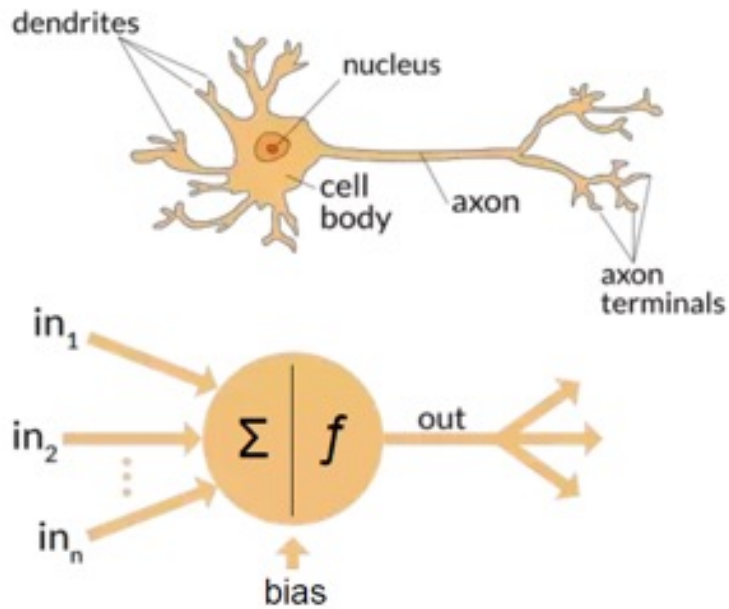


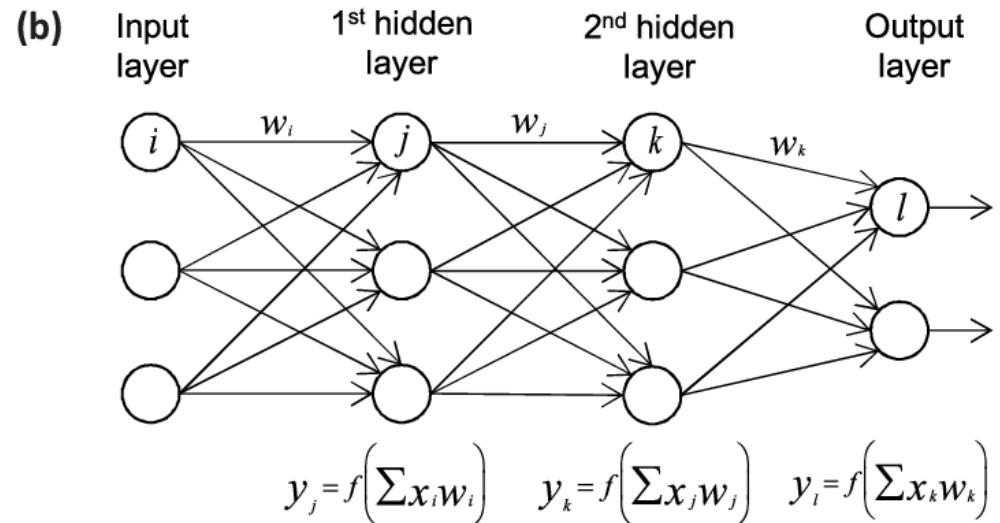
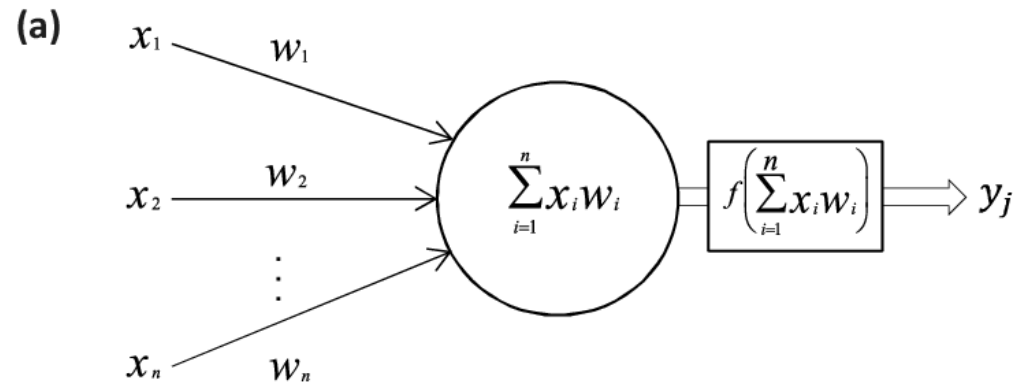
# Intro to Deep Learning

ECE30007 Intro to AI Project

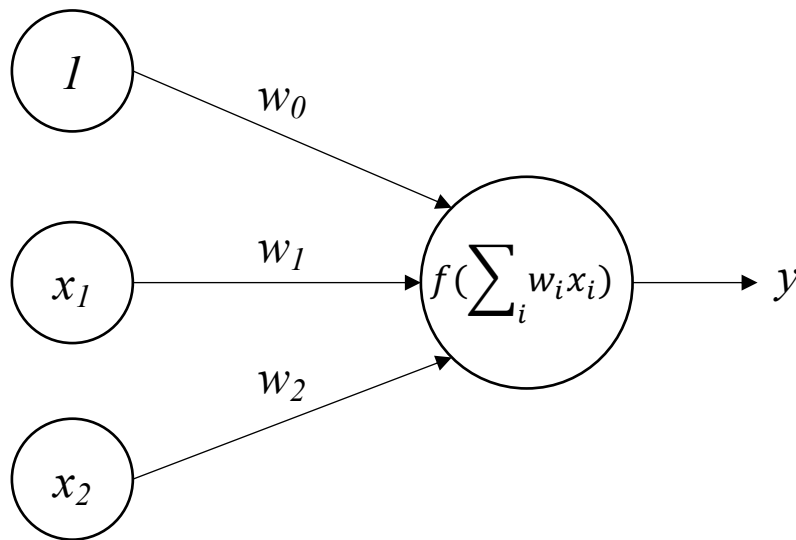
# Artificial Neural Network



Perceptron (Rosenblatt, 1957)



# Artificial Neural Network 1 (Perceptron)

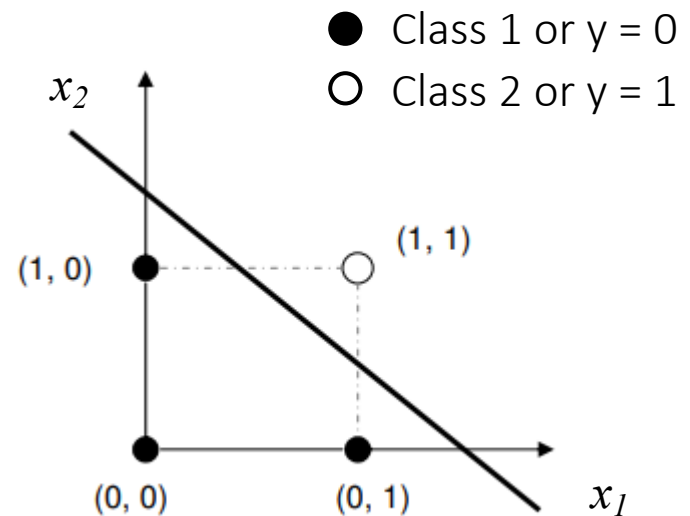


with a simple example of  $f$

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

if  $\sum_i w_i x_i > 0$   $y = 1$   
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  $w_1 = w_2 = 1$

For input (1,1)

