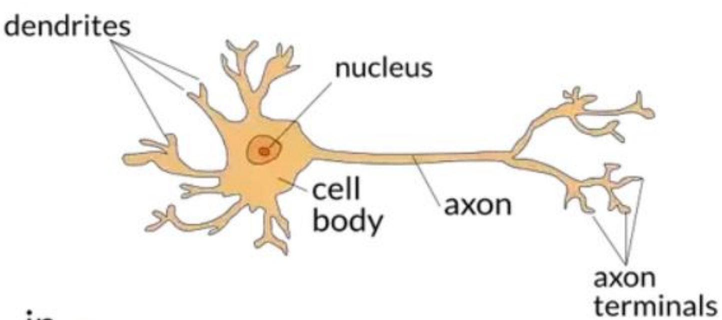
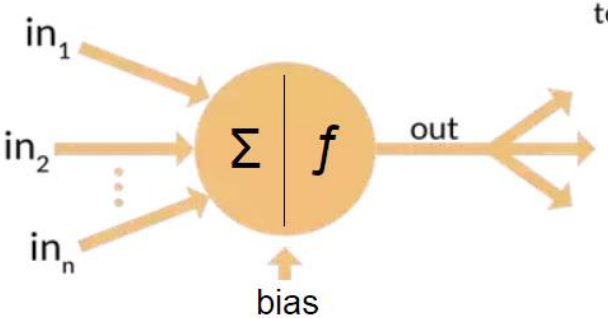
Intro to Deep Learning

ECE30007 Intro to AI Project

한동대학교
HANDONG GLOBAL UNIVERSITY

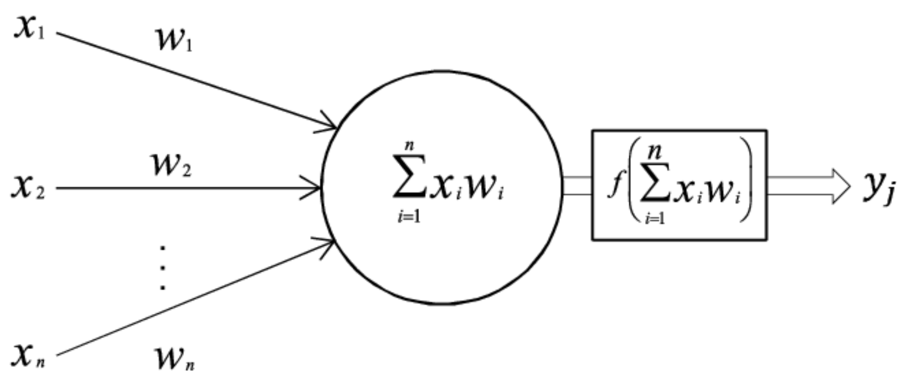
Artificial Neural Network



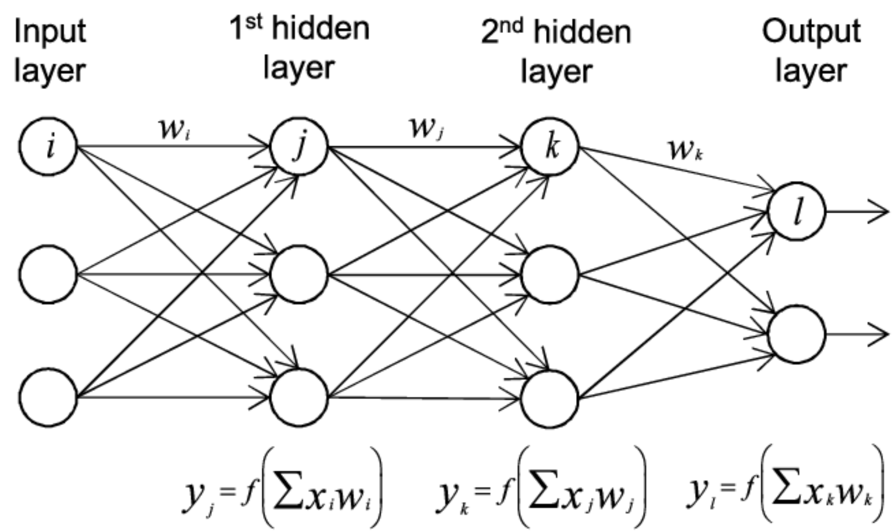


Perceptron (Rosenblatt, 1957)

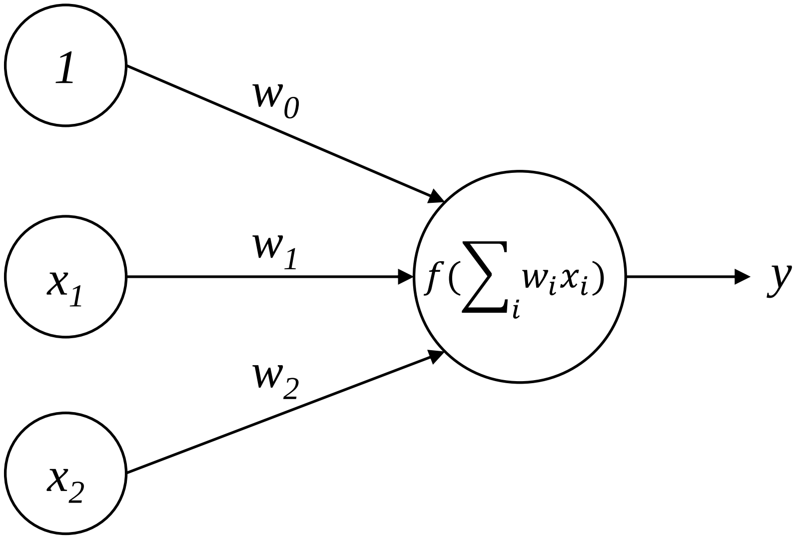
(a)

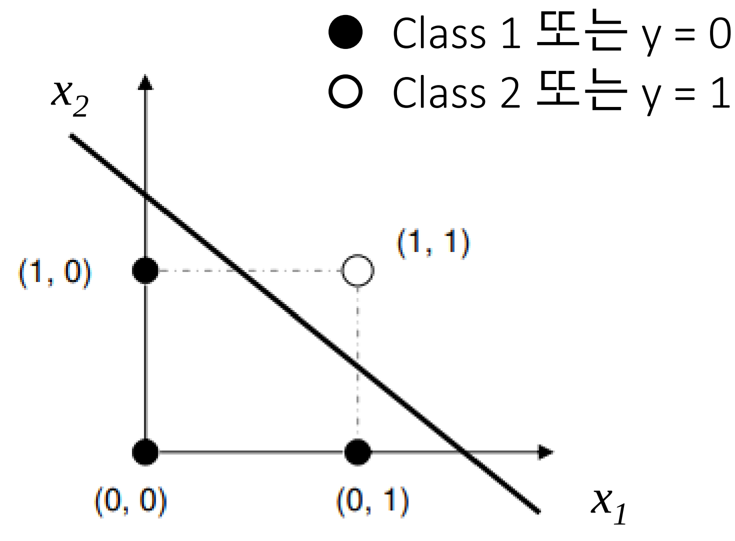


(b)



Artificial Neural Network





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

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

Otherwise $y = 0$

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

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

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)

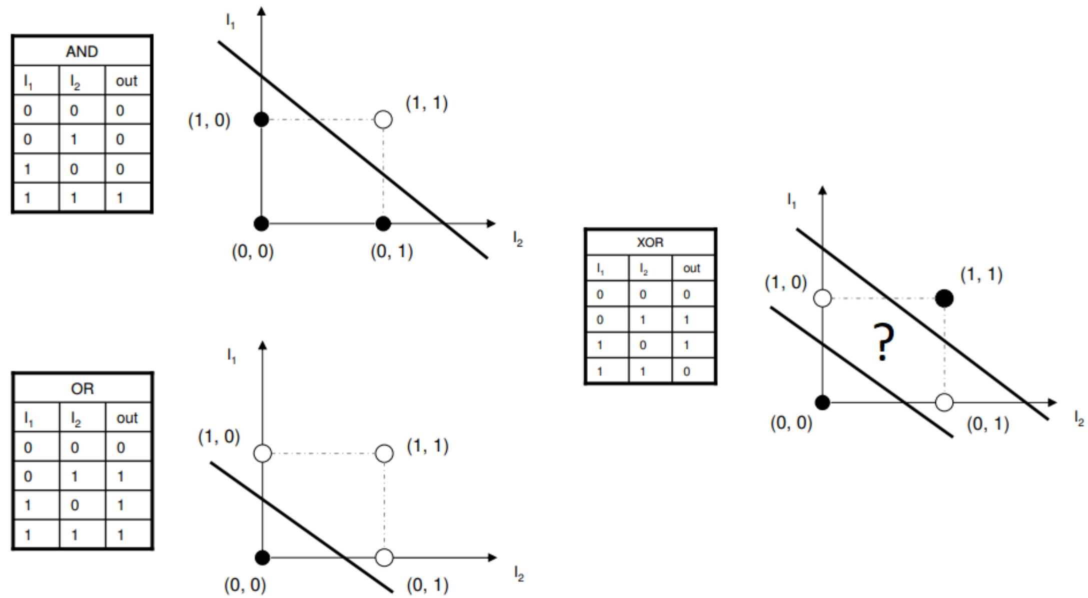
For input (0,1)

$\sum_i w_i x_i = -1 \times 1 + 1 \times 0 + 1 \times 1 = 0$,

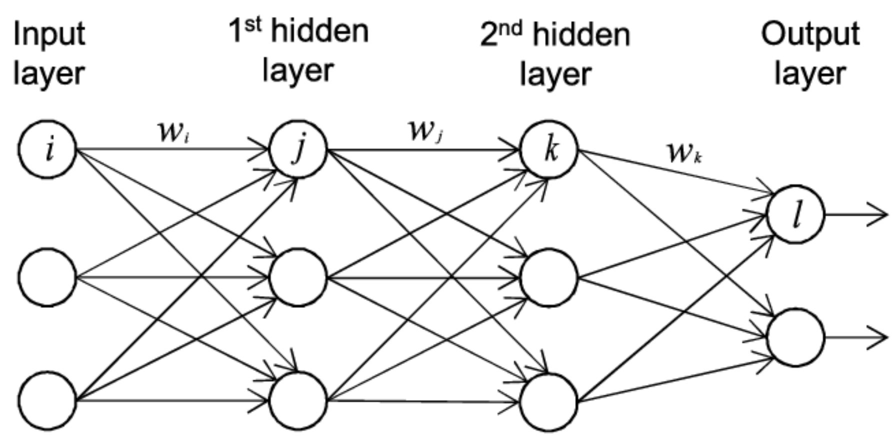
$f(\sum_i w_i x_i) = y = 0$. (Class 2)

