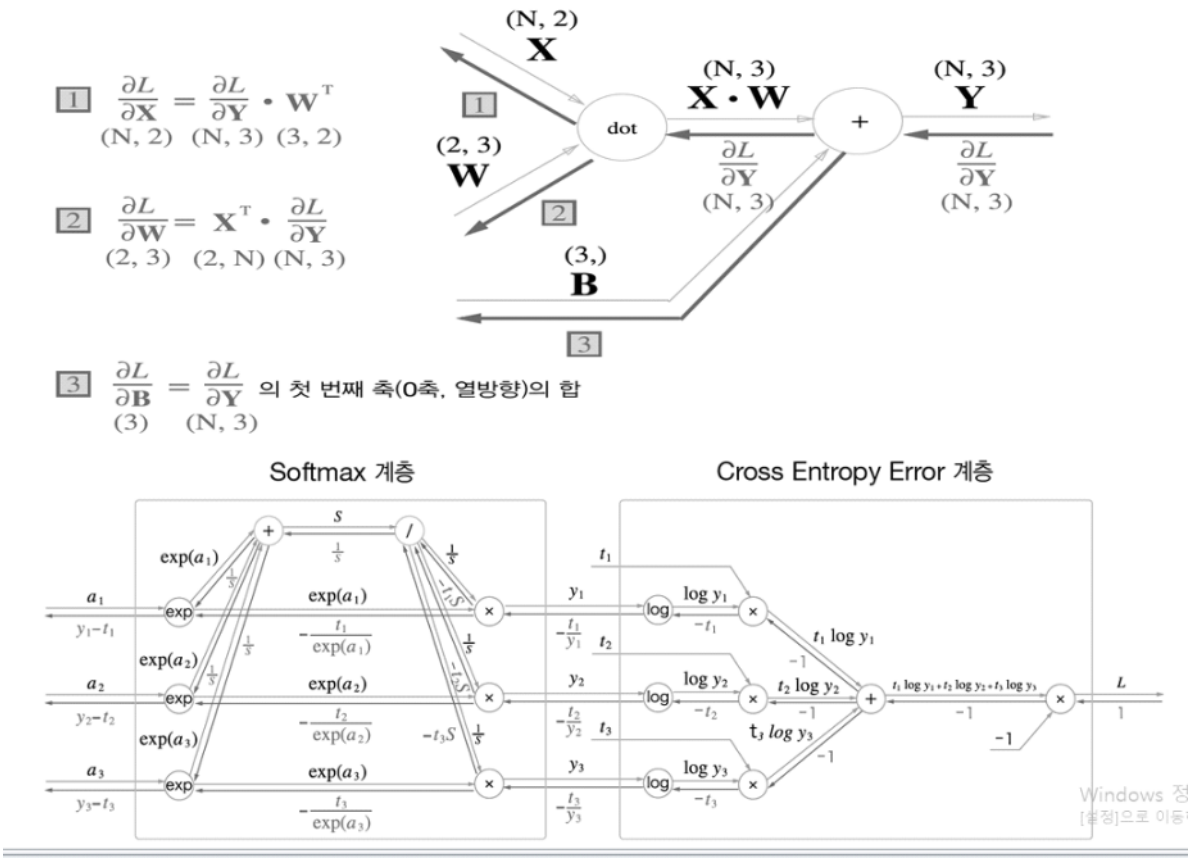
8/11 수업

2020년 8월 11일 화요일 오전 9:48

1장 numpy
2장 퍼셉트론
3장 3층 신경망 생성
4장 2층 신경망 생성 (수치미분)
5장 2층 신경망 생성 (오차역전파) --> 계산그래프
6장 신경망을 학습시키는 기술들
- 언더피팅을 방지하는 방법들
    - 고급 경사하강법
    - 가중치 초기값 선정 : 정규분포를 이루게끔 구성되어야 학습이 잘 된다.
        # 랜덤으로 생성한 가중치 행렬 * (Xavier(사비에르), He)

- 오버피팅을 방지하는 방법들
    - 가중치 감소
    - 드롭 아웃

7장 CNN
8장 자전거 타고 자유롭게 돌아다님
이미넷 대회에서 우승한 신경망 VGG 구현
신경망의 발전 역사 : 텐서플로우로 구현
object detection, yolo



신경망 구현시 가장 큰 어려움중 하나 - 오버피팅

오버피팅이 일어나는 경우?

1. 매개변수가 많고 표현력이 높은 모델
   (뉴런의 개수와 층이 불필요하게 많은 경우)

2. 훈련데이터가 적은 경우

- 한 클래스당 2000장이 필요하다.

가중치 감소 (p217)

학습과정에서 큰 가중치에 대해서는 그에 상응하는 큰 패널티를 부과하여 오버피팅을 억제하는 방법입니다.

	입력층	은닉층	출력층
고양이 사진	○	귀가 있어요	아주 큰 가중치를 출력
	○	꼬리가 있어요	가중치 출력
	○	집게발을 가지고 있어요	가중치 출력
	○	장난스럽게 보여요	가중치 출력

귀가 없는 고양이 사진 -----> 신경망 -------> 개
> 오버피팅 발생

해결방법 : 아주 큰 가중치에 큰 패널티를 부여해서 가중치 매개변수의 값이 작아지도록 만들면 된다.

오버피팅을 일으키는 신경망을 수술하는 방법

1. __init__ 함수 안에 클래스 내에서 사용할 변수를 정의한다.
```
self.weight_decay_lambda = 0.001  # 인스턴스 변수 추가
```

2. 손실(loss) 함수를 다음과 같이 고친다.
```
def loss(self, x):
    y = self.predict(x)
    weight_decay = 0
    for idx in range(1, 4):
        W = self.params['W' + str(idx)]
        weight_decay += 0.5 * self.weight_decay_lambda * np.sum(W**2)

    return self.lastlayer.forward(y, t) + weight_decay
```

문제93. 어제 마지막 문제로 만들었던 신경망에서 배치 정규화를 빼고 층을 7층으로 만들어서 구현하시오

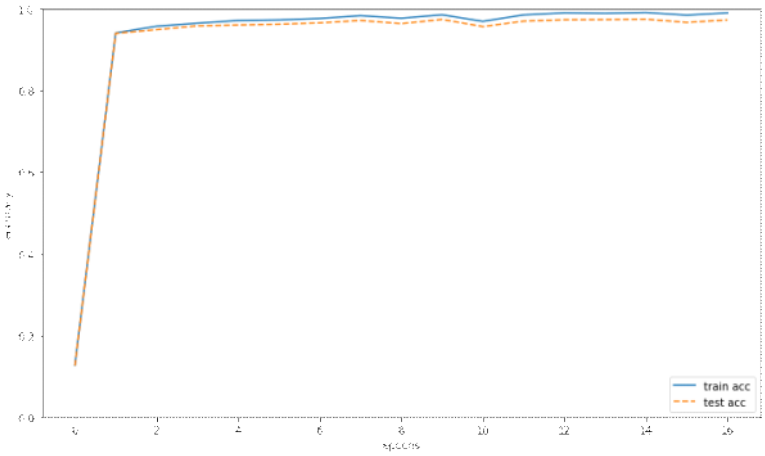
```
# coding: utf-8
import sys, os


sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
from common.layers import *
from common.gradient import numerical_gradient
from collections import OrderedDict
import matplotlib.pyplot as plt
from dataset.mnist import load_mnist

from common.optimizer import *


class multiLayerNet:

    def __init__(self, input_size, hidden_size, output_size, weight_init_std=0.01):
        # 가중치 초기화
        self.params = {}
        self.params['W1'] = np.random.randn(input_size, hidden_size) / np.sqrt(input_size) * np.sqrt(2)
        self.params['b1'] = np.zeros(hidden_size)
        self.params['W2'] = np.random.randn(hidden_size, hidden_size) / np.sqrt(hidden_size) * np.sqrt(2)
        self.params['b2'] = np.zeros(hidden_size)
        self.params['W3'] = np.random.randn(hidden_size, hidden_size) / np.sqrt(hidden_size) * np.sqrt(2)
        self.params['b3'] = np.zeros(hidden_size)
        self.params['W4'] = np.random.randn(hidden_size, hidden_size) / np.sqrt(hidden_size) * np.sqrt(2)
```

```
        self.params['b4'] = np.zeros(hidden_size)
        self.params['W5'] = np.random.randn(hidden_size, hidden_size) / np.sqrt(hidden_size) * np.sqrt(2)
        self.params['b5'] = np.zeros(hidden_size)
        self.params['W6'] = np.random.randn(hidden_size, hidden_size) / np.sqrt(hidden_size) * np.sqrt(2)
        self.params['b6'] = np.zeros(hidden_size)
        self.params['W7'] = np.random.randn(hidden_size, output_size) / np.sqrt(hidden_size) * np.sqrt(2)
        self.params['b7'] = np.zeros(output_size)


        # 계층 생성
        self.layers = OrderedDict()
        self.layers['Affine1'] = Affine(self.params['W1'], self.params['b1'])
        self.layers['Relu1'] = Relu()
        self.layers['Affine2'] = Affine(self.params['W2'], self.params['b2'])
        self.layers['Relu2'] = Relu()
        self.layers['Affine3'] = Affine(self.params['W3'], self.params['b3'])
        self.layers['Relu3'] = Relu()
        self.layers['Affine4'] = Affine(self.params['W4'], self.params['b4'])
        self.layers['Relu4'] = Relu()
        self.layers['Affine5'] = Affine(self.params['W5'], self.params['b5'])
        self.layers['Relu5'] = Relu()
        self.layers['Affine6'] = Affine(self.params['W6'], self.params['b6'])
        self.layers['Relu6'] = Relu()
        self.layers['Affine7'] = Affine(self.params['W7'], self.params['b7'])

        self.lastLayer = SoftmaxWithLoss()

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

        return x

    # x : 입력 데이터, t : 정답 레이블
    def loss(self, x, t):
        y = self.predict(x)
        return self.lastLayer.forward(y, t)

    def accuracy(self, x, t):
        y = self.predict(x)
        y = np.argmax(y, axis=1)
        if t.ndim != 1: t = np.argmax(t, axis=1)
        accuracy = np.sum(y == t) / float(x.shape[0])
        return accuracy

    # x : 입력 데이터, t : 정답 레이블

    def numerical_gradient(self, x, t):
        loss_W = lambda W: self.loss(x, t)

        grads = {}
        grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
        grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
        grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
        grads['b2'] = numerical_gradient(loss_W, self.params['b2'])
        grads['W3'] = numerical_gradient(loss_W, self.params['W3'])
        grads['b3'] = numerical_gradient(loss_W, self.params['b3'])
        grads['W4'] = numerical_gradient(loss_W, self.params['W4'])
        grads['b4'] = numerical_gradient(loss_W, self.params['b4'])
        grads['W5'] = numerical_gradient(loss_W, self.params['W5'])
        grads['b5'] = numerical_gradient(loss_W, self.params['b5'])
        grads['W6'] = numerical_gradient(loss_W, self.params['W6'])
        grads['b6'] = numerical_gradient(loss_W, self.params['b6'])
        grads['W7'] = numerical_gradient(loss_W, self.params['W7'])
        grads['b7'] = numerical_gradient(loss_W, self.params['b7'])

        return grads

    def gradient(self, x, t):
        # forward
        self.loss(x, t)
        # backward
        dout = 1
        dout = self.lastLayer.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['Affine1'].dW, self.layers['Affine1'].db
        grads['W2'], grads['b2'] = self.layers['Affine2'].dW, self.layers['Affine2'].db
        grads['W3'], grads['b3'] = self.layers['Affine3'].dW, self.layers['Affine3'].db
        grads['W4'], grads['b4'] = self.layers['Affine4'].dW, self.layers['Affine4'].db
        grads['W5'], grads['b5'] = self.layers['Affine5'].dW, self.layers['Affine5'].db
        grads['W6'], grads['b6'] = self.layers['Affine6'].dW, self.layers['Affine6'].db
        grads['W7'], grads['b7'] = self.layers['Affine7'].dW, self.layers['Affine7'].db
        return grads

# 데이터 읽기
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)
network = multiLayerNet(input_size=784, hidden_size=50, output_size=10)

# 하이퍼파라미터

iters_num = 10000  # 반복 횟수를 적절히 설정한다.
train_size = x_train.shape[0] # 60000 개
batch_size = 100  # 미니배치 크기
learning_rate = 0.1

train_loss_list = []
train_acc_list = []
test_acc_list = []

# 1에폭당 반복 수

iter_per_epoch = max(train_size / batch_size, 1)

optimizer = SGD(lr=0.01)

for i in range(iters_num): # 10000
    # 미니배치 획득  # 랜덤으로 100개씩 뽑아서 10000번을 수행하니까 백만번
    batch_mask = np.random.choice(train_size, batch_size) # 100개 씩 뽑아서 10000번 백만번
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]

    # 기울기 계산
    #grad = network.numerical_gradient(x_batch, t_batch)
    grad = network.gradient(x_batch, t_batch)

    grad = network.gradient(x_batch, t_batch)
    params = network.params
    optimizer.update(params , grad)

    # 매개변수 갱신
    for key in ('W1', 'b1', 'W2', 'b2', 'W3', 'b3', 'W4', 'b4', 'W5', 'b5', 'W6', 'b6', 'W7', 'b7'):
        network.params[key] -= learning_rate * grad[key]

    # 학습 경과 기록
    loss = network.loss(x_batch, t_batch)
    train_loss_list.append(loss) # cost 가 점점 줄어드는것을 보려고

    # 1에폭당 정확도 계산 # 여기는 훈련이 아니라 1에폭 되었을때 정확도만 체크
    if i % iter_per_epoch == 0: # 600 번마다 정확도 쌓는다.

        train_acc = network.accuracy(x_train, t_train)
        test_acc = network.accuracy(x_test, t_test)
        train_acc_list.append(train_acc) # 10000/600 개  16개 # 정확도가 점점 올라감
        test_acc_list.append(test_acc)  # 10000/600 개 16개 # 정확도가 점점 올라감
        print("train acc, test acc | " + str(train_acc) + ", " + str(test_acc))

# 그래프 그리기
markers = {'train': 'o', 'test': 's'}
x = np.arange(len(train_acc_list))
plt.figure(figsize=(12, 7))
plt.plot(x, train_acc_list, label='train acc')
plt.plot(x, test_acc_list, label='test acc', linestyle='--')
plt.xlabel("epochs")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
plt.legend(loc='lower right')
plt.show()
```

```
# import pickle
# with open('d:\\mnist_adam_He.pkl', 'wb') as f:
#     pickle.dump(network.params, f)
```

```
train acc, test acc | 0.12675, 0.1257
train acc, test acc | 0.9414166666666667, 0.9403
train acc, test acc | 0.9577666666666667, 0.9494
train acc, test acc | 0.9652166666666666, 0.9587
train acc, test acc | 0.9718666666666667, 0.9608
train acc, test acc | 0.9733333333333334, 0.9628
train acc, test acc | 0.9768, 0.966
train acc, test acc | 0.9837333333333333, 0.972
train acc, test acc | 0.9774166666666667, 0.9642
train acc, test acc | 0.9858833333333333, 0.9745
train acc, test acc | 0.9697666666666667, 0.957
train acc, test acc | 0.98565, 0.9703
train acc, test acc | 0.99055, 0.9737
train acc, test acc | 0.9894666666666667, 0.9741
train acc, test acc | 0.9913333333333333, 0.9751
train acc, test acc | 0.9850333333333333, 0.9672
train acc, test acc | 0.9905333333333334, 0.9733
```