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

$$f(\sum_i w_i x_i) = y = 1. \text{ (Class 1)}$$

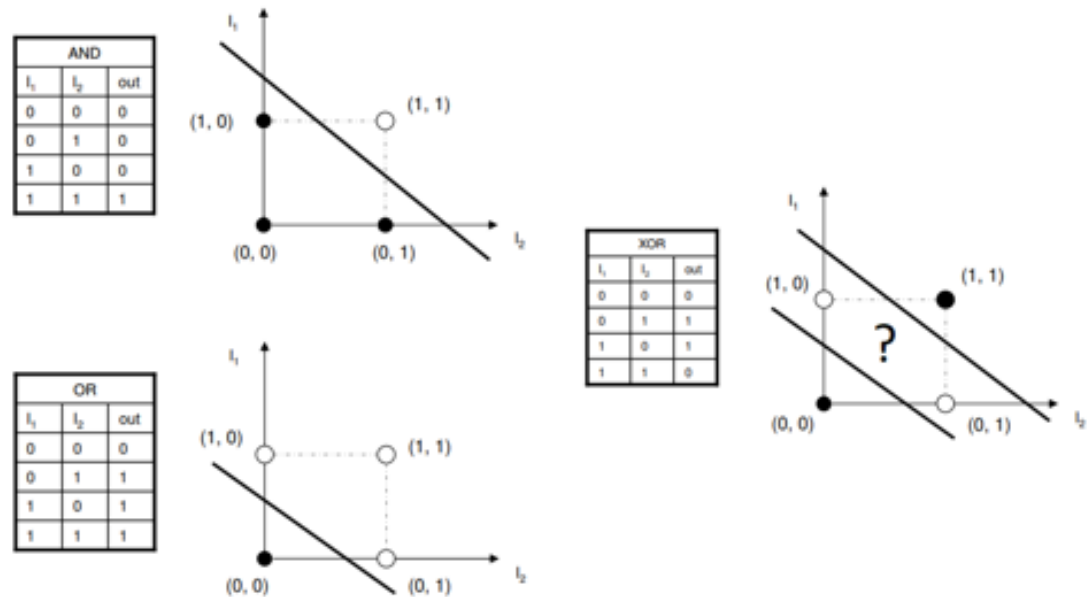
For input (0,1)

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

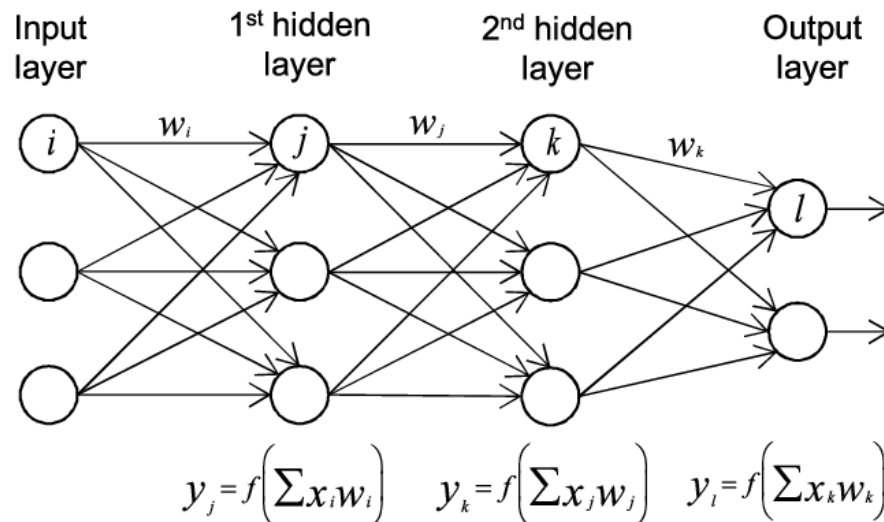
$$f(\sum_i w_i x_i) = y = 0. \text{ (Class 2)}$$

# Artificial Neural Network 2 (MLP)

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



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



# 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

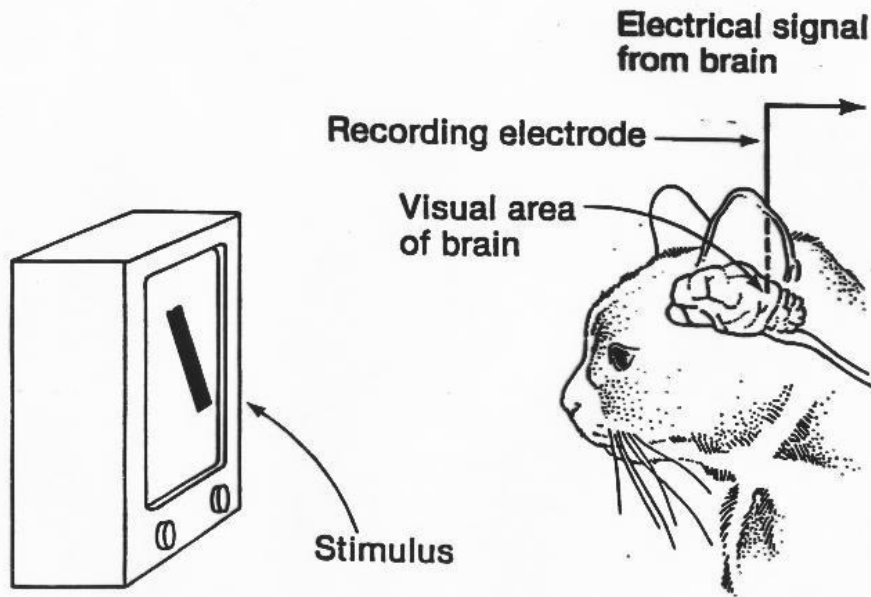


- single-image, multi-class
- more than 40 classes
- more than 50K images in total
- large, lifelike database

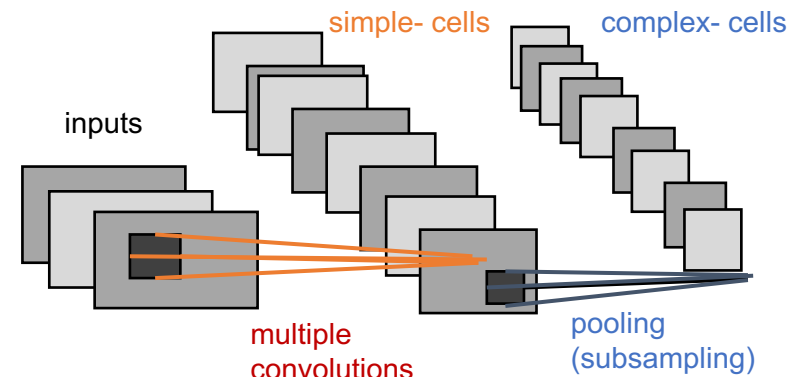
rank	team	method	accuracy (%)
1	IDSIA	Committee of CNNs	99.46
2	INI	Human performance	98.84
3	Sermanet	Multi-scale CNNs	98.31
4	CAOR	Random Forests	96.14

# from brain science

Hubel & Wiesel (1950s)



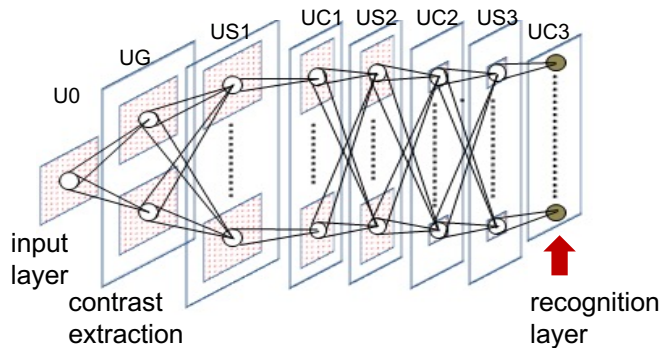
## Hubel-Wiesel system



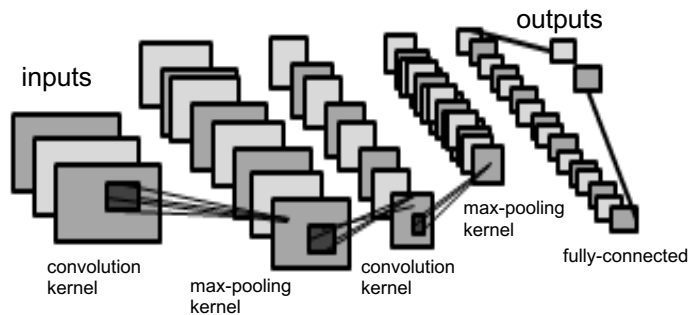
- simple cells detect local features.
- complex cells “pool” the outputs of simple cells within a retinotopic neighborhood.

