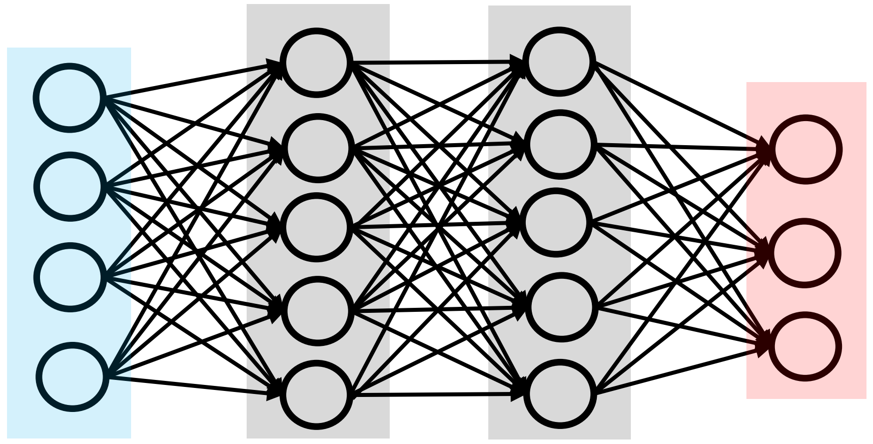
COS30082
Applied Machine Learning

Lecture 5
Convolutional Neural Network

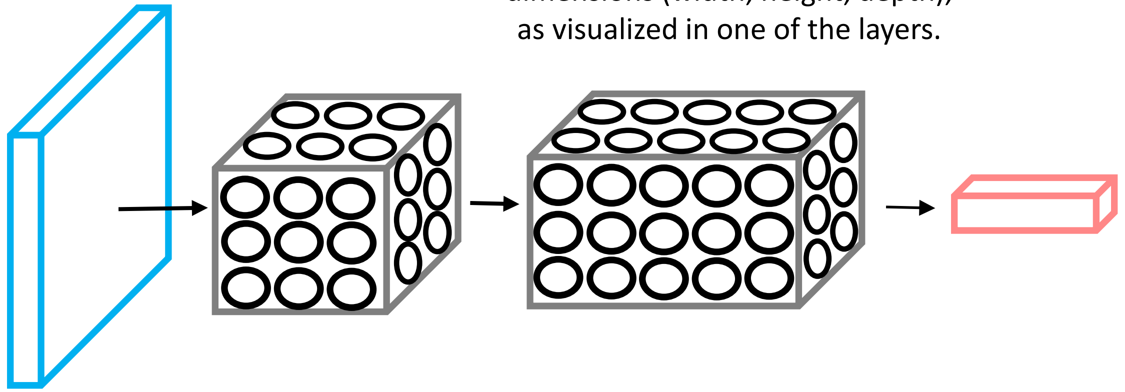
Objective of this lecture

- To understand what is a convolutional neural network (CNN)
- To learn different optimization techniques
- To explore the state-of-the-art CNN architectures.
- To learn the deep learning design process.
- CNN with Keras

Convolutional Neural Network



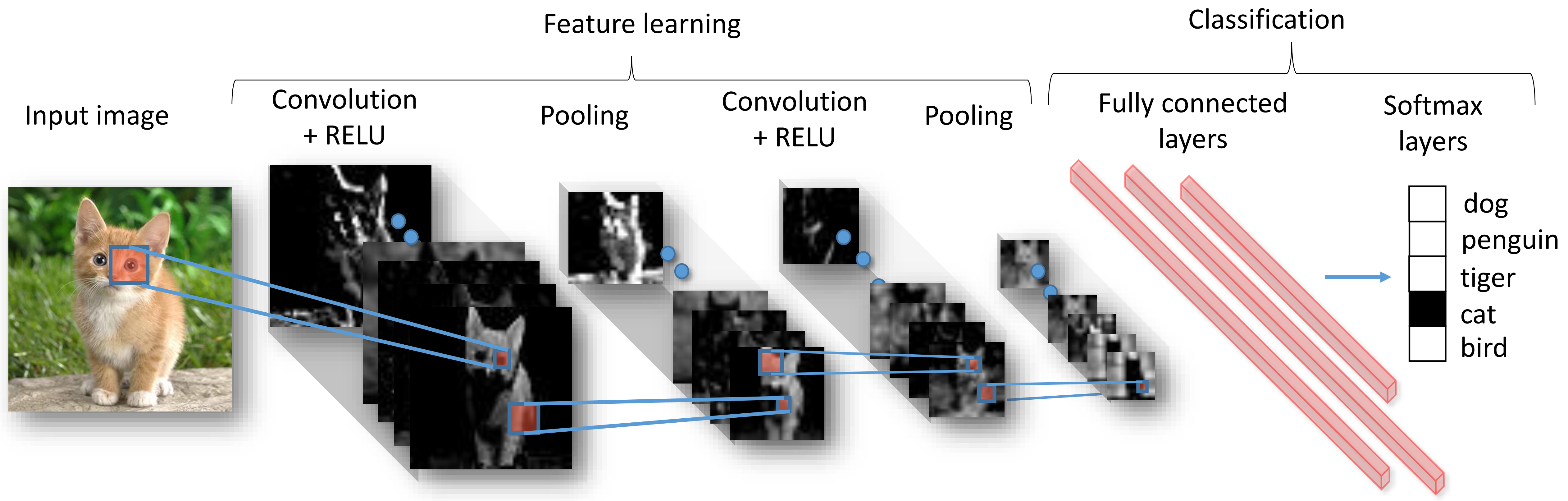
ANN with 2 hidden layers


CNN arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers.

CNN with 2 convolutional layers

- Convolutional neural network (ConvNets or CNNs) is a deep learning model which was initially designed to work with two-dimensional image data. It can be used also with both one and three-dimensional data.
- CNN are particularly well-suited for data that has a **spatial (or temporal) locality relationship**. This characteristic is most evident in tasks involving images, videos, and even audio or time series data, where the relationship between nearby elements is crucial for understanding the overall structure and content of the data.
- CNNs are used mainly for image processing, classifications, segmentation, object detections and also for other auto correlated data.

CNN architecture



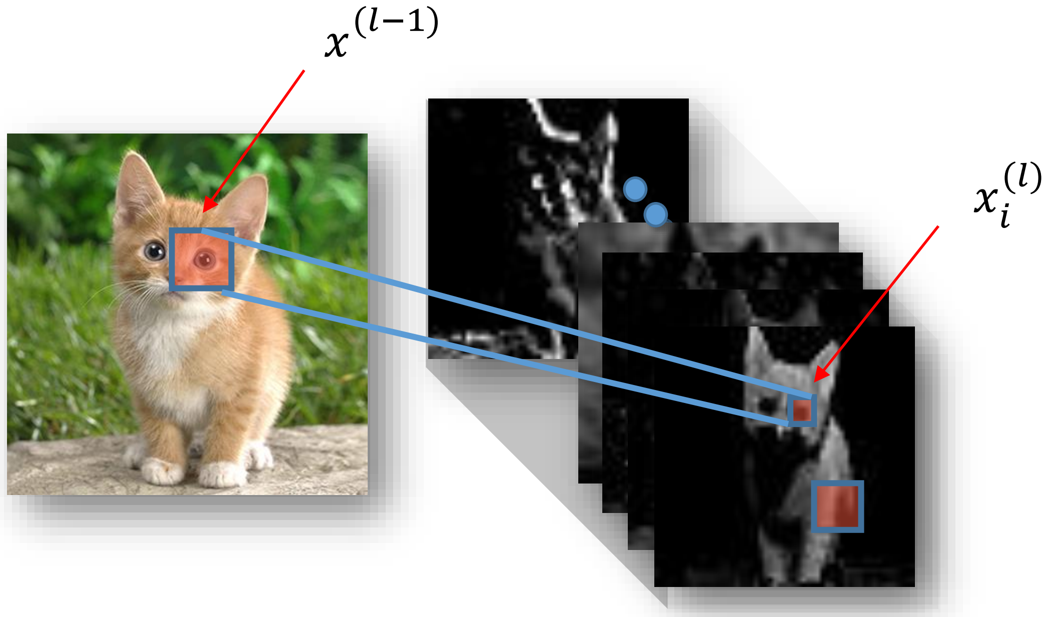
- In a typical CNN, each input image will pass through a series of **convolution layers** with filters (or **kernals**), **pooling**, **fully connected layers** (FC) and apply the **softmax** function to classify an object with probabilistic values between 0 and 1.

Convolutional layer

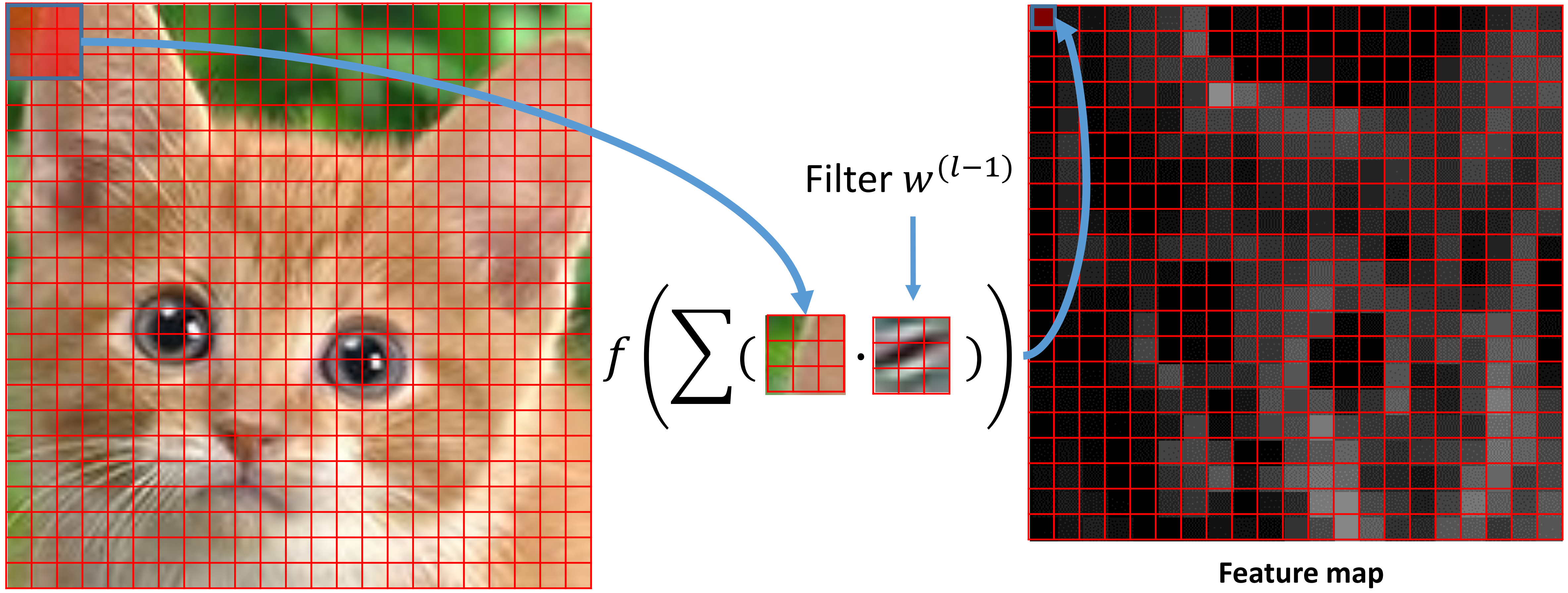
- Feature detectors
- Generate feature maps by convolving input with filters
- To compute the pixel value i of a r-th feature map for layer l, we use,

$$x_i^{(l)} = g(\sum_{j=1}^{k} x_j^{(l-1)} * w_j^{(l-1)} + b_i^{(l-1)})$$

where:
$g(\cdot)$ is an activation function
$x^{(l-1)}$ is one of the input regions
$w^{(l-1)}$ is the filter with size of $N \times N$
b_i is the bias
k is the total pixel of filter
$*$ elements-wise multiplication