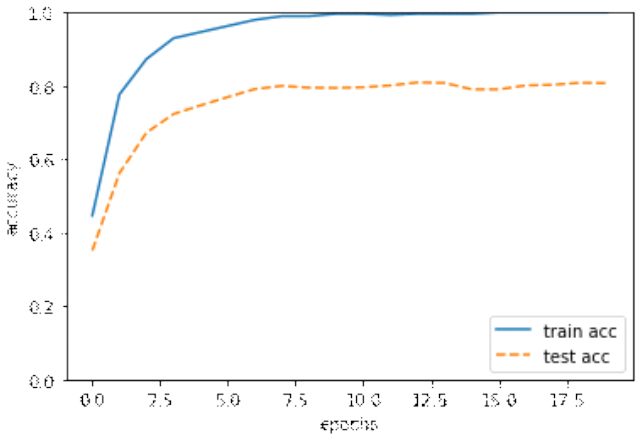
문제94. 어제 마지막 문제 전체 코드에 가중치 감소 를 처방하는 코드를 추가하고 돌리시오

```
from common.optimizer import*
import sys, os
sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
from common.layers import *
from common.gradient import numerical_gradient
from collections import OrderedDict
import matplotlib.pyplot as plt
from dataset.mnist import load_mnist

class TwoLayerNet:
    def __init__(self, input_size, hidden_size, output_size, weight_init_std=0.01):
        # 가중치 초기화
        self.params = {}
        self.params['W1'] = weight_init_std * np.random.randn(input_size, hidden_size)
        self.params['b1'] = np.zeros(hidden_size)
        self.params['W2'] = weight_init_std * np.random.randn(hidden_size, hidden_size)
        self.params['b2'] = np.zeros(hidden_size)
        self.params['W3'] = weight_init_std * np.random.randn(hidden_size, hidden_size)
        self.params['b3'] = np.zeros(hidden_size)
        self.params['W4'] = weight_init_std * np.random.randn(hidden_size, hidden_size)
        self.params['b4'] = np.zeros(hidden_size)
        self.params['W5'] = weight_init_std * np.random.randn(hidden_size, hidden_size)
        self.params['b5'] = np.zeros(hidden_size)
        self.params['W6'] = weight_init_std * np.random.randn(hidden_size, hidden_size)
        self.params['b6'] = np.zeros(hidden_size)
        self.params['W7'] = weight_init_std * np.random.randn(hidden_size, output_size)
        self.params['b7'] = np.zeros(output_size)
        self.weight_decay_lambda = 0.001
```

```
        # 계층 생성

        self.layers = OrderedDict()
        self.layers['Affine1'] = Affine(self.params['W1'], self.params['b1'])
        self.layers['BatchNorm1'] = BatchNormalization(gamma=1.0, beta = 0.)
        self.layers['Relu1'] = Relu()
        self.layers['Affine2'] = Affine(self.params['W2'], self.params['b2'])
        self.layers['BatchNorm2'] = BatchNormalization(gamma=1.0, beta = 0.)
        self.layers['Relu2'] = Relu()
        self.layers['Affine3'] = Affine(self.params['W3'], self.params['b3'])
        self.layers['BatchNorm3'] = BatchNormalization(gamma=1.0, beta = 0.)
        self.layers['Relu3'] = Relu()
        self.layers['Affine4'] = Affine(self.params['W4'], self.params['b4'])
        self.layers['BatchNorm4'] = BatchNormalization(gamma=1.0, beta = 0.)
        self.layers['Relu4'] = Relu()
        self.layers['Affine5'] = Affine(self.params['W5'], self.params['b5'])
        self.layers['BatchNorm5'] = BatchNormalization(gamma=1.0, beta = 0.)
        self.layers['Relu5'] = Relu()
        self.layers['Affine6'] = Affine(self.params['W6'], self.params['b6'])
        self.layers['BatchNorm6'] = BatchNormalization(gamma=1.0, beta = 0.)
        self.layers['Relu6'] = Relu()
        self.layers['Affine7'] = Affine(self.params['W7'], self.params['b7'])


        self.lastLayer = SoftmaxWithLoss()

    def predict(self, x):
        for layer in self.layers.values():
            x = layer.forward(x)
        return x
    # x : 입력 데이터, t : 정답 레이블
    def loss(self, x, t):
        y = self.predict(x)
        weight_decay = 0
        for idx in range(1, 3):
            W = self.params['W' + str(idx)]
            weight_decay += 0.5 * self.weight_decay_lambda * np.sum(W**2)

        return self.lastLayer.forward(y, t) + weight_decay

    def accuracy(self, x, t):
        y = self.predict(x)
        y = np.argmax(y, axis=1)
        if t.ndim != 1: t = np.argmax(t, axis=1)
        accuracy = np.sum(y == t) / float(x.shape[0])
        return accuracy
    # x : 입력 데이터, t : 정답 레이블

    def gradient(self, x, t):
        # forward
        self.loss(x, t)
        # backward
        dout = 1
        dout = self.lastLayer.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['Affine1'].dW, self.layers['Affine1'].db
        grads['W2'], grads['b2'] = self.layers['Affine2'].dW, self.layers['Affine2'].db
        grads['W3'], grads['b3'] = self.layers['Affine3'].dW, self.layers['Affine3'].db
        grads['W4'], grads['b4'] = self.layers['Affine4'].dW, self.layers['Affine4'].db
        grads['W5'], grads['b5'] = self.layers['Affine5'].dW, self.layers['Affine5'].db
        grads['W6'], grads['b6'] = self.layers['Affine6'].dW, self.layers['Affine6'].db
        grads['W7'], grads['b7'] = self.layers['Affine7'].dW, self.layers['Affine7'].db
        return grads

# 데이터 읽기
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)
x_train = x_train[:300]
t_train = t_train[:300]
network = TwoLayerNet(784,100,10)
# 하이퍼파라미터


epoch=20#int(input('에폭수를 입력하세요: '))
iter_per_epoch = max(train_size / batch_size, 1)
iters_num = iter_per_epoch*epoch  # 반복 횟수를 적절히 설정한다.
```

```
train_size = x_train.shape[0] # 60000 개
batch_size = 100  # 미니배치 크기
learning_rate = 0.1

train_loss_list = []
train_acc_list = []
test_acc_list = []

# 1에폭당 반복 수


optimizer = Adam()

for i in range(int(iters_num)): # 10000

    # 미니배치 획득  # 랜덤으로 100개씩 뽑아서 10000번을 수행하니까 백만번


    batch_mask = np.random.choice(train_size, batch_size) # 100개 씩 뽑아서 10000번 백만번
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]
    # 기울기 계산
    #grad = network.numerical_gradient(x_batch, t_batch)

    grad = network.gradient(x_batch, t_batch)
    params = network.params
    optimizer.update(params,grad)


    # 매개변수 갱신
    #for key in ('W1', 'b1', 'W2', 'b2', 'W3', 'b3'):
    #    network.params[key] -= learning_rate * grad[key]
    # 학습 경과 기록고
    # 1에폭당 정확도 계산 # 여기는 훈련이 아니라 1에폭 되었을때 정확도만 체크
    if i % iter_per_epoch == 0: # 600 번마다 정확도 쌓는다.
        train_acc = network.accuracy(x_train, t_train)
        test_acc = network.accuracy(x_test, t_test)
        train_acc_list.append(train_acc) # 10000/600 개  16개 # 정확도가 점점 올라감
        test_acc_list.append(test_acc)  # 10000/600 개 16개 # 정확도가 점점 올라감
        print("train acc, test acc | " + str(train_acc) + ", " + str(test_acc))
        loss = network.loss(x_batch, t_batch)
        train_loss_list.append(loss) # cost 가 점점 줄어드는것을 보려
        print(loss)

markers = {'train': 'o', 'test': 's'}
x = np.arange(len(train_acc_list))
plt.plot(x, train_acc_list, label='train acc')
plt.plot(x, test_acc_list, label='test acc', linestyle='--')
plt.xlabel("epochs")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
plt.legend(loc='lower right')
plt.show()
```

```
train acc, test acc | 1.0, 0.8013
0.3435901535865145
train acc, test acc | 1.0, 0.8031
0.31907638224351137
train acc, test acc | 1.0, 0.8085
0.30847822131458713
train acc, test acc | 1.0, 0.8075
0.271955222922099
```

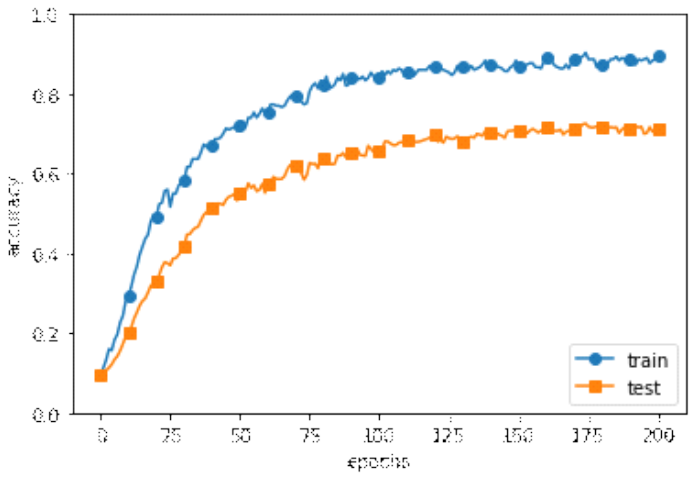


오버피팅 발생 예제1

```
# coding: utf-8
import os
import sys


sys.path.append(os.pardir)  # 親ディレクトリのファイルをインポートするための設定
import numpy as np
import matplotlib.pyplot as plt
from dataset.mnist import load_mnist
from common.multi_layer_net import MultiLayerNet
from common.optimizer import SGD

(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True)


# 過学習を再現するために、学習データを削減
x_train = x_train[:300]
t_train = t_train[:300]


# weight decay（荷重減衰）の設定 =======================
#weight_decay_lambda = 0 # weight decayを使用しない場合
weight_decay_lambda = 0.1
# ====================================================

network = MultiLayerNet(input_size=784, hidden_size_list=[100, 100, 100, 100, 100, 100], output_size=10,
                        weight_decay_lambda=weight_decay_lambda)
optimizer = SGD(lr=0.01)


max_epochs = 201
train_size = x_train.shape[0]
batch_size = 100

train_loss_list = []
train_acc_list = []
test_acc_list = []


iter_per_epoch = max(train_size / batch_size, 1)
epoch_cnt = 0

for i in range(1000000000):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]

    grads = network.gradient(x_batch, t_batch)
    optimizer.update(network.params, grads)

    if i % iter_per_epoch == 0:
        train_acc = network.accuracy(x_train, t_train)
        test_acc = network.accuracy(x_test, t_test)
        train_acc_list.append(train_acc)
        test_acc_list.append(test_acc)

        print("epoch:" + str(epoch_cnt) + ", train acc:" + str(train_acc) + ", test acc:" + str(test_acc))
```

```
        epoch_cnt += 1
        if epoch_cnt >= max_epochs:
            break


# 3.グラフの描画==========
markers = {'train': 'o', 'test': 's'}
x = np.arange(max_epochs)
plt.plot(x, train_acc_list, marker='o', label='train', markevery=10)
plt.plot(x, test_acc_list, marker='s', label='test', markevery=10)
plt.xlabel("epochs")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
plt.legend(loc='lower right')
plt.show()
```

```
epoch:196, train acc:0.89, test acc:0.7019
epoch:197, train acc:0.8766666666666667, test acc:0.7166
epoch:198, train acc:0.8833333333333333, test acc:0.7048
epoch:199, train acc:0.8866666666666667, test acc:0.7105
epoch:200, train acc:0.8966666666666666, test acc:0.7138
```



