

事前予測による物体検出の推論実行効率化

田中美帆¹ 鈴木貴久¹ 豊永慎也¹ 松倉隆一¹

概要： 機械学習の認識精度の向上により、日常のさまざまな場面で機械学習が用いられるようになってきた。例えば、道路や交差点に設置されるカメラにより、渋滞や事故などの状況を人間に代わって判断し、通知できる。しかしながら、GPU は高価であるため、GPU の処理を効率化して相対的にコストを下げる 것이 重要となる。意味のあるイベントは頻繁には発生しないことに着目し、簡易な方法で認識結果を予測することで、変化の少ないフレーム画像の認識処理を回避・抑制する方式を提案した。実現における課題は次の3つである。(1)前のフレーム画像と同じ推論結果が得られる画像(類似画像)の簡易判定方法。(2)GPU の推論処理と競合しないシステム全体のスケジューリング。(3)GPU での推論処理の上限に合わせたフレーム画像数の調整、である。評価では、高速道路のカメラ画像(25fps)から走行中の自動車台数を計測するケースを想定した。既存研究では、カメラ画像を 4fps に間引きし 4 並列で処理したが、本方式では 25fps のカメラ画像を類似度判定することができた。この結果、評価で利用したカメラ画像では、GPU 上での推論処理対象のフレームレートが 25fps から 1.3fps に変換され、同時に 8 台のカメラ画像を処理可能であることが示された。全体として 12.5 倍 (25fps を 8 並列、200fps)の処理向上が得られた。

Efficient Inference Technique with Pre-processing for Object Detection

MIHO TANAKA¹ TAKAHISA SUZUKI¹
SHINYA TOYONAGA¹ RYUICHI MATSUKURA¹

1. はじめに

機械学習の認識精度の向上により、日常のさまざまな場面で機械学習が使われるようになってきた。例えば、各種センサ、ビルに設置される照明や空調装置、工場における生産装置、監視カメラの映像を機械学習で分析し、混雑や事故を予測したり、ビルのエネルギー消費を削減したり、工場での生産管理や設備管理、街の見守りなどに役立てられている。従来は、装置の状態を定期的に確認したり、カメラ映像を常に観察したりして、人が検出を行っていた。しかし、人手による検出には限界があり、また、担当する人が習熟していないと判断ミスや見落としが発生するため、機械学習への期待は高い。

機械学習を利用するユースケースとして、既に設置されている監視カメラの画像分析が注目されている。近年、低コストで解像度の高いカメラが提供されている。しかし、多くのカメラ映像は、多くのモニタに映し出された映像をリアルタイムに監視する以外では、蓄積されるだけでトラブル等が発生したときに後から確認するために利用される。機械学習により、特に映像に映っている物体を検出することで、リアルタイムに状況を通知するサービスが実現されるようになってきた。例えば、道路に設置された監視カメラの動画映像を用いて渋滞状況を検出したり、交差点における歩行者や車両の動きを検出して交通事故対策に利用することが可能である。

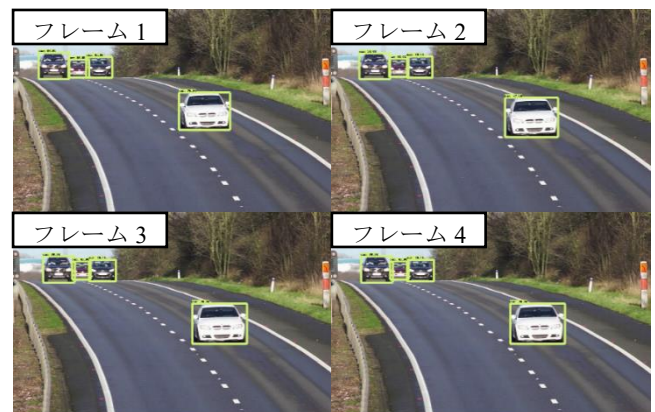


図1 高速道路の映像の連続したフレーム

カメラ画像の物体検出については、ニューラルネットワークにより複雑な推論モデルを実行可能になることにより、精度が格段に高くなった。また、GPU を利用して機械学習を処理するようになり、CPU より高速に処理することができる。しかし、カメラから入力される 30fps または 60fps の HD 画像をリアルタイムに推論処理することは、GPU の処理量を越えることが多い。GPU 性能は今後も上がると考えられるが、今後は GPU 及び CPU の計算資源の範囲内で、推論処理が最も効率良く実行されるようなアーキテクチャを考慮する必要がある。

また、カメラ画像の物体検出により検出するイベントは、それほど頻繁に発生するものではなく、どちらかという

¹ 富士通株式会社
Fujitsu Limited

まれに発生するイベントが多い。例えば、道路で発生する渋滞は、時間によると1日の5%以下[1]という報告もあり、渋滞を検出するシステムでは、30fpsで得られる全てのフレーム画像を物体検出することは無駄になる部分が多い。また、検出したい対象物のフレーム画像内の移動速度によっては、30fpsで得られた全ての画像を処理するのではなく、最初に10フレーム毎(3fps)の映像を推論処理することもある。一方で、推論処理に必要なフレーム画像のサイズは、使用するアルゴリズムによって異なるが、一般のカメラ画像の解像度に比較するとかなり小さい。そのため、カメラ画像の解像度を予め小さくしておいて、推論処理量を減らす工夫も必要である。

以上述べたように、GPUの効率化を実現するには、GPU内部の処理効率を高めるだけでなく、GPUに入力されるフレーム画像の入力頻度や解像度(画像サイズ)調整して、GPUの推論性能を最大に利用することが必要である。GPUの入力フレーム画像の条件は、ユースケースと推論アルゴリズムの組み合わせで決まる。そこで本稿では、ユースケースを絞り込み、GPUでの推論処理よりも簡易な方法でフレーム画像の変化を予測することで、認識結果が変化する可能性のあるフレーム画像のみをGPUで推論処理する方式を提案する。

本稿の構成を以下に示す。2章で前提とするユースケースや関連研究を紹介し、3章で従来手法の課題について示す。4章で提案方式の概要、5章で提案方式の評価内容および結果を示し、6章で本稿の結論と今後の課題について述べる。

