8. Convolutional Neural Networks

CNN은 1989년 LeCun의 논문에서 소개 되었다. 필기체 숫자 이미지를 인식하기 위한 모델로 동물의 시각 정보 처리과정을 모델링한 신경망 구조이다. 시각 정보가 세포에 들어오면 모든 신경세포에 자극이 전달되는 것이 아니라 해당 수용영역의 세포에서 자극을 받아들이는데 이를 모델링하여 컨볼루션 과정과 풀링 과정을 진행하는 신경망을 만든 것이다.

가. CNN 개요

그림 8-1은 학습된 CNN이 이미지를 식별하는 과정을 그림으로 보여주는 것이다. 첫 번째 층의 출력을 보면 입력이미지에 대해서 다양한 필터를 적용시킨 결과를 확인할 수 있다. 이후 층이 깊어질수록 이미지를 추상화시키며 특징을 추출하는 것을 확인할 수 있다. 신경망에 입력이 들어오면 입력의 공간영역 혹은 시간영역에서 컨볼루션 연산을 수행한다. 그림 8-2는 CNN의 기본구조를 개념적으로 보여주고 있다.

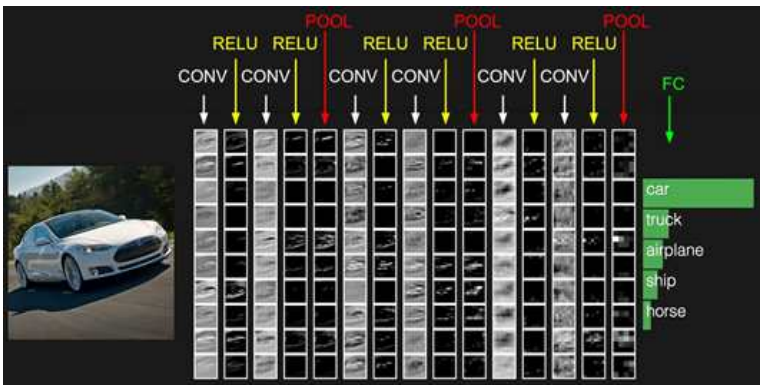

그림 8.1 CNN 각 층의 출력의 예

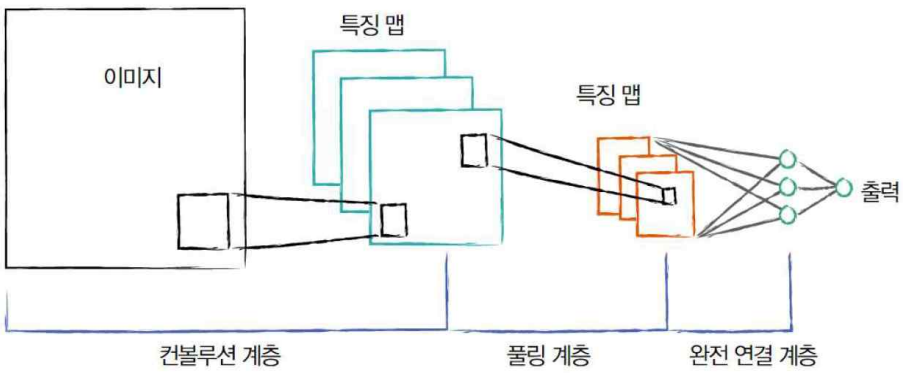

그림 8.2 CNN의 기본구조

나. 컨볼루션

그림 8-3에서는 2차원의 입력에 대한 컨볼루션 연산 과정을 보여준다. 입력영상의 일부가 필터와 가중합되어 출력되는데 하나의 필터가 시프트되어 전체이미지에 대한 출력을 계산한다.

