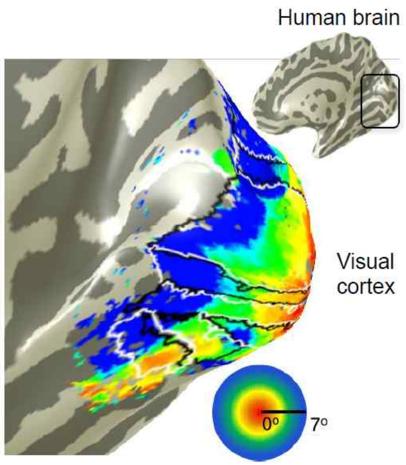
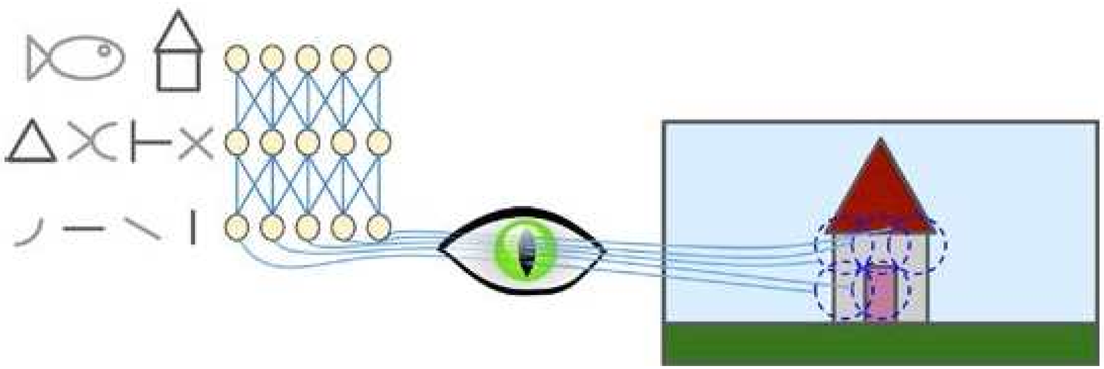
Convolution Neural Network

Part 1

시각 피질 (Visual Cortex)

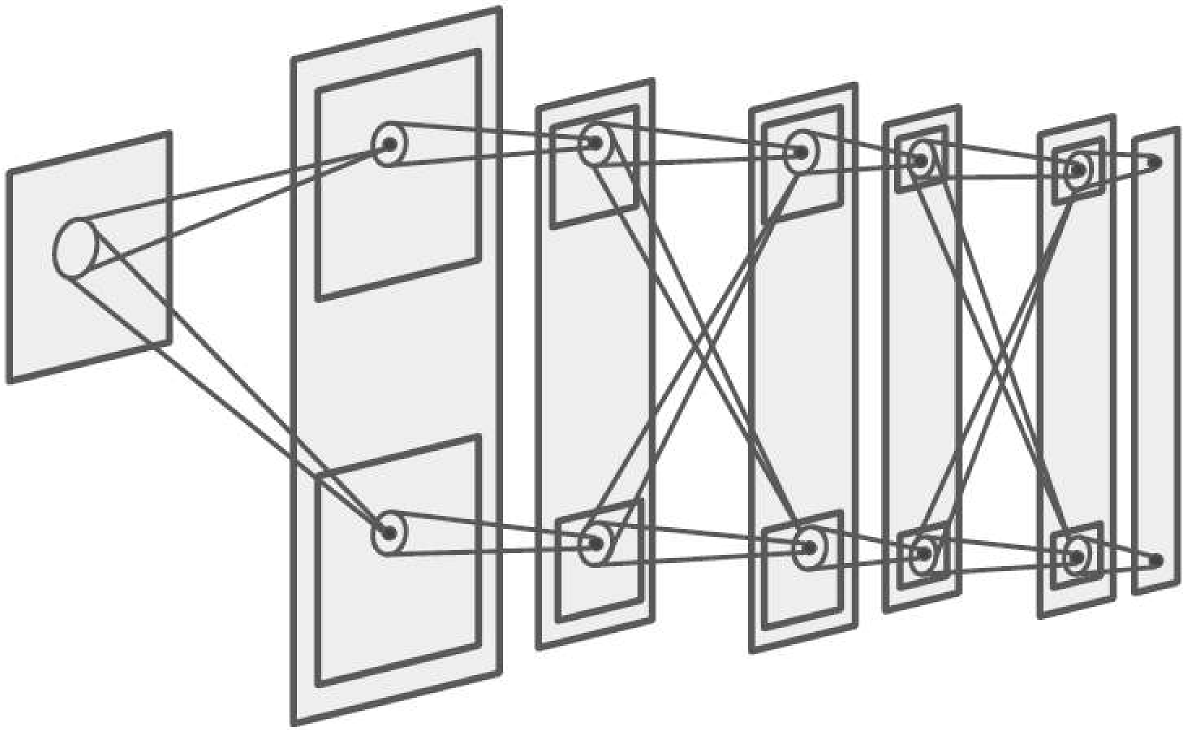
- Local receptive field
  - D.H.Hubel, T.Wiesel (1958)
  - 시각 피질 안의 많은 뉴런이 작은 국부 수용장 (local receptive field) 을 가진다는 것을 발견
    - 뉴런의 수용장들은 서로 겹칠 수 있어서, 합치면 전체 시야를 감싸게 됨
    - 어떤 뉴런은 수평선의 이미지에만 반응하고 반면 다른 뉴런은 다른 각도의 선분에 반응
    - 어떤 뉴런은 큰 수용장을 가져서 저수준 패턴이 조합된 더 복잡한 패턴에 반응
      ⇒ 고수준 뉴런이 이웃한 저수준 뉴런의 출력에 기반한다는 아이디어가 도출





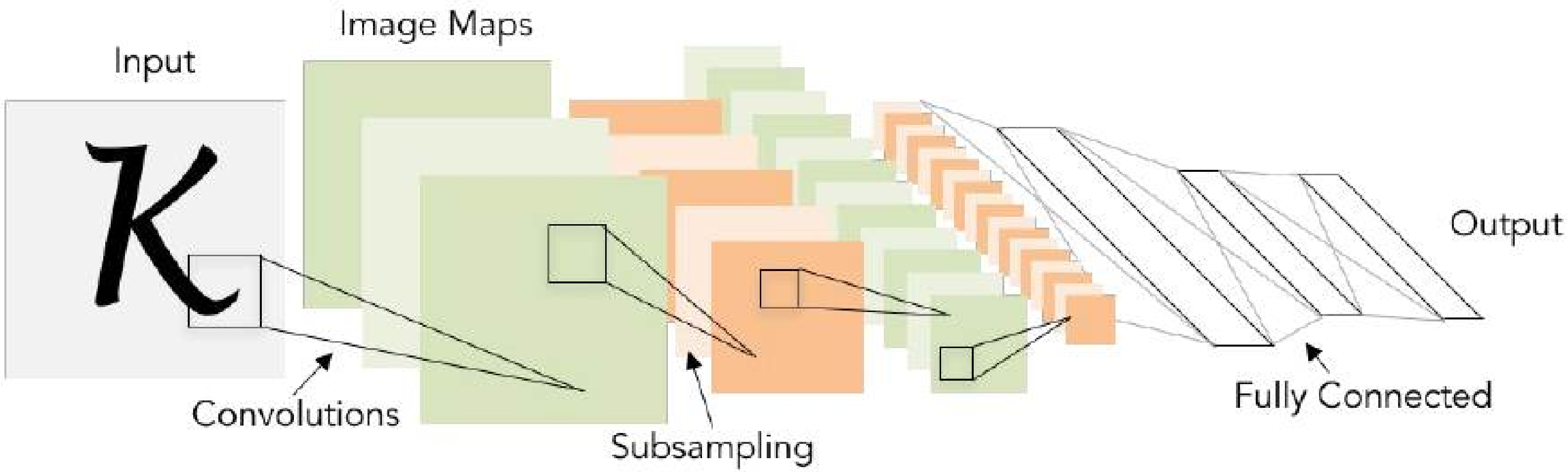
시각 피질 (Visual Cortex)

- Neocognition
  - K.Fukushima (1980)
  - 패턴인식을 위한 neural network model



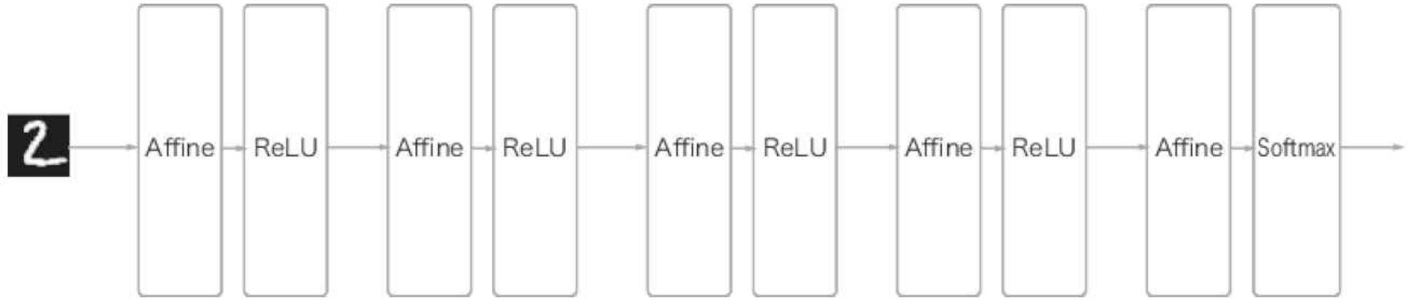
시각 피질 (Visual Cortex)

