令和４年度　特別研究報告書

画像生成ネットワークを用いた顔画像の日本アニメ風変換に関する検討

龍谷大学大学院　理工学研究科　情報メディア学専攻

T19M059　近澤勇太

指導教員　三好力　教授

内容梗概

本研究では顔画像をアニメキャラクター化するための新たなGANの損失を提案する.日本のアニメーションは主に手書きであり,アニメーターの手によって動かされることを前提としている.その中でも,アニメキャラクターは非常に高度な単純化と誇張表現を兼ね備えているため,実写の顔画像をアニメキャラクターに変換するようにGANに学習させることは難しい.

そこで,本研究では実写の顔画像をアニメキャラクターに変換するための新たな損失を提案した.新たな損失は,顔写真のランドマークとGeneratorが生成した画像のランドマークを比較するものである.生成画像のランドマーク検出には,新たにアニメキャラクターのデータセットでトレーニングしたDANをランドマーク検出器として使用した.そして,提案した手法をもとにネットワークの学習と画像生成を行い,既存の手法と比較して提案手法より上手くアニメキャラクター化できることを確認した.

abstract

In this research, we propose a new GAN loss function for translate face images into anime characters. Japanese animation is mostly hand-painted and is supposed to be animated by animators. Among them, anime characters have been strongly simplified and exaggerated, so it is difficult to train a GAN to convert real face images into anime characters.

Therefore, we propose a new loss function for translate real face images into anime characters. A new loss function compares the landmarks in the real face images with the landmarks in the image generated by the Generator. The landmark detection in generated images is made by a new DAN trained using anime character dataset. Then, we confirmed that the proposed method can create anime characters better than conventional methods.

目次

第 1 章 はじめに

近年,様々な SNS(ソーシャル・ネットワーキング・サービス)の発展により,多くの人が SNS のアカウントを所持している[1].特に日本では LINE,Twitter,Youtube などが上位を占める.この中でも,Twitter や Youtube はユーザ同士のコミュニケーションだけではなく,広く情報を発信するツールとしての側面を持っている.しかし,Twitter や Youtube などで実名や顔を晒すことは危険が伴うことがあり,特に日本ではその危惧から匿名での利用者が他国と比べて多い[2].このことからも,SNS 上で多くの人が顔を晒すことに抵抗があることが考えられる.

匿名での利用が多い中で,SNS で自身を表すアイコンには,キャラクターのイラストなどを使用する場合が多い.しかし,オリジナルのキャラクターのイラストを自前で準備することは難しいため,既存のアニメキャラクターや他人のイラストなどを無断で使用してしまう場合がある.それによって著作権上の問題が起こってしまう.そこで,顔画像をもとにして自動的に日本のアニメキャラクター風の画像を生成することができれば,キャラクター画像を容易に準備することができるため,著作権の侵害を減らすことができると考えた.

画像から画像への変換は,GAN(Generative Adversarial Networks)[3]の登場により盛んに研究されている.しかし,他の芸術的スタイルと比べて日本のアニメ風への変換は難しい部分が多い.日本のアニメーションは主に手書きであり,アニメーターの手によって動かされることを前提としているため,動かされる対象物と背景では絵のスタイルは異なる.その中でも,アニメキャラクターは非常に高度な単純化と誇張表現を兼ね備えており,アニメ風への変換は難しいとされる.アニメキャラクターの特徴として以下のようなものが挙げられる.

1.はっきりとしたエッジ
2.なめらかなテクスチャ
3.実写と比べて少ない線
4.表情がわかりやすいように誇張されたパーツ(目,口)
5.単純化されたパーツ(鼻など)

GAN を利用してアニメ風への変換を試みた研究には CartoonGAN[4]がある.これは日本のアニメーションのテクスチャスタイルに特化した手法で,風景画像をアニメ風に変換することができる.CartoonGAN では事前にトレーニングされた VGG ネットワーク[5]を使用し,変換前と変換後の画像に写り込んでいるオブジェクトを識別する.変換前と変換後で写り込んでいるオブジェクトの間の損失の差を小さくすることで,アニメ風のスタイルだけを転写できる.

本研究ではこの手法を参考に,顔画像に特化した新たな損失関数を提案する.提案する新しい損失関数は,元画像と変換後の顔のランドマーク座標の差を損失とするもので,ランドマーク損失と名付けた.元画像のランドマーク座標は Dlib[17]によって検出したものを使用し,変換後のアニメキャラクター画像のランドマーク検出には,Deep Alignment Network[7]を使用した検出器を新たにトレーニングし使用する.これは,画像全体から顔の特徴を抽出し,ランドマークの位置合わせを行うものである.この検出器を使用することで,顔パーツの位置あわせとアニメキャラクターの特徴をもつ画像を生成することを期待する.

本論文の構成は次の通りである.本章では,本研究が顔画像をアニメキャラクターに変換することを目的とすることを述べ,その研究を行う上での課題の説明を行った.2 章では,関連研究として,画像生成と画像から画像への変換タスクに関連する研究と,顔のランドマーク検出手法についての説明を行う.3 章では提案手法であるランドマーク損失を使用した GAN について述べる.4 章ではアニメキャラクターの顔画像検出器を作成する実験を行い,5 章ではランドマーク損失を使用して GAN のトレーニングと画像生成を行い,その有用性を示す.6 章では本論文のまとめと今後の展望について述べる.

第 2 章　関連研究

本章では,提案手法と関連する研究についての説明を行う.2.1 では根幹技術である畳み込みニューラルネットワークについて説明し,2.2 では本研究にも使用する VGG についての説明を行う.2.3 では生成ネットワークなどで用いられる ResNet について説明し,2.4 では画像生成手法の根幹となる敵対的生成ネットワークについて述べる.2.5 と 2.6 では本研究で使用する,アニメ風への変換を行う CartooonGAN と AnimeGAN について述べる.2.7 では GAN が生成した画像のランドマークを検出するためのランドマーク検出器を作成するのに使用する Deep Alignment Network について説明を行う.2.8,2.9,2.10 ではネットワークの学習に使用する主な活性化関数,損失関数,最適化アルゴリズムについて説明を行う.

2.1　畳み込みニューラルネットワーク

畳み込みニューラルネットワーク(Convolutional Neural Network:CNN)は主に畳み込み層,プーリング層,全結合層の 3 つの結合によって構成される.畳み込みニューラルネットワークのネットワーク構成の例を図 2.1 に示す.図に示したように,畳み込みニューラルネットワークは畳み込み層とプーリング層を串返し,最後に全結合層を通して出力する構成になっている.元の画像から,畳み込み層では局所的な特徴を抽出し,それをプーリング層でまとめる役割を持っている.畳み込み層で抽出した特徴をプーリング層でまとめることで,小さな微小なデータの変化に対して強くなる.全結合層では,取り出された特徴を一次元の全結合層では,取り出した特徴を一つのノードに結合し,活性化関数によって変換された値を出力する.出力層では,全結合層からの出力を元に,ソフトマックス関数などで確率に変換し分類を行う.

図 2. 1 畳み込みニューラルネットワークの構成例

畳み込み層では入力に対してフィルタを用いて畳み込み演算が行われる.フィルタは学習可能なパラメータである重みで構成される.入力が画像などの 2 次元の場合の畳み込み演算の例を図 2.2 に示す.図のように,対応する部分を掛け合わせその合計をとる.これをストライドに合わせて,ずらしていき計算を行う.畳み込み演算によって,入力と出力の次元は異なる.畳み込み演算による出力結果の次元は次式によって求めることができる.ここで,出力の高さをO_H,幅をO_w,フィルタの高さをF_H,幅をF_w,ストライドをSとする.

$$O_H = \frac{H + 2 - F_H}{S} + 1 \tag{2.1}$$

$$O_W = \frac{W + 2 - F_W}{S} + 1 \tag{2.2}$$

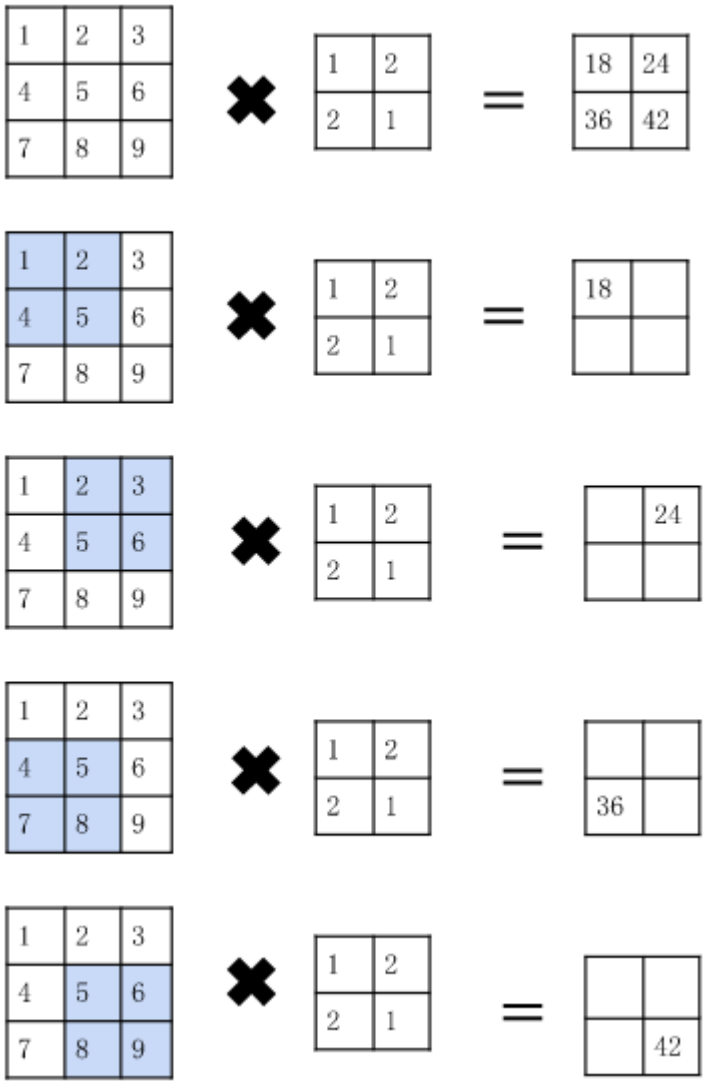

図 2.2　畳み込み演算の例

2.2 VGG

VGG[8]は大規模な画像データセットである ImageNet によってあらかじめ学習された CNN モデルの学習済みパラメータである.一般的にVGG16と呼ばれる畳み込み層 13 層, 全結合層 3 層のモデルと,VGG19 と呼ばれる畳み込み層 16 層,全結合層 3 層のモデルが使用される.VGG19 のネットワーク構成を図 2.3 に示す.Conv は畳み込み層を示し,MaxPooling はプーリング層,Dense は全結合層を示す.畳み込み層と全結合層の活性化関数はすべて Rectified Linear Units (ReLU)関数が使用され,softmax 関数を通して出力される

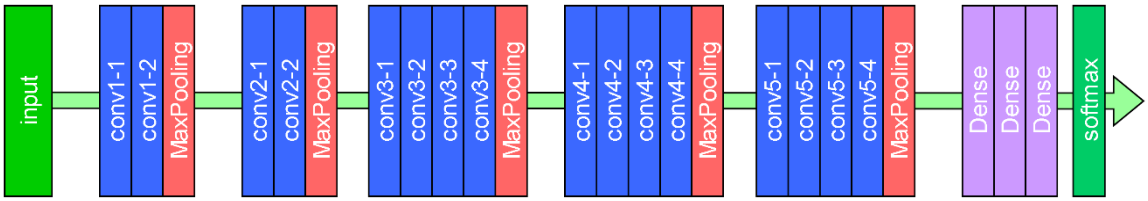

図 2. 3　VGG19 のネットワーク構成

2.3 ResNet

ResNet[9]はより深い CNN を実現するためのネットワークモデルである.多くの研究で,深いニューラルネットワークの方が複雑な関数を表現することができる可能性を示している.しかし,勾配消失が起こるため学習させることが困難になる.ResNet では残差ブロックと呼ばれる構造で勾配消失問題を軽減している.残差ブロックの構造を図 2.3 に示す.図のようにスキップ接続と呼ばれる変換元の情報を保持する接続を用いることで,勾配消失の問題を軽減する.ニューラルネットの入力を x,ニューラルネットによる出力を F(x)とすると,学習目標である関数を H(x)とスキップ接続は次式のように表すことができる.

$$H(x) := F(x) + x \tag{2.3}$$

これにより,ニューラルネット F(x)は H(x) − x を得るように学習を進める.

