OpenVINO™ を使用したモデル最適化のための合成データ生成 DL モデルのトレーニング - パート 1

この記事は、Medium に公開されている「Train a DL model for synthetic data generation for model optimization with OpenVINO — Part 1」の日本語参考訳です。原文は更新される可能性があります。原文と翻訳文の内容が異なる場合は原文を優先してください。

敵対的生成ネットワーク (GAN) は、ディープラーニングの分野に革命をもたらしました。GAN は、敵対的トレーニングからランダムノイズを入力とした合成データの生成まで、いくつかのアプリケーションを備えた優れたフレームワークであり、GAN で生成されたデータを使用したモデル最適化もその 1 つです。

この記事は、インテルの OpenVINO™ ツールキットの後援の下に開発された、Google* Summer of Code 2022 プロジェクト「モデル最適化のための合成データ生成 DL モデルのトレーニング」に関するブログのパート 1 です (パート 2 はこちら)。プロジェクトは 2 つのパートで構成されます。このパート 1 では、軽量のディープラーニング・モデルをトレーニングして合成画像を生成します。パート 2 では、パート 1 の事前トレーニング済みモデルを使用して、CIFAR-10 向けの合成画像のデータセットを生成します。このデータセットを、OpenVINO™ のトレーニング後の最適化ツールによるモデル最適化に使用します。その後、さまざまなコンピューター・ビジョン・モデルで、8 ビットのトレーニング後の量子化手法のパフォーマンスを評価します。

プロジェクトのページ: Google* Summer of Code (英語)、GitHub* リポジトリー (英語)

はじめに

画像生成

画像生成は、長い間研究されてきたコンピューター・ビジョンのタスクです。敵対的生成ネットワーク (GAN) [1] は、入力としてランダム・ノイズ・ベクトルのみを使用して合成画像を生成できます (図 1)。最近のモデル [2] では、ImageNet [3] のような複雑なデータセットでも、実際の画像と見分けがつかない画像を生成できるようになりました。これは興味深いタスクですが、実用的で複雑なのは、条件付き画像生成のタスクです。条件付き画像生成のタスクとは、生成モデルを使用して、入力条件に基づいてリアルな外観の画像を合成するコンピューター・ビジョンのタスクを指します (図 2)。条件には、属性、文章による説明、クラスラベルなどが含まれます。このトピックの最近の進歩により、高品質で高忠実度の画像を生成できるモデル [4][5] が示されましたが、パラメーターの数が非常に多く、膨大な計算リソースが必要になります。