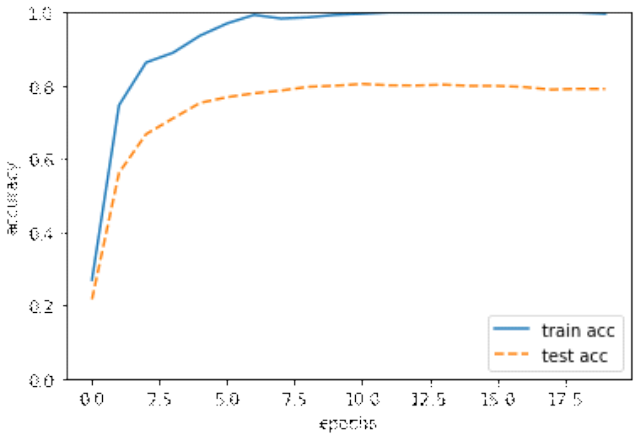
오버피팅 발생 예제2

```
from common.optimizer import*
import sys, os
sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
from common.layers import *
from common.gradient import numerical_gradient
from collections import OrderedDict
import matplotlib.pyplot as plt
from dataset.mnist import load_mnist


class TwoLayerNet:
    def __init__(self, input_size, hidden_size, output_size, weight_init_std=0.01):
        # 가중치 초기화
        self.params = {}
        self.params['W1'] = weight_init_std * np.random.randn(input_size, hidden_size)
        self.params['b1'] = np.zeros(hidden_size)
        self.params['W2'] = weight_init_std * np.random.randn(hidden_size, hidden_size)
        self.params['b2'] = np.zeros(hidden_size)
        self.params['W3'] = weight_init_std * np.random.randn(hidden_size, hidden_size)
        self.params['b3'] = np.zeros(hidden_size)
        self.params['W4'] = weight_init_std * np.random.randn(hidden_size, hidden_size)
        self.params['b4'] = np.zeros(hidden_size)
        self.params['W5'] = weight_init_std * np.random.randn(hidden_size, hidden_size)
        self.params['b5'] = np.zeros(hidden_size)
        self.params['W6'] = weight_init_std * np.random.randn(hidden_size, hidden_size)
        self.params['b6'] = np.zeros(hidden_size)
        self.params['W7'] = weight_init_std * np.random.randn(hidden_size, output_size)
        self.params['b7'] = np.zeros(output_size)
        self.weight_decay_lambda = 0.001
        # 계층 생성
```

```
        self.layers = OrderedDict()
        self.layers['Affine1'] = Affine(self.params['W1'], self.params['b1'])
        self.layers['BatchNorm1'] = BatchNormalization(gamma=1.0, beta = 0.)
        self.layers['Relu1'] = Relu()
        self.layers['Affine2'] = Affine(self.params['W2'], self.params['b2'])
        self.layers['BatchNorm2'] = BatchNormalization(gamma=1.0, beta = 0.)
        self.layers['Relu2'] = Relu()
        self.layers['Affine3'] = Affine(self.params['W3'], self.params['b3'])
        self.layers['BatchNorm3'] = BatchNormalization(gamma=1.0, beta = 0.)
        self.layers['Relu3'] = Relu()
        self.layers['Affine4'] = Affine(self.params['W4'], self.params['b4'])
        self.layers['BatchNorm4'] = BatchNormalization(gamma=1.0, beta = 0.)
        self.layers['Relu4'] = Relu()
        self.layers['Affine5'] = Affine(self.params['W5'], self.params['b5'])
        self.layers['BatchNorm5'] = BatchNormalization(gamma=1.0, beta = 0.)
        self.layers['Relu5'] = Relu()
        self.layers['Affine6'] = Affine(self.params['W6'], self.params['b6'])
        self.layers['BatchNorm6'] = BatchNormalization(gamma=1.0, beta = 0.)
        self.layers['Relu6'] = Relu()
        self.layers['Affine7'] = Affine(self.params['W7'], self.params['b7'])



        self.lastLayer = SoftmaxWithLoss()

    def predict(self, x):
        for layer in self.layers.values():
            x = layer.forward(x)
        return x
    # x : 입력 데이터, t : 정답 레이블
    def loss(self, x, t):
        y = self.predict(x)
        return self.lastLayer.forward(y, t)

    def accuracy(self, x, t):
        y = self.predict(x)
        y = np.argmax(y, axis=1)
        if t.ndim != 1: t = np.argmax(t, axis=1)
        accuracy = np.sum(y == t) / float(x.shape[0])
        return accuracy
    # x : 입력 데이터, t : 정답 레이블

    def gradient(self, x, t):
        # forward
        self.loss(x, t)
        # backward
        dout = 1
        dout = self.lastLayer.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['Affine1'].dW, self.layers['Affine1'].db
        grads['W2'], grads['b2'] = self.layers['Affine2'].dW, self.layers['Affine2'].db
        grads['W3'], grads['b3'] = self.layers['Affine3'].dW, self.layers['Affine3'].db
        grads['W4'], grads['b4'] = self.layers['Affine4'].dW, self.layers['Affine4'].db
        grads['W5'], grads['b5'] = self.layers['Affine5'].dW, self.layers['Affine5'].db
        grads['W6'], grads['b6'] = self.layers['Affine6'].dW, self.layers['Affine6'].db
        grads['W7'], grads['b7'] = self.layers['Affine7'].dW, self.layers['Affine7'].db
        return grads

# 데이터 읽기
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)
x_train = x_train[:300]
t_train = t_train[:300]
network = TwoLayerNet(784,100,10)
# 하이퍼파라미터


epoch=20#int(input('에폭수를 입력하세요: '))
iter_per_epoch = max(train_size / batch_size, 1)
iters_num = iter_per_epoch*epoch  # 반복 횟수를 적절히 설정한다.
train_size = x_train.shape[0] # 60000 개
batch_size = 100  # 미니배치 크기
learning_rate = 0.1

train_loss_list = []
```

```
train_acc_list = []
test_acc_list = []

# 1에폭당 반복 수


optimizer = Adam()

for i in range(int(iters_num)): # 10000

    # 미니배치 획득  # 랜덤으로 100개씩 뽑아서 10000번을 수행하니까 백만번


    batch_mask = np.random.choice(train_size, batch_size) # 100개 씩 뽑아서 10000번 백만번
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]
    # 기울기 계산
    #grad = network.numerical_gradient(x_batch, t_batch)

    grad = network.gradient(x_batch, t_batch)
    params = network.params
    optimizer.update(params,grad)


    # 매개변수 갱신
    #for key in ('W1', 'b1', 'W2', 'b2', 'W3', 'b3'):
    #    network.params[key] -= learning_rate * grad[key]
    # 학습 경과 기록고
    # 1에폭당 정확도 계산 # 여기는 훈련이 아니라 1에폭 되었을때 정확도만 체크
    if i % iter_per_epoch == 0: # 600 번마다 정확도 쌓는다.
        train_acc = network.accuracy(x_train, t_train)
        test_acc = network.accuracy(x_test, t_test)
        train_acc_list.append(train_acc) # 10000/600 개  16개 # 정확도가 점점 올라감
        test_acc_list.append(test_acc)  # 10000/600 개 16개 # 정확도가 점점 올라감
        print("train acc, test acc | " + str(train_acc) + ", " + str(test_acc))
        loss = network.loss(x_batch, t_batch)
        train_loss_list.append(loss) # cost 가 점점 줄어드는것을 보려
        print(loss)

markers = {'train': 'o', 'test': 's'}
x = np.arange(len(train_acc_list))
plt.plot(x, train_acc_list, label='train acc')
plt.plot(x, test_acc_list, label='test acc', linestyle='--')
plt.xlabel("epochs")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
plt.legend(loc='lower right')
plt.show()
```

```
train acc, test acc | 1.0, 0.7994
0.3944577016369147
train acc, test acc | 1.0, 0.7964
0.34342305056530037
train acc, test acc | 1.0, 0.7893
0.3105120761693141
train acc, test acc | 1.0, 0.7915
0.2836277506918979
train acc, test acc | 0.9966666666666667, 0.7911
0.2682438140369494
```



문제95. 저자가 만든 오버피팅을 일으키는 스크립트르르 이용해서 가중치 감소 테스트를 하시오

6장 폴더안에 overfit_weight_decay.py

처음 실습은 그냥 돌리면 weight_decay_lambda 값을 0으로 하고 테스트 하고
훈련데이터는 10개중 9개를 맞추고
테스트 데이터는 10개중 7개를 맞춘다.

두번째는 weight_decay_lambda 값을 0.01로 하고 테스트 한다,
epoch:200, train acc:1.0, test acc:0.7705

세번째는 weight_decay_lambda 값을 0.0로 하고 테스트 한다,
epoch:200, train acc:1.0, test acc:0.7674

```
# coding: utf-8
import os
import sys

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 common.multi_layer_net import MultiLayerNet
from common.optimizer import SGD

(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True)

# 오버피팅을 재현하기 위해 학습 데이터 수를 줄임
x_train = x_train[:300]
t_train = t_train[:300]

# weight decay(가중치 감쇠) 설정 =======================
#weight_decay_lambda = 0 # weight decay를 사용하지 않을 경우
weight_decay_lambda = 0.01
# ====================================================

network = MultiLayerNet(input_size=784, hidden_size_list=[100, 100, 100, 100, 100, 100], output_size=10,
                        weight_decay_lambda=weight_decay_lambda)
optimizer = SGD(lr=0.01) # 학습률이 0.01인 SGD로 매개변수 갱신

max_epochs = 201
train_size = x_train.shape[0]
batch_size = 100

train_loss_list = []
train_acc_list = []
test_acc_list = []

iter_per_epoch = max(train_size / batch_size, 1)
epoch_cnt = 0

for i in range(1000000000):
```

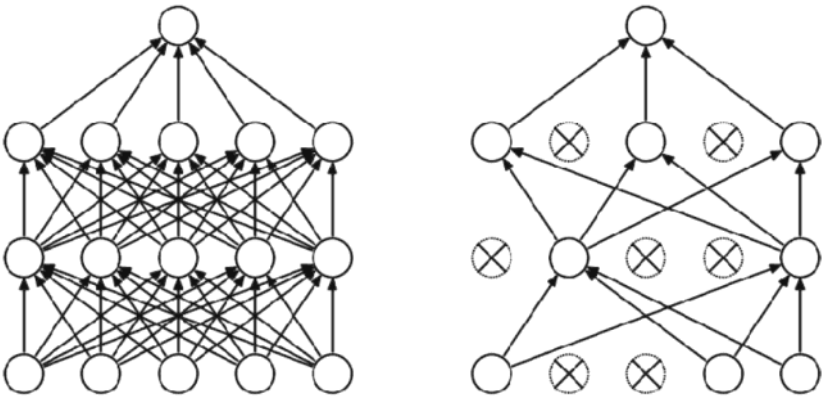
```
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]

    grads = network.gradient(x_batch, t_batch)
    optimizer.update(network.params, grads)

    if i % iter_per_epoch == 0:
        train_acc = network.accuracy(x_train, t_train)
        test_acc = network.accuracy(x_test, t_test)
        train_acc_list.append(train_acc)
        test_acc_list.append(test_acc)

        print("epoch:" + str(epoch_cnt) + ", train acc:" + str(train_acc) + ", test acc:" + str(test_acc))

        epoch_cnt += 1
        if epoch_cnt >= max_epochs:
            break


# 그래프 그리기==========
markers = {'train': 'o', 'test': 's'}
x = np.arange(max_epochs)
plt.plot(x, train_acc_list, marker='o', label='train', markevery=10)
plt.plot(x, test_acc_list, marker='s', label='test', markevery=10)
plt.xlabel("epochs")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
plt.legend(loc='lower right')
plt.show()
```

드롭 아웃(dropout) - p219

" 오버피팅을 억제하기 위해서 뉴런을 임의로 삭제하면서 학습하는 방법 "

% 면접문제 %
5. 오버피팅이 주로 어느경우애 발생합니까?
답 : 매개변수가 많고 표현력이 높은 모델에서 발생합니다.
그리고 훈련데이터가 적을때도 발생합니다.

<fig 6-22>



(a) 일반 신경망 (b) 드롭아웃을 적용한 신경망

드롭아웃 클래스

```
class Dropout:

    """
    http://arxiv.org/abs/1207.0580
    """
```

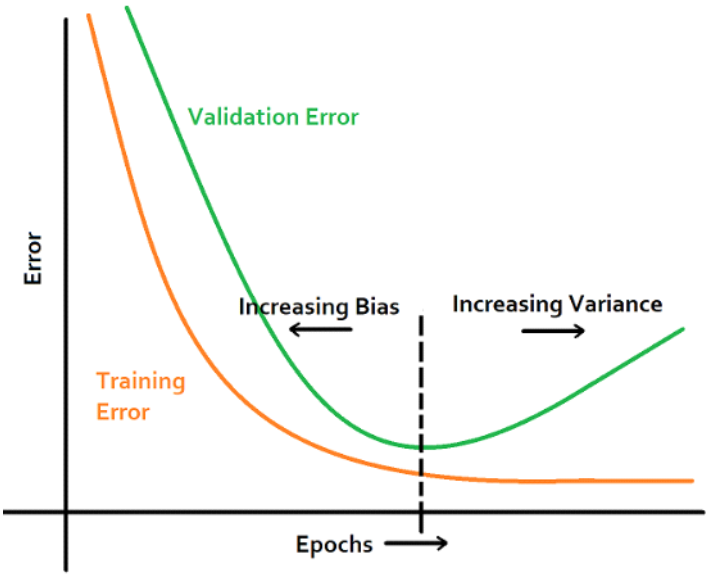
```
    def __init__(self, dropout_ratio=0.15): # 신경망에서 15%의 뉴런을 삭제하겠다.
        self.dropout_ratio = dropout_ratio
        self.mask = None


    def forward(self, x, train_flg=True):  # 훈련할때는 드롭아웃기능을 키고
                                           # 테스트 할 때는 드롭아웃 기능을 끈다,.
        if train_flg:
            self.mask = np.random.rand(*x.shape) > self.dropout_ratio
            #100% 중에서 85%만 남겨두고 15%의 노드는 삭제하겠다.
            return x * self.mask
        else:
            return x * (1.0 - self.dropout_ratio)

    def backward(self, dout):
        return dout * self.mask
```

```
self.layers = OrderedDict()
self.layers['Affine1'] = Affine(self.params['W1'], self.params['b1'])
self.layers['BatchNorm1'] = BatchNormalization(gamma=1.0, beta = 0.)
self.layers['Relu1'] = Relu()
self.layers['Dropout1'] = Dropout[]

self.layers['Affine2'] = Affine(self.params['W2'], self.params['b2'])
self.layers['BatchNorm2'] = BatchNormalization(gamma=1.0, beta = 0.)
self.layers['Relu2'] = Relu()
self.layers['Dropout2'] = Dropout[]
```

 # 오버피팅을 해결하는 방법

1. 가중치 감소
2. dropout
3. keras의 early stop 기능

Early Stopping



오버피팅을 방지 방법으로

1. validation set을 이용하여 k-fold cross validation을 통해, train방법을 피드백 하는데
   그 과정에서 learning_Rate를 높이는 Regularization을 한다고 했다.

