令和３年度 特別研究報告書

GANを利用した
通信データ圧縮手法の検討

龍谷大学　理工学部　情報メディア学科

T180402　牛田稜也

指導教員　三好力　教授

内容梗概

近年、社会の情報化が進んできており、それに伴ってデータをコンピュータ同士でインターネットを介してやり取りする機会が増加してきている。これにより大容量のデータをインターネットを利用して転送する機会も増加することが考えられる。しかし、現在のシステムでは大容量のデータを送信する場合に様々な手間や問題が発生する。したがって、今後は大容量のデータを手軽に問題を起こさず送信する手法を考えなければならない。

本研究では、本物のようなデータの生成を行う技術である GAN を利用して、データ送信の際に圧縮したデータと GAN の学習済みパラメータを送信し、受信側では GAN を利用して、圧縮したデータを入力として元のデータと酷似したデータ生成することで疑似的に復元する手法を提案する。実験では取り扱うデータを画像とし、強く圧縮された画像をもとに GAN が生成した画像が元画像を再現できているか、既存の圧縮技術以上の圧縮率を実現できるかを検証し、本手法の実用性を検討する。

目次

第 1 章 はじめに

1.1 研究目的

近年、情報技術の発展は目覚ましく、社会の情報化が進んでいる。これによりコンピュータ同士でのインターネットを介したデータのやり取りが増加してきており、今後さらに増加していくことが予想される。このような状況から、大容量のデータをインターネット上でやりとりする機会もまた増加していくと考えられる。

しかし、現在のメール等のデータ送信システムでは送信可能なデータ量に制限が設けられている。このような制限は送信先のコンピュータに大きな負担を掛けさせないようにするために設けられているが、この制限によって気軽に大容量のデータを送信することが難しい状態にある。[1]この制限は既存のシステムでも大容量のデータを転送するためのサービスを利用する、圧縮技術を用いてデータを圧縮するといったような方法で対処することは可能ではある。しかし、他のサービスを用いる方法ではデータの送信に手間がかかってしまい、既存の圧縮技術を用いてデータを圧縮する方法では用いる圧縮技術によってはあまり圧縮できない、元のデータよりも劣化したデータになってしまうという問題がある。[2]これらのことから、今後は大容量のデータを手軽にかつ元のデータの品質を維持したままデータを転送することを可能とする仕組みが求められると考えられる。

本研究では、本物のようなデータの生成を行う技術である GAN を利用して、データ送信の際に圧縮したデータと GAN の学習済みパラメータを送信し、受信側では GAN を利用して圧縮したデータを入力として元のデータと酷似したデータを生成することで疑似的に復元する手法を提案する。本論文では取り扱うデータを画像とし、実験では強く圧縮された画像をもとに GAN が生成した画像が元画像を再現できているか、既存の圧縮技術以上の圧縮率を実現できるかを検証し、本手法の実用性を検討する。

第 2 章 既存技術

2.1 Generative Adversarial Networks(GAN)

GAN は敵対的生成ネットワークと呼ばれる生成モデルの一種であり、データからその特徴を学習することにより元のデータに似た性質を持つ新たなデータの生成や、存在するデータをその特徴に沿って変換することを可能とする技術である。

GAN は Generator(生成器)と Discriminator(判別器)と呼ばれる 2 つのネットワークを組み合わせることで新たなデータの生成を可能としている。GAN のアーキテクチャを図 2.1 に示す。

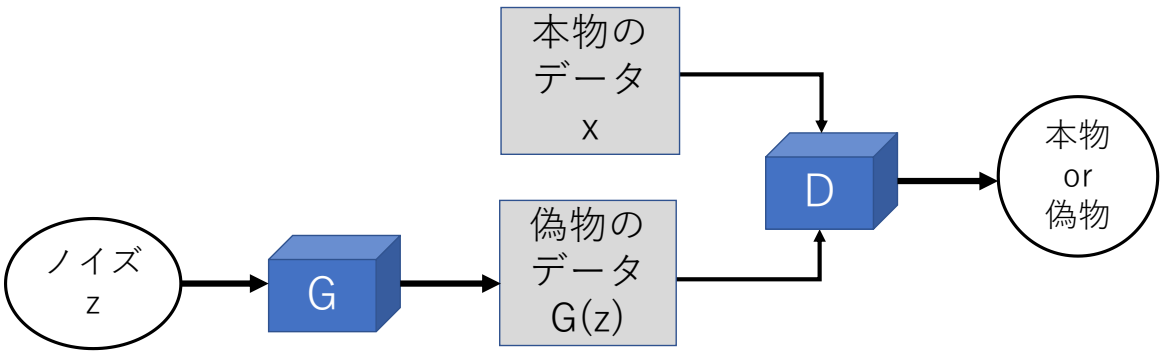

図 2.1 GAN の構造

図 2.1 における G は Generator を、D は Discriminator をそれぞれ表している。Generator ノイズベクトルと呼ばれるもの(z)を入力値として受け取り、そこからデータを出力する。これに対し Discriminator は本物のデータを基に Generator が生成したデータが本物のデータであるか Generator によって生成された偽物のデータであるかを判定する。これら 2 つのネットワークは、Generator は Discriminator を騙せるような画像を生成することを、Discriminator は Generator が生成した画像が偽物であると判別することをそれぞれ目的としており、GAN ではこれらの相反する目的を持つ 2 つのネットワークを互いに競わせることで精巧なデータを生成することが可能となる。

Generator と Discriminator の 2 つのネットワークの競合関係は、各ネットワークの損失関数を共有することで表現される。Generator の損失関数を式 2.1 に、Discriminator の損失関数を式 2.2 にそれぞれ示す。

$$\mathrm{L}_G = \frac{1}{m}\sum_{i=1}^{m} log\left(1 - D\big(G(z)\big)\right) \tag{2.1}$$

$$\mathrm{L}_D = \frac{1}{m}\sum_{i=1}^{m} \left[log\mathrm{D}(\mathrm{x}) + log\left(1 - D\big(G(z)\big)\right)\right] \tag{2.2}$$

式中の m はミニバッチのサイズである。式 2.1 は生成したデータが本物と判定された場合に最小となる関数であり、Generator ではこの損失関数の出力値が最小となるように学習を行う。式 2.2 は識別したデータが本物である場合に最大値(1)を出力する関数であるため、学習時にはこの関数の出力値が最大になるように学習が行われる。[3]

2.2 SRGAN

SRGAN は GAN から派生した手法の一つであり、画像の超解像に初めて GAN を適用したモデルである。SRGAN の Generator の構造を図 2.2 に、Discriminator の構造を図 2.3 にそれぞれ示す。

