画像変換ネットワークによる連続学習

松本　晨人†　　柳井　啓司†

† 電気通信大学情報理工学部総合情報学科　〒 182–8585 東京都調布市調布ヶ丘 1–5–1
E-mail: †{matsumo-a,yanai}@mm.inf.uec.ac.jp

あらまし　これまで Deep Convolutinal Neural Network (CNN) での連続学習 (continual learning) の手法が提案されてきたが, それらの多くは画像分類タスクでのものであった. そこでここでは画像変換タスクでの連続学習の研究を行う. 本論文は, 画像を生成する Encoder-Decoder CNN へモデルの重みを選択するマスクを用いた連続学習の手法である Piggyback [1] の適用, さらにそれに加えて残差関数を学習する Resblock [2] の追加を行うことで, 一つの CNN で複数の異なる画像変換タスクの連続学習の実現とその性能が個別に学習したモデルに匹敵するのを示すことを目的とする. 領域分割, 濃淡画像着色, スタイル変換による実験で Piggyback と Resblock の追加はそれぞれ異なる画像変換タスクの連続学習において個別に学習した CNN と同等の性能を発揮した. さらに最終的には Piggyback を Resblcok に適用することで小さいオーバーヘッドで高性能な連続学習を目指す.

キーワード　連続学習, 破壊的忘却, 画像変換

1 はじめに

人間や動物は生涯を通して継続的に知識を習得し, 微調整することができる. この能力は脳の豊富な神経認知機能によって実現している. その結果, 人間や動物は長期にわたって多くの経験を通して新しい知識を学習することが可能であり, 昔の知識を忘れることもない. このように以前学習した知識を保持したまま新たな知識に適応する学習のことを連続学習という. 現実の世界で動作させる CNN は連続的な情報や逐次的にタスクが与えられるため, そのような場面では長期にわたって知識を学習し, 微調整を繰り返す連続学習が重要である. また, 連続学習では以前学習したタスクを忘れないため, 大量に与えられる様々なタスクをこなすことが必要な汎用人工知能の達成に連続学習が貢献すると考えられる. 加えて複数のタスクを一つの CNN で実行できるため, 実用的な観点から学習済みモデルサイズを抑えることができ, CNN を利用したアプリケーションをスマートフォンやデバイスへ実装にも貢献すると考えられる.

汎用人工知能にも関係している連続学習であるが, 機械学習の特定の状況にのみ特化して学習する性質のために, CNN における連続学習は未解決問題となっている. 人間が連続学習を行う場合, 昔に学習したタスクを忘れることなく新しいタスクも学習できる. 一方, CNN の知識は学習に使用したデータセットに依存しており, データ分布の変化に適応するためにはデータセット全体に対して CNN のパラメータの再学習が必要となる. 時間の経過と共に与えられる新しいタスクについて学習していくにつれて, 昔のタスクの精度は低下していく. このように, CNN で連続学習を行うと新しいタスクの学習中に昔のタスクの学習結果を忘れてしまう致命的忘却 (catastrophic forgetting) が起こる. 致命的忘却は CNN に対してパフォーマンスの低下や新しい知識が古い知識を上書きすると言った現象を引き起こす. このため CNN で人間や動物のように連続学習を行うことが難しくなっている. 致命的忘却を回避する手法として疑似リハーサル [3] や蒸留 [4], EWC [5], ネットワークの拡張 [6], 剪定 [7], 重みの選択 [1] などが提案されている. ただし, これらの手法の多くは画像のクラス分類や強化学習, 画像生成に関するものであり, 画像変換タスクでは連続学習の研究はほとんど行われていない. そこで画像を生成する Encoder-Decoder CNN を用いた領域分割やスタイル変換, 着色の画像変換タスクでの連続学習の手法を提案する.

本論文は, 入力も出力も画像である画像変換タスクに対する連続学習を扱う. 具体的には, (1) Encoder-Decoder CNN モデルの重みを選択するマスクを用いた連続学習の手法である Piggyback [1] を適用する, または, (2) 入力した画像をそのまま出力する Auto Encoder に Resblock をタスクごとに追加する, ことで, 一つの CNN モデルで複数の異なる画像変換タスクの連続学習の実現とその性能が個別に学習したモデルに匹敵するのを示すことを目的とする. さらに将来的には (3) Piggyback を Resblcok に適用することで小さいオーバーヘッドで高性能な連続学習を目指す.

2 関連研究

学習済みの CNN を用いた新しいタスクの学習で一般的な方法は fine-tuning である. fine-tuning とは学習済み CNN の重みの一部を新しいタスク用に再学習させる手法のことである. 再学習で CNN の重みパラメータの値が変わるために昔のタスクの精度が低下する致命的忘却が起きてしまう. この致命的忘却を回避する手法として疑似リハーサル [3], 蒸留 [4], 最適化 [5], ネットワークの拡張 [6], 剪定 [7], 重みの選択 [1] などが挙げられる. ここでは特に最適化と重みの選択の手法について説明する.

2.1 EWC

EWC [5] とは Elastic Weight Consolidation の略で, EWC は特定の重みの学習を以前のタスクにとっての重要度に応じて値が変化しないようにするものである. これにより前のタスクでの学習結果の破壊が小さくなるため, 新しいタスクの学習に

よる前タスクの性能低下が防がれる．重みの重要度は CNN のパラメータフィッシャー情報行列で決定し，学習の速度は重みの重要度に比例したロスを学習損失関数にロスを与えることで調整する．フィッシャー情報行列で重要なパラメータを選択することで，以前学習したタスクにとって重要な重みを保ったまま新しいタスクの学習を可能にした．ただし，EWC には前のタスクと新しいタスクの内容が大きく離れている場合や多くのタスクを追加しようとした場合，重要度の高い重みの値が大きく変わってしまうため前タスクの性能が低下してしまう問題がある．