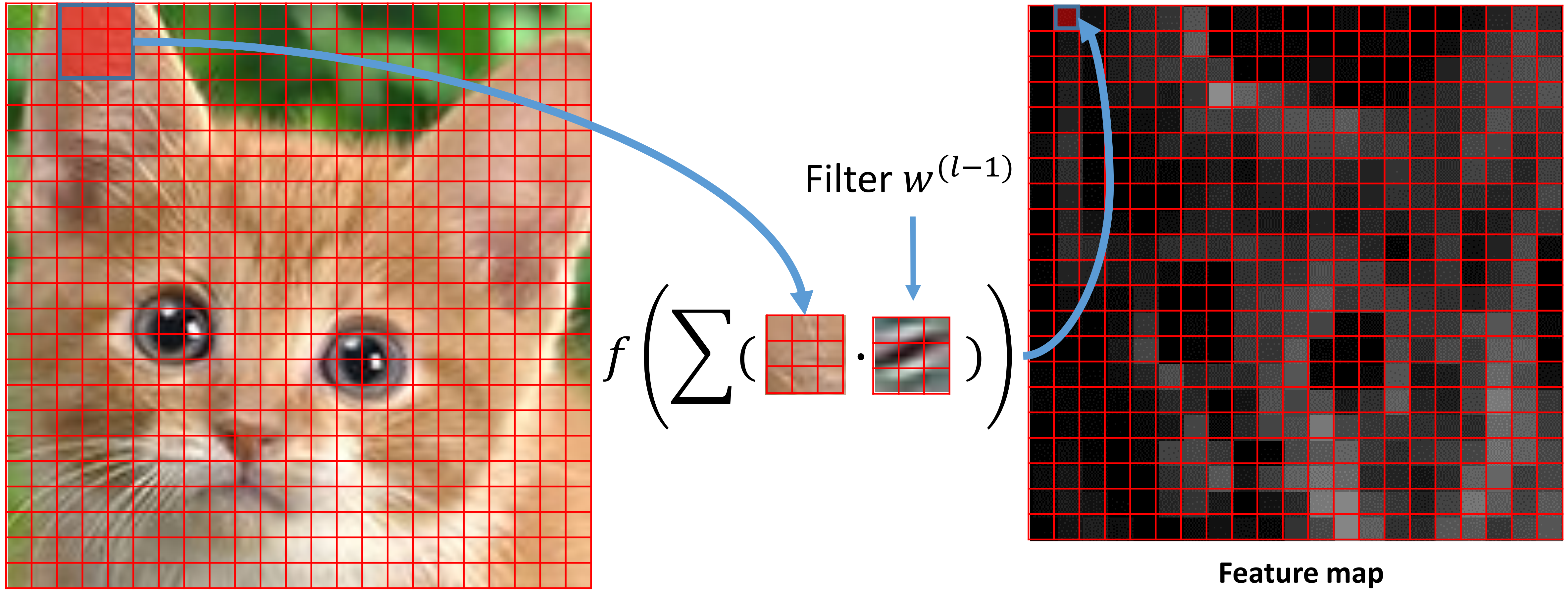
Convolutional layer



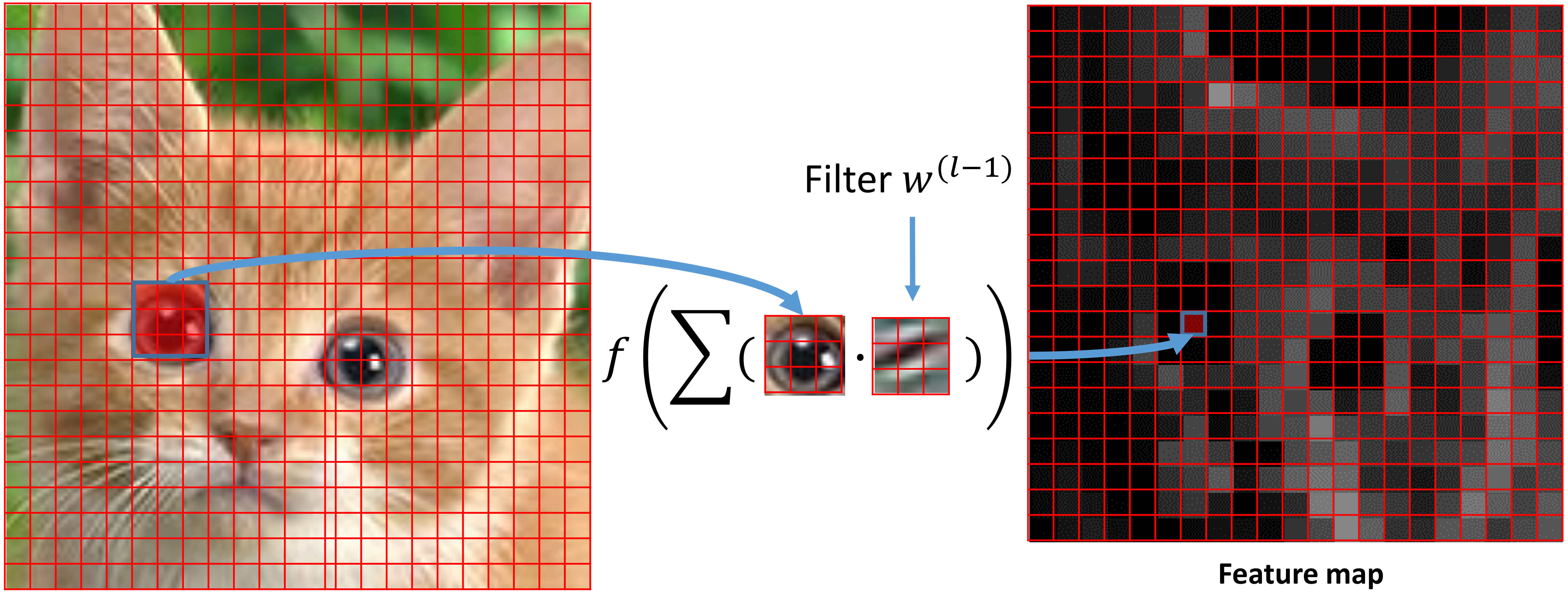
Input layer

Feature map

Convolutional layer



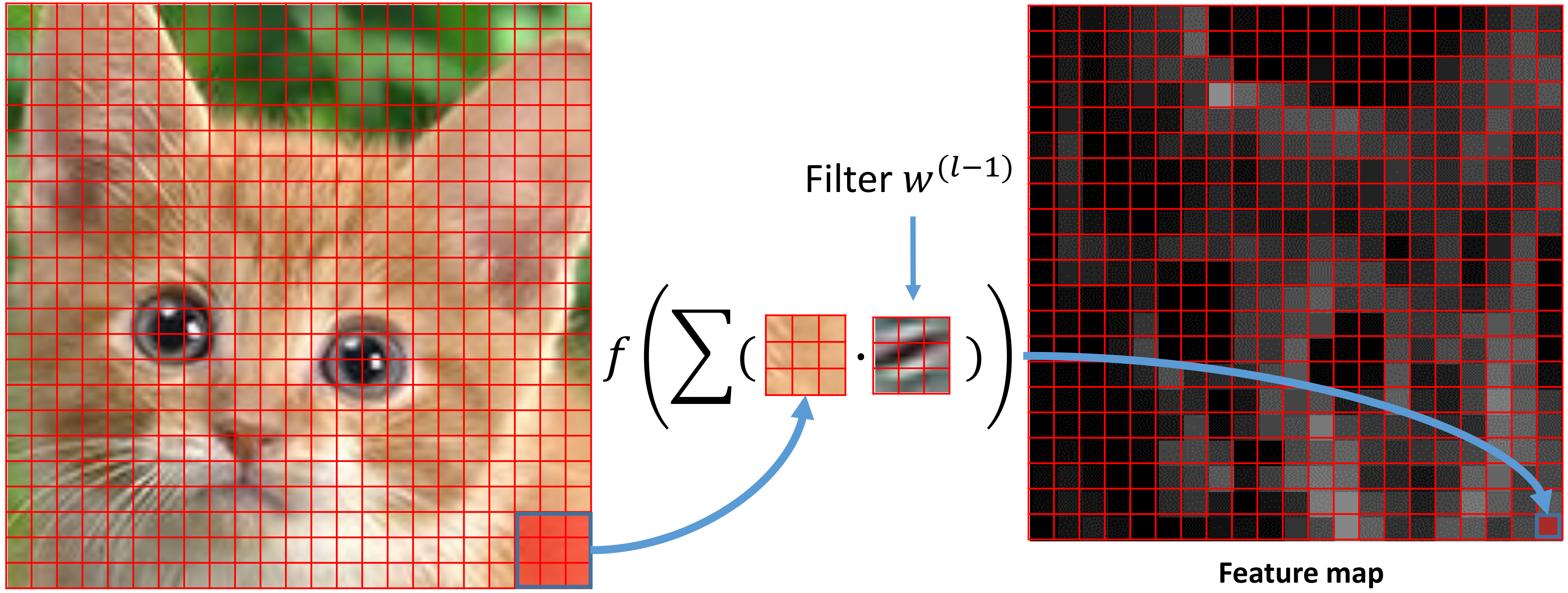
Convolutional layer



Input layer

Feature map

Convolutional layer



Input layer

Feature map

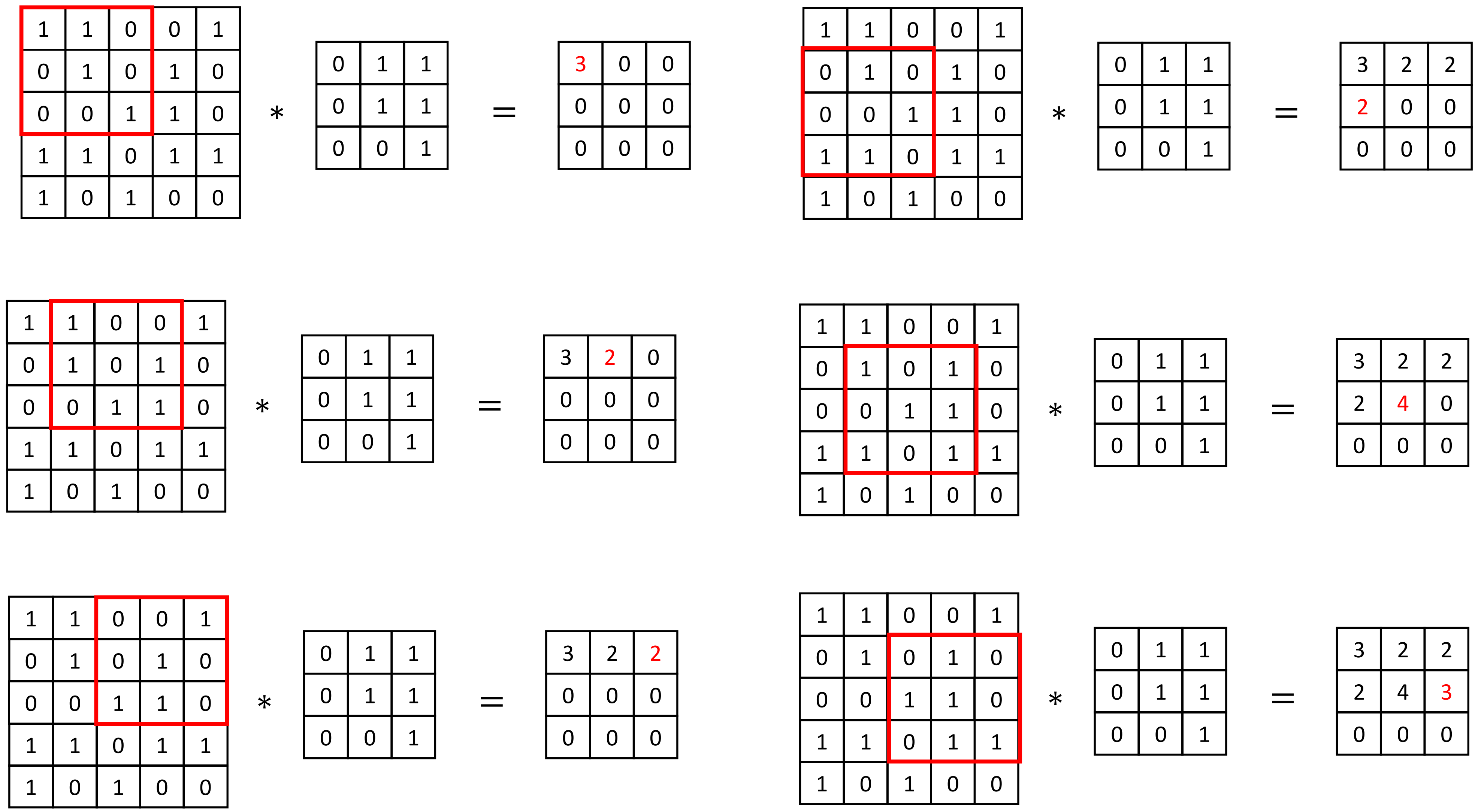
Mathematic operations

- Consider a 2 dimensional input feature map, $x^{(l-1)}$ with size = (d, d) = (5,5)
- Parameters
  - $w^{(l-1)}$filter 's size = (N, N) = (3,3)
  - Stride, s = 1

    **Stride is the number of pixels shifts over the input matrix.*
  - Padding = no
- $x^{(l)}$ feature map's output size $= \frac{d-N}{s} + 1 = \frac{5-3}{1} + 1 = 3$

$$\underbrace{x^{(l-1)} = \begin{array}{|c|c|c|c|c|}\hline 1 & 1 & 0 & 0 & 1 \\ \hline 0 & 1 & 0 & 1 & 0 \\ \hline 0 & 0 & 1 & 1 & 0 \\ \hline 1 & 1 & 0 & 1 & 1 \\ \hline 1 & 0 & 1 & 0 & 0 \\ \hline \end{array}}_{5\text{x}5} * \underbrace{w = \begin{array}{|c|c|c|}\hline 0 & 1 & 1 \\ \hline 0 & 1 & 1 \\ \hline 0 & 0 & 1 \\ \hline \end{array}}_{3\text{x}3} = \underbrace{x^{(l)} = \begin{array}{|c|c|c|}\hline 3 & 2 & 2 \\ \hline 2 & 4 & 3 \\ \hline 3 & 3 & 3 \\ \hline \end{array}}_{3\text{x}3}$$

Mathematic operations



$$\begin{bmatrix}1&1&0&0&1\\0&1&0&1&0\\0&0&1&1&0\\1&1&0&1&1\\1&0&1&0&0\end{bmatrix} * \begin{bmatrix}0&1&1\\0&1&1\\0&0&1\end{bmatrix} = \begin{bmatrix}3&0&0\\0&0&0\\0&0&0\end{bmatrix}$$

$$\begin{bmatrix}1&1&0&0&1\\0&1&0&1&0\\0&0&1&1&0\\1&1&0&1&1\\1&0&1&0&0\end{bmatrix} * \begin{bmatrix}0&1&1\\0&1&1\\0&0&1\end{bmatrix} = \begin{bmatrix}3&2&0\\0&0&0\\0&0&0\end{bmatrix}$$

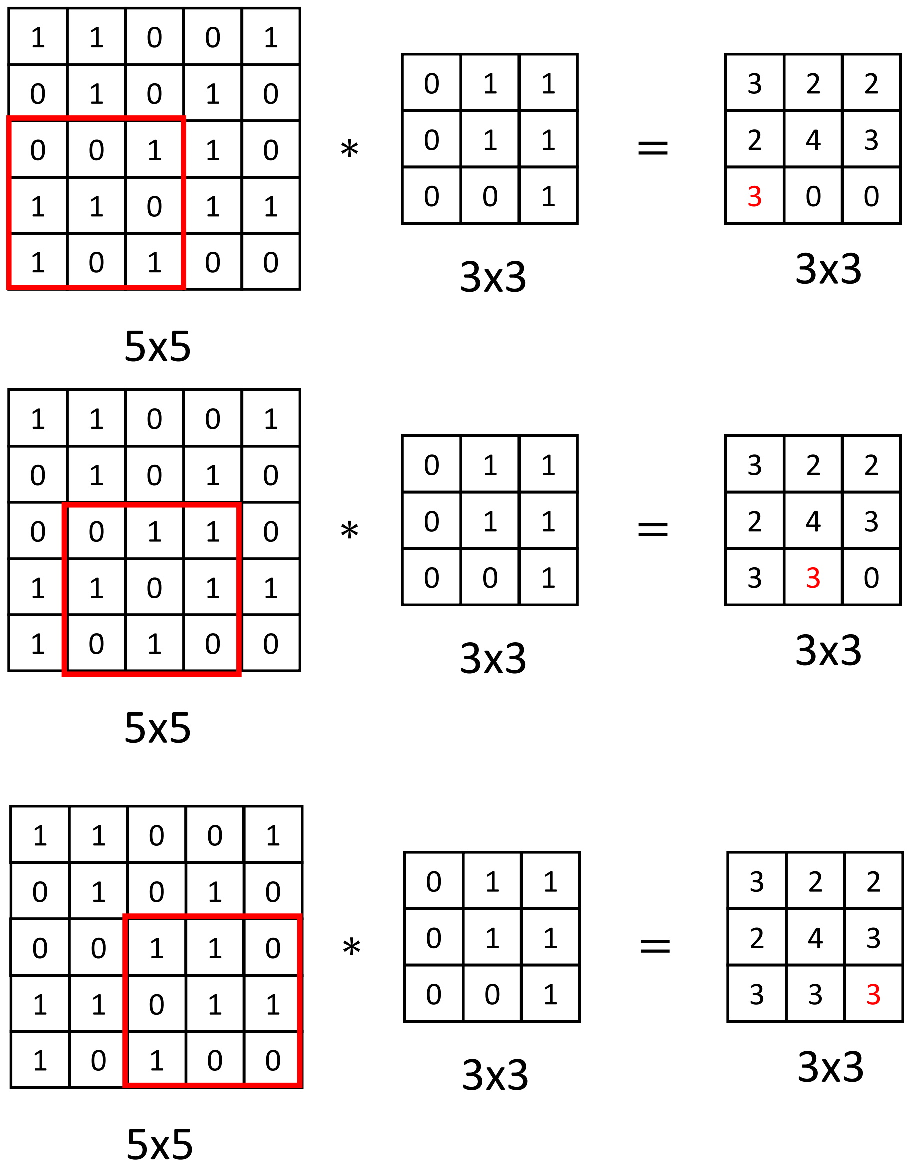
$$\begin{bmatrix}1&1&0&0&1\\0&1&0&1&0\\0&0&1&1&0\\1&1&0&1&1\\1&0&1&0&0\end{bmatrix} * \begin{bmatrix}0&1&1\\0&1&1\\0&0&1\end{bmatrix} = \begin{bmatrix}3&2&2\\0&0&0\\0&0&0\end{bmatrix}$$

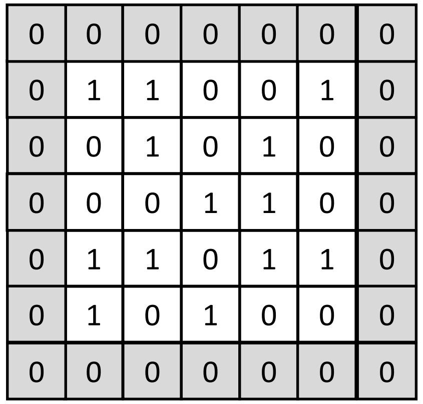
$$\begin{bmatrix}1&1&0&0&1\\0&1&0&1&0\\0&0&1&1&0\\1&1&0&1&1\\1&0&1&0&0\end{bmatrix} * \begin{bmatrix}0&1&1\\0&1&1\\0&0&1\end{bmatrix} = \begin{bmatrix}3&2&2\\2&0&0\\0&0&0\end{bmatrix}$$

$$\begin{bmatrix}1&1&0&0&1\\0&1&0&1&0\\0&0&1&1&0\\1&1&0&1&1\\1&0&1&0&0\end{bmatrix} * \begin{bmatrix}0&1&1\\0&1&1\\0&0&1\end{bmatrix} = \begin{bmatrix}3&2&2\\2&4&0\\0&0&0\end{bmatrix}$$

$$\begin{bmatrix}1&1&0&0&1\\0&1&0&1&0\\0&0&1&1&0\\1&1&0&1&1\\1&0&1&0&0\end{bmatrix} * \begin{bmatrix}0&1&1\\0&1&1\\0&0&1\end{bmatrix} = \begin{bmatrix}3&2&2\\2&4&3\\0&0&0\end{bmatrix}$$

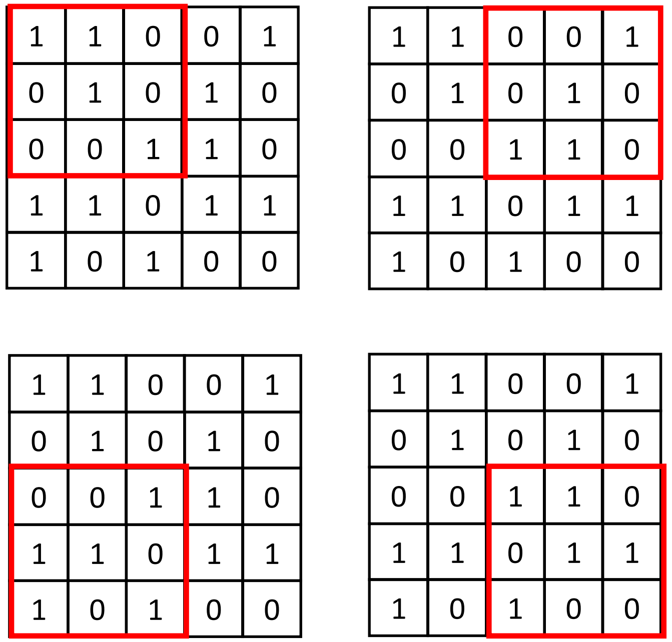
Mathematic operations


$$\begin{bmatrix}1&1&0&0&1\\0&1&0&1&0\\0&0&1&1&0\\1&1&0&1&1\\1&0&1&0&0\end{bmatrix} * \begin{bmatrix}0&1&1\\0&1&1\\0&0&1\end{bmatrix} = \begin{bmatrix}3&2&2\\2&4&3\\3&3&3\end{bmatrix}$$


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

If padding = (1,1),

- Input image’s size: (7,7)
- Feature map’s output size $= \frac{7-3}{1} + 1 = 5$


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

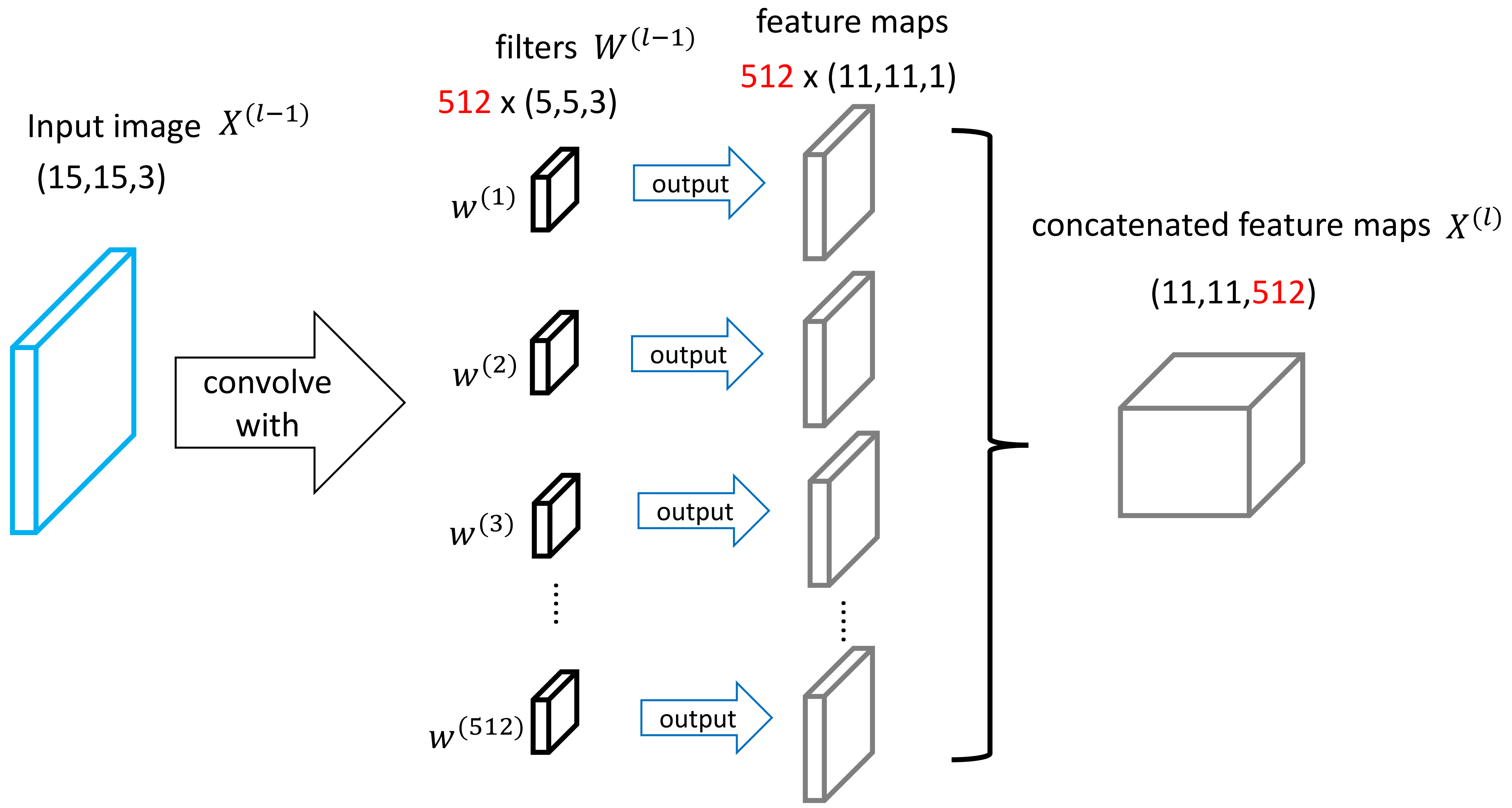
If stride = 2,

- Input image’s size: (5,5)
- Feature map’s output size $= \frac{5-3}{2} + 1 = 2$

