ディープラーニングフレームワークCaffeの分散処理における多種クライアントを用いた比較

一瀬 絢衣[1]　竹房 あつ子[2]　中田 秀基[3]　小口 正人[1]

概要：近年各種センサの普及とクラウドコンピューティング技術の習熟により，各種センサデータの取得や蓄積が容易になった．その結果監視カメラなどのデータをクラウドで収集して解析するアプリケーションも普及してきている．ここで，センサデータをそのまま送信してクラウドで処理する場合，プライバシや各種センサとクラウド間のネットワーク帯域の問題が生じてしまう．
我々は，クラウドへ送るデータのデータ量の削減とプライバシの保護を目的とし，機械学習フレームワーク Caffe の処理をクライアント側とクラウド側でパイプライン的に分散処理を行う手法を提案している．その評価として，ニューラルネットワークの分割箇所やパラメータを変化させ処理時間を比較してきた．本稿では，クライアント側として性能の異なる 3 種類の端末を用いて実験を行い，処理時間を比較した．結果から，端末の性能により全体の処理時間に影響する処理過程が異なり，効率のよい分散方法も異なることも確認できた．

Ayae ICHINOSE[1]　Atsuko TAKEFUSA[2]　Hidemoto NAKADA[3]　Masato OGUCHI[1]

1. はじめに

近年各種センサの普及やクラウドコンピューティング技術の習熟により，各種センサデータの取得や蓄積が容易になった．その結果監視カメラなどのセンサを用いたライフログデータをクラウドで収集して解析するアプリケーションも普及してきている．ネットワークカメラと呼ばれる，サーバ機能を持つカメラも比較的安価で手に入るようになり，防犯・監視用途や家庭でペットや子供の様子を遠隔地から見守るという用途などで広まっている．このようなサービスでは，一般家庭にサーバやストレージを設置して全ての処理を行うことは困難であるため，センサデータをそのまま送信してクラウドで処理するのが一般的である．センサデータの解析をクラウドで行う研究は多くされており，Twitter のセンチメント分析など小規模かつ大量のデータをクラウド内で効率よく解析する手法が検討されている．しかし，動画像データ解析においては，連続的に大容量データを転送する必要があるため，各種センサとクラウド間のネットワーク帯域の問題が生じてしまう．また，監視カメラの画像には個人の生活や行動に関する情報も含まれるため，プライバシの問題もある．
画像や映像の解析には，ディープラーニングと呼ばれる手法が広く使われている．ディープラーニングとは，人間の脳の神経回路がもつ仕組みを模した情報処理システムであるニューラルネットワークの中で識別を行う中間層を多層化したものを用いた機械学習であり，データからの特徴抽出を自動で行うことができるため，精度やスピードの向上という点で注目されている．ディープラーニングのフレームワークには TensorFlow[1] や Chainer[2] などがあり，その中の一つである Caffe[3] の特徴は，高速であることと，各種学習済みネットワークモデルが用意されていることである．ディープラーニングでは適切なネットワーク定義を作ることが一つの壁となるが，Caffe では用意されたネッ

1　お茶の水女子大学
2　国立情報学研究所
3　産業技術総合研究所

トワークモデルを利用することができるため，誰でも簡単に実験を行うことができる．

我々はこれまでの研究において，クラウドへ送るデータのデータ量の削減とプライバシの保護を目的とし，センサデータの解析の一部をクライアント側で行うフレームワークを提案している [4]．ディープラーニングフレームワーク Caffe の機械学習処理ネットワークを分割できるよう実装することで，クライアント側とクラウド側でのパイプライン的な分散処理を可能にした．また，クライアントのみでの処理，クライアントとクラウドでの分散処理，クラウドのみでの処理の 3 つの場合における処理時間の比較により提案手法の有効性を示した．

本研究ではクライアントとして一般家庭に置かれる小規模なセンサから，Array of Things[5] 等の都市モニタリング用のセンサボックスやエッジコンピューティングと呼ばれるセンサに近いサーバでの処理が可能な場合までを想定し，性能の異なる様々なクライアントが使用された場合の性能特性を調査する．クライアント側としてクラウド側と同質ノードで CPU のみを用いたノード，Jetson，Raspberry Pi の異なる性能の計算機を用いて実験を行い，それぞれの処理時間を比較した．実験には 32 × 32 画素の画像が 10 カテゴリに分類されたデータセットである CIFAR-10[6] を用い，ネットワークモデルの分割個所やパラメータを変化させ，データ量や精度，処理時間の変化を調査した．実験から，クライアント側の端末の性能が低い場合にはクライアント側で行う処理の多さが全体の処理時間に大きく影響し，クライアント側の性能が高い場合には通信時間，すなわち通信時のデータ量が大きく影響することが確認できた．また，そのバランスにより効率のよい分散方法も異なることが確認できた．

