「マルチメディア，分散，協調とモバイル
(DICOMO2021)シンポジウム」 令和3年6月-7月

リッチクライアントを用いた分散機械学習における画像認識の検討

高野 紗輝[1]　中尾 彰宏[2]　山本 周[2]　山口 実靖[3]　小口 正人[1]

概要：近年スマートフォンや IoT デバイスが普及し, 低遅延やネットワークの負荷分散が可能といった利点を持つエッジコンピューティングに注目が集まっている. 現在行われているこの分野の研究では, デバイス側はセンシングと通信を行うだけの位置付けとなっているものが多い. 一方で, 高性能な CPU や GPU を搭載したデバイスが登場し, これらのリッチクライアントを用いてデータの発生源であるデバイス側で処理を行うと, プライバシの保護や通信コストの削減が可能といった点で優れたシステムを構築できると考えられる. そこで, リッチクライアント上でも学習を行うエッジコンピューティングモデルの検討を行う. 本稿では, 機密性が高く, 容量の大きな顔画像を用いて実験を行い, 提案モデルにおいてプライバシや通信コスト, 負荷分散の点で利点があることを確認した.

A Study of Image Recognition in Distributed Machine Learning using Rich Clients

SAKI TAKANO[1]　AKIHIRO NAKAO[2]　SHU YAMAMOTO[2]　SANEYASU YAMAGUCHI[3]
MASATO OGUCHI[1]

1. はじめに

スマートフォンの普及および性能向上により, スマートフォン上に膨大なデータが蓄積されるようになった. また, Internet of Things (IoT) の普及によりエッジデバイスから生成されるデータが年々増大しており, 今後 IoT を用いたサービスが普及するにつれ, さらなる増加が予想される. 近年, スマートフォン上に提示されるおすすめ表示や画像認識などと様々な場面で機械学習が活用されるようになり, エッジデバイスで発生する大量のデータに対して低遅延で複雑な処理を行い, エッジデバイスで収集したデータも含めた機械学習が期待される.

現在主流となっているクラウドコンピューティングではユーザは地理的に遠く離れたクラウドにデータを送信し, クラウド内で処理された結果を応答として受け取る. しかし, エッジデバイス-クラウド間の遅延は数百ミリ秒に及ぶ場合があり, 帯域も多く必要とするため, リアルタイムアプリケーションや大量のデータを送受信するアプリケーションの実装には向いていない.

そこで注目されているのが, エッジデバイスに近いネットワークエッジにエッジサーバを配置して処理を行うことにより, 低遅延でサービスを提供できるエッジコンピューティングである [8]. エッジコンピューティングは, 遠隔にあるクラウドのサーバと比較して物理的に近い位置で処理を行うことにより，往復の通信遅延時間が低減され, ユーザの体感する応答性や操作性を向上させることが可能となる. また, エッジサーバで分散処理を行うことにより, エッジ-クラウド間におけるネットワークの負荷を削減することも可能である.

一方で, エッジコンピューティングの課題の一つにエッジデバイスが収集した生データの取り扱い方がある. 生データを機械学習処理のために収集元であるエッジデバイスからエッジサーバへと送信すると, データをエッジサーバなどデバイスの外部へと受け渡すことによるプライバシ

[1] お茶の水女子大学
Ochanomizu University
[2] 東京大学
the University of Tokyo
[3] 工学院大学
Kogakuin University

の問題や通信コストが高くなるという問題があり, ユーザ認証プロトコルの導入 [5] やエッジデバイス上でのデータの圧縮・特徴量の抽出 [13] などが考えられている. このように従来のエッジコンピューティングの研究においてエッジデバイス上でのデータの加工は考えられているものの, 性能の低いエッジデバイス側はあくまでデータを収集し, そのデータをエッジサーバに転送するという役割を果たしてきた.

しかし, エッジデバイスの性能向上により, CPU や GPU が搭載され, エッジデバイス内でも機械学習を動かすことのできる程の性能を持つリッチクライアントが登場したことで, より複雑なタスクもエッジデバイス上で実行することが可能となった. 生データではなくエッジデバイス上で行った学習の結果をエッジデバイス-エッジサーバ間で受け渡すことでプライバシや通信コストの問題の改善につながることが期待できる.

そこで, 我々はエッジデバイス上で機械学習を行い, その学習結果をエッジサーバと共有することでさらに学習を進め, より良い学習結果を得ることができる分散機械学習モデルを検討している. MNIST を用いた実験では, エッジサーバに学習を引き継いで高い精度の結果を得ることができた [12]. 本稿では, 実際にスマートフォンや IoT デバイス上での学習で用いられることが想定される顔画像データを用いて検討を行う.

本稿の構成は以下の通りである. 第 2 章で関連研究として提案モデルの元のアイディアとなるエッジ/フォグコンピューティングおよびリッチクライアントを用いた分散機械学習の例として Federated learning を紹介する. 第 3 章でエッジコンピューティングの課題について述べ, 第 4 章で解決手法を提案する. 第 5 章で提案モデルの実装と評価を行い, 第 6 章で考察を述べ, 第 7 章でまとめる.

