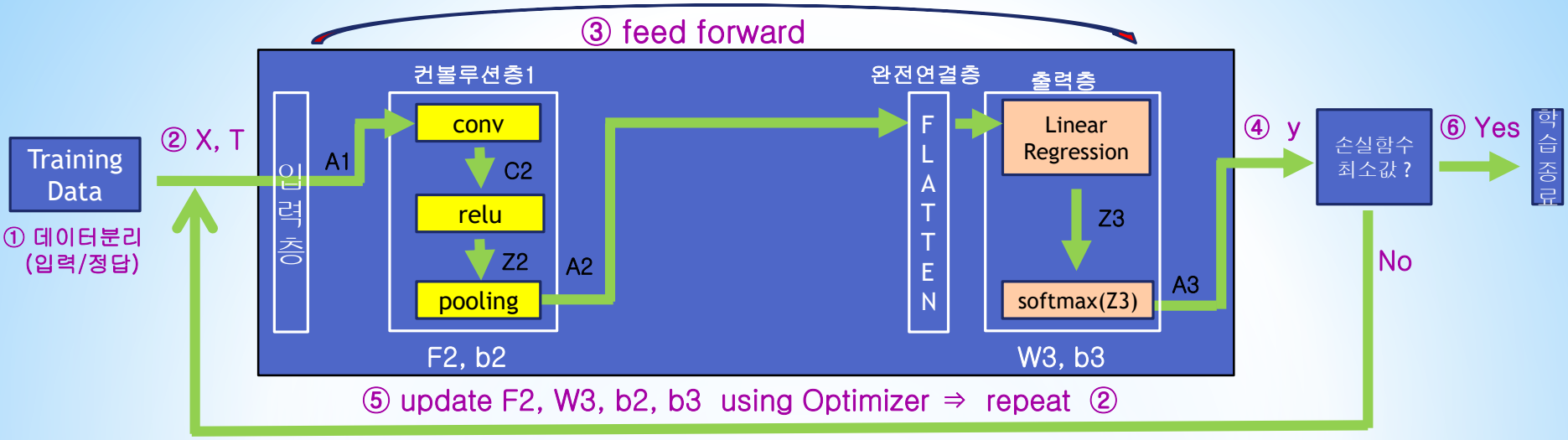
머신러닝/딥러닝을 위한

CNN (IV)

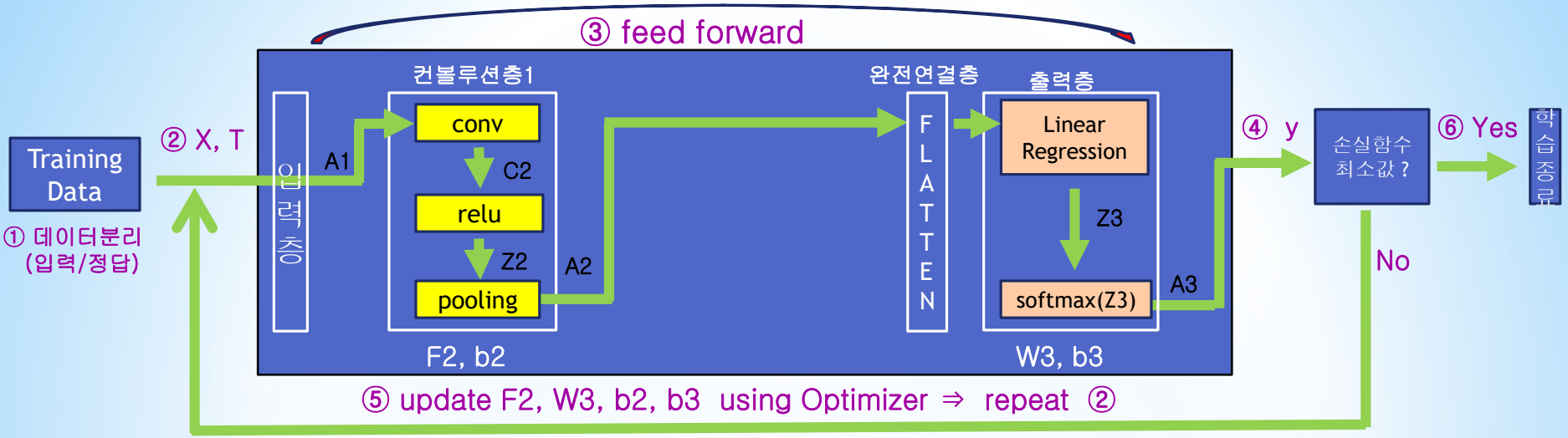
- Basic CNN Architecture (1 conv / 1 flatten) -