2. 関連研究

カメラ画像を利用して推論処理する技術については、既に多くの研究成果がある。本章では、カメラ画像を、GPUを利用した物体検出を実現する機械学習のアルゴリズムに関わる技術について説明する。これらのアルゴリズムは、GPUの進化により高度化・複雑化している。一方で、複雑なアルゴリズムを機械学習の専門家でないソフトウェア技術が利用できるようにフレームワークが提供されつつある。また、このフレームワーク自身が、GPUの利用効率を高める仕組みを持っていたり、外部にCPUやGPUのスケジュールを決定する技術が提案されたりしている。

本章の残りでは、はじめに既存技術の概要について触れ、これらの技術やその組み合わせにより実現出来ていない課題について述べる。

2.1 物体検出アルゴリズムとフレームワーク

カメラ画像における物体検出の機械学習アルゴリズムとして、YOLO[2]やSSD-ResNet[3]などの多くのアルゴリズムが提案されており、実際に商用利用されているシステム

も多い。YOLOは入力フレーム画像に対し高速に推論処理を行うことができる点が特徴である。SSD-ResNetは、高速な識別を可能とするSSD[4]の内部に、ResNet[5]を適用して推論精度の向上している。いずれも、入力されるカメラ画像に対して、画像内にある物体のラベル、その物体を含む領域(矩形)、物体と学習モデルの一致度を結果として出力する。汎用的な物体検出のアルゴリズムでは、事前に複数の物体を学習した、学習済みの重みが公開されているため、利用目的と学習済みの推論対象とが一致していれば、比較的容易に物体検出を実行することができる。

物体検出の適用先ユースケースとしては、例えばスマートシティの実現のために道路の監視カメラに適用され、車種ごとの車両の台数カウント[1][6]、流量の算出[7][8]、運転挙動の検出などが考えられている。また検出した台数などの情報を元に、信号制御などの更なる制御を行う[9]こともできる。実際のシステムでは、ユースケースごとの検出対象・非機能要件に応じて、物体検出のモデルを選定・学習し、実装する必要がある。

機械学習アルゴリズムをGPUで利用するには、学習モデルとカメラ画像データをGPUに転送し、GPUに対して推論処理の開始等を指示し、推論処理終了後にデータをGPUから取り出す必要がある。これらのGPUとのやり取りをライブラリ化し、使いやすくしたものがAIフレームワークである。GPUの利用効率化を実現する機能は、ソフトウェア開発者からできるだけ隠蔽する必要があるため、AIフレームワークの利用が必要である。

GPUを用いたAIフレームワークとしては、TensorFlow Serving[10]、NVIDIA Triton Inference Server[11]、NVIDIA DeepStream[12]など、GoogleやGPUを提供するNVIDIAが開発したフレームワークがある。GPUは開発された時期や性能によって違いがあるが、これらのフレームワークはその差異を吸収している。また、学習モデルや画像データは、推論処理毎にGPUに転送するのではなく、GPUのメモリ上に保持して何度も利用したり、推論処理結果のデータを次の処理でそのまま利用したりするなどの効率化を図ることが可能である。したがって、フレームワークを利用するシステムでは、十分なGPUサーバがある場合には、高いスケーラビリティを得ることできる。

2.2 GPUの利用効率化

GPUは、計算資源とメモリ資源をもつ独立した計算機能を持っており、これらの制御はメインコンピュータのCPUから行う。そのため、推論処理を行う場合には、事前の学習で得られた学習モデルをGPUメモリに転送してから処理を行う。一般に学習モデルの大きさは、数十MBから数百MBであり、非常に大きい。そのため、複数の推論アルゴリズムを1台のGPUで処理する場合には、推論アルゴリズムと学習モデルを何度も転送しなくても良いような工

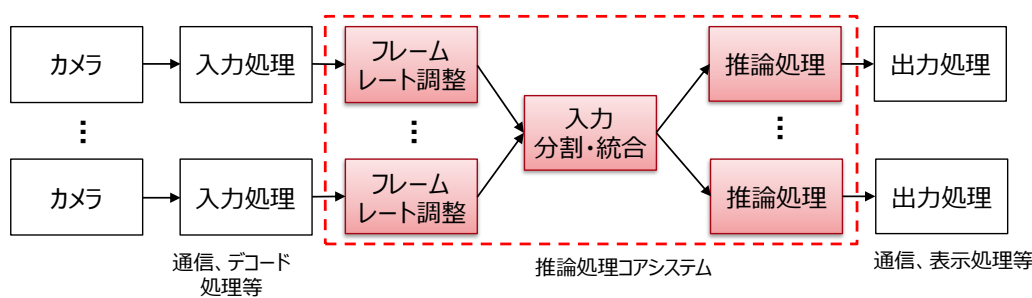


図2 システムの基本構成

夫が必要となる。

NVIDIA 製の GPU では、メモリ資源の制御については CUDA[13]、計算資源の制御については MPS (Multi Process Support) [14]が提供されている。これらの機能を利用することで、複数の推論処理を行う際に、計算資源の待ち時間やメモリ資源へのデータ転送を削減して、GPU の利用効率を高めることができる。実際には機械学習を利用するアプリケーションが、計算資源やメモリ資源の管理を意識してプログラムする必要がある。

しかし、複数の推論処理を同時に実行する場合、推論処理が使用する計算資源・メモリ資源の競合により推論スループットが低下することがある。アプリケーションで競合を意識しながらプログラミングすることは難しいので、この機能をスケジューラとして独立させて、アプリケーションとフレームワークを仲介する方式が提案されている。

筆者らは、GPU 内部での資源の競合・干渉が生じやすいタイミングを避けるように、推論処理実行タイミングを制御する方法[15]を既に提案している。GPU での推論処理が、GPU への学習モデル・画像データの転送、GPU での推論処理（ニューラルネットワークの計算）、GPU からの推論結果の転送の3つのフェーズからなる。この中で、複数の推論処理で、ニューラルネットワークの計算を同時に実行すると計算資源の競合が発生し、推論時間が長くなる。このため、アプリケーションとフレームワーク (Tensorflow) の間に資源制御機構において、ニューラルネットワークの計算が同時に起こらないように GPU への制御メッセージのタイミングを変えている。1GPU 上で SSD-ResNet50 を4並列実行する場合に、1.6 倍の推論スループット向上が得られたとしている。

