Machine Learning in Practice
#4: Convolutional Neural Networks
"*First Contact with TensorFlow*", Ch. 5

Sang-Hyun Yoon

Summer 2019

Popular Deep Learning Systems

딥러닝은 계층이 깊은 neural network에 기반한 기법들의 총칭

- Convolutional neural network (CNN)
  - for supervised learning
- Deep Q-network (DQN)
  - for reinforcement learning
- Recurrent neural network (RNN)
  - cyclic connection in hidden layers allowed
  - for variable-length dataset (e.g. text, speech)
- Deep belief network (DBN)
- Restricted Boltzmann mahine (RBM)
- Autoencoder, ...

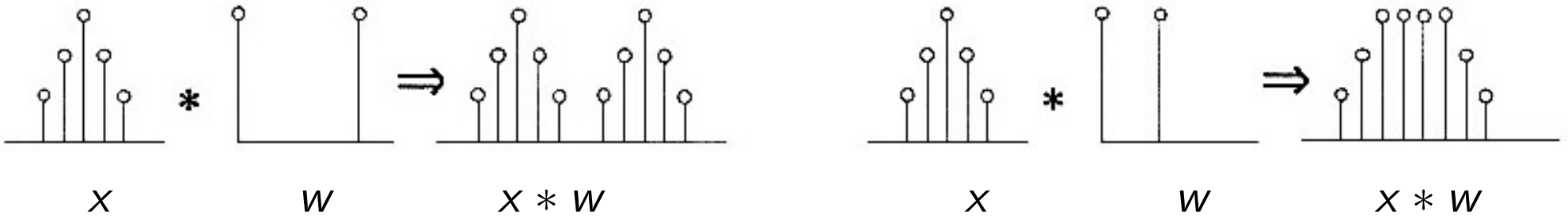
Neural network의 표현력은 뛰어나나 learnability 등의 문제로 사장되었다 딥러닝이 이를 극복하여 부활하게 됨

Convolution (합성곱) Operation in Calculus

Given $x : \mathbb{R} \to \mathbb{R}$ and $w : \mathbb{R} \to \mathbb{R}$, their convolution $x * w$ is

- $x * w : \mathbb{R} \to \mathbb{R}$

  $(x * w)(t) = \int_{-\infty}^{\infty} f(a)g(t-a)da$



Signal processing에서의 convolution의 역할

- 주어진 데이터 x에 filter w를 사용해 유용한 feature를 추출
  - 유용한 feature의 예: impulse response, 테두리, 색
  - $x * w$를 데이터 x의 feature map이라고 부름

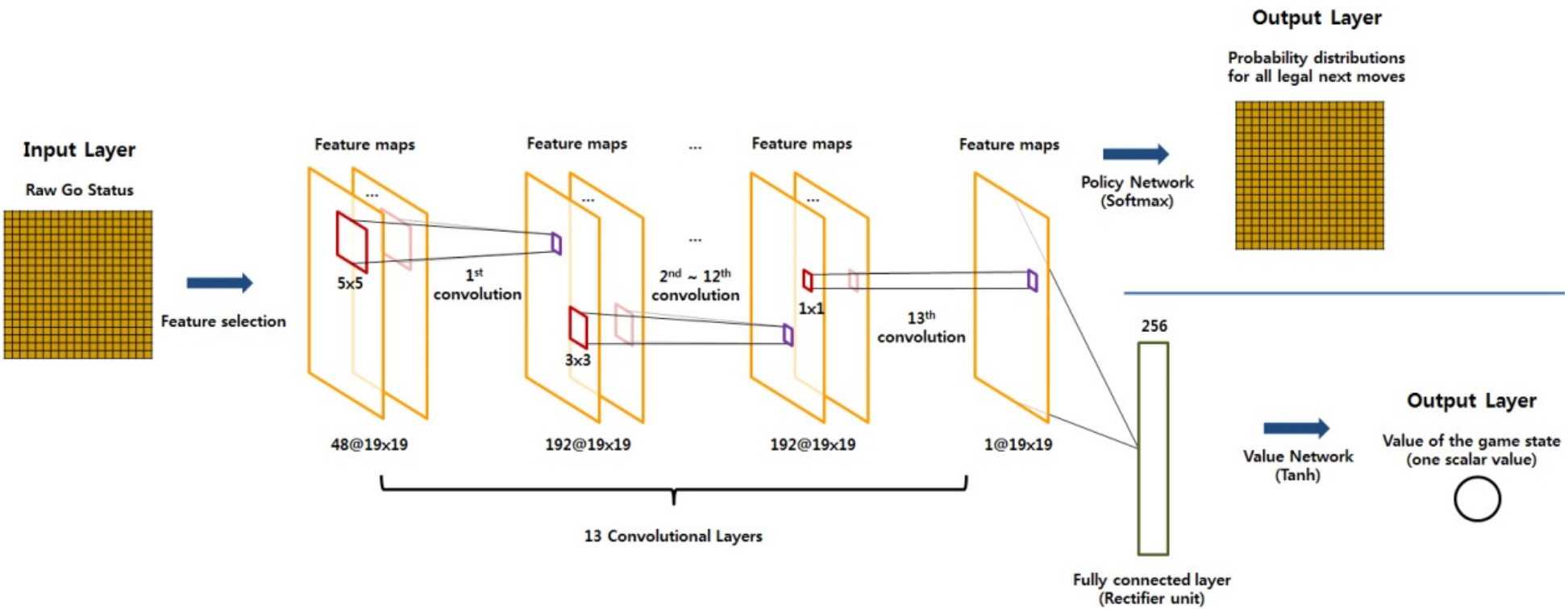
CNN에서의 convolution의 역할: proprocessing

- feature map을 잘 뽑으면 learning이 효율적으로 됨
- CNN의 핵심아이디어: convolution filter도 함께 learning

Outline

Convolutional Neural Networks

- hidden layer가 매우 많고
- 층간 연결방식이 convolution/pooling을 번갈아가면서 사용하다 마지막 층은 full connection
- Convolution/pooling 층간 연결은 sparse하게 (학습속도 ↑)
- 각 층마다 사용하는 activation 함수가 다름
  - ReLU, maxpool, softmax, hyperbolic, sigmoid 등



