

# 딥러닝 7주차 - CNN편

CNN (합성곱 신경망)

## 0. CNN 역사

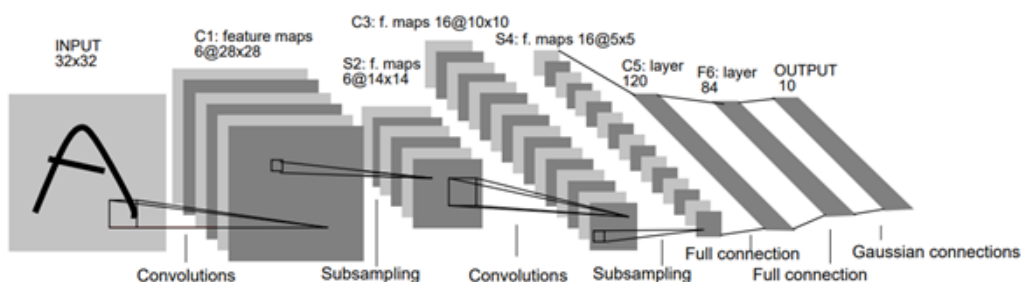
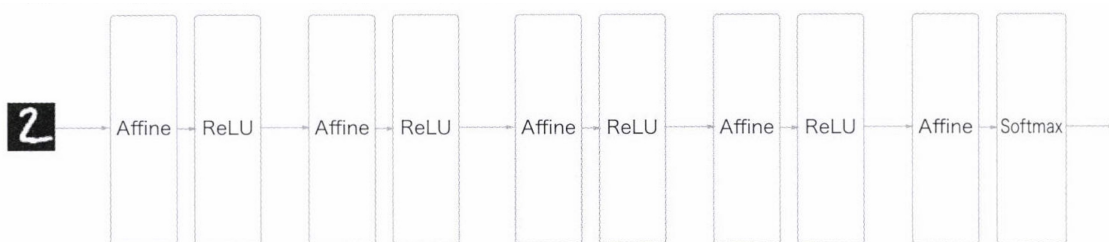


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

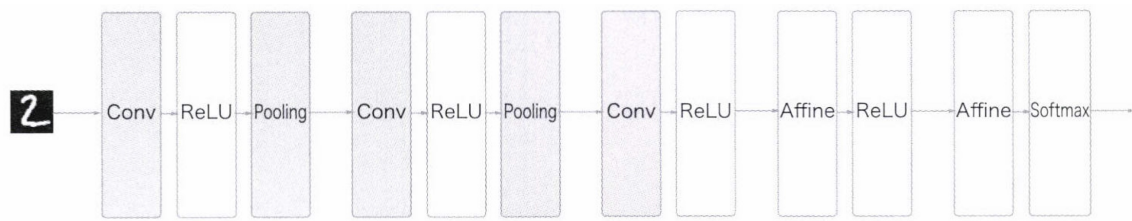
LeNet 구조의 그림

## 1. 전체 구조

이전 신경망 설계 += 합성곱 계층(convolutional layer) + 풀링계층(Pooling layer)



(그림1 Affine 계층)



(그림2 CNN 구조)

Conv = 합성곱 계층

Pooling = 풀링 계층

## 2. 합성곱 계층

# 패딩 # 스트라이드

### Affine 계층의 문제점

→ 데이터의 형상을 무시하기 때문!

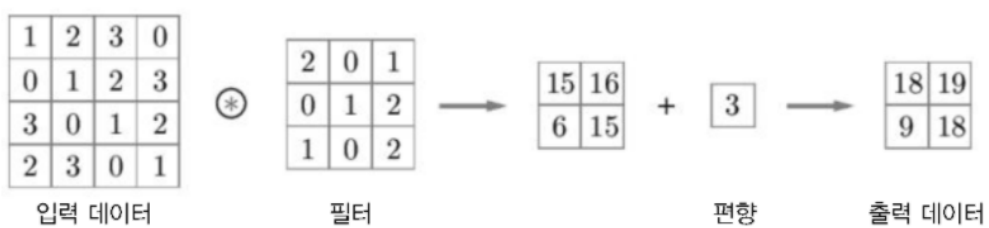
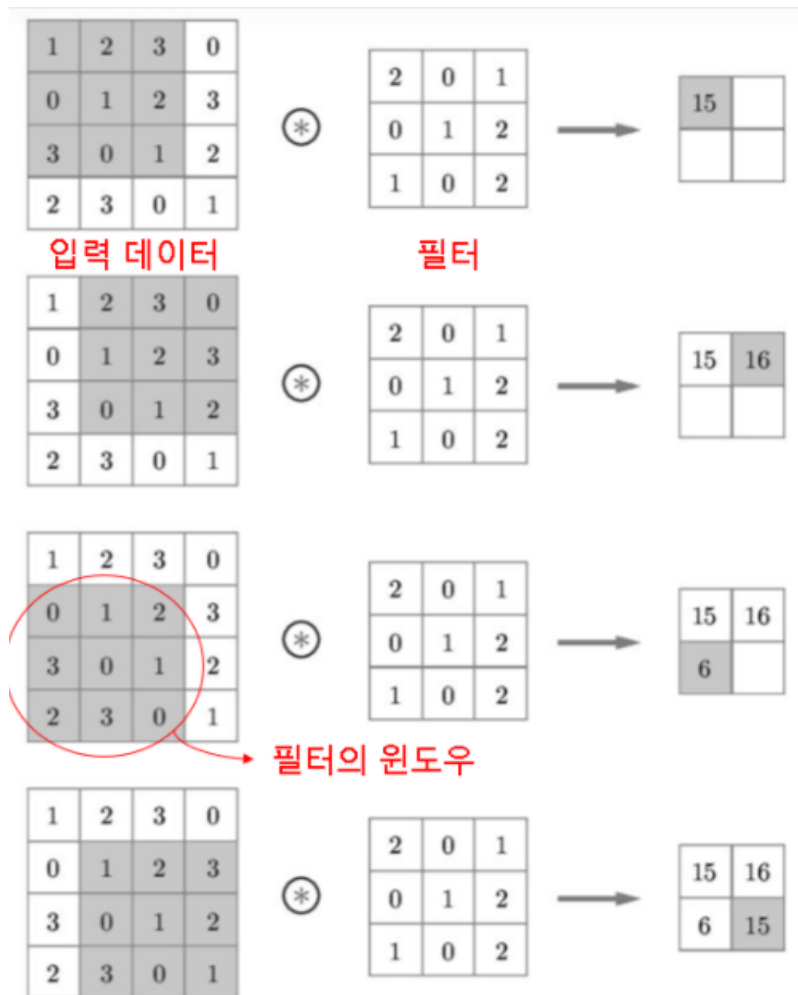
```
network = TwoLayerNet(input_size=784, hidden_size=50, output_size=10)
```