2. 関連研究

2.1 エッジ/フォグコンピューティング

エッジコンピューティングとは, ネットワークエッジにエッジサーバを配置し, データ処理を最大限エッジで行うコンピューティングモデルである. エッジコンピューティングの利点としては, クラウドコンピューティングと比較して低遅延である点やエッジデバイスで処理を行うことでクラウドサーバにかかる負荷を分散できる点, エッジデバイスからクラウドサーバへ送信されるデータ量を削減し, トラフィックの混雑を解消できる点が挙げられる.

論文 [10] ではエッジコンピューティングと似たモデルであるフォグコンピューティングについて一般的なモデルとアーキテクチャについて分析し, クラウドコンピューティングでは数十億のデバイスとクラウド間の長距離通信には通信遅延と帯域幅の圧迫という 2 つの問題が生じるが, クラウドで処理していたタスクをネットワークエッジに設置したフォグサーバにオフロードすることで解決されることが示されている.

このような利点を活かし, エッジ/フォグコンピューティングはスマートシティ [3] や高度道路交通システム [6] などで IoT アプリケーションに応用され, クラウドコンピューティングでは実装することができなかったリアルタイムに応答するシステムが構築されている.

2.2 Federated learning

近年, デバイスの性能向上により, 高性能な CPU や GPU を搭載したリッチクライアントが登場し, IoT デバイス上でサーバが行なっていた機械学習の処理を実行できるようになった. デバイス上で機械学習を行うモデルとして, Federated learning(連合学習) という分散型機械学習が提案された [9]. Federated learning では, まずクラウド上のデータで学習を行って得られた学習モデルを各デバイスに配布し, 各デバイスはそれぞれが収集した固有のデータを利用してさらに学習を進めた上で変更点の情報のみをクラウドに送信する. そして, クラウドは各デバイスから収集した変更点を平均化し, 元の学習モデルを改善してより良いモデルを作成する. このように各デバイスで収集したデータをデバイスの外部に受け渡さないため, プライバシを担保しつつデバイスにあるデータを機械学習に活用することが可能となる. Federated learning はエッジコンピューティングとは異なり, プライバシに配慮しながらエッジデバイスの情報をクラウドに集約し, クラウドが一括管理するコンピューティングモデルとなっている.

論文 [11] では, Federated learning を Google キーボードに応用した例が実装されており, デバイスの持つ固有のデータを受け渡すことなく, デバイス-クラウド間にまたがる分散機械学習が可能であることが示されている.

しかし, Federated learning におけるプライバシの保護は十分であるとは言えず, 論文 [4] ではクラウドに送信されるパラメタから元画像を鮮明に復元できてしまうことが示されている.

3. 研究課題

エッジコンピューティングにおいて機械学習を行う際に課題となっている問題について以下に示す.

(1) プライバシの保護が必要である.

エッジデバイスで収集したデータはほとんどの場合エッジデバイスのみが所有権を有するものであるため, ネットワークを介した通信やエッジサーバとのデータ共有のためにはプライバシを保護できるセキュリティを確保する必要が出てくる. この問題の解決方法として, エッジデバイスの外部へと送信する情報を学習の重みのみにすることができると, その情報から元の

データを復元することが難しいため, プライバシの保護が可能となる. しかし, 従来の研究で想定されている性能の低いエッジデバイスでは機械学習を動かすことができないため, このような実装はなされていない.

(2) プライバシや通信環境の問題からデータをエッジサーバに渡すことができない場合には学習を行うことができない.

近年, 欧州で EU 一般データ保護規則 (GDPR, General Data Protection Regulation) [1] が定められるなど, プライバシ保護への関心が高まっており, エッジデバイスで収集される個人データをサーバへ受け渡すことへの抵抗がさらに大きくなると予想できる. しかし, 従来のエッジコンピューティングモデルでは, 機密性が高くエッジデバイスの外へと持ち出したくないデータを用いたい場合に対応することができない. また, 従来のモデルではエッジデバイスが一時的にネットワークに繋がっていない場合, 取得データを即時的に活用したい場合にも対応することができない. この問題は, 必要に応じてエッジサーバへ情報を送るか送らないか判断して選別することができる形を作ることで解決することができるが, 従来のエッジサーバやクラウドが学習結果を集約するモデルでは実装が難しい.

(3) 通信コストの削減が必要である.

従来のエッジコンピューティングではエッジデバイス-エッジサーバ間において生データやそれに少し加工を施した容量の大きなデータの受け渡しが行われている. IoT アプリケーションの普及に対応するためにはネットワークを介して受け渡されるデータに対するさらなる工夫が必要である.

(4) エッジサーバの負荷を分散する必要がある.

1 台のエッジサーバに接続されるエッジデバイスの台数はクラウドコンピューティングの際と比較すると少ないものの, IoT デバイスの普及により場合によっては数千から数万台となり, エッジサーバに大きな負荷がかかることで遅延が発生すると予想される. 従来のエッジコンピューティングにおいてエッジサーバが行っていた学習の一部をエッジデバイスにオフロードすることでエッジサーバにかかる負荷を低減することが望ましいと考えられる.

(5) 各エッジデバイスの持つデータに適した学習結果でない可能性がある.