	conv 출력	relu 출력	pooling 출력	TensorFlow API
컨볼루션층1	A1 ⊛ F2 + b2 = C2 →	Z2 →	A2	conv: tf.nn.conv2d(…) relu: tf.nn.relu(…) pooling: tf.nn.max_pool(…)

	역할	TensorFlow API
완전연결층	컨볼루션 층의 3차원 출력 값을 1차원 벡터로 평탄화 작업 수행하여 일반 신경망 연결처럼 출력층의 모든 노드와 연결시켜주는 역할 수행	FLATTEN: tf.reshape(A2,…)
출력층	입력받은 값을 출력으로 0~1사이의 값으로 모두 확률화하며 출력 값들의 총합은 항상 1이 되도록 하는 역할 수행	softmax: tf.nn.softmax(Z3)

Overview – TensorFlow API tf.nn.conv2d(…)



conv

tf.nn.conv2d(input, filter, strides, padding,…)

input : 컨볼루션 연산을 위한 입력 데이터이며 [batch, in_height, in_width, in_channels] 예를들어, 100 개의 배치로 묶은 28X28 크기의 흑백 이미지를 입력 으로 넣을경우 **input 은 [100, 28,28, 1]** 로 나타냄

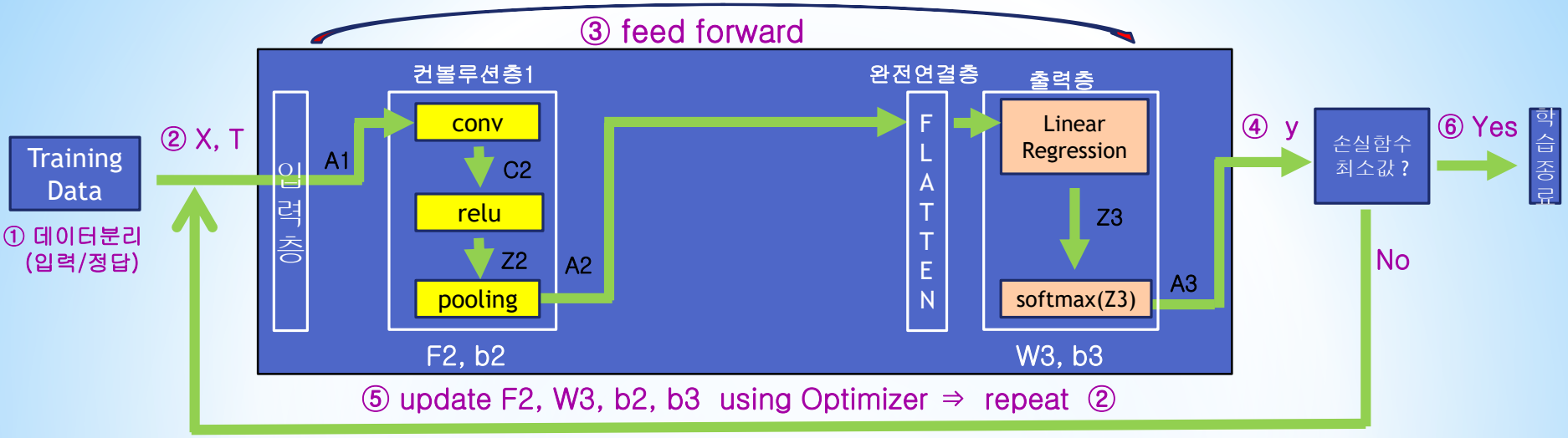
filter : 컨볼루션 연산에 적용할 필터이며 [filter_height, filter_width, in_channels, out_channels] 예를들어, 필터 크기 3X3 이며 **입력채널** 개수는 1이고 적용되는 필터 개수가 총 32개이면 **filter 는 [3, 3, 1, 32]** 로 나타냄