Artificial Neural Network

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



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



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

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

image processing

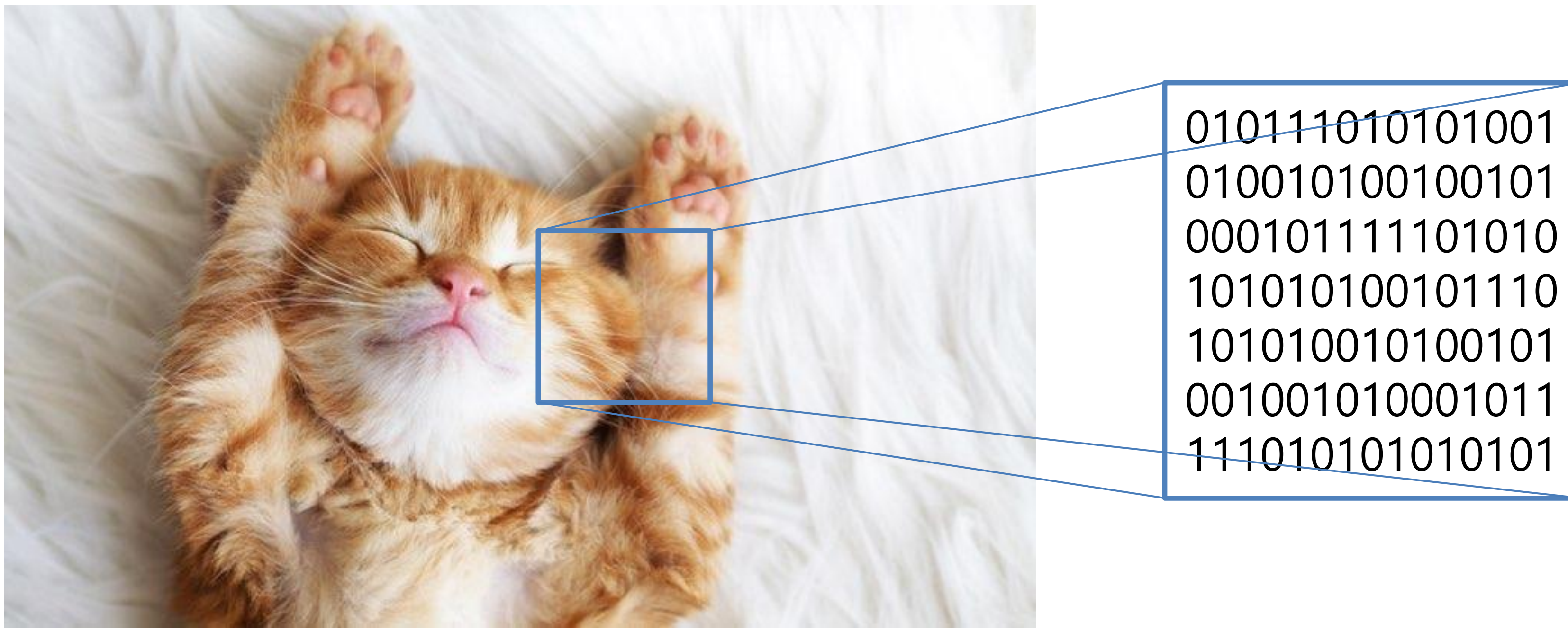
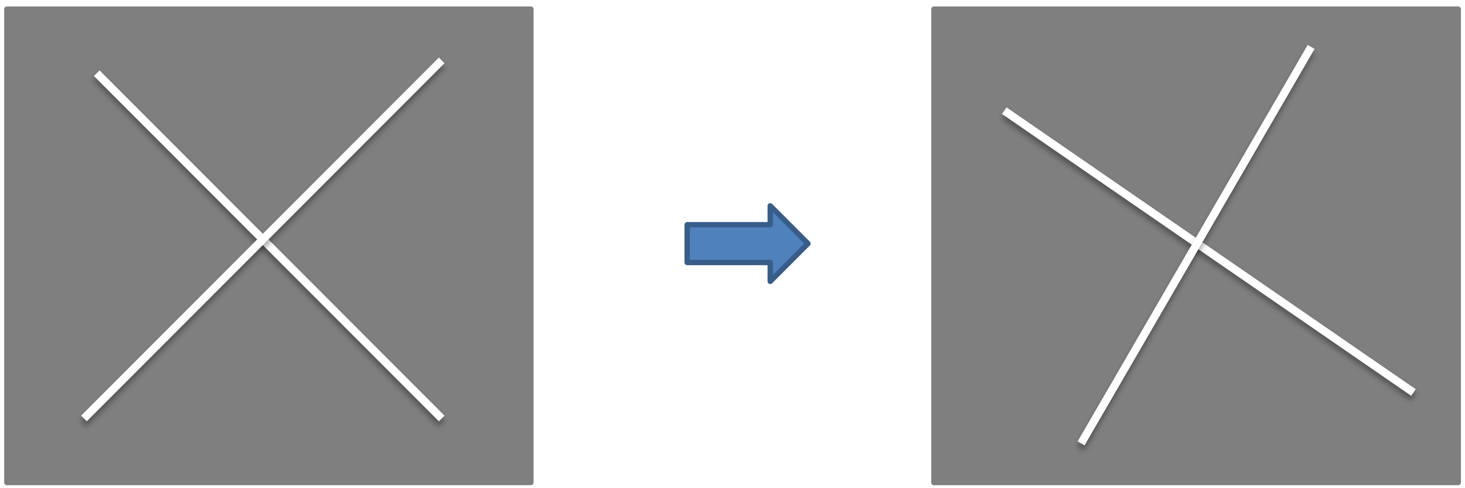


image processing



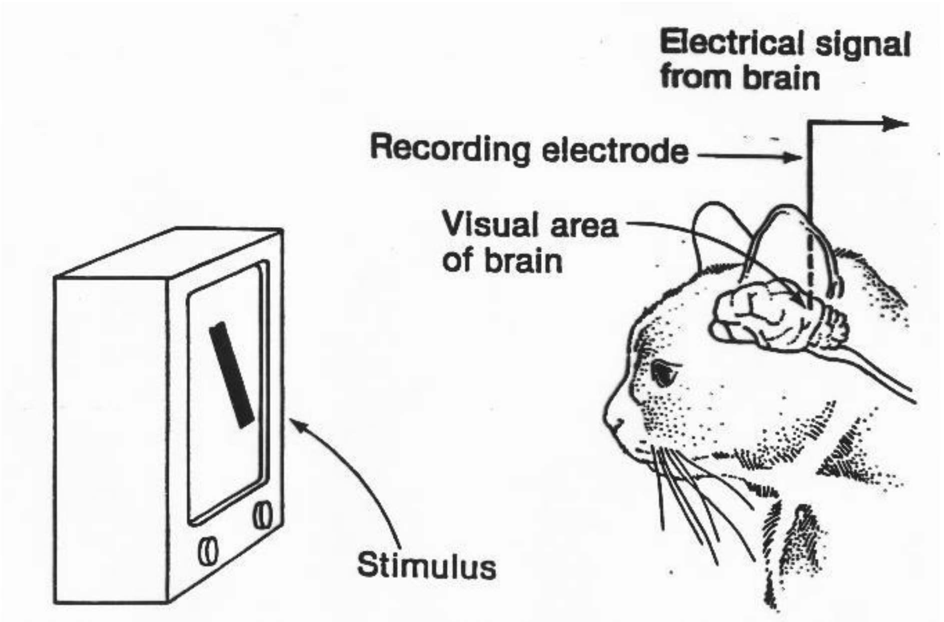
=X

=?

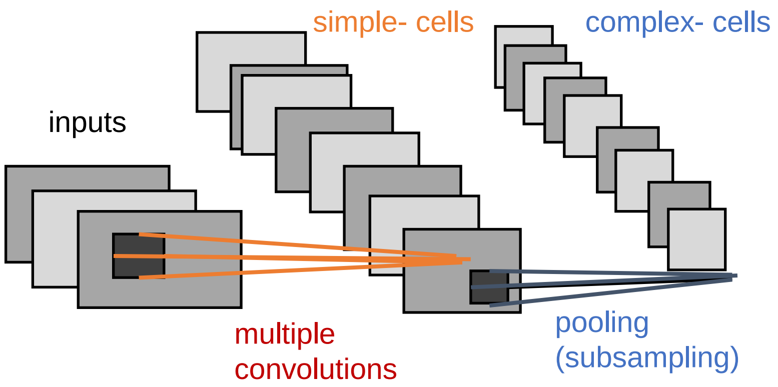
(출처:https://brohrer.github.io/how_convolutional_neural_networks_work.html)

from brain science

Hubel & Wiesel (1950s)



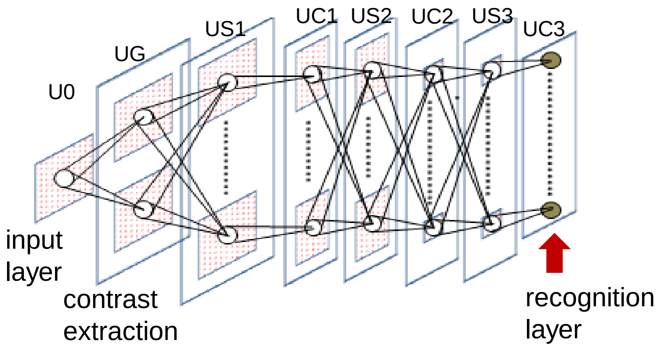
Hubel-Wiesel system



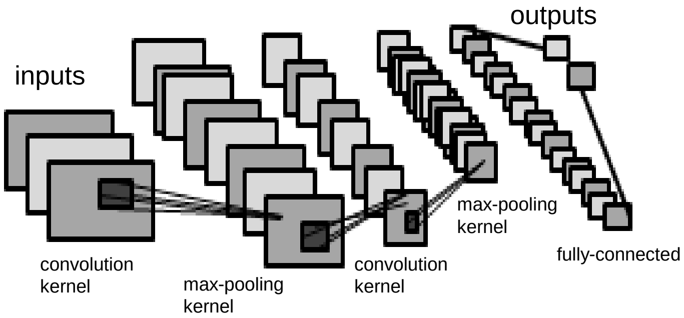
- simple cells detect local features.
- complex cells “pool” the outputs of simple cells within a retinotopic neighborhood.

deep learning history

- **“Neocognitron”** by K. Fukushima, 1980 Biological Cybernetics



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



algorithms

building blocks

since 1986

- restricted Boltzmann machine (RBM)
- auto-encoder (AE)
- sparse coding
- and so on

deep networks

since 2006

generative models

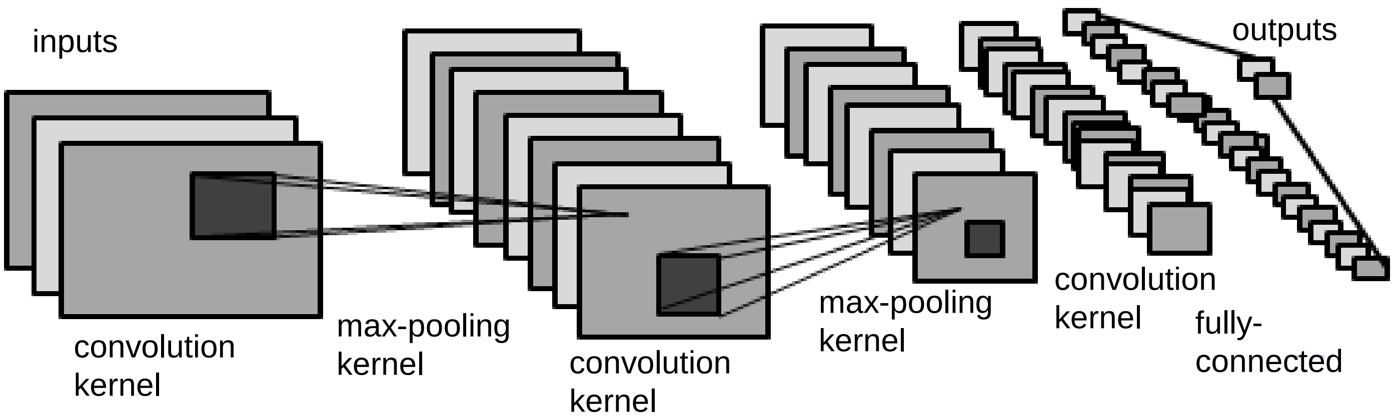
- deep belief networks (DBNs)
- generative stochastic networks (GSNs)
- deep Boltzmann machines (DBMs)
- and so on (VAE, GAN, Diffusion model)

discriminative models

- convolutional neural networks (CNNs)
- recurrent neural networks (RNNs)
- fine tuning of generative models
- and so on

