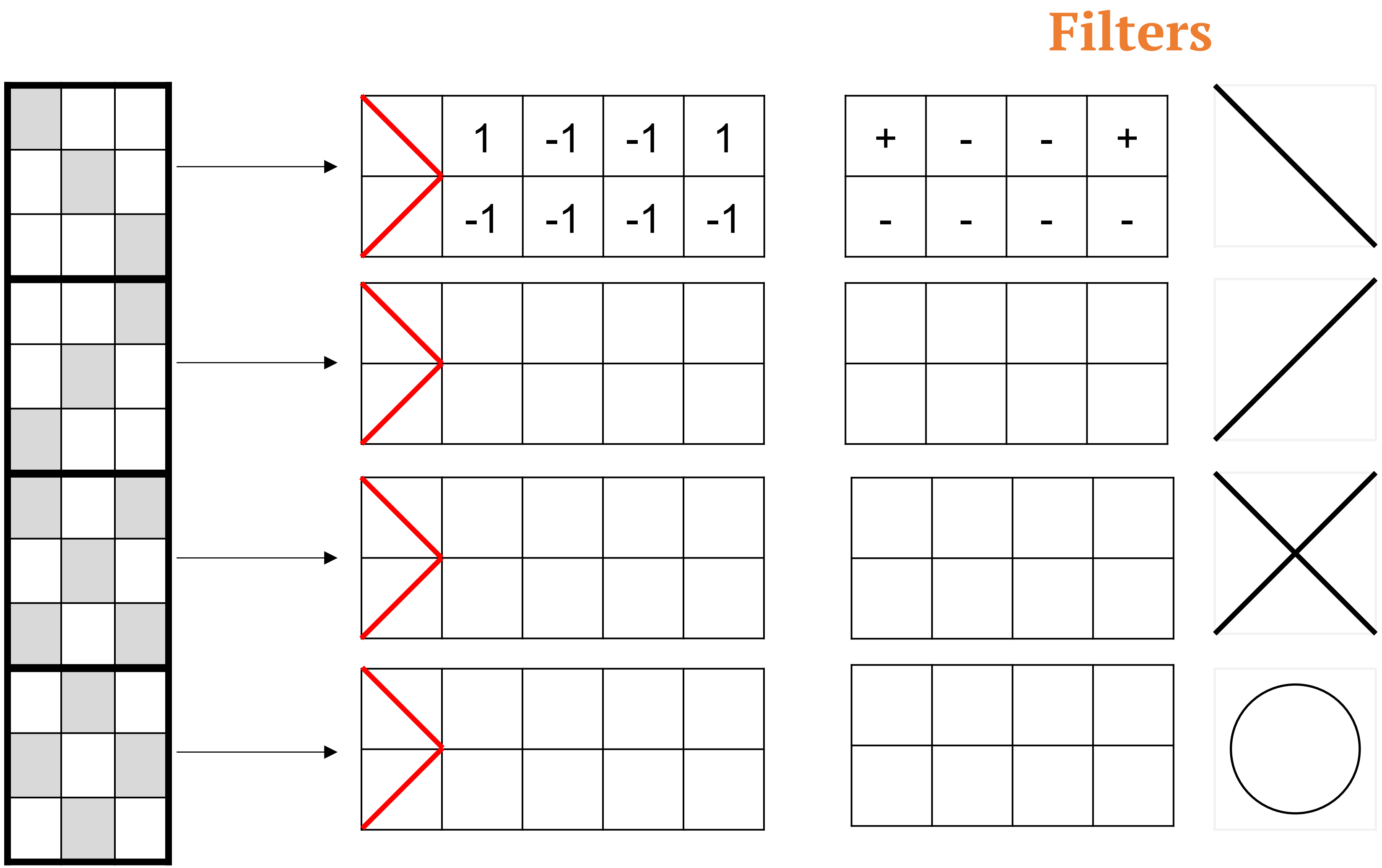
Convolutional Neural Network



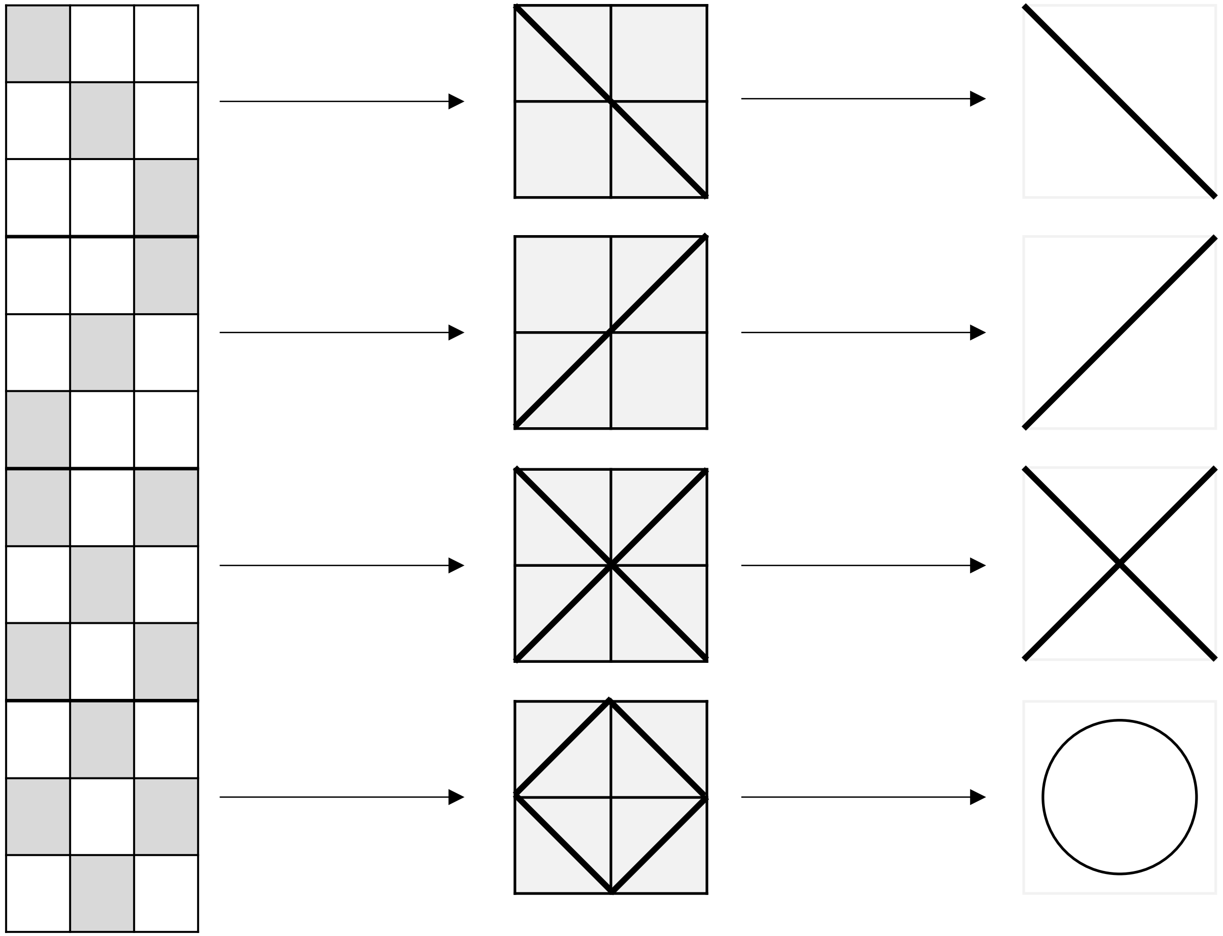
Convolutional Neural Network



Convolutional Neural Network

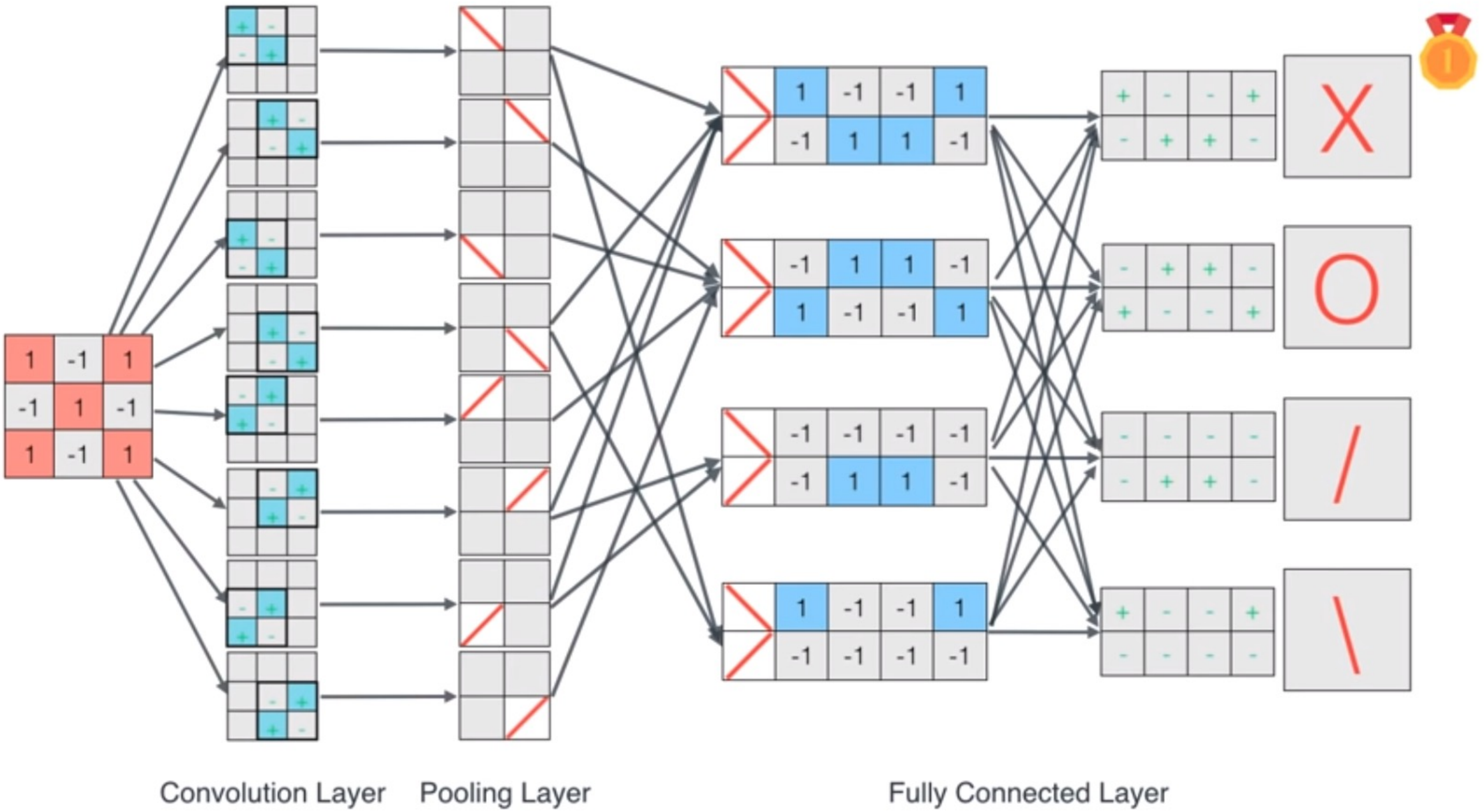


Convolutional Neural Network



Convolutional Layer