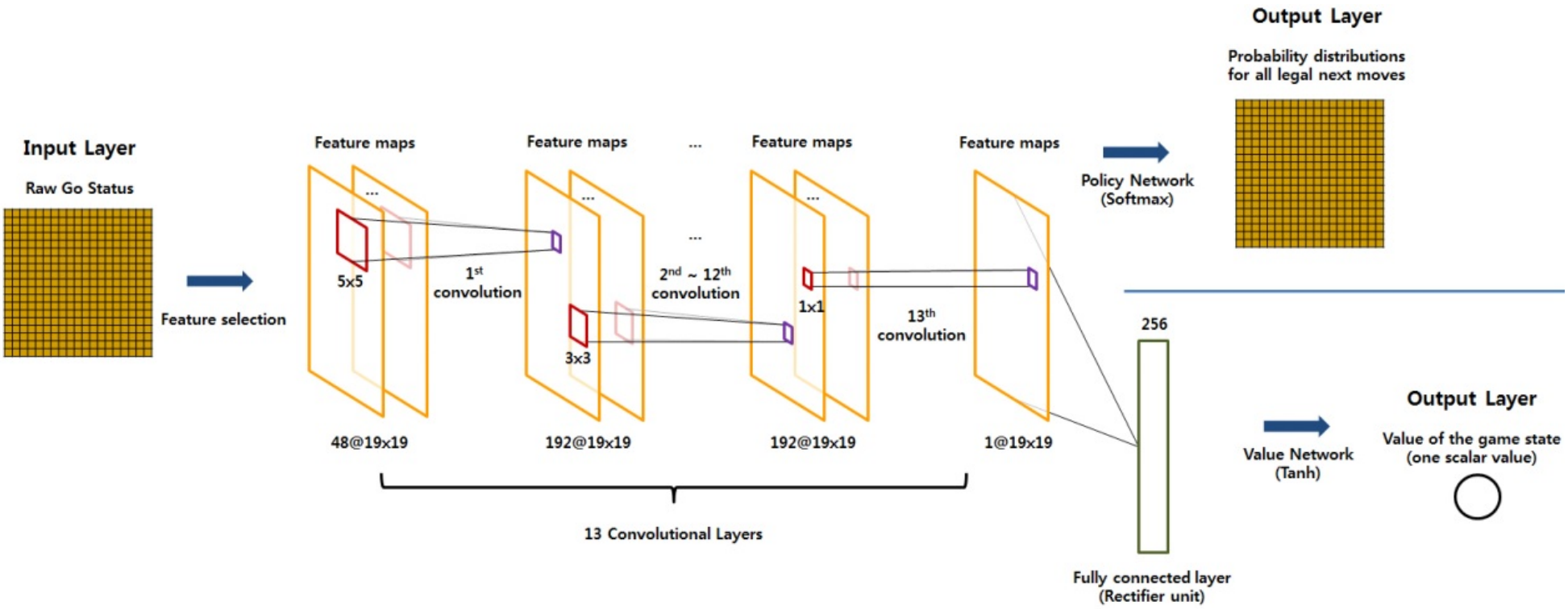
Convolution Layers (1/2)

- Convolution 층: 앞 층에서 인접한 노드끼리의 집합과 자기 층의 노드가 국소적으로 연결되어 있는 층 (뒤에서 자세히)
- Convolution 층의 역할은 feature 추출을 위한 filter
  - Filter의 특성은 연결 weight에 의해 결정
- (이미지로부터) feature를 추출/강조하여 다음 층으로 넘김

http://cs231n.stanford.edu

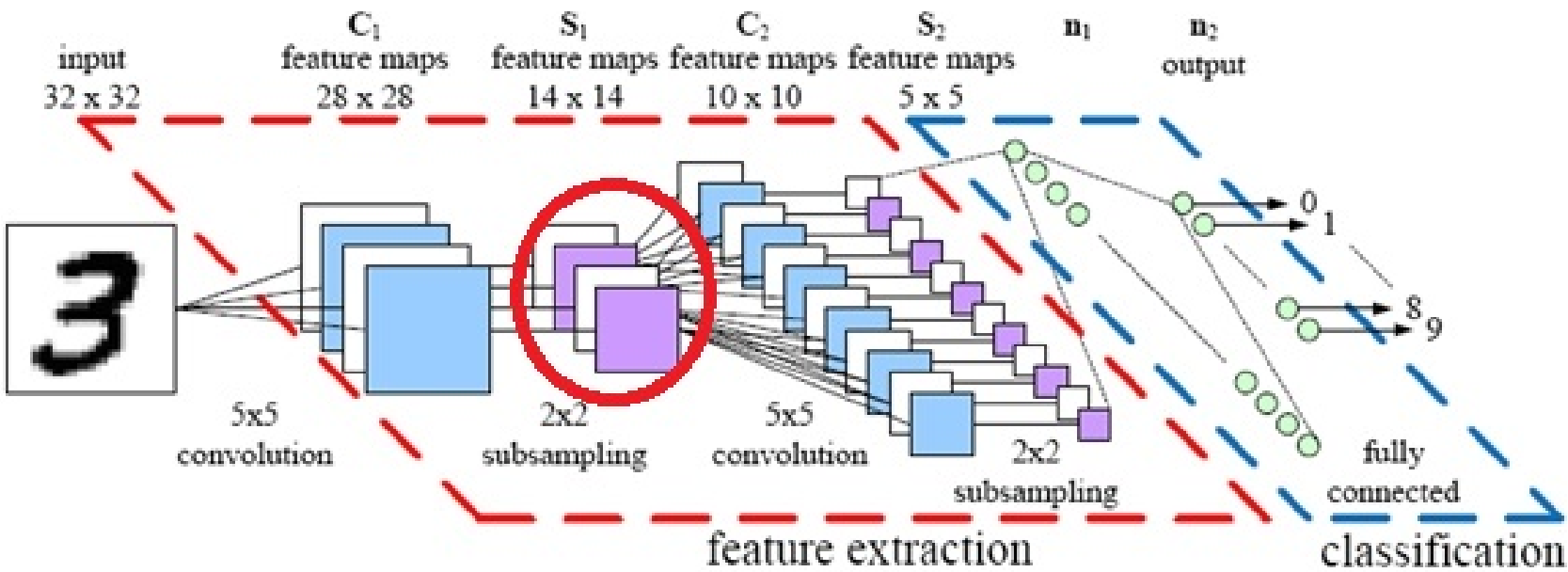
Convolution Layers (2/2)

