CI-002

CNN を用いたアニメ線画の自動彩色手法の開発および参考画像の構図の検証
Towards anime drawing colorization with reference-guided convolutional neural network and composition verification

南谷 大輔† 米澤 弘毅‡
Daisuke Nanya Kouki Yonezawa

1. はじめに

近年のコンピュータ技術の発達により、アニメ業界でも 3DCG を用いた映像制作や液晶ペンタブレットを用いた絵の描画などコンピュータを用いた作品制作が多数行われている。 また、作品制作にコンピュータを用いることで労力の削減を目指す取り組みも行われている。

モノクロの画像をカラー化する研究は実画像を対象としたものが多く、アニメやイラストなどの絵を対象とした研究はあまり行われていない。

本研究では、参考画像とアニメ線画をニューラルネットワークへ入力し、参考画像をヒントに線画を彩色する手法の開発を行う。これにより、希望する色で線画を彩色することが目標である。

2. 先行研究

2.1 Tag2Pix[1]

この手法は、タグ情報をヒントにして線画を彩色する手法であり、GAN（敵対的生成ネットワーク）が基になっている。タグ情報には、彩色したいパーツとそれに対応する色が紐づけられている。タグ情報は色情報に変換された後、Generator の Decoder の部分で線画情報と結合され、彩色された画像が出力される。

3. 提案手法

3.1 提案手法の概要

今回構築したネットワークは、以前提案したネットワーク[2]を改良したもので、GAN（敵対的生成ネットワーク）が基になっている。構築したネットワーク図を図 1 に示す。このネットワークは、参考画像から得られる色情報をヒントとして線画を彩色する。参考画像から得られる色情報については 3.3 節で詳細を説明する。

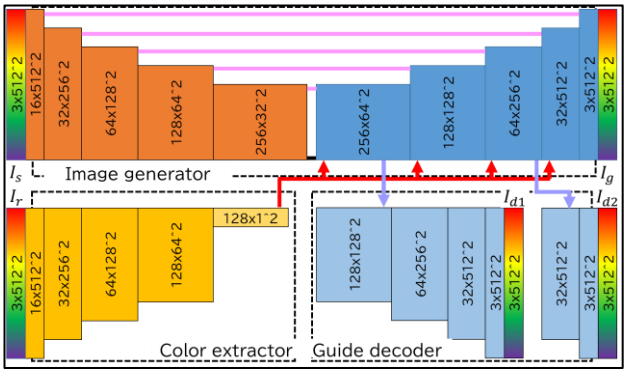

図 2　提案するニューラルネットワークの全体像

構築したネットワーク(図 1)について説明する。I_s は入力画像、I_r は参考画像、I_g は生成画像である。I_{d1} はガイドデコーダー1 による生成画像、I_{d2} はガイドデコーダー2 による生成画像である。ガイドデコーダーについては 3.4 で詳細を説明する。またブロック内に書かれている数字は、対象のブロックが出力する画像のサイズを次元数×画像のピクセル数の 2 乗と表している。Discriminator については Tag2Pix[1]と同じものを使用した。

3.2 畳み込みを用いた画像生成

画像は図 1 の Image Generator の部分で生成され、Encoder と Decoder から構成される。Encoder と Decoder はそれぞれ 5 個のブロックがあり、U-net 構造となっている。

Encoder のネットワークを図 2 に示す。Encoder は畳み込み層、インスタンス正規化層、Leaky ReLU 層の組み合わせで構成される。カッコの中はそれぞれの層のパラメータである。畳み込み層のパラメータについては、k はカーネル、 p はパディング、 s はストライドを表す。Leaky ReLU 層のパラメータは 0.2 と設定した。ただし、入力画像が最初に入力される Encoder のブロックの中にある全ての畳み込み層は、パディングの値が 1 となる。

