เฉลย 1 เฉลี่ย

2 ต้องคำนวณ

3 ได้ 5

4 Padding Input = Output

5. CNN นำไปใช้ [illegible] จริง

# CONVOLUTIONAL NEURAL NETWORK (CNN)

6. พ่อแม่ → ลูก cross over

7. สุ่มค่าใหม่ ๆ Mutation

8. รุ่นต่อไป replacement

9. fitness ดี

10. Global Minimum

Asst.Prof.Dr.Supakit Nootyaskool
IT-KMITL

# Study map

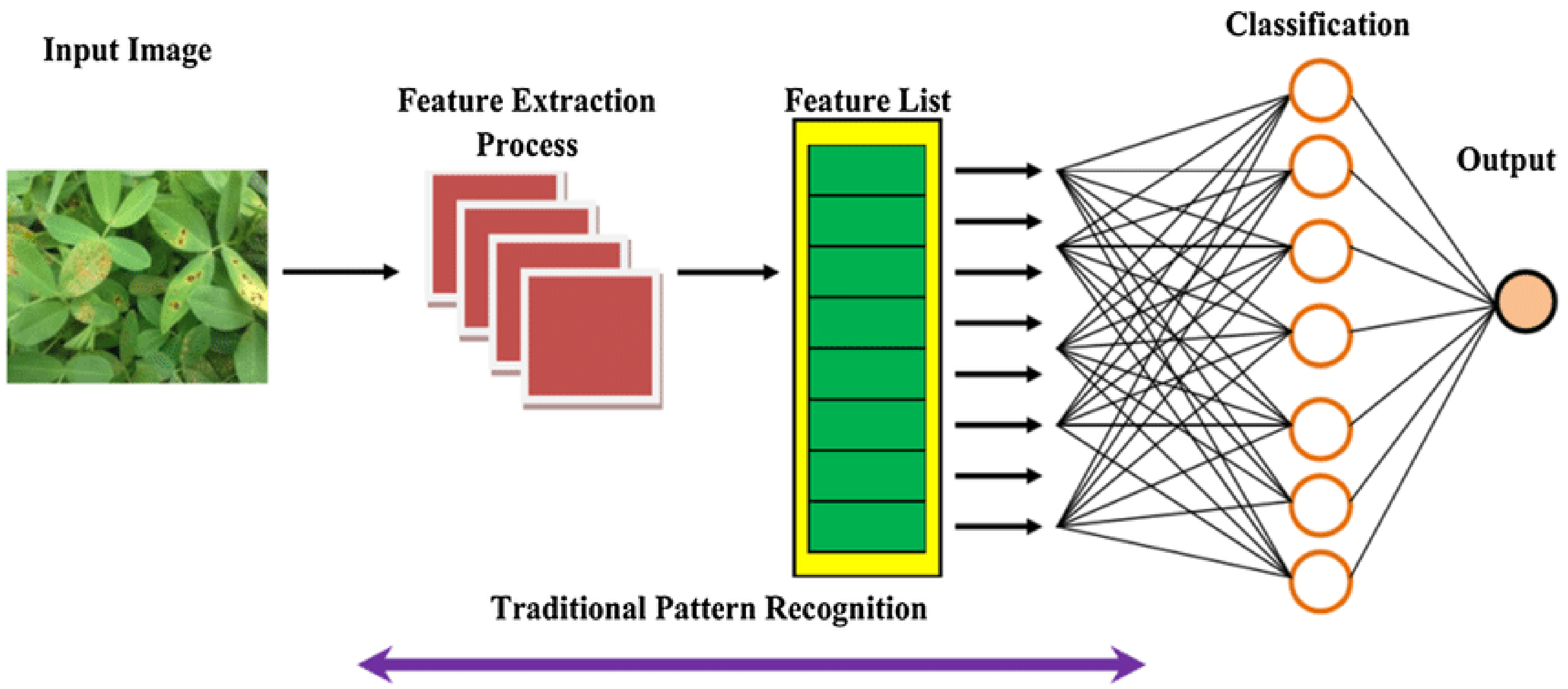
- 1.Basic programming
  - R-programming
- 2. Perceptron
  - Activity function
- 3. Feed Forward NN
  - Logistic function
- 4. Feed Forward NN
  - XOR gate
  - Multi-layer perceptron
- 5. Example & Library Feed Forward NN
  - N:N, 1:N model
  - iris dataset
- 6. Writing NN Code
  - Data scaling, Confusion matrix
  - Writing NN code
- 7. Recurrent Neural Network
- 8. Apply RNN & Library
- 9. GRU LSTM
- 10. CNN
- 11 Apply GA to NN

# Topics

- From traditional to modern image recognition
- Convolutional Neural Network
- Researches relate to CNN
- Develop CNN in R

# FROM TRADITIONAL TO MODERN

# Feature Extraction Process




Feature extraction is a part of the dimensionality reduction process, by

- getting the most important characteristic,
- reducing number of the variable.

# General Feature extraction in image processing

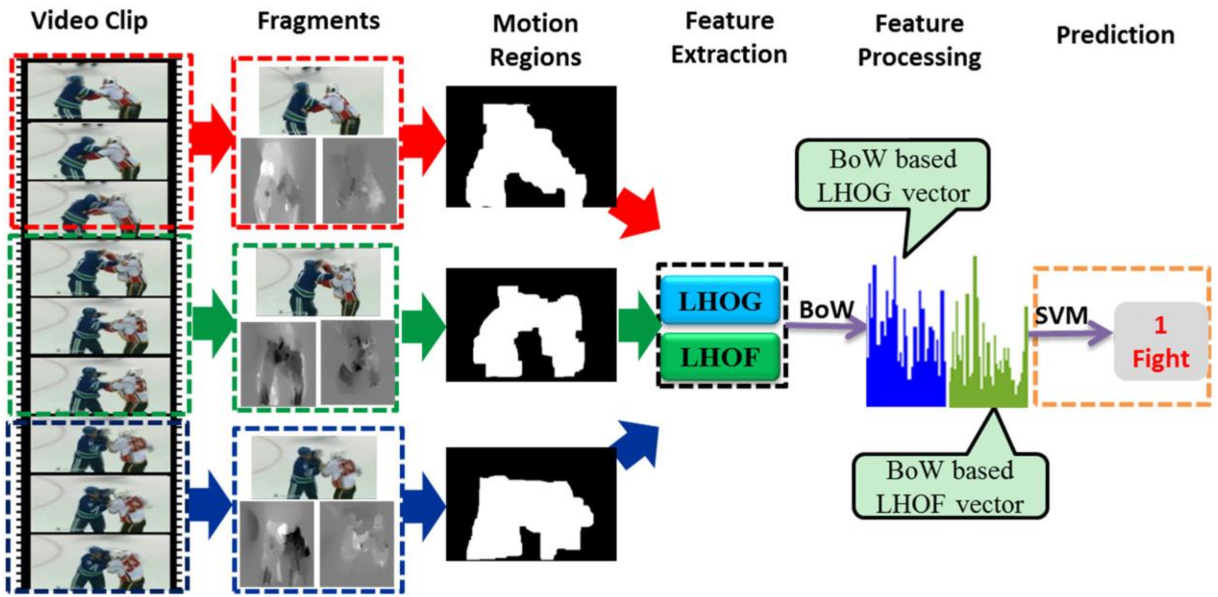
Original image

Applied Laplacian 3*3

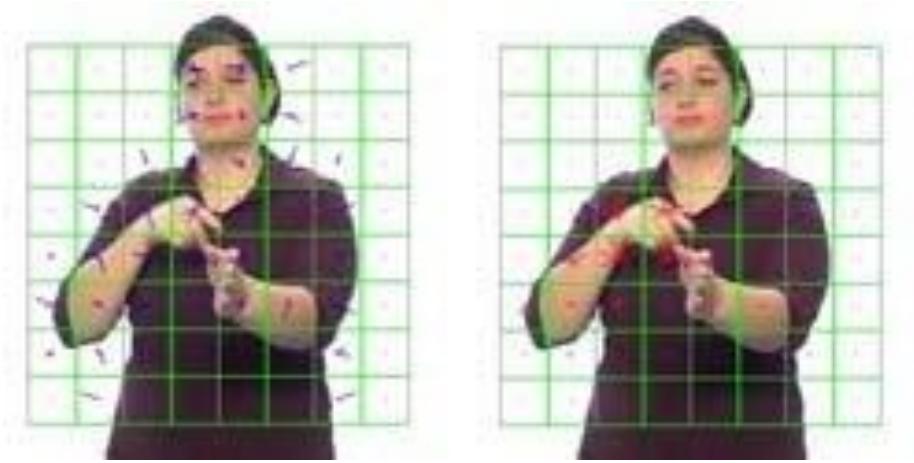
Applied Laplacian 5*5



# Example—Feature extraction in VIDEO



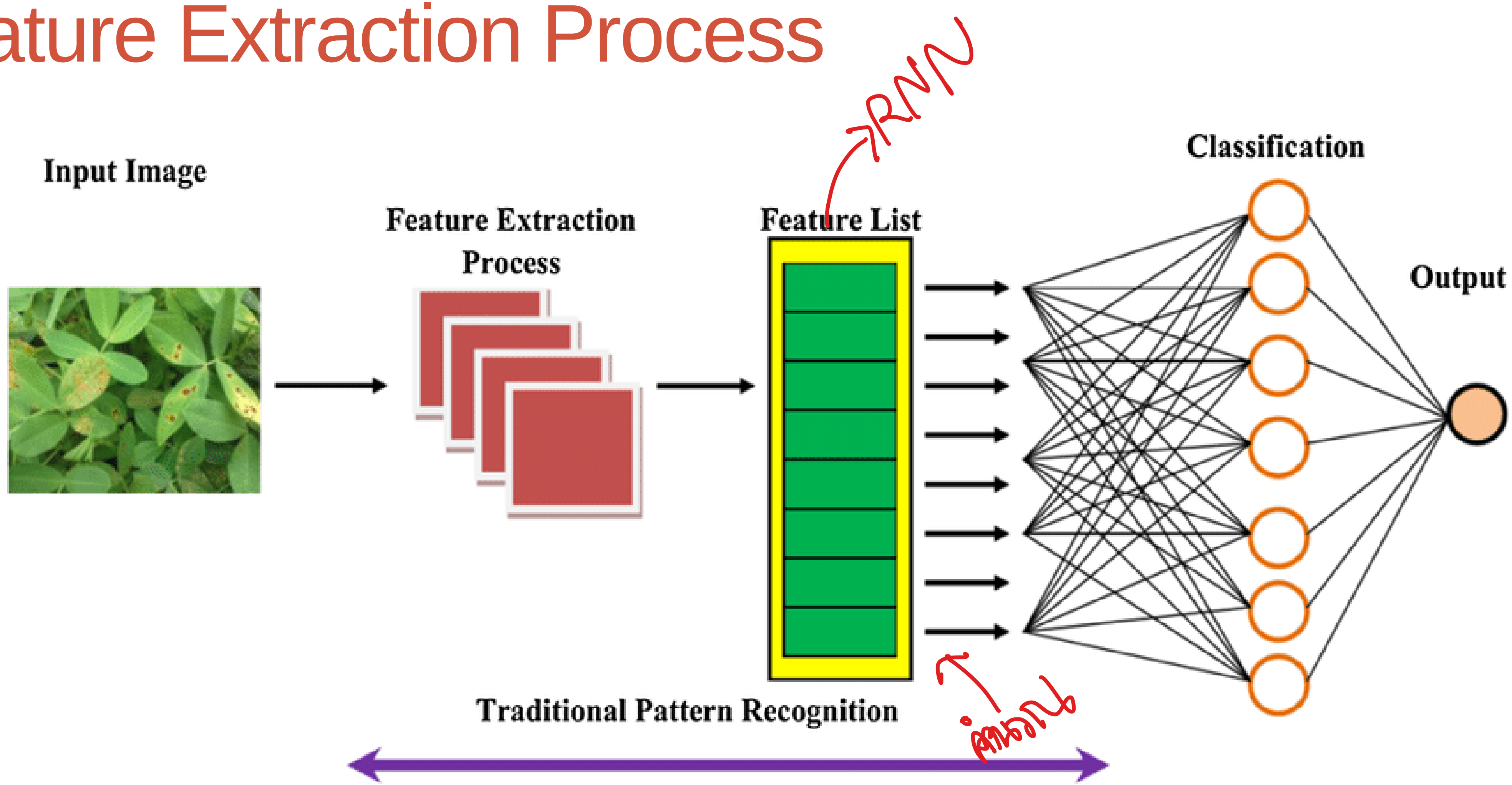

Violence detection in surveillance video using low-level features



DICTA-SIGN: Sign language recognition, generation and modelling with application in deaf communication

- Histogram of Gradients (HOG)
- Histogram of Optical Flow (HOF)

# Feature Extraction Process




A key point of deep learning is the process of automatic feature extraction.