既存研究では全てのエッジデバイスに対して, クラウドやエッジサーバ上の一般的なデータで学習を行い高精度となった同一のモデルを配布している. しかし, エッジデバイスはエッジサーバと異なる特有なデータを持つ場合があり, 配布された一般的なモデルでは精度が低い可能性がある.

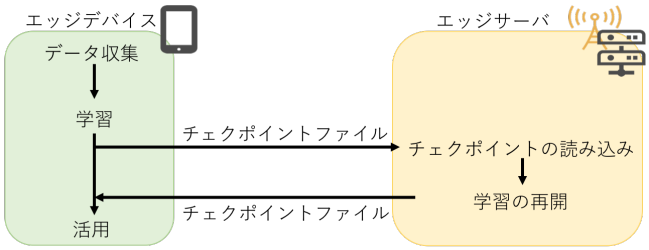

図 1: 提案モデル

4. 解決手法の提案

リッチクライアントの登場により, 機械学習等の複雑な処理もエッジデバイス上で行うことが可能になったことと合わせ, 上記の課題の解決を目指したリッチクライアントに適した分散機械学習モデルを提案する. 具体的には, エッジコンピューティングモデルにおいて, 従来エッジサーバ上で行っていたタスクの一部をエッジデバイスにオフロードすることでエッジデバイス上でも機械学習処理を行う. さらに, そこで得られた学習結果をエッジサーバと共有することで学習を進め, より良い学習結果をエッジデバイスで利用することのできるモデルを構築する.

提案モデルの概要図を図 1 に示す.

まず, エッジデバイスで収集したデバイス固有のデータを用いて学習を行う. ここで得られた結果は, エッジデバイス上で即時に利用することも可能であるが, より精度の高い結果を得たい場合には学習をエッジサーバへと引き継ぐ. この際, エッジサーバには生データは送信せず, エッジサーバで学習を再開させるために最低限必要な学習の重みを保存したチェックポイントファイルを送信する. エッジサーバは, 受け取ったチェックポイントファイルを読み込み, エッジサーバの持つデータで学習を再開する. 得られたチェックポイントファイルをエッジデバイスへと送信することで, エッジデバイスがより学習の進んだ結果を利用できると考えられる.

一方でエッジデバイスにはユーザ特有のデータが含まれている可能性や, データに偏りが生じている可能性があり, 学習をエッジサーバへと引き継いで一般的なデータで学習することで, 逆に学習精度が下がってしまうという懸念が生じる. そこで, エッジデバイスとエッジサーバで学習するデータに偏りがある場合にも対応できる改善モデルとして 図 2 を提案する. エッジデバイス上において, エッジサーバでの学習後に受け取ったチェックポイントファイルとその時点でのエッジデバイスでの学習結果を保存したチェックポイントファイルをそれぞれ読み込む. 精度を比較し, より精度の高い学習結果を利用する.

サーバに頼ることでより精度の上がった結果を使用する

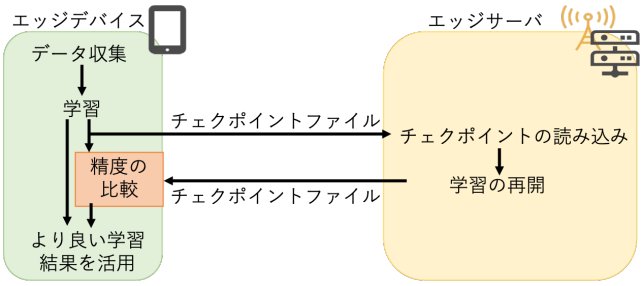

図 2: 学習データの偏りに対応した提案モデル

表 1: エッジサーバの性能

OS	Ubuntu 18.04 LTS
CPU	Intel Core i7-8700
GPU	GeForce RTX 2080Ti
Memory	32Gbyte

表 2: Jetson Nano の性能

OS	Ubuntu 18.04 LTS
CPU	Quad-core ARM A57 @ 1.43 GHz
GPU	128-core Maxwell
Memory	4 GB 64-bit LPDDR4 25.6 GB/s

ことができる一方で, エッジサーバに学習を引き継いで精度が落ちてしまった場合にはエッジデバイスのみで学習したより良い精度の結果を利用することができるモデルとなる.

5. 提案手法の実装と評価

5.1 データセット

インターネット上より集められた jpg 画像を人物ごとにフォルダ分けしてある Labeled Faces in the Wild (以下 lfw) [2] から, 1 人あたりの写真が 30 枚を超える人物について 30 枚ずつ抜き出して使用した. 33 人分のフォルダが作成され, 内訳は男性 28 人, 女性 5 人となった. 各写真について顔抽出を行い, 適切に抽出を行うことのできていない写真を取り除いた後, 各フォルダの 2 割を test データとした. 残りの写真を train データとし, ぼかし等の加工により 9 倍に水増しした.

5.2 実験環境

実験で使用したエッジサーバの性能を表 1 に, エッジデバイスとして使用した Jetson Nano の性能を表 2 に示す.