반대로 합성곱 계층은 형상을 유지한다. 3차원 데이터를 입력받고 다음 계층에도 3차원 데이터를 입력받는다. 이미지 자체를 제대로 이해할 수 있는 특징을 갖고 있다.

## 합성곱 연산

필터 연산 : 이미지 처리를 담당하는 합성곱 연산을 의미

커널 : 필터와 동일한 뜻을 의미함. (그림 참고)

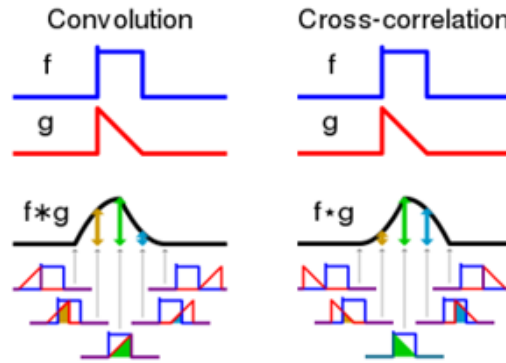


편향을 고려한다면 나오는 결과

편향은 항상 (1 \* 1) 형태를 띠고 있다!

## 교차상관과 합성곱 연산의 차이

참고 : p.230



참고) 교차상관과 차이

- 교차상관 공식 (참고용)

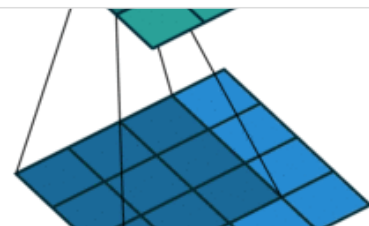
$$(f * g)(t) = \int_{-\infty}^{\infty} f(\alpha)g(t + \alpha)d\alpha$$

$$(f * g)(i, j) = \sum_{x=0}^{h-1} \sum_{y=0}^{w-1} f(x, y)g(i + x, j + y)$$

### Convolution vs Cross-correlation

합성곱convolution 혹은 컨볼루션 신경망은 주로 시각 분야 애플리케이션에 널리 사용됩니다. 위키피디아의 합성곱 정의를 보면 "하나의 함수와 또 다른 함수를 반전 이동한 값을 곱한 다음, 구간에 대해 적분하여 새로운 함수를 구하는 수학 연산자이

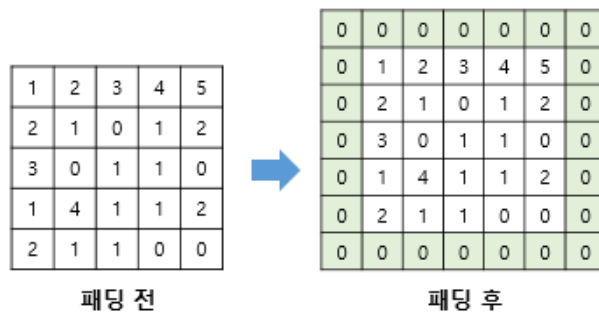
📄 <https://tensorflow.blog/2017/12/21/convolution-vs-cross-correlation/>



합성곱과 교차상관에 대해 더 알고 싶으시다면 😊

## Padding

합성곱 연산을 수행하기 전에 입력 데이터 주변을 특정 값으로 채우는 과정



### 합성곱 신경망 - 합성곱 연산과 교차상관 연산

합성곱은 두 함수에 적용하여 새로운 함수를 만드는 수학 연산자이다. 두 배열  $x$ 와  $w$ 가 있을 때, 둘 중 한 배열의 원소 순서를 뒤집은 후 왼쪽 부터 각 배열 원소끼리 곱한 후 더하는 연산 이다. 수식으로는  $x * w$ 로 표기한다. 위와 같은 방식으로 끝까지

