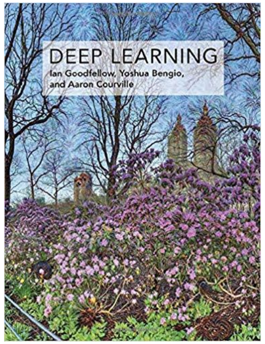
DEEP NEURAL NETWORKS (CONVOLUTIONAL NEURAL NETWORK)



By Aaron Courville, Ian Goodfellow, and Yoshua Bengio
Ch.6 Deep neural network
Ch.9 Convolutional Neural Network

TOPICS

Introduction to Deep Learning

Classical Computer Vision vs. Deep learning

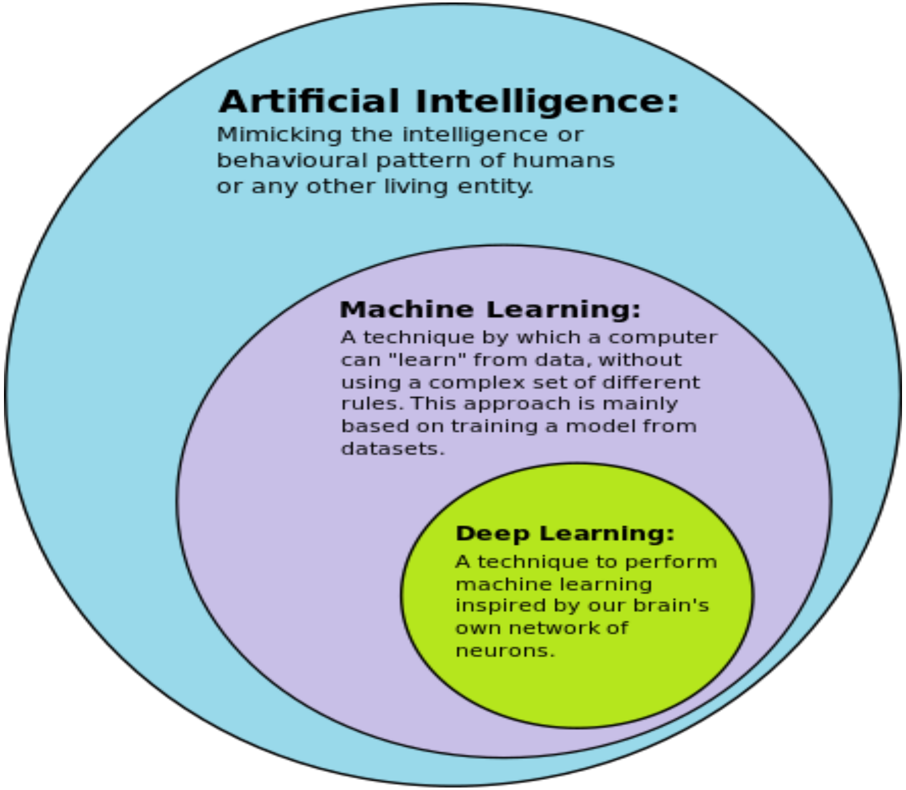
Introduction to Convolutional Networks

DEEP LEARNING

Deep learning (DL) is a subset of machine learning that uses multi-layered neural networks, called deep neural networks, to simulate the complex decision-making power of the human brain and progressively extract higher-level features from the input.

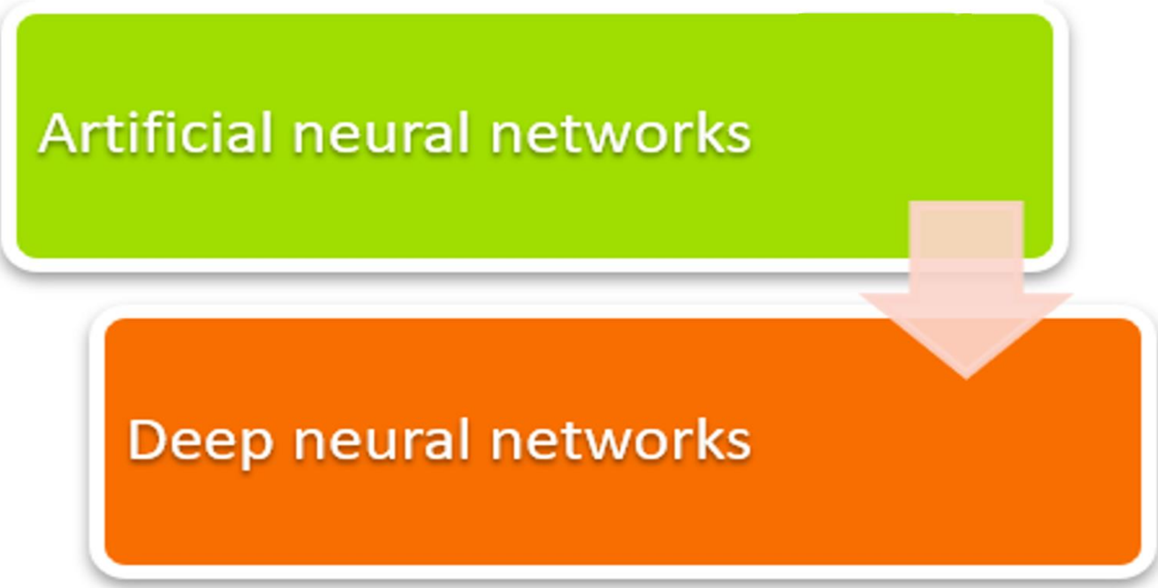
“Deep" refers to the use of multiple layers in the network.

DL can be either supervised, semi-supervised or unsupervised.



DEEP NEURAL NETWORK

A deep neural network (DNN) is an artificial neural network with multiple layers between the input and output layers.

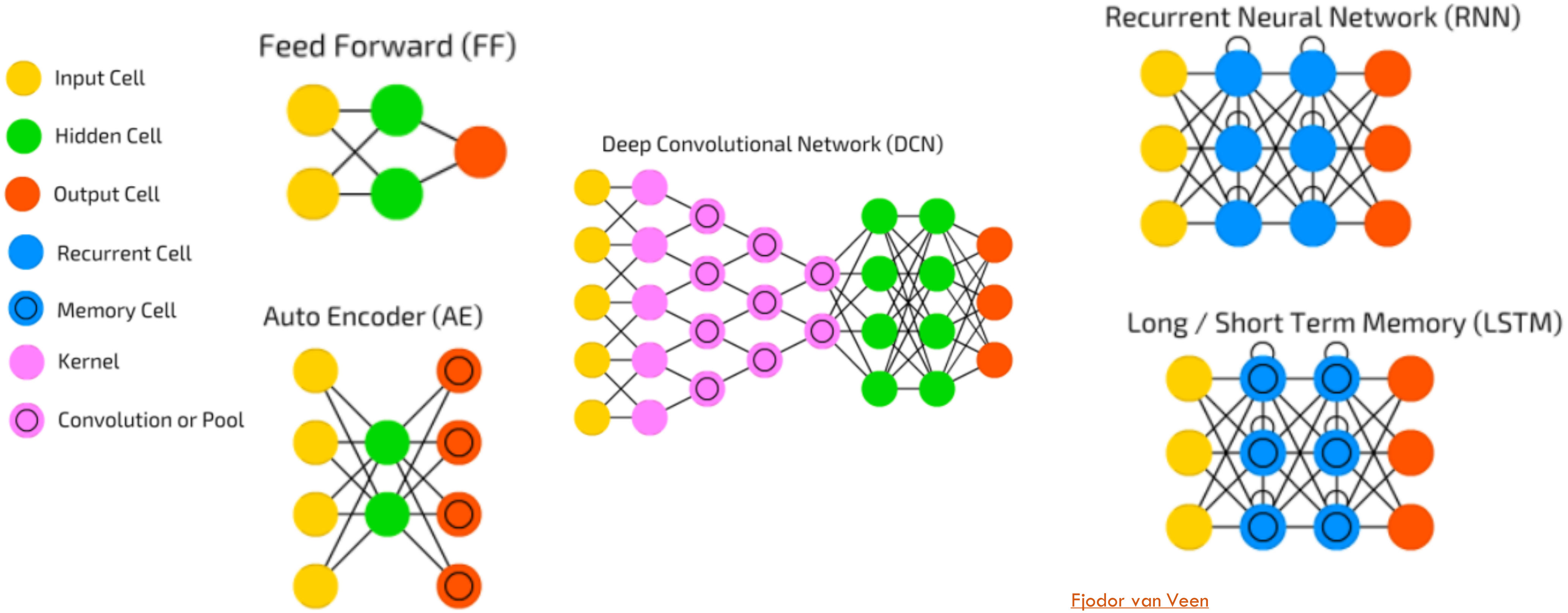


DEEP NEURAL NETWORKS TRAINING

Deep neural networks are trained in a similar manner to artificial neural networks:

- Parameter initialization
- Feedforward
- Back-propagation

ARTIFICIAL NEURAL NETWORK DIFFERENT STRUCTURE



[Fjodor van Veen](https://www.asimovinstitute.org/author/fjodorvanveen/)
https://www.asimovinstitute.org/author/fjodorvanveen/

DEEP LEARNING CHALLENGES

- **Requires large amounts of data.**

  → Solution: increase dataset using augmentation methods.

- **Prone to overfitting** because of the added layers, which allow to model rare dependencies in the training data.

  → Solution: regularization methods, dropout, or data sampling and augmentation methods.