これに対して、実際の利用シーンでは、検出した現象はまれにしか発生しないため、カメラから入力される全てのフレーム画像を推論するのではなく、前のフレーム画像との間に意味のある変化がある場合のみ、推論処理を行う方式[16]が提案されている。この論文では、固定された監視カメラが多いことから、通常何も検出されない画像を背景画像とし、背景画像との差分を検出して物体の有無を判定し、物体を含む領域の特定を行い、物体が何であるかを検出するという3つのフェーズに分けて分散処理する方式を提案

している。各フェーズの計算処理量は、前段に比較して後段の処理が圧倒的に大きい。そのため、前段で物体の有無を簡単に判定することで、GPU の負荷が高い推論処理の実行を抑制することができる。

3. 従来技術の課題

GPU を利用したカメラ画像の推論処理を効率化するには、GPU の資源管理を行うこと (GPU スケジューラ)、GPU で処理するフレーム画像数を削減する (フレームレート調整) ことの2つが必要になることがわかる (図2)。

3.1 GPU 資源管理

筆者ら[15]の方式では、GPU での推論処理を更に分解して、競合する処理のタイミングを制御することで、資源競合による GPU での処理速度低下を防いでいる。しかし、推論処理を行うフレーム画像の削減については、ユースケースに合わせて、入力するフレーム画像を定期的に間引くことで削減している。

一方、Zhang ら[16]では、個別の推論処理の効率化には触れず、処理負荷が最も高い物体検出の処理回数を抑制するために、物体の有無を前段で判断している。しかし、推論処理で利用する資源については、できるだけ処理の軽い方式を選択することにより解決している。

ここに挙げた2つの方式では、メインコンピュータの CPU と GPU の計算資源・メモリ資源を管理、制御するスケジューラが実現されている。筆者ら[15]の方式では、GPU は物体検出にのみ利用しており、この処理における計算資源とメモリ資源の管理と制御を行っている。一方、Zhang ら[16]の方式では、領域切り出しと物体検出の2つのフェーズで GPU を利用している。そのため、2つの処理のスケジューリングが必要である。彼らの提案手法は、3つの推論フェーズをパイプラインで処理するようにし、各フェーズがぶつからないように考慮されている。また、前段の処理は処理負荷が軽いことから、バッチ処理により複数回の処理をまとめて処理を行う。これにより1台の GPU で処理する際に、学習モデルを入れ替える回数が削減できるため、GPU での処理高速化が図ることができる。また、複数の

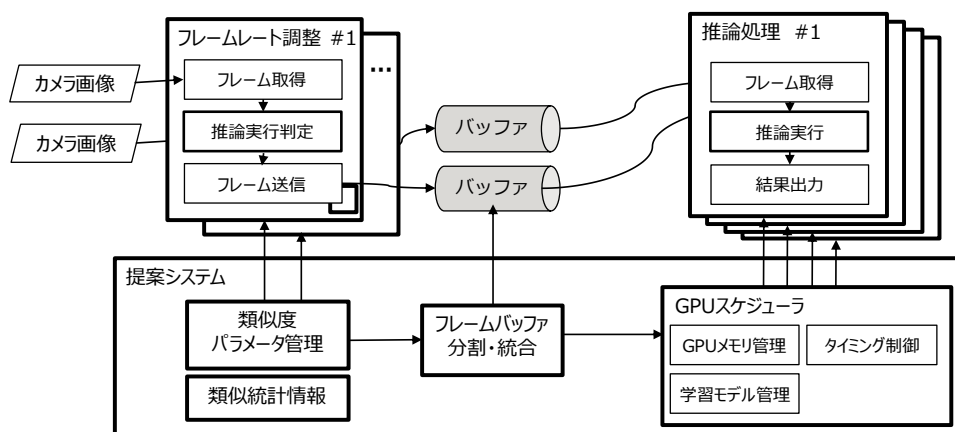


図3 提案システムの構成

GPU を利用する方式も検討しており、各フェーズを独立した GPU で実行することも可能である。しかし、Zhang らの方式は、各フェーズの処理内容の組み合わせが固定的になっており、ユースケースによって構成が決まってしまう。そのため、性能は高くなるが、汎用性が低いという課題があると考えられる。

本論文では、筆者らの方式を改良し、推論処理するフレーム画像の直前に推論処理をしたフレームとの類似性を識別することで、不必要な GPU 上の推論処理を削減する手法を提案する。この際に、類似性を判定する方式については、CPU で処理可能な方式を採用して、GPU 資源の管理をシステム全体に及ばないようにする。この方式では、推論処理に関しては複数のアルゴリズムで検証されているため、汎用性が高いシステムが実現可能である。

類似度判定で GPU を利用すると、推論処理と類似度判定での GPU リソースの競合が発生し、全体のスループットが向上しない。Zhang らの方式ではパイプライン化しているが、処理粒度が一定ではないのでパイプラインの設計難しい。

3.2 類似度判定

一般的なカメラでは、1 秒間に 30 フレーム、もしくは 60 フレームの映像を撮影できる。しかし、実際に推論が必要な場面でも 1/30 秒で状況が大きく変わることは多くない。図 1 で示した高速道路における自動車の動きでは、1 台の自動車がフレーム内に入ってから出るまで数秒から十秒かかっており、フレーム内に映っている自動車の台数を計測し渋滞を判断することが目的であれば 1 秒間に数フレームの映像があれば十分である。

3 章の最初に述べたように、推論処理を行う前に、推論処理を省略可能か判定する機能を入れることにより、計算負荷の高い推論処理を省略して全体の処理効率を高めることが可能である。推論処理が不要かどうかを判定する方法としては、(1)対象物の移動が少ないこと、(2)対象が映って

