二値分解を用いた物体検出モデルの近似計算による軽量化

○井上 涼太 †，大杉 佳史 †，平川 翼 †，山下 隆義 †，藤吉 弘亘 †

†：中部大学

inoue@mprg.cs.chubu.ac.jp

概要：DEtection TRansformer (DETR) は高精度な物体検出が可能である一方，ネットワークが大規模になる傾向があるため，知識蒸留や量子化などを適用した手法が提案されている．しかし，これらの手法は再学習を必要とし，既存の学習済みモデルの重みに対して蒸留や量子化を直接適用できない．そこで本研究では，学習済み DETR の重みと特徴量をベクトル分解法により二値化し，近似内積計算を適用したモデル圧縮と推論の高速化を再学習なしで行う Binary-decomposed DETR を提案する．評価実験より，精度低下を抑制しつつメモリサイズを削減できることを確認した．

＜キーワード＞物体検出，モデル圧縮，DETR

1. はじめに

物体検出は，物体の位置とクラスを推定するタスクである．自動車に物体検出を導入することで車両や歩行者などを認識して事故防止を促したり，工場の生産ラインに物体検出を導入することで外観検査を自動化することが可能になるなど様々な分野で活用されており，注目が集まっている．このような場面で利用されるのはエッジデバイスと呼ばれる，特定の処理に特化した小型のコンピュータである．代表的な物体検出手法として DEtection TRansformer (DETR) [1] がある．DETR は Transformer [2] を初めて採用した物体検出モデルであり，画像の特徴量を抽出する Convolutional Neural Network (CNN) に Transformer を組み合わせることで，物体検出を直接的な集合予測問題として捉えることが可能になり，End-to-End なモデル設計を実現している．DETR のモデル構造を図 1 に示す．DETR のように高性能な物体検出モデルは，ネットワークが大規模になる傾向があり，DETR-R101 モデルにおいては 60M ものパラメータで構成されている．そのため，計算リソースが限られているエッジデバイス等に DETR を導入するには，DETR のモデルサイズの圧縮と推論の高速化が不可欠である．

従来の研究では，DETR の効率を向上させるため，知識蒸留や量子化，学習の効率化などを DETR

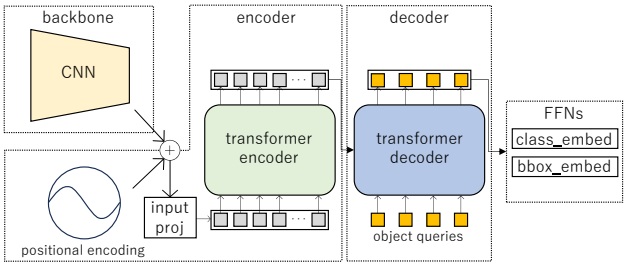

図 1 DETR のモデル構造

に適用した手法が提案されている [3, 4, 5, 6]．DETRDistill [5] は，物体検出を直接的な集合予測問題とする DETR に特化した知識蒸留フレームワークを提案している．Quantized (Q)-DETR [6] は量子化時のクエリ情報の歪みによる精度低下を Distribution Rectification Distillation (DRD) により抑制する量子化手法を提案している．しかし，これらの手法は再学習を必要とし，既存の学習済みモデルの重みに対して蒸留や量子化を直接適用することは困難である．

そこで本研究では，DETR の学習済みの重みと特徴量をベクトル分解法により二値化し，近似内積計算を適用したモデル圧縮と推論の高速化を再学習なしで行う Binary-decomposed DETR (BdDETR) を提案する．具体的には，DETR の畳み込み層と全結合層における各層の重みと特徴量を二値化する．そして，畳み込み層と全結合層の積和計算を二値同士の近似内積計算に置き換える．これにより，既存の学習済み DETR モデルにも容易に適用

でき，リソース制約のあるエッジデバイス上での高効率な物体検出が可能となる．

まとめると，本研究の貢献は以下の通りである：

- 本研究では，ベクトル分解法と近似内積計算を用いてモデル圧縮と推論の高速化を再学習なしで行う BdDETR を提案する．
- 評価実験より，BdDETR は精度低下を抑制しつつモデル圧縮と推論の高速化が可能であることを示す．

