Vol.2021-MBL-98 No.11
Vol.2021-UBI-69 No.11
2021/3/1

精度, 推論速度及び消費電力の制御による ニューラルネットワークの自動構築化手法

小澤 遼[1,a)] 大越 匡[1] 柘植 晃[1,2] 中澤 仁[3] 若月 駿尭[4,b)] 岸本 康成[4,c)] 豊田 真智子[4,d)]
八木 哲志[4,e)] 寺本 純司[4,f)]

概要：第 5 世代移動通信システム (5G) の実用化に伴い, IoT の普及が期待される. 近年, AI の基盤技術であるニューラルネットワーク (以下 NN と呼ぶ) を学習する深層学習に関する研究が盛んに行われており, IoT デバイスへの AI の搭載により社会課題の解決や業務の効率化が期待される. 一方で, あらゆる「モノ」がインターネットに接続されるようになるため, トラフィックの削減や通信を介した処理の高速化のため, エッジコンピューティングが重要となる. エッジコンピューティングのように IoT デバイスに AI を搭載し, IoT デバイス自身で処理するためには, 計算資源や消費電力に制約が生じ, 制約を満たすような AI の搭載が必須となる. これらの制約を満たすような AI を設計するためには, AI の基盤技術である NN に関する高度な専門知識と多くの時間を要する. 本研究では, NN を自動構築する Neural Architecture Search(NAS) 分野の研究である FBNet の損失関数を改良することにより, 特定デバイスに特化した画像分類タスクにおける精度, 推論速度, 消費電力を考慮した NN の自動構築手法を提案する. 入力として 達成したい精度を与えることで, 特定デバイスに特化した用途に応じた NN を自動構築することが可能である.

A Method for Automatic Construction of Neural Networks by Controlling Accuracy, Inference Speed and Power Consumption

Ryo Ozawa[1,a)] Tadashi Okoshi [1] Akira Tsuge[1,2] Jin Nakazawa[3] Toshiaki Wakatsuki[4,b)]
Yasunari Kishimoto[4,c)] Machiko Toyoda[4,d)] Satoshi Yagi[4,e)] Junji Teramoto[4,f)]

1. はじめに

5G の実用化に伴い, IoT の更なる普及が期待される. 総務省 [1] によると世界における IoT デバイスの数は, 2015 年と 2022 年を比較すると約 2 倍であり急激に増加することが予想される. また, 日本, 米国, 英国, ドイツの企業において, 2025 年までには, 大多数が AI, IoT を導入することが予想される [2].

IoT という概念のように世の中の, あらゆる「モノ」がインターネットに接続可能となり, センサを通じて実世界のデータが大量に取得可能になる. 従って, IoT デバイスから得られたデータを活用し, AI の基盤技術である NN を学習する深層学習を行い, 様々な IoT デバイスと AI を連携させることで, 業務の効率化や社会課題の解決が期待で

[1] 慶應義塾大学大学院政策・メディア研究科
Graduate School of Media and Governance, Keio University, 5322 Endo Fujisawa-shi, Kanagawa, 252-0882, Japan
[2] YRP 研究開発推進協会
YRP R&D Promotion Committee, YRP Center 1st Bldg. 3-4 Hikarinooka, Yokosuka-shi, Kanagawa, 239-0847 Japan
[3] 慶應義塾大学環境情報学部
Faculty of Information and Environment Keio University, 5322 Endo Fujisawa-shi, Kanagawa, 252-0882, Japan
[4] 日本電信電話株式会社 NTT ソフトウェアイノベーションセンタ
NTT Software Innovation Center 3-9-11, Midori-cho Musashino-shi, Tokyo, 180-8585, Japan
[a)] oza15015@keio.jp
[b)] toshiaki.wakatsuki.xg@hco.ntt.co.jp
[c)] yasunari.kishimoto.by@hco.ntt.co.jp
[d)] machiko.toyoda.hk@hco.ntt.co.jp
[e)] satoshi.yagi.ue@hco.ntt.co.jp
[f)] junji.teramoto.hg@hco.ntt.co.jp

きる. IoT デバイスの一例としてドローン, 自動車, 産業用ロボット, 家電等が挙げられる. これらの IoT デバイスと AI の連携の一例として 4G, 5G 等の通信を介したクラウド上での処理, つまり, クラウドコンピューティングが考えられる. このメリットは, IoT デバイスの CPU 性能の制約を考慮しなくて良い点である. 一方で, IoT デバイスをはじめとする膨大な数の「モノ」とクラウド間でのデータの送受信が生じるため通信量が膨大になり, 通信速度の低下や AI モデルの処理性能の低下が懸念される. 更に, 通信障害が発生すると AI モデルの推論が困難であることや通信遅延に伴う, 推論処理の遅延も懸念される.

近年, クラウドコンピューティングに対してエッジコンピューティングの重要性が高まっている. エッジコンピューティングとは, IoT デバイス等の個々のデバイス自身またはデバイス付近で処理を行う方式のことである. IoT デバイスでは GPU や一般的なコンピュータに搭載されている CPU と比較すると計算資源に限りがある. 精度の高い AI モデルは, パラメータ数が多くなり, 計算資源の乏しい IoT デバイスでは推論に非常に多くの時間を要する, または動作しない可能性がある. 従って, IoT デバイスでも動作するような軽量な AI モデル, つまり, 推論速度が高速な AI モデルの構築が求められる.

