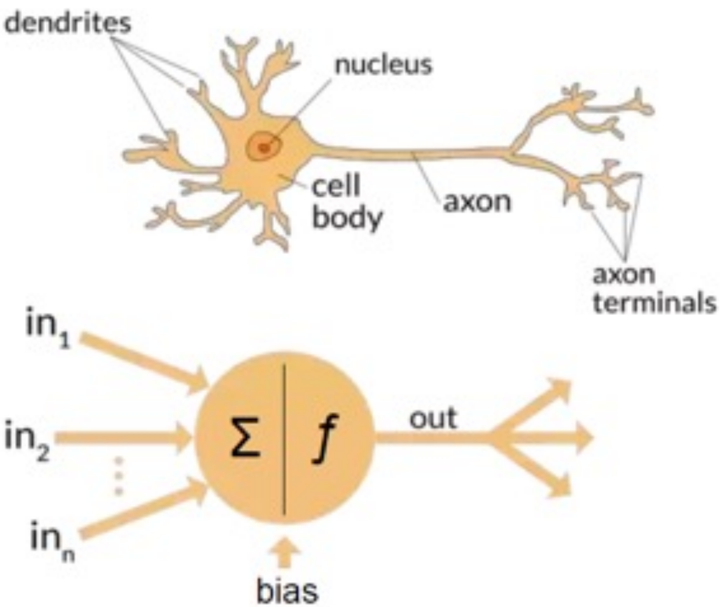
# Intro to Deep Learning

ECE30007 Intro to AI Project

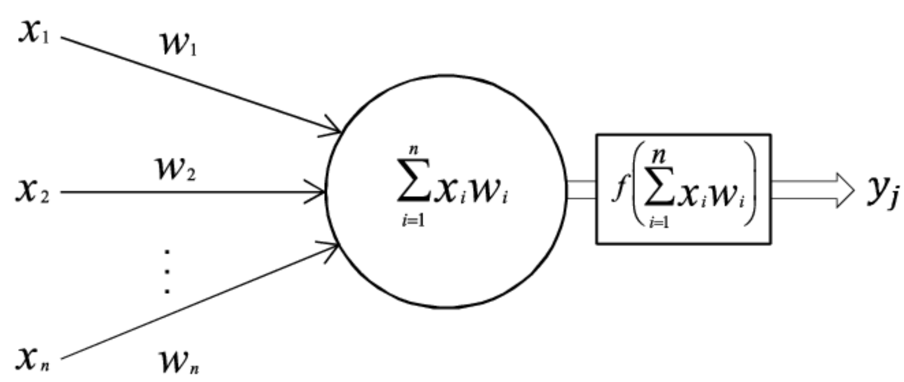
한동대학교
HANDONG GLOBAL UNIVERSITY

# Artificial Neural Network

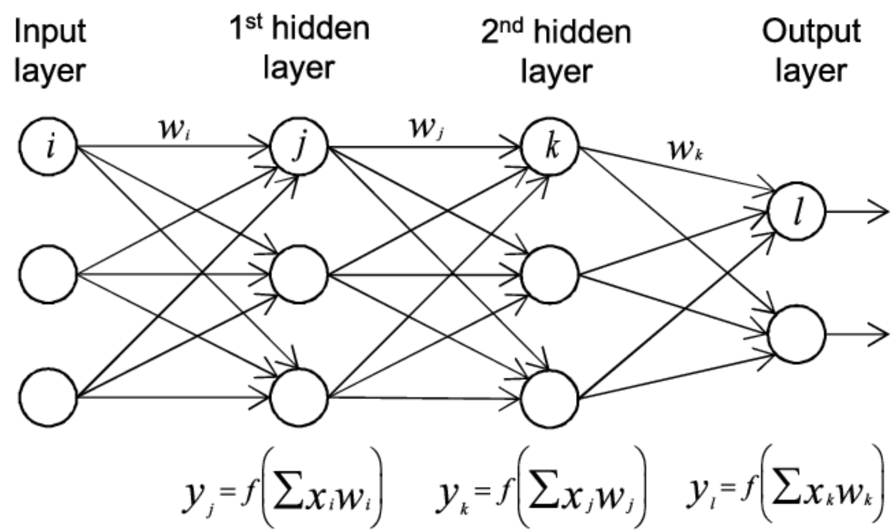



*Perceptron (Rosenblatt, 1957)*

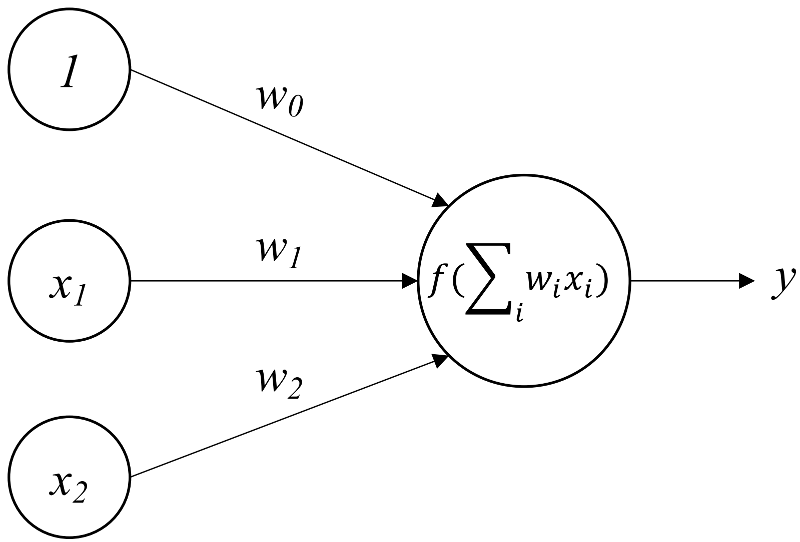
(a)

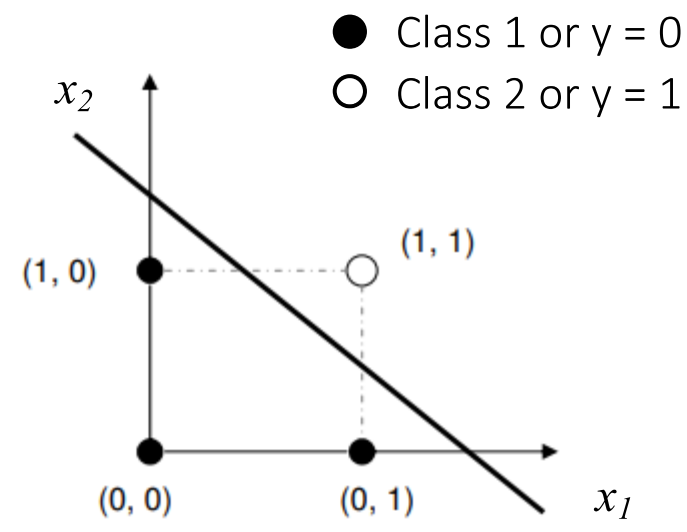



(b)



