

Nama : Muhamad Afri Marliansyah  
Nim : 1103210001

## Task 6 Computer Vision Hugging Face Course (Chapter 9-12)

### 1. Torch (Chapter 9)

#### Pytorch Quantization

PyTorch mendukung kuantisasi INT8 dibandingkan dengan model FP32 yang umum sehingga memungkinkan pengurangan 4x dalam ukuran model dan pengurangan 4x dalam persyaratan bandwidth memori sambil tetap mencapai akurasi yang sebanding untuk banyak aplikasi. Buku catatan ini menunjukkan cara mengkuantisasi model dari FP32 ke INT8 menggunakan perkakas kuantisasi PyTorch. Kami akan melatih model CNN sederhana pada mnist dan kemudian mengkuantisasinya menggunakan perkakas kuantisasi dan membandingkan akurasi dan ukuran model yang dikuantisasi dengan model FP32 asli.

```
%pip install torch torchvision
```

Kode di atas berfungsi untuk meng-instal PyTorch dan torchvision.

```
import torch #Mengimpor pustaka PyTorch, yang digunakan untuk membangun dan
melatih model deep learning.
import torch.nn as nn # Mengimpor modul `nn` dari PyTorch, yang menyediakan
berbagai lapisan dan fungsi untuk membangun jaringan saraf.
import torch.nn.functional as F # Mengimpor fungsi fungsional dari PyTorch, yang
berisi berbagai fungsi aktivasi, fungsi loss, dll.
import torch.optim as optim # Mengimpor modul `optim` dari PyTorch, yang
menyediakan algoritma optimisasi untuk memperbarui bobot model.
from torchvision import datasets, transforms # Mengimpor `datasets` dan
`transforms` dari `torchvision` untuk memudahkan pengolahan gambar dan dataset.
import torch.quantization # Mengimpor modul kuantisasi dari PyTorch untuk
mengoptimalkan model dengan mengurangi presisi data.
import pathlib # Mengimpor pustaka `pathlib` untuk memanipulasi dan bekerja
dengan path file dan direktori.
```

Penjelasan dari setiap baris kode di atas adalah sebagai berikut:

1. `import torch:`

Mengimpor pustaka PyTorch, yang digunakan untuk membangun dan melatih model deep learning (pembelajaran mendalam). PyTorch menyediakan berbagai alat untuk komputasi tensor, yang mendukung pemrosesan data besar dan pelatihan model.

2. `import torch.nn as nn:`

Mengimpor modul nn dari PyTorch. Modul ini menyediakan berbagai lapisan neural network (jaringan saraf) dan fungsi yang berguna dalam membangun model deep learning, seperti lapisan linear, konvolusional, atau rekursif, serta beberapa fungsi untuk pembelajaran.

3. `import torch.nn.functional as F:`

Mengimpor fungsi fungsional dari PyTorch yang menyediakan berbagai fungsi tambahan untuk deep learning, seperti fungsi aktivasi (ReLU, Sigmoid, dll.), fungsi loss (CrossEntropyLoss, MSELoss, dll.), dan operasi matematis lainnya yang digunakan dalam jaringan saraf.

4. `import torch.optim as optim:`

Mengimpor modul optim dari PyTorch, yang menyediakan berbagai algoritma optimisasi untuk memperbarui bobot model selama pelatihan, seperti Stochastic Gradient Descent (SGD), Adam, atau RMSprop. Optimizer digunakan untuk mengurangi kesalahan atau loss dalam model.

5. `from torchvision import datasets, transforms:`

Mengimpor datasets dan transforms dari pustaka torchvision.

- datasets berisi dataset yang siap digunakan (misalnya CIFAR-10, MNIST, dll.) untuk pelatihan dan pengujian model deep learning, yang sering digunakan untuk pengolahan gambar.
- transforms berisi berbagai fungsi untuk melakukan transformasi gambar seperti pengubahan ukuran, normalisasi, atau augmentasi gambar, yang membantu meningkatkan kinerja model dalam pengolahan gambar.

6. `import torch.quantization:`

Mengimpor modul kuantisasi dari PyTorch. Kuantisasi digunakan untuk mengurangi presisi data dalam model deep learning, yang membantu mengurangi ukuran model dan meningkatkan kecepatan inferensi tanpa mengurangi akurasi secara signifikan. Kuantisasi sering digunakan untuk menjalankan model di perangkat dengan daya komputasi terbatas, seperti perangkat mobile.

7. `import pathlib:`

Mengimpor pustaka pathlib, yang digunakan untuk memanipulasi dan bekerja dengan path file dan direktori. Pustaka ini menyediakan cara yang lebih sederhana dan lebih fleksibel untuk bekerja dengan file dan direktori dibandingkan menggunakan metode lama berbasis string.

### **Kuantisasi Dinamis**

Untuk kuantisasi dinamis, bobot dikuantisasi tetapi aktivasi dibaca atau disimpan dalam floating point dan aktivasi hanya dikuantisasi untuk komputasi.

### **Memuat dataset MNIST**

Pertama, kita memuat dataset MNIST

```
# Membuat pipeline transformasi yang akan diterapkan pada data gambar
transform = transforms.Compose([
    transforms.ToTensor(), # Mengubah gambar menjadi tensor PyTorch. Gambar
    akan dikonversi ke format tensor dengan dimensi (C, H, W), yaitu channel, tinggi,
    dan lebar
```

```

        transforms.Normalize((0.1307,), (0.3081,)) # Menormalkan gambar dengan
rata-rata (0.1307) dan deviasi standar (0.3081), yang sesuai dengan dataset MNIST
    ])

# Memuat dataset MNIST untuk pelatihan. Data akan disimpan di direktori './data',
dan jika belum ada, dataset akan diunduh. Transformasi yang telah didefinisikan
akan diterapkan pada data pelatihan
train_dataset = datasets.MNIST('./data', train=True, download=True,
transform=transform)

# Memuat dataset MNIST untuk pengujian (test). Transformasi yang sama akan
diterapkan pada data pengujian.
test_dataset = datasets.MNIST('./data', train=False, transform=transform)

```

Berikut adalah penjelasan dari setiap bagian kode yang diberikan:

1. Membuat pipeline transformasi:
2. `transform = transforms.Compose([`
3.     `transforms.ToTensor(),` # Mengubah gambar menjadi tensor PyTorch
4.     `transforms.Normalize((0.1307,), (0.3081,))` # Menormalkan gambar dengan rata-rata dan deviasi standar yang sesuai
5.     `])`
  - `transforms.Compose([ ... ])`: Ini digunakan untuk menggabungkan beberapa transformasi yang akan diterapkan pada gambar. Di sini, dua transformasi digunakan.
  - `transforms.ToTensor()`: Mengubah gambar dari format PIL image atau NumPy array menjadi tensor PyTorch. Gambar akan dikonversi ke tensor dengan dimensi (C, H, W) (channel, tinggi, lebar). Ini adalah langkah yang diperlukan karena model deep learning di PyTorch memerlukan data dalam bentuk tensor.
  - `transforms.Normalize((0.1307,), (0.3081,))`: Menormalkan gambar dengan rata-rata (0.1307,) dan deviasi standar (0.3081,). Nilai ini sesuai dengan statistik dari dataset MNIST. Normalisasi ini penting agar input ke model lebih stabil dan membantu model belajar dengan lebih baik. Normalisasi mengubah piksel gambar agar distribusinya lebih seimbang.
6. Memuat dataset MNIST untuk pelatihan:
7. `train_dataset = datasets.MNIST('./data', train=True, download=True, transform=transform)`
  - `datasets.MNIST`: Ini adalah cara untuk memuat dataset MNIST, yang merupakan dataset gambar tangan angka (0-9) yang sangat populer untuk melatih dan menguji model klasifikasi gambar.
  - `'./data'`: Direktori tempat dataset akan disimpan. Jika dataset belum ada di folder ini, maka dataset akan diunduh secara otomatis.
  - `train=True`: Menandakan bahwa kita ingin memuat data untuk pelatihan (training set).
  - `download=True`: Jika dataset belum ada di direktori yang ditentukan, PyTorch akan mengunduhnya secara otomatis.

- transform=transform: Ini adalah transformasi yang telah didefinisikan sebelumnya. Transformasi ini akan diterapkan pada setiap gambar dalam dataset sebelum digunakan dalam model.
- 8. Memuat dataset MNIST untuk pengujian:
- 9. test\_dataset = datasets.MNIST('./data', train=False, transform=transform)
  - train=False: Menandakan bahwa kita ingin memuat data untuk pengujian (test set).
  - transform=transform: Sama seperti pada dataset pelatihan, transformasi yang sama diterapkan pada gambar di dataset pengujian.

## Melatih Model

Selanjutnya, kami mendefinisikan model CNN sederhana dan kemudian melatihnya pada dataset MNIST

```
# Mendefinisikan kelas model neural network
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        # Layer konvolusi pertama, dengan input 1 channel (grayscale image),
        # output 12 channel, dan kernel 3x3
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=12, kernel_size=3)
        # Layer pooling maksimum dengan ukuran kernel 2x2 dan stride 2
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        # Layer Fully Connected (FC) untuk output 10 kelas (untuk klasifikasi
        # digit MNIST)
        self.fc = nn.Linear(12 * 13 * 13, 10)

    def forward(self, x):
        # Mengubah bentuk input menjadi 4 dimensi (-1 adalah batch size yang
        # otomatis ditentukan, 1 adalah jumlah channel, 28x28 adalah ukuran gambar)
        x = x.reshape(-1, 1, 28, 28)
        # Menjalankan input melalui layer konvolusi pertama dan aktivasi ReLU
        x = F.relu(self.conv1(x))
        # Menjalankan input melalui layer pooling
        x = self.pool(x)
        # Mengubah input menjadi bentuk vektor satu dimensi sebelum dimasukkan ke
        # layer FC
        x = x.reshape(x.size(0), -1)
        # Melalui layer FC dan menghasilkan output untuk 10 kelas
        x = self.fc(x)
        # Menggunakan log softmax untuk menghasilkan probabilitas log untuk
        # setiap kelas
        output = F.log_softmax(x, dim=1)
        return output

# Membuat DataLoader untuk data pelatihan dan pengujian, dengan ukuran batch 32
```

```

train_loader = torch.utils.data.DataLoader(train_dataset, 32)
test_loader = torch.utils.data.DataLoader(test_dataset, 32)

# Memilih device untuk pelatihan (GPU jika tersedia, jika tidak CPU)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Menetapkan jumlah epoch
epochs = 1

# Menginisialisasi model, optimizer, dan memindahkan model ke device yang telah
dipilih
model = Net().to(device)
optimizer = optim.Adam(model.parameters())

# Mengubah model ke mode pelatihan
model.train()

# Loop untuk setiap epoch
for epoch in range(1, epochs+1):
    # Loop untuk setiap batch dalam DataLoader pelatihan
    for batch_idx, (data, target) in enumerate(train_loader):
        # Memindahkan data dan target ke device yang telah dipilih
        data, target = data.to(device), target.to(device)
        # Mengatur gradien optimizer menjadi nol
        optimizer.zero_grad()
        # Melakukan forward pass untuk mendapatkan output dari model
        output = model(data)
        # Menghitung loss menggunakan Negative Log-Likelihood Loss (NLLLoss)
        loss = F.nll_loss(output, target)
        # Melakukan backward pass untuk menghitung gradien
        loss.backward()
        # Memperbarui parameter model dengan optimizer
        optimizer.step()
        # Menampilkan progres pelatihan (epoch, batch, dan loss)
        print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
            epoch, batch_idx * len(data), len(train_loader.dataset),
            100. * batch_idx / len(train_loader), loss.item()))

```

Kode ini mendefinisikan dan melatih model neural network untuk klasifikasi dataset MNIST menggunakan PyTorch. Pertama, kelas Net mendefinisikan arsitektur model dengan tiga komponen utama: layer konvolusi pertama yang mengubah gambar grayscale berukuran 28x28 menjadi 12 channel, layer pooling untuk mengurangi ukuran gambar dengan ukuran kernel 2x2, dan layer Fully Connected (FC) yang menghasilkan output untuk 10 kelas (untuk klasifikasi digit 0-9). Fungsi forward menangani aliran data melalui model, dengan langkah-langkah reshape

input, konvolusi, aktivasi ReLU, pooling, dan akhirnya log softmax untuk menghasilkan probabilitas log untuk setiap kelas.

Data pelatihan dan pengujian dimuat menggunakan DataLoader dengan batch size 32. Pemilihan perangkat dilakukan berdasarkan ketersediaan GPU, jika tidak ada, proses pelatihan dilakukan di CPU. Model diinisialisasi dan dipindahkan ke perangkat yang sesuai, dengan optimizer Adam digunakan untuk memperbarui bobot model. Selama pelatihan, model melakukan forward pass untuk menghasilkan output, menghitung Negative Log-Likelihood Loss, melakukan backward pass untuk menghitung gradien, dan memperbarui bobot menggunakan optimizer. Progres pelatihan dan nilai loss ditampilkan untuk setiap batch dalam satu epoch pelatihan.

### Kuantisasi Model

Setelah pelatihan, kita dapat mengkuantisasi model menggunakan fungsi ``torch.quantization.quantize_dynamic`` dari pytorch.

```
# Memindahkan model ke CPU
model.to('cpu')

# Melakukan quantization dinamis pada model, dengan mengubah tipe data layer
Linear menjadi 8-bit integer (qint8)
quantized_model = torch.quantization.quantize_dynamic(model, {torch.nn.Linear},
dtype=torch.qint8)
```

Kode ini memindahkan model ke CPU menggunakan perintah `model.to('cpu')`, yang memastikan bahwa model akan dijalankan pada perangkat CPU meskipun awalnya mungkin diinisialisasi untuk GPU. Selanjutnya, kode ini melakukan quantization dinamis pada model menggunakan fungsi `torch.quantization.quantize_dynamic`. Quantization adalah teknik untuk mengurangi ukuran model dan meningkatkan kecepatan inferensi dengan mengubah representasi parameter model dari floating point (biasanya 32-bit) menjadi format yang lebih rendah, seperti 8-bit integer (qint8). Dalam hal ini, proses quantization hanya diterapkan pada layer Linear dari model, yang merupakan lapisan yang menghubungkan neuron-neuron dalam jaringan saraf. Dengan mengonversi layer-layer tersebut menjadi tipe data yang lebih efisien, model menjadi lebih kecil dan lebih cepat, dengan sedikit atau tanpa kehilangan akurasi.

### Periksa Ukuran Model

Kita dapat melihat bahwa model terkuantisasi jauh lebih kecil daripada model asli

```
# Menentukan path direktori untuk menyimpan model
models_dir = pathlib.Path("./models/")
models_dir.mkdir(exist_ok=True, parents=True)

# Menyimpan model yang telah dilatih (model asli) ke dalam file dengan ekstensi
.p
torch.save(model.state_dict(), "./models/original_model.p")
```

```
# Menyimpan model yang sudah melalui proses quantization ke dalam file dengan
ekstensi .p
torch.save(quantized_model.state_dict(), "./models/quantized_model.p")

# Menampilkan daftar isi dari direktori "models" beserta ukuran file
%ls -lh models
```

Kode ini menyimpan dua versi model yang telah dilatih: model asli dan model yang telah melalui proses quantization. Pertama, direktori ./models/ dibuat jika belum ada menggunakan pathlib.Path. Kemudian, model asli disimpan dengan menggunakan torch.save ke dalam file original\_model.p, sementara model yang sudah melalui quantization disimpan dalam file quantized\_model.p. Terakhir, perintah %ls -lh models digunakan untuk menampilkan daftar file yang ada di dalam direktori ./models/ beserta ukuran masing-masing file.

### Periksa Akurasi

Kita dapat melihat bahwa model terkuantisasi memiliki akurasi yang sebanding dengan model asli

```
# Fungsi untuk menguji akurasi model
def test(model, device, data_loader, quantized=False):
    # Memindahkan model ke device (misalnya CPU atau GPU)
    model.to(device)
    # Menetapkan model ke mode evaluasi (non-training)
    model.eval()

    test_loss = 0 # Untuk menyimpan total kerugian
    correct = 0 # Untuk menghitung jumlah prediksi yang benar

    # Nonaktifkan perhitungan gradient (untuk menghemat memori dan mempercepat
inference)
    with torch.no_grad():
        # Loop untuk setiap batch data di data_loader
        for data, target in data_loader:
            # Memindahkan data dan target ke device yang sama dengan model
            data, target = data.to(device), target.to(device)
            # Melakukan prediksi dengan model
            output = model(data)
            # Menghitung kerugian menggunakan Negative Log Likelihood Loss
            test_loss += F.nll_loss(output, target, reduction='sum').item() #
Menjumlahkan kerugian per batch
            # Menentukan prediksi dengan nilai probabilitas terbesar
            pred = output.argmax(dim=1, keepdim=True)
            # Menghitung jumlah prediksi yang benar
```

```

        correct += pred.eq(target.view_as(pred)).sum().item()

# Menghitung rata-rata kerugian per sampel
test_loss /= len(data_loader.dataset)

# Mengembalikan akurasi dalam persen
return 100. * correct / len(data_loader.dataset)

# Menguji akurasi model asli
original_acc = test(model, "cpu", test_loader)
# Menguji akurasi model yang telah diquantize
quantized_acc = test(quantized_model, "cpu", test_loader)

# Menampilkan hasil akurasi dari kedua model
print('Original model accuracy: {:.0f}%'.format(original_acc))
print('Quantized model accuracy: {:.0f}%'.format(quantized_acc))

```

Fungsi ini digunakan untuk menguji akurasi model dengan menghitung seberapa baik model melakukan prediksi pada dataset pengujian. Fungsi test menerima empat argumen utama: model yang diuji, perangkat (CPU atau GPU), data loader yang menyediakan data pengujian, dan opsional parameter quantized untuk menandakan apakah model yang diuji adalah model yang sudah melalui proses quantization.

Proses dimulai dengan memindahkan model ke perangkat yang sesuai dan mengubah mode model menjadi eval (mode evaluasi, bukan pelatihan). Kemudian, fungsi melakukan inferensi pada setiap batch data dari data loader tanpa menghitung gradien (menggunakan torch.no\_grad() untuk menghemat memori). Untuk setiap batch, fungsi menghitung kerugian dengan menggunakan Negative Log Likelihood Loss dan menghitung jumlah prediksi yang benar. Akhirnya, akurasi dihitung sebagai persentase prediksi yang benar dibandingkan dengan total jumlah data.

Setelah itu, fungsi test dipanggil untuk menguji kedua model: model asli dan model yang telah diquantize. Akurasi dari kedua model kemudian dicetak dengan format persentase.

### Kuantisasi Statis Pasca-pelatihan

Kuantisasi statis pasca-pelatihan adalah saat bobot dan aktivasi dikuantisasi dan kalibrasi diperlukan pasca-pelatihan. Di sini, kami mengkuantisasi model menggunakan fungsi `torch.quantization.quantize_fx()` dari PyTorch dan membandingkan akurasi dan ukuran model yang dikuantisasi dengan model FP32 asli.

Untuk mengkuantisasi menggunakan alat kuantisasi statis pasca-pelatihan, pertama-tama tentukan model atau muat model yang telah dilatih sebelumnya, lalu buat pemetaan konfigurasi kuantisasi menggunakan default untuk mesin QNNPACK. Atur model ke mode evaluasi dan buat tensor



masukannya. Kemudian, persiapkan model untuk kuantisasi menggunakan fungsi `quantize_fx.prepare_fx()`. Ini melibatkan penerapan pemetaan konfigurasi kuantisasi dan persiapan model untuk menangani presisi int8. Model yang disiapkan kemudian dieksekusi pada tensor masukan. Terakhir, model terkuantisasi dengan memanggil `quantize_fx.convert_fx()` dan menyimpan model ke disk.

```
# Mengimpor modul yang diperlukan untuk kuantisasi
from torch.ao.quantization import (
    get_default_qconfig_mapping,
    get_default_qat_qconfig_mapping,
    QConfigMapping,
)
import torch.ao.quantization.quantize_fx as quantize_fx
import copy

# Memuat model yang telah dilatih sebelumnya
loaded_model = Net()
loaded_model.load_state_dict(torch.load("./models/original_model.p"))
model_to_quantize = copy.deepcopy(loaded_model)

# Mendapatkan konfigurasi kuantisasi default untuk 'qnnpack' (algoritma
kuantisasi yang digunakan)
qconfig_mapping = get_default_qconfig_mapping("qnnpack")

# Menetapkan model ke mode evaluasi
model_to_quantize.eval()

# Mengambil input FP32 pertama dari data test untuk mempersiapkan model untuk
kuantisasi
input_fp32 = next(iter(test_loader))[0][0:1]
input_fp32.to('cpu') # Memindahkan input ke CPU

# Mempersiapkan model untuk kuantisasi (membuat versi model yang siap untuk
kuantisasi)
model_fp32_prepared = quantize_fx.prepare_fx(model_to_quantize, qconfig_mapping,
input_fp32)

# Menjalankan model yang sudah dipersiapkan untuk kuantisasi dengan input FP32
model_fp32_prepared(input_fp32)

# Mengonversi model ke format INT8 setelah kuantisasi
model_int8 = quantize_fx.convert_fx(model_fp32_prepared)

# Menyimpan model yang telah dikuantisasi ke dalam file
torch.save(model_int8.state_dict(), "./models/post_quantized_model.p")
```

Kode ini melakukan kuantisasi pada model deep learning yang telah dilatih sebelumnya untuk meningkatkan efisiensi. Pertama, model yang telah dilatih dimuat kembali, dan salinan model dibuat. Konfigurasi kuantisasi default untuk algoritma qnnpack digunakan untuk mempersiapkan model agar dapat dikuantisasi. Input dari dataset pengujian digunakan untuk mempersiapkan model untuk kuantisasi. Setelah itu, model diproses menggunakan `quantize_fx.prepare_fx` untuk persiapan dan diuji pada input FP32. Selanjutnya, model dikonversi ke format INT8, yang lebih efisien dalam hal penggunaan memori dan komputasi. Model yang sudah dikuantisasi kemudian disimpan ke dalam file untuk penggunaan lebih lanjut.

### Periksa Ukuran Model

Sekali lagi, kita dapat melihat bahwa model terkuantisasi jauh lebih kecil daripada model asli

```
%ls -lh models
```

Perintah `%ls -lh models` digunakan untuk menampilkan daftar file yang ada dalam direktori `models` beserta informasi ukuran file dalam format yang lebih mudah dibaca (dalam byte, kilobyte, megabyte, dll). Perintah ini juga memberikan detail lain seperti hak akses file, jumlah link, pemilik, dan tanggal serta waktu terakhir file diubah. Dengan menggunakan perintah ini, pengguna dapat memeriksa isi direktori dan memastikan model yang telah disimpan, baik model asli maupun yang telah dikuantisasi, ada di dalam direktori tersebut beserta ukuran filenya.

### Periksa Akurasi

Sekali lagi, kita dapat melihat bahwa akurasi model terkuantisasi tidak jauh berbeda dengan akurasi aslinya

```
# Menguji model yang telah dikuantisasi
quantized_acc = test(model_int8, "cpu", test_loader, quantized=True)

# Mencetak akurasi model yang telah dikuantisasi
print('Post quantized model accuracy: {:.0f}%'.format(quantized_acc))
```

Kode ini digunakan untuk menguji akurasi model yang telah dikuantisasi setelah melalui proses kuantisasi. Fungsi `test()` dipanggil dengan model yang sudah dikuantisasi (`model_int8`), menggunakan data pengujian (`test_loader`), dan device yang dipilih (`"cpu"`). Parameter `quantized=True` menunjukkan bahwa model yang diuji adalah model yang telah mengalami kuantisasi. Setelah pengujian selesai, akurasi model yang telah dikuantisasi dihitung dan dicetak dalam persen dengan format `Post quantized model accuracy: ...%`. Hal ini bertujuan untuk melihat seberapa baik kinerja model setelah dikonversi ke format INT8.

## 2. Tmo (Chapter 9)

### Tensorflow Model Optimization Toolkit (TMO)

Dalam buku catatan ini, kami akan menunjukkan cara menggunakan TMO untuk mengoptimalkan model untuk penerapan. Kami melatih model pada set data MNIST lalu mengoptimalkannya menggunakan TMO. Kami kemudian akan membandingkan ukuran dan akurasi model yang dioptimalkan dengan model asli.

Pertama, kita instal TMO dan impor paket yang dibutuhkan.

```
%pip install -q tensorflow
%pip install -q tensorflow-model-optimization
```

Perintah `%pip install -q tensorflow` dan `%pip install -q tensorflow-model-optimization` digunakan untuk menginstal pustaka TensorFlow dan TensorFlow Model Optimization di lingkungan Python. TensorFlow adalah pustaka populer untuk pengembangan dan pelatihan model deep learning, sementara TensorFlow Model Optimization menyediakan berbagai alat untuk mengoptimalkan model, seperti kuantisasi dan pemangkasan, yang dapat meningkatkan efisiensi dan performa model. Opsi `-q` digunakan untuk mengurangi jumlah output selama proses instalasi agar lebih ringkas.

```
import tensorflow as tf
import tensorflow_model_optimization as tfmot
from tensorflow import keras
import pathlib
import numpy as np
```

Kode ini mengimpor beberapa pustaka yang diperlukan untuk bekerja dengan model deep learning di TensorFlow. `tensorflow` adalah pustaka utama untuk membangun dan melatih model, sementara `tensorflow_model_optimization` (`tfmot`) menyediakan alat untuk mengoptimalkan model, seperti kuantisasi dan pemangkasan. `keras` adalah API tinggi yang digunakan untuk membangun model dalam TensorFlow, dan `pathlib` digunakan untuk bekerja dengan jalur file dan direktori. `numpy` adalah pustaka untuk komputasi numerik, yang sering digunakan untuk manipulasi data dalam bentuk array multidimensi. Semua pustaka ini digunakan bersama untuk mengembangkan dan mengoptimalkan model deep learning.

```
# Memuat dataset MNIST
mnist = keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Menormalisasi gambar input sehingga setiap nilai piksel berada di antara 0 dan 1.
train_images = train_images / 255.0
test_images = test_images / 255.0
```

Kode ini memuat dataset MNIST, yang berisi gambar angka tangan (0-9) beserta labelnya. Dataset dibagi menjadi dua bagian: data pelatihan (`train_images` dan `train_labels`) dan data pengujian (`test_images` dan `test_labels`). Kemudian, gambar-gambar input dinormalisasi dengan membagi setiap nilai pikselnya dengan 255, sehingga nilai piksel berada di rentang 0 hingga 1.

Normalisasi ini penting untuk mempercepat proses pelatihan model deep learning dengan membuat skala data lebih konsisten.

```
# Mendefinisikan arsitektur model
model = keras.Sequential([
    keras.layers.InputLayer(input_shape=(28, 28)), # Lapisan input dengan bentuk (28, 28)
    keras.layers.Reshape(target_shape=(28, 28, 1)), # Mengubah bentuk input menjadi (28, 28, 1)
    keras.layers.Conv2D(filters=12, kernel_size=(3, 3), activation=tf.nn.relu), # Lapisan konvolusi dengan 12 filter dan ukuran kernel 3x3
    keras.layers.MaxPooling2D(pool_size=(2, 2)), # Lapisan pooling maksimum dengan ukuran 2x2
    keras.layers.Flatten(), # Mengubah array multidimensi menjadi satu dimensi
    keras.layers.Dense(10) # Lapisan dense dengan 10 neuron (sesuai dengan jumlah kelas digit)
])

# Melatih model klasifikasi digit
model.compile(optimizer='adam', # Optimizer Adam
              loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True), # Fungsi kerugian untuk klasifikasi
              metrics=['accuracy']) # Metrik akurasi
model.fit(
    train_images, # Data pelatihan
    train_labels, # Label pelatihan
    epochs=1, # Jumlah epoch pelatihan
    validation_data=(test_images, test_labels) # Data validasi
)
```

Kode ini mendefinisikan dan melatih model deep learning untuk klasifikasi digit menggunakan dataset MNIST. Arsitektur model terdiri dari lapisan input, reshaping untuk menyesuaikan bentuk data gambar, lapisan konvolusi dengan filter untuk ekstraksi fitur, lapisan pooling untuk mereduksi dimensi, lapisan flatten untuk mengubah data menjadi vektor satu dimensi, dan lapisan dense untuk klasifikasi dengan 10 neuron (sesuai jumlah kelas digit). Model menggunakan optimizer Adam dan fungsi kerugian SparseCategoricalCrossentropy untuk klasifikasi, serta metrik akurasi. Model dilatih selama 1 epoch dengan data pelatihan dan divalidasi menggunakan data pengujian.

```
tflite_models_dir = pathlib.Path("notebooks/Unit 9 - Model Optimization/models")
tflite_models_dir.mkdir(exist_ok=True, parents=True) # Membuat direktori untuk menyimpan model jika belum ada
converter = tf.lite.TFLiteConverter.from_keras_model(model) # Menginisialisasi konverter dari model Keras

# Tanpa kuantisasi
```

```

tflite_model = converter.convert() # Mengonversi model ke format TFLite
tflite_model_file = tflite_models_dir/"original_model.tflite" # Menentukan nama
file untuk model TFLite asli
tflite_model_file.write_bytes(tflite_model) # Menyimpan model ke file

# Dengan kuantisasi
converter.optimizations = [tf.lite.Optimize.DEFAULT] # Mengaktifkan optimisasi
default untuk kuantisasi
tflite_quant_model = converter.convert() # Mengonversi model ke format TFLite
dengan kuantisasi
tflite_model_quant_file = tflite_models_dir/"quantized_model.tflite" #
Menentukan nama file untuk model TFLite dengan kuantisasi
tflite_model_quant_file.write_bytes(tflite_quant_model) # Menyimpan model yang
telah dikuantisasi ke file

```

Kode ini mengonversi model Keras menjadi format TFLite (TensorFlow Lite) untuk digunakan di perangkat dengan sumber daya terbatas, seperti smartphone atau perangkat embedded. Pertama, direktori `tflite_models_dir` dibuat untuk menyimpan model yang dikonversi. Konverter `TFLiteConverter` digunakan untuk mengonversi model Keras ke format TFLite. Tanpa optimisasi atau kuantisasi, model dikonversi dan disimpan sebagai `original_model.tflite`. Selanjutnya, optimisasi untuk kuantisasi diaktifkan (`tf.lite.Optimize.DEFAULT`) untuk menghasilkan model yang lebih ringan dan efisien, yang kemudian dikonversi dan disimpan sebagai `quantized_model.tflite`. Kedua model tersebut disimpan dalam direktori yang telah ditentukan.

```
%ls -lh {tflite_models_dir}
```

Perintah `%ls -lh {tflite_models_dir}` akan menampilkan daftar file di dalam direktori `tflite_models_dir` dengan informasi ukuran file yang lebih mudah dibaca (misalnya dalam byte, kilobyte, atau megabyte). Ini berguna untuk memeriksa apakah model yang telah dikonversi dan dikuantisasi berhasil disimpan dengan ukuran yang sesuai. Dengan menggunakan flag `-lh`, hasilnya akan menyertakan informasi tambahan, seperti nama file, ukuran file, tanggal pembuatan atau modifikasi, dan lainnya, sehingga Anda bisa memverifikasi file mana yang ada di dalam direktori dan ukurannya.

```

# Fungsi pembantu untuk mengevaluasi model TF Lite menggunakan dataset "test".
def evaluate_model(interpreter):
    input_index = interpreter.get_input_details()[0]["index"] # Mendapatkan
indeks input model
    output_index = interpreter.get_output_details()[0]["index"] # Mendapatkan
indeks output model

    # Melakukan prediksi pada setiap gambar dalam dataset "test".
    prediction_digits = []
    for test_image in test_images:

```

```

        # Pra-pemrosesan: tambahkan dimensi batch dan ubah tipe data ke float32
        # agar sesuai dengan format data input model.
        test_image = np.expand_dims(test_image, axis=0).astype(np.float32)
        interpreter.set_tensor(input_index, test_image) # Menetapkan tensor
input

        # Menjalankan inferensi.
        interpreter.invoke()

        # Pasca-pemrosesan: hapus dimensi batch dan temukan digit dengan
        # probabilitas tertinggi.
        output = interpreter.tensor(output_index)
        digit = np.argmax(output()[0]) # Mendapatkan indeks dengan nilai
maksimum
        prediction_digits.append(digit)

        # Membandingkan hasil prediksi dengan label ground truth untuk menghitung
akurasi.
        accurate_count = 0
        for index in range(len(prediction_digits)):
            if prediction_digits[index] == test_labels[index]:
                accurate_count += 1
        accuracy = accurate_count * 1.0 / len(prediction_digits) # Menghitung
akurasi

        return accuracy

# Menginisialisasi interpreter untuk model asli.
interpreter = tf.lite.Interpreter(model_path=str(tflite_model_file))
interpreter.allocate_tensors() # Mengalokasikan tensor
print("Akurasi model asli = ", evaluate_model(interpreter)) # Evaluasi model
asli

# Menginisialisasi interpreter untuk model kuantisasi.
interpreter_quant = tf.lite.Interpreter(model_path=str(tflite_model_quant_file))
interpreter_quant.allocate_tensors() # Mengalokasikan tensor
print("Akurasi model kuantisasi = ", evaluate_model(interpreter_quant)) #
Evaluasi model kuantisasi

```

Pada kode ini, pertama-tama didefinisikan fungsi `evaluate_model`, yang digunakan untuk mengevaluasi akurasi model TensorFlow Lite (TFLite). Fungsi ini menerima interpreter sebagai parameter, yang merupakan objek untuk menginterpretasikan model TFLite yang telah dikonversi. Di dalam fungsi ini, pertama-tama diminta input dan output tensor dari model TFLite. Kemudian, untuk setiap gambar dalam dataset `test_images`, gambar tersebut diproses agar sesuai dengan format input model (dalam hal ini

dengan menambahkan dimensi batch dan mengubah tipe data ke float32). Gambar yang sudah diproses kemudian diberikan ke interpreter untuk melakukan inferensi menggunakan `interpreter.invoke()`.

Setelah inferensi, hasil output yang diperoleh diproses untuk menentukan angka dengan probabilitas tertinggi (menggunakan `np.argmax`). Kemudian, hasil prediksi dibandingkan dengan label asli dari dataset (`test_labels`) untuk menghitung akurasi model.

Setelah fungsi `evaluate_model` didefinisikan, kode menginisialisasi dua buah interpreter: satu untuk model asli (`tflite_model_file`) dan satu untuk model yang sudah dikuantisasi (`tflite_model_quant_file`). Model-model ini kemudian dievaluasi dengan menggunakan fungsi `evaluate_model`, dan akurasi dari kedua model dicetak.

Dengan cara ini, Anda dapat membandingkan akurasi model asli dan model yang telah dikuantisasi pada dataset yang sama.

```
prune_low_magnitude = tfmot.sparsity.keras.prune_low_magnitude # Mengimpor
fungsi pruning

# Menghitung langkah akhir untuk menyelesaikan pruning setelah 2 epoch.
batch_size = 128
epochs = 2
validation_split = 0.1 # 10% dari dataset pelatihan akan digunakan sebagai
dataset validasi.

num_images = train_images.shape[0] * (1 - validation_split) # Jumlah gambar
untuk pelatihan
end_step = np.ceil(num_images / batch_size).astype(np.int32) * epochs # Langkah
akhir pruning

# Mendefinisikan model untuk pruning.
pruning_params = {
    'pruning_schedule': tfmot.sparsity.keras.PolynomialDecay(
        initial_sparsity=0.50, # Tingkat sparsitas awal
        final_sparsity=0.80, # Tingkat sparsitas akhir
        begin_step=0, # Langkah awal untuk memulai pruning
        end_step=end_step # Langkah akhir untuk menyelesaikan pruning
    )
}

# Membuat model yang mendukung pruning.
model_for_pruning = prune_low_magnitude(model, **pruning_params)

# `prune_low_magnitude` membutuhkan recompilasi model.
model_for_pruning.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy']
)
```

```

)

# Menampilkan ringkasan model.
print(model_for_pruning.summary())

# Mendefinisikan callback untuk memperbarui langkah pruning.
callbacks = [
    tfmot.sparsity.keras.UpdatePruningStep(),
]

# Melatih model yang mendukung pruning.
model_for_pruning.fit(
    train_images, train_labels,
    batch_size=batch_size, epochs=epochs, validation_split=validation_split,
    callbacks=callbacks
)

```

Pada kode ini, pertama-tama diimpor fungsi `prune_low_magnitude` dari `tensorflow_model_optimization` (`tfmot`), yang digunakan untuk menerapkan teknik pruning pada model. Pruning adalah teknik untuk mengurangi ukuran model dengan menghapus bobot yang kurang penting atau mendekati nol, yang dapat meningkatkan efisiensi model dan mengurangi ukuran model.

Langkah pertama adalah mendefinisikan beberapa parameter untuk proses pruning. `end_step` dihitung berdasarkan jumlah gambar pelatihan, ukuran batch, dan jumlah epoch yang akan dijalani (2 epoch dalam hal ini). Ini menentukan berapa langkah pelatihan yang diperlukan untuk menyelesaikan proses pruning.

Kemudian, parameter pruning dijelaskan menggunakan `PolynomialDecay` yang mengatur tingkat sparsitas (persentase bobot yang akan dipangkas). `initial_sparsity` ditetapkan pada 50%, artinya setengah dari bobot model akan dipangkas pada awalnya, dan `final_sparsity` ditetapkan pada 80%, yang berarti 80% bobot akan dipangkas pada akhir proses pruning.

Setelah itu, model yang mendukung pruning dibuat dengan menggunakan fungsi `prune_low_magnitude`. Model ini perlu dikompilasi ulang untuk mendukung proses pelatihan dengan pruning.

Selanjutnya, callback `UpdatePruningStep` didefinisikan untuk memperbarui langkah pruning selama pelatihan. Model yang sudah dioptimalkan dengan pruning kemudian dilatih menggunakan dataset `train_images` dan `train_labels` selama 2 epoch, dengan 10% dari dataset digunakan untuk validasi. Callback akan memastikan langkah pruning diperbarui selama pelatihan.

Ringkasan model yang telah dimodifikasi untuk pruning ditampilkan menggunakan `model_for_pruning.summary()`.

```

# Mengevaluasi akurasi model awal (tanpa pruning).
_, baseline_model_accuracy = model.evaluate(
    test_images, test_labels, verbose=0)

# Mengevaluasi akurasi model setelah pruning.

```



```
_, model_for_pruning_accuracy = model_for_pruning.evaluate(
    test_images, test_labels, verbose=0)

# Menampilkan hasil akurasi.
print('Akurasi model awal:', baseline_model_accuracy)
print('Akurasi model setelah pruning:', model_for_pruning_accuracy)
```

Pada kode ini, pertama-tama dilakukan evaluasi akurasi untuk dua model: model awal (tanpa pruning) dan model yang telah melalui proses pruning.

1. **Evaluasi Model Awal:** Fungsi `model.evaluate()` digunakan untuk mengukur akurasi dari model asli (sebelum pruning). Data uji `test_images` dan `test_labels` digunakan untuk mengevaluasi model. Hasil evaluasi ini disimpan dalam variabel `baseline_model_accuracy`.
2. **Evaluasi Model Setelah Pruning:** Setelah model diterapkan teknik pruning, model yang sudah dipangkas (dalam variabel `model_for_pruning`) dievaluasi dengan cara yang sama menggunakan data uji. Hasil evaluasi ini disimpan dalam variabel `model_for_pruning_accuracy`.
3. **Menampilkan Hasil:** Kedua hasil akurasi (sebelum dan setelah pruning) ditampilkan menggunakan `print()`.

Dengan demikian, kode ini digunakan untuk membandingkan performa akurasi antara model sebelum dan setelah proses pruning, untuk melihat apakah pruning mempengaruhi kinerja model.

```
# Menghapus pruning dari model untuk menyiapkan model yang telah dipangkas.
model_for_export = tfmot.sparsity.keras.strip_pruning(model_for_pruning)

# Mengonversi model yang telah dipangkas ke format TFLite.
pruning_converter = tf.lite.TFLiteConverter.from_keras_model(model_for_export)
pruned_tflite_model = pruning_converter.convert()

# Menyimpan model TFLite yang telah dipangkas ke dalam file.
pruned_model_file = tflite_models_dir/"pruned_model.tflite"
pruned_model_file.write_bytes(pruned_tflite_model)
```

Pada kode ini, proses dimulai dengan menghapus efek pruning dari model yang telah dipangkas menggunakan fungsi `strip_pruning()` dari TensorFlow Model Optimization. Fungsi ini menghapus informasi pruning yang diterapkan sebelumnya, sehingga model siap untuk diekspor.

Setelah itu, model yang telah dipangkas disiapkan untuk diubah menjadi format TensorFlow Lite (TFLite). Proses ini dilakukan dengan menggunakan `TFLiteConverter.from_keras_model()` untuk mengonversi model Keras ke model TFLite. Model yang sudah dipangkas ini kemudian diubah menjadi format TFLite menggunakan metode `convert()`.

Terakhir, model TFLite yang telah dipangkas disimpan ke dalam file dengan ekstensi .tflite menggunakan fungsi `write_bytes()`. File ini disimpan di direktori yang telah ditentukan sebelumnya (`tflite_models_dir`), dengan nama file `pruned_model.tflite`.

Tujuan dari langkah-langkah ini adalah untuk menghasilkan model TFLite yang lebih ringan dan lebih efisien setelah proses pruning, yang dapat digunakan pada perangkat dengan sumber daya terbatas.

```
%ls -lh {tflite_models_dir}
```

Perintah `%ls -lh {tflite_models_dir}` digunakan untuk menampilkan daftar isi direktori `tflite_models_dir` dengan informasi yang lebih rinci, seperti ukuran file. Hasilnya adalah tampilan yang menunjukkan ukuran file setiap model TFLite yang telah disimpan dalam direktori tersebut, termasuk model asli, model yang dikuantisasi, dan model yang telah dipangkas. Perintah ini membantu untuk memastikan bahwa file yang dihasilkan telah disimpan dengan benar dan memungkinkan Anda memeriksa ukuran file untuk mengevaluasi efisiensi model setelah melalui berbagai proses optimasi (seperti kuantisasi dan pruning).

#### 4. TensorRT (Chapter 9)

Dalam notebook ini, kita akan menggunakan TensorRT untuk mengoptimalkan model PyTorch untuk inferensi. Kita akan melatih model CNN sederhana pada set data MNIST, mengonversinya ke mesin TensorRT menggunakan ONNX, lalu melakukan inferensi menggunakan model mesin TensorRT yang dioptimalkan dan mengevaluasi ukuran dan akurasi model. Notebook ini memerlukan GPU NVIDIA dengan dukungan CUDA atau perangkat NVIDIA Jetson.

```
%pip install torch torchvision
%pip install tensorrt==8.6.1
%pip install pycuda
%pip install pycuda onnx onnxruntime
%pip install onnxruntime
%pip install --no-cache-dir --extra-index-url https://pypi.nvidia.com pytorch-quantization==2.1.2
```

Perintah-perintah yang Anda sebutkan adalah untuk menginstal berbagai pustaka dan alat yang digunakan dalam pemrograman dan optimasi model AI, khususnya terkait dengan PyTorch, TensorRT, dan ONNX:

1. **%pip install torch torchvision:** Menginstal PyTorch (pustaka untuk pembelajaran mesin dan jaringan saraf) dan torchvision (pustaka untuk pengolahan gambar dan transformasi data gambar).
2. **%pip install tensorrt==8.6.1:** Menginstal TensorRT versi 8.6.1, pustaka dari NVIDIA untuk mengoptimalkan model AI, khususnya untuk inferensi yang sangat efisien menggunakan GPU NVIDIA.
3. **%pip install pycuda:** Menginstal PyCUDA, pustaka yang memungkinkan penggunaan CUDA (Compute Unified Device Architecture) dari NVIDIA untuk akselerasi komputasi menggunakan GPU.
4. **%pip install pycuda onnx onnxruntime:** Menginstal PyCUDA, ONNX (Open Neural Network Exchange), dan ONNX Runtime. ONNX adalah format pertukaran model yang memungkinkan penggunaan model yang dikembangkan di berbagai platform AI, dan ONNX Runtime adalah pustaka untuk inferensi model ONNX.