エッジコンピューティングのように IoT デバイス上で AI モデルの推論を行うためには, 推論速度に加えて, 消費電力にも制約が生じる. [3] によると CNN 等の深層学習技術を用いた AI モデルを IoT デバイスへの搭載が困難である理由は, 単位電力当たりの推論性能が不足しているためである. 現在, 製品化されている物体認識の AI モデルの電力性能は, 1TOPS/W 前後である. 一方で, 要求される推論性能は小型端末でも 2.2TOPS 以上で, 少なくとも 2W 以上の電力を必要とする. IoT デバイス等の小型端末に AI モデルを搭載し, 幅広く普及させるためには, 1W 当たりの推論性能を 100〜1000 倍に向上させる必要がある. このように IoT デバイスに AI モデルを搭載させるためには, 低消費電力である必要性や 1W 当たりの推論性能を向上させる必要性がある.

以上のことから, IoT デバイスに AI モデルを搭載するためには, 推論速度が高速で低消費電力である必要がある. 精度, 推論速度, 消費電力を考慮した NN の設計は, 高度な専門知識や多くの時間を要する. このような, 背景から NN を自動構築する NAS が登場した. NAS によって精度, 推論速度, 消費電力を考慮した NN の自動設計が可能であるが, 設定するパラメータによってどのような, NN が設計されるか不明確であることが多い. これでは, NN に関する知識が乏しい人が, NAS を用いて NN を設計することが困難であると考えられる. また, IoT デバイス等の小型端末に NN を搭載する上で, 用途に応じて精度, 推論速度, 消費電力について, 重要視する項目が異なる.

本研究では, NAS 分野の研究である FBNet[4] の損失関数を改良し, 2 つの手法を提案することにより, 特定デバイスに特化した画像分類タスクにおける精度, 推論速度, 消費電力を考慮した NN の自動構築手法を提案する. 入力として 達成したい精度を与えることで, 「精度を犠牲にしてでも, 高速で低消費電力」,「精度は程良く, 高速で消費電力」等の用途に応じた NN を自動構築することが可能である. 提案手法の有効性を検証するために, 提案手法と既存手法を画像分類タスクにおいて, 定量的な評価や構築される NN の構造を比較した.

2. 関連研究

2.1 Neural Architecture Search(NAS)

先行・関連研究について述べる上で, 必要であり, 本研究のベースである NAS について述べる. NAS とは, 図 3 に示すように NN の構造やパラメータを自動的に最適化する手法のことである. NAS は NN の構造を設計 (最適化), 学習時のパラメータの最適化, 学習の全てまたは一部を手動ではなく, 自動で行う. 本研究では, NAS の中でも, NN の構造の設計を自動で行うことを対象とする.

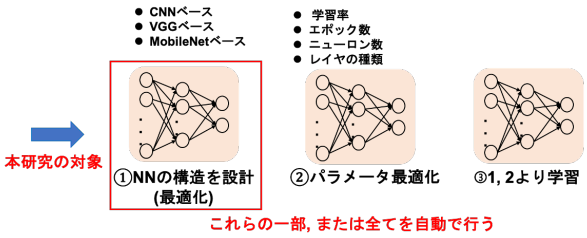

図 1 NAS の概要図

2.2 微分可能な NAS

古典的な NAS 手法では, 探索空間が離散的であるため微分ができず, NN の構造の自動設計 (アーキテクチャ探索) と学習を別々に行っている. つまり, 1 度探索された NN のアーキテクチャ (構造) を用いて NN を学習し, 精度を評価し, 精度が高くなるように繰り返して 探索するため多くの時間を要する. この課題を解決するために登場した研究が, DARTS[5] である. 離散的な探索空間を連続的な空間へ緩和することで, 従来の NN を学習するように, NN の構造の探索と学習を同時に行うことが可能になり, 探索の要する時間の大幅な削減を実現した. [5] の登場により, NAS 研究の多くは探索空間が連続的で微分可能であるため, 本研究でも微分可能な NAS を用いる.

2.3 精度・推論速度を考慮した NAS

計算資源に限りのある IoT デバイス等の小型端末に AI モデルを搭載するためには, NN の精度と推論速度を考慮して設計する必要がある. このような背景から, 精度, 推

論速度を考慮した NAS に関する研究がおこなわれている. [6] では携帯電話で NN モデルを実行し, latency(推論速度) を計測して損失関数に組み込むことで高精度を維持した推論速度が高速なモデルの自動構築を実現している. 従来の NAS ではオペレーション候補毎に分岐する構造であるが, [7] では分岐せずに単一のパスで探索することで学習を高速化させ, [6] の 5000 倍の高速化を実現し, 精度も維持した. [8] では, NAS 手法を用いて NN のレイヤ毎の量子化による圧縮により推論速度が高速なモデルの自動構築を実現した. [9] では強化学習を用いて精度を担保し, 推論速度が高速なモデルの自動構築を実現した. また, [10] では, デイバイス毎に NN の設計が不要な NAS の提案により, FBNet[4] や DARTS[5] よりも計算コストを大幅に削減しつつ, 精度を担保し, 推論速度が高速なモデルの自動構築を実現した. しかし, これらの研究では NN の精度, 推論速度のみを考慮した探索であり, 消費電力まで考慮できていない.

2.4 本研究のベースとなる研究 (FBNet)

本研究のベースとなる FBNet[4] について述べる. FBNet とは, 2.3 で述べた精度, 推論速度を考慮した NAS の一種であり, 図 2 に示すように微分可能な NAS 手法を用いて精度と特定デバイスの推論速度を考慮して NN の自動構築を実現した手法である. FBNet は, 損失関数に基づいて最適な NN の構造を学習 (探索) する NN と最終的に構造が決定した NN の学習という 2 段階で成り立つ. デバイス毎に推論速度を計測し, 損失関数に組み込むため, デバイスにとって最適な NN の構築が可能である.