Jetson Nano は GPU を搭載した小型 AI コンピュータボードであり, 近い将来, スマートフォンや様々な IoT デバイスがこのような性能を持つことが期待される. しかし, 性能はサーバと比較すると劣り, GPU のコア数がサーバは 4352 コアであるのに対し, Jetson Nano は 128 コアと大きな差がある.

本実験では分散処理に適している TensorFlow を機械学習に使用し, Jetson Nano -エッジサーバ間はイーサネットで接続した.

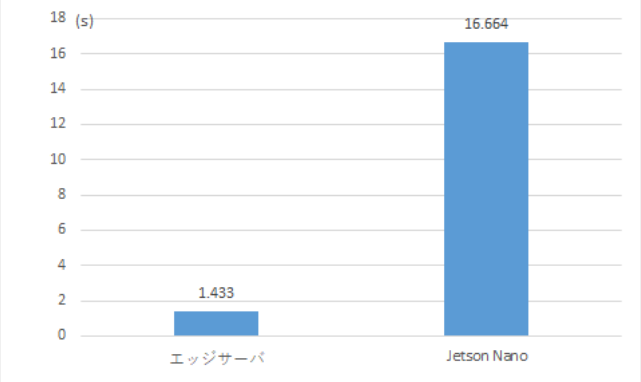

図 3: エッジサーバ, エッジデバイスによる MNIST の実行時間

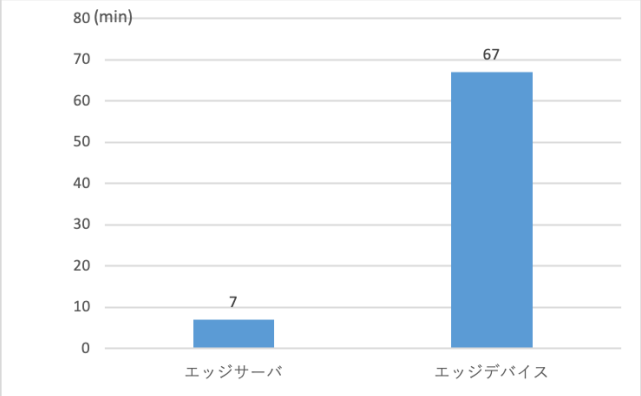

図 4: エッジサーバ, エッジデバイスによる lfw の実行時間 (同一 epoch 数)

5.3 予備実験

エッジサーバとエッジデバイスにおいて機械学習処理を行った際の実行時間を比較する.

MNIST において同一の学習をした結果を図 3 に示す. エッジサーバと比較してエッジデバイスは, およそ 12 倍の時間を要する. 一方で, この実験では MNIST を epoch 数を 5, 各 step 数を 200 で学習したが, エッジサーバ, エッジデバイス共におよそ 96%と同等精度で学習を行うことが可能であった.

次に, lfw を用いて同一の学習をした結果を図 4 に示す. エッジサーバと比較してエッジデバイスは, およそ 10 倍の時間を要し, エッジサーバでの精度は 65%, エッジデバイスでの精度は 56%となった. MNIST を用いた際とは異なり, 同一の学習を行っているにもかかわらず, 性能の低いエッジデバイスの精度がエッジサーバと比較して低い結果となった. しかし, 顔画像を 33 人に分類する実験であり, 学習を全く行わない際の精度は 3%であるため, 学習を行うことができていることが読み取れる.

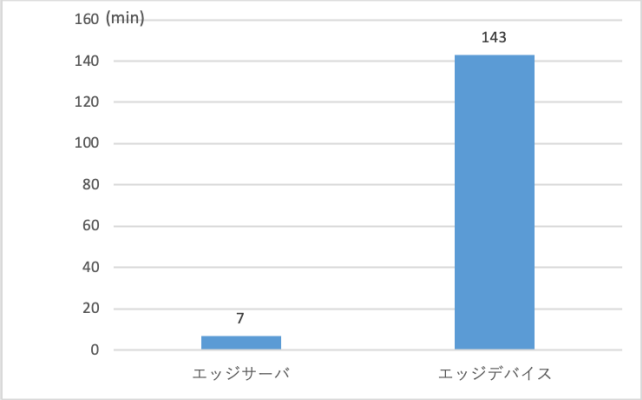

図 5: エッジサーバ, エッジデバイスによる lfw の実行時間 (同一精度)

lfw を用いてエッジデバイス, エッジサーバ共に精度が 65%となるよう学習をした結果を図 5 に示す. エッジデバイス上でもエッジサーバと同等精度の学習を行うことができるものの, およそ 20 倍の時間を要し, lfw を用いた学習では 65%の精度を得るために 2 時間以上の学習が必要となる.

このことから, エッジデバイスは低速ではあるが, エッジデバイス内のみでも学習可能であることが分かり, プライバシが非常に重要なデータもそのような形で学習に用いる事ができる. しかし, エッジデバイスのみでの学習では限界があり, エッジサーバとの連携が重要になると考えられる.