3	2
3	3

Mathematic operations

- *N* number of filters produces *N* number of feature maps

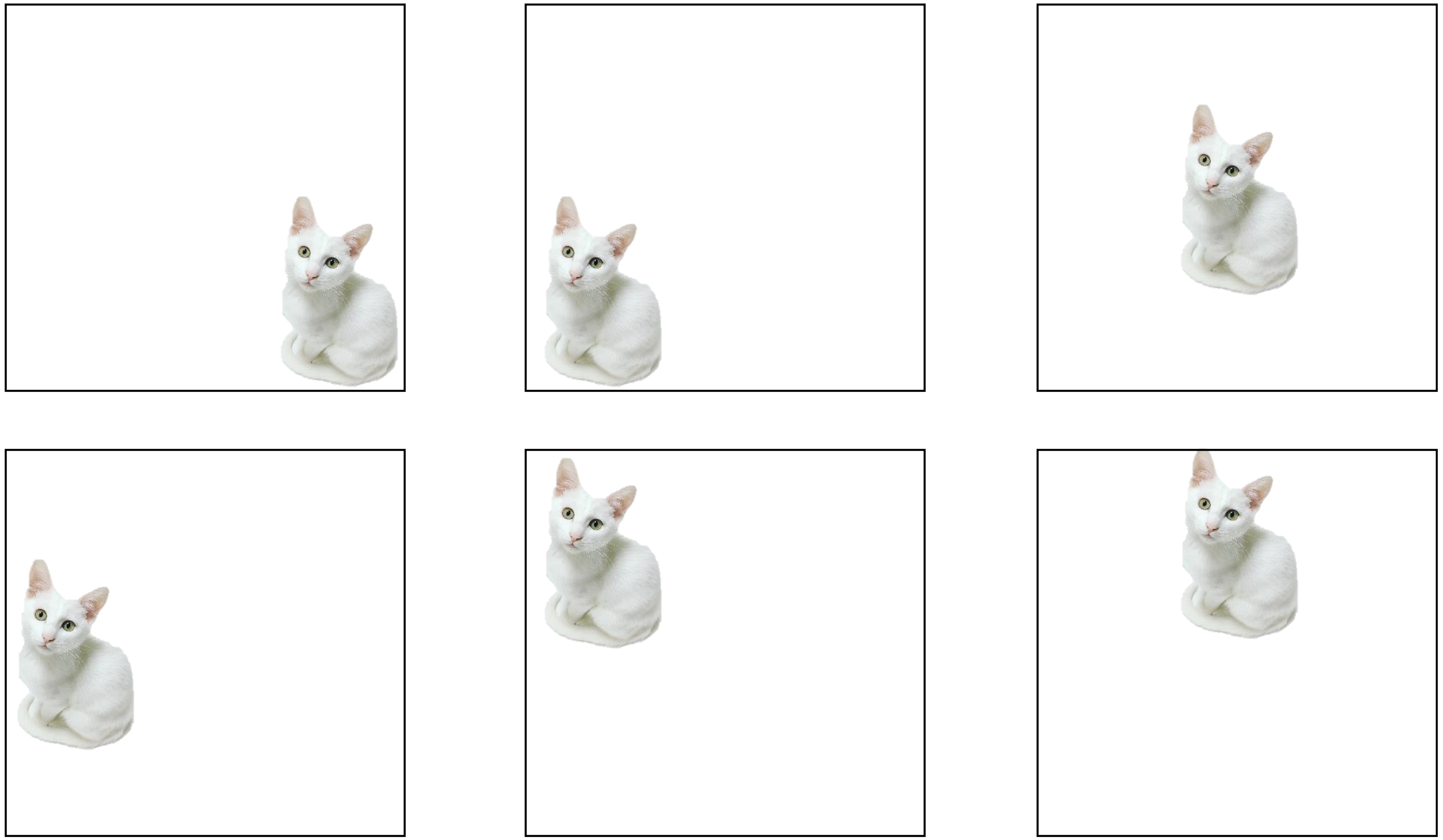


Weights sharing

- Weight sharing happens across the **receptive field** of the neurons(filters) in a particular layer. Weights are the numbers within each filter.
- So essentially we are trying **to learn a filter**. These filters act on a certain receptive field/ small section of the image.
- When the filter moves through the image, the filter does not change. The idea being, if an edge is important to learn in a particular part of an image, it is important in other parts of the image too.
- Weight sharing is to provide **translation invariance** (allows the network to recognize the same object in an image no matter where it appears) , which is fundamental to image-based recognition.

Translation invariance

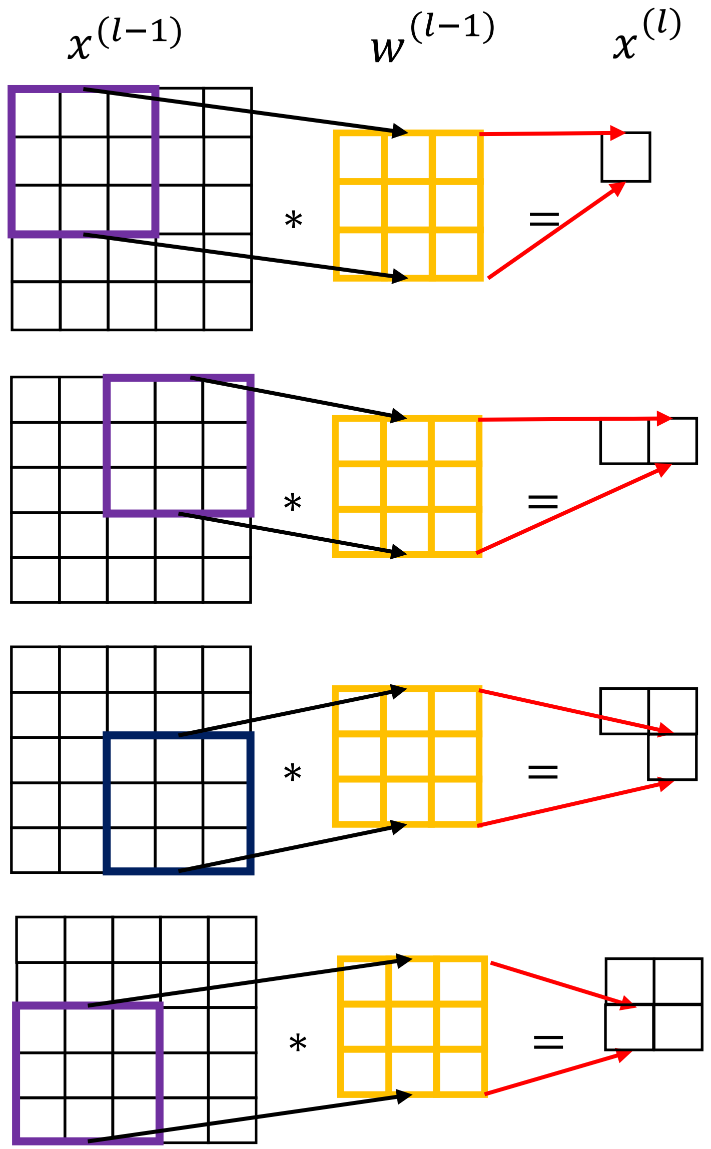
image size = 50 x 50 pixels



Sparsity of connection

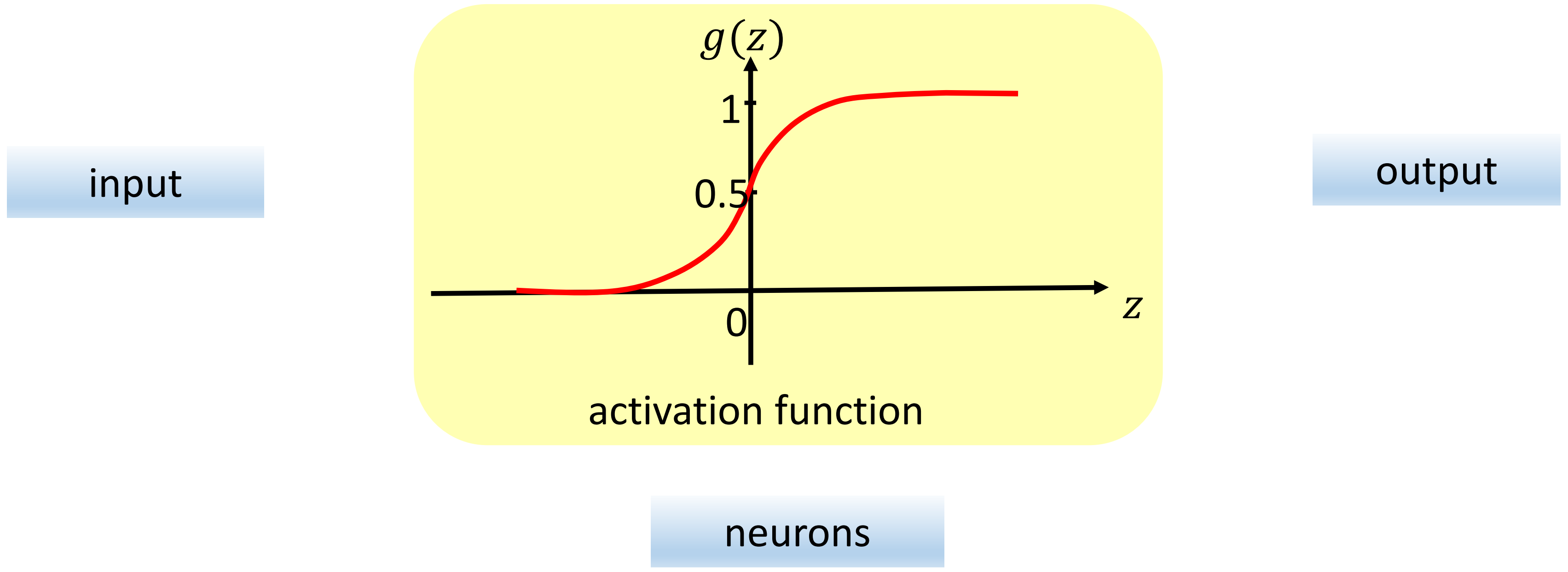
- The sparsity of connection means that each element of the output depends only on the small section of the input.
- For example, an element in the output of convolutional layer with 3 x 3 filter will depend only on 9 numbers from the input.
- As we get deeper into the network, the outputs will depend on less and less numbers.
- This allows us to train with less samples and prevent overfitting.

Convolutional layer



Activation function

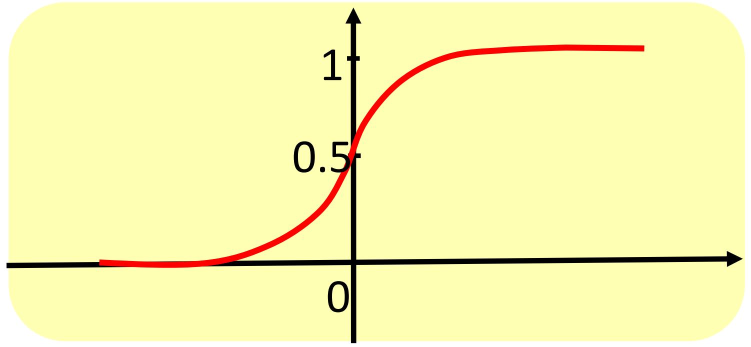
- The $g(\cdot)$ is an activation function that maps the input signals into output signals that are needed for the neural network to function.
- Increasingly, neural networks use **non-linear activation functions**, which can help the network learn complex data, calculate and learn almost any function representing a question, and provide accurate predictions.



Sigmoid and Tanh activation functions

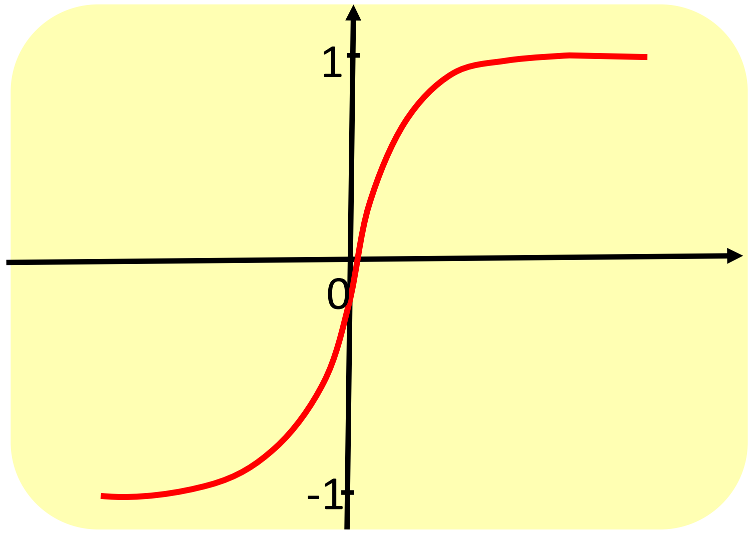
Sigmoid:

$$g(z) = \frac{1}{1 + e^{-z}}$$



Tanh:

$$g(z) = \tanh(z)$$

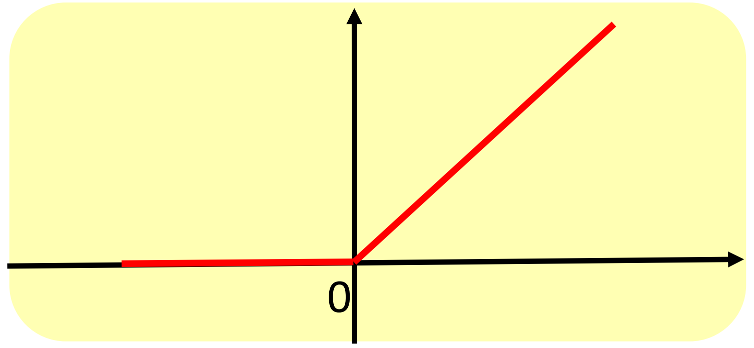


- Traditionally, two widely used nonlinear activation functions are the **sigmoid** and **hyperbolic tangent** activation functions.

Limitations of Sigmoid and Tanh

- **Limited sensitivity and saturation of the function**
  - Large values snap to 1.0 and small values snap to -1 or 0 for tanh and sigmoid respectively. The functions are only really sensitive to changes around their mid-point of their input, such as 0.5 for sigmoid and 0.0 for tanh.
- **Vanishing gradient (details will be explained later)**
  - In a feedforward network, the back propagated error typically decreases exponentially as a function of the distance from the final layer, resulting in the model impossible learn.
- **Computationally expensive**

ReLU

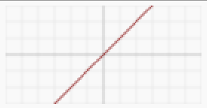
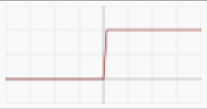
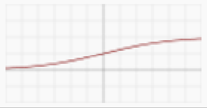
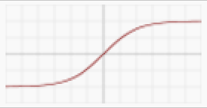
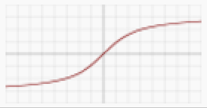
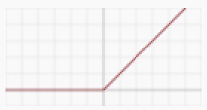
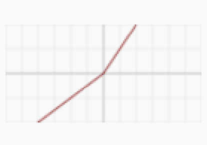
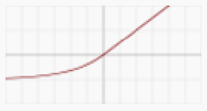
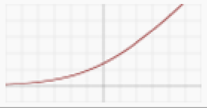


- ReLU stands for Rectified Linear Unit for a non-linear operation. The output is $f(x) = \max(x)$.
- Why ReLU is important :

'Because rectified linear units are nearly linear, they preserve many of the properties that make linear models easy to optimize with gradient-based methods. They also preserve many of the properties that make linear models generalize well.'

– Page 175, Deep Learning 2016

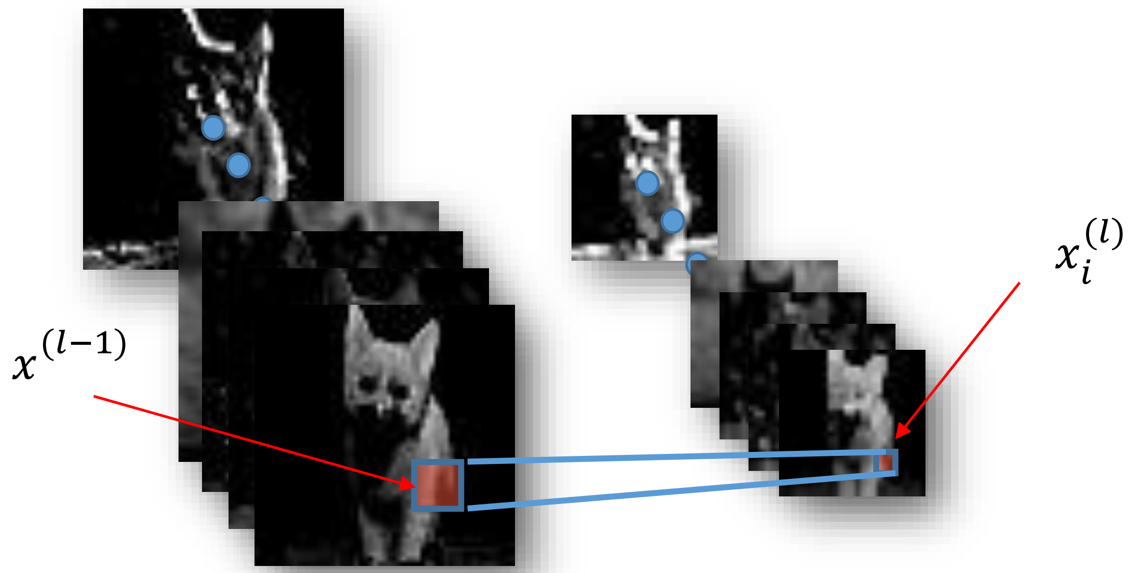
Other activation functions

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for} & x < 0 \\ 1 & \text{for} & x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} & x \neq 0 \\ ? & \text{for} & x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for} & x < 0 \\ x & \text{for} & x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} & x < 0 \\ 1 & \text{for} & x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for} & x < 0 \\ x & \text{for} & x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for} & x < 0 \\ 1 & \text{for} & x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for} & x < 0 \\ x & \text{for} & x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for} & x < 0 \\ 1 & \text{for} & x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