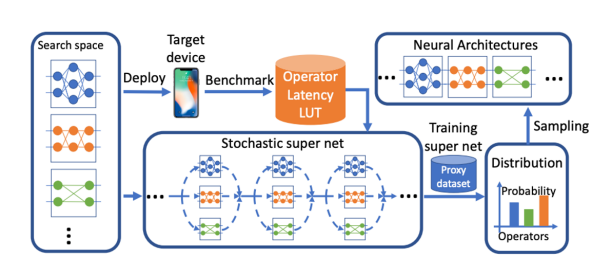

図 2 FBNet の概要図 [4]

図 3 に FBNet の探索空間とオペレーション候補を示す. 図中, 左の a において, Input Shape は入力される画像サイズ, Block は NN のレイヤの種類またはオペレーション候補 (フィルタの種類), c は出力チャネル数, n は Block の数, s はフィルタのストライド数を表している. Block 中の TBS は, NN のアーキテクチャ探索 (自動構築) 対象であり, 図中右, b の Block type のいずれかが選ばれる. また, Block 中の fc は全結合層であり, ImageNet データセットの画像を 1000 クラスに分類するため出力チャネル数を 1000 としている. 図中右の b はオペレーション候補であり, Expansion は入力画像に対する拡大比率, Kernel はフィルタサイズ, Group は畳み込みの Group を表している. FBNet では, 22Block に対して 9 通りのオペレーション候補, つまり, 合計 9^{22} 通りの組み合わせの中から最適な組み合わせを探索することになる.

Input Shape	Block	c	n	s
$224^2 \times 3$	3×3 conv	16	1	2
$112^2 \times 16$	TBS	16	1	1
$112^2 \times 16$	TBS	24	4	2
$56^2 \times 24$	TBS	32	4	2
$28^2 \times 32$	TBS	64	4	2
$14^2 \times 64$	TBS	112	4	1
$14^2 \times 112$	TBS	184	4	2
$7^2 \times 184$	TBS	352	1	1
$7^2 \times 352$	1×1 conv	1984	1	1
$7^2 \times 1504$	7×7 avgpool	-	1	1
1504	fc	1000	1	-

(a) FBNetの探索空間

Block type	Expansion	Kernel	Group
k3_e1	1	3	1
k3_e1_g2	1	3	2
k3_e3	3	3	1
k3_e6	6	3	1
k5_e1	1	5	1
k3_e1_g2	1	5	2
k5_e3	3	5	1
k5_e6	6	5	1
skip	-	-	-

(b) FBNetのオペレーション候補

図 3 FBNet の探索空間とオペレーション候補 [4]

続いて, FBNet の損失関数について述べる. 式 1 に FBNet の損失関数を示す. $CE(\alpha, w_\alpha)$ は選択されたアーキテクチャ α における画像分類誤差であり, 交差エントロピーを用いている. $\alpha \log(LAT(\alpha))^\beta$ は推論速度の損失であり, α, β は設定すべきパラメータである. 式 2 に LAT(α) の算出方法を示す. $b_l^{(a)}$ はアーキテクチャ α における l 層の Block_type, つまりオペレーション候補の推論速度である. NN のアーキテクチャは複数の層から成り立つため, 探索対象の全ての層の推論速度の和を LAT(α) として算出する. 式 1 によって画像分類誤差と推論速度の損失を最小化, つまり, 精度を担保しつつ, 推論速度が高速な NN を探索により構築することが期待できる.

$$L(\alpha, w_\alpha) = CE(\alpha, w_\alpha) * \alpha \log(LAT(\alpha))^\beta \quad (1)$$

$$LAT(\alpha) = \sum_l LAT(b_l^{(a)}) \quad (2)$$

FBNet では消費電力を考慮した探索はできていない. 一方で, MobileNetV2[11] の構造をベースとした探索であるため, 精度を担保した状態で推論速度が高速な NN を探索により構築することが期待できる. 本研究では, FBNet をベースとして精度, 推論速度, 消費電力を考慮して NN を自動構築する.

2.5 精度・推論速度・消費電力を考慮した NAS

IoT デバイス等の小型端末に AI モデルを搭載するためには, 推論速度に加えて消費電力も考慮する必要がある. [12] では遺伝的アルゴリズムを用いて精度, 推論速度, 消費電力を考慮して NN の自動構築を実現した. [13] では, FBNet をベースとして, 損失関数を改良することにより, 精度・推論速度・消費電力を考慮して NN の自動構築を実現した.

しかし, [12] は事前に NN の構造を設計し, 探索範囲を設定する必要があるため, NN の知識を有さない人が探索することは困難である. [13] は, 複数のパラメータを事前に設定する必要があり, パラメータにより どのような NN が探索により構築されるかが不明確である.

3. 提案手法

本章では, 精度, 推論速度, 消費電力を考慮し, 目標の精度 (正解率) を入力として与え, 用途に応じた NN を自動構築するための 2 つの提案手法について述べる.