# Image Recognition System

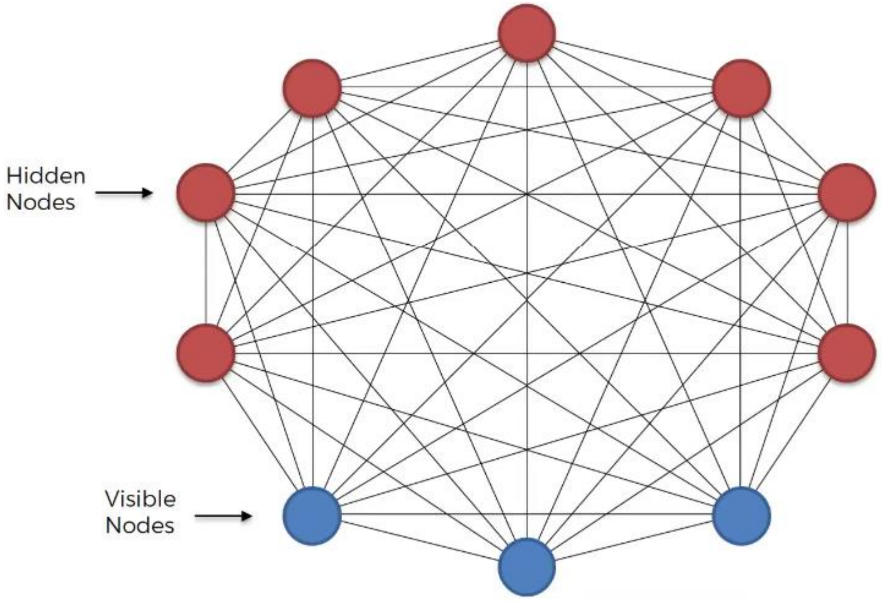
- Traditional
  - Consider/select feature extraction.
  - Researcher considers the feature-extraction techniques
- Modern
  - Apply automatic feature extraction
  - Automatic generate new feature with the system

# Four types in deep learning concepts

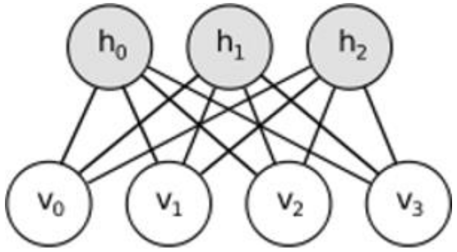
- Boltzmann Machine, 1985
- Autoencoders
- Convolutional Neural Network

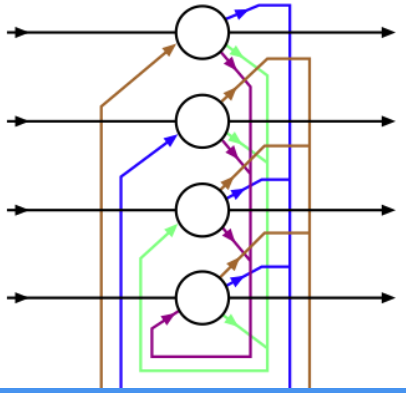
Guo Y, Liu Y, Oerlemans A, Lao S, Wu S, Lew MS (2016) Deep learning for visual understanding: a review.

# Boltzmann Machine (BM)




- First Introduced by Geoffrey Hilton, 1985.
- All node in the same layer.
- Relate to Markov random field
- Bidirectional connection
- Uses stochastic approach and likes Hopfield network
- In side consists of
  - Global energy (E)
  - State (0/1)
  - Weight
  - Bias

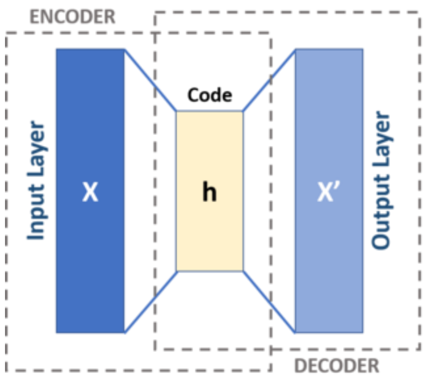


$$E = -\left(\sum_{i<j} w_{ij}\, s_i\, s_j + \sum_i \theta_i\, s_i\right)$$

# Autoencoder

สร้างโค้ดให้หลีบนโค้ด

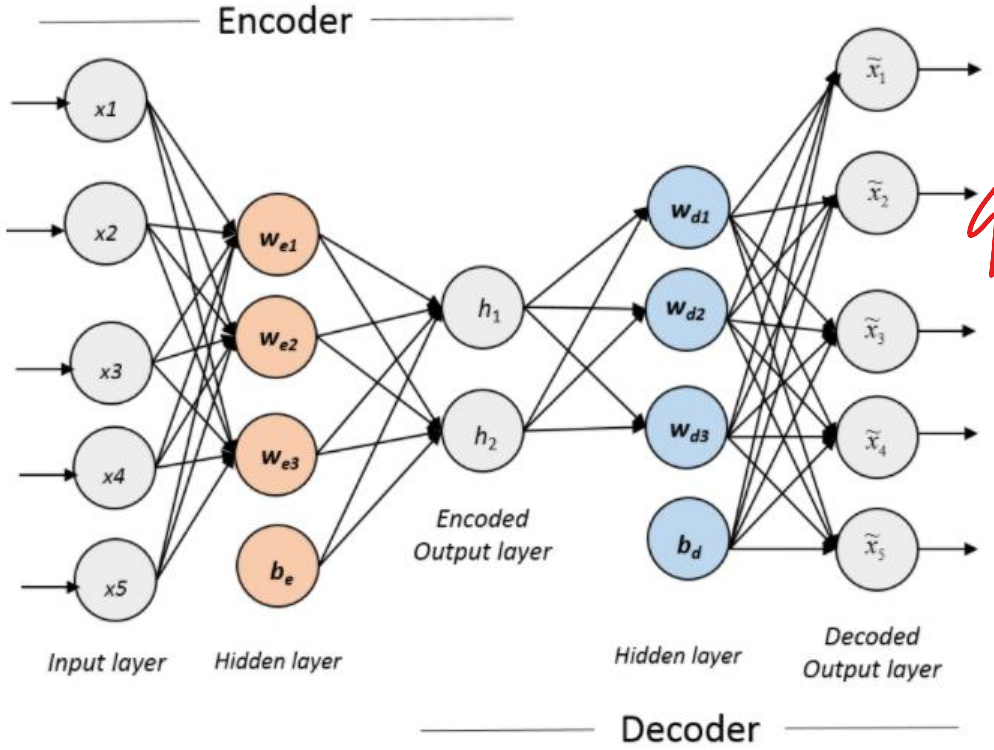



- System has two main parts,
  - Encoder
  - Decoder (output)
  - Output units are directly connected back to input units.
  - Attempts to reconstruct by decoding Xn back.
- Example

  (En)A → 41H ….41H → A (De)

$$h = f(x) = \sigma(W_e^T X + b_e)$$

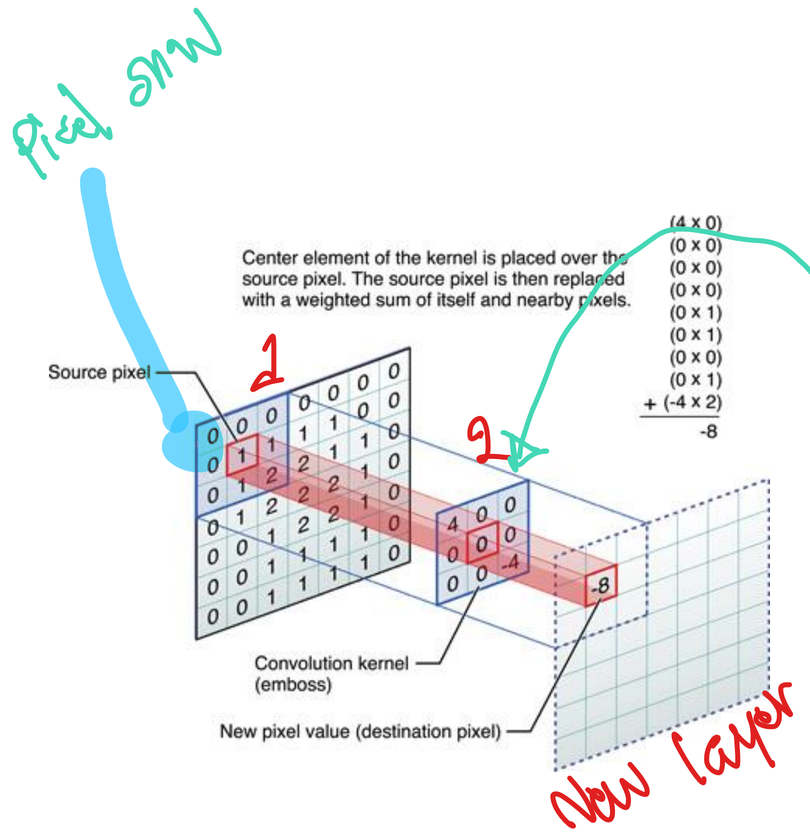
$$\widetilde{X} = f(h) = \sigma(W_d^T h + b_d)$$

