広島県立総合技術研究所東部工業技術センター研究報告　No. 37（2024）
技術ノート

深層学習モデルによるデータ生成（第１報）

ノイズ除去拡散確率モデル（第１報）

廣川　勝久

Data Generation Utilizing Deep Learning Techniques (Ⅰ)

Denoising Diffusion Probabilistic Model (Ⅰ)

HIROKAWA Katsuhisa

学習したデータから新たなデータの生成を可能とする複数の深層学習モデルが提案されている。これらの生成データを深層学習のデータ拡張に応用することで、各種深層学習の汎化能力を向上させる可能性があると考えられる。本報告では、ノイズ除去拡散確率モデル[arXiv:2006.11239 (2020)]を実際に実装し、そのモデルから生成される画像を拡散の度合いに応じて評価した。ノイズの拡散の度合いを調整することにより、1 枚のノイズ拡散画像から全く異なる画像や高い類似性を持つ画像を複数生成する方法について述べる。

キーワード：Denoising Diffusion Probabilistic Model, Deep Learning, Data Generation, 拡散モデル, 生成 AI

1. 緒　　言

近年、大手 IT 企業が提供する深層学習モデルは急速に巨大化しており、学習パラメータ数や必要な学習データ量、そして計算時間が増大している。このような深層学習モデルの学習においては、データ拡張技術などを用いて限られた学習データを増強し、汎化能力の向上を図る試みが行われている。

一方で、敵対的生成ネットワーク（**G**enerative **A**dversarial **N**etworks : **GAN**）[1,2] やオートエンコーダ（**A**uto **E**ncoder: **AE**）[3, 4] といった既存の学習データから新たなデータを生成する生成モデルが数多く開発されている。これらのモデルでは、学習データを生成モデル内の潜在変数空間にクラスタリングし、この潜在変数空間の特定の座標を指定することにより新しいデータを生成することが可能である。この生成されたデータは潜在変数空間座標が示すクラスタの特徴を反映している。

本稿では、このような状況を踏まえ、データ拡張に生成モデルの利用を検討するため、ノイズ除去拡散確率モデル(**D**enoising **D**iffusion **P**robabilistic **M**odel: **DDPM**)[5]を実際に実装し、新しい画像の生成について評価を行った結果を報告する。初期画像に対してノイズ拡散を行った場合、拡散の度合いに応じてモデルから生成される画像の特性を評価した。評価結果に基づき、ノイズの拡散の度合いを調整することにより、ノイズ拡散画像から全く異なる画像や類似性の高い画像を生成する方法を示す。

2. 生成モデル

2.1 データ拡張と生成モデル

異常検知を目的とする教師あり深層学習モデルにおいては，学習に必要な異常データが正常データに対して極めて少なく、学習が困難な場合が多い。また、運用時には教師データに存在しない未知の異常データが発生する可能性があり、モデルの精度の低下を招く一因となる。

深層学習モデルの汎化能力の向上を目的としたデータ拡張 [6] が数多く提案されているが、実世界では発生し得ないようなデータを合成し学習させることもあり、非現実的な学習が行われる可能性もある。

一方、生成モデルを活用することで学習した正常データから異常データを同数程度生成することができれば、それを異常検知の学習データに利用することが可能と思われる。またデータ拡張にこの手法を応用した場合、実在するデータの類似データが利用可能となるため、深層学習の分類精度が向上することが予想される。本稿ではこのような、データ拡張や異常検知のためのデータ生成を目的として、DDPM を選択し、その有効性を評価する。

2.2 Denoising Diffusion Probabilistic Model: DDPM

DDPM による画像のノイズ化は、わずかなガウシアンノイズを加える過程を繰り返すことで行なわれる。**図 1** にその過程を示す。今画像をXとし、初期画像X_0にガウシアンノイズを繰り返し加えながら、最終的にX_T画像が得られる場合を考える。この時、X_{t-1}からX_tへの画像生成は次式により与えられる。

$$X_t = \sqrt{1-\beta_t}X_{t-1} + \sqrt{\beta_t}\varepsilon \quad (1)$$
$$\because \varepsilon = N(0, I)$$