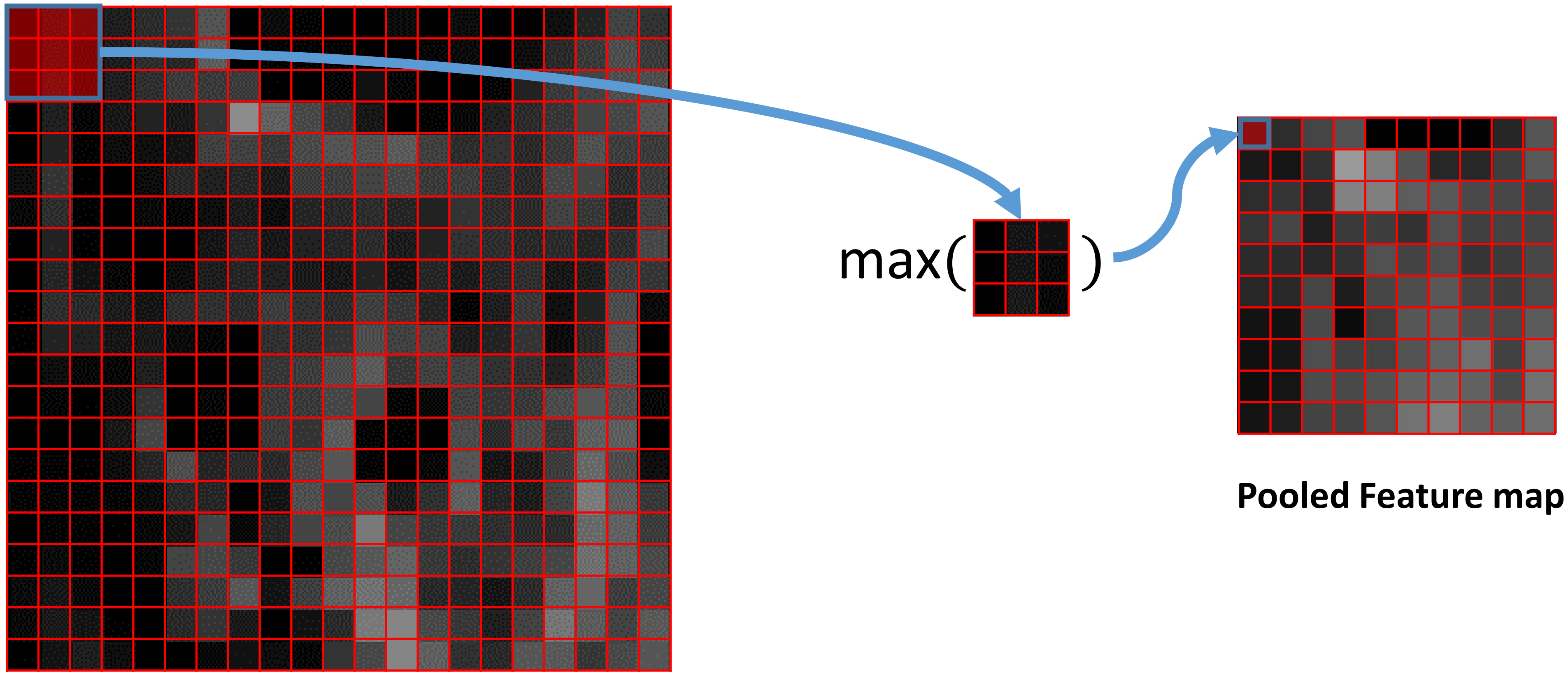
Pooling layer

- Feature selection
- Improve generalization
- Pooling method
  - Max pooling
  - Min pooling
  - Average pooling
  - and the others…
- Similarly,
  - $x_i^{(l)} = \max\left(x^{(l-1)}\right)$

  lets $x_i^{(l)}$ is the output pixel and $x^{(l-1)}$ is one of the input window region of the *r*-th feature map.

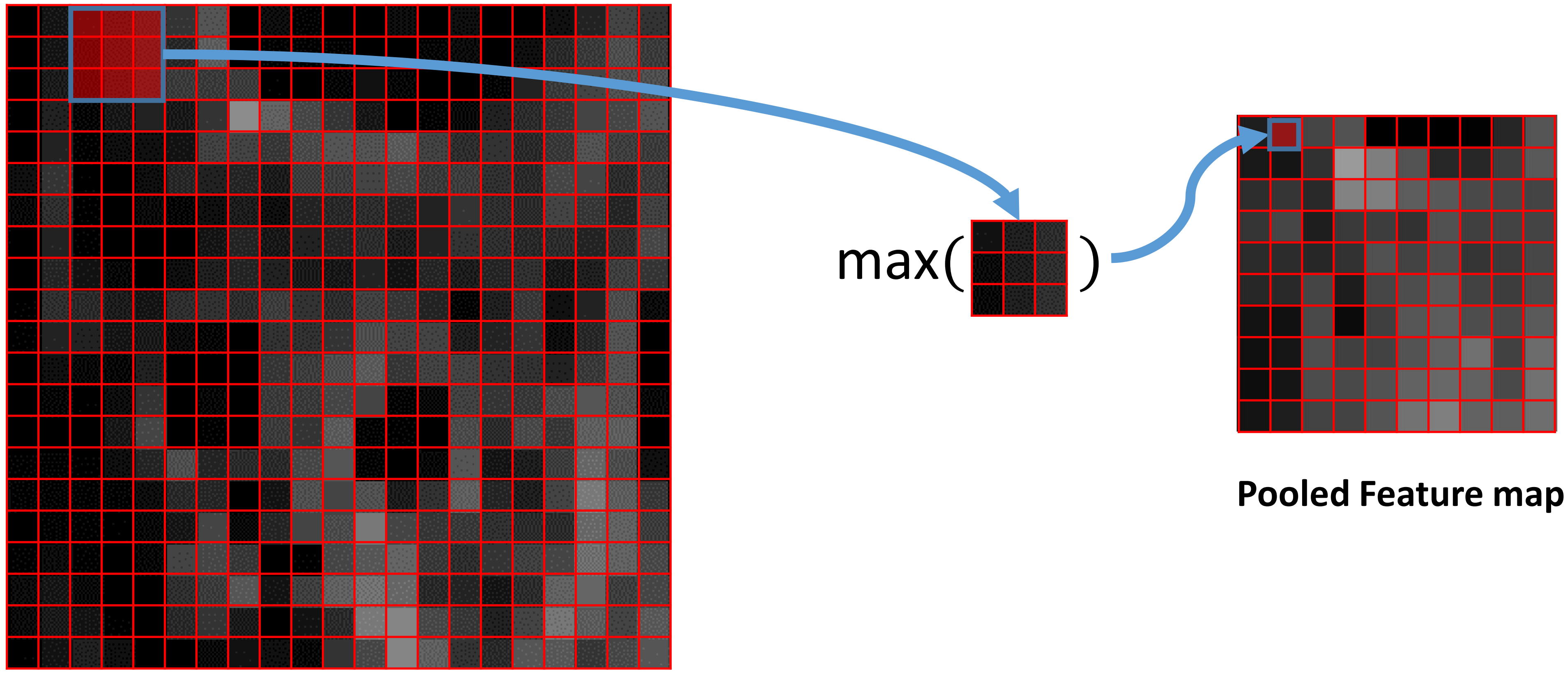


Pooling layer



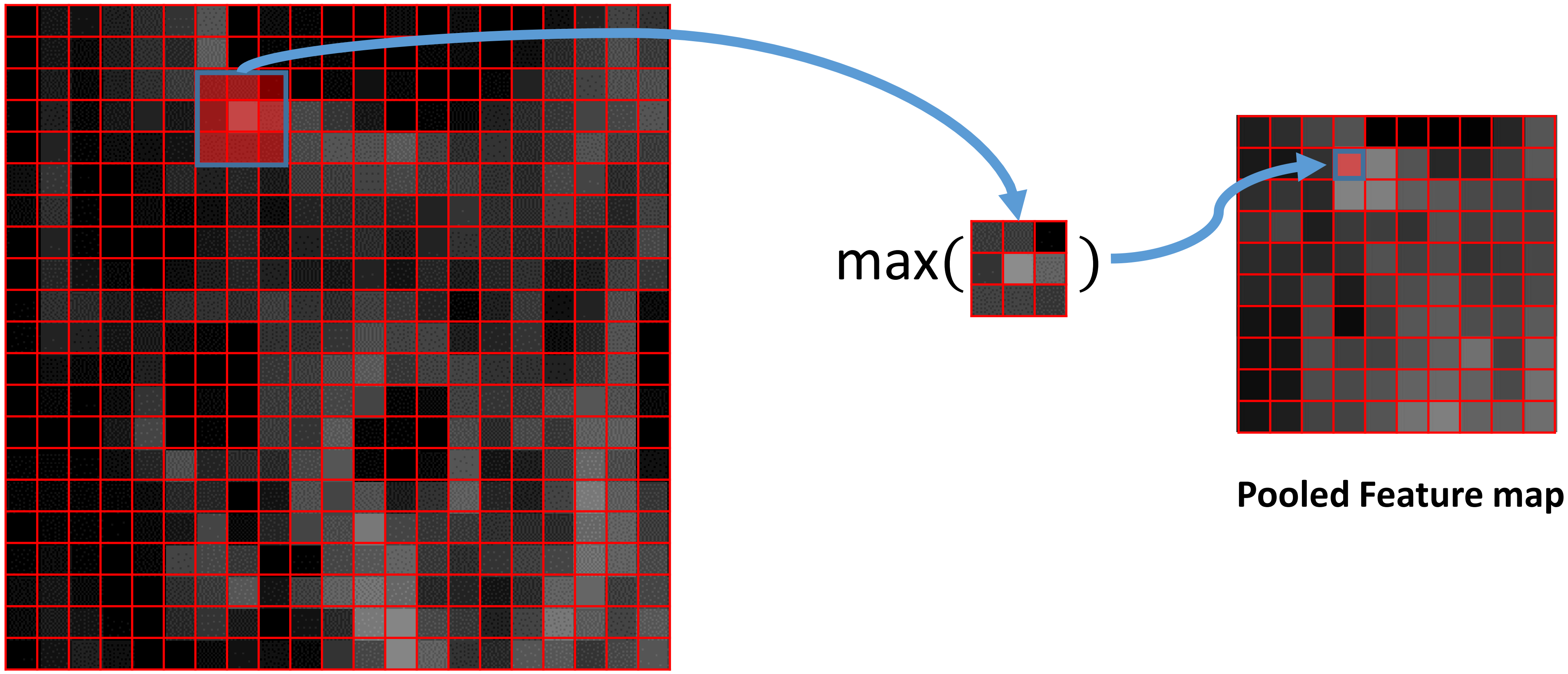
Pooled Feature map

Pooling layer



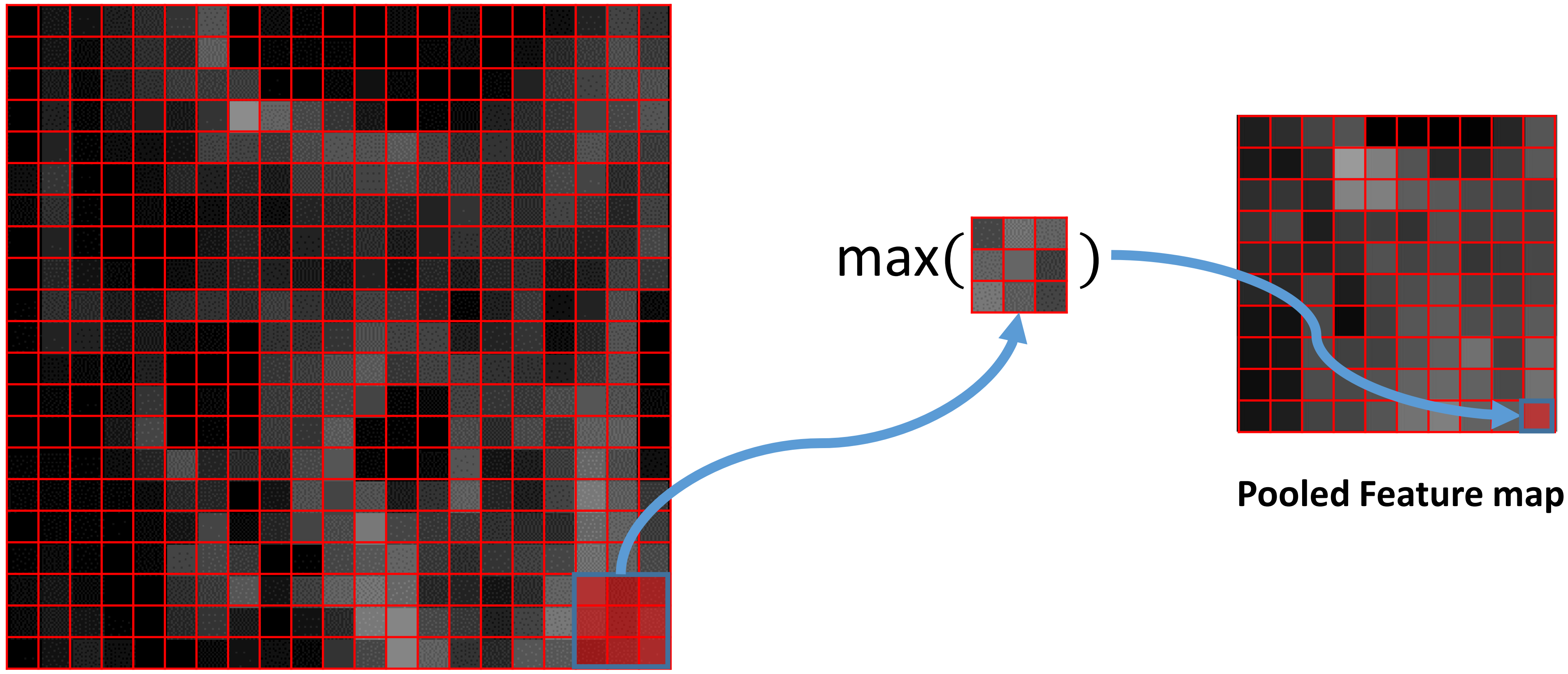
Pooled Feature map

Pooling layer



Pooled Feature map

Pooling layer



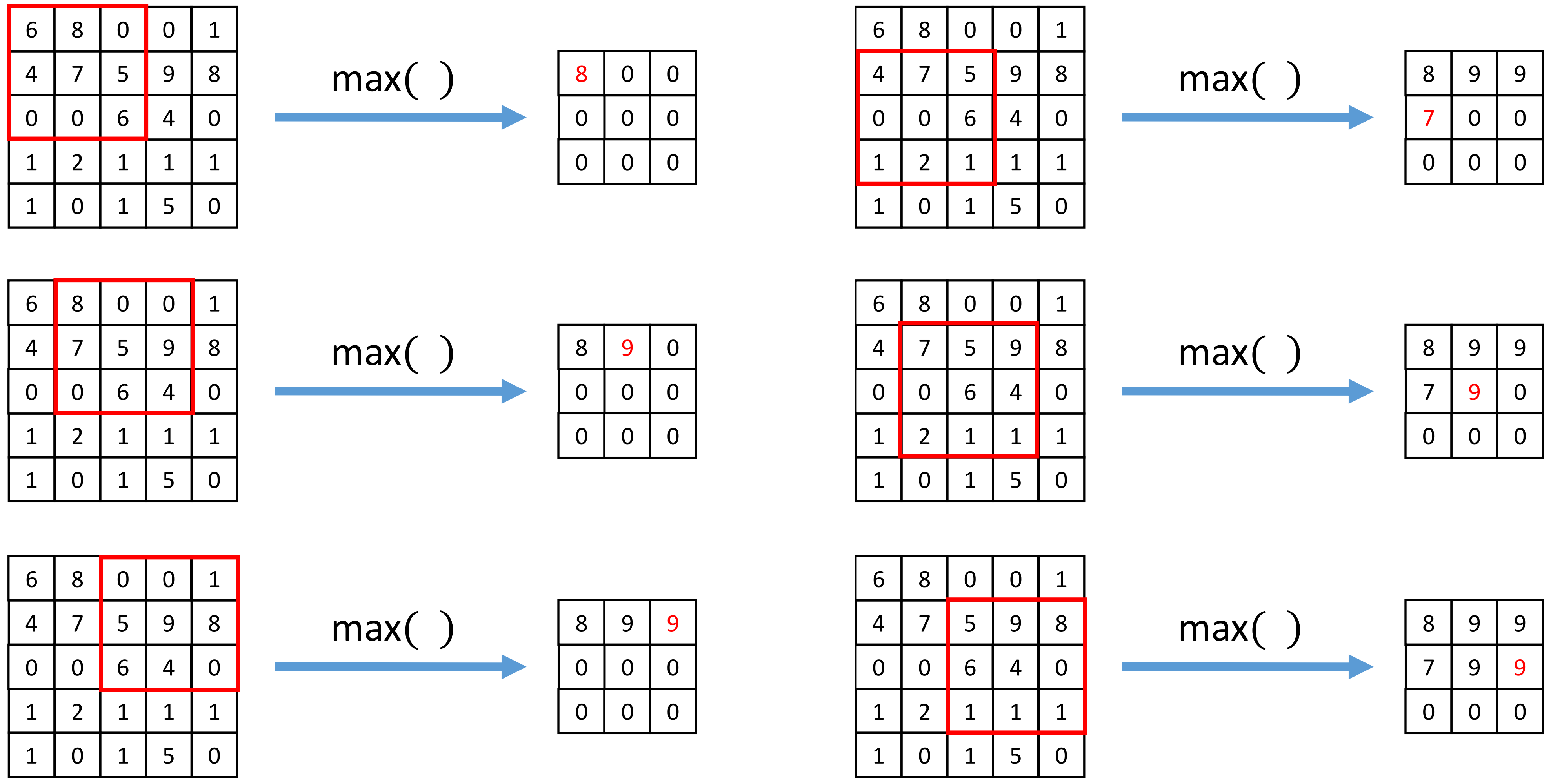
Pooled Feature map

Mathematic operations

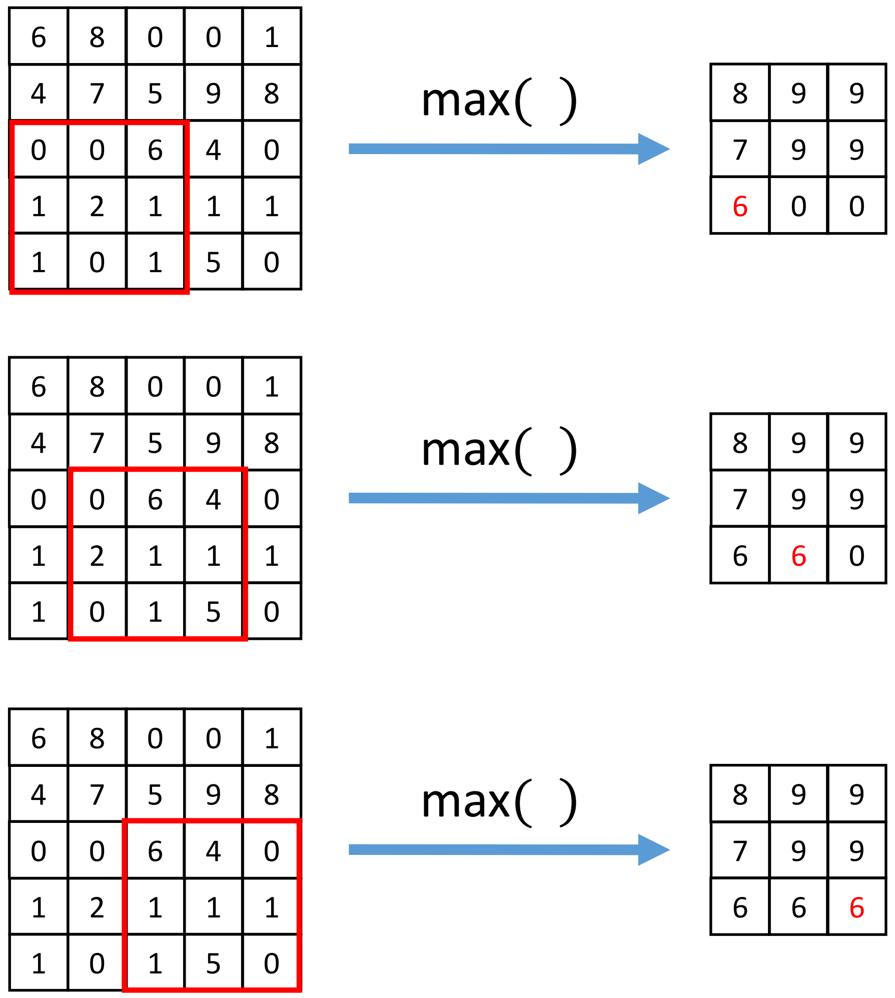
- Consider a 2 dimensional feature map, $x^{(l-1)}$ with size = (d, d) =(5,5)
- Parameters
  - Pool size = (N, N) = (3,3)
  - Stride, s = 1
  - Padding = no
- $x^{(l)}$ pooled map's output size $= \frac{d-N}{s} + 1 = \frac{5-3}{1} + 1 = 3$