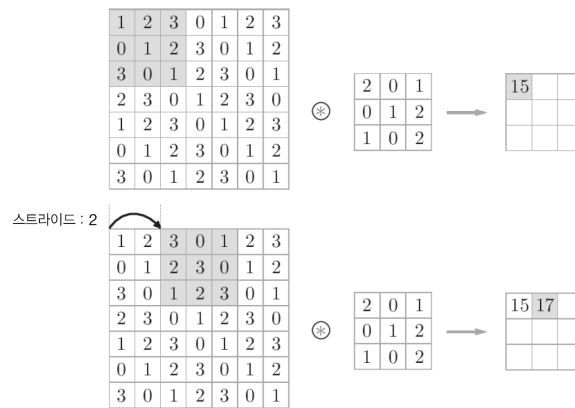
☺ <https://hyjykelly.tistory.com/62>

Deep Learning

Padding의 종류를 더 알고 싶으시다면 😊

## Stride

→ 필터를 적용하는 위치의 간격



stride 부가 설명 그림

- 입력 크기 ( $H, W$ )
- 필터 크기 ( $FH, FW$ )
- 출력 크기 ( $OH, OW$ )

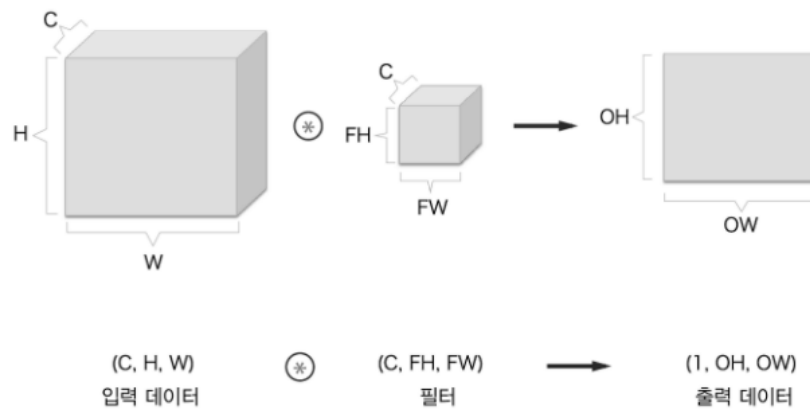
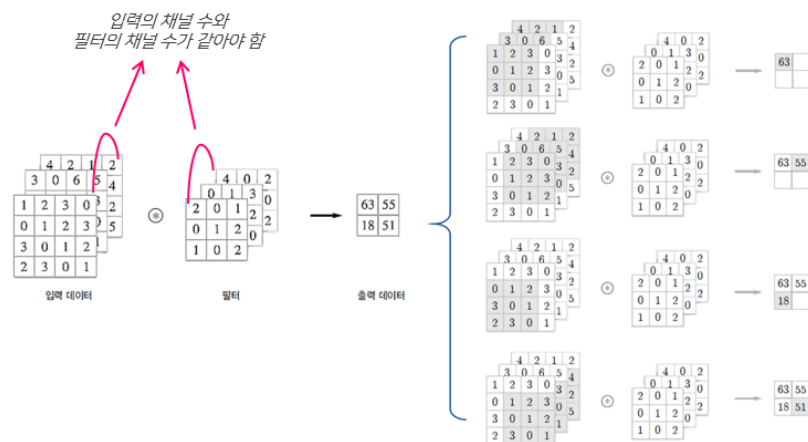
- Padding (  $P$  )
- Stride (  $S$  )

→ 출력 크기를 구하는 공식

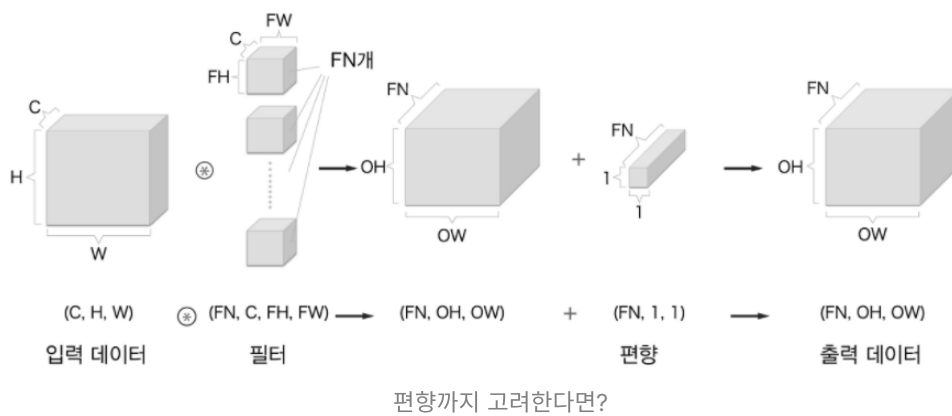
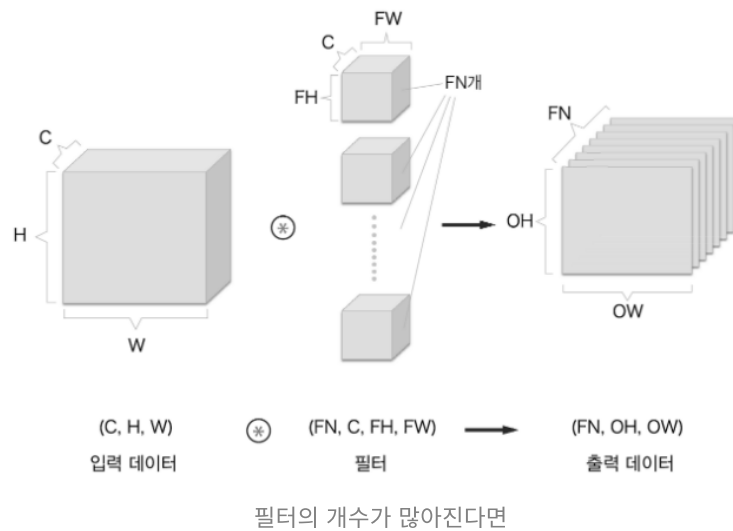
$$OH = (H + 2P - FH) / S + 1$$