- **Lack of reasoning.**

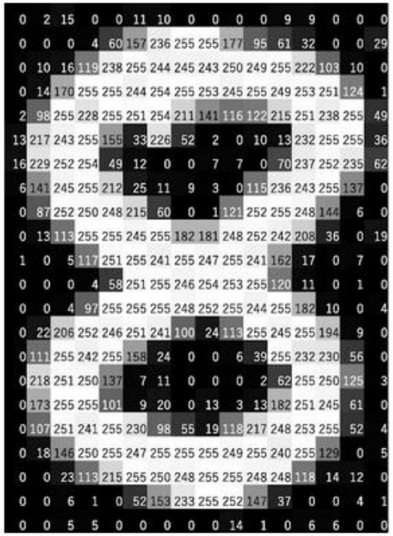
  → Solution: explainable AI (XAI).

- **Many training parameters,** (such as size: number of layers and number of units per layer, the learning rate, and initial weights). Exploring the parameter space to find the optimal parameters may not be feasible due to the cost in time and computational resources.

  → Solution: batching (computing the gradient on several training examples at once rather than individual examples), or employ Large processing capabilities (such as GPUs or the Intel Xeon Phi).

IMAGE REPRESENTATION - (GRAY SCALE IMAGE)

Represent Images in numbers or the pixel values, denote the intensity or brightness of the pixel. Smaller numbers (closer to zero) represent black, and larger numbers (closer to 255) denote white. The size of the matrix is equal to the number of pixels in the image here 22 X 16

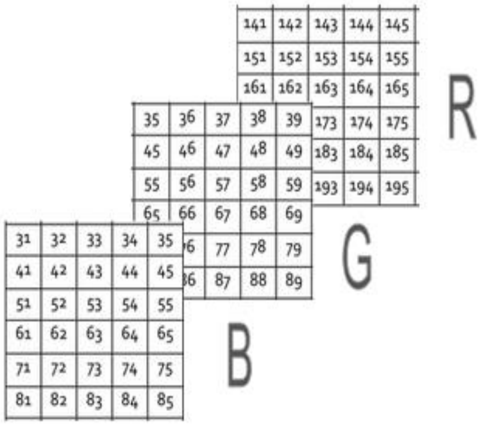


0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

IMAGE REPRESENTATION - (COLORED IMAGE)

In colored images we have three matrices for the three-color channels – Red, Green, and Blue. The three channels are superimposed to form a colored image.

Colour Image

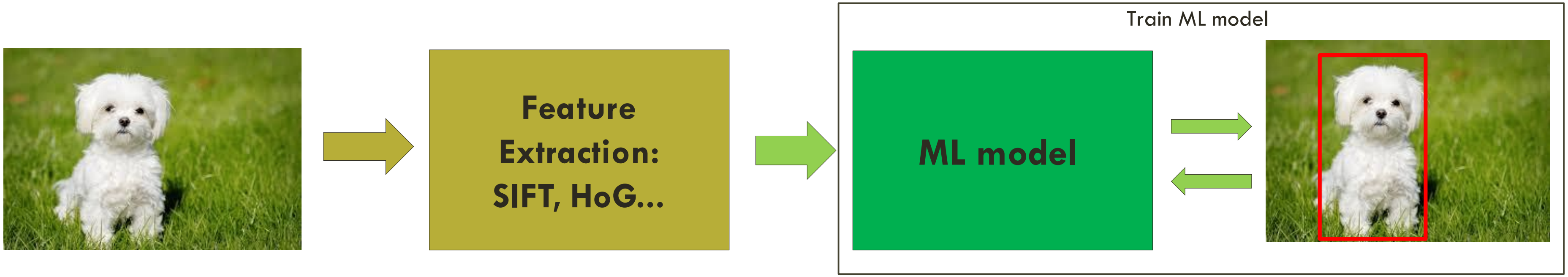


CLASSICAL COMPUTER VISION PIPELINE

1. Select / develop features:

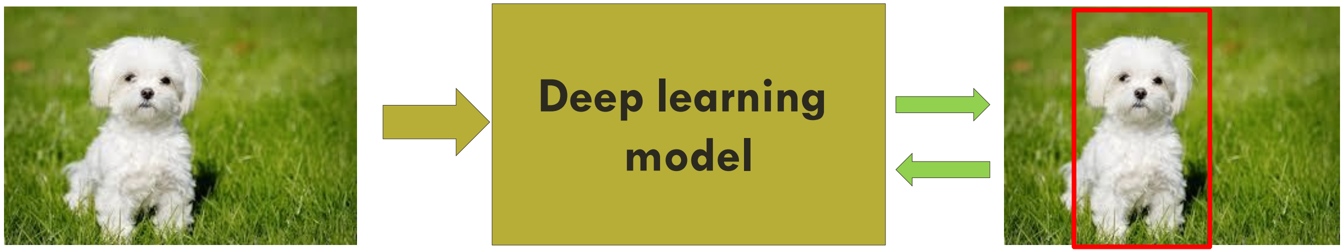
   Most common features extraction algorithms:
   - Scale-Invariant Feature Transform (SURF).
   - Histogram of Gradients (HoG).
   - Speeded-Up Robust Features (SIFT).
   - Radiation-Invariant Feature Transform (RIFT).

2. Add on top of this Machine Learning model for classification, prediction, or recognition and train it.

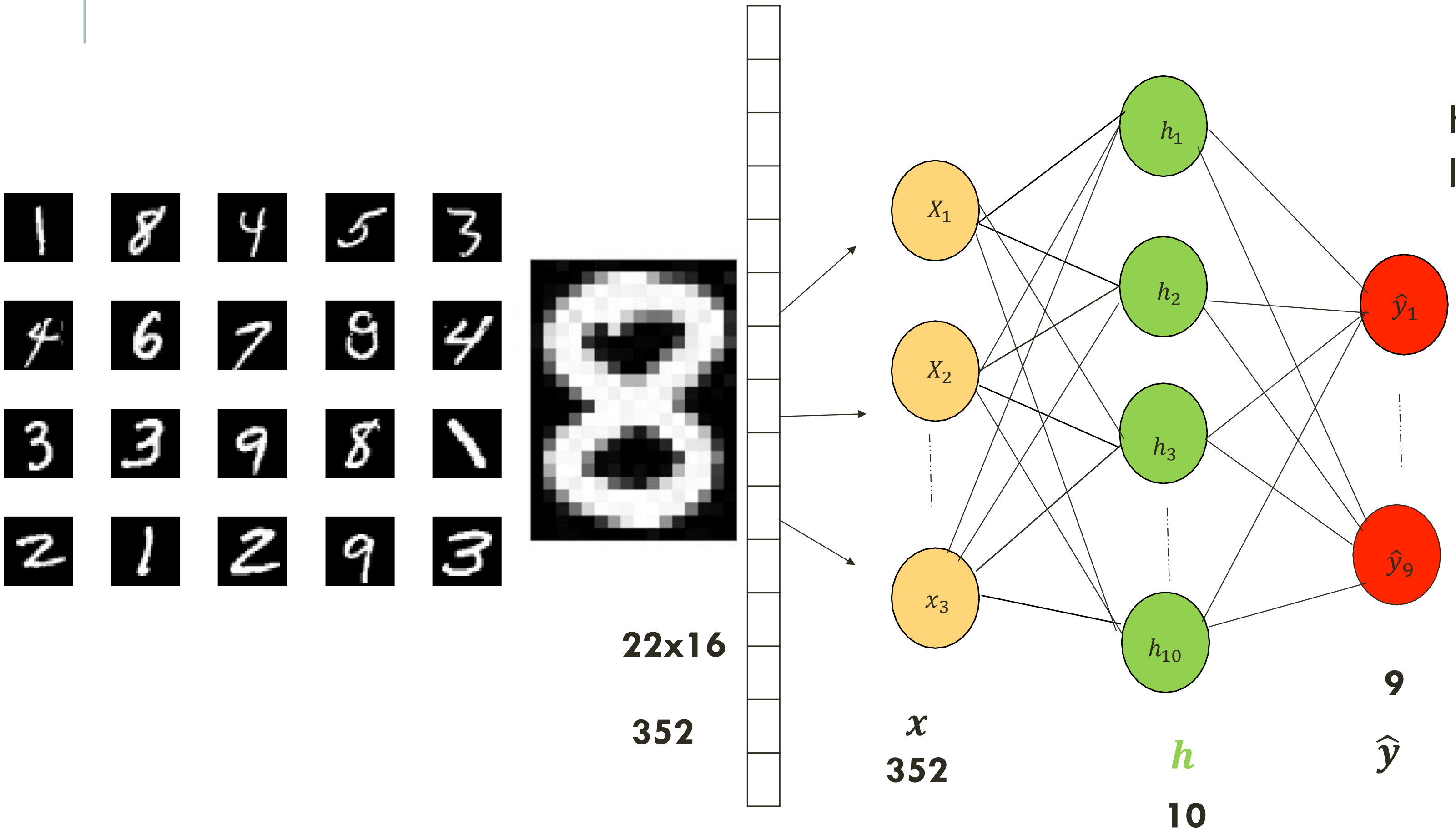


DEEP LEARNING BASED COMPUTER VISION PIPELINE.

- Build features automatically based on training data
- Combine feature extraction and classification



FEED-FORWARD NEURAL NETWORKS IN COMPUTER VERSION



How many parameters I need to estimate ?