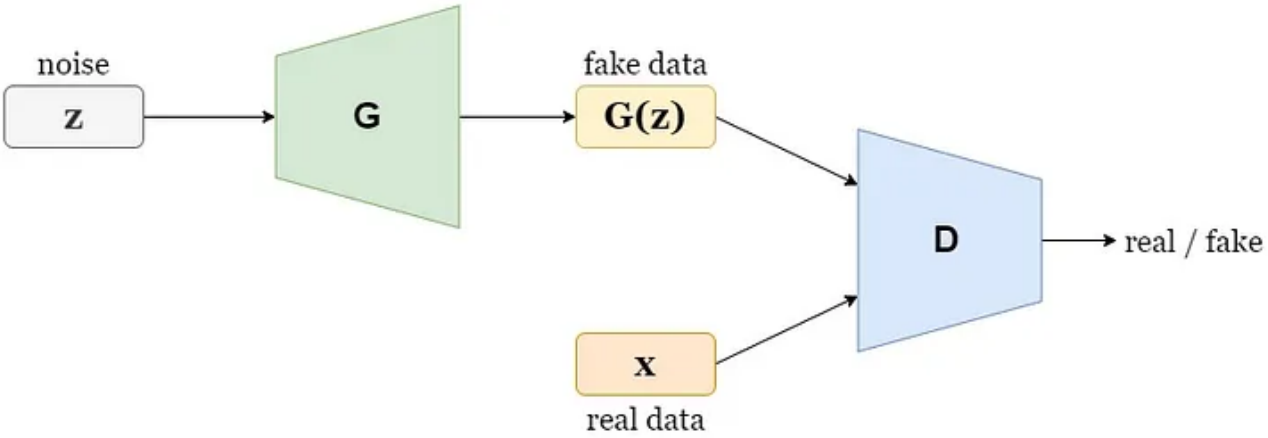

図 1: 敵対的生成ネットワーク (GAN) — 出典: 著者による画像

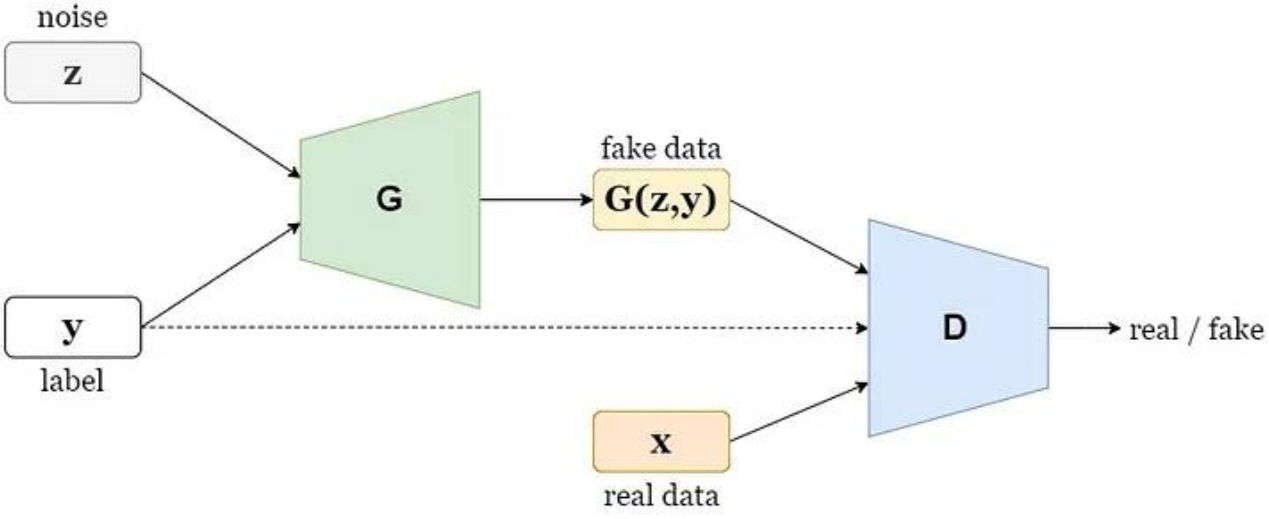

図 2: 条件付き敵対的生成ネットワーク (cGAN) — 出典: 著者による画像

知識蒸留

特に複雑なデータセットでは、GAN をゼロからトレーニングする手順は複雑です [6]。さらに、現在の最先端モデル [4][5] では、優れたパフォーマンスを達成するためにスケールアップする傾向があります。問題は、小さなモデルを使用して高品質の画像を生成できるかどうかです。Romero ほか (2014) [7] で、著者らは、ネットワーク圧縮のための知識蒸留フレームワークを提案しています。このフレームワークでは、事前トレーニング済みの教師ネットワーク (大きなモデル) の知識を使用して生徒ネットワーク (小さなモデル) をトレーニングすることにより、必要なパラメーターの数を大幅に減らしながら、前者と同等の結果を達成しています。Chang ほか (2020) [8] で、著者らは、このフレームワークを採用して、GAN 向けに設計されたブラックボックス知識蒸留手法を提案しています。彼らが提案したモデル、TinyGAN は、パラメーターの数を 16 分の 1 に減らしながら競争力のあるパフォーマンスを達成し、BigGAN の蒸留に成功しました。

プロジェクト

TinyGAN は競争力のあるパフォーマンスを維持したまま BigGAN のパラメーターの数を減らすことに成功しましたが、トレーニングと画像生成の両方で大量の計算リソースが必要なことは変わっていません。このプロジェクトのパート 1 の目的は、提案された蒸留フレームワークを活用して、TinyGAN より浅いネットワークで高品質の画像を生成できるかどうか調査することです。計算リソースが限られているため、CIFAR-10 [9] データセットを使用しました。各画像の空間サイズは大きくありませんが (32x32)、高品質の画像を生成するには大規模なモデルが必要になります。CIFAR-10 のクラス条件付き画像生成における現在の最先端モデルである StyleGAN2 には、2,000 万を超えるトレーニング可能なパラメーターが含まれています。次のセクションでは、使用する蒸留手法、トレーニングの目的、データセット、選択したネットワークのアーキテクチャーを説明します。

ブラックボックス蒸留

Chang ほか (2020) [8] で提案された知識蒸留フレームワークを使用します。このフレームワークでは教師ネットワークがブラックボックスとして適用され、その入出力ペアへの限られたアクセスのみ必要です。教師モデルは、クラスラベルとランダム・ノイズ・ベクトルを入力として生成された画像を収集します。作成されるコレクションには、生成された画像だけでなく、ノイズとクラスラベルの入力ベクトルも含まれます。このコレクション、つまりデータセットは、教師ありの手法で生徒ネットワークをトレーニングするために活用されます。このアプローチでは、モデルの内部の状態や中間の特徴についての知識は必要ありません。さらに、データセットの作成時に、教師ネットワークは生徒ネットワークのトレーニングに参加しなくなるため、教師ネットワークを破棄できます。