- LeNet-5
  - LeCun et. al. (1998)
  - 신경망의 문서인식 응용
  - Convolution layer / Pooling layer 등장



Structure

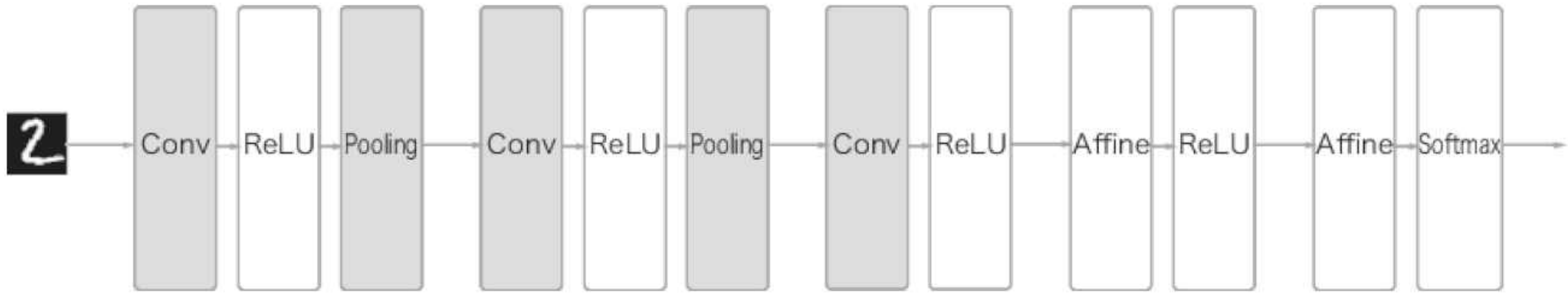
- Deep neural network (fully connected network)



영상 데이터 -> 1차원 벡터 : 입력 영상의 '형상' 무시

(ex) 32 x 32 x 3 image -> 3072 x 1 vector

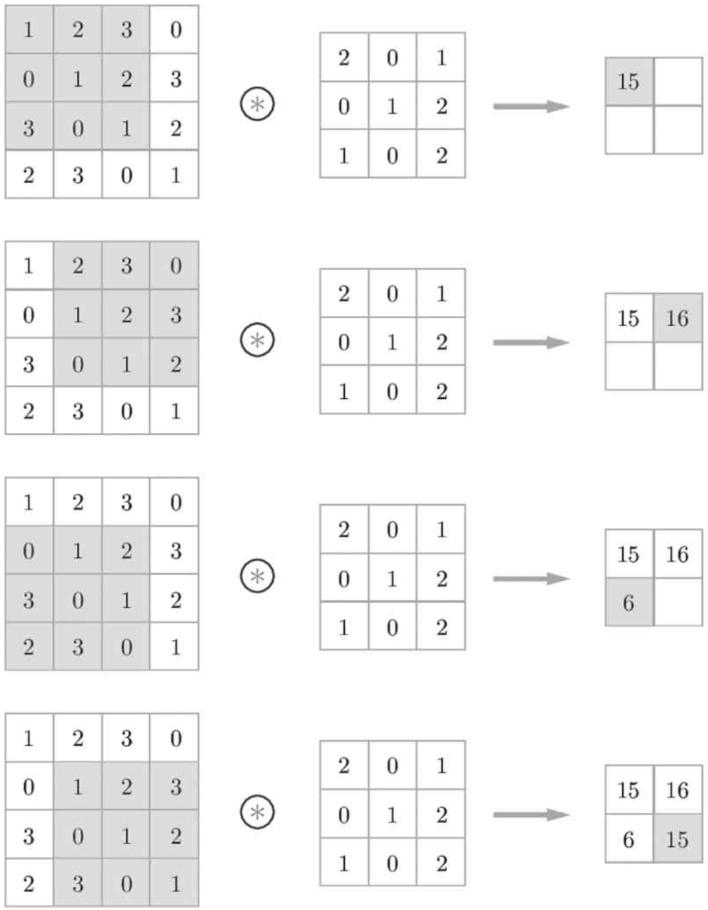
- Convolution neural network

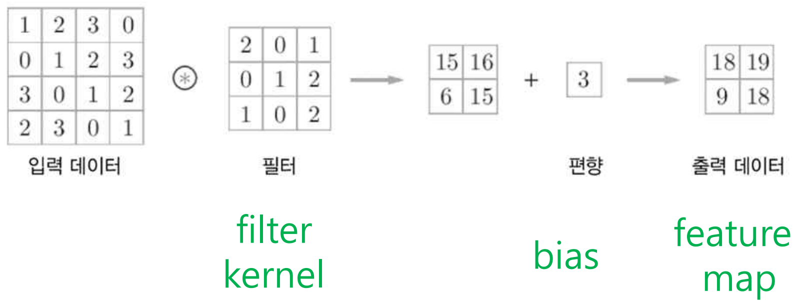


영상 데이터 -> 3차원 벡터 (가로,세로,채널) : 입력 영상의 '형상' 유지

Convolution Layer

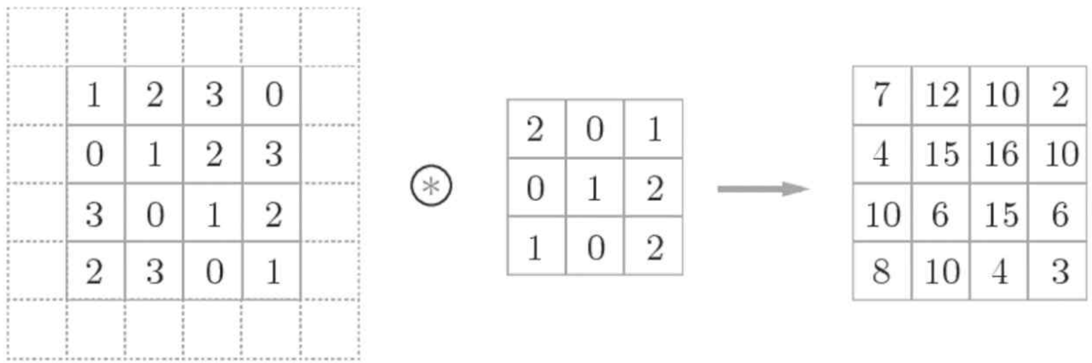
- Convolution 연산
  - 이웃 pixel 들의 공간적 특성값 (spatial feature)





Convolution Layer

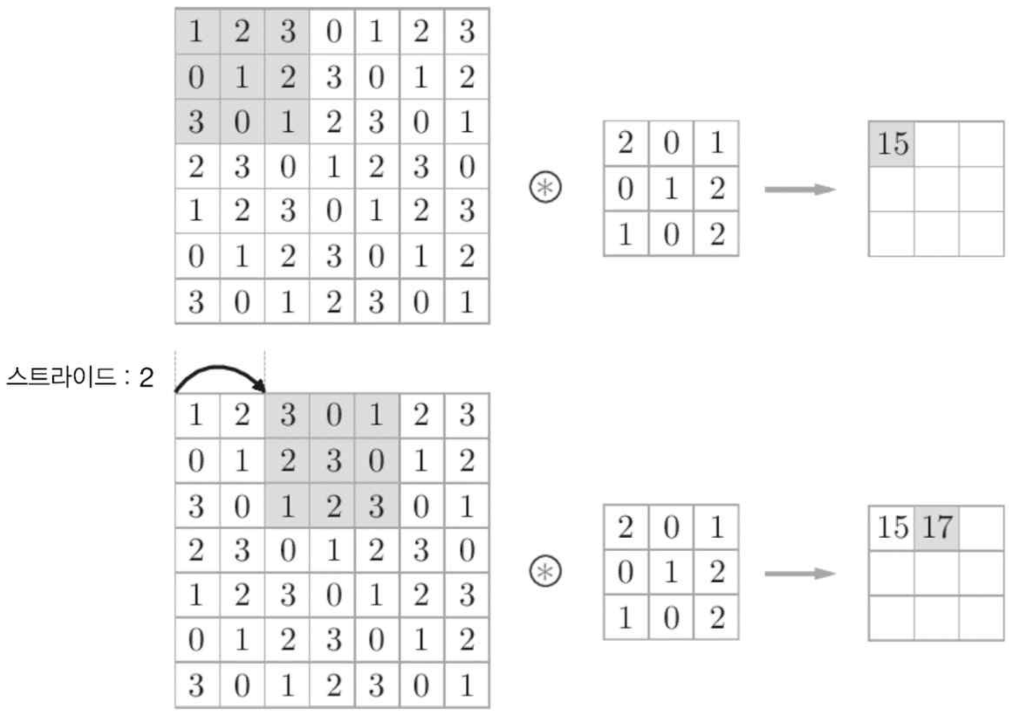
- Padding
  - 영상 모서리의 convolution 연산을 위하여, 입력 데이터 주변을 특정값으로 채움



(4, 4) (3, 3) (4, 4)