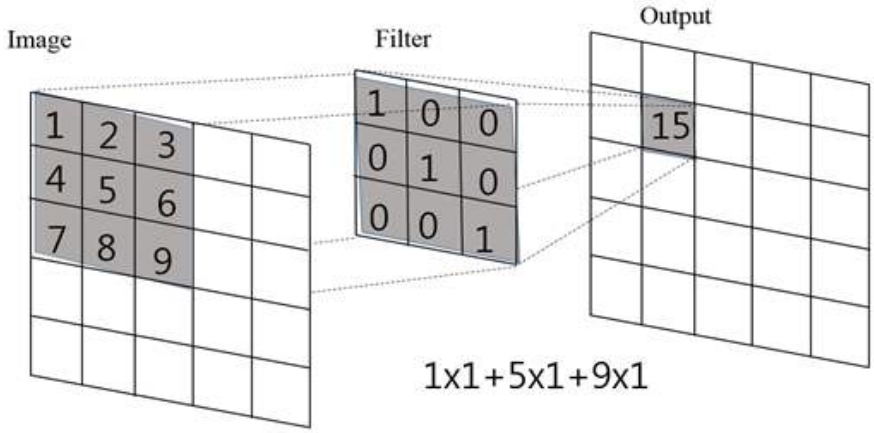

그림 8.3 컨볼루션 연산과정

실제 연산은 이미지와 필터의 상호상관(cross-correlation)으로 이루어지며 입력 이미지 I와 $W_f \times H_f$크기의 필터 F에 대한 출력 O의 관계는 식 8-1과 같이 나타낼 수 있다.

$$O(m,n) = \sum_{w=0}^{W_f-1} \sum_{h=0}^{H_f-1} I(m+w,\, n+h) F(w,h) \tag{8-1}$$

필터는 특징이 데이터에 있는지 여부를 검출해주는 함수이다. 예를 들어 곡선을 검출해주는 필터가 있다고 한다면, 곡선이 그려져 있는 곳에서는 큰 값이 나오고 없는 부분은 0에 수렴한다. 이로써 데이터유무를 확인할 수 있다. 아래 그림 8-4의 왼쪽 메트릭스를 보면 곡선을 따라서 30의 값이 있는 것을 확인할 수 있다.

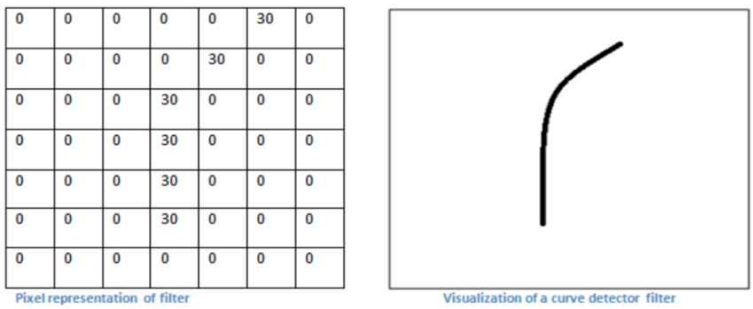

그림 8.4 컨볼루션에서 필터의 역할

컨볼루션 연산은 신경망의 구조로 인해 sparse connectivity 와 shared weight의 특성을 가진다. Sparse connectivity는 층 사이에 모든 유닛을 연결하는 것이 아니라 필터의 크기만큼 입력의 일부만 다음 층과 연결함으로써 나타나는 특징이다. 데이터의 표현능력은 저하되지만 모델의 복잡도는 낮아지는 효과를 얻을 수 있다. 또한 공간적으로 인접한 신호들에 대한 상관 관계가 비선형 필터를 통해 추출되므로 서로 다른 필터들을 사용하면 다양한 로컬 특징을 추출해 낼 수 있다. Shared weight는 하나의 필터를 전체 입력에 대해 반복적으로 사용함으로써 추정해야 하는 파라미터의 수가 필터의 크기만큼으로 줄어드는 특성이다. 이미 sparse connectivity 특성으로 인하여 학습시켜야 하는 가중치들의 수가 줄어든 상태에서 필터의 크기에 해당하는 수의 가중치를 전체 입력에 대해 적용함으로써 모델의 복잡도는 크게 낮아진다.

다. Padding

컨볼루션 연산을 수행 시, 고려해야 할 점이 있다. CNN에서는 필터를 하나가 아닌 여러 단계에 걸쳐서 계속 필터를 연속적으로 적용하여 특징을 추출하는 것을 최적화 해나가는데, 필터 적용 후에 결과 값이 작아지게 된다는 것은 데이터의 유실을 의미한다. 이를 방지하기 위해서 padding기법을 사용한다. 테두리에다가 0으로 값을 둔 하나의 입력을 가상적으로 있다고 만들어 두는 것이다. 이와 같은 이유는 2가지가 있는데, 1) 그림이 작아지는 것을 방지하며, 2) 패딩이 들어간 부분이 모서리라는 것을 알리는 기능이다.

