【要点まとめ】 次元間で繋がりのあるデータを扱うことができる。 畳み込みといえば画像を扱うイメージがあるが、画像以外にも適用することが可能である。

・LeNet 32*32サイズの画像を入力とする。 次元の繋がりを保ったまま特徴量を抽出していくことが可能。 後半の全結合層は、NNの考え方そのものと思ってよい。

フィルターとは全結合層でいうW（重み）である。 畳み込み演算を繰り返すことで画像が小さくなりすぎるのを防止するために、パディング（数値埋め）の概念がある。 ※パディングはフィルターを通す前に実施する。

プーリング層には重みの概念がない。 MaxPooling、AvgPoolingによって出力値が決定される。

【実装演習】 im2colのサンプルコードを写経する ※im2colはあとの演算を楽に実施するために、縦横の画像データを配列で持たせるように形式変換するfunctionである

```
import pickle
import numpy as np
from collections import OrderedDict
from common import layers
from common import optimizer
from data.mnist import load_mnist
import matplotlib.pyplot as plt

# 画像データを２次元配列に変換
# input_data: 入力値
# filter_h: フィルターの高さ
# filter_w: フィルターの横幅
# stride: ストライド
# pad: パディング
def im2col(input_data, filter_h, filter_w, stride=1, pad=0):
    # N: number, C: channel, H: height, W: width
    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) # (N, C, filter_h, filter_w,
out_h, out_w) -> (N, filter_w, out_h, out_w, C, filter_h)

    col = col.reshape(N * out_h * out_w, -1)
    return col
```

※参考資料：2_6_simple_convolution_network.ipynb

【実装演習考察】 畳み込み層での演算速度をUPさせる実装を学びました。 配列の形式変換は実務でも発生することが多く、理解は難しくありませんでした。 確認テストでもありましたが、入力画像とフィルタの大きさからの出力画像サイズ演算は自身でも手元で可能な計算なので 訓練していこうと思いました。

【自己学習】 参考書籍：ゼロからつくるDeepLearning なぜCNNという概念が必要なのか？ 通常のニューラルネットワーク（全結合層）では、縦・横・チャンネルという3次元の概念が1次元にまとめられてしまうので 画像においては特に重要である空間的情報が消失してしまう。 CNNではそれらを正しく理解できる（可能性がある）