

# 知能システム学特論レポート

(DL2 班) Caffe on Ubuntu

2015 年 7 月 6 日

## 1 報告者

15344203	有田 裕太
15344206	緒形 裕太
15344209	株丹 亮
12104125	宮本 和

## 2 進行状況

- 理論研究
- 畳み込みネットワークについて

## 3 理論研究

### 3.1 畳み込みの定義

濃淡地を各画素に格納したグレースケールの画像を考える。画像サイズを  $W \times W$  画素とし、画素をインデックス  $(i, j) (i = 0, \dots, W - 1, j = 0, \dots, W - 1)$  で表す。これまでインデックスは 1 から開始していたが、画像（及びそれに類するもの）を扱う場合はインデックスを 0 から始めることとする。画素  $(i, j)$  の画素値を  $x_{ij}$  と書き、負の値を含む実数値をとるとする。そして、この画像に適用する  $H \times H$  画素のフィルタ（サイズの小さい画像）を考える。このフィルタのインデックス  $(p, q) (p = 0, \dots, H - 1, q = 0, \dots, H - 1)$  で表し、画素値を  $h_{pq}$  と書く。

画像の畳み込みとは、画像とフィルタ間で定義される次の積和計算のことである。（正確には相関と呼ぶべきだが、フィルタを上下左右を反転すると同じなのでここでは畳み込みと呼んでいる。）

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

この計算は、フィルタの濃淡パターンと類似した濃淡パターンが入力画像上のどこにあるかを検出する働きがある。つまり、フィルタが表す特徴的な濃淡構造を、画像から抽出する「特徴抽出」の働きがある。

### 3.2 パディング

畳込みは画像にフィルタを重なり合う画素とおしの積を求めて、フィルタ全体の和を求める。したがって画像からフィルタがはみ出すような位置に重ねることができず、画像内にフィルタ全体が収まる範囲内でフィルタを動かすと、畳込み処理を行った後の画像サイズは入力画像は小さくなる。この様子を Fig. 1 に示す。

77	80	82	78	70	82	82	140
83	78	80	83	82	77	94	151
87	82	81	80	74	75	112	152
87	87	85	77	66	99	151	167
84	79	77	78	76	107	162	160
86	72	70	72	81	151	166	151
78	72	73	73	107	166	170	148
76	76	77	84	147	180	168	142

 $\otimes$ 

0.01	0.08	0.01
0.08	0.62	0.08
0.01	0.08	0.01

 $=$ 

79	80	81	79	79	98
82	81	79	75	81	114
85	83	77	72	99	144
79	77	77	79	112	155
73	71	73	89	142	162
73	73	77	110	160	166

Fig.1 8 × 8 の入力画像を畳込み処理した場合の出力画像のサイズ縮小の様子

このときの画像サイズは

$$\left(W - 2 \left\lfloor \frac{H}{2} \right\rfloor\right) \times \left(W - 2 \left\lfloor \frac{H}{2} \right\rfloor\right) \quad (3.1)$$

となる。 $\lfloor \cdot \rfloor$  は小数点以下を切り下げて整数化する演算子である。

一方で畳込みの結果が入力画像と同サイズに出力する場合、入力画像の外側に幅  $\lfloor H/2 \rfloor$  の余剰を設け、出力画像が元の入力画像と同サイズになるようにする。このとき余分に設けた部分の画素値を 0 とする方法をゼロパディング (zero-padding) と呼ぶ。しかしゼロパディングを行った場合、画像の周辺部が暗くなってしまうため、画像をその 4 辺で折り返して未定部分の画素値を決める方法や画像の最終囲の画素値をそのまま適用する方法等を用いる場合がある。

### 3.3 ストライド

フィルタの適用位置を 1 画素ずつではなく、数画素ずつずらして計算する場合がある。このずらす間隔をストライド (stride) という。ストライドを  $s$  とするとき、出力画像の画素値は

$$u_{ij} = \sum_{p=0}^{H-1} \sum_{q=0}^{H-1} x_{si+p, sj+q} h_{pq} \quad (3.2)$$

となり、出力画像のサイズは元の入力サイズの約  $1/s$  倍となる。サイズは正確には

$$((W - 1) / s + 1) \times ((W - 1) / s + 1) \quad (3.3)$$

と計算される。

大きなサイズの入力画像を扱う際に、畳込み層の出力側のユニット数が大きくなりすぎるのを防ぐために、2 以上のストライドが使われることがある。しかし、ストライドを大きくすることは画像特徴を取りこぼすことを意味し、性能を悪化させると考えられるため、できるだけ避けるべきである。また、プーリング層においても同様の考え方が用いられる。プーリング層ではその目的から 2 以上のストライドを用いられる。

	77	80	82	78	70	82	82	140	
	83	78	80	83	82	77	94	151	
	87	82	81	80	74	75	112	152	
	87	87	85	77	66	99	151	167	
	84	79	77	78	76	107	162	160	
	86	72	70	72	81	151	166	151	
	78	72	73	73	107	166	170	148	
	76	76	77	84	147	180	168	142	

 $\otimes$ 

0.01	0.08	0.01
0.08	0.62	0.08
0.01	0.08	0.01

 $=$ 

62	71	72	69	65	71	79	107
73	79	80	81	79	79	98	128
76	82	81	79	75	81	114	132
77	85	83	77	72	99	144	145
74	79	77	77	79	112	155	142
74	73	71	73	89	142	162	137
69	73	73	77	110	160	166	134
60	67	68	78	124	154	148	116

Fig.2 8 × 8 の入力画像にゼロパディングを施した場合

## 4 プログラミング

### 4.1 学習を実行する PC 環境の見直し

今まで caffe のサンプル実行，中間層の出力は CPU のみの演算で十分な応答を得られた．しかし与えられたデータセットにおいて学習を行うプログラムを実行する場合，非常に時間がかかることがわかった．

現時点では学習を行うために十分なデータ数を有するデータセットを準備できていないため，大規模なデータセットで学習済みの状態から目的とする別のデータセットへ学習し直す方法（ファインチューニング）を行った．ファインチューニングは学習データがあまり多くない場合でも学習を行うことができる．この手法を用いて試験的に学習用データとテスト用データを 6 枚ずつ用意して学習を行ったが，ファインチューニングが完全に完了するまで約 3 時間かかった．

このことから今後の課題研究を可能な限り円滑に行うために，学習時には GPU を使った並列計算が必要であることがわかった．したがって caffe がサポートしている NVIDIA の「CUDA」及び，Deep Learning 用の CUDA ライブラリ「cuDNN」を使用する必要があると考える．後者のライブラリはデベロッパー登録申請（CUDA Registered Developer Program）が必要であるので申請を行い，認可を待っている状態である．

## 5 今後の課題

- 理論研究を進める．
- Caffe を使いこなす