5.4 実験 1 (同一データによる学習)

5.4.1 実験概要

エッジデバイス, エッジサーバに作成した同一の顔画像データセットを与えて学習を行った. まずはじめにエッジデバイス上で epoch 数を 10, 各 epoch の step 数を 219 で学習を行い, 得られたチェックポイントファイルのみをエッジサーバへと転送する. エッジサーバ側では, 受け取ったチェックポイントファイルを読み込み, epoch 数を 20, 各 epoch の step 数を 219 で学習を再開させ, 得られたチェックポイントファイルをエッジデバイスへと送信する. ここでは, エッジサーバはエッジデバイスと比べ高性能であり, 短時間で学習を行うことができるため, より多くの学習を行うようにした.

5.4.2 実験結果

エッジデバイスでの学習後にエッジデバイス上で計測した精度 (①), そこで得られた結果をエッジサーバ上で計測した精度 (②), エッジサーバでさらに学習を行った後にエッジサーバ上で計測した精度 (③), およびその結果をエッジデバイス上で計測した精度 (④) を図 6 に示す. 精度は作成した lfw の test データを用いて計測した.

エッジデバイス上では①で示すように 44%まで学習できたことが分かり, そこで得られたチェックポイントをエッジサーバへと渡し, エッジサーバ側で計測すると精度は②で示すように同じく 44%となった. よって, エッジデバイスからエッジサーバへの学習結果の受け渡しにより, 学習精度は落ちないことが読み取れる. この時点で得られた結果は十分な精度であるとは言えないが, エッジサーバで学習を再開させることで③で示すように 65%の精度を得ることができ, ここで得られたチェックポイントをエッジデバイスに渡し, そこで精度を計測すると④で示すように同じく 65%と良い結果を得ることができた.

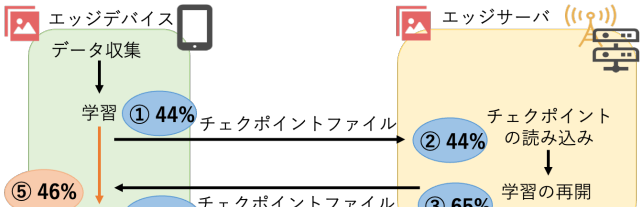

図 6: 同一データを与えた際の学習精度 (実験 1)

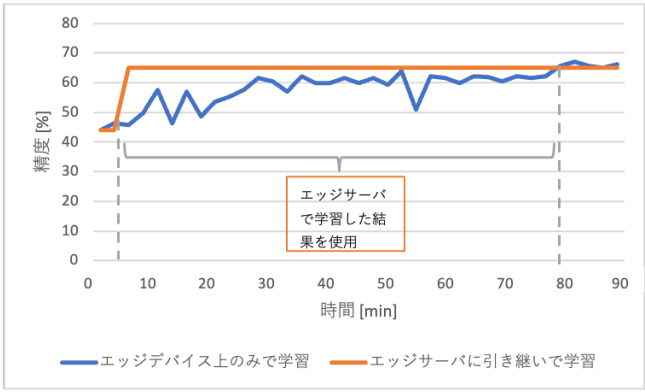

図 7: 同一データを与えた際の提案モデルの実行 (実験 1)

また, エッジデバイス上での学習結果をエッジサーバへ引き継いで学習を行い, エッジデバイスにその結果を返す間, エッジデバイス上で引き続き学習を進めると⑤で示すように学習精度は 46%となった.

5.4.3 提案モデルの実行

提案モデルを実行した際の動作を, エッジサーバにチェックポイントファイルを引き継ぐ前の学習が終了した時刻からの経過時間を横軸として図 7 に示す.

エッジデバイスがエッジサーバで引き続き学習した結果を受け取るまでの間はエッジデバイス上のみで学習した結果を即時的に使用する. その後エッジサーバ上で学習したより高い精度を持つ学習結果を受け取り, その結果を選択して活用する. エッジデバイス上のみでの学習でエッジサーバで学習した結果と同等の精度である 65%を得たい場合にはおよそ 80 分学習を行う必要がある. このことから, 提案モデルにおいてエッジデバイスがエッジサーバで学習した結果を得ることで, より高い精度の学習結果を早くに利用できることが分かる.

5.5 実験 2 (同種で異なる量のデータによる学習)

実験 1 ではエッジデバイス, エッジサーバに同一のデータを与えているが, 実際に想定されるのはエッジデバイスはそのデバイスで収集した独自のデータ, エッジサーバは様々なエッジデバイスから取得した一般的なデータという環境である. その結果エッジデバイス, エッジサーバで使用できるデータ量に差が生じる. 本実験ではエッジデバイス, エッジサーバに与えるデータ量を変化させ, その影響について考察する.

5.5.1 実験概要

実験 1 の実験環境においてエッジデバイス, エッジサーバに与えるデータ量を以下のように変化させる.

(1) 各人物の全写真のうち 2 割をエッジデバイスに, 残りの 8 割をエッジサーバに与える

これはエッジサーバに多くのデータが集約されているケースである. エッジサーバはエッジデバイスと比較し, メモリ容量が大きく, 様々なデバイスとつながることでより多くのデータを保持していることが容易に想定できる.

ここではまずエッジデバイス側で epoch 数を 50, 各 epoch の step 数を 45 で学習を行い, エッジサーバ側では epoch 数を 25, 各 epoch の step 数を 174 で学習を再開させる.