# Artificial Neural Network 1 (Perceptron)




with a simple example of *f*

$$f\left(\sum_i w_i x_i\right) = y$$

if $\sum_i w_i x_i > 0$ $\quad y = 1$

*otherwise* $\quad y = 0$

$$\sum_i w_i x_i = w_0 x_0 + w_1 x_1 + w_2 x_2$$

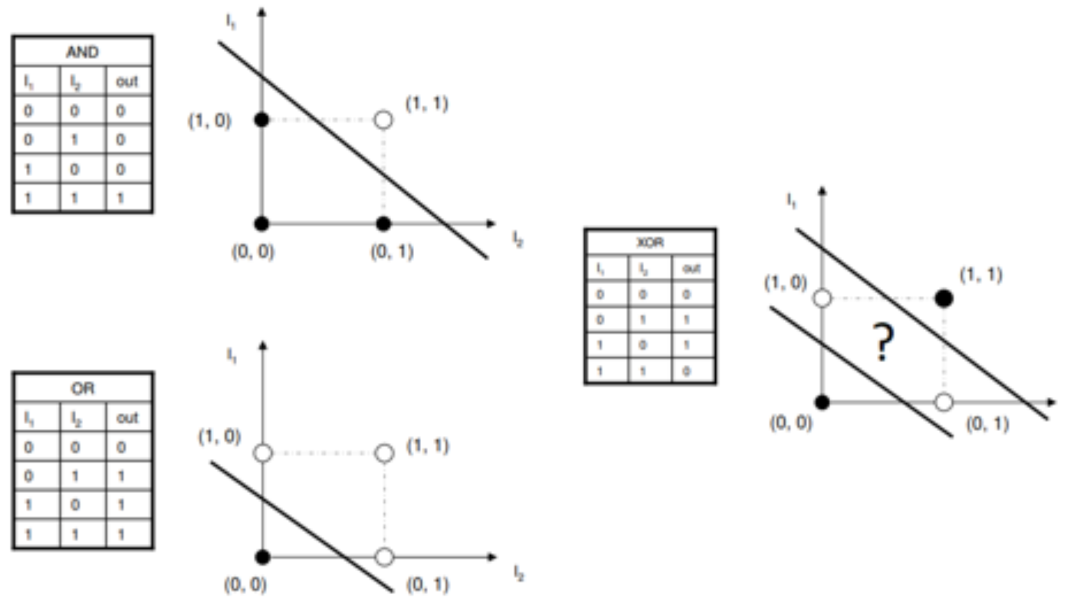
Let $w_0 = -1$ and $w_1 = w_2 = 1$

For input (1,1)
$\sum_i w_i x_i = -1\times 1 + 1\times 1 + 1\times 1 = 1$ ,
$f(\sum_i w_i x_i) = y = 1.$ (Class 1)

For input (0,1)
$\sum_i w_i x_i = -1\times 1 + 1\times 0 + 1\times 1 = 0$ ,
$f(\sum_i w_i x_i) = y = 0.$ (Class 2)

# Artificial Neural Network 2 (MLP)

Perceptron cannot solve
XOR problem
(or non-linear problem)

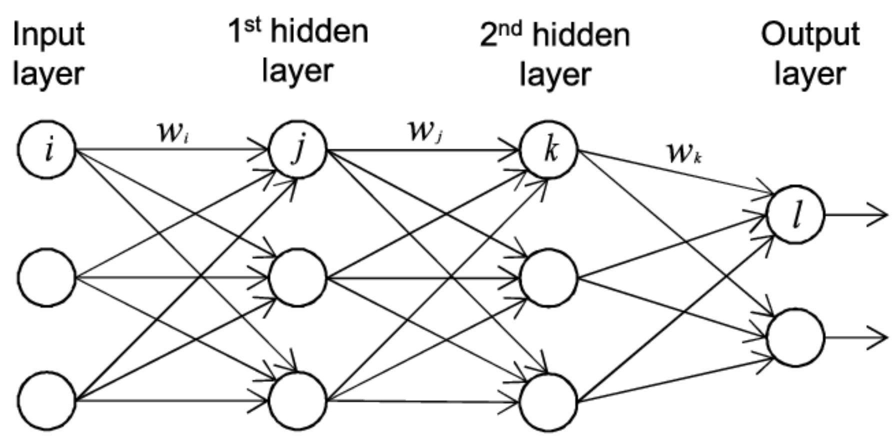



| AND | | |
|---|---|---|
| $I_1$ | $I_2$ | out |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| OR | | |
|---|---|---|
| $I_1$ | $I_2$ | out |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

| XOR | | |
|---|---|---|
| $I_1$ | $I_2$ | out |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

But Multi-layer network can
solve non-linear problem.




$y_j = f\left(\sum x_i w_i\right)$ $y_k = f\left(\sum x_j w_j\right)$ $y_l = f\left(\sum x_k w_k\right)$

# convolutional neural networks (CNNs)

- many practical applications
  - image recognition, speech recognition, Google's and Baidu's photo taggers
- won several competitions
  - ImageNet, Kaggle facial expression, Kaggle multimodal learning, German traffic signs, connectomics, handwriting, etc
- one of the few models that can be trained purely supervised

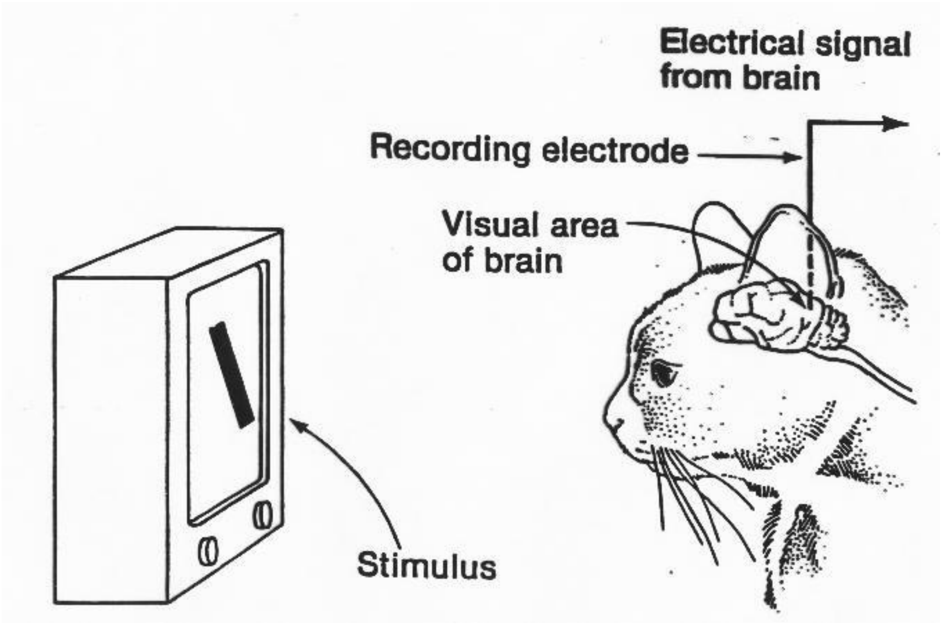
German traffic sign recognition competition

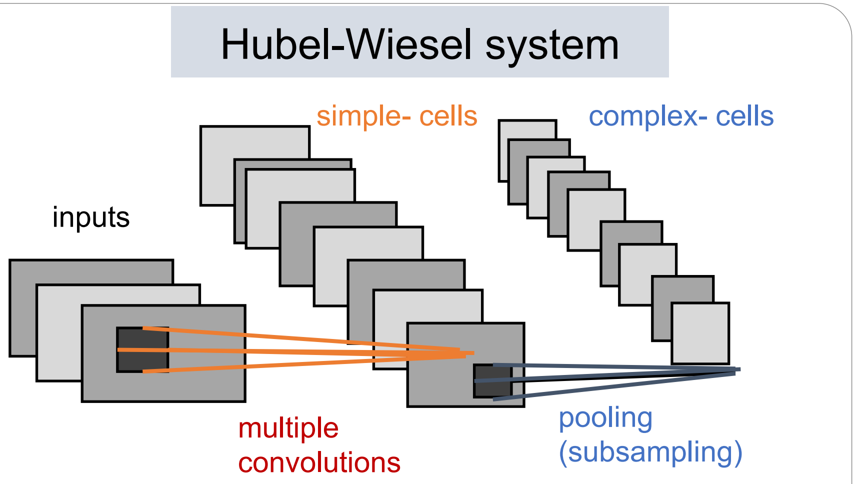
- single-image, multi-class
- more than 40 classes
- more than 50K images in total
- large, lifelike database

| rank | team | method | accuracy (%) |
|---|---|---|---|
| 1 | IDSIA | Committee of CNNs | 99.46 |
| 2 | INI | Human performance | 98.84 |
| 3 | Sermanet | Multi-scale CNNs | 98.31 |
| 4 | CAOR | Random Forests | 96.14 |

# from brain science

Hubel & Wiesel (1950s)

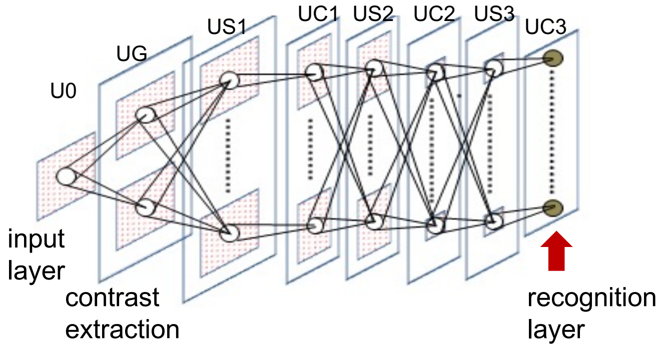


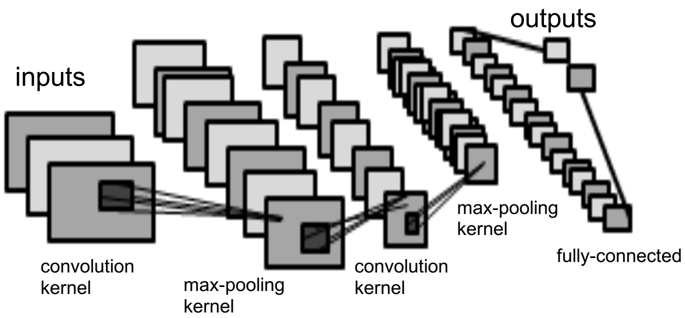

- simple cells detect local features.
- complex cells "pool" the outputs of simple cells within a retinotopic neighborhood.