5. **%pip install onnxruntime**: Menginstal ONNX Runtime, pustaka untuk menjalankan model ONNX secara efisien pada berbagai perangkat keras.
6. **%pip install --no-cache-dir --extra-index-url https://pypi.nvidia.com pytorch-quantization==2.1.2**: Menginstal pustaka pytorch-quantization versi 2.1.2 dari indeks tambahan NVIDIA, yang digunakan untuk kuantisasi model PyTorch agar lebih efisien untuk inferensi, terutama pada perangkat keras yang mendukung operasi dengan presisi lebih rendah.

```
7. import torch
8. import torch.nn as nn
9. import torch.nn.functional as F
10. import torch.optim as optim
11. from torchvision import datasets, transforms
12. import torch.quantization
13. import pathlib
14. import numpy as np
15. import torch.onnx
16. import tensorrt as trt
17. import pycuda.driver as cuda
18. import pycuda.autoinit
19. import onnx
20. import onnxruntime
21.
22. from pytorch_quantization import nn as quant_nn
23. from pytorch_quantization import quant_modules
24. from pytorch_quantization import calib
25. from tqdm import tqdm
```

The code imports several key libraries for deep learning and optimization. It uses PyTorch (torch, torch.nn, torch.optim) for building and training neural networks, as well as TorchVision for data loading and preprocessing. Quantization tools from pytorch\_quantization are imported to optimize model size and performance. TensorRT and ONNX are included for GPU acceleration and model interoperability. Additional libraries like pycuda enable GPU memory management, while onnxruntime allows executing ONNX models. Finally, tqdm is used for displaying progress bars during loops. These libraries are essential for developing, optimizing, and deploying efficient deep learning models.

```
transform=transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])

train_dataset = datasets.MNIST('./data', train=True,
download=True, transform=transform)
test_dataset = datasets.MNIST('./data', train=False, transform=transform)
```

```

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=12, kernel_size=3)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.fc = nn.Linear(12 * 13 * 13, 10)

    def forward(self, x):
        x = x.view(-1, 1, 28, 28)
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        output = F.log_softmax(x, dim=1)
        return output

train_loader = torch.utils.data.DataLoader(train_dataset, 32)
test_loader = torch.utils.data.DataLoader(test_dataset, 32)

device = "cpu"

epochs = 1

model = Net().to(device)
optimizer = optim.Adam(model.parameters())

model.train()

for epoch in range(1, epochs+1):
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
            epoch, batch_idx * len(data), len(train_loader.dataset),
            100. * batch_idx / len(train_loader), loss.item()))

MODEL_DIR = pathlib.Path("./models")
MODEL_DIR.mkdir(exist_ok=True)
torch.save(model.state_dict(), MODEL_DIR / "original_model.p")

```

```

x, _ = next(iter(train_loader))
torch.onnx.export(model,
                  x,
                  MODEL_DIR / "mnist_model.onnx",
                  export_params=True,
                  opset_version=10,
                  do_constant_folding=True,
                  input_names = ['input'],
                  output_names = ['output'],
                  dynamic_axes={'input' : {0 : 'batch_size'},
                              'output' : {0 : 'batch_size'}})

```

The code trains a simple convolutional neural network (CNN) on the MNIST dataset for digit classification. The MNIST dataset is loaded and preprocessed with normalization. The network consists of one convolutional layer followed by a max-pooling layer, and a fully connected layer. The training loop iterates over the dataset, computes the loss using negative log-likelihood, and updates the model using the Adam optimizer. After training, the model's parameters are saved to a .p file, and the trained model is exported to ONNX format, which allows it to be used across different frameworks. The model is stored in a directory called models, with both the PyTorch and ONNX model files saved for further use.

```

onnx_path = MODEL_DIR / "mnist_model.onnx"
trt_path = MODEL_DIR / 'mnist_engine_pytorch.trt'

# inisialisasi engine TensorRT dan parsing model ONNX
logger = trt.Logger(trt.Logger.WARNING)
builder = trt.Builder(logger)
network = builder.create_network(1 <<
int(trt.NetworkDefinitionCreationFlag.EXPLICIT_BATCH))

parser = trt.OnnxParser(network, logger)
parser.parse_from_file(str(onnx_path))

# menyiapkan konfigurasi builder dan profil optimasi
config = builder.create_builder_config()
config.set_memory_pool_limit(trt.MemoryPoolType.WORKSPACE, 1 << 30)

profile = builder.create_optimization_profile()
profile.set_shape("input", (32, 1, 28, 28), (32, 1, 28, 28), (32, 1, 28, 28))
config.add_optimization_profile(profile)

# serialisasi engine, kemudian simpan ke disk
serialized_engine = builder.build_serialized_network(network, config)
with open(str(trt_path), 'wb') as f:
    f.write(serialized_engine)

# bebaskan sumber daya

```

```
del builder
del network
```

The code converts a trained ONNX model to a TensorRT engine for optimized inference on NVIDIA GPUs. First, it initializes the TensorRT builder and logger. The ONNX model is loaded and parsed into a TensorRT network. Optimization profiles are set to define the expected input shape, and a memory pool is allocated for efficient execution. The builder then serializes the optimized network into a TensorRT engine, which is saved to disk for later use. This process helps accelerate inference on supported hardware by leveraging TensorRT's optimizations, such as layer fusion and kernel optimization. After building the engine, the resources are cleaned up.

```
def to_numpy(tensor):
    return tensor.detach().cpu().numpy() if tensor.requires_grad else
tensor.cpu().numpy()

def test_onnx(model_name, data_loader):
    onnx_model = onnx.load(model_name)
    onnx.checker.check_model(onnx_model)
    ort_session = onnxruntime.InferenceSession(model_name)
    test_loss = 0
    correct = 0
    for data, target in data_loader:
        ort_inputs = {ort_session.get_inputs()[0].name: to_numpy(data)}
        output = ort_session.run(None, ort_inputs)[0]
        output = torch.from_numpy(output)
        if target.shape[0] == 32: # batch terakhir mungkin lebih kecil dari 32
(perbaiki sementara)
            test_loss += F.nll_loss(output, target, reduction='sum').item() #
jumlahkan loss batch
            pred = output.argmax(dim=1, keepdim=True) # ambil indeks dengan
probabilitas log maksimum
            correct += pred.eq(target.view_as(pred)).sum().item()
        test_loss /= len(data_loader.dataset)
    return 100. * correct / len(data_loader.dataset)

def test_tensorrt(model_name, data_loader):
    with open(model_name, "rb") as f:
        serialized_engine = f.read()
    runtime = trt.Runtime(logger)
    engine = runtime.deserialize_cuda_engine(serialized_engine)
    context = engine.create_execution_context()
    input_size = trt.volume(engine.get_binding_shape(0))
    output_size = trt.volume(engine.get_binding_shape(1))
    # Alokasikan memori di perangkat
```

```

    d_input = cuda.mem_alloc(input_size * 4) # Asumsikan tipe data float32 4-
byte
    d_output = cuda.mem_alloc(output_size * 4)
    bindings=[int(d_input), int(d_output)]
    stream = cuda.Stream()
    h_output = np.empty(output_size, dtype=np.float32)
    test_loss = 0
    correct = 0
    for data, target in data_loader:
        # Buat array numpy untuk menampung data input dan output
        h_input = data.numpy().astype(np.float32)
        # Transfer data input ke perangkat
        cuda.memcpy_htod_async(d_input, h_input, stream)
        # Eksekusi model
        context.execute_async_v2(bindings, stream.handle, None)
        # Transfer prediksi kembali
        cuda.memcpy_dtoh_async(h_output, d_output, stream)
        # Sinkronkan thread
        stream.synchronize()
        output = h_output.reshape(context.get_tensor_shape('output'))
        output = torch.from_numpy(output)
        if target.shape[0] == 32: # batch terakhir mungkin lebih kecil dari 32
(perbaiki sementara)
            test_loss += F.nll_loss(output, target, reduction='sum').item()
            pred = output.argmax(dim=1, keepdim=True)
            correct += pred.eq(target.view_as(pred)).sum().item()
    test_loss /= len(data_loader.dataset)
    del context
    del engine
    cuda.Context.pop()
    return 100. * correct / len(data_loader.dataset)

acc = test_onnx(onnx_path, test_loader)
print(f"Akurasi model onnx adalah {acc}%")

trtr_acc = test_tensorrt(trt_path, test_loader)
print(f"Akurasi model tensorrt adalah {trtr_acc}%")

```

The code tests the accuracy of models in two formats: ONNX and TensorRT. The `test_onnx` function loads and checks an ONNX model, then uses `onnxruntime` to run inference on the test dataset and calculate accuracy. The `test_tensorrt` function does the same but for a TensorRT-optimized model. It loads the serialized TensorRT engine, allocates memory on the GPU, and performs inference asynchronously. Both functions compute the test loss and accuracy by comparing the model's

predictions with the ground truth labels. The results are printed for both the ONNX and TensorRT models. This allows a comparison of the models' performance in terms of accuracy.

## 5. Optimum (Chapter 9)

Buku catatan ini menunjukkan cara menggunakan Optimum untuk melakukan kuantisasi model yang dihosting di Hugging Face Hub menggunakan alat kuantisasi ONNX Runtime.

```
%pip install optimum
```

The command `%pip install optimum` installs the optimum package, which is a library designed for optimizing machine learning models, particularly those built with Hugging Face Transformers. optimum provides tools to improve model efficiency by enabling quantization, pruning, and the deployment of models on specialized hardware such as GPUs and TPUs. It integrates with popular machine learning frameworks like PyTorch and TensorFlow, allowing users to accelerate inference and reduce model size without significantly impacting accuracy.

```
pip install onnx
```

The command `pip install onnx` installs the onnx package, which provides an open-source framework for representing machine learning models. ONNX (Open Neural Network Exchange) enables models to be transferred between different machine learning frameworks like PyTorch, TensorFlow, and Scikit-Learn. By using ONNX, models trained in one framework can be exported, shared, and run efficiently in another, allowing for cross-platform compatibility. The package provides tools for model conversion, validation, and optimization, making it easier to deploy machine learning models across a wide range of environments.

```
pip install onnxruntime
```

The command `pip install onnxruntime` installs the onnxruntime package, which is a high-performance inference engine for executing machine learning models in the ONNX format. ONNX Runtime is optimized to run models across a variety of platforms and hardware, including CPUs, GPUs, and other accelerators. It supports both training and inference of ONNX models and provides tools for model optimization, speed improvements, and hardware-specific optimizations. By using ONNX Runtime, you can efficiently deploy models in production environments, ensuring fast and scalable inference performance.

```
from optimum.onnxruntime import ORTQuantizer, ORTModelForImageClassification
from functools import partial
from optimum.onnxruntime.configuration import AutoQuantizationConfig,
AutoCalibrationConfig
from onnxruntime.quantization import QuantType
from transformers import AutoFeatureExtractor
from PIL import Image
import requests
```

The code imports necessary libraries for working with ONNX models and optimizing them for inference. It uses Optimum ONNX Runtime tools like ORTQuantizer for model quantization



(reducing model size and increasing speed) and `ORTModelForImageClassification` for classifying images. Additionally, it sets up configurations for automatic quantization and calibration. `AutoFeatureExtractor` helps process input data, such as images, for classification models. PIL and requests are used to handle and load images from URLs. Overall, this setup enables efficient image classification with optimized ONNX models.

```
preprocessor = AutoFeatureExtractor.from_pretrained("optimum/vit-base-patch16-224")
model = ORTModelForImageClassification.from_pretrained("optimum/vit-base-patch16-224")
model.save_pretrained("models/vit-base-patch16-224")
```

The code uses Optimum to load a pre-trained Vision Transformer (ViT) model (vit-base-patch16-224) for image classification. The `AutoFeatureExtractor` is used to process input images for the model, and the `ORTModelForImageClassification` loads the model for inference using ONNX Runtime. The model is then saved locally in the directory `models/vit-base-patch16-224` for future use or deployment. This setup allows efficient image classification using a pre-trained ViT model optimized for ONNX.

```
quantizer = ORTQuantizer.from_pretrained(model)
dqconfig = AutoQuantizationConfig.avx512_vnni(is_static=False, per_channel=False)
dqconfig.weights_dtype = QuantType.QUInt8
model_quantized_path = quantizer.quantize(
    save_dir="models/vit-base-patch16-224-quantized-dynamic",
    quantization_config=dqconfig,
)
```

The code uses `ORTQuantizer` to apply dynamic quantization to a pre-trained Vision Transformer (ViT) model. It sets up quantization with a configuration optimized for Intel AVX-512 with VNNI support, using 8-bit integer precision for weights (QUint8). The quantized model is then saved to a specified directory, making it smaller and faster for inference, especially on compatible hardware. This helps in reducing the model's memory usage and improving its performance in production environments.

```
%ls -lh models/vit-base-patch16-224
%ls -lh models/vit-base-patch16-224-quantized-dynamic
```

It seems you're trying to check the contents of the directories where the models are stored (vit-base-patch16-224 and vit-base-patch16-224-quantized-dynamic). These commands list the files in the directories, showing their sizes, permissions, and other details. Would you like me to explain the expected results of these commands or guide you further with them?

```
def infer_ImageNet(classification_model, processor, image):
```

```

inputs = processor(images=image, return_tensors="pt")
outputs = classification_model(**inputs)
logits = outputs.logits
predicted_class_idx = logits.argmax(-1).item()
return classification_model.config.id2label[predicted_class_idx]

# Dapatkan contoh gambar
url = 'http://images.cocodataset.org/val2017/000000039769.jpg'
image = Image.open(requests.get(url, stream=True).raw)

res = infer_ImageNet(model, preprocessor, image)
print("Original model prediction:", res)

quantized_model =
ORTModelForImageClassification.from_pretrained(model_quantized_path)
dq_res = infer_ImageNet(quantized_model, preprocessor, image)
print("Quantized model prediction:", dq_res)
display(image)

```

memvalidasi model terkuantisasi dengan membandingkan hasil model asli dan model terkuantisasi.

Kita membuat fungsi untuk melakukan inferensi ketika diberikan model, prosesor, dan gambar, lalu mengembalikan hasil klasifikasi berdasarkan label ImageNet

```

quantizer = ORTQuantizer.from_pretrained(model)
static_qconfig = AutoQuantizationConfig.arm64(is_static=True, per_channel=False)

# Membuat dataset kalibrasi
def preprocess_fn(ex, processor):
    return processor(ex["image"])

calibration_dataset = quantizer.get_calibration_dataset(
    "zh-plus/tiny-imagenet",
    preprocess_function=partial(preprocess_fn, processor=preprocessor),
    num_samples=50,
    dataset_split="train",
)

# Membuat konfigurasi kalibrasi yang berisi parameter terkait kalibrasi.
calibration_config = AutoCalibrationConfig.minmax(calibration_dataset)

# Melakukan langkah kalibrasi: menghitung rentang kuantisasi aktivasi
ranges = quantizer.fit(
    dataset=calibration_dataset,
    calibration_config=calibration_config,
)

```

```

        operators_to_quantize=static_qconfig.operators_to_quantize,
    )

# Menerapkan kuantisasi statis pada model
model_quantized_path_static = quantizer.quantize(
    save_dir="models/vit-base-patch16-224-quantized-static",
    calibration_tensors_range=ranges,
    quantization_config=static_qconfig,
)

```

Untuk kuantisasi statis, mirip dengan kuantisasi statis ONNX Runtime, parameter dikuantisasi terlebih dahulu menggunakan set data kalibrasi. Metode ini lebih cepat daripada kuantisasi dinamis tetapi akurasi lebih rendah.

Saat menggunakan Optimum, set data kalibrasi dapat dibuat menggunakan metode ``quantizer.get_calibration_dataset()`` yang mengambil set data apa pun dari HuggingFace Hub atau folder lokal. Setelah set data kalibrasi dibuat dan konfigurasi kalibrasi ditentukan menggunakan ``AutoCalibrationConfig.minmax()``, lakukan kalibrasi dengan memanggil metode ``quantizer.fit()`` lalu kuantisasi model menggunakan metode ``quantizer.quantize()``.

```
%ls -lh models/vit-base-patch16-224-quantized-static
```

Untuk periksa ukuran model terkuantisasi.

```

static_quantized_model =
ORTModelForImageClassification.from_pretrained(model_quantized_path_static)
sq_res = infer_ImageNet(static_quantized_model, preprocessor, image)
print("Quantized model prediction (static):", sq_res)

```

validasi model terkuantisasi dengan membandingkan hasil model asli dan model terkuantisasi.

## 6. ONNX Runtime (Chapter 9)

Dalam buku catatan ini, kami akan menunjukkan cara menggunakan ONNX Runtime untuk mempercepat inferensi model yang dilatih dalam PyTorch. Selain itu, kami akan menggunakan ONNX untuk mengkuantisasi model ke presisi int8 untuk fur## Siapkan ONNX Runtime

Pertama, instal torch, torchvision, onnx, dan onnxruntime. Kemudian, impor modul yang diperlukan untuk meningkatkan kinerja dengan mengurangi jejak memori. Kami akan melatih model sederhana pada dataset MNIST dan kemudian mengonversinya ke format ONNX. Kami kemudian akan menggunakan ONNX Runtime untuk mempercepat inferensi model. Terakhir, kami akan mengkuantisasi model ke presisi int8

```

%pip install torch torchvision
%pip install onnx onnxruntime
import torch
import torch.nn as nn

```

```

import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
import torch.quantization
import pathlib
import numpy as np
import torch.onnx
import onnx
import onnxruntime
from onnxruntime.quantization import quantize_dynamic, quantize_static,
CalibrationDataReader, QuantType

```

Kode ini mengimpor pustaka yang diperlukan untuk membangun, melatih, dan melakukan kuantisasi pada model jaringan saraf menggunakan PyTorch dan ONNX. PyTorch digunakan untuk mendefinisikan dan melatih model jaringan saraf, sementara torchvision digunakan untuk menangani dataset seperti MNIST. Kode ini juga mencakup ONNX dan ONNX Runtime untuk mengekspor model, serta utilitas kuantisasi. Metode `quantize_dynamic` dan `quantize_static` digunakan untuk melakukan kuantisasi model dalam mode dinamis dan statis. Teknik kuantisasi ini penting untuk mengurangi ukuran model dan meningkatkan efisiensi inferensi dengan mengubah bobot dan operasi model ke format presisi lebih rendah, seperti INT8.

```

transform=transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])

train_dataset = datasets.MNIST('./data', train=True,
download=True,transform=transform)
test_dataset = datasets.MNIST('./data', train=False,transform=transform)

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=12, kernel_size=3)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.fc = nn.Linear(12 * 13 * 13, 10)

    def forward(self, x):
        x = x.view(-1, 1, 28, 28)
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        output = F.log_softmax(x, dim=1)
        return output

```

```

train_loader = torch.utils.data.DataLoader(train_dataset, 32)
test_loader = torch.utils.data.DataLoader(test_dataset, 32)

device = "cpu"

epochs = 1

model = Net().to(device)
optimizer = optim.Adam(model.parameters())

model.train()

for epoch in range(1, epochs+1):
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
            epoch, batch_idx * len(data), len(train_loader.dataset),
            100. * batch_idx / len(train_loader), loss.item()))

MODEL_DIR = pathlib.Path("./onnx_models")
MODEL_DIR.mkdir(exist_ok=True)
torch.save(model.state_dict(), MODEL_DIR / "original_model.p")

```

Kode ini melatih model jaringan saraf sederhana menggunakan dataset MNIST dengan PyTorch. Pertama, kode ini mempersiapkan transformasi data untuk mengonversi gambar ke tensor dan menormalisasi nilai piksel. Model yang digunakan adalah jaringan konvolusional dengan satu lapisan konvolusi diikuti lapisan pooling dan lapisan fully connected untuk klasifikasi digit. Model dilatih menggunakan algoritma Adam, dan fungsi kerugian yang digunakan adalah negative log likelihood loss (nll\_loss). Setelah melatih model selama satu epoch, bobot model disimpan ke dalam file dengan nama "original\_model.p" di direktori "onnx\_models".

```

x, _ = next(iter(train_loader))
torch.onnx.export(model,
    x,
    MODEL_DIR / "mnist_model.onnx",
    export_params=True,
    opset_version=10,
    do_constant_folding=True,
    input_names = ['input'],
    output_names = ['output'],

```

```
dynamic_axes={'input' : {0 : 'batch_size'},
               'output' : {0 : 'batch_size'}})
```

Kode ini mengekspor model PyTorch yang telah dilatih ke format ONNX (Open Neural Network Exchange). Pertama, kode mengambil satu batch data dari train\_loader untuk digunakan sebagai input saat mengekspor model. Kemudian, fungsi torch.onnx.export() digunakan untuk mengonversi model PyTorch menjadi model ONNX. Parameter yang digunakan antara lain menyertakan parameter model (export\_params=True), menentukan versi opset ONNX yang digunakan (opset\_version=10), serta mengaktifkan optimasi untuk pemangkasan konstanta (do\_constant\_folding=True). Selain itu, nama input dan output model ditentukan dengan input\_names dan output\_names, sementara dynamic\_axes mengonfigurasi agar ukuran batch input dan output dapat berubah dinamis. Model ONNX yang diekspor disimpan dalam file "mnist\_model.onnx" di dalam direktori MODEL\_DIR.

```
torch_out = model(x)

onnx_model = onnx.load(MODEL_DIR / "mnist_model.onnx")
onnx.checker.check_model(onnx_model)

ort_session = onnxruntime.InferenceSession(MODEL_DIR / "mnist_model.onnx",
providers=["CPUExecutionProvider"])

def to_numpy(tensor):
    return tensor.detach().cpu().numpy() if tensor.requires_grad else
tensor.cpu().numpy()

# menghitung prediksi output ONNX Runtime
ort_inputs = {ort_session.get_inputs()[0].name: to_numpy(x)}
ort_outs = ort_session.run(None, ort_inputs)

# membandingkan hasil ONNX Runtime dan PyTorch
np.testing.assert_allclose(to_numpy(torch_out), ort_outs[0], rtol=1e-03, atol=1e-
05)

print("Model yang diekspor telah diuji dengan ONNXRuntime, dan hasilnya terlihat
baik!")
```

Kode ini digunakan untuk memverifikasi bahwa model yang telah diekspor ke format ONNX menghasilkan hasil yang konsisten dengan model PyTorch. Pertama, model PyTorch digunakan untuk menghasilkan prediksi (torch\_out) dari input x. Kemudian, model ONNX yang telah diekspor dimuat menggunakan onnx.load() dan diperiksa kevalidannya dengan onnx.checker.check\_model(). Selanjutnya, ONNX Runtime (onnxruntime.InferenceSession) digunakan untuk menjalankan inferensi pada model ONNX, menghasilkan output prediksi (ort\_outs). Fungsi to\_numpy() digunakan untuk mengonversi tensor PyTorch menjadi array NumPy. Untuk memastikan kesamaan hasil antara prediksi model PyTorch dan ONNX,

`np.testing.assert_allclose()` digunakan untuk membandingkan kedua hasil dengan toleransi tertentu. Jika perbandingan berhasil, maka output akan mencetak pesan bahwa model yang diekspor telah diuji dengan ONNXRuntime dan hasilnya baik.

```
!python -m onnxruntime.quantization.preprocess --input {MODEL_DIR /  
"mnist_model.onnx"} --output {MODEL_DIR / "mnist_model_processed.onnx"}
```

```
model_fp32 = MODEL_DIR / "mnist_model_processed.onnx"  
model_quant = MODEL_DIR / "mnist_model_quant.onnx"  
quantized_model = quantize_dynamic(model_fp32, model_quant,  
weight_type=QuantType.QUInt8)
```

Kode ini digunakan untuk melakukan kuantisasi pada model ONNX. Pertama, perintah `onnxruntime.quantization.preprocess` digunakan untuk memproses model ONNX yang ada dan menyimpan hasilnya dalam file `mnist_model_processed.onnx`. Setelah itu, model yang telah diproses (`model_fp32`) akan dikuantisasi menggunakan `quantize_dynamic` untuk menghasilkan model kuantisasi (`model_quant`) dengan tipe bobot `QuantType.QUInt8`, yang berarti bobot model akan dikuantisasi menjadi format integer 8-bit. Hasil dari kuantisasi ini adalah model ONNX yang lebih efisien untuk dijalankan pada perangkat dengan sumber daya terbatas, seperti perangkat mobile atau embedded.

```
%ls -lh {MODEL_DIR}
```

bandingkan ukuran model asli, model terkuantisasi

```
def test_onnx(model_name, data_loader):  
    onnx_model = onnx.load(model_name)  
    onnx.checker.check_model(onnx_model)  
    ort_session = onnxruntime.InferenceSession(model_name)  
    test_loss = 0  
    correct = 0  
    for data, target in data_loader:  
        ort_inputs = {ort_session.get_inputs()[0].name: to_numpy(data)}  
        output = ort_session.run(None, ort_inputs)[0]  
        output = torch.from_numpy(output)  
        test_loss += F.nll_loss(output, target, reduction='sum').item() #  
jumlahkan loss batch  
        pred = output.argmax(dim=1, keepdim=True) # ambil indeks dengan  
probabilitas log maksimum  
        correct += pred.eq(target.view_as(pred)).sum().item()  
  
    test_loss /= len(data_loader.dataset)  
  
    return 100. * correct / len(data_loader.dataset)
```

```
acc = test_onnx(MODEL_DIR / "mnist_model.onnx", test_loader)
print(f"Akurasi model asli adalah {acc}%")

qacc = test_onnx(MODEL_DIR / "mnist_model_quant.onnx", test_loader)
print(f"Akurasi model kuantisasi adalah {qacc}%")
```

Kode ini digunakan untuk menguji akurasi model ONNX, baik sebelum maupun setelah proses kuantisasi. Fungsi `test_onnx` menerima nama file model ONNX dan loader data uji (`data_loader`) untuk menghitung akurasi model. Model ONNX dimuat menggunakan `onnx.load` dan diverifikasi dengan `onnx.checker.check_model`. Inferensi dilakukan melalui `onnxruntime.InferenceSession`, di mana data input dikonversi ke format numpy dan diprediksi oleh model ONNX. Prediksi dibandingkan dengan label aktual untuk menghitung jumlah prediksi yang benar. Akurasi model asli dihitung menggunakan file `mnist_model.onnx`, sedangkan akurasi model kuantisasi dihitung menggunakan file `mnist_model_quant.onnx`. Hasil akurasi masing-masing model ditampilkan untuk membandingkan performa model sebelum dan setelah proses kuantisasi.

```
class QuantDR(CalibrationDataReader):
    def __init__(self, torch_data_loader, input_name):
        self.torch_data_loader = torch_data_loader
        self.input_name = input_name
        self.datasize = len(torch_data_loader)
        self.enum_data = iter(torch_data_loader)

    def to_numpy(self, tensor):
        return tensor.detach().cpu().numpy() if tensor.requires_grad else
tensor.cpu().numpy()

    def get_next(self):
        batch = next(self.enum_data, None)
        if batch is not None:
            return {self.input_name: self.to_numpy(batch[0])}
        else:
            return None

    def rewind(self):
        self.enum_data = iter(self.torch_data_loader)

calibration_data = QuantDR(train_loader, ort_session.get_inputs()[0].name)
model__static_quant = MODEL_DIR / "mnist_model_static_quant.onnx"
static_quant_model = quantize_static(model_fp32, model__static_quant,
calibration_data, weight_type=QuantType.QInt8)
```



Kode ini mendefinisikan kelas QuantDR yang merupakan implementasi khusus dari CalibrationDataReader untuk proses kuantisasi statis model ONNX. Kelas ini mengambil torch\_data\_loader (dataloader PyTorch) dan nama input model ONNX sebagai parameter. Kelas ini memiliki fungsi get\_next untuk mengembalikan batch data input yang diformat sebagai numpy array untuk proses kalibrasi, serta rewind untuk mengulang enumerasi data dari awal. Selanjutnya, objek QuantDR dibuat menggunakan train\_loader dan nama input model ONNX untuk menyediakan data kalibrasi. Proses kuantisasi statis dilakukan menggunakan fungsi quantize\_static, dengan memberikan model ONNX asli, model kuantisasi yang dihasilkan, data kalibrasi, dan tipe kuantisasi (QuantType.QInt8). Model hasil kuantisasi statis disimpan dalam file mnist\_model\_static\_quant.onnx, yang memiliki bobot lebih kecil dan lebih efisien untuk inferensi pada perangkat keras tertentu.

```
%ls -lh {MODEL_DIR}
static_qacc = test_onnx(model__static_quant, test_loader)
print(f"Accuracy of the static quantized model is {static_qacc}%")
```

Kode ini mengevaluasi performa model kuantisasi statis setelah proses kalibrasi dan kuantisasi selesai. Pertama, direktori yang menyimpan model ditampilkan dengan ukuran file menggunakan perintah %ls -lh untuk memastikan model tersimpan dengan benar. Selanjutnya, model kuantisasi statis (mnist\_model\_static\_quant.onnx) diuji menggunakan data pengujian dari test\_loader melalui fungsi test\_onnx, dan akurasi model disimpan dalam variabel static\_qacc. Terakhir, akurasi model kuantisasi statis ditampilkan dalam bentuk persentase untuk membandingkan performanya dengan model asli atau kuantisasi dinamis.

## 7. Edge TPU (Chapter 9)

Dalam panduan ini, kita akan menggunakan TensorFlow 2 untuk membangun model klasifikasi gambar, melatihnya menggunakan dataset kucing & anjing, dan mengonversinya menjadi TensorFlow Lite melalui kuantisasi pasca-pelatihan. Model ini dibangun di atas MobileNet V2 yang telah dilatih sebelumnya, di mana awalnya kita melatih ulang hanya lapisan klasifikasi sambil menggunakan kembali lapisan ekstraktor fitur yang sudah dilatih. Selanjutnya, kita melakukan fine-tuning model dengan memperbarui bobot pada lapisan ekstraktor fitur tertentu, yang merupakan pendekatan lebih cepat dibandingkan melatih seluruh model dari awal. Setelah pelatihan selesai, kita menggunakan kuantisasi pasca-pelatihan untuk mengonversi semua parameter menjadi format int8, sehingga mengurangi ukuran model dan meningkatkan kecepatan inferensi. Format int8 ini sangat penting untuk kompatibilitas dengan Edge TPU yang terdapat pada perangkat Coral.

Lihat dokumentasi [coral.ai](https://coral.ai/docs/edgetpu/models-intro/) untuk informasi tambahan tentang cara membuat model yang kompatibel dengan Edge TPU. Penting untuk dicatat bahwa tutorial ini memerlukan TensorFlow 2.3 atau versi yang lebih baru untuk kuantisasi penuh, dan secara khusus mengharapkan model yang dibangun menggunakan Keras. Strategi konversi ini tidak kompatibel dengan model yang diimpor dari grafik beku (frozen graph). Jika Anda menggunakan TensorFlow 1.x, Anda dapat merujuk ke [versi 1.x dari tutorial ini](https://colab.research.google.com/github/google-coral/tutorials/blob/master/retrain\_classification\_ptq\_tf1.ipynb).

```
import tensorflow as tf
import os
import numpy as np
import matplotlib.pyplot as plt
```

Kode ini mengimpor pustaka yang diperlukan untuk pemrosesan dan analisis data menggunakan TensorFlow, manipulasi array numerik, dan visualisasi data. tensorflow digunakan untuk membangun dan melatih model pembelajaran mesin, os untuk operasi sistem file seperti pengelolaan direktori, numpy untuk operasi array numerik, dan matplotlib.pyplot untuk membuat grafik atau visualisasi data. Dengan mengimpor pustaka ini, kode siap untuk berbagai tugas seperti pelatihan model, pengelolaan data, analisis, dan penyajian hasil dalam bentuk visual.

```
_URL = 'https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip'
path_to_zip = tf.keras.utils.get_file('cats_and_dogs.zip', origin=_URL,
extract=True)
PATH = os.path.join(os.path.dirname(path_to_zip), 'cats_and_dogs_filtered')

train_dir = os.path.join(PATH, 'train')
validation_dir = os.path.join(PATH, 'validation')
```

Kode ini mengunduh dataset "Cats and Dogs" dari URL yang disediakan menggunakan tf.keras.utils.get\_file, yang menyimpan file zip dengan nama cats\_and\_dogs.zip dan mengekstraknya secara otomatis. Direktori hasil ekstraksi disimpan dalam variabel PATH, menunjuk ke folder cats\_and\_dogs\_filtered. Selanjutnya, direktori untuk data pelatihan (train\_dir) dan validasi (validation\_dir) dibuat dengan menggabungkan jalur folder utama dengan subfolder train dan validation menggunakan os.path.join. Kode ini mempersiapkan dataset untuk digunakan dalam pelatihan model pembelajaran mesin.

```
IMAGE_SIZE = 224
BATCH_SIZE = 64

train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
val_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(IMAGE_SIZE, IMAGE_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='categorical' # Ubah jika Anda memiliki jenis masalah yang
berbeda (misalnya, binary)
)

val_generator = val_datagen.flow_from_directory(
    validation_dir,
    target_size=(IMAGE_SIZE, IMAGE_SIZE),
    batch_size=BATCH_SIZE,
```



```
base_model.trainable = False
```

Kode ini membuat model dasar menggunakan arsitektur MobileNetV2 yang telah dilatih sebelumnya pada dataset ImageNet. Model ini digunakan sebagai ekstraktor fitur dalam proses transfer learning. Parameter `input_shape=IMG_SHAPE` menentukan ukuran input gambar (224x224 piksel dengan 3 saluran warna untuk RGB). Opsi `include_top=False` menghapus lapisan klasifikasi teratas, sehingga hanya lapisan konvolusi yang digunakan untuk menghasilkan fitur. Dengan `weights='imagenet'`, model memuat bobot yang sudah terlatih pada dataset ImageNet. Properti `base_model.trainable = False` membekukan bobot model, sehingga tidak diperbarui selama pelatihan untuk mempertahankan fitur yang telah dipelajari sebelumnya.

```
model = tf.keras.Sequential([
    base_model,
    tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(units=2, activation='softmax')
])
```

Kode ini membangun model neural network dengan menggunakan `base_model` MobileNetV2 yang sudah dilatih sebelumnya. Model ini terdiri dari beberapa lapisan tambahan untuk penyesuaian tugas klasifikasi gambar. Lapisan `Conv2D` dengan 32 filter dan ukuran kernel 3 digunakan untuk ekstraksi fitur lebih lanjut dari output base model. Lapisan `Dropout(0.2)` diterapkan untuk mencegah overfitting dengan membuang 20% dari neuron selama pelatihan. `GlobalAveragePooling2D()` mereduksi dimensi output dari fitur spasial menjadi nilai rata-rata global untuk tiap fitur. Lapisan `Dense` terakhir dengan 2 unit dan fungsi aktivasi `softmax` digunakan untuk klasifikasi dua kelas (misalnya, kucing dan anjing). Model ini dirancang untuk tugas klasifikasi gambar.

```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.summary()
```

Kode ini mengonfigurasi model yang telah dibangun untuk proses pelatihan. **`model.compile()`** menetapkan optimizer **`adam`**, yang merupakan algoritma optimisasi populer untuk pembelajaran mendalam, serta fungsi **`loss categorical_crossentropy`**, yang digunakan untuk masalah klasifikasi dengan lebih dari dua kelas. Fungsi ini menghitung perbedaan antara prediksi model dan label yang sebenarnya. Selain itu, **`metrics=['accuracy']`** menambahkan akurasi sebagai metrik evaluasi selama pelatihan dan pengujian. Setelah itu, **`model.summary()`** digunakan untuk menampilkan ringkasan struktur model, termasuk jumlah parameter yang dapat dilatih dan tipe lapisan yang digunakan.

```
print('Number of trainable weights = {}'.format(len(model.trainable_weights)))
```

Kode ini mencetak jumlah bobot yang dapat dilatih (trainable weights) dalam model menggunakan `model.trainable_weights`. `model.trainable_weights` adalah atribut yang berisi daftar bobot yang dapat diperbarui selama pelatihan, seperti bobot pada lapisan yang tidak dibekukan. Fungsi `len()` digunakan untuk menghitung jumlah elemen dalam daftar ini, yang menunjukkan berapa banyak parameter yang bisa dilatih oleh model. Kode ini membantu untuk memahami sejauh mana model dapat disesuaikan dan dipelajari selama proses pelatihan.

```
history = model.fit(train_generator,
                    steps_per_epoch=len(train_generator),
                    epochs=10,
                    validation_data=val_generator,
                    validation_steps=len(val_generator))
```

Kode ini melatih model menggunakan data pelatihan `train_generator` dan data validasi `val_generator` selama 10 epoch. Pada setiap epoch, model akan memproses sejumlah langkah yang ditentukan oleh `steps_per_epoch` dan menghitung akurasi serta loss menggunakan `validation_data` dan `validation_steps` untuk evaluasi. Model menggunakan `categorical_crossentropy` sebagai fungsi loss dan `accuracy` sebagai metrik untuk mengukur performa. Proses pelatihan ini menghasilkan nilai akurasi dan loss untuk setiap epoch pada data pelatihan dan validasi.

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()), 1])
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0, 1.0])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```

Kode ini digunakan untuk memvisualisasikan hasil pelatihan model dalam bentuk grafik. Grafik pertama menunjukkan perbandingan antara akurasi pelatihan dan akurasi validasi sepanjang epoch. Grafik kedua menampilkan perbandingan antara loss pelatihan dan loss validasi sepanjang epoch. Kedua grafik ini memberikan gambaran mengenai performa model selama proses pelatihan, termasuk apakah model mengalami overfitting atau underfitting. Dengan menggunakan matplotlib, kode ini memplotkan data yang disimpan dalam objek history untuk akurasi dan loss pada data pelatihan dan validasi, serta menambahkan label, judul, dan legend untuk memperjelas grafik.

```
print("Number of layers in the base model: ", len(base_model.layers))
base_model.trainable = True
fine_tune_at = 100

# Bekukan semua lapisan sebelum lapisan `fine_tune_at`
for layer in base_model.layers[:fine_tune_at]:
    layer.trainable = False
```

Kode ini digunakan untuk melakukan fine-tuning pada model yang telah dilatih sebelumnya, dalam hal ini model MobileNetV2. Pertama, kode mencetak jumlah lapisan (layers) pada model dasar base\_model. Selanjutnya, base\_model.trainable diatur menjadi True, yang memungkinkan lapisan-lapisan model tersebut untuk dilatih lebih lanjut. Kemudian, kode menentukan titik awal fine-tuning pada lapisan ke-100, dengan membekukan (menetapkan trainable = False) semua lapisan sebelum lapisan ke-100. Ini berarti hanya lapisan setelah lapisan ke-100 yang akan dilatih selama proses fine-tuning, sementara lapisan sebelumnya akan tetap beku (tidak diperbarui). Tujuan dari langkah ini adalah untuk mempertahankan fitur yang sudah dipelajari pada lapisan awal, sambil melatih bagian yang lebih dalam dari model untuk menyesuaikan dengan tugas atau dataset spesifik.

```
model.compile(optimizer=tf.keras.optimizers.Adam(1e-5),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.summary()
```

Kode ini digunakan untuk kompilasi model setelah melakukan fine-tuning pada model MobileNetV2. Pertama, optimizer Adam dengan learning rate yang sangat kecil (1e-5) digunakan untuk memperlambat pembaruan bobot saat melatih model agar tidak merusak pengetahuan yang sudah dipelajari sebelumnya. Kemudian, model dikompilasi dengan fungsi loss categorical\_crossentropy, yang umum digunakan untuk masalah klasifikasi multi-kelas, dan metrik yang digunakan adalah accuracy (akurasi). Setelah itu, kode memanggil model.summary() untuk menampilkan ringkasan arsitektur model, termasuk jumlah parameter dan lapisan yang ada di dalamnya.

```
print('Number of trainable weights = {}'.format(len(model.trainable_weights)))
history_fine = model.fit(train_generator,
                        steps_per_epoch=len(train_generator),
                        epochs=5,
                        validation_data=val_generator,
```

```
validation_steps=len(val_generator))
```

Kode ini pertama-tama mencetak jumlah bobot (weights) yang dapat dilatih (trainable weights) dalam model, yaitu jumlah parameter yang dapat diperbarui selama pelatihan. Setelah itu, model dilatih kembali menggunakan fit() dengan data pelatihan (train\_generator) dan data validasi (val\_generator) untuk 5 epoch. Selama pelatihan, steps\_per\_epoch dan validation\_steps diatur sesuai dengan jumlah batch dalam generator pelatihan dan validasi. history\_fine menyimpan riwayat pelatihan yang berisi metrik-metrik seperti akurasi dan loss pada setiap epoch.

```
acc = history_fine.history['accuracy']
val_acc = history_fine.history['val_accuracy']

loss = history_fine.history['loss']
val_loss = history_fine.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0,1.0])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```

Kode ini digunakan untuk memvisualisasikan hasil pelatihan model setelah fine-tuning. Pertama, akurasi dan loss untuk pelatihan dan validasi diambil dari history\_fine. Kemudian, dua subplot dibuat menggunakan matplotlib: satu untuk menampilkan akurasi pelatihan dan validasi dan satu lagi untuk menampilkan loss pelatihan dan validasi. Pada setiap subplot, data diplot untuk menunjukkan bagaimana metrik berubah seiring berjalannya epoch. Akurasi diplot pada subplot pertama, dan loss diplot pada subplot kedua. Grafik ini membantu menganalisis kinerja model selama pelatihan dan validasi.

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()
```

```
with open('mobilenet_v2_1.0_224.tflite', 'wb') as f:
    f.write(tflite_model)
```

Kode ini digunakan untuk mengonversi model Keras yang telah dilatih ke format TensorFlow Lite (TFLite) agar dapat dijalankan pada perangkat dengan sumber daya terbatas, seperti ponsel atau perangkat IoT. Pertama, TFLiteConverter digunakan untuk mengonversi model Keras yang telah dibuat ke format TFLite. Setelah konversi selesai, model TFLite disimpan ke dalam file dengan ekstensi .tflite menggunakan perintah write(), sehingga model dapat digunakan untuk inferensi pada platform yang mendukung TensorFlow Lite.

```
# Sebuah generator yang menyediakan dataset representatif
def representative_data_gen():
    dataset_list = tf.data.Dataset.list_files(PATH + '/train/*/*')
    for i in range(100):
        image = next(iter(dataset_list))
        image = tf.io.read_file(image)
        image = tf.io.decode_jpeg(image, channels=3)
        image = tf.image.resize(image, [IMAGE_SIZE, IMAGE_SIZE])
        image = tf.cast(image / 255., tf.float32)
        image = tf.expand_dims(image, 0)
        yield [image]

converter = tf.lite.TFLiteConverter.from_keras_model(model)
# Ini mengaktifkan kuantisasi
converter.optimizations = [tf.lite.Optimize.DEFAULT]
# Ini menetapkan dataset representatif untuk kuantisasi
converter.representative_dataset = representative_data_gen
# Ini memastikan bahwa jika ada operasi yang tidak bisa dikuantisasi, konverter
akan melemparkan error
converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
# Untuk kuantisasi integer penuh, meskipun tipe yang didukung default-nya hanya
int8, kita secara eksplisit menyatakannya untuk kejelasan.
converter.target_spec.supported_types = [tf.int8]
# Ini menetapkan input dan output tensor ke uint8 (ditambahkan pada r2.3)
converter.inference_input_type = tf.uint8
converter.inference_output_type = tf.uint8
tflite_model = converter.convert()

with open('mobilenet_v2_1.0_224_quant.tflite', 'wb') as f:
    f.write(tflite_model)
```

Kode ini digunakan untuk mengonversi model Keras yang telah dilatih ke format TensorFlow Lite (TFLite) dengan kuantisasi untuk meningkatkan efisiensi model pada perangkat dengan sumber daya terbatas. Fungsi representative\_data\_gen digunakan untuk menyediakan dataset representatif yang akan digunakan dalam proses kuantisasi, mengonversi gambar ke ukuran yang



sesuai dan normalisasi. Setelah itu, TFLiteConverter digunakan untuk mengonversi model Keras menjadi model TFLite, dengan mengaktifkan optimasi kuantisasi (`converter.optimizations`) dan menetapkan dataset representatif untuk kuantisasi (`converter.representative_dataset`). Konversi ini juga memastikan bahwa operasi yang tidak mendukung kuantisasi akan menghasilkan error, dan model menggunakan kuantisasi integer penuh dengan tipe data `int8` untuk input dan output (`converter.inference_input_type` dan `converter.inference_output_type`). Akhirnya, model yang telah dikuantisasi disimpan dalam format `.tflite` untuk digunakan pada perangkat yang mendukung TensorFlow Lite.

```
batch_images, batch_labels = next(val_generator)

logits = model(batch_images)
prediction = np.argmax(logits, axis=1)
truth = np.argmax(batch_labels, axis=1)

keras_accuracy = tf.keras.metrics.Accuracy()
keras_accuracy(prediction, truth)

print("Raw model accuracy: {:.3%}".format(keras_accuracy.result()))
```

Kode ini digunakan untuk menghitung akurasi model menggunakan data dari generator `val_generator`. Pertama, batch gambar dan label diambil dari `val_generator`. Kemudian, model digunakan untuk menghasilkan prediksi (logits) berdasarkan batch gambar. Prediksi tersebut dikonversi ke kelas dengan mengambil nilai `argmax` dari logits. Label sebenarnya (truth) juga dihitung dengan `argmax` dari `batch_labels`. `tf.keras.metrics.Accuracy()` digunakan untuk menghitung akurasi antara prediksi dan label sebenarnya. Hasil akurasi kemudian dicetak dalam format persentase. Kode ini memberikan gambaran seberapa akurat model dalam memprediksi data validasi.

```
def set_input_tensor(interpreter, input):
    input_details = interpreter.get_input_details()[0]
    tensor_index = input_details['index']
    input_tensor = interpreter.tensor(tensor_index())[0]
    # Input untuk model TFLite harus berupa uint8, jadi kita melakukan kuantisasi
    # pada data input kita.
    # CATATAN: Langkah ini hanya diperlukan karena kita menerima data input dari
    # ImageDataGenerator, yang telah menormalkan semua data gambar ke dalam format
    # float [0,1]. Ketika menggunakan
    # input bitmap, data sudah dalam format uint8 [0,255] jadi ini bisa diganti
    # dengan:
    # input_tensor[:, :] = input
    scale, zero_point = input_details['quantization']
    input_tensor[:, :] = np.uint8(input / scale + zero_point)

def classify_image(interpreter, input):
```

```

set_input_tensor(interpreter, input)
interpreter.invoke()
output_details = interpreter.get_output_details()[0]
output = interpreter.get_tensor(output_details['index'])
# Output dari model TFLite adalah uint8, jadi kita melakukan dekuantisasi pada
hasil:
scale, zero_point = output_details['quantization']
output = scale * (output - zero_point)
top_1 = np.argmax(output)
return top_1

interpreter = tf.lite.Interpreter('mobilenet_v2_1.0_224_quant.tflite')
interpreter.allocate_tensors()

# Mengumpulkan semua prediksi inferensi dalam sebuah daftar
batch_prediction = []
batch_truth = np.argmax(batch_labels, axis=1)

for i in range(len(batch_images)):
    prediction = classify_image(interpreter, batch_images[i])
    batch_prediction.append(prediction)

# Membandingkan semua prediksi dengan kebenaran dasar
tflite_accuracy = tf.keras.metrics.Accuracy()
tflite_accuracy(batch_prediction, batch_truth)
print("Akurasi Quant TF Lite: {:.3%}".format(tflite_accuracy.result()))

```

Kode ini digunakan untuk melakukan inferensi menggunakan model TFLite yang telah dikuantisasi dan menghitung akurasi prediksi. Fungsi `set_input_tensor()` digunakan untuk menyiapkan tensor input sesuai format yang dibutuhkan oleh model TFLite, dengan melakukan kuantisasi pada data gambar yang diterima dalam format float [0,1] dan mengonversinya menjadi uint8. Fungsi `classify_image()` melakukan inferensi pada gambar input, mengatur input tensor, menjalankan model TFLite, dan mengonversi hasil output ke format yang dapat dibaca. Setiap prediksi kemudian dihitung dan disimpan dalam daftar `batch_prediction`. Akurasi prediksi dibandingkan dengan label sebenarnya menggunakan `tf.keras.metrics.Accuracy()`, dan hasil akurasi dicetak sebagai persentase.

```

! curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -

! echo "deb https://packages.cloud.google.com/apt coral-edgetpu-stable main" |
sudo tee /etc/apt/sources.list.d/coral-edgetpu.list

! sudo apt-get update

```

```
! sudo apt-get install edgetpu-compiler
```

Kode ini digunakan untuk menginstal Edge TPU Compiler pada sistem Linux. Pertama, kode ini mengunduh dan menambahkan kunci GPG untuk repositori resmi dari Google untuk perangkat Edge TPU. Kemudian, menambahkan repositori coral-edgetpu-stable ke dalam daftar sumber apt pada sistem. Setelah itu, kode melakukan pembaruan pada paket sistem menggunakan sudo apt-get update, dan terakhir menginstal edgetpu-compiler, yang digunakan untuk mengonversi model TensorFlow ke format yang dapat dijalankan pada perangkat Edge TPU.

```
from google.colab import files

files.download('mobilenet_v2_1.0_224_quant_edgetpu.tflite')
files.download('catdogs_labels.txt')
```

Kode ini digunakan untuk mengunduh dua file dari Google Colab ke komputer lokal. Pertama, file mobilenet\_v2\_1.0\_224\_quant\_edgetpu.tflite, yang merupakan model TensorFlow Lite yang telah dikuantisasi untuk perangkat Edge TPU, diunduh. Kedua, file catdogs\_labels.txt, yang berisi label kelas dari dataset, juga diunduh. Fungsi files.download() dari Google Colab memungkinkan pengguna untuk mengunduh file langsung ke perangkat mereka.