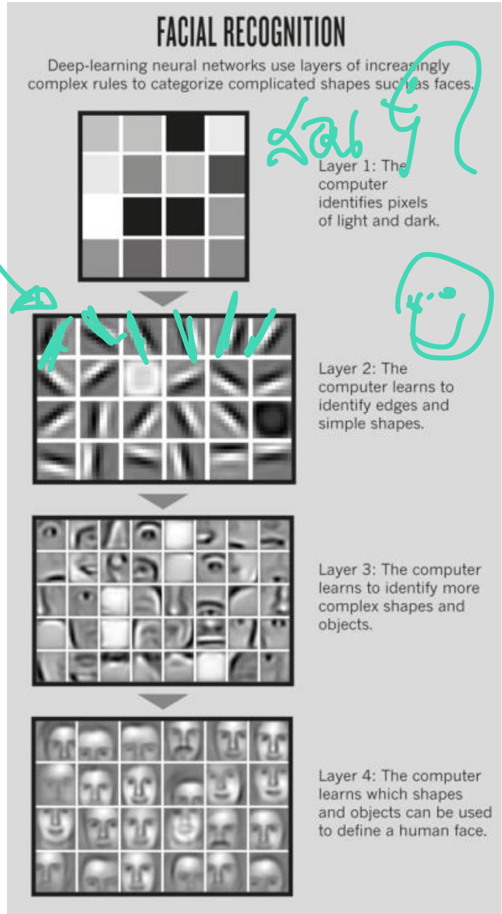



ขปอน

# Convolutional Neural Network

- Rather similar to BM and Autoencoder
- CNN difference on
  - Learning single-global-weight matrix between two layers
  - Mostly used in image recognition.
  - Many levels of identify image.
- "convolution" comes from filter operators




https://developer.apple.com/library/ios/documentation/Performance/Conceptual/vImage/ConvolutionOperations/ConvolutionOperations.html

# CONVOLUTION NEURAL NETWORK

Level 8 bit — 0-255
ระดับ

# Image and Pixel

A pixel

pixel สี

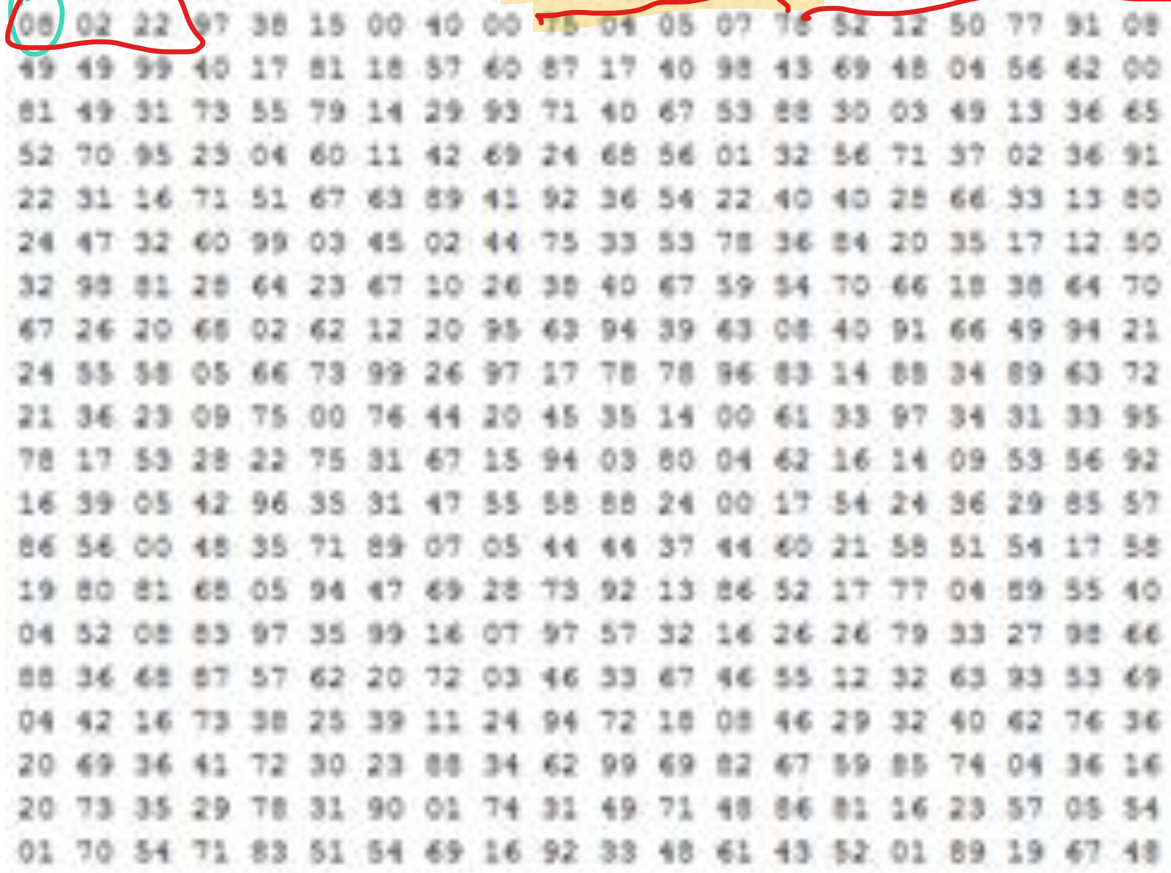
สี RGB

R G B

Level of brightness
Red = 0 to 255
Blue = 0 to 255
Green = 0 to 255
True color = R * B * G = 16.7 Million color levels

เลขแทนไฟล์ภาพ

ดำ จนขาว

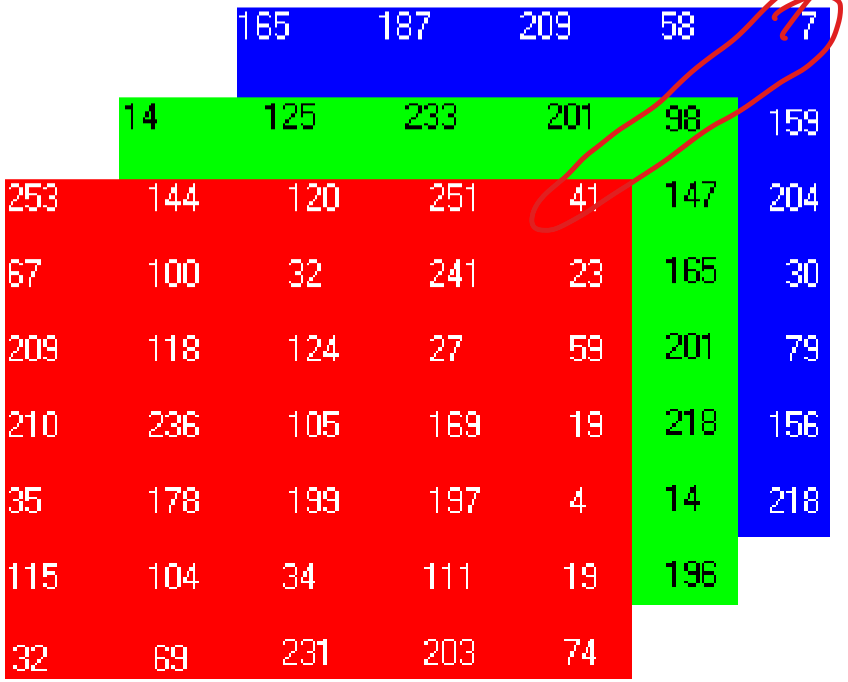


What We See

What Computers See

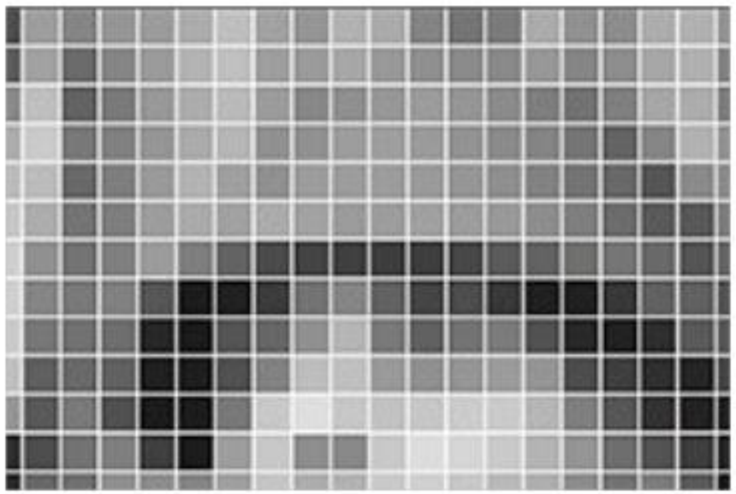
ขาว-ดำ 1 bit

# RGB color vs Grayscale




ตรงปลาย
เอาเลขมารวมกัน
÷3 ได้
Gray Scalle

บวก ÷3 + B



ส่วนเท่ากัน

สีไม่สำคัญ แปลง RGB → Gray scalle

# Color2Gray

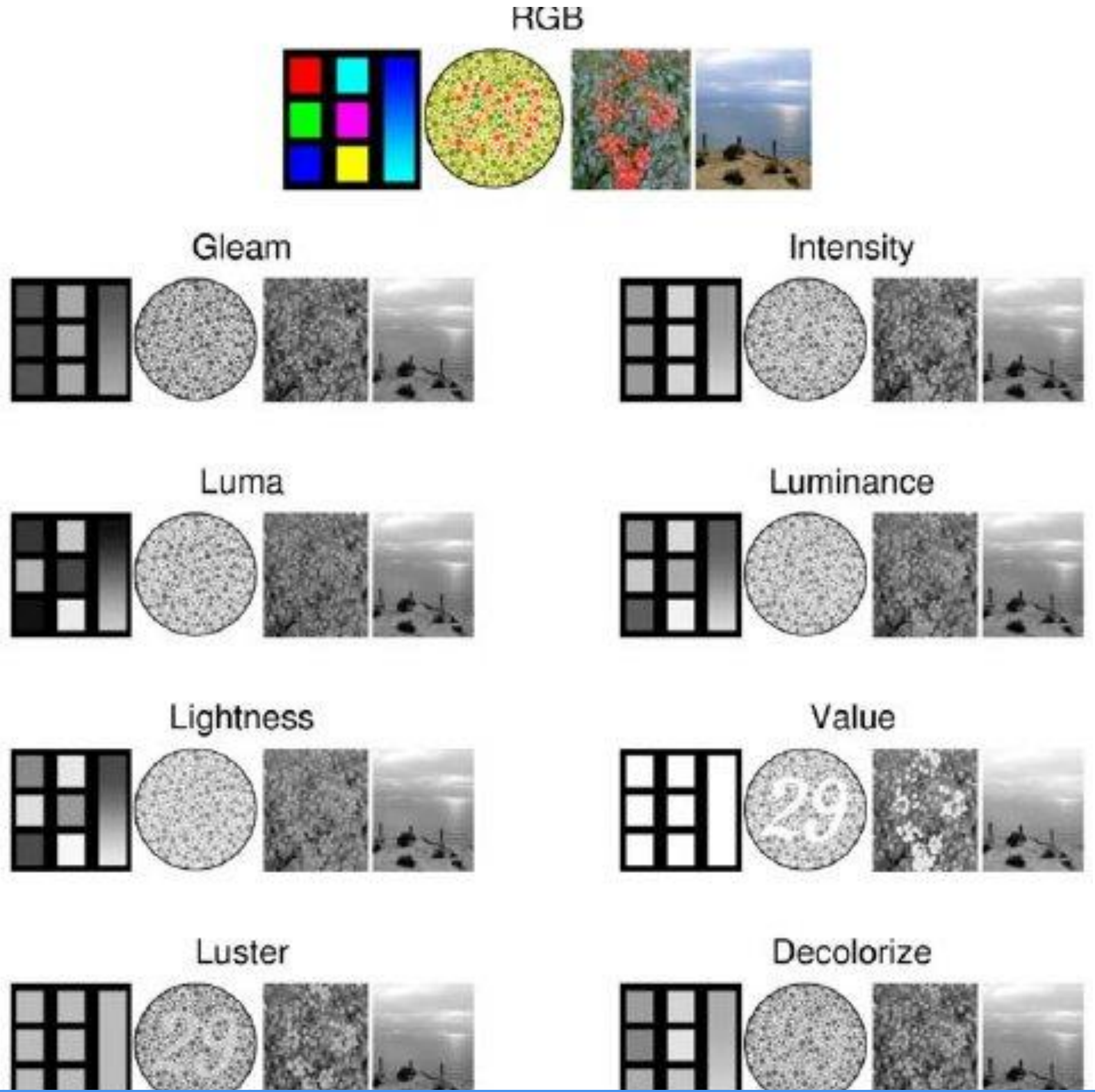
The Color2Gray algorithm is a 3-step process:

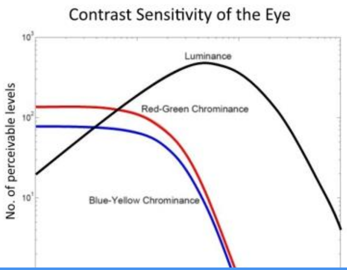
1) convert RGB inputs to a perceptually color space,

ความสว่างของสี ค. สว่าง

2) use chrominance (color) and luminance (brightness) differences to create grayscale target differences between nearby image pixels,

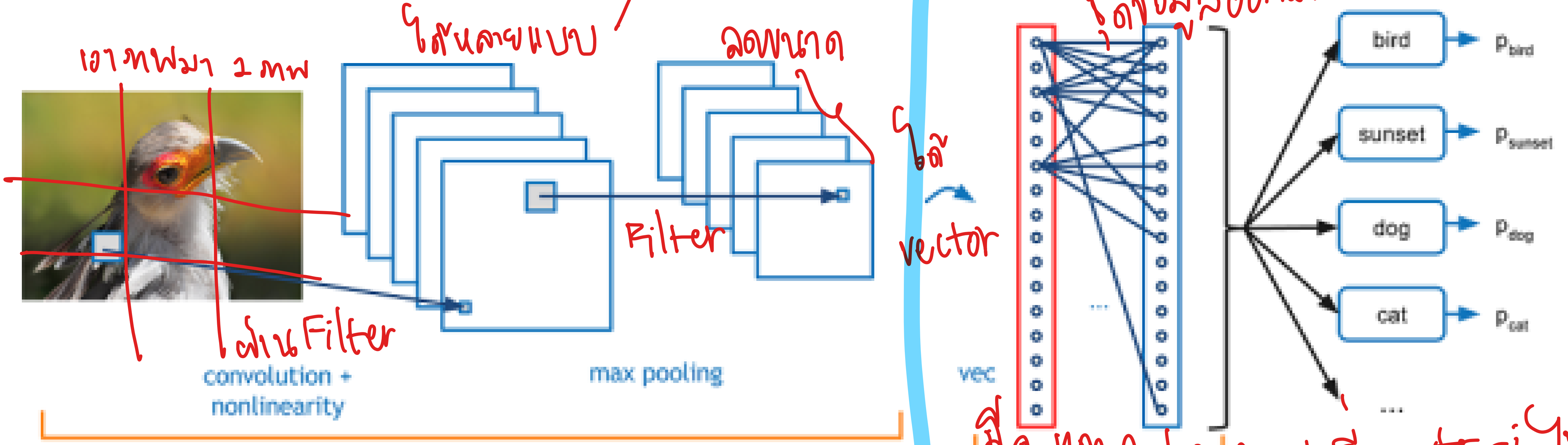
3) solve an optimization problem designed to selectively modulate the grayscale representation as a function of the chroma variation of the source image.



# CNN Concept

เพื่อดึงสิ่งเด่นๆออกมา




Convolution + Pooling layers

Fully connected layers

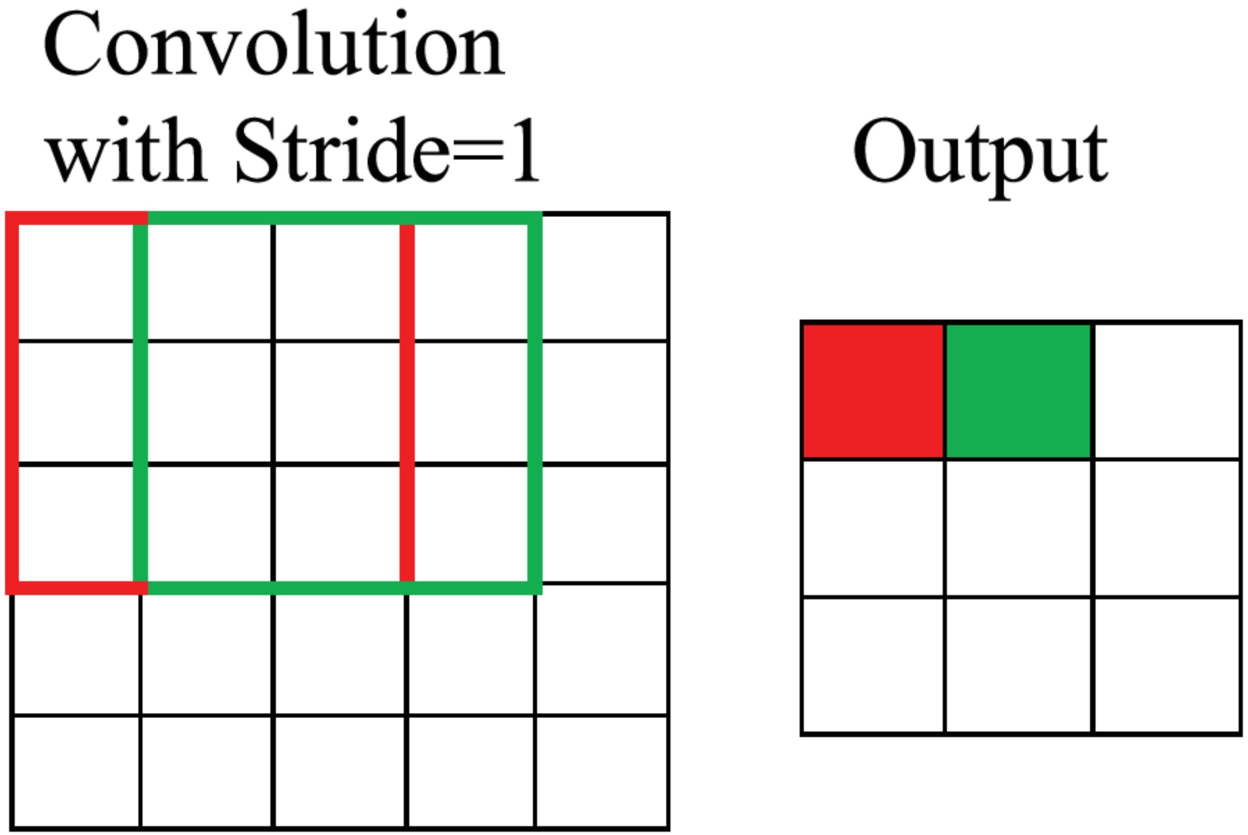
มี ค.แตกต่างมากพอที่จะ train ได้

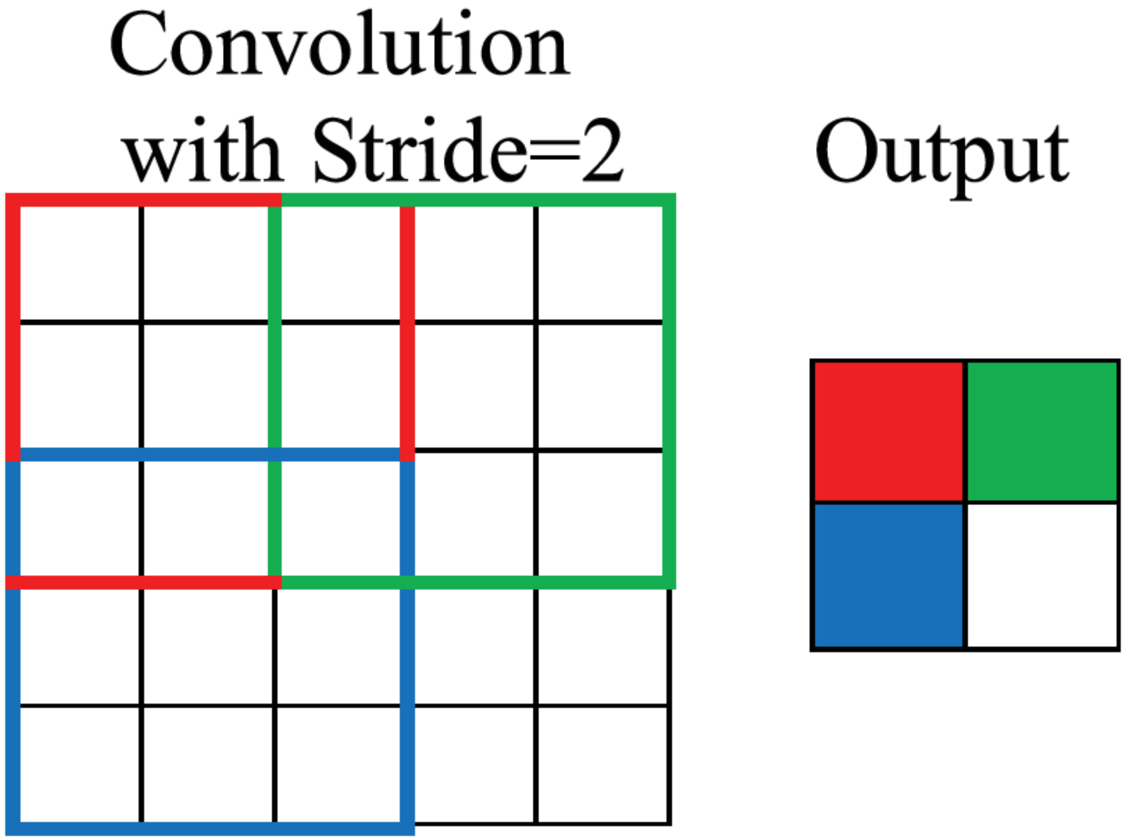
ถ้าคำนวณเป็น Parallel ได้
แบ่งๆ กล่อง แยกประมวลผลได้