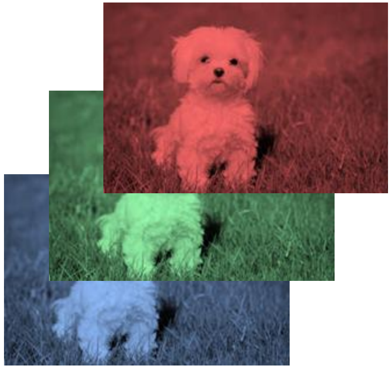
$$W^1 = 352 * 10$$

$$W^2 = 10 * 9$$

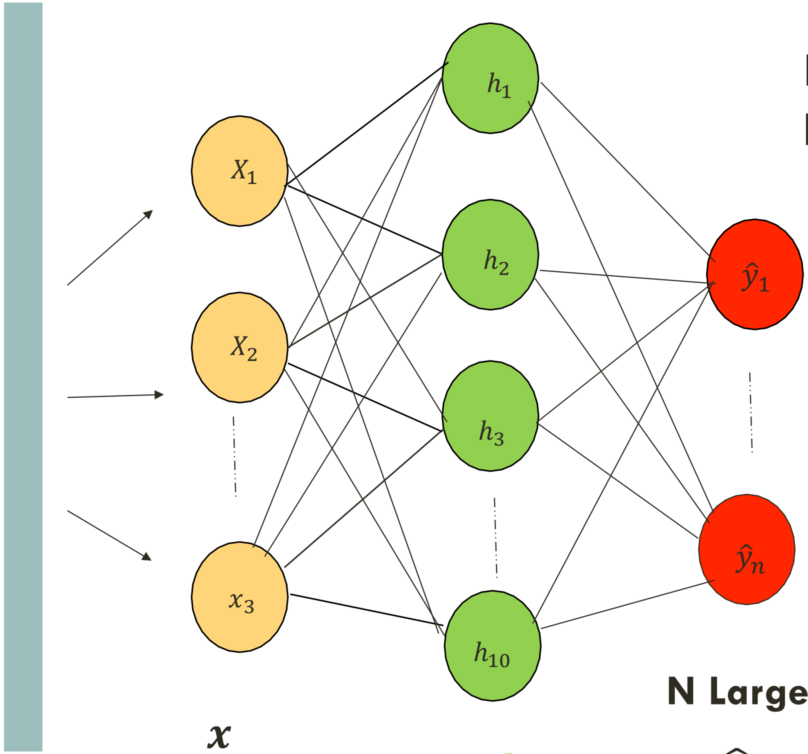
$$= 3610$$

FEED-FORWARD NEURAL NETWORKS IN COMPUTER VERSION

32X32



32X32X3



3072

$\boldsymbol{x}$

3072

$\boldsymbol{h}$

10

$\hat{\boldsymbol{y}}$

N Large number

How many parameters
I need to estimate ?

$$W^1 = 3072 * 10$$

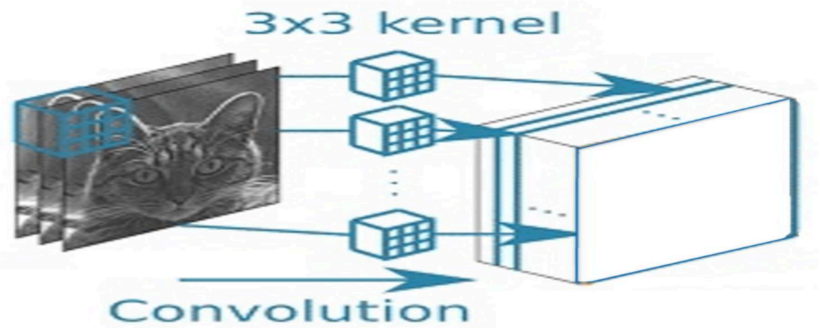
$$= \mathbf{30720}$$

Too many Parameters

WHY CONVOLUTIONAL NEURAL NETWORK ?

- **Reduce Number of parameters**
- **Edge Detection based method**

  The most important feature to classify images is the Shape. To extract the Shape features we need to extract the edges.

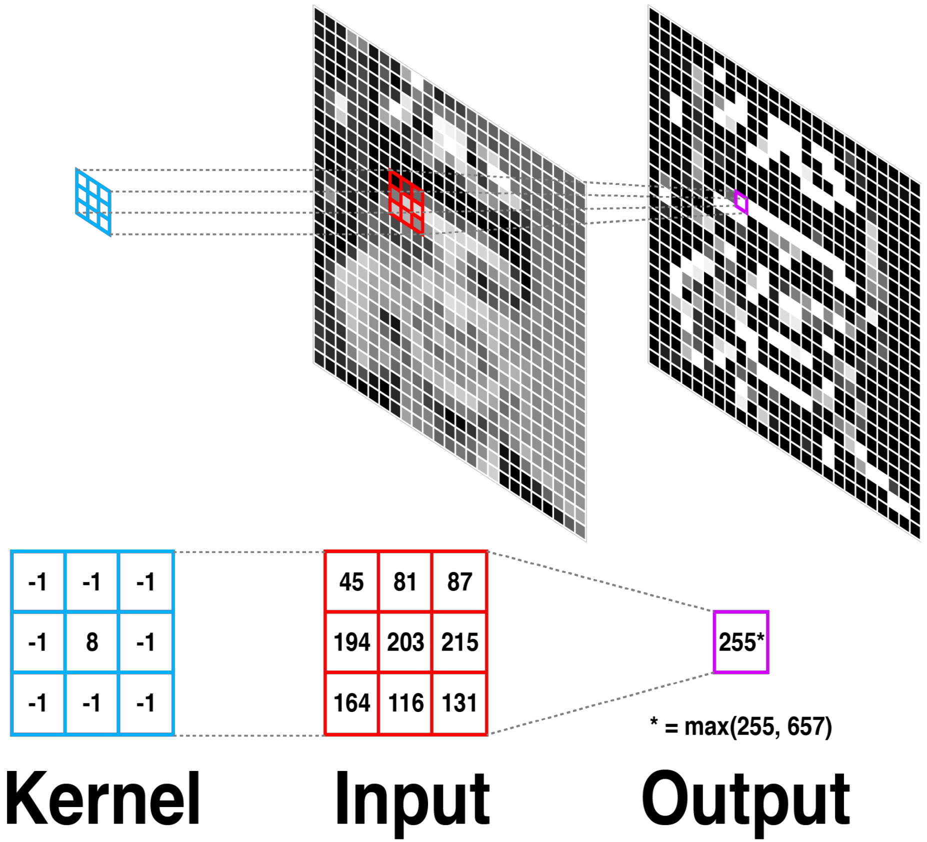


Convolutional Neural Networks (CNN) are the **most popular deep learning architecture in computer vision.**

WHAT IS CONVOLUTION?

A **convolution** operation is an element wise matrix multiplication operation. Where one of the matrices is the image, and the other is the filter or kernel that turns the image into something else. The output of this is the final convoluted image.

Convolutions have been used for a long time in image processing to blur and sharpen images, and perform other operations, such as, enhance edges and emboss.



KERNELS

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

Smoothing kernel

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

Prewitt Kernel
X Direction

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

Sobel Kernel
X Direction

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

Prewitt Kernel
Y Direction

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

Sobel Kernel
Y Direction

Edge detection kernels

0	-1	0
-1	5	-1
0	-1	0

Sharpening kernel

https://www.saama.com/different-kinds-convolutional-filters

EDGE DETECTION KERNEL EXAMPLE

Let’s take a 6X6 image of black and gray sections
To extract the edge and lower the dimension

$$\begin{array}{|c|c|c|c|c|c|}\hline 5 & 5 & 5 & 0 & 0 & 0 \\ \hline 5 & 5 & 5 & 0 & 0 & 0 \\ \hline 5 & 5 & 5 & 0 & 0 & 0 \\ \hline 5 & 5 & 5 & 0 & 0 & 0 \\ \hline 5 & 5 & 5 & 0 & 0 & 0 \\ \hline 5 & 5 & 5 & 0 & 0 & 0 \\ \hline \end{array} * \begin{array}{|c|c|c|}\hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline \end{array} = \begin{array}{|c|c|c|c|}\hline 0 & & & \\ \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline \end{array}$$

Note: this is element wise multiplication not dot product

EDGE DETECTION KERNEL EXAMPLE

Lets take a 6X6 image of black and gray sections
To extract the edge and lower the dimension

$$
\begin{array}{|c|c|c|c|c|c|}
\hline
5 & 5 & 5 & 0 & 0 & 0 \\ \hline
5 & 5 & 5 & 0 & 0 & 0 \\ \hline
5 & 5 & 5 & 0 & 0 & 0 \\ \hline
5 & 5 & 5 & 0 & 0 & 0 \\ \hline
5 & 5 & 5 & 0 & 0 & 0 \\ \hline
5 & 5 & 5 & 0 & 0 & 0 \\ \hline
\end{array}
*
\begin{array}{|c|c|c|}
\hline
1 & 0 & -1 \\ \hline
1 & 0 & -1 \\ \hline
1 & 0 & -1 \\ \hline
\end{array}
=
\begin{array}{|c|c|c|c|}
\hline
0 & 15 & & \\ \hline
 & & & \\ \hline
 & & & \\ \hline
 & & & \\ \hline
\end{array}
$$

EDGE DETECTION KERNEL EXAMPLE

Lets take a 6X6 image of black and gray sections
To extract the edge and lower the dimension

$$
\begin{array}{|c|c|c|c|c|c|}
\hline
5 & 5 & 5 & 0 & 0 & 0 \\
\hline
5 & 5 & 5 & 0 & 0 & 0 \\
\hline
5 & 5 & 5 & 0 & 0 & 0 \\
\hline
5 & 5 & 5 & 0 & 0 & 0 \\
\hline
5 & 5 & 5 & 0 & 0 & 0 \\
\hline
5 & 5 & 5 & 0 & 0 & 0 \\
\hline
\end{array}
*
\begin{array}{|c|c|c|}
\hline
1 & 0 & -1 \\
\hline
1 & 0 & -1 \\
\hline
1 & 0 & -1 \\
\hline
\end{array}
=
\begin{array}{|c|c|c|c|}
\hline
0 & 15 & 15 & 0 \\
\hline
 & & & \\
\hline
 & & & \\
\hline
 & & & \\
\hline
\end{array}
$$

EDGE DETECTION KERNEL EXAMPLE

Lets take a 6X6 image of black and gray sections
To extract the edge and lower the dimension

5	5	5	0	0	0
5	5	5	0	0	0
5	5	5	0	0	0
5	5	5	0	0	0
5	5	5	0	0	0
5	5	5	0	0	0

6X6 Input

*

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

3x3 filter/ kernel/ feature detector.