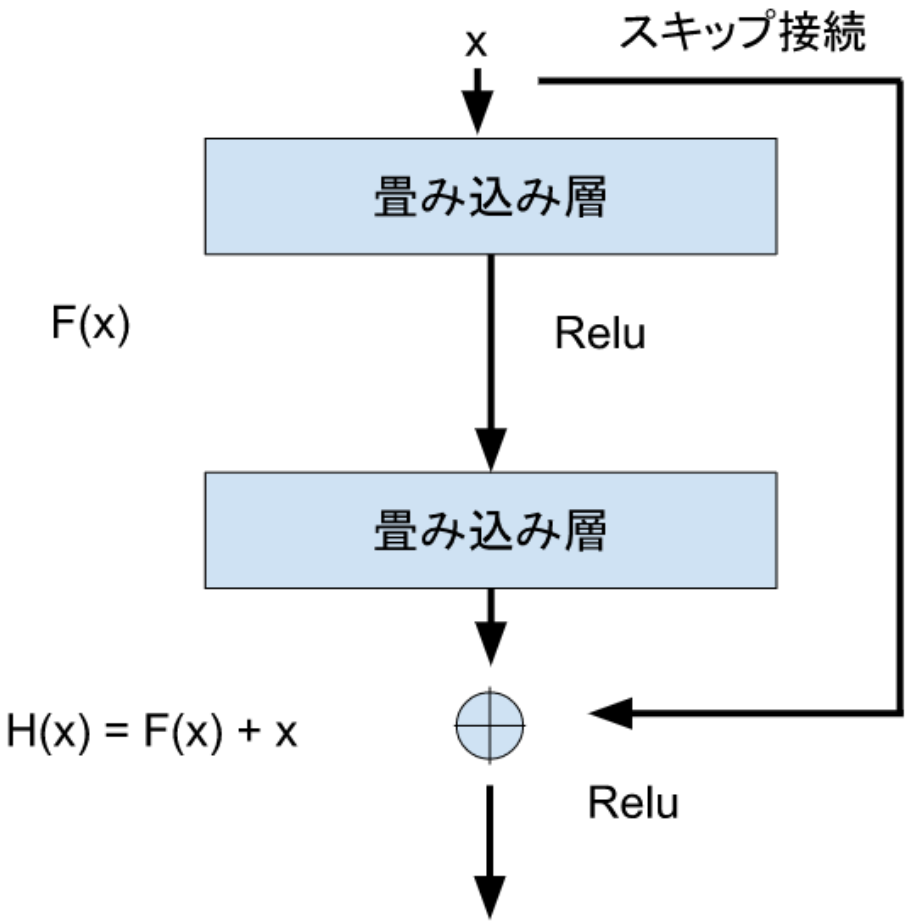

図 2. 4　残差ブロックの構造

2.4　敵対的生成ネットワーク

敵対的生成ネットワーク(Generative Adversarial Nets:GAN)とは Goodfellow らが提案したニューラルネットワークを用いた,画像生成モデルの一つである.GAN では訓練データに一致するようなデータを生成することを目的とした手法である.GAN の概略図を図 2.5 に示す.図のように GAN は画像を生成する Generator と,画像を識別する Discriminator の2つのネットワークを持っている.Generator は乱数を入力として受け取り,画像を生成する.Discriminator は訓練データと Generator が生成した画像を入力とし,それらが訓練データの画像であるか,Generator が生成した画像であるかの識別を行う.これら2つのモデルを交互に最適化していくことで最終的に,訓練データに似た画像を生成できるようになる.最適化する目的関数$V(G, D)$は次の式で表す.

$$\min_G \max_D V(G,D) = E_{x\sim P_{data(x)}}[\log D(x)] + E_{z\sim P_{noize(z)}}\left[\log\left(1 - D\left(G(z)\right)\right)\right] \quad (2.4)$$

ここでVは目的関数を示し,Dは Discriminator,Gは Generator を表す.また,z は乱数,xは訓練データの画像,P_{data}は訓練データの分布,P_{noize}は乱数の分布を表し,$D(x)$と$G(z)$は Discriminator と Generator の出力を表す.したがって,右辺の第 1 項目は Discriminator に訓練データxを入力したとき,Discriminator が訓練データ郡の画像であると推定する確率の期待値である.また,第 2 項目は Discriminator に Generator が生成する画像$G(z)$を入力したとき,Discriminator が生成画像であると推定する確率の期待値である. Generator は目的関数を最小化するように学習し, Discriminator は最大化するように交互に学習を行う.

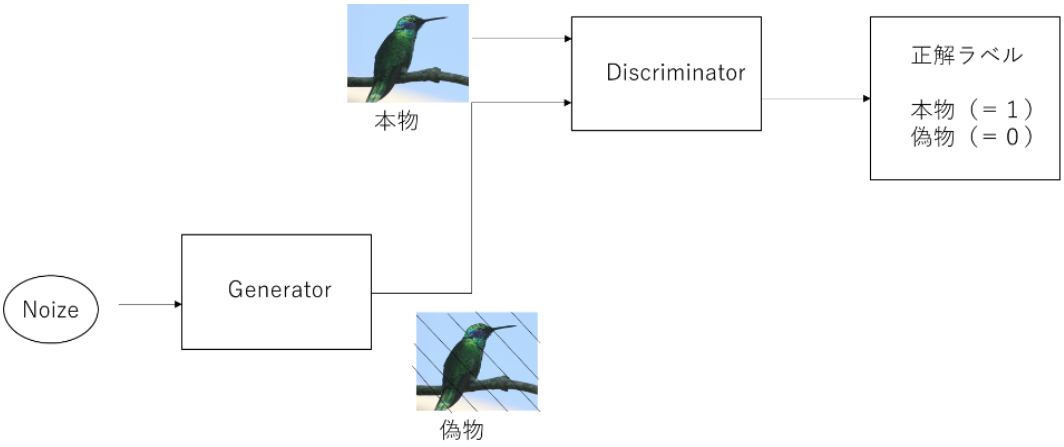

図 2. 5　GAN の構造

2.5 CartoonGAN

本研究で用いる生成モデルの一つである,Cartoo n GAN について説明を行う.CartoonGAN は GAN をベースに,風景写真をアニメ風のスタイルに変換する手法である.CartoonGAN では Generator の入力として風景写真の画像を使用し,Discriminator は,Generator が出力した画像とアニメ画像を入力として識別を行う.

CartoonGAN ではアニメ風の強調された輪郭線のある画像を生成するために,アニメ画像のデータセット$S_{data}(c)$から,輪郭線をぼかした$S_{data}(e)$のデータセットを作成する.Discriminator は $S_{data}(e)$を不正解データとして扱うことで,より輪郭線を強調した画像を生成できる.アニメ画像のデータセットの一部と,その輪郭線をぼかした画像を図 2.4 に示す.輪郭線のぼかしは以下の手順で行う.

1 Canny 法によるエッジ検出器で輪郭線ピクセルを検出[10]

2 エッジ領域を拡張

3 エッジ領域をガウス平滑化

図 2. 6　アニメ画像 $\boldsymbol{c_i}$(左)から輪郭線をぼかした画像$\boldsymbol{e_j}$(右)

©2013 プロジェクトラブライブ!

2.5.1 損失関数

CartoonGAN の損失関数$L(G,D)$は敵対的損失$L_{adv}(G,D)$)とコンテンツ損失$L_{con}(G,D)$の二つの損失で構成される.$L(G,D)$は次式で定義される.

$$L(G,D) = L_{adv}(G,D) + \omega L_{con}(G,D) \tag{2.5}$$

ここでωは二つの損失のバランスをとるための重みである.重みの値を大きくするほど,生成画像はもとの画像に近いものとなる.

2.5.1.1 敵対的損失

CartoonGAN で用いる二つの損失関数の一つ目は敵対的損失$L_{adv}(G,D)$である.これは風景写真をアニメ風に変換する役割を持っている.$L_{adv}(G,D)$は以下のように定義される.

$$L_{adv}(G,D) = E_{c_i \sim S_{data}(c)}[logD(c_i)] + E_{p_k \sim S_{data}(p)}\left[\log\left(1 - D\left(G(p_k)\right)\right)\right] + E_{e_j \sim S_{data}(e)}\left[\log\left(1 - D(e_j)\right)\right] \quad (2.6)$$

ここで, アニメ画像のデータセットを$S_{data}(c)$,アニメ画像の輪郭線をぼかしたデータセットを$S_{data}(e)$,風景写真のデータセットを$S_{data}(p)$とする.Discriminator はアニメ画像c_iと Generator は生成した画像$G(p_k)$を識別することで,Generator がアニメ風のテクスチャを持つ画像を生成するように導き,アニメ画像c_iと輪郭線をぼかした画像e_jの識別を行うことで,Generator がアニメ風の輪郭線を持つ画像を生成するように導く.

2.5.1.2 コンテンツ損失

コンテンツ損失$L_{con}(G,D)$は入力画像のコンテンツの同一性を保持するための損失関数である.事前に学習した VGG を使用し,VGG の深い層の特徴マップを使用する.コンテンツ損失は以下のように定義される.

$$L_{con}(G,D) = E_{p_i \sim S_{data}(p)}\left[\left\|VGG_l\left(G(p_i)\right) - VGG_l(p_i)\right\|_1\right] \quad (2.7)$$

ここでVGG_lはl層の出力である特徴マップを示す.また,損失の計算には平均絶対誤差を使用している.

2.6 AnimeGAN

AnimeGANは CartoonGAN を改良した手法で,より高性能に風景のアニメ化を行う.AnimeGAN では新たにグレースケールでのスタイル損失L_{gra},色再構成損失 L_{col}を提案している.また,敵対的損失L_{adv}には LSGAN[26]を採用している.AnimeGAN での損失関数は以下のように定義される.

$$L(G,D) = \omega_{adv}L_{adv}(G,D) + \omega_{con}L_{con}(G,D) + \omega_{gra}L_{gra}(G,D) + \omega_{col}L_{col}(G,D) \quad (2.8)$$

2.6.1 グレースケールスタイル損失

グレースケールスタイル損失では,より,アニメ的な質感を持つテクスチャと明確な輪郭線を持つ画像を生成するための損失である.コンテンツ損失L_{gra}は以下の式で定義される.

$$L_{gra}(G,D) = E_{p_i \sim S_{data}(p)}, E_{x_i \sim S_{data}(x)} \left[\left\| Gram\left(VGG_l\left(G(p_i)\right)\right) - Gram\left(VGG_l(x_i)\right) \right\|_1 \right] \quad (2.9)$$

ここで$S_{data}(x)$はアニメ画像のデータセットをグレースケール化したものである.

2.6.2 色再構成損失

CartoonGAN の生成画像の色は,アニメ画像側の影響を受けやすいため,元画像の色と大きくことなる部分が生まれてしまう.色再構成損失では,元画像である写真の色を再現するための損失である.色再構成損失では RGB 形式の画像を YUV 形式に変換して使用する.色再構成損失は以下の式で定義される.

$$L_{col}(G,D) = E_{p_i \sim S_{data}(p)}[\left\| Y\left(G(p_i)\right) - Y(p_i) \right\|_1 + \left\| Y\left(G(p_i)\right) - Y(p_i) \right\|_H + \left\| V\left(G(p_i)\right) - V(p_i) \right\|_H] \quad (2.10)$$

ここで$Y(p_i), U(p_i), V(p_i)$は YUV 形式の画像p_iの各チャンネルを表し,Hは Huber 損失を表す.

2.7 Deep Alignment Network

ここでは Deep Alignment Network(DAN)について説明する.DAN はカスケード形状回帰(CSR)に触発されている.CSR では,初期形状S_0を繰り返し更新することによって,顔のランドマークを予測する.初期形状は,学習に使用する顔画像のランドマークデータセットから各点の平均を計算したものを使用する.CSR の各反復による特徴付けは次式のようになる.

$$S_{t+1} = S_t + r_t\big(\Phi(I, S_t)\big)_1 \tag{2.11}$$

ここでS_tは繰り返しtでのランドマーク位置の推定値,r_tは画像Iからランドマーク位置で抽出された特徴量Φが与えられたときの,S_tの更新値を返す回帰関数である.特徴抽出法Φと下記方法r_tはいくつかある.例えば Supervised Descent Method(SDM) [DAN32]では SIFT[DAN19]特徴と単純な線形回帰分析を用いている.

DAN とCSR に基づくアプローチの違いは,DAN ではランドマーク周辺のパッチではなく,画像全体から特徴を抽出する点である.一般的に CSR ではランドマーク周辺を局所的に学習させる方が,画像全体から学習させるより精度が良くなる.DAN では各学習ステージでランドマークヒートマップという追加入力を導入することで実現される.ランドマークヒートマップは顔画像全体におけるランドマーク位置のステップごとの推定値を示し,この情報を各ステージ間で伝達する.

2.7.1 DAN の構造

DAN の概要を図 2.7 に示す.入力にはランドマーク推定値が,初期形状S_0と一致するように傾けた入力画像 I,ランドマークヒートマップH_t,前ステージ t-1 から生成される特徴画像F_tの 3 つを入力とする.最初の入力では,画像のみを入力に使用する.

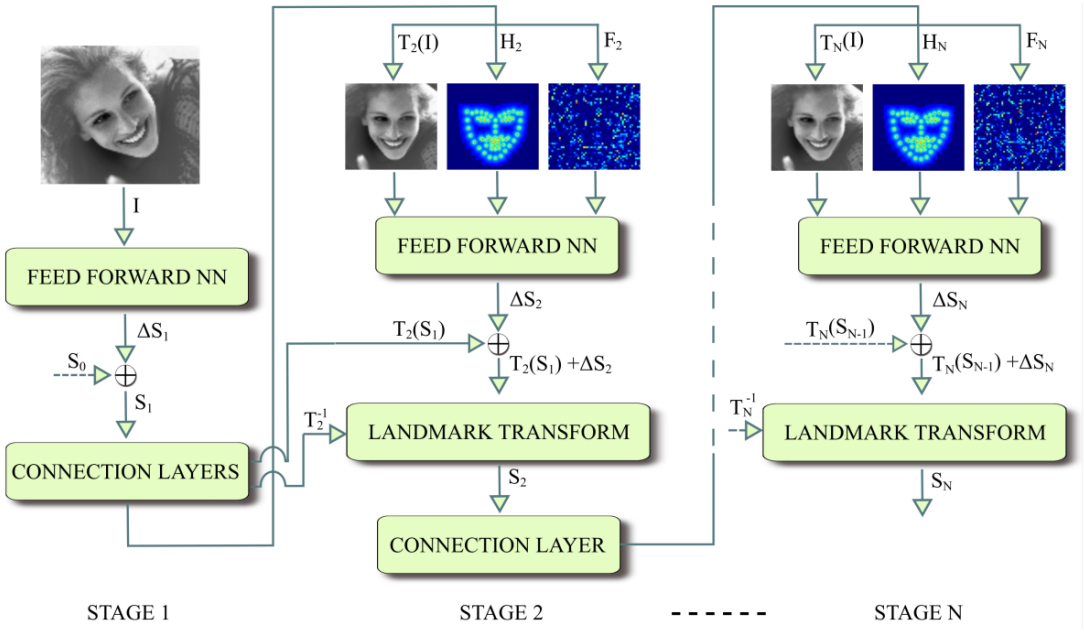

図 2. 7　DAN の概要図[6]

DAN のステージ 1 では,ランドマーク位置推定を行う順伝播型ニューラルネットワーク(Feedforward Neural Network:FNN)と次のステージへの入力を生成する接続層で構成される.接続層は,変換推定層,画像変換層,ランドマーク変換層,ヒートマップ生成層,特徴画像層から構成される.接続層の構成を図 2.8 に示す.変換T_{t+1}を生成する.この変換は入力画像Iと現在のランドマーク推定値S_tを,S_tが初期形状S_tに近くなるように移動させるために用いられる.変換されたランドマーク$T_{t+1}(S_t)$は,ヒートマップ生成層に渡される.逆変換T_{t+1}^{-1}は連続するステージの出力ランドマーク元の座標系に対応付けるために用いられる.