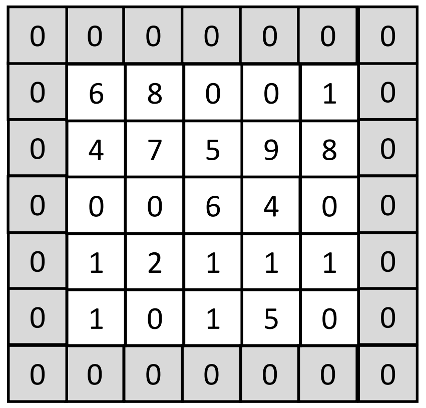
$$\underset{5 \times 5}{\overset{x^{(l-1)}}{\begin{array}{|c|c|c|c|c|}\hline 6 & 8 & 0 & 0 & 1 \\ \hline 4 & 7 & 5 & 9 & 8 \\ \hline 0 & 0 & 6 & 4 & 0 \\ \hline 1 & 2 & 1 & 1 & 1 \\ \hline 1 & 0 & 1 & 5 & 0 \\ \hline \end{array}}} \xrightarrow{\text{max( )}} \underset{3 \times 3}{\overset{x^{(l)}}{\begin{array}{|c|c|c|}\hline 8 & 9 & 9 \\ \hline 7 & 9 & 9 \\ \hline 6 & 6 & 6 \\ \hline \end{array}}}$$

Mathematic operations



Mathematic operations

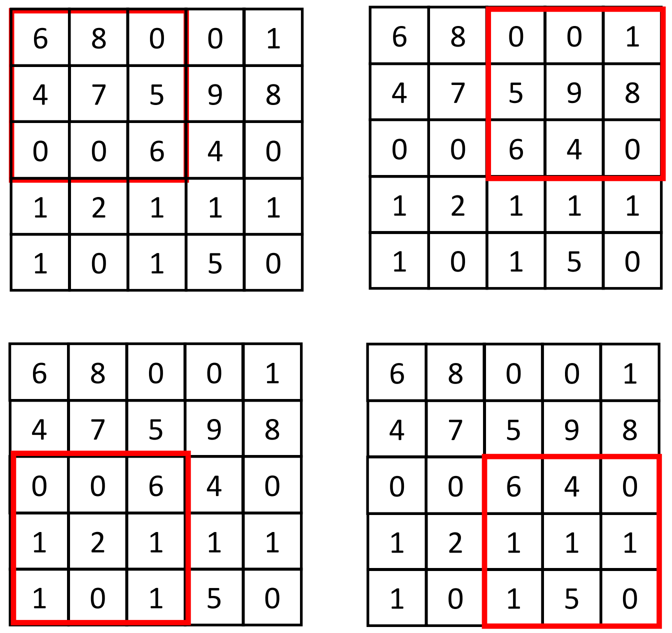




If padding = (1,1),

- Feature map's size: (7,7)
- Pooled map's output size

$$= \frac{7-3}{1} + 1 = 5$$



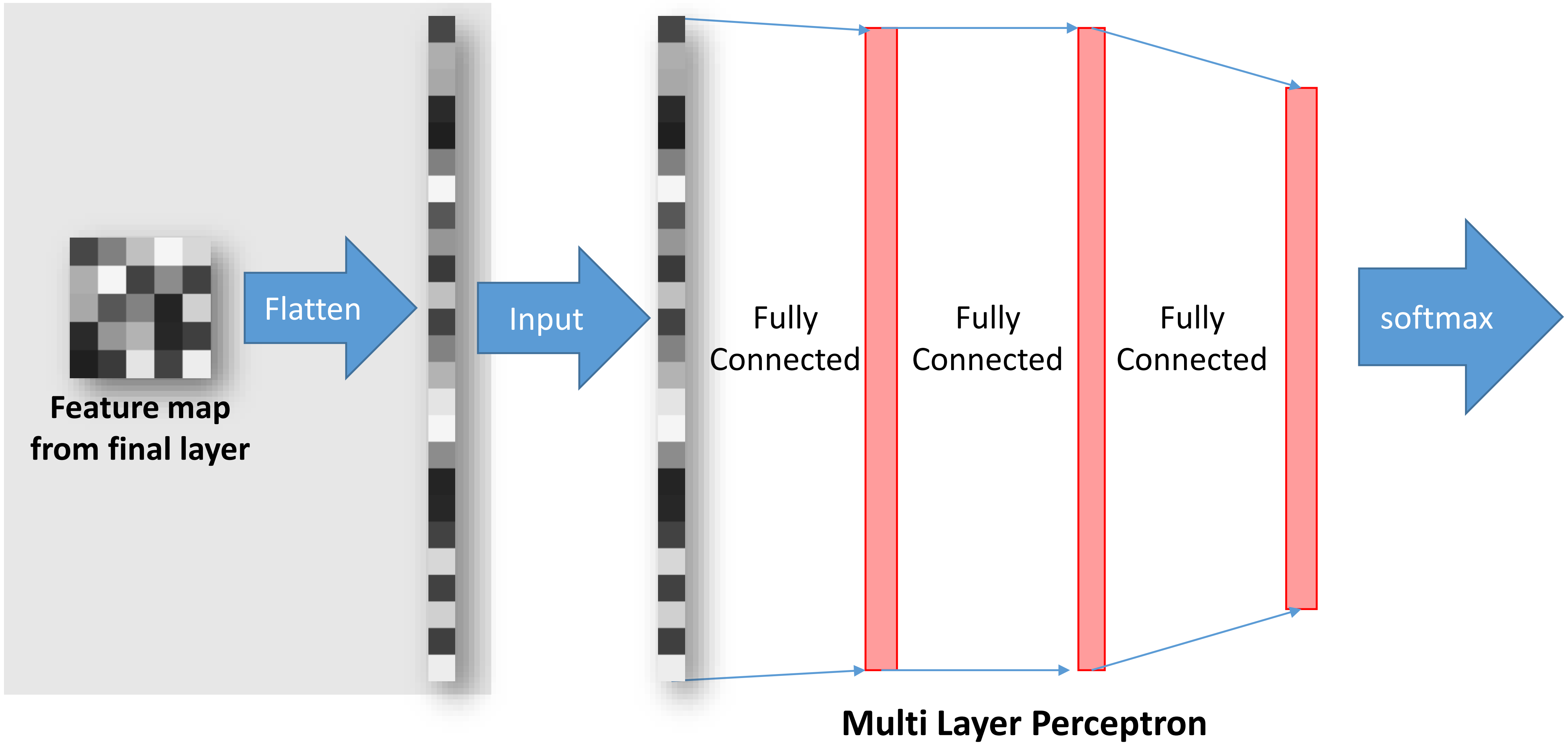
If stride = 2,

- Feature map's size : (5,5)
- Pooled map's output size

$$= \frac{5-3}{2} + 1 = 2$$

8	9
6	6

Fully connected layer

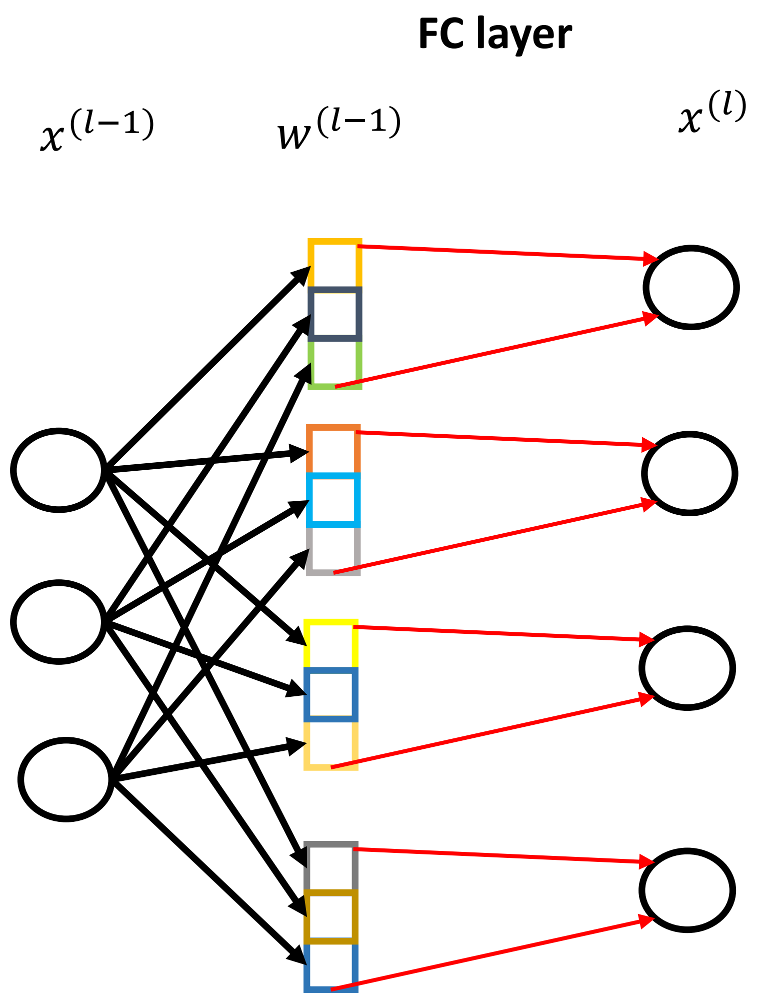


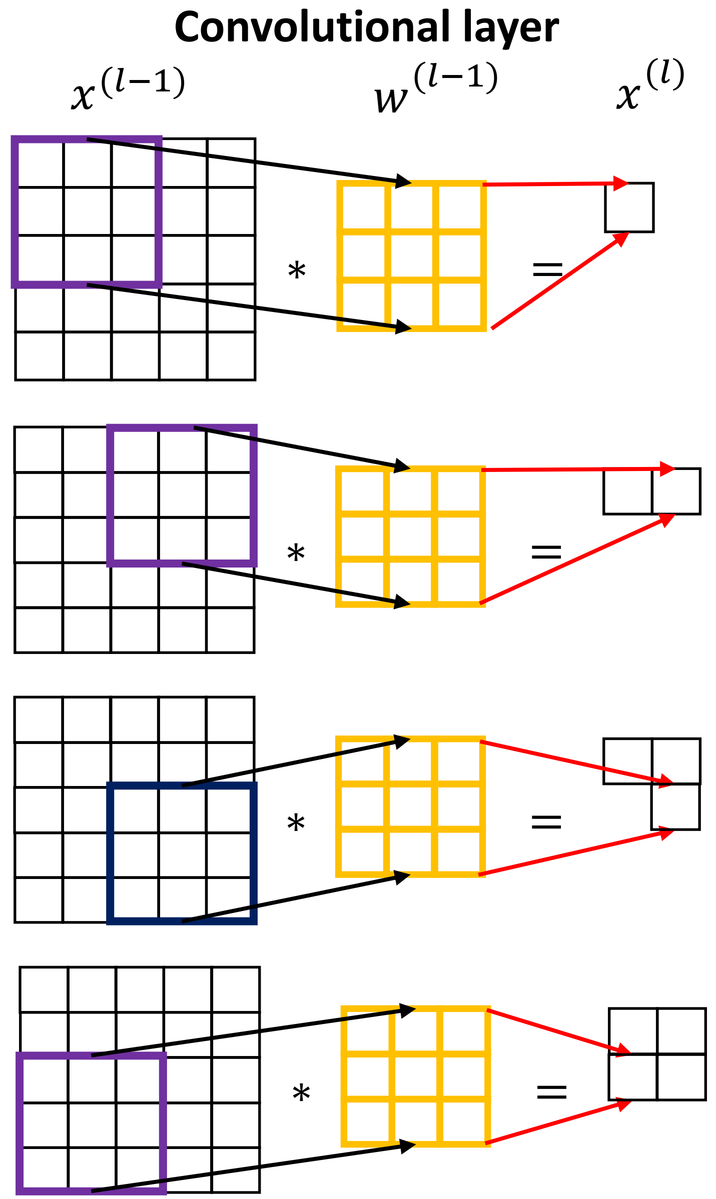
Fully connected layer

- In a fully connected (FC) layer, each neuron is connected to **every** neuron in the previous layer, and **each connection has it's own weight**, as seen in regular ANN.

- In contrast, in a convolutional layer, each neuron is only connected to **a few** nearby (local) neurons in the previous layer, and **the same set of weights** (and local connection layout) is used for every neuron.

Fully connected layer



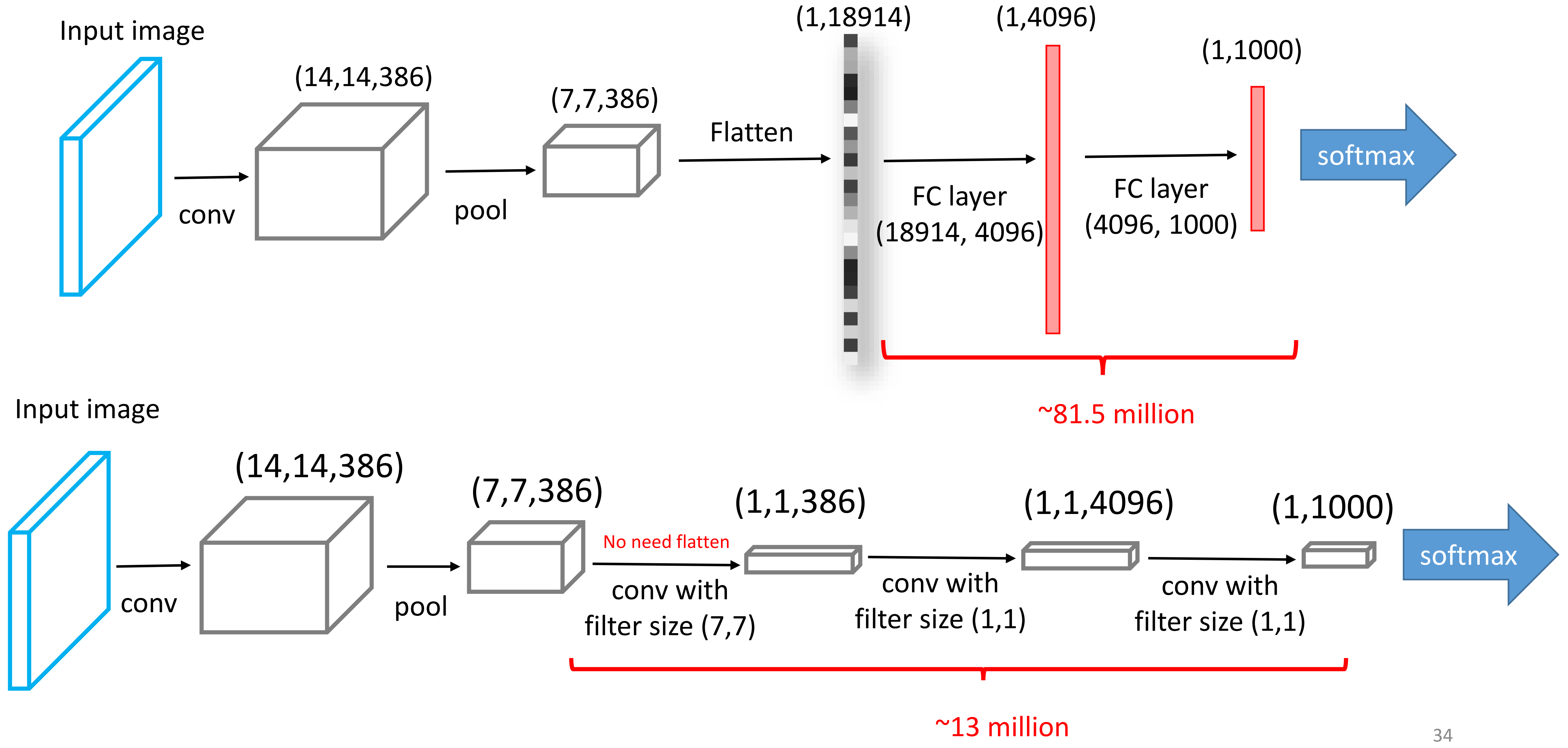


Replacing FC with convolutional layer

- The benefit of replacing a fully connected layer with a convolutional layer is that the number of parameters to adjust are reduced due to the fact that the **weights are shared** in a convolutional layer.

- It provides faster and more robust learning.