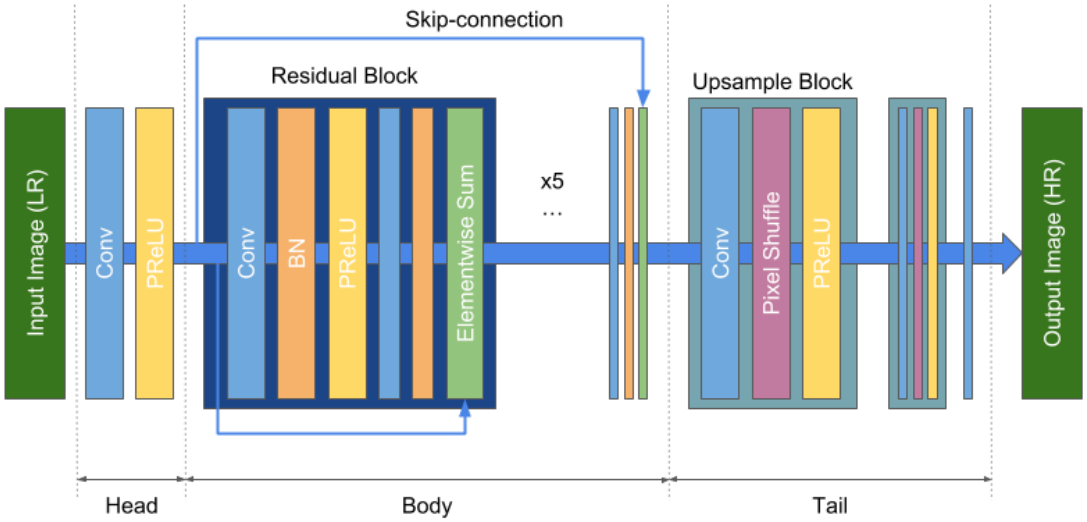

図 2.2 SRGAN の Generator

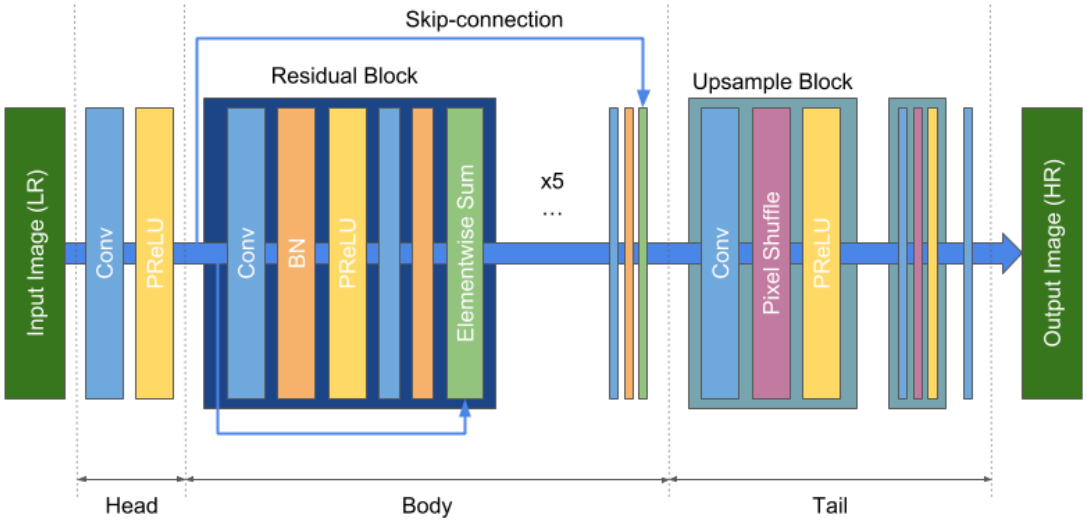

図 2.3 SRGAN の Discriminator

図 2.2 に示す Generator は ResNet を参考に構成されている。ResNet は Residual Block と呼ばれる畳み込みの入出力を足し合わせたものを積み重ねているネットワークであり、SRGAN も ResNet と同様に Residual Block を積み重ねた構造をしている。図 2.3 に示す Discriminator は単純に畳み込みを重ねていく構造になっている。このように構成される SRGAN による超解像は、従来の超解像手法に比べて知覚品質が高く自然な画像になる。[4]

2.3 既存の画像圧縮技術

2.3.1 JPEG

画像データには画質とは関係ない表面に見えない多量のデータが存在しており、これがデータのサイズを増やす要因となっている。JPEG ではこのような画質に影響を与えない部分のデータを削除することで画像を圧縮している。このような方法で圧縮を行っているため、JPEG 圧縮を利用することで非常に高い圧縮率を実現することができる。ただし、画像を保存するたびにデータを削除することになるため、画像の保存を繰り返すとその画像が劣化してしまうという問題点がある。[2]

2.3.2 PNG

画像データは全てビットと呼ばれる点の集合で表現されており、画像を保存する際にはそのビットを一つずつ保存していくことになる。しかし PNG を利用した場合は同じ色のパターンを再び保存することはせずに一つの括りとして処理をしているため、画像データの容量を抑えることができる。このような手法で圧縮を行っているため、この方法では画像の品質を保ったまま画像を軽量化することが可能となっている。ただし JPEG と比べた場合、画像の品質を維持できる代わりに圧縮率が低くなるという欠点がある。[2]

2.4 既存の圧縮技術の問題点

既存の圧縮技術として取り上げた代表的な二つの圧縮技術は、JPEG は非常に高い圧縮率の実現が可能という点、PNG は画像の品質を保ったまま画像の軽量化を行えるという点がそれぞれの強みとなっている。しかし、JPEG 圧縮はその過程で画像に含まれるデータを削除しているため、JPEG による保存を繰り返すとその画像が劣化してしまうという問題がある。また PNG 圧縮は画像の品質を維持することが可能だが、この方法による圧縮では圧縮率に関しては高くなることが期待できない。このように、現在の圧縮技術による圧縮は、元のデータの再現率とデータの圧縮率はトレードオフの関係にあるといえる。

そこで本研究ではデータの圧縮技術に GAN を組み合わせる。本研究では GAN の持つ本物のようなデータを生成するという特徴に着目し、GAN を用いてデータの復元を行うことで、データの送受信時という限定的な状況下において高い圧縮率と再現率を持つデータ圧縮技術を実現することが可能なのではないかと考えた。

第 3 章 提案手法

3.1 提案手法

データの送信において、GAN の Generator を用いて元のデータと同じ(またはそれに限りなく近い)データを生成することで圧縮したデータを疑似的に復元する手法を提案する。GAN には本物のようなデータを生成することができるという性質があることため、この技術を利用することで圧縮したデータから元のデータを生成することが可能なのではないかと考えた。

