Neural networks and deep learning

Convolutional Neural Network

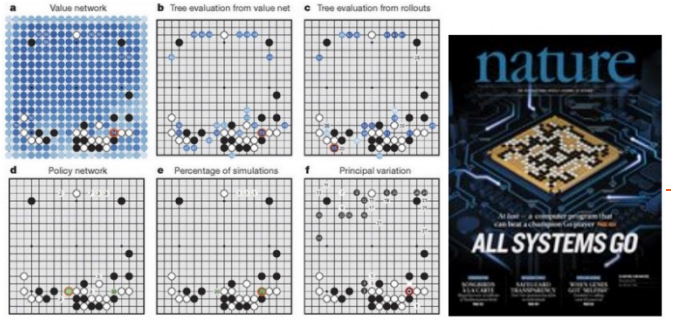
Kun Suo

Computer Science, Kennesaw State University

https://kevinsuo.github.io/

Application of convolution

AlphaGo

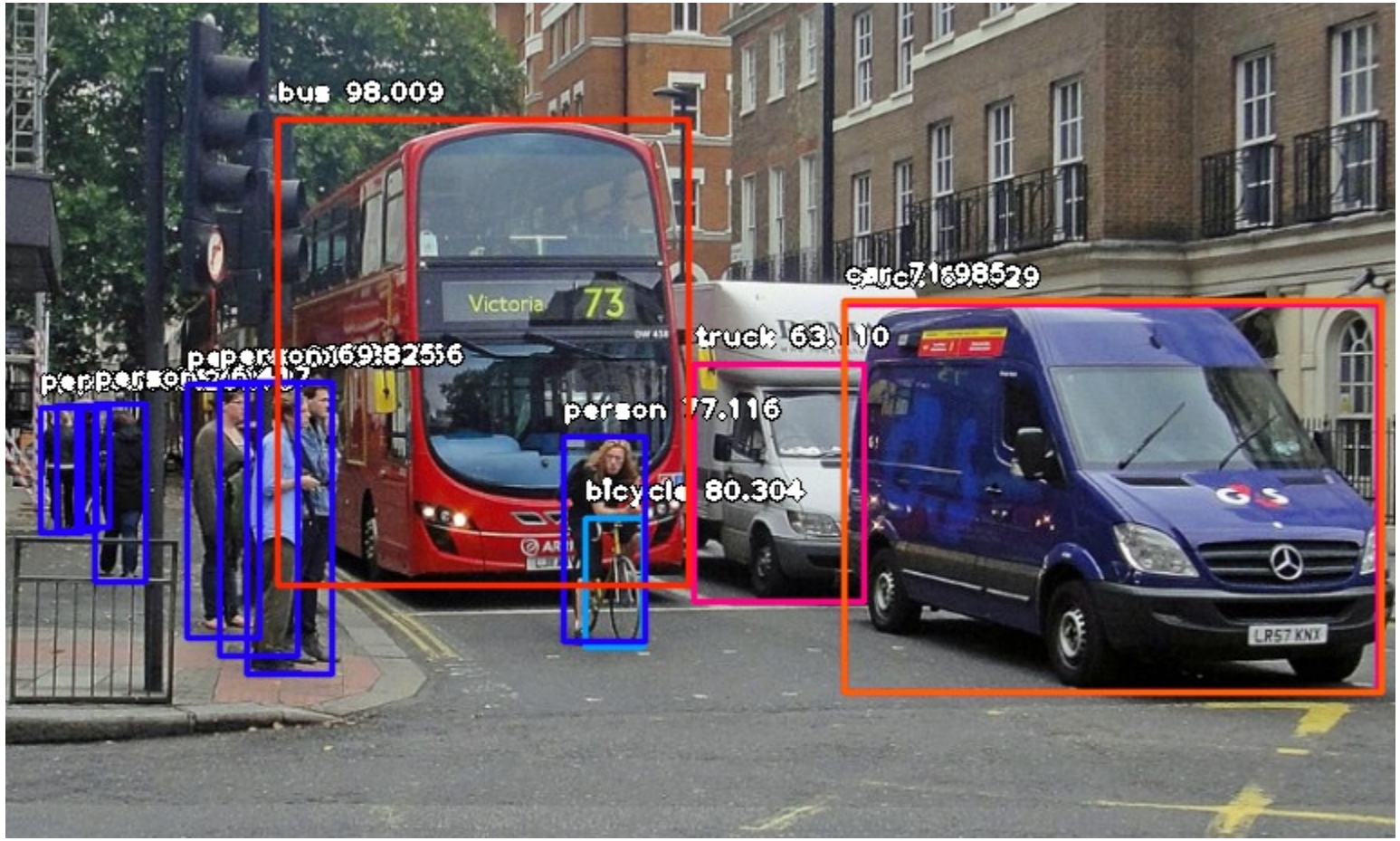


The input to the policy network is a 19 × 19 × 48 image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a 23 × 23 image, then convolves *k* filters of kernel size 5 × 5 with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a 21 × 21 image, then convolves *k* filters of kernel size 3 × 3 with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size 1 × 1 with stride 1, with a different bias for each position, and applies a softmax function. The match version of AlphaGo used *k* = 192 filters; Fig. 2b and Extended Data Table 3 additionally show the results of training with *k* = 128, 256 and 384 filters.

policy network:
[19x19x48] Input
CONV1: 192 5x5 filters , stride 1, pad 2 => [19x19x192]
CONV2..12: 192 3x3 filters, stride 1, pad 1 => [19x19x192]
CONV: 1 1x1 filter, stride 1, pad 0 => [19x19] *(probability map of promising moves)*

- Distributed System： 1202 CPU and 176 GPU
- Single machine： 48 CPU and 8 GPU
- Speed of each move： 3 ms to 2 us

Object Detection



Mask RCNN

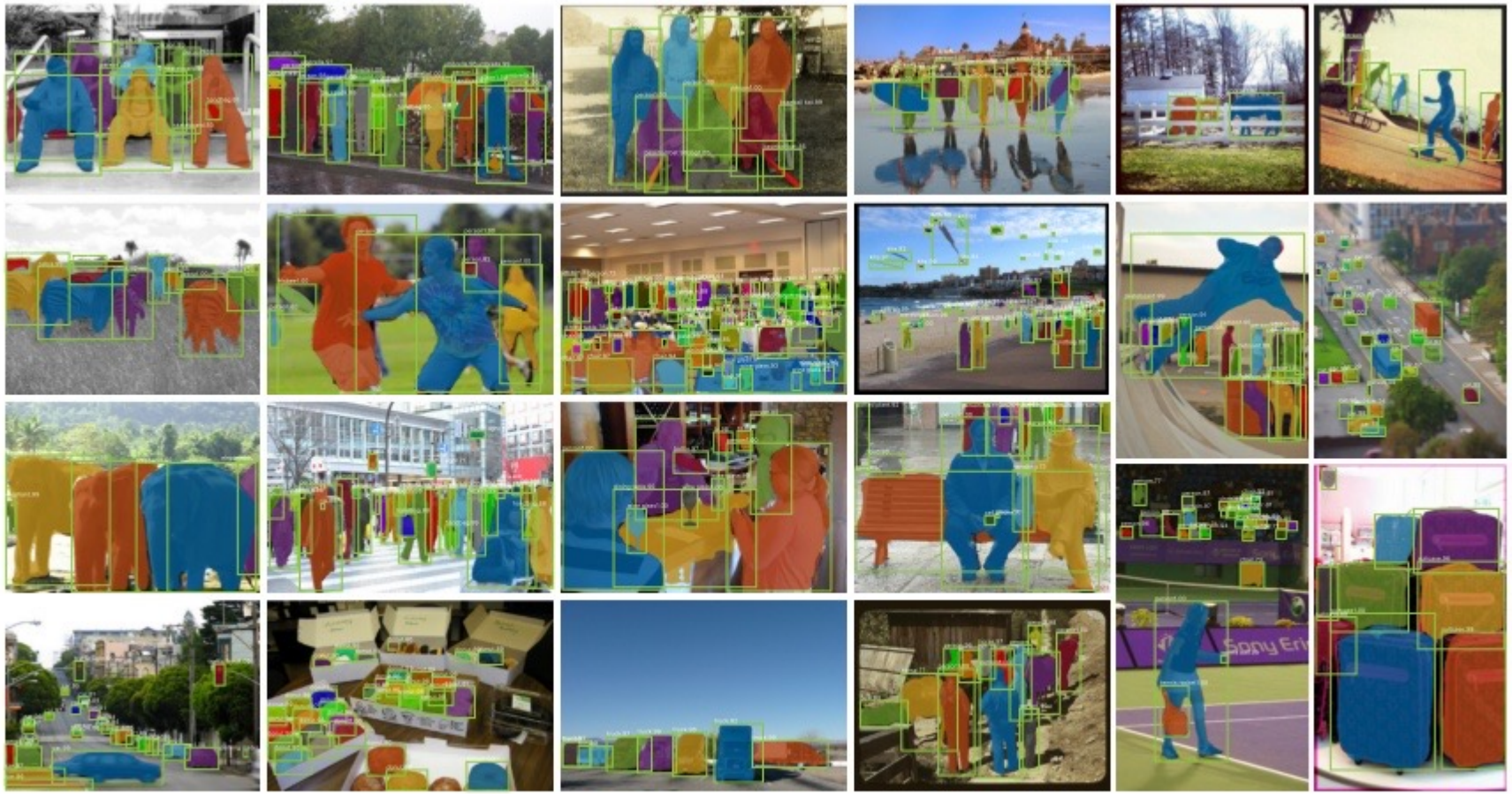
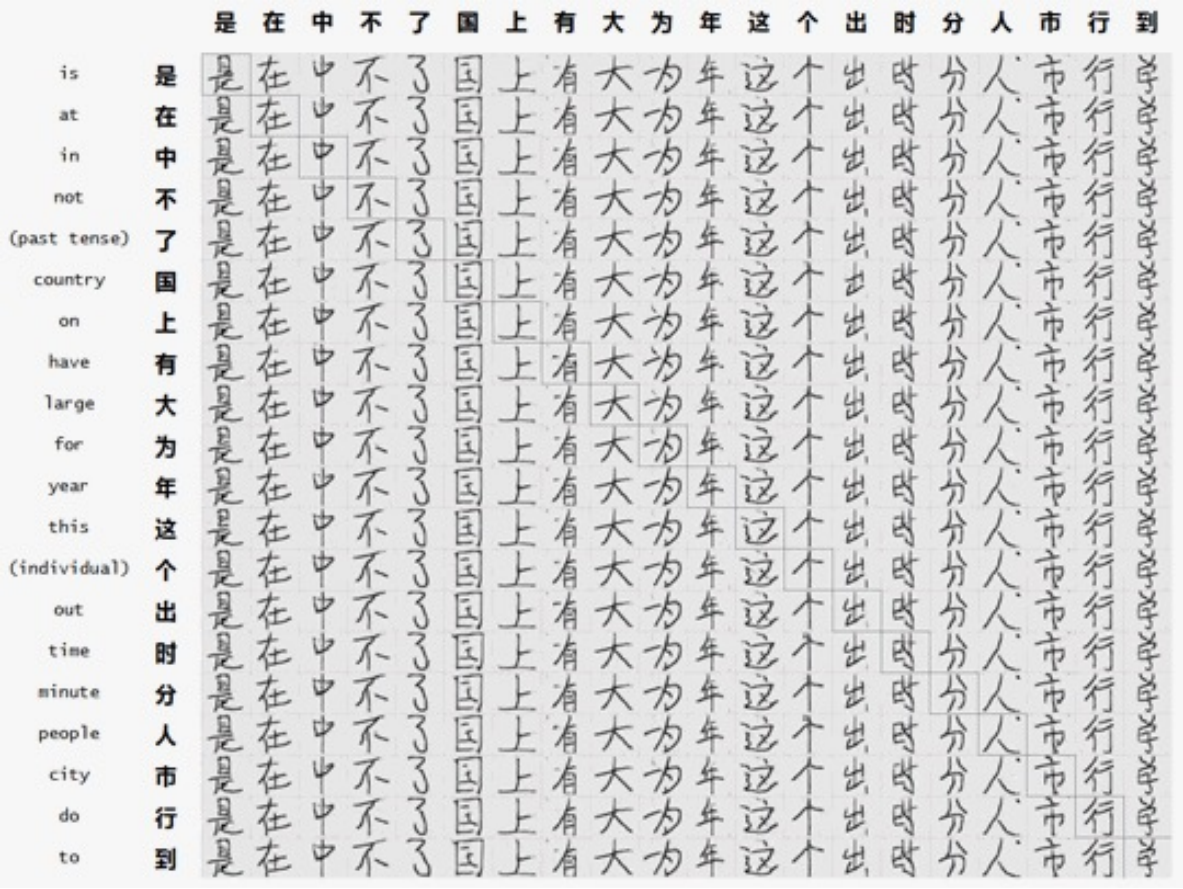


Figure 4. More results of **Mask R-CNN** on COCO test images, using ResNet-101-FPN and running at 5 fps, with 35.7 mask AP (Table 1).

OCR

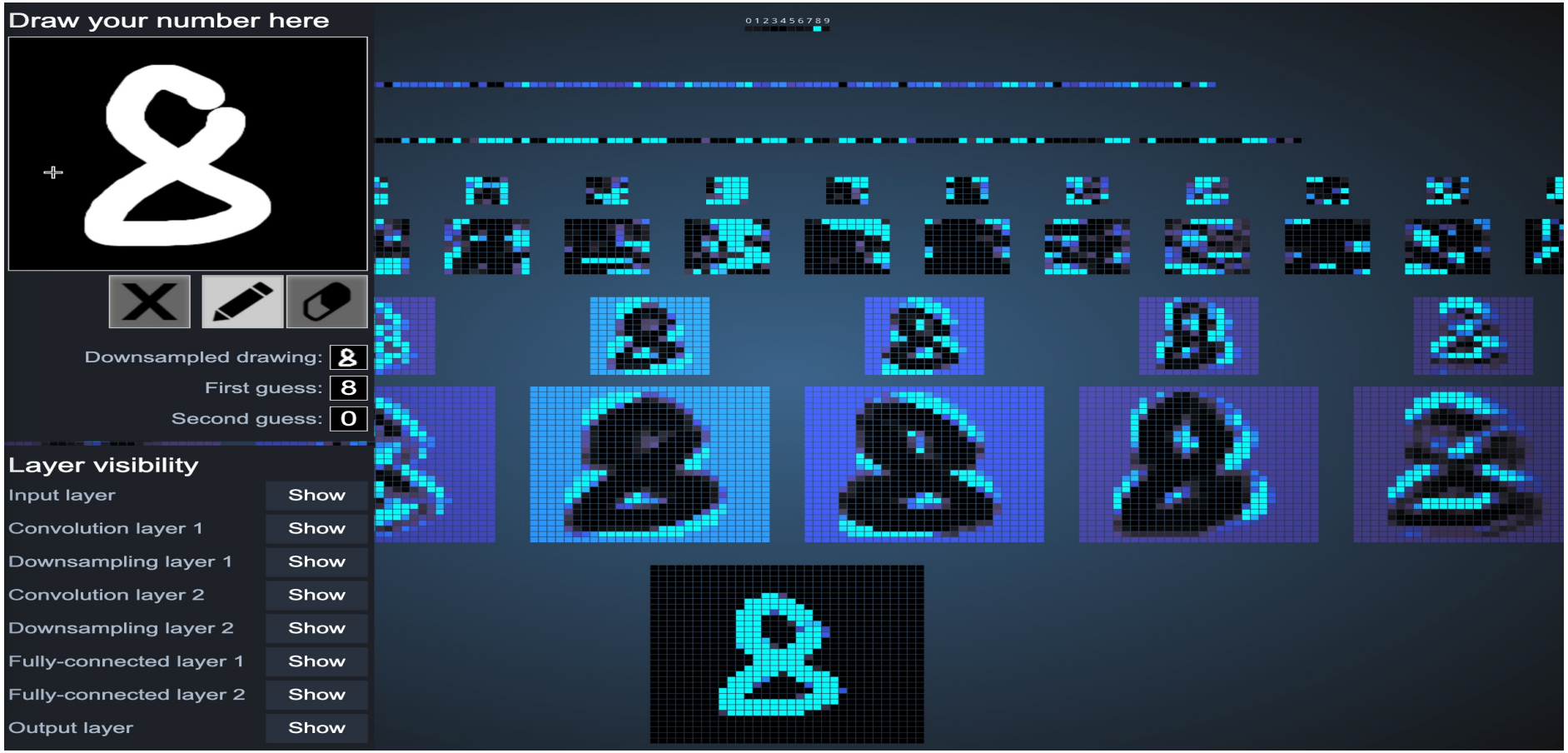

THE WORLD FAMOUS
BRIXHAM
FISH MARKET
BARBER
FRENCH WIGS & HAIRPIECES
THE
ROYAL
SCOTSMAN
SWINGBRIDGE
NESTON
BREWING
COMPANY
THIS WAY
HONOR ★ SERVICE ★ INTEGRITY
AMERICAN
BUILT
ARMS COMPANY
GARDA
BMW
CLUB
MULTIPLE SCLEROSIS
CHARITY RUN
MIZEN HEAD
MALIN HEAD
MAY 12TH 200
HOTEL
GREEN
NANTUCKET

Image generation



Number recognition

https://www.cs.ryerson.ca/~aharley/vis/conv/flat.html



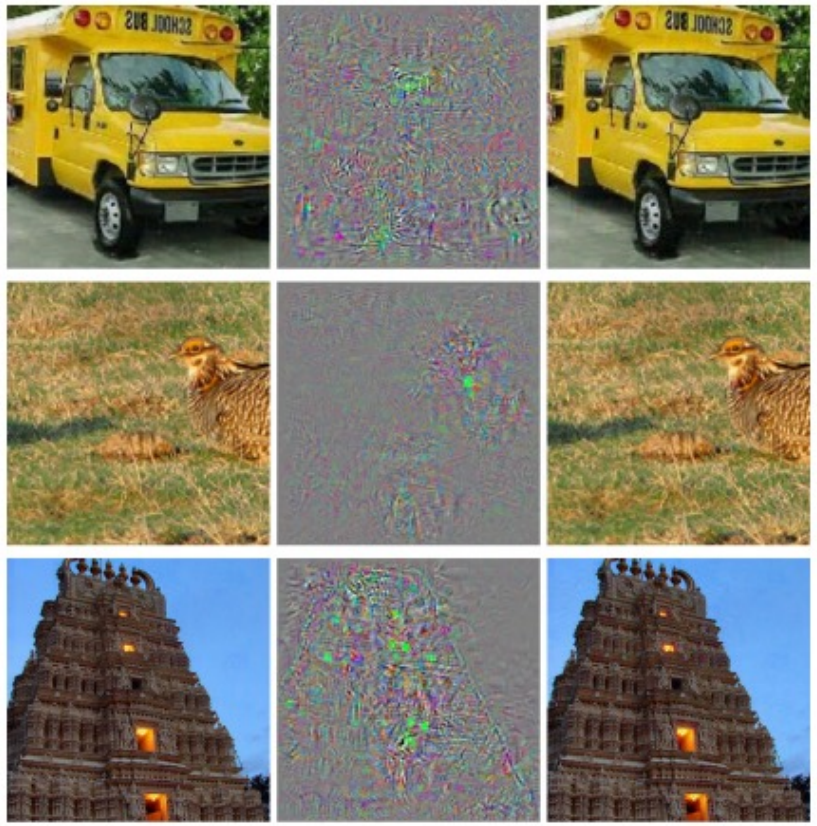
https://deepdreamgenerator.com/

Deep Dream

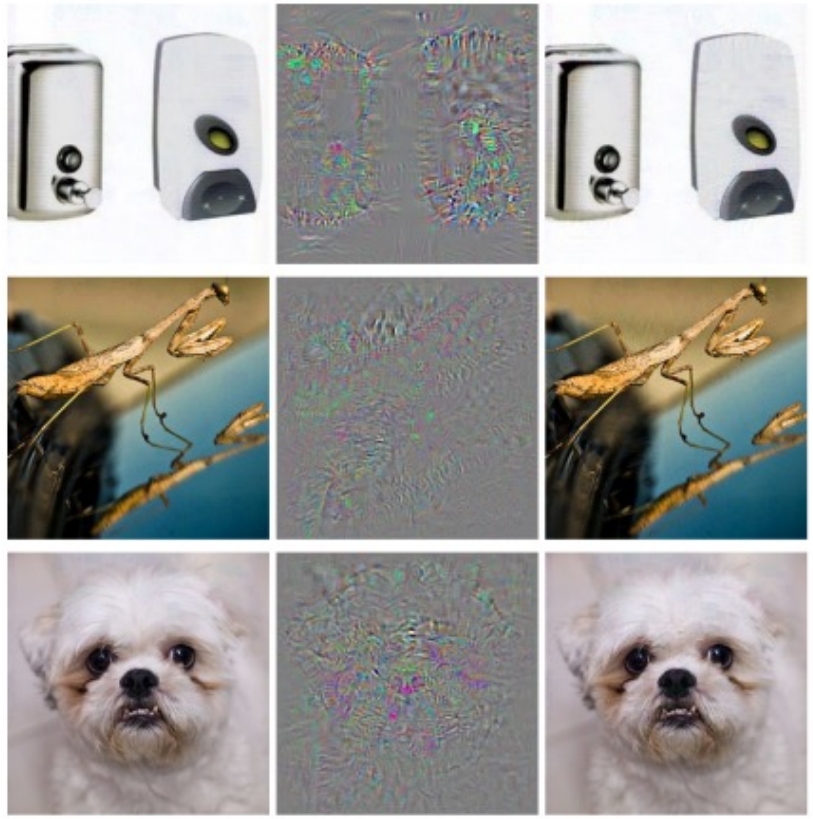
https://www.instapainting.com/ai-painter

