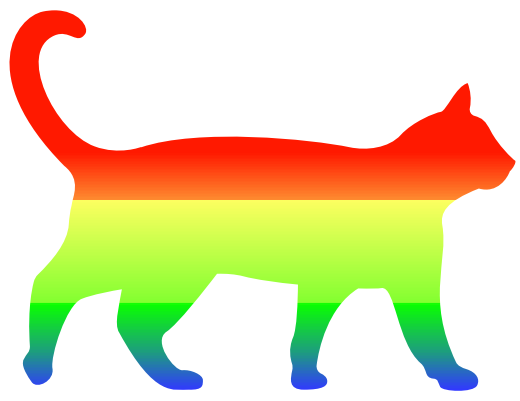
Computer Vision by DeepLearning with PyTorch



Noriaki Oshita

まえがき

ニューラルネットとくに深層学習の実装の方法はブラックボックスであることが多い.

それでも論文は出てくるし新しいなんとかNetというものは大量に生産される．本当に彼らは実装しているだろうか．いや実装しているから論文を書いているのだろう.

私の私情だが仕事で深層学習をやることになった．どうやって実装しようかと思ってまずはPyTorchで実装を試みた．だが，数十分後「どうやって実装すれば良いんだ．」PyTorchのことは分かったけれど実装をどうやったらできるのだろう．そして自分だけのニューラルネットを作るにはどうすれば良いのだろうと.

論文を見れば実装できるのか，それは違う．彼らは苦労して実装をできるようになっている.（と推測する）

その苦労をできる限りせずに自分だけのニューラルネットを作ることに時間を捧げることのほうが有意義だ．それをサポートするための本が本書である.

特にコンピュータビジョンのアルゴリズムについて力を入れた。

もくじ

1. コンピュータビジョンの歴史
2. Pythonの使い方
3. 深層学習について(CNN)
4. 画像認識アルゴリズムについて(R-CNN,FasterR-CNN,SSD,Yolo,M2Det,SelectiveSearch,NMS)
5. 応用

畳み込み層（convolution layer）

畳み込み層の最初のレイヤーのサイズに設定するのはその画像のテンソルのサイズである。

MNISTなら28×28の784サイズなのでinput_sizeを784で初期化する。そこまではいいだろう．では、隠れ層のノードの数はどうやって決めようとなったときに、あてずっぽに表現力を高めるために大きくすれば良いかといえば経験的にはそうなのだろうが、それでは心もとない。また、本書の理念からそのような考えは的を外れてしまう。

・中間層のノードの個数の決め方

・ストライド

画像上のフィルタの移動範囲を数画素ずつずらすこと。

feedforward convolutional network

ループがなく一方向にのみの流れのニューラルネットワークのことをfeedforward neural network(フィードフォワード ニューラルネットワーク)という.

畳み込み

以下のようなグレースケールの画像を考えます。
簡単のため画像は$W \times W \in \mathbb{R}^2$のサイズ、この場合$W = 4$とし、4×４のサイズとします。
各ピクセルに画素値$\{x_{ij} \in \mathbb{R} \mid i, j = 1,2,\cdots, W-1\}$を持っています。

次に小さな画像(filterと呼ぶ)を考えます。
$H \times H \in \mathbb{R}^2$のサイズとし$H = 2$とします。filterは画素値$\{h_{pq} \in \mathbb{R} \mid p, q = 1,2,\cdots, H-1\}$

