重み選択マスクを用いた画像変換ネットワークの連続学習

松本 晨人[1,a)]　柳井 啓司[1,b)]

概要

これまで CNN での連続学習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. そこで画像変換タスクでの連続学習の研究を行った. 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 [7] 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した.

1. はじめに

人間や動物は脳の豊富な神経認知機能によって生涯にわたって昔学習した知識を忘れることなく新しい知識を獲得することができる. このような学習を連続学習という. 連続学習は一つの CNN で複数のタスクに対応できるため, アプリ実装や汎用人口知能の実現に貢献すると考えられる. しかし, CNN の連続学習では新しいタスクを学習すると昔のタスクの精度が低下する破壊的忘却が起こる. 破壊的忘却を回避する様々な手法が提案されているが, それらの多くは画像のクラス分類や強化学習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. そこで本論文では 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. 具体的には, (1) Encoder-Decoder CNN の重みを選択する手法である Piggyback [7] の適用, または (2) 入力した画像をそのまま出力する Auto Encoder にタスク固有の Resblock 追加によって, 連続学習を実現する. さらに (1) と (2) を組み合わせた (3) Piggyback を Resblock に適用する手法を試した.

2. 関連研究

破壊的忘却を回避する手法として, ここでは最適化 [4] と重みの選択 [7] について説明する. EWC [4] 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. ただし重要な重みが異なるタスクの連続学習の場合, 各タスクの性能が低下してしまう問題がある. Piggyback [7] では EWC の問題を解決し, 高い精度で大量のタスクを連続学習することに成功した. Piggyback は学習済みのベース CNN から各タスクで重要な重みをマスクを用いて選択する手法である. パラメータを固定した CNN からそれぞれのタスクに最適化な重みを選択することで精度を高める. ベース CNN のパラメータを固定しているため破壊的忘却は起こらない. 本論文では一つ目の手法として, Piggyback を利用したものを提案する.

[1] 電気通信大学情報理工学部総合情報学科
[a)] matsumo-a@mm.inf.uec.ac.jp
[b)] yanai@cs.uec.ac.jp

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.

3.1 Piggyback を用いた Encoder Decoder CNN

一つ目の連続学習の手法は Piggyback [7] 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 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
(2) 実数マスクを作成
(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
(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
(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 [7] 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 [7] と同様に 5e-3 として実験を行った.

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 を学習するため破壊的忘却は当然起こらない．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．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．Auto Encoder と Resblock の構造は Zhu らの CycleGAN [9] を参考にした．構造の詳細を補助資料の図 11 に示す．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

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 [3] 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．

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)

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 Resblock の概略を図 1 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．ただし,“fine-tune” のタスク 5 はタスク 3 の後, fine-tuning したものである．

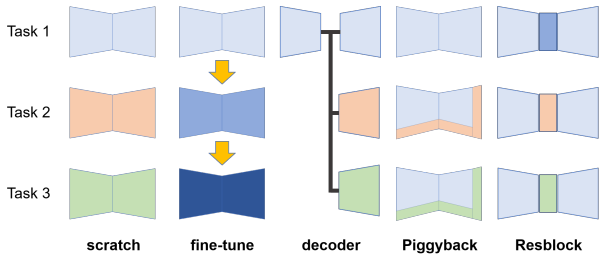

図 1 各手法の概略

学習時の損失関数は, タスク 1, 2 は Cross Entropy Loss, タスク 3 は L2, タスク 4, 5 は Johnson ら [3] 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 [1] 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 2 に示す．さらに各タスクで画像を生成した追加の例を補助資料の図 12, 図 13, 図 14, 図 15 にまとめた．“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．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いることがわかる．“Resblock” での領域分割タスクの精度が低くなった原因は学習時に最終層を固定しているため損失関数に通常の領域分割タスクに用いられる sigmoid cross entropy を使用せず, Adversarial Loss を使用しピクセル単位の変換の学習が出来なかったためと考えられる．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” では破壊的忘却が起きているが, “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あるといえる．

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	Resblock(6)
タスク 1 (mIoU(%))			**21.47**		0.57
タスク 2 (mIoU(%))	58.59	**64.87**	61.63	61.45	4.26
タスク 3	244.000	**237.92**	241.66	242.49	532.83
(MSE, SSIM)	0.9138	0.9148	0.9121	0.9058	**0.9281**
タスク 4	**0.3678**	0.3555	0.3595	0.3501	0.3467
(SSIM, total loss(epoch))	413833 (200)	**405893** (200)	473723 (200)	528587 (100)	460268 (200)
タスク 5	**447480** (6)	490490 (6)	544348 (6)	521476 (6)	520221 (6)
(total loss(epoch))					
タスク 1 after タスク 2	-	0.70	21.47	21.47	0.57
タスク 2 after タスク 3	-	1.87	61.63	61.45	4.26
タスク 3 after タスク 4	-	870.18	241.66	242.49	532.83
(MSE, SSIM)	-	0.5321	0.9121	0.9058	0.9281
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)