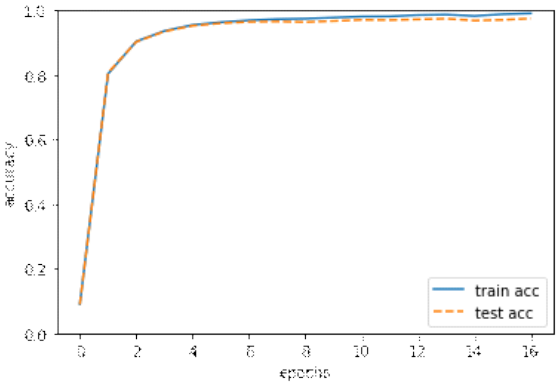
2. 조기종료(Early stropping)이 두번째 방법이다.
3. DropOut

■ 드롭아웃 적용전 3층 신경망

결과 : train acc, test acc | 0.98935, 0.9769

```
# coding: utf-8
import sys, os
sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
from common.layers import *
from common.gradient import numerical_gradient
from collections import OrderedDict
import matplotlib.pyplot as plt
from dataset.mnist import load_mnist


class TwoLayerNet:
    def __init__(self, input_size, hidden_size,hidden_size2, output_size, weight_init_std=0.01):
        # 가중치 초기화
        self.params = {}
        self.params['W1'] = weight_init_std * np.random.randn(input_size, hidden_size)
        self.params['b1'] = np.zeros(hidden_size)
        self.params['W2'] = weight_init_std * np.random.randn(hidden_size, hidden_size2)
        self.params['b2'] = np.zeros(hidden_size2)
        self.params['W3'] = weight_init_std * np.random.randn(hidden_size2, output_size)
        self.params['b3'] = np.zeros(output_size)
        # 계층 생성
        self.layers = OrderedDict()
        self.layers['Affine1'] = Affine(self.params['W1'], self.params['b1'])
        self.layers['Relu1'] = Relu()
        self.layers['Affine2'] = Affine(self.params['W2'], self.params['b2'])
        self.layers['Relu2'] = Relu()
        self.layers['Affine3'] = Affine(self.params['W3'], self.params['b3'])

        self.lastLayer = SoftmaxWithLoss()

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

        return x

    # x : 입력 데이터, t : 정답 레이블
    def loss(self, x, t):
        y = self.predict(x)
        return self.lastLayer.forward(y, t)

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

        accuracy = np.sum(y == t) / float(x.shape[0])
        return accuracy

    # x : 입력 데이터, t : 정답 레이블
    def numerical_gradient(self, x, t):
        loss_W = lambda W: self.loss(x, t)

        grads = {}
        grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
        grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
        grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
        grads['b2'] = numerical_gradient(loss_W, self.params['b2'])
        grads['W3'] = numerical_gradient(loss_W, self.params['W3'])
        grads['b3'] = numerical_gradient(loss_W, self.params['b3'])

        return grads

    def gradient(self, x, t):
        # forward
        self.loss(x, t)

        # backward
        dout = 1
        dout = self.lastLayer.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['Affine1'].dW, self.layers['Affine1'].db
        grads['W2'], grads['b2'] = self.layers['Affine2'].dW, self.layers['Affine2'].db
        grads['W3'], grads['b3'] = self.layers['Affine3'].dW, self.layers['Affine3'].db
        return grads
# 데이터 읽기
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)

network = TwoLayerNet(input_size=784, hidden_size=100,hidden_size2=50, output_size=10)

import  pickle

with  open('d:\\two_layer_w.pkl', 'wb') as  f:
    pickle.dump( network.params,  f)

# 하이퍼파라미터
iters_num = 10000  # 반복 횟수를 적절히 설정한다.
train_size = x_train.shape[0] # 60000 개
batch_size = 100  # 미니배치 크기
learning_rate = 0.1

train_loss_list = []
train_acc_list = []
test_acc_list = []

# 1에폭당 반복 수
iter_per_epoch = max(train_size / batch_size, 1)
print(iter_per_epoch) # 600

for i in range(iters_num): # 10000
    # 미니배치 획득  # 랜덤으로 100개씩 뽑아서 10000번을 수행하니까 백만번
    batch_mask = np.random.choice(train_size, batch_size) # 100개 씩 뽑아서 10000번 백만번
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]

    # 기울기 계산
    #grad = network.numerical_gradient(x_batch, t_batch)

    grad = network.gradient(x_batch, t_batch)

    # 매개변수 갱신
    for key in ('W1', 'b1', 'W2', 'b2','W3','b3'):
        network.params[key] -= learning_rate * grad[key]

    # 학습 경과 기록
    loss = network.loss(x_batch, t_batch)
    train_loss_list.append(loss) # cost 가 점점 줄어드는것을 보려고

    # 1에폭당 정확도 계산 # 여기는 훈련이 아니라 1에폭 되었을때 정확도만 체크

    if i % iter_per_epoch == 0: # 600 번마다 정확도 쌓는다.
        print(x_train.shape) # 60000,784
        train_acc = network.accuracy(x_train, t_train)
        test_acc = network.accuracy(x_test, t_test)
        train_acc_list.append(train_acc) # 10000/600 개  16개 # 정확도가 점점 올라감
        test_acc_list.append(test_acc)  # 10000/600 개 16개 # 정확도가 점점 올라감
        print("train acc, test acc | " + str(train_acc) + ", " + str(test_acc))



# 그래프 그리기
markers = {'train': 'o', 'test': 's'}
x = np.arange(len(train_acc_list))
plt.plot(x, train_acc_list, label='train acc')
plt.plot(x, test_acc_list, label='test acc', linestyle='--')
plt.xlabel("epochs")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
plt.legend(loc='lower right')
plt.show()
```



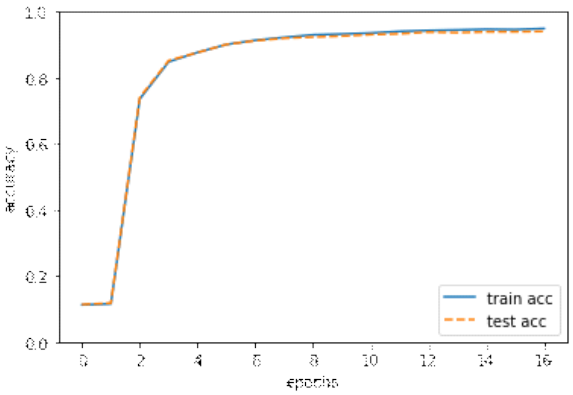
■ 드롭아웃 적용후 3층 신경망

train acc, test acc | 0.9489666666666666, 0.9462

```
# coding: utf-8
import sys, os
sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
from common.layers import *
from common.gradient import numerical_gradient
from collections import OrderedDict
import matplotlib.pyplot as plt
from dataset.mnist import load_mnist


class TwoLayerNet:
    def __init__(self, input_size, hidden_size,hidden_size2, output_size, weight_init_std=0.01):
        # 가중치 초기화
        self.params = {}
        self.params['W1'] = weight_init_std * np.random.randn(input_size, hidden_size)
        self.params['b1'] = np.zeros(hidden_size)
        self.params['W2'] = weight_init_std * np.random.randn(hidden_size, hidden_size2)
        self.params['b2'] = np.zeros(hidden_size2)
        self.params['W3'] = weight_init_std * np.random.randn(hidden_size2, output_size)
        self.params['b3'] = np.zeros(output_size)
        # 계층 생성
        self.layers = OrderedDict()
        self.layers['Affine1'] = Affine(self.params['W1'], self.params['b1'])
        self.layers['Relu1'] = Relu()
        self.layers['Dropout1'] = Dropout()

        self.layers['Affine2'] = Affine(self.params['W2'], self.params['b2'])
        self.layers['Relu2'] = Relu()
        self.layers['Affine3'] = Affine(self.params['W3'], self.params['b3'])

        self.lastLayer = SoftmaxWithLoss()

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

        return x

    # x : 입력 데이터, t : 정답 레이블
    def loss(self, x, t):
        y = self.predict(x)
        return self.lastLayer.forward(y, t)

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

        accuracy = np.sum(y == t) / float(x.shape[0])
        return accuracy

    # x : 입력 데이터, t : 정답 레이블
    def numerical_gradient(self, x, t):
```

```
        loss_W = lambda W: self.loss(x, t)

        grads = {}
        grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
        grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
        grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
        grads['b2'] = numerical_gradient(loss_W, self.params['b2'])
        grads['W3'] = numerical_gradient(loss_W, self.params['W3'])
        grads['b3'] = numerical_gradient(loss_W, self.params['b3'])

        return grads

    def gradient(self, x, t):
        # forward
        self.loss(x, t)

        # backward
        dout = 1
        dout = self.lastLayer.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['Affine1'].dW, self.layers['Affine1'].db
        grads['W2'], grads['b2'] = self.layers['Affine2'].dW, self.layers['Affine2'].db
        grads['W3'], grads['b3'] = self.layers['Affine3'].dW, self.layers['Affine3'].db
        return grads
# 데이터 읽기
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)

network = TwoLayerNet(input_size=784, hidden_size=100,hidden_size2=50, output_size=10)

import  pickle

with  open('d:\\two_layer_w.pkl', 'wb') as  f:
    pickle.dump( network.params,  f)

# 하이퍼파라미터
iters_num = 10000  # 반복 횟수를 적절히 설정한다.
train_size = x_train.shape[0] # 60000 개
batch_size = 100  # 미니배치 크기
learning_rate = 0.1

train_loss_list = []
train_acc_list = []
test_acc_list = []

# 1에폭당 반복 수
iter_per_epoch = max(train_size / batch_size, 1)
print(iter_per_epoch) # 600

for i in range(iters_num): # 10000
    # 미니배치 획득  # 랜덤으로 100개씩 뽑아서 10000번을 수행하니까 백만번
    batch_mask = np.random.choice(train_size, batch_size) # 100개 씩 뽑아서 10000번 백만번
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]

    # 기울기 계산
    #grad = network.numerical_gradient(x_batch, t_batch)

    grad = network.gradient(x_batch, t_batch)

    # 매개변수 갱신
    for key in ('W1', 'b1', 'W2', 'b2','W3','b3'):
        network.params[key] -= learning_rate * grad[key]

    # 학습 경과 기록
    loss = network.loss(x_batch, t_batch)
    train_loss_list.append(loss) # cost 가 점점 줄어드는것을 보려고
```

```
    # 1에폭당 정확도 계산 # 여기는 훈련이 아니라 1에폭 되었을때 정확도만 체크

    if i % iter_per_epoch == 0: # 600 번마다 정확도 쌓는다.
        print(x_train.shape) # 60000,784
        train_acc = network.accuracy(x_train, t_train)
        test_acc = network.accuracy(x_test, t_test)
        train_acc_list.append(train_acc) # 10000/600 개  16개 # 정확도가 점점 올라감
        test_acc_list.append(test_acc)  # 10000/600 개 16개 # 정확도가 점점 올라감
        print("train acc, test acc | " + str(train_acc) + ", " + str(test_acc))


# 그래프 그리기
markers = {'train': 'o', 'test': 's'}
x = np.arange(len(train_acc_list))
plt.plot(x, train_acc_list, label='train acc')
plt.plot(x, test_acc_list, label='test acc', linestyle='--')
plt.xlabel("epochs")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
plt.legend(loc='lower right')
plt.show()
```



■ 텐써플로우 2.0 keras 사용하기

1. 가상환경 만들기

```
conda create -n keras_study

conda env  list

activate  keras_study

conda  install  ipykernel

python -m ipykernel install --user --name keras_study --display-name "keras_study"

jupyter kernelspec list
```

2. keras_study 가상환경에 tensorlfow 설치

```
activate keras_study

pip install tensorflow-cpu
```

3. keras_study 가상환경에 spyder 설치

```
conda install spyder
```

4. keras_study 가상환경에 jupyter notebook 설치

```
conda install jupyter notebook
```

5. keras_study 가상환경에 sklearn 설치

```
pip install sklearn
```

6. keras_study 가상환경에 matplotlib 설치

```
pip install matplotlib
```

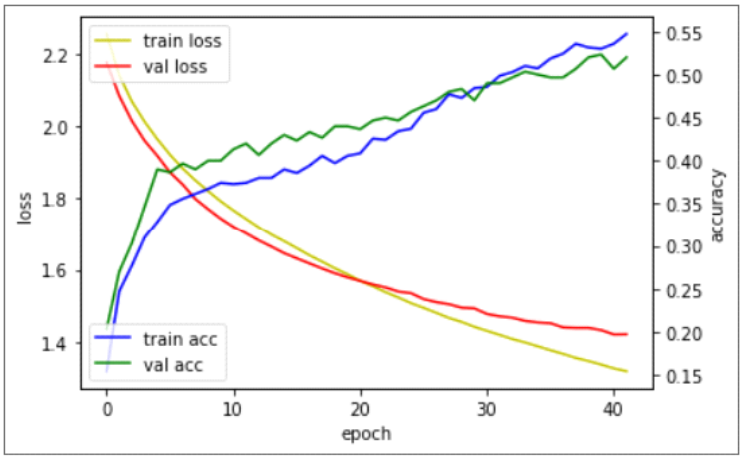
keras 의 EarlyStopping 기능

```
from tensorflow.keras.datasets import mnist

# 텐서플로우 저장소에서 데이터를 다운로드 받습니다.
(x_train, y_train), (x_test, y_test) = mnist.load_data(path='mnist.npz')

from sklearn.model_selection import train_test_split

# 훈련/검증 데이터를 얻기 위해 0.7/0.3의 비율로 분리합니다.
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size = 0.3, random_state = 777)

num_x_train = x_train.shape[0]
num_x_val = x_val.shape[0]
num_x_test = x_test.shape[0]


# 모델의 입력으로 사용하기 위한 전처리 과정입니다.
x_train = (x_train.reshape(-1, 28, 28, 1)) / 255
x_val = (x_val.reshape(-1, 28, 28, 1)) / 255
x_test = (x_test.reshape(-1, 28, 28, 1)) / 255

from tensorflow.keras.utils import to_categorical

# 각 데이터의 레이블을 범주형 형태로 변경합니다.
y_train = to_categorical(y_train)
y_val = to_categorical(y_val)
y_test = to_categorical(y_test)

from tensorflow.keras.models import Model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D, Dense
from tensorflow.keras.layers import Input

# 함수형 API는 Input()을 통해 입력값의 형태를 정의해주어야 합니다.
inputs = Input(shape = (28, 28, 1))
x = Conv2D(32, (3, 3), activation = 'relu')(inputs)
x = Conv2D(32, (3, 3), activation = 'relu')(x)
x = MaxPooling2D(strides = 2)(x)
x = GlobalAveragePooling2D()(x)
x = Dense(10, activation = 'softmax')(x)

# 위에서 정의한 층을 포함하고 있는 모델을 생성합니다.
model = Model(inputs = inputs, outputs = x)

model.compile(optimizer = 'adam',
              loss = 'categorical_crossentropy',
              metrics = ['acc'])

from tensorflow.keras.callbacks import EarlyStopping
```

```
# 콜백을 정의합니다.

callbacks = [EarlyStopping(monitor = 'val_loss', patience = 3, verbose = 1)]

# callbacks 인자를 통해 정의한 콜백을 전달합니다.

model.fit(x_train, y_train,
          batch_size = 32,
          validation_data = (x_val, y_val),
          epochs = 30,
          callbacks = callbacks)
```