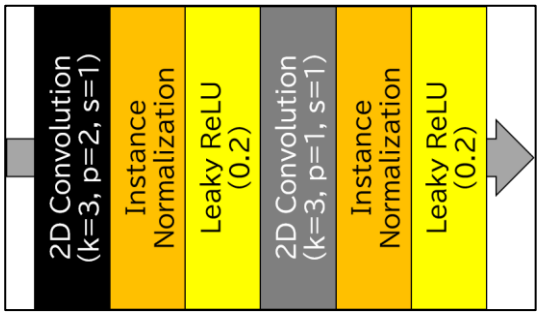

図 1　Encoder のネットワーク

次に Decoder のネットワークを図 3 に示す。Decoder のブロックでは SECat-ResNeXt Block を用いて線画と色情報を結合する。

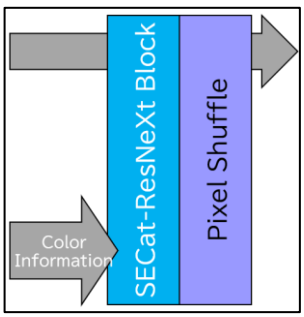

図 3　Decoder のネットワーク

†名城大学理工学研究科情報工学専攻・Meijo University
‡名城大学情報工学部情報工学科・Meijo University

ただし、最後の Decoder のブロックは図 3 と異なり、色情報との結合は行わない。最後はハイパーボリックタンジェント層を経由することで彩色された画像を生成する。最後の Decoder のネットワークを図 4 に示す。

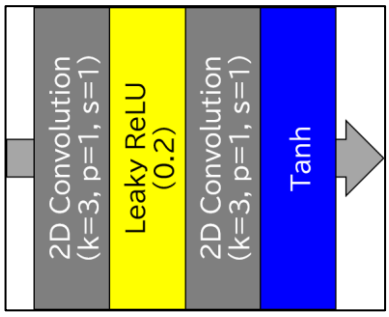

図 5　最後の Decoder のネットワーク

3.3 色情報の生成

色情報は図 1 の Color extractor の部分で生成され、4 個の Encoder のブロックから構成される。Encoder のネットワークを図 5 に示す。Color extractor は畳み込み層、ReLU 層、プーリング層から構成される。

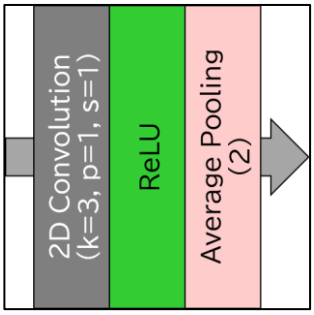

図 7　Color extractor のネットワーク

参考画像は Encoder による処理が終わった後、Global Average Pooling 層を経由して 128 次元の色情報に変換される。以前の論文[2]より、色情報の次元数は 128 次元が生成される画像の精度が最も高かったため、今回も色情報の次元数は 128 次元とした。

参考画像はネットワークの学習時に変形処理を行った後に Color extractor へ入力される。これは、テスト時に線画と参考画像の構図が異なる場合でも彩色できるようにするためである。変形処理は、torchvision ライブラリの RandomPerspective を用いて行った。distortion_scale を 0.5 と設定し、空白になった場所には白色で塗りつぶされるようにした。

ただし、テスト時は参考画像の変形処理は行わない。

3.4 ガイドデコーダー

ガイドデコーダーは 2 個あり、Decoder の途中の情報がガイドデコーダーへ入力される。以前構築したネットワーク[2]では、ガイドデコーダーは 1 個のみであったが、2 個にすることでより高品質な画像を生成できると考えガイドデコーダーの数を増やした。

ガイドデコーダーのネットワークを図 6 に示す。ガイドデコーダーは逆畳み込み層、Leaky ReLU 層から構成される。

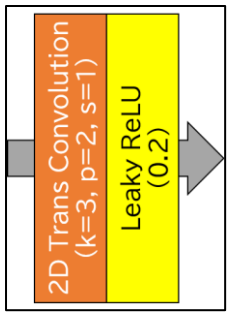

図 4　ガイドデコーダーのネットワーク

ただし、最後のブロックは図 6 と異なり、ハイパーボリックタンジェント層を経由する。最後のブロックのネットワークを図 7 に示す。

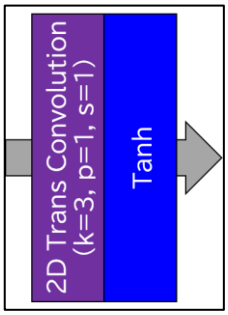

図 6　最後のガイドデコーダーのネットワーク

3.5 損失関数

Discriminator の損失関数は、GAN における画像生成分野で使われることの多い交差エントロピー誤差(BCEloss)を用いて以下のように計算される。

$$L_D = E_y\left[\log D(y)\right] + E_x\left[\log\left(1 - D\left(G_g(x,r)\right)\right)\right]$$