- 각 convolution 층은 하나 이상의 feature map들로 구성됨
- AlphaGo의 1st convolution 층은 48개의 19×19 feature map으로 구성되어 48가지 특징을 (자동으로) 추출
  - 흰 돌, 검은 돌, 빈 칸, 출, 활로, 과거기록 등
  - 2~12번째 convolution 층은 192개씩



Pooling Layers

- 각 convolution 층을 구성하는 여러개의 feature map 각각에 대해 pooling 층이 뒤따름
- Convolution 층의 출력 정보를 단순하게 압축 (subsamping)
- 2×2 max-pooling의 경우 2×2 영역에서 가장 큰 값을 선택
- Pooling 층의 또다른 역할: 이미지의 작은 변화를 흡수
  - 특정 영역에서 최대값을 계산하므로 범위내 작은 위치 변화에도 같은 결과를 출력



Summary

- Convolution 층은 feature(특징) 추출 역할
- Pooling 층은 변위를 흡수해 변화를 최소화하는 역할
- Convolution-pooling을 여러번 거치면서 구체적인 이미지 정보가 점점 추상화(e.g. 말/비행기/배) 됨
- 마지막 층만 fully connected되고 나머지 층간의 연결은 국부적으로만 sparse하게 이루어져 weight 갯수를 대폭 줄여서 learning 속도가 대폭 향상됨
- 그러면서도 층을 깊게 하고 convolution/pooling 층의 역할을 적절히 도입하여 정확도도 크게 개선됨
- Hyperparameter(층 갯수, 각 층의 feature map 갯수/크기 등)은 실험을 통해서 최적의 값을 찾아야 함
- Learning을 위한 최적화 계산식은 매우 복잡할 수 밖에 없으나 TensorFlow 같은 tool을 사용하면 누구나 쉽게

Outline

Recall: MNIST Dataset (for supervised learning)

- 손글씨 숫자 흑백 이미지. 각 이미지는 28×28 픽셀
- 훈련용 60,000개 및 테스트용 10,000개

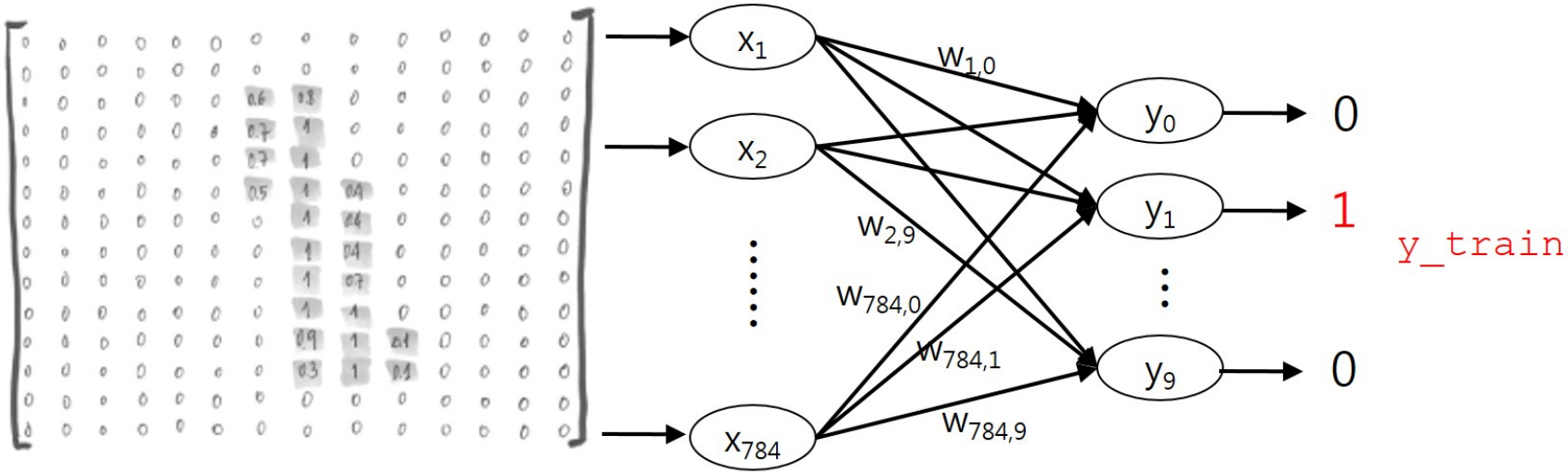
```
mnist = tensorflow.keras.datasets.mnist.load_data()
(x_train, y_train), (x_test, y_test) = mnist
```

- x/y_train은 훈련데이터, x/y_test는 테스트데이터
- x_train/test는 입력 이미지 (입력)
- y_train/test는 출력 숫자 (supervised learning에 필요)