# deep learning history

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

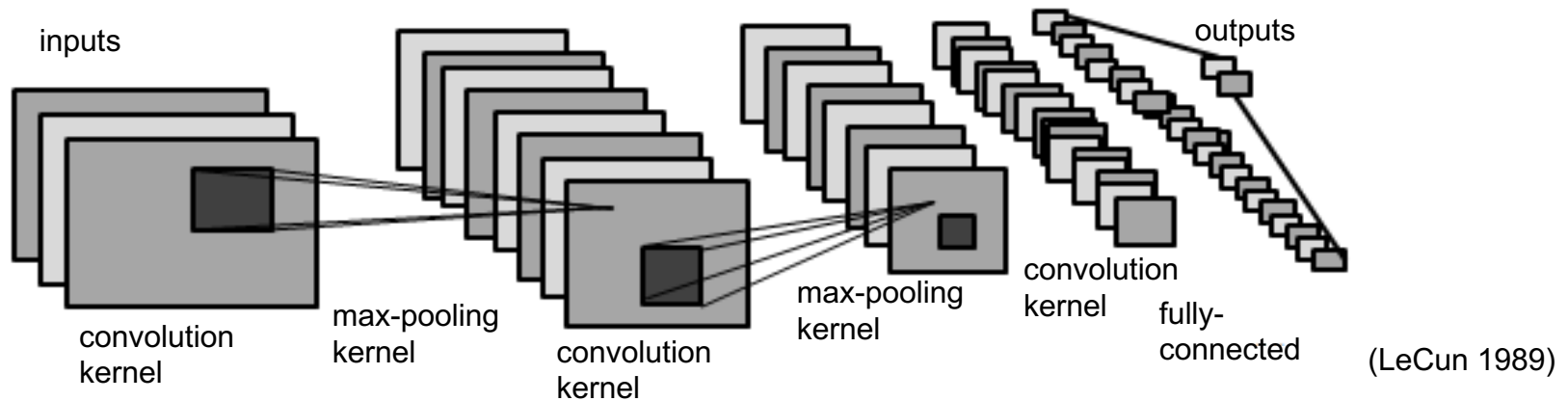


- **convolutional neural networks** by Y. LeCun et al., 1989 Neural Computation

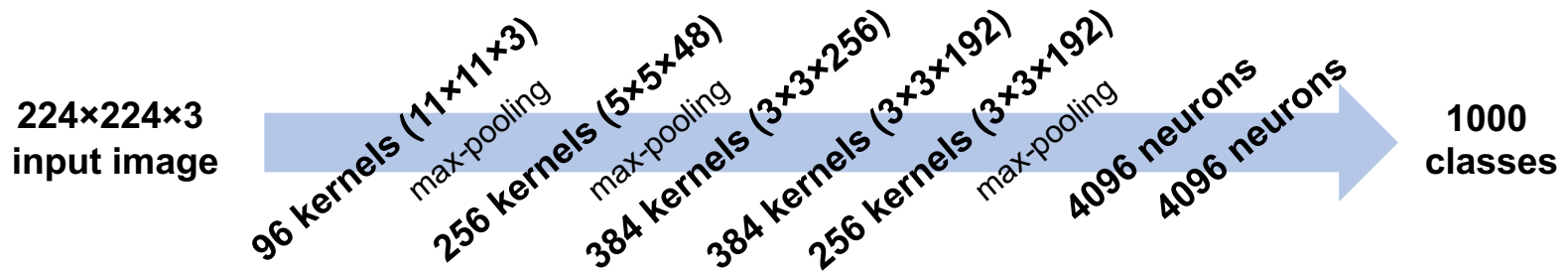


# examples of CNNs

## LeNet 1989

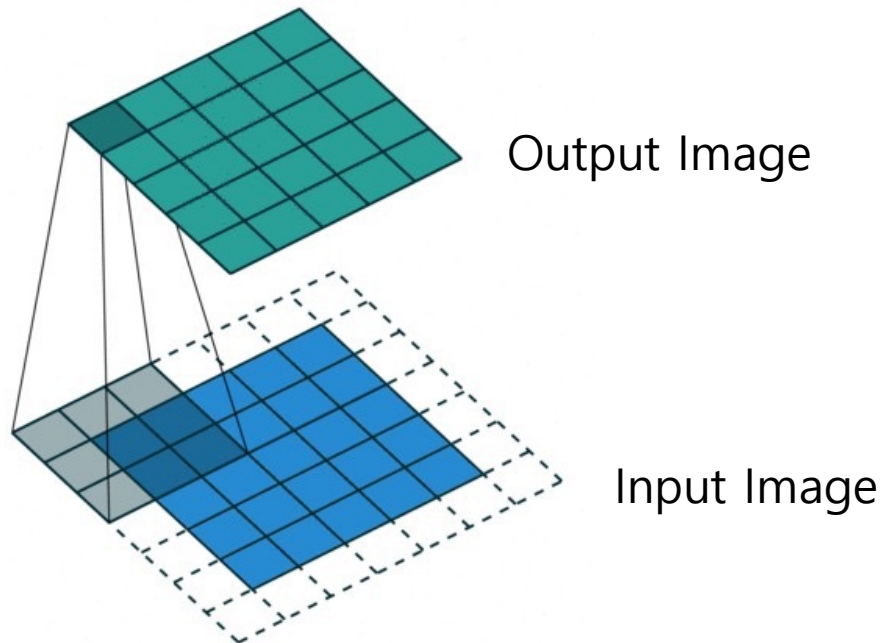


## AlexNet 2012





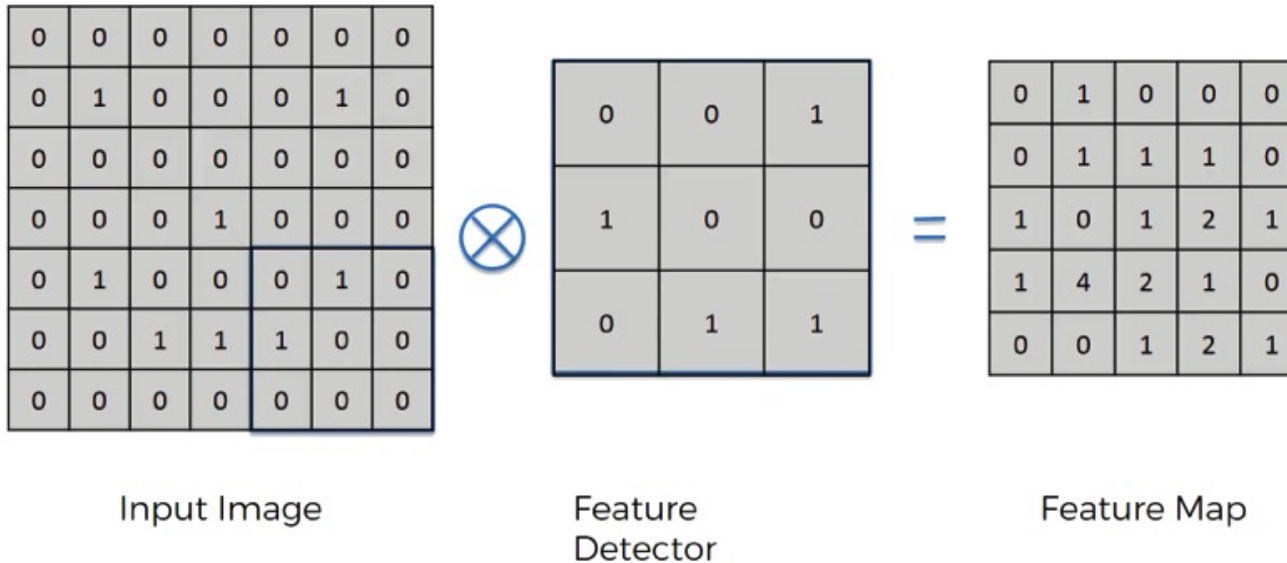
# convolution operation



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

(from [https://github.com/vdumoulin/conv\\_arithmetic](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.