## 8. Openvino (Chapter 9)

Dalam buku catatan ini, kami akan menunjukkan cara menggunakan perangkat OpenVINO untuk menerapkan model pembelajaran mendalam pada perangkat edge dan mengkuantisasi model untuk mengurangi ukuran model dan latensi inferensi. Kami akan melatih model CNN sederhana pada dataset MNIST, mengonversinya ke format OpenVINO IR, dan mengkuantisasi model ke presisi INT8. Kami kemudian akan membandingkan ukuran dan kinerja model terkuantisasi dengan model FP32 asli.

Pertama, kita perlu menginstal OpenVINO, NNCF, dan torch

```
%pip install -q "openvino>=2023.1.0" torch torchvision --extra-index-url
https://download.pytorch.org/whl/cpu
%pip install -q "nnf>=2.6.0"
```

---

44.7/44.7 MB 14.0 MB/s eta 0:00:00

etadata (setup.py) ...

---

70.6/70.6 kB 1.9 MB/s eta 0:00:00

---

207.3/207.3 kB 5.4 MB/s eta 0:00:00

etadata (setup.py) ...

---

1.3/1.3 MB 31.4 MB/s eta 0:00:00

---

1.7/1.7 MB 45.8 MB/s eta 0:00:00

---

422.9/422.9 kB 19.7 MB/s eta 0:00:00

---

4.2/4.2 MB 67.8 MB/s eta 0:00:00

---

249.1/249.1 kB 13.4 MB/s eta 0:00:00

---

77.1/77.1 kB 4.8 MB/s eta 0:00:00

---

119.4/119.4 kB 3.4 MB/s eta 0:00:00

e (setup.py) ...

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
import pathlib
import numpy as np
import opencvino as ov
import nncf
```

INFO:nncf:NNCF initialized successfully. Supported frameworks detected: torch, tensorflow, opencvino

## Melatih Model

Selanjutnya, tentukan dan latih model CNN sederhana pada dataset MNIST

```
transform=transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])
```

```
train_dataset = datasets.MNIST('./data', train=True, download=True, transform=transform)
```

```
test_dataset = datasets.MNIST('./data', train=False, transform=transform)
```

```
class Net(nn.Module):
```

```
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=12, kernel_size=3)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.fc = nn.Linear(12 * 13 * 13, 10)
```

```
    def forward(self, x):
        x = x.view(-1, 1, 28, 28)
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        output = F.log_softmax(x, dim=1)
        return output
```

```
train_loader = torch.utils.data.DataLoader(train_dataset, 32)
```

```
test_loader = torch.utils.data.DataLoader(test_dataset, 32)
```

```
device = "cpu"
```

```
epochs = 1
```

```
model = Net().to(device)
```

```
optimizer = optim.Adam(model.parameters())
```

```
model.train()
```

```
for epoch in range(1, epochs+1):
```

```
    for batch_idx, (data, target) in enumerate(train_loader):
```

```
        data, target = data.to(device), target.to(device)
```

```
        optimizer.zero_grad()
```

```
        output = model(data)
```

```
        loss = F.nll_loss(output, target)
```

```
        loss.backward()
```

```
        optimizer.step()
```

```
        print("Train Epoch: {} [{}/{}] ({:.0f}%)\tLoss: {:.6f}'.format(  
            epoch, batch_idx * len(data), len(train_loader.dataset),  
            100. * batch_idx / len(train_loader), loss.item())
```