Recall: Single-Layer Neural Network for Digit Classification

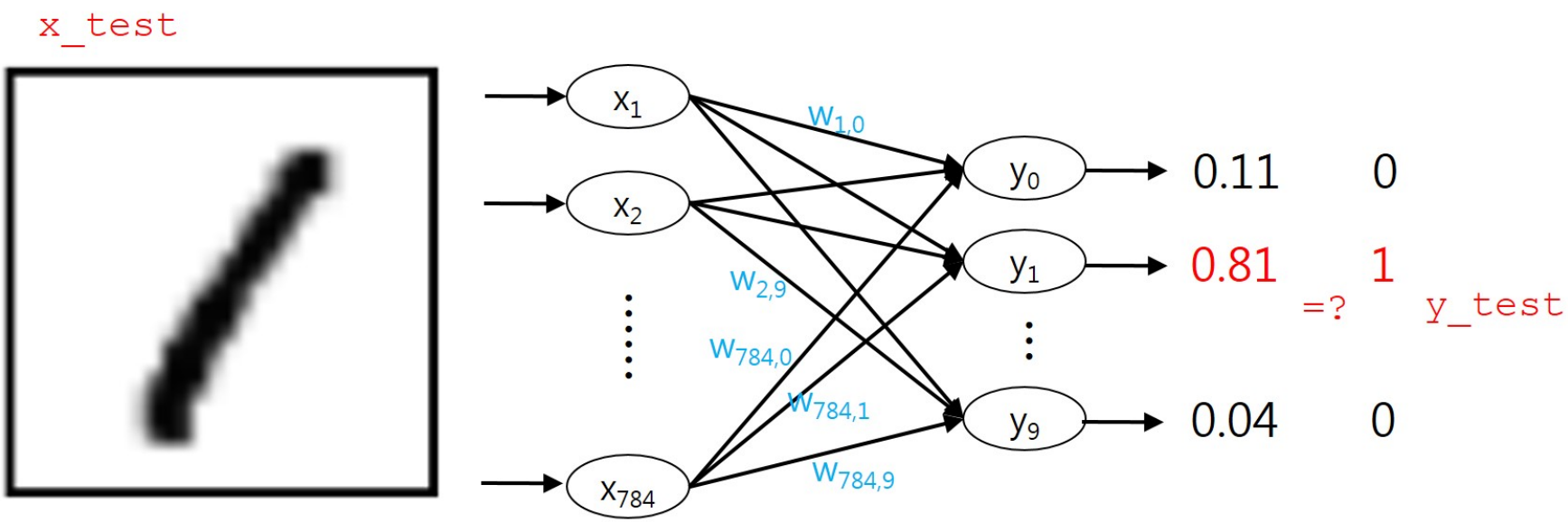
학습 단계 (off-line): `Train_NN.py`

x_train



Recall: Single-Layer Neural Network for Digit Classification

학습된 NN를 이용 (on-line): `Test_NN.py`

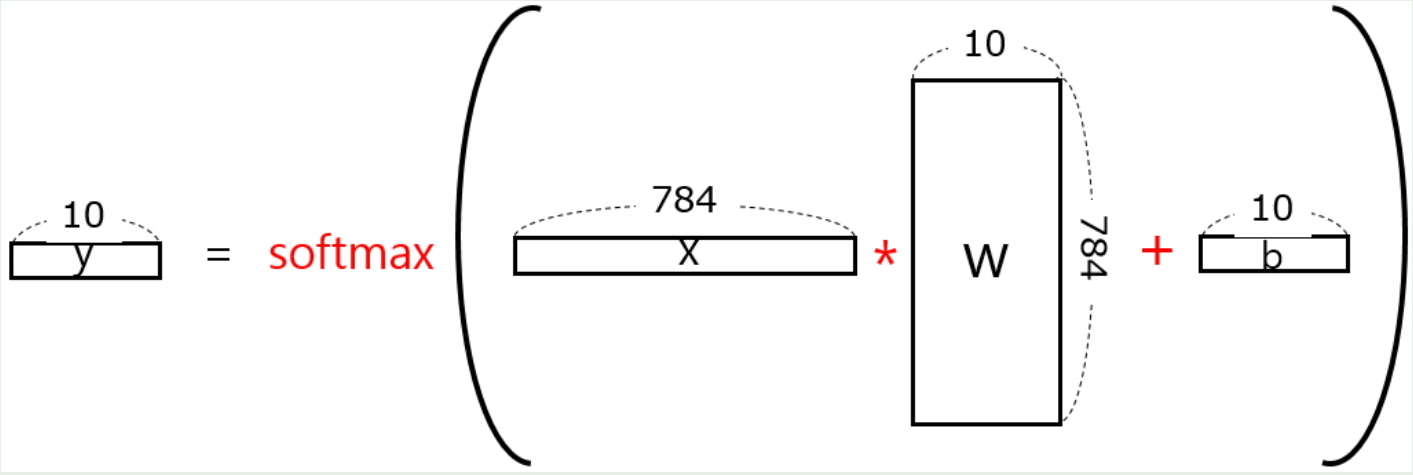


Recall: TensorFlow Code

- Train_NN.py: NN(x_train) ≈ y_train인 NN을 구성
  - 덤으로 유사도(NN(x_train), y_train)도 측정
  - 그리고 계산된 NN을 파일에 저정
- Test_NN.py: Train_NN.py에서 구성/저장한 NN을 읽은 후 유사도(NN(x_test), y_test)를 측정
  - 유사도(NN(x_train), y_train)보다 크게 낮으면 overfitting
- NN_comp_graph.py: Train_NN.py과 Test_NN.py에서 공통으로 사용하는 computation graph를 구성
  - graph의 입력 단자: x_train/test, y_train/test (placeholder 형태)
  - graph의 출력 단자: train, accuracy
    - train: NN(x_train) ≈ y_train인 NN을 구성하라는(즉, cost 함수 최적화) 나타내는 노드로 Train_NN.py에서만 사용
    - accuracy: 유사도(NN(x), y)

Data Structure for (Training) Image Data (1/2)

Recall: Single Layer Neural Network




```
x_train = tf.placeholder(tf.float32, [None,784])
y_train = tf.placeholder(tf.float32, [None,10])
```

- CNN에서는 이미지를 28×28 형태로 표현하여 2차원 공간적 특징들을 잡아낼 수 있도록 하는 것이 중요
- Single-layer NN은 fully connected라 이미지를 1차원 형태로 표현해도 문제 없으나, CNN은 국소적으로만 연결되어 2차원 형태로 이미지를 표현해야 제대로 학습 가능

Data Structure for (Training) Image Data (2/2) (p.118)