Pooling Layer

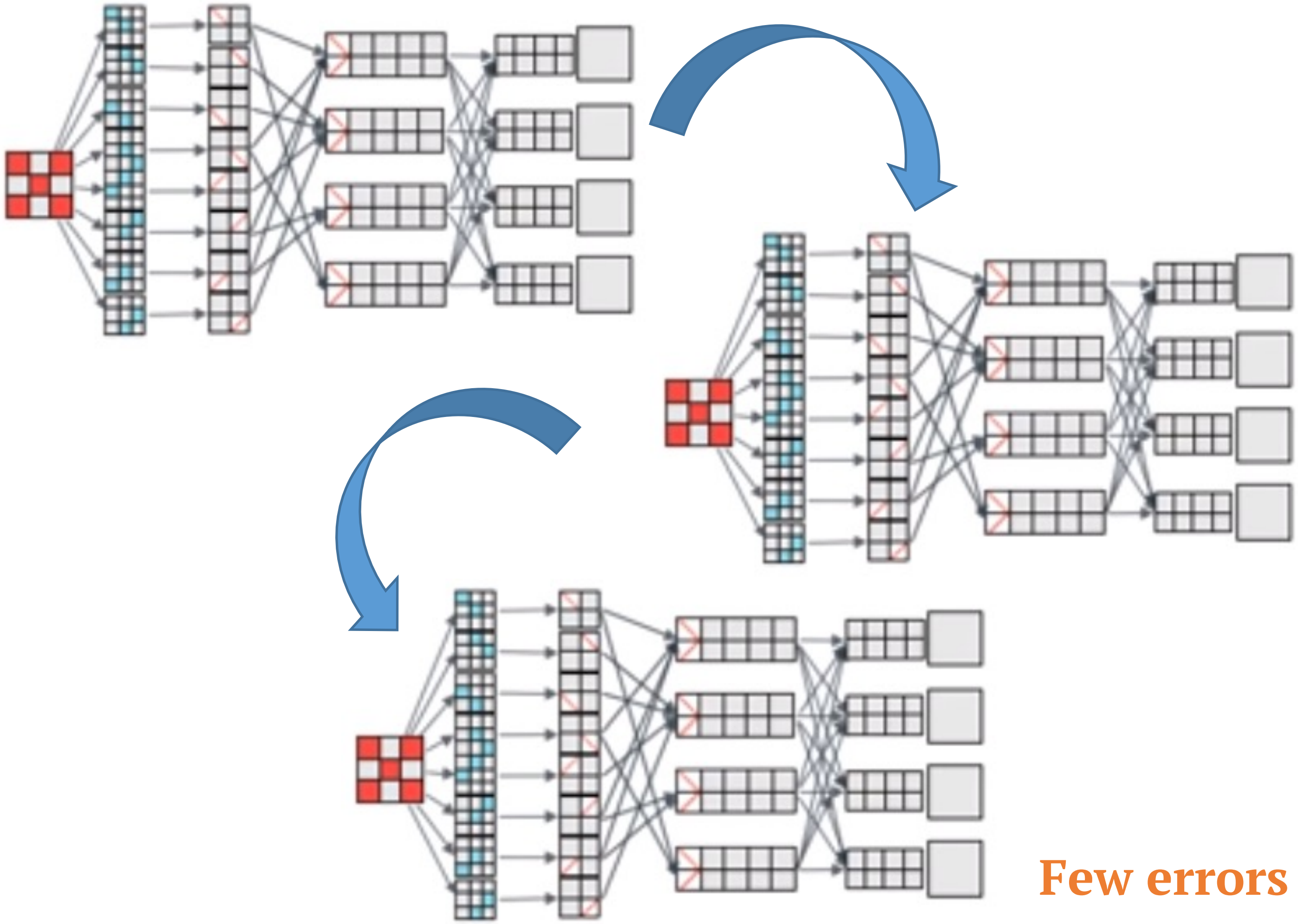
Fully Connected Layer



Convolution Layer
Pooling Layer
Fully Connected Layer
X
O
/
\

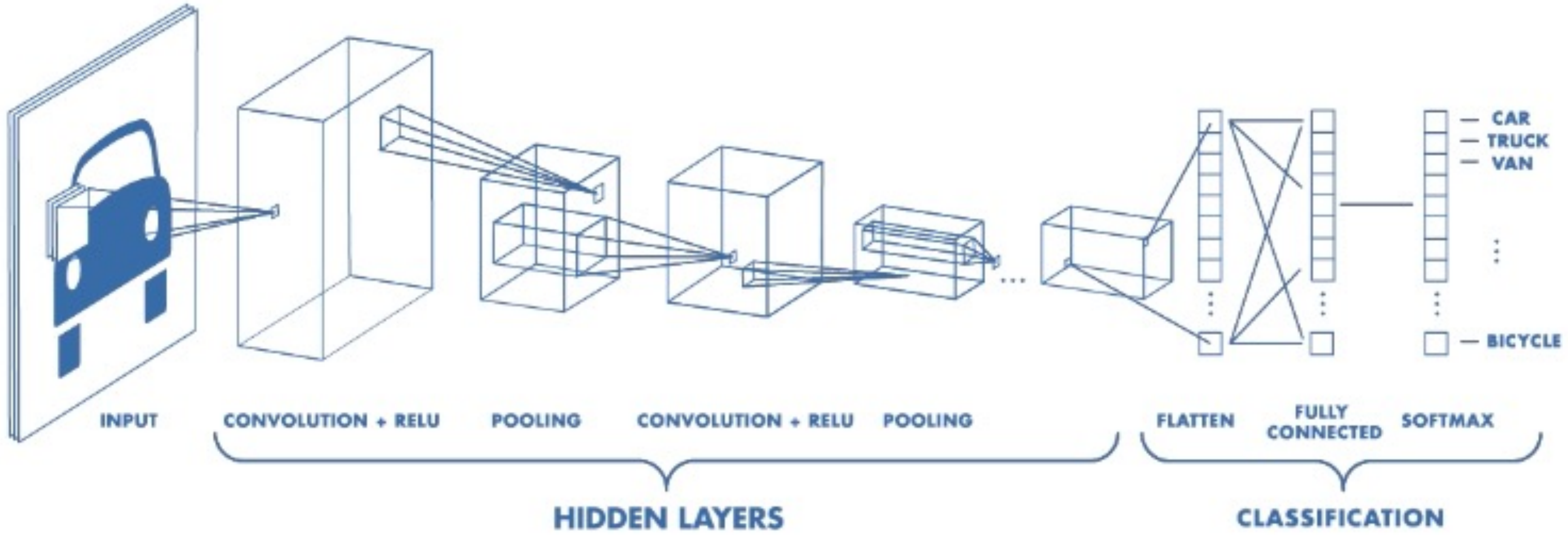
Gradient Descent

Lots of errors



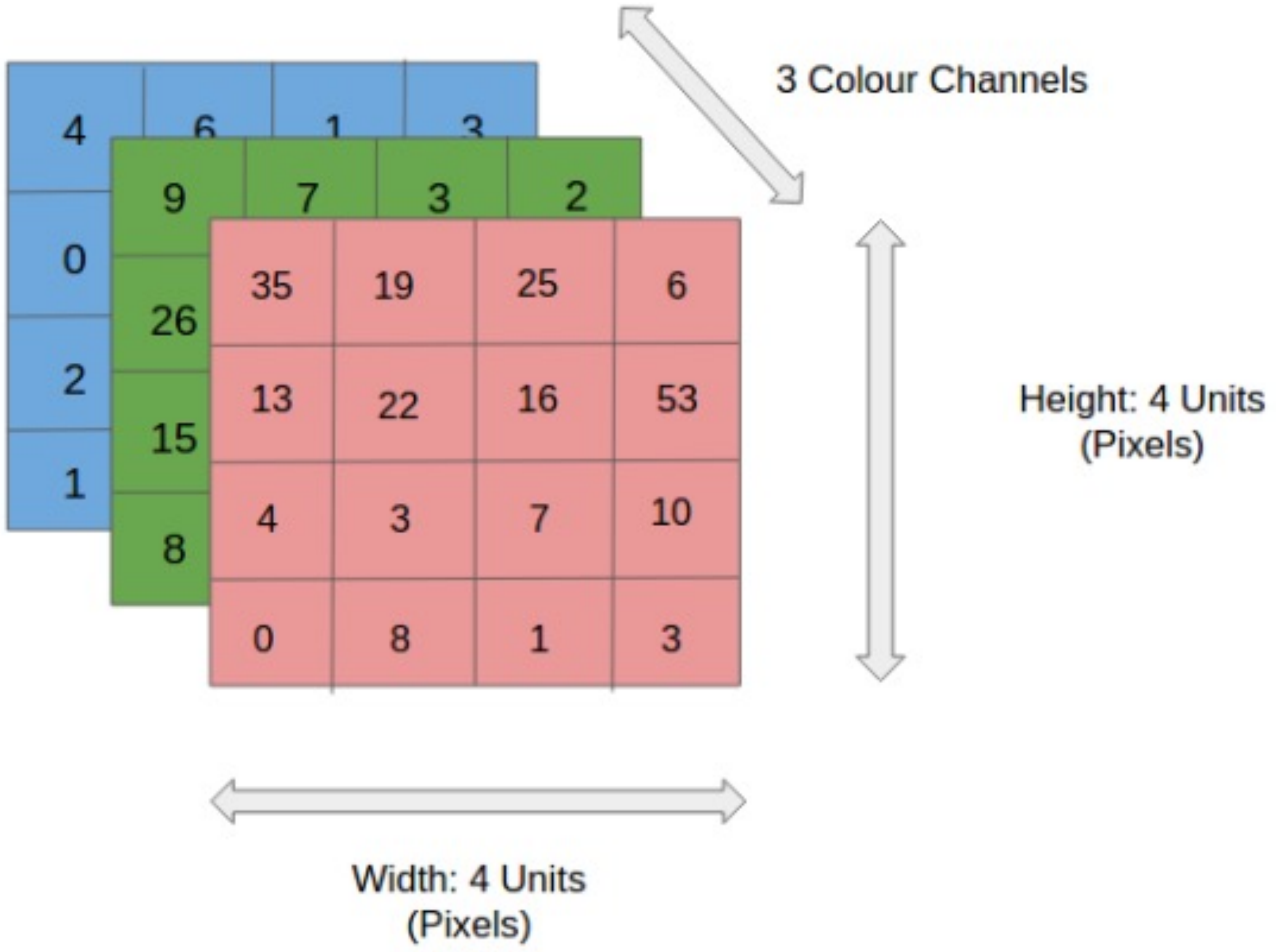
