Deep Learning for Computer Vision

Libao Zhang, Jie Ma

2017.11.28

Content

- **Neural Network**
- **Convolution Neural Network**
- **Deep Learning for Computer Vision**

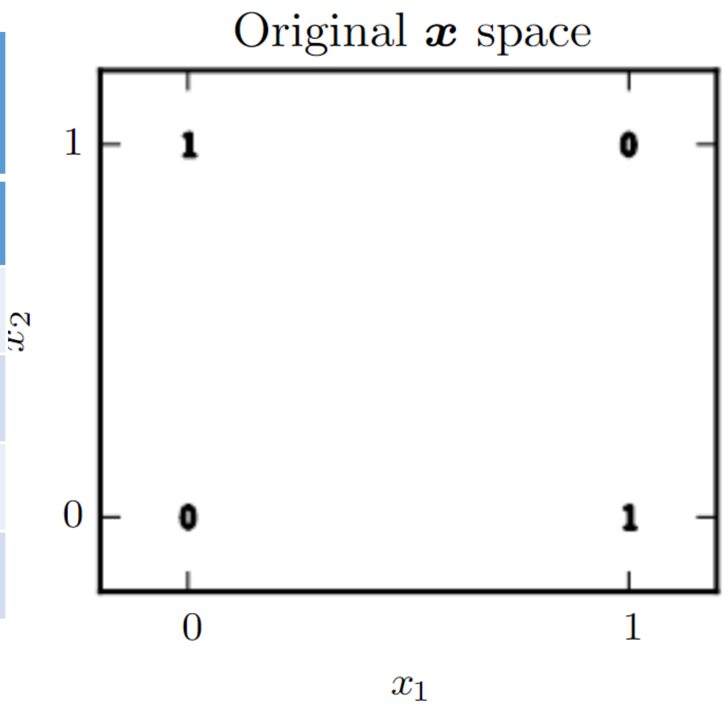
Content

- **Neural Network**
  - **Discussion from XOR**
  - **Forward Propagation**
  - **Back Propagation**
- **Convolution Neural Network**
- **Deep Learning for Computer Vision**

Neural Network: Discussion from XOR

- The XOR function ("exclusive or") is an operation on two binary values, x_1 and x_2. When exactly one of these binary values is equal to 1, the XOR function returns 1. Otherwise, it returns 0.

Input		Target Output
x_1	x_2	y
0	0	0
1	1	0
1	0	1
0	1	1



Neural Network: Discussion from XOR

- We can treat this problem as a regression problem and use a mean squared error(MSE) loss function.
- Target function:

$$y = f^*(x),$$

where $f^*(1,0) = 1, f^*(0,1) = 1, f^*(1,1) = 0, f^*(0,0) = 0$

- Our model:

$$y = f(x, \vartheta)$$

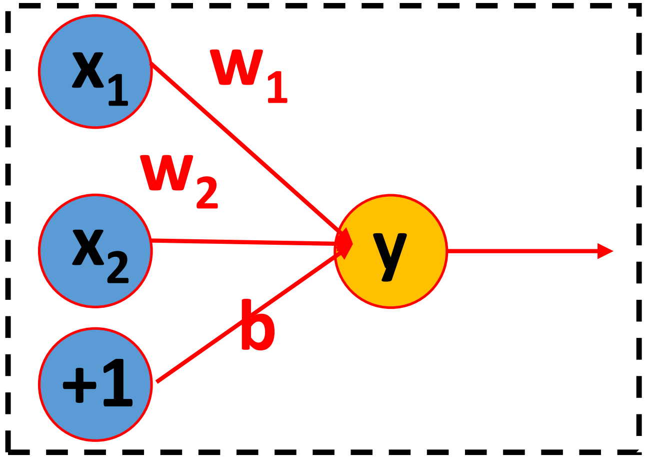
- MES loss function:

$$J(\theta) = \frac{1}{4}\left(\sum_{i=1}^{4}[f(x_i, \vartheta) - f^*(x_i)]^2\right)$$

Neural Network: Discussion from XOR

- Suppose that we choose a linear model, with θ consisting of w and b. Our model is defined to be:

$$f(x; w, b) = x_1 * w_1 + x_2 * w_2 + b$$
$$= (w_1, w_2) * \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \mathrm{b}$$
$$= Wx + b$$



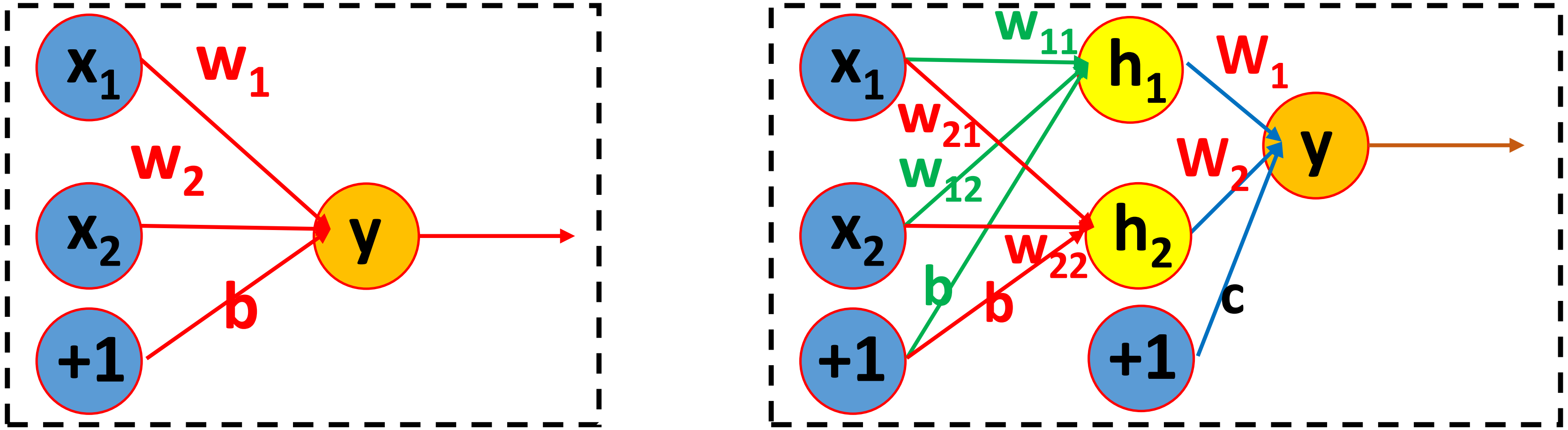
- Minimize $J(\theta)$, we obtain $\mathrm{W} = 0, b = \frac{1}{2}$

Input		Target Output	Regression by linear model
x_1	x_2	y^*	y
0	0	0	0.5
1	1	0	0.5
1	0	1	0.5
0	1	1	0.5

By Libao Zhang, Jie Ma

Neural Network: Discussion from XOR

- We must use a **nonlinear function** to describe the features.



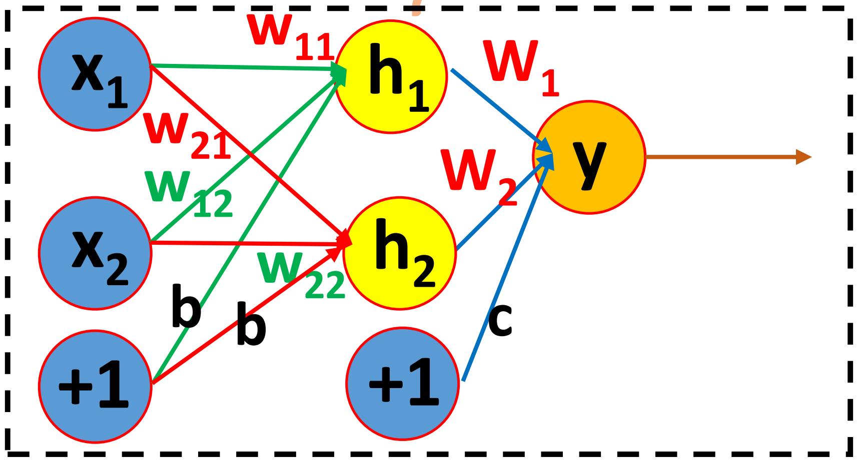
- The network now contains two function:

$$\begin{bmatrix} h1 \\ h2 \end{bmatrix} = [x_1 * w_{11} + x_2 * w_{12}, x_1 * w_{21} + x_2 * w_{22}]^T + b$$

$$= \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + b = wx + b$$

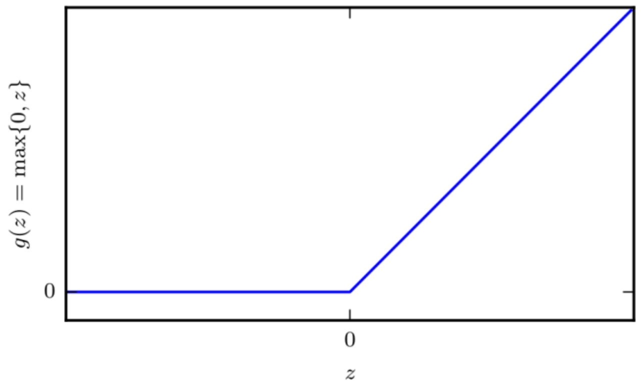
$$y = h1 * W_1 + h_2 * W_2 + c = \mathbf{Wh} + \mathbf{c}$$

Neural Network: Discussion from XOR



Rectified linear unit

$g(z) = max\{0, z\}$



- **The whole model:**

$$f(x; W, c, w, b) = W * max\{0, w * x + b\} + c$$

- **We can then specify a solution to the XOR problem:**

$$w = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \quad W = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

$$b = \begin{bmatrix} 0 \\ -1 \end{bmatrix} \qquad c = 0$$

Input		Target Output	Regression by linear model
x_1	x_2	y^*	y
0	0	0	0
1	1	0	0
1	0	1	1
0	1	1	1

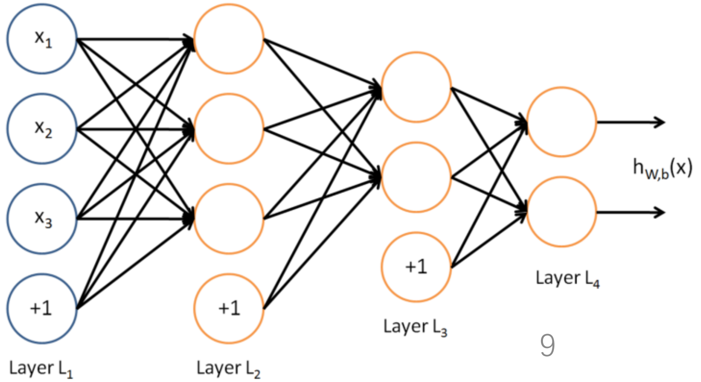
By Libao Zhang, Jie Ma

Neural Network: Feedforward Propagation

- There are $\boldsymbol{n_l}$ neurons in layer $\boldsymbol{l}$, the output of layer $\boldsymbol{l}$ is:

$$\boldsymbol{a}^{(l)} = [\boldsymbol{a}_1^{(l)}, \boldsymbol{a}_2^{(l)}, \cdots, \boldsymbol{a}_{n_l}^{(l)}]^T$$