いる画像の特徴が似ていることなどが指標になる。

画像処理の分野では、動画画像からフレーム画像に映っている物体の移動量を測定したり、画像の分類により類似した画像を判定したりする方法が既に実用化されている。

画像処理の領域における物体追跡の領域では、物体の移動傾向[17]または検出対象の物体との類似性を算出して追跡する方法[18][19]が提案されている。物体追跡の過程では、推論結果が変化するかどうかの判定を直接は行えないが、物体の移動によるフレーム内の変化を定量化できることが期待される。一般的に画像処理における物体追跡では、前回のフレーム内での物体の位置を元に、追跡すべき対象がフレーム上のどこに移動したかを算出することで、物体の追跡を行う[20]。[17]で提案される手法では、物体が全く映っていない背景画像と、物体が映り込んだ画像との明暗の差分をピクセル単位で取った後に、差分領域の変化の方向を算出する。また[18][19]では、2 つの画像の類似性を定量的に算出・比較する画像の特徴抽出手法を用いている。この特徴を用いて、追跡したい物体のテンプレート画像と、フレーム内でテンプレート画像に最も類似した領域を算出し、フレーム内での位置を追跡することができる。この特徴抽出手法は、著作権侵害確認のために、ある画像と既存の著作物との類似性を判断する[21]ためにも用いられている。

本論文では、物体の移動量や特徴の類似度を判定するアルゴリズムを利用して、推論処理が不要なフレーム画像の判定を行う。今回の目的では、推論結果が同じになるフレーム画像の除去が不十分であっても、状況が変わる境目のフレーム画像を見落とさないように、類似度の判定を厳密に行わず、処理速度を優先することで、全体の処理効率を高める方式を提案する。

4. 提案システム

4.1 システム概要

提案システムの構成を図 3 に示す。提案システムでは、1 台以上のカメラが接続され、通信やデコード処理を行う入力処理を経て、推論処理コアシステムに接続される。また、推論処理コアシステムは入力されたフレーム画像の推論結果を出力する。推論処理コアシステムは、推論実行判定と推論処理部からなり、推論実行判定では、前のフレーム画像と比較を行い、実際に推論処理を行うかどうかの判定を行う。

物体検出アルゴリズムは様々であり、今後も新しいアルゴリズムが提案されるため、推論実行判定部と推論処理部は特定の物体検出モデルに依存しないようにする。これは、ユースケースや撮影環境条件により、推論処理部選択される推論アルゴリズムが、YOLO や SSD-ResNet などの推論モデルから選択可能にするためである。また、推論モデルによって、使用する GPU 資源量が異なるため、GPU リソースが許される範囲で、任意の物体検出モデルを並列に実行することを前提とする。高速道路の車両数を計算する入力カメラ画像の場合、例えば朝夕の通勤時間帯と、昼や深夜とで走行する車両台数が大きく変化する。またこの変化は入力カメラ画像ごとに異なる。このような時間変化を小さい計算量で識別可能にするため、フレームの前後関係を使用することに着目し、入力装置ごとに推論実行判定部を通る構成としている。推論実行判定部の詳細は、4.2 節で説明する。

本システムで考慮されるべき点は、以下の 3 点である。

- (1) 前のフレーム画像と同じ推論結果が得られる画像（類似画像）の簡易な判定方法
- (2) GPU での推論処理と競合しない類似画像判定方法
- (3) GPU での推論処理の上限に合わせたフレーム画像数の調整方法

(1)の推論結果が同じになる類似画像の判定方法については、画像処理の分野でいくつかの方式が検討されているので、それを利用する。(2)の推論処理と競合しない判定方式については、できるだけ CPU で処理することがシステムを設計する上で重要である。Tensorflow 等のフレームワーク向けに用意されているフレーム画像のデコードや推論の前処理等のライブラリは、積極的に GPU を使って処理効率を高める工夫をしている。しかし、フレームワーク上に作られる部分とフレームワークの外で作られる機能を組み合わせるときには、予定通りの動作をしないことが多く、タイミングによって処理速度が低下することがある。(3)については、推論処理の全体効率を上げるには GPU を十分活用することが大事である。最も簡単な方法は、ユースケースに合わせてフレーム画像を定期的に間引くことである。

しかし、類似画像判定をすることにより、実際に映って

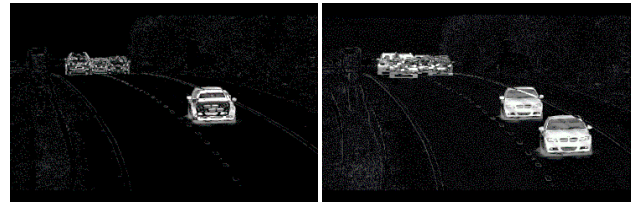


図 4 フレーム間の差分ピクセル領域

左図：フレーム間の差分領域が小さく、推論を行わないケース、

右図：差分領域が大きく、推論を実行するケース

いる物体の出現頻度により推論が必要となる頻度は動的に変動する。高速道路を通行する車両数から渋滞度合いを検出するシステムを想定すると、日常的に発生する朝夕のピークがあり、逆にほとんど通行する車両のない時間帯も存在する。ピークに合わせて GPU へ入力するフレーム画像数を決めてしまうと、ピーク以外の時間帯において GPU が空いてしまい、利用効率が下がってしまう。一方で、全体の平均に合わせてしまうと、平均を超えた部分については GPU で処理を待つバッファでの待ち時間が長くなり、処理の遅延時間として認識されるようになる。GPU への入力頻度をどの程度に設定するかはユースケースに依存する部分が多い。詳細については 4.3 節で説明する。

4.2 推論実行判定部

本機能は、GPU 上の推論処理よりも小さい計算量で、推論結果に変化がないと予測されるフレームを識別する。従来は、このような変化を検出するために、単体のフレームで判定を行うことが可能な機械学習の推論モデルを用いていた。しかしながら、計算量が大きく、実装とモデル構築のコストが大きいという問題がある。そこで本稿では、前後フレーム画像の変化の度合いをフレーム間の類似度として、軽量で GPU を利用しない類似度判定方式での解決を検討した。

類似度判定方式については、具体的に 2 つの方式を検討し、実際に評価を行った。1 つは、フレーム画像内の物体の移動量をフレーム間の差分を計測する方法、もう 1 つは、画像の特徴量を Perceptual Hash[22]により計算し、ハミング距離を求めて類似度を判定する方法である。以下では、これらの詳細について述べる。