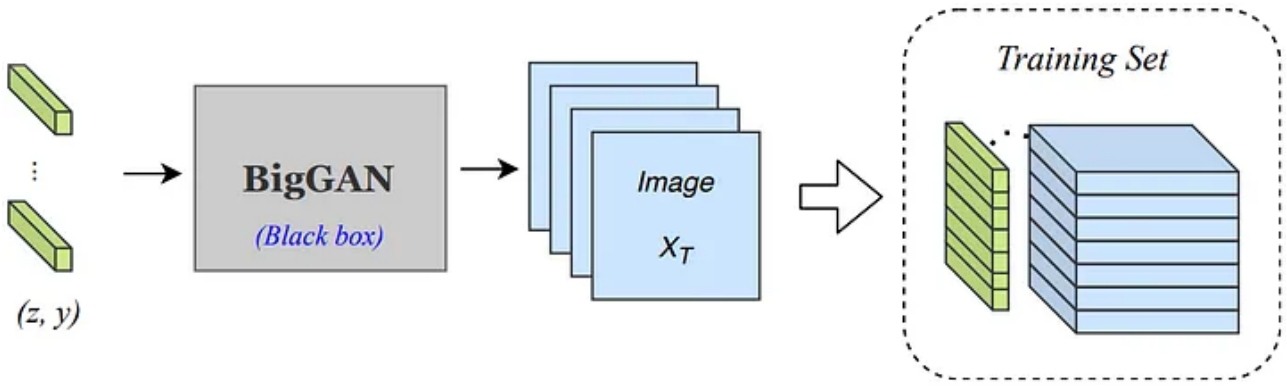

図 3: Chang ほか (2020) [8] で提案されたブラックボックス蒸留フレームワーク

目標

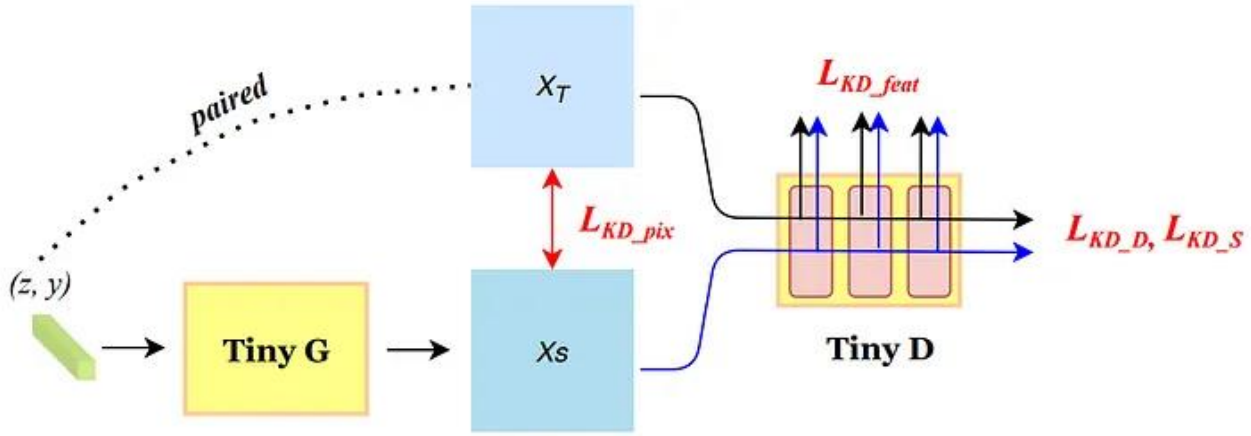

図 4: Chang ほか (2020) [8] のトレーニングの目標

知識蒸留フレームワークの一番の目的は、生徒の生成器を教師の生成器の分布に近似させること、または教師の機能を模倣することです。この目的を考慮して、[8] に記載されている目標を活用します。著者らは生成器について次の目標を提案しています。

$$\mathcal{L}_G = \mathcal{L}_{KD\_feat} + \lambda_1 \mathcal{L}_{KD\_pix} + \lambda_2 \mathcal{L}_{KD\_S} + \lambda_3 \mathcal{L}_{GAN\_S}$$

説明:

- 特徴レベルの蒸留損失は、生徒ネットワークで生成された画像と教師ネットワークで生成された画像の両方について、識別器のネットワークを使用して抽出された特徴マップから計算されます。生徒ネットワークで生成された画像と教師ネットワークで生成された画像のペアの異なるレベルの特徴マップ間の L1 距離の加重和として計算され、次に説明するピクセルレベルの蒸留損失のみを使用する場合に、生成された画像の不鮮明さを軽減するために使用されます。

$$L_{KD\_feat} = \mathbb{E}_{z,y}\left[\Sigma_i \alpha_i \| D_i(T(z,y),y) - D_i(S(z,y),y) \|_1\right]$$

- ピクセルレベルの蒸留損失は、教師ネットワークで生成された画像と生徒ネットワークで生成された画像間のピクセルレベルの L1 距離として計算されます。生徒ネットワークが教師の機能を模倣しようとしている間の管理を提供するため、この損失はトレーニングの初期段階で特に重要です。

$$L_{KD\_pix} = \mathbb{E}_{z\sim p(z), y\sim q(y)}\left[\|T(z,y) - S(z,y)\|_1\right]$$

- 敵対的蒸留損失は、生徒の生成器を教師の生成器の分布に近似させるために使用されます。識別器は、教師ネットワークで使用される入力に、生徒ネットワークで生成された画像の敵対的損失を提供するために使用されます。

$$L_{KD\_S} = -\mathbb{E}_{z,y}[D(S(z,y),y)]$$

- 敵対的 GAN 損失は、モデルが選択したデータセットに似た画像を生成することも学習するために使用されます。そのため、識別器を利用して、生徒ネットワークのランダムに生成された画像に対する敵対的損失を生成します。生成器の損失は、敵対的蒸留損失と同じ方法で計算されます。

識別器には、次の目標を使用します。

$$\mathcal{L}_D = \mathcal{L}_{KD\_D} + \lambda_4 \mathcal{L}_{GAN\_D}$$

説明:

- 敵対的蒸留損失は、識別器が生徒ネットワークで生成された画像と教師ネットワークで生成された画像を区別できるようにするために使用されます。この場合、生徒の画像は偽物として扱われ、教師の画像は本物として扱われます。

$$L_{KD\_D} = \mathbb{E}_{z,y}[\max(0, 1 - D(T(z,y),y)) + \max(0, 1 + D(S(z,y),y))]$$