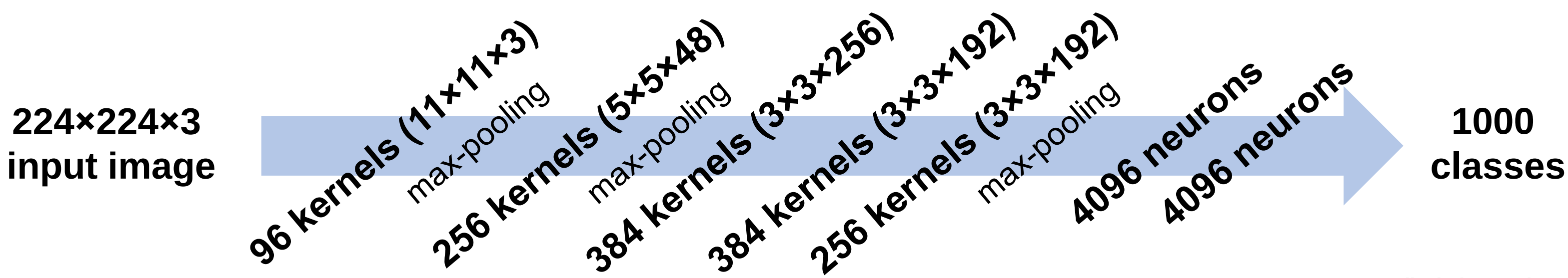
examples of CNNs

LeNet 1989



(LeCun 1989)

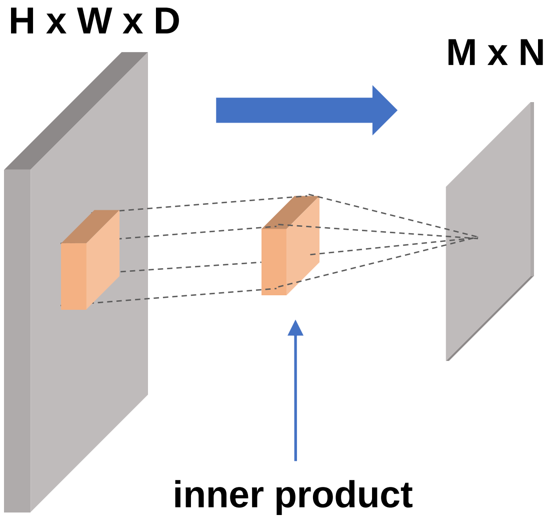
AlexNet 2012



(krizhevsky 2012)