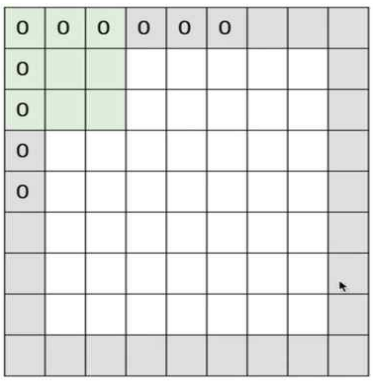

그림 8.5 CNN에서 패딩(Padding)

라. Stride

필터를 원본이미지에 적용하기 위해 왼쪽으로부터 지정한 값만큼 칸을 이동시키면서 특징을 추출해야 하는데, 이때 필터를 적용하는 간격을 Stride라고 한다. 또한 stride는 컨볼루션 연산을 수행할 때 필터를 시프트하는 크기를 의미하며 컨볼루션 연산의 연산량을 줄이기 위해 사용된다.

하나의 축에 대해 입력의 크기가 X, 필터의 크기는 K, 제로 패딩 사이즈는 P, stride의 크기는 S 일 때, 출력의 크기 Y는 식 8-2와 같이 나타낼 수 있다. 식 8-2의 결과가 정수가 아닐 경우 신경망이 제대로 동작할 수 없으므로 신경망의 설계 시 반드시 고려해야 한다.

$$Y = \frac{X - K + 2P}{S} + 1 \qquad (8\text{-}2)$$

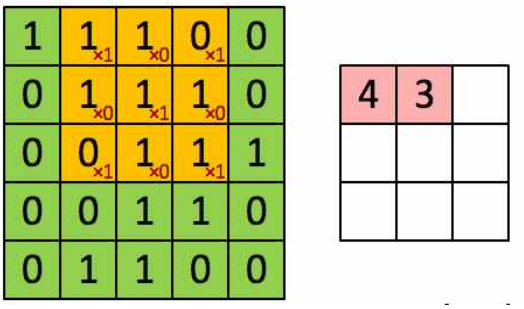

그림 8.6 CNN에서 Stride(stride=1인 경우)

마. 풀링(Pooling)

컨볼루션 과정 후에는 풀링 과정을 거치게 되는데 일종의 sub-sampling과정이다. 이와 같이 특징 수를 줄이는 것을 서브샘플링(sub-sampling)이라고 하며 이것을 통해 전체 특징의 수를 의도적으로 줄이는 이유는 데이터크기를 줄여 컴퓨터파워를 절약할 수 있고, 데이터가 줄어드는 과정에서 유실되기 때문에 오버피팅을 방지할 수 있다.

대표적인 pooling 방법으로는 max pooling과 average pooling이 있는데 max pooling은 신경이 입력된 자극들 중에 강한 신호만 전달하고 나머지는 무시하는 특성을 반영한 과정이고, average pooling은 입력된 자극들의 평균을 전달하는 과정이다. 일반적으로 이미지 식별에서는 max pooling을 많이 사용한다. 그림 8-7은 max pooling의 예를 보여준다.

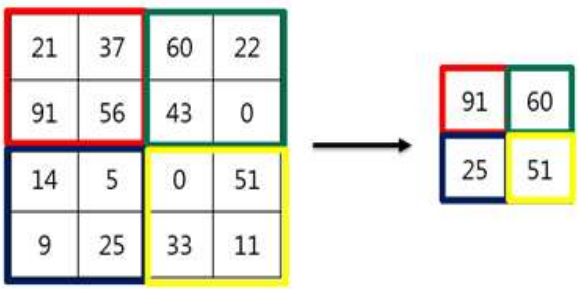

그림 8.7 CNN에서 Pooling(Max pooling)

바. CNN 구조

일반적인 신경망의 경우 각 층의 뉴런은 벡터의 형태로 구성되어있지만 CNN의 컨볼루션 층과 풀링 층은 W×H×C 의 3차원 구조로 이루어져 있다. 여기서 W와 H 는 각각 넓이와 높이를 뜻하고, C는 채널을 뜻한다. 그림 8-8은 일반적인 CNN의 예이다. 입력이미지가 RGB 이미지라면 3개의 채널을 갖는 입력이 된다.

이후 컨볼루션 층에서 채널의 수는 필터의 수와 같다. 컨볼루션 층의 필터의 수는 임의로 선택이 가능하며 필터의 형태에 따라 각각 다른 특징을 입력에서 추출할 수 있게 된다. 필터의 형태는 학습에 의해 목적에 맞도록 결정되며 이로써 사용자가 특징추출에 관여하지 않고 입력데이터만 입력하면 컨볼루션층에서 특징추출기능을 수행할 수 있다.