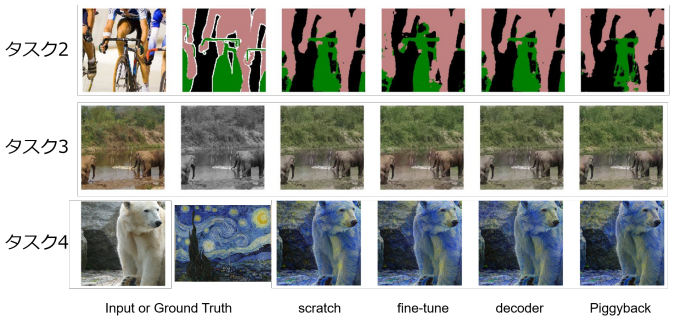

図 2 各タスクの生成結果 (上から順にタスク 2,3,4)

5. 考察

5.1 Piggyback のバイナリーマスク

ここでは Mallya ら [7] 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 3) 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 5, 図 6, 図 7, 図 8 に示す．また，Mallya ら [7] の ImageNet 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 4 に示す．図 5 から図 8 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 3 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 [7] 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 [6] 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．

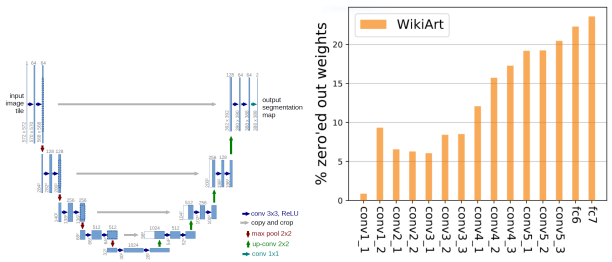

図 3 U-Net [8]

図 4 画像分類タスク [7]

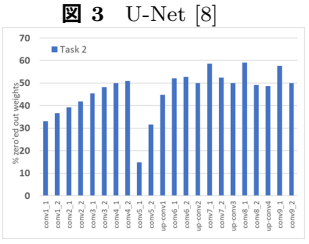

図 5 タスク 2(領域分割)

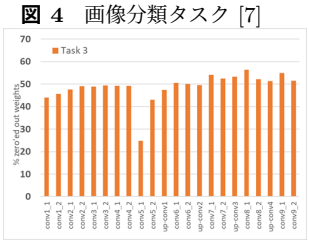

図 6 タスク 3(濃淡画像着色)

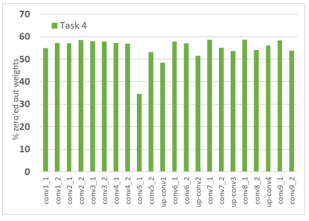

図 7 タスク 4(スタイル変換)

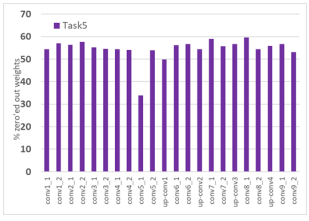

図 8 タスク 5(スタイル変換)

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 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 + Piggyback

ここでは本論文で実験した Piggyback と Resblock を組み合わせた手法についてまとめる. “Resblock” を用いてタスク 3(濃淡画像着色) を学習したのちに, そのモデルに “Piggyback” を用いてタスク 4(スタイル変換) を学習した. この二つの連続学習を組み合わせた手法を “Resblock+Piggyback” とする. 二つのタスクにおける “Resblock” のモデルサイズは 287.4MB(8.7+135.0×2) であるのに対し, “Resblock+Piggyback” のモデルサイズは 147.9MB(8.7 + 135.0 + 4.2) で 139.5MB 小さくなった. “Resblock+Piggyback” の生成結果を図 9 に示す. 図 9 の中央の列は “Resblock”, 右の列は “Resblock+Piggyback” の生成結果を表示している. 図 9 から “Resblock” と比較すると “Resblock+Piggyback” は低品質な画像を生成したことがわかる. “Resblock+Piggyback” の性能が向上しなかった原因は二つ考えられる. 一つ目の原因はパラメータ数が足らなかったことである. “Piggyback” を行った U-Net でマスクに使用した重みの数は 14,776,000 個であるのに対して, “Resblock+Piggyback” は “Piggyback” の半分以下の 7,077,888 個であり, 画像変換タスクに Piggyback を適応するのには重みが少なかったことが考えられる. 二つ目の原因は “Resblock” はタスクに固有な変換を獲得する学習法であることである. “Resblock+Piggyback” のバイナリーマスクの 0 の割合を図 10 に示す. “Piggyback” のバイナリーマスクの 0 の割合と比較すると 0 の割合が大きくなったことがわかり, 元の重みをあまり再利用していないといえる. Resblock は Encoder 部分から得た特徴量からタスク固有の変換を行うように学習したため, そのタスクに特化した重みを獲得した一方で, 汎用的な重みは習得できなかったのではないかと考えられる.