Few errors

Typical CNN architecture



Source: mathworks.com

Input Image

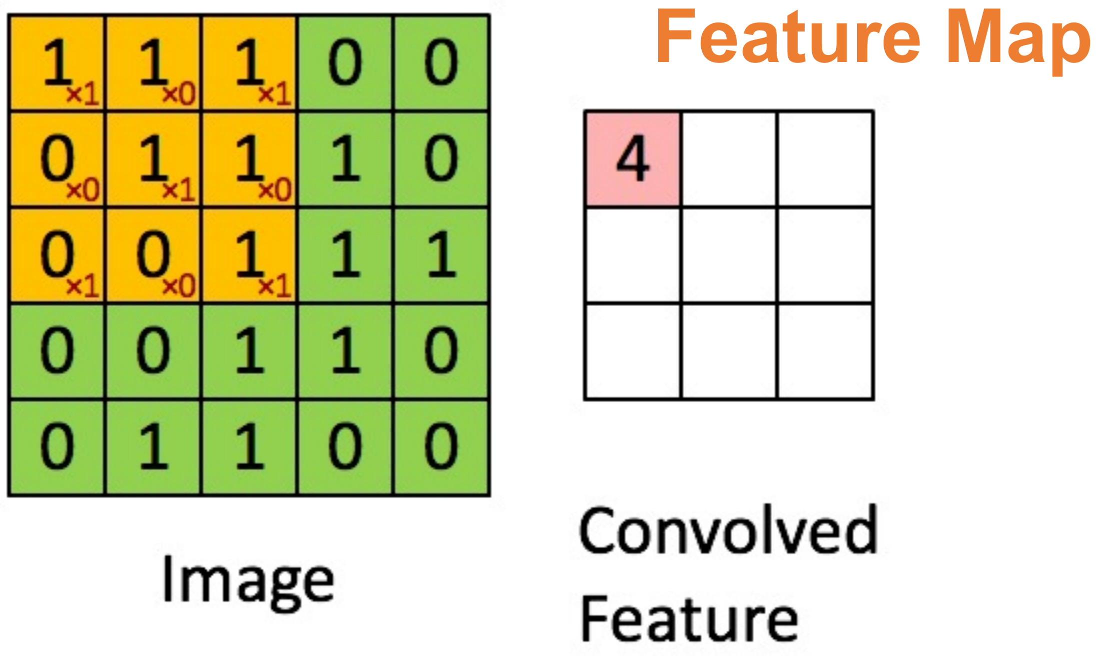


4
6
1
3
0
2
1
9
7
3
2
26
15
8
35
19
25
6
13
22
16
53
4
3
7
10
0
8
1
3
3 Colour Channels
Height: 4 Units
(Pixels)
Width: 4 Units
(Pixels)