# deep learning history

- **"Neocognitron"** by K. Fukushima, 1980 Biological Cybernetics

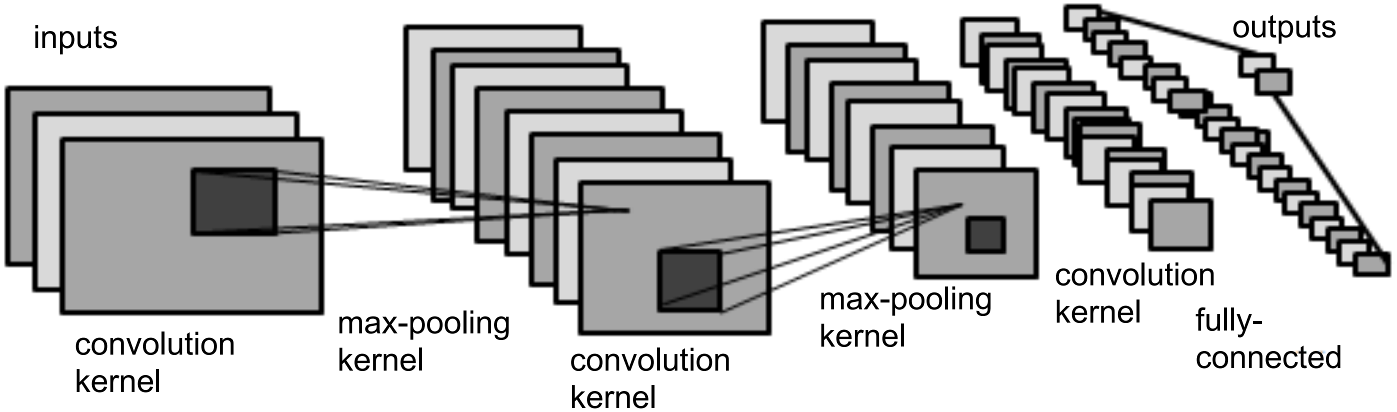



- **convolutional neural networks** by Y. LeCun et al., 1989 Neural Computation



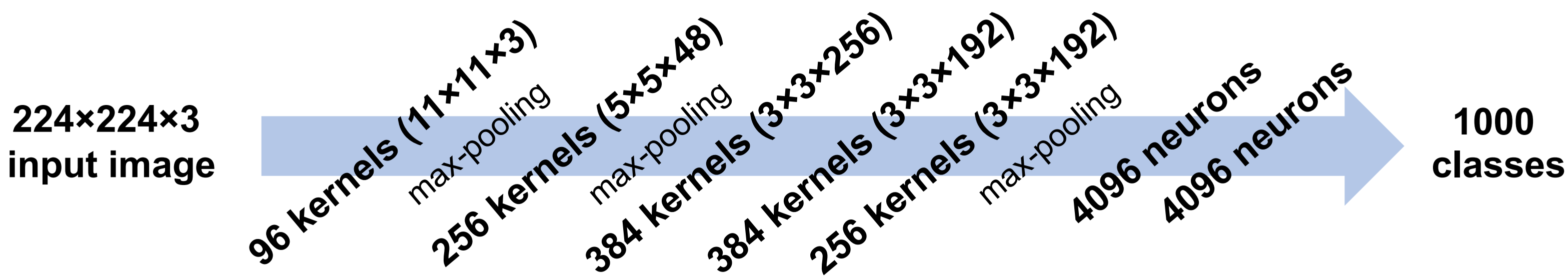

# examples of CNNs

## LeNet 1989



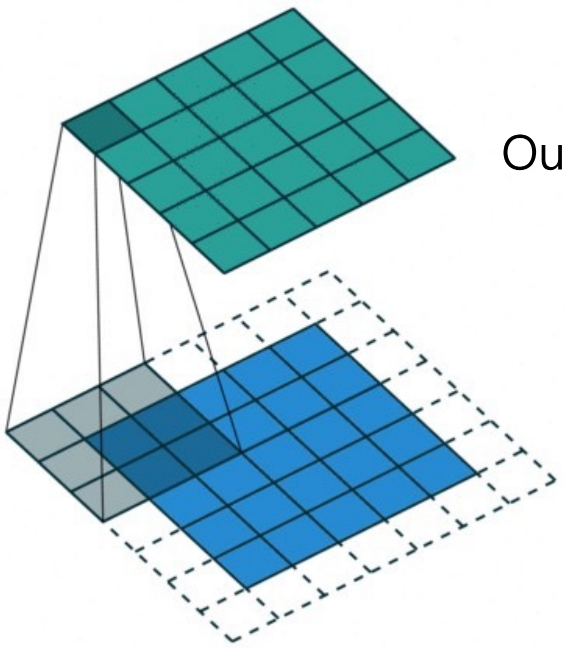

(LeCun 1989)

## AlexNet 2012




(krizhevsky 2012)

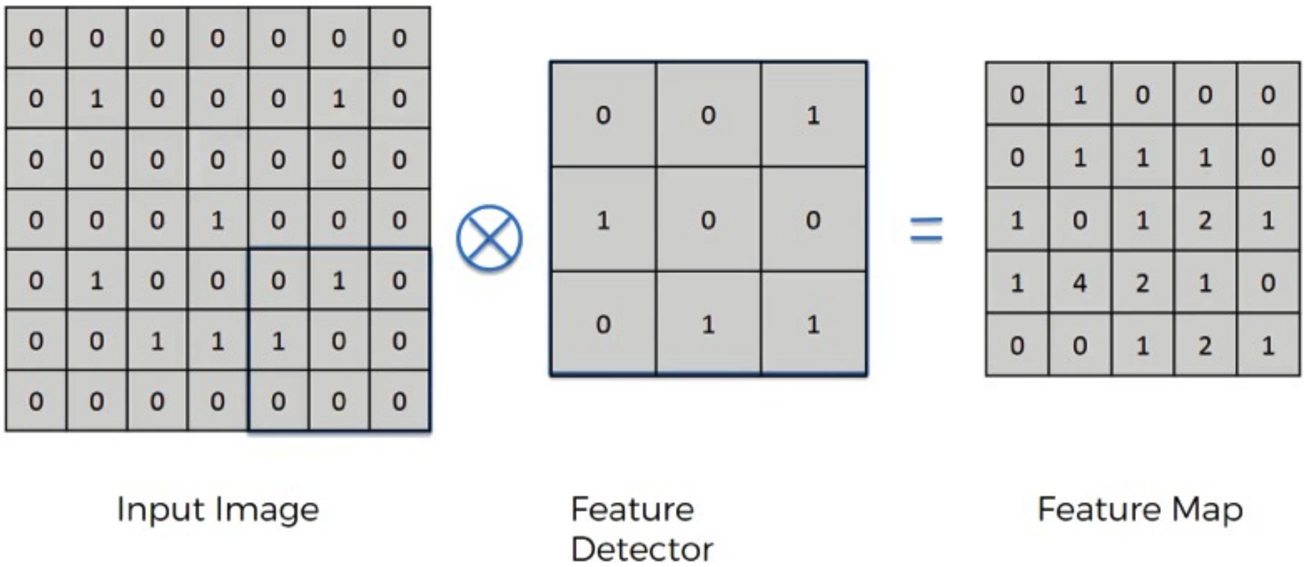
# convolution operation




1. Kernel(or filter)
2. Stride
3. Padding

(from https://github.com/vdumoulin/conv_arithmetic)