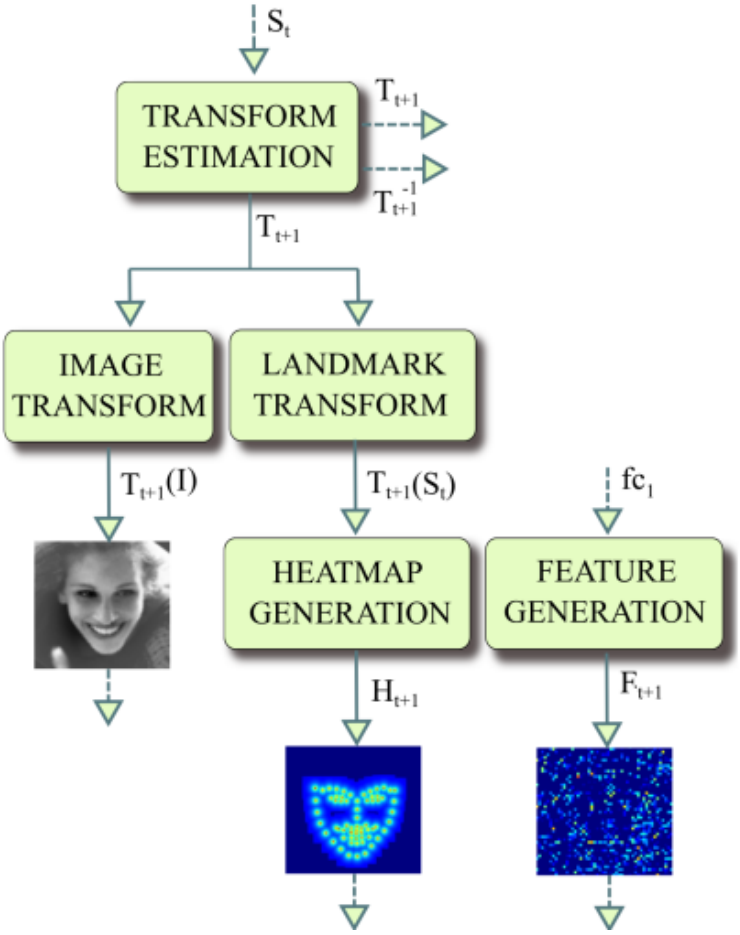

図 2. 8　DAN の接続層の構造[6]

2.7.2　順伝播型ニューラルネットワークの構造

DAN の各ステージのネットワーク構造を表 2.1 に示す.これは ImageNet IL SVRC2014 コンペティションで使用された VGG[8]を元に作成されている.max プーリング層と出力層を除いて,すべての層は活性化関数に ReLU を使用している.ドロップアウト層[13]は最初の全結合層の前に追加される.最後の層はランドマーク位置の現在の推定値に対する更新ΔS_tを出力する.

表 2.1　順伝播型ニューラルネットワークの構造[6]

	入力	出力	カーネル
畳み込み層	112×112×1	112×112×64	3×3×1
畳み込み層	112×112×64	112×112×64	3×3×64
プーリング層	112×112×64	56×56×64	2×2×1
畳み込み層	56×56×64	56×56×128	3×3×64
畳み込み層	56×56×128	56×56×128	3×3×128
プーリング層	56×56×128	28×28×128	2×2×1
畳み込み層	28×28×128	28×28×256	3×3×128
畳み込み層	28×28×256	28×28×256	3×3×256
プーリング層	28×28×256	14×14×256	2×2×1
畳み込み層	14×14×256	14×14×512	3×3×256
畳み込み層	14×14×512	14×14×512	3×3×512
プーリング層	14×14×512	7×7×512	2×2×1
全結合層 fc1	7×7×512	1×1×256	
全結合層 fc2	1×1×256	1×1×136	

2.7.3　標準的な形状への正規化

DAN では,入力画像Iは,ランドマークの現在の推定値と初期形状S_0と一致するように,各ステージで変換される.ここで初期形状S_0は,学習に使用するデータセットのランドマークの平均値である.これによって位置合わせタスクが簡略化され,精度が向上する.

T_{t+1}が推定されると,画像変換層とランドマーク変換層が,画像IとランドマークS_tを正規のポーズに変換する.画像はバイリニア補間法を用いて変形される.したがって,各ステージの出力は元の画像と一致するように変換し直している.DAN の任意の層での出力は以下のようになる.

$$S_t = T_t^{-1}(T_t(S_{t-1}) + \Delta S_t)_1 \tag{2.12}$$

ここで,ΔS_tはステージtにおける出力,T_t^{-1}は変換T_tの逆変換である.同様の正規化ステップはアフィン変換を使用している.

2.7.4 ランドマークヒートマップ

ランドマークヒートマップとは,ランドマークの位置で強度が最も強く,ランドマークからの距離に応じて減少する画像である.ランドマークヒートマップを使用することにより,畳み込みニューラルネットワークは前ステージで推定されたランドマークの位置を推論することができる.その結果,DAN は顔画像全体に基づいて顔の位置合わせを行うことができる.DAN ステージの入力では,ランドマークヒートマップが前ステージで生成されたランドマーク推定値に基づいて作成され,$T_t(S_{t-1})$に変換される.ヒートマップは以下の式で生成される.

$$H(x,y) = \frac{1}{1 + min_{s_i \in T_t(S_{t-1})} \|(x,y) - s_i\|} \tag{2.13}$$

ここでHはヒートマップ画像,s_iは$T_t(S_{t-1})$のi番目のランドマーク,x, yはランドマークの座標である.

2.7.5 特徴画像層

特徴画像層では前層の全結合層 fc1 から生成される画像を入力とし,3136 個のノードに全結合される.このノードが ReLU の活性化関数を持ち,出力は 56×56 の画像として出力し次のステージへ接続される.この接続により,前ステージで学習した情報を連続するステージに転送することができる.

2.7.6 学習手順

DAN では 1 ステージずつ学習が行われる.第 1 ステージでは誤差が改善されなくなるまで学習を行い,その後,接続層と第 2 ステージが追加され学習が行われる.DAN で使用する誤差には,瞳孔間の距離によって正規化されたランドマークの位置を使用します.これは次式で表される.

$$\min_{\Delta S_i} \frac{\|T_t^{-1}(T_t(S_{t-1}) + \Delta S_t) - S^*\|}{d_{ipd}} \tag{2.14}$$

ここでS^*は学習データのランドマーク,T_tはステージtの入力画像と形状を変換,d_{ipd}はS^*の瞳孔間の距離である.

2.8 活性化関数

ここでは,これまでに説明したCNNやGANなどで一般的に使用される,主な活性関数を説明する.

2.8.1 ReLU

ReLU(Rectified Linear Unit)はVGGなどの深いネットワークモデルにおいて,勾配消失を回避するために使用される.ReLUは次式によって表される.

$$f(x) = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases} \tag{2.15}$$

ReLUでは式のように0以下では0を出力し,1より大きい場合は入力をそのまま出力する関数である.ReLU 関数を横軸を入力,縦軸を出力とするとしてグラフに描画すると図 2.9のようになる.ReLU関数は主に中間層で使用される.

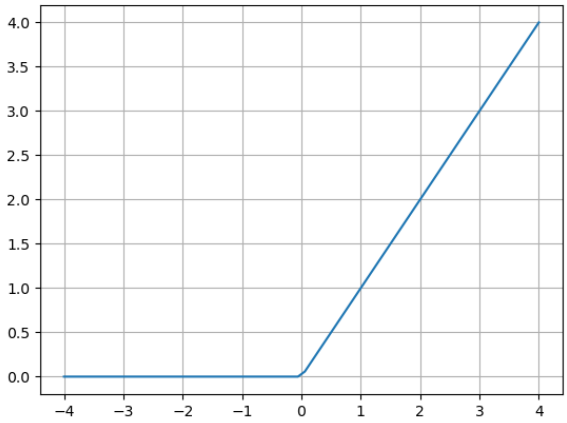

図 2. 9　ReLU関数のグラフ

2.8.2 Leaky ReLU

ReLU では,常に同じ値を出力し続けるようになってしまう場合がある.これは,x が負の数のとき ReLU は 0 のみ出力するからである.この問題を解決するための活性化関数が Leaky ReLU(Leaky Rectified Linear Unit)である.Leaky ReLU は次の式で表す

.

$$f(x) = \begin{cases} x & (x > 0) \\ ax & (x \leq 0) \end{cases} \tag{2.16}$$

ここでaはパラメータであり,$a = 0.1$とした場合の Leaky ReLU を図 2.10 にグラフで表す.

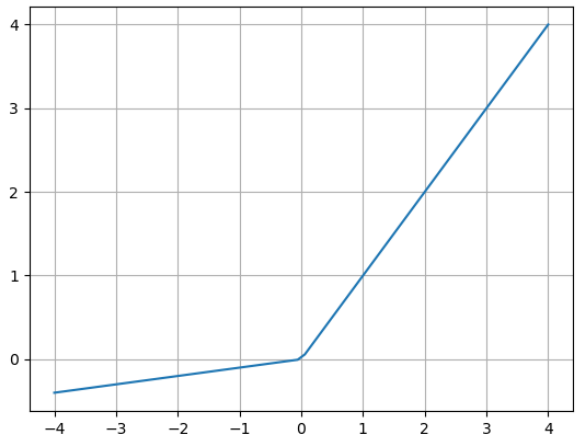

図 2. 10　Leaky ReLU 関数のグラフ

2.8.3 シグモイド関数

シグモイド関数は CNN の出力層や GAN の Discriminator の出力層などで使用される.シグモイド関数は次式によって定義される.

$$f(x) = \frac{1}{1 + e^{x}} \tag{2.17}$$

また,図 2.5 にシグモイド関数をグラフで表す.図のように 1 で収束し,入力が小さい程 0 に収束する.

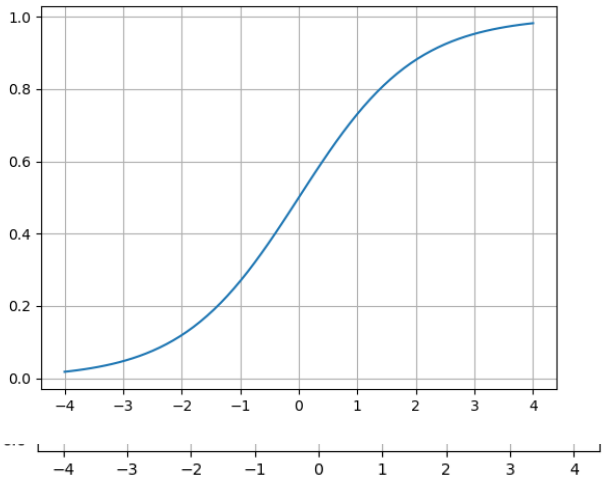

図 2. 11　シグモイド関数のグラフ

2.8.4　ソフトマックス関数

ソフトマックス関数は,シグモイド関数と同じく主に出力層に用いられる活性関数である.ソフトマックス関数は次式で表す.

$$f(x_i) = \frac{e^{x_i}}{\sum_{k=1}^{n} e^{x_k}} \tag{2.18}$$

ソフトマックス関数の出力は 0 から 1 の間であり,出力の総和は 1 となる.この性質によって,分類問題などにおいては出力を,そのクラスである確率として表すことができる.

2.9　損失関数

ここでは CNN や GAN などで使用される一般的な損失関数について説明する.損失関数は,学習データとネットワークの出力の差を計算する関数である.

2.9.1 平均絶対誤差

平均絶対誤差（Mean Absolute Error:MAE)は L1 損失とも呼ばれ,CartoonGAN や AnimeGAN などのコンテンツ損失の計算などに使用されている.L1 損失は外れ値が含まれるデータセットに対しても安定した予測結果を出力することができる.L1 損失は以下の式で表す.

$$MAE(y_i, \hat{y_i}) = \frac{1}{n}|\hat{y_i} - y_i| \tag{2.19}$$

式のように予測値y_iと正解値$\hat{y_i}$の差の平均をとったものが L1 損失である.

2.9.2 平均二乗誤差

平均二乗誤差(Mean Squared Error:MSE)は L2 損失とも呼ばれ,AnimeGAN の敵対的損失の計算などに使用される.L2 損失は L1 損失と比べ,誤差が 0 に近い値だと滑らかに変化する.L2 損失は以下の式で表す.

$$MSE(y_i, \hat{y_i}) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y_i} - y_i)^2 \tag{2.20}$$

式のように L2 損失は誤差の二乗であり,大きい誤差をより大きく表現している.