- 敵対的 GAN 損失は、識別器が生徒ネットワークで生成された画像と実際のデータの分布から生成された画像を区別できるようにするために使用されます。

$$L_{GAN\_D} = \mathbb{E}_{x,y}[\max(0, 1 - D(x,y))] + \mathbb{E}_{z,y}[\max(0, 1 + D(S(z,y),y))]$$

上記の敵対的損失はすべて、バイナリー・クロスエントロピーの代わりに敵対的損失のヒンジバージョン [10] を使用して計算されます。

データセット

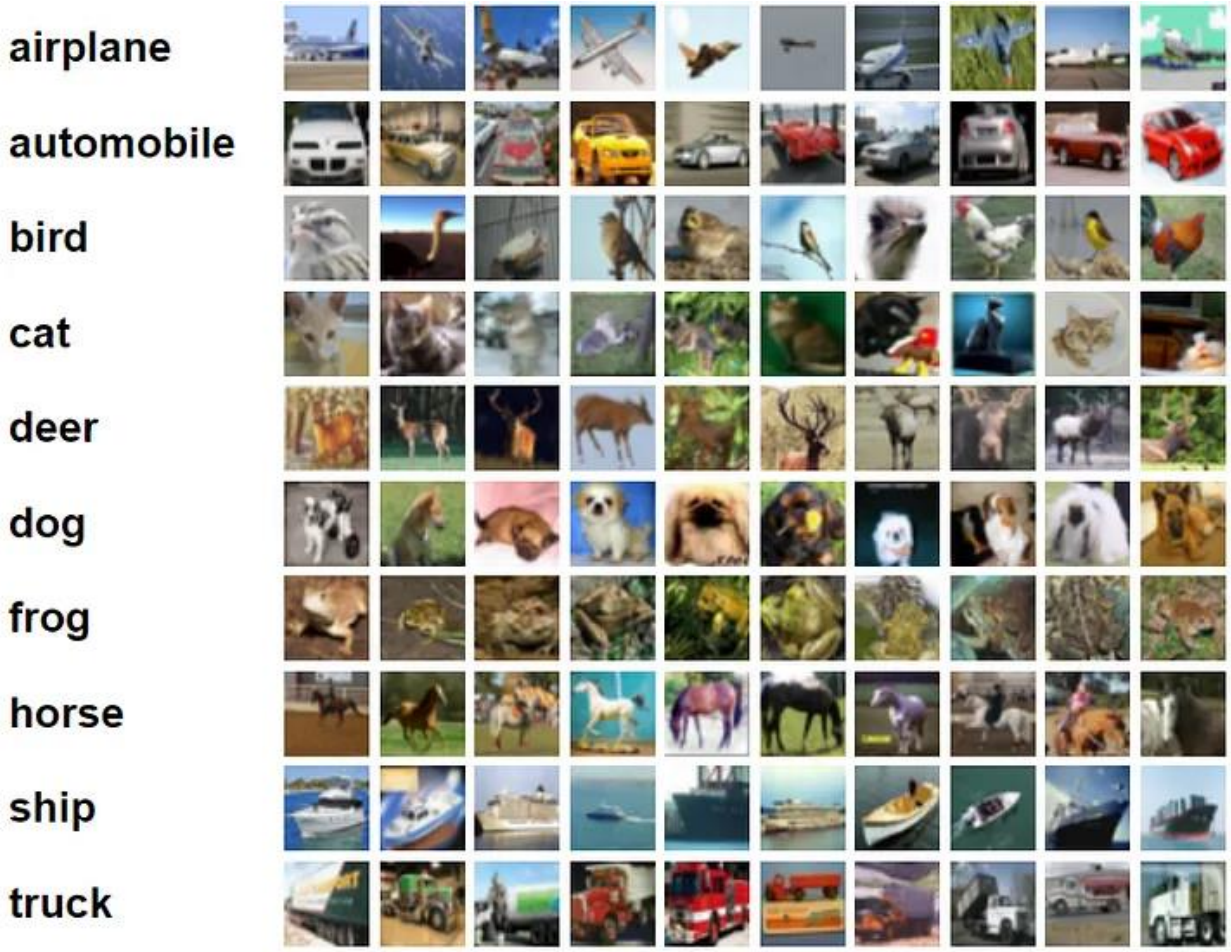

図 5: CIFAR-10 のクラスと各 10 枚のランダムな画像 (出典 (英語))

CIFAR-10 [9] データセットは、クラスごとに 6,000 枚の画像を含む、10 の異なるクラスカテゴリーの 60,000 枚のカラー画像で構成されます。各画像の空間サイズは大きくありませんが (32x32)、高品質の画像を生成するには大規模なモデルが必要になります。データセットの詳細は、公式サイト (英語) を参照してください。