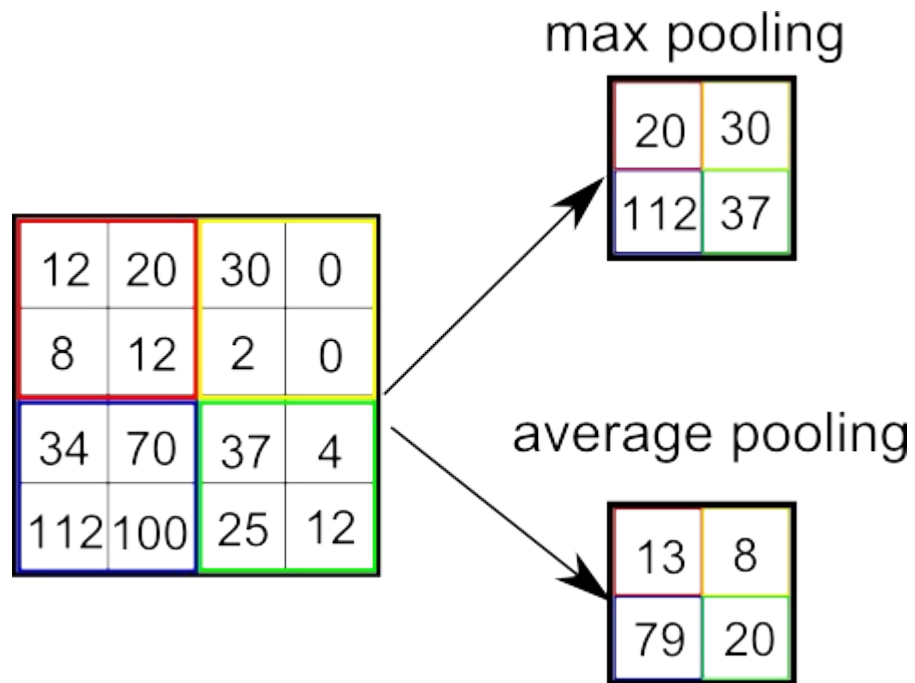
sampling



- sampling can reduce computation cost while keeping necessary information

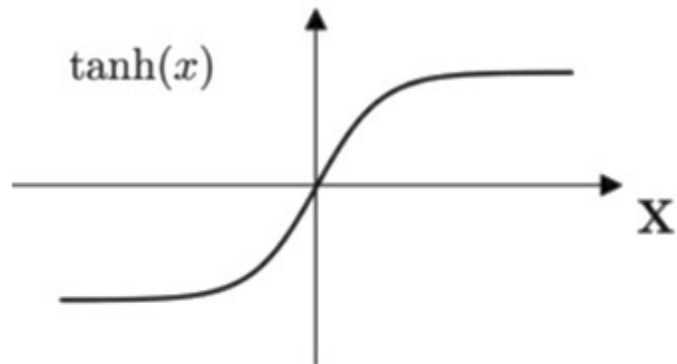
# pooling

- how to sample?

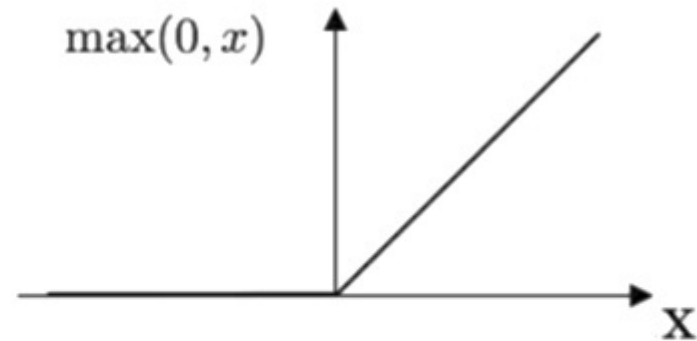


# activation function

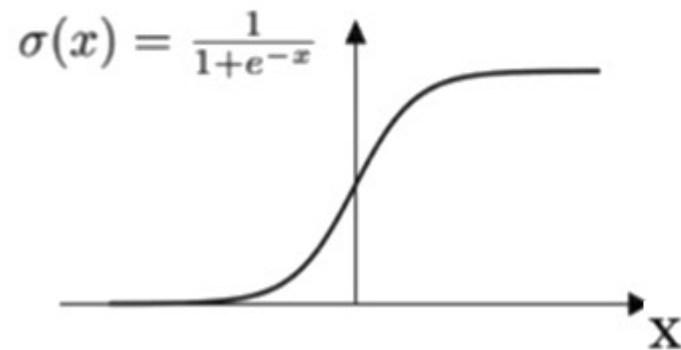
**Tanh**



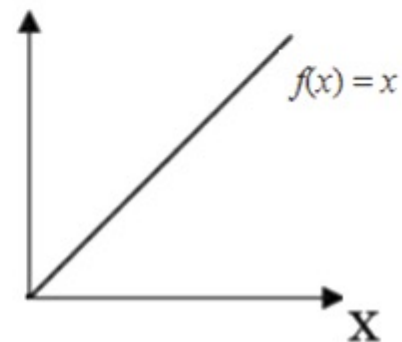
**ReLU**



**Sigmoid**



**Linear**



# 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

$$\text{softmax}(y)_i = \frac{\exp(y_i)}{\sum_j \exp(y_j)}$$

- 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**

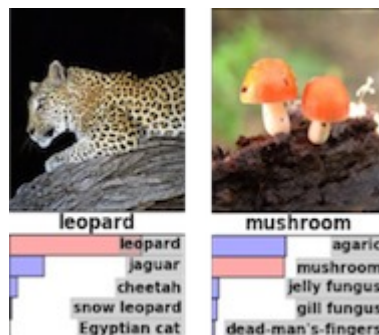
# the first deep learning on ImageNet data: AlexNet