提案する手法を図式化したものを図 3.1 に示す。

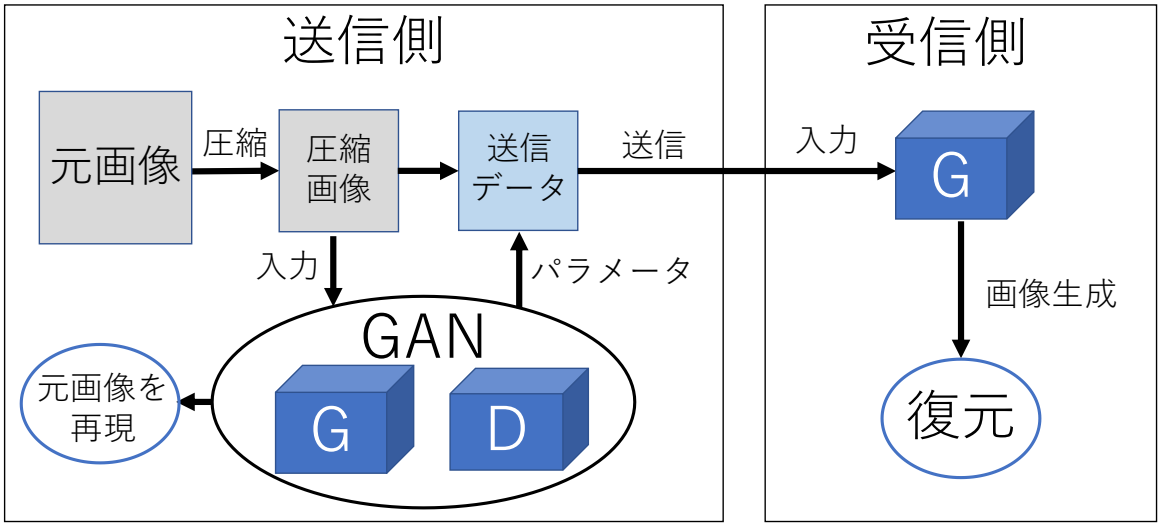

図 3.1 提案手法

図 3.1 における G は Generator を、D は Discriminator をそれぞれ表しており、送信側、受信側にある Generator はどちらも同じ構造をしているものを想定している。本手法では、まず送信側において送信データを含めた画像群をデータセットとそれらの画像を圧縮したものをデータセットとして GAN を学習し、圧縮後の画像から元の画像に酷似した画像を生成するように学習し、受信側では Generator がその学習によって得られたパラメータを用いて圧縮した画像から元の画像を再現することを目指す。元画像の再現に成功した場合、その時 Generator の重みなどのパラメータを記録し、圧縮した画像とともに送信データとしてまとめ受信側へと送信する。受信側では受け取った送信側の Generator のパラメータを送信側と同じ構造をしている受信側の Generator

にそのまま設定する。受信側の Generator は送信側の Generator と同じ構造をしているため、受信側の Generator に送信側で元画像を再現するために Generator に入力したものと同じデータを入力して画像を生成することで送信側において再現した画像と同じ画像を生成することができ、送信側における再現が成功している場合、受信側において元画像を疑似的に復元することができる。この方法により、データの性質を維持しつつ送信するデータ量のみを減らせるのではないかと考えた。

実験では GAN によって圧縮した画像を復元した後、復元した画像が元画像をどの程度再現することができているか、画像の生成に必要となる情報、本論文で提案するシステムを実現する場合に送信することになる情報量を調べ、画像の圧縮率等を既存の画像圧縮技術と比較することで本手法の実用性を検討する。

第 4 章 実験

4.1 実験概要

本実験では、圧縮した画像に対して GAN を使用し、圧縮した画像を復元することが可能であるかどうかを検証する。また、復元の際に GAN を利用するためにどのような情報が必要となるかを調べ、データ転送時のデータ量を減らす手法として実用的であるかどうかを検討する。

4.2 実験方法

実験では、SRGAN のアルゴリズムを利用して作成したプログラムを用いて学習及び復元を行う。データセットには「General-100」と呼ばれる 100 個の画像からなるデータセットを使用し、このデータセットに含まれている各画像を Python の画像処理用のライブラリである OpenCV を用いて画像のサイズを 128×128 に変更した後に画像の圧縮については、JPEG 圧縮のアルゴリズムを利用して既存技術よりも高い圧縮率になるように圧縮を行い、それらを圧縮したそれぞれの画像の元画像とペアとして学習を行う。その学習が終了した後、その学習によって得られた Generator のパラメータを利用して画像を生成する。画像の圧縮の際は JPEG 圧縮と同様の方法で、画像の欠損が分かりやすくなる程度まで圧縮する。その後、圧縮前の画像と圧縮後の画像について、画像ファイルのサイズ、ヒストグラム、画像を構成する ndarray 配列の一致率をそれぞれ比較する。ヒストグラムはピクセルの色の明るさをレベル別に分布したグラフで、ヒストグラムの比較では画像の色味を加味して画像の比較を行うことができる。ヒストグラムによる比較は OpenCV の関数で行うことができ、calcHist 関数により比較する画像をヒストグラム化した後に compareHist 関数を利用することで比較することができる。ndarray 配列による比較については、OpenCV で画像を読み込んだ場合画像は NumPy 配列の ndarray(以下、ndarray 配列)として扱われ、この ndarray 配列に格納されている要素は画素値を表しているため、NumPy を利用し 2 つの画像の画素値の一致率を計算することでそれらの画像の一致率を求めることができる。2 つの画像の画素値の一致率を求める計算式を以下の式(4.1)に示す。

$$2\text{ つの画像の画素値の一致率} = \frac{2\text{ つの画像の画素値の一致数}}{\text{画像の全要素数}} \qquad (4.1)$$

今回比較する画像のサイズは同じになるため、2 つの画像の画素値の一致数は NumPy の count_nonzero 関数を用いることで求めることができ、その値を画像が持つ全要素数で割ることで 2 つの画像の画素値の一致率を求めることができる。[8]本実験

ではこの 2 つの方法で元画像と生成画像を比較し、2 つの画像がどの程度似たものになっているかを調べることで圧縮した画像から圧縮前の画像と同じ画像が生成することが可能かどうかを検証する。また、Generator に入力するデータを調べ、データの転送時にどのようなデータが必要であるかを調べる。