2. 関連研究

本章では，DETR に対する知識蒸留と量子化によるモデル圧縮について紹介する．

2.1. 知識蒸留によるモデル圧縮

知識蒸留はモデル圧縮技術の一つであり，高性能な教師モデルから軽量で計算効率の高い生徒モデルへ知識を転移することで，メモリ使用量の削減や計算効率の向上を図る手法である．具体的には，教師モデルの出力や中間層の特徴量を，損失関数を通じて生徒モデルに学習させることで，教師モデルと近い性能を維持したままモデルサイズを小さくする．DETR に対して知識蒸留を適用した手法として DETRDistill [5] がある．DETRDistill は，物体検出を直接的な集合予測問題とする DETR に特化した知識蒸留フレームワークであり，3 つの蒸留で構成される．一つ目は，Hungarian-matching logits 蒸留により，教師モデルと生徒モデルの間で予測を一対一で最適に対応づける．二つ目は，Target-aware feature 蒸留により，教師モデルが持つオブジェクトに関する特徴量を生徒モデルに学習させる．三つめは，Query-prior Assignment 蒸留により，教師モデルの学習済みクエリを生徒モデルに適用し，安定した予測を促す．以上により，DETRDistill は，教師モデルと比較して精度の低下を最小限に抑えつつ，生徒モデルに効率的なモデル圧縮と推論速度の向上を実現している．

2.2. 量子化によるモデル圧縮

量子化はモデル圧縮技術の一つであり，ニューラルネットワークのパラメータなどを量子化することで，メモリ使用量の削減や計算効率の向上を図る手法である．具体的には，32 ビット浮動小数点のような高ビットで表現されているパラメータを 8 ビット整数のような低ビットへ変換する．DETR に対して量子化を適用した手法として Q-DETR [6] がある．Q-DETR は DETR を 2〜4 ビットのような低ビットで量子化することにより，計算コストとメモリ消費を大幅に削減している．また，マルチヘッドアテンションモジュールなどは量子化による情報の歪みが発生しやすく，精度低下につながることから，Distribution Rectification Distillation (DRD) を導入している．DRD では，自己情報エントロピーの最大化と条件付き情報エントロピーの最小化の 2 段階最適化により，量子化されたモデルの性能劣化を抑制している．

まとめると，DETR のモデル圧縮の研究は知識蒸留や量子化など様々な手法が提案されている．しかし，これらの手法は再学習を必要とし，既存の学習済みモデルの重みに対して蒸留や量子化を直接適用することは困難である．

3. 提案手法

本章では，DETR に対してベクトル分解法を適用したモデルである BdDETR について述べる．

3.1. ベクトル分解法による重みの二値化

二値同士の内積計算を行うためには実数パラメータを二値に変換する必要があり，二値に変換する方法としてベクトル分解がある．ベクトル分解は，重みベクトル $\mathbf{w} \in \mathbb{R}^D$ を二値基底行列 $\mathbf{M} \in \{-1, 1\}^{D\times k}$ とスケール係数ベクトル $\mathbf{c} \in \mathbb{R}^k$ に分解する．ここで，D は入力次元数，k は基底数である．重みベクトルを分解することで，入力データが二値の場合において実数同士の内積計算を二値同士の内積に置き換えることができる．ベクトル分解を最適化するために，Exhaustive 法 [7] を使用する．Exhaustive 法によるベクトル分解処理を Algorithm1 に示す．Exhaustive 法による分解は，コスト関数 E が最小化するような二値基底行列 $\mathbf{M}$ とスケール係数ベクトル $\mathbf{c}$ を求める．コスト関数を式 (1) に示す．

$$E = \| \mathbf{w} - \mathbf{Mc} \|_2^2 \tag{1}$$

Exhaustive 法による分解は，重みベクトル $\mathbf{w}$ を表現する $\mathbf{M}$ とスケール係数ベクトル $\mathbf{c}$ を全探索により最適化する．そのため，分解に時間を要するが

Algorithm 1 Exhaustive 法によるベクトル分解

Require: $\mathbf{w}$, k, L
  for i to L **do**
    $\mathbf{M}_i$ を $\{-1, 1\}$ の乱数により初期化
    repeat
      $\mathbf{c}_i = (\mathbf{M}_i^{\mathrm{T}}\mathbf{M}_i)^{-1}(\mathbf{M}_i^{\mathrm{T}}\mathbf{w})$
      $\mathbf{M}_i = \mathrm{argmin}_{\mathbf{M}_i \in \{-1,1\}^{D\times k}} ||\mathbf{w} - \mathbf{M}_i\mathbf{c}_i||_2^2$
    until $||\mathbf{w} - \mathbf{M}\mathbf{c}||_2^2$ が収束
    $\hat{\mathbf{M}}, \hat{\mathbf{c}} = \mathrm{argmin}_{\mathbf{M},\mathbf{c}} ||\mathbf{w} - \mathbf{M}\mathbf{c}||_2^2$
  end for
  $\hat{\mathbf{M}}, \hat{\mathbf{c}}$