컨볼루션과 풀링층의 뒤에는 추출한 특징을 기반으로 분류를 수행하는 신경망층이 존재한다. 이 층은 일반적인 신경망과 같이 뉴런이 벡터의 형태로 구성되어 있고 모든 뉴런이 가중치로 연결되어 있다.

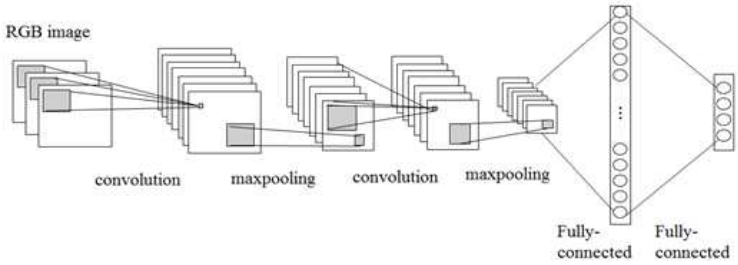

그림 8.8 CNN의 실제 구조의 예

사. 다양한 CNN 모델

CNN은 딥러닝 연구에서 가장 활발히 연구되어 왔던 구조이다. 특히 이미지분류 문제에서 좋은 성능을 보여왔다. MNIST, CIFAR, ImageNet 과 같은 이미지 분류를 위한 대표적인 데이터셋들에 대하여 다양한 구조의 CNN이 연구되어 왔다. 그 중 LeNet-5는 가장 먼저 발표되어 이미지 분류문제에서 CNN의 성능을 보여준 모델이다. 그 이후 GoogLeNet, VGGNet 와 같은 모델이 연구되어 성능이 향상되어 왔다. 본 절에서는 앞서 연구된 다양한 구조의 CNN모델을 소개한다.

1) Lenet-5

LeNet-5는 Lecun의 논문에서 소개된 CNN 구조체이다. 0~9까지의 필기체 숫자이미지를 인식해내는 신경망으로 두 단계의 컨볼루션 층과 두 단계의 full connection 은닉 층을 가지고 있다. 그림 8-9는 LeNet-5의 구조를 나타낸다.

총 7개의 layer로 구성되어 있으며, 3개의 컨볼루션 층, 2개의 sub-sampling layer, 그리고 하나의 fully connection layer와 하나의 출력 층을 포함한다. 첫 번째 컨볼루션 층에서 입력에 대한 6가지 kernel을 적용해 특징을 추출하고, 두 번째 컨볼루션 층에서는 앞에서 추출한 특징에 대해 10가지 kernel을 적용하여 특징을 추출한다. 각 특징 추출과정 후에는 sub-sampling layer를 통해 이미지를 1/4의 크기로 줄인다. 세 번째 컨볼루션 층에서는 kernel의 크기와 특징의 크기가 동일하므로 커널을 적용한 fully connection layer와 같다. 마지막으로 10개의 출력 노드를 가진 출력 층으로 10개의 클래스를 식별해내게 된다. Lenet-5 신경망의 특징은 단순한 신경망인데도 위상변화와 잡음에 강한 성능을 가진다는 것이다.

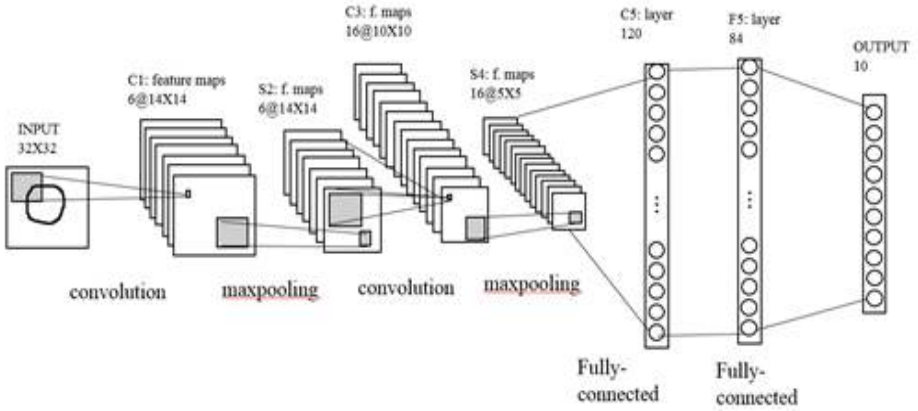

그림 8.9 Lenet-5 모델

2) VGGNet

ILSVRC(ImageNet Large Scale Visual Recognition Challenge)는 'ImageNet' 데이터셋을 분류하는 경연대회로, 2010년부터 개최되어 왔다. 2012년 토론토 대학 소속 팀이 AlexNet이라는 8층의 CNN을 사용하여 이전 대회까지 25.8%에 머물러 있던 분류 오류율을 16.4%까지 향상시킨 이 후, CNN은 이미지분류에서 활발히 연구되기 시작하였다.