$$OW = (W + 2P - FW) / S + 1$$

### 3차원 합성곱 연산



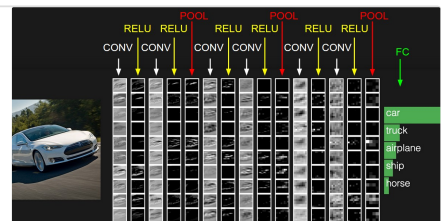
블록의 형상을 이해해야 합니다 😊



## CS231n Convolutional Neural Networks for Visual Recognition

Table of Contents: Convolutional Neural Networks are very similar to ordinary Neural Networks from the previous chapter: they are made up of neurons that have learnable weights and biases. Each neuron

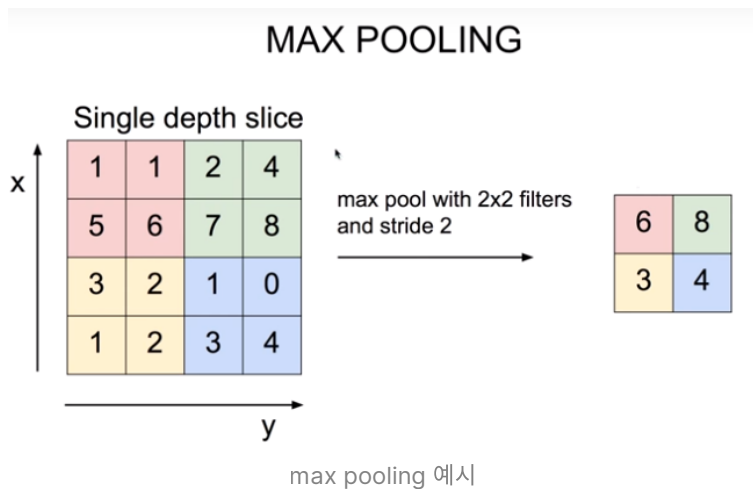
<https://cs231n.github.io/convolutional-networks/>



## Pooling 계층

세로와 가로 방향을 줄이는 연산

- Max Pooling (최대 풀링)



- 풀링 계층의 특징
  1. 학습해야 할 매개변수가 없다.
  2. 채널 수가 변하지 않는다.
  3. 입력의 변화에 영향을 적게 받는다.
- 풀링 계층을 사용하는 이유는?
  1. 데이터 차원의 감소(Overfitting 방지)
  2. VGG 이후부터는 pooling 과정에서 데이터에 대한 정보들이 날아가기 때문에 stride를 조정해서 합성곱 계층에서 차원을 감소시킨다.

#### Unsupervised Feature Learning and Deep Learning Tutorial

After obtaining features using convolution, we would next like to use them for classification. In theory, one could use all the extracted features with a classifier such as a softmax classifier, but this can be computationally challenging. Consider for instance images of size 96×96 pixels, and suppose we have learned 400 features over 8×8 inputs.  
<http://ufldl.stanford.edu/tutorial/supervised/Pooling/>

## Pooling, Conv 구현하기

### im2col은 뭘까?

→ for 문 대신 사용하는 함수이자, 4차원 혹은 3차원의 입력값으로 2차원으로 변환해준다.



단점 : stride를 적절하게 지정해도 중복되는 영역이 생길 수 밖에 없어서 기존 원소의 수보다 더 많은 원소를 다룬다. 메모리 소비에서 안 좋은 점을 갖고 있지만, 행렬 계산의 면에서는 좋은 방식으로 알려져있다.