2024. 8. 27 加工技術研究部　受理

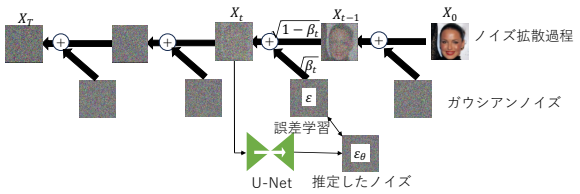

図 1　画像のノイズ拡散過程と学習

ガウシアンノイズεは$\sqrt{\beta_t}$だけ減衰され、X_{t-1}に加算される。また、X_{t-1}も加算されるガウシアンノイズを考慮して$\sqrt{1-\beta_t}$だけ減衰させる。さらに、(1) 式を用いることで、初期画像X_0から直接X_tを計算することが可能である。

$$X_t = X_{t-1}X_{t-2}\cdots X_0 = \sqrt{\bar{\alpha}_t}X_0 + \sqrt{(1-\bar{\alpha}_t)}\varepsilon \qquad (2)$$

$$\because \bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$$

これにより、X_tの生成には、初期画像X_0にガウシアンノイズを直接加えることで計算できるため，逐次的な計算を必要とせず、DDPM の学習速度を向上させている。

DDPM では、加えられるガウシアンノイズを推定可能な状態まで学習を行う。この推定により、ノイズが加えられた画像から少しずつノイズを除去し、画像を復元する過程を実現する。X_tの画像生成は(2)式により与えられ、加えられたノイズをεと DDPM が推定したノイズをε_θとの差を学習誤差∇_θとして次式により定義する。この誤差を最小化するよう学習が進められる。

$$\nabla_\theta = \left\| \varepsilon - \varepsilon_\theta\left(\sqrt{\bar{\alpha}_t}X_0 + \sqrt{(1-\bar{\alpha}_t)}\varepsilon, t\right)\right\|^2 \qquad (3)$$

学習が進むと、DDPM では与えられた画像X_tからX_{t-1}へ加えられたノイズを推定し、ノイズを除去することで逆変換が可能となる。生成過程では、次式によりノイズ除去を行い、逆拡散過程をたどる。

$$X_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(X_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\varepsilon_0(X_t, t)\right) + \sigma_t z \qquad (4)$$

$$\because z = \begin{cases} N(0,I), & t > 1 \\ 0, & else \end{cases}$$

$$\sigma_t{}^2 = \beta_t \quad or \quad \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$$

このノイズ除去を最終画像X_Tから初期画像X_0まで連続的に繰り返すことにより DDPM では新しい画像データが生成する。

2.3 DDPM の構造とノイズ拡散位置情報

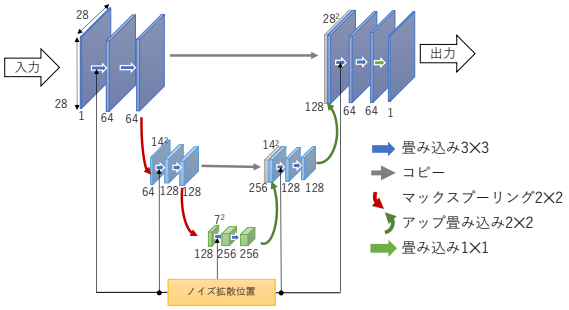

図 2　DDPM の構造

ガウシアンノイズを学習する DDPM には、画像とノイズ拡散過程の位置情報を入力に持つ U-Net[7]が実装されている。この U-Net はノイズ拡散過程の全ての位置で同じものを繰り返し使用しながら学習を行う。**図 2** に DDPM に実装した U-Net の構造を示す。DDPM の U-Net はセマンテックセグメンテーションのものと比較して、レイヤー数は少ない構造となっている。また、ノイズ拡散過程の位置情報を各レイヤーの畳み込み 1 層目に加算し、位置情報を学習する。拡散過程の位置tに対する情報は d 次元のベクトルデータとして与えられる [8]。

$$PE(t,2i) = \sin\left(\frac{t}{10000^{\frac{2i}{d}}}\right)$$
$$PE(t,2i+1) = \cos\left(\frac{t}{10000^{\frac{2i}{d}}}\right) \qquad (5)$$