strides: 컨볼루션 연산을 위해 필터를 이동시키는 간격을 나타냄. 예를들어 [1, 1, 1, 1] 로 strides를 나타낸다면 컨볼루션 적용을 위해 1 칸씩 이동 필터를 이동하는것을 의미함

padding: 'SAME' 또는 'VALID' 값을 가짐. padding='VALID' 라면 컨볼루션 연산 공식에 의해서 가로/세로(차원) 크기가 축소된 결과가 리턴됨. 그러나 padding='SAME' 으로 지정하면 입력 값의 가로/세로(차원) 크기와 같은 출력이 리턴되도록 작아진 차원 부분에 0 값을 채운 제로패딩을 수행함

※ **입력채널** : 직관적으로 **데이터가 들어오는 통로**라고 생각하면 무난함. 즉, 입력채널이 1 이라고 하면 데이터가 들어오는 통로가 1개 이며, 만약 입력채널이 32 라고 하면 32개의 통로를 통해서 입력데이터가 들어온다고 판단하면 무리가 없음

Overview – TensorFlow API tf.nn.max_pool(…)



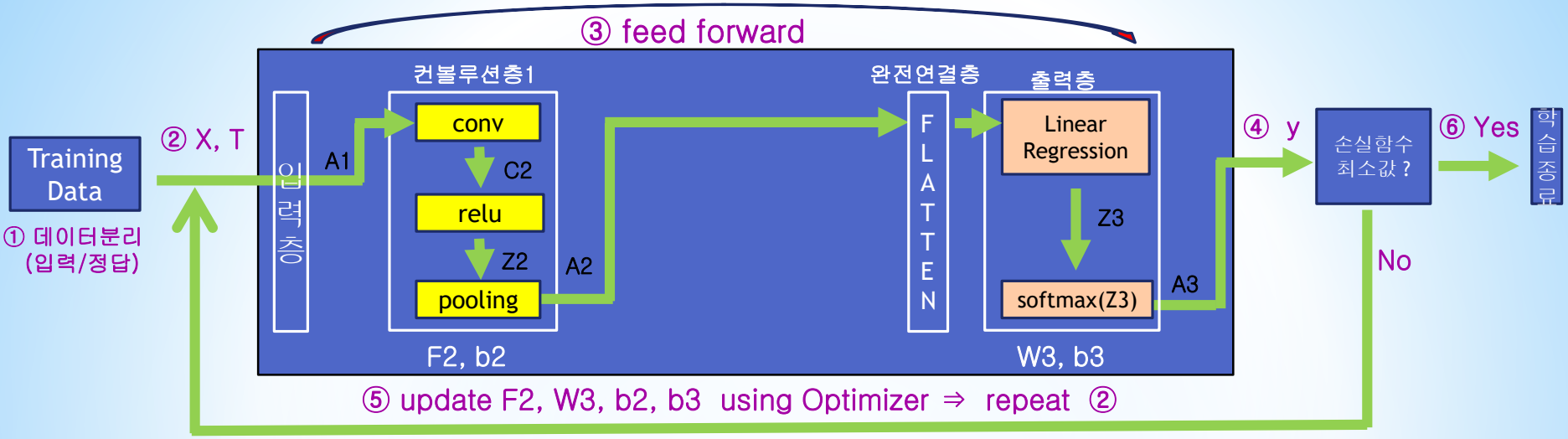
pooling tf.nn.max_pool(value, ksize, strides, padding,…)

value: [batch, height, width, channels] 형식의 입력데이터. 일반적으로 relu 를 통과한 출력결과를 말하며, 위 예제에서는 Z2 값이 value 임

ksize: 컨볼루션 신경망에서 일반적인 ksize는 다음과 같이 [1, height, width, 1] 형태로 표시함. 예를 들어 ksize = [1, 2, 2, 1] 이라면 2칸씩 이동하면서 출력결과 1 개를 만들어 낸다는 것을 의미함. 즉 4개 (2X2) 데이터 중에서 가장 큰 값 1 개를 찾아서 반환하는 역할을 수행함. 만약 ksize = [1, 3, 3, 1] 이라고 하면 3칸씩 이동, 즉 9개 (3X3) 데이터 중에서 가장 큰 값을 찾는다는 의미임