VGGNet은 옥스포드 대학의 VGG(Visual Geometry Group)팀이 개발한 모델로,앞서 설명한 2014년 IRSVRC에서 7.3%의 분류 오류율로 준우승한 CNN구조이다. 19층의 깊이의 CNN 구조이고 구조상으로 GoogLeNet에 비해 단순하나 0.6%차이로 유사한 성능을 보여준다. VGGNet은 필터의 크기를 3x3으로 고정한 후 신경망의 깊이가 성능에 어떠한 영향을 주는지 실험으로 연구한 결과물이다.

그림 8-10은 VGG팀이 실험한 VGGNet의 구조를 보여준다. 11, 13, 16, 19개 층으로 깊이를 변화시키며 실험을 진행하였다. 실험결과 층이 깊어질수록 분류 오류율이 감소 하였지만 16층 이상의 구조에서는 성능에 큰 차이가 없었다. 망이 깊어질수록 vanishing gradient 문제가 발생할 가능성이 높아지므로 이를 해결하기 위해 사전 학습된 신경망의 가중치를 사용하여 더 깊은 신경망의 가중치를 초기화 하였다. 그 결과 vanishing gradient 문제를 해결함과 동시에 학습시간을 단축시킬 수 있었다.

그림 8-10에 나타난 구조를 살펴보면 3x3컨볼루션층을 쌓아 올린 형태를 확인할 수 있는데 3x3컨볼루션을 2층으로 쌓을 경우 5x5컨볼루션 필터를 사용한 효과를 얻을 수 있고, 3층으로 쌓을 경우 7x7 컨볼루션 필터를 사용한 효과를 얻을 수 있다. 하지만 그것을 여러 층의 3x3컨볼루션 필터로 구성함으로써 학습해야 할 파라미터의 수가 감소하여 학습시간이 단축되고, 비선형 함수가 추가되어 성능이 향상된다.

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					

그림 8.10 VGGNet의 구조

3) GoogleNet(Inception model)

GoogLeNet은 2014년 ILSVRC에서 6.7%의 분류 오류율로 우승한 모델이다. GoogLeNet은 22층으로 구성된 CNN이며, 많은 수의 층을 효과적으로 학습시키기 위해 인셉션이라는 구조를 적용하였다. 그림 8-11은 인셉션의 구조를 보여준다. 같은 층에 1x1컨볼루션, 3x3컨볼루션, 5x5컨볼루션 등의 서로 다른 크기의 필터가 적용되어 다양한 특징을 추출한 후 결과를 합하도록 구성되어있고, 1x1컨볼루션을 사용하여 2개의 층으로 깊이를 증가시켜 놓은 것을 확인할 수 있다. 일반적으로 깊이를 증가시키면 연산량이 증가하게 되는데 1x1컨볼루션은 필터의 수를 감소시켜 연산량을 줄이는 역할을 한다. 이로 인해, 다양한 필터를 사용하고 깊이를 증가시키면서도 연산량을 줄일 수 있다는 장점을 가진다.

실제로 GoogLeNet은 AlexNet에 비해 14개 층이 더 깊지만 추정해야 할 파라미터의 수와 연산량은 오히려 더 적다. 그림 8-12는 GoogLeNet의 전체 구조를 표로 나타낸 것이다. 총 9개의 인셉션 구조가 사용되었으며, 각 인셉션 구조는 1x1컨볼루션의 영향으로 2개층의 깊이를 가진다.