=

0	15	15	0
0	15	15	0
0	15	15	0
0	15	15	0

4X4 convolved feature
Feature map

CONVOLUTION LAYER

The main building block of CNN is the convolutional layer.

Apply Convolutional Kernel (filter) at every location in the input, do element-wise matrix multiplication and sum the result (linear combination). This sum goes into the feature map.

The primary purpose of convolution layer is to extract features from the input.

Each convolution filter acts as a detector for a particular feature.

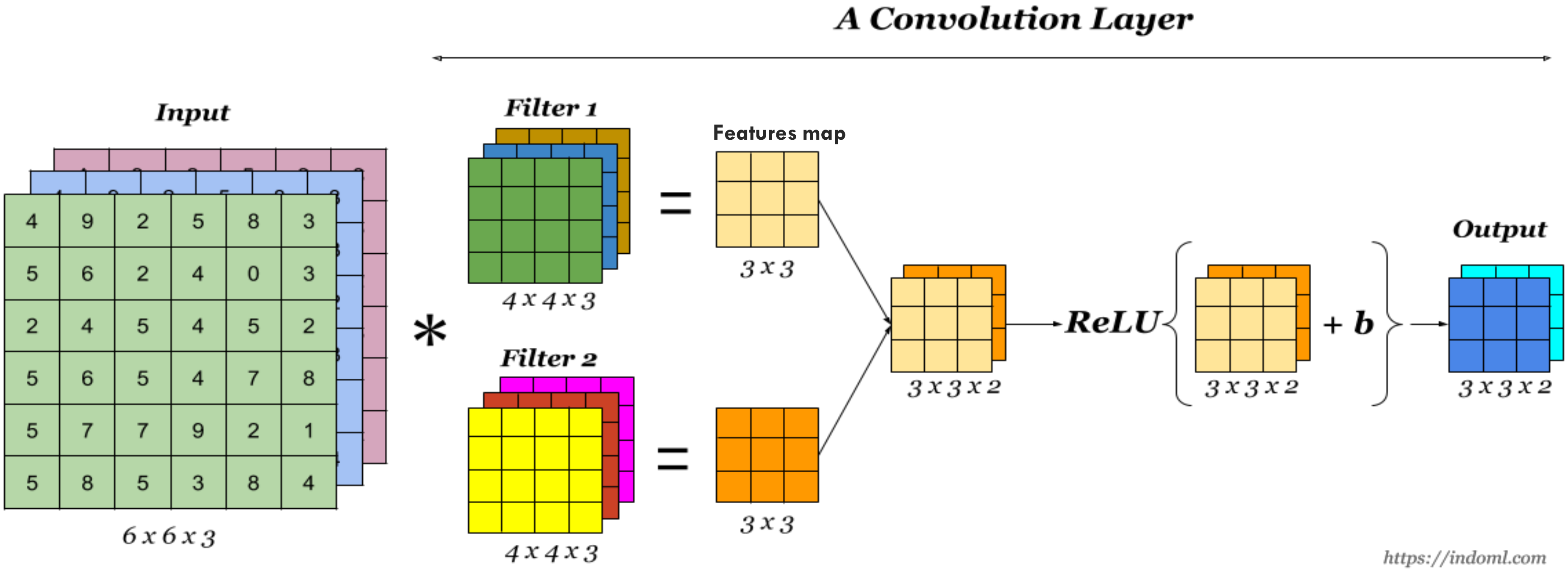
1x1	1x0	1x1	0	0
0x0	1x1	1x0	1	0
0x1	0x0	1x1	1	1
0	0	1	1	0
0	1	1	0	0

4		

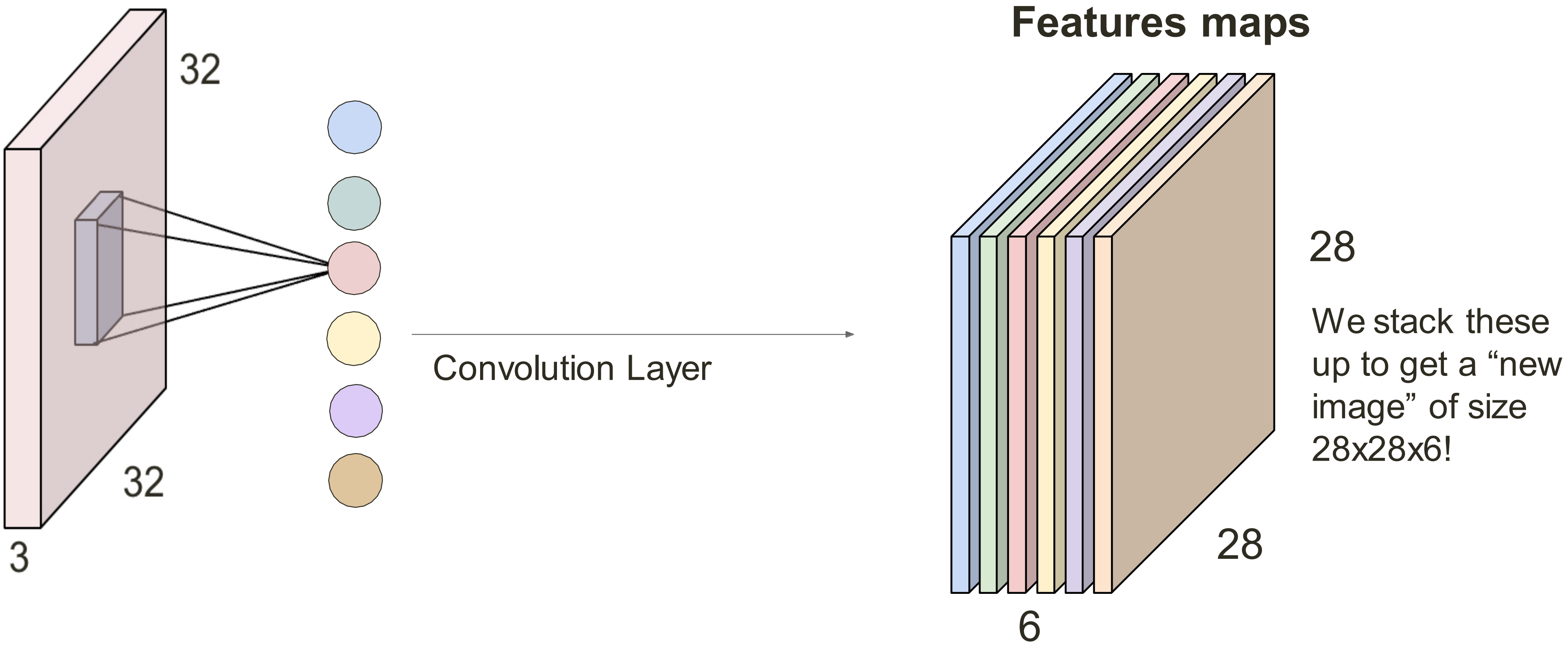
FEATURES (ACTIVATION) MAP

Feature map is the output of a convolutional layer.

performing multiple **convolutions on an input** using different filters **resulting in distinct feature maps.**

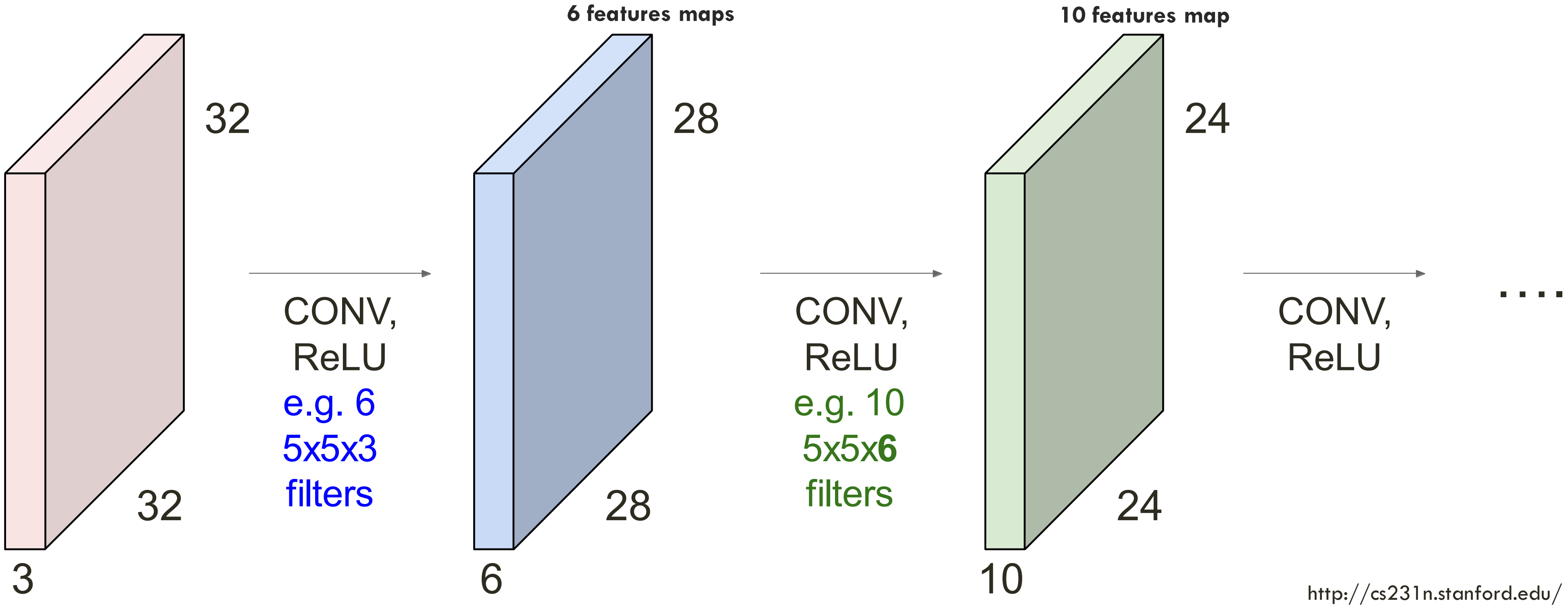


FOR EXAMPLE, IF WE HAD 6 5X5 FILTERS, WE'LL GET 6 SEPARATE FEATURES MAPS:

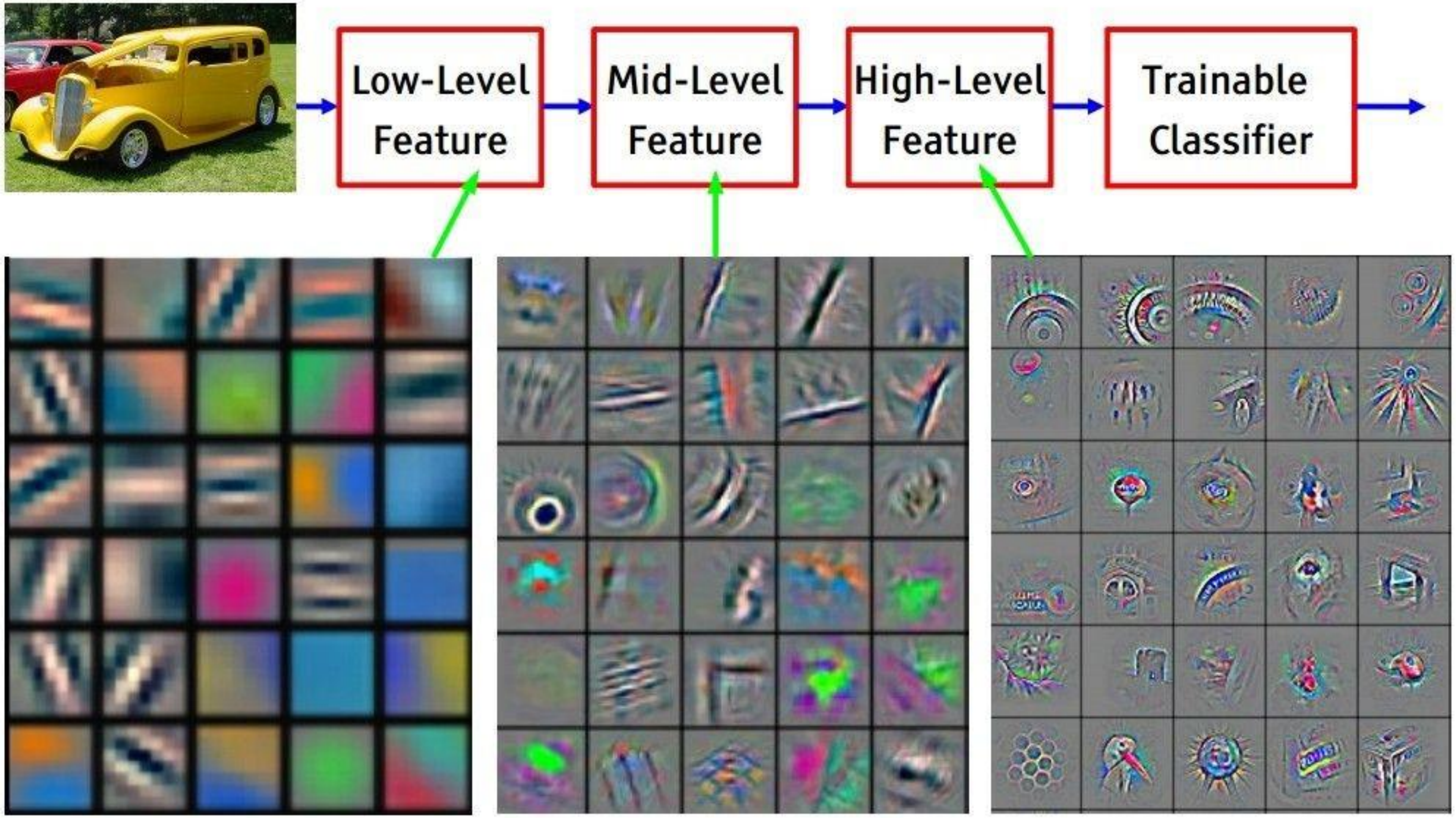


http://cs231n.stanford.edu/

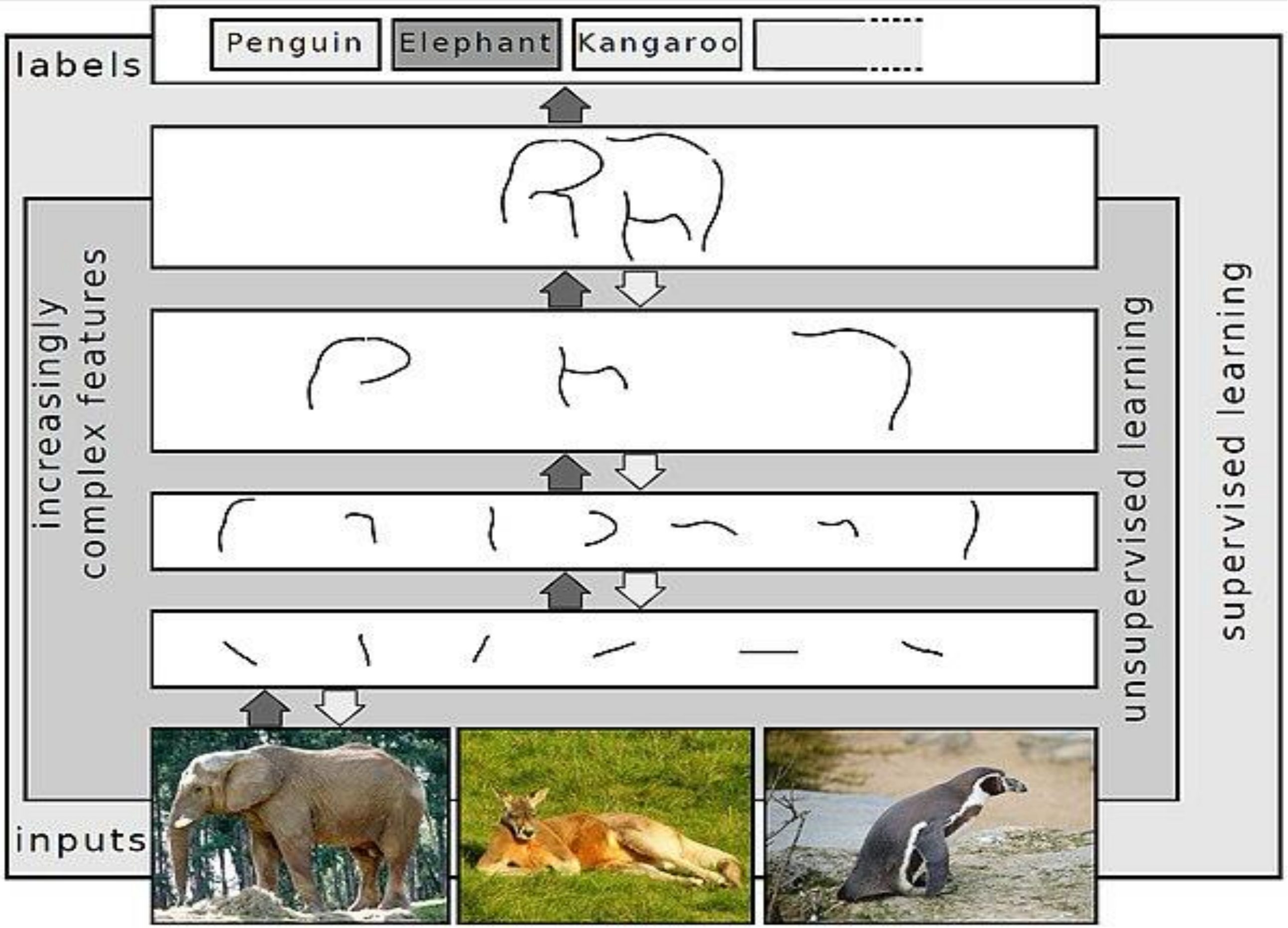
PREVIEW: CONVNET IS A SEQUENCE OF CONVOLUTIONAL LAYERS, INTERSPERSED WITH FEATURES FUNCTIONS