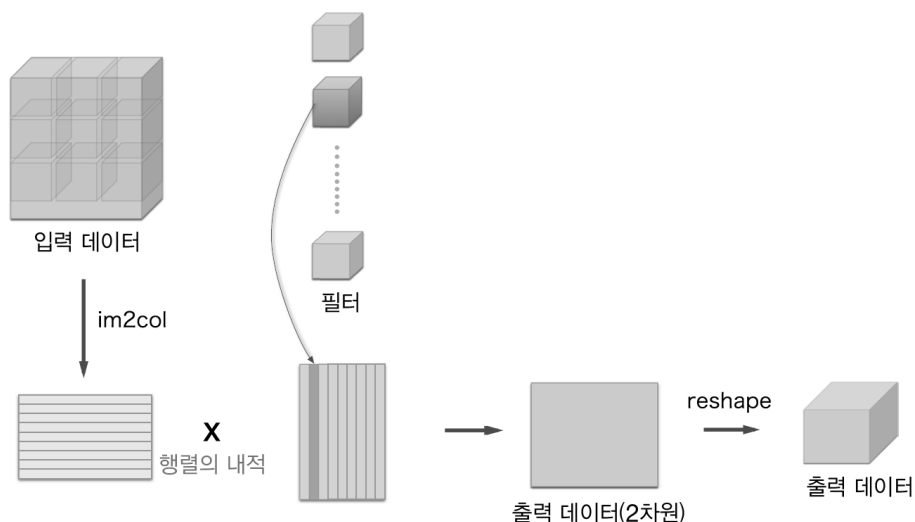


그림 참고 : im2col을 이용한 합성곱 연산 과정

```
def im2col(input_data, filter_h, filter_w, stride=1, pad=0):
    N, C, H, W = input_data.shape
    out_h = (H + 2 * pad - filter_h) // stride + 1
    out_w = (W + 2 * pad - filter_w) // stride + 1

    img = np.pad(input_data, [(0, 0), (0, 0), (pad, pad), (pad, pad)], 'constant')
    col = np.zeros((N, C, filter_h, filter_w, out_h, out_w))

    for y in range(filter_h):
        y_max = y + stride * out_h
        for x in range(filter_w):
            x_max = x + stride * out_w
            col[:, :, y, x, :, :] = img[:, :, y:y_max:stride, x:x_max:stride]

    col = col.transpose(0, 4, 5, 1, 2, 3).reshape(N * out_h * out_w, -1)
    return col
```

## Convolution 계층

```
class Convolution:
    def __init__(self, W, b, stride=1, pad=0):
        self.W = W # 필터(가중치)
        self.b = b # 편향 bias
        self.stride = stride # 스트라이드
        self.pad = pad # 패딩
```

```
def forward(self, x):
    FN, C, FH, FW = self.W.shape #필터 개수, 채널, 필터 높이, 필터 너비
    N, C, H, W = x.shape
    out_h = int(1 + (H + 2*self.pad - FH) / self.stride)
    out_w = int(1 + (W + 2*self.pad - FW) / self.stride)

    col = im2col(x, FH, FW, self.stride, self.pad) # 입력 데이터 전개
    col_W = self.W.reshape(FN, -1).T # 2차원 배열로 필터 전개
    out = np.dot(col, col_W) + self.b # 전개된 두 행렬의 곱

    out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)

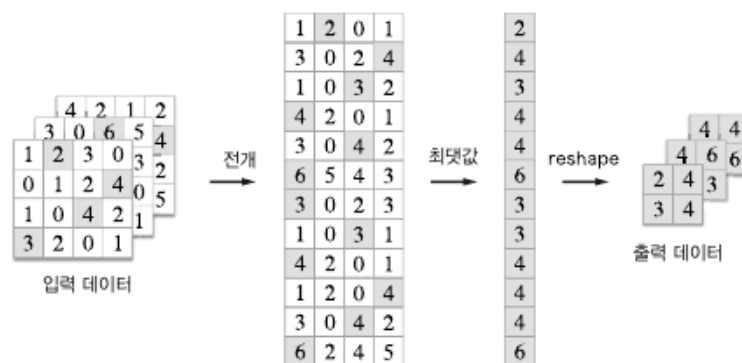
    return out
```

## Pooling 계층

→ 채널들이 독립적으로 분리되어 있기 때문에 합성곱 계층과 다르다.

구현 과정

1. 입력 데이터를 전개한다.
2. 행별 최대값을 구한다.
3. 적절한 모양으로 성형한다.



```
class Pooling:
    def __init__(self, pool_h, pool_w, stride=1, pad=0):
        self.pool_h = pool_h
        self.pool_w = pool_w
        self.stride = stride
        self.pad = pad

        self.x = None

    def forward(self, x):
        N, C, H, W = x.shape
```

```

out_h = int(1 + (H - self.pool_h) / self.stride)
out_w = int(1 + (W - self.pool_w) / self.stride)

col = im2col(x, self.pool_h, self.pool_w, self.stride, self.pad)
col = col.reshape(-1, self.pool_h*self.pool_w)

out = np.max(col, axis=1)
out = out.reshape(N, out_h, out_w, C).transpose(0, 3, 1, 2)

self.x = x

return out

def backward(self, dout):
    dout = dout.transpose(0, 2, 3, 1)

    pool_size = self.pool_h * self.pool_w
    dmax = np.zeros((dout.size, pool_size))
    dmax[np.arange(self.argmax.size), self.argmax.flatten()] = dout.flatten()
    dmax = dmax.reshape(dout.shape + (pool_size,))

    dcol = dmax.reshape(dmax.shape[0] * dmax.shape[1] * dmax.shape[2], -1)
    dx = col2im(dcol, self.x.shape, self.pool_h, self.pool_w, self.stride, self.pad)

    return dx

```

## CNN 구현하기

```

class SimpleConvNet:

    def __init__(self, input_dim=(1, 28, 28),
                  conv_param={'filter_num':30, 'filter_size':5, 'pad':0, 'stride':1},
                  hidden_size=100, output_size=10, weight_init_std=0.01):
        filter_num = conv_param['filter_num']
        filter_size = conv_param['filter_size']
        filter_pad = conv_param['pad']
        filter_stride = conv_param['stride']
        input_size = input_dim[1]
        conv_output_size = (input_size - filter_size + 2*filter_pad) / filter_stride + 1
        pool_output_size = int(filter_num * (conv_output_size/2) * (conv_output_size/2))

        self.params = {}
        self.params['W1'] = weight_init_std * \
            np.random.randn(filter_num, input_dim[0], filter_size, filter_size)
        self.params['b1'] = np.zeros(filter_num)
        self.params['W2'] = weight_init_std * \
            np.random.randn(pool_output_size, hidden_size)
        self.params['b2'] = np.zeros(hidden_size)
        self.params['W3'] = weight_init_std * \
            np.random.randn(hidden_size, output_size)
        self.params['b3'] = np.zeros(output_size)

        # 계층 생성
        self.layers = OrderedDict()
        self.layers['Conv1'] = Convolution(self.params['W1'], self.params['b1'],