Painting style migration

Adversarial examples



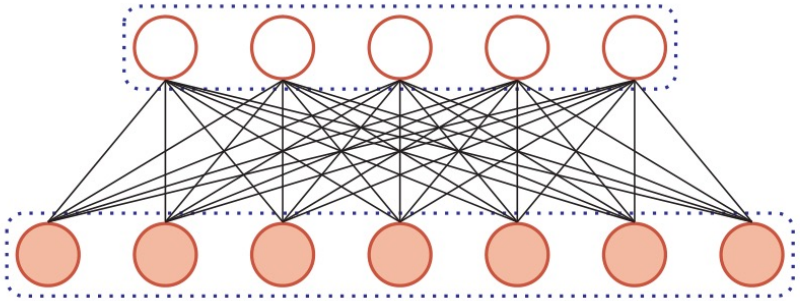
(a)



(b)

Fully connected feedforward neural network

- The weight matrix has many parameters



M

N

Num of weights = M*N

Fully connected feedforward neural network

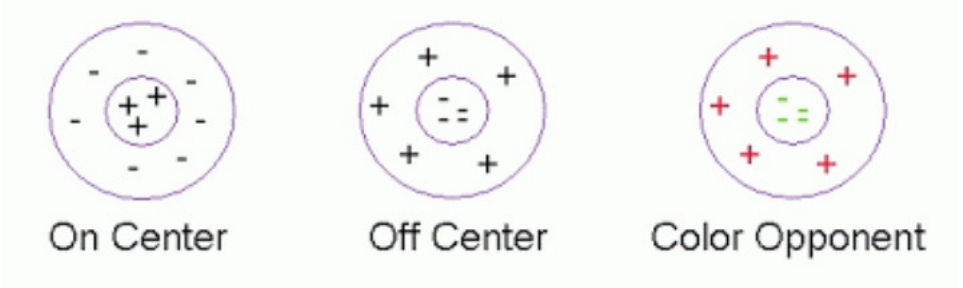
- Local invariance
  - Objects in natural images have local invariance characteristics
    - Operations such as scaling, translation, and rotation do not affect its semantic information.
  - The fully connected feedforward network is difficult to extract these local invariant features

Convolutional Neural Networks

Convolutional Neural Networks (CNN)

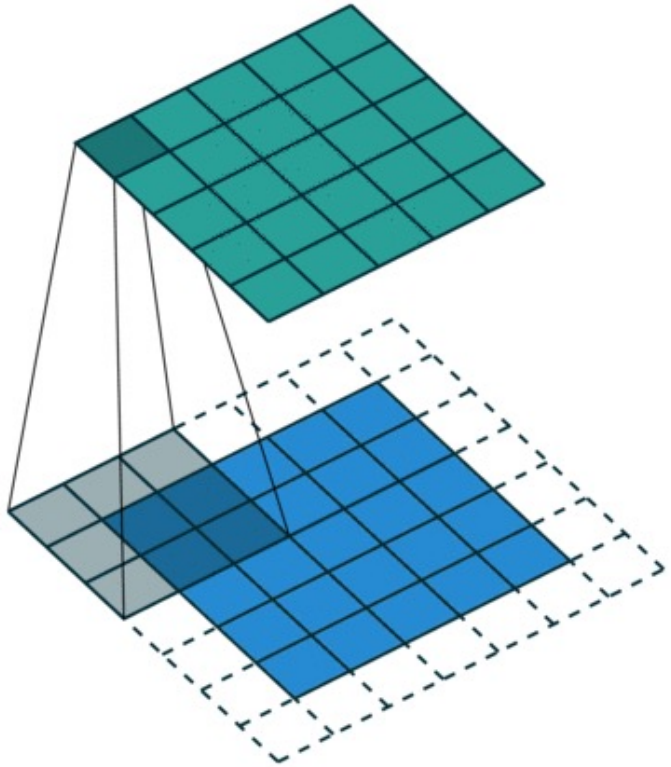
- A feedforward neural network
- Proposed by the biological mechanism of the Receptive Field
  - In the visual nervous system, the receptive field of a neuron refers to a specific area on the retina, and only stimulation in this area can activate the neuron.

Retinal receptive field



Convolutional Neural Networks

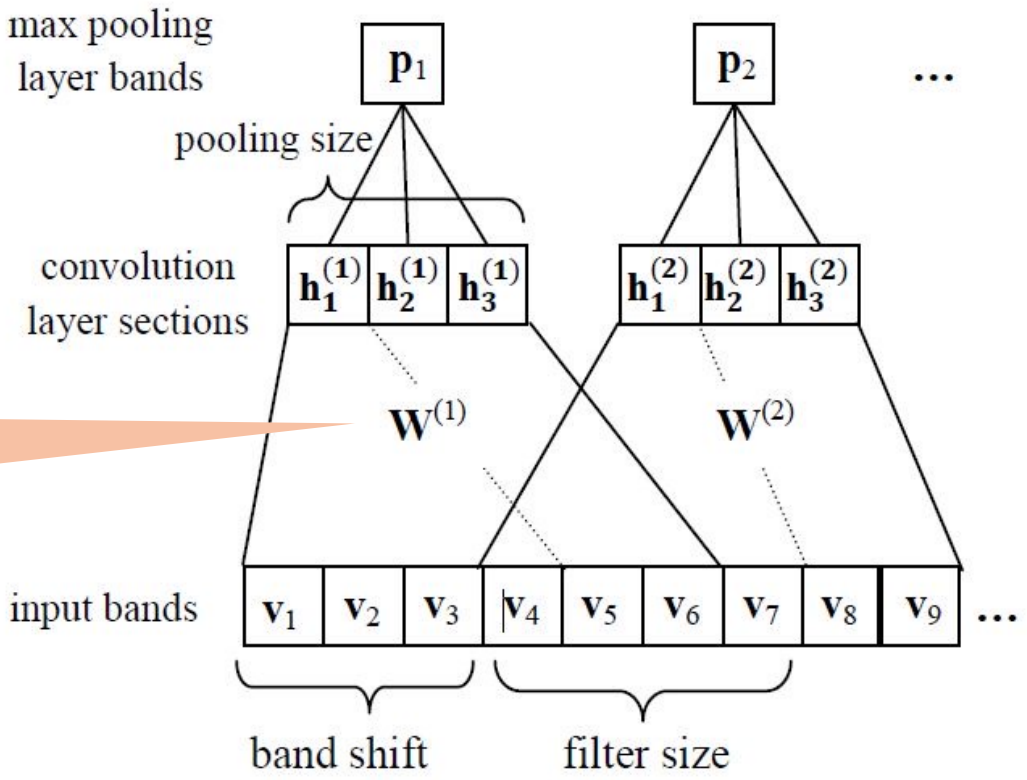
- Convolutional neural networks have three structural characteristics:
  - Local connection
  - Weight sharing
  - Sub-sampling in space or time



Convolutional Neural Networks

- Convolutional neural networks have three structural characteristics:
  - Local connection
  - Weight sharing
  - Sub-sampling in space or time

Each W is only connected to a part of the input



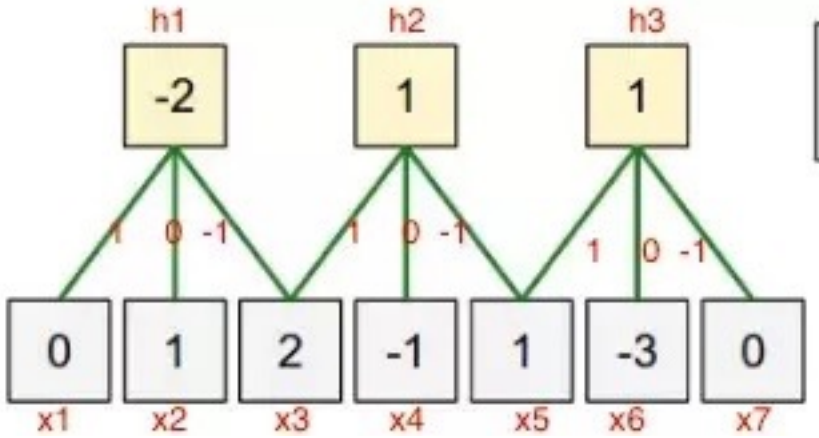
Convolutional Neural Networks

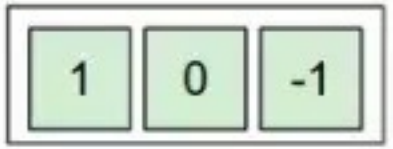
- Convolutional neural networks have three structural characteristics:
  - Local connection
  - Weight sharing
  - Sub-sampling in space or time

Input $x = [x_1, x_2, x_3, x_4, x_5, x_6, x_7]$

Hidden layer $h = [h_1, h_2, h_3]$

weight $w = [w_1, w_2, w_3] = [1, 0, -1]$

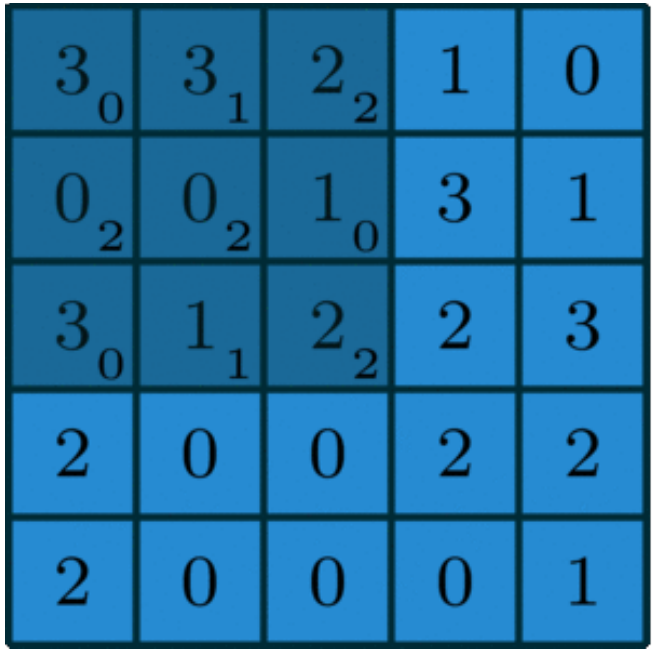




The weight value w is shared by h_1, h_2, h_3

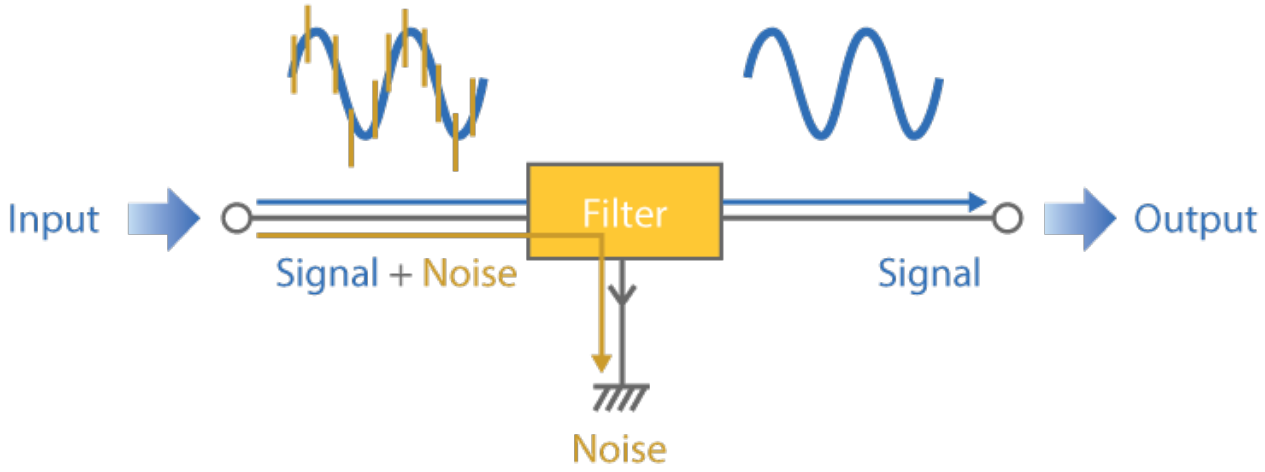
Convolutional Neural Networks

- Convolutional neural networks have three structural characteristics:
  - Local connection
  - Weight sharing
  - Sub-sampling in space or time



Convolution

- Convolution is often used in signal processing to calculate the delay accumulation of the signal.

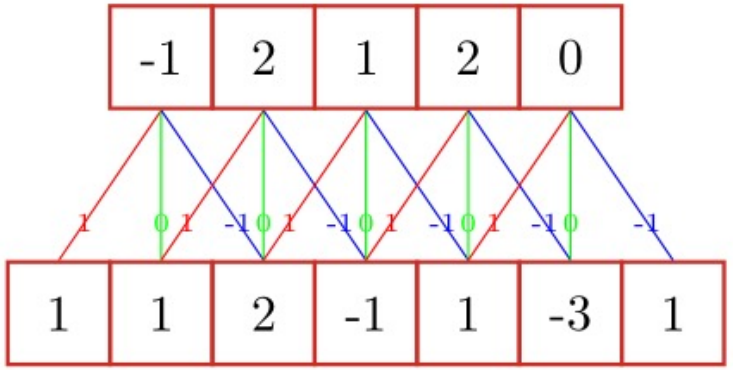


Convolution

- Given an input signal sequence x and filter w, the output of convolution is:

$$y_t = \sum_{k=1}^{K} w_k x_{t-k+1}$$

Filter: [1,0,-1]

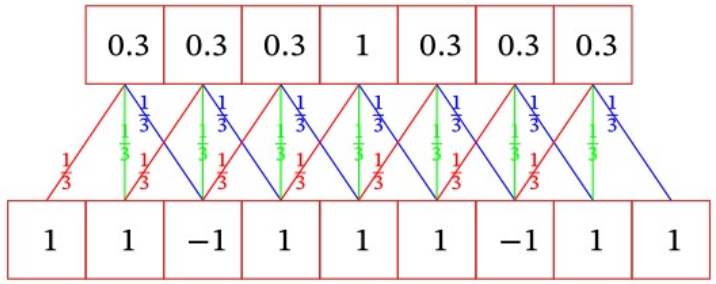


Layer i+1

Layer i

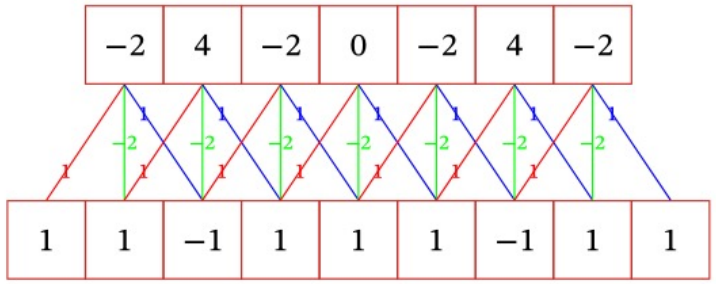
Convolution

- Different filters to extract different features in the signal sequence



Filter [1/3, 1/3, 1/3]

Low frequency information

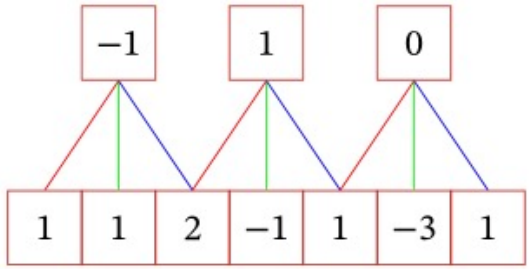


Filter [1, -2, 1]

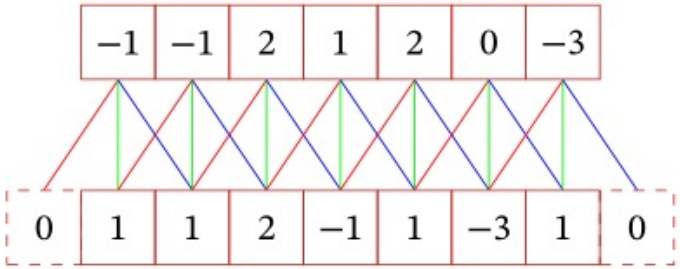
High frequency information

Convolution expansion

- Introduce the sliding step (stride) S and zero padding P of the filter



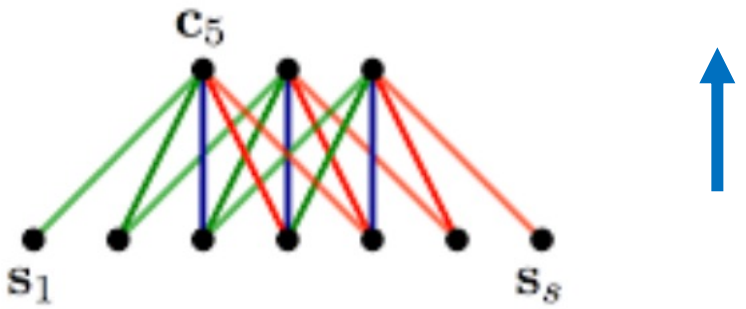
Step S = 2

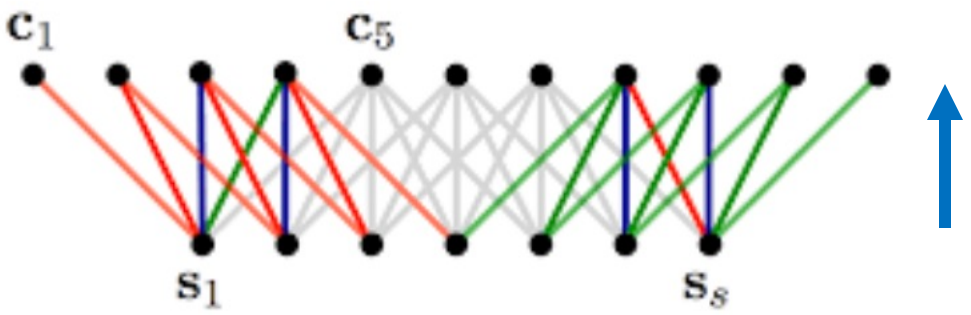


Zero padding P = 1

Convolution type

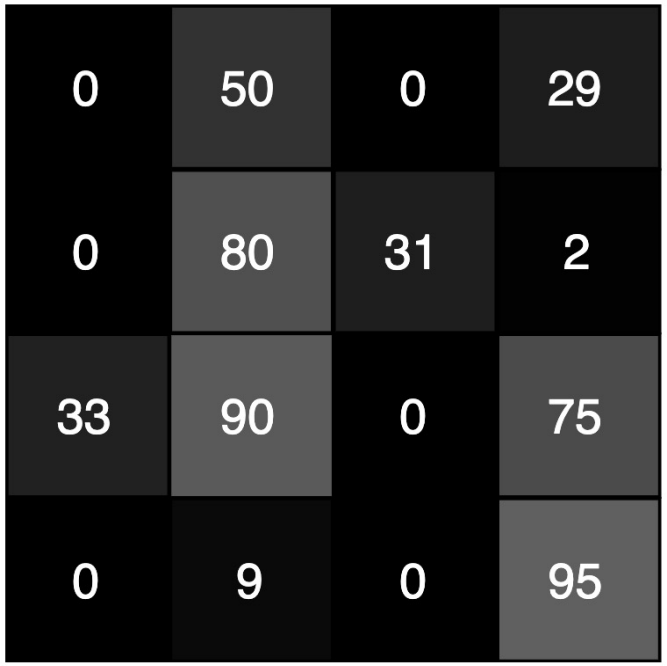
- The results of convolution can be divided into three categories according to the output length:
  - Narrow convolution: step size T = 1, no zero padding at both ends P = 0, the output length after convolution is $M - K + 1$
  - Wide convolution: step length T = 1, zero padding at both ends $P = K - 1$, output length after convolution $M + K - 1$
  - Constant-width convolution: step length T = 1, zero padding at both ends $P = (K - 1)/2$, output length after convolution M





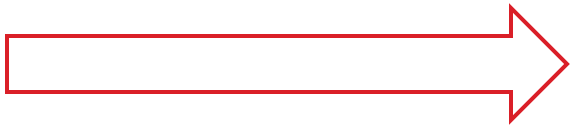
Two-dimensional convolution

- Consider this tiny 4x4 grayscale image and this 3x3 filter:



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

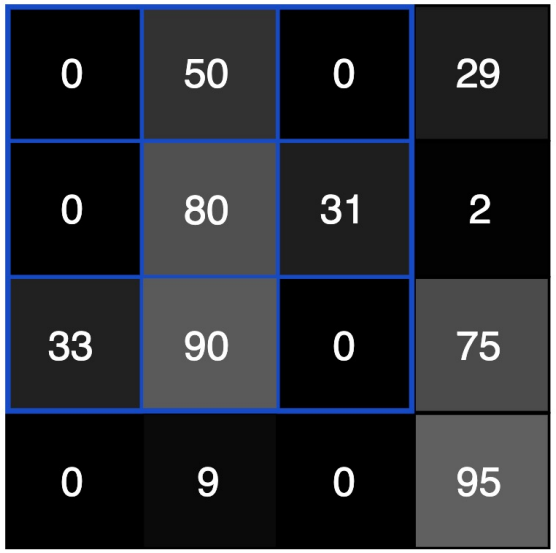
Convolve the input image and the filter to produce a 2x2 output image



?	?
?	?