```
MODEL_DIR = pathlib.Path("./models")
```

```
MODEL_DIR.mkdir(exist_ok=True)
```

```
torch.save(model.state_dict(), MODEL_DIR / "original_model.p")
```

Downloading <http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz>

Failed to download (trying next):

HTTP Error 403: Forbidden

Downloading <https://oss-ci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz>

Downloading <https://oss-ci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz> to  
./data/MNIST/raw/train-images-idx3-ubyte.gz

100% ██████████ 9.91M/9.91M [00:00<00:00, 37.8MB/s]

Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading <http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz>

Failed to download (trying next):

HTTP Error 403: Forbidden

Downloading <https://oss-ci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz>

Downloading <https://oss-ci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz> to  
./data/MNIST/raw/train-labels-idx1-ubyte.gz


100% ██████████ 28.9k/28.9k [00:00<00:00, 1.37MB/s]

Extracting ./data/MNIST/raw/train-labels-idx1-ubyte.gz to ./data/MNIST/raw

Downloading <http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz>

Failed to download (trying next):  
HTTP Error 403: Forbidden


Downloading <https://oss-ci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.gz>  
Downloading <https://oss-ci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.gz> to  
./data/MNIST/raw/t10k-images-idx3-ubyte.gz

100%  1.65M/1.65M [00:00<00:00, 10.8MB/s]

Extracting ./data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading <http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz>  
Failed to download (trying next):  
HTTP Error 403: Forbidden

Downloading <https://oss-ci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz>  
Downloading <https://oss-ci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz> to  
./data/MNIST/raw/t10k-labels-idx1-ubyte.gz

100%  4.54k/4.54k [00:00<00:00, 7.93MB/s]

Extracting ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw

Train Epoch: 1 [0/60000 (0%)] Loss: 2.361235  
Train Epoch: 1 [32/60000 (0%)] Loss: 2.350525  
Train Epoch: 1 [64/60000 (0%)] Loss: 2.055931  
Train Epoch: 1 [96/60000 (0%)] Loss: 1.860162  
Train Epoch: 1 [128/60000 (0%)] Loss: 2.009008  
Train Epoch: 1 [160/60000 (0%)] Loss: 2.025581  
Train Epoch: 1 [192/60000 (0%)] Loss: 1.507131  
Train Epoch: 1 [224/60000 (0%)] Loss: 1.683474  
Train Epoch: 1 [256/60000 (0%)] Loss: 1.674473  
Train Epoch: 1 [288/60000 (0%)] Loss: 1.344969  
Train Epoch: 1 [320/60000 (1%)] Loss: 1.399575  
Train Epoch: 1 [352/60000 (1%)] Loss: 1.257095  
Train Epoch: 1 [384/60000 (1%)] Loss: 1.258982  
Train Epoch: 1 [416/60000 (1%)] Loss: 1.130673  
Train Epoch: 1 [448/60000 (1%)] Loss: 1.063616  
Train Epoch: 1 [480/60000 (1%)] Loss: 1.435441  
Train Epoch: 1 [512/60000 (1%)] Loss: 1.314874  
Train Epoch: 1 [544/60000 (1%)] Loss: 0.928313  
Train Epoch: 1 [576/60000 (1%)] Loss: 1.237929  
Train Epoch: 1 [608/60000 (1%)] Loss: 1.284637  
Train Epoch: 1 [640/60000 (1%)] Loss: 1.209358  
Train Epoch: 1 [672/60000 (1%)] Loss: 0.874475  
Train Epoch: 1 [704/60000 (1%)] Loss: 0.972864  
Train Epoch: 1 [736/60000 (1%)] Loss: 0.840262  
Train Epoch: 1 [768/60000 (1%)] Loss: 0.888242  
Train Epoch: 1 [800/60000 (1%)] Loss: 0.651073  
Train Epoch: 1 [832/60000 (1%)] Loss: 0.837471  
Train Epoch: 1 [864/60000 (1%)] Loss: 0.806188

Train Epoch: 1 [896/60000 (1%)]	Loss: 0.835976
Train Epoch: 1 [928/60000 (2%)]	Loss: 0.839806
Train Epoch: 1 [960/60000 (2%)]	Loss: 0.583093
Train Epoch: 1 [992/60000 (2%)]	Loss: 0.760725
Train Epoch: 1 [1024/60000 (2%)]	Loss: 0.807001
Train Epoch: 1 [1056/60000 (2%)]	Loss: 0.964660
Train Epoch: 1 [1088/60000 (2%)]	Loss: 0.648870
Train Epoch: 1 [1120/60000 (2%)]	Loss: 0.977028
Train Epoch: 1 [1152/60000 (2%)]	Loss: 0.504983
Train Epoch: 1 [1184/60000 (2%)]	Loss: 0.537207
Train Epoch: 1 [1216/60000 (2%)]	Loss: 0.941454
Train Epoch: 1 [1248/60000 (2%)]	Loss: 0.721904
Train Epoch: 1 [1280/60000 (2%)]	Loss: 0.606859
Train Epoch: 1 [1312/60000 (2%)]	Loss: 0.600121
Train Epoch: 1 [1344/60000 (2%)]	Loss: 0.725463
Train Epoch: 1 [1376/60000 (2%)]	Loss: 0.584217
Train Epoch: 1 [1408/60000 (2%)]	Loss: 0.490149
Train Epoch: 1 [1440/60000 (2%)]	Loss: 0.384048
Train Epoch: 1 [1472/60000 (2%)]	Loss: 0.455806
Train Epoch: 1 [1504/60000 (3%)]	Loss: 0.687443
Train Epoch: 1 [1536/60000 (3%)]	Loss: 0.446643
Train Epoch: 1 [1568/60000 (3%)]	Loss: 0.571784
Train Epoch: 1 [1600/60000 (3%)]	Loss: 0.630656
Train Epoch: 1 [1632/60000 (3%)]	Loss: 0.379596
Train Epoch: 1 [1664/60000 (3%)]	Loss: 0.459402
Train Epoch: 1 [1696/60000 (3%)]	Loss: 0.256847
Train Epoch: 1 [1728/60000 (3%)]	Loss: 0.308232
Train Epoch: 1 [1760/60000 (3%)]	Loss: 0.404146
Train Epoch: 1 [1792/60000 (3%)]	Loss: 0.406047
Train Epoch: 1 [1824/60000 (3%)]	Loss: 0.399211
Train Epoch: 1 [1856/60000 (3%)]	Loss: 0.425314
Train Epoch: 1 [1888/60000 (3%)]	Loss: 0.363518
Train Epoch: 1 [1920/60000 (3%)]	Loss: 0.375452
Train Epoch: 1 [1952/60000 (3%)]	Loss: 0.482534
Train Epoch: 1 [1984/60000 (3%)]	Loss: 0.417107
Train Epoch: 1 [2016/60000 (3%)]	Loss: 0.718921
Train Epoch: 1 [2048/60000 (3%)]	Loss: 0.477730
Train Epoch: 1 [2080/60000 (3%)]	Loss: 0.384766
Train Epoch: 1 [2112/60000 (4%)]	Loss: 0.319897
Train Epoch: 1 [2144/60000 (4%)]	Loss: 0.161112
Train Epoch: 1 [2176/60000 (4%)]	Loss: 0.425548
Train Epoch: 1 [2208/60000 (4%)]	Loss: 0.338671
Train Epoch: 1 [2240/60000 (4%)]	Loss: 0.175272
Train Epoch: 1 [2272/60000 (4%)]	Loss: 0.381150
Train Epoch: 1 [2304/60000 (4%)]	Loss: 0.351732
Train Epoch: 1 [2336/60000 (4%)]	Loss: 0.332811
Train Epoch: 1 [2368/60000 (4%)]	Loss: 0.372650
Train Epoch: 1 [2400/60000 (4%)]	Loss: 0.667189
Train Epoch: 1 [2432/60000 (4%)]	Loss: 0.455162
Train Epoch: 1 [2464/60000 (4%)]	Loss: 0.305521
Train Epoch: 1 [2496/60000 (4%)]	Loss: 0.152689

Train Epoch: 1 [2528/60000 (4%)]	Loss: 0.337800
Train Epoch: 1 [2560/60000 (4%)]	Loss: 0.306266
Train Epoch: 1 [2592/60000 (4%)]	Loss: 0.396242
Train Epoch: 1 [2624/60000 (4%)]	Loss: 0.477573
Train Epoch: 1 [2656/60000 (4%)]	Loss: 0.600213
Train Epoch: 1 [2688/60000 (4%)]	Loss: 0.397128
Train Epoch: 1 [2720/60000 (5%)]	Loss: 0.437525
Train Epoch: 1 [2752/60000 (5%)]	Loss: 0.416159
Train Epoch: 1 [2784/60000 (5%)]	Loss: 0.499968
Train Epoch: 1 [2816/60000 (5%)]	Loss: 0.413313
Train Epoch: 1 [2848/60000 (5%)]	Loss: 0.205472
Train Epoch: 1 [2880/60000 (5%)]	Loss: 0.512961
Train Epoch: 1 [2912/60000 (5%)]	Loss: 0.349668
Train Epoch: 1 [2944/60000 (5%)]	Loss: 0.280673
Train Epoch: 1 [2976/60000 (5%)]	Loss: 0.293472
Train Epoch: 1 [3008/60000 (5%)]	Loss: 0.444924
Train Epoch: 1 [3040/60000 (5%)]	Loss: 0.546135
Train Epoch: 1 [3072/60000 (5%)]	Loss: 0.351596
Train Epoch: 1 [3104/60000 (5%)]	Loss: 0.366779
Train Epoch: 1 [3136/60000 (5%)]	Loss: 0.222119
Train Epoch: 1 [3168/60000 (5%)]	Loss: 0.217460
Train Epoch: 1 [3200/60000 (5%)]	Loss: 0.413290
Train Epoch: 1 [3232/60000 (5%)]	Loss: 0.171524
Train Epoch: 1 [3264/60000 (5%)]	Loss: 0.536314
Train Epoch: 1 [3296/60000 (5%)]	Loss: 0.314855
Train Epoch: 1 [3328/60000 (6%)]	Loss: 0.233785
Train Epoch: 1 [3360/60000 (6%)]	Loss: 0.413177
Train Epoch: 1 [3392/60000 (6%)]	Loss: 0.430045
Train Epoch: 1 [3424/60000 (6%)]	Loss: 0.263931
Train Epoch: 1 [3456/60000 (6%)]	Loss: 0.324396
Train Epoch: 1 [3488/60000 (6%)]	Loss: 0.413465
Train Epoch: 1 [3520/60000 (6%)]	Loss: 0.562005
Train Epoch: 1 [3552/60000 (6%)]	Loss: 0.281116
Train Epoch: 1 [3584/60000 (6%)]	Loss: 0.147645
Train Epoch: 1 [3616/60000 (6%)]	Loss: 0.257128
Train Epoch: 1 [3648/60000 (6%)]	Loss: 0.510743
Train Epoch: 1 [3680/60000 (6%)]	Loss: 0.254867
Train Epoch: 1 [3712/60000 (6%)]	Loss: 0.551757
Train Epoch: 1 [3744/60000 (6%)]	Loss: 0.322612
Train Epoch: 1 [3776/60000 (6%)]	Loss: 0.455829
Train Epoch: 1 [3808/60000 (6%)]	Loss: 0.437139
Train Epoch: 1 [3840/60000 (6%)]	Loss: 0.217705
Train Epoch: 1 [3872/60000 (6%)]	Loss: 0.252368
Train Epoch: 1 [3904/60000 (7%)]	Loss: 0.198167
Train Epoch: 1 [3936/60000 (7%)]	Loss: 0.238495
Train Epoch: 1 [3968/60000 (7%)]	Loss: 0.298169
Train Epoch: 1 [4000/60000 (7%)]	Loss: 0.289728
Train Epoch: 1 [4032/60000 (7%)]	Loss: 0.420278
Train Epoch: 1 [4064/60000 (7%)]	Loss: 0.522800
Train Epoch: 1 [4096/60000 (7%)]	Loss: 0.183180
Train Epoch: 1 [4128/60000 (7%)]	Loss: 0.505620



Train Epoch: 1 [4160/60000 (7%)]	Loss: 0.460950
Train Epoch: 1 [4192/60000 (7%)]	Loss: 0.259513
Train Epoch: 1 [4224/60000 (7%)]	Loss: 0.313975
Train Epoch: 1 [4256/60000 (7%)]	Loss: 0.495449
Train Epoch: 1 [4288/60000 (7%)]	Loss: 0.185691
Train Epoch: 1 [4320/60000 (7%)]	Loss: 0.339557
Train Epoch: 1 [4352/60000 (7%)]	Loss: 0.343936
Train Epoch: 1 [4384/60000 (7%)]	Loss: 0.276108
Train Epoch: 1 [4416/60000 (7%)]	Loss: 0.166742
Train Epoch: 1 [4448/60000 (7%)]	Loss: 0.487444
Train Epoch: 1 [4480/60000 (7%)]	Loss: 0.454409
Train Epoch: 1 [4512/60000 (8%)]	Loss: 0.118192
Train Epoch: 1 [4544/60000 (8%)]	Loss: 0.138187
Train Epoch: 1 [4576/60000 (8%)]	Loss: 0.185870
Train Epoch: 1 [4608/60000 (8%)]	Loss: 0.413018
Train Epoch: 1 [4640/60000 (8%)]	Loss: 0.415006
Train Epoch: 1 [4672/60000 (8%)]	Loss: 0.433336
Train Epoch: 1 [4704/60000 (8%)]	Loss: 0.174658
Train Epoch: 1 [4736/60000 (8%)]	Loss: 0.350715
Train Epoch: 1 [4768/60000 (8%)]	Loss: 0.344341
Train Epoch: 1 [4800/60000 (8%)]	Loss: 0.329110
Train Epoch: 1 [4832/60000 (8%)]	Loss: 0.225344
Train Epoch: 1 [4864/60000 (8%)]	Loss: 0.171116
Train Epoch: 1 [4896/60000 (8%)]	Loss: 0.225714
Train Epoch: 1 [4928/60000 (8%)]	Loss: 0.574102
Train Epoch: 1 [4960/60000 (8%)]	Loss: 0.407477
Train Epoch: 1 [4992/60000 (8%)]	Loss: 0.415412
Train Epoch: 1 [5024/60000 (8%)]	Loss: 0.247373
Train Epoch: 1 [5056/60000 (8%)]	Loss: 0.344024
Train Epoch: 1 [5088/60000 (8%)]	Loss: 0.345278
Train Epoch: 1 [5120/60000 (9%)]	Loss: 0.500590
Train Epoch: 1 [5152/60000 (9%)]	Loss: 0.783063
Train Epoch: 1 [5184/60000 (9%)]	Loss: 0.199486
Train Epoch: 1 [5216/60000 (9%)]	Loss: 0.160657
Train Epoch: 1 [5248/60000 (9%)]	Loss: 0.266674
Train Epoch: 1 [5280/60000 (9%)]	Loss: 0.511778
Train Epoch: 1 [5312/60000 (9%)]	Loss: 0.419176
Train Epoch: 1 [5344/60000 (9%)]	Loss: 0.349444
Train Epoch: 1 [5376/60000 (9%)]	Loss: 0.335621
Train Epoch: 1 [5408/60000 (9%)]	Loss: 0.212491
Train Epoch: 1 [5440/60000 (9%)]	Loss: 0.141352
Train Epoch: 1 [5472/60000 (9%)]	Loss: 0.159529
Train Epoch: 1 [5504/60000 (9%)]	Loss: 0.148920
Train Epoch: 1 [5536/60000 (9%)]	Loss: 0.711759
Train Epoch: 1 [5568/60000 (9%)]	Loss: 0.141776
Train Epoch: 1 [5600/60000 (9%)]	Loss: 0.327096
Train Epoch: 1 [5632/60000 (9%)]	Loss: 0.327733
Train Epoch: 1 [5664/60000 (9%)]	Loss: 0.205453
Train Epoch: 1 [5696/60000 (9%)]	Loss: 0.418594
Train Epoch: 1 [5728/60000 (10%)]	Loss: 0.380246
Train Epoch: 1 [5760/60000 (10%)]	Loss: 0.160007

Train Epoch: 1 [5792/60000 (10%)]	Loss: 0.234170
Train Epoch: 1 [5824/60000 (10%)]	Loss: 0.436410
Train Epoch: 1 [5856/60000 (10%)]	Loss: 0.212459
Train Epoch: 1 [5888/60000 (10%)]	Loss: 0.344597
Train Epoch: 1 [5920/60000 (10%)]	Loss: 0.253566
Train Epoch: 1 [5952/60000 (10%)]	Loss: 0.320889
Train Epoch: 1 [5984/60000 (10%)]	Loss: 0.079147
Train Epoch: 1 [6016/60000 (10%)]	Loss: 0.165534
Train Epoch: 1 [6048/60000 (10%)]	Loss: 0.247460
Train Epoch: 1 [6080/60000 (10%)]	Loss: 0.401033
Train Epoch: 1 [6112/60000 (10%)]	Loss: 0.218569
Train Epoch: 1 [6144/60000 (10%)]	Loss: 0.148964
Train Epoch: 1 [6176/60000 (10%)]	Loss: 0.126052
Train Epoch: 1 [6208/60000 (10%)]	Loss: 0.129940
Train Epoch: 1 [6240/60000 (10%)]	Loss: 0.373373
Train Epoch: 1 [6272/60000 (10%)]	Loss: 0.265173
Train Epoch: 1 [6304/60000 (11%)]	Loss: 0.201545
Train Epoch: 1 [6336/60000 (11%)]	Loss: 0.205228
Train Epoch: 1 [6368/60000 (11%)]	Loss: 0.133386
Train Epoch: 1 [6400/60000 (11%)]	Loss: 0.363859
Train Epoch: 1 [6432/60000 (11%)]	Loss: 0.212830
Train Epoch: 1 [6464/60000 (11%)]	Loss: 0.489006
Train Epoch: 1 [6496/60000 (11%)]	Loss: 0.296056
Train Epoch: 1 [6528/60000 (11%)]	Loss: 0.241534
Train Epoch: 1 [6560/60000 (11%)]	Loss: 0.149355
Train Epoch: 1 [6592/60000 (11%)]	Loss: 0.158308
Train Epoch: 1 [6624/60000 (11%)]	Loss: 0.192779
Train Epoch: 1 [6656/60000 (11%)]	Loss: 0.328236
Train Epoch: 1 [6688/60000 (11%)]	Loss: 0.192047
Train Epoch: 1 [6720/60000 (11%)]	Loss: 0.196194
Train Epoch: 1 [6752/60000 (11%)]	Loss: 0.145989
Train Epoch: 1 [6784/60000 (11%)]	Loss: 0.344861
Train Epoch: 1 [6816/60000 (11%)]	Loss: 0.351662
Train Epoch: 1 [6848/60000 (11%)]	Loss: 0.513910
Train Epoch: 1 [6880/60000 (11%)]	Loss: 0.471202
Train Epoch: 1 [6912/60000 (12%)]	Loss: 0.333955
Train Epoch: 1 [6944/60000 (12%)]	Loss: 0.276053
Train Epoch: 1 [6976/60000 (12%)]	Loss: 0.350191
Train Epoch: 1 [7008/60000 (12%)]	Loss: 0.678993
Train Epoch: 1 [7040/60000 (12%)]	Loss: 0.237580
Train Epoch: 1 [7072/60000 (12%)]	Loss: 0.138251
Train Epoch: 1 [7104/60000 (12%)]	Loss: 0.133054
Train Epoch: 1 [7136/60000 (12%)]	Loss: 0.558993
Train Epoch: 1 [7168/60000 (12%)]	Loss: 0.259257
Train Epoch: 1 [7200/60000 (12%)]	Loss: 0.576832
Train Epoch: 1 [7232/60000 (12%)]	Loss: 0.476989
Train Epoch: 1 [7264/60000 (12%)]	Loss: 0.456850
Train Epoch: 1 [7296/60000 (12%)]	Loss: 0.279541
Train Epoch: 1 [7328/60000 (12%)]	Loss: 0.428950
Train Epoch: 1 [7360/60000 (12%)]	Loss: 0.300076
Train Epoch: 1 [7392/60000 (12%)]	Loss: 0.077832

Train Epoch: 1 [7424/60000 (12%)]	Loss: 0.205784
Train Epoch: 1 [7456/60000 (12%)]	Loss: 0.164385
Train Epoch: 1 [7488/60000 (12%)]	Loss: 0.225342
Train Epoch: 1 [7520/60000 (13%)]	Loss: 0.340729
Train Epoch: 1 [7552/60000 (13%)]	Loss: 0.071517
Train Epoch: 1 [7584/60000 (13%)]	Loss: 0.441004
Train Epoch: 1 [7616/60000 (13%)]	Loss: 0.436266
Train Epoch: 1 [7648/60000 (13%)]	Loss: 0.252211
Train Epoch: 1 [7680/60000 (13%)]	Loss: 0.110888
Train Epoch: 1 [7712/60000 (13%)]	Loss: 0.282418
Train Epoch: 1 [7744/60000 (13%)]	Loss: 0.260526
Train Epoch: 1 [7776/60000 (13%)]	Loss: 0.251608
Train Epoch: 1 [7808/60000 (13%)]	Loss: 0.318517
Train Epoch: 1 [7840/60000 (13%)]	Loss: 0.505067
Train Epoch: 1 [7872/60000 (13%)]	Loss: 0.618598
Train Epoch: 1 [7904/60000 (13%)]	Loss: 0.289762
Train Epoch: 1 [7936/60000 (13%)]	Loss: 0.269888
Train Epoch: 1 [7968/60000 (13%)]	Loss: 0.300519
Train Epoch: 1 [8000/60000 (13%)]	Loss: 0.266738
Train Epoch: 1 [8032/60000 (13%)]	Loss: 0.103113
Train Epoch: 1 [8064/60000 (13%)]	Loss: 0.212246
Train Epoch: 1 [8096/60000 (13%)]	Loss: 0.799896
Train Epoch: 1 [8128/60000 (14%)]	Loss: 0.303924
Train Epoch: 1 [8160/60000 (14%)]	Loss: 0.184674
Train Epoch: 1 [8192/60000 (14%)]	Loss: 0.752690
Train Epoch: 1 [8224/60000 (14%)]	Loss: 0.418260
Train Epoch: 1 [8256/60000 (14%)]	Loss: 0.259160
Train Epoch: 1 [8288/60000 (14%)]	Loss: 0.394972
Train Epoch: 1 [8320/60000 (14%)]	Loss: 0.236309
Train Epoch: 1 [8352/60000 (14%)]	Loss: 0.111712
Train Epoch: 1 [8384/60000 (14%)]	Loss: 0.159742
Train Epoch: 1 [8416/60000 (14%)]	Loss: 0.381962
Train Epoch: 1 [8448/60000 (14%)]	Loss: 0.496466
Train Epoch: 1 [8480/60000 (14%)]	Loss: 0.404139
Train Epoch: 1 [8512/60000 (14%)]	Loss: 0.170115
Train Epoch: 1 [8544/60000 (14%)]	Loss: 0.357743
Train Epoch: 1 [8576/60000 (14%)]	Loss: 0.174380
Train Epoch: 1 [8608/60000 (14%)]	Loss: 0.431004
Train Epoch: 1 [8640/60000 (14%)]	Loss: 0.343361
Train Epoch: 1 [8672/60000 (14%)]	Loss: 0.772748
Train Epoch: 1 [8704/60000 (15%)]	Loss: 1.279581
Train Epoch: 1 [8736/60000 (15%)]	Loss: 0.266805
Train Epoch: 1 [8768/60000 (15%)]	Loss: 0.513104
Train Epoch: 1 [8800/60000 (15%)]	Loss: 0.293814
Train Epoch: 1 [8832/60000 (15%)]	Loss: 0.450886
Train Epoch: 1 [8864/60000 (15%)]	Loss: 0.451393
Train Epoch: 1 [8896/60000 (15%)]	Loss: 0.687786
Train Epoch: 1 [8928/60000 (15%)]	Loss: 0.256171
Train Epoch: 1 [8960/60000 (15%)]	Loss: 0.426841
Train Epoch: 1 [8992/60000 (15%)]	Loss: 0.422337
Train Epoch: 1 [9024/60000 (15%)]	Loss: 0.163284

Train Epoch: 1 [9056/60000 (15%)]	Loss: 0.356490
Train Epoch: 1 [9088/60000 (15%)]	Loss: 0.708086
Train Epoch: 1 [9120/60000 (15%)]	Loss: 0.320155
Train Epoch: 1 [9152/60000 (15%)]	Loss: 0.296848
Train Epoch: 1 [9184/60000 (15%)]	Loss: 0.121160
Train Epoch: 1 [9216/60000 (15%)]	Loss: 0.374481
Train Epoch: 1 [9248/60000 (15%)]	Loss: 0.359111
Train Epoch: 1 [9280/60000 (15%)]	Loss: 0.280189
Train Epoch: 1 [9312/60000 (16%)]	Loss: 0.325524
Train Epoch: 1 [9344/60000 (16%)]	Loss: 0.277184
Train Epoch: 1 [9376/60000 (16%)]	Loss: 0.485398
Train Epoch: 1 [9408/60000 (16%)]	Loss: 0.730220
Train Epoch: 1 [9440/60000 (16%)]	Loss: 0.290861
Train Epoch: 1 [9472/60000 (16%)]	Loss: 0.398497
Train Epoch: 1 [9504/60000 (16%)]	Loss: 0.245792
Train Epoch: 1 [9536/60000 (16%)]	Loss: 0.477833
Train Epoch: 1 [9568/60000 (16%)]	Loss: 0.378651
Train Epoch: 1 [9600/60000 (16%)]	Loss: 0.243460
Train Epoch: 1 [9632/60000 (16%)]	Loss: 0.354679
Train Epoch: 1 [9664/60000 (16%)]	Loss: 0.206117
Train Epoch: 1 [9696/60000 (16%)]	Loss: 0.071703
Train Epoch: 1 [9728/60000 (16%)]	Loss: 0.310517
Train Epoch: 1 [9760/60000 (16%)]	Loss: 0.582576
Train Epoch: 1 [9792/60000 (16%)]	Loss: 0.177693
Train Epoch: 1 [9824/60000 (16%)]	Loss: 0.077080
Train Epoch: 1 [9856/60000 (16%)]	Loss: 0.470587
Train Epoch: 1 [9888/60000 (16%)]	Loss: 0.070000
Train Epoch: 1 [9920/60000 (17%)]	Loss: 0.178085
Train Epoch: 1 [9952/60000 (17%)]	Loss: 0.321627
Train Epoch: 1 [9984/60000 (17%)]	Loss: 0.079717
Train Epoch: 1 [10016/60000 (17%)]	Loss: 0.389234
Train Epoch: 1 [10048/60000 (17%)]	Loss: 0.140721
Train Epoch: 1 [10080/60000 (17%)]	Loss: 0.167019
Train Epoch: 1 [10112/60000 (17%)]	Loss: 0.237098
Train Epoch: 1 [10144/60000 (17%)]	Loss: 0.182001
Train Epoch: 1 [10176/60000 (17%)]	Loss: 0.376732
Train Epoch: 1 [10208/60000 (17%)]	Loss: 0.293672
Train Epoch: 1 [10240/60000 (17%)]	Loss: 0.320165
Train Epoch: 1 [10272/60000 (17%)]	Loss: 0.320592
Train Epoch: 1 [10304/60000 (17%)]	Loss: 0.169560
Train Epoch: 1 [10336/60000 (17%)]	Loss: 0.178377
Train Epoch: 1 [10368/60000 (17%)]	Loss: 0.097576
Train Epoch: 1 [10400/60000 (17%)]	Loss: 0.195941
Train Epoch: 1 [10432/60000 (17%)]	Loss: 0.401728
Train Epoch: 1 [10464/60000 (17%)]	Loss: 0.262612
Train Epoch: 1 [10496/60000 (17%)]	Loss: 0.328914
Train Epoch: 1 [10528/60000 (18%)]	Loss: 0.031713
Train Epoch: 1 [10560/60000 (18%)]	Loss: 0.164654
Train Epoch: 1 [10592/60000 (18%)]	Loss: 0.135716
Train Epoch: 1 [10624/60000 (18%)]	Loss: 0.273164
Train Epoch: 1 [10656/60000 (18%)]	Loss: 0.170882

Train Epoch: 1 [10688/60000 (18%)]	Loss: 0.107075
Train Epoch: 1 [10720/60000 (18%)]	Loss: 0.433147
Train Epoch: 1 [10752/60000 (18%)]	Loss: 0.299199
Train Epoch: 1 [10784/60000 (18%)]	Loss: 0.164935
Train Epoch: 1 [10816/60000 (18%)]	Loss: 0.150959
Train Epoch: 1 [10848/60000 (18%)]	Loss: 0.354578
Train Epoch: 1 [10880/60000 (18%)]	Loss: 0.268454
Train Epoch: 1 [10912/60000 (18%)]	Loss: 0.203412
Train Epoch: 1 [10944/60000 (18%)]	Loss: 0.112397
Train Epoch: 1 [10976/60000 (18%)]	Loss: 0.461311
Train Epoch: 1 [11008/60000 (18%)]	Loss: 0.421561
Train Epoch: 1 [11040/60000 (18%)]	Loss: 0.064435
Train Epoch: 1 [11072/60000 (18%)]	Loss: 0.132820
Train Epoch: 1 [11104/60000 (19%)]	Loss: 0.242598
Train Epoch: 1 [11136/60000 (19%)]	Loss: 0.143071
Train Epoch: 1 [11168/60000 (19%)]	Loss: 0.150305
Train Epoch: 1 [11200/60000 (19%)]	Loss: 0.490214
Train Epoch: 1 [11232/60000 (19%)]	Loss: 0.253884
Train Epoch: 1 [11264/60000 (19%)]	Loss: 0.161356
Train Epoch: 1 [11296/60000 (19%)]	Loss: 0.100438
Train Epoch: 1 [11328/60000 (19%)]	Loss: 0.091880
Train Epoch: 1 [11360/60000 (19%)]	Loss: 0.354797
Train Epoch: 1 [11392/60000 (19%)]	Loss: 0.118765
Train Epoch: 1 [11424/60000 (19%)]	Loss: 0.290708
Train Epoch: 1 [11456/60000 (19%)]	Loss: 0.205115
Train Epoch: 1 [11488/60000 (19%)]	Loss: 0.309934
Train Epoch: 1 [11520/60000 (19%)]	Loss: 0.380978
Train Epoch: 1 [11552/60000 (19%)]	Loss: 0.649131
Train Epoch: 1 [11584/60000 (19%)]	Loss: 0.349180
Train Epoch: 1 [11616/60000 (19%)]	Loss: 0.504002
Train Epoch: 1 [11648/60000 (19%)]	Loss: 0.255471
Train Epoch: 1 [11680/60000 (19%)]	Loss: 0.769219
Train Epoch: 1 [11712/60000 (20%)]	Loss: 0.268758
Train Epoch: 1 [11744/60000 (20%)]	Loss: 0.312037
Train Epoch: 1 [11776/60000 (20%)]	Loss: 0.133957
Train Epoch: 1 [11808/60000 (20%)]	Loss: 0.146260
Train Epoch: 1 [11840/60000 (20%)]	Loss: 0.449131
Train Epoch: 1 [11872/60000 (20%)]	Loss: 0.239916
Train Epoch: 1 [11904/60000 (20%)]	Loss: 0.124950
Train Epoch: 1 [11936/60000 (20%)]	Loss: 0.245162
Train Epoch: 1 [11968/60000 (20%)]	Loss: 0.237751
Train Epoch: 1 [12000/60000 (20%)]	Loss: 0.141733
Train Epoch: 1 [12032/60000 (20%)]	Loss: 0.347512
Train Epoch: 1 [12064/60000 (20%)]	Loss: 0.336725
Train Epoch: 1 [12096/60000 (20%)]	Loss: 0.239358
Train Epoch: 1 [12128/60000 (20%)]	Loss: 0.075042
Train Epoch: 1 [12160/60000 (20%)]	Loss: 0.149843
Train Epoch: 1 [12192/60000 (20%)]	Loss: 0.462646
Train Epoch: 1 [12224/60000 (20%)]	Loss: 0.208184
Train Epoch: 1 [12256/60000 (20%)]	Loss: 0.422701
Train Epoch: 1 [12288/60000 (20%)]	Loss: 0.386480

Train Epoch: 1 [12320/60000 (21%)]	Loss: 0.200802
Train Epoch: 1 [12352/60000 (21%)]	Loss: 0.413293
Train Epoch: 1 [12384/60000 (21%)]	Loss: 0.133680
Train Epoch: 1 [12416/60000 (21%)]	Loss: 0.264165
Train Epoch: 1 [12448/60000 (21%)]	Loss: 0.293879
Train Epoch: 1 [12480/60000 (21%)]	Loss: 0.459864
Train Epoch: 1 [12512/60000 (21%)]	Loss: 0.188013
Train Epoch: 1 [12544/60000 (21%)]	Loss: 0.459734
Train Epoch: 1 [12576/60000 (21%)]	Loss: 0.670287
Train Epoch: 1 [12608/60000 (21%)]	Loss: 0.508337
Train Epoch: 1 [12640/60000 (21%)]	Loss: 0.576844
Train Epoch: 1 [12672/60000 (21%)]	Loss: 0.629397
Train Epoch: 1 [12704/60000 (21%)]	Loss: 0.140566
Train Epoch: 1 [12736/60000 (21%)]	Loss: 0.428469
Train Epoch: 1 [12768/60000 (21%)]	Loss: 0.334559
Train Epoch: 1 [12800/60000 (21%)]	Loss: 0.160120
Train Epoch: 1 [12832/60000 (21%)]	Loss: 0.230920
Train Epoch: 1 [12864/60000 (21%)]	Loss: 0.120150
Train Epoch: 1 [12896/60000 (21%)]	Loss: 0.221218
Train Epoch: 1 [12928/60000 (22%)]	Loss: 0.483194
Train Epoch: 1 [12960/60000 (22%)]	Loss: 0.299870
Train Epoch: 1 [12992/60000 (22%)]	Loss: 0.336192
Train Epoch: 1 [13024/60000 (22%)]	Loss: 0.466300
Train Epoch: 1 [13056/60000 (22%)]	Loss: 0.428001
Train Epoch: 1 [13088/60000 (22%)]	Loss: 0.159765
Train Epoch: 1 [13120/60000 (22%)]	Loss: 0.680687
Train Epoch: 1 [13152/60000 (22%)]	Loss: 0.080111
Train Epoch: 1 [13184/60000 (22%)]	Loss: 0.404088
Train Epoch: 1 [13216/60000 (22%)]	Loss: 0.044132
Train Epoch: 1 [13248/60000 (22%)]	Loss: 0.305705
Train Epoch: 1 [13280/60000 (22%)]	Loss: 0.132172
Train Epoch: 1 [13312/60000 (22%)]	Loss: 0.306234
Train Epoch: 1 [13344/60000 (22%)]	Loss: 0.231581
Train Epoch: 1 [13376/60000 (22%)]	Loss: 0.245857
Train Epoch: 1 [13408/60000 (22%)]	Loss: 0.114025
Train Epoch: 1 [13440/60000 (22%)]	Loss: 0.307131
Train Epoch: 1 [13472/60000 (22%)]	Loss: 0.115443
Train Epoch: 1 [13504/60000 (23%)]	Loss: 0.424187
Train Epoch: 1 [13536/60000 (23%)]	Loss: 0.114249
Train Epoch: 1 [13568/60000 (23%)]	Loss: 0.060136
Train Epoch: 1 [13600/60000 (23%)]	Loss: 0.075054
Train Epoch: 1 [13632/60000 (23%)]	Loss: 0.363503
Train Epoch: 1 [13664/60000 (23%)]	Loss: 0.331688
Train Epoch: 1 [13696/60000 (23%)]	Loss: 0.360006
Train Epoch: 1 [13728/60000 (23%)]	Loss: 0.466098
Train Epoch: 1 [13760/60000 (23%)]	Loss: 0.052720
Train Epoch: 1 [13792/60000 (23%)]	Loss: 0.087149
Train Epoch: 1 [13824/60000 (23%)]	Loss: 0.395972
Train Epoch: 1 [13856/60000 (23%)]	Loss: 0.157798
Train Epoch: 1 [13888/60000 (23%)]	Loss: 0.155576
Train Epoch: 1 [13920/60000 (23%)]	Loss: 0.214131

Train Epoch: 1 [13952/60000 (23%)]	Loss: 0.706569
Train Epoch: 1 [13984/60000 (23%)]	Loss: 0.293493
Train Epoch: 1 [14016/60000 (23%)]	Loss: 0.194185
Train Epoch: 1 [14048/60000 (23%)]	Loss: 0.396727
Train Epoch: 1 [14080/60000 (23%)]	Loss: 0.268848
Train Epoch: 1 [14112/60000 (24%)]	Loss: 0.376827
Train Epoch: 1 [14144/60000 (24%)]	Loss: 0.236186
Train Epoch: 1 [14176/60000 (24%)]	Loss: 0.212247
Train Epoch: 1 [14208/60000 (24%)]	Loss: 0.314091
Train Epoch: 1 [14240/60000 (24%)]	Loss: 0.213970
Train Epoch: 1 [14272/60000 (24%)]	Loss: 0.360393
Train Epoch: 1 [14304/60000 (24%)]	Loss: 0.433466
Train Epoch: 1 [14336/60000 (24%)]	Loss: 0.543242
Train Epoch: 1 [14368/60000 (24%)]	Loss: 0.473524
Train Epoch: 1 [14400/60000 (24%)]	Loss: 0.241399
Train Epoch: 1 [14432/60000 (24%)]	Loss: 0.173894
Train Epoch: 1 [14464/60000 (24%)]	Loss: 0.158725
Train Epoch: 1 [14496/60000 (24%)]	Loss: 0.199110
Train Epoch: 1 [14528/60000 (24%)]	Loss: 0.303048
Train Epoch: 1 [14560/60000 (24%)]	Loss: 0.283370
Train Epoch: 1 [14592/60000 (24%)]	Loss: 0.161919
Train Epoch: 1 [14624/60000 (24%)]	Loss: 0.319682
Train Epoch: 1 [14656/60000 (24%)]	Loss: 0.208707
Train Epoch: 1 [14688/60000 (24%)]	Loss: 0.453488
Train Epoch: 1 [14720/60000 (25%)]	Loss: 0.330323
Train Epoch: 1 [14752/60000 (25%)]	Loss: 0.469176
Train Epoch: 1 [14784/60000 (25%)]	Loss: 0.409799
Train Epoch: 1 [14816/60000 (25%)]	Loss: 0.326393
Train Epoch: 1 [14848/60000 (25%)]	Loss: 0.333444
Train Epoch: 1 [14880/60000 (25%)]	Loss: 0.158307
Train Epoch: 1 [14912/60000 (25%)]	Loss: 0.102794
Train Epoch: 1 [14944/60000 (25%)]	Loss: 0.203758
Train Epoch: 1 [14976/60000 (25%)]	Loss: 0.221711
Train Epoch: 1 [15008/60000 (25%)]	Loss: 0.240057
Train Epoch: 1 [15040/60000 (25%)]	Loss: 0.207177
Train Epoch: 1 [15072/60000 (25%)]	Loss: 0.136044
Train Epoch: 1 [15104/60000 (25%)]	Loss: 0.544462
Train Epoch: 1 [15136/60000 (25%)]	Loss: 0.275572
Train Epoch: 1 [15168/60000 (25%)]	Loss: 0.245099
Train Epoch: 1 [15200/60000 (25%)]	Loss: 0.238777
Train Epoch: 1 [15232/60000 (25%)]	Loss: 0.221766
Train Epoch: 1 [15264/60000 (25%)]	Loss: 0.171187
Train Epoch: 1 [15296/60000 (25%)]	Loss: 0.070149
Train Epoch: 1 [15328/60000 (26%)]	Loss: 0.232724
Train Epoch: 1 [15360/60000 (26%)]	Loss: 0.197420
Train Epoch: 1 [15392/60000 (26%)]	Loss: 0.163638
Train Epoch: 1 [15424/60000 (26%)]	Loss: 0.316620
Train Epoch: 1 [15456/60000 (26%)]	Loss: 0.174740
Train Epoch: 1 [15488/60000 (26%)]	Loss: 0.291124
Train Epoch: 1 [15520/60000 (26%)]	Loss: 0.202344
Train Epoch: 1 [15552/60000 (26%)]	Loss: 0.230123

Train Epoch: 1 [15584/60000 (26%)]	Loss: 0.206526
Train Epoch: 1 [15616/60000 (26%)]	Loss: 0.127834
Train Epoch: 1 [15648/60000 (26%)]	Loss: 0.148548
Train Epoch: 1 [15680/60000 (26%)]	Loss: 0.051221
Train Epoch: 1 [15712/60000 (26%)]	Loss: 0.320353
Train Epoch: 1 [15744/60000 (26%)]	Loss: 0.273563
Train Epoch: 1 [15776/60000 (26%)]	Loss: 0.295953
Train Epoch: 1 [15808/60000 (26%)]	Loss: 0.239413
Train Epoch: 1 [15840/60000 (26%)]	Loss: 0.335144
Train Epoch: 1 [15872/60000 (26%)]	Loss: 0.259306
Train Epoch: 1 [15904/60000 (27%)]	Loss: 0.170338
Train Epoch: 1 [15936/60000 (27%)]	Loss: 0.669196
Train Epoch: 1 [15968/60000 (27%)]	Loss: 0.397720
Train Epoch: 1 [16000/60000 (27%)]	Loss: 0.268118
Train Epoch: 1 [16032/60000 (27%)]	Loss: 0.516794
Train Epoch: 1 [16064/60000 (27%)]	Loss: 0.105116
Train Epoch: 1 [16096/60000 (27%)]	Loss: 0.123624
Train Epoch: 1 [16128/60000 (27%)]	Loss: 0.132451
Train Epoch: 1 [16160/60000 (27%)]	Loss: 0.101989
Train Epoch: 1 [16192/60000 (27%)]	Loss: 0.316335
Train Epoch: 1 [16224/60000 (27%)]	Loss: 0.049312
Train Epoch: 1 [16256/60000 (27%)]	Loss: 0.213901
Train Epoch: 1 [16288/60000 (27%)]	Loss: 0.183777
Train Epoch: 1 [16320/60000 (27%)]	Loss: 0.089868
Train Epoch: 1 [16352/60000 (27%)]	Loss: 0.427298
Train Epoch: 1 [16384/60000 (27%)]	Loss: 0.061976
Train Epoch: 1 [16416/60000 (27%)]	Loss: 0.266449
Train Epoch: 1 [16448/60000 (27%)]	Loss: 0.127755
Train Epoch: 1 [16480/60000 (27%)]	Loss: 0.294016
Train Epoch: 1 [16512/60000 (28%)]	Loss: 0.115967
Train Epoch: 1 [16544/60000 (28%)]	Loss: 0.535017
Train Epoch: 1 [16576/60000 (28%)]	Loss: 0.092738
Train Epoch: 1 [16608/60000 (28%)]	Loss: 0.208005
Train Epoch: 1 [16640/60000 (28%)]	Loss: 0.281274
Train Epoch: 1 [16672/60000 (28%)]	Loss: 0.445658
Train Epoch: 1 [16704/60000 (28%)]	Loss: 0.193962
Train Epoch: 1 [16736/60000 (28%)]	Loss: 0.353612
Train Epoch: 1 [16768/60000 (28%)]	Loss: 0.199829
Train Epoch: 1 [16800/60000 (28%)]	Loss: 0.103219
Train Epoch: 1 [16832/60000 (28%)]	Loss: 0.118840
Train Epoch: 1 [16864/60000 (28%)]	Loss: 0.225506
Train Epoch: 1 [16896/60000 (28%)]	Loss: 0.097970
Train Epoch: 1 [16928/60000 (28%)]	Loss: 0.149492
Train Epoch: 1 [16960/60000 (28%)]	Loss: 0.253780
Train Epoch: 1 [16992/60000 (28%)]	Loss: 0.265180
Train Epoch: 1 [17024/60000 (28%)]	Loss: 0.290173
Train Epoch: 1 [17056/60000 (28%)]	Loss: 0.133973
Train Epoch: 1 [17088/60000 (28%)]	Loss: 0.235192
Train Epoch: 1 [17120/60000 (29%)]	Loss: 0.122840
Train Epoch: 1 [17152/60000 (29%)]	Loss: 0.155024
Train Epoch: 1 [17184/60000 (29%)]	Loss: 0.208815



Train Epoch: 1 [17216/60000 (29%)]	Loss: 0.514645
Train Epoch: 1 [17248/60000 (29%)]	Loss: 0.065131
Train Epoch: 1 [17280/60000 (29%)]	Loss: 0.153330
Train Epoch: 1 [17312/60000 (29%)]	Loss: 0.044988
Train Epoch: 1 [17344/60000 (29%)]	Loss: 0.125941
Train Epoch: 1 [17376/60000 (29%)]	Loss: 0.229884
Train Epoch: 1 [17408/60000 (29%)]	Loss: 0.113854
Train Epoch: 1 [17440/60000 (29%)]	Loss: 0.183088
Train Epoch: 1 [17472/60000 (29%)]	Loss: 0.442838
Train Epoch: 1 [17504/60000 (29%)]	Loss: 0.104738
Train Epoch: 1 [17536/60000 (29%)]	Loss: 0.412251
Train Epoch: 1 [17568/60000 (29%)]	Loss: 0.258748
Train Epoch: 1 [17600/60000 (29%)]	Loss: 0.071174
Train Epoch: 1 [17632/60000 (29%)]	Loss: 0.243583
Train Epoch: 1 [17664/60000 (29%)]	Loss: 0.064179
Train Epoch: 1 [17696/60000 (29%)]	Loss: 0.335438
Train Epoch: 1 [17728/60000 (30%)]	Loss: 0.118049
Train Epoch: 1 [17760/60000 (30%)]	Loss: 0.239534
Train Epoch: 1 [17792/60000 (30%)]	Loss: 0.417219
Train Epoch: 1 [17824/60000 (30%)]	Loss: 0.142182
Train Epoch: 1 [17856/60000 (30%)]	Loss: 0.109118
Train Epoch: 1 [17888/60000 (30%)]	Loss: 0.203234
Train Epoch: 1 [17920/60000 (30%)]	Loss: 0.147876
Train Epoch: 1 [17952/60000 (30%)]	Loss: 0.261837
Train Epoch: 1 [17984/60000 (30%)]	Loss: 0.313082
Train Epoch: 1 [18016/60000 (30%)]	Loss: 0.133950
Train Epoch: 1 [18048/60000 (30%)]	Loss: 0.143244
Train Epoch: 1 [18080/60000 (30%)]	Loss: 0.261968
Train Epoch: 1 [18112/60000 (30%)]	Loss: 0.082158
Train Epoch: 1 [18144/60000 (30%)]	Loss: 0.172391
Train Epoch: 1 [18176/60000 (30%)]	Loss: 0.055494
Train Epoch: 1 [18208/60000 (30%)]	Loss: 0.178568
Train Epoch: 1 [18240/60000 (30%)]	Loss: 0.197512
Train Epoch: 1 [18272/60000 (30%)]	Loss: 0.129430
Train Epoch: 1 [18304/60000 (31%)]	Loss: 0.184105
Train Epoch: 1 [18336/60000 (31%)]	Loss: 0.220849
Train Epoch: 1 [18368/60000 (31%)]	Loss: 0.067937
Train Epoch: 1 [18400/60000 (31%)]	Loss: 0.165651
Train Epoch: 1 [18432/60000 (31%)]	Loss: 0.137686
Train Epoch: 1 [18464/60000 (31%)]	Loss: 0.259933
Train Epoch: 1 [18496/60000 (31%)]	Loss: 0.161412
Train Epoch: 1 [18528/60000 (31%)]	Loss: 0.089471
Train Epoch: 1 [18560/60000 (31%)]	Loss: 0.077903
Train Epoch: 1 [18592/60000 (31%)]	Loss: 0.432112
Train Epoch: 1 [18624/60000 (31%)]	Loss: 0.126236
Train Epoch: 1 [18656/60000 (31%)]	Loss: 0.096798
Train Epoch: 1 [18688/60000 (31%)]	Loss: 0.134512
Train Epoch: 1 [18720/60000 (31%)]	Loss: 0.275097
Train Epoch: 1 [18752/60000 (31%)]	Loss: 0.092512
Train Epoch: 1 [18784/60000 (31%)]	Loss: 0.098188
Train Epoch: 1 [18816/60000 (31%)]	Loss: 0.292608

Train Epoch: 1 [18848/60000 (31%)]	Loss: 0.062378
Train Epoch: 1 [18880/60000 (31%)]	Loss: 0.117146
Train Epoch: 1 [18912/60000 (32%)]	Loss: 0.054003
Train Epoch: 1 [18944/60000 (32%)]	Loss: 0.072346
Train Epoch: 1 [18976/60000 (32%)]	Loss: 0.166269
Train Epoch: 1 [19008/60000 (32%)]	Loss: 0.345410
Train Epoch: 1 [19040/60000 (32%)]	Loss: 0.188867
Train Epoch: 1 [19072/60000 (32%)]	Loss: 0.234531
Train Epoch: 1 [19104/60000 (32%)]	Loss: 0.238146
Train Epoch: 1 [19136/60000 (32%)]	Loss: 0.199796
Train Epoch: 1 [19168/60000 (32%)]	Loss: 0.167362
Train Epoch: 1 [19200/60000 (32%)]	Loss: 0.134212
Train Epoch: 1 [19232/60000 (32%)]	Loss: 0.372402
Train Epoch: 1 [19264/60000 (32%)]	Loss: 0.190219
Train Epoch: 1 [19296/60000 (32%)]	Loss: 0.086651
Train Epoch: 1 [19328/60000 (32%)]	Loss: 0.277142
Train Epoch: 1 [19360/60000 (32%)]	Loss: 0.161448
Train Epoch: 1 [19392/60000 (32%)]	Loss: 0.138540
Train Epoch: 1 [19424/60000 (32%)]	Loss: 0.145353
Train Epoch: 1 [19456/60000 (32%)]	Loss: 0.092597
Train Epoch: 1 [19488/60000 (32%)]	Loss: 0.205563
Train Epoch: 1 [19520/60000 (33%)]	Loss: 0.208139
Train Epoch: 1 [19552/60000 (33%)]	Loss: 0.220390
Train Epoch: 1 [19584/60000 (33%)]	Loss: 0.153869
Train Epoch: 1 [19616/60000 (33%)]	Loss: 0.143371
Train Epoch: 1 [19648/60000 (33%)]	Loss: 0.121472
Train Epoch: 1 [19680/60000 (33%)]	Loss: 0.058905
Train Epoch: 1 [19712/60000 (33%)]	Loss: 0.022638
Train Epoch: 1 [19744/60000 (33%)]	Loss: 0.075406
Train Epoch: 1 [19776/60000 (33%)]	Loss: 0.159465
Train Epoch: 1 [19808/60000 (33%)]	Loss: 0.415442
Train Epoch: 1 [19840/60000 (33%)]	Loss: 0.121503
Train Epoch: 1 [19872/60000 (33%)]	Loss: 0.057646
Train Epoch: 1 [19904/60000 (33%)]	Loss: 0.116535
Train Epoch: 1 [19936/60000 (33%)]	Loss: 0.173227
Train Epoch: 1 [19968/60000 (33%)]	Loss: 0.156700
Train Epoch: 1 [20000/60000 (33%)]	Loss: 0.207252
Train Epoch: 1 [20032/60000 (33%)]	Loss: 0.196379
Train Epoch: 1 [20064/60000 (33%)]	Loss: 0.330811
Train Epoch: 1 [20096/60000 (33%)]	Loss: 0.215777
Train Epoch: 1 [20128/60000 (34%)]	Loss: 0.061674
Train Epoch: 1 [20160/60000 (34%)]	Loss: 0.473655
Train Epoch: 1 [20192/60000 (34%)]	Loss: 0.204574
Train Epoch: 1 [20224/60000 (34%)]	Loss: 0.175617
Train Epoch: 1 [20256/60000 (34%)]	Loss: 0.095311
Train Epoch: 1 [20288/60000 (34%)]	Loss: 0.074566
Train Epoch: 1 [20320/60000 (34%)]	Loss: 0.275312
Train Epoch: 1 [20352/60000 (34%)]	Loss: 0.108959
Train Epoch: 1 [20384/60000 (34%)]	Loss: 0.032430
Train Epoch: 1 [20416/60000 (34%)]	Loss: 0.053598
Train Epoch: 1 [20448/60000 (34%)]	Loss: 0.101315

Train Epoch: 1 [20480/60000 (34%)]	Loss: 0.041213
Train Epoch: 1 [20512/60000 (34%)]	Loss: 0.132816
Train Epoch: 1 [20544/60000 (34%)]	Loss: 0.291790
Train Epoch: 1 [20576/60000 (34%)]	Loss: 0.157908
Train Epoch: 1 [20608/60000 (34%)]	Loss: 0.080027
Train Epoch: 1 [20640/60000 (34%)]	Loss: 0.271417
Train Epoch: 1 [20672/60000 (34%)]	Loss: 0.362902
Train Epoch: 1 [20704/60000 (35%)]	Loss: 0.391093
Train Epoch: 1 [20736/60000 (35%)]	Loss: 0.202505
Train Epoch: 1 [20768/60000 (35%)]	Loss: 0.373821
Train Epoch: 1 [20800/60000 (35%)]	Loss: 0.171024
Train Epoch: 1 [20832/60000 (35%)]	Loss: 0.219609
Train Epoch: 1 [20864/60000 (35%)]	Loss: 0.252174
Train Epoch: 1 [20896/60000 (35%)]	Loss: 0.601068
Train Epoch: 1 [20928/60000 (35%)]	Loss: 0.270804
Train Epoch: 1 [20960/60000 (35%)]	Loss: 0.300706
Train Epoch: 1 [20992/60000 (35%)]	Loss: 0.271597
Train Epoch: 1 [21024/60000 (35%)]	Loss: 0.340748
Train Epoch: 1 [21056/60000 (35%)]	Loss: 0.304664
Train Epoch: 1 [21088/60000 (35%)]	Loss: 0.144580
Train Epoch: 1 [21120/60000 (35%)]	Loss: 0.324020
Train Epoch: 1 [21152/60000 (35%)]	Loss: 0.128001
Train Epoch: 1 [21184/60000 (35%)]	Loss: 0.212621
Train Epoch: 1 [21216/60000 (35%)]	Loss: 0.105169
Train Epoch: 1 [21248/60000 (35%)]	Loss: 0.047132
Train Epoch: 1 [21280/60000 (35%)]	Loss: 0.415978
Train Epoch: 1 [21312/60000 (36%)]	Loss: 0.125972
Train Epoch: 1 [21344/60000 (36%)]	Loss: 0.280736
Train Epoch: 1 [21376/60000 (36%)]	Loss: 0.250920
Train Epoch: 1 [21408/60000 (36%)]	Loss: 0.179812
Train Epoch: 1 [21440/60000 (36%)]	Loss: 0.123846
Train Epoch: 1 [21472/60000 (36%)]	Loss: 0.027875
Train Epoch: 1 [21504/60000 (36%)]	Loss: 0.204780
Train Epoch: 1 [21536/60000 (36%)]	Loss: 0.126736
Train Epoch: 1 [21568/60000 (36%)]	Loss: 0.353541
Train Epoch: 1 [21600/60000 (36%)]	Loss: 0.345947
Train Epoch: 1 [21632/60000 (36%)]	Loss: 0.122205
Train Epoch: 1 [21664/60000 (36%)]	Loss: 0.139520
Train Epoch: 1 [21696/60000 (36%)]	Loss: 0.187730
Train Epoch: 1 [21728/60000 (36%)]	Loss: 0.118220
Train Epoch: 1 [21760/60000 (36%)]	Loss: 0.042279
Train Epoch: 1 [21792/60000 (36%)]	Loss: 0.027876
Train Epoch: 1 [21824/60000 (36%)]	Loss: 0.089974
Train Epoch: 1 [21856/60000 (36%)]	Loss: 0.064733
Train Epoch: 1 [21888/60000 (36%)]	Loss: 0.136328
Train Epoch: 1 [21920/60000 (37%)]	Loss: 0.197654
Train Epoch: 1 [21952/60000 (37%)]	Loss: 0.249383
Train Epoch: 1 [21984/60000 (37%)]	Loss: 0.142904
Train Epoch: 1 [22016/60000 (37%)]	Loss: 0.022992
Train Epoch: 1 [22048/60000 (37%)]	Loss: 0.082467
Train Epoch: 1 [22080/60000 (37%)]	Loss: 0.065162

Train Epoch: 1 [22112/60000 (37%)]	Loss: 0.290023
Train Epoch: 1 [22144/60000 (37%)]	Loss: 0.084108
Train Epoch: 1 [22176/60000 (37%)]	Loss: 0.364621
Train Epoch: 1 [22208/60000 (37%)]	Loss: 0.426728
Train Epoch: 1 [22240/60000 (37%)]	Loss: 0.088600
Train Epoch: 1 [22272/60000 (37%)]	Loss: 0.219765
Train Epoch: 1 [22304/60000 (37%)]	Loss: 0.138703
Train Epoch: 1 [22336/60000 (37%)]	Loss: 0.038520
Train Epoch: 1 [22368/60000 (37%)]	Loss: 0.039614
Train Epoch: 1 [22400/60000 (37%)]	Loss: 0.087805
Train Epoch: 1 [22432/60000 (37%)]	Loss: 0.157117
Train Epoch: 1 [22464/60000 (37%)]	Loss: 0.296987
Train Epoch: 1 [22496/60000 (37%)]	Loss: 0.437685
Train Epoch: 1 [22528/60000 (38%)]	Loss: 0.344894
Train Epoch: 1 [22560/60000 (38%)]	Loss: 0.462825
Train Epoch: 1 [22592/60000 (38%)]	Loss: 0.357924
Train Epoch: 1 [22624/60000 (38%)]	Loss: 0.286931
Train Epoch: 1 [22656/60000 (38%)]	Loss: 0.196831
Train Epoch: 1 [22688/60000 (38%)]	Loss: 0.156030
Train Epoch: 1 [22720/60000 (38%)]	Loss: 0.290309
Train Epoch: 1 [22752/60000 (38%)]	Loss: 0.213272
Train Epoch: 1 [22784/60000 (38%)]	Loss: 0.133104
Train Epoch: 1 [22816/60000 (38%)]	Loss: 0.083505
Train Epoch: 1 [22848/60000 (38%)]	Loss: 0.061324
Train Epoch: 1 [22880/60000 (38%)]	Loss: 0.192009
Train Epoch: 1 [22912/60000 (38%)]	Loss: 0.108917
Train Epoch: 1 [22944/60000 (38%)]	Loss: 0.033281
Train Epoch: 1 [22976/60000 (38%)]	Loss: 0.070324
Train Epoch: 1 [23008/60000 (38%)]	Loss: 0.248100
Train Epoch: 1 [23040/60000 (38%)]	Loss: 0.126872
Train Epoch: 1 [23072/60000 (38%)]	Loss: 0.611325
Train Epoch: 1 [23104/60000 (39%)]	Loss: 0.279310
Train Epoch: 1 [23136/60000 (39%)]	Loss: 0.175127
Train Epoch: 1 [23168/60000 (39%)]	Loss: 0.087509
Train Epoch: 1 [23200/60000 (39%)]	Loss: 0.258248
Train Epoch: 1 [23232/60000 (39%)]	Loss: 0.034772
Train Epoch: 1 [23264/60000 (39%)]	Loss: 0.097326
Train Epoch: 1 [23296/60000 (39%)]	Loss: 0.049415
Train Epoch: 1 [23328/60000 (39%)]	Loss: 0.236732
Train Epoch: 1 [23360/60000 (39%)]	Loss: 0.119999
Train Epoch: 1 [23392/60000 (39%)]	Loss: 0.101748
Train Epoch: 1 [23424/60000 (39%)]	Loss: 0.141922
Train Epoch: 1 [23456/60000 (39%)]	Loss: 0.392351
Train Epoch: 1 [23488/60000 (39%)]	Loss: 0.117958
Train Epoch: 1 [23520/60000 (39%)]	Loss: 0.144706
Train Epoch: 1 [23552/60000 (39%)]	Loss: 0.166972
Train Epoch: 1 [23584/60000 (39%)]	Loss: 0.301482
Train Epoch: 1 [23616/60000 (39%)]	Loss: 0.318055
Train Epoch: 1 [23648/60000 (39%)]	Loss: 0.102455
Train Epoch: 1 [23680/60000 (39%)]	Loss: 0.228866
Train Epoch: 1 [23712/60000 (40%)]	Loss: 0.481604

Train Epoch: 1 [23744/60000 (40%)]	Loss: 0.080936
Train Epoch: 1 [23776/60000 (40%)]	Loss: 0.118367
Train Epoch: 1 [23808/60000 (40%)]	Loss: 0.202183
Train Epoch: 1 [23840/60000 (40%)]	Loss: 0.237159
Train Epoch: 1 [23872/60000 (40%)]	Loss: 0.194303
Train Epoch: 1 [23904/60000 (40%)]	Loss: 0.192073
Train Epoch: 1 [23936/60000 (40%)]	Loss: 0.372773
Train Epoch: 1 [23968/60000 (40%)]	Loss: 0.125682
Train Epoch: 1 [24000/60000 (40%)]	Loss: 0.123340
Train Epoch: 1 [24032/60000 (40%)]	Loss: 0.209803
Train Epoch: 1 [24064/60000 (40%)]	Loss: 0.078511
Train Epoch: 1 [24096/60000 (40%)]	Loss: 0.107630
Train Epoch: 1 [24128/60000 (40%)]	Loss: 0.163969
Train Epoch: 1 [24160/60000 (40%)]	Loss: 0.146923
Train Epoch: 1 [24192/60000 (40%)]	Loss: 0.285417
Train Epoch: 1 [24224/60000 (40%)]	Loss: 0.083733
Train Epoch: 1 [24256/60000 (40%)]	Loss: 0.270430
Train Epoch: 1 [24288/60000 (40%)]	Loss: 0.132039
Train Epoch: 1 [24320/60000 (41%)]	Loss: 0.075219
Train Epoch: 1 [24352/60000 (41%)]	Loss: 0.069066
Train Epoch: 1 [24384/60000 (41%)]	Loss: 0.076752
Train Epoch: 1 [24416/60000 (41%)]	Loss: 0.110818
Train Epoch: 1 [24448/60000 (41%)]	Loss: 0.088240
Train Epoch: 1 [24480/60000 (41%)]	Loss: 0.102242
Train Epoch: 1 [24512/60000 (41%)]	Loss: 0.256756
Train Epoch: 1 [24544/60000 (41%)]	Loss: 0.157844
Train Epoch: 1 [24576/60000 (41%)]	Loss: 0.216324
Train Epoch: 1 [24608/60000 (41%)]	Loss: 0.317884
Train Epoch: 1 [24640/60000 (41%)]	Loss: 0.204497
Train Epoch: 1 [24672/60000 (41%)]	Loss: 0.253715
Train Epoch: 1 [24704/60000 (41%)]	Loss: 0.163132
Train Epoch: 1 [24736/60000 (41%)]	Loss: 0.195831
Train Epoch: 1 [24768/60000 (41%)]	Loss: 0.566284
Train Epoch: 1 [24800/60000 (41%)]	Loss: 0.113980
Train Epoch: 1 [24832/60000 (41%)]	Loss: 0.130745
Train Epoch: 1 [24864/60000 (41%)]	Loss: 0.075062
Train Epoch: 1 [24896/60000 (41%)]	Loss: 0.169911
Train Epoch: 1 [24928/60000 (42%)]	Loss: 0.302284
Train Epoch: 1 [24960/60000 (42%)]	Loss: 0.172879
Train Epoch: 1 [24992/60000 (42%)]	Loss: 0.053051
Train Epoch: 1 [25024/60000 (42%)]	Loss: 0.031443
Train Epoch: 1 [25056/60000 (42%)]	Loss: 0.058812
Train Epoch: 1 [25088/60000 (42%)]	Loss: 0.207451
Train Epoch: 1 [25120/60000 (42%)]	Loss: 0.116919
Train Epoch: 1 [25152/60000 (42%)]	Loss: 0.179122
Train Epoch: 1 [25184/60000 (42%)]	Loss: 0.241704
Train Epoch: 1 [25216/60000 (42%)]	Loss: 0.171271
Train Epoch: 1 [25248/60000 (42%)]	Loss: 0.068846
Train Epoch: 1 [25280/60000 (42%)]	Loss: 0.293516
Train Epoch: 1 [25312/60000 (42%)]	Loss: 0.104999
Train Epoch: 1 [25344/60000 (42%)]	Loss: 0.038148

Train Epoch: 1 [25376/60000 (42%)]	Loss: 0.106710
Train Epoch: 1 [25408/60000 (42%)]	Loss: 0.086893
Train Epoch: 1 [25440/60000 (42%)]	Loss: 0.101610
Train Epoch: 1 [25472/60000 (42%)]	Loss: 0.335475
Train Epoch: 1 [25504/60000 (43%)]	Loss: 0.028345
Train Epoch: 1 [25536/60000 (43%)]	Loss: 0.423391
Train Epoch: 1 [25568/60000 (43%)]	Loss: 0.103795
Train Epoch: 1 [25600/60000 (43%)]	Loss: 0.112549
Train Epoch: 1 [25632/60000 (43%)]	Loss: 0.097049
Train Epoch: 1 [25664/60000 (43%)]	Loss: 0.199730
Train Epoch: 1 [25696/60000 (43%)]	Loss: 0.167803
Train Epoch: 1 [25728/60000 (43%)]	Loss: 0.086167
Train Epoch: 1 [25760/60000 (43%)]	Loss: 0.098250
Train Epoch: 1 [25792/60000 (43%)]	Loss: 0.483704
Train Epoch: 1 [25824/60000 (43%)]	Loss: 0.152141
Train Epoch: 1 [25856/60000 (43%)]	Loss: 0.090974
Train Epoch: 1 [25888/60000 (43%)]	Loss: 0.135991
Train Epoch: 1 [25920/60000 (43%)]	Loss: 0.161062
Train Epoch: 1 [25952/60000 (43%)]	Loss: 0.133340
Train Epoch: 1 [25984/60000 (43%)]	Loss: 0.090294
Train Epoch: 1 [26016/60000 (43%)]	Loss: 0.155147
Train Epoch: 1 [26048/60000 (43%)]	Loss: 0.195266
Train Epoch: 1 [26080/60000 (43%)]	Loss: 0.042557
Train Epoch: 1 [26112/60000 (44%)]	Loss: 0.103356
Train Epoch: 1 [26144/60000 (44%)]	Loss: 0.081453
Train Epoch: 1 [26176/60000 (44%)]	Loss: 0.044868
Train Epoch: 1 [26208/60000 (44%)]	Loss: 0.118604
Train Epoch: 1 [26240/60000 (44%)]	Loss: 0.136444
Train Epoch: 1 [26272/60000 (44%)]	Loss: 0.124926
Train Epoch: 1 [26304/60000 (44%)]	Loss: 0.033971
Train Epoch: 1 [26336/60000 (44%)]	Loss: 0.323433
Train Epoch: 1 [26368/60000 (44%)]	Loss: 0.356071
Train Epoch: 1 [26400/60000 (44%)]	Loss: 0.228892
Train Epoch: 1 [26432/60000 (44%)]	Loss: 0.186152
Train Epoch: 1 [26464/60000 (44%)]	Loss: 0.178921
Train Epoch: 1 [26496/60000 (44%)]	Loss: 0.344832
Train Epoch: 1 [26528/60000 (44%)]	Loss: 0.106580
Train Epoch: 1 [26560/60000 (44%)]	Loss: 0.276910
Train Epoch: 1 [26592/60000 (44%)]	Loss: 0.095425
Train Epoch: 1 [26624/60000 (44%)]	Loss: 0.758437
Train Epoch: 1 [26656/60000 (44%)]	Loss: 0.078533
Train Epoch: 1 [26688/60000 (44%)]	Loss: 0.194645
Train Epoch: 1 [26720/60000 (45%)]	Loss: 0.611485
Train Epoch: 1 [26752/60000 (45%)]	Loss: 0.232901
Train Epoch: 1 [26784/60000 (45%)]	Loss: 0.150552
Train Epoch: 1 [26816/60000 (45%)]	Loss: 0.079482
Train Epoch: 1 [26848/60000 (45%)]	Loss: 0.365131
Train Epoch: 1 [26880/60000 (45%)]	Loss: 0.614699
Train Epoch: 1 [26912/60000 (45%)]	Loss: 0.131973
Train Epoch: 1 [26944/60000 (45%)]	Loss: 0.074916
Train Epoch: 1 [26976/60000 (45%)]	Loss: 0.029274

Train Epoch: 1 [27008/60000 (45%)]	Loss: 0.145251
Train Epoch: 1 [27040/60000 (45%)]	Loss: 0.044254
Train Epoch: 1 [27072/60000 (45%)]	Loss: 0.106624
Train Epoch: 1 [27104/60000 (45%)]	Loss: 0.112199
Train Epoch: 1 [27136/60000 (45%)]	Loss: 0.433056
Train Epoch: 1 [27168/60000 (45%)]	Loss: 0.346592
Train Epoch: 1 [27200/60000 (45%)]	Loss: 0.196450
Train Epoch: 1 [27232/60000 (45%)]	Loss: 0.260829
Train Epoch: 1 [27264/60000 (45%)]	Loss: 0.063519
Train Epoch: 1 [27296/60000 (45%)]	Loss: 0.085487
Train Epoch: 1 [27328/60000 (46%)]	Loss: 0.141891
Train Epoch: 1 [27360/60000 (46%)]	Loss: 0.057179
Train Epoch: 1 [27392/60000 (46%)]	Loss: 0.065473
Train Epoch: 1 [27424/60000 (46%)]	Loss: 0.153458
Train Epoch: 1 [27456/60000 (46%)]	Loss: 0.062168
Train Epoch: 1 [27488/60000 (46%)]	Loss: 0.302175
Train Epoch: 1 [27520/60000 (46%)]	Loss: 0.152875
Train Epoch: 1 [27552/60000 (46%)]	Loss: 0.057444
Train Epoch: 1 [27584/60000 (46%)]	Loss: 0.206816
Train Epoch: 1 [27616/60000 (46%)]	Loss: 0.182797
Train Epoch: 1 [27648/60000 (46%)]	Loss: 0.126039
Train Epoch: 1 [27680/60000 (46%)]	Loss: 0.243868
Train Epoch: 1 [27712/60000 (46%)]	Loss: 0.113813
Train Epoch: 1 [27744/60000 (46%)]	Loss: 0.024770
Train Epoch: 1 [27776/60000 (46%)]	Loss: 0.170130
Train Epoch: 1 [27808/60000 (46%)]	Loss: 0.221326
Train Epoch: 1 [27840/60000 (46%)]	Loss: 0.078001
Train Epoch: 1 [27872/60000 (46%)]	Loss: 0.089301
Train Epoch: 1 [27904/60000 (47%)]	Loss: 0.047550
Train Epoch: 1 [27936/60000 (47%)]	Loss: 0.104844
Train Epoch: 1 [27968/60000 (47%)]	Loss: 0.186828
Train Epoch: 1 [28000/60000 (47%)]	Loss: 0.082955
Train Epoch: 1 [28032/60000 (47%)]	Loss: 0.026554
Train Epoch: 1 [28064/60000 (47%)]	Loss: 0.089338
Train Epoch: 1 [28096/60000 (47%)]	Loss: 0.104267
Train Epoch: 1 [28128/60000 (47%)]	Loss: 0.181556
Train Epoch: 1 [28160/60000 (47%)]	Loss: 0.141335
Train Epoch: 1 [28192/60000 (47%)]	Loss: 0.138526
Train Epoch: 1 [28224/60000 (47%)]	Loss: 0.058083
Train Epoch: 1 [28256/60000 (47%)]	Loss: 0.127818
Train Epoch: 1 [28288/60000 (47%)]	Loss: 0.095102
Train Epoch: 1 [28320/60000 (47%)]	Loss: 0.029024
Train Epoch: 1 [28352/60000 (47%)]	Loss: 0.464215
Train Epoch: 1 [28384/60000 (47%)]	Loss: 0.244181
Train Epoch: 1 [28416/60000 (47%)]	Loss: 0.068677
Train Epoch: 1 [28448/60000 (47%)]	Loss: 0.047425
Train Epoch: 1 [28480/60000 (47%)]	Loss: 0.199403
Train Epoch: 1 [28512/60000 (48%)]	Loss: 0.104645
Train Epoch: 1 [28544/60000 (48%)]	Loss: 0.285142
Train Epoch: 1 [28576/60000 (48%)]	Loss: 0.135084
Train Epoch: 1 [28608/60000 (48%)]	Loss: 0.323632

Train Epoch: 1 [28640/60000 (48%)]	Loss: 0.465275
Train Epoch: 1 [28672/60000 (48%)]	Loss: 0.070362
Train Epoch: 1 [28704/60000 (48%)]	Loss: 0.241776
Train Epoch: 1 [28736/60000 (48%)]	Loss: 0.029824
Train Epoch: 1 [28768/60000 (48%)]	Loss: 0.163784
Train Epoch: 1 [28800/60000 (48%)]	Loss: 0.113918
Train Epoch: 1 [28832/60000 (48%)]	Loss: 0.089248
Train Epoch: 1 [28864/60000 (48%)]	Loss: 0.032596
Train Epoch: 1 [28896/60000 (48%)]	Loss: 0.047776
Train Epoch: 1 [28928/60000 (48%)]	Loss: 0.241324
Train Epoch: 1 [28960/60000 (48%)]	Loss: 0.209530
Train Epoch: 1 [28992/60000 (48%)]	Loss: 0.072230
Train Epoch: 1 [29024/60000 (48%)]	Loss: 0.141336
Train Epoch: 1 [29056/60000 (48%)]	Loss: 0.262192
Train Epoch: 1 [29088/60000 (48%)]	Loss: 0.130641
Train Epoch: 1 [29120/60000 (49%)]	Loss: 0.138841
Train Epoch: 1 [29152/60000 (49%)]	Loss: 0.331964
Train Epoch: 1 [29184/60000 (49%)]	Loss: 0.275829
Train Epoch: 1 [29216/60000 (49%)]	Loss: 0.127579
Train Epoch: 1 [29248/60000 (49%)]	Loss: 0.057250
Train Epoch: 1 [29280/60000 (49%)]	Loss: 0.319347
Train Epoch: 1 [29312/60000 (49%)]	Loss: 0.172430