# convolution operation




from https://www.superdatascience.com/

# convolution operation

| | | |
|---|---|---|
| -1 | -1 | -1 |
| -1 | 8 | -1 |
| -1 | -1 | -1 |


outline

from http://setosa.io/ev/image-kernels/

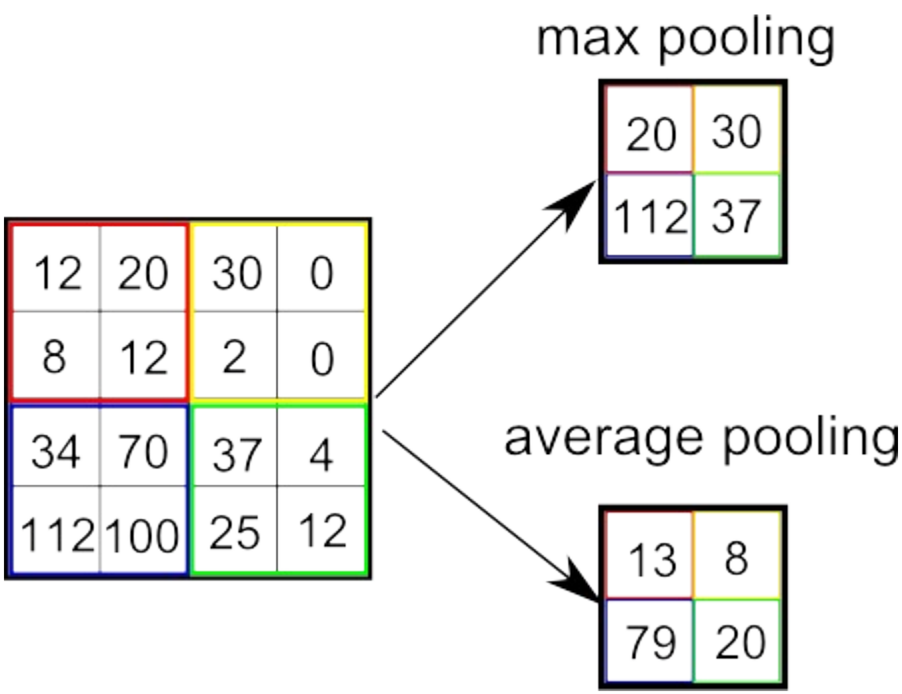
# pooling

- high resolution may not be necessary for a given task.


- sampling can reduce computation cost while keeping necessary information

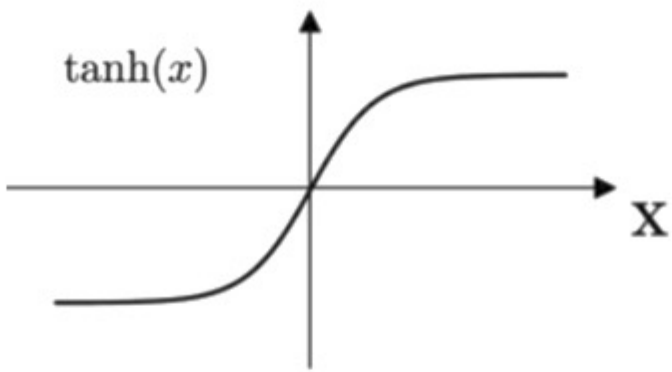
# pooling

- how to sample?

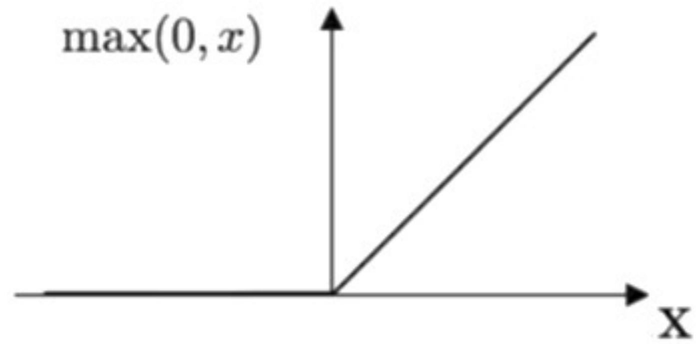


# activation function

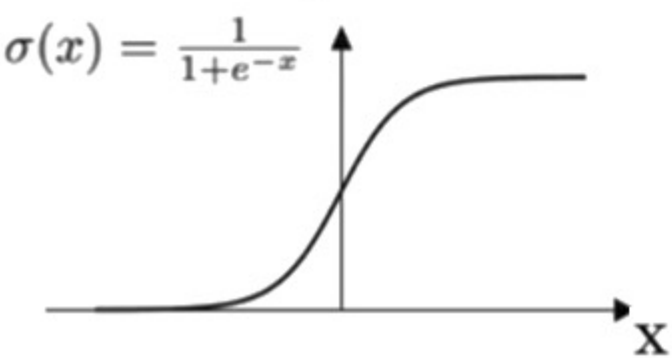
**Tanh**



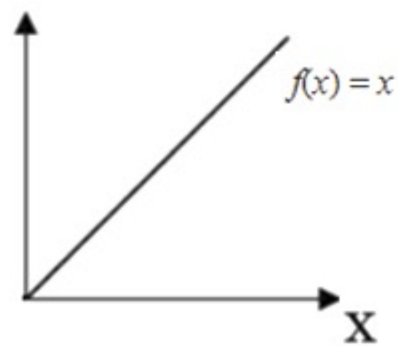

**ReLU**




**Sigmoid**




**Linear**






*https://docs.paperspace.com/machine-learning/wiki/activation-function*

# convolution layer

hyperparameters before training

**Convolution**

- Kernel
- Stride
- Padding

**Subsampling**

- Pooling type (Max or Average)
- Pooling size

**Activation Function**

- ReLU or Leaky ReLU

# softmax and loss function

- softmax: from output of neural network to probability

$$\mathrm{softmax}(\mathsf{y})_i = \frac{\exp(\mathsf{y}_i)}{\sum_j \exp(\mathsf{y}_j)}$$

- cross-entropy: loss function
  Given $p$ for ground truth, $q$ for predicted value from softmax.

$$H(p,q) = -\sum_x p(x)\,\log q(x).$$

# ImageNet Challenge

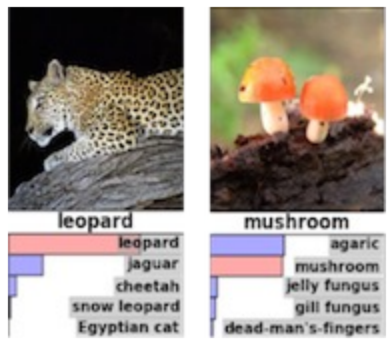
**1.2 Million Images(1,200,000), 1000 Categories**

# the first deep learning on ImageNet data: AlexNet

## ImageNet

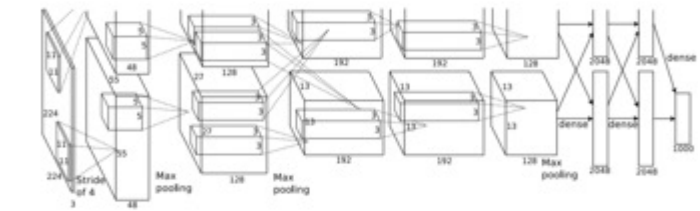
- ImageNet data set

Recognition task (Top-5)



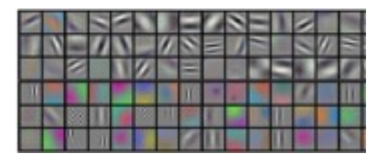

## network structure

253440–186624–64896–64896–43264–4096–4096–1000



- 8 layers: 650K neurons, 60M parameters

- Trained on 2 GPUs
- 5-6 days of training (90 iterations)