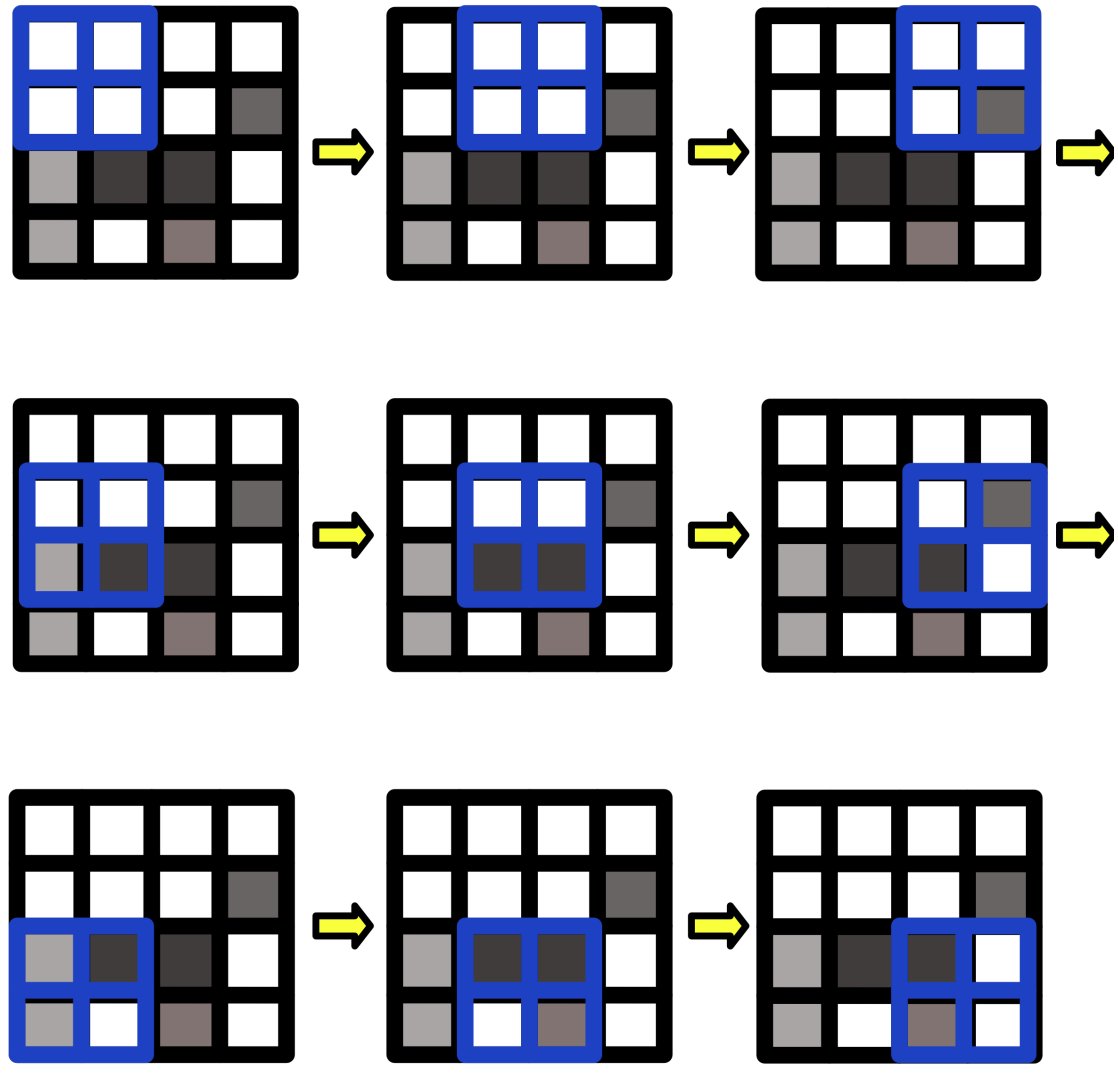
画像の畳み込みとは、画像とfilter間で定義される次に積和計算です。

$$u_{ij} = \sum_{p=0}^{H-1} \sum_{p=0}^{H-1} x_{i+p,j+q} h_{pq}$$

ただし、正確にはこの計算は相関と呼ばれるべきもので、畳み込みとは本来

$$u_{ij} = \sum_{p=0}^{H-1} \sum_{p=0}^{H-1} x_{i-p,j-q} h_{pq}$$

という計算のことです。この場合、相関は、フィルタを上下と左右方向に反転すると畳み込みと同じ結果を与えるので、実質的な違いはなく、最初の式を畳み込みと呼ぶことにします。



畳み込みの図

プーリング層（Pooling Layer）

プーリング層は通常、畳み込み層の直後に挿入されることが多いです。また畳み込み層とプーリング層が続いた後は全結合層（fully connected layer）が配置されます.