Convolution Layer

- Stride
  - Filter 를 적용하는 위치 간격



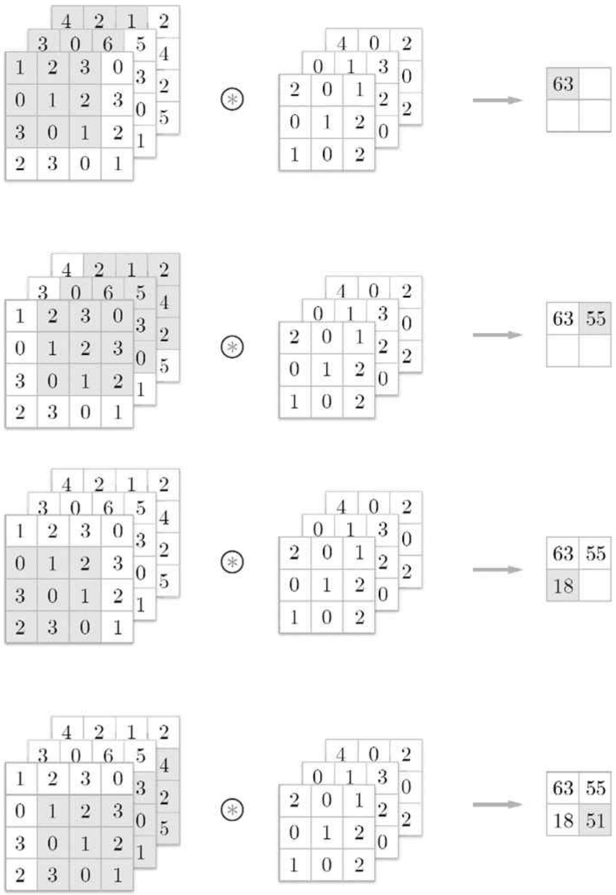
$$OH = \frac{H + 2P - FH}{S} + 1$$

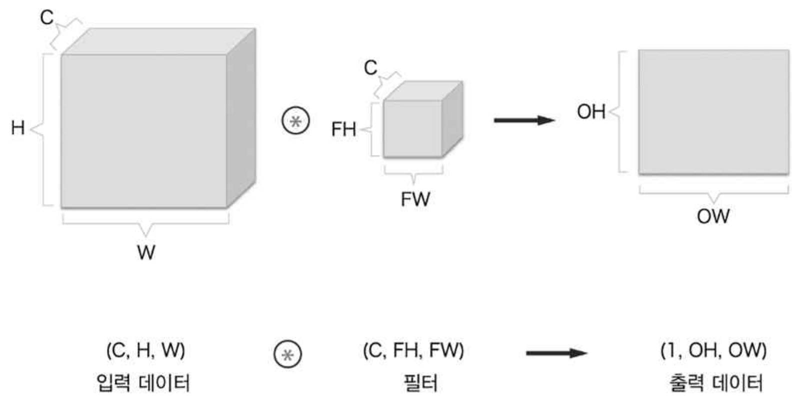
$$OW = \frac{W + 2P - FW}{S} + 1$$

입력크기: (H,W), Filter 크기: (FH, FW), 출력크기: (OH, OW)
Padding: P, Stride: S

Convolution Layer

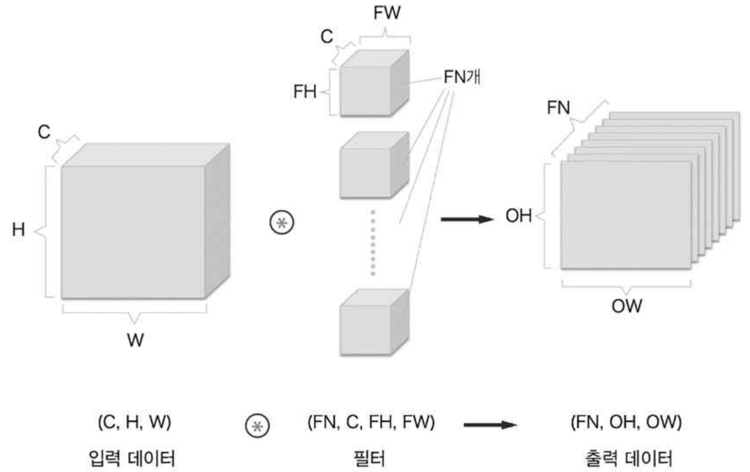
- Color image convolution
  - Convolution for 3 channels (R,G,B)
  - 3D data convolution





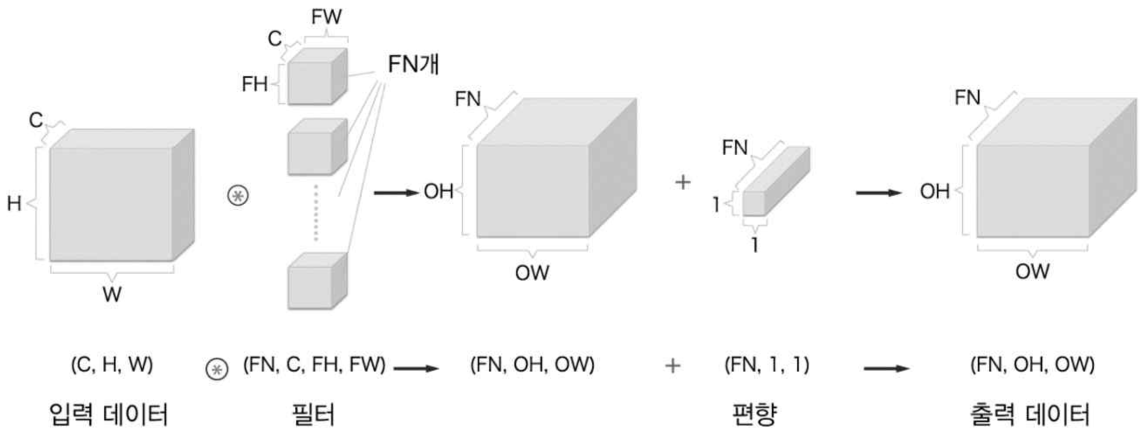
Convolution Layer

- 여러 개의 filter 사용



FN: 필터 갯수

- Bias 추가



Convolution Layer

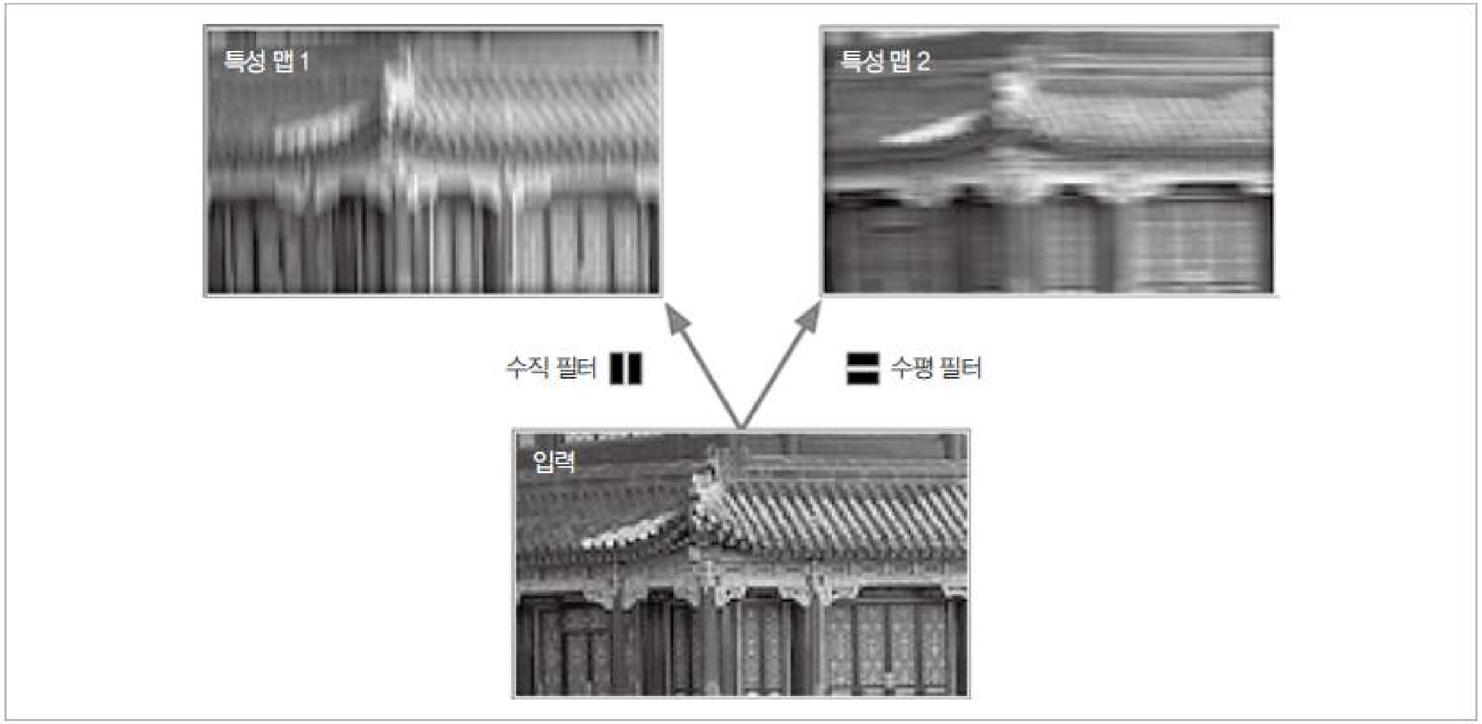
- Hyperparameters
  - Number of filters **K**
  - The filter size **F**
  - The stride **S**
  - The zero padding **P**

- 학습 parameter

  F^2CK and K biases (C: number of channels)