2 Components of CNNs

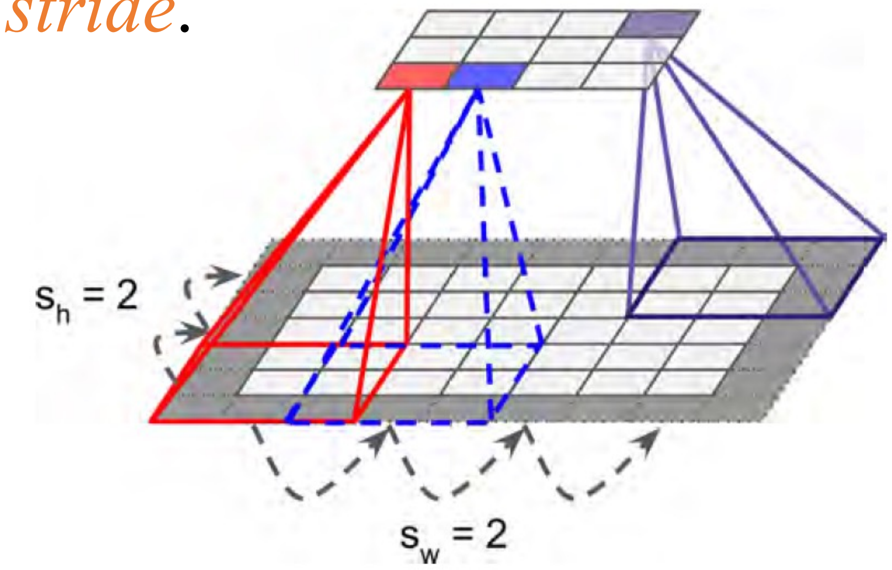
- Feature extraction – the hidden layers
  - Convolution layers – the kernel
  - Pooling layers
- Classification – the fully connected layers

Feature Extraction: Convolution Layer



Convolution Layer: Strides

- It is also possible to connect a large input layer to a much smaller layer by spacing out the receptive fields.
- The distance between two consecutive receptive fields is called the *stride*.



Convolution Layer: Strides

1	2	3	4	5	6	7
11	12	13	14	15	16	17
21	22	23	24	25	26	27
31	32	33	34	35	36	37
41	42	43	44	45	46	47
51	52	53	54	55	56	57
61	62	63	64	65	66	67
71	72	73	74	75	76	77

Convolve with 3x3 filters filled with ones

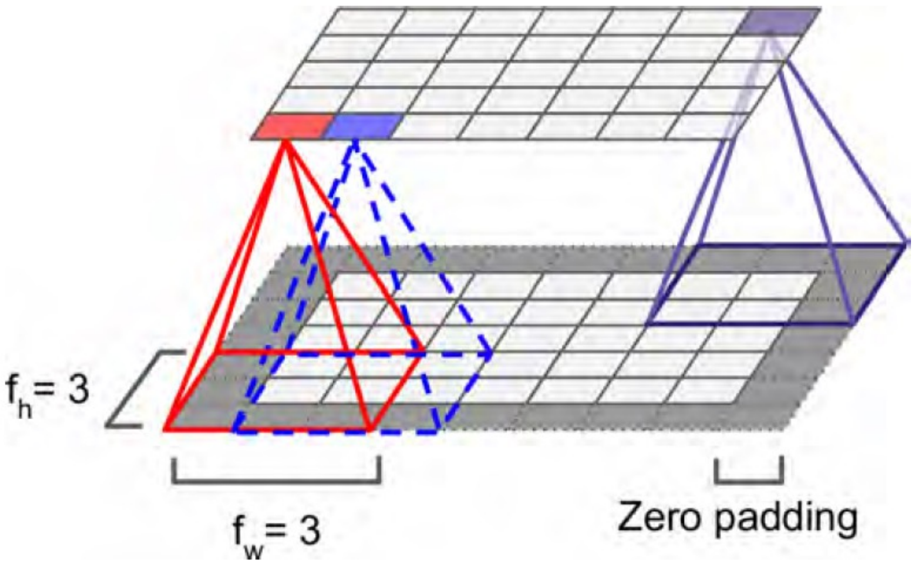
108	126	
288	306	

Stride of 2 pixels

(Source: Raghav Prabhu)

Convolution Layer: Border Effects and Padding

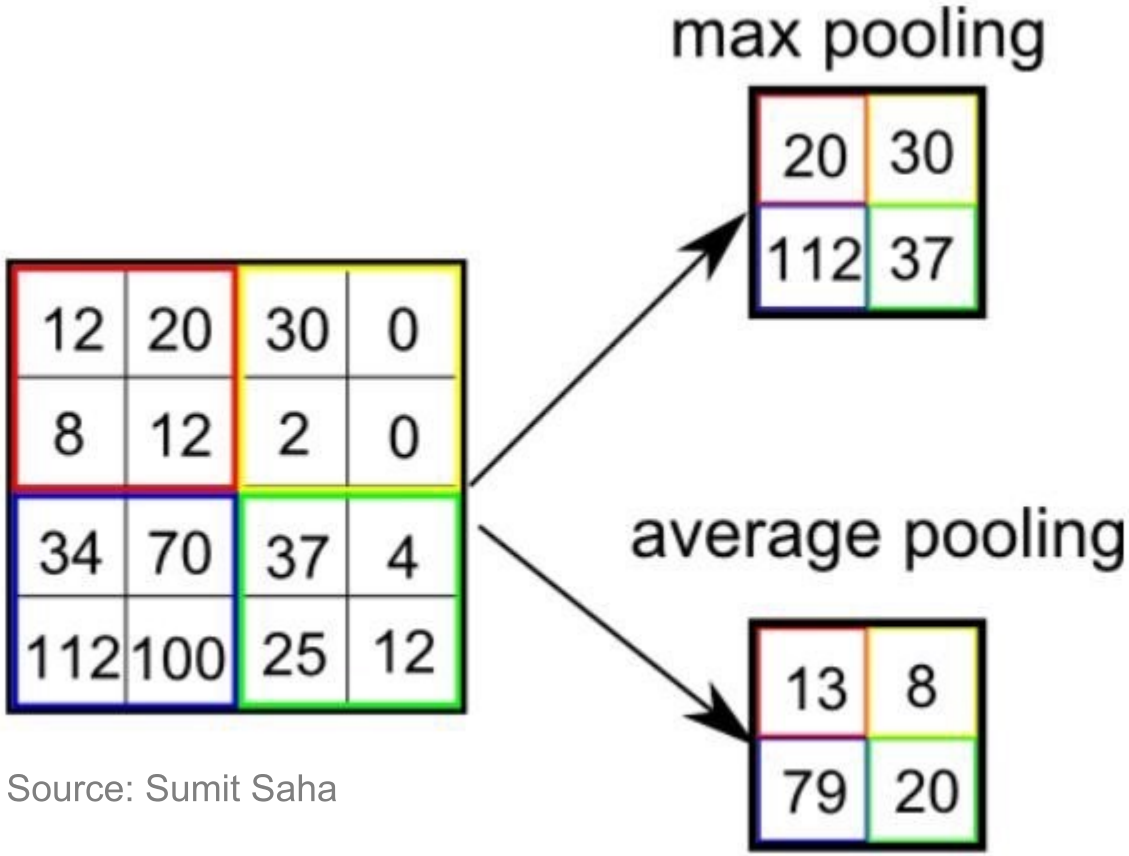
- In order for a layer to have the same height and width as the previous layer, it is common to add zeros around the inputs, as shown in the diagram. This is called *zero padding*.