教師ネットワーク

CIFAR-10 データセットには、StyleGAN2-ADA モデルを蒸留することにしました。このモデルは、CIFAR-10 [9] データセットの条件付き画像生成タスクで最先端のパフォーマンスを達成できます。この成功の主な理由は、提案された適応型識別器拡張メカニズムは、利用可能なデータが限られている場合にトレーニングを非常に安定させることができるためです。しかし、ここでは、モデルはブラックボックス画像の生成に利用されるため、研究のトレーニング手順と手法で必要なのはモデルの生成器の入出力ペアへのアクセスのみです。条件付き画像生成には、GitHub* の NVIDIA* Research Projects で公開されている StyleGAN2-ADA モデルの公式 PyTorch* 実装と、CIFAR-10 データセットの事前トレーニング済みモデルに提供された重みを使用します。StyleGAN を使用して、モデルで生成された画像と、対応する入力ノイズベクトルおよびラベルで構成される FakeCIFAR10 データセットを作成します。その後、上記の目標を使用して、教師ネットワーク (つまり、StyleGAN2-ADA) の機能を模倣するように、このデータセットを使用して生徒ネットワークをトレーニングします。一方、データセットの作成時に、StyleGAN2-ADA モデルは生徒ネットワークのトレーニング手順で使用されなくなるため破棄されます。

FakeCIFAR10

FakeCIFAR10 データセットは、StyleGAN2-ADA モデルで生成された 50,000 枚の合成画像で構成されます。10 のクラスごとに 5,000 枚の画像と、StyleGAN の生成器の入力として使用されたノイズベクトルが含まれます。ここから (英語) データセットをダウンロードするか、この手順 (英語) に従ってデータセットを再作成できます。

生徒ネットワーク

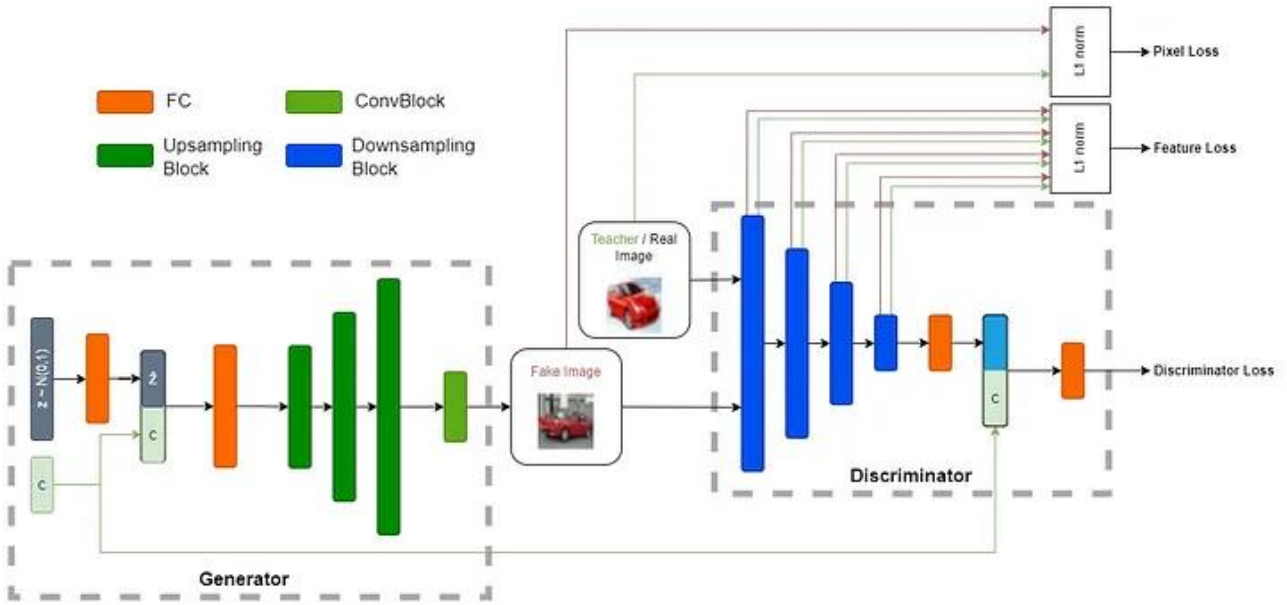

図 6: DiStyleGAN のアーキテクチャーの概要 — 出典: 著者による画像

生成器
最初に、全結合層を使用して、ガウス・ランダム・ノイズ・ベクトルが 128 次元に投影されます。次に、条件埋め込みと投影されたノイズベクトルが連結されて別の全結合層を通り、3 つの連続するアップサンプリング・ブロックが続きます。各アップサンプリング・ブロックは、アップサンプル層 (scale_factor=2、mode='nearest')、3x3 畳み込み (パディングあり)、バッチ正規化層、およびゲート線形ユニット (GLU) で構成されます。最後に、3x3 畳み込み (パディングあり) と双曲線正接活性化関数 (tanh) で構成される畳み込みブロックがあり、偽の画像が生成されます。

識別器
DiStyleGAN の識別器は 4 つの連続するダウンサンプリング・ブロック (4x4 ストライドの畳み込み、スペクトル正規化、LeakyReLU) で構成され、各ブロックは入力画像の空間サイズを 2 分の 1 に縮小します。これらの 4 つのブロックは、特徴損失の計算に使用される特徴マップも生成します。続いて、ロジットが平滑化され、128 次元に投影されて、クラス条件埋め込みと連結された後、最後の全結合層を通り、クラス条件付き識別器損失が生成されます。