http://cs231n.stanford.edu/



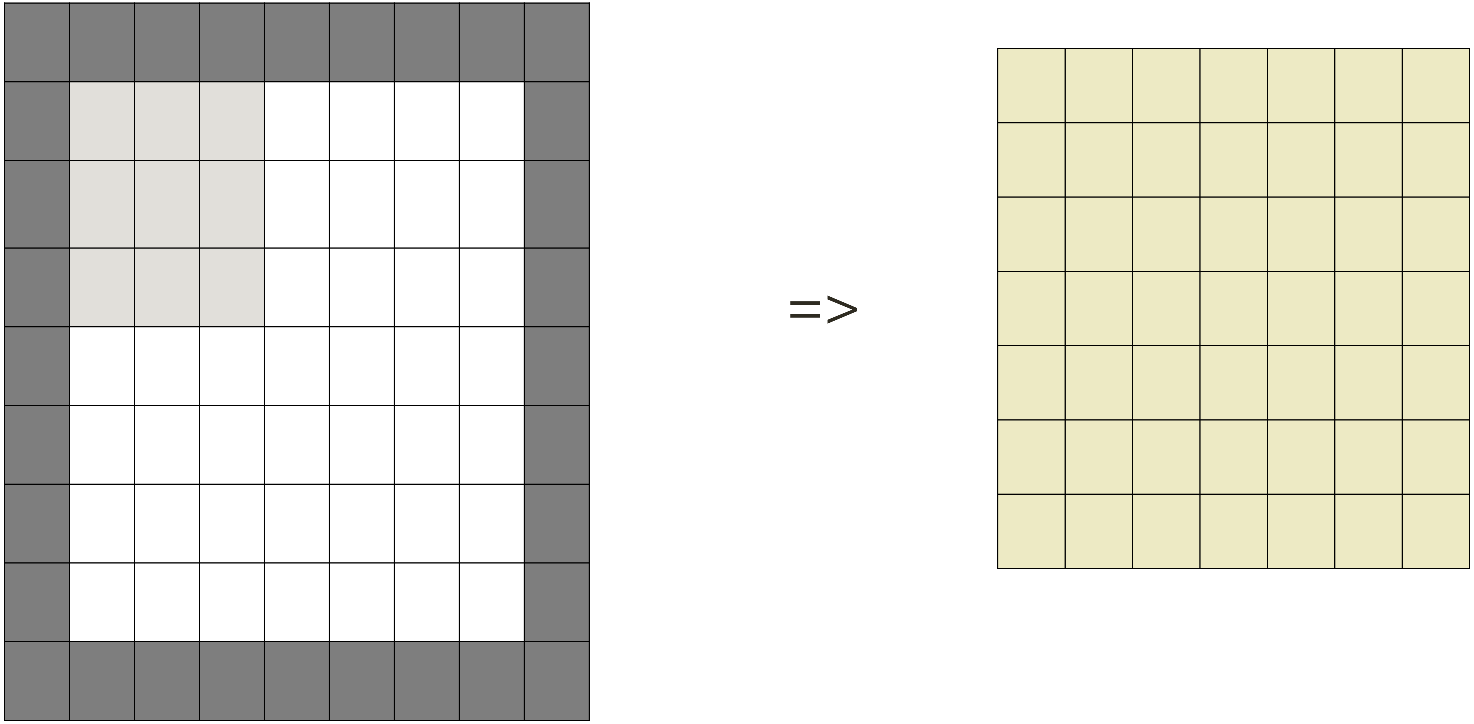
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]



labels
Penguin
Elephant
Kangaroo
increasingly complex features
unsupervised learning
supervised learning
inputs

PADDING

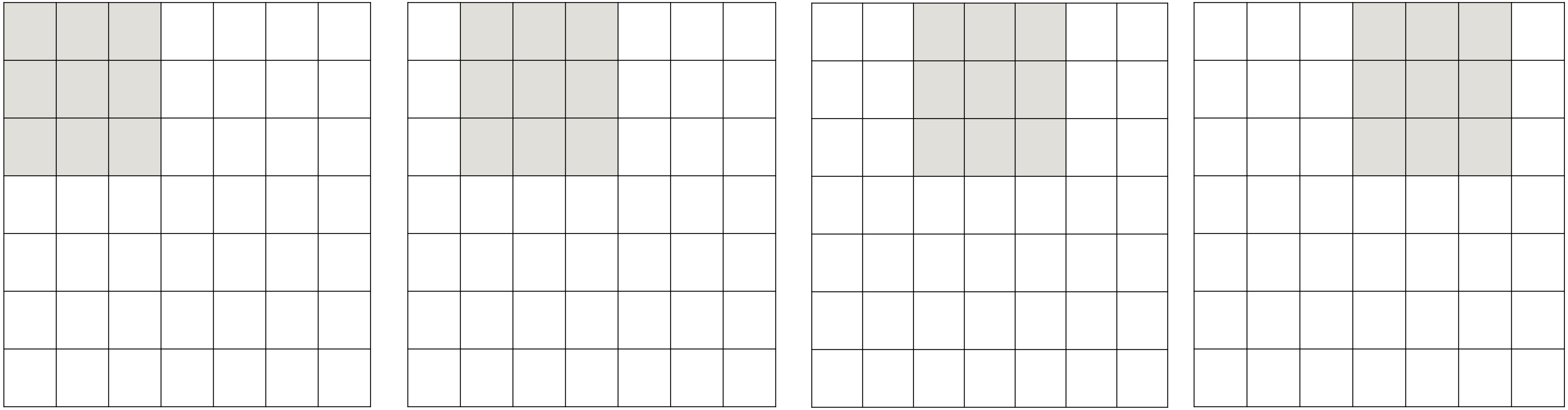
To maintain the dimension of output as in input, we add **padding** to the input matrix. either **with zeros or the values on the edge.**



FILTERS STRIDE

Stride **specifies how much the convolution filter is moved at each step** (by **default** the **value is 1**).

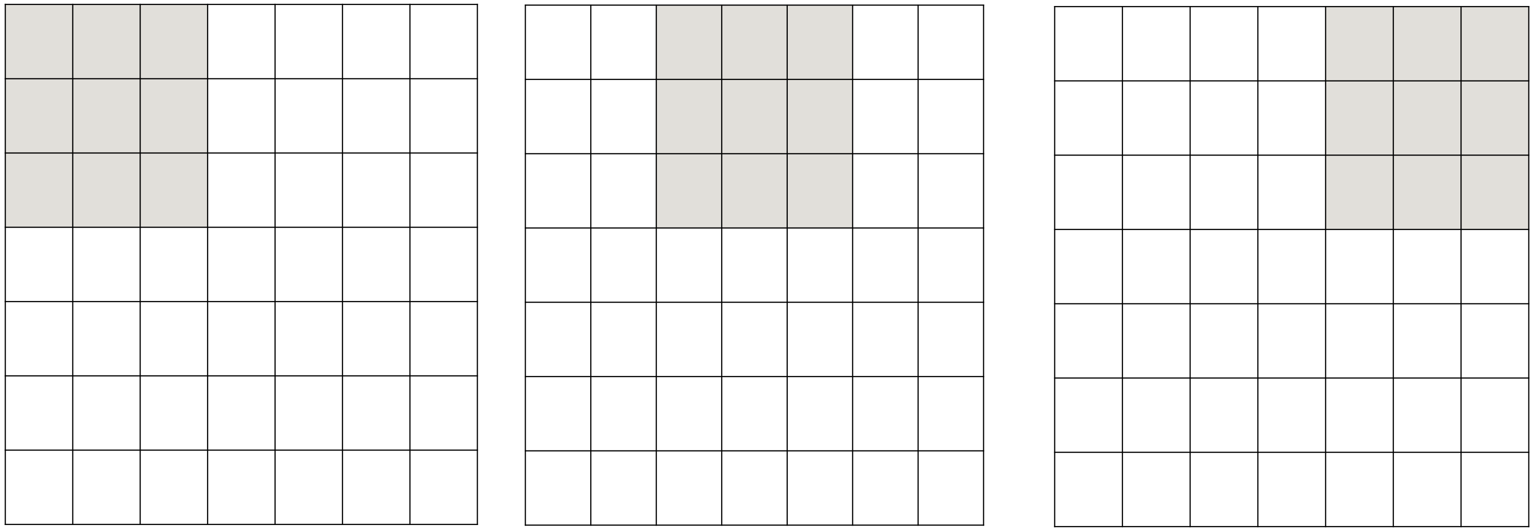
7X7 images with 3X3 Filter and 1 Stride



The output will be 5x5

FILTERS STRIDE

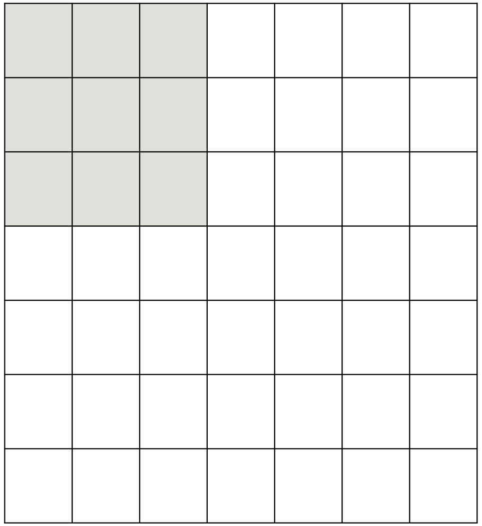
7X7 images with 3X3 Filter and Stride 2



The output will be 3x3

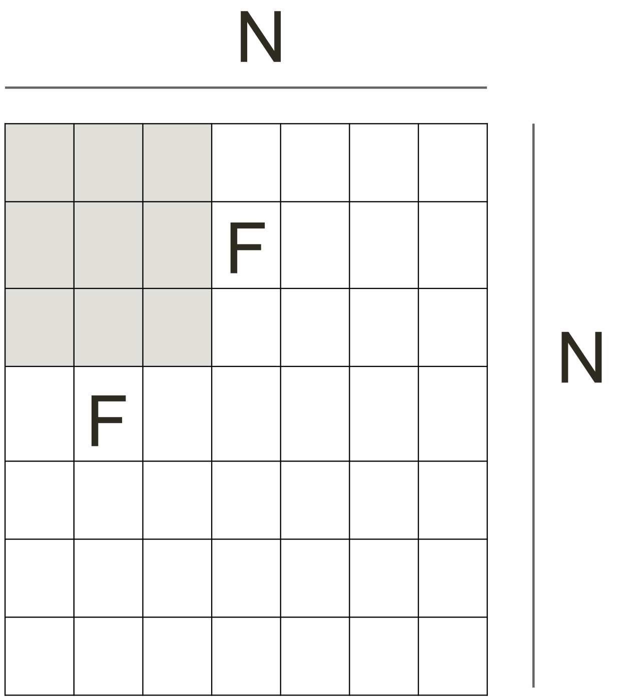
FILTERS STRIDE

7X7 images with 3X3 Filter and Stride 3



cannot apply 3x3 filter on 7x7 input with stride 3.