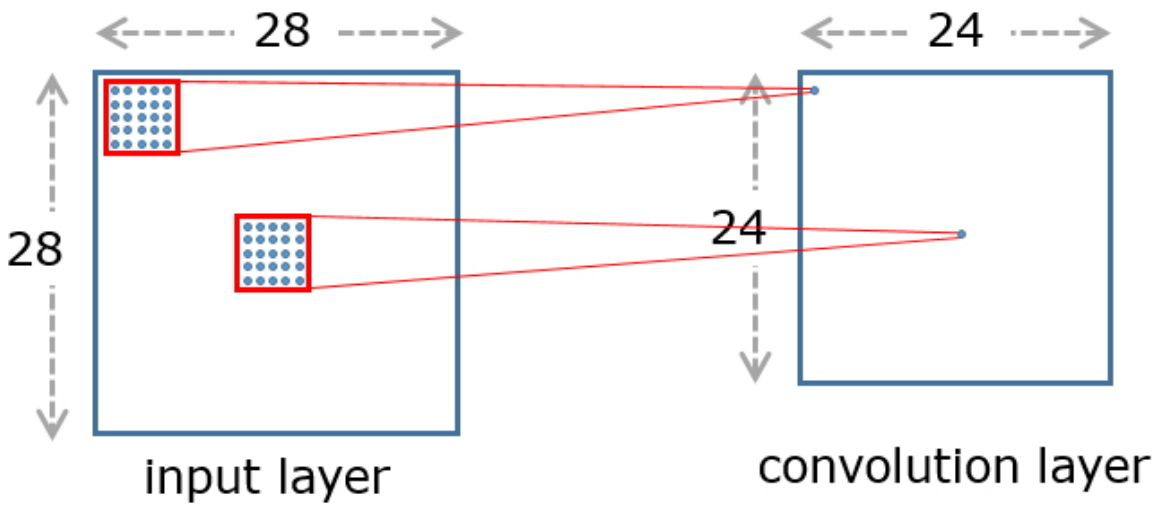
```
x_train = tf.placeholder(tf.float32, [None,784])
y_train = tf.placeholder(tf.float32, [None,10])
```

TensorFlow API document: reshape function

```
x_image = tf.reshape(x_train, [-1,28,28,1])
```

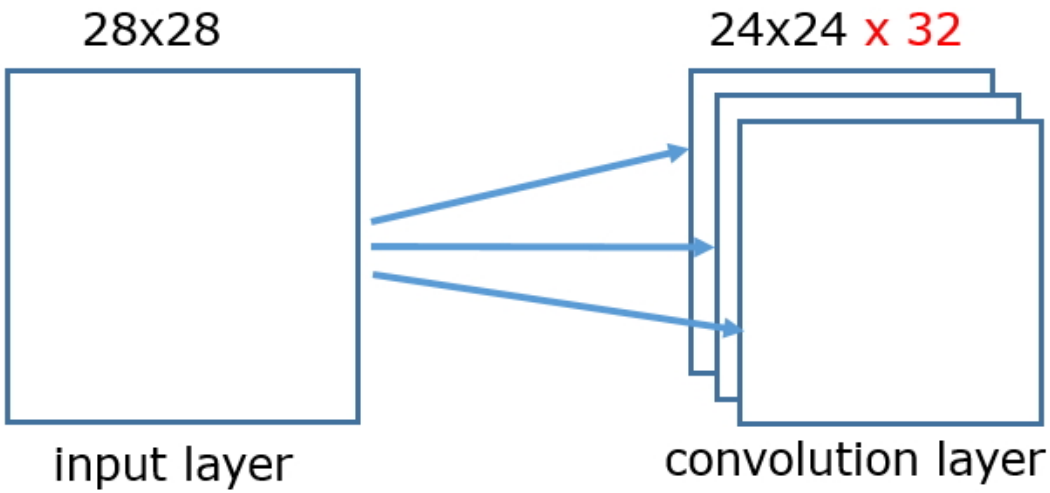
- -1은 이후 x_train에 들어갈 이미지의 갯수로 자동 계산됨
  - -1은 placeholder에서의 None과 같은 역할
  - 위의 reshape 함수 설명 링크 참조
- 28,28은 이미지 하나
- 마지막 차원의 1은 컬러 채널의 갯수
  - 여기서는 흑백으로 다루므로 컬러 채널은 한개
  - 컬러로 다루려면 R/G/B 값당 하나씩 필요하므로 1대신 3으로 하면 자연스럽게 처리할 수 있음

Convolution Layers (1/2)



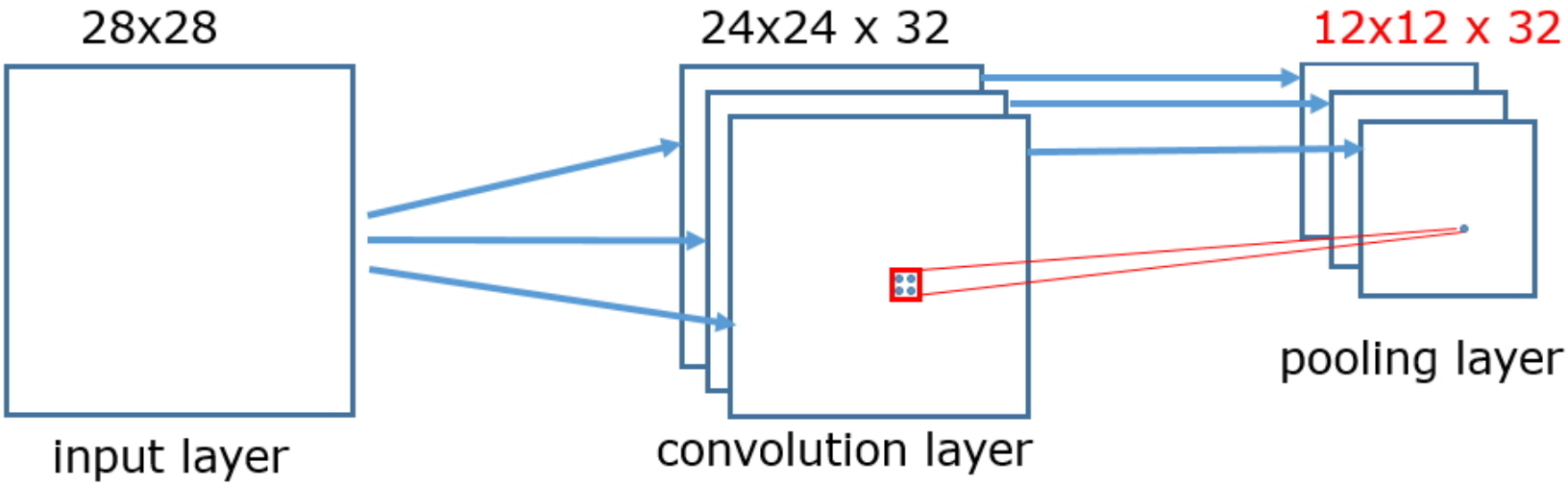
- 입력층의 각 5×5 영역을 convolution 층의 한 노드와 연결
  - 즉, convolution 층의 각 노드는 입력 층의 25개 노드와 연결
  - Weight가 25개만 있으면 됨! (784개가 아니고)
- 심지어 25개의 weight는 모든 노드들이 공유
  - Full connection하면 weight가 $784 \cdot 576$개 필요 (18063배!)
- 28×28 크기의 입력층을 5×5 크기의 sliding window가 훑고 지나가는 걸로 보면 됨. 이 window를 filter/kernel라 부름
- 한번에 1픽셀 이상 움직일 수도 있는데, 이 값을 stride
- 입력 이미지 바깥으로도 넘어갈 수 있음 (padding)