多層パーセプトロンで

CIFAR10の画像を分類する。

多層パーセプトロンとはパーセプトロンを多層にした構造を持つニューラルネットである.
また、順伝播型のニューラルネットといわれます。
実装はいたって簡単です。CIFAR１０とは、トロント大学のAlex Krizhevskyさんが作成したデータセットです。10個のオブジェクト（つまり１０のクラス）からできており、その画像のオブジェクトは飛行機、自動車、鳥、猫、鹿、犬、カエル、馬、船、トラックから構成されています。ちなみに100個のオブジェクトから構成されるCIFAR100というデータセットもあります。50000枚のトレーニング画像と10000枚のテスト画像に分かれています。

https://www.cs.toronto.edu/~kriz/cifar.html

・パーセプトロンとは

三つのノードの層からなる学習器のことをさすことが多く，さらに任意の識別関数を設定することができます．またノードは上の層のノードに全て接続している（全結合）ニューラルネットです．図で表すと次の通りです．８

自己符号化器(Autoencoder)

学習方法は誤差関数を確率的勾配降下法で最小化することによって学習します．具体的には誤差関数を最小化することによって，重みとバイアスを決定します．また次元削減としては主成分分析よりもはるかに良い結果を出すことが知られています。[]

変分自己符号化器(Variational Autoencoder)

原著は(Auto-Encoding Variational Bayes: https://arxiv.org/abs/1312.6114)です．どのように確率分布から直接，効率的に推論と学習をすれば良いでしょうか？

解析

まずは画像の加工をしないで、transforms.Resize()とtransforms.ToTensor()だけでやってみる。そして精度がイマイチな場合にrandomClipなどをやってみるということがいいと思います。

まずはシンプルなことから始めて徐々に複雑化していく。最初から複雑なモデルを使うと何が原因なのか分からなくなると思います。

torchvisionのtransformsで画像の設定をします。
transforms.Resize((w,h))の引数はでweightとheightのサイズを同じにして正方形にしています。

```
from torchvision import datasets, transforms
# MNISTの画像の例

input_size = 784
data_transforms = transforms.Compose([
        transforms.Resize((input_size,input_size)),
        transforms.ToTensor()])
train_dataset = ImageFolder('/home/noriakioshita/data/
train/', data_transforms)
test_dataset = ImageFolder('/home/noriakioshita/data/
test/', data_transforms)

train_loader = torch.utils.data.DataLoader(train_dataset,
    batch_size=batch_size, shuffle=True, num_workers=4)
test_loader = torch.utils.data.DataLoader(test_dataset,
    batch_size=batch_size, shuffle=False, num_workers=4)
```

Pythonの使い方

まずはベクトルをPythonで定義してみましょう。

```
lmport numpy as np


# 行ベクトルのを作る
vector_row = np.array([1,2,3])

# 列ベクトルを作る
vector_column = np.array([[1],
                          [2],
                          [3]])
```

次に２つの行列Aと行列Bを定義してみましょう。

```
matrixA = np.array([[2, 3],
                    [4, 6],
                    [8, 5]])

matrixB = np.array([[1, 4],
                    [3, 5],
                    [8, 5]])
```

次にこの行列Aと行列Bの演算をしてみましょう。
まずは行列の足し算です。(イラスト挿入予定)

```
matrixA + matrixB
```

output

```
array([[ 3,  7],
       [ 7, 11],
       [16, 10]])
```

行列の引き算も同様に

matrixA - matrixB

output

```
array([[ 1, -1],
       [ 1,  1],
       [ 0,  0]])
```

行列の積を計算してみましょう。
新たに行列の積を計算するために新たな行列を定義します。
実際にやってみましょう。

行列Aと行列Bを新たに以下のように定義します。

$$A = \begin{bmatrix} 2 & 3 \\ 4 & 6 \end{bmatrix}$$

$$B = \begin{bmatrix} 5 & 6 \\ 2 & 9 \end{bmatrix}$$

行列Aと行列Bの積,ABを計算しましょう！

$$AB = \begin{bmatrix} 2 & 3 \\ 4 & 6 \end{bmatrix} \begin{bmatrix} 5 & 6 \\ 2 & 9 \end{bmatrix}$$