(2) 各人物の全写真のうち 8 割をエッジデバイスに, 残りの 2 割をエッジサーバに与える

これはエッジデバイスにより多くのデータが存在するケースである. 最新のデータはデータの収集源であるエジデバイスの方がより多く保持していることが予想でき, このようなケースが想定される.

ここではまずエッジデバイス側で epoch 数を 12, 各 epoch の step 数を 174 で学習を行い, エッジサーバ側では epoch 数を 100, 各 epoch の step 数を 45 で学習を再開させる.

テストは実験 1 と同様の test データを用いて行う.

5.5.2 実験結果

上記で示した (1), (2) それぞれについて以下で結果をまとめる.

(1) エッジデバイス：2 割, エッジサーバ：8 割

各状態での学習精度を図 8 に示す.

エッジデバイス上において少ないデータで学習を行い 43%の精度を得た後, チェックポイントファイルをエッジサーバへと送信し精度を計測すると, 同じく 43%の精度を得ることができた. エッジサーバ上での学習によりエッジデバイス, エッジサーバ共に精度は 62%となった.

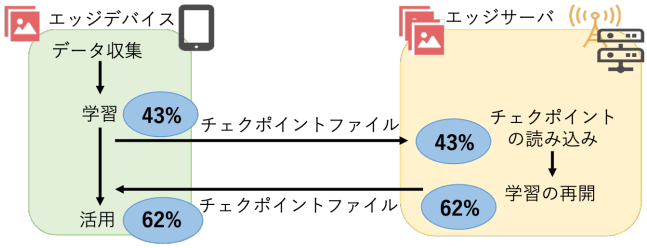

図 8: エッジデバイスに 2 割, エッジサーバに 8 割のデータを与えた際の学習精度 (実験 2(1))

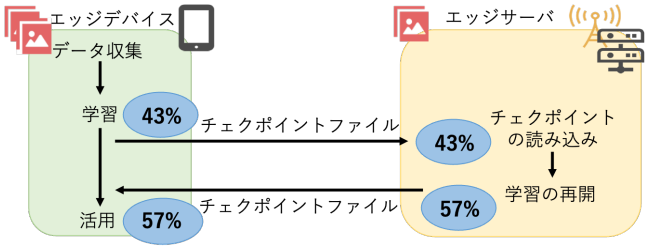

図 9: エッジデバイスに 8 割, エッジサーバに 2 割のデータを与えた際の学習精度 (実験 2(2))

(2) エッジデバイス：8 割, エッジサーバ：2 割

各状態での学習精度を図 9 に示す.

エッジデバイス上ににおける学習で 43%の精度を得た後, チェックポイントファイルをエッジサーバへと送信し精度を計測すると, 同じく 43%の精度を得ることができた. エッジサーバ上での学習によりエッジデバイス, エッジサーバ共に 57%の精度を得ることができた. このケースではエッジサーバの保持するデータ量が少ないため, エッジサーバ上での学習で実験 1 や実験 2(1) ほど高い精度を得ることはできなかった.

さらに, エッジデバイス上での学習およびエッジサーバ上での学習で得られたチェックポイントファイルは全て 16Mbyte であった. 一方で, 学習を全てエッジサーバで行うためにエッジデバイスで収集したと想定する lfw の画像データをエッジサーバへと渡すと, 加工を行わない画像では (1) で 28Mbyte, (2) で 112Mbyte, 顔を抜き出した画像では (1) で 6Mbyte, (2) で 23Mbyte 送信することとなり, 画像サイズが大きい場合やデータ数が多い場合にはチェックポイントファイルを利用することで通信データ量が削減できる結果となった.

5.5.3 提案モデルの実行例

提案モデルを実行した際における (1), (2) それぞれの動作を図 10, 図 11 に示す.

エッジデバイスがエッジサーバで引き続き学習した結果を受け取るまでの間はエッジデバイス上のみで学習した結果を即時的に使用し, エッジサーバ上で学習したより高い精度を持つ学習結果を受け取ると その結果を選択して活用する.

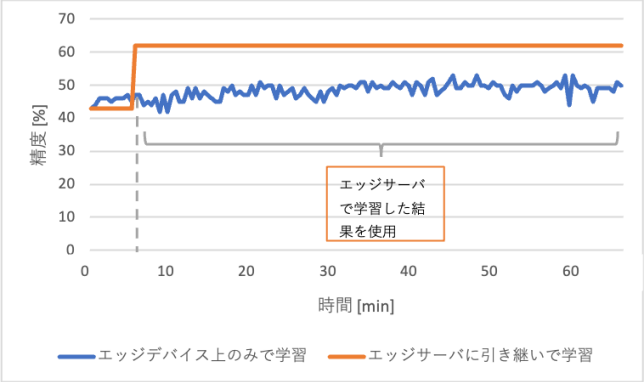

図 10: エッジデバイスに 2 割, エッジサーバに 8 割のデータを与えた際の提案モデルの実行 (実験 2(1))