A

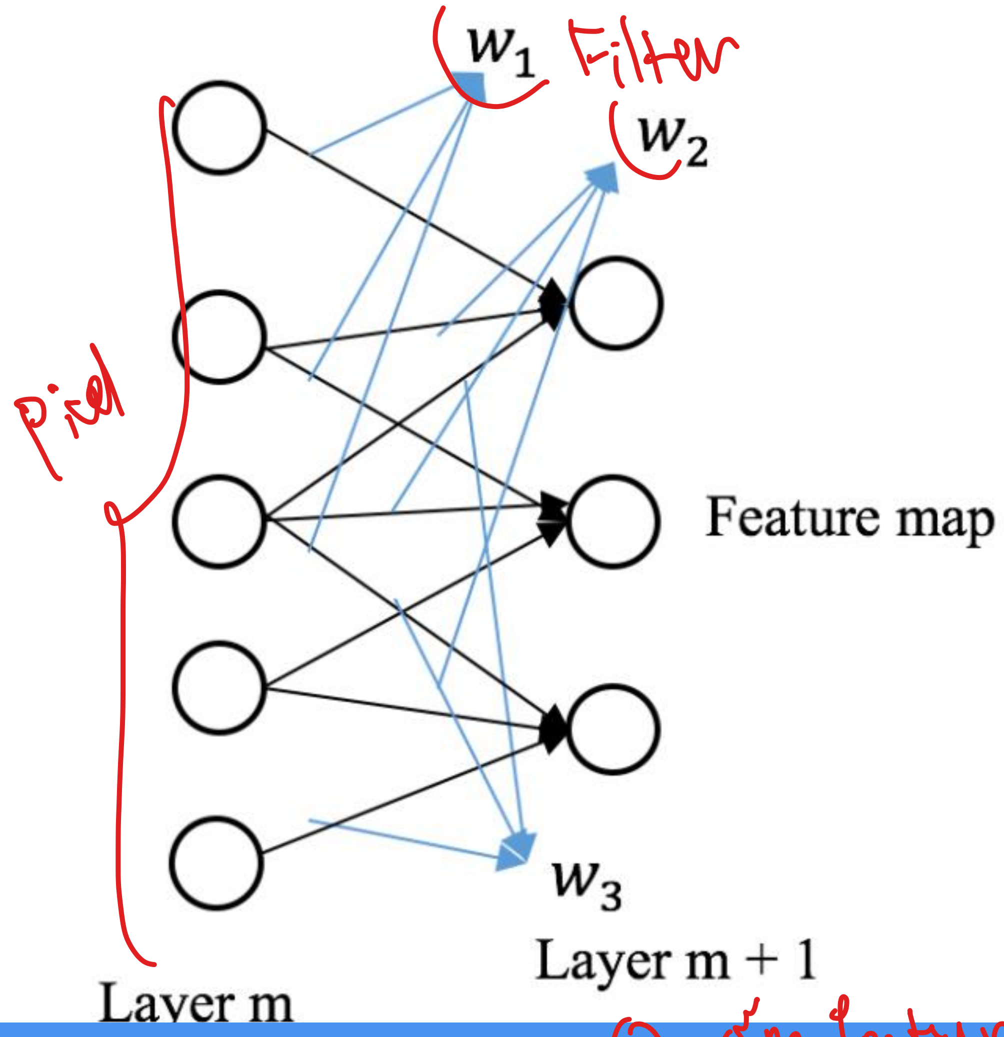
B

# Convolutional Layer: Stride



# Convolution likes weight adjustment




- Layer m has five nodes, and the filter has three units [w1, w2, w3]. We compute the dot product between the filter and the first three nodes in layer m and obtain the first node in the feature map
- Convolutional layers are used to extract features, such as edges and curves.
- (1*1 + 1* 1 + 1*1 = 3)

① คูณแบบคณิต

การลดขนาด

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 |

| | | |
|---|---|---|
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

Filter

| | | |
|---|---|---|
| 2 | 3 | 0 |
| 0 | 1 | 3 |
| 1 | 0 | 2 |

Feature map

output

② สร้าง feature จาก pixel × Filter

# Example

Model Generate (เก็บไว้)

**Understanding learned CNN features through Filter Decoding with Substitution**

Ivet Rafegas
Computer Vision Center
C. Sc. Dpt. UAB. Bellaterra (Barcelona)
irafegas@cvc.uab.cat

Maria Vanrell
Computer Vision Center
C. Sc. Dpt. UAB. Bellaterra (Barcelona)
maria@cvc.uab.cat

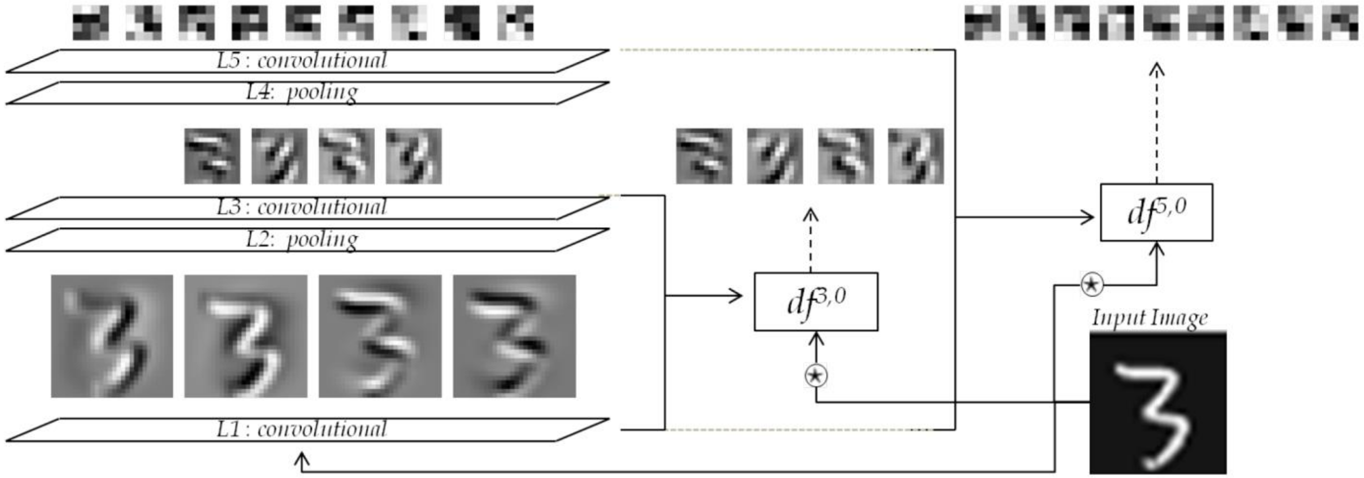


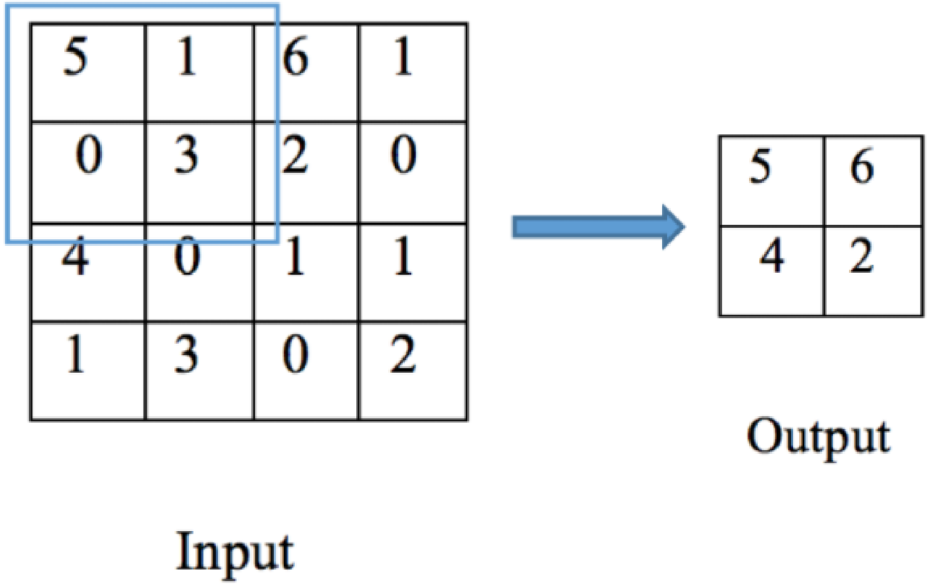

Figure 3: Comparison between image responses obtained through the network (left) and by convolving with decoded filters (right). Responses at different layers show similar activation for both cases, that validate the proposed approach. Obviously the error increase a with the increase in number of encoded layers.

0

# Pooling

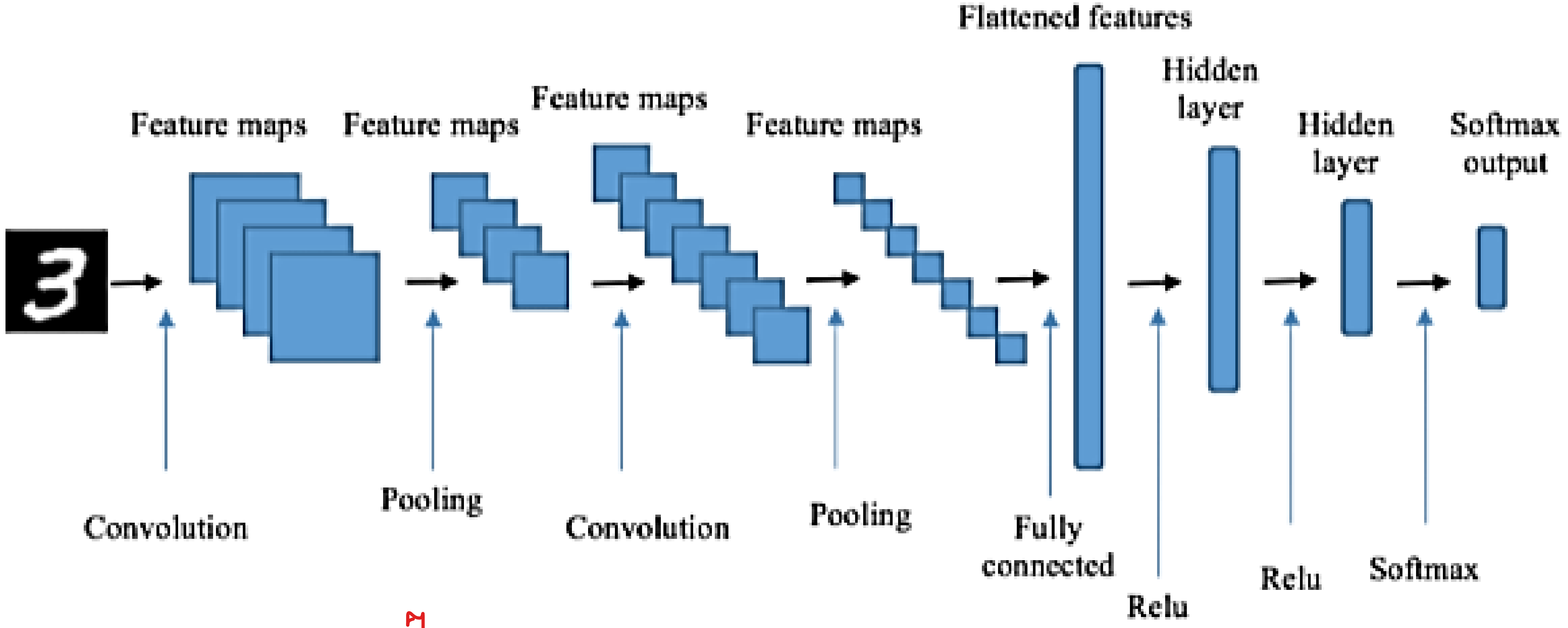
FHD 1920 x 1020 ≈ 2 M pixels

- Called down sampling layer
- Typical pooling methods include
  - max pooling
  - mean pooling
- Pooling helps us reducing overfitting with lower dimensional output.
- Example, we apply a 2*2 max pooling filter on a 4*4 feature map and output a 2*2 one