Train Epoch: 1 [29344/60000 (49%)]	Loss: 0.097197
Train Epoch: 1 [29376/60000 (49%)]	Loss: 0.062941
Train Epoch: 1 [29408/60000 (49%)]	Loss: 0.278525
Train Epoch: 1 [29440/60000 (49%)]	Loss: 0.129449
Train Epoch: 1 [29472/60000 (49%)]	Loss: 0.038297
Train Epoch: 1 [29504/60000 (49%)]	Loss: 0.205980
Train Epoch: 1 [29536/60000 (49%)]	Loss: 0.048637
Train Epoch: 1 [29568/60000 (49%)]	Loss: 0.235121
Train Epoch: 1 [29600/60000 (49%)]	Loss: 0.059188
Train Epoch: 1 [29632/60000 (49%)]	Loss: 0.161243
Train Epoch: 1 [29664/60000 (49%)]	Loss: 0.158795
Train Epoch: 1 [29696/60000 (49%)]	Loss: 0.263290
Train Epoch: 1 [29728/60000 (50%)]	Loss: 0.212891
Train Epoch: 1 [29760/60000 (50%)]	Loss: 0.114695
Train Epoch: 1 [29792/60000 (50%)]	Loss: 0.084568
Train Epoch: 1 [29824/60000 (50%)]	Loss: 0.427007
Train Epoch: 1 [29856/60000 (50%)]	Loss: 0.246241
Train Epoch: 1 [29888/60000 (50%)]	Loss: 0.156446
Train Epoch: 1 [29920/60000 (50%)]	Loss: 0.276994
Train Epoch: 1 [29952/60000 (50%)]	Loss: 0.282786
Train Epoch: 1 [29984/60000 (50%)]	Loss: 0.138032
Train Epoch: 1 [30016/60000 (50%)]	Loss: 0.166421
Train Epoch: 1 [30048/60000 (50%)]	Loss: 0.262558
Train Epoch: 1 [30080/60000 (50%)]	Loss: 0.112911
Train Epoch: 1 [30112/60000 (50%)]	Loss: 0.242238
Train Epoch: 1 [30144/60000 (50%)]	Loss: 0.211603
Train Epoch: 1 [30176/60000 (50%)]	Loss: 0.112037
Train Epoch: 1 [30208/60000 (50%)]	Loss: 0.191991
Train Epoch: 1 [30240/60000 (50%)]	Loss: 0.147303



Train Epoch: 1 [30272/60000 (50%)]	Loss: 0.070057
Train Epoch: 1 [30304/60000 (51%)]	Loss: 0.150755
Train Epoch: 1 [30336/60000 (51%)]	Loss: 0.175972
Train Epoch: 1 [30368/60000 (51%)]	Loss: 0.069726
Train Epoch: 1 [30400/60000 (51%)]	Loss: 0.128063
Train Epoch: 1 [30432/60000 (51%)]	Loss: 0.124977
Train Epoch: 1 [30464/60000 (51%)]	Loss: 0.171211
Train Epoch: 1 [30496/60000 (51%)]	Loss: 0.121654
Train Epoch: 1 [30528/60000 (51%)]	Loss: 0.114127
Train Epoch: 1 [30560/60000 (51%)]	Loss: 0.131207
Train Epoch: 1 [30592/60000 (51%)]	Loss: 0.175277
Train Epoch: 1 [30624/60000 (51%)]	Loss: 0.073266
Train Epoch: 1 [30656/60000 (51%)]	Loss: 0.093293
Train Epoch: 1 [30688/60000 (51%)]	Loss: 0.264545
Train Epoch: 1 [30720/60000 (51%)]	Loss: 0.193032
Train Epoch: 1 [30752/60000 (51%)]	Loss: 0.074657
Train Epoch: 1 [30784/60000 (51%)]	Loss: 0.182009
Train Epoch: 1 [30816/60000 (51%)]	Loss: 0.164132
Train Epoch: 1 [30848/60000 (51%)]	Loss: 0.163561
Train Epoch: 1 [30880/60000 (51%)]	Loss: 0.412430
Train Epoch: 1 [30912/60000 (52%)]	Loss: 0.092501
Train Epoch: 1 [30944/60000 (52%)]	Loss: 0.260083
Train Epoch: 1 [30976/60000 (52%)]	Loss: 0.092764
Train Epoch: 1 [31008/60000 (52%)]	Loss: 0.171809
Train Epoch: 1 [31040/60000 (52%)]	Loss: 0.104598
Train Epoch: 1 [31072/60000 (52%)]	Loss: 0.174890
Train Epoch: 1 [31104/60000 (52%)]	Loss: 0.527814
Train Epoch: 1 [31136/60000 (52%)]	Loss: 0.036122
Train Epoch: 1 [31168/60000 (52%)]	Loss: 0.529530
Train Epoch: 1 [31200/60000 (52%)]	Loss: 0.149955
Train Epoch: 1 [31232/60000 (52%)]	Loss: 0.279314
Train Epoch: 1 [31264/60000 (52%)]	Loss: 0.162017
Train Epoch: 1 [31296/60000 (52%)]	Loss: 0.561405
Train Epoch: 1 [31328/60000 (52%)]	Loss: 0.154206
Train Epoch: 1 [31360/60000 (52%)]	Loss: 0.123722
Train Epoch: 1 [31392/60000 (52%)]	Loss: 0.216789
Train Epoch: 1 [31424/60000 (52%)]	Loss: 0.212698
Train Epoch: 1 [31456/60000 (52%)]	Loss: 0.151638
Train Epoch: 1 [31488/60000 (52%)]	Loss: 0.129585
Train Epoch: 1 [31520/60000 (53%)]	Loss: 0.132592
Train Epoch: 1 [31552/60000 (53%)]	Loss: 0.148320
Train Epoch: 1 [31584/60000 (53%)]	Loss: 0.242123
Train Epoch: 1 [31616/60000 (53%)]	Loss: 0.028475
Train Epoch: 1 [31648/60000 (53%)]	Loss: 0.400140
Train Epoch: 1 [31680/60000 (53%)]	Loss: 0.245617
Train Epoch: 1 [31712/60000 (53%)]	Loss: 0.377225
Train Epoch: 1 [31744/60000 (53%)]	Loss: 0.167976
Train Epoch: 1 [31776/60000 (53%)]	Loss: 0.191449
Train Epoch: 1 [31808/60000 (53%)]	Loss: 0.038852
Train Epoch: 1 [31840/60000 (53%)]	Loss: 0.316284
Train Epoch: 1 [31872/60000 (53%)]	Loss: 0.049384

Train Epoch: 1 [31904/60000 (53%)]	Loss: 0.077230
Train Epoch: 1 [31936/60000 (53%)]	Loss: 0.398827
Train Epoch: 1 [31968/60000 (53%)]	Loss: 0.067327
Train Epoch: 1 [32000/60000 (53%)]	Loss: 0.268080
Train Epoch: 1 [32032/60000 (53%)]	Loss: 0.286115
Train Epoch: 1 [32064/60000 (53%)]	Loss: 0.111013
Train Epoch: 1 [32096/60000 (53%)]	Loss: 0.247470
Train Epoch: 1 [32128/60000 (54%)]	Loss: 0.119322
Train Epoch: 1 [32160/60000 (54%)]	Loss: 0.195408
Train Epoch: 1 [32192/60000 (54%)]	Loss: 0.103856
Train Epoch: 1 [32224/60000 (54%)]	Loss: 0.100074
Train Epoch: 1 [32256/60000 (54%)]	Loss: 0.251330
Train Epoch: 1 [32288/60000 (54%)]	Loss: 0.194469
Train Epoch: 1 [32320/60000 (54%)]	Loss: 0.499391
Train Epoch: 1 [32352/60000 (54%)]	Loss: 0.089586
Train Epoch: 1 [32384/60000 (54%)]	Loss: 0.254045
Train Epoch: 1 [32416/60000 (54%)]	Loss: 0.291898
Train Epoch: 1 [32448/60000 (54%)]	Loss: 0.284938
Train Epoch: 1 [32480/60000 (54%)]	Loss: 0.244894
Train Epoch: 1 [32512/60000 (54%)]	Loss: 0.062024
Train Epoch: 1 [32544/60000 (54%)]	Loss: 0.173776
Train Epoch: 1 [32576/60000 (54%)]	Loss: 0.016946
Train Epoch: 1 [32608/60000 (54%)]	Loss: 0.050657
Train Epoch: 1 [32640/60000 (54%)]	Loss: 0.368642
Train Epoch: 1 [32672/60000 (54%)]	Loss: 0.162855
Train Epoch: 1 [32704/60000 (55%)]	Loss: 0.055222
Train Epoch: 1 [32736/60000 (55%)]	Loss: 0.118292
Train Epoch: 1 [32768/60000 (55%)]	Loss: 0.323166
Train Epoch: 1 [32800/60000 (55%)]	Loss: 0.076894
Train Epoch: 1 [32832/60000 (55%)]	Loss: 0.094784
Train Epoch: 1 [32864/60000 (55%)]	Loss: 0.219311
Train Epoch: 1 [32896/60000 (55%)]	Loss: 0.172965
Train Epoch: 1 [32928/60000 (55%)]	Loss: 0.079667
Train Epoch: 1 [32960/60000 (55%)]	Loss: 0.022338
Train Epoch: 1 [32992/60000 (55%)]	Loss: 0.077127
Train Epoch: 1 [33024/60000 (55%)]	Loss: 0.119446
Train Epoch: 1 [33056/60000 (55%)]	Loss: 0.217917
Train Epoch: 1 [33088/60000 (55%)]	Loss: 0.030451
Train Epoch: 1 [33120/60000 (55%)]	Loss: 0.247062
Train Epoch: 1 [33152/60000 (55%)]	Loss: 0.056173
Train Epoch: 1 [33184/60000 (55%)]	Loss: 0.096500
Train Epoch: 1 [33216/60000 (55%)]	Loss: 0.292693
Train Epoch: 1 [33248/60000 (55%)]	Loss: 0.055649
Train Epoch: 1 [33280/60000 (55%)]	Loss: 0.180217
Train Epoch: 1 [33312/60000 (56%)]	Loss: 0.150359
Train Epoch: 1 [33344/60000 (56%)]	Loss: 0.095083
Train Epoch: 1 [33376/60000 (56%)]	Loss: 0.211406
Train Epoch: 1 [33408/60000 (56%)]	Loss: 0.101197
Train Epoch: 1 [33440/60000 (56%)]	Loss: 0.131370
Train Epoch: 1 [33472/60000 (56%)]	Loss: 0.094710
Train Epoch: 1 [33504/60000 (56%)]	Loss: 0.198248

Train Epoch: 1 [33536/60000 (56%)]	Loss: 0.120423
Train Epoch: 1 [33568/60000 (56%)]	Loss: 0.112333
Train Epoch: 1 [33600/60000 (56%)]	Loss: 0.112296
Train Epoch: 1 [33632/60000 (56%)]	Loss: 0.017255
Train Epoch: 1 [33664/60000 (56%)]	Loss: 0.129514
Train Epoch: 1 [33696/60000 (56%)]	Loss: 0.055528
Train Epoch: 1 [33728/60000 (56%)]	Loss: 0.121845
Train Epoch: 1 [33760/60000 (56%)]	Loss: 0.470404
Train Epoch: 1 [33792/60000 (56%)]	Loss: 0.055247
Train Epoch: 1 [33824/60000 (56%)]	Loss: 0.063096
Train Epoch: 1 [33856/60000 (56%)]	Loss: 0.022013
Train Epoch: 1 [33888/60000 (56%)]	Loss: 0.076203
Train Epoch: 1 [33920/60000 (57%)]	Loss: 0.013841
Train Epoch: 1 [33952/60000 (57%)]	Loss: 0.084656
Train Epoch: 1 [33984/60000 (57%)]	Loss: 0.114752
Train Epoch: 1 [34016/60000 (57%)]	Loss: 0.047957
Train Epoch: 1 [34048/60000 (57%)]	Loss: 0.321444
Train Epoch: 1 [34080/60000 (57%)]	Loss: 0.079594
Train Epoch: 1 [34112/60000 (57%)]	Loss: 0.163627
Train Epoch: 1 [34144/60000 (57%)]	Loss: 0.022605
Train Epoch: 1 [34176/60000 (57%)]	Loss: 0.056848
Train Epoch: 1 [34208/60000 (57%)]	Loss: 0.094771
Train Epoch: 1 [34240/60000 (57%)]	Loss: 0.114492
Train Epoch: 1 [34272/60000 (57%)]	Loss: 0.019184
Train Epoch: 1 [34304/60000 (57%)]	Loss: 0.108179
Train Epoch: 1 [34336/60000 (57%)]	Loss: 0.102269
Train Epoch: 1 [34368/60000 (57%)]	Loss: 0.062920
Train Epoch: 1 [34400/60000 (57%)]	Loss: 0.344749
Train Epoch: 1 [34432/60000 (57%)]	Loss: 0.121542
Train Epoch: 1 [34464/60000 (57%)]	Loss: 0.126295
Train Epoch: 1 [34496/60000 (57%)]	Loss: 0.250557
Train Epoch: 1 [34528/60000 (58%)]	Loss: 0.211149
Train Epoch: 1 [34560/60000 (58%)]	Loss: 0.140769
Train Epoch: 1 [34592/60000 (58%)]	Loss: 0.050130
Train Epoch: 1 [34624/60000 (58%)]	Loss: 0.059213
Train Epoch: 1 [34656/60000 (58%)]	Loss: 0.408430
Train Epoch: 1 [34688/60000 (58%)]	Loss: 0.236263
Train Epoch: 1 [34720/60000 (58%)]	Loss: 0.221717
Train Epoch: 1 [34752/60000 (58%)]	Loss: 0.171443
Train Epoch: 1 [34784/60000 (58%)]	Loss: 0.191763
Train Epoch: 1 [34816/60000 (58%)]	Loss: 0.286670
Train Epoch: 1 [34848/60000 (58%)]	Loss: 0.058099
Train Epoch: 1 [34880/60000 (58%)]	Loss: 0.144600
Train Epoch: 1 [34912/60000 (58%)]	Loss: 0.205483
Train Epoch: 1 [34944/60000 (58%)]	Loss: 0.058211
Train Epoch: 1 [34976/60000 (58%)]	Loss: 0.026430
Train Epoch: 1 [35008/60000 (58%)]	Loss: 0.064973
Train Epoch: 1 [35040/60000 (58%)]	Loss: 0.158034
Train Epoch: 1 [35072/60000 (58%)]	Loss: 0.086520
Train Epoch: 1 [35104/60000 (59%)]	Loss: 0.035540
Train Epoch: 1 [35136/60000 (59%)]	Loss: 0.215568

Train Epoch: 1 [35168/60000 (59%)]	Loss: 0.029160
Train Epoch: 1 [35200/60000 (59%)]	Loss: 0.052087
Train Epoch: 1 [35232/60000 (59%)]	Loss: 0.297691
Train Epoch: 1 [35264/60000 (59%)]	Loss: 0.024541
Train Epoch: 1 [35296/60000 (59%)]	Loss: 0.090537
Train Epoch: 1 [35328/60000 (59%)]	Loss: 0.064696
Train Epoch: 1 [35360/60000 (59%)]	Loss: 0.117486
Train Epoch: 1 [35392/60000 (59%)]	Loss: 0.167205
Train Epoch: 1 [35424/60000 (59%)]	Loss: 0.096069
Train Epoch: 1 [35456/60000 (59%)]	Loss: 0.402616
Train Epoch: 1 [35488/60000 (59%)]	Loss: 0.060097
Train Epoch: 1 [35520/60000 (59%)]	Loss: 0.034939
Train Epoch: 1 [35552/60000 (59%)]	Loss: 0.089741
Train Epoch: 1 [35584/60000 (59%)]	Loss: 0.089019
Train Epoch: 1 [35616/60000 (59%)]	Loss: 0.376925
Train Epoch: 1 [35648/60000 (59%)]	Loss: 0.049114
Train Epoch: 1 [35680/60000 (59%)]	Loss: 0.174816
Train Epoch: 1 [35712/60000 (60%)]	Loss: 0.088905
Train Epoch: 1 [35744/60000 (60%)]	Loss: 0.076680
Train Epoch: 1 [35776/60000 (60%)]	Loss: 0.065298
Train Epoch: 1 [35808/60000 (60%)]	Loss: 0.024238
Train Epoch: 1 [35840/60000 (60%)]	Loss: 0.206073
Train Epoch: 1 [35872/60000 (60%)]	Loss: 0.063235
Train Epoch: 1 [35904/60000 (60%)]	Loss: 0.147433
Train Epoch: 1 [35936/60000 (60%)]	Loss: 0.162099
Train Epoch: 1 [35968/60000 (60%)]	Loss: 0.162675
Train Epoch: 1 [36000/60000 (60%)]	Loss: 0.330755
Train Epoch: 1 [36032/60000 (60%)]	Loss: 0.299371
Train Epoch: 1 [36064/60000 (60%)]	Loss: 0.136373
Train Epoch: 1 [36096/60000 (60%)]	Loss: 0.261203
Train Epoch: 1 [36128/60000 (60%)]	Loss: 0.099135
Train Epoch: 1 [36160/60000 (60%)]	Loss: 0.109831
Train Epoch: 1 [36192/60000 (60%)]	Loss: 0.166811
Train Epoch: 1 [36224/60000 (60%)]	Loss: 0.098487
Train Epoch: 1 [36256/60000 (60%)]	Loss: 0.123565
Train Epoch: 1 [36288/60000 (60%)]	Loss: 0.061563
Train Epoch: 1 [36320/60000 (61%)]	Loss: 0.066926
Train Epoch: 1 [36352/60000 (61%)]	Loss: 0.098947
Train Epoch: 1 [36384/60000 (61%)]	Loss: 0.285460
Train Epoch: 1 [36416/60000 (61%)]	Loss: 0.455949
Train Epoch: 1 [36448/60000 (61%)]	Loss: 0.145592
Train Epoch: 1 [36480/60000 (61%)]	Loss: 0.111125
Train Epoch: 1 [36512/60000 (61%)]	Loss: 0.037172
Train Epoch: 1 [36544/60000 (61%)]	Loss: 0.127390
Train Epoch: 1 [36576/60000 (61%)]	Loss: 0.182546
Train Epoch: 1 [36608/60000 (61%)]	Loss: 0.087622
Train Epoch: 1 [36640/60000 (61%)]	Loss: 0.108206
Train Epoch: 1 [36672/60000 (61%)]	Loss: 0.023607
Train Epoch: 1 [36704/60000 (61%)]	Loss: 0.189524
Train Epoch: 1 [36736/60000 (61%)]	Loss: 0.298122
Train Epoch: 1 [36768/60000 (61%)]	Loss: 0.174500

Train Epoch: 1 [36800/60000 (61%)]	Loss: 0.143235
Train Epoch: 1 [36832/60000 (61%)]	Loss: 0.202095
Train Epoch: 1 [36864/60000 (61%)]	Loss: 0.113041
Train Epoch: 1 [36896/60000 (61%)]	Loss: 0.068798
Train Epoch: 1 [36928/60000 (62%)]	Loss: 0.116447
Train Epoch: 1 [36960/60000 (62%)]	Loss: 0.039076
Train Epoch: 1 [36992/60000 (62%)]	Loss: 0.061525
Train Epoch: 1 [37024/60000 (62%)]	Loss: 0.529090
Train Epoch: 1 [37056/60000 (62%)]	Loss: 0.396397
Train Epoch: 1 [37088/60000 (62%)]	Loss: 0.198140
Train Epoch: 1 [37120/60000 (62%)]	Loss: 0.240920
Train Epoch: 1 [37152/60000 (62%)]	Loss: 0.161226
Train Epoch: 1 [37184/60000 (62%)]	Loss: 0.200885
Train Epoch: 1 [37216/60000 (62%)]	Loss: 0.105800
Train Epoch: 1 [37248/60000 (62%)]	Loss: 0.266606
Train Epoch: 1 [37280/60000 (62%)]	Loss: 0.258384
Train Epoch: 1 [37312/60000 (62%)]	Loss: 0.101986
Train Epoch: 1 [37344/60000 (62%)]	Loss: 0.477924
Train Epoch: 1 [37376/60000 (62%)]	Loss: 0.423988
Train Epoch: 1 [37408/60000 (62%)]	Loss: 0.340345
Train Epoch: 1 [37440/60000 (62%)]	Loss: 0.540854
Train Epoch: 1 [37472/60000 (62%)]	Loss: 0.153347
Train Epoch: 1 [37504/60000 (63%)]	Loss: 0.172449
Train Epoch: 1 [37536/60000 (63%)]	Loss: 0.407044
Train Epoch: 1 [37568/60000 (63%)]	Loss: 0.169052
Train Epoch: 1 [37600/60000 (63%)]	Loss: 0.090422
Train Epoch: 1 [37632/60000 (63%)]	Loss: 0.095327
Train Epoch: 1 [37664/60000 (63%)]	Loss: 0.127573
Train Epoch: 1 [37696/60000 (63%)]	Loss: 0.274613
Train Epoch: 1 [37728/60000 (63%)]	Loss: 0.235983
Train Epoch: 1 [37760/60000 (63%)]	Loss: 0.138291
Train Epoch: 1 [37792/60000 (63%)]	Loss: 0.243806
Train Epoch: 1 [37824/60000 (63%)]	Loss: 0.279297
Train Epoch: 1 [37856/60000 (63%)]	Loss: 0.163921
Train Epoch: 1 [37888/60000 (63%)]	Loss: 0.122274
Train Epoch: 1 [37920/60000 (63%)]	Loss: 0.228450
Train Epoch: 1 [37952/60000 (63%)]	Loss: 0.066970
Train Epoch: 1 [37984/60000 (63%)]	Loss: 0.166001
Train Epoch: 1 [38016/60000 (63%)]	Loss: 0.186746
Train Epoch: 1 [38048/60000 (63%)]	Loss: 0.095433
Train Epoch: 1 [38080/60000 (63%)]	Loss: 0.097247
Train Epoch: 1 [38112/60000 (64%)]	Loss: 0.080697
Train Epoch: 1 [38144/60000 (64%)]	Loss: 0.042381
Train Epoch: 1 [38176/60000 (64%)]	Loss: 0.100111
Train Epoch: 1 [38208/60000 (64%)]	Loss: 0.072375
Train Epoch: 1 [38240/60000 (64%)]	Loss: 0.230935
Train Epoch: 1 [38272/60000 (64%)]	Loss: 0.039844
Train Epoch: 1 [38304/60000 (64%)]	Loss: 0.162653
Train Epoch: 1 [38336/60000 (64%)]	Loss: 0.084233
Train Epoch: 1 [38368/60000 (64%)]	Loss: 0.256323
Train Epoch: 1 [38400/60000 (64%)]	Loss: 0.210019

Train Epoch: 1 [38432/60000 (64%)]	Loss: 0.073867
Train Epoch: 1 [38464/60000 (64%)]	Loss: 0.068747
Train Epoch: 1 [38496/60000 (64%)]	Loss: 0.322170
Train Epoch: 1 [38528/60000 (64%)]	Loss: 0.118166
Train Epoch: 1 [38560/60000 (64%)]	Loss: 0.133216
Train Epoch: 1 [38592/60000 (64%)]	Loss: 0.222352
Train Epoch: 1 [38624/60000 (64%)]	Loss: 0.095560
Train Epoch: 1 [38656/60000 (64%)]	Loss: 0.138956
Train Epoch: 1 [38688/60000 (64%)]	Loss: 0.204336
Train Epoch: 1 [38720/60000 (65%)]	Loss: 0.025206
Train Epoch: 1 [38752/60000 (65%)]	Loss: 0.190145
Train Epoch: 1 [38784/60000 (65%)]	Loss: 0.095040
Train Epoch: 1 [38816/60000 (65%)]	Loss: 0.024194
Train Epoch: 1 [38848/60000 (65%)]	Loss: 0.025848
Train Epoch: 1 [38880/60000 (65%)]	Loss: 0.124811
Train Epoch: 1 [38912/60000 (65%)]	Loss: 0.123485
Train Epoch: 1 [38944/60000 (65%)]	Loss: 0.034290
Train Epoch: 1 [38976/60000 (65%)]	Loss: 0.066274
Train Epoch: 1 [39008/60000 (65%)]	Loss: 0.194905
Train Epoch: 1 [39040/60000 (65%)]	Loss: 0.050260
Train Epoch: 1 [39072/60000 (65%)]	Loss: 0.045542
Train Epoch: 1 [39104/60000 (65%)]	Loss: 0.186319
Train Epoch: 1 [39136/60000 (65%)]	Loss: 0.067324
Train Epoch: 1 [39168/60000 (65%)]	Loss: 0.249181
Train Epoch: 1 [39200/60000 (65%)]	Loss: 0.039769
Train Epoch: 1 [39232/60000 (65%)]	Loss: 0.097585
Train Epoch: 1 [39264/60000 (65%)]	Loss: 0.110578
Train Epoch: 1 [39296/60000 (65%)]	Loss: 0.370347
Train Epoch: 1 [39328/60000 (66%)]	Loss: 0.287592
Train Epoch: 1 [39360/60000 (66%)]	Loss: 0.183993
Train Epoch: 1 [39392/60000 (66%)]	Loss: 0.249149
Train Epoch: 1 [39424/60000 (66%)]	Loss: 0.146553
Train Epoch: 1 [39456/60000 (66%)]	Loss: 0.215484
Train Epoch: 1 [39488/60000 (66%)]	Loss: 0.039830
Train Epoch: 1 [39520/60000 (66%)]	Loss: 0.098152
Train Epoch: 1 [39552/60000 (66%)]	Loss: 0.071891
Train Epoch: 1 [39584/60000 (66%)]	Loss: 0.120558
Train Epoch: 1 [39616/60000 (66%)]	Loss: 0.233604
Train Epoch: 1 [39648/60000 (66%)]	Loss: 0.284970
Train Epoch: 1 [39680/60000 (66%)]	Loss: 0.124431
Train Epoch: 1 [39712/60000 (66%)]	Loss: 0.059272
Train Epoch: 1 [39744/60000 (66%)]	Loss: 0.208208
Train Epoch: 1 [39776/60000 (66%)]	Loss: 0.185901
Train Epoch: 1 [39808/60000 (66%)]	Loss: 0.096616
Train Epoch: 1 [39840/60000 (66%)]	Loss: 0.185221
Train Epoch: 1 [39872/60000 (66%)]	Loss: 0.113680
Train Epoch: 1 [39904/60000 (67%)]	Loss: 0.106692
Train Epoch: 1 [39936/60000 (67%)]	Loss: 0.333734
Train Epoch: 1 [39968/60000 (67%)]	Loss: 0.145872
Train Epoch: 1 [40000/60000 (67%)]	Loss: 0.089756
Train Epoch: 1 [40032/60000 (67%)]	Loss: 0.168120

Train Epoch: 1 [40064/60000 (67%)]	Loss: 0.117004
Train Epoch: 1 [40096/60000 (67%)]	Loss: 0.101927
Train Epoch: 1 [40128/60000 (67%)]	Loss: 0.206077
Train Epoch: 1 [40160/60000 (67%)]	Loss: 0.091891
Train Epoch: 1 [40192/60000 (67%)]	Loss: 0.060623
Train Epoch: 1 [40224/60000 (67%)]	Loss: 0.096493
Train Epoch: 1 [40256/60000 (67%)]	Loss: 0.119033
Train Epoch: 1 [40288/60000 (67%)]	Loss: 0.243768
Train Epoch: 1 [40320/60000 (67%)]	Loss: 0.028551
Train Epoch: 1 [40352/60000 (67%)]	Loss: 0.286006
Train Epoch: 1 [40384/60000 (67%)]	Loss: 0.119308
Train Epoch: 1 [40416/60000 (67%)]	Loss: 0.085582
Train Epoch: 1 [40448/60000 (67%)]	Loss: 0.100279
Train Epoch: 1 [40480/60000 (67%)]	Loss: 0.078620
Train Epoch: 1 [40512/60000 (68%)]	Loss: 0.109221
Train Epoch: 1 [40544/60000 (68%)]	Loss: 0.188035
Train Epoch: 1 [40576/60000 (68%)]	Loss: 0.143263
Train Epoch: 1 [40608/60000 (68%)]	Loss: 0.126353
Train Epoch: 1 [40640/60000 (68%)]	Loss: 0.217777
Train Epoch: 1 [40672/60000 (68%)]	Loss: 0.014850
Train Epoch: 1 [40704/60000 (68%)]	Loss: 0.094800
Train Epoch: 1 [40736/60000 (68%)]	Loss: 0.192484
Train Epoch: 1 [40768/60000 (68%)]	Loss: 0.052724
Train Epoch: 1 [40800/60000 (68%)]	Loss: 0.135375
Train Epoch: 1 [40832/60000 (68%)]	Loss: 0.034607
Train Epoch: 1 [40864/60000 (68%)]	Loss: 0.173368
Train Epoch: 1 [40896/60000 (68%)]	Loss: 0.010831
Train Epoch: 1 [40928/60000 (68%)]	Loss: 0.077189
Train Epoch: 1 [40960/60000 (68%)]	Loss: 0.321630
Train Epoch: 1 [40992/60000 (68%)]	Loss: 0.143879
Train Epoch: 1 [41024/60000 (68%)]	Loss: 0.060668
Train Epoch: 1 [41056/60000 (68%)]	Loss: 0.121688
Train Epoch: 1 [41088/60000 (68%)]	Loss: 0.176207
Train Epoch: 1 [41120/60000 (69%)]	Loss: 0.027332
Train Epoch: 1 [41152/60000 (69%)]	Loss: 0.073894
Train Epoch: 1 [41184/60000 (69%)]	Loss: 0.141418
Train Epoch: 1 [41216/60000 (69%)]	Loss: 0.073023
Train Epoch: 1 [41248/60000 (69%)]	Loss: 0.299266
Train Epoch: 1 [41280/60000 (69%)]	Loss: 0.226226
Train Epoch: 1 [41312/60000 (69%)]	Loss: 0.166658
Train Epoch: 1 [41344/60000 (69%)]	Loss: 0.078045
Train Epoch: 1 [41376/60000 (69%)]	Loss: 0.100042
Train Epoch: 1 [41408/60000 (69%)]	Loss: 0.158952
Train Epoch: 1 [41440/60000 (69%)]	Loss: 0.306853
Train Epoch: 1 [41472/60000 (69%)]	Loss: 0.097912
Train Epoch: 1 [41504/60000 (69%)]	Loss: 0.141075
Train Epoch: 1 [41536/60000 (69%)]	Loss: 0.171872
Train Epoch: 1 [41568/60000 (69%)]	Loss: 0.169721
Train Epoch: 1 [41600/60000 (69%)]	Loss: 0.156872
Train Epoch: 1 [41632/60000 (69%)]	Loss: 0.027606
Train Epoch: 1 [41664/60000 (69%)]	Loss: 0.047715

Train Epoch: 1 [41696/60000 (69%)]	Loss: 0.091078
Train Epoch: 1 [41728/60000 (70%)]	Loss: 0.111425
Train Epoch: 1 [41760/60000 (70%)]	Loss: 0.121648
Train Epoch: 1 [41792/60000 (70%)]	Loss: 0.087906
Train Epoch: 1 [41824/60000 (70%)]	Loss: 0.109796
Train Epoch: 1 [41856/60000 (70%)]	Loss: 0.050834
Train Epoch: 1 [41888/60000 (70%)]	Loss: 0.168663
Train Epoch: 1 [41920/60000 (70%)]	Loss: 0.181469
Train Epoch: 1 [41952/60000 (70%)]	Loss: 0.060966
Train Epoch: 1 [41984/60000 (70%)]	Loss: 0.279310
Train Epoch: 1 [42016/60000 (70%)]	Loss: 0.262304
Train Epoch: 1 [42048/60000 (70%)]	Loss: 0.082531
Train Epoch: 1 [42080/60000 (70%)]	Loss: 0.035229
Train Epoch: 1 [42112/60000 (70%)]	Loss: 0.160474
Train Epoch: 1 [42144/60000 (70%)]	Loss: 0.084649
Train Epoch: 1 [42176/60000 (70%)]	Loss: 0.133412
Train Epoch: 1 [42208/60000 (70%)]	Loss: 0.087630
Train Epoch: 1 [42240/60000 (70%)]	Loss: 0.066320
Train Epoch: 1 [42272/60000 (70%)]	Loss: 0.028919
Train Epoch: 1 [42304/60000 (71%)]	Loss: 0.100348
Train Epoch: 1 [42336/60000 (71%)]	Loss: 0.210061
Train Epoch: 1 [42368/60000 (71%)]	Loss: 0.081307
Train Epoch: 1 [42400/60000 (71%)]	Loss: 0.116033
Train Epoch: 1 [42432/60000 (71%)]	Loss: 0.321655
Train Epoch: 1 [42464/60000 (71%)]	Loss: 0.419802
Train Epoch: 1 [42496/60000 (71%)]	Loss: 0.586405
Train Epoch: 1 [42528/60000 (71%)]	Loss: 0.131263
Train Epoch: 1 [42560/60000 (71%)]	Loss: 0.201706
Train Epoch: 1 [42592/60000 (71%)]	Loss: 0.060886
Train Epoch: 1 [42624/60000 (71%)]	Loss: 0.044844
Train Epoch: 1 [42656/60000 (71%)]	Loss: 0.174515
Train Epoch: 1 [42688/60000 (71%)]	Loss: 0.207136
Train Epoch: 1 [42720/60000 (71%)]	Loss: 0.107354
Train Epoch: 1 [42752/60000 (71%)]	Loss: 0.081278
Train Epoch: 1 [42784/60000 (71%)]	Loss: 0.068594
Train Epoch: 1 [42816/60000 (71%)]	Loss: 0.201531
Train Epoch: 1 [42848/60000 (71%)]	Loss: 0.329888
Train Epoch: 1 [42880/60000 (71%)]	Loss: 0.133159
Train Epoch: 1 [42912/60000 (72%)]	Loss: 0.142410
Train Epoch: 1 [42944/60000 (72%)]	Loss: 0.225872
Train Epoch: 1 [42976/60000 (72%)]	Loss: 0.211184
Train Epoch: 1 [43008/60000 (72%)]	Loss: 0.065111
Train Epoch: 1 [43040/60000 (72%)]	Loss: 0.122561
Train Epoch: 1 [43072/60000 (72%)]	Loss: 0.145442
Train Epoch: 1 [43104/60000 (72%)]	Loss: 0.234005
Train Epoch: 1 [43136/60000 (72%)]	Loss: 0.087418
Train Epoch: 1 [43168/60000 (72%)]	Loss: 0.106706
Train Epoch: 1 [43200/60000 (72%)]	Loss: 0.235541
Train Epoch: 1 [43232/60000 (72%)]	Loss: 0.095314
Train Epoch: 1 [43264/60000 (72%)]	Loss: 0.051520
Train Epoch: 1 [43296/60000 (72%)]	Loss: 0.013228



Train Epoch: 1 [43328/60000 (72%)]	Loss: 0.112435
Train Epoch: 1 [43360/60000 (72%)]	Loss: 0.092680
Train Epoch: 1 [43392/60000 (72%)]	Loss: 0.047463
Train Epoch: 1 [43424/60000 (72%)]	Loss: 0.230135
Train Epoch: 1 [43456/60000 (72%)]	Loss: 0.083150
Train Epoch: 1 [43488/60000 (72%)]	Loss: 0.085729
Train Epoch: 1 [43520/60000 (73%)]	Loss: 0.090734
Train Epoch: 1 [43552/60000 (73%)]	Loss: 0.222298
Train Epoch: 1 [43584/60000 (73%)]	Loss: 0.038096
Train Epoch: 1 [43616/60000 (73%)]	Loss: 0.028492
Train Epoch: 1 [43648/60000 (73%)]	Loss: 0.292946
Train Epoch: 1 [43680/60000 (73%)]	Loss: 0.080436
Train Epoch: 1 [43712/60000 (73%)]	Loss: 0.019798
Train Epoch: 1 [43744/60000 (73%)]	Loss: 0.028803
Train Epoch: 1 [43776/60000 (73%)]	Loss: 0.083049
Train Epoch: 1 [43808/60000 (73%)]	Loss: 0.145800
Train Epoch: 1 [43840/60000 (73%)]	Loss: 0.074605
Train Epoch: 1 [43872/60000 (73%)]	Loss: 0.270222
Train Epoch: 1 [43904/60000 (73%)]	Loss: 0.131119
Train Epoch: 1 [43936/60000 (73%)]	Loss: 0.291006
Train Epoch: 1 [43968/60000 (73%)]	Loss: 0.318314
Train Epoch: 1 [44000/60000 (73%)]	Loss: 0.089947
Train Epoch: 1 [44032/60000 (73%)]	Loss: 0.125740
Train Epoch: 1 [44064/60000 (73%)]	Loss: 0.100886
Train Epoch: 1 [44096/60000 (73%)]	Loss: 0.230714
Train Epoch: 1 [44128/60000 (74%)]	Loss: 0.058207
Train Epoch: 1 [44160/60000 (74%)]	Loss: 0.080065
Train Epoch: 1 [44192/60000 (74%)]	Loss: 0.048828
Train Epoch: 1 [44224/60000 (74%)]	Loss: 0.151495
Train Epoch: 1 [44256/60000 (74%)]	Loss: 0.091295
Train Epoch: 1 [44288/60000 (74%)]	Loss: 0.104027
Train Epoch: 1 [44320/60000 (74%)]	Loss: 0.157413
Train Epoch: 1 [44352/60000 (74%)]	Loss: 0.265189
Train Epoch: 1 [44384/60000 (74%)]	Loss: 0.040979
Train Epoch: 1 [44416/60000 (74%)]	Loss: 0.220794
Train Epoch: 1 [44448/60000 (74%)]	Loss: 0.308796
Train Epoch: 1 [44480/60000 (74%)]	Loss: 0.087022
Train Epoch: 1 [44512/60000 (74%)]	Loss: 0.141084
Train Epoch: 1 [44544/60000 (74%)]	Loss: 0.090111
Train Epoch: 1 [44576/60000 (74%)]	Loss: 0.019933
Train Epoch: 1 [44608/60000 (74%)]	Loss: 0.083842
Train Epoch: 1 [44640/60000 (74%)]	Loss: 0.026902
Train Epoch: 1 [44672/60000 (74%)]	Loss: 0.033463
Train Epoch: 1 [44704/60000 (75%)]	Loss: 0.108504
Train Epoch: 1 [44736/60000 (75%)]	Loss: 0.307751
Train Epoch: 1 [44768/60000 (75%)]	Loss: 0.030259
Train Epoch: 1 [44800/60000 (75%)]	Loss: 0.089880
Train Epoch: 1 [44832/60000 (75%)]	Loss: 0.284851
Train Epoch: 1 [44864/60000 (75%)]	Loss: 0.187738
Train Epoch: 1 [44896/60000 (75%)]	Loss: 0.294181
Train Epoch: 1 [44928/60000 (75%)]	Loss: 0.051674

Train Epoch: 1 [44960/60000 (75%)]	Loss: 0.250865
Train Epoch: 1 [44992/60000 (75%)]	Loss: 0.160493
Train Epoch: 1 [45024/60000 (75%)]	Loss: 0.200153
Train Epoch: 1 [45056/60000 (75%)]	Loss: 0.090753
Train Epoch: 1 [45088/60000 (75%)]	Loss: 0.326508
Train Epoch: 1 [45120/60000 (75%)]	Loss: 0.240164
Train Epoch: 1 [45152/60000 (75%)]	Loss: 0.074984
Train Epoch: 1 [45184/60000 (75%)]	Loss: 0.185541
Train Epoch: 1 [45216/60000 (75%)]	Loss: 0.061406
Train Epoch: 1 [45248/60000 (75%)]	Loss: 0.111783
Train Epoch: 1 [45280/60000 (75%)]	Loss: 0.028420
Train Epoch: 1 [45312/60000 (76%)]	Loss: 0.022731
Train Epoch: 1 [45344/60000 (76%)]	Loss: 0.159868
Train Epoch: 1 [45376/60000 (76%)]	Loss: 0.079372
Train Epoch: 1 [45408/60000 (76%)]	Loss: 0.414155
Train Epoch: 1 [45440/60000 (76%)]	Loss: 0.254300
Train Epoch: 1 [45472/60000 (76%)]	Loss: 0.191329
Train Epoch: 1 [45504/60000 (76%)]	Loss: 0.262689
Train Epoch: 1 [45536/60000 (76%)]	Loss: 0.032757
Train Epoch: 1 [45568/60000 (76%)]	Loss: 0.107942
Train Epoch: 1 [45600/60000 (76%)]	Loss: 0.314520
Train Epoch: 1 [45632/60000 (76%)]	Loss: 0.098416
Train Epoch: 1 [45664/60000 (76%)]	Loss: 0.046750
Train Epoch: 1 [45696/60000 (76%)]	Loss: 0.028660
Train Epoch: 1 [45728/60000 (76%)]	Loss: 0.125824
Train Epoch: 1 [45760/60000 (76%)]	Loss: 0.236686
Train Epoch: 1 [45792/60000 (76%)]	Loss: 0.470084
Train Epoch: 1 [45824/60000 (76%)]	Loss: 0.079667
Train Epoch: 1 [45856/60000 (76%)]	Loss: 0.245235
Train Epoch: 1 [45888/60000 (76%)]	Loss: 0.254641
Train Epoch: 1 [45920/60000 (77%)]	Loss: 0.051044
Train Epoch: 1 [45952/60000 (77%)]	Loss: 0.370848
Train Epoch: 1 [45984/60000 (77%)]	Loss: 0.069367
Train Epoch: 1 [46016/60000 (77%)]	Loss: 0.169135
Train Epoch: 1 [46048/60000 (77%)]	Loss: 0.109917
Train Epoch: 1 [46080/60000 (77%)]	Loss: 0.291612
Train Epoch: 1 [46112/60000 (77%)]	Loss: 0.228657
Train Epoch: 1 [46144/60000 (77%)]	Loss: 0.115333
Train Epoch: 1 [46176/60000 (77%)]	Loss: 0.158650
Train Epoch: 1 [46208/60000 (77%)]	Loss: 0.072369
Train Epoch: 1 [46240/60000 (77%)]	Loss: 0.540321
Train Epoch: 1 [46272/60000 (77%)]	Loss: 0.479250
Train Epoch: 1 [46304/60000 (77%)]	Loss: 0.156283
Train Epoch: 1 [46336/60000 (77%)]	Loss: 0.194696
Train Epoch: 1 [46368/60000 (77%)]	Loss: 0.121448
Train Epoch: 1 [46400/60000 (77%)]	Loss: 0.249532
Train Epoch: 1 [46432/60000 (77%)]	Loss: 0.237363
Train Epoch: 1 [46464/60000 (77%)]	Loss: 0.045187
Train Epoch: 1 [46496/60000 (77%)]	Loss: 0.019343
Train Epoch: 1 [46528/60000 (78%)]	Loss: 0.023379
Train Epoch: 1 [46560/60000 (78%)]	Loss: 0.076869

Train Epoch: 1 [46592/60000 (78%)]	Loss: 0.115787
Train Epoch: 1 [46624/60000 (78%)]	Loss: 0.060859
Train Epoch: 1 [46656/60000 (78%)]	Loss: 0.042619
Train Epoch: 1 [46688/60000 (78%)]	Loss: 0.063583
Train Epoch: 1 [46720/60000 (78%)]	Loss: 0.385890
Train Epoch: 1 [46752/60000 (78%)]	Loss: 0.083741
Train Epoch: 1 [46784/60000 (78%)]	Loss: 0.081714
Train Epoch: 1 [46816/60000 (78%)]	Loss: 0.036230
Train Epoch: 1 [46848/60000 (78%)]	Loss: 0.227078
Train Epoch: 1 [46880/60000 (78%)]	Loss: 0.053137
Train Epoch: 1 [46912/60000 (78%)]	Loss: 0.069402
Train Epoch: 1 [46944/60000 (78%)]	Loss: 0.080849
Train Epoch: 1 [46976/60000 (78%)]	Loss: 0.088297
Train Epoch: 1 [47008/60000 (78%)]	Loss: 0.194310
Train Epoch: 1 [47040/60000 (78%)]	Loss: 0.053700
Train Epoch: 1 [47072/60000 (78%)]	Loss: 0.093053
Train Epoch: 1 [47104/60000 (79%)]	Loss: 0.072190
Train Epoch: 1 [47136/60000 (79%)]	Loss: 0.126330
Train Epoch: 1 [47168/60000 (79%)]	Loss: 0.015261
Train Epoch: 1 [47200/60000 (79%)]	Loss: 0.180929
Train Epoch: 1 [47232/60000 (79%)]	Loss: 0.277970
Train Epoch: 1 [47264/60000 (79%)]	Loss: 0.246771
Train Epoch: 1 [47296/60000 (79%)]	Loss: 0.088302
Train Epoch: 1 [47328/60000 (79%)]	Loss: 0.198457
Train Epoch: 1 [47360/60000 (79%)]	Loss: 0.145024
Train Epoch: 1 [47392/60000 (79%)]	Loss: 0.098604
Train Epoch: 1 [47424/60000 (79%)]	Loss: 0.146855
Train Epoch: 1 [47456/60000 (79%)]	Loss: 0.389142
Train Epoch: 1 [47488/60000 (79%)]	Loss: 0.123207
Train Epoch: 1 [47520/60000 (79%)]	Loss: 0.089829
Train Epoch: 1 [47552/60000 (79%)]	Loss: 0.155079
Train Epoch: 1 [47584/60000 (79%)]	Loss: 0.289078
Train Epoch: 1 [47616/60000 (79%)]	Loss: 0.073408
Train Epoch: 1 [47648/60000 (79%)]	Loss: 0.168230
Train Epoch: 1 [47680/60000 (79%)]	Loss: 0.292751
Train Epoch: 1 [47712/60000 (80%)]	Loss: 0.040309
Train Epoch: 1 [47744/60000 (80%)]	Loss: 0.214860
Train Epoch: 1 [47776/60000 (80%)]	Loss: 0.072726
Train Epoch: 1 [47808/60000 (80%)]	Loss: 0.055312
Train Epoch: 1 [47840/60000 (80%)]	Loss: 0.174753
Train Epoch: 1 [47872/60000 (80%)]	Loss: 0.078119
Train Epoch: 1 [47904/60000 (80%)]	Loss: 0.296280
Train Epoch: 1 [47936/60000 (80%)]	Loss: 0.283938
Train Epoch: 1 [47968/60000 (80%)]	Loss: 0.042554
Train Epoch: 1 [48000/60000 (80%)]	Loss: 0.102536
Train Epoch: 1 [48032/60000 (80%)]	Loss: 0.061606
Train Epoch: 1 [48064/60000 (80%)]	Loss: 0.112648
Train Epoch: 1 [48096/60000 (80%)]	Loss: 0.221756
Train Epoch: 1 [48128/60000 (80%)]	Loss: 0.076831
Train Epoch: 1 [48160/60000 (80%)]	Loss: 0.034705
Train Epoch: 1 [48192/60000 (80%)]	Loss: 0.039623

Train Epoch: 1 [48224/60000 (80%)]	Loss: 0.073098
Train Epoch: 1 [48256/60000 (80%)]	Loss: 0.195598
Train Epoch: 1 [48288/60000 (80%)]	Loss: 0.091668
Train Epoch: 1 [48320/60000 (81%)]	Loss: 0.166415
Train Epoch: 1 [48352/60000 (81%)]	Loss: 0.281681
Train Epoch: 1 [48384/60000 (81%)]	Loss: 0.147673
Train Epoch: 1 [48416/60000 (81%)]	Loss: 0.005704
Train Epoch: 1 [48448/60000 (81%)]	Loss: 0.015992
Train Epoch: 1 [48480/60000 (81%)]	Loss: 0.050980
Train Epoch: 1 [48512/60000 (81%)]	Loss: 0.178562
Train Epoch: 1 [48544/60000 (81%)]	Loss: 0.029129
Train Epoch: 1 [48576/60000 (81%)]	Loss: 0.098527
Train Epoch: 1 [48608/60000 (81%)]	Loss: 0.041085
Train Epoch: 1 [48640/60000 (81%)]	Loss: 0.274520
Train Epoch: 1 [48672/60000 (81%)]	Loss: 0.022088
Train Epoch: 1 [48704/60000 (81%)]	Loss: 0.085044
Train Epoch: 1 [48736/60000 (81%)]	Loss: 0.032542
Train Epoch: 1 [48768/60000 (81%)]	Loss: 0.051869
Train Epoch: 1 [48800/60000 (81%)]	Loss: 0.068551
Train Epoch: 1 [48832/60000 (81%)]	Loss: 0.058538
Train Epoch: 1 [48864/60000 (81%)]	Loss: 0.029075
Train Epoch: 1 [48896/60000 (81%)]	Loss: 0.091232
Train Epoch: 1 [48928/60000 (82%)]	Loss: 0.511188
Train Epoch: 1 [48960/60000 (82%)]	Loss: 0.306199
Train Epoch: 1 [48992/60000 (82%)]	Loss: 0.309768
Train Epoch: 1 [49024/60000 (82%)]	Loss: 0.093794
Train Epoch: 1 [49056/60000 (82%)]	Loss: 0.083408
Train Epoch: 1 [49088/60000 (82%)]	Loss: 0.095658
Train Epoch: 1 [49120/60000 (82%)]	Loss: 0.067401
Train Epoch: 1 [49152/60000 (82%)]	Loss: 0.027217
Train Epoch: 1 [49184/60000 (82%)]	Loss: 0.265231
Train Epoch: 1 [49216/60000 (82%)]	Loss: 0.126969
Train Epoch: 1 [49248/60000 (82%)]	Loss: 0.107114
Train Epoch: 1 [49280/60000 (82%)]	Loss: 0.067601
Train Epoch: 1 [49312/60000 (82%)]	Loss: 0.053585
Train Epoch: 1 [49344/60000 (82%)]	Loss: 0.042213
Train Epoch: 1 [49376/60000 (82%)]	Loss: 0.021005
Train Epoch: 1 [49408/60000 (82%)]	Loss: 0.116815
Train Epoch: 1 [49440/60000 (82%)]	Loss: 0.118311
Train Epoch: 1 [49472/60000 (82%)]	Loss: 0.444378
Train Epoch: 1 [49504/60000 (83%)]	Loss: 0.296403
Train Epoch: 1 [49536/60000 (83%)]	Loss: 0.544299
Train Epoch: 1 [49568/60000 (83%)]	Loss: 0.281183
Train Epoch: 1 [49600/60000 (83%)]	Loss: 0.146123
Train Epoch: 1 [49632/60000 (83%)]	Loss: 0.324566
Train Epoch: 1 [49664/60000 (83%)]	Loss: 0.116002
Train Epoch: 1 [49696/60000 (83%)]	Loss: 0.087722
Train Epoch: 1 [49728/60000 (83%)]	Loss: 0.118708
Train Epoch: 1 [49760/60000 (83%)]	Loss: 0.137159
Train Epoch: 1 [49792/60000 (83%)]	Loss: 0.061992
Train Epoch: 1 [49824/60000 (83%)]	Loss: 0.140918

Train Epoch: 1 [49856/60000 (83%)]	Loss: 0.075442
Train Epoch: 1 [49888/60000 (83%)]	Loss: 0.412003
Train Epoch: 1 [49920/60000 (83%)]	Loss: 0.095491
Train Epoch: 1 [49952/60000 (83%)]	Loss: 0.063785
Train Epoch: 1 [49984/60000 (83%)]	Loss: 0.105284
Train Epoch: 1 [50016/60000 (83%)]	Loss: 0.040940
Train Epoch: 1 [50048/60000 (83%)]	Loss: 0.160493
Train Epoch: 1 [50080/60000 (83%)]	Loss: 0.095286
Train Epoch: 1 [50112/60000 (84%)]	Loss: 0.135581
Train Epoch: 1 [50144/60000 (84%)]	Loss: 0.102524
Train Epoch: 1 [50176/60000 (84%)]	Loss: 0.074281
Train Epoch: 1 [50208/60000 (84%)]	Loss: 0.502544
Train Epoch: 1 [50240/60000 (84%)]	Loss: 0.142823
Train Epoch: 1 [50272/60000 (84%)]	Loss: 0.075768
Train Epoch: 1 [50304/60000 (84%)]	Loss: 0.141938
Train Epoch: 1 [50336/60000 (84%)]	Loss: 0.283496
Train Epoch: 1 [50368/60000 (84%)]	Loss: 0.380897
Train Epoch: 1 [50400/60000 (84%)]	Loss: 0.535483
Train Epoch: 1 [50432/60000 (84%)]	Loss: 0.148400
Train Epoch: 1 [50464/60000 (84%)]	Loss: 0.067824
Train Epoch: 1 [50496/60000 (84%)]	Loss: 0.346515
Train Epoch: 1 [50528/60000 (84%)]	Loss: 0.061201
Train Epoch: 1 [50560/60000 (84%)]	Loss: 0.229072
Train Epoch: 1 [50592/60000 (84%)]	Loss: 0.099557
Train Epoch: 1 [50624/60000 (84%)]	Loss: 0.347283
Train Epoch: 1 [50656/60000 (84%)]	Loss: 0.016518
Train Epoch: 1 [50688/60000 (84%)]	Loss: 0.185297
Train Epoch: 1 [50720/60000 (85%)]	Loss: 0.051805
Train Epoch: 1 [50752/60000 (85%)]	Loss: 0.150648
Train Epoch: 1 [50784/60000 (85%)]	Loss: 0.085557
Train Epoch: 1 [50816/60000 (85%)]	Loss: 0.103542
Train Epoch: 1 [50848/60000 (85%)]	Loss: 0.032275
Train Epoch: 1 [50880/60000 (85%)]	Loss: 0.171036
Train Epoch: 1 [50912/60000 (85%)]	Loss: 0.025809
Train Epoch: 1 [50944/60000 (85%)]	Loss: 0.032932
Train Epoch: 1 [50976/60000 (85%)]	Loss: 0.081804
Train Epoch: 1 [51008/60000 (85%)]	Loss: 0.041736
Train Epoch: 1 [51040/60000 (85%)]	Loss: 0.131616
Train Epoch: 1 [51072/60000 (85%)]	Loss: 0.073546
Train Epoch: 1 [51104/60000 (85%)]	Loss: 0.057703
Train Epoch: 1 [51136/60000 (85%)]	Loss: 0.289963
Train Epoch: 1 [51168/60000 (85%)]	Loss: 0.168982
Train Epoch: 1 [51200/60000 (85%)]	Loss: 0.167309
Train Epoch: 1 [51232/60000 (85%)]	Loss: 0.225256
Train Epoch: 1 [51264/60000 (85%)]	Loss: 0.236583
Train Epoch: 1 [51296/60000 (85%)]	Loss: 0.129674
Train Epoch: 1 [51328/60000 (86%)]	Loss: 0.133237
Train Epoch: 1 [51360/60000 (86%)]	Loss: 0.031187
Train Epoch: 1 [51392/60000 (86%)]	Loss: 0.019308
Train Epoch: 1 [51424/60000 (86%)]	Loss: 0.052511
Train Epoch: 1 [51456/60000 (86%)]	Loss: 0.073735

Train Epoch: 1 [51488/60000 (86%)]	Loss: 0.015489
Train Epoch: 1 [51520/60000 (86%)]	Loss: 0.267856
Train Epoch: 1 [51552/60000 (86%)]	Loss: 0.038852
Train Epoch: 1 [51584/60000 (86%)]	Loss: 0.126199
Train Epoch: 1 [51616/60000 (86%)]	Loss: 0.071650
Train Epoch: 1 [51648/60000 (86%)]	Loss: 0.098670
Train Epoch: 1 [51680/60000 (86%)]	Loss: 0.185203
Train Epoch: 1 [51712/60000 (86%)]	Loss: 0.094256
Train Epoch: 1 [51744/60000 (86%)]	Loss: 0.217367
Train Epoch: 1 [51776/60000 (86%)]	Loss: 0.048464
Train Epoch: 1 [51808/60000 (86%)]	Loss: 0.015049
Train Epoch: 1 [51840/60000 (86%)]	Loss: 0.074502
Train Epoch: 1 [51872/60000 (86%)]	Loss: 0.091387
Train Epoch: 1 [51904/60000 (87%)]	Loss: 0.126407
Train Epoch: 1 [51936/60000 (87%)]	Loss: 0.216228
Train Epoch: 1 [51968/60000 (87%)]	Loss: 0.430640
Train Epoch: 1 [52000/60000 (87%)]	Loss: 0.121389
Train Epoch: 1 [52032/60000 (87%)]	Loss: 0.046362
Train Epoch: 1 [52064/60000 (87%)]	Loss: 0.342444
Train Epoch: 1 [52096/60000 (87%)]	Loss: 0.105627
Train Epoch: 1 [52128/60000 (87%)]	Loss: 0.513947
Train Epoch: 1 [52160/60000 (87%)]	Loss: 0.123229
Train Epoch: 1 [52192/60000 (87%)]	Loss: 0.121566
Train Epoch: 1 [52224/60000 (87%)]	Loss: 0.177595
Train Epoch: 1 [52256/60000 (87%)]	Loss: 0.095384
Train Epoch: 1 [52288/60000 (87%)]	Loss: 0.082176
Train Epoch: 1 [52320/60000 (87%)]	Loss: 0.166243
Train Epoch: 1 [52352/60000 (87%)]	Loss: 0.141731
Train Epoch: 1 [52384/60000 (87%)]	Loss: 0.064210
Train Epoch: 1 [52416/60000 (87%)]	Loss: 0.070132
Train Epoch: 1 [52448/60000 (87%)]	Loss: 0.086391
Train Epoch: 1 [52480/60000 (87%)]	Loss: 0.022124
Train Epoch: 1 [52512/60000 (88%)]	Loss: 0.021207
Train Epoch: 1 [52544/60000 (88%)]	Loss: 0.019920
Train Epoch: 1 [52576/60000 (88%)]	Loss: 0.038426
Train Epoch: 1 [52608/60000 (88%)]	Loss: 0.042958
Train Epoch: 1 [52640/60000 (88%)]	Loss: 0.131199
Train Epoch: 1 [52672/60000 (88%)]	Loss: 0.223806
Train Epoch: 1 [52704/60000 (88%)]	Loss: 0.240760
Train Epoch: 1 [52736/60000 (88%)]	Loss: 0.192560
Train Epoch: 1 [52768/60000 (88%)]	Loss: 0.059294
Train Epoch: 1 [52800/60000 (88%)]	Loss: 0.232359
Train Epoch: 1 [52832/60000 (88%)]	Loss: 0.090846
Train Epoch: 1 [52864/60000 (88%)]	Loss: 0.115852
Train Epoch: 1 [52896/60000 (88%)]	Loss: 0.345059
Train Epoch: 1 [52928/60000 (88%)]	Loss: 0.458079
Train Epoch: 1 [52960/60000 (88%)]	Loss: 0.423353
Train Epoch: 1 [52992/60000 (88%)]	Loss: 0.090902
Train Epoch: 1 [53024/60000 (88%)]	Loss: 0.120995
Train Epoch: 1 [53056/60000 (88%)]	Loss: 0.045684
Train Epoch: 1 [53088/60000 (88%)]	Loss: 0.078648

Train Epoch: 1 [53120/60000 (89%)]	Loss: 0.049585
Train Epoch: 1 [53152/60000 (89%)]	Loss: 0.394276
Train Epoch: 1 [53184/60000 (89%)]	Loss: 0.135362
Train Epoch: 1 [53216/60000 (89%)]	Loss: 0.238982
Train Epoch: 1 [53248/60000 (89%)]	Loss: 0.019231
Train Epoch: 1 [53280/60000 (89%)]	Loss: 0.011516
Train Epoch: 1 [53312/60000 (89%)]	Loss: 0.076304
Train Epoch: 1 [53344/60000 (89%)]	Loss: 0.021351
Train Epoch: 1 [53376/60000 (89%)]	Loss: 0.118556
Train Epoch: 1 [53408/60000 (89%)]	Loss: 0.007448
Train Epoch: 1 [53440/60000 (89%)]	Loss: 0.084748
Train Epoch: 1 [53472/60000 (89%)]	Loss: 0.234186
Train Epoch: 1 [53504/60000 (89%)]	Loss: 0.094708
Train Epoch: 1 [53536/60000 (89%)]	Loss: 0.153525
Train Epoch: 1 [53568/60000 (89%)]	Loss: 0.286278
Train Epoch: 1 [53600/60000 (89%)]	Loss: 0.075334
Train Epoch: 1 [53632/60000 (89%)]	Loss: 0.194178
Train Epoch: 1 [53664/60000 (89%)]	Loss: 0.130249
Train Epoch: 1 [53696/60000 (89%)]	Loss: 0.036903
Train Epoch: 1 [53728/60000 (90%)]	Loss: 0.127884
Train Epoch: 1 [53760/60000 (90%)]	Loss: 0.016326
Train Epoch: 1 [53792/60000 (90%)]	Loss: 0.108567
Train Epoch: 1 [53824/60000 (90%)]	Loss: 0.190327
Train Epoch: 1 [53856/60000 (90%)]	Loss: 0.305833
Train Epoch: 1 [53888/60000 (90%)]	Loss: 0.211938
Train Epoch: 1 [53920/60000 (90%)]	Loss: 0.025191
Train Epoch: 1 [53952/60000 (90%)]	Loss: 0.091421
Train Epoch: 1 [53984/60000 (90%)]	Loss: 0.223957
Train Epoch: 1 [54016/60000 (90%)]	Loss: 0.060511
Train Epoch: 1 [54048/60000 (90%)]	Loss: 0.089902
Train Epoch: 1 [54080/60000 (90%)]	Loss: 0.240981
Train Epoch: 1 [54112/60000 (90%)]	Loss: 0.021067
Train Epoch: 1 [54144/60000 (90%)]	Loss: 0.062322
Train Epoch: 1 [54176/60000 (90%)]	Loss: 0.179766
Train Epoch: 1 [54208/60000 (90%)]	Loss: 0.014015
Train Epoch: 1 [54240/60000 (90%)]	Loss: 0.062849
Train Epoch: 1 [54272/60000 (90%)]	Loss: 0.078552
Train Epoch: 1 [54304/60000 (91%)]	Loss: 0.090620
Train Epoch: 1 [54336/60000 (91%)]	Loss: 0.019826
Train Epoch: 1 [54368/60000 (91%)]	Loss: 0.234527
Train Epoch: 1 [54400/60000 (91%)]	Loss: 0.075489
Train Epoch: 1 [54432/60000 (91%)]	Loss: 0.122838
Train Epoch: 1 [54464/60000 (91%)]	Loss: 0.032527
Train Epoch: 1 [54496/60000 (91%)]	Loss: 0.305654
Train Epoch: 1 [54528/60000 (91%)]	Loss: 0.125937
Train Epoch: 1 [54560/60000 (91%)]	Loss: 0.110018
Train Epoch: 1 [54592/60000 (91%)]	Loss: 0.025760
Train Epoch: 1 [54624/60000 (91%)]	Loss: 0.078422
Train Epoch: 1 [54656/60000 (91%)]	Loss: 0.087428
Train Epoch: 1 [54688/60000 (91%)]	Loss: 0.099418
Train Epoch: 1 [54720/60000 (91%)]	Loss: 0.056109

Train Epoch: 1 [54752/60000 (91%)]	Loss: 0.142697
Train Epoch: 1 [54784/60000 (91%)]	Loss: 0.139602
Train Epoch: 1 [54816/60000 (91%)]	Loss: 0.101257
Train Epoch: 1 [54848/60000 (91%)]	Loss: 0.153780
Train Epoch: 1 [54880/60000 (91%)]	Loss: 0.225870
Train Epoch: 1 [54912/60000 (92%)]	Loss: 0.146682
Train Epoch: 1 [54944/60000 (92%)]	Loss: 0.133555
Train Epoch: 1 [54976/60000 (92%)]	Loss: 0.078900
Train Epoch: 1 [55008/60000 (92%)]	Loss: 0.103713
Train Epoch: 1 [55040/60000 (92%)]	Loss: 0.145299
Train Epoch: 1 [55072/60000 (92%)]	Loss: 0.015040
Train Epoch: 1 [55104/60000 (92%)]	Loss: 0.063056
Train Epoch: 1 [55136/60000 (92%)]	Loss: 0.020167
Train Epoch: 1 [55168/60000 (92%)]	Loss: 0.095948
Train Epoch: 1 [55200/60000 (92%)]	Loss: 0.083206
Train Epoch: 1 [55232/60000 (92%)]	Loss: 0.036686
Train Epoch: 1 [55264/60000 (92%)]	Loss: 0.135531
Train Epoch: 1 [55296/60000 (92%)]	Loss: 0.107257
Train Epoch: 1 [55328/60000 (92%)]	Loss: 0.317193
Train Epoch: 1 [55360/60000 (92%)]	Loss: 0.041333
Train Epoch: 1 [55392/60000 (92%)]	Loss: 0.038497
Train Epoch: 1 [55424/60000 (92%)]	Loss: 0.160182
Train Epoch: 1 [55456/60000 (92%)]	Loss: 0.081803
Train Epoch: 1 [55488/60000 (92%)]	Loss: 0.172017
Train Epoch: 1 [55520/60000 (93%)]	Loss: 0.169902
Train Epoch: 1 [55552/60000 (93%)]	Loss: 0.043880
Train Epoch: 1 [55584/60000 (93%)]	Loss: 0.170220
Train Epoch: 1 [55616/60000 (93%)]	Loss: 0.030655
Train Epoch: 1 [55648/60000 (93%)]	Loss: 0.072032
Train Epoch: 1 [55680/60000 (93%)]	Loss: 0.019340
Train Epoch: 1 [55712/60000 (93%)]	Loss: 0.314014
Train Epoch: 1 [55744/60000 (93%)]	Loss: 0.031961
Train Epoch: 1 [55776/60000 (93%)]	Loss: 0.190121
Train Epoch: 1 [55808/60000 (93%)]	Loss: 0.140166
Train Epoch: 1 [55840/60000 (93%)]	Loss: 0.144252
Train Epoch: 1 [55872/60000 (93%)]	Loss: 0.113253
Train Epoch: 1 [55904/60000 (93%)]	Loss: 0.056865
Train Epoch: 1 [55936/60000 (93%)]	Loss: 0.047335
Train Epoch: 1 [55968/60000 (93%)]	Loss: 0.046658
Train Epoch: 1 [56000/60000 (93%)]	Loss: 0.082043
Train Epoch: 1 [56032/60000 (93%)]	Loss: 0.020829
Train Epoch: 1 [56064/60000 (93%)]	Loss: 0.160045
Train Epoch: 1 [56096/60000 (93%)]	Loss: 0.025142
Train Epoch: 1 [56128/60000 (94%)]	Loss: 0.088419
Train Epoch: 1 [56160/60000 (94%)]	Loss: 0.181278
Train Epoch: 1 [56192/60000 (94%)]	Loss: 0.146032
Train Epoch: 1 [56224/60000 (94%)]	Loss: 0.201759
Train Epoch: 1 [56256/60000 (94%)]	Loss: 0.165464
Train Epoch: 1 [56288/60000 (94%)]	Loss: 0.128347
Train Epoch: 1 [56320/60000 (94%)]	Loss: 0.072927
Train Epoch: 1 [56352/60000 (94%)]	Loss: 0.083642



Train Epoch: 1 [56384/60000 (94%)]	Loss: 0.105705
Train Epoch: 1 [56416/60000 (94%)]	Loss: 0.012661
Train Epoch: 1 [56448/60000 (94%)]	Loss: 0.361809
Train Epoch: 1 [56480/60000 (94%)]	Loss: 0.268915
Train Epoch: 1 [56512/60000 (94%)]	Loss: 0.034615
Train Epoch: 1 [56544/60000 (94%)]	Loss: 0.014314
Train Epoch: 1 [56576/60000 (94%)]	Loss: 0.125279
Train Epoch: 1 [56608/60000 (94%)]	Loss: 0.024190
Train Epoch: 1 [56640/60000 (94%)]	Loss: 0.241183
Train Epoch: 1 [56672/60000 (94%)]	Loss: 0.044230
Train Epoch: 1 [56704/60000 (95%)]	Loss: 0.043047
Train Epoch: 1 [56736/60000 (95%)]	Loss: 0.071070
Train Epoch: 1 [56768/60000 (95%)]	Loss: 0.086990
Train Epoch: 1 [56800/60000 (95%)]	Loss: 0.156655
Train Epoch: 1 [56832/60000 (95%)]	Loss: 0.144715
Train Epoch: 1 [56864/60000 (95%)]	Loss: 0.033659
Train Epoch: 1 [56896/60000 (95%)]	Loss: 0.074382
Train Epoch: 1 [56928/60000 (95%)]	Loss: 0.004407
Train Epoch: 1 [56960/60000 (95%)]	Loss: 0.013143
Train Epoch: 1 [56992/60000 (95%)]	Loss: 0.134437
Train Epoch: 1 [57024/60000 (95%)]	Loss: 0.153952
Train Epoch: 1 [57056/60000 (95%)]	Loss: 0.144690
Train Epoch: 1 [57088/60000 (95%)]	Loss: 0.035902
Train Epoch: 1 [57120/60000 (95%)]	Loss: 0.017743
Train Epoch: 1 [57152/60000 (95%)]	Loss: 0.066168
Train Epoch: 1 [57184/60000 (95%)]	Loss: 0.065662
Train Epoch: 1 [57216/60000 (95%)]	Loss: 0.090536
Train Epoch: 1 [57248/60000 (95%)]	Loss: 0.100338
Train Epoch: 1 [57280/60000 (95%)]	Loss: 0.133462
Train Epoch: 1 [57312/60000 (96%)]	Loss: 0.054276
Train Epoch: 1 [57344/60000 (96%)]	Loss: 0.012724
Train Epoch: 1 [57376/60000 (96%)]	Loss: 0.077731
Train Epoch: 1 [57408/60000 (96%)]	Loss: 0.042601
Train Epoch: 1 [57440/60000 (96%)]	Loss: 0.026199
Train Epoch: 1 [57472/60000 (96%)]	Loss: 0.067795
Train Epoch: 1 [57504/60000 (96%)]	Loss: 0.131908
Train Epoch: 1 [57536/60000 (96%)]	Loss: 0.051189
Train Epoch: 1 [57568/60000 (96%)]	Loss: 0.035879
Train Epoch: 1 [57600/60000 (96%)]	Loss: 0.150324
Train Epoch: 1 [57632/60000 (96%)]	Loss: 0.118210
Train Epoch: 1 [57664/60000 (96%)]	Loss: 0.102978
Train Epoch: 1 [57696/60000 (96%)]	Loss: 0.146087
Train Epoch: 1 [57728/60000 (96%)]	Loss: 0.349397
Train Epoch: 1 [57760/60000 (96%)]	Loss: 0.254003
Train Epoch: 1 [57792/60000 (96%)]	Loss: 0.028538
Train Epoch: 1 [57824/60000 (96%)]	Loss: 0.015535
Train Epoch: 1 [57856/60000 (96%)]	Loss: 0.141704
Train Epoch: 1 [57888/60000 (96%)]	Loss: 0.043405
Train Epoch: 1 [57920/60000 (97%)]	Loss: 0.015563
Train Epoch: 1 [57952/60000 (97%)]	Loss: 0.109384
Train Epoch: 1 [57984/60000 (97%)]	Loss: 0.065912

Train Epoch: 1 [58016/60000 (97%)]	Loss: 0.029101
Train Epoch: 1 [58048/60000 (97%)]	Loss: 0.102594
Train Epoch: 1 [58080/60000 (97%)]	Loss: 0.091162
Train Epoch: 1 [58112/60000 (97%)]	Loss: 0.012016
Train Epoch: 1 [58144/60000 (97%)]	Loss: 0.004158
Train Epoch: 1 [58176/60000 (97%)]	Loss: 0.017425
Train Epoch: 1 [58208/60000 (97%)]	Loss: 0.007588
Train Epoch: 1 [58240/60000 (97%)]	Loss: 0.116006
Train Epoch: 1 [58272/60000 (97%)]	Loss: 0.011334
Train Epoch: 1 [58304/60000 (97%)]	Loss: 0.036827
Train Epoch: 1 [58336/60000 (97%)]	Loss: 0.035747
Train Epoch: 1 [58368/60000 (97%)]	Loss: 0.126501
Train Epoch: 1 [58400/60000 (97%)]	Loss: 0.039656
Train Epoch: 1 [58432/60000 (97%)]	Loss: 0.009310
Train Epoch: 1 [58464/60000 (97%)]	Loss: 0.064086
Train Epoch: 1 [58496/60000 (97%)]	Loss: 0.007966
Train Epoch: 1 [58528/60000 (98%)]	Loss: 0.013777
Train Epoch: 1 [58560/60000 (98%)]	Loss: 0.006616
Train Epoch: 1 [58592/60000 (98%)]	Loss: 0.044859
Train Epoch: 1 [58624/60000 (98%)]	Loss: 0.074704
Train Epoch: 1 [58656/60000 (98%)]	Loss: 0.030530
Train Epoch: 1 [58688/60000 (98%)]	Loss: 0.028037
Train Epoch: 1 [58720/60000 (98%)]	Loss: 0.005328
Train Epoch: 1 [58752/60000 (98%)]	Loss: 0.014556
Train Epoch: 1 [58784/60000 (98%)]	Loss: 0.166993
Train Epoch: 1 [58816/60000 (98%)]	Loss: 0.232269
Train Epoch: 1 [58848/60000 (98%)]	Loss: 0.023619
Train Epoch: 1 [58880/60000 (98%)]	Loss: 0.012596
Train Epoch: 1 [58912/60000 (98%)]	Loss: 0.004414
Train Epoch: 1 [58944/60000 (98%)]	Loss: 0.004098
Train Epoch: 1 [58976/60000 (98%)]	Loss: 0.004197
Train Epoch: 1 [59008/60000 (98%)]	Loss: 0.003310
Train Epoch: 1 [59040/60000 (98%)]	Loss: 0.013133
Train Epoch: 1 [59072/60000 (98%)]	Loss: 0.013788
Train Epoch: 1 [59104/60000 (99%)]	Loss: 0.013862
Train Epoch: 1 [59136/60000 (99%)]	Loss: 0.004828
Train Epoch: 1 [59168/60000 (99%)]	Loss: 0.004543
Train Epoch: 1 [59200/60000 (99%)]	Loss: 0.042834
Train Epoch: 1 [59232/60000 (99%)]	Loss: 0.019034
Train Epoch: 1 [59264/60000 (99%)]	Loss: 0.159751
Train Epoch: 1 [59296/60000 (99%)]	Loss: 0.107439
Train Epoch: 1 [59328/60000 (99%)]	Loss: 0.062891
Train Epoch: 1 [59360/60000 (99%)]	Loss: 0.078799
Train Epoch: 1 [59392/60000 (99%)]	Loss: 0.190603
Train Epoch: 1 [59424/60000 (99%)]	Loss: 0.059825
Train Epoch: 1 [59456/60000 (99%)]	Loss: 0.230520
Train Epoch: 1 [59488/60000 (99%)]	Loss: 0.002267
Train Epoch: 1 [59520/60000 (99%)]	Loss: 0.004152
Train Epoch: 1 [59552/60000 (99%)]	Loss: 0.006593
Train Epoch: 1 [59584/60000 (99%)]	Loss: 0.015351
Train Epoch: 1 [59616/60000 (99%)]	Loss: 0.021916