ここで x は入力画像、r は参考画像、y は正解画像を表す. また $G_g(\cdot)$ は Generator による生成画像、$D(\cdot)$ は Discriminator の出力値を表す

Generator はガイドデコーダーを 2 個備えており，その損失関数は交差エントロピー誤差(BCEloss)と平均絶対誤差(l1-loss)から以下のように計算される。

$$L_G = E_x\left[\log D\left(G_g(x,r)\right)\right] + \lambda\left(E_{x,y}\left[\left\|y - G_g(x,r)\right\|_1 + \beta_1\left\|y - G_{d1}(x,r)\right\|_1 + \beta_2\left\|y - G_{d2}(x,r)\right\|_1\right]\right)$$

ここで $G_{d1}(\cdot)$ はガイドデコーダー1 による生成画像、$G_{d2}(\cdot)$ はガイドデコーダー2 による生成画像である。また $\lambda, \beta_1, \beta_2$ はパラメータである。今回は $\lambda = 2000$，$\beta_1 = 0.8$，$\beta_2 = 0.2$ と設定した。

4. 検証結果

4.1 ネットワークの学習

ネットワークの学習には、Kaggle で提供されているデータセット[3]を使用した。データセットには、カラー画像と対応する線画がペアになっている。画像は全部で 17769 枚あり、その中から 14424 枚をネットワークの学習用、3545 枚をテスト用として利用した。

エポック数は 1 から 100 まで学習を行い、オプティマイザーは Adam オプティマイザーを使用した。Generator と Discriminator の損失値はすべてのエポックで、学習データは 5 エポックおきに保存した。

ネットワークの学習は、Intel Xeon W-2265 の CPU を 1 個と NVIDIA RTX A4500 の GPU を 1 個搭載したコンピュータで実施した。

訓練データは毎回シャッフルして学習を行い、再現性を確保するためにシード値は固定した。

4.2 出力画像の検証

テスト用データで出力画像の確認を行った。図 8 は左から順にネットワークへ入力する参考画像、線画と線画に対するカラー画像である。

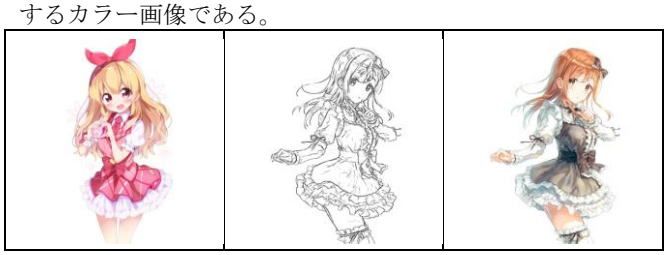

図 8　ネットワークへ入力する参考画像、線画と線画に対するカラー画像

図 9 は、図 8 の左の画像を参考画像、中の画像を線画としてネットワークに入力した場合の出力画像である。左の列から順にバッチサイズを 1,2,4 で学習した場合の出力画像である。また、上の行から順にエポック数を 60,80,100 で学習した場合の出力画像である。