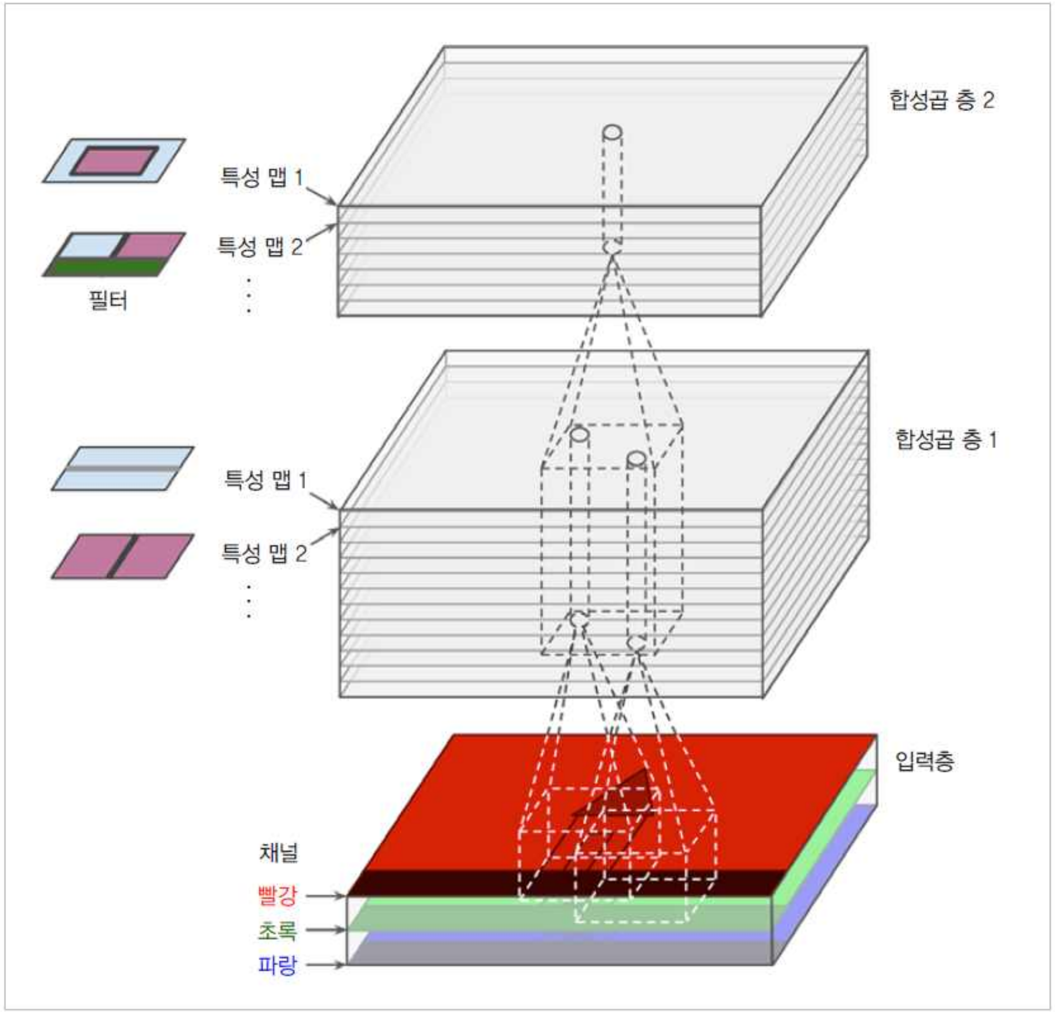
Convolution Layer

- Filter (kernel) & feature map



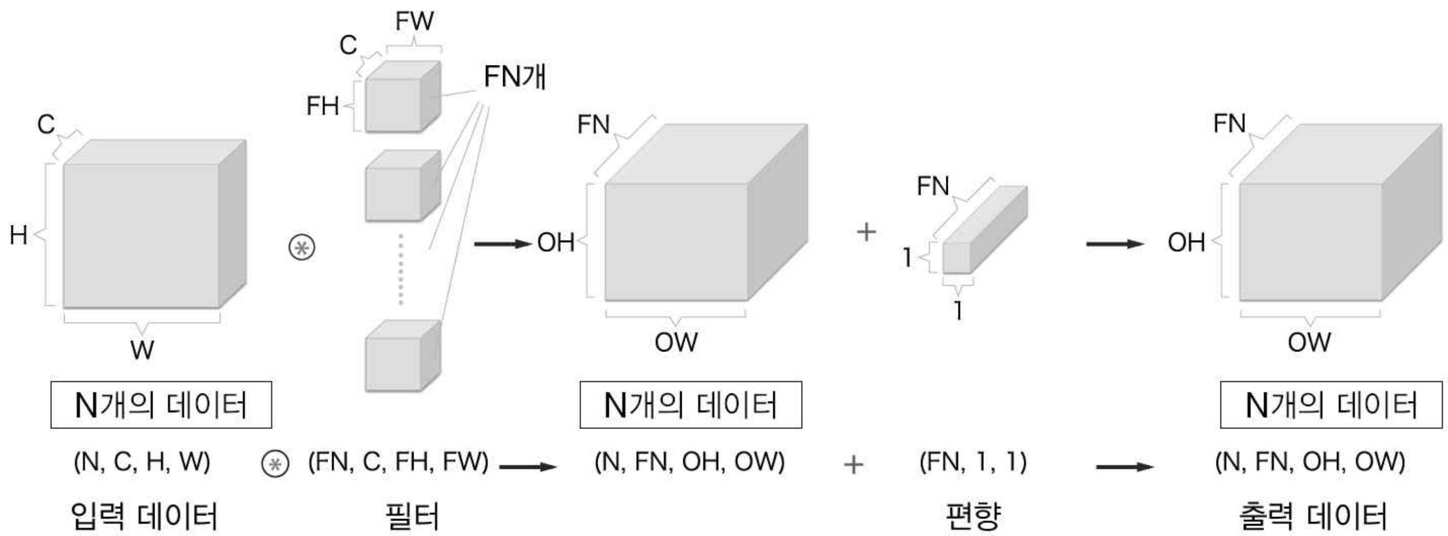
Convolution Layer

- Filter & Feature map



Convolution Layer

- Batch 처리



N: batch 데이터 수

- 4차원 데이터 (Tensor) 의 흐름
- 연산내용: 행렬 곱셈 & 덧셈
- Convolution 연산으로 local feature 를 추출하여 feature map 출력

Convolution Layer

- Implementation

```
class Convolution:
    def __init__(self, W, b, stride=1, pad=0):
        self.W = W
        self.b = b
        self.stride = stride
        self.pad = pad

        # 중간 데이터 ( backward 시 사용 )
        self.x = None
        self.col = None
        self.col_W = None

        # 가중치와 편향 매개변수의 기울기
        self.dW = None
        self.db = None

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

        col = im2col(x, FH, FW, self.stride, self.pad)
        col_W = self.W.reshape(FN, -1).T

        out = np.dot(col, col_W) + self.b
        out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)

        self.x = x
        self.col = col
        self.col_W = col_W

        return out
```

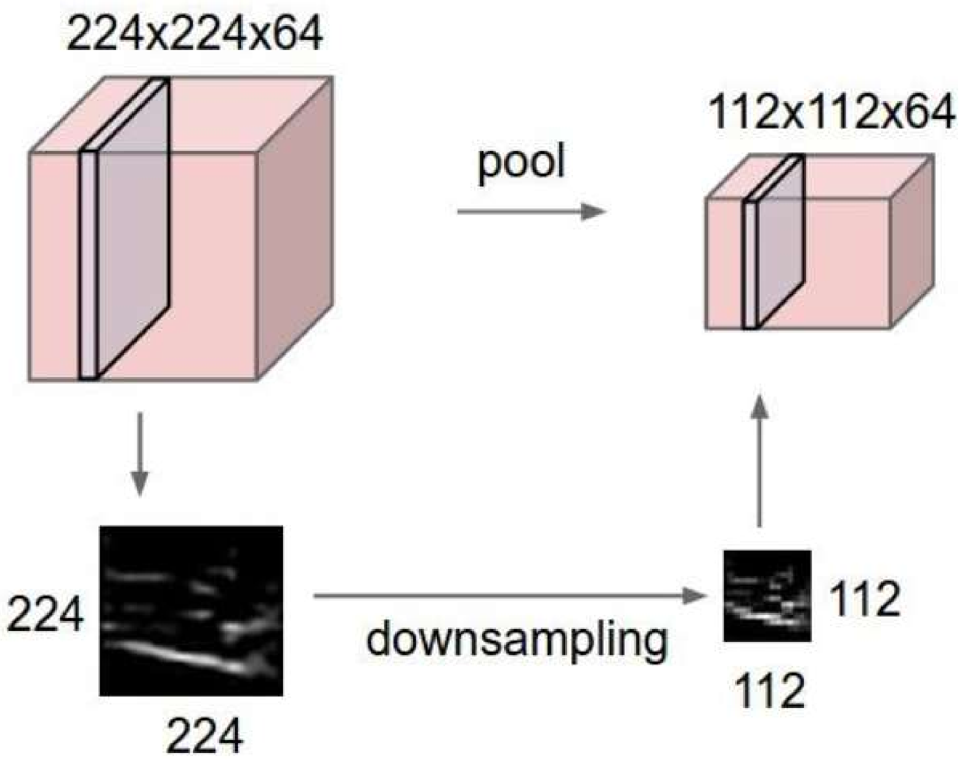
```
    def backward(self, dout):
        FN, C, FH, FW = self.W.shape
        dout = dout.transpose(0,2,3,1).reshape(-1, FN)

        self.db = np.sum(dout, axis=0)
        self.dW = np.dot(self.col.T, dout)
        self.dW = self.dW.transpose(1, 0).reshape(FN, C, FH, FW)

        dcol = np.dot(dout, self.col_W.T)
        dx = col2im(dcol, self.x.shape, FH, FW, self.stride, self.pad)

        return dx
```