FEATURES (ACTIVATION) MAP SIZE



To compute size of features map:

(N + 2P - F) / stride + 1

e.g. N = 7, F = 3, P = 0:

stride 1 => (7 + 0 – 3)/1 + 1 = 5

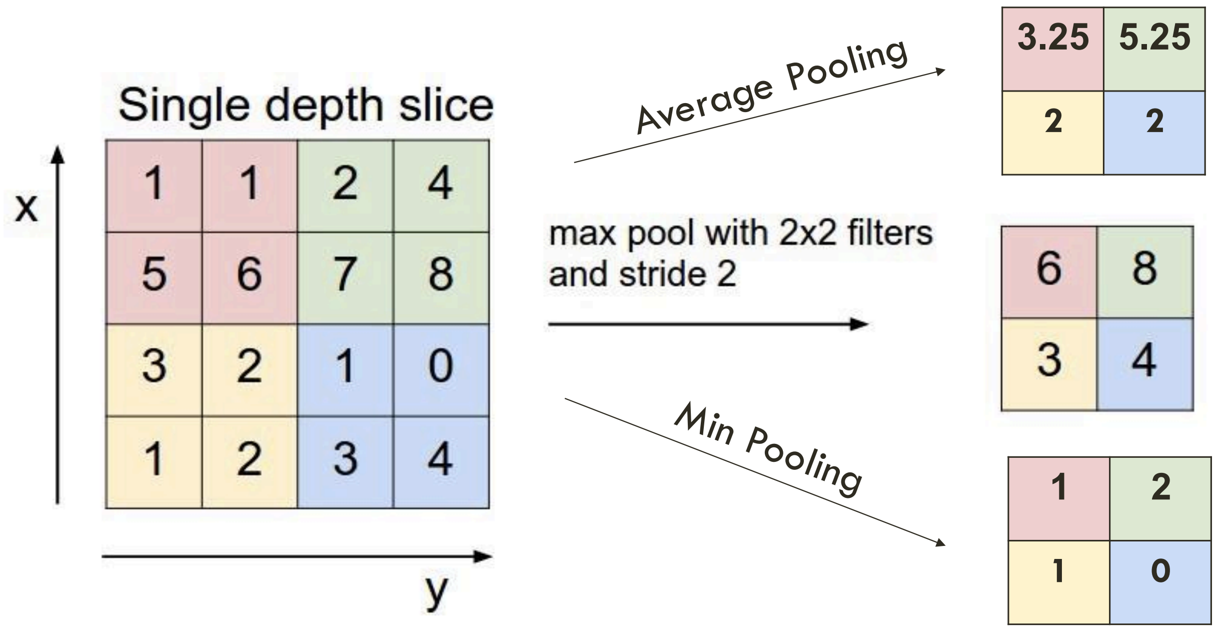
Stride 2 => (7 + 0 - 3)/2 + 1 = 3

stride 3 => (7 + 0 - 3)/3 + 1 = 2.33

POOLING LAYER

It comes between two convolutional layers to reduce the dimensionality of the feature space

- Average Pooling
- Max Pooling
- Min Pooling

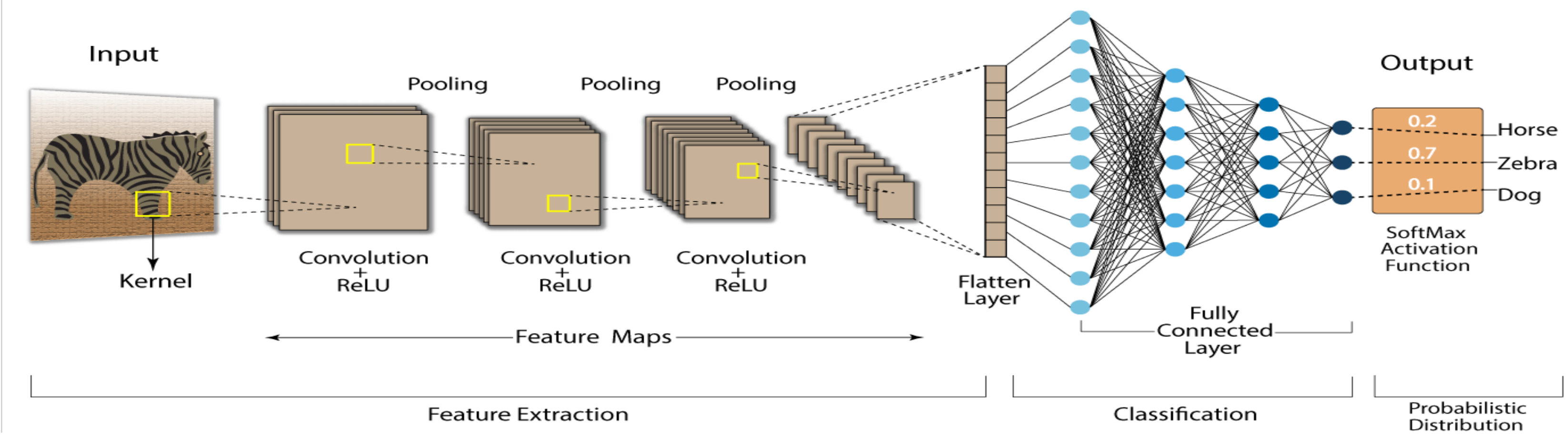


Source : CS231n Convolutional Neural Network

CNN ARCHITECTURE

A CNN model can be thought as a **combination of two components: feature extraction and the classification.**

- The **convolution and pooling layers perform feature extraction.**
- The **fully connected layers** act as a **classifier on top of these features.** They **learn** how **to use the features map** to **correctly classify the images** by **assigning a probability** for the input image class.



CNN ARCHITECTURES CONFIGURATION

Convolutional Networks are made up of only three types of layers: CONV, POOL (we assume Max pool unless stated otherwise) and FC (short for fully-connected).

Layer Patterns

INPUT → [CONV → RELU]*N → [POOL?]*M → [FC RELU]*K → FC

INPUT → [CONV → RELU → POOL]*2 →FC → Relu → FC

INPUT → [CONV → RELU → CONV → RELU → POOL]*3 → [FC → RELU]*2 → FC

CNN ARCHITECTURES CONFIGURATION

Layer Sizing Patterns

- **Input layer:** It is recommended to have the input layer size be a multiple of 2. Common numbers are 32 (e.g. CIFAR-10), 64, 96 (e.g. STL-10), 224 (e.g. common ImageNet ConvNets), 384, and 512.
- **The conv layers** should use small filters (e.g. 3x3 or at most 5x5) at first, using a stride of **S=1**, and padding the input volume with zeros to not alter the spatial dimensions of the input.
- The **pool layers** The most common setting is to use max-pooling with 2x2 receptive fields (i.e. F=2), and with a stride of 2 (i.e. S=2).