## ImageNet

- ImageNet data set

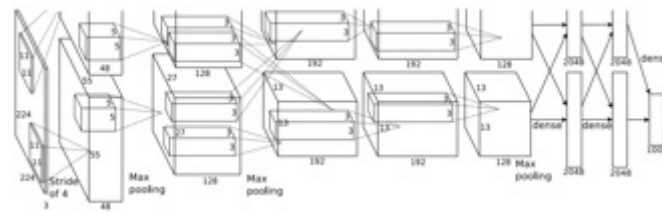


## Recognition task (Top-5)



## 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

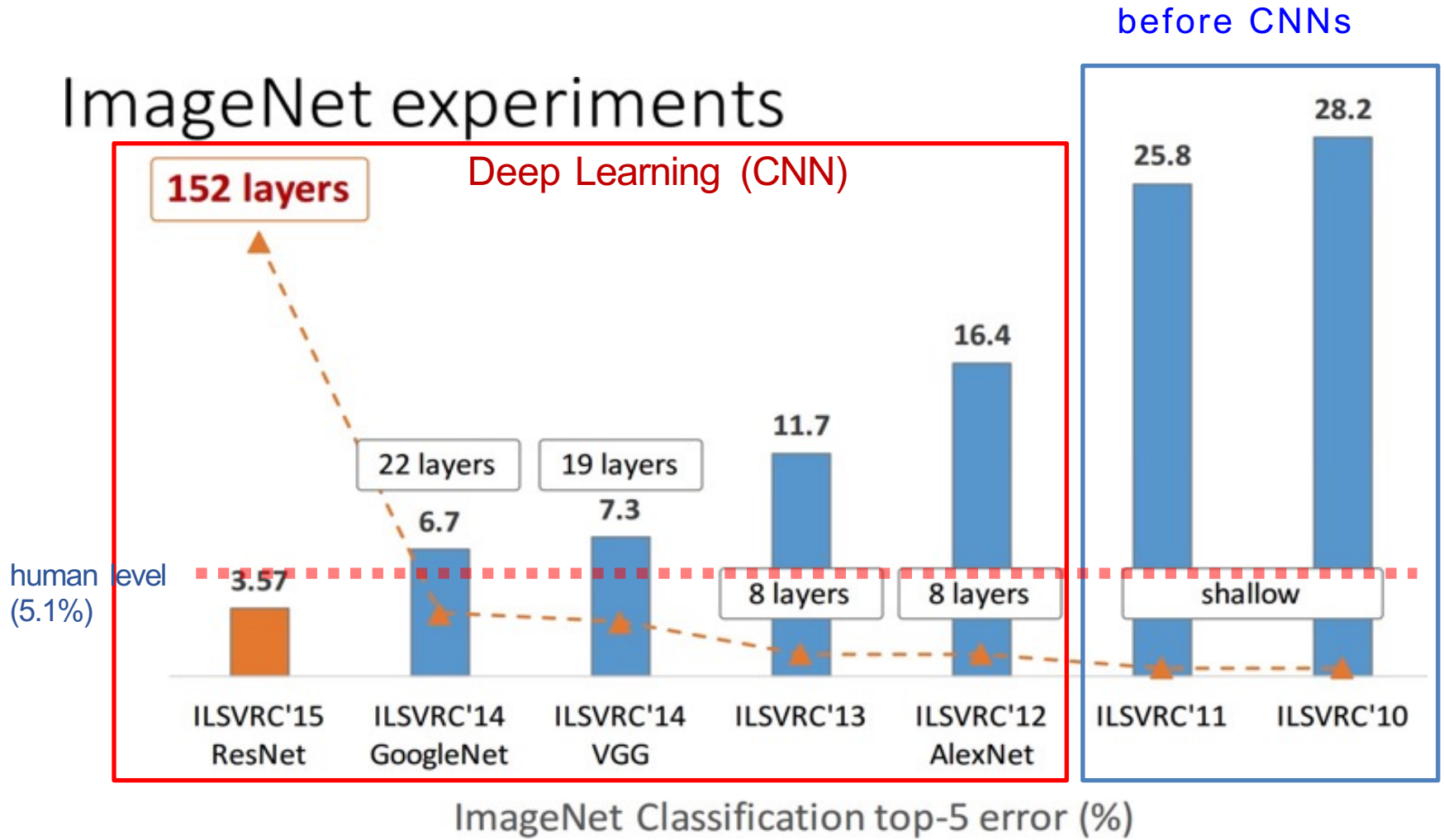
(Krizhevsky et al 2012)

## 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

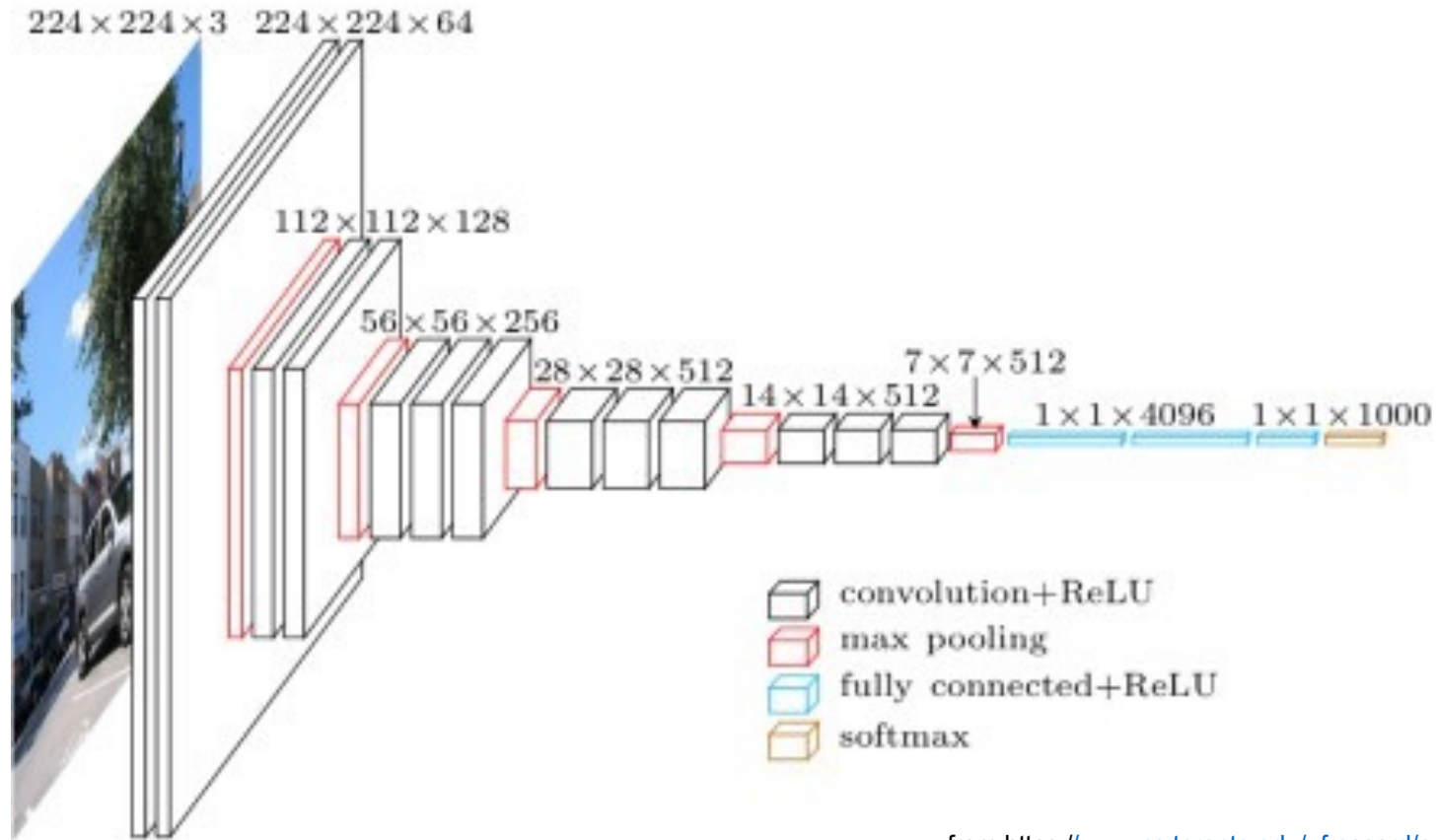
## ImageNet experiments



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

# VGG Network

- simple architecture and very popular.