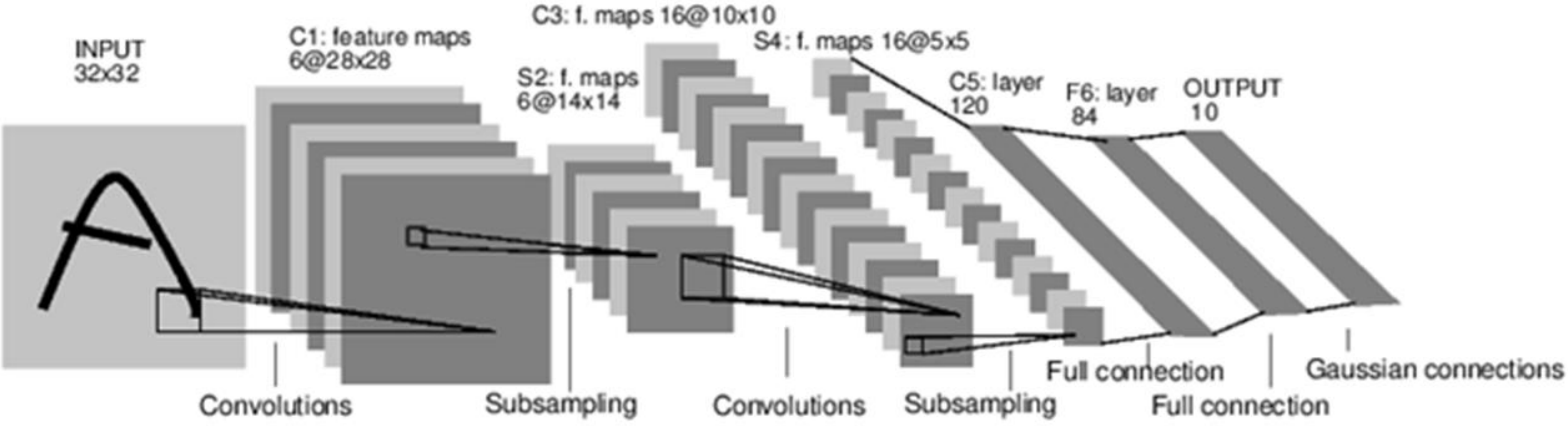



5 is the maximum value in the window

# Example Structure CNN model

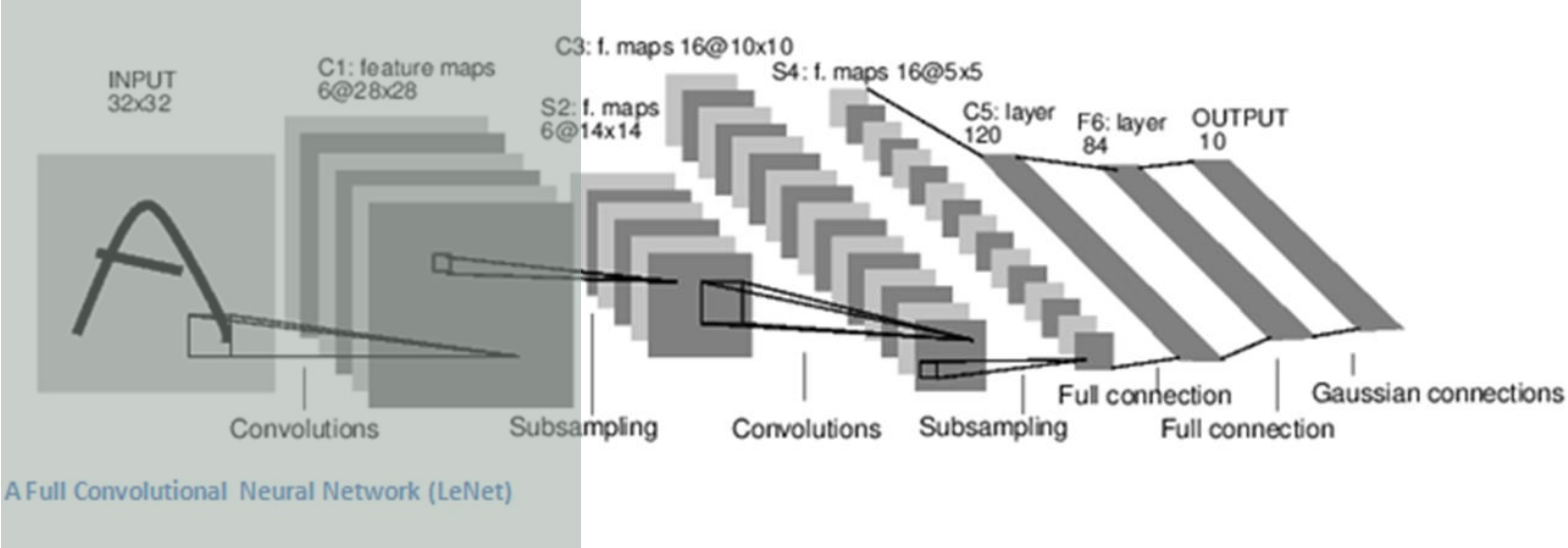


# CNN process




A Full Convolutional Neural Network (LeNet)

# Convolutional Layer




- The convolutional layer converts an input image to matrix form
- Kernel filter is applied during the conversion process.
- Method of stride and padding are applied

# Stride 2 1-D, 2-D, and 3-D Winograd for Convolutional Neural Networks

Juan Yepez and Seok-Bum Ko, *Senior Member, IEEE*

***Abstract*— Convolutional neural networks (CNNs) have been widely adopted for computer vision applications. CNNs require many multiplications, making their use expensive in terms of both computational complexity and hardware. An effective method to mitigate the number of required multiplications is via the Winograd algorithm. Previous implementations of CNNs based on Winograd use the 2-D algorithm $F(2 \times 2, 3 \times 3)$, which reduces computational complexity by a factor of 2.25 over regular convolution. However, current Winograd implementations only apply when using a stride (shift displacement of a kernel over an input) of 1. In this article, we presented a novel method to apply the Winograd algorithm to a stride of 2. This method is valid for one, two, or three dimensions. We also introduced new Winograd versions compatible with a kernel of size 3, 5, and 7. The algorithms were successfully implemented on an NVIDIA K20c GPU. Compared to regular convolutions, the implementations for stride 2 are 1.44 times faster for a $3 \times 3$ kernel, 2.04× faster for a $5 \times 5$ kernel, 2.42× faster for a $7 \times 7$ kernel, and 1.73× faster for a $3 \times 3 \times 3$ kernel. Additionally, a CNN accelerator using a novel processing element (PE) performs two 2-D Winograd stride 1, or one 2-D Winograd stride 2, and operations per clock cycle was implemented on an Intel Arria-10 field-programmable gate array (FPGA). We accelerated the original and our proposed modified**

The Winograd minimal filtering algorithms (WMFAs), capable of being used for any stride [11], take advantage of overlapping computations between adjacent windows [12] to reduce the number of multiplications required for convolution, trading multiplication for addition. Given that the hardware required for multiplication is complex and large compared to that of a simple adder, the multiplication–addition tradeoff proposed by Winograd is desirable.

Modern CNN architectures replace pooling layers with strided convolutions for downsampling [13], where stride is defined as the element-wise shift displacement of a kernel over an input along a particular axis [14]. Convolutional layers learn feature properties during training; conversely, pooling is a fixed downsampling operation, and pooling layers have no trainable weights. A convolutional layer with stride $>1$ is advantageous in that it has trainable parameters and downsamples.

Recent architectures (e.g., the MobileNet family) use increasingly more layers with stride $>1$. Therefore, it is impor-

# Convolutional Layer: Padding

- Padding is increasement pixels to an image processed by the kernel of CNNs
- Researcher applies padding for preserving size of input image = same size.

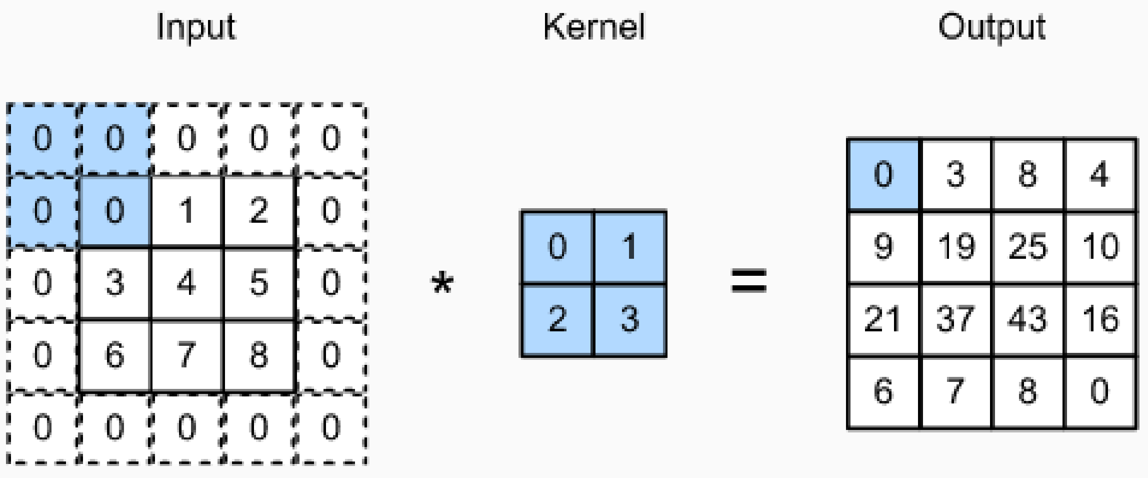
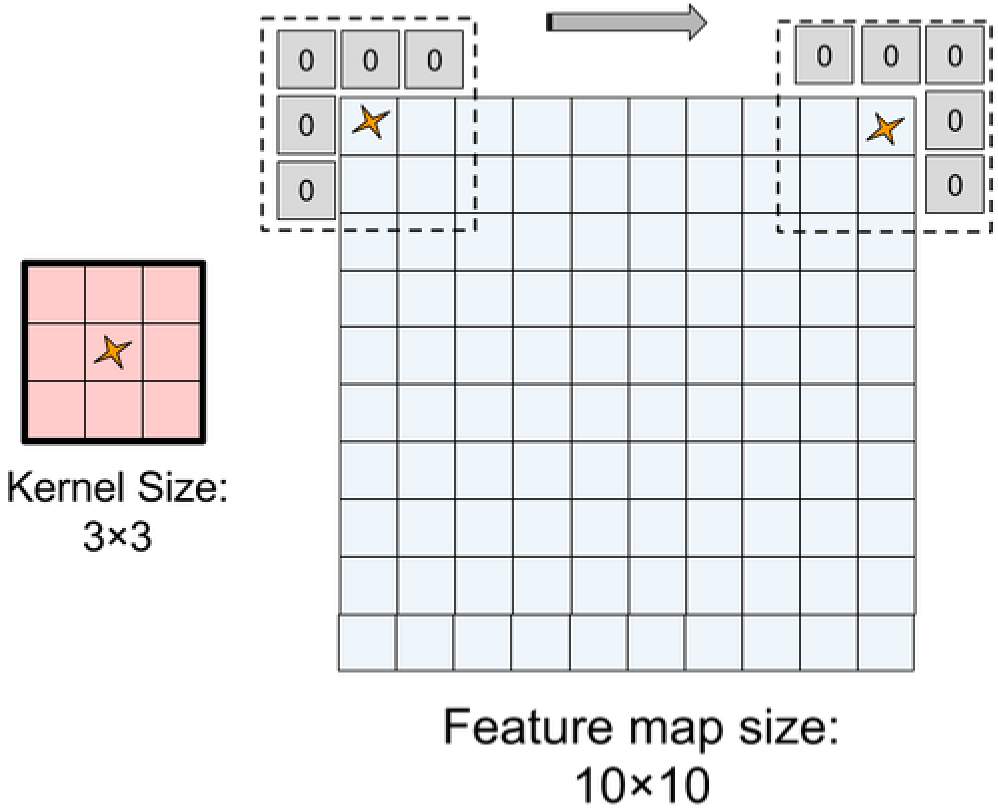



Fig. 6.3.1 Two-dimensional cross-correlation with padding.