2.9.3 Huber 損失

Huber 損失はSmooth L1 損失とも呼び,AnimeGAN の色再構成損失の計算で使用される.Huber 損失は指定したパラメータ δ の範囲内では二乗の計算を行い,範囲外では絶対艇を使った計算を行う.Huber 損失は次の式で定義される.

$$Huber(a) = \begin{cases} \frac{1}{2}a^2 & |a| \le \delta \\ \delta\left(|a| - \frac{1}{2}\delta\right) & |a| > \delta \end{cases} \tag{2.21}$$

ここでaは予測値と正解値の差である.Huber 損失は L1 損失と L2 損失のいくつかの欠点を克服した損失である.L1 損失では誤差が 0 の場合に微分ができず,0 に近い値でも勾配が大きいという欠点がある.また,L2 損失には外れ値の影響を受けやすいという欠点をもつ.Huber 損失ではパラメータδの範囲内では,L2 損失と同様に滑らかに変化し,範囲外では L1 損失のように直線的に変化する.

2.9.4 交差エントロピー誤差

交差エントロピー誤差(Cross Entropy Loss)は VGG などの多クラス分類などのタスクで使用される.正解ラベル ti と予測確率 yi とすると交差エントロピー誤差は次式で表す.

$$H(p.q) = -\sum_{i-1} t_i log y_i \tag{2.22}$$

クラス分類において正解ラベルt_iは one-hot 表現で表され,正解クラスのみ 1,それ以外は 0 で構成される.そのため,計算は正解クラスの時のみ計算される.また,$log y_i$は予測確立y_iが 1 に近い値であるほど,0 に近い値を返す.したがって,y_iが$t_i = 1$に近いほど誤差が小さくなる.

2.9.5 バイナリ交差エントロピー

バイナリ交差エントロピーは主に 2 クラス分類の場合に使用され,CartoonGAN の敵対的損失の計算などに使用される.バイナリ交差エントロピーは次の式で表す.

$$H(p,q) = -t_i log y_i - (1 - t_i) log(1 - y_i) \tag{2.23}$$

バイナリ交差エントロピーでは交差エントロピーと異なり,t_i が 0 のときも計算が行われる.したがって,t_i が 1 の時のy_iの値が 1 に近いほど,また,t_i が 0 のときのy_iの値が,0 に近いほど誤差が小さくなる.

2.10 最適化アルゴリズム

ここでは CNN や GAN などで使用される一般的な最適化アルゴリズムについて説明する.最適化アルゴリズムは損失を最小化するのに用いられるアルゴリズムである.最適化アルゴリズムは様々な手法が提案されているが,本研究でも使用するいくつかの最適化アルゴリズムについて説明する.

2.10.1 確率的勾配降下法

確率的勾配降下法(stochastic gradient descent:SGD)は,局所最適解への収束という問題点を,重みの更新ごとにランダムにサンプルを選び出すことで解消した手法である.SGDは次式のように重みの更新を行っていく.

$$w_{t+1} = w_t - \eta \frac{\partial L}{\partial w_t} \tag{2.24}$$

ここで重みをw,学習係数をη,損失関数をLと表す.学習係数ηはモデルに合わせて決定する必要がある.

2.10.2 MomentumSGD

SGD では,勾配が最も大きい方向に重みベクトルを更新していく.そのため,学習が進むと,最小値付近の勾配の小さい地点で振動が起こり,学習が不安定になる場合がある.MomentumSGD では現在の勾配に過去の勾配を加えることで振動を抑える.また,過去の勾配を加えることで初期の学習を加速させることができる.MomentumSGD の更新式は次のようになる.

$$v_t = \alpha v_{t-1} - \eta \frac{\partial L}{\partial w_t} \tag{2.24}$$

$$w_{t+1} = w_t + v_t \tag{2.25}$$

ここでvを Momentum(慣性)と呼ぶ.

2.10.3 AdaGrad

AdaGrad[23]は,学習の進行にあわせて学習係数を調整していく手法である.学習係数の更新と重みの更新は次式のように行う.

$$h_0 = \varepsilon \tag{2.26}$$

$$h_t = h_{t-1} + \left(\frac{\partial L}{\partial w_t}\right)^2 \tag{2.27}$$

$$\eta_t = \frac{\eta_0}{\sqrt{h_t}} \tag{2.28}$$

$$w_{t+1} = w_t - \eta_t \frac{\partial L}{\partial w_t} \tag{2.29}$$

ここで初期学習係数η_0と,パラメータεはモデルに合わせて設定する必要がある.

2.10.4 RMSProp

RMSProp は AdaGrad を改良した手法であり,勾配の二乗ではなく,勾配の二乗の移動平均をとるように変更したものである.重みの更新は次式のように行う.

$$h_t = \alpha h_{t-1} + (1-\alpha)\left(\frac{\partial L}{\partial w_t}\right)^2 \tag{2.30}$$

$$\eta_t = \frac{\eta_0}{\sqrt{h_t + \varepsilon}} \tag{2.31}$$

$$w_{t+1} = w_t - \eta_t \frac{\partial L}{\partial w_t} \tag{2.32}$$

ここで$t-1$のときh=0 である.パラメータ α によって過去の勾配の影響を抑え,現在の勾配ベクトル$\left(\frac{\partial L}{\partial w_t}\right)^2$を優先して反映させる.

2.10.5 Adam

Adam は Momentum SGD と RMSProp を組み合わせた手法である.Adam では移動平均と二乗の移動平均をそれぞれm_t, v_tと定義する.m_tと v_tは次式のようになる.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\frac{\partial L}{\partial w_t} \tag{2.33}$$

$$v_t = \beta_2 m_{t-1} + (1-\beta_2)\left(\frac{\partial L}{\partial w_t}\right)^2 \tag{2.34}$$

ここでβ_1, β_2は RMSProp のパラメータ α αと同様のものである.m_t, v_tは偏差を含むため,次式のように補正を行う.

$$\hat{m}_t = \frac{m_t}{1-\beta_2^t} \tag{2.35}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{2.36}$$

したがって重みの更新は次式のように行われる.

$$w_{t+1} = w_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \varepsilon}} \tag{2.37}$$

第 3 章　提案手法

3.1　ランドマーク損失

実写の顔画像をアニメ風に変換するための新たな損失関数を提案する．一般に画像のアニメ風への変換は困難である.困難な理由としては第 1 章でも説明したように以下 5 つの要因が考えられる.

1.はっきりとしたエッジ
2.なめらかなテクスチャ
3.実写と比べて少ない線
4.表情がわかりやすいように誇張されたパーツ(目,口)
5.単純化されたパーツ(鼻など)

風景画像を対象とした CartoonGAN や AnimeGAN では 1~3 の問題について効果的な結果を示している.4 と 5 の問題を解決するために提案する新しい損失関数をランドマーク損失と呼ぶ.ランドマーク損失は,風景画像を対象としている CartoonGAN と AnimeGAN のモデルに新たに追加し,顔画像を対象としてアニメキャラクターの顔特徴を持つ画像を生成する.ランドマーク損失では Generator が変換した画像の顔パーツのランドマークを検出する.その後,検出したランドマーク座標と,変換前のランドマークの位置の誤差を最小化することで,元画像に近く,座標にランドマーク検出器が検出できるアニメキャラクターの顔パーツを配置できるように学習を導く.入力前の画像ランドマーク検出にはあらかじめ Dlib[17]を使用し検出する.Generator の出力した画像のランドマーク検出にはアニメ画像を使用して学習を行った DAN を使用する.誤差の計算には,ランドマークを用いた学習で優れた結果を残している Wing Loss[14]を使用する.GAN 全体の損失とランドマーク損失の説明は 3.5 節で行う.

3.2　ランドマークモデル

ランドマークモデルには Marco らの手法[15]をもとに iBUG[16]を採用した.しかし,人の顔よりも簡易化されたアニメキャラクターの顔では,人の顔のすべてのランドマークをそのまま使用することは難しい.そのため,いくつかのランドマークを削除して使用する.アニメではキャラクターの鼻や口は簡略化され点や線で表すのが一般的である.また,今回の実験では,実写真の顔画像とアニメキャラクターのランドマーク座標の比較を行う.しかし,アニメ

キャラクターと実写の顔画像では目の大きさが大きく異なる.アニメキャラクターの特徴の一つでもある,大きな目を持つ画像を生成するためには,実写の顔画像のランドマーク座標をそのまま使うことはできない.そこで,アニメ画像のデータセットの平均的な目の大きさに合わせて,下まぶたの 3 点のランドマークの位置を 8.8 ピクセル下げて使用する.以上をもとに,今回の手法でのランドマークモデルは,眉毛 :各 3 個,目:各 6 個,鼻 1 個,口 4 個,輪郭 17 個の 40 個のランドマークを使用する.図 3.1 に,40 個のランドマークを描画した顔画像とアニメキャラクター画像を示す.

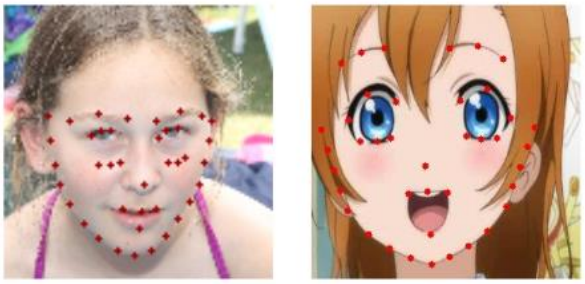

図 3. 1　新しいランドマークモデルの顔画像とキャラクター画像

3.3　アニメキャラクターの顔検出器

Generator が生成した画像のランドマークを検出するためのランドマーク検出器について説明する.

アニメキャラクターの顔検出やランドマーク検出には Dlib[17],や OpenCV[18]をもとにした優れた検出器がいくつか存在するが,既存のものでは GAN の学習過程で生成する画像を顔と認識しない.また,一枚の顔画像で複数の顔を検出する場合などもあり,一定の出力を得ることが難しい.

そこで Deep Alignment Network(DAN)で新たにランドマーク検出器を作成する.DAN では顔画像全体から特徴を抽出し,学習データのランドマークの平均から位置合わせをしていく手法である.学習データの特徴を全く持たない画像でも学習データの平均値を出力し,アニメキャラクターの画像で学習した検出器では実写の顔画像で正確なランドマークを検出しない.したがって,今回の手法に適した検出器である.

3.4 Wing Loss

Wing Loss は Zhen らが提案した顔のランドマーク検出分野での損失関数である.ランドマーク検出分野での損失関数にはL1(平均絶対誤差)やL2(平均二条誤差)が使われているが,それぞれ問題がある.L1 では傾きが 1 で一定のため,大きく誤ったランドマークに引っ張られやすく,L2 も外れ値に大きく値を増加させてしまう.Wing Loss では小程度の誤差には大きく増加し,そこからなだらかな傾きに変わる.式 3.1 に Wing Loss を示す.ここでwは非線形部分の範囲を設定する定数であり, εはである非線形領域の曲率を定める定数である.また,$C = w\ln(1 + |x|/\varepsilon)$であり,線形部分と非線形部分を滑らかに繋ぐ定数である.

$$wing(x) = \begin{cases} w\ln(1 + |x|/\varepsilon) & if\ |x| < w \\ |x| - C & otherwise \end{cases} \tag{3.1}$$

また, $\varepsilon = 2.0, w = 5,10$の時の Wing Loss とL1,L2 損失を,横軸を入力x,縦軸を損失関数の出力として図 3.2 に示す.図のように他の損失関数と比べて,中程度の誤差で値が大きくなり,外れ値と比較しても小さくなりすぎない損失関数になっている.

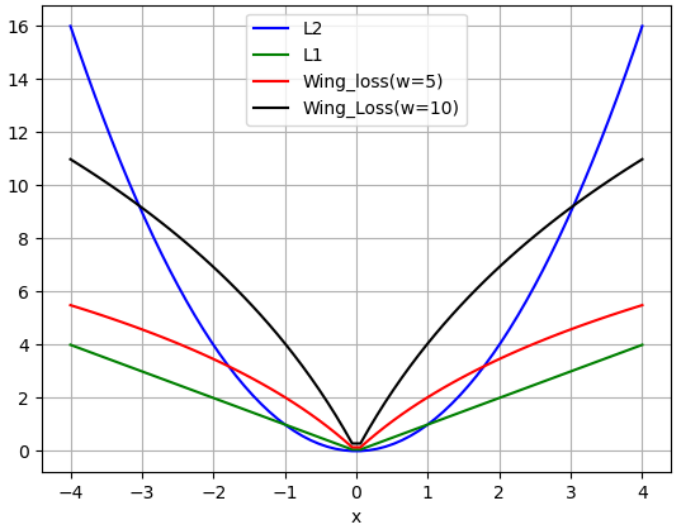

図 3. 2　誤差$\boldsymbol{x}$に対する損失関数の出力

3.5 損失関数

提案手法で使用する複数の損失関数について説明する.提案手法ではCartoonGAN,AnimeGAN の手法をもとに,新たにランドマーク損失を加えたものである.ランドマーク損失を$L_{land}(G,D)$,ランドマーク損失の重みをω_{land}とすると CartoonGAN での全体の損失関数は次式で表す.

$$CartoonL(G,D) = \omega_{adv}L_{adv}(G,D) + \omega_{con}L_{con}(G,D)+\omega_{land}L_{land}(G,D) \tag{3.2}$$

ここで敵対的損失L_{adv}は Generator と Discriminator に適用され,コンテンツ損失L_{con}とランドマーク損失L_{land}は Generator のみに使用される.