4.3 実験環境

本実験に用いた GAN のネットワークの構造は、図 2.2 及び図 2.3 に示した SRGAN のネットワーク構造と同様のものとなっており、Discriminator は単純な畳み込みを行い、Generator は入力画像をダウンサンプリングしたのちに Residual Block による特徴抽出を行った後、ネットワークの前半部の特徴マップを後半部のサイズが同じ特徴マップに連結する Skip connection を結び、入力特徴マップのピクセルを並び替えて高解像度の特徴マップを出力する Pixel Shuffler によるアップサンプリングにより復元画像の生成を行う構造になっている。Discriminator ではまず元画像、圧縮画像からそれぞれバッチサイズ数の半分の画像を取り出し、元画像とすべてが 1 で構成されるラベルのバイナリクロスエントロピー誤差を取って学習を行い、その後 Generator が圧縮画像を元に生成した画像とすべてが 0 で構成されるラベルのバイナリクロスエントロピー誤差をとり学習を行う。バイナリクロスエントロピー誤差は、ネットワークの出力と正解の間にどの程度差があるのかを示す尺度であり、以下の式(4.2)により定義される。[9]

$$H(p,q) = -\sum_{i} p_i log q_i \tag{4.2}$$

式(4.2) pは真値の確率変数、qは予測した確率変数となる。Generator では Discriminator と同様に画像を取り出し元画像を正解データとし、圧縮画像、正解データとした元画像、すべてが 1 で構成されるラベルでバイナリクロスエントロピー誤差を取って学習を行い復元画像を生成し、その画像と元画像をそれぞれ画像認識のための学習済みネットワークである VGG16 に入力したときのそれぞれの出力の平均二乗誤差により計算される Content Loss を求め、生成した復元画像とすべてが 1 で構成されるラベルでバイナリクロスエントロピー誤差を取り学習を行う。この Discriminator と Generator の学習をエポック数分繰り返すことで、生成する画像の精度を高めている。

また、学習の際のGANの各ハイパーパラメータの設定をまとめたものを以下の表 4.1 に示す。なお、Generator、Discriminator における学習率や学習率の減衰量に関しては機会学習のためのライブラリである keras に用意されているオプティマイザの一つである Adam のデフォルト設定となっている。

表 4.1　本実験における GAN のハイパーパラメータの設定

パラメータ名	設定値
エポック数	1000
バッチサイズ	32
Discriminator の入力層数	元画像の画像サイズ
Discriminator の出力層数	Discriminator の入力層数により決定
Generator の入力層数	圧縮画像の画像サイズ
Generator 出力層数	Generator の生成画像および

4.4 実験結果

実験に用いた画像を図 4.1 に示す。また、その画像を圧縮した画像を図 4.2 に、圧縮した画像を作成したプログラムによって復元(生成)した結果を図 4.3 としてそれぞれ以下に示す。

図 4.1　実験に用いた画像

図 4.2　図 4. 1 を圧縮した画像

図 4.3　プログラムにより生成した画像

図 4.1 に示す元画像はビットマップファイルとなっており、図 4.2 に示す圧縮後の画像のファイルサイズは 10,059 バイト(圧縮率約 79.6%)となっている。また、元の画像(図 4.1)と生成した画像(図 4.3)について、そのファイルサイズ OpenCV 及び NumPy を用いて比較したヒストグラムと ndarray 配列の一致率をまとめたものを以下の表 4.2 に示す。

表 4.2　元画像(図 4.1)と生成画像(図 4.3)の比較結果

	元画像	生成画像
ファイルサイズ	49,206 バイト	49,206 バイト
ヒストグラムの一致率	0.741	
ndarray 配列の一致率	0.0183	

ビットマップファイルはピクセル数と色数の乗算によってファイルサイズが決定するため、元画像と生成した画像のファイルサイズに変化はなかった。画像の再現度に関してはヒストグラムの一致率は 74.1%という値となったのに対し、ndarray 配列の一致率は 1.83%という低い値となった。

また、同様のプログラムを用いて JPEG 形式の画像(以下 JPEG 画像)においても同様の方法で実験を行った。実験に用いた JPEG 画像を図 4.4 に、その JPEG 画像を圧縮した画像を図 4.5 に、圧縮した JPEG 画像を元にプログラムで生成した画像を図 4.6 としてそれぞれ以下に示す。

図 4.4　実験に用いた JPEG 画像

図 4.5　図 4.4 を圧縮した画像

図 4.6　プログラムにより生成した画像

図 4.5 に示す圧縮後画像のファイルサイズは 7,958 バイト(圧縮率約 37.8%)となっている。これらの画像に関しても元画像と生成画像を比較した結果を以下の表 4.3 に示す。

表 4.3　元画像(図 4.4)と生成画像(図 4.6)の比較結果

	元画像	生成画像
ファイルサイズ	12,787 バイト	5,660 バイト
ヒストグラムの一致率	0.959	
ndarray 配列の一致率	0.0568	

JPEG ファイルを対象に行った実験ではプログラムで生成した画像のファイルサイズは、元画像と生成した画像のファイルサイズのおよそ半分の値となった。画像の再現度に関してはヒストグラムの一致率は 95.9%、ndarray 配列の一致率は 5.68%という値となり、ビットマップファイルで実験を行ったときに比べてどちらも高い値となった。

次に、既存の圧縮技術との比較のために図 4.1 を JPEG、PNG それぞれの形式に圧縮し、それらの画像と元画像のヒストグラム、ndarray 配列それぞれの一致率を調べまとめたものを以下の表 4.4 に示す。

表 4.4　元画像(図 4.1)と JPEG 画像・PNG 画像の比較結果

	JPEG 画像	PNG 画像
ファイルサイズ	14,323 バイト	43,471 バイト
ヒストグラムの一致率	0.794	1.00
ndarray 配列の一致率	0.0737	1.00

画像の圧縮率は JPEG 圧縮が 70.9%、PNG 圧縮が 11.7%となり、画像の類似度については JPEG 圧縮はヒストグラムの一致率が 79.4%、ndarray 配列の一致率が 7.37%と本実験で生成した画像に近い値になり、PNG 圧縮はヒストグラムの一致率、ndarray 配列の一致率ともに 100%という結果になった。

最後に、Generator で画像を生成する際に必要となる情報を調べた。Generator で画像を生成する際に Generator に入力する情報を以下の表 4.5 に示す。

表 4.5 画像の生成に必要な情報

入力情報
圧縮画像
圧縮画像の高さ
圧縮画像の幅
圧縮画像のチャンネル数
Generator の学習済みパラメータ

上記のうち、圧縮画像の高さ、幅、チャンネル数についてはデータの送信後に OpenCV 等を用いることで得られる数値であるため、画像を送信した後でも取得することができる。したがって、本論文で提案する手法において送信後に画像を復元するために必要となる情報は圧縮画像と Generator の学習済みパラメータの 2 つとなる。本論文で提案する手法を用いて本実験で使用した画像を送信する場合に想定される送信データのデータ量をまとめたものを以下の表 4.6 に示す。

表 4.6 想定される送信データのデータ量

画像	図 4.1	図 4.4
圧縮画像	10,059 バイト	7,958 バイト
Generator の学習済みパラメータ	2,881,136 バイト	2,881,136 バイト
合計	2,891,195 バイト	2,889,094 バイト

上記の通り、想定される送信データのデータ量はビットマップ画像、JPEG 画像どちらの場合においても実験に使用した元画像のファイルサイズよりも大きくなる結果となった。