4.2.1 フレーム間差分量による類似フレーム判定

単純な方法として、2 つのフレーム画像の比較を行い、異なる領域のピクセル数で判定する方式(フレーム間差分)を検討した。これはカメラが固定され、パン・チルト・ズーム (TPZ) を行わずに常に同じ範囲を撮影するとき使用できる。本稿では、計算量を小さくするために、比較する 2 枚のフレーム画像 X_0 , X_1 をグレースケール画像に変換し、 i 番目のピクセル X_{0i} , X_{1i} の輝度の差を求める。この差分値が予め決めた閾値 T_1 内であれば、類似ピクセル

と判断し、このピクセル数を数えて類似度 $D_{X0,X1}$ とする。

$$0 \leq T1 \leq \text{all_pixel}$$

$$d_{T1,i} = \begin{cases} 1 & (T1 < \text{abs}(X0_i, X1_i)) \\ 0 & (T1 \geq \text{abs}(X0_i, X1_i)) \end{cases}$$

$$D_{X0,X1} = \sum_i^{\text{all_pixel}} d_{T1,i}$$

推論実行判定部では、前々回推論したフレームを $X0$ 、前回推論したフレームを $X1$ 、新たに到着したフレームを $X1'$ として、フレーム $X0$ と $X1$ 、フレーム $X1$ と $X1'$ それぞれの類似度を計算する。この類似度が閾値 $T2$ を超えていなければ、この2枚のフレーム画像は類似であり、推論処理は不要 ($F(D_{X0,X1}, D_{X1,X1'})=0$) と判断し、次のフレーム画像と基準フレーム画像の類似度を測定する。もし閾値 $T2$ を超えていた場合には、推論処理が必要な画像として推論処理部に転送すると同時に、このフレーム画像を新しい基準フレーム画像として次のフレーム画像との比較を行う。

$$0 \leq T2 \leq \text{all_pixel}$$

$$F(D_{X0,X1}, D_{X1,X1'}) = \begin{cases} 1 & (T2 \leq |D_{X0,X1} - D_{X1,X1'}|) \\ 0 & (T2 > |D_{X0,X1} - D_{X1,X1'}|) \end{cases}$$

例えば、図1の高速道路の画像では、車両の台数が増えずに手前方向に移動するだけであれば類似画像であり、推論結果となる車両台数は変化しないと判定して推論処理を行わないようにする。一方、奥の方から新しい車両が現れたり、手前の車両がカメラの視界から消えたりした場合には基準フレームとの間に大きな差が生じ、類似画像とは判定されないで推論処理を実行する (図4)。

単純な方式であるため、安定した精度を確保することは難しい。特に、物体が画面の中で交差・重複したり、移動により物体の大きさが変わったりする場合に、一定の閾値によるフレーム内の変化の検出が難しい。また、カメラ自体の振動や、樹木や影などの背景の振動、降雨や天候による明るさの変化などの影響によって、差分として検出される領域の誤差が大きくなってしまう。このように、ユースケースによってはノイズの影響を受けやすいという欠点がある。しかし、計算量が非常に小さく、短時間で計算することが可能であるため、多くのフレーム画像をフィルタするには有利な方法である。

4.2.2 ハッシュ値による類似フレーム判定

フレーム画像の特徴を画像処理により数値化し、この数値によって類似度を判定する方式である。この1つの方法

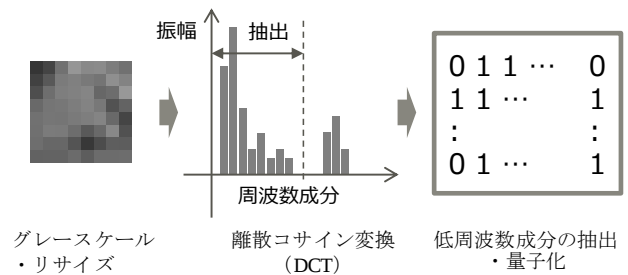


図5 pHash 算出の流れ

として、今回は Perceptual Hash (pHash) [22]を使用する。pHash は画像等のマルチメディアデータ用のハッシュ値をまとめるアルゴリズムであり、似ている2つの画像のハッシュ値は近い値になることが特徴である。

pHash によるハッシュ値算出の流れを、図5に示す。pHash は、画像を離散コサイン変換により輝度の変化を周波数成分に変換する。次に、変換された周波数成分のうち、一定数の低周波数成分のベクトルを得る。その後、周波数成分の平均値を元に、量子化したものをハッシュ値として用いる。この手法では、低周波数成分のみを使用することで、細かいノイズによる変化の影響を小さくし、画像のおおまかな特徴を取り出すことができる。フレーム画像間の類似度は、量子化された64ビットの文字列の各桁同士を比較し、値が異なる桁の和(ハミング距離)を用いて判定する。ハミング距離が大きいほどフレーム画像の差が大きい(類似度が低い)として判断することができる。

pHash を用いて類似度 $DP_{X0,X1}$ を求めるには、まず2つのフレーム $X0$, $X1$ からそれぞれ得られるハッシュ値 $h0$, $h1$ のハミング距離 $HD_{h0,h1}$ を求める。ビット長 B のハッシュ値の各ビット b 同士を比較して排他的論理和を計算し、ビットが1となる位の数の和を計算する。このハミング距離 $HD_{h0,h1}$ 閾値 $T3$ を超えていなければ、この2枚のフレーム画像は類似であり、推論処理は不要と判断する。

$$HD_{h0,h1} = \sum_b^B \text{XOR}(h0_b, h1_b)$$

$$DP_{X0,X1} = \begin{cases} 1 & (T3 < HD_{h0,h1}) \\ 0 & (T3 \geq HD_{h0,h1}) \end{cases}$$

$$0 \leq T3 \leq B$$

4.3 推論処理への入力データ量の調整

物体検出で使われる YOLO や SSD-Resnet の処理時間は、入力画像に関わらずほぼ一定になる。そのため、この処理時間に合わせて入力データを GPU に渡すことで、GPU は最も効率良く利用できる。一方で、カメラ画像は 30fps や 60fps と高い頻度でフレーム画像が発生するため、GPU に

入力するフレーム画像の頻度を調整する機能が必要である。多くのシステムでは、カメラ画像を一定間隔で間引きしてフレームレートを下げて利用したり、フレーム画像を分割して複数の GPU で並列処理を行うことが多い。