- Denote the weight associated with the connection between unit $\boldsymbol{j}$ in layer $\boldsymbol{l}$, and unit $\boldsymbol{i}$ in layer $\boldsymbol{l+1}$: $\boldsymbol{W}^{(l)} = \left[\boldsymbol{w}_{ij}^{(l)}\right]_{n_{l+1} * n_l}$
- The bias associated with unit $\boldsymbol{i}$ in layer $\boldsymbol{l+1}$: $\boldsymbol{b}_i^{(l)}$

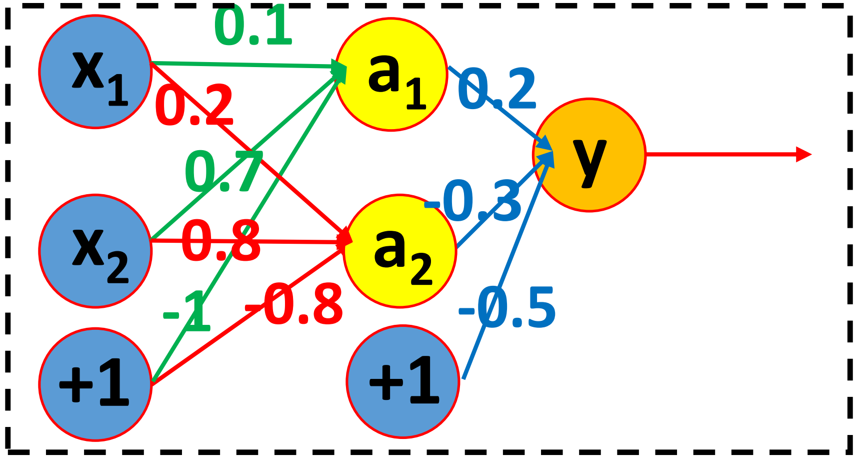


Neural Network: Feedforward propagation

- **Feedforward propagation algorithm:**

$$\boldsymbol{z}^{(l+1)} = \boldsymbol{W}^{(l)}\boldsymbol{a}^{(l)} + \boldsymbol{b}^{(l)}$$

$$\boldsymbol{a}^{(l+1)} = \boldsymbol{f}(\boldsymbol{z}^{(l+1)})$$



- **Example:**

Input: $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$

Layer1: $W^{(1)} = \begin{bmatrix} 0.1 & 0.7 \\ 0.2 & 0.8 \end{bmatrix}$, $b^{(1)} = \begin{bmatrix} -1 \\ -0.8 \end{bmatrix}$

Layer2: $W^{(2)} = [0.2 \quad -0.3]$, $b^{(2)} = -0.5$

- **Feedforward propagation:**

$$z^{(2)} = W^{(1)}a^{(1)} + b^{(1)} = \begin{bmatrix} 0.1 & 0.7 \\ 0.2 & 0.8 \end{bmatrix} * \begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} -1 \\ -0.8 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 1 \end{bmatrix}$$

$$a^{(2)} = f\left(z^{(2)}\right) = \begin{bmatrix} 0.5 \\ 1 \end{bmatrix}$$

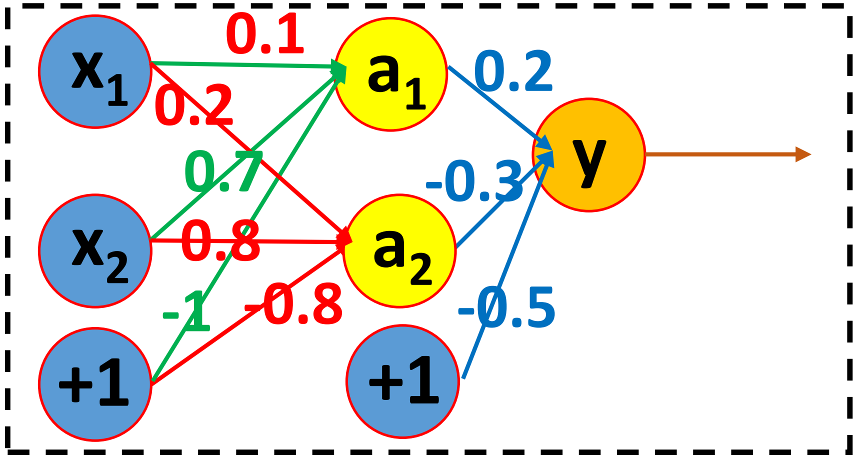
$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)} = [0.2 \ -0.3] * \begin{bmatrix} 0.5 \\ 1 \end{bmatrix} + (-0.5) = -0.7$$

$$a^{(3)} = f\left(z^{(3)}\right) = 0$$

Neural Network: Back Propagation

- **Question:**

How to obtain the appropriate parameters?



Neural Network: Feedforward propagation

- For a regression or classification task, we have dataset with **input feature** and **ground truth**.
- Divide the dataset into 2 parts, one for **training** and one for **testing**.
- In **training set** $\boldsymbol{D}$, we want to minimize the difference between **the label of training data and the model's predictions.**
- MSE loss function:

$$J(\theta) = \sum_{i \in D} \frac{1}{2} ||f(x_i; \theta) - y_i||^2$$

- Cross-entropy loss function:

$$J(\theta) = -\sum_{i \in D} \sum_{j=1}^{k} 1\{y_i = k\} ln\left(f_k(x_i; \theta)\right)$$

By Libao Zhang, Jie Ma

Neural Network: Feedforward propagation

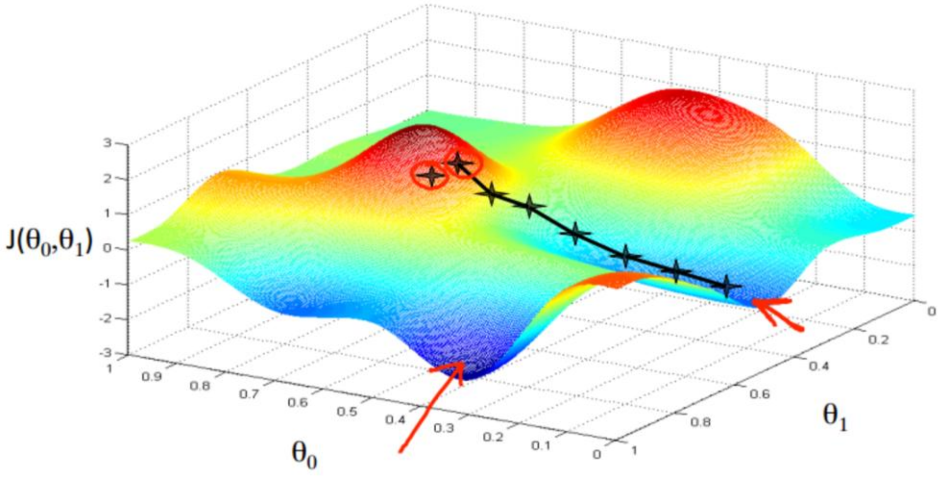
- It's very difficult to find the **minimum** by traditional **gradient-based** method.

So we utilize **gradient descent algorithm:**

$$\theta^{(k+1)} = \theta^{(k)} - \boxed{\alpha}\,\boxed{\left.\frac{\partial J(\theta)}{\partial \theta}\right|_{\theta=\theta^{(k)}}}$$

gradient

learning rate



Neural Network: Feedforward propagation

gradient descent algorithm

$$j(x_i;\theta) = \frac{1}{2}\|f(x_i;\theta) - y_i\|^2$$

Batch Gradient Descent

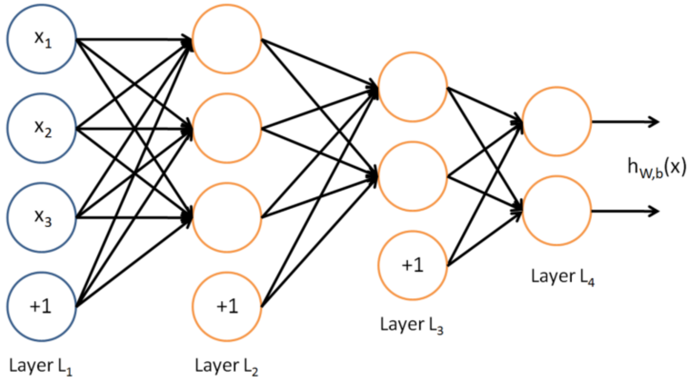
$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \boldsymbol{\alpha} \sum_{i=1}^{N} \frac{\partial \boldsymbol{j}(\boldsymbol{x}_i;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\bigg|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(k)}}$$

Stochastic Gradient Descent

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \boldsymbol{\alpha} \frac{\partial \boldsymbol{j}(\boldsymbol{x}_i;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\bigg|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(k)}}$$

Mini-batch Gradient Descent $\boldsymbol{D} = [\boldsymbol{D}_1, \boldsymbol{D}_2, \cdots, \boldsymbol{D}_M]$

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \boldsymbol{\alpha} \sum_{i \in D_i} \frac{\partial \boldsymbol{j}(\boldsymbol{x}_i;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\bigg|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(k)}}$$



Neural Network: Summary

- Step1 : **Construct** a neural network, confirm the number of neurons in each layer.
- Step2: Confirm the **loss function** : MSE, Cross-entropy.
- Step3: Confirm the **optimization method**: batch size and epochs.
- Step4: **Random initialize** the parameters (weights and bias).
- Step5: Divide the dataset into two part, **train: test**=4:1 or 9:1.
- Step6: **Training** model.
- Step7: **Evaluate** model, by loss or accuracy.