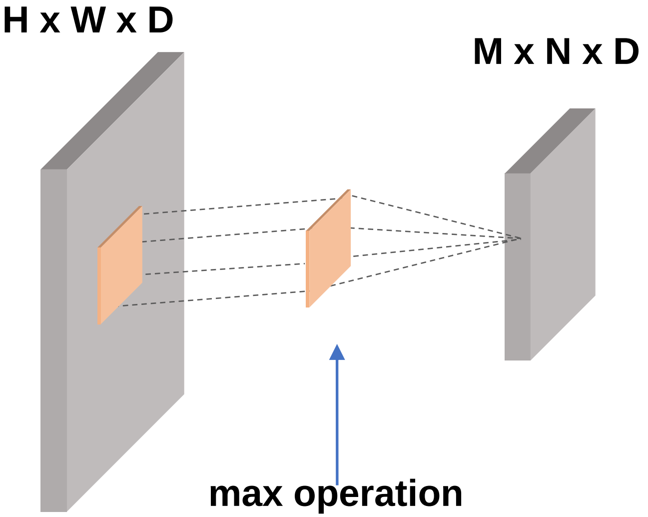
forward prop in CNNs

- **for convolutional layers**
  - kernel size, stride
  - number of kernels

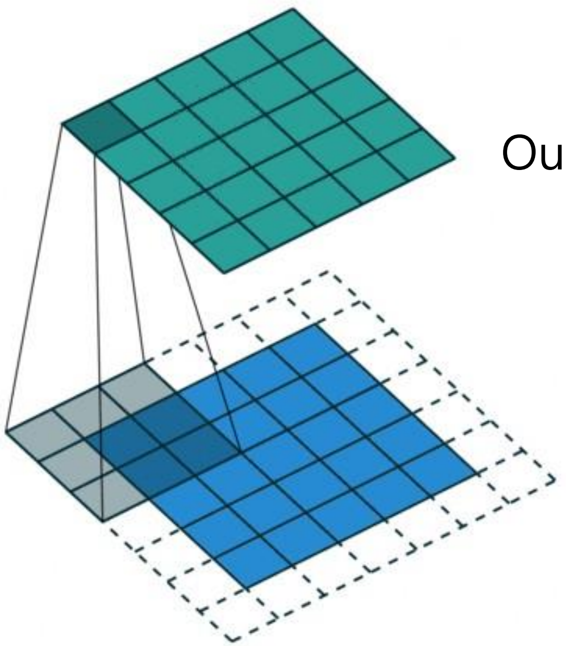


- **for pooling layers**
  - kernel size, stride

max-pooling: (Poggio 1999 nature)
a key mechanism for object recognition



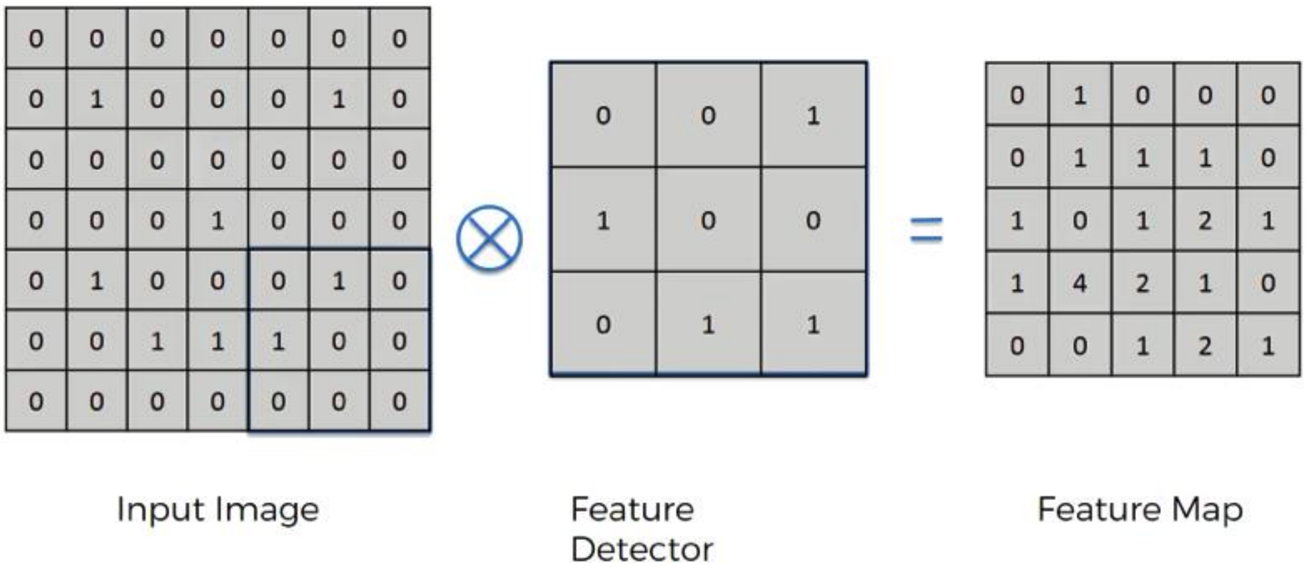
convolution operation



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

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

convolution operation



from https://www.superdatascience.com/

convolution operation

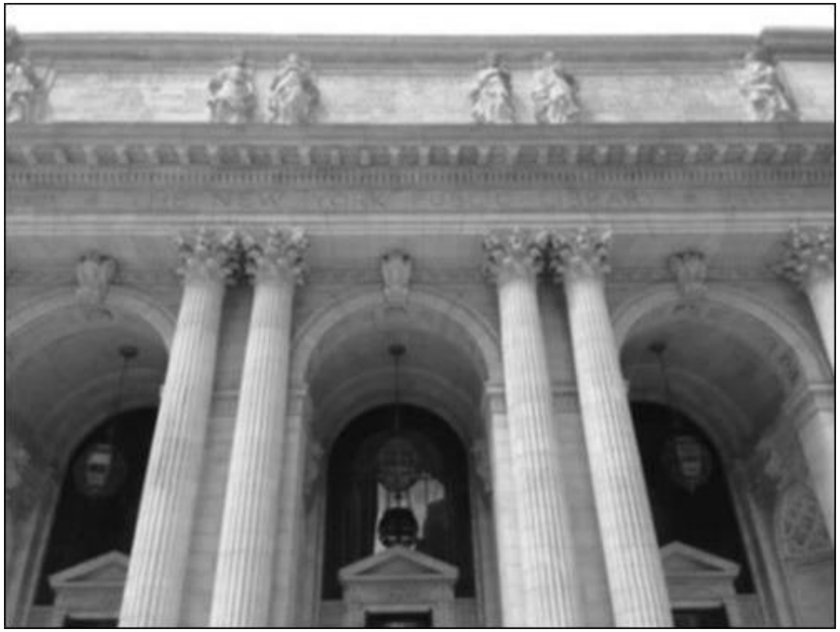
-1	-1	-1
-1	8	-1
-1	-1	-1

outline

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

convolution operation

0.06	0.12	0.06
0.12	0.25	0.12
0.06	0.12	0.06



blur

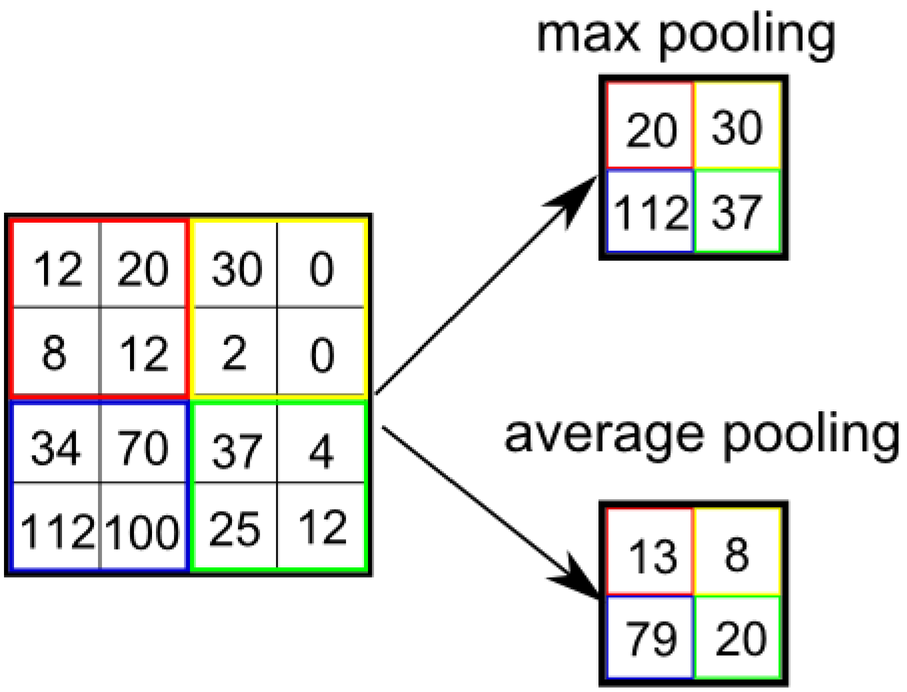
pooling

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

- sampling can reduce computation cost while keeping necessary information

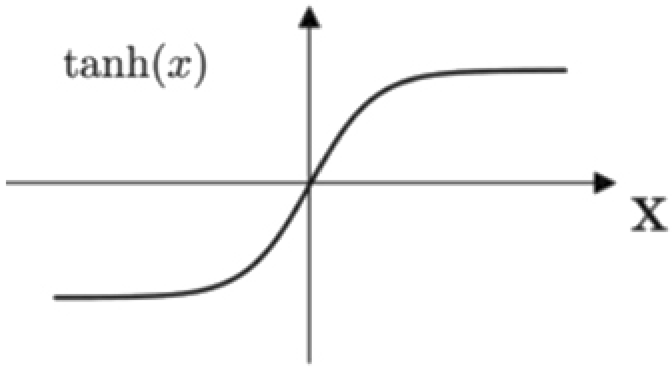
pooling

- how to sample?

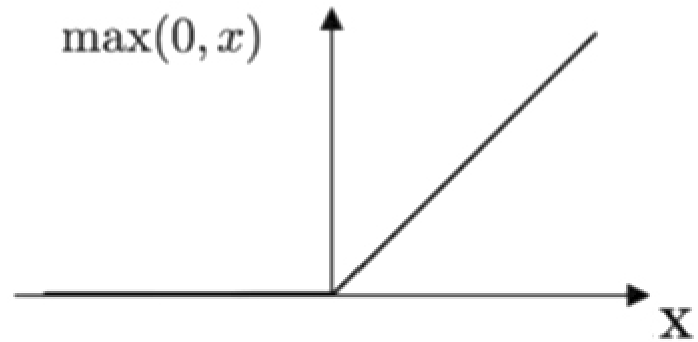


activation function

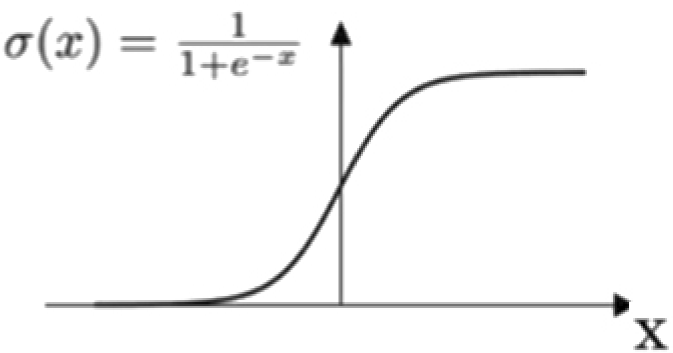
Tanh



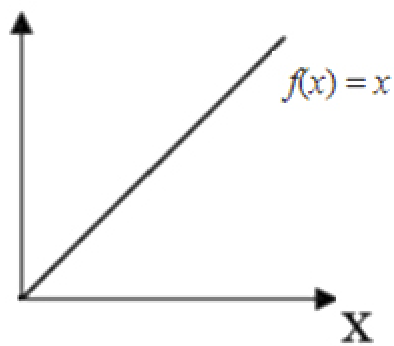
ReLU



Sigmoid



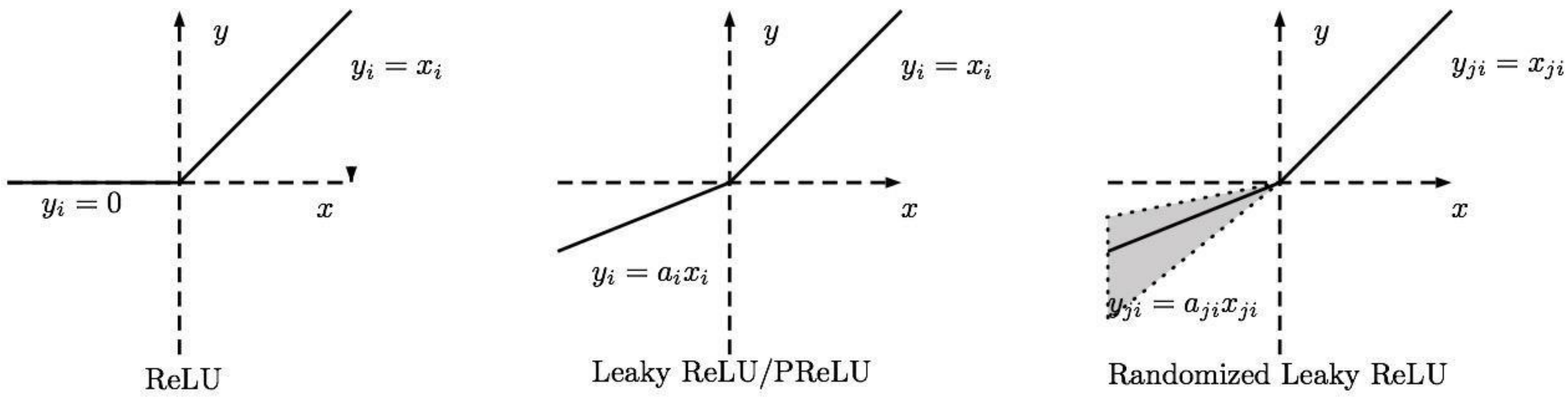
Linear



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

activation function

Rectified Linear Unit (ReLU)



f(x)=max(0,x)

f(x)=max(ax,x)

f(x)=max(ax,x)