Converting FC to convolutional layer



Recap: Softmax layer

- Softmax layer is typically the final output layer in a neural network that performs multi-class classification (for example: object recognition).
- The **softmax** function takes a vector $z = [z_1 \,, z_2, .\,., z_k]$ of k output classes and maps them to a probability distribution, with each value in the **range** $(\mathbf{0}, \mathbf{1})$ and all the values **summing to 1**.

$$h_\theta^{(c)}(x) = \text{softmax}(z_c) = \frac{e^{z_c}}{\sum_{i=1}^{k} e^{z_i}} \quad \text{where } 1 \le i \le k$$

Objective of this lecture

SWINBURNE UNIVERSITY OF TECHNOLOGY SARAWAK CAMPUS

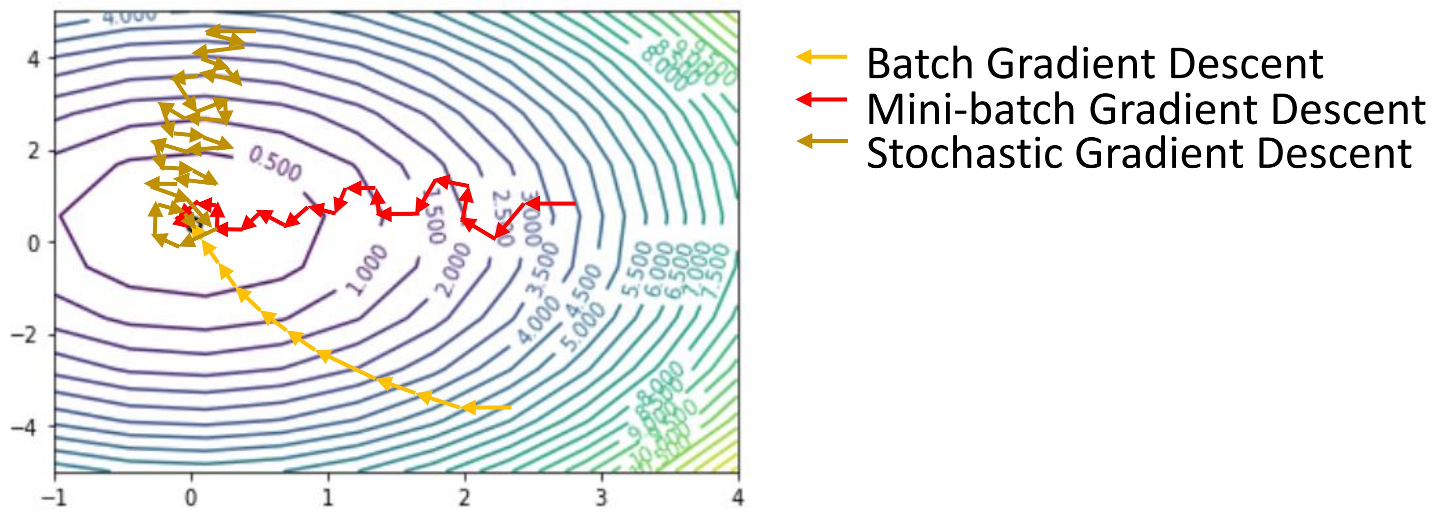
- To understand what is a convolutional neural network (CNN)
- To learn different optimization techniques
- To explore the state-of-the-art CNN architectures.
- To learn the deep learning design process.
- CNN with Keras

Recap: Optimization algorithms

- Recall the traditional optimization algorithms we have discussed previously: **batch**, **stochastic**, and **mini-batch** gradient descent (GD).
- The update equation for normal stochastic gradient descent:

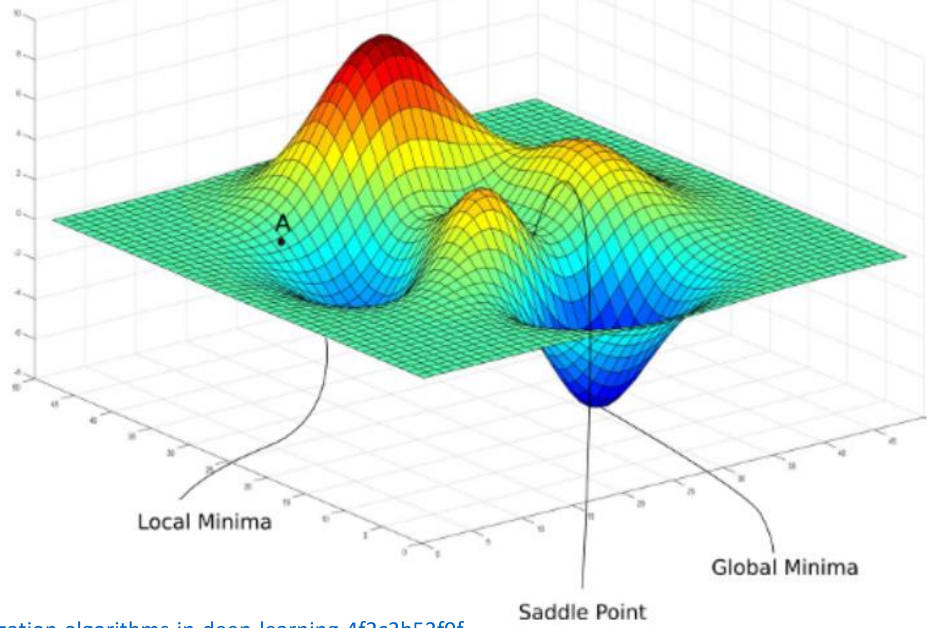
$$w_t \leftarrow w_{t-1} - \propto \left[\frac{\delta J}{\delta w}\right]_{w_{t-1}} \qquad \text{where } t = \text{new},\ t-1 = \text{old}$$

- Using the similar equation, w is often chosen to be updated based on mini-batch GD.



Recap: Optimization algorithms

- The loss function of logistic regression, linear regression and SVM only contain a minima (or a maxima) but no saddle points because the function is convex.
- Saddle points are present in the loss function of Neural Networks because the function is **complex** and **non-convex**.
- Using traditional optimization approach like SGD, if we are stuck with a local minimum or a saddle point, it will be very difficult to escape because the **gradient term becomes zero**.



https://medium.com/@ashwin8april/optimization-algorithms-in-deep-learning-4f2c3b53f9f

Optimization algorithms

- SGD with Momentum

  Momentum helps accelerate SGD in the relevant direction and dampens oscillations by adding a fraction of the update vector of the past step to the current update vector.

- Nesterov Accelerated Gradient (NAG)

  Similar to momentum, but the gradient is calculated at the position ahead in the direction of the momentum. This anticipatory update prevents overshooting and improves convergence.

- Adagrad

  This algorithm adapts the learning rate to the parameters, performing smaller updates for parameters associated with frequently occurring features, and larger updates for parameters associated with infrequent features. It's particularly useful for sparse data.

Optimization algorithms

- RMSprop

  RMSprop adjusts the learning rate by dividing by an exponentially decaying average of squared gradients. This allows it to adapt the learning rate for each of the parameters.

- Adam (Adaptive Moment Estimation):

  Combines elements of momentum and RMSprop, maintaining a moving average of both the gradients and their squared values to adjust the learning rate for each parameter. Adam is widely used due to its high performance in a variety of problems.

- Adadelta:

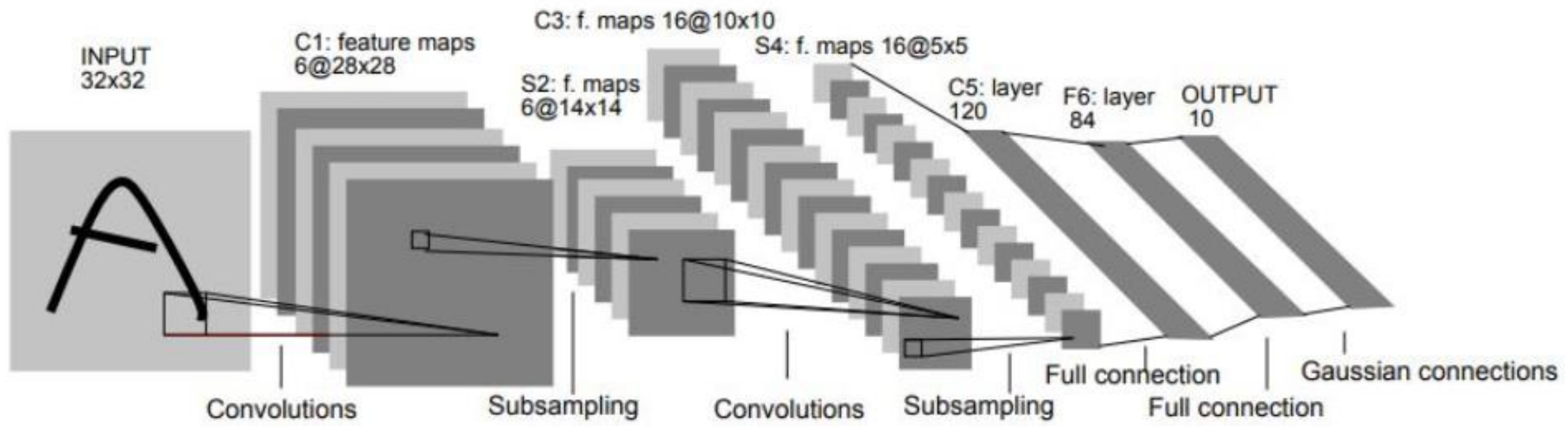
  An extension of Adagrad that seeks to reduce its aggressive, monotonically decreasing learning rate by calculating a window of accumulated past gradients.

Objective of this lecture

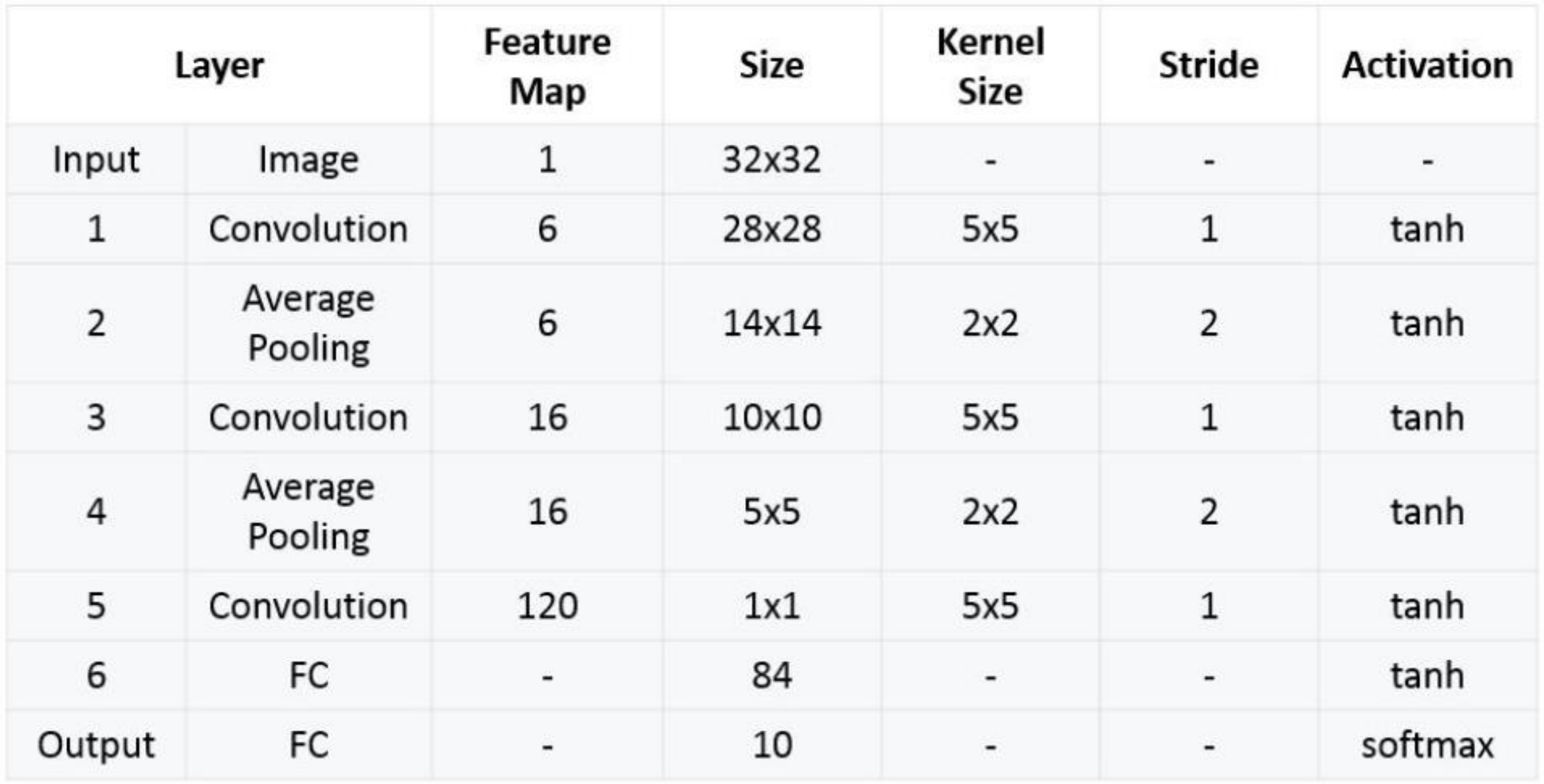
- To understand what is a convolutional neural network (CNN)
- To learn different optimization techniques
- ⭐ To explore the state-of-the-art CNN architectures.
- To learn the deep learning design process.
- CNN with Keras

CNN architectures – LeNet-5

- Proposed by Yann LeCun, Leon Bottou, Yosuha Bengio and Patrick Haffner in 1990’s.
- Designed for handwritten and machine-printed character recognition



LeNet-5 configuration

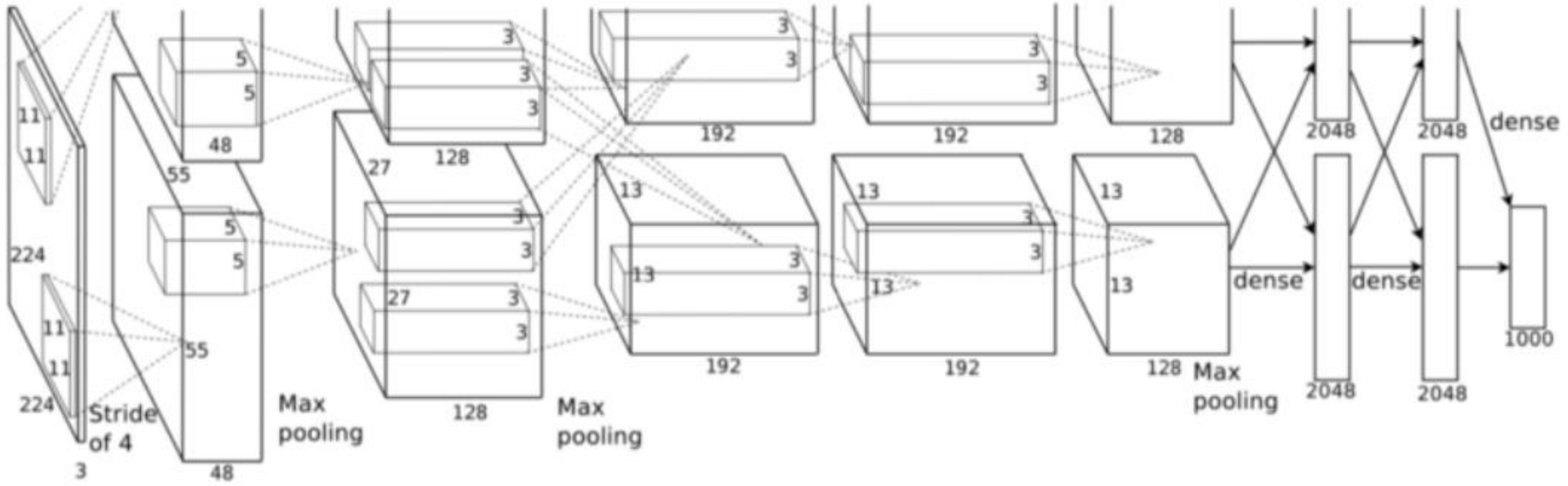


Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	FC	-	84	-	-	tanh
Output	FC	-	10	-	-	softmax

https://engmrk.com/lenet-5-a-classic-cnn-architecture/

CNN architectures – AlexNet

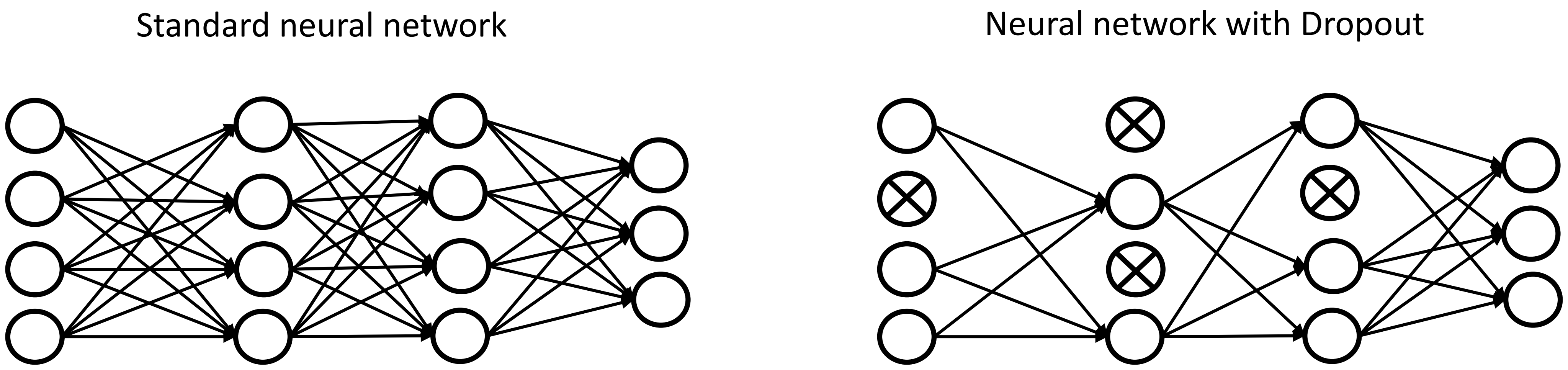
- The Network had a very similar architecture to LeNet, but was deeper, bigger, and featured Convolutional Layers stacked on top of each other (previously it was common to only have a single CONV layer always immediately followed by a POOL layer).