Convolution Layers (2/2)



- Filter(kernel) 하나는 하나의 feature(특성)을 감지하는 역할
  - 'Filter로 feature map을 만든다'라고 부름
  - https://docs.gimp.org/en/plug-in-convmatrix.html
- 감지하고 싶은 각 feature에 하나씩 여러개의 filter를 사용하면 좋음 (예: AlphaGo는 48개)
- 각 filter마다 feature map이 하나씩 생성되어 각 convolution 층은 여러개의 feature map으로 구성됨
- 위 예는 32개의 feature map으로 구성되는데 필요한 weight는 $5 \cdot 5 \cdot 32 = 800$개

Pooling Layers



- Convolution 층의 출력 정보를 단순하게 압축 (subsamping)
- 2×2 max-pooling의 경우 2×2 영역에서 가장 큰 값을 선택
  - convolution 층처럼 sliding window로 생성되는게 아니고 tile로 나뉘어 각각 만들어짐
- Pooling 층의 역할: 이미지의 작은 변화를 흡수
  - 특정 영역에서 최대값을 계산하므로 범위내 작은 위치 변화에도 같은 결과를 출력 (상대적 위치 vs. 절대적 위치)
- convolution-pooling 층을 묶어서 feature map으로 부르기도

TensorFlow Code: Convolution-Pooling Layers (1/2) (pp.125-127)

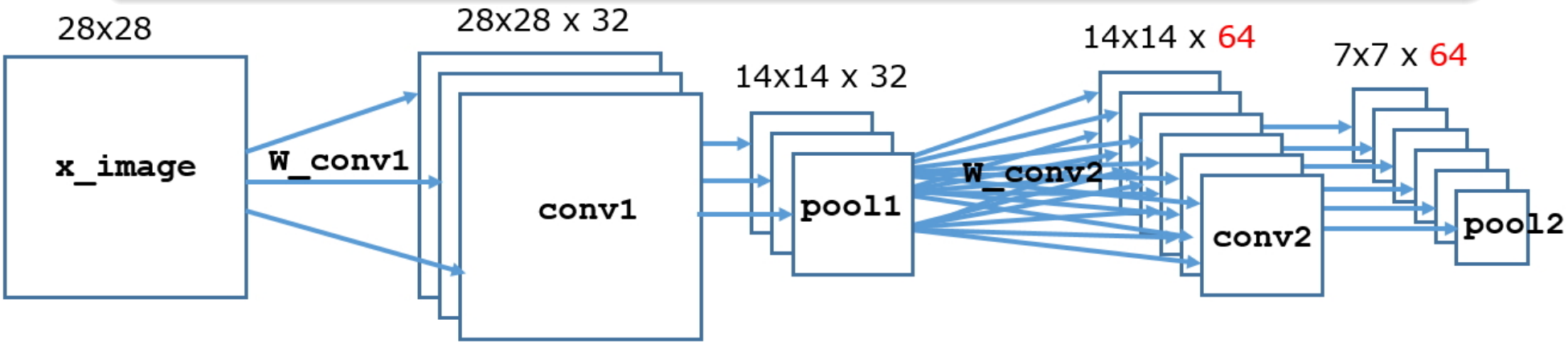
```
x_train = tf.placeholder(tf.float32, [None,784])
x_image = tf.reshape(x_train, [-1,28,28,1])
```

```
W_conv1 = tf.Variable(tf.fill([5,5,1,32], 0.0))  # weight
b_conv1 = tf.Variable(tf.fill([32], 0.1))        # bias

c1 = tf.nn.conv2d(x_image, W_conv1,   # shape:[-1,28,28,32]
                  strides=[1,1,1,1], padding='SAME')
conv1 = tf.nn.relu(c1 + b_conv1)
pool1 = tf.nn.max_pool(conv1, ksize=[1,2,2,1],  # 2x2
                  strides=[1,2,2,1], padding='SAME')
```

- API document: conv2d, max_pool, relu
- [5,5,1,32]: 5×5 filter 32개를 사용 (1은 컬러 채널 갯수)
- padding을 "VALID"로 하면 앞 슬라이드 그림처럼 24×24. "SAME"로 하면 28×28로 되고 채워지는 값은 링크 참조
- relu는 단순히 $\max(0, x)$를 리턴 (sigmoid보다 훨씬 간단)

TensorFlow Code: Convolution-Pooling Layers (2/2) (p.127)

```
pool1 = tf.nn.max_pool(conv1, ksize=[1,2,2,1], ... )
```