strides: max pooling을 위해 윈도우를 이동시키는 간격을 나타냄. 예를들어 [1, 2, 2, 1] 로 strides를 나타낸다면 max pooling 적용을 위해 2 칸씩 이동하는 것을 의미함

padding: max pooling 에서의 padding 값은 max pooling 을 수행하기에는 데이터가 부족한 경우에 주변을 0 등으로 채워주는 역할을 함. 예를들어 max pooling 에서 풀링층으로 들어오는 입력데이터가 7X7 이고, 데이터를 2개씩 묶어 최대값을 찾아내는 연산을 하기에는 입력으로 주어진 데이터가 부족한 상황임 (즉, 최소 8X8 이어야 가능). 이때 padding='SAME' 이면, 부족한 데이터 부분을 0 등으로 채운 후에 데이터를 2개씩 묶어 최대값을 뽑아낼 수 있음



read_data_sets() 를 통해 객체형태인 mnist로 받아오고 입력데이터와 정답데이터는 MNIST_data/ 디렉토리에 저장이 되는데, one_hot=True 옵션을 통해 정답데이터는 one-hot encoding 형태로 저장됨

mnist 객체는 train, test, validation 3개의 데이터 셋으로 구성되어 있으며. num_examples 값을 통해 데이터의 개수 확인 가능함

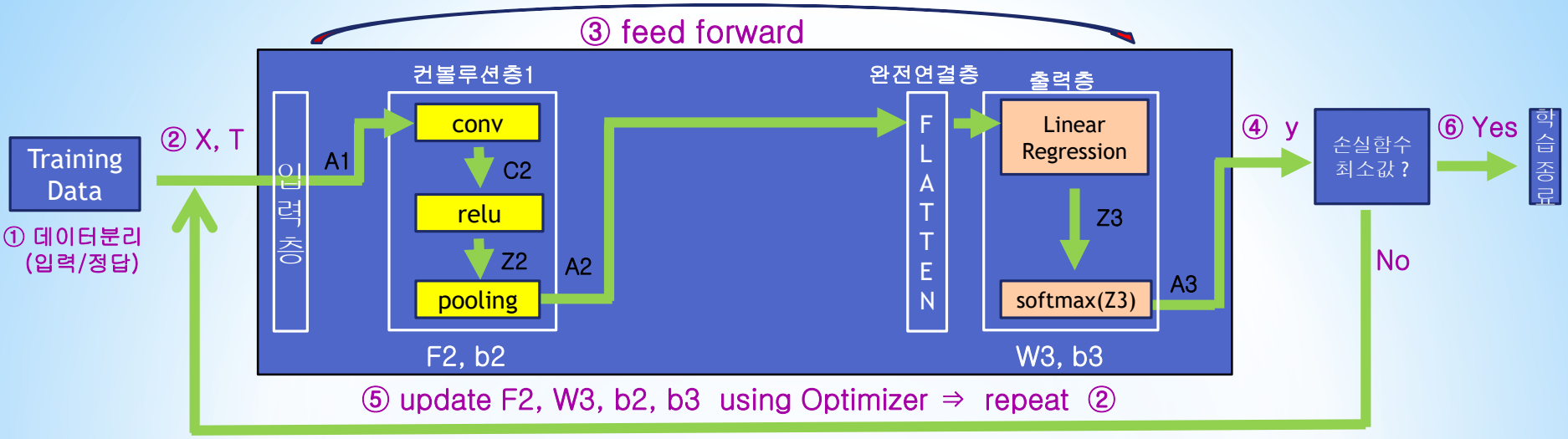
데이터는 784(28x28)개의 픽셀을 가지는 이미지와 one-hot encoding 되어 있는 label(정답)을 가지고 있음

①

```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
import numpy as np
from datetime import datetime


mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

print("")
print("train.num = ", mnist.train.num_examples,
      ", test.num = ", mnist.test.num_examples,
      ", validation.num = ", mnist.validation.num_examples)
```



학습율(learning_rate), 반복횟수(epochs), 한번에 입력으로 주어지는 데이터 개수인 배치 사이즈(batch_size) 등의 하이퍼 파라미터 설정