3. DDPM の実装

3.1　画像データ

本研究では、白黒画像データとして **MNIST**：Modified National Institute of Standards and Technology のデータベースによる手書き数字を [9]、カラー画像には Celeb Faces Attributes (CelebA) Dataset[10]による顔画像を用いた。MNIST は 28 ピクセル×28 ピクセル、CelebA は 64 ピクセル×64 ピクセルで、それぞれ学習を行った。

3.2 DDPM の実装

DDPM に実装した U-Net のソースコードを次に示す。この U-Net では拡散位置の情報を各レイヤーに加えて学習した。CelebA の実験では、エンコードとデコードに各 1 層を追加した構造を採用した。

```
class UNET(nn.Module):
    def __init__(self):
        super().__init__()
        self.layer1 = DownConv(classter1, classter2)
        self.layer2 = DownConv(classter2, classter3)
        self.layer3 = UpConv(classter3, classter4,
                             classter5, flag=0)
        self.layer4 = UpConv(classter3 + classter5,
                             classter5, classter6, flag=0)
        self.layer5 = UpConv(classter2 + classter6,
                             classter6, channel_num, flag=1)
    def forward(self, img, position):
        org1, down1 = self.layer1(img, position)
```

```
        org2, down2 = self.layer2(down1, position)
        up3 = self.layer3(down2, down2, 0, position)
        up4 = self.layer4(up3, org2, Layer3size, position)
        img = self.layer5(up4, org1, Layer4size, position)
        return img
```

(5) 式の拡散位置情報の以下のように行った。次元数は 100 程度設定した。

```
def position_embedding(time, pos_dim):
    PE = torch.zeros((time, pos_dim))
    n = torch.tensor(10000)
    for t in range(time):
        for i in range(pos_dim // 2):
            co = t / torch.pow(n, 2 * i / pos_dim)
            PE[t, 2 * i] = torch.sin(co)
            PE[t, 2 * i + 1] = torch.cos(co)
    return PE
```

拡散位置情報は全結合層のニューラネットワーク使って出力数を U-Net のクラスタ数に一致させる。畳み込み層では U-Net の各クラスタにその拡散位置情報を加える。

```
class Convolution(nn.Module):
    def __init__(self, L1, L2):
        super().__init__()
        self.position = PositionEncoder(position_dim, L1)
        self.conv1 = nn.Conv2d(L1, L2,
                               kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(L2, L2,
                               kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(L2)
        self.bn2 = nn.BatchNorm2d(L2)
        self.relu = nn.GELU)
    def forward(self, img, positions):
        positions = self.position(positions)
        positions = positions.view(img.size(0),
                           img.size(1), 1, 1)
        h = self.conv1(img + positions)
        h = self.relu(self.bn1(h))
        h = self.conv2(h)
        h = self.relu(self.bn2(h))
        return h
```

CelebA データの実験では BatchNorm2d の代わりに GroupNorm を用いた。

```
class PositionEncoder(nn.Module):
    def __init__(self, time_dim, ch):
        super().__init__()
        self.fc1 = nn.Linear(time_dim, ch)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(ch, ch)
    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x
```

オリジナル画像のノイズ付加は(2)式に従って全ての拡散位置に対する$\bar{\alpha}_t$を計算後、要求された拡散位置の画像のみを求める。

```
def noise(x, eps, beta, t):
    alpha = 1 - beta
    alpha_av = torch.cumprod(alpha, dim=0)
    alpha_avt = alpha_av[t].view(x.size(0), 1, 1, 1)
    x_t = torch.sqrt(alpha_avt) * x
          + torch.sqrt(1 - alpha_avt) * eps
    return x_t
```