프로그램소스
https://github.com/WegraLee/deep-learning-from-scratch

Pooling Layer

- 목적
  - 계산량과 메모리 사용량 감소
  - 학습 파라메터 수 감소

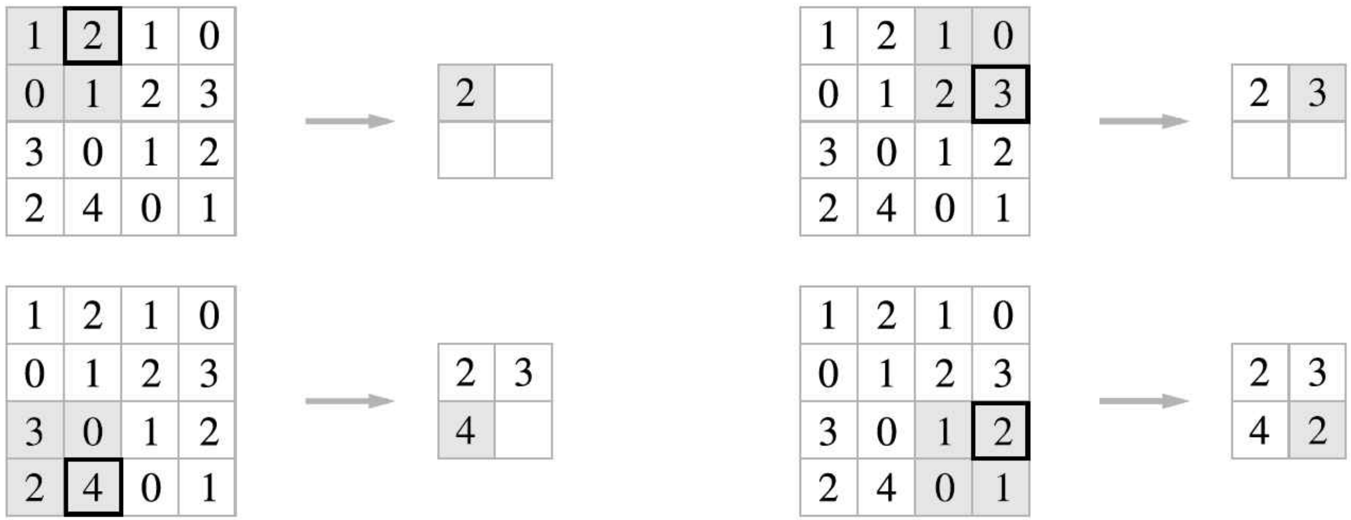
  ☞ 과적합 문제 개선



Pooling Layer

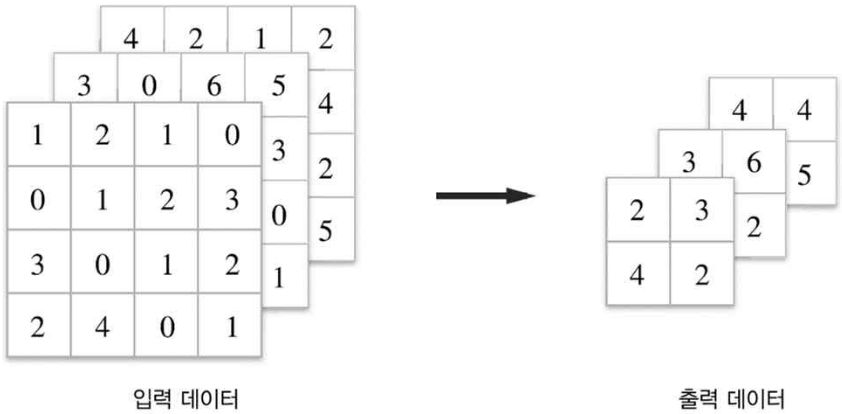
- Pooling layer
  - 가로 세로 방향의 공간을 줄이는 연산
  - 종류: Max pooling, average pooling 등
  - 학습 대상이 되는 parameter 가 없음

Max-pooling

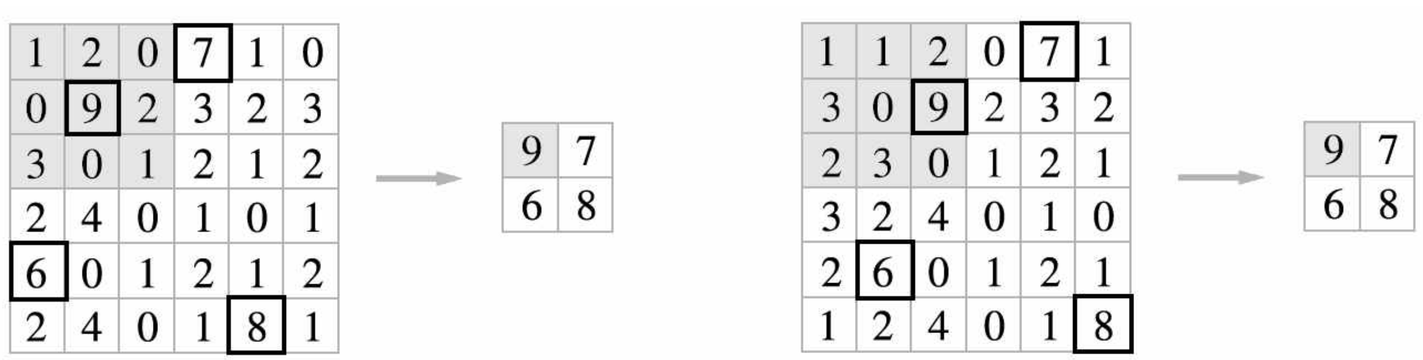


Pooling Layer

- 채널 수 유지



- 입력의 변화에 강인



Pooling Layer

- Hyperparameter
  - Spatial extent **F**
  - Stride **S**

- 학습 parameter
  - none

Pooling Layer

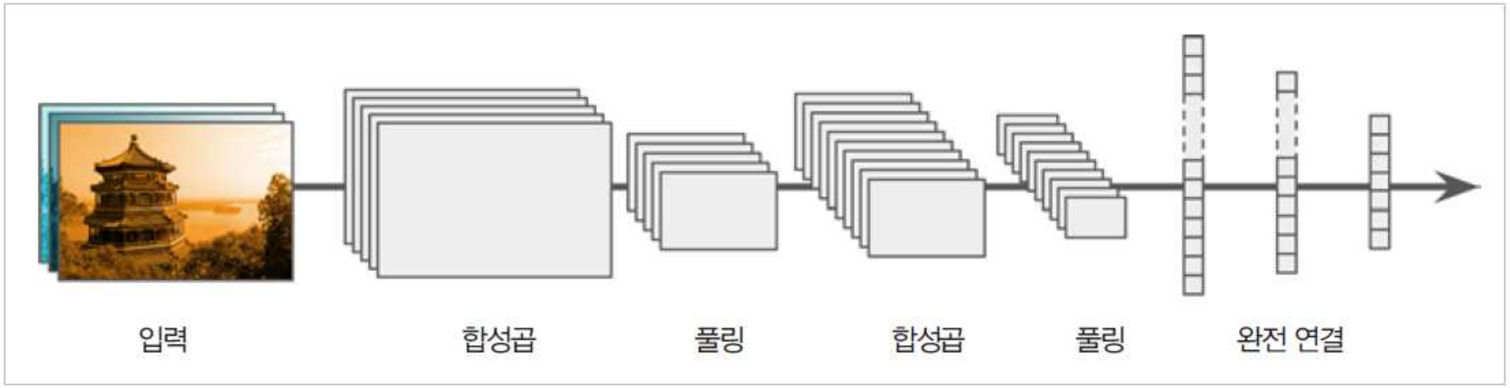
- Implementation

```
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
        self.arg_max = 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)

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

        self.x = x
        self.arg_max = arg_max

        return out
```

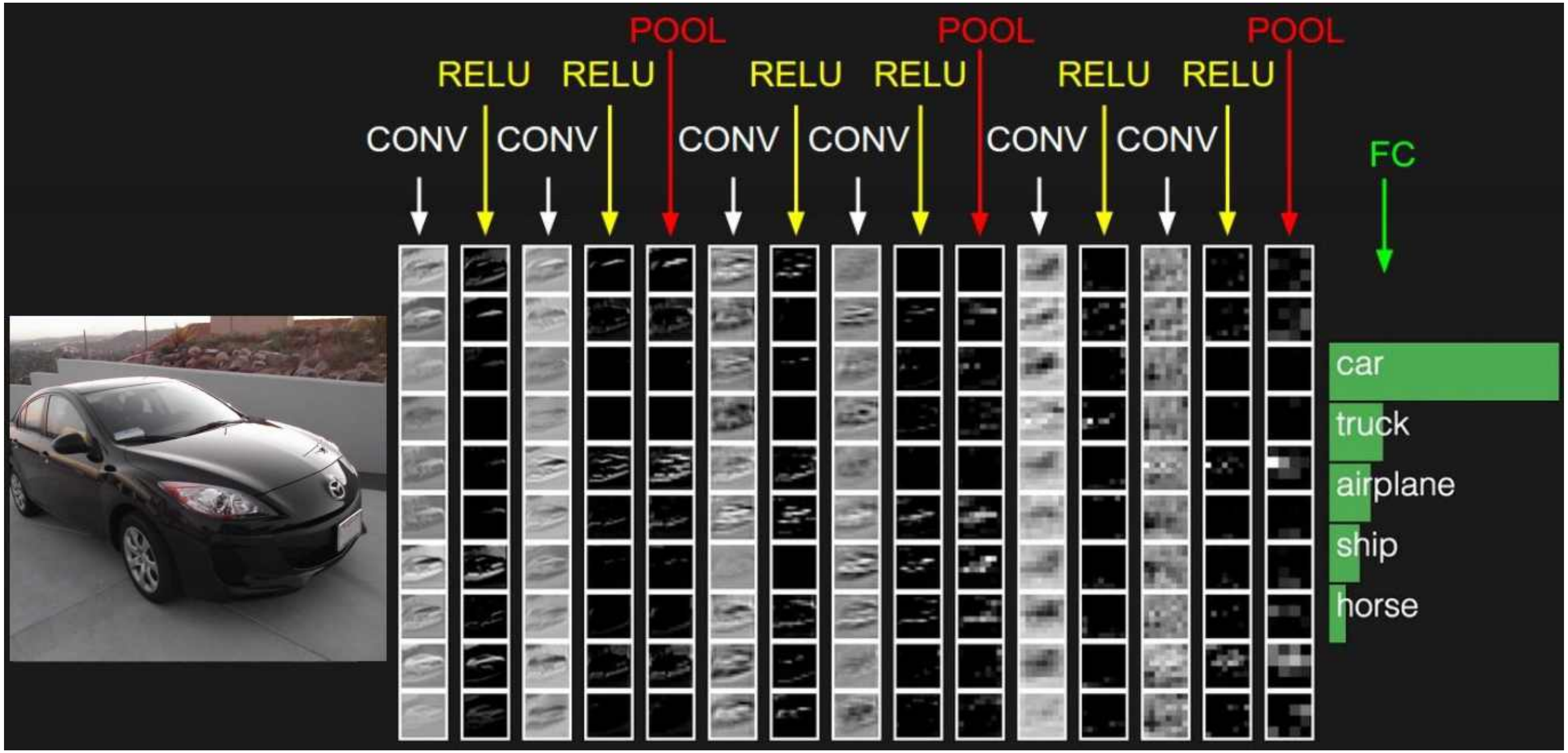
```
def backward(self, dout):
        dout = dout.transpose(0, 2, 3, 1)

        pool_size = self.pool_h * self.pool_w
        dmax = np.zeros((dout.size, pool_size))
        dmax[np.arange(self.arg_max.size), self.arg_max.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
```