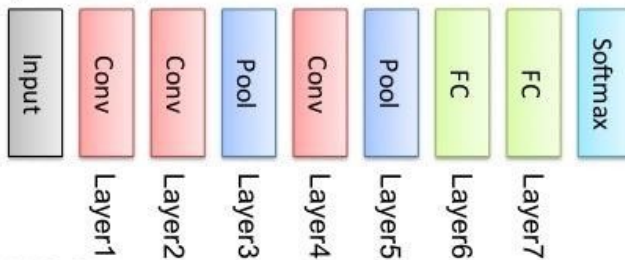


from <https://www.cs.toronto.edu/~frossard/post/vgg16/>

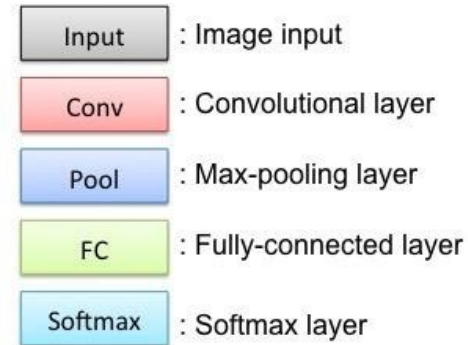
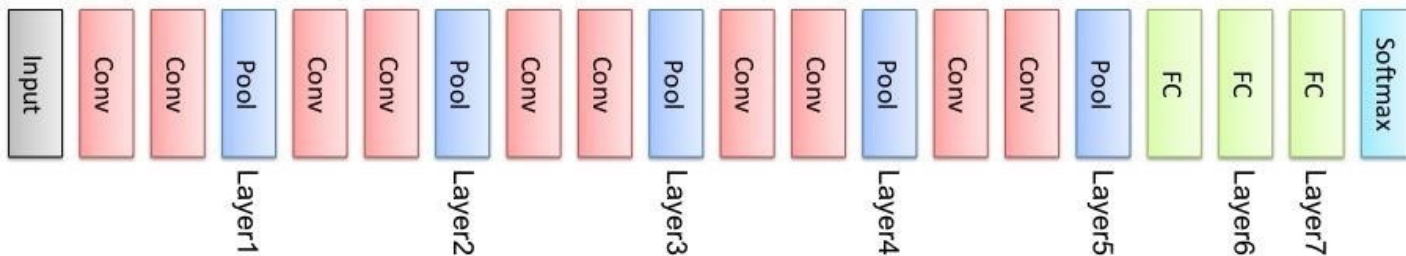


# VGG Network

AlexNet



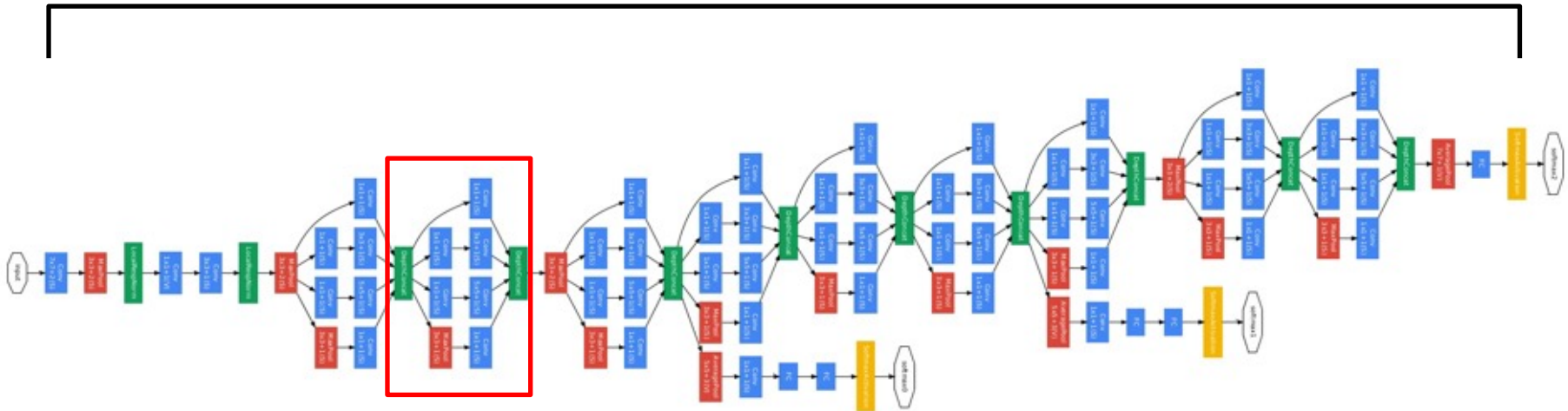
VGGNet



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

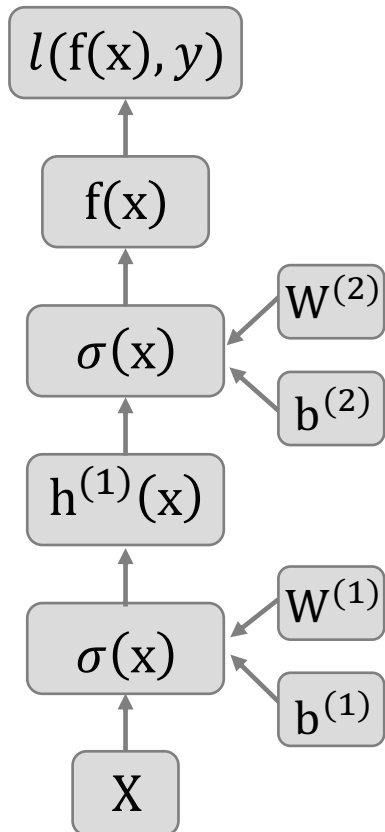
```
>> pip install torch  
>> pip install torchvision  
>> pip install scikit-image
```

- 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).
```

# 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.



data



results



Practice\_ImageNet\_student.ipynb



Practice\_MNIST\_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

```
1 from __future__ import print_function
2 import argparse
3 import torch
4 import torch.nn as nn
5 import torch.nn.functional as F
6 import torch.optim as optim
7 from torchvision import datasets, transforms
```

# Models

```
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)
```

```
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

```
1 def train(model, device, train_loader, optimizer, epoch, log_interval):
2     model.train()
3     for batch_idx, (data, target) in enumerate(train_loader):
4         data, target = data.to(device), target.to(device)
5         optimizer.zero_grad()
6         output = model(data)
7         loss = F.nll_loss(output, target)
8         loss.backward()
9         optimizer.step()
10
11         if batch_idx % log_interval == 0:
12             print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
13                 epoch, batch_idx * len(data), len(train_loader.dataset),
14                 100. * batch_idx / len(train_loader), loss.item()))
15
16             torch.save(model.state_dict(), "./results/mnist_cnn.pt")
17
18
19 def test(model, device, test_loader):
20     model.eval()
21     test_loss = 0
22     correct = 0
23     with torch.no_grad():
24         for data, target in test_loader:
25             data, target = data.to(device), target.to(device)
26             output = model(data)
27
28             # sum up batch loss
29             test_loss += F.nll_loss(output, target, reduction='sum').item()
30
31             # get the index of the max log-probability
32             pred = output.argmax(dim=1, keepdim=True)
33             correct += pred.eq(target.view_as(pred)).sum().item()
34
35     test_loss /= len(test_loader.dataset)
36
37     print('\nTest: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
38         test_loss, correct, len(test_loader.dataset),
39         100. * correct / len(test_loader.dataset)))
```