```

Train Epoch: 1 [59648/60000 (99%)] Loss: 0.185691
Train Epoch: 1 [59680/60000 (99%)] Loss: 0.224874
Train Epoch: 1 [59712/60000 (100%)] Loss: 0.938763
Train Epoch: 1 [59744/60000 (100%)] Loss: 0.263601
Train Epoch: 1 [59776/60000 (100%)] Loss: 0.063714
Train Epoch: 1 [59808/60000 (100%)] Loss: 0.032220
Train Epoch: 1 [59840/60000 (100%)] Loss: 0.019377
Train Epoch: 1 [59872/60000 (100%)] Loss: 0.015368
Train Epoch: 1 [59904/60000 (100%)] Loss: 0.706151
Train Epoch: 1 [59936/60000 (100%)] Loss: 0.068000
Train Epoch: 1 [59968/60000 (100%)] Loss: 0.039564

```

## Konversi ke OpenVINO IR

Kemudian, konversi model ke format OpenVINO IR

```

core = ov.Core()
example_input = next(iter(test_loader))[0]
ov_model = ov.convert_model(model, example_input=example_input)
ov.save_model(ov_model, MODEL_DIR / f"openvino_ir.xml")

```

No CUDA runtime is found, using CUDA\_HOME='/usr/local/cuda'

## Kuantisasi

Untuk mengkuantisasi model menggunakan NNCF, pertama-tama, buat fungsi transformasi untuk mengonversi tensor torch ke array NumPy, lalu gunakan fungsi yang dibuat bersama dengan pemuat data pytorch untuk membuat set data kalibrasi menggunakan kelas Dataset dari NNCF. Selanjutnya, kuantisasi model menggunakan fungsi quantize dari NNCF. Terakhir, kompilasi model yang dikuantisasi dan simpan sebagai format OpenVINO IR.

```

def transform_fn(data_item):
    images, _ = data_item
    return images.numpy()

```

```

calibration_dataset = nncf.Dataset(train_loader, transform_fn)
quantized_model = nncf.quantize(ov_model, calibration_dataset)
model_int8 = ov.compile_model(quantized_model)
input_fp32 = next(iter(test_loader))[0][0:1]
res = model_int8(input_fp32)
ov.save_model(quantized_model, MODEL_DIR / f"quant_openvino_ir.xml")

```

```

{"model_id":"9733e6bdbabe4540976f16f8925e6580","version_major":2,"version_minor":0}

```

```

{"model_id":"40e80ed2a7c545a4b2ff962d731279c7","version_major":2,"version_minor":0}

```

## Periksa Ukuran

Bandingkan ukuran model FP32 dan INT8

```
%ls -lh {MODEL_DIR}
```

```
total 176K
```

```
-rw-r--r-- 1 root root 40K Jan  4 06:17 openvino_ir.bin
-rw-r--r-- 1 root root 11K Jan  4 06:17 openvino_ir.xml
-rw-r--r-- 1 root root 82K Jan  4 06:17 original_model.p
-rw-r--r-- 1 root root 21K Jan  4 06:17 quant_openvino_ir.bin
-rw-r--r-- 1 root root 16K Jan  4 06:17 quant_openvino_ir.xml
```

## Periksa Akurasi

Evaluasi akurasi model INT8 dan bandingkan dengan model FP32

```
def test_ov(model, data_loader):
    compiled_model = ov.compile_model(model)
    test_loss = 0
    correct = 0
    for data, target in data_loader:
        output = torch.tensor(compiled_model(data)[0])
        test_loss += F.nll_loss(output, target, reduction='sum').item() # jumlahkan loss batch
        pred = output.argmax(dim=1, keepdim=True) # dapatkan indeks probabilitas log maksimum
        correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(data_loader.dataset)

    return 100. * correct / len(data_loader.dataset)

# Menghitung akurasi model asli
acc = test_ov(ov_model, test_loader)
print(f"Akurasi model asli: {acc}")

# Menghitung akurasi model yang telah dikuantisasi
qacc = test_ov(quantized_model, test_loader)
print(f"Akurasi model yang telah dikuantisasi: {qacc}")
```

Akurasi model asli: 96.37

Akurasi model yang telah dikuantisasi: 96.76

## Menggunakan DCGAN untuk Menghasilkan Gambar Paru-paru Sintetis

- DCGAN diperkenalkan dalam makalah Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks oleh Radford, dkk.

Dalam notebook ini kita akan:

- Mengunduh gambar X-ray paru-paru dari Hugging Face Hub
- Memproses gambar

- Menggunakan DCGAN untuk menghasilkan gambar sintetis
- Memvisualisasikan proses pelatihan
- Notebook demo ini diadaptasi dari [sini](#) dan [sini](#). Notebook ini dijalankan selama beberapa jam menggunakan High-RAM.

## Tentang Dataset

*Ringkasan yang diambil dari [Identifying Medical Diagnoses and Treatable Diseases by Image-Based Deep Learning](#):*

- Dataset ini disusun dalam 3 folder (train, test, val) dan berisi subfolder untuk setiap kategori gambar (Pneumonia/Normal). Terdapat 5.863 gambar X-ray (JPEG) dan 2 kategori (Pneumonia/Normal).
- Gambar X-ray dada (anterior-posterior) dipilih dari kohor retrospektif pasien anak-anak berusia satu hingga lima tahun dari Guangzhou Women and Children's Medical Center, Guangzhou. Semua pencitraan X-ray dada dilakukan sebagai bagian dari perawatan klinis rutin pasien.
- Untuk analisis gambar X-ray dada, semua radiografi dada awalnya disaring untuk kontrol kualitas dengan menghapus semua hasil scan yang berkualitas rendah atau tidak terbaca. Diagnosa untuk gambar tersebut kemudian dinilai oleh dua dokter ahli sebelum disetujui untuk melatih sistem AI. Untuk mengantisipasi kesalahan penilaian, set evaluasi juga diperiksa oleh seorang ahli ketiga.

## Sumber Dataset

### Pertama, kita akan menginstal dependensinya

```
!pip install -q datasets huggingface_hub datasets torch torchvision accelerate
```

```

— 0.0/521.2 kB ? eta -:::--
143.4/521.2 kB 4.1 MB/s eta 0:00:01
— 512.0/521.2 kB 8.8 MB/s eta 0:00:01
— 521.2/521.2
kB 6.0 MB/s eta 0:00:00
— 265.7/265.7 kB 10.1 MB/s eta 0:00:00
— 115.3/115.3 kB 7.4 MB/s eta 0:00:00
— 134.8/134.8 kB 10.1 MB/s eta 0:00:00

```

### Selanjutnya, kita akan mengunduh dataset dari Hugging Face Hub.

```
from datasets import load_dataset # Mengimpor fungsi load_dataset dari pustaka 'datasets'
```

```
# Memuat dataset gambar dada yang digunakan untuk mendeteksi pneumonia pada X-ray
```

```
dataset = load_dataset("hf-vision/chest-xray-pneumonia") # Memuat dataset 'chest-xray-pneumonia' dari Hugging Face
```

**Sebelum menjalankan model, ada baiknya untuk sedikit menjelajahi kumpulan data.**

Kita akan mulai dengan mencari panjang kumpulan data.

```
len(dataset['train'])
```

5216

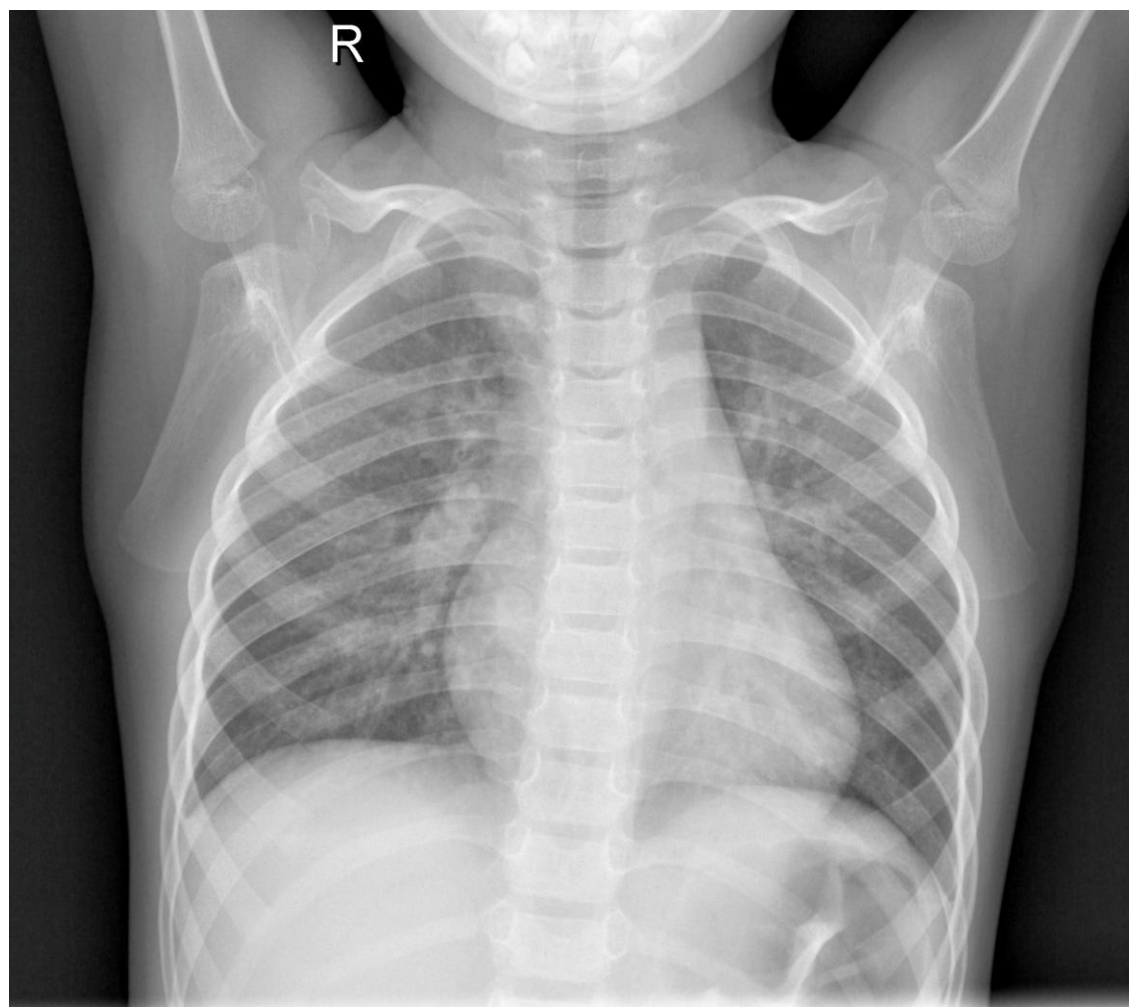
Mari kita visualisasikan 4 gambar pertama dari kumpulan data, yang diberi label "normal".

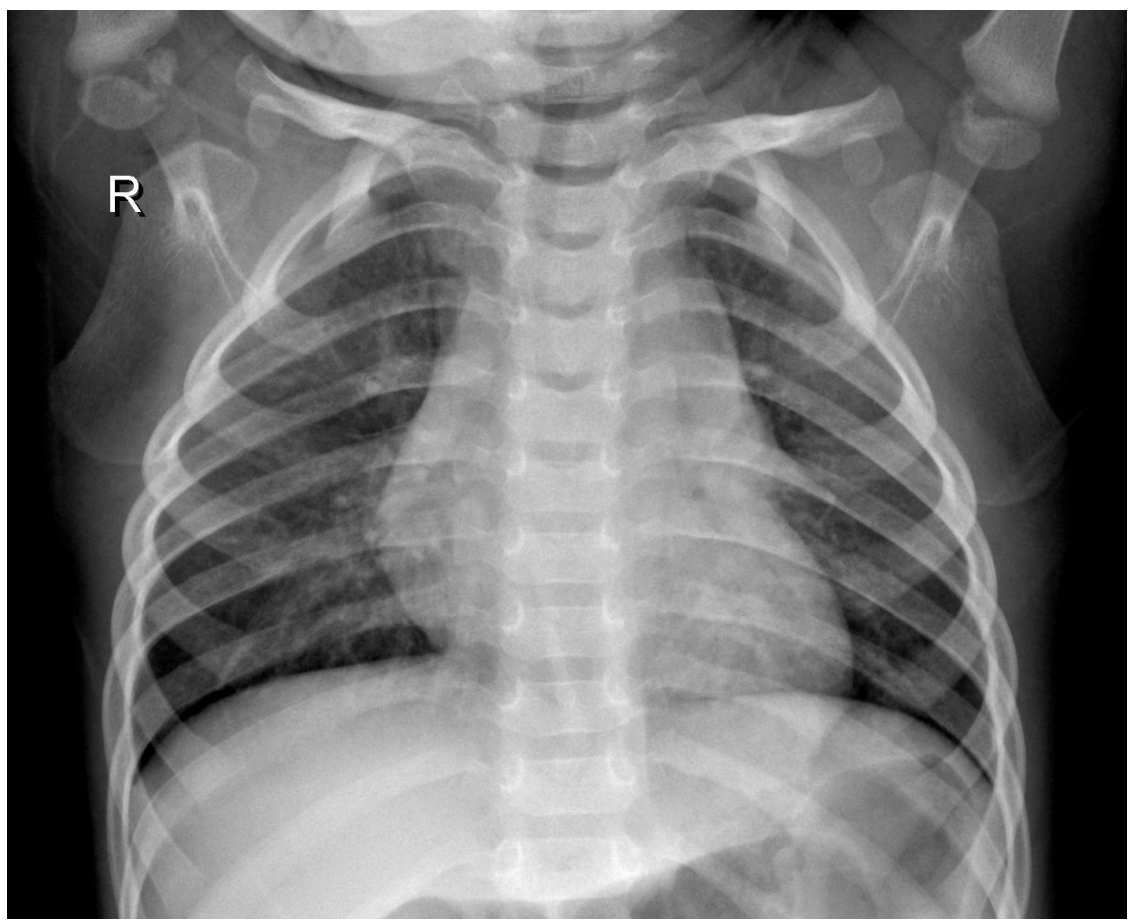
```
from PIL import Image # Mengimpor modul Image dari pustaka PIL untuk membuka dan memanipulasi gambar
```

```
from IPython.display import display # Mengimpor fungsi display untuk menampilkan gambar di notebook
```

```
# Menampilkan gambar pertama pada dataset pelatihan (train)
```

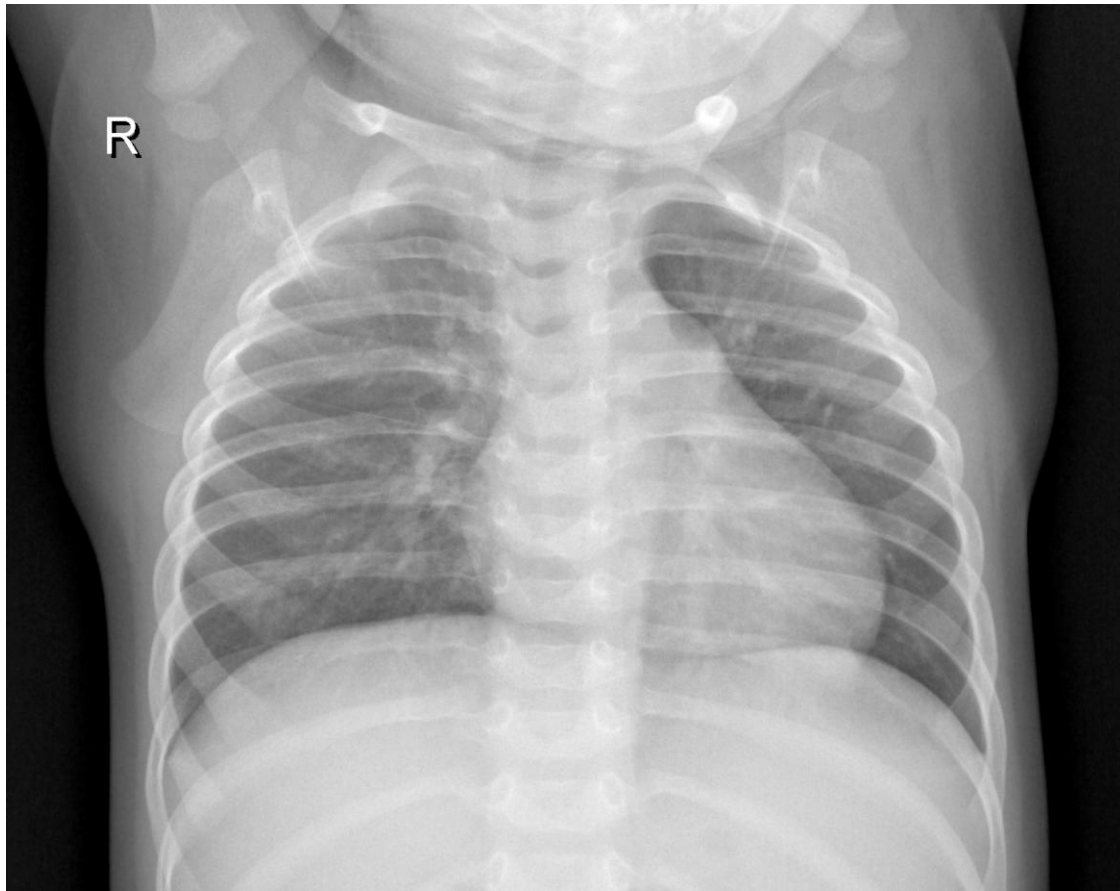
```
for i in range(4): # Loop untuk menampilkan 4 gambar pertama dalam dataset pelatihan  
    display(dataset["train"][i]['image']) # Mengambil gambar ke-i dari split "train" dan menampilkannya
```





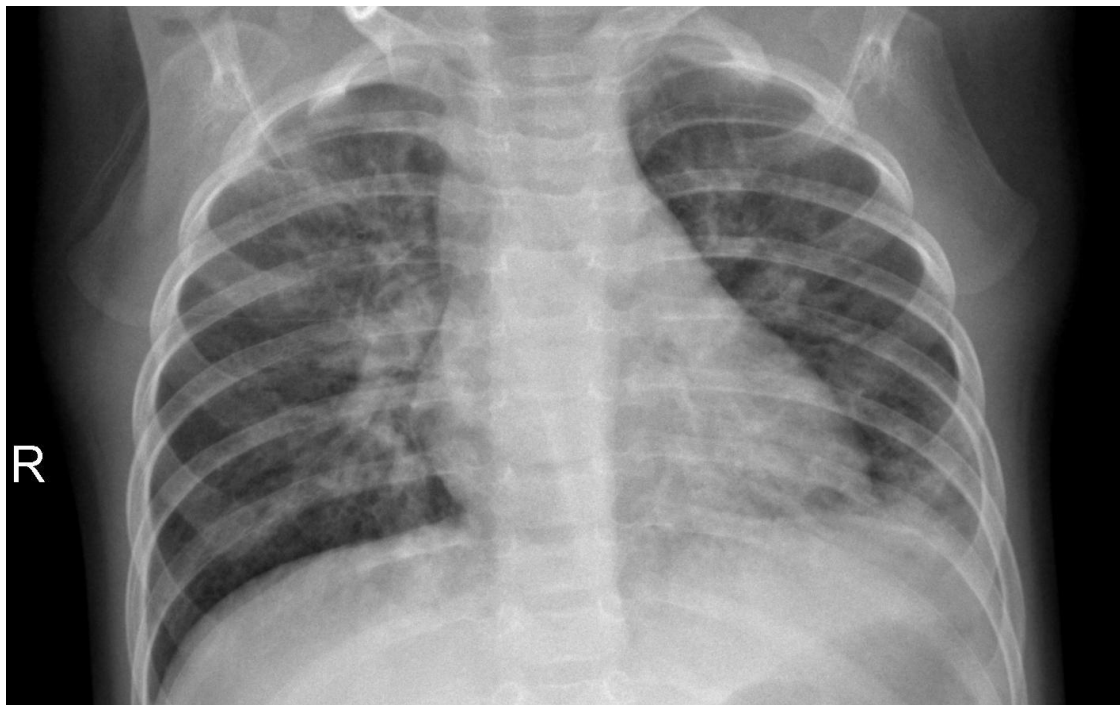






Sekarang mari kita lihat 4 gambar terakhir dalam kumpulan data, yang diberi label "pneumonia."