次に AnimeGAN にランドマーク損失を加えると AnimeGAN 全体の損失関数は次式のようになる.

$$\begin{aligned} AnimeL(G,D) = {} & \omega_{adv}L_{adv}(G,D) + \omega_{con}L_{con}(G,D) \\ & +\omega_{gra}L_{gra}(G,D)+\omega_{col}L_{col}(G,D) + \omega_{land}L_{land}(G,D) \end{aligned} \tag{3.3}$$

ここで敵対的損失L_{adv}は Generator と Discriminator に適用され,コンテンツ損失L_{con},グレースケールスタイル損失L_{gra},色再構成損失 L_{col},ランドマーク損失L_{land}は Generator のみに使用される.

ランドマーク損失L_{land}は次の式で表す.

$$L_{land}(G,D) = E_{p_i \sim S_{data}(p)}\left[\left\| DAN\left(G(p_i)\right) - land_{p_i}\right\|_w\right] \tag{3.4}$$

ここでDANはアニメキャラクターのランドマーク検出器を表し,$DAN\left(G(p_i)\right)$ は Generator が入力p_iから出力した画像のランドマーク座標を表す.また$land_{p_i}$は入力画像p_iのランドマーク座標を表す.

第 4 章　アニメキャラクターの顔のランドマーク検出器の作成

4 章では GAN の生成画像のランドマーク検出を行うのに使用するランドマーク検出器のトレーニングを行った.ランドマーク検出器には顔画像や漫画の顔画像のランドマーク検出で高い性能を発揮している Deep Alignment Network を使用した.

4.1　データセットの作成

データセットには,アニメ「ラブライブ」シーズン 1 とシーズン 2 の動画から切り出した 10512 枚の画像を使用した.続いて 10512 枚の画像から mmdetection [19]と mmpose[20]を使用したアニメキャラクターの顔検出器[21]を使用し,5697 枚の顔画像と 28 点の顔のランドマーク座標を抽出した.しかし,このランドマークモデルでは輪郭が 5 点しかなく 3 章で述べたランドマークモデルには合わないため,簡単な操作と手動で輪郭を 17 点,合計で 40 点のランドマークに拡張を行った.以下にそのアルゴリズムを示す.

まず,顔画像に対して平滑化と二値化を行い,輪郭線を強調させる.次に,5 点のランドマークの中点をとり,そこから画像の高さ方向に輪郭線の探索を行う.検出できた場合はその点を新たなランドマークとする.この操作をもう一度繰り返し,17 点に拡張する.影や陰影などの影響でうまく行かなかった画像に対しては,手動で行った.拡張した 40 点のランドマーク座標はテキストで保存し使用する.

このようにして取得した 5697 枚のアニメキャラクターの顔画像とランドマークを使用し,Deep Alignment Network を学習させる.また,アニメキャラクターの顔画像はあらかじめグレースケールに変換して使用する。ネットワークの学習には 4000 枚,テストには 1140 枚,残り 557 枚を検証用に使用する.

4.2　学習条件

使用する Deep Alignment Network モデルは 2.7 節で説明したものを使用する.最適化には,初期ステップサイズ 0.001,ミニバッチサイズ 64 の Adam を使用した.また,論文では瞳孔間距離によって正規化されたランドマーク座標を損失として用いているが,今回使用するデータセットには瞳孔のランドマークは存在しない.そのため,ユークリッド距離と二乗平均平方根誤差(RMSE)を使用する[22].予測値pのランドマーク座標(x_p, y_p)と元データのランドマーク座標(x_d, y_d)とするとユークリッド距離は次の式で表す.

$$d_i = \sqrt{\left(x_d - x_p\right)^2 + \left(y_d - y_p\right)^2} \tag{4.1}$$

また,RMSE は次式のようになる.

$$RMSE = \sqrt{\frac{1}{Q}\sum_{n=1}^{Q}\left(x_d - x_p\right)_n^2 + \left(y_d - y_p\right)_n^2} \tag{4.2}$$

ここで,Q はランドマークの要素数であり,実験では 40 個のランドマークを使用するので実際の値は 40 である.

DAN は,2.7 節で説明したように,ステージごとに分けて学習を行う.今回の実験では,2 ステージまで 100 エポックずつ訓練を行った.

4.3 実験環境

本実験で使用したライブラリ等の実験環境を以下に示す.実装はすべて python で行い DAN モデルの実装には Tensorflow を使用した.また,モデルの学習には NVIDIA GeForce RTX 2080 Ti, Intel(R) Core(TM) i7-7700 3.6GHz CPU, メモリ 16GB,SSD サイズ 7.5TB で行った。

・Ubuntu 22.04

・python 3.7.15

・Numpy 1.18.5

・Tensorflow-gpu 1.15.0

・opencv 4.6.0

4.4 実験結果

図 4.1 に検証用データセットの画像を使用し,正解データを描画したものと,DAN が出力した結果のランドマークを描画したものを示す.同一画像で,左側は正解データであり,右側は DAN が出力した結果である.正面を向いた画像以外にも,横を向いた画像などでも正確に描画できていることがわかる.また,ユークリッド距離誤差が 5 以上のものを外れ値

とし,検証データでの RMSE,外れ値の数とその割合,外れ値を持つ画像枚数とその割合を表 4.1 に示す.表より,検証用データ全体の RMSE は 1.62 であった.また,外れ値の割合は 3.17%であったが,外れ値を持つ画像の割合は 31.0%であった.これは,眉毛が髪の毛で隠れている画像が多いためであると考えられる.その他の外れ値を持つ画像の例を図 4.2 に示す.図のように,顔の輪郭が髪の毛以外のもので隠されている場合などでは検出が難しい.

表 4.1　ランドマーク検出器の検証結果

	RMSE	外れ値である座標	外れ値の割合(%)	外れ値を持つ画像(枚)	外れ値を持つ画像の割合(%)
検証結果	1.62	707	3.17	179	31.0

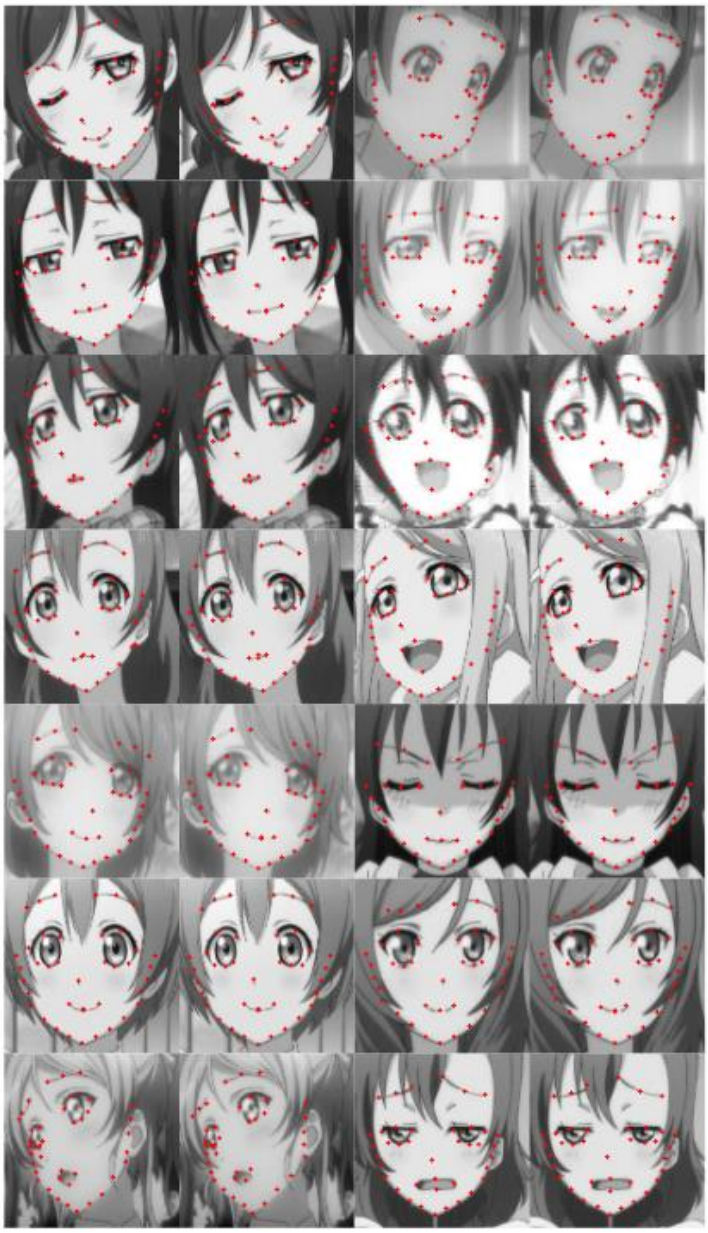

図 4.1　検証用データ(左)と検証結果(右)

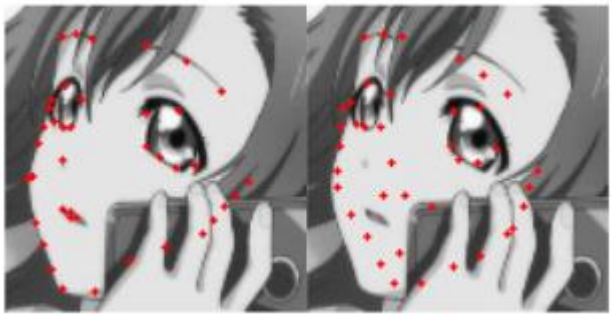

図 4. 2　誤差の大きい画像

第 5 章　ランドマーク損失による GAN の作成

5.1　概要

提案するランドマーク損失を使用し,GAN の学習と画像の生成を行う.ランドマーク損失は CartoonGAN と AnimeGAN に新たな損失として追加し,各モデルを学習させる.また,生成画像のランドマーク検出器に使用するのは,4 章で作成した DAN によるアニメキャラクターのランドマーク検出器である.この検出器の性質として以下のものが挙げられる.

1. 画像全体から特徴を抽出
2. アニメキャラクターの顔が持つ特徴によってランドマークの位置合わせが行われる.
3. ノイズのような画像（生成器のトレーニングが不十分な状態）に対しては,DAN の訓練データの平均値を返す.

これらの性質を利用することで,正しくランドマークを検出された画像はアニメキャラクターの顔の特徴を持つ画像であると言える.したがって,各パーツの位置合わせ以外だけではなく,各パーツの変換にも良い影響を与えられると考えられる.

実験では CartoonGANと AnimeGAN のモデルに,新たにランドマーク検出器を追加して画像生成を行い,生成した画像を比べることで,ランドマーク損失の有用性を検証する.

5.2　データ

まず,アニメキャラクターの顔画像は,4 章でも使用した 5697 枚の画像を使用した.Generator の入力となる,実写の顔画像には NVIDIA によって公開されている Flicker Face Dataset の中から 6000 枚を使用した.画像のサイズはアニメ,実写ともに 112×112 に揃えて使用する.

ランドマーク損失で使用する実写の顔画像のランドマークは,事前に Dlib 検出器を用いて検出し,3 章で提案した 40 個のランドマークモデルに合わせている.

5.3 CartoonGAN,AnimeGAN のネットワークモデル

CartoonGAN と AnimeGAN では Generator,Discriminator とコンテンツ損失用の VGG19 の 3 つネットワークを使用している.CartoonGAN の Generator の構成を図 5.1 に示す.ここで Conv は畳み込み層,Norm はインスタンス正規化層,SC はスキップ接続を表し,k はカーネルサイズ,n は特徴マップの数,s は畳み込み層のストライドを表す.図のように Generator は 2 回のダウンサンプリングを経て 8 つの残差ブロックに接続される.残差ブロックの後,2 回のアップサンプリングで画像に戻される. 次に Discriminator の構成を図 5.2 に示す.Discriminator では VGG のようなクラス分類ではなく,入力画像がアニメであるかの判断を行うだけであるため浅い層で構成される.また,活性化関数は Generator と異なり,パラメータ a=0.2 の Leaky ReLu を使用する.

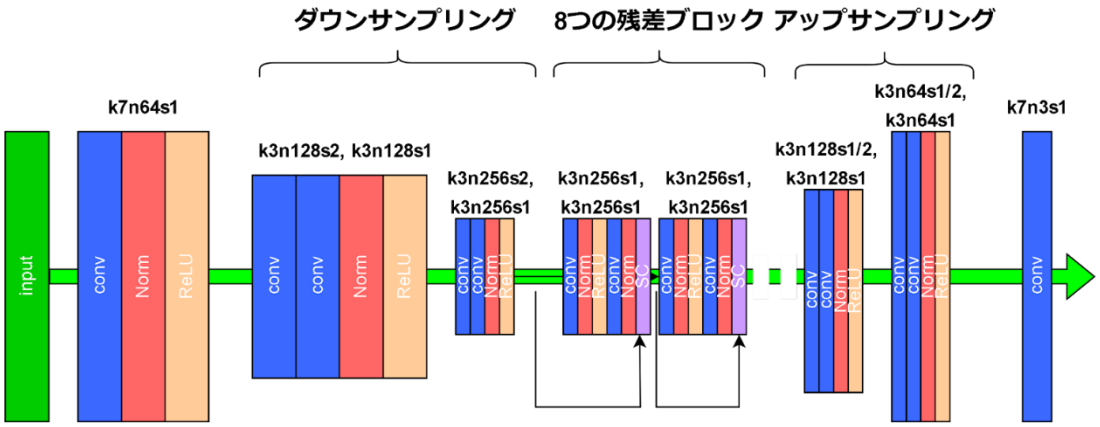

図 5. 1　Generator のネットワーク構成

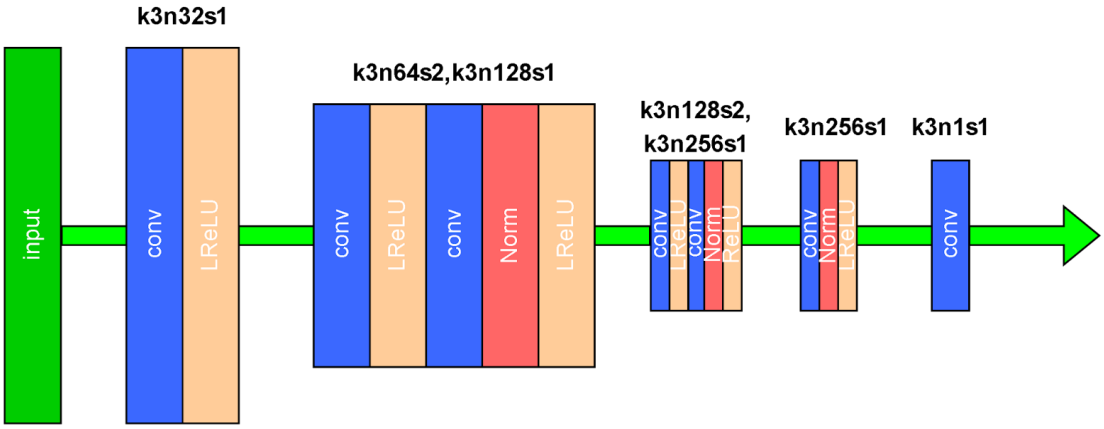

図 5. 2　Discriminator のネットワーク構成

5.4 モデルのトレーニング

顔画像からアニメキャラクターへの変換は,ペアリングされていないデータを用いて行うため,ランダムに初期化を行うと最適化は極小値で簡単に捕まってしまう.そこで,Generator は最初の 4 エポックはコンテンツ損失のみで学習を行う.

