3D Semantic Segmentation with Submanifold Sparse Convolutional Networks

備考

著者

Benjamin Graham, Martin Engelcke, Laurens van der Maaten

Abst

畳み込みネットワークは，画像，動画，3D形状などの時空間データを解析するためのデファクトスタンダードです．このようなデータの中には，写真のように自然に高密度になるものもありますが，他の多くのデータソースは本質的に疎です．例えば，LiDARスキャナーやRGB-Dカメラで取得された3D点群などです．標準的な「密な」畳み込みネットワークの実装は，このような疎なデータに適用すると非常に効率が悪い．本研究では，空間的に疎なデータをより効率的に処理するように設計された新しい疎な畳み込み演算を導入し，それを用いて空間的に疎な畳み込みネットワークを開発した．本研究では，3D点群の意味的なセグメンテーションを含む2つの課題において，「サブマニフォールド・スパース・コンボリューショナル・ネットワーク（SSCN）」と呼ばれるモデルの高い性能を実証した．特に，最近開催されたセマンティックセグメンテーションコンペティションのテストセットにおいて，我々のモデルはこれまでの最先端技術を上回る性能を示した．

1. Introduction

畳み込みネットワーク（ConvNets）は，写真，ビデオ，3Dサーフェースモデルなど，空間的および時間的構造を持つデータの分析を含む幅広いタスクに対応する最先端の手法です．このようなデータは，多くの場合，高密度に配置された（2Dまたは3D）グリッドで構成されていますが，他のデータセットは自然に疎になります．例えば，手書きの文字は2次元空間の1次元の線で構成され，RGB-Dカ

メラで撮影された写真は3次元の点群であり，ポリゴンメッシュモデルは3次元空間の2次元の面を形成しています．

特に，3次元以上のグリッド上にあるデータには，「次元の呪い」がかかります．このような状況では，データ処理に必要な計算資源を削減するために，可能な限りデータのスパース性を利用することがますます重要になります．例えば，RGB-Dビデオのように，4次元の構造を持つデータを解析する際には，スパース性を利用することが非常に重要です．

従来の畳み込みネットワークの実装は，密度の高いグリッド上のデータに最適化されており，疎なデータを効率的に処理することはできませんでした．最近では，疎なデータを効率的に処理できるように改良された畳み込みネットワークの実装が数多く発表されている[3, 4, 18]．これらの実装の中には，数学的には通常の畳み込みネットワークと同じですが，FLOPsやメモリなどの計算資源が少なくて済むものもあります[3, 4]．先行研究では，im2col演算のスパースバージョンを用いて，計算と記憶を「アクティブな」サイトに制限したり[4]，[22]のvotingアルゴリズムを用いて不要なゼロによる乗算を削減したり[3]しています．OctNets [18]は，関心領域の外側にあるグリッドの部分に「平均化された」隠れた状態を生成するように畳み込み演算子を修正しています．

これまでに開発されたスパース型の畳み込みネットワークでは，各層のスパースデータを "フル "の畳み込みで "ダイレーション(膨張・拡張など) "してしまうという欠点がありました．本研究では，ネットワーク全体で同じレベルのスパース性を維持する畳み込みネットワークを作成できることを示しています．そのため，ResNets [7]やDenseNets [9]のように，大幅に層数の多いネットワークを学習することが実用的になります．

この目的のために，我々はスパース・コンボリューション（SC）を実行するための新しい実装を開発し，サブマニフォールド・スパース・コンボリューション（SSC）と呼ばれる新しい畳み込み演算子を導入した．これらの演算子を，例えば図1に示す例のように，3D点群の効率的なセマンティック・セグメンテーションに最適化されたサブマニフォールド・スパース・コンボリューション・ネットワーク（SSCN）の基礎として使用する．

図1: ShapeNet part -segmentation challenge [23]で得られたオブジェクトの3D点群の例．点の色はパーツ・ラベルを表している