トレーニング

1. GitHub* リポジトリー (英語) をクローンします。
2. 要件 (英語) ファイルの Python* パッケージをインストールします。(Python* 3.10)
3. ここから (英語) FakeCIFAR10 データセットをダウンロードして zip ファイルを展開するか、この手順 (英語) に従って create_dataset.py (英語) スクリプトを使用してデータセットを再作成します。
4. 次のいずれかのオプションを使用してモデルをトレーニングします。
    - Python* のデフォルトの設定を使用します。

```python
from distylegan import DiStyleGAN
model = DiStyleGAN()
model.train(
    dataset="./fakecifar/dataset",
    save="results"
)
```

train.py hosted with ♥ by GitHub view raw

Python* で DiStyleGAN をトレーニング

    - 対応する Python* スクリプト (英語) のコマンドライン・オプションを使用します。

```
$ python distylegan.py train -h
usage: distylegan.py train [-h] --dataset DATASET --save SAVE [--
c_dim C_DIM] [--lambda_ganD LAMBDA_GAND] [--lambda_ganG LAMBDA_GANG]
[--lambda_pixel LAMBDA_PIXEL] [--nc NC][--ndf NDF] [--ngf NGF] [--
project_dim PROJECT_DIM] [--transform TRANSFORM] [--z_dim Z_DIM] [--
adam_momentum ADAM_MOMENTUM] [--batch_size BATCH_SIZE] [--
checkpoint_interval CHECKPOINT_INTERVAL] [--checkpoint_path
CHECKPOINT_PATH] [--device DEVICE] [--epochs EPOCHS] [--gstep GSTEP]
[--lr_D LR_D] [--lr_G LR_G] [--lr_decay LR_DECAY] [--num_test
NUM_TEST] [--num_workers NUM_WORKERS] [--real_dataset REAL_DATASET]
options:
  -h, --help            show this help message and exit
Required arguments for the training procedure:
  --dataset DATASET     Path to the dataset directory of the fake
CIFAR10 data generated by the teacher network
  --save SAVE           Path to save checkpoints and results
Optional arguments about the network configuration:
  --c_dim C_DIM         Condition dimension (Default: 10)
  --lambda_ganD LAMBDA_GAND
                        Weight for the adversarial GAN loss of the
                        Discriminator (Default: 0.2)
  --lambda_ganG LAMBDA_GANG
                        Weight for the adversarial distillation loss
                        of the Generator (Default: 0.01)
  --lambda_pixel LAMBDA_PIXEL
                        Weight for the pixel loss of the Generator
                       (Default: 0.2)
  --nc NC               Number of channels for the images
                        (Default: 3)
  --ndf NDF             Number of discriminator filters in the first
                        convolutional layer (Default: 128)
  --ngf NGF             Number of generator filters in the first
                        convolutional layer (Default: 256)
```

```
  --project_dim PROJECT_DIM
                        Dimension to project the input condition
                       (Default: 128)
  --transform TRANSFORM
                        Optional transform to be applied on a sample
                        image (Default: None)
  --z_dim Z_DIM         Noise dimension (Default: 512)
Optional arguments about the training procedure:
  --adam_momentum ADAM_MOMENTUM
                        Momentum value for the Adam optimizers'
                        betas (Default: 0.5)
  --batch_size BATCH_SIZE
                        Number of samples per batch (Default: 128)
  --checkpoint_interval CHECKPOINT_INTERVAL
                        Checkpoints will be saved every `
                        checkpoint_interval` epochs (Default: 20)
  --checkpoint_path CHECKPOINT_PATH
                        Path to previous checkpoint
  --device DEVICE       Device to use for training ('cpu' or 'cuda')
                       (Default: If there is a CUDA device
                        available, it will be used for training)
  --epochs EPOCHS       Number of training epochs (Default: 150)
  --gstep GSTEP         The number of discriminator updates after
                        which the generator is updated using the
                        full loss(Default: 10)
  --lr_D LR_D           Learning rate for the discriminator's Adam
                        optimizer (Default: 0.0002)
  --lr_G LR_G           Learning rate for the generator's Adam
                        optimizer (Default: 0.0002)
  --lr_decay LR_DECAY   Iteration to start decaying the learning
                        rates for the Generator and the
                        Discriminator(Default: 350000)
  --num_test NUM_TEST   Number of generated images for evaluation
                       (Default: 30)
  --num_workers NUM_WORKERS
                        Number of subprocesses to use for data
                        loading (Default: 0, whichs means that the
                        data will be loaded in the main process.)
  --real_dataset REAL_DATASET
                        Path to the dataset directory of the real
                        CIFAR10 data. (Default: None, it will be
                        downloaded and saved in the parent directory
                        of input `dataset` path)
```