近似精度の良い分解が可能である．Algorithm1 より，重みベクトルの二値化はまず，$\mathbf{M}$ を $\{-1, 1\}$，$\mathbf{c}$ を実数の乱数により初期化する．次に，$\mathbf{M}$ と $\mathbf{c}$ を最適化する．このとき，$\mathbf{M}$ と $\mathbf{c}$ を同時に最適化することは困難であるため，$\mathbf{M}$ と $\mathbf{c}$ を交互に最適化する．具体的にはまず，$\mathbf{M}$ を固定して式 (1) を最小化する $\mathbf{c}$ を求める．次に，$\mathbf{c}$ を固定して式 (1) を最小化する $\mathbf{M}$ を求める．この最適化を式 (1) の値が収束するまで繰り返す．Exhaustive 法によるベクトル分解の近似精度は初期値に依存するため，初期値を L 回変更する．そして，L 回変更した結果から式 (1) が最も小さい $\mathbf{M}$ と $\mathbf{c}$ を最終的な分解結果とする．

3.1.1. 畳み込み層へのベクトル分解適用

畳み込み層のベクトル分解の流れを図 2 に示す．畳み込み層へのベクトル分解は，第 l 層目の n 番目の特徴量に対応する m 番目の重みフィルタ $\mathbf{w}_{n,m}^l \in \mathbb{R}^{H\times H}$ に対して適用することを考える．しかし，行列である重みフィルタに対してベクトル分解を直接適用することは困難である．そこで，重みフィルタをベクトル表現に変換し，H^2 次元のベクトルとして扱うことでベクトル分解を適用可能にする．また，1 枚当たりのフィルタサイズが小さい場合でもベクトル分解・近似内積計算の効果が得られるように，重みフィルタを 1 枚だけではなくチャンネル方向のフィルタも利用してベクトル表現する．従って，分解する重みベクトルは $\mathbf{W}_n \leftarrow \{\mathbf{w}_{n,1}, \mathbf{w}_{n,2}, ..., \mathbf{w}_{n,D}\} \in \mathbb{R}^{D\cdot H^2}$ と定義できる．畳み込み層では，各重みベクトル $\mathbf{W}$ に対してベクトル分解を適用する．

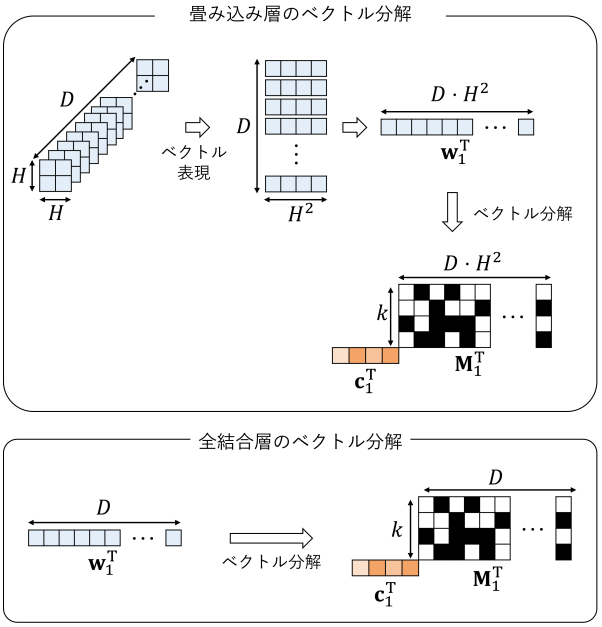

図 2　ベクトル分解の流れ

3.1.2. 全結合層へのベクトル分解適用

全結合層のベクトル分解の流れを図 2 に示す．全結合層へのベクトル分解は，第 l 層目における n 番目のノードに対応する結合重み $\mathbf{w}_n^l \in \mathbb{R}^D$ に対して適用することを考える．全結合層において，n 番目のノードに対応する結合重みは D 次元であり，出力ノード数 N だけ存在する．全結合層では，各重みベクトル $\mathbf{w}^l$ に対してベクトル分解を適用する．