表1では，最近開催されたパート・ベース・セグメンテーション・コンペ[23]のテスト・セットにおけるSSCNの性能を示し，コンペで上位に入賞したいくつかのエントリーと比較している．SSCNは，これらのエントリーのすべてを上回っている．本ライブラリのソースコードは，オンラインで公開されている．

3. Spatial Sparsity for Convolutional Networks

ここでは，d次元畳み込みネットワークを，(d+1)次元のテンソルを入力とするネットワークと定義する．入力テンソルには，d個の時空間次元（長さ，幅，高さ，時間など）と，さらに1個の特徴空間次元（RGBカラーチャンネルや表面法線ベクトルなど）が含まれる．入力は，サイトのd次元グリッドに対応し，各サイトは特徴ベクトルに関連付けられている．入力されたサイトは，特徴ベクトルのいずれかの要素が基底状態でない場合，すなわち非ゼロである場合にアクティブであると定義する．データが自然にスパースにならない場合は，特徴ベクトルが基底状態からわずかな距離しか離れていない入力サイトを排除するために，閾値を使用することができる．入力テンソルは(d+1)次元であるが，活動はd次元の現象であり，特徴次元に沿った線全体がアクティブまたは非アクティブであることに留意されたい．

> - 入力される画像の各画素(サイト)はそれぞれ特徴ベクトルと関連付けられている．
> - 画素ごとに関連付けられた特徴ベクトルのいずれかの要素が非ゼロ -> アクティブ
> - 特徴ベクトルが基底状態からわずかしか離れていないものは，閾値を用いて削除 -> スパースにする

同様に，d次元の畳み込みネットワークの隠れ層は，特徴空間ベクトルのd次元グリッドで表される．入力データをネットワークに伝達する際，入力となる層のサイトのいずれかがアクティブであれば，隠れた層のサイトはアクティブになる．(サイズ3の畳み込みを使用する場合，各サイトは下の隠れ層の3dサイトに接続されていることに注意してください)．隠れ層の活動は，各層が次の層の活動状態のセットを決定するという帰納的な定義に従っている．各隠れ層では，非アクティブなサイトはすべて同じ特徴ベクトルを持っています(すべてゼロ)．基底状態の値は，トレーニング時にはフォワードパスごとに1回，テスト時にはすべてのフォワードパスについて1回だけ計算すればよい．これにより，計算量とメモリ使用量を大幅に削減することができます．

> 特徴ベクトルが基底状態(すべての要素がゼロ)のものは，トレーニング時にはフォワードパスごとに1回，テスト時にはすべてのフォワードパスについて1回だけ計算すればよい．

特に，入力データのスパース性に対応するために畳み込み演算が修正されていないため，上述のフレームワークは過度に制限されていると主張している．入力データに1つのアクティブサイトが含まれている場合，3dの畳み込みを適用すると，3dの活性サイトが存在することになります．同じ大きさのコンボリューションを2回目に適用すると 5dの活性サイトが得られる，というように．VGGネットワーク，ResNets，DenseNets[8, 9, 20]など，数十から数百の畳み込み層からなる最新の畳み込みネットワーク・アーキテクチャを実装する場合，このようなアクティブ・サイトの数の急激な増加は見通しが悪い．

> **サブマニホールド拡張問題**
> 入力データに1つのアクティブサイトが含まれている場合
>
> - 3dの畳み込みを適用する -> 3d アクティブサイト
> - 続けて 3dの畳み込みを適用する -> 5d アクティブサイト
>
> このようなアクティブ・サイトの数の急激な増加は見通しが悪い．

もちろん，単一の活性部位しかない入力に畳み込みネットワークが適用されることはあまりないが，入力データが2次元以上の空間における1次元の曲線や，3次元以上の空間における2次元の曲面で構成されている場合も，前述のダイレーション問題は同様に問題となる．この問題を「サブマニホールド拡張問題」と呼んでいる．図2に示すように，この格子に3×3の小さな畳み込みをかけても，格子のスパース性が急速に失われてしまう．