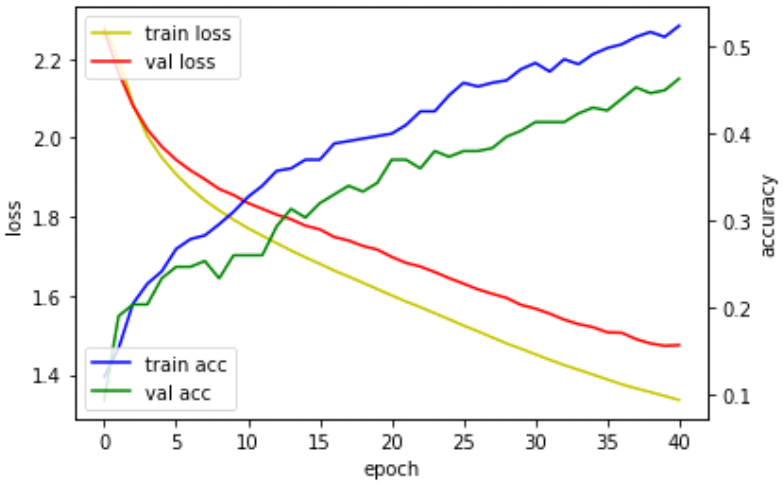
케라스 얼리스탑 다른 예제

https://ssongnote.tistory.com/11




```
# 0. 사용할 패키지 불러오기
from tensorflow.python.keras.utils import np_utils
from tensorflow.python.keras.datasets import mnist
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dense,Activation
import numpy as np

np.random.seed(3)

# 1. 데이터셋 준비하기


# Training set과 Test set 불러오기
(x_train,y_train),(x_test,y_test) = mnist.load_data()
print("x_train : ",x_train.shape)
print("y_train : ",y_train.shape)
print("x_test : ",x_test.shape)
print("y_test : ",y_test.shape)


# Training set과 Validation set 분리
x_val = x_train[50000:]
y_val = y_train[50000:]
x_train = x_train[:50000]
y_train = y_train[:50000]

# Dataset Preprocessing
x_train = x_train.reshape(50000,784).astype('float32') / 255.0
x_val = x_val.reshape(10000,784).astype('float32') / 255.0
x_test = x_test.reshape(10000,784).astype('float32') / 255.0


# Training set과 Validation set 고르기
train_rand_idxs = np.random.choice(50000,700)
val_rand_idxs = np.random.choice(10000,300)
x_train = x_train[train_rand_idxs]
y_train = y_train[train_rand_idxs]
x_val = x_val[val_rand_idxs]
y_val = y_val[val_rand_idxs]
```

```
# Label data one-hot encoding
y_train = np_utils.to_categorical(y_train)
y_val = np_utils.to_categorical(y_val)
y_test = np_utils.to_categorical(y_test)


# 2. 모델 구성하기
model = Sequential()
model.add(Dense(units=2,input_dim=28*28,activation='relu'))
model.add(Dense(units=10,activation='softmax'))


# 3. 모델 학습과정 설정하기
model.compile(loss='categorical_crossentropy',optimizer = 'sgd',metrics=['accuracy'])


# 4. 모델 학습시키기
from tensorflow.python.keras.callbacks import EarlyStopping
early_stopping = EarlyStopping()
hist = model.fit(x_train,y_train,epochs=1000, batch_size=10, validation_data=(x_val,y_val),callbacks=[early_stopping])


# 5. 학습 과정 살펴보기
%matplotlib inline
import matplotlib.pyplot as plt

fig, loss_ax = plt.subplots()

acc_ax = loss_ax.twinx()

loss_ax.plot(hist.history['loss'],'y',label='train loss')
loss_ax.plot(hist.history['val_loss'],'r',label='val loss')
acc_ax.plot(hist.history['accuracy'],'b',label='train acc')
acc_ax.plot(hist.history['val_accuracy'],'g',label='val acc')

loss_ax.set_xlabel('epoch')
loss_ax.set_ylabel('loss')
acc_ax.set_ylabel('accuracy')

loss_ax.legend(loc='upper left')
acc_ax.legend(loc='lower left')
plt.show()


# 6. 모델 평가하기
loss_and_metrics = model.evaluate(x_test,y_test,batch_size=32)

print('##### Test Result #####')
print('loss : ',str(loss_and_metrics[0]))
print('Accuracy : ',str(loss_and_metrics[1]))
```




```
Epoch 41/1000
70/70 [==============================] - 0s 1ms/step - loss: 1.3364 - accuracy: 0.5243 - val_loss: 1.4753 -
val_accuracy: 0.4633
```

```
313/313 [==============================] - 0s 863us/step - loss: 1.5017 - accuracy: 0.4721
##### Test Result #####
loss :  1.5016628503799438
Accuracy :  0.47209998965263367
```

7장. 합성곱 신경망 (CNN)

CNN (Convolution Neural Network)

합성곱 신경망이란?
" Convolution 층과 pooling 층을 포함하는 신경망 "

기존 신경망과의 차이는?
- 기존 방법 : Affine --> ReLU (완전 연결 계층)
- CNN : conv --> ReLU --> pooling --> 완전 연결 계층

CNN을 이용하지 않은 기존 층의 문제점?

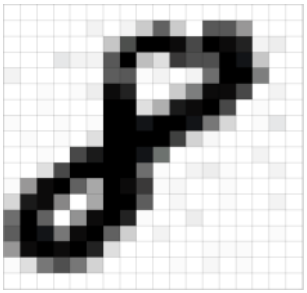
" 이미지의 형상이 무시 된다. "

필기체(이미지) --> 28x28의 1차원 데이터로 변경해서 784개의 데이터를 첫 Affine 계층에 입력한게 기존 방법이다.
---> 문제점?
- 형상을 무시하고 모든 입력 데이터를 동등한 뉴런으로 취급하기 때문에 이미지가 갖는 본질적인 패턴을 읽지 못한다.
따라서 합성곱이 필요하다

- 합성곱 계층이 이 문제를 해결하는데 어떻게 해결하냐면
원본이미지를 가지고 여러 개의 feature map을 만들어서 완전 연결 계층에 입력한다.

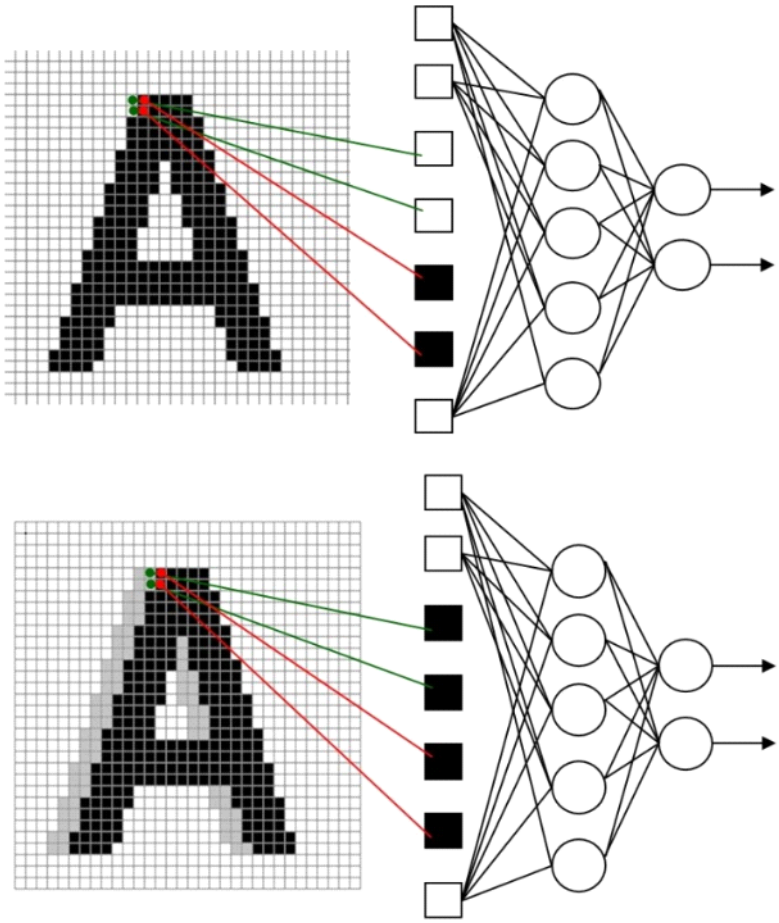
■ CNN 을 이해하기 위한 그림들 순서대로 나열

1. 아래의 숫자 8 필기체 784개의 픽셀값이 신경망에 입력이 되는데



2. CNN 을 이용하지 않았을때의 문제점은 ?

전체 글자에서 단지 2 픽셀값만 달라지거나 2픽셀씩 이동만 하더라도
새로운 학습 데이터로 처리를 해줘야 하는 문제점이 있다.

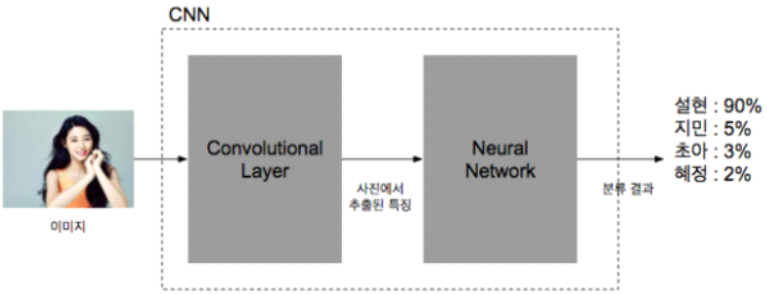


또한 글자의 크기가 달라지거나,
글자가 회전되거나,
글자에 변형(distortion)이 조금만 생기더라도
새로운 학습 데이터를 넣어주지 않으면 좋은 결과를 기대하기 어렵다.

결과적으로 기존 multi-layered neural network는
글자의 topology는 고려하지 않고,
말 그대로 raw data에 대해 직접적으로 처리를 하기 때문에
엄청나게 많은 학습 데이터를 필요로 하고,
또한 거기에 따른 학습 시간을 대가로 지불해야 하는 문제점이 있다.

3. 그래서 CNN 을 사용하게 되면 ?

CNN은 전통적인 뉴럴 네트워크 앞에 여러 계층의 컨볼루셔널 계층을 붙인 모양이 되는데, 그 이유는 다음과 같다. CNN은 앞의 컨볼루셔널 계층을 통해서 입력 받은 이미지에 대한 특징(Feature)를 추출하게 되고, 이렇게 추출된 특징을 기반으로 기존의 뉴럴 네트워크를 이용하여 분류를 해내게 된다.



CNN의 구조

CNN의 과정은 아래 그림과 같이 나타낼 수 있다.

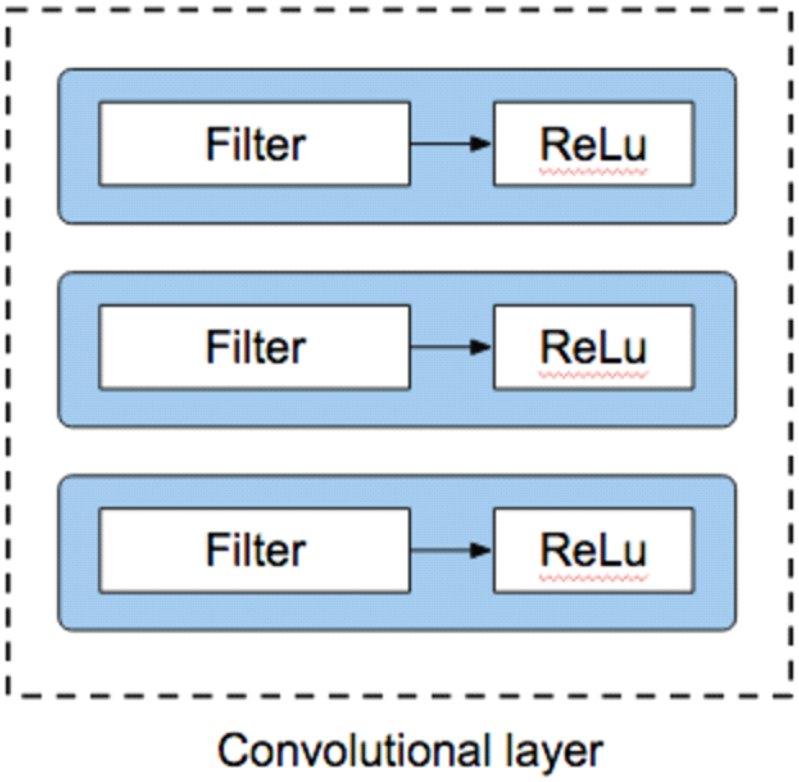
CNN의 과정은 크게 보면 다음과 같은 3단계 과정으로 이루어 진다.

1. 특징을 추출하기 위한 단계
2. topology 변화에 영향을 받지 않도록 해주는 단계
3. 분류기 단계

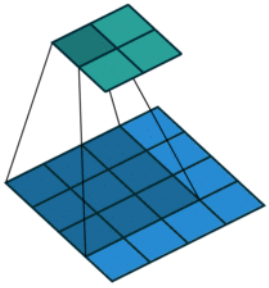
4. 그럼 위의 1번에서 특징을 어떻게 추출하는지 ?

컨볼루셔널 레이어는 앞에서 설명 했듯이 입력 데이타로 부터 특징을 추출하는 역할을 한다.

컨볼루셔널 레이어는 특징을 추출하는 기능을 하는 필터(Filter)와,
이 필터의 값을 비선형 값으로 바꾸어 주는 액티베이션 함수(Activiation 함수)로 이루어진다

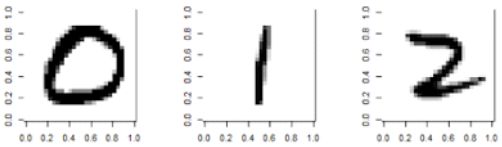


■ **Striding 1 의 Convolution 연산 (합성곱 연산)**

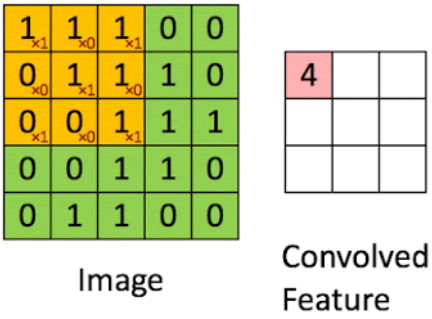


컨볼루션 연산을 통해 이미지의 특징들을 추출하는 역활을 수행한다.