기본 CNN 구조

- Typical CNN
  - Convolution layer + Pooling layer + Fully connected layer



기본 CNN 구조



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

CNN 구현 - SimpleConvNet

```
class SimpleConvNet:
    """단순한 합성곱 신경망
     conv - relu - pool - affine - relu - affine - softmax

    Parameters
    ----------
    input_size : 입력 크기 (MNIST의 경우엔 784)
    hidden_size_list : 각 은닉층의 뉴런 수를 담은 리스트 (e.g. [100, 100, 100])
    output_size : 출력 크기 (MNIST의 경우엔 10)
    activation : 활성화 함수 - 'relu' 혹은 'sigmoid'
    weight_init_std : 가중치의 표준편차 지정 (e.g. 0.01)
        'relu'나 'he'로 지정하면 'He 초깃값'으로 설정
        'sigmoid'나 'xavier'로 지정하면 'Xavier 초깃값'으로 설정
    """
    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))

        # 가중치 초기화
        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)
```

```python
        # 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):
        """손실 함수를 구한다.

        Parameters
        ----------
        x : 입력 데이터
        t : 정답 레이블
        """
        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]
```

```python
    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)]
```

CNN 구현 – MNIST 학습

```
# coding: utf-8
import sys, os
sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
import matplotlib.pyplot as plt
from dataset.mnist import load_mnist
from simple_convnet import SimpleConvNet
from common.trainer import Trainer

# 데이터 읽기
(x_train, t_train), (x_test, t_test) = load_mnist(flatten=False)

# 시간이 오래 걸릴 경우 데이터를 줄인다.
#x_train, t_train = x_train[:5000], t_train[:5000]
#x_test, t_test = x_test[:1000], t_test[:1000]

max_epochs = 20

network = SimpleConvNet(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)

trainer = Trainer(network, x_train, t_train, x_test, t_test,
                  epochs=max_epochs, mini_batch_size=100,
                  optimizer='Adam', optimizer_param={'lr': 0.001},
                  evaluate_sample_num_per_epoch=1000)
trainer.train()

# 매개변수 보존
network.save_params("params.pkl")
print("Saved Network Parameters!")
```

```python
# 그래프 그리기
markers = {'train': 'o', 'test': 's'}
x = np.arange(max_epochs)
plt.plot(x, trainer.train_acc_list, marker='o', label='train', markevery=2)
plt.plot(x, trainer.test_acc_list, marker='s', label='test', markevery=2)
plt.xlabel("epochs")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
plt.legend(loc='lower right')
plt.show()
```

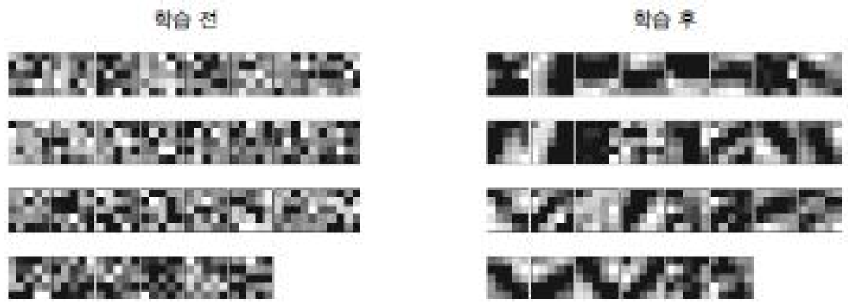
CNN 구현 – Visualize Filter

```
# coding: utf-8
import numpy as np
import matplotlib.pyplot as plt
from simple_convnet import SimpleConvNet

def filter_show(filters, nx=8, margin=3, scale=10):
    """
    c.f. https://gist.github.com/aidiary/07d530d5e08011832b12#file-draw_weight-py
    """
    FN, C, FH, FW = filters.shape
    ny = int(np.ceil(FN / nx))

    fig = plt.figure()
    fig.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=0.05, wspace=0.05)

    for i in range(FN):
        ax = fig.add_subplot(ny, nx, i+1, xticks=[], yticks=[])
        ax.imshow(filters[i, 0], cmap=plt.cm.gray_r, interpolation='nearest')
    plt.show()


network = SimpleConvNet()
# 무작위(랜덤) 초기화 후의 가중치
filter_show(network.params['W1'])

# 학습된 가중치
network.load_params("params.pkl")
filter_show(network.params['W1'])
```