```
from PIL import Image # Mengimpor modul Image dari pustaka PIL untuk membuka dan  
manipulasi gambar  
from IPython.display import display # Mengimpor fungsi display untuk menampilkan gambar  
dalam notebook Jupyter  
  
# Menampilkan gambar dari indeks 5211 hingga 5214 pada dataset pelatihan (train)  
for i in range(5211, 5215): # Loop untuk menampilkan gambar dengan indeks 5211 hingga  
5214  
    display(dataset["train"][i]["image"]) # Mengambil gambar ke-i dari split "train" dan  
menampilkannya
```





**Kami akan mengimpor pustaka yang diperlukan**

**from** IPython.display **import** HTML *# Untuk menampilkan HTML dalam notebook Jupyter*

**import** matplotlib.animation **as** animation *# Digunakan untuk membuat animasi dengan matplotlib*

**import** matplotlib.pyplot **as** plt *# Pustaka untuk visualisasi data (grafik) di Python*

```
import math # Modul matematika untuk fungsi matematika dasar
import numpy as np # Pustaka untuk operasi array dan komputasi numerik (termasuk aljabar linier)
import os # Modul untuk berinteraksi dengan sistem file dan direktori
import random # Modul untuk operasi acak (misalnya, memilih elemen acak)
import torch # Pustaka utama untuk operasi tensor dan komputasi di PyTorch
import torch.nn as nn # Modul untuk mendefinisikan model jaringan saraf (neural network) di PyTorch
import torch.optim as optim # Modul untuk optimasi dalam pelatihan jaringan saraf
```

```
from PIL import Image # Pustaka untuk manipulasi gambar, digunakan untuk membuka dan menyimpan gambar
import torchvision.transforms as transforms # Modul untuk menerapkan transformasi gambar dalam dataset
from torchvision.transforms import CenterCrop, Compose, Normalize, Resize, ToTensor # Fungsi transformasi untuk pra-pemrosesan gambar
import torchvision.utils as utils # Pustaka utilitas untuk bekerja dengan gambar dalam batch
from torch.utils.data import DataLoader # Modul untuk memuat data dalam batch selama pelatihan
```

### **Sekarang kita akan melakukan praproses data**

- Kita akan menstandarisasi gambar menggunakan Resize dan CenterCrop.
- Kita akan mengubah gambar menjadi tensor.
- Kita akan menormalkan data:
- Rata-rata (0,5, 0,5, 0,5) memastikan bahwa data gambar kira-kira berpusat di sekitar nol. Ini dapat membantu mencegah masalah yang terkait dengan gradien yang menghilang atau meledak selama pelatihan.
- Deviasi standar ke (0,5, 0,5, 0,5), Anda menskalakan data input agar berada dalam rentang [-1, 1]. Rentang ini sering kali lebih disukai untuk jaringan generator di GAN

```
import math
import matplotlib.pyplot as plt
import numpy as np
import os
import random
import torch
import torch.nn as nn
import torch.optim as optim
from PIL import Image
import torchvision.transforms as transforms
from torchvision.transforms import CenterCrop, Compose, Normalize, Resize, ToTensor
import torchvision.utils as utils
from torch.utils.data import DataLoader
from datasets import load_dataset
```

```

# Menentukan parameter dan pengaturan awal
batch_size = 128 # Ukuran batch untuk pelatihan
nb_gpu = 1 # Jumlah GPU yang tersedia
nb_workers = 2 # Jumlah pekerja untuk memuat data (sesuaikan dengan sumber daya sistem)

# Tentukan perangkat yang akan digunakan (GPU atau CPU)
device = torch.device("cuda:0" if (torch.cuda.is_available() and nb_gpu > 0) else "cpu")

# Parameter input dan ukuran gambar
n_channels = 3 # Jumlah saluran gambar (RGB)
image_size = 64 # Ukuran gambar yang akan digunakan
input_shape = (image_size, image_size) # Bentuk input gambar

# Setel seed untuk reproduksibilitas
seed = 22
random.seed(seed) # Setel seed untuk random library
torch.manual_seed(seed) # Setel seed untuk PyTorch
torch.backends.cudnn.deterministic = True # Memastikan hasil yang deterministik pada GPU
torch.backends.cudnn.benchmark = False # Menonaktifkan pencarian algoritma yang lebih cepat (untuk konsistensi)

# Definisikan transformasi gambar yang akan diterapkan
transform = Compose(
    [
        Resize(image_size), # Mengubah ukuran gambar menjadi 64x64
        CenterCrop(image_size), # Memotong gambar di bagian tengah
        ToTensor(), # Mengonversi gambar menjadi tensor PyTorch
        Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)), # Menormalkan gambar untuk setiap saluran
    ]
)

# Fungsi untuk menerapkan transformasi pada dataset
def transforms(examples):
    examples["image"] = [transform(image.convert("RGB")) for image in examples["image"]]
    return examples

# Memuat dataset dan menerapkan transformasi
dataset = load_dataset("hf-vision/chest-xray-pneumonia") # Menggunakan dataset X-ray pneumonia
transformed_dataset = dataset.with_transform(transforms)

# Membuat DataLoader untuk batch data selama pelatihan
dataloader = DataLoader(
    transformed_dataset["train"], batch_size=batch_size, shuffle=True, num_workers=nb_workers
)

```

```
# Mengambil satu batch data dari DataLoader  
real_batch = next(iter(dataloader))
```

```
# Menampilkan gambar pertama dalam batch yang diambil  
plt.imshow(real_batch["image"][0].permute(1, 2, 0)) # Mengonversi tensor ke format yang bisa  
ditampilkan  
plt.axis("off") # Menonaktifkan axis untuk visualisasi gambar  
plt.show()
```

```
# Informasi perangkat dan dimensi batch  
print(f"Device: {device}")  
print(f"Input shape: {input_shape}")  
print(f"Batch size: {batch_size}")  
print(f"Number of GPU(s) available: {nb_gpu}")
```

### **Sekarang kita akan membuat model**

Pertama, kita tentukan beberapa parameter.

```
# Ukuran vektor laten z (yaitu ukuran input generator), sama seperti yang dijelaskan dalam  
makalah DCGAN  
nz = 100 # Ukuran vektor input (ruang laten) untuk generator. Ini biasanya adalah vektor  
berdimensi 100 yang diambil dari distribusi normal yang digunakan sebagai input untuk  
generator.
```

```
# Ukuran peta fitur pada generator  
ngf = 64 # Jumlah filter (peta fitur) di lapisan pertama generator. Meningkatkan angka ini  
meningkatkan kompleksitas jaringan generator.
```

```
# Ukuran peta fitur pada diskriminator  
ndf = 64 # Jumlah filter (peta fitur) di lapisan pertama diskriminator. Nilai yang lebih tinggi  
meningkatkan kapasitas diskriminator.
```

```
# Jumlah epoch pelatihan  
num_epochs = 100 # Jumlah kali dataset lengkap akan digunakan selama pelatihan. Lebih  
banyak epoch dapat menghasilkan kinerja model yang lebih baik, tetapi juga dapat  
menyebabkan overfitting.
```

```
# Tingkat pembelajaran untuk optimizer, nilai yang sama seperti yang dijelaskan dalam makalah  
DCGAN  
lr = 0.0002 # Tingkat pembelajaran mengontrol seberapa besar langkah yang diambil oleh  
optimizer selama pelatihan. Nilai yang lebih kecil dapat memperlambat pelatihan, tetapi dapat  
membantu menghindari langkah yang terlalu besar.
```

```
# Hyperparameter Beta1 untuk optimizer Adam, nilai yang sama seperti yang dijelaskan dalam  
makalah DCGAN
```

$\beta_1 = 0.5$  *# Ini adalah istilah momentum pertama dalam optimizer Adam. Biasanya diatur ke 0.5 untuk GAN, karena telah terbukti membantu meningkatkan stabilitas pelatihan.*

Bobot model diinisialisasi secara acak dari distribusi Normal dengan  $\text{mean}=0$ ,  $\text{stdev}=0.02$ . Bobot ini diterapkan pada setiap lapisan Generator dan Diskriminator.

```
def weights_init(m):
    # Mendapatkan nama kelas dari objek 'm'
    classname = m.__class__.__name__

    # Jika kelasnya mengandung 'Conv' (menandakan layer konvolusi)
    if classname.find('Conv') != -1:
        # Inisialisasi bobot dengan distribusi normal, rata-rata 0.0 dan deviasi standar 0.02
        nn.init.normal_(m.weight.data, 0.0, 0.02)

    # Jika kelasnya mengandung 'BatchNorm' (menandakan lapisan normalisasi batch)
    elif classname.find('BatchNorm') != -1:
        # Inisialisasi bobot dengan distribusi normal, rata-rata 1.0 dan deviasi standar 0.02
        nn.init.normal_(m.weight.data, 1.0, 0.02)
        # Set bias menjadi 0
        nn.init.constant_(m.bias.data, 0)
```

### **Tentukan Generator**

*# Kode Generator*

```
class Generator(nn.Module):
    def __init__(self, nb_gpu):
        super(Generator, self).__init__()
        self.nb_gpu = nb_gpu

    # Menyusun layer-layer utama dari Generator menggunakan Sequential
    self.main = nn.Sequential(
        # input adalah Z (latent vector), masuk ke dalam layer konvolusi terbalik pertama
        # (ConvTranspose2d)

        nn.ConvTranspose2d(nz, ngf * 8, 4, 1, 0, bias=False), # Layer pertama: z menjadi 4x4
        # feature map
        nn.BatchNorm2d(ngf * 8), # Batch Normalization
        nn.ReLU(True), # Aktivasi ReLU

        # state size: (ngf*8) x 4 x 4

        nn.ConvTranspose2d(ngf * 8, ngf * 4, 4, 2, 1, bias=False), # Layer kedua: mengubah
        # ukuran menjadi 8x8
        nn.BatchNorm2d(ngf * 4),
        nn.ReLU(True),

        # state size: (ngf*4) x 8 x 8
```



```

        nn.ConvTranspose2d(ngf * 4, ngf * 2, 4, 2, 1, bias=False), # Layer ketiga: mengubah
        ukuran menjadi 16x16
        nn.BatchNorm2d(ngf * 2),
        nn.ReLU(True),

```

*# state size: (ngf\*2) x 16 x 16*

```

        nn.ConvTranspose2d(ngf * 2, ngf, 4, 2, 1, bias=False), # Layer keempat: mengubah
        ukuran menjadi 32x32
        nn.BatchNorm2d(ngf),
        nn.ReLU(True),

```

*# state size: (ngf) x 32 x 32*

```

        nn.ConvTranspose2d(ngf, n_channels, 4, 2, 1, bias=False), # Layer kelima:
        menghasilkan gambar akhir 64x64
        nn.Tanh() # Fungsi aktivasi Tanh untuk menghasilkan nilai antara -1 dan 1

```

*# state size: (n\_channels) x 64 x 64*

)

*# Fungsi forward untuk menghitung keluaran dari Generator*

```

def forward(self, input):
    return self.main(input)

```

## **Buat Generator**

*# Membuat instance dari Generator*

```

gen_net = Generator(nb_gpu).to(device)

```

*# Menangani multi-GPU jika diperlukan*

```

if (device.type == 'cuda') and (nb_gpu > 1):
    gen_net = nn.DataParallel(gen_net, list(range(nb_gpu)))

```

*# Menerapkan fungsi ``weights\_init`` untuk menginisialisasi bobot secara acak*

```

gen_net.apply(weights_init)

```

*# Mencetak model*

```

print(gen_net)

```

Generator(

(main): Sequential(

(0): ConvTranspose2d(100, 512, kernel\_size=(4, 4), stride=(1, 1), bias=False)

(1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(2): ReLU(inplace=True)

(3): ConvTranspose2d(512, 256, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)

(4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

```

(5): ReLU(inplace=True)
(6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
(7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(8): ReLU(inplace=True)
(9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
(10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(11): ReLU(inplace=True)
(12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
(13): Tanh()
)
)

```

### Definisikan Diskriminator

```
class Discriminator(nn.Module):
```

```

    def __init__(self, nb_gpu):
        super(Discriminator, self).__init__()
        self.nb_gpu = nb_gpu
        self.main = nn.Sequential(
            # input adalah ``(nb_channels) x 64 x 64``, dimensi input yang diterima oleh
discriminator

            # Layer konvolusi pertama dengan 1 saluran input, mengubahnya menjadi feature map
dengan 'ndf' saluran.
            nn.Conv2d(n_channels, ndf, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True), # Fungsi aktivasi LeakyReLU dengan alpha 0.2
            # ukuran state setelah layer ini menjadi ``(ndf) x 32 x 32``

            # Layer konvolusi kedua, menambah jumlah saluran feature map menjadi 'ndf*2'
            nn.Conv2d(ndf, ndf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 2), # Batch Normalization untuk stabilisasi pelatihan
            nn.LeakyReLU(0.2, inplace=True), # Fungsi aktivasi LeakyReLU
            # ukuran state setelah layer ini menjadi ``(ndf*2) x 16 x 16``

            # Layer konvolusi ketiga, meningkatkan jumlah saluran menjadi 'ndf*4'
            nn.Conv2d(ndf * 2, ndf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 4), # Batch Normalization untuk stabilisasi pelatihan
            nn.LeakyReLU(0.2, inplace=True), # Fungsi aktivasi LeakyReLU
            # ukuran state setelah layer ini menjadi ``(ndf*4) x 8 x 8``

            # Layer konvolusi keempat, meningkatkan jumlah saluran menjadi 'ndf*8'
            nn.Conv2d(ndf * 4, ndf * 8, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 8), # Batch Normalization untuk stabilisasi pelatihan
            nn.LeakyReLU(0.2, inplace=True), # Fungsi aktivasi LeakyReLU
            # ukuran state setelah layer ini menjadi ``(ndf*8) x 4 x 4``

            # Layer konvolusi terakhir yang menghasilkan output 1 saluran (nilai 0 atau 1)
            nn.Conv2d(ndf * 8, 1, 4, 1, 0, bias=False),

```

```

        nn.Sigmoid() # Fungsi aktivasi Sigmoid untuk menghasilkan nilai probabilitas antara 0
dan 1
    )

```

```

def forward(self, input):
    return self.main(input) # Mengarahkan input melalui semua layer yang telah didefinisikan

```

### **Ciptakan Diskriminator**

*# Membuat model Discriminator*

```
disc_net = Discriminator(nb_gpu).to(device)
```

*# Menangani multi-GPU jika diinginkan*

```
if (device.type == 'cuda') and (nb_gpu > 1):
```

*# Jika menggunakan lebih dari satu GPU, model akan dibagi untuk dijalankan di beberapa GPU*

```
    disc_net = nn.DataParallel(disc_net, list(range(nb_gpu)))
```

*# Menerapkan fungsi ``weights\_init`` untuk menginisialisasi bobot secara acak*

*# Fungsi ini akan menginisialisasi bobot dengan distribusi normal (mean=0, stdev=0.2)*

```
disc_net.apply(weights_init)
```

*# Mencetak arsitektur model*

```
print(disc_net)
```

Discriminator(

(main): Sequential(

(0): Conv2d(3, 64, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)

(1): LeakyReLU(negative\_slope=0.2, inplace=True)

(2): Conv2d(64, 128, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)

(3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(4): LeakyReLU(negative\_slope=0.2, inplace=True)

(5): Conv2d(128, 256, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)

(6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(7): LeakyReLU(negative\_slope=0.2, inplace=True)

(8): Conv2d(256, 512, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)

(9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(10): LeakyReLU(negative\_slope=0.2, inplace=True)

(11): Conv2d(512, 1, kernel\_size=(4, 4), stride=(1, 1), bias=False)

(12): Sigmoid()

)

)

### **Ada dua jenis kesalahan statistik:**

- Jenis 1 - Positif Palsu
- Jenis 2 - Negatif Palsu

Dalam kasus kita, diskriminator dapat membuat dua jenis kesalahan. Pertama, diskriminator dapat memberi label gambar asli sebagai "palsu". Kedua, diskriminator dapat memberi label gambar palsu sebagai "asli".

```
# Gambar asli dan discriminator berpikir itu asli - Keputusan yang benar
```

```
target = 1
```

```
output = 0.99
```

```
# Menghitung loss untuk keputusan yang benar (genuine image dan output mendekati 1)
```

```
# Ini adalah fungsi binary cross entropy
```

```
print(-(target * math.log(output) + (1 - target) * math.log(1 - output)))
```

```
# Gambar asli dan discriminator berpikir itu palsu - Keputusan yang salah
```

```
target = 1
```

```
output = 0.01
```

```
# Menghitung loss untuk keputusan yang salah (genuine image dan output mendekati 0)
```

```
# Output yang mendekati 0 berarti discriminator berpikir itu palsu, tetapi targetnya adalah 1 (genuine image)
```

```
print(-(target * math.log(output) + (1 - target) * math.log(1 - output)))
```

```
# Gambar palsu tetapi discriminator berpikir itu asli - Keputusan yang salah
```

```
target = 0
```

```
output = 0.99
```

```
# Menghitung loss untuk keputusan yang salah (fake image dan output mendekati 1)
```

```
# Output yang mendekati 1 berarti discriminator berpikir gambar itu asli, tetapi targetnya adalah 0 (fake image)
```

```
print(-(target * math.log(output) + (1 - target) * math.log(1 - output)))
```

```
0.01005033585350145
```

```
4.605170185988091
```

```
4.605170185988091
```

**Kami akan menggunakan Binary Cross Entropy Loss sebagai fungsi kerugian kami.**

- BCELoss mengukur perbedaan antara probabilitas yang diprediksi dan nilai target aktual.

```
# Set real and fake label values (following GAN paper convention)
```

```
real_label = 1. # Label untuk gambar yang asli
```

```
fake_label = 0. # Label untuk gambar yang palsu
```

```
# Define loss function
```

```
criterion = nn.BCELoss() # Fungsi loss Binary Cross-Entropy
```

**Kami menyiapkan pengoptimal menggunakan parameter yang disediakan oleh makalah DCGAN**

```
optimizerD = optim.Adam(disc_net.parameters(), lr=lr, betas=(beta1, 0.999))
```

```
optimizerG = optim.Adam(gen_net.parameters(), lr=lr, betas=(beta1, 0.999))
```

## Pelatihan

**Kami melakukan inisialisasi dengan gangguan acak.**

```
fixed_noise = torch.randn(64, nz, 1, 1, device=device)
```

**Kami akan menyimpan titik pemeriksaan model ke beberapa folder. Anda juga dapat mengunggah model ke Hugging Face Hub.**

```
# Flags - Untuk setiap epoch
```

```
show_images = True # Menandakan apakah gambar yang dihasilkan akan ditampilkan setelah setiap epoch
```

```
save_images = True # Menandakan apakah gambar yang dihasilkan akan disimpan setelah setiap epoch
```

```
save_model = True # Menandakan apakah model Generator dan Discriminator akan disimpan setelah setiap epoch
```

```
# Fungsi untuk menyimpan model DCGAN
```

```
def save_dcgan(netG, netD, path_checkpoint):
```

```
    checkpoint = {"g_model_state_dict": gen_net.state_dict(), # Menyimpan bobot model Generator
```

```
                  "d_model_state_dict": disc_net.state_dict(), # Menyimpan bobot model Discriminator
    }
```

```
    torch.save(checkpoint, path_checkpoint) # Menyimpan checkpoint ke file dengan path yang ditentukan
```

```
# Fungsi untuk membuat direktori baru jika belum ada
```

```
def makedir(new_dir):
```

```
    if not os.path.exists(new_dir): # Memeriksa apakah direktori sudah ada
```

```
        os.makedirs(new_dir) # Jika tidak ada, buat direktori baru
```

```
# Membuat folder untuk menyimpan gambar dan model yang dilatih
```

```
makedir("images") # Membuat folder 'images' untuk menyimpan gambar yang dihasilkan
```

```
makedir("models") # Membuat folder 'models' untuk menyimpan model yang telah dilatih
```

## Mari buat training loop

**Untuk setiap epoch:**

- **Perbarui Discriminator:**

- Bersihkan gradien dari discriminator
- Ambil batch gambar nyata dari dataset
- Buat tensor untuk label nyata
- Lewatkan gambar nyata melalui discriminator dan ambil output
- Hitung loss discriminator untuk data nyata
- Lakukan backpropagation untuk gradien

- **Perbarui Generator:**
  - Bersihkan gradien dari generator
  - Isi tensor label dengan label nyata (berusaha untuk menipu discriminator)
  - Lewatkan data palsu yang dihasilkan melalui discriminator
  - Hitung loss generator berdasarkan respons discriminator
  - Lakukan backpropagation untuk gradien
- Perbarui parameter Generator dan Discriminator berdasarkan gradien
- Cetak statistik pelatihan
- Simpan loss untuk pemetaan
- Simpan model

```
# Load 9 images
nb_images = 9 # Jumlah gambar yang akan ditampilkan
nb_row = math.ceil(math.sqrt(nb_images)) # Menentukan jumlah baris untuk menampilkan gambar
```

```
# Training Loop
data_len = len(dataloader) # Panjang dataset untuk iterasi
```

```
# Daftar untuk melacak progress
img_list = [] # Menyimpan gambar yang dihasilkan oleh Generator
G_losses = [] # Menyimpan nilai loss untuk Generator
D_losses = [] # Menyimpan nilai loss untuk Discriminator
```

```
# Untuk setiap epoch
for epoch in range(num_epochs):
    # Untuk setiap batch dalam dataloader (tergantung pada batch_size dan jumlah gambar)
    for i, data in enumerate(dataloader, 0):
```

```
        # (1) Update jaringan D: memaksimalkan  $\log(D(x)) + \log(1 - D(G(z)))$ 
```

```
        ## Latih dengan batch asli
        disc_net.zero_grad() # Zero grad untuk Discriminator
        real_cpu = data['image'].to(device) # Ambil batch gambar asli
        b_size = real_cpu.size(0) # Ukuran batch
        label = torch.full((b_size,), real_label, dtype=torch.float, device=device) # Label untuk gambar asli (1)
        output = disc_net(real_cpu).view(-1) # Prediksi Discriminator untuk gambar asli
        errD_real = criterion(output, label) # Hitung loss Discriminator untuk gambar asli
        errD_real.backward() # Backprop untuk Discriminator
        D_x = output.mean().item() # Mean output untuk monitoring
```

```
        ## Latih dengan batch palsu
```

```

noise = torch.randn(b_size, nz, 1, 1, device=device) # Buat noise sebagai input Generator
fake = gen_net(noise) # Hasilkan gambar palsu dengan Generator
label.fill_(fake_label) # Ganti label menjadi 0 untuk gambar palsu
output = disc_net(fake.detach()).view(-1) # Prediksi Discriminator untuk gambar palsu
errD_fake = criterion(output, label) # Hitung loss Discriminator untuk gambar palsu
errD_fake.backward() # Backprop untuk Discriminator
D_G_z1 = output.mean().item() # Mean output untuk monitoring
errD = errD_real + errD_fake # Total loss Discriminator
optimizerD.step() # Update parameter Discriminator

# (2) Update jaringan G: memaksimalkan  $\log(D(G(z)))$ 

gen_net.zero_grad() # Zero grad untuk Generator
label.fill_(real_label) # Label palsu diubah menjadi 1 untuk cost Generator
output = disc_net(fake).view(-1) # Prediksi Discriminator untuk gambar palsu
errG = criterion(output, label) # Hitung loss Generator
errG.backward() # Backprop untuk Generator
D_G_z2 = output.mean().item() # Mean output untuk monitoring
optimizerG.step() # Update parameter Generator

# (3) Metrics & Evaluation

# Tampilkan statistik training setiap 50 batch (jika dataset besar, frekuensi print bisa
dikurangi)
if i % 50 == 0:
    print(f'Epoch: {epoch}/{num_epochs} Batches: {i}/{data_len} \t Loss_D:
{errD.item():.4f} Loss_G: {errG.item():.4f} D(x): {D_x:.4f} D(G(z)): {D_G_z1:.4f} /
{D_G_z2:.4f}')

    # Simpan Loss untuk plotting nanti
    G_losses.append(errG.item()) # Simpan loss Generator
    D_losses.append(errD.item()) # Simpan loss Discriminator

# Hasilkan gambar palsu untuk melihat bagaimana kinerja Generator pada fixed_noise setiap
epoch
if show_images == True and epoch % 10 == 0:
    with torch.no_grad():
        fake = gen_net(fixed_noise).detach().cpu() # Hasilkan gambar palsu dengan fixed_noise
        img_list.append(vutils.make_grid(fake[:nb_images], padding=2, normalize=True,
nrow=nb_row)) # Gabungkan gambar palsu menjadi grid

    plt.figure(figsize=(3, 3))
    plt.axis("off")
    plt.imshow(np.transpose(img_list[-1], (1, 2, 0))) # Tampilkan gambar

    if save_images == True:

```

```
plt.savefig(f'images/epoch_{epoch}_gen_images.png') # Simpan gambar yang
dihasilkan
```

```
plt.show() # Tampilkan gambar
```

```
# Simpan model setiap 5 epoch
```

```
if epoch % 5 == 0:
```

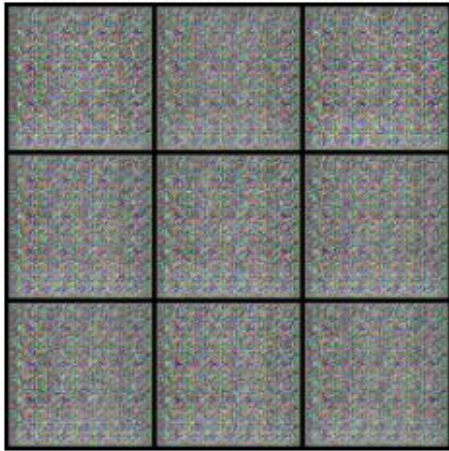
```
    if save_model:
```

```
        save_dcgan(gen_net, disc_net,
path_checkpoint=f'models/chest_epoch_{epoch}_checkpoint.pkl')
```

```
# Simpan model akhir setelah training selesai
```

```
save_dcgan(gen_net, disc_net, path_checkpoint="models/chest_final_epoch_checkpoint.pkl")
```

```
Epoch: 0/100 Batches: 0/41 Loss_D: 1.7490 Loss_G: 5.9989 D(x): 0.7465 D(G(z)):
0.7043 / 0.0043
```



```
Epoch: 1/100 Batches: 0/41 Loss_D: 0.0243 Loss_G: 24.2771 D(x): 0.9881 D(G(z)):
0.0000 / 0.0000
```

```
Epoch: 2/100 Batches: 0/41 Loss_D: 0.4267 Loss_G: 8.7650 D(x): 0.8978 D(G(z)):
0.1689 / 0.0005
```

```
Epoch: 3/100 Batches: 0/41 Loss_D: 0.4963 Loss_G: 5.8759 D(x): 0.7268 D(G(z)):
0.0264 / 0.0043
```

```
Epoch: 4/100 Batches: 0/41 Loss_D: 0.6237 Loss_G: 7.0209 D(x): 0.8969 D(G(z)):
0.3507 / 0.0027
```

```
Epoch: 5/100 Batches: 0/41 Loss_D: 0.4415 Loss_G: 5.2512 D(x): 0.9424 D(G(z)):
0.2803 / 0.0096
```

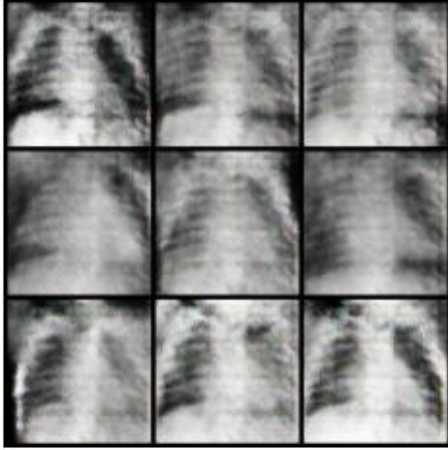
```
Epoch: 6/100 Batches: 0/41 Loss_D: 0.6055 Loss_G: 4.2842 D(x): 0.8455 D(G(z)):
0.2987 / 0.0326
```

```
Epoch: 7/100 Batches: 0/41 Loss_D: 0.6063 Loss_G: 3.2278 D(x): 0.7392 D(G(z)):
0.2032 / 0.0757
```

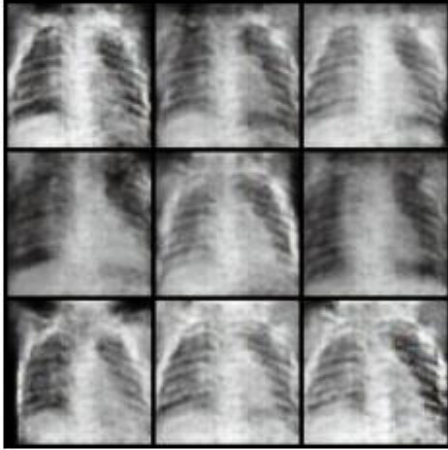
```
Epoch: 8/100 Batches: 0/41 Loss_D: 1.0650 Loss_G: 4.8959 D(x): 0.4922 D(G(z)):
0.0195 / 0.0198
```



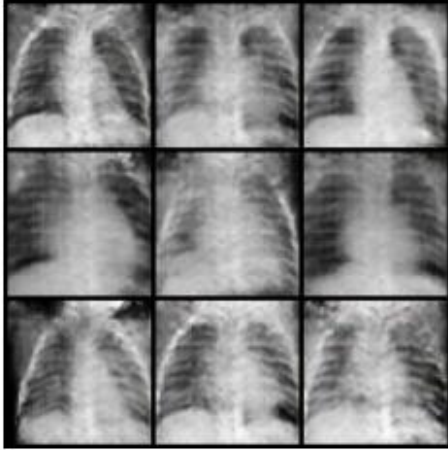
Epoch: 9/100 Batches: 0/41 Loss\_D: 1.0724 Loss\_G: 3.2590 D(x): 0.4525 D(G(z)):  
0.0167 / 0.1196  
Epoch: 10/100 Batches: 0/41 Loss\_D: 0.3465 Loss\_G: 4.8959 D(x): 0.8183 D(G(z)):  
0.0855 / 0.0192



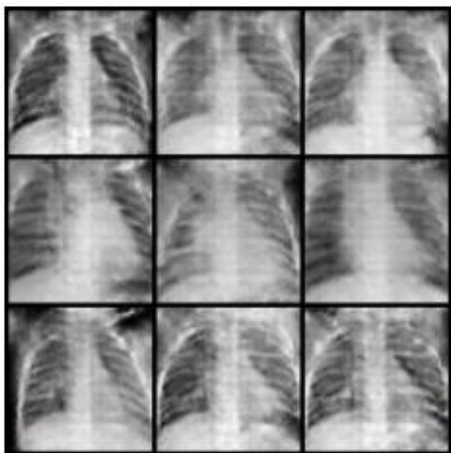
Epoch: 11/100 Batches: 0/41 Loss\_D: 0.3994 Loss\_G: 3.8819 D(x): 0.7852 D(G(z)):  
0.0996 / 0.0461  
Epoch: 12/100 Batches: 0/41 Loss\_D: 0.5003 Loss\_G: 6.1120 D(x): 0.8837 D(G(z)):  
0.2773 / 0.0095  
Epoch: 13/100 Batches: 0/41 Loss\_D: 0.9394 Loss\_G: 8.8413 D(x): 0.9589 D(G(z)):  
0.5382 / 0.0035  
Epoch: 14/100 Batches: 0/41 Loss\_D: 0.5378 Loss\_G: 5.6782 D(x): 0.9091 D(G(z)):  
0.3216 / 0.0141  
Epoch: 15/100 Batches: 0/41 Loss\_D: 1.1694 Loss\_G: 1.9285 D(x): 0.4388 D(G(z)):  
0.0275 / 0.1963  
Epoch: 16/100 Batches: 0/41 Loss\_D: 0.3184 Loss\_G: 3.8789 D(x): 0.9070 D(G(z)):  
0.1698 / 0.0420  
Epoch: 17/100 Batches: 0/41 Loss\_D: 0.3399 Loss\_G: 4.6745 D(x): 0.8992 D(G(z)):  
0.1671 / 0.0388  
Epoch: 18/100 Batches: 0/41 Loss\_D: 1.0132 Loss\_G: 5.3549 D(x): 0.4455 D(G(z)):  
0.0112 / 0.0689  
Epoch: 19/100 Batches: 0/41 Loss\_D: 0.2892 Loss\_G: 4.2982 D(x): 0.8872 D(G(z)):  
0.1316 / 0.0242  
Epoch: 20/100 Batches: 0/41 Loss\_D: 0.3916 Loss\_G: 5.1620 D(x): 0.8942 D(G(z)):  
0.2130 / 0.0135



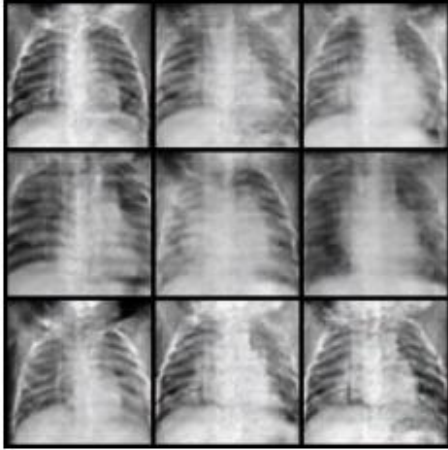
Epoch: 21/100 Batches: 0/41 Loss\_D: 0.4804 Loss\_G: 4.9852 D(x): 0.8237 D(G(z)):  
0.1964 / 0.0366  
Epoch: 22/100 Batches: 0/41 Loss\_D: 0.6222 Loss\_G: 6.4758 D(x): 0.9047 D(G(z)):  
0.3392 / 0.0113  
Epoch: 23/100 Batches: 0/41 Loss\_D: 0.3871 Loss\_G: 4.5977 D(x): 0.8427 D(G(z)):  
0.1504 / 0.0237  
Epoch: 24/100 Batches: 0/41 Loss\_D: 0.5972 Loss\_G: 2.3590 D(x): 0.6530 D(G(z)):  
0.0494 / 0.1515  
Epoch: 25/100 Batches: 0/41 Loss\_D: 0.3761 Loss\_G: 5.7796 D(x): 0.9359 D(G(z)):  
0.2295 / 0.0097  
Epoch: 26/100 Batches: 0/41 Loss\_D: 0.8485 Loss\_G: 9.2337 D(x): 0.9603 D(G(z)):  
0.5004 / 0.0009  
Epoch: 27/100 Batches: 0/41 Loss\_D: 0.5286 Loss\_G: 7.2651 D(x): 0.9009 D(G(z)):  
0.2927 / 0.0029  
Epoch: 28/100 Batches: 0/41 Loss\_D: 1.0127 Loss\_G: 5.8978 D(x): 0.4810 D(G(z)):  
0.0081 / 0.0269  
Epoch: 29/100 Batches: 0/41 Loss\_D: 0.3588 Loss\_G: 4.8103 D(x): 0.7695 D(G(z)):  
0.0348 / 0.0205  
Epoch: 30/100 Batches: 0/41 Loss\_D: 0.4935 Loss\_G: 3.7121 D(x): 0.6907 D(G(z)):  
0.0178 / 0.0683



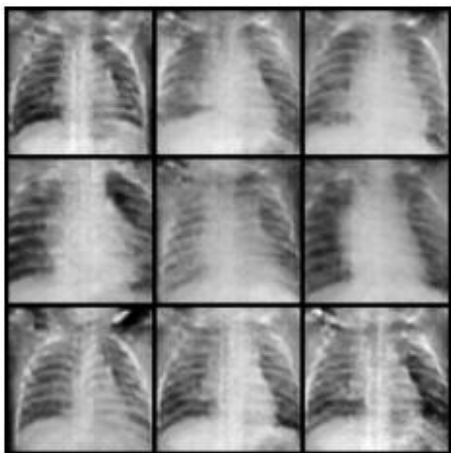
Epoch: 31/100 Batches: 0/41 Loss\_D: 0.3723 Loss\_G: 7.8815 D(x): 0.9102 D(G(z)):  
0.1980 / 0.0016  
Epoch: 32/100 Batches: 0/41 Loss\_D: 0.4455 Loss\_G: 7.2354 D(x): 0.9143 D(G(z)):  
0.2610 / 0.0026  
Epoch: 33/100 Batches: 0/41 Loss\_D: 0.4950 Loss\_G: 5.7993 D(x): 0.8638 D(G(z)):  
0.2515 / 0.0146  
Epoch: 34/100 Batches: 0/41 Loss\_D: 0.3703 Loss\_G: 5.9850 D(x): 0.9680 D(G(z)):  
0.2568 / 0.0040  
Epoch: 35/100 Batches: 0/41 Loss\_D: 0.3604 Loss\_G: 4.7984 D(x): 0.7650 D(G(z)):  
0.0211 / 0.0251  
Epoch: 36/100 Batches: 0/41 Loss\_D: 0.3342 Loss\_G: 3.5419 D(x): 0.7955 D(G(z)):  
0.0361 / 0.0443  
Epoch: 37/100 Batches: 0/41 Loss\_D: 0.4370 Loss\_G: 8.4410 D(x): 0.9621 D(G(z)):  
0.2948 / 0.0006  
Epoch: 38/100 Batches: 0/41 Loss\_D: 0.5716 Loss\_G: 8.4374 D(x): 0.9369 D(G(z)):  
0.3563 / 0.0016  
Epoch: 39/100 Batches: 0/41 Loss\_D: 0.5009 Loss\_G: 5.0394 D(x): 0.7556 D(G(z)):  
0.0688 / 0.0261  
Epoch: 40/100 Batches: 0/41 Loss\_D: 0.9008 Loss\_G: 10.2842 D(x): 0.9721 D(G(z)):  
0.5334 / 0.0003



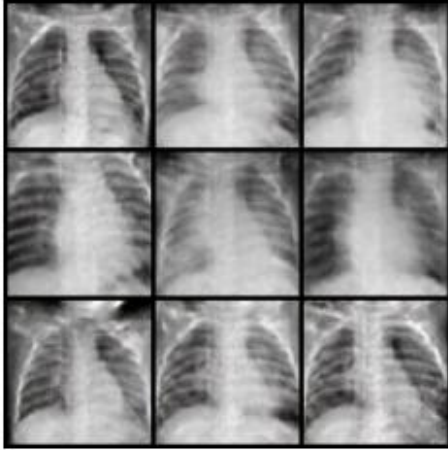
Epoch: 41/100 Batches: 0/41 Loss\_D: 0.3612 Loss\_G: 4.8958 D(x): 0.8714 D(G(z)):  
0.1649 / 0.0161  
Epoch: 42/100 Batches: 0/41 Loss\_D: 0.9022 Loss\_G: 9.4897 D(x): 0.9477 D(G(z)):  
0.4736 / 0.0015  
Epoch: 43/100 Batches: 0/41 Loss\_D: 0.4238 Loss\_G: 7.5241 D(x): 0.9358 D(G(z)):  
0.2733 / 0.0038  
Epoch: 44/100 Batches: 0/41 Loss\_D: 0.3078 Loss\_G: 3.6827 D(x): 0.8144 D(G(z)):  
0.0554 / 0.0499  
Epoch: 45/100 Batches: 0/41 Loss\_D: 0.8525 Loss\_G: 3.3433 D(x): 0.5673 D(G(z)):  
0.0250 / 0.1141  
Epoch: 46/100 Batches: 0/41 Loss\_D: 0.2602 Loss\_G: 5.0318 D(x): 0.9174 D(G(z)):  
0.1378 / 0.0157  
Epoch: 47/100 Batches: 0/41 Loss\_D: 0.6738 Loss\_G: 5.6179 D(x): 0.6142 D(G(z)):  
0.0045 / 0.0214  
Epoch: 48/100 Batches: 0/41 Loss\_D: 0.6038 Loss\_G: 8.0198 D(x): 0.9345 D(G(z)):  
0.3570 / 0.0020  
Epoch: 49/100 Batches: 0/41 Loss\_D: 0.1763 Loss\_G: 3.8473 D(x): 0.9305 D(G(z)):  
0.0901 / 0.0400  
Epoch: 50/100 Batches: 0/41 Loss\_D: 0.7195 Loss\_G: 9.2358 D(x): 0.9761 D(G(z)):  
0.4314 / 0.0013



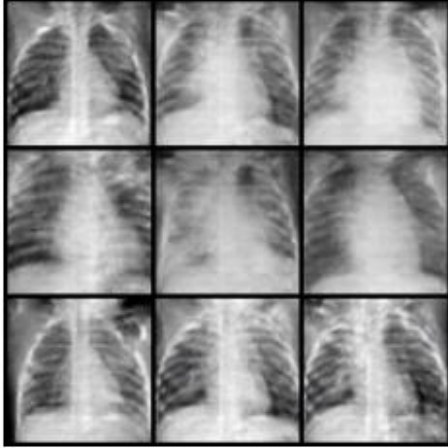
Epoch: 51/100 Batches: 0/41 Loss\_D: 0.5445 Loss\_G: 9.1247 D(x): 0.9759 D(G(z)):  
0.3498 / 0.0006  
Epoch: 52/100 Batches: 0/41 Loss\_D: 0.1573 Loss\_G: 5.0348 D(x): 0.9131 D(G(z)):  
0.0491 / 0.0144  
Epoch: 53/100 Batches: 0/41 Loss\_D: 0.3477 Loss\_G: 3.4077 D(x): 0.7721 D(G(z)):  
0.0376 / 0.0963  
Epoch: 54/100 Batches: 0/41 Loss\_D: 0.2288 Loss\_G: 5.1131 D(x): 0.9834 D(G(z)):  
0.1695 / 0.0200  
Epoch: 55/100 Batches: 0/41 Loss\_D: 0.3416 Loss\_G: 3.8026 D(x): 0.8111 D(G(z)):  
0.0642 / 0.0457  
Epoch: 56/100 Batches: 0/41 Loss\_D: 0.2704 Loss\_G: 4.1919 D(x): 0.8339 D(G(z)):  
0.0478 / 0.0302  
Epoch: 57/100 Batches: 0/41 Loss\_D: 0.1978 Loss\_G: 5.2274 D(x): 0.9327 D(G(z)):  
0.1046 / 0.0117  
Epoch: 58/100 Batches: 0/41 Loss\_D: 0.2569 Loss\_G: 4.1626 D(x): 0.8894 D(G(z)):  
0.0955 / 0.0260  
Epoch: 59/100 Batches: 0/41 Loss\_D: 0.6322 Loss\_G: 4.5040 D(x): 0.6793 D(G(z)):  
0.0298 / 0.0586  
Epoch: 60/100 Batches: 0/41 Loss\_D: 0.3238 Loss\_G: 6.1864 D(x): 0.9101 D(G(z)):  
0.1672 / 0.0093



Epoch: 61/100 Batches: 0/41 Loss\_D: 0.1893 Loss\_G: 4.6082 D(x): 0.9659 D(G(z)):  
0.1296 / 0.0169  
Epoch: 62/100 Batches: 0/41 Loss\_D: 0.3223 Loss\_G: 3.9225 D(x): 0.8018 D(G(z)):  
0.0268 / 0.0386  
Epoch: 63/100 Batches: 0/41 Loss\_D: 0.4898 Loss\_G: 4.6077 D(x): 0.7793 D(G(z)):  
0.1070 / 0.0333  
Epoch: 64/100 Batches: 0/41 Loss\_D: 0.5247 Loss\_G: 8.6471 D(x): 0.9828 D(G(z)):  
0.3367 / 0.0006  
Epoch: 65/100 Batches: 0/41 Loss\_D: 0.5674 Loss\_G: 3.0294 D(x): 0.6670 D(G(z)):  
0.0192 / 0.0917  
Epoch: 66/100 Batches: 0/41 Loss\_D: 0.1109 Loss\_G: 5.2579 D(x): 0.9402 D(G(z)):  
0.0420 / 0.0111  
Epoch: 67/100 Batches: 0/41 Loss\_D: 0.3597 Loss\_G: 4.1041 D(x): 0.8289 D(G(z)):  
0.0904 / 0.0353  
Epoch: 68/100 Batches: 0/41 Loss\_D: 0.3779 Loss\_G: 6.6307 D(x): 0.9670 D(G(z)):  
0.2611 / 0.0045  
Epoch: 69/100 Batches: 0/41 Loss\_D: 0.2519 Loss\_G: 4.6626 D(x): 0.9475 D(G(z)):  
0.1457 / 0.0208  
Epoch: 70/100 Batches: 0/41 Loss\_D: 0.7155 Loss\_G: 8.1055 D(x): 0.9391 D(G(z)):  
0.3623 / 0.0079

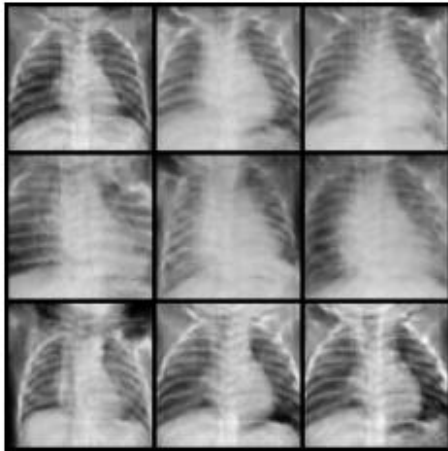


Epoch: 71/100 Batches: 0/41 Loss\_D: 0.3057 Loss\_G: 7.3229 D(x): 0.9739 D(G(z)):  
0.2224 / 0.0039  
Epoch: 72/100 Batches: 0/41 Loss\_D: 0.2175 Loss\_G: 3.0961 D(x): 0.8964 D(G(z)):  
0.0785 / 0.0646  
Epoch: 73/100 Batches: 0/41 Loss\_D: 0.1757 Loss\_G: 4.4642 D(x): 0.8944 D(G(z)):  
0.0442 / 0.0219  
Epoch: 74/100 Batches: 0/41 Loss\_D: 0.2879 Loss\_G: 7.0595 D(x): 0.9705 D(G(z)):  
0.2094 / 0.0015  
Epoch: 75/100 Batches: 0/41 Loss\_D: 0.1941 Loss\_G: 4.6531 D(x): 0.9225 D(G(z)):  
0.0963 / 0.0156  
Epoch: 76/100 Batches: 0/41 Loss\_D: 0.0901 Loss\_G: 5.0488 D(x): 0.9609 D(G(z)):  
0.0412 / 0.0393  
Epoch: 77/100 Batches: 0/41 Loss\_D: 0.2513 Loss\_G: 6.4773 D(x): 0.9900 D(G(z)):  
0.1872 / 0.0064  
Epoch: 78/100 Batches: 0/41 Loss\_D: 0.1867 Loss\_G: 5.1808 D(x): 0.8977 D(G(z)):  
0.0593 / 0.0088  
Epoch: 79/100 Batches: 0/41 Loss\_D: 0.2551 Loss\_G: 3.8574 D(x): 0.8744 D(G(z)):  
0.0873 / 0.0327  
Epoch: 80/100 Batches: 0/41 Loss\_D: 0.1611 Loss\_G: 4.9111 D(x): 0.9538 D(G(z)):  
0.0980 / 0.0126



Epoch: 81/100 Batches: 0/41 Loss\_D: 0.3120 Loss\_G: 3.5961 D(x): 0.7946 D(G(z)):  
0.0252 / 0.0686  
Epoch: 82/100 Batches: 0/41 Loss\_D: 0.2934 Loss\_G: 3.2050 D(x): 0.8448 D(G(z)):  
0.0759 / 0.0756  
Epoch: 83/100 Batches: 0/41 Loss\_D: 0.1811 Loss\_G: 6.7747 D(x): 0.9881 D(G(z)):  
0.1278 / 0.0039  
Epoch: 84/100 Batches: 0/41 Loss\_D: 0.3295 Loss\_G: 6.1604 D(x): 0.9066 D(G(z)):  
0.1780 / 0.0048  
Epoch: 85/100 Batches: 0/41 Loss\_D: 0.1892 Loss\_G: 4.7336 D(x): 0.9321 D(G(z)):  
0.1005 / 0.0137  
Epoch: 86/100 Batches: 0/41 Loss\_D: 0.2133 Loss\_G: 4.8942 D(x): 0.9658 D(G(z)):  
0.1462 / 0.0108  
Epoch: 87/100 Batches: 0/41 Loss\_D: 0.1428 Loss\_G: 4.9418 D(x): 0.9164 D(G(z)):  
0.0417 / 0.0133  
Epoch: 88/100 Batches: 0/41 Loss\_D: 0.0908 Loss\_G: 5.1108 D(x): 0.9599 D(G(z)):  
0.0443 / 0.0150  
Epoch: 89/100 Batches: 0/41 Loss\_D: 0.2526 Loss\_G: 4.2105 D(x): 0.8353 D(G(z)):  
0.0174 / 0.0387  
Epoch: 90/100 Batches: 0/41 Loss\_D: 0.1421 Loss\_G: 6.2948 D(x): 0.9303 D(G(z)):  
0.0514 / 0.0063





Epoch: 91/100 Batches: 0/41 Loss\_D: 0.0877 Loss\_G: 4.0303 D(x): 0.9421 D(G(z)): 0.0221 / 0.0272  
 Epoch: 92/100 Batches: 0/41 Loss\_D: 0.8601 Loss\_G: 12.0379 D(x): 0.9916 D(G(z)): 0.4066 / 0.0026  
 Epoch: 93/100 Batches: 0/41 Loss\_D: 0.1288 Loss\_G: 5.1381 D(x): 0.9594 D(G(z)): 0.0768 / 0.0096  
 Epoch: 94/100 Batches: 0/41 Loss\_D: 0.2618 Loss\_G: 6.5630 D(x): 0.9281 D(G(z)): 0.1400 / 0.0064  
 Epoch: 95/100 Batches: 0/41 Loss\_D: 0.1283 Loss\_G: 5.5483 D(x): 0.9234 D(G(z)): 0.0373 / 0.0083  
 Epoch: 96/100 Batches: 0/41 Loss\_D: 0.1388 Loss\_G: 4.8562 D(x): 0.9324 D(G(z)): 0.0574 / 0.0167  
 Epoch: 97/100 Batches: 0/41 Loss\_D: 0.2580 Loss\_G: 4.6301 D(x): 0.9064 D(G(z)): 0.1129 / 0.0206  
 Epoch: 98/100 Batches: 0/41 Loss\_D: 0.3799 Loss\_G: 8.8145 D(x): 0.9363 D(G(z)): 0.2030 / 0.0039  
 Epoch: 99/100 Batches: 0/41 Loss\_D: 0.5083 Loss\_G: 3.1941 D(x): 0.7248 D(G(z)): 0.0314 / 0.1010

### **Mari lihat grafik dari losses**

Kita dapat melihat bahwa loss Generator dimulai sangat tinggi, karena diinisialisasi dengan noise acak. Sebaliknya, loss Discriminator relatif rendah karena gambar-gambar yang digunakan adalah noise acak dan jelas palsu. Seiring Generator semakin membaik, loss-nya menurun karena semakin sulit bagi Discriminator untuk membedakan antara gambar nyata dan palsu (dan sebaliknya, loss Discriminator meningkat). Selama pelatihan, Generator dan Discriminator melanjutkan proses adversarial ini, dan kita melihat bahwa losses akhirnya stabil dan berfluktuasi saat salah satu model membaik.

Idealnya, kita ingin model mencapai keseimbangan di mana Generator menghasilkan gambar berkualitas tinggi yang menantang bagi Discriminator, menghasilkan loss Generator dan Discriminator yang rendah.