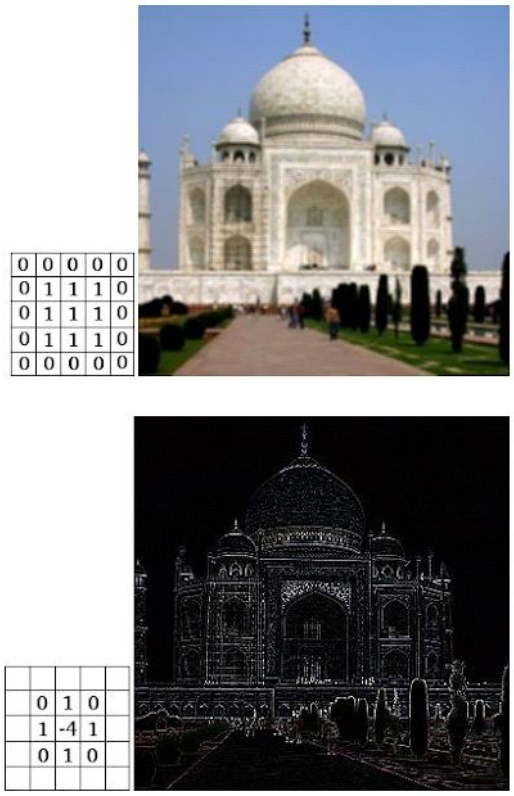
MNIST는 28*28의 행렬형태로 표현되는 gray scale의 손글씨 예제들입니다.
각각의 픽셀은 0~255의 숫자를 갖죠. MLP는 기본적으로 벡터 인풋을 받기 때문에 (뉴럴 넷에서 인풋 layer가 1열로 길게 늘어선거 보이시죠?)
28*28 행렬을 **784 짜리 긴 벡터**로 먼저 바꿉니다.
그런 뒤 이러한 데이터들을 MLP에 통과시키고 (forward propagation) 에러 값을 계산한 후
W를 순차적으로 업데이트(**back-propagation**)하는 전략을 취하죠.



반변 CNN은 다릅니다. 28*28 형태를 그대로 이용하죠.
그리고 어떻게 하냐면 (예를 들어) 3*3 짜리 커널 형렬로 한번 훑어서(sweep) 이 데이터를 다른 형태로 변형시키는 것입니다.
이 과정을 convolution이라고 하는데, 사실 그렇게 어려운 연산 아닙니다. 그냥 윈도우에 해당하는 원소들끼리 각각 곱해서 더해주는 것이지요.



이렇게 커널 윈도우로 이미지를 훑으면 어떻게 되느냐... 뭔가 변하겠지요? 원본에 이상한 짓을 해놨으니 말이죠 ㅎㅎ 바로 다음과 같이 변하게 된답니다.



Convolution의 결과. 왼쪽이 커널행렬의 모습이다. (From GIMP)

대충 얘기하자면 CNN에서 convolution이란 **주위값들을 반영해 중앙의 값을 변화**시키는 것이라고 할 수 있죠.
어떤 쪽으로 변화시키냐고요? 여기서 바로 CNN의 핵심이 있는 것입니다. CNN은 **목적하는 작업의 성공률이 높도록** 이미지를 변형해 갑니다.
예를 들어 classification이라면 classification이 더욱 잘되도록 원본으로부터 2차 이미지들을 형성하는 것이지요.

용어설명을 하고 넘어가자면, 이렇게 만들어진 왜곡된 이미지들을 **feature map**이라고 합니다.
결국 CNN은 원본 입력을 받으면 다양한 커널행렬을 이용해 여러개의 feature map들을 만든 후, 이를 토대로 classification을 진행하는 것이지요.
그러니 원본 딱 하나만 가지고 학습을 했을 때보단, 다양한 feature map을 가지고 학습을 하는게 더 유리할 수 있겠죠?

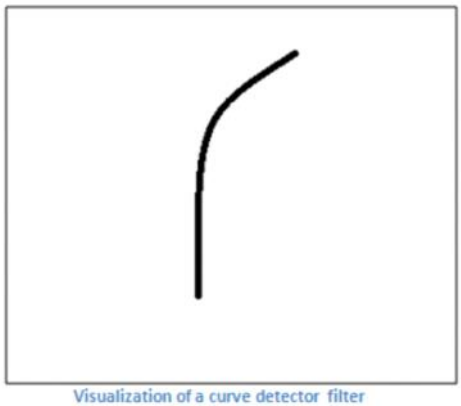
또 중요한 점은 이 **커널행렬이 학습 가능**하다는 것입니다. MLP에서는 Wx+b에서 W가 학습 가능했자나요.
사실 convolution도 곱하기와 더하기로 이루어져 있으니 이 연산을 back propagation을 통해 학습하지 못할 이유가 없겠죠.
이 연산을 예를 들어 W*X 라고 표현한다면 (W는 커널 행렬, X는 적용되는 해당 이미지) 여기서의 W도 학습 가능하다는 이야기입니다. 짱이죠? ㅎㅎㅎ

5. 여기서 필터란 무엇이냐면 ?

필터는 그 특징이 데이타에 있는지 없는지를 검출해주는 함수이다.
예를 들어 아래와 같이 곡선을 검출해주는 필터가 있다고 하자.

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



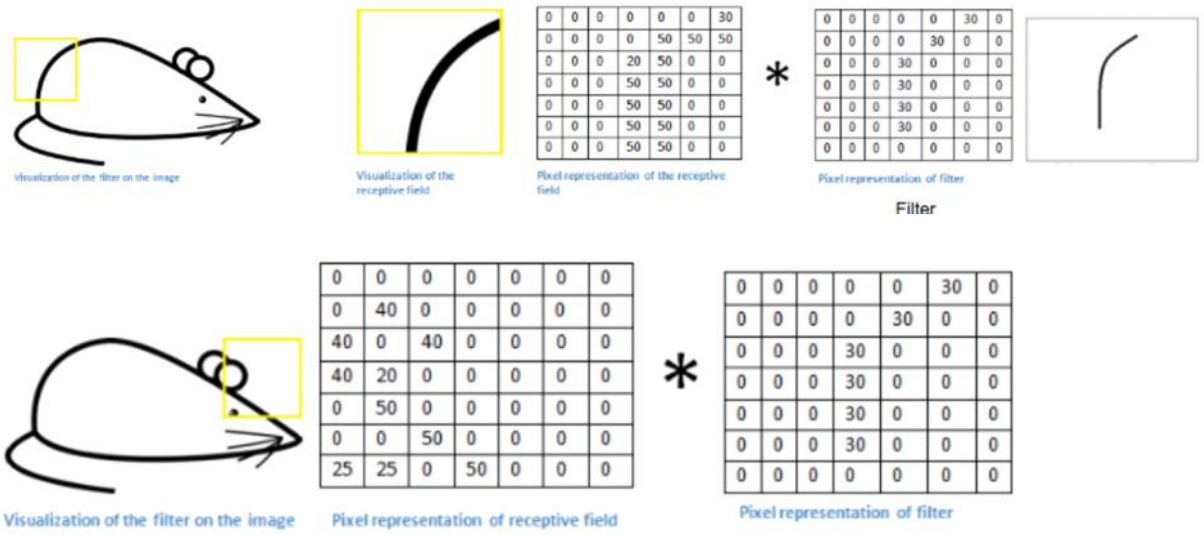
Visualization of a curve detector filter

결과 값이 0에 수렴하게 나온다.

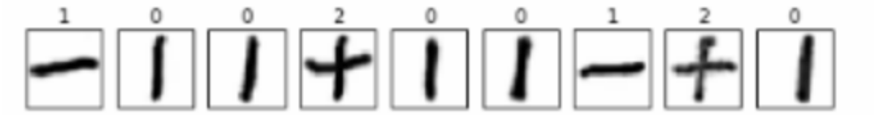
즉 필터는 입력받은 데이타에서 그 특성을 가지고 있으면 결과 값이 큰값이 나오고,
특성을 가지고 있지 않으면 결과 값이 0에 가까운 값이 나오게 되서 데이타가 그 특성을 가지고 있는지 없는지 여부를 알 수 있게 해준다.

다중 필터의 적용
입력값에는 여러가지 특징이 있기 때문에 하나의 필터가 아닌 여러개의 다중 필터를 같이 적용하게 된다.

다음과 같이 |,+,- 모양을 가지고 있는 데이타가 있다고 하자



필터들

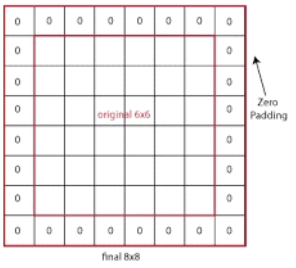


6. 패딩 이란 ?

" 합성곱 연산을 수행하기전, 입력 데이터 주변을 특정값으로 채워 늘리는것을 말한다."

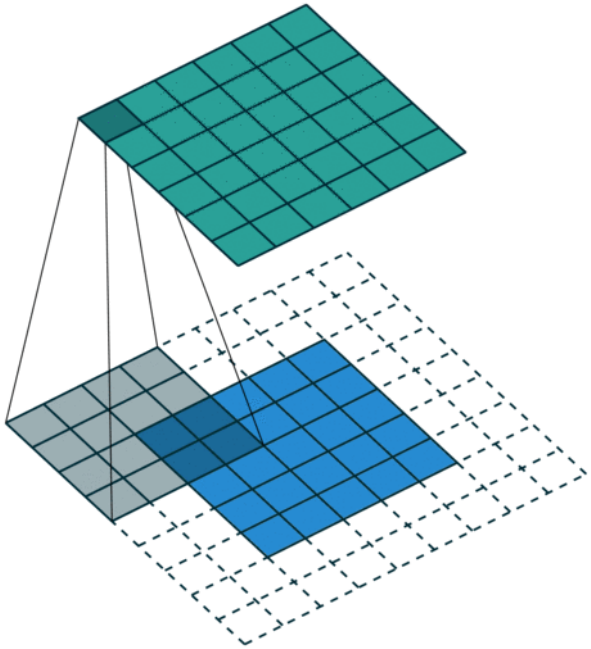
- 패딩이 필요한 이유 ?

패딩을 하지 않을 경우 data 의 공간크기는 합성곱 계층을 지날때 마다 작아지게 되므로 가장 자리 정보들이 사라지게 되는 문제가 발생하기 때문에 패딩을 사용한다.



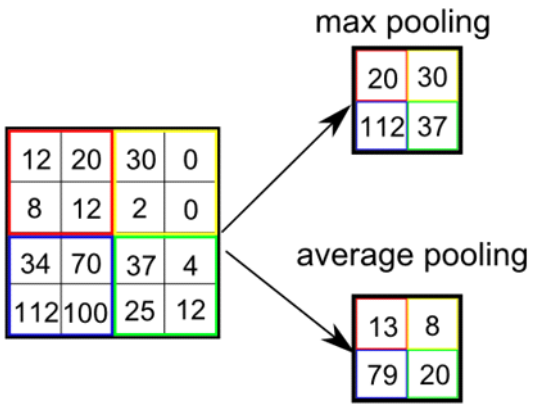
0	0	0	0	0	0
0	35	19	25	6	0
0	13	22	16	53	0
0	4	3	7	10	0
0	9	8	1	3	0
0	0	0	0	0	0

- 제로 패딩후 다시 합성곱 연산



7. 풀링(Pooling) 이란 ?

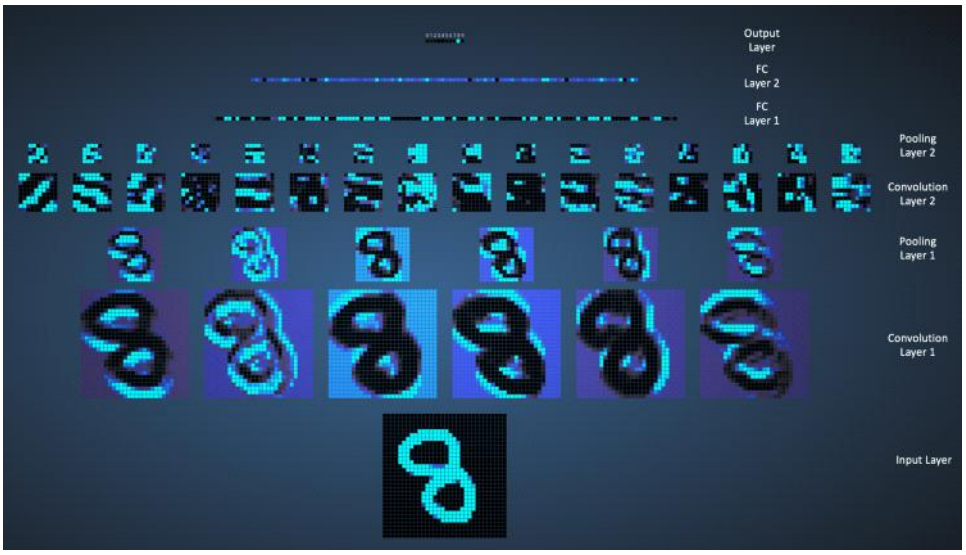
" 말그대로 출력 값에서 일부분만을 취하는 기능"



CNN은 **pooling**이라는 단계를 한번 더 거치죠.
이것은 단순히 사이즈를 줄이는 과정이라고 생각하면 좋을 것 같은데요,
convolution이 이렇게저렇게 망쳐놓은(?) 그림들을 각각 부분에서 대표들을 뽑아 사이즈가 작은 이미지를 만드는 것이지요.
마치 사진을 축소하면 해상도가 좋아지는 듯한 효과와 비슷하달까요? ㅎㅎ

그런데 놀랍게도 단순한 '골라내기' 과정인 pooling은 성능향상에 매우 큰 기여를 합니다.
여기엔 여러가지 이유가 있지만 (미세한 translation에 대해 invariant한 feature를 제공한다, noise의 역할을 상쇄시킨다 등등)
하지만 제겐 아직도 이 부분이 매우 많은 발전 가능성이 있어 보이네요. 단순한게 이렇게 효과가 좋다면, 좀더 머리를 쓴다면 얼마나 더 나은 효과를 볼 수 있을까요? >.<

8. 그럼 총정리해서 Mnist 데이터가 CNN 신경망으로 들어온다면 ?