3.1 本研究における NN の基本構造と探索空間

本研究では, 10 クラスに分類された 32 ピクセル*32 ピクセルの画像データセット CIFAR-10 を用いる. 従って, 画像を 10 クラス分類するタスクを解くことができる NN のアーキテクチャを探索するために, 表 1 に示すような NN の基本構造と探索空間を定義した. Input Shape は入力される画像サイズ, Block は NN のレイヤの種類またはオペレーション候補 (フィルタの種類), c は出力チャネル数, n は Block の数, s はフィルタのストライド数を表している. Block 中の TBS は, NN のアーキテクチャ探索 (自動構築) 対象であり, 2.4 で述べた FBNet での, 図 2 の右側, b の Block type のいずれかが選ばれる. FBNet と同様に合計 9^{22} 通りの組み合わせの中から最適な組み合わせを探索することになる.

表 1 本研究における NN の基本構造と探索空間

Input Shape	Block	c	n	s
$32^2 \times 3$	3×3 conv	16	1	1
$32^2 \times 16$	TBS	24	1	1
$16^2 \times 24$	TBS	32	4	2, 1, 1, 1
$8^2 \times 32$	TBS	64	4	2, 1, 1, 1
$4^2 \times 64$	TBS	96	4	2, 1, 1, 1
$4^2 \times 96$	TBS	160	4	1, 1, 1, 1
$2^2 \times 160$	TBS	320	4	2, 1, 1, 1
$2^2 \times 320$	TBS	320	1	1
$2^2 \times 320$	1×1 conv	320	1	1
$1^2 \times 1280$	fc	10	1	-

3.2 画像分類の正解率に応じた分類誤差の操作

提案手法 1, 2 に共通することとして, 画像分類の正解率に応じた分類誤差の操作について述べる. 画像分類の正解率に応じた分類誤差の操作を図で表現したものを, 図 4 に示す. 図 4 におけるグラフは, 厳密には画像分類の誤差として用いられる Cross Entropy Loss(交差エントロピー誤差) とは異なるが, 説明のため, このような図を用いることにする. 従来の交差エントロピー誤差では, 誤差と正解率には相関があり, 誤差の値が小さくなるほど, 正解率が上昇する. 本研究では, 事前に設定した目標の正解率 ($Goal_{acc}$) を達成した時点で, 分類誤差の値を 0 になるように操作する. 分類誤差は交差エントロピー誤差であり, 底を e とした対数により算出しており, 連続値であるため, 勾配降下法を用いた最適化が可能である. 目標の正解率を達成した途端, 交差エントロピー誤差の値を 0 へ操作してしまうと誤差の値が離散値になってしまう. このような操作では, 連続値から離散値へ変換されてしまい, 勾配降下法による学習が困難である. 従って, 本研究では 図 4 に示すように目標の正解率 ($Goal_{acc}$) を基点として, 分類誤差の値を連続的な値となるように操作する. 操作後の交差エントロピーのことを Custom Cross Entropy Loss(以下, CCEL と呼ぶ) とする. 目標の正解率に近づくほど CCEL が 0 に近づくように操作する, 図 4 に示したような, 画像分類の正解率に応じた分類誤差の操作のアルゴリズムは 3.4, 3.5 で述べる.

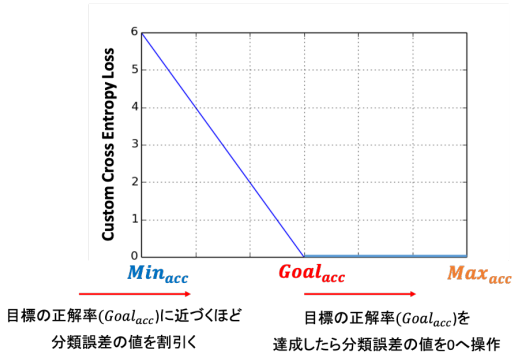

図 4 画像分類の正解率に応じた画像分類誤差の操作

3.3 提案手法 1 における分類誤差の底上げ

図 5 に操作済みの分類誤差, CCEL を底上げするグラフを示す. 左側は一般的な交差エントロピー誤差を示すグラフである. 提案手法 1 では, 図 5 の右側のグラフのように 3.2 で述べた分類誤差を操作した CCEL の値に 1 を加えることで最小値が 1 になるように操作する.

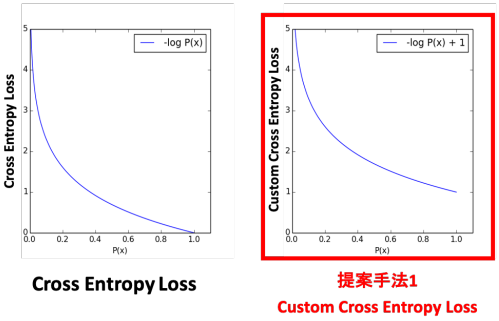

図 5 分類誤差の底上げによる操作

3.4 提案手法 1 のアルゴリズムと損失関数

3.2, 3.3 で述べた内容より, 提案手法 1 のアルゴリズムと

損失関数について述べる. 提案手法 1 のアルゴリズムをアルゴリズム 1 に示す. batch ごとに分類誤差である loss と画像分類における正解率 Acc を算出する. その後, 正解率 Acc を目標の正解率 $Goal_{acc}$ から 0.1 を引いた値で除算し, その値を画像分類誤差操作係数 rate として算出する. そして, loss から loss と rate で積をとった値で, 差をとった値を正解率に応じた分類誤差を操作した値, CCEL とする. 例えば, 目標の正解率が 65%である場合, $Goal_{acc}$ を 0.65 とし, 学習時の正解率が 55%に近づくほど, CCEL の値も 0 に近づく. 更に, 目標の正解率を達成した時点で急激に 0, つまり離散的な変化にならないよう, 目標の正解率 $Goal_{acc}$ を基点として連続的な変換が期待できる. 目標の正解率から, 0.1 引いた値を用いる理由は, 予備実験を通じて目標の正解率を大幅に上回る傾向を確認したためである. また, lat_{loss} と $energy_{loss}$ は, それぞれ推論速度と消費電力の損失である.