Neural Network: Experiment

```
model = Sequential()
model.add(Dense(512, input_shape=(784,)))
model.add(Activation('relu'))

model.add(Dense(512))
model.add(Activation('relu'))

model.add(Dense(10))
model.add(Activation('softmax'))

model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])

history = model.fit(X_train, Y_train,
                    batch_size = batch_size,
                    nb_epoch = nb_epoch,
                    verbose = 1,
                    validation_data = (X_test, Y_test))

score = model.evaluate(X_test, Y_test, verbose=0)
```

layer1

layer2

layer3

Activation layer

Model summary

Loss function

Optimizer

Training and testing set
Batch size
Epoch

Evaluate

Neural Network: Result

	20 neurons	vs	512neurons
epoch1	loss: 0.5601 - acc: 0.8527 - val_loss: 0.3027 - val_acc: 0.9148		loss: 0.2734 - acc: 0.9208 - val_loss: 0.1291 - val_acc: 0.9622
epoch2	loss: 0.2874 - acc: 0.9183 - val_loss: 0.2617 - val_acc: 0.9256		loss: 0.1179 - acc: 0.9653 - val_loss: 0.1007 - val_acc: 0.9694
epoch3	loss: 0.2540 - acc: 0.9273 - val_loss: 0.2380 - val_acc: 0.9300		loss: 0.0819 - acc: 0.9749 - val_loss: 0.0764 - val_acc: 0.9769
epoch4	loss: 0.2300 - acc: 0.9343 - val_loss: 0.2193 - val_acc: 0.9368		loss: 0.0635 - acc: 0.9810 - val_loss: 0.0757 - val_acc: 0.9763
epoch5	loss: 0.2108 - acc: 0.9395 - val_loss: 0.2152 - val_acc: 0.9363		loss: 0.0507 - acc: 0.9843 - val_loss: 0.0630 - val_acc: 0.9809
epoch6	loss: 0.1963 - acc: 0.9439 - val_loss: 0.1940 - val_acc: 0.9414		loss: 0.0426 - acc: 0.9871 - val_loss: 0.0636 - val_acc: 0.9802
epoch7	loss: 0.1853 - acc: 0.9466 - val_loss: 0.1952 - val_acc: 0.9436		loss: 0.0355 - acc: 0.9891 - val_loss: 0.0633 - val_acc: 0.9816
epoch8	loss: 0.1764 - acc: 0.9488 - val_loss: 0.1835 - val_acc: 0.9460		loss: 0.0325 - acc: 0.9904 - val_loss: 0.0616 - val_acc: 0.9830
epoch9	loss: 0.1691 - acc: 0.9511 - val_loss: 0.1856 - val_acc: 0.9452		loss: 0.0272 - acc: 0.9918 - val_loss: 0.0625 - val_acc: 0.9819
epoch10	loss: 0.1629 - acc: 0.9525 - val_loss: 0.1747 - val_acc: 0.9494		loss: 0.0233 - acc: 0.9925 - val_loss: 0.0640 - val_acc: 0.9816

val_loss: 0.1747 - val_acc: 0.9494

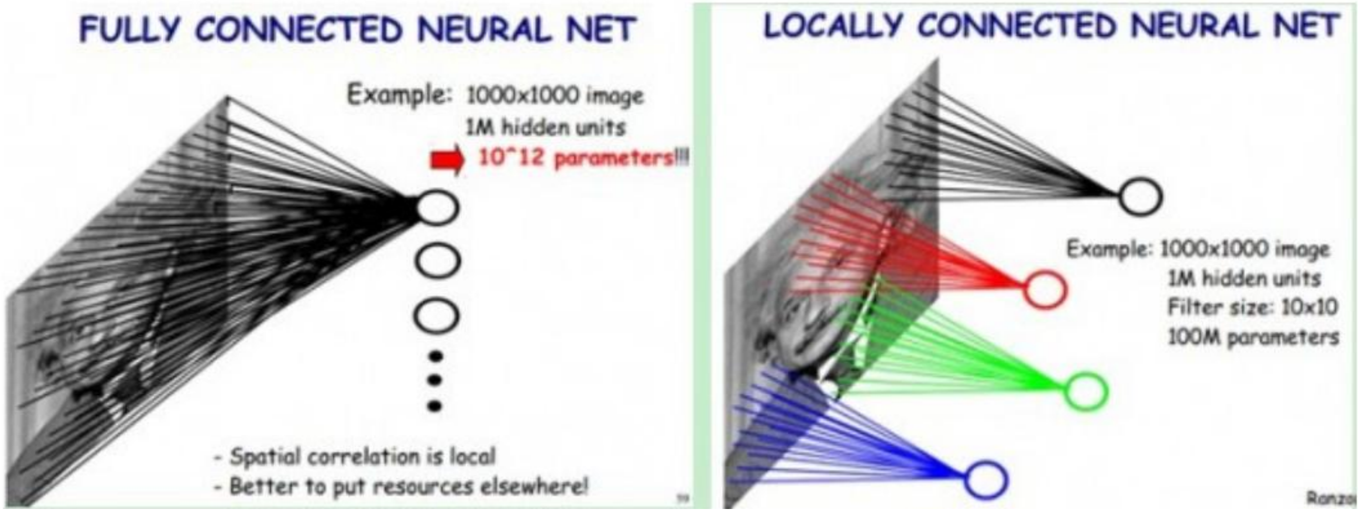
val_loss: 0.0640 - val_acc: 0.9816

By Libao Zhang, Jie Ma

Content

- ~~Neural Network~~
- **Convolution Neural Network**
  - Motivation
  - Convolutional layer
  - Max-pooling layer
  - Classical CNN
- Deep Learning for Computer Vision

Convolutional Neural Network: Motivations

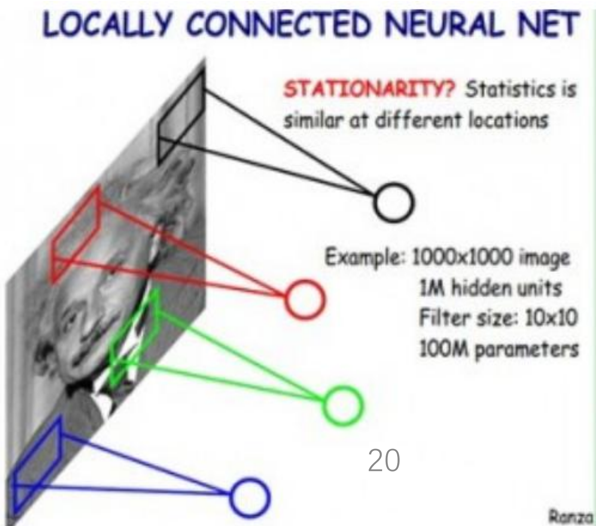


- **There are two approach to reduce the number of parameters:**
- **Receptive field.**
- **Sharing weights.**

Convolutional Neural Network: Convolution layer

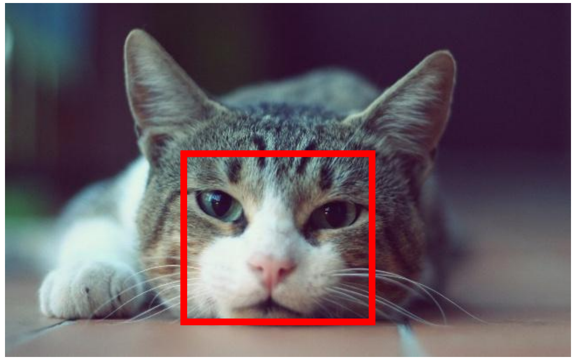
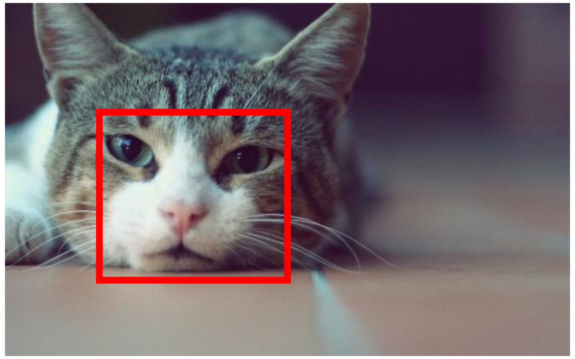
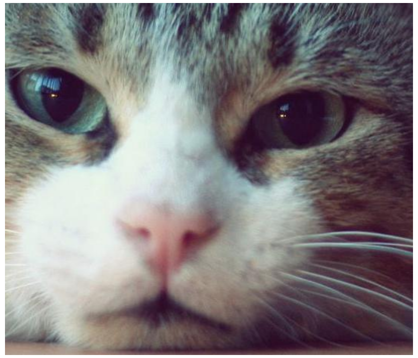
- **Fully connected network:**
- With larger images (e.g., **96x96 images**), there are about **10^4 input units**, and assuming you want to learn **100 features**, you would have on the order of **10^6** parameters to learn.

- **Locally connected network:**
- Each hidden unit will connect **to only a small contiguous region** of pixels in the input.



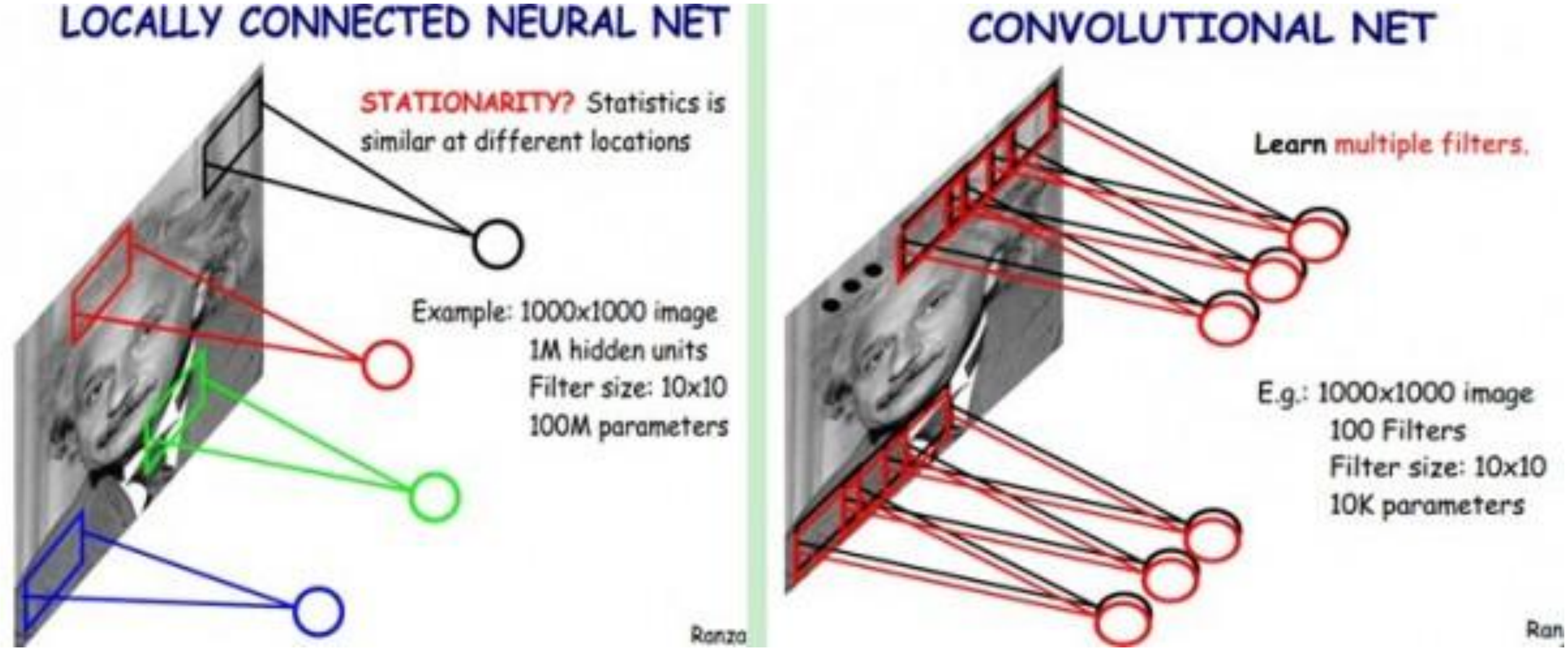
Convolutional Neural Network: Convolution layer

- Natural images have the property of being **stationary**

- The statistics of one part of the image are the **same as any other part**.
- This suggests that the features that we learn at one part of the image can also be applied to other parts of the image, and **we can use the same features at all locations**.