2. ディープラーニング

ディープラーニングとは，人間の脳の神経回路がもつ仕組みを模した情報処理システムであるニューラルネットワークの中で，識別を行う中間層を多層化したものを用いた機械学習を指す．中間層が複数になっていることにより何段階かで認識を繰り返し，色や形状，全体像など複数の特徴を抽出してより正確な識別が可能となっており，画像や音声の認識に広く使われている．

Caffe (Convolutional Architecture for Fast Feature Embedding) はディープラーニングフレームワークの一つであり，Berkeley Vision and Learning Center が中心となって開発を進めている．Caffe は特定の機能を持つモジュールを組み合わせて構成されており，各モジュールが相互に通信することでシステム全体としての動作を決定している．これにより，新しいデータフォーマットやネットワーク層への拡張が容易にできる．Caffe は C++で実装されており，他言語から Caffe が提供する API を利用して C++で実装されたコアモジュールを利用することができる．さらに，GPU に対応していることから高速な処理が可能である．学習済みネットワークモデルが提供されているため誰でも簡単に実験を行うことができるという特徴がある．

Caffe は，ディープラーニングの中でも畳込みニューラルネットと呼ばれる，主に画像認識に応用される構造に対応している．畳込みニューラルネットは，通常畳込みとプーリングという画像処理の基本的な演算を行う層がペアで複数回繰り返されたあと，隣接層間のユニットが全結合した全結合層が配置される構造になっている．畳込み層では入力画像に対しフィルタを適用し，フィルタをずらしながら各重なりで両者の積和計算を行うことによって，フィルタが表す特徴的な濃淡構造を画像から抽出する．プーリング層では画像上で正方領域をとり，この中に含まれる画素値を使って一つの画素値を求め，畳込み層で抽出された特徴の位置感度を若干低下させる．

Caffe のネットワークに利用される各層について記述する．

- Convolution
  畳込み計算を行う．フィルタ数とカーネルサイズを設定する必要があり，オプションとしてフィルタの適用位置の間隔を表すストライドや入力データの縁に加えるピクセル数を表すパディング，チャネルの分割の数を表すグループを設定することが可能である．
- Pooling
  プーリングの計算を行う．カーネルサイズの指定が必要であり，プーリングの方法や，パディング，ストライドの設定が可能である．プーリングの方法には，画素値の集合から最大値を選ぶ最大プーリングや，平均値を計算する平均プーリングなどが挙げられる．
- Local Response Normalization
  インプットの正規化を行うことにより，側方抑制を行う．
- Inner Product
  インプットをベクトルとして扱い，単一ベクトルのアウトプットを生成する．全結合層である．

3. 分散ディープラーニングフレームワーク

我々は，図 1 で示す分散ディープラーニングフレームワークを提案，実装している．畳込みニューラルネットワーク処理シーケンスを，クライアント側とクラウド側に分離する．クライアント側とクラウド側は，独立した Caffe のプロセスとなる．ネットワーク処理シーケンスを分離するために，新たに Source と Sink と呼ぶ仮想的な層を導入した．

Sink はクライアント側ネットワークの末端に位置し，Source はクラウド側ネットワークの起点となる．Sink と Source は TCP/IP で接続される．Sink は上流レイヤからデータを受け取ると，Source に転送し，Source からの Ack

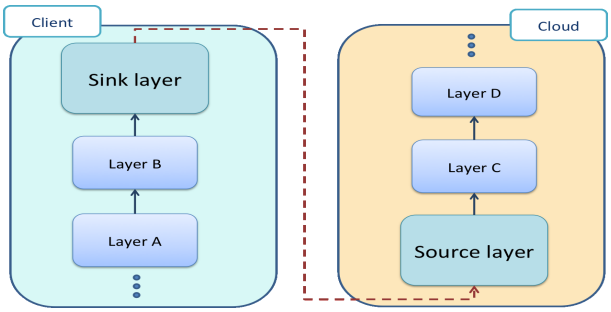

図 1　分散ディープラーニングフレームワーク

```
layer {
  name: "sink1"
  type: "Sink"
  bottom: "pool1"
  sink_param: {
          host_name: "server.example.com"
          port:      3000
    }
}
```