②

```
# Hyper-Parameter
learning_rate = 0.001   # 학습율
epochs = 30             # 반복횟수
batch_size = 100        # 한번에 입력으로 주어지는 MNIST 개수
```

입력과 정답을 위한 placeholder 노드 정의(X, T)

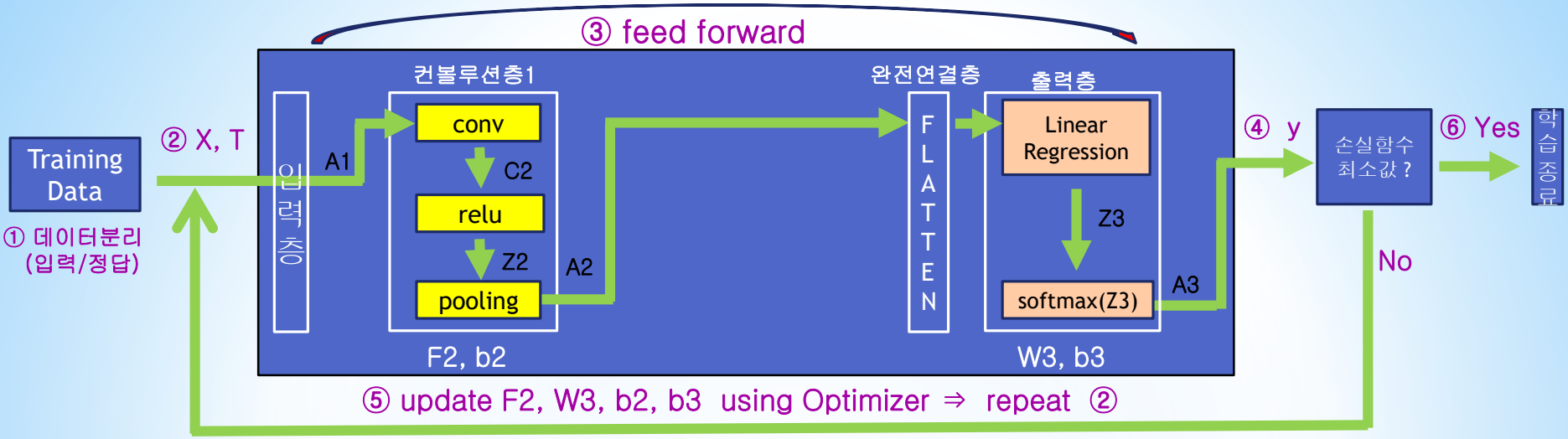
입력층의 출력 값 A1은 784개 픽셀 값을 가지고 있는 MNIST 데이터 이지만 컨볼루션 연산을 수행하기 위해서 28 X 28 X 1 차원을 가지도록 reshape 함.

②

```
# 입력과 정답을 위한 플레이스홀더 정의
X = tf.placeholder(tf.float32, [None, 784])

T = tf.placeholder(tf.float32, [None, 10])

# 입력층의 출력 값. 컨볼루션 연산을 위해 reshape 시킴
A1 = X_img = tf.reshape(X, [-1, 28, 28, 1])   # image 28 X 28 X 1 (black/white)
```



③ 컨볼루션층1

입력채널 (데이터가 들어오는 통로가 1개, 즉 입력 A1 은 1개의 통로를 통해서 들어옴)