from http://www.datasciencecentral.com/m/blogpost?id=6448529%3ABlogPost%3A408853

convolution layer

Convolution

- Kernel
- Stride
- Padding

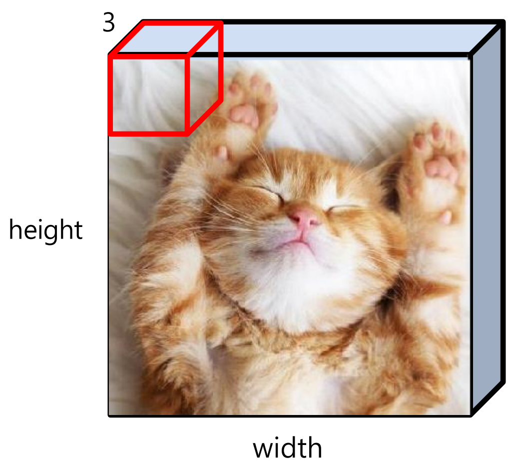
Subsampling

- Max Pooling
- Average Pooling

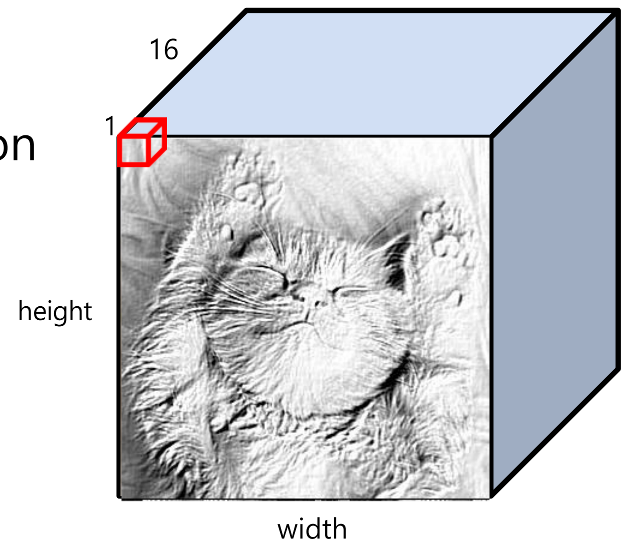
Activation Function

- ReLU
- Leaky ReLU

convolution layer: step 1

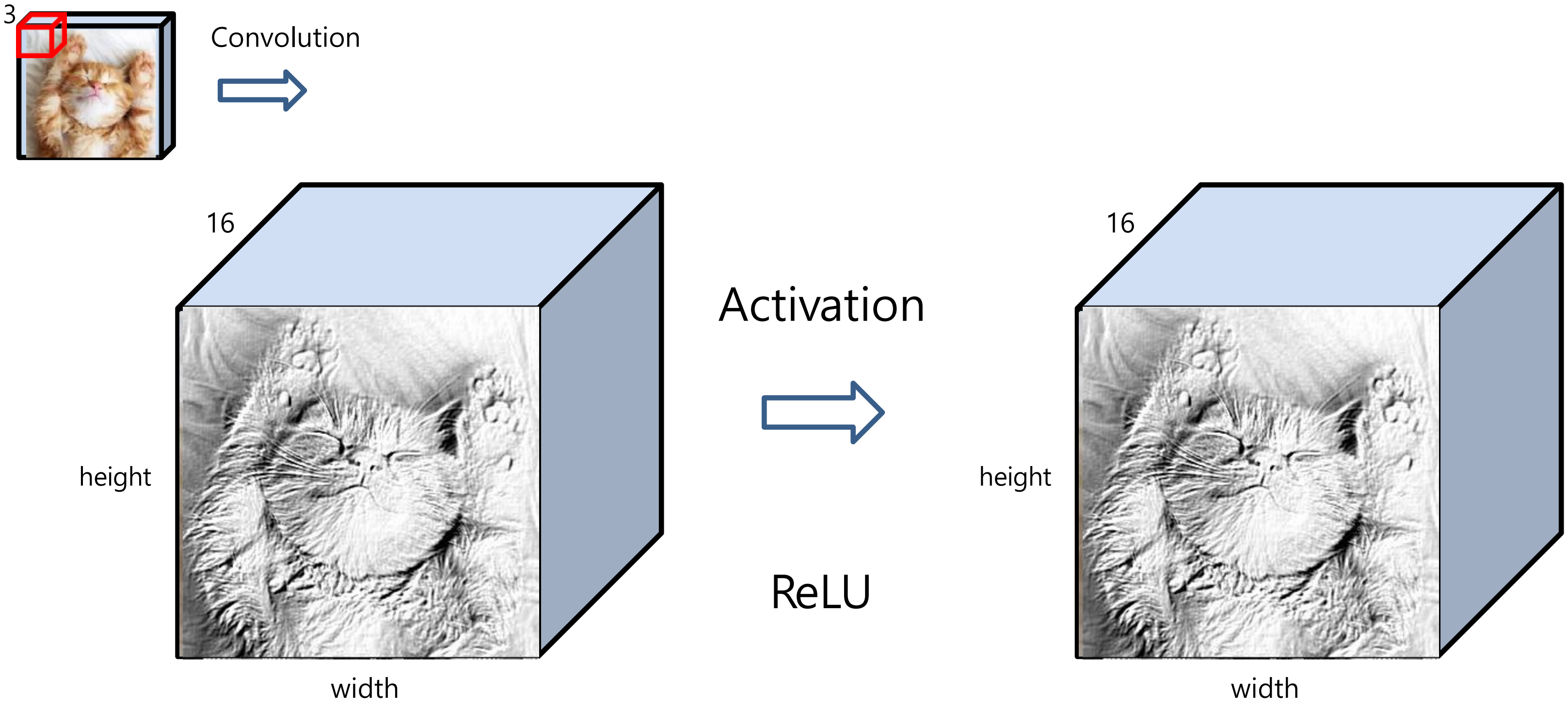


Convolution

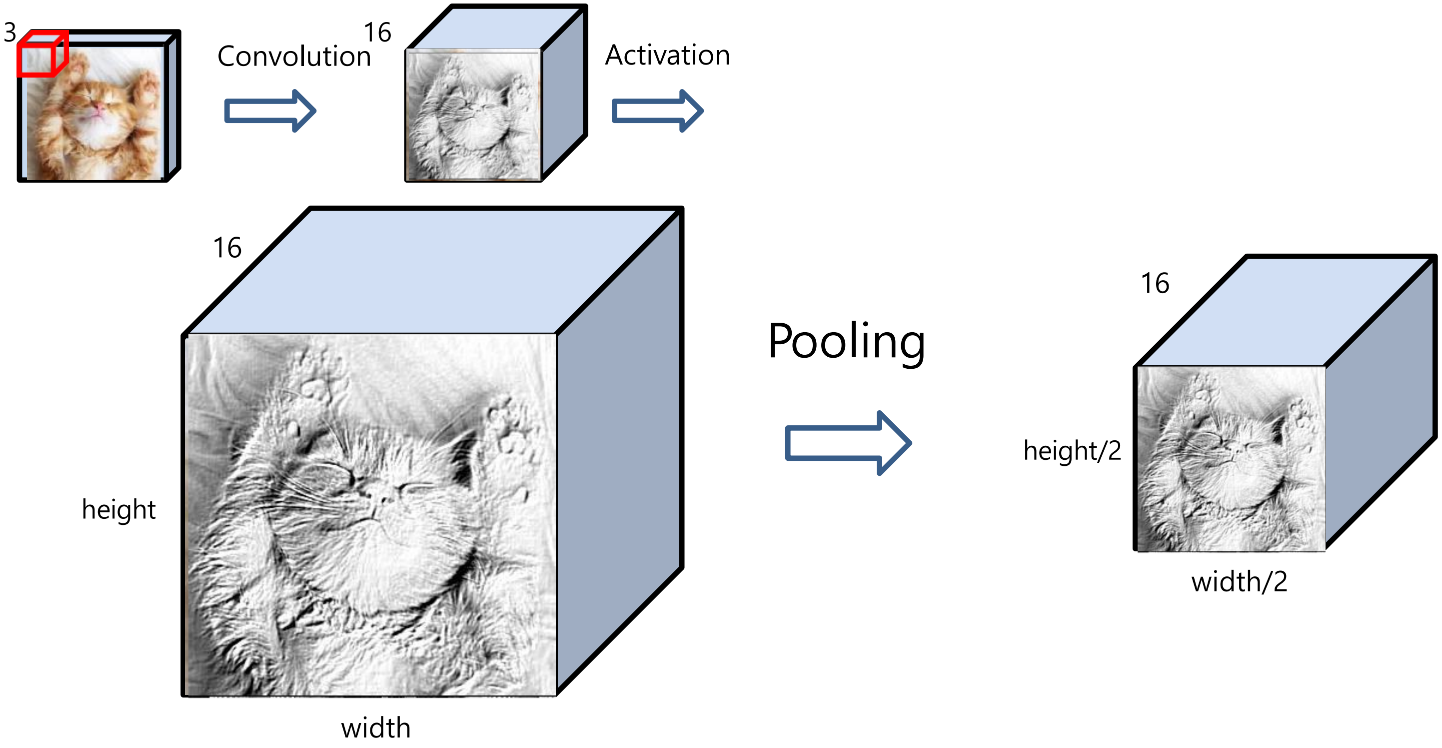


3x3 filter
stride 1
padding 1
channel 16

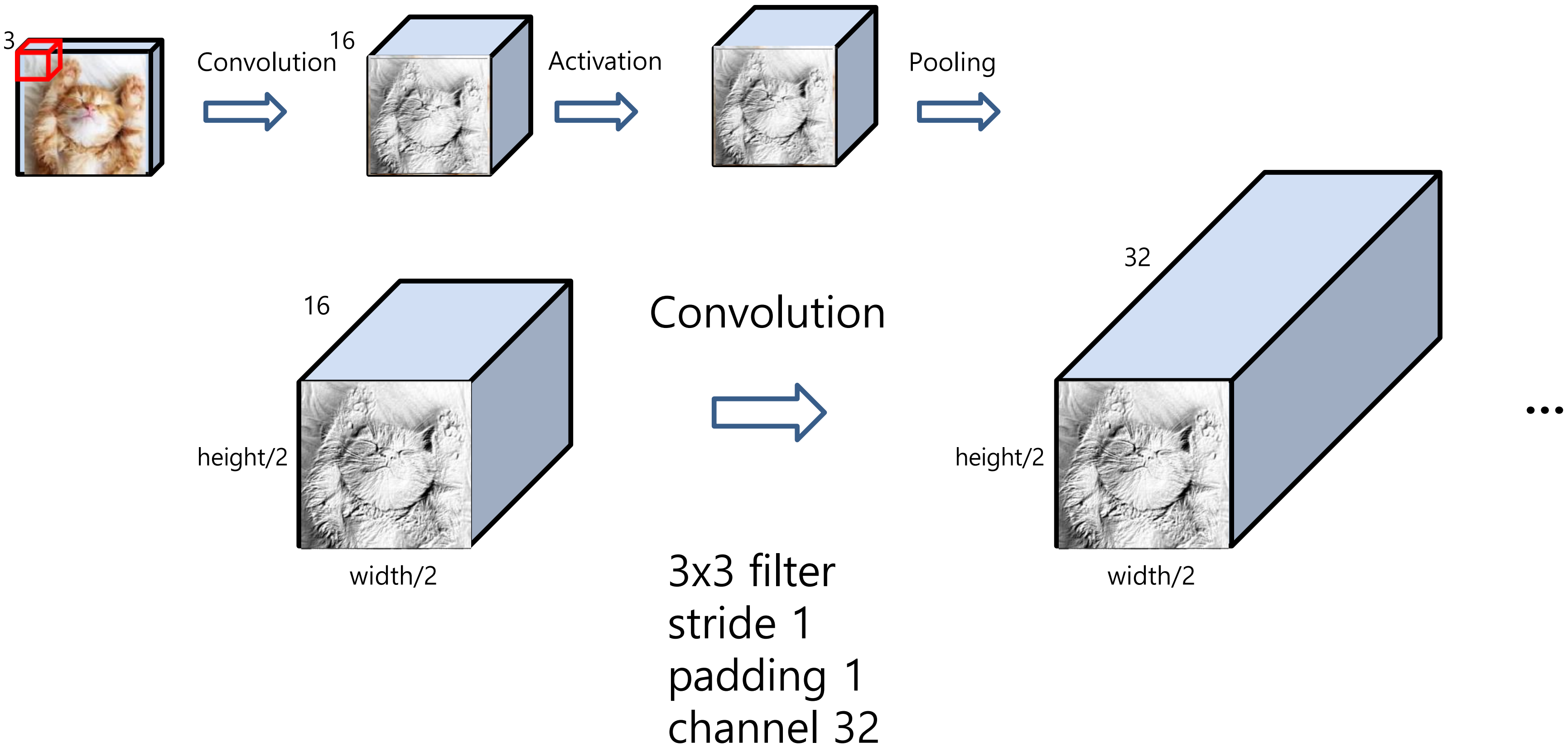
convolution layer: step 2



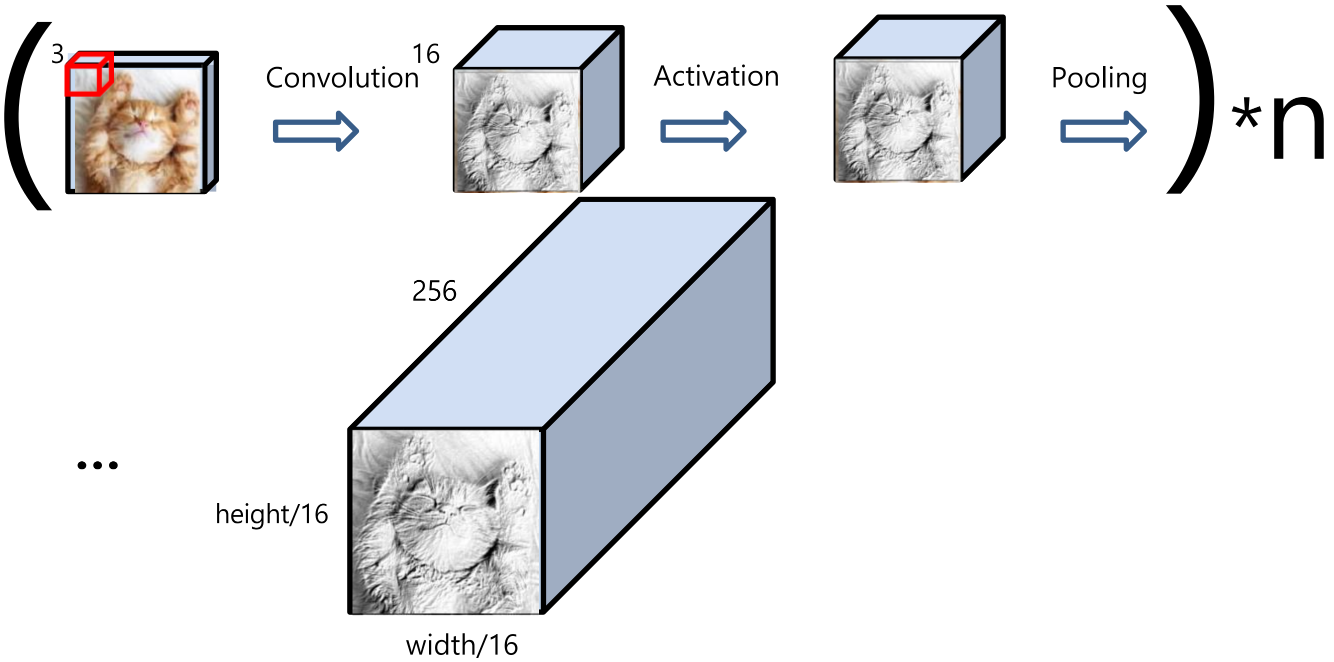
