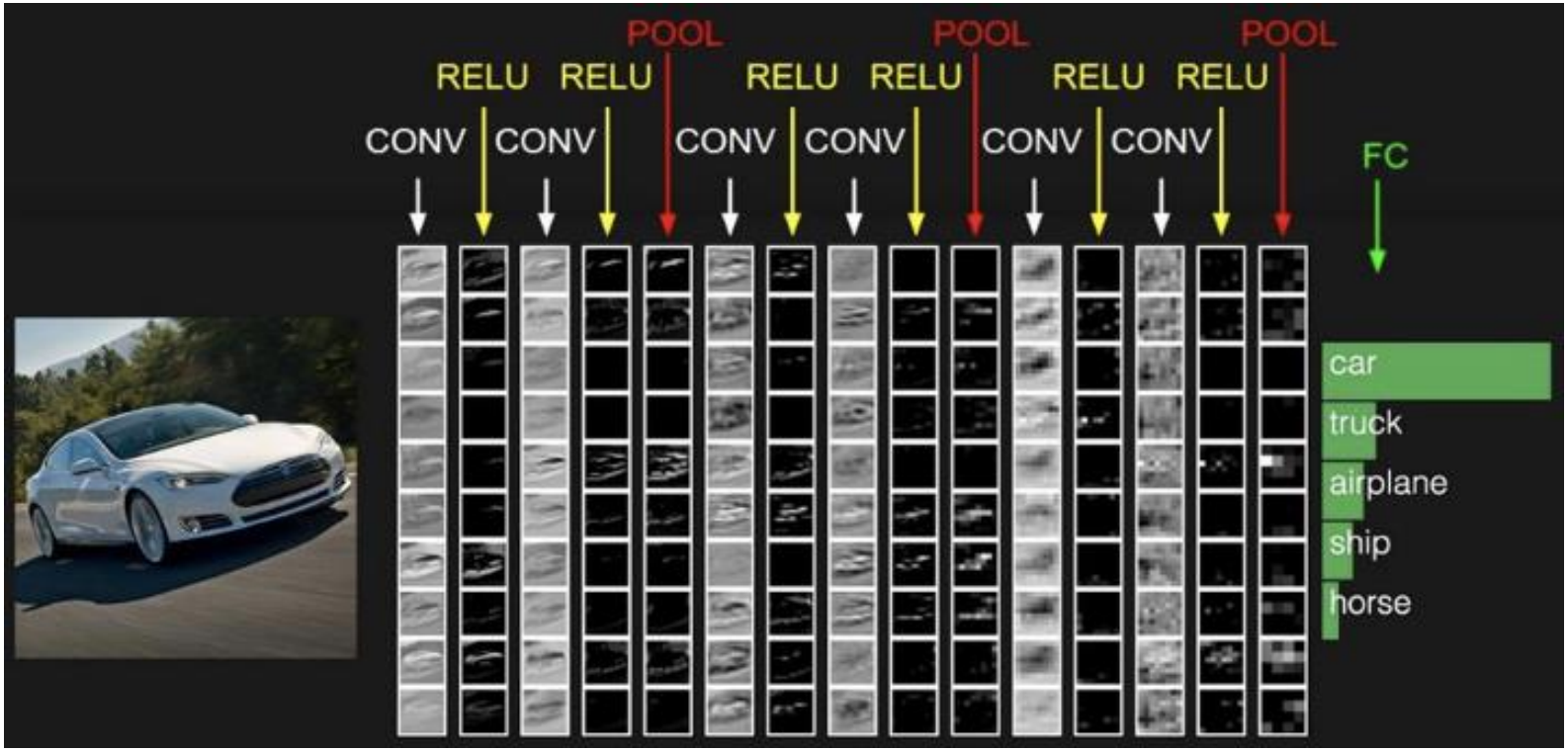
7. CNN(2)

AILab
Hanyang Univ.