0 is black and 255 is white

Two-dimensional convolution



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

Image Value	Filter Value	Result
0	-1	0
50	0	0
0	1	0
0	-2	0
80	0	0
31	2	62
33	-1	-33
90	0	0
0	1	0

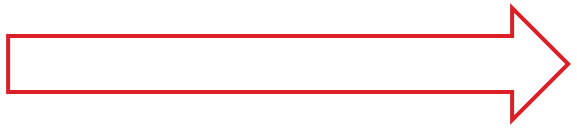
$$62 - 33 = \boxed{29}$$

Two-dimensional convolution

0	50	0	29
0	80	31	2
33	90	0	75
0	9	0	95

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

Convolve the input image and the filter to produce a 2x2 output image

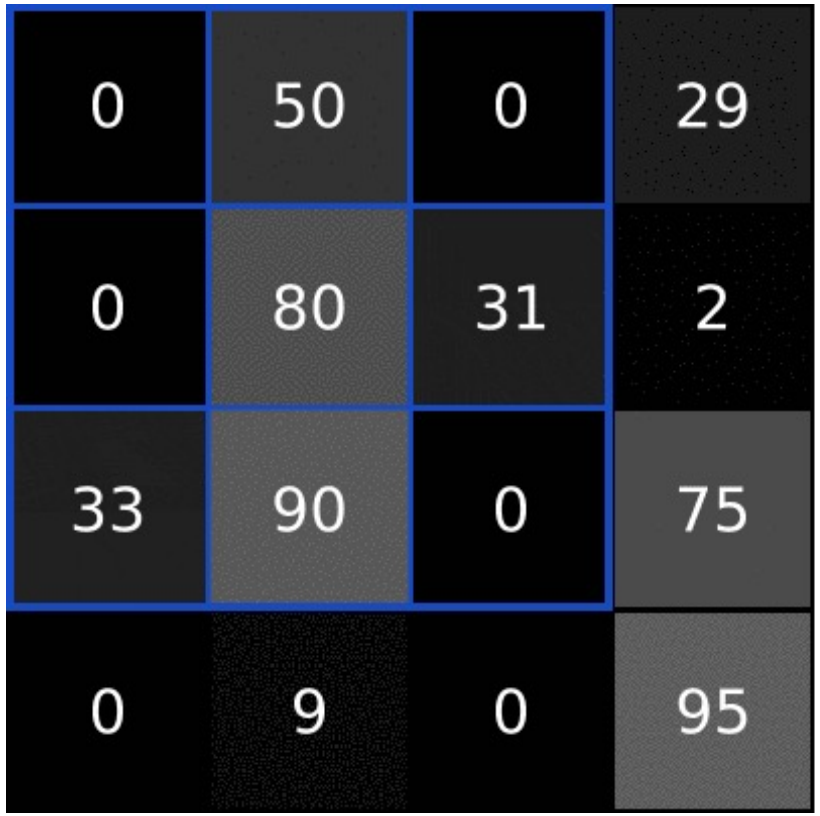


29	?
?	?

$$62 - 33 = \boxed{29}$$

Two-dimensional convolution

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



29	?
?	?

Two-dimensional convolution

- In image processing, the image is input into the neural network in the form of a two-dimensional matrix, so we need two-dimensional convolution.

$$y_{ij} = \sum_{u=1}^{U} \sum_{v=1}^{V} w_{uv} x_{i-u+1, j-v+1}.$$

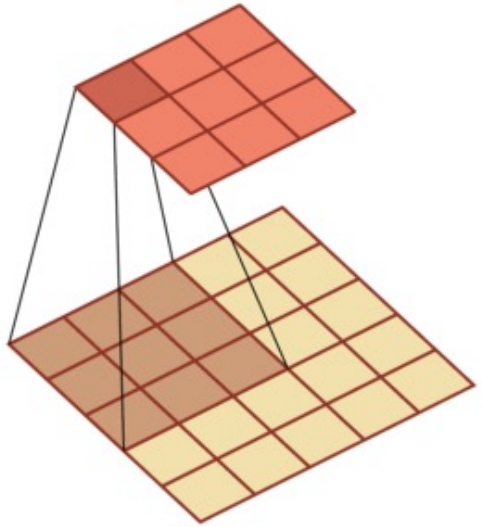
1	1	1 ×−1	1 ×0	1 ×0
−1	0	−3 ×0	0 ×0	1 ×0
2	1	1 ×0	−1 ×0	0 ×1
0	−1	1	2	1
1	2	1	1	1

*

1	0	0
0	0	0
0	0	−1

=

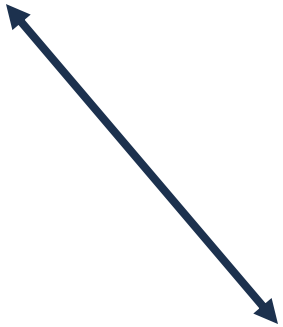
0	−2	−1
2	2	4
−1	0	0

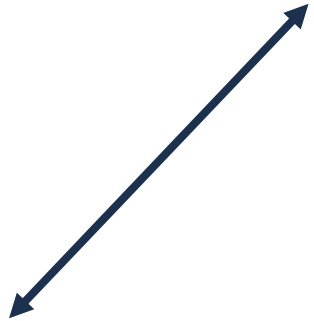


How is the convolution useful? Example

How to use CNN to describe the characteristics of this image?

What are the characteristics of this image?





How is the convolution useful? Example

How to use CNN to describe the characteristics of this image?

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

How is the convolution useful? Example

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

Convolutional kernel

1	0	0
0	1	0
0	0	1

characteristic

2	0	1	0	1
0	3	0	1	0
1	0	3	0	1
0	1	0	3	0
1	0	1	0	2

The larger the value is, the higher possibility it has the characteristics

How is the convolution useful?

- Sobel operator: https://en.wikipedia.org/wiki/Sobel_operator

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

The vertical Sobel filter

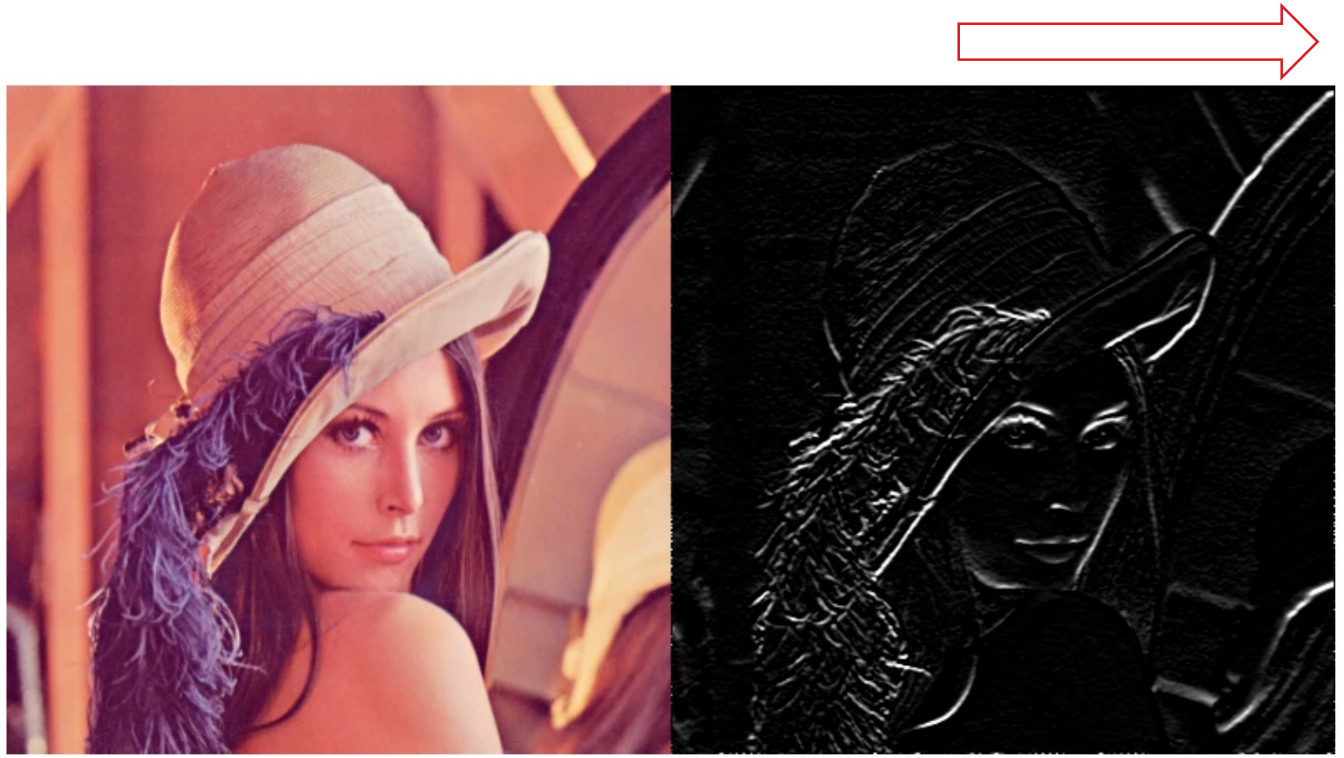
The vertical Sobel filter detects vertical edges

How is the convolution useful?

- Sobel operator: https://en.wikipedia.org/wiki/Sobel_operator

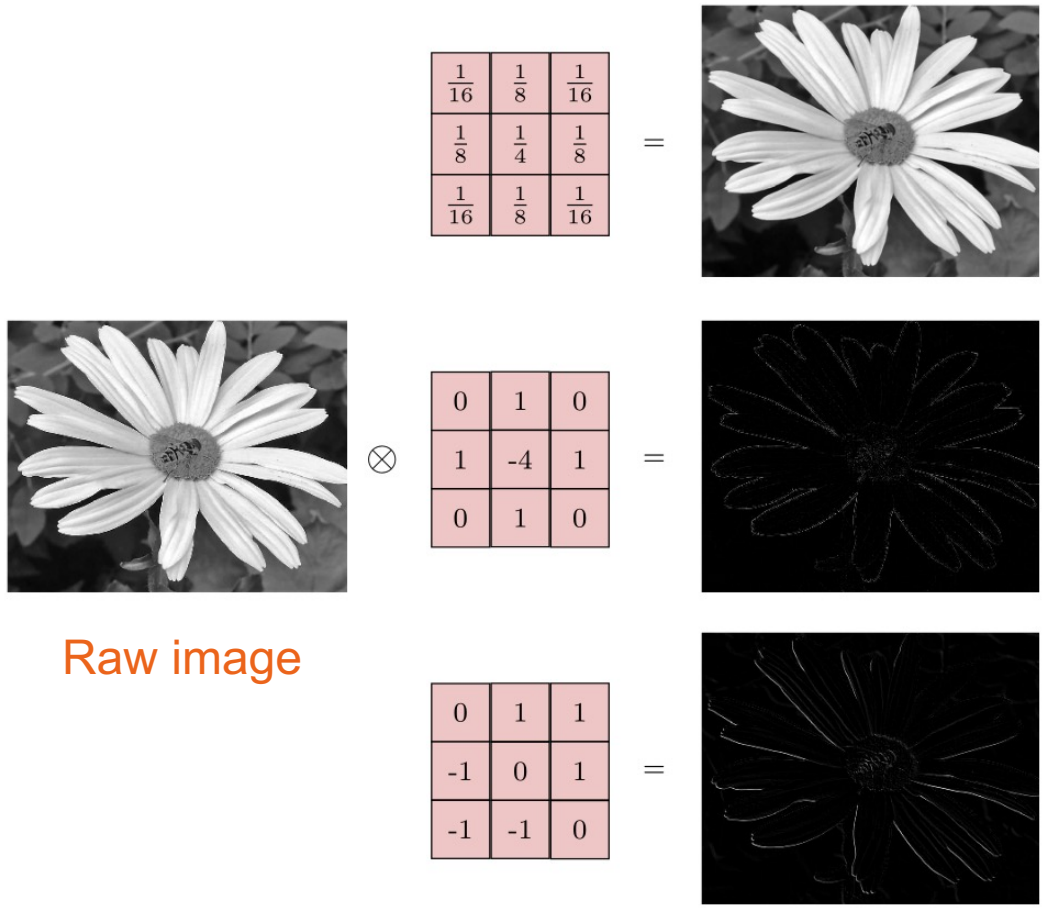
1	2	1
0	0	0
-1	-2	-1

The horizontal Sobel filter



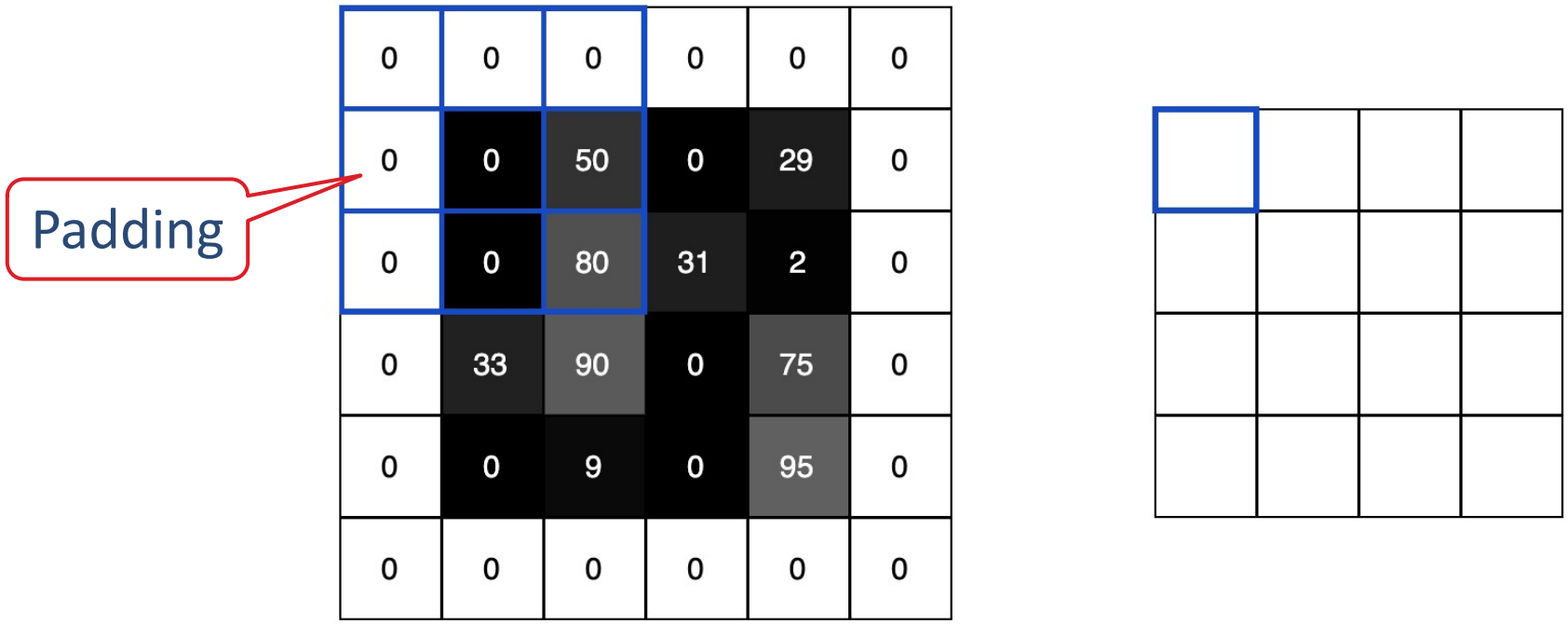
The horizontal Sobel filter detects horizontal edges

Convolution as a feature extractor



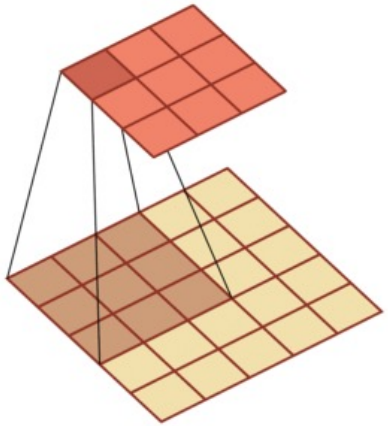
Two-dimensional convolution

- Padding: to have the output image be the same size as the input image, we add zeros around the image so we can overlay the filter in more places

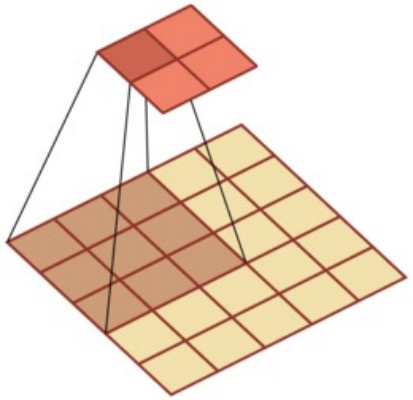


A 4x4 input convolved with a 3x3 filter to produce a 4x4 output using same padding