Locally connected



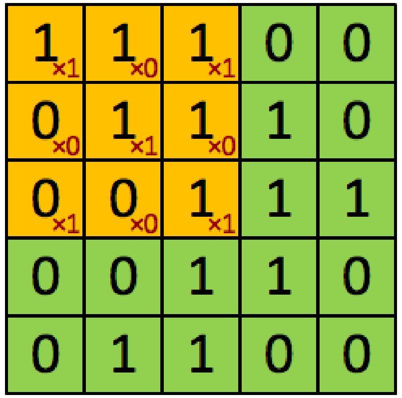
without sharing weights vs **with sharing weights**

Convolution for single-channel:

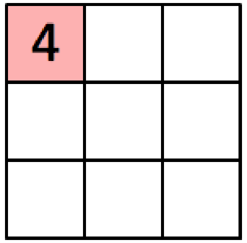
$$f(x,y)\circ w=\sum_{s=-a}^{a}\sum_{t=-b}^{b}w(s,t)\cdot f(x-s,y-t)$$

$\Updownarrow$

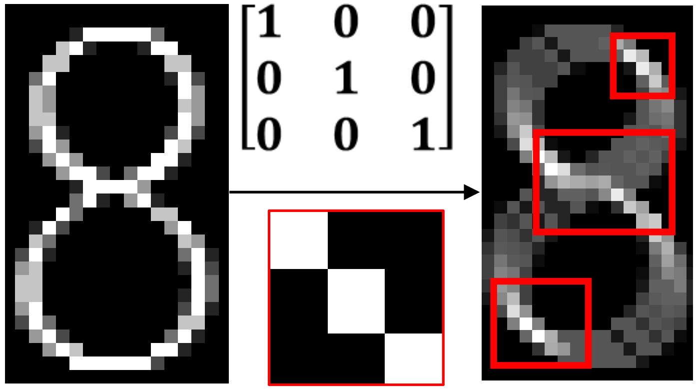
rotate the kernel 180° & correlation



Image



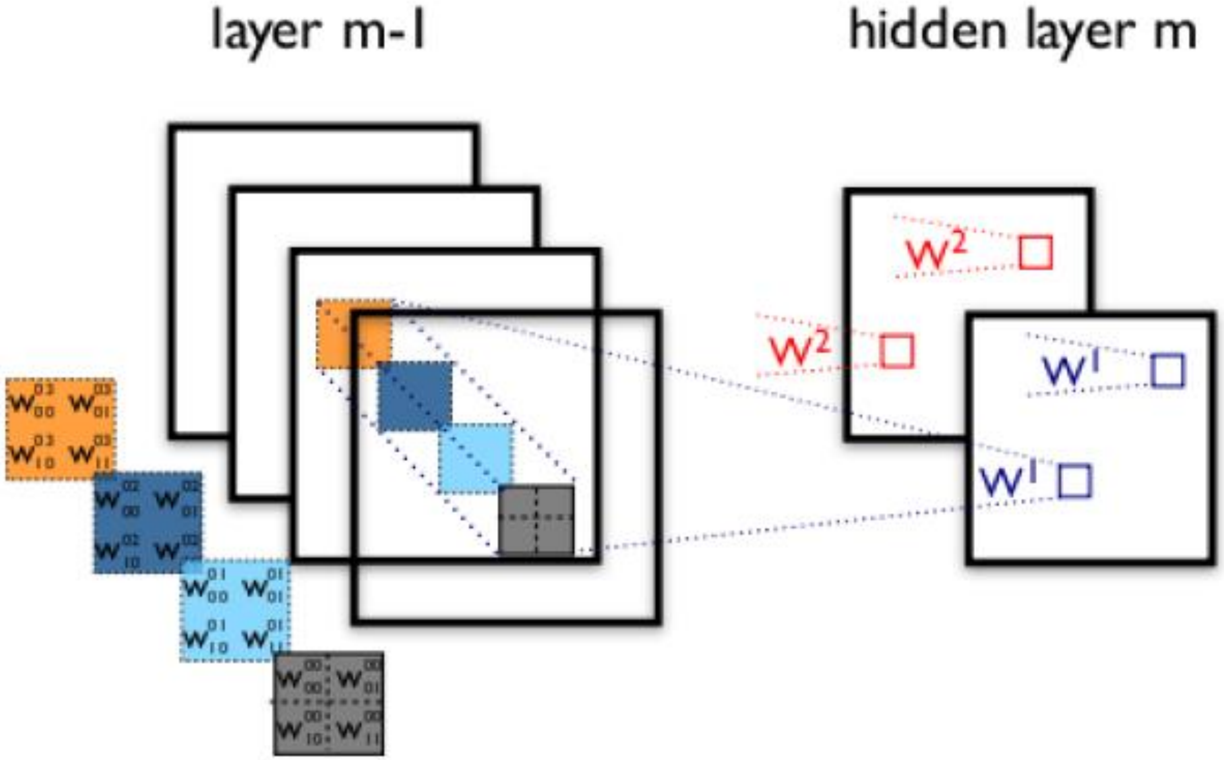
Convolved Feature



By Libao Zhang, Jie

Convolution for multi-channel:

Supposing that there are n_{m-1} channels in layer m-1 and n_m channels in layer m and there are kernels with size $s \times t$,the number of parameters of layer m-1 and layer m is $n_{m-1} \times n_m \times s \times t + n_m$.

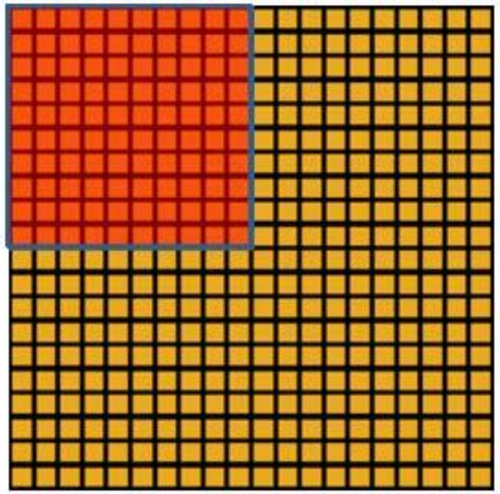


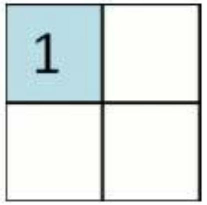
The number of paramaters of layer m-1 and layer m is 4*2*2*2+2=34.

Convolutional Neural Network: Max-pooling Layer

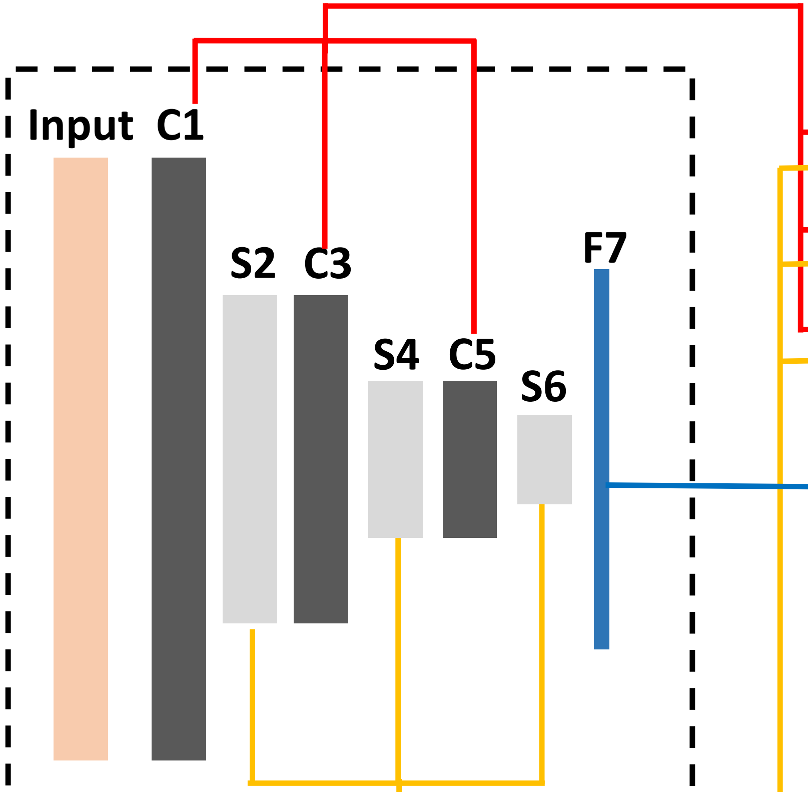
- For a input gray image with 96*96, if we use 400 8*8 kernels to obtain feature maps, we will get 400 feature maps with size (96-8+1)*(96-8+1) .
- It means we have 400*(96-8+1)*(96-8+1)=3168400 features!!

  - Max-pooling
  - Mean-pooling





Convolutional Neural Network: Experiment



The architecture of CNN

```
model = Sequential()
model.add(Conv2D(10,(3,3),activation='relu',input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(10,(3,3),activation='relu',input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(10,(3,3),activation='relu',input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(10))
model.add(Activation('softmax'))

model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])

history = model.fit(X_train, Y_train,
                    batch_size = batch_size,
                    nb_epoch = nb_epoch,
```

Convolutional Neural Network: Experiment

	20 neurons	vs	CNN
epoch1	loss: 0.5601 - acc: 0.8527 - val_loss: 0.3027 - val_acc: 0.9148		loss: 1.0958 - acc: 0.6510 - val_loss: 0.4778 - val_acc: 0.8579
epoch2	loss: 0.2874 - acc: 0.9183 - val_loss: 0.2617 - val_acc: 0.9256		loss: 0.3964 - acc: 0.8837 - val_loss: 0.2943 - val_acc: 0.9148
epoch3	loss: 0.2540 - acc: 0.9273 - val_loss: 0.2380 - val_acc: 0.9300		loss: 0.2926 - acc: 0.9128 - val_loss: 0.2399 - val_acc: 0.9277
epoch4	loss: 0.2300 - acc: 0.9343 - val_loss: 0.2193 - val_acc: 0.9368		loss: 0.2453 - acc: 0.9257 - val_loss: 0.2042 - val_acc: 0.9392
epoch5	loss: 0.2108 - acc: 0.9395 - val_loss: 0.2152 - val_acc: 0.9363		loss: 0.2138 - acc: 0.9355 - val_loss: 0.2097 - val_acc: 0.9372
epoch6	loss: 0.1963 - acc: 0.9439 - val_loss: 0.1940 - val_acc: 0.9414		loss: 0.1895 - acc: 0.9424 - val_loss: 0.1813 - val_acc: 0.9436
epoch7	loss: 0.1853 - acc: 0.9466 - val_loss: 0.1952 - val_acc: 0.9436		loss: 0.1724 - acc: 0.9477 - val_loss: 0.1529 - val_acc: 0.9545
epoch8	loss: 0.1764 - acc: 0.9488 - val_loss: 0.1835 - val_acc: 0.9460		loss: 0.1569 - acc: 0.9523 - val_loss: 0.1454 - val_acc: 0.9564
epoch9	loss: 0.1691 - acc: 0.9511 - val_loss: 0.1856 - val_acc: 0.9452		loss: 0.1440 - acc: 0.9558 - val_loss: 0.1523 - val_acc: 0.9540
epoch10	loss: 0.1629 - acc: 0.9525 - val_loss: 0.1747 - val_acc: 0.9494		loss: 0.1354 - acc: 0.9582 - val_loss: 0.1236 - val_acc: 0.9620