Feature Extraction: Pooling Layer

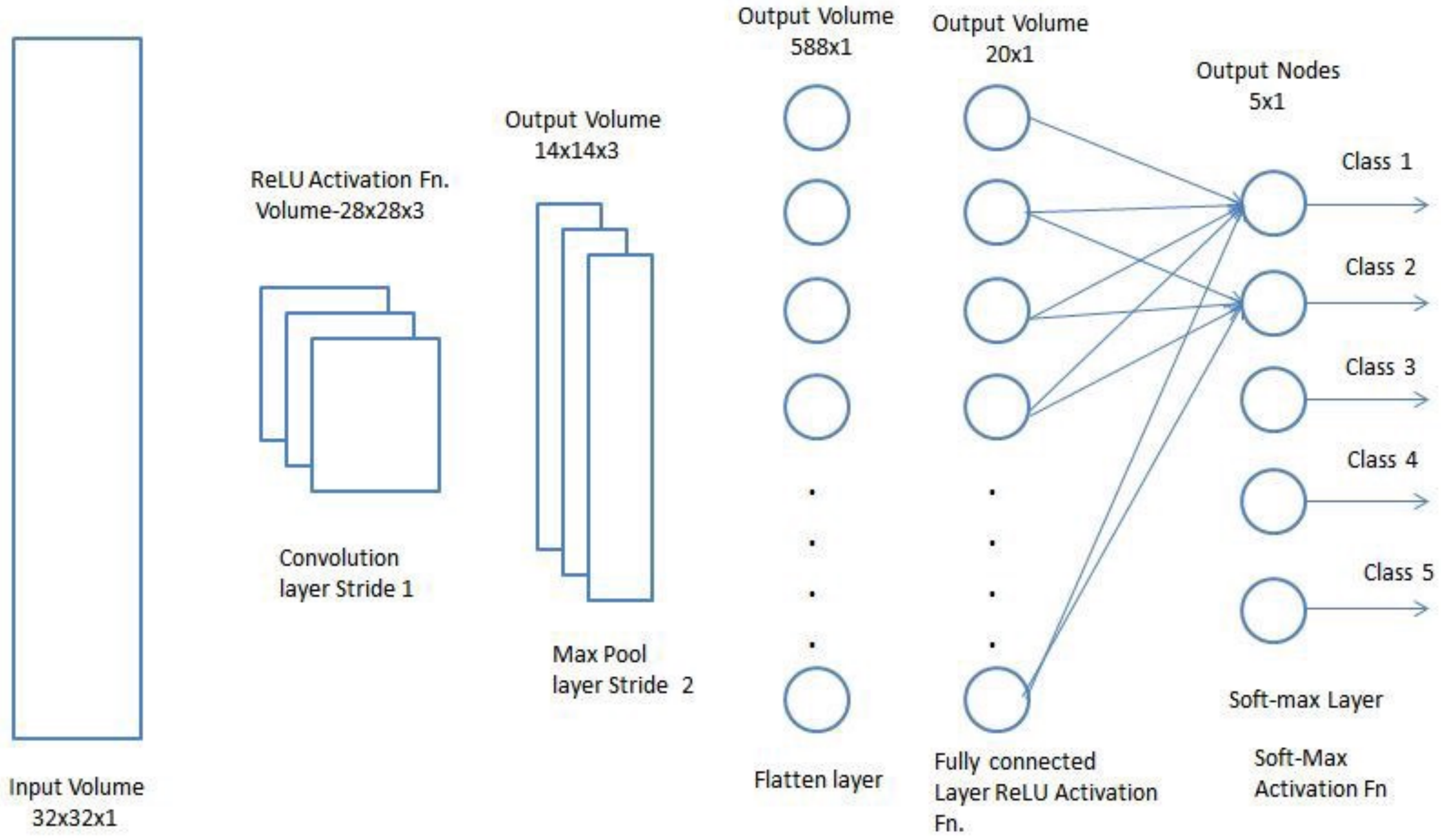
- It is common to add a pooling layer in between CNN layers
  - to continuously reduce the dimensionality
    - to reduce the number of parameters and computation in the network.
  - to shortens the training time
  - to control overfitting.
- Types of pooling
  - Max pooling
  - Average pooling
  - Sum pooling