# Parameters and Data load

```
1 seed = 1
2 epochs = 2
3 batch_size = 32
4 test_batch_size = 1000
5 lr = 0.001 # learning rate
6 momentum = 0.9
7 log_interval = 200
8 save_model = True
9
10 use_cuda = torch.cuda.is_available()
11 device = torch.device("cuda" if use_cuda else "cpu")
12 kwargs = {'num_workers': 1, 'pin_memory': True} if use_cuda else {}
13
14 transform=transforms.Compose([
15     transforms.ToTensor(),
16     transforms.Normalize((0.1307,), (0.3081,)) ])
17
18 train_loader = torch.utils.data.DataLoader(
19     datasets.MNIST('./data', train=True, download=True, transform=transform),
20     batch_size=batch_size, shuffle=True, **kwargs)
21
22 test_loader = torch.utils.data.DataLoader(
23     datasets.MNIST('./data', train=False, transform=transform),
24     batch_size=test_batch_size, shuffle=True, **kwargs)
25
```



# NetFF Model training/testing

```
1 torch.manual_seed(seed)
2
3 model = NetFF().to(device)
4 optimizer = optim.SGD(model.parameters(), lr=lr, momentum=momentum)
5
6 for epoch in range(1, epochs + 1):
7     train(model, device, train_loader, optimizer, epoch, log_interval)
8     test(model, device, test_loader)
9
10 if (save_model):
11     torch.save(model, "./results/mnist_NetFF.pth")
```

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

```
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

```
1 torch.manual_seed(seed)
2
3 model = NetCNN().to(device)
4 optimizer = optim.SGD(model.parameters(), lr=lr, momentum=momentum)
5
6 for epoch in range(1, epochs + 1):
7     train(model, device, train_loader, optimizer, epoch, log_interval)
8     test(model, device, test_loader)
9
10 if (save_model):
11     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 available, !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
```

< > mnist\_test\_images



test0.jpg



test1.jpg



test2.jpg



test3.jpg



test4.jpg



test5.jpg



test6.jpg



test11.jpg



test12.jpg



test13.jpg



test14.jpg



test15.jpg



test16.jpg



test17.jpg

# 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

```
1 !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) without downloading from the server.

\* Please do not run this code for now.

```
1 import torch
2 import pretrainedmodels
3 import pretrainedmodels.utils as utils
4
5 model_name = 'nasnetalarge' # could be fbresnet152 or inceptionresnetv2
6 model = pretrainedmodels.__dict__[model_name](num_classes=1000, pretrained='imagenet')
7 model.eval()
8
9 load_img = utils.LoadImage()
10
11 # transformations depending on the model
12 # rescale, center crop, normalize, and others (ex: ToBGR, ToRange255)
13 tf_img = utils.TransformImage(model)
14
15 # save the model for later
16 torch.save(model, './results/imagenet_nasnetalarge.pth')
```