val_loss: 0.1747 - val_acc: 0.9494

val_loss: 0.1236 - val_acc: 0.9620

28*28*20+20 = 15700 parameters vs **2030 parameters**

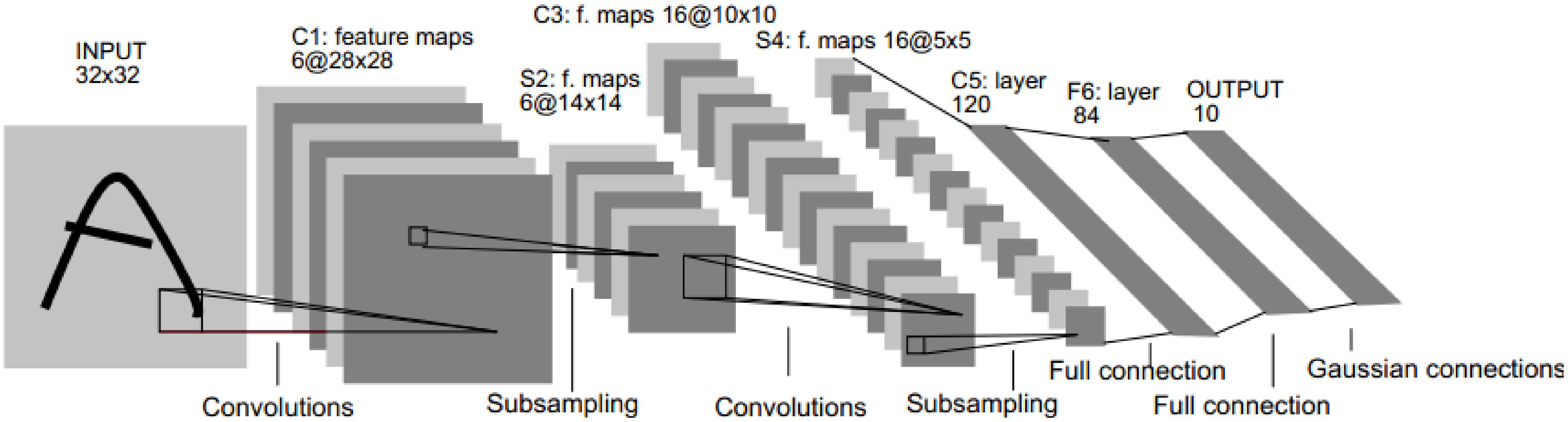
Classical CNN model

- **Lenet——1998**
- **Alexnet——2012**
- **VGG-net——2014**

Classical CNN model——Lenet

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

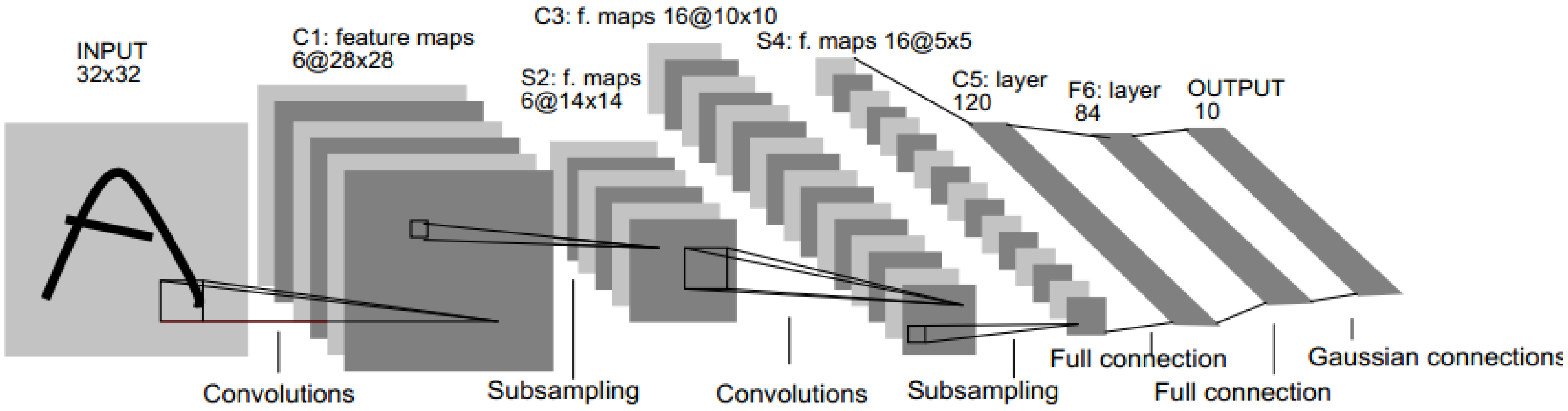
- 7 layers:
- Input layer: 1@32*32
- layer1: 6@5*5 kernel feature map1: 6@28*28
- layer2: 2*2 max-pooling feature map2: 6@14*14
- layer3: 16 feature map



By Libao Zhang, Jie Ma

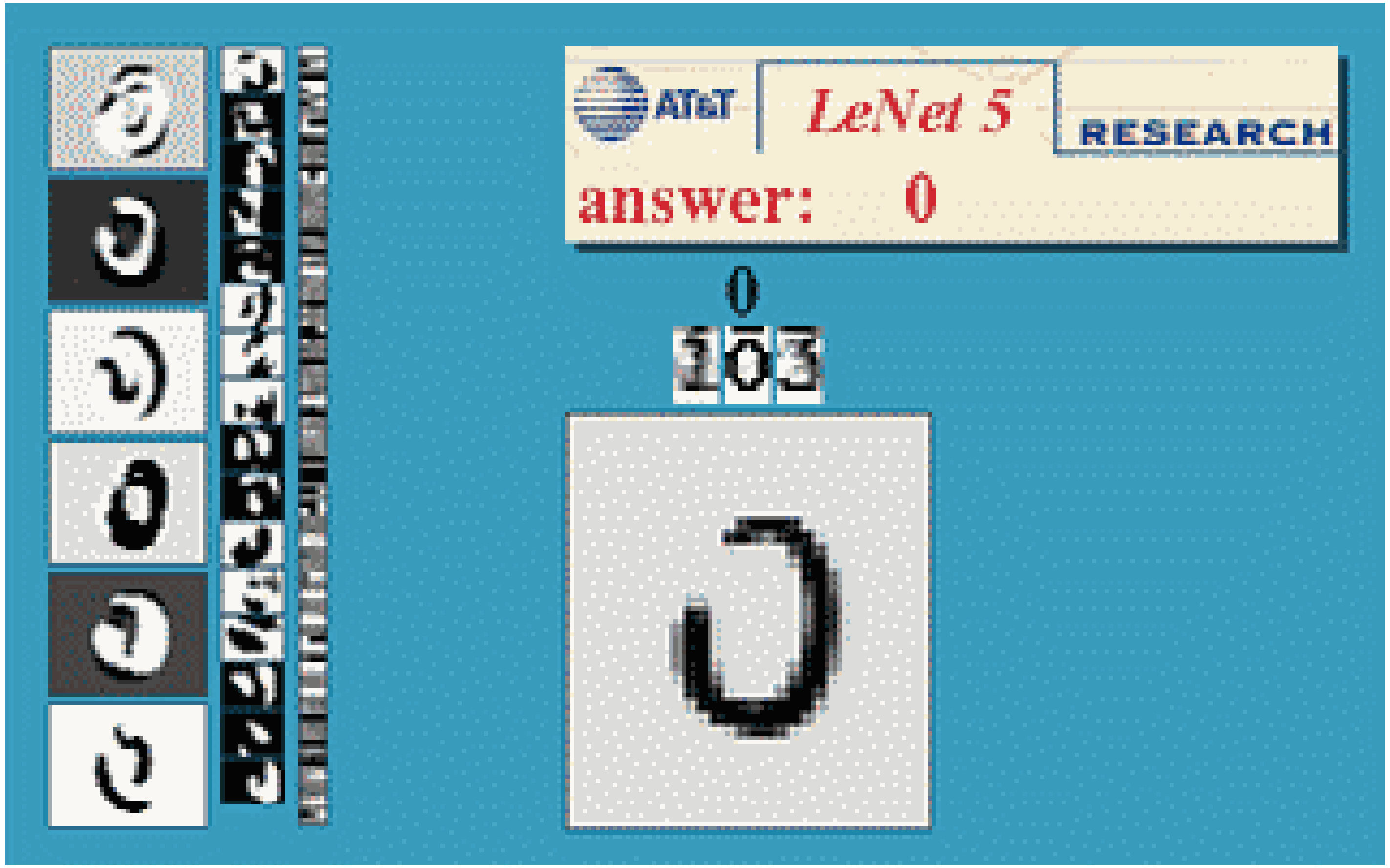
Classical CNN: Lenet

- layer4: max-pooling — feature map4: 16@5*5
- layer5: fully connected-120 — feature map: 120
- layer6: fully connected-84 — featuremap:84
- layer7: Gaussian conections — feature map:10



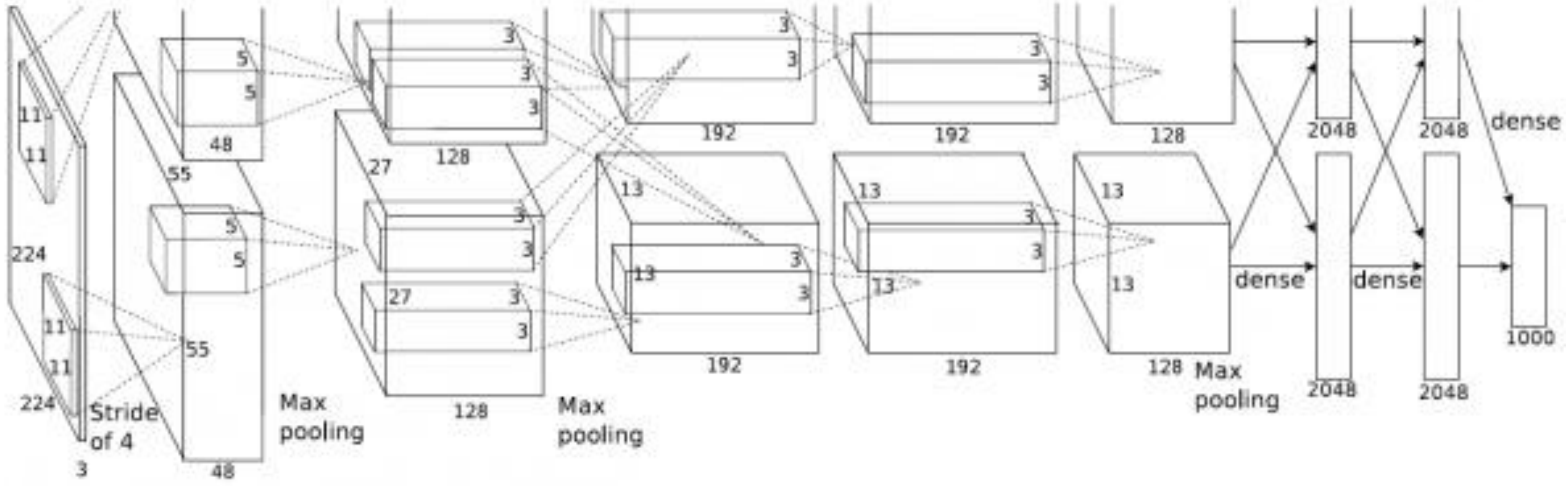
Classical CNN: Lenet

- **Demo from http://yann.lecun.com/exdb/lenet/multiples.html**



Classical CNN model——Alexnet

- The champion of LSVRC2010, top 5 error:15.3%, top 1 error:37.5%.
- 1.2 million high resolution image, 1000 classes.
- 60 million parameters , 650,000 neurons, 8 learning layers.