図 9 生成結果

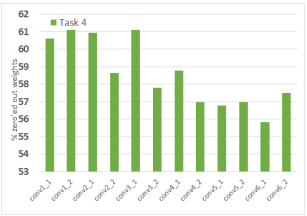

図 10 マスクの 0 の割合

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 での領域分割や “Resblock + Piggyback” の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 Ubernet [5] 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.

参考文献

[1] Gatys, L. A., Ecker, A. S. and Bethge, M.: Image Style Transfer Using Convolutional Neural Networks, *Proc.of IEEE Computer Vision and Pattern Recognition (CVPR)* (2016).

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

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

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

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

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

[7] Mallya, A., Lazebnik, S. and Davis, D.: Piggyback: Adapting a Single Network to Multiple Tasks by Learning to Mask Weights, *Proc.of European Conference on Computer Vision (ECCV)*, pp. 67–82 (2018).

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

[9] Zhu, J., Park, T., Isola, P. and Efros, A.: Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, *Proc.of IEEE International Conference on Computer Vision (ICCV)* (2017).

補助資料

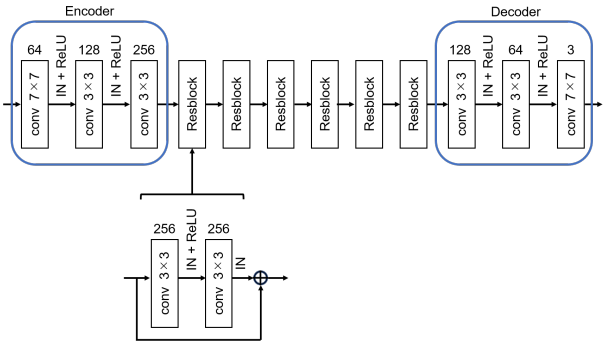

図 11　Auto Encoder と Resblock 構造

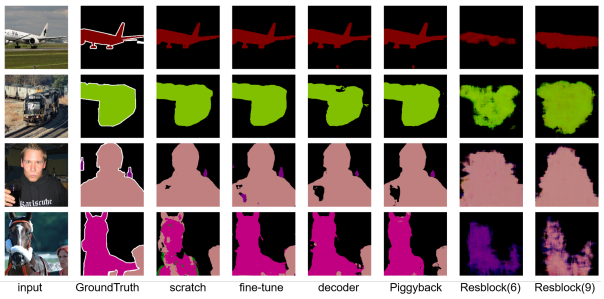

図 12　タスク 2 の結果 (Pascal VOC での領域分割)

図 13　タスク 3 の結果 (濃淡画像着色)

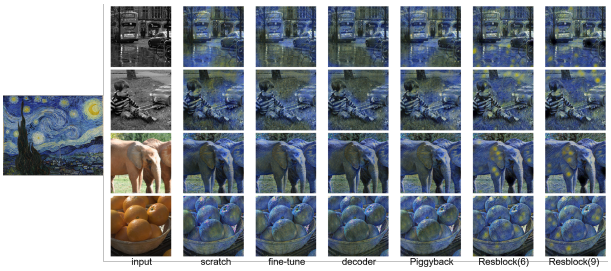

図 14　タスク 4 の結果 (スタイル変換)

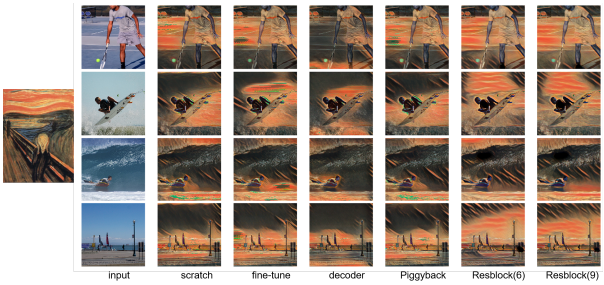

図 15　タスク 5 の結果 (スタイル変換)