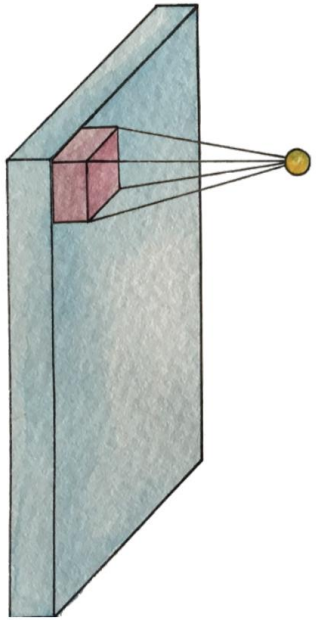
오늘 실습 내용

- CNN review
- Implementing Convolutional Neural Network

CNN review


POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

CNN review : convolution



one number! = Wx + b

(0x1) + (0x1) + (0x1) + (1x0) + … + (1x-1) = -3

Input Image

0	0	0	0	0
1	1	1	1	1
1	1	1	1	1
0	0	0	0	0
0	0	0	0	0

Filter1

1	1	1
0	0	0
-1	-1	-1

Filter2

1	0	-1
1	0	-1
1	0	-1

Feature Map

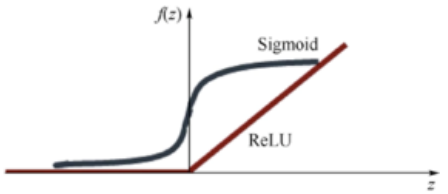
Output 1

-3	-3	-3
0	0	0
3	3	3

Output 2

0	0	0
0	0	0
0	0	0

Activation function 적용



CNN review

(1) Convolution : 하나의 함수와 또 다른 함수를 반전 이동한 값을 곱한 다음, 구간에 대해 적분하여 새로운 함수를 구하는 수학 연산자

(2) Filter : 특성을 검출해주는 함수, 우리가 학습해야 할 weight

(3) Channel : 입력데이터의 Channel = 필터의 Channel / 필터의 개수 = output의 개수(= output channel)

(4) Feature Map : convolution해서 나온 output = 추출된 특성

(5) Stride : 필터를 적용하는 위치의 간격

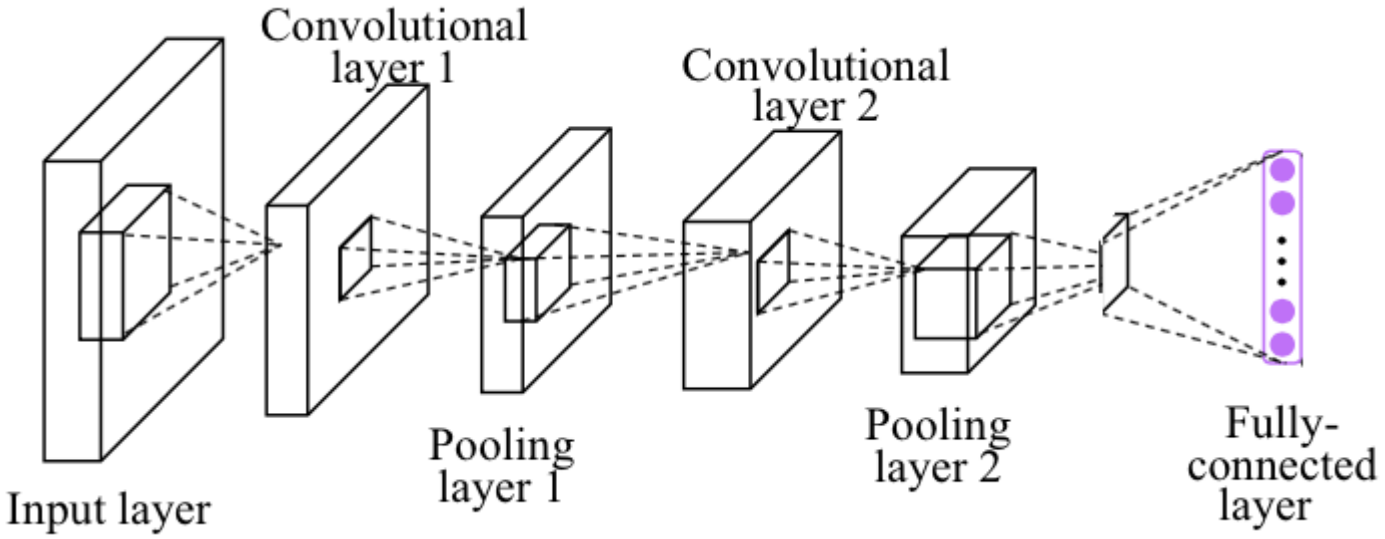
(6) Pooling : 가로 세로 방향의 공간을 줄이는 연산

- 이미지의 크기를 줄이기 때문에 학습할 노드의 수가 줄어들어 학습속도를 높이는 효과
- 출력의 해상도를 낮춰 변형이나 이동에 대한 민감도를 감소
- 하지만, 정보의 손실이 일어남 => Padding 사용
- CNN에서는 일반적으로 Max Pooling 사용