3.2. Quantization sub-layer による入力の量子化

二値化した重みベクトルと特徴量の近似内積計算を行うためには，負の実数値で構成された特徴量も二値化する必要がある．しかし，特徴量は入力サンプルに依存しているため，重みベクトルのように事前に二値化することは困難である．そこで，オンラインで実数値を高速に量子化する Quantization sub-layer を導入する．Quantization sub-layer は量子化幅を可変にすることで負の実数値の量子化を可能にする．具体的にはまず，特徴量 $\mathbf{z}_{i,j,m}^l$ の最大値と最小値から量子化幅 Δd を式 (2) を用いて求める．

$$\Delta d = \frac{\max(\mathbf{z}_{i,j,m}^l) - \min(\mathbf{z}_{i,j,m}^l)}{2^Q - 1} \qquad (2)$$

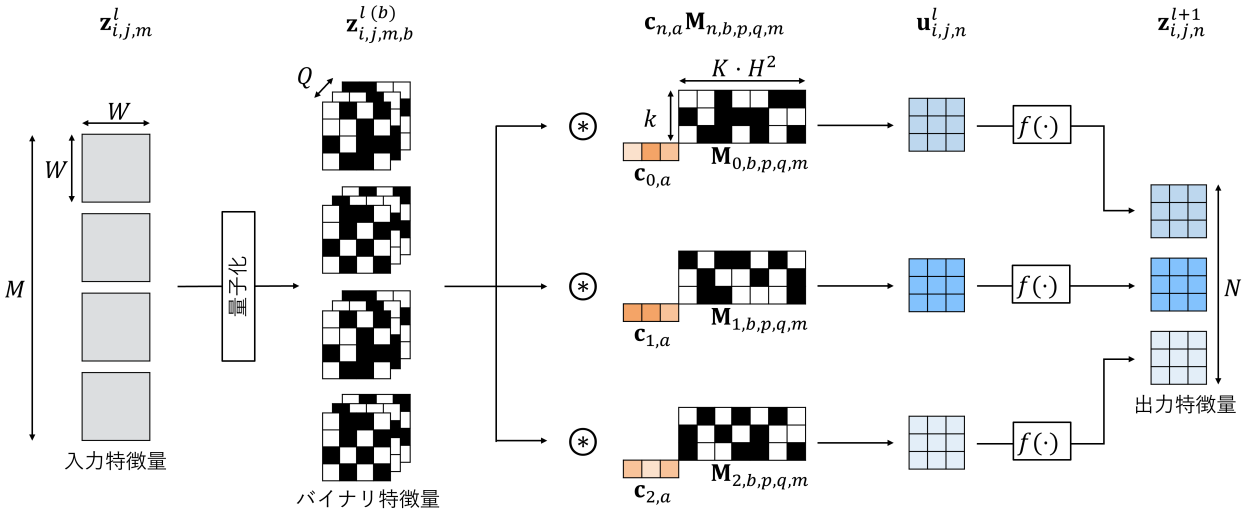

図 3　畳み込み層の推論処理の流れ

ここで，Q は量子化ビット数である．Quantization sub-layer による量子化では，Q が大きいほど量子化の近似精度が向上する一方，近似内積計算に必要な計算量が増加し計算速度が低下する．また，Q が小さいほど近似計算に必要な計算量が減少し計算速度が高速化する一方，量子化の近似精度が低下する．次に，特徴量の最小値が 0 になるように式 (3) を用いて値をシフトする．

$$\mathbf{z}^l_{i,j,m}{}' = \frac{\mathbf{z}^l_{i,j,m} - \mathbf{1}\min(\mathbf{z}^l_{i,j,m})}{\mathbf{1}Q} \tag{3}$$

特徴量をシフトすることで，本来は量子化できない負の値を量子化することが可能となる．最後に，シフトした特徴量 $\mathbf{z}^l_{i,j,m}{}'$ を量子化する．このとき，特徴量は実数値であるため，四捨五入を用いて整数値に丸めて量子化する．量子化後はバイナリコード $\mathbf{z}^{l\,(b)}_{i,j,m,b} \in \{0,1\}$ が生成される．ここで，b はビット番号である．生成されたバイナリコードは式 (4) を用いて復元可能である．

$$\begin{aligned}\mathbf{x}_i &\approx \sum_{b=0}^{Q-1} 2^b \Delta d \mathbf{z}^{l\,(b)}_{i,j,m,b} + \mathbf{1}\min(\mathbf{z}^l_{i,j,m}) \\ &\approx \Delta d \sum_{b=0}^{Q-1} 2^b \mathbf{z}^{l\,(b)}_{i,j,m,b} + \mathbf{1}\min(\mathbf{z}^l_{i,j,m})\end{aligned} \tag{4}$$