図 2　Sink の設定ファイル例

```
layer {
  name: "source1"
  type: "Source"
  top: "pool1"
  source_param: {
            port:      3000
    }
    reshape_param{ shape { dim : [ 100, 32, 16, 16 ] } }
}
```

図 3　Source の設定ファイル例

を待つ．Source は Sink からのデータを受け取ると，Ack を返してから，下流にデータを流す．クライアント側プロセスと，クラウド側プロセスは，パインプライン的に並列に動作することに注意されたい．Sink と Source の設定ファイル例を図 2 と図 3 に示す．Sink にはクラウド側プロセスのホスト名とポート番号を指定する．Source にはポート番号を指定する． 一つのネットワークを分割する際に，複数のリンクが分断される場合には，複数の Sink-Source ペアが必要になる．分散処理を行うことにより，映像や画像そのものでなく特徴量をクラウドに送るため元画像への復元が困難となり，プライバシが確保される．またデータ量を小さくしてから送ることにより，ネットワーク帯域の制限による性能劣化も軽減できることを確認した．

4. 実験

クラウド側と同質ノードで CPU のみを用いたノード，Jetson，Raspberry Pi の異なる性能のクライアント計算機を用いて処理時間の計測を行い，それぞれの処理時間の比較により提案手法の有効性について考察する．処理時間は，クライアント側，クラウド側として 2 つの端末を利用して

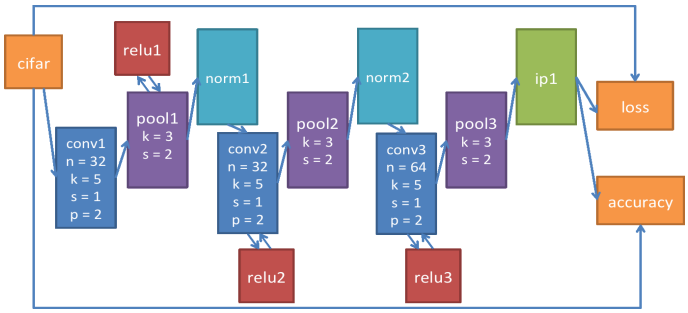

図 4　CIFAR-10 識別用ネットワークモデル

表 1　パラメータ

n：num_output	フィルタ数
p：pad	パディングの幅
k：kernel_size	フィルタの大きさ
s：stride	フィルタの適用位置の間隔
g：group	チャネルの分割数

1 バッチの識別にかかる時間を計測したものを用いた．始めに実験に用いたデータセット，ネットワークモデルについて記述し，分散方法を説明した後，実験に用いた端末を記述し，それぞれの実験の結果から得られた提案手法の評価を述べる．

4.1 CIFAR-10 識別用ネットワークモデル

実験では CIFAR-10 のデータセットを用いた．CIFAR-10 は 10 種類のカテゴリに分類される 32 × 32 画素の画像のデータセットである．Caffe では CIFAR-10 データセットの識別用ネットワークモデルが提供されており，その構造は図 4 のようになっている．2 節に示した各層と対応しており，conv が Convolution，pool が Pooling，norm が Local Response Normalization，ip が Inner Product を示す．relu は Rectified Linear Unit であり，一度データを受け取り，受け取った層にデータを返す．デフォルトでは入力値 x に対し，max(x, 0) を返す活性化関数である．各層で定義されているパラメータは表 1 に示す通りである．
Caffe においてデータはバッチサイズ，チャネル数，2 次元イメージサイズの 4 次元の配列で格納されており，チャネル数は直前の畳込み層におけるフィルタ数と一致する．テストにおけるバッチサイズはデフォルトでは 100 に設定されており，100 枚の画像を一度に処理している．データ量は提供されているネットワークモデルのデフォルトの値では始めの (100 × 3 × 32 × 32)byte からフィルタ数 32 の conv1 層を通り (100 × 32 × 32 × 32)byte へ，次にストライド 2 の pool1 層を通り (100 × 32 × 16 × 16)byte へと変化している．ストライドが 2 以上であると出力画像の大きさが小さくなるため，データ量が少なくなる．