Trained filters

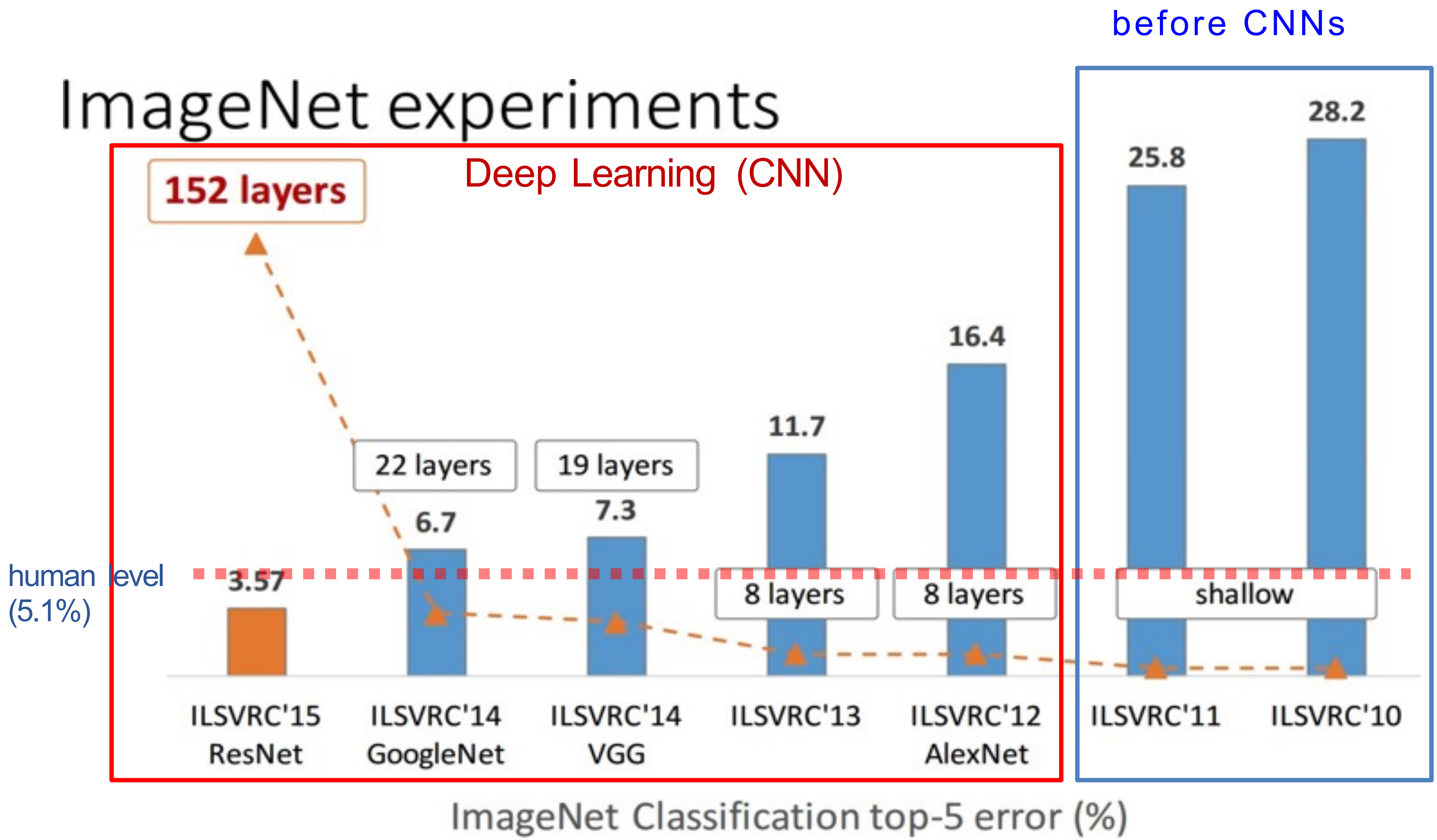
- 1000/10,184 categories
- 1.2M/8.9M training images
- 50K validation
- 150K test

(Krizhevsky et al 2012)

results on ILSVRC-2010/ImageNet2009 (error %)

| | previous SOTA | deep learning (CNNs) |
|---|---|---|
| top-1 | 45.7 / 78.1 | 37.5 / **67.4** |
| top-5 | 25.7 / 60.9 | 17.0 / **40.9** |

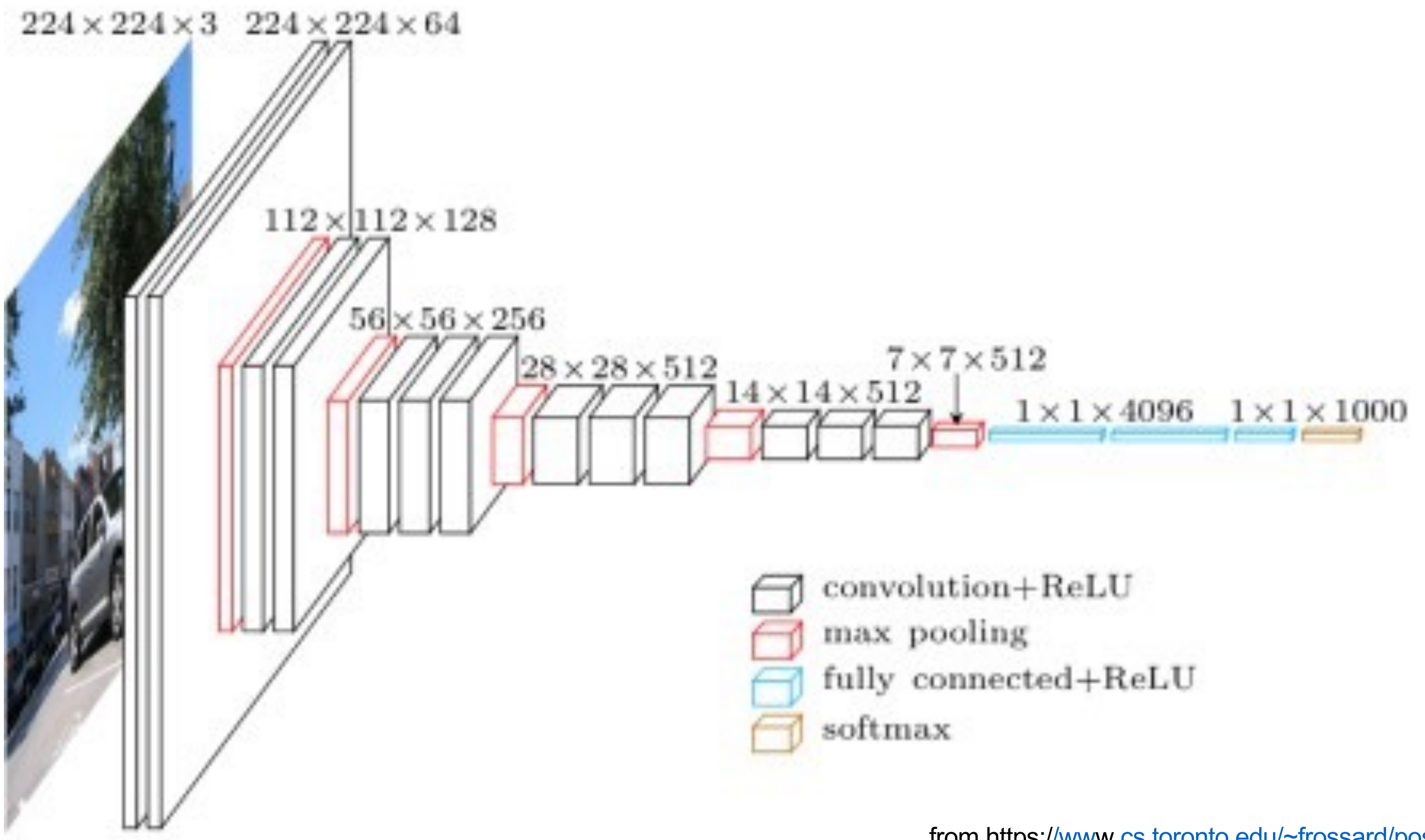
# improvement after AlexNet




from Kaiming He "Deep residual learning for image recognition." ICML. 2016.