```

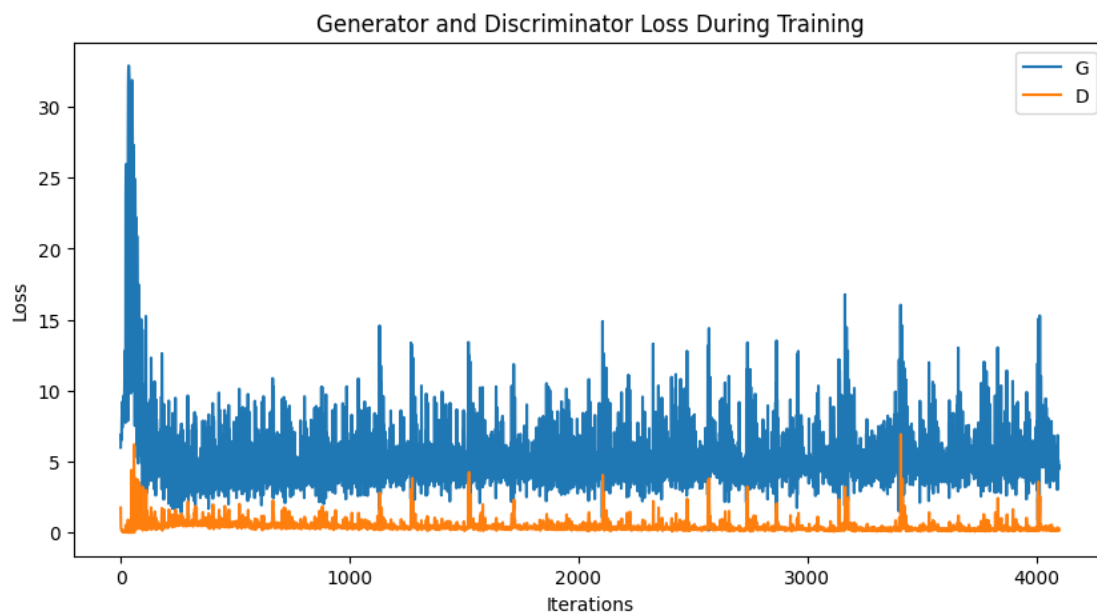
from IPython.display import HTML
import matplotlib.animation as animation
import matplotlib.pyplot as plt

```

```

plt.figure(figsize=(10,5))
plt.title("Generator and Discriminator Loss During Training")
plt.plot(G_losses,label="G")
plt.plot(D_losses,label="D")
plt.xlabel("Iterations")
plt.ylabel("Loss")
plt.legend()
plt.show()

```



**Kita akan membuat animasi untuk memvisualisasikan pelatihan.**

*# Meningkatkan batas default yang ditetapkan oleh Matplotlib untuk menyematkan animasi*

```

import matplotlib
matplotlib.rcParams['animation.embed_limit'] = 100 # 100 MB

```

*# Membuat figure untuk animasi dengan ukuran (8x8)*

```

fig = plt.figure(figsize=(8, 8))

```

*# Menghilangkan sumbu pada plot*

```

plt.axis("off")

```

*# Mengatur tata letak plot untuk memberi ruang antar elemen gambar*

```

plt.tight_layout(pad=2.0, h_pad=2.0, w_pad=2.0)

```

*# Judul kosong dan gambar pertama untuk animasi*

```

title = plt.title("")

```

```
img_plot = plt.imshow(np.transpose(img_list[0], (1, 2, 0)), animated=True)
```

```
# Fungsi untuk memperbarui gambar dan judul setiap frame
```

```
def update_title_and_image(frame):
```

```
    # Memperbarui gambar pada frame yang sesuai
```

```
    img_plot.set_array(np.transpose(img_list[frame], (1, 2, 0)))
```

```
    # Memperbarui teks judul untuk menunjukkan epoch yang sedang ditampilkan
```

```
    title.set_text(f"Epoch {frame}/{num_epochs}")
```

```
# Membuat animasi dengan fungsi update, jumlah frame sebanyak jumlah gambar yang disimpan, dan interval waktu 100ms
```

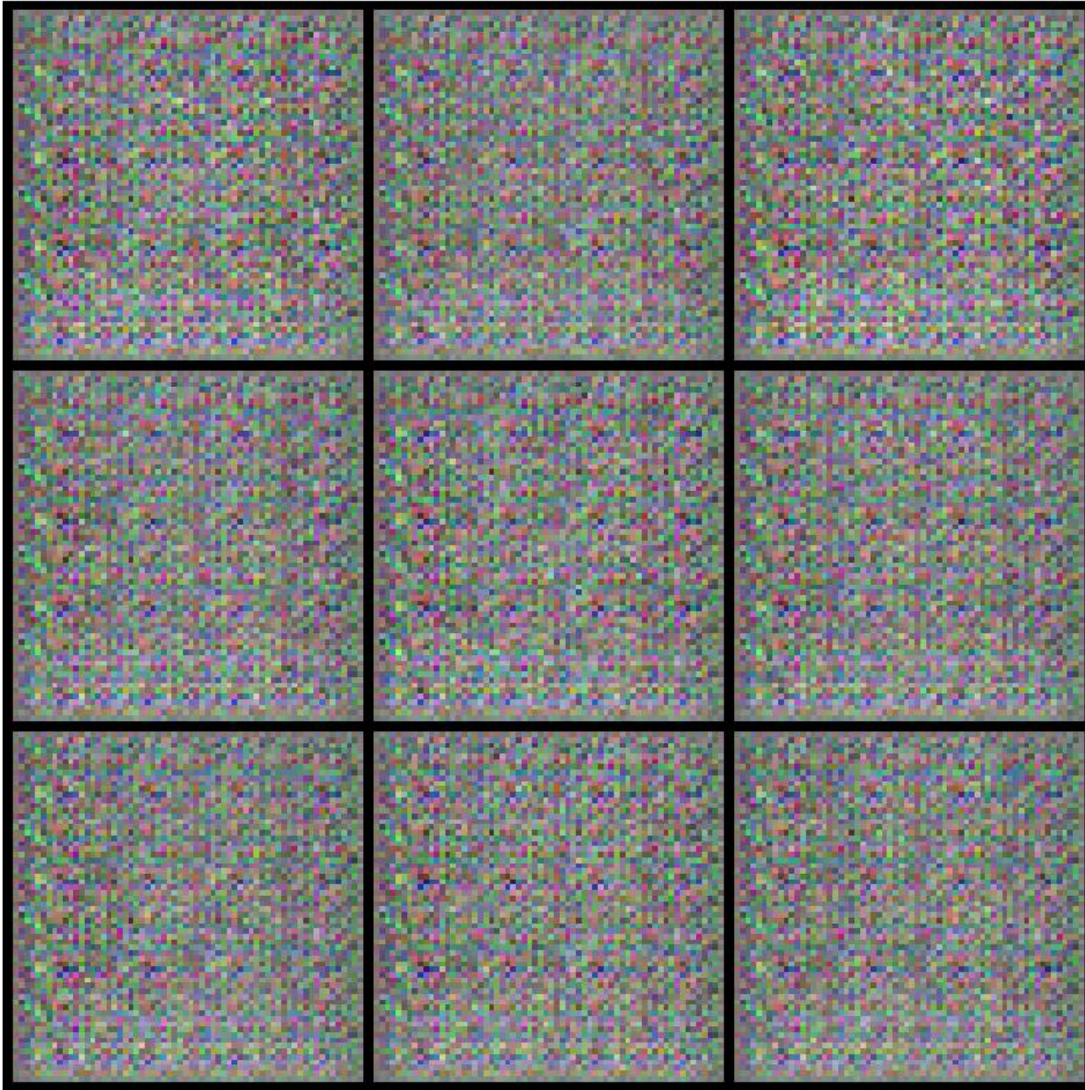
```
ani = animation.FuncAnimation(fig, update_title_and_image, frames=len(img_list),  
interval=100, repeat_delay=5000)
```

```
# Menampilkan animasi dalam format HTML
```

```
HTML(ani.to_jshtml())
```

```
<IPython.core.display.HTML object>
```

Epoch 0/100



```
# Menyimpan animasi sebagai file GIF di folder 'images'  
ani.save('images/dcgan_training_animation.gif', writer='pillow')
```

**Mari gunakan model kita untuk membuat beberapa gambar paru-paru sintetis.**

```
# Menentukan path atau lokasi penyimpanan untuk model yang sudah dilatih  
path_checkpoint = "models/chest_final_epoch_checkpoint.pkl"
```

```
# Jumlah gambar yang diinginkan  
num_img = 16  
# Menghitung jumlah baris dalam grid gambar  
nb_row = math.ceil(math.sqrt(num_img))
```

```
# Membuat noise acak untuk input generator  
random_noise = torch.randn(num_img, nz, 1, 1, device=device)
```

```

# Menginstansiasi generator baru
new_gen= Generator(nb_gpu).to(device)

# Menangani multi-GPU jika diperlukan
if (device.type == 'cuda') and (nb_gpu > 1):
    new_gen = nn.DataParallel(new_gen, list(range(nb_gpu)))

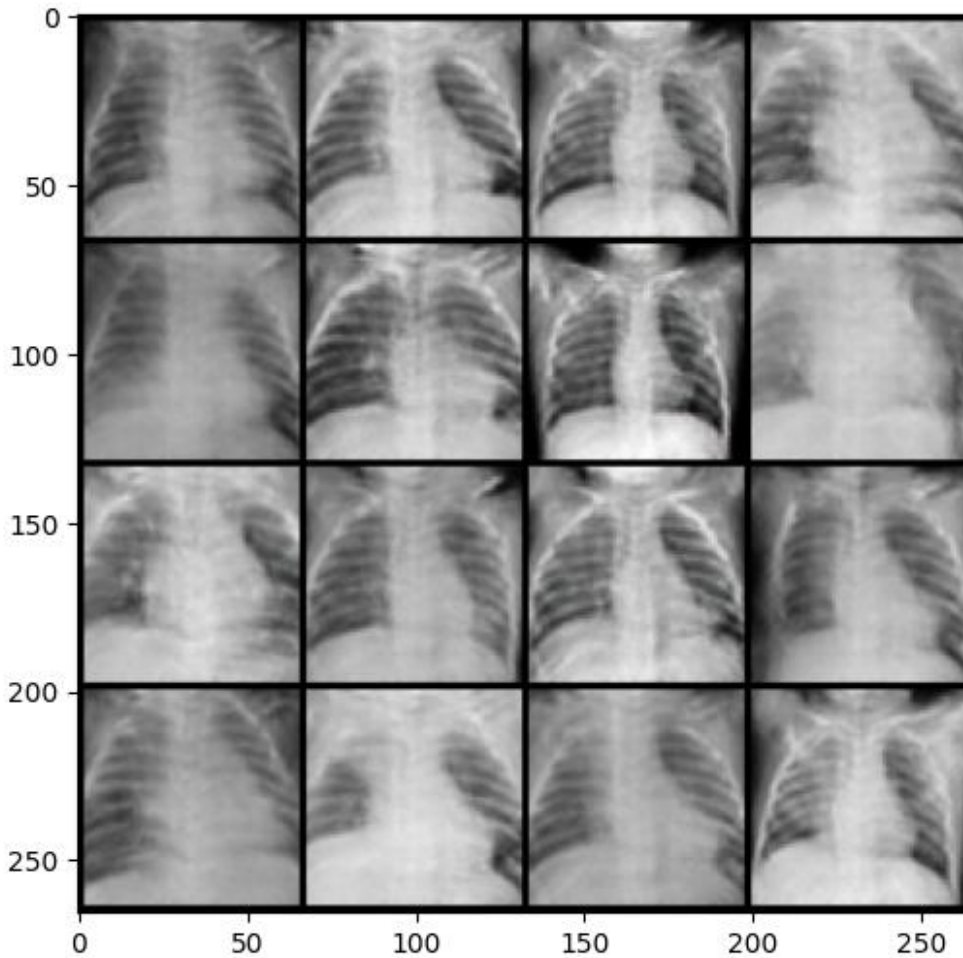
# Memuat bobot model dari path checkpoint yang telah disimpan
checkpoint = torch.load(path_checkpoint, map_location="cpu")
state_dict_g = checkpoint["g_model_state_dict"]
new_gen.load_state_dict(state_dict_g) # Memuat bobot generator ke model baru

# Menghasilkan gambar dari noise acak
with torch.no_grad(): # Menghentikan gradien untuk proses inferensi
    fake_data = new_gen(random_noise).detach().cpu() # Mendapatkan hasil output generator

# Membuat grid gambar yang dihasilkan
img_grid = vutils.make_grid(fake_data, padding=2, normalize=True, nrow=nb_row).cpu()
img_grid = np.transpose(img_grid, (1, 2, 0)) # Mengubah urutan channel gambar untuk
penampilan

# Menampilkan gambar
plt.figure(figsize=(6, 6))
plt.imshow(img_grid) # Menampilkan grid gambar
plt.show() # Menampilkan plot

```



Berikutnya, kita akan membuat Discriminator baru dari model yang kita simpan.

```
# Menginstansiasi discriminator baru
new_dis = Discriminator(nb_gpu).to(device)

# Menangani multi-GPU jika diperlukan
if (device.type == 'cuda') and (nb_gpu > 1):
    new_dis = nn.DataParallel(new_dis, list(range(nb_gpu))) # Menangani distribusi model ke
    beberapa GPU

# Memuat bobot model dari path checkpoint yang telah disimpan
checkpoint = torch.load(path_checkpoint, map_location="cpu")
state_dict_d = checkpoint["d_model_state_dict"] # Mendapatkan state dictionary dari
discriminator
new_dis.load_state_dict(state_dict_d) # Memuat bobot discriminator ke model baru

<All keys matched successfully>
```

Kami akan menggunakan generator baru kami untuk membuat beberapa gambar paru-paru sintetis. Diskriminator baru akan memberi kami probabilitas bahwa gambar paru-paru sintetis itu

nyata. Jika probabilitasnya lebih besar dari ambang batas (dalam kasus kami 70%), kami akan menambahkannya ke serangkaian gambar yang "baik".

Kami dapat melihat bahwa beberapa gambar yang "baik" itu buram. Ini menyoroti mengapa sangat penting untuk meminta ahli radiologi mengevaluasi gambar paru-paru sintetis!

```
# Menentukan threshold untuk klasifikasi gambar yang diterima oleh discriminator  
threshold = 0.70
```

```
# Membuat list untuk menyimpan gambar-gambar yang memiliki probabilitas >= threshold  
pass_threshold = []
```

```
# Loop untuk menghasilkan gambar sampai list pass_threshold memiliki 64 gambar  
while len(pass_threshold) < 64:
```

```
    # Membuat noise acak untuk input generator  
    random_noise = torch.randn(num_img, nz, 1, 1, device=device)
```

```
    # Menghasilkan gambar dengan noise yang telah dibuat  
    with torch.no_grad(): # Tidak memerlukan gradient untuk operasi inferensi  
        fake_data = new_gen(random_noise)
```

```
    # Menyetel discriminator ke mode evaluasi (tidak ada update parameter)  
    new_dis.eval()
```

```
    # Melalui discriminator untuk mendapatkan probabilitas real/fake untuk gambar yang dihasilkan  
    output_probabilities = new_dis(fake_data)
```

```
    # Membuat mask (Jika probabilitas < threshold, set ke False, jika >= threshold set True)  
    good_image_mask = output_probabilities >= threshold
```

```
    # Menghapus dimensi tambahan dari tensor (torch.Size([16, 1, 1, 1]) menjadi torch.Size([16]))  
    good_image_mask = good_image_mask.squeeze()
```

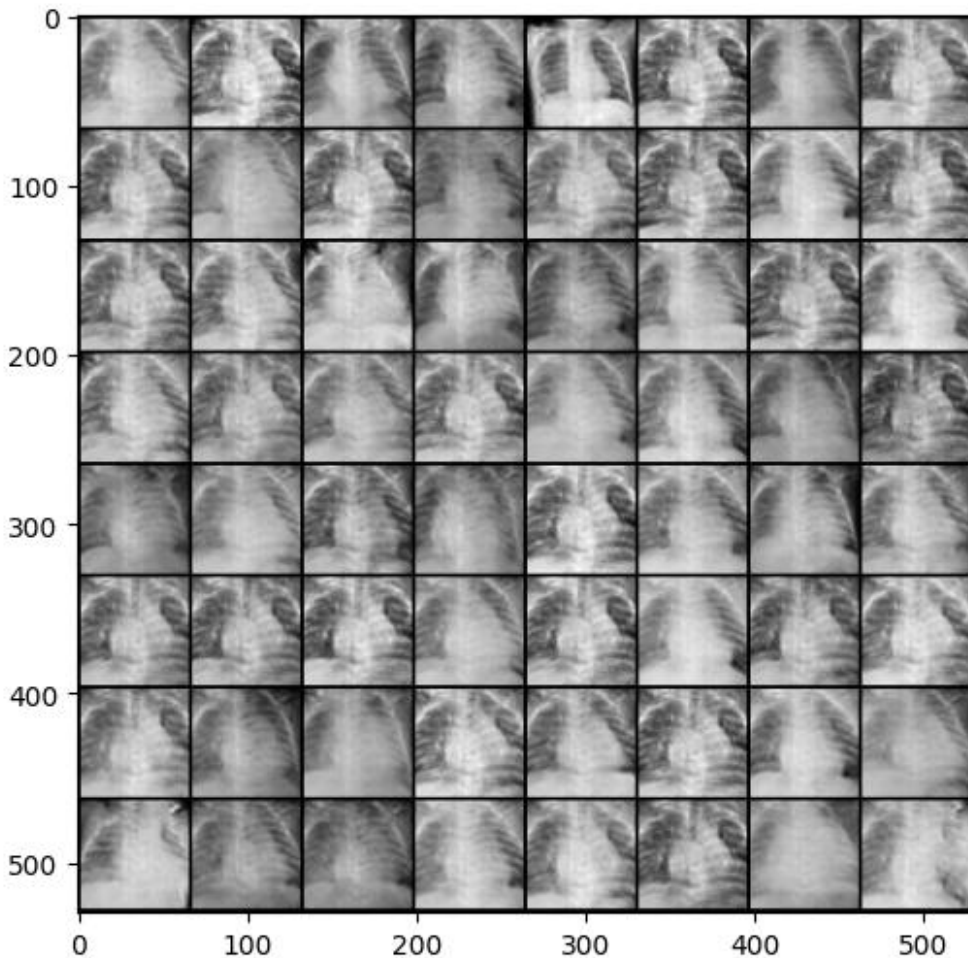
```
    # Menyaring gambar yang dipilih berdasarkan mask yang dihasilkan  
    good_images = fake_data[good_image_mask > threshold]
```

```
    # Jika good_images tidak kosong  
    if good_images.numel() > 0:  
        # Loop melalui tensor pada dimensi pertama (dimensi gambar)  
        for i in range(good_images.size(0)):  
            # Menambahkan gambar yang disaring ke list pass_threshold  
            pass_threshold.append(good_images[i])
```

```
# Menampilkan gambar-gambar yang telah disaring dan memenuhi threshold  
img_grid = vutils.make_grid(pass_threshold, padding=2, normalize=True, nrow=8).cpu()
```



```
img_grid = np.transpose(img_grid, (1, 2, 0))
plt.figure(figsize=(6, 6))
plt.imshow(img_grid)
plt.show()
```



### Kutipan:

Kermany, Daniel; Zhang, Kang; Goldbaum, Michael (2018), “Labeled Optical Coherence Tomography (OCT) and Chest X-Ray Images for Classification”, Mendeley Data, V2, doi: 10.17632/rscbjbr9sj.2

### Membuat Dataset Sintetis dengan sdxl-turbo

- Notebook ini mendemonstrasikan bagaimana cara menghasilkan dataset sintetis untuk tujuan artistik menggunakan model `text2img stabilityai/sdxl-turbo`. Model ini diperkenalkan dalam makalah Adversarial Diffusion Distillation oleh Sauer, dkk. Model



ini adalah model difusor hibrida yang menambahkan kehilangan adversarial yang ditemukan dalam GAN untuk mempercepat proses pembuatan gambar.

- Kita akan menghasilkan gambar fractal flame. Sebuah fractal adalah pola yang berulang dan tampak sama tidak peduli seberapa banyak Anda memperbesar atau memperkecilnya. Bagian "flames" dalam fractal flames adalah nama keren untuk pola dinamis dan berwarna-warni yang bisa Anda buat dengan jalur rekursif yang digunakan untuk merender gambar.
- Bobot model diunduh dari Hugging Face Hub. Notebook ini didasarkan pada SDXL-Turbo Model Card. Anda juga bisa mempelajari lebih lanjut tentang penggunaan model ini di sini.

### Catatan Penting:

- Sebuah watermark tak terlihat disematkan pada gambar agar dapat diidentifikasi sebagai gambar yang dihasilkan oleh AI.
- Model SDXL Turbo dilisensikan di bawah Lisensi Stability AI Non-Commercial Research Community, Hak Cipta (c) Stability AI Ltd. Semua Hak Dilindungi Undang-Undang.
- Penggunaan di luar cakupan: representasi faktual atau nyata dari orang atau peristiwa.
- Notebook ini dijalankan dengan T4 High-RAM.

### Pertama, mari instal paket yang diperlukan.

```
!pip install -q diffusers transformers accelerate omegaconf torchvision imagehash  
huggingface_hub datasets
```

---

```
— 1.8/1.8 MB 10.4 MB/s eta 0:00:00
```

---

```
— 265.7/265.7 kB 11.9 MB/s eta 0:00:00
```

---

```
— 79.5/79.5 kB 8.2 MB/s eta 0:00:00
```

---

```
— 296.5/296.5 kB 11.2 MB/s eta 0:00:00
```

---

```
— 507.1/507.1 kB 16.4 MB/s eta 0:00:00
```

---

```
— 117.0/117.0 kB 15.0 MB/s eta 0:00:00
```

```
etadata (setup.py) ...
```

```
— 115.3/115.3 kB 12.0 MB/s eta 0:00:00
```

---

```
— 134.8/134.8 kB 17.6 MB/s eta 0:00:00
```

```
e (setup.py) ...
```

**Mari bergabung tanpa token akses WRITE.**

**from** huggingface\_hub **import** notebook\_login

notebook\_login()

{"model\_id":"2bdbaf03d16b4c1caeea0a8942fcd63b","version\_major":2,"version\_minor":0}

*# Mengimpor pustaka os untuk menangani operasi sistem file*

**import** os

*# Menentukan direktori untuk gambar yang dihasilkan*

generated\_images\_dir = "/content/fractal\_flame"

*# Membuat direktori baru jika belum ada*

os.makedirs(generated\_images\_dir, exist\_ok=True)

Ada beberapa variasi pada nyala api fraktal yang dikaitkan dengan berbagai algoritma. Buku catatan ini menghasilkan variasi "linier". Latihan yang menyenangkan adalah mencoba variasi yang berbeda atau mengubah warna.

*# Mengimpor pustaka untuk menghasilkan gambar dari teks*

**from** diffusers **import** AutoPipelineForText2Image

**import** torch

**from** PIL **import** Image, ImageShow

*# Memuat model pipeline untuk generasi gambar berbasis teks dari model pre-trained*

pipe = AutoPipelineForText2Image.from\_pretrained("stabilityai/sdxl-turbo",  
torch\_dtype=torch.float16, variant="fp16")

*# Menyambungkan pipeline ke perangkat GPU*

pipe.to("cuda")

*# Menyusun prompt yang digunakan untuk menghasilkan gambar*

*# Variasi bentuk: linear, sinusoidal, swirl, spherical, horseshoe, polar, hankerchief, heart, disc, hyperbolic, fisheye*

prompt = "fractal flame, linear, in shades of cherry red, orange, and fushia, vibrant colors, 8K"

*# Tentukan jumlah gambar yang akan dihasilkan*

num\_images\_to\_generate = 100

images = []

*# Loop untuk menghasilkan gambar*

**for** idx **in** range(num\_images\_to\_generate):

*# Menghasilkan gambar dari pipeline dengan prompt dan parameter tertentu*

image = pipe(prompt=prompt, num\_inference\_steps=1, guidance\_scale=1.0).images[0]

*# Menyimpan gambar yang dihasilkan ke dalam direktori yang sudah ditentukan*

image.save(f"fractal\_flame/image\_{idx}.png")

*# Menambahkan gambar ke dalam list images*  
images.append(image)

```
{"model_id":"d73af29bb0e6432f98d0975ef29ee2d9","version_major":2,"version_minor":0}  
{"model_id":"86fa47dd864f404f8d5a98534424b05f","version_major":2,"version_minor":0}  
{"model_id":"96ba3e021bd0418b925c832d1fcfd6b8","version_major":2,"version_minor":0}  
{"model_id":"bd181def4a2f4f9cb4993ca480c8ee3a","version_major":2,"version_minor":0}  
{"model_id":"a81a32c9d268403a996d31e6f9f572a3","version_major":2,"version_minor":0}  
{"model_id":"8d207f6bc44a4512b9ed8dfb50959f90","version_major":2,"version_minor":0}  
{"model_id":"28360d91067e4f01b4d14c64fd405590","version_major":2,"version_minor":0}  
{"model_id":"a7b1e0bdf3d7481c8e7801a4ac265e27","version_major":2,"version_minor":0}  
{"model_id":"e6f000ce82754586b1d342f56441dfee","version_major":2,"version_minor":0}  
{"model_id":"a5fe3c721fdd42e9bfed021d7d6e9050","version_major":2,"version_minor":0}  
{"model_id":"2eda8198c75c4f8fa31c9ac8ac812161","version_major":2,"version_minor":0}  
{"model_id":"fb82c0be1abc4e90a6de9b7360f920cb","version_major":2,"version_minor":0}  
{"model_id":"45a37f56184348ae969ed21173568936","version_major":2,"version_minor":0}  
{"model_id":"5ba4cb50fa4a452a8a024bc36e7449aa","version_major":2,"version_minor":0}  
{"model_id":"12702003902c4e31a758450e3c5db536","version_major":2,"version_minor":0}  
{"model_id":"4629ceca172f46ee9f853c1c267716cb","version_major":2,"version_minor":0}  
{"model_id":"ea91467ca88548a2a2d70620161a13c9","version_major":2,"version_minor":0}  
{"model_id":"d8efc471f5084b89b64245eab8d98cfd","version_major":2,"version_minor":0}  
{"model_id":"543e84d016b740b48fd2b2d2ea38e805","version_major":2,"version_minor":0}  
{"model_id":"66c24bc25d364ecb89314df8f9cb6da5","version_major":2,"version_minor":0}  
{"model_id":"a2fa4270dcab4052998dcaf0f6ce47bd","version_major":2,"version_minor":0}  
{"model_id":"719c6647931742b788f1286fa07689b8","version_major":2,"version_minor":0}  
{"model_id":"08d1348475f14030bfc152d7785617b5","version_major":2,"version_minor":0}  
{"model_id":"e35a3417fb8a42739c40278e6b0c9f14","version_major":2,"version_minor":0}  
{"model_id":"b5b0d60917a640d68e7c69c3daff1023","version_major":2,"version_minor":0}
```

{ "model\_id": "c351d770f9ca404abd54f20648d4aa9f", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "f0ffca49fa4546a6a94dd85d25f55820", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "59d13e877a8641878bdb42e2e3f49a75", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "22a742dbf1e540eebba19b795aa27813", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "3a5f3bab9812460eb0fe50a2ed66e3ff", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "f314c9b1a02646258d03c97351cbf155", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "89fc8d7fe9c64c29a201e108f1ebe4f5", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "340fea6bb7aa42669773c1b085af7831", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "980b7e01a02c4f86b19137fd2f13177a", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "49dfc51140bc4238b6ede639dbcdfcc6", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "431d0d21625c403f84cb3fd72d47bd2b", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "3350056f1b9b4ad38abf3ed2f301ee2d", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "01e44f19f35e41b7b83413afa717cfe", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "ccc8a8004bb840d3b086dc76022b4f7e", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "7cd2267cc82b436097dd199c117c1a23", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "61a122c2c55a474f8459ec2f632ceac2", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "930d093b996f4773bf6ae184f1b3ccb4", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "429df5d19fdd46eebd69a2a08e4c8599", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "0c1849a4f95041e7ac8c1bdc54d5c986", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "100508c23b8747168ddb543eeb88975b", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "1bbbca2433684d348f8184360fb3c17c", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "6082c74bbbf6485eb301145d36f7438b", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "b808b66975d34f51943725e76ffafded", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "a986f68a67e44aebabadaedae027f1d6", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "406f420e9fb740efa923577246d7a6ea", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "1f4fb2fa0c9a400594c8b966f76a485b", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "ec05318b07734d72952b1a36eaff8bdd", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "7fdf382972ec4b05b63143c089949720", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "b90c7fbb523c4f0d9da400282b33461b", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "c33fe0427c4343cca746f84dac906388", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "0345a18bbff747e3a0cd201ca829006c", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "3c415bad42f54744bcc5371944ef9d8e", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "f543f24aed664480a9d72f2d3f244947", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "72c2a888ecd646c18bc6f9dfc0fe0c0e", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "15fc685d2a2f4145851be4c8a75c61a1", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "e30ba2c6a2fe40fcabbf3a312b1bd838", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "d5a11bfbaecc491ba0d61cacff96517f", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "1fd28c7266df48beb9d1acf2dd2c700b", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "810f444f5d804e7b9771fcc5e76c570f", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "8115cc28c458459a89e99c2fe fd17dc0", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "278a116492c44f61b3206694cb5047b6", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "0d8ede164a924087923e8b3c086dc661", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "e93c529da056470b90ea4f41ca8921fe", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "57c7215691904f78808b5e89fedf32db", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "0f2a1b85fad742bbb6f69fd8e4b381bca", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "26c5cfc058ce42088eb248aa0d2b6d07", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "3a41ea3e2d854bffb6c62b8c3624fc5a", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "b0b5b4484ce4431b8f260db02c221d34", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "20d0fa393ad044b8af4e028b6f3b2ce5", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "13856fa4217a408ca543d62924c6905d", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "94af01e175d74da686a307d2d6475193", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "6d5b2d627c40492083419ef5cb0e604b", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "da9bb648f8a54bcf9c9b880df03811dc", "version\_major": 2, "version\_minor": 0 }

{ "model\_id": "49480d7cc3934224a5b5224fa5d57810", "version\_major": 2, "version\_minor": 0 }

```

{"model_id":"db057d540b0744b2a3e95fad37c6e094","version_major":2,"version_minor":0}
{"model_id":"5649d883fcc7484abfdde17eb7c4fd55","version_major":2,"version_minor":0}
{"model_id":"e2ecf09c38e244d1ab3c442bbca15570","version_major":2,"version_minor":0}
{"model_id":"06ddf116c1d04747ac374363d6d6c4a5","version_major":2,"version_minor":0}
{"model_id":"0bec22e182c74a43a6c71b1cb44d0584","version_major":2,"version_minor":0}
{"model_id":"0a3965e2f8af4d36a0242678393867eb","version_major":2,"version_minor":0}
{"model_id":"6953a5ae34a041d18877f6f83b5698f3","version_major":2,"version_minor":0}
{"model_id":"2093cefaeaa54ad1a037cf2e3fc3c50c","version_major":2,"version_minor":0}
{"model_id":"8cd5260df60b4ed8932d339a3f395a9d","version_major":2,"version_minor":0}
{"model_id":"a5cd5e28abde43af893f860755308169","version_major":2,"version_minor":0}
{"model_id":"349f3ebd8eff436d87910508852deff8","version_major":2,"version_minor":0}
{"model_id":"c3079fbd50234e45a26fb93d02513c54","version_major":2,"version_minor":0}
{"model_id":"4958e98ba89340e0921a1286a2b985de","version_major":2,"version_minor":0}
{"model_id":"6190da5eacaa48e4ad78a922ec10061b","version_major":2,"version_minor":0}
{"model_id":"091e1e9ff2a843b0b3147d6b4fb6b13a","version_major":2,"version_minor":0}
{"model_id":"c39b232274954e9a91efab0f8ca78155","version_major":2,"version_minor":0}
{"model_id":"0b107ed42e6049cab0928d9954ae3ada","version_major":2,"version_minor":0}
{"model_id":"b30c1d55ae7b4988ac0534e10244e37d","version_major":2,"version_minor":0}
{"model_id":"4e8554b636714a20b66d529f131a662b","version_major":2,"version_minor":0}
{"model_id":"5a2e3eee84b540b59f51b93bebef1b5f","version_major":2,"version_minor":0}
{"model_id":"544a6aad3fda4da0a0b062e23d2c9300","version_major":2,"version_minor":0}
{"model_id":"7281394cfb054f6a8ed840250ea275ba","version_major":2,"version_minor":0}

```

*# Mengimpor pustaka untuk operasi matematika dan plotting*

**import** math

**import** matplotlib.pyplot as plt

*# Menentukan jumlah gambar yang akan ditampilkan*

num\_images\_to\_display = 6

*# Fungsi untuk menampilkan gambar-gambar dalam satu baris*

**def** plot\_images(images, title=None):

```

# Mengatur ukuran gambar agar pas untuk menampilkan beberapa gambar dalam satu baris
plt.figure(figsize=(20, 20))

# Loop untuk menampilkan gambar-gambar
for i in range(num_images_to_display):
    # Menambahkan subplot untuk setiap gambar
    ax = plt.subplot(1, num_images_to_display, i+1)

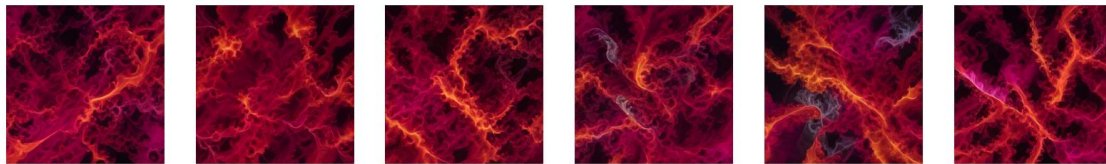
    # Menambahkan judul jika diberikan
    if title is not None:
        plt.title(title)

    # Menampilkan gambar yang ke-i
    plt.imshow(images[i])

    # Menonaktifkan tampilan sumbu pada gambar
    plt.axis("off")

# Memanggil fungsi untuk menampilkan gambar
plot_images(images)

```



## Mendorong kumpulan data ke Hugging Face Hub

Sekarang setelah kita membuat kumpulan data, mari tambahkan ke hub Hugging Face.

```
!apt-get install git-lfs
```

```
Reading package lists... Done
```

```
Building dependency tree... Done
```

```
Reading state information... Done
```

```
git-lfs is already the newest version (3.0.2-1ubuntu0.2).
```

```
0 upgraded, 0 newly installed, 0 to remove and 24 not upgraded.
```

## Muat kumpulan data kami

```
from datasets import load_dataset
```

```
dataset = load_dataset("imagefolder", data_dir="fractal_flame")
```

```
{"model_id": "0e084b247ad9438c8a694b863ee32126", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "1aa5b54b512a4b52ab34142c9b03a7a0", "version_major": 2, "version_minor": 0}
```

**Sekarang kita dapat mengirim dataset ke Hub.**

```
user_name = "kfahn"
```

```
repo_name = "fractal_flame"
```

```
dataset.push_to_hub(f"{user_name}/{repo_name}")
```

```
{"model_id":"552bdaa154534502831c985cd9097a7e","version_major":2,"version_minor":0}
```

```
{"model_id":"dba91a75b46e4dd88e20aec4830a6e78","version_major":2,"version_minor":0}
```

```
{"model_id":"b66d4c65f2e74d0e86c08aed51082fcf","version_major":2,"version_minor":0}
```

```
CommitInfo(commit_url='https://huggingface.co/datasets/kfahn/fractal_flame2/commit/b2fa3bc4d9367a11bc62045110809b29fe121346', commit_message='Upload dataset',  
commit_description='', oid='b2fa3bc4d9367a11bc62045110809b29fe121346', pr_url=None,  
pr_revision=None, pr_num=None)
```

LINK YOUTUBE : <https://youtu.be/8c5IQHF7XJs>