2.2 Piggyback

一方，Piggyback [1] では EWC の問題である前タスクの性能低下を解決し，高い精度で多くのタスクをベースの CNN で学習することに成功した．Piggyback ではまず初めに比較的大規模なデータセットで汎用的なベース CNN を学習し，その後新たなタスクを追加で学習する時はタスク毎に重要度の高い重みを選択する 0 と 1 の値を持つバイナリマスクを学習する．バイナリマスクのみを学習しするためベース CNN の重みの値は変化せず，前のタスクの性能が低下することはない．Piggyback の新しい部分は，EWC などでは学習のやり方を最適化しているのに対して，ベースの CNN は固定して，そこから重要な重みを選択して利用することである．本論文では一つ目の手法として，Piggyback を Encoder-Decoder CNN に適用したものを提案する．

2.3 Residual Connection

画像変換タスクでは Resblock 付き Encoder-Deocder CNN が多く使われている．Resblock とは He らが提案した Residual Network [2] で使用されたもので，層への入力を用いてその層の出力との残差関数を学習するものである．Resblock 付き Encoder-Decoder CNN は Johnson らの Fast Neural Style Transfer [8] で初めて使われ，その後 Isola, Zhu らの pix2pix [9] や CycleGAN [10] で使われた．Resblock 付き Encoder-Decoder CNN のアーキテクチャは Encoder-Resblocks-Decoder の順番で構成されており，Encoder 部分で入力画像の特徴量を抽出し，Resblock 部分で特徴量をタスクに沿って変換し，Decoder 部分で特徴量から画像を生成する (図 1)．Johnson らはこの論文 [8] で Residual Connection は恒等関数の学習を容易にすることから，Resblock 付き Encoder-Decoder は入力画像の構図を維持したまま別の画像を出力する画像変換タスクにとって非常に有効であると主張している．Johnson ら [8] はこのアーキテクチャを用いて様々なタイプのスタイル変換や超高解像を行った．また pix2pix [9] や CycleGAN [10] ではスタイル変換の他に領域分割，線画着色，ドメイン変換などを行った．このように Resblock 付き Encoder-Decoder CNN が多様な画像変換タスクに対応できるのは Resblock がタスク固有の変換能力を獲得しているためと考え，本論文では二つ目の手法として，画像を入力するとそれと同じ出力を得る Auto-Encoder に Resblock を組み合わせた Resblock 付き Encoder-Decoder CNN を提案する．

3 手　　法

本論文では一つの CNN で複数の異なる画像変換タスクの連続学習を行う．実験は Piggyback を利用するものと Resblock を利用するものの二種類の手法で行った．

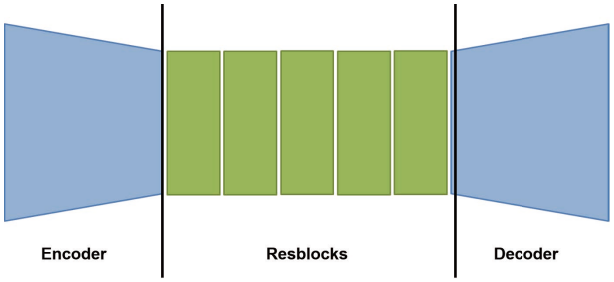

図 1　Resblock 付き Encoder-Decoder の構造

3.1 Piggyback

一つ目の連続学習の手法は Piggyback [1] を Encoder-Decoder CNN に適用するものである．Piggyback は一番初めに学習したベース CNN のパラメータを固定し，タスクを追加するごとにタスク固有のバイナリーマスクを学習する手法である．Piggyback の論文では新しいマスクを加える場合，タスクごとの最終出力レイヤーを準備している．これに従い，本論文では画像変換タスクでの連続学習に関して Encoder-Decoder CNN，タスクごとのバイナリーマスクと最終出力レイヤーを準備した．これにより，タスクを追加するごとに Encoder-Decoder CNN のパラメータと同じ数のバイナリーマスクと各タスクの最終出力レイヤーのオーバヘッドが生じる．

Piggyback の学習手順を図 2 に示す．

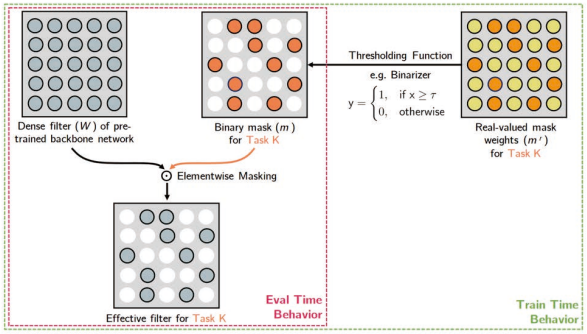

図 2　Piggyback でのマスクの学習 ([1] から引用)

Piggyback ではまず始めに画像変換タスクの一つについて CNN を学習し，このベース CNN を用いて追加タスクのマスクを学習する．マスクの学習は以下の手順で行う．

（1） ベース CNN の重みを固定 (図 2 の W)
（2） 実数マスクを作成 (図 2 の m^r)
（3） 実数マスクを閾値で 2 値化してバイナリーマスクを作成 (図 2 の m)
（4） ベース CNN の重みにバイナリーマスクを要素ごとにかけて重みを選択 (図 2 の Effective filter)
（5） (4) を用いて順伝播，逆伝播を行い勾配を計算
（6） 実数マスクの更新
（7） (3) ～ (6) を繰り返す

実数マスクとはベース CNN の重みの数と同じ数の実数行列のことである．実験では実数マスクの初期化は Piggyback [1] の論文と同様に全てのパラメータを 1e-2 で初期化した．また，

バイナリーマスクを作成するときに使用する閾値も [1] と同様に 5e-3 として実験を行った.

本実験では, 追加したタスクでも高い精度を発揮するために高機能なベース CNN が必要だと考えた. そこで, 一番初めの画像変換タスクを約 20 万枚の画像が 80 種類のオブジェクトでアノテーションされている MS COCO の領域分割にした. 二番目以降のタスクでは, ベース CNN から新しく追加するタスクに有効な重みを選択するバイナリーマスクを学習する. さらに本実験では, CNN のアーキテクチャとして Encoder と Decoder の間にスキップコネクションを持ち, 様々な画像変換タスクに用いられる U-Net [11] を使用した. ただし, 出力層以外の活性化関数 ReLU の前の Batch Normalization の代わりに学習の収束を早める効果のある Instance Normalization を使用した.