# ImageNet class names

```

1 import csv
2 name_file = './data/imagenet_install/class_names.csv'
3
4 imagenet_class = {}
5 file_in = csv.reader(open(name_file))
6 for row in file_in:
7     imagenet_class[int(row[0])] = row[1]
8
9 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',
19: 'hobby',
20: 'kingfisher, Haliaeetus leucorhynchus',
21: 'mallard, mallard duck',
22: 'little blue heron, Ardea herodias',
23: 'great egret, Ardea alba',
24: 'great white egret, Ardea alba',
25: 'baldpate, Bucephala albeola',
26: 'trumpeter, Puffinus puffinus',
27: 'booby, Sula nebouxi',
28: 'red-tailed tropicbird, Tropicus termis',
29: 'red-footed booby, Sula nebouxi',
30: 'penguin, Spheniscus patagonicus',
31: 'fairy tern, Sterna bergii',
32: 'black noddie, Sturnella bergii',
33: 'great frigatebird, Stercorarius macurus',
34: 'lesser frigatebird, Stercorarius fuscus',
35: 'brown booby, Sula leucogaster',
36: 'red booby, Sula sula',
37: 'white booby, Sula leucogaster',
38: 'black booby, Sula dactylatra',
39: 'blue booby, Sula nebouxi',
40: 'brown booby, Sula leucogaster',
41: 'red booby, Sula sula',
42: 'white booby, Sula leucogaster',
43: 'black booby, Sula dactylatra',
44: 'blue booby, Sula nebouxi',
45: 'brown booby, Sula leucogaster',
46: 'red booby, Sula sula',
47: 'white booby, Sula leucogaster',
48: 'black booby, Sula dactylatra',
49: 'blue booby, Sula nebouxi',
50: 'brown booby, Sula leucogaster',
51: 'red booby, Sula sula',
52: 'white booby, Sula leucogaster',
53: 'black booby, Sula dactylatra',
54: 'blue booby, Sula nebouxi',
55: 'brown booby, Sula leucogaster',
56: 'red booby, Sula sula',
57: 'white booby, Sula leucogaster',
58: 'black booby, Sula dactylatra',
59: 'blue booby, Sula nebouxi',
60: 'brown booby, Sula leucogaster',
61: 'red booby, Sula sula',
62: 'white booby, Sula leucogaster',
63: 'black booby, Sula dactylatra',
64: 'blue booby, Sula nebouxi',
65: 'brown booby, Sula leucogaster',
66: 'red booby, Sula sula',
67: 'white booby, Sula leucogaster',
68: 'black booby, Sula dactylatra',
69: 'blue booby, Sula nebouxi',
70: 'brown booby, Sula leucogaster',
71: 'red booby, Sula sula',
72: 'white booby, Sula leucogaster',
73: 'black booby, Sula dactylatra',
74: 'blue booby, Sula nebouxi',
75: 'brown booby, Sula leucogaster',
76: 'red booby, Sula sula',
77: 'white booby, Sula leucogaster',
78: 'black booby, Sula dactylatra',
79: 'blue booby, Sula nebouxi',
80: 'brown booby, Sula leucogaster',
81: 'red booby, Sula sula',
82: 'white booby, Sula leucogaster',
83: 'black booby, Sula dactylatra',
84: 'blue booby, Sula nebouxi',
85: 'brown booby, Sula leucogaster',
86: 'red booby, Sula sula',
87: 'white booby, Sula leucogaster',
88: 'black booby, Sula dactylatra',
89: 'blue booby, Sula nebouxi',
90: 'brown booby, Sula leucogaster',
91: 'red booby, Sula sula',
92: 'white booby, Sula leucogaster',
93: 'black booby, Sula dactylatra',
94: 'blue booby, Sula nebouxi',
95: 'brown booby, Sula leucogaster',
96: 'red booby, Sula sula',
97: 'white booby, Sula leucogaster',
98: 'black booby, Sula dactylatra',
99: 'blue booby, Sula nebouxi',
100: 'brown booby, Sula leucogaster',
101: 'red booby, Sula sula',
102: 'white booby, Sula leucogaster',
103: 'black booby, Sula dactylatra',
104: 'blue booby, Sula nebouxi',
105: 'brown booby, Sula leucogaster',
106: 'red booby, Sula sula',
107: 'white booby, Sula leucogaster',
108: 'black booby, Sula dactylatra',
109: 'blue booby, Sula nebouxi',
110: 'brown booby, Sula leucogaster',
111: 'red booby, Sula sula',
112: 'white booby, Sula leucogaster',
113: 'black booby, Sula dactylatra',
114: 'blue booby, Sula nebouxi',
115: 'brown booby, Sula leucogaster',
116: 'red booby, Sula sula',
117: 'white booby, Sula leucogaster',
118: 'black booby, Sula dactylatra',
119: 'blue booby, Sula nebouxi',
120: 'brown booby, Sula leucogaster',
121: 'red booby, Sula sula',
122: 'white booby, Sula leucogaster',
123: 'black booby, Sula dactylatra',
124: 'blue booby, Sula nebouxi',
125: 'brown booby, Sula leucogaster',
126: 'red booby, Sula sula',
127: 'white booby, Sula leucogaster',
128: 'black booby, Sula dactylatra',
129: 'blue booby, Sula nebouxi',
130: 'brown booby, Sula leucogaster',
131: 'red booby, Sula sula',
132: 'white booby, Sula leucogaster',
133: 'black booby, Sula dactylatra',
134: 'blue booby, Sula nebouxi',
135: 'brown booby, Sula leucogaster',
136: 'red booby, Sula sula',
137: 'white booby, Sula leucogaster',
138: 'black booby, Sula dactylatra',
139: 'blue booby, Sula nebouxi',
140: 'brown booby, Sula leucogaster',
141: 'red booby, Sula sula',
142: 'white booby, Sula leucogaster',
143: 'black booby, Sula dactylatra',
144: 'blue booby, Sula nebouxi',
145: 'brown booby, Sula leucogaster',
146: 'red booby, Sula sula',
147: 'white booby, Sula leucogaster',
148: 'black booby, Sula dactylatra',
149: 'blue booby, Sula nebouxi',
150: 'brown booby, Sula leucogaster',
151: 'red booby, Sula sula',
152: 'white booby, Sula leucogaster',
153: 'black booby, Sula dactylatra',
154: 'blue booby, Sula nebouxi',
155: 'brown booby, Sula leucogaster',
156: 'red booby, Sula sula',
157: 'white booby, Sula leucogaster',
158: 'black booby, Sula dactylatra',
159: 'blue booby, Sula nebouxi',
160: 'brown booby, Sula leucogaster',
161: 'red booby, Sula sula',
162: 'white booby, Sula leucogaster',
163: 'black booby, Sula dactylatra',
164: 'blue booby, Sula nebouxi',
165: 'brown booby, Sula leucogaster',
166: 'red booby, Sula sula',
167: 'white booby, Sula leucogaster',
168: 'black booby, Sula dactylatra',
169: 'blue booby, Sula nebouxi',
170: 'brown booby, Sula leucogaster',
171: 'red booby, Sula sula',
172: 'white booby, Sula leucogaster',
173: 'black booby, Sula dactylatra',
174: 'blue booby, Sula nebouxi',
175: 'brown booby, Sula leucogaster',
176: 'red booby, Sula sula',
177: 'white booby, Sula leucogaster',
178: 'black booby, Sula dactylatra',
179: 'blue booby, Sula nebouxi',
180: 'brown booby, Sula leucogaster',
181: 'red booby, Sula sula',
182: 'white booby, Sula leucogaster',
183: 'black booby, Sula dactylatra',
184: 'blue booby, Sula nebouxi',
185: 'brown booby, Sula leucogaster',
186: 'red booby, Sula sula',
187: 'white booby, Sula leucogaster',
188: 'black booby, Sula dactylatra',
189: 'blue booby, Sula nebouxi',
190: 'brown booby, Sula leucogaster',
191: 'red booby, Sula sula',
192: 'white booby, Sula leucogaster',
193: 'black booby, Sula dactylatra',
194: 'blue booby, Sula nebouxi',
195: 'brown booby, Sula leucogaster',
196: 'red booby, Sula sula',
197: 'white booby, Sula leucogaster',
198: 'black booby, Sula dactylatra',
199: 'blue booby, Sula nebouxi',
200: 'brown booby, Sula leucogaster',
201: 'red booby, Sula sula',
202: 'white booby, Sula leucogaster',
203: 'black booby, Sula dactylatra',
204: 'blue booby, Sula nebouxi',
205: 'brown booby, Sula leucogaster',
206: 'red booby, Sula sula',
207: 'white booby, Sula leucogaster',
208: 'black booby, Sula dactylatra',
209: 'blue booby, Sula nebouxi',
210: 'brown booby, Sula leucogaster',
211: 'red booby, Sula sula',
212: 'white booby, Sula leucogaster',
213: 'black booby, Sula dactylatra',
214: 'blue booby, Sula nebouxi',
215: 'brown booby, Sula leucogaster',
216: 'red booby, Sula sula',
217: 'white booby, Sula leucogaster',
218: 'black booby, Sula dactylatra',
219: 'blue booby, Sula nebouxi',
220: 'brown booby, Sula leucogaster',
221: 'red booby, Sula sula',
222: 'white booby, Sula leucogaster',
223: 'black booby, Sula dactylatra',
224: 'blue booby, Sula nebouxi',
225: 'brown booby, Sula leucogaster',
226: 'red booby, Sula sula',
227: 'white booby, Sula leucogaster',
228: 'black booby, Sula dactylatra',
229: 'blue booby, Sula nebouxi',
230: 'brown booby, Sula leucogaster',
231: 'red booby, Sula sula',
232: 'white booby, Sula leucogaster',
233: 'black booby, Sula dactylatra',
234: 'blue booby, Sula nebouxi',
235: 'brown booby, Sula leucogaster',
236: 'red booby, Sula sula',
237: 'white booby, Sula leucogaster',
238: 'black booby, Sula dactylatra',
239: 'blue booby, Sula nebouxi',
240: 'brown booby, Sula leucogaster',
241: 'red booby, Sula sula',
242: 'white booby, Sula leucogaster',
243: 'black booby, Sula dactylatra',
244: 'blue booby, Sula nebouxi',
245: 'brown booby, Sula leucogaster',
246: 'red booby, Sula sula',
247: 'white booby, Sula leucogaster',
248: 'black booby, Sula dactylatra',
249: 'blue booby, Sula nebouxi',
250: 'brown booby, Sula leucogaster',
251: 'red booby, Sula sula',
252: 'white booby, Sula leucogaster',

```

# classification with images

in the directory './data/images/'

< > images



cat\_2.jpg



cat\_224.jpg



IMG\_7543.JPG



KakaoTalk\_20191  
010\_180...925.jpg



# classification of a single image

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

1  import torch
2
3  try:
4      model # does exist
5  except NameError: # model does not exist
6      import pretrainedmodels.utils as utils
7      model = torch.load('./results/imagenet_nasnetalarge.pth')
8      load_img = utils.LoadImage()
9      tf_img = utils.TransformImage(model)
10
11  # your file name
12  img_file = './data/images/cat_224.jpg'
13
14  input_img = load_img(img_file)
15  input_tensor1 = tf_img(input_img)
16  input_tensor2 = input_tensor1.unsqueeze(0)
17
18  output_logits = model(input_tensor2) # 1x1000
19
20  print("{} is {}: {}".format(img_file ,output_logits.argmax(),
21                               imagenet_class[int(output_logits.argmax())]))
```

./data/images/cat\_224.jpg is [281: tabby, tabby cat]

# classification of many images in a directory

```
1  try:
2      model # does exist
3  except NameError: # model does not exist
4      import pretrainedmodels.utils as utils
5      model = torch.load('./results/imagenet_nasnetalarge.pth')
6      load_img = utils.LoadImage()
7      tf_img = utils.TransformImage(model)
8
9  import glob
10 dir_path = './data/images/'
11 img_list = glob.glob(dir_path+'*.*')
12
13 for img_file in img_list:
14     input_img = load_img(img_file)
15     input_tensor1 = tf_img(input_img)
16     input_tensor2 = input_tensor1.unsqueeze(0)
17     output_logits = model(input_tensor2) # 1x1000
18
19     print("{} is [{}: {}]".format(img_file.split('/')[-1], output_logits.argmax(),
20                                   imagenet_class[int(output_logits.argmax())]))
```

imagesWcat\_2.jpg is [282: tiger cat]

imagesWcat\_224.jpg is [281: tabby, tabby cat]

imagesWIMG\_7543.JPG is [559: folding chair]

imagesWKakaoTalk\_20191010\_180310925.jpg is [504: coffee mug]

# Practice: classify your own images

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