画像生成

1. GitHub* リポジトリー (英語) をクローンします。
2. 要件 (英語) ファイルの Python* パッケージをインストールします。(Python* 3.10)
3. 事前トレーニング済みモデルの checkpoint.zip (英語) ファイルをダウンロードして、リポジトリーの root ディレクトリーで zip ファイルを展開します。
4. 次のいずれかのオプションを使用して合成サンプルを生成します。
    - Python* の例を使用します。

```python
from distylegan import DiStyleGAN
model = DiStyleGAN()
images= model.generate(
    checkpoint_path="../checkpoint",
    nsamples=100,
    label=[0, 3, 7] # or label=x (int in range [0,9]), label=None
    save="synthetic-samples",
    batch_size=32
)
```

generate.py hosted with ❤ by GitHub view raw

Python* で DiStyleGAN を使用してサンプルを生成

    - 対応する Python* スクリプト (英語) のコマンドライン・オプションを使用します。

```
$ python distylegan.py generate -h
usage: distylegan.py generate [-h] --checkpoint_path CHECKPOINT_PATH
--nsamples NSAMPLES --save SAVE [--label
[{0,1,2,3,4,5,6,7,8,9} ...]] [--batch_size BATCH_SIZE]
options:
  -h, --help            show this help message and exit
Required arguments for the generation procedure:
  --checkpoint_path CHECKPOINT_PATH
                        Path to previous checkpoint (the directory
                        must contain the generator.pt and
                        config.json files)
  --nsamples NSAMPLES   Number of samples to generate per label
  --save SAVE           Path to save the generated images to
Optional arguments about the generation procedure:
  --label [{0,1,2,3,4,5,6,7,8,9} ...]
                        Class label(s) for the samples (Default:
                        None, random labels) --> e.g. --label 0 3 7
  --batch_size BATCH_SIZE
                        Number of samples per batch (Default: 32)
```

    - リポジトリーの webapp/ ディレクトリー内でコマンド `flask run` を実行して Flask webapp を使用します。次に、コマンドラインに表示されるリンク (例えば http://127.0.0.1:5000) をクリックします。図 7 の画面が表示されます。

図 7: DiStyleGAN を使用した合成サンプル生成用の Flask webapp — 出典: 著者による画像

評価

モデルのパフォーマンスを定性的および定量的の両方で評価しました。

定性的評価

定性的評価として、入力と同じノイズベクトルに対して、トレーニングの各エポックで合成サンプルを生成しました。次に、トレーニング中の合成サンプルの変化の GIF ファイルを生成する gifmaker.py (英語) スクリプトを使用して、生成されたサンプルの画質からトレーニングの進行状況を調査しました。図 8 は、DiStyleGAN モデルのトレーニング用に作成されたファイルを示しています。

図 8: トレーニング中の DiStyleGAN の合成サンプルの変化 — 出典: 著者による画像

さらに、Zhan ほか (2017) [12] から着想を得た t-SNE アルゴリズム [13] を利用して、GAN の一般的な問題である**モード崩壊** (生成器が類似した出力ばかり生成してしまう問題) がトレーニング済みモデルで発生しているかどうかを調査しました。生成された 5,000 個のサンプル (クラスごとに 500 個のサンプル) に tsne.py (英語) スクリプトを使用し、事前トレーニング済みの VGG19 モデルを使用して 2D グリッド上に合成サンプルの t-SNE 可視化を作成し、4,096 次元の特徴を抽出した後、PCA アルゴリズムを使用して 300 次元に圧縮

し、t-SNE アルゴリズムを使用してデカルト座標にマップしました。この手順により、同様に見える画像がグリッド内の隣接するタイルに配置され、モード崩壊が発生しているかどうかを確認できます。図 9 は、事前トレーニング済みモデルのグリッドを示しています。目立ったモード崩壊のパターンは見つかりませんでした。