3.2 Resblock 付き Encoder-Decoder CNN

二つ目の連続学習の手法は Resblock 付き Encoder-Decoder CNN の中間部分にある Resblock をタスクごとに入れ替えるものである. Encoder と Decoder の間に Resblock を挿入して学習することで, Resblock が学習するタスクに固有な特徴量の操作を獲得する. タスクごとに個別の Resblock を学習するため catastrophic forgetting は当然起こらない. よって, 本手法では事前に学習した入力画像と同じ画像を出力する Auto-Encoder とタスクごとの Resblock を準備した. これにより, タスクを追加で学習するごとに入れ替える Resblock の分のオーバヘッドが生じる. Piggyback ではタスク毎に最終レイヤーを交換していたがこちらでは行わなず, こちらの手法のオーバーヘッドは Resblock の分のみである. また, Resblock は Auto Encoder の Encoder 部分が抽出した特徴量に対して固有の変換を行い Decoder 部分が入力された特徴量から画像を生成する. このため, 別々のタスクで学習した Resblock を連結することで二つの変換を連続できる可能性があると考えられる. これに関しては考察にまとめた.

学習は以下の手順で行う. ただし, さらに新たなタスクを追加で学習する場合は, (3) で学習した Resblock を外し再度 (3) を行う.

（1） Auto-Encoder を学習

（2） Auto-Encoder のパラメータを固定する

（3） Encoder と Decoder の間に Resblock を入れ追加タスクを学習する

こちらの手法でも, 様々な画像変換タスクで画像を生成するために高機能な Auto-Encoder が重要であると考えた. そこで, Auto-Encoder 学習に MS COCO の一般画像, MS COCO の領域分割の結果画像, Wiki Art の絵画画像のデータセットを利用し, 損失関数は Adversarial Loss を用いた. 本実験で新たなタスクを学習する場合は, 上記のように学習した Auto-Encoder の Encoder 部分と Decoder 部分の間に Resblock を追加したものを用いた. Auto-Encoder と Resblock の構造は Zhu らの CycleGAN [10] を参考にした. 構造の詳細を図 3 に示す. 図 3 の上部が Resblock 付き Auto-Encdeor , 下部が Resblock の構造を示している. こちらも Piggyback と同様に Instance Normalization を使用した. Encoder 部分と Decoder 部分に追加する Resblock の数も CycleGan を参考に 6 個と 9 個で実験を行った.

最終的には 3.1 の Piggyback と 3.2 の Resblock を組み合わせた手法を目指す.

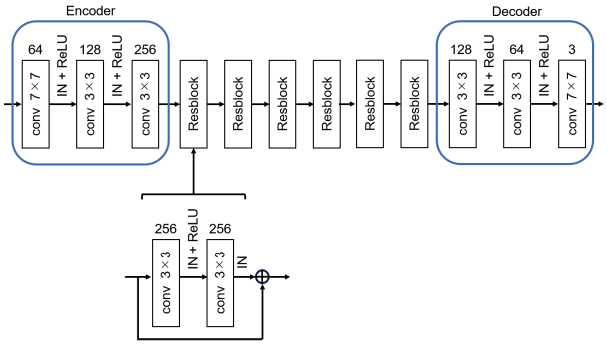

図 3 Auto-Encoder と Resblock 構造

4 実　　験

本実験では異なるものを含む四種類の画像変換タスクの連続学習を行い, 提案手法の性能を評価した. タスクは, 領域分割, 濃淡画像着色, スタイル変換 [8] である. 各タスクの内容を表 1 に示す. 学習は表 1 に示すタスク 1, 2, 3, 4, 5 を逐次実行した. ただし, ベースの一つである “fine-tune” のタスク 5 はタスク 3 の後, fine-tuning したものである.

表 1 各タスクの内容

タスク番号	データセット	内容
タスク 1	MS COCO	領域分割
タスク 2	Pascal VOC	領域分割
タスク 3	MS COCO (グレイスケールに変換)	濃淡画像着色
タスク 4	MS COCO	スタイル変換 (Gogh)
タスク 5	MS COCO	スタイル変換 (Munk)

MS COCO と Pacal VOC はどちらも人や乗り物, 動物といった一般画像のデータセットである. MS COCO は 80 種類のカテゴリーを含む約 33 万枚の画像で構成される大規模なデータセットである. Pascal VOC は MS COCO の 80 種類のカテゴリー内の 20 種類を含む約 1 万枚の画像で構成されるデータセットである. タスク 1 とタスク 2 で同じ領域分割を行う理由は, 同じタスクを異なるデータセットやカテゴリーで連続学習が可能かを確認するためである. タスク 3 以降は領域分割とは種類の異なるタスクで実験を行い, 異なるタスク間での連続学習が可能かを検証する. 様々な種類のタスクに対応させるためにタスク 1 のデータセットは MS COCO のような大規模なものを使用した.

本実験では比較のために三種類のベースラインを用意した. ベースラインと Piggyback と Resblcok の概略を図 4 に示す. 三種類のベースラインはそれぞれ, タスクごとに個別にスクラッチから学習する “scratch”, 前のタスクからモデル全体を fine-tuning する “fine-tune”, タスク 1 で学習した Encoder とタスクごとに学習した Decoder を組み合わせた “decoder” である. また提案手法は Piggyback を利用したものを “Piggyback”, Resblock 付き Encoder-Decoder CNN を “Resblock” とする.