本論文では、推論実行判定機能により、GPU での推論処理の実行が必要と判断されるフレーム画像は、ダイナミックに変動することが予想される。そのために、単純なフレーム調整は難しい。ますます GPU での推論処理の頻度と一致させることが難しくなる。本提案では、推論実行判定後のフレーム画像の平均値を求めて、この平均値により調整を行う。1 つのカメラ画像（※語彙揺れ）のフレーム中で検出すべき物体の数を表すために、Target-Object-Ratio (TOR) を用いる[16]。num_{all_frames} は一定長の動画に含まれるすべてのフレーム数、num_{target_object_frames} は推論実行により検出されるべきオブジェクトを含むフレーム数を表す。

$$TOR = \frac{num_{target_object_frames}}{num_{all_frames}}$$

これに対し、GPU の推論スループット上限を $P_{max_throughput}$ 、カメラ画像 j の TOR_j を用いると、評価時のカメラ画像の論理的な並列度 M は、以下の式を満たす最大値で表される。

$$P_{max_throughput} \geq \sum_j^M TOR_j$$

評価結果は次章で述べるが、この平均値は GPU での処理能力を下回るため、複数のカメラ画像を 1 台の GPU に入力して並列実行することになる。理論的には、類似判定後のフレーム画像の平均値と GPU の処理性能を一致させることで良いが、入力されるフレーム画像の頻度が平均を超えるときには、過剰分の画像がバッファに蓄積されるため、推論結果が得られるまでに遅延が発生する。そのため、この遅延をどこまで許容できるかを決めた上で、GPU の処理性能に余裕を持たせた設定が必要となる。

5. 評価

提案システムにおける推論実行判定機能と推論処理の入力画像処理機能について評価を行った。推論機能は[15]で開発した機能を前提にしている。

表 1 に評価環境の詳細を示す。今回は GPU サーバとして、AWS の GPU インスタンスを使用した。CUDA や cuDNN などの GPU の利用に必要なライブラリがインストールされていれば、TensorFlow で記述されたプログラムには大きな変更を加えることなく GPU で推論処理を実行することができる。また画像処理のライブラリとしては OpenCV を

表 1 評価環境

インスタンスタイプ	g4dn.xlarge
CPU	Intel Xeon 8259CL@2.50GHz
vCPU	4
CPUメモリ	16GB
GPU	NVIDIA Tesla T4
GPUメモリ	16GB
OSバージョン	Ubuntu 18.04
Pythonバージョン	3.7
TensorFlowバージョン	2.3
NVIDIA Driverバージョン	450.51.05
CUDA Toolkitバージョン	10.0.130
MPS	有効

表 2 類似フレーム削減結果

フレーム間類似度判定方式	類似フレームと判定された割合[%]	推論した画像のフレームレート[fps]
単純間引き(4fps)	84	4.0
フレーム間差分による判定	88	2.9
pHashによる判定	95	1.3

使用した。GPU 上の推論負荷とのバランスを取るため、OpenCV は CUDA を使用せず、全て CPU 上で動作する。

入力カメラ画像については、各スレッドの前処理部で同一の内容のカメラ画像を用いることで、スレッドごとに平均推論実行回数が大きく変化しないようにしている。また、推論実行タイミングが極端に偏らないよう、読み出し時刻をずらして計測した。

評価で利用したのは、図 1 に示した高速道路において車両が奥から手前方向に流れる動画である。25fps の 60 秒間の動画を利用して評価を行っている。60 秒間、絶えず車両が走行しているが、途中、車両数が少なくなる時間がある。これによって、類似度判定の結果、推論処理が必要と判定される映像の頻度は一定でなく、変動している。このときのシステムの動作について考察を行った。

5.1 推論実行判定機能の効果

本評価では、まず入力カメラ画像に対する推論実行判定の効果として、類似フレームによる画像の削減率を計測した。フレーム画像の類似度は、4 章で述べた 2 つの方式である。

表 2 に、高速道路を撮影した入力カメラ画像において、類似するフレームとして判定された割合を示す。比較のために、単純に間引きするケースを含めた。フレーム間差分、pHash の両方の方式で、単純に間引く場合より推論処理が削減されていることが分かる。各フレームを確認したところ、カメラの視界内に車両が現れたり、視界から消えたタイミングの画像が削除されたりしていないことが確認できた。したがって、同じ推論結果が得られる推論処理を事前

に検出し、推論処理すべき画像を絞り込めたといえる。単純に間引きをしたケースでは、変化するタイミングではなく変化後の最初のフレームで検出するため、同様の結果は得られない。

類似判定の処理時間については、フレーム間差分で 0.4 ミリ秒、pHash で 3.1 ミリ秒であった。フレーム間差分では処理時間は短くなっているが、必要な画像以外を絞り込むという点では性能が低いといえる。フレーム間差分の精度が低い理由としては、物体の差分をピクセル数で表現しているため、画像上、遠くの小さな物体と近くの大きな物体が同じピクセル数で表現されてしまうため、近くの物体の影響が大きくなることが挙げられる。個別の物体が占めるピクセル数で割って移動した割合として表現できれば精度が上がるが計算量が多くなる欠点がある。一方、pHash では、離散コサイン変換により、画像全体の特徴を周波数成分で数値化するため、物体の遠近に大きさの影響は少ないと考えられる。また、フレーム間差分よりもノイズの影響を受けにくいのが特長である。

フレーム間差分、pHash の 2 つの方式について、2 つのフレーム画像が類似であると判定する閾値の設定がとても難しい。この評価では、閾値を変えて何度か測定を行い、変化点となる全てのフレーム画像が含まれる最小値を評価結果として示した。閾値を大きくして類似度の判定を緩くすると、推論実行判定における絞り込みが不十分となり、推論処理の実行回数が増え、全体の処理量が増えてしまう。逆に閾値を小さくすると、推論実行判定における絞り込みが過剰に行われ、変化点となるフレーム画像の位置が正確でなくなってしまう。変化点となるフレーム画像を取りこぼさないように適切な閾値を決定することは非常に難しいことがわかる。

また、遠景の物体小さく映るケースや物体同士が接触したり重なったりしているケース、路面や背景と車両の色が似ているケースも精度が悪くなる要因である。今回は、ともに元の画像をグレースケールに変換し、輝度をベースに判定を行っている。元の画像に含まれるカラー情報等を利用することで重なる物体の分離精度を改善することができる。しかし、その分処理量が増えるため、処理量と精度をいかにバランスさせるかが課題である。