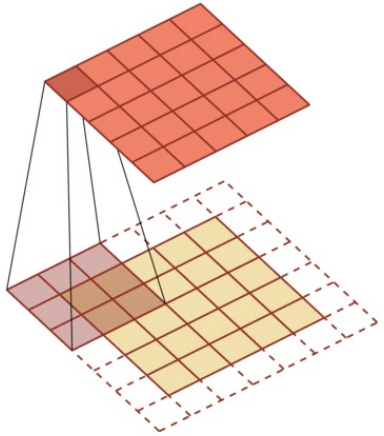
Two-dimensional convolution



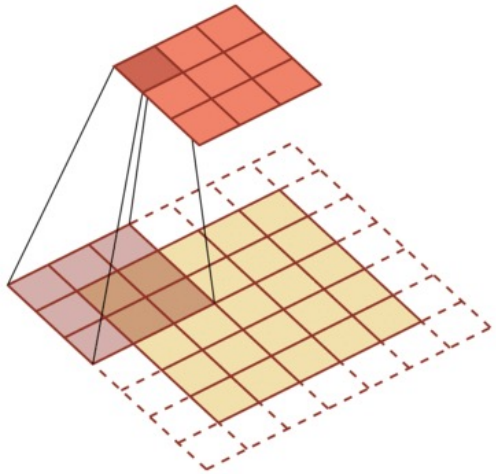
Step S = 1
Zero padding P = 0



Step S = 2
Zero padding P = 0



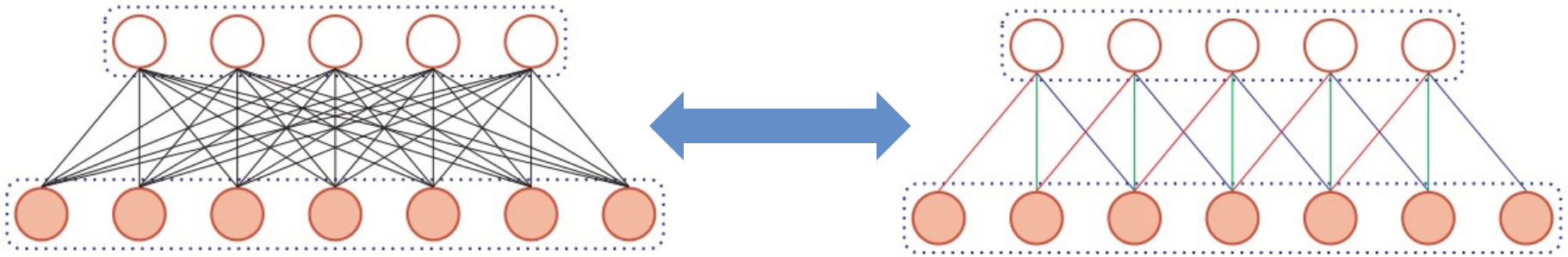
Step S = 1
Zero padding P =1



Step S = 2
Zero padding P = 1

Convolutional Neural Network

- Use convolutional layers instead of fully connected layers



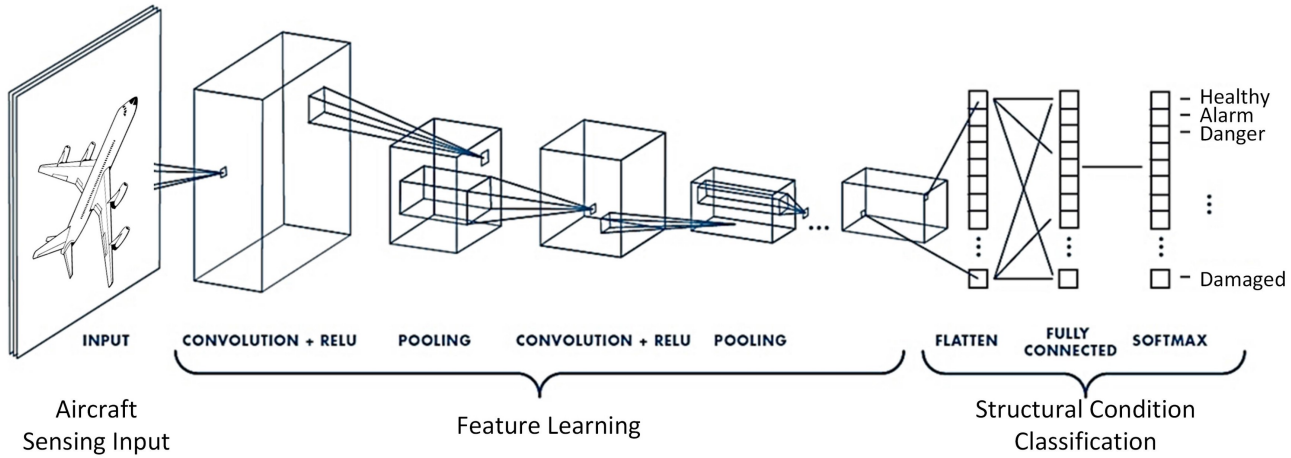
fully connected layers

convolutional layers

Multiple convolution kernels

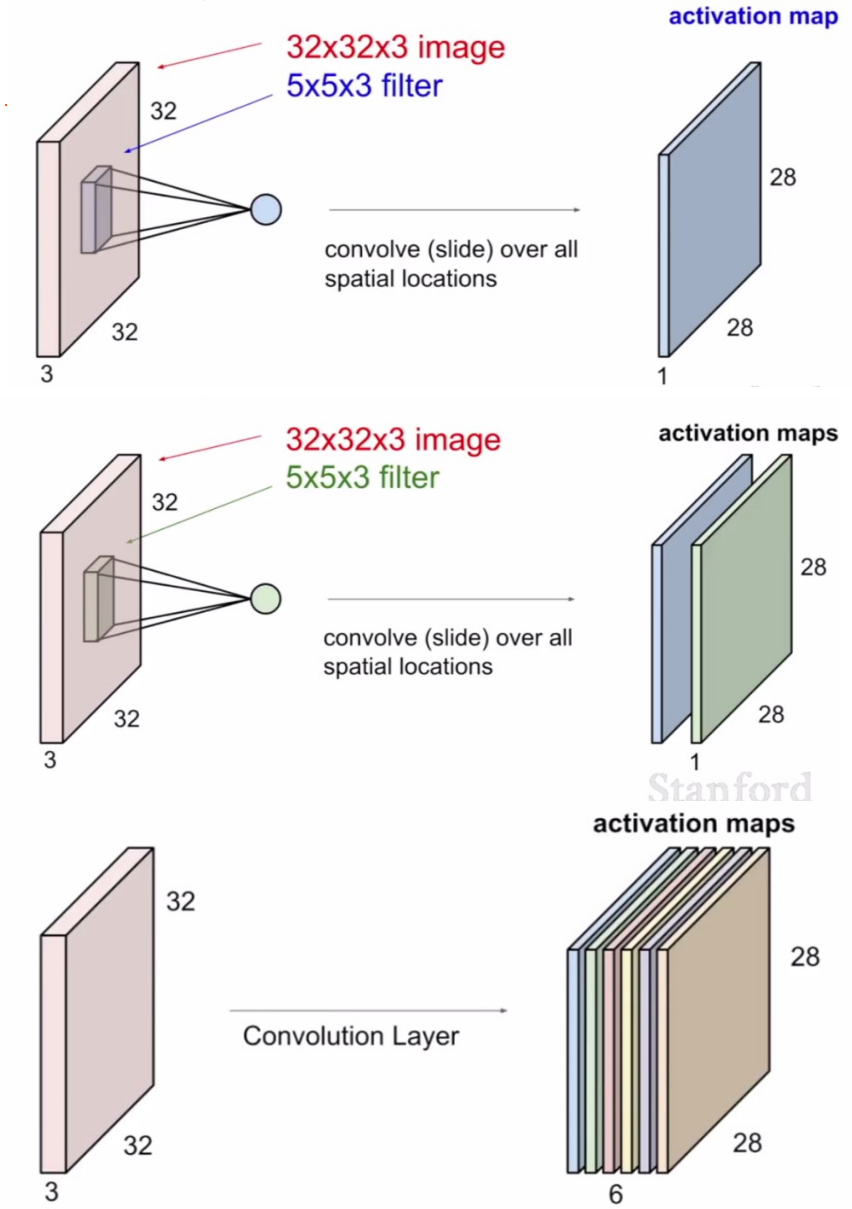
- Feature Map: Features obtained after image convolution.
  - Convolution kernel as a feature extractor

- Convolutional layer
  - Input: D feature maps M×N×D
  - Output: P feature maps M'×N'×P



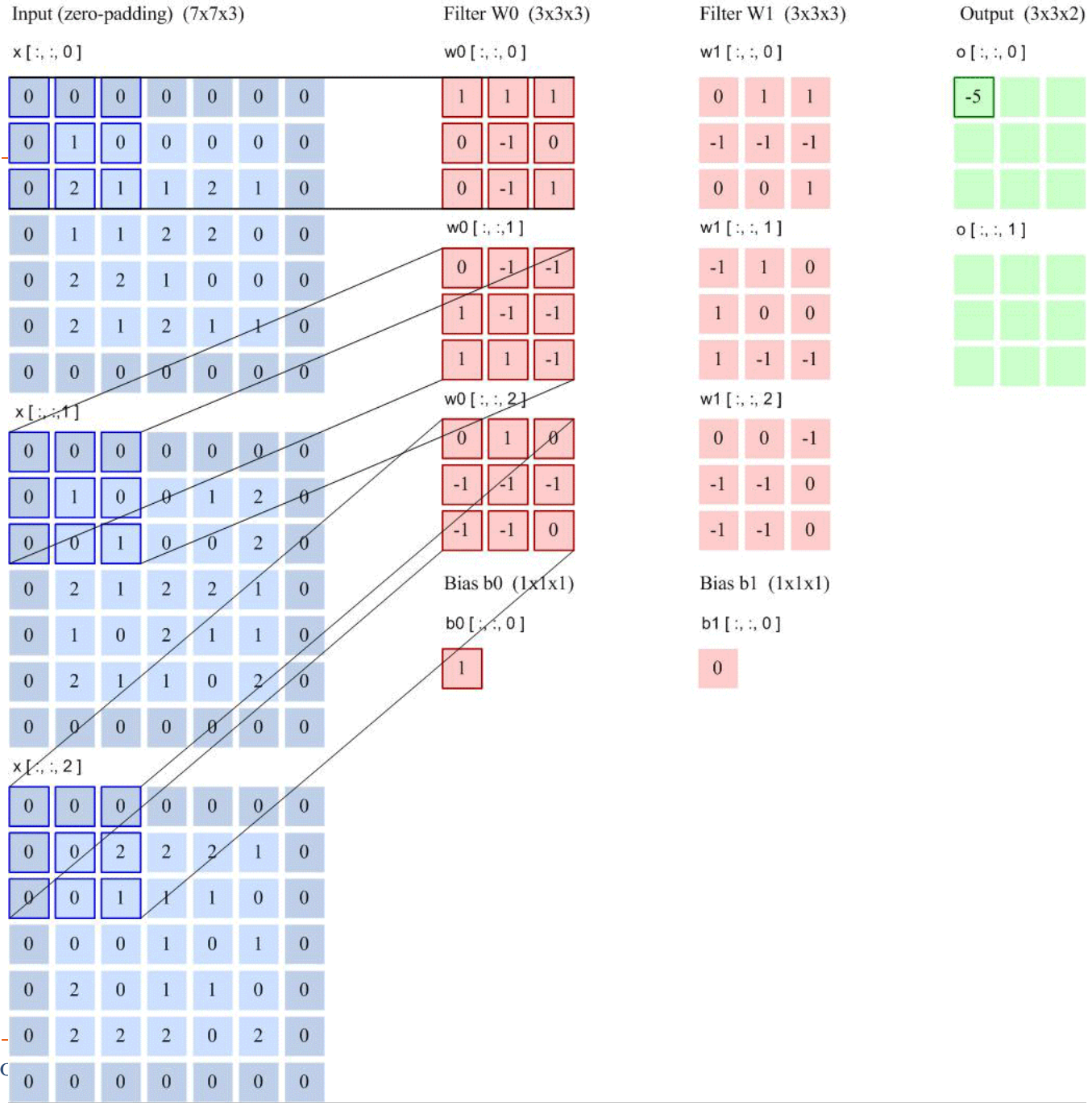
Multiple convolution kernels

- There is a formula which is used in determining the dimension of the activation maps:
  - (N + 2P — F)/ S + 1; where N = Dimension of image (input) file
    - P = Padding
    - F = Dimension of filter
    - S = Stride



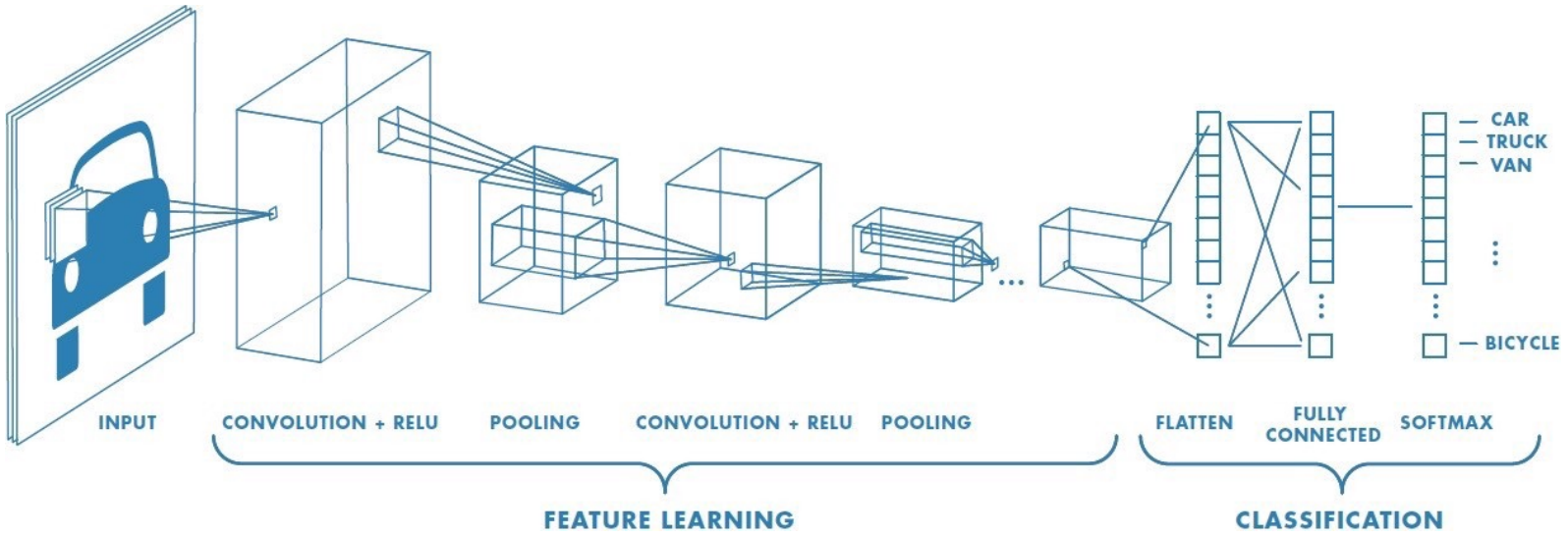
Convolutional layer mapping

Step = 2
Filter 3*3
Filter number = 6
zero padding = 1



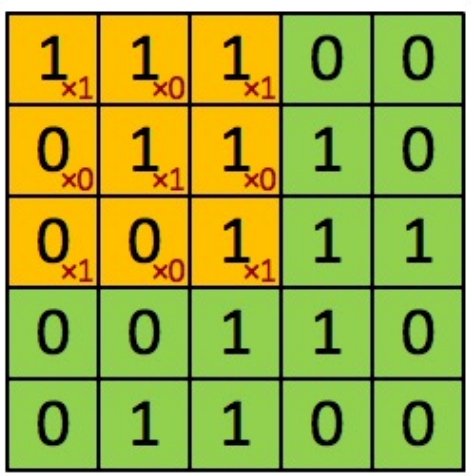
Convolutional Neural Network

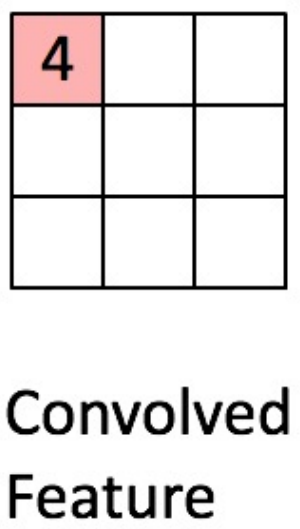
- A typical CNN consists of 3 parts:
  - Convolutional layer: Extract local features in the image
  - Pooling layer: Significantly reduce the magnitude of parameters (dimensionality reduction)
  - Fully connected layer: Used to output the desired result

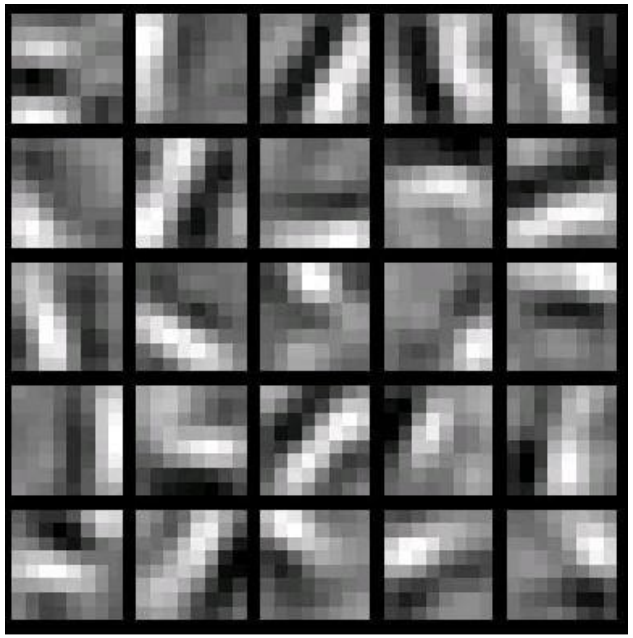


Convolutional layer

- Extract local features in the image





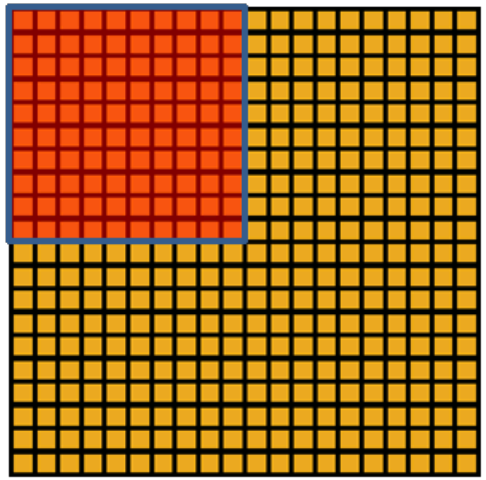


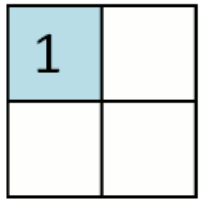
Use a convolution kernel to filter each small area of the image to get the feature value of these small areas

Extract local features in the picture

Pooling layer

- Data dimensionality reduction to avoid overfitting





Convolved feature

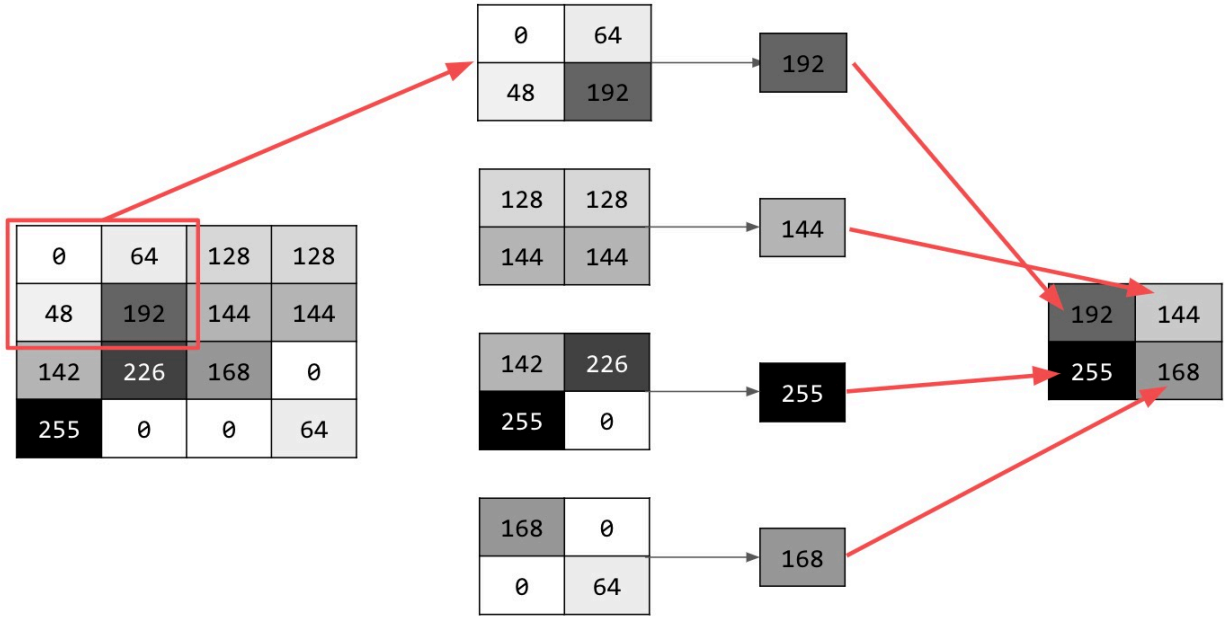
Pooled feature

The original picture is 20×20 and we downsample it, the sampling window is 10×10, and finally it is downsampled into a 2×2 feature map

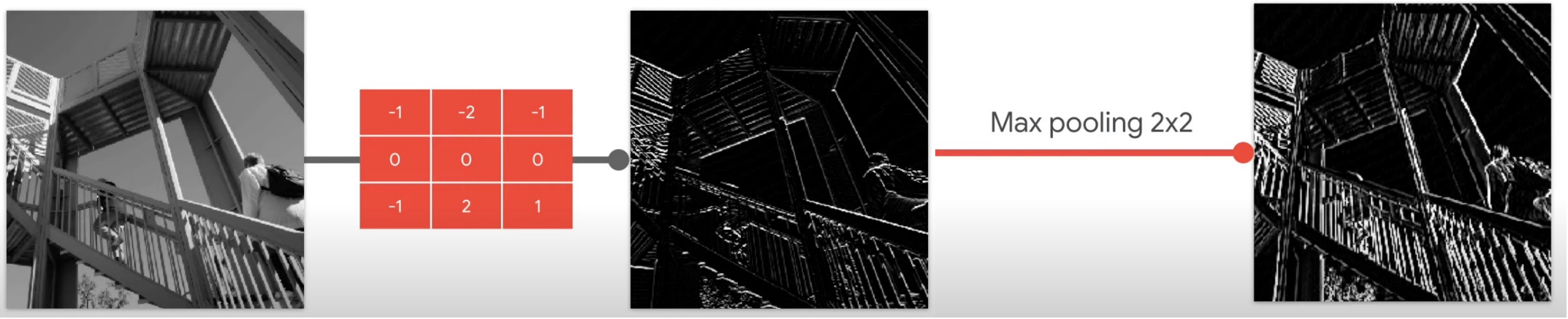
The pooling layer can reduce the data dimension more effectively than the convolutional layer. This can not only greatly reduce the amount of calculation, but also effectively avoid overfitting

Pooling layer example

- Take the largest of those (hence *max* pooling) and load it into the new image.
- The new image will be one-fourth the size of the old.