프로그램소스: https://github.com/WegraLee/deep-learning-from-scratch

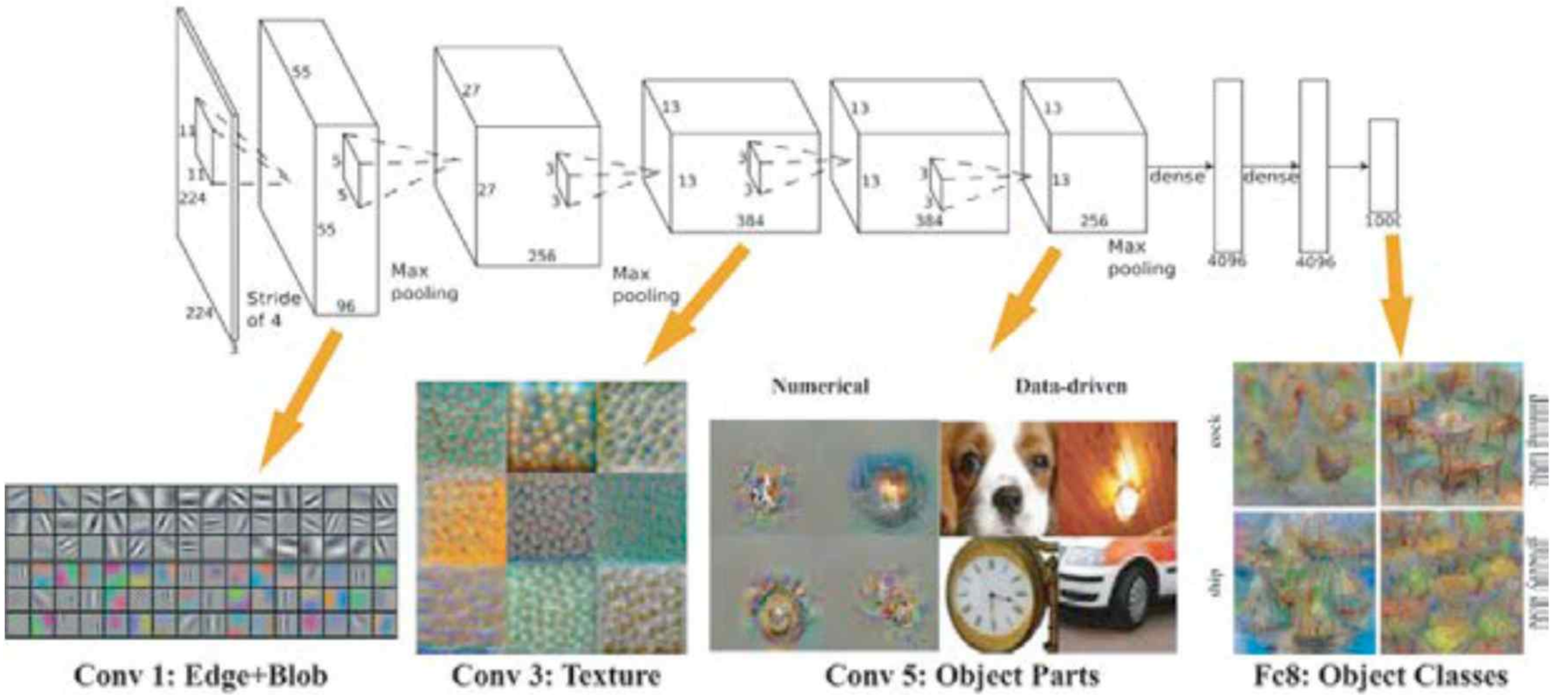
CNN 구현 – Visualize Filter

- Parameter (Weight)
  - 학습 전: random
  - 학습 후: 규칙 성이 생김
    - 가로/세로 에지 검출 등



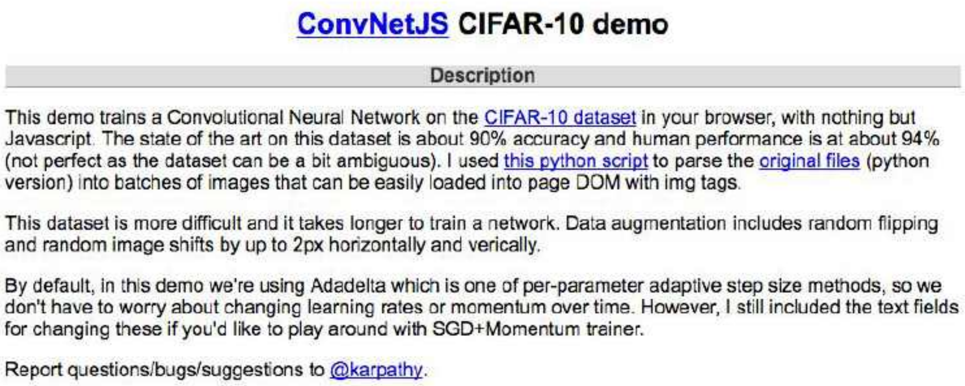
CNN 구현 – Visualize Filter

- Layer 깊이에 따른 정보 변화



1번째 층은 에지와 블롭, 3번째 층은 텍스처, 5번째 층은 사물의 일부, 마지막 완전연결 계층은 사물의 클래스(개, 자동차 등)에 뉴런이 반응

CNN Demo


ConvNetJS CIFAR-10 demo

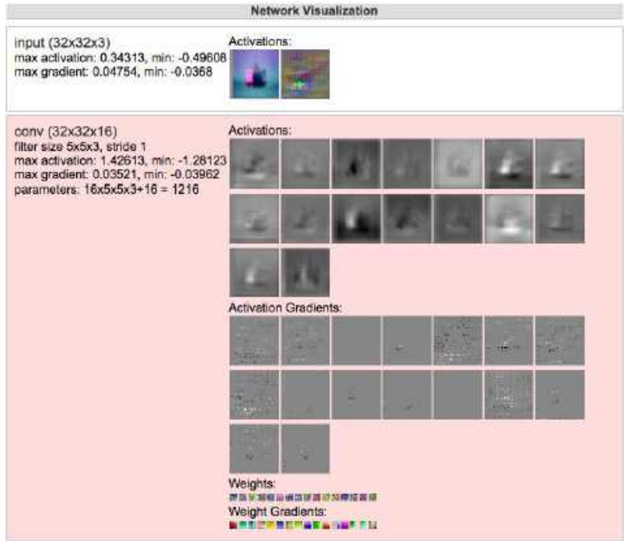
Description

This demo trains a Convolutional Neural Network on the CIFAR-10 dataset in your browser, with nothing but Javascript. The state of the art on this dataset is about 90% accuracy and human performance is at about 94% (not perfect as the dataset can be a bit ambiguous). I used this python script to parse the original files (python version) into batches of images that can be easily loaded into page DOM with img tags.

This dataset is more difficult and it takes longer to train a network. Data augmentation includes random flipping and random image shifts by up to 2px horizontally and verically.

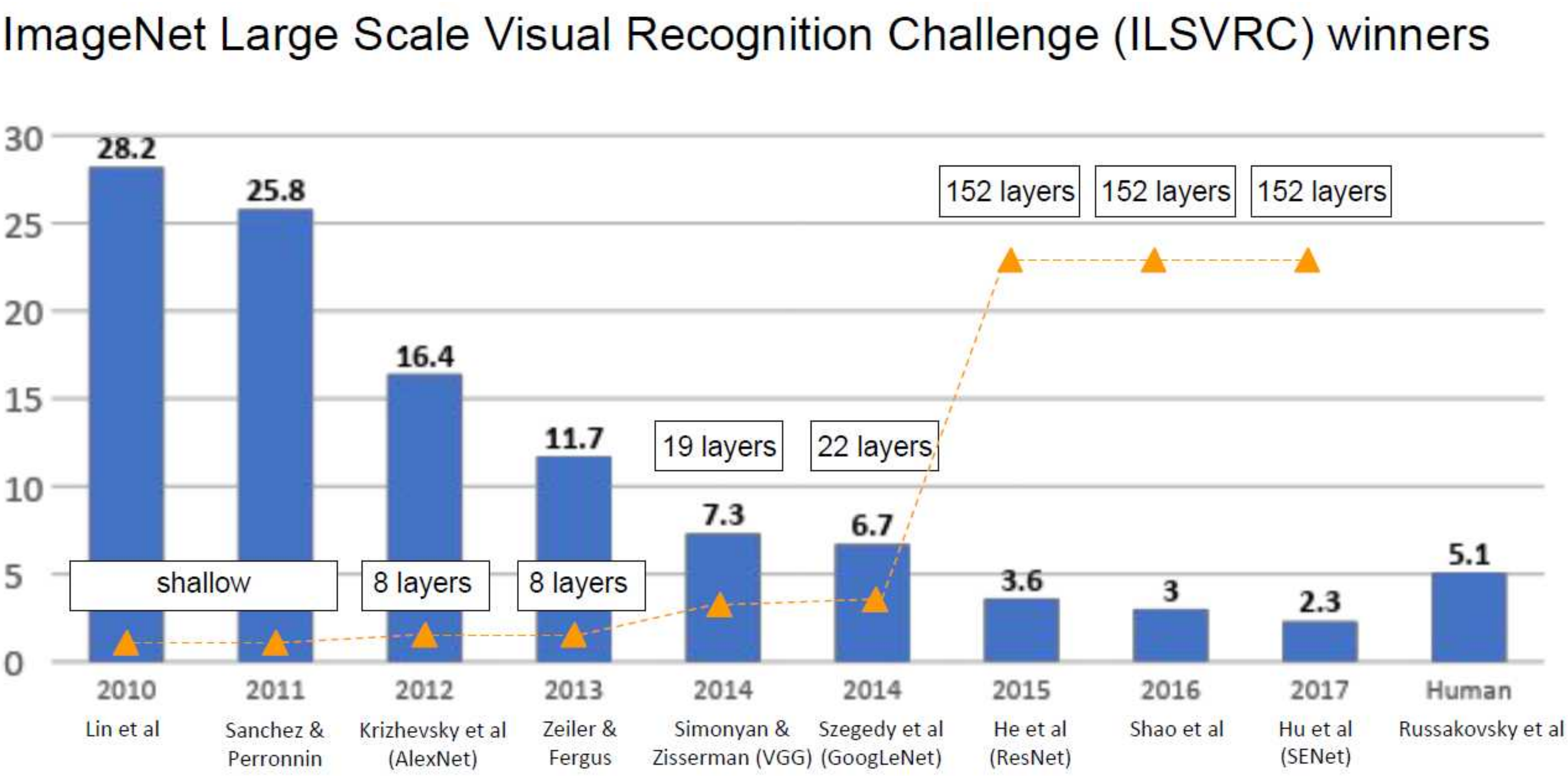
By default, in this demo we're using Adadelta which is one of per-parameter adaptive step size methods, so we don't have to worry about changing learning rates or momentum over time. However, I still included the text fields for changing these if you'd like to play around with SGD+Momentum trainer.

Report questions/bugs/suggestions to @karpathy.



http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

CNN 구조


ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners
30
25
20
15
10
5
0
28.2
25.8
16.4
11.7
7.3
6.7
3.6
3
2.3
5.1
shallow
8 layers
8 layers
19 layers
22 layers
152 layers
152 layers
152 layers
2010
Lin et al
2011
Sanchez & Perronnin
2012
Krizhevsky et al (AlexNet)
2013
Zeiler & Fergus
2014
Simonyan & Zisserman (VGG)
2014
Szegedy et al (GoogLeNet)
2015
He et al (ResNet)
2016
Shao et al
2017
Hu et al (SENet)
Human
Russakovsky et al

CNN 구조

- LeNet-5
  - Yann LeCun (1988)
  - MNIST 문제 적용

층	종류	특성 맵	크기	커널 크기	스트라이드	활성화 함수
출력	완전 연결	–	10	–	–	RBF
F6	완전 연결	–	84	–	–	tanh
C5	합성곱	120	1×1	5×5	1	tanh
S4	평균 풀링	16	5×5	2×2	2	tanh
C3	합성곱	16	10×10	5×5	1	tanh
S2	평균 풀링	6	14×14	2×2	2	tanh
C1	합성곱	6	28×28	5×5	1	tanh
입력	입력	1	32×32	–	–	–

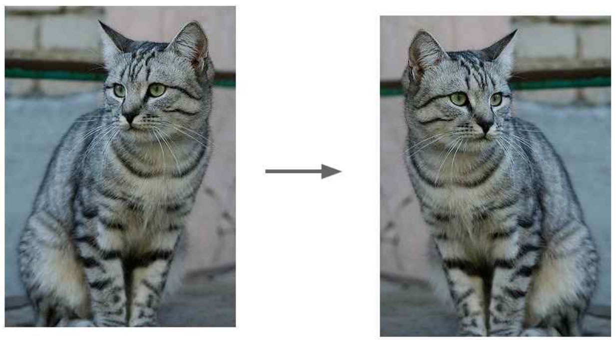
CNN 구조

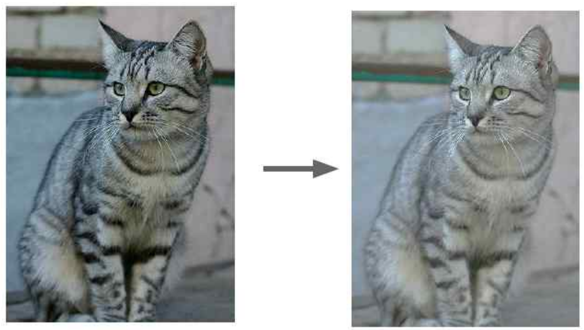
- AlexNet
  - Alex Krizhevsky (2012)
  - ImageNet 대회 우승 (error rate: 17%)
  - 특징
    - 연속적 Conv. Layer
    - DropOut 적용:50%
    - Data augmentation 기법 적용