Algorithm 1 提案手法 1 のアルゴリズム

for $i = 1$ to $epoch$ **do**
  for $j = 1$ to $len(train\_data)/batch\_size$ **do**
    $loss_{i,j} \leftarrow -\sum_x p(x) \log q(x)$
    $\mathbf{rate_{i,j}} \leftarrow \mathbf{Acc_{i,j}}/(\mathbf{Goal_{acc}} - \mathbf{0.1})$
    if $Acc_{i,j} \geqq (Goal_{acc} - 0.1)$ **then**
      $CCEL_{i,j} \leftarrow 0$
    else
      $\mathbf{CCEL_{i,j}} \leftarrow \mathbf{loss_{i,j}} - (\mathbf{rate_{i,j}} * \mathbf{loss_{i,j}})$
    end if
    $\mathbf{CCEL_{i,j}} \leftarrow \mathbf{CCEL_{i,j}} + \mathbf{1}$
    $\mathbf{TotalLoss_{i,j}} \leftarrow \mathbf{CCEL} * (\mathbf{lat_{loss}} + \mathbf{energy_{loss}})$
    $TotalLoss_{i,j}.backward()$
  end for
end for

アルゴリズム 1 に基づく提案手法 1 の損失関数を式 3 示す. NN 全体の損失である TotalLoss は推論速度の損失 lat_{loss} と消費電力の損失 $energy_{loss}$ の和をとった値と操作済みの分類誤差, CCEL との積をとった値とする. A は探索時のアーキテクチャ, β は事前に設定するパラメータである. 式 3 の損失関数は, CCEL に強く依存する. 従って, CCEL の値が 0 である場合, その他の損失の値に関わらず, TotalLoss も 0 になる. これでは, 推論速度と消費電力の損失が増加しても問題がないと判断される可能性がある. この問題を解決するために, アルゴリズム 1 にて CCEL の最小値が 1 になるよう操作した. 最小値が 1 になるため, 推論速度と消費電力の損失が上昇した場合は全体の損失も増加する. 従って, 推論速度と消費電力の損失を減少させることが期待できる. また, 式 3 に示した提案手法 1 の損失関数は CCEL に強く依存, つまり精度を重要視しつつ, 推論速度と消費電力の損失を減少させることを期待した.

$$Total_{loss} = CCEL * (\log_e(lat(A))^{\beta} + \log_e(energy(A))^{\beta}) \tag{3}$$

3.5 提案手法 2 のアルゴリズムと損失関数

3.2 で述べた内容より, 提案手法 2 のアルゴリズムと損失関数について述べる. 提案手法 2 のアルゴリズムをアルゴリズム 2 に示す. アルゴリズム 1 と大部分が, 同じであるため詳細は割愛し, 違いについて述べる. アルゴリズム 1 とは異なり, 正解率 Acc を目標の正解率 $Goal_{acc}$ から 0.1 を引かずに, $Goal_{acc}$ で除算する. そして, 3.3 で述べたような分類誤差の底上げはしない. 従って, CCEL の最小値は 0 となる.

Algorithm 2 提案手法 2 のアルゴリズム

for $i = 1$ to $epoch$ **do**
  for $j = 1$ to $len(train\_data)/batch\_size$ **do**
    $loss_{i,j} \leftarrow -\sum_x p(x) \log q(x)$
    $\mathbf{rate_{i,j}} \leftarrow \mathbf{Acc_{i,j}}/\mathbf{Goal_{acc}}$
    if $Acc_{i,j} \geqq Goal_{acc}$ **then**
      $CCEL_{i,j} \leftarrow 0$
    else
      $\mathbf{CCEL_{i,j}} \leftarrow \mathbf{loss_{i,j}} - (\mathbf{rate_{i,j}} * \mathbf{loss_{i,j}})$
    end if
    $\mathbf{TotalLoss_{i,j}} \leftarrow \mathbf{CCEL} + \mathbf{lat_{loss}} + \mathbf{energy_{loss}}$
    $TotalLoss_{i,j}.backward()$
  end for
end for

アルゴリズム 2 に基づく提案手法 2 の損失関数を式 4 に示す. NN 全体の損失である TotalLoss は操作済みの分類誤差, CCEL, 推論速度の損失 lat_{loss} と消費電力の損失 $energy_{loss}$ の 3 つの和をとった値とする. 式 3 に示した提案手法 1 の損失関数とは異なり CCEL, つまり, 精度に依存しない. 従って, 提案手法 1 とは異なり, CCEL の値が最小値 0 になっても推論速度と消費電力の損失を減少させることが期待できるため, CCEL を底上げしない. また, 全体の損失が, 3 つの損失の和で表現されているため, 特定の損失に依存しない. 従って, 3 つの損失のバランスを考慮して全体の損失を減少させることが期待できる.

$$Total_{loss} = CCEL + \log_e(lat(A))^{\beta} + \log_e(energy(A))^{\beta} \tag{4}$$

4. 実験

本章では, 3 章で述べた提案手法を用いて実験を行う. 本実験では, 提案手法に基づいて最適な NN のアーキテクチャ (構造) を探索する NN の学習 (NN が NN を生成する) と最終的に構造が決定した NN の学習という 2 段階で成り立つ. 提案手法により, 構造が決定した NN と FBNet を含む既存モデルにおける精度, 推論速度, 消費電力について比

較し, 提案手法の有効性を検証する. 更に, 学習によって得られた NN のアーキテクチャを可視化し, 精度, 推論速度, 消費電力の結果とパラメータ数について考察する.