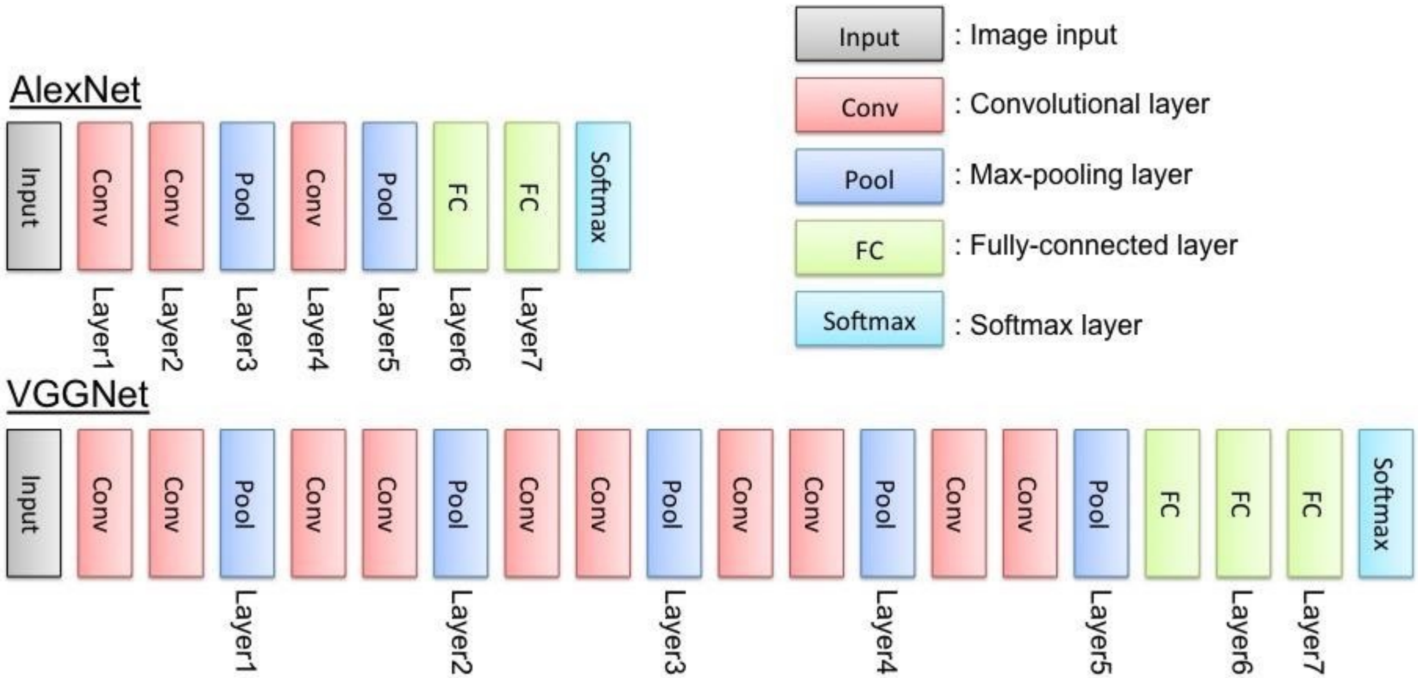
# VGG Network

- simple architecture and very popular.




from https://www.cs.toronto.edu/~frossard/post/vgg16/

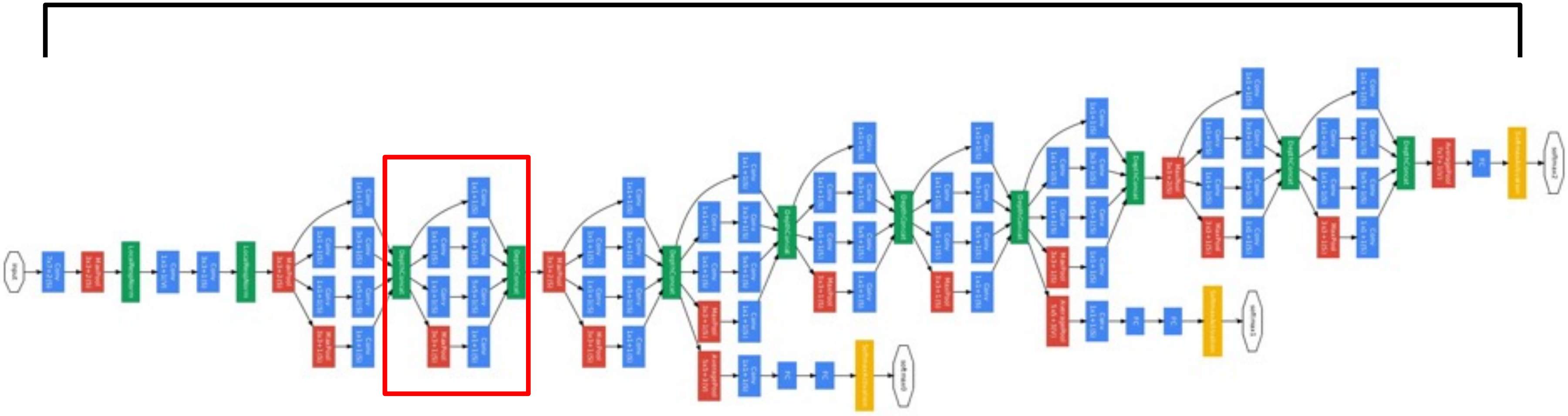
# VGG Network




from http://www.hirokatsukataoka.net/research/cnnfeatureevaluation/cnnfeatureevaluation.html

# Google Network

22 Layer



from https://arxiv.org/pdf/1409.4842.pdf

# PyTorch

PYTORCH

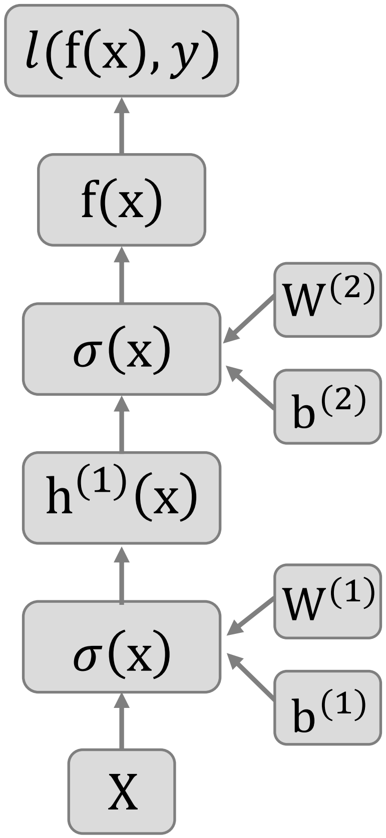
You may need to install

>> pip install torch
>> pip install torchvision
>> pip install scikit-image

- Python-based scientific computing package
  - to use the power of GPUs instead of Numpy
  - to provide maximum flexibility and speed for deep learning

```
import torch
x = torch.Tensor(5,3)
y = torch.Tensor(5,3)
print(x, y)
```

# Autograd




- each object has an fprop/bprop method,
- forward propagation:
  - calling fprop of each box in the right order
- backpropagation:
  - calling bprop in the reverse order

- **a large portion of deep learning research is based on Theano, PyTorch or TensorFlow**

# torch.nn

## Torch.nn?

Neural Network Module.

Easy to make neural network
such as Linear, CNN, RNN, and so on…

```
class NetFF(nn.Module):
    def __init__(self):
        super(NetFF, self).__init__()
        self.fc1 = nn.Linear(784, 500)
        self.fc2 = nn.Linear(500, 300)
        self.fc3 = nn.Linear(300, 100)
        self.fc4 = nn.Linear(100, 10)

    def forward(self, x):
        x = x.view(-1, 784)
        x = torch.tanh(self.fc1(x))
        x = torch.tanh(self.fc2(x))
        x = torch.tanh(self.fc3(x))
        x = self.fc4(x)
        return F.log_softmax(x, dim=1)
```

# loss and optimizer

## Loss function

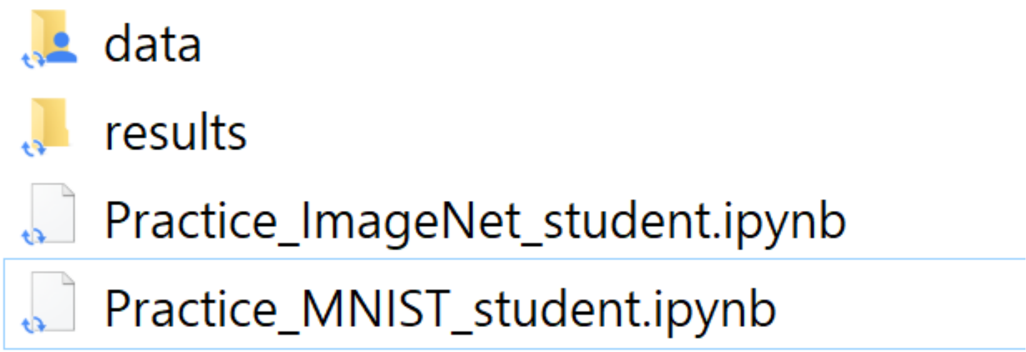
How to define the loss? (MSE, RMSE, CrossEntropy, L1, NLL, etc..)