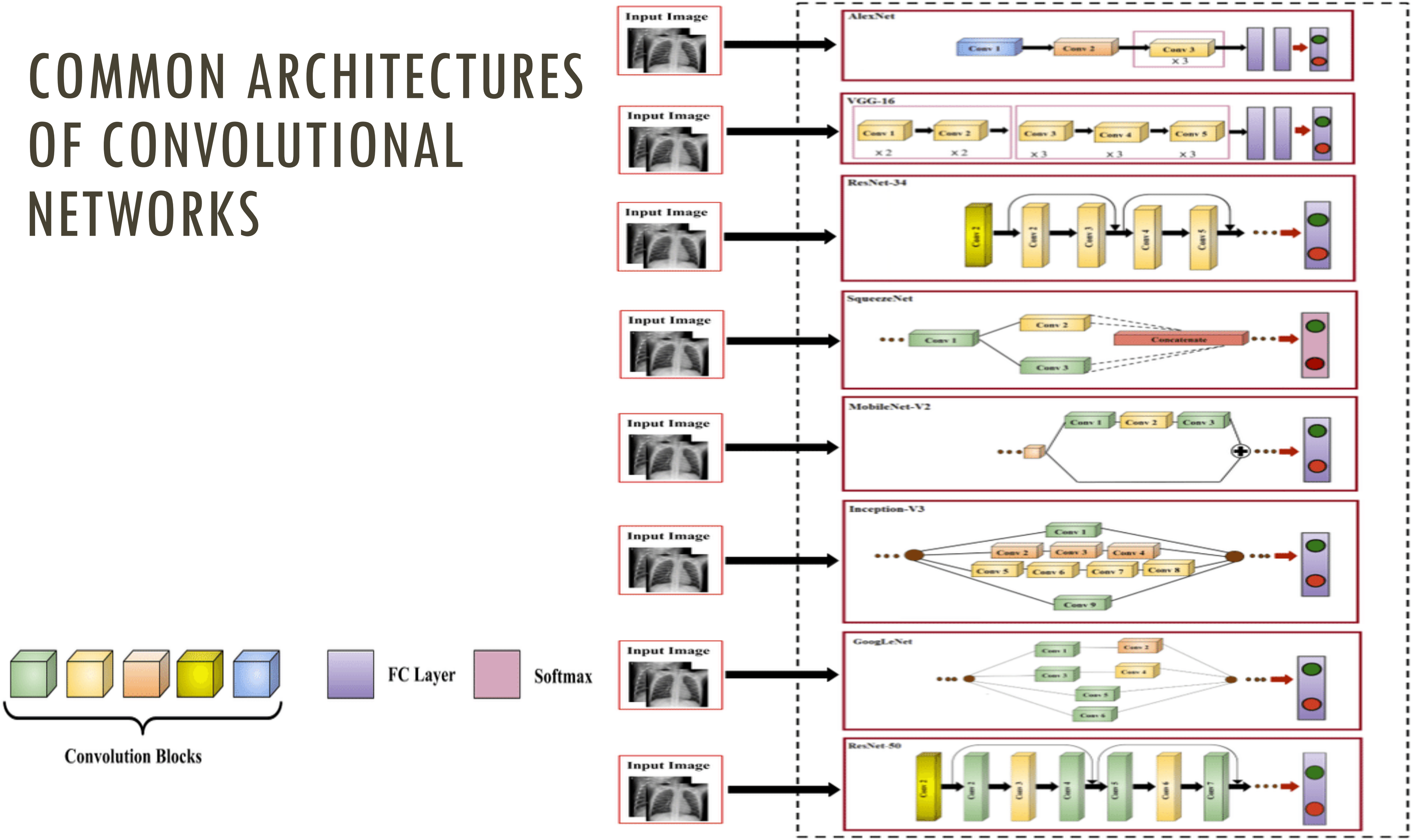
CNN TRAINING

CNN is trained the same way like ANN,
back-propagation with gradient descent.

Architecture: 4 convolution + pooling layers, followed by 2 fully connected layers. The input is an image of size (150, 150, 3) and the output is binary.

```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', name='conv_1', input_shape=(150, 150, 3)))
model.add(MaxPooling2D((2, 2), name='maxpool_1'))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same', name='conv_2'))
model.add(MaxPooling2D((2, 2), name='maxpool_2'))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', name='conv_3'))
model.add(MaxPooling2D((2, 2), name='maxpool_3'))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', name='conv_4'))
model.add(MaxPooling2D((2, 2), name='maxpool_4'))
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(512, activation='relu', name='dense_1'))
model.add(Dense(128, activation='relu', name='dense_2'))
model.add(Dense(1, activation='sigmoid', name='output'))
```

COMMON ARCHITECTURES OF CONVOLUTIONAL NETWORKS

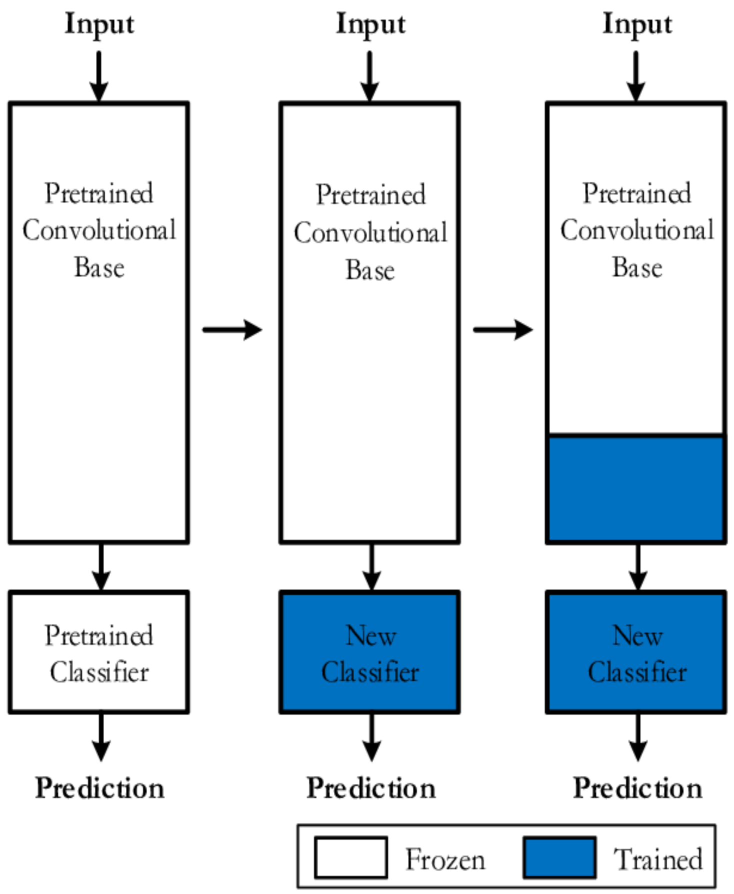


TRANSFER LEARNING

You need a lot of a data if you want to train/use CNNs?

Transfer learning: the reuse of a previously learned model (pre-trained) on a new problem.

The weights of a Neural Network created for a particular problem are used for another such problem.

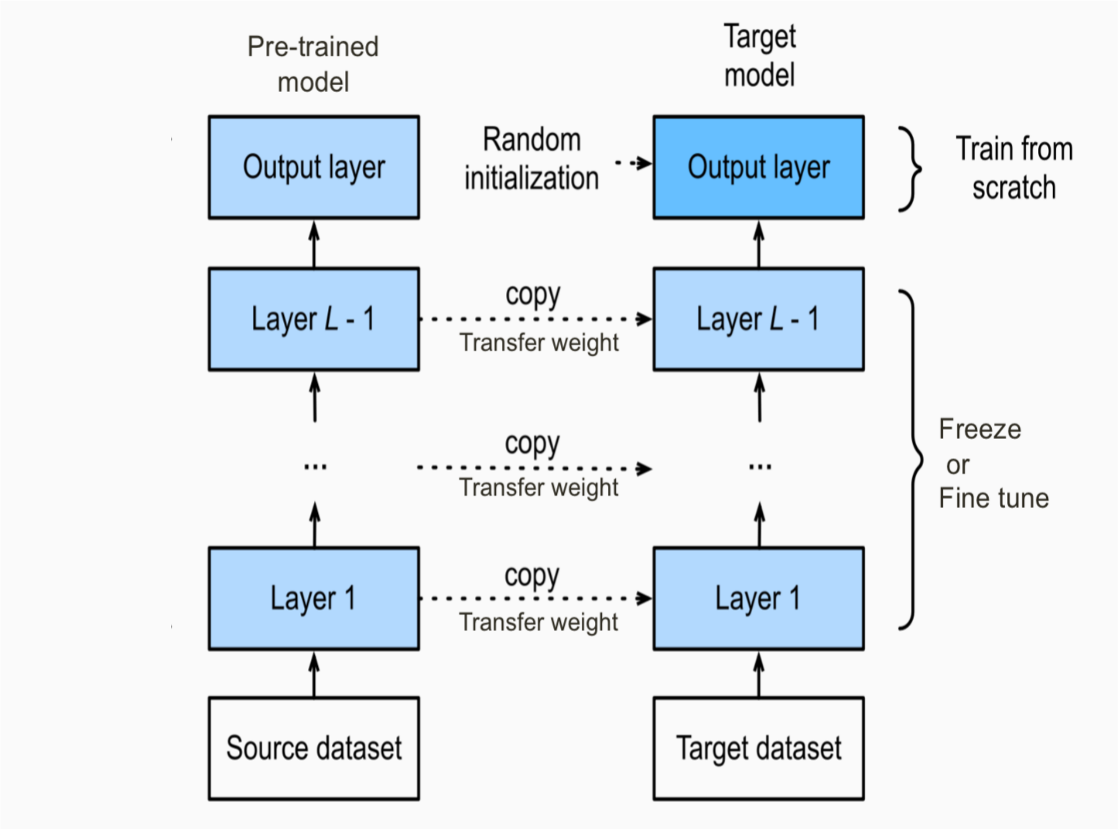


TRANSFER LEARNING

	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

MAIN STEPS TO IMPLEMENT TRANSFER LEARNING:

1. Obtain the pre-trained model.
2. Create a base model from the pre-trained model.
3. Download weights of pre-trained layers.
4. Freeze layers so they do not change during training.
5. Add new trainable layers.
6. Train the new layers on the data set with a large learning rate.
7. Improve the model via fine-tuning the pre-trained model with a very low learning rate.



```
# Importing deep learning libraries
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, BatchNormalization, Conv2D, MaxPool2D,Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
Import tensorflow as tf

# Loading pre-trained model
vgg19 = tf.keras.applications.vgg19.VGG19()
vgg19.summary()

# Add layers of pre trained model
vgg_model = Sequential()
For layer in vgg19.layers[:-1]:
          vgg_model.add(layer)

# Freezes the weights and other trainable parameters  of retrained layers
for layer in vgg_model.layers:
           layer.trainable = False

# Add new output layers
vgg_model.add(Dense(units=2, activation="softmax"))

# Compile and train model
vgg_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
vgg_model.fit(x = train_batches, steps_per_epoch = len(train_batches), epochs=5, callbacks=[early_stopping])

# Test model
predVGG19 = vgg_model.predict(x=test_batches)
test_loss, test_acc = vgg_model.evaluate(test_batches, verbose=2)
```

TRANSFER LEARNING

Transfer learning with CNNs is pervasive… But recent results show it might not always be necessary!

Training from scratch can work just as well as training from a pre-trained model for object detection. But it takes 2-3x as long to train.

Collecting more data is better than fine tuning on a related task