5.2 推論入力画像の分離・統合

5.1 節では、推論実行判定機能において、実際に推論処理を行う画像の絞り込みが可能であることを示した。今回前提とする推論処理部では、SSD-ResNet50 を 16fps で推論処理可能である。絞り込みを行ったフレーム画像は大幅にこの数字を下回ることから、複数のカメラ画像を同時に入力して接続することになる。ここでは、複数のフレーム画像を統合して、1 台の GPU に入力するケースを評価した。

表 3 に今回の評価における並列度を示す。pHash のケー

表 3 並列処理結果

フレーム間類似度判定方式	並列度	平均推論回数[fps]	推論実行判定フレーム数[fps]
フレーム間差分による判定	5	14.5	125
pHashによる判定	8	10.4	200

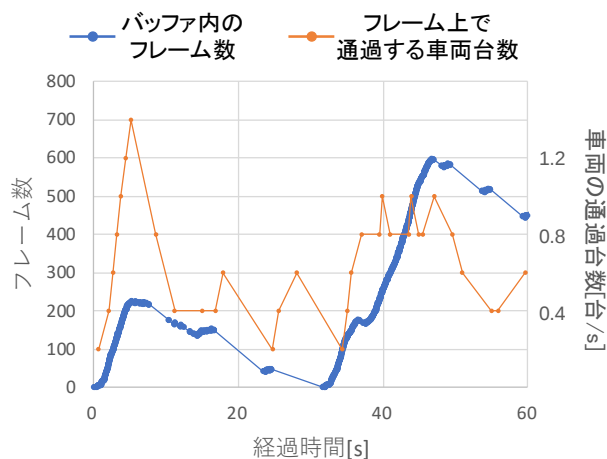
スでは、平均推論回数が 10.4fps であり、GPU での性能限界である 16fps には余裕があり、計算上は 12 並列まで可能なはずである。しかし、今回使用したサーバの CPU では、8 並列が処理性能での上限であった。そのため、評価は 8 並列で行っている。

実際に 60 秒間の映像を処理したときの、フレーム上で通過する車両台数と推論処理を待つフレーム画像数を測定した。ここで通過した車両とは、奥から手前方向に走行している車両が、画像の下端から消えたことを指す。図 6(a)はフレーム間差分による結果、図 6(b)は pHash による結果である。平均して毎秒 0.7 台前後の車両が流れていることが分かる。この中で、最初の 5 秒間と後半 40 秒前後から道路上の車両数が小さなピークを描いていることがわかる。

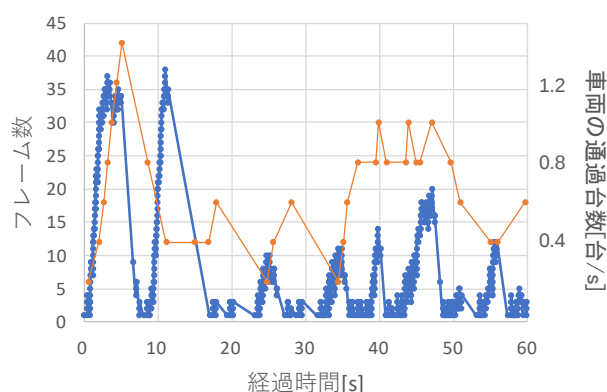
推論処理へのバッファについては、車両の通過台数が増えると、推論処理待ちのフレーム画像が増加傾向になることがわかる。車両の通過台数が 0.5 台/秒を越えるあたりから増加に転じ、それ以下になると減少する。また、後半では待ちフレーム数が 600 に達しており、この時点でのデータが全て推論処理されるまで 38 秒程度かかることになる。したがって、撮影された画像が推論完了までの遅延時間に制限がある場合には、このフレーム数に収まるように並列度を調整する必要がある。一方、pHash においてもフレーム数の増加が見られるが、ピークで 40 フレーム程度である。これは GPU の性能 12fps に対して、8fps に絞り込まれているためであり、GPU の処理リソースに余裕があるためと考えられる。また、フレーム数が増加するのは車両の通過台数が 0.7~0.8 台に達するときである。

5.3 考察

提案システムは、OpenCV や Tensorflow を利用し、汎用性を考慮したシステムになっている。特に、推論実行判定部と推論処理部は疎結合としており、比較的自由的に組み合わせで利用可能である。表 4 では、提案システムと FFS-VA と比較を行った。FFS-VA では、推論処理を実行する前に、3 段階で異なるアルゴリズムによりフレーム画像をかなり絞り込んでいる。最後の推論処理では、必要なイベントが含まれるフレーム画像のみが選択され、処理回数が抑えられる。そのため、強力な推論アルゴリズムを使用して、絞り込まれたフレームに対して正確な推論を行うことができる。途中の絞り込み段階での処理では、入力される画像サイズがそれぞれ最適化され、高速に処理が行われている。その結果、全体のスループットでは、提案方式より 4.5 倍



(a) フレーム間差分



(b) pHash

図6 評価中のフレームキュー内のフレーム数遷移

高速に実行されている。

提案方式では、推論実行判定、推論処理の各段階では画像サイズ等の最適化を行っておらず、全体のスループットに改善の余地が残されている。一方で、機能モジュールを接続する段階で、個別のチューニングを行っていないため、機能モジュール構成が固定的にならず、汎用性が維持されていることが利点である。

6. おわりに

本論文では、カメラ画像から物体検出するシステムにおいて、推論結果を簡単な方法で予測することで、GPUによる推論回数を削減して推論実行を効率化する方法について述べた。

カメラは様々な場所に設置され、道路における渋滞や事故などの状況を人間に代わって判断し、通知するような場面で広く使われている。しかし、GPUは高価なためにその利用が限られている。GPUの利用効率をあげることで相対的に利用コストを削減することが望まれる。我々は、注目されるイベントが頻繁に発生しないことに着目し、イベン