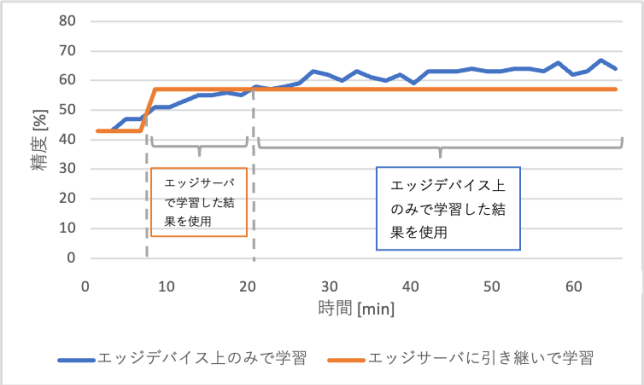

図 11: エッジデバイスに 8 割, エッジサーバに 2 割のデータを与えた際の提案モデルの実行 (実験 2(2))

エッジデバイスの保持するデータ量の少ない (1) では, 長時間学習を行った場合であっても 50%程度までしか精度が上がらない. そのため エッジサーバで学習した結果の方が良い精度となり, それを使用し続ける選択をする.

一方でエッジサーバの保持するデータ量の少ない (2) において, エッジサーバ上での学習では 57%の精度しか得ることができないが, 時間はかかるもののエッジデバイス上での学習では 64%まで精度を上げることが可能であった. そのため, エッジデバイスのみでの学習で得られた精度がエッジサーバから受け取った学習結果の精度を上回ると, エッジデバイスのみでの学習で得られた結果を使用する選択をする.

5.6 実験 3 (個人データを含む学習)

実験 1 および実験 2 ではエッジデバイス, エッジサーバ共に全ての人物のデータを与えている. しかし, エッジデバイスには個人の顔写真など機密性が高くエッジサーバに情報を一切渡したくないデータが含まれている可能性があり, エッジサーバで使用できるデータの種類が限定されることが想定される.

本実験では, 実験 1 で使用したデータセットに自分の顔写真を加え, それが学習結果に及ぼす影響について考察する.

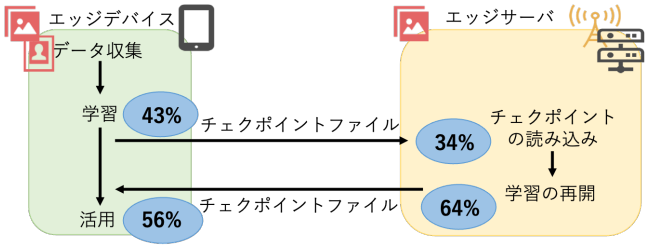

図 12: 個人データが含まれている際の学習精度 (実験 3)

5.6.1 実験概要

自分の顔を切り抜いた画像を 149 枚用意し, そのうちの約 2 割である 29 枚を test データに加えた. 残りの 120 枚はぼかし等の加工により 9 倍に水増しし, train データに加えた. 自分のデータは test データ, train データ共に約 14%を占めていることとなった. このデータセットをエッジデバイスに与え, エッジサーバには実験 1 と同じ個人データの含まれないデータセットを与える.

はじめにエッジデバイス上で epoch 数を 6, 各 epoch の step 数を 252 で学習を行い, 得られたチェックポイントファイルのみをエッジサーバへと転送する. エッジサーバ側では, 受け取ったチェックポイントファイルを読み込み, epoch 数を 20, 各 epoch の step 数を 219 で学習を再開させ, 得られたチェックポイントファイルをエッジデバイスへと送信する.

5.6.2 実験結果

各状態での学習精度を図 12 に示す.

エッジデバイス上での学習後, 個人データの含まれる test データを用いて精度を計測すると 43%であった. 得られたチェックポイントをエッジサーバへと渡し, エッジサーバ側で個人データの含まれない test データを用いて計測すると精度は 34%と下がった. エッジサーバで個人データの含まれない train データを使用して学習を再開させることで個人データの含まれない test データを用いた際の精度が 64%となり, ここで得られたチェックポイントをエッジデバイスに渡し, 個人データの含まれる test データを用いて精度を計測すると 56%となった.

一方で, 個人データに関しての精度を計測するために個人データのみが含まれる test データを作成し精度を計測すると, エッジデバイスでの学習後には 93%と高い精度を得ることができるが, エッジサーバでの学習後には 3%と全く学習を行わない際と同等の精度となった. よって, エッジサーバでの学習によりエッジデバイスでの学習が上書きされ, 個人データに関する精度が下がったと考えられる.

5.6.3 提案モデルの実行

提案モデルを実行した際の動作を図 13 に示す.