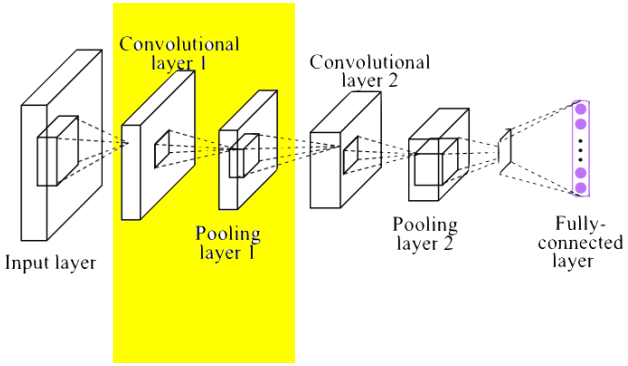
(7) Fully Connected Layer : 이전 계층의 모든 뉴런과 결합된 형태의 레이어 (ex. MLP)

```
W = tf.get_variable(name=name + "_W", shape=[filter_size, filter_size, fin, fout],
                    initializer=tf.contrib.layers.xavier_initializer())
b = tf.get_variable(name=name + "_b", shape=[fout], initializer=tf.contrib.layers.xavier_initializer(0.0))
C = tf.nn.conv2d(din, W, strides=[1, 1, 1, 1], padding='SAME')
R = tf.nn.relu(tf.nn.bias_add(C, b))
```

Implementing Convolutional Neural Network



Implementing Convolutional Neural Network

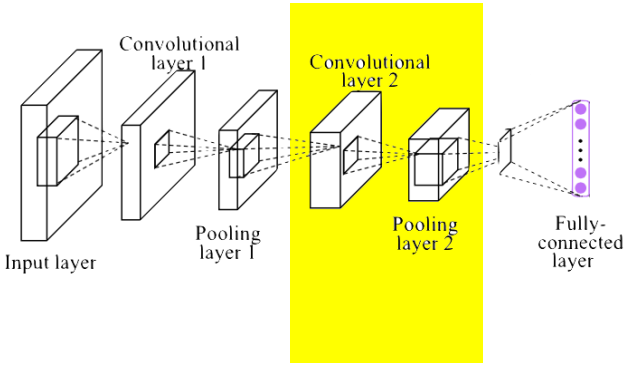


(1) Conv layer 1 + Pooling layer 1

```
# input place holders
X = tf.placeholder(tf.float32, [None, 784]) # img 28x28x1 (black/white)
X_img = tf.reshape(X, [-1, 28, 28, 1])
Y = tf.placeholder(tf.float32, [None, 10])


# L1 ImgIn shape=(?, 28, 28, 1)
W1 = tf.Variable(tf.random_normal([3, 3, 1, 32], stddev=0.01))
#    Conv     -> (?, 28, 28, 32)
#    Pool     -> (?, 14, 14, 32)
L1 = tf.nn.conv2d(X_img, W1, strides=[1, 1, 1, 1], padding='SAME')
L1 = tf.nn.relu(L1)
L1 = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
'''
Tensor("Conv2D:0", shape=(?, 28, 28, 32), dtype=float32)
Tensor("Relu:0", shape=(?, 28, 28, 32), dtype=float32)
Tensor("MaxPool:0", shape=(?, 14, 14, 32), dtype=float32)
'''
```

Implementing Convolutional Neural Network

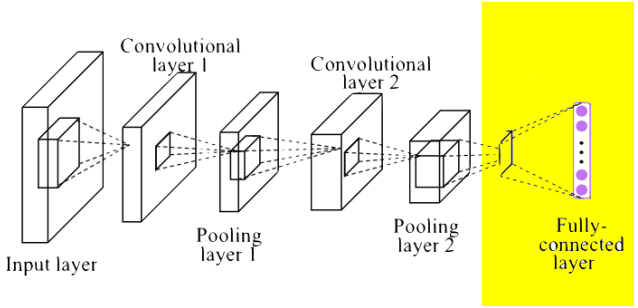


(2) Conv layer 2 + Pooling layer 2

```
'''
Tensor("Conv2D:0", shape=(?, 28, 28, 32), dtype=float32)
Tensor("Relu:0", shape=(?, 28, 28, 32), dtype=float32)
Tensor("MaxPool:0", shape=(?, 14, 14, 32), dtype=float32)
'''


# L2 ImgIn shape=(?, 14, 14, 32)
W2 = tf.Variable(tf.random_normal([3, 3, 32, 64], stddev=0.01))
#    Conv      ->(?, 14, 14, 64)
#    Pool      ->(?, 7, 7, 64)
L2 = tf.nn.conv2d(L1, W2, strides=[1, 1, 1, 1], padding='SAME')
L2 = tf.nn.relu(L2)
L2 = tf.nn.max_pool(L2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
L2 = tf.reshape(L2, [-1, 64 * 7 * 7])
'''
Tensor("Conv2D_1:0", shape=(?, 14, 14, 64), dtype=float32)
Tensor("Relu_1:0", shape=(?, 14, 14, 64), dtype=float32)
Tensor("MaxPool_1:0", shape=(?, 7, 7, 64), dtype=float32)
Tensor("Reshape_1:0", shape=(?, 3136), dtype=float32)
'''
```