Feature Extraction: Pooling Layer



Source: Sumit Saha

Classification: Fully Connected Layer

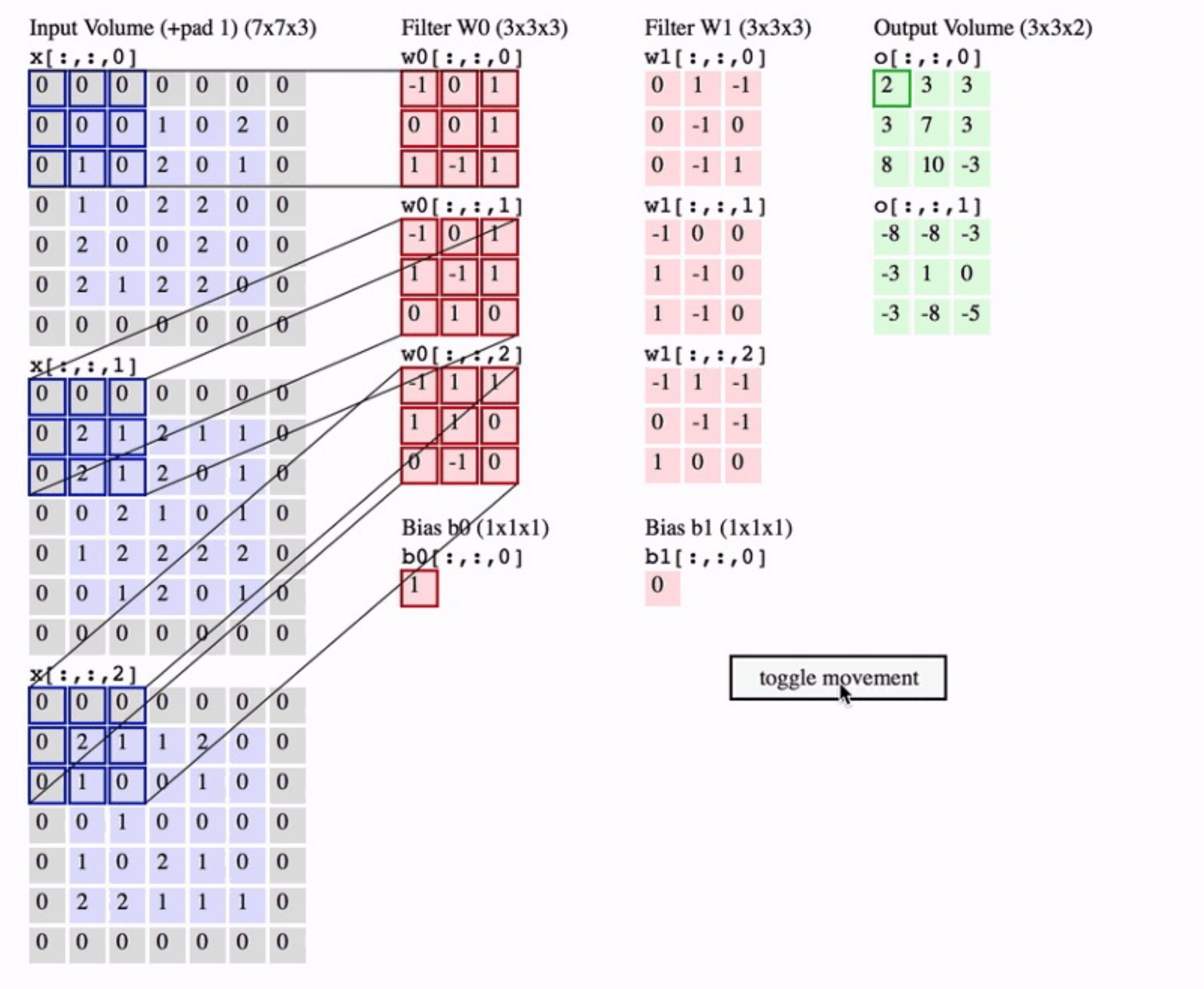


Source: Sumit Saha

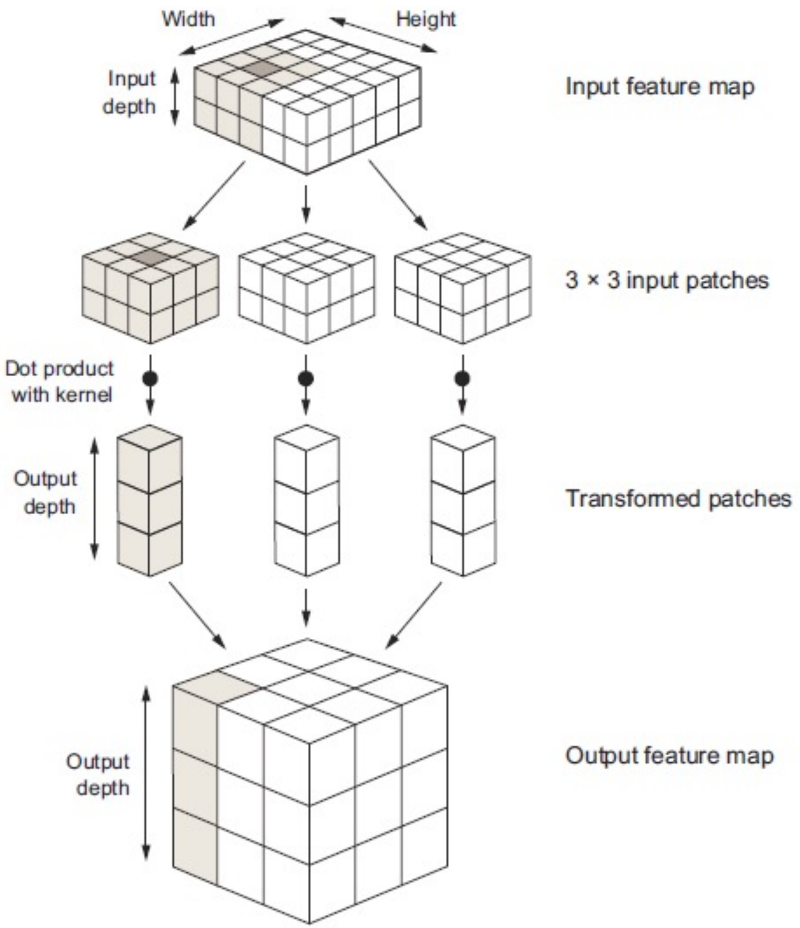
CNN’s 4 Key Hyperparameters

- The kernel size
- The filter count (how many filters we want to use)
- Stride (how big the steps of the filter are)
- Padding

Source: http://cs231n.github.io/convolutional-networks/



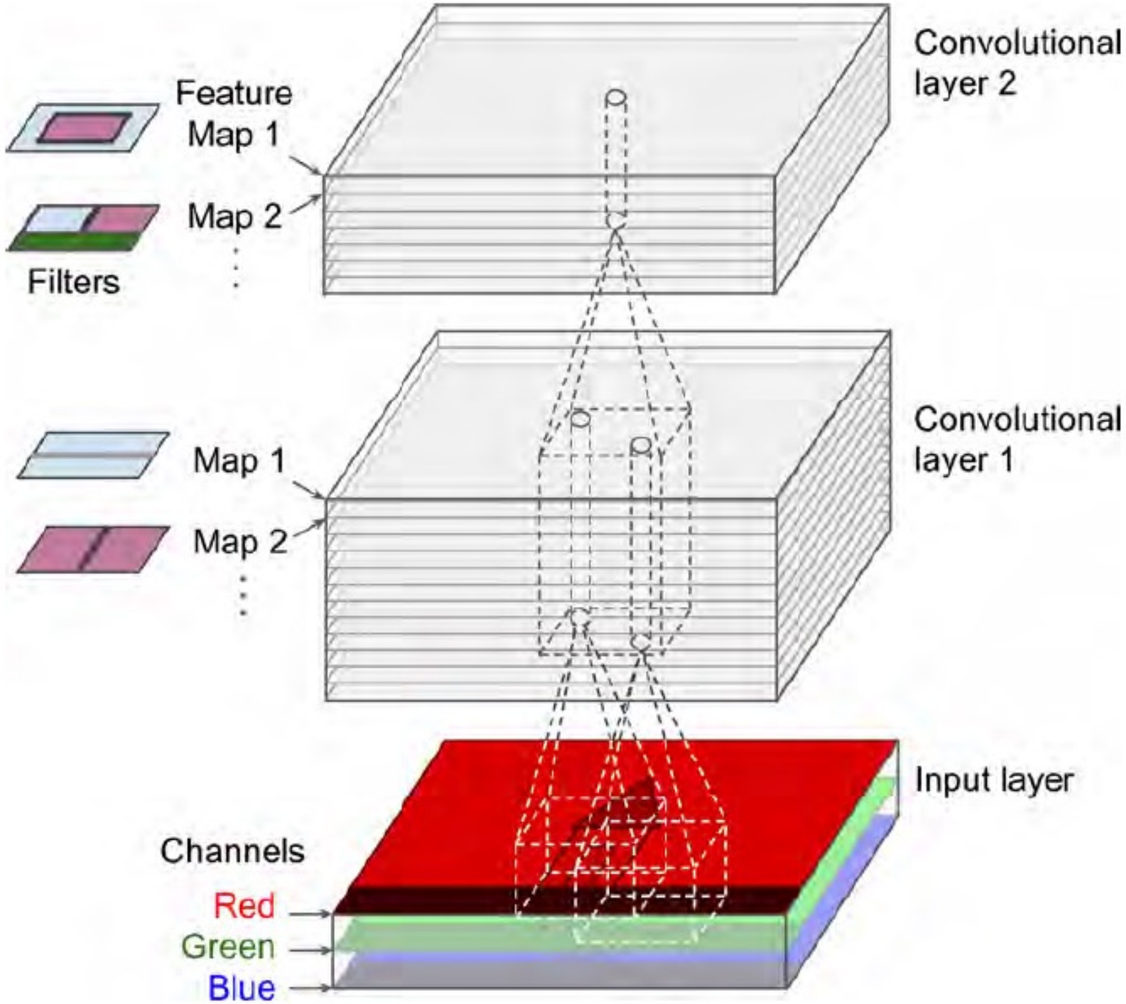
How convolution works



Note that the output width and height may differ from the input width and height.
They may differ for two reasons:

- **Border effects**, which can be countered by padding the input feature map
- The use of ***strides***

Convolution layer with multiple feature maps



Other CNN Architectures

- Classical architecture:
  - LeNet-5 (1998)
- Three winners of the ILSVRC challenge:
  - AlexNet (2012)
  - GoogLeNet (2014)
  - ResNet (2015)