学習時の損失関数は, タスク 1, 2 は Cross Entropy Loss, タ

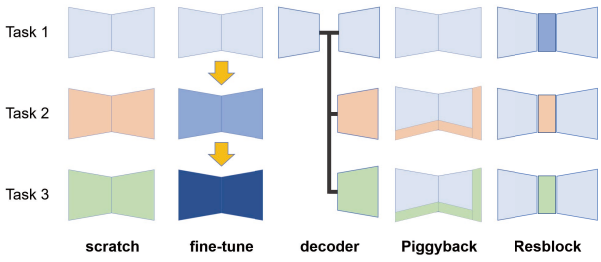

図 4　各手法の概略

表 2　連続学習の実験結果 (各タスクの内容は表 1 を参照)

	scratch	fine-tune	decoder	Piggyback	Resblcok(6)	Resblcok(9)
タスク 1 (mIoU(%))			**21.47**		0.57	0.71
タスク 2 (mIoU(%))	58.59	**64.87**	61.63	61.45	4.26	9.54
タスク 3 (MSE, SSIM)	244.000 0.9138	**237.92** 0.9148	241.66 0.9121	242.49 0.9058	532.83 0.9281	527.13 **0.9286**
タスク 4 (SSIM, total loss(epoch))	**0.3678** 413833 (200)	0.3555 **405893** (200)	0.3595 473723 (200)	0.3501 528587 (100)	0.3467 460268 (200)	0.3524 442211(200)
タスク 5 (total loss(epoch))	**447480** (6)	490490 (6)	544348 (6)	521476 (6)	520221 (6)	494600 (6)
タスク 1 after タスク 2	-	0.70	21.47	21.47	0.57	0.71
タスク 2 after タスク 3	-	1.87	61.63	61.45	4.26	9.54
タスク 3 after タスク 4 (MSE, SSIM)	- -	870.18 0.5321	241.66 0.9121	242.49 0.9058	532.83 0.9281	527.13 0.9286
モデルサイズ (MB)	282.0 (56.4*5)	282.0 (56.4*5)	138.4 (56.4+20.5*4)	**63.6** (56.4+1.8*4)	683.7 (8.7+135.0*5)	1021.2 (8.7+202.5*5)

スク 3 は L2, タスク 4, 5 は Johnson ら [8] の Content Loss と Style Loss を足し合わせたものを使用した. ただし, “Resblock” ではタスク 2 を領域分割した結果を画像として出力する画像生成タスクと捉え, クロスエントロピーロス以外に L1, L2, Adversarial Loss でも学習を行った. 入力は, タスク 1, 2, 4, 5 は RGB 画像, タスク 3 はグレイスケール画像を使用し, 出力は, タスク 1, 2 はそれぞれ 81 チャネル, 21 チャネルのセグメンテーションマップ, タスク 3 は 2 チャネルの YCbCr 表現の CbCr 成分, タスク 4, 5 は 3 チャネルの RGB 画像とした. 評価には, それぞれテストデータセットを利用し, タスク 1, 2 は mean Intersection over Union (mIoU), タスク 3 は Mean Square Error (MSE) と Structural Similarity (SSIM), タスク 4 は Gatys のスタイル変換の論文 [12] に記載されている図 6.A との SSIM と Content Loss と Style Loss を足し合わせた total loss, タスク 5 はタスク 4 と同様の total loss で性能を評価した.

表 2 に各タスクの評価と新たにタスクを学習した後で元のタスクを評価した結果を示す. また各タスクで画像を生成した例を図 5, 図 6, 図 7, 図 8, に示す. “Resblock” の欄の括弧の中の数は CNN 内に含まれる resblock の数を表している. 表 2 の “Resblock” は Adversarial Loss で学習した結果を表示している. その他の損失関数での学習結果については後の 5.2.1 にまとめた. 表 2 から一つ目の提案手法である “Piggyback” は一番精度が高いベースラインとほぼ同等の性能を発揮していることがわかる. 二つ目の提案手法である “Resblock” は領域分割のタスク 1, 2 の精度は低いが, その他の画像変換タスクではかなり高い性能を発揮しており, Resblock の数が多いほど CNN の性能も高くなっていることもわかる. また, 新たにタスクを学習した後に昔のタスクの精度を評価した結果から, “scratch” と “fine-tune” では catastrophic forgetting が起きているが, “decoder” と “Piggyback”, “Resblock” ではそれが起きていないことがわかる. さらに “decoder” と “Piggyback” のモデルサイズを比較すると, “Piggyback” は “decoder” の半分以下となった. これらのことから, 異なる複数の画像変換タスクの連続学習において, “Piggyback” は少ないオーバヘッドで “fine-tuning” とほぼ同等の性能を発揮しているといえる. また, “Resblock” はピクセル単位の変換は苦手だが入力画像の形状を維持したまま異なる風貌の画像に変換するのが得意なモデルで, Resblock の数が 6 個よりも 9 個の方が性能は上がるがその分オーバーヘッドも増加するモデルであるといえる.

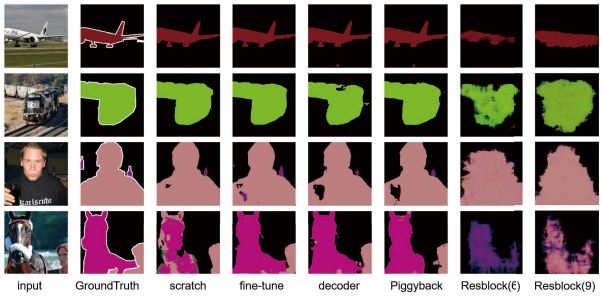

図 5　タスク 2 の結果 (Pascal VOC での領域分割)

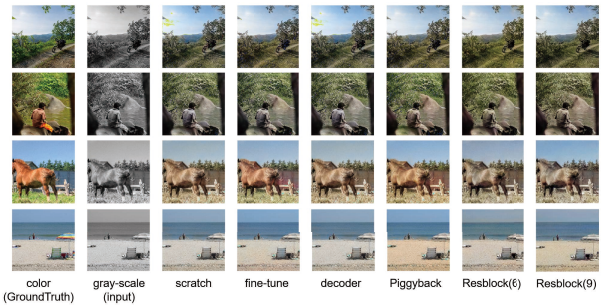

図 6　タスク 3 の結果 (濃淡画像着色)