表4 従来手法との比較

	提案手法	Zhangらの方式[16]
GPU数(最小)	GPU x1	GPU x2
フレーム削減	pHash/フレーム差分 CPUのみで処理	SDD+SNM+T-YOLO CPU+GPUで処理
推論アルゴリズム	SSD-ResNet50	Reference-NN (YOLOv2, YOLOv3)
スケジューリング	CPU/GPU独立	パイプライン
入力画像(カメラ)	640x360	600x400
類似フレーム判定	640x360	100x100
推論処理	640x360	412x412
TOR	0.05(pHash)	0.129
全体スループット	200fps(GPUx1)	930fps(GPUx2)
汎用性	推論アルゴリズム非依存 ユースケース毎にパラ メータを決める必要	推論アルゴリズム依存 組み合わせによりGPU処 理スケジューリング必要

トが含まれる可能性の高い画像を事前に絞り込み、推論処理の回数を抑制する方式を提案した。

提案システムでは、前のフレーム画像との類似性を計算し、同じ推論結果が得られる（新たなイベントが発生しない）ことを判定する。画像の類似性を Perceptual Hash(pHash) を利用して判定することで、平均で 95% のフレーム画像の推論処理を抑制できることを確認した。pHash は CPU のみで計算処理を行い、推論処理で利用する GPU との資源協業を回避した。この結果、CPU・GPU の負荷を考慮して並列度を高めることで、全体で 200fps、従来方式に比較して 12.5 倍のカメラ画像を収容可能であることを示した。

さらに GPU の利用効率を高めるために、入力カメラ画像の内容の変化に応じて、自動で CPU・GPU の負荷のバランスを取ることと、時間変動する推論対象のフレーム数の変化に対し、待ち時間や識別精度が許される範囲で、閾値の設定等により入力量を動的に調整することが今後の課題として得られた。

参考文献

- [1] Z. Zhang, K. Liu, F. Gao, X. Li and G. Wang, "Vision-based vehicle detecting and counting for traffic flow analysis," 2016 International Joint Conference on Neural Networks (IJCNN), 2016, pp. 2267-2273.
- [2] Redmon, Joseph and Farhadi, Ali YOLOv3: An Incremental Improvement. (2018), cite arxiv:1804.02767Comment: Tech Report .
- [3] X. Lu, X. Kang, S. Nishide and F. Ren, "Object detection based on SSD-ResNet," 2019 IEEE 6th International Conference on Cloud Computing and Intelligence Systems (CCIS), 2019, pp. 89-92.
- [4] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Fu, and A. Berg, "SSD: Single Shot MultiBox Detector," In Computer Vision – ECCV 2016, Springer International Publishing, ECCV 2016, pp. 21–37.
- [5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun.. Deep Residual Learning for Image Recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770-778.
- [6] N. O. Alsrehin, A. F. Klaib and A. Magableh, "Intelligent Transportation and Control Systems Using Data Mining and Machine Learning Techniques: A Comprehensive Study," in IEEE Access, vol. 7, pp. 49830-49857, 2019.

- [7] L. Li, L. Chen, X. Huang and J. Huang, "A Traffic Congestion Estimation Approach from Video Using Time-Spatial Imagery," 2008 First International Conference on Intelligent Networks and Intelligent Systems, 2008, pp. 465-469.
- [8] Ganesh Ananthanarayanan, Victor Bahl, Landon Cox, Alex Crown, Shadi Noghahi, and Yuanchao Shu. 2019. Video Analytics - Killer App for Edge Computing. In Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '19). Association for Computing Machinery, New York, NY, USA, 695–696.
- [9] N. V. Hung, L. C. Tran, N. H. Dung, T. M. Hoang and N. T. Dzong, "A traffic monitoring system for a mixed traffic flow via road estimation and analysis," 2016 IEEE Sixth International Conference on Communications and Electronics (ICCE), 2016, pp. 375-378.
- [10] TensorFlow Serving,
<https://www.tensorflow.org/tfx/guide/serving>, online, Last accessed 10 May 2021
- [11] NVIDIA Triton Inference Server,
<https://developer.nvidia.com/nvidia-triton-inference-server>, online, Last accessed 10 May 2021
- [12] NVIDIA DeepStream, <https://developer.nvidia.com/deepstream-sdk>, online, Last accessed 10 May 2021
- [13] "CUDA Toolkit Documentation 2.1 Kernels"..
<https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#kernels> online, Last accessed 10 May 2021.
- [14] "Multi Process Service",
https://docs.nvidia.com/deploy/pdf/CUDA_Multi_Process_Service_Overview.pdf, online, Last accessed 10 May 2021.
- [15] 鈴木貴久, 田中美帆, 豊永慎也, 松倉隆一. 推論多重実行における GPU 資源利用効率化技術. 研究報告マルチメディア通信と分散処理 (DPS), No. 65, Mar. 2021.
- [16] C. Zhang, Q. Cao, H. Jiang, W. Zhang, J. Li and J. Yao, "A Fast Filtering Mechanism to Improve Efficiency of Large-Scale Video Analytics," in IEEE Transactions on Computers, vol. 69, no. 6, pp. 914-928, 1 June 2020.
- [17] A. J. Lipton, H. Fujiyoshi and R. S. Patil, "Moving target classification and tracking from real-time video," Proceedings Fourth IEEE Workshop on Applications of Computer Vision. WACV'98 (Cat. No.98EX201), Princeton, NJ, USA, 1998, pp. 8-14.
- [18] S. S. Sengar and S. Mukhopadhyay, "Moving object tracking using Laplacian-DCT based perceptual hash," 2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET), Chennai, India, 2016, pp. 2345-2349.
- [19] Fengjiao Fan, Guoming Gao and Jianping Li, "Visual object tracking based on perceptual hash algorithm," 2015 12th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP), Chengdu, China, 2015, pp. 233-236
- [20] S. R. Balaji and S. Karthikeyan, "A survey on moving object tracking using image processing," 2017 11th International Conference on Intelligent Systems and Control (ISCO), Coimbatore, India, 2017, pp. 469-474.
- [21] X. Wang, K. Pang, X. Zhou, Y. Zhou, L. Li and J. Xue, "A Visual Model-Based Perceptual Image Hash for Content Authentication," in IEEE Transactions on Information Forensics and Security, vol. 10, no. 7, pp. 1336-1349, July 2015.
- [22] Kind of Like That,
<http://www.hackerfactor.com/blog/?/archives/529-Kind-of-Like-That.html>, online, Last accessed 10 May, 2021.