4.2 分散方法

本稿で実験に利用した，通信時のデータ量を考慮した 2

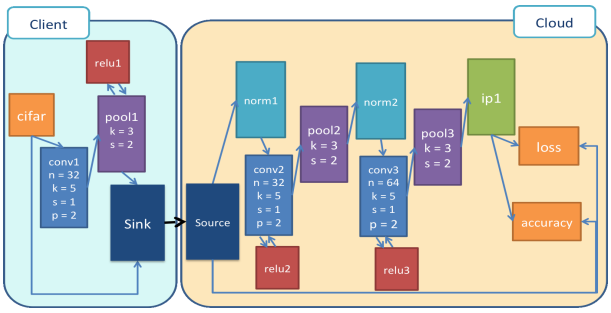

図 5　分散方法 1

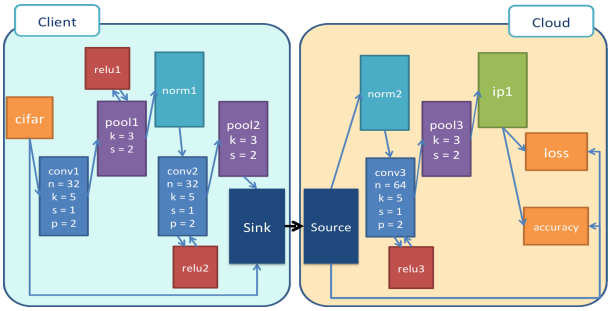

図 6　分散方法 2

つの分散方法を示す．生のデータの 1 バッチあたりのデータ量は 307.2KB である．

- 分散方法 1
  pool1 層，norm1 層間でネットワークを分離する (図 5)．データ量はデフォルト値では生のデータの 3 分の 8 倍となっている．ここで，conv1 層のフィルタ数を削減させることにより通信時のデータ量を削減させることを考える．フィルタ数の削減は識別率の低下に繋がることが考えられるため，フィルタ数をデフォルト値の 32 から減らした際の識別率の変化を調査した．conv1 層フィルタ数と 1 バッチあたりの通信時のデータ量，計測した識別率の対応を表 2 に示す．結果から，少ないフィルタ数で識別率が収束していることが確認でき，デフォルト値の 32 の場合に 78.11%であるのに対し，4 の場合でも 73.35%となった．このとき，通信時のデータ量は生のデータと比較して 3 分の 1 となっている．つまり，畳込み層のフィルタ数を削減することにより，高い識別率を保ちながら通信時のデータ量を削減できることが確認できた．
- 分散方法 2
  pool2 層，norm2 層間でネットワークを分離する (図 6)．データ量はデフォルト値では生のデータの 3 分の 2 倍となっている．conv2 層のフィルタ数を削減させて通信時のデータ量をさらに削減させることを考える．分散方法 1 と同様，フィルタ数をデフォルト値の 32 から減らした際の識別率の変化を調査した．conv2 層のフィルタ数と 1 バッチあたりの通信時のデータ量，計測した識別率の対応の対応を表 3 に示す．conv1 層と同様，conv2 層のフィルタ数を削減した際も高い識別率を保っていることが確認できた．フィルタ数が 4 の場合でも 0.7355 を計測し，このとき通信時のデータ量は生のデータを送信した際と比較して 12 分の 1 となっている．分散 1 と同様に，高い識別率を保ちながら通信時のデータ量を削減できることが確認できた．

表 4　クラウド側端末の性能

OS	Ubuntu 16.04.1 LTS
CPU	Intel(R) Xeon(R) CPU W5590 @3.33GHz (8 コア) × 2 ソケット
Memory	8Gbyte
GPGPU	NVIDIA GeForce GTX 980

表 5　Jetson の性能

OS	Ubuntu 14.04.5 LTS
CPU	NVIDIA 4-Plus-1 Quad-Core ARM Cortex-A15
Memory	8Gbyte
GPU	NVIDIA Tegra124 PM375

表 6　Raspberry Pi の性能」

OS	Linux raspberrypi 4.1.19+
CPU	900MHz quad-core ARM Cortex-A7r
Memory	1Gbyte

4.3　処理時間計測による提案手法の評価

クライアント側，クラウド側として 2 つの端末を利用して 1 バッチの識別にかかる処理時間を計測する．クライアント側で全ての処理を行う場合 (Client)，提案手法 [4] により処理を分散させる場合 (Distribution)，データをそのまま送信してクラウドで処理を行う場合 (Cloud) の 3 つの処理時間を比較する．”Distribution” と”Cloud” では，クライアント側とクラウド側で同期して処理しているため，通信時間や待ち時間を含むクライアント側の処理時間を用いた．