式 (4) より，第一項で復元係数を用いてバイナリコードの復元を行い，第二項で特徴量をシフトする際に生じたオフセットを計算している．

3.3.　推論処理

BdDETR では，畳み込み層と全結合層の積和計算を二値同士の近似内積計算に置き換えることで識別計算を高速化する．

3.3.1.　畳み込み層における識別計算

通常の畳み込み計算では，実数値で表される特徴量 $\mathbf{z}_{i,j,m}$ の局所領域と重みフィルタ $\mathbf{h}_{p,q,m,n}$ の積和計算により n 番目の特徴量 $\mathbf{u}_{i,j,n}$ の出力を得る．BdDETR では実数値による積和計算を二値に変換して出力を計算する．BdDETR の畳み込み層の推論処理を図 3 に示す．まず，入力特徴量を Quantization sub-layer により量子化する．これにより，二値特徴量 $\mathbf{z}^{l\,(b)}_{b,i} \in \{0,1\}^{QDW^2}$ が生成される．その後，二値特徴量とベクトル分解により得られた二値基底行列 $\mathbf{M}_{n,b,p,q,m}$ とスケール係数ベクトル $\mathbf{c}_{m,a}$ を用いて畳み込み計算を行う．l 層目の出力 $\mathbf{z}^{l+1}_{i,j,n}$ は式 (5) を用いて表される．

$$\begin{aligned}\mathbf{z}^{l+1}_{i,j,n} &= f(\mathbf{u}^l_{i,j,n}) \\ \mathbf{u}^l_{i,j,n} &= \sum_{m=0}^{D-1}\sum_{p=0}^{H-1}\sum_{q=0}^{H-1} \mathbf{z}^l_{i+p,j+q,m}\mathbf{h}_{p,q,m,n} \\ &\approx \sum_{a=0}^{k-1} \hat{\mathbf{c}}_{n,a}\Delta d \sum_{b=0}^{Q-1} 2^b \sum_{m=0}^{D-1}\sum_{p=0}^{H-1}\sum_{q=0}^{H-1} \\ &\qquad \hat{\mathbf{M}}_{n,b,p,q,m}\mathbf{z}^{l\,(b)}_{i+p,j+q,m,b} + \mathbf{1}\min(\mathbf{z}^l)\end{aligned} \tag{5}$$

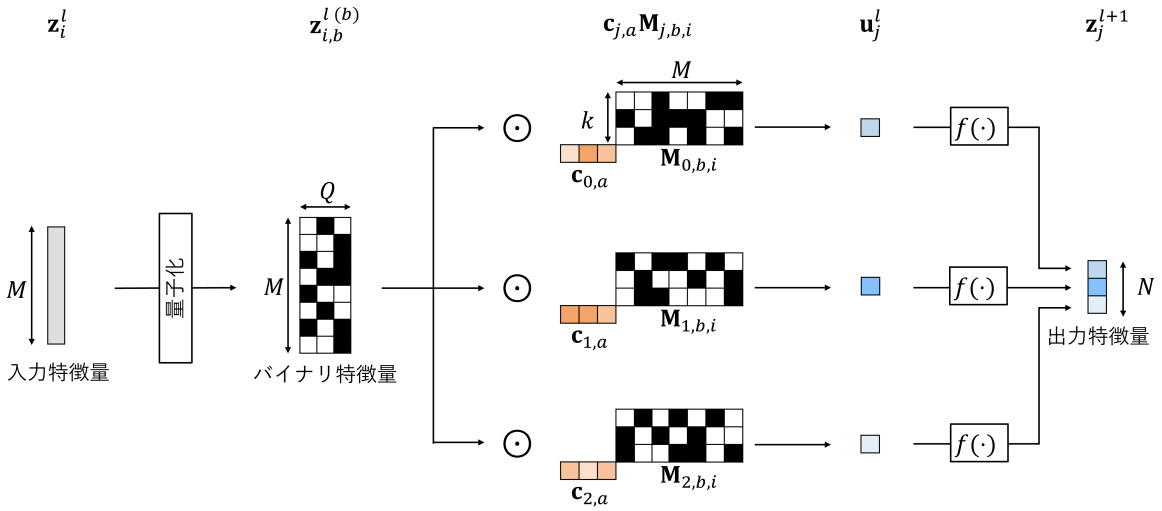

図 4　全結合層の推論処理の流れ

3.3.2.　全結合層における識別計算

通常の全結合層では実数値の特徴量 $\mathbf{z}_i^l$ と結合重み $\mathbf{h}$ の内積計算により $\mathbf{u}_j$ の出力を得る．内積計算を効率的に行うために，畳み込み層と同様に実数値を二値に変換して内積計算を行う．BdDETR の全結合層の推論処理を図 4 に示す．まず，入力特徴量を Quantization sub-layer により量子化する．これにより，入力特徴量 $\mathbf{z}^l \in \mathbb{R}^D$ から二値特徴行列 $\mathbf{z}^{l\,(b)} \in \{0,1\}^{D\times Q}$ が生成される．そして，出力ノード j の出力を二値特徴行列と事前に分解した二値の重みベクトルを用いて式 (6) により，近似計算が行われる．