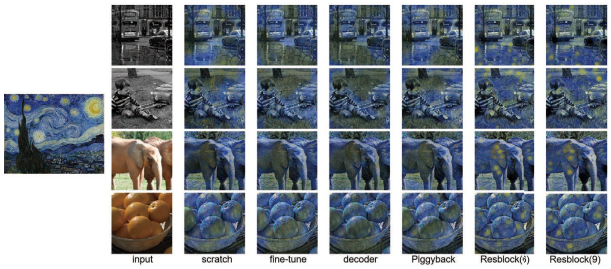

図 7　タスク 4 の結果 (スタイル変換)

5　考　　察

5.1　Piggyback のバイナリーマスク

ここでは Mallya ら [1] と同様に Piggyback で学習したバイナリーマスクの分析を行う. 学習したバイナリーマスクの値が 0 になった数を調べた. これによって各タスクを行うときにベース CNN に対してどの程度の変更が必要であったか, または MS COCO の領域分割タスクで初期化したベース CNN が各タスクに対してどの程度有効であったかを測定した. U-Net (図 9) の各レイヤー毎に学習したバイナリーマスクの 0 の割合をタスク毎に図 10 から図 13 に示す. また, Mallya ら [1] の ImageNet

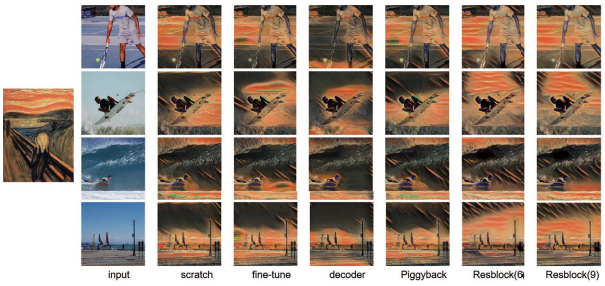

図 8　タスク 5 の結果 (スタイル変換)

の分類タスクで初期化した VGG16 で Wiki Art の分類タスクを Piggyback で学習した結果を図 14 に示す．図 10 から図 13 のグラフの横軸は U-Net の各レイヤーを表しており，グラフの conv と up-conv は U-Net の図 9 の conv, up-conv と対応している．VGG16 による分類タスクでのバイナリーマスクの 0 の割合は低レイヤーでは低く，高いレイヤーになるほど高くなる傾向があった．このため分類タスク同士での Piggyback では，低レイヤーではベース CNN の再使用率が高く，高レイヤーになるほどデータセットに固有の変換が行われていると考えられる．一方 U-Net の場合，タスク 2 の Encoder 部分では上記の特徴が僅かにみられるが，他のタスクの Encoder 部分や全タスクの Decoder 部分は上記のような特徴はみられず，どのレイヤーでも 0 の割合が 50%から 60%程度となった．また，U-Net のタスク 2 の Encoder 部分の一番初めのレイヤーの 0 の割合は VGG16 の先頭のレイヤーと比べて約 40 ポイント高くなっており，全体的にベース CNN の重みの再使用率が低くなった．さらに，タスク 3 以降の領域分割以外のタスクでは低レイヤーの 0 の割合も大きくなった．VGG16 の分類タスク同士での Piggyback と比べて全レイヤーで 0 の割合が高くなったのは下記の三つに原因があると考えられる．一つ目の理由は実験で利用した U-Net のレイヤー数が VGG16 よりも多いためである．CNN 全体のパラメータ数が増加することで一つのレイヤーが持つ重要なパラメータ数が少なくなり，一つのレイヤーで取得する特徴の種類が減少したのではないかと考えた．二つ目の理由はベース CNN を学習したタスクと新しく学習したタスクの種類が異なるためである．Mallya ら [1] の実験ではタスク毎にデータセットを変更して分類タスクのみの連続学習を行った．一方，本論文の実験ではベースの CNN を領域分割タスクで学習し，追加のタスクとして領域分割，濃淡画像着色，スタイル変換のベース CNN で学習したものとは異なるタスクの連続学習も行った．タスクが変わることで重要な重みも変化した，もしくはマスクを 0 にすることで元のレイヤーとは異なる変換を実現したと考えられる．三つ目の理由は CNN において重要であるパラメータがレイヤー全体の内 50%程度であるかもしれないというためである．Mallya らの Packnet [7] では事前に学習した CNN のパラメータの 50%を剪定した場合でも，剪定する前の CNN の精度から 1%未満の性能劣化で済んでいる．さらに実験結果の中で興味深いのは conv5_1 の 0 の割合が全てのタスクで低いということだ．このことから，異なるタスクであっても共通の変換が行われている可能性があると考えられる．

次に各マスクの類似度について分析を行った．全てのマスク

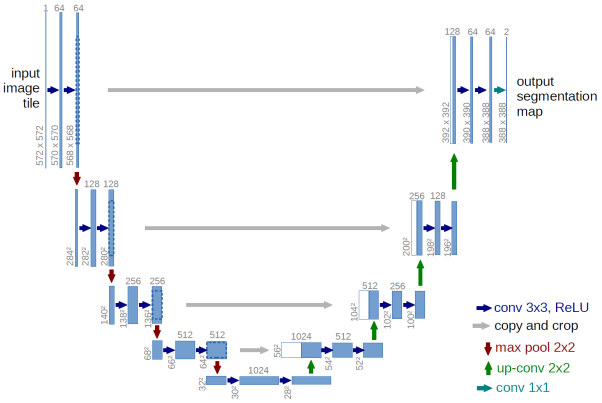

図 9　U-Net のアーキテクチャ ([11] から引用)