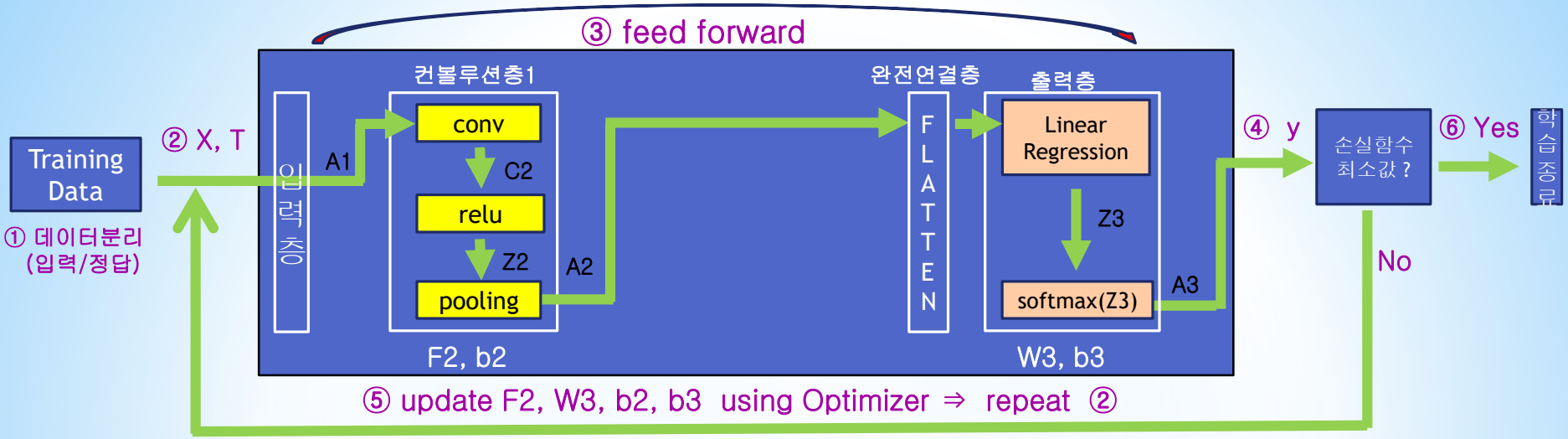
```
# 1번째 컨볼루션 층, 3X3X32 필터
W2 = tf.Variable(tf.random_normal([3, 3, 1, 32], stddev=0.01))
#W2 = tf.Variable(tf.random_normal([3, 3, 1, 32]))
#b2 = tf.Variable(tf.constant(0.1, shape=[32]))
b2 = tf.Variable(tf.random_normal([32]))

# 1번째 컨볼루션 연산을 통해 28 X 28 X1  => 28 X 28 X 32
C2 = tf.nn.conv2d(A1, W2, strides=[1, 1, 1, 1], padding='SAME')

# relu
Z2 = tf.nn.relu(C2+b2)

# 1번째 max pooling을 통해 28 X 28 X 32  => 14 X 14 X 32
A2 = P2 = tf.nn.max_pool(Z2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
```

가중치는 표준편차 0.01 범위에서 random 생성됨



③ 완전연결층

```
# 14X14 크기를 가진 32개의 activation map을 flatten 시킴
A2_flat = P2_flat = tf.reshape(A2, [-1, 14*14*32])
```

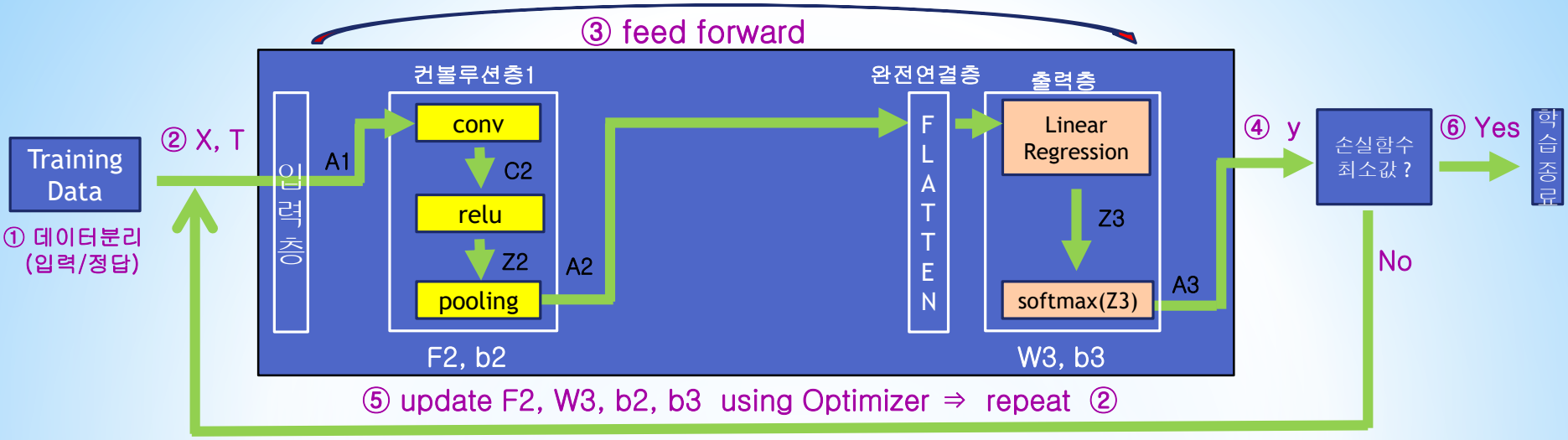
(14*14*32) 노드를 가지도록 reshape 시켜줌. 즉 완전연결층과 출력층(10개 노드)은 (14*14*32) X 10 개 노드가 완전연결 되어 있음