ノイズ除去も同様に全ての$\bar{\alpha}_t$を計算後、拡散位置の画像のみのノイズを(4)式に従い除去する。

```
def denoise(x, out, t, beta):
    alpha = 1 - beta
    alpha_av = torch.cumprod(alpha, dim=0)
    alpha_t = alpha[t].view(x.size(0), 1, 1, 1)
    alpha_avt = alpha_av[t].view(x.size(0), 1, 1, 1)
    a = 1 / torch.sqrt(alpha_t)
    b = (1 - alpha_t) / torch.sqrt(1 - alpha_avt)
    x_t = a * (x - b * out)
    if t[0] == 0:
        return x_t
    alpha_avt_pre = alpha_av[t - 1].view(x.size(0), 1, 1,
1)
    sigma = torch.sqrt((1 - alpha_t)
          * (1 - alpha_avt_pre) / (1 - alpha_avt))
    noise = torch.randn_like(x)
    x_t = x_t + sigma * noise
    return x_t
```

メインプログラムでは、拡散位置をランダムに設定し、ノイズ画像から付加されたガウシアンノイズを推定するようモデルを学習させた。

```
        base_noise = torch.randn_like(images)
        select_time = torch.randint( low=0, high=TIME,
                                    size=(batch_size,))
        pe = time_position[select_time, :]
        x = noise(images, base_noise, BETA_t, select_time)

        output = model(x, pe)
        loss = criterion(output, base_noise)
        loss.backward()
        optimizer.step()
```

4. 実験結果

4.1 実験パラメータ

DDPM におけるノイズ拡散のステップ回数は T=1000 と設定した。減衰係数は$\beta_1 = 0.0001$から$\beta_T = 0.02$まで線形的に増加させた。また、画像の値は、-1～1 の範囲に規格化した。MNIST 画像の学習では、各レイヤーのクラスター数は**図 2** の設定とした。

4.2 MNIST の実験結果

まず、DDPM に 6000 枚の MNIST 画像を用いて 1000 エポックの学習を行った。最適化関数には Adam を用いた。学習後、未学習の画像からノイズ拡散画像を生成し、ノイズ除去過程を経て画像データを生成した画像を**図 3** に示す。DDPM では、学習に用いたデータの特徴を持つ持ちながら、ノイズ拡散に用いた未学習データとは異なるデータが生成される。比較画像として DCGAN による生成画像データを**図 4** に示す。DCGAN との比較でも DDPM は同等の品質の画像を生成できることが確認された。

4.3 CelebA の実験結果

次に CelebA 画像の実験結果について示す。CelebA の学習には**図 2** の U-Net にレイヤーをエンコード、デコードとも 1 層加えた構造を用いた。CelebA の約 20 万人の画像から 15000 人をランダムに選び学習を行った。学習後、評価用の画像より生成したノイズ拡散画像から画像を生成したものを**図 5** に示す。実験結果は MNIST 画像の実験と同様に、初期画像とは異なる人物画像が生成された。**D**eep **C**onvolutional GAN: **DCGAN** による比較画像を**図 6** に示す。CelebA 画像の画像についても DDPM の画像生成能力は DCGAN と同等の結果が得られた。