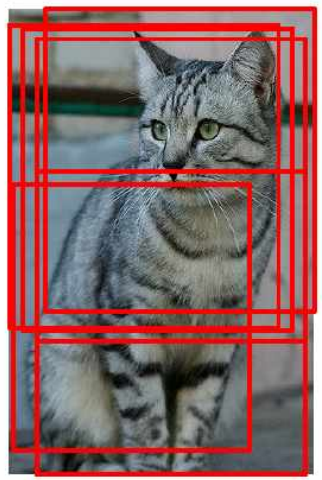
층	종류	특성 맵	크기	커널 크기	스트라이드	패딩	활성화 함수
출력	완전 연결	–	1,000	–	–	–	Softmax
F10	완전 연결	–	4,096	–	–	–	ReLU
F9	완전 연결	–	4,096	–	–	–	ReLU
F8	최대 풀링	256	6×6	3×3	2	valid	–
C7	합성곱	256	13×13	3×3	1	same	ReLU
C6	합성곱	384	13×13	3×3	1	same	ReLU
C5	합성곱	384	13×13	3×3	1	same	ReLU
S4	최대 풀링	256	13×13	3×3	2	valid	–
C3	합성곱	256	27×27	5×5	1	same	ReLU
S2	최대 풀링	96	27×27	3×3	2	valid	–
C1	합성곱	96	55×55	11×11	4	valid	ReLU
입력	입력	3 (RGB)	227×227	–	–	–	–

CNN 구조

- Data Augmentation
  - 테스트 성능을 높이기 위하여 학습 데이터 증강
    - Horizontal flips
    - Contrast & brightness
    - Random crops /Scales (ResNet)

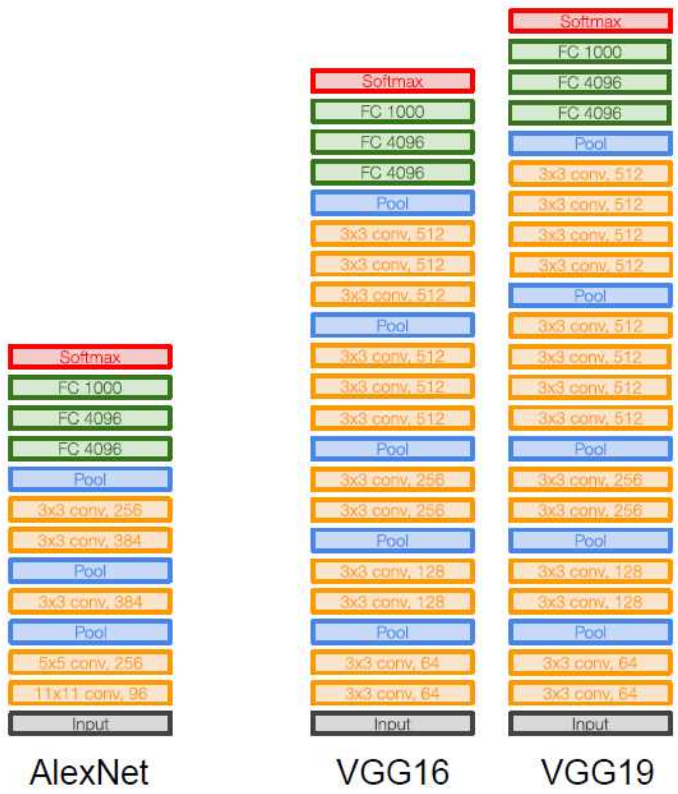






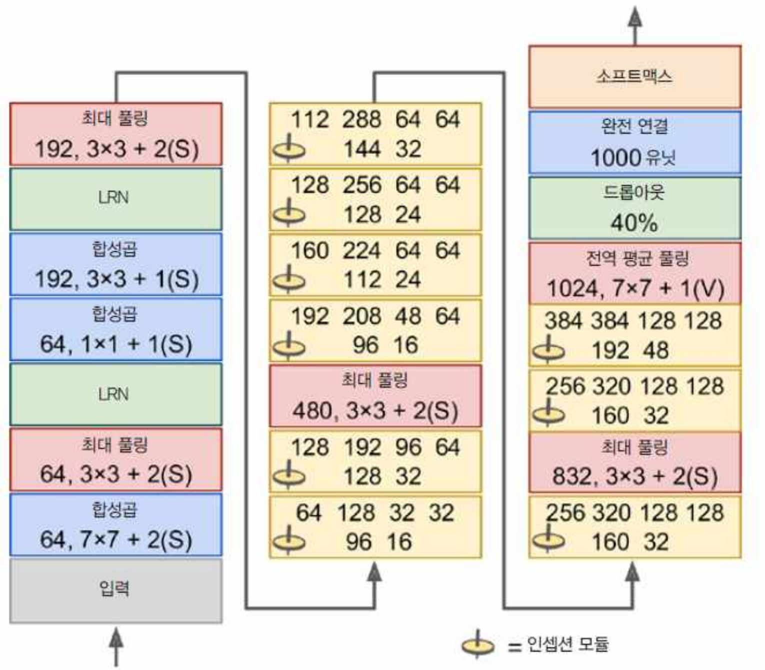
CNN 구조

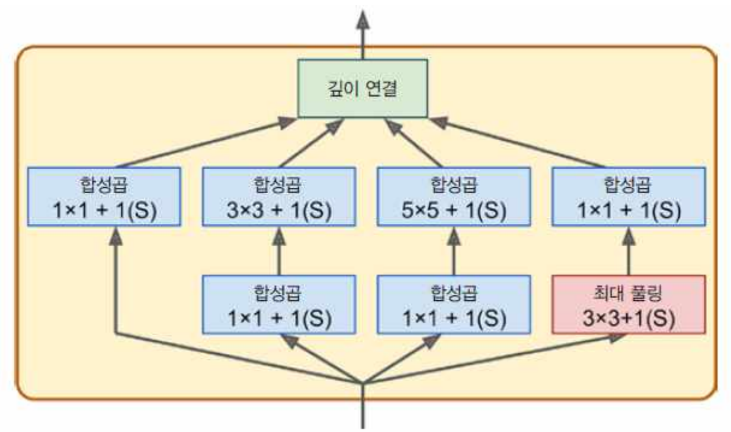
- VGGNet
  - Simonyan & Zisserman, 2014
  - Alex net 대비 Conv. Kernel size 를 줄이고 layer 를 늘림
    - 여러 개의 3x3 conv. 는 1개의 7x7 conv. 와 effective receptive field 가 같음
    - Deeper, more non-linearities



CNN 구조

- GoogleLeNet
  - Christian Szegedy (2015)
  - ILSVRC 2014 대회 우승 (error rate: 7%)
  - 특징
    - Inception module
    - 22 layers
    - 파라메터 수 감소 (AlexNet 대비 12배, VGG-16 대비 27배 감소)

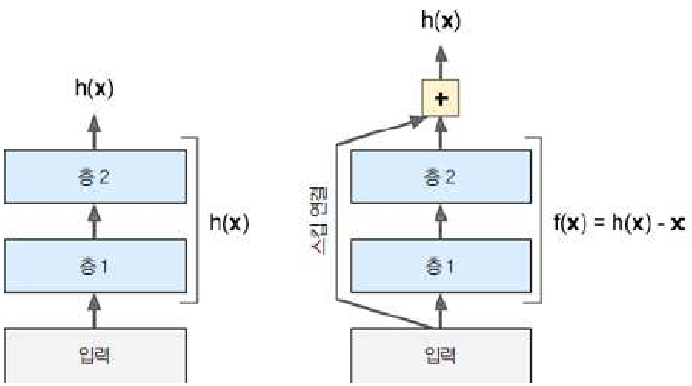




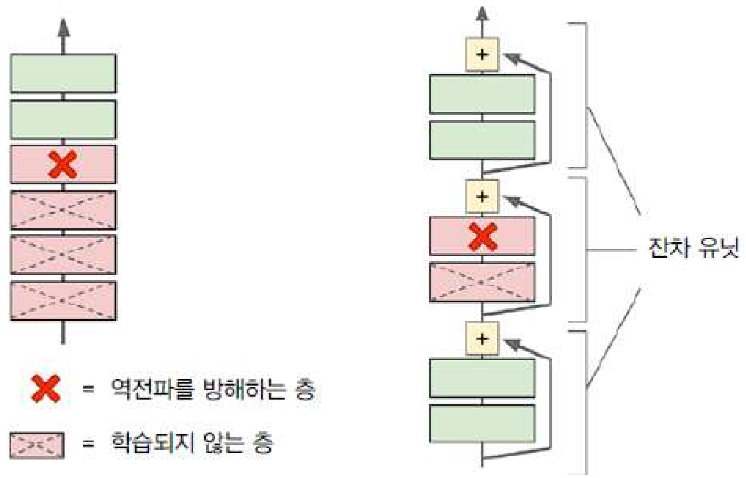
Inception module

CNN 구조

- ResNet (residual network)
  - Kaiming He (2015)
  - ILSVRC 2015 대회 우승 (error rate 3.6%)
  - 특징
    - 152 layers
    - Skip connection & residual learning: back propagation 개선



skip connection



Residual learning

CNN 구조

- ResNet (residual network)