Pooling layer example

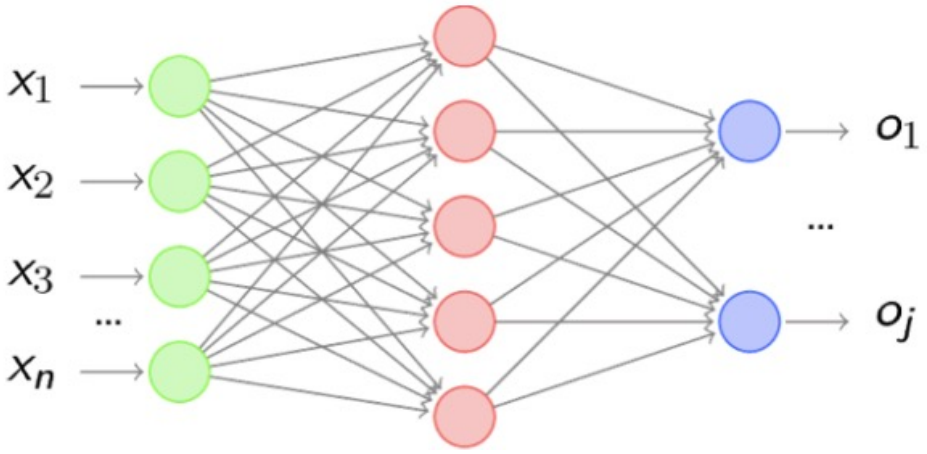


The image is now 512x512

The image is now 256x256, one-fourth of its original size, and the detected features have been enhanced despite less data now being in the image.

Fully connected layer

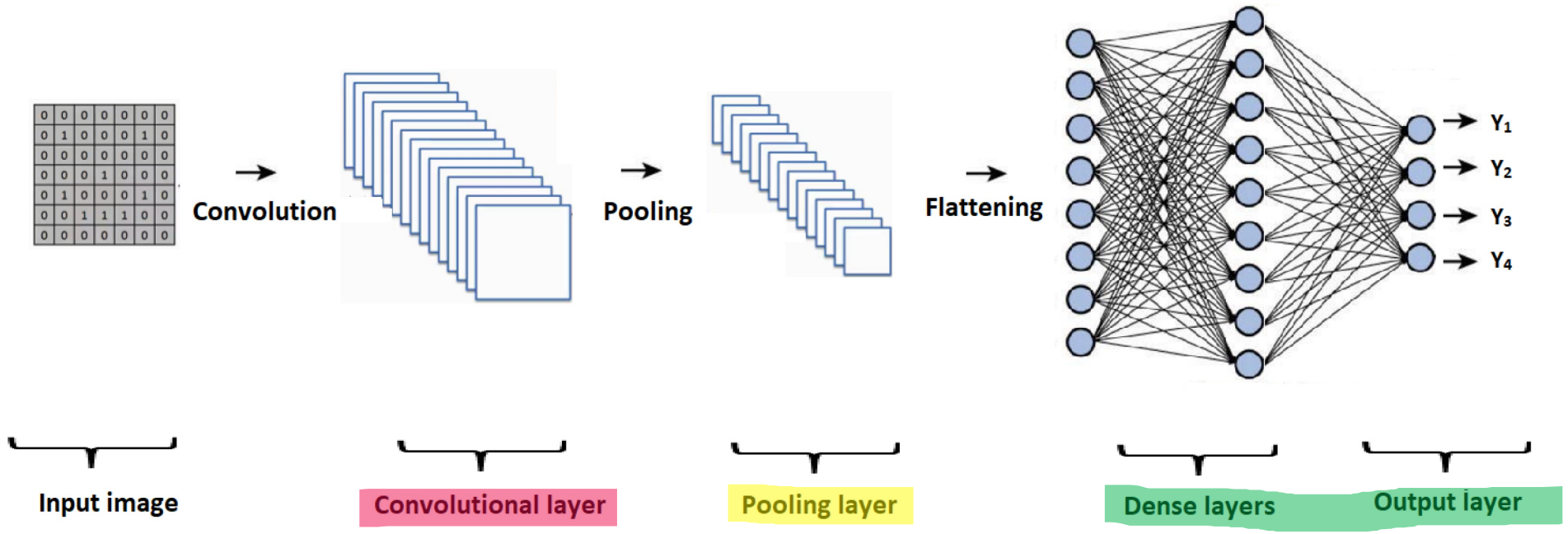
- The data processed by the convolutional layer and the pooling layer is input to the fully connected layer to obtain the final desired result



After the data has been reduced by the convolutional layer and the pooling layer, the fully connected layer can "run", otherwise the amount of data is too large, the calculation cost is high, and the efficiency is low

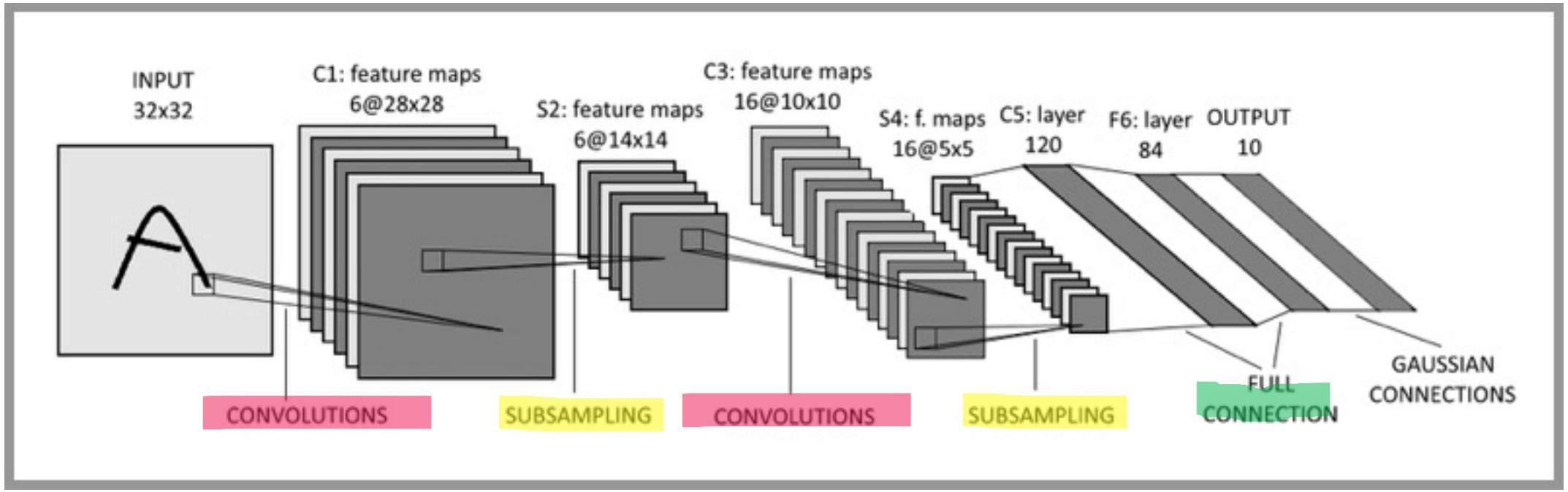
Put them together

- Convolutional network is composed of convolutional layer, convergence layer, and fully connected layer.



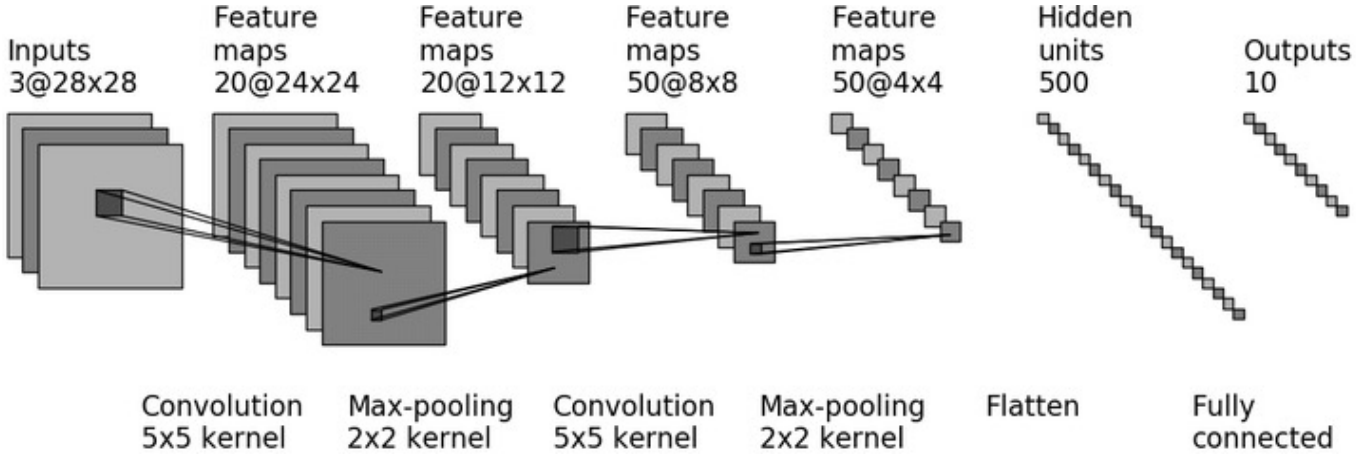
Put them together: LeNet-5

Multilayer structure

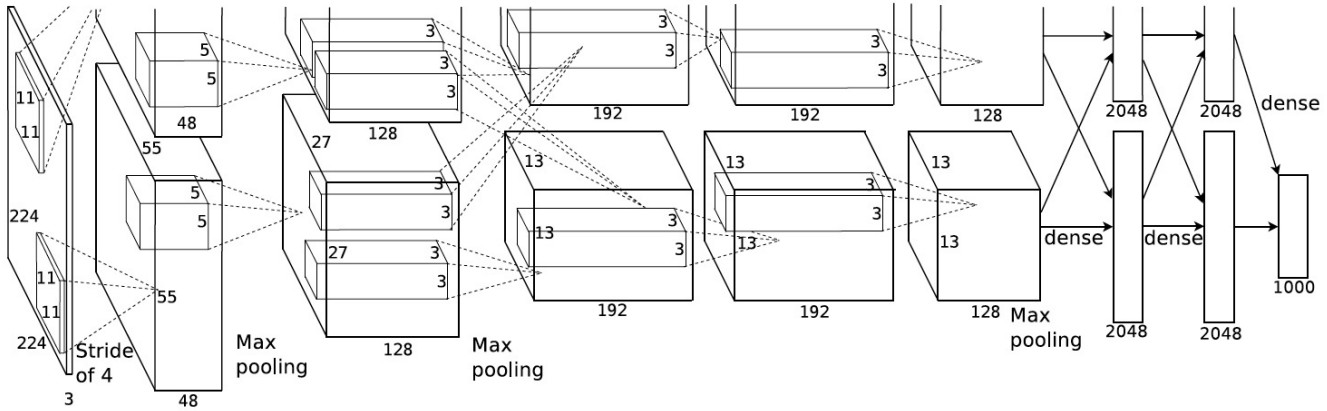


Some examples

Lenet-5



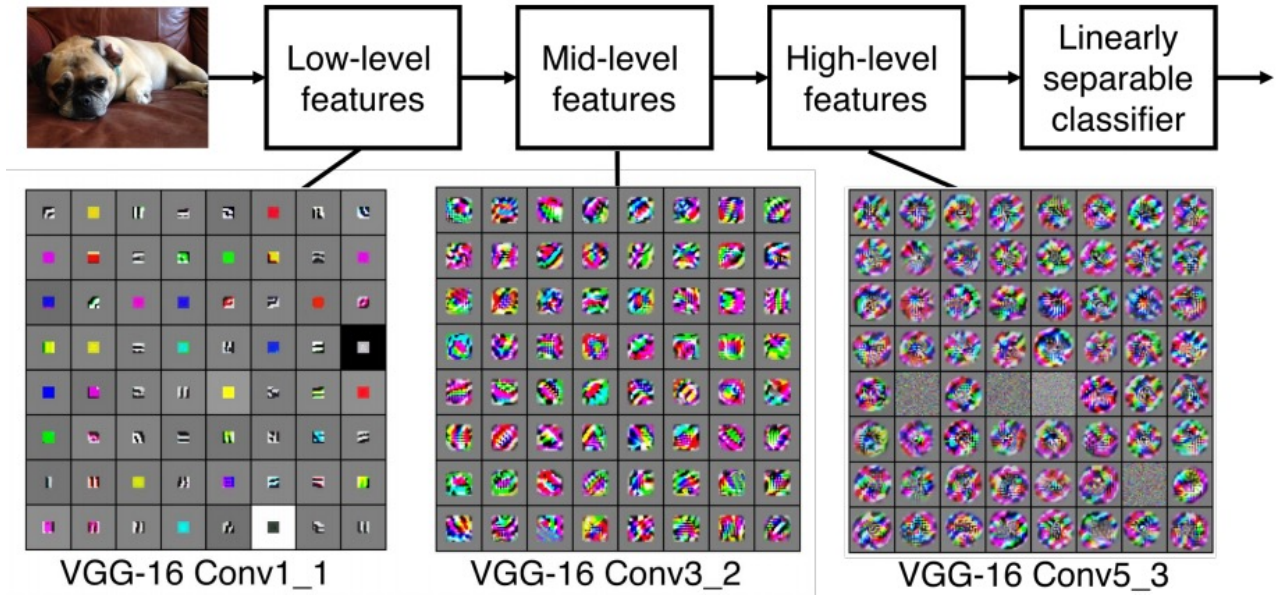
alexnet



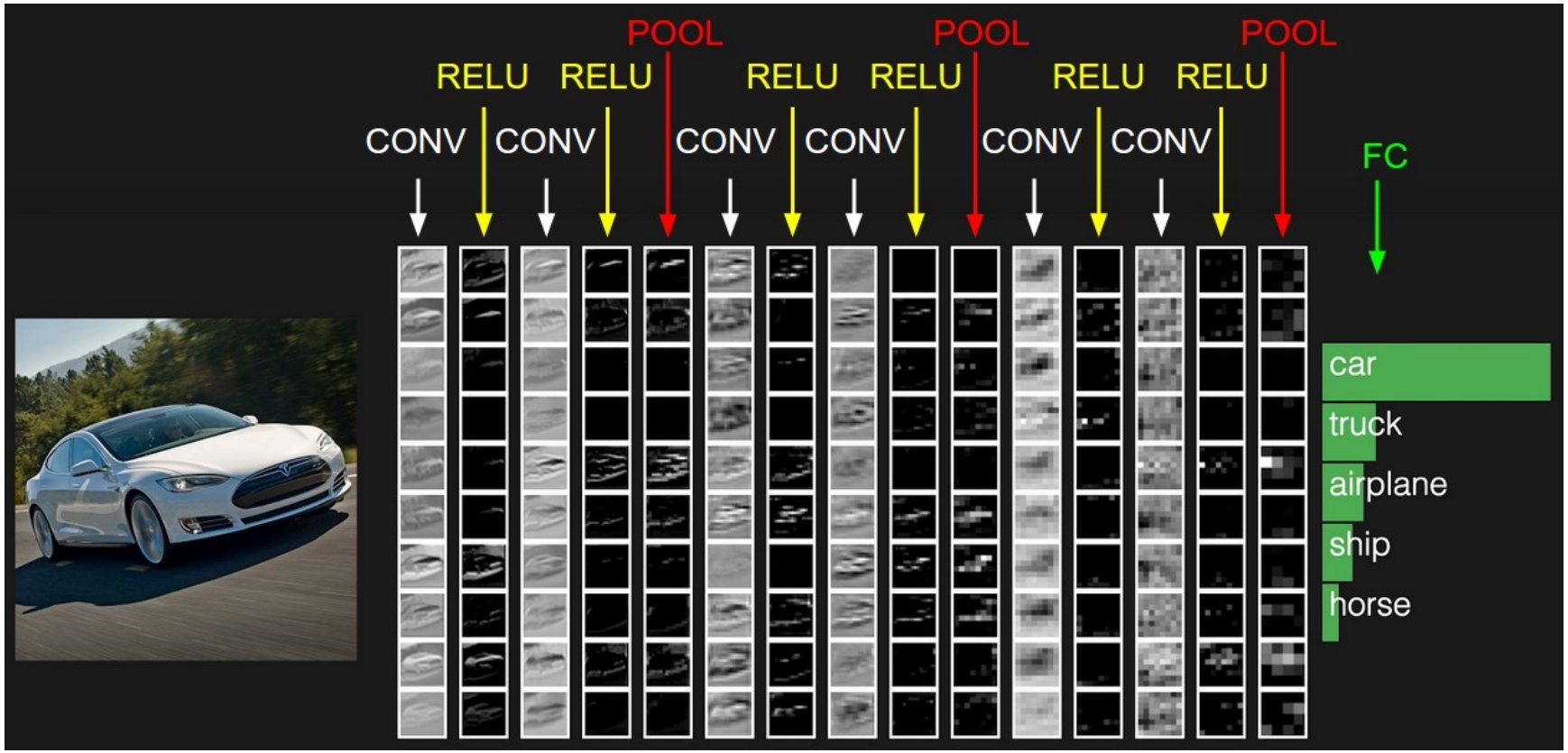
Representation Learning

- In order to improve the accuracy of the machine learning system, we need to convert the input information into effective features/representations

- Representation Learning: Automatically learn effective features to solve the semantic gap (the inconsistency and difference between the low-level features of the input data and the high-level semantic information)