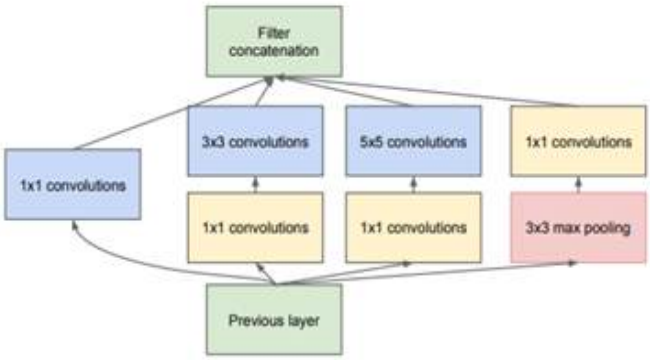

그림 8.11 GoogleNet의 인셉션(Inception) 구조

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

그림 8.12 GoogleNet의 구조

아. CNN 모델 구현

1) 데이터 load

```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("./mnist/data/", one_hot=True)
```

2) 신경망 모델 구성을 위한 데이터 정의

```
X = tf.placeholder(tf.float32, [None, 28, 28, 1])
Y = tf.placeholder(tf.float32, [None, 10])
keep_prob = tf.placeholder(tf.float32)
```

- 기존 모델에서는 입력 값을 28x28 하나의 차원으로 구성하였으나, CNN 모델을 사용하기 위해 2차원 평면 구조로 만든다.
- tf.placeholder 함수인자 [None, 28, 28, 1]
  : None -> 입력 데이터의 개수
  : 28, 28 -> 28x28 이미지
  : 1 -> 그레이 이미지이므로 채널수가 1, 컬러의 경우 RGB의 3개의 채널이므로 3

3) CNN 망 구성 : 첫 번째 층의 구성

```
W1 = tf.Variable(tf.random_normal([3, 3, 1, 32], stddev=0.01))
L1 = tf.nn.conv2d(X, 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')
```

- W1 [3 3 1 32] -> [3 3]: 커널 크기, 1: 입력값 X 의 채널수, 32: 필터 개수
- stride=[1,1,1,1] -> 필터링 시 한칸씩 움직인다.
- L1 Conv shape=(?, 28, 28, 32)
- Pool -> (?, 14, 14, 32), strides=[1, 2, 2, 1] -> 슬라이딩 시 2칸씩 움직인다.
- ksize=[1,2,2,1] : pooling 커널사이즈가 2x2로 한다.
- tf.nn.conv2d 를 이용해 한칸씩 움직이는 컨볼루션 레이어를 쉽게 만든다.
- padding='SAME' 은 커널 슬라이딩시 최외곽에서 한칸 밖으로 더 움직이는 옵션

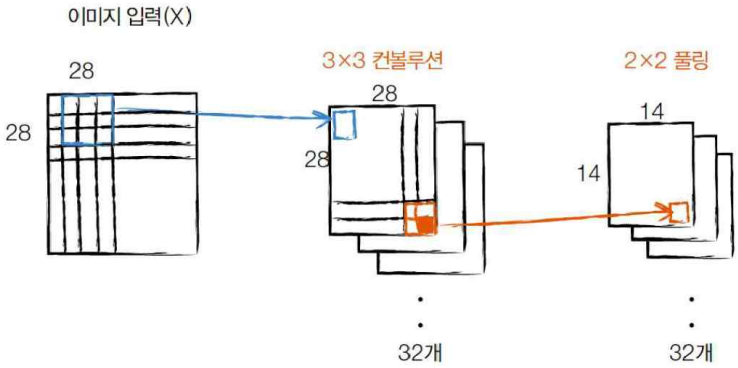

그림 8.13 CNN 첫 번째 계층의 구성

4) CNN 망 구성 : 두 번째 층의 구성

```
W2 = tf.Variable(tf.random_normal([3, 3, 32, 64], stddev=0.01))
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 Conv shape=(?, 14, 14, 64)
- Pool ->(?, 7, 7, 64)
- W2 의 [3,3,32,64] 에서 32는 L1 에서 출력된 W1 의 마지막 차원, 필터의 크기

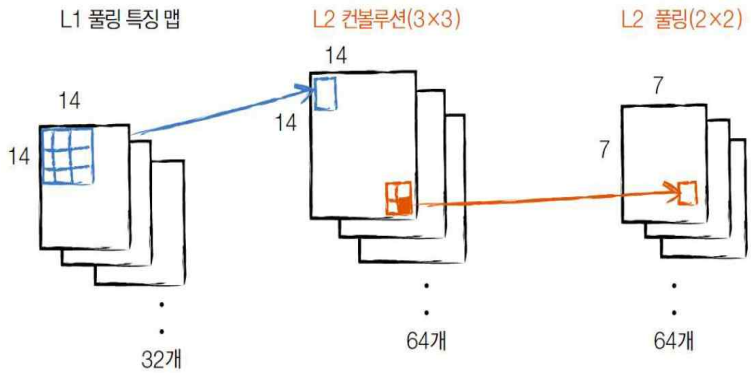

그림 8.14 CNN 두 번째 계층의 구성

5) CNN 망 구성 : 세 번째 층의 구성