GAN は 100 エポックまでトレーニングを行い,Generator の学習率は 4 エポックまでは 0.0001,それ以降は 0.00008,Discriminator の学習率は 0.00016 とする.損失の最小化には Adam optimizer を使用する.各損失のバランスを取るための重みは論文を参考に CartoonGAN では ω_{adv} =1.0, ω_{con} =1.5, ω_{land} =0.5,AnimeGAN では ω_{adv}=300,ω_{con}=1.5,ω_{gra}=3.0,ω_{col} =10.0,ω_{land}=0.5 で行い,ランドマーク損失を計算するための WingLoss のパラメータはw=5.0,ε=2.0 を使用する.また,DAN によるランドマーク検出はグレースケール画像で行うため,Generator によって生成された画像はグレースケールに変換されてランドマーク検出器に入力される.

5.5 実験環境

本実験で使用したライブラリ等の実験環境を以下に示す.実装はすべて python で行い CartoonGAN,AnimeGAN のモデルの実装には Pytorch を使用した.また,モデルの学習には NVIDIA GeForce RTX 2080 Ti, Intel(R) Core(TM) i7-7700 3.6GHz CPU, メモリ 16GB,SSD サイズ 7.5TB で行った。

・Ubuntu 22.04
・python 3.7.15
・Numpy 1.18.5
・Pytorch1.4.0
・opencv 4.6.0
・Tensorflow-gpu 1.15.0

5.6 結果

5.2 節のデータセットで学習を行った結果を図 5.3 に示す.図より提案手法であるランドマーク損失を AnimeGAN に加えた場合,顔領域のテクスチャ変換が行えていることがわかる.また,目に関してもアニメキャラクターのものに変換できている.また,口や鼻なども簡易化されていることがわかる.しかし,CartoonGAN にランドマーク損失を加えたものは,す

べての画像でほぼ同じ出力しかされておらず,学習が失敗している.また,風景画像を対象としている CartoonGAN のみの実験では,入力画像と同じ画像が出力されてしまっている.しかし,同じく風景画像を対象としている AnimeGAN ではテクスチャの変換ができており,また,目などの変形も見られる.これは,敵対的損失に対してコンテンツ損失の重みが強すぎるためと考えられる.

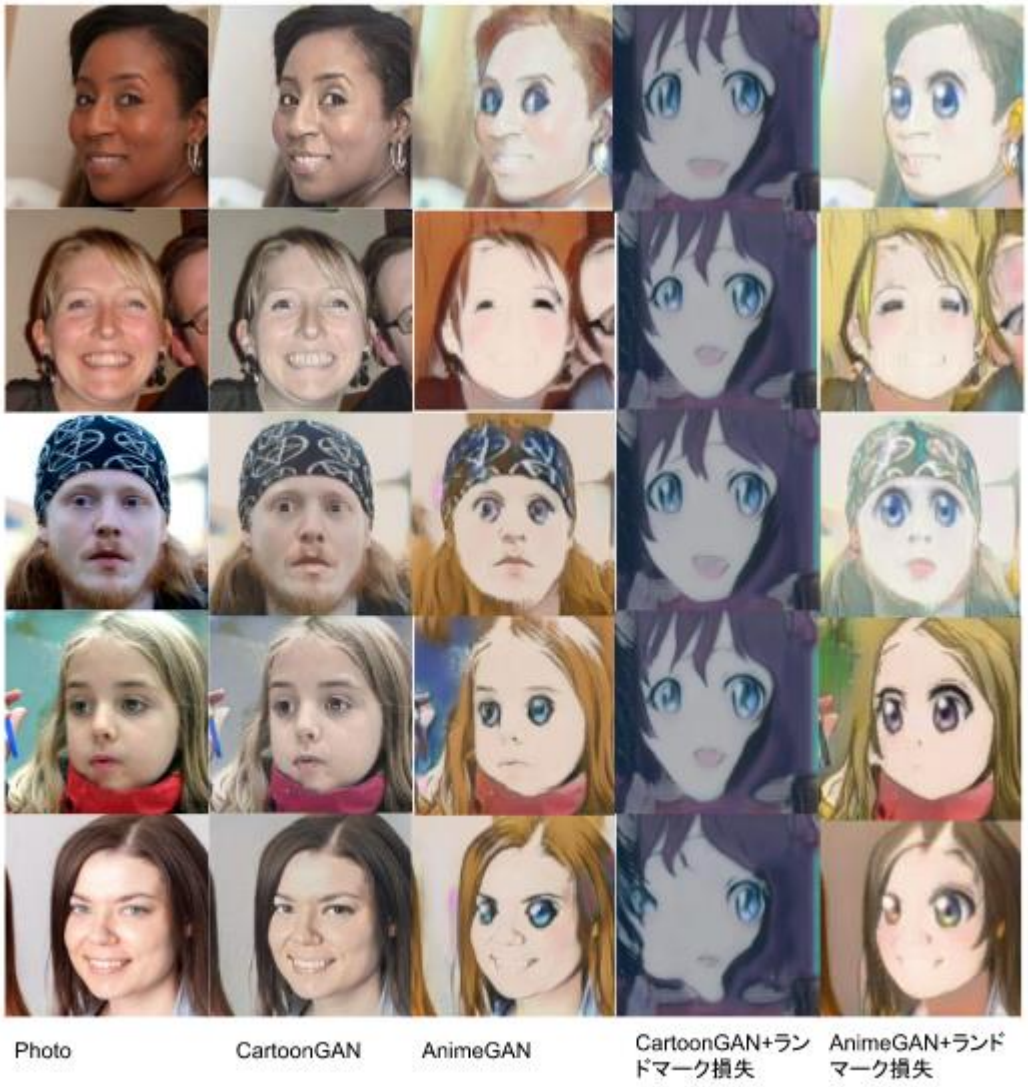

図 5. 3　画像生成結果

次にランドマーク損失が画像生成に影響を与えることを示すために,生成画像についてランドマークの検出を行う.検出結果を図 5.4 に示す.左が元画像,真ん中が元画像をランドマーク検出器で検出したもの,右の画像が変換後の画像をランドマーク検出器で検出したものである.比較すると変換後の画像の方が,ランドマークが顔の付近に密集し,顔のよう

な形を形成していることがわかる.したがって,実写の顔画像がアニメ画像に近づくほど損失は小さくなっていくことがわかる.

図 5. 4　生成画像のランドマーク検出結果

5.7 考察

結果より,提案手法であるランドマーク損失は,顔画像をアニメキャラクターに変換するのに効果的であると言える.ランドマーク損失は,特に顔領域脳テクスチャや顔パーツにおいて,既存の手法より高品質なものを生成することができる.

しかし,AnimeGAN＋ランドマーク損失で何度か学習をおこなったが,学習がうまく行かない場合があった.CartoonGAN と同様にコンテンツ損失の影響を大きく受け,元画像とほぼ同じ画像が出力される場合や,CartoonGAN＋ランドマーク損失の場合のようにどの画像でも同じ出力しかしない場合の 2 つのパターンがあった.1つ目の原因はコンテンツ損失による初期化によるものと思われる.しかし,コンテンツ損失による初期化を行わずに学習を始めた場合も画像の生成はうまく行かず,ノイズのような画像が生成されてしまった.

また,CartoonGAN のみの実験も何度か繰り返したが,初期化後の画像から大きく変化することはなかった.このことからも,CartoonGAN や AnimeGAN による学習は難しいことが考えられる.

2 つ目の原因はランドマーク損失にあると思われる.結果で示したように,ランドマーク損失は実写の顔画像では大きくばらつき,アニメキャラクターの顔に近づくことでランドマークはまとまったものとなる.ランドマーク検出器では,画像全体から特徴を抽出しているため,画像のテクスチャが与える影響が大きいと思われる.実写の顔写真の原型を残す画像よりも,原型を残さず全体的にアニメのテクスチャを持つ画像のほうが,ランドマーク損失が小さくなってしまう場合,CartoonGAN＋ランドマーク損失のような結果になると考えられる.

第 6 章　おわりに

本研究では顔画像を日本のアニメキャラクターに変換するためのランドマーク損失を提案し,風景画像を対象としている CartoonGAN と AnimeGAN の改善を行った.

まず,GAN の生成画像のランドマークを検出するランドマーク検出器を Deep Alignment Network で作成した.作成したランドマーク検出器はアニメキャラクター顔画像の時のみ正しいランドマークを出力し,実写の顔画像では出力しなかった.この結果より,検出器を使用したランドマーク損失は,顔画像をアニメキャラクターに変換するように GAN の学習を誘導できると考えられる.

次に,ランドマーク損失を CartoonGAN と AnimeGAN に導入し,学習と画像生成を行った.提案したランドマーク損失は特に AnimeGAN の手法に組み合わせることで,高性能に顔領域のテクスチャを変換し,また,顔パーツをアニメのものに変換することができた.しかし,ランドマーク損失単体での学習は難しく,また,学習に与える影響が大きいためパラメータの設定などの課題が残った.

今後の展望としては,ランドマーク損失を CycleGAN といった他のモデルの損失として使用することが挙げられる.

謝辞

本論文の作成にあたり,多くの助言,指導をしてくださった三好力教授に心からお礼申し上げます.また,議論に協力してくださった三好研究室の皆様や学友の皆様に心から感謝いたします.

参考文献

[1]総務省情報通信政策研究所,令和2年度情報通信メディアの利用時間と情報行動に関する調査報告書, https://www.soumu.go.jp/main_content/000765258.pdf (参照 2022-12-01)

[2]総務省情報通信国際戦略局情報通信経済室,ICTの進化がもたらす社会へのインパクトに 関する調査研究の請負

https://www.soumu.go.jp/johotsusintokei/linkdata/h26_08_houkoku.pdf(参照 2022-12-01)

[3]Goodfellow, I.J, et al, Generative adversarial nets, Proceedings 28th Annual Conference on Neural Information Processing Systems 2014, NIPS 2014, Montreal, QC, Canada, pp. 2672–2680,2014

[4] Chen, Y, Lai, Y.K, Liu, Y.J, CartoonGAN generative adversarial networks for photo cartoonization.,Proceedings 31st Meeting of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, United States, pp. 9465–9474, 2018

[5] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. CoRR,abs/1409.1556, 2014

[6] Kowalski, M.Naruniec, J.Trzcinski.Deep alignment network: A convolutional neural network for robust face alignment.Proceedings of the International Conference on Computer Vision & Pattern Recognition (CVPRW), Faces-in-the-wild Workshop/Challenge, vol. 3, pp. 6, 2017

[7] K. O'Shea, R. Nash, An Introduction to Convolutional Neural Networks, 2015 arXiv preprint arXiv:1511.0845

[8] K. Simonyan and A. Zisserman, Very deep convolutional networks for large-scale image recognition. Computing Research Repository, abs/1409.1556, 2014.

[9]] K. He, X. Zhang, S. Ren, and J. Sun, Deep residual learning for image recognition, Proceedings of the IEEE conference on computer vision and pattern recognition, pp.770–778, 2016.

[10] J. Canny, A computational approach to edge detection, IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-8(6),pp.679–698, 1986.

[11]X. Xiong and F. D. la Torre, Supervised descent method andits applications to face alignment. In Computer Vision andPattern Recognition, 2013 IEEE Conference, pp.532–539, June 2013.

[12]D. G. Lowe. Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision,60,pp.91–110,2004

[13] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15, pp.1929–1958, 2014.

[14]Zhen-Hua Feng, Josef Kittler, Muhammad Awais, Patrik Huber and Xiao-Jun Wu:Wing Loss for Robust Face Landmark Localisation with Convolutional Neural Networks,arXiv preprint arXiv:1711.06753, 2017

[15]Stricker, M., Augereau, O., Kise, K. and Iwata, M.:Face Landmark Detection for Manga Images, arXiv preprint arXiv:1811.03214 (2018).