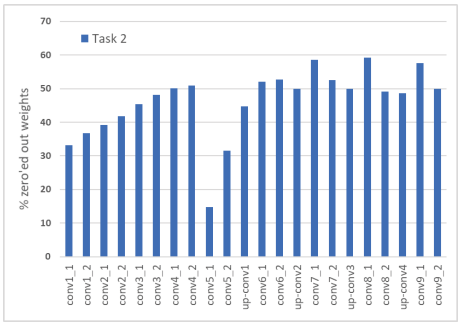

図 10　タスク 2(Pascal VOC 領域分割) のバイナリーマスクの 0 の割合

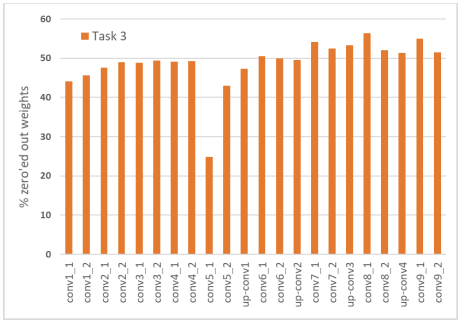

図 11　タスク 3(濃淡画像着色) のバイナリーマスクの 0 の割合

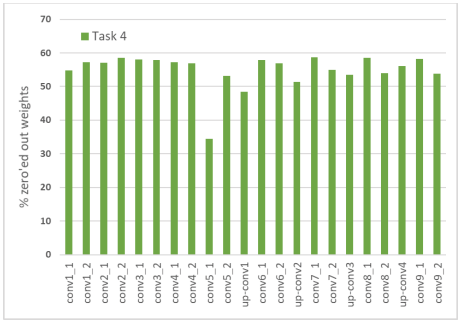

図 12　タスク 4(スタイル変換) のバイナリーマスクの 0 の割合

はバイナリマスクであるので，マスク同士で排他的論理和をとることで類似度を求めた．各タスクのバイナリーマスク同士の類似度行列を表 3 に示す．表 3 から領域分割タスクであるタスク 1 と 2 とスタイル変換タスクであるタスク 4 と 5 の組の類似度は高くなり，領域分割とスタイル変換の組であるタスク 1 と

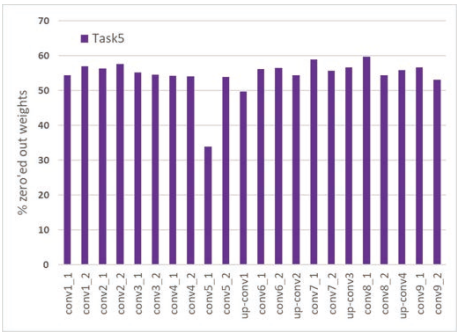

図 13　タスク 5(スタイル変換) のバイナリーマスクの 0 の割合

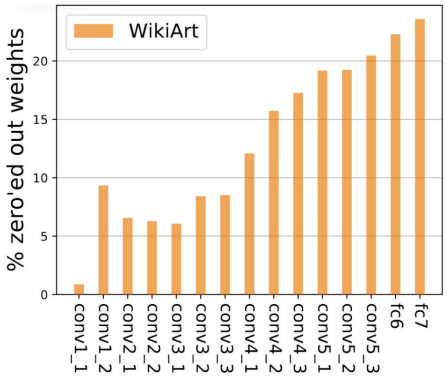

図 14　画像分類タスク [1] でのバイナリーマスクの 0 の割合

4,5 の組の類似度は低くなることが分かった. このことから, 各タスクで重要な重みは異なっているおり, 似ているタスクの間ででは共通の重みが使われ, タスクの種類が異なる物同士では異なる重みが使われていると考えられる.

表 3　各タスクのバイナリーマスクの類似度行列

	タスク 1	タスク 2	タスク 3	タスク 4
タスク 2	0.5075	-	-	-
タスク 3	0.5042	0.5054	-	-
タスク 4	0.4326	0.5034	0.5020	-
タスク 5	0.4529	0.5029	0.5025	0.5210

5.2　Resblock の領域分割タスクにおける損失関数

第 4 章の実験結果である表 4.2 から, 提案手法である “Resblock” では領域分割のタスク 1,2 の精度が低いことがわかる. “Resblock” を用いて領域分割を行った生成画像の失敗例を図 15 に示す. 図 15 から “Resblock” による領域分割では検出した領域が大幅にずれている, 異なるクラスに分類してしまう, 領域を検出できないといった不具合が発生することがわかった. そこで, “Resblock” においてタスク 2 の領域分割タスクを様々な損失関数を用いて学習した. 損失関数は L1, L2, Adversarial Loss, Cross Entropy Loss を使用した. L1, L2, Adversarial Loss の学習は “Resblock” で出力した RGB 画像と RGB で表現したセグメンテーションマップを用いて行った. mIoU を求める場合, 第 4 章でも述べたように出力した RGB 画像をピクセルごとにクラスごとに設定したカラーマップと比較して最も近い色のクラスに変換したセグメンテーションラベルを用いた. Cross Entropy Loss でのみ “Piggyback” のように最終出力層を 21 チャネルの畳み込み層に付け替えて, 通常の領域分割と同様に学習を行った.

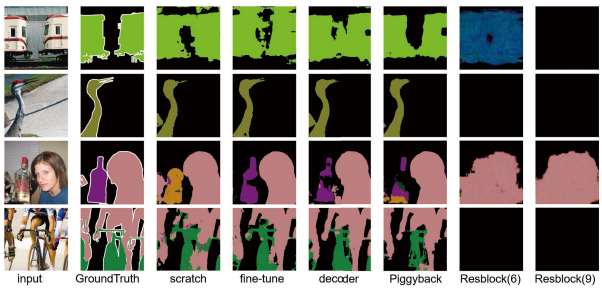

図 15　タスク 2(Pascal VOC 領域分割) の結果 (失敗例)

各損失関数の結果を表 4 に示す. また, その生成結果を図 16 に示す. Adversarial Loss の後ろの括弧は追加した Resblock の数を表す. 表 4 から Cross Entropy Loss が “Resblock” の中で最も高い 52.55%を達成した. しかし, 表 4.2 でタスク 2 の最高精度を出した “fine-tune” の 64.87%と比較すると, Cross Entropy Loss で学習した “Resblock” は 12.32 ポイント低い結果となった. Cross Entropy Loss の次に精度が高かったのは Adversarial Loss (9) であった. しかし, その精度は 9.54%で Cross Entropy Loss から 43.01 ポイント低くなった. このように損失関数の選択にかかわらず精度が良くならない原因としては “Resblock” は Auto Encoder を使用しているためだと考えられる. Auto Encoder は RGB 画像を出力する CNN であり, ピクセル単位のクラス分類を行う領域分割を行うのとは異なる変換を CNN は行っているため, Auto Encoder は領域分割に向かない CNN であるためと考えられる. さらに表 4 から Adversarial Loss の (6) と (9) を比較すると Resblock の数が多い (9) の方が 5.28 ポイント高いことが分かった. このことから Auto Encoder に追加される Resblock の数が 6 個よりも 3 個多い 9 個の方が高い表現力を持つことができると考えられる. 次に図 16 を見ると, 背景クラスは正確にできていることがわかる. これはほぼすべてのデータセットに背景のアノテーションがなされていたために, 学習の機会が多い背景を検出する能力を CNN が獲得したのではないかと考えられる. また, 領域分割の生成結果から精度は低いながらも “Resblock” による領域分割をできていると考えられる. このため, Auto Encoder の初期化の仕方を工夫する, Cross Entropy Loss のように各クラスの情報量を用いるような損失関数を考えることで, 最終出力層を取り換えることなく Cross Entropy Loss の性能に近づけることができるのではないかと考えられる.