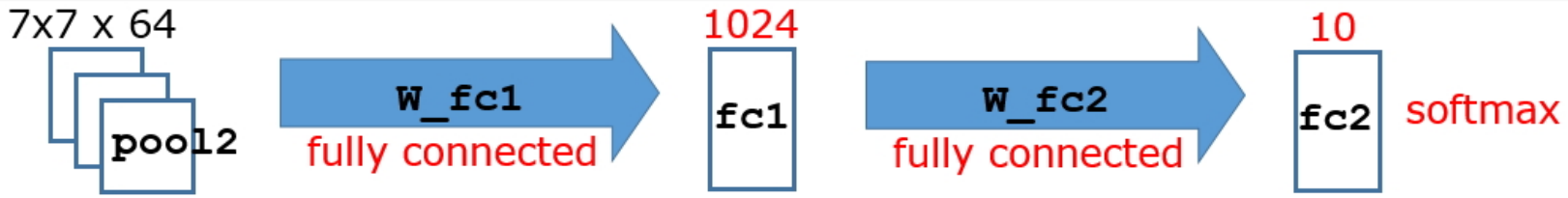



```
W_conv2 = tf.Variable(tf.fill([5,5,32,64], 0.0))
b_conv2 = tf.Variable(tf.fill([64], 0.1))

c2 = tf.nn.conv2d(pool1, W_conv2, ...) # shape:[-1,14,14,64]
conv2 = tf.nn.relu(c2 + b_conv2)
pool2 = tf.nn.max_pool(conv2, ksize=[1,2,2,1], ... )
```

- Convolution-pooling 층을 여러 계층 쌓아 올릴 수 있음
- W_conv2의 weight는 $5 \cdot 5 \cdot 32 \cdot 64$개 (5×5 filter $32 \cdot 64$개)
- conv2d대신 depthwise_conv2d를 쓰려면 filter 32개만 사용

TensorFlow Code: Output Layers (1/2) (pp.128-129)

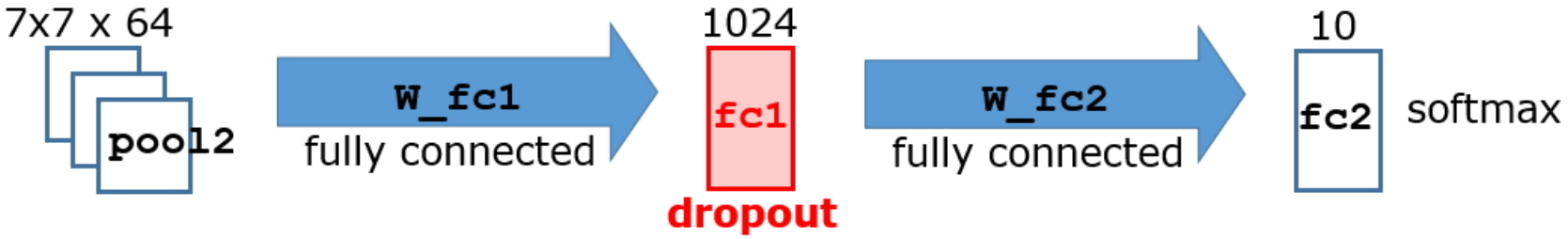
```
pool2 = tf.nn.max_pool(conv2, ksize=[1,2,2,1], ... )
```




```
pool2_flat = tf.reshape(pool2, [-1, 7*7*64])  # for full conn.

W_fc1 = tf.Variable(tf.fill([7*7*64, 1024], 0.0))
b_fc1 = tf.Variable(tf.fill([1024], 0.1))

fc1 = tf.nn.relu(tf.matmul(pool2_flat, W_fc1) + b_fc1)

W_fc2 = tf.Variable(tf.fill([1024, 10], 0.0))
b_fc2 = tf.Variable(tf.fill([10], 0.1))

y = tf.nn.softmax(tf.matmul(fc1, W_fc2) + b_fc2)
```

- 출력층은 full connection 층 2개로 구성되고 softmax값

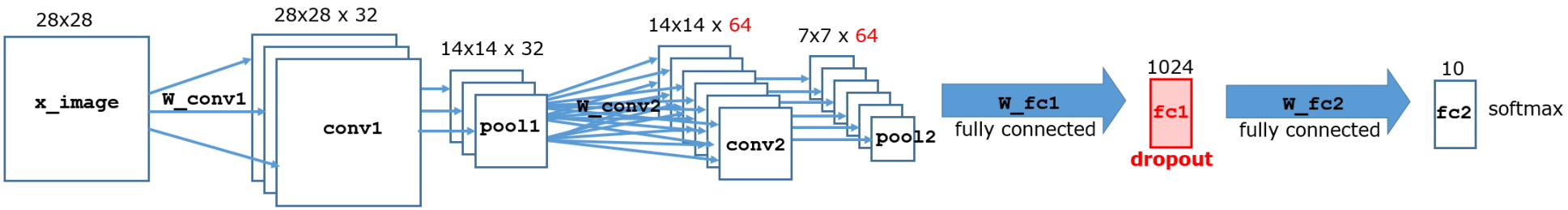
TensorFlow Code: Output Layers (2/2) (pp.128-129)




```
fc1 = tf.nn.relu(tf.matmul(pool2_flat, W_fc1) + b_fc1)

keep_prob = tf.placeholder(tf.float32)
fc1_drop = tf.nn.dropout(fc1, keep_prob)
...
y = tf.nn.softmax(tf.matmul(fc1_drop, W_fc2) + b_fc2)
```

- dropout을 사용하여 fc1의 노드의 일부를 랜덤하게 삭제
  - 각 노드가 삭제되지 않을 확률을 keep_prob로 넘김
- Representability를 위해 많은 노드가 필요는 한데, 매우 상세한 모델이 만들어 지면서 overfitting이 일어날 수 있음
  - 데이터의 차원에 비해 더 많은 'knob'(weight)를 가지면 발생
- Dropout을 사용하면 overfitting을 막는데 도움이 됨