[16]Sagonas, C., Tzimiropoulos, G., Zafeiriou, S., Pantic, M.: 300 faces in-the-wild challenge: The first face landmark localization challenge. In: Computer Vision Workshops (ICCVW), 2013 IEEE International Conference on, pp. 397–403, 2013

[17]Dlib C++ Library http://dlib.net/

[18]OpenCV http://opencv.org/

[19]] Kai Chen, Jiangmiao Pang, Jiaqi Wang, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jianping Shi, Wanli Ouyang, et al. Hybrid task cascade for instance segmentation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp.4974– 4983, 2019.

[20]MMPose Contributors,OpenMMLab Pose Estimation Toolbox and Benchmark ,https://github.com/open-mmlab/mmpose,2020.

[21]hysts,Anime Face Detector,https://github.com/hysts/anime-face-detector,2021.

[22]Eslami, M., Neuschaefer-Rube, C. & Serrurier, A. Automatic vocal tract landmark localization from midsagittal MRI data. Sci Rep 10, pp.1468, 2020.

[23]Duchi, John, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." Journal of Machine Learning Research 12.Jul, pp.2121-2159,2011

[24]Zeiler, Matthew D. "ADADELTA: an adaptive learning rate method." arXiv preprint arXiv:1212.5701,2012

[25] Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980,2014

[26] Mao, X., Li, Q., Xie, H., Lau, R.Y., Wang, Z., Smolley, S.P.: Least squares generative adversarial networks. In: Proceedings 2017 IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, pp. 2813–2821,2017.

発表履歴

1. 近澤勇太：″指定画像と類似した画像生成のためのデータセットの構築,″ 2018 年度龍谷大学卒業論文, (2019).

2. CHIKAZAWA Yuta, MIYOSHI Tsutomu："Construction of GAN dataset to generate images similar to copyrighted images," Proceedings of the 11th International Conference on Information Science and Application 2020 (ICISA2020), Virtual Conference, 31509 (2020).

3. CHIKAZAWA Yuta, MIYOSHI Tsutomu："Construction of GAN Dataset to Generate Images Similar to Copyrighted Images," Lecture Notes in Electrical Engineering 739 Information Science and Applications Proceedings of ICISA2020, Springer, ISBN:978-981-33-6384-7, ISBN:978-981-33-6385-4, pp.393-398 (2021).

付録　ソースコード

Land_loss.py

```
$ cat Land_loss.py
import tensorflow as tf
import tensorflow.contrib as tf_contrib
import os
import sys
import torch
import numpy as np
import dan_model
import cv2
import math
import matplotlib.pyplot as plt
mean_shape = None
imgs_mean = None
imgs_std = None
class VGG16Model(dan_model.Model):
    def __init__(self,num_lmark,data_format=None):
        img_size=112
        filter_sizes=[64,128,256,512]
        num_convs=2
        kernel_size=3
        super(VGG16Model,self).__init__(
            num_lmark=num_lmark,
            img_size=img_size,
            filter_sizes=filter_sizes,
            num_convs=num_convs,
            kernel_size=kernel_size,
            data_format=data_format
        )
def dan_main(model_function, input_function, file_path=None):
    os.environ['TF_ENABLE_WINOGRAD_NONFUSED'] = '1'
    model_function = tf.contrib.estimator.replicate_model_fn(model_function,loss_reduction=tf.losses.Reduction.MEAN)
    session_config = tf.ConfigProto(
        inter_op_parallelism_threads=0,
        intra_op_parallelism_threads=0,
        allow_soft_placement=True)
        #device_count ={ 'CPU' : 1,'GPU': 0}
    run_config = tf.estimator.RunConfig().replace(save_checkpoints_secs=1e9,
                                          session_config=session_config)
    estimator = tf.estimator.Estimator(
        model_fn=model_function, model_dir='./model_dir', config=run_config,
        params={
              'dan_stage':2,
              'num_lmark':40,
              'data_format': 'channels_last',
              'batch_size': 8,
              'multi_gpu': False,
          })

    predict_results = estimator.predict(input_function)
    return predict_results
def dan_model_fn(features,
                 groundtruth,
                 mode,
                 stage,
                 num_lmark,
                 model_class,
                 mean_shape,
                 imgs_mean,
                 imgs_std,
                 data_format, multi_gpu=False):
    if isinstance(features, dict):
        features = features['image']
    model = model_class(num_lmark,data_format)
    resultdict = model(features,
                       stage==1 and mode==tf.estimator.ModeKeys.TRAIN,
                       stage==2 and mode==tf.estimator.ModeKeys.TRAIN,
                       mean_shape,imgs_mean,imgs_std)
    mode = tf.estimator.ModeKeys.PREDICT
    if mode == tf.estimator.ModeKeys.PREDICT:
        return tf.estimator.EstimatorSpec(
        mode=mode,
        predictions=resultdict
        )
    loss_s1 = tf.reduce_mean(tf.reduce_mean(tf.sqrt(tf.reduce_sum(tf.squared_difference(groundtruth,resultdict['s1_ret']),-1)),-1) /
tf.sqrt(tf.reduce_sum(tf.squared_difference(tf.reduce_max(groundtruth,1),tf.reduce_min(groundtruth,1)),-1)))
    loss_s2 = tf.reduce_mean(tf.reduce_mean(tf.sqrt(tf.reduce_sum(tf.squared_difference(groundtruth,resultdict['s2_ret']),-1)),-1) /
tf.sqrt(tf.reduce_sum(tf.squared_difference(tf.reduce_max(groundtruth,1),tf.reduce_min(groundtruth,1)),-1)))
    with tf.control_dependencies(tf.get_collection(tf.GraphKeys.UPDATE_OPS,'s1')):
        optimizer_s1 = tf.train.AdamOptimizer(0.001)
        if multi_gpu:
            optimizer_s1 = tf.contrib.estimator.TowerOptimizer(optimizer_s1)
        train_op_s1 = optimizer_s1.minimize(loss_s1,global_step=tf.train.get_or_create_global_step(),
                                    var_list=tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES, 's1'))
    with tf.control_dependencies(tf.get_collection(tf.GraphKeys.UPDATE_OPS,'s2')):
```

```
        optimizer_s2 = tf.train.AdamOptimizer(0.001)
        if multi_gpu:
            optimizer_s2 = tf.contrib.estimator.TowerOptimizer(optimizer_s2)
        train_op_s2 = optimizer_s2.minimize(loss_s2,global_step=tf.train.get_or_create_global_step(),
                                    var_list=tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES, 's2'))
    loss = loss_s1 if stage == 1 else loss_s2
    train_op = train_op_s1 if stage == 1 else train_op_s2

    if (mode == tf.estimator.ModeKeys.TRAIN or
        mode == tf.estimator.ModeKeys.EVAL):
            loss = loss_s1 if stage == 1 else loss_s2
    else:
        loss = None
    if mode == tf.estimator.ModeKeys.TRAIN:
        train_op = train_op_s1 if stage == 1 else train_op_s2
    else:
        train_op = None
    return tf.estimator.EstimatorSpec(
        mode=mode,
        predictions=resultdict,
        loss=loss,
        train_op=train_op
        )
def vgg16_input_fn(is_training,data_dir,batch_size=64,num_epochs=1,num_parallel_calls=1, multi_gpu=False):
    img_path,pts_path = get_filenames(data_dir)
    def decode_img_pts(img,pts,is_training):
        img = cv2.imread(img.decode(), cv2.IMREAD_GRAYSCALE)
        pts = np.loadtxt(pts.decode(),dtype=np.float32,delimiter=',')
        return img[:,:,np.newaxis].astype(np.float32),pts.astype(np.float32)
    map_func=lambda img,pts,is_training:tuple(tf.py_func(decode_img_pts,[img,pts,is_training],[tf.float32,tf.float32]))
    img = tf.data.Dataset.from_tensor_slices(img_path)
    pts = tf.data.Dataset.from_tensor_slices(pts_path)
    dataset = tf.data.Dataset.zip((img, pts))
    num_images = len(img_path)
    return dan_run_loop_modified.process_record_dataset(dataset,is_training,batch_size,
                                    num_images,map_func,num_epochs,num_parallel_calls,
                                    examples_per_epoch=num_images, multi_gpu=multi_gpu)
def vgg16_model_fn(features, labels, mode, params):
        return dan_model_fn(features=features,
                        groundtruth=labels,
                        mode=tf.estimator.ModeKeys.PREDICT,
                        stage=2,
                        num_lmark=40,
                        model_class=VGG16Model,
                        mean_shape=mean_shape,
                        imgs_mean=imgs_mean,
                        imgs_std=imgs_std,
                        data_format='channels_last',
                        multi_gpu=False)
def tf_input_fn(img_input, land_num):
    return 0
def img_input_fn(aaa, land_m):
    def _input_fnn():
        yield (aaa, land_m)
    def input_fnnn():
        img_size = 112
        num_lmark = 40
        datasetss = tf.data.Dataset.from_generator(_input_fnn ,(tf.float32,tf.float32),(tf.TensorShape([img_size,img_size]),tf.TensorShape([num_lmark,2])))
        return datasetss
    return input_fnnn
def WING_Loss(fake_land, real_land):
    w = 5.0
    epsilon=2.0
    x = fake_land - real_land
    c = w * (1.0 - math.log(1.0 + w/epsilon))
    absolute_x = np.abs(x)
    losses = np.where(
        np.greater(w, absolute_x),
        w*np.log(1.0 + absolute_x/epsilon),
        absolute_x - c
    )
    l_s = np.sum(losses,axis=2)
    W_loss = np.mean(np.sum(l_s,axis=1),axis=0)/2
    return W_loss
#ランドマーク損失
def land_gen_loss(fake,x,real_land, batch_size):
    mean_shape = None
    imgs_mean = None
    imgs_std = None
    w_l = 0.0
    i = 0
    #print(fake.shape)
    fake_b_land = np.zeros([batch_size,40,2])
    for i in range(batch_size):
        fake_img = fake[i]
        fake_np = (fake_img.detach().cpu().numpy().copy().transpose(1,2,0)+1)/2
        fake_np_255 = fake_np * 255
        fake_gray = cv2.cvtColor(fake_np_255, cv2.COLOR_RGB2GRAY)
        #fake_gray = 0.299 * fake_np[:, :, 0] + 0.587 * fake_np[:, :, 1] + 0.114 * fake_np[:, :, 2]
        land_fake =np.zeros([40,2])
        input_img = img_input_fn(fake_gray,land_fake)
        fake_res = dan_main(vgg16_model_fn, input_img)
        for xx in fake_res:
            fake_land = xx['s2_ret']
            img = xx['img']
```

```
        img = cv2.cvtColor(img,cv2.COLOR_GRAY2RGB)
        #np.savetxt('./'+str(i)+'c_pred.pts', fake_land, delimiter=" ", fmt='%i')
        #for lm in fake_land:
            #print(lm[0],lm[1])
            #cv2.circle(img, (lm[0], lm[1]), 1, (0,0,255),-1)
        #cv2.imwrite('./'+str(i)+' c_red.png', img)
        #np.savetxt('./'+str(i)+'c_pred.pts', fake_land, delimiter=" ", fmt='%i')
    fake_b_land[i] = fake_land.astype(np.float32)
    #fake_b_land[i] = torch.from_numpy(fake_land.astype(np.float32)).clone()
    #print(fake_land.shape,real_land[i].shape)
    #w_l += WING_Loss(fake_land, real_land)
    i+=1
    #print(w_l)
w_l = WING_Loss(fake_b_land,real_land)
print(w_l)
return w_l
```

Landmark_reshape.py

```
$ cat Landmark_reshape.py
import os
import cv2
land_dir = 'dataset/landmark_train'
land_40 = 'dataset/40_land_train'
img_dir = 'dataset/train_photo'
save_img = 'dataset/40_img'
files = os.listdir(land_dir)
face_40x = [0] * 40
face_40y = [0] * 40
x = [0] * 68
y = [0] * 68
land_point = [18,20,22,23,25,27,37,38,39,40,41,42,43,44,45,46,47,48,31,60,67,56,58]
for name in files:
    land_path = os.path.join(land_dir, name)
    save_path = os.path.join(land_40, name)
    img_name = name.replace('txt','png')
    img_path = os.path.join(img_dir, img_name)
    save_img_path = os.path.join(save_img, img_name)
    img = cv2.imread(img_path)
    f = open(land_path,'r')
    ii = 0
    land = []
    for line in f:
        landstr = line.split(' ')
        x[ii] = int(landstr[0])
        y[ii] = int(landstr[1])
        ii += 1
    f.close()
    y[39] += 8.8
    y[40] += 8.8
    y[41] += 8.8
    y[45] += 8.8
    y[46] += 8.8
    y[47] += 8.8
    for a in range(17):
        face_40x[a] = x[a]
        face_40y[a] = y[a]
    i = 17
    for b in range(23):
        l_p = land_point[b] - 1
        face_40x[i+b] = x[l_p]
        face_40y[i+b] = y[l_p]
    ff = open(save_path, 'x')
    for c in range(40):
        img_p = cv2.circle(img, (face_40x[c],face_40y[c]), 1, (255,0,0), -1)
        point = str(face_40x[c])+' '+str(face_40y[c])
        ff.write(point)
        ff.write('\n')
    ff.close()
    cv2.imwrite(save_img_path, img_p)
```