```

```

        conv_param['stride'], conv_param['pad']))

self.layers['Relu1'] = Relu()
self.layers['Pool1'] = Pooling(pool_h=2, pool_w=2, stride=2)
self.layers['Affine1'] = Affine(self.params['w2'], self.params['b2'])
self.layers['Relu2'] = Relu()
self.layers['Affine2'] = Affine(self.params['w3'], self.params['b3'])

self.last_layer = SoftmaxWithLoss()

def predict(self, x):
    for layer in self.layers.values():
        x = layer.forward(x)

    return x

def loss(self, x, t):

    y = self.predict(x)
    return self.last_layer.forward(y, t)

def accuracy(self, x, t, batch_size=100):
    if t.ndim != 1 : t = np.argmax(t, axis=1)

    acc = 0.0

    for i in range(int(x.shape[0] / batch_size)):
        tx = x[i*batch_size:(i+1)*batch_size]
        tt = t[i*batch_size:(i+1)*batch_size]
        y = self.predict(tx)
        y = np.argmax(y, axis=1)
        acc += np.sum(y == tt)

    return acc / x.shape[0]

def numerical_gradient(self, x, t):
    """기울기를 구한다 (수치미분) .
    Parameters
    -----
    x : 입력 데이터
    t : 정답 레이블
    Returns
    -----
    각 층의 기울기를 담은 사전(dictionary) 변수
        grads['w1']·grads['w2']·... 각 층의 가중치
        grads['b1']·grads['b2']·... 각 층의 편향
    """
    loss_w = lambda w: self.loss(x, t)

    grads = {}
    for idx in (1, 2, 3):
        grads['w' + str(idx)] = numerical_gradient(loss_w, self.params['w' + str(idx)])
        grads['b' + str(idx)] = numerical_gradient(loss_w, self.params['b' + str(idx)])

    return grads

def gradient(self, x, t):
    """기울기를 구한다(오차역전파법).
    Parameters
    -----
    x : 입력 데이터
    t : 정답 레이블
    Returns
    -----
    각 층의 기울기를 담은 사전(dictionary) 변수
        grads['w1']·grads['w2']·... 각 층의 가중치
        grads['b1']·grads['b2']·... 각 층의 편향

```

```

"""
# forward
self.loss(x, t)

# backward
dout = 1
dout = self.last_layer.backward(dout)

layers = list(self.layers.values())
layers.reverse()
for layer in layers:
    dout = layer.backward(dout)

# 결과 저장
grads = {}
grads['w1'], grads['b1'] = self.layers['Conv1'].dw, self.layers['Conv1'].db
grads['w2'], grads['b2'] = self.layers['Affine1'].dw, self.layers['Affine1'].db
grads['w3'], grads['b3'] = self.layers['Affine2'].dw, self.layers['Affine2'].db

return grads

def save_params(self, file_name="params.pkl"):
    params = {}
    for key, val in self.params.items():
        params[key] = val
    with open(file_name, 'wb') as f:
        pickle.dump(params, f)

def load_params(self, file_name="params.pkl"):
    with open(file_name, 'rb') as f:
        params = pickle.load(f)
    for key, val in params.items():
        self.params[key] = val

    for i, key in enumerate(['Conv1', 'Affine1', 'Affine2']):
        self.layers[key].W = self.params['W' + str(i+1)]
        self.layers[key].b = self.params['b' + str(i+1)]

```

## Tensorflow로 구현한 CNN

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d_2 (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_4 (Conv2D)	(None, 13, 13, 64)	18496
max_pooling2d_3 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_5 (Conv2D)	(None, 4, 4, 64)	36928
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 64)	65600
dense_1 (Dense)	(None, 10)	650
Total params: 122,570		
Trainable params: 122,570		
Non-trainable params: 0		

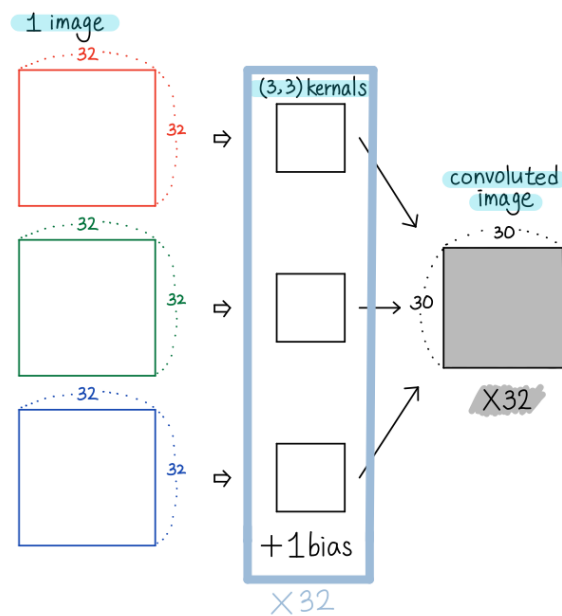
구조 요약 사진

```
tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3))
-> (None, 30, 30, 32), param # = 896
```

Parameter =  $32 * (27 + 1) = 896$

feature map = 32개

No padding 을 선택해서  $30 * 30$ 으로 image가 바뀜



Layer1 참고 사진

```
model.add(layers.MaxPooling2D((2, 2))) → (None, 15, 15, 32)
```