Implementing Convolutional Neural Network



(3) Fully-Connected layer

```
'''
Tensor("Conv2D_1:0", shape=(?, 14, 14, 64), dtype=float32)
Tensor("Relu_1:0", shape=(?, 14, 14, 64), dtype=float32)
Tensor("MaxPool_1:0", shape=(?, 7, 7, 64), dtype=float32)
Tensor("Reshape_1:0", shape=(?, 3136), dtype=float32)
'''

# Final FC 64*7*7(3136) inputs -> 10 outputs
W3 = tf.get_variable("W3", shape=[64 * 7 * 7, 10], initializer=tf.contrib.layers.xavier_initializer())
b = tf.Variable(tf.random_normal([10]))
hypothesis = tf.matmul(L2, W3) + b


# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=hypothesis, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```

(4) 결과

hyper parameters

learning_rate = 0.001

training_epochs = 15

batch_size = 100

```
Learning Started!
Epoch: 0001 cost = 0.356375834
Epoch: 0002 cost = 0.095613570
Epoch: 0003 cost = 0.069865302
Epoch: 0004 cost = 0.056996062
Epoch: 0005 cost = 0.047304538
Epoch: 0006 cost = 0.041087208
Epoch: 0007 cost = 0.036835764
Epoch: 0008 cost = 0.032967508
Epoch: 0009 cost = 0.028243347
Epoch: 0010 cost = 0.025517992
Epoch: 0011 cost = 0.022477280
Epoch: 0012 cost = 0.020839543
Epoch: 0013 cost = 0.017557776
Epoch: 0014 cost = 0.015614095
Epoch: 0015 cost = 0.013486644
Learning Finished!
Accuracy: 0.9882

Process finished with exit code 0
```

Deep CNN

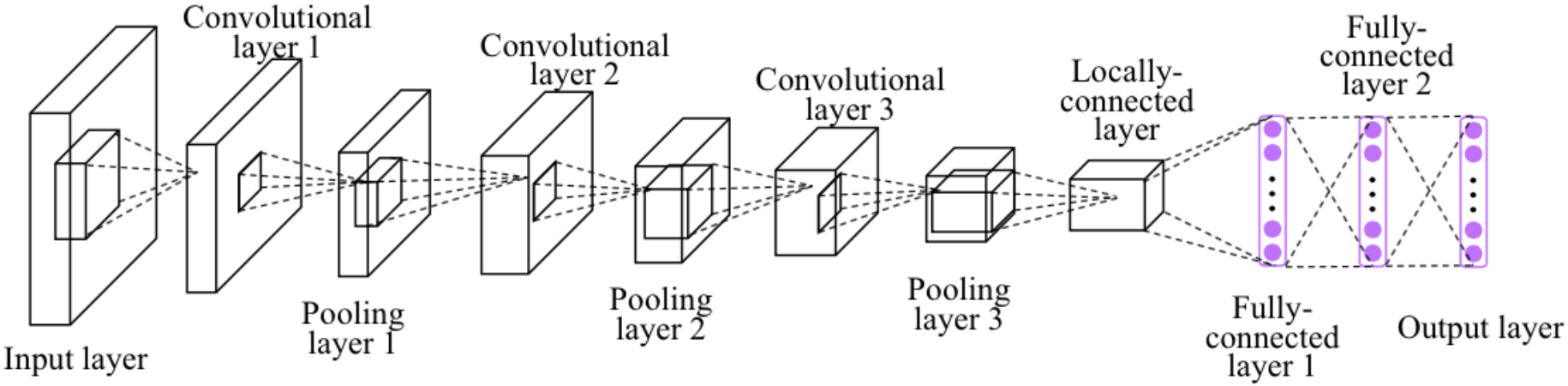


image by http://personal.ie.cuhk.edu.hk/~ccloy/project_target_code/index.html

과제

- CNN 활용하여 MNIST 95% 이상 정확도 보이기
- 소스와 결과 캡쳐 GitLab에 제출
- 과제 기한 : **다음주 수요일 23:59** 까지
- 수업시간에 한 경우 바로 검사받고 **GitLab**에 제출
- GitLab 관련 사용법은 첨부 파일 확인