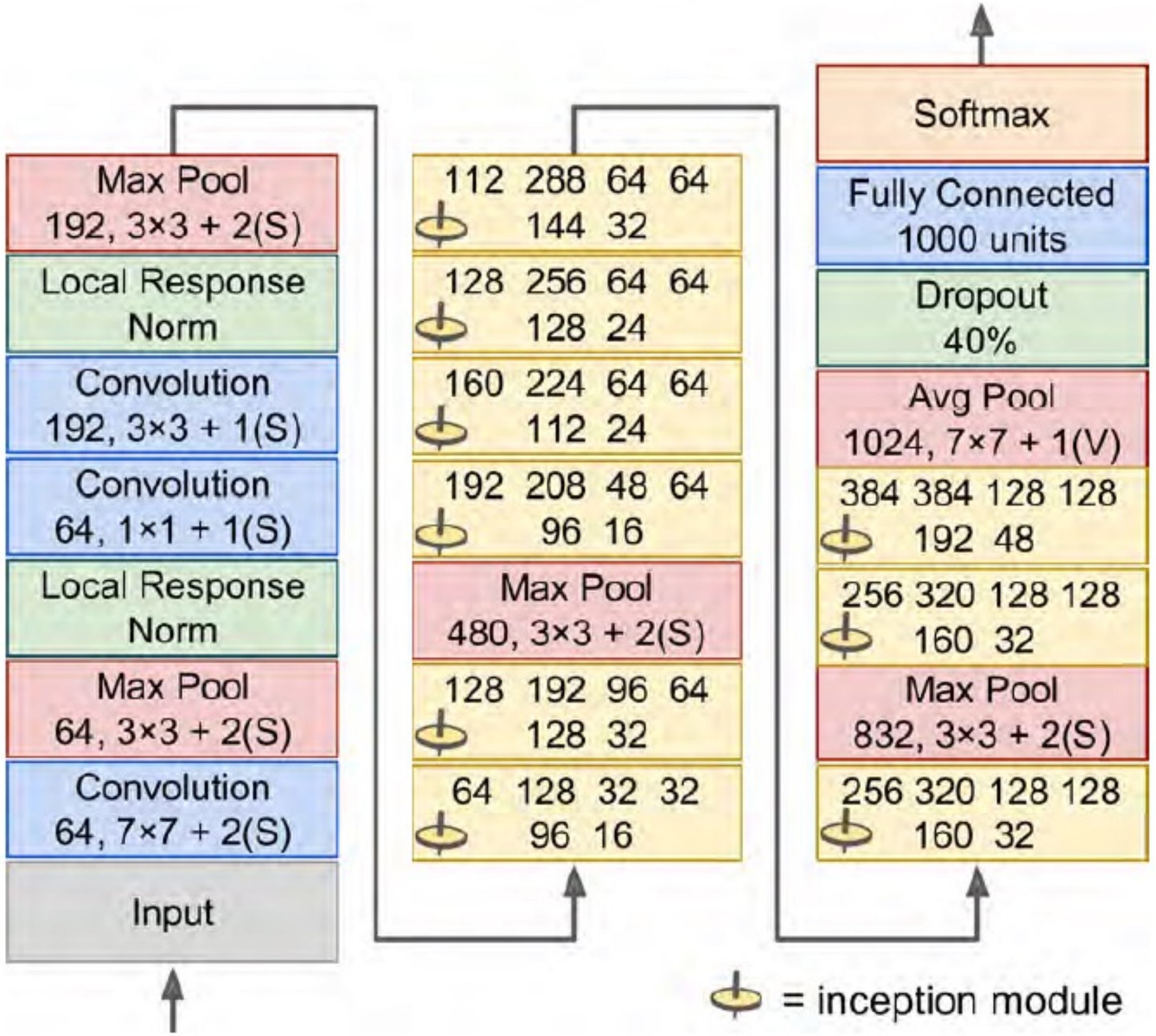
LeNet-5 Architecture

Layer	Type	Maps	Size	Kernel size	Stride	Activation
Out	Fully Connected	–	10	–	–	RBF
F6	Fully Connected	–	84	–	–	tanh
C5	Convolution	120	1 × 1	5 × 5	1	tanh
S4	Avg Pooling	16	5 × 5	2 × 2	2	tanh
C3	Convolution	16	10 × 10	5 × 5	1	tanh
S2	Avg Pooling	6	14 × 14	2 × 2	2	tanh
C1	Convolution	6	28 × 28	5 × 5	1	tanh
In	Input	1	32 × 32	–	–	–

AlexNet

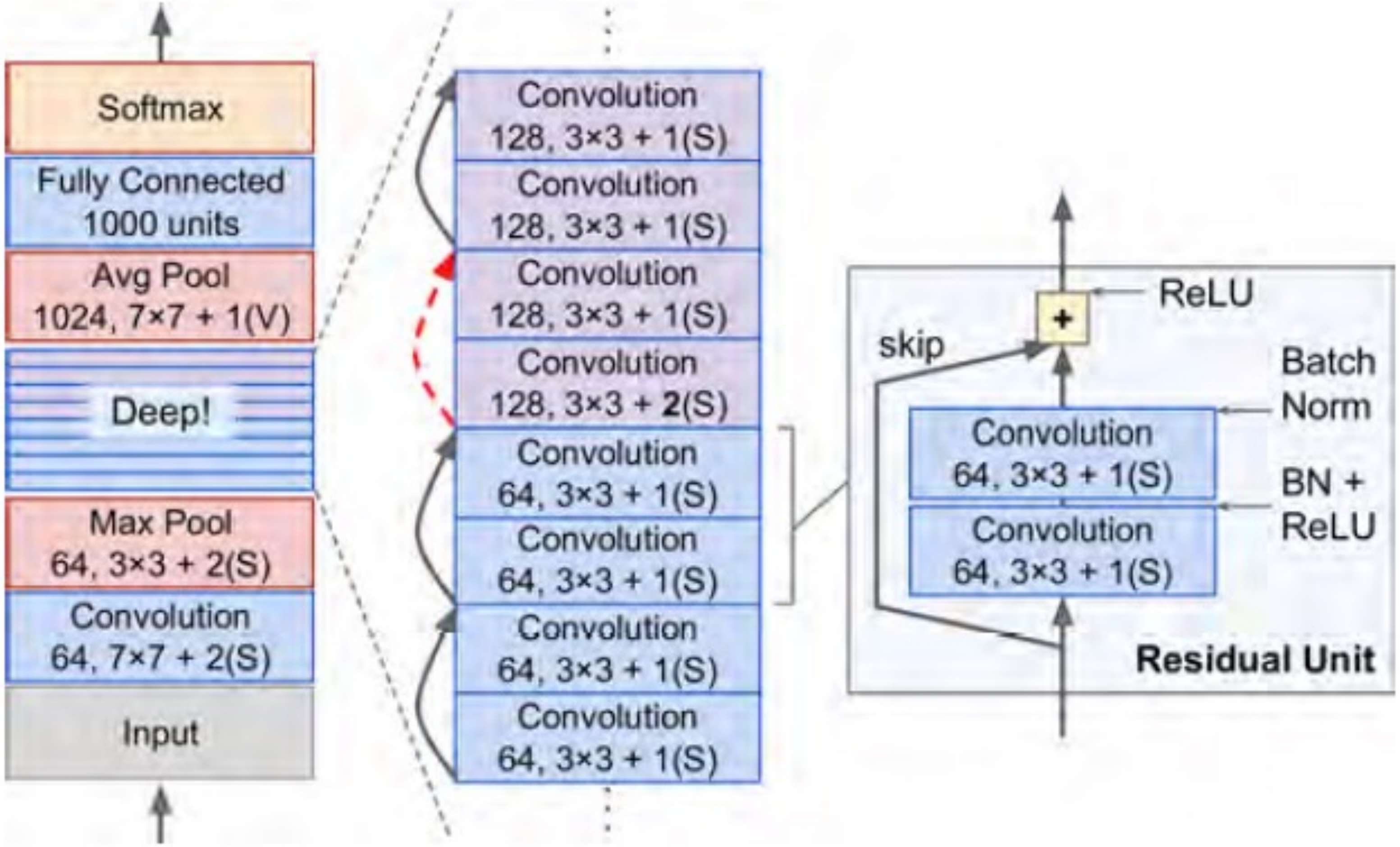
Layer	Type	Maps	Size	Kernel size	Stride	Padding	Activation
Out	Fully Connected	–	1,000	–	–	–	Softmax
F9	Fully Connected	–	4,096	–	–	–	ReLU
F8	Fully Connected	–	4,096	–	–	–	ReLU
C7	Convolution	256	13 × 13	3 × 3	1	SAME	ReLU
C6	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
C5	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
S4	Max Pooling	256	13 × 13	3 × 3	2	VALID	–
C3	Convolution	256	27 × 27	5 × 5	1	SAME	ReLU
S2	Max Pooling	96	27 × 27	3 × 3	2	VALID	–
C1	Convolution	96	55 × 55	11 × 11	4	SAME	ReLU
In	Input	3 (RGB)	224 × 224	–	–	–	–