表 4　“Resblock” を用いた領域分割での各損失関数の結果

	L1	L2	Adversarial Loss (6)	Adversarial Loss (9)	Cross Entropy Loss
タスク 2 (mIoU(%))	7.11	4.49	4.26	9.54	52.55

5.3　Resblock の連結

ここでは “Resblock” で学習した Resblock の分析を行う. “Resblock” では Auto Encoder 部分のパラメータは固定した状態で Resblock を Encoder と Decoder の間に追加し, タスク別に Resblock を学習した. そのため学習した Resblock は Auto Encoder が抽出した特徴量に対してタスク固有の変換を

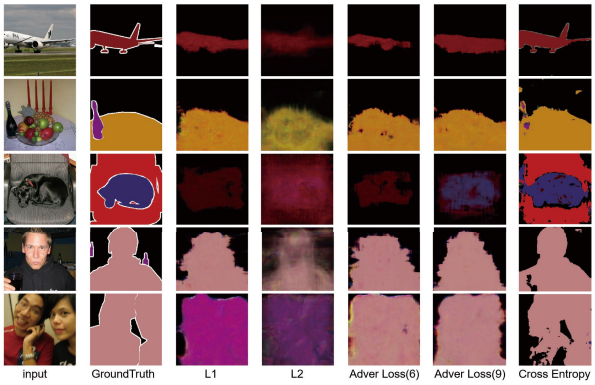

図 16　タスク 2(Pascal VOC 領域分割) の損失関数別の結果

獲得したはずである．そこで Resblock の性質を確認するために, 各タスクで学習した Resblock を二つ組み合わせて出力した画像の観察を行った．Resblock の組み合わせ方の概略を図 17 に示す．Resblock の組み合わせ方としては, ベースラインとして二つの “Resblock” を組み合わせて二段階で変換する “Two Stage”, 二つの Resblock を逐次的に連結した “Sequentially”, 二つの Resblock を交互に連結した “Alternately” で実験を行った．タスクの組み合わせは, 一つ目は濃淡画像着色のタスク 3 とし, もう一方はタスク 1,2,3,4 とした．ただし, “Two Stage” では二段階変換の中間部分で Decoder->Encoder の順に変換が行われるが, Auto Encoder の Encoder と Decoder の位置が逆になったものと考えられるので, “Two Stage” は中間部分の Decoder に入力される特徴量と Decoder->Encoder で得られる特徴量は等価変換であり, “Two Stage” の中間部分を省略したものが “Sequentially” と同一のものであると仮定できる．

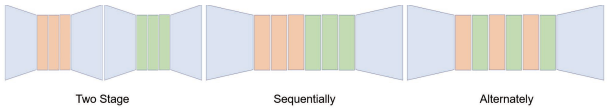

図 17　Resblock の連結の概要

Resblock の連結変換の結果を図 18 から図 21 に示す．全体の連結変換を通して興味深い点は, “Two Stage” と “Sequentially” の生成画像は完全に同一のものにならなかったことである．これは Decoder 部分が Auto Encoder の学習に使われていない画像の信号に反応できなかったためだと考えられる．Auto Encoder の学習時に使用するデータの種類を増やすことで改善される可能性がある．

濃淡画像着色と領域分割タスクの組み合わせである図 18 と図 19 の場合, “Two Stage” では領域分割ができており, “Sequentially” と “Alternately” とどちらも背景と物体の領域を表示するような出力となった．しかし, 領域の位置とクラスの推定がほとんどできていないことがわかる．また, “Sequentially” と “Alternately” の出力結果には領域分割のようなものの中に元画像の形が薄く見られる．これは領域分割の Resblock に濃淡画像着色の Resblock が組み合わさっているためと考えられる．

次は濃淡画像着色同士の組み合わせである図 20 の場合である．ここでは, ベースラインとして濃淡画像着色を一度行った “One Stage” の結果も追加した．図 20 から四つの全ての方法で濃淡画像を着色した出力画像を得た．“Sequentially” では黄色がかった画像, “Alternately” では薄い色合いの画像が出力された．同じ Resblock の部分が二重になった “Alternately” の濃淡画像着色が薄い着色になるのは興味深いと感じた．

最後に濃淡画像着色とスタイル変換の組み合わせである図 21 の場合, 三つの全ての方法でスタイル変換をした画像が得られた．しかし, “Sequentially” の生成結果は “Two Stage” と比べるとスタイルの特徴が崩れ, ノイズが乗るようになった．また, “Alternately” の生成結果もスタイルの特徴が “Sequentially” よりもさらに崩れた．

以上のことから, “Resblock” で学習した Resblock には Resblock の入力には Auto Encoder の Encoder 部分から得られる特徴量が適している, 特定のタスクの Resblock を一箇所に集めて変換するほうが綺麗に画像を生成できるといった性質があると考えられる．