4.3.1　実験環境

クライアント側の端末としてクラウド側と同質のノードで CPU のみを用いたノード，Jetson，Raspberry Pi を用いて実験を行った．クラウド側として用いたノード，Jetson，Raspberry Pi の性能をそれぞれ表 4，表 5，表 6 に示す．いずれの実験においてもクラウド側では GPGPU を用いた．また，実験ではクライアントとクラウド間のネットワーク帯域を一般的なセンサとクラウド間の通信環境を想定して，10Mbps とした．

4.3.2　同質ノード CPU

クライアント側としてクラウドと同質ノードを用いた場合の分散方法 1 の処理時間を図 7，分散方法 2 の処理時間を図 8 に示す．クライアント側では CPU のみを用いた．横軸が直前の畳込み層のフィルタ数を表し，縦軸が処理時間を表す．分散方法 1 では．”Distribution” の通信時の

表 2　conv1 層のフィルタ数を変化させた際の通信時のデータ量 (KB) と識別率 (%)

conv1 層のフィルタ数	4	8	12	16	20	24	28	32
通信時のデータ量 (KB)	102.4	204.8	307.2	409.6	512.0	614.4	716.8	819.2
識別率 (%)	73.35	76.41	76.66	77.21	77.11	78.18	77.54	78.11

表 3　conv2 層のフィルタ数を変化させた際の通信時のデータ量 (KB) と識別率 (%)

conv2 層のフィルタ数	4	8	12	16	20	24	28	32
通信時のデータ量 (KB)	25.6	51.2	76.8	102.4	128.0	153.6	179.2	204.8
識別率 (%)	73.35	76.56	77.16	77.13	77.71	77.63	77.99	78.11

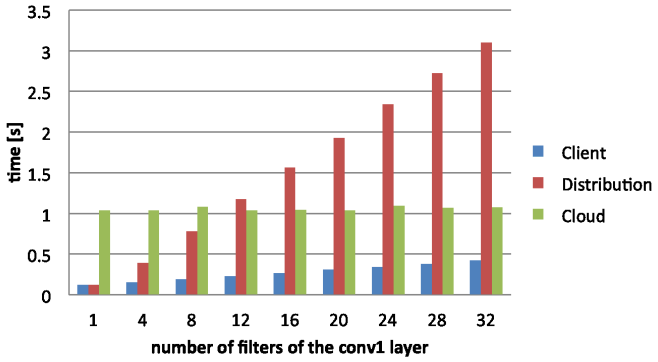

図 7　同質ノードを用いた場合の分散方法 1 の処理時間

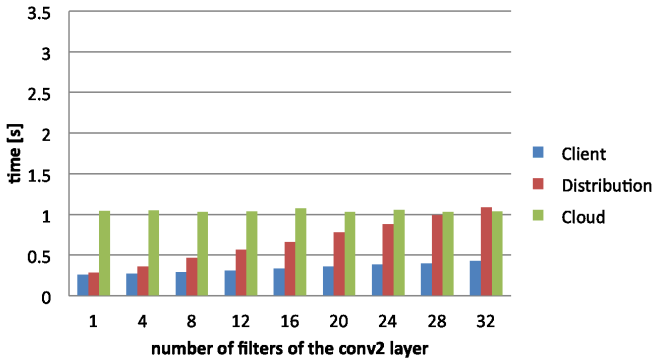

図 8　同質ノードを用いた場合の分散方法 2 の処理時間

データ量が生のデータ量より小さくなるフィルタ数 8 以下の場合に，”Distribution” が”Cloud” よりも高速に処理できた．分散方法 2 では，”Distribution” は”Cloud” よりも通信時のデータ量が常に小さいため，フィルタ数 28 以下の場合に速くなることが確認できた．用いた端末の性能が高いため，結果が画像処理にかかる時間ではなく通信時間に依存し，通信時のデータ量の削減が全体処理時間の削減に大きく影響することが確認できた．通信を行わずにクライアント側で全ての処理を行う”Client” が最も高速であったが，クライアント側に高性能な端末を用意できない場合もある．

4.3.3　Jetson

クライアント側としてやや高性能なセンサ端末である Jetson を用い，GPU を用いた場合の分散方法 1 の処理時間を図 9，分散方法 2 の処理時間を図 10，CPU のみで行った場合の分散方法 1 の処理時間を図 11，分散方法 2 の処理時間を図 12 に示す．