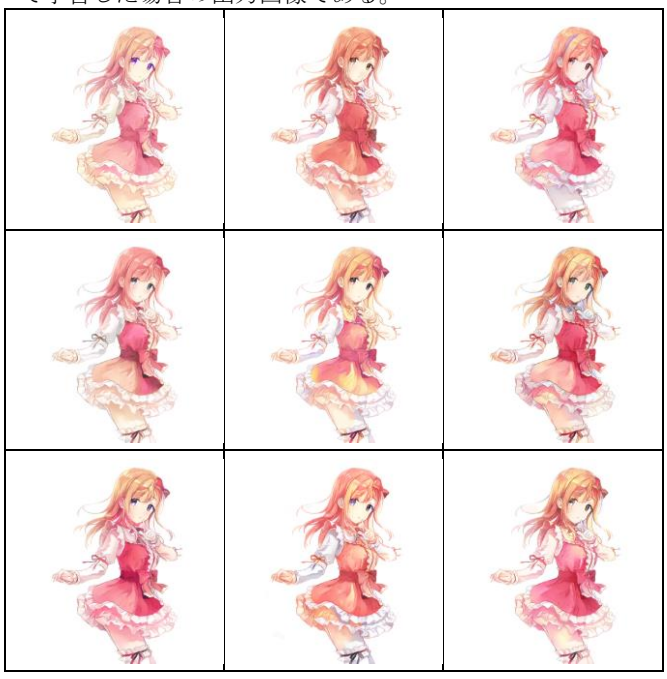

図 9　ネットワークの出力画像

図 10 は、出力画像を各部位ごとに拡大した画像である。上の行から顔、手、服、足の部位であり、左の列から順にバッチサイズを 1,2,4 で学習した場合の出力画像である。なお、エポック数 100 の学習データを用いて彩色を行った。

出力画像を見ると参考画像の色使いを基に線画が彩色されていることが確認できる。また、全体的にエポック数が増加するときれいに彩色されていることも確認できる。

しかし、学習時のバッチサイズを変更することで出力画像に細かな差を確認することもできる。具体的には、バッチサイズが 2 よりも 1 や 4 の方がより自然に彩色できているように見える。特に足の部分で違いが顕著に表れている。バッチサイズが 2 では、白く塗られてしまい彩色に失敗していることが読み取れる。一方、バッチサイズが 1 や 4 では、肌色で塗られておりきれいに彩色されているように見える。

他にもバッチサイズが 4 では、目の部分に青色のノイズが発生し、バッチサイズが 1 では、服の部分で色が濃く彩色されるといった特徴が確認できた。

図 10　各部位を拡大した出力画像

次に線画を変形させ、どこまで自然に彩色できるのか確認する。図 11 に変形させた線画から生成される画像の一覧を示す。参考画像は図 8 の左の画像を使用した。なお、バッチサイズが 4 で、エポック数 100 の学習データを用いて彩色を行った。

図 11 の左上と中上は、画像全体を左右に移動した場合の出力画像である。右上と二行目左は、画像の頂点を移動させ、画像全体を歪ませた場合の出力画像である。二行目中と二行目右は、順に画像全体を時計回りと半時計回りに 10 度回転させた場合の出力画像である。左下と中下と右下はそれぞれ時計回りに 90 度、180 度、270 度回転させた場合の出力画像である。

出力画像を確認すると、一行目と二行目の画像は左手の部分が白色で彩色されてしまっているが、概ね高品質な画像が生成されているように見える。一方、三行目の大きく回転させた場合は、目や髪などの細かなパーツで彩色ミスが見受けられた。よって、画像が大きく回転しなければ、多少の歪みがあっても高品質に彩色されることがわかった。

なお、線画の変形処理はフリーソフトである GIMP を用いて行った。

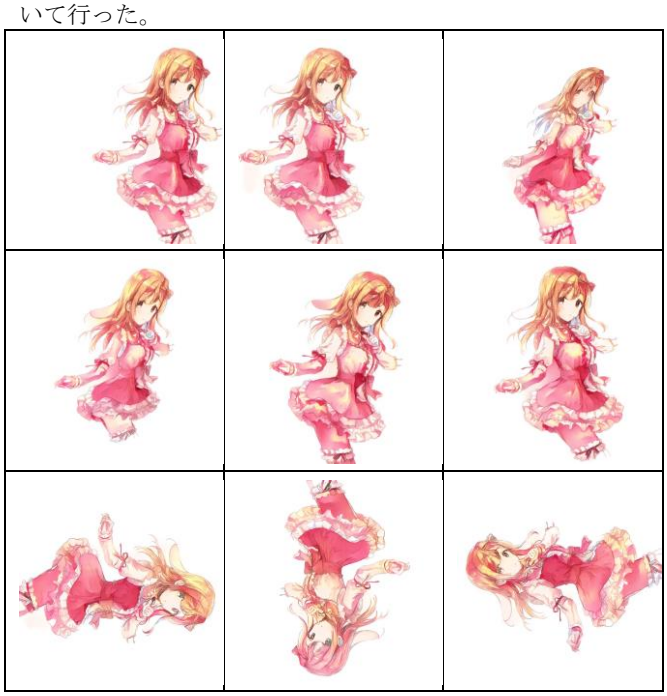

図 11　変形した線画から生成される画像

4.3 損失値の確認

Discriminator の損失値の推移を図 12 に示す。バッチサイズは 1,2,4 の 3 種類で検証を行った。

バッチサイズは 2 が最も速く損失値の収束が速くなることが確認できた。バッチサイズが 4 の場合は、1 の場合よりも遅く収束することが読み取れる。よって、バッチサイズが大きいほど速く収束するとは言えないことがわかった。また、損失値は 0.1 の辺りで収束されることが読み取れる。