Classical CNN: Alexnet

11 layers:

Input layer: 3@224*224

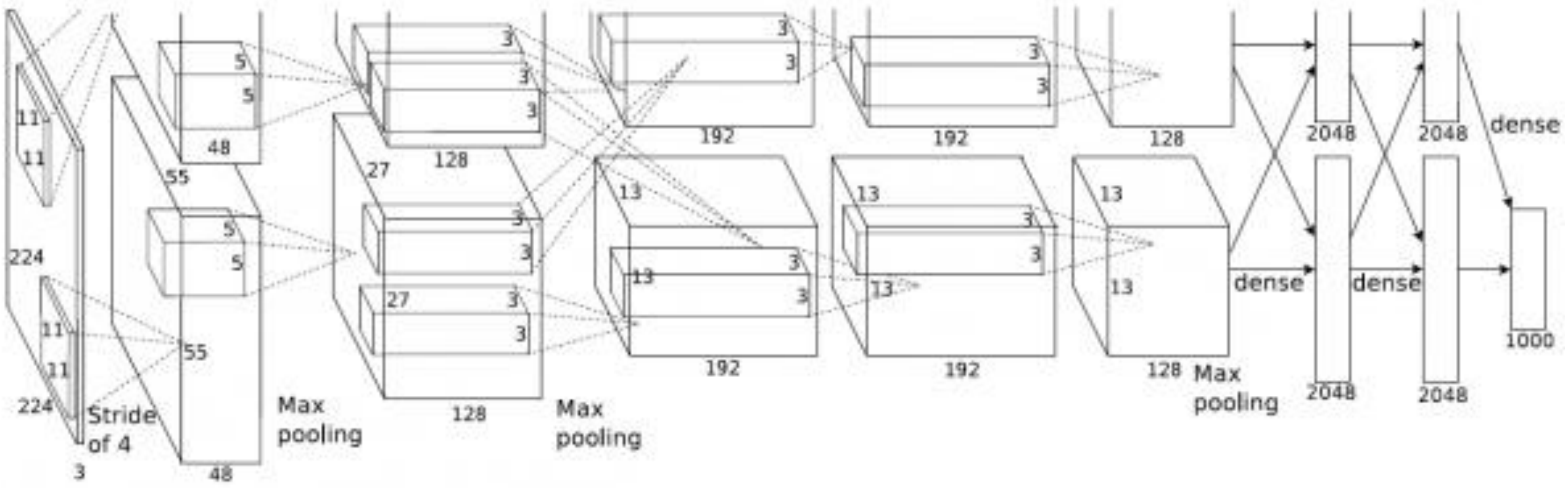
layer1:96@11*11 kernels,stride4 — **feature map1:96@55*55**

layer2:max-pooling 3*3, stride 2 — **feature map2:96@27*27**

layer3:256@ 5*5 kernels — **feature map3: 256@27*27**

layer4:max-pooling 3*3, stride 2 — **feature map4:256@13*13**

layer5:384@ 5*5 kernels — **feature map5: 384@13*13**



Classical CNN: Alexnet

layer6:384@ 5*5 kernels — feature map6: 384@13*13

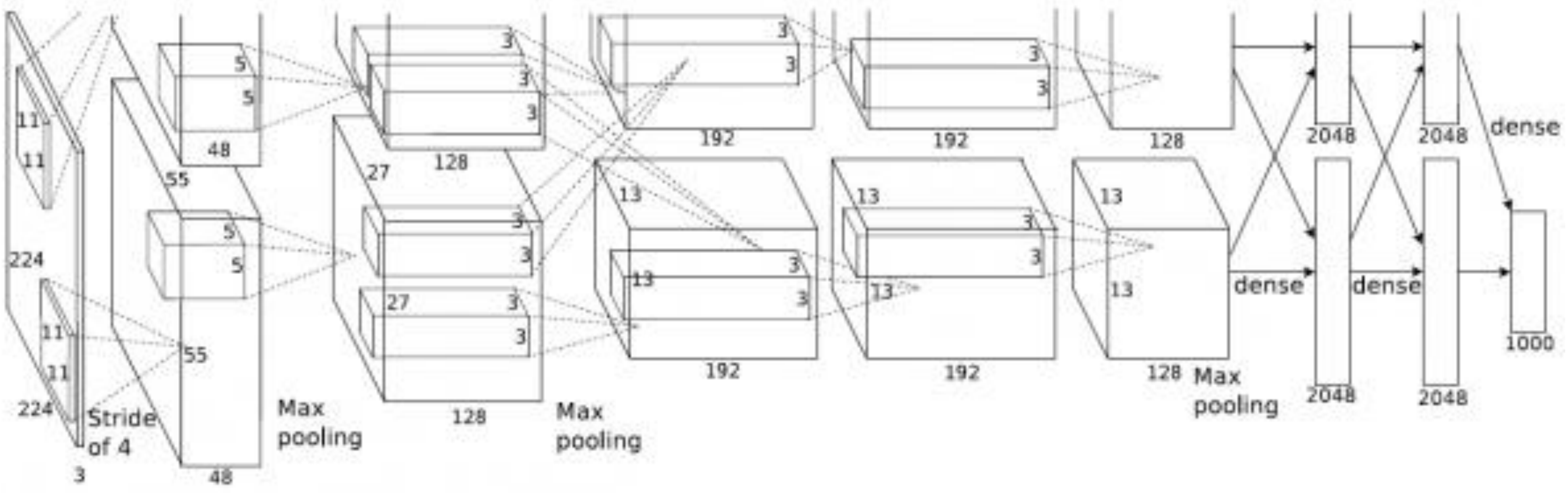
layer7:256@ 5*5 kernels — feature map7: 256@13*13

layer8:max-pooling 3*3, stride 2 — feature map8:256@6*6

layer9:fc7-4096 — feature map9: 4096

layer10:fc7-4096 — feature map10: 4096

fc11+softmax: output 1000



Classical CNN model——VGGnet

- **The champion of LSVRC2014, top 5 error:6.8%, top 1 error:23.7%.**

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

conv3-64

Why did they choose kernel with size 3*3 rather than 7*7 or 11*11?

Why did they choose kernel with size 3*3 rather than 7*7 or 11*11?

- **Reducing paramter.**
  - A stack of two 3×3 conv. layers (without spatial pooling in between) has an effective receptive field of 5×5.
  - Three such layers have a 7 × 7 effective receptive field.

- **Increasing the nonlinearity of the decision function.**
  - With 3 Relu nonlinear function.

Content

- ~~Neural Network~~
- ~~Convolution Neural Network~~
- Deep Learning for Computer Vision
  - AutoEncoder
  - Denoising AutoEncoder

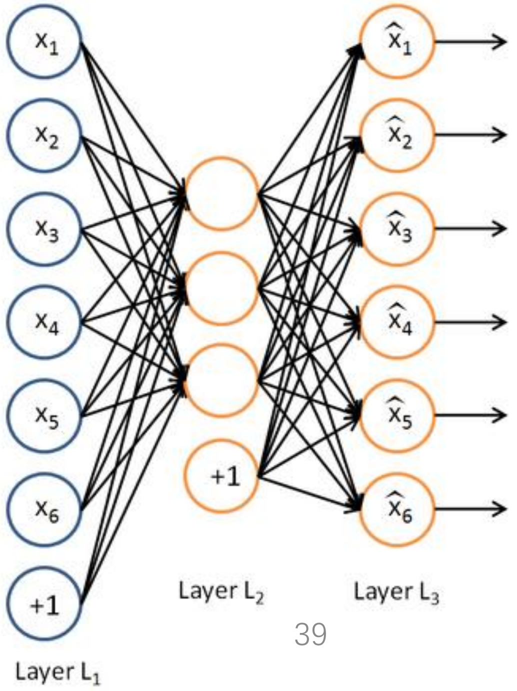
Deep Learning for Computer Vision： AutoEncoder

- So far, we have described the application of neural networks to supervised learning, in which we have labeled training examples.

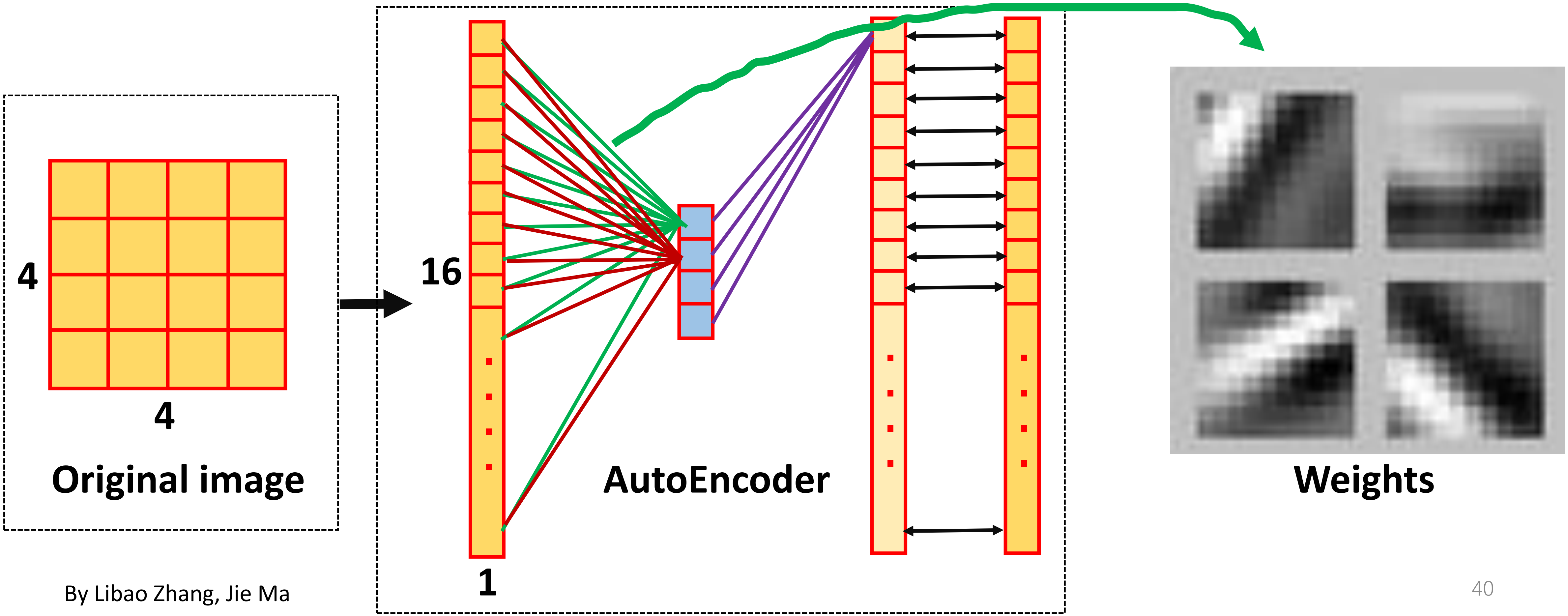
- Now suppose we have only a set of unlabeled training data $\{x^{(1)}, x^{(2)}, x^{(3)}, \cdots\}$. An autoencoder neural network is an unsupervised learning algorithm that applies backpropagation, setting the target values to be equal to the inputs.