convolution layer: step 3



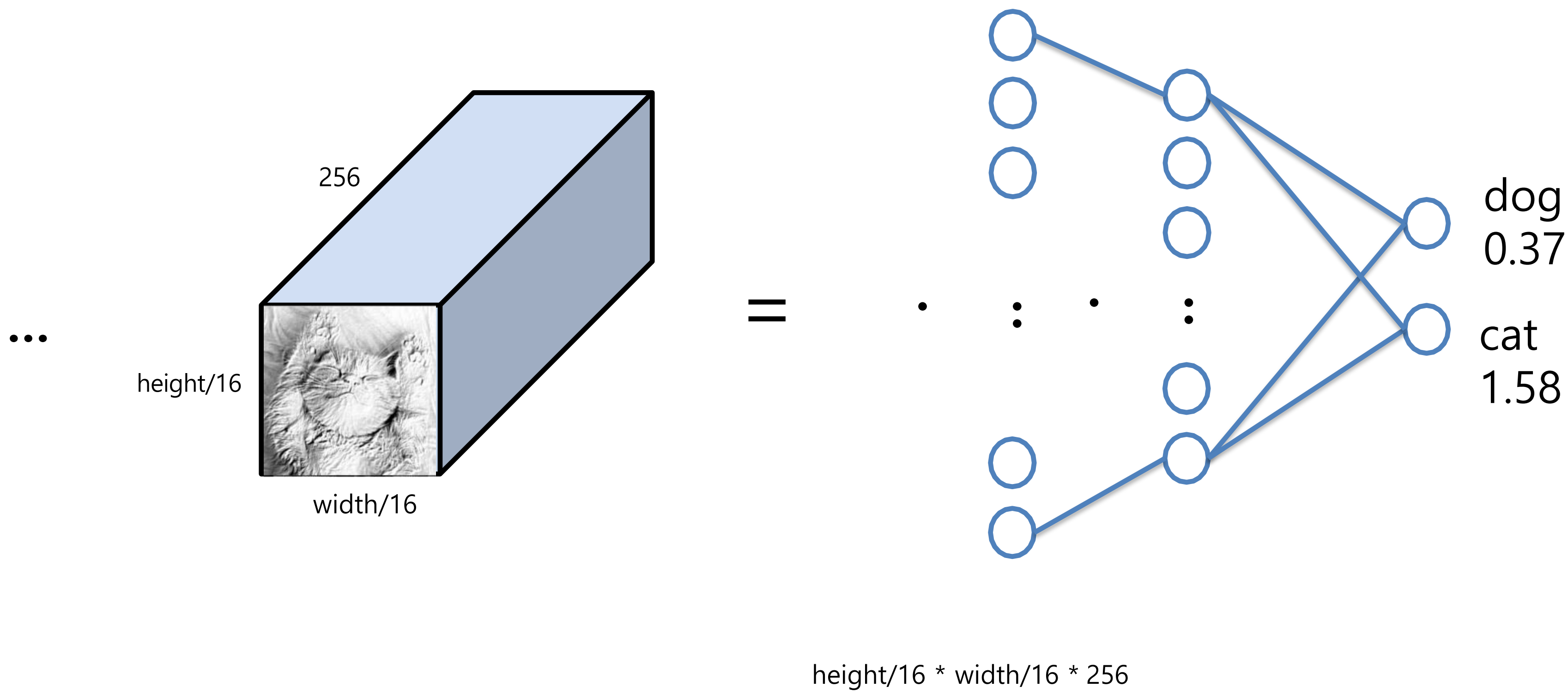
convolution layer: step 1 again



iterative convolution layer



fully connected layer



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

IMAGENET

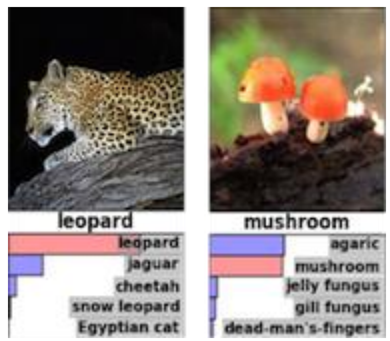
1.2 Million Images(1,200,000), 1000 Categories

the first deep learning on ImageNet data: AlexNet

ImageNet

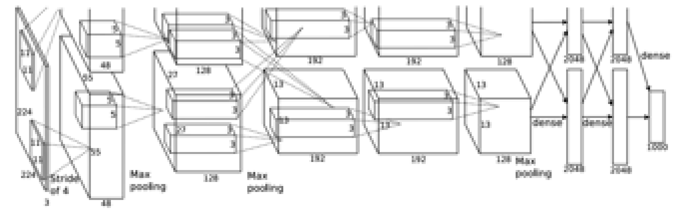
- ImageNet data set

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

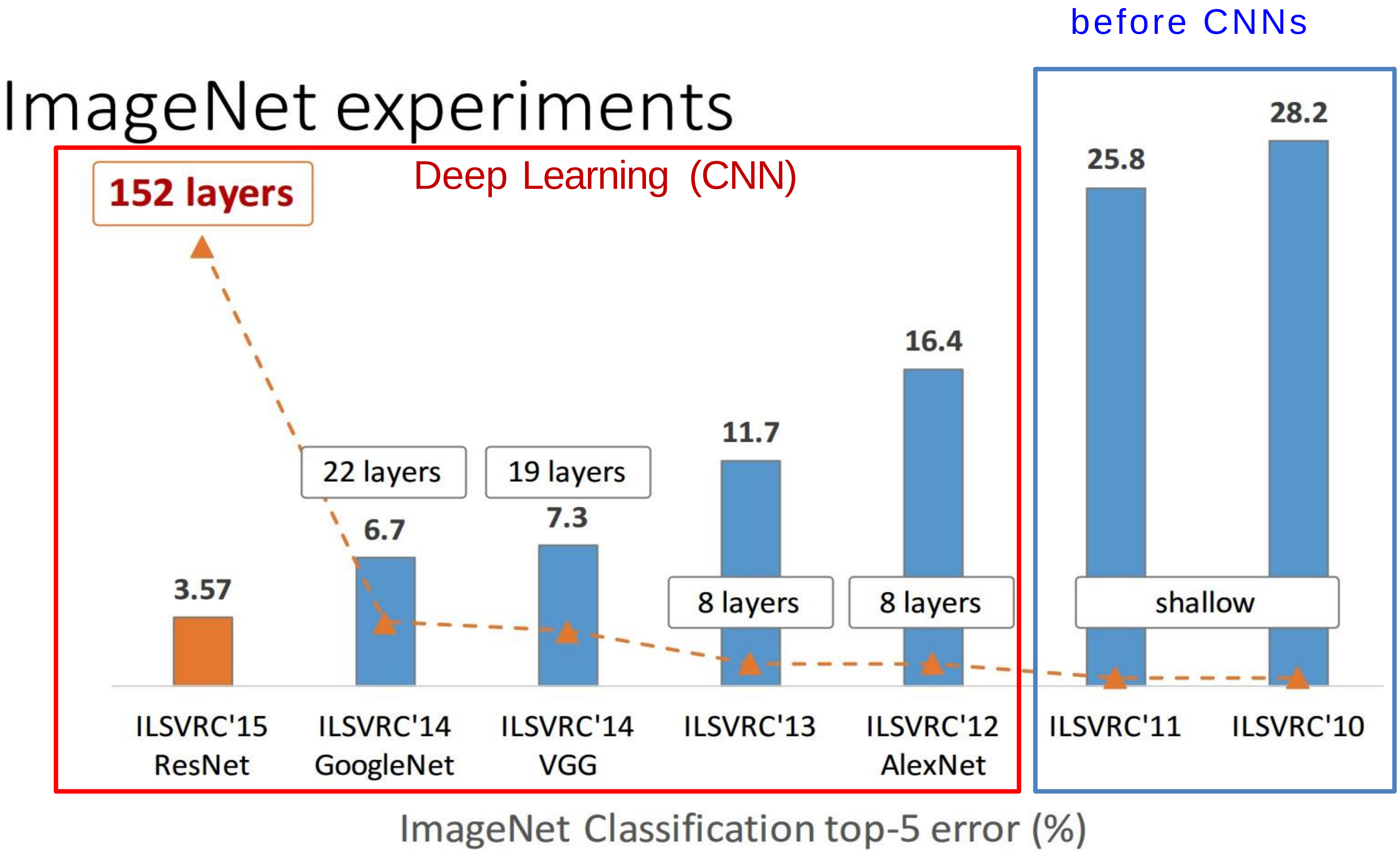
- 1000/10,184 categories
- 1.2M/8.9M training images
- 50K validation
- 150K test