$$\begin{aligned}\mathbf{u}_j^l &= \sum_{i=0}^{D-1} \mathbf{h}_{j,i}\mathbf{z}_i^l \\ &\approx \sum_{a=0}^{k-1} \hat{\mathbf{c}}_{j,a}\Delta d \sum_{b=0}^{Q-1} 2^b \sum_{i=0}^{D-1} \hat{\mathbf{M}}_{j,b,i}\mathbf{z}_{b,i}^{l\,(b)} + \mathbf{1}\min(\mathbf{z}^l)\end{aligned} \tag{6}$$

式 (6) の $\hat{\mathbf{M}}_{j,b,i} \in \{-1,1\}$ と $\mathbf{z}_{b,i}^{l\,(b)} \in \{0,1\}$ は二値であるため，式 (7) を用いて論理演算とビットカウントにより計算することができる．

$$\begin{aligned}&\hat{\mathbf{M}}_{j,b,i}^l\mathbf{z}_{b,i}^{l\,(b)} = \\ &\quad 2\times \mathrm{POPCNT}(\mathrm{AND}(\hat{\mathbf{M}}_{j,b,i}^l, \mathbf{z}_{b,i}^{l\,(b)})) - \parallel \mathbf{z}_i^l \parallel\end{aligned} \tag{7}$$

ここで，ビットカウントは StreamingSIMD Extensions (SSE) 4.2 に実装されている POPCNT 関数を使用することで高速に演算が可能である．

4.　評価実験

本章では，提案手法である BdDETR の有効性を確認するため，DETR と BdDETR のモデルサイズと物体認識精度，推論時間を比較する．

4.1.　実験条件

本実験では，Common Objects in Context (COCO) 2017 [8] を評価用データセットに使用する．COCO2017 とは，物体検出を含む画像認識タスク用データセットである．学習用として 118,000 枚，評価用として 40,700 枚，検証用として 5,000 枚が用意されている．比較で使用するモデルのベースは，ImageNet [9] で事前学習された ResNet-50 [10] を backbone として採用し，COCO2017 で学習された DETR の重みを使用する．

評価において，バッチサイズは 2，初期化回数の最低値は 65，初期化回数の最大値は 75，基底数と量子化ビット数は 8 とする．また，計算に使用する CPU は Intel Core i9-12900，GPU は GeForce RTX 3080 Ti とする．

本実験では，物体認識精度を Average Precision (AP) を用いて評価する．AP とは，モデルが検出した結果のうち正しく検出できた割合を [0, 1] で表す評価指標であり，1 に近いほど物体精度が高いことを示す．AP の下付き文字の S，M，L はバウンディングボックスのサイズを表しており，それぞれ Small，Medium，Large を対象にして算出した精度を示す．

表 1　物体認識精度とモデルサイズ，推論時間の比較

モデル	適用レイヤ	AP	AP_S	AP_M	AP_L	モデルサイズ [MB]	推論時間 [s]
DETR (GPU)	-	44.9	24.0	51.0	66.5	158.1	1.8
DETR (CPU)	-	44.9	24.0	51.0	66.5	158.1	444.6
BdDETR (CPU)	backbone	42.0	22.7	46.2	64.4	92.6	193.6
BdDETR (CPU)	encoder	41.9	21.1	46.9	64.5	136.8	414.8
BdDETR (CPU)	decoder	42.5	20.9	47.5	65.4	132.7	411.8
BdDETR (CPU)	input_proj	42.8	22.7	48.4	65.8	156.6	411.3
BdDETR (CPU)	class_embed	42.7	21.1	47.4	65.7	158.0	417.3
BdDETR (CPU)	bbox_embed	42.0	21.5	46.7	65.1	157.7	415.5
BdDETR (CPU)	All	40.0	20.1	43.8	62.6	44.0	183.3
BdDETR (CPU)	ba, en, de	41.0	20.9	45.8	63.6	45.9	186.2