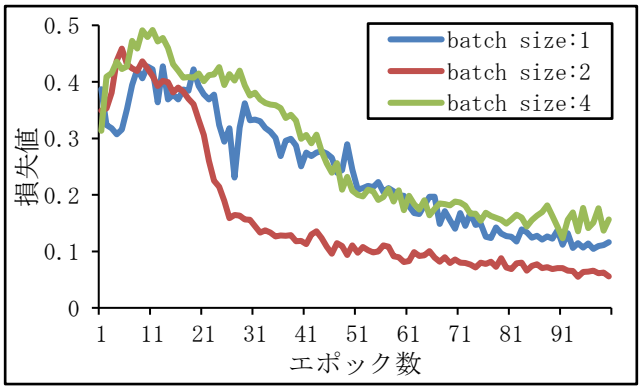

図 12　学習の進行による Discriminator の損失値の変動

Generator の損失値の推移を図 13 に示す。バッチサイズは先ほどと同じく 1,2,4 の 3 種類で検証を行った。

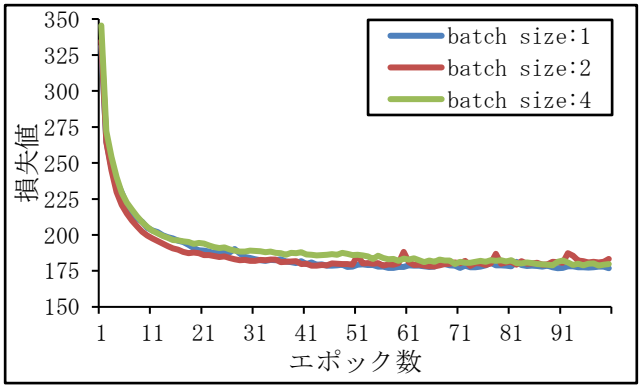

図 13　学習の進行による Generator の損失値の変動

エポック数 40 辺りまでバッチサイズ 2 が最も速く損失値が下がり、その後はバッチサイズ 1 が低い損失値を取ることが確認できた。バッチサイズが 4 の場合は、1 の場合よりも遅く収束することが読み取れる。よって、Generator もバッチサイズが大きいほど速く収束するとは言えないことがわかった。また、Generator は Discriminator より速く収束し、エポック数 50 辺りで損失値は約 175 で収束されることが読み取れる。損失値が一般的な GAN よりも大きいのは、損失関数のパラメータで$\lambda = 2000$という大きな値を扱っているためである。

5. おわりに

本研究では、アニメ線画の彩色を目的としてネットワークの構築と検証を行った。多少の歪みであれば、きれいに彩色されることが確認できた。今後は、Attention 機構を搭載した手法[4]などを参考に、より高品質な画像を生成できるネットワークの構築を目指したい。

参考文献

[1] H. Kim, H.Y. Jhoo, E. Park, S. Yoo, Tag2Pix: Line Art Colorization Using Text Tag With SECat and Changing Loss, in: 2019: pp. 9056–9065. https://openaccess.thecvf.com/content_ICCV_2019/html/Kim_Tag2Pix_Line_Art_Colorization_Using_Text_Tag_With_SECat_and_ICCV_2019_paper.html (accessed April 20, 2022).

[2] D. Nanya, K. Yonezawa, Towards automatic colorization with reference-guided neural network, in: Information Processing Society of Japan, 2023.

[3] TAEBUM KIM, Anime Sketch Colorization Pair, Kaggle.Com. (2019). https://www.kaggle.com/datasets/ktaebum/anime-sketch-colorization-pair.

[4] C. Yu, M. Xiangqiao, M. P.Y., L. Xueting, L. Tong-Yee, L. Ping, AnimeDiffusion: Anime Face Line Drawing Colorization via Diffusion Models, (2023). https://arxiv.org/pdf/2303.11137.