## Optimizer

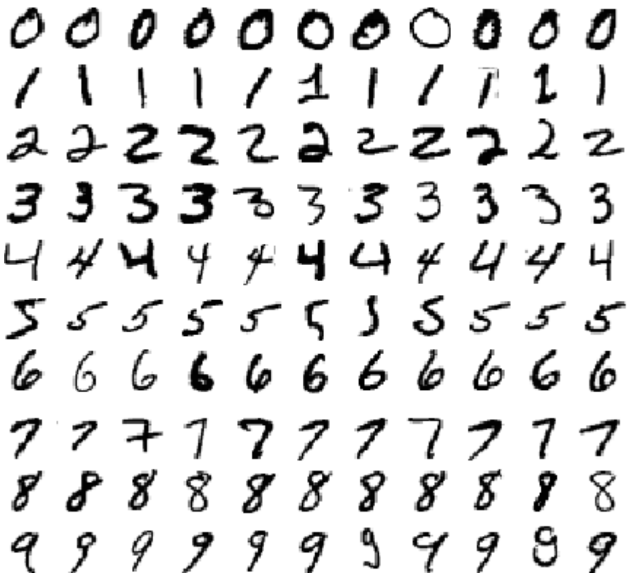
How to update weights ? (SGD, Adam, RMSProp, AdaDelta, etc…)

# Practice

- Please check that
  you have two folders / two files in your current directory.



# Practice – MNIST



**MNIST dataset [28X28] = [1 X 784]**

**Hand Written Digit Number from 0 to 9**
**Data includes data and label.**

**Pytorch Dataset(torchvision) provides**
**50,000 images to train,**
**10,000 images to test.**

# Packages

```
from __future__ import print_function
import argparse
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
```

# Models

```python
class NetFF(nn.Module):
    def __init__(self):
        super(NetFF, self).__init__()
        self.fc1 = nn.Linear(784, 500)
        self.fc2 = nn.Linear(500, 300)
        self.fc3 = nn.Linear(300, 100)
        self.fc4 = nn.Linear(100, 10)

    def forward(self, x):
        x = x.view(-1, 784)
        x = torch.tanh(self.fc1(x))
        x = torch.tanh(self.fc2(x))
        x = torch.tanh(self.fc3(x))
        x = self.fc4(x)
        return F.log_softmax(x, dim=1)
```

```python
class NetCNN(nn.Module):
    def __init__(self):
        super(NetCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5, 1)
        self.conv2 = nn.Conv2d(20, 50, 5, 1)
        self.fc1 = nn.Linear(4*4*50, 500)
        self.fc2 = nn.Linear(500, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 2, 2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2, 2)
        x = x.view(-1, 4*4*50)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)
```

# train() and test() functions

```
def train(model, device, train_loader, optimizer, epoch, log_interval):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()

        if batch_idx % log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]₩tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))

            torch.save(model.state_dict(),"./results/mnist_cnn.pt")


def test(model, device, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)

            # sum up batch loss
            test_loss += F.nll_loss(output, target, reduction='sum').item()

            # get the index of the max log-probability
            pred = output.argmax(dim=1, keepdim=True)
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)

    print('₩nTest: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)₩n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))
```

# Parameters and Data load

```
seed = 1
epochs = 2
batch_size = 32
test_batch_size = 1000
lr = 0.001 # learning rate
momentum = 0.9
log_interval = 200
save_model = True

use_cuda = torch.cuda.is_available()
device = torch.device("cuda" if use_cuda else "cpu")
kwargs = {'num_workers': 1, 'pin_memory': True} if use_cuda else {}

transform=transforms.Compose([
   transforms.ToTensor(),
   transforms.Normalize((0.1307,), (0.3081,)) ])

train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('./data', train=True, download=True, transform=transform),
    batch_size=batch_size, shuffle=True, **kwargs)

test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('./data', train=False, transform=transform),
    batch_size=test_batch_size, shuffle=True, **kwargs)
```

# NetFF Model training/testing

```python
import os # if kernel dies, these two lines fix the problem.
os.environ['KMP_DUPLICATE_LIB_OK']='True'
```

```python
torch.manual_seed(seed)

model = NetFF().to(device)
optimizer = optim.SGD(model.parameters(), lr=lr, momentum=momentum)

for epoch in range(1, epochs + 1):
    train(model, device, train_loader, optimizer, epoch, log_interval)
    test(model, device, test_loader)

if (save_model):
    torch.save(model,"./results/mnist_NetFF.pth")
```

```
Train Epoch: 1 [0/60000 (0%)]	Loss: 2.339498
Train Epoch: 1 [6400/60000 (11%)]	Loss: 1.123757
Train Epoch: 1 [12800/60000 (21%)]	Loss: 0.737093
Train Epoch: 1 [19200/60000 (32%)]	Loss: 0.440414
Train Epoch: 1 [25600/60000 (43%)]	Loss: 0.489146
Train Epoch: 1 [32000/60000 (53%)]	Loss: 0.544215
Train Epoch: 1 [38400/60000 (64%)]	Loss: 0.437548
Train Epoch: 1 [44800/60000 (75%)]	Loss: 0.231311
Train Epoch: 1 [51200/60000 (85%)]	Loss: 0.271293
Train Epoch: 1 [57600/60000 (96%)]	Loss: 0.363241

Test: Average loss: 0.2999, Accuracy: 9168/10000 (92%)

Train Epoch: 2 [0/60000 (0%)]	Loss: 0.520948
Train Epoch: 2 [6400/60000 (11%)]	Loss: 0.113256
Train Epoch: 2 [12800/60000 (21%)]	Loss: 0.188349
Train Epoch: 2 [19200/60000 (32%)]	Loss: 0.412683
Train Epoch: 2 [25600/60000 (43%)]	Loss: 0.328352
Train Epoch: 2 [32000/60000 (53%)]	Loss: 0.235215
Train Epoch: 2 [38400/60000 (64%)]	Loss: 0.130424
Train Epoch: 2 [44800/60000 (75%)]	Loss: 0.371435
Train Epoch: 2 [51200/60000 (85%)]	Loss: 0.135371
```

# NetCNN Model training/testing

```python
torch.manual_seed(seed)

model = NetCNN().to(device)
optimizer = optim.SGD(model.parameters(), lr=lr, momentum=momentum)

for epoch in range(1, epochs + 1):
    train(model, device, train_loader, optimizer, epoch, log_interval)
    test(model, device, test_loader)

if (save_model):
    torch.save(model,"./results/mnist_NetCNN.pth")
```

```
Train Epoch: 1 [0/60000 (0%)]	Loss: 2.294408
Train Epoch: 1 [6400/60000 (11%)]	Loss: 0.469601
Train Epoch: 1 [12800/60000 (21%)]	Loss: 0.433760
Train Epoch: 1 [19200/60000 (32%)]	Loss: 0.331650
Train Epoch: 1 [25600/60000 (43%)]	Loss: 0.225021
Train Epoch: 1 [32000/60000 (53%)]	Loss: 0.367144
Train Epoch: 1 [38400/60000 (64%)]	Loss: 0.108309
Train Epoch: 1 [44800/60000 (75%)]	Loss: 0.031551
Train Epoch: 1 [51200/60000 (85%)]	Loss: 0.080511
Train Epoch: 1 [57600/60000 (96%)]	Loss: 0.229363

Test: Average loss: 0.1017, Accuracy: 9694/10000 (97%)

Train Epoch: 2 [0/60000 (0%)]	Loss: 0.108063
Train Epoch: 2 [6400/60000 (11%)]	Loss: 0.101820
Train Epoch: 2 [12800/60000 (21%)]	Loss: 0.033056
Train Epoch: 2 [19200/60000 (32%)]	Loss: 0.054642
Train Epoch: 2 [25600/60000 (43%)]	Loss: 0.037946
Train Epoch: 2 [32000/60000 (53%)]	Loss: 0.059596
Train Epoch: 2 [38400/60000 (64%)]	Loss: 0.007043
Train Epoch: 2 [44800/60000 (75%)]	Loss: 0.112242
Train Epoch: 2 [51200/60000 (85%)]	Loss: 0.052512
```