第 5 章 考察

5.1 考察

今回、画像を JPEG 圧縮と同様の方法で欠損が分かりやすくなる程度まで圧縮し、その画像を利用して GAN による元画像と同じ画像の生成を行った。表 4.1 より、ビットマップ画像を用いて行った実験で生成した画像は、元画像とのヒストグラムの一致率は 74.1%という結果で色合いに関しては大まかな部分は再現できているが、所々に元の画像と色の濃さがわずかに違う、細かい部分の表現が再現できていないといったような元画像との相違点が人の目でも見てわかる程度に存在している結果となった。また、ndarray 配列の一致率に関しては 1.83%と非常に低い値になっており、画像の画素値の構成に関してはそのほとんどが変化する結果となった。JPEG 画像に関しても同様の実験を行ったが、こちらに関してはヒストグラムの一致率が 95.9%と非常に高い数値となっており、色合いに関しては高い精度で再現できていると考えられるが、ndarray 配列の一致率については 5.68%となり、ビットマップ画像の時と同様に非常に低い値となっていた。どちらのパターンでも ndarray 配列の一致率が低い値となっていることから、今回用いたプログラムによる画像の復元では画像の画素値の構成までは正確に復元することができないといえる。また、図 4.1 の画像で実験を行った際は生成画像の色の濃さがわずかに異なる部分(画像内の花の部分)が存在した。圧縮した後も色が分かりやすい部分に関してはその見た目を再現することができているため、元画像と異なっている部分についてはその部分の色の構成が複雑であったことが原因なのではないかと考えられる。したがって、画像を圧縮する際に圧縮率を色の違いを細かく判別できる程度に調整することができれば GAN によって人の目では判別できない程度に元画像に近い画像を生成でき、閲覧を目的とする画像ならば高い品質と高い圧縮率の両方を同時に実現することができるのではないかと考えられる。

続いて、本論文にて提案した手法の実用性についての検討を行う。今回の実験で生成した画像と JPEG 圧縮・PNG 圧縮した画像を比較すると、生成した画像の元画像の再現度は PNG 圧縮した画像ほど高くはならず、比較的 JPEG 圧縮した画像に近くなったことが分かる。したがって、今回の実験で生成した画像の性質は JPEG 圧縮した画像に近いものになったといえる。また、元の画像を圧縮した際のファイルサイズと比較した場合、GAN で画像を復元する場合はそのサイズを JPEG 圧縮・PNG 圧縮した場合に比べて小さくすることができた。送信するデータ量に関しては、今回の実験においては Generator の学習済みパラメータのサイズが元画像のファイルサイズ比べ非常に大きくなった。このことから、本論文で提案する手法は今回実験に用いたようなファイルサイズの小さなデータを送信する際にはあまり有効ではないといえるが、学習したパラメータのサイズを大幅に超えるような大容量のデータや合計するとファイルサイズが学習したパラメータのサイズを大きく超える複数のデータを送信するような状況においては、データを送信するという状況下においてデータの性質を維持しつつ送信するデータ量のみを大きく減らすという

目的は達成することが可能であるため有効であると考えられる。ただし、今回の実験ではデータを学習する際に数時間の時間を要したため、実際にシステムとして実現する際にはこの時間を減らす工夫が必要になると考えられる。

第 6 章 結論

6.1 結論

本研究では圧縮したデータを GAN を利用して復元することで通信データを圧縮することができるかどうか、この手法が実用的であるかどうかについての検討を行った。今回行った実験では画像データに対して圧縮・復元を行い、閲覧用の画像に関してはそれを送信するという状況下において既存の圧縮技術よりも高い圧縮率で、画像の性質をほとんど変えることなく送信することが可能であるという結果を得ることができた。同時に、図 4.1 のように復元した際に画像の色合いがわずかに変化するといったことが発生する可能性があることも分かった。提案手法の実用性については、ファイルサイズが小さなデータに関してはあまり有効ではないが、大容量のデータ、大量のデータといったファイルサイズが非常に大きくなるデータに関しては有効であるということが分かった。

今後の課題としては、どのような条件で色合いに変化が生じてしまうのか、画像の圧縮率はどの程度まで上げることができるか、GAN の学習の際のエポック数等はどの程度が適切か、どうすれば学習にかかる時間を減らすことができるかといったことを模索することが挙げられる。また、今回は画像を対象として実験を行ったが、データという言葉には画像以外も含まれるため、そういったデータに対しても本論文で提案する手法が適用可能かどうかを検証する必要がある。

謝辞

本研究を進めるにあたり、多忙の中ご指導いただきました三好力教授に心から感謝致します。また、研究に関して多くの助言をいただいた三好研究室の皆様に厚く御礼申し上げます。

参考文献

[1] Box Japan.“メール添付はもうやめませんか？その問題点を解説”. Boxsquare.2020-3-11.
https://www.boxsquare.jp/blog/issue-of-attached-file.html ,(参照 2021-10-27)
[2] Bigmac 編集部.” PNG と JPEG の違いとは？あなたの画像に最適な保存形式を！”.Bigmac inc..2019-06-28
https://big-mac.jp/column/what-is-the-difference-between-png-and-jpeg/ ,(参照 2021-10-27).
[3]平内雅則.“GAN：敵対的生成ネットワークとは何か～「教師なし学習」による画像生成”.iMagazine.2018-09-04.
https://www.imagazine.co.jp/gan%EF%BC%9A%E6%95%B5%E5%AF%BE%E7%9A%84%E7%94%9F%E6%88%90%E3%83%8D%E3%83%83%E3%83%88%E3%83%AF%E3%83%BC%E3%82%AF%E3%81%A8%E3%81%AF%E4%BD%95%E3%81%8B%E3%80%80%EF%BD%9E%E3%80%8C%E6%95%99%E5%B8%AB/ ,(参照 2021-10-27).
[4] NegativeMind. “GAN (Generative Adversarial Networks)：敵対的生成ネットワーク”.NegativeMindException -Negative Simulation, Positive Planning--. 2021-06-13.
https://blog.negativemind.com/2019/06/22/generative-adversarial-networks/ , (参照 2021-10-27).
[5]内田奏.”【Intern CV Report】超解像の歴史探訪 -SRGAN 編-”. Sansan Builders Blog Sansan のものづくりを支えるメンバーの技術やデザイン、プロダクトマネジメントの情報を発信.2019-04-29
https://buildersbox.corp-sansan.com/entry/2019/04/29/110000 ,(参照 2021-10-27).
[6]hima zin.” 超解像技術-SRGAN-実装してみた(Tensorflow 2.0) 訓練フェーズ編”. Qiita. 202012-06.
https://qiita.com/hima_zin331/items/fcac94f39493642c145e ,(参照 2021-12-10).
[7]Ryuichi Kato.” OpenCV を利用して画像を圧縮(encode→decode)する”. Qiita. 2018-02-17.
https://qiita.com/ka10ryu1/items/5fed6b4c8f29163d0d65 , (参照 2021-12-01).
[8]OFFECE54.”【Python】OpenCV と NumPy で2つの画像を比較(完全一致、部分一致の比率)”. OFFECE54.2020-11-13.
https://office54.net/python/module/opencv-numpy-compare ,(参照 2021-12-20).
[9] Coding Memorandum.” CrossEntropy と BinaryCrossEntropy について理解する”. Coding Memorandum.2020-12-02.
https://yaakublog.com/crossentropy_binarycrossentropy ,(参照 2021-12-24)