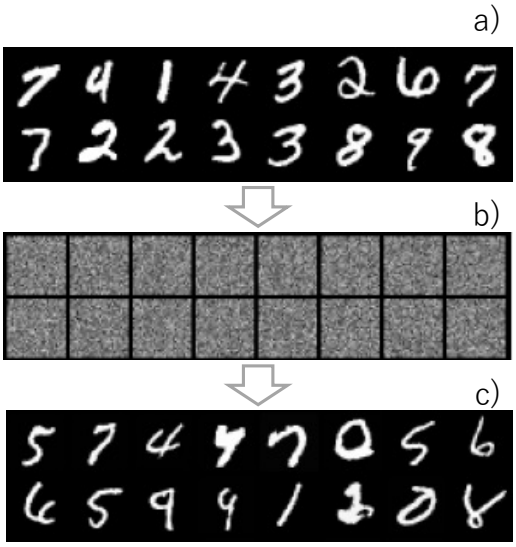

図 3　DDPM による MNIST の画像生成
a) 未学習画像，b) ノイズ拡散画像
c) 生成画像

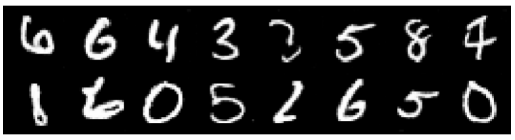

図 4　DCGAN による MNIST の画像生成

さらに、ノイズ拡散画像から複数の画像を生成する実験を行った。1 枚のノイズ拡散画像を複製し、それぞれにノイズ除去を適用した結果、全く異なる人物の画像が生成された（**図 7**）。生成された画像間に相関関係は無いと考えられる。一方**図 8** の、ノイズ拡散位置 t=300 から生成された画像では，ノイズ拡散前の初期画像に似た特徴を持つ画像が生成された。ただし細部が少しずつ異なる。これにより、DDPM ではノイズ拡散位置を調整することで、初期画像との類似性を調整することが可能であることが示された。

図 9 は Cutmix[6]の手法を用いて 2 枚の画像を合成し，t=300 までノイズ拡散させた後、ノイズ除去により画像を生成した結果を示す。初期画像では、それぞれの画像の境界が鮮明であるが、生成画像では滑らかに調整された境界を持つ画像が得られた。この結果から、DDPM を用いることで学習画像の中から、初期画像の一部を他の画像に置換したデータ拡張が可能であることが分かった。

5. 結　　言

本研究では、データ拡張に生成モデルを利用するため、ノイズ除去拡散確率モデル(DDPM)を実際に実装し生成された画像について評価を行った。実験の結果、初期画像からノイズ拡散を最終ステップまで行った場合、1 枚の拡散画像から、生成された複数の画像には、相互に全く相

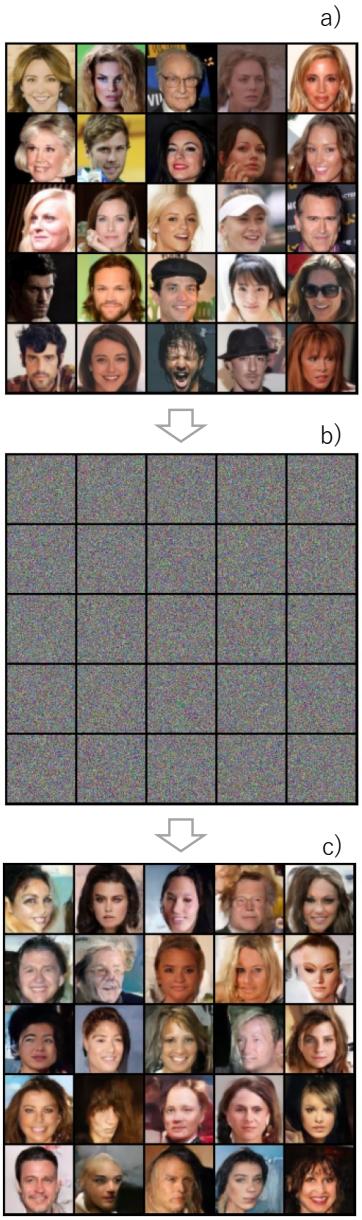

図 5　DDPM による CelebA の画像生成
a) 初期画像，b) ノイズ拡散画像
c) 生成画像

図 6　DCGAN による CelebA の画像生成

関が見られないことが確認された。また、ノイズ拡散の途中段階から生成された画像には細部の異なる類似した画像が複数枚得られた。さらに、画像の一部に他の画像を貼り付けた合成画像から生成された場合、それぞれの画像の特徴を保持しつつ、それぞれの画像の境界が滑らかに調整された画像が生成されることを確認した。