GoogLeNet



Max Pool
192, 3×3 + 2(S)
Local Response Norm
Convolution
192, 3×3 + 1(S)
Convolution
64, 1×1 + 1(S)
Local Response Norm
Max Pool
64, 3×3 + 2(S)
Convolution
64, 7×7 + 2(S)
Input
112 288 64 64
144 32
128 256 64 64
128 24
160 224 64 64
112 24
192 208 48 64
96 16
Max Pool
480, 3×3 + 2(S)
128 192 96 64
128 32
64 128 32 32
96 16
Softmax
Fully Connected
1000 units
Dropout
40%
Avg Pool
1024, 7×7 + 1(V)
384 384 128 128
192 48
256 320 128 128
160 32
Max Pool
832, 3×3 + 2(S)
256 320 128 128
160 32
= inception module

ResNet Architecture



Transfer Learning

> "I think transfer learning is the key to general intelligence. And I think the key to doing transfer learning will be the acquisition of conceptual knowledge that is abstracted away from perceptual details of where you learned it from."
>
> \- Demis Hassabis
>
> CEO, DeepMind

What is transfer learning?

- A machine learning technique where a model trained on one task is re-purposed on a second related task
  - For example, if you trained a simple classifier to predict whether an image contains a backpack, you could use the knowledge that the model gained during its training to recognize other objects like sunglasses.
- Mostly used in Computer Vision and Natural Language Processing Tasks
  - because of the huge amount of computational power that is needed for them

How to use transfer learning?

- Two common approaches:
  - Develop model
  - Pre-trained model

Develop Model Approach

1. **Select Source Task**.
   - select a related predictive modeling problem with an abundance of data
2. **Develop Source Model**.
   - develop a skillful model for this first task
   - The model must be better than a naive model
3. **Reuse Model**.
   - The model fit on the source task can then be used as the starting point for a model on the second task of interest.
   - This may involve using all or parts of the model, depending on the modeling technique used.
4. **Tune Model**.

Pre-Trained Model Approach

1. **Select Source Model**.
   - choose an available pre-trained source model
2. **Reuse Model**.
   - use the pre-trained model can as the starting point for the second task of interest
3. **Tune Model**.

* common in the field of deep learning *

Examples of Transfer Learning with Image Data

- It is common to use a deep learning model pre-trained for a large and challenging image classification task such as the ImageNet 1000-class photograph classification competition
- The research organizations often release their final model under a permissive license for reuse
  - Oxford VGG Model:

    http://www.robots.ox.ac.uk/~vgg/research/very_deep/
  - Google Inception Model

    https://github.com/tensorflow/models/tree/master/inception
  - Microsoft ResNet Model

    https://github.com/KaimingHe/deep-residual-networks
- These models can take days or weeks to train on modern hardware.
- These models can be downloaded and incorporated directly into new models that expect image data as input.

Training CNN in Keras

```
cnn = models.Sequential()
cnn.add(layers.Conv2D(40, kernel_size=5, padding="same",
                      input_shape=(28, 28, 1), activation = 'relu',name ='conv1_1'))
cnn.add(layers.MaxPool2D((2, 2)))
cnn.add(layers.Dropout(0.25))
cnn.add(layers.Conv2D(32, kernel_size=(3, 3),
                      activation='relu',kernel_initializer='he_normal',name ='conv1_2'))
cnn.add(layers.MaxPool2D((2, 2)))
cnn.add(layers.Dropout(0.25))
cnn.add(layers.Flatten())
cnn.add(layers.Dense(64, activation='relu'))
cnn.add(layers.BatchNormalization())
cnn.add(layers.Dropout(0.25))
cnn.add(layers.Dense(10, activation='softmax'))
```

Input Mnist data

Shape (28 , 28 , 1)

Covolution layer
40, 5*5

MaxPooling 2*2
Dropout(0.25)

Covolution layer
32, 3*3

MaxPooling 2*2
Dropout(0.25)

Flatten layer

Hidden layer
64 Node

BatchNormalization()
Dropout(0.25)

Output layer
10 Node

```
cnn = models.Sequential()
cnn.add(layers.Conv2D(40, kernel_size=5, padding="same",
                      input_shape=(28, 28, 1), activation = 'relu',name ='conv1_1'))
cnn.add(layers.MaxPool2D((2, 2)))
cnn.add(layers.Dropout(0.25))
cnn.add(layers.Conv2D(32, kernel_size=(3, 3),
                      activation='relu',kernel_initializer='he_normal',name ='conv1_2'))
cnn.add(layers.MaxPool2D((2, 2)))
cnn.add(layers.Dropout(0.25))
cnn.add(layers.Flatten())
cnn.add(layers.Dense(64, activation='relu'))
cnn.add(layers.BatchNormalization())
cnn.add(layers.Dropout(0.25))
cnn.add(layers.Dense(10, activation='softmax'))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv1_1 (Conv2D)             (None, 28, 28, 40)        1040      
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 40)        0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 14, 14, 40)        0         
_________________________________________________________________
conv1_2 (Conv2D)             (None, 12, 12, 32)        11552     
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 32)          0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 6, 6, 32)          0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 1152)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 64)                73792     
_________________________________________________________________
batch_normalization_1 (Batch (None, 64)                256       
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_2 (Dense)              (None, 10)                650       
=================================================================
Total params: 87,290
Trainable params: 87,162
Non-trainable params: 128
```

Data augmentation

```
# Define a generator for train set and test set

train_datagen = image.ImageDataGenerator(rescale=1./255,
                                  rotation_range=40,
                                  width_shift_range=0.2,
                                  height_shift_range=0.2,
                                  shear_range=0.2,
                                  zoom_range=0.2,
                                  horizontal_flip=False)

test_datagen = image.ImageDataGenerator(rescale=1./255)
```

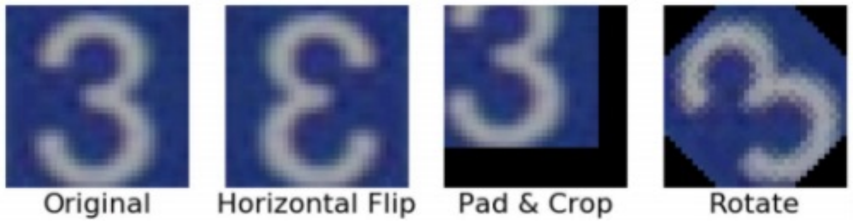
```
# Create an Iterator object.
train_generator = train_datagen.flow(X_train,y_train,
                                 batch_size = BATCH_SIZE,
                                 seed=0)

validate_generator = test_datagen.flow(X_val,y_val,
                                 batch_size = BATCH_SIZE,
                                 shuffle=False)
```

Using the ImageDataGenerator module to generate more data.

It help generate more variation of the data which help prevent overfit and generalize better.

https://keras.io/preprocessing/image



Transfer Learning

```
from keras.applications import vgg16

vgg = vgg16.VGG16(include_top=False,
                  weights='imagenet',
                  input_shape=(150,150,3))
prev_cnn = models.load_model('your_previos_model.h5')
prev_cnn.summary()
```

\# Use .pop() to remove the last layer
\# In this case, we want to remove last two layer

```
prev_cnn.pop()
prev_cnn.pop()
```

If we don't want to train these layer, we have to freeze these layer.

```
prev_cnn.trainable = False
```

Or Freeze a specific layers
\# Freeze first 3 layer

```
for i in range(3):
    prev_cnn.layers[i].trainable = False
```

What is transfer learning https://towardsdatascience.com/transfer-learning-946518f95666
which transfer learning method to use https://medium.com/@14prakash/transfer-learning-using-keras-d804b2e04ef8
The following is a tutorial code to load and freeze some layer