(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**

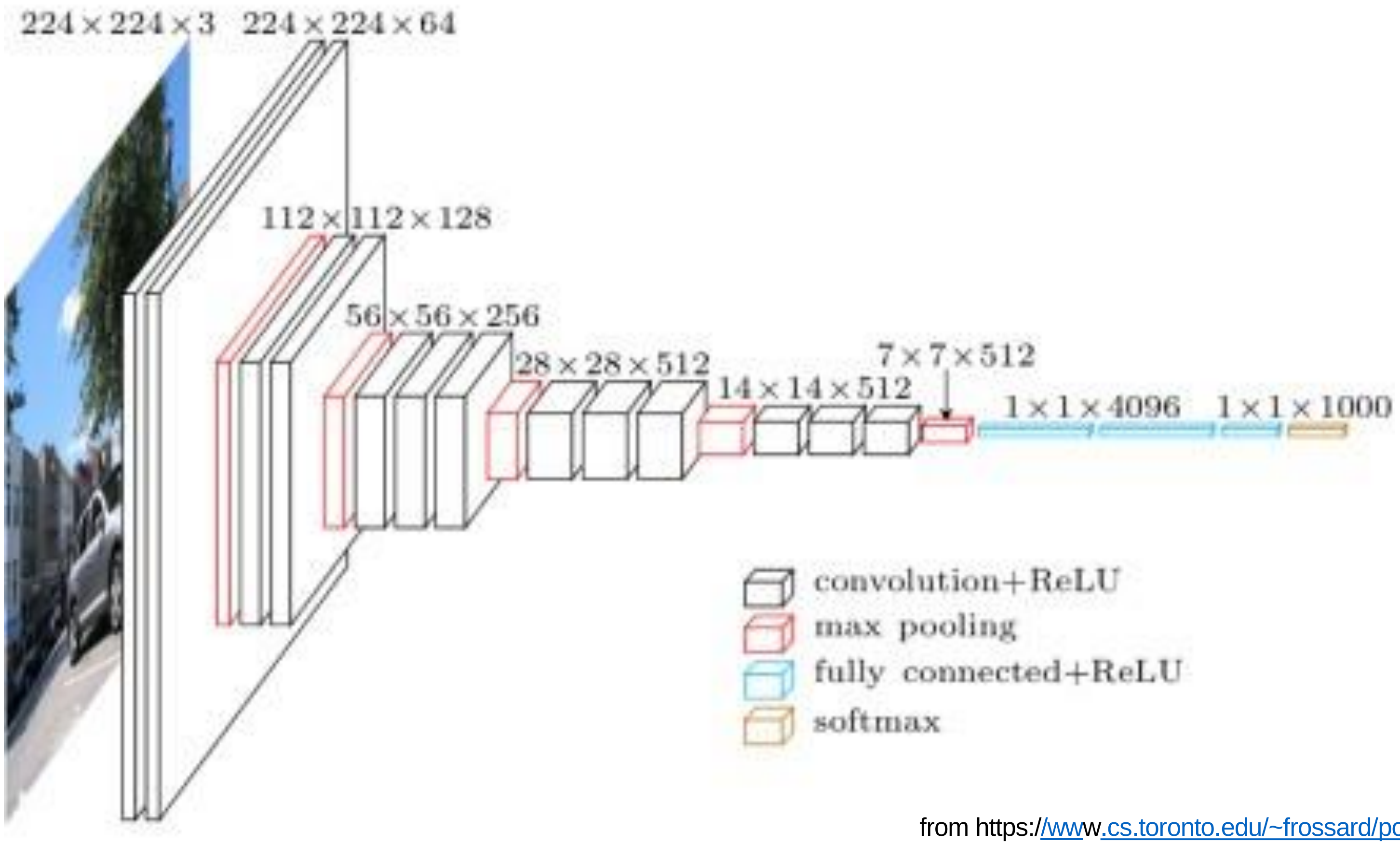
improvement after AlexNet



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

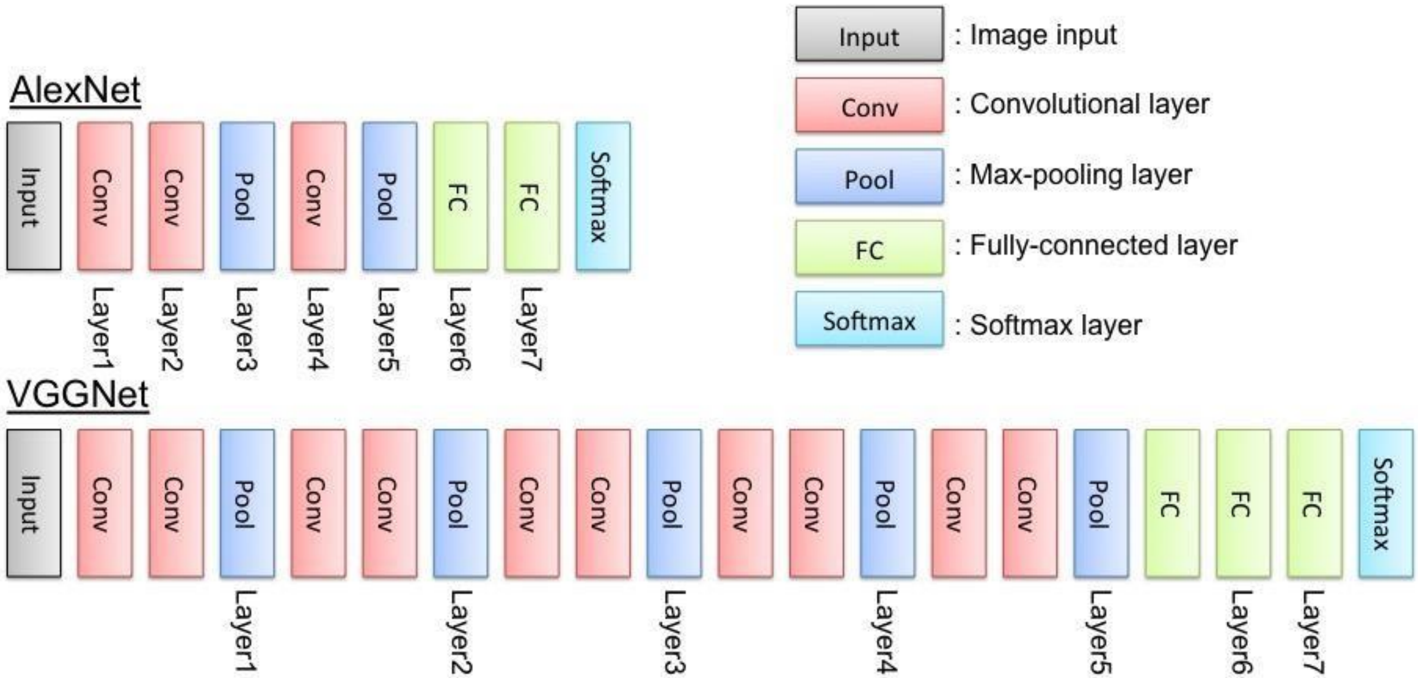
VGG Network

- simple architecture and very popular.



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

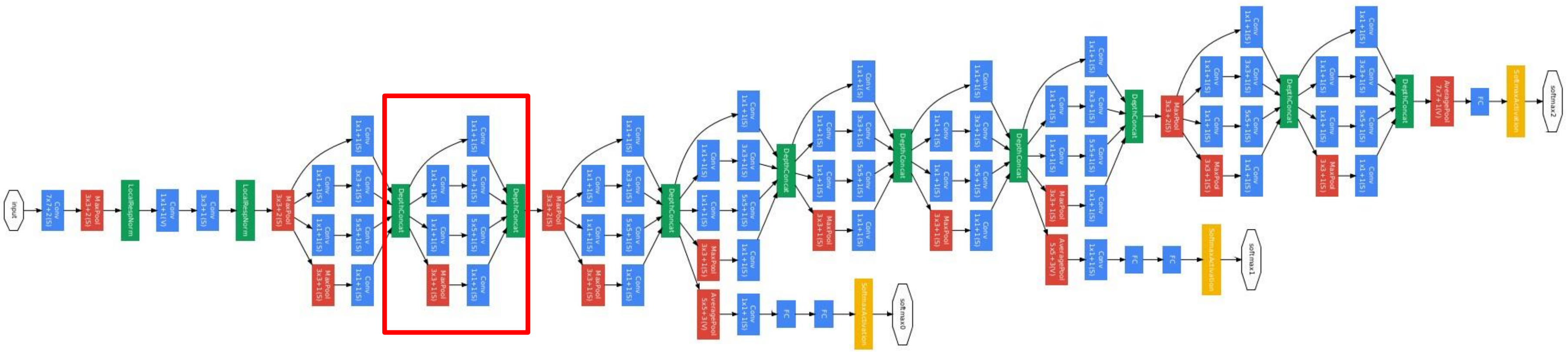
VGG Network



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

Google Network

22 Layer



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

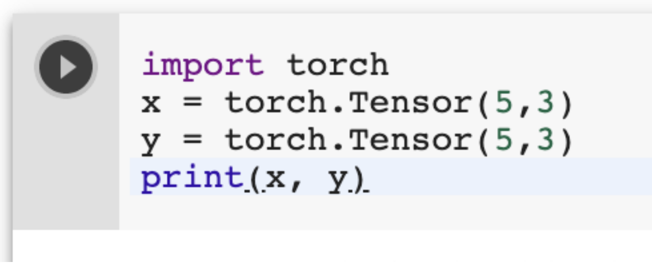
PyTorch

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)
```



PyTorch

Converting a Torch Tensor to a Numpy array and vice versa is breeze (Torch Tensor ←→ Numpy)

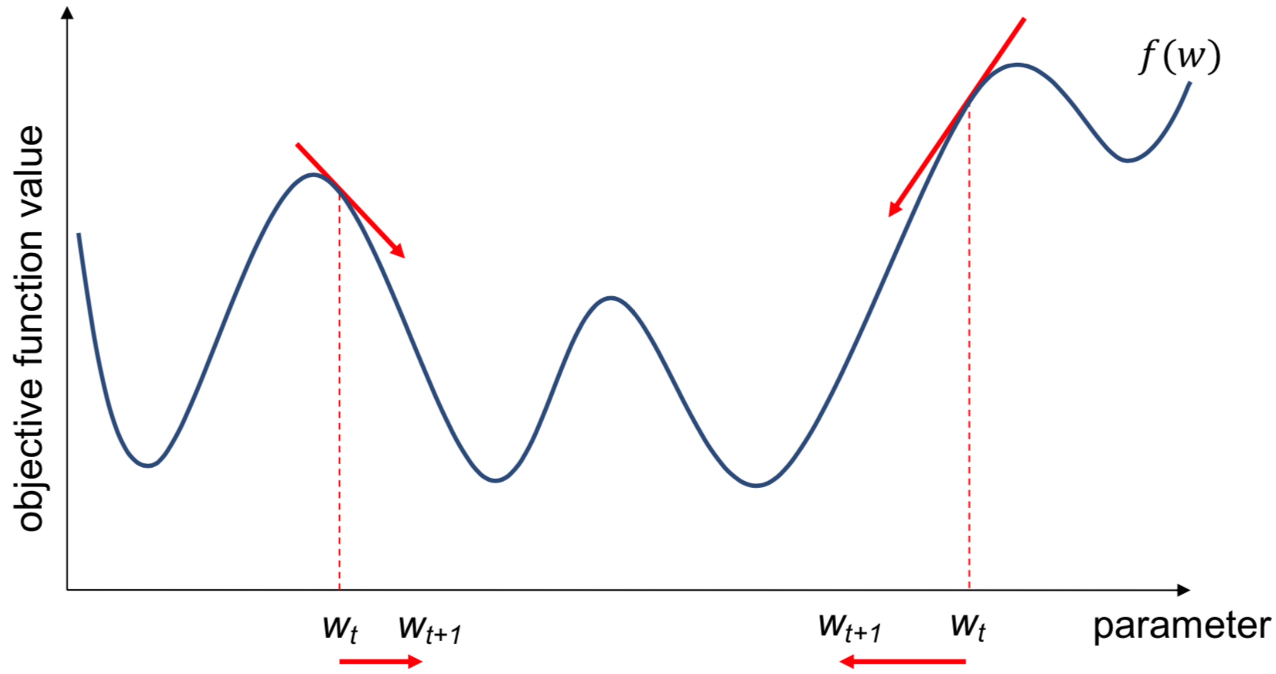
```
a = torch.ones(5)
print(a)
b = a.numpy()
print(b)
c = torch.from_numpy(b)
print(c)
```