4.1 実験環境

実験環境について述べる. NN の構造を探索する NN の学習には GPU, アーキテクチャが決定した NN の学習は CPU を用いる. GPU は, NVIDIA GeForce GTX 1080, CPU は Intel(R) Core(TM) i7-6950X CPU@3.00GHz を用いる. 消費電力については, 特殊な装置が必要であるため先行研究 [13] で計測したオペレーション候補毎の熱量 (J) を消費電力の損失として用いる. 計測環境は Raspberry Pi 3 Model B(ARM Cortex-A53 4 コア, 1.2 GHz) である. 推論速度は, CPU 上でオペレーション候補ごとに事前に計測した値を用いる.

4.2 提案手法を用いたアーキテクチャ探索の実験設定

提案手法を用いた NN のアーキテクチャ探索の実験設定について述べる. 提案手法 1 により探索される NN モデルを S1, 提案手法 2 は S2 と呼ぶ. 提案手法 1 との比較のため, FBNet の損失関数を用いて FBNet1 は α を 0.5, β を 0.6, FBNet2 は α を 0.2, β を 0.4 として探索する. FBNet3 は提案手法 2 との比較のため, 先行研究 [13] の損失関数を用いて α を 0.5, β を 0.5, γ を 1.5, θ を 0.5 として探索する. S1, S2 モデルは全てパラメータは β のみで値を 0.5, 目標の正解率 $Goal_{acc}$ を 1.0, 0.8, 0.65, 0.50, 0.35 として探索する. 設定した $Goal_{acc}$ の値を提案手法の後に続くように, モデル名を定義した. 例えば, 提案手法 1 を用いて $Goal_{acc}$ が 0.65 の場合は S1-L65 である. 提案手法 2 において, $Goal_{acc}$ が 1.0 の S2-Normal は, 目標の正解率が 100% であるため, 分類誤差の値は操作しない.

4.3 提案手法の探索により決定した NN の構造

提案手法に基づいて最適な NN の構造を探索する NN の学習結果について述べる. 提案手法の探索により決定した NN の構造を可視化した. 図 6 に提案手法 1 により決定した NN の構造を示す. 既存モデルである MobileNetV2, 改良を加えていない FBNet の損失関数を用いた探索により獲得された FBNet1, FBNet2 と比較すると提案手法の探索によって獲得されたモデルは全て, NN の層数が少なくパラメータサイズが小さい結果となった. 提案手法 1 を用いた NN のアーキテクチャ探索により獲得された NN は目標の正解率が小さいほど, NN の層数とパラメータ数が減少している. 従って, 精度の高いモデルや精度は低いが高速で低消費電力な NN モデル等の用途に応じた NN モデルが探索により構築できていることが期待できる. 図 7 に提案手法 2 により決定した NN の構造を示す. 先行研究 [13] における損失関数を用いた探索により獲得された FBNet3 と比較すると提案手法 2 の探索によって獲得されたモデル S2-Normal, S2-Limit80 とは NN の層数とパラメータ数が同じ結果となった. その他については, 提案手法 2 を用いた NN のアーキテクチャ探索により獲得された NN は目標の正解率が小さいほど, NN の層数とパラメータ数が減少している. 従って, 精度の高いモデルや精度は低いが高速で低消費電力な NN モデル等の用途に応じた NN モデルが探索により構築できていることが期待できる. 2 つの提案手法により決定した NN の構造は, 全て MobileNetV2 よりも層数が少なく, パラメータサイズも小さい結果となった.

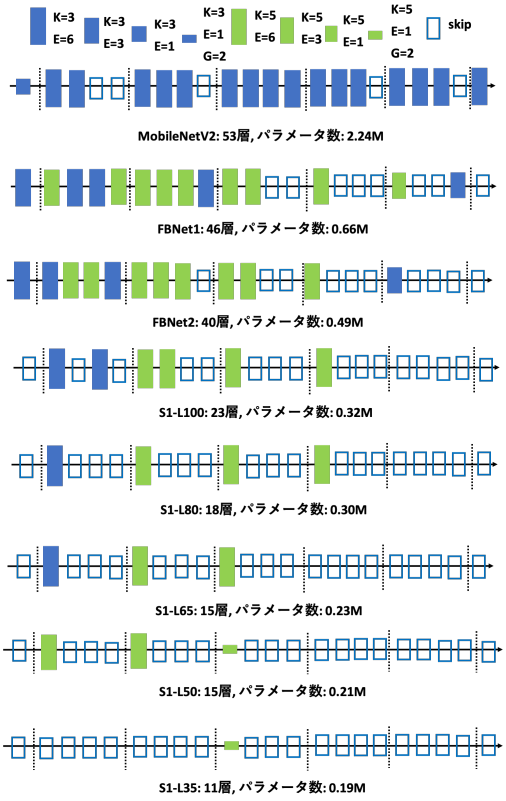

図 6 提案手法 1 の探索により決定した NN の構造

図 7 提案手法 2 の探索により決定した NN の構造

4.4 提案手法により構造が決定した NN の学習

実験の 2 段階目である, 最終的に構造が決定した NN の学習の設定について述べる. epoch は 140, 学習率は 0.01, weight_decay は 1e-4, momentum は 0.9, batch_size は 32 として NN を学習する. 使用するデータセットは CIFAR-10 であり, 10 クラス分類タスクを解く NN を学習する. つまり, 一般的な NN の学習と同様であり, 分類誤差には交差エントロピーを用いる.