AlexNet’s special features and practices

- ReLU nonlinearity instead of tanh and sigmoid – improve training time
- Multiple GPU – allow bigger model, improve training time
- Overlapping pooling – improve accuracy
- Data augmentation – reduce overfitting
- **Dropout** – a regularization approach to reduce overfitting

Dropout

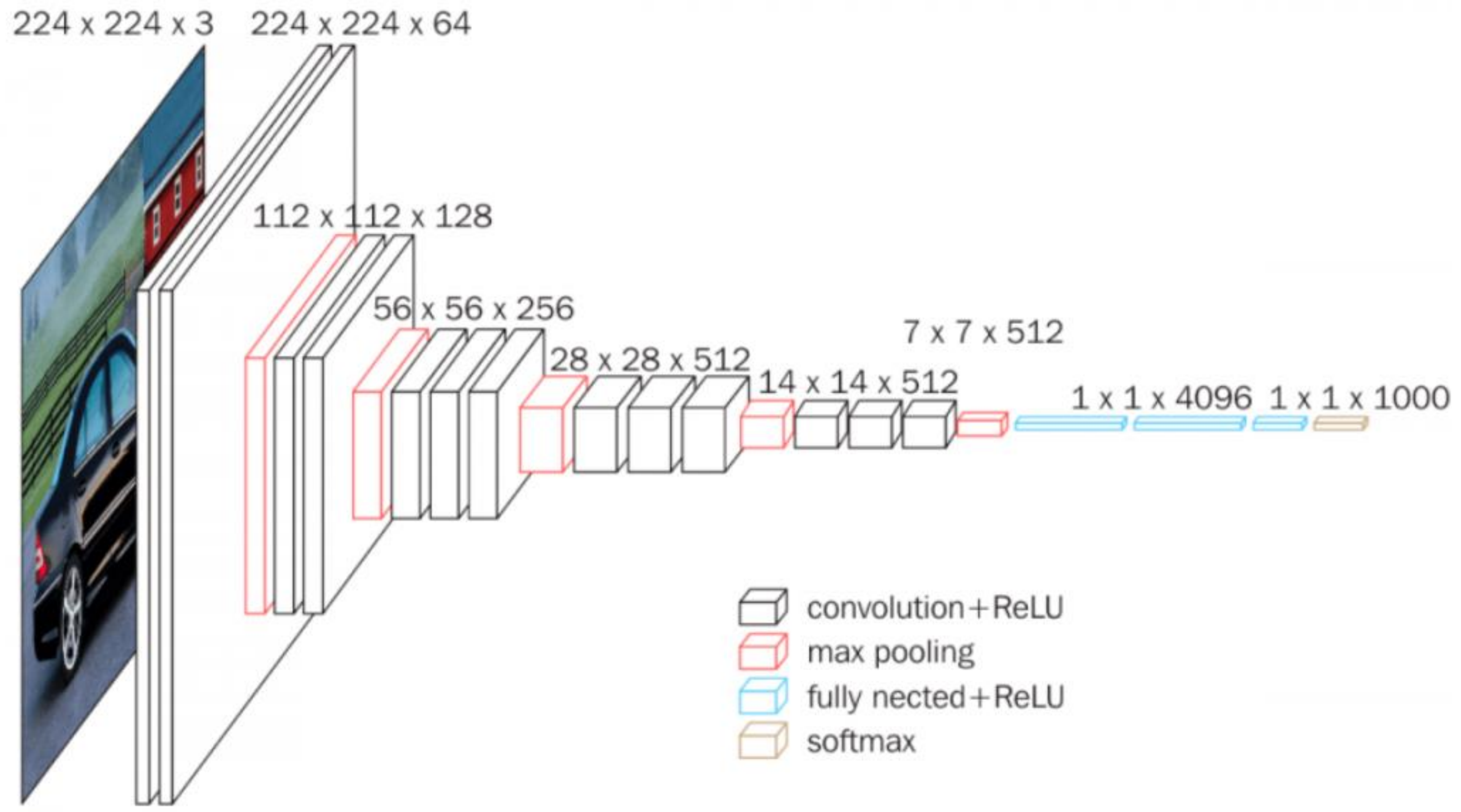


- Individual nodes are either dropped out of the net with a *1-p* probability or retained with a *p* probability.
- The inbound and outbound edges to a dropped-out node are also removed.
- For example, **Dropout(0.5),** a new set of nodes is randomly chosen to be dropped based on the specified dropout rate.

Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." *The journal of machine learning research* 15.1 (2014): 1929-1958.

CNN architectures – VGG

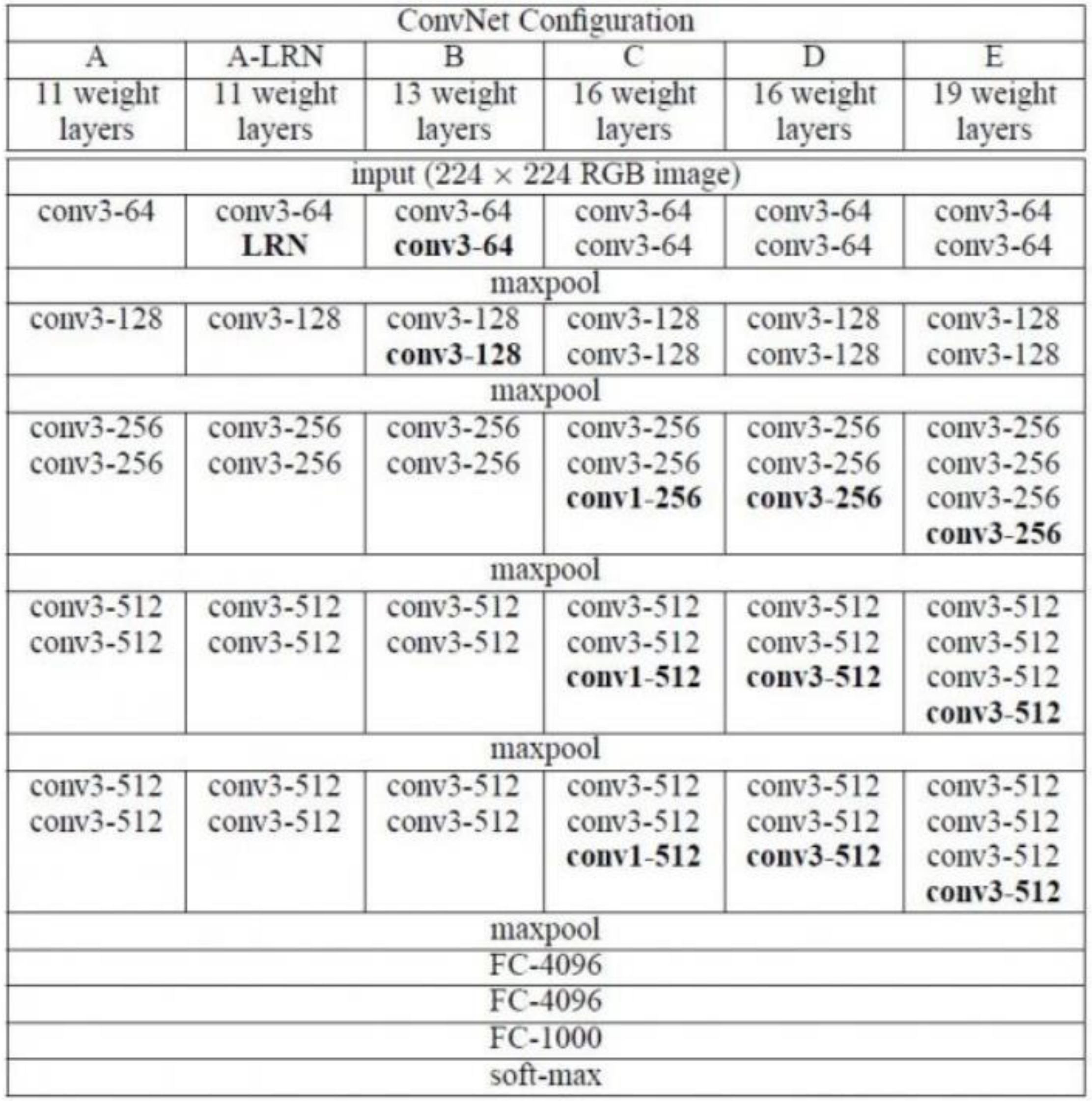
- VGG is an improvement over AlexNet. Its main contribution has been to show that network depth is an essential element for good performance.



VGG's special features

- It replaces the large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) in AlexNet with **multiple 3X3** kernel-sized filters one after another.
- Blocks with the same filter size are applied multiple times to extract more complex and representative features. This concept of blocks/modules became a common theme in networks after the VGG.
- Limitation: It has a lot of parameters (140M). Most of these parameters are in the first fully connected layer.

VGG configurations

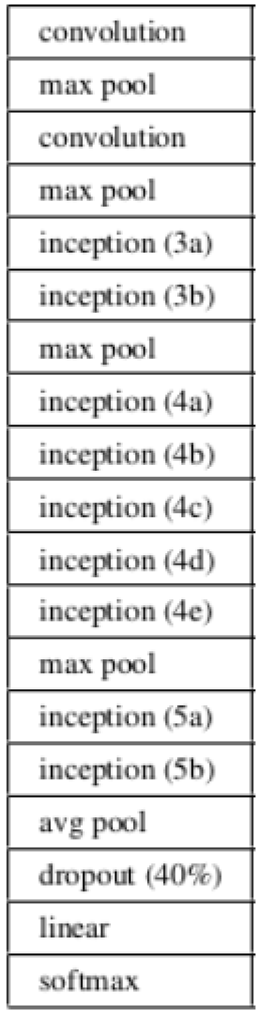


ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

- All configurations follow the generic design present in architecture and differ only in the depth.
  - E.g., 11 weight layers in the network A (8 conv. and 3 FC layers) to 19 weight layers in the network E (16 conv. and 3 FC layers).
- Their final best network is VGG-16 (13 conv. and 3 FC layers).

CNN architectures – GoogleNet

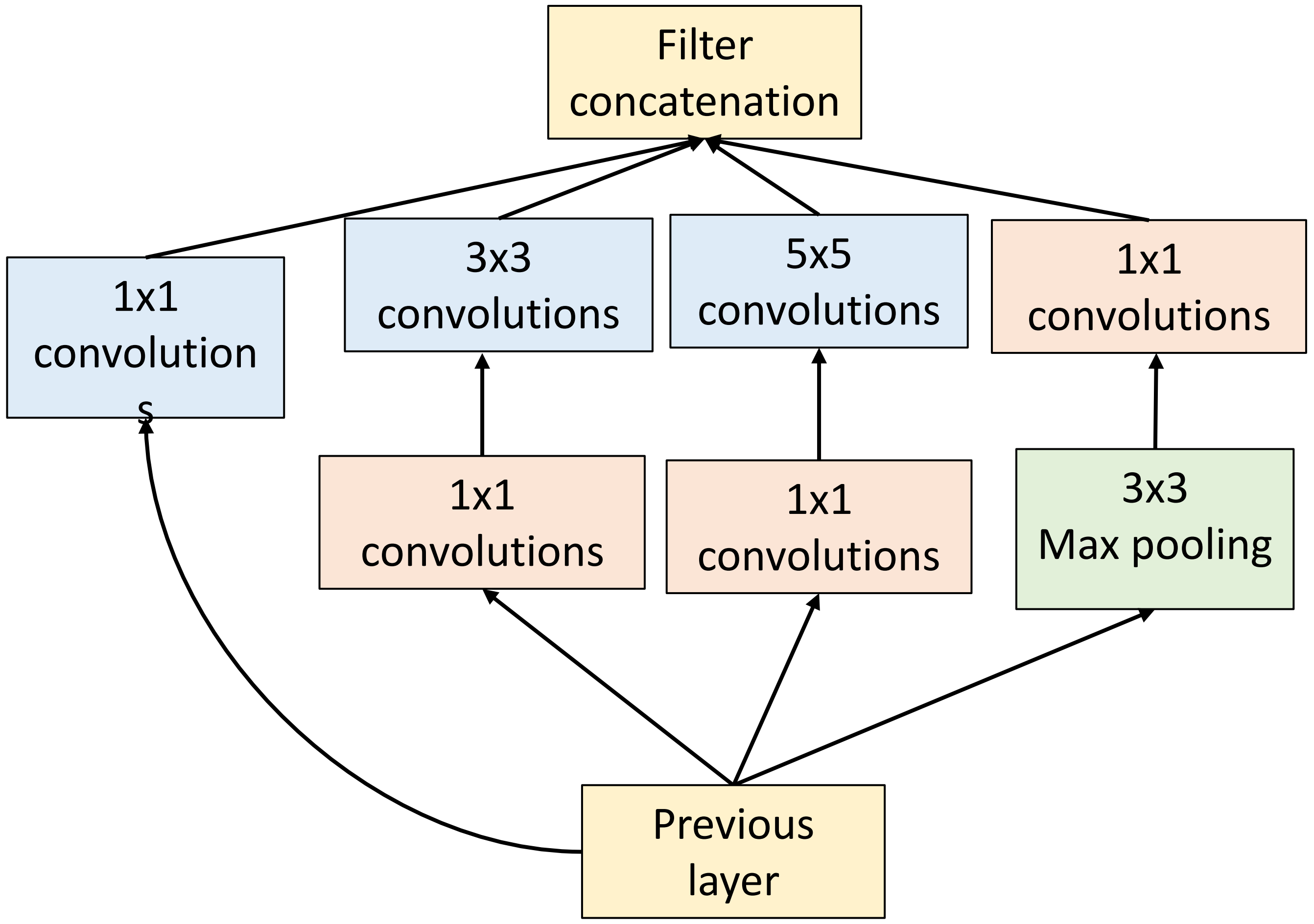
- Googlenet main contribution was the development of an **Inception layer** that dramatically reduced the number of parameters in the network (4M, compared to AlexNet with 60M).



Szegedy, Christian, et al. "Going deeper with convolutions." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015.

Inception layer

An Inception layer is a CNN module combining parallel convolutions of various filter sizes and pooling, concatenated to capture features at multiple scales.



- Inception Layer is a combination of all those layers (namely, 1×1 convolutional layer, 3×3 convolutional layer, 5×5 convolutional layer) with their output filter banks concatenated into a single output vector forming the input of the next stage.
- 1×1 Convolutional layer before applying another layer is mainly used for dimensionality reduction.
- A parallel Max Pooling layer provides another option to the inception layer.

Intuition behind Inception layer

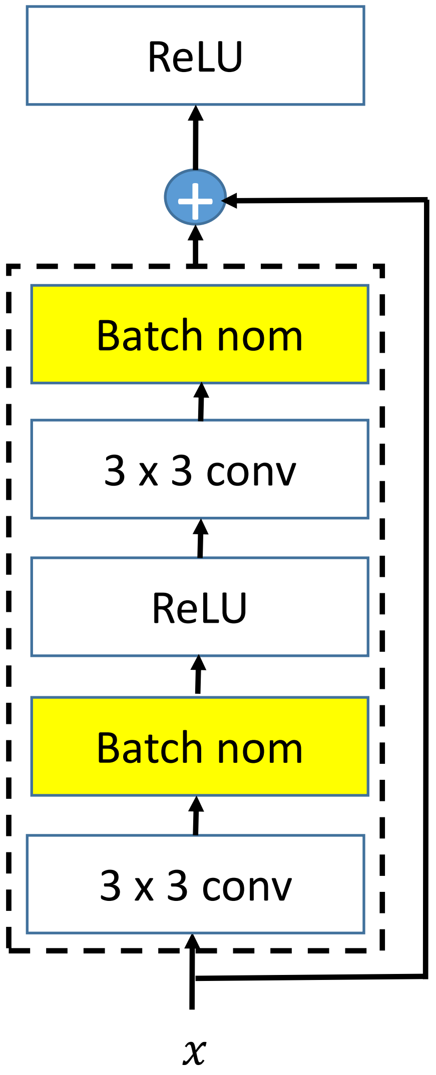
- The intuition behind of inception layer is to allow the internal layers to pick and choose which filter size will be relevant to learn the required information.
  - E.g., it would probably take a lower filter size for the left image and a higher filter size for the right image.

CNN architectures – ResNet

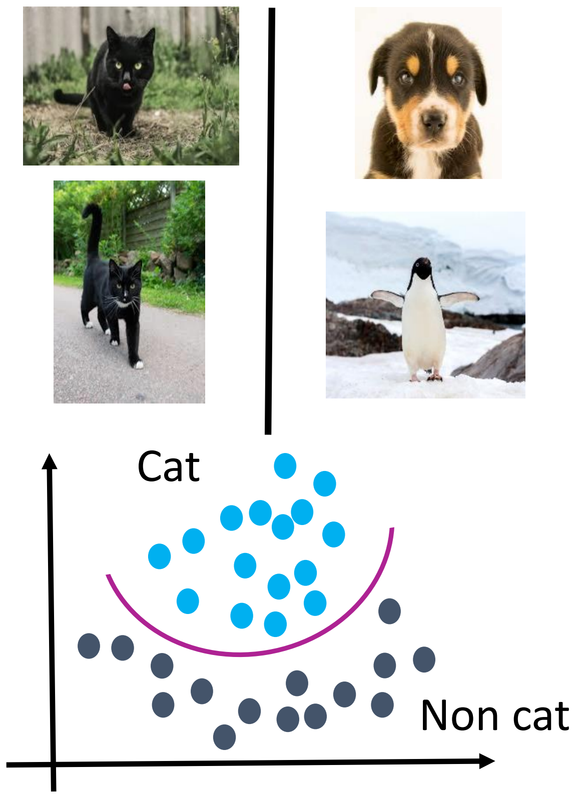
- Problem of deeper networks
  - **Vanishing gradient** – Neglecting the earlier layers.
  - **Curse of dimensionality** – Causes degradation problem. Shallower networks sometimes learn better than their deeper counterparts.
- Residual Network was developed by Kaiming He et al. to solve the above problems.
- ResNet uses **skip connections**, also known as residual connections, which allow gradients to flow through a network directly, bypassing multiple layers without attenuation during backpropagation. These connections add the output from an earlier layer to a later layer, effectively creating shorter paths for the gradient signal and thus mitigating the vanishing gradient problem.

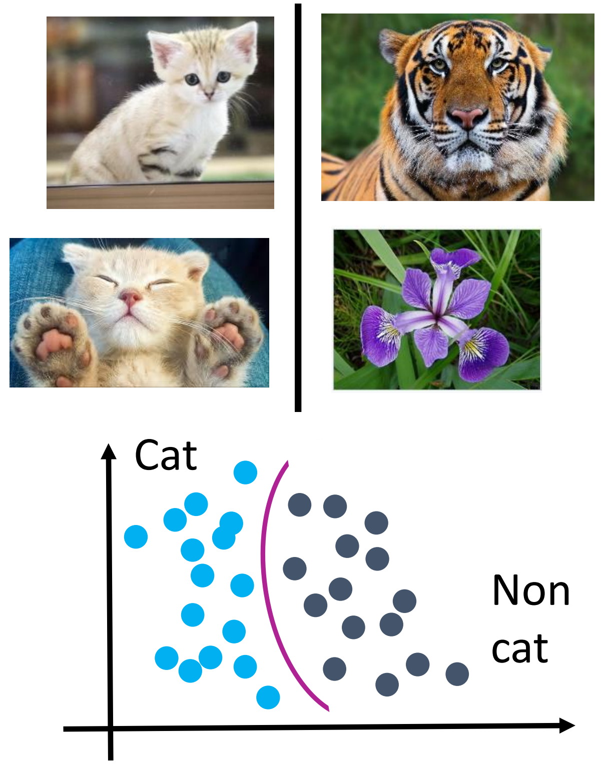
He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

Batch normalisation



- **Batch normalization** (BN) is adopted right after each convolution and before activation.
- BN is used to encounter the internal covariance shift caused by the changes in the distribution of the layers' input.





Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." *arXiv preprint arXiv:1502.03167* (2015).

Objective of this lecture

- To understand what is a convolutional neural network (CNN)
- To learn different optimization techniques
- To explore the state-of-the-art CNN architectures.
- To learn the deep learning design process.
- CNN with Keras

Practical methodology

The recommended practical design process for the application of deep learning techniques:

1. **Determine your goals**: What is the problem you intend to solve? Classification problem? Regression problem? The problem will define the loss function as well as the target label.
2. **Establishing a working end-to-end pipeline**: Establish an appropriate performance metrics such as precision, recall, cosine similarity matrices and etc. And, deployment of baseline models for initial experiment
3. **Determine the bottlenecks in performance**: Overfitting? Underfitting? Defect in the data or software?
4. **Repeatedly make changes to improve the system**: Increase dataset size, adjust hyper parameter, changing algorithms.

Determine your goals

- **Case studies:** As part of the automatic parking system, a new car payment system based on the authentication of license plate numbers will be proposed. An automated system for the registration of cars entering the car park must therefore be put in place.

- **Problem to solve:**
    - Car license plate recognition for automatic parking system
    - Both regression and classification problem involved as you have to detect where the license plate is located (regression) and what the license plate numbers/characters are (classification).