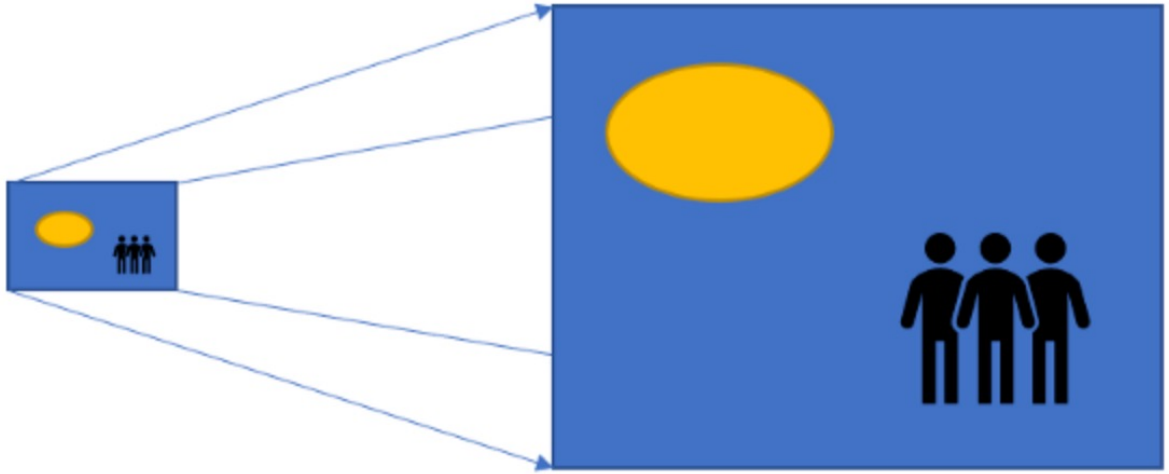
Representation Learning



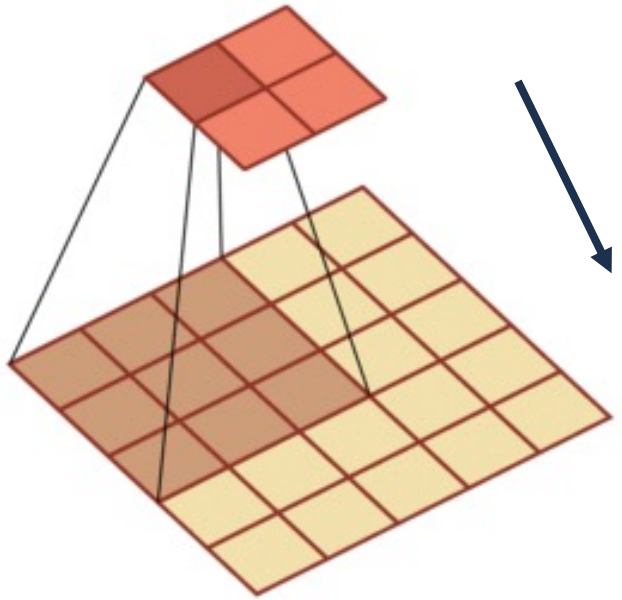
Other types of convolution

Transposed Convolution/Deconvolution

- When we use neural networks to generate pictures, we often need to convert some low-resolution pictures into high-resolution pictures

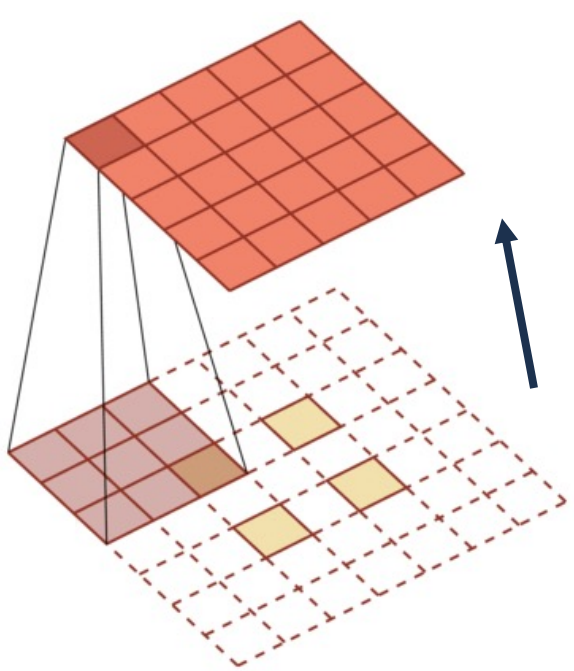


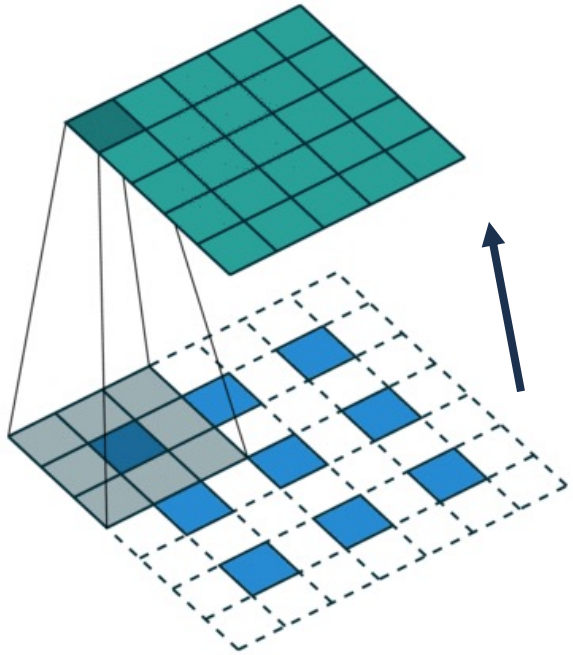
one-to-many mapping



Fractional stride convolution

- Mapping low-dimensional features to high-dimensional features





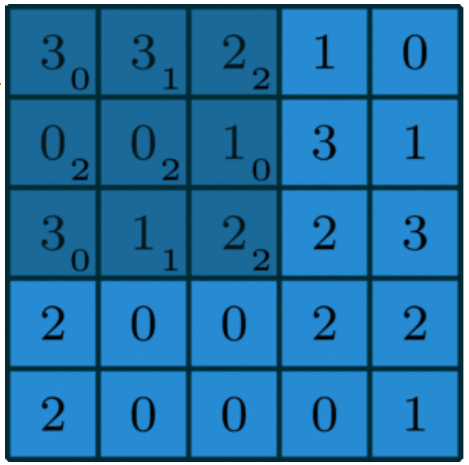
Fractional stride convolution

- In traditional 3*3 convolution, each point of the feature map in the upper layer will be convolved 3 times

- If this pixel is exactly the boundary point of the segmentation, it will blend with the surrounding pixels, which will cause the boundary of the entire semantic segmentation to be unclear

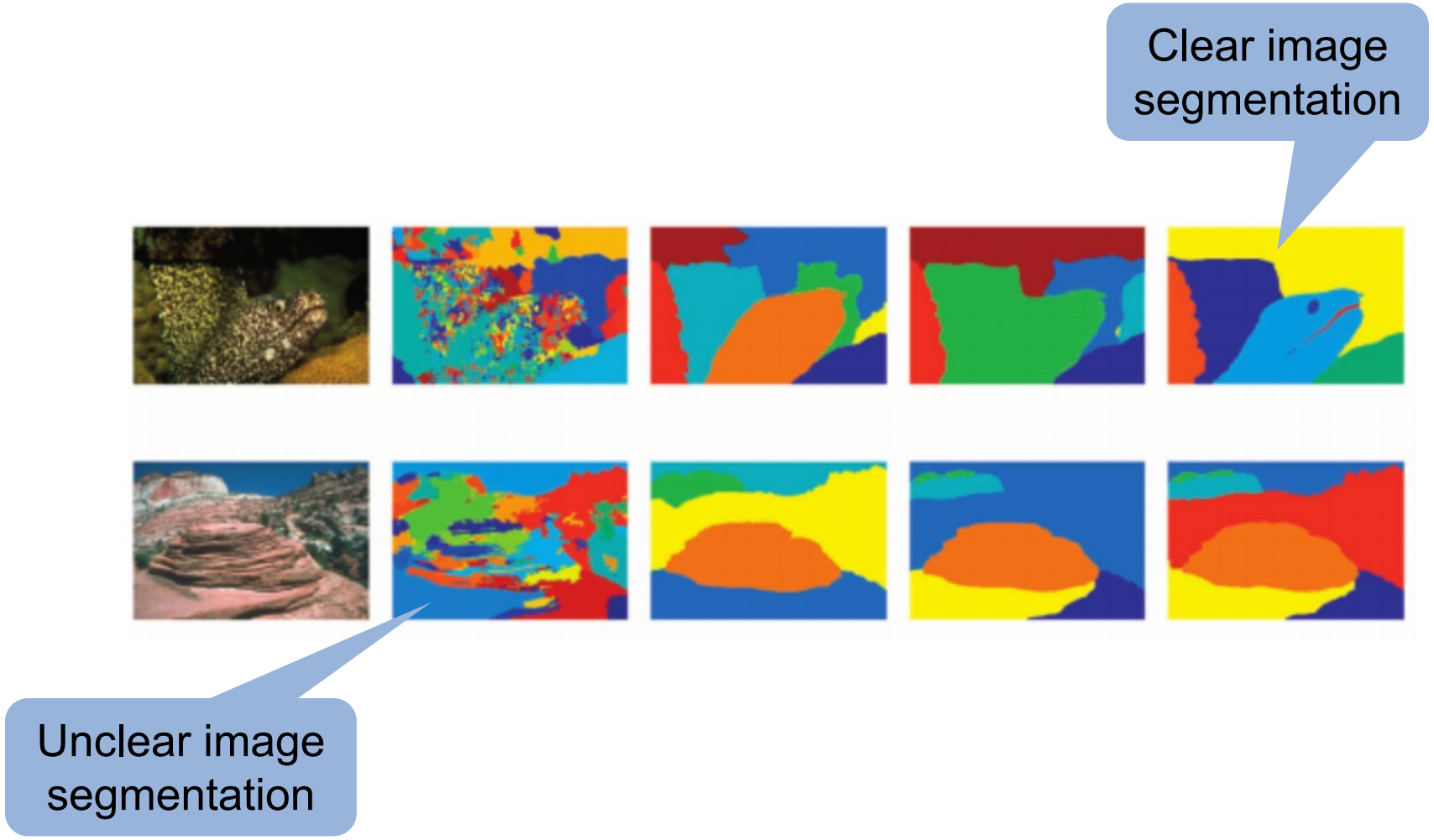
Dilated Convolution

- In traditional 3*3 convolution, each point of the feature map in the upper layer will be convolved 3 times



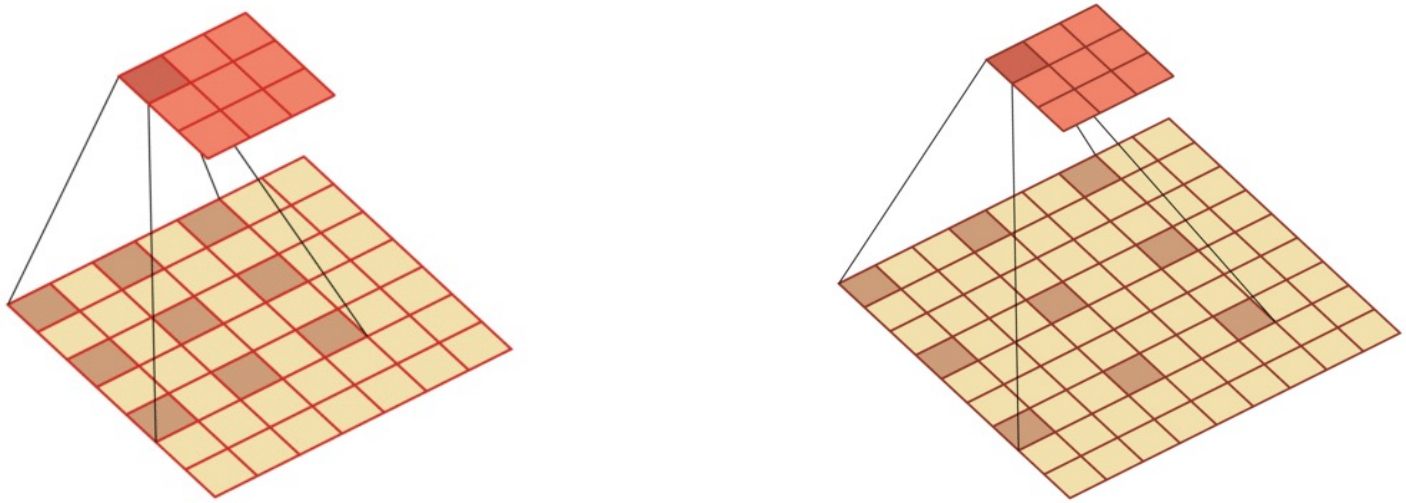
- If this pixel is exactly the boundary point of the segmentation, it will blend with the surrounding pixels, which will cause the boundary of the entire semantic segmentation to be unclear

Dilated Convolution



Dilated Convolution

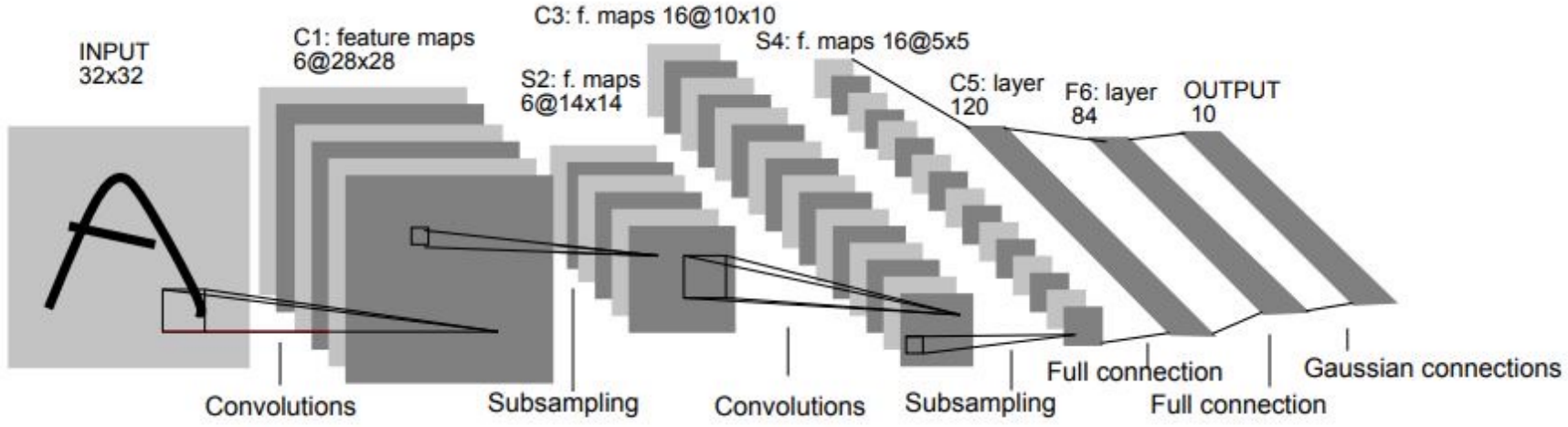
- Dilated convolution, which reduces the number of times each pixel is convolved to one layer, which can effectively focus on the semantic information of local pixel blocks and affect the fineness of segmentation



Typical convolutional network

LeNet-5

- LeNet-5 is a very successful neural network model.
  - The handwritten digit recognition system based on LeNet-5 was used by many American banks in the 1990s to recognize handwritten digits on checks.
  - LeNet-5 has 7 layers.



Large Scale Visual Recognition Challenge

2012 Teams	%error
Supervision (Toronto)	15.3
ISI (Tokyo)	26.1
VGG (Oxford)	26.9
XRCE/INRIA	27.0
UvA (Amsterdam)	29.6
INRIA/LEAR	33.4

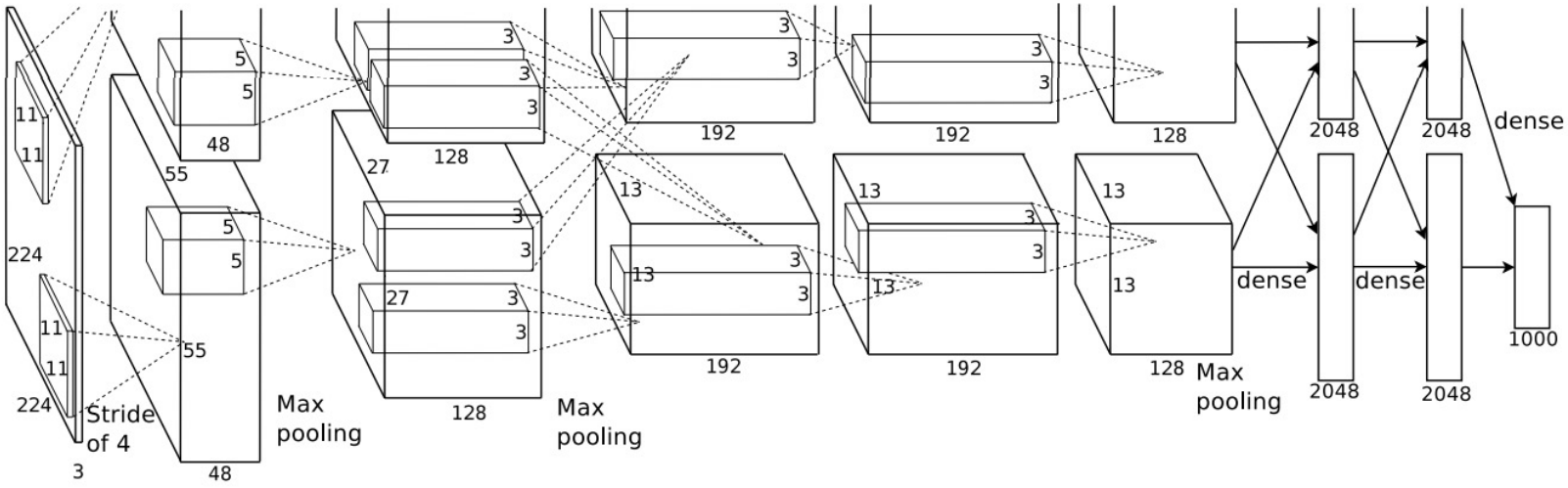
2013 Teams	%error
Clarifai (NYU spinoff)	11.7
NUS (singapore)	12.9
Zeiler-Fergus (NYU)	13.5
A. Howard	13.5
OverFeat (NYU)	14.1
UvA (Amsterdam)	14.2
Adobe	15.2
VGG (Oxford)	15.2
VGG (Oxford)	23.0

2014 Teams	%error
GoogLeNet	6.6
VGG (Oxford)	7.3
MSRA	8.0
A. Howard	8.1
DeeperVision	9.5
NUS-BST	9.7
TTIC-ECP	10.2
XYZ	11.2
UvA	12.1

https://medium.com/@smallfishbigsea/a-walk-through-of-alexnet-6cbd137a5637

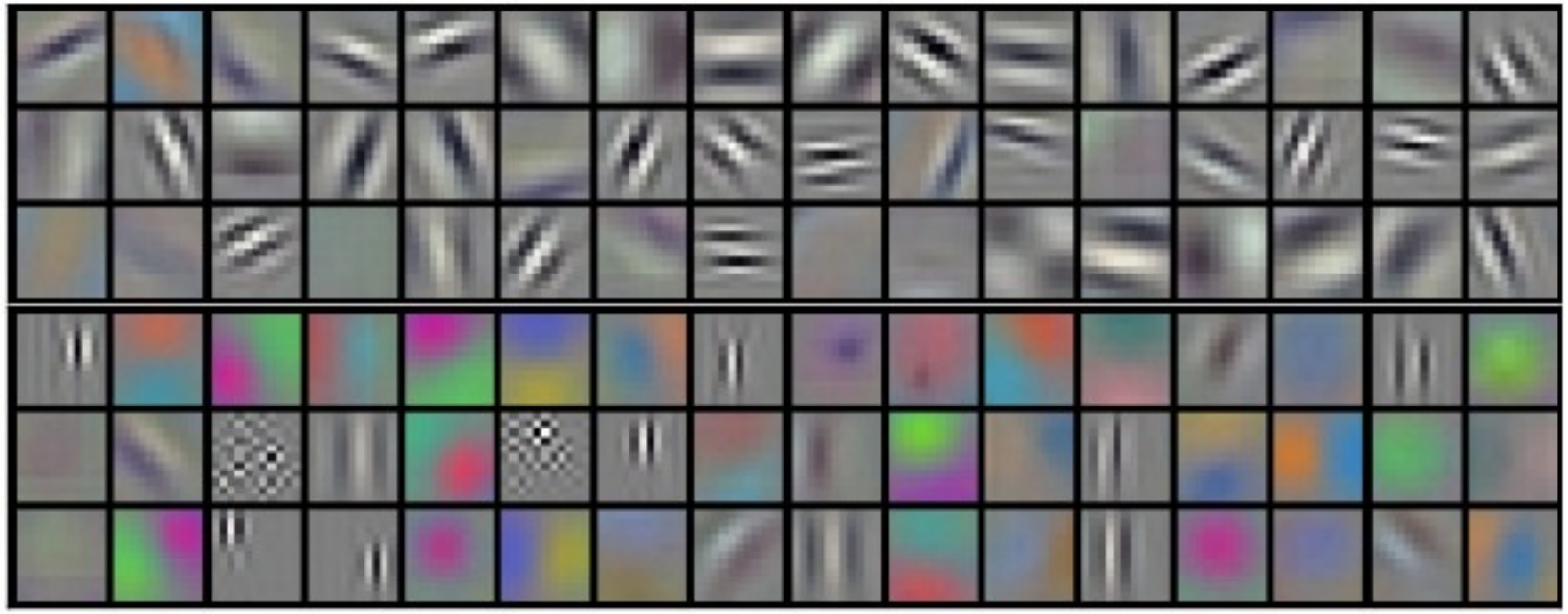
AlexNet

- 2012 ILSVRC winner
  - Top 5 error of 16% compared to runner-up with 26% error
  - The first modern deep convolutional network model
  - 5 convolutional layers, 3 pooling layers and 3 fully connected layers