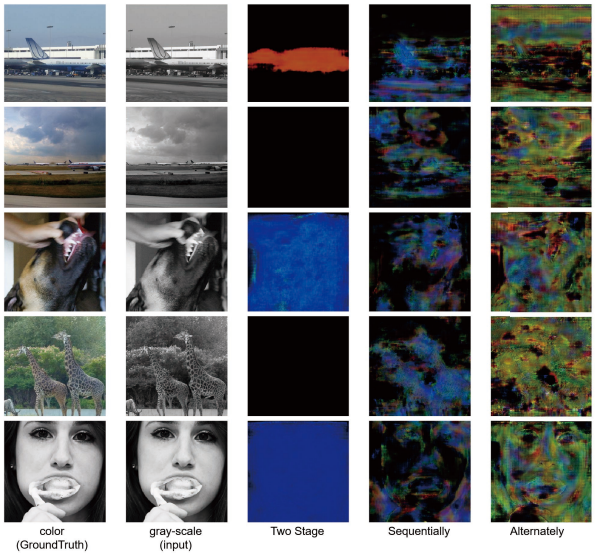

図 18　タスク 3(濃淡画像着色) とタスク 1(MS COCO 領域分割) の連結

6　ま　と　め

本論文では異なる複数の画像変換タスクの連続学習を行った．実験から, 領域分割と濃淡画像着色, スタイル変換の連続学習において, Encoder-Decoder CNN に Piggyback を適用することで最小限のオーバーヘッドでベースラインと同程度の性能を発揮すること, また Resblock 付き Encoder-Decoder CNN は領域分割以外のタスクで個別に学習したモデルと同等のかそれ以上の性能を発揮することがわかった．

今後は Piggyback で学習したバイナリーマスクの圧縮や Piggyback と Resblock を組み合わせることによるオーバーヘッドの削減や, Resblock での領域分割の学習の工夫による性能の向上を行いたい．また, 実験では三種類のタスクのみでの実験であり, 提案手法の汎用性を判断するには不十分であった．そこで, CVPR 2017 PASCAL in Detail Workshop Challenge で行われた十種類の分類タスクを行う Visual Domain Decathlon のような設定を画像変換タスクに適応したものや, Kokkinos が

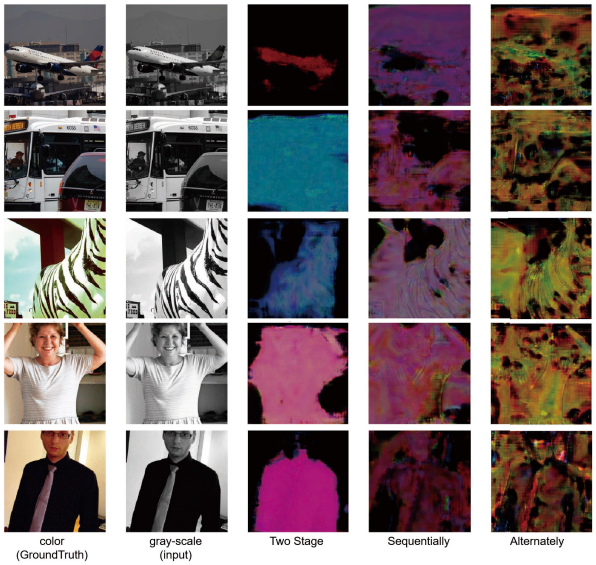

図 19　タスク 3(濃淡画像着色) とタスク 2(Pascal VOC 領域分割) の連結

図 20　タスク 3(濃淡画像着色) とタスク 3(濃淡画像着色) の連結

Ubernet [13] で行った実験を参考に追加実験を行い, 提案手法の汎用性を確認したい.

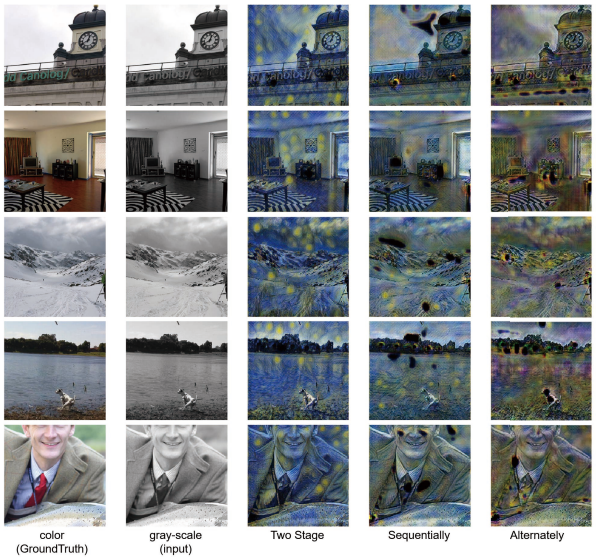

図 21　タスク 3(濃淡画像着色) とタスク 4(スタイル変換) の連結

文　　献

[1] A. Mallya, S. Lazebnik, and D. Davis. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proc.of European Conference on Computer Vision (ECCV)*, pp. 67–82, 2018.

[2] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proc.of IEEE Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[3] A. Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. In *Jour.of Connection Science*, Vol. 7, pp. 123–146. Citeseer, 1995.

[4] K. Shmelkov, C. Schmid, and K. Alahari. Incremental learning of object detectors without catastrophic forgetting. In *Proc.of IEEE International Conference on Computer Vision (ICCV)*, 2017.

[5] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. In *Proc.of the National Academy of Sciences (PNAS)*, Vol. abs/1612.00796, 2016.

[6] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. In *arXiv preprint arXiv:1606.04671*, 2016.

[7] A. Mallya and S. Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proc.of IEEE Computer Vision and Pattern Recognition (CVPR)*, pp. 7765–7773, 2018.

[8] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *Proc.of European Conference on Computer Vision (ECCV)*, 2016.

[9] P Isola, J Zhu, T. Zhou, and A. Efros. Image-to-image translation with conditional adversarial networks. In *Proc.of IEEE Computer Vision and Pattern Recognition (CVPR)*, 2017.

[10] J. Zhu, T. Park, P. Isola, and A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proc.of IEEE International Conference on Computer Vision (ICCV)*, 2017.

[11] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Proc.of International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pp. 234–241. Springer, 2015.

[12] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *Proc.of IEEE Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[13] I. Kokkinos. Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. In *Proc.of IEEE Computer Vision and Pattern Recognition (CVPR)*, pp. 6129–6138, 2017.