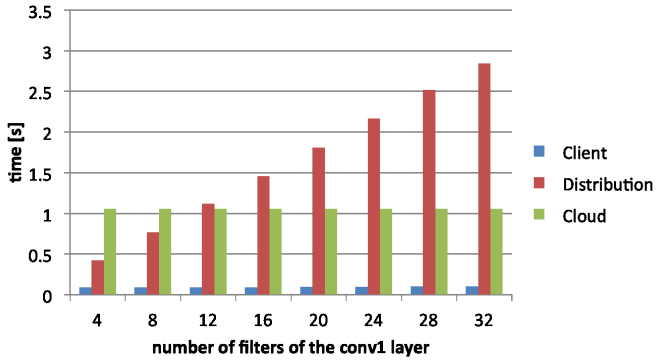

図 9　Jetson で GPU を使用した場合の分散方法 1 の処理時間

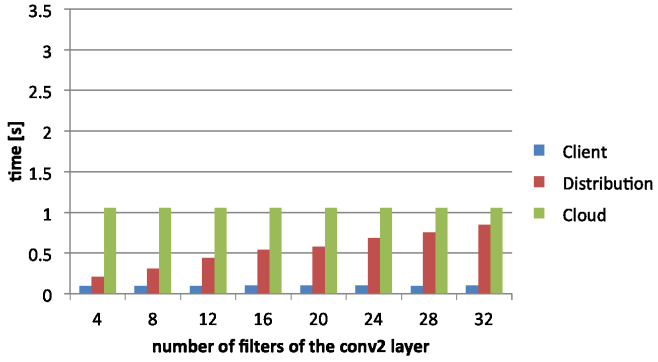

図 10　Jetson で GPU を使用した場合の分散方法 2 の処理時間

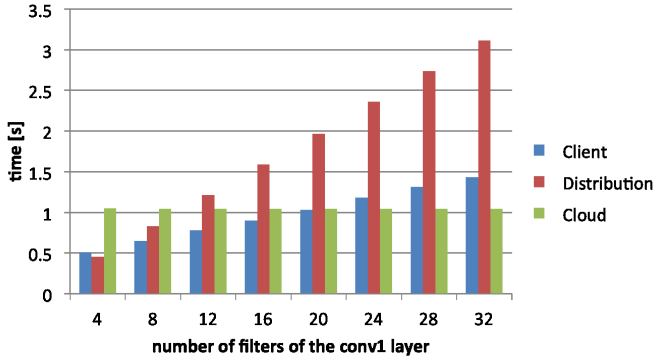

図 11　Jetson で CPU のみを使用した場合の分散方法 1 の処理時間

GPU を用いた場合は，クラウド側と同質ノードの CPU を用いた場合よりも処理が速くなったため，”Client” と”Dis-

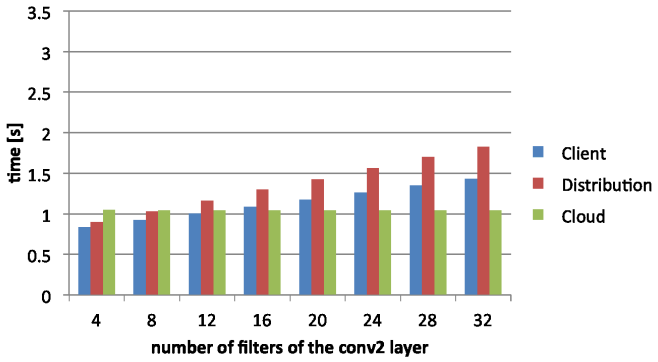

図 12　Jetson で CPU のみを使用した場合の分散方法 2 の処理時間

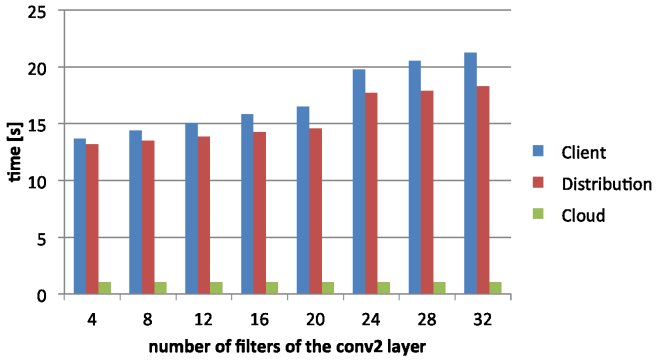

図 13　Raspberry Pi を用いた場合の分散方法 1 の処理時間

tribution” の処理時間がさらに削減された．全体の処理時間に対する通信時間の割合がさらに高くなり，分散方法 2 ではフィルタ数がデフォルトの場合でも”Distribution” が”Cloud” よりも高速に処理できた．分散方法 1 では，フィルタ数をデフォルト値の 32 に設定した場合では通信時のデータ量が多くなってしまうため”Distribution” の処理時間が”Cloud” の約 2.7 倍になった．一方，フィルタ数 8 以下の通信時のデータ量が削減されている場合は”Cloud” より高速に処理できた．