$$\boldsymbol{y}^{(i)} = \boldsymbol{x}^{(i)}$$

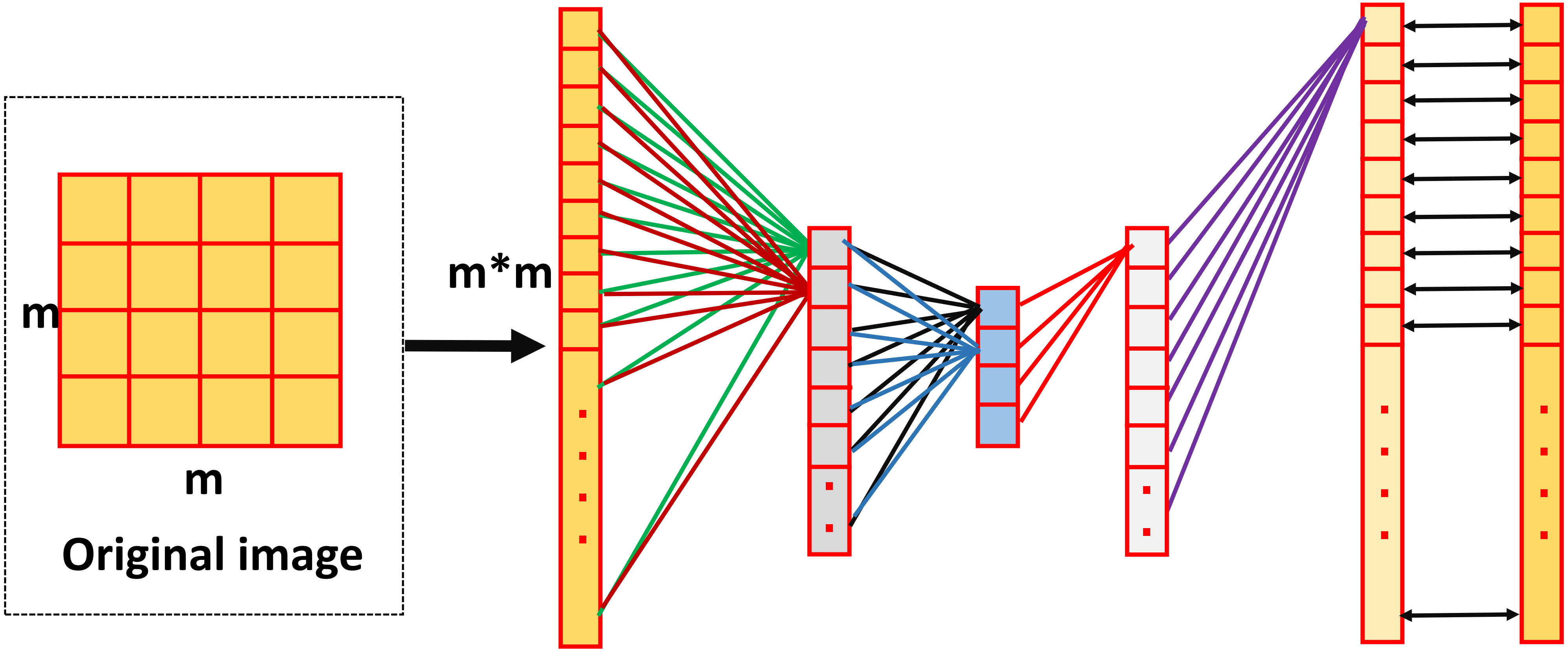
- The AutoEncoder tries to learn a function $f(x,\theta) = x$
- Suppose the inputs x are the pixel intensity values from a 10*10 image. so $n_1 = 100$, and there are $n_2 = 50$ hidden units in layer2.
- Note that we also have $y \in \Re^{100}$. Since there are only 50 hidden units, the network is forced to learn a compressed representation of the input.



- **Visualizing a trained Autoencoder:**
- **We will visualize the function computed by hidden unit i ——which depends on the paramaters weight $W^{(1)}_{ij}$**



- Stack AutoEncoder



Stack AutoEncoder

• Stack AutoEncoder: Experiment

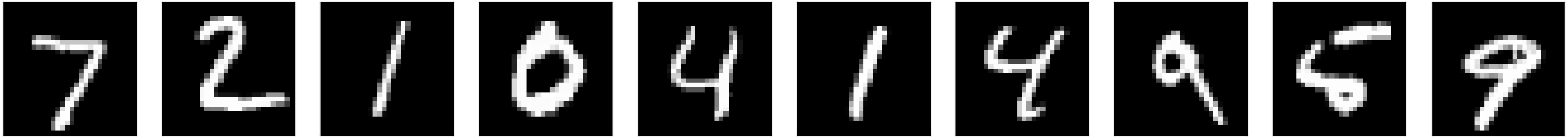
```
model = Sequential()
model.add(Conv2D(10,(3,3),activation='relu',border_mode='same',input_shape=input_shape))  #10@28*28
model.add(MaxPooling2D(pool_size=(2, 2)))                                                 #10@14*14

model.add(Conv2D(20,(3,3),activation='relu',border_mode='same',input_shape=input_shape))  #10@14*14
model.add(MaxPooling2D(pool_size=(2, 2)))                                                 #10@7*7

model.add(Conv2D(30,(3,3),activation='relu',border_mode='same',input_shape=input_shape))  #10@7*7

model.add(UpSampling2D(size=(2, 2)))                                                    #10@14*14
model.add(Conv2D(20,(3,3),activation='relu',border_mode='same',input_shape=input_shape))  #10@14*14

model.add(UpSampling2D(size=(2, 2)))                                                    #10@28*28
model.add(Conv2D(10,(3,3),activation='relu',border_mode='same',input_shape=input_shape))  #10*28*28

model.add(Conv2D(1,(3,3),activation='relu',border_mode='same',input_shape=input_shape))


model.compile(loss='mse',
              optimizer=RMSprop(),
              metrics=['accuracy'])

history = model.fit(X_train, X_train,
                    batch_size = batch_size,
                    nb_epoch = nb_epoch,
                    verbose = 1,
                    validation_data = (X_test, X_test))
```

Encoder

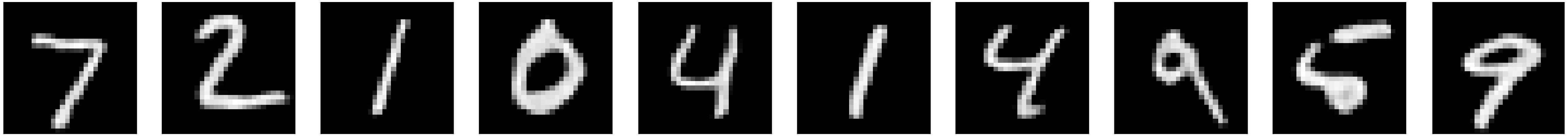
Decoder

- **Stack AutoEncoder: Experiment**

Original image



decode image



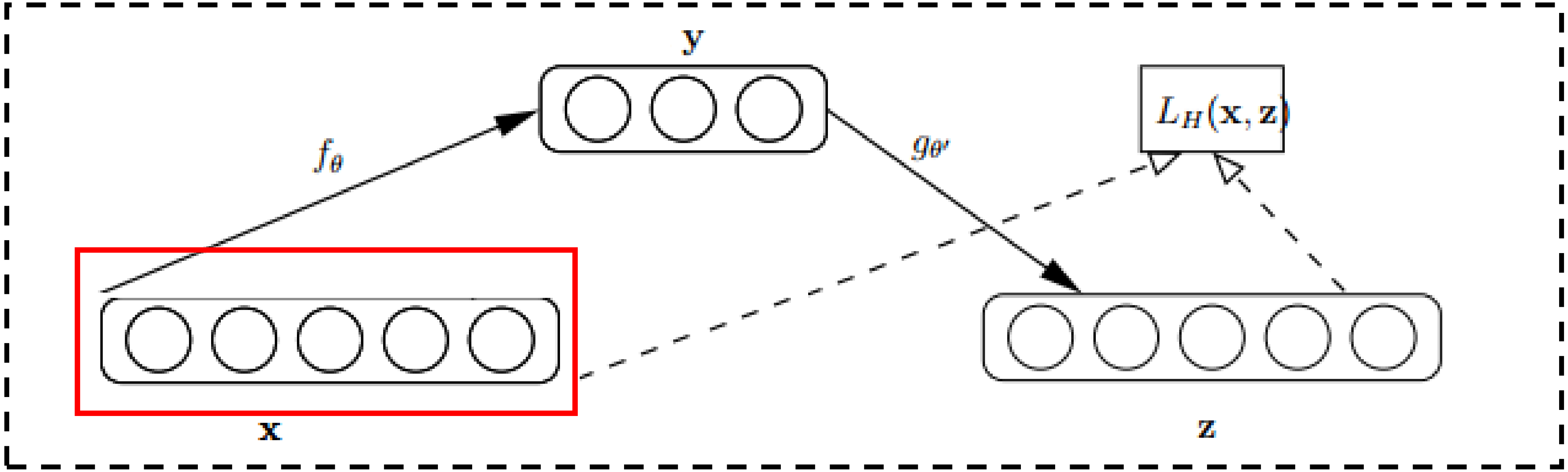
Deep Learning for Computer Vision:Denosing AutoEncoder