# Testing with one test image

trained model

```
load_model = torch.load("./results/mnist_NetCNN.pth")
```

```
from skimage import io # if skimage is not availabe, !pip install scikit-image

img_name = './data/mnist_test_images/test0.jpg'
test_img = io.imread(img_name).reshape(28,28)
test_data = transform(test_img).view(1,1,28,28).to(device)
with torch.no_grad():
  output=load_model(test_data)
print(img_name, output.argmax(dim=1).cpu().numpy()[0])
```

```
./data/mnist_test_images/test0.jpg 5
```

# Testing with multiple test image

```
from skimage import io
import time
import glob

file_list = glob.glob("./data/mnist_test_images/*.jpg")
for img_name in file_list:
  test_img = io.imread(img_name).reshape(28,28)
  test_data = transform(test_img).view(1,1,28,28).to(device)
  with torch.no_grad():
    output=load_model(test_data)
  print(img_name, output.argmax(dim=1).cpu().numpy()[0])
```

```
./data/mnist_test_images/test15.jpg 7
./data/mnist_test_images/test14.jpg 1
./data/mnist_test_images/test16.jpg 2
./data/mnist_test_images/test17.jpg 8
./data/mnist_test_images/test13.jpg 6
./data/mnist_test_images/test12.jpg 3
./data/mnist_test_images/test10.jpg 3
./data/mnist_test_images/test11.jpg 5
./data/mnist_test_images/test6.jpg 1
./data/mnist_test_images/test7.jpg 3
./data/mnist_test_images/test5.jpg 2
./data/mnist_test_images/test4.jpg 9
./data/mnist_test_images/test0.jpg 5
./data/mnist_test_images/test1.jpg 0
./data/mnist_test_images/test3.jpg 1
./data/mnist_test_images/test2.jpg 4
./data/mnist_test_images/test9.jpg 4
./data/mnist_test_images/test8.jpg 1
./data/mnist_test_images/test20.jpg 4
./data/mnist_test_images/test19.jpg 9
./data/mnist_test_images/test18.jpg 6
```

# ImageNet Example

- it takes too much time to train.
- so, we will just test with a pretrained model

# configuration and loading model

## Installing pretrainedmodels

```
!pip install pretrainedmodels
```

## Model download and preparation

1. Model will be downloaded at the designated directory. (Takes 10~20 min.)
2. Transform functions will be used to transform input.

- But let's use the provided model(in this course) withough downloading from the server.
  * Please do not run this code for now.

```
import torch
import pretrainedmodels
import pretrainedmodels.utils as utils

model_name = 'nasnetalarge' # could be fbresnet152 or inceptionresnetv2
model = pretrainedmodels.__dict__[model_name](num_classes=1000, pretrained='imagenet')
model.eval()

load_img = utils.LoadImage()

# transformations depending on the model
# rescale, center crop, normalize, and others (ex: ToBGR, ToRange255)
tf_img = utils.TransformImage(model)

# save the model for later
torch.save(model, './results/imagenet_nasnetalarge.pth')
```

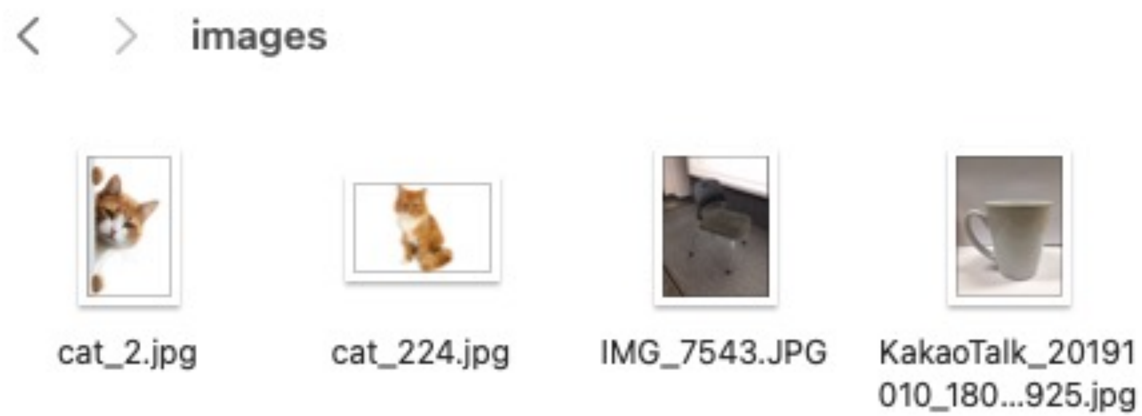
# ImageNet class names

```python
import csv
name_file = './data/imagenet_install/class_names.csv'

imagenet_class  = {}
file_in = csv.reader(open(name_file))
for row in file_in:
  imagenet_class[int(row[0])] = row[1]

imagenet_class
```

```
{0: 'tench, Tinca tinca',
 1: 'goldfish, Carassius auratus',
 2: 'great white shark, white shark, man-eater, man-eating shark, Carcharodon carcharias',
 3: 'tiger shark, Galeocerdo cuvieri',
 4: 'hammerhead, hammerhead shark',
 5: 'electric ray, crampfish, numbfish, torpedo',
 6: 'stingray',
 7: 'cock',
 8: 'hen',
 9: 'ostrich, Struthio camelus',
 10: 'brambling, Fringilla montifringilla',
 11: 'goldfinch, Carduelis carduelis',
 12: 'house finch, linnet, Carpodacus mexicanus',
 13: 'junco, snowbird',
 14: 'indigo bunting, indigo finch, indigo bird, Passerina cyanea',
 15: 'robin, American robin, Turdus migratorius',
 16: 'bulbul',
 17: 'jay',
 18: 'magpie',
```

# classification with images

in the directory ‘./data/images/’



# classification of a single image

```python
import os # if kernel dies, these two lines fix the problem.
os.environ['KMP_DUPLICATE_LIB_OK']='True'
```

```python
import torch

try:
  model # does exist
except NameError: # model does not exist
  import pretrainedmodels.utils as utils
  model = torch.load('./results/imagenet_nasnetalarge.pth')
  load_img = utils.LoadImage()
  tf_img = utils.TransformImage(model)

# your file name
img_file = './data/images/cat_224.jpg'

input_img = load_img(img_file)
input_tensor1 = tf_img(input_img)
input_tensor2 = input_tensor1.unsqueeze(0)

output_logits = model(input_tensor2) # 1x1000

print("{} is [{}: {}]".format(img_file ,output_logits.argmax(),
                              imagenet_class[int(output_logits.argmax())]))
```

```
./data/images/cat_224.jpg is [281: tabby, tabby cat]
```

# classification of many images in a directory

```
try:
  model # does exist
except NameError: # model does not exist
   import pretrainedmodels.utils as utils
   model = torch.load('./results/imagenet_nasnetalarge.pth')
   load_img = utils.LoadImage()
   tf_img = utils.TransformImage(model)

import glob
dir_path = './data/images/'
img_list = glob.glob(dir_path+'*.*')

for img_file in img_list:
   input_img = load_img(img_file)
   input_tensor1 = tf_img(input_img)
   input_tensor2 = input_tensor1.unsqueeze(0)
   output_logits = model(input_tensor2) # 1x1000

   print("{} is [{}: {}]".format(img_file.split('/')[-1] ,output_logits.argmax(),
                                 imagenet_class[int(output_logits.argmax())]))
```

```
images\cat_2.jpg is [282: tiger cat]
images\cat_224.jpg is [281: tabby, tabby cat]
images\IMG_7543.JPG is [559: folding chair]
images\KakaoTalk_20191010_180310925.jpg is [504: coffee mug]
```

# Practice: classify your own images

- take photos
- put the photos in the ‘./data/images’ directory
- classify them