エッジデバイスがエッジサーバで引き続き学習した結果を受け取るまでの間はエッジデバイス上のみで学習した結果を即時的に使用し, エッジサーバ上で学習したより高い

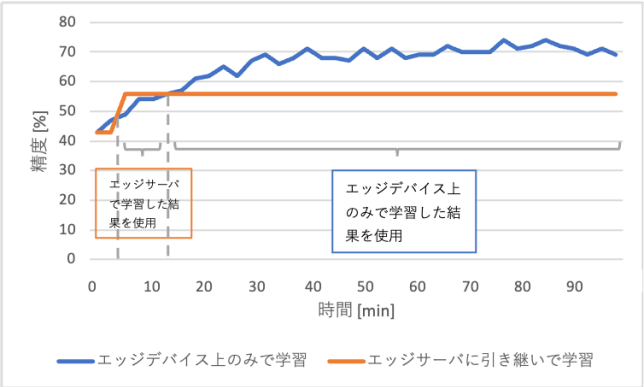

図 13: 個人データが含まれている際の提案モデルの実行 (実験 3)

精度を持つ学習結果を受け取ると その結果を選択して活用する. 個人データを含まないデータを用いたエッジサーバでの学習では 56%の精度しか得ることができない. 一方で, エッジデバイス上で個人データを含むデータを用いて学習を進めることで, そのエッジデバイスに適した学習を行うことができ, 高い精度を得ることが可能である. エッジデバイス上のみで学習を行った結果の精度がエッジサーバから受け取った学習結果の精度を上回ると, エッジデバイスのみでの学習で得られた結果を使用する選択をする.

6. 結論

エッジデバイス上のみでの学習では精度が不十分な場合にエッジサーバへと学習を引き継ぐことで, 短時間でより精度の高い学習結果を得ることができ, 本提案モデルを用いることで本研究の課題としてあげたプライバシや通信コストの面で利点がある. また, エッジデバイス上で学習を行うことにより, エッジデバイス上の機密性の高いデータも含んだ学習を行うことが可能となり, エッジデバイスのデータに適した学習を行うことができる.

7. まとめと今後の課題

従来のエッジコンピューティングで課題となっているプライバシの保護や通信コストの削減を目的として, リッチクライアントに適した分散機械学習モデルの検討を行った.

従来のエッジコンピューティングモデルにおいて, エッジデバイスでも機械学習処理を行い, 学習結果をエッジサーバと共有することでさらに学習を進めるモデルを提案し, 学習データとして顔画像データ, エッジデバイス側に Jetson Nano を用いて実験を行なった. その結果, エッジサーバに学習を引き継いで高い精度の結果を得ることができ, 生データをエッジデバイス上に留めておくことでプライバシの保護および通信コストの削減が可能であることが示された. また, エッジデバイスがエッジサーバと異なるデータを持つ際にエッジサーバ上での学習結果を用いることで精度が下がるという問題が生じるが, エッジデバイス上で精度の比較を行うことでより良い学習結果を使用できることを示した.

今後は Federated learning において研究が進められている偏りのあるデータに対応する方法 [7] を参考にしつつ, 偏りのあるデータに対応したより良いモデルの検討を考えている.

謝辞

本研究は一部, JST CREST JPMJCR1503 の支援を受けたものである.

参考文献

[1] General Data Protection Regulation. `https://eur-lex.europa.eu/eli/reg/2016/679/oj`. (2021/04 閲覧).

[2] Labeled Faces in the Wild. `http://vis-www.cs.umass.edu/lfw/`.

[3] N. Chen, Y. Chen, S. Song, C. Huang, and X. Ye. Poster abstract: Smart urban surveillance using fog computing. In *2016 IEEE/ACM Symposium on Edge Computing (SEC)*, pp. 95–96, 2016.

[4] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. Inverting gradients – how easy is it to break privacy in federated learning?, 2020.

[5] Junho Lee, Dongwook Kim, Jinhyun Park, and Hyungweon Park. A multi-server authentication protocol achieving privacy protection and traceability for 5g mobile edge computing. *Proc. of the 39th IEEE International Conference on Consumer Electronics (ICCE 2021)*, January 2021.

[6] J. Lin, W. Yu, X. Yang, Q. Yang, X. Fu, and W. Zhao. A real-time en-route route guidance decision scheme for transportation-based cyberphysical systems. *IEEE Transactions on Vehicular Technology*, Vol. 66, No. 3, pp. 2551–2566, 2017.

[7] Mehryar Mohri, Gary Sivek, and Ananda Theertha Suresh. Agnostic federated learning, 2019.

[8] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella. On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration. *IEEE Communications Surveys Tutorials*, Vol. 19, No. 3, pp. 1657–1681, 2017.

[9] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications, 2019.

[10] S. Yang. Iot stream processing and analytics in the fog. *IEEE Communications Magazine*, Vol. 55, No. 8, pp. 21–27, 2017.

[11] T. Yang, G. Andrew, Hubert Eichner, Haicheng Sun, W. Li, Nicholas Kong, D. Ramage, and F. Beaufays. Applied federated learning: Improving google keyboard query suggestions. *ArXiv*, Vol. abs/1812.02903, , 2018.

[12] 高野紗輝, 中尾彰宏, 山本周, 山口実靖, 小口正人. リッチクライアント-エッジサーバ間での分散機械学習に関する一検討. 第 13 回データ工学と情報マネジメントに関するフォーラム (DEIM2021).

[13] 塩田純, 滝澤允, 田中裕之, 高橋紀之, 小林英嗣. 画像処理におけるエッジコンピューティングを用いた垂直分散処理方式の検討. Technical Report 2, 日本電信電話株式会社, 日本電信電話株式会社, 日本電信電話株式会社, 日本電信電話株式会社, nov 2016.