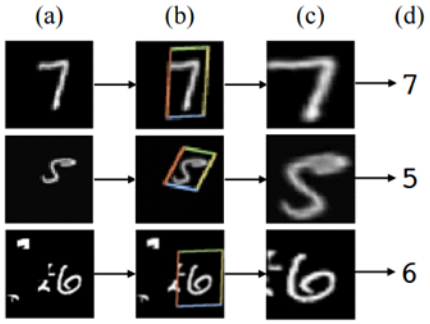


Figure 1: The result of using a spatial transformer as the first layer of a fully-connected network trained for distorted MNIST digit classification. (a) The input to the spatial transformer network is an image of an MNIST digit that is distorted with random translation, scale, rotation, and clutter. (b) The localisation network of the spatial transformer predicts a transformation to apply to the input image. (c) The output of the spatial transformer, after applying the transformation. (d) The classification prediction produced by the subsequent fully-connected network on the output of the spatial transformer. The spatial transformer network (a CNN including a spatial transformer module) is trained end-to-end with only class labels – no knowledge of the groundtruth transformations is given to the system.

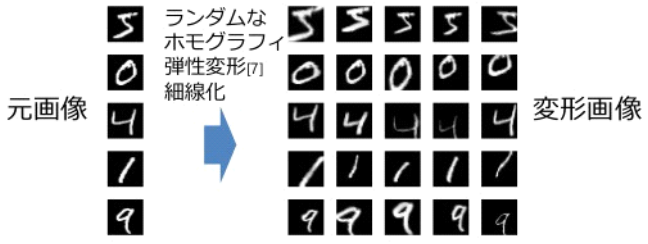
Great overview of the function of a Spatial Tranfomer module

データ拡張とは？

データ拡張とは、データに適切な画像処理を施して
学習サンプルを増やすこと

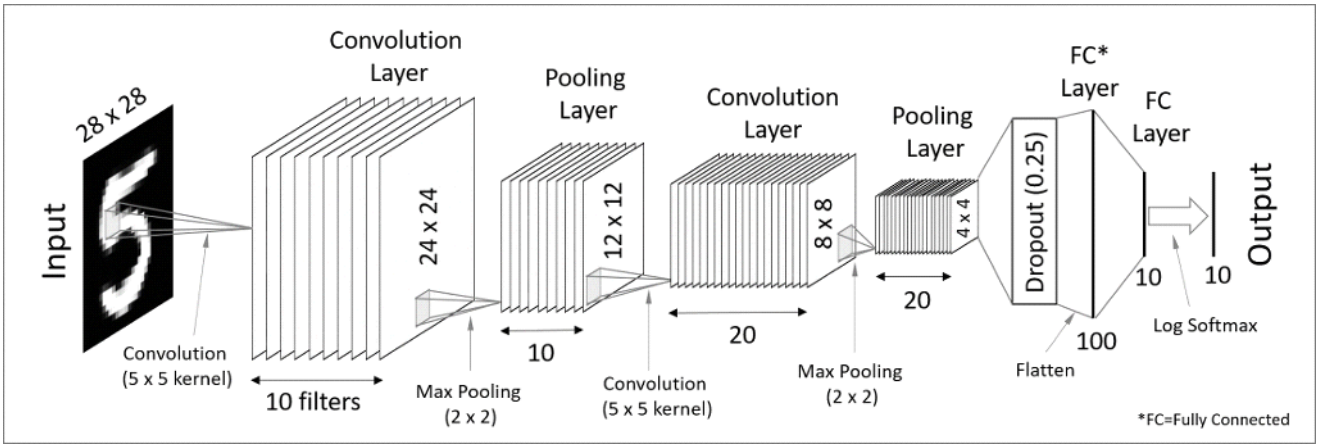
例）MNISTデータセット[6]

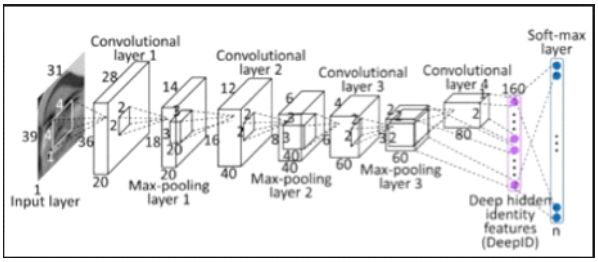
[6] Y. LeCun, "Gradient-based learning applied to document recognition" Proc IEEE, 86(11), 1998.



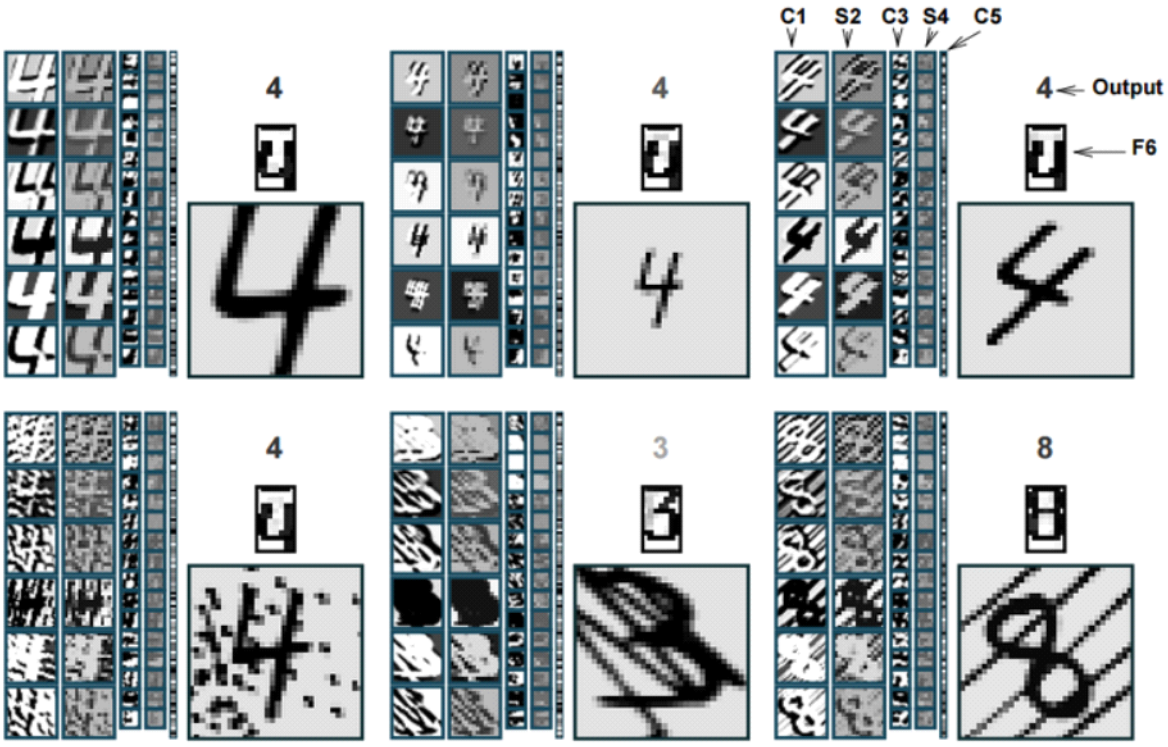
[7] P. Y. Simard, et al, "Best practices for convolutional neural networks applied to visual document Analysis", ICDAR, 2003.

 26/33 ITLAB





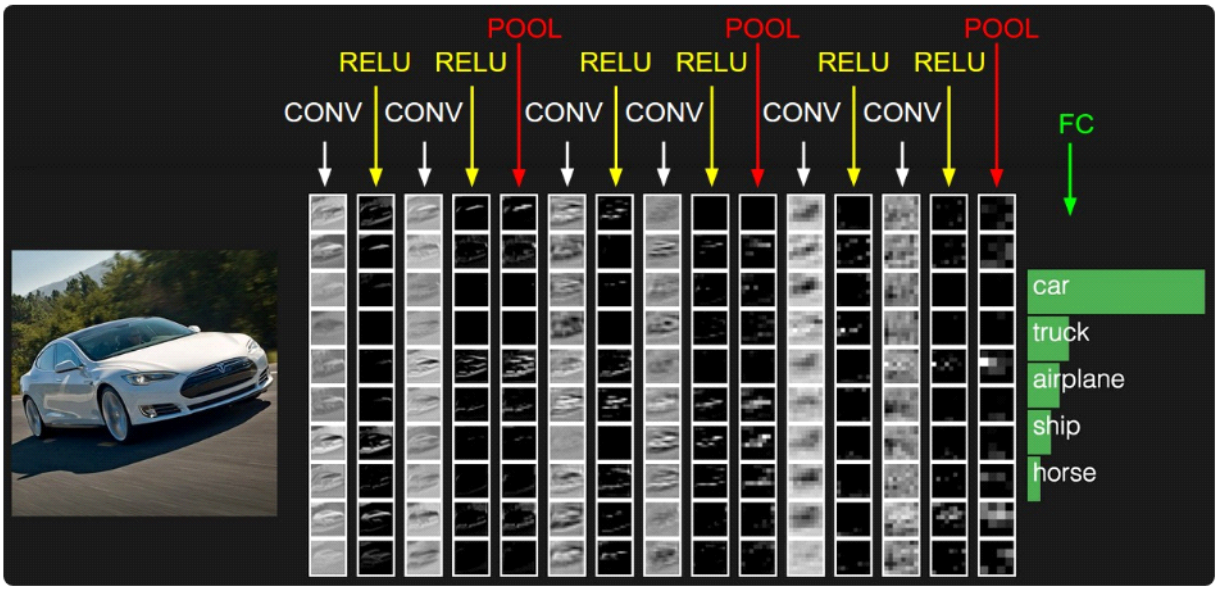
9. 뉴럴 네트워크만 사용했을때보다 CNN 을 사용했을때 더 큰 효과를 보는 이유는?



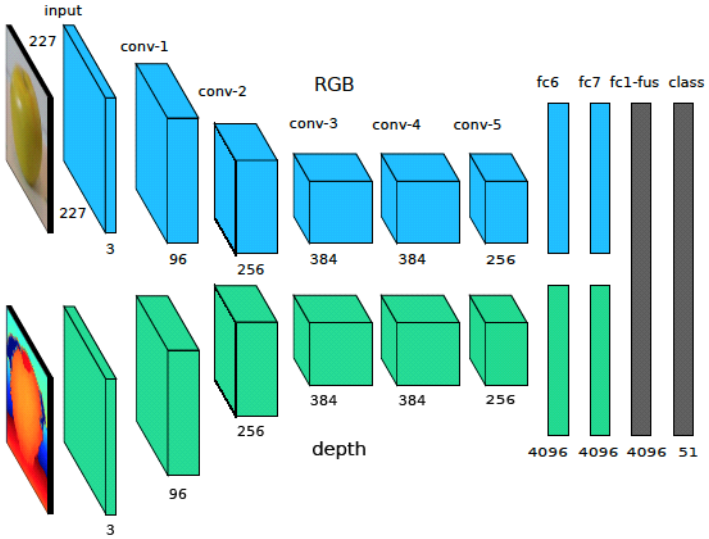
위 그림에서 C1과 C3는 각각의 convolution을 거쳐서 얻어진 feature map 영상을 보여준다.
C1 단계에서는 6개의 다른 feature map을 얻을 수 있도록 서로 다른 convolution kernel이 적용이 되었고,
C3 단계에서는 16개의 다른 feature map을 얻을 수 있는 서로 다른 convolution이 적용 되었다.
C1과 C2 convolution 단계 및 S2와 S4 pooling 단계를 거치면서,
topology 변화에 강인한 특성을 갖게 되었으며, 결과적으로 인식 능력이 크게 개선되었다.

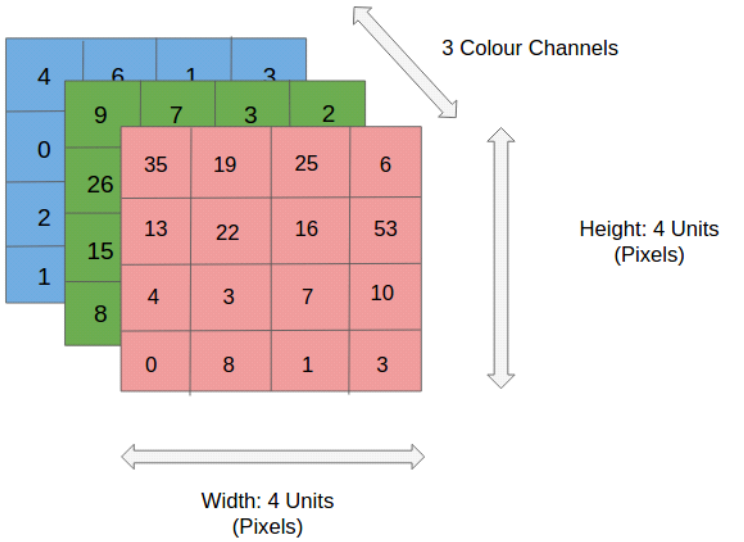
위 그림을 보면, 상단의 비교적 깨끗한 영상뿐만 아니라 하단처럼 상당한 잡음이 있는 경우에 대해서도,
각각의 convolution kernel이 각각 다른 특징을 추출해내는 것을 알 수가 있다.

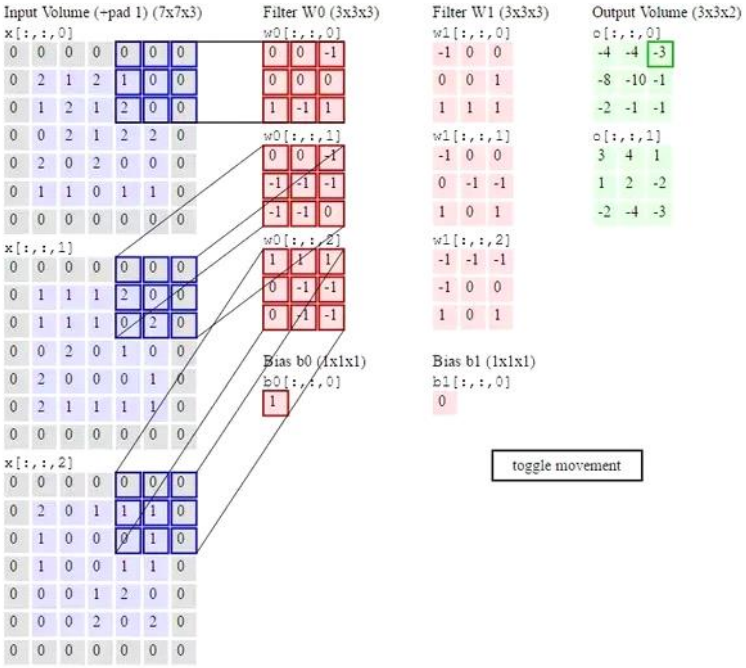
10. Mnist 말고 다른 이미지로 다시 총정리 해보면 ?