```
W3 = tf.Variable(tf.random_normal([7 * 7 * 64, 256], stddev=0.01))
L3 = tf.reshape(L2, [-1, 7 * 7 * 64])
L3 = tf.matmul(L3, W3)
L3 = tf.nn.relu(L3)
L3 = tf.nn.dropout(L3, keep_prob) #drop-out 적용
```

- FC 레이어: 입력값 7x7x64 -> 출력값 256
- Full connect를 위해 직전의 Pool 사이즈인 (?,7,7,64)를 참고하여 차원을 줄인다.
- Reshape ->(?, 256) :

6) CNN 망 구성 : 네 번째 층의 구성(Fully connected layer : 완전 연결층)

```
W4 = tf.Variable(tf.random_normal([256, 10], stddev=0.01))
model = tf.matmul(L3, W4)
```

- 최종 출력값 L3 출력 256개를 입력값으로 받아서 0~9 레이블인 10개의 출력 값

7) 비용함수 및 최적화

```
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
        logits=model, labels=Y))
optimizer = tf.train.AdamOptimizer(0.001).minimize(cost)
```

- 아래처럼 최적화 함수를 RMSPropOptimizer 로 바꿔서 사용 가능

```
#optimizer = tf.train.RMSPropOptimizer(0.001, 0.9).minimize(cost)
```

8) 신경망 모델 학습

```
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
batch_size = 100
total_batch = int(mnist.train.num_examples / batch_size)

for epoch in range(15):
    total_cost = 0
    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        #이미지 데이터를 CNN 모델을 위한 자료형태인 [28 28 1]로 재구성
        batch_xs = batch_xs.reshape(-1, 28, 28, 1)
        _, cost_val = sess.run([optimizer, cost],
                               feed_dict={X: batch_xs,Y: batch_ys, keep_prob: 0.7})
        total_cost += cost_val

    print('Epoch:', '%04d' % (epoch + 1),
          'Avg. cost =', '{:.3f}'.format(total_cost / total_batch))

print('최적화 완료!')
```

9) 결과 확인

```
is_correct = tf.equal(tf.argmax(model, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
print('정확도:', sess.run(accuracy,
                 feed_dict={X: mnist.test.images.reshape(-1, 28, 28, 1),
                            Y: mnist.test.labels, keep_prob: 1}))
```

< Example 8-1 > CNN을 이용한 MNIST 식별

```
import tensorflow as tf

from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("./mnist/data/", one_hot=True)

#신경망 모델 구성
# 기존 모델에서는 입력 값을 28x28 하나의 차원으로 구성하였으나,
# CNN 모델을 사용하기 위해 2차원 평면과 특성치의 형태를 갖는 구조로 만듭니다.
X = tf.placeholder(tf.float32, [None, 28, 28, 1])
Y = tf.placeholder(tf.float32, [None, 10])
keep_prob = tf.placeholder(tf.float32)

# 각각의 변수와 레이어는 다음과 같은 형태로 구성됩니다.
# W1 [3 3 1 32] -> [3 3]: 커널 크기, 1: 입력값 X 의 특성수, 32: 필터 갯수
# L1 Conv shape=(?, 28, 28, 32)
#    Pool      ->(?, 14, 14, 32)
W1 = tf.Variable(tf.random_normal([3, 3, 1, 32], stddev=0.01))
# tf.nn.conv2d 를 이용해 한칸씩 움직이는 컨볼루션 레이어를 쉽게 만들 수 있습니다.
# padding='SAME' 은 커널 슬라이딩시 최외곽에서 한칸 밖으로 더 움직이는 옵션
L1 = tf.nn.conv2d(X, W1, strides=[1, 1, 1, 1], padding='SAME')
L1 = tf.nn.relu(L1)
# Pooling 역시 tf.nn.max_pool 을 이용하여 쉽게 구성할 수 있습니다.
L1 = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
# L1 = tf.nn.dropout(L1, keep_prob)

# L2 Conv shape=(?, 14, 14, 64)
#    Pool      ->(?, 7, 7, 64)
# W2 의 [3, 3, 32, 64] 에서 32 는 L1 에서 출력된 W1 의 마지막 차원, 필터의 크기
W2 = tf.Variable(tf.random_normal([3, 3, 32, 64], stddev=0.01))
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.nn.dropout(L2, keep_prob)