答え

```
A = np.array([[2,3],[4,6]])
B = np.array([[5,6],[2,9]])

# 行列Aと行列Bの積
np.dot(A, B)
```

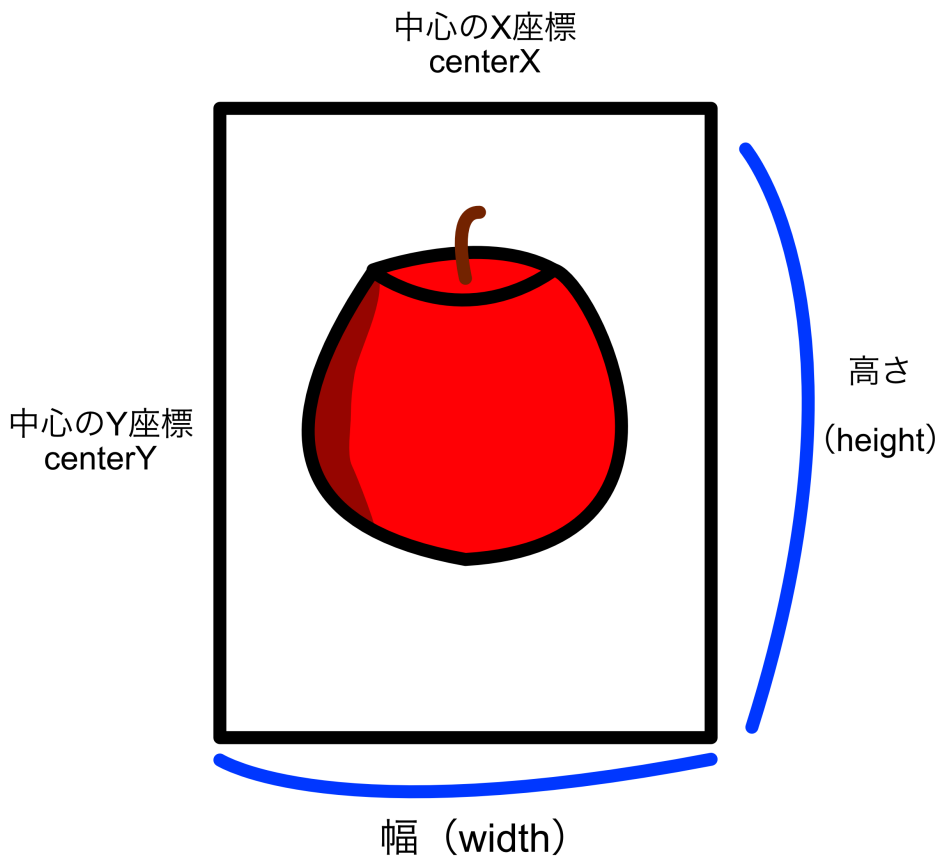
output

```
array([[16, 39],
       [32, 78]])
```

A*Bとやりたくなるかもしれませんが、それはアダマール積(要素ごとの積)になるのでご注意を。

SSD(SingleShotMultiBoxDetector)

バウンディングボックス(Bounding Box)とは物体の境界を矩形で抽出した座標のことである。



アンカーボックスとは事前に大きさを指定された矩形の座標のことである．具体的にはともに、中心のX座標、中心のY座標、幅、高さ(centerX,centerX,width,height)から構成される。SSDはfeed-foward convolutional networkをベースに作られている。
feed-foward convolutional networkとはループを持たないネットワークでかつ、一方向のみの流れのみのネットワークのことである。つまり、流れが逆方向に戻ったりなどはしないということである。（イラスト挿入予定）

参考文献

・機械学習プロフェッショナルシリーズ「深層学習」：岡谷貴之

・Machine Learning with Python Cookbook : Chris Albon