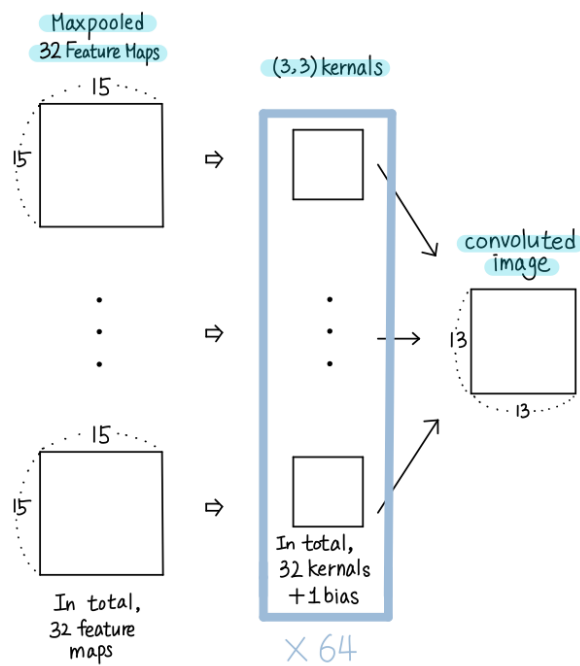
MaxPooling 과정으로 H 와 W 가 각각 절반으로 줄어드는 과정

```
model.add(layers.conv2D(64, (3, 3), activation = 'relu'))  
-> (None, 13, 13, 64), param # 18496
```

Kernel 개수 = 64개

Parameter =  $(9 \times 32 + 1) \times 64 = 18496$ 개

가장자리 부분이 제거되어 13\*13으로 convoluted image가 나옴



Layer 3 그림

```
model.add(layers.MaxPooling2D((2, 2))) → (None, 6, 6, 64)
```

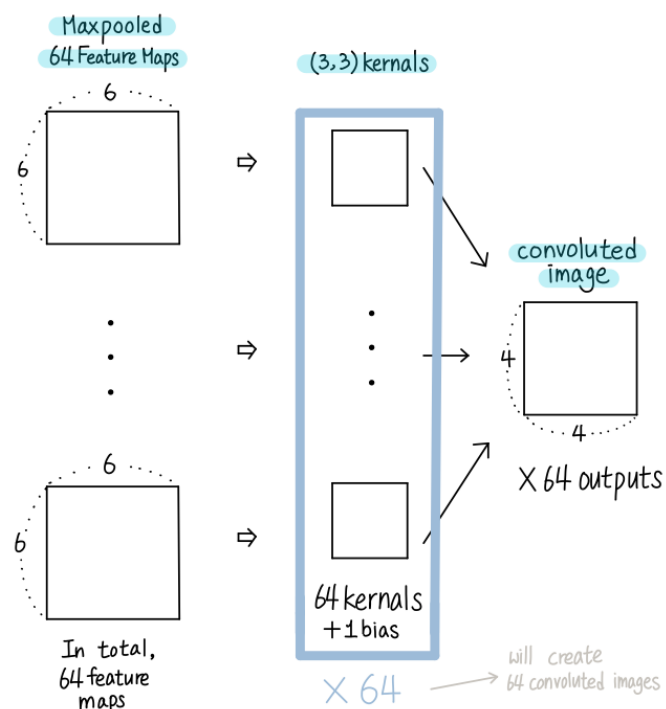
개수를 유지하지만, 크기를 절반으로 삭제하는 과정

```
model.add(layers.Conv2D(64, (3, 3), activation='relu')) → (None, 4, 4, 64) param#36928
```

결과 : 4\*4 픽셀을 갖는 64개의 convoluted image가 나옴

Parameter =  $(3*3*64+1)*64 = 36928$ 개

64개의 3\*3 kernel이 주어짐.



Layer5 참고 사진

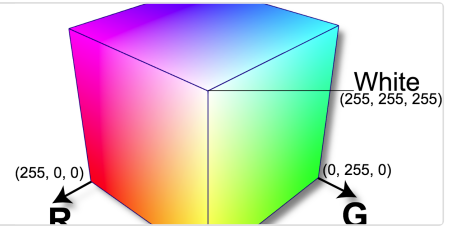
```
model.add(layers.Flatten())  
model.add(layers.Dense(64, activation='relu'))  
model.add(layers.Dense(10, activation='softmax'))
```



## [Tensorflow 2.0] 합성곱 신경망: CNN

이 포스팅에 관해. 이 글은 이미지를 tf.data로 가져오는 이전 포스팅의 다음 단계로, 이미지 데이터에 관한 기본적인 신경망을 이해하는 데 주력합니다. 이번 포스팅에서는 학습 데이터셋과 테스트 데이터셋이 다 있는 CIFAR images를 사용하겠습니다.