- **Question:**

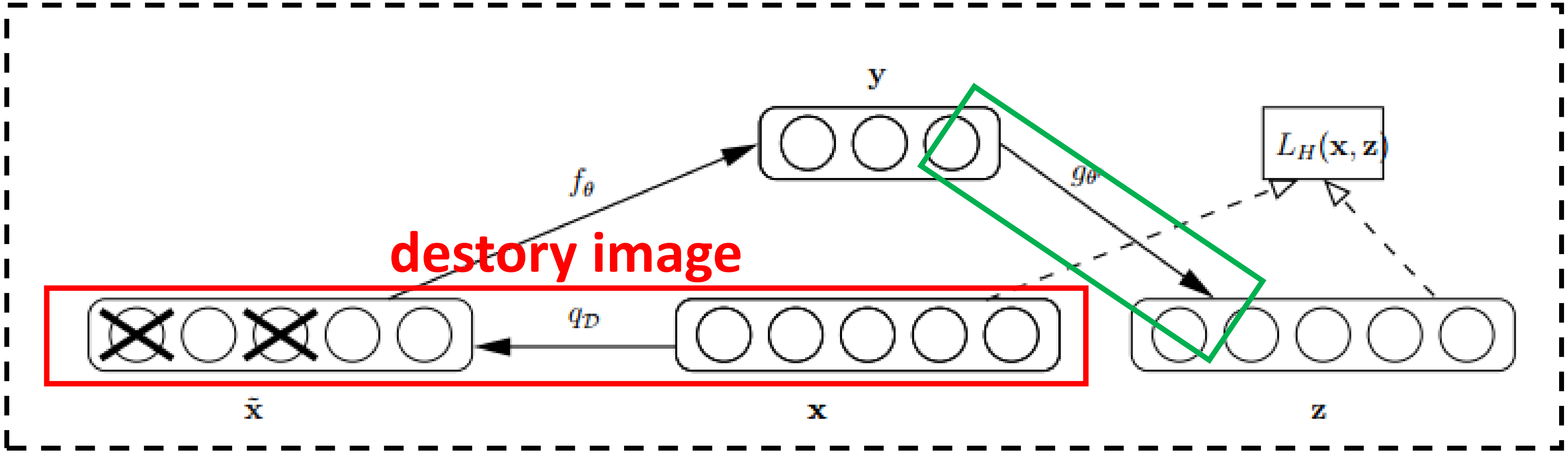
  How to learn more robust features?

- **In order to obtain more robust feature, we use damaged input to reconstruct the image.**

$$x \xrightarrow{\text{destory image}} \tilde{x} \xrightarrow{\text{reconstruct}} \hat{x}$$

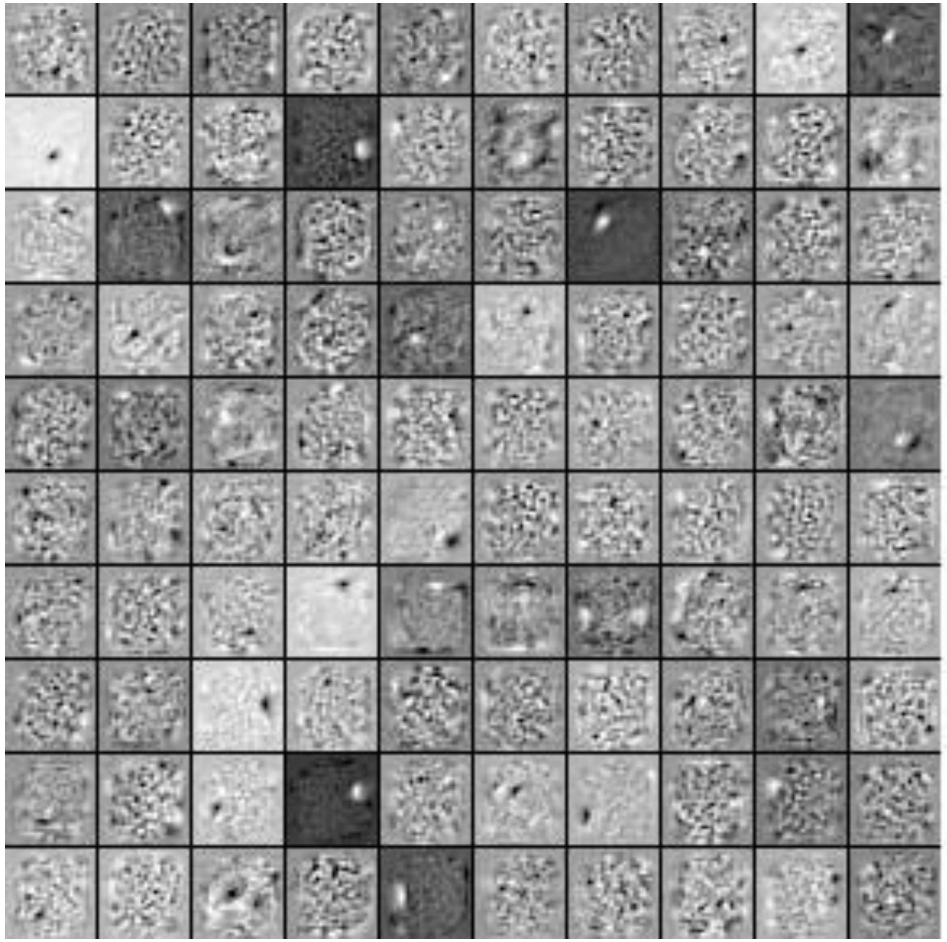


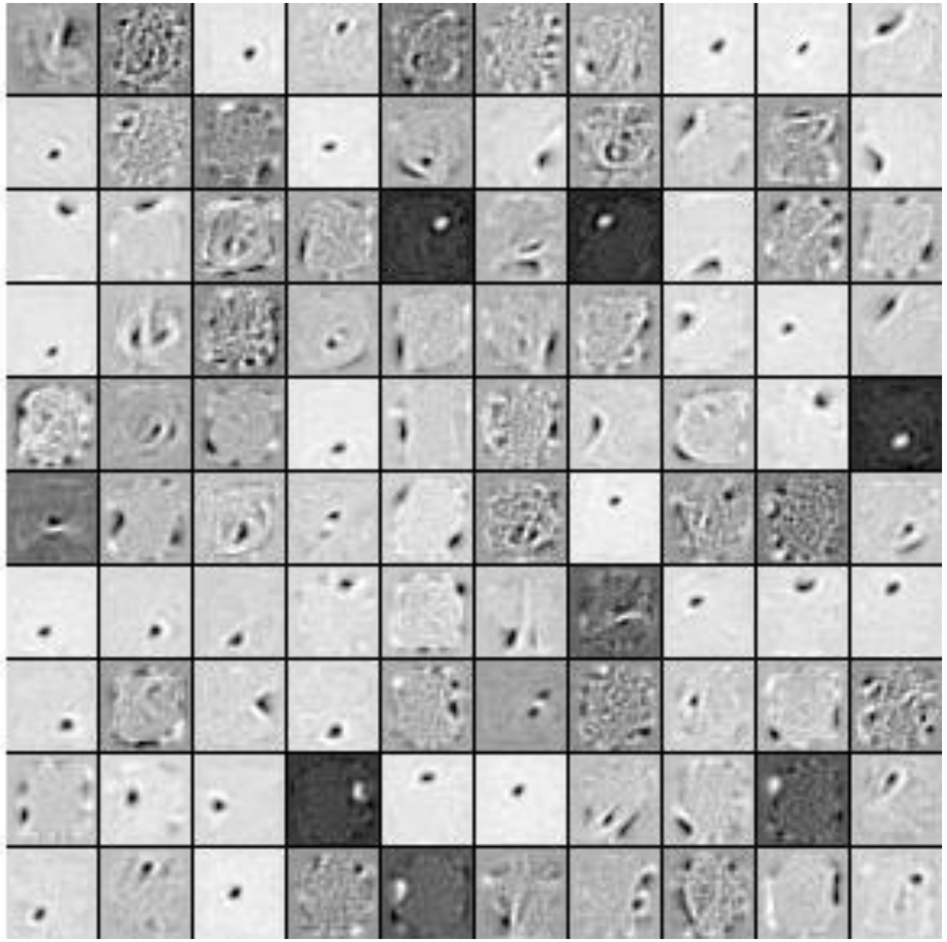
AutoEncoder



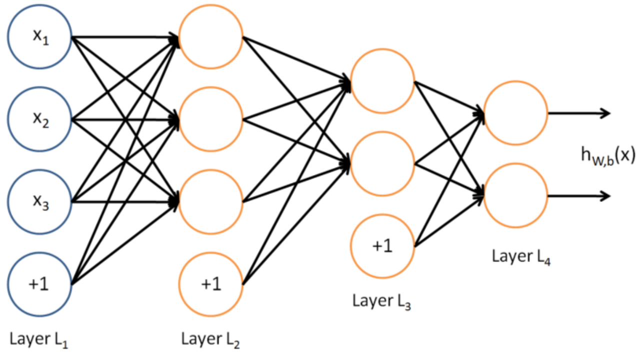
Denosing AutoEncoder

- **Comparision between AutoEncoder and Denosing AutoEncoder.**



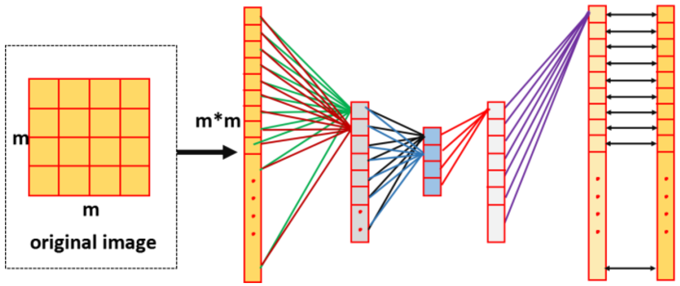


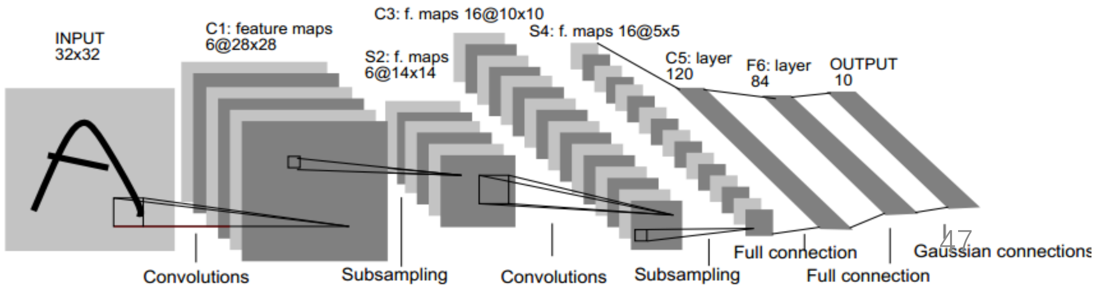
Summary



- **Neural Network**
  - **Discussion from XOR**
  - **Forward Propagation**
  - **Back Propagation**
- **Convolution Neural Network**
  - **Motivation**
  - **Convolutional layer**
  - **Max-pooling layer**
  - **Classical CNN**
    - **Lenet**
    - **Alexnet**
    - **VGG**
- **Deep Learning for Computer Vision**
  - **AutoEncoder** — **Stack AutoEncoder**
  - **Denoise AutoEncoder**

By Libao Zhang, Jie Ma





Public email address

deeplearning_class@163.com