TensorFlow Code: Cost Function (p.129)




```
x_train = tf.placeholder(tf.float32, [None,784])
y_train = tf.placeholder(tf.float32, [None,10])

W_conv1 = tf.Variable(tf.fill([5,5,1,32], 0.0))  # weight
W_conv2 = tf.Variable(tf.fill([5,5,32,64], 0.0))
W_fc1 = tf.Variable(tf.fill([7*7*64, 1024], 0.0))
W_fc2 = tf.Variable(tf.fill([1024, 10], 0.0))
b_conv1, b_conv2, b_fc1, b_fc2, ...

y = tf.nn.softmax(tf.matmul(fc1_drop, W_fc2) + b_fc2)

cross_entropy = -tf.reduce_sum(y_train*tf.log(y))  # scalar
```

Cross entropy $-\sum_{j=0}^{9} y_j^{\text{train}} \cdot \log y_j$ is minimized when $y^{\text{train}} = y$

TensorFlow Code: Training (pp.130-131)

- 4주차처럼 cross_entropy를 최소화하는 weight/bias 찾음
- Dropout을 위한 keep_prob는 훈련시 0.5 (사용시 1.0)
- AdamOptimizer로 optimizer를 변경했음에 유의

```
x_train = tf.placeholder(tf.float32, shape=[None,784])
y_train = tf.placeholder(tf.float32, shape=[None,10])
keep_prob = tf.placeholder(tf.float32)
...
cross_entropy = -tf.reduce_sum(y_in*tf.log(y_conv))
train = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)

num_steps = 1000
for i in range(num_steps):
    sess.run(train, feed_dict = ..., keep_prob:0.5})
```

TensorFlow Code: Testing Trained CNN (pp.130-131)

- 4주차처럼 파일에 저장된 변수들을 읽어서 accuracy 계산
- 학습된 CNN을 사용할 때는 dropout을 하면 안되므로 keep_prob는 1.0
- 99% 이상의 정확도

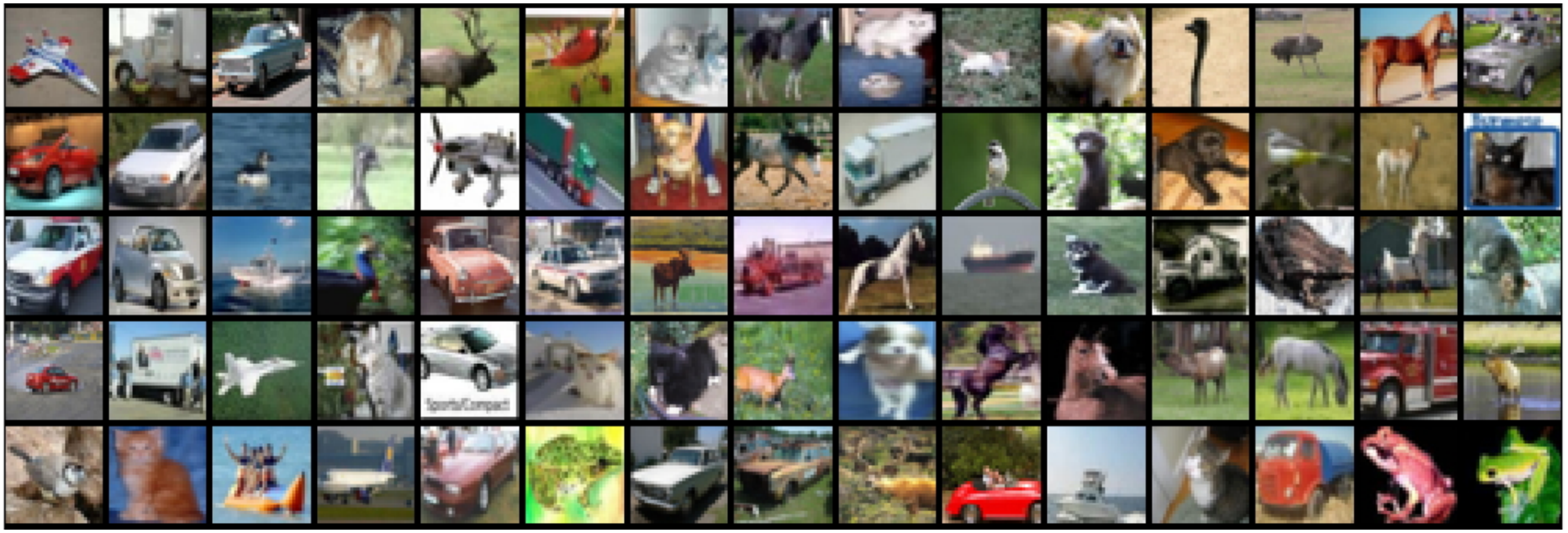
```
...
correct = tf.equal(tf.argmax(y,1), tf.argmax(y_train,1))
accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))

tf.train.Saver().restore(sess, "./model/mnist_model.ckpt")  # re
sess.run(accuracy, feed_dict = ... , keep_prob:1.0})
```

KERAS Implementation of Digit Classifier (CNN version)

https://keras.io/layers/convolutional

```
# 1st convolution-pooling layer
model.add(Convolution2D(filters=32, kernel_size=(5,5), strides=(1,1),
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2), padding="same"))

# 2nd convolution-pooling layer
model.add(Convolution2D(filters=64, kernel_size=(5,5), strides=(1,1),
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2), padding="same"))
model.add(Flatten())

# 1st full-connection layer
model.add(Dense(1000))
model.add(Activation("relu"))
model.add(Dropout(0.5))

# 2nd full-connection layer
model.add(Dense(10))
model.add(Activation("softmax"))
```

Homework: Keras Implementation of Image Classifier



- Train_CNN_cifar.py의 NN_model 함수 구현
  - 지난 시간과 마찬가지로 Train_NN_Keras.py의 NN_mode_with_hidden_layer함수를 참조
- 자세한 hyperparameter는 handout 참조
- Test_NN_cifar.py로 테스트하면 70%의 정확도..
  - 더욱 큰 규모의 모델로 더 올릴 수 있음
- model 파일은 생략하고 .py만 cs3.ksa@gmail.com로 제출