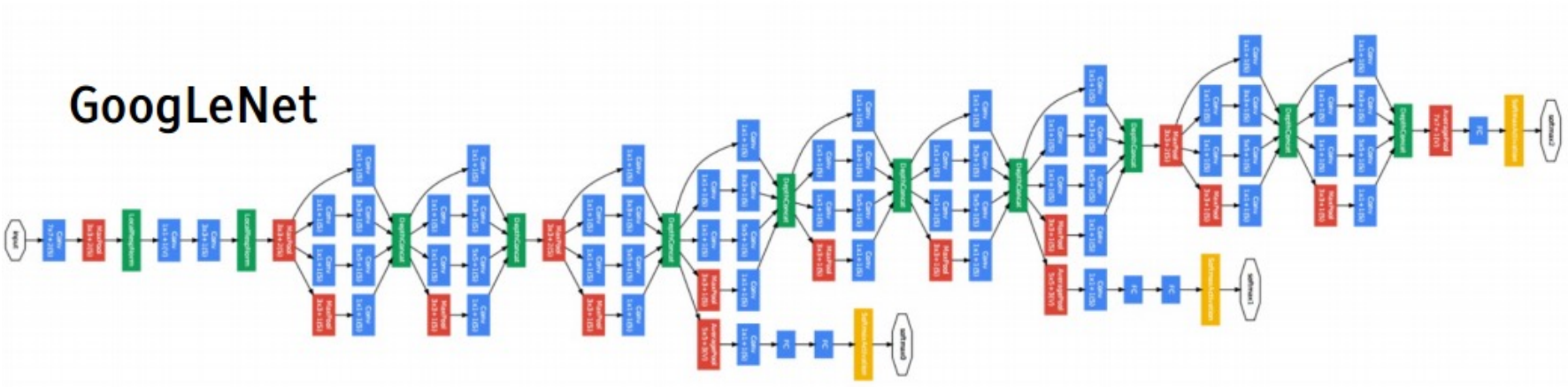
CNN visualization: filters

- Filters in AlexNet: 96 filters [11x11x3]



Inception

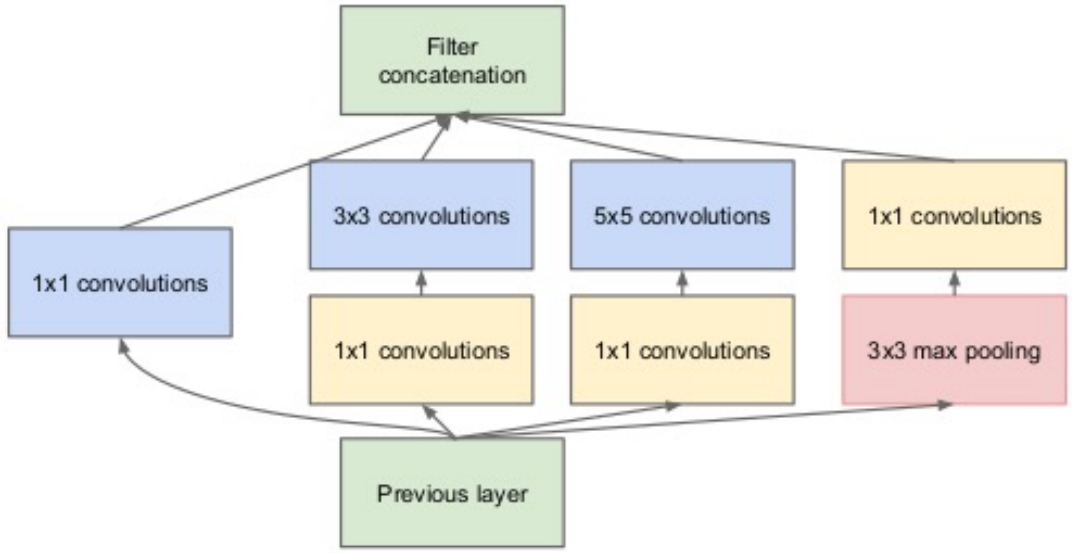
- 2014 ILSVRC winner (22 layers)
  - Parameters: GoogLeNet: 4M VS AlexNet: 60M
  - Error rate: 6.7%
  - The Inception network is made up of multiple inception modules and a small number of convergence layers.



Inception model v1

- In the Inception network, a convolutional layer contains multiple convolution operations of different sizes, called the Inception module.

- The Inception module uses convolution kernels of different sizes such as 1 × 1, 3 × 3, 5 × 5, etc., and stitches (stacks) the obtained feature maps in depth as output feature maps.



Inception model v3

- Use a multi-layered small convolution kernel to replace a large convolution kernel to reduce the number of calculations and parameters.
  - Use two layers of 3 × 3 convolution to replace the 5 × 5 convolution in v1
  - Use consecutive n × 1 and 1 × n to replace the convolution of n × n.

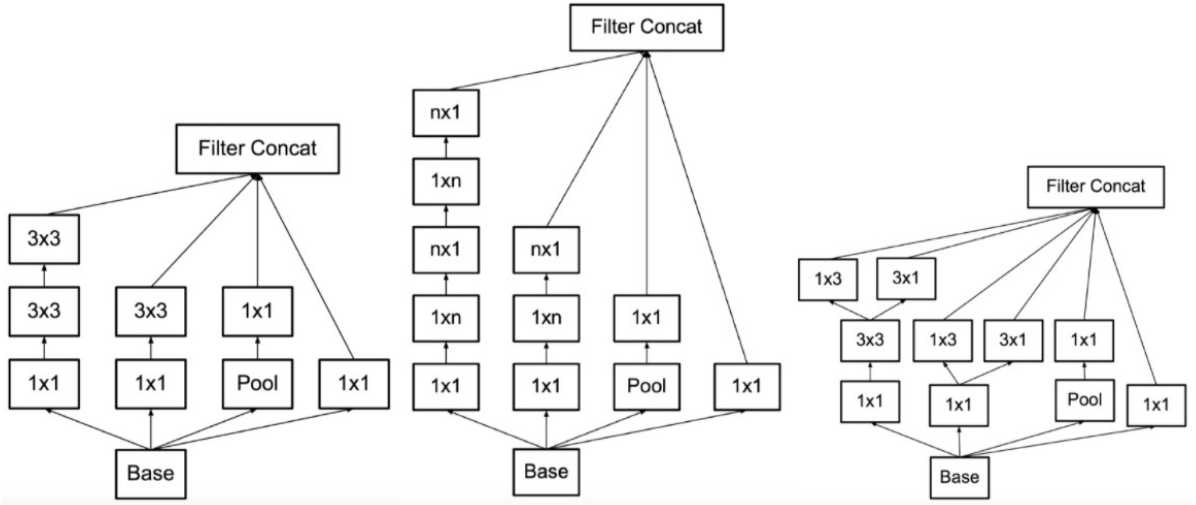


Fig. 5. 3 x Inception-A | Fig. 6. 5 x Inception-B | Fig. 7. 2 x Inception-C

Residual Network

- Will the accuracy of the network increase as the number of layers of the network increases?

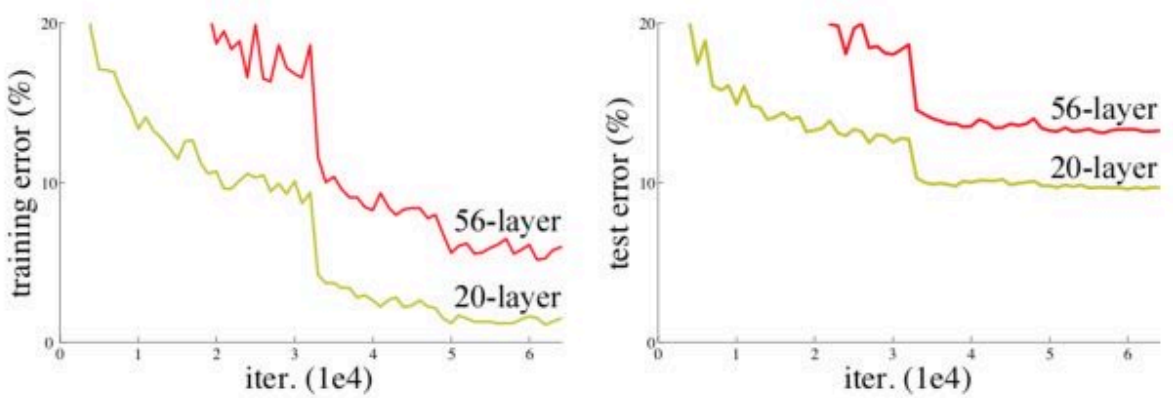
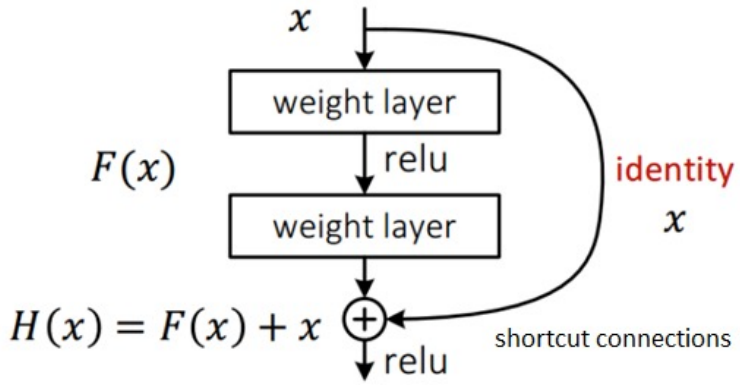


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

The 56-layer network has a larger error than the 20-layer network in terms of training error or test error

Residual Network

- Suppose that the input of a certain neural network is x, and the expected output is H(x)

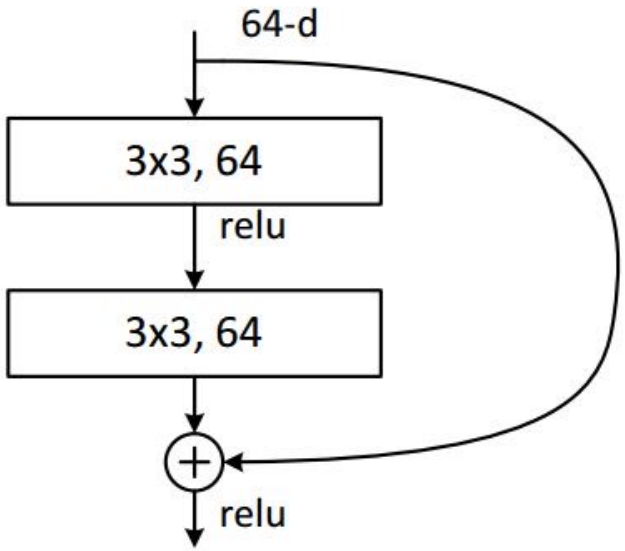


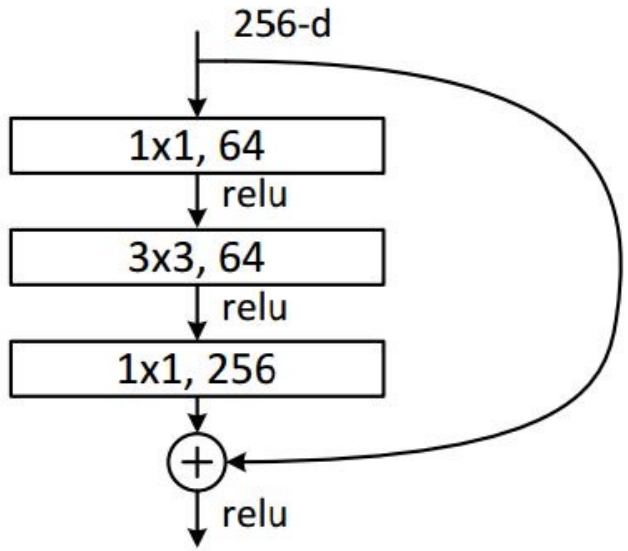
- If you have learned a saturated accuracy rate (or when the error of the lower layer is found to become larger), then the next learning goal is changed to the learning of identity mapping (which is H(x) = x)

- In the residual network structure diagram of the above figure, H(x)=F(x)+x, when F(x)=0, then H(x)=x. Therefore, the following training goal is to approach the residual result F(x) to 0 so that as the network deepens, the accuracy does not decrease

Residual unit

- There are two types of ResNet blocks, a two-layer structure and a three-layer structure

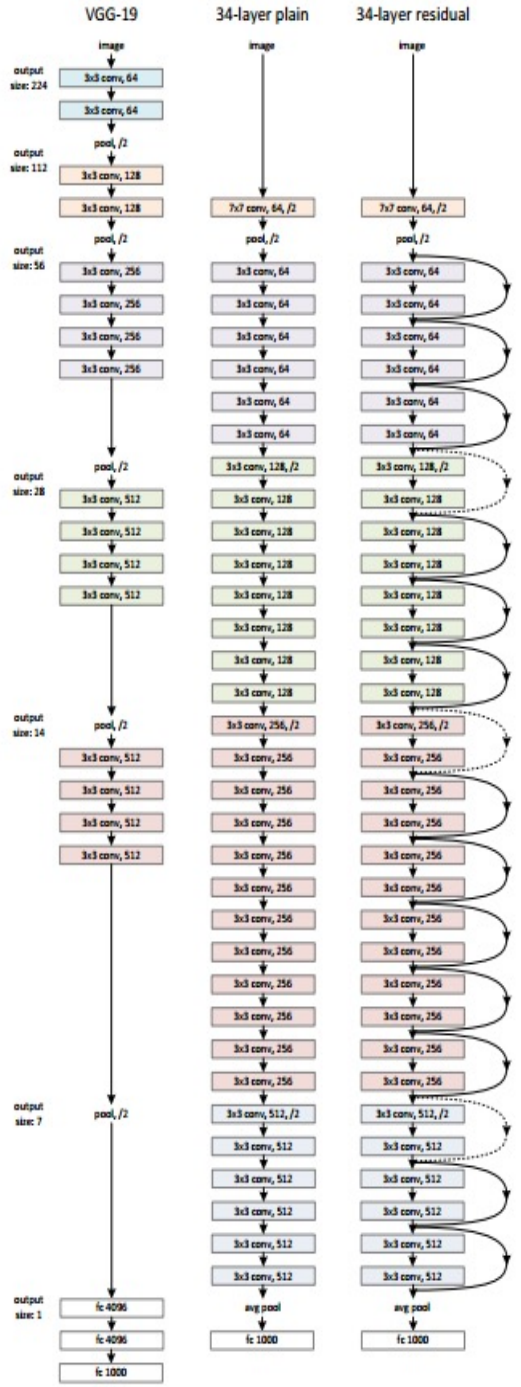




ResNet can make neural networks reach dozens of layers, hundreds of layers or even thousands of layers

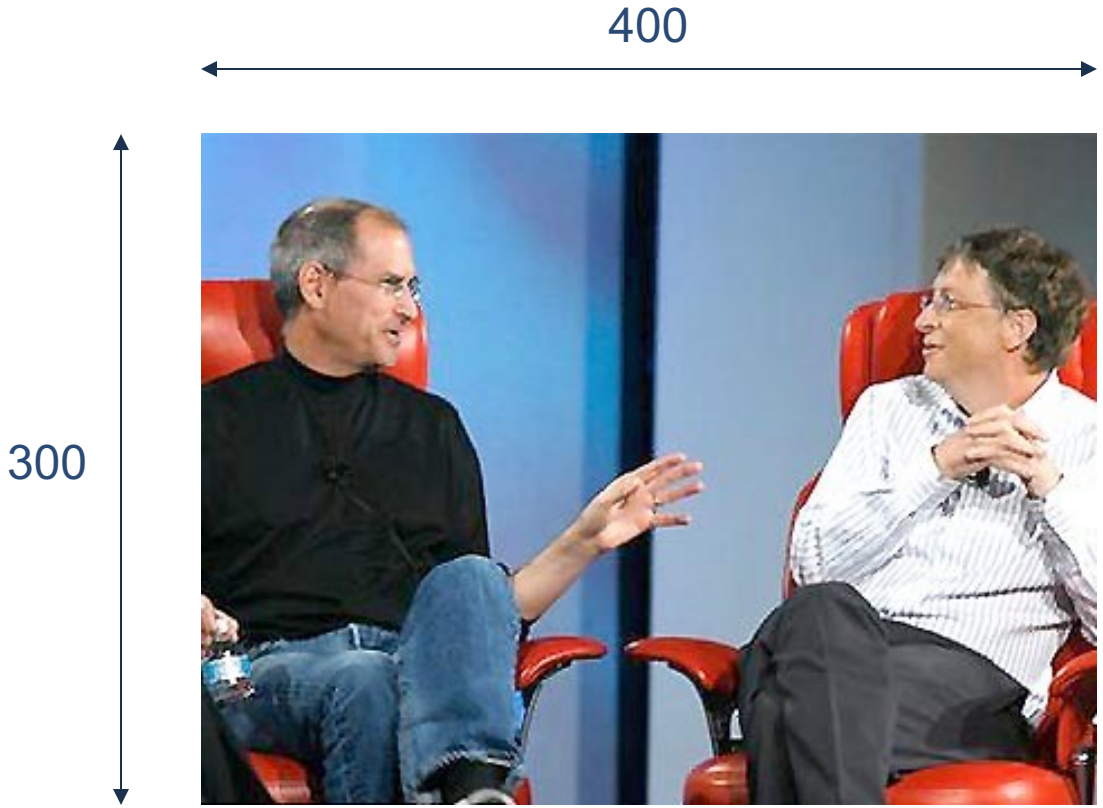
ResNet

- 2015 ILSVRC winner (152 layers)
  - Error ratio： 3.57%



Build a CNN

https://pinetools.com/rgb-channels-image



Matrix PI[400, 300, 3]

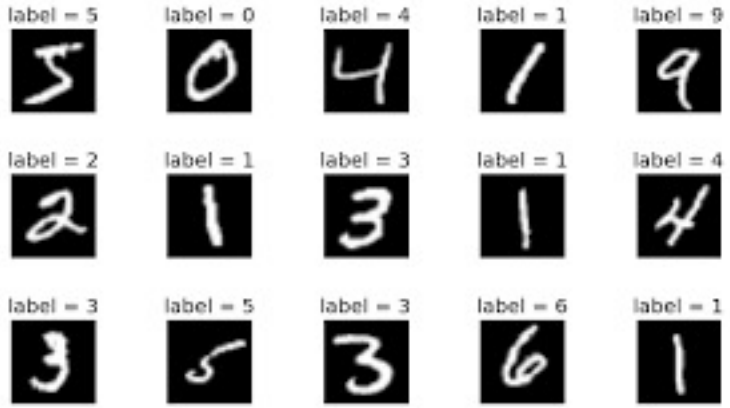
Build CNN using Keras

- Learn to use Keras to build a convolutional neural network step by step

- Use convolutional neural network to recognize handwritten digits (MNIST data set) and achieve a correct rate of over 99%

The MNIST database of handwritten digits

- http://yann.lecun.com/exdb/mnist/



- The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples

- It has four parts:
  - Training set images: train-images-idx3-ubyte.gz (9.9 MB, 47 MB after decompression, containing 60,000 samples)
  - Training set labels: train-labels-idx1-ubyte.gz (29 KB, 60 KB after decompression, containing 60,000 labels)
  - Test set images: t10k-images-idx3-ubyte.gz (1.6 MB, 7.8 MB after decompression, including 10,000 samples)
  - Test set labels: t10k-labels-idx1-ubyte.gz (5KB, 10 KB after decompression, containing 10,000 labels)