付録

学習用コード[6]

```
import tensorflow as tf
import tensorflow.keras.layers as kl
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.python.keras import backend as K
import cv2
import re
import numpy as np
import matplotlib.pyplot as plt
import argparse as arg
import sys
import os

# Generator
class Generator(tf.keras.Model):
    def __init__(self, input_shape):
        super().__init__()

        input_shape_ps = (input_shape[0], input_shape[1], 64)

        # Pre stage(Down Sampling)
        self.pre = [
            kl.Conv2D(64,  kernel_size=9,  strides=1,  padding="same",
                      input_shape=input_shape),
                      kl.Activation(tf.nn.relu)
        ]

        # Residual Block
        self.res = [
            [
                Res_block(64, input_shape) for _ in range(7)
            ]
        ]

        # Middle stage
        self.middle = [
            kl.Conv2D(64, kernel_size=3, strides=1, padding="same"),
            kl.BatchNormalization()
        ]

        # Pixel Shuffle(Up Sampling)
        self.ps = [
            [
                Pixel_shuffler(128, input_shape_ps) for _ in range(2)
            ],
            kl.Conv2D(3,  kernel_size=9,  strides=4,  padding="same",
                      activation="tanh")
        ]

    def call(self, x):
        # Pre stage
        pre = x
        for layer in self.pre:
            pre = layer(pre)

        # Residual Block
        res = pre
        for layer in self.res:
            for _layer in layer:
                res = _layer(res)

        # Middle stage
        middle = res
        for layer in self.middle:
            middle = layer(middle)
        middle += pre

        # Pixel Shuffle
        out = middle
        for layer in self.ps:
            if isinstance(layer, list):
                for _layer in layer:
                    out = _layer(out)
            else:
                out = layer(out)

        return out


# Discriminator
class Discriminator(tf.keras.Model):
    def __init__(self, input_shape):
        super().__init__()

        self.conv1 = kl.Conv2D(64, kernel_size=3, strides=1,
                               padding="same",
                               input_shape=input_shape)
        self.act1 = kl.Activation(tf.nn.relu)

        self.conv2 = kl.Conv2D(64, kernel_size=3, strides=2,
                               padding="same")
        self.bn1 = kl.BatchNormalization()
        self.act2 = kl.LeakyReLU()

        self.conv3 = kl.Conv2D(128, kernel_size=3, strides=1,
                               padding="same")
        self.bn2 = kl.BatchNormalization()
        self.act3 = kl.LeakyReLU()

        self.conv4 = kl.Conv2D(128, kernel_size=3, strides=2,
                               padding="same")
        self.bn3 = kl.BatchNormalization()
        self.act4 = kl.LeakyReLU()

        self.conv5 = kl.Conv2D(256, kernel_size=3, strides=1,
                               padding="same")
        self.bn4 = kl.BatchNormalization()
        self.act5 = kl.LeakyReLU()

        self.conv6 = kl.Conv2D(256, kernel_size=3, strides=2,
                               padding="same")
        self.bn5 = kl.BatchNormalization()
        self.act6 = kl.LeakyReLU()

        self.conv7 = kl.Conv2D(512, kernel_size=3, strides=1,
                               padding="same")
        self.bn6 = kl.BatchNormalization()
        self.act7 = kl.LeakyReLU()

        self.conv8 = kl.Conv2D(512, kernel_size=3, strides=2,
                               padding="same")
        self.bn7 = kl.BatchNormalization()
        self.act8 = kl.LeakyReLU()

        self.flt = kl.Flatten()

        self.dens1 = kl.Dense(1024, activation=kl.LeakyReLU())
        self.dens2 = kl.Dense(1, activation="sigmoid")

    def call(self, x):
        d1 = self.act1(self.conv1(x))
        d2 = self.act2(self.bn1(self.conv2(d1)))
        d3 = self.act3(self.bn2(self.conv3(d2)))
        d4 = self.act4(self.bn3(self.conv4(d3)))
        d5 = self.act5(self.bn4(self.conv5(d4)))
        d6 = self.act6(self.bn5(self.conv6(d5)))
        d7 = self.act7(self.bn6(self.conv7(d6)))
        d8 = self.act8(self.bn7(self.conv8(d7)))

        d9 = self.dens1(self.flt(d8))
        d10 = self.dens2(d9)

        return d10


# Pixel Shuffle
class Pixel_shuffler(tf.keras.Model):
    def __init__(self, out_ch, input_shape):
        super().__init__()

        self.conv = kl.Conv2D(out_ch, kernel_size=3, strides=1,
                              padding="same",
                              input_shape=input_shape)
        self.act = kl.Activation(tf.nn.relu)

    # forward proc
    def call(self, x):
        d1 = self.conv(x)
        d2 = self.act(tf.nn.depth_to_space(d1, 2))

        return d2


# Residual Block
class Res_block(tf.keras.Model):
    def __init__(self, ch, input_shape):
        super().__init__()

        self.conv1 = kl.Conv2D(ch, kernel_size=3, strides=1,
                               padding="same",
                               input_shape=input_shape)
        self.bn1 = kl.BatchNormalization()
        self.av1 = kl.Activation(tf.nn.relu)

        self.conv2 = kl.Conv2D(ch, kernel_size=3, strides=1,
                               padding="same")
        self.bn2 = kl.BatchNormalization()

        self.add = kl.Add()

    def call(self, x):
```

```
        d1 = self.av1(self.bn1(self.conv1(x)))
        d2 = self.bn2(self.conv2(d1))

        return self.add([x, d2])


# Train
class trainer():
    def __init__(self, c_img, o_img):
        c_shape = c_img.shape    # Compressed Image shape
        o_shape = o_img.shape    # OriginalImage shape

        # Content Loss Model setup
        input_tensor = tf.keras.Input(shape=o_shape)
        self.vgg = VGG16(include_top=False, input_tensor=input_tensor)
        self.vgg.trainable = False
        # VGG16 block3_conv3 output
        self.vgg.outputs = [self.vgg.layers[9].output]

        # Content Loss Model
        self.cl_model = tf.keras.Model(input_tensor, self.vgg.outputs)

        # Discriminator
        discriminator_ = Discriminator(o_shape)
        inputs = tf.keras.Input(shape=o_shape)
        outputs = discriminator_(inputs)
        self.discriminator = tf.keras.Model(inputs=inputs, outputs=outputs)
        self.discriminator.compile(optimizer=tf.keras.optimizers.Adam(),

                                   loss=tf.keras.losses.BinaryCrossentropy(),
                                        metrics=['accuracy'])

        # Generator
        self.generator = Generator(c_shape)

        # Combined Model setup
        c_input = tf.keras.Input(shape=c_shape)
        ge_output = self.generator(c_input)

        self.discriminator.trainable = False    # Discriminator train Disable
        d_fake = self.discriminator(ge_output)

        # GAN Model
        self.gan    =    tf.keras.Model(inputs=c_input,    outputs=[ge_output,
                                    d_fake])
        self.gan.compile(optimizer=tf.keras.optimizers.Adam(),
                                    loss=[self.Content_loss,
                                    tf.keras.losses.BinaryCrossentropy()],
                                    loss_weights=[1., 1e-3])

    # Content loss
    def Content_loss(self, o_img, ge_img):
        return K.mean(K.abs(K.square(self.cl_model(o_img)