```
tensor([1., 1., 1., 1., 1.])
[1. 1. 1. 1. 1.]
tensor([1., 1., 1., 1., 1.])
```

PyTorch - Cuda

- CUDA Tensors
  - Tensors can be moved onto any device using the .to method

Autograd

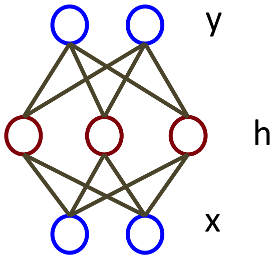


- The autograd provides automatic differentiation for all operations on Tensors.
- Once you finish your computation, you can call .backward() for autograd

- **Autograd allows you to automatically compute gradients.**

Autograd

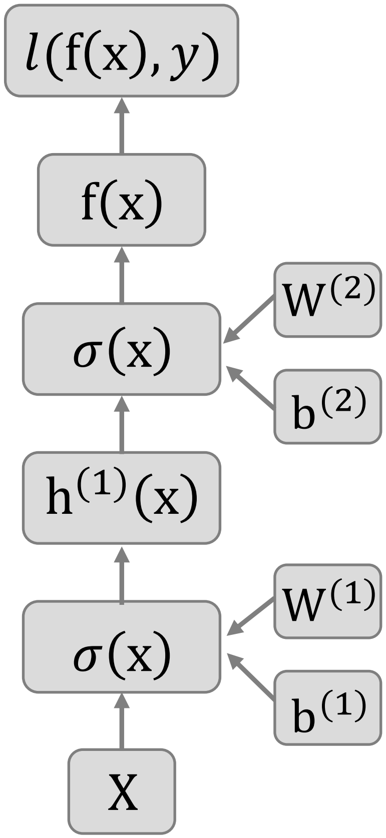
- **feed-forward neural networks**



$$h_t = \sigma(\boldsymbol{W_{xh}} x_t + b_h),$$
$$y_t = W_{hy} h_t + b_y,$$

Autograd allows you to automatically compute gradients.

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…)

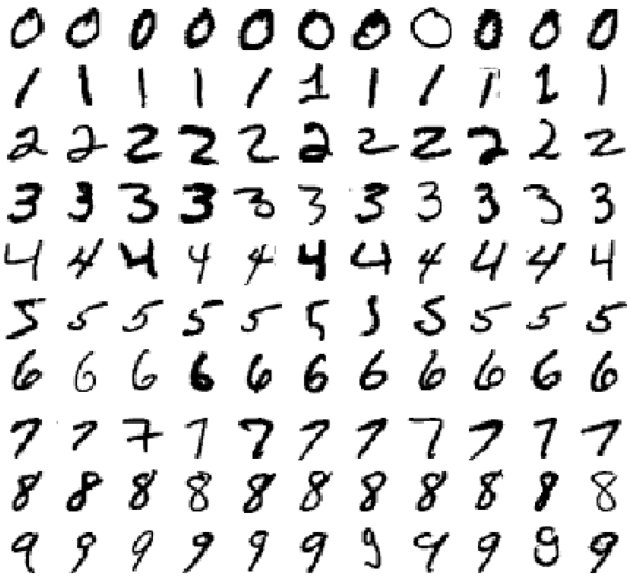
Practice

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

- Folders
  - data
  - results

- Script files
  - Week13_Lab_MNIST_student.ipynb
  - Week13_Lab_ImageNet_student.ipynb

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.pth")


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

```python
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

```
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: 2.309637
Train Epoch: 1 [6400/60000 (11%)]	Loss: 1.329176
Train Epoch: 1 [12800/60000 (21%)]	Loss: 0.706647
Train Epoch: 1 [19200/60000 (32%)]	Loss: 0.438433
Train Epoch: 1 [25600/60000 (43%)]	Loss: 0.593319
Train Epoch: 1 [32000/60000 (53%)]	Loss: 0.353528
Train Epoch: 1 [38400/60000 (64%)]	Loss: 0.276715
Train Epoch: 1 [44800/60000 (75%)]	Loss: 0.546666
Train Epoch: 1 [51200/60000 (85%)]	Loss: 0.174126
Train Epoch: 1 [57600/60000 (96%)]	Loss: 0.285674

Test: Average loss: 0.3004, Accuracy: 9155/10000 (92%)

Train Epoch: 2 [0/60000 (0%)]	Loss: 0.343893
Train Epoch: 2 [6400/60000 (11%)]	Loss: 0.128626
Train Epoch: 2 [12800/60000 (21%)]	Loss: 0.349224
Train Epoch: 2 [19200/60000 (32%)]	Loss: 0.306493
Train Epoch: 2 [25600/60000 (43%)]	Loss: 0.306759
Train Epoch: 2 [32000/60000 (53%)]	Loss: 0.119653
Train Epoch: 2 [38400/60000 (64%)]	Loss: 0.246272
Train Epoch: 2 [44800/60000 (75%)]	Loss: 0.407206
Train Epoch: 2 [51200/60000 (85%)]	Loss: 0.331753
Train Epoch: 2 [57600/60000 (96%)]	Loss: 0.114386

Test: Average loss: 0.2307, Accuracy: 9343/10000 (93%)
```

NetFF Model training/testing

```
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: 2.303299
Train Epoch: 1 [6400/60000 (11%)]	Loss: 0.593885
Train Epoch: 1 [12800/60000 (21%)]	Loss: 0.222533
Train Epoch: 1 [19200/60000 (32%)]	Loss: 0.395274
Train Epoch: 1 [25600/60000 (43%)]	Loss: 0.148499
Train Epoch: 1 [32000/60000 (53%)]	Loss: 0.187254
Train Epoch: 1 [38400/60000 (64%)]	Loss: 0.202007
Train Epoch: 1 [44800/60000 (75%)]	Loss: 0.050570
Train Epoch: 1 [51200/60000 (85%)]	Loss: 0.163213
Train Epoch: 1 [57600/60000 (96%)]	Loss: 0.074786

Test: Average loss: 0.1089, Accuracy: 9687/10000 (97%)

Train Epoch: 2 [0/60000 (0%)]	Loss: 0.322160
Train Epoch: 2 [6400/60000 (11%)]	Loss: 0.089705
Train Epoch: 2 [12800/60000 (21%)]	Loss: 0.052976
Train Epoch: 2 [19200/60000 (32%)]	Loss: 0.108583
Train Epoch: 2 [25600/60000 (43%)]	Loss: 0.084108
Train Epoch: 2 [32000/60000 (53%)]	Loss: 0.114488
Train Epoch: 2 [38400/60000 (64%)]	Loss: 0.161090
Train Epoch: 2 [44800/60000 (75%)]	Loss: 0.079185
Train Epoch: 2 [51200/60000 (85%)]	Loss: 0.008939
Train Epoch: 2 [57600/60000 (96%)]	Loss: 0.097126

Test: Average loss: 0.0607, Accuracy: 9810/10000 (98%)
```

Testing with one test image

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

```
from skimage import io

img_name = './data/mnist_test_images/test2.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/test2.jpg 4
```

Testing with multiple test image

```python
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])
  time.sleep(0.5)
```

```
./data/mnist_test_images\test9.jpg 4
./data/mnist_test_images\test7.jpg 3
./data/mnist_test_images\test8.jpg 1
./data/mnist_test_images\test3.jpg 1
./data/mnist_test_images\test19.jpg 9
./data/mnist_test_images\test2.jpg 4
./data/mnist_test_images\test20.jpg 4
./data/mnist_test_images\test4.jpg 9
./data/mnist_test_images\test17.jpg 8
./data/mnist_test_images\test18.jpg 6
./data/mnist_test_images\test6.jpg 1
./data/mnist_test_images\test5.jpg 2
```

ImageNet Example

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

ImageNet class names

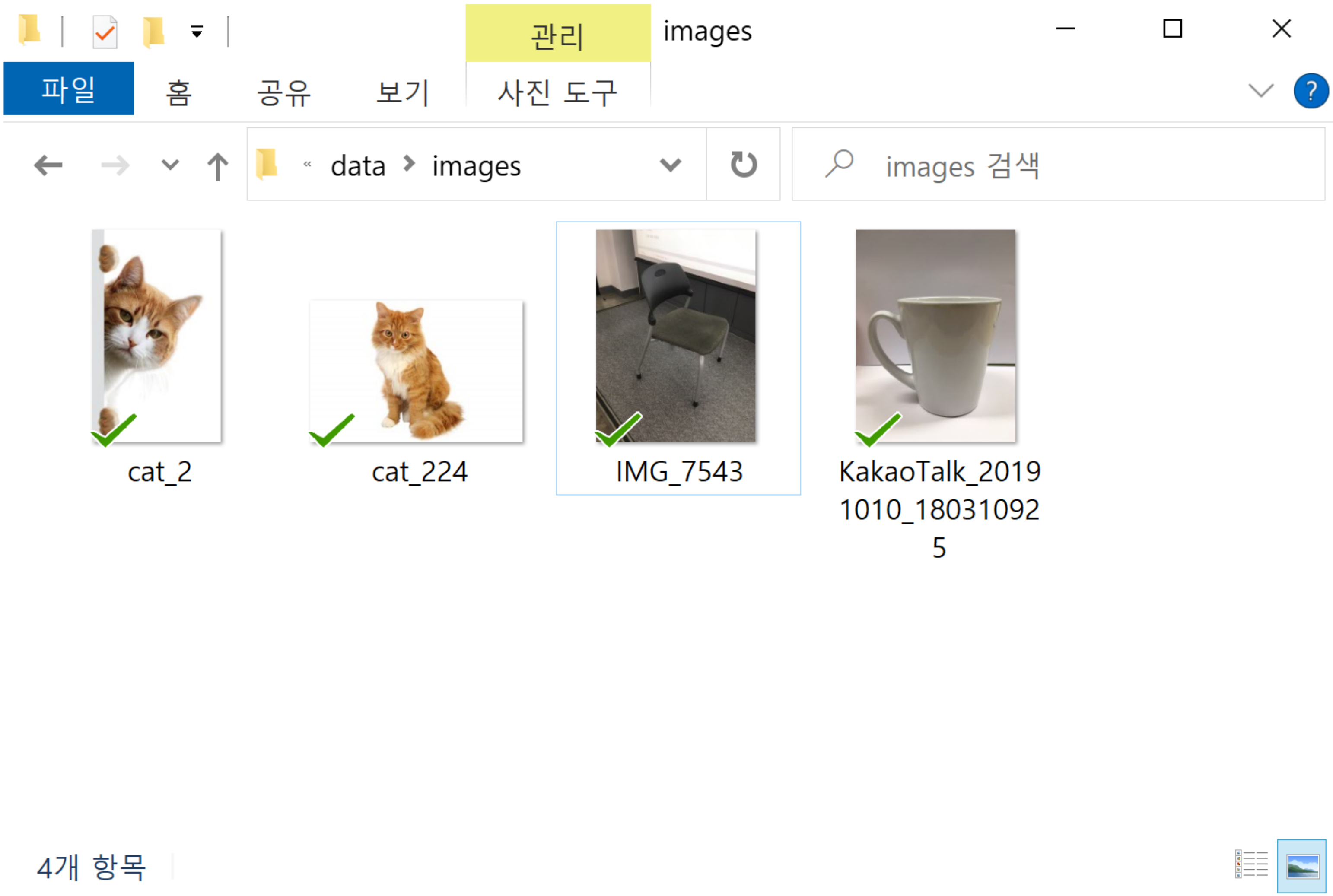
```
import csv
name_file = './data/imagenet_install/class_names.csv'

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



classification of a single image

```
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 image

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