# Activity 10.1

```
rm(list=ls())

i = c(4,9,2,5,8,3,
      5,6,2,4,0,3,
      2,4,5,4,5,2,
      5,6,5,4,7,8,
      5,7,7,9,2,1,
      5,8,5,3,8,4)

h = c(1,0,-1,
      1,0,-1,
      1,0,-1)

img = matrix(i,nrow = 6, byrow=TRUE)
img

#KERNEL FILTER
feamatrix = matrix(h,nrow=3,byrow=TRUE)
feamatrix
```

```
#APPLY FEATURE TO IMAGE INPUT
mulresult = rep(NA,1)
c = 1
print(img)
for (j in 1:4)
{
    m = j+2
    for (i in 1:4)
    {
        k =i+2
        #str = sprintf("%d:%d / %d:%d",j,m,i,k)
        #print(str)
        print(img[j:m,i:k])
        partimg = img[j:m,i:k]
        mulfilter = partimg * feamatrix
        mulresult[c] = sum(mulfilter)
        print(mulfilter)
        c = c+1
    }
    print("======")
}
mulresult

matfilter = matrix(mulresult,nrow=4,byrow=TRUE)
matfilter
```

```
> img
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    4    9    2    5    8    3
[2,]    5    6    2    4    0    3
[3,]    2    4    5    4    5    2
[4,]    5    6    5    4    7    8
[5,]    5    7    7    9    2    1
[6,]    5    8    5    3    8    4
```

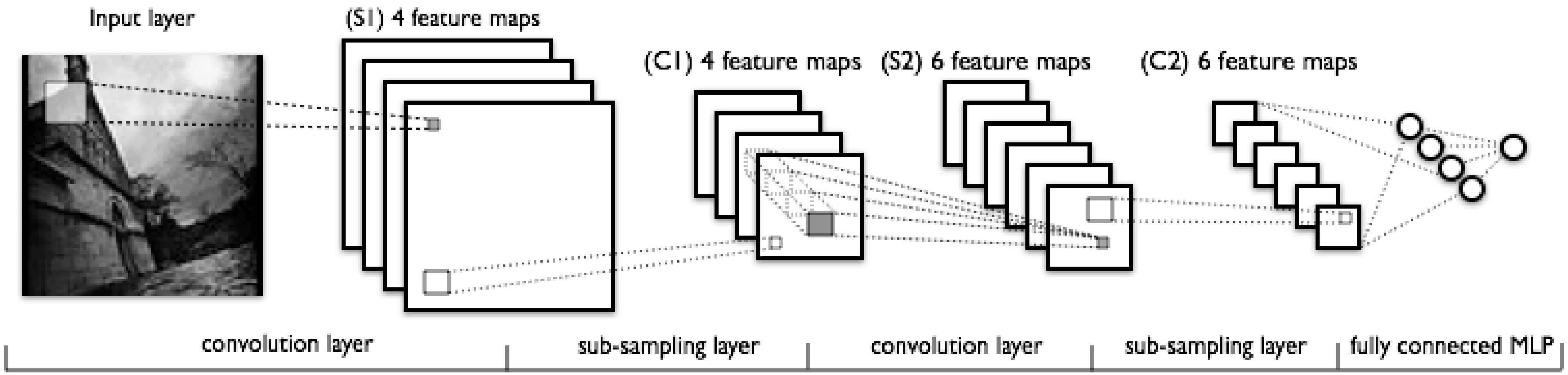
```
> feamatrix
     [,1] [,2] [,3]
[1,]    1    0   -1
[2,]    1    0   -1
[3,]    1    0   -1
```

```
> matfilter
     [,1] [,2] [,3] [,4]
[1,]    2    6   -4    5
[2,]    0    4    0   -1
[3,]   -5    0    3    6
[4,]   -2    5    0    3
```

# RESEARCH IN CNN

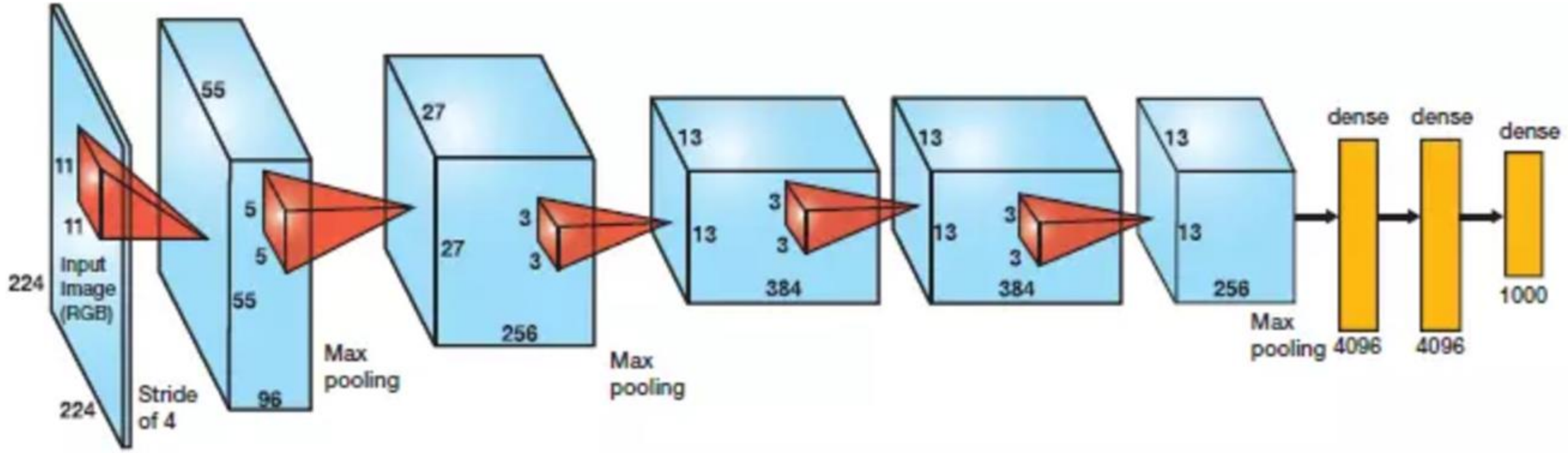
# LeNet

- The classical CNN architecture
- http://deeplearning.net/tutorial/lenet.html
- You may use TensorFlow "layers" or "Keras" for the implementation



# AlexNet

- AlexNet
  - consists of **5 Convolutional Layers** and **3 Fully Connected Layers**
  - Overlapping Max Pooling
  - ReLU Nonlinearity
  - Reducing Overfitting
  - Data Augmentation
  - Dropout

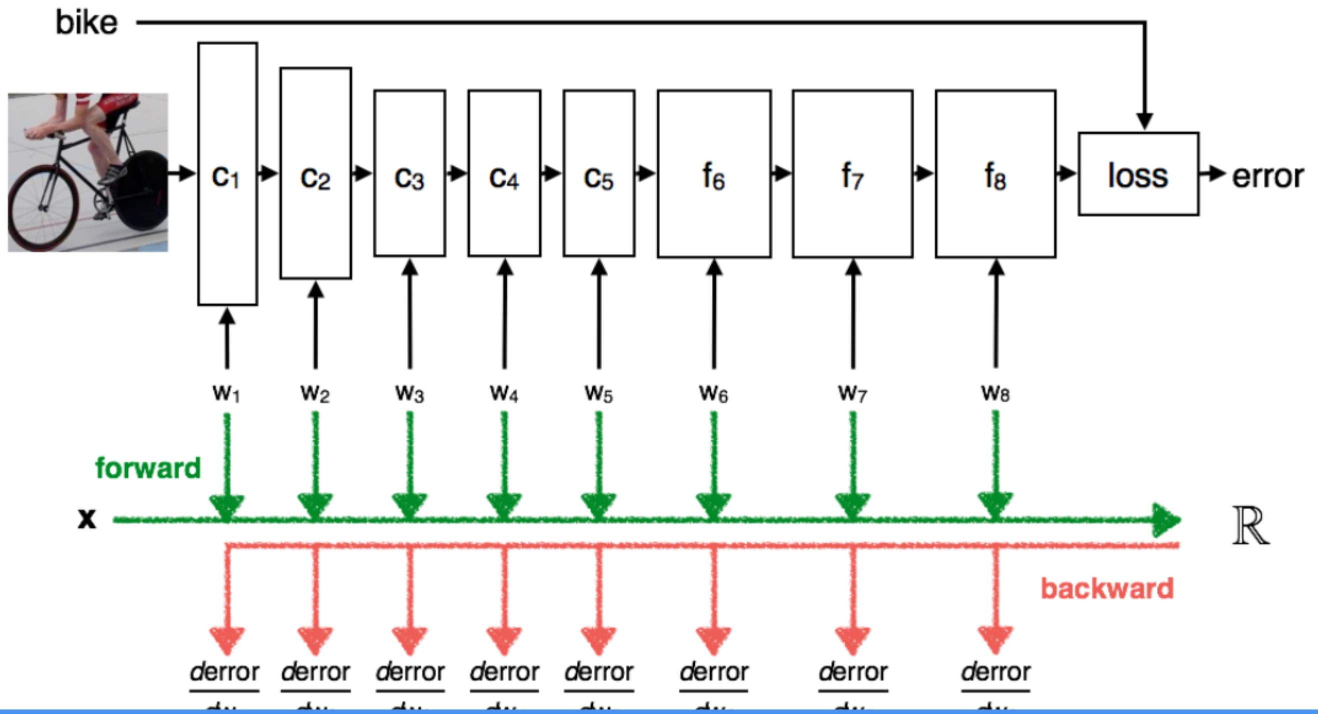


# VGGNet

- Forward pass: Convolve $C_i$ with $W_i$ (kernel) to generate the next layer $C_{i+1}$
- Backpropagation: Use gradient decent method [d(error)/d$W_i$] to update $W_i$

ไป - กลับ

This is an Oxford Visual Geometry Group computer vision practical, authored by Andrea Vedaldi and Zisserman (Release 2017a).



# Subsampling (subs)

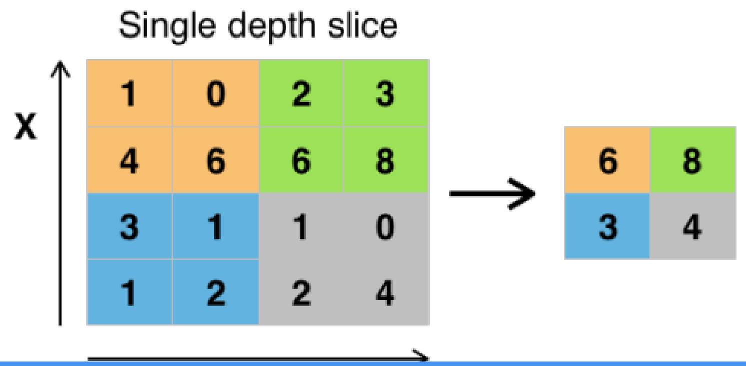
คล้าย pooling

ไม่มี over lab

- Subsampling allows features to be flexibly positioned around a specific area, example:
  - Subsample an output (a matrix of 2x2)
  - Sample s=(

| | |
|---|---|
| a | b |
| c | d |

- It can be achieved by two methods:
  - Take average : s=(a+b+c+d)/4, or
  - Max pooling : s= max(a,b,c,d)




Max pooling

# DEVELOPMENT CNN IN R

# CNN libraries in R

TensorFlow RStudio

- https://tensorflow.rstudio.com/tutorials/advanced/images/cnn/
- library(keras)
  - Keras is written in Python (First)
  - Keras for R (Start 2017)
  - Python speed over R 15% (https://towardsdatascience.com/r-vs-python-image-classification-with-keras-1fa99a8fef9b)
- Keras has many advantages:
  - Easy to build complex models in a few lines of code
  - Code recycling: by  CNN toolkits to Tensorflow or vice versa
  - Seamless use of GPU/CPU

(CNN)

This tutorial demonstrates training a simple Convolutional Neural Network (CNN) to classify CIFAR images. Because this tutorial uses the Keras Sequential API, creating and training our model will take just a few lines of code.