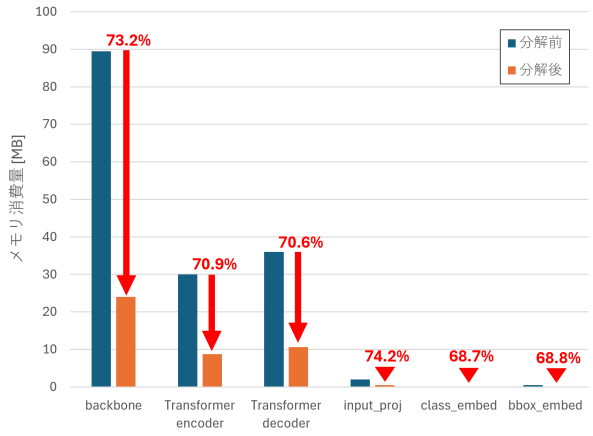

図 5　各層におけるメモリ圧縮率

4.2.　メモリ圧縮率の比較

各層におけるメモリ圧縮率を図 5 に示す．図 5 より，各層の重みをベクトル分解することで backbone は 89.45MB から 23.98MB で 73.2%，encoder は 29.99MB から 8.71MB で 70.9%，decoder は 35.99MB から 10.59MB で 70.6%，input_proj は 2.00MB から 0.51MB で 74.2%，class_embed は 0.09MB から 0.02MB で 68.7%，bbox_embed は 0.50MB から 0.15MB で 68.8% のメモリ圧縮率であることが確認できる．また，モデル全体で約 158MB のパラメータをベクトル分解により約 44MB に削減し，全体のメモリ圧縮率は約 72.2% であることが確認できる．この結果より，backbone，encoder，decoder のモデルサイズが bbox などと比べると大きいことから，backbone，encoder，decoder の圧縮がモデル全体の効率化において重要である．

4.3.　物体認識精度の比較

AP により算出した物体認識精度 [%] とモデルサイズ，推論時間の平均を表 1 に示す．ここで，DETR (CPU) は CPU のみで動作するように計算方法を変更している．また，適用レイヤはベクトル分解を適用したレイヤを表しており，ba は backbone，en は transformer.encoder，de は transformer.decoder，All はすべてのレイヤにベクトル分解を適用した結果である．表 1 より，全ての層にベクトル分解法を適用した場合，モデルサイズは 72.1% 削減できているが精度は 4.9pt 低下したことが確認できる．メモリ消費量が少ないレイヤである input_proj，class_embed，bbox_embed にベクトル分解法を適用すると精度低下は少ないが，モデルサイズや推論速度も削減量が少ない．一方で，ベクトル分解法をメモリ消費量が多いレイヤである backbone，encoder，decoder のみに限定すると，精度低下を 3.9pt に抑制しつつメモリサイズを 70.9% 削減できたことが確認できる．また，BdDETR の推論速度は DETR よりも大幅に高速化したことが確認できる．以上のことから，精度低下を抑制しつつモデル圧縮する場合は適用レイヤをメモリ消費量が多いレイヤのみに絞ることが有効である．

4.4.　基底数と量子化ビット数を変更した場合の物体認識精度の比較

BdDETR は精度低下を抑制しつつモデル圧縮を行う場合，メモリ消費量が多いレイヤに限定してベクトル分解を適用することが有効であると考えられる．そこで，メモリ消費量が最も多い backbone

表 2　backbone の基底数と量子化ビット数を変更した場合の比較

モデル	基底数 量子化ビット数	AP	AP_S	AP_M	AP_L	モデルサイズ [MB]	推論時間 [s]
DETR	-	44.9	24.0	51.0	66.5	158.1	444.6
BdDETR	8	41.0	20.9	45.8	63.6	45.9	186.2
BdDETR	7	41.0	21.8	44.5	63.3	42.9	163.1
BdDETR	6	39.2	21.6	43.6	59.2	39.9	140.5
BdDETR	5	31.7	13.4	32.8	54.6	36.9	121.0
BdDETR	4	11.5	7.1	15.1	20.2	33.9	105.5
BdDETR	3	0.4	0.0	0.7	0.7	30.9	92.4
BdDETR	2	0.0	0.0	0.0	0.0	27.9	83.4