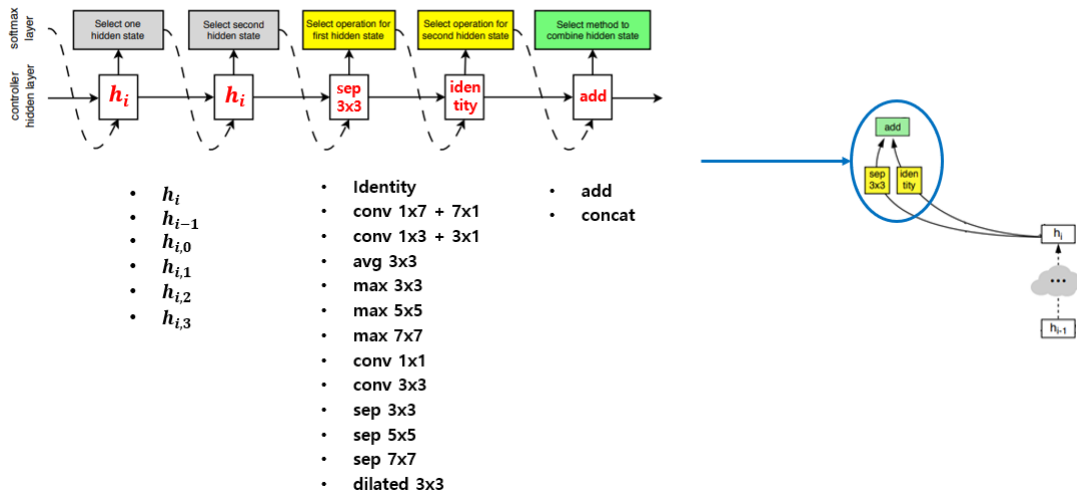
<https://ericabae.medium.com/tensorflow-2-0-%ED%95%A9%EC%84%B1%EA%B3%B1-%EC%8B%A0%EA%B2%BD%EB%A7%9D-cnn-bfd925298c9b>



## 최근 알고리즘 동향

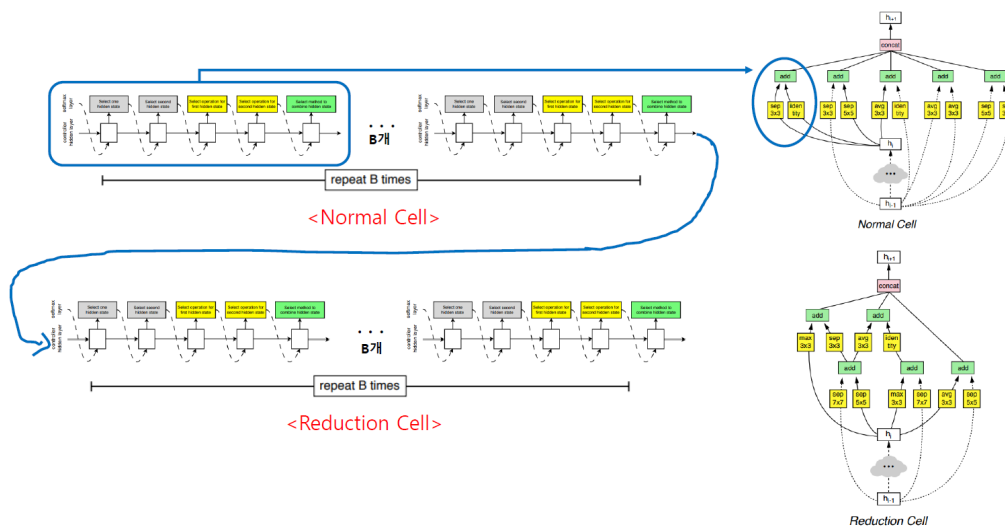
### NASNet

- 이전 모델 : Nas
- 본 내용은 CIFAR-10 데이터셋을 사용한 논문의 내용을 설명
- Block
  1. 2개의 연산을 수행하여 하나의 feature map을 출력
  2. 5개의 Rnn controller가 하나의 block을 결정
  3. 2번 과정에서 hidden state input, operation, combine operation 이 사용됨



NasNet block 생성 표

- Convolution Cell
  1. Normal cell, reduction cell 이 존재함
  2. Reduction cell 이 입력 feature map의 크기의 절반 cell 의미
  3. cell 들은 block으로 구성되어 있다.
  4. 두 개의 cell 차이는 block 연산 시의 stride




Convolutional Cell 관련 참고 자료

나스넷은 객체 감지 및 이미지 분류와 같은 다른 응용 프로그램에서 유용하게 사용될 수 있다. 보강 학습을 사용해 AI 모델을 자동화할 수 있다. 오토ML은 특정 작업에 대해 하위 AI 네트워크를 개발하는 컨트롤러 신경 네트워크 역할을 한다.

나스넷의 임무는 인물, 신호등, 자동차, 배낭, 핸드백 등을 포함한 물체를 실시간 비디오를 통해 인식하는 것이다.


<https://github.com/titu1994/Keras-NASNet>

NasNet 코드 github

 <https://arxiv.org/pdf/1611.01578.pdf>

NasNet 논문

## 참고

Learning Transferable Architectures for Scalable Image Recognition 리뷰  
안녕하세요, 오늘은 최근 주목받고 있는 AutoML 관련 논문을 리뷰하려고 합니다. 논문 제목은 "Learning Transferable Architectures for Scalable Image Recognition"이며 올해, 2018년 CVPR에 발표된 논문입니다. 이 논문을 소개하기에 앞서 AutoML에 대해 간략하게  
 <https://hoya012.github.io/blog/Learning-Transferable-Architectures-for-Scalable-Image-Recognition-Review/>

**hoya012**  
**Deep Learning**  
**Blog!**

## 정리

1. CNN 은 완전연결 계층 네트워크에 합성곱 계층과 풀링 계층을 새로 추가한다.
2. 합성곱 계층과 풀링 계층은 `im2col` 함수를 이용하면 더 간단하고 효율적으로 구현할 수 있다.
3. CNN을 시각화해보면 계층이 깊어질수록 고급 정보가 추출되는 모습을 확인할 수 있다.