## Setup

```
library(tensorflow)
library(keras)
```

## Download and prepare the CIFAR10 dataset

The CIFAR10 dataset contains 60,000 color images in 10 classes, with 6,000 images in each class. The dataset is divided into 50,000 training images and 10,000 testing images. The classes are mutually exclusive and there is no overlap between them.

```
cifar <- dataset_cifar10()
```

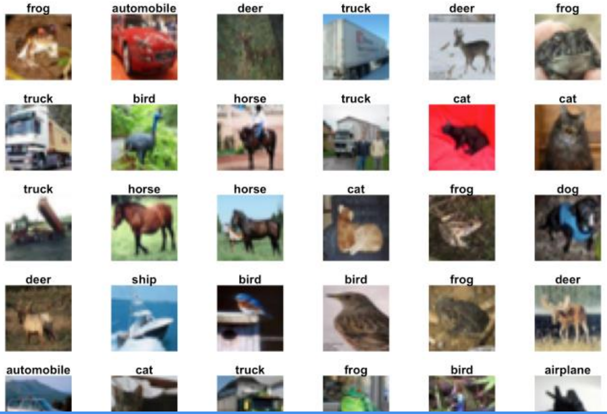
## Verify the data

To verify that the dataset looks correct, let's plot the first 25 images from the training set and display the class name below each image.

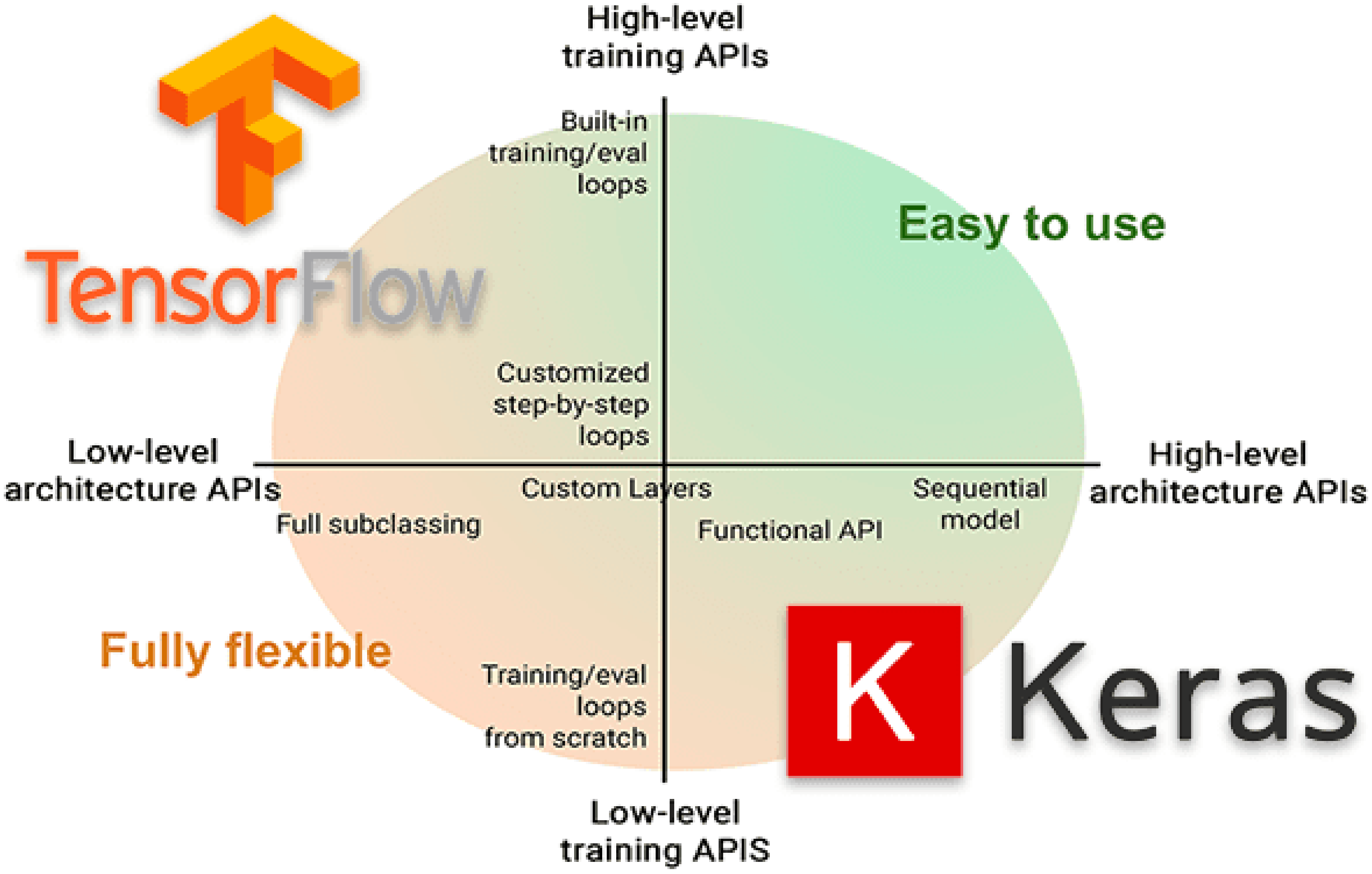
```
class_names <- c('airplane', 'automobile', 'bird', 'cat', 'deer',
                 'dog', 'frog', 'horse', 'ship', 'truck')

index <- 1:30

par(mfcol = c(5,6), mar = rep(1, 4), oma = rep(0.2, 4))
cifar$train$x[index,,,] %>%
  purrr::array_tree(1) %>%
  purrr::set_names(class_names[cifar$train$y[index] + 1]) %>%
  purrr::map(as.raster, max = 255) %>%
  purrr::iwalk(~{plot(.x); title(.y)})
```



# Summary


TensorFlow
High-level
training APIs
Built-in
training/eval
loops
Easy to use
Customized
step-by-step
loops
Low-level
architecture APIs
Custom Layers
Full subclassing
Functional API
Sequential
model
High-level
architecture APIs
Fully flexible
Training/eval
loops
from scratch
K
Keras
Low-level
training APIS

# Introduction

## Keras

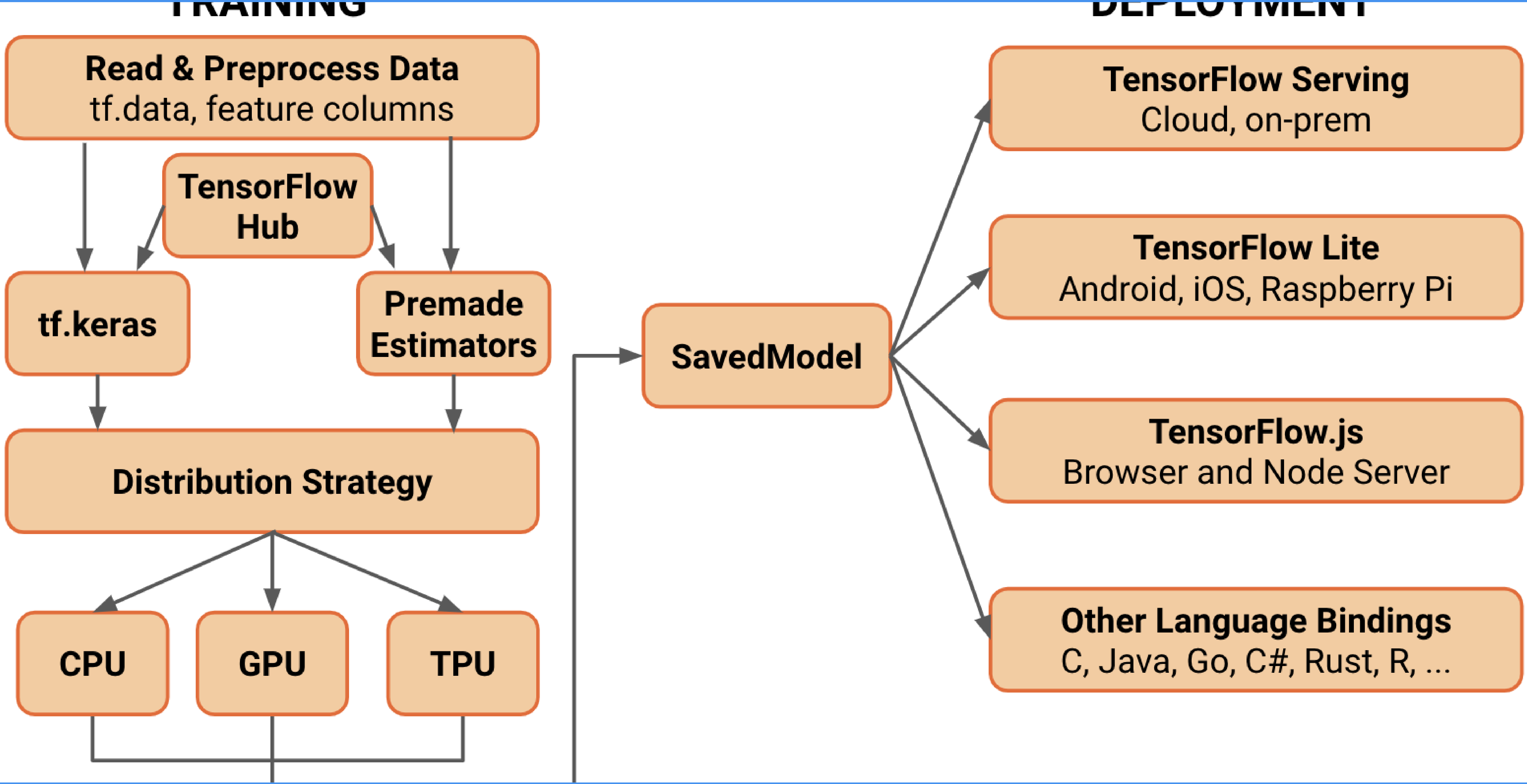
Keras is an open source neural network library written in Python. It is capable of running on top of TensorFlow. It is designed to enable fast experimentation with **deep neural networks**.

## TensorFlow

TensorFlow is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library that is used for **machine learning** applications like neural networks.

## PyTorch

PyTorch is an open source **machine learning** library for Python, based on Torch. It is used for applications such as **natural language processing** and was developed by Facebook's AI


Read & Preprocess Data
tf.data, feature columns
TensorFlow Hub
tf.keras
Premade Estimators
Distribution Strategy
CPU
GPU
TPU
SavedModel
TensorFlow Serving
Cloud, on-prem
TensorFlow Lite
Android, iOS, Raspberry Pi
TensorFlow.js
Browser and Node Server
Other Language Bindings
C, Java, Go, C#, Rust, R, ...