Jetson で CPU のみを使用した場合はクライアント側の処理速度が遅くなるため，全体の処理時間にクライアント側の処理の多さも大きく影響していることが確認できた．通信時間とクライアント側の処理の量がどちらも全体の処理時間に大きく影響し，分散方法 2 においてもフィルタ数が 12 以上の場合に”Distributioin” の処理時間が最も長くなるという結果であった．しかし，通信時のデータ量を削減させた場合には”Cloud” の処理時間が最も長くなっていることが確認できた．また，分散方法 1 ではクライアント側の処理が少ないため，通信時間も削減されるフィルタ数が 4 の場合に，”Distribution” が最も速いことが確認できた．

4.3.4　Raspberry Pi

クライアント側として通常のセンサ端末を想定した Raspberry Pi を用いた場合の分散方法 1 の処理時間を図 13，分散方法 2 の処理時間を図 14 に示す．クラウド側と同質ノードの CPU を用いた場合と異なり，分散方法 1，分散方法 2 ともに”Cloud” が最も速いという結果であった．クライアント側の性能が低く処理が遅いため，全体の処理時間にクライアント側での処理の多さが大きく影響している．しかし，データの匿名化や，多数のクライアントがクラウドへ大量のデータを転送することを想定すると，クライアントでの前処理やデータ量の削減は必要な処理だと考えられる．

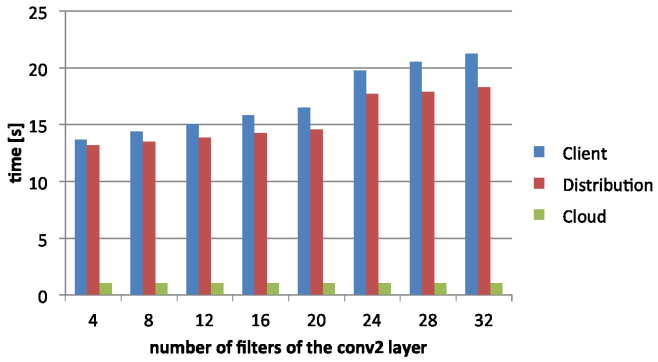

図 14　Raspberry Pi を用いた場合の分散方法 2 の処理時間

4.3.5　考察

クライアント側として性能の異なる端末を用いた実験結果の比較により，端末の性能，クライアントでの処理量，クラウドへの転送量によって全体の処理時間が異なり，効率のよい分散方法も異なることが確認できた．性能が高い場合は全体の処理時間が 2 つの端末間の通信時間に依存するため，通信時のデータ量を削減することができる提案手法を用いた場合の処理時間が短く，有効性が示された．性能が低い場合は全体の処理時間に性能の低い端末で行う処理の多さが大きく影響するため，クラウド側で全ての処理を行う場合が最も速いことがわかった．しかし，実環境においてはクラウドに対し多数のクライアントが接続されるため，大量のデータを多数のクライアントから収集するには帯域の課題が残る．また，一般ユーザの監視カメラなどから得られる生のデータをクラウドに送る場合にはプライバシの問題も解決できない．よって，一部の処理をクライアント側で行ってからクラウド側へ送ることができる提案手法は有効だと考えられる．

5.　関連研究

近年ニューラルネットワークは正確にパターンを分類し識別するのに広く用いられている [7][8]．しかし，そのフレームワークの多くは一つの計算機で GPU を活用して高速に実行することに焦点が置かれている．またクラウド内で様々な環境で Caffe を動かす研究が行われている [9]．この研究では，Caffe の内部を再実装し，既存の実行環境より 6.3 倍のスループットを実現している．この向上により，CNN のエンドツーエンドの学習速度がコンピュータの処理速度に直接比例するようになっており，CPU-GPU の環

境での性能の向上が可能である．しかし，これはクラウド内での高速な処理を可能にするものであり，センサとクラウド間の分散実行は想定していない．

研究 [10] では，ネットワーク環境でのニューラルネットワークによるフォント識別システムが構築されている．この研究は，シンクライアントで動作するシステムを提案しており，クライアント側は入出力のみで，サーバ側でニューラルネットの識別処理を行っている．