# FC 레이어: 입력값 7x7x64 -> 출력값 256
# Full connect를 위해 직전의 Pool 사이즈인 (?, 7, 7, 64) 를 참고하여 차원을 줄인다
#    Reshape  ->(?, 256)
W3 = tf.Variable(tf.random_normal([7 * 7 * 64, 256], stddev=0.01))
L3 = tf.reshape(L2, [-1, 7 * 7 * 64])
L3 = tf.matmul(L3, W3)
L3 = tf.nn.relu(L3)
L3 = tf.nn.dropout(L3, keep_prob)

# 최종 출력값 L3 에서의 출력 256개를 입력값으로 받아서 0~9 레이블인 10개의 출력값
W4 = tf.Variable(tf.random_normal([256, 10], stddev=0.01))
model = tf.matmul(L3, W4)

cost=tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=model, labels=Y))
optimizer = tf.train.AdamOptimizer(0.001).minimize(cost)
# 최적화 함수를 RMSPropOptimizer 로 바꿔서 결과를 확인해봅시다.
# optimizer = tf.train.RMSPropOptimizer(0.001, 0.9).minimize(cost)
```

< Example 8-1 > 계속

```
# 신경망 모델 학습
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

batch_size = 100
total_batch = int(mnist.train.num_examples / batch_size)

for epoch in range(15):
    total_cost = 0

    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        # 이미지 데이터를 CNN 모델을 위한 자료형태인 [28 28 1] 의 형태로 재구성합니
다.
        batch_xs = batch_xs.reshape(-1, 28, 28, 1)

        _, cost_val = sess.run([optimizer, cost],
                               feed_dict={X: batch_xs,
                                          Y: batch_ys,
                                          keep_prob: 0.7})
        total_cost += cost_val

    print('Epoch:', '%04d' % (epoch + 1),
          'Avg. cost =', '{:.3f}'.format(total_cost / total_batch))

print('최적화 완료!')

# 결과 확인
is_correct = tf.equal(tf.argmax(model, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
print('정확도:', sess.run(accuracy,
                          feed_dict={X: mnist.test.images.reshape(-1, 28, 28, 1),
                                     Y: mnist.test.labels,
                                     keep_prob: 1}))
```

< Example 8-2 > tf.layers API를 이용한 CNN 모델 : MNIST 식별

```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("./mnist/data/", one_hot=True)

#신경망 모델 구성
X = tf.placeholder(tf.float32, [None, 28, 28, 1])
Y = tf.placeholder(tf.float32, [None, 10])
is_training = tf.placeholder(tf.bool)

# 기본적으로 inputs, outputs size, kernel_size 만 넣어주면
# 활성화 함수 적용은 물론, 컨볼루션 신경망을 만들기 위한 나머지 수치들은 알아서 계산
# 특히 Weights 를 계산하는데 xavier_initializer 를 사용
L1 = tf.layers.conv2d(X, 32, [3, 3], activation=tf.nn.relu)
L1 = tf.layers.max_pooling2d(L1, [2, 2], [2, 2])
L1 = tf.layers.dropout(L1, 0.7, is_training)

L2 = tf.layers.conv2d(L1, 64, [3, 3], activation=tf.nn.relu)
L2 = tf.layers.max_pooling2d(L2, [2, 2], [2, 2])
L2 = tf.layers.dropout(L2, 0.7, is_training)

L3 = tf.contrib.layers.flatten(L2)
L3 = tf.layers.dense(L3, 256, activation=tf.nn.relu)
L3 = tf.layers.dropout(L3, 0.5, is_training)

model = tf.layers.dense(L3, 10, activation=None)

cost=tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=model, labels=Y))
optimizer = tf.train.AdamOptimizer(0.001).minimize(cost)

#신경망 모델 학습
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
batch_size = 100
total_batch = int(mnist.train.num_examples/batch_size)

for epoch in range(15):
    total_cost = 0

    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        batch_xs = batch_xs.reshape(-1, 28, 28, 1)
        _, cost_val = sess.run([optimizer, cost],
                               feed_dict={X: batch_xs,Y: batch_ys,is_training: True})
        total_cost += cost_val
    print('Epoch:', '%04d' % (epoch + 1),
          'Avg. cost =', '{:.4f}'.format(total_cost / total_batch))
print('최적화 완료!')
#결과 확인
is_correct = tf.equal(tf.argmax(model, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
print('정확도:', sess.run(accuracy,
                          feed_dict={X: mnist.test.images.reshape(-1, 28, 28, 1),
                                     Y: mnist.test.labels, is_training: False}))
```