                                    -    self.cl_model(ge_img))))

    # PSNR
    def psnr(self, o_img, ge_img):
        return cv2.PSNR(o_img, ge_img)

    def train(self, c_imgs, o_imgs, out_path, batch_size, epoch):
        g_loss_plt = []
        d_loss_plt = []
        path = os.path.join(out_path, "graph6.jpg")
        plt.figure(figsize=(12.8, 8.0), dpi=100)

        h_batch = int(batch_size / 2)

        real_lab = np.ones((h_batch, 1))    # Original image label
        fake_lab = np.zeros((h_batch, 1))    #Generation image
                                    label(Discriminator side)
        gan_lab = np.ones((h_batch, 1))

        # train run
        for epoch in range(epoch):
            # - Train Discriminator -

            # Original image random pickups
            idx = np.random.randint(0, o_imgs.shape[0], h_batch)
            o_img = o_imgs[idx]

            # Compressed image random pickups
            c_img = c_imgs[idx]

            # Discriminator enabled train
            self.discriminator.trainable = True

            # train by Original image
            d_real_loss = self.discriminator.train_on_batch(o_img, real_lab)


            # train by Geration image
            ge_img = self.generator.predict(c_img)
            d_fake_loss = self.discriminator.train_on_batch(ge_img, fake_lab)

            # Discriminator average loss
            d_loss = 0.5 * np.add(d_real_loss, d_fake_loss)

            # - Train Generator -

            # Original image random pickups
            idx = np.random.randint(0, o_imgs.shape[0], h_batch)
            o_img = o_imgs[idx]

            # Compressed image random pickups
            c_img = c_imgs[idx]

            # train by Generator
            self.discriminator.trainable = False
            g_loss = self.gan.train_on_batch(c_img, [o_img, gan_lab])

            # Epoch num, Discriminator/Generator loss, PSNR
            print("Epoch:    {0}    D_loss:    {1:.7f}    G_loss:    {2:.3f}    PSNR:
                                {3:.3f}".format(epoch + 1, d_loss[0], g_loss[0],
                                self.psnr(o_img, ge_img)))

            d_loss_plt.append(d_loss[0])
            g_loss_plt.append(g_loss[0])

            # Plotting the loss value
            if (epoch + 1) % 50 == 0:
                plt.plot(d_loss_plt)
                plt.plot(g_loss_plt)
                plt.savefig(path, bbox_inches='tight', pad_inches=0.1)

        print("___Training finished¥n¥n")

        # Parameter-File and Graph Saving
        print("___Saving parameter...")
        self.generator.save_weights(os.path.join(out_path, "parameter.h5"))

        plt.plot(d_loss_plt, label="D_loss")
        plt.plot(g_loss_plt, label="G_loss")
        plt.savefig(path, bbox_inches='tight', pad_inches=0.1)
        print("___Successfully completed¥n¥n")

# Load dataset
def load_dataset(data_dir, c_data_dir, he, wi):
    Quality = 5
    encode_param = [int(cv2.IMWRITE_JPEG_QUALITY), Quality]
    print("¥n___Creating a dataset...")

    prc = ['/', '-', '¥¥', '|']
    cnt = 0

    print("Number of image in a directory: {}".format(len(os.listdir(data_d
                                ir))))

    c_imgs = []
    o_imgs = []

    #Load nomal images
    for c in os.listdir(data_dir):
        d = os.path.join(data_dir, c)

        _, ext = os.path.splitext(c)
        if ext.lower() not in ['.jpg', '.png', '.bmp']:
            continue

        img = cv2.imread(d)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, (he, wi))

        o_imgs.append(img)

        cnt += 1

    #Load compressed images
    for c in os.listdir(c_data_dir):
        d = os.path.join(c_data_dir, c)

        _, ext = os.path.splitext(c)
        if ext.lower() not in ['.jpg', '.png', '.bmp']:
            continue

        img = cv2.imread(d)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, (he, wi))

        c_imgs.append(img)

        cnt += 1

    # Compressed image
    c_imgs = tf.convert_to_tensor(c_imgs, np.float32)
    c_imgs = (c_imgs.numpy() - 127.5) / 127.5


    # Original image
    o_imgs = tf.convert_to_tensor(o_imgs, np.float32)
    o_imgs = (o_imgs.numpy() - 127.5) / 127.5

    print("¥n___Successfully completed loading images¥n")

    return c_imgs, o_imgs
```

```
def main():
    # Command line option
    parser = arg.ArgumentParser(description='GAN training')
    parser.add_argument('--data_dir', '-d', type=str, default=None,
                        help='Specify the image folder path (If not specified,
                            an error)')
    parser.add_argument('--c_data_dir', '-r', type=str, default=None,
                        help='Specify the image folder path (If not specified,
                            an error)')
    parser.add_argument('--out', '-o', type=str,
                        default=os.path.dirname(os.path.abspath(__file__)),
                        help='Specify where to save parameters
                            (default: ./parameter.h5)')
    parser.add_argument('--batch_size', '-b', type=int, default=32,
                        help='Specify the mini-batch size (default: 32)')
    parser.add_argument('--epoch', '-e', type=int, default=1000,
                        help='Specify the number of times to train (default:
                            1000)')
    parser.add_argument('--he', '-he', type=int, default=128,
                        help='Resize height (default: 128)')
    parser.add_argument('--wi', '-wi', type=int, default=128,
                        help='Resize width (default: 128)')
    args = parser.parse_args()

    # Image folder not specified. -> Exception
    if args.data_dir is None:
        print("¥nException: Folder not specified.¥n")
        sys.exit()
    # An image folder that does not exist was specified. -> Exception
    if os.path.exists(args.data_dir) is False:
        print("¥nException: Folder ¥"{}¥" is
                            not found.¥n".format(args.data_dir))
        sys.exit()
    # When 0 is entered for either width/height or Reduction ratio.
                             -> Exception
    if args.he == 0 or args.wi == 0:
        print("¥nException: Invalid value has been entered.¥n")
        sys.exit()

    # Create output folder (If the folder exists, it will not be created.)
    os.makedirs(args.out, exist_ok=True)

    # Setting info
    print("=== Setting information ===")
    print("# Images folder: {}".format(os.path.abspath(args.data_dir)))
    print("# Compressed images folder: {}".format(os.path.abspath(args.c_
                            data_dir)))
    print("# Output folder: {}".format(args.out))
    print("# Minibatch-size: {}".format(args.batch_size))
    print("# Epoch: {}".format(args.epoch))
    print("")
    print("# Height: {}".format(args.he))
    print("# Width: {}".format(args.wi))
    print("============================")

    # dataset creation
    c_imgs, o_imgs = load_dataset(args.data_dir, args.c_data_dir
                            args.he,    args.wi)

    print("___Start training...")
    Trainer = trainer(c_imgs[0], o_imgs[0])
    Trainer.train(c_imgs, o_imgs, out_path=args.out,
                            batch_size=args.batch_size, epoch=args.epoch)


if __name__ == '__main__':
    main()
```