11. 흑백이 아니라 RGB 라면 ?







참고 사이트들

http://bcho.tistory.com/1149

https://laonple.blog.me/220587920012

합성곱 연산을 컴퓨터로 구현하는 방법

합성곱 하는 그림 gif

원본이미지 필터 = feature map

```
1 2 3 0       2 0 1       15 16
0 1 2 3   ◎   0 1 2        6 17
3 0 1 2       1 0 2
2 3 0 2
```

문제93. 아래의 두 행렬을 만들고 합성곱한 결과인 15를 파이썬으로 출력하시오

```
import numpy as np
```

```
a = np.array([[1,2,3],[0,1,2],[3,0,1]])
filt = np.array([[2,0,1],[0,1,2],[1,0,2]])

np.sum(a * filt)
```

```
15
```

문제94. 아래의 4x4 행렬에서 아래의 3x3 행렬만 추출하시오

```
import numpy as np

a = np.array([[1,2,3,0],[0,1,2,3],[3,0,1,2],[2,3,0,2]])
a[:3,:3]
```

```
array([[1, 2, 3],
       [0, 1, 2],
       [3, 0, 1]])
```

문제95. 아래의 합성곱을 파이썬으로 구현하시오

원본이미지 필터 = feature map

1 2 3 0 2 0 1 15 16
0 1 2 3 ◎ 0 1 2 6 17
3 0 1 2 1 0 2
2 3 0 2

```
import numpy as np

a = np.array([[1,2,3,0],[0,1,2,3],[3,0,1,2],[2,3,0,2]])
filt = np.array([[2,0,1],[0,1,2],[1,0,2]])

f_map = []
for i in range(a.shape[0] - filt.shape[0] + 1):
    for k in range(a.shape[1] - filt.shape[1] + 1):
        f_map.append(np.sum(a[i:i+3, k:k+3] * filt))

f_map = np.array(f_map).reshape(2,2)
print(f_map)
```

```
[[15 16]
 [ 6 17]]
```

```
print('합성곱 연산')

import numpy as np

image=np.array([1,2,3,0,0,1,2,3,3,0,1,2,2,3,0,2]).reshape(4,4)
filter=np.array([2,0,1,0,1,2,1,0,2]).reshape(3,3)
feature=np.zeros([2,2])
# print(image)
h,w = image.shape
fh,fw=filter.shape
for i in range(fh-1):
    for j in range(fw-1):
        feature[i,j]=np.sum(image[i:fw+i,j:fh+j]*filter)
print(feature)
```

```
feature[0,0] = np.sum( image[ 0:3, 0:3] * filter )
feature[0,1] = np.sum( image[ 0:3, 1:4] * filter )
feature[1,0] = np.sum( image[ 1:4, 0:3] * filter )
feature[1,1] = np.sum( image[ 1:4, 1:4] * filter )
print(feature[0,1])
```

패딩 - p 232

원본이미지 필터 = feature map

1 2 3 0 2 0 1 15 16
0 1 2 3 ◎ 0 1 2 6 17
3 0 1 2 1 0 2
2 3 0 2

4x4 3x3 2x2

conv --> relu --> pooling --> conv --> relu --> pooling --> conv --> ....

conv : 이미지의 특징을 잡아내는 역할
feature map을 필터의 개수 만큼 만들면서 이미지의 특징을 잡는다.

pooling : 이미지가 선명해지게 하는 역할

% 면접 문제 %

6. 패딩을 하는 이유?

답 : 패딩을 하지 않을 경우 data의 공간 크기는 합성곱 계층이 지날때마다 작아지게 되므로 가장자리 정보들이 사라지게 되는 문제가 발생하기 때문에 패딩을 해야합니다.

패딩이란?

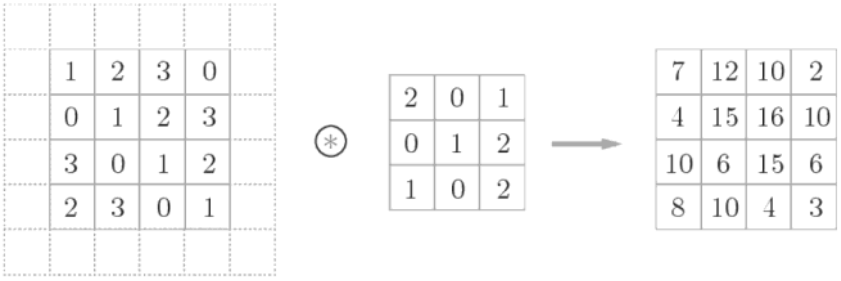
합성곱 연산을 수행하기 전에 입력 데이터 주변을 특정값으로 채워 늘리는것을 패딩이라고 한다.

예제1 : 아래의 2x2 행렬을 제로패딩 1해서 4x4 행렬로 만드시오

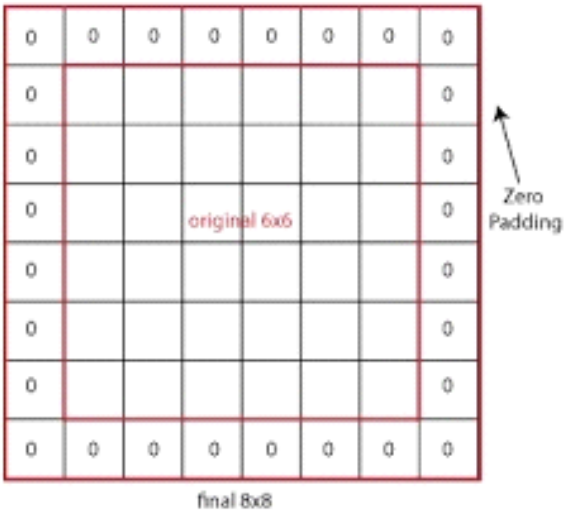
18 19 0 0 0 0
9 18 ---> 0 18 19 0
0 9 18 0
0 0 0 0

```
import numpy as np

result = np.array([[18,19],[9,18]])
result_pad = np.pad(result, pad_width=1, mode='constant', constant_values=0)
print(result_pad)
```

```
[[ 0  0  0  0]
 [ 0 18 19  0]
 [ 0  9 18  0]
 [ 0  0  0  0]]
```



(4, 4) (3, 3) (4, 4)
입력 데이터(패딩 : 1) 필터 출력 데이터



0	0	0	0	0	0
0	35	19	25	6	0
0	13	22	16	53	0
0	4	3	7	10	0
0	9	8	1	3	0
0	0	0	0	0	0

제로 패딩후 다시 합성곱 연산

예제2. 아래의 입력 데이터를 1제로 패딩하고 3x 3 필터로 합성곱해서 feature map을 출력하면 feature map의 행렬은 몇행 몇열인지 출력하시오

원본이미지 필터 = feature map

1 2 3 0 2 0 1 15 16
0 1 2 3 ◎ 0 1 2 6 17
3 0 1 2 1 0 2
2 3 0 2

4x4 3x3 2x2

```
import numpy as np

a = np.array([[1,2,3,0],[0,1,2,3],[3,0,1,2],[2,3,0,2]])
filt = np.array([[2,0,1],[0,1,2],[1,0,2]])

a_pad = np.pad(a, pad_width=1, mode='constant', constant_values=0)

f_map = []
for i in range(a_pad.shape[0] - filt.shape[0] + 1):
    for k in range(a_pad.shape[1] - filt.shape[1] + 1):
        f_map.append(np.sum(a_pad[i:i+3, k:k+3] * filt))

i , w = a_pad.shape[0] - filt.shape[0] + 1, a_pad.shape[1] - filt.shape[1] + 1

f_map = np.array(f_map).reshape(i, w)
print(f_map.shape)
```

```
(4, 4)
[[ 7 12 10  2]
 [ 4 15 16 10]
 [10  6 17  6]
 [ 8 10  6  4]]
```

위와 같이 입력 데이터가 4x4 행렬에 3x3 필터를 합성곱 했을 때 출력 데이터가 4x4 행렬이 되려면 입력 데이터를 패딩 몇으로 했을때 출력 데이터가 4x4 행렬로 출력될 수 있는가?

답 : 1

공식: 책 p234 식 7.1
< e 7.1>

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

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

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

P : 패딩
OH : 출력데이터의 높이(행)
S : 스트라이드
H : 입력 데이터의 높이(행)
FH : 필터의 높이(행)

입력 이미지 --> 컨볼루션 층 --> 행/열이 같은 입력 이미지

padding = same (텐서 플로우)

3차원 데이터의 합성곱 연산 - p 235

이미지의 색깔이 보통 흑백이 아니라 RGB 컬러 이므로
RGB(Red, Green, Blue) 컬러에 합성곱을 해야한다.

예제1. 레드벨벳의 아이린 사진을 3차원 행렬로 변환하시오

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mgimg

img = Image.open('아이린.jpg')
img_pixel = np.array(img)
plt.imshow(img_pixel)  #  (500, 500, 3)  =  (가로, 세로, 색조(RGB))
print(img_pixel)
```

```
[[[ 79  96 104]
  [113 118 138]
  [147 137 172]
  ...
  [ 50  72  51]
  [ 50  72  51]
  [ 50  72  51]]

 [[101 118 128]
  [137 142 162]
  [169 159 194]
  ...
  [ 51  70  50]
  [ 51  70  50]
  [ 50  69  49]]
```

[RED GREEN BLUE] 행렬로 표시
픽셀 하나의 숫자가 0 - 255 사이로 되어있고 숫자가 클수록 밝은색이다.
(500, 500, 3) --> (가로, 세로, 색상[0 : RED, 1 : GREEN, 2 : BLUE])

문제96. 아이린 사진에서 red부분의 행렬만 출하시오

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mgimg

img = Image.open('아이린.jpg')
img_pixel = np.array(img)
plt.imshow(img_pixel)  #  (500, 500, 3)  =  (가로, 세로, 색조(RGB))
print(img_pixel[:,:,0])
```

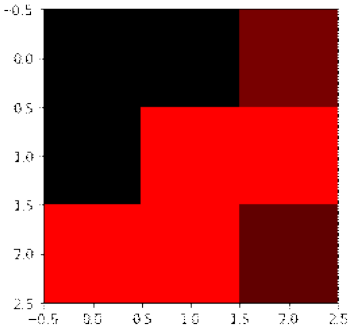
```
[[ 79 113 147 ...  50  50  50]
 [101 137 169 ...  51  51  50]
 [116 152 181 ...  49  49  49]
 ...
 [219 219 222 ... 163 173 184]
 [222 222 224 ... 163 174 184]
 [222 223 225 ... 165 176 184]]
```

문제97. 아이린 사진을 3차원 합성곱 하기 위한 필터를 3x3 RGB 필터로 생성하시오

```
filter = np.random.randn(1,3,3,3)
filter
```

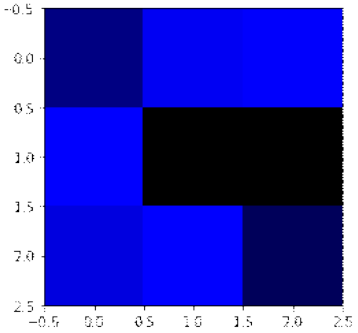
문제98. 위의 필터에서 red 부분 행렬만 추출하고 red 부분만 시각화 하시오

```
filter = np.random.randn(3,3,3)
filter[:,:,0]
filter[:,:,1] = 0
filter[:,:,2] = 0
plt.imshow(filter)
plt.show()
```



문제99. 이번에는 blue 부분만 가져와서 출력하시오

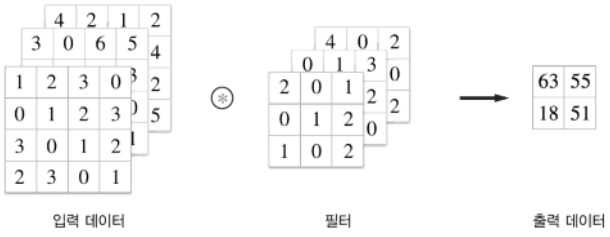
```
filter = np.random.randn(3,3,3)
filter[:,:,2]
filter[:,:,0] = 0
filter[:,:,1] = 0
plt.imshow(filter)
plt.show()
```



오늘의 마지막 문제

문제100. 아래의 행렬의 3차원 합성곱을 파이썬으로 구현하시오

(원본이미지 3차원, 필터도 3차원)




```
import numpy as np

x=np.array([[[1,2,3,0],
             [0,1,2,3],  # --> red 행렬
             [3,0,1,2],
             [2,3,0,1]],

            [[2,3,4,1],
             [1,2,3,4],  # --> green 행렬
             [4,1,2,3],
             [3,4,1,2]],

            [[3,4,5,2],  # --> blue 행렬
             [2,3,4,5],
             [5,2,3,4],
             [4,5,2,3]]])

f=np.array([[[2,0,1],
             [0,1,2],
```

```
                [1,0,2]],

                [[3,1,2],
                 [1,2,3],
                 [2,1,3]],

                 [[4,2,3],
                 [2,3,4],
                 [3,2,4]]])

xc, xh, xw = x.shape
fc, fh, fw = f.shape
feature = np.zeros([xh-fh +1, xw - fw + 1])

for i in range(xh-fh +1):   # 0, 1
    for j in range(xw - fw + 1): # 0, 1
        for k in range(x.shape[0]):  #  0, 1, 2
            feature[i][j] += np.sum(x[k, i:fh+i, j:fw+j] * f[k])

print(feature)
```

```
[[156. 162.]
 [126. 156.]]
```

다른사람 답 :

```
#%% 문제 100
def convolution(a,filter,stride=1):
    ah,aw=a.shape
    fh,fw=filter.shape

    res=[]
    if (a.shape[0]-filter.shape[0])%stride==0:
        for i in range(0,a.shape[0]-filter.shape[0]+1,stride):
            for j in range(0,a.shape[1]-filter.shape[1]+1,stride):
                res.append(np.sum(a[i:i+filter.shape[0],j:j+filter.shape[1]]*filter))

    oh=int((ah-fh)/stride)+1
    ow=int((aw-fw)/stride)+1

    return np.array(res).reshape(oh,ow)

zero=np.zeros(4).reshape(2,2)

for i in range(x.shape[0]):
    zero+=convolution(x[i],f[i])

print(zero)
```

```
c, h, w = x.shape
fc, fh, fw = f.shape

result = np.zeros((h - fh + 1, w - fw + 1))

for i in range(h - fh + 1):
    for j in range(w - fw + 1):
        result[i, j] = np.sum(x[:, i : i + fh, j : j + fw] * f)

print(result)
```