また，XNOR.ai[11] では，サーバへの接続をせずに処理能力の低いクライアントのみで学習させることが可能である．これらに対し，本研究ではニューラルネットの識別処理をクライアント側とサーバ側で負荷分散をさせることにより，より適切な処理が可能となる．

ニューラルネットワークの分散処理としてはDIANNEミドルウェアフレームワークが挙げられる [12]．通常のニューラルネットが入力層，出力層と一つ以上の隠れ層から構成されるのに対し，DIANNEミドルウェアフレームワークでは各々のニューラルネットワークの層がモジュールで構成される．このモジュール的アプローチにより複数の異なるデバイスに分散してニューラルネットワークの構成要素を実行することができる．一方 DIANNE では通信時のデータ量や分散させた際の処理時間については言及されておらず，全体として処理時間が長くなってしまうことも考えられる．本研究では，一台の端末が行う処理の量に加えて通信時のデータ量を考慮することにより処理時間の削減を図り，一般消費者のライフログの処理における効率のよいフレームワークを考えている．

6. まとめと今後の課題

本研究では，プライバシやネットワーク帯域を考慮したセンサデータ解析処理を目的とし，ディープラーニングフレームワーク Caffe のネットワークの分離を実装し，センサデータの解析の一部をクライアント側で行うフレームワークを提案してきた．本稿では性能の異なるクライアント端末を用いて処理時間を測定し，比較した．結果から，クライアント側の端末の性能が低い場合にはクライアント側で行う処理の多さが全体の処理時間に大きく影響し，クライアント側の性能が高い場合には通信時間，すなわちクライアントとクラウド間の通信データ量が大きく影響することが確認できた．また，それらのバランスにより効率のよい分散方法も異なることが確認できた．

今後の課題として，クライアント側の端末数を増やし，より実環境に近い実験を行うことを検討している．

謝辞

この成果の一部は，JSPS 科研費 JP16K00177 および国立研究開発法人新エネルギー・産業技術総合開発機構（ＮＥＤＯ）の委託業務の結果得られたものです．

参考文献

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. and Zheng, X. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems (2015). `http://download.tensorflow.org/paper/whitepaper2015.pdf`. pp. 1-19.

[2] Tokui, S., Oono, K., Hido, S. and Clayton, J.: Chainer: a Next-Generation Open Source Framework for Deep Learning, *In Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS)* (2015). 6 pages.

[3] Jia, Y. et al.: Caffe: Convolutional Architecture for Fast Feature Embedding, *arXiv preprint arXiv:1408.5093* (2014).

[4] Ichinose, A., Takefusa, A., Nakada, H. and Oguchi, M.: Pipeline-based Processing of the Deep Learning Framework Caffe, *In proceedings of 11th ACM International Conference on Ubiquitous Information Management and Communication (IMCOM2017)*.

[5] Array of Things, `https://arrayofthings.github.io/`.

[6] Krizhevsky, A. et al.: CIFAR-10.

[7] Kriahevsky, A., Sutskever, I. and Hinton, G.: ImageNet classification with deep convolutional neural networks, *NIPS* (2012).

[8] Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R. and Lecun, Y.: OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks, *International Conference on Learning Representations (ICLR 2014)* (2014).

[9] Hadjis, S., Abuzaid, F., Zhang, C. and Ré, C.: Caffe Con Troll: Shallow Ideas to Speed Up Deep Learning, *Proceedings of the Fourth Workshop on Data Analytics in the Cloud*, DanaC'15, pp. 2:1–2:4 (2015).

[10] Wang, Z., Yang, J., Jin, H., Brandt, J., Agarwala, A., Wang, Z., Song, Y., Hsieh, J., Shechtman, E., Kong, S. and Huang, T. S.: DeepFont: A System for Font Recognition and Similarity, *In Proceedings of the 23rd ACM international conference on Multimedia (MM'15)*, pp. 451–459 (2015).

[11] XNOR.AI, `https://xnor.ai/`.

[12] De Coninck, E., Verbelen, T., Vankeirsbilck, B., Bohez, S., Leroux, S. and Simoens, P.: DIANNE: Distributed Artificial Neural Networks for the Internet of Things, *In Proceedings of the 2Nd Workshop on Middleware for Context-Aware Applications in the IoT*, M4IoT 2015, New York, NY, USA, ACM, pp. 19–24 (2015).