About Keras

- Keras is an advanced neural network API written in Python. It can run with TensorFlow, CNTK, or Theano as the backend

- https://keras.io/

- Benefits:
  - Allow simple and fast prototyping (due to user-friendliness, high modularity, and scalability).
  - Supports convolutional neural networks and recurrent neural networks, and a combination of the two.
  - Runs seamlessly on CPU and GPU

Install Keras

 # GPU version

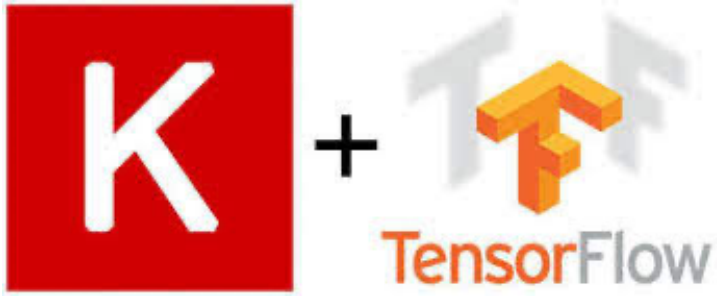
>>> pip3 install --upgrade tensorflow-gpu

 # CPU version

>>> pip3 install --upgrade tensorflow

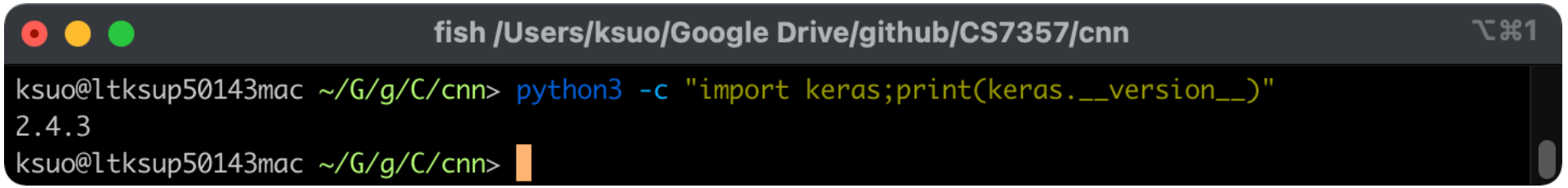
 # Keras installation

>>> pip3 install keras -U --pre



Confirm successful installation

>>> python3 -c "import keras;print(keras.__version__)"



Step 1: Import libraries and modules

- Import the Sequential model (equivalent to a table for building blocks)

```
from keras.models import Sequential
```

- Into various layers (building blocks of various shapes)

```
from keras.layers import Conv2D, MaxPool2D
from keras.layers import Dense, Flatten
```

- Import the to_categorical function to convert the data later

```
from keras.utils import to_categorical
```

Step 2: Import data

- Use Keras to load the handwritten digits dataset MNIST

```
from keras.datasets import mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

- Check the size of the train data

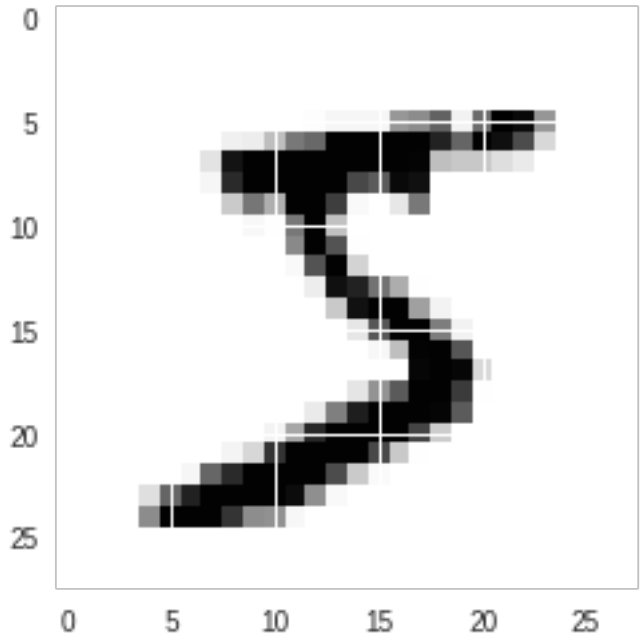
```
print(x_train.shape)

# (60000, 28, 28)
```

60000 samples, the size is 28 pixels x 28 pixels

Step 2: Import data

- Take a look at what these handwritten numbers look like

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.imshow(x_train[0])
```

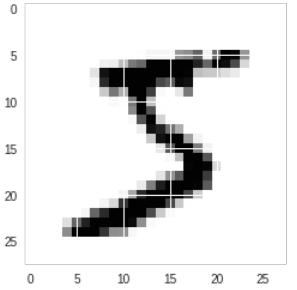


Step 3: Preprocess data

- Keras must explicitly declare the size of the input image depth, so we convert the data set from shape (n, rows, cols) to (n, rows, cols, channels)

```
img_x, img_y = 28, 28
x_train = x_train.reshape(x_train.shape[0], img_x, img_y, 1)
x_test = x_test.reshape(x_test.shape[0], img_x, img_y, 1)
```

The depth of the MNIST image is 1



Color image has 3 RGB channels

Step 3: Preprocess data

- Standardize the data:

```
x_train = x_train.astype('float32’)
x_test = x_test.astype('float32’)
x_train /= 255
x_test /= 255
```

Why One-Hot Encode Data in Machine Learning? https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/

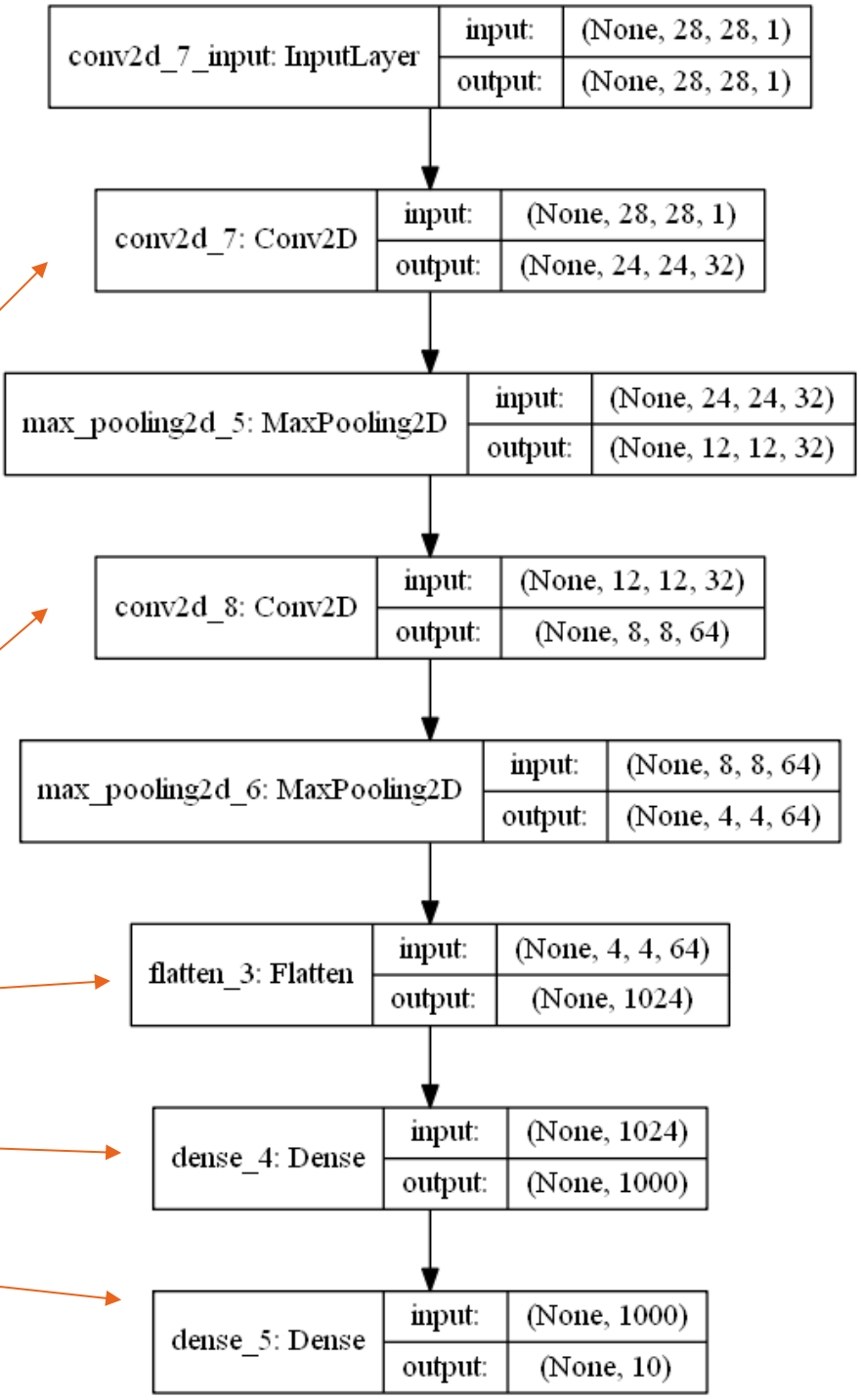
- Convert the labeled values (y_train, y_test) to the form of One-Hot Encode

```
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

Step 4: define model structure

- We refer to the following figure to define a model structure

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(5,5), activation='relu',
          input_shape=(img_x, img_y, 1)))
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
model.add(Conv2D(64, kernel_size=(5,5), activation='relu'))
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
model.add(Flatten())
model.add(Dense(1000, activation='relu'))
model.add(Dense(10, activation='softmax'))
```



Step 5: Compile

- Declare loss function and optimizer (SGD, Adam, etc.)

```
model.compile(optimizer='adam’,
              loss='categorical_crossentropy’,
              metrics=['accuracy’])
```

- Losses in Keras: https://keras.io/api/losses/

- Optimizers in Keras: https://keras.io/api/optimizers/

Available optimizers

- SGD
- RMSprop
- Adam
- Adadelta
- Adagrad
- Adamax
- Nadam
- Ftrl

Available losses

Note that all losses are available both via a class handle handles enable you to pass configuration arguments to `CategoricalCrossentropy(from_logits=True)`), and the in a standalone way (see details below).

Probabilistic losses

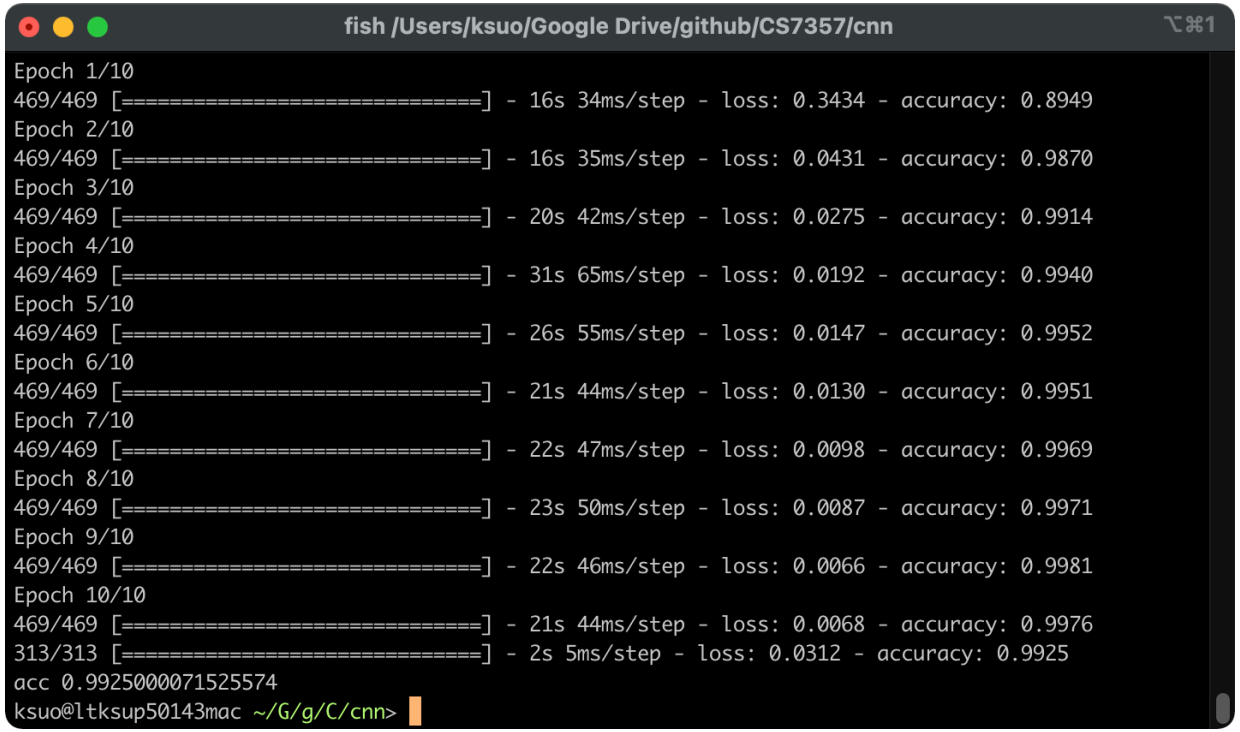
- BinaryCrossentropy class
- CategoricalCrossentropy class
- SparseCategoricalCrossentropy class
- Poisson class
- binary_crossentropy function
- categorical_crossentropy function
- sparse_categorical_crossentropy function
- poisson function
- KLDivergence class
- kl_divergence function

Step 6: Train

- Pass in the training set for training

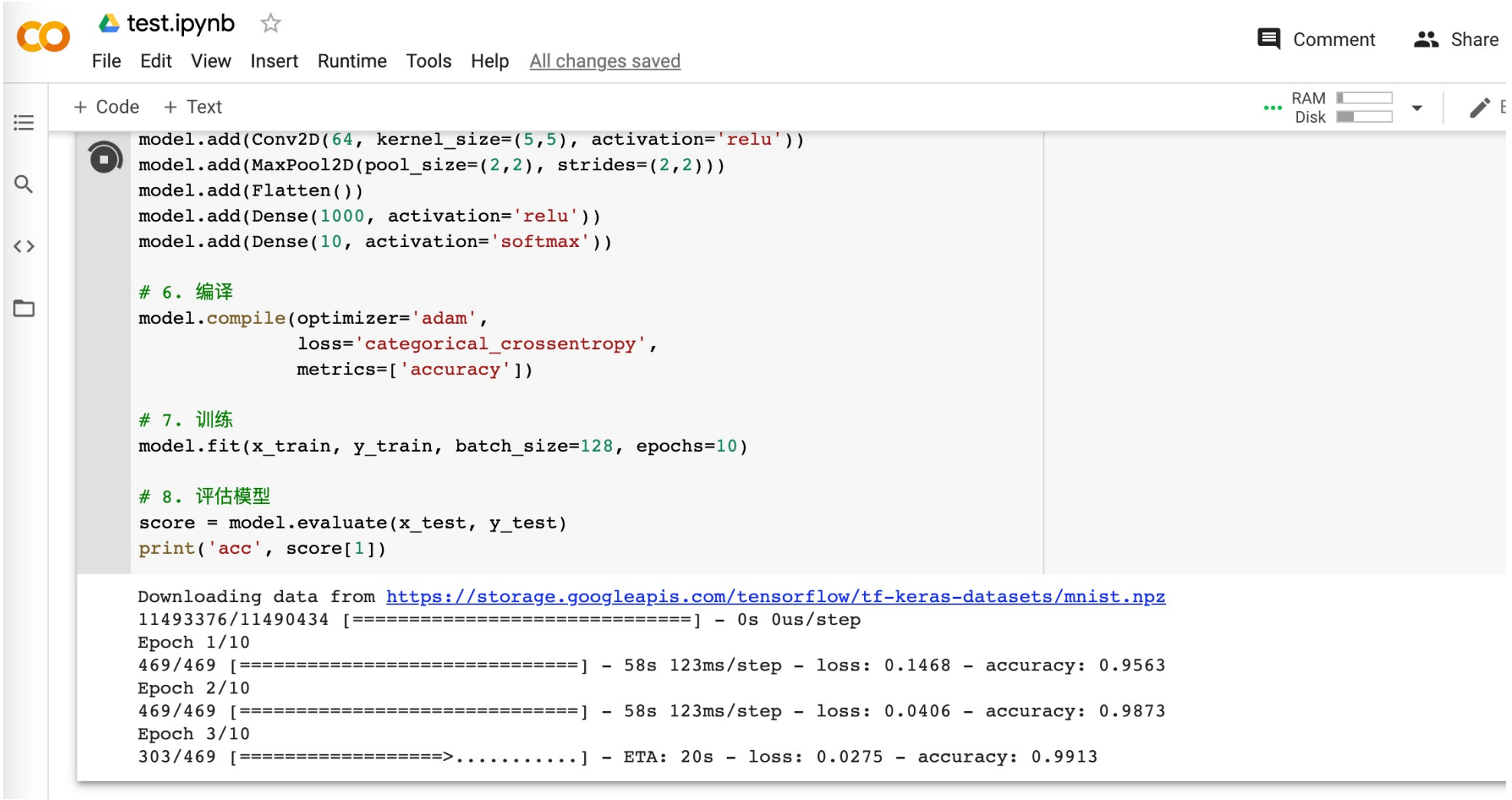
model.fit(x_train, y_train, batch_size=128, epochs=10)

The process of training on a personal computer



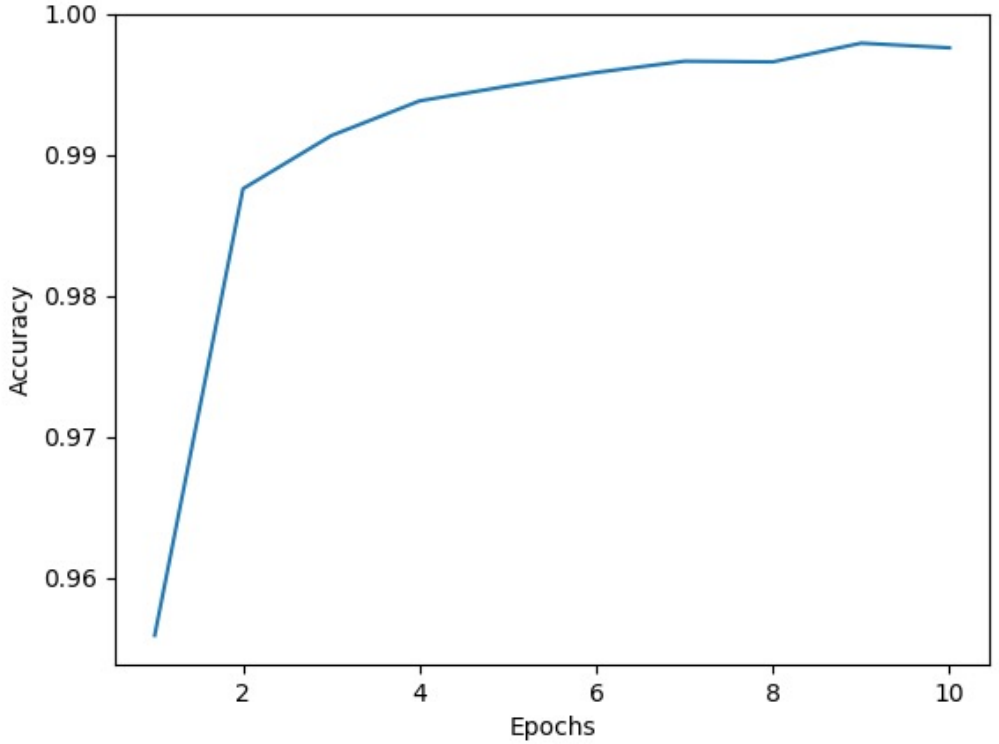
Step 6: Train

We can also train the model on colaboratory: colab.research.google.com



Step 6: Train

- After 10 epochs of training the above model, we achieve an accuracy of 99.2%. You can see the improvement in the accuracy for each epoch in the figure below:



Step 7: Evaluate the model

- Use the test set to evaluate the model model

```
score = model.evaluate(x_test, y_test)
print('acc', score[1])
# acc 0.9926
```

The accuracy rate is over 99%!

- Source code:

https://github.com/kevinsuo/CS7357/blob/master/cnn/cnn.py