Establishing a working end-to-end pipeline

- **Data collection**: collect raw data and provide label.
- **Performance metrics** for **multi-class classification** (A-Z characters, 1-9 numbers):
  - top-1 or top-5 accuracy
  - Mean average precision (mAP): calculate the average precision (AP) for each class and then average among all classes
- **Performance metrics** for **object detection** (car license plate):
  - Intersection over Union (IOU)
  - Precision and Recall
  - mAP

Establishing a working end-to-end pipeline

- **Default Baseline models**:
  - Determine the complexity of your problem to decide between deep and non-deep approaches. (Since our problem falls into an ‘AI-complete’ category, hence deep learning model is recommended)
  - Choose a model based on the structure of your input data: fixed size vectors (FC), image (CNN), sequential data (RNN). (Our problem will be dealing with images, hence CNN is recommended)
  - CNN architecture for object detection: R-CNN, Fast- RCNN, Faster-RCNN, YOLO (we will discuss this in the future lecture)
  - CNN architecture for character classification: AlexNet or VGG. (Choose the network according to the complexity of your data)

Problems encountered

- Problems that may be encountered in modeling deep neural network:
  - Overfitting
  - Underfitting
  - Gradient exploding
  - Gradient vanishing
  - Data and software defect

Recap: Overfitting

- What is overfitting?
  - A model that can fit well to the training data, but not generalized to new and unseen testing data.
  - Low bias, high variance

- Normal causes of overfitting:
  - Model is too complex.
  - Model was trained with a small amount of training data.

Solution to Overfitting

- **Gather more training data**
- **Dataset augmentation**: Translation, flip, mirror, noise, colour augmentation and etc.
- **Reduce** the **complexity** of the network: increase number of neurons, layers and weights parameters.
- Apply **regularization** techniques: L1/L2 weight decay, dropout, early stopping

Recap: Underfitting

- What is Underfitting?
  - A model that performs poorly on both the training and testing data
  - High bias, low variance

- Normal causes of Underfitting:
  - Model is too simple.

Solution to Underfitting

- **Increase** the model **complexity**: add new layers, increase number of neurons
- **Increase training time**
- **Reduce dropout**

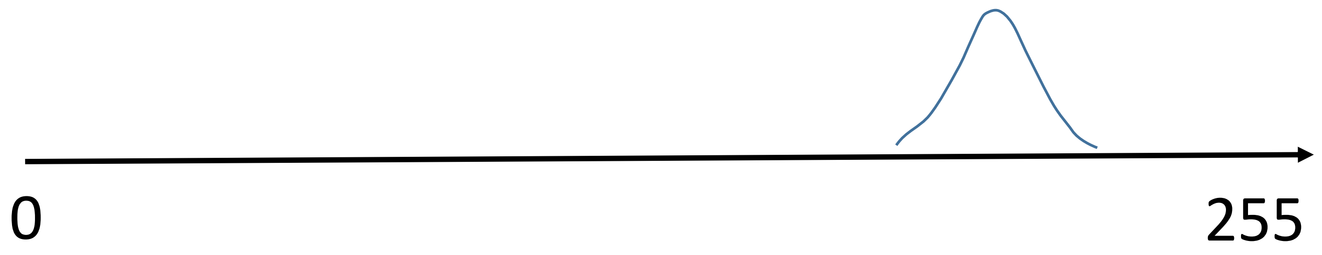
Gradient Exploding

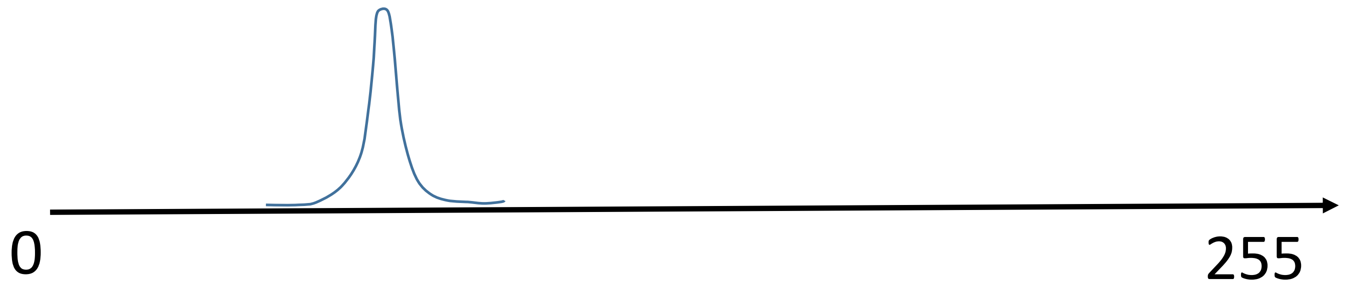
- Gradient exploding is a problem in training deep neural networks where gradients can grow exponentially during backpropagation, causing very large weight updates and leading to numerical instability.
- This often results in model training failure, with weights becoming NaN (not a number) or diverging to infinity. It's especially prevalent in architectures with many layers.

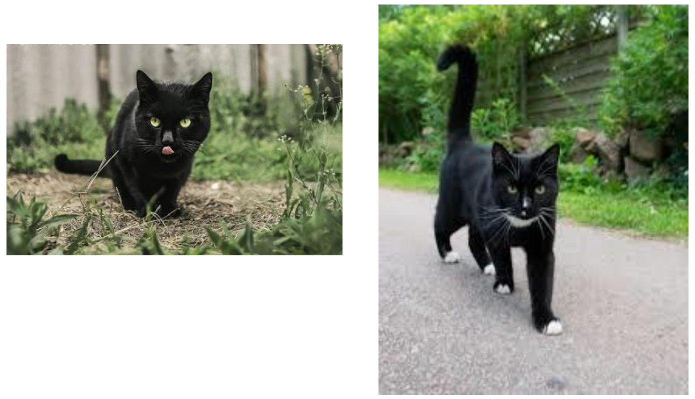
Solutions to Gradient Exploding

- **Gradient clipping:** forcing the gradient values (element-wise) to a specific minimum or maximum value if the gradient exceeded an expected range.
- Use **smaller learning rate**
- Weight regularization
- Data preparation: **data normalisation**
- Re-design the network to have fewer layers.

Data normalisation



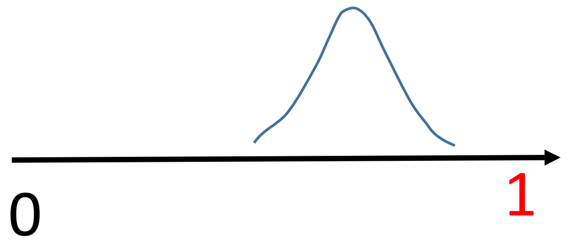




Data normalisation

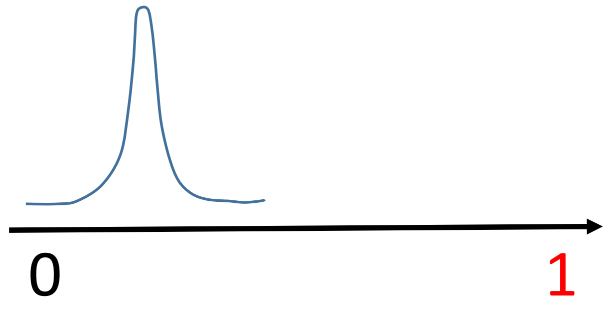
- Min-max scaling
  - Data is scaled to a fixed range, usually 0 to 1.

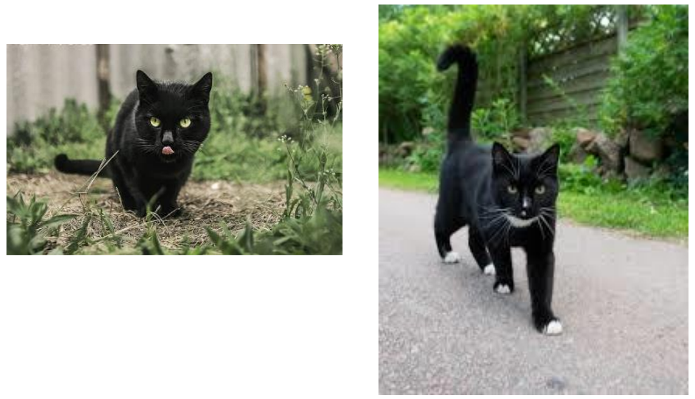
$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$



- Standardization
  - Data is rescaled to having properties of **μ** = 0, **σ** = 1

$$x_{norm} = \frac{x - \boldsymbol{\mu}}{\boldsymbol{\sigma}}$$





Gradient Vanishing

- Gradient vanishing occurs when gradients become increasingly small during backpropagation, causing weights in earlier layers to barely update.

Solution to Gradient Vanishing

- Use **ReLU** activation instead of Sigmoid: Sigmoid function squishes a large input space into a small input space between 0 and 1. Therefore, a large change in the input of the sigmoid function will cause a small change in the output. Hence, the derivative becomes small.
- Use **residual** based network: It provides residual connections straight to earlier layers.
- **Batch normalization**: It forces the activations of a layer to follow a single distribution, independent of the changes in the parameters of upstream layers
- Combine careful **initialization of weights**

Debugging strategies

- **Qualitative analysis**: Observe the windows detected by the trained model to find out whether car license plates are captured.
- **Quantitative analysis**: Examine the misclassification made by the network using the returned softmax probability. For example, the model may incorrectly identify the character Q as 0, which may be due to incorrect labels of the data.
- **Fit a tiny data**: Bugs in algorithms are often difficult to catch. Try to fit the model using a tiny data (says 5 or 10 samples) and check the results.
- **Monitor the learning curves**

Objective of this lecture

- To understand what is a convolutional neural network (CNN)
- To learn different optimization techniques
- To explore the state-of-the-art CNN architectures.
- To learn the deep learning design process.
- CNN with Keras

TensorFlow and Keras

- TensorFlow: An open-source machine learning library developed by Google for complex computations, supporting scalable deep learning models with a flexible architecture.
- Keras: A high-level neural networks API, running on top of TensorFlow, designed for human-friendly, rapid experimentation with deep learning.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten
```

Define a dense model

```
# Define the model
model = Sequential([
    # Add a Dense layer with 64 units and ReLU activation function
    Dense(64, activation='relu', input_shape=(784,)),
                        # here 784 for flattened 28x28 images
    Dense(64, activation='relu'),
    # Add a Dense layer with softmax for a 10-class classification problem
    Dense(10, activation='softmax')
])
```

In addition to the Dense layer, there are other layers such as Flatten, Dropout, Conv2D, and so on.

Define a dense model – another syntax

```
# Define the model
model = Sequential()

# Add a Dense layer with 64 units and ReLU activation function
model.add(Dense(64, activation='relu', input_shape=(784,)))
                    # here 784 for flattened 28x28 images

model.add(Dense(64, activation='relu’))

# Add a Dense layer with softmax for a 10-class classification problem
model.add(Dense(10, activation='softmax'))
```

Compile the model

```
# Compile the model
model.compile(optimizer='adam',  # Optimizer
          loss='categorical_crossentropy',  # Loss function to use
          metrics=['accuracy'])  # Metrics to monitor

# Print model summary
model.summary()
```

Train, evaluate and make predictions

```
# Train the model for 10 full cycles, in batches of 32 samples
model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=2)
    # the verbose parameter controls the verbosity of the output.
    # 0: no output 1: progress bar 2:  output one line for each epoch.

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test, verbose=2)
print(f"Test loss: {loss}")
print(f"Test accuracy: {accuracy}")

# Make predictions
predictions = model.predict(X_test)
```

Build a convolutional model (CNN)

```
# Define the model
model = Sequential([
    # Convolutional layer with 32 filters, a kernel size of 3x3, and ReLU activation
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),

    # Max pooling layer with a pool size of 2x2. Without explicitly setting the strides
    # parameter, the strides default to the same value as the pool size.
    MaxPooling2D((2, 2)),        # MaxPooling2D(pool_size=(2, 2), strides=(2, 2))

    # Flatten the output of the pooling layer to feed into a dense layer
    Flatten(),
    # Dense layer with 64 units and ReLU activation
    Dense(64, activation='relu'),
    # Output layer with 10 units and softmax activation (for a 10-class classification)
    Dense(10, activation='softmax')
])
```

Compile and train the CNN

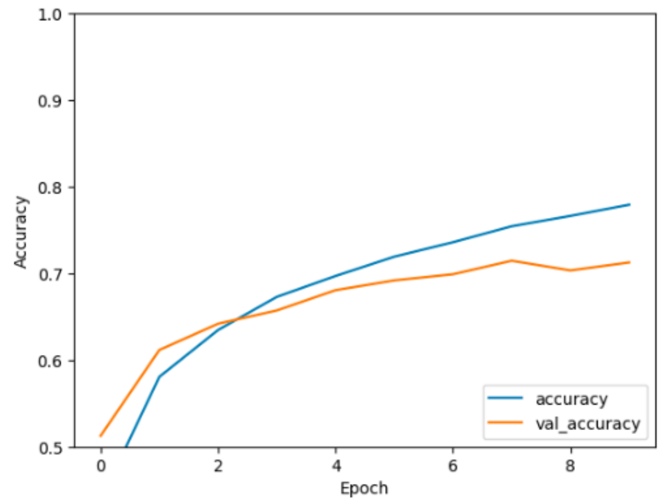
```
model.compile(optimizer='adam',
          loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
          metrics=['accuracy’])
# logits=true means that the inputs are expected to be raw called logits
# which is directly from the output of the neural network's last layer.
# Internally, the SparseCategoricalCrossentropy loss function will apply the
# softmax function to these logits to obtain probabilities, which are then used
# to compute the cross-entropy loss.

history = model.fit(train_images, train_labels, epochs=3,
                validation_data=(test_images, test_labels))
```

Evaluate the CNN

```
# Plot training and validation accuracy values
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy’)

plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right’)
```



```
# Evaluate the model with the test dataset and print the test accuracy
test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
print(test_acc)
```

Make predictions with CNN – read images

SWINBURNE UNIVERSITY OF TECHNOLOGY SARAWAK CAMPUS

```
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image        # pip install pillow
image_paths = ['/content/drive/My Drive/dog.png', '/content/drive/My Drive/cat.png']
images = []   # Initialize an empty list to store the image arrays
for path in image_paths:
    image = Image.open(path) # Load the image
    image_rgb = image.convert('RGB’) # Convert the image to RGB
    image_resized = image_rgb.resize((32, 32)) # Resize the image to 32x32 pixels
    # Convert the resized image to a numpy array and append to the list
    images.append(np.array(image_resized))
normalized_images = [image / 255.0 for image in images]
```

Make predictions with CNN – plot images

```
# Plot the images in a grid
N = 2  # Number of images you want to display
plt.figure(figsize=(10, 10))
for i in range(N):
    plt.subplot(2, 3, i + 1)   # 2 row 3 column grid, plot at the  i+1 position
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(normalized_images[i])
plt.show()
```

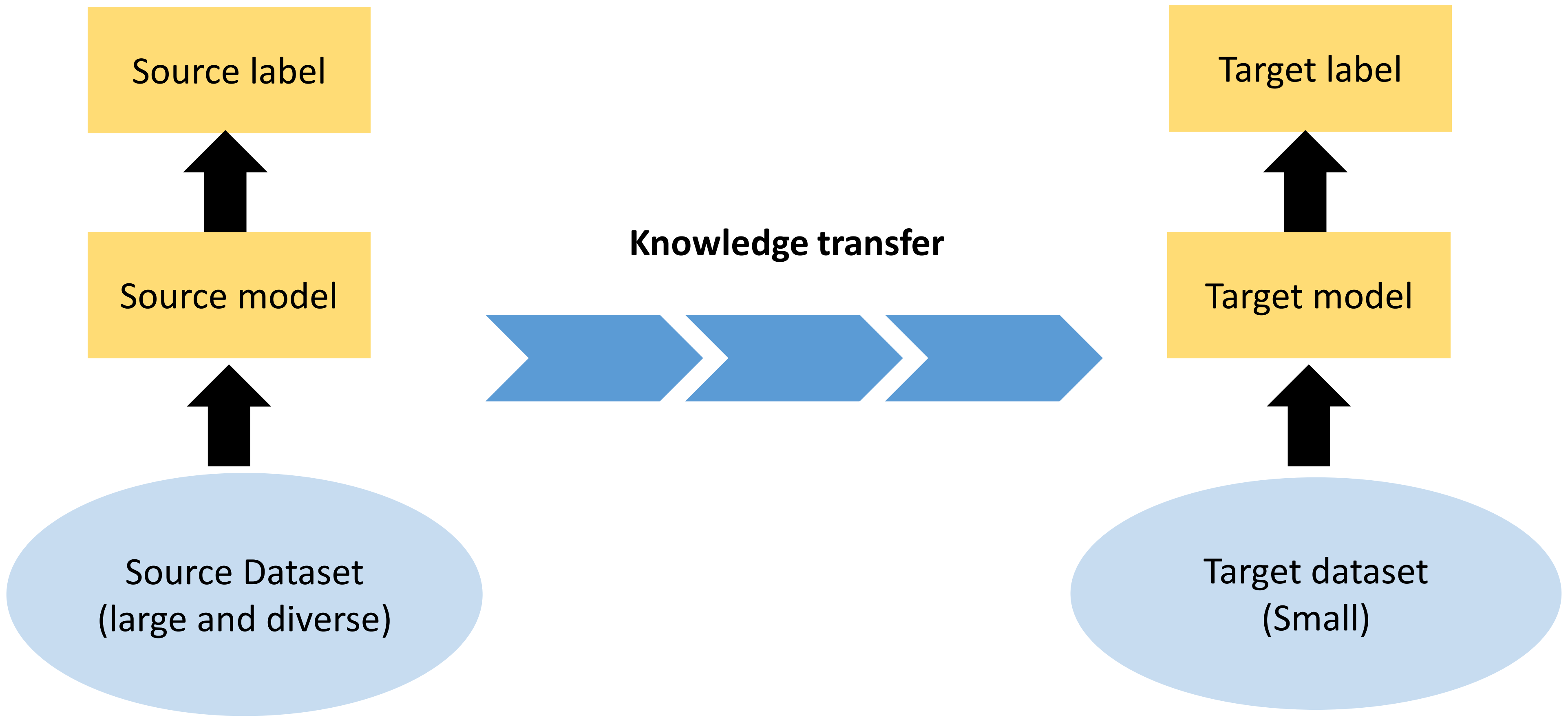
Make predictions with CNN – classify images

```
# Ensure normalized_images is a NumPy array
normalized_images_array = np.stack(normalized_images)
# Make predictions
predictions = model.predict(normalized_images_array)

# Process predictions for each image
predicted_classes = np.argmax(predictions, axis=1)
    # axis=1, argmax returns the index of the maximum value in each row.
# Assuming 'class_names' is a list or function mapping indices to class names
predicted_class_names = [class_names[i] for i in predicted_classes]
for i, predicted_name in enumerate(predicted_class_names):
   print(f'Image {i}: Predicted class: {predicted_name}')
```

Next lecture

- Transfer learning


Source label
Source model
Source Dataset (large and diverse)
Knowledge transfer
Target label
Target model
Target dataset (Small)