図2：「サブマニホールド」ダイレーションの例．左：元の曲線．真ん中：重み1/9の通常の3×3畳み込みを行った結果．右図：同じ畳み込みを再度行った結果．
通常の畳み込みでは，畳み込み層ごとに特徴のスパース性が大幅に減少する．

4. Submanifold Convolutional Networks

本研究では，サブマニフォールドダイレーション問題に対する簡単な解決策を検討し，畳み込みの出力をアクティブな入力点のセットにのみ制限する．この方法の潜在的な問題は，ネットワークの隠れ層が，入力データを分類するために必要なすべての情報を受け取らない可能性があることである．特に，2つの隣接する連結成分が完全に独立して処理されてしまう．この問題を解決するために，プーリングまたはストライドコンボリューション演算を組み込んだ畳み込みネットワークを使用している．このような演算は，我々が研究しているスパースな畳み込みネットワーク(4)において重要であり，入力中の切断されたコンポーネント間で情報を流すことができるからである．構成要素が空間的

に近ければ近いほど，構成要素が中間表現で「通信」するために必要なストライド演算は少なくなる．

> **サブマニホールド拡張問題の解決策**
> 畳み込みの出力をアクティブな入力点のセットにのみ制限する．
> しかし，隠れ層が，入力データを分類するために必要なすべての情報を受け取らない可能性がある．
> 特に，2つの隣接する連結成分が完全に独立して処理されてしまう．
> この問題を解決するために，プーリングまたはストライドコンボリューション演算を組み込んだ畳み込みネットワークを使用する．

4.1. Sparse Convolutional Operations

入力特徴プレーンが m 個，出力特徴プレーンが n 個，フィルターサイズが f，ストライドが s のスパースな畳み込みを，$SC(m, n, f, s)$ と定義する．入力のサイズが l であれば，出力のサイズは $(l - f + s)/s$ となる．通常の畳み込みや[4]のスパース畳み込みとは異なり，SC畳み込みでは，非活性部位からの入力をゼロと仮定することで，非活性部位の基底状態を破棄する．この一見小さな変更により，計算コストが約50%削減されます．

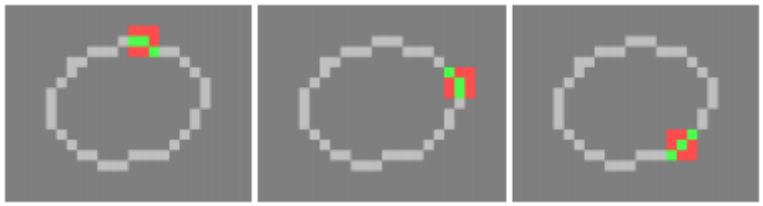

図3：異なるアクティブな空間位置を中心としたSSC(-, -, 3)の受容野．フィールド内のアクティブな場所は緑で示されている．赤色の場所はSSCによって無視されるため，アクティブな場所のパターンは変わらない．

Submanifold sparse convolution.

この論文の主な貢献は，別のスパースなコンボリューションの定義です．fは奇数を表すとする．
我々はサブマニフォールド・スパース・コンボリューション$SSC(m, n, f)$を修正$SC(m, n, f, s = 1)$コンボリューションとして定義する．まず，出力が入力と同じ大きさになるように，入力の両側に$(f - 1)/2$個のゼロを埋めます．次に，入力の対応する部位のサイトがf個アクティブであれば (すなわち，受容野の中央の部位がアクティブであれば)，出力部位がアクティブであると制限する．出力部位がアクティブであると判断された場合，その出力特徴ベクトルはSSCコンボリューションによって計算される（図3参照）．表2は，通常の畳み込み(C)演算と，当社のSCおよびSSC畳み込みの計算量とメモリ使用量を示している．

Active	Type	C	SC	SSC
Yes	FLOPs	$3^d mn$	amn	amn
	Memory	n	n	n
No, $a > 0$	FLOPs	$3^d mn$	amn	0
	Memory	n	n	0
No, $a = 0$	FLOPs	$3^d mn$	0	0
	Memory	n	0	0