生成用コード[6]

```
import tensorflow as tf
import tensorflow.keras.layers as kl
import cv2
from PIL import Image
import numpy as np
import argparse as arg
import os
import sys

# Generator
class Generator(tf.keras.Model):
    def __init__(self, input_shape):
        super().__init__()

        input_shape_ps = (input_shape[0], input_shape[1], 64)

        # Pre stage(Down Sampling)
        self.pre = [
            kl.Conv2D(64, kernel_size=9, strides=1,
                      padding="same", input_shape=input_shape),
            kl.Activation(tf.nn.relu)
        ]

        # Residual Block
        self.res = [
            [
                Res_block(64, input_shape) for _ in range(7)
            ]
        ]

        # Middle stage
        self.middle = [
            kl.Conv2D(64, kernel_size=3, strides=1, padding="same"),
            kl.BatchNormalization()
        ]

        # Pixel Shuffle(Up Sampling)
        self.ps = [
            [
                Pixel_shuffler(128, input_shape_ps) for _ in range(2)
            ],
            kl.Conv2D(3, kernel_size=9, strides=4, padding="same",
                      activation="tanh")
        ]

    def call(self, x):
        # Pre stage
        pre = x
        for layer in self.pre:
            pre = layer(pre)

        # Residual Block
        res = pre
        for layer in self.res:
            for _layer in layer:
                res = _layer(res)

        # Middle stage
        middle = res
        for layer in self.middle:
            middle = layer(middle)
        middle += pre

        # Pixel Shuffle
        out = middle
        for layer in self.ps:
            if isinstance(layer, list):
                for _layer in layer:
                    out = _layer(out)
            else:
                out = layer(out)

        return out

# Pixel Shuffle
class Pixel_shuffler(tf.keras.Model):
    def __init__(self, out_ch, input_shape):
        super().__init__()

        self.conv = kl.Conv2D(out_ch, kernel_size=3, strides=1,
                              padding="same",
                              input_shape=input_shape)
        self.act = kl.Activation(tf.nn.relu)

    # forward proc
    def call(self, x):
        d1 = self.conv(x)
        d2 = self.act(tf.nn.depth_to_space(d1, 2))

        return d2

# Residual Block
class Res_block(tf.keras.Model):
    def __init__(self, ch, input_shape):
        super().__init__()

        self.conv1 = kl.Conv2D(ch, kernel_size=3, strides=1,
                               padding="same",
                               input_shape=input_shape)
        self.bn1 = kl.BatchNormalization()
        self.av1 = kl.Activation(tf.nn.relu)

        self.conv2 = kl.Conv2D(ch, kernel_size=3, strides=1,
                               padding="same")
        self.bn2 = kl.BatchNormalization()

        self.add = kl.Add()

    def call(self, x):
        d1 = self.av1(self.bn1(self.conv1(x)))
        d2 = self.bn2(self.conv2(d1))

        return self.add([x, d2])

def main():
    # Command line option
    parser = arg.ArgumentParser(description=' GAN prediction')
    parser.add_argument('--param', '-p', type=str, default=None,
                        help='Specify learned parameters (If not specified,
                            an error)')
    parser.add_argument('--c_data_img', '-c', type=str, default=None,
                        help='Specify an image file (If not specified, an
                            error)')
    parser.add_argument('--out', '-o', type=str,
                        default=os.path.join(os.path.dirname(os.path.a
                        bspath(__file__)), "result"),
                        help='Specify the destination (default: ./result)')
    args = parser.parse_args()

    # Parameter-File not specified. -> Exception
    if args.param is None:
        print("¥nException: Trained Parameter-File not specified.¥n")
        sys.exit()
    # An Parameter-File that does not exist was specified. -> Exception
    if os.path.exists(args.param) is False:
        print("¥nException: Trained Parameter-File {} is not
                            found.¥n".format(args.param))
        sys.exit()
    # Image not specified. -> Exception
    if args.c_data_img is None:
        print("¥nException: Compressed Image not specified.¥n")
        sys.exit()

    # Setting info
    print("=== Setting information ===")
    print("# Trained Prameter-File: {}".format(os.path.abspath(args.param)))
    print("# Compressed Image: {}".format(args.c_data_img))
    print("# Output folder: {}".format(args.out))
    print("============================")

    # Load compressed image
    c_img = cv2.imread(args.c_data_img)
    he, wi, ch = c_img.shape

    # Create output folder (If the folder exists, it will not be created.)
    os.makedirs(args.out, exist_ok=True)

    # Network Setup
    model = Generator(input_shape=(he, wi, ch))
    model.build((None, he, wi, ch))
    model.load_weights(args.param)

    # Image processing
    c_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    c_img = cv2.resize(c_img, (he, wi))
    c_img = tf.convert_to_tensor(c_img, np.float32)
    c_img = tf.convert_to_tensor(c_img, np.float32)
    c_img = (c_img - 127.5) / 127.5
    c_img = c_img[np.newaxis, :, :, :]

    # Generate Image
    re = model.predict(c_img)


# Generation Image processing
    re = np.reshape(re, (he, wi, ch))
    re = re * 127.5 + 127.5
    re = np.clip(re, 0.0, 255.0)

    # Generation Image output
    n = int(1)
    while True:
        if os.path.exists("./result/result_{}.bmp".format(n)):
            n += 1
        else:
            g_img.save(os.path.join(args.out, "result_{}.bmp".format(n)))
            break

if __name__ == "__main__":
    main()
```