の基底数と量子化ビット数を変更し，物体認識精度を比較する．AP により算出した物体認識精度 [%] とモデルサイズ，推論時間の平均を表 2 に示す．ここで，DETR と BdDETR は CPU 上で実行し，BdDETR は backbone，encoder，decoder のみにベクトル分解を適用する．また，encoder と decoder の基底数と量子化ビット数は 8 に固定し，backbone の基底数と量子化ビット数のみを変更する．表 2 より，backbone の基底数と量子化ビット数を減少させることで，モデルサイズと推論時間が大幅に削減されることが確認できる．一方で，物体認識精度は基底数と量子化ビット数の減少に伴い低下していることが確認できる．これは，ベクトル分解による近似誤差が大きくなり，backbone で抽出された特徴量の正確性が失われるためだと考えられる．このことから，backbone の圧縮設定が全体の効率性を保ちながら認識精度を維持する上で重要な要因であると考えられる．以上のことから，BdDETR がエッジデバイス上での効率的な物体認識を可能にする一方，精度低下を最小限に抑えるために，どのレイヤにベクトル分解を適用するかや基底数・量子化ビット数の最適化が重要であることが考えられる．

5. おわりに

本研究では，学習済み DETR の重みと特徴量をベクトル分解法により二値化し，近似内積計算を適用したモデル圧縮と推論の高速化を再学習なしで行う Binary-decomposed DETR を提案した．評価実験では，モデル全体で約 158MB のパラメータをベクトル分解により約 44MB に削減できることを確認した．また，精度低下を抑制しつつモデル圧縮する場合，全てのレイヤに対してベクトル分解を適用するのではなく，メモリ消費量が多いレイヤのみに限定してベクトル分解を適用することが有効であることを確認した．今後は，エッジデバイス上で動作可能にするために推論時に使用する CPU の命令セット変更や，エッジデバイス上で物体認識精度を比較して提案手法の有効性を確認する．

参考文献

[1] N. Carion, *et al.*,“End-to-End Object Detection with Transformers ”, ECCV, 2020.
[2] A. Vaswani, *et al.*, “ Attention is all you need ”, NeurIPS, 2017.
[3] X. Zhu, *et al.*,“ Deformable detr: Deformable transformers for end-to-end object detection ”, arXiv preprint arXiv:2010.04159, 2020.
[4] F. Li, *et al.*,“ Dn-detr: Accelerate detr training by introducing query denoising ”, CVPR, 2022.
[5] J. Chang, *et al.*,“ DETRDistill: A Universal Knowledge Distillation Framework for DETR-families ”, ICCV, 2023.
[6] S. Xu, *et al.*,“Q-DETR: An Efficient Low-Bit Quantized Detection Transformer ”, CVPR, 2023.
[7] Y. Yamauchi, *et al.*,“Distance Computation Between Binary Code and Real Vector for Efficient Keypoint Matching ”, CVA, 2013.
[8] T. Lin, *et al.*,“ Microsoft coco: Common objects in context ”, ECCV, 2014.
[9] J. Deng,, *et al.*,“ImageNet: A large-scale hierarchical image database ”, CVPR, 2009.
[10] K. He, *et al.*,“ Deep Residual Learning for Image Recognition ”, CVPR, 2016.

井上 涼太：2023 年 中部大学工学部情報工学科卒業，現在 中部大学大学院修士課程在学中．異常検知，モーション生成の研究に従事．

大杉 佳史：2024 年 中部大学工学部情報工学科卒業．物体検出，モデル圧縮の研究に従事．

平川 翼：2013 年 広島大学大学院博士課程前期修了，2014 年 広島大学大学院博士課程後期入学，2017 年 中部大学研究員

(〜2019 年)，2017 年 広島大学大学院博士後期課程修了．2019 年 中部大学特任助教，2021 年 中部大学講師．2014 年 独立行政法人日本学術振興会特別研究員 DC1．2014 年 ESIEE Paris 客員研究員 (〜2015 年)．コンピュータビジョン，パターン認識，医用画像処理の研究に従事．

山下 隆義：2002 年 奈良先端科学技術大学院大学博士前期課程修了，2002 年 オムロン株式会社入社，2011 年 中部大学大学院博士後期課程修了 (社会人ドクター)，2014 年 中部大学講師，2017 年 同大学准教授，2021 年 同大学教授．人の理解に向けた動画像処理，パターン認識・機械学習の研究に従事．

藤吉 弘亘：1997 年 中部大学大学院博士後期課程修了, 1997 年 米カーネギーメロン大学ロボット工学研究所 Postdoctoral Fellow, 2000 年 中部大学工学部情報工学科講師, 2004 年 中部大学准教授, 2005 年 米カーネギーメロン大学ロボット工学研究所客員研究員 (〜2006 年), 2010 年 中部大学教授, 2014 年 名古屋大学客員教授．計算機視覚，動画像処理，パターン認識・理解の研究に従事．