4.5 提案手法 1 と既存モデルの比較

表 2 に提案手法 1 により構築された NN の学習結果と既存モデルの比較を示す. 既存手法のモデルを既存モデルとし, S1-L100 より上部に記載されているモデルのことを指す. DARTS, FBNet1, FBNet2 については, これらを用いてアーキテクチャが決定した NN を学習した. ResNet18, vgg11_bn, densenet121, DART について, 消費電力 (熱量) の計測のためには特殊な装置が必要であり, 計測が困難であった. [14] ではパラメータ数と消費電力には相関関係があることを示しているため, 消費電力 (熱量) の計測が困難なモデルについてはパラメータ数での比較とする. 推論速度は CIFAR-10 データセットのテストデータ, 10000 枚の画像分類に要した時間を示している.

最も正解率が高い, つまり精度が高いモデルは DARTS により構築されたモデルである. パラメータ数は, MobileNetV2 と大差がなく, IoT デバイス等の小型端末にも搭載が可能であると考えられるが, 推論速度が遅すぎてしまう. ResNet18, vgg11_bn, densenet121 については, 軽量なモデルと言われる, MobileNetV2 と比較すると推論速度が高速であり, 計算資源に限りのある IoT デバイスでも処理が可能であると考えられる. 一方で, パラメータ数が大きいため消費電力が非常に大きいと考えられる. 従って, バッテリーの利用が想定される IoT デバイスで動作させることは困難であると考えられる. FBNet の従来の損失関数と事実上, 同様である FBNet1, FBNet2 では精度が高い状態で, MobileNetV2 等の既存モデルよりも推論速度が高速で, パラメータ数が少ない結果となった, 提案手法 1 により構築された S1-L100 は, FBNet1, FBNet2 と比較し, 正解率が約 2%低下しているが, 推論速度を 1/2 未満, パラメータ数も大幅に減少させることができている. その他の提案手法 1 により構築された NN モデルは目標の正解率が低いほど, 正解率が低くなり, 推論速度が高速で低消費電力なモデルを構築できている. これらの結果は概ね期待通りである. 一方で, S1-L50 においては, S1-L80, S1-L65 よりも推論速度が遅くなった. しかし, 消費電力とパラメータ数は小さい結果となっている. 推論速度が高速であれば, 期待通りであったが, 損失関数では精度, 推論速度, 消費電力の 3 つを考慮しているため, このような結果が得られても不自然ではない結果である.

表 2 提案手法 1 の学習結果と既存モデルの比較

モデル名	正解率 (検証データ)	推論速度	消費電力 (熱量 (J))	パラメータ数
MobileNetV2	94.2%	40.1(s)	21.1(J)	2.2M
ResNet18	93.3%	8.92(s)	-	11.2M
vgg11_bn	92.1%	16.6(s)	-	129M
densenet121	94.1%	26.6(s)	-	7.00M
DARTS	**97.4%**	266(s)	-	3.3M
FBNet1	91.8%	16.3(s)	11.2(J)	0.66M
FBNet2	91.7%	17.2(s)	9.55(J)	0.49M
S1-L100	90.0%	6.35(s)	3.46(J)	0.32M
S1-L80	88.3%	4.48(s)	2.31(J)	0.30M
S1-L65	86.5%	4.18(s)	1.67(J)	0.23M
S1-L50	85.6%	4.83(s)	1.07(J)	0.21M
S1-L35	70.0%	**1.03(s)**	**0.450(J)**	**0.19M**

4.6 提案手法 2 と既存モデルの比較

表 3 に提案手法 2 により構築された NN の学習結果と既存モデルの比較を示す. 既存モデルは, 提案手法 1 の表 2 での FBNet1, FBNet2 を除き, FBNet3 を追加した. 先行研究 [13] で定義された損失関数を用いて構築された FBNet3 は, 既存モデルと比較すると精度は劣るが, 推論速度が高速で低消費電力, かつパラメータ数が小さい結果となった. また, 提案手法 2 により構築された S2-Normal, S2-L80 は FBNet3 と NN の構造が全く同じであり, 精度, 推論速度, 消費電力, パラメータ数も同じ結果となった. その他のモデルについては, 目標の正解率が低いほど, 正解率が低くなり, 推論速度が高速で低消費電力なモデルを構築できている. これらの結果は概ね期待通りである.

表 3 提案手法 2 の学習結果と既存モデルの比較

モデル	正解率 (検証データ)	推論速度	消費電力 (熱量 (J))	パラメータ数
MobileNetV2	94.2%	40.1(s)	21.1(J)	2.2M
ResNet18	93.3%	8.92(s)	-	11.2M
vgg11_bn	92.1%	16.6(s)	-	129M
densenet121	94.1%	26.6(s)	-	7.00M
DARTS	**97.4%**	266(s)	-	3.3M
FBNet3	87.5%	5.20(s)	1.62(J)	0.23M
S2-Normal	87.4%	5.44(s)	1.62(J)	0.23M
S2-L80	87.3%	5.21(s)	1.62(J)	0.23M
S2-L65	85.2%	4.63(s)	1.07(J)	0.21M
S2-L50	77.0%	4.50(s)	1.04(J)	0.20M
S2-L35	69.3%	**1.02(s)**	**0.450(J)**	**0.19M**

5. 考察

5.1 提案手法 1 の考察

提案手法 1 によって構築された NN は S1-L100 以外の全てにおいて目標の正解率を超え, 推論速度が高速で低消費電力なモデルとなった. 目標の正解率が 100%の S1-L100