③ 출력층

```
# 출력층
W3 = tf.Variable(tf.random_normal([14*14*32, 10], stddev=0.01))
b3 = tf.Variable(tf.random_normal([10]))

# 출력층 선형회귀  값 Z3, 즉 softmax 에 들어가는 입력 값
Z3 = logits = tf.matmul(A2_flat, W3) + b3     # 선형회귀 값 Z3

y = A3 = tf.nn.softmax(Z3)
```



출력층 선형회귀 값(logits) Z3와 정답 T 를 이용하여 손실 함수 loss 정의. 여기서 한번에 입력되는 배치사이즈가 100 이므로 Z3와 T shape 은 (100X10) 이며, tf.nn.softmax_cross_entropy_with_logits_v2(…) 에 의해 100 개의 데이터에 대해 각각의 소프트맥스가 계산된 후에 정답과의 비교를 통해 크로스 엔트로피 손실 함수 값이 계산되고, tf.reduce_mean(…) 에 의해서 100개의 손실함수 값의 평균이 계산됨

④, ⑤

```
loss = tf.reduce_mean( tf.nn.softmax_cross_entropy_with_logits_v2(logits=Z3, labels=T) )

optimizer = tf.train.AdamOptimizer(learning_rate)

train = optimizer.minimize(loss)
```

출력층 선형회귀 값(logits) Z3 와 정답 T 를 이용하여 cross entropy 손실함수 계산

one-hot encoding 에 의해서 출력 층 계산 값 A3와 정답 T 는 (batch_size X 10) shape을 가지는 행렬임. 따라서 argmax 의 두번째 인자에 1을 주어 행 단위로 A3와 T를 비교함. 위 예에서는 batch_size 가 100 이므로 (100X10) 행렬에 대해서 행 단위로 비교하고 있음

⑥

```
# batch_size X 10 데이터에 대해 argmax를 통해 행단위로 비교함
predicted_val = tf.equal( tf.argmax(A3, 1), tf.argmax(T, 1) )

# batch_size X 10 의 True, False 를 1 또는 0 으로 변환
accuracy = tf.reduce_mean(tf.cast(predicted_val, dtype=tf.float32))
```

출력층의 계산 값 A3와 정답 T 에 대해, 행 단위로 값을 비교함

```
with  tf.Session()  as sess:

    sess.run(tf.global_variables_initializer())  # 변수 노드(tf.Variable) 초기화

    start_time = datetime.now()

    for i  in range(epochs):    # 30 번 반복수행

        total_batch = int(mnist.train.num_examples / batch_size)  # 55,000 / 100

        for step in range(total_batch):

            batch_x_data, batch_t_data = mnist.train.next_batch(batch_size)

            loss_val, _ = sess.run([loss, train], feed_dict={X: batch_x_data, T: batch_t_data})

            if step % 100 == 0:
                print("epochs = ", i, ", step = ", step, ", loss_val = ", loss_val)

    end_time = datetime.now()

    print("₩nelapsed time = ", end_time - start_time)

    # Accuracy 확인
    test_x_data = mnist.test.images     # 10000 X 784
    test_t_data = mnist.test.labels     # 10000 X 10

    accuracy_val = sess.run(accuracy, feed_dict={X: test_x_data, T: test_t_data})

    print("₩nAccuracy = ", accuracy_val)
```

[실습] 앞의 예제에서 AdamOptimizer를,
GradientDescentOptimizer 로 변경하여 테스트

=> 3X3 크기의 32개 필터, 1 stride, 2X2 max pooling, padding 있음