train.py

```
$ diff -u pytroch-animeGANorg/train.py  pytorch-animeGAN/train.py
--- pytorch-animeGANorg/train.py                2023-02-08 11:35:01.512788649 +0900
+++ pytorch-animeGAN/train.py                   2023-01-23 12:43:47.298373667 +0900
@@ -1,9 +1,11 @@
 import torch
 import argparse
+import pickle
 import os
 import cv2
 import numpy as np
 import torch.optim as optim
+import land_loss
 from multiprocessing import cpu_count
 from torch.utils.data import DataLoader
 from modeling.anime_gan import Generator
@@ -20,17 +22,24 @@
 gaussian_mean = torch.tensor(0.0)
 gaussian_std = torch.tensor(0.1)
 def parse_args():
     parser = argparse.ArgumentParser()
     parser.add_argument('--dataset', type=str, default='Hayao')
-    parser.add_argument('--data-dir', type=str, default='/content/dataset')
+    parser.add_argument('--data-dir', type=str, default='content/dataset')
     parser.add_argument('--epochs', type=int, default=100)
-    parser.add_argument('--init-epochs', type=int, default=5)
+    parser.add_argument('--init-epochs', type=int, default=4)
     parser.add_argument('--batch-size', type=int, default=6)
-    parser.add_argument('--checkpoint-dir', type=str, default='/content/checkpoints')
-    parser.add_argument('--save-image-dir', type=str, default='/content/images')
+    parser.add_argument('--checkpoint-dir', type=str, default='content/checkpoints')
+    parser.add_argument('--save-image-dir', type=str, default='content/images')
     parser.add_argument('--gan-loss', type=str, default='lsgan', help='lsgan / hinge / bce')
     parser.add_argument('--resume', type=str, default='False')
     parser.add_argument('--use_sn', action='store_true')
@@ -39,11 +48,11 @@
     parser.add_argument('--lr-g', type=float, default=2e-4)
     parser.add_argument('--lr-d', type=float, default=4e-4)
     parser.add_argument('--init-lr', type=float, default=1e-3)
-    parser.add_argument('--wadvg', type=float, default=10.0, help='Adversarial loss weight for G')
-    parser.add_argument('--wadvd', type=float, default=10.0, help='Adversarial loss weight for D')
+    parser.add_argument('--wadvg', type=float, default=300.0, help='Adversarial loss weight for G')
+    parser.add_argument('--wadvd', type=float, default=300.0, help='Adversarial loss weight for D')
     parser.add_argument('--wgra', type=float, default=3.0, help='Gram loss weight')
-    parser.add_argument('--wcol', type=float, default=30.0, help='Color loss weight')
+    parser.add_argument('--wcol', type=float, default=10.0, help='Color loss weight')
     parser.add_argument('--d-layers', type=int, default=3, help='Discriminator conv layers')
     parser.add_argument('--d-noise', action='store_true')
@@ -51,9 +60,10 @@
 def collate_fn(batch):
-    img, anime, anime_gray, anime_smt_gray = zip(*batch)
+    img,land, anime, anime_gray, anime_smt_gray = zip(*batch)
     return (
         torch.stack(img, 0),
+        np.stack(land,0),
         torch.stack(anime, 0),
         torch.stack(anime_gray, 0),
         torch.stack(anime_smt_gray, 0),
@@ -76,12 +86,14 @@
     assert args.gan_loss in {'lsgan', 'hinge', 'bce'}, f'{args.gan_loss} is not supported'
-def save_samples(generator, loader, args, max_imgs=2, subname='gen'):
+def save_samples(generator, loader,epoch, args, max_imgs=2, subname='gen'):
     '''
     Generate and save images
     '''
     generator.eval()
+
+
     max_iter = (max_imgs // args.batch_size) + 1
     fake_imgs = []

@@ -99,7 +111,7 @@
     fake_imgs = np.concatenate(fake_imgs, axis=0)

     for i, img in enumerate(fake_imgs):
-        save_path = os.path.join(args.save_image_dir, f'{subname}_{i}.jpg')
+        save_path = os.path.join(args.save_image_dir, f'{subname}_{epoch}_{i}.jpg')
         cv2.imwrite(save_path, img[..., ::-1])
@@ -159,13 +171,15 @@
         if e < args.init_epochs:
             # Train with content loss only
             set_lr(optimizer_g, args.init_lr)
-            for img, *_ in bar:
+            #for img,land, *_ in bar:
                 img = img.cuda()
                 optimizer_g.zero_grad()
                 fake_img = G(img)
                 loss = loss_fn.content_loss_vgg(img, fake_img)
+                #l_loss = land_loss.land_gen_loss(fake_img, img, land, batch_size)
+
                 loss.backward()
```

```
                optimizer_g.step()
@@ -175,11 +189,11 @@
            set_lr(optimizer_g, args.lr_g)
            save_checkpoint(G, optimizer_g, e, args, posfix='_init')
-           save_samples(G, data_loader, args, subname='initg')
+            save_samples(G, data_loader,e, args, subname='initg')
            continue
        loss_tracker.reset()
-       for img, anime, anime_gray, anime_smt_gray in bar:
+       for img,land, anime, anime_gray, anime_smt_gray in bar:
            # To cuda
            img = img.cuda()
            anime = anime.cuda()
@@ -215,26 +230,36 @@
            fake_img = G(img)
            fake_d = D(fake_img)
+           adv_loss, con_loss, gra_loss, col_loss = loss_fn.compute_loss_G(
+           fake_img, img, fake_d, anime_gray)

+           l_loss =0.05 * land_loss.land_gen_loss(fake_img, img, land, batch_size)
            loss_g = adv_loss + con_loss + gra_loss + col_loss
            loss_g.backward()
            optimizer_g.step()
-           loss_tracker.update_loss_G(adv_loss, gra_loss, col_loss, con_loss)
+            loss_tracker.update_loss_G(adv_loss, gra_loss, col_loss, con_loss,l_loss)
-           avg_adv, avg_gram, avg_color, avg_content = loss_tracker.avg_loss_G()
+            avg_adv, avg_gram, avg_color, avg_content,avg_land = loss_tracker.avg_loss_G()
            avg_adv_d = loss_tracker.avg_loss_D()
-           bar.set_description(f'loss G: adv {avg_adv:2f} con {avg_content:2f} gram {avg_gram:2f} color {avg_color:2f} / loss D: {avg_adv_d:2f}')
+            bar.set_description(f'loss G: adv {avg_adv:2f} con {avg_content:2f} gram {avg_gram:2f} color {avg_color:2f} land{avg_land:2f} / loss D: {avg_adv_d:2f}')
        if e % args.save_interval == 0:
            save_checkpoint(G, optimizer_g, e, args)
            save_checkpoint(D, optimizer_d, e, args)
-           save_samples(G, data_loader, args)
+            save_samples(G, data_loader,e, args)
 if __name__ == '__main__':
     args = parse_args()
```

dataset.py

```
$ diff -u pytorch-animeGANorg/dataset.py pytorch-animeGAN/dataset.py
--- pytorch-animeGANorg/dataset.py              2023-02-08 11:35:01.364770654 +0900
+++ pytorch-animeGAN/dataset.py                 2023-01-26 17:58:34.568716816 +0900
@@ -35,9 +35,10 @@
         self.debug_samples = args.debug_samples or 0
         self.data_dir = data_dir
         self.image_files =  {}
-        self.photo = 'train_photo'
-        self.style = f'{anime_dir}/style'
-        self.smooth =  f'{anime_dir}/smooth'
+        self.photo = 'train_pho'
+        self.land = 'train_land'
+        self.style = 'lovelive_112/style'
+        self.smooth =  'lovelive_112/smooth'
         self.dummy = torch.zeros(3, 256, 256)

         for opt in [self.photo, self.style, self.smooth]:
@@ -62,7 +63,7 @@
         return len(self.image_files[self.smooth])

     def __getitem__(self, index):
-        image = self.load_photo(index)
+        image, landmarks = self.load_photo(index)
         anm_idx = index
         if anm_idx > self.len_anime - 1:
             anm_idx -= self.len_anime * (index // self.len_anime)
@@ -70,14 +71,28 @@
         anime, anime_gray = self.load_anime(anm_idx)
         smooth_gray = self.load_anime_smooth(anm_idx)

-        return image, anime, anime_gray, smooth_gray
+        return image,landmarks, anime, anime_gray, smooth_gray

     def load_photo(self, index):
         fpath = self.image_files[self.photo][index]
         image = cv2.imread(fpath)[:,:,::-1]
         image = self._transform(image, addmean=False)
         image = image.transpose(2, 0, 1)
-        return torch.tensor(image)
+
+        l_dir = fpath.replace('pho','land')
+        land_path = l_dir.replace('png','txt')
+        ff = open(land_path,'r')
+        land = np.zeros([40,2])
+        i = 0
+        for line in ff:
+            landstr = line.split(' ')
+            land[i][0] = int(landstr[0])
+            land[i][1] = int(landstr[1])
+            i+=1
+        landmarks = land.astype('float32').reshape(-1,2)
+        return torch.tensor(image),landmarks
     def load_anime(self, index):
         fpath = self.image_files[self.style][index]
```

land_CartoonGAN.py

```
$ diff -u pytorch-CartoonGANorg/CartoonGAN.py pytorch-CartoonGAN/land_CartoonGAN.py
--- pytorch-CartoonGANorg/CartoonGAN.py                 2023-02-08 11:48:16.482030915 +0900
+++ pytorch-CartoonGAN/land_CartoonGAN.py               2023-02-08 12:09:30.169728626 +0900
@@ -1,4 +1,4 @@
-import os, time, pickle, argparse, networks, utils
+import os, time, pickle, argparse, networks, utils,land_loss
 import torch
 import torch.nn as nn
 import torch.optim as optim
@@ -15,11 +15,11 @@
 parser.add_argument('--out_ngc', type=int, default=3, help='output channel for generator')
 parser.add_argument('--in_ndc', type=int, default=3, help='input channel for discriminator')
 parser.add_argument('--out_ndc', type=int, default=1, help='output channel for discriminator')
-parser.add_argument('--batch_size', type=int, default=8, help='batch size')
+parser.add_argument('--batch_size', type=int, default=32, help='batch size')
 parser.add_argument('--ngf', type=int, default=64)
 parser.add_argument('--ndf', type=int, default=32)
 parser.add_argument('--nb', type=int, default=8, help='the number of resnet block layer for generator')
-parser.add_argument('--input_size', type=int, default=256, help='input size')
+parser.add_argument('--input_size', type=int, default=112, help='input size')
 parser.add_argument('--train_epoch', type=int, default=100)
 parser.add_argument('--pre_train_epoch', type=int, default=10)
 parser.add_argument('--lrD', type=float, default=0.0002, help='learning rate, default=0.0002')
@@ -37,7 +37,7 @@
 print('-------------- End ----------------')

 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
+#device = torch.device('cpu')
 if torch.backends.cudnn.enabled:
     torch.backends.cudnn.benchmark = True

@@ -63,7 +64,10 @@
         transforms.ToTensor(),
         transforms.Normalize(mean=(0.5, 0.5, 0.5), std=(0.5, 0.5, 0.5))
 ])
-train_loader_src = utils.data_load(os.path.join('data', args.src_data), 'train', src_transform, args.batch_size, shuffle=True, drop_last=True)
+image_dir = 'data/src_data/train'
+
+train_loader_src = utils.land_img_load(image_dir, src_transform, args.batch_size)
+#train_loader_src = utils.data_load(os.path.join('data', args.src_data), 'train', src_transform, args.batch_size, shuffle=True, drop_last=True)
 train_loader_tgt = utils.data_load(os.path.join('data', args.tgt_data), 'pair', tgt_transform, args.batch_size, shuffle=True, drop_last=True)
 test_loader_src = utils.data_load(os.path.join('data', args.src_data), 'test', src_transform, 1, shuffle=True, drop_last=True)

@@ -97,6 +101,7 @@

 # loss
 BCE_loss = nn.BCELoss().to(device)
+MSE_loss = nn.MSELoss().to(device)
 L1_loss = nn.L1Loss().to(device)

 # Adam optimizer
@@ -117,16 +122,17 @@
     for epoch in range(args.pre_train_epoch):
         epoch_start_time = time.time()
         Recon_losses = []
+        #for x, land in train_loader_src:
         for x, _ in train_loader_src:
             x = x.to(device)
-
+
             # train generator G
             G_optimizer.zero_grad()
-
+            #l_loss = land_loss.land_gen_loss(G_,x, land, args.batch_size)
             Recon_loss = 10 * L1_loss(G_feature, x_feature.detach())
             Recon_losses.append(Recon_loss.item())
             pre_train_hist['Recon_loss'].append(Recon_loss.item())
@@ -184,25 +191,27 @@
-    for (x, _), (y, _) in zip(train_loader_src, train_loader_tgt):
+    for (x, land), (y, _) in zip(train_loader_src, train_loader_tgt):
         e = y[:, :, :, args.input_size:]
         y = y[:, :, :, :args.input_size]
         x, y, e = x.to(device), y.to(device), e.to(device)
+        land = land.to(device)
@@ -213,16 +222,31 @@
         G_optimizer.zero_grad()
         G_ = G(x)
+
+        l_loss = land_loss.land_gen_loss(G_,x, land, args.batch_size)
-        Gen_loss = D_fake_loss + Con_loss
+        Gen_loss = (300 * D_fake_loss)+(0.5*l_loss) + Con_loss
@@ -235,6 +259,11 @@
     print(
     '[%d/%d] - time: %.2f, Disc loss: %.3f, Gen loss: %.3f, Con loss: %.3f' % ((epoch + 1), args.train_epoch, per_epoch_time,
```

```
torch.mean(torch.FloatTensor(Disc_losses)),
          torch.mean(torch.FloatTensor(Gen_losses)), torch.mean(torch.FloatTensor(Con_losses))))
+     if(epoch % 2 == 0):
+         torch.save(G.state_dict(), os.path.join(args.name +'_results', str(epoch)+'generator_latest.pkl'))
+         torch.save(D.state_dict(), os.path.join(args.name +'_results', str(epoch)+'discriminator_latest.pkl'))

      if epoch % 2 == 1 or epoch == args.train_epoch - 1:
          with torch.no_grad():
```