については既存モデルでも, 達成が困難なため, 当然の結果であるが, 精度は高い結果である. 提案手法 1 の損失関数では, 精度を示す, 分類誤差に推論速度と消費電力の損失の和の値との積をとるため, 3 つに依存関係が生じる. 特に分類誤差は 10 クラス分類であるため, 減少させることが容易であることから, 分類誤差が提案手法 1 に強い影響を及ぼすと考えられる. 従って, 目標の正解率を大幅に超える結果になったと考えられる. 目標の正解率が低くなるほど, 分類誤差の値は学習時の早い段階で最小値になるよう操作される. 分類誤差を, これ以上, 減少させようがないと判断されることにより, 推論速度と消費電力の損失を減少させ, 結果的に推論速度が高速で低消費電力な NN を構築できたと考えられる. また, 分類誤差の算出に用いる交差エントロピーの値が 0 でなくても, Softmax 関数を用いて画像を分類するため, 分類タスクに正解してしまう可能性がある. 更に, 10 クラス分類であるため, 比較的, 容易なタスクである. このような要因と, 精度に依存する損失関数の特性から分類誤差の値を 0 に操作しても目標の正解率を大幅に超えると考えられる.

5.2 提案手法 2 の考察

提案手法 2 の損失関数は, 分類誤差と推論速度と消費電力の損失の和をとるため, 提案手法 1 とは異なり 3 つの依存関係が緩和される. 従って, 精度, 推論速度, 消費電力の損失の依存関係が薄く, 全てのバランスを考慮しようとするため, 分類誤差の値を操作したとしても, 全体の損失において, モデル間に差異が生じないと考えられる. このような理由から, FBNet3, S2-Normal, S2-L80 においては 構造が全く同じであり, 精度, 推論速度, 消費電力, パラメータ数が同じになったと考えられる. 一方で, S2-L65, S2-L50, S2-L35 については, 概ね期待通りの結果である. これは, 目標の正解率を大きく下げることで, 学習時の早い段階で分類誤差の値を最小値の 0 に操作するため, 全体の損失として推論速度と消費電力の損失を減少させることが, 全体の損失の減少に繋がると判断されたためであると考えられる. 目標の正解率を大きく超えた理由は, 提案手法 1 と同様であると考える.

6. おわりに

本研究では, FBNet の損失関数を改良し, 2 つの手法を提案した. 提案手法 1 は, 目標の正解率に応じて, 精度を重要視した NN のアーキテクチャ探索を実現した. 提案手法 2 では, 目標の正解率に応じて精度, 推論速度, 消費電力の 3 つのバランスを考慮した NN のアーキテクチャ探索を実現した. 2 つの提案手法では目標の正解率を大きく超え, 様々な用途に応じた NN モデルを探索により構築することが可能である.

今後の展望として, 目標の推論速度や消費電力を設定した探索や, セマンティックセグメンテーションや物体認識を解くことのできる NN を探索する手法への応用の実現を目指す.

謝辞 本研究の一部は, 日本電信電話株式会社 NTT ソフトウェアイノベーションセンタに支援頂いた. ここに深く感謝の意を表します.

参考文献

[1] 総務省, 令和 2 年版 情報通信白書 | IoT デバイスの急速な普及, https://www.soumu.go.jp/johotsusintokei/whitepaper/ja/r02/pdf/n1400000.pdf
[2] 総務省, 平成 30 年版 情報通信白書 | PDF 版, https://www.soumu.go.jp/johotsusintokei/whitepaper/ja/h30/pdf/index.html
[3] 日経エレクトロニクス, 日経 BP 社 (2020 年 9 月号)
[4] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In CVPR, 2019.
[5] H. Liu, K. Simonyan, and Y. Yang. DARTS: Differentiable architecture search. ICLR, 2019.
[6] Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. MnasNet: Platform-aware neural architecture search for mobile. CVPR, 2019.
[7] Stamoulis, D., Ding, R., Wang, D., Lymberopoulos, D., Priyantha, B., Liu, J., Marculescu, D.: Single-Path NAS: Designing Hardware-Efficient ConvNets in less than 4 Hours. In: European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (2019)
[8] B. Wu, Y. Wang, P. Zhang, Y. Tian, P. Vajda, and K. Keutzer. Mixed precision quantization of convnets via differentiable neural architecture search. arXiv preprint arXiv:1812.00090, 2018.
[9] Yanqi Zhou, Siavash Ebrahimi, Sercan O Arık, Haonan Yu, Hairong Liu, and Greg Diamos. Resource-efficient neural architect. arXiv preprint arXiv:1806.07912, 2018.
[10] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once for all: Train one network and specialize it for efficient deployment. In International Conference on Learning Representations, 2020.
[11] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. CVPR, 2018.
[12] Dai, X., et al.: Chamnet: towards efficient network design through platform-aware model adaptation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 11398–11407 (2019)
[13] Sai Vineeth, Kalluru Srinivas, Harideep Nair, Vinay Vidyasagar, Hardware Aware Neural Network Architectures using FbNet. https://arxiv.org/abs/1906.07214, 2019.
[14] Q. Qin, J. Ren, J. Yu, H. Wang, L. Gao, J. Zheng, Y. Feng, J. Fang, and Z. Wang, “To compress, or not to compress: Characterizing deep learning model compression for embedded inference,” in Proc. ISPA/IUCC/BDCloud/SocialCom/SustainCom, 2018, pp. 729–736.