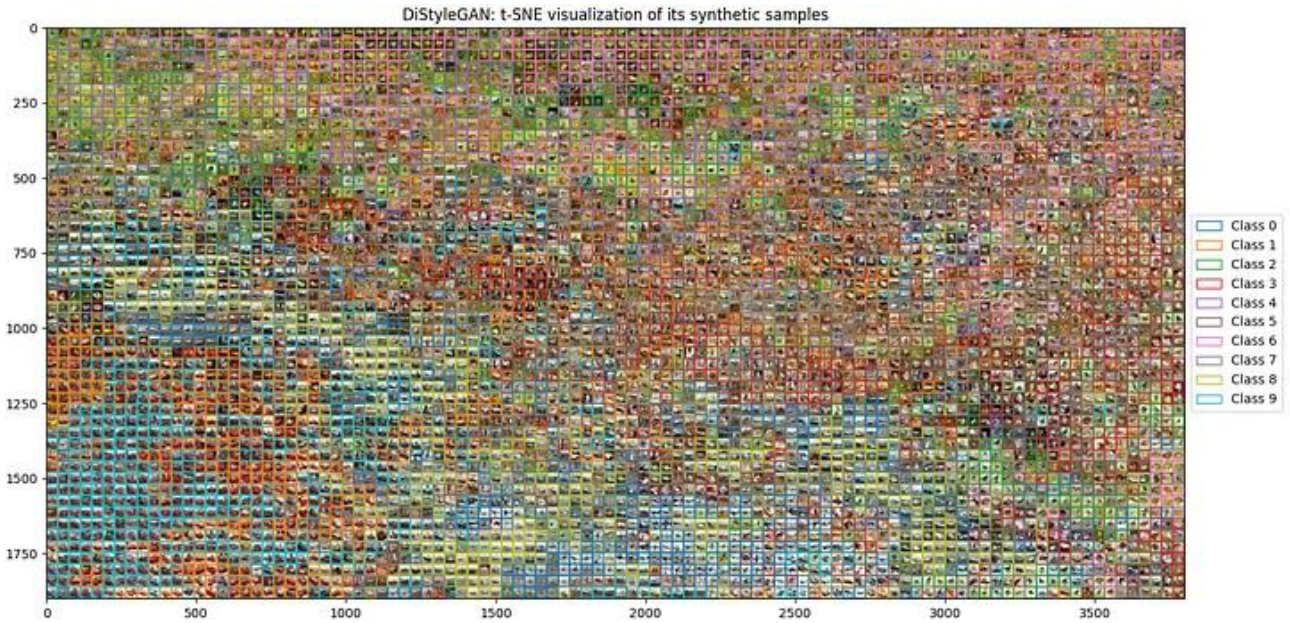

図 9: DiStyleGAN の合成サンプルの t-SNE 可視化 — 出典: 著者による画像

定量的評価

定量的評価として、Inception Score (IS、インセプション・スコア) [14] と Fréchet Inception Distance (FID、フレシェ・インセプション距離) [15] を計算しました。GitHub* (英語) の Junho Kim (英語) と Ahmed Fares (英語) による 2 つのメトリックの TensorFlow* 実装を使用しました。図 10 に、50,000 枚 (クラスごとに 5,000 枚) の合成画像を計算した結果を示します。インセプション・スコアの計算には 10 回の分割を使用しました。

Inception Score	Fréchet Inception Distance
6.78 (± 0.08)	42.30

図 10: CIFAR-10 の DiStyleGAN のインセプション・スコアとフレシェ・インセプション距離

まとめ

条件付き画像生成向けのモデルの開発は困難な作業です。CIFAR-10 のような一見すると単純なデータセットでも、高品質の画像を生成できるのは、非常に多くのパラメーターを含む巨大なモデルだけです。DCGAN [16] と同様のネットワーク・アーキテクチャーを使用しつつ、Romero ほか (2014) [7] で提案された知識蒸留フレームワークを採用することにより、インセプション・スコア・メトリックに関してタスクのパフォーマンスを向上させることができました。プロジェクトのパート 2 では、DiStyleGAN により生成された合成データを使用して、さまざまなコンピューター・ビジョン・モデルのパフォーマンスを最適化します。

参考文献 (英語)

[1] Goodfellow, Ian, et al. "Generative adversarial nets." Advances in neural information processing systems 27 (2014).

[2] Sauer, Axel, Katja Schwarz, and Andreas Geiger. "Stylegan-xl: Scaling stylegan to large diverse datasets." arXiv preprint arXiv:2202.00273 1 (2022).

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, ImageNet: A Large-Scale Hierarchical Image Database. IEEE Computer Vision and Pattern Recognition (CVPR), 2009.

[4] Kang, Minguk, et al. "Rebooting acgan: Auxiliary classifier gans with stable training." Advances in Neural Information Processing Systems 34 (2021): 23505–23518.

[5] Brock, Andrew, Jeff Donahue, and Karen Simonyan. "Large scale GAN training for high fidelity natural image synthesis." arXiv preprint arXiv:1809.11096 (2018).

[6] Salimans, Tim, et al. "Improved techniques for training gans." Advances in neural information processing systems 29 (2016).

[7] Romero, Adriana, et al. "Fitnets: Hints for thin deep nets." arXiv preprint arXiv:1412.6550 (2014).

[8] Chang, Ting-Yun, and Chi-Jen Lu. "Tinygan: Distilling biggan for conditional image generation." Proceedings of the Asian Conference on Computer Vision. 2020.

[9] Krizhevsky, Alex, and Geoffrey Hinton. "Learning multiple layers of features from tiny images." (2009): 7.

[10] Lim, Jae Hyun, and Jong Chul Ye. "Geometric gan." arXiv preprint arXiv:1705.02894 (2017).

[11] Karras, Tero, et al. "Training generative adversarial networks with limited data." Advances in Neural Information Processing Systems 33 (2020): 12104–12114.

[12] Zhang, Han, et al. "Stackgan++: Realistic image synthesis with stacked generative adversarial networks." *IEEE transactions on pattern analysis and machine intelligence* 41.8 (2018): 1947–1962.

[13] Laurens van der Maaten and Geoffrey Hinton, "Visualizing Data using t-SNE", Journal of Machine Learning Research, vol. 9, no. 86, pp. 2579–2605, 2008.

[14] Salimans, Tim, et al. "Improved techniques for training gans." *Advances in neural information processing systems* 29 (2016).

[15] Heusel, Martin, et al. "Gans trained by a two time-scale update rule converge to a local nash equilibrium." *Advances in neural information processing systems* 30 (2017).

[16] Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." *arXiv preprint arXiv:1511.06434* (2015).

OpenVINO™ ツールキットとは

AI を加速する無償のツールである OpenVINO™ ツールキットは、インテルが無償で提供しているインテル製の CPU や GPU、VPU、FPGA などのパフォーマンスを最大限に活用して、コンピューター・ビジョン、画像関係をはじめ、自然言語処理や音声処理など、幅広いディープラーニング・モデルで推論を最適化し高速化する推論エンジン/ツールスイートです。

OpenVINO™ ツールキット・ページでは、ツールの概要、利用方法、導入事例、トレーニング、ツール・ダウンロードまでさまざまな情報を提供しています。ぜひ特設サイトにアクセスしてみてください。

https://www.intel.co.jp/content/www/jp/ja/internet-of-things/openvino-toolkit.html