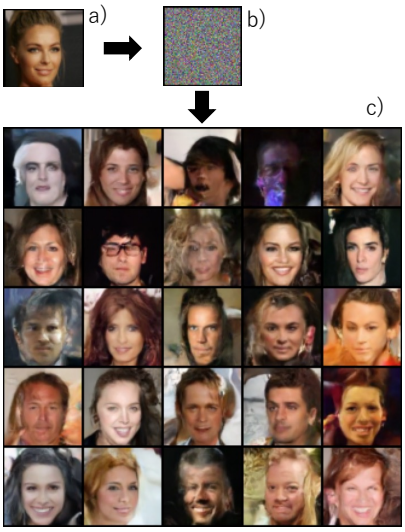

図 7　ノイズ拡散画像からの画像生成(t=1000)
a) 初期画像，b) ノイズ拡散画像
c) 生成画像

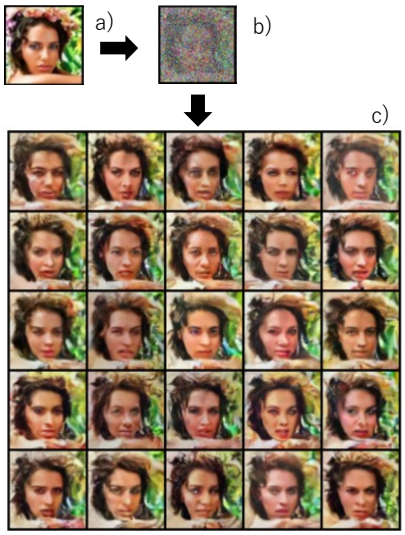

図 8　ノイズ拡散画像からの画像生成(t=300)
a) 初期画像，b) ノイズ拡散画像
c) 生成画像

このようにノイズ除去拡散確率モデルを用いた画像生成では、ノイズ拡散の割合や画像の貼り付けなどにより、実在するような多様な画像を生成することができ、深層学習モデルのデータ拡張に応用できる可能性が示された。特に、異常検知などの学習では少ない異常データから、ノイズ除去拡散確率モデルを利用することにより、現実に発生する可能性がある異常データを多数発生する手法として期待される。

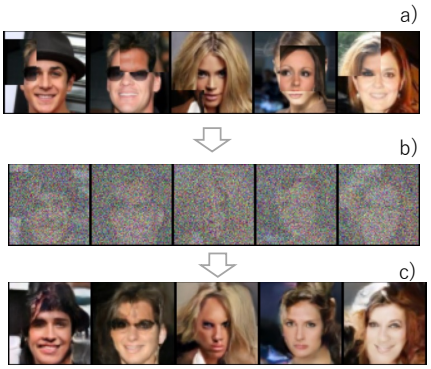

図 9　Cutmix による画像生成(t=300)
a) 初期画像，b) ノイズ拡散画像
c) 生成画像

文　　献

1) Jonathan Ho and Stefano Ermon：Generative adversarial imitation learning，arXiv:**1606.03476**（2016）.
2) 廣川 勝久：広島県立総合技術研究所東部工業技術センター研究報告，強化学習における各種手法の比較（第 3 報）**35**（2022）.
3) 廣川 勝久：広島県立総合技術研究所東部工業技術センター研究報告，深層学習による異常検知手法の簡単な比較（第 1 報）**35**（2022）.
4) 廣川 勝久：広島県立総合技術研究所東部工業技術センター研究報告，深層学習による異常検知手法の簡単な比較（第 2 報）**36**（2023）.
5) Jonathan Ho, Ajay Jain and Pieter Abbeel : Denoising Diffusion Probabilistic Models, arXiv:**2006.11239** (2020).
6) 廣川 勝久，花房 龍男，中濱 久雄：広島県立総合技術研究所東部工業技術センター研究報告，深層学習による画像の領域分割（第 2 報）**37**（2024）.
7) 廣川 勝久，花房 龍男，中濱 久雄：広島県立総合技術研究所東部工業技術センター研究報告，深層学習による画像の領域分割（第 1 報）**36**（2023）.
8) Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin: Attention Is All You Need, arXiv:**1706.03762** (2017).
9) THE MNIST DATABASE of handwritten digits：http://yann.lecun.com/exdb/mnist/.
10) WWW: CelebFaces Attributes (CelebA), https://www.kaggle.com/datasets/jessicali9530/celeba-dataset/.