表2：通常の畳み込み(C)，スパース畳み込み(SC)，サブマニホールド・スパース畳み込み(SSC)の3つの畳み込みの計算量とメモリ使用量．ここでは，サイズ$f = 3$，パディング$s = 1$の畳み込みを，d次元の単一の場所で考える．ここで，aは空間位置へのアクティブな入力の数，mは入力特徴プレーンの数，nは出力特徴プレーンの数である．

SSC コンボリューションは，スパース性構造を保持するという点でOctNets [18]と似ている．しかし，OctNetsとは異なり，SSC 畳み込みの実装では，空の空間は計算やメモリのオーバーヘッドにはならない．

Other operators.

SC と SSC を使って畳み込みネットワークを構築するには，活性化関数，バッチ正規化，プーリングも必要である．活性化関数は通常通り定義されるが，活性部位のセットに限定される．同様に，バッチ正規化は，通常のバッチ正規化をアクティブサイトの集合に適用したものとして定義する．Max-pooling$MP(f, s)$とaverage-pooling$AP(f, s)$の演算は，$SC(-, -, f, s)$の変形として定義される．MPはゼロベクトルと受容野内の入力特徴ベクトルの最大値をとる．APは，アクティブな入力ベクトルの和の$f - d$倍を計算する．また，デコンボリューション[25]演算$DC(-, -, f, s)$を$SC(-, -, f, s)$コンボリューションの逆数として定義する．DC コンボリューションから得られるアクティブな出力サイトのセットは，対応するSCコンボリューションへの入力アクティブサイトのセットとまったく同じである：入力サイトと出力サイトの間の接続は単に反転している．

4.2. Implementation

(S)SCコンボリューションを効率的に実装するために，入力/隠れ層の状態を，1つのハッシュテーブル(5)と1つの行列の2つの部分に分けて保存する．行列は，サイズが$a \times m$で，a個の活性部位ごとに1つの行が含まれています．ハッシュテーブルには，すべての活性部位の（位置，行）のペアが含まれている．位置は整数の座標のタプルで，行番号は特徴行列の対応する行を示す．フィルタサイズfの畳み込みを考えると，$F = \{0, 1, \ldots, f - 1\}^d$とすると，畳み込みフィルタの空間サイズを表す．ルールブックは，それぞれが2列のf^d個の整数行列の集まり$R = (R_i : i \in F)$と定義する．$SC(m, n, f, s)$コンボリューションを実装するには，次のようにする．

1. 入力されたハッシュテーブルを一度イテレートします．入力レイヤーのポイントと，それを見ることができる出力レイヤーのすべてのポイントを反復することで，出力ハッシュテーブルとルールブックをその場で構築します．出力サイトが最初に訪問されると，出力ハッシュテーブルに新しいエントリが作成される．出力 y の受容野の点 i に位置する各アクティブ入力 x について，ルールブックの要素 R_i に行(input-hash(x), output-hash(y))を追加する．
2. 出力行列をすべてゼロに初期化します．各 $i \in F$ について，パラメータ行列 $W^i \in R^{m \times n}$がある．R_i の各行 (j, k) について，入力特徴行列の j 番目の行に W^i を乗じ，出力特徴行列の k 番目の行に加える．これは，行列-行列の乗算-加算なので，GPUで効率的に行うことができます．

> **SCの実装**
> 入力 / 隠れ層の状態を，1つのハッシュテーブルと1つの行列の2つに分けて保存．
> a は空間位置へのアクティブな入力の数，m は入力特徴プレーンの数
> 行列:
>
> - サイズ: $a \times m$
> - a個の活性部位ごとに1つの行が含まれる．
>
> ハッシュテーブル:
>
> - すべての活性部位の（位置，行）のペアが含まれている．
> - 位置: 整数の座標のタプル
> - 行　: 特徴行列の対応する行を示す．

> **ハッシュテーブル**
> データの入った箱に名札を付けて，名札と中身のデータをセットで管理するやり方．
> key と values をペアで管理するデータ構造

> **イテレータ**
> イテレーションとは，反復，繰り返しという意味の英単語